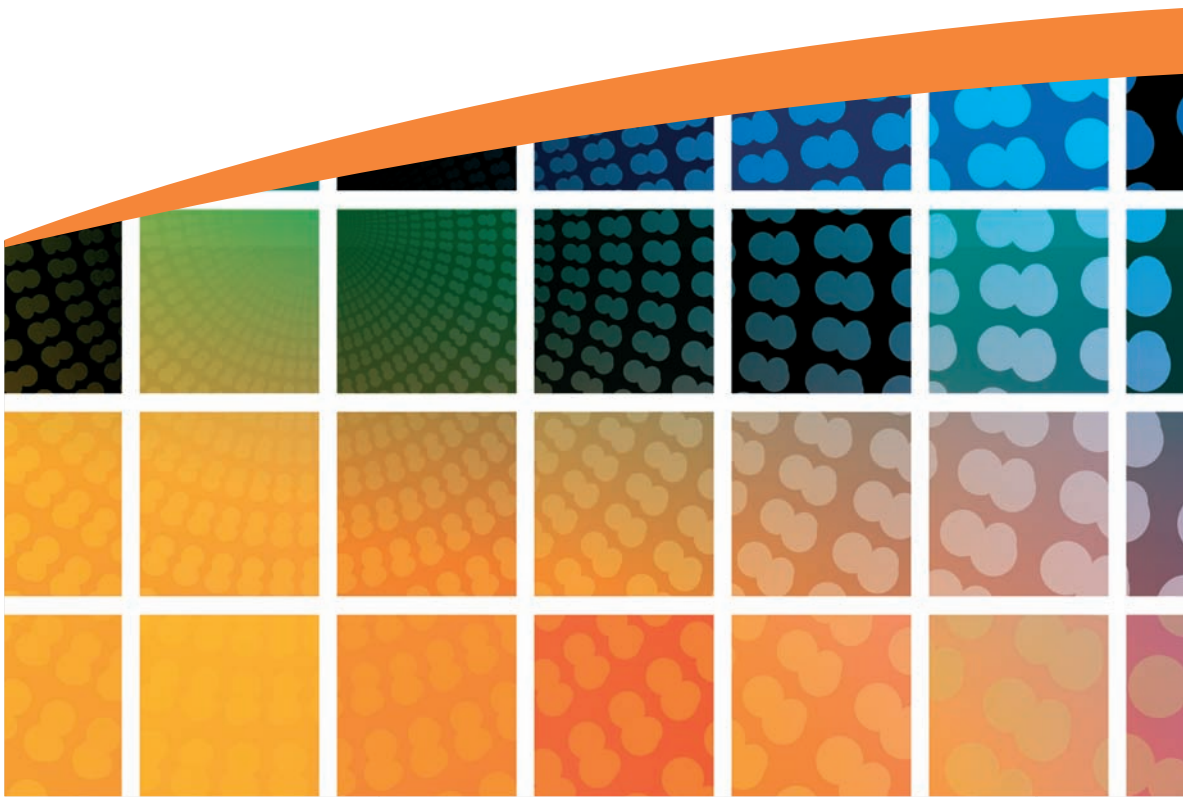
Référence

jQuery

Simplifiez et enrichissez vos développements JavaScript

J. Chaffer et K. Swedberg
Préfacé par John Resing, créateur de jQuery



Réseaux et télécom

Programmation

Développement web

Sécurité

Système d'exploitation

PEARSON

jQuery

Simplifiez et enrichissez vos développements JavaScript

Jonathan Chaffer
Karl Swedberg

Traduit par Hervé Soulard
avec la contribution technique de Didier Mouronval

PEARSON

Publié par Pearson Education France
47 *bis*, rue des Vinaigriers
75010 PARIS
Tél. : 01 72 74 90 00
www.pearson.fr

ISBN : 978-2-7440-4142-6

Titre original : *Learning jQuery 1.3*

Traduit de l'américain par Hervé Soulard

Contribution technique : Didier Mouronval

ISBN original : 978-1-847196-70-5

Édition originale publiée par Packt
www.packtpub.com

Table des matières

Avant-propos VII

Préface à l'édition française IX

À propos des auteurs XI

Préface XIII

Organisation du livre XIII
Prérequis XV
Public du livre XVI
Conventions typographiques XVI
Votre avis XVII
Télécharger le code XVII

1 Premiers pas 1

1.1 Intérêt de jQuery 2
1.2 Efficacité de jQuery 3
1.3 Historique du projet jQuery 4
1.4 Première page web avec jQuery 5
1.5 En résumé 12

2 Sélecteurs 13

2.1 Le DOM 13
2.2 La fonction *$()* 14
2.3 Sélecteurs CSS 15
2.4 Sélecteurs d'attribut 18
2.5 Sélecteurs personnalisés 20
2.6 Méthodes de parcours du DOM 24
2.7 Accéder aux éléments du DOM 27
2.8 En résumé 27

3 Événements 29

3.1 Effectuer des tâches au chargement de la page 29
3.2 Événements simples 33
3.3 Événements composés 42
3.4 Le périple d'un événement 45
3.5 Modifier le périple : l'objet *event* 47

3.6 Retirer un gestionnaire d'événements 52
3.7 Simuler une action de l'utilisateur 55
3.8 En résumé 59

4 Effets 61
4.1 Modifier les styles CSS 61
4.2 Bases du masquage/affichage 66
4.3 Effets et vitesse 68
4.4 Effets composés 69
4.5 Créer des animations personnalisées 71
4.6 Effets simultanés ou séquentiels 75
4.7 En résumé 82

5 Manipulation du DOM 83
5.1 Manipuler des attributs 83
5.2 Insérer de nouveaux éléments 87
5.3 Déplacer des éléments 89
5.4 Envelopper des éléments 97
5.5 Copier des éléments 98
5.6 Les méthodes du DOM en bref 105
5.7 En résumé 107

6 AJAX 109
6.1 Charger des données à la demande 110
6.2 Choisir le format des données 125
6.3 Passer des données au serveur 126
6.4 Surveiller la requête 132
6.5 AJAX et les événements 136
6.6 Questions de sécurité 137
6.7 Autres solutions 139
6.8 En résumé 144

7 Manipulation des tables 145
7.1 Tri et pagination 146
7.2 Modifier l'aspect de la table 173
7.3 En résumé 197

8 Manipulation des formulaires 199
8.1 Améliorer un formulaire de base 199
8.2 Formulaires compacts 222
8.3 Manipuler des données numériques 236
8.4 En résumé 257

9 Carrousels et prompteurs 259
9.1 Prompteur de nouvelles 260
9.2 Carrousel d'images 276
9.3 En résumé 301

10 Utilisation des plugins 303
10.1 Rechercher des plugins et de l'aide 303
10.2 Utiliser un plugin 304
10.3 Le plugin Form 305
10.4 La bibliothèque jQuery UI 307
10.5 Autres plugins recommandés 317
10.6 En résumé 328

11 Développement de plugins 329
11.1 Ajouter de nouvelles fonctions globales 329
11.2 Ajouter des méthodes à l'objet jQuery 333
11.3 Méthodes de parcours du DOM 337
11.4 Ajouter des méthodes abrégées 340
11.5 Paramètres d'une méthode 344
11.6 Ajouter une expression de sélection 353
11.7 Distribuer un plugin 355
11.8 En résumé 357

Annexe A Ressources en ligne 359
A.1 Documentation sur jQuery 359
A.2 Références JavaScript 360
A.3 Compacteurs de code JavaScript 361
A.4 Référence (X)HTML 361
A.5 Références CSS 362
A.6 Blogs 362
A.7 Frameworks de développement web utilisant jQuery 365

Annexe B Outils de développement 367
B.1 Outils pour Firefox 367
B.2 Outils pour Internet Explorer 368
B.3 Outils pour Safari 369
B.4 Outils pour Opera 370
B.5 Autres outils 370

Annexe C Fermetures en JavaScript 373
C.1 Fonctions internes 373
C.2 Interactions entre fermetures 377

C.3 Fermetures dans jQuery 378
C.4 Dangers liés aux fuites de mémoire 381
C.5 En résumé 384

Annexe D Référence rapide 385
D.1 Expressions de sélection 385
D.2 Méthodes de parcours du DOM 387
D.3 Méthodes d'événement 389
D.4 Méthodes d'effet 391
D.5 Méthodes de manipulation du DOM 392
D.6 Méthodes AJAX 395
D.7 Autres méthodes 396

Index 399

Avant-propos

J'ai été très heureux d'apprendre que Karl Swedberg et Jonathan Chaffer entreprenaient l'écriture de *jQuery.* En tant que premier ouvrage traitant de jQuery, ce livre a fixé un niveau de qualité que les autres sur le même thème, ou sur JavaScript en général, se sont efforcés d'atteindre. Depuis sa sortie, il a toujours fait partie des titres JavaScript en tête des ventes, principalement en raison de sa qualité et de sa prise en compte des détails.

J'étais particulièrement content que Karl et Jonathan s'attellent à cette tâche car je les connais bien et je savais qu'ils seraient d'excellents auteurs. Travaillant au cœur de l'équipe jQuery, j'ai eu l'occasion de connaître Karl au cours des deux dernières années, notamment pour la rédaction de cet ouvrage. Lorsque je regarde le résultat, ses compétences de développeur et d'enseignant de la langue anglaise étaient manifestement bien adaptées à ce projet.

J'ai également eu l'opportunité de rencontrer les auteurs en personne, un fait plutôt rare dans le monde des projets open-source distribués. Ils sont toujours des membres fidèles de la communauté jQuery.

La bibliothèque jQuery est utilisée par des personnes d'horizons différents, notamment des concepteurs, des développeurs et des programmeurs plus ou moins expérimentés. Il en est de même au sein de l'équipe jQuery. Ses membres aux connaissances variées apportent leurs avis sur la direction à donner au projet. Il existe cependant un point commun entre tous les utilisateurs de jQuery : ils constituent une communauté de développeurs et de concepteurs dont l'objectif est de simplifier le développement en JavaScript.

Dire qu'un projet open-source est de type communautaire ou que son but est d'aider les nouveaux utilisateurs tient du cliché. Toutefois, dans le cas de jQuery, cela n'a rien d'un vœu pieux. Au sein de l'équipe jQuery, les personnes qui s'occupent de la communauté jQuery, qui rédigent la documentation ou qui développent des plugins sont plus nombreuses que celles en charge du code de base. Bien que les responsables du développement de la bibliothèque fassent preuve d'un dynamisme élevé, c'est la communauté qui s'est établie autour de jQuery qui fait toute la différence entre un projet léthargique et un projet qui répond à chaque besoin, voire les anticipe.

Notre façon de mener le projet et votre manière d'utiliser le code sont fondamentalement très différentes de celles que l'on rencontre dans la plupart des projets open-source ou avec la plupart des bibliothèques JavaScript. Les membres du projet et de la communauté jQuery sont de véritables experts. Ils savent pourquoi jQuery est différent et font de leur mieux pour transmettre ces connaissances aux utilisateurs.

Pour comprendre ce qu'est la communauté jQuery, la lecture d'un article ne vous suffira pas. Vous devez vous y impliquer pour prendre pleinement conscience de ce qui s'y passe. J'espère que vous trouverez l'occasion de participer. Rejoignez-nous sur les forums, les listes de diffusion et les blogs, et laissez-nous vous guider lors de votre apprentissage de jQuery.

La bibliothèque jQuery n'est pas qu'un morceau de code. Elle représente la somme d'expériences qui ont conduit à son existence, avec les nombreux hauts et bas, les difficultés de son développement et l'excitation de la voir grandir et réussir. Son évolution s'est faite en phase avec les utilisateurs et les membres de l'équipe, avec un souci constant de compréhension et d'adaptation.

Lorsque j'ai constaté que cet ouvrage considérait jQuery comme un outil unifié, non comme une compilation des expériences qu'il cache, j'étais à la fois décontenancé et enthousiaste. Voir comment les autres apprennent, comprennent et façonnent jQuery, voilà ce qui rend ce projet si stimulant.

Je ne suis pas le seul à entretenir avec jQuery une relation différente d'une relation normale entre un utilisateur et un outil. Je ne suis pas certain de pouvoir expliquer pourquoi c'est ainsi, mais j'ai pu le constater de nombreuses fois – cet instant unique où le visage de l'utilisateur s'illumine lorsqu'il réalise à quel point jQuery va l'aider.

À un moment précis, le déclic se passe. L'utilisateur de jQuery comprend que l'outil dont il se sert est bien plus qu'un simple outil. L'écriture d'applications web dynamiques prend alors pour lui un tout autre sens. C'est une chose incroyable et réellement ma préférée dans le projet jQuery.

Je souhaite que vous ayez l'opportunité d'éprouver également cette sensation.

John Resig
Créateur de jQuery

Préface à l’édition française

L’apparition, il y a quelques années, de ce que l’on appelle le "Web 2.0" a suscité un regain d’intérêt pour le langage JavaScript. J’appelle Web 2.0 la capacité d’une page Internet à interagir avec son utilisateur, au point parfois de devenir une véritable application. Ainsi, JavaScript, autrefois cantonné à des actions minimes et souvent mésestimé, est petit à petit devenu quasiment incontournable dans la conception de sites se voulant modernes.

La principale difficulté lorsque l’on aborde un développement JavaScript est de tenir compte des différences d’interprétation et de syntaxe entre les divers navigateurs. Si Internet Explorer est celui qui donne le plus de fil à retordre, il ne faut pas oublier que, même pour les autres, des différences existent !

Ces deux facteurs conjugués, la forte demande en JavaScript pour obtenir des sites dynamiques et les différentes implémentations de ce langage, font que de nombreuses bibliothèques sont apparues ces dernières années afin de faciliter le travail des développeurs. Le but de ces bibliothèques (*frameworks* en anglais) est double : uniformiser la syntaxe de JavaScript pour la rendre compatible avec tous les navigateurs et ajouter de nouvelles fonctionnalités au langage. Parmi ces bibliothèques, jQuery semble aujourd’hui la plus populaire et la plus utilisée de toutes. Son succès vient probablement de son concept modulaire.

En plus des intentions générales citées précédemment, jQuery a été pensé selon trois idées majeures :

- faciliter la sélection d’ensembles d’éléments d'un document HTML ;
- offrir une capacité de chaînage des instructions sur les collections récupérées ;
- permettre aux développeurs de créer eux-mêmes et facilement des extensions personnalisées.

C’est l’articulation de ces trois principes, au-delà de la simple syntaxe, qui vous est présentée dans ce livre. Vous apprendrez à créer des fonctions concises, simples et réutilisables.

L'ensemble des explications s'appuie sur des exemples précis et fonctionnels, qui se mettent en place au fur et à mesure des notions traitées. Vous découvrirez ainsi immédiatement comment intégrer concrètement les fonctions abordées, mais aussi comment se construit un code jQuery.

Ces exemples ont le mérite de bien insister sur l'intérêt d'utiliser la bibliothèque de façon globale. En effet, trop souvent, ce genre de bibliothèque n'est employé que pour l'une de ses fonctionnalités (par exemple les fonctions AJAX), ce qui est une ineptie en termes de conception. C'est comme si vous utilisiez une photo haute définition comme vignette dans une page HTML sans l'optimiser au préalable. Il est dommage de charger une bibliothèque telle que jQuery sans tirer profit de l'ensemble de ses avantages et de toute sa puissance.

Didier Mouronval

Responsable des rubriques JavaScript et Ajax de developpez.com

À propos des auteurs

Jonathan Chaffer est le directeur technique de Structure Interactive, une agence web localisée à Grand Rapids, Michigan. Il y supervise des projets de développement web fondés sur diverses technologies et participe également aux tâches de programmation.

Au sein de la communauté open-source, Jonathan participe activement au projet Drupal, qui a adopté jQuery comme framework JavaScript. Il est l'auteur du module CCK (*Content Construction Kit*), très employé pour la gestion du contenu structuré sur les sites Drupal. Il est chargé des principales restructurations du système de menus de Drupal et de la référence de l'API développeur.

Jonathan vit à Grand Rapids avec sa femme, Jennifer.

Il souhaite remercier Jenny pour son enthousiasme et son soutien infatigables, Karl pour la motivation qu'il lui transmet lorsque l'esprit est faible, et la communauté Ars Technica pour son inspiration constante vers l'excellence technique.

Karl Swedberg est développeur web chez Fusionary Media à Grand Rapids, Michigan. Il passe le plus clair de son temps à mettre en œuvre des conceptions, en mettant l'accent sur du code HTML conforme aux standards, des styles CSS propres et du JavaScript non intrusif. En tant que membre de l'équipe du projet jQuery et contributeur actif à la liste de discussion jQuery, Karl participe à des ateliers et à des conférences et propose des formations professionnelles en Europe et en Amérique du Nord.

Avant son aventure dans le développement web, Karl a travaillé comme réviseur, comme professeur d'anglais au lycée et comme cafetier. Son intérêt pour la technologie s'est révélé au début des années 1990 alors qu'il travaillait chez Microsoft à Redmond, Washington, sans faiblir depuis.

Karl préférerait passer du temps avec sa femme, Sara, et ses deux enfants, Benjamin et Lucia.

Il souhaite remercier Sara pour son amour loyal et son soutien, et ses deux charmants enfants. Il exprime son plus profond respect envers Jonathan Chaffer pour son expertise en programmation, ainsi que sa gratitude pour avoir voulu écrire cet ouvrage avec lui.

Il remercie également John Resig pour avoir créé la meilleure bibliothèque JavaScript au monde et avoir réuni une communauté extraordinaire autour de ce projet. Ses remerciements vont au personnel de Packt Publishing, aux relecteurs techniques du livre, à jQuery Cabal et à tous ceux qui ont apporté leur aide et leurs idées.

Préface

L'histoire a commencé en 2005 par un travail passionné de John Resig, un petit génie du JavaScript qui travaille à présent pour la Mozilla Corporation. Inspiré par les pionniers du domaine, tels que Dean Edwards et Simon Willison, Resig a réuni un ensemble de fonctions pour faciliter la recherche d'éléments dans une page web et leur attribuer des comportements. Avant qu'il n'annonce publiquement son projet en janvier 2006, il avait ajouté la modification du DOM et les animations de base. Il a nommé son projet jQuery afin de souligner le rôle central de la recherche (*query*) des éléments dans une page web et leur manipulation par du code JavaScript. Au cours des quelques années qui ont suivi, le jeu des fonctionnalités de jQuery s'est développé, les performances se sont améliorées et certains des sites les plus populaires d'Internet l'ont adopté. Bien que Resig reste le développeur en chef du projet, jQuery a pu véritablement s'épanouir dans le monde open-source, jusqu'à s'enorgueillir aujourd'hui d'une équipe de développeurs JavaScript de premier plan et d'une communauté dynamique de milliers de développeurs.

Grâce à la bibliothèque JavaScript jQuery, vous pouvez améliorer vos sites web, quel que soit votre niveau d'expérience. En un seul fichier de taille réduite, elle offre de nombreuses fonctionnalités, une syntaxe facile à apprendre et une compatibilité robuste entre les navigateurs. Par ailleurs, les centaines de plugins développés pour étendre les fonctionnalités de jQuery en font un outil pratiquement indispensable lors de n'importe quel développement de scripts côté client.

jQuery est une introduction en douceur aux concepts de jQuery. Grâce à cet ouvrage, vous pourrez ajouter des interactions et des animations à vos pages, même si vos précédentes tentatives en JavaScript ont pu vous laisser perplexe. Il va vous aider à franchir les obstacles dressés par AJAX, les événements, les effets et les fonctionnalités avancées du langage JavaScript. Il contient également un court guide de référence de la bibliothèque jQuery, vers lequel vous pourrez vous tourner en cas de besoin.

Organisation du livre

Au Chapitre 1, *Premiers pas*, vous ferez vos premières armes avec jQuery. Ce chapitre commence par une description de cette bibliothèque JavaScript et de ce qu'elle peut vous apporter. Il explique comment obtenir et configurer jQuery, puis passe à l'écriture d'un premier script.

Le Chapitre 2, *Sélecteurs*, détaille l'utilisation des expressions de sélection de jQuery, ainsi que les méthodes de parcours du DOM afin de rechercher des éléments dans la page, où qu'ils se trouvent. Vous utiliserez jQuery pour appliquer des styles à divers éléments de la page, une opération qu'il est parfois impossible de réaliser avec du CSS pur.

Au Chapitre 3, *Événements*, vous utiliserez le mécanisme de gestion des événements de jQuery pour déclencher des comportements suite à des événements générés par le navigateur. Vous verrez comment jQuery facilite l'insertion non intrusive d'événements, avant même que le chargement de la page ne soit terminé. Ce chapitre abordera également des sujets plus avancés, comme la propagation des événements, la délégation et les espaces de noms.

Le Chapitre 4, *Effets*, présente des techniques d'animation avec jQuery. Il explique comment masquer, afficher et déplacer des éléments de la page en employant des effets à la fois utiles et agréables à l'œil.

Le Chapitre 5, *Manipulation du DOM*, montre comment modifier une page à la demande. Il vous apprendra à intervenir sur la structure d'un document HTML, ainsi que sur son contenu.

Au Chapitre 6, *AJAX*, vous découvrirez les nombreuses méthodes proposées par jQuery pour simplifier l'accès aux fonctionnalités côté serveur, sans recourir aux actualisations pénibles de la page.

Dans les trois chapitres suivants (7, 8 et 9), vous travaillerez sur plusieurs exemples réels, en réunissant tout ce que vous avez appris aux chapitres précédents pour créer des solutions jQuery robustes à des problèmes courants.

Au Chapitre 7, *Manipulation des tables*, vous trierez, filtrerez et mettrez en style des informations de manière à obtenir une mise en forme agréable et fonctionnelle des données.

Le Chapitre 8, *Manipulation des formulaires*, vous permettra de maîtriser les détails de la validation côté client, vous y concevrez une présentation de formulaire adaptative et mettrez en œuvre des fonctionnalités interactives côté client, telles que l'auto-complétion dans les champs du formulaire.

Le Chapitre 9, *Carrousels et prompteurs*, montre comment améliorer la présentation et l'intérêt des éléments d'une page en les affichant sous forme de groupes plus faciles à gérer. Vous ferez en sorte que les informations apparaissent et disparaissent, que ce soit automatiquement ou sous le contrôle de l'utilisateur.

Dans les deux chapitres suivants (10 et 11), vous dépasserez les méthodes standard de jQuery pour explorer les extensions tierces de la bibliothèque et verrez différentes manières de l'étendre vous-même.

Le Chapitre 10, *Utilisation des plugins*, s'intéresse au plugin Form et aux plugins officiels pour l'interface utilisateur réunis sous le nom jQuery UI. Vous y apprendrez également à rechercher d'autres plugins jQuery populaires et à connaître leurs fonctions.

Au Chapitre 11, *Développement de plugins*, vous verrez comment tirer profit des possibilités extraordinaires d'extension de jQuery pour développer vos propres plugins. Vous créerez vos propres fonctions utilitaires, ajouterez des méthodes à l'objet jQuery, écrirez des expressions de sélection personnalisées, etc.

L'Annexe A, *Ressources en ligne*, recommande plusieurs sites web qui proposent des informations sur différents sujets en rapport avec jQuery, JavaScript et le développement web en général.

À l'Annexe B, *Outils de développement*, vous découvrirez quelques programmes et utilitaires tiers pour l'édition et le débogage du code jQuery depuis votre environnement de développement personnel.

L'Annexe C, *Fermetures en JavaScript*, présente tout ce que vous devez savoir sur le concept de fermetures – ce qu'elles sont et comment les utiliser pour votre propre bénéfice.

L'Annexe D, *Référence rapide*, donne une vue d'ensemble de la bibliothèque jQuery, y compris chacune de ses méthodes et ses expressions de sélection. Sa présentation facile à parcourir vous sera précieuse en ces moments où vous savez ce que vous voulez faire mais sans être certain du nom de la méthode ou du sélecteur d'éléments.

Prérequis

Pour écrire et exécuter le code illustré dans cet ouvrage, vous avez besoin des éléments suivants :

- un éditeur de texte ;
- un navigateur web moderne, comme Chrome de Google, Firefox de Mozilla, Safari d'Apple ou Internet Explorer de Microsoft ;
- la bibliothèque jQuery, en version 1.3.2 ou ultérieure, téléchargeable depuis le site **http://jquery.com/**.

Par ailleurs, pour mettre en œuvre les exemples AJAX du Chapitre 6, vous aurez besoin d'un serveur compatible PHP.

Public du livre

Ce livre est destiné aux concepteurs web qui souhaitent ajouter des éléments interactifs à leurs créations et aux développeurs qui veulent donner à leurs applications web la meilleure interface utilisateur. Des connaissances minimales en programmation JavaScript sont indispensables ; vous devez être à l'aise avec la syntaxe de ce langage. Vous devez posséder les bases du balisage HTML et des styles CSS. Aucune connaissance de jQuery n'est requise, pas plus qu'une expérience avec d'autres bibliothèques JavaScript.

Conventions typographiques

Cet ouvrage emploie plusieurs styles pour le texte afin de mettre en exergue les différentes informations. Voici quelques exemples de ces styles, avec leur signification.

Dans le texte, le code est présenté de la manière suivante : "Nous pouvons inclure d'autres contextes à l'aide de la directive `include`."

Lorsqu'un élément de code correspond à une valeur indiquée par l'utilisateur, il est écrit en italique de la manière suivante : `.find(`*`sélecteur`*`)`.

Voici un exemple de bloc de code :

```
<html>
  <head>
    <title>Le titre</title>
  </head>
  <body>
    <div>
      <p>Un paragraphe.</p>
      <p>Un autre paragraphe.</p>
      <p>Encore un autre paragraphe.</p>
    </div>
  </body>
</html>
```

Lorsque nous souhaitons attirer votre attention sur une partie d'un bloc de code, les lignes ou les éléments correspondants sont indiqués en gras :

```
$(document).ready(function() {
  $('a[href^=mailto:]').addClass('mailto');
  $('a[href$=.pdf]').addClass('pdflink');
  $('a[href^=http][href*=henry]')
    .addClass('henrylink');
});
```

Les entrées et les sorties de la ligne de commande se présentent de la manière suivante :

```
fonctExt():
Fonction externe
Fonction interne
```

Les *termes nouveaux* et les *mots importants* sont écrits dans une police italique. Les mots affichés à l'écran, par exemple dans les menus ou les boîtes de dialogue, apparaissent dans le texte sous la forme suivante : "Notez l'icône PDF à droite du lien HAMLET, l'icône d'enveloppe à côté du lien EMAIL, ainsi que le fond blanc et la bordure noire autour du lien HENRY V."

ATTENTION

Les avertissements signalent des actions à éviter.

ASTUCE

Ces encadrés présentent des astuces.

INFO

Ces notes proposent des informations supplémentaires sur le sujet en cours.

Votre avis

L'avis de nos lecteurs étant toujours le bienvenu, n'hésitez pas à nous faire part de vos commentaires. Pour cela, rendez-vous sur la page dédiée à cet ouvrage sur le site web Pearson (**http://www.pearson.fr**) et cliquez sur le lien RÉAGIR.

Si vous souhaitez proposer la publication d'un titre dont vous avez besoin ou si vous souhaitez devenir auteur, ouvrez la page CONTACTS sur le site web Pearson, remplissez le formulaire présenté et envoyez-le au service EDITORIAL.

Malgré tout le soin apporté à la rédaction du contenu de cet ouvrage, des erreurs sont toujours possibles. Si vous en rencontrez, que ce soit dans le texte ou dans le code, merci de nous les indiquer en allant sur la page CONTACTS du site web Pearson et en envoyant le formulaire rempli au service GÉNÉRAL.

Télécharger le code

Les fichiers des exemples de code sont disponibles depuis le site web Pearson (**http://www.pearson.fr**), en suivant le lien CODES SOURCES sur la page dédiée à ce livre. Ils contiennent les instructions permettant de les employer.

1

Premiers pas

Au sommaire de ce chapitre

- Intérêt de jQuery
- Efficacité de jQuery
- Historique du projet jQuery
- Première page web avec jQuery
- En résumé

Le World Wide Web est aujourd'hui un environnement dynamique et ses utilisateurs ont des exigences élevées quant à l'aspect et aux fonctions des sites. Pour construire des sites interactifs intéressants, les développeurs se tournent vers des bibliothèques JavaScript, comme jQuery, qui leur permettent d'automatiser les tâches courantes et de simplifier les plus complexes. La popularité de jQuery vient de sa capacité à simplifier un grand nombre de tâches.

Les fonctionnalités de jQuery étant nombreuses, il peut sembler difficile de savoir par où commencer. Toutefois, sa conception fait preuve de cohérence et de symétrie. La plupart de ses concepts sont empruntés à la structure de *HTML* et de *CSS* (*Cascading Style Sheets*). Les concepteurs ayant peu d'expérience en programmation peuvent ainsi démarrer rapidement, car la plupart des développeurs web maîtrisent mieux ces deux technologies que JavaScript. Dans ce premier chapitre, nous allons écrire un programme jQuery opérationnel constitué uniquement de trois lignes de code. Les programmeurs expérimentés bénéficieront également de cette cohérence conceptuelle, comme nous le verrons plus loin dans les chapitres traitant de sujets plus avancés.

Voyons à présent ce que jQuery peut apporter.

1.1 Intérêt de jQuery

La bibliothèque jQuery fournit une couche d'abstraction générique pour les scripts web classiques. Elle est donc utile pour la plupart des développements de scripts. En raison de sa nature extensible, il est impossible d'étudier toutes ses utilisations dans un même ouvrage, d'autant que le développement de plugins proposant de nouvelles possibilités est permanent. Toutefois, les fonctionnalités standard permettent de répondre aux besoins suivants :

- **Accéder aux éléments d'un document.** Sans l'aide d'une bibliothèque JavaScript, il faut écrire de nombreuses lignes de code pour parcourir l'arborescence du *DOM* (*Document Object Model*) et localiser des parties spécifiques de la structure d'un document HTML. jQuery fournit un mécanisme de sélection robuste et efficace qui permet de retrouver n'importe quels éléments d'un document afin de les examiner ou de les manipuler.
- **Modifier l'aspect d'une page web.** CSS propose une solution puissante pour modifier le rendu des documents, mais elle montre ses limites lorsque les navigateurs web ne respectent pas les mêmes standards. Avec jQuery, les développeurs peuvent contourner ce problème en se fondant sur une prise en charge identique des standards quels que soient les navigateurs. Par ailleurs, la bibliothèque permet de modifier les classes ou les propriétés de style appliquées à une partie du document, même après que la page a été affichée.
- **Altérer le contenu d'un document.** jQuery ne se borne pas à des changements cosmétiques mais permet de modifier le contenu du document. Du texte peut être changé, des images peuvent être insérées ou interverties, des listes être réordonnées, l'intégralité de la structure du contenu HTML peut être revue et étendue. Toutes ces possibilités sont permises par une *API* (*Application Programming Interface*) simple d'emploi.
- **Répondre aux actions de l'utilisateur.** Même les comportements les plus élaborés et les plus puissants ne sont d'aucune utilité si leur exécution n'est pas contrôlée. La bibliothèque jQuery propose une solution élégante pour intercepter une grande variété d'événements, comme un clic sur un lien, sans avoir à mélanger des gestionnaires d'événements au code HTML. De plus, cette API de gestion des événements permet de passer outre les incohérences des navigateurs, qui constituent une véritable nuisance pour les développeurs web.
- **Animer les modifications d'un document.** Pour la bonne mise en œuvre des comportements interactifs, le concepteur doit également fournir un retour visuel à l'utilisateur. La bibliothèque jQuery apporte son aide en proposant des animations, comme les effets de fondu et de volet, ainsi qu'une boîte à outils pour en construire de nouvelles.

- **Récupérer des informations à partir d'un serveur sans actualiser la page.** Ce type de code, connu sous le nom *AJAX* (*Asynchronous JavaScript And XML*), aide les développeurs web à créer un site réactif proposant des fonctionnalités riches. La bibliothèque jQuery permet de retirer de ce code les complexités propres aux navigateurs et de laisser les développeurs se concentrer sur l'aspect serveur.
- **Simplifier les tâches JavaScript courantes.** Outre les fonctionnalités de jQuery orientées document, la bibliothèque apporte des améliorations aux constructions JavaScript de base, comme les itérations et la manipulation des tableaux.

1.2 Efficacité de jQuery

Avec l'intérêt récent porté au HTML dynamique, les frameworks JavaScript ont proliféré. Certains sont spécialisés et se focalisent sur une ou deux des tâches précédentes. D'autres tentent de réunir au sein d'un même paquetage tous les comportements et toutes les animations imaginables. Pour offrir les diverses fonctionnalités décrites précédemment, tout en restant compact, jQuery emploie plusieurs stratégies :

- **Exploiter CSS.** En fondant le mécanisme de localisation des éléments de la page sur les *sélecteurs CSS*, jQuery hérite d'une méthode concise mais lisible pour exprimer une structure de document. La bibliothèque devient un point d'entrée pour les concepteurs qui souhaitent ajouter des comportements à la page, car la connaissance de la syntaxe de CSS est un prérequis au développement web professionnel.
- **Accepter les extensions.** Pour éviter les dangers de la prolifération des fonctionnalités, jQuery relègue la prise en charge des cas d'utilisation spéciaux aux *plugins*. La procédure de création de nouveaux plugins étant simple et parfaitement documentée, elle a encouragé le développement d'une grande diversité de modules créatifs et utiles. Par ailleurs, les fonctionnalités standard de jQuery sont elles-mêmes réalisées par des plugins et peuvent être retirées pour obtenir une bibliothèque plus compacte.
- **Masquer les excentricités du navigateur.** Malheureusement, le développement web est confronté au non-respect des standards par les différents navigateurs. Une part importante d'une application web peut être dédiée aux différentes mises en œuvre des fonctionnalités pour chaque plateforme. Si le monde des navigateurs en constante évolution rend impossible la création d'une base de code parfaitement indépendante de la plateforme pour les fonctions élaborées, jQuery ajoute une *couche d'abstraction* qui normalise les tâches courantes, permettant ainsi de simplifier le code et d'en réduire la taille.
- **Manipuler des ensembles.** Lorsque nous demandons à jQuery de "trouver tous les éléments de classe `collapsible` et de les masquer", il n'est pas nécessaire de parcourir chaque élément retourné. En effet, les méthodes comme `.hide()` sont

conçues pour travailler automatiquement sur des ensembles d'objets à la place d'objets individuels. Grâce à cette technique, nommée *itération implicite*, les constructions de type boucle sont généralement inutiles et le code devient plus court.

- **Autoriser plusieurs actions sur une ligne.** Pour éviter un abus des variables temporaires ou des répétitions coûteuses, jQuery emploie avec la plupart de ses méthodes un motif de programmation appelé *chaînage*. Autrement dit, le résultat de la plupart des opérations sur un objet est l'objet lui-même, auquel la prochaine action peut être appliquée.

Grâce à toutes ces stratégies, le paquetage jQuery a pu rester compact – moins de 20 ko après compression –, tout en proposant des techniques qui permettent au code qui utilise la bibliothèque de rester lui aussi compact.

L'élégance de la bibliothèque vient en partie de sa conception et en partie de son processus évolutionniste encouragé par la communauté dynamique qui s'est développée autour du projet. Les utilisateurs de jQuery se rassemblent pour discuter non seulement du développement des plugins, mais également des améliorations du noyau de la bibliothèque. L'Annexe A, *Ressources en ligne*, détaille plusieurs ressources communautaires proposées aux développeurs jQuery.

En dépit de tout le travail nécessaire à la conception d'un système aussi souple et robuste, le produit final est libre d'utilisation. Ce projet open-source est à la fois sous licence *GNU Public License* (adaptée à une utilisation dans d'autres projets open-source) et sous licence *MIT License* (pour faciliter son utilisation dans des logiciels propriétaires).

1.3 Historique du projet jQuery

Cet ouvrage s'articule autour de jQuery 1.3.x, c'est-à-dire la version la plus récente de la bibliothèque au moment de l'écriture de ces lignes. L'objectif de la bibliothèque – apporter une solution simple au problème de recherche des éléments dans une page web et de leur manipulation – n'a pas changé au cours de son développement, contrairement à certains détails de sa syntaxe et à ses fonctionnalités. Ce court historique du projet décrit les changements les plus importants entre les versions.

- **Phase de développement public.** John Resig a mentionné pour la première fois une amélioration de la bibliothèque "Behaviour" de Prototype en août 2005. Ce nouveau framework est officiellement sorti le 14 janvier 2006 sous le nom *jQuery.*
- **jQuery 1.0 (août 2006).** Cette première version stable de la bibliothèque disposait déjà d'une prise en charge robuste des sélecteurs CSS, de la gestion des événements et des interactions AJAX.

- **jQuery 1.1 (janvier 2007).** Cette version a considérablement assaini l'API. Les méthodes rarement employées ont été combinées afin de réduire le nombre de méthodes à apprendre et à documenter.
- **jQuery 1.1.3 (juillet 2007).** Cette version intermédiaire a intégré de nombreuses améliorations des performances du moteur de sélection de jQuery. À partir de là, l'efficacité de jQuery est devenue supérieure à celle des bibliothèques JavaScript comparables, comme Prototype, Mootools et Dojo.
- **jQuery 1.2 (septembre 2007).** La syntaxe *XPath* pour la sélection des éléments a été retirée de cette version car elle était redondante avec celle de CSS. La personnalisation des effets est devenue beaucoup plus souple et le développement des plugins, beaucoup plus simple grâce à l'ajout des *événements liés à un espace de noms*.
- **jQuery UI (septembre 2007).** Cette nouvelle suite de plugins a remplacé le plugin Interface, certes réputé mais vieillissant. Une riche collection de widgets préfabriqués était incluse, ainsi qu'un ensemble d'outils pour construire des éléments sophistiqués comme des interfaces avec glisser-déposer.
- **jQuery 1.2.6 (mai 2008).** Les fonctionnalités du plugin Dimensions de Brandon Aaron ont été intégrées à la bibliothèque principale.
- **jQuery 1.3 (janvier 2009).** Une refonte majeure du moteur de sélection (*Sizzle*) a permis une grande amélioration des performances de la bibliothèque. La *délégation des événements* est officiellement prise en charge.

INFO

Les notes pour les versions antérieures de jQuery peuvent être consultées sur le site web du projet (**http://docs.jquery.com/History_of_jQuery**).

1.4 Première page web avec jQuery

Puisque nous avons vu l'étendue des fonctionnalités que propose jQuery, voyons à présent comment mettre en œuvre cette bibliothèque.

Télécharger jQuery

Sur le site web officiel de jQuery (**http://jquery.com/**), vous trouverez toujours la version la plus récente de la bibliothèque et son actualité. Pour démarrer, nous avons besoin d'une copie de jQuery, téléchargeable directement depuis la page d'accueil du site. Plusieurs versions de la bibliothèque peuvent être disponibles à tout moment. En tant que développeurs de sites, la version non compressée la plus récente nous convien-

dra parfaitement. Elle pourra être remplacée par une version compressée dans les environnements de production.

Aucune installation n'est nécessaire. Pour utiliser la bibliothèque jQuery, il suffit de la placer sur le site dans un emplacement public. Puisque JavaScript est un langage interprété, aucune phase de compilation ou de construction n'est requise. Lorsqu'une page a besoin de jQuery, il suffit simplement de faire référence au fichier correspondant depuis le document HTML.

Configurer le document HTML

La plupart des exemples d'utilisation de jQuery comprennent trois composants : le document HTML lui-même, les fichiers CSS définissant ses styles et les fichiers JavaScript pour sa manipulation. Dans notre premier exemple, nous allons utiliser une page contenant un extrait de livre, avec plusieurs classes appliquées à différents éléments.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

    <title>De l'autre côté du miroir</title>

    <link rel="stylesheet" href="alice.css" type="text/css" />

    <script src="jquery.js" type="text/javascript"></script>
    <script src="alice.js" type="text/javascript"></script>
  </head>

  <body>
    <h1>De l'autre côté du miroir</h1>
    <div class="author">par Lewis Carroll</div>
    <div class="chapter" id="chapter-1">
      <h2 class="chapter-title">1. La maison du miroir</h2>
      <p>Sur la table, tout près d'Alice, il y avait un livre. Tout en
        observant le Roi Blanc (car elle était encore un peu inquiète à son
        sujet, et se tenait prête à lui jeter de l'encre à la figure au cas où
        il s'évanouirait de nouveau), elle se mit à tourner les pages pour
        trouver un passage qu'elle pût lire... <span class="spoken">« car
        c'est écrit dans une langue que je ne connais pas »,</span> se
        dit-elle.</p>
      <p>Et voici ce qu'elle avait sous les yeux :</p>
      <div class="poem">
        <h3 class="poem-title">YKCOWREBBAJ</h3>
        <div class="poem-stanza">
          <div>sevot xueutcils sel ; eruehlirg tiaté lI'</div>
          <div>: tneialbirv te edniolla'l rus tneiaryG</div>
          <div>; sevogorob sel tneialla xuerovilf tuoT</div>
          <div>.tneialfirnuob sugruof snohcrev seL</div>
        </div>
      </div>
```

```
      <p>Elle se cassa la tête là-dessus pendant un certain temps, puis,
        brusquement, une idée lumineuse lui vint à l'esprit : <span
        class="spoken">« Mais bien sûr ! c'est un livre du
        Miroir ! Si je le tiens devant un miroir, les mots seront de
        nouveau comme ils doivent être. »</span></p>
      <p>Et voici le poème qu'elle lut :</p>
      <div class="poem">
        <h3 class="poem-title">JABBERWOCKY</h3>
        <div class="poem-stanza">
          <div>'Il était grilheure ; les slictueux toves</div>
          <div>Gyraient sur l'alloinde et vriblaient :</div>
          <div>Tout flivoreux allaient les borogoves ;</div>
          <div>Les verchons fourgus bourniflaient.</div>
        </div>
      </div>
    </div>
  </body>
</html>
```

INFO

L'organisation des fichiers sur le serveur n'a pas réellement d'importance. Les références entre les fichiers doivent simplement être adaptées à l'organisation choisie. Dans la plupart des exemples étudiés dans cet ouvrage, nous désignerons les fichiers par des chemins relatifs (`../images/foo.png`) à la place de chemins absolus (`/images/foo.png`). Le code pourra ainsi être exécuté localement sans nécessiter un serveur web.

Immédiatement après le préambule HTML classique, la feuille de style est chargée. Pour cet exemple, elle est relativement réduite.

```
body {
  font: 62.5% Arial, Verdana, sans-serif;
}
h1 {
  font-size: 2.5em;
  margin-bottom: 0;
}
h2 {
  font-size: 1.3em;
  margin-bottom: .5em;
}
h3 {
  font-size: 1.1em;
  margin-bottom: 0;
}
.poem {
  margin: 0 2em;
}
.highlight {
  font-style: italic;
  border: 1px solid #888;
  padding: 0.5em;
  margin: 0.5em 0;
  background-color: #ffc;
}
```

Après la référence à la feuille de style, les fichiers JavaScript sont inclus. Il est important que la balise `<script>` pour jQuery soit placée *avant* celle de nos scripts. Dans le cas contraire, la bibliothèque ne serait pas disponible au moment où notre code tenterait de l'utiliser.

INFO

Dans la suite de cet ouvrage, seules les parties pertinentes des fichiers HTML et CSS seront présentées. Les fichiers complets sont disponibles en français sur le site de l'éditeur (**http://www.pearson.fr**) et en anglais sur le site web dédié à ce livre (**http://book.learning-jquery.com**).

La page obtenue est illustrée à la Figure 1.1.

Figure 1.1

La page avant application d'un style avec jQuery.

Nous allons utiliser jQuery pour appliquer un nouveau style au texte du poème.

INFO

Cet exemple a pour objectif d'illustrer une utilisation simple de jQuery. Dans la réalité, un tel style pourrait être appliqué uniquement à l'aide de CSS.

Ajouter jQuery

Notre code personnalisé doit être placé dans le second fichier JavaScript, actuellement vide, qui est inclus dans le document HTML en utilisant `<script src="alice.js" type="text/javascript"></script>`. Pour cet exemple, nous avons seulement besoin de trois lignes de code :

```
$(document).ready(function() {
  $('.poem-stanza').addClass('highlight');
});
```

Rechercher le texte du poème

Dans jQuery, la sélection d'une partie du document constitue l'opération fondamentale. Pour cela, nous utilisons la construction `$()`. De manière générale, une chaîne de caractères contenant une expression de sélection CSS est passée en paramètre. Dans notre exemple, nous souhaitons rechercher toutes les parties du document auxquelles la classe `poem-stanza` est appliquée. Par conséquent, le sélecteur est très simple. Nous rencontrerons des options plus complexes tout au long de cet ouvrage. Au Chapitre 2, nous détaillerons les différentes manières de localiser les éléments d'un document.

La fonction `$()` est en réalité une fabrique pour *l'objet jQuery*, qui constitue la brique de base avec laquelle nous travaillerons à partir de maintenant. L'objet jQuery encapsule des éléments du DOM, zéro ou plus, et nous permet d'interagir avec eux de différentes manières. Dans notre cas, nous souhaitons modifier l'aspect de ces parties de la page et, pour cela, nous changerons les classes appliquées au texte du poème.

Injecter la nouvelle classe

Comme la plupart des méthodes de jQuery, la méthode `.addClass()` a un nom parfaitement explicite : elle applique une classe CSS à la partie de la page que nous avons sélectionnée. Elle prend comme seul paramètre le nom de la classe à ajouter. Cette méthode ainsi que son complément `.removeClass()` nous permettront d'observer facilement le fonctionnement de jQuery au cours de notre exploration des différentes expressions de sélection. Pour le moment, notre exemple ajoute simplement la classe `highlight`, que la feuille de style définit comme du texte en italique avec une bordure.

Vous remarquerez qu'aucune itération n'est requise pour ajouter la classe à toutes les strophes du poème. Nous l'avons indiqué, jQuery utilise une *itération implicite* dans les méthodes comme `.addClass()` et un seul appel de fonction suffit pour modifier toutes les parties sélectionnées du document.

Exécuter le code

`$()` et `.addClass()` suffisent pour atteindre notre but, c'est-à-dire changer l'aspect du texte du poème. Cependant, si cette ligne de code est insérée en tant que telle dans l'entête du document, elle n'aura aucun effet. Le code JavaScript est généralement exécuté

dès qu'il est rencontré par le navigateur et, au moment du traitement de l'en-tête, il n'existe encore aucun contenu HTML auquel appliquer un style. Nous devons retarder l'exécution du code jusqu'à ce que le DOM soit disponible.

Pour contrôler l'exécution d'un code JavaScript, le mécanisme classique consiste à invoquer ce code depuis des *gestionnaires d'événements*. Il existe de nombreux gestionnaires pour les événements déclenchés par l'utilisateur, comme les clics avec les boutons de la souris et les appuis sur les touches du clavier. Si nous n'avions pas jQuery à notre disposition, nous devrions employer le gestionnaire `onload`, qui est invoqué après que la page, ainsi que toutes ses images, a été affichée. Pour déclencher l'appel à notre code suite à l'événement `onload`, nous devons le placer dans une fonction :

```
function highlightPoemStanzas() {
  $('.poem-stanza').addClass('highlight');
}
```

Ensuite, nous associons la fonction à l'événement en y faisant référence dans la balise HTML `<body>` :

```
<body onload="highlightPoemStanzas();">
```

De cette manière, notre code est exécuté lorsque le chargement de la page est terminé.

Cette approche présente quelques inconvénients. Nous avons altéré le contenu HTML lui-même pour mettre en place ce comportement. Ce couplage étroit entre la structure et la fonction conduit à un encombrement du code, avec un risque de répétition des mêmes appels de fonctions sur différentes pages ou, dans le cas d'autres événements tels que les clics de souris, sur chaque instance d'un élément d'une page. L'ajout de nouveaux comportements impose une intervention en plusieurs endroits, ce qui augmente les sources d'erreurs et complique le travail parallèle des concepteurs et des programmeurs.

Pour éviter ce problème, jQuery nous permet de planifier l'invocation de fonctions après que le DOM a été chargé, sans attendre l'affichage des images, en utilisant la construction `$(document).ready()`. Avec la fonction définie précédemment, nous pouvons écrire la ligne suivante :

```
$(document).ready(highlightPoemStanzas);
```

Cette technique ne nécessite aucune modification du code HTML. Le comportement est totalement associé à partir du fichier JavaScript. Au Chapitre 3, nous verrons comment répondre à d'autres types d'*événements*, en séparant également leurs effets de la structure HTML.

La solution proposée reste toutefois peu économique, car nous avons dû définir une fonction, `highlightPoemStanzas()`, alors qu'elle est employée immédiatement et une seule fois. Autrement dit, nous avons créé un identifiant dans l'espace de noms global des fonctions, sans que cela nous apporte un réel avantage, avec l'inconvénient d'avoir à nous rappeler de ne pas le réutiliser. Comme d'autres langages de programmation,

JavaScript propose une réponse à cette inefficacité : les *fonctions anonymes* (parfois appelées *fonctions lambda*). Grâce à ces fonctions, nous pouvons écrire le code tel qu'il était initialement présenté :

```
$(document).ready(function() {
  $('.poem-stanza').addClass('highlight');
});
```

En utilisant le mot clé `function` sans préciser un nom de fonction, nous définissons une fonction exactement là où elle est requise, non avant. Cela permet de réduire le désordre dans le code et nous conduit à trois lignes de JavaScript. Cet idiome est extrêmement pratique avec le code jQuery car de nombreuses méthodes prennent en argument une fonction et toutes ces fonctions sont rarement réutilisables.

Lorsque cette syntaxe est employée pour définir une fonction anonyme dans le corps d'une autre fonction, il est possible de créer une *fermeture* (*closure*). Il s'agit d'un concept élaboré et puissant qu'il est indispensable de maîtriser car la définition d'un grand nombre de fonctions imbriquées peut avoir des conséquences et des implications inattendues sur l'usage de la mémoire. Nous y reviendrons en détail à l'Annexe C.

Le produit final

Notre code JavaScript étant en place, nous obtenons la page illustrée à la Figure 1.2.

Figure 1.2

La page après application d'un style avec jQuery.

Les strophes du poème sont à présent en italique et placées dans des boîtes, comme précisé par la feuille de style `alice.css`. En effet, notre code JavaScript a ajouté la classe `highlight` à ces éléments du document.

1.5 En résumé

À présent, nous comprenons pourquoi un développeur optera pour un framework JavaScript au lieu d'écrire tout le code à partir de zéro, même pour les tâches les plus élémentaires. Nous avons vu quelques cas dans lesquels la bibliothèque jQuery excelle et pourquoi son choix s'impose. Nous savons également quelles tâches elle permet de simplifier.

Dans ce chapitre, nous avons appris à disposer de jQuery dans le code JavaScript de la page web, à utiliser la fonction `$()` pour localiser les éléments de la page ayant une certaine classe, à invoquer `.addClass()` pour appliquer des styles supplémentaires à ces parties de la page, et à appeler `$(document).ready()` pour déclencher l'exécution de ce code après le chargement de la page.

L'exemple choisi a permis d'illustrer le fonctionnement de jQuery, mais il n'est pas très utile dans les cas réels. Au chapitre suivant, nous étendrons ce code pour explorer les sélections sophistiquées de jQuery, en choisissant des utilisations pratiques de cette technique.

2

Sélecteurs

Au sommaire de ce chapitre

- Le DOM
- La fonction *$()*
- Sélecteurs CSS
- Sélecteurs d'attribut
- Sélecteurs personnalisés
- Méthodes de parcours du DOM
- Accéder aux éléments du DOM
- En résumé

La bibliothèque jQuery se fonde sur la puissance des sélecteurs CSS (*Cascading Style Sheets*) pour que nous puissions accéder rapidement et facilement à des éléments ou à des groupes d'éléments du DOM. Dans ce chapitre, nous examinerons quelques sélecteurs CSS, ainsi que des sélecteurs propres à jQuery. Nous examinerons également le parcours du DOM à l'aide des méthodes de jQuery, qui apportent une plus grande souplesse encore.

2.1 Le DOM

L'un des aspects les plus puissants de jQuery réside dans sa capacité à sélectionner aisément des éléments dans le DOM (*Document Object Model*). Le DOM est une sorte de structure généalogique arborescente. HTML, comme d'autres langages de balisage, utilise ce modèle pour décrire les relations entre les éléments d'une page. Pour faire référence à ces relations, nous employons la terminologie associée aux relations familiales : parents, enfants, etc. Un exemple simple permettra de mieux comprendre l'application de la métaphore généalogique à un document :

```
<html>
  <head>
    <title>Le titre</title>
  </head>
  <body>
    <div>
      <p>Un paragraphe.</p>
      <p>Un autre paragraphe.</p>
      <p>Encore un autre paragraphe.</p>
    </div>
  </body>
</html>
```

`<html>` est l'*ancêtre* de tous les autres éléments ; autrement dit, tous les autres éléments sont des *descendants* de `<html>`. Les éléments `<head>` et `<body>` sont non seulement des descendants mais également des *enfants* de `<html>`. De même, en plus d'être l'ancêtre de `<head>` et de `<body>`, `<html>` est également leur *parent*. Les éléments `<p>` sont des enfants (et des descendants) de `<div>`, des descendants de `<body>` et de `<html>`, et des *frères*. Pour de plus amples informations concernant l'affichage de la structure arborescente du DOM à l'aide de logiciels tiers, consultez l'Annexe B.

Avant d'aller plus loin, il est important de noter que le jeu d'éléments obtenu à l'aide des sélecteurs et des méthodes est toujours placé dans un objet jQuery. Les objets jQuery permettent de manipuler très facilement les éléments recherchés dans une page. Nous pouvons aisément lier des *événements* à ces objets et leur ajouter de jolis effets, tout comme *enchaîner* de multiples modifications ou effets. Toutefois, les objets jQuery diffèrent des éléments du DOM ou des listes de nœuds et, en tant que tels, n'offrent pas nécessairement les mêmes méthodes et propriétés pour certaines tâches. À la fin de ce chapitre, nous verrons comment accéder aux éléments du DOM qui sont encapsulés dans un objet jQuery.

2.2 La fonction *$()*

Quel que soit le type de sélecteur que nous voulons employer dans jQuery, l'instruction commence toujours avec le symbole du dollar et des parenthèses : `$()`. Tout ce qu'il est possible d'utiliser dans une feuille de style peut également être placé entre guillemets et inséré entre les parenthèses. Ainsi, nous pouvons ensuite appliquer des méthodes jQuery à la collection d'éléments obtenue.

Les trois constituants de base des sélecteurs sont un *nom de balise*, un *identifiant* (*ID*) et une *classe*. Ils peuvent être employés de manière indépendante ou en association avec d'autres sélecteurs. Le Tableau 2.1 donne un exemple pour chacun de ces trois sélecteurs lorsqu'ils sont employés indépendamment.

Éviter un conflit entre jQuery et d'autres bibliothèques JavaScript

Dans jQuery, le symbole du dollar ($) est un alias pour `jQuery`. Lorsque plusieurs bibliothèques sont utilisées dans une même page, il est possible que des conflits surviennent car la fonction `$()` est très souvent définie par les bibliothèques JavaScript. Pour éviter de tels conflits, nous pouvons remplacer chaque occurrence de `$` par `jQuery` dans le code jQuery. Le Chapitre 10 proposera d'autres solutions à ce problème.

Tableau 2.1 : Utilisation des trois sélecteurs de base

Sélecteur	*CSS*	*jQuery*	*Description*
Nom de balise	`p`	`$('p')`	Sélectionne tous les paragraphes du document.
Identifiant	`#un-id`	`$('#un-id')`	Sélectionne l'élément unique du document dont l'identifiant est `un-id`.
Classe	`.une-classe`	`$('.une-classe')`	Sélectionne tous les éléments du document dont la classe est `une-classe`.

Nous l'avons mentionné au Chapitre 1, lorsque nous associons des méthodes à la fonction `$()`, les éléments encapsulés dans l'objet jQuery sont parcourus automatiquement et implicitement. Par conséquent, nous pouvons généralement éviter l'*itération explicite*, par exemple avec une boucle `for`, qui est souvent de mise dans les scripts manipulant le DOM.

Puisque les bases sont à présent posées, nous sommes prêts à utiliser les sélecteurs de manière plus élaborée.

2.3 Sélecteurs CSS

La bibliothèque jQuery prend en charge pratiquement tous les sélecteurs décrits dans les spécifications 1 à 3 de CSS (voir le site du World Wide Web Consortium à l'adresse **http://www.w3.org/Style/CSS/#specs**). Ainsi, tant que JavaScript est activé, les développeurs peuvent enrichir leurs sites web sans se demander si le navigateur de l'utilisateur, en particulier Internet Explorer 6, comprendra ou non les sélecteurs élaborés.

INFO

Les développeurs jQuery sérieux doivent toujours appliquer les concepts d'*amélioration progressive* et de *dégradation élégante* à leur code afin de s'assurer qu'une page sera toujours affichée correctement, si ce n'est joliment, que JavaScript soit activé ou non. Nous reviendrons sur ces concepts tout au long de cet ouvrage.

Pour comprendre le fonctionnement de jQuery avec les sélecteurs CSS, nous nous appuierons sur une structure souvent employée par les sites web pour la navigation : la liste imbriquée non ordonnée.

```
<ul id="selected-plays" class="clear-after">
  <li>Comédies
    <ul>
      <li><a href="/commeilvousplaira/">Comme il vous plaira</a></li>
      <li>Tout est bien qui finit bien</li>
      <li>Le Songe d'une nuit d'été</li>
      <li>La Nuit des rois</li>
    </ul>
  </li>
  <li>Tragédies
    <ul>
      <li><a href="hamlet.pdf">Hamlet</a></li>
      <li>Macbeth</li>
      <li>Roméo et Juliette</li>
    </ul>
  </li>
  <li>Historiques
    <ul>
      <li>Henry IV (<a href="mailto:henryiv@king.co.uk">email</a>)
      <ul>
        <li>Partie I</li>
        <li>Partie II</li>
      </ul>
    </li>
    <li><a href="http://www.shakespeare.co.uk/henryv.htm">Henry V</a></li>
    <li>Richard II</li>
    </ul>
  </li>
</ul>
```

Vous remarquerez que l'identifiant de la première balise `<ul>` est `selected-plays`, mais qu'aucune classe n'est associée aux balises `<li>`. Lorsque aucun style n'est appliqué, la liste est affichée conformément à la Figure 2.1.

Figure 2.1

Aspect de la liste sans application d'un style.

Elle se présente comme attendu, avec des puces et des éléments organisés verticalement et indentés en fonction de leur niveau.

Appliquer un style aux éléments de liste

Supposons que nous voulions que les éléments de premier niveau, et *uniquement* ceux-là, soient organisés à l'horizontale. Nous pouvons commencer par définir une classe `horizontal` dans la feuille de style :

```
.horizontal {
  float: left;
  list-style: none;
  margin: 10px;
}
```

La classe `horizontal` fait en sorte que l'élément soit flottant à gauche de celui qui le suit, lui retire la puce s'il s'agit d'un élément de liste et ajoute une marge de 10 pixels sur ses quatre côtés.

Au lieu d'associer directement la classe `horizontal` à un élément dans le document HTML, nous l'ajoutons dynamiquement aux éléments de premier niveau de la liste, c'est-à-dire COMÉDIES, TRAGÉDIES et HISTORIQUES. Cela nous permet d'illustrer l'utilisation des sélecteurs dans jQuery :

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
});
```

Notre code jQuery débute par l'enveloppe `$(document).ready()`, qui s'exécute dès que le DOM a été chargé.

La deuxième ligne utilise le *combinateur d'enfant* (`>`) pour ajouter la classe `horizontal` à tous les éléments de premier niveau et à eux seuls. Le sélecteur placé dans la fonction `$()` signifie "rechercher chaque élément de liste (`li`) qui est un enfant (`>`) de l'élément dont l'identifiant est `selected-plays` (`#selected-plays`)".

La Figure 2.2 présente l'affichage de la liste après que la classe a été appliquée.

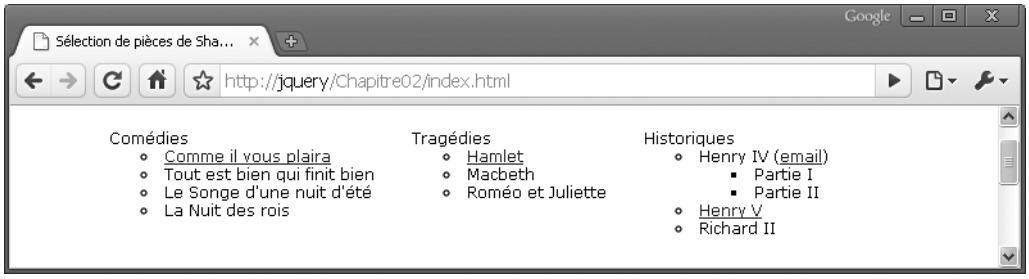


Figure 2.2

Aspect de la liste après application d'une classe aux éléments de premier niveau.

Pour donner un style à tous les autres éléments, c'est-à-dire ceux qui ne sont pas de premier niveau, nous avons plusieurs méthodes à notre disposition. Puisque nous avons déjà appliqué la classe `horizontal` aux éléments de premier niveau, nous pouvons sélectionner les éléments secondaires en utilisant une *pseudo-classe de négation* pour identifier tous les éléments de liste qui ne sont pas de la classe `horizontal`. Cela se passe dans la troisième ligne de code :

```
$(document).ready(function() {
  $('#selected-plays > li').addClass('horizontal');
  $('#selected-plays li:not(.horizontal)').addClass('sub-level');
});
```

Cette fois-ci, nous sélectionnons chaque élément de liste (`li`) qui est un descendant de l'élément dont l'identifiant est `selected-plays` (`#selected-plays`) et dont la classe n'est pas `horizontal` (`:not(.horizontal)`).

Lorsque nous ajoutons la classe `sub-level` à ces éléments, un arrière-plan gris leur est attribué par la feuille de style. La Figure 2.3 illustre la nouvelle présentation de la liste imbriquée.

Figure 2.3

Aspect de la liste après application d'une classe aux éléments secondaires.

2.4 Sélecteurs d'attribut

Les *sélecteurs d'attributs* de CSS sont particulièrement utiles. Ils permettent de désigner un élément par l'une de ses propriétés HTML, comme l'attribut `title` d'un lien ou l'attribut `alt` d'une image. Par exemple, pour sélectionner toutes les images qui possèdent un attribut `alt`, nous écrivons le code suivant :

```
$('img[alt]')
```

INFO

Dans les versions antérieures à la 1.2, jQuery utilisait la syntaxe du *langage XML Path* (*XPath*) pour ses sélecteurs d'attribut et proposait plusieurs autres sélecteurs XPath. Même si ces sélecteurs ont depuis été retirés de la bibliothèque jQuery standard, ils restent disponibles sous forme d'un plugin (**http://plugins.jquery.com/project/xpath/**).

Appliquer un style aux liens

Les sélecteurs d'attribut utilisent une syntaxe issue des expressions régulières pour identifier la valeur au début (`^`) ou à la fin (`$`) d'une chaîne de caractères. Nous pouvons également nous servir de l'astérisque (`*`) pour indiquer une valeur placée n'importe où dans une chaîne et un point d'exclamation (`!`) pour indiquer l'inverse d'une valeur.

Supposons que nous souhaitions appliquer des styles différents aux différents types de liens. Nous commençons par les définir dans la feuille de style :

```
a {
  color: #00c;
}
a.mailto {
  background: url(images/mail.png) no-repeat right top;
  padding-right: 18px;
}
a.pdflink {
  background: url(images/pdf.png) no-repeat right top;
  padding-right: 18px;
}
a.henrylink {
  background-color: #fff;
  padding: 2px;
  border: 1px solid #000;
}
```

Ensuite, nous utilisons jQuery pour ajouter les classes `mailto`, `pdflink` et `henrylink` aux liens appropriés.

Pour ajouter une classe à tous les liens de type courrier électronique, nous construisons un sélecteur qui recherche toutes les ancres (`a`) dont l'attribut `href` (`[href`) commence par `mailto:` (`^=mailto:]`) :

```
$(document).ready(function() {
  $('a[href^=mailto:]').addClass('mailto');
});
```

Pour ajouter une classe à tous les liens vers des fichiers PDF, nous utilisons le symbole du dollar à la place du symbole de l'accent circonflexe. En effet, nous voulons sélectionner des liens ayant un attribut `href` qui se *termine* par `.pdf` :

```
$(document).ready(function() {
  $('a[href^=mailto:]').addClass('mailto');
  $('a[href$=.pdf]').addClass('pdflink');
});
```

Il est également possible de combiner les sélecteurs d'attribut. Par exemple, nous pouvons ajouter la classe `henrylink` à tous les liens dont l'attribut `href` commence par `http` et inclut `henry` :

```
$(document).ready(function() {
  $('a[href^=mailto:]').addClass('mailto');
  $('a[href$=.pdf]').addClass('pdflink');
  $('a[href^=http][href*=henry]').addClass('henrylink');
});
```

La Figure 2.4 présente la nouvelle version de la liste après que ces trois classes ont été appliquées aux trois types de liens.

Figure 2.4

Aspect de la liste après application des classes aux différents types de liens.

Notez l'icône PDF à droite du lien HAMLET, l'icône d'enveloppe à côté du lien EMAIL, ainsi que le fond blanc et la bordure noire autour du lien HENRY V.

2.5 Sélecteurs personnalisés

À la grande diversité de sélecteurs CSS, jQuery ajoute ses propres sélecteurs personnalisés. Pour la plupart, ils permettent de sélectionner certains éléments dans un groupe. La syntaxe est identique à celle des *pseudo-classes CSS*, dans laquelle le sélecteur commence par des deux-points (`:`). Par exemple, pour sélectionner le deuxième élément parmi l'ensemble obtenu à l'aide d'un sélecteur de balises `<div>` dont la classe est `horizontal`, nous écrivons le code suivant :

```
$('div.horizontal:eq(1)')
```

La partie `:eq(1)` sélectionne le deuxième élément de l'ensemble car les indices des tableaux JavaScript commencent à zéro. À l'opposé, CSS numérote à partir de un. Par conséquent, un sélecteur CSS comme `$('div:nth-child(1)')` sélectionne tous les éléments `<div>` qui sont le premier enfant de leur parent (toutefois, dans ce cas, il est plus simple d'utiliser `$('div:first-child')`).

Appliquer un style en alternance aux lignes d'une table

Avec `:odd` et `:even`, jQuery propose deux sélecteurs personnalisés très intéressants. Voyons comment utiliser l'un d'eux pour créer des bandes dans la table suivante :

```
<table>
  <tr>
    <td>Comme il vous plaira</td>
    <td>Comédie</td>
    <td></td>
  </tr>
  <tr>
    <td>Tout est bien qui finit bien</td>
    <td>Comédie</td>
    <td>1601</td>
  </tr>
  <tr>
    <td>Hamlet</td>
    <td>Tragédie</td>
    <td>1604</td>
  </tr>
  <tr>
    <td>Macbeth</td>
    <td>Tragédie</td>
    <td>1606</td>
  </tr>
  <tr>
    <td>Roméo et Juliette</td>
    <td>Tragédie</td>
    <td>1595</td>
  </tr>
  <tr>
    <td>Henry IV, Partie I</td>
    <td>Historique</td>
    <td>1596</td>
  </tr>
  <tr>
    <td>Henry V</td>
    <td>Historique</td>
    <td>1599</td>
  </tr>
</table>
```

Nous complétons la feuille de style de manière à appliquer un style à toutes les lignes de la table et à définir une classe `alt` pour les lignes paires :

```
tr {
  background-color: #fff;
}
.alt {
  background-color: #ccc;
}
```

Notre code jQuery associe la classe `alt` aux lignes paires de la table (balises `<tr>`) :

```
$(document).ready(function() {
  $('tr:odd').addClass('alt');
});
```

Attention, pourquoi utilisons-nous le sélecteur `:odd` (impair) pour cibler des lignes paires ? Comme dans le cas du sélecteur `:eq()`, `:odd` et `:even` se fondent sur la numérotation JavaScript, qui commence à zéro. Par conséquent, la première ligne possède le numéro 0 (pair), la deuxième, le numéro 1 (impair), etc. La Figure 2.5 montre le résultat de notre petit bout de code sur la table.

Figure 2.5

Application alternée d'un style aux lignes d'une table.

Vous constaterez un résultat sans doute inattendu si la page contient plusieurs tables. Par exemple, puisque la dernière ligne de cette table possède un fond blanc, la première ligne de la table suivante aura un fond gris. Pour éviter ce type de problème, la solution consiste à utiliser le sélecteur `:nth-child()`. Il peut prendre en argument un nombre pair ou impair. Attention, cependant, car `:nth-child()` est le seul sélecteur jQuery qui commence à un. Pour obtenir le même effet que précédemment et pour qu'il reste cohérent sur les multiples tables d'un document, nous modifions le code de la manière suivante :

```
$(document).ready(function() {
  $('tr:nth-child(even)').addClass('alt');
});
```

Supposons que, pour une raison ou pour une autre, nous souhaitions mettre en exergue les cellules de la table qui font référence à l'une des pièces HENRY. Pour cela, après avoir ajouté une classe à la feuille de style de manière à placer le texte en gras et en italique (`.highlight {font-weight:bold; font-style: italics;}`), il suffit d'ajouter à notre code jQuery la ligne suivante, fondée sur le sélecteur `:contains()` :

```
$(document).ready(function() {
  $('tr:nth-child(even)').addClass('alt');
  $('td:contains(Henry)').addClass('highlight');
});
```

La Figure 2.6 montre notre jolie table à bandes, dans laquelle les pièces HENRY sont mises en exergue.

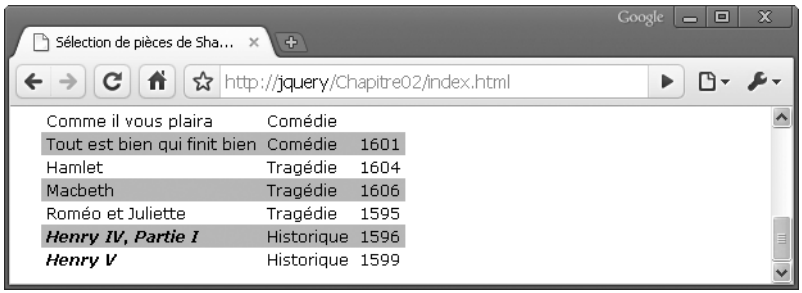


Figure 2.6

Application d'un style à certaines cellules de la table.

Il est important de noter que le sélecteur `:contains()` est sensible à la casse. Si nous avions utilisé `$('td:contains(henry)')`, c'est-à-dire sans le H majuscule, aucune cellule n'aurait été sélectionnée.

Bien évidemment, il existe des solutions pour créer des tables à bandes et mettre du texte en exergue sans passer par jQuery ni utiliser un script côté client. Quoi qu'il en soit, la combinaison de jQuery et de CSS constitue une bonne solution pour mettre en place des styles de ce type lorsque le contenu est généré dynamiquement et que nous n'avons accès ni au contenu HTML ni au code côté serveur.

Sélecteurs pour formulaires

Lorsqu'on manipule des formulaires, les sélecteurs personnalisés de jQuery peuvent simplifier la sélection des éléments recherchés. Le Tableau 2.2 décrit quelques-uns de ces sélecteurs.

Tableau 2.2 : Sélecteurs jQuery pour les formulaires

Sélecteur	*Correspondance*
`:text`, `:checkbox`, `:radio`, `:image`, `:submit`, `:reset`, `:password`, `:file`	Éléments de saisie dont l'attribut de type est égal au nom du sélecteur (sans les deux-points). Par exemple, `:text` sélectionne `<input type="text">`.
`:input`	Éléments de type `input`, `textarea`, `select` et `button`.
`:button`	Éléments `button` et `input` dont l'attribut de type est égal à `button`.
`:enabled`	Éléments de formulaire activés.
`:disabled`	Éléments de formulaire désactivés.
`:checked`	Boutons radio ou cases à cocher sélectionnés.
`:selected`	Éléments `option` sélectionnés.

Comme les autres sélecteurs, il est possible de combiner les sélecteurs pour formulaires afin d'obtenir une meilleure précision. Par exemple, nous pouvons sélectionner tous les boutons radio cochés (non les cases à cocher) avec `$(':radio:checked')` ou sélectionner toutes les zones de saisie de mot de passe et les champs de saisie de texte désactivés avec `$(':password, :text:disabled')`. Avec les sélecteurs personnalisés, nous appliquons les principes de base de CSS pour construire la liste des éléments correspondants.

2.6 Méthodes de parcours du DOM

Les sélecteurs jQuery étudiés jusqu'à présent permettent de sélectionner des éléments dans l'arborescence du DOM, que ce soit *latéralement* ou vers le *bas*, et de filtrer les résultats. S'il s'agissait de la seule manière de sélectionner les éléments, nos possibilités seraient relativement limitées (même si les expressions de sélection sont plutôt robustes, en particulier par rapport aux solutions classiques d'accès au DOM). En de nombreuses occasions, il est nécessaire de sélectionner un élément *parent* ou un élément *ancêtre*. C'est à ce moment-là que les méthodes jQuery pour le parcours du DOM entrent en scène. Elles permettent de parcourir le DOM dans toutes les directions.

Pour certaines méthodes, l'expression de sélection s'écrit de manière quasi identique. Par exemple, la ligne `$('tr:odd').addClass('alt');` employée initialement pour ajouter la classe `alt` peut être récrite avec la méthode `.filter()` :

```
$('tr').filter(':odd').addClass('alt');
```

Cependant, en général, les deux manières de sélectionner les éléments se complètent. De plus, la méthode `.filter()` est particulièrement puissante car elle peut prendre une fonction en argument. Cette fonction permet de créer des tests complexes pour le choix des éléments qui feront partie du jeu correspondant. Par exemple, supposons que nous voulions ajouter une classe à tous les liens externes. jQuery ne propose aucun sélecteur pour répondre à ce besoin. Sans une fonction de filtre, nous serions obligés de parcourir explicitement chaque élément et de les tester un par un. En revanche, grâce à la *fonction de filtre*, nous pouvons nous appuyer sur l'itération implicite de jQuery et garder un code concis :

```
$('a').filter(function() {
  return this.hostname && this.hostname != location.hostname;
}).addClass('external');
```

La deuxième ligne applique un filtre à deux critères sur la collection d'éléments `<a>` :

1. Ils doivent posséder un attribut `href` avec un nom de domaine (`this.hostname`). Ce test permet d'exclure les liens `mailto` et ceux d'autres espèces.

2. Le nom de domaine ciblé par le lien (à nouveau `this.hostname`) ne doit pas correspondre (`!=`) au nom de domaine de la page courante (`location.hostname`).

Plus précisément, la méthode `.filter()` itère sur le jeu d'éléments correspondants, en testant la valeur de retour de la fonction appliquée à chacun. Si elle retourne `false`, l'élément est retiré de la collection obtenue. Si elle retourne `true`, il est conservé.

Revenons à présent à notre table à bandes pour voir si nous ne pourrions pas exploiter les méthodes de parcours.

Appliquer un style à certaines cellules

Précédemment, nous avons ajouté la classe `highlight` à toutes les cellules qui contiennent le texte Henry. Pour appliquer un style à la cellule qui suit chaque cellule contenant ce nom, nous pouvons partir du sélecteur écrit et lui enchaîner la méthode `next()` :

```
$(document).ready(function() {
  $('td:contains(Henry)').next().addClass('highlight');
});
```

La Figure 2.7 illustre la nouvelle présentation de la table.

Figure 2.7

Application d'un style aux cellules qui suivent certaines autres cellules.

La méthode `.next()` sélectionne uniquement l'élément frère suivant. Pour mettre en exergue toutes les cellules qui viennent après celle qui contient Henry, nous pouvons employer la méthode `.nextAll()` :

```
$(document).ready(function() {
  $('td:contains(Henry)').nextAll().addClass('highlight');
});
```

INFO

Comme nous pouvions le supposer, les méthodes `.next()` et `.nextAll()` ont leurs homologues `.prev()` et `.prevAll()`. Par ailleurs, la méthode `.siblings()` sélectionne tous les autres éléments au même niveau du DOM, qu'ils viennent avant ou après l'élément précédemment sélectionné.

Pour inclure la cellule d'origine (celle qui contient Henry) avec les cellules qui suivent, nous pouvons ajouter la méthode `.andSelf()` :

```
$(document).ready(function() {
  $('td:contains(Henry)').nextAll().andSelf().addClass('highlight');
});
```

Vous le constatez, il existe différentes manières de combiner les sélecteurs et les méthodes de parcours pour sélectionner la même collection d'éléments. Voici par exemple une autre façon de sélectionner chaque cellule d'une ligne dont au moins l'une contient le nom Henry :

```
$(document).ready(function() {
  $('td:contains(Henry)').parent().children().addClass('highlight');
});
```

Dans ce cas, au lieu de parcourir transversalement les éléments frères, nous remontons d'un niveau dans le DOM avec `.parent()`, vers l'élément `<tr>`, puis nous sélectionnons toutes les cellules de la ligne avec `.children()`.

Enchaîner des méthodes

Les combinaisons de méthodes de parcours que nous venons de présenter illustrent les possibilités de *chaînage* de jQuery. Avec jQuery, il est possible de sélectionner plusieurs jeux d'éléments et de leur appliquer plusieurs opérations, le tout en une seule ligne de code. Ce chaînage permet non seulement de garder un code JavaScript concis, mais également d'améliorer les performances d'un script lorsque l'alternative est de préciser à nouveau un sélecteur.

Il est aussi possible de découper une même ligne de code en plusieurs lignes de manière à faciliter la lecture. Par exemple, une succession de méthodes peut s'écrire sur une seule ligne :

```
$('td:contains(Henry)').parent().find('td:eq(1)')
    .addClass('highlight').end().find('td:eq(2)')
                            .addClass('highlight');
```

ou sur sept lignes :

```
$('td:contains(Henry)') // Rechercher chaque cellule qui contient "Henry".
.parent()               // Sélectionner son parent.
.find('td:eq(1)')       // Rechercher la deuxième cellule descendante.
.addClass('highlight')  // Ajouter la classe "highlight".
.end()                // Retourner au parent de la cellule qui contient "Henry".
.find('td:eq(2)')       // Rechercher la troisième cellule descendante.
.addClass('highlight'); // Ajouter la classe "highlight".
```

Évidemment, le parcours du DOM dans cet exemple est absurde. Nous ne conseillons pas une telle approche, d'autant qu'il existe des méthodes plus simples et plus directes.

Le but de cet exemple est simplement de démontrer l'extrême souplesse de l'enchaînement des méthodes.

Nous pourrions le comparer à la lecture à haute voix d'un paragraphe sans reprendre son souffle. Cela permet d'aller vite, mais les auditeurs risquent d'avoir du mal à comprendre ce qui est lu. En découpant l'instruction en plusieurs lignes et en ajoutant des commentaires, nous pouvons gagner du temps sur le long terme.

2.7 Accéder aux éléments du DOM

Les expressions de sélection et la plupart des méthodes jQuery retournent un objet jQuery. En général, c'est ce que nous souhaitons, en raison de l'itération implicite et des possibilités de chaînage que cela permet.

Toutefois, il peut arriver que nous ayons besoin d'accéder directement à un *élément du DOM* dans le code. Par exemple, nous pourrions avoir besoin de fournir le jeu d'éléments obtenu à une autre bibliothèque JavaScript. Ou bien nous pourrions avoir besoin d'accéder au nom de balise d'un élément, qui est disponible sous forme de *propriété* de l'élément du DOM. Pour ces cas relativement rares, jQuery propose la méthode `.get()`. Pour accéder au premier élément du DOM référencé par un objet jQuery, nous utilisons `.get(0)`. Si l'accès à l'élément doit se faire dans une boucle, nous pouvons écrire `.get(indice)`. Ainsi, pour connaître le nom de balise d'un élément dont l'identifiant est `mon-element`, nous écrivons :

```
var maBalise = $('#mon-element').get(0).tagName;
```

Pour plus de commodité, jQuery propose un raccourci à `.get()`. Au lieu d'écrire la ligne précédente, nous pouvons utiliser des crochets après le sélecteur :

```
var maBalise = $('#mon-element')[0].tagName;
```

Ne soyez pas surpris par le fait que cette syntaxe ressemble à un tableau d'éléments du DOM. Utiliser les crochets équivaut à retirer l'enveloppe jQuery pour accéder à la liste des nœuds, tandis qu'utiliser l'*indice* (dans ce cas 0) équivaut à récupérer l'élément du DOM.

2.8 En résumé

Grâce aux techniques décrites dans ce chapitre, nous pouvons à présent appliquer un style aux éléments de premier niveau et aux éléments secondaires d'une liste imbriquée en utilisant les *sélecteurs CSS* de base, appliquer différents styles à différents types de liens en utilisant des *sélecteurs d'attribut*, ajouter des bandes à une table en utilisant les *sélecteurs jQuery personnalisés* `:odd` et `:even` ou le sélecteur CSS élaboré `:nth-child()`, et mettre en exergue du texte dans certaines cellules de la table en *chaînant* des méthodes jQuery.

Jusque-là, nous avons employé l'événement `$(document).ready()` pour ajouter une classe aux éléments obtenus. Au chapitre suivant, nous allons voir comment ajouter une classe en réponse aux événements générés par l'utilisateur.

3

Événements

Au sommaire de ce chapitre

- Effectuer des tâches au chargement de la page
- Événements simples
- Événements composés
- Le périple d'un événement
- Modifier le périple : l'objet *event*
- Retirer un gestionnaire d'événements
- Simuler une action de l'utilisateur
- En résumé

En JavaScript, il existe plusieurs méthodes pour réagir aux actions de l'utilisateur et aux autres événements. Nous devons les utiliser de manière à exploiter, au moment opportun, les techniques jQuery décrites précédemment et celles que nous verrons plus loin. Nous obtiendrons ainsi des pages dynamiques et réactives. S'il est parfaitement possible de mettre tout cela en œuvre avec du code JavaScript pur, la bibliothèque jQuery étend et améliore les mécanismes de gestion des événements, en leur donnant une syntaxe plus élégante tout en les rendant plus puissants.

3.1 Effectuer des tâches au chargement de la page

Nous avons déjà vu une solution pour que jQuery réagisse au chargement de la page web. Le gestionnaire d'événement `$(document).ready()` permet de déclencher l'exécution du code d'une fonction, mais les possibilités ne s'arrêtent pas là.

Fixer le moment d'exécution

Au Chapitre 1, nous avons vu que `$(document).ready()` était la solution jQuery pour effectuer des tâches habituellement déclenchées par l'événement `onload` de JavaScript.

Cependant, si les résultats sont équivalents, les actions ne sont pas vraiment exécutées au même moment.

L'événement `window.onload` est déclenché lorsqu'un document est intégralement chargé dans le navigateur, c'est-à-dire lorsque chaque élément de la page est prêt à être manipulé avec JavaScript. Ainsi, nous n'avons pas à tenir compte de l'ordre de chargement des éléments dans le code.

En revanche, un gestionnaire d'événements enregistré avec `$(document).ready()` est invoqué lorsque le DOM est prêt à être utilisé. Cela signifie que tous les éléments sont accessibles depuis les scripts, mais pas nécessairement que chaque fichier référencé ait été téléchargé. Dès que le contenu HTML a été téléchargé et converti en une arborescence DOM, le code peut s'exécuter.

INFO

Pour s'assurer que les styles ont également été appliqués à la page avant que le code JavaScript ne s'exécute, il faut placer les balises `<link rel="stylesheet">` avant les balises `<script>` dans l'élément `<head>` du document.

Par exemple, prenons le cas d'une page qui affiche une galerie de photos. Elle contient potentiellement de nombreuses images volumineuses, qu'il est possible de masquer, afficher, déplacer et manipuler avec jQuery. Si nous fondons l'interface sur l'événement `onload`, les utilisateurs devront attendre que toutes les images soient intégralement chargées avant de pouvoir accéder à la page. Par ailleurs, si les comportements personnalisés ne sont pas encore associés aux éléments qui possèdent des comportements par défaut, comme les liens, les actions de l'utilisateur peuvent provoquer des résultats inattendus. En revanche, si la configuration se fait à partir de `$(document).ready()`, l'interface est disponible beaucoup plus tôt, avec les comportements appropriés.

INFO

De manière générale, il est préférable d'utiliser `$(document).ready()` à la place du gestionnaire `onload`, mais il ne faut pas oublier que, tous les fichiers n'étant pas nécessairement chargés, certains attributs, comme la hauteur et la largeur d'une image, peuvent ne pas être disponibles. Lorsque l'accès à ces données est requis, il faudra mettre en place un gestionnaire `onload` ou, de préférence, utiliser jQuery pour définir un gestionnaire pour l'événement `load` ; les deux mécanismes peuvent parfaitement coexister.

Plusieurs scripts sur une page

Pour enregistrer des gestionnaires d'événements sans les indiquer directement dans le balisage HTML, le mécanisme JavaScript classique consiste à affecter une fonction à

l'attribut correspondant de l'élément du DOM. Par exemple, supposons que nous ayons défini la fonction suivante :

```
function faireAction() {
  // Effectuer une tâche...
}
```

Nous pouvons l'intégrer directement au balisage HTML :

```
<body onload="faireAction();">
```

ou l'affecter à l'événement avec du code JavaScript :

```
window.onload = faireAction;
```

Ces deux solutions permettent d'exécuter la fonction après que la page a été chargée, la seconde ayant l'avantage de séparer clairement le comportement du balisage.

INFO

Lorsqu'une fonction est affectée en tant que gestionnaire, son nom est utilisé sans les parenthèses. Si les parenthèses sont présentes, la fonction est invoquée immédiatement. Sans les parenthèses, le nom identifie simplement la fonction et peut être utilisé pour l'invoquer ultérieurement.

Avec une seule fonction, cette approche fonctionne plutôt bien. Toutefois, supposons que nous ayons défini une seconde fonction :

```
function faireUneAutreAction() {
  // Effectuer une autre tâche...
}
```

Nous pouvons alors tenter d'invoquer cette fonction lors du chargement de la page :

```
window.onload = faireUneAutreAction;
```

Malheureusement, cette affectation annule la première. L'attribut `.onload` ne pouvant enregistrer qu'une seule référence de fonction à la fois, nous ne pouvons pas ajouter un nouveau comportement à celui existant.

En revanche, le mécanisme `$(document).ready()` prend parfaitement en charge ce cas. Chaque appel de la méthode ajoute la nouvelle fonction à une file de comportements interne. Lorsque la page est chargée, toutes les fonctions présentes dans la file sont exécutées dans l'ordre où elles ont été enregistrées.

INFO

Pour être honnête, jQuery n'a pas le monopole des solutions à ce problème. Nous pouvons écrire une fonction JavaScript qui invoque le gestionnaire `onload` existant puis appelle le gestionnaire indiqué. Cette approche, retenue notamment par la fonction `addLoadEvent()`

de Simon Willison, évite les conflits entre les gestionnaires rivaux, comme c'est le cas avec `$(document).ready()`, mais certains des avantages mentionnés lui font défaut. Les méthodes propres aux navigateurs, comme `document.addEventListener()` et `document.attachEvent()`, apportent une fonctionnalité semblable, mais jQuery permet d'arriver au même résultat sans que nous ayons à tenir compte des incompatibilités entre les navigateurs.

Raccourcis pour la concision du code

La construction `$(document).ready()` invoque la méthode `.ready()` sur un objet jQuery obtenu à partir de l'élément `document` du DOM. Cette opération étant très fréquente, la fonction `$()` en propose un raccourci. En l'invoquant sans argument, elle se comporte comme si `document` était passé en paramètre. Autrement dit, nous pouvons écrire :

```
$(document).ready(function() {
  // Notre code...
});
```

de la manière suivante :

```
$().ready(function() {
  // Notre code...
});
```

Par ailleurs, la fonction `$()` accepte une autre fonction en argument, auquel cas jQuery appelle implicitement `.ready()`. Par conséquent, nous pouvons écrire :

```
$(function() {
  // Notre code...
});
```

Bien que ces autres syntaxes soient plus courtes, nous conseillons d'opter pour la version longue car elle montre plus clairement ce que fait le code.

Coexistence avec d'autres bibliothèques

Pour simplifier la mise en œuvre d'une page, il faudra parfois employer plusieurs bibliothèques JavaScript. En raison de sa concision et de sa commodité, elles sont nombreuses à utiliser l'identifiant `$`, entraînant des conflits de noms.

Pour les éviter, jQuery fournit la méthode `.noConflict()`. Elle rend la gestion de l'identifiant `$` aux autres bibliothèques. Voici comment l'utiliser :

```
<script src="prototype.js" type="text/javascript"></script>
<script src="jquery.js" type="text/javascript"></script>
<script type="text/javascript">
  jQuery.noConflict();
</script>
<script src="monscript.js" type="text/javascript"></script>
```

Tout d'abord, l'autre bibliothèque est incluse (Prototype dans cet exemple). Ensuite, jQuery est chargée et s'accapare l'identifiant `$`. Puis l'invocation de `.noConflict()` libère `$` de manière à le redonner à la première bibliothèque (Prototype). Dans notre script, nous pouvons utiliser les deux bibliothèques, mais l'invocation des méthodes jQuery doit se faire avec l'identifiant `jQuery` à la place de `$`.

Pour nous simplifier la vie, la méthode `.ready()` a un autre tour dans sa manche. La fonction de rappel passée en argument peut prendre un seul paramètre : l'objet jQuery lui-même. Nous pouvons ainsi le renommer `$` sans craindre les conflits :

```
jQuery(document).ready(function($) {
  // Dans le code suivant, nous pouvons utiliser $ normalement !
});
```

Ou, avec la syntaxe abrégée décrite précédemment :

```
jQuery(function($) {
  // Code qui utilise $.
});
```

3.2 Événements simples

Nous venons de voir comment effectuer une tâche après le chargement de la page, mais il existe bien d'autres moments où réaliser des opérations. Tout comme nous pouvons intercepter l'événement de chargement d'une page avec `<body onload="">` ou `window.onload`, JavaScript permet de réagir aux événements provenant des utilisateurs, comme les clics de souris (`onclick`), la modification des champs d'un formulaire (`onchange`) ou le redimensionnement de la taille des fenêtres (`onresize`). Lorsqu'ils sont affectés directement aux éléments dans le DOM, ces gestionnaires présentent les inconvénients déjà mentionnés pour `onload`. jQuery propose une meilleure solution pour gérer ces événements.

Un sélecteur de style simple

Pour servir d'illustration à certaines techniques de gestion des événements, nous allons offrir à l'internaute la possibilité de modifier l'aspect d'une page. Il pourra cliquer sur des boutons permettant de passer en vue normale, dans une vue où le texte est affiché dans une colonne étroite et dans une vue où la zone de contenu est affichée avec des caractères plus grands.

Dans un exemple réel, le développeur web sérieux utiliserait ici le principe d'*amélioration progressive*. Le sélecteur de style serait caché lorsque JavaScript est indisponible ou, mieux encore, utiliserait des liens vers des versions alternatives de la page. Pour les besoins de ce didacticiel, nous supposerons que tous les utilisateurs ont activé JavaScript.

Voici le balisage HTML du sélecteur de style :

```
<div id="switcher">
  <h3>Sélecteur de style</h3>
  <div class="button selected" id="switcher-default">
    Par défaut
  </div>
  <div class="button" id="switcher-narrow">
    Colonne étroite
  </div>
  <div class="button" id="switcher-large">
    Grande police
  </div>
</div>
```

La Figure 3.1 présente la page résultante, avec le contenu HTML restant et quelques éléments de style CSS.

Figure 3.1

La présentation initiale de page.

Nous allons commencer par rendre opérationnel le bouton GRANDE POLICE. Pour obtenir la nouvelle présentation de la page, nous définissons un nouveau style CSS :

```
body.large .chapter {
  font-size: 1.5em;
}
```

L'idée est ensuite d'appliquer la classe `large` à la balise `<body>` pour que l'aspect de la page soit modifié conformément à la feuille de style. Avec les connaissances acquises au Chapitre 2, nous savons déjà comment procéder :

```
$('body').addClass('large');
```

Toutefois, nous voulons que cela se produise suite au clic sur le bouton GRANDE POLICE, non après le chargement de la page. Pour cela, nous allons utiliser la méthode `.bind()`. Elle permet d'indiquer n'importe quel *événement JavaScript* et de lui *associer* un comportement. Dans notre exemple, l'événement se nomme `click` et le comportement est une fonction constituée de la ligne précédente :

```
$(document).ready(function() {
  $('#switcher-large').bind('click', function() {
    $('body').addClass('large');
  });
});
```

À présent, si nous cliquons sur le bouton GRANDE POLICE, notre code est exécuté et le texte s'affiche dans une police de caractères plus grande (voir Figure 3.2).

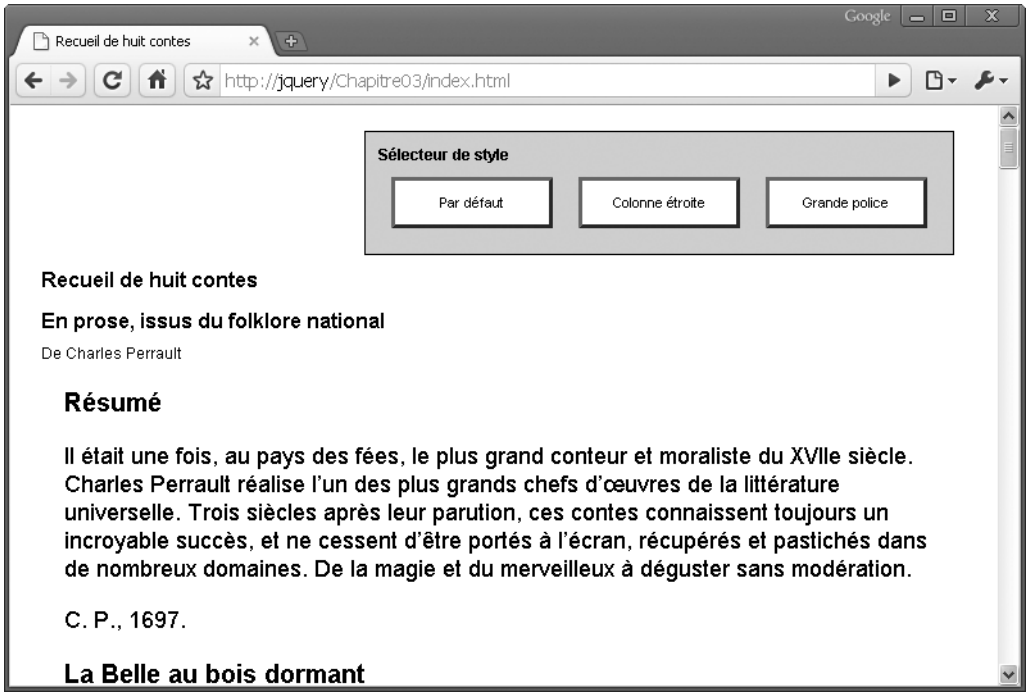


Figure 3.2

Passer la page dans une grande police.

La liaison d'un événement n'est pas plus compliquée que cela. Les avantages mentionnés pour la méthode `.ready()` s'appliquent également dans ce cas. Il est possible d'invoquer plusieurs fois `.bind()` pour ajouter de nouveaux comportements au même événement.

Toutefois, cette solution n'est pas nécessairement la plus élégante ni la plus efficace. Au cours de ce chapitre, nous étendrons et améliorerons ce code, jusqu'à obtenir une version dont nous pourrons être fiers.

Activer les autres boutons

Le bouton GRANDE POLICE fonctionne à présent comme voulu. Nous allons donc appliquer un traitement semblable aux deux autres boutons (PAR DÉFAUT et COLONNE ÉTROITE) pour qu'ils réalisent leur propre opération. Cela n'a rien de complexe : nous utilisons `.bind()` pour enregistrer un gestionnaire de `click`, en ajoutant et en retirant les classes comme nécessaire. Voici le nouveau code :

```
$(document).ready(function() {
  $('#switcher-default').bind('click', function() {
    $('body').removeClass('narrow');
    $('body').removeClass('large');
  });
  $('#switcher-narrow').bind('click', function() {
    $('body').addClass('narrow');
    $('body').removeClass('large');
  });
  $('#switcher-large').bind('click', function() {
    $('body').removeClass('narrow');
    $('body').addClass('large');
  });
});
```

Il est combiné à une règle CSS pour la classe `narrow` :

```
body.narrow .chapter {
  width: 400px;
}
```

Désormais, en cliquant sur le bouton COLONNE ÉTROITE, le style CSS correspondant est appliqué et le texte est affiché différemment (voir Figure 3.3).

En cliquant sur le bouton PAR DÉFAUT, les deux classes sont retirées de la balise `<body>` et la page revient dans sa présentation initiale.

Contexte d'un gestionnaire d'événements

Notre sélecteur fonctionne correctement, mais l'utilisateur ne sait pas quel bouton est actif. Pour proposer ce retour visuel, nous allons appliquer la classe `selected` au bouton sur lequel l'utilisateur a cliqué et la retirer aux deux autres boutons. La classe `selected` met simplement le texte du bouton en gras :

Figure 3.3

Afficher le texte de la page dans une colonne étroite.

```
.selected {
  font-weight: bold;
}
```

Pour modifier la classe, nous pouvons procéder comme précédemment, en faisant référence à chaque bouton par son identifiant et en appliquant ou en retirant les classes, mais nous allons examiner une solution plus élégante et évolutive qui se fonde sur le *contexte* d'exécution des gestionnaires d'événements.

Lorsqu'un gestionnaire d'événements est invoqué, le mot clé `this` fait référence à l'élément du DOM auquel le comportement a été associé. Nous l'avons mentionné précédemment, la fonction `$()` peut prendre en argument un élément du DOM ; c'est d'ailleurs son utilisation principale. En écrivant `$(this)` dans le gestionnaire d'événements, nous créons un objet jQuery qui correspond à l'élément ciblé et nous pouvons le manipuler comme si nous l'avions obtenu à l'aide d'un sélecteur CSS.

Nous pouvons donc écrire le code suivant :

```
$(this).addClass('selected');
```

Si nous plaçons cette ligne dans chacun des trois gestionnaires, la classe est ajoutée dès que l'utilisateur clique sur un bouton. Pour retirer la classe des autres boutons, nous pouvons tirer profit de l'itération implicite de jQuery :

```
$('#switcher .button').removeClass('selected');
```

Cette ligne retire la classe de tous les boutons présents dans le sélecteur de style. En plaçant les instructions dans l'ordre adéquat, nous obtenons le code suivant :

```
$(document).ready(function() {
  $('#switcher-default').bind('click', function() {
    $('body').removeClass('narrow');
    $('body').removeClass('large');
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-narrow').bind('click', function() {
    $('body').addClass('narrow');
    $('body').removeClass('large');
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-large').bind('click', function() {
    $('body').removeClass('narrow');
    $('body').addClass('large');
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```

La Figure 3.4 montre que le sélecteur de style indique à présent le bouton actif.

En généralisant le contexte du gestionnaire à toutes les instructions, nous pouvons être plus efficaces. Le code de mise en exergue peut être extrait dans un gestionnaire séparé car il est identique pour les trois boutons :

```
$(document).ready(function() {
  $('#switcher-default').bind('click', function() {
    $('body').removeClass('narrow').removeClass('large');
  });
  $('#switcher-narrow').bind('click', function() {
    $('body').addClass('narrow').removeClass('large');
  });
  $('#switcher-large').bind('click', function() {
    $('body').removeClass('narrow').addClass('large');
  });
  $('#switcher .button').bind('click', function() {
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```

Cette optimisation tire parti des trois caractéristiques de jQuery que nous avons étudiées. Premièrement, l'*itération implicite* permet de lier le même gestionnaire de `click`

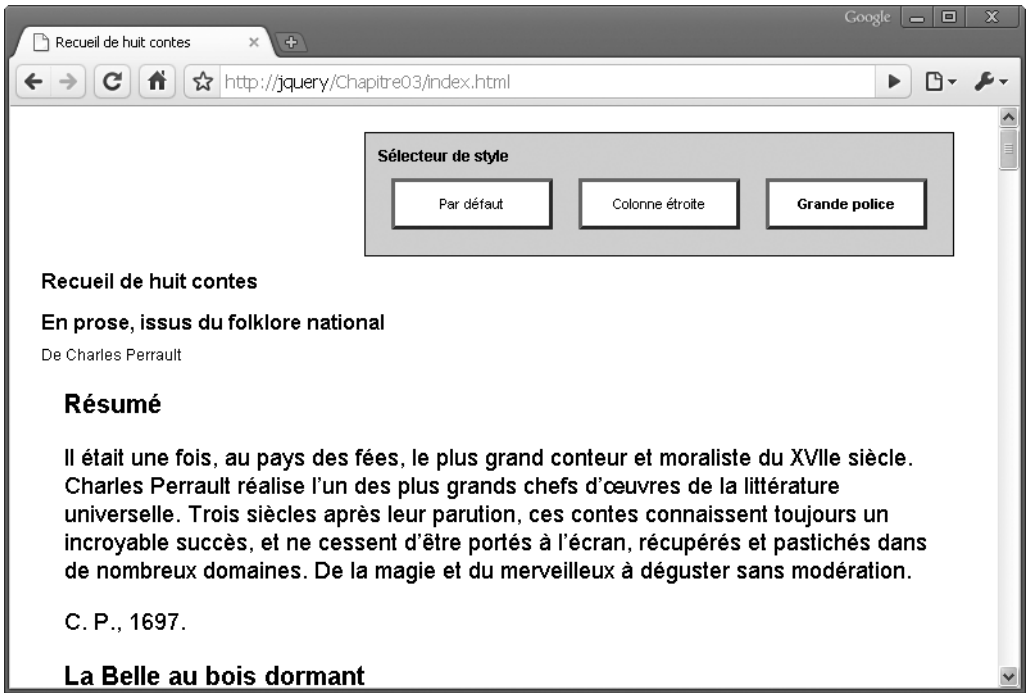


Figure 3.4

Le bouton sur lequel l'utilisateur a cliqué est indiqué en gras.

à chaque bouton par un seul appel à `.bind()`. Deuxièmement, la *file des comportements* permet de lier deux fonctions au même événement `click`, sans que le second remplace le premier. Troisièmement, nous utilisons les possibilités de *chaînage* de jQuery pour regrouper l'ajout et la suppression des classes dans une seule ligne de code.

Autre consolidation

L'optimisation du code que nous venons d'effectuer est un exemple de *remaniement*, c'est-à-dire une modification du code permettant de réaliser la même tâche de manière plus efficace et/ou plus élégante. Afin d'illustrer d'autres opportunités de remaniement, examinons les comportements associés à chaque bouton. L'argument de la méthode `.removeClass()` est facultatif. Lorsqu'il est absent, la méthode retire toutes les classes de l'élément. Nous pouvons alléger notre code en exploitant cette possibilité de la manière suivante :

```
$(document).ready(function() {
  $('#switcher-default').bind('click', function() {
    $('body').removeClass();
  });
```

```
    $('#switcher-narrow').bind('click', function() {
      $('body').removeClass().addClass('narrow');
    });
    $('#switcher-large').bind('click', function() {
      $('body').removeClass().addClass('large');
    });
    $('#switcher .button').bind('click', function() {
      $('#switcher .button').removeClass('selected');
      $(this).addClass('selected');
    });
  });
```

L'ordre des opérations a légèrement changé pour tenir compte de la suppression plus générale des classes. Nous devons commencer par invoquer `.removeClass()` afin qu'elle n'annule pas les effets de l'appel à `.addClass()`.

ATTENTION

Nous pouvons retirer toutes les classes sans inquiétude car le contenu HTML est également sous notre responsabilité. Si vous écrivez du code en vue de sa réutilisation, par exemple sous forme de plugin, vous devez tenir compte des classes qui peuvent exister et les conserver.

Chaque gestionnaire de bouton exécute désormais un code semblable. Nous pouvons facilement l'extraire dans notre gestionnaire général de clic sur un bouton :

```
$(document).ready(function() {
  $('#switcher .button').bind('click', function() {
    $('body').removeClass();
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
  $('#switcher-narrow').bind('click', function() {
    $('body').addClass('narrow');
  });
  $('#switcher-large').bind('click', function() {
    $('body').addClass('large');
  });
});
```

Vous aurez remarqué que le gestionnaire général est placé avant les gestionnaires plus spécifiques. La méthode `.removeClass()` doit être invoquée avant `.addClass()`. Nous pouvons le certifier car jQuery appelle toujours les gestionnaires d'événements dans l'ordre de leur enregistrement.

