Christian Wenz

LE GUIDE DE SURVIE

JavaScript®

L'ESSENTIEL DU CODE ET DES COMMANDES

PEARSON

JavaScript®

Christian Wenz

Publié par CampusPress
47 bis, rue des Vinaigriers
75010 PARIS
Tél : 01 72 74 90 00

Réalisation PAO : Léa B

Auteur : Christian Wenz

ISBN : 978-2-7440-4003-0

Titre original : *JavaScript® Phrasebook*

Traduit de l'américain par :
Nathalie Le Guillou de Penanros

ISBN original : 0-672-32880-1

Sams Publishing
800 East 96th,
Indianapolis, Indiana 46240 USA

Table des matières

A propos de l'auteur

Christian Wenz est un orateur professionnel, auteur, formateur et consultant, spécialisé dans les technologies du Web. Il a rédigé ou a participé à l'écriture de plus de quarante ouvrages. Il collabore régulièrement à des magazines d'informatique réputés et participe à des conférences dans le monde entier. Christian Wenz a contribué à l'élaboration de plusieurs paquetages PHP dans le dépôt PEAR et assure par ailleurs la maintenance d'un module Perl CPAN. Il a obtenu un diplôme d'informatique à l'université technique de Munich, en Allemagne, où il vit et travaille actuellement. C'est également le premier professionnel européen à avoir obtenu la certification Zend et le principal fondateur du Consortium sur la sécurité PHP.

Introduction

En 1999, j'ai écrit un livre sur JavaScript. Au tout début, il s'est très bien vendu, puis les ventes ont commencé à diminuer. Elles se sont pourtant suffisamment stabilisées pour parvenir à la septième édition cet automne, même si un léger déclin se faisait toujours sentir.

Tout a considérablement changé à la fin de l'année dernière : les ventes ont soudain remonté, tout comme celles des autres titres du même segment. Cette évolution tient en partie à AJAX. La technologie en elle-même n'est pas nouvelle mais le terme l'est. En février 2005, Jesse James Garrett établit l'acronyme ; depuis lors, le monde du Web semble comme pris de folie. Et même s'il est possible d'expliquer AJAX en deux minutes, il faut une bonne connaissance des divers aspects de JavaScript pour le comprendre. Cela explique la demande croissante d'informations poussées sur JavaScript et a aussi mené à l'écriture de ce *Guide de survie JavaScript*.

Lorsque nous avons créé la série d'ouvrages en 2005, nous c'est-à-dire Damon Jordan, Mark Taber et moi-même, nous voulions créer une sorte de version adaptée des lexiques de langue : ici, les phrases et les expressions communes sont traduites dans une langue étrangère, le JavaScript. Mais à la différence des lexiques ordinaires, cet ouvrage propose également des explications sur le code. Sans cela, vous risqueriez fort de vous trouver dans des situations embarrassantes, quelle que soit la langue.

Cet ouvrage n'est pas une introduction à JavaScript. Les fonctions élémentaires y sont traitées mais nous avons tenté de mettre l'accent sur des informations de niveau intermédiaire à avancé. L'idée est que, notamment si vos connaissances de JavaScript sont un peu rouillées, vous puissiez trouver les principaux problèmes et leurs solutions dans cet ouvrage. Il vous servira donc de référence pour surmonter rapidement les problèmes que vous rencontrez lors du développement. N'hésitez pas à le parcourir pour découvrir des fonctions JavaScript auxquelles vous n'auriez peut-être pas pensé auparavant.

Ce livre n'est pas non plus un livre de recettes présentant des solutions interminables et inflexibles à des problèmes sans importance. Notre objectif était de limiter autant que possible les extraits de code pour une approche pratique. Cela vous permet d'adapter la technique présentée à vos propres applications et à un scénario spécifique. Pour y parvenir, nous ne présentons que les éléments de code essentiels à l'exemple. Le code est généralement constitué d'éléments `<script>` et de quelques autres balises HTML pour lier le tout. Une application Web moderne doit au moins être compatible avec le XHTML, mais ce n'est pas l'objet de notre ouvrage.

Nous avons pris grand soin à ce que le code fonctionne sur autant de navigateurs que possible. Et même si Internet Explorer et diverses moutures de Mozilla (y compris Firefox) dominent le marché, Opera, Safari et Konqueror y ont aussi leur place. Ainsi, même si l'accent est mis sur les deux premiers types de navigateurs, nous préciserons les incompatibilités ou les problèmes existant avec des navigateurs moins importants. Et en ce qui concerne les parts de marché, seuls les navigateurs toujours actuels seront traités ; ainsi, nous ne ferons jamais mention des versions 4 de Netscape ou d'Internet Explorer.

Les séquences de code de ce titre ainsi que les mises à jour se trouvent sur le site **http://JavaScript.phrasebook.org/**. Un nom est associé à la plupart des listings, pour que vous puissiez rapidement retrouver les fichiers correspondant à chaque extrait. Si vous souhaitez nous faire part de vos commentaires ou si vous rencontrez une erreur, faites-nous le savoir ! Si une expression que vous souhaitiez trouver n'est pas mentionnée, n'hésitez pas à nous contacter. Dites-nous si vous pensez à des éléments qui pourraient apparaître dans les prochaines éditions de cet ouvrage, mais vous pouvez également nous faire part de celles qui vous semblent superflues. La liste des expressions potentielles était bien plus longue que ce que nous avons conservé, nous avons donc dû passer par une phase pénible qui a consisté à sacrifier du contenu ; nous espérons que le résultat vous satisfera.

Christian Wenz

1

Les bases de JavaScript

Ce chapitre traite de certaines bases concernant JavaScript. Il ne s'intéresse pas explicitement à la syntaxe du langage, un aspect abordé dans suffisamment de didacticiels et d'ouvrages et qui n'entre pas dans l'objectif de cet ouvrage. Toutefois, nous expliquerons en détail des aspects essentiels comme l'insertion de code JavaScript dans une page. Nous ferons également un peu d'histoire et relaterons les faits de guerre des navigateurs pour vous préparer au chapitre suivant.

Compréhension de JavaScript (et de son histoire)

JavaScript est un langage de script côté client, ce qui signifie qu'il s'exécute côté client, dans un navigateur Web. JavaScript peut aussi être employé côté serveur et en dehors d'un navigateur, mais ce n'est pas l'objet de cet ouvrage. Si le navigateur est compatible, JavaScript donne

accès à la page en cours et permet au script de déterminer les propriétés du client, de rediriger l'utilisateur vers une autre page, d'accéder aux cookies, etc.

JavaScript naît en septembre 1995, parallèlement à la sortie de la version 2.0 du navigateur Netscape, la première à être dotée du langage de script. A cette époque, le langage s'appelle Mocha puis, à sa sortie, LiveScript ; Netscape conclut ensuite un accord marketing avec Sun (le créateur de Java) et décide de renommer le langage en décembre de cette année, il devient JavaScript.

Le concept connaît immédiatement du succès : Microsoft en intègre une prise en charge dans Internet Explorer versions 3 et suivantes (au milieu de l'année 1996). Pour des raisons légales, une mouture de Microsoft du langage est alors appelée JScript. JScript était plus ou moins compatible avec JavaScript mais a commencé à inclure des fonctions supplémentaires, spécifiques à Internet Explorer (ce qui, à quelques exceptions près, ne s'est jamais véritablement installé).

En 1997, est publiée la norme ECMAScript (ECMA-262) ; JavaScript en est donc la première implémentation. La norme ne précise que le langage et non les fonctions des hôtes environnants (par exemple, comment accéder à la fenêtre courante du navigateur ou en ouvrir une nouvelle). ECMAScript devient norme ISO en 1998.

Aux alentours de 1997 ou 1998, la guerre des navigateurs entre Netscape et Microsoft atteint son apogée, les deux fournisseurs ajoutant une nouvelle fonctionnalité incompatible à la version 5 de leurs navigateurs. Le reste n'est qu'histoire : Netscape abandonne l'idée de publier une version 5 du navigateur et décide de tout recommencer avec Netscape 6 ; Internet Explorer peut ainsi augmenter ses parts de marché, passant à plus de 90 %.

Il a d'ailleurs fallu plusieurs années au projet Mozilla, alors en cours de création, pour revenir à la vie. A noter que le navigateur Firefox est fondé sur Mozilla et commence alors à gagner des parts de marché sur Microsoft.

Du point de vue de JavaScript, peu de choses ont évolué au cours de ces dernières années. Firefox 1.5, sorti fin 2005, prend en charge la nouvelle version JavaScript 1.6, mais les modifications sont assez limitées et Internet Explorer est loin de les prendre en charge. Mais avec Internet Explorer 7 et Firefox 2.0 bientôt à paraître (et déjà disponibles en version test), l'époque est intéressante pour les développeurs Web.

D'autres navigateurs prennent en charge JavaScript. Les différences sont subtiles mais peuvent parfois être très embarrassantes lors du développement avec une application Web indifférente aux navigateurs. Parmi les navigateurs qui acceptent actuellement JavaScript, on trouve :

- Internet Explorer ;
- Mozilla et ses dérivés (Firefox, Epiphany, Camino, Galeon, etc.) ;
- Opera ;
- Konqueror ;
- Safari.

Etablissement d'un système de test

Nous venons de le mentionner, plusieurs navigateurs acceptent JavaScript. Il faut généralement assurer une prise en charge pour la plupart d'entre eux. Ainsi, par exemple, le site Web **http://marketshare.hitslink.co/report.apsx?qprid=3** montre qu'en mars 2006, Internet

Explorer et Firefox ensemble comptaient pour près de 95 % de la part de marché des navigateurs, suivis par Safari (un peu plus de 3 %). Les navigateurs Netscape détenaient environ 1 % et Opera 0,5 %, à peu près la même chose que tous les autres navigateurs réunis (y compris Konqueror).

Quelle est donc la meilleure stratégie pour tester un site Web sur un maximum de systèmes, avec le moins d'efforts possible ?

En fait, tout dépend du public auquel s'adresse votre site Web. Si vous ciblez principalement des utilisateurs de Mac, vous devrez procéder à un test important sur le navigateur Safari, puisqu'il est livré par défaut avec les versions récentes de Mac OS X.

Quel que soit le type de site Web utilisé, Internet Explorer bénéficie toujours d'une très forte part de marché, même sur des sites Web plutôt centrés sur l'open source. Pour le test, il vous faut donc Internet Explorer et un système Windows (ou au moins un élément du style Virtual PC ou VMware, avec une machine virtuelle Windows). Tous les navigateurs Mozilla partagent la même base de code pour le rendu et pour JavaScript, peu importe donc la plate-forme que vous utilisez (même s'il existe des différences minimes). Vous pourriez, par exemple, utiliser Firefox sur la machine Windows sur laquelle réside également votre installation Internet Explorer.

Opera s'exécute aussi sous Windows (et quelques autres systèmes, notamment Linux et Mac), la part de Windows comprend donc un navigateur de plus.

Les deux seuls grands navigateurs restants sont Safari et Konqueror, ce dernier étant le navigateur par défaut pour l'utilisation du gestionnaire de fenêtres KDE.

Par chance, tous deux partagent plus ou moins la même base de code : Safari utilise le moteur KHTML, au cœur du rendu de Konqueror. Deux options sont alors possibles :

- établir une boîte Linux (ou une machine virtuelle Linux) avec KDE et Konqueror ;
- établir un système Mac (ou acheter un Mac Intel avec BootCamp pour avoir un double démarrage sous Windows et OS X).

Vous devriez ainsi obtenir un bon système pour commencer le test. Plus le site Web grandit, plus grand sera le nombre des systèmes cible que vous devrez prendre en charge ; à partir d'un certain point, vous n'aurez pas d'autre choix que d'installer et de tester tous les navigateurs que vous souhaitez utiliser.

En ce qui concerne Internet Explorer, la version 6 est la principale, la version 5.x a presque disparu et les versions 4 et antérieures n'existent plus depuis longtemps. Le test sous IE 6 convient donc la plupart du temps. Disposer de versions différentes est bien entendu assez souhaitable mais nécessite généralement un système Windows pour chacun d'entre eux, Windows ne pouvant être installé qu'une fois sur un système.

Une solution a été découverte par accident, pour l'installation parallèle de plusieurs versions d'IE. Vous en trouverez la description originale à l'adresse **http://labs.insert-title.com/labs/Multiple-IEs-in-Windows_article795.aspx** et de plus amples informations à l'adresse **http://www.positioniserything.net/articles/multiIE.html**.

Enfin, testez votre site Web lorsque JavaScript est activé et désactivé dans les navigateurs.

Configuration de navigateurs Web

Par défaut, la plupart des navigateurs Web activés pour JavaScript prennent en charge ce langage. D'ailleurs, la toute première version de Netscape à accepter JavaScript ne disposait même pas d'une fonction pour le désactiver !

Il est toutefois possible de le désactiver, vous devez donc trouver comment simuler cette situation (et comment demander aux utilisateurs de l'activer). Cela dépend non seulement des navigateurs utilisés mais parfois aussi de la version du navigateur. Dans le navigateur Firefox 1.5, JavaScript peut être activé dans le menu Outils, Options, Contenu, Activer JavaScript. Sous Internet Explorer 6, vous devez creuser un peu plus. Cliquez sur Outils, Options Internet, Sécurité, zone Internet, Personnaliser le niveau, Script, Active Scripting, Activer.

Astuce

Internet Explorer versions 6.0 et 7.0 (sous Windows XP, 2003 et Vista uniquement) possède une fonction de sécurité qui empêche l'exécution de JavaScript sur les pages locales (voir la Figure 1.1). Cela est en fait assez utile mais peut se révéler ennuyeux lorsque vous testez une application. Il existe deux contournements : utilisez un serveur Web local pour tester votre application ou désactivez simplement le message d'erreur en choisissant Outils, Options Internet, Avancé, Sécurité, Autoriser le contenu actif à s'exécuter dans les fichiers de la zone Ordinateur local.

Figure 1.1 : Message d'erreur assez ennuyeux avec JavaScript sur les pages locales.

Inclusion du code JavaScript

```
<script type="text/JavaScript">
  window.alert("Welcome to JavaScript!");
</script>
```

Le code JavaScript peut se présenter de deux manières : intégré dans une page HTML ou intégré dans un fichier externe. La manière la plus fréquente de l'inclure consiste à utiliser les éléments `<script>`. Vous pouvez le placer n'importe où, il est alors exécuté après que cette partie de la page HTML a été chargée et analysée. Le code qui précède (ficher `script.html` dans l'archive à télécharger) ouvre une fenêtre modale et affiche un texte plutôt simple.

L'attribut `type` prévoit le type MIME pour JavaScript. Précédemment, on utilisait `language="JavaScript"` ; toutefois, puisque ce n'était pas standardisé, il vaut mieux utiliser `type` et le type MIME à la place. Dans cet ouvrage, nous suivons la méthode adoptée par de nombreux sites Web de nos jours : on utilise à la fois `type` et `language`.

De même, par le passé, il était possible de cibler un script sur un numéro de version spécifique de JavaScript, comme ceci :

```
<script language="JavaScript1.6">
   window.alert("Only with JavaScript 1.6 !");
</script>
```

Ce n'est quasiment plus usité. L'implémentation de cette fonction génère un certain nombre de bogues dans les navigateurs, et il existe de meilleures manières de tester les capacités JavaScript d'un navigateur.

Info

Dans certains anciens didacticiels, vous trouverez des conseils pour utiliser les commentaires HTML présentés de la façon suivante :

```
<script language="JavaScript"><!--
  // ...
//--></script>
```

Cela servait auparavant à gérer les navigateurs qui ne savaient rien de JavaScript. Or, même ceux qui n'acceptent pas JavaScript connaissent l'élément <script> et savent l'ignorer (lui et son contenu). Ces commentaires HTML ne sont donc plus nécessaires.

Utilisation de fichier(s) JavaScript externe(s)

```
<script language="JavaScript" type="text/JavaScript"
   src="script.js"></script>
```

Il est très pratique d'utiliser un fichier JavaScript externe (ou plusieurs), notamment lorsque vous réutilisez du code JavaScript dans votre site Web. Il ne contient que le

code JavaScript et pas d'éléments `<script>`. Un élément `<script>` est toutefois utilisé pour le charger, comme on le voit dans le listing précédent (fichier `scriptsrc.html`). L'attribut `src` contient l'URL du script externe ; les URL absolues (et donc les serveurs distants) sont également possibles.

Attention

Sachez que le code du fichier externe n'est disponible qu'après le chargement complet du fichier externe. Ainsi, notamment lorsque vous appelez cette fonctionnalité du fichier externe à partir de la page locale, n'oubliez pas que le fichier externe risque de ne pas être encore disponible.

Chargement dynamique de fichiers JavaScript

```
var s = document.createElement("script");
```

Il est parfois nécessaire de charger du code JavaScript à la demande, alors même qu'un site s'exécute. Par exemple, selon ce que saisit l'utilisateur, il faudra peut-être charger un fichier JavaScript externe.

Une manière consiste à utiliser `document.write()` pour ajouter dynamiquement un nouvel élément `<script>` à la page, mais cela ne fonctionne pas avec certains navigateurs et n'est donc pas recommandé. Une bien meilleure solution consiste à utiliser DOM (voir Chapitre 5). Vous créez d'abord un nouvel élément `<script>` et définissez les attributs appropriés. Vous ajoutez ensuite cet élément au DOM de la page. Le code est généralement placé dans la section `<head>`. Le listing suivant montre l'ensemble du code ;

notez qu'il y a un élément `<head>` pour faire fonctionner le code. La Figure 1.2 montre le résultat de ce code.

```
<html>
<head>
<title>JavaScript</title>
</head>
<body>
<script language="JavaScript"
  type="text/JavaScript">
  var s = document.createElement("script");
  s.setAttribute("type", "text/JavaScript");
  s.setAttribute("language", "JavaScript");
  s.setAttribute("src", "script.js");
  document.getElementsByTagName("head")[0].appendChild(s);
</script>
</body>
</html>
```

Ajout dynamique d'un script (scriptdynamic.html)

Pseudo-URL dans JavaScript

```
<a href="JavaScript:window.alert('Welcome to JavaScript!');
➥">click here for a surprise</a>
```

Une autre manière d'appeler du code JavaScript consiste à utiliser une pseudo-URL. Lorsqu'une URL qui commence par `JavaScript:` est chargée, le code qui se trouve derrière est exécuté, comme on le voit dans le code précédent (fichier `url.html`).

Il existe plusieurs manières d'utiliser cette URL, sous la forme d'une image, d'un lien ou d'une CSS (*Cascading Style Sheets,* feuilles de style en cascade) mais on utilise généralement un lien. Sachez toutefois que ce lien fonc-

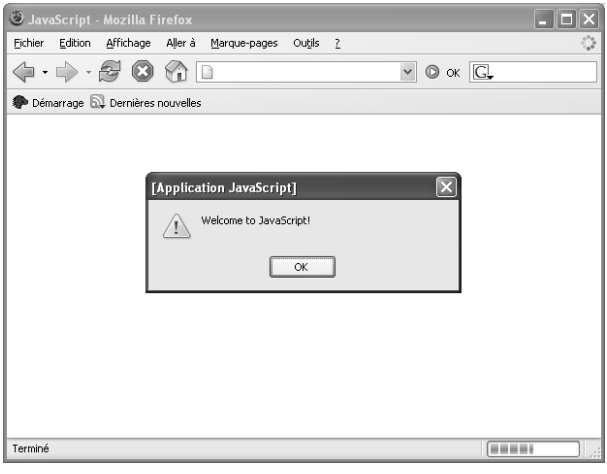

Figure 1.2 : La fenêtre modale provient du fichier externe qui a été chargé de manière dynamique.

tionne uniquement avec un navigateur compatible JavaScript et sur lequel il est activé.

Attention

Lorsque le code suivant le préfixe d'URL `JavaScript:` renvoie quelque chose, le résultat est affiché à l'écran, ce qui n'est généralement pas souhaitable. Vous pouvez utiliser la fonction spéciale de JavaScript `void()` pour l'éviter :

```
JavaScript:void(code_qui_renvoie_quelquechose());
```

Exécution de JavaScript avec des gestionnaires d'événements

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

La troisième manière d'exécuter du code JavaScript (les deux premières étant les éléments `<script>` et les pseudo-URL `JavaScript:`) se fait par l'intermédiaire d'un gestionnaire d'événements. La plupart des éléments HTML acceptent quelques événements ; par exemple, la balise `<body>` accepte l'élément de chargement. En utilisant le préfixe `on`, vous pouvez joindre le code à cet événement (vous trouverez plus d'options au Chapitre 6, "Programmation orientée objet et événements"). Ainsi, le code suivant exécute la fonction `showText()` après que le document a été totalement chargé (par rapport au marquage HTML de la page, et non des images ou d'autres données externes).

```
<html>
<head>
<title>JavaScript</title>
<script language="JavaScript"
  type="text/JavaScript">
  function showText() {
    window.alert("Welcome to JavaScript!");
  }
</script>
</head>
<body onload="showText();">
</body>
</html>
```

Utilisation d'un gestionnaire d'événements JavaScript (event.html)

Attention

Une erreur commune consiste à croire que le préfixe URL `JavaScript:` peut être utilisé avec les gestionnaires d'événements, de la manière suivante :

```
<body onload="JavaScript:showText();">
```

Mais c'est une légende : en effet, pourrait-il en être autrement si le code JavaScript ne pouvait avoir la valeur d'un attribut de gestionnaire d'événements ? Omettez donc `JavaScript:` et fournissez simplement le code à exécuter lorsque l'événement associé est déclenché.

Navigateurs sans JavaScript

```
<script type="text/JavaScript">
  document.write("Welcome to JavaScript!");
</script>
<noscript>
  Welcome to plain HTML!
</noscript>
```

Selon des études récentes, jusqu'à 10 % des utilisateurs ont désactivé JavaScript dans leurs navigateurs, du fait des règlements des entreprises, de la crainte des vulnérabilités de sécurité et pour d'autres raisons. Vous devez donc vous assurer que votre site Web puisse être utilisé également sans JavaScript.

Pour y parvenir, vous pouvez utiliser l'élément `<noscript>`. Les navigateurs dans lesquels JavaScript est activé ignorent cet élément et son contenu, tandis que ceux dans lesquels il est désactivé en affichent le contenu. Le code qui précède (fichier `noscript.html`) génère le résultat présenté à la Figure 1.3 lorsqu'on utilise un navigateur sans JavaScript (comme le navigateur texte Lynx) ; la Figure 1.4 montre un navigateur qui prend en charge le langage de script.

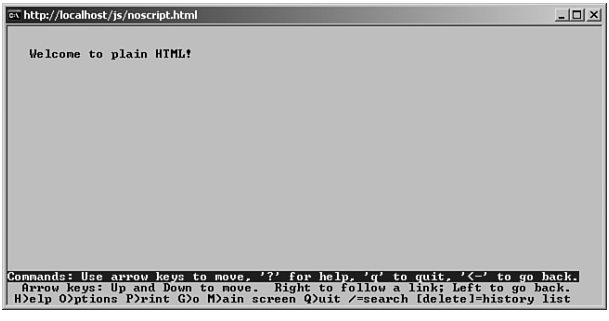

Figure 1.3 : Le code précédent dans un navigateur sans JavaScript.

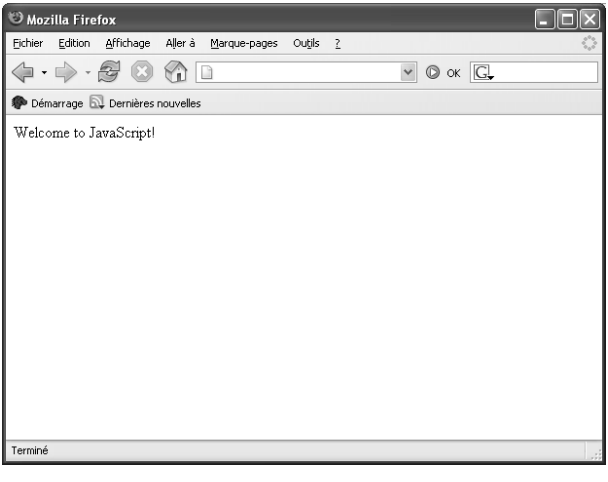

Figure 1.4 : Le code précédent dans un navigateur avec JavaScript.