Enfin, nous pouvons totalement nous débarrasser des gestionnaires spécifiques en tirant profit du *contexte d'événement*. Puisque le mot clé `this` nous retourne un élément du DOM à la place d'un objet jQuery, nous pouvons nous servir des propriétés natives du DOM pour déterminer l'identifiant de l'élément sur lequel l'utilisateur a cliqué. Nous pouvons ainsi lier le même gestionnaire à tous les boutons et y effectuer les différentes actions :

```
$(document).ready(function() {
  $('#switcher .button').bind('click', function() {
    $('body').removeClass();
    if (this.id == 'switcher-narrow') {
      $('body').addClass('narrow');
    }
    else if (this.id == 'switcher-large') {
      $('body').addClass('large');
    }
    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```

Raccourcis pour les événements

L'association d'un gestionnaire à un événement, par exemple un simple événement `click`, est une opération si fréquente que jQuery propose une solution plus concise pour la réaliser. Les *méthodes abrégées pour les événements* opèrent de la même manière que leurs équivalents `.bind()`, mais demandent une saisie moindre.

Par exemple, notre sélecteur de style peut invoquer la méthode `.click()` à la place de `.bind()` :

```
$(document).ready(function() {
  $('#switcher .button').click(function() {
    $('body').removeClass();
    if (this.id == 'switcher-narrow') {
      $('body').addClass('narrow');
    }
    else if (this.id == 'switcher-large') {
      $('body').addClass('large');
    }

    $('#switcher .button').removeClass('selected');
    $(this).addClass('selected');
  });
});
```

Le Tableau 3.1 recense les différentes méthodes abrégées pour les événements. Chacune associe un gestionnaire à l'événement éponyme.

Tableau 3.1 : Les méthodes abrégées pour les événements

`blur`	`error`	`keyup`	`mouseout`	`scroll`
`change`	`focus`	`load`	`mouseover`	`select`
`click`	`keydown`	`mousedown`	`mouseup`	`submit`
`dblclick`	`keypress`	`mousemove`	`resize`	`unload`

3.3 Événements composés

La plupart des méthodes jQuery pour la gestion des événements correspondent directement aux événements JavaScript natifs. Toutefois, quelques gestionnaires personnalisés ont été ajoutés pour des raisons de commodité et d'optimisation entre navigateurs. Nous avons déjà rencontré l'un d'eux : `.ready()`. Les méthodes `.toggle()` et `.hover()` en sont deux autres ; il s'agit de *gestionnaires d'événements composés* car elles interceptent une combinaison d'actions de l'utilisateur et y répondent en utilisant plusieurs fonctions.

Afficher et masquer des fonctions avancées

Supposons que nous souhaitions pouvoir masquer le sélecteur de style lorsqu'il n'est pas utile. Pour masquer des fonctionnalités avancées, une solution commode consiste à les rendre escamotables. Nous allons faire en sorte qu'un clic sur le libellé masque les boutons et ne laisse affiché que le libellé. Un autre clic sur le libellé réaffichera les boutons. Nous avons besoin d'une nouvelle classe pour les boutons masqués :

```
.hidden {
  display: none;
}
```

Pour mettre en place cette fonctionnalité, nous pourrions enregistrer l'état courant des boutons dans une variable et vérifier la valeur de celle-ci à chaque clic sur le libellé de manière à savoir si la classe `hidden` doit être ajoutée ou retirée aux boutons. Nous pourrions également vérifier directement l'existence de cette classe sur un bouton et nous servir de cette information pour décider de l'action à mener. Cependant, jQuery propose la méthode `.toggle()`, qui prend tout cela en charge à notre place.

INFO

jQuery définit en réalité deux méthodes `.toggle()`. Pour de plus amples informations sur la méthode d'*effet* de même nom, qui se distingue par des types d'arguments différents, consultez la page **http://docs.jquery.com/Effects/toggle**.

La méthode d'événement `.toggle()` prend au moins deux arguments, chacun étant une fonction. Le premier clic sur l'élément provoque l'exécution de la première fonction, le deuxième clic, l'exécution de la deuxième fonction, etc. Après que chaque fonction a été invoquée, le cycle reprend à partir de la première. Grâce à `.toggle()`, nous pouvons mettre en œuvre notre sélecteur de style escamotable assez facilement :

```
$(document).ready(function() {
  $('#switcher h3').toggle(function() {
    $('#switcher .button').addClass('hidden');
  }, function() {
```

```
    $('#switcher .button').removeClass('hidden');
  });
});
```

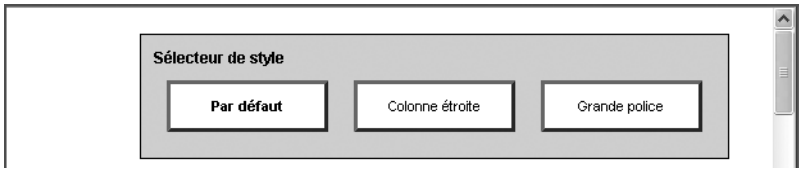
Après un premier clic, tous les boutons sont masqués (voir Figure 3.5), un second les affiche à nouveau (voir Figure 3.6).

Figure 3.5

Un premier clic sur le libellé masque les boutons.

Figure 3.6

Un second clic sur le libellé réaffiche les boutons.



Nous avons à nouveau exploité l'itération implicite, cette fois-ci pour masquer tous les boutons en une seule fois sans avoir besoin d'un élément englobant.

Pour ce type d'escamotage, jQuery propose un autre mécanisme. La méthode `.toggleClass()` permet de vérifier automatiquement l'existence de la classe avant de l'appliquer ou de la retirer :

```
$(document).ready(function() {
  $('#switcher h3').click(function() {
    $('#switcher .button').toggleClass('hidden');
  });
});
```

Dans notre cas, `.toggleClass()` constitue probablement la solution la plus élégante. `.toggle()` est une méthode plus générale pour effectuer alternativement deux actions différentes ou plus.

Signaler les éléments cliquables

En utilisant l'événement `click` avec des éléments de la page normalement non cliquables, nous avons fabriqué une interface qui donne peu d'indications sur le fait que les boutons, qui ne sont en réalité que des éléments `<div>`, constituent la partie *active* de la page qui attend une action de l'utilisateur. Pour remédier à ce défaut, nous pouvons associer un effet de survol aux boutons de manière à indiquer clairement qu'ils peuvent réagir au clic :

```
#switcher .hover {
  cursor: pointer;
  background-color: #afa;
}
```

La spécification CSS mentionne la pseudo-classe nommée `:hover`. Elle permet de changer le style d'un élément lorsque le pointeur de la souris le survole. Dans Internet Explorer 6, cette fonctionnalité se borne aux liens et nous ne pouvons donc pas l'utiliser pour d'autres éléments dans un code multinavigateur. En revanche, jQuery nous permet d'utiliser JavaScript pour modifier le style d'un élément et, évidemment, d'effectuer n'importe quelle action, lorsque le pointeur de la souris arrive sur l'élément et lorsqu'il en sort.

À l'instar de `.toggle()` dans notre exemple précédent, la méthode `.hover()` prend deux fonctions en argument. La première est exécutée lorsque le pointeur de la souris entre dans l'élément sélectionné, la seconde, lorsqu'il le quitte. Pour obtenir l'effet de survol, nous pouvons modifier à ces moments-là les classes appliquées aux boutons :

```
$(document).ready(function() {
  $('#switcher .button').hover(function() {
    $(this).addClass('hover');
  }, function() {
    $(this).removeClass('hover');
  });
});
```

Nous employons à nouveau l'itération implicite et le contexte d'événement pour garder un code simple et concis. Lorsque le pointeur de la souris survole un bouton, notre classe lui est appliquée et produit l'affichage illustré à la Figure 3.7.

Figure 3.7

Le survol d'un bouton met celui-ci en exergue.

En utilisant `.hover()`, nous évitons également les problèmes associés à la propagation des événements en JavaScript. Pour les comprendre, nous devons examiner la façon dont JavaScript décide de l'élément qui traite un événement donné.

3.4 Le périple d'un événement

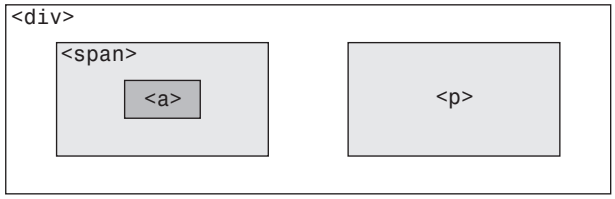
Lorsqu'un événement se produit sur une page, tous les éléments de l'arborescence du DOM ont l'opportunité de le prendre en charge. Prenons l'extrait de page suivant :

```
<div class="foo">
  <span class="bar">
    <a href="http://www.example.com/">
      Zoe ma grande fille veut que je boive ce whisky dont je ne veux pas.
    </a>
  </span>
  <p>
    Voyez ce bon fakir moqueur pousser un wagon en jouant du xylophone.
  </p>
</div>
```

La Figure 3.8 représente ce balisage sous forme d'éléments imbriqués.

Figure 3.8

L'imbrication des éléments de la page.

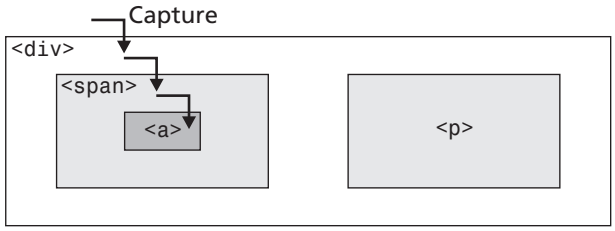


Quel que soit l'événement, plusieurs éléments peuvent logiquement être responsables de sa prise en charge. Par exemple, le clic sur le lien de la page peut être traité par les éléments `<div>`, `<span>` ou `<a>`. En effet, ils sont tous les trois sous le pointeur de la souris en même temps. En revanche, l'élément `<p>` ne fait pas partie de l'interaction.

Pour permettre à plusieurs éléments de répondre au clic, une stratégie consiste à *capturer l'événement*. Avec cette approche, l'événement est tout d'abord passé à l'élément le plus englobant puis, successivement, aux éléments internes. Dans notre exemple, cela signifie que le `<div>` reçoit en premier l'événement, puis le `<span>` et finalement le `<a>`. La Figure 3.9 illustre cette stratégie.

Figure 3.9

Principe de la capture de l'événement.



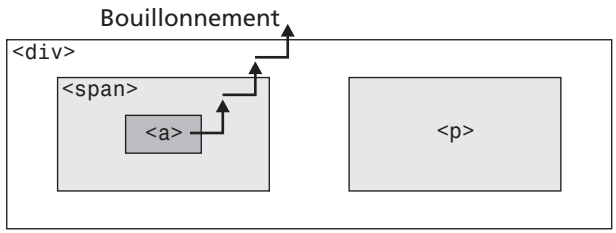
INFO

Techniquement, dans la mise en œuvre de la capture d'événements par les navigateurs, chaque élément *s'enregistre* en tant qu'auditeur des événements qui surviennent dans leurs descendants. L'approximation faite ici suffit à nos besoins.

La stratégie opposée se nomme *bouillonnement de l'événement*. L'événement est envoyé à l'élément le plus interne et, après que celui-ci a eu l'opportunité de le traiter, l'événement remonte vers les éléments externes. Dans notre exemple, le `<a>` traiterait en premier l'événement, puis ce serait au tour du `<span>` et enfin du `<div>` (voir Figure 3.10).

Figure 3.10

Principe du bouillonnement de l'événement.



Les développeurs des différents navigateurs ont, sans surprise, choisi des modèles différents pour la propagation des événements. La spécification du DOM, qui a fini par être rédigée, stipule que les deux stratégies doivent être employées : tout d'abord, l'événement est *capturé* par les éléments externes et transmis aux éléments internes, puis l'événement *bouillonne* vers le sommet de l'arborescence du DOM. Les gestionnaires d'événements peuvent être enregistrés dans l'une des phases du processus.

Tous les navigateurs n'ont pas été mis à jour pour se conformer à ce nouveau standard et, pour ceux qui prennent en charge la capture, elle doit généralement être activée explicitement. Par conséquent, pour assurer la compatibilité entre les navigateurs, jQuery enregistre toujours les gestionnaires d'événements dans la phase de remontée. Nous pouvons donc toujours supposer que l'élément le plus interne a en premier l'opportunité de répondre à n'importe quel événement.

Effets secondaires du bouillonnement d'événement

La remontée d'événement peut provoquer des comportements inattendus, en particulier lorsque le mauvais élément répond à un `mouseover` ou à un `mouseout`. Prenons le cas d'un gestionnaire d'événements `mouseout` associé au `<div>` de notre exemple. Lorsque le pointeur de la souris quitte le `<div>`, le gestionnaire de `mouseout` est invoqué, comme attendu. Puisqu'il se trouve au sommet de la hiérarchie, aucun autre élément ne reçoit

l'événement. En revanche, lorsque le pointeur quitte l'élément `<a>`, un événement `mouseout` lui est passé. Cet événement remonte ensuite jusqu'au `<span>` et au `<div>`, déclenchant l'invocation du même gestionnaire. Cette séquence de remontée n'est pas vraiment souhaitée. Dans le cas des boutons de notre sélecteur de style, cela pourrait conduire à la désactivation prématurée de la mise en exergue.

La méthode `.hover()` tient compte de ces problèmes de remontée et, lorsque nous l'utilisons pour associer des événements, nous pouvons ignorer les problèmes liés à la réception d'un événement `mouseover` ou `mouseout` par le mauvais élément. En cela, `.hover()` est une alternative séduisante à la liaison individuelle des événements de la souris.

INFO

Si une action doit être effectuée uniquement lorsque le pointeur de la souris entre dans un élément ou en sort, mais pas les deux, vous pouvez lier les événements jQuery `mouseenter` et `mouseleave`, qui évitent également les problèmes du bouillonnement. Toutefois, puisque ces événements sont souvent utilisés par paire, la méthode `.hover()` constitue généralement la meilleure solution.

Le scénario `mouseout` que nous venons de décrire illustre le besoin d'une contrainte sur la portée d'un événement. Si la méthode `.hover()` prend en charge ce cas précis, nous rencontrerons d'autres situations dans lesquelles nous devrons borner un événement d'un point de vue spatial (empêcher sa propagation à certains éléments) ou temporel (empêcher sa propagation à certains moments).

3.5 Modifier le périple : l'objet *event*

Nous avons déjà décrit une situation dans laquelle la remontée d'événements peut être source de problèmes. De manière à illustrer un cas où la méthode `.hover()` n'apporte aucune aide, nous allons modifier la procédure d'escamotage mise en œuvre.

Supposons que nous souhaitions étendre la zone cliquable qui déclenche l'escamotage ou l'apparition du sélecteur de style. Pour cela, nous pouvons retirer le gestionnaire d'événements du libellé `<h3>` et l'attribuer à son élément `<div>` englobant :

```
$(document).ready(function() {
  $('#switcher').click(function() {
    $('#switcher .button').toggleClass('hidden');
  });
});
```

Suite à cette modification, l'intégralité de la zone occupée par le sélecteur de style est cliquable et permet de modifier son affichage. Toutefois, en cliquant sur un bouton,

nous masquons également le sélecteur de style après avoir modifié le style du contenu. Ce comportement inattendu vient du bouillonnement de l'événement. Il est tout d'abord traité par les boutons, puis il remonte l'arborescence du DOM jusqu'à atteindre l'élément `<div id="switcher">`, sur lequel notre nouveau gestionnaire est activé, avec pour conséquence de masquer les boutons.

Pour résoudre ce problème, nous devons accéder à l'*objet* `event`. Il s'agit d'une construction JavaScript passée à tout gestionnaire d'événements d'un élément lors de son invocation. Il fournit différentes informations concernant l'événement, comme la position du pointeur de la souris au moment de l'événement. Certaines de ses méthodes permettent également d'intervenir sur le périple de l'événement dans le DOM.

Pour utiliser l'objet `event` dans nos gestionnaires, nous devons simplement ajouter un paramètre à la fonction :

```
$(document).ready(function() {
  $('#switcher').click(function(event) {
    $('#switcher .button').toggleClass('hidden');
  });
});
```

Cibles d'un événement

L'objet `event` est à présent disponible dans le gestionnaire, par l'intermédiaire de la variable `event`. La propriété `event.target` permet de contrôler la cible d'un événement. Elle fait partie de l'API du DOM, mais elle n'est pas disponible dans tous les navigateurs ; jQuery modifie l'objet `event` pour que cette propriété existe dans tous les navigateurs. Grâce à `.target`, nous pouvons déterminer l'élément du DOM qui était le premier à recevoir l'événement. Autrement dit, dans le cas d'un événement `click`, l'élément sur lequel l'utilisateur a cliqué. Puisque nous pouvons ainsi connaître l'élément du DOM qui traite l'événement, nous écrivons le code suivant :

```
$(document).ready(function() {
  $('#switcher').click(function(event) {
    if (event.target == this) {
      $('#switcher .button').toggleClass('hidden');
    }
  });
});
```

Il vérifie que l'utilisateur a bien cliqué sur l'élément `<div id="switcher">`, non sur l'un de ses sous-éléments. À présent, un clic sur l'un des boutons n'escamotera pas le sélecteur de style, contrairement à un clic sur l'arrière-plan du sélecteur. Toutefois, un clic sur le libellé `<h3>` n'aura aucun effet car il s'agit également d'un sous-élément. Au lieu de placer la vérification à ce niveau, nous pouvons modifier le comportement des boutons de manière à atteindre notre objectif.

Stopper la propagation d'un événement

L'objet `event` fournit la méthode `.stopPropagation()` pour interrompre totalement le processus de remontée de l'événement. Comme `.target`, cette méthode est une fonctionnalité purement JavaScript, mais sa fiabilité n'est pas garantie sur tous les navigateurs. Toutefois, tant que les gestionnaires d'événements sont enregistrés à l'aide de jQuery, nous pouvons l'employer sans inquiétude.

Nous retirons le test `event.target == this` précédent et ajoutons du code dans les gestionnaires de `click` associés aux boutons :

```
$(document).ready(function() {
  $('#switcher .button').click(function(event) {
    $('body').removeClass();
    if (this.id == 'switcher-narrow') {
      $('body').addClass('narrow');
    }
    else if (this.id == 'switcher-large') {
      $('body').addClass('large');
    }
    $('#switcher .button').removeClass('selected');
      $(this).addClass('selected');
      event.stopPropagation();
  });
});
```

Nous ajoutons évidemment un paramètre à la fonction utilisée pour le traitement de `click` de manière à avoir accès à l'objet `event`. Ensuite, nous invoquons simplement `event.stopPropagation()` pour éviter que d'autres éléments du DOM répondent à l'événement. À présent, les clics sur les boutons sont pris en charge uniquement par les boutons ; les clics sur d'autres éléments du sélecteur de style l'afficheront ou le masqueront.

Actions par défaut

Si notre gestionnaire de l'événement `click` était associé à un lien (`<a>`) à la place d'un `<div>` générique, nous ferions face à un autre problème. Lorsqu'un utilisateur clique sur un lien, le navigateur charge une nouvelle page. Ce comportement ne correspond pas à un *gestionnaire d'événements* tels que ceux décrits précédemment. Il s'agit d'une *action par défaut* pour un clic sur un lien. De manière équivalente, un appui sur la touche Entrée pendant l'édition d'un formulaire déclenche l'événement `submit` sur le formulaire, qui est ensuite envoyé.

Si ces actions par défaut ne sont pas souhaitées, l'invocation de `.stopPropagation()` sur l'événement ne sera d'aucune aide car elles ne font pas partie du flux normal de propagation d'un événement. Pour empêcher que l'événement ne déclenche l'action par défaut, il faut utiliser la méthode `.preventDefault()`.

INFO

L'invocation de `.preventDefault()` est souvent utile après avoir effectué quelques tests sur le contexte de l'événement. Par exemple, au cours de la soumission d'un formulaire, vous pourriez vouloir vérifier que les champs obligatoires sont remplis et, dans la négative, empêcher que l'action par défaut ne soit effectuée. Nous y reviendrons au Chapitre 8.

La propagation des événements et les actions par défaut sont des mécanismes indépendants. L'un peut être arrêté pendant que l'autre se poursuit. Pour stopper les deux, le gestionnaire d'événements doit retourner `false`. Cela équivaut à une invocation de `.stopPropagation()` et de `.preventDefault()` sur l'événement.

Déléguer un événement

Le bouillonnement n'est pas toujours gênant. Nous pouvons même souvent l'utiliser à notre avantage. La *délégation d'événement* est une technique intéressante qui se fonde sur le bouillonnement. Elle nous permet d'utiliser un gestionnaire d'événements d'un seul élément pour effectuer le travail de plusieurs.

INFO

Dans jQuery 1.3, deux nouvelles méthodes, `.live()` et `.die()`, sont apparues. Elles jouent le même rôle que `.bind()` et `.unbind()`, mais elles se fondent sur la délégation d'événement pour bénéficier des avantages décrits dans cette section. Pour de plus amples informations concernant ces méthodes, consultez la page **http://docs.jquery.com/Events/live**.

Dans notre exemple, des gestionnaires de `click` sont associés à seulement trois éléments `<div class="button">`. Que se passerait-il s'ils étaient plus nombreux ? Ce cas se produit plus fréquemment qu'on pourrait le croire. Par exemple, prenez un grand tableau d'informations dans lequel chaque ligne contient un élément interactif ayant besoin d'un gestionnaire de `click`. Grâce à l'itération implicite, il est très facile d'associer tous ces gestionnaires, mais les performances peuvent se dégrader en raison des boucles internes effectuées par jQuery et de la mémoire requise par les gestionnaires.

À la place, nous pouvons lier un seul gestionnaire de `click` à un élément ancêtre et, grâce au bouillonnement, un `click` non interrompu finira par atteindre cet ancêtre, où le traitement requis sera réalisé.

Afin d'illustrer cette technique, appliquons-la à notre sélecteur de style, même si le nombre d'éléments ne l'impose pas. Nous l'avons vu précédemment, la propriété `event.target` nous permet de connaître l'élément qui se trouvait sous le pointeur de la souris au moment du clic :

```
$(document).ready(function() {
  $('#switcher').click(function(event) {
    if ($(event.target).is('.button')) {
      $('body').removeClass();
      if (event.target.id == 'switcher-narrow') {
        $('body').addClass('narrow');
      }
      else if (event.target.id == 'switcher-large') {
        $('body').addClass('large');
      }
      $('#switcher .button').removeClass('selected');
      $(event.target).addClass('selected');
      event.stopPropagation();
    }
  });
});
```

Dans ce code, nous utilisons une nouvelle méthode nommée `.is()`. Elle accepte en argument les *expressions de sélection* décrites au Chapitre 2 et compare l'objet jQuery courant à celui obtenu avec le sélecteur. Si au moins l'un des éléments du jeu correspond au sélecteur, `.is()` retourne `true`. Dans notre code, la construction `$(event.target).is('.button')` vérifie si l'élément sur lequel l'utilisateur a cliqué possède la classe `button`. Dans l'affirmative, le corps de la condition est exécuté, avec une modification importante : le mot clé `this` fait à présent référence au `<div id="switcher">` et, chaque fois que nous voulons faire référence au bouton sur lequel on a cliqué, nous devons le désigner par `event.target`.

INFO

Vous pouvez également vérifier la présence d'une classe sur un élément en utilisant la méthode `.hasClass()`. Toutefois, `.is()` est plus souple et permet de tester n'importe quelle expression de sélection.

Ce code présente cependant un effet secondaire involontaire. Le clic sur un bouton escamote le sélecteur, comme cela se produisait lors de l'ajout de l'appel à `.stopPropagation()`. Le gestionnaire d'affichage du sélecteur est à présent associé au même élément que le gestionnaire des boutons. Par conséquent, l'interruption du bouillonnement n'empêche pas l'exécution de la procédure d'escamotage. Pour résoudre ce problème, nous pouvons retirer l'appel à `.stopPropagation()` et ajouter à la place un autre test `.is()` :

```
$(document).ready(function() {
  $('#switcher').click(function(event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  });
});
```

```
$(document).ready(function() {
  $('#switcher').click(function(event) {
    if ($(event.target).is('.button')) {
      $('body').removeClass();
      if (event.target.id == 'switcher-narrow') {
        $('body').addClass('narrow');
      }
      else if (event.target.id == 'switcher-large') {
        $('body').addClass('large');
      }
      $('#switcher .button').removeClass('selected');
      $(event.target).addClass('selected');
    }
  });
});
```

Cet exemple est un peu complexe pour sa taille. Toutefois, lorsque le nombre d'éléments possédant des gestionnaires d'événements augmente, la délégation d'événement est une technique de choix.

INFO

Vous le verrez, la délégation d'événement est également utile dans d'autres situations, par exemple lorsque de nouveaux éléments sont ajoutés à l'aide des méthodes de manipulation du DOM (voir Chapitre 5) ou des procédures AJAX (voir Chapitre 6).

3.6 Retirer un gestionnaire d'événements

Il arrive parfois qu'un gestionnaire d'événements précédemment enregistré ne soit plus utile. L'état de la page a peut-être évolué et l'action effectuée n'a alors plus de sens. Ce type de situation peut être pris en charge par des instructions conditionnelles dans le gestionnaire d'événements, mais il est sans doute plus élégant de le *retirer* totalement.

Supposons que nous voulions que notre sélecteur de style escamotable reste ouvert lorsque la page ne se trouve pas dans le style normal. Si le bouton COLONNE ÉTROITE ou le bouton GRANDE POLICE est sélectionné, un clic sur l'arrière-plan du sélecteur de style ne doit avoir aucun effet. Pour cela, suite au clic sur l'un des boutons autres que PAR DÉFAUT, nous pouvons invoquer la méthode `.unbind()` de manière à retirer le gestionnaire d'escamotage :

```
$(document).ready(function() {
  $('#switcher').click(function(event) {
    if (!$(event.tsearget).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  });
  $('#switcher-narrow, #switcher-large').click(function() {
    $('#switcher').unbind('click');
  });
});
```

Un clic sur un bouton comme COLONNE ÉTROITE retire ainsi le gestionnaire de `click` sur le `<div>` du sélecteur de style, et un clic sur l'arrière-plan du rectangle ne le masque plus. Toutefois, le bouton n'est plus opérationnel ! En effet, il est également lié à l'événement `click` du `<div>` du sélecteur, car nous avons récrit le code de gestion du bouton en utilisant la délégation d'événement. Autrement dit, l'instruction `$('#switcher').unbind('click')` retire les deux comportements.

Espace de noms d'un événement

Notre invocation de `.unbind()` doit être plus précise afin de ne pas retirer les deux gestionnaires de `click` enregistrés. Pour cela, nous pouvons utiliser les *espaces de noms des événements*. Au moment de la liaison d'un événement, nous pouvons fournir des informations complémentaires qui permettront d'identifier ultérieurement le gestionnaire. Pour utiliser les espaces de noms, nous devons revenir à la méthode non abrégée de liaison des gestionnaires d'événements, c'est-à-dire `.bind()`.

Le premier paramètre passé à `.bind()` correspond au nom de l'événement JavaScript que nous souhaitons surveiller. Nous pouvons opter pour une syntaxe particulière qui permet de créer des sous-catégories d'événements.

```
$(document).ready(function() {
  $('#switcher').bind('click.collapse', function(event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  });
  $('#switcher-narrow, #switcher-large').click(function() {
    $('#switcher').unbind('click.collapse');
  });
});
```

Le suffixe `.collapse` n'est pas interprété par le système de gestion des événements ; les événements `click` sont traités par cette fonction comme si nous avions écrit simplement `.bind('click')`. Toutefois, l'ajout d'un espace de noms nous permet de retirer uniquement le gestionnaire désigné, sans affecter celui associé aux boutons.

INFO

Nous allons le voir, il existe d'autres manières de rendre l'appel à `.unbind()` plus précis. Cependant, les espaces de noms des événements constituent un outil intéressant, en particulier pour la création de *plugins* (voir Chapitre 11).

Lier à nouveau des événements

Un clic sur les boutons COLONNE ÉTROITE ou GRANDE POLICE désactive à présent la fonctionnalité d'escamotage. Toutefois, nous souhaitons que ce comportement soit rétabli

lors d'un clic sur le bouton PAR DÉFAUT. Pour cela, nous devons *lier à nouveau* le gestionnaire suite au clic sur le bouton PAR DÉFAUT.

Tout d'abord, nous nommons la fonction du gestionnaire afin de pouvoir la réutiliser sans avoir à répéter le code :

```
$(document).ready(function() {
  var toggleStyleSwitcher = function(event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  };
  $('#switcher').bind('click.collapse', toggleStyleSwitcher);
});
```

Vous constatez que nous choisissons une nouvelle syntaxe pour la définition de la fonction. Au lieu d'utiliser le mot clé `function`, nous affectons une *fonction anonyme* à une *variable locale*. Grâce à cette méthode, les définitions des gestionnaires d'événements et des autres fonctions sont semblables ; les deux syntaxes sont fonctionnellement équivalentes.

Rappelons également que la méthode `.bind()` prend une *référence de fonction* en second argument. Lorsqu'on utilise une fonction nommée ici, il ne faut pas oublier d'omettre les parenthèses après le nom, car les parenthèses provoqueraient l'invocation de la fonction au lieu d'en passer une référence.

Puisque la fonction est nommée, nous pouvons la lier à nouveau sans répéter sa définition :

```
$(document).ready(function() {
  var toggleStyleSwitcher = function(event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  };
  $('#switcher').bind('click.collapse', toggleStyleSwitcher);
  $('#switcher-narrow, #switcher-large').click(function() {
    $('#switcher').unbind('click.collapse');
  });
  $('#switcher-default').click(function() {
    $('#switcher').bind('click.collapse', toggleStyleSwitcher);
  });
});
```

La procédure d'escamotage est liée au chargement du document, déliée lors d'un clic sur COLONNE ÉTROITE ou GRANDE POLICE et liée à nouveau lors du clic sur PAR DÉFAUT.

Nous avons réussi à éviter un piège potentiel ici. Lorsqu'un gestionnaire est lié à un événement dans jQuery, les gestionnaires précédents restent actifs. Autrement dit, en cas de multiples clics successifs sur le bouton PAR DÉFAUT, plusieurs exemplaires du gestionnaire `toggleStyleSwitcher` seront liés, conduisant à un comportement étrange

lors d'un clic sur le `<div>`. Ce serait effectivement le cas si nous avions utilisé des fonctions anonymes. Toutefois, puisque la fonction est nommée et que nous utilisons le même nom tout au long du code, le comportement n'est lié qu'une seule fois. La méthode `.bind()` n'associe pas un gestionnaire d'événements à un élément s'il y est déjà associé.

Le nommage de la fonction présente un autre intérêt : nous n'avons plus à utiliser les espaces de noms. La méthode `.unbind()` est en mesure de délier un gestionnaire spécifique en passant la fonction correspondante en second argument :

```
$(document).ready(function() {
  var toggleStyleSwitcher = function(event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  };
  $('#switcher').click(toggleStyleSwitcher);
  $('#switcher-narrow, #switcher-large').click(function() {
    $('#switcher').unbind('click', toggleStyleSwitcher);
  });
  $('#switcher-default').click(function() {
    $('#switcher').click(toggleStyleSwitcher);
  });
});
```

Il existe également une version abrégée pour le cas où nous souhaiterions délier un gestionnaire d'événements juste après son premier déclenchement. Voici comment utiliser la méthode `.one()` :

```
$(document).ready(function() {
  $('#switcher').one('click', toggleStyleSwitcher);
});
```

Cela permet d'effectuer l'action une seule fois.

3.7 Simuler une action de l'utilisateur

Parfois, il peut être utile d'exécuter le code que nous avons lié à un événement, même si les circonstances normales de l'événement ne se produisent pas. Par exemple, supposons que nous souhaitions que le sélecteur de style soit initialement escamoté. Pour cela, nous pouvons masquer les boutons en utilisant la feuille de style ou en invoquant la méthode `.hide()` à partir d'un gestionnaire `$(document).ready()`. Il existe cependant une autre solution : simuler un clic sur le sélecteur de style afin que le mécanisme d'escamotage mis en place soit déclenché.

La méthode `.trigger()` nous permet de mettre en œuvre cette solution :

```
$(document).ready(function() {
  $('#switcher').trigger('click');
});
```

Lorsque la page est chargée, le sélecteur est escamoté, comme si l'utilisateur avait cliqué dessus (voir Figure 3.11). Si nous masquions du contenu en voulant une solution compatible avec les navigateurs dans lesquels JavaScript n'est pas activé, cette approche serait une bonne manière de mettre en place une *amélioration progressive*.

Figure 3.11

Escamoter le sélecteur de style en simulant un clic.

La méthode `.trigger()` propose le même jeu de raccourcis que la méthode `.bind()`. Lorsque ces versions abrégées sont utilisées sans argument, l'action est déclenchée au lieu d'être liée :

```
$(document).ready(function() {
  $('#switcher').click();
});
```

Événements du clavier

Dans ce nouvel exemple, nous allons ajouter des raccourcis clavier à notre sélecteur de style. En appuyant sur la touche équivalant à la première lettre d'un style, la page est affichée comme si l'utilisateur avait cliqué sur le bouton correspondant. Pour mettre en œuvre cette fonctionnalité, nous devons examiner les *événements du clavier*, dont le comportement diffère légèrement des *événements de la souris*.

Il existe deux types d'événements du clavier : ceux qui proviennent directement du clavier (`keyup` et `keydown`) et ceux qui proviennent de la saisie d'un texte (`keypress`). Un événement issu de la saisie d'un seul caractère peut correspondre à plusieurs touches, par exemple lorsque la touche Maj est utilisée conjointement à la touche X pour obtenir la lettre X majuscule. Si les détails de l'implémentation varient d'un navigateur à l'autre, voici une règle générale relativement fiable : si vous souhaitez connaître la

touche sur laquelle l'utilisateur a appuyé, vous devez intercepter les événements `keyup` ou `keydown` ; si vous souhaitez connaître le caractère affiché sur l'écran suite à cette action, vous devez auditer l'événement `keypress`. Dans notre cas, puisque nous voulons savoir si l'utilisateur a appuyé sur les touches P, C ou G, nous utiliserons `keyup`.

Ensuite, nous devons déterminer les éléments concernés par l'événement. La réponse est moins évidente que dans le cas des événements de la souris, où le pointeur nous indiquait clairement la cible de l'événement. La cible d'un événement du clavier est l'élément qui détient le *focus du clavier*. Il existe plusieurs manières de changer l'élément qui a le focus, notamment en cliquant avec la souris ou en appuyant sur la touche Tab. Par ailleurs, tous les éléments ne reçoivent pas le focus. Seuls ceux qui possèdent par défaut des comportements orientés clavier, comme les champs de formulaire, les liens et les éléments avec une propriété `.tabIndex`, sont candidats.

Dans notre cas, l'élément qui possède le focus n'a pas réellement d'importance. Nous souhaitons que notre sélecteur réagisse dès que l'utilisateur appuie sur l'une des touches. Le bouillonnement d'événement va encore se révéler pratique, car nous pouvons associer l'événement `keyup` à l'élément `document` en étant certains qu'il finira par arriver jusqu'à nous.

Enfin, nous devons savoir quelle touche a déclenché le gestionnaire de `keyup`. Pour cela, nous pouvons examiner l'objet `event`. La propriété `.keyCode` de l'événement contient un identifiant de la touche appuyée, qui, pour les touches alphabétiques, correspond à la valeur ASCII de la lettre majuscule. Nous pouvons donc nous fonder sur cette valeur pour simuler un clic sur le bouton approprié :

```
$(document).ready(function() {
  $(document).keyup(function(event) {
    switch (String.fromCharCode(event.keyCode)) {
      case 'P':
        $('#switcher-default').click();
        break;
      case 'C':
        $('#switcher-narrow').click();
        break;
      case 'G':
        $('#switcher-large').click();
        break;
    }
  });
});
```

Un appui sur l'une de ces trois touches simule un clic sur le bouton correspondant, à condition que l'événement du clavier ne soit pas interrompu par des fonctionnalités du navigateur comme "la recherche du texte lorsque la saisie débute" mise en œuvre par Firefox.

Au lieu d'utiliser `.trigger()` pour simuler un clic, voyons comment déplacer le code dans une fonction afin que plusieurs gestionnaires puissent l'invoquer – dans notre cas, ceux de `click` et de `keyup`. Bien qu'elle ne s'impose pas dans notre exemple, cette technique permet de réduire la redondance du code.

```
$(document).ready(function() {
  // Activer l'effet de survol sur les boutons du sélecteur de style.
  $('#switcher .button').hover(function() {
    $(this).addClass('hover');
  }, function() {
    $(this).removeClass('hover');
  });

  // Permettre l'affichage et le masquage du sélecteur de style.
  var toggleStyleSwitcher = function(event) {
    if (!$(event.target).is('.button')) {
      $('#switcher .button').toggleClass('hidden');
    }
  };
  $('#switcher').click(toggleStyleSwitcher);

  // Simuler un clic de manière à débuter dans l'état escamoté.
  $('#switcher').click();

  // La fonction setBodyClass() modifie le style de la page.
  // L'état du sélecteur de style est également actualisé.
  var setBodyClass = function(className) {
    $('body').removeClass();
    $('body').addClass(className);
    $('#switcher .button').removeClass('selected');
    $('#switcher-' + className).addClass('selected');
    if (className == 'default') {
      $('#switcher').click(toggleStyleSwitcher);
    }
    else {
      $('#switcher').unbind('click', toggleStyleSwitcher);
      $('#switcher .button').removeClass('hidden');
    }
  };

  // Invoquer setBodyClass() lors du clic sur un bouton.
  $('#switcher').click(function(event) {
    if ($(event.target).is('.button')) {
      if (event.target.id == 'switcher-default') {
        setBodyClass('default');
      }
      if (event.target.id == 'switcher-narrow') {
        setBodyClass('narrow');
      }
      else if (event.target.id == 'switcher-large') {
        setBodyClass('large');
      }
    }
  });
```

```
  // Invoquer setBodyClass() lors de l'appui sur une touche.
  $(document).keyup(function(event) {
    switch (String.fromCharCode(event.keyCode)) {
      case 'P':
        setBodyClass('default');
        break;
      case 'C':
        setBodyClass('narrow');
        break;
      case 'G':
        setBodyClass('large');
        break;
    }
  });
});
```

3.8 En résumé

Les points examinés dans ce chapitre nous permettent plusieurs choses :

- Utiliser plusieurs bibliothèques JavaScript dans une même page en invoquant la méthode `.noConflict()`.
- Définir des *gestionnaires d'événements de la souris* pour réagir au clic effectué par l'utilisateur sur un élément de la page, en utilisant `.bind()` ou `.click()`.
- Examiner le *contexte d'événement* de manière à réaliser des actions différentes selon l'élément de la page sur lequel on a cliqué, même lorsque le gestionnaire est lié à plusieurs éléments.
- Basculer entre l'affichage et le masquage d'un élément de la page, en utilisant la méthode `.toggle()`.
- Mettre en exergue l'élément de la page qui se trouve sous le pointeur de la souris, en utilisant `.hover()`.
- Intervenir sur la *propagation d'un événement* pour choisir les éléments qui traiteront l'événement, en utilisant `.stopPropagation()` et `.preventDefault()`.
- Mettre en œuvre la *délégation d'événement* de manière à réduire le nombre de gestionnaires d'événements liés.
- Invoquer `.unbind()` pour retirer un gestionnaire d'événements lorsqu'il est devenu inutile.
- Isoler des gestionnaires d'événements connexes à l'aide d'un *espace de noms d'événements* afin de les manipuler comme un groupe.
- Déclencher l'exécution de gestionnaires d'événements avec `.trigger()`.

- Employer les *gestionnaires d'événements du clavier* pour réagir aux appuis sur les touches du clavier par l'utilisateur, en utilisant `.keyup()`.

Lorsqu'elles sont combinées, toutes ces possibilités permettent de construire des pages relativement interactives. Au chapitre suivant, nous verrons comment fournir un retour visuel à l'utilisateur au cours de ces interactions.

4

Effets

Au sommaire de ce chapitre

- Modifier les styles CSS
- Bases du masquage/affichage
- Effets et vitesse
- Effets composés
- Créer des animations personnalisées
- Effets simultanés ou séquentiels
- En résumé

Si les images sont plus parlantes que les mots, alors, dans le monde JavaScript, les effets rendent les images encore plus parlantes. Avec les jeux d'*effets* visuels simples de jQuery, nous pouvons facilement donner de l'impact à nos actions, et même construire nos propres *animations* sophistiquées.

Les effets jQuery ajoutent sans conteste une certaine classe à la page, par exemple avec des éléments qui s'affichent progressivement au lieu d'apparaître immédiatement. Mais ils améliorent également l'utilisation de la page, en aidant l'internaute à s'orienter lorsque des modifications y sont apportées, en particulier avec les applications AJAX. Dans ce chapitre, nous examinerons ces effets et les combinerons de manière intéressante.

4.1 Modifier les styles CSS

Avant d'entrer dans les détails des effets jQuery, nous devons étudier rapidement CSS. Dans les chapitres précédents, nous avons modifié l'aspect d'un document en définissant des classes dans une feuille de style séparée et en les ajoutant ou en les retirant ensuite à l'aide de jQuery. Cette forme d'injection des styles CSS dans un contenu HTML est généralement conseillée, car elle limite le rôle de la feuille de style à la présentation d'une page. Toutefois, il peut arriver que nous ayons besoin d'appliquer des

styles qui n'ont pas encore été définis dans une feuille de style, ou qui ne peuvent pas l'être aisément. Pour de tels cas, jQuery propose la méthode `.css()`.

Cette méthode joue le rôle d'*accesseur* (*getter*) et de *mutateur* (*setter*). Pour obtenir la valeur d'une propriété de style, nous passons simplement son nom sous forme d'une chaîne de caractères, par exemple `.css('backgroundColor')`. jQuery est capable d'interpréter les propriétés dont le nom est constitué de plusieurs mots, que ce soit dans leur version avec tiret, comme dans la notation CSS (`background-color`), ou dans leur version avec majuscule au début des mots internes, comme dans la notation du DOM (`backgroundColor`). Pour fixer la valeur des propriétés de style, la méthode `.css()` propose deux variantes. La première prend en argument une seule propriété de style et sa valeur, la seconde, une *mappe* de couples propriété-valeur :

```
.css('propriété','valeur')
.css({propriété1: 'valeur1', 'propriété-2': 'valeur2'})
```

Les programmeurs JavaScript expérimentés auront reconnu dans ces mappes jQuery des *objets littéraux* JavaScript.

INFO

Les valeurs numériques ne sont pas placées entre des apostrophes, contrairement aux chaînes de caractères. Toutefois, pour les mappes, les apostrophes ne sont pas obligatoires autour des noms des propriétés lorsqu'ils sont écrits en utilisant la notation du DOM.

Nous utilisons la méthode `.css()` de la même manière que la méthode `.addClass()`, c'est-à-dire en l'*enchaînant* à un sélecteur et en l'*associant* à un événement. Pour illustrer cela, revenons au sélecteur de style du Chapitre 3, mais avec un balisage HTML différent :

```
<div id="switcher">
  <div class="label">Taille du texte</div>
  <button id="switcher-default">Par défaut</button>
  <button id="switcher-large">Plus grande</button>
  <button id="switcher-small">Plus petite</button>
</div>
<div class="speech">
  <p>Devant Douaumont, en février, mon tour arriva, et je partis. Je suis la
     ligne et répare plusieurs fois jusqu'à la ferme de Thiaumont. Les balles
     de mitrailleuses rasaient le boyau, et j'étais aveuglé et couvert de terre
     par les obus.</p>
</div>
```

Avec quelques règles de style, nous obtenons la page illustrée à la Figure 4.1.

Dans cette version du sélecteur de style, nous utilisons des éléments `<button>`. En cliquant sur les boutons PLUS GRANDE ou PLUS PETITE, nous augmentons ou diminuons la taille du texte contenu dans `<div class="speech">`. Un clic sur le bouton PAR DÉFAUT replace le texte de `<div class="speech">` dans sa taille initiale.

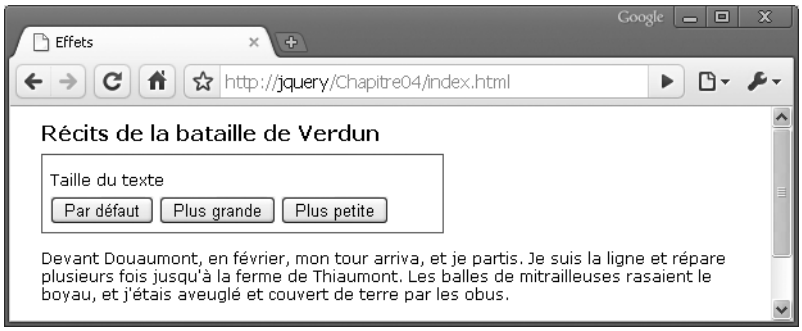


Figure 4.1
Aspect initial de la page.

Si nous voulions simplement fixer la taille de la police à une valeur figée, nous pourrions toujours utiliser la méthode `.addClass()`. Mais nous souhaitons à présent que la taille du texte continue à augmenter ou à diminuer lors de chaque appui sur le bouton correspondant. Même s'il est possible de définir une classe séparée pour chaque clic et de les attribuer au fur et à mesure, une approche plus simple est de calculer à chaque fois la nouvelle taille du texte en obtenant la taille actuelle et en l'augmentant d'un certain facteur, par exemple 40 %.

Notre code commence par les gestionnaires d'événements `$(document).ready()` et `$('#switcher-large').click()` :

```
$(document).ready(function() {
  $('#switcher-large').click(function() {
  });
});
```

Nous pouvons facilement obtenir la taille du texte à l'aide de la méthode `.css()` : `$('div.speech').css('fontSize')`. Néanmoins, puisque la valeur retournée se terminera par "px", nous devons retirer cette partie pour effectuer des calculs avec la valeur. Par ailleurs, lorsqu'il est prévu d'utiliser un objet jQuery plusieurs fois, il est préférable de *mettre en cache* le sélecteur en plaçant l'objet jQuery résultant dans une variable.

```
$(document).ready(function() {
  var $speech = $('div.speech');
  $('#switcher-large').click(function() {
    var num = parseFloat($speech.css('fontSize'));
  });
});
```

La première ligne de `$(document).ready()` enregistre dans une variable l'objet qui correspond au `<div class="speech">`. Vous remarquerez l'utilisation du symbole `$` dans le nom de la variable, `$speech`. Puisque le caractère `$` est accepté dans les variables JavaScript, nous l'utilisons pour indiquer que la variable contient un objet jQuery.

Dans le gestionnaire `.click()`, nous utilisons `parseFloat()` pour obtenir uniquement la valeur numérique de la propriété `fontSize`. Cette fonction examine une chaîne de

gauche à droite jusqu'à ce qu'elle rencontre un caractère non numérique. La suite de chiffres est convertie en nombre à virgule flottante (réel). Par exemple, elle convertit la chaîne "12" dans le nombre 12. Par ailleurs, puisqu'elle retire les caractères non numériques qui se trouvent à la fin de la chaîne, "12px" devient 12. Si la chaîne ne commence pas par un caractère numérique, `parseFloat()` retourne `NaN`, qui signifie *Not a Number* ou *Non un nombre*.

Pour terminer, si nous augmentons la taille de 40 %, il ne nous reste plus qu'à multiplier `num` par `1.4` et à fixer la taille de police en concaténant `num` et `'px'` :

```
$(document).ready(function() {
  var $speech = $('div.speech');
  $('#switcher-large').click(function() {
    var num = parseFloat($speech.css('fontSize') );
    num *= 1.4;
    $speech.css('fontSize', num + 'px');
  });
});
```

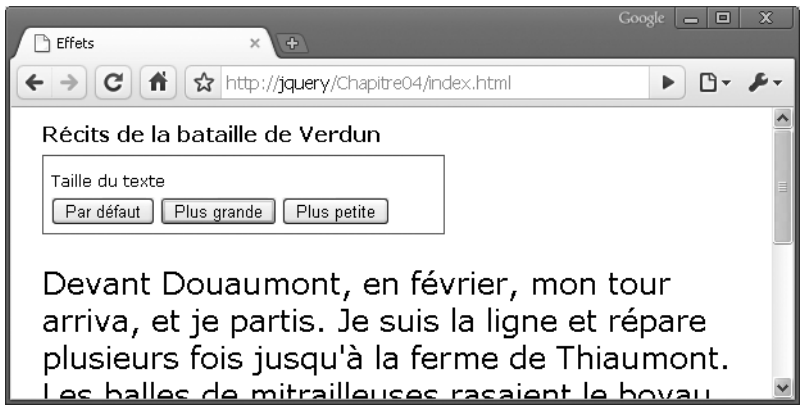
INFO

L'instruction `num *= 1.4` est une version abrégée de `num = num * 1.4`. Nous pouvons utiliser le même type de raccourci avec les autres opérations : addition, `num += 1.4` ; soustraction, `num -= 1.4` ; division, `num /= 1.4` ; et modulo (reste d'une division), `num %= 1.4`.

Si l'utilisateur clique à présent sur le bouton PLUS GRANDE, le texte s'affiche avec une taille de police plus grande. Un autre clic et le texte devient encore plus grand (voir Figure 4.2).

Figure 4.2

Le texte après deux clics sur le bouton PLUS GRANDE.



Pour que le bouton PLUS PETITE diminue la taille de la police, nous divisons la taille courante par `1.4` au lieu de la multiplier par cette valeur. Mais nous allons réunir les deux

opérations dans un même gestionnaire `.click()` associées à tous les éléments `<button>` du `<div id="switcher">`. Après avoir obtenu la valeur numérique de la taille, nous la multiplions ou la divisons en fonction de l'identifiant du bouton sur lequel l'utilisateur a cliqué. Voici la nouvelle version du code :

```
$(document).ready(function() {
  var $speech = $('div.speech');
  $('#switcher button').click(function() {
    var num = parseFloat( $speech.css('fontSize') );
    if (this.id == 'switcher-large') {
      num *= 1.4;
    } else if (this.id == 'switcher-small') {
      num /= 1.4;
    }
    $speech.css('fontSize', num + 'px);
  });
});
```

Au Chapitre 3, nous avons vu que nous pouvions accéder à la propriété `id` de l'élément du DOM référencé par `this`. Nous retrouvons cela dans les instructions `if` et `else if`. Dans cet exemple, il est plus efficace d'utiliser `this` que de créer un objet jQuery uniquement pour tester la valeur d'une propriété.

Nous avons également besoin d'une solution pour redonner sa valeur initiale à la taille de la police. Pour cela, nous pouvons simplement enregistrer cette taille dans une variable dès que le DOM est prêt et l'utiliser ensuite dans la gestion du clic sur le bouton PAR DÉFAUT. Pour cette gestion, nous pouvons ajouter une autre instruction `else if`, mais un `switch` semble plus approprié :

```
$(document).ready(function() {
  var $speech = $('div.speech');
  var defaultSize = $speech.css('fontSize');
  $('#switcher button').click(function() {
    var num = parseFloat( $speech.css('fontSize') );
    switch (this.id) {
      case 'switcher-large':
        num *= 1.4;
        break;
      case 'switcher-small':
        num /= 1.4;
        break;
      default:
        num = parseFloat(defaultSize);
    }
    $speech.css('fontSize', num + 'px');
  });
});
```

Nous consultons la valeur de `this.id` pour changer la taille de la police, mais, si cette valeur n'est pas `'switcher-large'` ou `'switcher-small'`, nous choisissons la taille de police initiale.

4.2 Bases du masquage/affichage

Sans paramètres, les méthodes `.hide()` et `.show()` peuvent être considérées comme des méthodes abrégées intelligentes pour `.css('display','string')`, où `'string'` correspond à la valeur d'affichage adéquate. Elles ont pour effet de masquer ou d'afficher immédiatement les éléments sélectionnés, sans aucune animation.

La méthode `.hide()` affecte `display:none` à l'attribut de style des éléments du jeu correspondant. Son côté intelligent réside dans le fait qu'elle mémorise la valeur de la propriété `display`, en général `block` ou `inline`, avant de la fixer à `none`. À l'inverse, la méthode `.show()` replace la propriété `display` des éléments du jeu correspondant à la valeur qu'elle avait avant sa modification à `none`.

INFO

Pour de plus amples informations concernant la propriété `display` et la représentation visuelle de ses valeurs dans une page web, visitez le Mozilla Developer Center à l'adresse **https://developer.mozilla.org/fr/CSS/display/** et étudiez les exemples à **https://developer.mozilla.org/samples/cssref/display.html**.

Cette caractéristique de `.show()` et de `.hide()` se révèle particulièrement utile pour masquer les éléments dont la valeur par défaut de la propriété `display` est redéfinie dans une feuille de style. Par exemple, la propriété `display` de l'élément `<li>` est fixée par défaut à `block`, mais nous pouvons la modifier à `inline` pour obtenir un menu horizontal. En utilisant la méthode `.show()` sur cet élément `<li>` masqué, sa propriété `display` n'est pas réinitialisée à sa valeur par défaut, `block`, ce qui évite qu'il ne soit remis sur sa propre ligne. À la place, l'élément est restauré dans son état précédent, `display:inline`, préservant ainsi le menu horizontal.

Nous pouvons illustrer rapidement ces deux méthodes en ajoutant un second paragraphe et un lien "Pour en savoir plus..." après le premier paragraphe du contenu HTML d'exemple :

```
<div id="switcher">
  <div class="label">Taille du texte</div>
  <button id="switcher-default">Par défaut</button>
  <button id="switcher-large">Plus grande</button>
  <button id="switcher-small">Plus petite</button>
</div>
<div class="speech">
  <p>Devant Douaumont, en février, mon tour arriva, et je partis. Je suis la
     ligne et répare plusieurs fois jusqu'à la ferme de Thiaumont. Les balles
     de mitrailleuses rasaient le boyau, et j'étais aveuglé et couvert de terre
     par les obus.</p>
  <p>Le matin, au petit jour, nous recevons l'ordre de poser une nouvelle
     ligne. Nous voilà donc partis à quatre ou cinq avec les bobines ;
     seulement, cette fois, il faisait jour et l'ennemi nous tirait dessus avec
     les mitrailleuses des glacis du fort de Douaumont.</p>
```

```
    <a href="#" class="more">Pour en savoir plus...</a>
</div>
```

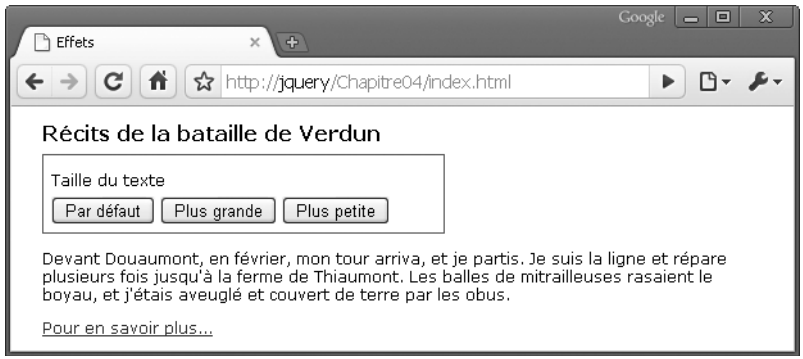
Lorsque le DOM est prêt, nous masquons le second paragraphe :

```
$(document).ready(function() {
  $('p:eq(1)').hide();
});
```

La page résultante est illustrée à la Figure 4.3.

Figure 4.3

Au chargement de la page, le second paragraphe est masqué.



Ensuite, lorsque l'utilisateur clique sur POUR EN SAVOIR PLUS... à la fin du premier paragraphe, le lien est masqué et le second paragraphe est affiché (voir Figure 4.4) :

```
$(document).ready(function() {
  $('p:eq(1)').hide();
  $('a.more').click(function() {
    $('p:eq(1)').show();
    $(this).hide();
    return false;
  });
});
```

Figure 4.4

Un clic sur le lien affiche le second paragraphe.

Notez le retour de la valeur `false`. Cela permet d'éviter que le lien n'exécute son *action par défaut*.

Les méthodes `.hide()` et `.show()` sont faciles d'emploi, mais le résultat n'est pas très clinquant. Pour que la page soit plus attrayante, nous pouvons faire intervenir une vitesse d'exécution.

4.3 Effets et vitesse

Lorsque nous fixons une *vitesse*, ou plus précisément une *durée*, aux méthodes `.show()` ou `.hide()`, elles mettent en place une animation qui se produit sur la période de temps indiquée. Par exemple, la méthode `.hide(vitesse)` diminue simultanément la hauteur, la largeur et l'opacité d'un élément jusqu'à zéro, puis applique à ce moment-là la règle CSS `display:none`. La méthode `.show(vitesse)` augmente la hauteur de l'élément du haut vers le bas, la largeur de la gauche vers la droite et l'opacité de 0 à 1, jusqu'à ce que son contenu soit totalement visible.

Ajouter une vitesse

Quel que soit l'effet jQuery, nous pouvons préciser l'une des trois vitesses prédéfinies : `'slow'`, `'normal'` et `'fast'`. Avec `.show('slow')`, l'animation se termine en 0,6 seconde, avec `.show('normal')`, en 0,4 seconde et, avec `.show('fast')`, en 0,2 seconde. Pour une meilleure précision, nous pouvons même indiquer un nombre de millisecondes, par exemple `.show(850)`. Contrairement aux noms des vitesses prédéfinies, les nombres ne sont pas placés entre apostrophes.

Ajoutons une vitesse à notre exemple pour l'affichage du second paragraphe des récits de la bataille de Verdun :

```
$(document).ready(function() {
  $('p:eq(1)').hide();
  $('a.more').click(function() {
    $('p:eq(1)').show('slow');
    $(this).hide();
    return false;
  });
});
```

La Figure 4.5 montre l'aspect du paragraphe lorsque la moitié de la durée de l'effet, environ, est écoulée.

Fondu enchaîné

Si les méthodes `.show()` et `.hide()` animées sont certainement attrayantes, elles peuvent parfois être trop clinquantes. jQuery propose donc deux autres animations prédéfinies, pour un effet plus subtil. Par exemple, pour que l'intégralité du paragraphe s'affiche en ne changeant que son opacité, nous pouvons utiliser `.fadeIn('slow')` :

Figure 4.5

L'aspect du second paragraphe à la moitié de l'animation.


```
$(document).ready(function() {
  $('p:eq(1)').hide();
  $('a.more').click(function() {
    $('p:eq(1)').fadeIn('slow');
    $(this).hide();
    return false;
  });
});
```

La Figure 4.6 illustre l'effet obtenu. Avec `.fadeIn()`, les dimensions du bloc du paragraphe sont tout d'abord fixées afin que le contenu puisse s'y afficher progressivement. Pour diminuer progressivement l'opacité, nous utilisons `.fadeOut()`.

Figure 4.6

Afficher le paragraphe en modifiant son opacité.

4.4 Effets composés

Nous avons parfois besoin d'inverser la visibilité des éléments, au lieu de les afficher une fois comme dans l'exemple précédent. Le basculement peut être obtenu en examinant tout d'abord la visibilité des éléments sélectionnés, puis en invoquant la méthode

appropriée. Nous modifions le script de la manière suivante, en utilisant à nouveau les effets de fondu :

```
$(document).ready(function() {
  var $firstPara = $('p:eq(1)');
  $firstPara.hide();
  $('a.more').click(function() {
    if ($firstPara.is(':hidden')) {
      $firstPara.fadeIn('slow');
      $(this).text('Pour en savoir moins...');
    } else {
      $firstPara.fadeOut('slow');
      $(this).text('Pour en savoir plus...');
    }
    return false;
  });
});
```

Comme nous l'avons fait précédemment dans ce chapitre, nous plaçons en cache notre sélecteur de manière à éviter un nouveau parcours du DOM. Par ailleurs, nous ne masquons plus le lien sur lequel l'utilisateur a cliqué. À la place, nous modifions son texte.

Une instruction `if else` est une solution parfaitement sensée pour inverser la visibilité des éléments. Mais, à l'aide des *effets composés* de jQuery, nous pouvons nous passer des instructions conditionnelles (dans cet exemple, nous avons cependant besoin d'une telle instruction pour le texte du lien). jQuery fournit la méthode `.toggle()`, qui opère comme `.show()` et `.hide()` et qui, à l'instar de ces deux méthodes, peut être employée avec ou sans un argument de vitesse. L'autre méthode composée se nomme `.slideToggle()`. Elle affiche ou masque des éléments en augmentant ou en diminuant progressivement leur hauteur. Voici une nouvelle version du script fondée sur la méthode `.slideToggle()` :

```
$(document).ready(function() {
  var $firstPara = $('p:eq(1)');
  $firstPara.hide();
  $('a.more').click(function() {
    $firstPara.slideToggle('slow');
    var $link = $(this);
    if ( $link.text() == "Pour en savoir plus..." ) {
      $link.text('Pour en savoir moins...');
    } else {
      $link.text('Pour en savoir plus...');
    }
    return false;
  });
});
```

Cette fois-ci, la répétition concernerait `$(this)`. Par conséquent, nous l'enregistrons dans la variable `$link` pour améliorer les performances et la lisibilité du code. Par ailleurs, l'instruction conditionnelle sert uniquement à modifier le lien du texte, elle vérifie le contenu de ce lien à la place de la visibilité du second paragraphe.

4.5 Créer des animations personnalisées

Outre les méthodes pour les effets prédéfinis, jQuery met à notre disposition une méthode puissante, `.animate()`, pour créer nos propres animations personnalisées. Elle existe en deux variantes. La première prend quatre arguments :

1. Une *mappe* de propriétés et de valeurs de style, semblable à la mappe de `.css()` mentionnée précédemment dans ce chapitre.
2. Une *vitesse* facultative, qui peut être indiquée à l'aide de l'une des chaînes prédéfinies ou en millisecondes.
3. Un *type d'easing*[1] facultatif ; cette option avancée sera présentée au Chapitre 10.
4. Une *fonction de rappel* facultative, sur laquelle nous reviendrons plus loin dans ce chapitre.

Voici un exemple d'invocation de cette méthode avec quatre arguments :

```
.animate({propriété1: 'valeur1', propriété2: 'valeur2'},
  vitesse, easing, function() {
    alert("L'animation est terminée.");
  }
);
```

La seconde variante prend deux arguments : une mappe de propriétés et une mappe d'options :

```
.animate({propriétés}, {options})
```

En réalité, le second argument regroupe les trois derniers de la première forme dans une mappe et ajoute deux nouvelles options. En utilisant les sauts de ligne pour faciliter la lecture, la seconde variante prend la forme suivante :

```
.animate({
  propriété1: 'valeur1',
  propriété2: 'valeur2'
}, {
  duration: 'valeur',
  easing: 'valeur',
  complete: function() {
    alert("L'animation est terminée.");
  },
  queue: boolean,
  step: callback
});
```

1. N.d.T. : selon Adobe, le terme *easing* se rapporte à l'accélération ou à la décélération graduelle, durant une animation, qui permet de rendre les animations plus réalistes. Cette technique crée une accélération ou une décélération d'apparence plus naturelle en ajustant graduellement le taux de modification.

Pour le moment, nous utiliserons la première variante de la méthode `.animate()`, mais nous passerons à la seconde lorsque nous présenterons les effets séquentiels.

Inverser l'effet de fondu

Lorsque nous avons vu les effets composés, avez-vous remarqué qu'il n'existait pas toujours une méthode pour réaliser un effet inverse ? Par exemple, si les méthodes de l'effet de glissement ont bien une méthode `.slideToggle()`, il n'existe en revanche aucune méthode `.fadeToggle()` pour accompagner `.fadeIn()` et `.fadeOut()`. Cela dit, nous pouvons utiliser la méthode `.animate()` pour créer facilement notre propre animation de fondu inversé. Dans l'exemple précédent, nous remplaçons `.slideToggle()` par notre propre animation :

```
$(document).ready(function() {
  $('p:eq(1)').hide();
  $('a.more').click(function() {
    $('p:eq(1)').animate({opacity: 'toggle'}, 'slow');
    var $link = $(this);
    if ( $link.text() == "Pour en savoir plus..." ) {
      $link.text('Pour en savoir moins...');
    } else {
      $link.text('Pour en savoir plus...');
    }
    return false;
  });
});
```

Comme l'illustre l'exemple, la méthode `.animate()` accepte des *valeurs abrégées* pour les propriétés CSS – `'show'`, `'hide'` et `'toggle'` – de manière à simplifier notre travail lorsque les méthodes abrégées ne sont pas adaptées à une certaine tâche.

Animer plusieurs propriétés

Avec la méthode `.animate()`, nous pouvons affecter simultanément n'importe quelle combinaison de propriétés. Par exemple, pour créer un effet de glissement et de fondu sur le second paragraphe, nous pouvons simplement ajouter un couple propriété-valeur pour `height` dans la mappe des propriétés de `.animate()` :

```
$(document).ready(function() {
  $('p:eq(1)').hide();
  $('a.more').click(function() {
    $('p:eq(1)').animate({
      opacity: 'toggle',
      height: 'toggle'
    },
    'slow');
    var $link = $(this);
    if ( $link.text() == "Pour en savoir plus..." ) {
      $link.text('Pour en savoir moins...');
    } else {
      $link.text('Pour en savoir plus...');
```

```
    }
    return false;
  });
});
```

De plus, nous pouvons non seulement utiliser les propriétés de style pour les effets, mais également d'autres propriétés comme `left`, `top`, `fontSize`, `margin`, `padding` et `borderWidth`. Revenons au script qui modifie la taille du texte des paragraphes et animons l'augmentation ou la diminution de la taille en substituant simplement la méthode `.animate()` à la méthode `.css()` :

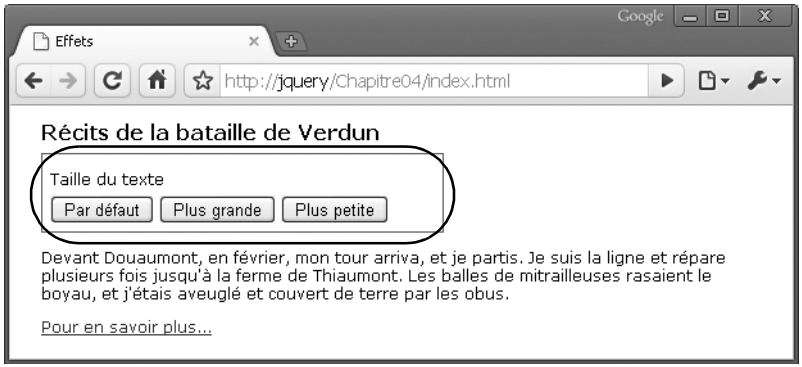
```
$(document).ready(function() {
  var $speech = $('div.speech');
  var defaultSize = $speech.css('fontSize');
  $('#switcher button').click(function() {
    var num = parseFloat( $speech.css('fontSize') );
    switch (this.id) {
      case 'switcher-large':
        num *= 1.4;
        break;
      case 'switcher-small':
        num /= 1.4;
        break;
      default:
        num = parseFloat(defaultSize);
    }
    $speech.animate({fontSize: num + 'px'}, 'slow');
  });
});
```

Les propriétés supplémentaires nous permettent également de créer des effets beaucoup plus complexes. Par exemple, nous pouvons déplacer un élément du côté gauche de la page vers le côté droit, tout en augmentant sa hauteur de 20 pixels et en modifiant sa bordure à 5 pixels.

Appliquons cet effet au rectangle `<div id="switcher">`. La Figure 4.7 présente son aspect avant que nous ne l'animions.

Figure 4.7

Le sélecteur de taille du texte avant son animation.



La largeur de la page n'étant pas figée, nous devons calculer la distance du déplacement du rectangle avant de pouvoir l'aligner sur le bord droit de la page. Si l'on suppose que la largeur du paragraphe est de 100 %, nous pouvons soustraire la largeur du rectangle TAILLE DU TEXTE à celle du paragraphe. La méthode `.width()` de jQuery est souvent pratique pour de tels calculs, mais elle ne tient pas compte des espacements gauche et droit ni de l'épaisseur des bordures. Depuis jQuery version 1.2.6, nous disposons toutefois de la méthode `.outerWidth()`, qui tient compte de ces caractéristiques dans la détermination de la largeur. Nous l'utilisons donc dans notre exemple. L'animation est déclenchée en cliquant sur le libellé TAILLE DU TEXTE, situé juste au-dessus des boutons. Voici le code correspondant :

```
$(document).ready(function() {
  $('div.label').click(function() {
    var paraWidth = $('div.speech p').outerWidth();
    var $switcher = $(this).parent();
    var switcherWidth = $switcher.outerWidth();
    $switcher.animate({left: paraWidth - switcherWidth,
          height: '+=20px', borderWidth: '5px'}, 'slow');
  });
});
```

Vous remarquerez que la valeur de la propriété `height` est précédée de `+=`. Cette expression, introduite dans jQuery 1.2, indique une *valeur relative*. Par conséquent, au lieu d'animer la hauteur jusqu'à 20 pixels, elle est animée jusqu'à la hauteur courante plus 20 pixels.

Ce code augmente bien la hauteur du `<div>` et épaissit ses bordures, mais, pour le moment, son ancrage à gauche ne peut pas être modifié. Nous devons autoriser le changement de sa position dans la feuille de style CSS.

Positionnement avec CSS

Lorsqu'on utilise `.animate()`, il ne faut surtout pas oublier les limites imposées par CSS sur les éléments qui sont animés. Par exemple, la modification de la propriété `left` n'aura aucun effet sur les éléments si leur positionnement CSS n'est pas fixé à `relative` ou `absolute`. Pour tous les éléments de type bloc, le positionnement CSS par défaut est `static`. Il indique précisément que les éléments resteront à leur place si nous tentons de les déplacer sans modifier auparavant la valeur de leur propriété `position`.

INFO

Pour de plus amples informations concernant le positionnement absolu et relatif, consultez l'article *Absolutely Relative* de Joe Gillespie (**http://www.wpdfd.com/issues/78/absolutely_relative/**).

Nous modifions la feuille de style pour que le positionnement de `<div id="switcher">` soit relatif :

```
#switcher {
  position: relative;
}
```

À présent, en cliquant sur le libellé TAILLE DU TEXTE, l'animation se termine dans l'état illustré à la Figure 4.8.

Figure 4.8

Le sélecteur de taille du texte lorsque l'animation est terminée.

4.6 Effets simultanés ou séquentiels

Nous venons de le voir, la méthode `.animate()` est très pratique pour appliquer simultanément des effets sur un jeu d'éléments. Cependant, nous souhaitons parfois placer les effets dans une *file* afin qu'ils produisent séquentiellement.

Manipuler un seul jeu d'éléments

Lorsque plusieurs effets sont appliqués au même jeu d'éléments, il est très facile de les mettre à la suite par simple chaînage. Pour illustrer cette méthode, nous allons à nouveau déplacer le rectangle TAILLE DU TEXTE vers la droite, en augmentant sa hauteur et l'épaisseur de sa bordure. Cependant, cette fois-ci, nous réaliserons les trois effets séquentiellement, en plaçant chacun d'eux dans sa propre méthode `.animate()` et en les chaînant :

```
$(document).ready(function() {
  $('div.label').click(function() {
    var paraWidth = $('div.speech p').outerWidth();
    var $switcher = $(this).parent();
    var switcherWidth = $switcher.outerWidth();
    $switcher
      .animate({left: paraWidth - switcherWidth}, 'slow')
      .animate({height: '+=20px'}, 'slow')
```

```
        .animate({borderWidth: '5px'}, 'slow');
  });
});
```

Nous pouvons évidemment mettre les trois méthodes `.animate()` sur la même ligne, mais, pour des questions de facilité de lecture, nous les avons placées chacune sur leur propre ligne et les avons indentées.

Toutes les méthodes d'effets jQuery, pas seulement la méthode `.animate()`, peuvent ainsi être mises à la suite par chaînage. Par exemple, nous pouvons appliquer la séquence d'effets suivante sur `<div id="switcher">` :

1. Diminuer son opacité à 0,5 avec `.fadeTo()`.
2. Le déplacer vers la droite avec `.animate()`.
3. Augmenter son opacité à 1 avec `.fadeTo()`.
4. Le masquer avec `.slideUp()`.
5. L'afficher à nouveau avec `.slideDown()`.

Pour cela, il suffit que le code enchaîne ces effets dans cet ordre :

```
$(document).ready(function() {
  $('div.label').click(function() {
    var paraWidth = $('div.speech p').outerWidth();
    var $switcher = $(this).parent();
    var switcherWidth = $switcher.outerWidth();
    $switcher
      .fadeTo('fast',0.5)
      .animate({ 'left': paraWidth - switcherWidth }, 'slow')
      .fadeTo('slow',1.0)
      .slideUp('slow')
      .slideDown('slow');
  });
});
```

Toutefois, comment pouvons-nous simultanément déplacer le `<div>` vers la droite et diminuer son opacité de moitié ? Si les deux animations se produisent à la même vitesse, il suffit de les combiner dans une méthode `.animate()`. Cependant, dans cet exemple, le fondu est rapide (vitesse `'fast'`) tandis que le déplacement vers la droite est lent (vitesse `'slow'`). C'est dans de telles circonstances que la seconde variante de la méthode `.animate()` s'impose :

```
$(document).ready(function() {
  $('div.label').click(function() {
  var paraWidth = $('div.speech p').outerWidth();
  var $switcher = $(this).parent();
  var switcherWidth = $switcher.outerWidth();
  $switcher
    .fadeTo('fast',0.5)
    .animate({'left': paraWidth - switcherWidth},
             {duration: 'slow', queue: false})
```

```
      .fadeTo('slow',1.0)
      .slideUp('slow')
      .slideDown('slow');
  });
});
```

Le deuxième argument, une mappe d'options, précise l'option `queue`, qui, lorsqu'elle vaut `false`, fait que l'animation démarre simultanément à la précédente.

Dans une suite d'effets appliqués à un même jeu d'éléments, il faut savoir que le séquencement ne s'applique pas automatiquement aux méthodes qui ne mettent pas en œuvre des effets, comme `.css()`. Supposons que nous voulions changer la couleur d'arrière-plan de `<div id="switcher">` à rouge après `.slideUp()` mais avant `slideDown()`. Nous essayons donc le code suivant :

```
$(document).ready(function() {
  $('div.label').click(function() {
  var paraWidth = $('div.speech p').outerWidth();
  var $switcher = $(this).parent();
  var switcherWidth = $switcher.outerWidth();
  $switcher
    .fadeTo('fast',0.5)
    .animate({'left': paraWidth - switcherWidth}, 'slow')
    .fadeTo('slow',1.0)
    .slideUp('slow')
    .css('backgroundColor','#f00')
    .slideDown('slow');
  });
});
```

Mais, bien que la modification de l'arrière-plan se trouve à la place dans la chaîne, elle a lieu juste après le clic sur le libellé.

Pour ajouter de telles méthodes dans la séquence, une solution consiste à employer la méthode nommée fort à propos `.queue()`. Voici comment l'utiliser dans notre script :

```
$(document).ready(function() {
  $('div.label').click(function() {
  var paraWidth = $('div.speech p').outerWidth();
  var $switcher = $(this).parent();
  var switcherWidth = $switcher.outerWidth();
  $switcher
    .fadeTo('fast',0.5)
    .animate({'left': paraWidth - switcherWidth}, 'slow')
    .fadeTo('slow',1.0)
    .slideUp('slow')
    .queue(function() {
      $switcher
        .css('backgroundColor', '#f00')
        .dequeue();
    })
    .slideDown('slow');
  });
});
```

Lorsqu'une *fonction de rappel* est passée à la méthode `.queue()`, comme c'est le cas ici, elle ajoute cette fonction à la file des effets associée aux éléments correspondants. Dans cette fonction, nous fixons la couleur de l'arrière-plan à rouge, puis invoquons la méthode `.dequeue()`. Grâce à cet appel, la séquence d'animation reprend là où elle a été détournée et termine la chaîne avec la ligne `.slideDown('slow')`. Sans l'invocation de `.dequeue()`, l'animation se serait arrêtée.

INFO

Pour de plus amples informations concernant les méthodes `.queue()` et `.dequeue()`, ainsi que des exemples d'utilisation, consultez la page **http://docs.jquery.com/Effects**.

Nous verrons une autre manière de placer dans la file des méthodes non liées aux effets lorsque nous examinerons l'application des effets à plusieurs jeux d'éléments.

Manipuler plusieurs jeux d'éléments

Lorsque les effets sont appliqués à des jeux d'éléments différents, ils se produisent virtuellement au même moment. Pour le constater, nous allons faire glisser un paragraphe vers le bas tout en en faisant glisser un autre vers le haut. Tout d'abord, nous ajoutons d'autres paragraphes au fichier HTML des récits de la bataille de Verdun :

```
<div id="switcher">
  <div class="label">Taille du texte</div>
  <button id="switcher-default">Par défaut</button>
  <button id="switcher-large">Plus grande</button>
  <button id="switcher-small">Plus petite</button>
</div>
<div class="speech">
  <p>Devant Douaumont, en février, mon tour arriva, et je partis. Je suis la
     ligne et répare plusieurs fois jusqu'à la ferme de Thiaumont. Les balles
     de mitrailleuses rasaient le boyau, et j'étais aveuglé et couvert de terre
     par les obus.</p>
  <p>Le matin, au petit jour, nous recevons l'ordre de poser une nouvelle
     ligne. Nous voilà donc partis à quatre ou cinq avec les bobines ;
     seulement, cette fois, il faisait jour et l'ennemi nous tirait dessus avec
     les mitrailleuses des glacis du fort de Douaumont.</p>
  <a href="#" class="more">Pour en savoir plus...</a>
  <p>Sur le coup de midi, je reçois l'ordre de poser une nouvelle ligne de
     Thiamont à Fleury. Nous partons à deux, chacun avec une bobine de fil.
     </p>
  <p>Mon camarade déroule, moi je suis en attachant le fil à tout ce que je
     peux trouver comme support. A un moment donné, j'ai beau chercher, je ne
     trouve rien comme support, lorsque j'aperçois plusieurs corps
     étendus ; je me précipite et un tour de fil dans les jambes de l'un,
     un tour dans celles des autres, ma ligne se trouve fixée dans la plaine.
     </p>
</div>
```

Ensuite, pour que nous puissions voir ce qui se produit au cours de l'animation, nous affectons au troisième paragraphe une bordure de 1 pixel et au quatrième, un fond gris. Nous masquons également ce dernier lorsque le DOM est prêt :

```
$(document).ready(function() {
  $('p:eq(2)').css('border', '1px solid #333');
  $('p:eq(3)').css('backgroundColor', '#ccc').hide();
});
```

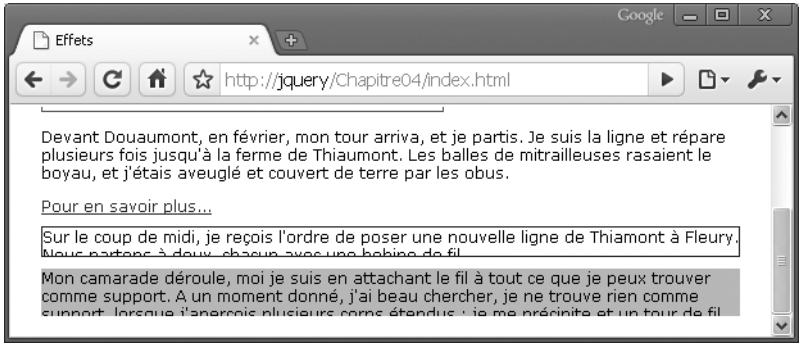
Enfin, nous invoquons la méthode `.click()` sur le troisième paragraphe afin qu'un clic le fasse glisser vers le haut (pour le faire disparaître) et fasse glisser le quatrième vers le bas (pour le faire apparaître) :

```
$(document).ready(function() {
  $('p:eq(2)')
    .css('border', '1px solid #333')
    .click(function() {
      $(this).slideUp('slow').next().slideDown('slow');
    });
  $('p:eq(3)').css('backgroundColor', '#ccc').hide();
});
```

La Figure 4.9 illustre ces deux effets à mi-chemin et confirme qu'ils se produisent quasi simultanément.

Figure 4.9

Effets simultanés sur des éléments différents.



Le troisième paragraphe, qui était initialement visible, est à présent à moitié masqué, tandis que le quatrième, qui était initialement masqué, est à présent à moitié affiché.

Fonctions de rappel

Pour pouvoir enchaîner des effets sur des éléments différents, jQuery permet de passer une *fonction de rappel* à chaque méthode d'effet. Nous l'avons vu avec les gestionnaires d'événements et la méthode `.queue()`, les fonctions de rappel ne sont rien d'autre que des fonctions passées en argument des méthodes. Dans le contexte des effets, elles apparaissent en dernier argument de la méthode.

En utilisant une fonction de rappel pour mettre deux effets de glissement à la suite, nous pouvons faire en sorte que l'animation du quatrième paragraphe se termine avant celle du troisième. Commençons par présenter l'utilisation d'une fonction de rappel avec la méthode `.slideDown()` :

```
$(document).ready(function() {
  $('p:eq(2)')
    .css('border', '1px solid #333')
    .click(function() {
      $(this).next().slideDown('slow',function() {
        // Ce code est exécuté après que l'effet de glissement
        // vers le bas du quatrième paragraphe est terminé.
      });
    });
  $('p:eq(3)').css('backgroundColor', '#ccc').hide();
});
```

Toutefois, nous devons faire attention à la cible du glissement vers le haut. Le contexte de `$(this)` a changé car la fonction de rappel se trouve dans la méthode `.slideDown()`. Par conséquent, `$(this)` ne correspond plus au troisième paragraphe comme c'est le cas dans la méthode `.click()`. Puisque la méthode `.slideDown()` est invoquée sur `$(this).next()`, son code voit en `$(this)` le frère suivant, c'est-à-dire le quatrième paragraphe. Par conséquent, si nous plaçons `$(this).slideUp('slow')` dans la fonction de rappel, nous masquons en réalité le paragraphe que nous venons de faire apparaître.

Pour que les références à `$(this)` restent cohérentes, il suffit d'enregistrer `$(this)` dans une variable dès le début de la méthode `.click()` : `var $thirdPara = $(this)`. La variable `$thirdPara` fait alors référence au troisième paragraphe, que ce soit dans ou à l'extérieur de la fonction de rappel. Voici le nouveau code qui utilise cette variable :

```
$(document).ready(function() {
  var $thirdPara = $('p:eq(2)');
  $thirdPara
    .css('border', '1px solid #333')
    .click(function() {
      $(this).next().slideDown('slow',function() {
        $thirdPara.slideUp('slow');
      });
    });
  $('p:eq(3)').css('backgroundColor', '#ccc').hide();
});
```

L'utilisation de `$thirdPara` dans la fonction de rappel `.slideDown()` est en rapport avec les propriétés des *fermetures*. À l'Annexe C, nous reviendrons sur ce sujet important mais difficile à maîtriser.

La Figure 4.10 illustre les effets à mi-parcours. Cette fois-ci, les troisième et quatrième paragraphes sont visibles ; le glissement vers le bas du quatrième est terminé, tandis que le glissement vers le haut du troisième va commencer.

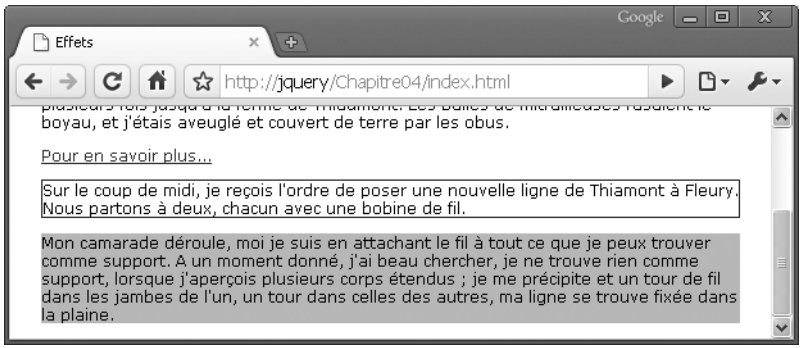


Figure 4.10

L'animation du troisième paragraphe débute lorsque celle du quatrième est terminée.

Puisque nous connaissons à présent les fonctions de rappel, revenons au code qui modifiait la couleur d'arrière-plan vers la fin d'une suite d'effets. Nous pouvons remplacer la méthode `.queue()` utilisée précédemment par une fonction de rappel :

```
$(document).ready(function() {
  $('div.label').click(function() {
    var paraWidth = $('div.speech p').outerWidth();
    var $switcher = $(this).parent();
    var switcherWidth = $switcher.outerWidth();
    $switcher
      .fadeTo('slow',0.5)
      .animate({'left': paraWidth - switcherWidth}, 'slow')
      .fadeTo('slow',1.0)
      .slideUp('slow', function() {
        $switcher
          .css('backgroundColor', '#f00');
      })
      .slideDown('slow');
  });
});
```

Dans cette version également, la couleur d'arrière-plan de `<div id="switcher">` passe au rouge après le glissement vers le haut et avant le glissement vers le bas.

En bref

Avec toutes ces possibilités de mise en application des effets, il peut être difficile de déterminer s'ils se produiront simultanément ou séquentiellement. Un bref récapitulatif facilitera la réflexion :

1. Les effets appliqués à un même jeu d'éléments sont :
   - *simultanés* lorsqu'ils concernent plusieurs propriétés dans une même méthode `.animate()` ;
   - *séquentiels* lorsqu'ils sont mis en œuvre par un enchaînement de méthodes, sauf si l'option `queue` est fixée à `false`.

2. Les effets appliqués à plusieurs jeux d'éléments sont :
   - *simultanés* par défaut ;
   - *séquentiels* lorsqu'ils sont mis en œuvre dans la fonction de rappel d'un autre effet ou dans celle de la méthode `.queue()`.

4.7 En résumé

En utilisant les méthodes d'effet étudiées dans ce chapitre, nous sommes à présent capables d'augmenter et de diminuer progressivement la taille d'un texte, que ce soit avec `.css()` ou `.animate()`. Nous connaissons également différentes manières d'appliquer différents effets pour masquer et afficher progressivement des éléments de la page, ainsi que pour animer des éléments, simultanément ou séquentiellement.

Dans les quatre premiers chapitres de cet ouvrage, tous les exemples manipulaient des éléments définis dans le fichier HTML de la page. Au Chapitre 5, nous présenterons plusieurs façons d'utiliser jQuery pour créer de nouveaux éléments et les insérer où bon nous semble dans le DOM.

5

Manipulation du DOM

Au sommaire de ce chapitre

- Manipuler des attributs
- Insérer de nouveaux éléments
- Déplacer des éléments
- Envelopper des éléments
- Copier des éléments
- Les méthodes du DOM en bref
- En résumé

Tel le magicien qui fait sortir un bouquet de fleurs de nulle part, jQuery permet de créer des éléments, des attributs et du texte dans une page, comme par magie. Mais ce n'est pas tout. Avec jQuery, nous pouvons également les faire disparaître. Nous pouvons même transformer le bouquet de fleurs en colombe.

5.1 Manipuler des attributs

Tout au long des quatre premiers chapitres, nous avons employé les méthodes `.addClass()` et `.removeClass()` pour illustrer la modification de l'aspect des éléments d'une page. Ces deux méthodes manipulent l'attribut `class` ou, dans le jargon DOM, la propriété `className`. La méthode `.addClass()` crée l'attribut ou lui ajoute une classe, tandis que `.removeClass()` le supprime ou lui retire une classe. Par ailleurs, avec la méthode `.toggleClass()`, qui alterne entre l'ajout et la suppression d'une classe, nous disposons d'un mécanisme efficace et robuste pour la gestion des classes.

Toutefois, l'attribut `class` n'est pas le seul que nous puissions manipuler. Pour examiner ou modifier les autres attributs, comme `id`, `rel` ou `href`, jQuery fournit les méthodes `.attr()` et `.removeAttr()`. Elles peuvent même servir pour l'attribut `class`, mais

les méthodes spécialisées `.addClass()` et `.removeClass()` doivent être préférées car elles prennent en charge les éléments ayant plusieurs classes, comme `<div class="un deux">`.

Attributs autres que *class*

Sans l'aide de jQuery, la manipulation de certains attributs n'est pas aisée. Par ailleurs, jQuery nous permet de modifier plusieurs attributs à la fois, de la même manière que nous avons manipulé plusieurs propriétés CSS à l'aide de la méthode `.css()` au Chapitre 4. Par exemple, nous pouvons très facilement fixer la valeur des attributs `id`, `rel` et `title` des liens, tous à la fois. Partons du balisage HTML suivant :

```
<h1 id="f-title">Flatland : une aventure à plusieurs dimensions</h1>
<div id="f-author">par Edwin A. Abbott</div>
<h2>Partie 1, Section 3</h2>
<h3 id="f-subtitle">
  Des habitants de Flatland
</h3>
<div id="excerpt">un extrait</div>

<div class="chapter">

  <p class="square">Les Membres des Professions Libérales et les Gentilshommes
    sont des Carrés (c'est à cette classe que j'appartiens personnellement) et
    des Figures à Cinq-Côtés ou <a
    href="http://fr.wikipedia.org/wiki/Pentagone_(figure)">Pentagones</a>.
  </p>
  <p class="nobility hexagon">Vient ensuite la Noblesse, qui comporte plusieurs
    degrés, en commençant par les Figures à Six-Côtés, ou <a
    href="http://fr.wikipedia.org/wiki/Hexagone">Hexagones</a>, et ainsi de
    suite, le nombre des côtés s'élevant sans cesse, jusqu'aux Personnages qui
    reçoivent le titre honorable de Polygones. Enfin, lorsque le nombre des
    côtés devient si grand, et que les côtés eux-mêmes sont si petits qu'il est
    impossible de distinguer la Figure d'un <a
    href="http://fr.wikipedia.org/wiki/Cercle">Cercle</a>, elle entre dans la
    classe Circulaire ou Ecclésiastique : c'est l'ordre le plus élevé de tous.
  </p>
  <p><span class="pull-quote"><span class="drop">Chez nous, </span>une Loi de
    la Nature veut qu'un enfant mâle ait toujours <strong>un côté de plus
    </strong> que son père</span>, de sorte que chaque génération s'élève (en
    règle générale) d'un échelon sur la voie du progrès et de l'anoblissement.
    Ainsi le fils d'un Carré sera un Pentagone ; le fils du Pentagone, un
    Hexagone ; etc.
  </p>
<!-- . . . le code continue . . . -->
</div>
```

Nous pouvons itérer sur les liens contenus dans `<div class="chapter">` et leur appliquer les attributs un par un. Si nous voulons donner une même valeur d'attribut à tous les liens, nous pouvons écrire une seule ligne de code dans le gestionnaire `$(document).ready()` :

```
$(document).ready(function() {
  $('div.chapter a').attr({'rel': 'external'});
});
```

Cette solution fonctionne car nous souhaitons que le nouvel attribut `rel` ait la même valeur pour chaque lien. Toutefois, les attributs que nous ajoutons ou modifions doivent souvent avoir des valeurs différentes selon l'élément. Par exemple, dans un document donné, chaque `id` d'élément doit être unique pour que le code JavaScript ait un comportement prévisible. Pour affecter un `id` à chaque lien, nous abandonnons la ligne précédente et optons pour la méthode `.each()` de jQuery :

```
$(document).ready(function() {
  $('div.chapter a').each(function(index) {
    $(this).attr({
      'rel': 'external',
      'id': 'wikilink-' + index
    });
  });
});
```

La méthode `.each()`, qui opère comme un *itérateur explicite*, est une version plus pratique de la boucle `for`. Elle peut être utilisée lorsque le code de traitement de chaque élément de la collection obtenue à l'aide du sélecteur est trop complexe pour l'*itération implicite*. Dans notre exemple, la fonction anonyme de la méthode `.each()` reçoit un indice, qui est ajouté à chaque identifiant. L'argument `index` joue le rôle de compteur, qui commence à zéro pour le premier lien et augmente de un pour chaque lien successif. Ainsi, l'affectation de la valeur `'wikilink-' + index` à l'attribut `id` donne au premier lien l'identifiant `wikilink-0`, au second, l'identifiant `wikilink-1`, etc.

INFO

En réalité, nous aurions pu nous contenter de l'itération implicite dans ce cas, car la méthode `.attr()` accepte une fonction en second argument, de manière semblable à la méthode `.filter()` rencontrée au Chapitre 2 (pour de plus amples informations, consultez la page **http://docs.jquery.com/Attributes/attr#keyfn**). Cependant, l'utilisation de `.each()` semble mieux correspondre à nos besoins.

Nous utilisons l'attribut `title` pour inviter l'internaute à consulter l'article Wikipédia qui correspond au terme du lien. Dans notre exemple, le sélecteur est simple car tous les liens pointent vers Wikipédia. Toutefois, il est sans doute préférable que l'expression de sélection soit un peu plus précise de manière à ne retenir que les liens dont l'attribut `href` contient `wikipedia`, simplement pour le cas où nous déciderions d'ajouter ultérieurement un lien vers un autre site que Wikipédia :

```
$(document).ready(function() {
  $('div.chapter a[href*=wikipedia]').each(function(index) {
    var $thisLink = $(this);
```

```
      $thisLink.attr({
        'rel': 'external',
        'id': 'wikilink-' + index,
        'title': 'en savoir plus sur ' + $thisLink.text() + ' sur Wikipédia'
      });
    });
});
```

Vous remarquerez que nous enregistrons `$(this)` dans la variable `$thisLink`, car cette référence est utilisée plusieurs fois.

Après que les valeurs des trois attributs ont été fixées, le balisage HTML du premier lien est devenu le suivant :

```
<a href="http://fr.wikipedia.org/wiki/Pentagone_(figure)" rel="external"
   id="wikilink-0" title="en savoir plus sur Pentagones sur
   Wikipédia">Pentagones</a>
```

La fonction *$()* revisitée

Depuis le début de ce livre, la fonction `$()` nous sert à accéder aux éléments d'un document. En un sens, elle constitue le cœur de la bibliothèque jQuery, puisque nous l'utilisons chaque fois que nous voulons associer un effet, un événement ou une propriété à une collection d'éléments correspondants.

Mais la fonction `$()` cache un autre secret entre ses parenthèses – une fonctionnalité si puissante qu'elle peut non seulement modifier l'apparence visuelle d'une page, mais également son contenu réel. En plaçant simplement un morceau de code HTML entre les parenthèses, nous pouvons créer une structure DOM totalement nouvelle.

Rappel concernant l'accessibilité

Vous ne devez pas oublier les risques qu'il y a à proposer certaines fonctionnalités, certaines améliorations visuelles et certaines informations textuelles uniquement aux internautes dont les navigateurs web prennent en charge JavaScript. Les informations importantes doivent être accessibles à tous, pas uniquement aux utilisateurs qui disposent du logiciel compatible.

Les pages de FAQ proposent souvent un lien de *retour au début* du document après chaque question-réponse. Ces liens n'ont, à notre avis, aucune existence sémantique et peuvent donc être ajoutés à l'aide d'un code JavaScript, en tant qu'amélioration proposée aux visiteurs. Dans notre exemple, nous allons ajouter un lien de retour vers le début de la page après chaque paragraphe, ainsi que l'ancre ciblée par ces liens. Tout d'abord, nous créons simplement les nouveaux éléments :

```
$(document).ready(function() {
   $('<a href="#top">retour au début</a>');
  $('<a id="top"></a>');
});
```

La Figure 5.1 illustre l'aspect de la page à ce stade. Où sont donc les liens de retour vers le début et l'ancre ? Ne devraient-ils pas apparaître ? La réponse est non. En effet, si les deux lignes créent bien les éléments, ceux-ci ne sont pas encore ajoutés à la page. Pour cela, nous devons employer l'une des nombreuses *méthodes d'insertion* de jQuery.

Figure 5.1

L'aspect initial de la page.

5.2 Insérer de nouveaux éléments

jQuery propose deux méthodes pour insérer des éléments avant d'autres éléments : `.insertBefore()` et `.before()`. Elles ont la même fonction, leur différence résidant dans la manière de les *chaîner* à d'autres méthodes. Deux autres méthodes, `.insertAfter()` et `.after()`, jouent des rôles équivalents, mais, comme leur nom le suggère, elles insèrent des éléments après d'autres éléments. Pour ajouter les liens de retour au début, nous invoquons la méthode `.insertAfter()` :

```
$(document).ready(function() {
  $('<a href="#top">retour au début</a>').insertAfter('div.chapter p');
  $('<a id="top"></a>');
});
```

La méthode `.after()` produirait le même résultat que `.insertAfter()`, mais l'expression de sélection doit être indiquée avant la méthode, non après. Avec `.after()`, la première ligne qui suit `$(document).ready()` devient :

```
$('div.chapter p').after('<a href="#top">retour au début</a>');
```

Avec `.insertAfter()`, nous pouvons continuer à manipuler l'élément `<a>` créé en chaînant simplement des méthodes supplémentaires. Avec `.after()`, ces méthodes opéreraient à la place sur les éléments obtenus à l'aide du sélecteur `$('div.chapter p')`.

À présent, puisque nous insérons les liens dans la page (et dans le DOM) après chaque paragraphe qui apparaît dans `<div class="chapter">`, les liens de retour au début sont affichés (voir Figure 5.2).

Figure 5.2

Les liens de retour au début du document sont insérés.

Malheureusement, les liens ne sont pas encore opérationnels, car il manque l'ancre `id="top"`. Pour l'ajouter, nous pouvons nous servir de l'une des méthodes qui insèrent des éléments dans d'autres éléments :

```
$(document).ready(function() {
  $('<a href="#top">retour au début</a>').insertAfter('div.chapter p');
  $('<a id="top" name="top"></a>').prependTo('body');
});
```

Le code ajouté insère l'ancre juste au début de l'élément `<body>`, c'est-à-dire au début de la page. À présent, puisque les liens ont été ajoutés à l'aide de la méthode `.insertAfter()`, comme l'ancre avec la méthode `.prependTo()`, nous disposons d'un ensemble de liens totalement opérationnels pour revenir au début de la page.

Toutefois, il est inutile d'afficher des liens de retour au début lorsque le début de la page est visible. Une rapide amélioration du script permet d'ajouter les liens uniquement après le quatrième paragraphe, par exemple. Pour cela, il suffit simplement de modifier l'expression de sélection : `.insertAfter('div.chapter p:gt(2)')`. Pourquoi `2` ? Il ne faut pas oublier que les indices JavaScript commencent à zéro. Par conséquent, le premier paragraphe possède l'indice 0, le deuxième, l'indice 1, le troisième, l'indice 2, et le quatrième, l'indice 3. Notre expression de sélection débute l'insertion des liens après chaque paragraphe dès que l'indice est supérieur à 2, c'est-à-dire 3.

La Figure 5.3 (page suivante) illustre le changement apporté par cette nouvelle expression de sélection.

5.3 Déplacer des éléments

Avec les liens de retour au début, nous avons créé de nouveaux éléments et les avons ajoutés dans la page. Mais nous pouvons également prendre des éléments de la page et les déplacer à un autre endroit. Cette possibilité peut être utile pour mettre en forme dynamiquement des notes de bas de page. Une telle note apparaît déjà dans le texte original de Flatland utilisé dans notre exemple, mais, pour illustrer notre propos, nous allons considérer d'autres parties du texte comme des notes de bas de page :

```
<p>Il est bien rare &mdash; en proportion du très grand nombre de naissances
   Isocèles &mdash; qu'un Triangle Équilatéral authentique et certifiable
   naisse de parents Isocèles. <span class="footnote">« Quel besoin a-t-on d'un
   certificat ? » demandera peut-être un critique de Spaceland. « La
   procréation d'un Fils Carré n'est-elle pas un certificat de la Nature
   elle-même et ne prouve-t-elle pas l'équilatéralité du Père ? » Je répondrai
   qu'aucune Dame de condition n'accepterait d'épouser un Triangle non
   certifié. On a vu parfois des Triangles légèrement irréguliers donner
   naissance à des rejetons Carrés ; mais, dans presque tous les cas,
   l'Irrégularité de la première génération réapparaît dans la troisième qui,
   soit ne parvient pas à atteindre le rang de Pentagone, soit retombe dans le
   Triangulaire.</span> Pour arriver à ce résultat, toute une série de mariages
   mixtes calculés avec soin est d'abord nécessaire ; encore faut-il que ceux
   qui aspirent à devenir les ancêtres du futur Équilatéral s'exercent pendant
   un laps de temps prolongé à la frugalité, à la maîtrise de soi, et qu'à
   travers des générations successives s'opère un développement patient,
   systématique et continu de l'intellect Isocèle.
</p>
```

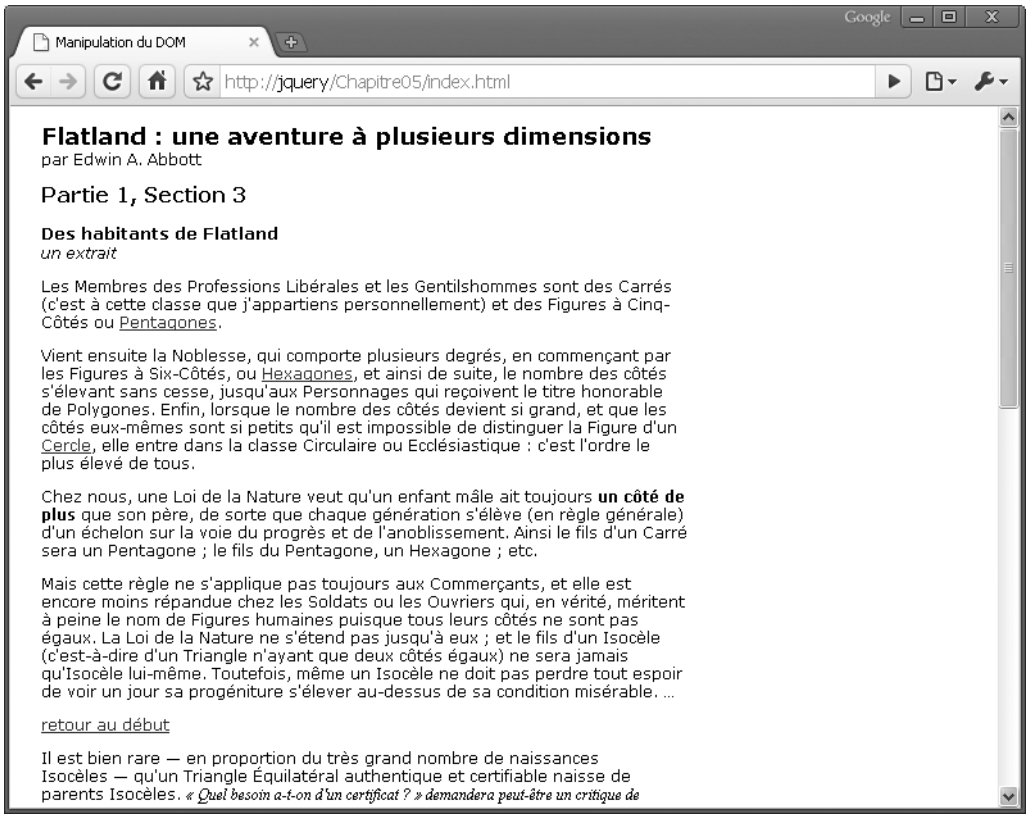


Figure 5.3

Les liens de retour vers le début n'apparaissent qu'après le quatrième paragraphe.