Lorsque JavaScript est utilisé avec des liens, l'extrait suivant ne fonctionne pas comme prévu :

```
<a href="#" onclick="window.alert('Welcome to
    ➥JavaScript!'); ">click here</a>
```

A première vue, tout est correct : les utilisateurs possédant JavaScript obtiennent la fenêtre modale, ceux qui ne l'ont pas cliquent sur un lien menant vers une étiquette de texte vide ; rien d'inattendu ne survient.

Toutefois, cela présente un petit inconvénient : lorsque l'utilisateur a cliqué sur le lien JavaScript, un navigateur activé pour JavaScript affiche la fenêtre, mais suit tout de même le lien. Généralement, cela n'a aucun effet, mais le navigateur passe en haut de la page car l'étiquette de texte n'a pas été trouvée.

Pour éviter cela, vérifiez simplement que le code du gestionnaire d'événements renvoie `false`. Cela annule tous les autres effets que l'événement actuel peut avoir ; ici, le lien ne sera pas suivi :

```
<a href="#" onclick="window.alert('Welcome to
    ➥JavaScript!'); return false;">click here</a>
```

Eviter que les navigateurs activés pour JavaScript suivent le lien (link.html)

2

Expressions communes

Certaines tâches JavaScript récurrentes doivent être réalisées quasiment chaque jour. Elles sont à la base de nombreuses applications JavaScript mais n'entrent dans aucune catégorie particulière. Nous présentons donc au cours de ce chapitre une suite de problèmes fréquents, avec leurs solutions.

Détection du type de navigateur

```
window.alert(navigator.appName);
```

Même si les implémentations de navigateurs par JavaScript sont, de nos jours, assez compatibles les unes avec les autres (en particulier si on se souvient de l'époque de la guerre des navigateurs, à la fin des années 1990), la détection du type de navigateur constitue une partie essentielle de la boîte à outils du développeur JavaScript.

L'objet JavaScript `navigator` propose des informations sur le navigateur. Sa propriété `userAgent`, qui contient la chaîne d'identification complète du navigateur, est très utile mais parfois difficile à analyser. Elle est aussi envoyée dans l'en-tête `User-Agent` HTTP de chaque requête.

Pour déterminer simplement le type d'un navigateur, il suffit d'employer la propriété `appName`, comme le montre le code précédent. Le Tableau 2.1 contient les valeurs `appName` des navigateurs les plus importants.

Tableau 2.1 : Valeurs `appName` pour divers navigateurs

Navigateur	`appName`
Internet Explorer	`Microsoft Internet Explorer`
Navigateurs Mozilla	`Netscape`
Konqueror (KDE)	`Konqueror`
Apple Safari	`Netscape`
Navigateur Opera	`Opera`

Vous le voyez, le navigateur Safari renvoie un nom incorrect. Pour contrebalancer cet effet, vous pouvez rechercher `navigator.userAgent` pour certains types de navigateurs. Et puisque le navigateur Opera risque d'être mal identifié (bien qu'il stocke `"Opera"` dans `navigator.userAgent`), il convient de le vérifier en premier.

```
<script language="JavaScript"
  type="text/JavaScript">
var uA = navigator.userAgent;
var browserType = "unknown";
if (uA.indexOf("Opera") > -1) {
  browserType = "Opera";
} else if (uA.indexOf("Safari") > -1) {
  browserType = "Safari";
} else if (uA.indexOf("Konqueror") > -1) {
  browserType = "Konqueror";
} else if (uA.indexOf("Gecko") > -1) {
  browserType = "Mozilla";
} else if (uA.indexOf("MSIE") > -1) {
  browserType = "Internet Explorer";
}
window.alert(browserType);
</script>
```

Détermination du type de navigateur (browser.html)

Avec quelques efforts de plus, ce script peut être prolongé pour distinguer les dérivés de Mozilla (Firefox, Epiphany, Galeon, Camino, SeaMonkey, etc.).

Détection du numéro de version du navigateur

Pour déterminer le numéro de version du navigateur, il existe plusieurs manières. La plupart du temps, vous devez rechercher `navigator.userAgent`, ce qui peut ressembler à ceci :

```
Mozilla/5.0 (Windows; U; Windows NT 5.1; en;
     rv:1.8.0.3) Gecko/20060426 Firefox 1.5.0.3
Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
     rv:1.4) Gecko/20030619 Netscape/7.1 (ax)
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
     SV1; .NET CLR 1.0.3705; .NET CLR 1.1.4322;
     .NET CLR 2.0.50727)
Mozilla/5.0 (compatible; Konqueror/3.4; FreeBSD)
     KHTML/3.4.2 (like Gecko)
Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en)
     AppleWebKit/418 (KHTML, like Gecko)
     Safari/417.9.3
Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en)
     AppleWebKit/312.8 (KHTML, like Gecko)
     Safari/312.6
Opera/9.00 (Windows NT 5.1; U; en)
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1 en)
     Opera 9.00
```

Vous le voyez, selon le type du navigateur, le numéro de version est enfoui ailleurs dans la valeur de `navigator.userAgent`. Vous aurez alors la tâche fastidieuse de traiter tous les navigateurs et de vous tenir au courant des nouvelles méthodes. Il existe toutefois quelques ressources Web pour implémenter une détection des navigateurs et qui sont assez bien faites. Vous trouverez de la documentation et des codes sur ces sites Web :

- **http://www.webreference.com/tools/browser/JavaScript.html** ;
- **http://www.gemal.dk/browserspy/basic.html**.

Vérification des capacités du navigateur

```
if (document.getElementById) {
  // ...
```

Vous le voyez dans l'exemple précédent, se fier aux numéros de version des navigateurs n'est pas seulement difficile, c'est également gager les possibilités d'évolution. Il vaut mieux vérifier spécifiquement la prise en charge des objets spéciaux.

Par exemple, pour utiliser DOM (voir Chapitre 5, "DOM et DHTML"), vous pourrez tenter d'utiliser le code précédent. Si la méthode `getElementById()` est implémentée, `document.getElementById` (sans parenthèses) renvoie une référence à la fonction. Si elle est utilisée dans une condition, elle renvoie `true`. Le code associé est alors exécuté.

Autre exemple : Internet Explorer accepte les objets ActiveX pour certaines applications, par exemple la prise en charge du XML. Toutefois, seules les versions Windows de IE connaissent ActiveX. La vérification systématique pour Internet Explorer génère donc des problèmes pour les utilisateurs de Mac. Dès lors, si vous vérifiez spécifiquement la prise en charge d'ActiveX, vous éviterez ces problèmes :

```
if (window.ActiveXObject) {
  // ...
}
```

Empêcher la mise en cache

```
document.write("<img src=\"image.png?" +
  Math.random() + "\" />");
```

Grâce aux en-têtes côté serveur, il est possible d'éviter la mise en cache d'images de type "contenu dynamique" ainsi que de pages HTML. Mais, la méthode n'est pas infaillible, puisque certains navigateurs ou serveurs proxy peuvent ignorer ces paramètres. Pour y remédier, ajoutez un paramètre de chaîne de requête insignifiant à l'URL, comme dans l'extrait suivant : `Math.random()` renvoie un nombre aléatoire compris entre 0 et 1, par exemple `0,1296601696732852`. L'annexer à une image ne modifie généralement pas les données envoyées du serveur, mais la requête est totalement nouvelle pour le navigateur. L'image (ou les autres données) n'est donc pas mise en cache.

Redirection du navigateur

```
location.href = "newPage.html";
```

La propriété `location.href` permet un accès en lecture et en écriture à l'URL de la page en cours. Conséquence, régler `location.href` sur une autre valeur redirige le navigateur, qui charge alors la nouvelle page, comme le montre le code précédent.

La page précédente arrive ensuite dans l'historique du navigateur. Si vous souhaitez remplacer l'ancienne page dans l'historique (pour que le bouton Précédent ne fonctionne pas comme l'on s'y attendrait ici), utilisez la méthode `location.replace()` :

```
location.replace("newPage.html");
```

Astuce

Cela peut également être réalisé avec du HTML :

```
<meta http-equiv="Refresh" content="X; URL=Y" />
```

L'emplacement X désigne le nombre de secondes à patienter avant que la nouvelle page ne soit chargée ; Y indique la nouvelle URL.

Rechargement de la page

```
location.reload();
```

La méthode `reload()` de l'objet `location` recharge la page en cours, l'équivalent de `location.href = location.href`. Si vous apportez le paramètre `true`, la mise en cache est désactivée et le navigateur procède à un rechargement "dur" à partir du serveur. Toutefois, ce n'est pas non plus infaillible puisqu'il peut exister un serveur proxy, placé entre deux, qui peut disposer d'une copie en cache de la page demandée. Vous pourriez donc utiliser la technique décrite dans ce chapitre et qui consiste à empêcher la mise en cache :

```
location.search = "?" + Math.random();
```

L'expression modifie la chaîne de requête (`location.search`) de la page en cours, en rechargeant efficacement le `reload()` de l'URL.

Création d'un nombre aléatoire

```
var rand = min + Math.floor((max - min + 1) *
  Math.random());
```

La méthode `random()` de l'objet `Math` calcule un nombre pseudo-aléatoire entre 0 et 1 (non compris). On préfère généralement un nombre aléatoire compris entre 1 et 10 par exemple. Avec un petit calcul mathématique, l'opération est possible. Pour l'exemple, multipliez le résultat de `Math.random()` par 10, ce qui génère bien un nombre entre 0 et 10 (exclus). Si vous arrondissez ensuite cette valeur, vous obtenez un nombre entier compris entre 0 et 9 (inclus). Ajoutez 1 et vous obtenez un nombre compris entre 1 et 10.

Le code qui précède généralise cette option et crée un nombre compris entre `min` et `max`.

Informations de date et d'heure

```
var d = new Date();
var mdy = (d.getMonth()+1) + "/" + d.getDate() +
```

L'objet `Date` de JavaScript permet d'accéder à la date du jour et peut aussi effectuer certains calculs de date (en utilisant la valeur "epoch", c'est-à-dire le nombre de millisecondes depuis le 1er janvier 1970). Le Tableau 2.2 contient les méthodes les plus importantes de la classe `Date`. Le code qui précède crée une date au format `mois/jour/année`.

Tableau 2.2 : quelques propriétés de `Date`

Méthode	Description
`getDate()`	Jour du mois
`getFullYear()`	Année à quatre chiffres
`getHours()`	Heures
`getMinutes()`	Minutes
`getMonth()`	Mois moins 1 (!)
`getSeconds()`	Secondes
`getTime()`	Valeur "epoch" (date de début de mesure du temps sur un système d'exploitation)
`toString()`	Représentation en chaîne
`toUTCString()`	Représentation en chaîne UTC

Comprendre les expressions régulières

Les expressions régulières sont, pour le dire simplement, des motifs qui peuvent être mis en correspondance avec des chaînes. Un motif dans une expression régulière contient une chaîne sur laquelle peut porter une recherche dans une chaîne plus importante. Mais cela peut aussi être réalisé (plus rapidement) en utilisant `indexOf()`. L'avantage des expressions régulières réside dans le fait que certaines fonctions spéciales, comme les caractères de remplacement, sont disponibles. Le Tableau 2.3 présente quelques caractères spéciaux, accompagnés de leurs significations.

Tableau 2.3 : Caractères spéciaux dans les expressions régulières

Caractère spécial	Description	Exemple
^	Début de la chaîne	`^a` désigne une chaîne qui commence par `a`
$	Fin de la chaîne	`a$` désigne une chaîne qui se termine par `a`
?	0 ou 1 fois (fait référence au caractère ou à l'expression précédent)	`ab?` signifie `a` ou `ab`
*	0 fois ou plus (fait référence au caractère ou à l'expression précédent)	`ab*` signifie `a` ou `ab` ou `abb` ou…
+	1 ou plusieurs fois (fait référence au caractère ou à l'expression précédent)	`ab+` signifie `ab` ou `abb` ou `abbb` ou…

Tableau 2.3 : Caractères spéciaux dans les expressions régulières *(suite)*

Caractère spécial	Description	Exemple
[…]	Caractères alternatifs	`PHP[45]` signifie `PHP4` ou `PHP5`
- (utilisé entre crochets)	Une suite de valeurs	`ECMAScript [3-5]` signifie `ECMAScript 3` ou `ECMAScript 4` ou `ECMAScript 5`
^ (utilisé entre crochets)	Correspond à tout sauf aux caractères suivants	`[^A-C]` désigne `D` ou `E` ou `F` ou…
|	Motifs alternatifs	`ECMAScript 3|ECMAScript 4` désigne `ECMAScript 3` ou `ECMAScript 4`, tout comme `ECMAScript (3|4)`
(...)	Définit un sous-motif	`(a)(b)` signifie `ab`, mais avec deux sous-motifs (`a` et `b`)
.	N'importe quel caractère	`.` signifie `a`, `b`, `c`, `0`, `1`, `2`, `$`, `^`,…
`{min, max}`	Nombre minimal et maximal d'occurrences ; si min ou max est omis, cela signifie 0 ou infinite	`a {1, 3}` signifie `a`, `aa` ou `aaa`. `a{,3}` signifie une chaîne vide, `a`, `aa` ou `aaa`. `a{1,}` signifie `a`, `aa`, `aaa`,…
\	Echappement du caractère suivant	\. désigne .

D'autres caractères spéciaux et expressions sont disponibles, notamment le caractère qui fait référence à un chiffre (`\d`).

Recherche avec des expressions régulières

```
zip.test("Indianapolis, IN 46240");
```

La définition d'une expression régulière dans JavaScript peut s'effectuer de deux manières :

- `var zip = new RegEx("\\d{5}");`
- `var zip = /\d{5}/;`

Il n'y a pas de différence fonctionnelle entre les deux ; vous devez simplement prendre en compte l'échappement du caractère. La méthode `test()` de l'expression vérifie ensuite si une chaîne contient l'expression régulière :

```
var found = zip.test("Indianapolis, IN 46240"); //vrai
```

Pour obtenir la correspondance réelle, utilisez la fonction `exec()`. La méthode renvoie un tableau. Son premier élément est la correspondance entière et les éléments suivants sont toutes les sous-correspondances (si on utilise des parenthèses dans l'expression régulière).

```
var matches = zip.exec("Indianapolis, IN 46240");
// ["46240"]
```

Astuce

La méthode `match()` renvoie toutes les correspondances ; `exec()` ne renvoie que l'expression courante, généralement la première. Toutefois, si vous appelez `exec()` plusieurs fois, toutes les correspondances sont renvoyées.

Remplacement du texte

```
var address = /(\w+), ([A-Z]{2}) (\d{5})/;
var sams = "Indianapolis, IN 46240";
var result = sams.replace(address, "$3 $1, $2");
```

La méthode `replace()`, que prend en charge chaque chaîne JavaScript, peut remplacer du texte. Elle recherche une expression régulière et remplace la chaîne trouvée par une autre chaîne. Dans cette chaîne de remplacement, il est possible de procéder à une référence de retour sur les sous-correspondances. `$0` pointe vers la première correspondance, `$1` référence la première sous-correspondance (sans parenthèses), `$2` la deuxième sous-correspondance, etc. Le code qui précède recherche la ville, l'Etat des Etats-Unis et le code postal, puis les réagence. Le résultat est `"46240 Indianapolis, IN"`.

Navigation dans l'historique d'un navigateur

```
window.history.back();
window.history.forward();
```

L'historique du navigateur est représenté par l'objet `history` (une propriété de l'objet `window`) et contient une liste de pages Web visitées préalablement (et, si possible après) la page en cours. Et même s'il est techniquement possible de déplacer quelques éléments d'historique, les contraintes

de sécurité ne laissent qu'une manière pratique de procéder : reculer d'une page et avancer d'une page. Les deux méthodes suivantes l'implémentent :

- `back()` passe à la page précédente de l'historique (comme le fait le bouton Précédent).
- `forward()` passe à la page suivante de l'historique (comme le fait le bouton Suivant).

Affichage de la date de modification de la page

```
document.write(document.lastModified);
```

Dès qu'un serveur Web envoie une ressource au client, il envoie également la date à laquelle le document a été modifié pour la dernière fois. Généralement, le serveur Web extrait ces informations du système de fichiers, mais cet en-tête peut aussi être modifié ou tout simplement ne pas être envoyé. De toutes les façons, vous pouvez utiliser ces informations, par exemple comme c'est indiqué dans le code précédent. Vous pouvez donc disposer d'une date de modification plus ou moins réaliste sur votre page.

Récupération des paramètres GET

```
var ls = location.search.substring(1);
var namevalue = ls.split("&");
```

Habituellement, les informations GET sont évaluées côté serveur mais JavaScript a également accès à ces informations *via* sa propriété `location.search`. Toutefois, les données qui s'y trouvent résident dans des paires nom-valeur.

Le code suivant décode ces données en utilisant la méthode `split()` de JavaScript. Le tableau associatif qui en résulte est ensuite présenté, simplement pour montrer qu'il fonctionne comme on s'y attend ; reportez-vous à la Figure 2.1 pour connaître le résultat.

```
<script language="JavaScript"
  type="text/JavaScript">
var getdata = [];
if (location.search.length > 1) {
  var ls = location.search.substring(1);
  var namevalue = ls.split("&");
  for (var i=0; i<namevalue.length; i++) {
    var data = namevalue[i].split("=");
    getdata[data[0]] = data[1];
  }
}

var s = "";
for (var el in getdata) {
  s += el + ": " + getdata... + "\n";
}
alert(s);
</script>
```

Analyse de la chaîne de requête (querystring.html)

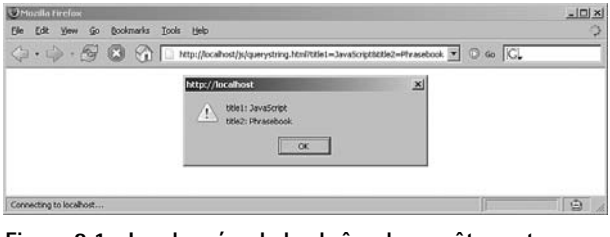

Figure 2.1 : Les données de la chaîne de requête sont analysées et affichées.

Demande de confirmation à l'utilisateur

```
<a href="anyPage.html"
  onclick="return window.confirm('Do you really want to
           do this?');">Click here</a>
```

JavaScript propose une prise en charge limitée des fenêtres modales. La méthode `window.alert()` est assez commune mais il existe d'autres options. Avec `window.confirm()`, l'utilisateur se voit présenter une fenêtre de type Oui/Non. S'il clique sur Oui, `window.confirm()` renvoie `true`, et `false` dans le cas contraire. Le code qui précède (fichier `confirm.html`) l'utilise comme valeur de retour pour un lien. Ainsi, si l'utilisateur clique sur Non, le navigateur ne suit pas le lien.

Attention

Sachez que cette boîte de dialogue est traduite par les navigateurs, vous devez donc éviter d'inscrire un texte du style "Cliquez sur Oui pour..." car les personnes disposant d'un système étranger risquent de ne pas voir s'afficher de bouton Oui.

Demande de données utilisateur

```
var name = window.prompt("Enter your name!", "<Your name>");
```

La méthode `window.prompt()` permet aux utilisateurs de saisir du texte dans un champ de texte d'une seule ligne (voir Figure 2.2). Ces informations correspondent à la valeur de retour de l'appel de méthode et peuvent ensuite être utilisées dans le script.

```
<script language="JavaScript"
  type="text/JavaScript">
var name =
  window.prompt("Enter your name!", "<Your name>");
if (name != null) {
  window.alert("Hello, " + name + "!");
}
</script>
```

Demande de données utilisateur (prompt.html)

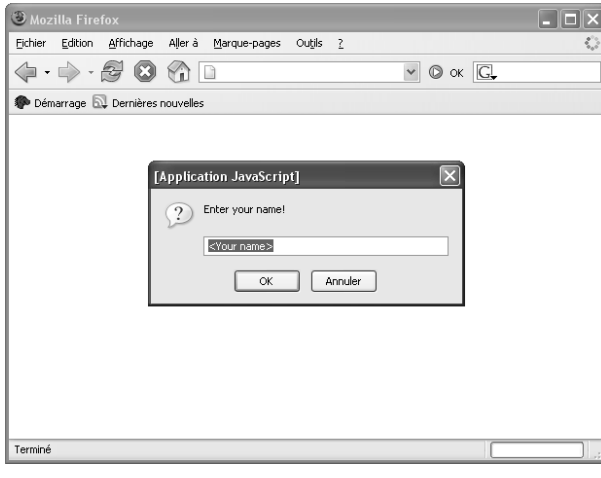

Figure 2.2 : La boîte de saisie générée par window.prompt().

Info

Sachez que si l'utilisateur clique sur le bouton Annuler ou appuie sur la touche d'échappement, `window.prompt()` renvoie `null`. Le code qui précède vérifie que, si l'utilisateur clique sur le bouton OK, les données saisies sont affichées.

3

Images et animations

Aux débuts du Web, l'un des premiers effets obtenus avec JavaScript concernait la manipulation d'images, avec, par exemple, les images sensibles : l'utilisateur déplace le pointeur de la souris sur une image qui change alors d'apparence.

Netscape a introduit l'accès JavaScript pour les images dans son navigateur version 3. A l'époque, Internet Explorer 3 pour Windows en était à sa version bêta, il était donc trop tard pour y intégrer cette fonctionnalité. En revanche, Internet Explorer 3 pour Macintosh, sorti après la version Windows, a pu bénéficier de l'accès aux images JavaScript. Internet Explorer pour Windows a attendu la version 4 pour pouvoir accéder à son tour aux images.

Cela permet d'obtenir non seulement de jolis effets graphiques, mais aussi des applications plus sophistiquées, comme les diaporamas.

Création de boutons sensibles

```
<img src="inactive.gif"
  onmouseover="this.src='active.gif';"
  onmouseover="this.src='inactive.gif';" />
```

L'un des plus anciens effets du World Wide Web consiste à modifier l'apparence d'un graphique lorsque le curseur de la souris passe dessus. Aussi appelée "effet de survol", cette technique s'obtient avec la pseudo-classe CSS `hover` non standard, prise en charge par certains navigateurs.

Pour modifier un graphique, il faut d'abord y accéder, puis modifier sa propriété `src`. A noter que les navigateurs acceptent plusieurs méthodes pour accéder aux graphiques, mais les exemples suivants fonctionnent en toute cohérence dans les navigateurs modernes :

- Définir l'attribut `name` de l'image et y accéder *via* `document.images["name"]`.
- Définir l'attribut `name` de l'image et y accéder *via* `document.images.name` (certains caractères spéciaux comme les espaces ou les tirets ne sont pas autorisés dans l'attribut `name`).
- Accéder à l'image en utilisant sa position dans la page, *via* le tableau `document.images` : `document.images[0]` pour la première image, `document.images[1]` pour la deuxième image, etc.
- Définir l'attribut `id` de l'image et y accéder *via* `document.getElementById("id")`.

L'événement qui se déclenche lorsque le curseur passe sur une image s'intitule `mouseover`. Il faut alors réinitialiser l'image sur son état d'origine. Le code qui précède y parvient et place le tout dans l'élément `<img>` ; vous pourriez

également utiliser une fonction JavaScript personnalisée générique. Le code suivant crée une fonction qui attend le nom de l'image et son URL dans une fonction, puis l'appelle en utilisant `onmouserover/onmouseout` :

```
<script language="JavaScript"
  type="text/JavaScript">
function changeImage(name, url) {
  if (document.images[name]) {
    document.images[name].src = url;
  }
}
</script>
<img
  name="myImage"
  src="inactive.gif"
  onmouseover="changeImage('myImage', 'active.gif');"
  onmouseout="changeImage('myImage', 'inactive.gif');"
/>
```

Un bouton de survol (hover.html)

La Figure 3.1 montre deux boutons : le pointeur de la souris se trouve sur celui de droite.

Attention

Les anciens navigateurs (y compris Netscape 4.x) ne prennent pas en charge `onmouseover/onmouseout` pour des éléments autres que les liens. Dans ce cas, vous devez intégrer l'image dans un lien HTML et utiliser `onmouserover/onmouseout` dans la balise `<a>` :

```
<a
  href="#"
  onmouseover="..." onmouseout="">
  <img src="..." />
</a>
```

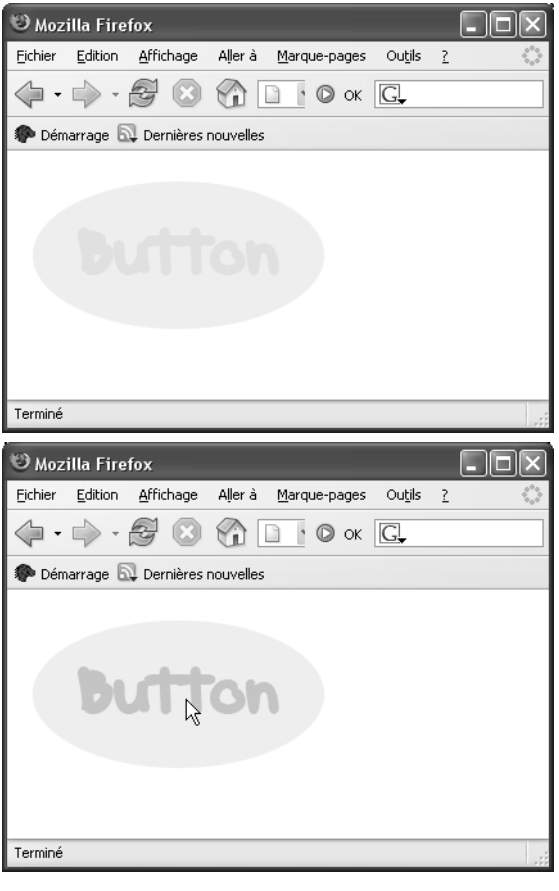

Figure 3.1 : Les boutons changent d'aspect lorsque le pointeur passe dessus.

Préchargement d'images

```
var i = new Image();
i.src = "";
```

Même si l'effet de survol du curseur, mentionné à la section précédente, est facile à mettre en place et joli à regarder, il présente un inconvénient structurel. Lorsque le pointeur passe sur une image, le graphique de remplacement est chargé à partir du serveur, ce qui peut être assez long et gêner les utilisateurs.

Il est donc souhaitable de précharger ces images une fois que la page elle-même a été chargée. Avec JavaScript, vous pouvez demander au navigateur de le faire. Elles sont alors placées dans son cache (à moins que la mise en cache ne soit désactivée dans le navigateur ou que le serveur Web n'envoie des en-têtes HTTP interdisant la mise en cache locale) et chargées instantanément lorsque l'effet de survol est déclenché.

Pour parvenir à cet effet, vous devez instancier l'objet `Image` de JavaScript et régler sa propriété `src`.

L'opération charge l'image en arrière-plan, sans même l'afficher. Il faut alors simplement vérifier que le préchargement est exécuté lorsque la page HTML a été totalement chargée, comme le montre le listing suivant :

```
<script language="JavaScript"
  type="text/JavaScript">
function preloadImage(url) {
  var i = new Image();
  i.src = url;
}
</script>
<body onload="preloadImage('active.gif');">
</body>
```

Préchargement d'images (preload.html)

Si plusieurs images doivent être préchargées simultanément (par exemple si les éléments de navigation emploient un effet de survol), vous pouvez utiliser une fonction plus générique acceptant un tableau avec des images :

```
<script language="JavaScript"
  type="text/JavaScript">
function preloadImages(urls) {
  var img = new Array();
  for (var i=0; i<urls.length; i++) {
    img[img.length] = new Image();
    img[img.length - 1].src = urls[i];
  }
}

window.onload = function() {
  var img = new Array(
    "active.gif", "inactive.gif", "other.gif");
  preloadImages(img);
}
</script>
```

Préchargement de plusieurs images (preload-array.html)

Astuce

Grâce au HTML standard, vous pouvez précharger des images sans avoir à les afficher :

```
<div style="display:none;">
  <img src="active.gif" height="1" width="1" />
  <img src="inactive.gif" height="1" width="1" />
  <img src="other.gif" height="1" width="1" />
</div>
```

Toutefois, lorsque ces images ne servent que pour les effets JavaScript, ne les préchargez qu'en utilisant JavaScript, faute de quoi les utilisateurs sans JavaScript préchargeront également les fichiers, sans obtenir les effets visuels.

Animation des graphiques

```
document.images["animation"].src = urls[pos];
window.setTimeout("animate(" + (pos + 1) + ");", 500);
```

Le format de fichier GIF, initialement produit par Compuserve, est le seul à accepter l'animation et à être accepté par la plupart des grands navigateurs. Il présente cependant des limites importantes, notamment sa palette de couleurs à 8 bits qui n'autorise que 256 couleurs. A noter qu'une norme PNG-24 est en cours de création mais ne possède malheureusement pas de capacités d'animation.

Dans ce cas, le mieux à faire consiste à utiliser JavaScript. La modification d'un graphique est possible comme cela a été montré au cours des sections précédentes, le seul ingrédient manquant est `setInterval()` ou `setTimeout()`. Le code suivant parcourt un tableau d'images et passe à l'image suivante au bout de 500 millisecondes (une demi-seconde) :

```
<script language="JavaScript" type="text/JavaScript">
var urls;

function animate(pos) {
  pos %= urls.length;
  document.images["animation"].src = urls[pos];
  window.setTimeout("animate(" + (pos + 1) + ");", 500);
}

window.onload = function() {
  urls = new Array(
    "0.png", "1.png", "2.png", "3.png",
    "4.png", "5.png", "6.png");
  animate(0);
}
</script>
<img src="0.png" name="animation" />
```

Animation par les images (animate.html)

La fonction `animate()` prend un paramètre : la position de l'image suivante à afficher. La liste des images à parcourir est enregistrée dans une variable globale, puisque `setTimeout()` ne peut pas utiliser les variables locales qui ne sont pas disponibles pour tous.

Avec `pos %= urls.length`, le pointeur ciblant la liste d'images est paramétré au début, après que la dernière image a été affichée, ce qui permet de générer une boucle infinie. Si l'animation doit s'arrêter après la dernière image, remplacez cette ligne par le code suivant :

```
if (pos == urls.length) {
  return false;
}
```

Etirement des graphiques

```
document.images["bar"].width += 5;
```

Il n'est parfois même pas nécessaire de créer plusieurs graphiques pour construire une animation. Ainsi, le simple fait d'étirer une image peut créer un effet saisissant. Ce n'est généralement pas très bien perçu sur le Web, puisque seuls les formats de fichiers bitmap sont pris en charge (à l'exception de Flash ou de SVG, qui n'est pas disponible dans les navigateurs, dans leur configuration par défaut). L'étirement d'un graphique exige donc du navigateur qu'il calcule des informations d'image supplémentaires, généralement en copiant des pixels.

Dans le cas d'une barre de progression, un simple graphique peut suffire. Dans un exemple simple, vous utiliserez une image de 1×1 pixel. S'il est utilisé dans la page HTML avec une largeur de 20 pixels et une hauteur de 10 pixels, vous obtenez une image de 20×10 pixels.

En animant la largeur de l'image avec JavaScript, vous avez l'impression que la barre de progression est en mouvement.

La largeur et la hauteur d'une image sont accessibles par JavaScript grâce aux propriétés `width` et `height`, qui peuvent être lues et écrites. Le code suivant déplace la barre de progression à l'aide d'un simple graphique de 1 × 1 pixel, `black.gif`.

```
<script language="JavaScript"
  type="text/JavaScript">
function progress() {
  if (document.images["bar"].width < 200) {
    document.images["bar"].width += 5;
    document.images["bar"].height = 5;
  } else {
    clearInterval(ID);
  }
 }

var ID;
window.onload = function() {
  ID = setInterval("progress();", 500);
}
</script>
<img src="black.gif" name="bar" />
```

Une barre de progression animée (progress.html)

La Figure 3.2 présente l'exemple en application : l'image 1 × 1 a été étirée dynamiquement par JavaScript. La fonction `progress()` est appelée toutes les 500 millisecondes, grâce à l'appel `setInterval()`. Une fois que la barre de progression a atteint sa longueur maximale (dans cet exemple, environ 200 pixels), l'animation est arrêtée avec `clearInterval()`.

Figure 3.2 : Le graphique est animé pour ressembler à une barre de progression.

Attention

Si vous ne réglez pas explicitement la hauteur de l'image de la barre de progression, le fait d'étirer sa largeur étirera également sa hauteur, ce qui gâche l'effet souhaité.

Visualisation de l'état de chargement de la page avec une barre de progression

```
if (document.images[i].complete) {
  loaded++;
}
```

Savoir si la page a été ou non complètement chargée ne peut pas être déterminé avec du JavaScript uniquement, mais il est possible d'en faire une estimation assez juste.

La propriété `complete` d'une image fournit une valeur booléenne indiquant si l'image s'est totalement chargée. De même, une image accepte l'événement `load`, de sorte que `onload` puisse être utilisé pour déclencher l'exécution du code une fois l'image chargée.

Le code suivant parcourt toutes les images de la page et compte combien d'entre elles ont été totalement chargées. Cela génère une estimation en pourcentage de la part de la page totalement chargée (bien entendu, sans inclure les applets et les médias intégrés, ni les tailles des fichiers image).

```
<script language="JavaScript" type="text/JavaScript">
function showprogress() {
  if (document.images.length == 0) {
    return false;
  }
  var loaded = 0;
  for (var i=0; i<document.images.length; i++) {
    if (document.images[i].complete) {
      loaded++;
    }
  }
  var percentage = Math.round(
    100 * loaded / document.images.length);
  document.getElementById("progress").innerHTML =
    percentage + "% loaded";
  if (percentage == 100) {
    window.clearInterval(ID);
  }
}

var ID = window.setInterval("showprogress();", 500);
</script>
<p name="progress">0% loaded</p>
```

La marque de progression du chargement (loadprogress.html)

Pour réellement reproduire cet effet, vous devez intégrer dans votre page les images qui prennent du temps à charger, soit parce qu'elles sont très volumineuses, soit parce que le serveur subit une charge lourde. Les téléchargements de code pour ce livre comprennent le script PHP `gfx.php` qui repose sur l'extension GD de PHP et crée une image colorée aléatoirement, avec un délai allant jusqu'à quinze secondes. La Figure 3.3 montre le résultat lorsque deux des trois graphiques ont été chargés.

Info

D'autres événements intéressants à signaler pour les images sont `abort` (lorsque le chargement de l'image est interrompu par l'utilisateur) et `error` (lorsqu'il y a une erreur lors du chargement des images, par exemple quand le fichier n'existe pas). Ils peuvent être utilisés avec les gestionnaires d'événements `onabort` et `onerror`.

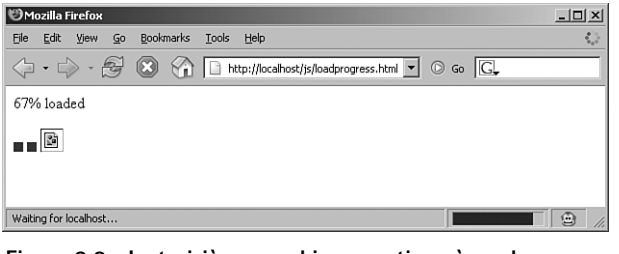

Figure 3.3 : Le troisième graphique continue à se charger.

Utilisation des images réactives

JavaScript peut aussi être utilisé dans des images réactives côté client, mais dans quelques cas seulement :

- Chaque lien dans l'image réactive peut être un lien `JavaScript:`, ce qui permet au navigateur d'appeler JavaScript lorsque l'utilisateur clique sur une zone sensible de l'image.
- Des événements comme `mouseover`, `mouseout` et `load` sont également disponibles dans les images réactives pour les formes définies dans les éléments `<map>`.
- L'image réactive peut être définie *via* JavaScript, en utilisant la propriété `useMap`. Toutefois, il existe généralement une seule image réactive convenant pour une seule image donnée.

4

CSS

Aux premiers temps du World Wide Web, le contenu était roi et la mise en page plutôt délaissée. Le Web a ensuite développé un certain côté commercial, amenant les développeurs à abuser du HTML. Ce langage de marquage s'est alors transformé en un outil de conception inadapté. Grâce aux CSS (feuilles de style en cascade), la situation s'est un peu améliorée : une page HTML a ainsi pu recevoir un style et le HTML redevenir un langage de marquage qui définit le contenu et la structure d'une page, et non plus son aspect.

Grâce à JavaScript, les effets des CSS peuvent être appliqués à la volée. La plupart des sections de ce chapitre sont très génériques, permettant d'appliquer ces techniques à n'importe quel objectif de conception. Certaines résolvent des problèmes plus spécifiques.

Accès aux styles CSS

```
document.getElementById("para").style.fontWeight ="strong";
```

JavaScript peut définir n'importe quelle commande CSS et utilise "presque" comme propriété le nom de la commande CSS. Un problème demeure toutefois : certains caractères, comme le tiret, ne sont pas autorisés dans une propriété JavaScript. Mais de nombreuses commandes CSS (`font-weight` par exemple) contiennent des tirets. Le langage JavaScript utilise alors une syntaxe de type majusculeDeuxièmeMot : à l'exception du premier, chaque mot commence par une lettre majuscule. Ainsi, la commande CSS `font-weight` peut être définie à l'aide de la propriété `fontWeight`.

Toutes les commandes CSS sont accessibles à l'aide de la propriété de style de chaque élément HTML de style de la page. Il existe deux manières communes pour accéder à ces éléments :

- utiliser les gestionnaires d'événements sous la forme d'attributs HTML et envoyer une référence à l'élément actuel comme paramètre : `<p onmouseover="handlerFunction(this);" />` ;
- utiliser `document.getElementById()`.

Le listing suivant montre la seconde méthode. L'élément `<p>` est sélectionné à l'aide de `document.getElementById();`, puis la commande CSS `font-weight` est paramétrée :

```
<script language="JavaScript"
  type="text/JavaScript">
function makeBold() {
  document.getElementById("para").style.fontWeight ="bold";
  window.setTimeout("makeLighter();", 1000);
```

```
}

function makeLighter() {
  document.getElementById("para").style.fontWeight =
    ➥"lighter";
  window.setTimeout("makeBold();", 1000);
}

window.onload = makeBold;
</script>
<p id="para">CSS and JavaScript</p>
```

Modification d'une commande CSS (style.html)

Astuce

La console JavaScript, avec le navigateur Mozilla, affiche une erreur lorsqu'une valeur inadéquate est appliquée au style choisi, comme le montre la Figure 4.1. Cela est extrêmement commode pour le débogage.

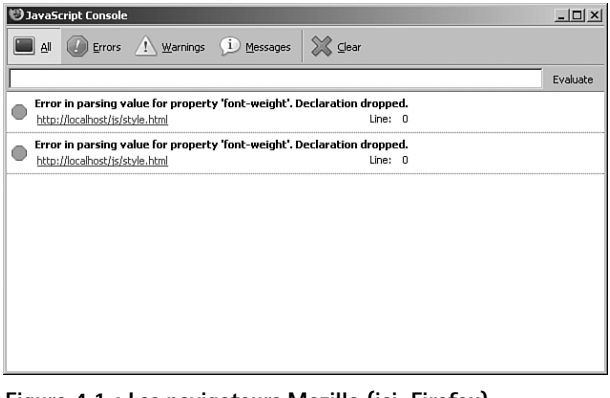

Figure 4.1 : Les navigateurs Mozilla (ici, Firefox) refusent les valeurs CSS inadéquates.

Accès aux classes CSS

```
document.getElementById("para").className = "strong";
```

La manière la plus usitée d'appliquer du CSS à une page HTML consiste à utiliser les classes. Avec JavaScript, la classe de chaque élément est accessible avec la propriété `className`. Le code suivant implémente l'expression précédente avec la méthode par classe :

```
<script language="JavaScript"
  type="text/JavaScript">
function makeBold() {
  document.getElementById("para").className ="strong";
  window.setTimeout("makeLighter();", 1000);
}

function makeLighter() {
  document.getElementById("para").className ="weak";
  window.setTimeout("makeBold();", 1000);
}

window.onload = makeBold;
</script>
<style type="text/css">
  .strong { font-weight: bold; }
  .weak { font-weight: lighter; }
</style>
<p id="para">CSS and JavaScript</p>
```

Modification de la classe CSS (classname.html)

Le code qui précède modifie la classe pour le texte à chaque seconde.

Accès aux feuilles de style individuelles

```
document.styleSheets[0].disabled = true;
```

Lorsqu'une page se compose de plusieurs feuilles de style (élément `<style>` avec un style intégré ou une feuille de style externe), vous pouvez utiliser JavaScript pour basculer entre elles, les activer et les désactiver, ainsi que les combiner.

La propriété `styleSheets` de l'objet `document` contient toutes les feuilles de style de la page, dans l'ordre dans lequel elles sont chargées ou dans lequel elles apparaissent sur cette page.

La propriété la plus importante de chaque feuille de style s'intitule `disabled`. Si elle est réglée sur `true`, elle devient invisible et n'affecte plus la mise en page.

Vous pouvez accéder à une feuille de style spécifique de deux façons :

- en utilisant l'identifiant de la feuille de style comme indice pour le tableau `styleSheets` ;
- en utilisant le numéro de la feuille de style (qui commence par 0) comme indice pour le tableau `styleSheets`.

La première manière ne fonctionne pas correctement avec les navigateurs Mozilla, utilisez donc l'indice numérique, comme le montre le code suivant :

```
<script language="JavaScript"
  type="text/JavaScript">
function makeBold() {
  document.styleSheets[0].disabled = false;
```

```
  document.styleSheets[1].disabled = true;
  window.setTimeout("makeLighter();", 1000);
}

function makeLighter() {
  document.styleSheets[0].disabled = true;
  document.styleSheets[1].disabled = false;
  window.setTimeout("makeBold();", 1000);
}

window.onload = makeBold;
</script>
<style type="text/css" id="strong">
  p { font-weight: bold; }
</style>
<style type="text/css" id="weak">
  p { font-weight: lighter; }
</style>
<p>CSS and JavaScript</p>
```

Modification de la feuille de style (stylesheets.html)

Accès aux règles de CSS individuelles

```
document.styleSheets[0].rules[0].style.color =
  randomColor();
document.styleSheets[0].cssRules[0].style.color =
  randomColor();
```

Les règles individuelles d'une feuille de style sont également accessibles par la voie de la programmation, mais les navigateurs Web diffèrent les uns des autres. Internet Explorer accepte la propriété `rules`, tandis que tous les autres navigateurs utilisent la propriété `cssRules`. La seule exception revient au navigateur Opera, qui n'accepte ni l'une ni l'autre.

Sachez que vous pouvez accéder aux règles puis, par exemple, les modifier. Chaque règle se comporte comme un élément HTML générique : vous utilisez la propriété `style` pour accéder à tous les styles, puis pour les modifier ou en ajouter.

Pour l'exemple suivant, une fonction d'aide génère une couleur aléatoire au format RVB :

```
function randomColor() {
  var chars = "0123456789abcdef";
  var color = "#";
  for (var i=0; i<6; i++) {
    var rnd = Math.floor(16 * Math.random());
    color += chars.charAt(rnd);
  }
  return color;
}
```

L'expression précédente est ensuite étendue. Tout d'abord, chaque feuille de style contient les règles pour les éléments `<p>` et `<span>`. Ces deux règles reçoivent ensuite une commande CSS supplémentaire : la couleur est réglée sur une valeur aléatoire. L'utilité de cet exemple simple peut ne pas paraître évidente dans un cas concret, mais il est possible d'adapter la technologie sous-jacente de manière assez flexible.

```
<script language="JavaScript"
  type="text/JavaScript">
function makeBold() {
  document.styleSheets[0].disabled = false;
  document.styleSheets[1].disabled = true;
  if (document.styleSheets[0].rules) {
    document.styleSheets[0].rules[0].style.color =
    ➥randomColor();
    document.styleSheets[0].rules[1].style.color =
    ➥randomColor();
```

```
  } else if (document.styleSheets[0].cssRules) {
    document.styleSheets[0].cssRules[0].style.color =
     ➥randomColor();
    document.styleSheets[0].cssRules[1].style.color =
     ➥randomColor();
  }
  window.setTimeout("makeLighter();", 1000);
}

function makeLighter() {
  document.styleSheets[0].disabled = true;
  document.styleSheets[1].disabled = false;
  if (document.styleSheets[0].rules) {
    document.styleSheets[1].rules[0].style.color =
     ➥randomColor();
    document.styleSheets[1].rules[1].style.color =
     ➥randomColor();
  } else if (document.styleSheets[0].cssRules) {
    document.styleSheets[1].cssRules[0].style.color =
     ➥randomColor();
    document.styleSheets[1].cssRules[1].style.color =
     ➥randomColor();
  }
  window.setTimeout("makeBold();", 1000);
}

window.onload = makeBold;
</script>
<style type="text/css" id="strong">
  p { font-weight: bold; }
  span { font-style: italic; }
</style>
<style type="text/css" id="weak">
  p { font-weight: lighter; }
  span { font-style: normal; }
</style>
<p>CSS <span>and</span> JavaScript</p>
```

Accès aux règles CSS (rules.html)

La Figure 4.2 montre le résultat : même si l'ouvrage est imprimé en gris, vous constaterez différentes couleurs (ou plutôt niveaux de gris) dans le paragraphe et dans l'élément qui s'y trouve.

Figure 4.2 : Les couleurs du texte sont appliquées de manière aléatoire aux règles CSS individuelles.

Laisser disparaître le contenu d'un site Web

```
document.getElementById(show).style.display = "block";
document.getElementById(hide).style.display = "none";
```

Un effet JavaScript assez commun sur le Web, et qui utilise les CSS, consiste à laisser les éléments d'une page apparaître ou disparaître à la demande. Certaines pages affichent un écran d'attente sur une page qui se charge lentement. Une fois la page chargée, l'écran d'attente disparaît. Un autre exemple consiste à fournir plusieurs onglets, un seul affiché à la fois, et à passer de l'un à l'autre en utilisant JavaScript, sans qu'il soit besoin d'adresser au serveur de longues requêtes.

Il existe plusieurs manières de procéder avec JavaScript. Vous pouvez utiliser DOM (vous trouverez certaines expressions pour faire apparaître ou disparaître des éléments DOM au Chapitre 5, "DOM et DHTML"). Deux possibilités sont présentées dans l'expression.

On trouve d'abord la commande CSS `visibility`, que vous pouvez régler sur `visible` ou `hidden` pour faire apparaître ou disparaître l'élément associé, ce qui se traduit en une propriété JavaScript `visibility`. En voici le code :

```
<script language="JavaScript"
  type="text/JavaScript">
  function showHide(show, hide) {
    document.getElementById(show).style.visibility =
    ➥"visible";
    document.getElementById(hide).style.visibility =
    ➥"hidden";
  }
</script>
<p> <br /> </p>
<p id="tab1"
  style="position: absolute; top: 5px; left: 5px;">
  Tab 1
</p>
<p id="tab2" style="position: absolute; top: 5px;
     left: 5px; visibility: hidden;">
  Tab 2
</p>
<input type="button" value="Tab 1"
  onclick="showHide('tab1', 'tab2');" />
<input type="button" value="Tab 2"
  onclick="showHide('tab2', 'tab1');" />
```

Réglage de la visibilité d'un élément (visibility.html)

Toutefois, comme vous le verrez, le positionnement absolu entraîne des différences subtiles selon les navigateurs.

Il vaut mieux utiliser une mise en page par blocs et définir la propriété `display` de l'élément sur `block` ou `none` :

```
<script language="JavaScript"
  type="text/JavaScript">
  function showHide(show, hide) {
    document.getElementById(show).style.display = "block";
    document.getElementById(hide).style.display = "none";
  }
</script>
<p id="tab1">
  Tab 1
</p>
<p id="tab2" style="display: none;">
  Tab 2
</p>
<input type="button" value="Tab 1"
  onclick="showHide('tab1', 'tab2');" />
<input type="button" value="Tab 2"
  onclick="showHide('tab2', 'tab1');" />
```

Réglage du mode d'affichage d'un élément (display.html)

La Figure 4.3 montre le résultat : cliquer sur le second bouton fait apparaître le second onglet.