```
<p><span class="pull-quote">Dans notre pays, quand un Vrai Triangle Équilatéral
   naît de parents Isocèles, c'est un événement dont on se réjouit<span
   class="drop"> à plusieurs lieues à la ronde.</span></span> Après un sévère
   examen effectué par le Conseil Sanitaire et Social, l'enfant, s'il est
   certifié Régulier, est admis au cours d'une cérémonie solennelle dans la
   classe des Équilatéraux. Il est immédiatement enlevé à ses parents, qui se
   sentent partagés entre l'orgueil et l'affliction, et adopté par quelque
   Équilatéral sans descendance. <span class="footnote">Celui-ci s'engage par
   serment à ne plus jamais laisser l'enfant pénétrer dans son ancien domicile
   ou même jeter les yeux sur un membre de sa famille, de crainte que
   l'organisme dont le développement est si récent ne retombe, sous l'effet
   d'une imitation inconsciente, jusqu'à son niveau héréditaire.</span>
</p>
<p>Combien elle est admirable, cette Loi de la Compensation ! <span
   class="footnote">Et comme elle prouve à merveille le bien-fondé, le
   caractère conforme à la nature, et j'irais presque jusqu'à dire les origines
   divines de la constitution aristocratique qui régit les États de Flatland !
   </span> En utilisant judicieusement cette Loi naturelle, les Polygones et
   les Cercles sont presque toujours en mesure d'étouffer la sédition au
```

```
    berceau : il leur suffit pour cela de mettre à profit les réserves d'espoir
    irrépressibles et illimitées que recèle l'esprit humain.&hellip;
</p>
```

Chacun de ces trois paragraphes inclut une note de bas de page balisée par `<span class="footnote"></span>`. Grâce à ce balisage HTML, nous pouvons conserver le contexte de la note. Dans la feuille de style, une règle CSS applique une police italique aux notes de bas de page (voir Figure 5.4).

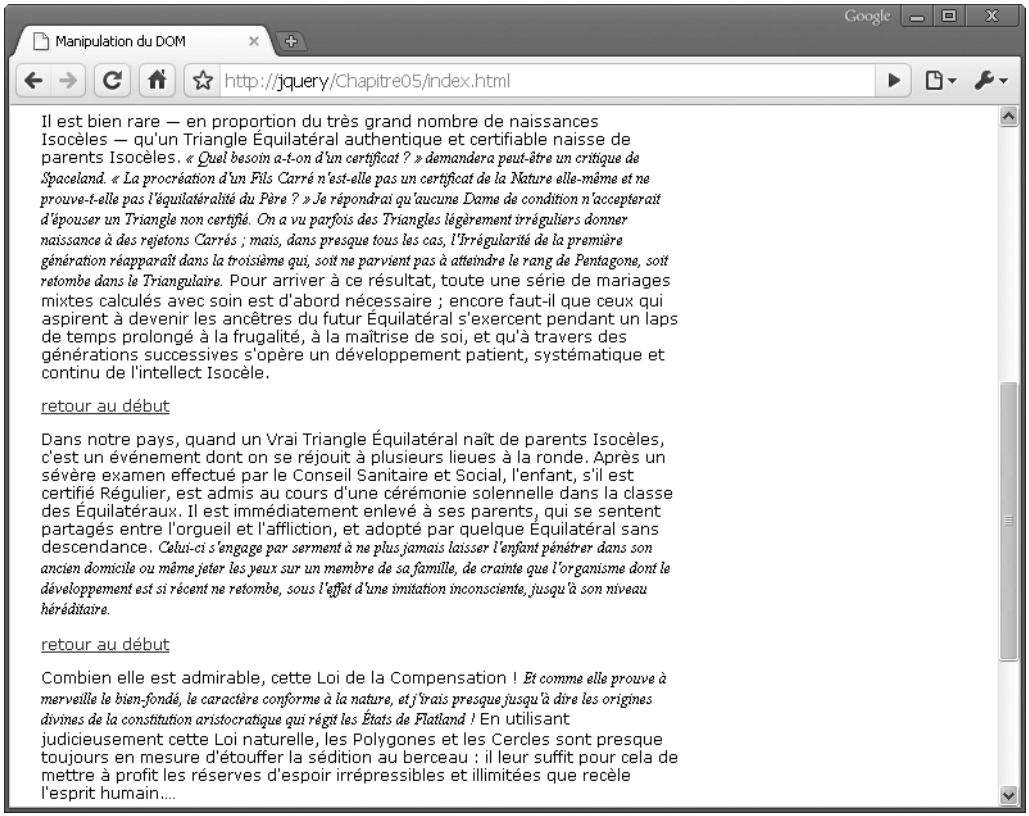


Figure 5.4

Les trois nouveaux paragraphes comprennent des notes de bas de page affichées en italique.

Nous pouvons extraire les notes de bas de page et les déplacer entre les éléments `<div class="chapter">` et `<div id="footer">`. N'oubliez pas que, même en utilisant l'itération implicite, l'ordre des insertions est prédéfini et commence au sommet de l'arborescence du DOM, pour se poursuivre vers le bas. Puisqu'il est important de conserver l'ordre des notes de bas de page à leur nouvel emplacement, nous devons utiliser `.insertBefore('#footer')`.

Chaque note de bas de page est remise juste avant `<div id="footer">`, de manière que la première se trouve entre `<div class="chapter">` et `<div id="footer">`, la deuxième, entre la première et `<div id="footer">`, etc. Si nous utilisions à la place `.insertAfter('div.chapter')`, les notes de bas de page apparaîtraient dans l'ordre inverse. Voici donc notre code :

```
$(document).ready(function() {
  $('span.footnote').insertBefore('#footer');
});
```

Nous faisons cependant face à un problème important. Les notes apparaissent dans des balises `<span>` et sont donc par défaut affichées en ligne, c'est-à-dire l'une après l'autre, sans aucune séparation (voir Figure 5.5).

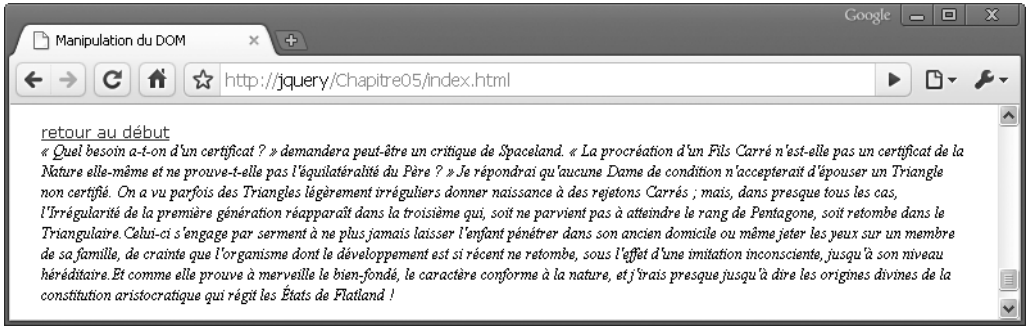


Figure 5.5

Par défaut, les notes de bas de page sont affichées en ligne.

Une solution à ce problème consiste à modifier la feuille de style CSS pour que les éléments `<span>` soient affichés comme des blocs, mais uniquement s'ils ne se trouvent pas dans `<div class="chapter">` :

```
span.footnote {
  font-style: italic;
  font-family: "Times New Roman", Times, serif;
  display: block;
  margin: 1em 0;
}
.chapter span.footnote {
  display: inline;
}
```

La Figure 5.6 présente un affichage plus correct des notes de bas de page.

Les notes de bas de page sont à présent séparées l'une de l'autre, mais nous n'en avons pas terminé. Voici une procédure qui permet d'obtenir une meilleure présentation des notes de bas de page :

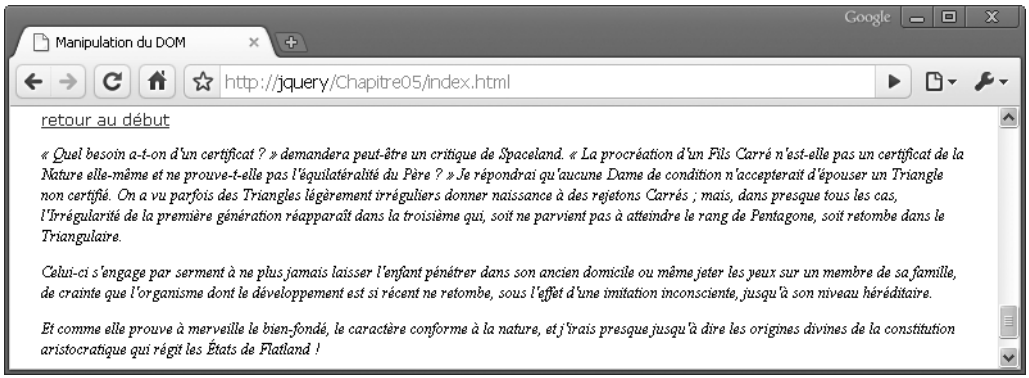


Figure 5.6

Les notes de bas de page sont affichées comme des blocs.

1. Marquer dans le texte l'emplacement de la référence de la note.
2. Numéroter chaque référence et ajouter le numéro correspondant au début de la note elle-même.
3. Créer un lien entre la référence dans le texte et la note de bas de page correspondante, ainsi qu'entre la note de bas de page et l'emplacement de sa référence (pour revenir au texte).

Toutes ces étapes peuvent être accomplies à partir d'une méthode `.each()`, mais nous devons commencer par définir un élément conteneur pour les notes en bas de la page :

```
$(document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
});
```

Puisque les notes de bas de page doivent être numérotées, nous utilisons une liste ordonnée `<ol id="notes"></ol>`, qui générera automatiquement la numérotation à notre place. Nous donnons l'identifiant `notes` à la liste et l'insérons après `<div class="chapter">`.

Marquer, numéroter et lier au contexte

À présent, nous sommes prêts à marquer et à numéroter les emplacements d'où ont été extraites les notes :

```
$(document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
  $('span.footnote').each(function(index) {
    $(this)
      .before(
        ['<a href="#foot-note-',
```

```
            index+1,
            '" id="context-',
            index+1,
            '" class="context">',
            '<sup>' + (index+1) + '</sup>',
            '</a>'
          ].join('')
        )
    });
});
```

Nous partons du sélecteur que nous avons utilisé avec l'exemple plus simple de notes de bas de page, mais nous lui chaînons la méthode `.each()`.

Dans la méthode `.each()`, nous commençons par `$(this)`, qui représente chaque note de bas de page successive et lui chaînons la méthode `.before()`.

Après leur *concaténation*, les éléments du tableau qui se trouve entre les parenthèses de la méthode `.before()` génèrent un lien en exposant qui sera inséré avant chaque élément `<span>` de la note de bas de page. Voici par exemple le premier lien tel qu'il sera ajouté au DOM :

```
<a href="#foot-note-1" id="context-1" class="context"><sup>1</sup></a>
```

Au premier abord, la syntaxe pourrait ne pas vous paraître familière. Voyons donc de manière plus précise ce qui se passe. Entre les parenthèses de la méthode `.before()`, nous plaçons tout d'abord des crochets (`[]`) qui représentent un *tableau littéral*. Chaque élément du tableau est suivi d'une virgule, à l'exception, et c'est important, du dernier. Nous avons placé chaque élément sur sa propre ligne afin de faciliter la lecture. Ensuite, après avoir construit le tableau, nous le convertissons en une chaîne de caractères à l'aide de la méthode JavaScript `.join()`. Son argument est une chaîne vide, représentée par les deux apostrophes, car rien ne doit apparaître entre chaque élément du tableau lorsqu'ils sont concaténés.

Notez l'utilisation de `index+1`. Puisque le comptage commence à zéro, nous ajoutons `1` pour que les attributs `href` commencent à `#foot-note-1`, les attributs `id` à `#context-1` et le texte du lien à `1`. La valeur de l'attribut `href` est particulièrement importante, car elle doit correspondre exactement à celle de l'attribut `id` de la note de bas de page (sans le caractère `#`).

Nous pouvons obtenir le même résultat à l'aide d'une longue chaîne concaténée au lieu d'invoquer `.join()` sur un tableau :

```
.before('<a href="#foot-note-' + (index+1) + '" id="context-' + (index+1) +
        '" class="context"><sup>' + (index+1) + '</sup></a>');
```

Toutefois, l'utilisation du tableau nous paraît donner un code plus facile à gérer.

INFO

Sur le Web, vous trouverez de nombreux débats concernant les différences de performances entre l'invocation de `.join()` sur un tableau et la concaténation des chaînes. Pour les plus curieux, l'article disponible à l'adresse **http://www.sitepen.com/blog/2008/05/09/string-performance-an-analysis/** étudie les performances des deux techniques en se fondant sur plusieurs tests.

Toutefois, dans la plupart des situations, les différences ne sont pas perceptibles. Si les performances d'un script posent problème, il existe bien d'autres facteurs dont l'impact est beaucoup plus important (comme la mise en cache des sélecteurs).

La Figure 5.7 montre les trois marqueurs des notes de bas de page liées.

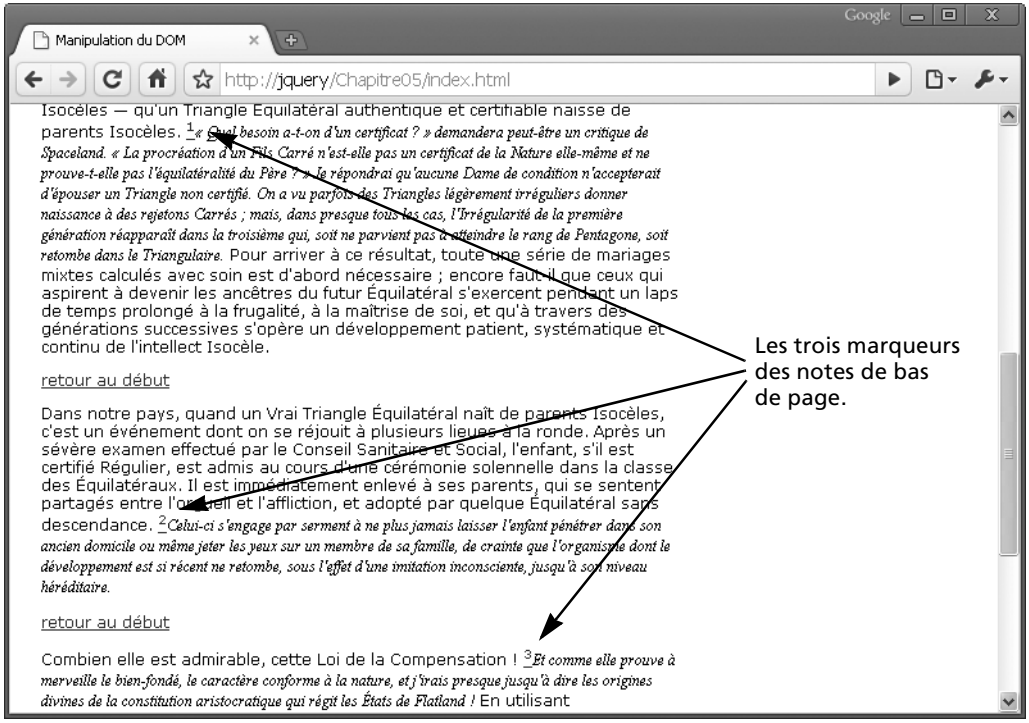


Figure 5.7

Les références des notes sont insérées dans le texte sous forme de liens.

Déplacer les notes de bas de page

L'étape suivante consiste à déplacer les éléments `<span class="footnote">`, comme nous l'avons fait dans l'exemple simple. Cependant, nous les déposerons cette fois-ci

dans le nouvel élément `<ol id="notes">` créé. Nous utilisons `.appendTo()` de manière à conserver l'ordre des notes de bas de page car elles sont insérées successivement à la fin de l'élément :

```
$(document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
  $('span.footnote').each(function(index) {
    $(this)
      .before(
        ['<a href="#foot-note-',
         index+1,
         '" id="context-',
         index+1,
         '" class="context">',
         '<sup>' + (index+1) + '</sup>',
         '</a>'
        ].join('')
      )
      .appendTo('#notes')
  });
});
```

N'oubliez pas que l'appel de la méthode `.appendTo()` est toujours chaîné à `$(this)`. Autrement dit, nous demandons d'ajouter le `<span>` de la note de bas de page à l'élément dont l'identifiant est `'notes'`.

Pour chaque note de bas de page déplacée, nous ajoutons un autre lien qui cible la référence de la note dans le texte :

```
$(document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
  $('span.footnote').each(function(index) {
    $(this)
      .before(
        ['<a href="#foot-note-',
         index+1,
         '" id="context-',
         index+1,
         '" class="context">',
         '<sup>' + (index+1) + '</sup>',
         '</a>'
        ].join('')
      )
      .appendTo('#notes')
      .append(' (<a href="#context-' + (index+1) +'">contexte</a>)');
  });
});
```

L'attribut `href` pointe vers l'identifiant du marqueur correspondant. La Figure 5.8 présente les notes de bas de page avec le lien vers leur contexte.

Il manque aux notes leur numéro. Elles ont été construites à partir d'une liste `<ol>`, mais elles n'ont pas été placées chacune dans leur propre élément `<li>`.

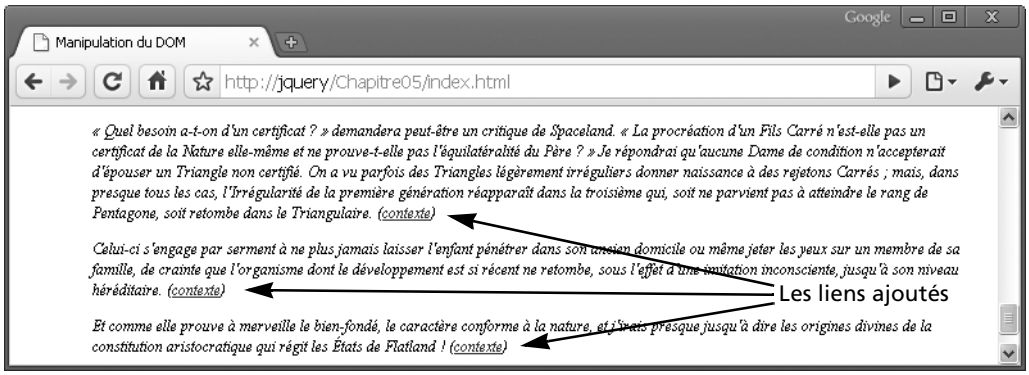


Figure 5.8

Des liens ajoutés aux notes de bas de page permettent de revenir au texte.

5.4 Envelopper des éléments

Pour envelopper des éléments autour d'autres éléments, la principale méthode jQuery se nomme `.wrap()`. Puisque chaque `$(this)` doit être enveloppé par `<li></li>`, nous complétons le code des notes de bas de page de la manière suivante :

```
$(document).ready(function() {
  $('<ol id="notes"></ol>').insertAfter('div.chapter');
  $('span.footnote').each(function(index) {
    $(this)
      .before(
        ['<a href="#foot-note-',
         index+1,
         '" id="context-',
         index+1,
         '" class="context">',
         '<sup>' + (index+1) + '</sup>',
         '</a>'
        ].join('')
      )
      .appendTo('#notes')
      .append(' (<a href="#context-' + (index+1) +'">contexte</a>)')
      .wrap('<li id="foot-note-' + (index+1) + '"></li>');
  });
});
```

À présent, les éléments `<li>` sont complets, avec un attribut `id` qui correspond à l'attribut `href` du marqueur de la note. Nous avons enfin des notes de bas de page numérotées et liées (voir Figure 5.9).

Bien entendu, nous aurions pu ajouter les numéros avant chaque note de bas de page en procédant comme pour leur insertion dans les paragraphes, mais il est très satisfaisant de générer dynamiquement un balisage sémantique avec du code JavaScript.

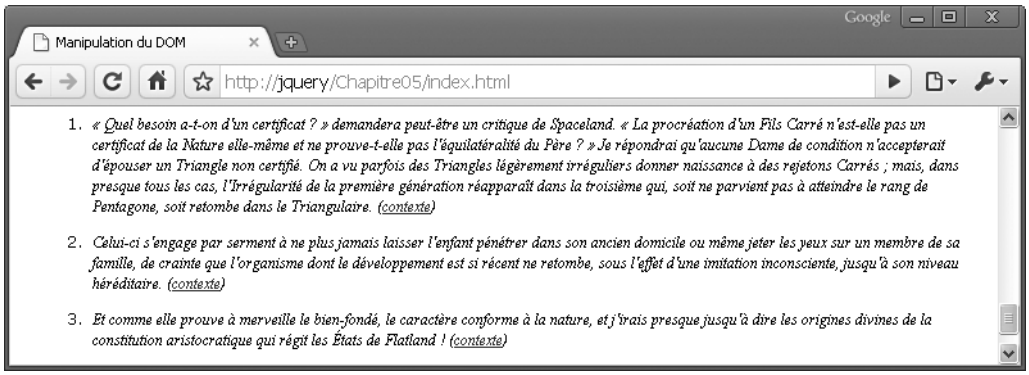


Figure 5.9

La présentation finale des notes de bas de page.

INFO

Les autres méthodes jQuery pour envelopper des éléments sont `.wrapAll()` et `.wrapInner()`. Pour de plus amples informations, consultez les pages **http://docs.jquery.com/Manipulation/wrapAll** et **http://docs.jquery.com/Manipulation/wrapInner**.

5.5 Copier des éléments

Jusque-là, nous avons inséré de nouveaux éléments créés, déplacé des éléments depuis un emplacement dans le document vers un autre et enveloppé de nouveaux éléments autour d'éléments existants. Cependant, il arrive parfois que nous ayons besoin de copier des éléments. Par exemple, un menu de navigation qui apparaît en haut de la page peut être copié et ajouté également en bas de la page. Dès que la présentation visuelle d'une page peut être améliorée par recopie d'éléments, nous pouvons faire intervenir un script. En effet, pourquoi écrire deux fois la même chose, en multipliant d'autant les risques d'erreur, alors que nous pouvons l'écrire une seule fois et laisser jQuery se charger du reste ?

Pour copier des éléments, la méthode `.clone()` de jQuery convient parfaitement. Elle prend une collection d'éléments correspondants et en crée une copie pouvant être utilisée ultérieurement. À l'instar des éléments créés avec du code, les éléments copiés n'apparaîtront pas dans le document tant que nous n'utiliserons pas l'une des méthodes d'insertion.

Par exemple, la ligne suivante crée une copie du premier paragraphe contenu dans `<div class="chapter">` :

```
$('div.chapter p:eq(0)').clone();
```

La Figure 5.10 montre que cela n'affecte en rien le contenu de la page.

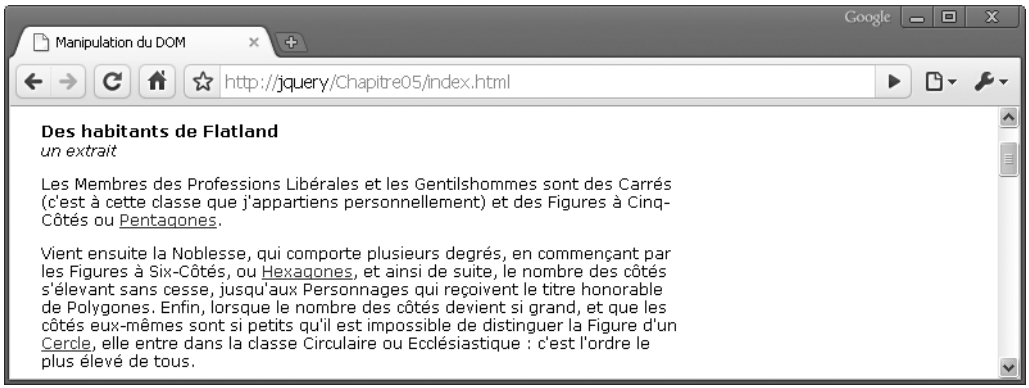


Figure 5.10

La copie d'un élément ne change pas le contenu de la page.

Pour que le paragraphe copié fasse partie du contenu du document, nous pouvons l'insérer avant `<div class="chapter">` :

```
$('div.chapter p:eq(0)').clone().insertBefore('div.chapter');
```

À présent, le premier paragraphe apparaît deux fois et, puisque sa première instance ne se trouve plus dans l'élément `<div class="chapter">`, les styles associés à ce `<div>` ne lui sont pas appliqués, notamment la largeur (voir Figure 5.11).

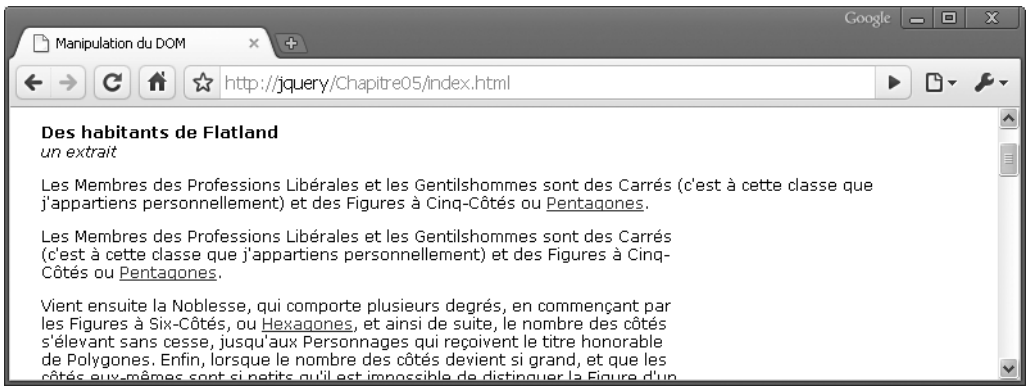


Figure 5.11

Le paragraphe cloné a été ajouté avant le contenu du chapitre.

Pour reprendre une analogie familière à la plupart des gens, `.clone()` est aux méthodes d'insertion ce que "copier" est à "coller".

Cloner avec des événements

Par défaut, la méthode `.clone()` ne copie pas les *événements* liés à l'élément correspondant ni à ses descendants. Cependant, lorsque son seul argument booléen est fixé à `true`, elle clone également les événements : `.clone(true)`. Cette copie des événements est très pratique car nous n'avons plus besoin de lier à nouveau les événements manuellement, comme nous avons pu le faire au Chapitre 3.

Citer des passages

Comme leurs homologues imprimés, de nombreux sites web utilisent des *passages* pour mettre en valeur de petites parties du texte et attirer l'œil du lecteur. Nous pouvons facilement ajouter ce type de citation en utilisant la méthode `.clone()`. Tout d'abord, examinons le troisième paragraphe de notre texte d'exemple :

```
<p>
  <span class="pull-quote">
  <span class="drop">Chez nous, </span>une Loi de la Nature veut qu'un enfant
  mâle ait toujours <strong>un côté de plus</strong> que son père</span>, de
  sorte que chaque génération s'élève (en règle générale) d'un échelon sur la
  voie du progrès et de l'anoblissement. Ainsi le fils d'un Carré sera un
  Pentagone ; le fils du Pentagone, un Hexagone ; etc.
</p>
```

Il commence par `<span class="pull-quote">`. Cette classe sera la cible de notre clonage. Après avoir collé le texte copié du `<span>` à un autre emplacement, nous devons modifier ses propriétés de style pour le distinguer du texte normal.

Petit tour par CSS

Pour obtenir le style souhaité, nous ajoutons la classe `pulled` à la copie de l'élément `<span>`. Voici la règle de style définie pour cette classe :

```
.pulled {
  background: #e5e5e5;
  position: absolute;
  width: 145px;
  top: -20px;
  right: -180px;
  padding: 12px 5px 12px 10px;
  font: italic 1.4em "Times New Roman", Times, serif;
}
```

Le passage extrait reçoit un arrière-plan gris clair, un espacement et une police différente. Mais le plus important est qu'il est positionné de manière absolue, vingt pixels au-dessus et à droite de l'ancêtre positionné (en absolu ou en relatif) le plus proche dans le DOM. Si un positionnement autre que `static` n'est appliqué à aucun ancêtre, le passage de texte sera positionné relativement à l'élément `<body>` du document. C'est pour cette raison que nous devons vérifier dans le code jQuery que la propriété `position` de l'élément parent du passage extrait est fixée à `relative`.

Si le positionnement haut est relativement intuitif, il n'est pas forcément évident de comprendre comment le bloc du passage extrait sera placé à vingt pixels à droite de son parent positionné. Tout d'abord, nous partons de la largeur totale du bloc du passage extrait. Elle correspond à la valeur de la propriété `width` plus l'espacement gauche et droit : 145 px + 5 px + 10 px, c'est-à-dire 160 px. Nous fixons ensuite la propriété `right` du bloc. Une valeur 0 alignerait le bord droit du passage avec celui de son parent. Par conséquent, pour placer son côté gauche à vingt pixels à droite de son parent, nous devons le déplacer dans le sens opposé de plus de vingt pixels par rapport à la largeur totale, c'est-à-dire –180 px.

Retour au code

Revenons à présent à jQuery. Nous commençons par une expression de sélection de tous les éléments `<span class="pull-quote">` et utilisons la méthode `.each()` afin d'effectuer plusieurs opérations sur chacun d'eux :

```
$(document).ready(function() {
  $('span.pull-quote').each(function(index) {
    //...
  });
});
```

Ensuite, nous recherchons le paragraphe parent de chaque passage extrait et ajustons sa propriété CSS `position` :

```
$(document).ready(function() {
  $('span.pull-quote').each(function(index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
  });
});
```

Une fois encore, nous enregistrons dans une variable le sélecteur qui sera utilisé plusieurs fois afin d'améliorer les performances et la lisibilité du code.

Nous sommes à présent certains que le CSS du passage extrait est prêt. Nous pouvons donc cloner chaque `<span>`, ajouter la classe `pulled` à la copie et insérer celle-ci au début du paragraphe :

```
$(document).ready(function() {
  $('span.pull-quote').each(function(index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
    $(this).clone()
      .addClass('pulled')
      .prependTo($parentParagraph);
  });
});
```

Puisque nous choisissons un positionnement absolu du passage extrait, son emplacement dans le paragraphe n'a pas d'importance. Tant qu'il reste à l'intérieur du paragraphe, il sera positionné relativement à ses bords supérieur et droit, conformément à nos règles CSS. En revanche, si nous voulions appliquer un flottement au passage extrait, son emplacement dans le paragraphe affecterait son positionnement vertical.

Le paragraphe, accompagné du passage extrait, est illustré à la Figure 5.12.

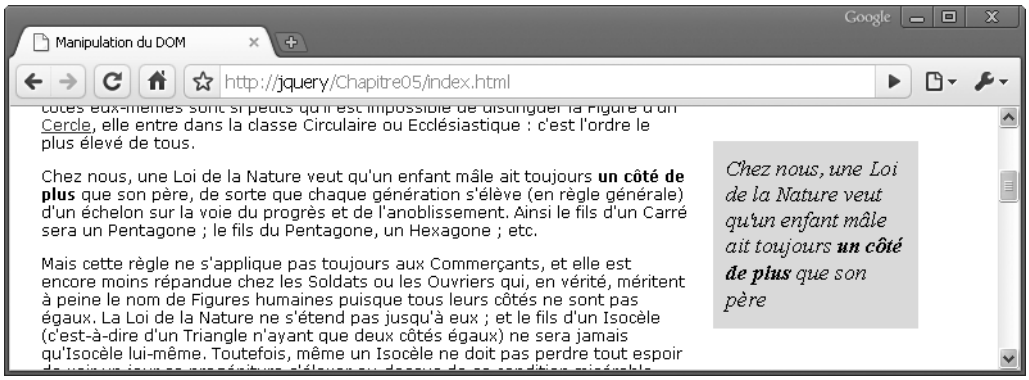


Figure 5.12

Le passage extrait est placé en regard de son élément parent.

C'est un bon début, mais, en général, les passages extraits ne conservent pas la mise en forme du texte, comme c'est le cas dans notre exemple avec les mots "un côté de plus" affichés en gras. Nous voudrions que le texte du `<span class="pull-quote">` soit débarrassé de toute balise du type `<strong>`, `<em>` ou `<a href>`. Par ailleurs, nous aimerions pouvoir modifier légèrement le passage extrait, en retirant certains mots et en les remplaçant par des points de suspension. Pour cela, nous avons placé quelques mots du texte d'exemple dans une balise `<span>` : `<span class="drop">Chez nous, </span>`.

Nous commençons par ajouter les points de suspension, puis nous remplaçons le contenu HTML du passage par une version texte uniquement :

```
$(document).ready(function() {
  $('span.pull-quote').each(function(index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
    var $clonedCopy = $(this).clone();
    $clonedCopy
      .addClass('pulled')
      .find('span.drop')
        .html('&hellip;')
      .end()
      .prependTo($parentParagraph);
    var clonedText = $clonedCopy.text();
```

```
        $clonedCopy.html(clonedText);
    });
});
```

La procédure de clonage débute par l'enregistrement de la copie dans une variable. Cette variable est indispensable car la manipulation de la copie ne peut pas se faire par un seul chaînage de méthodes. Notez également qu'après avoir recherché `<span class="drop">` et avoir remplacé son contenu HTML par des points de suspension (`&hellip;`), nous utilisons `.end()` pour sortir de la requête `.find('span.drop')`. Ainsi, nous insérons l'intégralité de la copie, pas uniquement les points de suspension, au début du paragraphe.

À la fin du code, nous affectons à une autre variable, `clonedText`, le contenu textuel de la copie. Ensuite, nous utilisons ce contenu en remplacement du contenu HTML de la copie. La nouvelle présentation des passages extraits est illustrée à la Figure 5.13.

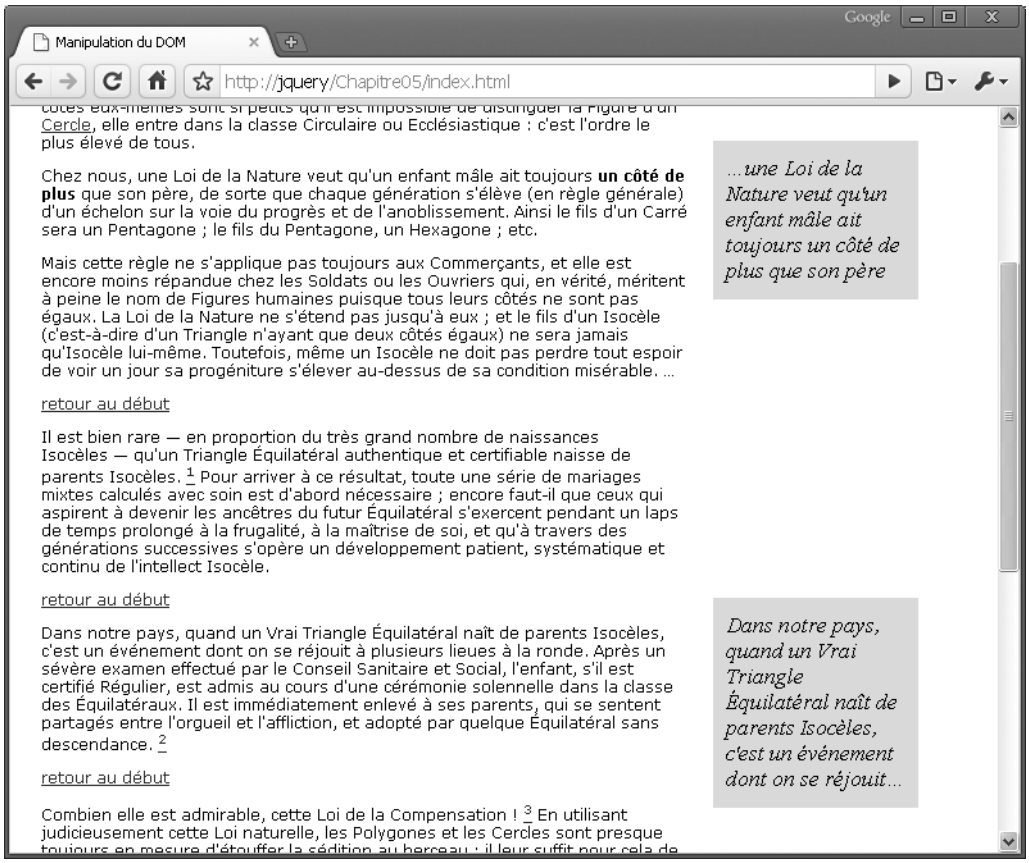


Figure 5.13

Quelques modifications sont apportées au contenu des passages extraits.

Un `<span class="pull-quote">` a également été ajouté à un autre paragraphe pour que le code fonctionne sur plusieurs éléments.

Embellir les passages extraits

L'extraction des passages fonctionne à présent comme voulu, avec suppression des éléments enfants et remplacement d'une partie du texte par des points de suspension.

Cependant, puisque nous avons également pour objectif d'améliorer l'aspect visuel de la page, nous allons ajouter des angles arrondis et une ombre portée aux passages extraits. Mais la hauteur variable des blocs des passages extraits pose problème car elle nous oblige à ajouter deux images d'arrière-plan à un élément, ce qui est impossible avec les navigateurs actuels, à l'exception des versions les plus récentes de Safari.

Pour contourner cette limitation, nous pouvons envelopper les passages dans un autre `<div>` :

```
$(document).ready(function() {
  $('span.pull-quote').each(function(index) {
    var $parentParagraph = $(this).parent('p');
    $parentParagraph.css('position', 'relative');
    var $clonedCopy = $(this).clone();
    $clonedCopy
      .addClass('pulled')
      .find('span.drop')
        .html('&hellip;')
      .end()
      .prependTo($parentParagraph)
      .wrap('<div class="pulled-wrapper"></div>');
    var clonedText = $clonedCopy.text();
    $clonedCopy.html(clonedText);
  });
});
```

Nous devons également modifier les règles CSS pour tenir compte du nouveau `<div>` et des deux images d'arrière-plan :

```
.pulled-wrapper {
  background: url(pq-top.jpg) no-repeat left top;
  position: absolute;
  width: 160px;
  right: -180px;
  padding-top: 18px;
}
.pulled {
  background: url(pq-bottom.jpg) no-repeat left bottom;
  position: relative;
  display: block;
  width: 140px;
  padding: 0 10px 24px 10px;
  font: italic 1.4em "Times New Roman", Times, serif;
}
```

Certaines des règles appliquées précédemment à `<span class="pulled">` le sont à présent à `<div class="pulled-wrapper">`. Nous procédons à quelques ajustements de la largeur et de l'espacement pour tenir compte des bordures de l'image d'arrière-plan. Nous modifions également les propriétés `position` et `display` de la règle `.pulled` afin que la présentation soit correcte sur tous les navigateurs.

La Figure 5.14 présente l'aspect final des passages extraits.

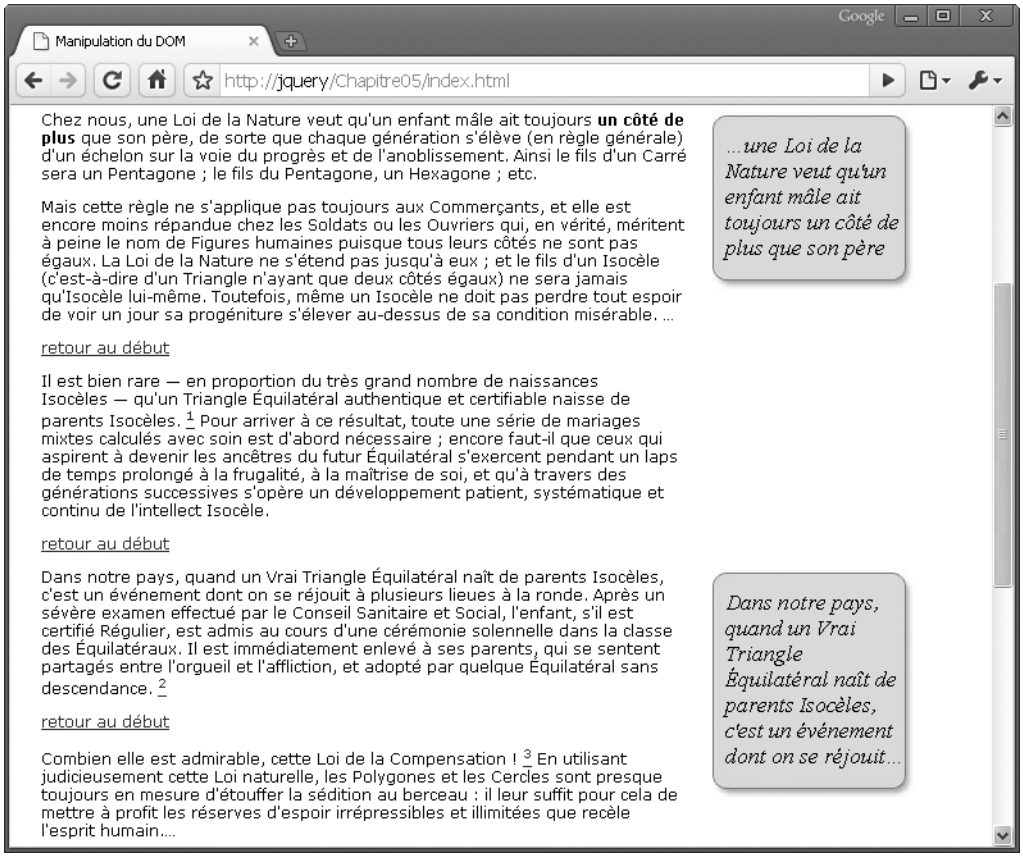


Figure 5.14

L'aspect final des passages extraits.

5.6 Les méthodes du DOM en bref

Les nombreuses méthodes de manipulation du DOM fournies par jQuery varient en termes d'objectifs et d'emplacement. Le récapitulatif suivant vous permettra de déterminer les méthodes que vous pouvez utiliser pour atteindre ces objectifs.

1. Pour *créer* de nouveaux éléments HTML, utilisez la fonction `$()`.
2. Pour *insérer* de nouveaux éléments *dans* chaque élément correspondant, utilisez les méthodes suivantes :
   - `.append()`;
   - `.appendTo()`;
   - `.prepend()`;
   - `.prependTo()`.
3. Pour *insérer* de nouveaux éléments *à côté* de chaque élément correspondant, utilisez les méthodes suivantes :
   - `.after()`;
   - `.insertAfter()`;
   - `.before()`;
   - `.insertBefore()`.
4. Pour *insérer* de nouveaux éléments *autour* de chaque élément correspondant, utilisez les méthodes suivantes :
   - `.wrap()`;
   - `.wrapAll()`;
   - `.wrapInner()`.
5. Pour *remplacer* chaque élément correspondant par de nouveaux éléments ou du texte, utilisez les méthodes suivantes :
   - `.html()`;
   - `.text()`;
   - `.replaceAll()`;
   - `.replaceWith()`.
6. Pour *supprimer* des éléments *dans* chaque élément correspondant, utilisez la méthode suivante :
   - `.empty()`.
7. Pour *retirer* d'un document chaque élément correspondant et ses descendants, sans réellement les supprimer, utilisez la méthode suivante :
   - `.remove()`.

5.7 En résumé

Dans ce chapitre, nous avons créé, copié, réassemblé et embelli du contenu en utilisant les méthodes jQuery de modification du DOM. Nous avons appliqué ces méthodes sur une seule page web, transformant ainsi des paragraphes généraux en un extrait littéraire stylé accompagné de notes de bas de page, de citations de passages et de liens.

La partie didacticiel de l'ouvrage est presque terminée. Toutefois, avant d'étudier des exemples plus complexes, nous allons passer du côté du serveur *via* les méthodes AJAX de jQuery.

6

AJAX

Au sommaire de ce chapitre

- Charger des données à la demande
- Choisir le format des données
- Passer des données au serveur
- Surveiller la requête
- AJAX et les événements
- Questions de sécurité
- Autres solutions
- En résumé

Ces dernières années, il était fréquent de juger les sites selon leur utilisation de certaines technologies. *AJAX* a certainement été l'un des mots les plus en vogue pour décrire de nouvelles applications web. Ce terme était employé dans de nombreux contextes différents, pour englober un ensemble de possibilités et de techniques connexes.

D'un point de vue technique, AJAX est l'acronyme de *Asynchronous JavaScript and XML*. Voici les technologies impliquées dans une solution AJAX :

- JavaScript, pour capturer les actions de l'utilisateur ou d'autres événements du navigateur ;
- l'objet `XMLHttpRequest`, qui permet d'effectuer des requêtes sur le serveur sans interrompre les autres tâches du navigateur ;
- des fichiers au format XML sur le serveur ou dans d'autres formats de données semblables, comme HTML et JSON ;
- du code JavaScript supplémentaire, pour interpréter les données transmises par le serveur et les présenter sur la page.

La technologie AJAX a été accueillie comme le sauveur du Web, car elle a permis de transformer des pages web statiques en applications web interactives. En raison des incohérences dans la mise en œuvre de l'objet `XMLHttpRequest` par les navigateurs, de nombreux frameworks sont apparus pour aider les développeurs à maîtriser son utilisation ; jQuery en fait partie.

Voyons à présent si AJAX peut effectivement faire des miracles.

6.1 Charger des données à la demande

Derrière toute la publicité qu'on a pu lui faire, AJAX n'est qu'un mécanisme permettant de charger des données depuis le *serveur* vers le navigateur web, ou *client*, sans un rafraîchissement de la page. Ces données peuvent prendre plusieurs formes et plusieurs solutions s'offrent à nous pour les traiter lorsqu'elles arrivent. Nous allons illustrer cela en effectuant la même tâche de base de différentes manières.

Nous allons construire une page qui affiche les entrées d'un dictionnaire, regroupées sous la première lettre du terme. Le balisage HTML suivant définit la zone de contenu :

```
<div id="dictionary">
</div>
```

Vous ne rêvez pas, la page est initialement vide. Nous allons nous servir des différentes méthodes AJAX de jQuery pour remplir ce `<div>` avec les entrées du dictionnaire.

Nous aurons besoin d'un système pour déclencher le chargement. Pour cela, nous ajoutons des liens auxquels nous associerons des *gestionnaires d'événements* :

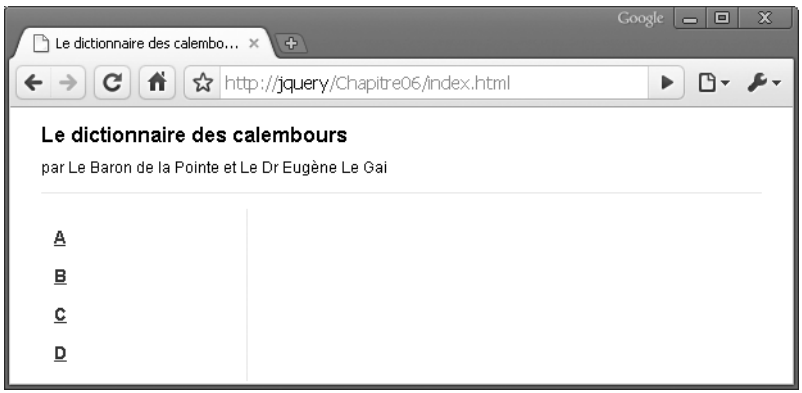
```
<div class="letters">
  <div class="letter" id="letter-a">
   <h3><a href="#">A</a></h3>
  </div>
  <div class="letter" id="letter-b">
    <h3><a href="#">B</a></h3>
  </div>
  <div class="letter" id="letter-c">
    <h3><a href="#">C</a></h3>
  </div>
  <div class="letter" id="letter-d">
    <h3><a href="#">D</a></h3>
  </div>
</div>
```

INFO

Comme toujours, dans le cadre d'une implémentation réelle, il faut utiliser le principe d'*amélioration progressive* de manière que la page soit opérationnelle même en l'absence de JavaScript. Pour simplifier notre exemple, les liens n'auront aucun effet tant que nous ne leur aurons pas attribué des comportements avec jQuery.

Avec quelques règles CSS, la page se présente telle qu'à la Figure 6.1.

Figure 6.1
La page initiale du dictionnaire.



Nous pouvons à présent nous focaliser sur l'obtention d'un contenu pour la page.

Ajouter du contenu HTML

Les applications AJAX ne font souvent que demander au serveur des morceaux de contenu HTML. Cette technique, parfois désignée sous l'acronyme AHAH (*Asynchronous HTTP and HTML*), est très simple à mettre en œuvre avec jQuery. Tout d'abord, nous avons besoin du contenu HTML à insérer. Il est enregistré dans le fichier `a.html` au côté de notre document principal. Voici le début de ce fichier HTML secondaire :

```
<div class="entry">
  <h3 class="term">ACCOMPLI</h3>
  <div class="part">adj.</div>
  <div class="definition">
    On critiquait devant un bon curé ses paroissiennes. « Cependant, »
    répondit-il, « je les vois presque toutes à Complies. »
    </div>
</div>

<div class="entry">
  <h3 class="term">ACROSTICHE</h3>
  <div class="part">n.</div>
  <div class="definition">
    Celui que nous citons ici offre un gracieux jeu de mots.
    <div class="quote">
      <div class="quote-line"><strong>L</strong>ouis est un héros sans peur
        et sans reproche.</div>
      <div class="quote-line"><strong>O</strong>n désire le voir. Aussitôt
        qu'on l'approche,</div>
      <div class="quote-line"><strong>U</strong>n sentiment d'amour enflamme
        tous les cœurs.</div>
      <div class="quote-line"><strong>I</strong>l ne trouve chez nous que des
        adorateurs.</div>
```

```
        <div class="quote-line"><strong>S</strong>on image est partout, excepté
          dans ma poche.</div>
      </div>
    </div>
  </div>

  <div class="entry">
    <h3 class="term">ADMIRATION</h3>
    <div class="part">n.</div>
    <div class="definition">
      On demandait à des soldats en campagne si on pouvait compter sur eux et
      quel était le thermomètre de leur enthousiasme. L'un d'eux répondit :
      « Nous en sommes à <em>la demi-ration</em>. »
    </div>
  </div>
```

La page se poursuit avec d'autres entrées dans cette structure HTML. Sans mise en forme appropriée, l'aspect de cette page est plutôt brut[1] (voir Figure 6.2).

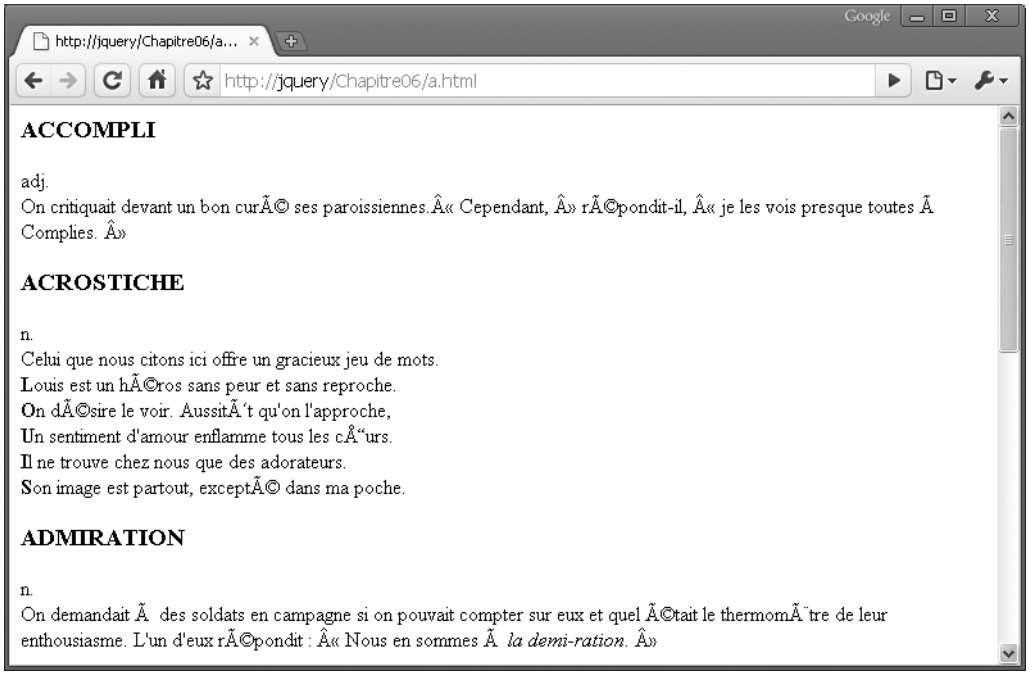


Figure 6.2

Affichage rudimentaire d'un contenu HTML.

1. N.d.T. : les caractères accentués ne sont pas affichés correctement car la structure HTML seule ne précise pas l'encodage des caractères adéquat.

Cette présentation est due au fait que le fichier `a.html` ne constitue pas un véritable document HTML : il ne contient aucune des balises `<html>`, `<head>` ou `<body>` habituellement requises. Ce type de fichier est appelé *extrait* ou *fragment*. Il n'existe que pour être inséré dans un autre document HTML :

```
$(document).ready(function() {
  $('#letter-a a').click(function() {
   $('#dictionary').load('a.html');
   return false;
  });
});
```

La méthode `.load()` s'occupe de tous les détails à notre place. Nous indiquons la cible du fragment HTML en utilisant un sélecteur jQuery classique et passons en argument de cette méthode l'URL du fichier à charger. Un clic sur le premier lien charge le fichier et le place dans `<div id="dictionary">`. Le navigateur affiche le nouveau contenu HTML après son insertion (voir Figure 6.3).

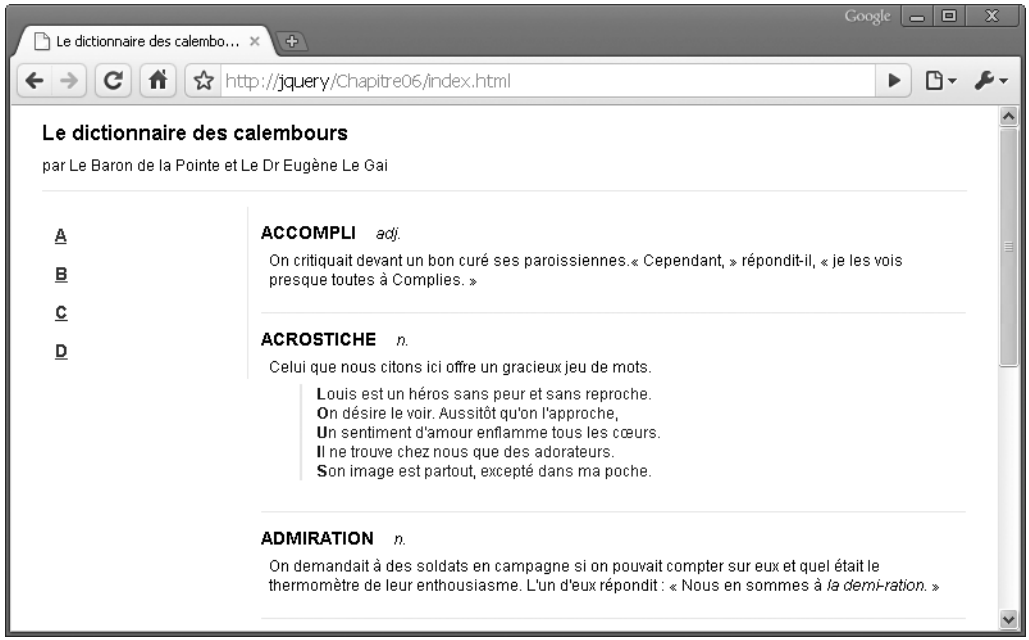


Figure 6.3

Affichage du fragment HTML après insertion dans un document HTML.

Le fragment HTML n'est plus brut, mais stylé. En effet, les règles CSS du document principal sont immédiatement appliquées à cet extrait dès qu'il est inséré.

Si vous testez cet exemple, les définitions du dictionnaire apparaîtront probablement instantanément après avoir cliqué sur le lien. C'est l'un des dangers du test local des applications. Il est difficile de tenir compte des délais de transmission des documents au travers du réseau. Supposons que nous ajoutions une boîte d'alerte qui s'affiche après que les définitions sont chargées :

```
$(document).ready(function() {
  $('#letter-a a').click(function() {
    $('#dictionary').load('a.html');
    alert('Définitions chargées !');
    return false;
  });
});
```

La structure de ce code nous laisse supposer que le message est affiché uniquement lorsque le chargement des définitions est terminé. En effet, l'exécution d'un code JavaScript se fait généralement de manière *synchrone*, c'est-à-dire une tâche après l'autre.

Cependant, lorsque ce code particulier est testé sur un serveur web en production, le message d'alerte peut être affiché et avoir disparu avant que le changement ne soit terminé, en raison de la latence du réseau. En effet, tous les appels AJAX sont par défaut *asynchrones*. Si ce n'était pas le cas, cette technologie se serait nommée SJAX. Un chargement asynchrone signifie qu'une fois que la requête HTTP d'obtention du fragment HTML a été émise, l'exécution du script reprend immédiatement sans attendre la réponse. Plus tard, le navigateur reçoit la réponse du serveur et la prend en charge. Ce comportement est généralement celui attendu ; il est peu convivial de bloquer le navigateur web en attendant l'arrivée des données.

Si des actions doivent être reportées à la fin du chargement, jQuery fournit pour cela une *fonction de rappel*. Nous en verrons un exemple plus loin.

Manipuler des objets JavaScript

Si recevoir à la demande du HTML intégralement balisé se révèle très pratique, il se peut que nous voulions traiter les données avant qu'elles ne soient affichées. Dans ce cas, nous devons les obtenir sous forme d'une structure manipulable avec du code JavaScript.

Grâce aux sélecteurs de jQuery, nous pouvons parcourir le balisage HTML reçu et le manipuler, mais il doit tout d'abord être inséré dans le document. En utilisant un format de données JavaScript natif, le code nécessaire sera plus réduit.

Obtenir un objet JavaScript

Nous l'avons vu, les *objets JavaScript* ne sont que des ensembles de *couples clé-valeur* qui peuvent être définis succinctement à l'aide d'accolades (`{}`). Quant aux *tableaux*

JavaScript, ils sont définis dynamiquement avec des crochets (`[]`). En combinant ces deux concepts, nous pouvons facilement exprimer des structures de données très complexes et riches.

Le terme JSON (*JavaScript Object Notation*) a été inventé par Douglas Crockford pour désigner cette syntaxe simple. Elle offre une alternative concise au format XML parfois volumineux :

```
{
  "cle": "valeur",
  "cle2": [
    "tableau",
    "d'",
    "éléments"
  ]
}
```

INFO

Pour de plus amples informations concernant les avantages potentiels de JSON, ainsi que des exemples de mise en œuvre dans différents langages de programmation, consultez le site **http://json.org/**.

Il existe différentes manières d'encoder les données dans ce format. Dans notre exemple, nous plaçons les entrées du dictionnaire dans un fichier JSON nommé `b.json`, dont voici le début :

```
[
  {
    "term": "BADAUD",
    "part": "n.",
    "definition": "L'expression de badaud qu'on applique aux Parisiens comme
      une injure, vient, dit-on, d'un mot celtique qui signifie batelier, parce
      que les Parisiens faisaient autrefois un grand commerce par eau. La ville
      de Paris porte même un navire pour armoiries.",
    "quote": [
      "De peur d'offenser sa patrie,",
      "Journel, mon imprimeur, digne enfant de Paris,",
      "Ne veut rien imprimer sur la badauderie.",
      "Journel est bien de son pays."
    ],
    "author": "Ménage"
  },
  {
    "term": "BOITER",
    "part": "v. tr.",
    "definition": "Un homme qui venait d'acheter un cheval, à la vue, reconnut
      qu'il était boiteux et voulut résilier son marché. Mais le vendeur
      témoigna qu'il l'avait prévenu de ce défaut, en lui disant : Il boite et
      mange bien."
  },
```

```
{
  "term": "BONHEUR",
  "part": "n.",
  "definition": "Le moyen d'être heureux en ménage, c'est de se marier au
    point du jour, parce qu'alors on est sûr d'avoir fait un mariage de
    bonne heure."
},
```

Pour obtenir ces données, nous utiliserons la méthode `$.getJSON()`. Elle récupère le fichier, le traite et retourne au code appelant l'objet JavaScript résultant.

Fonctions jQuery globales

Jusqu'à présent, toutes les méthodes jQuery que nous avons employées sont associées à un objet jQuery construit par la fonction `$()`. Les sélecteurs nous permettent de préciser les nœuds du DOM sur lesquels nous souhaitons travailler en invoquant des méthodes. En revanche, la fonction `$.getJSON()` est différente. Elle ne s'applique logiquement à aucun élément du DOM ; l'objet résultant est fourni au script, non injecté dans la page. C'est pourquoi `getJSON()` est une méthode de l'*objet jQuery global* (un objet unique appelé `jQuery` ou `$` défini une seule fois par la bibliothèque jQuery), à la place d'une *instance d'objet jQuery* individuelle (les objets créés par la fonction `$()`).

Si, comme dans d'autres langages orientés objet, la notion de classe existait en JavaScript, `$.getJSON()` serait une *méthode de classe*. Pour nous, ce type de méthode est une *fonction globale*, c'est-à-dire une fonction qui utilise l'*espace de noms* de jQuery pour ne pas entrer en conflit avec d'autres noms de fonctions.

Pour utiliser cette fonction, nous lui fournissons le nom du fichier :

```
$(document).ready(function() {
  $('#letter-b a').click(function() {
    $.getJSON('b.json');
    return false;
  });
});
```

En cliquant sur le lien correspondant, ce code ne semble avoir aucun effet. L'appel de la fonction charge le fichier, mais les données résultantes ne sont pas utilisées. Pour cela, nous devons employer une *fonction de rappel*.

La fonction `$.getJSON()` prend un second argument qui désigne la fonction invoquée lorsque le chargement est terminé. Nous l'avons mentionné précédemment, les appels AJAX sont *asynchrones* et la fonction de rappel constitue une technique pour attendre les données transmises au lieu d'exécuter le code directement. La fonction de rappel prend également un argument, qui contient les données résultantes. Nous pouvons donc écrire le code suivant :

```
$(document).ready(function() {
  $('#letter-b a').click(function() {
```

```
    $.getJSON('b.json', function(data) {
    });
    return false;
  });
});
```

Dans ce code, nous utilisons une fonction de rappel *anonyme*, comme nous en avons l'habitude pour un code jQuery concis. Toutefois, rien n'empêche de passer une fonction nommée.

Dans la fonction, nous pouvons utiliser la variable `data` pour examiner la structure de données. Nous devons parcourir le tableau de premier niveau, en construisant le contenu HTML qui correspond à chaque élément. Pour cela, une boucle `for` classique suffit, mais nous allons profiter de cette occasion pour présenter une autre fonction globale utile : `$.each()`. Nous avons rencontré son homologue au Chapitre 5, la méthode `.each()`. Au lieu d'opérer sur un objet jQuery, cette fonction prend en premier argument un tableau ou une mappe, et une fonction de rappel en second argument. À chaque tour de boucle, l'*indice d'itération* et l'élément courants du tableau ou de la mappe sont passés en paramètres à la fonction de rappel :

```
$(document).ready(function() {
  $('#letter-b a').click(function() {
    $.getJSON('b.json', function(data) {
      $('#dictionary').empty();
      $.each(data, function(entryIndex, entry) {
        var html = '<div class="entry">';
        html += '<h3 class="term">' + entry['term'] + '</h3>';
        html += '<div class="part">' + entry['part'] + '</div>';
        html += '<div class="definition">';
        html += entry['definition'];
        html += '</div>';
        html += '</div>';
        $('#dictionary').append(html);
      });
    });
    return false;
  });
});
```

Avant la boucle, nous vidons l'élément `<div id="dictionary">` de manière à le remplir avec notre nouveau contenu HTML construit. Ensuite, nous appelons `$.each()` pour examiner chaque élément et construire une structure HTML à partir de son contenu. Enfin, nous convertissons ce balisage HTML en une arborescence DOM en l'ajoutant dans la balise `<div>`.

INFO

Cet exemple suppose que les données se prêtent parfaitement à un balisage HTML. Autrement dit, elles ne doivent pas contenir certains caractères spéciaux, comme <.

Il ne nous reste plus qu'à prendre en charge les entrées contenant des citations, ce que nous faisons avec une autre boucle `$.each()` :

```
$(document).ready(function() {
  $('#letter-b a').click(function() {
    $.getJSON('b.json', function(data) {
      $('#dictionary').empty();
      $.each(data, function(entryIndex, entry) {
        var html = '<div class="entry">';
        html += '<h3 class="term">' + entry['term'] + '</h3>';
        html += '<div class="part">' + entry['part'] + '</div>';
        html += '<div class="definition">';
        html += entry['definition'];
        if (entry['quote']) {
          html += '<div class="quote">';
          $.each(entry['quote'], function(lineIndex, line) {
            html += '<div class="quote-line">' + line + '</div>';
          });
          if (entry['author']) {
            html += '<div class="quote-author">' + entry['author'] + '</div>';
          }
          html += '</div>';
        }
        html += '</div>';
        html += '</div>';
        $('#dictionary').append(html);
      });
    });
    return false;
  });
});
```

Lorsque ce code est en place, nous cliquons sur le lien B et confirmons le bon fonctionnement de notre code (voir Figure 6.4).

INFO

Le format JSON est concis, mais peu indulgent. Chaque crochet, accolade, apostrophe et virgule doit être présent et justifié, ou le fichier ne pourra pas être chargé. Dans la plupart des navigateurs, vous ne recevrez aucun message d'erreur. Le script échouera en silence.

Exécuter un script

Parfois, nous ne souhaitons pas charger tout le code JavaScript nécessaire dès le chargement de la page. Il se peut même que nous ne sachions pas quels sont les scripts utiles tant que l'utilisateur n'a pas effectué certaines actions. Nous pourrions ajouter dynamiquement des balises `<script>` au fur et à mesure des besoins, mais une solution plus élégante pour injecter le code supplémentaire consiste à laisser jQuery charger directement le fichier `.js`.

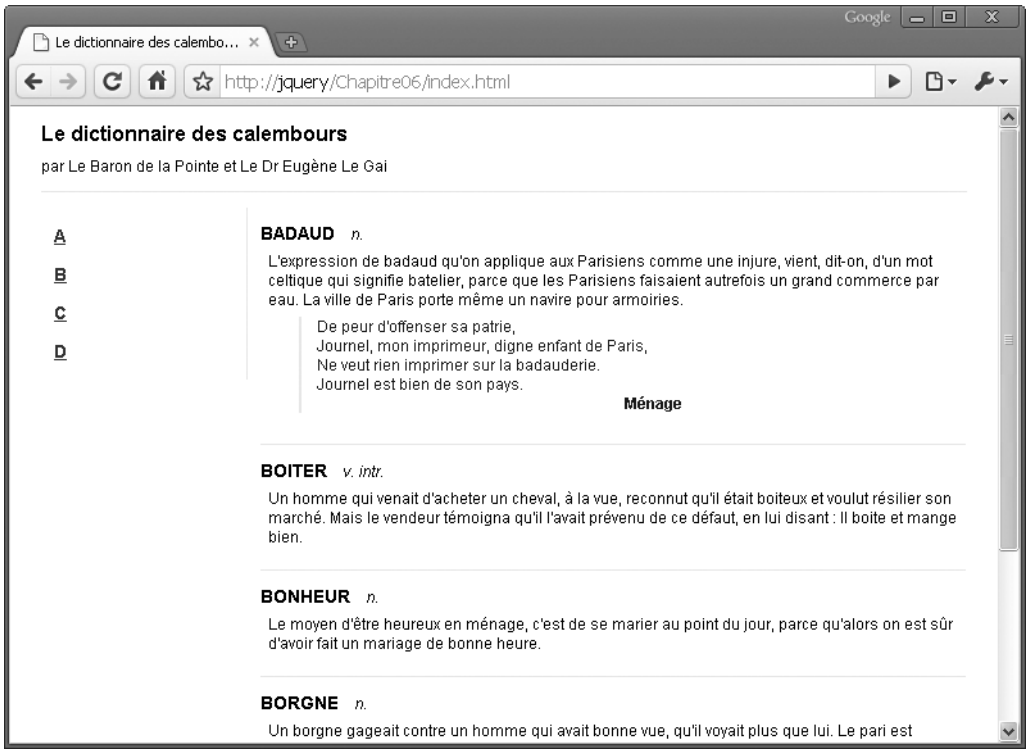


Figure 6.4

Chargement des entrées enregistrées dans une structure JSON.

Le chargement d'un script est aussi simple que le chargement d'un fragment HTML. Nous utilisons la fonction globale `$.getScript()`, qui, comme son homologue, prend en argument l'URL du fichier du script :

```
$(document).ready(function() {
  $('#letter-c a').click(function() {
    $.getScript('c.js');
    return false;
  });
});
```

Dans l'exemple précédent, nous devions traiter les données obtenues afin d'exploiter le fichier chargé. Avec un fichier de script, la procédure est automatique : le script est simplement exécuté.

Les scripts obtenus de cette manière sont exécutés dans le contexte global de la page en cours. Cela signifie qu'ils ont accès à toutes les fonctions et variables définies globalement, notamment celles de jQuery. Nous pouvons donc imiter l'exemple JSON pour préparer et insérer du contenu HTML sur la page lors de l'exécution du script et placer ce code dans `c.js` :

```
var entries = [
  {
    "term": "CHAIR",
    "part": "n.",
    "definition": "M. Ch. Monselet, après avoir entendu M. Saisset,..."
  },
  {
    "term": "CHALEUR",
    "part": "n.",
    "definition": "Bautru se promenait le chapeau à la main, par un soleil..."
  },
  {
    "term": "CHANT",
    "part": "n.",
    "definition": "On dit que le rossignol ne chante plus lorsqu'il est..."
  },
  {
    "term": "CHAUDRON",
    "part": "n.",
    "definition": "Brunet disait que le vase qu'on appelle chaudron..."
  },
  {
    "term": "CHER",
    "part": "n.",
    "definition": "C'est un département où l'on ne peut pas vivre..."
  },
  {
    "term": "COMBLÉ",
    "part": "adj.",
    "definition": "Un très-gros homme, arrêté au bord d'un fossé, disait :..."
  }
];

var html = '';

$.each(entries, function() {
  html += '<div class="entry">';
  html += '<h3 class="term">' + this['term'] + '</h3>';
  html += '<div class="part">' + this['part'] + '</div>';
  html += '<div class="definition">' + this['definition'] + '</div>';
  html += '</div>';
});

$('#dictionary').html(html);
```

La Figure 6.5 montre le résultat que l'on obtient en cliquant sur le lien C.

Charger un document XML

XML fait partie de l'acronyme AJAX et, pourtant, nous n'avons pas encore chargé un document XML. Néanmoins, cette procédure est simple et reprend assez fidèlement la technique JSON. Tout d'abord, nous avons besoin d'un fichier XML, `d.xml`, qui contient les données à afficher :

Figure 6.5

Le contenu est inséré en chargeant un script.

```
<?xml version="1.0" encoding="UTF-8"?>
<entries>
  <entry term="DÉCRET" part="n.">
    <definition>
     Un savant prétend que les mots décret et décréter ont été inventés par
     Minos roi et législateur de la Crète.
    </definition>
  </entry>
  <entry term="DÉSASTRE" part="n.">
    <definition>
      Dans les derniers jours du Directoire, on trouva un matin, sur la porte
      du Luxembourg, un magnifique soleil fraîchement peint, et portant au
      milieu de ses rayons ce seul mot<![CDATA[ ]]>: la République. Les
      Parisiens comprirent le rébus, et tout le monde le lisait. (La République
      dans le plus grand des astres.)
    </definition>
  </entry>
  <entry term="DESCARTES" part="n.">
    <definition>
      Le marquis de Saint-Aulaire, qui, à la fin du XVIIe siècle, fit les
      délices de la cour de la duchesse du Maine, fut prié un jour par cette
      princesse de lui expliquer le système de Newton. La sachant zélée
      cartésienne, le spirituel marquis éluda la question en improvisant ce
      petit couplet sur l'air des fraises :
    </definition>
    <quote author="Marquis de Saint-Aulaire">
      <line>Princesse, détachons-nous</line>
      <line>De Newton, de Descartes ;</line>
      <line>Ces deux espèces de fous</line>
      <line>N'ont jamais vu le dessous</line>
```

```
      <line>Des cartes,</line>
      <line>Des cartes,</line>
      <line>Des cartes.</line>
    </quote>
  </entry>
  <entry term="DÉTESTABLE" part="adj.">
    <definition>
      On s'est réjoui beaucoup en 1858 d'avoir une si grande suite de jours
      d'été stables.
    </definition>
  </entry>
</entries>
```

Bien évidemment, ces données peuvent être fournies de différentes manières, certaines étant plus proches de la structure établie pour les exemples HTML et JSON précédents. Toutefois, nous illustrons ici certaines caractéristiques de XML qui permettent à ce format d'être plus facile à lire, par exemple en utilisant des *attributs* pour les termes à la place des balises.

Notre fonction débute de manière semblable aux précédentes :

```
$(document).ready(function() {
  $('#letter-d a').click(function() {
    $.get('d.xml', function(data) {
    });
    return false;
  });
});
```

Cette fois-ci, le travail est réalisé par la fonction `$.get()`. En général, elle récupère simplement le fichier indiqué par l'URL et passe le texte brut correspondant à la fonction de rappel. Toutefois, si le *type MIME* fourni par le serveur indique que la réponse est au format XML, la fonction de rappel manipulera l'arborescence DOM correspondante.

Par chance, jQuery offre de nombreuses fonctions de parcours du DOM. Nous pouvons employer les méthodes `.find()`, `.filter()` et autres sur le document XML comme nous le ferions sur du contenu HTML :

```
$(document).ready(function() {
  $('#letter-d a').click(function() {
    $.get('d.xml', function(data) {
      $('#dictionary').empty();
      $(data).find('entry').each(function() {
        var $entry = $(this);
        var html = '<div class="entry">';
        html += '<h3 class="term">' + $entry.attr('term') + '</h3>';
        html += '<div class="part">' + $entry.attr('part') + '</div>';
        html += '<div class="definition">';
        html += $entry.find('definition').text();
        var $quote = $entry.find('quote');
        if ($quote.length) {
          html += '<div class="quote">';
          $quote.find('line').each(function() {
```

```
            html += '<div class="quote-line">' + $(this).text() + '</div>';
          });
          if ($quote.attr('author')) {
            html += '<div class="quote-author">'
              + $quote.attr('author') + '</div>';
          }
          html += '</div>';
        }
        html += '</div>';
        html += '</div>';
        $('#dictionary').append($(html));
      });
    });
    return false;
  });
});
```

La Figure 6.6 montre qu'un clic sur le lien D produit le résultat escompté.

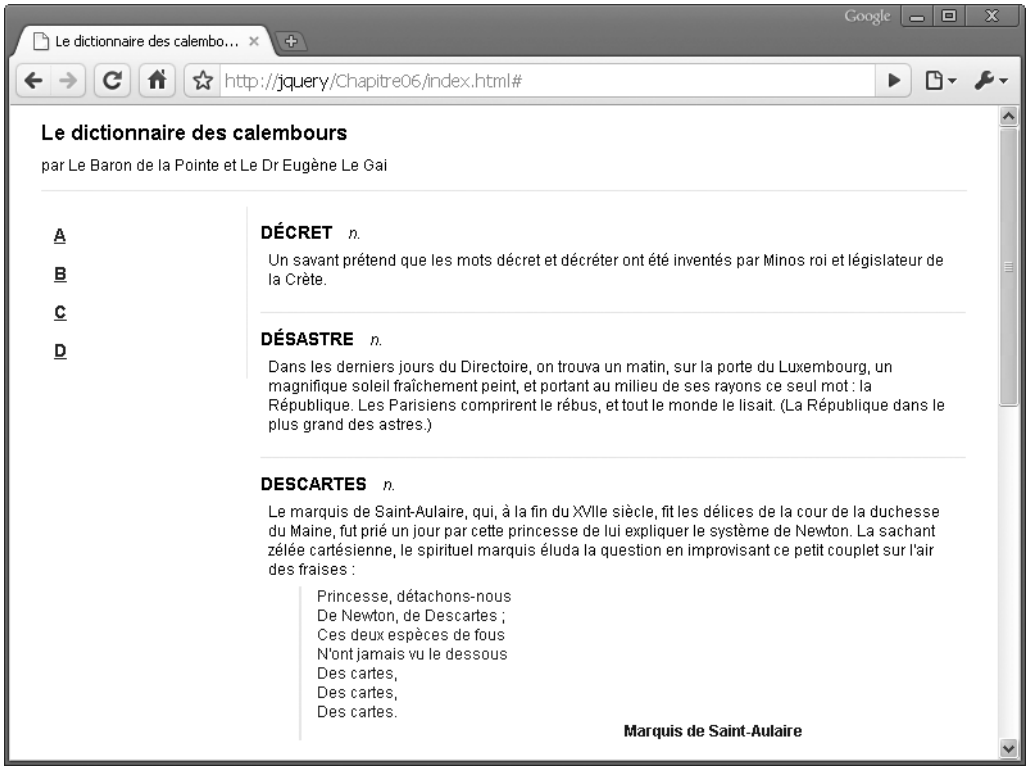


Figure 6.6

Insertion d'un contenu XML.

Il s'agit d'une nouvelle utilisation des méthodes de parcours du DOM que nous avons déjà employées, soulignant, si c'était encore nécessaire, la flexibilité de la prise en charge des sélecteurs CSS par jQuery. La syntaxe CSS sert généralement à embellir les pages HTML, et les sélecteurs définis dans les fichiers `.css` standard utilisent des noms de balises HTML comme `div` et `body` pour localiser du contenu. En revanche, avec jQuery, nous pouvons employer des noms de balises XML quelconques, comme `entry` et `definition` dans cet exemple, en plus des balises HTML standard.

Le moteur de sélection élaboré de jQuery facilite également la localisation de parties du document XML pour des cas plus complexes. Par exemple, supposons que nous souhaitions limiter l'affichage aux entrées qui contiennent des citations et leur auteur. Pour ne sélectionner que les entrées qui comprennent des éléments `<quote>`, nous remplaçons `entry` par `entry:has(quote)`. Pour retenir uniquement les entrées dont les éléments `<quote>` possèdent un attribut `author`, nous écrivons `entry: has(quote[author])`. La ligne contenant le sélecteur initial devient alors :

```
$(data).find('entry:has(quote[author])').each(function() {
```

Cette nouvelle expression de sélection restreint les entrées sélectionnées (voir Figure 6.7).

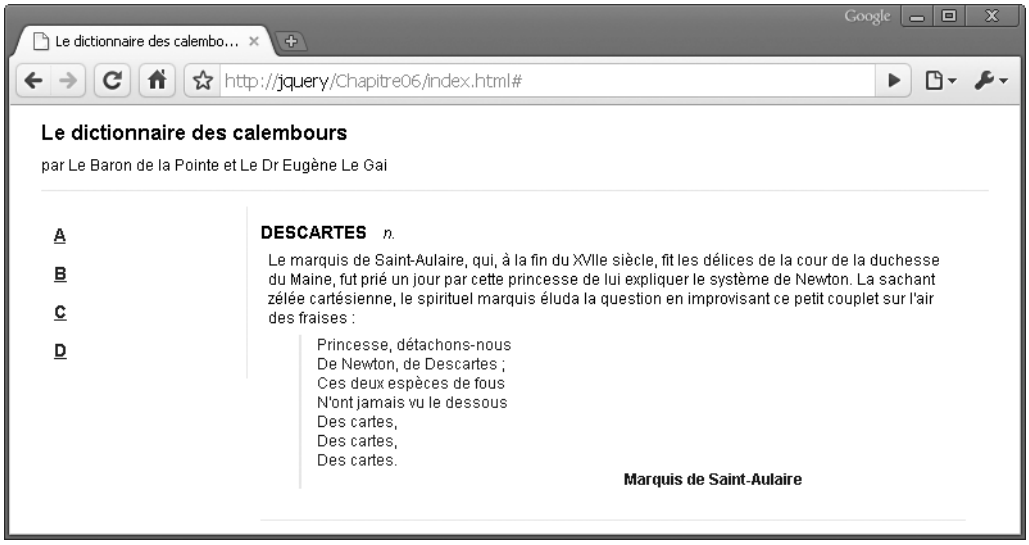


Figure 6.7

Uniquement les entrées comprenant une citation avec son auteur.

6.2 Choisir le format des données

Nous avons examiné quatre formats pour les données externes, chacun pouvant être traité par les fonctions AJAX natives de jQuery. Nous avons également vérifié que ces quatre formats permettent de répondre aux besoins, en chargeant des informations dans une page existante lorsque l'utilisateur les demande, non avant. Se pose donc naturellement la question du choix du format à retenir dans nos applications.

Les *fragments HTML* demandent très peu de travail. Les données externes peuvent être chargées et insérées dans la page à l'aide d'une seule méthode simple, qui n'implique aucune fonction de rappel. Pour ajouter le nouveau contenu HTML à la page existante, aucun parcours des données n'est nécessaire. En revanche, la structure des données d'une application ne convient pas nécessairement à d'autres applications. Le fichier externe est fortement couplé au conteneur ciblé.

Les *fichiers JSON* ont une structure qui facilite la réutilisation. Ils sont concis et faciles à lire. Il faut parcourir la structure de données pour en extraire les informations et les présenter sur la page, mais cette procédure peut être mise en œuvre avec des techniques JavaScript standard. Puisque les fichiers peuvent être parsés par un simple appel à la fonction JavaScript `eval()`, la lecture d'un fichier JSON est extrêmement rapide. Toutefois, toute utilisation d'`eval()` présente des risques. Des erreurs dans le fichier JSON conduisent à un échec silencieux ou à des effets secondaires sur la page. Par conséquent, les données doivent être préparées soigneusement par un tiers de confiance.

Les *fichiers JavaScript* offrent la souplesse maximale, mais ne constituent pas un véritable mécanisme de stockage de données. Puisque les fichiers sont propres au langage, ils ne peuvent pas servir à fournir des informations à des systèmes disparates. En revanche, grâce à cette possibilité de charger un fichier JavaScript, les comportements rarement nécessaires peuvent être placés dans des fichiers séparés pour n'être chargés qu'en cas de besoin. La taille du code s'en trouve ainsi réduite.

Les *documents XML* sont les rois de la portabilité. Puisque XML est devenu la *lingua franca* des services web, l'utilisation de ce format permet de réutiliser les données ailleurs. Par exemple, Flickr (**http://flickr.com/**), del.icio.us (**http://del.icio.us/**) et Upcoming (**http://upcoming.org/**) exportent des représentations XML de leurs données, ce qui a permis de développer de nombreuses applications tierces à partir de ces informations. Cependant, le format XML est quelque peu volumineux et risque d'être plus lent à analyser et à manipuler que d'autres solutions.

Si l'on considère toutes ces caractéristiques, il est plus facile de fournir des données externes sous forme de fragments HTML, tant qu'elles n'ont pas à être employées dans d'autres applications. Si les données doivent être réutilisées dans d'autres applications dont le code est disponible, JSON se révèle souvent un bon choix en raison de ses per-

formances et de sa taille. Lorsque l'application distante est inconnue, XML constitue la meilleure garantie d'interopérabilité.

Plus que toute autre considération, il faut savoir si les données sont déjà disponibles. Dans l'affirmative, il est probable qu'elles se trouvent déjà dans l'un de ces formats et le choix est alors tout fait.

6.3 Passer des données au serveur

Les exemples précédents ont expliqué comment obtenir des fichiers de données statiques à partir du serveur web. Cependant, la technique AJAX est encore plus intéressante lorsque le serveur peut constituer *dynamiquement* les données en fonction d'informations fournies par le navigateur. Sur ce point, nous pouvons également compter sur jQuery. Toutes les méthodes décrites jusqu'à présent peuvent être modifiées pour que le transfert des données soit bidirectionnel.

INFO

Puisque l'illustration de cette technique implique une interaction avec le serveur web, nous allons devoir utiliser pour la première fois du code côté serveur. Les exemples présentés se fondent sur *PHP*, un langage de script largement employé et disponible gratuitement. Nous n'expliquerons pas comment configurer le serveur web avec PHP. Pour cela, vous pouvez consulter les sites web d'Apache (**http://apache.org/**), de PHP (**http://php.net/**) ou de la société qui héberge votre site.

Effectuer une requête GET

Pour servir d'exemple à la communication entre un client et un serveur, nous allons développer un script qui envoie une seule entrée du dictionnaire au navigateur à chaque requête. L'entrée choisie dépendra d'un paramètre transmis par le navigateur. Le script prend l'entrée correspondante à partir d'une structure de données semblable à la suivante :

```
<?php
$entries = array(
  'ÉCRIRE' => array(
    'part' => 'v. tr.',
    'definition' => "Un savant, connu par un nasillement extraordinaire,
      assistait à la lecture d'un ouvrage historique. -- Cet ouvrage est mal
      écrit, s'écria-t-il, un style prétentieux, plein d'affectation ! Il faut
      avant tout écrire comme on parle. -- C'est fort bien, dit un ami de
      l'auteur, mais alors, vous qui parlez du nez, vous devez écrire de
      même.",
  ),
  'EFFORT' => array(
    'part' => 'n.',
    'definition' => "Voyant un homme qui avait le nez très-gros, Odry disait :
       -- En faisant cet homme-là la nature a fait un nez fort.",
  ),
```

```
...

  'ÉTÉ' => array(
    'part' => 'n.',
    'definition' => "On boit tant de thé en hiver dans les soirées de Londres,
      qu'on a dit que les Anglais faisaient de l'hiver la saison des thés.",
  ),
  'EXÉCUTIF' => array(
    'part' => 'n.',
    'definition' => "Quand l'Assemblée constituante eut restreint, comme on
      sait, l'autorité royale de Louis XVI, on fit cette épigramme :",
    'quote' => array(
      "Entre savants, quelquefois on dispute.",
      "D'où vient ce nom : pouvoir exécutif",
      "Que donne au roi le corps législatif ?",
      "Eh ! le voici : trop faible pour la lutte,",
      "C'est un pouvoir, hélas ! qui s'exécute.",
    ),
  ),
);
?>
```

Dans une version réelle de cet exemple, les données seraient enregistrées dans une base de données et chargées à la demande. Puisque, dans notre version, les données font partie du script, le code pour les obtenir est relativement simple. Nous examinons le terme transmis et construisons le fragment HTML à afficher :

```
<?php
$term = mb_strtoupper($_REQUEST['term'], 'utf-8');
if (isset($entries[$term])) {
  $entry = $entries[$term];
  $html = '<div class="entry">';
  $html .= '<h3 class="term">';
  $html .= $term;
  $html .= '</h3>';
  $html .= '<div class="part">';
  $html .= $entry['part'];
  $html .= '</div>';
  $html .= '<div class="definition">';
  $html .= $entry['definition'];
  if (isset($entry['quote'])) {
    $html .= '<div class="quote">';
    foreach ($entry['quote'] as $line) {
      $html .= '<div class="quote-line">'. $line .'</div>';
    }
    if (isset($entry['author'])) {
      $html .= '<div class="quote-author">'. $entry['author'].'</div>';
    }
    $html .= '</div>';
  }
  $html .= '</div>';
  $html .= '</div>';
  print($html);
}
?>
```

Les requêtes sur ce script, nommé `e.php`, retournent le fragment HTML correspondant au terme indiqué dans les paramètres de GET. Par exemple, en accédant au script avec `e.php?term=exécutif`, nous obtenons le contenu HTML illustré à la Figure 6.8[2].

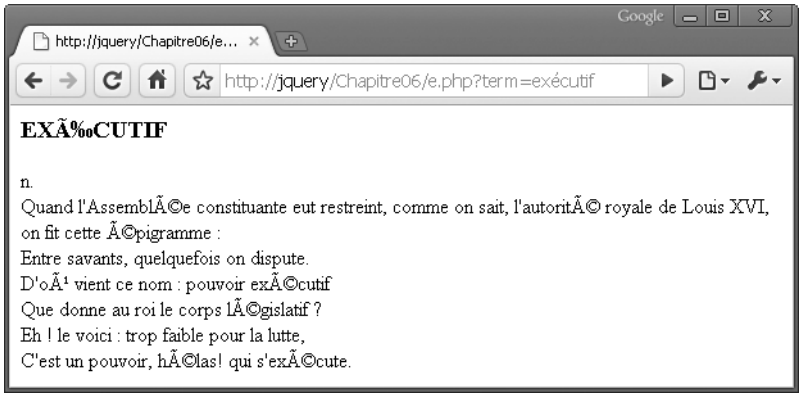


Figure 6.8
Entrée du dictionnaire fournie par un script PHP.

Une fois encore, nous remarquons la présentation quelque peu basique de ce fragment HTML. En effet, il n'est pas inséré dans un véritable document HTML et les règles CSS ne sont pas appliquées.

Puisque nous sommes en train d'expliquer comment transmettre des données au serveur, nous allons utiliser une autre méthode pour demander des entrées du dictionnaire. À la place d'un lien sur une lettre de l'alphabet, nous proposerons une liste de liens pour chaque terme et ferons en sorte qu'un clic sur l'un des liens charge la définition correspondante. Pour cela, nous ajoutons le balisage HTML suivant :

```
<div class="letter" id="letter-e">
  <h3>E</h3>
  <ul>
    <li><a href="e.php?term=écrire">Écrire</a></li>
    <li><a href="e.php?term=effort">Effort</a></li>
    <li><a href="e.php?term=emploi">Emploi</a></li>
    <li><a href="e.php?term=encorné">Encorné</a></li>
    <li><a href="e.php?term=encre">Encre</a></li>
    <li><a href="e.php?term=enseigner">Enseigner</a></li>
    <li><a href="e.php?term=envieux">Envieux</a></li>
    <li><a href="e.php?term=épiciers">Épiciers</a></li>
    <li><a href="e.php?term=esprit">Esprit</a></li>
    <li><a href="e.php?term=été">Été</a></li>
    <li><a href="e.php?term=exécutif">Exécutif</a></li>
  </ul>
</div>
```

2. N.d.T : les caractères accentués ne sont pas affichés correctement car le fragment HTML reçu n'est pas inclus dans un document HTML qui fixe l'encodage adéquat.

Nous devons modifier notre code JavaScript pour qu'il invoque le script PHP avec les paramètres adéquats. Pour cela, nous pouvons employer le mécanisme `.load()` normal, en concaténant la chaîne de requête à l'URL et en demandant directement les données avec des adresses de la forme `e.php?term=exécutif`. Toutefois, nous pouvons faire en sorte que jQuery construise la chaîne de requête à partir d'une mappe passée à la fonction `$.get()` :

```
$(document).ready(function() {
  $('#letter-e a').click(function() {
    $.get('e.php', {'term': $(this).text()}, function(data) {
      $('#dictionary').html(data);
    });
    return false;
  });
});
```

Puisque nous connaissons à présent les interfaces AJAX de jQuery, le fonctionnement de ce code nous est familier. La seule différence réside dans le second paramètre, qui nous permet de fournir une mappe de clés et de valeurs qui seront intégrées à la chaîne de requête. Dans notre cas, la clé est toujours `term`, mais la valeur est extraite du contenu de chaque lien. En cliquant sur le dernier lien de la liste, la définition correspondante s'affiche (voir Figure 6.9).

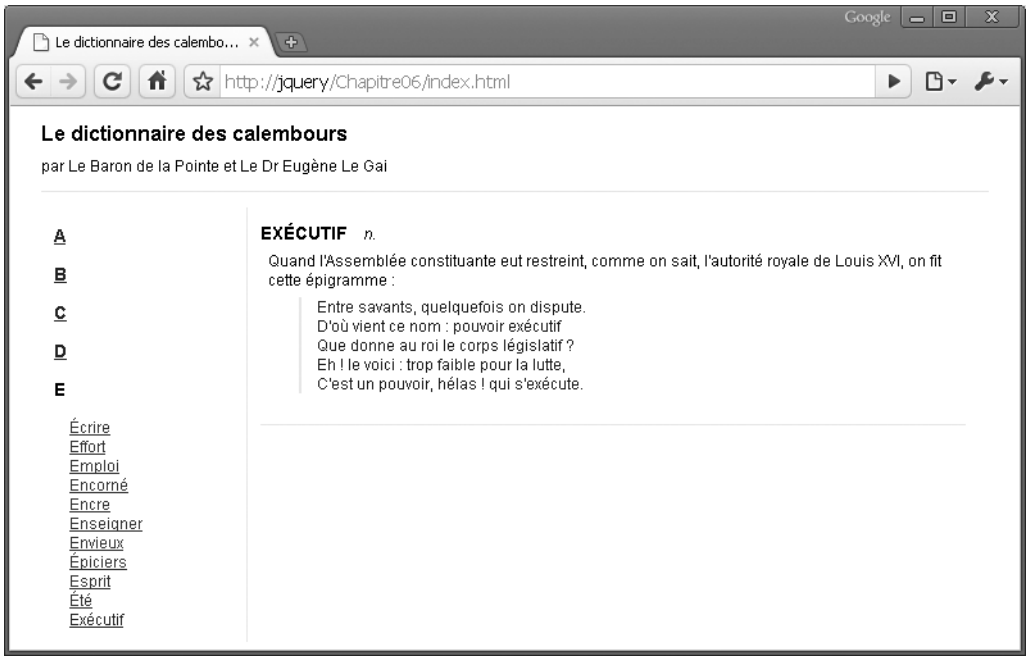


Figure 6.9

La définition du terme transmis est affichée.

Les adresses de tous les liens sont indiquées dans le source HTML, même si nous ne les utilisons pas dans le code. Ainsi, lorsque JavaScript est désactivé ou indisponible, l'utilisateur pourrait encore disposer d'une méthode de navigation (toujours notre principe d'*amélioration progressive*) à condition d'adapter le code PHP pour qu'il renvoie dans ce cas une page HTML complète. Pour éviter qu'un clic sur les liens ne déclenche l'action par défaut, le gestionnaire d'événements doit retourner `false`.

Effectuer une requête POST

Les requêtes HTTP de type POST sont presque identiques à celles de type GET. La différence notable réside dans le fait que la méthode GET place ses arguments dans la chaîne de requête, contrairement à la méthode POST. Toutefois, dans les appels AJAX, cette distinction est invisible à l'utilisateur normal. En général, la seule raison de choisir une méthode par rapport à l'autre est une volonté de se conformer aux standards du code côté serveur ou pour transmettre de grandes quantités de données ; la méthode GET impose des limites plus strictes. Notre script PHP est conçu pour prendre en charge les deux méthodes. Par conséquent, pour passer de GET à POST, nous devons simplement modifier la fonction jQuery invoquée :

```
$(document).ready(function() {
  $('#letter-e a').click(function() {
    $.post('e.php', {'term': $(this).text()}, function(data) {
      $('#dictionary').html(data);
    });
    return false;
  });
});
```

Les arguments sont identiques, mais la requête est à présent de type POST. Nous pouvons simplifier le code en invoquant la méthode `.load()`, qui utilise POST par défaut lorsqu'une mappe lui est passée en argument :

```
$(document).ready(function() {
  $('#letter-e a').click(function() {
    $('#dictionary').load('e.php', {'term': $(this).text()});
    return false;
  });
});
```

Cette version plus concise fonctionne de manière identique (voir Figure 6.10).

Sérialiser un formulaire

Pour envoyer des données au serveur, l'utilisateur doit souvent remplir un *formulaire*. Au lieu d'employer le mécanisme normal de soumission d'un formulaire, qui charge la réponse dans la fenêtre du navigateur, nous pouvons utiliser la boîte à outils AJAX de jQuery pour soumettre le formulaire de manière asynchrone et placer la réponse à l'intérieur de la page en cours.

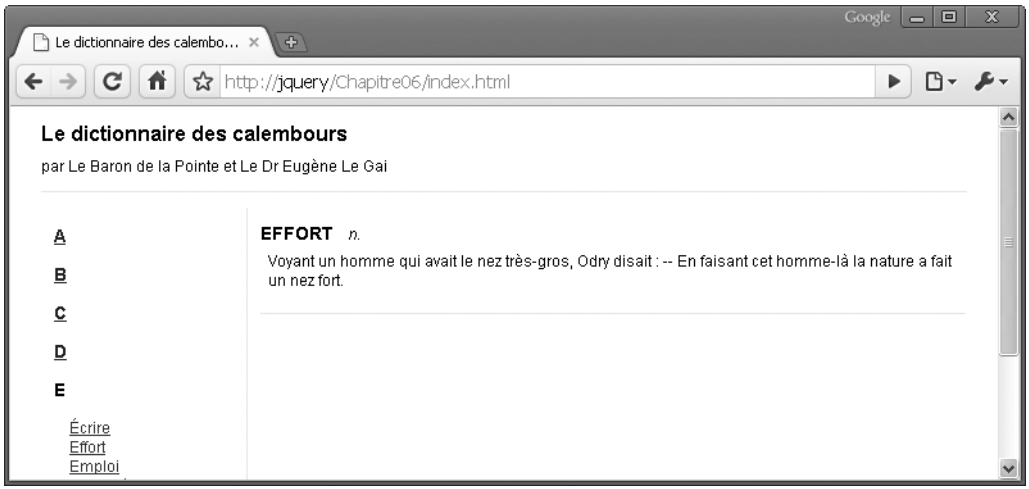


Figure 6.10

Obtention d'une définition par la méthode POST.

Pour illustrer cela, nous construisons un formulaire simple :

```
<div class="letter" id="letter-f">
  <h3>F</h3>
  <form>
    <input type="text" name="term" value="" id="term" />
    <input type="submit" name="search" value="rechercher" id="search" />
  </form>
</div>
```

Cette fois-ci, le script PHP retournera l'ensemble des entrées du dictionnaire qui contiennent le terme recherché. La structure de données conserve le format précédent, mais la logique diffère :

```
foreach ($entries as $term => $entry) {
  if (strpos($term, mb_strtoupper($_REQUEST['term'])) !== FALSE) {
    $html = '<div class="entry">';
    $html .= '<h3 class="term">';
    $html .= $term;
    $html .= '</h3>';
    $html .= '<div class="part">';
    $html .= $entry['part'];
    $html .= '</div>';
    $html .= '<div class="definition">';
    $html .= $entry['definition'];
    if (isset($entry['quote'])) {
      foreach ($entry['quote'] as $line) {
        $html .= '<div class="quote-line">'. $line .'</div>';
      }
      if (isset($entry['author'])) {
        $html .= '<div class="quote-author">'.$entry['author'] .'</div>';
```

```
        }
      }
      $html .= '</div>';
      $html .= '</div>';
      print($html);
    }
}
```

La fonction `strpos()` recherche le terme indiqué dans la chaîne passée en argument. Nous pouvons réagir à la soumission du formulaire et construire les paramètres de requête appropriés en parcourant l'arborescence du DOM :

```
$(document).ready(function() {
  $('#letter-f form').submit(function() {
    $('#dictionary').load('f.php',
      {'term': $('input[name="term"]').val()});
    return false;
  });
});
```

Ce code produit l'effet attendu, mais la recherche des champs de saisie en fonction de leur nom et leur ajout un par un à une mappe se révèlent fastidieux. Par ailleurs, cette solution n'est pas très efficace lorsque le formulaire devient plus complexe. Heureusement, jQuery propose un raccourci pour cet idiome fréquent. La méthode `.serialize()` opère sur un objet jQuery et convertit les éléments du DOM sélectionnés en une chaîne de requête qui peut être passée dans une requête AJAX. Voici comment généraliser notre gestionnaire de soumission :

```
$(document).ready(function() {
  $('#letter-f form').submit(function() {
    $.get('f.php', $(this).serialize(), function(data) {
      $('#dictionary').html(data);
    });
    return false;
  });
});
```

Désormais, le même script permet de prendre en charge des formulaires comprenant un grand nombre de champs. La Figure 6.11 montre les entrées qui correspondent à notre recherche.

6.4 Surveiller la requête

Jusqu'à présent, nous nous sommes contentés d'effectuer un appel à une méthode AJAX et d'attendre patiemment la réponse. Cependant, il peut être utile d'en savoir un peu plus sur la progression de la requête HTTP. Lorsque ce besoin se fait sentir, nous pouvons utiliser différentes méthodes jQuery pour enregistrer des *fonctions de rappel* qui seront invoquées lors d'événements AJAX.

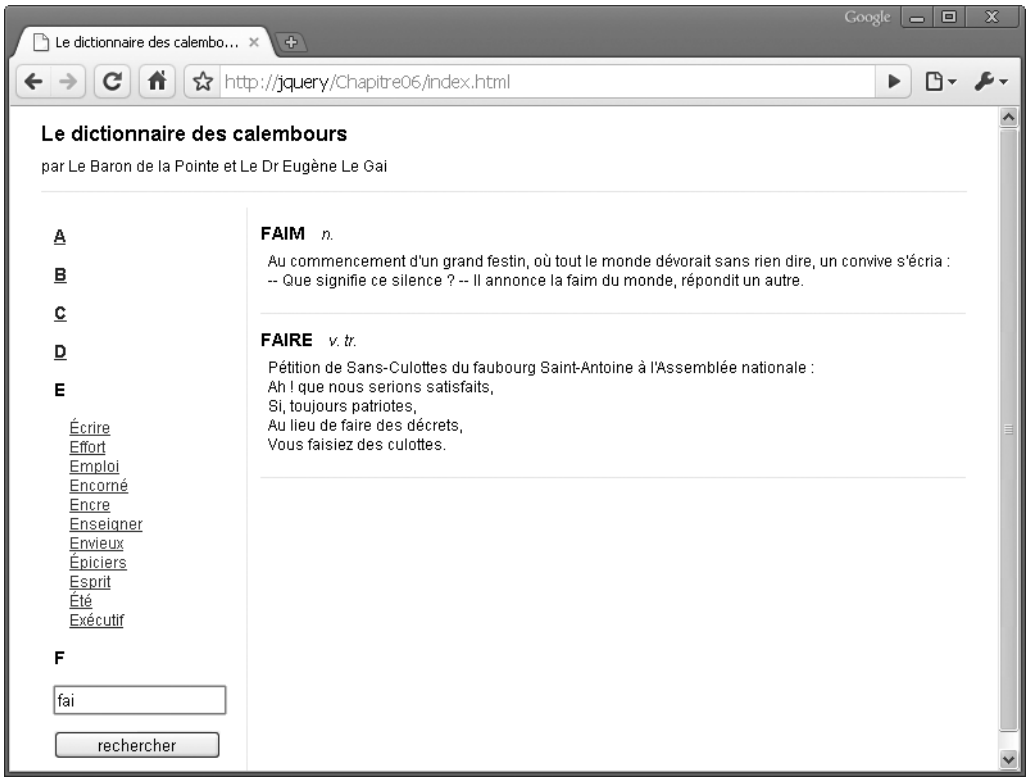


Figure 6.11

Affichage des entrées qui contiennent le terme recherché.

Les méthodes `.ajaxStart()` et `.ajaxStop()` sont deux exemples de ces fonctions d'observation et peuvent être associées à n'importe quel objet jQuery. Lorsqu'un appel AJAX débute, sans autre transfert en cours, la fonction de rappel de `.ajaxStart()` est invoquée. De même, lorsque la dernière requête active se termine, la fonction de rappel définie par `.ajaxStop()` est exécutée. Tous les observateurs sont *globaux*, car ils sont invoqués pour tous les échanges AJAX, quel que soit le code qui les a initiés.

Nous pouvons nous servir de ces méthodes pour fournir à l'utilisateur un retour en cas de connexion réseau lente.

Un message de chargement approprié est ajouté au contenu HTML de la page :

```
<div id="loading">
  Chargement en cours...
</div>
```

Ce message n'est qu'un fragment HTML quelconque. Il pourrait inclure une image GIF animée pour, par exemple, représenter une pulsation. Dans notre cas, nous définissons quelques règles CSS afin d'afficher le message comme illustré à la Figure 6.12.

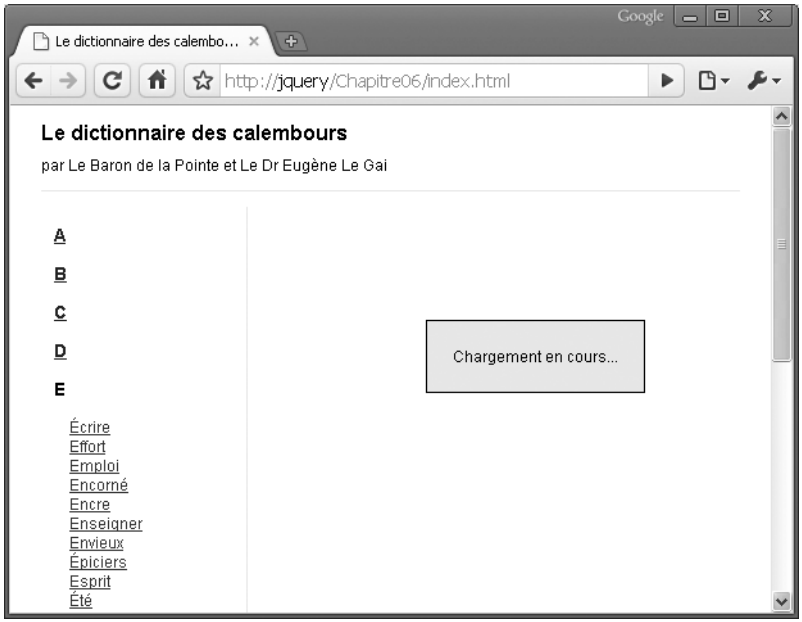


Figure 6.12
L'indicateur de chargement des définitions.

Toutefois, pour rester dans l'esprit de l'*amélioration progressive*, ce contenu HTML n'est pas placé directement dans la page. Il n'est pertinent que si JavaScript est disponible. Par conséquent, nous l'ajoutons en utilisant jQuery :

```
$(document).ready(function() {
  $('<div id="loading">Chargement en cours...</div>')
    .insertBefore('#dictionary')
});
```

Pour que le message soit initialement masqué, la feuille de style affecte la valeur `none` à la propriété `display` de ce `<div>`. Pour qu'il apparaisse au moment opportun, nous lui associons un observateur avec `.ajaxStart()` :

```
$(document).ready(function() {
  $('<div id="loading">Chargement en cours...</div>')
    .insertBefore('#dictionary')
    .ajaxStart(function() {
      $(this).show();
    });
});
```

La procédure de masquage est directement chaînée à ce code :

```
$(document).ready(function() {
  $('<div id="loading">Chargement en cours...</div>')
    .insertBefore('#dictionary')
    .ajaxStart(function() {
      $(this).show();
    }).ajaxStop(function() {
      $(this).hide();
    });
});
```

Et voilà, l'affichage du chargement en cours est opérationnel.

Notez que ces méthodes n'ont aucun rapport avec la manière dont les communications AJAX se passent. La méthode `.load()` associée au lien A et la méthode `.getJSON()` associée au lien B conduisent toutes deux au déclenchement de ces actions.

Dans notre cas, ce comportement global est adapté. En revanche, si nous devons être plus spécifiques, nous avons quelques solutions. Certaines méthodes d'observation, comme `.ajaxError()`, passent à leur fonction de rappel une référence à l'objet `XMLHttpRequest`. Elle peut servir à différencier les requêtes et à mettre en œuvre des comportements différents. Nous pouvons également effectuer un traitement plus spécifique en utilisant la fonction de bas niveau `$.ajax()`, sur laquelle nous reviendrons plus loin.

Le plus souvent, les interactions avec la requête se fondent sur la fonction de rappel invoquée en cas de réussite. Nous l'avons déjà employée plusieurs fois dans nos exemples pour interpréter les données transmises par le serveur et insérer les résultats dans la page. Mais elle peut également servir à d'autres formes de retour d'informations. Reprenons notre exemple `.load()` :

```
$(document).ready(function() {
  $('#letter-a a').click(function() {
    $('#dictionary').load('a.html');
    return false;
  });
});
```

Nous pouvons l'améliorer en faisant en sorte que le contenu s'affiche en fondu au lieu d'apparaître immédiatement. Pour cela, nous utilisons la fonction de rappel qui peut être passée à `.load()` et invoquée à la fin du changement :

```
$(document).ready(function() {
  $('#letter-a a').click(function() {
    $('#dictionary').hide().load('a.html', function() {
      $(this).fadeIn();
    });
    return false;
  });
});
```

Tout d'abord, nous masquons l'élément cible et démarrons le chargement. Une fois celui-ci terminé, la fonction de rappel est invoquée pour afficher avec un effet de fondu l'élément rempli.

6.5 AJAX et les événements

Supposons que nous souhaitions que chaque terme du dictionnaire puisse contrôler l'affichage de sa définition ; en cliquant sur le terme, la définition associée est affichée ou masquée. À l'aide des techniques décrites jusqu'à présent, la mise en œuvre est relativement immédiate :

```
$(document).ready(function() {
  $('.term').click(function() {
    $(this).siblings('.definition').slideToggle();
  });
});
```

Lors du clic sur un terme, ce code recherche les frères de l'élément dont la classe est `definition`. Ensuite, il les affiche ou les masque par un effet de glissement.

Tout semble parfait, mais, en réalité, un clic ne déclenche aucune action. En effet, les termes n'ont pas encore été ajoutés au document au moment de la liaison des gestionnaires de `click`. Même si nous parvenions à associer des gestionnaires de `click` à ces éléments, un clic sur une lettre différente ferait que les gestionnaires ne seraient plus liés.

Il s'agit d'un problème classique lorsque le contenu d'une partie de la page est obtenu par une requête AJAX. La solution, également classique, consiste à *relier* les gestionnaires à chaque actualisation de cette partie de la page. Toutefois, cette méthode est fastidieuse car le code de liaison d'un événement doit être invoqué dès que la structure DOM de la page évolue.

Au Chapitre 3, nous avons présenté une alternative qui se révèle souvent meilleure. Nous pouvons utiliser la *délégation d'événement*, en liant l'événement à un élément ancêtre qui ne change jamais. Dans ce cas, nous associons le gestionnaire de `click` à l'élément `document` avec la méthode `.live()` et interceptons les clics de cette manière :

```
$(document).ready(function() {
  $('.term').live('click', function() {
    $(this).siblings('.definition').slideToggle();
  });
});
```

La méthode `.live()` demande au navigateur d'observer tous les clics qui se produisent n'importe où sur la page. Si, et seulement si, l'élément cliqué correspond au sélecteur `.term`, alors le gestionnaire est exécuté. À présent, le comportement défini se produira sur n'importe quel terme, même s'il est ajouté ultérieurement *via* une requête AJAX.

6.6 Questions de sécurité

Pour toutes ses utilisations dans la construction d'applications web dynamiques, l'objet `XMLHttpRequest`, qui se trouve au cœur de l'utilisation d'AJAX dans jQuery, est soumis à des limites strictes. Pour empêcher les attaques de type XSS (*cross-site scripting*), il est normalement impossible de demander un document à un serveur autre que celui d'où provient la page d'origine.

Ce fonctionnement est généralement adapté. Par exemple, certaines personnes soulignent que l'analyse d'une structure JSON avec `eval()` est risquée. Si du code malveillant est inséré dans le fichier de données, il peut être exécuté par `eval()`. Cependant, puisque le fichier de données doit résider sur le même serveur que la page web, il est aussi facile d'injecter du code dans ce fichier que de l'injecter directement dans la page. Autrement dit, dans le cas d'un chargement de fichiers JSON approuvés, `eval()` ne présente pas de problèmes particuliers quant à la sécurité.

En revanche, il arrive souvent que des données doivent être obtenues à partir d'une source tierce. Pour contourner les barrières de sécurité et mettre en œuvre cette opération, il existe plusieurs solutions.

Une méthode consiste à demander au serveur de charger les données distantes, puis de les retransmettre au client. Cette approche est très puissante car le serveur peut effectuer un prétraitement des données si nécessaire. Par exemple, nous pouvons charger des fichiers XML contenant des flux RSS de nouvelles à partir de plusieurs sources, les agréger en un seul flux sur le serveur et publier ce nouveau fichier au client qui le demande.

Pour charger des données à partir d'un emplacement distant sans impliquer le serveur, il faut être plus astucieux. Une méthode répandue pour le chargement de fichiers JavaScript étrangers consiste à injecter des balises `<script>` à la demande. Puisque jQuery nous aide à insérer de nouveaux éléments du DOM, le code est simple :

```
$(document.createElement('script'))
  .attr('src', 'http://example.com/exemple.js')
  .appendTo('head');
```

En réalité, la méthode `$.getScript()` met automatiquement en place cette technique si elle détecte un hôte distant dans l'URL passée en argument. Ce cas est donc pris en charge pour nous.

Le navigateur exécutera le script chargé, mais aucun mécanisme ne permet de récupérer les résultats générés. C'est pourquoi cette technique exige une coopération avec l'hôte distant. Le script chargé doit effectuer une certaine action, comme fixer la valeur d'une variable globale qui a un effet sur l'environnement local. Les services qui publient des scripts exécutables de cette manière fournissent également une API pour interagir avec le script distant.

Une autre solution fonde le chargement des données distantes sur la balise HTML `<iframe>`. Cet élément permet d'utiliser n'importe quelle URL comme source des données, même si elle ne correspond pas au serveur qui héberge la page. Les données peuvent être facilement chargées et affichées sur la page en cours. En revanche, leur manipulation exige une coopération équivalente à celle de l'approche fondée sur la balise `<script>`. Les scripts placés dans l'élément `<iframe>` doivent fournir explicitement les données en utilisant des objets du document parent.

Utiliser JSONP pour les données distantes

La technique des balises `<script>` pour récupérer des fichiers JavaScript depuis une source distante peut être adaptée à l'obtention de fichiers JSON depuis un autre serveur. Pour cela, nous devons modifier légèrement le fichier JSON sur le serveur. Il existe plusieurs manières de procéder, dont l'une est directement prise en charge par jQuery : JSONP (*JSON with Padding*).

Un fichier au format JSONP est constitué d'un fichier JSON standard placé entre des parenthèses et commence par une chaîne de caractères. Cette chaîne de remplissage (*padding*) est déterminée par le client qui demande les données. En raison des parenthèses et selon la chaîne fournie, le client peut déclencher l'invocation d'une fonction ou l'affectation d'une variable.

L'implémentation PHP de la technique JSONP est plutôt simple :

```
<?php
  print($_GET['callback'] .'('. $data .')');
?>
```

Dans ce cas, la variable `$data` contient une chaîne de caractères qui représente un fichier JSON. Lorsque le script est invoqué, le paramètre `callback` de la chaîne de requête est ajouté au fichier résultant qui est envoyé au client.

Pour illustrer cette technique, nous devons simplement modifier légèrement notre exemple JSON précédent afin qu'il invoque cette source de données distante. Pour cela, la fonction `$.getJSON()` utilise un caractère substituable particulier, `?` :

```
$(document).ready(function() {
  var url = 'http://examples.learningjquery.com/jsonp/g.php';
  $('#letter-g a').click(function() {
    $.getJSON(url + '?callback=?', function(data) {
      $('#dictionary').empty();
      $.each(data, function(entryIndex, entry) {
        var html = '<div class="entry">';
        html += '<h3 class="term">' + entry['term'] + '</h3>';
        html += '<div class="part">' + entry['part'] + '</div>';
        html += '<div class="definition">';
        html += entry['definition'];
        if (entry['quote']) {
```

```
            html += '<div class="quote">';
            $.each(entry['quote'], function(lineIndex, line) {
              html += '<div class="quote-line">' + line + '</div>';
            });
            if (entry['author']) {
              html += '<div class="quote-author">' + entry['author'] + '</div>';
            }
            html += '</div>';
          }
          html += '</div>';
          html += '</div>';
          $('#dictionary').append(html);
        });
      });
      return false;
    });
  });
```

Normalement, nous ne sommes pas autorisés à récupérer les données JSON depuis un serveur distant (`examples.learningjquery.com` dans ce cas). Toutefois, puisque ce fichier est conçu pour fournir des données au format JSONP, nous pouvons obtenir les données en ajoutant une chaîne de requêtes à l'URL, en utilisant `?` comme paramètre substituable pour la valeur de l'argument `callback`. Au moment de la requête, jQuery remplace `?`, parse les résultats et les passe à la fonction de rappel sous forme de données comme s'il s'agissait d'une requête JSON locale.

Notez que les précautions concernant la sécurité restent valables. Tout ce que le serveur choisit de retourner au navigateur sera exécuté sur l'ordinateur de l'utilisateur. La technique JSONP ne doit être employée qu'avec des données issues d'une source de confiance.

6.7 Autres solutions

La boîte à outils AJAX fournie par jQuery est bien remplie. Nous avons examiné plusieurs possibilités, mais nous n'avons utilisé que les outils du dessus. Si les variantes sont trop nombreuses pour être toutes décrites ici, nous allons présenter les principales solutions de personnalisation des communications AJAX.

La méthode AJAX de bas niveau

Nous avons vu plusieurs méthodes qui déclenchent des transactions AJAX. En interne, jQuery associe chacune de ces méthodes à des variantes de la fonction globale `$.ajax()`. Au lieu de supposer une opération AJAX particulière, cette fonction prend en argument une mappe d'options qui personnalise son fonctionnement.

Notre premier exemple chargeait un fragment HTML en utilisant `$('#dictionary').load('a.html')`. Cette opération peut être accomplie à l'aide de `$.ajax()` :

```
$.ajax({
  url: 'a.html',
  type: 'GET',
  dataType: 'html',
  success: function(data) {
    $('#dictionary').html(data);
  }
});
```

Nous devons préciser explicitement la méthode de requête, le type des données retournées et le traitement des données obtenues. Il est évident que cette approche demande un travail plus important au programmeur, mais il bénéficie alors d'une plus grande flexibilité.

Voici quelques fonctionnalités spéciales permises par l'utilisation de la méthode de bas niveau `$.ajax()` :

- Empêcher le navigateur de mettre en cache les réponses du serveur. Cela sera utile si le serveur génère dynamiquement les données.
- Enregistrer différentes fonctions de rappel qui seront exécutées suite à la réussite de la requête, sa terminaison sur une erreur ou dans tous les cas.
- Supprimer les gestionnaires globaux, comme ceux définis avec `$.ajaxStart()`, qui sont déclenchés normalement par toutes les interactions AJAX.
- Fournir un nom d'utilisateur et un mot de passe pour une authentification auprès de l'hôte distant.

Pour plus de détails sur l'utilisation des options disponibles, consultez le guide de référence jQuery ou la référence de l'API à **http://docs.jquery.com/Ajax/jQuery.ajax**.

Modifier les valeurs par défaut

La fonction `$.ajaxSetup()` nous permet de préciser les valeurs par défaut de chaque option utilisée dans les invocations des méthodes AJAX. Elle prend en argument une mappe d'options identique à celle passée à la méthode `$.ajax()`. Ces valeurs sont ensuite utilisées par toutes les requêtes AJAX ultérieures, sauf indication contraire :

```
$.ajaxSetup({
  url: 'a.html',
  type: 'POST',
  dataType: 'html'
});
$.ajax({
  type: 'GET',
  success: function(data) {
    $('#dictionary').html(data);
  }
});
```

Le code se comporte de la même manière que dans notre exemple `$.ajax()` précédent. L'URL de la requête devient une valeur par défaut grâce à l'appel à `$.ajaxSetup()`. Il n'est donc plus utile de la préciser lors de l'invocation de la méthode `$.ajax()`. Si la valeur du paramètre `type` a été fixée par défaut à `POST`, rien ne nous empêche de lui donner la valeur `GET` lors de l'appel à `$.ajax()`.

Charger des parties d'une page HTML

La première technique AJAX présentée, la plus simple, consistait à obtenir un fragment HTML et à l'insérer dans la page. Cependant, il arrive parfois que le serveur dispose du contenu HTML dont nous avons besoin, mais qu'il se trouve déjà à l'intérieur d'une page HTML. Lorsqu'il n'est pas facile d'obtenir du serveur les données dans le format souhaité, jQuery peut apporter son aide côté client.

Prenons un cas semblable à notre premier exemple, mais dans lequel le document qui contient les définitions du dictionnaire est une page HTML complète :

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <title>Le dictionnaire des calembours : H</title>
    <link rel="stylesheet" href="dictionary.css"
      type="text/css" media="screen" />
  </head>

  <body>
    <div id="container">
      <div id="header">
        <h2>Le dictionnaire des calembours : H</h2>
        <div class="author">
          par Le Baron de la Pointe et Le Dr Eugène Le Gai
        </div>
      </div>

      <div id="dictionary">
        <div class="entry">
          <h3 class="term">HACHÉES</h3>
          <div class="part">adj.</div>
          <div class="definition">
            Quelles sont les lettres les plus maltraitées ? -- Les lettres H E.
          </div>
        </div>

        <div class="entry">
          <h3 class="term">HACHER</h3>
          <div class="part">v. tr.</div>
          <div class="definition">
            Madame de Sévigné disait des pendules à secondes, qu'elle ne les
            aimait pas, parce qu'elles hachent la vie trop menu.
          </div>
        </div>
      </div>
```

```
    </div>
  </body>
</html>
```

Nous pouvons charger l'intégralité du document dans notre page en utilisant le code développé précédemment :

```
$(document).ready(function() {
  $('#letter-h a').click(function() {
    $('#dictionary').load('h.html');
    return false;
  });
});
```

L'effet produit est assez étrange (voir Figure 6.13). En effet, certains éléments de la page HTML ne sont pas censés être affichés.