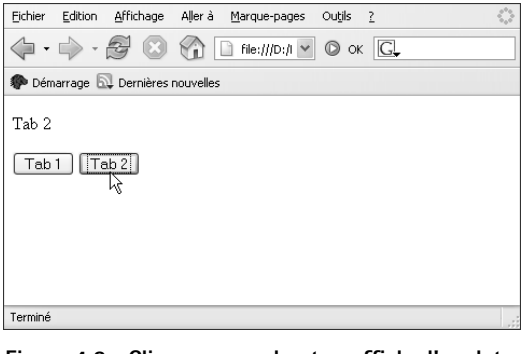

Figure 4.3 : Cliquer sur un bouton affiche l'onglet associé.

Application de JavaScript aux sélecteurs CSS

```
Behaviour.register(cssrules);
```

En 1999, le W3C proposait des "extensions comportementales aux CSS" (**http://w3.org/TR/1999/WD-becss-19990804**), une méthode consistant à lier du code JavaScript à des comportements CSS. L'idée de base était d'éviter l'utilisation des attributs HTML bien connus de type `on`*`xxx`*. Après 1999, ce domaine n'avait pas beaucoup progressé, mais Ben Nolan réussit à proposer un contournement (ou plutôt un piratage) pour permettre un concept analogue. Pour cela, il faut employer la bibliothèque JavaScript (**http://www.bennolan.com/behaviour/**) sous la forme du fichier `behaviour.js`. Cette collection de code permet de fournir des règles contenant du code JavaScript sous la forme de sélecteurs CSS.

Pour que cela fonctionne, il faut d'abord un fichier JavaScript externe qui définisse les règles CSS. Pour l'essentiel, vous créez un objet avec des sélecteurs CSS sous forme de clés, puis les fonctions de gestion d'événements JavaScript sous forme de valeurs. Ensuite, vous appelez la méthode `Behaviour.register()` pour signaler ces règles à la bibliothèque de comportements :

```
var cssrules = {
  "p.para" : function(e){
     e.onmouseover = function(){
       this.style.fontWeight = "bold";
     }
     e.onmouseout = function(){
       this.style.fontWeight = "normal";
     }
```

```
  },
  "#term" : function(e){
      e.onmouseover = function(){
        this.style.fontStyle = "oblique";
      },
      e.onmouseout = function(){
        this.style.fontStyle = "normal";
      }
    }
  };

Behaviour.register(cssrules);
```

Règles CSS/JavaScript (rules.js)

Il reste à charger la bibliothèque JavaScript et le précédent fichier JavaScript de règles. Le code JavaScript est ensuite exécuté lorsque les événements associés (dans l'exemple : `mouseover` et `mouseout`) sont déclenchés.

```
<script language="JavaScript"
  type="text/JavaScript" src="behaviour.js">
</script>
<script language="JavaScript"
  type="text/JavaScript" src="rules.js">
</script>
<p class="para">CSS <span id="term">and</span>
  JavaScript</p>
```

Utilisation de la bibliothèque de comportements JavaScript (behaviour.html)

La Figure 4.4 montre le résultat : lorsque le pointeur de la souris passe sur l'élément `<span>`, deux gestionnaires d'événements sont appelés.

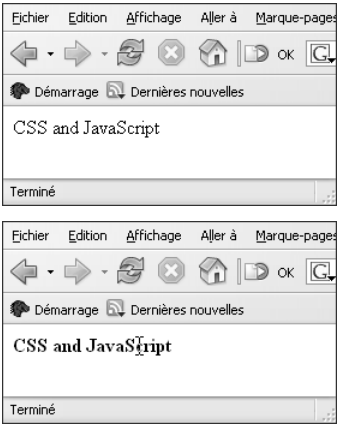

Figure 4.4 : Le code JavaScript est exécuté.

Info

Le principal avantage de cette méthode réside dans le fait que le code JavaScript se trouve dans des fichiers externes et qu'il n'encombre pas le marquage HTML de la page. Du point de vue de l'architecture, l'idée est assez brillante et fonctionne dans tous les navigateurs.

Modification du curseur de la souris

```
document.getElementById("button").style.cursor = "help";
```

Tous les systèmes d'exploitation sont livrés avec un certain jeu de curseurs de souris et CSS en accepte un sous-ensemble. L'apparence est un peu différente selon les systèmes d'exploitation, mais la mise en page générale est assez similaire.

La commande CSS en question est `cursor`, et elle est également disponible à partir de JavaScript, comme le montre le code suivant :

```
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  document.getElementById("helpButton").style.cursor
     ➥= "help";
};
</script>
<input type="submit" id="helpButton" value="Get help" />
```

Modification du curseur (cursor.html)

Vous voyez le résultat à la Figure 4.5 : le curseur de la souris comprend maintenant un point d'interrogation pour préciser que le bouton permet d'obtenir de l'aide.

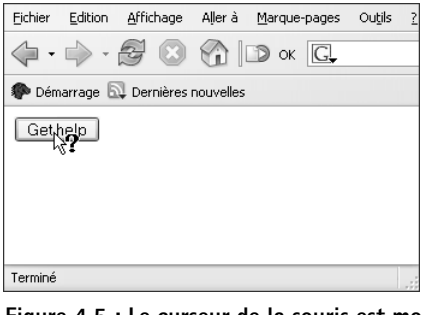

Figure 4.5 : Le curseur de la souris est modifié.

Attention

Chaque curseur de souris réalise un objectif. Réfléchissez donc bien avant de modifier l'apparence d'un curseur, vous ne devez pas irriter vos utilisateurs !

Styles de curseur CSS

Voici les valeurs autorisées pour la propriété `cursor` de JavaScript/CSS :

- `auto`
- `crosshair`
- `default`
- `e-resize`
- `help`
- `move`
- `n-resize`
- `ne-resize`
- `nw-resize`
- `pointer`
- `s-resize`
- `se-resize`
- `sw-resize`
- `test`
- `w-resize`
- `wait`

5

DOM et DHTML

Ce chapitre traite de deux techniques JavaScript vaguement liées : DOM et DHTML. Ces deux aspects de JavaScript existent depuis de nombreuses années, mais ce n'est que depuis la chute des parts de marché de Netscape 4.x, passant en dessous de celle de Mozilla il y a quelques années, qu'il a été possible de les utiliser conjointement. A l'origine, pourtant, ils n'avaient pas grand-chose en commun. Ce chapitre traite de DOM, puisque de nombreuses expressions de cet ouvrage peuvent être plus ou moins classées sous la catégorie "DHTML".

DOM

DOM (*Document Object Model*, modèle objet de document) est assez explicite : il propose un modèle d'objet (et, avec lui, une API, une interface de programmation d'applications) pour un document. Dans le contexte du Web, on fait évidemment référence à un document HTML.

L'une des meilleures manières de le visualiser consiste à utiliser l'inspecteur DOM livré avec les navigateurs Mozilla. Dans le DOM, tous les éléments de page sont placés dans une hiérarchie arborescente. Chaque balise HTML correspond à un nœud de l'arborescence, avec des nœuds inférieurs et des nœuds parent. De même, chaque partie de texte correspond à son propre nœud DOM (un nœud de texte, pour être exact). La Figure 5.1 montre l'inspecteur DOM pour un simple échantillon de document.

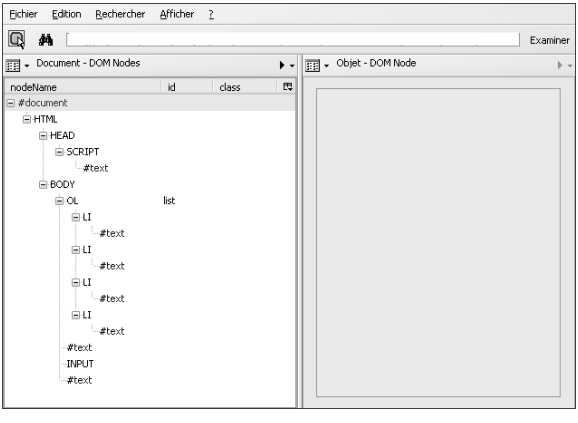

Figure 5.1 : L'inspecteur DOM montre tous les nœuds d'un document.

Bien entendu, l'API DOM accepte les méthodes pour accéder aux éléments de l'arborescence DOM, mais également pour ajouter ou supprimer des éléments. Il est donc possible de modifier quasiment n'importe quel élément d'une page.

Une question demeure : que vaut-il mieux faire, utiliser DOM ou des objets JavaScript standard comme `forms`

ou `images` ? Généralement, DOM est plus flexible mais aussi parfois plus difficile. Ainsi, si vous pouvez utiliser un raccourci JavaScript, faites-le. Sinon, familiarisez-vous avec DOM.

Info

Vous trouverez de plus amples informations sur DOM à l'adresse du consortium W3C (World Wide Web Consortium) : http://w3c.org/DOM.

DHTML

DOM est un "vrai " terme technique, alors que DHTML (le HTML dynamique) ne l'est pas. C'est un terme de marketing pur. Lorsqu'il a été créé, il signifiait en fait "JavaScript pour les navigateurs de version 4 et supérieure". A cette époque, Netscape Navigator et Internet Explorer se battaient dans la célèbre guerre des navigateurs et, avec leur version 4, les deux éditeurs ont inclus des fonctions JavaScript plus dynamiques, malheureusement assez incompatibles les unes avec les autres. La plupart des effets DHTML couvrent le positionnement des éléments et leur modification en cas d'action de l'utilisateur.

Netscape 4 et Internet Explorer 4 ayant disparu depuis longtemps, les versions récentes des deux marques (n'oubliez pas que Netscape 4 a été suivi par Netscape 6, qui se fonde sur le moteur Gecko de Mozilla) acceptent raisonnablement bien DOM (même s'il manque encore certains éléments). DHTML est donc rené de ses cendres, puisque les effets du JavaScript dynamique sont désormais possibles avec une petite quantité de code spécifique au navigateur et de recherche du client.

Accès à des éléments spécifiques

```
document.getElementById("para")
```

Lorsque vous travaillez avec DOM, la meilleure manière d'accéder par la suite à un élément sur la page consiste à lui donner un identifiant unique, ou ID. La méthode DOM `document.getElementById()` accède ensuite à l'élément donné et vous permet de poursuivre : modifier les éléments, ajouter des sous-éléments ou encore naviguer dans l'arborescence DOM.

Dans l'exemple suivant, on accède à l'élément `<p>`. Selon le type du navigateur, la représentation en chaîne de la classe d'élément est différente. Tandis qu'Internet Explorer produit simplement `[object]`, les navigateurs Mozilla sont plus verbeux et donnent plus d'informations : `[object HTMLParagraphElement]`.

```
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  window.alert(document.getElementById("para"));
}
</script>
<p id="para">JavaScript Phrasebook</p>
```

Accès à un élément par son identifiant (getelementbyid.html)

Attention

DOM n'est accessible que lorsque l'ensemble du document a été chargé. C'est la raison pour laquelle son code d'accès n'est exécuté qu'après que l'événement `load` a été déclenché.

Accès aux balises

document.getElementsByTagName("p")

Une autre manière d'accéder aux éléments de la page en cours consiste à passer par leurs noms de balise. Dès que vous devez travailler sur un jeu d'éléments représenté par des balises HTML identiques (par exemple, tous les éléments de liste ou tous les paragraphes, tout ce qui n'est pas représenté par une autre propriété de l'objet `document` JavaScript), la méthode `document.getElementsByTagName()` peut être utilisée. Vous apportez le nom de la balise et vous obtenez un tableau de tous les éléments, que vous pouvez ensuite traiter.

Le code suivant accède simplement à tous les éléments `<p>` et les compte :

```
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  window.alert(
    document.getElementsByTagName("p") +
    ➥" (" + document.getElementsByTagName("p").length
    ➥+ " elements)");
}
</script>
<p>JavaScript Phrasebook</p>
<p>Sams Publishing</p>
```

Accès aux éléments par nom de balise (getelementsbytagname.html)

La sortie du code précédent est `[object HTMLCollection] (2 elements)` ; Internet Explorer une fois de plus donne moins d'informations, en produisant seulement `[object] (2 elements)`.

Navigation dans l'arborescence DOM

Une fois à l'intérieur de DOM, vous pouvez naviguer dans la structure, monter et descendre ou aller à droite et à gauche. Voici une liste des propriétés les plus importantes que possède chaque nœud DOM :

- `firstChild` : premier nœud enfant ;
- `lastChild` : dernier nœud enfant ;
- `childNodes` : tous les nœuds enfant (sous forme de tableau) ;
- `parentNode` : nœud parent ;
- `nextSibling` : nœud suivant au même niveau (à droite) ;
- `previousSibling` : nœud précédent au même niveau (à gauche).

De même, `nodeName` renvoie le nom de la balise du nœud (ou `#text` pour les nœuds de texte) tandis que `nodeValue` renvoie la valeur d'un nœud (fonction utile avec les nœuds de texte).

Détermination des informations de nœud

```
s += nodeName + ": " + nodeValue + " (" + nodeType + ")\n";
```

Nous l'avons indiqué à la section précédente, il est possible de rassembler des informations sur un nœud, ce qui est extrêmement utile lorsqu'on travaille avec des données DOM arbitraires.

`nodeName` donne des informations sur le nom du nœud (nom de balise ou `#text` pour les nœuds de texte), mais `nodeValue` ne sert que pour les nœuds de texte et renvoie le texte réel du nœud. La troisième catégorie d'informations provient de la propriété `nodeType`, qui donne des informations sur le type du nœud. Le Tableau 5.1 contient une liste de toutes les valeurs possibles de `nodeType`.

Tableau 5.1 : Types de nœuds

Type de nœud	Description
1	Balise
2	Attribut
3	Texte (espaces compris)
8	Commentaire HTML
9	Document
10	DTD
11	Fragment

Le code suivant analyse ensuite une simple structure DOM et produit des informations concernant tous les nœuds enfant. La Figure 5.2 présente le résultat dans le navigateur.

```
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  var s = "";
  with (document.getElementById("para")) {
    for (var i=0; i<childNodes.length; i++) {
      with (childNodes[i]) {
        s += nodeName + ": " + nodeValue +
             " (" + nodeType + ")\n";
      }
    }
  }
  window.alert(s);
}
</script>
<p id="para"><em>JavaScript</em> Phrasebook</p>
```

Récupération des informations de nœud (nodeinfo.html)

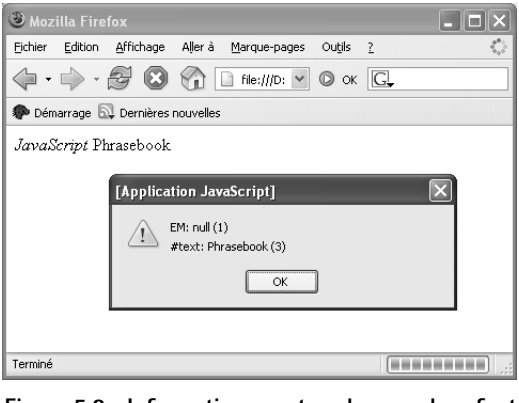

Figure 5.2 : Informations sur tous les nœuds enfant.

Suppression d'éléments

```
list.removeChild(list.lastChild);
```

La méthode `removeChild()` que possède chaque nœud peut servir à éliminer un nœud de l'arborescence DOM. Sachez que vous devez appeler cette méthode à partir de l'élément parent du nœud à effacer et fournir le nœud en paramètre. L'exemple suivant présente une liste et supprime le dernier élément chaque fois que l'utilisateur clique sur le bouton ; reportez-vous à la Figure 5.3 pour voir le résultat dans le navigateur.

```
<script language="JavaScript"
  type="text/JavaScript">
function removeItem() {
  var list = document.getElementById("list");
  if (list.childNodes.length > 0) {
    list.removeChild(list.lastChild);
  }
}
</script>
<ol id="list">
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
  <li>Item</li>
</ol>
<input type="button" onclick="removeItem();"
  value="Remove item" />
```

Suppression de nœuds (remove.html)

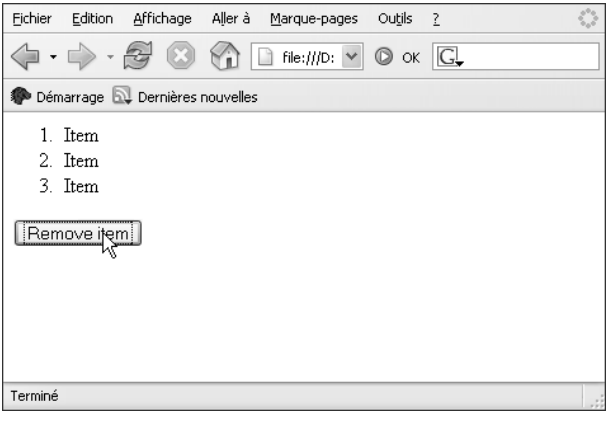

Figure 5.3 : Le dernier nœud est supprimé.

Astuce

Lorsque vous n'avez qu'un accès direct au nœud à supprimer (`curNode` dans le code suivant), cette méthode fonctionne :

```
curNode.parentNode.removeChild(curNode);
```

Si vous appelez ce code dans une fonction gestionnaire pour le nœud, vous pouvez remplacer `curNode` par `this`.

Ajout d'éléments

```
list.appendChild(newNode);
```

Les nouveaux nœuds DOM sont créés à l'aide de la méthode `document.createElement()`. Elle génère un nouvel élément, en utilisant le nom de balise fourni en paramètre.

Cet élément peut ensuite être inséré dans l'arborescence DOM. La méthode (`node`) utilisée le plus souvent est `appendChild()`. Elle ajoute un nouvel enfant à la fin de la liste.

Le code suivant ajoute un nouvel élément de liste vide (`<li>`) à la fin de la liste dès que l'utilisateur clique sur le bouton :

```
<script language="JavaScript"
  type="text/JavaScript">
function addItem() {
  var list = document.getElementById("list");
  var newNode = document.createElement("li");
  list.appendChild(newNode);
}
</script>
<ul id="list"><li>Item</li></ul>
<input type="button" onclick="addItem();"
  value="Add item" />
```

Ajout de nœuds (add.html)

L'inconvénient de cette approche réside dans le fait que le nouvel élément est toujours ajouté à la fin de la liste des enfants. Pour y remédier, utilisez la méthode `insertBefore()` qui permet d'insérer un nouveau nœud *avant* tout autre nœud (pour l'insérer n'importe où, sauf à la fin de la liste). Comme pour les paramètres, vous donnez d'abord le nouveau nœud, puis le nouvel élément de même niveau.

Le code suivant insère un nouvel élément `<li>` au *début* de la liste chaque fois que l'utilisateur clique sur le bouton (voir Figure 5.4).

```
<script language="JavaScript"
  type="text/JavaScript">
function addItem() {
  var list = document.getElementById("list");
  var newNode = document.createElement("li");
  list.insertBefore(newNode, list.firstChild);
}
</script>
<ul id="list"><li>Item</li></ul>
<input type="button" onclick=vaddItem();"
  value="Add item" />
```

Ajout de nœuds au début de la liste des enfants (addbefore.html)

Figure 5.4 : Les éléments de liste sont ajoutés au début de la liste.

Création d'éléments de texte

```
var newTextNode = document.createTextNode("Item " + nr);
```

Pour placer du texte dans un élément, il faut un nœud de texte qui soit un sous-élément du nœud réel. La méthode `createTextNode()` crée ce nœud de texte ; il vous suffit de fournir le texte lui-même.

Dans le code suivant, les éléments de liste sont à nouveau ajoutés mais, cette fois-ci, ils contiennent du texte. Vous créez donc d'abord un nœud de texte, puis vous l'ajoutez à un autre nœud (qui peut ensuite être ajouté à un autre nœud, qui peut ensuite être ajouté à un autre nœud… Vous avez compris l'idée).

```
<script language="JavaScript"
  type="text/JavaScript">
var nr = 1;
function addItem() {
  var list = document.getElementById("list");
  var newNode = document.createElement("li");
  nr++;
  var newTextNode =
    document.createTextNode("Item " + nr);
  newNode.appendChild(newTextNode);
  list.appendChild(newNode);
}
</script>
<ul id="list"><li>Item 1</li></ul>
<input type="button" onclick="addItem();"
  value="Add item" />
```

Création (et ajout) d'un nœud de texte (textnode.html)

Travail avec les attributs

```
newLink.setAttribute("href",
    "http://www.samspublishing.com/");
```

Jusqu'à présent, ce chapitre s'est intéressé aux balises et aux nœuds de texte ordinaires. Il manque toujours un élément majeur, les attributs. Vous pouvez y accéder sous la forme de nœuds mais la manière la plus commode est d'utiliser la méthode `setAttribute()` : il vous suffit de fournir le nom de l'attribut et sa valeur. Dans le code suivant, le contenu des nouveaux éléments `<li>` est un lien hypertexte, avec un jeu d'attributs `href` :

```
<script language="JavaScript"
  type="text/JavaScript">
var nr = 1;
function addItem() {
  var list = document.getElementById("list");
  var newNode = document.createElement("li");
  var newLink = document.createElement("a");
  newLink.setAttribute("href",
  ➥"http://www.samspublishing.com/");
  nr++;
  var newTextNode =
    document.createTextNode("Item " + nr);
  newLink.appendChild(newTextNode);
  newNode.appendChild(newLink);
  list.appendChild(newNode);
}
</script>
<ul id="list"><li>Item 1</li></ul>
<input type="button" onclick="addItem();"
  value="Add item" />
```

Réglage des attributs (attributes.html)

La Figure 5.5 montre que tous les liens ajoutés à la liste pointent vers l'URL fournie.

Figure 5.5 : Liens hypertexte générés de manière dynamique.

Clonage d'éléments

```
var newItem = oldItem.cloneNode(true);
```

Au lieu de créer de nouveaux nœuds chaque fois, vous pouvez également cloner un nœud existant. La méthode `cloneNode()`, accessible pour chaque nœud, le fait pour vous. Vous pouvez décider de ne cloner que le nœud et son attribut ou de cloner tous les nœuds enfant (et leurs nœuds enfant, etc.). Si vous fournissez `true` en paramètre à `cloneNode()`, une "copie en profondeur" est effectuée, laquelle prend en compte également les enfants ; `false` ne copie que le nœud lui-même.

```
<script language="JavaScript" type="text/JavaScript">
var nr = 1;
function addItem(cloneMode) {
  var list = document.getElementById("list");
  var oldItem = list.firstChild;
  var newItem = oldItem.cloneNode(cloneMode);
  list.appendChild(newItem);
}
</script>
<ul id="list"><li><a href="http://www.samspublishing.com/">
➥Item 1</a></li></ul>
<input type="button" onclick="addItem(true);"
  value="Clone all" />
<input type="button" onclick="addItem(false);"
  value="Clone node only" />
```

Clonage de nœuds (clone.html)

Lorsque vous cliquez sur le premier bouton, l'ensemble du nœud (y compris les sous-éléments, comme le lien et le texte de l'élément du lien) est copié ; le second bouton ne clone que le nœud, ce qui génère un nouvel élément de liste vide. Etudiez la Figure 5.6 pour voir la différence.

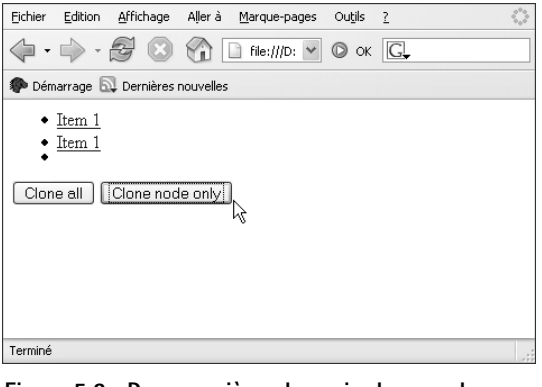

Figure 5.6 : Deux manières de copier les nœuds.

Remplacement d'éléments

```
list.replaceChild(newNode, list.firstChild);
```

Si vous supprimez un nœud et que vous en insériez un autre à la même place, la méthode `replaceChild()` vous évite d'insérer une partie du codage. Vous indiquez le nouveau et l'ancien nœud et JavaScript fait le reste pour vous. N'oubliez pas que vous devez appeler cette méthode à partir de l'élément parent de l'ancien et du nouveau nœud !

```
<script language="JavaScript"
  type="text/JavaScript">
var nr = 1;
function addItem() {
  var list = document.getElementById("list");
  var newNode = document.createElement("li");
  nr++;
  var newTextNode =
    document.createTextNode("Item " + nr);
  newNode.appendChild(newTextNode);
  list.replaceChild(newNode, list.firstChild);
}
</script>
<ul id="list"><li>Item 1</li></ul>
<input type="button" onclick="addItem();"
  value="Replace item" />
```

Remplacement des nœuds (replace.html)

Le code qui précède remplace le premier enfant (nœud) de la liste par un nouveau nœud.

Création d'une liste à puces à partir de données JavaScript

```
var newItem = document.createElement("li");
var newText = document.createTextNode(data[i]);
newItem.appendChild(newText);
list.appendChild(newItem);
```

Notamment dans le contexte des services Web et AJAX (voir Chapitres 10 et 11), il arrive souvent qu'on doive afficher de manière dynamique les données reçues du serveur. Une bonne technique consiste à utiliser une liste HTML. L'exemple de code suivant propose une fonction `createList()` qui attend un tableau, contenant des valeurs pour le transformer en liste.

```
<script language="JavaScript"
  type="text/JavaScript">
function createList(data) {
  var list = document.createElement("ul");
  for (var i = 0; i < data.length; i++) {
    var newItem = document.createElement("li");
    var newText = document.createTextNode(data[i]);
    newItem.appendChild(newText);
    list.appendChild(newItem);
  }
  return list;
}

window.onload = function() {
  var list = createList(
    ["one", "two", "three", "four", "five"]);
  document.body.appendChild(list);
}
</script>
```

Création d'une liste (list.html)

Notez que `document.body` est un raccourci de l'élément `<body>` (autrement, vous pourriez utiliser `document.getElementByTagName("body")[0]`) ; `appendChild()` ajoute ensuite la liste HTML à la fin de la page.

Création d'un tableau à partir de données JavaScript

```
var td = document.createElement("td");
var newText = document.createTextNode(data[i][j]);
td.appendChild(newText);
tr.appendChild(td);
```

Le tableau se situe à un niveau de complexité supérieur à celui de la liste. Vous devez d'abord utiliser l'élément `<tbody>` (et peut-être aussi `<tbody>` et/ou `<tfoot>`). Vous risqueriez sinon de ne rien voir dans Internet Explorer.

La fonction d'aide `createTable()` attend un tableau à plusieurs dimensions. Chaque élément correspond à une liste de valeurs à afficher. Le premier élément de tableau contient le texte de l'en-tête de chaque colonne.

On le voit, le code devient de plus en plus long mais la conception de base reste la même : vous créez des nœuds et des nœuds de texte, puis vous les annexez les uns aux autres dans l'ordre qui convient. La Figure 5.7 montre le tableau qui en résulte.

```
<script language="JavaScript"
  type="text/JavaScript">
function createTable(data) {
  var table = document.createElement("table");
  var thead = document.createElement("thead");
  var tr = document.createElement("tr");
```

```
  for (var i = 0; i < data[0].length; i++) {
    var th = document.createElement("th");
    var newText =
      document.createTextNode(data[0][i]);
    th.appendChild(newText);
    tr.appendChild(th);
  }
  thead.appendChild(tr);
  table.appendChild(thead);

  var tbody = document.createElement("tbody");
  for (var i = 1; i < data.length; i++) {
    var tr = document.createElement("tr");
    for (var j=0; j < data[i].length; j++) {
      var td = document.createElement("td");
      var newText =
        document.createTextNode(data[i][j]);
      td.appendChild(newText);
      tr.appendChild(td);
    }
    tbody.appendChild(tr);
  }

  table.appendChild(tbody);
  return table;
}

window.onload = function() {
  var table = createTable([
    ["1", "2", "3", "4", "5"],
    ["one", "two", "three", "four", "five"],
    ["un", "deux", "trois", "quatre", "cinq"],
    ["eins", "zwei", "drei", "vier", "fünf"]]);
  document.body.appendChild(table);
}
</script>
```

Création d'un tableau (table.html)

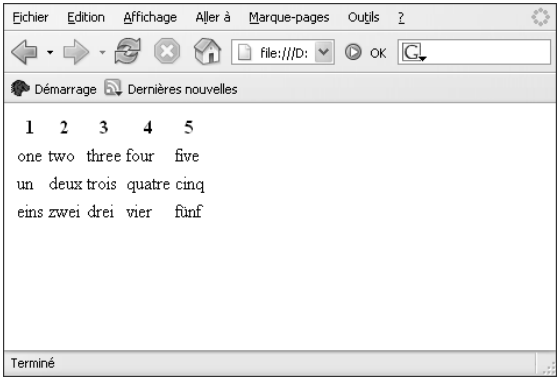

Figure 5.7 : Tableau généré dynamiquement.

Modifications de fragments HTML

```
list.innerHTML += newNode;
```

Quand vous modifiez des nœuds de texte (soit en les remplaçant, soit en utilisant la propriété `nodeValue`), la modification ne porte que sur le texte lui-même, mais pas sur des fragments HTML complets, notamment des sous-éléments. Pour ce faire, la propriété `innerHTML` de chaque élément peut se révéler utile. Même si elle n'est pas standardisée et ne fait pas partie d'une spécification DOM, elle fonctionne bien. Avec elle, vous pouvez modifier le code HTML interne de n'importe quel élément HTML, voire fournir de nouveaux sous-éléments, comme le montre le listing suivant :

```
<script language="JavaScript"
  type="text/JavaScript">
var nr = 1;
```

```
function addItem() {
  var list = document.getElementById("list");
  nr++;
  var newNode = "<li>Item " + nr + "</li>";
  list.innerHTML += newNode;
}
</script>
<ul id="list"><li>Item 1</li></ul>
<input type="button" onclick="addItem();"
  value="Add item" />
```

Ajout d'éléments via *innerHTML (innerhtml.html)*

Ainsi, au lieu d'utiliser la façon propre et pratique de DOM, `innerHTML` écrit simplement le code HTML dans un élément. Les deux méthodes présentent leurs avantages et leurs inconvénients : `innerHTML` nécessite que vous vous occupiez du codage des caractères par vous-même, mais vous permet par ailleurs d'ajouter ou de modifier plusieurs éléments à la fois.

Positionnement des éléments

```
el.style.left = "0px";
el.style.posLeft = 0;
el.style.top = "0px";
el.style.posTop = 0;
```

Les CSS permettent de placer des éléments de deux manières : absolue et relative. Peu importe la méthode que vous choisissez, grâce à JavaScript vous pouvez régler les valeurs de positionnement. Généralement, le positionnement absolu est plus pratique puisque vous n'avez pas besoin de calculer les positions relatives des éléments imbriqués.

Dans la plupart des navigateurs, la propriété `left` définit la coordonnée `x` de l'élément et la propriété `top` sert pour la coordonnée `y`. Les valeurs ne sont pas numériques mais, comme c'est l'habitude dans les CSS, incluent une unité, généralement `px` pour pixels.

Pour Internet Explorer, il vous faut pratiquer différemment. Les propriétés `posLeft` et `posTop` définissent les positions horizontale et verticale, pour lesquelles, cette fois-ci, vous fournissez simplement une valeur numérique et non une unité.

La façon la plus commode consiste à définir toutes ces propriétés, puisqu'il n'y a pas d'effets secondaires. Cela vous évite d'avoir à rechercher le client.

Le code suivant place les éléments `<div>` dans le coin supérieur gauche. Notez que cet élément se trouve maintenant sur le texte de la page.

```
<script language="JavaScript"
  type="text/JavaScript">
function position() {
  var el = document.getElementById("element");
  el.style.left = "0px";
  el.style.posLeft = 0;
  el.style.top = "0px";
  el.style.posTop = 0;
}
window.onload = position;
</script>
<h1>My Portal</h1>
<p>Some sample text. Some sample text.
  Some sample text. Some sample text.
  Some sample text. Some sample text.
  Some sample text. Some sample text.
  Some sample text. Some sample text.
  Some sample text. Some sample text. </p>
```

```
<div id="element" style="position: absolute;
  ➥background-color: #eee; border: 1px solid">
JavaScript Phrasebook
</div>
```

Positionnement d'un élément (position.html)

Déplacement d'éléments

```
id = window.setInterval("animate();", 100);
```

Un scénario DHTML plutôt rare, mais encore utilisé, consiste à positionner un élément, mais également à le déplacer et donc à l'animer. Pour ce faire, les méthodes `window.setTimeout()` et `window.setInterval()` sont assez pratiques. Le code suivant anime une bannière publicitaire sur la diagonale de la page. Le seul problème potentiel réside dans la manière d'animer la position. Pour les propriétés Internet Explorer (`posLeft`, `posTop`), il suffit d'ajouter une valeur. Pour `left` et `top`, vous devez d'abord déterminer l'ancienne position, puis lui ajouter une valeur. La fonction JavaScript `parseInt()` extrait le contenu numérique d'une chaîne comme `"123px"`. Toutefois, `parseInt()` renvoie `NaN` si aucune valeur n'est trouvée dans `left` ou `top`. La fonction d'aide suivante s'occupe donc de cette situation ; dans ce cas, elle renvoie plutôt 0 :

```
function myParseInt(s) {
  var ret = parseInt(s);
  return (isNaN(ret) ? 0 : ret);
}
```

Ensuite, le code suivant anime la bannière et s'arrête après 50 itérations :

```
<script language="JavaScript"
  type="text/JavaScript">
var nr = 0;
var id = null;
function animate() {
  nr++;
  if (nr > 50) {
    window.clearInterval(id);
    document.getElementById("element").style
     ➥.visibility = "hidden";
  } else {
    var el = document.getElementById("element");
    el.style.left =
      (myParseInt(el.style.left) + 5) + "px";
    el.style.posLeft += 5;
    el.style.top =
      (myParseInt(el.style.top) + 5) + "px";
    el.style.posTop += 5;
  }
}
window.onload = function() {
  id = window.setInterval("animate();", 100);
};
</script>
<h1>My Portal</h1>
<div id="element" style="position: absolute;
➥background-color: #eee; border: 1px solid">
JavaScript Phrasebook
</div>
```

Animation d'un élément (animation.html, extrait)

Création d'une navigation toujours apparente

```
window.onload = positionNavigation;
window.onscroll = positionNavigation;
```

Il est parfois important qu'une partie d'une page soit toujours visible. Selon la nature du site, il peut s'agir d'une bannière publicitaire (Geocities, qui fait maintenant partie de Yahoo!, a été parmi les premiers à mettre cela en place) ou des éléments de navigation. Une fois de plus, on intègre pour cela un élément `<div>`. Avec une navigation toujours apparente, la position des éléments reste la même, quand bien même l'utilisateur fait défiler la page. Ainsi, le code qui précède sert à appeler le code de positionnement à la fois lorsque la page se charge et lorsqu'elle défile.

Pour calculer la nouvelle position, il faut calculer le décalage actuel du défilement. Internet Explorer propose pour cela `document.body.scrollLeft` et `document.body.scrollTop` ; les anciens navigateurs utilisent `window.pageXOffset` et `window.pageYOffset`. Le code suivant maintient la position des éléments de navigation et la Figure 5.8 montre le résultat.

```
<script language="JavaScript"
  type="text/JavaScript">
function positionNavigation() {
  var nav = document.getElementById("navigation");
  var x, y;
  var navwidth = 155;
  if (window.innerWidth) {
    x = window.innerWidth + window.pageXOffset - navwidth;
    y = window.pageYOffset + 10;
  } else {
    with (document.body) {
      x = clientWidth + scrollLeft - navwidth;
```

```
      y = scrollTop + 10;
    }
  }
  nav.style.left = x + "px";
  nav.style.posLeft = x;
  nav.style.top = y + "px";
  nav.style.posTop = y;
}
window.onload = positionNavigation;
window.onscroll = positionNavigation;
</script>
<h1>My Portal</h1>
<div id="navigation" style="position: absolute;
   background-color: #eee; border: 1px solid">
<a href="http://www.samspublishing.com/">Sams Publishing
</a><br />
<a href="http://www.amazon.com/gp/product/0672328801">
  This book at Amazon</a><br />
<a href="http://www.hauser-wenz.de/blog/">Author's weblog
</a>
</div>
```

Maintien de la position d'un élément (sticky.html, extrait)

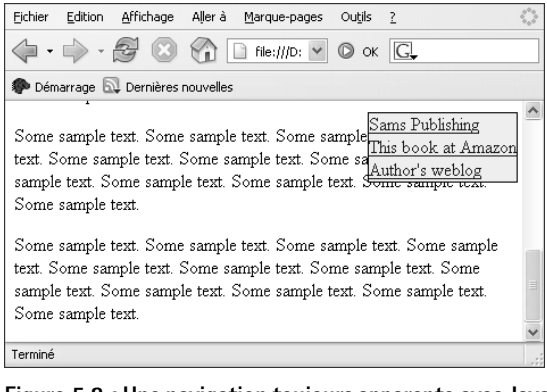

Figure 5.8 : Une navigation toujours apparente avec JavaScript.

Création d'une publicité contextuelle en Flash

```
<param name="wmode" value="transparent" />
```

Les systèmes de blocage des fenêtres contextuelles devenant de plus en plus communs dans les navigateurs, les publicistes sont de plus en plus créatifs. Une fonction négligée des animations Flash intégrées dans la page peut être employée : lorsque vous définissez le paramètre `wmode` sur `"transparent"`, comme dans le code précédent, l'animation Flash est transparente, vous pouvez donc la placer sur le contenu de la page, ce qui crée une fenêtre contextuelle.

Le fichier `flashad.swf` est une simple animation Flash qui comprend un bouton faisant disparaître la publicité lorsque l'utilisateur clique dessus. Le code ActionScript suivant (un langage également fondé sur ECMAScript) y parvient en appelant le code JavaScript dans le navigateur :

```
on (release) {
  getURL("JavaScript:void(document.getElementById(
     'banner').style.visibility='hidden');");
}
```

Le code de la page lui-même est assez comparable au code de la section précédente : comme dans une navigation toujours apparente, l'annonce Flash doit toujours être visible.

```
if (window.innerWidth) {
  x = window.pageXOffset +
    ➥Math.round((window.innerWidth - navwidth) / 2);
  y = window.pageYOffset + 10;
} else {
```

```
  with (document.body) {
    x = scrollLeft +
      Math.round((clientWidth - navwidth) / 2);
    y = scrollTop + 10;
  }
}
```

Bannière publicitaire Flash transparente, section JavaScript (flashad.html, extrait)

La bannière en elle-même se trouve dans un élément `<div>`. Attention, l'identifiant de la bannière doit correspondre à l'identifiant du code ActionScript qui tente de la rendre invisible.

```
<div id="banner" style="position: absolute;">
<object
classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/
  shockwave/cabs/flash/swflash.cab#version=7,0,0,0"
  width="400" height="550" id="flashad">
  <param name="movie" value="flashad.swf" />
  <param name="quality" value="high" />
  <param name="wmode" value="transparent" />
  <param name="bgcolor" value="#ffffff" />
  <embed src="flashad.swf" quality="high"
      wmode="transparent" bgcolor="#ffffff"
      width="400" height="550" name="flashad"
      type="application/x-shockwave-flash"
      pluginspage="http://www.macromedia.com/go/
      getflashplayer" />
</object>
</div>
```

Une bannière publicitaire Flash transparente, section HTML (flashad.html, extrait)

La Figure 5.9 montre le résultat : la bannière réside sur le contenu de la plage ; de plus, le fond de l'animation Flash est transparent.

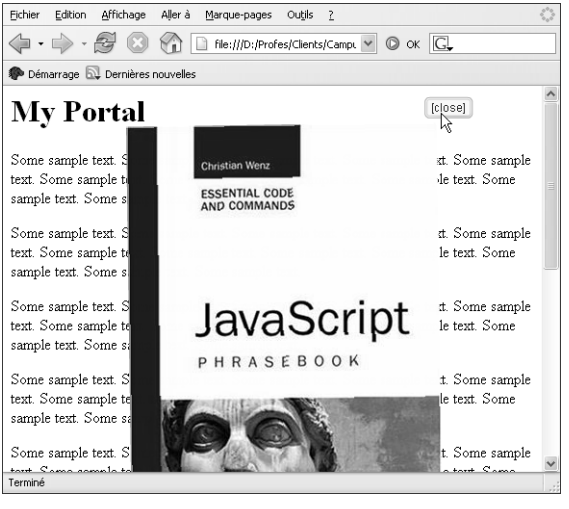

Figure 5.9 : Une bannière Flash transparente.

Attention

Le fait que les annonces Flash n'utilisent pas de fenêtres contextuelles ne signifie pas que les utilisateurs soient plus indulgents à leur égard. Utilisez donc cette technique avec parcimonie et vérifiez qu'il soit bien possible d'afficher la publicité, sinon le contenu de la page est invisible et l'utilisateur – et client potentiel – risque également de disparaître !

6

Programmation orientée objet et événements

Plus les applications JavaScript sont avancées, plus il est nécessaire de structurer le code. Une manière de procéder consiste à utiliser la programmation orientée objet. JavaScript en lui-même n'est pas un langage orienté objet, mais plutôt un langage fondé sur les objets. La prise en charge de ce style de programmation est donc assurée, même si elle est assez limitée.

Le chapitre aborde ensuite la gestion générale des événements dans JavaScript. Outre les bases, vous découvrirez des événements spéciaux (clavier et souris).

Création d'une classe

```
function UniversalClass() { }
```

Il n'existe pas de mot clé distinctif pour les classes dans JavaScript, chacune se définissant sous forme de fonction. La différence entre une fonction ordinaire et ce type de fonction réside dans la manière dont elle est ensuite appelée, avec le mot clé `new`. Le listing suivant implémente une classe simple ; après instanciation, la boîte `window.alert()` s'affiche.

```
<script language="JavaScript"
  type="text/JavaScript">
function UniversalClass() {
  window.alert("Welcome to my class!");
}

var uc = new UniversalClass();
</script>
```

Une classe simple (class.html)

Accès aux membres des classes

```
this.Count = count;
```

Lorsque vous travaillez avec des membres de classe, il est primordial de tous les définir explicitement. Pour qu'un code externe puisse accéder aux membres des classes (les propriétés et les méthodes), il faut utiliser le mot clé `this`. Ainsi, si vous définissez une fonction `XYZ()` dans la classe, sachez qu'elle devient disponible sous forme de méthode de classe, mais uniquement si vous ajoutez ce code :

```
this.XYZ = XYZ;
```

Dans cette méthode, vous pouvez également accéder aux propriétés de classe en utilisant `this`, mais elles doivent aussi être définies. Pour accéder aux membres de classe à la fois en interne et en externe, séparez le nom d'instance du nom de membre par un point (.).

Le code suivant implémente une classe simple. La seule méthode de classe, `Count()` (nom interne : `count()`), prend un paramètre, puis compte dans la langue donnée. La Figure 6.1 montre le résultat.

```
<script language="JavaScript"
  type="text/JavaScript">
function UniversalCounter() {
  this.copyright = "(C) 2006 JavaScript Phrasebook";
  this.Count = count;
  var numbers = {
    "en": "one, two, three",
    "fr": "un, deux, trois",
    "de": "eins, zwei, drei"
  };
  function count(language) {
    if (numbers[language]) {
      window.alert(numbers[language]);
    } else {
      window.alert("Unknown language");
    }
  }
}

var uc = new UniversalCounter();
uc.Count("fr");
</script>
```

Une classe avec des membres (members.html)

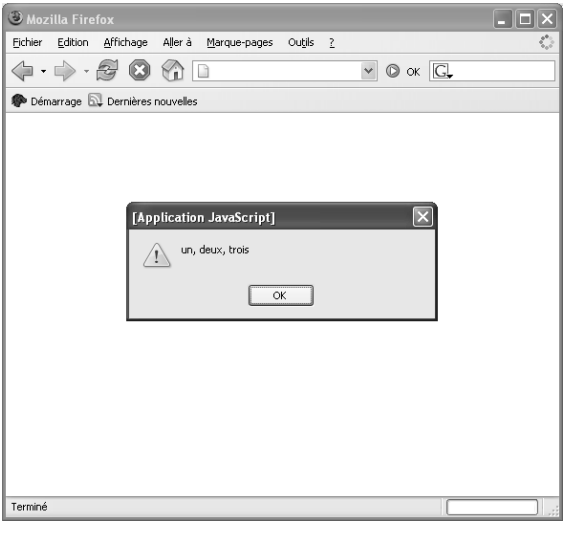

Figure 6.1 : Le "compteur universel".

Info

Un autre moyen de créer des objets consiste à utiliser le constructeur **Object**, de la manière suivante :

```
var uc = new Object();
uc.copyright = "(C) 2006 JavaScript Phrasebook ";
uc.printCopyright = function() {
  window.alert(this.copyright);
};
```

Emulation de membres de classes privées

JavaScript ne possède pas de modificateurs de classe comme `public`, `protected` et `private` pour choisir les personnes pouvant accéder aux membres de classes et celles qui ne le peuvent pas. Toutefois, on le voit à la section précédente, il existe une manière assez simple d'émuler des membres de classes privées : si vous "n'enregistrez" pas une variable ou une fonction dans la classe en utilisant le mot clé `this`, elle n'est visible qu'en interne. Section précédente, l'objet `numbers` est inaccessible depuis l'extérieur mais il est utilisé en interne par la méthode `Count()/count()`.

Héritage de classes

```
UniversalCounter.prototype = new UniversalTranslator();
```

JavaScript ne bénéficie d'aucun système d'héritage pour les classes. Cependant, il est à nouveau possible d'émuler un élément analogue en utilisant le mot clé `prototype`, avec lequel vous pouvez fournir des membres de classes valables pour tous les objets, y compris les objets hérités. Lorsque JavaScript doit rechercher une propriété ou une méthode (par exemple lorsque `object.methodname()` sera exécuté), il recherche d'abord dans la classe, puis dans l'objet `prototype`. Cela permet une sorte d'héritage.

Dans l'exemple suivant, la classe `UniversalTranslator` définit un membre (`copyright`). La classe `UniversalCounter` est ensuite implémentée de manière analogue à ce qui a été présenté précédemment. La commande suivante définit la propriété `prototype` de la classe `UniversalCounter` sur une nouvelle instance de la classe `UniversalTranslator`.

Conséquence, la classe `UniversalCounter` hérite de toutes les propriétés de la classe `UniversalTranslator` et peut y accéder :

```
<script language="JavaScript"
  type="text/JavaScript">
function UniversalTranslator() {
  this.copyright = "(C) 2006 JavaScript Phrasebook";
}

function UniversalCounter() {
  this.Count = count;

  var numbers = {
    "en": "one, two, three",
    "fr": "un, deux, trois",
    "de": "eins, zwei, drei"
  };
  function count(language) {
    if (numbers[language]) {
      window.alert(numbers[language] +
      ➥" [" + this.copyright + "]");
    } else {
      window.alert("Unknown language");
    }
  }
}
UniversalCounter.prototype =
  new UniversalTranslator();

var uc = new UniversalCounter();
uc.Count("de");
</script>
```

Héritage de classe avec prototype (inheritance.html)

La Figure 6.2 montre que cela fonctionne véritablement : la propriété `copyright` est accessible depuis la classe `UniversalCounter`, même si elle est définie dans la classe `UniversalTranslator`.

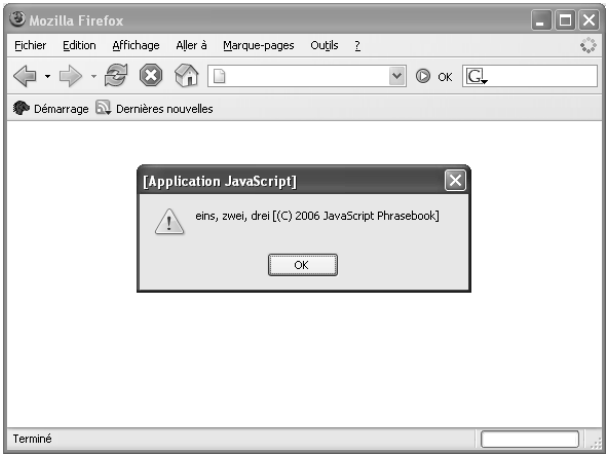

Figure 6.2 : La propriété de classe héritée est présentée.