Figure 6.13

La page HTML est incluse dans la page englobante.

Pour retirer le balisage superflu, nous pouvons employer une autre caractéristique de la méthode `.load()`. Lorsque nous indiquons l'URL du document à charger, nous pouvons également préciser une expression de sélection jQuery. Dans ce cas, elle sert à localiser une partie du document chargé, qui sera la seule à être insérée dans la page. Nous employons cette technique pour extraire du document uniquement les entrées du dictionnaire et les placer dans la page :

```
$(document).ready(function() {
  $('#letter-h a').click(function() {
    $('#dictionary').load('h.html .entry');
    return false;
  });
});
```

Le contenu HTML non pertinent n'est alors plus inclus dans la page (voir Figure 6.14).

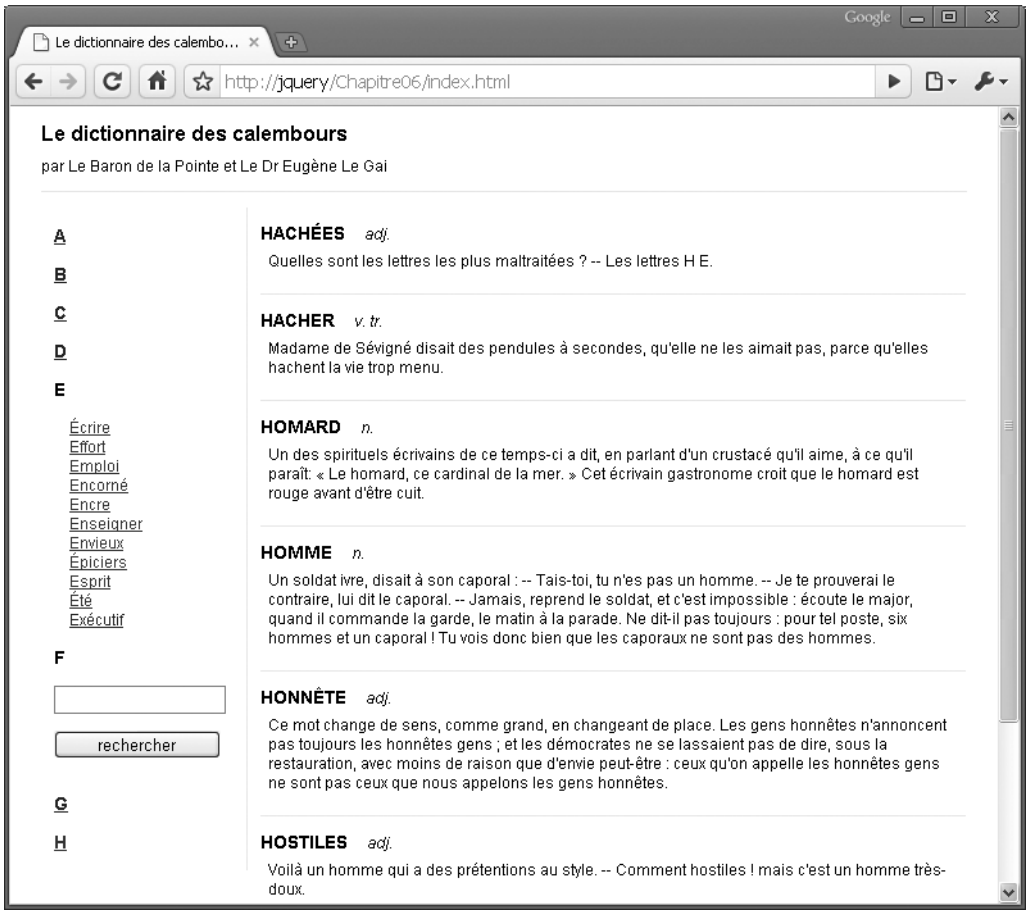


Figure 6.14

Le balisage HTML superflu n'est plus inséré dans la page.

6.8 En résumé

Nous avons appris que les méthodes AJAX fournies par jQuery peuvent nous aider à charger des données dans différents formats à partir du serveur sans actualiser la page. Nous pouvons exécuter des scripts à partir du serveur et lui renvoyer des données.

Nous avons également vu comment aborder les problèmes posés par les techniques de chargement asynchrones, comme garder des gestionnaires liés après le changement et charger des données à partir d'un serveur tiers.

Ce chapitre conclut la partie didacticiels de cet ouvrage. Nous maîtrisons à présent les principaux outils fournis par la bibliothèque jQuery : sélecteurs, événements, effets, manipulation du DOM et requêtes asynchrones sur le serveur. Toutefois, jQuery peut nous aider de bien d'autres manières et nous en découvrirons quelques-unes proposées par des plugins. Toutefois, commençons par combiner ces techniques afin d'améliorer nos pages web de manière novatrice et intéressante.

7

Manipulation des tables

Au sommaire de ce chapitre

- Tri et pagination
- Modifier l'aspect de la table
- En résumé

Les premiers chapitres ont présenté la bibliothèque jQuery au travers de didacticiels qui se focalisaient sur chaque composant de jQuery et les illustraient à partir d'exemples. Dans les Chapitres 7 à 9, nous procédons de manière inverse : nous prenons des exemples de problèmes réels et voyons comment les résoudre avec des méthodes jQuery.

Une librairie en ligne nous servira de modèle de site web, mais les techniques développées peuvent également s'appliquer à une grande diversité de sites, allant des blogs aux portfolios en passant par les vitrines commerciales et les intranets d'entreprise. Les Chapitres 7 et 8 se focalisent sur des constructions très répandues dans les sites web : les *tables* et les *formulaires*. Le Chapitre 9 présente deux façons d'améliorer visuellement des informations : les *carrousels* et les *prompteurs* animés.

Ces dernières années, les standards web ont vu leurs spécifications se préciser et ont donc commencé à être mieux appliqués. Les mises en page fondées sur les tables ont ainsi peu à peu été abandonnées au profit des conceptions à base de CSS. Si les tables ont souvent servi de solution de dépannage dans les années 1990 pour créer des présentations multicolonnes et d'autres agencements complexes compatibles avec la plupart des navigateurs, elles n'ont jamais été conçues dans ce but. En revanche, CSS est une technologie créée expressément pour ces questions de présentation.

Toutefois, nous n'allons pas débattre ici du rôle des tables. Dans ce chapitre, nous utiliserons jQuery pour appliquer des techniques permettant d'améliorer la lisibilité, l'utilisabilité et l'apparence visuelle de conteneurs dont le balisage sémantique indique clairement qu'ils contiennent des *données tabulaires*. Pour plus de détails concernant l'application d'un balisage HTML sémantique aux tables, vous pouvez consulter le

billet "Bring on the Tables" publié par Roger Johansson à l'adresse **http://www.456bereastreet.com/archive/200410/bring_on_the_tables/**.

Certaines des techniques que nous appliquons aux tables sont disponibles dans des plugins, comme le module Table Sorter de Christian Bach. Pour de plus amples informations, rendez-vous sur le catalogue des plugins jQuery à l'adresse **http://plugins.jquery.com/**.

Dans ce chapitre, nous aborderons le tri, la pagination, la mise en exergue des lignes, les info-bulles, la réduction et le développement des lignes, ainsi que le filtrage.

7.1 Tri et pagination

Les deux opérations les plus fréquentes effectuées sur les données tabulaires sont le *tri* et la *pagination*. Lorsque la table est grande, les informations doivent pouvoir être réorganisées. Malheureusement, ces opérations ne sont pas les plus simples à mettre en œuvre.

Nous allons commencer par examiner le tri d'une table, en réordonnant des données pour qu'elles soient plus utiles à la personne qui les consulte.

Tri côté serveur

Pour trier des données, une solution classique consiste à effectuer cette opération sur le serveur. Les données présentées dans des tables proviennent souvent d'une *base de données* et le code qui les extrait de la base peut donc les demander dans un certain ordre, par exemple en utilisant la clause `ORDER BY` du langage SQL. Lorsque l'on a accès au code côté serveur, il est très facile de choisir un ordre de tri initial raisonnable.

Toutefois, le tri est plus utile lorsque l'internaute peut préciser lui-même l'ordre. Une solution très utilisée se fonde sur des liens pour les en-têtes de la table (`<th>`) qui représentent des colonnes pouvant servir au tri. Ces liens ciblent la page courante, mais ajoutent une *chaîne de requête* indiquant la colonne qui sert de base au tri :

```
<table id="mes-donnees">
  <thead>
    <tr>
      <th class="nom">
        <a href="index.php?tri=nom">Nom</a>
      </th>
      <th class="date">
        <a href="index.php?tri=date">Date</a>
      </th>
    </tr>
  </thead>
<tbody>
...
</tbody>
</table>
```

Le serveur analyse le paramètre de la chaîne de requête et l'utilise pour retourner le contenu de la base de données dans l'ordre approprié.

Éviter les actualisations de la page

Cette solution est simple, mais la page doit être actualisée à chaque opération de tri. Nous l'avons vu, jQuery nous permet de supprimer ces actualisations en utilisant des méthodes AJAX. Si les en-têtes de colonnes sont des liens, nous pouvons ajouter du code jQuery pour remplacer ces liens par des requêtes AJAX :

```
$(document).ready(function() {
  $('#mes-donnees th a').click(function() {
    $('#mes-donnees tbody').load($(this).attr('href'));
    return false;
  });
});
```

Suite au clic sur un lien, jQuery envoie une requête AJAX au serveur qui cible la même page. Lors d'une requête AJAX, jQuery fixe l'en-tête HTTP `X-Requested-With` à la valeur `XMLHttpRequest` afin que le serveur puisse savoir que cette requête est de type AJAX. Le code du serveur peut alors être écrit de manière à renvoyer uniquement le contenu de l'élément `<tbody>`, sans le balisage HTML qui l'entoure. Nous pouvons ainsi récupérer la réponse et l'insérer à la place de l'élément `<tbody>` existant.

Il s'agit là d'un exemple d'amélioration progressive. Avec les liens pour le tri côté serveur, la page fonctionne parfaitement sans impliquer un quelconque code JavaScript. Toutefois, lorsque JavaScript est disponible, AJAX détourne la requête de la page et permet d'effectuer un tri sans avoir à recharger l'intégralité de la page.

Tri en JavaScript

Toutefois, il arrive que nous ne souhaitions pas attendre que le serveur retourne les données triées ou que nous ne disposions pas d'un langage de script côté serveur. Dans ce cas, une alternative possible consiste à réaliser le tri dans le navigateur en utilisant un script JavaScript côté client.

Par exemple, supposons qu'une table recense des livres, avec les noms des auteurs, les dates de publication et les prix (voir Figure 7.1) :

```
<table class="sortable">
  <thead>
    <tr>
      <th></th>
      <th>Titre</th>
      <th>Auteur(s)</th>
      <th>Date</th>
      <th>Prix</th>
    </tr>
  </thead>
```

```
    <tbody>
      <tr>
        <td><img src="../images/covers/small/1847192386.png" width="49"
          height="61" alt="Building Websites with Joomla! 1.5 Beta 1" />
        </td>
        <td>Building Websites with Joomla! 1.5 Beta 1</td>
        <td>Hagen Graf</td>
        <td>Févr 2007</td>
        <td>40,49 €</td>
      </tr>
      <tr>
        <td><img src="../images/covers/small/1904811620.png" width="49"
          height="61" alt="Learning Mambo: A Step-by-Step Tutorial to Building
          Your Website" />
        </td>
        <td>Learning Mambo: A Step-by-Step Tutorial to Building Your Website
        </td>
        <td>Douglas Paterson</td>
        <td>Déc 2006</td>
        <td>40,49 €</td>
      </tr>
    </tbody>
  </table>
```

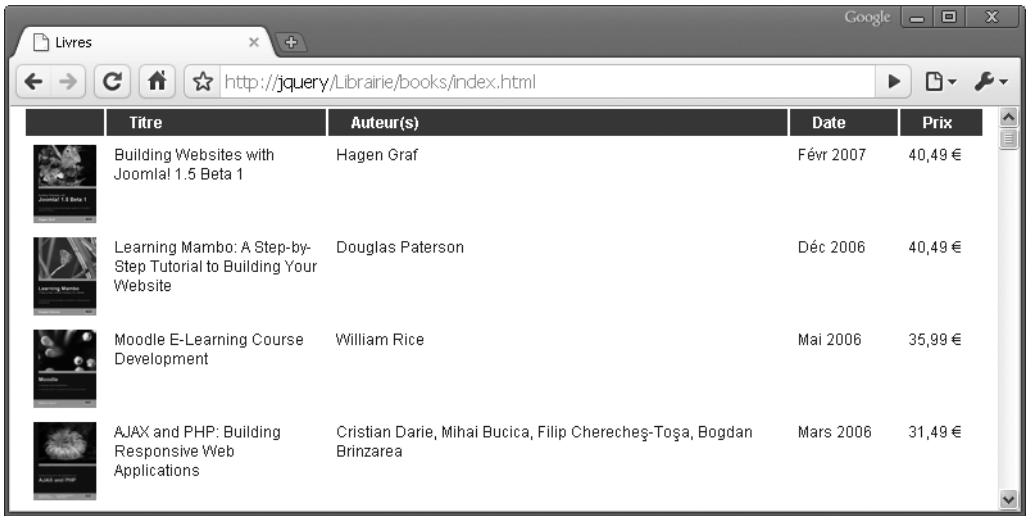


Figure 7.1

Une liste de livres affichée dans une table.

Nous voudrions transformer les en-têtes du tableau en boutons de manière à pouvoir trier les ouvrages en fonction des colonnes correspondantes. Voyons comment procéder.

Groupes de lignes

Vous aurez remarqué que nous utilisons les balises `<thead>` et `<tbody>` pour séparer les données en *groupes de lignes*. De nombreux développeurs HTML omettent ces balises, mais elles peuvent se révéler utiles pour constituer des sélecteurs CSS plus pratiques. Par exemple, supposons que nous souhaitions appliquer alternativement des bandes à la table, en évitant les lignes d'en-tête :

```
$(document).ready(function() {
  $('table.sortable tbody tr:odd').addClass('odd');
  $('table.sortable tbody tr:even').addClass('even');
});
```

Ce code permet d'alterner les couleurs appliquées aux lignes de la table, sans toucher à l'en-tête (voir Figure 7.2).

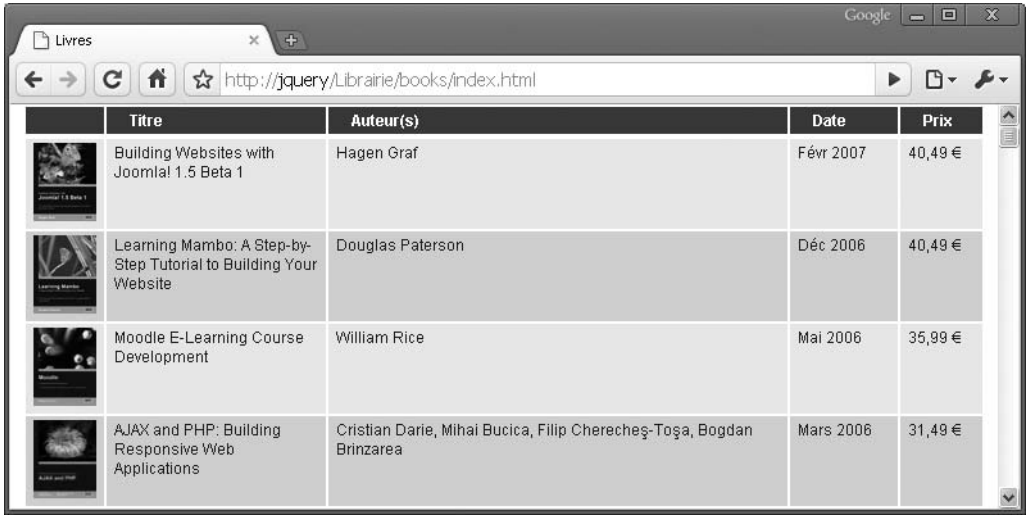


Figure 7.2

Application de bandes aux lignes de la table, sans toucher à l'en-tête.

Grâce à ces balises de regroupement, nous pourrons facilement sélectionner et manipuler les lignes de données sans affecter l'en-tête.

Tri alphabétique simple

Nous allons à présent effectuer un tri fondé sur la colonne TITRE de la table. Nous avons besoin d'une classe pour la cellule d'en-tête afin de pouvoir la sélectionner correctement :

```
<thead>
  <tr>
    <th></th>
    <th class="sort-alpha">Titre</th>
    <th>Auteur(s)</th>
    <th>Date</th>
    <th>Prix</th>
  </tr>
</thead>
```

Tri des tableaux en JavaScript

Pour le tri, nous pouvons employer la méthode JavaScript `.sort()`. Elle effectue un tri en place du tableau et accepte une *fonction de comparaison* en argument. Cette fonction compare deux éléments du tableau et doit retourner une valeur positive ou négative selon l'élément qui doit être placé en premier dans le tableau trié.

Prenons par exemple un simple tableau de nombres :

```
var arr = [52, 97, 3, 62, 10, 63, 64, 1, 9, 3, 4];
```

Nous pouvons le trier en invoquant `arr.sort()`. Les éléments se trouvent ensuite dans l'ordre suivant :

```
[1, 10, 3, 3, 4, 52, 62, 63, 64, 9, 97]
```

Vous le constatez, les éléments sont, par défaut, triés dans l'ordre *lexicographique* (par ordre alphabétique). Dans cet exemple, il serait plus censé de les trier *numériquement*. Pour cela, nous pouvons fournir une fonction de comparaison à la méthode `.sort()` :

```
arr.sort(function(a,b) {
  if (a < b)
    return -1;
  if (a > b)
    return 1;
  return 0;
});
```

Elle retourne une valeur négative si `a` doit être placé en premier dans le tableau trié, une valeur positive si `b` doit être placé en premier, et `0` si l'ordre des éléments n'a pas d'importance. La méthode `.sort()` peut ainsi classer les éléments dans l'ordre approprié :

```
[1, 3, 3, 4, 9, 10, 52, 62, 63, 64, 97]
```

Utiliser un comparateur pour trier les lignes de la table

Voici notre programme de tri initial :

```
$(document).ready(function() {
  $('table.sortable').each(function() {
    var $table = $(this);
    $('th', $table).each(function(column) {
      var $header = $(this);
```

```
        if ($header.is('.sort-alpha')) {
          $header.addClass('clickable').hover(function() {
            $header.addClass('hover');
          }, function() {
            $header.removeClass('hover');
          }).click(function() {
            var rows = $table.find('tbody > tr').get();
            rows.sort(function(a, b) {
              var keyA = $(a).children('td').eq(column).text()
                .toUpperCase();
              var keyB = $(b).children('td').eq(column).text()
                .toUpperCase();
              if (keyA < keyB) return -1;
              if (keyA > keyB) return 1;
              return 0;
            });
            $.each(rows, function(index, row) {
              $table.children('tbody').append(row);
            });
          });
        }
      });
    });
  });
```

Tout d'abord, notez que nous utilisons la méthode `.each()` pour que l'itération soit *explicite*. Nous pouvons lier un gestionnaire de `click` à toutes les cellules d'en-tête de classe `sort-alpha` en invoquant simplement `$('table.sortable th.sort-alpha').click()`, mais cette solution nous fait manquer un élément d'information essentiel : l'*indice de la colonne* qui correspond à la cellule d'en-tête sur laquelle l'utilisateur a cliqué. Puisque `.each()` passe l'indice de l'itération à sa fonction de rappel, nous pourrons nous en servir pour déterminer la cellule appropriée dans chaque ligne de données.

Après avoir identifié la cellule d'en-tête, nous récupérons un tableau de toutes les lignes de données. Cet exemple montre parfaitement comment `.get()` permet de transformer un objet jQuery en un tableau de nœuds du DOM ; même si, par de nombreux aspects, les objets jQuery ressemblent à des tableaux, ils ne disposent pas des méthodes des tableaux natifs, comme `.sort()`.

Puisque nous disposons à présent d'un tableau de nœuds du DOM, nous pouvons les trier, mais il nous faut pour cela écrire la fonction de comparaison adéquate. Nous souhaitons trier les lignes conformément au contenu textuel des cellules. La fonction de comparaison examine donc ces informations. Nous savons quelle cellule utiliser pour le tri car nous avons mémorisé l'indice de la colonne dans l'appel à `.each()` englobant. Nous convertissons le texte en majuscules car les comparaisons de chaînes en JavaScript sont *sensibles à la casse* et nous voulons que le tri soit *insensible à la casse*. Nous enregistrons les valeurs clés dans des variables pour éviter les répétitions de code, nous effectuons la comparaison et nous retournons une valeur positive ou négative conformément à la règle décrite précédemment.

Enfin, nous parcourons les lignes du tableau trié et les réinsérons dans la table. Puisque la méthode `.append()` ne clone pas les nœuds, ils sont déplacés au lieu d'être copiés. La table est à présent triée.

Voilà donc un exemple de *dégradation élégante*. Dans l'exemple, nous disposions d'une solution qui nécessitait l'actualisation de la page et nous l'avons améliorée en utilisant AJAX lorsque JavaScript est disponible. Dans le cas présent, la technique de tri ne peut pas fonctionner si JavaScript n'est pas activé ; nous supposons que le serveur ne propose aucun langage de script. La classe `clickable` est donc ajoutée par le code, ce qui permet d'obtenir une interface qui signale la possibilité de tri (par une image d'arrière-plan) uniquement lorsque le script peut s'exécuter. Dans le cas contraire, la page reste toujours opérationnelle, mais dans une version dégradée, sans la fonctionnalité de tri.

Puisque nous avons déplacé les lignes, les bandes de couleurs ne sont plus appropriées, comme l'illustre la Figure 7.3. Nous devons donc réappliquer les couleurs après le tri. Pour cela, nous plaçons le code de construction des bandes dans une fonction que nous invoquons lorsque c'est nécessaire :

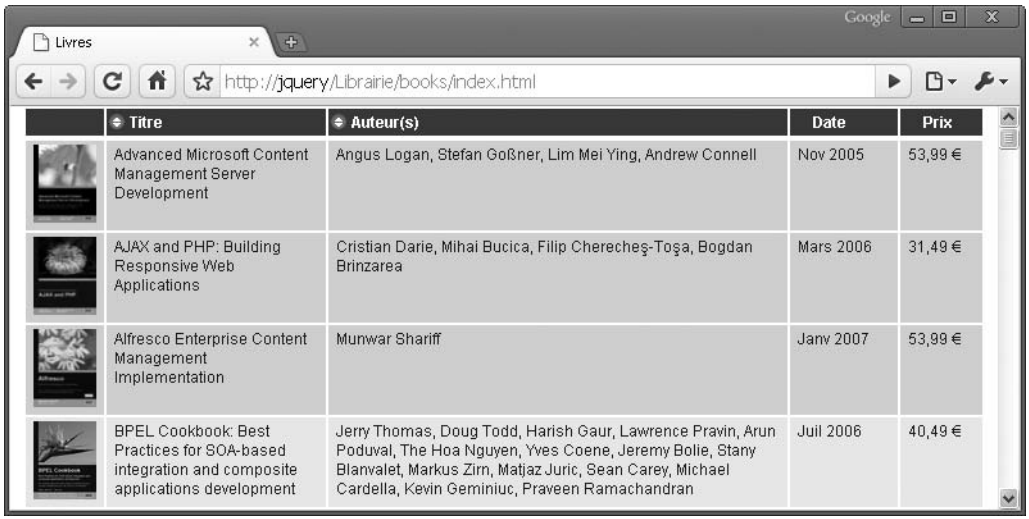


Figure 7.3

Les lignes déplacées conservent leur couleur d'arrière-plan.

```
$(document).ready(function() {
  var alternateRowColors = function($table) {
    $('tbody tr:odd', $table)
      .removeClass('even').addClass('odd');
    $('tbody tr:even', $table)
      .removeClass('odd').addClass('even');
  };
```

```
$('table.sortable').each(function() {
  var $table = $(this);
  alternateRowColors($table);
  $('th', $table).each(function(column) {
    var $header = $(this);
    if ($header.is('.sort-alpha')) {
      $header.addClass('clickable').hover(function() {
        $header.addClass('hover');
      }, function() {
        $header.removeClass('hover');
      }).click(function() {
        var rows = $table.find('tbody > tr').get();
        rows.sort(function(a, b) {
          var keyA = $(a).children('td').eq(column).text()
            .toUpperCase();
          var keyB = $(b).children('td').eq(column).text()
            .toUpperCase();
          if (keyA < keyB) return -1;
          if (keyA > keyB) return 1;
          return 0;
        });
        $.each(rows, function(index, row) {
          $table.children('tbody').append(row);
        });
        alternateRowColors($table);
      });
    }
  });
});
});
```

La Figure 7.4 montre que les bandes sont à nouveau correctement appliquées.

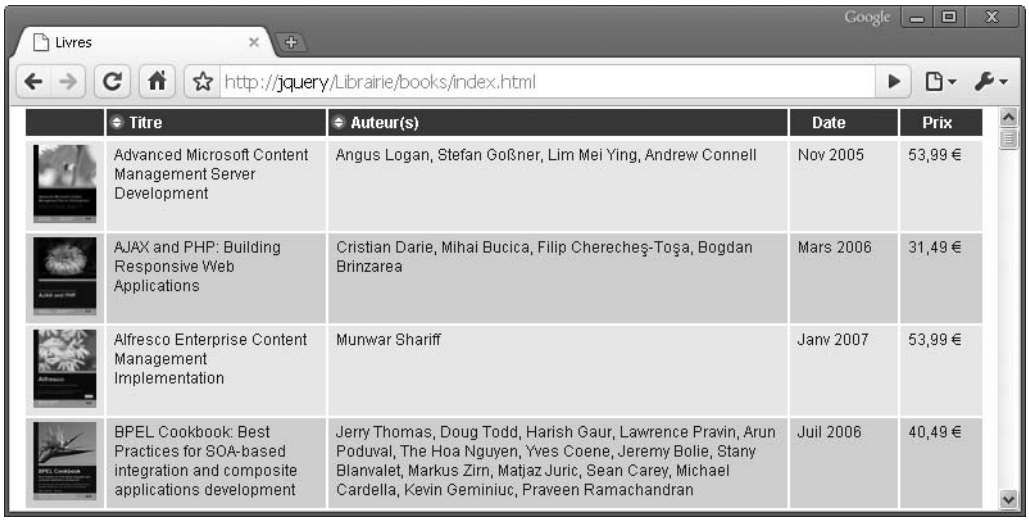


Figure 7.4

Les bandes sont reconstruites après le tri.

Puissance des plugins

La fonction `alternateRowColors()` que nous venons d'écrire constitue un bon candidat à une conversion en *plugin*. En réalité, toute opération que nous souhaitons appliquer à une collection d'éléments du DOM peut facilement devenir un plugin. Pour cela, nous devons modifier légèrement la fonction existante :

```
jQuery.fn.alternateRowColors = function() {
  $('tbody tr:odd', this)
    .removeClass('even').addClass('odd');
  $('tbody tr:even', this)
    .removeClass('odd').addClass('even');
  return this;
};
```

Nous avons réalisé trois changements importants :

- Elle est définie comme une nouvelle propriété de `jQuery.fn` à la place d'une fonction autonome. Cette opération enregistre la fonction en tant que méthode de plugin.
- Nous utilisons le mot clé `this` à la place du paramètre `$table`. Dans une méthode de plugin, `this` fait référence à l'objet jQuery sur lequel la méthode est invoquée.
- Enfin, la fonction retourne `this`. Lorsque la valeur de retour est l'objet jQuery, la nouvelle méthode peut être *chaînée*.

INFO

Pour de plus amples informations concernant le développement de plugins jQuery, consultez le Chapitre 11. Il décrit la création d'un plugin prêt à être proposé à la communauté, contrairement au petit exemple donné ici qui n'est destiné qu'à notre propre code.

Avec notre nouveau plugin, nous pouvons remplacer `alternateRowColors($table)` par `$table.alternateRowColors()`, qui est une instruction jQuery plus naturelle.

Performances

Notre code fonctionne, mais il est relativement lent. L'origine de ces performances médiocres se trouve dans la fonction de comparaison, qui effectue une quantité de travail relativement importante et est invoquée plusieurs fois au cours du tri. Autrement dit, le temps superflu passé dans le traitement est amplifié.

L'*algorithme de tri* utilisé par JavaScript n'est pas défini par la norme. Il peut s'agir d'un tri simple, comme le *tri par permutation* (de complexité en $O(n^2)$, au pire), ou d'une méthode sophistiquée, comme le *tri rapide* (en $O(n \log n)$ en moyenne). Toutefois, nous pouvons affirmer que la multiplication par deux du nombre d'éléments dans

un tableau fera plus que doubler la durée totale d'exécution de la fonction de comparaison.

Le remède à la lenteur de notre comparateur réside dans un *précalcul* des clés de la comparaison. Nous partons de notre fonction de tri actuelle :

```
rows.sort(function(a, b) {
  var keyA = $(a).children('td').eq(column).text()
    .toUpperCase();
  var keyB = $(b).children('td').eq(column).text()
    .toUpperCase();
  if (keyA < keyB) return -1;
  if (keyA > keyB) return 1;
  return 0;
});
```

Nous extrayons le calcul des clés dans une boucle séparée :

```
$.each(rows, function(index, row) {
  row.sortKey = $(row).children('td').eq(column)
    .text().toUpperCase();
});
rows.sort(function(a, b) {
  if (a.sortKey < b.sortKey) return -1;
  if (a.sortKey > b.sortKey) return 1;
  return 0;
});
$.each(rows, function(index, row) {
  $table.children('tbody').append(row);
  row.sortKey = null;
});
```

Dans la nouvelle boucle, nous réalisons tout le travail coûteux et enregistrons le résultat dans une nouvelle propriété `.sortKey`. Ce type de propriété, associée à un élément du DOM, sans être un attribut normal du DOM, s'appelle *expando*. L'emplacement d'enregistrement choisi est très pratique, car nous avons besoin d'une clé pour chaque ligne de la table. Nous pouvons ensuite examiner cet attribut dans la fonction de comparaison et obtenir un tri beaucoup plus rapide.

INFO

Nous fixons la propriété expando à `null` après en avoir terminé : nous faisons le ménage après notre passage. Si ce n'est pas strictement nécessaire dans ce cas, il faut prendre cette bonne habitude car les propriétés expando qui traînent peuvent provoquer des *fuites de mémoire*. Pour de plus amples informations, consultez l'Annexe C.

À la place des propriétés expando, nous pouvons opter pour le mécanisme de stockage des données proposé par jQuery. La méthode `.data()` enregistre ou récupère des informations quelconques associées à des éléments de la page, tandis que la méthode `.removeData()` supprime de telles informations stockées :

```
$.each(rows, function(index, row) {
  $(row).data('sortKey', $(row).children('td')
    .eq(column).text().toUpperCase());
});
rows.sort(function(a, b) {
  if ($(a).data('sortKey') < $(b).data('sortKey'))
    return -1;
  if ($(a).data('sortKey') > $(b).data('sortKey'))
    return 1;
  return 0;
});
$.each(rows, function(index, row) {
  $table.children('tbody').append(row);
  $(row).removeData('sortKey');
});
```

L'utilisation de `.data()` à la place des propriétés expando peut, parfois, se révéler plus pratique, car nous manipulons plus souvent des objets jQuery que directement des nœuds du DOM. Elle permet également d'éviter les problèmes potentiels liés aux fuites de mémoire dans Internet Explorer. Toutefois, pour la suite de ce chapitre, nous conservons les propriétés expando de manière à mettre en pratique le passage entre les opérations sur les nœuds du DOM et celles sur les objets jQuery.

Déterminer les clés de tri

Nous voulons à présent appliquer le même type de tri à la colonne AUTEUR(S) de la table. En ajoutant la classe `sort-alpha` à la cellule d'en-tête de cette colonne, nous pouvons utiliser notre code existant. Dans l'idéal, les auteurs doivent être triés en fonction de leur nom de famille, non de leur prénom. Puisque certains des ouvrages sont rédigés par plusieurs auteurs et que quelques auteurs ont des deuxièmes prénoms ou des initiales, nous avons besoin d'une aide extérieure pour déterminer la partie du texte qui servira de clé au tri. Nous pouvons fournir cette aide en plaçant la partie pertinente de la cellule dans une balise :

```
<tr>
  <td><img src="../images/covers/small/1847192386.png" width="49" height="61"
    alt="Building Websites with Joomla! 1.5 Beta 1" />
  </td>
  <td>Building Websites with Joomla! 1.5 Beta 1</td>
  <td>Hagen <span class="sort-key">Graf</span></td>
  <td>Févr 2007</td>
  <td>40,49 €</td>
</tr>
<tr>
  <td><img src="../images/covers/small/1904811620.png" width="49" height="61"
    alt="Learning Mambo: A Step-by-Step Tutorial to Building Your Website" />
  </td>
  <td>Learning Mambo: A Step-by-Step Tutorial to Building Your Website</td>
  <td>Douglas <span class="sort-key">Paterson</span></td>
  <td>Déc 2006</td>
  <td>40,49 €</td>
</tr>
```

Nous modifions ensuite notre code de tri pour prendre en compte cette balise, sans perturber le comportement existant pour la colonne TITRE qui est correct. En concaténant la clé de tri marquée avant la clé qui a déjà été calculée, nous pouvons effectuer le tri sur le nom de famille lorsqu'il est repéré, ou sur l'intégralité de la chaîne dans le cas contraire :

```
$.each(rows, function(index, row) {
  var $cell = $(row).children('td').eq(column);
  row.sortKey = $cell.find('.sort-key').text().toUpperCase()
    + ' ' + $cell.text().toUpperCase();
});
```

Le tri fondé sur la colonne AUTEUR(S) utilise à présent la clé fournie et se fait sur le nom de famille (voir Figure 7.5). Si les noms de famille sont identiques, l'intégralité de la chaîne permet de les départager.

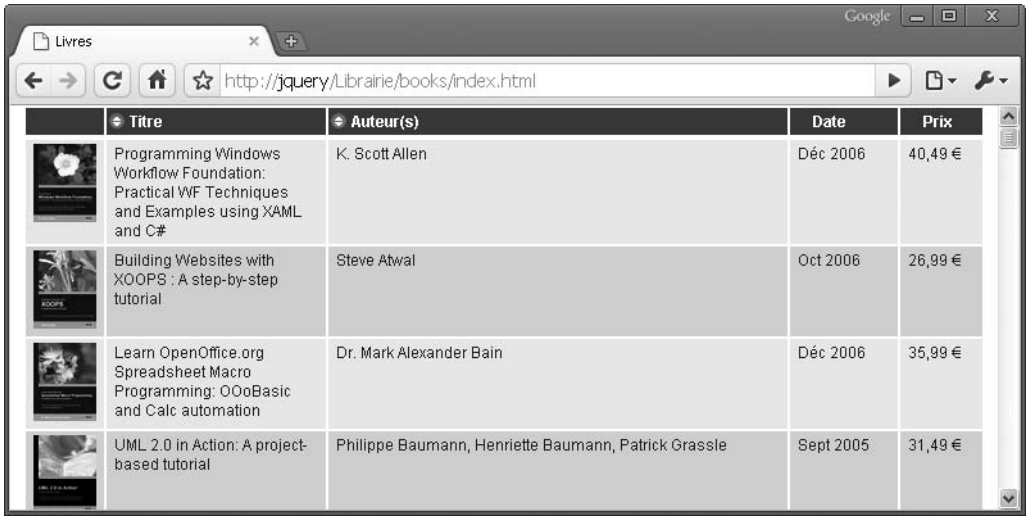


Figure 7.5

Tri de la table selon le nom de famille de l'auteur.

Trier d'autres types de données

L'utilisateur doit pouvoir trier la table non seulement selon les colonnes TITRE et AUTEUR(S), mais également selon les colonnes DATE et PRIX. Puisque nous avons généralisé notre fonction de comparaison, elle peut prendre en charge d'autres types de données, mais les clés calculées doivent tout d'abord être accordées à ces types. Par exemple, dans le cas du prix, nous devons convertir la chaîne en valeur numérique, mais il faut tout d'abord préparer la chaîne de caractères :

```
var key = parseFloat($cell.text().replace(/[\D,]/g, ''));
row.sortKey = isNaN(key) ? 0 : key;
```

L'expression régulière retire tous les caractères non numériques, ainsi que la virgule. Le résultat est passé à la fonction `parseFloat()`. La valeur retournée par `parseFloat()` doit ensuite être vérifiée, car, si aucun nombre n'a pu être obtenu à partir du texte, elle vaut `NaN` (*non un nombre*). Une telle valeur pourrait perturber le comportement de `.sort()`.

Pour les cellules comportant une date, la procédure est plus complexe. Elle se fonde sur l'objet JavaScript `Date`, mais sa méthode `parse()` ne comprend que les dates au format IETF. Nous devons donc commencer par convertir les noms de mois en anglais, ce que nous faisons avec une mappe :

```
var monthFRtoEN = {
  'janv': 'Jan', 'févr': 'Feb', 'mars': 'Mar', 'avr': 'Apr',
  'mai': 'May',  'juin': 'Jun', 'juil': 'Jul', 'août': 'Aug',
  'sept': 'Sep', 'oct': 'Oct',  'nov': 'Nov',  'déc': 'Dec'
};
var monthFR = $cell.text().replace(/[^a-zéû]/gi, '');
var dateEN = $cell.text().replace(monthFR, monthFRtoEN[monthFR.toLowerCase()]);
row.sortKey = Date.parse('1 ' + dateEN);
```

La première expression régulière permet d'extraire uniquement le nom du mois du texte de la cellule. La deuxième expression régulière remplace dans le texte de la cellule le nom du mois en français par le nom du mois anglais, qui est obtenu en consultant la mappe de conversion. La variable `dateEN` contient ainsi la date en anglais.

Dans la table, les dates précisent uniquement le mois et l'année. Puisque `Date.parse()` a besoin d'une date complète, nous ajoutons `1` au début de la chaîne. Nous complétons ainsi le mois et l'année par un jour, puis la combinaison est convertie en un nombre de millisecondes compatible avec notre fonction de comparaison.

Nous pouvons distribuer ces expressions dans des fonctions séparées et invoquer la fonction appropriée selon la classe de l'en-tête de la table :

```
jQuery.fn.alternateRowColors = function() {
  $('tbody tr:odd', this)
    .removeClass('even').addClass('odd');
  $('tbody tr:even', this)
    .removeClass('odd').addClass('even');
  return this;
};
$(document).ready(function() {
  $('table.sortable').each(function() {
    var $table = $(this);
    $table.alternateRowColors();
    $('th', $table).each(function(column) {
      var $header = $(this);
      var findSortKey;
      if ($header.is('.sort-alpha')) {
```

```
          findSortKey = function($cell) {
            return $cell.find('.sort-key')
              .text().toUpperCase() + ' ' + $cell.text().toUpperCase();
          };
        }
        else if ($header.is('.sort-numeric')) {
          findSortKey = function($cell) {
            var key = $cell.text().replace(/[\D,]/g, '');
            key = parseFloat(key);
            return isNaN(key) ? 0 : key;
          };
        }
        else if ($header.is('.sort-date')) {
          findSortKey = function($cell) {
            var monthFRtoEN = {
              'janv': 'Jan', 'févr': 'Feb', 'mars': 'Mar', 'avr': 'Apr',
              'mai': 'May',  'juin': 'Jun', 'juil': 'Jul', 'août': 'Aug',
              'sept': 'Sep', 'oct': 'Oct',  'nov': 'Nov',  'déc': 'Dec'
            };
            var monthFR = $cell.text().replace(/[^a-zéû]/gi, '');
            var dateEN = $cell.text().replace(monthFR,
              monthFRtoEN[monthFR.toLowerCase()]);
            return Date.parse('1 ' + dateEN);
          };
        }
        if (findSortKey) {
          $header.addClass('clickable').hover(function() {
            $header.addClass('hover');
          }, function() {
            $header.removeClass('hover');
          }).click(function() {
            var rows = $table.find('tbody > tr').get();
            $.each(rows, function(index, row) {
              var $cell = $(row).children('td').eq(column);
              row.sortKey = findSortKey($cell);
            });
            rows.sort(function(a, b) {
              if (a.sortKey < b.sortKey) return -1;
              if (a.sortKey > b.sortKey) return 1;
              return 0;
            });
            $.each(rows, function(index, row) {
              $table.children('tbody').append(row);
              row.sortKey = null;
            });
            $table.alternateRowColors();
          });
        }
      });
    });
  });
```

La variable `findSortKey` sert à la fois de fonction de calcul de la clé et d'indicateur signalant si l'en-tête de la colonne est marqué par une classe lui permettant d'être utilisée pour le tri. Nous pouvons à présent trier la table en fonction de la date (voir Figure 7.6) ou du prix (voir Figure 7.7).

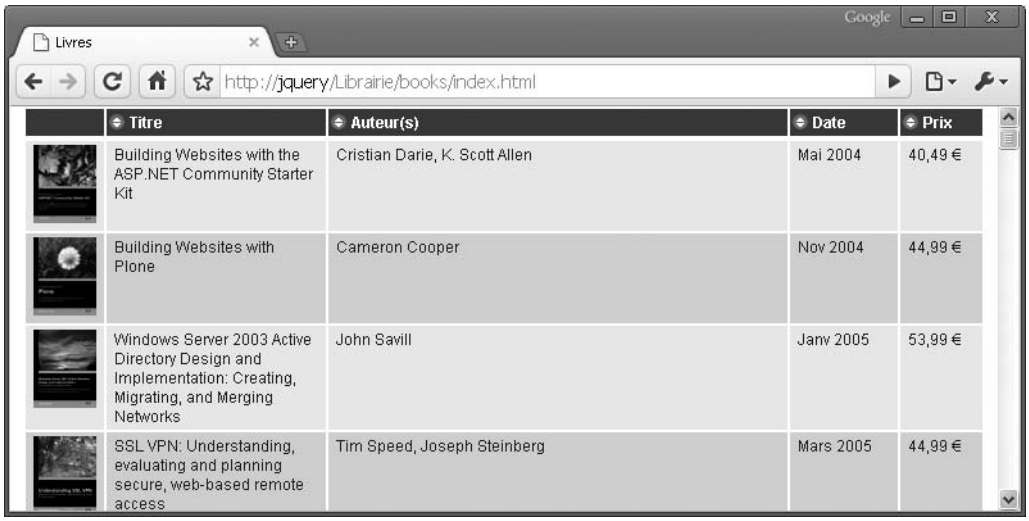


Figure 7.6

Tri de la table en fonction de la date de publication.

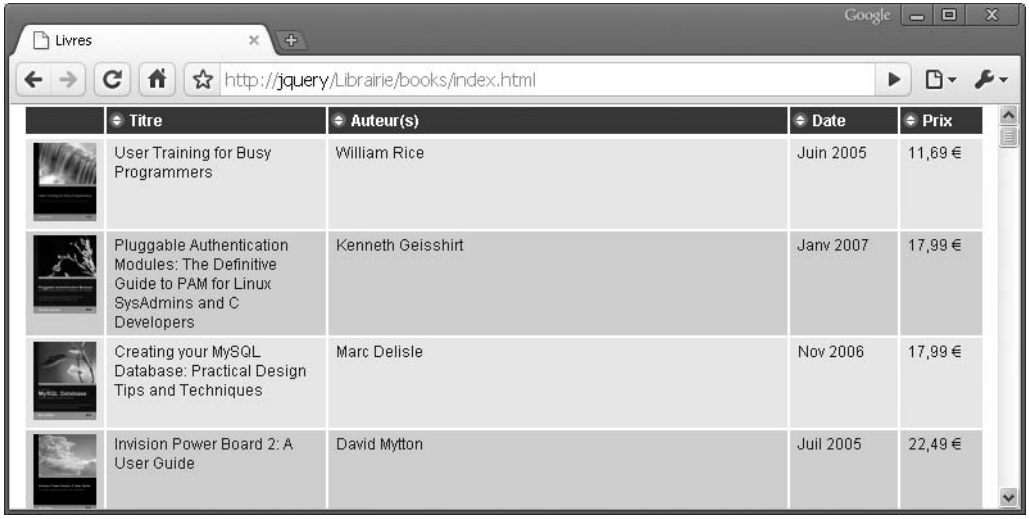


Figure 7.7

Tri de la table en fonction du prix du livre.

Mettre en exergue une colonne

Nous souhaitons améliorer l'interface utilisateur en lui rappelant visuellement l'action qu'il a effectuée. En mettant en exergue la colonne qui a été utilisée pour le tri, nous pouvons attirer son attention sur la partie de la table qui l'intéresse certainement. Puis-

que nous savons déjà comment sélectionner les cellules de la colonne de la table, l'application d'une classe à ces cellules ne pose aucune difficulté :

```
$table.find('td').removeClass('sorted')
  .filter(':nth-child(' + (column + 1) + ')')
  .addClass('sorted');
```

Ce bout de code commence par retirer la classe `sorted` à toutes les cellules, puis l'ajoute à celles qui se trouvent dans la colonne utilisée pour le tri. Nous devons ajouter `1` à l'indice de colonne obtenu précédemment, car le sélecteur `:nth-child()` compte à partir de un, non de zéro. Avec ce code en place, nous obtenons la mise en exergue de la colonne après une opération de tri (voir Figure 7.8).

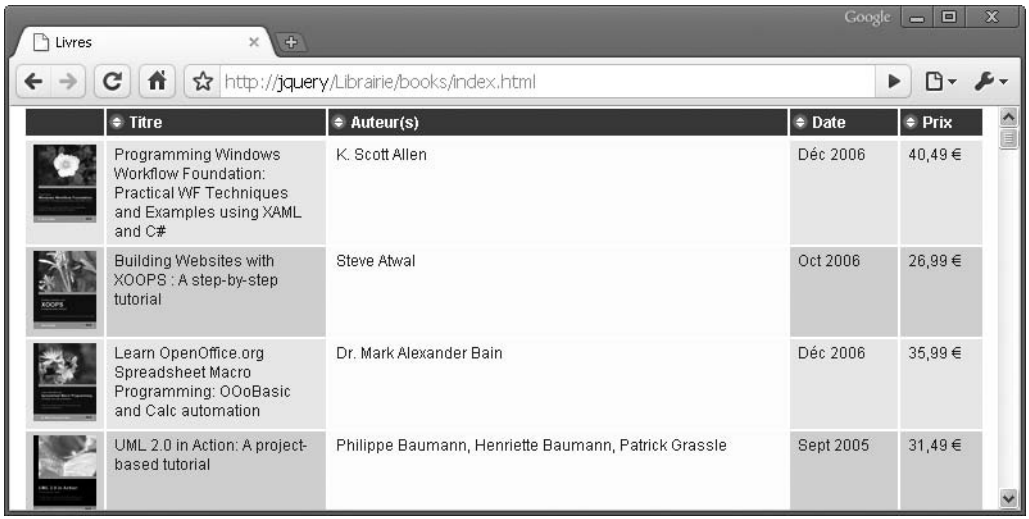


Figure 7.8

La colonne qui a servi au tri est surlignée.

Inverser le sens du tri

Comme dernière amélioration, nous allons permettre le choix entre un tri *croissant* et un tri *décroissant*. Lorsque l'utilisateur clique sur une colonne qui est déjà triée, nous voulons inverser le sens du tri. Pour cela, nous échangeons simplement les valeurs retournées par la fonction de comparaison en utilisant une variable :

```
if (a.sortKey < b.sortKey) return -sortDirection;
if (a.sortKey > b.sortKey) return sortDirection;
```

Lorsque `sortDirection` est égale à `1`, le tri se fait comme précédemment. Lorsqu'elle est égale à `-1`, le tri est inversé. Nous pouvons utiliser des classes pour suivre la direction du tri d'une colonne :

```
jQuery.fn.alternateRowColors = function() {
  $('tbody tr:odd', this)
    .removeClass('even').addClass('odd');
  $('tbody tr:even', this)
    .removeClass('odd').addClass('even');
  return this;
};
$(document).ready(function() {
  $('table.sortable').each(function() {
    var $table = $(this);
    $table.alternateRowColors();
    $('th', $table).each(function(column) {
      var $header = $(this);
      var findSortKey;
      if ($header.is('.sort-alpha')) {
        findSortKey = function($cell) {
          return $cell.find('.sort-key')
            .text().toUpperCase() + ' ' + $cell.text().toUpperCase();
        };
      }
      else if ($header.is('.sort-numeric')) {
        findSortKey = function($cell) {
          var key = $cell.text().replace(/[\D,]/g, '');
          key = parseFloat(key);
          return isNaN(key) ? 0 : key;
        };
      }
      else if ($header.is('.sort-date')) {
        findSortKey = function($cell) {
          var monthFRtoEN = {
            'janv': 'Jan', 'févr': 'Feb', 'mars': 'Mar', 'avr': 'Apr',
            'mai': 'May',  'juin': 'Jun', 'juil': 'Jul', 'août': 'Aug',
            'sept': 'Sep', 'oct': 'Oct',  'nov': 'Nov',  'déc': 'Dec'
          };
          var monthFR = $cell.text().replace(/[^a-zéû]/gi, '');
          var dateEN = $cell.text().replace(monthFR,
            monthFRtoEN[monthFR.toLowerCase()]);
          return Date.parse('1 ' + dateEN);
        };
      }
      if (findSortKey) {
        $header.addClass('clickable').hover(function() {
          $header.addClass('hover');
        }, function() {
          $header.removeClass('hover');
        }).click(function() {
          var sortDirection = 1;
          if ($header.is('.sorted-asc')) {
            sortDirection = -1;
          }
          var rows = $table.find('tbody > tr').get();
          $.each(rows, function(index, row) {
            var $cell = $(row).children('td').eq(column);
            row.sortKey = findSortKey($cell);
          });
          rows.sort(function(a, b) {
```

```
            if (a.sortKey < b.sortKey) return -sortDirection;
            if (a.sortKey > b.sortKey) return sortDirection;
            return 0;
          });
          $.each(rows, function(index, row) {
            $table.children('tbody').append(row);
            row.sortKey = null;
          });
          $table.find('th').removeClass('sorted-asc')
            .removeClass('sorted-desc');
          if (sortDirection == 1) {
            $header.addClass('sorted-asc');
          }
          else {
            $header.addClass('sorted-desc');
          }
          $table.find('td').removeClass('sorted')
            .filter(':nth-child(' + (column + 1) + ')')
            .addClass('sorted');
          $table.alternateRowColors();
        });
      }
    });
  });
});
```

Puisque nous utilisons des classes pour mémoriser le sens du tri, nous pouvons en profiter pour appliquer un style à l'en-tête de la colonne de manière à indiquer ce sens (voir Figure 7.9).

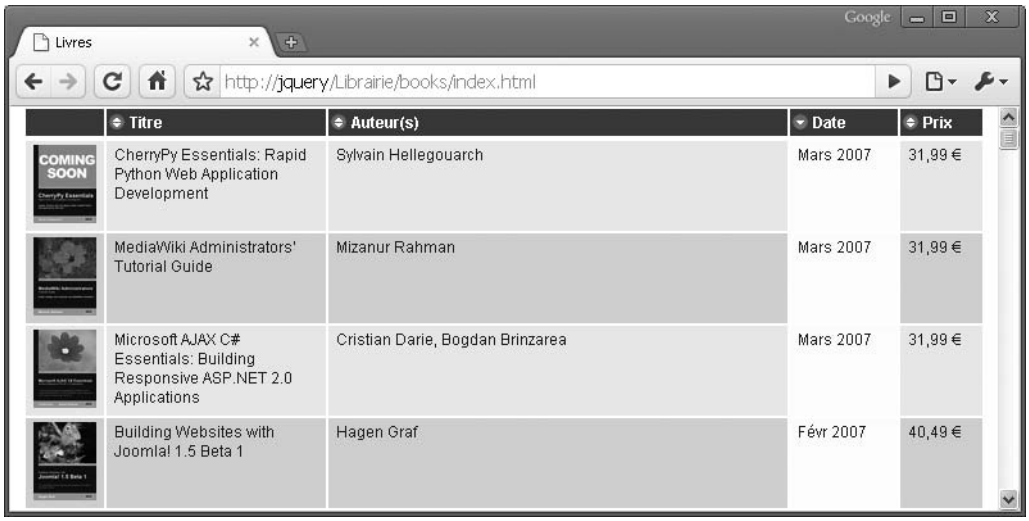


Figure 7.9

L'en-tête de la colonne indique le sens du tri, croissant ou décroissant.

Pagination côté serveur

Le tri est une bonne méthode pour se frayer un chemin parmi de grandes quantités de données et rechercher des informations. Mais nous pouvons également aider l'utilisateur à se focaliser sur une portion d'un grand ensemble de données en utilisant la *pagination*.

À l'instar du tri, la pagination s'effectue souvent sur le serveur. Si les données à afficher sont enregistrées dans une base de données, il est très facile d'extraire une page d'informations à la fois en utilisant la clause `LIMIT` de MySQL, `ROWNUM` d'Oracle ou toute méthode équivalente proposée par le moteur de base de données.

Notre premier exemple de tri se fondait sur des informations envoyées au serveur par l'intermédiaire de la chaîne de requête. C'est une technique que nous pouvons également employer pour demander au serveur d'effectuer la pagination, par exemple avec `index.php?page=52`. De la même manière que précédemment, cette opération peut se faire avec un chargement total de la page ou en utilisant AJAX pour récupérer uniquement une partie de la table. Cette stratégie ne dépend pas du navigateur et peut prendre en charge de grands ensembles de données.

Combiner le tri et la pagination

Les données qui sont suffisamment nombreuses pour bénéficier du tri ont également de grandes chances de pouvoir tirer profit de la pagination. Il n'est pas inhabituel de penser à combiner ces deux techniques pour afficher des données. Cependant, puisqu'elles affectent toutes deux la collection de données présente sur la page, il est important d'examiner leurs interactions lors de la mise en œuvre.

Le tri et la pagination peuvent se faire sur le serveur ou dans le navigateur web. Toutefois, les stratégies mises en place pour ces deux opérations doivent être synchronisées ou nous risquons d'obtenir un comportement déroutant. Supposons, par exemple, que nous ayons une table de huit lignes et deux colonnes, triée initialement sur la première colonne. Si le tri se fait ensuite sur la deuxième colonne, de nombreuses lignes peuvent changer de position (voir Figure 7.10).

Voyons à présent ce qui se passe si nous ajoutons la pagination. Supposons que le serveur ne fournisse que les quatre premières lignes et que le navigateur trie les données. Si la pagination est réalisée par le serveur et le tri, par le navigateur, la procédure de tri n'a pas accès à l'ensemble des données et les résultats sont incorrects (voir Figure 7.11).

Seules les données présentes sur la page peuvent être manipulées par le code JavaScript. Pour éviter que cela ne devienne un problème, les deux opérations doivent être réalisées sur le serveur (demander au serveur les données appropriées pour chaque page

Figure 7.10

Selon la colonne de tri, les lignes changent de position.

A	4
B	5
C	2
D	7
E	1
F	8
G	3
H	6

Avant

E	1
C	2
G	3
A	4
B	5
H	6
D	7
F	8

Après

Figure 7.11

La pagination sur le serveur suivie du tri dans le navigateur donne des résultats incorrects.

A	4
B	5
C	2
D	7

Avant

C	2
A	4
B	5
D	7

Après

Figure 7.12

Les deux opérations effectuées au même endroit donnent des résultats corrects.

A	4
B	5
C	2
D	7

Avant

E	1
C	2
G	3
A	4

Après

ou chaque opération de tri) ou dans le navigateur (toutes les données doivent être accessibles en permanence au code JavaScript) pour que les premiers résultats affichés correspondent aux premières lignes du jeu de données (voir Figure 7.12).

Pagination en JavaScript

Voyons donc comment ajouter la pagination JavaScript à la table qui peut déjà être triée dans le navigateur. Tout d'abord, concentrons-nous sur l'affichage d'une page de données, sans tenir compte, pour le moment, de l'action de l'utilisateur :

```
$(document).ready(function() {
  $('table.paginated').each(function() {
    var currentPage = 0;
    var numPerPage = 10;
    var $table = $(this);
    $table.find('tbody tr').hide()
      .slice(currentPage * numPerPage, (currentPage + 1) * numPerPage)
      .show();
  });
});
```

Ce code affiche la première page, avec dix lignes de données.

Une fois encore, nous nous appuyons sur la présence d'un élément `<tbody>` pour distinguer les données et les en-têtes ; nous ne voulons pas que les informations d'en-tête ou de pied disparaissent lors du passage à la page suivante. Pour sélectionner les lignes qui contiennent des données, nous masquons toutes les premières lignes, puis sélectionnons les lignes sur la page en cours et les affichons. La méthode `.slice()` assure la même fonction que la méthode éponyme de l'objet JavaScript `Array` : elle réduit la sélection aux éléments qui se trouvent entre les deux positions indiquées.

Le point le plus complexe de ce code réside dans les expressions données au filtre `.slice()`. Nous devons déterminer les indices des lignes qui correspondent au début et à la fin de la page en cours. Pour la première ligne, nous multiplions simplement le numéro de la page courante par le nombre de lignes sur chaque page. En multipliant le nombre de lignes par le numéro de la page courante plus un, nous obtenons la première ligne de la page suivante. Ces valeurs conviennent parfaitement car la méthode `.slice()` récupère les lignes jusqu'à celle indiquée par le second paramètre, sans l'inclure.

Afficher le sélecteur de page

Pour que l'utilisateur puisse interagir avec le système de pagination, nous devons placer un *sélecteur de page* à côté de la table. Il s'agit d'un ensemble de liens permettant de naviguer vers les différentes pages de données. Pour cela, nous pourrions simplement ajouter les liens correspondant aux pages dans le document HTML, mais cela irait à l'encontre du principe d'*amélioration progressive* que nous voulons respecter. À la place, nous devons ajouter les liens en utilisant du code JavaScript, ce qui évitera aux utilisateurs qui n'ont pas activé les fonctionnalités de script d'être induits en erreur par des liens non opérationnels.

Pour afficher les liens, nous calculons le nombre de pages et créons le nombre d'éléments du DOM équivalents :

```
var numRows = $table.find('tbody tr').length;
var numPages = Math.ceil(numRows / numPerPage);
var $pager = $('<div class="pager"></div>');
for (var page = 0; page < numPages; page++) {
  $('<span class="page-number">' + (page + 1) + '</span>')
    .appendTo($pager).addClass('clickable');
}
$pager.insertBefore($table);
```

Le nombre de pages est obtenu en divisant le nombre de lignes de données par le nombre d'éléments affichés sur chaque page. Puisque la division peut ne pas être entière, nous arrondissons le résultat avec `Math.ceil()` de manière que la dernière page

partielle soit accessible. Nous utilisons ensuite cette valeur pour créer des boutons pour chaque page et positionnons le nouveau sélecteur de page au-dessus de la table (voir Figure 7.13).

Figure 7.13

Le sélecteur de page est placé au-dessus de la table.

Activer les boutons du sélecteur de page

Pour que les nouveaux boutons soient opérationnels, nous devons mettre à jour la variable `currentPage` et lancer ensuite notre procédure de pagination. À première vue, nous pouvons fixer `currentPage` à la valeur de `page`, qui correspond à la valeur courante de l'itérateur qui crée les boutons :

```
$(document).ready(function() {
  $('table.paginated').each(function() {
    var currentPage = 0;
    var numPerPage = 10;
    var $table = $(this);
    var repaginate = function() {
      $table.find('tbody tr').hide()
        .slice(currentPage * numPerPage, (currentPage + 1) * numPerPage)
        .show();
    };
    var numRows = $table.find('tbody tr').length;
    var numPages = Math.ceil(numRows / numPerPage);
    var $pager = $('<div class="pager"></div>');
    for (var page = 0; page < numPages; page++) {
      $('<span class="page-number"></span>').text(page + 1)
        .click(function() {
          currentPage = page;
          repaginate();
        }).appendTo($pager).addClass('clickable');
    }
    $pager.insertBefore($table);
  });
});
```

Cela fonctionne, car la nouvelle fonction `repaginate()` est invoquée lors du chargement de la page et lors d'un clic sur un lien de page. Cependant, tous les liens affichent une table vide (voir Figure 7.14).

Figure 7.14

Les liens du sélecteur fonctionnent, mais affichent une table vide.

Le problème vient de la définition du gestionnaire de `click`, qui crée une fermeture. Ce gestionnaire fait référence à la variable `page` définie *en dehors* de la fonction. Lors de la modification de la variable au tour de boucle suivant, les gestionnaires de `click` que nous avons déjà configurés pour les boutons précédents sont également affectés. Par conséquent, si le sélecteur présente sept boutons, chacun cible la page 8, c'est-à-dire la valeur finale de `page` à la fin de la boucle.

INFO

Pour de plus amples informations concernant les fermetures, consultez l'Annexe C.

Pour résoudre ce problème, nous allons employer l'une des fonctionnalités les plus élaborées des méthodes jQuery de liaison des événements. Nous pouvons ajouter des *données d'événement personnalisées* au gestionnaire au moment de sa liaison et elles seront disponibles lors de son invocation :

```
$('<span class="page-number"></span>').text(page + 1)
  .bind('click', {newPage: page}, function(event) {
    currentPage = event.data['newPage'];
    repaginate();
  }).appendTo($pager).addClass('clickable');
```

Le nouveau numéro de page est passé au gestionnaire par l'intermédiaire de la propriété `data` de l'événement. Ainsi, ce numéro échappe aux hasards de la fermeture et conserve la valeur qu'il avait au moment de la liaison du gestionnaire. À présent, les liens du sélecteur conduisent aux différentes pages (voir Figure 7.15).

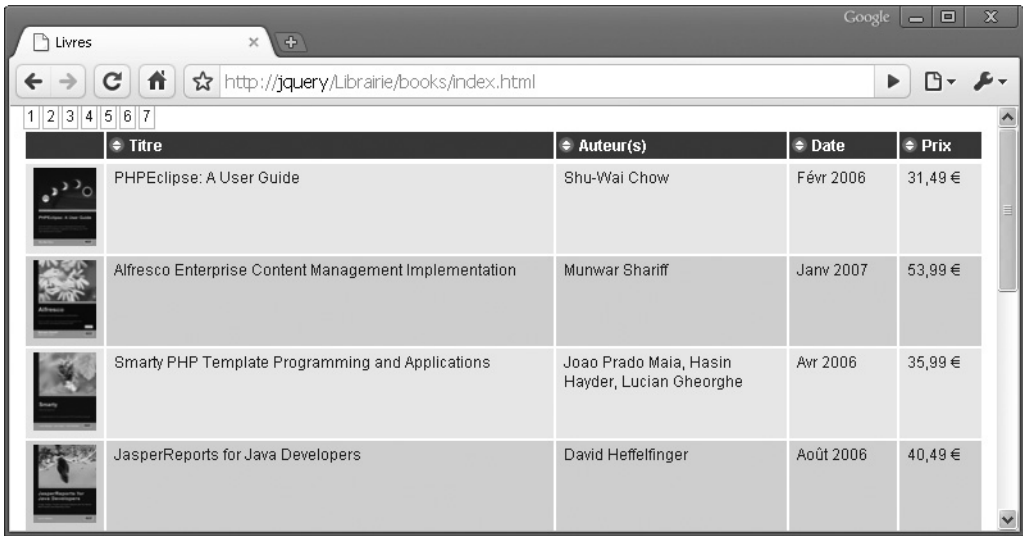


Figure 7.15

Le deuxième bouton mène à la deuxième page.

Signaler la page actuelle

Le sélecteur de page serait plus convivial si le numéro de la page en cours était mis en exergue. Pour cela, il suffit de modifier la classe des boutons lors de chaque clic :

```
$(document).ready(function() {
  $('table.paginated').each(function() {
    var currentPage = 0;
    var numPerPage = 10;
    var $table = $(this);
    var repaginate = function() {
      $table.find('tbody tr').hide()
        .slice(currentPage * numPerPage, (currentPage + 1) * numPerPage)
        .show();
    };
    var numRows = $table.find('tbody tr').length;
    var numPages = Math.ceil(numRows / numPerPage);
    var $pager = $('<div class="pager"></div>');
    for (var page = 0; page < numPages; page++) {
      $('<span class="page-number"></span>').text(page + 1)
        .bind('click', {newPage: page}, function(event) {
          currentPage = event.data['newPage'];
          repaginate();
          $(this).addClass('active').siblings().removeClass('active');
        }).appendTo($pager).addClass('clickable');
    }
    $pager.insertBefore($table)
      .find('span.page-number:first').addClass('active');
  });
});
```

Comme le montre la Figure 7.16, la page actuellement sélectionnée est signalée.

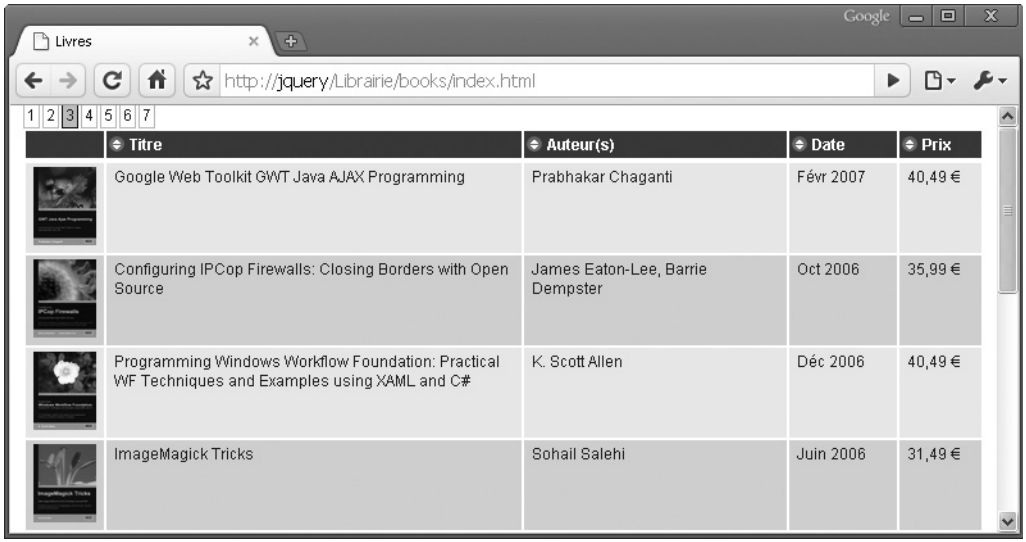


Figure 7.16

Le sélecteur indique la page en cours.

Pagination avec tri

Nous avons débuté cette section en indiquant que les procédures de tri et de pagination devaient tenir compte l'une de l'autre de manière à éviter un résultat déroutant. Puisque notre sélecteur de page est opérationnel, nous devons faire en sorte que les opérations de tri respectent la sélection courante de la page.

Pour cela, il suffit d'invoquer la fonction `repaginate()` dès qu'un tri a été réalisé. Toutefois, la *portée* de la fonction pose problème. Nous ne pouvons pas appeler `repaginate()` à partir de notre procédure de tri car elle se trouve dans un autre gestionnaire `$(document).ready()`. Nous pourrions simplement réunir les deux parties de code, mais nous allons être plus malins. Nous *découplons* les comportements de manière que la procédure de tri demande une pagination si ce comportement existe, et l'ignore sinon. Pour y parvenir, nous utiliserons un *gestionnaire d'événements personnalisé*.

Lors de notre étude précédente de la gestion des événements, nous nous sommes bornés aux événements qui sont déclenchés par le navigateur web, comme `click` et `mouseup`. Cependant, les méthodes `.bind()` et `.trigger()` peuvent également prendre en charge d'autres événements : nous pouvons préciser n'importe quelle chaîne de caractères comme nom d'événement. Grâce à cette possibilité, nous définissons un nouvel événement nommé `repaginate` pour remplacer la fonction que nous appelions :

```
$table.bind('repaginate', function() {
  $table.find('tbody tr').hide()
    .slice(currentPage * numPerPage, (currentPage + 1) * numPerPage).show();
});
```

L'invocation de `repaginate()` est ensuite remplacée par l'appel suivant :

```
$table.trigger('repaginate');
```

Nous pouvons également utiliser ce type d'appel pour le tri. Il n'aura aucun effet si la table ne propose pas de sélecteur de page, ce qui nous permet d'associer les deux possibilités comme bon nous semble.

Version finale du code

Voici la version finale du code qui met en œuvre le tri et la pagination :

```
jQuery.fn.alternateRowColors = function() {
  $('tbody tr:odd', this).removeClass('even').addClass('odd');
  $('tbody tr:even', this).removeClass('odd').addClass('even');
  return this;
};

$(document).ready(function() {
  $('table.sortable').each(function() {
    var $table = $(this);
    $table.alternateRowColors();
    $('th', $table).each(function(column) {
      var $header = $(this);
      var findSortKey;
      if ($header.is('.sort-alpha')) {
        findSortKey = function($cell) {
          return $cell.find('.sort-key').text().toUpperCase()
            + ' ' + $cell.text().toUpperCase();
        };
      }
      else if ($header.is('.sort-numeric')) {
        findSortKey = function($cell) {
          var key = $cell.text().replace(/[\D,]/g, '');
          key = parseFloat(key);
          return isNaN(key) ? 0 : key;
        };
      }
      else if ($header.is('.sort-date')) {
        findSortKey = function($cell) {
          var monthFRtoEN = {
            'janv': 'Jan', 'févr': 'Feb', 'mars': 'Mar', 'avr': 'Apr',
            'mai': 'May',  'juin': 'Jun', 'juil': 'Jul', 'août': 'Aug',
            'sept': 'Sep', 'oct': 'Oct',  'nov': 'Nov',  'déc': 'Dec'
          };
          var monthFR = $cell.text().replace(/[^a-zéû]/gi, '');
          var dateEN = $cell.text().replace(monthFR,
            monthFRtoEN[monthFR.toLowerCase()]);
          return Date.parse('1 ' + dateEN);
        };
      }
```

```
      if (findSortKey) {
        $header.addClass('clickable').hover(function() {
          $header.addClass('hover');
        }, function() {
          $header.removeClass('hover');
        }).click(function() {
          var sortDirection = 1;
          if ($header.is('.sorted-asc')) {
            sortDirection = -1;
          }
          var rows = $table.find('tbody > tr').get();
          $.each(rows, function(index, row) {
            var $cell = $(row).children('td').eq(column);
            row.sortKey = findSortKey($cell);
          });
          rows.sort(function(a, b) {
            if (a.sortKey < b.sortKey) return -sortDirection;
            if (a.sortKey > b.sortKey) return sortDirection;
            return 0;
          });
          $.each(rows, function(index, row) {
            $table.children('tbody').append(row);
            row.sortKey = null;
          });
          $table.find('th').removeClass('sorted-asc')
            .removeClass('sorted-desc');
          if (sortDirection == 1) {
            $header.addClass('sorted-asc');
          }
          else {
            $header.addClass('sorted-desc');
          }
          $table.find('td').removeClass('sorted')
            .filter(':nth-child(' + (column + 1) + ')')
            .addClass('sorted');
          $table.alternateRowColors();
          $table.trigger('repaginate');
        });
      }
    });
  });
});

$(document).ready(function() {
  $('table.paginated').each(function() {
    var currentPage = 0;
    var numPerPage = 10;
    var $table = $(this);
    $table.bind('repaginate', function() {
      $table.find('tbody tr').hide()
        .slice(currentPage * numPerPage,
          (currentPage + 1) * numPerPage)
        .show();
    });
    var numRows = $table.find('tbody tr').length;
    var numPages = Math.ceil(numRows / numPerPage);
```

```
    var $pager = $('<div class="pager"></div>');
    for (var page = 0; page < numPages; page++) {
      $('<span class="page-number"></span>').text(page + 1)
        .bind('click', {newPage: page}, function(event) {
          currentPage = event.data['newPage'];
          $table.trigger('repaginate');
          $(this).addClass('active')
            .siblings().removeClass('active');
        }).appendTo($pager).addClass('clickable');
    }
    $pager.insertBefore($table)
      .find('span.page-number:first').addClass('active');
  });
});
```

7.2 Modifier l'aspect de la table

Nous avons examiné plusieurs manières d'ordonner les lignes de données d'une table pour que l'utilisateur puisse trouver facilement les informations qu'il recherche. Toutefois, il arrive souvent que même après un tri ou une pagination la quantité de données à examiner soit encore très importante. Nous pouvons aider l'utilisateur en manipulant non seulement l'ordre et la quantité de lignes affichées, mais également leur présentation.

Mettre une ligne en exergue

Pour orienter le regard de l'utilisateur, une solution pratique consiste à *mettre en exergue* des lignes, ce qui donne une indication visuelle sur l'importance des données. Afin d'examiner quelques stratégies de mise en exergue, nous avons besoin d'une table adaptée. Cette fois, nous prenons une table contenant des éléments d'actualité. Elle est un peu plus complexe que la précédente, car certaines lignes correspondent à des *sous-titres*, en plus des titres principaux. Voici la structure HTML correspondante :

```
<table>
  <thead>
    <tr>
      <th>Date</th>
      <th>Intitulé</th>
      <th>Auteur</th>
      <th>Sujet</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th colspan="4">2008</th>
    </tr>
    <tr>
      <td>28 sept</td>
        <td>jQuery, Microsoft, and Nokia</td>
        <td>John Resig</td>
        <td>tierce partie</td>
    </tr>
    ...
```

```
      <tr>
        <td>15 janv</td>
        <td>jQuery 1.2.2: 2nd Birthday Present</td>
        <td>John Resig</td>
        <td>sortie</td>
      </tr>
    </tbody>
    <tbody>
      <tr>
        <th colspan="4">2007</th>
      </tr>
      <tr>
        <td>8 déc</td>
        <td>jQuery Plugins site updated</td>
        <td>Mike Hostetler</td>
        <td>annonce</td>
      </tr>
      ...
      <tr>
        <td>11 janv</td>
        <td>Selector Speeds</td>
        <td>John Resig</td>
        <td>source</td>
      </tr>
    </tbody>
    ...
  </table>
```

Notez les multiples sections `<tbody>`. Ce balisage HTML valide permet de regrouper des lignes. Nous avons placé les sous-titres de section dans ces groupes, en utilisant des éléments `<th>` pour les séparer. La Figure 7.17 montre l'aspect de cette table, après avoir défini quelques styles CSS de base.

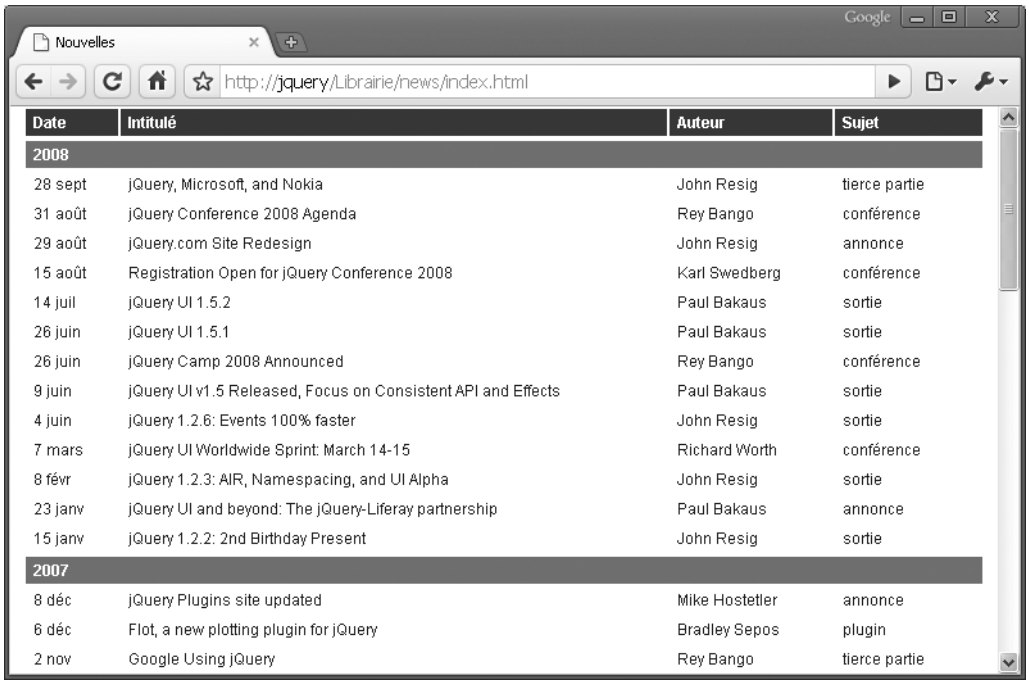
Effet de bande sur les lignes

Nous avons déjà vu un exemple simple de mise en exergue des lignes dans ce chapitre, ainsi qu'au Chapitre 2. L'*effet de bande* est souvent utilisé pour orienter précisément le regard de l'utilisateur sur différentes colonnes.

Pour mettre en œuvre cet effet, il suffit de quelques lignes de code qui ajoutent des classes aux lignes paires et impaires :

```
$(document).ready(function() {
  $('table.striped tr:odd').addClass('odd');
  $('table.striped tr:even').addClass('even');
});
```

Si ce code fonctionne parfaitement avec les structures de table simples, le schéma pair-impair n'est plus satisfaisant en cas d'ajout de lignes qui ne doivent pas être prises en compte dans l'effet, comme les lignes de sous-titres utilisées pour les années dans notre table. Par exemple, la ligne 2006 correspond à une ligne paire et celles qui viennent avant et après sont toutes deux impaires. La Figure 7.18 montre que le résultat obtenu n'est pas celui attendu.



Date	Intitulé	Auteur	Sujet
2008			
28 sept	jQuery, Microsoft, and Nokia	John Resig	tierce partie
31 août	jQuery Conference 2008 Agenda	Rey Bango	conférence
29 août	jQuery.com Site Redesign	John Resig	annonce
15 août	Registration Open for jQuery Conference 2008	Karl Swedberg	conférence
14 juil	jQuery UI 1.5.2	Paul Bakaus	sortie
26 juin	jQuery UI 1.5.1	Paul Bakaus	sortie
26 juin	jQuery Camp 2008 Announced	Rey Bango	conférence
9 juin	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	sortie
4 juin	jQuery 1.2.6: Events 100% faster	John Resig	sortie
7 mars	jQuery UI Worldwide Sprint: March 14-15	Richard Worth	conférence
8 févr	jQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	sortie
23 janv	jQuery UI and beyond: The jQuery-Liferay partnership	Paul Bakaus	annonce
15 janv	jQuery 1.2.2: 2nd Birthday Present	John Resig	sortie
2007			
8 déc	jQuery Plugins site updated	Mike Hostetler	annonce
6 déc	Flot, a new plotting plugin for jQuery	Bradley Sepos	plugin
2 nov	Google Using jQuery	Rey Bango	tierce partie

Figure 7.17

Aspect initial de la table.

14 janv	jQuery Birthday: 1.1, New Site, New Docs	John Resig	annonce
13 janv	jQuery wallpapers	Nate Cavanaugh	divers
11 janv	Selector Speeds	John Resig	source
2006			
27 déc	The Path to 1.1	John Resig	source
18 déc	Meet The People Behind jQuery	John Resig	annonce
13 déc	Helping you understand jQuery	John Resig	didacticiel

Figure 7.18

Le schéma pair-impair n'est pas satisfaisant.

En utilisant la pseudo-classe `:nth-child()` présentée au Chapitre 2, nous pouvons faire en sorte que l'alternance du motif reprenne au début après chaque ligne de sous-titre (voir Figure 7.19) :

```
$(document).ready(function() {
  $('table.striped tr:nth-child(odd)').addClass('odd');
  $('table.striped tr:nth-child(even)').addClass('even');
});
```

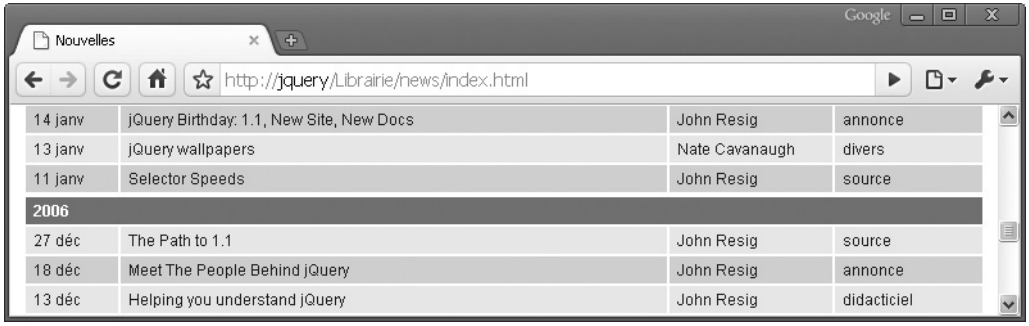


Figure 7.19

L'alternance des bandes reprend au début après une ligne de sous-titre.

Chaque groupe de lignes commence à présent par une ligne impaire, mais les lignes de sous-titre font partie du calcul. Pour ne plus prendre en compte ces lignes, nous utilisons la pseudo-classe `:has()` (voir Figure 7.20) :

```
$(document).ready(function() {
  $('table.striped tr:not(:has(th)):odd').addClass('odd');
  $('table.striped tr:not(:has(th)):even').addClass('even');
});
```

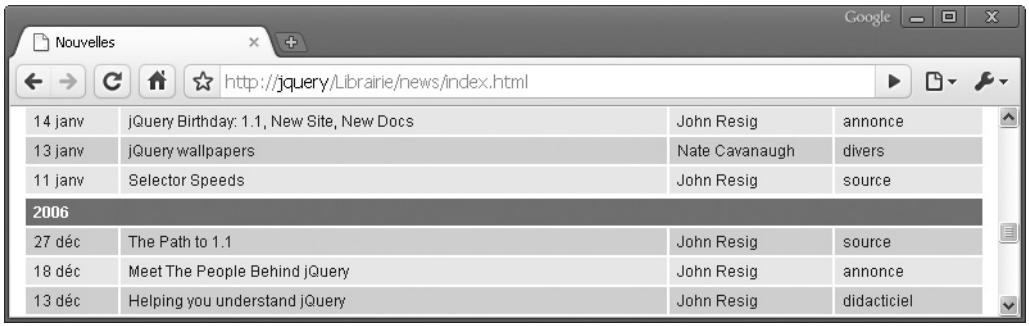


Figure 7.20

Les lignes de sous-titre ne sont plus prises en compte.

Les sous-titres sont à présent exclus, mais les groupes commencent par une ligne paire ou impaire en fonction de la classe appliquée à la ligne de données précédente. Pour concilier ces deux comportements, la solution risque d'être difficile à trouver. Une approche simple se fonde sur une itération explicite avec la méthode `.each()` :

```
$(document).ready(function() {
  $('table.striped tbody').each(function() {
    $(this).find('tr:not(:has(th)):odd').addClass('odd');
    $(this).find('tr:not(:has(th)):even').addClass('even');
 });
});
```

Dans ce cas, les bandes sont appliquées à chaque groupe de manière indépendante et les lignes de sous-titre sont exclues du traitement (voir Figure 7.21).

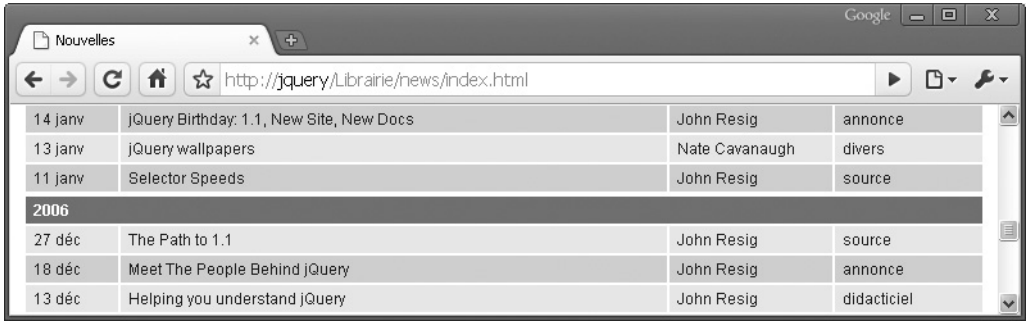


Figure 7.21

Traitement indépendant des groupes de lignes.

Effet de bande élaboré

Ces manipulations des lignes paires et impaires nous ont conduits à mettre en œuvre des techniques complexes. Dans les tables particulièrement denses, l'alternance de la couleur des lignes peut perturber le regard et il pourrait être préférable d'alterner les couleurs sur un intervalle plus long. Pour prendre un exemple, nous modifierons l'effet de bande sur notre table en changeant de couleur toutes les trois lignes.

Au Chapitre 2, nous avons présenté la méthode `.filter()`, qui permet de sélectionner des éléments de la page de manière très souple. En se rappelant que `.filter()` peut prendre non seulement une expression de sélection, mais également une *fonction de filtre*, nous écrivons le code suivant :

```
$(document).ready(function() {
  $('table.striped tbody').each(function() {
    $(this).find('tr:not(:has(th))').filter(function(index) {
      return (index % 6) < 3;
    }).addClass('odd');
  });
});
```

En peu de lignes, ce code réalise beaucoup de choses. Nous allons donc l'étudier morceau par morceau.

Comme précédemment, nous utilisons la méthode `.each()` pour traiter séparément les groupes de lignes. Puisque les bandes de trois lignes doivent reprendre après chaque sous-titre, cette technique nous permet de travailler une section à la fois. Ensuite, nous invoquons `.find()`, comme dans notre dernier exemple, pour localiser toutes les lignes qui ne contiennent pas d'éléments `<th>`, et qui ne sont donc pas des sous-titres.

Ensuite, nous devons sélectionner les trois premiers éléments de la collection obtenue, sauter trois éléments, et ainsi de suite. C'est là où la méthode `.filter()` entre en scène. La fonction de filtre prend un argument qui contient l'indice de l'élément dans la collection obtenue, c'est-à-dire le numéro de la ligne dans la section de la table en cours de manipulation. Si, et uniquement si, notre fonction de filtre retourne `true`, l'élément reste dans la collection.

L'*opérateur modulo* (`%`) nous fournit l'information dont nous avons besoin. L'expression `index % 6` donne le reste de la division du numéro de la ligne par six. Si ce reste est égal à 0, 1 ou 2, nous marquons la ligne comme étant impaire. S'il vaut 3, 4 ou 5, la ligne est paire.

Tel que présenté, le code marque uniquement les ensembles impairs de ligne. Pour appliquer également la classe `even`, nous pouvons écrire un autre filtre opérant de manière inverse ou être un peu plus astucieux :

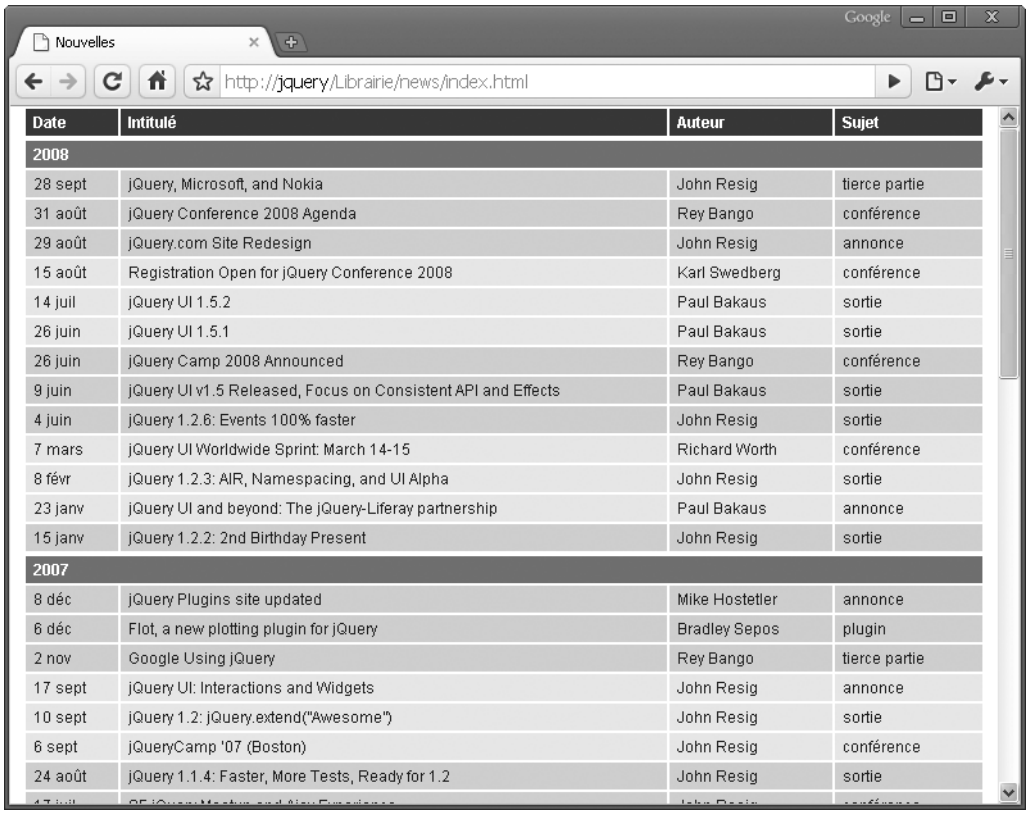
```
$(document).ready(function() {
  $('table.striped tbody').each(function() {
    $(this).find('tr:not(:has(th))').addClass('even')
      .filter(function(index) {
        return (index % 6) < 3;
    }).removeClass('even').addClass('odd');
  });
});
```

Ce code applique la classe `even` à toutes les lignes, puis la retire lorsque nous ajoutons la classe `odd`. Les bandes sont ainsi appliquées par groupes de lignes, avec une reprise au début de chaque section de la table (voir Figure 7.22).

Mise en exergue interactive

Nous pouvons apporter une autre amélioration visuelle à notre table d'articles d'actualité en fondant la mise en exergue des lignes sur les actions de l'utilisateur. Dans notre exemple, nous répondrons au clic sur le nom d'un auteur en mettant en exergue toutes les lignes qui contiennent ce nom dans la cellule AUTEUR. Comme nous l'avons fait pour les bandes, nous pouvons modifier l'aspect de ces lignes mises en exergue en ajoutant une classe :

```
#content tr.highlight {
  background: #ff6;
}
```



Date	Intitulé	Auteur	Sujet
2008			
28 sept	jQuery, Microsoft, and Nokia	John Resig	tierce partie
31 août	jQuery Conference 2008 Agenda	Rey Bango	conférence
29 août	jQuery.com Site Redesign	John Resig	annonce
15 août	Registration Open for jQuery Conference 2008	Karl Swedberg	conférence
14 juil	jQuery UI 1.5.2	Paul Bakaus	sortie
26 juin	jQuery UI 1.5.1	Paul Bakaus	sortie
26 juin	jQuery Camp 2008 Announced	Rey Bango	conférence
9 juin	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	sortie
4 juin	jQuery 1.2.6: Events 100% faster	John Resig	sortie
7 mars	jQuery UI Worldwide Sprint: March 14-15	Richard Worth	conférence
8 févr	jQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	sortie
23 janv	jQuery UI and beyond: The jQuery-Liferay partnership	Paul Bakaus	annonce
15 janv	jQuery 1.2.2: 2nd Birthday Present	John Resig	sortie
2007			
8 déc	jQuery Plugins site updated	Mike Hostetler	annonce
6 déc	Flot, a new plotting plugin for jQuery	Bradley Sepos	plugin
2 nov	Google Using jQuery	Rey Bango	tierce partie
17 sept	jQuery UI: Interactions and Widgets	John Resig	annonce
10 sept	jQuery 1.2: jQuery.extend("Awesome")	John Resig	sortie
6 sept	jQueryCamp '07 (Boston)	John Resig	conférence
24 août	jQuery 1.1.4: Faster, More Tests, Ready for 1.2	John Resig	sortie

Figure 7.22

Le changement de couleur intervient toutes les trois lignes.

Puisqu'il est important que cette nouvelle classe permette de distinguer les lignes mises en exergue, nous choisissons de changer la couleur d'arrière-plan par rapport à celle des classes `even` et `odd`.

Nous devons à présent sélectionner la cellule appropriée et lui associer un comportement à l'aide de la méthode `.click()` :

```
$(document).ready(function() {
  var $authorCells = $('table.striped td:nth-child(3)');
  $authorCells.click(function() {
    // Effectuer la mise en exergue.
  });
});
```

Nous utilisons la pseudo-classe `:nth-child(n)` dans l'expression de sélection. Cela nous permet de désigner la troisième colonne qui contient l'information concernant l'auteur. Si la structure de la table venait à changer, il faudrait mettre cette constante `3`

en un seul endroit afin qu'elle soit facile à modifier. C'est pourquoi, ainsi que pour une question d'efficacité, nous enregistrons le résultat du sélecteur dans la variable `$authorCells` au lieu de le répéter chaque fois qu'il est requis.

ATTENTION

N'oubliez pas que, contrairement aux indices JavaScript, la pseudo-classe `:nth-child(n)` de CSS commence à compter à partir de un, non de zéro.

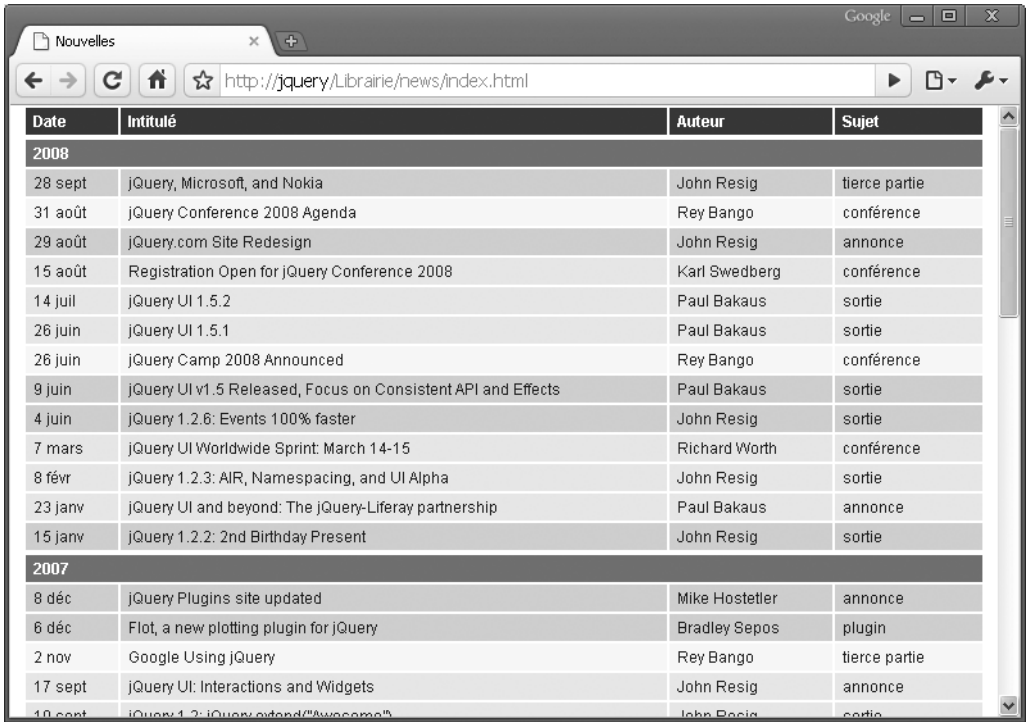
Lorsque l'utilisateur clique sur une cellule de la troisième colonne, nous voulons comparer le texte de cette cellule à celui de la cellule correspondante dans chaque autre ligne. S'ils sont identiques, la présence de la classe `highlight` est inversée. Autrement dit, la classe est ajoutée si elle n'est pas déjà présente et retirée dans le cas contraire. Nous pouvons ainsi cliquer sur une cellule d'auteur pour retirer la mise en exergue de la ligne si nous avons déjà cliqué sur cette cellule ou une autre contenant le même auteur.

```
$(document).ready(function() {
  var $authorCells = $('table.striped td:nth-child(3)');
  $authorCells.click(function() {
    var authorName = $(this).text();
    $authorCells.each(function(index) {
      if (authorName == $(this).text()) {
        $(this).parent().toggleClass('highlight');
      }
    });
  });
});
```

Le code fonctionne parfaitement, excepté lorsque l'utilisateur clique à la suite sur deux noms d'auteurs différents. Au lieu de basculer la mise en exergue des lignes qui correspondent au premier auteur vers celles qui correspondent au suivant, les deux groupes de lignes finissent avec la classe `highlight`. Pour éviter ce comportement, nous ajoutons une instruction `else` dans laquelle nous supprimons la classe `highlight` sur toutes les lignes qui ne contiennent pas le nom de l'auteur sélectionné :

```
$(document).ready(function() {
  var $authorCells = $('table.striped td:nth-child(3)');
  $authorCells.click(function() {
    var authorName = $(this).text();
    $authorCells.each(function(index) {
      if (authorName == $(this).text()) {
        $(this).parent().toggleClass('highlight');
      }
      else {
        $(this).parent().removeClass('highlight');
      }
    });
  });
});
```

À présent, si nous cliquons sur Rey Bango, par exemple, tous les articles contenant cet auteur sont beaucoup plus faciles à repérer (voir Figure 7.23).



Date	Intitulé	Auteur	Sujet
2008			
28 sept	jQuery, Microsoft, and Nokia	John Resig	tierce partie
31 août	jQuery Conference 2008 Agenda	Rey Bango	conférence
29 août	jQuery.com Site Redesign	John Resig	annonce
15 août	Registration Open for jQuery Conference 2008	Karl Swedberg	conférence
14 juil	jQuery UI 1.5.2	Paul Bakaus	sortie
26 juin	jQuery UI 1.5.1	Paul Bakaus	sortie
26 juin	jQuery Camp 2008 Announced	Rey Bango	conférence
9 juin	jQuery UI v1.5 Released, Focus on Consistent API and Effects	Paul Bakaus	sortie
4 juin	jQuery 1.2.6: Events 100% faster	John Resig	sortie
7 mars	jQuery UI Worldwide Sprint: March 14-15	Richard Worth	conférence
8 févr	jQuery 1.2.3: AIR, Namespacing, and UI Alpha	John Resig	sortie
23 janv	jQuery UI and beyond: The jQuery-Liferay partnership	Paul Bakaus	annonce
15 janv	jQuery 1.2.2: 2nd Birthday Present	John Resig	sortie
2007			
8 déc	jQuery Plugins site updated	Mike Hostetler	annonce
6 déc	Flot, a new plotting plugin for jQuery	Bradley Sepos	plugin
2 nov	Google Using jQuery	Rey Bango	tierce partie
17 sept	jQuery UI: Interactions and Widgets	John Resig	annonce

Figure 7.23

Les articles du même auteur (Rey Bango) sont mis en exergue.

Si nous cliquons ensuite sur John Resig dans l'une des cellules, les lignes qui correspondent aux articles de Rey Bango ne sont plus mises en exergue, contrairement à celles de John Resig.

Info-bulles

Bien que la mise en exergue des lignes soit une fonctionnalité utile, rien n'indique à l'utilisateur qu'elle existe. Nous pouvons remédier à cette situation en attribuant à toutes les cellules AUTEUR la classe `clickable`, dont la règle de style modifie l'apparence du pointeur de la souris lorsqu'il se trouve sur la cellule :

```
$(document).ready(function() {
  var $authorCells = $('table.striped td:nth-child(3)');
  $authorCells
    .addClass('clickable')
```

```
    .click(function() {
      var authorName = $(this).text();
      $authorCells.each(function(index) {
        if (authorName == $(this).text()) {
          $(this).parent().toggleClass('highlight');
        }
        else {
          $(this).parent().removeClass('highlight');
        }
      });
    });
});
```

La classe `clickable` va dans le bon sens, mais l'utilisateur ne sait toujours pas ce qui se produira s'il clique sur la cellule. Pour autant qu'il puisse l'imaginer, ce clic déclenchera certainement un autre comportement, comme le conduire vers une autre page. Il faut donc lui fournir des indications sur les effets de ses actions.

Les *info-bulles* sont utilisées par bon nombre d'applications logicielles, y compris les navigateurs web. Nous pouvons répondre à notre problème d'utilisabilité en affichant une info-bulle lorsque le pointeur de la souris survole une cellule AUTEUR. Le texte de l'info-bulle peut décrire aux utilisateurs les résultats de l'action, avec un message du type "Mettre en exergue tous les articles rédigés par Rey Bango.". Ce message sera placé dans un `<div>`, lui-même ajouté à l'élément `<body>`. La variable `$tooltip` sera utilisée tout au long du script pour faire référence à ce nouvel élément créé :

```
var $tooltip = $('<div id="tooltip"></div>').appendTo('body');
```

Voici les trois opérations de base que nous effectuerons à plusieurs reprises sur notre info-bulle :

1. Afficher l'info-bulle lorsque le pointeur de la souris se trouve au-dessus de l'élément interactif.
2. Masquer l'info-bulle lorsque le pointeur de la souris quitte la zone.
3. Repositionner l'info-bulle lorsque le pointeur de la souris se déplace.

Nous allons écrire des fonctions pour chacune de ces tâches, puis nous les associerons aux événements du navigateur à l'aide de jQuery.

Commençons par `positionTooltip()`, qui sera invoquée lorsque le pointeur de la souris se déplace au-dessus d'une cellule AUTEUR :

```
var positionTooltip = function(event) {
  var tPosX = event.pageX;
  var tPosY = event.pageY + 20;
  $tooltip.css({top: tPosY, left: tPosX});
};
```

Nous examinons les propriétés `pageX` et `pageY` de l'objet `event` pour fixer le positionnement haut et gauche de l'info-bulle. Lorsque cette fonction est invoquée en réponse à un événement de la souris, comme `mousemove`, `event.pageX` et `event.pageY` contiennent les coordonnées du pointeur de la souris. Par conséquent, `tPosX` et `tPosY` désignent un emplacement sur l'écran qui se trouve à vingt pixels sous le pointeur de la souris.