Attention

Seuls les membres de classe sont hérités, mais pas le code du constructeur de classe. Pour que ce soit le cas, vous devez définir une méthode spécifique comme constructeur de classe et l'appeler manuellement dans la classe dérivée.

Extension d'objets JavaScript intégrés

```
Date.prototype.isLeapYear = isLeapYear;
```

La propriété `prototype` peut aussi servir pour prolonger les classes JavaScript intégrées. Dans le code suivant, une fonction `isLeapYear()` est implémentée. Elle détermine si la valeur de retour de `getFullYear()` est une année bissextile. Notez que la méthode `getFullYear()` n'est pas implémentée ; grâce à `prototype`, `isLeapYear()` devient une méthode de l'objet `Date` et a donc accès à `Date.getFullYear()`.

```
<script language="JavaScript"
  type="text/JavaScript">
function isLeapYear() {
  var y = this.getFullYear();
  return (y % 4 == 0 && (y % 100 != 0 || y % 400 == 0));
}
Date.prototype.isLeapYear = isLeapYear;

var d = new Date();
window.alert(d.isLeapYear());
</script>
```

Extension de la classe Date (extend.html)

Améliorations orientées objet apportées à JavaScript dans Microsoft Atlas

Le cadre d'aplications Microsoft AJAX Atlas (**http://atlas.asp.net/**) implémente également plusieurs nouvelles extensions orientées objet pour JavaScript, qui simplifient la mise en place de quelques techniques orientées objet standard. Parmi ces fonctionnalités, on trouve :

- les espaces de nom ;
- l'héritage et l'accès aux méthodes de base ;
- les classes et les méthodes abstraites ;
- les interfaces.

AJAX faisant actuellement beaucoup d'émules, de plus en plus de bibliothèques, assorties d'une prise en charge orientée objet par JavaScript, émergent. Une autre option qui présente bien le système orientée objet de JavaScript est `prototype.js` (**http://prototype.conio.net/**).

Réaction aux événements JavaScript

```
button.addEventListener("click", eventHandler, false);
button.attachEvent("onclick", eventHandler);
```

La réaction aux événements JavaScript peut être réalisée de deux manières :

- avec l'attribut HTML : `<body onload="xyz();">` ;
- avec l'attribut JavaScript `onXXX` : `window.onload = xyz;`.

Il existe toutefois plusieurs mécanismes d'événements concurrentiels dans les différents navigateurs. Internet Explorer accepte que des événements soient attachés à un élément à l'aide de la méthode `attachEvent()`. Le nom de

l'événement ici est égal à l'attribut HTML, pour que vous puissiez utiliser `"onload"`, par exemple (même si l'événement est appelé `"load"`).

Tous les autres grands navigateurs acceptent la méthode `addEventListener()`, qui fait partie du modèle W3C. Ici, vous apportez le nom de l'événement, donc juste `"load"` au lieu d'`"onload"`.

L'exemple suivant montre comment attacher un événement à un bouton de manière que cela fonctionne dans tous les navigateurs :

```
<script language="JavaScript"
  type="text/JavaScript">
function eventHandler() {
  window.alert("Event fired!");
}

window.onload = function() {
  var button =
    document.getElementById("eventButton");
  if (button.addEventListener) {
    button.addEventListener("click", eventHandler, false);
  } else if (button.attachEvent) {
    button.attachEvent("onclick", eventHandler);
  }
};
</script>
<input type="button" id="eventButton"
  value="Click me!" />
```

Attachement d'un événement (attach.html)

Info

Vous pouvez également supprimer les gestionnaires d'événements. Internet Explorer utilise `detachEvent()`, tandis que d'autres navigateurs suivent les recommandations du W3C et nomment leurs fonctions `removeEventListener()`.

Comprendre le "bouillonnement" des événements

En matière de gestion des événements, les navigateurs actuels optent pour l'un de ces deux concepts. Internet Explorer opère selon ce qu'on appelle le "bouillonnement d'événements" : un événement est déclenché depuis l'élément dans lequel il survient, puis il remonte la structure DOM. Il est donc possible de le capturer et d'y réagir à différents endroits. Imaginons le marquage suivant :

```
<div><p><em>JavaScript</em> Phrasebook</p></div>
```

Si le curseur de la souris survole le texte JavaScript, l'événement `mouseover` est déclenché dans l'élément `<em>`, puis remonte dans les éléments `<p>` et `<div>`.

L'autre modèle est celui du W3C, pris en charge par les navigateurs Mozilla, Opera et Safari/Konqueror. Ici, les événements sont tout d'abord envoyés vers le bas, vers l'élément cible, puis ils remontent. Ainsi, dans l'exemple précédent, l'événement "visite" les éléments `<div>`, `<p>` et `<em>`, puis remonte *via* les éléments `<p>` et `<div>`. Lorsque vous ajoutez un écouteur d'événements, vous pouvez indiquer, dans le troisième paramètre de `addEventListener()`, si l'événement sera intercepté lors de sa descente (`true`) ou lors de sa remontée (`false`).

Lorsqu'un événement a été intercepté, il est aussi possible de l'empêcher de replonger ou de remonter.

Dans Internet Explorer, réglez la propriété `cancelBubble` de l'événement sur `false` :

```
window.event.cancelBubble = false;
```

Le modèle du W3C accepte la méthode `stopPropagation()`.

```
e.stopPropagation();
```

Vous le voyez, dans Internet Explorer, l'événement actuel est toujours disponible *via* `window.event`, tandis que d'autres navigateurs reçoivent automatiquement l'événement sous forme de paramètre (ici appelé `e`) pour l'écouteur d'événements.

Evénements de clavier

```
document.onkeydown = showKey;
```

Les événements du clavier ne font pas partie de DOM niveau 1 ou 2, mais ils sont implémentés dans les navigateurs récents. L'accès aux événements diffère de la méthode générale (`window.event` dans Internet Explorer ; l'événement servant de paramètre automatique de la fonction dans tous les autres navigateurs). Mais alors, la propriété `keyCode` renvoie le code ASCII de la touche, qui peut ensuite être traité, comme dans le code suivant :

```
done. Christian<script language="JavaScript"
  type="text/JavaScript">
function showKey(e) {
  var key;
  if (window.event) {
    key = window.event.keyCode;
  } else {
    key = e.keyCode;
  }
  key = String.fromCharCode(key);
  document.getElementById("para").innerHTML += key;
}

window.onload = function() {
  document.onkeydown = showKey;
}
</script>
<p id="para">Click and type here: </p>
```

Ecoute des événements du clavier (key.html)

La Figure 6.3 présente les touches qui sont activées.

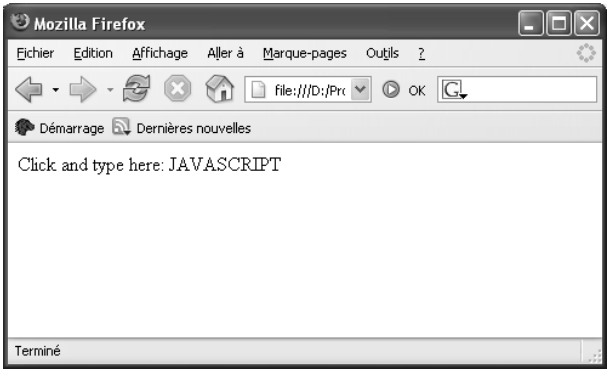

Figure 6.3 : Les touches activées sont affichées.

Info

Tous les caractères de la Figure 6.3 sont présentés en majuscules car JavaScript prend en compte la touche sur laquelle l'utilisateur a appuyé, mais non s'il s'agit d'une majuscule ou d'une minuscule. Vous pouvez toutefois savoir si des touches spéciales ont été enfoncées en étudiant les propriétés `altKey`, `ctrlKey` et `shiftKey`.

Envoi d'un formulaire avec la touche Entrée

```
if (key == 13) {document.forms[0].submit();
}
```

Selon le type et la configuration du navigateur, le fait d'appuyer sur la touche Entrée dans un champ de formulaire ne permet pas toujours d'envoyer le formulaire.

Quelquefois, par exemple, le bouton qui envoie le formulaire se trouve dans un autre cadre. Dans ce cas, vous pouvez ajouter un peu de code JavaScript pour vous assurer que la touche Entrée permet d'envoyer les données du formulaire.

Il suffit, pour implémenter cette fonctionnalité, d'utiliser l'écouteur d'événement de touche standard de la section précédente. Le code de la touche Entrée est 13. Ainsi, la présence de ce code indique que le formulaire est envoyé :

```
<script language="JavaScript"
  type="text/JavaScript">
function checkKey(e) {
  var key;
  if (window.event) {
    key = window.event.keyCode;
  } else {
    key = e.keyCode;
  }
  if (key == 13) {
    document.forms[0].submit();
  }
}
window.onload = function() {
  document.forms[0].elements["field"].onkeydown =
   ➥checkKey;
}
</script>
<form>
<input name="field" type="text" />
</form>
```

Envoi d'un formulaire lorsque la touche Entrée a été activée (formsubmit.html)

Evénements de souris

```
document.onmousemove = showPosition;
```

L'utilisation de la souris donne lieu à un événement assez intéressant à écouter (en dehors de `click`, bien entendu) : `mousemove`. La position de la souris peut en effet être déterminée si l'on étudie certaines de ses propriétés. Une fois de plus, ces propriétés dépendent du groupe de navigateurs auquel appartient le client, Internet Explorer ou le reste du monde :

- `clientX` et `clientY` pour Internet Explorer ;
- `pageX` et `pageY` pour tous les autres navigateurs.

Voici un listing complet montrant la position actuelle de la souris dans la barre d'état du navigateur (s'il y en a) :

```
<script language="JavaScript"
  type="text/JavaScript">
function showPosition(e) {
  var x, y;
  if (window.event) {
    x = window.event.clientX;
    y = window.event.clientY;
  } else {
    x = e.pageX;
    y = e.pageY;
  }
  window.status = "x: " + x + ", y: " + y;
}
window.onload = function() {
  document.onmousemove = showPosition;
}
</script>
```

Suivi des mouvements de la souris (mouse.html)

Attention

Certains navigateurs n'autorisent pas JavaScript à actualiser la barre d'état, notamment les récentes versions de Firefox. Vous n'obtenez pas de message d'erreur mais la barre d'état reste inchangée. La Figure 6.4 montre le résultat dans un navigateur qui accepte la modification du texte dans la barre d'état.

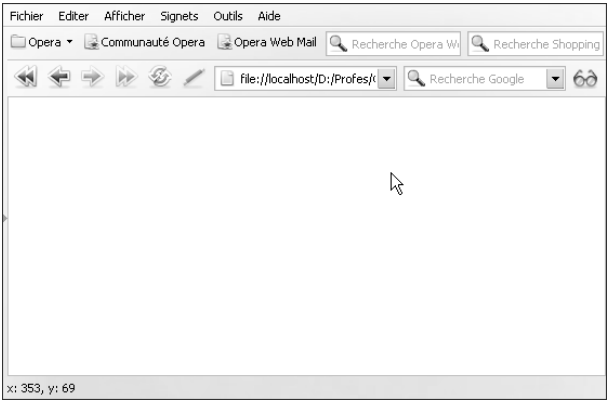

Figure 6.4 : La position de la souris est sans cesse mise à jour.

Distinguer les boutons de souris

Pour savoir lequel des boutons de souris a été activé (lorsque vous écoutez l'événement `click`), les navigateurs proposent, une fois de plus, deux solutions différentes. Les deux utilisent la propriété `button` de la propriété de l'événement mais les valeurs sont différentes.

Mozilla se conforme au standard du W3C : 0 désigne le bouton gauche, 1 le bouton médian, 2 le bouton droit. Toutefois, le reste du monde, dont Microsoft et d'autres navigateurs, adopte une méthode plus logique : 1 désigne le bouton gauche, 2 le bouton droit et 4 le bouton du milieu. Comme vous le voyez, ces chiffres sont tous des puissances de 2. Vous pouvez donc associer ces valeurs. Par exemple, la valeur 3 indique que les boutons droite et gauche ont été enfoncés en même temps.

7

Les cookies

JavaScript est une pure technologie côté client. En théorie, elle peut être utilisée sur le serveur, mais cela n'a rien à voir avec une exécution dans le navigateur ; la syntaxe est similaire mais le contexte totalement différent.

JavaScript se limite généralement à la page en cours et n'a pas accès aux éléments provenant du serveur. Il existe toutefois quelques très rares exceptions à cette règle. L'une d'elles est expliquée au Chapitre 11, "AJAX et sujets annexes". Une autre est traitée ici.

Les cookies ne sont pas une technologie spécifique au navigateur, mais bien un mécanisme du modèle client/serveur visant à surmonter une limitation importante du protocole HTTP. HTTP ne possède pas d'état, ce qui signifie qu'il ne possède pas de mémoire. Un client ouvre une connexion vers un serveur, récupère un document (une image ou d'autres données), puis ferme la connexion. Par la suite, même si le serveur envoie des données à ce client spécifique, il le fera sans le reconnaître.

Les cookies peuvent aider à surmonter cette restriction. Un serveur envoie des informations de texte courtes au client, appelées cookie. Si et seulement si ces données sont acceptées et enregistrées localement, le client les renvoie avec chaque requête adressée au même serveur. L'application du serveur reconnaît alors l'utilisateur.

L'aspect intéressant, c'est que JavaScript peut à la fois obtenir (lire) et définir (écrire) des cookies, ce qui permet de contourner le protocole HTTP. De nouveaux scénarios apparaissent, jusqu'alors impossibles avec le HTTP simple.

Les autres mécanismes permettant d'obtenir un résultat similaire sont les sessions, mais elles doivent être implémentées par la technologie côté serveur et ne concernent pas JavaScript.

Les cookies

Lorsqu'un serveur Web envoie un cookie à un client, il utilise une entrée d'en-tête HTTP ressemblant à ceci :

```
Set-Cookie: session=abcd1234; expires=Tue,
   20-Feb-2007 12:10:52 GMT; path=/;
   domain=.samspublishing.com
```

Le client reçoit ensuite les informations de cookie et, selon ses capacités ou sa configuration, réalise l'une des quatre actions suivantes :

- Il ignore le cookie.
- Il accepte le cookie.
- Il demande à l'utilisateur s'il veut accepter le cookie.
- Il refuse le cookie.

Les première et dernière actions sont identiques. Il est impossible au serveur Web de savoir si un cookie a été refusé par l'utilisateur, refusé par la configuration du client ou ignoré par le client du fait de l'absence de prise en charge des cookies.

S'ils sont acceptés, les cookies sont ensuite renvoyés au serveur, si une série de conditions est respectée. L'entrée d'en-tête HTTP associée ressemble alors à ceci :

```
Cookie: session=abcd1234
```

Un cookie peut être lié à un domaine et à un chemin. Il est donc généralement renvoyé uniquement au serveur émetteur. Sachez qu'il est possible d'écraser la valeur de domaine dans un cookie, mais certains navigateurs le refusent alors automatiquement.

De même, les cookies présentent certaines limites. Tous les navigateurs ne les prennent pas en charge de la même manière, mais les configurations minimales suivantes sont obligatoires :

- 4 Ko (4 096 octets) par cookie ;
- 20 cookies par domaine ;
- 300 cookies au total.

Info

Il n'existe pas de spécification officielle des cookies pris en charge dans les navigateurs, mais tous les clients importants acceptent une spécification "préliminaire" et propriétaire publiée par Netscape dans les années 1990. Elle est toujours disponible pour consultation à l'adresse http://wp.netscape.com/newsrf/std/cookie_spec.html.

Paramétrage des cookies

```
document.cookie = "myLanguage=JavaScript";
```

Le paramétrage d'un cookie avec JavaScript revient à paramétrer la propriété `cookie` de l'objet `document`. Il est important d'utiliser le même format que celui du protocole HTTP pour envoyer un cookie au client. Pour les "cookies ordinaires" (sans date d'expiration ni autres restrictions, comme indiqué aux sections suivantes), le nom et la valeur du cookie doivent être séparés par un signe égal (=). Le nom du cookie ne doit pas contenir de caractères spéciaux comme des espaces, des points-virgules et des espaces blancs ; ces caractères spéciaux doivent être codés en URL (par exemple, un espace se transforme en `%20`, la représentation hexadécimale de sa valeur ASCII). La spécification recommande le codage, sans le rendre obligatoire.

Vous le voyez dans le listing suivant, paramétrer plusieurs cookies peut se faire en réglant plusieurs valeurs sur `document.cookie`. Ainsi, écrire sur `document.cookie` n'écrase pas les cookies précédents mais en ajoute. Seule exception à cette règle, s'il existe déjà un cookie du même nom, le client tente de l'écraser.

```
<script language="JavaScript"
  type="text/JavaScript">
document.cookie = "myLanguage=JavaScript";
document.cookie =
  "myOtherLanguage=PHP:%20Hypertext%20Preprocessor"
</script>
```

Paramétrage d'un cookie avec JavaScript (setcookie.html)

La Figure 7.1 montre le résultat dans le navigateur s'il est configuré pour interroger l'utilisateur avant de paramétrer un cookie.

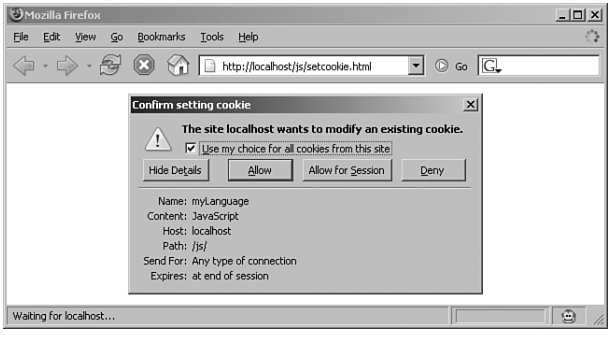

Figure 7.1 : Le navigateur tente de paramétrer le cookie envoyé par JavaScript.

Lecture des cookies

```
var cookies = document.cookie.split(/; /g);
```

Lorsqu'il accède à `document.cookie`, JavaScript récupère une liste de tous les cookies que le navigateur doit envoyer au serveur actuel. Malheureusement, cet accès n'est pas disponible sous forme de tableau mais de chaîne. Par exemple, la section précédente générerait la valeur suivante pour `document.cookie` :

```
myLanguage=JavaScript;
    ➥myOtherLanguage=PHP:%20Hypertext%20Preprocessor
```

Ainsi, pour récupérer les cookies de cette valeur, il faudrait suivre les étapes suivantes :

1. Séparer la chaîne de cookie à "; " (pour obtenir des cookies individuels).
2. Identifier le premier signe égal (=) dans chaque cookie comme délimiteur de valeur et de nom.

Puisque les valeurs de cookie peuvent contenir des signes égal, c'est la première occurrence du signe qui doit être utilisée. Le code suivant affiche tous les cookies sous la forme d'un tableau HTML. Avant d'être écrits dans la page, les caractères spéciaux HTML sont correctement échappés grâce à la fonction `HtmlEscape()`.

```
<script language="JavaScript"
  type="text/JavaScript">
document.write("<table><tr><th>Name</th>
     <th>Value</th></tr>");
var cookies = document.cookie.split(/; /g);
for (var i=0; i<cookies.length; i++) {
  var cookie = cookies[i];
  if (cookie.indexOf("=") == -1) {
    continue;
  }
  var name =
    cookie.substring(0, cookie.indexOf("="));
  var value =
    cookie.substring(cookie.indexOf("=") + 1);
  document.write("<tr><td>" +
                 HtmlEscape(name) +
                 "</td><td>" +
                 HtmlEscape(unescape(value)) +
                 "</td></tr>");
}
document.write("</table>");
</script>
```

Lecture de cookies (getcookies.html, extrait)

La Figure 7.2 montre la sortie possible du listing précédent.

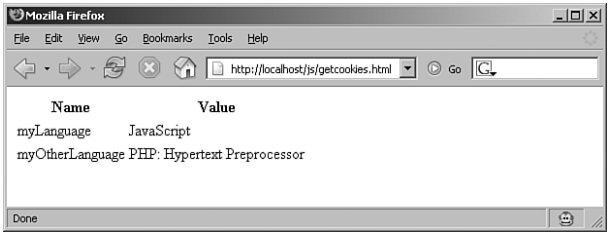

Figure 7.2 : Tous les cookies du système sont affichés dans le navigateur.

Lorsque la valeur d'un cookie spécifique est exigée, une fonction d'aide peut être pratique. Elle recherche le nom du cookie donné et renvoie la valeur (tout ce qui se trouve à droite du nom du cookie, jusqu'au prochain point-virgule à la fin de la chaîne) :

```
function getCookie(name) {
  var pos = document.cookie.indexOf(name + "=");
  if (pos == -1) {
    return null;
  } else {
    var pos2 = document.cookie.indexOf(";", pos);
    if (pos2 == -1) {
      return unescape(
        document.cookie.substring(pos + name.length + 1));
    } else {
      return unescape(
        document.cookie.substring(pos + name.length + 1,
        ➥pos2));
    }
  }
}
```

Lecture d'un seul cookie (getsinglecookie.html)

Astuce

N'oubliez jamais d'enlever l'échappement sur les données de valeur des cookies. Dans l'exemple donné, cela a été réalisé avec `unescape()` car les cookies originaux ont été échappés à l'aide d'`escape()`. Avec un autre schéma d'échappement, vous devez utiliser le mécanisme d'annulation d'échappement adéquat dans votre code.

Etablissement d'une date d'expiration

```
document.cookie="myLanguage=JavaScript; expires=Tue,
                ➥25-Dec-2007 12:34:56 GMT"
```

Par défaut, les cookies expirent lorsque l'utilisateur ferme le navigateur. Ils sont ensuite appelés "cookies temporaires". A l'inverse, on trouve les cookies persistants, qui survivent à la fermeture de la session du navigateur. Ils existent toujours lorsque le navigateur est redémarré.

Pour que cela fonctionne, le cookie a besoin d'une date d'expiration indiquant la période pendant laquelle il peut demeurer dans le navigateur. La spécification des cookies exige que la date d'expiration soit indiquée en GMT (heure de Greenwich) et que le format utilisé soit le suivant :

```
Wdy, JJ-Moi-AAAA HH:MM:SS GMT
```

Voici une valeur possible :

```
Tue, 25-Dec-2007 12:34:56 GMT
```

Le format complique singulièrement la détermination d'informations de date correctes (en particulier pour le jour de la semaine) ; à noter que des bibliothèques externes peuvent vous venir en aide.

Lorsque la date est déterminée, elle doit être utilisée dans la valeur `document.cookie`, comme l'a montré le code du début de cette section (voir `expirecookie.html`).

La Figure 7.3 montre le résultat de ce code sur un système utilisant le fuseau horaire GMT +1, donc d'une heure en avance sur l'horaire GMT. Bien entendu, ces informations ne sont affichées que lorsque le navigateur est configuré pour interroger l'utilisateur lorsqu'un cookie arrive.

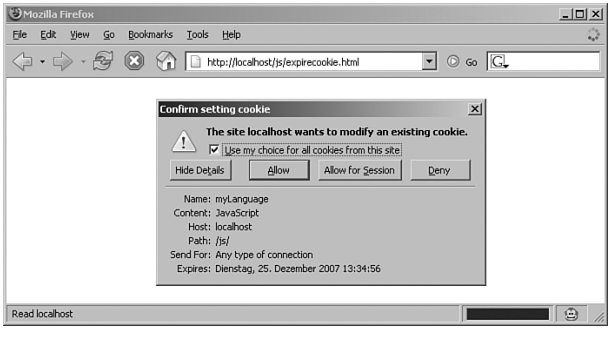

Figure 7.3 : Ce cookie est livré avec une date d'expiration.

Autres options de cookies

```
document.cookie="myLanguage=JavaScript; expires=Tue,
                ➥25-Dec-2007 12:34:56 GMT; path=/;
                ➥domain=.example.com; secure"
```

Même si la date d'expiration d'un cookie est de loin la fonction la plus utilisée, il en existe d'autres :

- `path`. Définit le chemin pour accéder au cookie. La valeur par défaut correspond au chemin en cours. Si elle est réglée sur `/`, l'ensemble du site reçoit le cookie.

- `domain`. Précise le(s) domaine(s) à utiliser. Selon la spécification, au moins deux points sont nécessaires dans le nom de domaine, mais de nombreux navigateurs ignorent cette spécificité. Si elle est réglée sur `.exemple.com`, `www.exemple.com` et `sousdomaine.exemple.com` reçoivent le cookie.
- `secure`. Si cette fonction est configurée, c'est-à-dire simplement placée dans la chaîne du cookie, celui-ci n'est transmis que par une connexion sécurisée (HTTPS).

Le code qui précède paramètre un cookie lié au domaine `.exemple.com` et sera transmis par des connexions HTTPS uniquement. Notez que le listing associé (`cookieoptions.html`) ne fonctionnera probablement pas sur votre système à moins que ne vous modifiiez le nom de domaine, pour utiliser le véritable nom de domaine employé.

Attention

Pour écraser un cookie, assurez-vous d'utiliser les mêmes fonctions (`name`, `path`, `domain`, `secure`) que lors de sa définition ; sa date d'expiration et sa valeur peuvent bien entendu changer. Le navigateur crée alors un nouveau cookie.

Suppression de cookies

```
document.cookie="myLanguage=JavaScript; expires=Thu,
                ➥25-Dec-1980 12:34:56 GMT"
```

Le paramétrage d'une valeur de cookie sur une chaîne vide détruit efficacement les données du cookie mais n'élimine pas le cookie lui-même. Pour s'en débarrasser, il faut définir sa date d'expiration sur une date passée. Puisque la mesure

Utilisation de cookies HTTP uniquement

Les versions récentes du navigateur Web Internet Explorer de Microsoft prennent en charge une option supplémentaire pour les cookies : `HttpOnly`. Si elle est définie, le cookie est transmis par HTTP uniquement mais ne peut pas être lu avec JavaScript. Cette option permet d'éviter certains problèmes de sécurité de quelques sites Web qui autorisent le vol de données de cookies sensibles avec JavaScript. Toutefois, `HttpOnly` est une option propriétaire et n'est prise en charge que par les navigateurs Microsoft. Vous en saurez plus à l'adresse **http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp**.

du temps UNIX commence le 1er janvier 1970, optez pour une date proche de ce jour. Le code qui précède (voir `deletecookie.html`) utilise la date de Noël 1980. Le navigateur Web constate que cette date est passée (à moins que l'utilisateur n'ait modifié les paramètres régionaux) et supprime le cookie. Une fois de plus, utilisez les mêmes fonctions de cookie que celles de son paramétrage.

Vérification de la prise en charge des cookies

```
document.cookie="CookieTestTemp=success"
document.cookie="CookieTestPers=success; expires=Tue,
                ➥ 25-Dec-2012 12:34:56 GMT"
```

La seule manière de découvrir si un cookie a été paramétré consiste à le rechercher lors de la prochaine requête

au serveur. Avec JavaScript, l'opération peut être réalisée instantanément dans une page. Le code suivant définit d'abord deux cookies (l'un temporaire et l'autre persistant) et vérifie si ces deux actions ont réussi. Les cookies de test sont à nouveau effacés, uniquement si leur paramétrage a réussi.

```
<script language="JavaScript"
  type="text/JavaScript">
document.cookie="CookieTestTemp=success"
document.cookie="CookieTestPers=success; expires=
   Tue, 25-Dec-2012 12:34:56 GMT"
if (document.cookie.indexOf(
  "CookieTestTemp=success") != -1) {
  window.alert("Temporary cookies are supported!")
  document.cookie="CookieTestTemp=JavaScript;
   expires=Thu, 25-Dec-1980 12:34:56 GMT"
} else {
  window.alert(
    "Temporary cookies are not supported!")
}
if (document.cookie.indexOf(
  "CookieTestPers=success") != -1) {
  window.alert("Persistent cookies are supported!")
  document.cookie="CookieTestPers=JavaScript;
   expires=Thu, 25-Dec-1980 12:34:56 GMT"
} else {
  window.alert(
    "Persistent cookies are not supported!")
}
</script>
```

Vérification de la prise en charge des cookies par le navigateur (checkcookie.html)

Enregistrement de plusieurs informations dans un cookie

```
for (var el in data) {namevalue += "&" + escape(el) +
    "=" + escape(data[el]);
}
```

N'oubliez pas que des restrictions s'imposent à un site dirigé par les cookies : seuls 20 cookies par domaine sont autorisés. Si les utilisateurs reçoivent un avertissement à l'arrivée d'un cookie, plus vous en envoyez et plus vous risquez de les ennuyer. Stocker plusieurs informations dans un seul cookie peut donc être intelligent.

Pour cela, il faut sérialiser les données du cookie. Il existe différentes manières, dont certaines sont très avancées. Toutefois, pour stocker une liste de paires nom/valeur, le plus simple reste le codage URL :

```
function serialize(data) {
  var namevalue = "";
  for (var el in data) {
    namevalue += "&" + escape(el) + "=" + escape(data[el]);
  }
  return namevalue.substring(1);
}
```

Sérialisation d'un objet (serialize.html, extrait)

Lorsqu'un cookie contient des données complexes, on peut utiliser un code comparable à celui-ci :

```
document.cookie = "data=" + serialize(myComplexData)
```

L'opération inverse nécessite un peu plus de code, mais consiste principalement à séparer les données en paires individuelles et à extraire les noms et les valeurs du résultat :

```
function unserialize(data) {
  var object = new Array();
  var pairs = data.split(/&/g);
  for (var i=0; i<pairs.length; i++) {
    if (pairs[i].indexOf("=") > -1) {
      object[unescape(pairs[i].substring(
        ➥0, pairs[i].indexOf("=")))] =
        ➥unescape(pairs[i].substring(
        ➥pairs[i].indexOf("=") + 1));
    }
  }
  return object;
}
```

Désérialisation d'un objet (serialize.html, extrait)

Le fichier de code `serialize.html` sérialise et désérialise un objet, sans utiliser de cookies pour raccourcir un peu le code.

Info

Vous trouverez à l'adresse http://www.iconico.com/workshop/jsSerializer/ un sérialiseur plus complexe, qui transforme un objet en XML. Le Chapitre 11, "AJAX et sujets annexes", propose aussi de plus amples informations, dans sa partie consacrée au JSON.

8

Les formulaires

Les formulaires sont un aspect très utile du HTML lorsqu'on travaille avec les technologies côté serveur car ils permettent de "communiquer" avec le serveur : des données sont saisies dans le formulaire qui est ensuite envoyé au serveur.

Les formulaires HTML sont également très intéressants pour JavaScript. Leurs options sont quelque peu limitées, mais JavaScript apporte son aide. Attention, les données utilisateur devront être validées, les formulaires peuvent n'accepter que certains types d'entrées, l'envoi de formulaires peut n'être possible que lorsque certaines exigences sont respectées, etc. Tout cela, et bien d'autres choses, est rendu possible par JavaScript, comme vous allez le voir.

Formulaires HTML avec JavaScript

document.forms[0]

Généralement, on accède à un élément HTML grâce à son identifiant, puis avec `document.getElementById()`. Toutefois, pour les formulaires HTML, on utilisera `document.forms`. En effet, on fait appel à l'attribut de nom de chaque élément de formulaire pour l'envoyer au serveur.

`document.forms` représente un tableau de tous les formulaires de la page en cours. Ainsi, si la page ne contient qu'un formulaire, on y accède par `document.forms[0]`. Les formulaires peuvent aussi obtenir un nom :

```
<form name="mainForm">
...
</form>
```

On accède alors au formulaire avec `document.forms["mainForm"]`.

Tous les éléments du formulaire sont également référencés par leurs noms, qui servent d'indices de tableau pour sa propriété `elements`. Imaginons que le premier formulaire d'une page possède un élément portant l'attribut `name="element1"`. Le code JavaScript suivant y accède :

```
document.forms[0].elements["element1"]
```

Il existe d'autres manières, plus courtes, d'accéder à ces informations. Ainsi, un formulaire nommé `"mainForm"` et un élément `"element1"` permettent ce raccourci :

```
document.mainForm.element1
```

On emploie habituellement la manière plus détaillée qui fait appel au tableau `forms` et plus particulièrement au tableau `elements`, car elle n'est autorisée qu'avec un accès automatisé aux éléments de formulaire.

JavaScript est capable de modifier les éléments de formulaire, d'agir sur certains événements déclenchés et d'envoyer le formulaire (ou d'empêcher son envoi). De même, il montre toute son utilité dans le domaine de la validation des données de formulaire mais gardez toujours à l'esprit qu'il peut être désactivé : le formulaire doit donc également fonctionner sans JavaScript.

Astuce

Chaque élément de formulaire accepte la propriété `form`, qui pointe vers le formulaire dans lequel il réside. Ainsi, `this.form` est souvent employé dans le code pour que des éléments de champ de formulaire accordent un accès facile au formulaire de l'élément, sans avoir à parcourir le tableau `document.forms`.

Accès aux champs de texte

```
window.alert(f.elements["textfield"].value);
```

HTML accepte trois types de champs :

- des champs de texte sur une seule ligne : `<input type="text" />` ;
- des champs de texte sur plusieurs lignes : `<textarea></textarea>` ;
- des champs de mot de passe : `<input type="password" />`.

Ces champs se comportent différemment dans un navigateur Web, mais l'accès depuis JavaScript est assez similaire pour les trois. Leur attribut `value` contient le texte du champ. Il peut servir pour lire et pour écrire le texte du champ.

Le code suivant montre deux choses : comment accéder à la propriété du champ, puis comment utiliser `this.form` pour offrir un accès facile au formulaire du champ.

```
<script language="JavaScript"
  type="text/JavaScript">
function showText(f) {
  window.alert(f.elements["textfield"].value);
}
</script>
<form>
  <input type="text" name="textfield" />
  <input type="button" value="Show text"
    onclick="showText(this.form);" />
</form>
```

Accès à un champ de texte (textfield.html)

La Figure 8.1 montre ce qui se passe si un utilisateur entre du texte dans le champ puis clique sur le bouton.

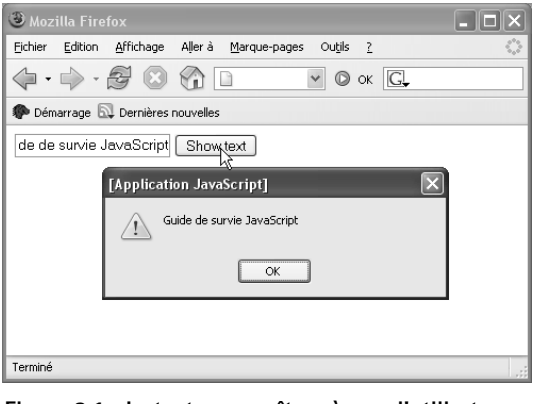

Figure 8.1 : Le texte apparaît après que l'utilisateur a cliqué sur le bouton.

Info

Cette méthode, qui utilise la propriété `value` pour accéder aux données du champ de formulaire, fonctionne également pour les champs masqués (`<input type="hidden" />`).

Accès aux cases à cocher

```
f.elements["chkbox"].checked ?
  "checked." : "not checked."
```

Une case à cocher HTML ne connaît que deux états, cochée ou non cochée. Lorsque vous travaillez avec JavaScript, le scénario le plus commun consiste à accéder à cet état.

La propriété `checked` de la case à cocher est une valeur booléenne renvoyant `true` si la case est cochée et `false` dans le cas contraire. Le code suivant illustre cela :

```
<script language="JavaScript"
  type="text/JavaScript">
function showStatus(f) {
  window.alert("The checkbox is " +
    (f.elements["chkbox"].checked ?
      "checked." : "not checked."));
}
</script>
<form>
  <input type="checkbox" name="chkbox" />
  <input type="button" value="Show status"
    onclick="showStatus(this.form);" />
</form>
```

Accès à une case à cocher (checkbox.html)

Accès à des boutons radio

```
var btn = f.elements["radiobutton"][i];
s += btn.value + ": " + btn.checked + "\n";
```

A la différence des cases à cocher, les boutons radio HTML sont toujours présentés par groupe. Cela signifie que plusieurs boutons radio peuvent avoir le même attribut `name`, mais que leurs attributs `value` diffèrent. Ainsi, `document.forms[`*`nombre`*`].elements[`*`groupeboutonsradio`*`]` accède à l'ensemble du groupe de boutons radio, c'est-à-dire un tableau. Chaque sous-élément de ce tableau correspond à un bouton radio et accepte la propriété `checked`, laquelle fonctionne de la même manière que celle de la case à cocher : `true` signifie que le bouton radio est activé et `false` le contraire.

L'accès à la valeur de chaque bouton est également possible : c'est la propriété `value` qui s'en occupe.

Le code suivant analyse tous les boutons radio et donne leur état :

```
<script language="JavaScript"
  type="text/JavaScript">
function showStatus(f) {
  var s = "";
  for (var i=0; i<f.elements["radiobutton"].length;
    i++) {
    var btn = f.elements["radiobutton"][i];
    s += btn.value + ": " + btn.checked + "\n";
  }
  window.alert(s);
}
</script>
<form>
  <input type="radio" name="radiobutton"
    value="R" />red
```

```
  <input type="radio" name="radiobutton"
    value="G" />green
  <input type="radio" name="radiobutton"
    value="B" />blue
  <input type="button" value="Show status"
    onclick="showStatus(this.form);" />
</form>
```

Accès à un groupe de boutons radio (radiobutton.html)

Accès à des listes de sélection

```
var index = f.elements["selectionlist"].selectedIndex;
```

Une liste de sélection HTML comprend un élément `<select>`, qui pose les bases de la liste et fournit son nom complet (dans son attribut `name`). Les éléments de liste sont représentés par deux sections `<option>` et contiennent une légende (les données présentées dans le navigateur) et une valeur (les données envoyées au serveur lorsque le formulaire est envoyé).

Lorsque vous travaillez avec JavaScript, deux manières d'accéder aux données de liste sont possibles :

- `selectedIndex`. Fournit l'indice (en commençant par `0`) de l'élément de liste actuellement sélectionné ; une valeur de `-1` signifie qu'aucune valeur n'a été sélectionnée (applicable uniquement pour les listes dont la taille est supérieure à `1`).
- `options`. Représente un tableau comptant toutes les options de liste. Chaque option accepte la propriété `selected`. Lorsqu'elle renvoie `true`, cela signifie que l'option de liste est sélectionnée.

Généralement, `selectedIndex` suffit pour la validation. La méthode avec `options` est assez pratique lorsqu'on accède également à l'élément de liste sélectionné. Ensuite, l'attribut `value` de l'option sélectionnée fournit des données envoyées au serveur et la propriété `text` renvoie la légende visible dans le navigateur.

Le listing suivant accède à toutes les informations importantes concernant l'option sélectionnée :

```
<script language="JavaScript"
  type="text/JavaScript">
function showStatus(f) {
  var index = f.elements["selectionlist"].selectedIndex;
  if (index == -1) {
    window.alert("No element selected");
  } else {
    var element = f.elements["selectionlist"]
    ➥.options[index];
    window.alert("Element #" + index + " (caption: " +
    ➥element.text + ", value: " + element.value + ")
    ➥selected");
  }
}
</script>
<form>
  <select name="selectionlist" size="3">
    <option value="R">red</option>
    <option value="G">green</option>
    <option value="B">blue</option>
  </select>
  <input type="button" value="Show status"
    onclick="showStatus(this.form);" />
</form>
```

Accès à une liste de sélection (selectionlist.html)

Info

Les navigateurs récents acceptent également le raccourci suivant pour accéder à la valeur de l'élément de liste actuellement sélectionné :

```
f.elements["selectionlist"].value
```

Toutefois, pour assurer la compatibilité maximale avec le navigateur, la technique suivante nécessite un peu plus de saisie, mais fonctionne également sur les navigateurs plus anciens :

```
f.elements["selectionlist"].options[
➥f.elements["selectionlist"].selectedIndex].value
```

Accès à une liste à choix multiple

```
s += "Element #" + i
  + " (" + option.text
  + "/" + option.value
  + ") " + (option.selected ? "selected." : "not selected."
  + "\n";
```

Lorsqu'une liste de sélection HTML récupère l'attribut `multiple="multiple"`, il est possible de sélectionner plusieurs éléments. Dans ce cas, la propriété `selectedIndex` n'est plus très utile car elle informe uniquement sur le *premier* élément sélectionné dans la liste. Il faut alors utiliser une boucle `for` pour parcourir *tous* les éléments de la liste. Lorsque la propriété `selected` vaut `true`, cela signifie que l'élément de liste est sélectionné.

Le code suivant donne des informations sur tous les éléments de liste et indique notamment s'ils sont sélectionnés :

```
<script language="JavaScript"
  type="text/JavaScript">
function showStatus(f) {
  var s = "";
```

```
  var list = f.elements["selectionlist"];
  for (var i=0; i<list.options.length; i++) {
    var option = list.options[i];
    s += "Element #" + i
      + " (" + option.text
      + "/" + option.value
      + ") " + (option.selected ?
      ➥"selected." : "not selected.")
      + "\n";
  }
  window.alert(s);
}
</script>
<form>
  <select name="selectionlist" size="3"
    multiple="multiple">
    <option value="R">red</option>
    <option value="G">green</option>
    <option value="B">blue</option>
  </select>
  <input type="button" value="Show status"
    onclick="showStatus(this.form);" />
</form>
```

Accès à une liste à choix multiples (selectionlistmultiple.html)

Figure 8.2, seuls quelques éléments sont sélectionnés.

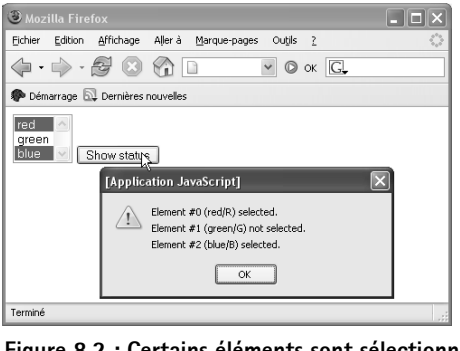

Figure 8.2 : Certains éléments sont sélectionnés.

JavaScript et l'envoi de fichiers

JavaScript permet aussi d'accéder aux éléments HTML pour l'envoi de fichiers (`<input type="file" />`). Toutefois, au cours de ces dernières années, les développeurs de navigateurs Web ont de plus en plus restreint l'accès JavaScript à ce type de champ de formulaire. Et il existe une bonne raison à cela. Imaginez un code JavaScript hostile qui définisse la valeur du champ de chargement de fichiers sur `/etc/passwd` par exemple, puis envoie automatiquement le formulaire. Cela n'est généralement plus possible. De même, certains navigateurs affichent un message dès que les fichiers sont chargés avec un formulaire (voir la Figure 8.3). Vous devez donc éviter de contrôler les champs d'envoi de formulaires avec JavaScript ; ils fonctionnent correctement avec le HTML de base et la technologie côté serveur.

Figure 8.3 : Konqueror vous avertit que des fichiers sont envoyés avec le formulaire.

Désactivation des éléments de formulaire

```
f.elements["password"].disabled = true;
```

Parfois, il est important que les éléments de formulaire ne puissent pas être modifiés. Les scénarios suivants n'en sont que quelques exemples :

- des éléments de formulaires permettant d'afficher des données, par exemple de longs contrats de licence emballés dans un champ `<textarea>` ;
- des éléments activés ou désactivés par interaction de l'utilisateur ;
- des données sous forme d'éléments "verrouillés", utilisées lorsque le formulaire est envoyé.

JavaScript propose plusieurs options pour parvenir à cet effet. Chaque élément de formulaire prend en charge la propriété `disabled`. Lorsqu'elle est définie sur `true`, l'élément devient grisé et ne peut plus être modifié. Le listing suivant présente un formulaire pour s'identifier et qui peut aussi être utilisé pour s'inscrire ; si l'utilisateur souhaite s'inscrire, on ne lui demande plus son mot de passe. Le champ du mot de passe peut donc être grisé lorsque le formulaire passe en mode inscription :

```
<script language="JavaScript"
  type="text/JavaScript">
function enable(f) {
  f.elements["password"].disabled = false;
}

function disable(f) {
  f.elements["password"].disabled = true;
```

```
}
</script>
<form>
  <input type="radio" name="radiobutton"
    value="login"checked="checked"
    onclick="enable(this.form);" />Login
  <input type="radio" name="radiobutton"
    value="register"
    onclick="disable(this.form);" />Register<br />
  Username <input type="text" name="username" />
  <br />
  Password <input type="password"
    name="password" /><br />
</form>
```

Désactivation des éléments de formulaire (disabled.html)

A la Figure 8.4, on peut constater les effets de ce code : le champ de texte est grisé lorsque le bouton radio Register est sélectionné.

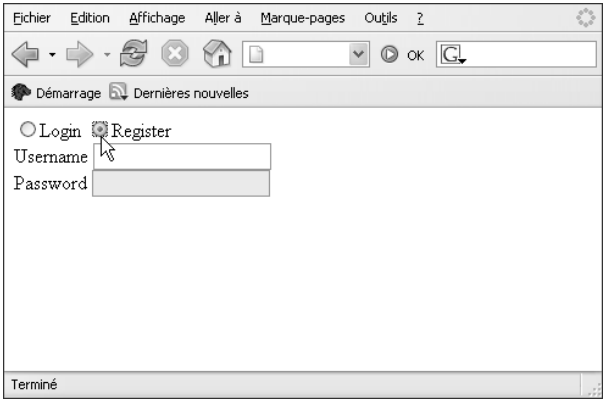

Figure 8.4 : Le champ de mot de passe est maintenant désactivé.

Il n'est parfois pas souhaitable qu'un champ soit grisé. Pour désactiver un élément de formulaire, on peut également utiliser sa propriété JavaScript `readOnly`. Lorsqu'elle est réglée sur `true`, l'apparence du champ de formulaire reste la même mais le champ ne peut plus être modifié. Voici le code qui l'implémente :

```
function enable(f) {
  f.elements["password"].readOnly = false;
}

function disable(f) {
  f.elements["password"].readOnly = true;
}
```

Création d'éléments de formulaire en lecture seule (readonly.html, extrait)

Info

Les anciens navigateurs faisaient appel à une autre astuce simple pour mettre les champs de texte en lecture seule. Une fois que le champ obtenait le focus, celui-ci disparaissait immédiatement :

```
<input type="text" onfocus="this.blur();" />
```

Attention

Vous pouvez utiliser JavaScript pour désactiver un champ de formulaire ou le mettre en lecture seule, mais sachez que JavaScript peut facilement être désactivé et qu'une requête HTTP peut également être contrefaite. Ne vous fiez donc pas à cet effet de JavaScript et validez toujours vos données côté serveur !

Envoi d'un formulaire

```
document.forms[0].submit();
```

Généralement, un bouton d'envoi (`<input type="submit" />`) ou une image d'envoi (`<input type="image" />`) permet d'envoyer les données du formulaire au serveur Web. JavaScript peut aussi employer sa méthode `submit()`. Cela permet au programmeur d'utiliser un lien HTML mais aussi d'augmenter la flexibilité pour le concepteur. N'oubliez pas que le code suivant ne fonctionne que lorsque JavaScript est activé :

```
<form>
  <input type="hidden" name="field" value="data" />
</form>
<a href="JavaScript:document.forms[0].submit();">
   Submit form</a>
```

Envoi d'un formulaire (submit.html)

Empêcher l'envoi

```
<form onsubmit="return checkform(this);">
```

Il existe de bonnes raisons d'empêcher le navigateur d'envoyer un formulaire, par exemple lorsque certains champs obligatoires n'ont pas été remplis. Pour ce faire, il faut renvoyer `false` dans le code de gestion de l'événement `submit` du formulaire :

```
<form onsubmit="return false;">
```

Bien entendu, le code doit décider, en fonction des données entrées, si le formulaire peut être envoyé. C'est généralement une fonction personnalisée qui s'en occupe et renvoie, au final, `true` ou `false`.