Nous passons ensuite à la fonction `showTooltip()`, qui affiche l'info-bulle à l'écran :

```
var showTooltip = function(event) {
  var authorName = $(this).text();
  $tooltip
    .text('Mettre en exergue tous les articles rédigés par ' + authorName)
    .show();
  positionTooltip(event);
};
```

Cette fonction est relativement simple. Nous remplissons le texte de l'info-bulle en utilisant une chaîne de caractères construite à partir du contenu de la cellule (c'est-à-dire le nom de l'auteur) et l'affichons.

La fonction `positionTooltip()` positionne ensuite l'info-bulle à l'emplacement approprié sur la page. Puisque l'info-bulle a été ajoutée à l'élément `<body>`, nous devons définir une règle CSS pour qu'elle flotte au-dessus de la page à l'emplacement calculé :

```
#tooltip {
  position: absolute;
  z-index: 2;
  background: #efd;
  border: 1px solid #ccc;
  padding: 3px;
}
```

Enfin, nous écrivons une fonction `hideTooltip()` simple :

```
var hideTooltip = function() {
  $tooltip.hide();
};
```

Puisque nous disposons à présent des fonctions permettant d'afficher, de masquer et de positionner l'info-bulle, nous pouvons les intégrer à notre code :

```
$(document).ready(function() {
  var $authorCells = $('table.striped td:nth-child(3)');
  var $tooltip = $('<div id="tooltip"></div>').appendTo('body');
  var positionTooltip = function(event) {
    var tPosX = event.pageX;
    var tPosY = event.pageY + 20;
    $tooltip.css({top: tPosY, left: tPosX});
  };
  var showTooltip = function(event) {
    var authorName = $(this).text();
    $tooltip
      .text('Mettre en exergue tous les articles rédigés par ' + authorName)
      .show();
```

```
      positionTooltip(event);
    };
    var hideTooltip = function() {
      $tooltip.hide();
    };
    $authorCells
      .addClass('clickable')
      .hover(showTooltip, hideTooltip)
      .mousemove(positionTooltip)
      .click(function(event) {
        var authorName = $(this).text();
        $authorCells.each(function(index) {
          if (authorName == $(this).text()) {
            $(this).parent().toggleClass('highlight');
          }
          else {
            $(this).parent().removeClass('highlight');
          }
        });
      });
});
```

Notez que les arguments des méthodes `.hover()` et `.mousemove()` font référence à des fonctions qui sont définies ailleurs. C'est pourquoi nous omettons les parenthèses, qui sinon déclencheraient l'invocation des fonctions. L'info-bulle apparaît dès que le pointeur de la souris passe au-dessus d'une cellule AUTEUR, suit le déplacement de la souris et disparaît lorsque le pointeur quitte la cellule (voir Figure 7.24).

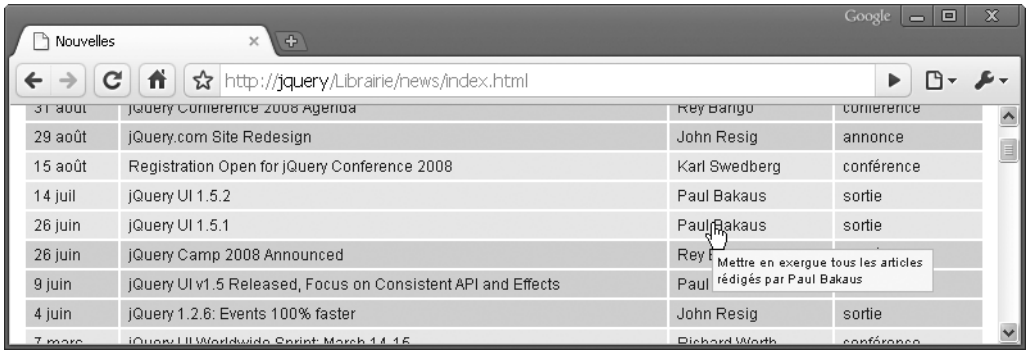


Figure 7.24

Une info-bulle d'explication apparaît sous le pointeur de la souris.

L'implémentation actuelle fonctionne, mais l'info-bulle suggère de cliquer sur une cellule pour mettre en exergue les articles, même s'ils le sont déjà (voir Figure 7.25).

Nous devons modifier le texte de l'info-bulle lorsque la ligne pointée possède la classe `highlight`. Pour cela, nous ajoutons une instruction conditionnelle dans la fonction

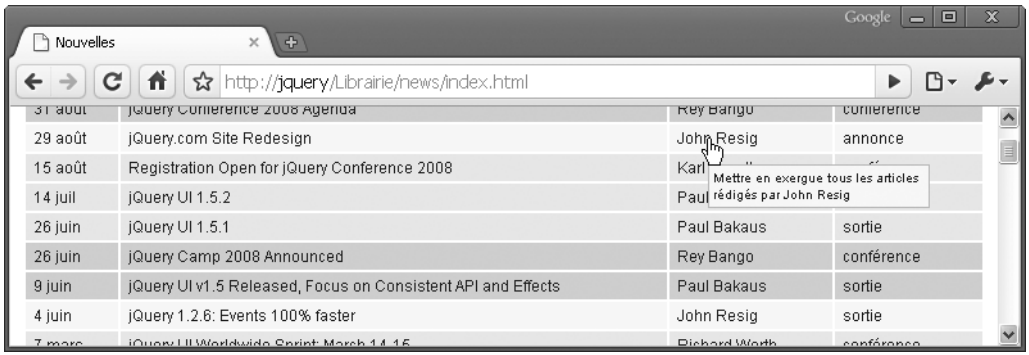


Figure 7.25

L'indication donnée par l'info-bulle n'est pas correcte.

`showTooltip()` de manière à vérifier la présence de la classe. Selon que l'élément `<tr>` parent de la cellule pointée possède ou non la classe `highlight`, nous changeons le texte de l'info-bulle :

```
var action = 'Mettre en exergue';
if ($(this).parent().is('.highlight')) {
  action = 'Ne plus mettre en exergue';
}
$tooltip
  .text(action + ' tous les articles rédigés par ' + authorName)
  .show();
```

Le texte de l'info-bulle est corrigé lorsque le pointeur de la souris entre dans une cellule, mais nous devons également l'adapter chaque fois que l'utilisateur clique. Pour cela, nous invoquons la fonction `showTooltip()` depuis le gestionnaire de l'événement `click` :

```
$(document).ready(function() {
  var $authorCells = $('table.striped td:nth-child(3)');
  var $tooltip = $('<div id="tooltip"></div>').appendTo('body');
  var positionTooltip = function(event) {
    var tPosX = event.pageX;
    var tPosY = event.pageY + 20;
    $tooltip.css({top: tPosY, left: tPosX});
  };
  var showTooltip = function(event) {
    var authorName = $(this).text();
    var action = 'Mettre en exergue';
    if ($(this).parent().is('.highlight')) {
      action = 'Ne plus mettre en exergue';
    }
    $tooltip
      .text(action + ' tous les articles rédigés par ' + authorName)
      .show();
    positionTooltip(event);
  };
```

```
    var hideTooltip = function() {
      $tooltip.hide();
    };
    $authorCells
      .addClass('clickable')
      .hover(showTooltip, hideTooltip)
      .mousemove(positionTooltip)
      .click(function(event) {
        var authorName = $(this).text();
        $authorCells.each(function(index) {
          if (authorName == $(this).text()) {
            $(this).parent().toggleClass('highlight');
          }
          else {
            $(this).parent().removeClass('highlight');
          }
        });
        showTooltip.call(this, event);
      });
});
```

Avec la fonction JavaScript `call()`, nous pouvons invoquer `showTooltip()` comme si elle s'exécutait dans la portée du gestionnaire de `click` associée à la cellule contenant le nom de l'auteur. Lorsqu'une fonction est appelée, le mot clé `this` correspond à l'objet JavaScript qui l'a appelée. La méthode JavaScript `call()` permet de contourner cette règle en précisant en premier argument la valeur souhaitée de `this`, dans ce cas la cellule de la table.

L'info-bulle affiche une indication plus correcte lorsque le pointeur de la souris survole une ligne qui est déjà mise en exergue (voir Figure 7.26).

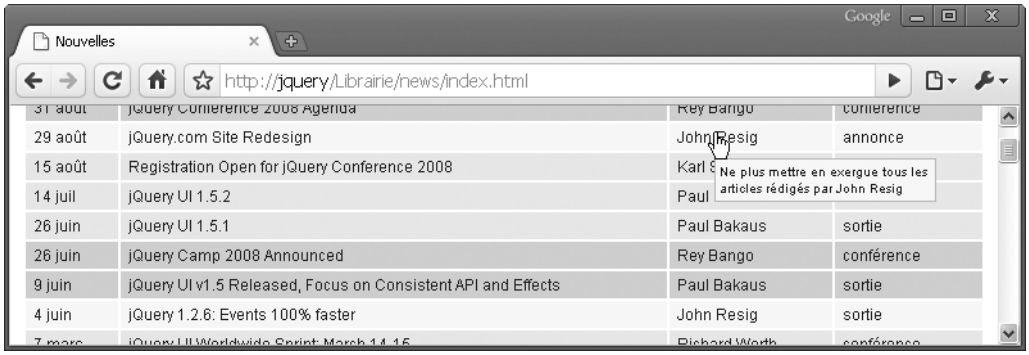


Figure 7.26

L'indication donnée par l'info-bulle est adaptée à la situation.

Réduire et développer des sections

Lorsqu'un jeu de données volumineux est divisé en sections, il peut être utile de masquer les informations qui ne présentent pas d'intérêt à un moment donné. Dans notre table des articles d'actualité, les lignes sont regroupées par année. Réduire, ou masquer, les articles d'une année peut être une bonne manière d'obtenir une vue d'ensemble de toutes les données de la table sans avoir à les faire défiler.

Pour que les sections de la table soient escamotables, nous devons tout d'abord créer un élément qui servira à déclencher ce comportement. La possibilité de réduction d'une section est souvent représentée par un signe moins, tandis que la fonctionnalité de développement est signalée par un signe plus. Nous insérons les icônes correspondantes en JavaScript, en suivant les techniques d'*amélioration progressive* standard :

```
$(document).ready(function() {
  var collapseIcon = '../images/bullet_toggle_minus.png';
  var collapseText = 'Réduire cette section';
  var expandIcon = '../images/bullet_toggle_plus.png';
  var expandText = 'Développer cette section';
  $('table.collapsible tbody').each(function() {
    var $section = $(this);
    $('<img />').attr('src', collapseIcon)
      .attr('alt', collapseText)
      .prependTo($section.find('th'));
  });
});
```

Au début de la fonction, nous enregistrons dans des variables l'emplacement des icônes, ainsi que leur texte de remplacement. Nous pouvons ainsi les modifier et y faire référence très facilement. L'injection de l'image se fait dans une boucle `.each()`, qui nous sera utile plus tard lorsque nous devrons à nouveau faire référence à l'élément `<tbody>` englobant ; il sera accessible au travers de la variable `$section`.

Ensuite, nous devons faire en sorte que les icônes déclenchent la réduction et le développement des sections. L'ajout d'une classe `clickable` permet d'apporter le retour visuel nécessaire à l'utilisateur, et une classe appliquée à l'élément `<tbody>` nous permet de suivre l'état de visibilité des lignes :

```
$(document).ready(function() {
  var collapseIcon = '../images/bullet_toggle_minus.png';
  var collapseText = 'Réduire cette section';
  var expandIcon = '../images/bullet_toggle_plus.png';
  var expandText = 'Développer cette section';
  $('table.collapsible tbody').each(function() {
    var $section = $(this);
    $('<img />').attr('src', collapseIcon)
      .attr('alt', collapseText)
      .prependTo($section.find('th'))
      .addClass('clickable')
      .click(function() {
```

```
          if ($section.is('.collapsed')) {
            $section.removeClass('collapsed')
              .find('tr:not(:has(th))').fadeIn('fast');
            $(this).attr('src', collapseIcon)
              .attr('alt', collapseText);
          }
          else {
            $section.addClass('collapsed')
              .find('tr:not(:has(th))').fadeOut('fast');
            $(this).attr('src', expandIcon)
              .attr('alt', expandText);
          }
        });
    });
});
```

Voici les opérations effectuées lors d'un clic :

1. Ajouter ou retirer la classe `collapsed` sur l'élément `<tbody>`, pour le suivi de l'état actuel de la section correspondante de la table.
2. Localiser toutes les lignes de la section qui ne contiennent pas un intitulé et les afficher ou les masquer avec un effet de fondu.
3. Inverser l'état actuel de l'icône, en changeant ses attributs `src` et `alt` de manière à refléter le comportement qu'elle déclenchera désormais lors d'un clic.

Une fois que ce code est en place, un clic sur l'icône RÉDUIRE CETTE SECTION affichée à côté de 2007 permet d'obtenir la présentation de la table illustrée à la Figure 7.27.

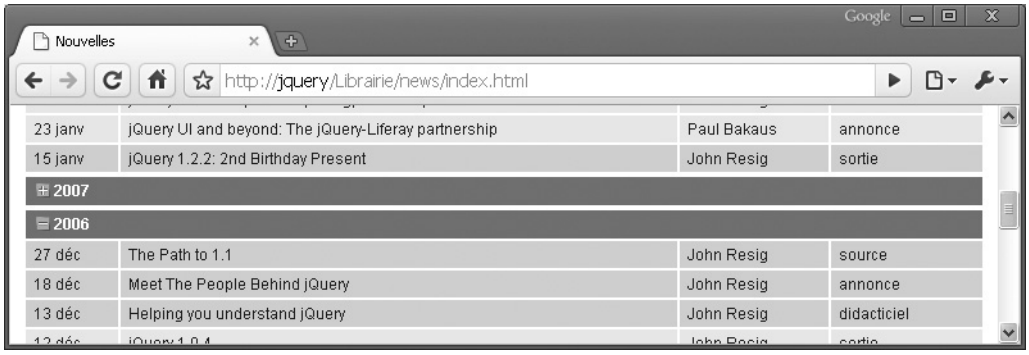


Figure 7.27

Réduction des articles de l'année 2007.

Les articles de l'année 2007 ne sont pas supprimés. Ils sont simplement masqués, jusqu'à ce que l'utilisateur clique sur l'icône DÉVELOPPER CETTE SECTION qui apparaît désormais sur cette ligne de sous-titre.

ATTENTION

Les lignes d'une table posent un problème particulier pour l'animation, car les navigateurs utilisent des valeurs différentes (`table-row` et `block`) pour la propriété de visibilité `display`. Les méthodes `.hide()` et `.show()` sans animation ont toujours un comportement cohérent pour les lignes d'une table. Si vous souhaitez ajouter une animation, vous pouvez également utiliser `.fadeIn()` et `.fadeOut()`.

Appliquer un filtre

Nous avons vu que le tri et la pagination sont des techniques qui permettent aux utilisateurs de se focaliser sur les données pertinentes d'une table. Elles peuvent toutes deux être mises en œuvre côté serveur ou en JavaScript. Le *filtrage* complète notre arsenal de stratégies de présentation des données. En affichant à l'utilisateur uniquement les lignes de la table qui correspondent à un critère donné, nous lui évitons d'être distrait par des informations inutiles.

Nous avons déjà étudié la réalisation d'un type de filtre : la *mise en exergue* d'un ensemble de lignes. Nous allons à présent développer cette idée en masquant les lignes qui ne correspondent pas au filtre.

Nous commençons par créer un emplacement pour les liens de filtrage. Conformément au principe d'amélioration progressive, nous ajoutons ces contrôles en utilisant du code JavaScript afin qu'ils ne soient proposés qu'aux internautes ayant activé la prise en charge des scripts :

```
$(document).ready(function() {
  $('table.filterable').each(function() {
    var $table = $(this);
    $table.find('th').each(function(column) {
      if ($(this).is('.filter-column')) {
        var $filters = $('<div class="filters"></div>');
        $('<h3></h3>')
          .text('Filtrer par ' + $(this).text() + ' :')
          .appendTo($filters);
        $filters.insertBefore($table);
      }
    });
  });
});
```

Pour pouvoir réutiliser ce code avec d'autres tables, nous récupérons le libellé du filtre à partir des en-têtes de colonnes. Nous disposons à présent d'un titre qui attend quelques boutons (voir Figure 7.28).

Figure 7.28

Les débuts du filtrage.

Options de filtrage

Nous pouvons à présent passer à la mise en œuvre d'un filtre. Pour commencer, nous allons ajouter des filtres pour deux sujets connus. Le code correspondant ressemble à l'exemple de mise en exergue des articles écrits par un auteur :

```
$(document).ready(function() {
  $('table.filterable').each(function() {
    var $table = $(this);
    $table.find('th').each(function(column) {
      if ($(this).is('.filter-column')) {
        var $filters = $('<div class="filters"></div>');
        $('<h3></h3>')
          .text('Filtrer par ' + $(this).text() + ' :')
          .appendTo($filters);
        var keywords = ['conférence', 'sortie'];
        $.each(keywords, function(index, keyword) {
          $('<div class="filter"></div>').text(keyword)
            .bind('click', {key: keyword}, function(event) {
              $('tr:not(:has(th))', $table).each(function() {
                var value = $('td', this).eq(column).text();
                if (value == event.data['key']) {
                  $(this).show();
                }
                else {
                  $(this).hide();
                }
              });
              $(this).addClass('active')
                .siblings().removeClass('active');
            }).addClass('clickable').appendTo($filters);
        });
        $filters.insertBefore($table);
      }
    });
  });
});
```

Nous partons d'un tableau statique de mots clés qui serviront de critères de tri. Nous le parcourons et créons un lien de filtre pour chacun des mots clés. Comme dans l'exemple de pagination, nous utilisons le paramètre `data` de la méthode `.bind()` pour éviter les problèmes liés aux *fermetures*. Ensuite, dans le gestionnaire de `click`, nous comparons le contenu de chaque cellule et le mot clé, et masquons la ligne s'ils ne correspondent pas. Puisque notre sélecteur de lignes exclut celles qui contiennent un élément `<th>`, nous n'avons pas besoin de nous préoccuper des sous-titres.

La Figure 7.29 montre les liens qui correspondent aux mots clés définis.

Figure 7.29

Application d'un filtre sur un critère prédéfini.

Obtenir les options de filtrage à partir du contenu

Nous devons étendre les options de filtrage de manière à couvrir l'ensemble des sujets contenus dans la table. Au lieu de figer tous les sujets dans le code, nous pouvons les obtenir à partir du texte affiché par la table. Nous modifions donc la définition de `keywords` :

```
var keywords = {};
$table.find('td:nth-child(' + (column + 1) + ')')
  .each(function() {
    keywords[$(this).text()] = $(this).text();
  });
```

Ce code se fonde sur deux astuces :

1. En utilisant une *mappe* à la place d'un *tableau* pour stocker les mots clés trouvés, nous supprimons automatiquement les doublons. Chaque clé ne peut avoir qu'une seule valeur, et les clés sont toujours uniques.

2. La fonction `$.each()` de jQuery nous permet de manipuler de la même manière des tableaux et des mappes. Le code n'a donc pas besoin d'être modifié.

Toutes les options de filtrage sont à présent disponibles (voir Figure 7.30).

Figure 7.30

Les critères du filtre sont obtenus depuis le contenu de la table.

Retirer le filtre

Nous voulons également proposer un moyen de revenir à la liste complète après l'avoir filtrée. Pour cela, nous ajoutons une option qui correspond à tous les sujets :

```
$('<div class="filter">Tous</div>').click(function() {
  $table.find('tbody tr').show();
  $(this).addClass('active')
    .siblings().removeClass('active');
}).addClass('clickable active').appendTo($filters);
```

Ce lien permet simplement d'afficher toutes les lignes de la table. Pour faire bonne mesure, il est initialement actif (voir Figure 7.31).

Interagir avec un autre code

Avec notre code pour le tri et la pagination, nous avons vu que nous ne pouvions pas développer isolément les différentes fonctionnalités. Les comportements peuvent parfois interagir de manière surprenante. Par conséquent, il est préférable de revenir sur notre travail précédent pour examiner sa coexistence avec les nouvelles possibilités de filtrage ajoutées.

Figure 7.31

Un lien permet d'afficher toutes les lignes de la table.

Effet de bande

L'effet de bande élaboré appliqué aux lignes est perturbé par la fonctionnalité de filtrage. Puisqu'il n'est pas réappliqué après le filtrage de la table, les lignes conservent leur couleur comme si les lignes masquées étaient toujours présentes.

Pour tenir compte des lignes filtrées, le code d'application des bandes doit être en mesure de les identifier. La pseudo-classe jQuery `:visible` peut nous aider à obtenir les lignes concernées par les bandes. Nous profitons de cette modification pour préparer le code de création des bandes à être invoqué depuis d'autres endroits du script. Pour cela, nous créons un type d'événement personnalisé, comme nous l'avons fait pour la combinaison du tri et de la pagination.

```
$(document).ready(function() {
  $('table.striped').bind('stripe', function() {
    $('tbody', this).each(function() {
      $(this).find('tr:visible:not(:has(th))')
        .removeClass('odd').addClass('even')
        .filter(function(index) {
          return (index % 6) < 3;
      }).removeClass('even').addClass('odd');
    });
  }).trigger('stripe');
});
```

Ce code fondé sur la pseudo-classe `:visible` fonctionne parfaitement dans les versions de jQuery antérieures à la 1.3.2. À partir de cette version, la visibilité d'un élément est détectée différemment et notre code n'est plus opérationnel. Nous remplaçons donc la ligne :

```
$(this).find('tr:visible:not(:has(th))')
```

par :

```
$(this).find('tr:not(:has(th))')
  .filter(function() {
    return this.style.display != 'none';
  })
```

Dans le code de filtrage, nous pouvons à présent invoquer `$table.trigger('stripe')` après chaque application d'un filtre. Avec le nouveau gestionnaire d'événements et ses déclenchements en place, l'effet de bande est respecté même après le filtrage de la table (voir Figure 7.32).

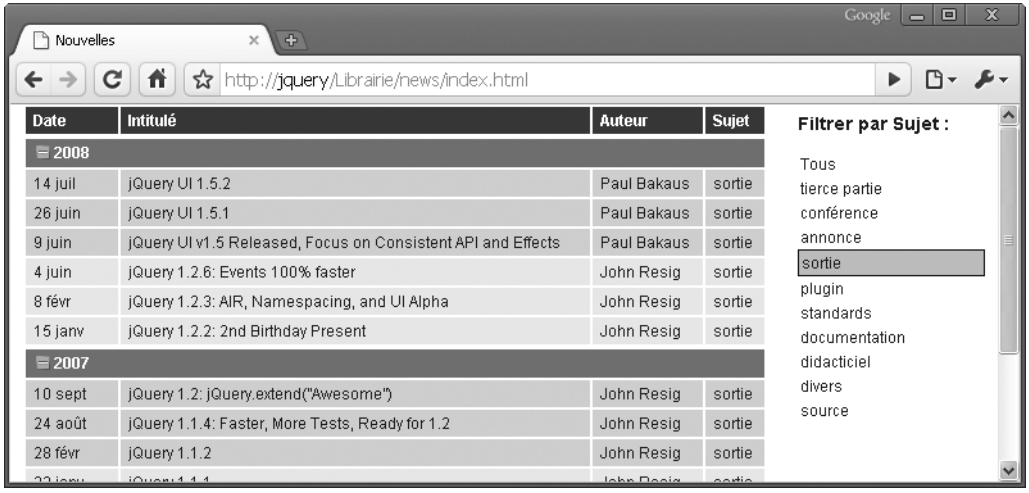


Figure 7.32

Application de l'effet de bande après filtrage.

Réduction et développement

La réduction et le développement des sections ajoutés précédemment entrent également en conflit avec les filtres. Lorsqu'une section est réduite et qu'un nouveau filtre est sélectionné, tous les éléments correspondants sont affichés même ceux qui se trouvent dans la section fermée. De même, lorsque la table est filtrée et qu'une section est développée, tous les éléments de cette section sont affichés, qu'ils correspondent ou non au filtre.

Pour répondre à ce dernier cas, une solution consiste à modifier l'affichage et le masquage des lignes. Si nous utilisons une classe pour indiquer qu'une ligne doit être masquée, il n'est plus nécessaire d'invoquer explicitement `.hide()` et `.show()`. En remplaçant `.hide()` par `.addClass('filtered')` et `.show()` par `.removeClass('fil-`

`tered')`, et en définissant une règle CSS pour la classe, nous pouvons obtenir un masquage et un affichage des lignes qui s'intègrent mieux au code de réduction. Si la classe est retirée et si la ligne est réduite, elle n'est plus affichée par mégarde.

L'ajout de cette nouvelle classe `filtered` nous facilite également la résolution du problème inverse. Nous pouvons tester la présence de `filtered` au moment du développement d'une section et sauter les lignes correspondantes au lieu de les afficher. Pour tester cette classe, il suffit d'ajouter `:not(.filtered)` à l'expression de sélection utilisée pour le développement des sections.

Les fonctionnalités coopèrent parfaitement, chacune pouvant afficher et masquer les lignes indépendamment.

Version finale du code

Notre deuxième exemple a illustré l'application d'un effet de bande à une table, la mise en exergue de lignes, les info-bulles, la réduction et le développement de sections, ainsi que le filtrage. Voici l'intégralité du code JavaScript qui correspond à cet exemple :

```
$(document).ready(function() {
  $('table.striped').bind('stripe', function() {
    $('tbody', this).each(function() {
      $(this).find('tr:not(:has(th))')
        .filter(function() {
          return this.style.display != 'none';
        })
        .removeClass('odd').addClass('even')
        .filter(function(index) {
          return (index % 6) < 3;
        }).removeClass('even').addClass('odd');
    });
  }).trigger('stripe');
});

$(document).ready(function() {
  var $authorCells = $('table.striped td:nth-child(3)');
  var $tooltip = $('<div id="tooltip"></div>').appendTo('body');
  var positionTooltip = function(event) {
    var tPosX = event.pageX;
    var tPosY = event.pageY + 20;
    $tooltip.css({top: tPosY, left: tPosX});
  };
  var showTooltip = function(event) {
    var authorName = $(this).text();
    var action = 'Mettre en exergue';
    if ($(this).parent().is('.highlight')) {
      action = 'Ne plus mettre en exergue';
    }
    $tooltip
      .text(action + ' tous les articles rédigés par ' + authorName)
      .show();
```

```
    positionTooltip(event);
  };
  var hideTooltip = function() {
    $tooltip.hide();
  };

  $authorCells
    .addClass('clickable')
    .hover(showTooltip, hideTooltip)
    .mousemove(positionTooltip)
    .click(function(event) {
      var authorName = $(this).text();
      $authorCells.each(function(index) {
        if (authorName == $(this).text()) {
          $(this).parent().toggleClass('highlight');
        }
        else {
          $(this).parent().removeClass('highlight');
        }
      });
      showTooltip.call(this, event);
    });
});

$(document).ready(function() {
  var collapseIcon = '../images/bullet_toggle_minus.png';
  var collapseText = 'Réduire cette section';
  var expandIcon = '../images/bullet_toggle_plus.png';
  var expandText = 'Développer cette section';
  $('table.collapsible tbody').each(function() {
    var $section = $(this);
    $('<img />').attr('src', collapseIcon)
      .attr('alt', collapseText)
      .prependTo($section.find('th'))
      .addClass('clickable')
      .click(function() {
        if ($section.is('.collapsed')) {
          $section.removeClass('collapsed')
            .find('tr:not(:has(th)):not(.filtered)')
            .fadeIn('fast');
          $(this).attr('src', collapseIcon)
            .attr('alt', collapseText);
        }
        else {
          $section.addClass('collapsed')
            .find('tr:not(:has(th))')
            .fadeOut('fast', function() {
              $(this).css('display', 'none');
            });
          $(this).attr('src', expandIcon)
            .attr('alt', expandText);
        }
        $section.parent().trigger('stripe');
      });
  });
});
```

```
$(document).ready(function() {
  $('table.filterable').each(function() {
    var $table = $(this);

    $table.find('th').each(function(column) {
      if ($(this).is('.filter-column')) {
        var $filters = $('<div class="filters"></div>');
        $('<h3></h3>')
          .text('Filtrer par ' + $(this).text() + ' :')
          .appendTo($filters);

        $('<div class="filter">all</div>').click(function() {
          $table.find('tbody tr').removeClass('filtered');
          $(this).addClass('active')
            .siblings().removeClass('active');
          $table.trigger('stripe');
        }).addClass('clickable active').appendTo($filters);

        var keywords = {};
        $table.find('td:nth-child(' + (column + 1) + ')')
          .each(function() {
            keywords[$(this).text()] = $(this).text();
          });

        $.each(keywords, function(index, keyword) {
          $('<div class="filter"></div>').text(keyword)
            .bind('click', {key: keyword}, function(event) {
              $('tr:not(:has(th))', $table).each(function() {
                var value = $('td', this).eq(column).text();
                if (value == event.data['key']) {
                  $(this).removeClass('filtered');
                }
                else {
                  $(this).addClass('filtered');
                }
              });
              $(this).addClass('active')
                .siblings().removeClass('active');
              $table.trigger('stripe');
            }).addClass('clickable').appendTo($filters);
        });

        $filters.insertBefore($table);
      }
    });
  });
});
```

7.3 En résumé

Dans ce chapitre, nous avons exploré plusieurs manières de manipuler les tables de nos sites, pour les transformer en conteneurs élégants et fonctionnels de nos données. Nous avons vu comment *trier* les données d'une table, en utilisant des clés de différents types (mots, nombres, dates), et comment la *paginer* en portions plus faciles à consulter. Nous

avons appris des techniques JavaScript sophistiquées permettant d'ajouter des *bandes aux lignes* et d'afficher des *info-bulles*. Nous avons également étudié la *réduction* et le *développement* de sections de contenu, ainsi que le *filtrage* et la *mise en exergue* des lignes qui correspondent à un critère donné.

Nous avons même abordé brièvement quelques sujets avancés, comme le tri et la pagination côté serveur avec des techniques AJAX, le calcul dynamique de la position des éléments sur la page et l'écriture d'un plugin jQuery.

Nous l'avons vu, les tables HTML dont la sémantique est correcte cachent un haut niveau de subtilité et de complexité. Heureusement, jQuery nous aide à maîtriser ces constructions, de manière à révéler tout l'intérêt des données tabulaires.

8

Manipulation des formulaires

Au sommaire de ce chapitre

- Améliorer un formulaire de base
- Formulaires compacts
- Manipuler des données numériques
- En résumé

Il est difficile de trouver un site web qui n'utilise pas un *formulaire* pour obtenir un retour de la part de l'utilisateur. Depuis les débuts d'Internet, les formulaires ont servi à transmettre des informations entre l'utilisateur final et l'éditeur du site web, certes de manière fiable, mais souvent avec peu d'élégance ou de style. Ce manque de classe était sans doute lié au voyage répétitif et ardu entre le serveur et le navigateur, ou bien provenait-il des éléments intransigeants imposés aux formulaires et de leur réticence à suivre la dernière mode. Quelle que soit la raison, ce n'est que récemment, avec le regain d'intérêt pour les scripts côté client, que les formulaires ont retrouvé un second souffle, un objectif et du style. Dans ce chapitre, nous verrons comment ressusciter les formulaires. Nous allons améliorer leur aspect, créer des procédures de validation, les employer pour des calculs et envoyer discrètement les résultats au serveur.

8.1 Améliorer un formulaire de base

Lorsqu'on utilise jQuery pour des sites web, il faut toujours s'interroger sur l'aspect et le fonctionnement des pages en cas d'indisponibilité de JavaScript (sauf, bien sûr, si l'on sait précisément qui sont les visiteurs et comment sont configurés leurs navigateurs). Attention, cela ne signifie pas que l'indisponibilité de JavaScript doive empêcher la création d'un site plus joli et aux fonctionnalités complètes. Le principe d'*amélioration progressive* est très répandu chez les développeurs JavaScript, car il respecte le besoin de tous les utilisateurs tout en proposant des fonctions supplémentaires

à la plupart. Pour illustrer ce principe dans le contexte des formulaires, nous allons créer un formulaire de contact, que nous améliorerons avec jQuery, tant au niveau de son aspect que de son fonctionnement.

Amélioration progressive de l'aspect

Tout d'abord, apportons quelques modifications esthétiques à notre formulaire. Lorsque JavaScript n'est pas activé, le premier jeu de champs s'affiche comme l'illustre la Figure 8.1.

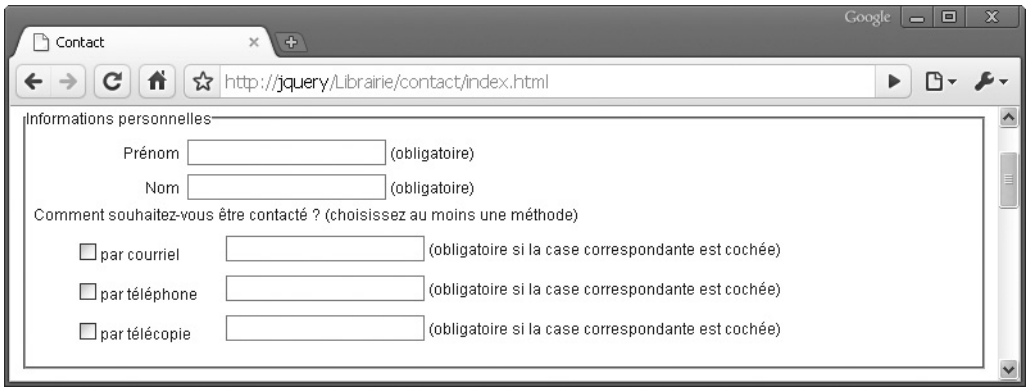


Figure 8.1

Présentation par défaut des champs du formulaire.

Si elle semble répondre aux besoins et explique aux utilisateurs comment remplir les champs, cette partie du formulaire mériterait d'être améliorée. Nous allons intervenir progressivement sur trois aspects :

1. Modifier le DOM pour que l'application d'un style aux balises `<legend>` soit plus flexible.
2. Remplacer le message (OBLIGATOIRE) sur les champs obligatoires par un astérisque (`*`) et le message (OBLIGATOIRE SI LA CASE CORRESPONDANTE EST COCHÉE) sur les champs spéciaux par deux astérisques (`**`). Afficher en gras le libellé de chaque champ obligatoire et placer une légende au début du formulaire pour expliquer la signification de l'astérisque et de l'astérisque double.
3. Masquer le champ de saisie correspondant à chaque case à cocher lors du chargement de la page, puis les afficher ou les masquer selon que l'utilisateur coche ou décoche les cases.

Nous commençons par le contenu HTML du `<fieldset>` :

```
<fieldset>
  <legend>Informations personnelles</legend>
  <ol>
    <li>
      <label for="first-name">Prénom</label>
      <input class="required" type="text" name="first-name" id="first-name" />
      <span>(obligatoire)</span>
    </li>
    <li>
      <label for="last-name">Nom</label>
      <input class="required" type="text" name="last-name" id="last-name" />
      <span>(obligatoire)</span>
    </li>
    <li>Comment souhaitez-vous être contacté ?
        (choisissez au moins une méthode)
      <ul>
        <li>
          <label for="by-email">
            <input type="checkbox" name="by-contact-type"
                value="E-mail" id="by-email" />
            par courriel
          </label>
          <input class="conditional" type="text" name="email" id="email" />
          <span>(obligatoire si la case correspondante est cochée)</span>
        </li>
        <li>
          <label for="by-phone">
            <input type="checkbox" name="by-contact-type"
                value="Phone" id="by-phone" />
            par téléphone
          </label>
          <input class="conditional" type="text" name="phone" id="phone" />
          <span>(obligatoire si la case correspondante est cochée)</span>
        </li>
        <li>
          <label for="by-fax">
            <input type="checkbox" name="by-contact-type"
                value="Fax" id="by-fax" />
            par télécopie
          </label>
          <input class="conditional" type="text" name="fax" id="fax" />
          <span>(obligatoire si la case correspondante est cochée)</span>
        </li>
      </ul>
    </li>
  </ol>
</fieldset>
```

Vous remarquerez que chaque élément d'information ou couple d'éléments est considéré comme un *élément de liste* (`<li>`). Tous les éléments sont placés dans une *liste ordonnée* (`<ol>`) et les cases à cocher, avec leur champ de texte, sont placées dans une *liste non ordonnée* imbriquée (`<ul>`). Par ailleurs, nous utilisons `<label>` pour préciser le nom des champs. Pour les champs de texte, `<label>` précède l'élément `<input>` ; pour les cases à cocher, il englobe `<input>`. Même s'il n'existe aucune structure stan-

dard pour les éléments d'un jeu de champs, la liste ordonnée semble adaptée à la représentation sémantique des éléments d'information dans un formulaire de contact.

Avec ce contenu HTML disponible, nous sommes prêts à utiliser jQuery pour procéder à une amélioration progressive.

La légende

La *légende* du formulaire est depuis tout temps un élément difficile à styler avec CSS. Les différences d'interprétation par les navigateurs et les limites sur le positionnement rendent cet exercice particulièrement périlleux. Néanmoins, si nous voulons utiliser des éléments de page bien structurés et véhiculant un sens, la légende est un choix séduisant, si ce n'est attrayant visuellement, pour ajouter un titre à notre élément `<fieldset>` du formulaire.

Avec uniquement un balisage HTML et des styles CSS, nous sommes forcés à un compromis entre un balisage sémantique et une conception flexible. Toutefois, rien ne nous empêche de modifier le balisage HTML lors du chargement de la page, de manière à convertir chaque `<legend>` en `<h3>` pour ceux qui consultent la page, tandis que les systèmes de lecture de la page, ainsi que les navigateurs sans JavaScript, continuent à voir l'élément `<legend>`. Grâce à la méthode jQuery `.replaceWith()`, cette opération est très simple :

```
$(document).ready(function() {
  $('legend').each(function(index) {
    $(this).replaceWith('<h3>' + $(this).text() + '</h3>');
  });
});
```

Dans ce cas, nous ne pouvons pas nous fonder sur l'itération implicite de jQuery. Pour chaque élément que nous remplaçons, nous devons insérer son contenu textuel correspondant. C'est pourquoi nous utilisons la méthode `.each()`, qui nous permet d'obtenir ce texte à partir de `$(this)`.

À présent, si nous modifions la feuille de style pour appliquer un fond bleu et une couleur blanche au texte des éléments `<h3>`, le premier jeu de champs du formulaire apparaît tel qu'illustré à la Figure 8.2.

Une autre solution permet de conserver les éléments `<legend>`, mais enveloppe leur contenu dans une balise `<span>` :

```
$(document).ready(function() {
  $('legend').wrapInner('<span></span>');
});
```

Cette approche présente au moins deux avantages par rapport au remplacement de `<legend>` par `<h3>` : elle garde la signification sémantique de `<legend>` pour les lec-

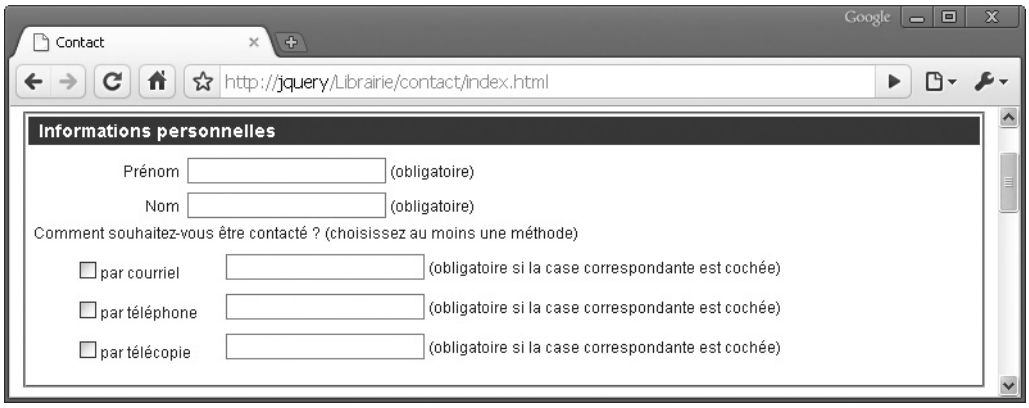


Figure 8.2

Remplacement de la légende par un niveau de titre.

teurs d'écran qui prennent en charge JavaScript et nécessite un travail moins important de la part du script. Elle a pour inconvénient de complexifier l'application du style à l'intitulé. Nous devons au moins fixer la propriété `position` des éléments `<fieldset>` et `<span>`, ainsi que les propriétés `padding-top` du `<fieldset>` et `width` du `<span>` :

```
fieldset {
  position: relative;
  padding-top: 1.5em;
}
legend span {
  position: absolute;
  width: 100%;
}
```

Que nous choisissions de remplacer les éléments `<legend>` du formulaire ou d'insérer un `<span>` à l'intérieur, ils sont à présent suffisamment améliorés pour nous satisfaire. Il est temps de passer au message des champs obligatoires.

Message des champs obligatoires

Dans le formulaire de contact, les champs obligatoires possèdent la classe `required`, ce qui nous permet de leur appliquer un style et d'analyser la saisie de l'utilisateur ; la classe `conditional` est appliquée aux champs de saisie qui correspondent à une méthode de contact. Nous allons utiliser ces classes pour modifier les instructions affichées entre parenthèses à droite des champs.

Nous commençons par initialiser les variables `requiredFlag` et `conditionalFlag`, puis nous remplissons l'élément `<span>` placé à côté de chaque champ obligatoire en utilisant le contenu de ces variables :

```
$(document).ready(function() {
  var requiredFlag = ' * ';
  var conditionalFlag = ' ** ';

  $('form :input')
    .filter('.required')
      .next('span').text(requiredFlag).end()
    .end()
    .filter('.conditional')
      .next('span').text(conditionalFlag);
});
```

En utilisant `.end()`, nous pouvons étendre la chaîne des méthodes de manière à continuer à travailler sur la même collection d'éléments et à minimiser le nombre d'objets créés et les traversées du DOM. Chaque appel à la méthode `.end()` ramène la sélection un pas en arrière et retourne les éléments qui étaient sélectionnés avant la dernière méthode de parcours. Dans le script, nous en utilisons deux à la suite : le premier appel à `.end()` retourne la collection obtenue par `.filter('.required')`, le second, celle fournie par `$('form :input')`. Ainsi, lorsque `.filter('.conditional')` sélectionne les éléments dont la classe est `conditional`, il examine l'ensemble des champs de saisie du formulaire.

Puisqu'un seul astérisque (`*`) ne suffira sans doute pas à attirer l'attention de l'utilisateur, nous ajoutons également la classe `req-label` à l'élément `<label>` de chaque champ obligatoire et appliquons `font-weight:bold` à cette classe, tout cela en poursuivant la chaîne :

```
$(document).ready(function() {
  var requiredFlag = ' * ';
  var conditionalFlag = ' ** ';

  $('form :input')
    .filter('.required')
      .next('span').text(requiredFlag).end()
      .prev('label').addClass('req-label').end()
    .end()
    .filter('.conditional')
      .next('span').text(conditionalFlag);
});
```

Une telle chaîne d'invocations risque d'être difficile à lire. C'est pourquoi une utilisation cohérente des passages à la ligne et de l'indentation est essentielle.

La Figure 8.3 présente le jeu de champs après modification du texte et ajout de la classe.

Pas mal pour un début. Néanmoins, les messages des champs obligatoires n'étaient pas inutiles, ils étaient simplement trop répétitifs. Reprenons la première instance de chaque message et affichons-la au-dessus du formulaire, à côté du symbole qui le représente.

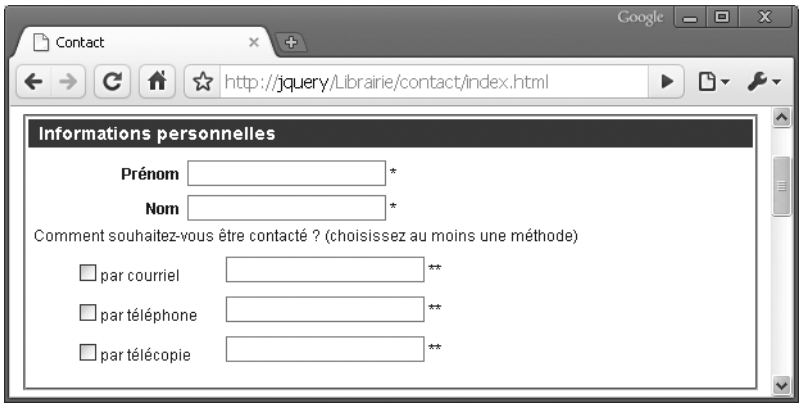


Figure 8.3

Traitement des messages des deux types de champs obligatoires.

Avant de remplir les éléments `<span>` à l'aide du message correspondant, nous devons enregistrer les messages initiaux dans deux variables. Ensuite, nous pouvons retirer les parenthèses à l'aide d'une *expression régulière* :

```
$(document).ready(function() {
  var requiredFlag = ' * ';
  var conditionalFlag = ' ** ';

  var requiredKey = $('input.required:first').next('span').text();
  var conditionalKey = $('input.conditional:first').next('span').text();

  requiredKey = requiredFlag + requiredKey.replace(/^\((.+)\)$/,'$1');
  conditionalKey = conditionalFlag + conditionalKey.replace(/^\((.+)\)$/,'$1');

  // ... suite du code.
});
```

Les deux premières lignes supplémentaires déclarent `requiredKey` et `conditionalKey`, les variables dans lesquelles sont enregistrés les textes des messages. Les deux lignes suivantes modifient le texte contenu dans ces variables, en concaténant le symbole de référence et le texte associé, sans les parenthèses. L'expression régulière et la méthode `.replace()` méritent sans doute quelques explications.

Expression régulière

L'expression régulière est placée entre les deux barres obliques et prend la forme suivante : `/^\((.+)\)$/`. Le premier caractère, `^`, indique que ce qui suit doit apparaître au début de la chaîne. Il est suivi de deux caractères, `\(`, qui recherchent une parenthèse ouvrante. La barre oblique inverse applique l'*échappement* au caractère suivant pour indiquer au parseur de l'expression régulière qu'il doit le traiter littéralement. Cet échappement est nécessaire car les parenthèses font partie des caractères ayant une signification particulière dans les expressions régulières. Les quatre caractères suivants,

`(.+)`, recherchent un ou plusieurs (la signification du `+`) caractères quelconques à la suite (représentés par le `.`) et les placent dans un groupe de capture grâce aux parenthèses. Les trois derniers caractères, `\)$`, recherchent une parenthèse fermante à la fin de la chaîne. En résumé, l'expression régulière correspond à une chaîne qui commence par une parenthèse ouvrante, suivie d'un groupe de caractères et terminée par une parenthèse fermante.

La méthode `.replace()` recherche, dans le contexte indiqué, une chaîne de caractères représentée par une expression régulière et la remplace par une autre chaîne. Voici sa syntaxe :

```
'contexte'.replace(/expression-régulière/, 'remplacement')
```

Dans nos deux méthodes `.replace()`, les chaînes sont les variables `requiredKey` et `conditionalKey`. La signification des expressions régulières a été donnée précédemment. Une virgule sépare l'expression régulière de la chaîne de remplacement, qui, dans notre cas, est `'$1'`. Le paramètre `$1` représente le premier groupe de capture de l'expression régulière. Puisque notre expression régulière possède un groupe de un ou de plusieurs caractères entouré de parenthèses, la chaîne de remplacement correspondra au contenu de ce groupe, sans les parenthèses.

Insérer la légende des messages

Nous disposons à présent des messages sans les parenthèses et pouvons donc les insérer au-dessus du formulaire avec les indicateurs correspondants :

```
$(document).ready(function() {
  var requiredFlag = ' * ';
  var conditionalFlag = ' ** ';

  var requiredKey = $('input.required:first').next('span').text();
  var conditionalKey = $('input.conditional:first').next('span').text();

  requiredKey = requiredFlag + requiredKey.replace(/^\((.+)\)$/,'$1');
  conditionalKey = conditionalFlag + conditionalKey.replace(/^\((.+)\)$/,'$1');

  $('<p></p>')
    .addClass('field-keys')
    .append(requiredKey + '<br />')
    .append(conditionalKey)
    .insertBefore('#contact');
});
```

Les cinq nouvelles lignes doivent désormais vous sembler relativement familières. Voici leur fonction :

1. Créer un nouvel élément de paragraphe.
2. Affecter la classe `field-keys` au paragraphe.

3. Ajouter le contenu de `requiredKey` et un passage à la ligne au paragraphe.
4. Ajouter le contenu de `conditionalKey` au paragraphe.
5. Insérer le paragraphe et tout ce que nous avons ajouté avant le formulaire de contact.

Lorsqu'on utilise `.append()` avec une chaîne HTML, comme c'est le cas ici, il faut vérifier que l'échappement est appliqué aux caractères HTML spéciaux. Dans notre exemple, la méthode `.text()` utilisée lors de la déclaration des variables a procédé à cette vérification pour nous.

En définissant quelques styles pour la classe `.field-keys`, nous obtenons le résultat illustré à la Figure 8.4.

Figure 8.4

Déplacement des messages avant le formulaire.

Notre intervention jQuery sur le premier jeu de champs est quasiment terminée.

Affichage conditionnel des champs

Améliorons à présent le groupe de champs qui demande à l'internaute par quels moyens il souhaite être contacté. Puisque les champs de saisie ne doivent être remplis que si les cases correspondantes sont cochées, nous pouvons les masquer, ainsi que les doubles astérisques, lors du chargement initial du document :

```
$(document).ready(function() {
  $('input.conditional').next('span').andSelf().hide();
});
```

La Figure 8.5 montre que le jeu de champs est à présent plus organisé.

Figure 8.5

Les champs de saisie des méthodes de contact sont initialement masqués.

Pour que ces champs et ces astérisques apparaissent au moment opportun, nous associons la méthode `.click()` à chaque case à cocher. Cette opération se fait dans le contexte de chaque champ de saisie conditionnel afin que nous puissions fixer la valeur de deux variables qui seront réutilisées par la suite :

```
$(document).ready(function() {
  $('input.conditional').next('span').andSelf().hide()
  .end().end()
  .each(function() {
    var $thisInput = $(this);
    var $thisFlag = $thisInput.next('span');
    $thisInput.prev('label').find(':checkbox')
    .click(function() {
      // suite du code...
    });
  });
});
```

Nous appelons à nouveau la méthode `.end()`, cette fois-ci pour que la méthode `.each()` s'applique au sélecteur `$('input.conditional')` d'origine.

Nous disposons à présent d'une variable pour le champ de saisie et la référence de message en cours. Lorsque l'utilisateur clique sur la case à cocher, nous vérifions si elle est cochée. Dans l'affirmative, nous affichons le champ de saisie, la référence au message et ajoutons la classe `req-label` à l'élément `<label>` parent :

```
$(document).ready(function() {
  $('input.conditional').next('span').andSelf().hide()
  .end().end()
  .each(function() {
    var $thisInput = $(this);
    var $thisFlag = $thisInput.next('span');
    $thisInput.prev('label').find(':checkbox')
    .click(function() {
      if (this.checked) {
        $thisInput.show();
```

```
          $thisFlag.show();
          $(this).parent('label').addClass('req-label');
        }
      });
    });
});
```

Dans notre exemple, nous vérifions si une case est cochée en utilisant `this.checked`, car nous avons un accès direct au nœud du DOM *via* le mot clé `this`. Lorsque le nœud du DOM est inaccessible, nous pouvons utiliser `$('selector').is(':checked')`, car `.is()` retourne une valeur booléenne (`true` ou `false`).

Il reste deux points à résoudre :

1. Faire en sorte que les cases soient décochées lors du chargement initial de la page, car certains navigateurs conservent l'état des éléments du formulaire après actualisation de la page.
2. Ajouter une clause `else` qui masque les éléments conditionnels et retire la classe `req-label` lorsque la case est décochée.

```
$(document).ready(function() {
  $('input.conditional').next('span').andSelf().hide()
  .end().end()
  .each(function() {
    var $thisInput = $(this);
    var $thisFlag = $thisInput.next('span');
    $thisInput.prev('label').find(':checkbox')
    .attr('checked', false)
    .click(function() {
      if (this.checked) {
        $thisInput.show();
        $thisFlag.show();
        $(this).parent('label').addClass('req-label');
      } else {
        $thisInput.hide();
        $thisFlag.hide();
        $(this).parent('label').removeClass('req-label');
      }
    });
  });
});
```

Nous concluons ainsi cette partie de la mise en forme du formulaire. Nous allons passer à la validation du formulaire côté client.

Valider le formulaire

Avant d'utiliser jQuery pour ajouter la validation d'un formulaire, il ne faut pas oublier une règle importante : la *validation côté client* ne remplace pas la validation côté serveur. Une fois encore, nous ne pouvons pas supposer que les utilisateurs aient activé JavaScript. S'il faut absolument vérifier que certains champs ont été saisis ou que cette saisie doit se faire dans un format particulier, JavaScript seul ne peut pas garantir le

résultat. En effet, certains internautes préfèrent désactiver JavaScript, certains périphériques ne le prennent tout simplement pas en charge et quelques utilisateurs peuvent envoyer intentionnellement des données malveillantes en contournant les protections posées par JavaScript.

Dans ce cas, pourquoi vouloir mettre en œuvre une validation avec jQuery ? La validation d'un formulaire sur le client présente un avantage par rapport à la validation côté serveur : un *retour immédiat*. Le code côté serveur, qu'il s'agisse d'ASP, de PHP ou de tout autre acronyme, impose un rechargement de la page (à moins que l'accès ne soit asynchrone, ce qui, de toute manière, exige JavaScript). Avec jQuery, nous pouvons tirer parti de la réponse rapide du code côté client en effectuant une validation sur chaque champ obligatoire lorsqu'il perd le focus (événement `blur`) ou lors de l'appui sur une touche (événement `keyup`).

Champs obligatoires

Pour notre formulaire de contact, nous allons tester la présence de la classe `required` sur un champ de saisie lorsque l'internaute utilise la touche Tab ou clique en dehors du champ. Avant d'examiner le code, nous devons revenir rapidement aux champs de saisie conditionnels. Pour simplifier notre procédure de validation, nous pouvons ajouter la classe `required` à l'élément `<input>` lorsqu'il est affiché, et la retirer lorsqu'il est masqué. Voici la nouvelle version de cette partie du code :

```
$thisInput.prev('label').find(':checkbox')
  .attr('checked', false)
  .click(function() {
    if (this.checked) {
      $thisInput.show().addClass('required');
      $thisFlag.show();
      $(this).parent('label').addClass('req-label');
    } else {
      $thisInput.hide().removeClass('required');
      $thisFlag.hide();
      $(this).parent('label').removeClass('req-label');
    }
  });
```

Toutes les classes `required` étant en place, nous sommes prêts à signaler à l'utilisateur que l'un des champs requis est vide. Un message est affiché après l'indicateur de champ obligatoire et un style est appliqué à l'élément `<li>` du champ au travers de la classe `warning` de manière à avertir l'utilisateur :

```
$(document).ready(function() {
  $('form :input').blur(function() {
    if ($(this).hasClass('required')) {
      var $listItem = $(this).parents('li:first');
      if (this.value == '') {
        var errorMessage = 'Ce champ doit être rempli';
        $('<span></span>')
          .addClass('error-message')
```

```
            .text(errorMessage)
            .appendTo($listItem);
          $listItem.addClass('warning');
        }
      }
    });
});
```

Le code comprend deux instructions `if` pour chaque événement `blur` sur un champ du formulaire. La première vérifie l'existence de la classe `required`, tandis que la seconde teste si la valeur du champ est une chaîne vide. Lorsque les deux conditions sont satisfaites, nous construisons un message d'erreur, le plaçons dans `<span class="error-message">` et l'ajoutons au `<li>` parent.

Lorsque le champ vide est l'un des champs conditionnels (il n'est obligatoire que si la case correspondante est cochée), nous souhaitons modifier légèrement le message. Nous allons concaténer un message de précision au message d'erreur standard. Pour cela, nous imbriquons une instruction `if` supplémentaire qui vérifie la présence de la classe `conditional` lorsque les deux conditions précédentes sont satisfaites :

```
$(document).ready(function() {
  $('form :input').blur(function() {
    if ($(this).hasClass('required')) {
      var $listItem = $(this).parents('li:first');
      if (this.value == '') {
        var errorMessage = 'Ce champ doit être rempli';
        if ($(this).hasClass('conditional')) {
          errorMessage += ', lorsque la case correspondante est cochée';
        }
        $('<span></span>')
          .addClass('error-message')
          .text(errorMessage)
          .appendTo($listItem);
        $listItem.addClass('warning');
      }
    }
  });
});
```

Notre code fonctionne parfaitement la première fois que l'utilisateur quitte un champ vide. Cependant, deux problèmes apparaissent lorsqu'il entre et quitte à nouveau le champ (voir Figure 8.6).

Si le champ reste vide, le message d'erreur est répété autant de fois que l'utilisateur quitte le champ. Si du texte est saisi dans le champ, la classe `warning` n'est pas retirée. Bien évidemment, nous voulons afficher un seul message par champ et le retirer lorsque l'utilisateur corrige l'erreur. Nous résolvons les deux problèmes en retirant la classe `warning` de l'élément `<li>` parent du champ en cours et tout `<span class="error-message">` du même `<li>` lors du déclenchement de l'événement `blur` sur le champ, avant d'exécuter la validation :

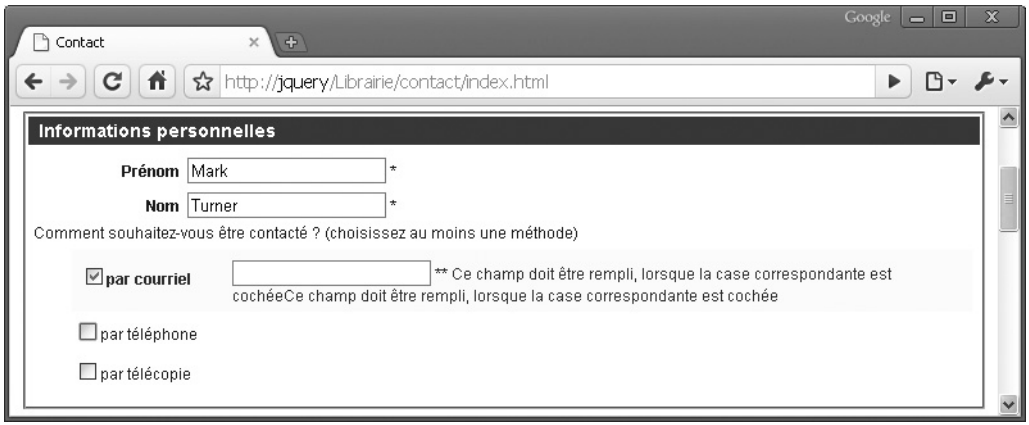


Figure 8.6

Problème de répétition du message d'erreur.

```
$(document).ready(function() {
  $('form :input').blur(function() {
    $(this).parents('li:first').removeClass('warning')
    .find('span.error-message').remove();
    if ($(this).hasClass('required')) {
      var $listItem = $(this).parents('li:first');
      if (this.value == '') {
        var errorMessage = 'Ce champ doit être rempli';
        if ($(this).hasClass('conditional')) {
          errorMessage += ', lorsque la case correspondante est cochée';
        }
        $('<span></span>')
          .addClass('error-message')
          .text(errorMessage)
          .appendTo($listItem);
        $listItem.addClass('warning');
      }
    }
  });
});
```

Notre script de validation des champs obligatoires et des champs obligatoires conditionnels est enfin opérationnel. En entrant et en quittant plusieurs fois des champs obligatoires, les messages d'erreur s'affichent à présent correctement (voir Figure 8.7).

Cependant, nous devons encore retirer la classe `warning` de l'élément `<li>` et ses éléments `<span class="error-message">` lorsque l'utilisateur décoche une case. Pour cela, nous revenons au code précédent de traitement des cases à cocher et ajoutons le déclenchement de l'événement `blur` sur le champ de texte correspondant lorsque la case est décochée :

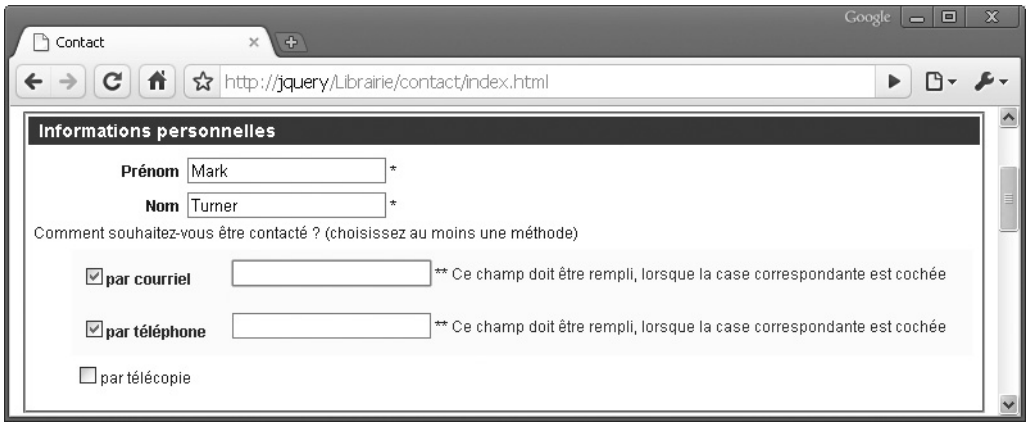


Figure 8.7

L'affichage des messages d'erreur se fait correctement.

```
if (this.checked) {
  $thisInput.show().addClass('required');
  $thisFlag.show();
  $(this).parent('label').addClass('req-label');
} else {
  $thisInput.hide().removeClass('required').blur();
  $thisFlag.hide();
  $(this).parent('label').removeClass('req-label');
}
```

Désormais, lorsqu'une case est décochée, les styles d'avertissement et les messages d'erreur disparaissent.

Formats obligatoires

Nous devons encore mettre en œuvre un autre type de validation dans notre formulaire de contact. Il concerne les *formats de saisie*. Il est parfois utile d'afficher un avertissement lorsque le texte indiqué dans un champ n'est pas correct (au lieu qu'il soit simplement vide). Les champs de saisie d'une adresse électronique, d'un numéro de téléphone ou d'un numéro de carte bancaire font de parfaits candidats à ce type d'avertissement. À titre d'exemple, nous allons mettre en place une expression régulière relativement simple pour tester l'adresse électronique saisie. Avant d'examiner en détail cette expression régulière, voici l'intégralité du code de validation :

```
$(document).ready(function() {
  // ... suite du code ...
  if (this.id == 'email') {
    var $listItem = $(this).parents('li:first');
    if ($(this).is(':hidden')) {
      this.value = '';
    }
```

```
      if (this.value != '' && !/.+@.+\.[a-zA-Z]{2,4}$/.test(this.value)) {
        var errorMessage = "Veuillez utiliser un format d'adresse valide"
          + ' (par exemple paul@example.com)';
        $('<span></span>')
          .addClass('error-message')
          .text(errorMessage)
          .appendTo($listItem);
        $listItem.addClass('warning');
      }
    }
    // ... suite du code ...
});
```

Ce code réalise les opérations suivantes :

- Tester si l'identifiant du champ correspond à celui de l'adresse électronique. Dans l'affirmative :
  - Affecter l'élément de liste parent à une variable.
  - Vérifier si le champ d'adresse est masqué. Dans l'affirmative (ce qui se produit lorsque la case correspondante est décochée), fixer sa valeur à une chaîne vide. Ainsi, le code de suppression de la classe d'avertissement et du message d'erreur fonctionne correctement pour ce champ.
  - Vérifier que le champ ne contient pas une chaîne vide et que sa valeur ne correspond pas à l'expression régulière. En cas de réussite des deux tests :
    - Créer un message d'erreur.
    - Insérer le message dans un élément `<span class="error-message">`.
    - Ajouter le `<span class="error-message">` et son contenu à l'élément de liste parent.
    - Appliquer la classe `warning` à l'élément de liste parent.

Examinons à présent l'expression régulière seule :

```
!/.+@.+\.[a-zA-Z]{2,4}$/.test(this.value)
```

Bien qu'elle ressemble à celle créée précédemment dans ce chapitre, elle utilise la méthode `.test()` à la place de `.replace()`. En effet, nous souhaitons simplement qu'elle retourne `true` ou `false`, sans procéder à un remplacement. L'expression régulière est placée entre les deux barres obliques. Elle est ensuite comparée à la chaîne qui se trouve entre les parenthèses de `.test()` ; dans ce cas, la valeur du champ de saisie est l'adresse électronique.

Avec cette expression régulière, nous recherchons un groupe de un ou de plusieurs caractères (`.+`), suivi d'un symbole `@`, suivi d'un autre groupe de un ou de plusieurs caractères. Jusque-là, une chaîne comme `lucie@example` passe le test avec succès,

comme des millions d'autres variantes, même s'il ne s'agit pas d'une adresse électronique valide.

Nous pouvons rendre le test plus précis, en recherchant un caractère `.`, suivi de deux à quatre lettres (de `a` à `z`) à la fin de la chaîne. C'est précisément ce que fait la partie restante de l'expression régulière. Elle commence par rechercher un caractère entre `a` et `z` ou entre `A` et `Z` (`[a-zA-Z]`). Elle précise ensuite qu'une lettre dans cette plage peut apparaître entre deux et quatre fois uniquement (`{2,4}`). Enfin, elle insiste pour que ces deux à quatre lettres apparaissent à la fin de la chaîne (`$`). À présent, une chaîne comme `lucie@example.com` retournera `true`, tandis que `lucie@example`, `lucie@example.2fn`, `lucie@example.example` ou `lucia-example.com` retournera `false`.

Cependant, nous voulons que `true` soit retourné (et le code de traitement correspondant, exécuté) uniquement si l'adresse saisie n'est pas dans le bon format. C'est pourquoi nous faisons précéder l'expression de test par un point d'exclamation (*opérateur non*) :

```
!/.+@.+\.[a-zA-Z]{2,4}$/.test(this.value)
```

Vérification finale

Le code de validation du formulaire de contact est presque terminé. Nous pouvons vérifier une nouvelle fois les champs du formulaire lorsque l'utilisateur tente de le soumettre, mais la vérification est cette fois-ci globale. À partir du gestionnaire d'événements `.submit()` associé au formulaire, non au bouton ENVOYER, nous déclenchons l'événement `blur` sur tous les champs obligatoires :

```
$(document).ready(function() {
  $('form').submit(function() {
    $('#submit-message').remove();
    $(':input.required').trigger('blur');
  });
});
```

Vous remarquerez que nous avons ajouté une ligne pour retirer un élément `<div id="submit-message">` qui n'existe pas encore. Nous l'ajouterons à l'étape suivante. Nous le supprimons ici car nous savons déjà que nous devrons le faire, en raison des problèmes de création de multiples messages d'erreur rencontrés précédemment.

Après avoir déclenché l'événement `blur`, nous connaissons le nombre total de classes `warning` présentes dans le formulaire. S'il est positif, nous créons un nouvel élément `<div id="submit-message">` et l'insérons avant le bouton ENVOYER, là où l'utilisateur le remarquera. Nous arrêtons également la soumission du formulaire :

```
$(document).ready(function() {
  $('form').submit(function() {
    $('#submit-message').remove();
    $(':input.required').trigger('blur');
```

```
    var numWarnings = $('.warning', this).length;
    if (numWarnings) {
      $('<div></div>')
      .attr({
        'id': 'submit-message',
        'class': 'warning'
      })
      .append('Veuillez corriger les erreurs avec les ' +
          numWarnings + ' champs')
      .insertBefore('#send');
      return false;
    }
  });
});
```

Outre la demande générique de correction des erreurs, le message indique le nombre de champs concernés (voir Figure 8.8).

Figure 8.8

L'internaute doit corriger ses erreurs de saisie dans trois champs.

Nous pouvons néanmoins faire mieux. Au lieu d'indiquer simplement le nombre d'erreurs, nous pouvons afficher la liste des champs concernés :

```
$(document).ready(function() {
  $('form').submit(function() {
    $('#submit-message').remove();
    $(':input.required').trigger('blur');
    var numWarnings = $('.warning', this).length;
    if (numWarnings) {
      var list = [];
      $('.warning label').each(function() {
        list.push($(this).text());
      });
      $('<div></div>')
      .attr({
        'id': 'submit-message',
        'class': 'warning'
      })
      .append('Veuillez corriger les erreurs dans les ' +
          numWarnings + ' champs suivants :<br />')
      .append('&bull; ' + list.join('<br />&bull; '))
      .insertBefore('#send');
      return false;
    };
  });
});
```

La première modification du code se trouve dans la variable `list`, qui contient initialement un tableau vide. Ensuite, nous prenons chaque libellé qui est un descendant d'un élément possédant la classe `warning` et plaçons le texte qu'il contient dans le tableau `list` (avec la fonction JavaScript native `push`). Le texte de chaque libellé est un élément distinct dans le tableau `list`.

Nous modifions légèrement le message du `<div id="submit-message">` et y ajoutons le tableau `list`. En utilisant la fonction JavaScript native `join()` pour convertir le tableau en chaîne de caractères, nous insérons chaque élément du tableau, avec un passage à la ligne et une puce (voir Figure 8.9).

Figure 8.9

La liste des champs à corriger.

Nous l'admettons, le balisage HTML de la liste des champs a un objectif de présentation, non de sémantique. Toutefois, pour une liste éphémère, qui est générée par le code JavaScript lors de la dernière étape et qui sera détruite dès que possible, nous excuserons ce code développé rapidement, dans un souci de concision et de facilité.

Manipuler des cases à cocher

Notre formulaire de contact comprend également une partie DIVERS constituée d'une liste de cases à cocher (voir Figure 8.10).

Pour compléter nos améliorations du formulaire de contact, nous allons aider l'utilisateur à gérer cette liste. Une suite de dix cases risque de le décourager, en particulier s'il veut en cocher la majorité, voire toutes. Dans une telle situation, une option qui permet de cocher ou de décocher toutes les cases sera utile. Nous allons donc l'ajouter.

Nous créons tout d'abord un nouvel élément `<li>`, y plaçons un `<label>`, dans lequel nous insérons `<input type="checkbox" id="discover-all">` et du texte. Nous ajoutons le tout à l'élément `<ul>` dans `<li class="discover">` :

```
$(document).ready(function() {
  $('<li></li>')
  .html('<label><input type="checkbox" id="discover-all" />' +
      ' <em>cocher tout</em></label>')
  .prependTo('li.discover > ul');
});
```

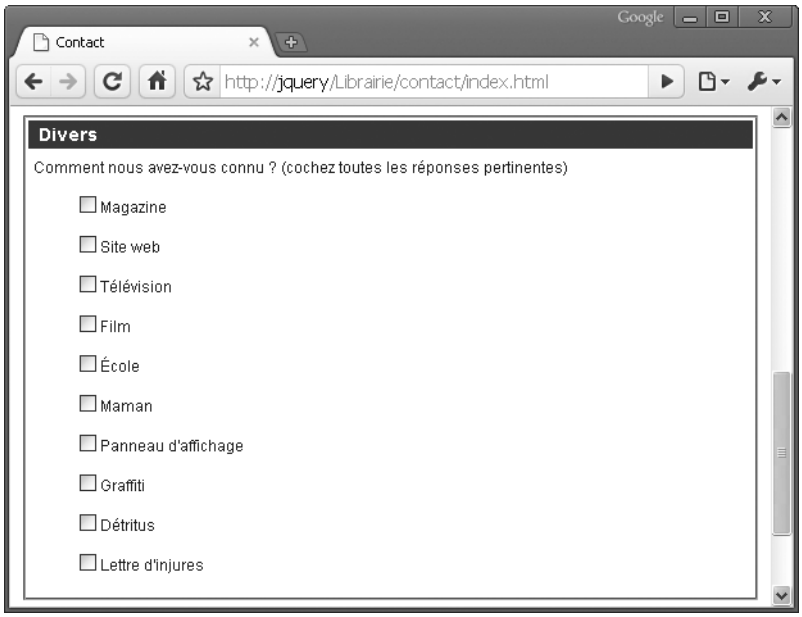


Figure 8.10

La section des cases à cocher.

Nous disposons à présent d'une nouvelle case à cocher avec le libellé COCHER TOUT. Cependant, pour qu'elle fasse quelque chose, nous devons lui associer une méthode `.click()` :

```
$(document).ready(function() {
  $('<li></li>')
  .html('<label><input type="checkbox" id="discover-all" />' +
      ' <em>cocher tout</em></label>')
  .prependTo('li.discover > ul');
  $('#discover-all').click(function() {
    var $checkboxes = $(this).parents('ul:first').find(':checkbox');
    if (this.checked) {
      $checkboxes.attr('checked', true);
    } else {
      $checkboxes.attr('checked', '');
    }
  });
});
```

Dans ce gestionnaire d'événements, nous fixons tout d'abord la valeur de la variable `$checkboxes`, qui est constituée d'un objet jQuery contenant chaque case à cocher de la liste. Grâce à cette affectation, nous devons simplement cocher les cases si COCHER TOUT est cochée et les décocher si COCHER TOUT est décochée.

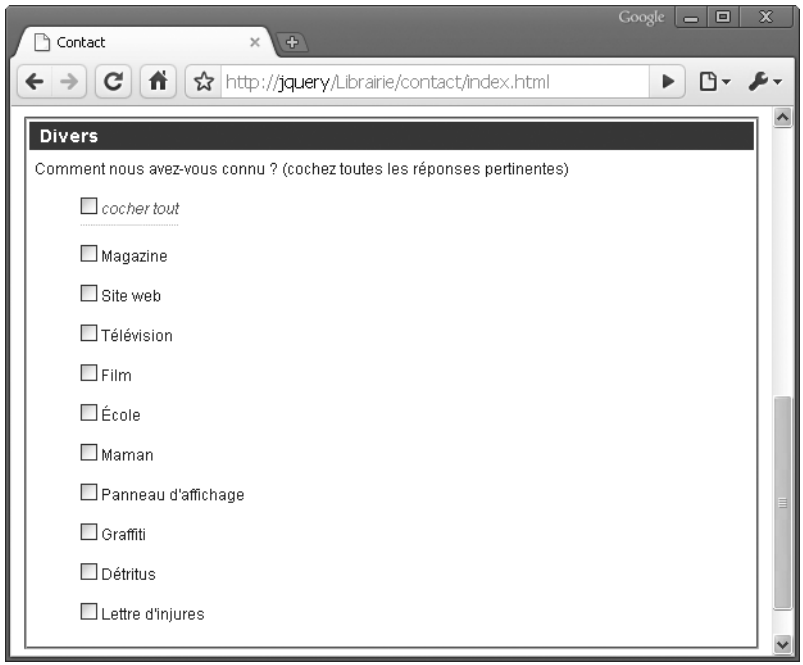
La touche finale consiste à ajouter une classe `checkall` au libellé de la case COCHER TOUT et à modifier son texte à DÉCOCHER TOUT après qu'elle a été cochée par l'utilisateur :

```
$(document).ready(function() {
  $('<li></li>')
  .html('<label><input type="checkbox" id="discover-all" />' +
      ' <em>cocher tout</em></label>')
  .prependTo('li.discover > ul');
  $('#discover-all').click(function() {
    var $checkboxes = $(this) .parents('ul:first').find(':checkbox');
    if (this.checked) {
      $(this).next().text(' décocher tout');
      $checkboxes.attr('checked', true);
    } else {
      $(this).next().text(' cocher tout');
      $checkboxes.attr('checked', '');
    };
  })
  .parent('label').addClass('checkall');
});
```

La Figure 8.11 présente le groupe de cases à cocher, avec la case COCHER TOUT.

Figure 8.11

Une nouvelle case permet de cocher toutes les autres cases en un clic.



Après avoir coché la case COCHER TOUT, le groupe se présente tel qu'illustré à la Figure 8.12.

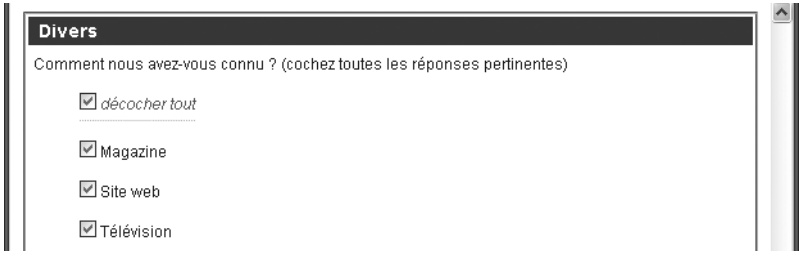


Figure 8.12

La nouvelle case permet également de décocher toutes les autres cases.

Version finale du code

Voici la version finale du code pour le formulaire de contact :

```
$(document).ready(function() {

  // Améliorer le style des éléments de formulaire.

  $('legend').each(function(index) {
    $(this).replaceWith('<h3>' + $(this).text() + '</h3>');
  });

  var requiredFlag = ' * ';
  var conditionalFlag = ' ** ';
  var requiredKey = $('input.required:first').next('span').text();
  var conditionalKey = $('input.conditional:first').next('span').text();

  requiredKey = requiredFlag + requiredKey.replace(/^\((.+)\)$/,'$1');
  conditionalKey = conditionalFlag + conditionalKey.replace(/^\((.+)\)$/,'$1');

  $('<p></p>')
    .addClass('field-keys')
    .append(requiredKey + '<br />')
    .append(conditionalKey)
    .insertBefore('#contact');

  $('form :input')
    .filter('.required')
      .next('span').text(requiredFlag).end()
      .prev('label').addClass('req-label').end()
    .end()
    .filter('.conditional')
      .next('span').text(conditionalFlag);

  // Cases à cocher : champs de saisie conditionnels.

  $('input.conditional').next('span').andSelf().hide()
  .end().end()
  .each(function() {
    var $thisInput = $(this);
    var $thisFlag = $thisInput.next('span');
    $thisInput.prev('label').find(':checkbox')
    .attr('checked', false)
    .click(function() {
```

```
    if (this.checked) {
      $thisInput.show().addClass('required');
      $thisFlag.show();
      $(this).parent('label').addClass('req-label');
    } else {
      $thisInput.hide().removeClass('required').blur();
      $thisFlag.hide();
      $(this).parent('label').removeClass('req-label');
    }
  });
});

// Valider les champs sur l'événement.

$('form :input').blur(function() {
  $(this).parents('li:first').removeClass('warning')
  .find('span.error-message').remove();

  if ($(this).hasClass('required')) {
    var $listItem = $(this).parents('li:first');
    if (this.value == '') {
      var errorMessage = 'Ce champ doit être rempli';
      if ($(this).is('.conditional')) {
        errorMessage += ', lorsque la case correspondante est cochée';
      }
      $('<span></span>')
        .addClass('error-message')
        .text(errorMessage)
        .appendTo($listItem);
      $listItem.addClass('warning');
    }
  }

  if (this.id == 'email') {
    var $listItem = $(this).parents('li:first');
    if ($(this).is(':hidden')) {
      this.value = '';
    }
    if (this.value != '' && !/.+@.+\.[a-zA-Z]{2,4}$/.test(this.value)) {
      var errorMessage = "Veuillez utiliser un format d'adresse valide"
          + ' (par exemple paul@example.com)';
      $('<span></span>')
        .addClass('error-message')
        .text(errorMessage)
        .appendTo($listItem);
      $listItem.addClass('warning');
    }
  }
});

// Valider le formulaire à la soumission.

$('form').submit(function() {
  $('#submit-message').remove();
  $(':input.required').trigger('blur');
  var numWarnings = $('.warning', this).length;
```

```
    if (numWarnings) {
      var fieldList = [];
      $('.warning label').each(function() {
        fieldList.push($(this).text());
      });
      $('<div></div>')
      .attr({
        'id': 'submit-message',
        'class': 'warning'
      })
      .append('Veuillez corriger les erreurs dans les ' +
          numWarnings + ' champs suivants :<br />')
      .append('&bull; ' + fieldList.join('<br />&bull; '))
      .insertBefore('#send');
      return false;
    };
  });

  // Cases à cocher.
  $('form :checkbox').removeAttr('checked');

  // Cases à cocher avec (dé)cocher tout.
  $('<li></li>')
  .html('<label><input type="checkbox" id="discover-all" />' +
      ' <em>cocher tout</em></label>')
  .prependTo('li.discover > ul');
  $('#discover-all').click(function() {
    var $checkboxes = $(this) .parents('ul:first').find(':checkbox');
    if (this.checked) {
      $(this).next().text(' décocher tout');
      $checkboxes.attr('checked', true);
    } else {
      $(this).next().text(' cocher tout');
      $checkboxes.attr('checked', '');
    };
  })
  .parent('label').addClass('checkall');
});
```

Nous avons apporté de nombreuses améliorations au formulaire de contact, mais nous pouvons aller beaucoup plus loin, notamment en ce qui concerne la validation. Vous trouverez un plugin de validation flexible à l'adresse **http://plugins.jquery.com/project/validate/**.

8.2 Formulaires compacts

Certains formulaires sont beaucoup plus simples que des formulaires de contact. En réalité, de nombreux sites ajoutent souvent à chaque page un formulaire comprenant un seul champ : une *fonction de recherche* sur le site. Les habituelles informations présentées dans un formulaire, comme les libellés des champs, les boutons de soumission et le texte, sont superflues dans une portion de la page aussi petite. Nous pouvons utiliser jQuery pour réduire la taille du formulaire tout en conservant ses fonctionnalités et

même améliorer son comportement pour qu'il soit plus facile à utiliser que son équivalent pleine page.

Texte substituable pour les champs

L'élément `<label>` d'un champ de formulaire est un composant indispensable pour l'accessibilité des sites web. Chaque champ doit être libellé afin que les lecteurs d'écran et les autres dispositifs d'assistance puissent identifier le champ utilisé et son rôle. Par ailleurs, dans le code source HTML, le libellé permet de décrire le champ :

```
<form id="search" action="search/index.php" method="get">
  <label for="search-text">rechercher sur le site</label>
  <input type="text" name="search-text" id="search-text" />
</form>
```

Sans style particulier, le libellé est placé avant le champ (voir Figure 8.13).

Figure 8.13

Le libellé d'un champ.

Même si la place occupée par le libellé est réduite, cette seule ligne de texte peut être de trop avec la mise en page de certains sites. Nous pourrions masquer le texte avec un style CSS, mais l'utilisateur ne sera plus informé du rôle du champ. À la place, nous allons ajouter une classe au formulaire de recherche et utiliser CSS pour positionner le libellé au-dessus du champ, uniquement lorsque JavaScript est activé :

```
$(document).ready(function() {
  var $search = $('#search').addClass('overlabel');
});
```

Grâce à cette seule ligne, nous ajoutons une classe au formulaire de recherche et enregistrons le sélecteur dans une variable pour pouvoir y faire référence ultérieurement. La feuille de style définit des règles pour la classe `overlabel` :

```
.overlabel {
  position: relative;
}

.overlabel label {
  position: absolute;
  top: 6px;
  left: 3px;
  color: #999;
  cursor: text;
}
```

La classe ajoutée positionne correctement le libellé et affiche le texte en gris afin d'indiquer qu'il pourra être remplacé (voir Figure 8.14).

Figure 8.14

Le libellé est superposé au champ.

L'effet obtenu est intéressant, mais il présente deux problèmes (voir Figure 8.15) :

1. Le libellé masque le texte que l'utilisateur saisit dans le champ.
2. Pour entrer dans le champ de saisie, il faut utiliser la touche Tab. En effet, le libellé recouvre le champ, ce qui empêche l'utilisateur de cliquer dans la zone de saisie.

Figure 8.15

Le libellé masque la saisie.

Pour résoudre le premier problème, nous devons masquer le texte du libellé lorsque le champ reçoit le focus, et l'afficher à nouveau lorsqu'il le perd et que l'utilisateur n'a rien saisi dans le champ.

INFO

Pour de plus amples informations concernant le focus du clavier, consultez le Chapitre 3.

Le code qui masque le texte du libellé est relativement simple :

```
$(document).ready(function() {
  var $search = $('#search').addClass('overlabel');
  var $searchInput = $search.find('input');
  var $searchLabel = $search.find('label');

  $searchInput
  .focus(function() {
    $searchLabel.hide();
  })
  .blur(function() {
    if (this.value == '') {
```

```
      $searchLabel.show();
    }
  });
});
```

Le libellé est désormais masqué lorsque l'utilisateur saisit du texte dans le champ (voir Figure 8.16).

Figure 8.16

Lors de la saisie, le libellé disparaît.

Le second problème est à présent relativement facile à résoudre. Nous pouvons masquer le texte du libellé et donner à l'utilisateur accès au champ de saisie en faisant en sorte qu'un clic sur le libellé déclenche l'événement `focus` sur le champ :

```
$(document).ready(function() {
  var $search = $('#search').addClass('overlabel');
  var $searchInput = $search.find('input');
  var $searchLabel = $search.find('label');

  $searchInput
  .focus(function() {
    $searchLabel.hide();
  })
  .blur(function() {
    if (this.value == '') {
      $searchLabel.show();
    }
  });

  $searchLabel.click(function() {
    $searchInput.trigger('focus');
  });
});
```

Enfin, nous devons traiter le cas où du texte est conservé dans le champ de saisie après l'actualisation de la page – il s'agit d'un cas semblable à celui rencontré précédemment avec les champs de saisie conditionnels pour la validation du formulaire. Lorsque le champ de saisie contient une valeur, le libellé est masqué :

```
$(document).ready(function() {
  var $search = $('#search').addClass('overlabel');
  var $searchInput = $search.find('input');
  var $searchLabel = $search.find('label');

  if ($searchInput.val()) {
    $searchLabel.hide();
  }
```

```
    $searchInput
    .focus(function() {
      $searchLabel.hide();
    })
    .blur(function() {
      if (this.value == '') {
        $searchLabel.show();
      }
    });

    $searchLabel.click(function() {
      $searchInput.trigger('focus');
    });
  });
```

Nous avons choisi d'utiliser un libellé au lieu d'insérer une valeur par défaut directement dans le champ de saisie. En effet, cette technique a pour avantage de pouvoir être adaptée à n'importe quel champ de texte, sans avoir à s'inquiéter d'un éventuel conflit avec un script de validation.

Auto-complétion AJAX

Pour développer le champ de recherche, nous allons lui offrir une *auto-complétion* de son contenu. Cette fonctionnalité permettra aux utilisateurs de commencer la saisie du terme recherché et de se voir présenter tous les termes possibles qui commencent par les caractères déjà saisis. Puisque la liste des termes peut être extraite d'une base de données associée au site, l'utilisateur sait que la recherche du terme proposé produira un résultat. Par ailleurs, si la base de données propose les termes par ordre de popularité ou par nombre de résultats, l'utilisateur sera guidé dans ses recherches.

L'auto-complétion est un sujet très complexe, avec des subtilités introduites par les différentes actions de l'utilisateur. Nous allons mettre en œuvre un exemple opérationnel, mais nous n'avons pas la place d'explorer tous les concepts élaborés, comme limiter le débit des requêtes ou la complétion multiterme. Le widget d'auto-complétion disponible dans la bibliothèque jQuery UI peut être utilisé pour les implémentations réelles simples et peut servir de point de départ pour des mises en œuvre plus complexes. Vous le trouverez sur le site **http://ui.jquery.com/**.

Pour écrire une procédure d'auto-complétion, l'idée est de réagir aux appuis sur les touches du clavier et d'envoyer une requête AJAX au serveur en passant le contenu du champ. La réponse contiendra une liste des complétions possibles pour le champ. Le script présente ensuite cette liste dans une fenêtre affichée sous le champ.

Côté serveur

Nous avons besoin d'un code côté serveur pour traiter les requêtes. Dans le cas d'une mise en œuvre réelle, nous utiliserions une base de données pour produire la liste des complétions possibles. Dans notre exemple, un simple script PHP fournit une réponse figée :

```
<?php
  if (strlen($_REQUEST['search-text']) < 1) {
    print '[]';
    exit;
  }
  $terms = array(
    'accès',
    'action',
    // suite de la liste...
    'xaml',
    'xoops',
  );
  $possibilities = array();
  foreach ($terms as $term) {
    if (strpos($term, strtolower($_REQUEST['search-text'])) === 0) {
      $possibilities[] = "'". str_replace("'", "\\'", $term)."'";
    }
  }
  print ('['. implode(', ', $possibilities) .']');
```

La page compare la chaîne fournie avec le début de chaque terme et construit un *tableau JSON* avec les correspondances. Les manipulations effectuées sur la chaîne, avec `str_replace()` et `implode()`, s'assurent que la sortie du script est une structure JSON correctement formatée afin d'éviter les erreurs JavaScript pendant l'analyse.

Côté navigateur

Nous pouvons à présent effectuer une requête sur ce script PHP à partir de notre code JavaScript :

```
$(document).ready(function() {
  var $autocomplete = $('<ul class="autocomplete"></ul>')
  .hide()
  .insertAfter('#search-text');
  $('#search-text').keyup(function() {
    $.ajax({
      'url': '../search/autocomplete.php',
      'data': {'search-text': $('#search-text').val()},
      'dataType': 'json',
      'type': 'GET',
      'success': function(data) {
        if (data.length) {
          $autocomplete.empty();
          $.each(data, function(index, term) {
            $('<li></li>').text(term).appendTo($autocomplete);
          });
          $autocomplete.show();
        }
      }
    });
  });
});
```

Nous devons utiliser l'événement `keyup`, non `keydown` ou `keypress`, pour le déclenchement de la requête AJAX. En effet, `keydown` ou `keypress` se produisent pendant la procédure d'appui sur la touche, avant que le caractère ne soit entré dans le champ. Si nous réagissions à ces événements et émettions la requête, la liste des suggestions serait à des lieues du texte recherché. Après la saisie du troisième caractère, par exemple, la requête AJAX serait effectuée avec les deux premiers uniquement. En traitant l'événement `keyup`, nous évitons ce problème.

Dans notre feuille de style, nous positionnons la liste des complétions de manière absolue pour qu'elle chevauche le texte qui se trouve en dessous. Lors de la saisie dans le champ de recherche, nous obtenons les termes reconnus (voir Figure 8.17).

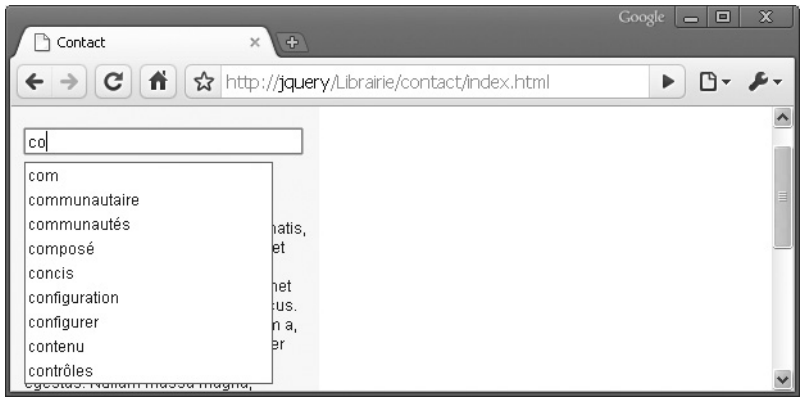


Figure 8.17
Affichage des complétions possibles.

Pour afficher correctement la liste des suggestions, nous devons tenir compte du mécanisme d'auto-complétion intégré à certains navigateurs web. Les navigateurs mémorisent souvent ce que les utilisateurs ont saisi dans un champ et affichent ces propositions lors de l'utilisation suivante du formulaire. Cela risque d'être perturbant lorsque notre mécanisme d'auto-complétion est également en place (voir Figure 8.18).

Heureusement, il est possible de désactiver cette fonctionnalité dans les navigateurs en fixant à `off` l'attribut `autocomplete` du champ du formulaire. Nous pouvons le faire directement dans le contenu HTML, mais nous remettrions alors en cause le principe d'amélioration progressive car nous désactiverions la fonctionnalité d'auto-complétion du navigateur sans proposer la nôtre. Nous ajoutons donc cet attribut depuis le script :

```
$('#search-text').attr('autocomplete', 'off')
```

Remplir le champ de recherche

L'intérêt de notre liste de suggestions est limité si nous ne pouvons pas placer l'un des termes proposés dans le champ de recherche. Tout d'abord, nous autorisons la confirmation d'une suggestion par un clic de souris :

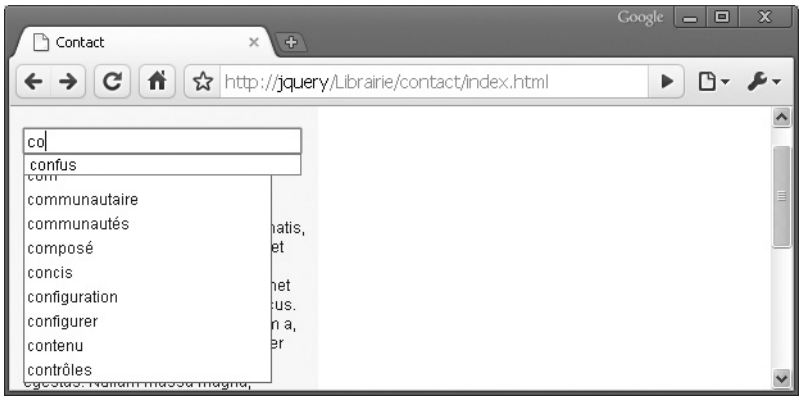


Figure 8.18

Deux mécanismes d'auto-complétion en concurrence.

```
'success': function(data) {
  if (data.length) {
    $autocomplete.empty();
    $.each(data, function(index, term) {
      $('<li></li>').text(term)
      .appendTo($autocomplete)
      .click(function() {
        $('#search-text').val(term);
        $autocomplete.hide();
      });
    });
    $autocomplete.show();
  }
}
```

Grâce à cette modification, l'élément de la liste sur lequel l'utilisateur clique est ajouté dans le champ de recherche. Nous masquons également la fenêtre des suggestions après avoir sélectionné une proposition.

Navigation au clavier

Puisque l'utilisateur a déjà les mains sur le clavier pour saisir le terme à rechercher, pourquoi ne pas lui permettre d'effectuer la sélection à partir du clavier. Pour cela, nous devons conserver une trace de l'élément sélectionné. Tout d'abord, nous ajoutons une fonction d'aide qui enregistre l'indice de l'élément et met en place les effets visuels qui indiquent l'élément actuellement sélectionné :

```
var selectedItem = null;
var setSelectedItem = function(item) {
  selectedItem = item;
  if (selectedItem === null) {
    $autocomplete.hide();
    return;
  }
  if (selectedItem < 0) {
    selectedItem = 0;
  }
```

```
    if (selectedItem >= $autocomplete.find('li').length) {
      selectedItem = $autocomplete.find('li').length - 1;
    }
    $autocomplete.find('li').removeClass('selected')
      .eq(selectedItem).addClass('selected');
    $autocomplete.show();
  };
```

La variable `selectedItem` sera fixée à `null` si aucun élément n'est sélectionné. Puisque nous invoquons toujours `setSelectedItem()` pour modifier la valeur de la variable, nous pouvons être certains que la liste des suggestions est seulement visible lorsqu'un élément est sélectionné.

Les deux tests numériques sur la valeur de `selectedItem` permettent de rester dans la plage de sélection valide. Sans eux, `selectedItem` pourrait prendre n'importe quelle valeur, y compris négative. La fonction s'assure que la valeur actuelle de `selectedItem` correspond toujours à un indice valide dans la liste des suggestions.

Voici la version du code fondée sur la nouvelle fonction :

```
$('#search-text').attr('autocomplete', 'off').keyup(function() {
  $.ajax({
    'url': '../search/autocomplete.php',
    'data': {'search-text': $('#search-text').val()},
    'dataType': 'json',
    'type': 'GET',
    'success': function(data) {
      if (data.length) {
        $autocomplete.empty();
        $.each(data, function(index, term) {
          $('<li></li>').text(term)
          .appendTo($autocomplete)
          .mouseover(function() {
            setSelectedItem(index);
          })
          .click(function() {
            $('#search-text').val(term);
            $autocomplete.hide();
          });
        });
        setSelectedItem(0);
      }
      else {
        setSelectedItem(null);
      }
    }
  });
});
```

Cette nouvelle version présente des bénéfices immédiats. Tout d'abord, la liste des suggestions est masquée lorsqu'une recherche ne produit aucun résultat. Ensuite, nous avons ajouté un gestionnaire de `mouseover` qui met en exergue l'élément sous le pointeur de la souris. Enfin, la première proposition est surlignée dès l'affichage de la liste des suggestions (voir Figure 8.19).

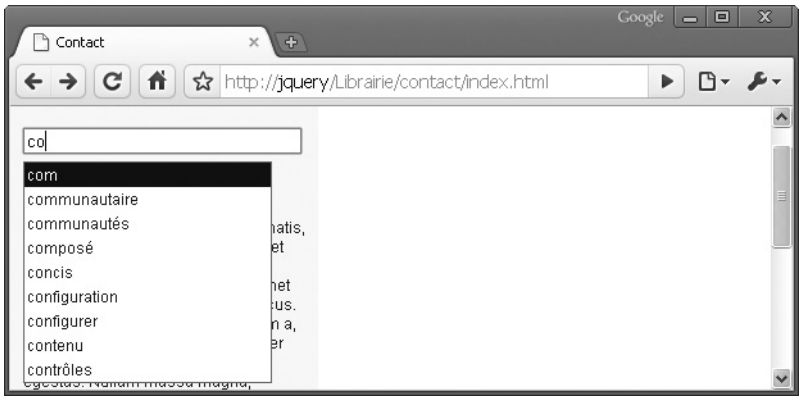


Figure 8.19

L'entrée sélectionnée est repérée.

Nous devons à présent faire en sorte que la sélection d'une proposition puisse se faire à l'aide des touches du clavier.

Prise en charge des touches de direction

L'attribut `keyCode` de l'objet `event` permet de déterminer la touche qui a été enfoncée. Nous pouvons ainsi surveiller les codes `38` et `40`, qui correspondent aux touches de direction haut et bas, et réagir de façon appropriée :

```
$('#search-text').attr('autocomplete', 'off').keyup(function(event) {
  if (event.keyCode > 40 || event.keyCode == 8) {
    // Les touches ayant les codes 40 et inférieurs sont particulières
    // (Entrée, touches de direction, Échap, etc.).
    // Le code 8 correspond à la touche Retour arrière.
    $.ajax({
      'url': '../search/autocomplete.php',
      'data': {'search-text': $('#search-text').val()},
      'dataType': 'json',
      'type': 'GET',
      'success': function(data) {
        if (data.length) {
          $autocomplete.empty();
          $.each(data, function(index, term) {
            $('<li></li>').text(term)
            .appendTo($autocomplete)
            .mouseover(function() {
              setSelectedItem(index);
            })
            .click(function() {
              $('#search-text').val(term);
              $autocomplete.hide();
            });
          });
          setSelectedItem(0);
        }
        else {
          setSelectedItem(null);
        }
```

```
        }
      });
    }
    else if (event.keyCode == 38 && selectedItem !== null) {
      // Appui sur la touche Flèche vers le haut.
      setSelectedItem(selectedItem - 1);
      event.preventDefault();
    }
    else if (event.keyCode == 40 && selectedItem !== null) {
      // Appui sur la touche Flèche vers le bas.
      setSelectedItem(selectedItem + 1);
      event.preventDefault();
    }
  });
```

Notre gestionnaire de `keyup` vérifie le `keyCode` qui a été envoyé et agit en conséquence. Les requêtes AJAX ne sont pas effectuées si la touche enfoncée est une touche spéciale, comme une touche de direction ou la touche Échap. Lorsqu'une touche de direction est détectée et que la liste de suggestions est affichée, le gestionnaire sélectionne l'élément précédent (`-1`) ou suivant (`+1`) en fonction de la touche. Puisque nous utilisons `setSelectedItem()` pour garder les valeurs dans la plage d'indices valides pour la liste, l'utilisateur ne peut pas dépasser les extrémités de la liste.

Insérer une suggestion dans le champ

Nous devons à présent gérer la touche Entrée. Lorsque la liste des suggestions est affichée, un appui sur cette touche doit remplir le champ à l'aide de l'élément sélectionné. Puisque nous effectuons cette opération en deux endroits, nous extrayons la procédure de remplissage du champ (écrite précédemment pour un clic de souris) et créons une fonction séparée :

```
var populateSearchField = function() {
  $('#search-text').val($autocomplete
    .find('li').eq(selectedItem).text());
  setSelectedItem(null);
};
```

Le gestionnaire de `click` appelle alors cette fonction. C'est également le cas dans le traitement de la touche Entrée :

```
$('#search-text').keypress(function(event) {
  if (event.keyCode == 13 && selectedItem !== null) {
    // Appui sur la touche Entrée.
    populateSearchField();
    event.preventDefault();
  }
});
```

Ce gestionnaire n'est plus associé à l'événement `keyup`, mais à `keypress`. Cette modification est nécessaire pour empêcher que l'appui sur la touche ne soumette le formulaire. Si nous attendions le déclenchement de l'événement `keyup`, la soumission serait déjà en cours.

Retirer la liste des suggestions

Il nous reste un dernier point à traiter. La liste des suggestions doit être masquée lorsque l'utilisateur choisit de faire autre chose sur la page. Nous devons commencer par prendre en compte la touche Échap dans le gestionnaire de `keyup` pour que l'utilisateur puisse fermer la liste de cette manière :

```
else if (event.keyCode == 27 && selectedItem !== null) {
  // Appui sur la touche Échap.
  setSelectedItem(null);
}
```

Plus important encore, nous devons masquer la liste lorsque le champ de recherche perd le focus. Voici une première tentative simple :

```
$('#search-text').blur(function(event) {
  setSelectedItem(null);
});
```

Elle a cependant un effet secondaire inattendu. Puisqu'un clic de souris sur la liste retire le focus du champ, ce gestionnaire est invoqué et la liste est masquée. Autrement dit, notre gestionnaire de `click` défini précédemment n'est jamais appelé et il est impossible d'utiliser la souris pour sélectionner une proposition dans la liste.

La solution à ce problème n'est pas simple. Le gestionnaire de `blur` sera toujours appelé avant le gestionnaire de `click`. Une possibilité consiste à masquer la liste lors de la perte du focus, mais en commençant par attendre une fraction de seconde :

```
$('#search-text').blur(function(event) {
  setTimeout(function() {
    setSelectedItem(null);
  }, 250);
});
```

Ainsi, l'événement `click` a une chance d'être déclenché sur l'élément de la liste avant qu'elle ne soit masquée.

Auto-complétion contre recherche dynamique

L'exemple précédent s'est focalisé sur l'auto-complétion du champ de texte car cette technique s'applique à de nombreux formulaires. Toutefois, il est parfois préférable d'utiliser une autre approche appelée *recherche dynamique*. Dans ce cas, la recherche du contenu se fait au cours de la saisie de l'utilisateur.

D'un point de vue fonctionnel, l'auto-complétion et la recherche dynamique se ressemblent énormément. Dans les deux cas, l'appui sur une touche déclenche une requête AJAX vers le serveur, en passant le contenu actuel du champ. Les résultats sont ensuite placés dans une liste déroulante sous le champ. Dans le cas de l'auto-complétion, les résultats correspondent aux termes de recherche possibles. Avec la recherche dynamique, ils correspondent aux pages réelles qui contiennent les termes de recherche saisis.

Du côté JavaScript, le code de mise en œuvre de ces deux fonctionnalités est quasiment identique. Nous n'entrerons donc pas dans les détails. Le choix de la technique à utiliser est une question de compromis. La recherche dynamique apporte plus d'informations à l'utilisateur, mais elle demande généralement plus de ressources.

Mentionnons également que, pour des sites francophones, la recherche peut se faire sans tenir compte des accents, des cédilles et autres caractères spéciaux comme les ligatures. Pour cela, il suffit d'effectuer dans le script PHP les remplacements appropriés à l'aide d'expressions régulières.

Version finale du code

Voici la version finale du code pour la présentation du champ de recherche et la mise en œuvre de l'auto-complétion :

```
$(document).ready(function() {
  var $search = $('#search').addClass('overlabel');
  var $searchInput = $search.find('input');
  var $searchLabel = $search.find('label');

  if ($searchInput.val()) {
    $searchLabel.hide();
  }
  $searchInput
  .focus(function() {
    $searchLabel.hide();
  })
  .blur(function() {
    if (this.value == '') {
      $searchLabel.show();
    }
  });

  $searchLabel.click(function() {
    $searchInput.trigger('focus');
  });

  var $autocomplete = $('<ul class="autocomplete"></ul>')
    .hide()
    .insertAfter('#search-text');
  var selectedItem = null;
  var setSelectedItem = function(item) {
    selectedItem = item;
    if (selectedItem === null) {
      $autocomplete.hide();
      return;
    }
    if (selectedItem < 0) {
      selectedItem = 0;
    }
    if (selectedItem >= $autocomplete.find('li').length) {
      selectedItem = $autocomplete.find('li').length - 1;
    }
```

```
  $autocomplete.find('li').removeClass('selected')
    .eq(selectedItem).addClass('selected');
  $autocomplete.show();
};

var populateSearchField = function() {
  $('#search-text').val($autocomplete.find('li').eq(selectedItem).text());
  setSelectedItem(null);
};

$('#search-text')
.attr('autocomplete', 'off')
.keyup(function(event) {
  if (event.keyCode > 40 || event.keyCode == 8) {
    // Les touches ayant les codes 40 et inférieurs sont particulières
    // (Entrée, touches de direction, Échap, etc.).
    // Le code 8 correspond à la touche Espace arrière.
    $.ajax({
      'url': '../search/autocomplete.php',
      'data': {'search-text': $('#search-text').val()},
      'dataType': 'json',
      'type': 'GET',
      'success': function(data) {
        if (data.length) {
          $autocomplete.empty();
          $.each(data, function(index, term) {
            $('<li></li>').text(term)
            .appendTo($autocomplete)
            .mouseover(function() {
              setSelectedItem(index);
            }).click(populateSearchField);
          });
          setSelectedItem(0);
        }
        else {
          setSelectedItem(null);
        }
      }
    });
  }
  else if (event.keyCode == 38 && selectedItem !== null) {
    // Appui sur la touche Flèche vers le haut.
    setSelectedItem(selectedItem - 1);
    event.preventDefault();
  }
  else if (event.keyCode == 40 && selectedItem !== null) {
    // Appui sur la touche Flèche vers le bas.
    setSelectedItem(selectedItem + 1);
    event.preventDefault();
  }
  else if (event.keyCode == 27 && selectedItem !== null) {
    // Appui sur la touche Échap.
    setSelectedItem(null);
  }
}).keypress(function(event) {
  if (event.keyCode == 13 && selectedItem !== null) {
```

```
        // Appui sur la touche Entrée.
        populateSearchField();
        event.preventDefault();
      }
    }).blur(function(event) {
      setTimeout(function() {
        setSelectedItem(null);
      }, 250);
    });
  });
```

8.3 Manipuler des données numériques

Les fonctionnalités des formulaires que nous venons d'étudier s'appliquent aux saisies textuelles effectuées par l'utilisateur. Toutefois, les données contenues dans les formulaires peuvent être numériques. Lorsque les valeurs manipulées sont des nombres, nous pouvons améliorer les formulaires de bien d'autres manières.

Dans le cadre de notre librairie, le panier d'achat constitue un parfait exemple de formulaire numérique. L'utilisateur doit pouvoir actualiser les quantités de produits achetés et nous devons lui présenter des données numériques pour les prix et les montants totaux.

Structure de la table du panier d'achat

Le balisage HTML du panier d'achat décrit une structure de table parmi les plus complexes que nous ayons rencontrées jusque-là :

```
<form action="checkout.php" method="post">
  <table id="cart">
    <thead>
      <tr>
        <th class="item">Produit</th>
        <th class="quantity">Quantité</th>
        <th class="price">Prix</th>
        <th class="cost">Total</th>
      </tr>
    </thead>
    <tfoot>
      <tr class="subtotal">
        <td class="item">Sous-total</td>
        <td class="quantity"></td>
        <td class="price"></td>
        <td class="cost">152,95 €</td>
      </tr>
      <tr class="tax">
        <td class="item">TVA</td>
        <td class="quantity"></td>
        <td class="price">5,5 %</td>
        <td class="cost">8,41 €</td>
      </tr>
      <tr class="shipping">
        <td class="item">Livraison</td>
```

```
      <td class="quantity">5</td>
      <td class="price">2 € par produit</td>
      <td class="cost">10,00 €</td>
    </tr>
    <tr class="total">
      <td class="item">Total</td>
      <td class="quantity"></td>
      <td class="price"></td>
      <td class="cost">171,36 €</td>
    </tr>
    <tr class="actions">
      <td></td>
      <td>
        <input type="button" name="recalculate"
            value="Recalculer" id="recalculate" />
      </td>
      <td></td>
      <td>
        <input type="submit" name="submit"
            value="Commander" id="submit" />
      </td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td class="item">
        Building Telephony Systems With Asterisk
      </td>
      <td class="quantity">
        <input type="text" name="quantity-2" value="1"
            id="quantity-2" maxlength="3" />
      </td>
      <td class="price">26,99 €</td>
      <td class="cost">26,99 €</td>
    </tr>
    <tr>
      <td class="item">
        Smarty PHP Template Programming and Applications
      </td>
      <td class="quantity">
        <input type="text" name="quantity-1" value="2"
            id="quantity-1" maxlength="3" />
      </td>
      <td class="price">35,99 €</td>
      <td class="cost">71,98 €</td>
    </tr>
    <tr>
      <td class="item">
        Creating your MySQL Database
      </td>
      <td class="quantity">
        <input type="text" name="quantity-3" value="1"
            id="quantity-3" maxlength="3" />
      </td>
      <td class="price">17,99 €</td>
      <td class="cost">17,99 €</td>
    </tr>
```

```
        <tr>
          <td class="item">
            Drupal: Creating Blogs, Forums, Portals, and Community Websites
          </td>
          <td class="quantity">
            <input type="text" name="quantity-4" value="1"
                id="quantity-4" maxlength="3" />
          </td>
          <td class="price">35,99 €</td>
          <td class="cost">35,99 €</td>
        </tr>
      </tbody>
    </table>
  </form>
```

La table inclut un élément `<tfoot>` dont l'utilisation est plutôt rare. À l'instar de `<thead>`, il regroupe un ensemble de lignes de la table. Bien que cet élément soit placé avant le corps de la table, son affichage sur la page se fait après (voir Figure 8.20).

Figure 8.20

L'élément `<tfoot>` est affiché après le corps de la table.

Cette organisation du code source, si elle n'est pas intuitive aux concepteurs qui réfléchissent en termes de présentation de la table, est utile aux personnes présentant un handicap visuel. En effet, lorsque la table est lue par des dispositifs d'assistance, le pied est lu avant le corps potentiellement long, ce qui permet à l'utilisateur d'avoir un résumé du contenu à venir.

Nous avons également ajouté une classe à chaque cellule de la table de manière à identifier la colonne qui la contient. Au chapitre précédent, nous avons expliqué comment retrouver la colonne d'une cellule en examinant son indice dans sa ligne. Ici, nous faisons un compromis qui nous permet de simplifier le code JavaScript, au prix d'un bali-

sage HTML légèrement plus complexe. Grâce à la classe qui identifie la colonne de chaque cellule, nos sélecteurs seront un peu plus simples.

Avant de passer à la manipulation des champs du formulaire, nous allons appliquer un effet de bande à la table avec notre code habituel :

```
$(document).ready(function() {
  $('#cart tbody tr:nth-child(even)').addClass('alt');
});
```

À nouveau, nous faisons en sorte d'appliquer les couleurs uniquement aux lignes qui se trouvent dans le corps de la table (voir Figure 8.21).

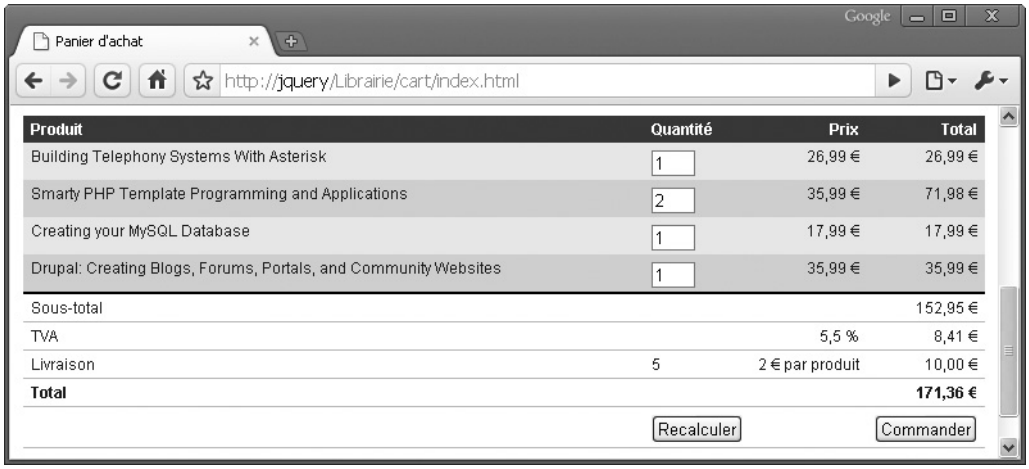


Figure 8.21

Application des bandes aux lignes des produits.

Rejeter toute saisie non numérique

Au cours de l'amélioration du formulaire de contact, nous avons présenté quelques *techniques de validation* des données saisies. Nous avons vérifié que le texte entré par l'utilisateur correspondait à ce que nous attendions, ce qui nous a permis de lui fournir des instructions avant que le formulaire ne soit envoyé au serveur. Nous allons à présent examiner le pendant de la validation, appelé *masque de saisie*.

La validation confronte les données saisies par l'utilisateur à certains critères définissant leur validité. Un masque de saisie applique les critères pendant que l'utilisateur entre les données et interdit simplement les touches invalides. Par exemple, dans notre formulaire de panier d'achat, les champs de saisie ne doivent contenir que des nombres. Nous pouvons écrire du code qui annule les effets des appuis sur des touches non numériques :

```
$('td.quantity input').keypress(function(event) {
  if (event.which && (event.which < 48 || event.which > 57)) {
    event.preventDefault();
  }
});
```

Dans la fonction d'auto-complétion pour le champ de recherche, nous avons intercepté l'événement `keyup`. Nous avons pu ainsi examiner la propriété `.keyCode` de l'objet `event` afin de connaître la touche enfoncée. À présent, nous observons à la place l'événement `keypress`. Cet événement ne dispose pas d'une propriété `.keyCode`, mais propose à la place `.which`. Elle indique le caractère ASCII qui correspond à la séquence de touches.

Si la frappe produit un caractère (autrement dit, il ne s'agit pas d'une touche de direction, de Suppr ou d'une autre touche d'édition) et que ce caractère se trouve en dehors de la plage des codes ASCII représentant des nombres, nous invoquons la méthode `.preventDefault()` sur l'objet `event`. Nous l'avons vu précédemment, cela interrompt le traitement de l'événement par le navigateur. Dans ce cas, cela signifie que le caractère n'est pas inséré dans le champ. Les champs de quantité acceptent à présent uniquement des nombres.

Calculs numériques

Passons à la manipulation des nombres saisis par l'utilisateur dans le formulaire de panier d'achat. Ce formulaire comprend un bouton RECALCULER, qui doit déclencher la soumission du formulaire au serveur, où les nouveaux totaux sont calculés, puis le formulaire est à nouveau présenté à l'utilisateur. Toutefois, cet échange n'est pas nécessaire. Tout ce travail peut être réalisé dans le navigateur avec jQuery.

Dans le formulaire, le calcul le plus simple se situe au niveau de la ligne LIVRAISON, qui affiche la quantité totale des produits commandés. Lorsque l'utilisateur modifie une quantité dans l'une des lignes des produits, nous devons additionner toutes les valeurs saisies pour obtenir le nouveau total et l'afficher dans la cellule correspondante :

```
var $quantities = $('td.quantity input');
$quantities.change(function() {
  var totalQuantity = 0;
  $quantities.each(function() {
    var quantity = parseInt(this.value, 10);
    totalQuantity += quantity;
  });
  $('tr.shipping td.quantity').text(String(totalQuantity));
});
```

Plusieurs possibilités s'offrent à nous quant à l'événement déclenchant cette opération de calcul. Nous pouvons observer l'événement `keypress` et effectuer le calcul à chaque appui sur une touche. Nous pouvons également observer l'événement `blur`, qui est déclenché chaque fois que l'utilisateur quitte le champ. Toutefois, nous allons être un peu plus économes en temps processeur et effectuer les calculs uniquement lors du

déclenchement de l'événement `change`. Ainsi, nous calculons le nouveau total uniquement si l'utilisateur quitte le champ après avoir saisi une valeur différente.

La quantité totale est obtenue à l'aide d'une simple boucle `.each()`. La propriété `.value` d'un champ permet d'obtenir sa valeur sous forme d'une chaîne de caractères. Nous utilisons donc la fonction intégrée `parseInt()` pour convertir cette valeur en un entier. Cette approche permet d'éviter des situations étranges dans lesquelles l'addition est interprétée comme une concaténation de chaînes, puisque ces deux opérations utilisent le même symbole pour l'opérateur. À l'inverse, nous devons passer une chaîne à la méthode jQuery `.text()` pour afficher le résultat du calcul. Nous utilisons donc la fonction `String()` pour convertir la quantité totale calculée en une chaîne de caractères.

Lorsque l'utilisateur change une quantité, le total est désormais actualisé automatiquement (voir Figure 8.22).

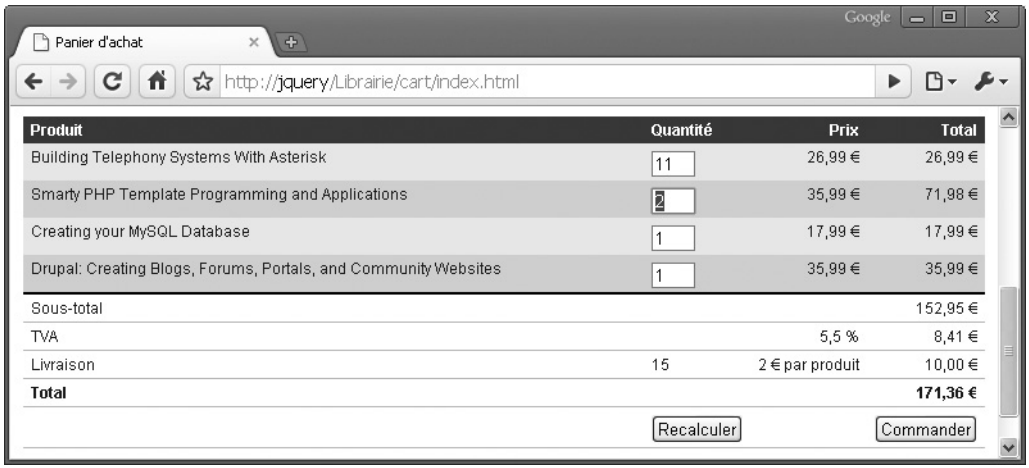


Figure 8.22

Actualisation automatique de la quantité totale.

Parser *et mettre en forme la monnaie*

Nous pouvons à présent passer aux totaux affichés dans la colonne de droite. Chaque coût total d'une ligne est calculé en multipliant la quantité saisie par le prix du produit. Puisque chaque ligne fait l'objet de plusieurs opérations, nous commençons par remanier le code de calcul des quantités, en le fondant sur la ligne à la place du champ :

```
$('#cart tbody tr').each(function() {
  var quantity = parseInt($('td.quantity input', this).val(), 10);
  totalQuantity += quantity;
});
```

Nous obtenons le même résultat, mais disposons désormais d'un endroit pratique où ajouter le calcul du coût total de chaque ligne :

```
$('td.quantity input').change(function() {
  var totalQuantity = 0;
  $('#cart tbody tr').each(function() {
    var price = parseFloat($('td.price', this).text()
        .replace(',', '.').replace(/[^\d\.]/g, ''));
    price = isNaN(price) ? 0 : price;
    var quantity = parseInt($('td.quantity input', this).val());
    var cost = quantity * price;
    var costFR = cost.toString().replace('.', ',');
    $('td.cost', this).text(costFR + ' €');
    totalQuantity += quantity;
  });
  $('tr.shipping td.quantity').text(String(totalQuantity));
});
```

Nous récupérons le prix de chaque produit dans la table en employant la technique déjà utilisée précédemment pour trier les lignes en fonction du prix. L'expression régulière remplace la virgule par un point et retire tout caractère qui n'est ni un chiffre ni un point, puis la chaîne résultante est passée à `parseFloat()`, qui interprète la valeur comme un nombre réel. Puisque le calcul se fera avec ce résultat, nous vérifions qu'un nombre a été trouvé et fixons le prix à `0` si ce n'est pas le cas. Enfin, nous multiplions le prix par la quantité pour obtenir le coût total. Cette valeur réelle est convertie en une chaîne de caractères, dans laquelle nous remplaçons le point par une virgule et que nous plaçons ensuite dans la colonne de total en lui ajoutant le symbole €. La Figure 8.23 illustre notre calcul des totaux.

Figure 8.23

Calcul du montant total de chaque ligne d'un produit.

Gérer les chiffres après la virgule

Même si nous avons placé le symbole de l'euro après le montant total, JavaScript ne sait pas que nous manipulons des valeurs monétaires. Pour l'ordinateur, il s'agit simplement de nombres, qui doivent être affichés comme tels. Autrement dit, si les centimes du total se terminent par un ou deux zéros, ceux-ci ne sont pas affichés (voir Figure 8.24).

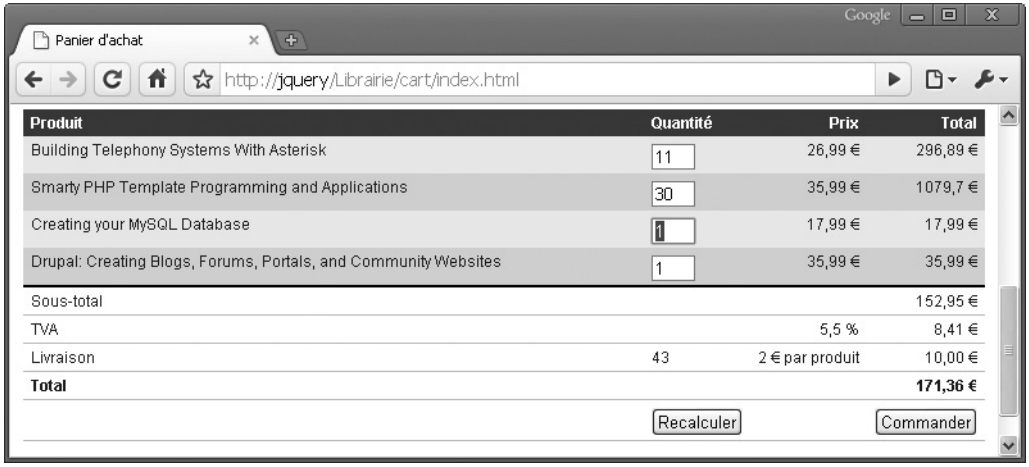


Figure 8.24

Les zéros de fin ne sont pas affichés.

Vous le constatez, le total 1079,70 € s'affiche sous la forme 1079,7 €. Pire encore, les limites sur la précision des calculs peuvent parfois conduire à des erreurs d'arrondi et laisser croire que le résultat est faux (voir Figure 8.25).

Heureusement, la solution à ces deux problèmes est simple. La classe JavaScript `Number` offre plusieurs méthodes pour prendre en charge ce genre de cas, et `.toFixed()` convient parfaitement ici. Cette méthode prend en argument un nombre de chiffres après la virgule et retourne une chaîne qui représente le nombre réel arrondi à cette précision :

```
$('#cart tbody tr').each(function() {
  var price = parseFloat($('td.price', this).text()
    .replace(',', '.').replace(/[^\d\.]/g, ''));
  price = isNaN(price) ? 0 : price;
  var quantity = parseInt($('td.quantity input', this).val());
  var cost = quantity * price;
  var costFR = cost.toFixed(2).replace('.', ',');
  $('td.cost', this).text(costFR + ' €');
  totalQuantity += quantity;
});
```

La Figure 8.26 montre que les totaux ont à présent l'aspect d'une valeur monétaire normale.

Figure 8.25

Effet des limites sur la précision des calculs.

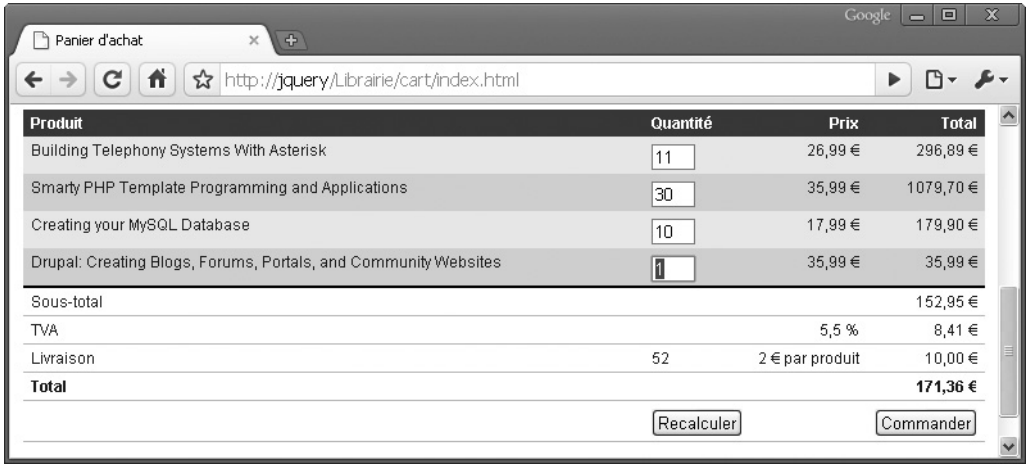


Figure 8.26

Mise en forme des valeurs monétaires.

INFO

Après une longue suite d'opérations arithmétiques, l'arrondi des nombres réels peut cumuler tellement d'erreurs que la méthode `.toFixed()` ne puisse pas les masquer. Dans les applications, la meilleure manière de gérer les valeurs monétaires consiste à les enregistrer et à les manipuler en centimes, sous forme d'entiers. La virgule est ajoutée uniquement lors de l'affichage.

Autres calculs

Les autres calculs nécessaires sur la page suivent un schéma semblable. Pour le sous-total, nous pouvons additionner les totaux de chaque ligne lorsqu'ils sont calculés et afficher le résultat en utilisant la mise en forme précédente (voir Figure 8.27) :

```
$('td.quantity input').change(function() {
  var totalQuantity = 0;
  var totalCost = 0;
  $('#cart tbody tr').each(function() {
    var price = parseFloat($('td.price', this).text()
      .replace(',', '.').replace(/[^\d\.]/g, ''));
    price = isNaN(price) ? 0 : price;
    var quantity = parseInt($('td.quantity input', this).val());
    var cost = quantity * price;
    var costFR = cost.toFixed(2).replace('.', ',');
    $('td.cost', this).text(costFR + ' €');
    totalQuantity += quantity;
    totalCost += cost;
  });
  $('tr.shipping td.quantity').text(String(totalQuantity));
  var subTotalCostFR = totalCost.toFixed(2).replace('.', ',');
  $('tr.subtotal td.cost').text(subTotalCostFR + ' €');
});
```

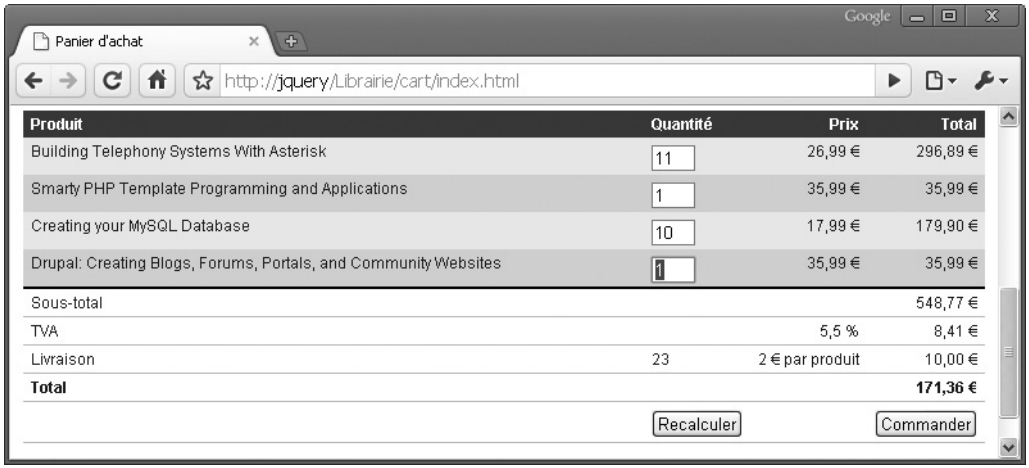


Figure 8.27

Calcul et mise en forme du sous-total.

Arrondir des valeurs

Pour calculer la TVA, nous devons diviser le taux par 100 et multiplier ensuite la valeur obtenue par le sous-total. Puisque le montant de la TVA est toujours arrondi, nous devons nous assurer que la bonne valeur est utilisée à la fois dans l'affichage et dans les

calculs ultérieurs. La fonction JavaScript `Math.ceil()` permet d'arrondir un nombre à l'entier supérieur le plus proche, mais, puisque nous manipulons des euros et des centimes d'euro, nous devons être plus astucieux :

```
var taxRate = parseFloat($('tr.tax td.price').text()
  .replace(',', '.').replace(/[^\d\.]/g, '')) / 100;
var tax = Math.ceil(totalCost * taxRate * 100) / 100;
var taxFR = tax.toFixed(2).replace('.', ',');
$('tr.tax td.cost').text(taxFR + ' €');
totalCost += tax;
```

Le montant de la TVA est tout d'abord multiplié par 100 afin d'obtenir une valeur en centimes, non en euros. Cette valeur peut ensuite être arrondie par `Math.ceil()`, puis divisée par 100 pour la convertir à nouveau en euros. Enfin, nous invoquons la méthode `.toFixed()` pour obtenir le résultat correct (voir Figure 8.28).

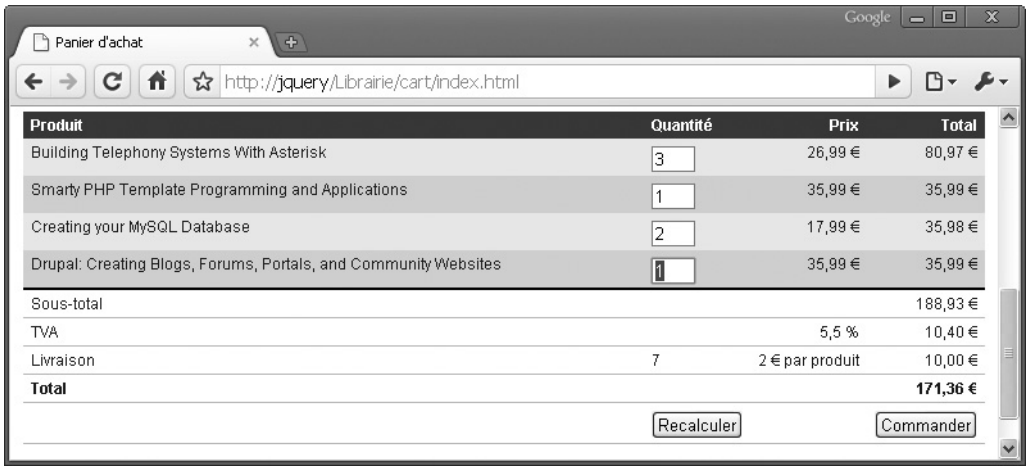


Figure 8.28

Calcul du montant de la TVA.

Touches finales

Le calcul du montant de la livraison est plus simple que celui de la TVA car, dans notre exemple, il n'implique aucun arrondi. Il suffit de multiplier le coût de livraison d'un produit par le nombre de produits :

```
$('tr.shipping td.quantity').text(String(totalQuantity));
var shippingRate = parseFloat($('tr.shipping td.price').text()
  .replace(',', '.').replace(/[^\d\.]/g, ''));
var shipping = totalQuantity * shippingRate;
var shippingFR = shipping.toFixed(2).replace('.', ',');
$('tr.shipping td.cost').text(shippingFR + ' €');
totalCost += shipping;
```

Tout au long de nos calculs, nous avons maintenu à jour le montant total. Il ne nous reste plus qu'à mettre en forme la variable `totalCost` avant de l'afficher :

```
var totalCostFR = totalCost.toFixed(2).replace('.', ',');
$('tr.total td.cost').text(totalCostFR + ' €');
```

À présent, nous avons intégralement reproduit les calculs qui seraient effectués sur le serveur et nous pouvons donc masquer le bouton RECALCULER :

```
$('#recalculate').hide();
```

Cette modification illustre à nouveau notre principe d'*amélioration progressive*. Tout d'abord, nous faisons en sorte que la page fonctionne correctement sans JavaScript. Ensuite, nous utilisons jQuery pour effectuer les mêmes tâches, mais de manière plus élégante lorsque c'est possible (voir Figure 8.29).

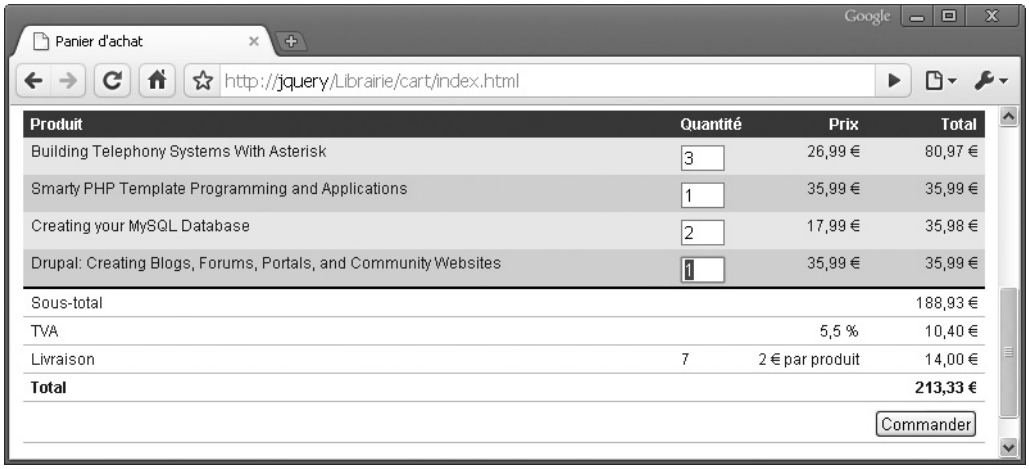


Figure 8.29

Version jQuery du formulaire de panier d'achat.

Retirer des produits

Si l'utilisateur souhaite retirer des produits de son panier, il peut modifier le champ QUANTITÉ pour le fixer à zéro. Cependant, nous pouvons mettre en place un comportement plus rassurant en ajoutant des boutons RETIRER explicites pour chaque produit. Un clic sur le bouton aura le même effet que la modification du champ QUANTITÉ, mais le retour visuel renforcera le fait que le produit ne sera pas facturé.

Nous commençons par ajouter les nouveaux boutons. Puisqu'ils ne fonctionneront pas sans JavaScript, nous ne les plaçons pas dans le fichier HTML, mais utilisons jQuery pour les insérer dans chaque ligne :

```
$('<th> </th>')
  .insertAfter('#cart thead th:nth-child(2)');
$('#cart tbody tr').each(function() {
  $deleteButton = $('<img />').attr({
    'width': '16',
    'height': '16',
    'src': '../images/cross.png',
    'alt': 'retirer du panier',
    'title': 'retirer du panier',
    'class': 'clickable'
  });
  $('<td></td>')
    .insertAfter($('td:nth-child(2)', this))
    .append($deleteButton);
});
$('<td> </td>')
  .insertAfter('#cart tfoot td:nth-child(2)');
```

Nous devons créer des cellules vides dans les lignes d'en-tête et de pied de la table pour que les colonnes restent alignées. Les boutons sont créés et ajoutés dans les lignes du corps uniquement (voir Figure 8.30).

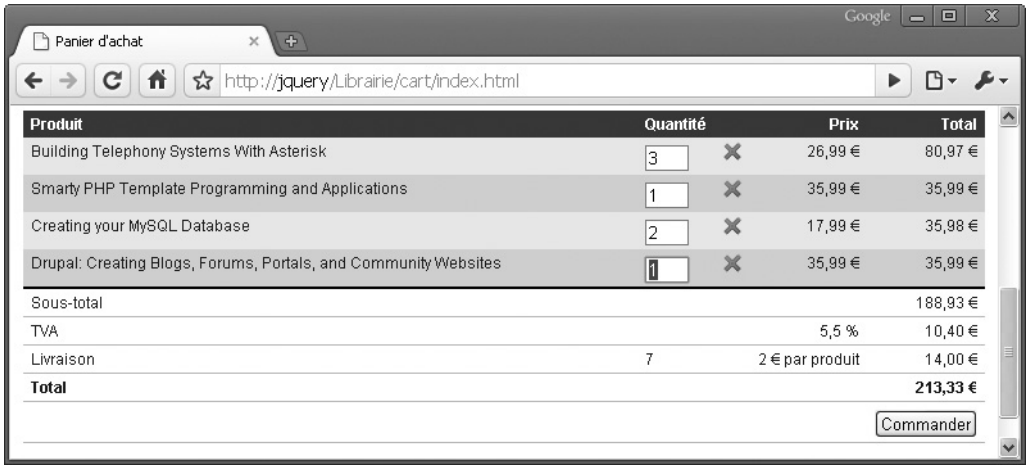


Figure 8.30

Ajout des boutons de suppression.

Nous devons à présent associer un comportement aux boutons. Nous changeons la définition d'un bouton de manière à ajouter un gestionnaire de `click` :

```
$deleteButton = $('<img />').attr({
  'width': '16',
  'height': '16',
  'src': '../images/cross.png',
  'alt': 'retirer du panier',
```

```
    'title': 'retirer du panier',
    'class': 'clickable'
  }).click(function() {
    $(this).parents('tr').find('td.quantity input')
      .val(0);
  });
```

Ce gestionnaire obtient le champ QUANTITÉ placé sur la même ligne que le bouton et fixe sa valeur à `0`. Si le champ est bien mis à jour, les calculs sont désynchronisés (voir Figure 8.31).

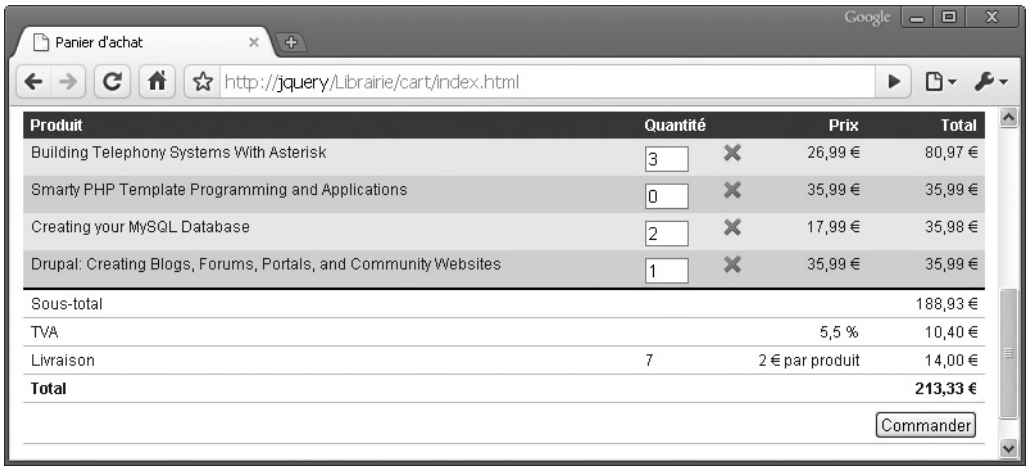


Figure 8.31

La suppression d'un produit n'actualise pas les montants.

Nous devons déclencher les calculs comme si l'utilisateur avait manuellement modifié la valeur du champ :

```
$deleteButton = $('<img />').attr({
  'width': '16',
  'height': '16',
  'src': '../images/cross.png',
  'alt': 'retirer du panier',
  'title': 'retirer du panier',
  'class': 'clickable'
}).click(function() {
  $(this).parents('tr').find('td.quantity input')
    .val(0).trigger('change');
});
```

À présent, les totaux sont actualisés lors d'un clic sur le bouton (voir Figure 8.32).

Figure 8.32

La suppression d'un produit actualise les montants.

Passons à l'effet visuel. Nous masquons la ligne qui correspond au bouton sur lequel l'utilisateur a cliqué afin que le produit soit clairement retiré du panier (voir Figure 8.33) :

```
$deleteButton = $('<img />').attr({
  'width': '16',
  'height': '16',
  'src': '../images/cross.png',
  'alt': 'retirer du panier',
  'title': 'retirer du panier',
  'class': 'clickable'
}).click(function() {
  $(this).parents('tr').find('td.quantity input')
    .val(0).trigger('change')
  .end().hide();
});
```

Même si la ligne est masquée, le champ est toujours présent dans le formulaire. Autrement dit, il sera envoyé avec les autres informations du formulaire et le code côté serveur retirera le produit.

Notre effet de bande sur la table est perturbé par la disparition de la ligne. Pour corriger ce problème, nous déplaçons notre code d'application de cet effet dans une fonction de manière à pouvoir l'invoquer à nouveau ultérieurement. Nous en profitons pour modifier le code afin que l'alternance des bandes tienne compte des lignes masquées. Malheureusement, même si nous écartons les lignes masquées par un filtre, nous ne pouvons pas utiliser le sélecteur `:nth-child(even)` car il appliquera la classe `alt` à toutes les lignes visibles qui sont un enfant "pair" de leur parent. Voyons ce qui se passe

Figure 8.33

La ligne du produit retiré est masquée.

lorsque nous utilisons le code suivant sur les quatre lignes de la table alors que la deuxième est masquée :

```
$('#cart tbody tr').removeClass('alt')
  .filter(':visible:nth-child(even)').addClass('alt');
```

Nous obtenons le balisage suivant (résumé) :

```
<tr> ... </tr>
<tr style="display: none"> ... </tr>
<tr> ... </tr>
<tr class="alt"> ... </tr>
```

Trois lignes sont visibles, mais la classe `alt` est appliquée uniquement à la quatrième. Nous avons donc besoin de l'expression de sélection `:visible:odd` puisqu'elle choisira une ligne sur deux après avoir écarté de la sélection les lignes masquées (et elle tient compte du sélecteur qui commence à compter à partir de un, non de zéro). Nous l'avons vu au Chapitre 2, l'utilisation de `:odd` ou de `:even` peut avoir des résultats inattendus si plusieurs éléments `<tbody>` sont présents ; ce n'est pas le cas dans notre exemple. Voici la nouvelle version de la fonction avec ce sélecteur :

```
var stripe = function() {
  $('#cart tbody tr').removeClass('alt')
    .filter(':visible:odd').addClass('alt');
};
stripe();
```

Nous l'invoquons après la suppression d'une ligne (voir Figure 8.34) :

```
$deleteButton = $('<img />').attr({
  'width': '16',
  'height': '16',
```

```
    'src': '../images/cross.png',
    'alt': 'retirer du panier',
    'title': 'retirer du panier',
    'class': 'clickable'
  }).click(function() {
    $(this).parents('tr').find('td.quantity input')
      .val(0).trigger('change')
    .end().hide();
    stripe();
  });
```

Figure 8.34

Réapplication des bandes après la suppression d'un produit.

Nous terminons ainsi une autre amélioration avec jQuery, qui n'a aucune implication sur le code côté serveur. Pour celui-ci, l'utilisateur a simplement saisi la valeur `0` dans un champ du formulaire, mais, pour l'utilisateur, il s'agit d'une opération de suppression différente d'une modification de la quantité.

Modifier les informations de livraison

La page du panier d'achat comprend également un formulaire pour les informations de livraison. En réalité, au moment du chargement de la page, il ne s'agit pas d'un formulaire et, si JavaScript n'est pas activé, ces informations sont placées dans une petite boîte sur le côté droit de la zone de contenu, avec un lien vers une page à partir de laquelle l'utilisateur peut les modifier (voir Figure 8.35).

En revanche, lorsque JavaScript est disponible, et avec la puissance de jQuery, nous pouvons convertir ce petit lien en un formulaire complet. Pour cela, nous allons demander le formulaire à une page PHP. En général, les données du formulaire seront enregistrées dans une base de données, mais pour le besoin de notre démonstration nous les conservons de manière statique dans un tableau PHP.

Figure 8.35

La boîte affichant les informations de livraison.

Pour obtenir le formulaire et le faire apparaître dans la boîte LIVRAISON À, nous utilisons la méthode `$.get()` dans le gestionnaire d'événements `.click()` :

```
$(document).ready(function() {
  $('#shipping-name').click(function() {
    $.get('shipping.php', function(data) {
      $('#shipping-name').remove();
      $(data).hide().appendTo('#shipping').slideDown();
    });
    return false;
  });
});
```

En vérifiant l'absence de la variable serveur `$_SERVER['HTTP_X_REQUESTED_WITH']` avant d'afficher la plus grande partie de la page, la page PHP (`shipping.php`) retourne uniquement un fragment de la page complète, le formulaire, suite à la requête effectuée avec la méthode `$.get()`.

Dans la fonction de rappel de la méthode `$.get()`, nous supprimons le nom sur lequel nous venons de cliquer et le remplaçons par le formulaire et ses données fournis par `shipping.php`. La fonction retourne `false` afin que l'événement par défaut faisant suite au clic sur le lien (chargement de la page désignée par l'attribut `href`) ne se produise pas. La boîte LIVRAISON À est devenue un formulaire modifiable (voir Figure 8.36).

L'utilisateur peut à présent modifier les informations de livraison sans quitter la page.

Dans l'étape suivante, nous devons détourner la soumission du formulaire et envoyer les données modifiées au serveur avec jQuery. Nous commençons par sérialiser les données saisies dans le formulaire et les enregistrerons dans la variable `postData`. Nous les envoyons ensuite au serveur en utilisant à nouveau `shipping.php` :

```
$(document).ready(function() {
  $('#shipping form').submit(function() {
    var postData = $(this).serialize();
    $.post('shipping.php', postData);
    return false;
  };
});
```

Figure 8.36

Le formulaire de saisie des informations de livraison.

INFO

Le plugin jQuery Form (**http://www.malsup.com/jquery/form/**) propose une méthode `.serialize()` plus robuste. Nous vous conseillons de l'utiliser dans la plupart des cas de soumission d'un formulaire avec AJAX.

À ce stade, nous pouvons supprimer le formulaire et remettre la boîte LIVRAISON À dans son état d'origine. Nous effectuons cette opération dans la fonction de rappel de la méthode `$.post()` que nous venons d'employer :

```
$(document).ready(function() {
  $('#shipping form').submit(function() {
    var postData = $(this).serialize();
    $.post('shipping.php', postData, function(data) {
      $('#shipping form').remove();
      $(data).appendTo('#shipping');
    });
    return false;
  };
});
```

Mais cela ne fonctionne pas ! En procédant ainsi, le gestionnaire d'événements `.submit()` est lié au formulaire LIVRAISON À dès que le DOM est chargé, mais le formulaire ne se trouve dans le DOM qu'après que l'utilisateur a cliqué sur le nom du destinataire. L'événement ne peut pas être associé à quelque chose qui n'existe pas.

Pour contourner ce problème, nous plaçons le code de création du formulaire dans une fonction nommée `editShipping()` et le code de soumission ou de suppression du formulaire dans une fonction nommée `saveShipping()`. Nous pouvons ensuite lier `saveShipping()` dans la fonction de rappel de `$.get()`, après la création du formulaire.

De la même manière, nous pouvons lier `editShipping()` lorsque le DOM est prêt et que le lien de modification de la livraison est recréé dans la fonction de rappel de `$.post()` :

```
$(document).ready(function() {
  var editShipping = function() {
    $.get('shipping.php', function(data) {
      $('#shipping-name').remove();
      $(data).hide().appendTo('#shipping').slideDown();
      $('#shipping form').submit(saveShipping);
    });
    return false;
  };

  var saveShipping = function() {
    var postData = $('#shipping :input').serialize();
    $.post('shipping.php', postData, function(data) {
      $('#shipping form').remove();
      $(data).appendTo('#shipping');
      $('#shipping-name').click(editShipping);
    });
    return false;
  };

  $('#shipping-name').click(editShipping);
});
```

Le code a créé une sorte de motif circulaire, dans lequel une fonction permet à l'autre de lier à nouveau ses gestionnaires d'événements respectifs.

Version finale du code

En réunissant le tout, le code du panier d'achat occupe environ quatre-vingts lignes. Cela représente finalement assez peu si l'on considère les fonctionnalités proposées, en particulier la façon de coder pour améliorer la lisibilité. Si la réduction du nombre de lignes avait été un aspect important, plusieurs instructions jQuery auraient pu être fusionnées grâce au chaînage. Quoi qu'il en soit, voici la version finale du code de mise en œuvre de la page du panier d'achat, qui conclut ce chapitre sur les formulaires :

```
$(document).ready(function() {
  var stripe = function() {
    $('#cart tbody tr').removeClass('alt')
    .filter(':visible:odd').addClass('alt');
  };
  stripe();

  $('#recalculate').hide();

  $('.quantity input').keypress(function(event) {
    if (event.which && (event.which < 48 || event.which > 57)) {
      event.preventDefault();
    }
```

```
  }).change(function() {
    var totalQuantity = 0;
    var totalCost = 0;
    $('#cart tbody tr').each(function() {
      var price = parseFloat($('.price', this).text()
        .replace(',', '.').replace(/[^\d\.]/g, ''));
      price = isNaN(price) ? 0 : price;
      var quantity = parseInt($('.quantity input', this).val(), 10);
      var cost = quantity * price;
      var costFR = cost.toFixed(2).replace('.', ',');
      $('.cost', this).text(costFR + ' €');
      totalQuantity += quantity;
      totalCost += cost;
    });
    var subTotalCostFR = totalCost.toFixed(2).replace('.', ',');
    $('.subtotal .cost').text(subTotalCostFR + ' €');
    var taxRate = parseFloat($('.tax .price').text()
      .replace(',', '.').replace(/[^\d\.]/g, '')) / 100;
    var tax = Math.ceil(totalCost * taxRate * 100) / 100;
    var taxFR = tax.toFixed(2).replace('.', ',');
    $('.tax .cost').text(taxFR + ' €');
    totalCost += tax;
    $('.shipping .quantity').text(String(totalQuantity));
    var shippingRate = parseFloat($('.shipping .price').text()
      .replace(',', '.').replace(/[^\d\.]/g, ''));
    var shipping = totalQuantity * shippingRate;
    var shippingFR = shipping.toFixed(2).replace('.', ',');
    $('.shipping .cost').text(shippingFR + ' €');
    totalCost += shipping;
    var totalCostFR = totalCost.toFixed(2).replace('.', ',');
    $('.total .cost').text(totalCostFR + ' €');
  });

  $('<th> </th>')
  .insertAfter('#cart thead th:nth-child(2)');
  $('#cart tbody tr').each(function() {
    $deleteButton = $('<img />').attr({
      'width': '16',
      'height': '16',
      'src': '../images/cross.png',
      'alt': 'retirer du panier',
      'title': 'retirer du panier',
      'class': 'clickable'
    }).click(function() {
      $(this).parents('tr').find('td.quantity input')
        .val(0).trigger('change')
      .end().hide();
      stripe();
    });
    $('<td></td>')
    .insertAfter($('td:nth-child(2)', this))
    .append($deleteButton);
  });
  $('<td> </td>')
  .insertAfter('#cart tfoot td:nth-child(2)');
});
```

```
$(document).ready(function() {
  var editShipping = function() {
    $.get('shipping.php', function(data) {
      $('#shipping-name').remove();
      $(data).hide().appendTo('#shipping').slideDown();
      $('#shipping form').submit(saveShipping);
    });
    return false;
  };
  var saveShipping = function() {
    var postData = $(this).serialize();
    $.post('shipping.php', postData, function(data) {
      $('#shipping form').remove();
      $(data).appendTo('#shipping');
      $('#shipping-name').click(editShipping);
    });
    return false;
  };
  $('#shipping-name').click(editShipping);
});
```

8.4 En résumé

Dans ce chapitre, nous avons proposé des solutions pour améliorer l'aspect et le comportement des éléments de formulaires HTML. Nous avons appris à appliquer des styles aux formulaires tout en conservant leur balisage sémantique d'origine, à masquer et à afficher de manière conditionnelle des champs en fonction de la valeur d'autres champs, et à valider le contenu des champs avant la soumission du formulaire et pendant la saisie des données. Nous avons examiné d'autres fonctionnalités, comme l'autocomplétion AJAX pour les champs de texte, l'autorisation de la saisie de certains caractères dans un champ et le calcul sur des valeurs numériques dans des champs. Nous avons également vu comment soumettre des formulaires avec AJAX, sans actualisation de la page.

L'élément `form` constitue souvent le cœur d'un site interactif. Grâce à jQuery, nous pouvons facilement améliorer le processus de remplissage des formulaires, tout en conservant leur utilité et leur souplesse.

9

Carrousels et prompteurs

Au sommaire de ce chapitre

- Prompteur de nouvelles
- Carrousel d'images
- En résumé

Nous avons étudié plusieurs façons de masquer les informations lorsqu'elles ne sont pas utiles et de les révéler à la demande, notamment les accordéons escamotables. Néanmoins, ces solutions étaient relativement élémentaires et nous souhaitons parfois que l'affichage et le masquage des informations se fassent avec plus de classe. Parmi ces animations élaborées, on trouve entre autres les *carrousels*, les *diaporamas* et les *prompteurs*. Elles ont toutes en commun une mise en œuvre accrocheuse et attrayante pour passer rapidement d'un élément d'information à un autre.

Dans ce troisième et dernier chapitre de mise en application, nous allons explorer ces animations et les combiner à des techniques AJAX et des styles CSS pour épater l'internaute.

Nous détaillerons deux exemples : un prompteur de nouvelles et un carrousel d'images. Ils nous permettront d'apprendre à mettre en œuvre les points suivants :

- animer la position d'un élément ;
- parser des documents XML ;
- préparer des effets de style élaborés fondés sur une opacité partielle ;
- obtenir des informations provenant de sources différentes ;
- créer un élément d'interface utilisateur pour le défilement horizontal ;
- composer des calques pour des badges et des superpositions ;
- effectuer un zoom sur une image.

9.1 Prompteur de nouvelles

Pour notre premier exemple, un *prompteur* de nouvelles, nous allons créer un flux d'actualité et afficher chaque titre, ainsi qu'un résumé de l'article, par défilement. Le texte sera affiché progressivement, le restera pendant un certain temps pour que l'utilisateur puisse le lire, puis glissera hors de la page comme s'il était écrit sur un tambour tournant en boucle sur la page.

Préparer la page

La mise en œuvre d'une version basique de cette fonctionnalité n'est pas très complexe. Toutefois, en faire une version de production exige un certain doigté.

Comme toujours, nous commençons par un contenu HTML. Le flux d'actualité sera placé dans la barre latérale de la page :

```
<h3>Actualités</h3>
<div id="news-feed">
  <a href="news/index.html">Nouvelles versions</a>
</div>
```

Pour le moment, la zone qui affiche le flux d'actualité contient un seul lien vers la page principale des nouvelles (voir Figure 9.1).

Figure 9.1
Aspect initial du flux d'actualité.

Il s'agit de notre scénario de *dégradation élégante*, pour le cas où l'utilisateur n'aurait pas activé JavaScript. Le contenu que nous manipulerons proviendra d'un *flux RSS* réel.

Les règles CSS pour cet élément `<div>` sont importantes, car elles déterminent non seulement la partie de chaque article qui sera affichée en même temps, mais également l'emplacement des éléments d'actualité sur la page. Voici les règles CSS pour la zone d'affichage des nouvelles et pour chaque article :

```
#news-feed {
  position: relative;
  height: 200px;
  width: 17em;
  overflow: hidden;
}
```

```
.headline {
  position: absolute;
  height: 200px;
  top: 210px;
  overflow: hidden;
}
```

Vous noterez que la hauteur de la zone contenant une nouvelle (représentée par la classe `headline`) et celle du conteneur global sont toutes deux égales à `200px`. Par ailleurs, puisque les éléments `headline` sont positionnés de manière absolue relativement à `#news-feed`, nous sommes en mesure d'aligner le bord supérieur de la nouvelle avec le bord inférieur du conteneur. Ainsi, en fixant la propriété `overflow` de `#news-feed` à `hidden`, les nouvelles sont initialement masquées.

Il existe une autre raison de fixer la propriété `position` des nouvelles à `absolute` : pour que l'emplacement d'un élément puisse être animé sur la page, son positionnement doit être `absolute` ou `relative`, à la place du positionnement `static` par défaut.

Le balisage HTML et les règles CSS étant en place, nous pouvons injecter les titres d'actualité provenant d'un flux RSS. Tout d'abord, nous plaçons le code dans une méthode `.each()` qui joue le rôle d'une sorte d'instruction `if` et concentre le code dans un *espace de noms* privé :

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
  });
});
```

Habituellement, la méthode `.each()` est employée pour itérer sur une collection d'éléments potentiellement vaste. Mais, ici, le sélecteur `#news-feed` correspond à un identifiant et seules deux actions sont possibles. La fonction `$()` crée un objet jQuery qui correspond à un élément unique dont l'identifiant est `news-feed`, ou, si aucun élément de la page ne possède cet identifiant, elle produit un objet jQuery vide. L'appel à `.each()` exécute donc le code associé si, et seulement si, l'objet jQuery n'est pas vide.

Au début de la boucle `.each()`, le conteneur des nouvelles est vidé afin qu'il soit prêt à recevoir un nouveau contenu (voir Figure 9.2).

Obtenir le flux

Pour lire le flux de nouvelles, nous utilisons la méthode `$.get()`, qui fait partie des nombreuses fonctions AJAX de jQuery permettant de communiquer avec le serveur. Nous l'avons vu, cette méthode permet de manipuler du contenu fourni par une source distante en définissant un *gestionnaire de réussite*. Le contenu du flux est passé à ce gestionnaire sous forme d'une structure XML. Nous pouvons ensuite employer le moteur de sélection de jQuery pour traiter ces données :

Figure 9.2

Initialement, le prompteur de nouvelles est vide.

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
    $.get('news/feed.xml', function(data) {
      $('rss item', data).each(function() {
        // Manipuler les nouvelles.
      });
    });
  });
});
```

INFO

Pour de plus amples informations concernant `$.get()` et les autres méthodes AJAX, consultez le Chapitre 6.

Nous devons à présent combiner les parties de chaque article en un bloc HTML utilisable. Nous pouvons à nouveau parcourir les articles du flux avec `.each()` et construire des liens pour les titres :

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
    $.get('news/feed.xml', function(data) {
      $('rss item', data).each(function() {
        var $link = $('<a></a>')
          .attr('href', $('link', this).text())
          .text($('title', this).text());
        var $headline = $('<h4></h4>').append($link);

        $('<div></div>')
          .append($headline)
          .appendTo($container);
      });
    });
  });
});
```

Nous récupérons le contenu des éléments `<title>` et `<link>` de chaque nouvelle et l'utilisons pour construire l'élément `<a>`. Ce lien est enveloppé dans un élément `<h4>`. Nous plaçons chaque nouvelle dans `<div id="news-feed">`, mais, pour le moment, sans appliquer la classe `headline` au `<div>` de la nouvelle afin de pouvoir suivre la progression de notre travail (voir Figure 9.3).

Figure 9.3
Ajout des titres des nouvelles.

Outre les titres d'actualité, nous souhaitons afficher quelques informations concernant chaque nouvelle, notamment sa date de publication et son résumé :

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
    $.get('news/feed.xml', function(data) {
      $('rss item', data).each(function() {
        var $link = $('<a></a>')
          .attr('href', $('link', this).text())
          .text($('title', this).text());
        var $headline = $('<h4></h4>').append($link);

        var pubDate = new Date(
          $('pubDate', this).text());
        var pubMonth = pubDate.getMonth() + 1;
        var pubDay = pubDate.getDate();
        var pubYear = pubDate.getFullYear();
        var $publication = $('<div></div>')
          .addClass('publication-date')
          .text(pubDay + '/' + pubMonth + '/' + pubYear);
        var $summary = $('<div></div>')
          .addClass('summary')
          .html($('description', this).text());

        $('<div></div>')
          .append($headline, $publication, $summary)
          .appendTo($container);
      });
```

```
      });
    });
});
```

Dans le flux RSS, les informations de date sont données au format RFC 822, avec la date, l'heure et le fuseau horaire (par exemple `Sun, 28 Sep 2008 18:01:55 +0000`). Puisque ce format n'est pas particulièrement agréable à lire, nous utilisons l'objet JavaScript `Date` pour produire une date plus concise (par exemple `28/9/2008`).

Le résumé de l'article est plus facile à obtenir et à mettre en forme. Toutefois, nous devons signaler que, dans notre flux d'exemple, ce contenu peut inclure des entités HTML. Pour être certain que jQuery ne leur applique pas automatiquement l'échappement, le résumé est ajouté avec la méthode `.html()` à la place de la méthode `.text()`.

Les nouveaux éléments créés sont placés dans le document par la méthode `.append()`. Nous utilisons une nouvelle fonctionnalité de cette méthode : lorsque plusieurs arguments sont passés, ils sont tous ajoutés à la suite (voir Figure 9.4).

Figure 9.4

Ajout de la date de publication et du résumé.

Vous le constatez, le titre, la date de publication, le lien et le résumé de chaque nouvelle sont à présent en place. Avant de pouvoir les animer, il nous reste à appliquer la classe `headline` avec `.addClass('headline')`, ce qui masquera les nouvelles en raison des règles CSS définies précédemment.

Préparer le prompteur

Puisque les articles d'actualité visibles changeront au fil du temps, nous devons mettre en place un mécanisme qui permet de savoir lesquels sont visibles et ceux qui ne le sont pas. Nous commençons par définir deux variables, la première pour la nouvelle visible, la seconde pour celle qui vient de disparaître. Leur valeur initiale est `0` :

```
var currentHeadline = 0, oldHeadline = 0;
```

Nous nous occupons également du positionnement initial des titres. Pour rappel, la feuille de style définie affecte à la propriété `top` des nouvelles une valeur de dix pixels supérieure à la hauteur du conteneur. Puisque la propriété `overflow` du conteneur est fixée à `hidden`, les nouvelles sont initialement masquées. Si nous enregistrons cette propriété dans une variable, nous facilitons le déplacement ultérieur des articles à cette position :

```
var hiddenPosition = $container.height() + 10;
```

Nous souhaitons également que le premier article soit visible dès le chargement de la page. Pour cela, nous fixons sa propriété `top` à `0`.

```
$('div.headline').eq(currentHeadline).css('top', 0);
```

La Figure 9.5 présente le prompteur dans son état initial.

Figure 9.5

Le prompteur après le chargement de la page.

Enfin, nous enregistrons le nombre total de nouvelles, pour l'utiliser ultérieurement, et définissons une variable de temporisation, qui servira à la mise en place d'une pause entre chaque rotation :

```
var headlineCount = $('div.headline').length;
var pause;
```

Pour le moment, il est inutile de donner une valeur à `pause` ; elle sera fixée au cours de la rotation. Néanmoins, nous pouvons toujours déclarer des variables locales avec `var` de manière à éviter les conflits avec des variables globales éponymes.

La fonction de rotation des nouvelles

Nous sommes prêts à faire défiler les titres, les dates et les résumés. Nous allons pour cela définir une fonction qui nous permettra d'effectuer facilement cette opération chaque fois que nous en aurons besoin.

Tout d'abord, nous mettons à jour les variables qui servent au suivi de la nouvelle active. L'*opérateur modulo* (`%`) nous permet de boucler sur le nombre d'articles. Nous pouvons ajouter un à la valeur de `currentHeadline` lors de chaque appel à notre fonction, puis effectuer le modulo entre cette valeur et celle de `headlineCount` de manière à conserver des numéros de nouvelles valides.

INFO

Au Chapitre 7, nous avons employé la même technique pour la couleur des bandes appliquées aux tables.

Nous devons également actualiser `oldHeadline` afin de pouvoir manipuler facilement l'article qui disparaît :

```
var headlineRotate = function() {
  currentHeadline = (oldHeadline + 1) % headlineCount;
  // Animer l'emplacement de la nouvelle.
  oldHeadline = currentHeadline;
};
```

Passons à présent au code qui fait défiler les nouvelles. Tout d'abord, nous ajoutons une animation qui fait disparaître l'ancien article. Ensuite, nous ajoutons une autre animation qui fait apparaître le suivant :

```
var headlineRotate = function() {
  currentHeadline = (oldHeadline + 1) % headlineCount;
  $('div.headline').eq(oldHeadline).animate(
    {top: -hiddenPosition}, 'slow', function() {
      $(this).css('top', hiddenPosition);
    });
  $('div.headline').eq(currentHeadline).animate(
    {top: 0}, 'slow', function() {
      pause = setTimeout(headlineRotate, 5000);
    });
  oldHeadline = currentHeadline;
};
```

Dans les deux cas, l'animation se fait sur la propriété `top` de la nouvelle. Les nouvelles sont masquées car leur propriété `top` est fixée à la valeur `hiddenPosition`, qui est un nombre supérieur à la hauteur du conteneur. En animant cette propriété jusqu'à la valeur `0`, nous affichons progressivement un article. En poursuivant l'animation jusqu'à la valeur `-hiddenPosition`, nous le faisons disparaître.

INFO

Au Chapitre 4, nous avons vu que la propriété CSS `top` a un effet uniquement lorsque le positionnement de l'élément est `absolute` ou `relative`.

Dans les deux cas, nous indiquons également une fonction de rappel invoquée à la fin de l'animation. Lorsque l'ancien article a complètement disparu, sa propriété `top` est réinitialisée à la valeur `hiddenPosition` afin qu'il soit prêt à réapparaître ultérieurement. Lorsque l'animation du nouvel article est terminée, nous préparons la transition suivante. Pour cela, nous appelons la fonction JavaScript `setTimeout()`, qui définit la fonction qui sera invoquée après une certaine durée. Dans ce cas, nous déclenchons l'appel à `headlineRotate()` après une pause de cinq secondes (5 000 millisecondes).

Nous avons créé un cycle d'activité. Lorsqu'une animation est terminée, la suivante est prête à démarrer. Il nous reste à appeler une première fois la fonction. Pour cela, nous utilisons à nouveau `setTimeout()` de manière à déclencher la première rotation cinq secondes après la lecture du flux RSS. Notre prompteur de nouvelles est à présent opérationnel :

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
    $.get('news/feed.xml', function(data) {
      $('rss item', data).each(function() {
        var $link = $('<a></a>')
          .attr('href', $('link', this).text())
          .text($('title', this).text());
        var $headline = $('<h4></h4>').append($link);
        var pubDate = new Date($('pubDate', this).text());
        var pubMonth = pubDate.getMonth() + 1;
        var pubDay = pubDate.getDate();
        var pubYear = pubDate.getFullYear();
        var $publication = $('<div></div>')
          .addClass('publication-date')
          .text(pubDay + '/' + pubMonth + '/' + pubYear);
        var $summary = $('<div></div>')
          .addClass('summary')
          .html($('description', this).text());
        $('<div></div>')
          .addClass('headline')
          .append($headline, $publication, $summary)
          .appendTo($container);
      });
      var currentHeadline = 0, oldHeadline = 0;
      var hiddenPosition = $container.height() + 10;
      $('div.headline').eq(currentHeadline).css('top', 0);
      var headlineCount = $('div.headline').length;
      var pause;
      var headlineRotate = function() {
        currentHeadline = (oldHeadline + 1) % headlineCount;
        $('div.headline').eq(oldHeadline).animate(
          {top: -hiddenPosition}, 'slow', function() {
            $(this).css('top', hiddenPosition);
          });
        $('div.headline').eq(currentHeadline).animate(
          {top: 0}, 'slow', function() {
```

```
            pause = setTimeout(headlineRotate, 5000);
          });
        oldHeadline = currentHeadline;
      };
      pause = setTimeout(headlineRotate, 5000);
    });
  });
});
```

Durant l'animation, nous pouvons constater que la partie supérieure de l'ancien article est coupée, tandis que le nouveau voit sa partie inférieure tronquée (voir Figure 9.6).

Figure 9.6

Transition entre l'affichage de deux nouvelles.

Effectuer une pause lors du survol

Bien que le prompteur soit pleinement opérationnel, il nous reste à résoudre un problème d'utilisabilité : un article peut disparaître de la zone d'affichage avant que l'utilisateur ait pu cliquer sur l'un de ses liens. Il est alors obligé d'attendre le défilement complet de toutes les nouvelles avant d'avoir une seconde chance. Pour éviter ce genre de désagréments, nous pouvons faire en sorte que le prompteur fasse une pause lorsque le pointeur de la souris survole une nouvelle :

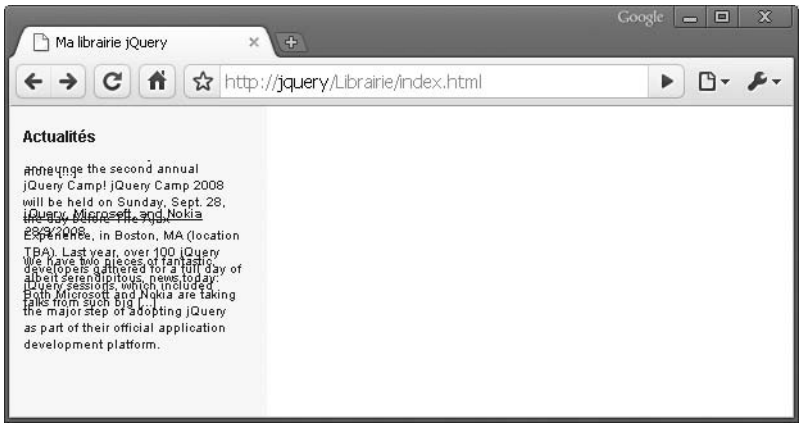
```
$container.hover(function() {
  clearTimeout(pause);
}, function() {
  pause = setTimeout(headlineRotate, 250);
});
```

Lorsque la souris entre dans la zone d'affichage de l'article, le premier gestionnaire `.hover()` invoque la fonction JavaScript `clearTimeout()`. Elle annule la temporisation en cours, empêchant l'appel à `headlineRotate()`. Lorsque la souris quitte la zone d'affichage, le second gestionnaire de `.hover()` remet en place la temporisation, déclenchant un appel à `headlineRotate()` après un délai de 250 millisecondes.

Le code semble fonctionner parfaitement. Toutefois, si l'utilisateur déplace rapidement et de manière répétée le pointeur de la souris sur et hors du `<div>`, un effet désagréable se produit. Plusieurs nouvelles sont animées en même temps et se superposent dans la zone d'affichage (voir Figure 9.7).

Figure 9.7

Un effet indésirable dû aux pauses.



Pour supprimer ce comportement, nous devons intervenir au cœur du code. Avant la fonction `headlineRotate()`, nous ajoutons une variable supplémentaire :

```
var rotateInProgress = false;
```

Dès la première ligne de la fonction, nous vérifions si une rotation est en cours. Il faut que la valeur de `rotateInProgress` soit à `false` pour que nous puissions exécuter à nouveau le code. C'est pourquoi nous plaçons tout le contenu de la fonction dans une instruction `if`. Dès le début de ce code conditionnel, nous fixons la variable `rotateInProgress` à `true` et, dans la fonction de rappel de la seconde méthode `.animate()`, nous la remettons à `false` :

```
var headlineRotate = function() {
  if (!rotateInProgress) {
    rotateInProgress = true;
    currentHeadline = (oldHeadline + 1) % headlineCount;
    $('div.headline').eq(oldHeadline).animate(
      {top: -hiddenPosition}, 'slow', function() {
        $(this).css('top', hiddenPosition);
      });
    $('div.headline').eq(currentHeadline).animate(
      {top: 0}, 'slow', function() {
        rotateInProgress = false;
        pause = setTimeout(headlineRotate, 5000);
      });
    oldHeadline = currentHeadline;
  }
};
```

Ces quelques lignes supplémentaires améliorent considérablement notre prompteur. Un survol rapide et répété de la zone ne provoque plus la superposition des nouvelles. Néanmoins, un tel comportement de l'utilisateur révèle un problème tenace : le prompteur déclenche deux ou trois animations immédiatement l'une après l'autre au lieu de les séparer de cinq secondes.

Ce problème vient du fait qu'une temporisation peut s'activer pendant que l'utilisateur déplace le pointeur de la souris hors du `<div>` et avant que la temporisation en cours ne soit écoulée. Nous devons donc mettre en place une nouvelle protection, en utilisant la variable `pause` pour signaler l'imminence d'une autre animation. Pour cela, nous fixons cette variable à `false` lorsque la temporisation est annulée ou lorsqu'elle est terminée. Nous pouvons à présent tester la valeur de la variable `pause` pour nous assurer qu'aucune minuterie n'est active avant d'en déclencher une nouvelle :

```
var headlineRotate = function() {
  if (!rotateInProgress) {
    rotateInProgress = true;
    pause = false;
    currentHeadline = (oldHeadline + 1) % headlineCount;
    $('div.headline').eq(oldHeadline).animate(
      {top: -hiddenPosition}, 'slow', function() {
        $(this).css('top', hiddenPosition);
      });
    $('div.headline').eq(currentHeadline).animate(
      {top: 0}, 'slow', function() {
        rotateInProgress = false;
        if (!pause) {
          pause = setTimeout(headlineRotate, 5000);
        }
      });
    oldHeadline = currentHeadline;
  }
};
if (!pause) {
  pause = setTimeout(headlineRotate, 5000);
}
$container.hover(function() {
  clearTimeout(pause);
  pause = false;
}, function() {
  if (!pause) {
    pause = setTimeout(headlineRotate, 250);
  }
});
```

Désormais, notre prompteur de nouvelles peut encaisser sans broncher toutes les tentatives de perturbation menées par l'utilisateur.

Lire un flux provenant d'un domaine différent

Le flux de nouvelles utilisé dans notre exemple est un fichier local, mais nous pourrions vouloir lire un flux provenant d'un autre site. Nous l'avons vu au Chapitre 6, les requêtes AJAX ne peuvent pas, de manière générale, cibler un site autre que celui qui héberge la page consultée. Dans ce chapitre, nous avons présenté le format de données JSONP en tant que méthode permettant de contourner cette limitation. Dans cette section, nous supposons que nous ne pouvons pas agir sur la source de données et qu'une solution différente doit donc être imaginée.

Pour accéder au fichier *via* AJAX, nous écrivons du code côté serveur qui sert de *relais*, ou *proxy*, pour la requête et fait ainsi croire au code JavaScript côté client que le fichier XML se trouve sur le serveur alors qu'il réside ailleurs. Nous écrivons un petit script PHP pour extraire le contenu des nouvelles depuis notre serveur et retransmettre ces données vers le script jQuery demandeur. Ce script, nommé `feed.php`, peut être invoqué comme nous l'avons fait précédemment pour `feed.xml` :

```
$.get('news/feed.php', function(data) {
  // ...
});
```

Dans le fichier `feed.php`, nous extrayons le contenu du flux d'actualité à partir du site distant, puis l'affichons en sortie du script :

```
<?php
  header('Content-Type: text/xml');
  print file_get_contents('http://jquery.com/blog/feed');
?>
```

Nous devons indiquer explicitement que le contenu de la page est de type `text/xml`. En effet, cela permet à jQuery de le récupérer et de l'analyser comme s'il s'agissait d'un document XML statique normal.

INFO

Certains hébergeurs web interdisent l'utilisation de la fonction PHP `file_get_contents()` pour accéder à des fichiers distants car cela pose des problèmes de sécurité. Dans ce cas, d'autres solutions peuvent être disponibles, comme la bibliothèque cURL (**http://wiki.dreamhost.com/CURL**).

Ajouter un indicateur de chargement

Puisque la lecture d'un fichier distant peut prendre un certain temps, en fonction de différents facteurs, il est préférable d'informer l'utilisateur du chargement en cours. Pour cela, avant d'effectuer notre requête AJAX, nous ajoutons sur la page une image qui sert d'*indicateur de chargement* :

```
var $loadingIndicator = $('<img/>')
  .attr({
    'src': 'images/loading.gif',
    'alt': 'Chargement en cours. Veuillez patienter.'
  })
  .addClass('news-wait')
  .appendTo($container);
```

Ensuite, sur la première ligne de la fonction de rappel de `$.get()` invoquée en cas de réussite, nous retirons l'image de la page par une simple instruction :

```
$loadingIndicator.remove();
```

À présent, lors du chargement de la page, si l'obtention des nouvelles prend un certain temps, l'utilisateur voit l'indicateur de chargement à la place d'une zone vide (voir Figure 9.8).

Figure 9.8
L'indicateur de chargement en cours.

Cette image est un GIF animé qui, une fois affiché par le navigateur web, donne l'impression de tourner pour signaler une activité en cours.

INFO

Vous pouvez facilement créer des images GIF animées servant d'indicateurs de chargement AJAX en utilisant le service proposé par le site **http://ajaxload.info/**.

Effet de fondu en dégradé

Avant de mettre de côté notre exemple de prompteur de nouvelles, nous allons lui ajouter une touche finale : le texte de l'article va s'afficher en fondu en partant du bas du conteneur. Nous voulons obtenir un effet de *fondu en dégradé*, dans lequel le texte est opaque en haut de la zone et transparent en bas.

Toutefois, il n'est pas possible d'appliquer plusieurs opacités sur un même élément de texte. Pour simuler cet effet, nous recouvrons la zone de l'effet par une série d'élé-

ments, chacun ayant sa propre opacité. Ces bandes seront des éléments `<div>` partageant quelques propriétés de style que nous déclarons dans notre fichier CSS :

```
.fade-slice {
  position: absolute;
  width: 20em;
  height: 2px;
  background: #efd;
  z-index: 3;
}
```

Elles ont toutes la largeur et la couleur d'arrière-plan de leur élément conteneur, `<div id="news-feed">`. Cela permet de tromper l'œil de l'utilisateur et de lui faire croire que le texte s'affiche en fondu au lieu d'être recouvert par un autre élément.

Nous créons à présent les éléments `<div class="fade-slice">`. Pour que leur nombre soit correct, nous commençons par déterminer la hauteur en pixels de la zone de l'effet. Dans ce cas, nous choisissons 25 % de la hauteur du `<div id="news-feed">`. Nous utilisons une boucle `for` pour itérer sur la hauteur de cette zone, en créant une nouvelle bande pour chaque segment de deux pixels du dégradé :

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
    var fadeHeight = $container.height() / 4;
    for (var yPos = 0; yPos < fadeHeight; yPos += 2) {
      $('<div></div>')
        .addClass('fade-slice')
        .appendTo($container);
    }
  });
});
```

Nous avons à présent vingt-cinq bandes, une pour chaque segment de deux pixels dans une zone de dégradé de cinquante pixels, mais elles sont toutes empilées par-dessus le conteneur. Pour que notre astuce fonctionne, chacune d'elles doit avoir une position et une opacité différentes. Nous pouvons nous servir de la variable d'itération `yPos` pour déterminer mathématiquement les propriétés `opacity` et `top` de chaque élément :

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();
    var fadeHeight = $container.height() / 4;
    for (var yPos = 0; yPos < fadeHeight; yPos += 2) {
      $('<div></div>').css({
        opacity: yPos / fadeHeight,
        top: $container.height() - fadeHeight + yPos
      }).addClass('fade-slice').appendTo($container);
    }
  });
});
```

Puisque ces calculs sont un peu difficiles à comprendre, nous présentons les valeurs dans un tableau. Les valeurs d'opacité passent progressivement du transparent à l'opaque, tandis que les valeurs de `top` commencent en haut de la zone de fondu (150) et augmentent jusqu'à la hauteur du conteneur (voir Tableau 9.1).

Tableau 9.1 : Les valeurs d'opacité et de positionnement calculées

yPos	*opacity*	*top*
0	0	150
2	0.04	152
4	0.08	154
6	0.12	156
8	0.16	158
	...	
40	0.80	190
42	0.84	192
44	0.88	194
46	0.92	196
48	0.96	198

N'oubliez pas que, le positionnement haut du dernier `<div class="fade-slice">` étant égal à `198`, sa hauteur de deux pixels recouvrira parfaitement les deux pixels inférieurs du `<div>` conteneur haut de deux cents pixels.

Avec le code correspondant en place, le texte dans la zone de l'article présente un joli effet de fondu en dégradé lorsqu'il passe du transparent à l'opaque dans la partie inférieure du conteneur (voir Figure 9.9).

Version finale du code

Notre premier prompteur est à présent terminé. Les nouvelles sont prises sur un serveur distant, mises en forme, affichées progressivement une à une et joliment stylées :

```
$(document).ready(function() {
  $('#news-feed').each(function() {
    var $container = $(this);
    $container.empty();

    var fadeHeight = $container.height() / 4;
    for (var yPos = 0; yPos < fadeHeight; yPos += 2) {
```

Figure 9.9

L'effet de fondu en dégradé sur le texte.


```
  $('<div></div>').css({
    opacity: yPos / fadeHeight,
    top: $container.height() - fadeHeight + yPos
  }).addClass('fade-slice').appendTo($container);
}

var $loadingIndicator = $('<img/>')
  .attr({
    'src': 'images/loading.gif',
    'alt': 'Chargement en cours. Veuillez patienter.'
  })
  .addClass('news-wait')
  .appendTo($container);

$.get('news/feed.php', function(data) {
  $loadingIndicator.remove();
  $('rss item', data).each(function() {
    var $link = $('<a></a>')
      .attr('href', $('link', this).text())
      .text($('title', this).text());
    var $headline = $('<h4></h4>').append($link);

    var pubDate = new Date($('pubDate', this).text());
    var pubMonth = pubDate.getMonth() + 1;
    var pubDay = pubDate.getDate();
    var pubYear = pubDate.getFullYear();
    var $publication = $('<div></div>')
      .addClass('publication-date')
      .text(pubDay + '/' + pubMonth + '/' + pubYear);

    var $summary = $('<div></div>')
      .addClass('summary')
      .html($('description', this).text());

    $('<div></div>')
      .addClass('headline')
      .append($headline, $publication, $summary)
      .appendTo($container);
  });
```

```
        var currentHeadline = 0, oldHeadline = 0;
        var hiddenPosition = $container.height() + 10;
        $('div.headline').eq(currentHeadline).css('top', 0);
        var headlineCount = $('div.headline').length;
        var pause;
        var rotateInProgress = false;

        var headlineRotate = function() {
          if (!rotateInProgress) {
            rotateInProgress = true;
            pause = false;
            currentHeadline = (oldHeadline + 1) % headlineCount;
            $('div.headline').eq(oldHeadline).animate(
              {top: -hiddenPosition}, 'slow', function() {
                $(this).css('top', hiddenPosition);
              });
            $('div.headline').eq(currentHeadline).animate(
              {top: 0}, 'slow', function() {
                rotateInProgress = false;
                if (!pause) {
                  pause = setTimeout(headlineRotate, 5000);
                }
              });
            oldHeadline = currentHeadline;
          }
        };
        if (!pause) {
          pause = setTimeout(headlineRotate, 5000);
        }

        $container.hover(function() {
          clearTimeout(pause);
          pause = false;
        }, function() {
          if (!pause) {
            pause = setTimeout(headlineRotate, 250);
          }
        });
      });
    });
  });
```

9.2 Carrousel d'images

Comme autre exemple d'animation du contenu de la page, nous allons mettre en œuvre une *galerie d'images* pour la page d'accueil de notre librairie. Cette galerie mettra en avant quelques livres, avec pour chacun un lien vers son image de couverture agrandie. Contrairement à l'exemple précédent, où les nouvelles étaient affichées à intervalle régulier, nous utiliserons jQuery pour faire défiler les images sur l'écran en réponse à une action de l'utilisateur.

INFO

Un autre mécanisme permettant de faire défiler un ensemble d'images est mis en œuvre par le plugin jCarousel pour jQuery. Par ailleurs, la grande flexibilité du plugin SerialScroll permet de faire défiler tout type de contenu. Bien que le résultat obtenu avec ces plugins ne soit pas totalement identique à celui de notre galerie, ils permettent de réaliser des effets de haute qualité avec très peu de code. Pour de plus amples informations concernant l'utilisation des plugins, consultez le Chapitre 10.

Préparer la page

Comme toujours, nous commençons par un balisage HTML et des styles CSS pour que les utilisateurs sans JavaScript puissent obtenir une représentation attrayante et opérationnelle des informations :

```
<div id="featured-books">
  <div class="covers">
    <a href="images/covers/large/1847190871.jpg"
      title="Community Server Quickly">
      <img src="images/covers/medium/1847190871.jpg"
        width="120" height="148"
        alt="Community Server Quickly" />
      <span class="price">35,99 €</span>
    </a>
    <a href="images/covers/large/1847190901.jpg"
      title="Deep Inside osCommerce: The Cookbook">
      <img src="images/covers/medium/1847190901.jpg"
        width="120" height="148"
        alt="Deep Inside osCommerce: The Cookbook" />
      <span class="price">44,99 €</span>
    </a>
    <a href="images/covers/large/1847190979.jpg"
      title="Learn OpenOffice.org Spreadsheet Macro Programming:
        OOoBasic and Calc automation">
      <img src="images/covers/medium/1847190979.jpg"
        width="120" height="148"
        alt="Learn OpenOffice.org Spreadsheet Macro Programming:
          OOoBasic and Calc automation" />
      <span class="price">35,99 €</span>
    </a>
    <a href="images/covers/large/1847190987.jpg"
      title="Microsoft AJAX C# Essentials:
        Building Responsive ASP.NET 2.0 Applications">
      <img src="images/covers/medium/1847190987.jpg"
        width="120" height="148"
        alt="Microsoft AJAX C# Essentials:
          Building Responsive ASP.NET 2.0 Applications" />
      <span class="price">31,99 €</span>
    </a>
    <a href="images/covers/large/1847191002.jpg"
      title="Google Web Toolkit GWT Java AJAX Programming">
      <img src="images/covers/medium/1847191002.jpg"
        width="120" height="148"
        alt="Google Web Toolkit GWT Java AJAX Programming" />
```

```
        <span class="price">40,49 €</span>
      </a>
      <a href="images/covers/large/1847192386.jpg"
        title="Building Websites with Joomla! 1.5 Beta 1">
        <img src="images/covers/medium/1847192386.jpg"
          width="120" height="148"
          alt="Building Websites with Joomla! 1.5 Beta 1" />
        <span class="price">40,49 €</span>
      </a>
    </div>
  </div>
```

Chaque image est placée dans une balise d'ancre, qui pointe vers une version agrandie de la couverture. Un prix est également indiqué pour chaque ouvrage ; ils seront initialement cachés et nous utiliserons JavaScript pour les afficher au moment opportun.

De manière à ne pas trop encombrer la page d'accueil, nous souhaitons afficher uniquement trois couvertures à la fois. Lorsque JavaScript n'est pas disponible, nous pouvons obtenir ce résultat en fixant la propriété `overflow` du conteneur à `scroll` et en ajustant sa largeur en conséquence :

```
#featured-books {
  position: relative;
  background: #ddd;
  width: 440px;
  height: 186px;
  overflow: scroll;
  margin: 1em auto;
  padding: 0;
  text-align: center;
  z-index: 2;
}

#featured-books .covers {
  position: relative;
  width: 840px;
  z-index: 1;
}

#featured-books a {
  float: left;
  margin: 10px;
  height: 146px;
}

#featured-books .price {
  display: none;
}
```

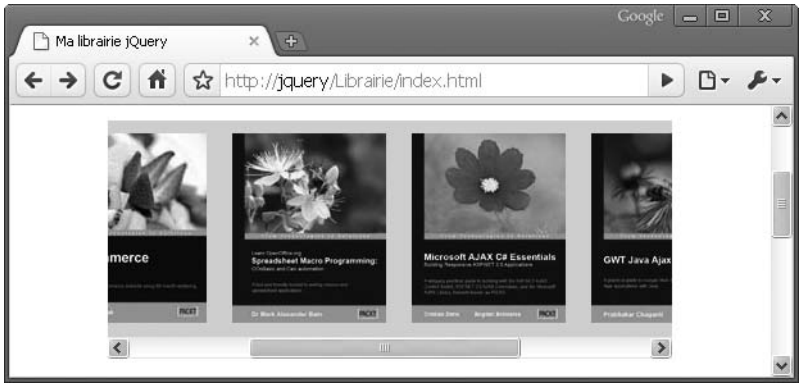
Ces règles de style méritent quelques explications. La valeur de la propriété `z-index` de l'élément le plus extérieur doit être supérieure à celle de l'élément qu'il contient ; ainsi, Internet Explorer peut masquer la partie de l'élément interne qui déborde de son conteneur. Nous fixons la largeur de l'élément externe à `440px` de manière à accepter trois

images, avec une marge de `10px` autour de chacune et un espace supplémentaire de `20px` pour la barre de défilement.

Avec ces styles en place, les images peuvent être parcourues à l'aide de la barre de défilement standard du système (voir Figure 9.10).

Figure 9.10

Présentation standard des livres mis en avant.



Modifier les styles avec JavaScript

Puisque nous avons fait le nécessaire pour que notre galerie fonctionne sans JavaScript, nous allons à présent retirer quelques raffinements. La barre de défilement standard sera redondante lorsque nous mettrons en place notre propre mécanisme, et la disposition automatique des couvertures avec la propriété `float` entrera en conflit avec le positionnement dont nous aurons besoin pour les animer. Par conséquent, notre première tâche consiste à redéfinir certains styles :

```
$(document).ready(function() {
  var spacing = 140;

  $('#featured-books').css({
    'width': spacing * 3,
    'height': '166px',
    'overflow': 'hidden'
  }).find('.covers a').css({
    'float': 'none',
    'position': 'absolute',
    'left': 1000
  });

  var $covers = $('#featured-books .covers a');

  $covers.eq(0).css('left', 0);
  $covers.eq(1).css('left', spacing);
  $covers.eq(2).css('left', spacing * 2);
});
```

La variable `spacing` sera utile dans de nombreux calculs. Elle représente la largeur d'une image de couverture, plus l'espacement ajouté de chaque côté. La largeur de l'élément conteneur peut donc être fixée précisément de manière à présenter trois images, sans l'espace requis par la barre de défilement. Nous fixons évidemment la propriété `overflow` à `hidden` pour faire disparaître la barre de défilement.

Les images de couverture sont toutes positionnées de manière absolue, avec la coordonnée `left` fixée à `1000` pour qu'elles se trouvent hors de la zone visible. Ensuite, nous déplaçons les trois premières couvertures dans la zone visible, une à la fois. La variable `$covers`, qui contient tous les éléments d'ancre, se révélera également pratique ultérieurement.

La Figure 9.11 montre que les trois premières couvertures sont visibles, sans mécanisme de défilement.

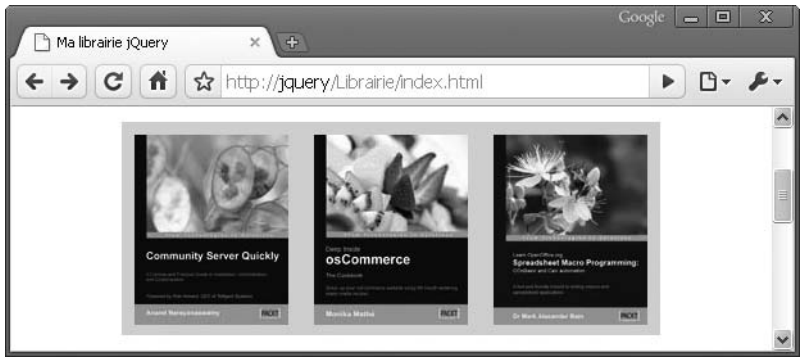


Figure 9.11
Positionnement initial de trois couvertures, sans mécanisme de défilement.

Décaler les images sur un clic

À présent, nous devons ajouter le code qui répond au clic sur l'une des images d'extrémité et réordonne les couvertures en conséquence. Si le clic se fait sur la couverture de gauche, cela signifie que l'utilisateur souhaite voir les autres images qui se trouvent à sa gauche. Autrement dit, nous devons décaler les couvertures vers la droite. À l'inverse, si la couverture de droite est cliquée, nous devons décaler les images vers la gauche. Puisque nous souhaitons un fonctionnement en *carrousel*, les images masquées à gauche sont ajoutées à droite. Pour commencer, nous modifions simplement l'emplacement des images, sans les animer :

```
$(document).ready(function() {
  var spacing = 140;

  $('#featured-books').css({
    'width': spacing * 3,
    'height': '166px',
    'overflow': 'hidden'
  }).find('.covers a').css({
```

```
      'float': 'none',
      'position': 'absolute',
      'left': 1000
    });

    var setUpCovers = function() {
      var $covers = $('#featured-books .covers a');
      $covers.unbind('click');

      // Image de gauche : défilement vers la droite (pour afficher les
      // images qui se trouvent à gauche).
      $covers.eq(0)
        .css('left', 0)
        .click(function(event) {
          $covers.eq(2).css('left', 1000);
          $covers.eq($covers.length - 1).prependTo('#featured-books .covers');
          setUpCovers();
          event.preventDefault();
        });

      // Image de droite : défilement vers la gauche (pour afficher les
      // images qui se trouvent à droite).
      $covers.eq(2)
        .css('left', spacing * 2)
        .click(function(event) {
          $covers.eq(0).css('left', 1000);
          $covers.eq(0).appendTo('#featured-books .covers');
        setUpCovers();
        event.preventDefault();
      });

      // Image au centre.
      $covers.eq(1)
        .css('left', spacing);
    };
    setUpCovers();
  });
```

La nouvelle fonction `setUpCovers()` comprend le code de positionnement de l'image écrit précédemment. En l'encapsulant dans une fonction, nous pouvons renouveler le positionnement des images après que les éléments ont été réordonnés ; cette possibilité est importante, comme nous le verrons plus loin.

Notre exemple se fonde sur six couvertures au total, référencées par les nombres 0 à 5 en JavaScript, et les images 0, 1 et 2 sont visibles. Lors d'un clic sur l'image 0, nous voulons décaler toutes les images d'une position vers la droite. Nous sortons l'image 2 de la zone visible (avec `.css('left', 1000)`), puisqu'elle ne doit plus être affichée après le décalage. Ensuite, nous déplaçons l'image 5 située en fin de liste vers la première place (avec `.prependTo()`). Cette opération réordonne toutes les images et, lors d'un appel suivant à `setUpCovers()`, l'ancienne image 5 est devenue l'image 0, la n° 0 est devenue la n° 1, et la n° 1 est devenue la n° 2. Le code de positionnement mis en œuvre par cette fonction suffit donc à déplacer les couvertures à leur nouvel emplacement (voir Figure 9.12).

Figure 9.12

Déplacement des couvertures suite au clic sur l'image à l'extrême droite.

En cliquant sur l'image 2, la procédure est inversée. Cette fois-ci, l'image 0 est masquée, puis déplacée à la fin de la file. Cela décale la n° 1 à l'emplacement 0, la n° 2, à l'emplacement 1, et la n° 3, à l'emplacement 2.

Pour éviter des anomalies dans les actions de l'utilisateur, nous devons faire attention à deux points :

1. Nous devons appeler `.preventDefault()` dans notre gestionnaire de `click`, car les couvertures sont placées dans des liens menant à une version agrandie. Si nous n'invoquons pas cette fonction, le lien est suivi et l'effet n'est pas déclenché.
2. Nous devons délier tous les gestionnaires de `click` au début de la fonction `setUpCovers()` ou nous finirions par avoir plusieurs gestionnaires liés à la même image suite à la rotation du carrousel.

Ajouter une animation de glissement

Il peut être difficile de comprendre ce qui se passe lors d'un clic sur l'image. En effet, les couvertures se décalent instantanément et l'utilisateur risque de penser qu'elles ont simplement changé au lieu d'être déplacées. Pour résoudre ce problème, nous ajoutons une animation qui fait glisser les couvertures au lieu qu'elles apparaissent simplement à leur nouvel emplacement. Pour cela, nous devons revoir la fonction `setUpCovers()` :

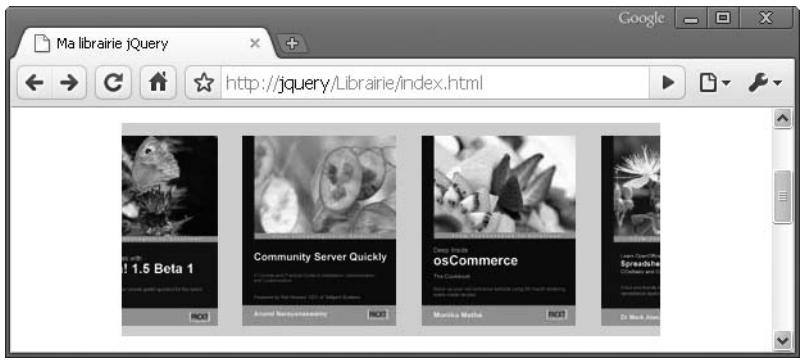
```
var setUpCovers = function() {
  var $covers = $('#featured-books .covers a');
  $covers.unbind('click');

  // Image de gauche : défilement vers la droite (pour afficher les
  // images qui se trouvent à gauche).
  $covers.eq(0)
    .css('left', 0)
    .click(function(event) {
      $covers.eq(0).animate({'left': spacing}, 'fast');
      $covers.eq(1).animate({'left': spacing * 2}, 'fast');
      $covers.eq(2).animate({'left': spacing * 3}, 'fast');
```

```
        $covers.eq($covers.length - 1)
          .css('left', -spacing)
          .animate({'left': 0}, 'fast', function() {
            $(this).prependTo('#featured-books .covers');
            setUpCovers();
          });
        event.preventDefault();
      });

    // Image de droite : défilement vers la gauche (pour afficher les
    // images qui se trouvent à droite).
    $covers.eq(2)
      .css('left', spacing * 2)
      .click(function(event) {
        $covers.eq(0)
          .animate({'left': -spacing}, 'fast', function() {
            $(this).appendTo('#featured-books .covers');
            setUpCovers();
          });
        $covers.eq(1).animate({'left': 0}, 'fast');
        $covers.eq(2).animate({'left': spacing}, 'fast');
        $covers.eq(3)
          .css('left', spacing * 3)
          .animate({'left': spacing * 2}, 'fast');
        event.preventDefault();
      });

    // Image au centre.
    $covers.eq(1)
      .css('left', spacing);
  };
```

Suite au clic sur l'image de gauche, nous déplaçons les trois images visibles vers la droite en les décalant d'une largeur d'image (par réutilisation de la variable `spacing` définie précédemment). Cette partie est simple, mais nous devons également faire en sorte que la nouvelle image devienne visible. Pour cela, nous prenons celle qui se trouve à la fin de la file et ajustons sa position à l'écran pour qu'elle se trouve hors de la zone visible sur le côté gauche (`-spacing`). Ensuite, nous la faisons glisser vers la droite avec les autres éléments (voir Figure 9.13).

Figure 9.13

Glissement des couvertures vers la droite.



Bien que l'animation tienne compte du déplacement initial, nous devons toujours ajuster l'ordre des couvertures en invoquant à nouveau `setUpCovers()`. Dans le cas contraire, le clic suivant ne fonctionnerait pas correctement. Puisque `setUpCovers()` modifie la position des couvertures, nous devons reporter son invocation à la fin de l'animation. C'est pourquoi nous la plaçons dans la fonction de rappel de l'animation.

Un clic sur l'image à l'extrême droite réalise une animation semblable, mais inversée. Cette fois-ci, l'image à l'extrême gauche quitte la zone visible et est déplacée à la fin de la file avant l'appel à `setUpCovers()` lorsque l'animation est terminée. La nouvelle image de droite est, quant à elle, mise à son nouvel emplacement (`spacing * 3`) avant que son animation ne débute.

Afficher des icônes d'action

Notre carrousel d'images tourne parfaitement, mais rien n'indique à l'utilisateur qu'il peut cliquer sur les couvertures pour les faire défiler. Nous pouvons l'aider en affichant des icônes appropriées lorsque le pointeur de la souris survole les images.

Dans notre exemple, les icônes seront placées par-dessus les images existantes. En donnant une valeur adéquate à la propriété `opacity`, la couverture placée en dessous sera toujours visible même lorsque l'icône sera affichée. Nous choisissons des icônes monochromes afin que la couverture ne soit pas trop assombrie (voir Figure 9.14).

Figure 9.14

Les icônes de représentation des actions.

Nous avons besoin de trois icônes, deux associées aux couvertures de gauche et de droite qui indiquent à l'utilisateur qu'il peut faire défiler les images, et une pour la couverture centrale afin de lui signaler qu'il peut obtenir une version agrandie de l'image. Nous créons des éléments HTML qui font référence aux icônes et les enregistrons dans des variables en vue de leur utilisation ultérieure :

```
var $leftRollover = $('<img/>')
  .attr('src', 'images/left.gif')
  .addClass('control')
  .css('opacity', 0.6)
  .css('display', 'none');
var $rightRollover = $('<img/>')
  .attr('src', 'images/right.gif')
  .addClass('control')
  .css('opacity', 0.6)
  .css('display', 'none');
```

```
var $enlargeRollover = $('<img/>')
  .attr('src', 'images/enlarge.gif')
  .addClass('control')
  .css('opacity', 0.6)
  .css('display', 'none');
```

Vous aurez évidemment remarqué la redondance dans ce code. Pour la réduire, nous factorisons les parties redondantes dans une fonction, que nous appellerons pour créer chaque icône requise :

```
function createControl(src) {
  return $('<img/>')
    .attr('src', src)
    .addClass('control')
    .css('opacity', 0.6)
    .css('display', 'none');
}

var $leftRollover = createControl('images/left.gif');
var $rightRollover = createControl('images/right.gif');
var $enlargeRollover = createControl('images/enlarge.gif');
```

Dans les règles de style, nous fixons la propriété `z-index` de ces contrôles à une valeur supérieure à celle des images et nous les positionnons de manière absolue afin qu'elles se superposent aux couvertures :

```
#featured-books .control {
  position: absolute;
  z-index: 3;
  left: 0;
  top: 0;
}
```

Puisque les trois icônes possèdent la même classe `control`, nous pourrions être tentés de fixer la propriété `opacity` dans la feuille de style CSS. Malheureusement, l'opacité d'un élément n'est pas prise en charge de la même manière dans tous les navigateurs. Ainsi, dans Internet Explorer, pour indiquer une opacité de 60 %, il faut écrire `filter: alpha(opacity=60)`. Au lieu de nous empêtrer dans la gestion de toutes ces distinctions, nous fixons l'opacité à l'aide de la méthode `.css()` de jQuery, qui s'occupera de tous ces détails.

À présent, dans les gestionnaires de `hover`, nous devons simplement placer les images au bon endroit dans le DOM et les afficher :

```
var setUpCovers = function() {
  var $covers = $('#featured-books .covers a');

  $covers.unbind('click mouseenter mouseleave');

  // Image de gauche : défilement vers la droite (pour afficher les
  // images qui se trouvent à gauche).
  $covers.eq(0)
    .css('left', 0)
```

```
      .click(function(event) {
        $covers.eq(0).animate({'left': spacing}, 'fast');
        $covers.eq(1).animate({'left': spacing * 2}, 'fast');
        $covers.eq(2).animate({'left': spacing * 3}, 'fast');
        $covers.eq($covers.length - 1)
          .css('left', -spacing)
          .animate({'left': 0}, 'fast', function() {
            $(this).prependTo('#featured-books .covers');
            setUpCovers();
          });
        event.preventDefault();
      }).hover(function() {
        $leftRollover.appendTo(this).show();
      }, function() {
        $leftRollover.hide();
      });

    // Image de droite : défilement vers la gauche (pour afficher les
    // images qui se trouvent à droite).
    $covers.eq(2)
      .css('left', spacing * 2)
      .click(function(event) {
        $covers.eq(0)
          .animate({'left': -spacing}, 'fast', function() {
            $(this).appendTo('#featured-books .covers');
            setUpCovers();
          });
        $covers.eq(1).animate({'left': 0}, 'fast');
        $covers.eq(2).animate({'left': spacing}, 'fast');
        $covers.eq(3)
          .css('left', spacing * 3)
          .animate({'left': spacing * 2}, 'fast');
        event.preventDefault();
      }).hover(function() {
        $rightRollover.appendTo(this).show();
      }, function() {
        $rightRollover.hide();
      });

    // Image au centre.
    $covers.eq(1)
      .css('left', spacing)
      .hover(function() {
        $enlargeRollover.appendTo(this).show();
      }, function() {
        $enlargeRollover.hide();
      });
  };
```

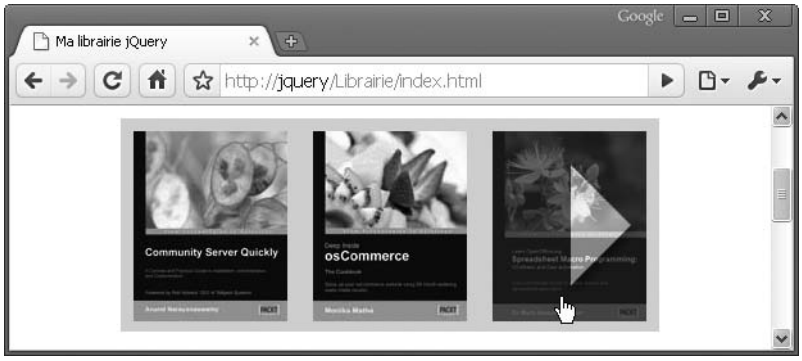
Comme précédemment pour `click`, nous commençons par délier les gestionnaires de `mouseenter` et `mouseleave` au début de la fonction `setUpCovers()` afin que les comportements associés au survol ne se cumulent pas. Dans ce code, nous exploitons une autre possibilité de la méthode `.unbind()` : les gestionnaires de différents événements peuvent être déliés en même temps en séparant les noms des événements par des espaces.

Pourquoi `mouseenter` et `mouseleave` ? Lorsque nous invoquons la méthode `.hover()`, jQuery transforme cet appel en deux liaisons d'événements. La première fonction fournie est associée au gestionnaire de l'événement `mouseenter`, la seconde, à celui de l'événement `mouseleave`. Par conséquent, pour retirer les gestionnaires indiqués avec `.hover()`, nous devons délier `mouseenter` et `mouseleave`.

À présent, lorsque le pointeur de la souris survole une couverture, l'icône appropriée est affichée par-dessus l'image (voir Figure 9.15).

Figure 9.15

Affichage de l'icône appropriée lors du survol d'une couverture.



Agrandir l'image

Notre galerie d'images est opérationnelle, avec un carrousel qui permet à l'utilisateur d'afficher la couverture qui l'intéresse. Un clic sur l'image centrale mène à une version agrandie de la couverture en question. Toutefois, nous pouvons mieux exploiter cette possibilité d'agrandissement de l'image.

Au lieu de conduire l'utilisateur vers une URL distincte lorsqu'il clique sur l'image centrale, nous pouvons afficher la couverture agrandie par-dessus la page elle-même.

INFO

Plusieurs variantes de l'affichage d'informations par-dessus la page sont disponibles sous la forme de plugins jQuery. FancyBox, ShadowBox, Thickbox, SimpleModal et jqModal font partie des plus populaires. Pour de plus amples informations concernant l'utilisation des plugins, consultez le Chapitre 10.

Pour cette image de couverture agrandie, nous avons besoin d'un nouvel élément `image`, que nous créons au moment de l'instanciation des icônes :

```
var $enlargedCover = $('<img/>')
  .addClass('enlarged')
  .hide()
  .appendTo('body');
```

À cette nouvelle classe, nous appliquons des règles de style semblables à celles définies précédemment :

```
img.enlarged {
  position: absolute;
  z-index: 5;
  cursor: pointer;
}
```

Le positionnement absolu permet d'afficher la couverture par-dessus les autres images positionnées, car la valeur de la propriété `z-index` est supérieure à celles déjà attribuées. Il nous reste à positionner réellement l'image agrandie lorsque l'utilisateur clique sur l'image au centre du diaporama :

```
// Image au centre : agrandir la couverture.
$covers.eq(1)
  .css('left', spacing)
  .click(function(event) {
    $enlargedCover.attr('src', $(this).attr('href'))
      .css({
        'left': ($('body').width() - 360) / 2,
        'top' : 100,
        'width': 360,
        'height': 444
      }).show();
    event.preventDefault();
  })
  .hover(function() {
    $enlargeRollover.appendTo(this).show();
  }, function() {
    $enlargeRollover.hide();
  });
```

Nous exploitons les liens déjà présents dans le source HTML pour connaître l'emplacement du fichier de l'image de couverture agrandie sur le serveur. Nous l'extrayons de l'attribut `href` du lien et l'affectons à l'attribut `src` de l'image agrandie.

Ensuite, nous positionnons cette image. Les propriétés `top`, `width` et `height` sont, pour le moment, figées, mais la propriété `left` exige un petit calcul. Nous voulons que l'image agrandie soit centrée sur la page, mais nous ne pouvons connaître à l'avance les coordonnées permettant d'obtenir ce placement. Nous déterminons l'axe central de la page en mesurant la largeur de l'élément `<body>` et en la divisant par deux. Puisque notre image agrandie doit être placée à cheval sur cet axe et que sa largeur est égale à 360 pixels, sa propriété `left` est donc `($('body').width() - 360) / 2`. La couverture est ainsi positionnée au bon endroit, c'est-à-dire centrée horizontalement sur la page (voir Figure 9.16).

Figure 9.16

Affichage de la couverture agrandie au centre de la page.

Masquer la couverture agrandie

Nous avons besoin d'un mécanisme pour enlever la couverture agrandie. La solution la plus simple consiste à la faire disparaître lors d'un clic :

```
// Image au centre : agrandir la couverture.
$covers.eq(1)
  .css('left', spacing)
  .click(function(event) {
    $enlargedCover.attr('src', $(this).attr('href'))
      .css({
        'left': ($('body').width() - 360) / 2,
        'top' : 100,
        'width': 360,
        'height': 444
      })
      .show()
      .one('click', function() {
        $enlargedCover.fadeOut();
      });
    event.preventDefault();
  })
  .hover(function() {
    $enlargeRollover.appendTo(this).show();
  }, function() {
    $enlargeRollover.hide();
  });
```

En invoquant la méthode `.one()` pour lier ce gestionnaire de `click`, nous écartons deux problèmes potentiels. Si le gestionnaire est lié à l'aide de la méthode `.bind()`, l'utilisateur peut cliquer sur l'image pendant qu'elle disparaît. Cela déclenche un nouvel appel au gestionnaire. Par ailleurs, puisque nous réutilisons le même élément d'image pour chaque couverture agrandie, la liaison est effectuée à chaque agrandissement. Si nous ne faisons rien pour délier le gestionnaire, ils vont se cumuler. Grâce à `.one()`, nous sommes certains que les gestionnaires sont retirés après usage.

Afficher un bouton de fermeture

Cette mise en œuvre suffit à retirer la couverture agrandie, mais l'utilisateur ne sait pas qu'il peut cliquer sur l'image pour la faire disparaître. Nous pouvons l'aider en ajoutant un bouton FERMER sous forme d'un *badge*. La création de ce bouton ressemble à la définition des autres éléments *singletons* que nous avons utilisés – ceux qui apparaissent une seule fois – et nous pouvons donc appeler la fonction utilitaire précédente :

```
var $closeButton = createControl('images/close.gif')
  .addClass('enlarged-control')
  .appendTo('body');
```

Lors d'un clic sur la couverture centrale, l'image agrandie est affichée, puis nous devons positionner et faire apparaître le bouton :

```
$closeButton.css({
  'left': ($('body').width() - 360) / 2,
  'top' : 100
}).show();
```

Puisque les coordonnées du bouton FERMER sont identiques à celles de la couverture agrandie, leurs coins supérieurs gauches sont alignés (voir Figure 9.17).

Un comportement est déjà associé à l'image pour la masquer lorsque l'utilisateur clique dessus. Dans ce type de situation, nous pouvons nous appuyer sur le *bouillonnement d'événement* pour que le clic sur le bouton FERMER provoque le même effet. Toutefois, contrairement à ce que l'on pourrait croire, le bouton FERMER n'est pas un *élément descendant* de la couverture. Nous avons positionné ce bouton de manière absolue au-dessus de la couverture. Autrement dit, un clic sur le bouton n'est pas transmis à l'image agrandie. Nous devons donc gérer les clics sur le bouton FERMER :

```
// Image au centre : agrandir la couverture.
$covers.eq(1)
  .css('left', spacing)
  .click(function(event) {
    $enlargedCover.attr('src', $(this).attr('href'))
      .css({
        'left': ($('body').width() - 360) / 2,
        'top' : 100,
        'width': 360,
        'height': 444
      })
      .show()
```

Figure 9.17

Ajout d'un bouton de fermeture.


```
    .one('click', function() {
      $closeButton.unbind('click').hide();
      $enlargedCover.fadeOut();
    });
  $closeButton
    .css({
      'left': ($('body').width() - 360) / 2,
      'top' : 100
    })
    .show()
    .click(function() {
      $enlargedCover.click();
    });
  event.preventDefault();
})
.hover(function() {
  $enlargeRollover.appendTo(this).show();
}, function() {
  $enlargeRollover.hide();
});
```

Lorsque nous affichons le bouton FERMER, nous lui associons un gestionnaire de `click`. Ce gestionnaire déclenche simplement le gestionnaire de `click` déjà lié à l'image agrandie. Nous devons néanmoins modifier ce gestionnaire pour y masquer le bouton FERMER. Nous en profitons également pour délier le gestionnaire de `click` de manière à éviter leur cumul.

Ajouter un autre badge

Le source HTML comprend les prix des livres, que nous nous proposons d'afficher sur la couverture agrandie. Nous allons appliquer la technique développée pour le bouton FERMER à du contenu textuel à la place d'une image.

Nous créons à nouveau un *élément singleton* au début du code JavaScript :

```
var $priceBadge = $('<div/>')
  .addClass('enlarged-price')
  .css('opacity', 0.6)
  .css('display', 'none')
  .appendTo('body');
```

Puisque le prix sera partiellement transparent, un contraste élevé entre la couleur du texte et l'arrière-plan est préférable :

```
.enlarged-price {
  background-color: #373c40;
  color: #fff;
  width: 80px;
  padding: 5px;
  font-size: 18px;
  font-weight: bold;
  text-align: right;
  position: absolute;
  z-index: 6;
}
```

Avant de pouvoir afficher le badge indiquant le prix, nous devons le remplir avec cette information tirée du contenu HTML. Dans le gestionnaire de `click` de la couverture centrale, le mot clé `this` fait référence à l'élément du lien. Puisque le prix est indiqué dans un élément `<span>` à l'intérieur du lien, il est très facile d'obtenir le texte correspondant :

```
var price = $(this).find('.price').text();
```

Nous pouvons alors afficher le badge en même temps que la couverture agrandie :

```
$priceBadge.css({
  'right': ($('body').width() - 360) / 2,
  'top' : 100
}).text(price).show();
```

Les styles définis placent le prix dans le coin supérieur droit de l'image (voir Figure 9.18).

Après avoir ajouté `$priceBadge.hide();` dans le gestionnaire de `click` de la couverture, la gestion de ce nouveau badge est terminée.

Figure 9.18

Affichage du prix du livre dans un nouveau badge.

Animer l'agrandissement de la couverture

Lorsque l'utilisateur clique sur la couverture centrale, la version agrandie apparaît immédiatement au centre de la page. Pour apporter plus de classe à cet affichage, nous pouvons utiliser les animations intégrées à jQuery. Nous allons mettre en œuvre une transition harmonieuse entre la vignette de la couverture et sa version agrandie.

Pour cela, nous devons connaître les coordonnées de départ de l'animation, c'est-à-dire l'emplacement de la couverture centrale sur la page. Pour obtenir cette information, il nous faut parcourir intelligemment le DOM avec du code JavaScript, mais jQuery va nous faciliter la tâche. La méthode `.offset()` retourne un objet qui contient les coordonnées `left` et `top` d'un élément relativement à la page. Nous pouvons ensuite insérer les propriétés `width` et `height` de l'image dans cet objet pour encapsuler les informations de positionnement :

```
var startPos = $(this).offset();
startPos.width = $(this).width();
startPos.height = $(this).height();
```

Les coordonnées de destination sont à présent relativement simples à calculer. Nous les plaçons dans un objet semblable :

```
var endPos = {};
endPos.width = startPos.width * 3;
endPos.height = startPos.height * 3;
endPos.top = 100;
endPos.left = ($('body').width() - endPos.width) / 2;
```

Ces deux objets sont utilisés comme des *mappes* d'attributs CSS, que nous pouvons passer à des méthodes telles que `.css()` et `.animate()` :

```
$enlargedCover.attr('src', $(this).attr('href'))
  .css(startPos)
  .show()
  .animate(endPos, 'normal', function() {
    $enlargedCover
      .one('click', function() {
        $closeButton.unbind('click').hide();
        $priceBadge.hide();
        $enlargedCover.fadeOut();
      });
    $closeButton
      .css({
        'left': endPos.left,
        'top' : endPos.top
      })
      .show()
      .click(function() {
        $enlargedCover.click();
      });
    $priceBadge
      .css({
        'right': endPos.left,
        'top' : endPos.top
      })
      .text(price)
      .show();
});
```

Vous remarquerez que le bouton FERMER et le prix ne peuvent pas être ajoutés tant que l'animation n'est pas terminée. Nous déplaçons donc le code correspondant dans la fonction de rappel de la méthode `.animate()`. Par ailleurs, nous profitons de cette opportunité pour simplifier les appels à `.css()` sur ces éléments en réutilisant les informations de positionnement calculées pour la couverture agrandie.

À présent, le passage de la vignette à l'image agrandie se fait progressivement (voir Figure 9.19).

Retarder les animations jusqu'au chargement de l'image

Notre animation est harmonieuse, mais elle dépend d'une connexion rapide avec le serveur. Si le téléchargement de la couverture agrandie prend du temps, le début de l'animation risque d'afficher la croix rouge qui indique une image inexistante ou de continuer à afficher l'image précédente. Nous pouvons faire en sorte que la transition

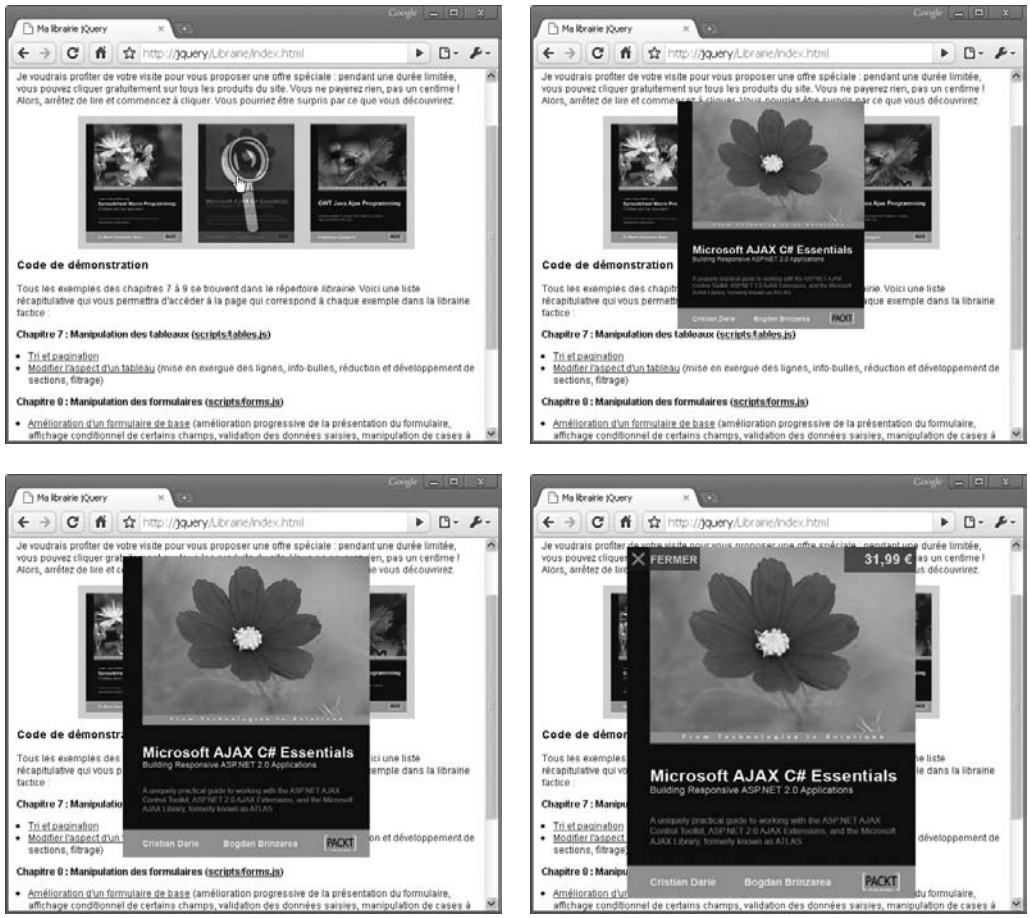


Figure 9.19

Passage progressif de la vignette à la couverture agrandie.

reste élégante en attendant que l'image soit intégralement reçue pour démarrer l'animation :

```
$enlargedCover.attr('src', $(this).attr('href'))
  .css(startPos)
  .show();

var performAnimation = function() {
  $enlargedCover.animate(endPos, 'normal', function() {
    $enlargedCover.one('click', function() {
      $closeButton.unbind('click').hide();
      $priceBadge.hide();
      $enlargedCover.fadeOut();
    });
```

```
      $closeButton
        .css({
          'left': endPos.left,
          'top' : endPos.top
        })
        .show()
        .click(function() {
          $enlargedCover.click();
        });
      $priceBadge
        .css({
          'right': endPos.left,
          'top' : endPos.top
        })
        .text(price)
        .show();
    });
};

if ($enlargedCover[0].complete) {
  performAnimation();
}
else {
  $enlargedCover.bind('load', performAnimation);
}
```

Nous devons prendre en compte deux cas : soit l'image est disponible quasi immédiatement (en raison du cache), soit son téléchargement prend du temps. Dans le premier cas, l'attribut `complete` de l'image est égal à `true` et nous pouvons invoquer immédiatement la nouvelle fonction `performAnimation()`. Dans le second cas, nous devons attendre que le téléchargement de l'image soit terminé avant d'appeler `performAnimation()`. Il s'agit d'un des rares cas où l'événement `load` standard du DOM nous est plus utile que l'événement `ready` de jQuery. Puisque `load` est déclenché sur une page, une image ou un cadre lorsque l'intégralité du contenu correspondant a été téléchargée, nous pouvons l'observer afin d'être certains que l'image puisse être affichée correctement. À ce moment-là, le gestionnaire est exécuté et l'animation est réalisée.

INFO

Nous utilisons la syntaxe `.bind('load')` à la place de la méthode abrégée `.load()` pour une question de clarté. En effet, `.load()` est également une méthode AJAX. Les deux syntaxes sont interchangeables.

Internet Explorer et Firefox interprètent différemment l'action à effectuer lorsque l'image se trouve dans le cache du navigateur. Firefox déclenche immédiatement l'événement `load` pour JavaScript, tandis qu'Internet Explorer ne l'envoie jamais car aucun chargement n'est effectué. Notre test de l'attribut `complete` permet de passer outre cette différence d'interprétation.

Ajouter un indicateur de chargement

Si la connexion réseau est lente, nous risquons de faire face à une situation embarrassante due au temps de chargement de l'image. La page semble bloquée pendant ce téléchargement. Comme nous l'avons fait pour le prompteur de nouvelles, nous pouvons signaler à l'utilisateur qu'une activité est en cours par l'affichage d'un *indicateur de chargement*.

Cet indicateur est une autre image *singleton* affichée au moment opportun :

```
var $waitThrobber = $('<img/>')
  .attr('src', 'images/wait.gif')
  .addClass('control')
  .css('z-index', 4)
  .hide();
```

Pour cette image, nous choisissons un GIF animé (voir Figure 9.20), car cela permet d'indiquer plus clairement à l'utilisateur qu'une activité a lieu.

Figure 9.20
Le GIF animé servant d'indicateur de chargement.

L'élément étant défini, deux lignes suffisent à mettre en place notre indicateur de chargement. Au tout début de notre gestionnaire de `click` associé à l'image centrale, avant de faire quoi que ce soit d'autre, nous affichons l'indicateur :

```
$waitThrobber.appendTo(this).show();
```

Et, au tout début de la fonction `performAnimation()`, lorsque nous savons que l'image a été chargée, nous le retirons :

```
$waitThrobber.hide();
```

Ces deux modifications permettent d'ajouter un indicateur de chargement à la couverture agrandie. L'animation se superpose au coin supérieur gauche de la couverture centrale (voir Figure 9.21).

Version finale du code

Ce chapitre n'a présenté qu'une petite partie des possibilités offertes par l'animation d'images animées et des prompteurs sur le Web. Voici la version finale du code de mise en œuvre du carrousel d'images :

Figure 9.21

Affichage de l'indicateur de chargement sur la couverture centrale.

```
$(document).ready(function() {
  var spacing = 140;

  function createControl(src) {
    return $('<img/>')
      .attr('src', src)
      .addClass('control')
      .css('opacity', 0.6)
      .css('display', 'none');
  }

  var $leftRollover = createControl('images/left.gif');
  var $rightRollover = createControl('images/right.gif');
  var $enlargeRollover = createControl('images/enlarge.gif');
  var $enlargedCover = $('<img/>')
    .addClass('enlarged')
    .hide()
    .appendTo('body');
  var $closeButton = createControl('images/close.gif')
    .addClass('enlarged-control')
    .appendTo('body');
  var $priceBadge = $('<div/>')
    .addClass('enlarged-price')
    .css('opacity', 0.6)
    .css('display', 'none')
    .appendTo('body');
  var $waitThrobber = $('<img/>')
    .attr('src', 'images/wait.gif')
    .addClass('control')
    .css('z-index', 4)
    .hide();

  $('#featured-books').css({
    'width': spacing * 3,
    'height': '166px',
    'overflow': 'hidden'
  }).find('.covers a').css({
    'float': 'none',
    'position': 'absolute',
    'left': 1000
  });
```

```
var setUpCovers = function() {
  var $covers = $('#featured-books .covers a');

  $covers.unbind('click mouseenter mouseleave');

  // Image de gauche : défilement vers la droite (pour afficher les
  // images qui se trouvent à gauche).
  $covers.eq(0)
    .css('left', 0)
    .click(function(event) {
      $covers.eq(0).animate({'left': spacing}, 'fast');
      $covers.eq(1).animate({'left': spacing * 2}, 'fast');
      $covers.eq(2).animate({'left': spacing * 3}, 'fast');
      $covers.eq($covers.length - 1)
        .css('left', -spacing)
        .animate({'left': 0}, 'fast', function() {
          $(this).prependTo('#featured-books .covers');
          setUpCovers();
        });
      event.preventDefault();
    }).hover(function() {
      $leftRollover.appendTo(this).show();
    }, function() {
      $leftRollover.hide();
    });

  // Image de droite : défilement vers la gauche (pour afficher les
  // images qui se trouvent à droite).
  $covers.eq(2)
    .css('left', spacing * 2)
    .click(function(event) {
      $covers.eq(0)
        .animate({'left': -spacing}, 'fast', function() {
          $(this).appendTo('#featured-books .covers');
          setUpCovers();
        });
      $covers.eq(1).animate({'left': 0}, 'fast');
      $covers.eq(2).animate({'left': spacing}, 'fast');
      $covers.eq(3)
        .css('left', spacing * 3)
        .animate({'left': spacing * 2}, 'fast');
      event.preventDefault();
    }).hover(function() {
      $rightRollover.appendTo(this).show();
    }, function() {
      $rightRollover.hide();
    });

  // Image au centre : agrandir la couverture.
  $covers.eq(1)
    .css('left', spacing)
    .click(function(event) {
      $waitThrobber.appendTo(this).show();
      var price = $(this).find('.price').text();
      var startPos = $(this).offset();
      startPos.width = $(this).width();
```

```
        startPos.height = $(this).height();
        var endPos = {};
        endPos.width = startPos.width * 3;
        endPos.height = startPos.height * 3;
        endPos.top = 100;
        endPos.left = ($('body').width() - endPos.width) / 2;

        $enlargedCover.attr('src', $(this).attr('href'))
          .css(startPos)
          .show();

        var performAnimation = function() {
          $waitThrobber.hide();
          $enlargedCover.animate(endPos, 'normal', function() {
            $enlargedCover.one('click', function() {
              $closeButton.unbind('click').hide();
              $priceBadge.hide();
              $enlargedCover.fadeOut();
            });
            $closeButton
              .css({
                'left': endPos.left,
                'top' : endPos.top
              })
              .show()
              .click(function() {
                $enlargedCover.click();
              });
            $priceBadge
              .css({
                'right': endPos.left,
                'top' : endPos.top
              })
              .text(price)
              .show();
          });
        };

        if ($enlargedCover[0].complete) {
          performAnimation();
        }
        else {
          $enlargedCover.bind('load', performAnimation);
        }
        event.preventDefault();
      })
      .hover(function() {
        $enlargeRollover.appendTo(this).show();
      }, function() {
        $enlargeRollover.hide();
      });
  };
  setUpCovers();
});
```

9.3 En résumé

Dans ce chapitre, nous avons examiné des éléments de la page qui changent au fil du temps, soit de leur propre fait, soit en réponse à une action de l'utilisateur. Ces *diaporamas* et *prompteurs* peuvent faire toute la différence entre une présence web moderne et une conception traditionnelle des sites. Nous avons étudié la présentation d'un flux d'informations XML sur la page, ainsi que le défilement planifié d'éléments afin de les faire apparaître puis disparaître. Au cours de la mise en œuvre d'un affichage d'images dans une galerie de type carrousel, nous avons également vu comment agrandir une image pour une vue plus détaillée, en ajoutant de manière non intrusive une animation et des contrôles d'interface utilisateur.

La conjugaison de ces techniques donne vie à des pages qui seraient sinon fades, tout en améliorant la convivialité des applications web. Grâce à la puissance de jQuery, les animations et les effets peuvent être mis en œuvre sans trop d'efforts.

10

Utilisation des plugins

Au sommaire de ce chapitre

- Rechercher des plugins et de l'aide
- Utiliser un plugin
- Le plugin Form
- La bibliothèque jQuery UI
- Autres plugins recommandés
- En résumé

Tout au long des chapitres précédents, nous avons examiné différentes manières d'utiliser la bibliothèque jQuery, pour réaliser une grande variété de tâches. Nous n'avons pas encore vraiment exploré un autre de ses aspects importants, son extensibilité. Aussi puissant que puisse être le noyau de jQuery, son élégante *architecture de plugins* a permis aux développeurs d'étendre la bibliothèque pour la rendre encore plus riche en fonctionnalités.

La communauté jQuery en pleine expansion a créé des centaines de plugins, allant des petits assistants à la sélection jusqu'aux widgets complets pour l'interface utilisateur. Au Chapitre 7, nous avons déjà pu constater la puissance des plugins et en avons même créé un relativement simple. Dans ce chapitre, nous expliquerons comment rechercher les plugins développés par d'autres programmeurs et les inclure dans nos pages web. Nous examinerons le très populaire plugin Form, ainsi que la bibliothèque officielle jQuery UI. Nous présenterons également plusieurs autres plugins très utilisés, que nous ne pouvons que recommander.

10.1 Rechercher des plugins et de l'aide

Le site web de jQuery propose un vaste catalogue des plugins disponibles (**http://plugins.jquery.com/**), avec une notation effectuée par les utilisateurs, une gestion des

versions et la possibilité de signaler des bogues. Ce catalogue constitue également un bon point de départ pour la recherche de documentation. Chaque plugin disponible peut être téléchargé sous forme de fichier .zip et nombre d'entre eux possèdent des liens vers des pages de démonstration, du code d'exemple et des didacticiels.

Un plus grand nombre de plugins encore sont disponibles dans des entrepôts de code généraux, comme **http://github.com/**, et sur les blogs des développeurs de plugins.

Si vous ne trouvez pas de réponse à toutes vos questions à partir du catalogue de plugins, du site web d'un auteur et des commentaires sur les plugins, vous pouvez vous tourner vers les groupes Google dédiés à jQuery, notamment **http://groups.google.com/group/jquery-en/** (en anglais) et **http://groups.google.com/group/jquery-fr/** (en français). La plupart des auteurs de plugins contribuent fréquemment à la liste de discussion et sont toujours prêts à aider les nouveaux utilisateurs.

10.2 Utiliser un plugin

L'utilisation d'un plugin jQuery est très simple. La première étape consiste à l'inclure dans l'élément `<head>` du document, en vérifiant qu'il apparaît après le fichier de jQuery :

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
  <script src="jquery.js" type="text/javascript"></script>
  <script src="jquery.plugin.js" type="text/javascript"></script>
  <script src="custom.js" type="text/javascript"></script>
  <title>Exemple</title>
</head>
```

Ensuite, il suffit d'inclure notre propre fichier JavaScript, dans lequel nous invoquons les méthodes créées ou étendues par le plugin. Très souvent, la réalisation d'une opération demande simplement d'ajouter une seule ligne dans la méthode `$(document).ready()` de notre fichier :

```
$(document).ready(function() {
  $('#monID').unPlugin();
});
```

De nombreux plugins font également preuve de flexibilité en fournissant des paramètres facultatifs qui nous permettent de modifier leur comportement à loisir. En général, cette configuration se fait en passant une *mappe* en argument de la méthode :

```
$(document).ready(function() {
  $('#monID').unPlugin ({
    send: true,
    message: 'Ce plugin est super !'
  });
});
```

La syntaxe des plugins jQuery ressemble fortement à celle des méthodes du noyau de la bibliothèque. Puisque nous savons à présent comment inclure un plugin dans une page web, nous pouvons en examiner deux parmi les plus utilisés.

10.3 Le plugin Form

Le plugin Form est un parfait exemple de script qui transforme une tâche difficile et complexe en quelque chose d'extrêmement simple. Le fichier du plugin et sa documentation détaillée sont disponibles à l'adresse **http://www.malsup.com/jquery/form/**.

Au cœur du plugin, nous trouvons la méthode `.ajaxForm()`. Pour convertir un formulaire classique en un formulaire AJAX, une seule ligne de code suffit :

```
$(document).ready(function() {
  $('#monFormulaire').ajaxForm();
});
```

Cet exemple prépare `<form id="monFormulaire">` pour que l'internaute puisse le soumettre sans que la page courante ne soit actualisée. En soi, cette fonctionnalité est déjà intéressante, mais le véritable intérêt se trouve dans la mappe d'options que nous pouvons passer à la méthode. Par exemple, le code suivant invoque `.ajaxForm()` en précisant les options `target`, `beforeSubmit` et `success` :

```
$(document).ready(function() {
  function validateForm() {
    // Le code de validation du formulaire est placé ici.
    // Nous pouvons retourner false pour annuler la soumission.
  };
  $('#test-form').ajaxForm({
    target: '#log',
    beforeSubmit: validateForm,
    success: function() {
      alert('Merci pour vos commentaires !');
    }
  });
});
```

L'option `target`[1] désigne le ou les éléments, dans notre cas l'élément dont l'identifiant est `"log"`, qui seront mis à jour à partir de la réponse du serveur.

L'option `beforeSubmit` précise les opérations qui seront réalisées avant que le formulaire ne soit soumis. Dans notre exemple, elle fait référence à la fonction `validateForm()`. Si cette fonction retourne `false`, le formulaire n'est pas soumis.

1. N.d.T. : attention, il ne faut pas confondre cette option avec l'attribut HTML `target` de la balise `<form>`.

L'option `success` indique les tâches qui seront effectuées après une soumission réussie du formulaire. Dans l'exemple, elle affiche simplement un message d'alerte afin que l'utilisateur sache que l'envoi du formulaire s'est parfaitement déroulé.

Voici les autres options disponibles avec les méthodes `.ajaxForm()` et `.ajaxSubmit()` :

- `url`. L'URL vers laquelle les données du formulaire seront envoyées, si elle diffère de l'attribut `action` du formulaire.
- `type`. La méthode employée pour soumettre le formulaire – `GET` ou `POST`. La valeur par défaut est donnée par l'attribut `method` du formulaire. Si cet attribut est absent, la méthode `GET` est choisie par défaut.
- `dataType`. Le type de données attendu pour la réponse du serveur. Les valeurs possibles sont `null`, `xml`, `script` et `json`. La valeur par défaut est `null` et correspond à une réponse HTML.
- `resetForm`. La valeur par défaut de ce booléen est `false`. Lorsqu'il est fixé à `true`, tous les champs du formulaire sont réinitialisés à leur valeur par défaut après la réussite de la soumission.
- `clearForm`. La valeur par défaut de ce booléen est `false`. Lorsqu'il est fixé à `true`, tous les champs du formulaire sont effacés après la réussite de la soumission.

Le plugin Form propose d'autres méthodes de gestion des formulaires et de leurs données. Sur le site **http://www.malsup.com/jquery/form/**, vous trouverez de plus amples informations concernant ces méthodes, ainsi que des démonstrations et des exemples.

Astuces et conseils

En général, la méthode `.ajaxForm()` est plus pratique que la méthode `.ajaxSubmit()`, mais elle offre moins de souplesse. Si nous souhaitons que le plugin s'occupe de la liaison des événements à notre place et invoque pour nous la méthode `.ajaxSubmit()` au moment approprié, nous devons utiliser `.ajaxForm()`. Si nous voulons disposer d'un contrôle plus précis sur la gestion de l'événement `submit`, il faut opter pour `.ajaxSubmit()`.

Par défaut, `.ajaxForm()` et `.ajaxSubmit()` utilisent les valeurs des attributs `action` et `method` indiquées dans le balisage du formulaire. Tant que le balisage du formulaire est valide, le plugin fonctionne comme attendu, sans que nous ayons besoin d'un quelconque réglage. Par ailleurs, nous bénéficions automatiquement des avantages de l'*amélioration progressive* ; le formulaire est totalement opérationnel même lorsque JavaScript est désactivé.

Normalement, lorsqu'un formulaire est envoyé et que l'élément utilisé pour cette soumission est nommé, ses attributs `name` et `value` sont transmis avec les autres données du formulaire. Sur ce point, la méthode `.ajaxForm()` a une approche *proactive*, en ajoutant des gestionnaires de `click` à tous les éléments `<input type="submit">` afin qu'elle puisse déterminer celui qui a servi à l'envoi du formulaire. En revanche, la méthode `.ajaxSubmit()` a une approche *réactive* et n'a aucun moyen de connaître cette information. Elle ne capture pas l'élément utilisé pour la soumission. La même distinction concerne les éléments `<input type="image">` : `.ajaxForm()` les prend en charge, tandis que `.ajaxSubmit()` les ignore.

À moins qu'un fichier ne soit joint à la soumission du formulaire, les méthodes `.ajaxForm()` et `.ajaxSubmit()` passent leur argument `options` à la méthode `$.ajax()`, qui fait partie du noyau de jQuery. Par conséquent, les options reconnues par la méthode `$.ajax()` peuvent être transmises au travers du plugin Form. Grâce à cette fonctionnalité, les réponses aux formulaires AJAX peuvent être encore plus robustes :

```
$(document).ready(function() {
  $(#monFormulaire).ajaxForm({
    timeout: 2000,
    error: function (xml, status, e) {
      alert(e.message);
    }
  });
});
```

Lorsque le niveau de personnalisation n'a pas besoin d'être aussi élevé, nous pouvons passer aux méthodes `.ajaxForm()` et `.ajaxSubmit()` une fonction à la place d'une mappe d'options. Puisque cette fonction est traitée comme un gestionnaire invoqué en cas de réussite, nous pouvons obtenir le texte de la réponse renvoyée par le serveur :

```
$(document).ready(function() {
  $(#monFormulaire).ajaxForm(function(responseText) {
    alert(responseText);
  });
});
```

10.4 La bibliothèque jQuery UI

Si le plugin Form ne propose qu'une seule fonction, tout en la faisant très bien, le plugin jQuery UI offre de nombreuses fonctionnalités, tout en les faisant également très bien. En réalité, jQuery UI n'est pas tant un plugin qu'une bibliothèque de plugins.

Dirigée par Paul Bakaus, l'équipe jQuery UI a créé de nombreux *composants d'interaction* de base et des *widgets* permettant de proposer à l'internaute des pages web qui s'apparentent à des applications bureautiques. Parmi les composants d'interaction, nous trouvons des méthodes pour faire glisser, déposer, trier et redimensionner des éléments.

Dans sa version stable, la collection de widgets propose un menu accordéon, un sélecteur de date, une boîte de dialogue, un curseur et des onglets. D'autres widgets font l'objet d'un développement actif. Par ailleurs, jQuery UI propose un jeu d'effets élaborés qui complètent les animations de jQuery.

Puisque la bibliothèque est trop vaste pour la détailler intégralement dans ce chapitre, nous nous limiterons aux effets graphiques, au composant d'interaction Sortable et au widget Dialog. Les modules jQuery, leur documentation et des démonstrations sont disponibles sur le site **http://ui.jquery.com/**.

Effets

Le module des effets de jQuery UI est composé d'un fichier central et d'un ensemble de fichiers pour chaque effet. Le fichier central contient des animations orientées couleur et classe, ainsi qu'un *easing* élaboré.

Animation des couleurs

Après avoir référencé le fichier central des effets dans le document, la méthode `.animate()` accepte de nouvelles propriétés de style, comme `borderTopColor`, `backgroundColor` et `color`. Par exemple, nous pouvons animer un élément en le faisant passer progressivement d'un texte noir sur fond blanc à un texte blanc sur fond noir :

```
$(document).ready(function() {
  $('#mydiv').animate({
    color: '#fff',
    backgroundColor: '#000'
  }, 'slow');
});
```

La Figure 10.1 montre l'aspect du `<div>` lorsque l'animation en est à un peu plus de la moitié. Le texte de l'élément est en train de devenir blanc, tandis que son arrière-plan est presque noir.

Figure 10.1

Animation des couleurs d'un élément.

Animation des classes

Les trois méthodes de manipulation des classes que nous avons utilisées dans les chapitres précédents, c'est-à-dire `.addClass()`, `.removeClass()` et `.toggleClass()`, prennent à présent un second paramètre facultatif précisant la vitesse de l'animation. Nous pouvons les invoquer sous la forme `.addClass('highlight', 'fast')`, `.removeClass('highlight', 'slow')` ou `.toggleClass('highlight, 1000)`.

Easing élaboré

Les nouvelles fonctions d'easing font varier la vitesse et la position auxquelles les transitions se produisent. Par exemple, la fonction `easeInQuart` termine une animation quatre fois plus rapidement qu'elle ne l'a commencée. Nous pouvons indiquer une fonction d'easing personnalisée dans toutes les méthodes d'animation standard de jQuery ou des méthodes d'effet de jQuery UI. Pour cela, nous passons un argument supplémentaire ou ajoutons une option à une *mappe d'options*, selon la syntaxe choisie. Par exemple, pour préciser la fonction `easeInQuart` à l'animation des couleurs, nous pouvons ajouter un argument supplémentaire :

```
$(document).ready(function() {
  $('#mydiv').animate({
    color: '#fff',
    backgroundColor: '#000'
  }, 'slow', 'easeInQuart');
});
```

Nous pouvons également ajouter une option à la deuxième mappe :

```
$(document).ready(function() {
  $('#mydiv').animate({
    color: '#fff',
    backgroundColor: '#000'
  }, {
    duration: 'slow',
    easing: 'easeInQuart'
  });
});
```

Sur le site **http://gsgd.co.uk/sandbox/jquery/easing/**, vous trouverez des démonstrations pour l'ensemble des fonctions d'easing.

Effets supplémentaires

Les fichiers de chaque effet ajoutent diverses transitions, toutes pouvant être mises en œuvre avec la méthode `.effect()`, certaines étendant également les fonctionnalités des méthodes jQuery `.show()`, `.hide()` et `.toggle()`. Par exemple, l'effet `explode`, qui masque des éléments en les faisant exploser en un nombre de morceaux précisé, peut être obtenu avec la méthode `.effect()` :

```
$(document).ready(function() {
  $('#explode').effect('explode', {pieces: 16}, 800);
});
```

Il peut également être appliqué avec la méthode `.hide()` :

```
$(document).ready(function() {
  $('#explode').hide('explode', {pieces: 16}, 800);
});
```

Dans les deux cas, l'élément, dont l'état initial est illustré à la Figure 10.2 et celui à mi-animation, à la Figure 10.3, finit masqué.

Figure 10.2
Apparence initiale de l'élément.

Figure 10.3
Apparence de l'état à mi-animation.

Composants d'interaction

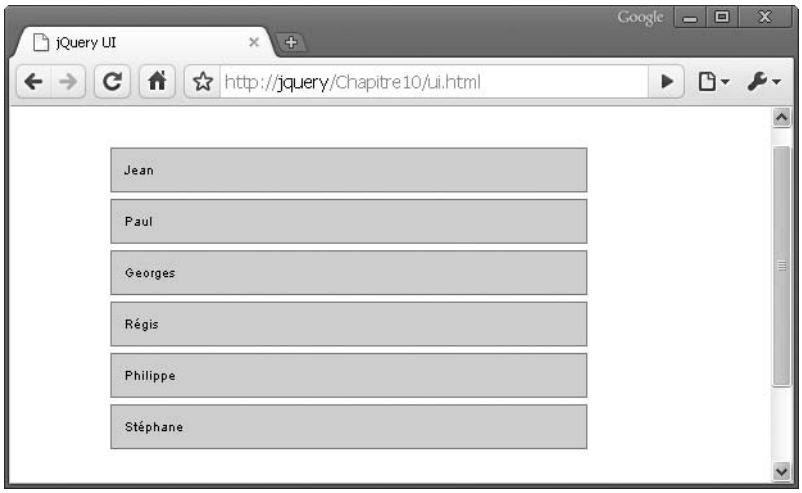
Entre autres *composants d'interaction*, jQuery UI propose Sortable, qui peut transformer n'importe quel groupe d'éléments en une liste acceptant le glisser-déposer. La Figure 10.4 présente une liste non ordonnée, avec des styles CSS appliqués à chaque élément.

Le balisage HTML correspondant est relativement simple :

```
<ul id="sort-container">
  <li>Jean</li>
  <li>Paul</li>
  <li>Georges</li>
```

Figure 10.4

Une liste non ordonnée stylée.




```
  <li>Régis</li>
  <li>Philippe</li>
  <li>Stéphane</li>
</ul>
```

Pour que la liste puisse être réorganisée, nous ajoutons simplement le code suivant :

```
$(document).ready(function() {
  $('#sort-container').sortable();
});
```

Cette unique ligne dans le gestionnaire `$(document).ready()` nous donne la possibilité de faire glisser chaque élément et de le déposer à un emplacement différent de la liste (voir Figure 10.5).

Figure 10.5

La liste peut être réorganisée par glisser-déposer.

Nous pouvons améliorer cette interaction avec l'utilisateur en ajoutant des options à la méthode `.sortable()`. Elle reconnaît plus de trente options, mais nous n'en utiliserons que quelques-unes dans notre exemple :

```
$(document).ready(function() {
  $('#sort-container').sortable({
    opacity: .5,
    cursor: 'move',
    axis: 'y'
  });
});
```

Les deux premières options, `opacity` et `cursor`, se comprennent d'elles-mêmes. La troisième, `axis`, limite le mouvement de l'élément selon l'axe indiqué, dans ce cas l'axe `y` (voir Figure 10.6).

Figure 10.6

Le déplacement de l'élément stylé se fait selon l'axe indiqué.

Comme l'illustre la couleur d'arrière-plan plus claire de l'élément en cours de déplacement, nous profitons d'une classe qui lui est attribuée automatiquement, `ui-sortable-helper`, pour appliquer un style à cette classe.

Pour de plus amples informations concernant les composants d'interaction standard de jQuery UI, rendez-vous sur la page **http://docs.jquery.com/UI#Interaction**.

Widgets

Outre les briques de construction, jQuery UI fournit tout un ensemble de *widgets* robustes pour l'interface utilisateur. Il s'agit de composants immédiatement opérationnels, semblables aux éléments auxquels nous ont habitués les applications bureautiques. Par exemple, le widget Dialog se fonde sur les composants Draggable et Resizable pour créer une boîte de dialogue, que nous n'avons pas à construire nous-mêmes.

Comme les autres widgets pour l'interface utilisateur, Dialog accepte un grand nombre d'options. Sa méthode `.dialog()` peut également prendre des chaînes en argument pour modifier le contenu affiché. Dans sa version la plus simple, la méthode `.dialog()` convertit un élément existant en une boîte de dialogue et l'affiche avec le contenu de l'élément. Nous pouvons, par exemple, partir d'une structure `<div>` simple :

```
<div id="dlg">Ma boîte de dialogue</div>
```

Vous ne serez pas surpris de constater que ce `<div>` apparaît sous forme d'un bloc de texte (voir Figure 10.7).

Figure 10.7

Initialement, l'élément est affiché comme un bloc de texte.

Dès que le DOM est prêt, nous pouvons invoquer la boîte de dialogue de base dans notre fichier JavaScript :

```
$(document).ready(function() {
  $('#dlg').dialog();
});
```

Le texte est alors enveloppé dans une boîte de dialogue (voir Figure 10.8).

Figure 10.8

Transformation de l'élément en une boîte de dialogue.

Cette boîte de dialogue peut être redimensionnée en cliquant sur l'un des bords et en le faisant glisser. Elle peut également être déplacée en cliquant n'importe où dans la partie haute, juste en dessous de la bordure supérieure. Il est même possible de la fermer en cliquant sur le lien placé dans le coin supérieur gauche.

Évidemment, nous pouvons obtenir beaucoup mieux en appliquant des styles. Si les styles qui permettent aux widgets d'être opérationnels sont fournis par jQuery UI, c'est à nous de définir leur apparence. La Figure 10.9 présente la boîte de dialogue à laquelle un *thème* par défaut est appliqué.

Figure 10.9

Application d'un thème à la boîte de dialogue.

Les différentes zones sont désormais clairement indiquées. De plus, le pointeur de la souris change lorsqu'il arrive sur les parties de la boîte de dialogue qui peuvent servir au déplacement et au redimensionnement.

À l'instar des autres méthodes de jQuery UI, `.dialog()` reconnaît plusieurs options. Certaines d'entre elles affectent l'aspect de la boîte de dialogue, tandis que d'autres permettent de déclencher des événements. Voici un exemple de ces options :

```
$(document).ready(function() {
  var $dlg = $('#dlg');
  var dlgText = $dlg.text();
  $dlg.dialog({
    autoOpen: false,
    title: dlgText,
    open: function() {
      $dlg.empty();
    },
    buttons: {
      'ajouter un message': function() {
        $dlg.append('<p>Message ajouté</p>');
      },
      'effacer les messages': function() {
        $('p', $dlg).remove();
      }
    }
  });
  $('#do-dialog').click(function() {
    $dlg.dialog('open');
  });
});
```

La boîte de dialogue est initialement masquée, pour être ouverte lorsque l'utilisateur clique sur un bouton dont l'identifiant est `do-dialog`. Nous déplaçons également le texte initial de la boîte de dialogue vers la zone de titre et ajoutons deux boutons, l'un avec AJOUTER UN MESSAGE en tant que libellé et l'autre avec EFFACER LES MESSAGES. Une fonction est associée à chaque bouton de manière à ajouter et à effacer des messages. La Figure 10.10 montre l'aspect de la boîte de dialogue après trois clics sur le bouton AJOUTER UN MESSAGE.

Figure 10.10
La boîte de dialogue améliorée.

Il existe de nombreuses autres options pour configurer l'affichage et le comportement des boîtes de dialogue. Pour de plus amples informations, consultez la page **http://docs.jquery.com/UI/Dialog#options**.

ThemeRoller

ThemeRoller est un ajout récent à la bibliothèque jQuery UI. Il s'agit d'un moteur interactif accessible au travers du Web pour créer des thèmes pour les widgets de l'interface utilisateur (voir Figure 10.11). Grâce à ThemeRoller, il est possible de créer rapidement et facilement des éléments hautement personnalisés à l'aspect professionnel. Nous l'avons indiqué, un thème par défaut a été appliqué à la boîte de dialogue créée précédemment. Il a été produit par ThemeRoller sans modifier aucun des paramètres.

Pour générer un jeu de styles totalement différent, il suffit d'aller sur la page **http://jqueryui.com/themeroller/**, de modifier les différents paramètres et d'appuyer sur le bouton DOWNLOAD THEME. Le fichier .zip contenant les feuilles de style et les images peut ensuite être placé dans le répertoire approprié. Par exemple, en choisissant des couleurs et des textures différentes, nous pouvons transformer l'aspect de notre boîte de dialogue précédente en moins de cinq minutes (voir Figure 10.12).

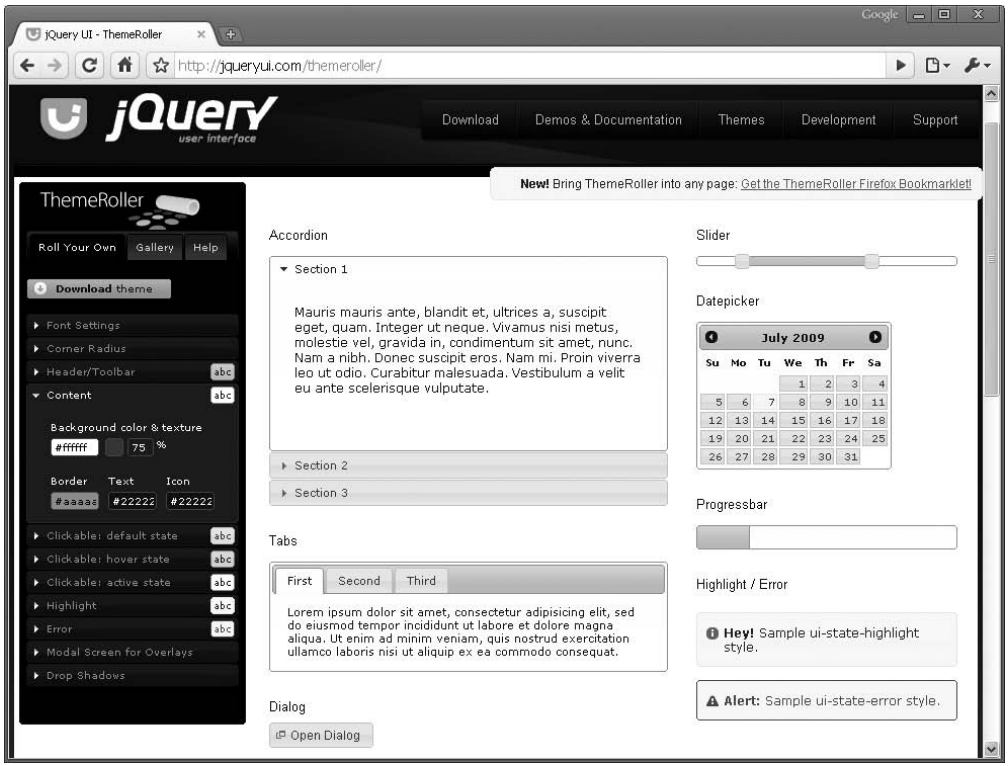


Figure 10.11

Le moteur de thèmes ThemeRoller.

Figure 10.12

Application d'un autre thème à la boîte de dialogue.

10.5 Autres plugins recommandés

Outre les plugins présentés dans ce chapitre et ailleurs dans cet ouvrage, nous recommandons ceux mentionnés dans cette section, pas uniquement en raison de leur popularité, mais également du fait de la robustesse de leur code.

Formulaires

Au Chapitre 8, nous avons étudié plusieurs manières de manipuler les formulaires. Les plugins suivants permettent d'accomplir les tâches associées avec facilité.

Autocomplete

- **http://bassistance.de/jquery-plugins/jquery-plugin-autocomplete/**
- **http://plugins.jquery.com/project/autocompletex**

Écrit par Jörn Zaefferer, l'un des développeurs du noyau de jQuery, le plugin Autocomplete propose une liste des correspondances possibles au fur et à mesure que l'utilisateur saisit un texte (voir Figure 10.13).

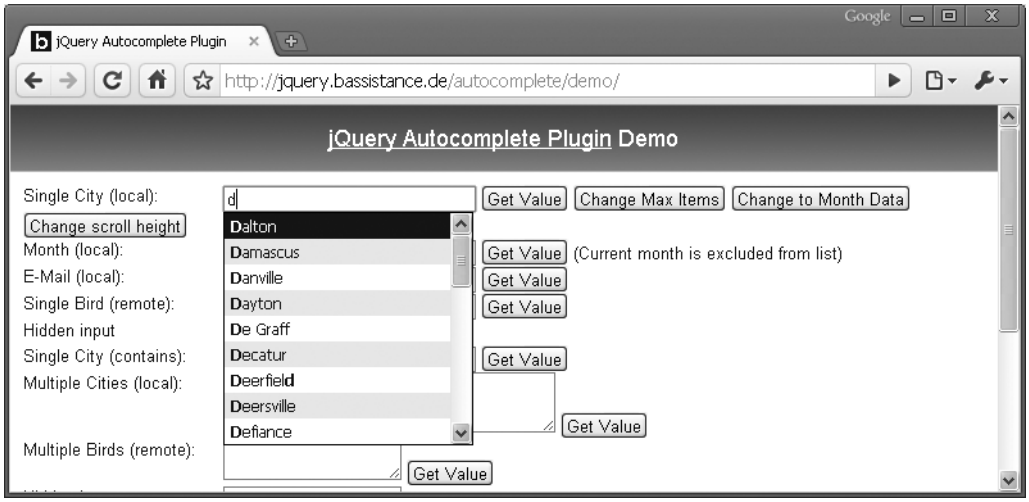


Figure 10.13
Démonstration du plugin Autocomplete.

Validation

- **http://bassistance.de/jquery-plugins/jquery-plugin-validation/**
- **http://plugins.jquery.com/project/validate**

Validation, un autre plugin dû à Jörn Zaefferer, est un outil extrêmement flexible pour la validation des informations saisies dans un formulaire en fonction d'un grand nombre de critères (voir Figure 10.14).

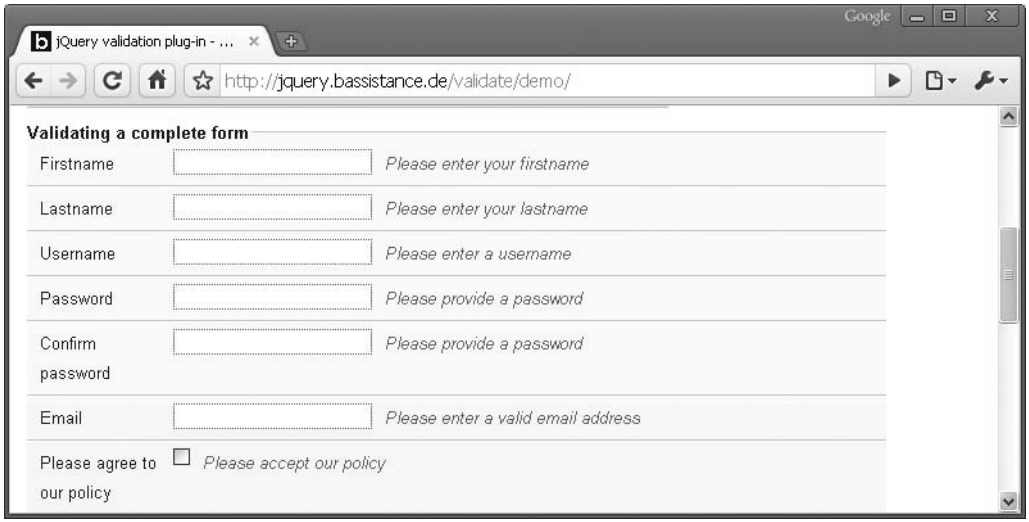


Figure 10.14

Démonstration du plugin Validation.

Jeditable

- **http://www.appelsiini.net/projects/jeditable**
- **http://plugins.jquery.com/project/jeditable**

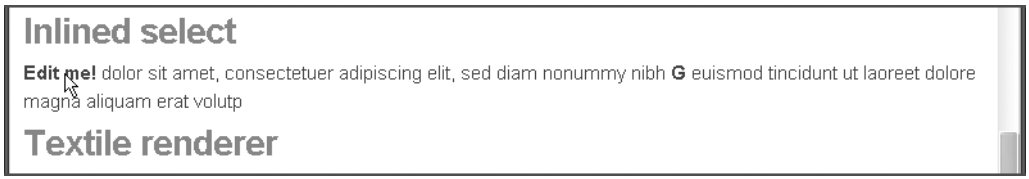
Le plugin Jeditable convertit les éléments qui ne servent pas dans les formulaires en zones de saisie modifiables lorsque l'utilisateur effectue une certaine action, comme un clic ou un double-clic (voir Figure 10.15). Le contenu modifié est automatiquement envoyé au serveur pour y être enregistré.

Masked Input

- **http://digitalbush.com/projects/masked-input-plugin/**
- **http://plugins.jquery.com/project/maskedinput**

Le plugin Masked Input propose aux utilisateurs une manière plus conviviale de saisir des données dans un certain format, comme des dates, des numéros de téléphone ou des numéros de sécurité sociale. Il place automatiquement certains caractères dans le champ, comme des barres obliques pour les dates, tout en autorisant uniquement la saisie des caractères définis par les options (voir Figure 10.16).

❶ Clic sur le texte à modifier



❷ Sélection du texte de remplacement

❸ Le texte est remplacé dans la page

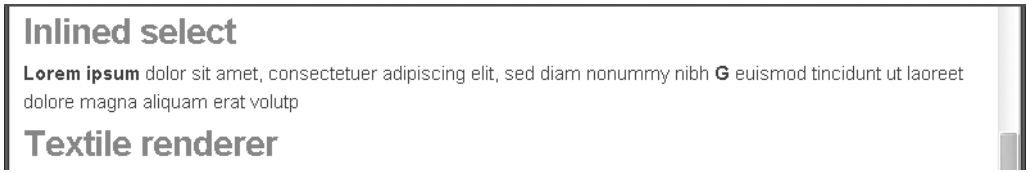
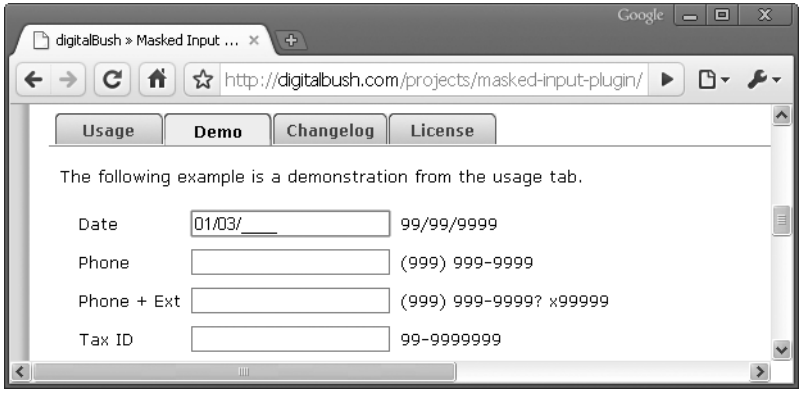


Figure 10.15

Démonstration du plugin Jeditable.

Figure 10.16

Démonstration du plugin Masked Input.



Tables

Au Chapitre 7, nous avons présenté des techniques pour organiser, embellir et mettre en valeur des données tabulaires. Plusieurs développeurs de plugins ont regroupé des procédures permettant de faciliter ces tâches.

Tablesorter

- **http://tablesorter.com/**
- **http://plugins.jquery.com/project/tablesorter**

Le plugin Tablesorter peut convertir n'importe quelle table comprenant des éléments `<thead>` et `<tbody>` en une table triable qui ne requiert pas une actualisation de la page (voir Figure 10.17). Il offre une fonction de tri multicolonne, des parseurs prenant en charge différents formats, comme les dates, les heures, la monnaie et les URL, un tri secondaire "masqué" et des possibilités d'extension au travers d'un système de widget.

Figure 10.17

Démonstration du plugin Tablesorter.

jqGrid

- **http://www.trirand.com/blog/**
- **http://plugins.jquery.com/project/jqGrids**

jqGrid est un contrôle JavaScript compatible AJAX qui permet aux développeurs de présenter et de manipuler dynamiquement des données tabulaires sur le Web (voir Figure 10.18). Il propose des options pour la modification en ligne, l'édition des cellules, la navigation par page, la sélection d'éléments multiples, les grilles secondaires et les grilles arborescentes. La documentation complète est disponible à l'adresse **http://www.secondpersonplural.ca/jqgriddocs/**.

Flexigrid

- **http://code.google.com/p/flexigrid/**
- **http://plugins.jquery.com/project/flexigrid**

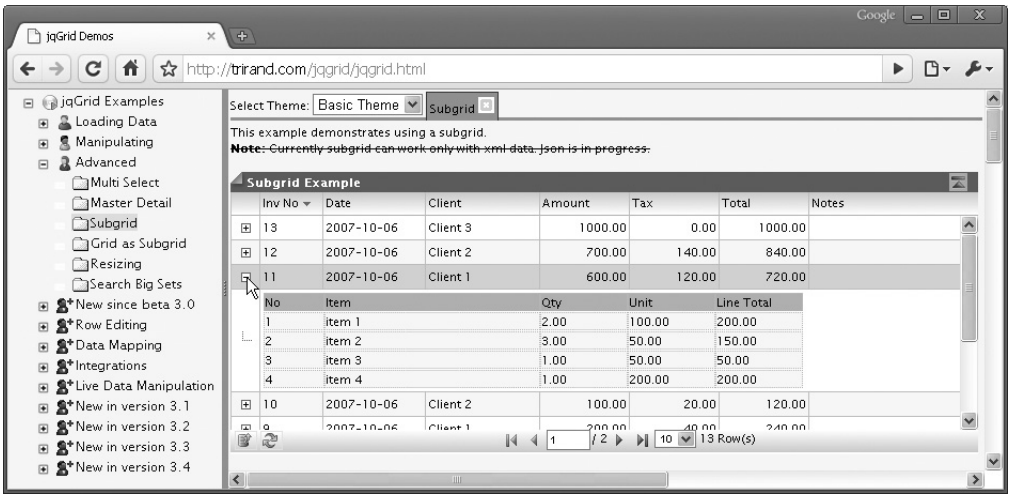


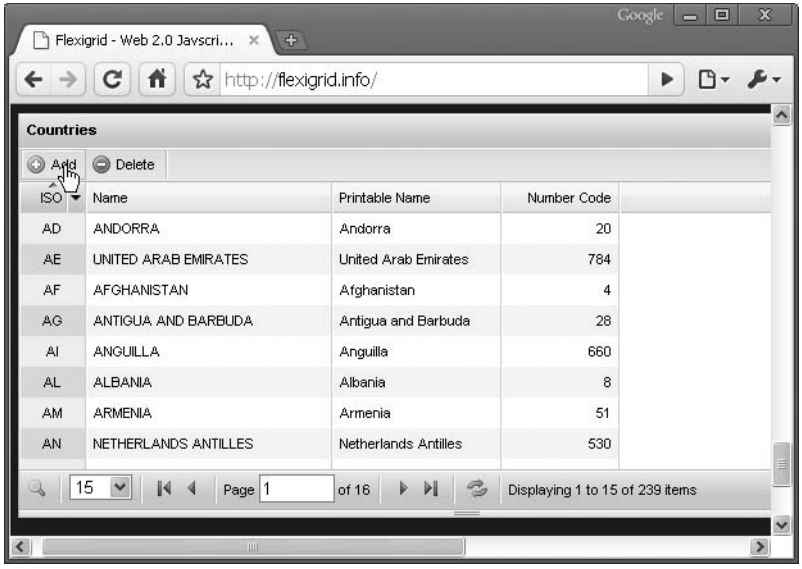
Figure 10.18

Démonstration du plugin jqGrid.

Comme jqGrid, Flexigrid est un plugin de présentation des données sous forme de grille (voir Figure 10.19). Parmi ses nombreuses fonctionnalités, citons la prise en charge de JSON, la pagination, la recherche rapide, l'affichage, le masquage et le redimensionnement des colonnes, ainsi que le tri des lignes.

Figure 10.19

Démonstration du plugin Flexigrid.



Images

La manipulation des images requiert souvent un traitement intensif côté serveur. Toutefois, quelques créateurs de plugins ont développé des solutions JavaScript de traitement simple des images en se fondant sur jQuery.

Jcrop

- **http://deepliquid.com/content/Jcrop.html**
- **http://plugins.jquery.com/project/Jcrop**

Jcrop constitue une solution rapide et simple pour ajouter le rognage des images dans les applications web. Ce plugin permet notamment de verrouiller les proportions de l'image, de préciser les tailles minimale et maximale, de prendre en charge le clavier, de faciliter les interactions avec l'utilisateur et de personnaliser les styles (voir Figure 10.20).

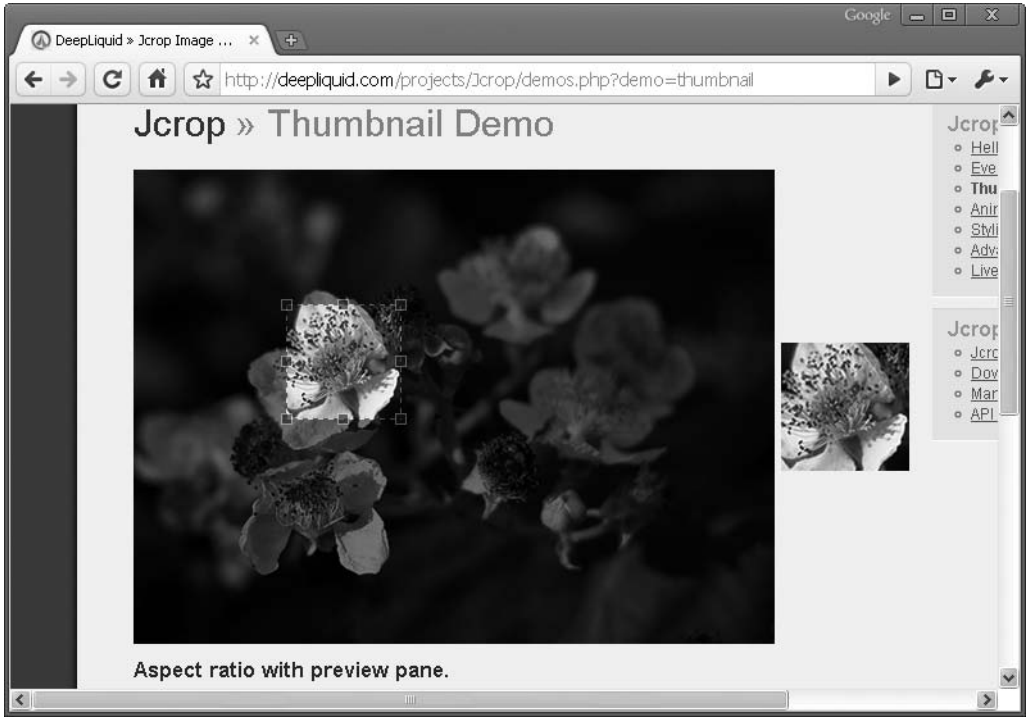


Figure 10.20

Démonstration du plugin Jcrop.

Magnify

- **http://www.jnathanson.com/index.cfm?page=jquery/magnify/magnify**
- **http://plugins.jquery.com/project/magnify**

Si on lui fournit une image réduite proportionnellement et une grande image, le plugin Magnify génère une "loupe" comme celles couramment employées pour afficher les détails d'un produit et les zooms (voir Figure 10.21).

Figure 10.21

Démonstration du plugin Magnify.

Lightbox et boîte de dialogue modales

Nos exemples du Chapitre 9 ont expliqué comment afficher des informations de détail au-dessus d'une page sans utiliser une fenêtre *popup*. Cette fonctionnalité est souvent appelée *lightbox*. Les plugins suivants facilitent la création de ces fenêtres d'affichage superposées.

FancyBox

- **http://fancy.klade.lv/**

Ce plugin pour lightbox met l'accent sur le style, avec son aspect Mac et son élégant effet d'ombre portée (voir Figure 10.22, page suivante). Outre l'affichage automatique des images redimensionnées, FancyBox peut prendre en charge du contenu en ligne ou `<iframe>`.

Figure 10.22
Démonstration du plugin FancyBox.

Thickbox

- **http://jquery.com/demo/thickbox/**

Thickbox est un plugin polyvalent pour lightbox qui peut afficher une seule image, plusieurs images, du contenu en ligne, du contenu `<iframe>` ou du contenu obtenu par une requête AJAX, dans une boîte de dialogue modale personnalisée (voir Figure 10.23).

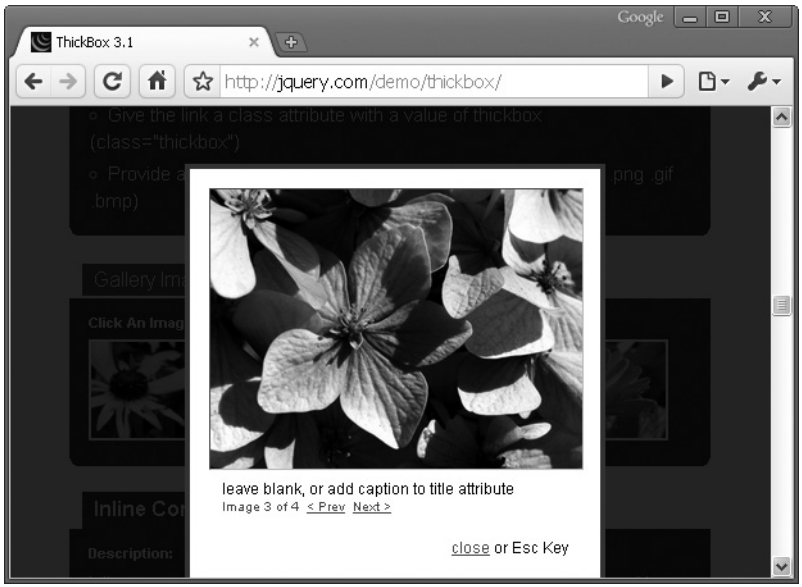


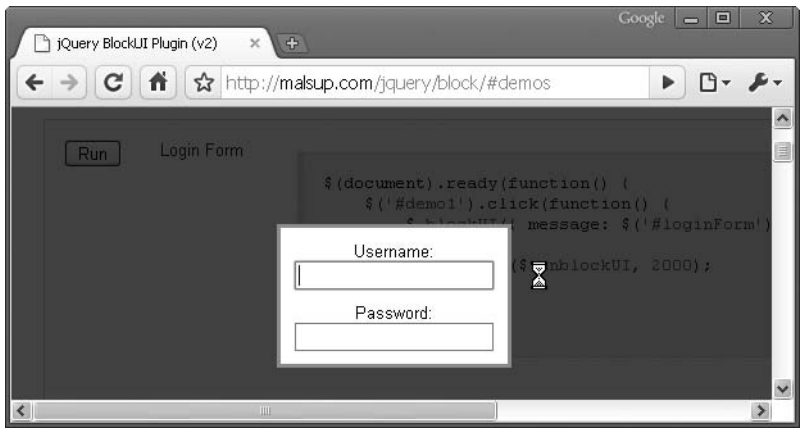
Figure 10.23
Démonstration du plugin Thickbox.

BlockUI

- **http://malsup.com/jquery/block/**
- **http://plugins.jquery.com/project/blockUI**

Le plugin BlockUI simule un comportement synchrone, sans bloquer le navigateur. Lorsqu'il est activé, il empêche l'utilisateur d'interagir avec l'intégralité ou une partie de la page, jusqu'à ce qu'il soit désactivé (voir Figure 10.24).

Figure 10.24
Démonstration du plugin BlockUI.



jqModal

- **http://dev.iceburg.net/jquery/jqModal/**
- **http://plugins.jquery.com/project/jqModal**

Le plugin jqModal apporte une solution légère pour la création de boîtes de dialogue modales, tout en restant puissant et flexible (voir Figure 10.25, page suivante). En mettant l'accent sur les possibilités d'extension, il laisse une grande partie de la gestion des interactions et de la définition des styles aux développeurs web qui l'utilisent.

Création de graphiques

À l'instar de la manipulation des images, la création de graphiques est une activité traditionnellement effectuée sur le serveur et nécessitant un traitement important. Des programmeurs ingénieux ont développé plusieurs solutions pour créer des graphiques dans le navigateur et ont regroupé ces techniques dans les plugins présentés ici.

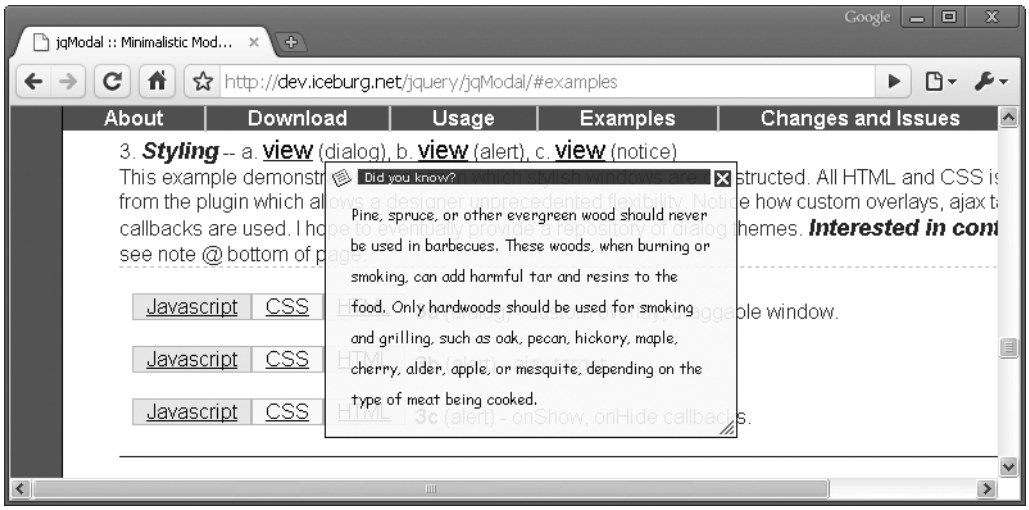


Figure 10.25

Démonstration du plugin jqModal.

Flot

- **http://code.google.com/p/flot/**
- **http://plugins.jquery.com/project/flot**

Le plugin Flot se sert de l'élément `<canvas>` pour produire des graphiques à partir de jeux de données et permet à l'utilisateur de les modifier (voir Figure 10.26). En ajoutant le script Excanvas fourni, Flot devient compatible avec Internet Explorer, par traduction des instructions de Canvas au format VML d'Internet Explorer.

Sparklines

- **http://omnipotent.net/jquery.sparkline/**
- **http://plugins.jquery.com/project/sparklines**

Nommé d'après un concept popularisé par Edward Tufte, expert en visualisation des données, le plugin Sparklines génère des minigraphiques simples en ligne (voir Figure 10.27). Comme Flot, Sparklines utilise l'élément `<canvas>` pour générer les graphiques, mais la compatibilité avec Internet Explorer est prise en charge par le plugin lui-même, au lieu de se fonder sur Excanvas.

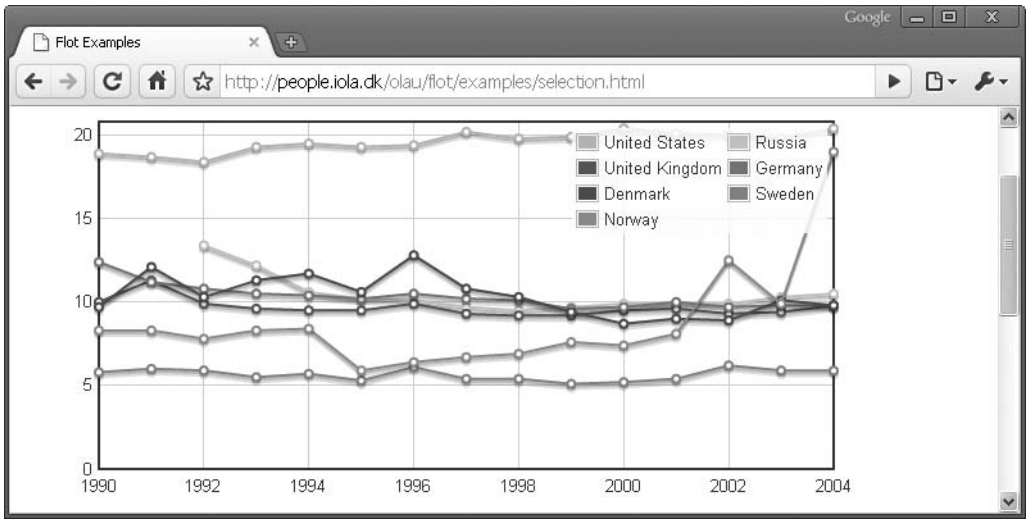


Figure 10.26

Démonstration du plugin Flot.

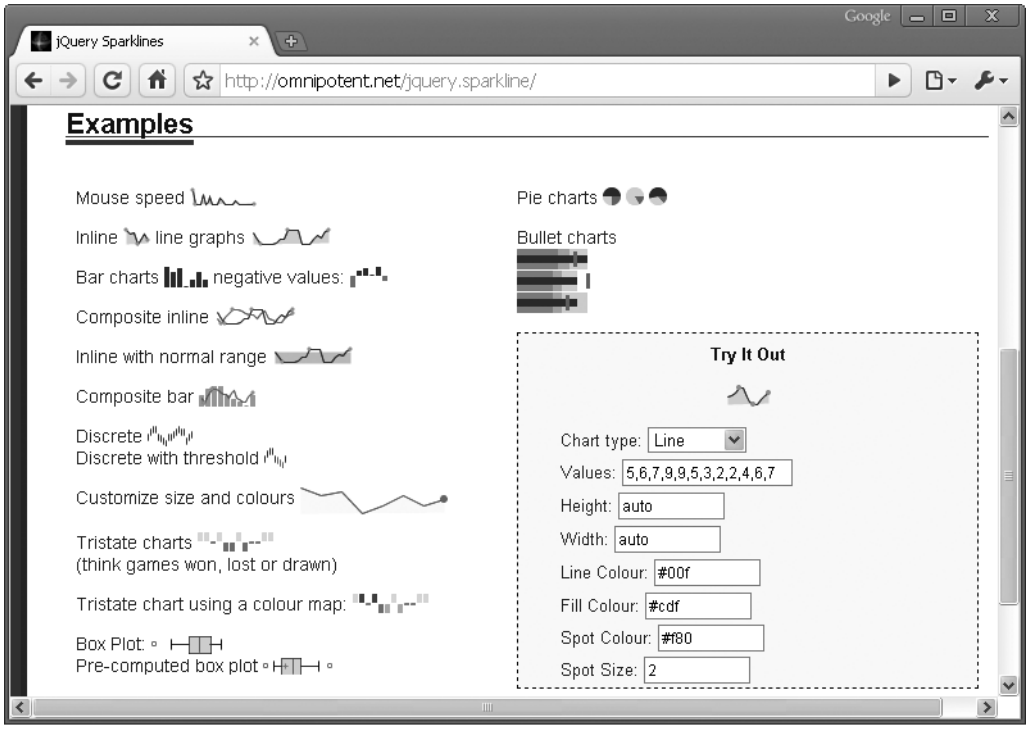


Figure 10.27

Démonstration du plugin Sparklines.

Événements

Nous l'avons vu de nombreuses fois, jQuery propose tout un ensemble d'outils pour intercepter et réagir aux événements provenant de l'utilisateur, comme les clics de souris et les appuis sur les touches. Cependant, même si la bibliothèque standard propose un grand nombre d'options, il existe toujours des techniques plus élaborées à explorer. Les plugins suivants facilitent la mise en œuvre d'une gestion des événements moins commune.

hoverIntent

- **http://cherne.net/brian/resources/jquery.hoverIntent.html**
- **http://plugins.jquery.com/project/hoverIntent**

Le plugin hoverIntent fournit une seule méthode, qui remplace `.hover()` lorsqu'il est important d'éviter le déclenchement accidentel d'animations au moment où le pointeur de la souris survole ou quitte un élément. Elle tente de déterminer les intentions de l'utilisateur en surveillant la modification de la vitesse du pointeur de la souris. Ce plugin est particulièrement efficace lorsqu'il est employé conjointement à une navigation par menus déroulants.

Live Query

- **http://github.com/brandonaaron/livequery/**
- **http://plugins.jquery.com/project/livequery**

À l'instar de la méthode `.live()` intégrée à jQuery, le plugin Live Query associe et maintient dynamiquement des liaisons d'événements dans le DOM, quel que soit le moment de création des éléments. Il constitue une autre mise en œuvre, qui se révèle plus appropriée dans certains cas.

10.6 En résumé

Dans ce chapitre, nous avons vu comment employer des plugins tiers avec nos pages web. Nous avons détaillé les plugins Form et jQuery UI. Nous avons également mentionné quelques autres plugins intéressants. Au chapitre suivant, nous allons exploiter l'architecture d'extension de jQuery pour développer différents plugins de notre cru.

11

Développement de plugins

Au sommaire de ce chapitre

- Ajouter de nouvelles fonctions globales
- Ajouter des méthodes à l'objet jQuery
- Méthodes de parcours du DOM
- Ajouter des méthodes abrégées
- Paramètres d'une méthode
- Ajouter une expression de sélection
- Distribuer un plugin
- En résumé

Les plugins tiers disponibles apportent une multitude de solutions pour améliorer nos scripts, mais nous devons parfois aller plus loin. Lorsque nous écrivons du code qui pourra être réutilisé par d'autres personnes, ou même par nous, nous devons le proposer sous forme de plugin. Heureusement, le développement d'un plugin ne demande pas un travail beaucoup plus important que l'écriture du code qui l'utilise.

Dans ce chapitre, nous allons examiner la création de différentes sortes de plugins, du plus simple au plus complexe. Nous commencerons par des plugins qui offrent de nouvelles fonctions globales, puis nous fournirons différents types de méthodes de l'objet jQuery. Nous verrons également comment ajouter de nouvelles expressions au moteur de sélection de jQuery et conclurons par quelques conseils à propos de la distribution d'un plugin aux autres développeurs.

11.1 Ajouter de nouvelles fonctions globales

Certaines fonctionnalités intégrées à jQuery sont disponibles au travers de *fonctions globales*. Nous l'avons vu, il s'agit en réalité de *méthodes* de l'objet jQuery, mais, d'un point de vue pratique, ces fonctions sont placées dans l'*espace de noms* `jQuery`.

La fonction `$.ajax()` en est un premier exemple. Toutes ses possibilités peuvent être offertes au travers d'une fonction globale normale nommée `ajax()`, mais cette solution nous soumet aux conflits sur les noms des fonctions. En plaçant la fonction dans l'espace de noms `jQuery`, nous n'avons plus à nous préoccuper des conflits potentiels avec d'autres méthodes de jQuery.

Pour ajouter une fonction dans l'espace de noms `jQuery`, il suffit d'affecter la nouvelle fonction en tant que *propriété* de l'objet jQuery :

```
jQuery.fonctionGlobale = function() {
  alert('Un test, juste un test.');
};
```

Ensuite, dans le code qui utilise ce plugin, nous pouvons invoquer la fonction :

```
jQuery.fonctionGlobale();
```

Nous pouvons également employer l'alias `$` pour l'appel de la fonction :

```
$.fonctionGlobale();
```

Le fonctionnement est identique à un appel de fonction normale, conduisant à l'affichage du message d'alerte.

Ajouter plusieurs fonctions

Si notre plugin doit proposer plusieurs fonctions globales, nous pouvons les déclarer indépendamment :

```
jQuery.fonctionUne = function() {
  alert('Un test, juste un test.');
};
jQuery.fonctionDeux = function(param) {
  alert('Le paramètre est "' + param + '".');
};
```

Les deux méthodes étant définies, nous pouvons les invoquer de manière normale :

```
$.fonctionUne();
$.fonctionDeux('test');
```

Il existe également une autre syntaxe pour définir nos fonctions, fondée sur la fonction `$.extend()` :

```
jQuery.extend({
  fonctionUne: function() {
    alert('Un test, juste un test.');
  },
  fonctionDeux: function(param) {
    alert('Le paramètre est "' + param + '".');
  }
});
```

Le résultat est identique.

Toutefois, nous prenons un nouveau risque concernant la pollution de l'espace de noms. Même si l'utilisation de l'espace de noms `jQuery` nous protège des conflits de noms avec la plupart des fonctions et des variables JavaScript, il n'en est rien avec les noms des fonctions définies par d'autres plugins jQuery. Pour éviter ce problème, il est préférable d'encapsuler toutes les fonctions globales d'un plugin dans un objet :

```
jQuery.monPlugin = {
  fonctionUne: function() {
    alert('Un test, juste un test.');
  },
  fonctionDeux: function(param) {
    alert('Le paramètre est "' + param + '".');
  }
};
```

Cette approche crée pour nos fonctions globales un autre espace de noms nommé `jQuery.monPlugin`. Même si nous continuons à prétendre informellement que ces fonctions sont "globales", il s'agit en réalité de méthodes de l'objet `monPlugin`, lui-même une propriété de l'objet `jQuery` global. Par conséquent, nous devons inclure le nom du plugin dans les appels à nos fonctions :

```
$.monPlugin.fonctionUne();
$.monPlugin.fonctionDeux('test');
```

Grâce à cette technique, et à un nom de plugin suffisamment unique, nous sommes totalement protégés contre les conflits d'espace de noms dans nos fonctions globales.

Où est l'intérêt ?

Nous possédons à présent les bases du développement de plugins. Après avoir enregistré nos fonctions dans un fichier nommé `jquery.monplugin.js`, nous pouvons inclure ce script et utiliser nos fonctions à partir d'autres scripts de la page. Mais en quoi est-ce différent d'un autre fichier JavaScript que nous pouvons créer et inclure ?

Nous avons déjà mentionné les avantages de regrouper notre code dans l'espace de noms de l'objet `jQuery`. Par ailleurs, le développement de notre bibliothèque de fonctions sous forme d'extension jQuery présente un autre avantage : puisque nous savons que jQuery est inclus, les fonctions peuvent utiliser cette bibliothèque.

INFO

Même si jQuery est inclus, vous ne pouvez pas supposer que le raccourci `$` soit disponible. N'oubliez pas que la méthode `$.noConflict()` peut libérer le contrôle de ce raccourci. Vos plugins doivent donc toujours invoquer les méthodes de jQuery en utilisant `jQuery` ou définir eux-mêmes le raccourci $, comme nous l'expliquerons plus loin.

Créer une méthode utilitaire

Bon nombre des fonctions globales fournies par la bibliothèque jQuery standard sont des *méthodes utilitaires*. Autrement dit, elles proposent des raccourcis pour effectuer facilement des tâches fréquentes. Les méthodes de manipulation des tableaux `$.each()`, `$.map()` et `$.grep()` en sont de bons exemples. Pour illustrer la création de telles méthodes, nous allons ajouter `$.sum()`.

Notre nouvelle méthode prend en argument un tableau, effectue la somme des valeurs qu'il contient et retourne le résultat. Le code du plugin est plutôt court :

```
jQuery.sum = function(array) {
  var total = 0;

  jQuery.each(array, function(index, value) {
    total += value;
  });

  return total;
};
```

Vous remarquerez que nous avons utilisé la méthode `$.each()` pour parcourir le tableau. Nous pourrions certainement utiliser une simple boucle `for()` ici mais, puisque nous sommes certains que la bibliothèque jQuery a été chargée avant notre plugin, nous pouvons opter pour la syntaxe à laquelle nous sommes habitués.

Pour tester notre plugin, nous construisons une page simple qui affiche les entrées et les sorties de la fonction :

```
<body>
  <p>Contenu du tableau :</p>
  <ul id="array-contents"></ul>
  <p>Somme des valeurs :</p>
  <div id="array-sum"></div>
</body>
```

Nous écrivons également un court script qui place les valeurs du tableau et leur somme dans les emplacements que nous avons définis :

```
$(document).ready(function() {
  var myArray = [52, 97, 0.5, -22];
  $.each(myArray, function(index, value) {
    $('#array-contents').append('<li>' + value + '</li>');
  });
  $('#array-sum').append($.sum(myArray));
});
```

La Figure 11.1 présente la page HTML qui valide le fonctionnement de notre plugin.

Nous savons que les espaces de noms apportent une protection et que les plugins jQuery garantissent la disponibilité de la bibliothèque. Toutefois, il s'agit uniquement d'avantages organisationnels. Pour réellement profiter de la puissance des plugins jQuery, nous devons apprendre à créer de nouvelles *méthodes* sur des instances individuelles de l'objet jQuery.

Figure 11.1
Notre premier plugin est opérationnel.

11.2 Ajouter des méthodes à l'objet jQuery

La plupart des fonctionnalités intégrées à jQuery sont fournies par ses méthodes d'objet. C'est dans ce contexte que les plugins prennent tout leur sens. Dès que nous envisageons d'écrire une fonction qui opère sur le DOM, nous devons songer à créer à la place une méthode d'objet.

Nous avons vu que l'ajout de fonctions globales passe par l'extension de l'objet `jQuery` avec de nouvelles méthodes. L'ajout de méthodes d'instance est comparable, mais il concerne l'objet `jQuery.fn` :

```
jQuery.fn.maMethode = function() {
  alert('Rien ne se produit.');
}
```

INFO

L'objet `jQuery.fn` est un alias de `jQuery.prototype`, proposé pour plus de concision.

Nous pouvons ensuite invoquer cette nouvelle méthode depuis notre code après une expression de sélection :

```
$('div').maMethode();
```

Cette invocation affiche le message d'alerte. Mais, puisque nous ne manipulons pas les nœuds du DOM obtenus, nous aurions très bien pu écrire une fonction globale. La mise en œuvre d'une méthode agit normalement sur son *contexte*.

Contexte d'une méthode d'objet

Dans une méthode de plugin, le mot clé `this` désigne l'objet jQuery qui a invoqué la méthode. Par conséquent, nous pouvons invoquer n'importe quelle méthode jQuery standard sur `this` ou extraire ses nœuds du DOM et les manipuler :

```
jQuery.fn.showAlert = function() {
  alert('Vous avez sélectionné ' + this.length + ' éléments.');
}
```

Pour étudier les possibilités offertes par le contexte d'objet, nous allons écrire un petit plugin qui manipule les classes des éléments sélectionnés. Notre nouvelle méthode prend en argument deux noms de classes et les échange sur chaque élément :

```
jQuery.fn.swapClass = function(class1, class2) {
  if (this.hasClass(class1)) {
    this.removeClass(class1).addClass(class2);
  }
  else if (this.hasClass(class2)) {
    this.removeClass(class2).addClass(class1);
  }
};
```

Nous commençons par tester si `class1` est affectée à l'élément obtenu et, dans l'affirmative, la remplaçons par `class2`. Sinon nous testons la présence de `class2` et lui substituons `class1` si nécessaire. Si aucune des classes n'est présente, nous ne faisons rien.

Pour tester notre méthode, nous avons besoin d'un contenu HTML :

```
<ul>
  <li>Lorem ipsum dolor sit amet</li>
  <li class="this">Consectetur adipisicing elit</li>
  <li>Sed do eiusmod tempor incididunt ut labore</li>
  <li class="that">Magna aliqua</li>
  <li class="this">Ut enim ad minim veniam</li>
  <li>Quis nostrud exercitation ullamco</li>
  <li>Laboris nisi ut aliquip ex ea commodo</li>
  <li class="that">Duis aute irure dolor</li>
</ul>
<input type="button" value="Échanger les classes" id="swap" />
```

Les éléments de classe `this` sont affichés en gras, tandis que ceux de classe `that` sont affichés en italique (voir Figure 11.2).

Figure 11.2

Aspect initial de la page.

Un clic sur le bouton invoque notre méthode :

```
$(document).ready(function() {
  $('#swap').click(function() {
    $('li').swapClass('this', 'that');
```

```
    return false;
  });
});
```

Mais cela ne fonctionne pas correctement. En cliquant sur le bouton, la classe `that` est appliquée à toutes les lignes (voir Figure 11.3).

Figure 11.3

Le premier essai n'est pas concluant.

Nous avons oublié qu'une expression de sélection jQuery peut correspondre à zéro, un ou plusieurs éléments. Nous devons tenir compte de ces différents cas lors de la conception d'une méthode de plugin. Dans notre exemple, nous invoquons `.hasClass()`, qui examine uniquement le premier élément sélectionné. À la place, nous devons vérifier et opérer sur chaque élément de manière indépendante.

Pour garantir un comportement correct quel que soit le nombre d'éléments sélectionnés, la solution la plus simple consiste à toujours appeler `.each()` sur le contexte de méthode. Nous imposons ainsi une *itération implicite*, qui est importante pour la coopération entre les méthodes de plugin et les méthodes intégrées. Dans l'appel à `.each()`, `this` fait référence à chaque élément du DOM tour à tour. Nous pouvons ainsi corriger notre code pour tester séparément chaque élément sélectionné et appliquer les classes :

```
jQuery.fn.swapClass = function(class1, class2) {
  this.each(function() {
    var $element = jQuery(this);
    if ($element.hasClass(class1)) {
      $element.removeClass(class1).addClass(class2);
    }
    else if ($element.hasClass(class2)) {
      $element.removeClass(class2).addClass(class1);
    }
  });
};
```

ATTENTION

Puisque le mot clé `this` représente l'objet qui a appelé la méthode, il fait référence à un *objet jQuery* dans le corps de la méthode d'objet, mais fait référence à un *élément du DOM* dans l'invocation de `.each()`.

À présent, lorsque nous cliquons sur le bouton, les classes sont échangées sans affecter les éléments auxquels aucune des deux classes n'est attribuée (voir Figure 11.4).

Figure 11.4

L'échange des classes fonctionne.

Chaîner des méthodes

Outre l'itération implicite, les utilisateurs de jQuery doivent pouvoir s'appuyer sur le *chaînage*. Autrement dit, toutes les méthodes de plugin doivent retourner un objet jQuery, à moins qu'une méthode ne soit clairement conçue pour retrouver un autre élément d'information. L'objet jQuery retourné est habituellement celui désigné par `this`. Si nous utilisons `.each()` pour itérer sur `this`, nous pouvons simplement le retourner en résultat :

```
jQuery.fn.swapClass = function(class1, class2) {
  return this.each(function() {
    var $element = jQuery(this);
    if ($element.hasClass(class1)) {
      $element.removeClass(class1).addClass(class2);
    }
    else if ($element.hasClass(class2)) {
      $element.removeClass(class2).addClass(class1);
    }
  });
};
```

Précédemment, lorsque nous avons invoqué `.swapClass()`, nous avons commencé une nouvelle instruction pour manipuler les éléments. Grâce à l'instruction de retour, nous pouvons à présent chaîner des méthodes intégrées à notre méthode de plugin :

```
$(document).ready(function() {
  $('#swap').click(function() {
    $('li')
      .swapClass('this', 'that')
      .css('text-decoration', 'underline');
    return false;
  });
});
```

La Figure 11.5 montre que le contenu de la liste est bien souligné après échange des classes `this` et `that`.

Figure 11.5

La méthode de plugin est compatible avec le chaînage.

11.3 Méthodes de parcours du DOM

Dans certains cas, nous voulons que la méthode de plugin modifie les éléments du DOM référencés par l'objet jQuery. Par exemple, supposons que nous ajoutions une méthode de parcours du DOM qui recherche les grands-parents des éléments obtenus :

```
jQuery.fn.grandparent = function() {
  var grandparents = [];
  this.each(function() {
    grandparents.push(this.parentNode.parentNode);
  });
  grandparents = jQuery.unique(grandparents);
  return this.setArray(grandparents);
};
```

Cette méthode crée un nouveau tableau `grandparents` et le remplit en itérant sur tous les éléments actuellement référencés par l'objet jQuery. Nous utilisons la propriété DOM standard `.parentNode` pour rechercher les éléments grands-parents, qui sont alors ajoutés au tableau `grandparents`. Les doublons éventuels en sont retirés par un appel à `$.unique()`, puis la méthode jQuery `.setArray()` affecte le nouveau tableau en tant que collection des éléments sélectionnés. Ainsi, par un seul appel de méthode, nous pouvons rechercher et manipuler les grands-parents d'un élément.

Pour tester notre méthode, nous construisons une structure `<div>` arborescente :

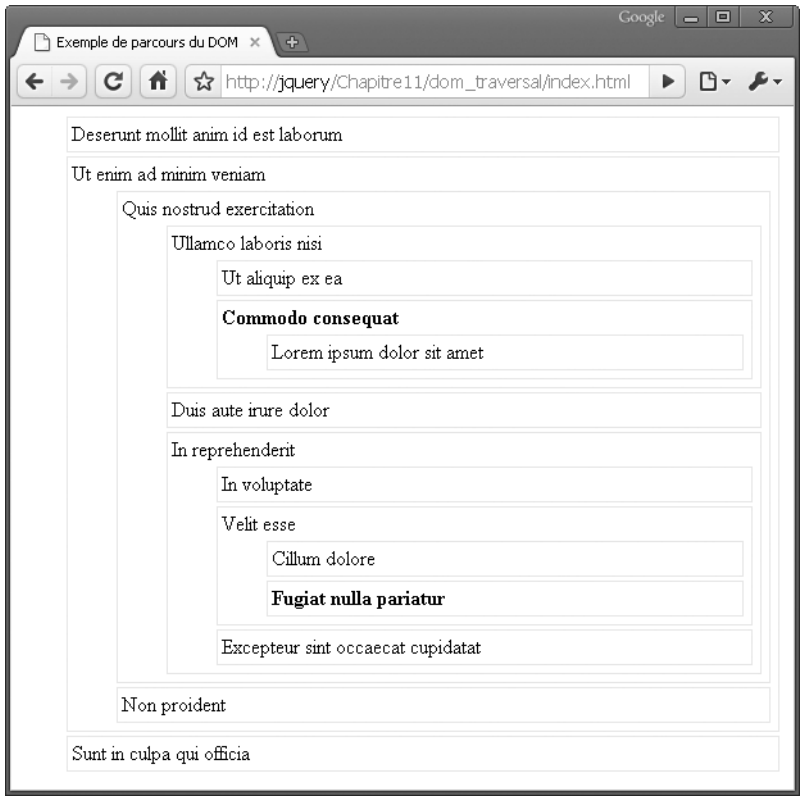
```
<div>Deserunt mollit anim id est laborum</div>
<div>Ut enim ad minim veniam
  <div>Quis nostrud exercitation
    <div>Ullamco laboris nisi
      <div>Ut aliquip ex ea</div>
      <div class="target">Commodo consequat
        <div>Lorem ipsum dolor sit amet</div>
      </div>
    </div>
```

```
      <div>Duis aute irure dolor</div>
      <div>In reprehenderit
        <div>In voluptate</div>
        <div>Velit esse
          <div>Cillum dolore</div>
          <div class="target">Fugiat nulla pariatur</div>
        </div>
        <div>Excepteur sint occaecat cupidatat</div>
      </div>
    </div>
    <div>Non proident</div>
  </div>
  <div>Sunt in culpa qui officia</div>
```

Les éléments cibles (`<div class="target">`) sont identifiables à leur texte en gras (voir Figure 11.6).

Figure 11.6

L'arborescence pour les tests.



Nous pouvons retrouver les grands-parents des éléments en utilisant notre nouvelle méthode :

```
$(document).ready(function() {
  $('.target').grandparent().addClass('highlight');
});
```

La classe `highlight` met en italique les deux grands-parents des éléments cibles (voir Figure 11.7).

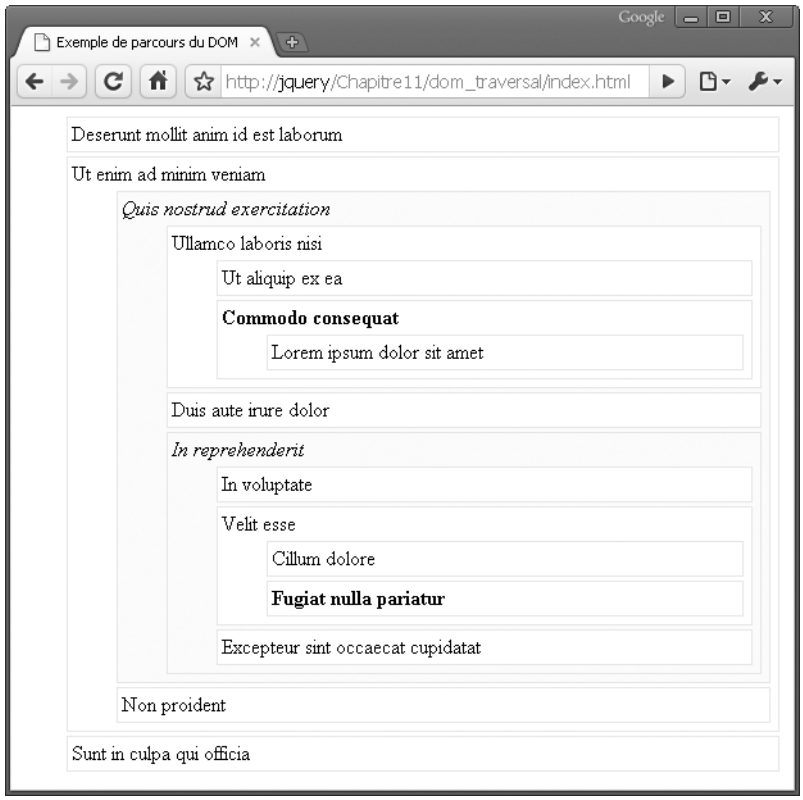


Figure 11.7

Les éléments grands-parents sont correctement trouvés.

Toutefois, cette méthode est *destructive* : l'objet jQuery est modifié. L'effet devient évident si nous enregistrons l'objet jQuery dans une variable :

```
$(document).ready(function() {
  var $target = $('.target');
  $target.grandparent().addClass('highlight');
  $target.hide();
});
```

Ce code est censé mettre en exergue les éléments grands-parents, puis masquer les éléments cibles. Cependant, il masque les grands-parents (voir Figure 11.8).

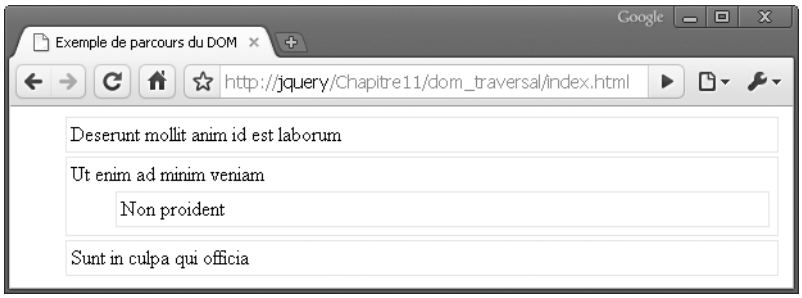


Figure 11.8

Illustration de la modification de l'objet jQuery.

L'objet jQuery enregistré dans la variable `$target` a été modifié par la recherche des grands-parents. Pour éviter cela, nous devons faire en sorte que la méthode ne soit pas destructive. Nous utilisons donc la *pile* interne de jQuery associée à chaque objet :

```
jQuery.fn.grandparent = function() {
  var grandparents = [];
  this.each(function() {
    grandparents.push(this.parentNode.parentNode);
  });
  grandparents = jQuery.unique(grandparents);
  return this.pushStack(grandparents);
};
```

En appelant `.pushStack()` à la place de `.setArray()`, nous créons un nouvel objet jQuery au lieu de modifier l'ancien. À présent, l'objet `$target` n'est pas modifié et les objets cibles d'origine sont masqués par notre code (voir Figure 11.9).

Par ailleurs, grâce à `.pushStack()`, notre plugin devient compatible avec les méthodes `.end()` et `.andSelf()`. Nous pouvons donc les chaîner correctement. Le code suivant ajoute la classe `highlight` aux éléments cibles (voir Figure 11.10) :

```
$(document).ready(function() {
  $('.target').grandparent().andSelf().addClass('highlight');
});
```

INFO

Les méthodes de parcours du DOM, comme `.children()`, étaient des opérations destructives dans jQuery 1.0. Elles ne le sont plus depuis la version 1.1.

11.4 Ajouter des méthodes abrégées

De nombreuses méthodes fournies par jQuery sont en réalité des *raccourcis* vers d'autres méthodes sous-jacentes. Par exemple, la plupart des *méthodes d'événement* sont des raccourcis pour `.bind()` ou `.trigger()`, et plusieurs méthodes AJAX invoquent en interne `$.ajax()`. Grâce à ces méthodes abrégées, il est plus pratique d'utiliser des fonctionnalités qui passeraient sinon par de nombreuses options complexes.

Figure 11.9

L'objet jQuery n'est plus modifié et les objets cibles sont masqués.

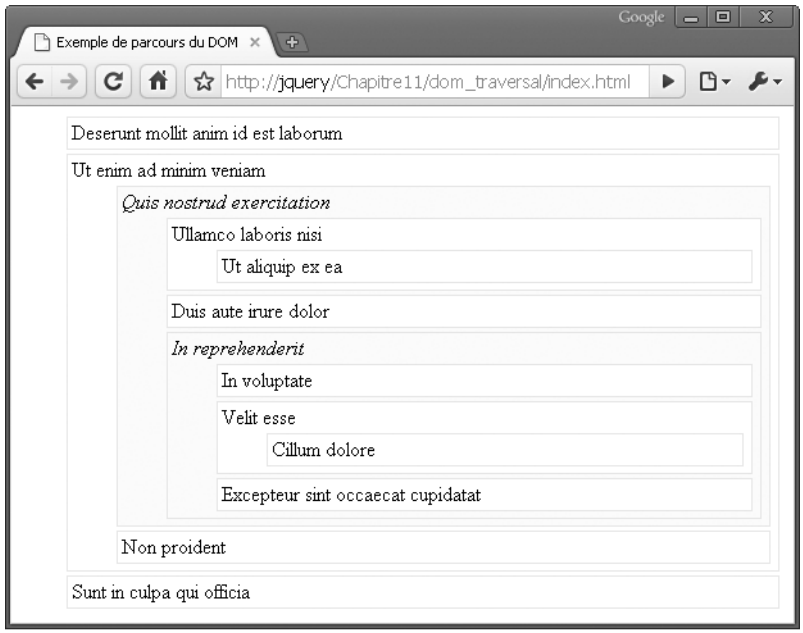
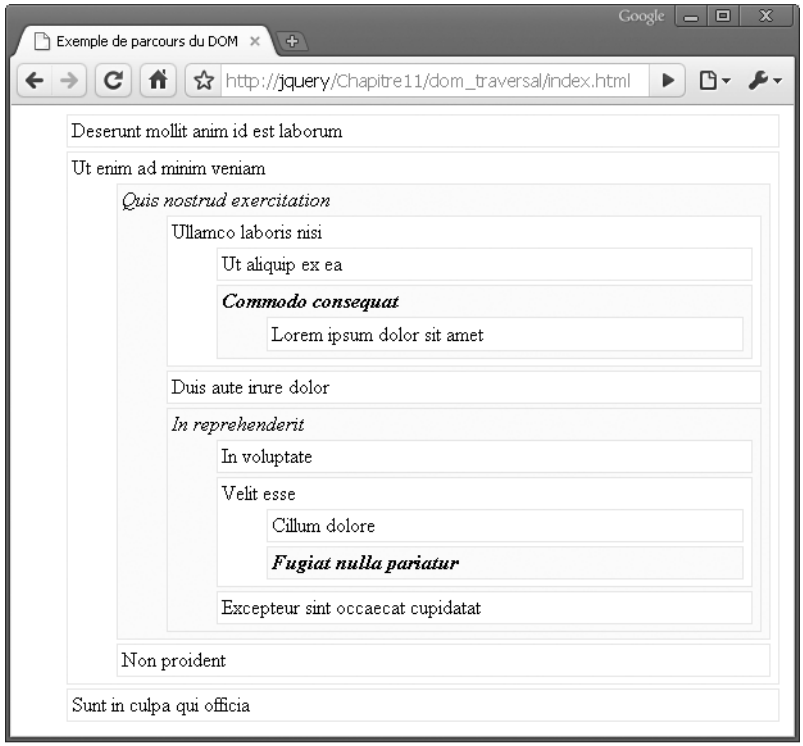


Figure 11.10

Le plugin est compatible avec la méthode `.andSelf()`.



La bibliothèque jQuery doit conserver un équilibre précaire entre commodité et complexité. Chaque méthode ajoutée à la bibliothèque peut aider les développeurs à écrire plus rapidement certaines parties du code, mais la taille de la base de code augmente et les performances risquent de baisser. C'est pourquoi de nombreuses méthodes abrégées pour une fonctionnalité interne sont placées dans des plugins. Nous pouvons ainsi choisir celles qui sont utiles à chaque projet et omettre celles qui ne présentent pas d'intérêt.

Lorsque l'on constate qu'un idiome se répète dans le code, la création d'une méthode abrégée s'impose. Par exemple, supposons que nous animions fréquemment des éléments en employant une combinaison des techniques intégrées de "glissement" et de "fondu". La combinaison de ces effets signifie que la hauteur et l'opacité d'un élément doivent être animées simultanément. Grâce à la méthode `.animate()`, cette opération est simple :

```
.animate({height: 'hide', opacity: 'hide'});
```

Nous pouvons alors créer trois méthodes abrégées pour réaliser cette animation lors de l'affichage et du masquage des éléments :

```
jQuery.fn.slideFadeOut = function() {
  return this.animate({
    height: 'hide',
    opacity: 'hide'
  });
};

jQuery.fn.slideFadeIn = function() {
  return this.animate({
    height: 'show',
    opacity: 'show'
  });
};

jQuery.fn.slideFadeToggle = function() {
  return this.animate({
    height: 'toggle',
    opacity: 'toggle'
  });
};
```

Nous pouvons à présent appeler `.slideFadeOut()` dès que c'est nécessaire pour déclencher l'animation. Puisque, dans la définition d'une méthode de plugin, `this` fait référence à l'objet jQuery courant, l'animation concernera tous les éléments sélectionnés à la fois.

Pour un plugin complet, les nouvelles méthodes doivent prendre en charge les mêmes paramètres que les méthodes abrégées standard. En particulier, des méthodes comme `.fadeIn()` peuvent prendre en arguments une vitesse et une fonction de rappel. Puisque `.animate()` reconnaît également ces paramètres, leur prise en charge dans nos méthodes est simple. Nous les acceptons simplement en argument et les transmettons à `.animate()` :

```
jQuery.fn.slideFadeOut = function(speed, callback) {
  return this.animate({
    height: 'hide',
    opacity: 'hide'
  }, speed, callback);
};

jQuery.fn.slideFadeIn = function(speed, callback) {
  return this.animate({
    height: 'show',
    opacity: 'show'
  }, speed, callback);
};

jQuery.fn.slideFadeToggle = function(speed, callback) {
  return this.animate({
    height: 'toggle',
    opacity: 'toggle'
  }, speed, callback);
};
```

Nous disposons désormais de méthodes abrégées personnalisées qui se comportent comme leurs équivalents intégrés. Pour illustrer leur fonctionnement, nous avons besoin d'une simple page HTML :

```
<body>
  <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
    tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
    veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
    commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
    velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
    cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id
    est laborum.</p>
  <div class="controls">
    <input type="button" value="Glissement et fondu fermé" id="out" />
    <input type="button" value="Glissement et fondu ouvert" id="in" />
    <input type="button" value="Inversion" id="toggle" />
  </div>
</body>
```

Notre script invoque simplement les nouvelles méthodes lors des clics sur les boutons :

```
$(document).ready(function() {
  $('#out').click(function() {
    $('p').slideFadeOut('slow');
    return false;
  });
  $('#in').click(function() {
    $('p').slideFadeIn('slow');
    return false;
  });
  $('#toggle').click(function() {
    $('p').slideFadeToggle('slow');
    return false;
  });
});
```

La Figure 11.11 montre en trois étapes que l'animation se passe comme attendu.

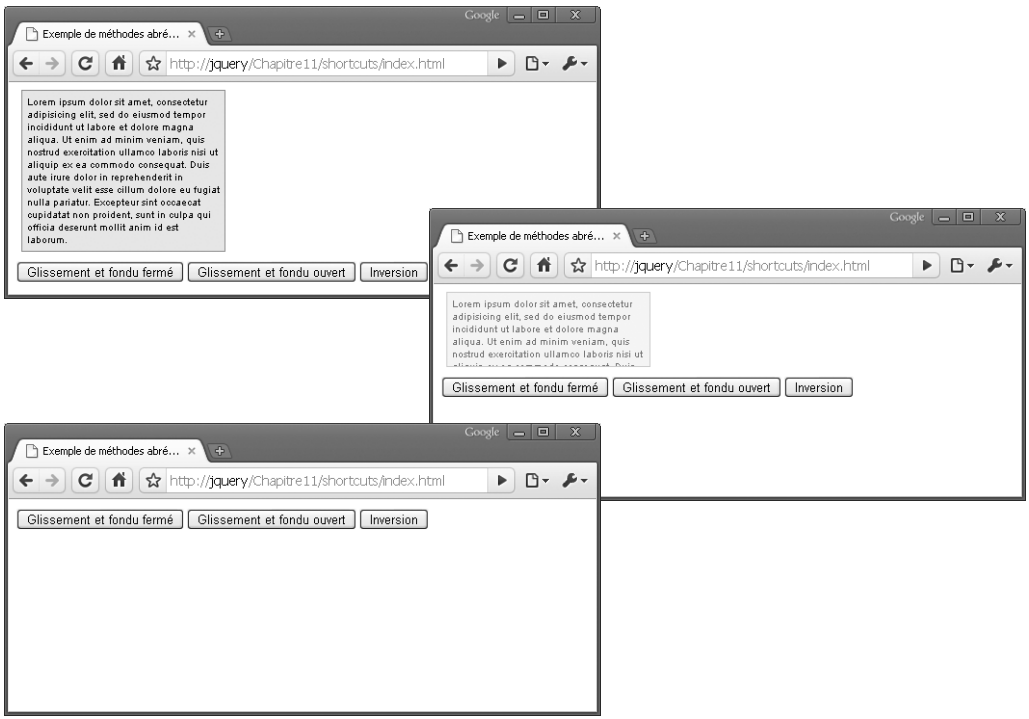


Figure 11.11

Animation de la hauteur et de l'opacité du bloc de texte par une méthode abrégée.

11.5 Paramètres d'une méthode

Nous venons d'étudier plusieurs exemples de méthodes de plugin, certaines prenant des paramètres explicites. Nous l'avons vu, le contexte de la méthode est toujours disponible au travers du mot clé `this`, mais nous pouvons également fournir des informations supplémentaires pour influencer le fonctionnement de la méthode. Jusqu'à présent, les paramètres étaient peu nombreux, mais la liste peut s'allonger énormément. Pour gérer les paramètres d'une méthode et pour faciliter le travail des utilisateurs de nos plugins, il existe plusieurs astuces.

Notre premier exemple sera celui d'une méthode de plugin qui ajoute une ombre à un bloc de texte. Nous utilisons une technique semblable à celle employée au Chapitre 9 pour l'effet de fondu sur le prompteur des nouvelles : des éléments partiellement transparents sont affichés par superposition en différents endroits de la page.

```
jQuery.fn.shadow = function() {
  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < 5; i++) {
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left + i,
          top: $originalElement.offset().top + i,
          margin: 0,
          zIndex: -1,
          opacity: 0.1
        })
        .appendTo('body');
    }
  });
};
```

Pour chaque élément sur lequel cette méthode est invoquée, nous créons des copies de l'élément, en modifiant leur opacité. Ces clones sont positionnés de manière absolue, avec un décalage variable à partir de l'élément d'origine.

Nous allons tester notre plugin sur un contenu HTML simple (voir Figure 11.12) :

```
<body>
  <h1>Zoe ma grande fille veut que je boive ce whisky dont je ne veux pas.</h1>
</body>
```

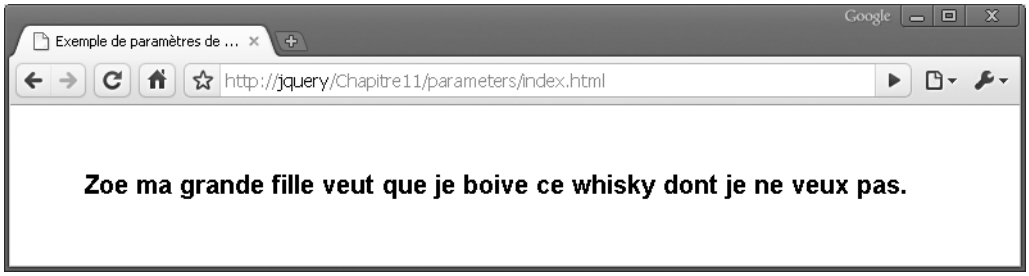


Figure 11.12

La phrase de texte pour nos tests.

Pour le moment, notre méthode de plugin ne prend aucun paramètre et son invocation est donc simple, avec le résultat illustré à la Figure 11.13 :

```
$(document).ready(function() {
  $('h1').shadow();
});
```

Figure 11.13

Invocation de la méthode du plugin sans paramètres.

Paramètres simples

Nous allons à présent ajouter une certaine flexibilité à la méthode du plugin. Le fonctionnement de cette méthode se fonde sur plusieurs valeurs numériques que l'utilisateur pourrait souhaiter modifier. Pour cela, nous les transformons en paramètres :

```
jQuery.fn.shadow = function(slices, opacity, zIndex) {
  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < slices; i++) {
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left + i,
          top: $originalElement.offset().top + i,
          margin: 0,
          zIndex: zIndex,
          opacity: opacity
        })
        .appendTo('body');
    }
  });
};
```

Pour invoquer notre méthode, nous devons à présent fournir ces trois valeurs (voir Figure 11.14) :

```
$(document).ready(function() {
  $('h1').shadow(10, 0.1, -1);
});
```

Nos paramètres ont l'effet souhaité – l'ombre est plus longue et utilise deux fois plus de copies que précédemment –, mais l'interface de la méthode n'est pas vraiment idéale. Il est très facile de confondre les trois nombres et il est impossible de déduire logiquement leur ordre. Il est préférable de libeller les paramètres, tant pour la personne qui appelle la méthode que pour celle qui devra lire et interpréter le code ultérieurement.

Figure 11.14

La méthode avec trois paramètres permet de varier l'effet.

Mappes de paramètres

Dans l'API de jQuery, nous avons rencontré plusieurs exemples de *mappes* servant à passer des paramètres à une méthode. Cette solution permet de présenter les différentes options à l'utilisateur du plugin, mieux que la simple liste de paramètres précédente. Une mappe donne un libellé à chaque paramètre et retire toute exigence sur leur ordre. Par ailleurs, chaque fois que c'est possible, il est préférable d'imiter l'API de jQuery dans les plugins car cela améliore la cohérence du code et, par conséquent, la facilité d'utilisation.

```
jQuery.fn.shadow = function(opts) {
  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < opts.slices; i++) {
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left + i,
          top: $originalElement.offset().top + i,
          margin: 0,
          zIndex: opts.zIndex,
          opacity: opts.opacity
        })
        .appendTo('body');
    }
  });
};
```

La seule modification de notre interface réside dans la manière de référencer chaque paramètre. Au lieu d'utiliser un nom de variable séparé, nous accédons à chaque valeur en tant que propriété de l'argument `opts` de la fonction.

Pour invoquer la méthode, nous devons passer une mappe de valeurs à la place de trois nombres individuels (voir Figure 11.15) :

```
$(document).ready(function() {
  $('h1').shadow({
    slices: 5,
    opacity: 0.25,
    zIndex: -1
  });
});
```

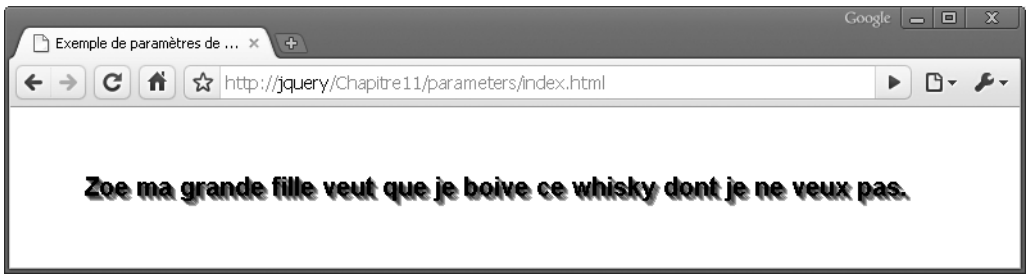


Figure 11.15

Paramétrage de l'effet au travers d'une mappe.

Par simple lecture de l'appel de la méthode, il est facile de comprendre le rôle de chaque paramètre.

Valeurs par défaut des paramètres

Plus le nombre de paramètres d'une méthode augmente, moins il est probable que nous voulions toujours préciser chacun d'eux. La définition de *valeurs par défaut* permet de faciliter l'utilisation de l'interface d'un plugin. Heureusement, grâce à la mappe utilisée pour indiquer les paramètres, cette technique est simple à mettre en œuvre. Toute propriété absente de la mappe est remplacée par sa valeur par défaut :

```
jQuery.fn.shadow = function(options) {
  var defaults = {
    slices: 5,
    opacity: 0.1,
    zIndex: -1
  };
  var opts = jQuery.extend(defaults, options);

  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < opts.slices; i++) {
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left + i,
          top: $originalElement.offset().top + i,
          margin: 0,
```

```
            zIndex: opts.zIndex,
            opacity: opts.opacity
          })
          .appendTo('body');
      }
    });
};
```

Dans la méthode, nous définissons une nouvelle mappe nommée `defaults`. La fonction utilitaire `$.extend()` prend la mappe `options` passée en argument et utilise ses valeurs pour remplacer celles par défaut, sans toucher aux éléments omis.

Nous invoquons toujours la méthode avec une mappe, mais nous pouvons à présent préciser uniquement les paramètres qui doivent avoir une valeur différente de celle par défaut (voir Figure 11.16) :

```
$(document).ready(function() {
  $('h1').shadow({i
    opacity: 0.05
  });
});
```

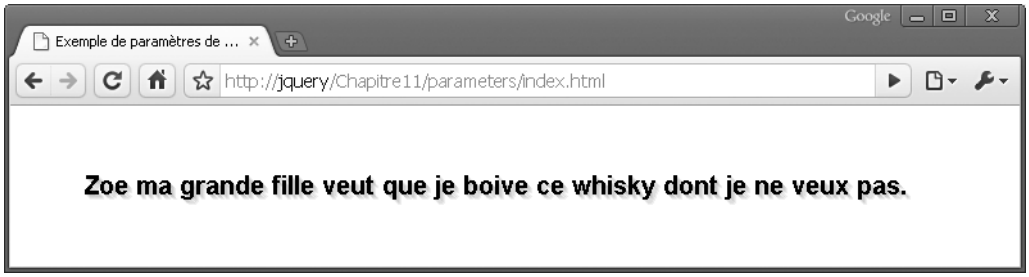


Figure 11.16

La mappe précise uniquement les paramètres dont la valeur change.

Les paramètres non précisés gardent leur valeur par défaut. La méthode `$.extend()` accepte même les valeurs `null`. Ainsi, si les paramètres par défaut conviennent parfaitement, nous pouvons invoquer notre méthode très simplement sans générer d'erreur :

```
$(document).ready(function() {
  $('h1').shadow();
});
```

Fonctions de rappel

Bien entendu, certains paramètres de méthodes peuvent être plus complexes qu'une simple valeur numérique. Dans l'API de jQuery, nous avons très souvent rencontré des paramètres de type *fonction de rappel*. Ces fonctions apportent une très grande flexibilité au plugin sans imposer un travail important au moment de sa création.

Pour utiliser une fonction de rappel dans notre méthode, nous devons simplement accepter un objet fonction en paramètre et invoquer cette fonction à l'endroit adéquat dans la méthode. À titre d'exemple, nous étendons notre méthode d'ombrage du texte pour que l'utilisateur puisse définir l'emplacement de l'ombre relativement au texte :

```
jQuery.fn.shadow = function(options) {
  var defaults = {
    slices: 5,
    opacity: 0.1,
    zIndex: -1,
    sliceOffset: function(i) {
      return {x: i, y: i};
    }
  };
  var opts = jQuery.extend(defaults, options);

  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < opts.slices; i++) {
      var offset = opts.sliceOffset(i);
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left
            + offset.x,
          top: $originalElement.offset().top
            + offset.y,
          margin: 0,
          zIndex: opts.zIndex,
          opacity: opts.opacity
        })
        .appendTo('body');
    }
  });
};
```

Chaque composante de l'ombre a un décalage différent par rapport au texte d'origine. Précédemment, ce décalage était simplement égal à l'indice de la composante. À présent, nous le calculons à l'aide de la fonction `sliceOffset()`, qui est un paramètre que l'utilisateur peut redéfinir. Par exemple, nous pouvons fournir des valeurs négatives pour le décalage dans les deux dimensions :

```
$(document).ready(function() {
  $('h1').shadow({
    sliceOffset: function(i) {
      return {x: -i, y: -2*i};
    }
  });
});
```

Avec cette fonction de rappel, l'ombre se décale à présent vers le haut et la gauche, non plus vers le bas et la droite (voir Figure 11.17).

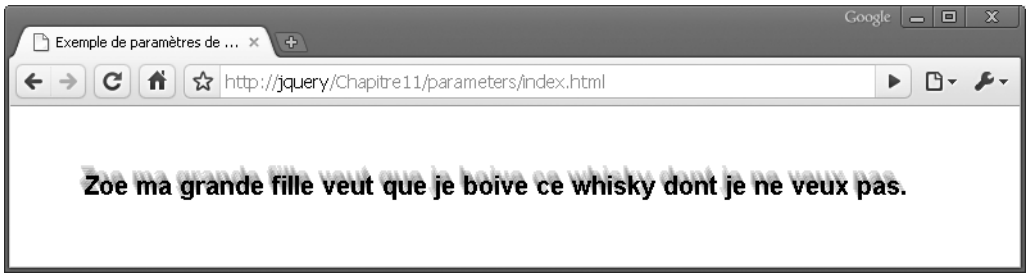


Figure 11.17

Une fonction de rappel permet de modifier la création de l'ombre.

La fonction de rappel autorise des modifications simples de la direction de l'ombre, ou un positionnement beaucoup plus sophistiqué si l'utilisateur du plugin écrit la fonction adéquate. Lorsque la fonction de rappel n'est pas précisée, le comportement par défaut est utilisé.

Valeurs par défaut personnalisables

Nous l'avons vu, nous pouvons améliorer l'utilisation de nos plugins en proposant des valeurs par défaut raisonnables pour les paramètres de notre méthode. Cependant, il arrive que la valeur par défaut raisonnable soit difficile à trouver. Lorsqu'un script invoque plusieurs fois notre plugin avec des valeurs de paramètres différentes de celles définies par défaut, la possibilité de changer ces valeurs par défaut permet de réduire significativement la quantité de code à écrire.

Pour que les valeurs par défaut puissent être personnalisées, nous devons les sortir de la définition de la méthode et les placer dans un endroit accessible au code externe :

```
jQuery.fn.shadow = function(options) {
  var opts = jQuery.extend({}, jQuery.fn.shadow.defaults, options);

  return this.each(function() {
    var $originalElement = jQuery(this);
    for (var i = 0; i < opts.slices; i++) {
      var offset = opts.sliceOffset(i);
      $originalElement
        .clone()
        .css({
          position: 'absolute',
          left: $originalElement.offset().left + offset.x,
          top: $originalElement.offset().top + offset.y,
          margin: 0,
          zIndex: opts.zIndex,
          opacity: opts.opacity
        })
        .appendTo('body');
    }
```

```
  });
};

jQuery.fn.shadow.defaults = {
  slices: 5,
  opacity: 0.1,
  zIndex: -1,
  sliceOffset: function(i) {
    return {x: i, y: i};
  }
};
```

Les valeurs par défaut se trouvent à présent dans l'*espace de noms* du plugin et il est possible de les référencer directement avec `$.fn.shadow.defaults`. Nous avons également modifié notre appel à `$.extend()` pour en tenir compte. Puisque nous réutilisons la même mappe de valeurs par défaut pour chaque appel à `.shadow()`, nous ne pouvons plus autoriser `$.extend()` à la modifier. À la place, nous passons une mappe vide (`{}`) comme premier argument de `$.extend()` et c'est ce nouvel objet qui est modifié.

Le code qui utilise notre plugin peut à présent modifier les valeurs par défaut, qui seront employées pour tous les appels ultérieurs à `.shadow()`. Des options peuvent toujours être fournies au moment de l'invocation de la méthode :

```
$(document).ready(function() {
  $.fn.shadow.defaults.slices = 10;
  $('h1').shadow({
    sliceOffset: function(i) {
      return {x: -i, y: i};
    }
  });
});
```

Ce script crée une ombre à dix composantes, puisqu'il s'agit de la nouvelle valeur par défaut, mais dirige également l'ombre vers la gauche et le bas du fait de la fonction de rappel `sliceOffset` passée en argument de la méthode (voir Figure 11.18).

Figure 11.18

Changement d'une valeur par défaut et passage d'une fonction de rappel en argument.

11.6 Ajouter une expression de sélection

Les parties internes de jQuery peuvent également être étendues. Au lieu d'ajouter de nouvelles méthodes, nous pouvons personnaliser celles qui existent déjà. Par exemple, il est fréquent d'étendre les *expressions de sélection* de jQuery pour proposer des options ésotériques.

La *pseudo-classe* est le type d'expression de sélection le plus facile à ajouter. Les pseudo-classes sont les expressions qui commencent par des deux-points, comme `:checked` ou `:nth-child()`. Pour illustrer la procédure de création d'une expression de sélection, nous allons construire une pseudo-classe nommée `:css()`. Ce nouveau sélecteur nous permettra de localiser des éléments en fonction des valeurs numériques de leurs attributs CSS.

Lorsqu'une expression de sélection est utilisée pour rechercher des éléments, jQuery prend ses instructions dans une mappe interne appelée `expr`. Elle contient du code JavaScript qui est exécuté sur un élément et, s'il retourne la valeur `true`, l'élément est ajouté au jeu de résultats. Nous pouvons ajouter de nouvelles expressions à cette mappe en utilisant la fonction `$.extend()` :

```
jQuery.extend(jQuery.expr[':'], {
  'css': function(element, index, matches, set) {
    var parts = /([\w-]+)\s*([<>=]+)\s*(\d+)/.exec(matches[3]);
    var value = parseFloat(jQuery(element).css(parts[1]));

    switch (parts[2]) {
      case '<':
        return value < parseInt(parts[3]);
      case '<=':
        return value <= parseInt(parts[3]);
      case '=':
      case '==':
        return value == parseInt(parts[3]);
      case '>=':
        return value >= parseInt(parts[3]);
      case '>':
        return value > parseInt(parts[3]);
    }
  }
});
```

Ce code indique à jQuery que `css` est une chaîne valide qui peut venir après des deux-points dans une expression de sélection et que, si c'est le cas, la fonction indiquée doit être invoquée pour déterminer si l'élément fait partie de la collection résultante.

Dans notre exemple, la fonction d'évaluation reçoit quatre paramètres :

- `element`. L'élément du DOM considéré. Il est requis pour la plupart des sélecteurs.
- `index`. L'indice de l'élément du DOM dans le jeu de résultats. Il est utile pour les sélecteurs du type `:eq()` et `:lt()`.

- `matches`. Un tableau contenant le résultat de l'expression régulière qui a servi à analyser le sélecteur. En général, `matches[3]` est le seul élément pertinent du tableau ; dans un sélecteur de la forme `:a(b)`, l'élément `matches[3]` contient `b`, c'est-à-dire le texte entre les parenthèses.
- `set`. L'intégralité des éléments du DOM qui correspondent jusqu'à présent. Ce paramètre est rarement nécessaire.

Les sélecteurs de pseudo-classes doivent utiliser les informations contenues dans ces quatre arguments pour déterminer si l'élément appartient ou non à la collection résultante. Dans notre cas, nous avons uniquement besoin de `element` et de `matches`.

Dans la fonction de sélection, nous utilisons une expression régulière pour obtenir les parties intéressantes du sélecteur. Nous souhaitons qu'un sélecteur de la forme `:css(width < 200)` retourne tous les éléments dont la largeur est inférieure à 200. Nous devons donc examiner le texte placé entre les parenthèses afin d'extraire le nom de la propriété (`width`), l'opérateur de comparaison (`<`) et la valeur à comparer (`200`). L'expression régulière `/([\w-]+)\s*([<>=]+)\ s*(\d+)/` effectue cette recherche et place les trois parties de la chaîne dans le tableau `parts`.

Ensuite, nous devons déterminer la valeur courante de la propriété. Nous nous servons de la méthode jQuery `.css()` pour obtenir la valeur de la propriété dont le nom est précisé dans le sélecteur. Puisque cette valeur est retournée comme une chaîne de caractères, nous appelons `parseFloat()` pour la convertir en une valeur numérique.

Enfin, nous procédons à la comparaison. Une instruction `switch` détermine le type de la comparaison en fonction du contenu du sélecteur, puis le résultat de la comparaison (`true` ou `false`) est retourné.

Nous disposons à présent d'une nouvelle expression de sélection que nous pouvons employer n'importe où dans notre code jQuery. Un document HTML simple nous permettra d'effectuer nos tests (voir Figure 11.19) :

```
<body>
  <div>Deserunt mollit anim id est laborum</div>
  <div>Ullamco</div>
  <div>Ut enim ad minim veniam laboris</div>
  <div>Quis nostrud exercitation consequat nisi</div>
  <div>Ut aliquip</div>
  <div>Commodo</div>
  <div>Lorem ipsum dolor sit amet ex ea</div>
</body>
```

Grâce à notre nouveau sélecteur, il est très facile de mettre en exergue les éléments les moins larges de cette liste (voir Figure 11.20) :

```
$(document).ready(function() {
  $('div:css(width < 100)').addClass('highlight');
});
```

Figure 11.19

Le document de test du sélecteur.

Figure 11.20

Sélection des éléments les moins larges.

11.7 Distribuer un plugin

Lorsque le plugin est terminé, nous souhaitons le publier afin que les autres développeurs puissent profiter du code, voire l'améliorer. Pour cela, nous pouvons le déposer dans le catalogue officiel des plugins de jQuery à **http://plugins.jquery.com/**. Sur ce site, nous pouvons ouvrir une session ou nous inscrire si ce n'est déjà fait, puis suivre les instructions des options du plugin et envoyer une archive .zip de son code. Mais, avant cela, nous devons vérifier que le plugin est parfaitement terminé et prêt à être utilisé par tout le monde.

Nous devons respecter quelques règles d'écriture des plugins pour qu'ils puissent fonctionner correctement avec d'autres codes. Nous en avons déjà vu certaines, mais elles sont regroupées ici pour plus de commodité.

Conventions de nommage

Tout fichier de plugin doit être nommé `jQuery.monPlugin.js`, où `monPlugin` correspond au nom du plugin. Dans le fichier, toutes les fonctions globales doivent être regroupées dans un objet nommé `jQuery.monPlugin`, excepté lorsqu'il n'y en a qu'une. Dans ce cas, elle peut être une fonction appelée simplement `jQuery.monPlugin()`.

Le nommage des méthodes est plus souple, mais les noms doivent être uniques autant que possible. Si une seule méthode est définie, elle doit s'appeler `jQuery.fn.monPlugin()`. Si plusieurs méthodes sont définies, chaque nom doit commencer par celui du plugin pour éviter toute confusion. Il est préférable d'éviter les noms de méthodes courts et ambigus, comme `.load()` ou `.get()`, qui peuvent être confondus avec des méthodes définies dans d'autres plugins.

Utiliser l'alias *$*

Les plugins jQuery ne doivent pas supposer que l'alias `$` soit disponible. À la place, il faut écrire à chaque fois le nom complet `jQuery`.

Pour les plugins assez longs, les développeurs trouvent souvent que l'absence du raccourci `$` rend le code plus difficile à lire. Pour y remédier, le raccourci peut être créé dans la portée du plugin en définissant et en exécutant une fonction. Voici la syntaxe qui permet à la fois de définir et d'exécuter une fonction :

```
(function($) {
  // Code à exécuter.
})(jQuery);
```

La fonction enveloppante prend un seul paramètre dans lequel nous passons l'objet jQuery global. Puisque l'argument se nomme `$`, nous pouvons utiliser l'alias `$` à l'intérieur de la fonction sans craindre les conflits.

Interfaces de méthode

Toutes les méthodes jQuery sont appelées dans le contexte d'un objet jQuery. Par conséquent, `this` désigne un objet qui peut faire référence à un ou à plusieurs éléments du DOM. Toutes les méthodes doivent avoir un comportement correct vis-à-vis des éléments sélectionnés. En général, les méthodes doivent appeler `this.each()` pour itérer sur tous les éléments sélectionnés, en les traitant chacun tour à tour.

Les méthodes doivent retourner l'objet jQuery pour conserver le chaînage. Si la collection des objets sélectionnés est modifiée, un nouvel objet doit être créé en invoquant `.pushStack()` et cet objet doit être retourné. Si la méthode retourne autre chose qu'un objet jQuery, la documentation doit l'indiquer clairement.

Lorsque des méthodes prennent plusieurs options, il est préférable d'utiliser une mappe en argument afin de libeller les options et de ne pas fixer un ordre. Des valeurs par défaut doivent être définies dans une mappe, qui pourra être redéfinie si nécessaire.

Les définitions de méthodes doivent se terminer par un point-virgule (;) pour que les compacteurs de code puissent effectuer une analyse correcte des fichiers. Par ailleurs, les plugins peuvent commencer par un caractère point-virgule afin que les autres scripts mal écrits ne provoquent pas de conflits après la compression.

Style de la documentation

Une documentation doit être ajoutée dans le fichier avant chaque définition de fonction ou de méthode, en utilisant le format ScriptDoc. Pour de plus amples informations sur ce format, consultez le site **http://www.scriptdoc.org/**.

11.8 En résumé

Dans ce dernier chapitre, nous avons vu que les fonctionnalités fournies en standard par jQuery ne limitent en rien les possibilités de la bibliothèque. Les plugins disponibles permettent d'étendre la liste des fonctionnalités et nous pouvons créer nos propres plug-ins pour aller encore plus loin.

Les plugins que nous avons créés proposent différentes fonctionnalités, notamment l'ajout de fonctions globales qui se fondent sur la bibliothèque jQuery, de nouvelles méthodes de l'objet jQuery pour manipuler les éléments du DOM, des méthodes extensibles qu'il est facile de personnaliser, et des expressions de sélection améliorées pour rechercher de différentes manières les éléments du DOM.

Avec ces outils à disposition, nous pouvons modeler jQuery, ainsi que notre propre code JavaScript, comme bon nous semble.

Annexe A

Ressources en ligne

Les ressources en ligne suivantes constituent un point de départ pour en apprendre plus sur jQuery, JavaScript et le développement web de manière générale. Les sources d'information sont beaucoup trop nombreuses sur le Web pour que cette annexe puisse en faire une liste exhaustive. Par ailleurs, vous trouverez également des informations précieuses dans d'autres publications imprimées, mais nous ne les citons pas ici.

A.1 Documentation sur jQuery

Les ressources suivantes constituent des références et fournissent des détails sur la bibliothèque jQuery.

Wiki jQuery

- **http://docs.jquery.com/**

Ce site est un wiki. Autrement dit, son contenu peut être modifié par des internautes. Vous y trouverez des informations sur l'intégralité de l'API de jQuery, des didacticiels, des guides de démarrage et plus encore.

API de jQuery

- **http://remysharp.com/jquery-api/**

Si la documentation officielle se trouve sur le site **jquery.com**, celui-ci propose également des informations sur l'API.

Navigateur dans l'API de jQuery

- **http://jquery.bassistance.de/api-browser-1.2/**

Jörn Zaefferer a réuni l'API de jQuery dans un système pratique avec navigation arborescente. Ce site propose également une fonctionnalité de recherche, ainsi qu'un tri alphabétique et par catégorie.

Visual jQuery

- **http://www.visualjquery.com/**

Ce système de navigation dans l'API, conçu par Yehuda Katz et mis à jour par Remy Sharp, est à la fois élégant et pratique. Il propose également une présentation rapide des méthodes de plusieurs plugins jQuery.

Visionneuse de l'API de jQuery avec Adobe AIR

- **http://remysharp.com/downloads/jquery-api-browser.air.zip**

Remy Sharp a conditionné l'API de jQuery dans une application Adobe AIR de manière à pouvoir la consulter hors ligne.

A.2 Références JavaScript

Les sites suivants proposent des références et des guides sur le langage JavaScript en général, sans se focaliser sur jQuery.

Centre de développement Mozilla

- **https://developer.mozilla.org/fr/JavaScript**

Ce site comprend une référence complète sur JavaScript, un guide de programmation avec JavaScript, des liens vers des outils utiles et bien d'autres informations.

Dev.opera

- **http://dev.opera.com/articles/**

Bien qu'orienté vers son propre navigateur, le site d'Opera pour les développeurs web inclut de nombreux articles intéressants sur JavaScript.

Référence JScript de MSDN

- **http://msdn.microsoft.com/fr-fr/library/x85xxsf4.aspx**

La référence JScript sur MSDN décrit l'intégralité des fonctions, des objets et autres éléments. Elle est particulièrement utile pour comprendre la mise en œuvre Microsoft du standard ECMAScript dans Internet Explorer.

Quirksmode

- **http://www.quirksmode.org/**

Le site Quirksmode de Peter-Paul Koch est une formidable ressource pour comprendre les différences d'implémentation des fonctions JavaScript et des propriétés CSS dans les navigateurs.

Boîte à outils JavaScript

- **http://www.javascripttoolbox.com/**

La boîte à outils JavaScript de Matt Kruse propose un large assortiment de bibliothèques JavaScript rustiques, des conseils avisés sur les meilleures pratiques JavaScript et une collection de ressources JavaScript décrites ailleurs sur le Web.

A.3 Compacteurs de code JavaScript

Au cours de la phase finale de construction d'un site, il est souvent avisé de compacter le code JavaScript. Cette procédure permet de réduire le temps de téléchargement des pages.

YUI Compressor

- **http://developer.yahoo.com/yui/compressor/**

Ce compacteur JavaScript proposé par Yahoo! UI Library permet de réduire la taille du code source jQuery. L'outil Java en ligne de commande peut être téléchargé gratuitement. Le code résultant est très efficace, tant en termes de taille de fichier que de performances, et peut même faire l'objet d'une compression Gzip supplémentaire.

JSMin

- **http://www.crockford.com/javascript/jsmin.html**

Créé par Douglas Crockford, JSMin est un filtre qui retire les commentaires et les espaces inutiles dans les fichiers JavaScript. En général, il permet de diviser par deux la taille des fichiers, conduisant à des téléchargements plus rapides, en particulier s'il est associé à une compression du fichier sur le serveur.

Embellisseur de code

- **http://www.prettyprinter.de/**

Cet outil intervient sur le code JavaScript qui a été compressé. Il remet en place les passages à la ligne et l'indentation lorsque c'est possible. Ses différentes options permettent d'ajuster le résultat obtenu.

A.4 Référence (X)HTML

La bibliothèque jQuery est plus efficace lorsqu'elle est associée à des documents HTML et XHTML sémantiques correctement balisés. La ressource suivante constitue une aide sur ces langages de balisage.

Page d'accueil des langages de balisage du W3C

- **http://www.w3.org/MarkUp/**

Le W3C (*World Wide Web Consortium*) est en charge de la définition du standard pour (X)HTML. La page d'accueil du groupe de travail sur HTML est un bon point de départ pour consulter les spécifications et les guides.

A.5 Références CSS

Les effets et les animations que nous avons rencontrés se fondent sur la puissance des feuilles de style en cascade (CSS, *Cascading Style Sheets*). Pour ajouter à vos sites des décorations visuelles, vous pouvez consulter les ressources CSS suivantes.

Page d'accueil des feuilles de style en cascade du W3C

- **http://www.w3.org/Style/CSS/**

La page d'accueil CSS du W3C fournit des liens vers des didacticiels, des spécifications, des suites de tests et d'autres ressources.

Fiche CSS de Mezzoblue

- **http://mezzoblue.com/css/cribsheet/**

Dave Shea propose cette fiche CSS pour faciliter le processus de conception et fournir une référence rapide à consulter lorsque vous faites face à un problème.

Position is everything

- **http://www.positioniseverything.net/**

Ce site est un catalogue des bogues des navigateurs vis-à-vis de CSS. Il explique également comment les contourner.

A.6 Blogs

Dans toute technologie dynamique, de nouvelles techniques et de nouvelles fonctionnalités sont constamment développées et proposées. Pour rester à jour, il suffit de consulter de temps en temps les ressources suivantes, qui proposent une actualité du développement web.

Le blog de jQuery

- **http://jquery.com/blog/**

John Resig et d'autres contributeurs au blog officiel de jQuery publient des annonces concernant les nouvelles versions et d'autres initiatives en cours dans le projet. Ils proposent parfois des didacticiels et d'autres informations éditoriales.

Learning jQuery

- **http://www.learningjquery.com/**

Karl Swedberg est responsable de ce blog qui s'intéresse aux didacticiels, aux techniques et aux annonces concernant jQuery. Parmi les auteurs invités se trouvent Mike Alsup et Brandon Aaron, qui font partie de l'équipe du projet jQuery.

Ajaxian

- **http://ajaxian.com/**

Ce blog fréquemment mis à jour a été initié par Dion Almaer et Ben Galbraith. Il propose de nombreuses nouvelles et fonctionnalités pour JavaScript, ainsi que des didacticiels occasionnels.

John Resig

- **http://ejohn.org/**

Le créateur de jQuery, John Resig, discute de sujets JavaScript avancés sur son blog personnel.

JavaScript ant

- **http://javascriptant.com/**

Ce site propose un catalogue d'articles concernant JavaScript et ses utilisations dans les navigateurs web modernes. Vous y trouverez également une liste organisée des ressources JavaScript disponibles sur le Web.

Robert's talk

- **http://www.robertnyman.com/**

Robert Nyman rédige des billets sur le développement pour Internet, notamment sur les scripts côté client.

Standards web avec imagination

- **http://www.dustindiaz.com/**

Le blog de Dustin Diaz propose des articles sur la conception et le développement web, en mettant l'accent sur JavaScript.

Snook

- **http://snook.ca/**

Le blog de Jonathan Snook s'intéresse au développement web et à la programmation en général.

Ressources JavaScript de Matt Snider

- **http://mattsnider.com/**

Le blog de Matt Snider est dédié à la compréhension de JavaScript et de ses nombreux frameworks répandus.

I can't

- **http://icant.co.uk/**
- **http://www.wait-till-i.com/**
- **http://www.onlinetools.org/**

Ces trois sites de Christian Heilmann proposent des billets de blog, des exemples de code et de longs articles concernant JavaScript et le développement web.

Scripts pour le DOM

- **http://domscripting.com/blog/**

Le blog de Jeremy Keith commence là où le livre *DOM scripting* s'arrête. Il constitue une ressource inestimable pour du JavaScript non intrusif.

As days pass by

- **http://www.kryogenix.org/code/browser/**

Stuart Langridge mène des expériences sur l'utilisation avancée du DOM dans le navigateur.

A list apart

- **http://www.alistapart.com/**

Ce site explore la conception, le développement et la signification du contenu web, avec un intérêt particulier sur les standards web et les meilleures pratiques.

A.7 Frameworks de développement web utilisant jQuery

Lorsque les développeurs de projets open-source ont connu l'existence de jQuery, ils ont été nombreux à inclure cette bibliothèque JavaScript dans leur propre système. Voici une liste non exhaustive de ces frameworks :

- Digitalus Site Manager (**http://code.google.com/p/digitalus-site-manager/**) ;
- Drupal (**http://drupal.org/**) ;
- DutchPIPE (**http://dutchpipe.org/**) ;
- Hpricot (**http://code.whytheluckystiff.net/hpricot/**) ;
- JobberBase (**http://www.jobberbase.com/**) ;
- Laconica (**http://laconi.ca/**) ;
- Piwik (**http://piwik.org/**) ;
- Pommo (**http://pommo.org/**) ;
- simfony (**http://www.symfony-project.org/**) ;
- SPIP (**http://www.spip.net/**) ;
- Textpattern (**http://www.textpattern.com/**) ;
- Trac (**http://trac.edgewall.org/**) ;
- WordPress (**http://wordpress.org/)** ;
- Z-Blog (**http://www.rainbowsoft.org/zblog**).

Vous trouverez une liste plus complète en allant sur la page Sites Using jQuery à l'adresse **http://docs.jquery.com/Sites_Using_jQuery**.

Annexe B

Outils de développement

Si la documentation aide à la résolution des problèmes rencontrés dans les applications JavaScript, rien ne remplace de bons outils de développement. Heureusement, il existe de nombreux paquetages logiciels pour inspecter et déboguer du code JavaScript, dont la plupart sont gratuits.

B.1 Outils pour Firefox

Mozilla Firefox est un navigateur choisi par de nombreux développeurs web et c'est pourquoi il dispose des outils de développement parmi les plus complets et les plus reconnus.

Firebug

- **http://www.getfirebug.com/**

L'extension Firebug pour Firefox est indispensable à tout développement jQuery. Voici certaines de ses fonctionnalités :

- un excellent inspecteur du DOM pour rechercher les noms et les sélecteurs des éléments du document ;
- des outils de manipulation des styles CSS pour comprendre pourquoi une page a une certaine apparence et la modifier ;
- une console JavaScript interactive ;
- un débogueur JavaScript, avec suivi des variables et trace de l'exécution du code.

Barre d'outils du développeur web

- **http://chrispederick.com/work/web-developer/**

Cette barre d'outils non seulement entre en concurrence avec Firebug dans le domaine de l'inspection du DOM, mais propose également des outils pour les tâches courantes, comme la manipulation des cookies, l'inspection des formulaires et le redimensionnement de la page. Vous pouvez vous servir de cette barre d'outils pour désactiver JavaScript de manière à vérifier que le site se dégrade de manière élégante lorsque le navigateur de l'utilisateur offre moins de possibilités.

Venkman

- **http://www.mozilla.org/projects/venkman/**

Venkman est le débogueur JavaScript officiel du projet Mozilla. Il fournit un environnement de dépannage issu de GDB, qui sert au débogage des programmes écrits dans d'autres langages.

Testeur d'expressions régulières

- **http://sebastianzartner.ath.cx/new/downloads/RExT/**

Il n'est pas toujours facile de construire les bonnes expressions régulières pour la correspondance des chaînes de caractères en JavaScript. Cette extension de Firefox permet de tester facilement des expressions régulières en utilisant une interface de saisie du texte de recherche.

B.2 Outils pour Internet Explorer

Les sites n'ont pas toujours le même comportement dans Internet Explorer que dans les autres navigateurs web. Il est donc important de disposer d'outils de débogage pour cette plate-forme.

Barre d'outils du développeur pour Microsoft Internet Explorer

- **http://www.microsoft.com/downloads/details.aspx?FamilyID=e59c3964-672d-4511-bb3e-2d5e1db91038**

La barre d'outils du développeur propose essentiellement une vue de l'arborescence DOM de la page web. Il est possible de localiser visuellement des éléments et d'apporter des modifications à la volée avec de nouvelles règles CSS. Elle apporte également d'autres aides au développement, comme une règle permettant de mesurer les éléments sur la page.

Visual Web Developer de Microsoft

- **http://msdn.microsoft.com/fr-fr/express/aa700797.aspx**

Cette version de Visual Studio peut servir à inspecter et à déboguer du code JavaScript. Pour exécuter interactivement le débogueur dans la version gratuite (Visual Web Developer Express), suivez la procédure détaillée sur le site **http://www.berniecode.com/blog/2007/03/08/how-to-debug-javascript-with-visual-web-developer-express/**.

DebugBar

- **http://www.debugbar.com/**

DebugBar propose un inspecteur du DOM, ainsi qu'une console JavaScript pour le débogage.

Drip

- **http://Sourceforge.net/projects/ieleak/**

Les fuites de mémoire dans le code JavaScript peuvent conduire à des problèmes de performances et de stabilité avec Internet Explorer. Drip facilite la détection et l'isolation de ces fuites.

Pour de plus amples informations concernant les causes des fuites de mémoire dans Internet Explorer, consultez l'Annexe C.

B.3 Outils pour Safari

Safari est le dernier-né en tant que plate-forme de développement, mais il existe néanmoins des outils pour examiner le code lorsque son comportement est différent dans ce navigateur.

Menu Développement

Depuis Safari 3.1, une option de l'onglet AVANCÉES des préférences donne accès à un menu spécial nommé DÉVELOPPEMENT. Lorsque ce menu est activé, un inspecteur web et une console JavaScript sont disponibles.

Inspecteur web

- **http://trac.webkit.org/wiki/Web%20Inspector**

Safari 3 offre la possibilité d'inspecter les éléments de la page et de collecter des informations, notamment concernant les règles CSS appliquées.

Les versions actuelles de WebKit ont beaucoup amélioré l'inspecteur web, en lui donnant plusieurs des excellentes fonctionnalités de Firebug, comme un débogueur JavaScript intégré nommé Drosera (**http://trac.webkit.org/wiki/Drosera**).

B.4 Outils pour Opera

Bien que ses parts de marché sur le segment des navigateurs soient faibles, Opera est un acteur important dans les systèmes embarqués et les périphériques mobiles. Ses possibilités doivent être examinées avec attention au cours d'un développement web.

Dragonfly

Encore au début de son développement, Dragonfly est un environnement de débogage prometteur pour les navigateurs Opera sur les ordinateurs ou les périphériques mobiles. Ses fonctionnalités sont comparables à celles de Firebug, y compris le débogage JavaScript et l'inspection et la modification du DOM et des styles CSS.

B.5 Autres outils

Si les outils précédents étaient spécifiques à un navigateur, les utilitaires suivants ont une gamme d'utilisation plus large.

Firebug Lite

- **http://www.getfirebug.com/lite.html**

L'extension Firebug est limitée au navigateur web Firefox, mais certaines de ses fonctionnalités peuvent être reproduites en incluant le script Firebug Lite dans la page web. Ce paquetage simule la console Firebug, autorisant même les appels à `console.log()` dans tous les navigateurs, sans générer d'erreurs JavaScript.

NitobiBug

- **http://www.nitobibug.com/**

À l'instar de Firebug Lite, NitobiBug est un outil multinavigateur qui offre certaines des fonctionnalités de l'extension Firebug en plus robuste et abouti. Sa force réside dans son inspecteur du DOM et des objets, même s'il propose également une console. Pour invoquer la console et l'inspecteur, il suffit d'inclure une référence au fichier JavaScript de Nitobi et d'appeler `nitobi.Debug.log()`.

Extension TextMate pour jQuery

- **http://github.com/kswedberg/jquery-tmbundle/**

Cette extension de TextMate, un éditeur de texte très répandu sur Mac OS X, apporte la mise en exergue syntaxique pour les méthodes et les sélecteurs jQuery, la complétion du code pour les méthodes et un accès à une référence rapide de l'API depuis le code. Le paquetage est également compatible avec l'éditeur de texte E pour Windows.

Charles

- **http://www.xk72.com/charles/**

Lors du développement d'applications fortement orientées AJAX, il est utile de voir précisément quelles données sont échangées entre le navigateur et le serveur. Le proxy de débogage web Charles affiche tout le trafic HTTP entre deux extrémités, y compris les requêtes web normales, le trafic HTTPS, les échanges Flash remoting et les réponses AJAX.

Fiddler

- **http://www.fiddlertool.com/fiddler/**

Fiddler est un autre proxy de débogage HTTP, dont les caractéristiques sont comparables à celles de Charles. Si l'on en croit son site, Fiddler "comporte un sous-système puissant d'écriture de script à base d'événements et peut être étendu en utilisant n'importe quel langage .NET".

Aptana

- **http://www.aptana.com/**

Cet IDE de développement web basé sur Java est libre et multiplate-forme. Disposant de fonctionnalités d'édition du code standard et élaborées, il inclut la documentation complète de l'API de jQuery et dispose de son propre débogueur JavaScript fondé sur Firebug.

Annexe C

Fermetures en JavaScript

Tout au long de cet ouvrage, nous avons rencontré de nombreuses méthodes jQuery qui prennent des *fonctions* en argument. Nos exemples ont donc créé, invoqué et passé des fonctions. Si une compréhension élémentaire des mécanismes internes de JavaScript suffit à mettre en œuvre cette possibilité, les effets secondaires de nos actions peuvent parfois sembler étranges si nous ne maîtrisons pas les caractéristiques du langage. Dans cette annexe, nous allons étudier un aspect des fonctions des plus ésotérique, quoique répandu, appelé *fermetures* (*closures*).

Notre présentation s'articulera autour de nombreux petits exemples de code qui nous serviront à afficher des messages. Au lieu d'utiliser des mécanismes de journalisation propres au navigateur, comme `console.log()` pour Firefox, ou de créer une suite de boîtes de dialogue `alert()`, nous nous servirons d'une courte méthode de plugin :

```
jQuery.fn.print = function(message) {
  return this.each(function() {
    $('<div class="result" />')
      .text(String(message))
      .appendTo($(this).find('.results'));
  });
};
```

Après avoir défini cette méthode, nous invoquons `$('#example').print('salut')` pour ajouter le texte "salut" à `<div id="example">`.

C.1 Fonctions internes

JavaScript a la chance de faire partie des langages de programmation qui prennent en charge les *fonctions internes*. De nombreux langages de programmation classiques, comme C, placent toutes les fonctions dans une même portée au niveau supérieur. En revanche, les langages qui reconnaissent les fonctions internes permettent de placer de petites fonctions utilitaires là où elles sont requises, évitant ainsi la *pollution de l'espace de noms*.

Une fonction interne n'est rien d'autre qu'une fonction définie à l'intérieur d'une autre fonction, par exemple :

```
function outerFn() {
  function innerFn() {
  }
}
```

Dans cet exemple, la fonction interne `innerFn()` est définie à l'intérieur de la portée de `outerFn()`. Autrement dit, l'appel à `innerFn()` n'est valide que depuis `outerFn()`. Le code suivant génère une erreur JavaScript :

```
function outerFn() {
  $('#example-2').print('Fonction externe');
  function innerFn() {
    $('#example-1').print('Fonction interne');
  }
}
$('#example-1').print('innerFn():');
innerFn();
```

Pour éviter l'erreur, nous pouvons invoquer `innerFn()` à partir de `outerFn()` :

```
function outerFn() {
  $('#example-2').print('Fonction externe');
  function innerFn() {
    $('#example-2').print('Fonction interne');
  }
  innerFn();
}
$('#example-2').print('outerFn():');
outerFn();
```

Voici la sortie obtenue :

```
outerFn():
Fonction externe
Fonction interne
```

Cette technique se révèle particulièrement utile pour les petites fonctions effectuant une tâche unique. Par exemple, les algorithmes *récursifs* sans API récursive s'expriment généralement mieux avec une fonction interne.

La grande évasion

Le mystère s'épaissit lorsque les *références de fonctions* sont utilisées. Certains langages, comme Pascal, ne prennent en charge les fonctions internes que dans le but de *masquer du code*. Ces fonctions sont enterrées à jamais avec leurs fonctions parents. En revanche, JavaScript nous permet de passer des fonctions comme s'il s'agissait de n'importe quel autre type de données. Autrement dit, les fonctions internes peuvent échapper à leurs geôliers.

La voie de l'évasion peut prendre de nombreuses directions. Par exemple, supposons que la fonction soit affectée à une *variable globale* :

```
var globalVar;
function outerFn() {
  $('#example-3').print('Fonction externe');
  function innerFn() {
    $('#example-3').print('Fonction interne');
  }
  globalVar = innerFn();
}
$('#example-3').print('outerFn():');
outerFn();
$('#example-3').print('globalVar():');
globalVar();
```

L'appel à `outerFn()` après la définition de la fonction modifie la variable globale `globalVar`, qui fait alors référence à `innerFn()`. Autrement dit, l'appel suivant à `globalVar()` fonctionne exactement comme un appel interne à `innerFn()`, et les instructions d'affichage sont exécutées :

```
outerFn():
Fonction externe
globalVar():
Fonction interne
```

Notez qu'un appel à `innerFn()` depuis l'extérieur de `outerFn()` génère toujours une erreur ! Même si la fonction s'est évadée par l'intermédiaire de la référence enregistrée dans la variable globale, le nom de la fonction reste enfermé dans la portée de `outerFn()`.

Une référence de fonction peut également s'échapper de sa fonction parent en passant par la *valeur de retour* :

```
function outerFn() {
  $('#example-4').print('Fonction externe');
  function innerFn() {
    $('#example-4').print('Fonction interne');
  }
  return innerFn();
}
$('#example-4').print('var fnRef = outerFn():');
var fnRef = outerFn();
$('#example-4').print(fnRef():');
fnRef();
```

Dans ce cas, aucune variable n'est modifiée par `outerFn()`, qui retourne à la place une référence à `innerFn()`. L'invocation de `outerFn()` retourne cette référence, qui est enregistrée puis invoquée, pour produire à nouveau le même message :

```
var fnRef = outerFn():
Fonction externe
fnRef():
Fonction interne
```

Puisque les fonctions internes peuvent être invoquées au travers d'une référence, même après que la fonction n'est plus dans la portée, JavaScript doit garder disponibles les fonctions référencées aussi longtemps qu'il est possible de les invoquer. Chaque varia-

ble qui fait référence à la fonction est gérée par le *système d'exécution* de JavaScript. Ensuite, lorsque la dernière a disparu, le ramasse-miettes de JavaScript entre en scène et libère la mémoire auparavant occupée.

Portée de variable

Les fonctions internes peuvent évidemment posséder leurs propres variables, dont la portée est alors limitée à la fonction elle-même :

```
function outerFn() {
  function innerFn() {
    var innerVar = 0;
    innerVar++;
    $('#example-5').print('innerVar = ' + innerVar);
  }
  return innerFn();
}
var fnRef = outerFn();
fnRef();
fnRef();
var fnRef2 = outerFn();
fnRef2();
fnRef2();
```

Chaque fois que la fonction est appelée, que ce soit au travers d'une référence ou par un autre moyen, une nouvelle variable `innerVar` est créée, incrémentée, puis affichée :

```
innerVar = 1
innerVar = 1
innerVar = 1
innerVar = 1
```

À l'instar de n'importe quelle autre fonction, les fonctions internes peuvent faire référence à des variables globales :

```
var globalVar = 0;
function outerFn() {
  function innerFn() {
    globalVar++;
    $('#example-6').print('globalVar = ' + globalVar);
  }
  return innerFn();
}
var fnRef = outerFn();
fnRef();
fnRef();
var fnRef2 = outerFn();
fnRef2();
fnRef2();
```

Dans ce cas, la fonction incrémente la variable lors de chaque appel :

```
globalVar = 1
globalVar = 2
globalVar = 3
globalVar = 4
```

Qu'en est-il des variables locales de la fonction parent ? Puisque la fonction interne hérite de la portée de son parent, ces variables peuvent également être utilisées :

```
function outerFn() {
  var outerVar = 0;
  function innerFn() {
    outerVar++;
    $('#example-7').print('outerVar = ' + outerVar);
  }
  return innerFn();
}
var fnRef = outerFn();
fnRef();
fnRef();
var fnRef2 = outerFn();
fnRef2();
fnRef2();
```

Les appels à notre fonction ont alors un comportement plus intéressant :

```
outerVar = 1
outerVar = 2
outerVar = 1
outerVar = 2
```

Nous obtenons un mélange des deux effets précédents. Les appels à `innerFn()` par l'intermédiaire de chaque référence incrémentent indépendamment `outerVar`. Notez que le deuxième appel à `outerFn()` ne réinitialise pas la valeur de `outerVar`, mais crée à la place une nouvelle *instance* de `outerVar` liée à la portée du deuxième appel de fonction. En conséquence, après les premiers appels, une autre invocation de `fnRef()` afficherait la valeur `3` et un autre appel à `fnRef2()` afficherait également `3`. Les deux compteurs sont totalement séparés.

Lorsqu'une référence dans une fonction interne trouve un moyen de sortir de la portée dans laquelle la fonction a été définie, cela crée une *fermeture* sur cette fonction. Les variables qui ne sont ni des paramètres ni des variables locales à la fonction interne sont appelées *variables libres*, et l'environnement d'un appel à la fonction externe les *ferme*. Le fait que la fonction fasse référence à une variable locale de la fonction externe accorde un sursis à la variable. La mémoire n'est pas libérée lorsque la fonction se termine, puisqu'elle peut encore être nécessaire à la fermeture.

C.2 Interactions entre fermetures

Lorsqu'il existe plusieurs fonctions internes, des fermetures peuvent avoir des effets difficiles à anticiper. Supposons que nous associions notre fonction d'incrémentation à une autre fonction, qui ajoute deux :

```
function outerFn() {
  var outerVar = 0;
  function innerFn1() {
    outerVar++;
    $('#example-8').print('(1) outerVar = ' + outerVar);
  }
  function innerFn2() {
    outerVar += 2;
    $('#example-8').print('(2) outerVar = ' + outerVar);
  }
  return {'fn1': innerFn1, 'fn2': innerFn2};
}
var fnRef = outerFn();
fnRef.fn1();
fnRef.fn2();
fnRef.fn1();
var fnRef2 = outerFn();
fnRef2.fn1();
fnRef2.fn2();
fnRef2.fn1();
```

Nous retournons des références aux deux fonctions en utilisant une *mappe* (une autre manière pour une référence à une fonction interne de s'évader de sa fonction parent). Les deux fonctions sont invoquées par l'intermédiaire des références :

```
(1) outerVar = 1
(2) outerVar = 3
(1) outerVar = 4
(1) outerVar = 1
(2) outerVar = 3
(1) outerVar = 4
```

Les deux fonctions internes font référence à la même variable locale et partagent donc le même *environnement de fermeture*. Lorsque `innerFn1()` incrémente de `1` la variable `outerVar`, elle fixe sa nouvelle valeur initiale pour l'appel à `innerFn2()`, et *vice versa*. Cependant, nous voyons une fois encore que tout appel ultérieur à `outerFn()` crée de nouvelles instances de ces fermetures, avec un nouvel environnement de fermeture correspondant. Les amateurs de *programmation orientée objet* remarqueront que nous avons en réalité créé un nouvel *objet*, avec les variables libres jouant le rôle des *variables d'instance* et les fermetures, celui des *méthodes d'instance*. Les variables sont également *privées*, car il est impossible de les référencer directement en dehors de la portée englobante. Nous avons donc de réelles données privées orientées objet.

C.3 Fermetures dans jQuery

Les méthodes jQuery que nous avons rencontrées prennent souvent au moins une fonction en argument. Pour des raisons pratiques, nous utilisons des *fonctions anonymes* afin que le comportement de la fonction soit défini là où il est requis. Autrement dit, ces fonctions se trouvent rarement dans l'espace de noms de premier niveau ; il s'agit généralement de fonctions internes, qui peuvent donc facilement créer des fermetures.

Arguments de *$(document).ready()*

Pratiquement tout le code que nous écrivons avec jQuery finit par se retrouver dans une fonction passée en argument à `$(document).ready()`. Cela nous permet d'être certains que le DOM a été chargé avant que le code ne soit exécuté, ce qui est généralement indispensable pour du code jQuery intéressant. Lorsqu'une fonction est créée et passée à `.ready()`, une référence à cette fonction est enregistrée dans l'objet jQuery global. Cette référence est ensuite invoquée ultérieurement lorsque le DOM est prêt.

Puisque la construction `$(document).ready()` se trouve généralement au sommet de la structure du code, cette fonction ne fait pas réellement partie d'une fermeture. En revanche, puisque notre code est habituellement écrit dans cette fonction, tout le reste constitue une fonction interne :

```
$(document).ready(function() {
  var readyVar = 0;
  function innerFn() {
    readyVar++;
    $('#example-9').print('readyVar = ' + readyVar);
  }
  innerFn();
  innerFn();
});
```

Cet exemple ressemble aux nombreux exemples précédents, excepté que, dans ce cas, la fonction externe est la fonction de rappel passée à `$(document).ready()`. Puisque `innerFn()` est définie à l'intérieur de cette fonction et qu'elle fait référence à `readyVar`, qui se trouve dans la portée de la fonction de rappel, `innerFn()` et son environnement créent une fermeture. Nous pouvons le constater en remarquant que la valeur de la variable `readyVar` persiste entre les appels à la fonction :

```
readyVar = 1
readyVar = 2
```

Le fait que la plupart du code jQuery se trouve dans le corps d'une fonction se révèle utile. En effet, cela permet d'éviter certains *conflits sur l'espace de noms*. Par exemple, c'est grâce à cette caractéristique que nous pouvons utiliser `jQuery.noConflict()` pour rendre le raccourci `$` à d'autres bibliothèques, tout en étant capable de le définir localement pour l'utiliser dans `$(document).ready()`.

Gestionnaires d'événements

La construction `$(document).ready()` englobe généralement le reste de notre code, y compris l'affectation des *gestionnaires d'événements*. Puisque les gestionnaires sont des fonctions, il s'agit donc de fonctions internes. Puisque ces fonctions internes sont enregistrées et invoquées ultérieurement, elles peuvent créer des fermetures. Un simple gestionnaire de `click` illustrera ce propos :

```
$(document).ready(function() {
  var counter = 0;
  $('#example-10 a.add').click(function() {
    counter++;
    $('#example-10').print('counter = ' + counter);
    return false;
  });
});
```

Puisque la variable `counter` est déclarée dans le gestionnaire `.ready()`, elle n'est disponible qu'au code jQuery présent dans ce bloc, non au code extérieur. Cependant, elle peut être référencée par le code du gestionnaire de `click`, qui incrémente cette variable et affiche sa valeur. Une fermeture étant créée, la même instance de `counter` est référencée à chaque clic sur le lien. Cela signifie que les messages affichent une suite de valeurs incrémentées au lieu de `1` :

```
counter = 1
counter = 2
counter = 3
```

Des gestionnaires d'événements peuvent partager leurs environnements de fermeture, comme n'importe quelle autre fonction :

```
$(document).ready(function() {
  var counter = 0;
  $('#example-11 a.add').click(function() {
    counter++;
    $('#example-11').print('counter = ' + counter);
    return false;
  });
  $('#example-11 a.subtract').click(function() {
    counter--;
    $('#example-11').print('counter = ' + counter);
    return false;
  });
});
```

Puisque ces deux fonctions font référence à la même variable `counter`, les opérations d'incrémentation et de décrémentation associées aux deux liens opèrent sur la même variable :

```
counter = 1
counter = 2
counter = 1
counter = 0
```

Les exemples précédents ont utilisé des *fonctions anonymes*, comme nous en avons l'habitude dans notre code jQuery. Du point de vue de la construction des fermetures, cela ne fait aucune différence ; les fermetures peuvent provenir de fonctions anonymes ou de fonctions nommées. Par exemple, nous pouvons écrire une fonction anonyme qui affiche l'indice d'un élément dans un objet jQuery :

```
$(document).ready(function() {
  $('#example-12 a').each(function(index) {
    $(this).click(function() {
      $('#example-12').print('index = ' + index);
      return false;
    });
  });
});
```

Puisque la fonction la plus interne est définie dans la fonction de rappel de `.each()`, le code crée autant de fonctions qu'il y a de liens. Chacune de ces fonctions est attachée au gestionnaire de `click` d'un des liens. L'environnement de fermeture des fonctions comprend `index`, puisqu'il s'agit d'un paramètre de la fonction de rappel de `.each()`. Le fonctionnement est identique à celui que l'on obtiendrait si le gestionnaire de `click` était écrit sous forme d'une fonction nommée :

```
$(document).ready(function() {
  $('#example-13 a').each(function(index) {
    function clickHandler() {
      $('#example-13').print('index = ' + index);
      return false;
    }
    $(this).click(clickHandler);
  });
});
```

La version fondée sur la fonction anonyme est simplement un peu plus courte. L'emplacement de la fonction nommée reste cependant important :

```
$(document).ready(function() {
  function clickHandler() {
    $('#example-14').print('index = ' + index);
    return false;
  }
  $('#example-14 a').each(function(index) {
    $(this).click(clickHandler);
  });
});
```

Cette version génère une erreur JavaScript lors du clic sur un lien, car `index` ne se trouve pas dans l'environnement de fermeture de `clickHandler()`. Il s'agit d'une variable libre, qui n'est donc pas définie dans ce contexte.

C.4 Dangers liés aux fuites de mémoire

En JavaScript, la mémoire est gérée en utilisant la technique dite du *ramasse-miettes*. Il se distingue en cela des langages de bas niveau, comme C, qui obligent les programmeurs à *réserver* des blocs de mémoire et à les *libérer* explicitement lorsqu'ils ne sont plus utilisés. D'autres langages, comme Objective-C, aident le programmeur en mettant en œuvre un mécanisme de *comptage des références*, qui lui permet de connaître le nombre de composants du programme qui utilisent une zone de mémoire précise et de

la nettoyer lorsqu'elle n'est plus utilisée. *A contrario*, JavaScript est un langage de haut niveau qui s'occupe lui-même de ces aspects.

Dès qu'un nouvel élément résidant en mémoire, comme un objet ou une fonction, est rencontré dans le code JavaScript, une zone mémoire lui est réservée. Lorsque l'objet est passé d'une fonction à l'autre et affecté à des variables, d'autres parties du code y font référence. JavaScript conserve une trace de ces pointeurs et, lorsque le dernier a disparu, la mémoire occupée par l'objet est libérée. Prenons l'exemple de la chaîne de pointeurs illustrée à la Figure C.1.

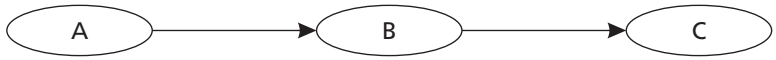


Figure C.1
Objets reliés par des pointeurs.

L'objet A possède une propriété qui pointe sur B, et une propriété de B pointe sur C. Même si l'objet A est le seul qui contient une variable dans la portée courante, les trois objets doivent rester en mémoire en raison des différents pointeurs. En revanche, lorsque A n'est plus dans la portée, par exemple à la fin de la fonction dans laquelle il est déclaré, il peut être collecté par le ramasse-miettes. Ensuite, plus aucun pointeur ne vise l'objet B, qui peut donc être libéré. Enfin, le même raisonnement s'applique à C.

La Figure C.2 présente une chaîne de référence plus complexe à traiter. Nous avons ajouté une propriété à l'objet C pour faire référence à l'objet B. Dans ce cas, même lorsque A est libéré, il existe toujours un pointeur sur B venant de C. Cette *référence circulaire* nécessite un traitement particulier de la part de JavaScript, qui doit remarquer que la boucle est isolée des variables dans la portée.

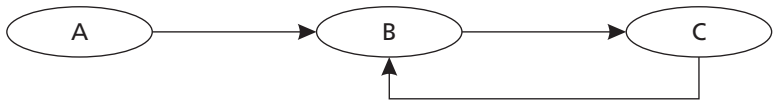


Figure C.2
Références circulaires sur des objets.

Références circulaires accidentelles

Les fermetures peuvent conduire à la création de références circulaires. Puisque les fonctions sont des objets qui doivent être conservés en mémoire, toutes les variables qui se trouvent dans leur environnement de fermeture sont également conservées :

```
function outerFn() {
  var outerVar = {};
  function innerFn() {
    alert(outerVar);
  }
```

```
  outerVar.fn = innerFn;
  return innerFn;
};
```

Dans cet exemple, un objet nommé `outerVar` est créé et référencé depuis le code de la fonction interne `innerFn()`. Ensuite, une propriété de `outerVar` qui pointe sur `innerFn()` est créée, et `innerFn()` est retournée. Cela crée une fermeture sur `innerFn()`, qui fait référence à `outerVar`, qui fait à son tour référence à `innerFn()`. Cependant, la boucle peut être encore plus insidieuse :

```
function outerFn() {
  var outerVar = {};
  function innerFn() {
    alert('Salut');
  }
  outerVar.fn = innerFn;
  return innerFn;
};
```

Dans cette nouvelle version, nous avons modifié `innerFn()` pour qu'elle ne fasse plus référence à `outerVar`. Toutefois, la boucle est toujours là ! Même si `outerVar` n'est pas référencée à partir de `innerFn()`, elle se trouve toujours dans l'*environnement de fermeture* de `innerFn()`. Toutes les variables dans la portée de `outerFn()` sont implicitement référencées par `innerFn()` du fait de la fermeture. Par conséquent, les fermetures peuvent très facilement conduire à des références circulaires.

Fuites de mémoire dans Internet Explorer

En général, toutes ces considérations ne constituent pas un problème, car JavaScript est en mesure de détecter ces boucles et de les nettoyer lorsqu'elles deviennent orphelines. En revanche, Internet Explorer a quelques difficultés à gérer une classe particulière de références circulaires. Lorsqu'une boucle comprend des éléments du DOM et des objets JavaScript normaux, IE est incapable de libérer les uns ou les autres car ils sont pris en charge par des gestionnaires de mémoire différents. Ces boucles sont libérées uniquement à la fermeture du navigateur, ce qui peut mener à une consommation excessive de la mémoire au fil du temps. Le simple gestionnaire d'événements suivant crée une telle boucle :

```
$(document).ready(function() {
  var div = document.getElementById('foo');
  div.onclick = function() {
    alert('Salut');
  };
});
```

L'affectation du gestionnaire de `click` crée une fermeture avec `div` dans un environnement de fermeture. Mais `div` possède une référence de retour vers la fermeture, au travers de la propriété `onclick`. Ainsi, la boucle ne peut pas être libérée par Internet Explorer, même lorsque l'internaute passe à une autre page.

Les bonnes nouvelles

Écrivons à présent le même code, mais avec des constructions jQuery normales :

```
$(document).ready(function() {
  var $div = $('#foo');
  $div.click(function() {
    alert('Salut');
  });
});
```

Même si une fermeture est toujours créée, avec le même type de boucle que précédemment, ce code ne conduit pas à une fuite de mémoire dans IE. En effet, jQuery est au courant des potentialités de fuites et libère manuellement tous les gestionnaires d'événements qu'il affecte. Tant que nous employons les méthodes jQuery de liaison d'événements pour nos gestionnaires, nous n'avons pas à craindre les fuites provoquées par cet idiome particulièrement répandu.

Pourtant, cela ne signifie pas que nous soyons sauvés. Nous devons toujours faire attention lors de la manipulation des éléments du DOM. La liaison d'objets JavaScript à des éléments du DOM peut toujours provoquer des fuites de mémoire dans Internet Explorer. jQuery fait simplement en sorte que ce cas soit moins fréquent.

Voilà pourquoi jQuery met à notre disposition un autre outil pour éviter ces fuites. Au Chapitre 7, nous avons vu que la méthode `.data()` permet d'associer des informations aux éléments du DOM, à la manière des propriétés expando. Puisque ces données ne sont pas enregistrées directement dans une propriété expando (jQuery utilise une mappe interne pour enregistrer les données avec des identifiants qu'il crée), une référence circulaire n'est jamais formée et nous évitons le problème des fuites de mémoire. Dès qu'une propriété expando semble constituer un mécanisme de stockage pratique pour des données, il faut envisager l'utilisation de `.data()` pour plus de sécurité.

C.5 En résumé

Les fermetures JavaScript sont une caractéristique puissante du langage. Elles sont généralement plutôt utiles pour masquer des variables à un autre code, ce qui évite les conflits sur les noms de variables. Puisque des fonctions sont souvent passées en argument des méthodes dans jQuery, la création de fermetures par inadvertance est fréquente. En les maîtrisant, nous pouvons écrire du code plus efficace et plus concis, et, avec un peu de soin et grâce à l'aide des protections intégrées à jQuery, nous pouvons éviter les problèmes de fuites de mémoire auxquels elles peuvent mener.

Annexe D

Référence rapide

Cette annexe se veut une référence rapide à l'API de jQuery, notamment pour les expressions de sélection et les méthodes. Ce sujet est abordé de manière plus détaillée dans le livre *jQuery Reference Guide* et sur le site de la documentation de jQuery à **http://docs.jquery.com**.

D.1 Expressions de sélection

La fonction `$()` de jQuery sert à rechercher des éléments sur la page, pour les manipuler ensuite. Elle prend en argument une *expression de sélection*, sous la forme d'une chaîne de caractères dont la syntaxe s'apparente à celle de CSS. Les expressions de sélection ont été détaillées au Chapitre 2 et sont recensées au Tableau D.1.

Tableau D.1 : Les expressions de sélection

Sélecteur	*Correspondance*
`*`	Tous les éléments.
`#id`	L'élément dont l'identifiant est celui indiqué.
`élément`	Tous les éléments du type indiqué.
`.classe`	Tous les éléments de la classe indiquée.
`a, b`	Les éléments qui correspondent à `a` ou à `b`.
`a b`	Les éléments `b` qui sont des descendants de `a`.
`a > b`	Les éléments `b` qui sont des enfants de `a`.
`a + b`	Les éléments `b` qui viennent juste après `a`.
`a ~ b`	Les éléments `b` qui sont des frères de `a`.
`:first`	Le premier élément dans le jeu résultant.

Tableau D.1 : Les expressions de sélection *(suite)*

Sélecteur	*Correspondance*
`:last`	Le dernier élément dans le jeu résultant.
`:not(a)`	Tous les éléments du jeu résultant qui ne correspondent pas à `a`.
`:even`	Les éléments d'indice pair dans le jeu résultant (commence à 0).
`:odd`	Les éléments d'indice impair dans le jeu résultant (commence à 0).
`:eq(indice)`	L'élément d'indice indiqué dans le jeu résultant (commence à 0).
`:gt(indice)`	Tous les éléments du jeu résultant qui viennent après (supérieurs à) l'indice indiqué (commence à 0).
`:lt(indice)`	Tous les éléments du jeu résultant qui viennent avant (inférieurs à) l'indice indiqué (commence à 0).
`:header`	Les éléments d'intitulé, comme `<h1>` ou `<h2>`.
`:animated`	Les éléments en cours d'animation.
`:contains(texte)`	Les éléments qui contiennent le texte indiqué.
`:empty`	Les éléments sans nœuds enfants.
`:has(a)`	Les éléments dont un élément descendant correspond à `a`.
`:parent`	Les éléments qui ont des nœuds enfants.
`:hidden`	Les éléments masqués, que ce soit par CSS ou ceux de type `<input type="hidden" />`.
`:visible`	L'inverse de `:hidden`.
`[attr]`	Les éléments qui possèdent l'attribut `attr`.
`[attr=valeur]`	Les éléments dont l'attribut `attr` est égal à `valeur`.
`[attr!=valeur]`	Les éléments dont l'attribut `attr` est différent de `valeur`.
`[attr^=valeur]`	Les éléments dont l'attribut `attr` commence par `valeur`.
`[attr$=valeur]`	Les éléments dont l'attribut `attr` se termine par `valeur`.
`[attr*=valeur]`	Les éléments dont l'attribut `attr` contient la sous-chaîne `valeur`.
`:nth-child(indice)`	Les éléments qui sont l'enfant d'indice indiqué de leur élément parent (commence à 1).
`:nth-child(even)`	Les éléments qui sont un enfant d'indice pair de leur élément parent (commence à 1).
`:nth-child(odd)`	Les éléments qui sont un enfant d'indice impair de leur élément parent (commence à 1).

Tableau D.1 : Les expressions de sélection *(suite)*

Sélecteur	*Correspondance*
`:nth-child(`*`formule`*`)`	Les éléments qui sont un enfant d'indice `n` de leur parent (commence à 1). La formule est de la forme `an+b` pour les entiers `a` et `b`.
`:first-child`	Les éléments qui sont le premier enfant de leur parent.
`:last-child`	Les éléments qui sont le dernier enfant de leur parent.
`:only-child`	Les éléments qui sont le seul enfant de leur parent.
`:input`	Tous les éléments `<input>`, `<select>`, `<textarea>` et `<button>`.
`:text`	Les éléments `<input>` de type `text`.
`:password`	Les éléments `<input>` de type `password`.
`:radio`	Les éléments `<input>` de type `radio`.
`:checkbox`	Les éléments `<input>` de type `checkbox`.
`:submit`	Les éléments `<input>` de type `submit`.
`:image`	Les éléments `<input>` de type `image`.
`:reset`	Les éléments `<input>` de type `reset`.
`:button`	Les éléments `<input>` de type `button` et les éléments `<button>`.
`:file`	Les éléments `<input>` de type `file`.
`:enabled`	Les éléments de formulaire activés.
`:disabled`	Les éléments de formulaire désactivés.
`:checked`	Les cases à cocher et les boutons radios sélectionnés.
`:selected`	Les éléments `<option>` sélectionnés.

D.2 Méthodes de parcours du DOM

Après avoir créé un objet jQuery avec `$()`, nous pouvons modifier la collection des éléments correspondants en invoquant les *méthodes de parcours du DOM*. Elles ont été présentées en détail au Chapitre 2 et sont recensées au Tableau D.2.

Tableau D.2 : Les méthodes de parcours du DOM

Méthode	*Retourne un objet jQuery qui contient…*
`.filter(`*`sélecteur`*`)`	Les éléments sélectionnés qui correspondent au sélecteur indiqué.
`.filter(`*`rappel`*`)`	Les éléments sélectionnés pour lesquels la fonction `rappel` retourne `true`.

Tableau D.2 : Les méthodes de parcours du DOM *(suite)*

Méthode	*Retourne un objet jQuery qui contient…*
`.eq(`*`indice`*`)`	L'élément sélectionné qui se trouve à l'indice indiqué (commence à 0).
`.slice(`*`début`*`, [`*`fin`*`])`	Les éléments sélectionnés qui se trouvent dans la plage d'indices indiquée (commence à 0).
`.not(`*`sélecteur`*`)`	Les éléments sélectionnés qui ne correspondent pas au sélecteur indiqué.
`.add(`*`sélecteur`*`)`	Les éléments sélectionnés et tout élément supplémentaire qui correspond au sélecteur indiqué.
`.find(`*`sélecteur`*`)`	Les éléments descendants qui correspondent au sélecteur.
`.contents()`	Les nœuds enfants (y compris les nœuds de texte).
`.children([`*`sélecteur`*`])`	Les nœuds enfants, avec filtrage facultatif par le sélecteur indiqué.
`.next([`*`sélecteur`*`])`	Le frère qui vient immédiatement après chaque élément sélectionné, avec filtrage facultatif par le sélecteur indiqué.
`.nextAll([`*`sélecteur`*`])`	Tous les frères qui suivent chaque élément sélectionné, avec filtrage facultatif par le sélecteur indiqué.
`.prev([`*`sélecteur`*`])`	Le frère qui vient immédiatement avant chaque élément sélectionné, avec filtrage facultatif par le sélecteur indiqué.
`.prevAll([`*`sélecteur`*`])`	Tous les frères qui précèdent chaque élément sélectionné, avec filtrage facultatif par le sélecteur indiqué.
`.siblings([`*`sélecteur`*`])`	Tous les frères, avec filtrage facultatif par le sélecteur indiqué.
`.parent([`*`sélecteur`*`])`	Le parent de chaque élément sélectionné, avec filtrage facultatif par le sélecteur indiqué.
`.parents([`*`sélecteur`*`])`	Tous les ancêtres, avec filtrage facultatif par le sélecteur indiqué.
`.closest(`*`sélecteur`*`)`	Le premier élément qui correspond au sélecteur, en partant de l'élément sélectionné et en remontant l'arborescence du DOM.
`.offsetParent()`	Le parent positionné (par exemple de manière relative ou absolue) du premier élément sélectionné.
`.andSelf()`	Les éléments sélectionnés, ainsi que le jeu d'éléments sélectionnés précédent placé sur la pile interne de jQuery.
`.end()`	Le jeu précédent d'éléments sélectionnés dans la pile interne de jQuery.
`.map(`*`rappel`*`)`	Le résultat de la fonction `rappel` lorsqu'elle est invoquée sur chaque élément sélectionné.

D.3 Méthodes d'événement

Pour réagir aux actions de l'utilisateur, nous devons enregistrer nos gestionnaires en utilisant les *méthodes d'événement*. Notez que de nombreux événements du DOM s'appliquent uniquement à certains types d'éléments ; ces subtilités ne sont pas détaillées dans cette section. Les méthodes d'événement ont été détaillées au Chapitre 3 et sont recensées au Tableau D.3.

Tableau D.3 : Les méthodes d'événement

Méthode	*Description*
`.ready(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le DOM et CSS seront intégralement chargés.
`.bind(`*`type`*`, [`*`données`*`], `*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'événement `type` sera envoyé à l'élément.
`.one(`*`type`*`, [`*`données`*`], `*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'événement `type` sera envoyé à l'élément. La liaison est supprimée après l'appel de `gestionnaire`.
`.unbind([`*`type`*`], [`*`gestionnaire`*`])`	Supprime les liaisons sur l'élément (pour l'événement `type`, le `gestionnaire` particulier ou toutes les liaisons).
`.live(`*`type`*`, `*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'événement `type` sera envoyé à l'élément, avec utilisation de la délégation d'événement.
`.die(`*`type`*`, [`*`gestionnaire`*`])`	Supprime les liaisons sur l'élément précédemment lié avec `.live()`.
`.blur(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'élément perdra le focus du clavier.
`.change(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'élément perdra le focus du clavier et que sa valeur aura été changée.
`.click(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lors d'un clic sur l'élément.
`.dblclick(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lors d'un double-clic sur l'élément.
`.error(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'élément recevra un événement d'erreur (dépend du navigateur).
`.focus(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'élément recevra le focus du clavier.
`.keydown(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une touche sera enfoncée alors que l'élément possède le focus du clavier.

Tableau D.3 : Les méthodes d'événement *(suite)*

Méthode	*Description*
`.keypress(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une touche sera appuyée alors que l'élément possède le focus du clavier.
`.keyup(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une touche sera relâchée alors que l'élément possède le focus du clavier.
`.load(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le chargement de l'élément sera terminé.
`.mousedown(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le bouton de la souris sera enfoncé au-dessus de l'élément.
`.mouseenter(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le pointeur de la souris entrera dans l'élément. Le bouillonnement n'a pas d'effet.
`.mouseleave(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le pointeur de la souris quittera l'élément. Le bouillonnement n'a pas d'effet.
`.mousemove(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le pointeur de la souris se déplacera au-dessus de l'élément.
`.mouseout(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le pointeur de la souris quittera l'élément.
`.mouseover(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le pointeur de la souris entrera dans l'élément.
`.mouseup(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le bouton de la souris sera relâché au-dessus de l'élément.
`.resize(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'élément sera redimensionné.
`.scroll(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque la position de défilement de l'élément changera.
`.select(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque le texte dans l'élément sera sélectionné.
`.submit(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'élément (un formulaire) sera envoyé.
`.unload(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsque l'élément sera retiré de la mémoire.
`.hover(`*`entrée`*`, `*`sortie`*`)`	`entrée` sera invoqué lorsque la souris entrera dans l'élément et `sortie` lorsque la souris en sortira.
`.toggle(`*`gestionnaire1`*`, `*`gestionnaire2`*`,...)`	`gestionnaire1` sera invoqué lors du premier clic de souris sur l'élément, `gestionnaire2` sera invoqué lors du deuxième clic, et ainsi de suite pour les clics suivants.

Tableau D.3 : Les méthodes d'événement *(suite)*

Méthode	*Description*
`.trigger(type, [données])`	Invoque l'événement `type` sur les éléments et exécute l'action par défaut pour l'événement.
`.triggerHandler(type, [données])`	Invoque l'événement `type` sur le premier élément, sans exécuter les actions par défaut ni les événements par délégation.
`.blur()`	Déclenche l'événement `blur`.
`.change()`	Déclenche l'événement `change`.
`.click()`	Déclenche l'événement `click`.
`.dblclick()`	Déclenche l'événement `dblclick`.
`.error()`	Déclenche l'événement `error`.
`.focus()`	Déclenche l'événement `focus`.
`.keydown()`	Déclenche l'événement `keydown`.
`.keypress()`	Déclenche l'événement `keypress`.
`.keyup()`	Déclenche l'événement `keyup`.
`.select()`	Déclenche l'événement `select`.
`.submit()`	Déclenche l'événement `submit`.

D.4 Méthodes d'effet

Les *méthodes d'effet* sont utilisées pour réaliser des animations sur les éléments du DOM. Elles ont été détaillées au Chapitre 4 et sont recensées au Tableau D.4.

Tableau D.4 : Les méthodes d'effet

Méthode	*Description*
`.show()`	Affiche les éléments sélectionnés.
`.hide()`	Masque les éléments sélectionnés.
`.show(vitesse, [rappel])`	Affiche les éléments sélectionnés, avec animation de la hauteur, de la largeur et de l'opacité.
`.hide(vitesse, [rappel])`	Masque les éléments sélectionnés, avec animation de la hauteur, de la largeur et de l'opacité.
`.toggle([vitesse], [rappel])`	Affiche ou masque les éléments sélectionnés.
`.slideDown([vitesse], [rappel])`	Affiche les éléments sélectionnés, avec un effet de glissement.

Tableau D.4 : Les méthodes d'effet *(suite)*

Méthode	*Description*
`.slideUp([vitesse], [rappel])`	Masque les éléments sélectionnés, avec un effet de glissement.
`.slideToggle([vitesse], [rappel])`	Affiche ou masque les éléments sélectionnés, avec un effet de glissement.
`.fadeIn([vitesse], [rappel])`	Affiche les éléments sélectionnés, avec un effet de fondu vers une opacité totale.
`.fadeOut([vitesse], [rappel])`	Masque les éléments sélectionnés, avec un effet de fondu vers une transparence totale.
`.fadeTo(vitesse, opacité, [rappel])`	Ajuste l'opacité des éléments sélectionnés.
`.animate(attributs, [vitesse], [easing], [rappel])`	Effectue une animation personnalisée des attributs CSS indiqués.
`.animate(attributs, options)`	Interface simplifiée de la méthode `.animate()`, pour le contrôle de la file d'animation.
`.stop([viderFile], [allerAFin])`	Arrête l'animation en cours, puis démarre les éventuelles animations placées dans la file.
`.queue()`	Récupère la file d'animation associée au premier élément correspondant.
`.queue(rappel)`	Ajoute la fonction `rappel` à la fin de la file.
`.queue(nouvelleFile)`	Remplace la file par une nouvelle.
`.dequeue()`	Exécute l'animation suivante de la file.

D.5 Méthodes de manipulation du DOM

Les *méthodes de manipulation du DOM* ont été détaillées au Chapitre 5 et sont recensées au Tableau D.5.

Tableau D.5 : Les méthodes de manipulation du DOM

Méthode	*Description*
`.attr(clé)`	Obtient la valeur de l'attribut `clé`.
`.attr(clé, valeur)`	Affecte `valeur` à l'attribut `clé`.
`.attr(clé, fn)`	Affecte à l'attribut `clé` le résultat de la fonction `fn` (invoquée sur chaque élément sélectionné).

Tableau D.5 : Les méthodes de manipulation du DOM *(suite)*

Méthode	*Description*
`.attr(`*`mappe`*`)`	Fixe les valeurs des attributs, donnés sous forme de couples clé-valeur.
`.removeAttr(`*`clé`*`)`	Supprime l'attribut `clé`.
`.addClass(`*`classe`*`)`	Ajoute la classe indiquée à chaque élément sélectionné.
`.removeClass(`*`classe`*`)`	Retire la classe indiquée à chaque élément sélectionné.
`.toggleClass(`*`classe`*`)`	Retire la classe indiquée si elle est présente, sinon l'ajoute, pour chaque élément sélectionné.
`.hasClass(`*`classe`*`)`	Retourne `true` si l'un des éléments sélectionnés possède la classe indiquée.
`.html()`	Récupère le contenu HTML du premier élément sélectionné.
`.html(`*`valeur`*`)`	Fixe le contenu HTML de chaque élément sélectionné à `valeur`.
`.text()`	Récupère le contenu textuel de tous les éléments sélectionnés dans une seule chaîne.
`.text(`*`valeur`*`)`	Fixe le contenu textuel de chaque élément sélectionné à `valeur`.
`.val()`	Récupère l'attribut `value` du premier élément sélectionné.
`.val(`*`valeur`*`)`	Fixe l'attribut `value` de chaque élément à `valeur`.
`.css(`*`clé`*`)`	Récupère l'attribut CSS `clé`.
`.css(`*`clé`*`, `*`valeur`*`)`	Fixe l'attribut CSS `clé` à `valeur`.
`.css(`*`mappe`*`)`	Fixe les valeurs des attributs CSS donnés sous forme de couple clé-valeur.
`.offset()`	Récupère les coordonnées haute et gauche, en pixels, du premier élément sélectionné, relativement à la page.
`.position()`	Récupère les coordonnées haute et gauche, en pixels, du premier élément sélectionné, relativement à l'élément retourné par `.offsetParent()`.
`.scrollTop()`	Récupère la position de défilement vertical du premier élément sélectionné.
`.scrollTop(`*`valeur`*`)`	Fixe à `valeur` la position de défilement vertical de tous les éléments sélectionnés.
`.scrollLeft()`	Récupère la position de défilement horizontal du premier élément sélectionné.

Tableau D.5 : Les méthodes de manipulation du DOM *(suite)*

Méthode	*Description*
`.scrollLeft(`*`valeur`*`)`	Fixe à `valeur` la position de défilement horizontal de tous les éléments sélectionnés.
`.height()`	Récupère la hauteur du premier élément sélectionné.
`.height(`*`valeur`*`)`	Fixe à `valeur` la hauteur de tous les éléments sélectionnés.
`.width()`	Récupère la largeur du premier élément sélectionné.
`.width(`*`valeur`*`)`	Fixe à `valeur` la largeur de tous les éléments sélectionnés.
`.innerHeight()`	Récupère la hauteur du premier élément sélectionné, en incluant l'espacement, mais sans les bordures.
`.innerWidth()`	Récupère la largeur du premier élément sélectionné, en incluant l'espacement, mais sans les bordures.
`.outerHeight([`*`avecMarges`*`])`	Récupère la hauteur du premier élément sélectionné, en incluant l'espacement, les bordures et, facultativement, les marges.
`.outerWidth([`*`avecMarges`*`])`	Récupère la largeur du premier élément sélectionné, en incluant l'espacement, les bordures et, facultativement, les marges.
`.append(`*`contenu`*`)`	Insère du contenu à la fin de chaque élément sélectionné.
`.appendTo(`*`sélecteur`*`)`	Insère les éléments sélectionnés à la fin des éléments qui correspondent au sélecteur.
`.prepend(`*`contenu`*`)`	Insère du contenu au début de chaque élément sélectionné.
`.prependTo(`*`sélecteur`*`)`	Insère les éléments sélectionnés au début des éléments qui correspondent au sélecteur.
`.after(`*`contenu`*`)`	Insère du contenu après chaque élément sélectionné.
`.insertAfter(`*`sélecteur`*`)`	Insère les éléments sélectionnés après chaque élément qui correspond au sélecteur.
`.before(`*`contenu`*`)`	Insère du contenu avant chaque élément sélectionné.
`.insertBefore(`*`sélecteur`*`)`	Insère les éléments sélectionnés avant chaque élément qui correspond au sélecteur.
`.wrap(`*`contenu`*`)`	Enveloppe chaque élément sélectionné dans `contenu`.
`.wrapAll(`*`contenu`*`)`	Enveloppe tous les éléments sélectionnés dans `contenu`.
`.wrapInner(`*`contenu`*`)`	Enveloppe le contenu de chaque élément sélectionné dans `contenu`.

Tableau D.5 : Les méthodes de manipulation du DOM *(suite)*

Méthode	*Description*
`.replaceWith(contenu)`	Remplace les éléments sélectionnés par `contenu`.
`.replaceAll(sélecteur)`	Remplace les éléments qui correspondent au sélecteur par les éléments sélectionnés.
`.empty()`	Retire les nœuds enfants de chaque élément sélectionné.
`.remove([sélecteur])`	Retire du DOM les nœuds sélectionnés, avec filtrage facultatif par le sélecteur.
`.clone([true|false])`	Effectue une copie de tous les éléments sélectionnés, en incluant facultativement les gestionnaires d'événements.
`.data(clé)`	Obtient l'élément de données `clé` associé au premier élément sélectionné.
`.data(clé, valeur)`	Affecte `valeur` à l'élément de données `clé` associé à chaque élément sélectionné.
`.removeData(clé)`	Supprime l'élément de données `clé` associé à chaque élément sélectionné.

D.6 Méthodes AJAX

Grâce aux *méthodes AJAX*, nous pouvons obtenir des informations à partir du serveur sans actualiser la page. Ces méthodes ont été détaillées au Chapitre 6 et sont recensées au Tableau D.6.

Tableau D.6 : Les méthodes AJAX

Méthode	*Description*
`$.ajax(options)`	Effectue une requête AJAX en utilisant les options fournies. Cette méthode de bas niveau est généralement appelée par d'autres méthodes plus commodes.
`.load(url, [données], [rappel])`	Effectue une requête AJAX sur `url` et place la réponse dans les éléments sélectionnés.
`$.get(url, [données], [rappel], [typeRetour])`	Effectue une requête AJAX sur `url` en utilisant la méthode GET.
`$.getJSON(url, [données], [rappel])`	Effectue une requête AJAX sur `url`, en traitant la réponse comme une structure de données JSON.
`$.getScript(url, [rappel])`	Effectue une requête AJAX sur `url`, en évaluant la réponse comme du code JavaScript.

Tableau D.6 : Les méthodes AJAX *(suite)*

Méthode	*Description*
`$.post(`*`url`*`, [`*`données`*`], [`*`rappel`*`], [`*`typeRetour`*`])`	Effectue une requête AJAX sur `url` en utilisant la méthode POST.
`.ajaxComplete(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une requête AJAX sera terminée.
`.ajaxError(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une requête AJAX se terminera par une erreur.
`.ajaxSend(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une transaction AJAX débutera.
`.ajaxStart(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une transaction AJAX débutera alors qu'aucune autre n'est active.
`.ajaxStop(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une transaction AJAX se terminera alors qu'aucune autre n'est active.
`.ajaxSuccess(`*`gestionnaire`*`)`	`gestionnaire` sera invoqué lorsqu'une transaction AJAX se terminera avec succès.
`$.ajaxSetup(`*`options`*`)`	Fixe les options par défaut de toutes les transactions AJAX ultérieures.
`.serialize()`	Encode les valeurs d'un ensemble de contrôles de formulaire dans une chaîne de paramètres.
`.serializeArray()`	Encode les valeurs d'un ensemble de contrôles de formulaire dans une structure de données JSON.
`$.param(`*`mappe`*`)`	Encode une mappe de valeurs dans une chaîne de paramètres.

D.7 Autres méthodes

Les méthodes utilitaires recensées au Tableau D.7 n'entrent pas dans les catégories précédentes, mais elles sont souvent très utiles dans les scripts jQuery.

Tableau D.7 : Autres méthodes utilitaires

Méthode ou propriété	*Description*
`$.support`	Retourne une mappe de propriétés qui indiquent les fonctionnalités et les standards pris en charge par le navigateur.
`$.each(`*`collection`*`, `*`rappel`*`)`	Itère sur une collection, en exécutant la fonction de rappel sur chaque élément.

Tableau D.7 : Autres méthodes utilitaires *(suite)*

Méthode ou propriété	*Description*
`$.extend(`*`cible`*`, `*`ajout`*`,...)`	Modifie l'objet `cible` en lui ajoutant les propriétés provenant des autres objets indiqués.
`$.grep(`*`tableau`*`, `*`rappel`*`, [`*`inverser`*`])`	Filtre le tableau en utilisant la fonction de rappel pour les comparaisons.
`$.makeArray(`*`objet`*`)`	Convertit un objet en un tableau.
`$.map(`*`tableau`*`, `*`rappel`*`)`	Construit un nouveau tableau composé des résultats retournés par la fonction de rappel invoquée sur chaque élément.
`$.inArray(`*`valeur`*`, `*`tableau`*`)`	Détermine si la valeur se trouve dans le tableau.
`$.merge(`*`tableau1`*`, `*`tableau2`*`)`	Fusionne le contenu de deux tableaux.
`$.unique(`*`tableau`*`)`	Retire du tableau tous les éléments du DOM redondants.
`$.isFunction(`*`objet`*`)`	Détermine si l'objet est une fonction.
`$.trim(`*`chaîne`*`)`	Supprime les espaces à la fin de la chaîne.
`$.noConflict([`*`extreme`*`])`	Redonne à `$` la définition qu'il avait avant le chargement de jQuery.
`.hasClass(`*`nomClasse`*`)`	Détermine si un élément sélectionné possède la classe indiquée.
`.is(`*`sélecteur`*`)`	Détermine si un élément sélectionné correspond à l'expression de sélection indiquée.
`.each(`*`rappel`*`)`	Itère sur les éléments sélectionnés, en exécutant la fonction de rappel sur chacun d'eux.
`.length`	Obtient le nombre d'éléments sélectionnés.
`.get()`	Obtient un tableau des nœuds du DOM qui correspondent aux éléments sélectionnés.
`.get(`*`indice`*`)`	Obtient le nœud du DOM qui correspond à l'élément sélectionné d'indice indiqué.
`.index(`*`élément`*`)`	Obtient l'indice du nœud du DOM indiqué au sein du jeu des éléments sélectionnés.

Index

Symboles

$(), fonction
- à propos 14-15, 86-87
- constituants de base
  - classe 14
  - itération explicite, éviter 15
- fonctionnalité 86
- lien de retour au début 86
- méthodes d'insertion, utiliser 87

$(document).ready(), gestionnaire d'événements 30

A

.addClass(), méthode 63

AHAH, à propos 111

AJAX
- à propos 109
- compatibilité avec 109
- délégation d'événement 136
- données, charger 110-124
- fichiers XML 109
- fonctions d'observation 132-136
- fragment HTML, charger 141-143
- gestionnaires
  - de click, ajouter 136
  - lier à nouveau 136
- JavaScript 109
- méthodes
  - .ajaxStart() 133, 134
  - .ajaxStop() 133
  - de bas niveau 139-140
  - liste 395-396
  - .load() 143
- options
  - par défaut, modifier 140-143
  - supplémentaires 139-143
- sécurité 137-139
- XMLHttpRequest, objet 109

.ajaxForm(), méthode 305

.ajaxStart(), méthode 133, 134

.ajaxSubmit(), méthode 306

.animate(), méthode 71

Animations personnalisées, créer
- méthodes
  - .animate() 71
  - .fadeIn() 72
  - .fadeToggle() 72
- propriétés
  - modifier 72-75
  - positionnement CSS 74-75

Anonymes, fonctions 11

Attributs
- autres que de classe, modifier 84-86
- de classe, manipuler 83-87
- .each(), méthode 85

Auto-complétion (dans les formulaires compacts)
- à propos 226-234
- clavier, naviguer au 229-231
- code côté serveur 226-227
- contre recherche dynamique 233-234
- dans le navigateur 227-228
- liste de suggestions, masquer 233
- mécanisme intégré au navigateur 228
- objectif 226
- texte d'un champ de recherche, fixer 228-229

Auto-complétion (dans les formulaires compacts) (suite)
touches
de direction 231-232
Entrée 232

B

.bind(load), syntaxe 296
Blogs
A List Apart 365
Ajaxian 363
conception web par Dustin Diaz 364
développements Internet par Robert Nyman 363
JavaScript ant 363
John Resig 363
jQuery 363
didacticiels 363
programmation/développement web 364
ressources JavaScript de Matt Snider 364
scripts pour le DOM 364
sites de Christian Heilmann 364
utilisations élaborées du DOM 364
Bouillonnement d'événement 290

C

Carrousel d'images
animation de glissement, ajouter 282-284
anomalies dues aux actions de l'utilisateur, éviter 282
code final 297-300
cycle 280, 281
icônes d'action, afficher 284-287
images, agrandir 287-297
à propos 287
animation 293-294
animations, reporter 294-296
badge 292
badge, sur la couverture agrandie 290
bouton Fermer 290-291
couverture agrandie, fermer 289-290
élément singleton, créer 292
indicateur de chargement, ajouter 297
images, changer 280-287
implémenter 276-300
page web, préparer 277-280
révision du style en Javascript 279-280
Chargement de la page, tâches
$(document).ready(), utiliser 30
code
concision 32
planifier l'exécution 29-30
collisions, empêcher 32-33
effectuer 29-33
.noConflict(), méthode 32
scripts multiples 30-32
.clone(), méthode 98
Code, concision
$(), fonction 32
.ready(), fonction 32
Communication client-serveur
boîte à outils AJAX de jQuery 130
formulaire, construire 131
requêtes
GET 126-130
POST 130
CSS
à propos 13
modification en ligne
.addClass(), méthode 63
.css(), méthode 61-65
éléments button 62, 64, 65
objets littéraux 62
référence
fiche de Mezzoblue 362
page d'accueil du W3C 362
position is everything 362
sélecteurs
à propos 15
combinateur d'enfant 17
éléments de liste, styler 17-18
pseudo-classe de négation 18
utiliser 16, 17
.css(), méthode 61-65

D

.data(), méthode 155
.dequeue(), méthode 78
.dialog(), méthode 314
Document XML, charger 120-124
DOM
- à propos 2, 13
- éléments 335
  - accéder 27
  - bouillonnement d'événement 46
  - bouillonnement d'événement, inconvénients 46-47
  - capture d'événement 45
  - hiérarchie 45
- exemple 13
- méthodes de manipulation
  - liste 392-395
  - utiliser 105-106
- méthodes de parcours 337-340
  - à propos 24
  - cellule de catégorie, styler 25-26
  - chaînage 26-27
  - chaînage, avantage 26
  - effet bénéfique 340
  - effet secondaire 339
  - éléments du DOM, accéder 27
  - fonction de filtrage 24
  - liste 387-388
  - pile interne à jQuery 340

Données tabulaires
- à propos 145
- paginer 146-173
- trier 146-173

Données, AJAX
- amélioration progressive 110
- extraire de
  - document XML 125
  - fichiers JavaScript 125
  - fichiers JSON 125
  - fragments HTML 125
- HTML, ajouter 111-114
- liens, associer des gestionnaires d'événements 110
- objets JavaScript
  - manipuler 114-120
  - obtenir 114-116

E

.each(), méthode 85
.effect(), méthode 309-310
Effets
- composés 69-70
- en file d'attente 75
- fonction de rappel 79-81
- fondu en dégradé sur le prompteur de nouvelles 272-274
- jeu d'éléments
  - multiple, manipuler 78-79, 82
  - unique, manipuler 75-78, 82
- méthodes 42
  - .dequeue(), utiliser 78
  - liste 391-392
  - non d'effet, ajouter 77
- multiples, appliquer 75-78
- simultanés, créer 75-82

Éléments
- citations de passages 100
  - styler 104-105
- contexte
  - lier 93-95
  - marquer 93-95
  - numéroter 93-95
- copier 98-105
- déplacer 89-96
- insérer 87-89
- jQuery, premiers pas 101
- méthodes
  - .clone() 98, 100
  - .wrap() 97-98
- notes de bas de page
  - ajouter 95-96
  - marqueurs 95
- opacité 285
- règles
  - CSS, appliquer 92
  - de style, appliquer 100

Événements
bouillonnement 290
inconvénients 46-47
composés 42-44
éléments du DOM, hiérarchie 45
événement click 43
fonctions avancées, afficher 42-43
fonctions avancées, masquer 42-43
.hover(), méthode 42, 44
.toggle(), méthode 42
du clavier (dans le sélecteur de style)
ajouter 56-59
focus clavier 57
.keyCode, propriété 57
keydown 57
keyup 57
keydown 57
keyup 57
liés à un espace de noms 5
méthodes
abrégées 41
liste 389-391
plugins pour 328
sélecteur de style 33-41
traiter 33-41
event, objet
accéder 48
actions par défaut 49-50
délégation d'événement 50-52
exemple 50, 52
méthodes
.is() 51
.preventDefault() 50
.stopPropagation() 49
.target, propriété 48
utiliser 48
Expressions de sélection
créer 353-354
liste 385-387
paramètres
element 353
index 353
matches 354
set 354
pseudo-classe, ajouter 353
utiliser 353, 354

F

.fadeIn(), méthode 72
.fadeIn(slow), méthode 68
.fadeOut, méthode 69
.fadeToggle(), méthode 72
Fermetures 11, 373
arguments de $(document).ready() 379
dans jQuery 378-381
fonctions anonymes
dans jQuery 378
exemple 380
gestionnaires d'événements, affecter 379-381
interagir avec 377-378
mappe utilisée 378
Firefox, outils pour
barre d'outils du développeur 368
Firebug 367
caractéristiques 367
testeur d'expression régulière 368
Venkman 368
Fonctions
$(), à propos 14-15, 86-87
anonymes 11
globales
ajouter à l'espace de noms de jQuery 330
avantage 331
exemple 330
méthodes utilitaires, créer 332
multiples, ajouter 330-331
internes
à propos 373-377
affecter à une variable globale 374
avantages 373
environnement d'exécution JavaScript 376
portée des variables 376-377
ramasse-miettes 376
utilisation 374
valeur de retour 375
lambda 11
.ready() 32

Form, plugin
à propos 305-307
méthodes
.ajaxForm() 305, 306
.ajaxSubmit() 306
options
beforeSubmit 305
sucess 306
target 305
Formulaires
amélioration progressive 200-207
compacts
à propos 222-236
auto-complétion 226-234
champ de recherche, code 234-236
libellé d'un champ 223
libellé, masquer 224
libellé, styler 223
solutions aux problèmes 224, 225
données numériques
boucle .each() 241
bouton Retirer, ajouter 247-252
boutons, ajouter 247
boutons, modifier 248
calculs numériques 240-247
chiffres après la virgule, gérer 243-244
frais de livraison, calculer 246
informations de livraison, modifier 252-255
lignes, supprimer 250
masque de saisie 239
monnaie, mettre en forme 241-242
monnaie, parser 241-242
panier d'achat, code 255-257
panier d'achat, structure de la table 236-239
sous-total, calculer 245
taxes, arrondir 245-246
validation de la saisie 239
messages d'un champ
expression régulière 205-206
légende, insérer 206-207
modifier 203, 205
styles, définir 207
plugins pour 317-318
sélecteurs personnalisés 23-24
style 200-207
à propos 200
case à cocher, manipuler 217-219
case Cocher tout, créer 217-219
formulaire de contact 220-222
groupe, améliorer 200
informations personnelles, embellir 207-209
légende, appliquer 202-203
messages d'un champ, modifier 203-207
validation
ajouter 209-217
champs 210-213
côté client, avantages 210
expression régulière 214
formats d'entrée, implémenter 213-215
règles 209
tâches effectuées par le code 214
vérifier 215-217
Fuites de mémoire, dangers des
problème d'Internet Explorer 383
références circulaires accidentelles 382-383

G

Gestionnaire
d'événements 10
espace de noms d'un événement, utiliser 53
lier à nouveau 53-55
retirer 52-55
passé en argument 31
GNU Public License 4
Graphiques, plugins pour 325-326

H

.hide(), méthode 66
attributs de style en ligne, fixer 66
éléments, restaurer 66
exemple 66, 67
fonctionnalités 66
inconvénients 68

.hide(vitesse), méthode 68
.hover(), méthode 42, 44

I

Images, plugins pour 322-323
Info-bulles 181-186
.insertAfter(), méthode 87
.insertBefore(), méthode 87
Internet Explorer, outils pour
DebugBar 369
Developer Toolbar 368
Drip 369
Microsoft Visual Web Developer 369
.is(), méthode 51

J

JavaScript
compresseurs de code
embellisseur de code 361
JSMin 361
YUI Compressor 361
objet
$.getJSON(), définir 116
$.getJSON(), en tant que fonction globale 116
$.getJSON(), en tant que méthode de classe 116
jQuery global 116
manipuler 114-120
retrouver 114-116
retrouver, avec $.getJSON() 116
script, exécuter 118-120
pagination
ajouter 165-171
boutons de pagination, activer 167-168
données d'événement personnalisées, ajouter 168
numéro de page actuelle, mettre en exergue 169-170
opérations de tri, avec la sélection de page 170-171
sélecteur de page, afficher 166-167
références
boîte à outils JavaScript 361
Centre de développement Mozilla 360
Dev.opera 360
Quirksmode 360
référence JScript de MSDN 360
tri
alphabétique de base 149-153
appliquer 157
balises de regroupement des lignes 149
colonne, mettre en exergue 160-161
croissant, autoriser 161
.data(), méthode 155
décroissant, autoriser 161
données 157-159
exemple 147
expando 155
inverser 161-163
par permutation 154
plugins 154
jQuery
$(), fonction 14-15
à propos 1
actions 2
ajouter 9-11
architecture de plugins 303
document HTML, préparer 6-8
documentation
API de jQuery 359
jQuery graphique 360
navigateur dans l'API de jQuery 359
visionneuse de l'API de jQuery avec Adobe AIR 360
wiki jQuery 359
éléments, insérer 87-89
fonction de rappel 79-81
fonctionnalités
actions multiples, autoriser 4
connaissances en CSS, exploiter 3
couche d'abstraction 3
itération implicite 3
plugins 3
schéma de chaînage 4
frameworks de développement web 365

intérêts 2-3
AJAX 3
méthodes
.animate() 71
d'objet, ajouter 333-337
de manipulation du DOM 105-106
de parcours du DOM 24-27
.insertAfter() 87
.insertBefore() 87
projet
jQuery 1.1 5
jQuery 1.1.3 5
jQuery 1.2 5
jQuery 1.2.6 5
jQuery 1.3 5
jQuery UI 5
phase de développement public 4
références
expressions de sélection 385-387
méthodes AJAX 395-396
méthodes d'effet 391-392
méthodes d'événement 389-391
méthodes de manipulation du DOM 392-395
méthodes de parcours du DOM 387-388
méthodes diverses 396-397
ressources en ligne 359-365
blogs 362-365
compresseurs de code JavaScript 361
documentation jQuery 359-360
référence CSS 362
référence JavaScript 360-361
sélecteurs
CSS 15-18
personnalisés 20-24
télécharger 5-6
texte d'un poème, ajouter 9
utiliser 6

jQuery UI, plugin
à propos 307-315
bouton Effacer les messages, ajouter 315
composants d'interaction 310-312
étendre 312
.dialog(), options de la méthode 314
Dialog, widget 312-315
effets 308-310
animations sur la classe 309
animations sur la couleur 308
.effect(), méthode 309-310
explode 309
fonctions d'easing élaboré 309
thème, appliquer 314
ThemeRoller 315

jQuery, objet 9
code, exécuter 9-11
itération implicite 9-11
méthodes, ajouter
chaînage 336-337
contexte d'objet, examiner 333-336
nouvelle classe, injecter 9

JSON, à propos 115

JSONP, à propos 138

K

.keyCode, propriété 57

keydown, événement 57

keyup, événement 57

L

lambda, fonctions 11

Lightbox, plugins pour 323-325

Lignes d'une table
effet de bandes 174-177
méthodes élaborées 177-178
mettre en exergue 173-181
action de l'utilisateur 178-181

.load(), méthode 143

M

Méthodes
abrégées
.animate() 342
avantages 342
d'événements 340-344

Méthodes (suite)
abrégées
éléments, afficher 342
éléments, masquer 342
personnalisées 343
.addClass() 63
AJAX de bas niveau
à propos 139
possibilités 140
.ajaxForm() 305
.ajaxStart() 133, 134
.ajaxSubmit() 306
.animate() 71
.clone() 98
.css() 61-65
.data() 155
.dequeue() 78
.dialog() 314
diverses, liste 396-397
.each() 85
.effect() 309-310
.fadeIn() 72
.fadeToggle() 72
.hover() 42, 44
.insertAfter() 87
.insertBefore() 87
.is() 51
.load() 143
.noConflict() 32
.preventDefault() 50
.stopPropagation() 49
.toggle() 42
.wrap() 97-98
MIT License 4
Modulo (%), opérateur 266
MSDN JScript, à propos 360

N

.noConflict(), méthode 32

O

Opera, outils pour, Dragonfly 370
Outils
Aptana 371
Charles 371
Fiddler 371
Firebug Lite 370
NitobiBug 370
pour Firefox 367-368
pour Internet Explorer 368-369
pour Opera 370
pour Safari 369-370
TextMate pour jQuery 371

P

Pagination
code 171-173
côté serveur 164-165
à propos 164
tri et pagination 164-165
JavaScript 165-171
Paramètres de méthode
à propos 344
fonction de rappel 349-351
employer 349-351
mappes 347-348
ombre sur un bloc 344
par défaut, personnaliser 351-352
simples 346
valeurs par défaut 348-349
PHP, langage de script 126
Plugins
catalogue 303
développer
conventions de nommage 356
expression de sélection, ajouter 353-354
fonctions globales, ajouter 329-332
interfaces de méthodes 356-357
méthodes abrégées, ajouter 340-344
méthodes de parcours du DOM 337-340
paramètres de méthodes 344-352
règles d'écriture 355-357
style de documentation 357

pour les événements 328
hoverIntent 328
Live Query 328
pour les formulaires 317-318
Autocomplete 317
Jeditable 318
Masked Input 318
Validation 317
pour les graphiques 325-326
Flot 326
Sparklines 326
pour les images 322-323
Jcrop 322
Magnify 323
pour les tables 319-321
Flexigrid 320
jqGrid 320
Tablesorter 320
pour lightbox 323-325
BlockUI 325
FancyBox 323
jqModal 325
Thickbox 324
rechercher 303-304
utiliser 304-305

Pointeurs 382

.preventDefault(), méthode 50

Prompteur de nouvelles
à propos 260
code final 274-276
dégradation élégante, scénario 260
effet de fondu en dégradé 272-274
flux, obtenir 261-264
d'un domaine différent 271-272
gestionnaire de réussite 261
indicateur de chargement, ajouter 271-272
préparer 264-265
page web 260-261
problème d'utilisabilité, résoudre 268-270
rotation des titres, fonction 265-268

R

Ramasse-miettes 381

.ready(), fonction 32

Recherche dynamique 233

Références circulaires 382

S

Safari, outils pour
à propos 369-370
inspecteur web 369
menu Développement 369

Sécurité
balise HTML <iframe>, utiliser 138
données distantes
charger 137
format JSONP 138-139

Sélecteur de style
boutons
Colonne étroite, activer 36
Grande police, activer 34
Par défaut, activer 36
code, exécuter 40
contexte
d'événement, exploiter 40
de gestionnaire 36-39
événements du clavier, ajouter 56-59
remanier 39

Sélecteurs
d'attribut
à propos 18-20
classes, ajouter 19
liens, styler 19-20
styles, définir 19
personnalisés 20-24
à propos 20-24
effet de bande 21-23
numérotation à partir de un 20
numérotation à partir de zéro 20
pseudo-classe CSS, syntaxe 20

.show(), méthode 66
exemple 66, 67
fonctionnalités 66
inconvénients 68

.show(fast), méthode 68
.show(normal), méthode 68
.show(slow), méthode 68
.show(vitesse), méthode 68
.stopPropagation(), méthode 49
Styles
cellule de catégorie 25-26
liens 19-20
lignes alternées 21-23
Système de comptage des références 381

T

Tables
aspect, modifier
clickable, classe 182
code JavaScript 195-197
filtrer 189-195
info-bulle 181-186
info-bulle, fonction showTooltip() 185
info-bulle, masquer 183
info-bulle, placer 183
info-bulle, positionner 182
info-bulle, style du texte 184
lignes, mettre en exergue 173-181
sections, développer 187-189
sections, réduire 187-189
filtrage
développer 194-195
effet de bandes 193-194
implémenter 190-192
options, extraire du contenu 191-192
réduire 194-195
retirer 192
plugins pour 319-321
.toggle(), méthode 42
Tri
à propos 146
alphabétique en JavaScript
appliquer 156-157
comparateur, utiliser 150-153
dégradation élégante, exemple 152
méthode JavaScript .sort(), utiliser 150
plugins, modifications 154
côté serveur 146-147
actualisation de la page, éliminer 147
amélioration progressive, exemple 147
chaîne de requête, utiliser 146
en JavaScript 147-163
et pagination, code 171-173
.trigger(), méthode
dégradation élégante, implémenter 56
page
développer 55
réduire 55
raccourcis 56

V

Venkman, outil pour Firefox 368

W

W3C, à propos 362
.wrap(), méthode 97-98

X

(X)HTML
page d'accueil du W3C 362
référence 361-362
XML Path (XPath) 18
XMLHttpRequest, objet 109

Référence

jQuery

Simplifiez et enrichissez vos développements JavaScript

Pour mettre en œuvre des sites interactifs attrayants, les développeurs se tournent vers des bibliothèques JavaScript, qui leur permettent d'automatiser les tâches courantes et de simplifier les plus complexes. jQuery, la plus populaire d'entre elles, est particulièrement appréciée des développeurs tant pour sa cohérence conceptuelle que ses performances.

Dans cet ouvrage, les auteurs partagent leurs connaissances, leur expérience et leur passion pour jQuery afin de vous aider à comprendre comment cette bibliothèque fonctionne et vous permettre d'en tirer le meilleur parti. Si vos précédentes tentatives de développement JavaScript vous ont laissé perplexe, ils vous aideront à franchir les obstacles dressés par AJAX, les événements, les effets et les fonctionnalités avancées du langage JavaScript.

Ce livre vous apportera tous les outils dont vous avez besoin pour rester à l'avant-garde du développement web.

TABLE DES MATIÈRES

- Premiers pas
- Sélecteurs
- Événements
- Effets
- Manipulation du DOM
- AJAX
- Manipulation des tableaux
- Manipulation des formulaires
- Carrousels et prompteurs
- Utilisation des plugins
- Développement de plugins
- Ressources en ligne
- Outils de développement
- Fermetures en JavaScript
- Référence rapide

À propos des auteurs

Jonathan Chaffer, directeur technique d'une agence web du Michigan, participe activement au projet Drupal, qui a adopté jQuery comme framework JavaScript.

Karl Swedberg, développeur web, est membre de l'équipe du projet jQuery et contributeur actif à la liste de discussion jQuery.

Programmation
Développement web

Niveau : Tous niveaux
Configuration : Multiplate-forme

PEARSON

Pearson Education France
47 bis, rue des Vinaigriers
75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

ISBN : 978-2-7440-4142-6