Le formulaire suivant ne peut être envoyé que s'il existe une valeur dans le champ de texte :

```
<script language="JavaScript"
  type="text/JavaScript">
function checkform(f) {
  if (f.elements["textfield"].value == "") {
    return false;
  } else {
    return true;
  }
}
</script>
<form onsubmit="return checkform(this);">
  Username <input type="text" name="textfield" />
  <input type="submit" value="Submit data" />
</form>
```

Le formulaire ne peut être envoyé que lorsque le champ de texte est rempli (nosubmit.html)

Une fois de plus, cela ne fonctionne que lorsque JavaScript est activé, raison de plus pour valider toutes les données côté serveur !

Eviter les envois répétés de formulaires

```
<form action="delay.php">
  <input type="submit" value="Submit data"
    onclick="this.disabled = true;"/>
</form>
```

Lorsque le script qui analyse les données de formulaire est long à s'exécuter, les utilisateurs ont tendance à essayer d'accélérer les choses en cliquant de nouveau sur le bouton

d'envoi. Or, notamment pendant des opérations d'achat en ligne, paiements par carte de crédit et autres transactions importantes, le résultat peut être assez onéreux. La page Web doit donc interdire d'elle-même les envois répétés de formulaires.

Le code qui précède (dans le fichier `prevent.html` des téléchargements de code de cet ouvrage) en est un exemple : lorsque l'utilisateur clique sur le bouton d'envoi, sa propriété `disabled` est réglée sur `true`, ce qui interdit de cliquer à nouveau sur le bouton (si JavaScript est activé). La Figure 8.5 en montre le résultat dans le navigateur. Le bouton est grisé.

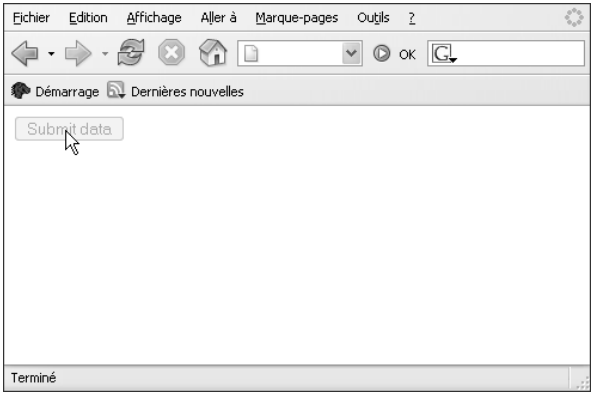

Figure 8.5 : Il est impossible de cliquer deux fois sur le bouton.

Le listing précédent et le listing suivant envoient tous deux les données de formulaire vers un script PHP appelé `delay.php`, lequel attend cinq secondes avant d'envoyer les données. Cela émule une connexion ou un serveur lent, le scénario adopté dans cette section.

Une autre manière consiste à maintenir le bouton actif, mais à déterminer si l'utilisateur a déjà cliqué dessus, comme le montre le listing suivant :

```
<script language="JavaScript"
  type="text/JavaScript">
var submitted = false;
function checkform(f) {
  if (submitted) {
    window.alert("Form has already been submitted!");
    return false;
  } else {
    submitted = true;
    return true;
  }
}
</script>
<form action="delay.php"
  onsubmit="return checkform(this);">
  <input type="submit" value="Submit data" />
</form>
```

Empêcher l'envoi répété d'un formulaire en maintenant le bouton d'envoi actif (prevent-status.html)

Attention

Même si ce procédé est assez commode, vous devez savoir que, parfois, le serveur ferme la connexion au client de manière inattendue (ou vice versa). Le transfert de données s'arrête alors mais le code JavaScript ne le sait pas. Conséquence, il est impossible de cliquer deux fois sur le bouton (comme prévu initialement) mais les données n'ont pas été correctement transmises. L'utilisateur doit donc recharger le formulaire pour pouvoir à nouveau envoyer les données.

Donner le focus à un champ

```
document.forms[0].elements["textfield"].focus();
```

La plupart des sites Web de recherche, comme Google, possèdent une fonction pratique : dès que la page est chargée, le curseur se place automatiquement dans le champ de recherche. L'opération est assez facile à réaliser grâce au code précédent ; la méthode `focus()` donne le focus au champ de formulaire. Le listing complet est disponible dans le fichier `focus.html`.

Attention

Sachez que plus votre page est importante, plus elle contient de champs de formulaire et plus cette fonction peut être inadaptée. Imaginez une page qui se charge lentement. Pendant ce temps, l'utilisateur peut avoir déjà avancé dans un autre champ. A la fin du chargement, ce code met le focus sur le premier champ, en l'ôtant du champ dans lequel se trouve actuellement l'utilisateur.

Sélection de texte dans un champ

```
field.setSelectionRange(0, field.value.length);
```

Les champs de texte sont parfois préremplis d'une mention expliquant à l'utilisateur ce qu'il doit faire, comme ceci :

```
<input type="text" name="textfield"
  value="&lt;Enter data here&gt;" />
```

C'est un peu gênant lorsque l'utilisateur tente de saisir quelque chose dans le champ : il doit d'abord sélectionner cette mention pour pouvoir l'écraser. Si le texte était présélectionné, l'opération serait plus facile.

Les principaux navigateurs acceptent la sélection d'un texte dans un champ, mais une autre façon d'opérer existe. Avec Internet Explorer, il faut appeler la méthode `createTextRange()` du champ, qui crée une plage de sélection. Les méthodes `moveStart()` et `moveEnd()` en définissent les limites ; la méthode `select()` sélectionne ensuite le texte.

Les navigateurs Mozilla d'une part et les navigateurs Opera et Konqueror/KDE d'autre part disposent d'une autre interface pour cette fonctionnalité. La méthode `setSelectionRange()` sélectionne le contenu d'un champ. Elle attend comme second paramètre la longueur de la plage, tandis que `moveEnd()` dans Internet Explorer attend la position du dernier caractère à sélectionner.

Le listing suivant sélectionne l'ensemble du texte dans les principaux navigateurs puis établit le focus sur le champ. Ainsi, si l'utilisateur commence à taper du texte, la mention de départ est immédiatement remplacée.

```
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  var field = document.forms[0] .elements["textfield"];
  if (field.createTextRange) {
    var range = field.createTextRange();
    range.moveStart("character", .0);
    range.moveEnd("character", field.value.length - 1);
    range.select();
  } else if (field.setSelectionRange) {
    field.setSelectionRange(0, field.value.length);
  }
  field.focus();
};
</script>
<form>
  <input type="text" name="textfield"
    value="&lt;Enter data here&gt;" />
</form>
```

Présélection de texte dans un champ (selecttext.html)

La Figure 8.6 montre le résultat. Lorsque ce champ de formulaire sera validé, la valeur par défaut ne sera pas considérée comme une donnée valable.

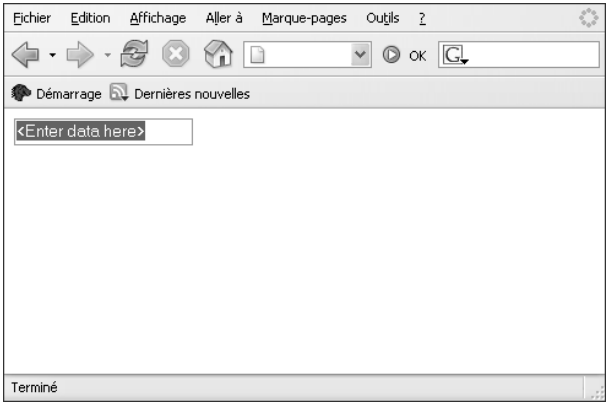

Figure 8.6 : Le texte est sectionné et le champ a le focus.

Astuce

Vous pouvez envisager de ne préremplir le champ que si JavaScript est disponible ; dans le cas contraire, les utilisateurs devront sélectionner manuellement (et supprimer) la mention préalable du champ de texte, ce que permet le code suivant :

```
field.value = "<Enter data here>";
```

Vider les champs de texte en cas de clic

```
if (this.value == originaldata) {
  this.value = "";
}
```

Une autre fonction utile de l'expression précédente consiste à effacer le champ de texte si l'utilisateur clique dedans (la présélection du texte est effacée dans ce cas). Toutefois, il ne suffit pas d'ajouter `onclick="this.value"='';"` à la balise `<input>`. Imaginez que l'utilisateur ait saisi des données personnalisées dans le champ, puis décidé de le quitter, le texte disparaîtrait… et l'utilisateur probablement aussi.

Il faut donc enregistrer le texte original lors du chargement de la page. Ainsi, lorsque l'utilisateur clique dans le champ de texte et que le texte qui s'y trouve est le texte original, le champ peut être vidé, comme le montre le listing suivant :

```
<script language="JavaScript"
  type="text/JavaScript">
var originaldata;

window.onload = function() {
  var field = document.forms[0] .elements["textfield"];
  originaldata = field.value;
  if (field.createTextRange) {
    var range = field.createTextRange();
    range.moveStart("character", .0);
    range.moveEnd("character", field.value.length - 1);
    range.select();
  } else if (field.setSelectionRange) {
    field.setSelectionRange(0, field.value.length);
  }
  field.focus();
  field.onclick = function() {
```

```
    if (this.value == originaldata) {
      this.value = "";
    }
  };
}
</script>
<form>
  <input type="text" name="textfield"
    value="&lt;Enter data here&gt;" />
</form>
```

Effacement du champ (clear.html)

Notez que l'événement `focus` ne peut pas être utilisé pour cette tâche ; Internet Explorer effacerait immédiatement le champ en cas de chargement du document (même si l'appel à `focus()` survient avant que le gestionnaire d'événements ne soit paramétré).

Validation des champs de texte

```
if (field.value.replace(/\s/g, "") == "") {
  window.alert("Invalid data!");
  field.focus();
```

La validation des données fournies par l'utilisateur peut prendre plusieurs aspects. Quelquefois, il est important de vérifier qu'un champ obligatoire a été rempli, à d'autres moments, il convient de procéder à une validation précise des données.

Le code précédent (fichier `mandatory.html`) vérifie les données d'un champ de texte. La méthode JavaScript `replace()` supprime l'espace blanc (expression régulière `/\s/g`) dans la chaîne puis vérifie s'il reste quelque chose, de sorte que les caractères d'espacement ne satisfassent pas à eux seuls à la condition "vider ce champ".

Validation de formulaires

JavaScript est un très bon outil pour valider les formulaires à la volée. Il permet de communiquer des commentaires immédiats aux utilisateurs sur les données saisies (par exemple, en utilisant un élément du style `onblur="validateData(this);"`), mais vous devez toujours envisager que JavaScript puisse être désactivé et les requêtes HTTP ne doivent pas provenir d'un navigateur Web. Ainsi, avant d'utiliser les données du client, validez-les également sur le serveur.

Vérifiez également que le formulaire lui-même fonctionne lorsque JavaScript est désactivé, puisque environ 10 % des utilisateurs ne peuvent pas utiliser ce langage de script (ou ne le veulent pas).

Enfin, la validation elle-même ne doit être qu'une fonction de commodité pour les utilisateurs et non un outil énervant, masquant les informations pour le visiteur. Utilisez-la donc avec mesure.

Dans le listing suivant, les données du champ de texte sont validées par rapport à une expression régulière identifiant un code postal américain :

```
<script language="JavaScript"
  type="text/JavaScript">
function validate(field) {
  var regex = /^\d{5}$/;
  if (!regex.test(field.value)) {
    window.alert("Invalid postal code!");
    field.value = "";
    field.focus();
  }
}
</script>
```

```
<form onsubmit="return checkform(this);">
  US postal code <input type="text" name="code"
                  onblur="validate(this);" />
</form>
```

Validation d'un champ par rapport à une expression régulière (mandatory-regex.html)

Validation de cases à cocher

```
if (!f.elements["terms"].checked) {
  window.alert("Terms &amp;amp; conditions must be accepted!");
  f.elements["terms"].focus();
  return false;
} else {
  return true;
}
```

Lorsqu'une case doit être cochée, sa propriété `checked` doit valoir `true`, faute de quoi le formulaire ne doit pas pouvoir être envoyé. Le code qui précède, disponible dans un listing complet (fichier `mandatory-checkbox.html`), montre le code qui pourrait être appelé au déclenchement de l'événement `submit` du formulaire.

Validation de boutons radio

```
if (radio.checked) {
  return true;
}
```

Lorsqu'un groupe de boutons radio est obligatoire, il suffit que *l'un* de ses boutons soit activé (en fait, il n'est pas possible d'activer plusieurs boutons par groupe, mais il est toujours possible qu'aucun ne le soit). Grâce au tableau `elements` du formulaire, le groupe de boutons radio peut

être sélectionné sous forme de tableau, dans lequel il est nécessaire qu'il y ait au moins un bouton radio dans lequel `checked` vaut `true` pour que le groupe soit considéré rempli. Le listing suivant implémente cette vérification :

```
<script language="JavaScript"
  type="text/JavaScript">
function checkform(f) {
  for (var i=0; i<f.elements["colors"].length; i++) {
    var radio = f.elements["colors"][i];
    if (radio.checked) {
      return true;
    }
  }
  // aucun bouton coché
  window.alert("No color selected!");
  f.elements["colors"][0].focus();
  return false;
}
</script>
<form onsubmit="return checkform(this);">
  <input type="radio" name="colors" value="R" />
     red<br />
  <input type="radio" name="colors" value="G" />
     green<br />
  <input type="radio" name="colors" value="B" />
     blue<br />
  <input type="submit" value="Submit data" />
</form>
```

Validation d'un groupe de boutons radio (mandatory-radio.html)

Info

Vous pouvez envisager une manière moins ostentatoire d'avertir l'utilisateur d'une erreur dans le formulaire. Vous pourriez par exemple utiliser DOM ou l'objet `Image` pour afficher une petite icône d'erreur près de tous les champs n'ayant pas été correctement remplis. Cela convient bien sûr pour toutes les expressions de validation de ce chapitre.

Validation des listes de sélection

```
if (f.elements["colors"].selectedIndex == -1) {
  window.alert("No color selected!");
  f.elements["colors"].focus();
  return false;
} else {
  return true;
}
```

Il existe plusieurs manières de valider une liste de sélection. La plus simple consiste à vérifier sa propriété `selectedIndex`. Si sa valeur est `-1`, cela signifie qu'aucun élément n'a été sélectionné et que le formulaire n'a donc pas été suffisamment rempli. Le code qui précède en est une illustration (version complète dans le fichier `mandatory-list.html`).

Cependant, cette méthode ne fonctionne que lorsque la liste de sélection possède un attribut `size` paramétré sur une valeur supérieure à 1 et ne contient pas d'élément vide (à des fins de démonstration). Mieux encore, vous pouvez rechercher les attributs `value` de tous les éléments sélectionnés. Si la valeur est une chaîne vide, l'élément n'est pas pris en compte. Une boucle `for` passe en revue tous les éléments de la liste à la recherche des éléments sélectionnés affichant une valeur réelle. Cette procédure possède aussi l'avantage de fonctionner avec plusieurs listes de sélection.

```
<script language="JavaScript"
  type="text/JavaScript">
function checkform(f) {
  for (var i=0; i<f.elements["colors"].options.length; i++) {
    var element = f.elements["colors"].options[i];
    if (element.selected && element.value != "") {
      return true;
    }
  }
  window.alert("No color selected!");
  f.elements["colors"].focus();
  return false;
}
</script>
<form onsubmit="return checkform(this);">
  <select name="colors" size="9"
    multiple="mul tiple">
    <option value="">Select a color</option>
    <option value="R">red</option>
    <option value="G">green</option>
    <option value="B">blue</option>
    <option value="">———-</option>
    <option value="C">cyan</option>
    <option value="M">magenta</option>
    <option value="Y">yellow</option>
    <option value="K">black</option>
  </select>
  <input type="submit" value="Submit data" />
</form>
```

Validation d'une liste de sélection (mandatory-list-loop.html)

Vous le voyez à la Figure 8.7, les deux éléments sélectionnés ne comptent pas.

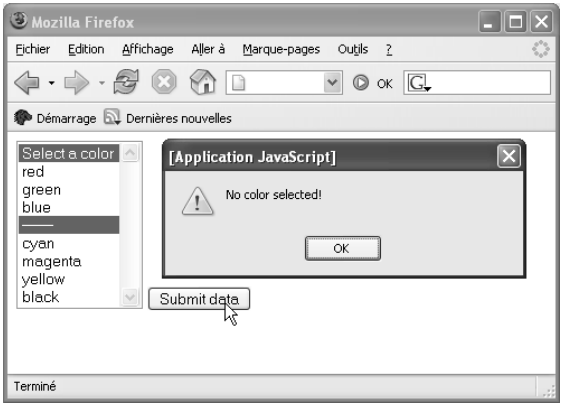

Figure 8.7 : Le formulaire ne peut être envoyé que si un élément "réel" a été sélectionné.

Validation automatique d'un formulaire

```
switch (element.type)
```

En intensifiant un peu ses efforts, le programmeur peut amener JavaScript à valider automatiquement un formulaire, à condition que tous ses champs soient obligatoires. Il suffit alors de les passer tous en revue. Chaque propriété `type` de l'élément renvoie le type du champ ; le Tableau 8.1 montre une liste des plus importants. Ainsi, la validation survient en fonction du type de champ.

Tableau 8.1 : Types de champs de formulaires de JavaScript

Type	Description
`button`	Bouton HTML
`checkbox`	Case à cocher
`password`	Champ de mot de passe
`radio`	Bouton radio
`reset`	Bouton de réinitialisation
`select-one`	Liste de sélection
`select-multiple`	Liste à choix multiples
`submit`	Bouton d'envoi
`text`	Champ de texte sur une seule ligne
`textarea`	Champ de texte sur plusieurs lignes

Le code suivant traite tout particulièrement les groupes de boutons radio. Puisque la boucle qui parcourt les éléments de formulaire étudie chacun d'entre eux, c'est-à-dire chaque bouton radio, elle conserve une liste des groupes de boutons déjà cochés. Si ce n'était pas le cas, un groupe de trois boutons radio serait coché trois fois, et trois messages d'erreur seraient générés si aucun bouton n'était sélectionné.

```
<script language="JavaScript"
  type="text/JavaScript">
function checkform(f) {
  var errortext = "";
  var checkedgroups = "";
  for (var i=0; i<f.elements.length; i++) {
    var element = f.elements[i];
    switch (element.type) {
      case "text":
      case "textarea":
      case "password":
        if (element.value.replace(/\s/g, "") == "") {
          errortext += element.name + "\n";
        }
        break;
      case "checkbox":
        if (!element.checked) {
          errortext += element.name + "\n";
        }
        break;
      case "radio":
        var group = f.elements[element.name];
        if (checkedgroups.indexOf("[" +
          element.name + "]") > -1) {
          continue;
        } else {
          checkedgroups += "[" + element.name + "]";
        }
        var groupok = false;
        for (var j=0; j<group.length; j++) {
          if (group[j].checked) {
            groupok = true;
          }
        }
        if (!groupok) {
          errortext += element.name + "\n";
        }
```

```
        break;
      case "select-one":
      case "select-multiple":
        var selectok = false;
        for (var j=0; j<element.options.length;
          j++) {
          var item = element.options[j];
          if (item.selected && item.value != "") {
            selectok = true;
          }
        }
        if (!selectok) {
          errortext += element.name + "\n";
        }
        break;
    }
  }
  if (errortext == "") {
    return true;
  } else {
    window.alert(
      "The following fields have not been correctly
        ➥filled out:\n\n"
      + errortext);
    return false;
  }
}
</script>
```

Validation automatique d'un formulaire (validate.html, extrait)

La Figure 8.8 présente le résultat : aucun des champs du formulaire ne contient de contenu approprié. Notez cependant que vous pouvez modifier ce script et peut-être exclure les cases à cocher de la liste, puisque ces éléments de formulaire sont, la plupart du temps, optionnels.

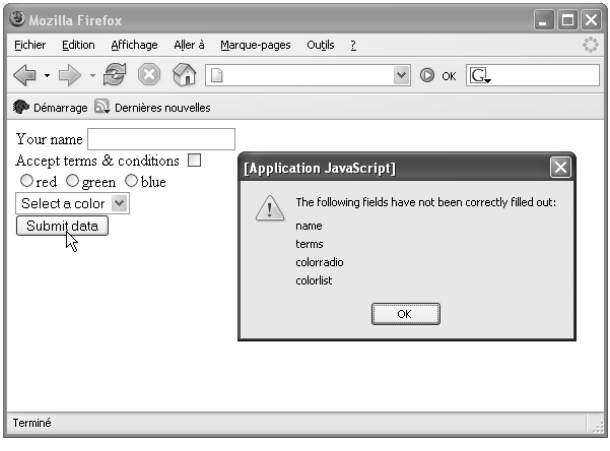

Figure 8.8 : Le formulaire est automatiquement validé.

Implémentation de la navigation avec une liste de sélection

```
var url = list.options[list.selectedIndex].value;
```

Une navigation dite de "liaison rapide" (*quick link*) présente une liste de sélection contenant deux éléments de navigation. Lorsque l'utilisateur en sélectionne un, le navigateur charge la page associée.

En matière d'implémentation, cet effet plaisant et très souvent utilisé est assez dérisoire. Les URL de destination de tous les éléments de navigation sont stockées dans les attributs `value` ; une fonction JavaScript vérifie

d'abord si un élément "réel" a été sélectionné, puis charge l'URL :

```
<script language="JavaScript"
  type="text/JavaScript">
function loadURL(list) {
  var url = list.options[list.selectedIndex].value;
  if (url != "") {
    location.href = url;
  }
}
</script>
<form>
  <select name="urls" onchange="loadURL(this);">
    <option value="">Please select...</option>
    <option value="http://www.samspublishing.com/">
       SAMS</option>
    <option value="http://www.hauser-wenz.de/blog/">
       H&amp;amp;W blog</option>
    <option value="http://www.damonjordan.com/">
       Damon</option>
  </select>
</form>
```

Une navigation par liaison rapide (quicklink.html)

Implémentation d'une navigation hiérarchique avec une liste de sélection

```
for (var i=0; i<links.length; i++) {
  elements["urls"].options[
    elements["urls"].options.length] =
      new Option(links[i].title, links[i].url);
}
```

Deux listes de sélection peuvent être appariées afin de fournir une navigation hiérarchique sur un site. La première

contient plusieurs catégories ; lorsque l'utilisateur en sélectionne une, la seconde liste affiche l'ensemble de ses éléments.

Pour JavaScript, il convient de stocker la liste des éléments ; d'autre part, les listes doivent être remplies en fonction de la première sélection. Voici la liste de liens, au format JSON (voir le Chapitre 11, "AJAX et sujets annexes", pour en savoir plus sur cette notation).

```
var data = {
  "search" : [
    {"title": "Google", "url":"http://www.google.com/"},
    {"title": "MSN", "url":"http://search.msn.com/"}
  ],
  "browsers" : [
   {"title": "Firefox", "url": "http://www.mozilla.com/
     ➥firefox/"},
    {"title": "Opera", "url": "http://www.opera.com/"}
  ]
};
```

Ensuite, le code suivant ajoute des options à la seconde liste de sélection, en employant le constructeur `Option` (syntaxe : d'abord la légende, ensuite la valeur) :

```
<script language="JavaScript"
  type="text/JavaScript">
function loadElements(f) {
  with (f) {
    var category = elements["categories"].options[
      elements["categories"].selectedIndex].value;
    if (category != "") {
      var links = data[category];
      elements["urls"].options.length = 0;
      elements["urls"].options[0] =
      ➥new Option("Please select...", "#");
      for (var i=0; i<links.length; i++) {
        elements["urls"].options[
```

```
        ➥elements["urls"].options.length] =
        ➥new Option(links[i].title, links[i].url);
      }
    }
  }
}
</script>
<form>
  <select name="categories"
    onchange="loadElements(this.form);">
    <option value="">Please select...</option>
    <option value="search">Search Engines</option>
    <option value="browsers">Browsers</option>
  </select>
  <select name="urls" onchange="loadURL(this);">
  </select>
</form>
```

Une navigation hiérarchique (navigation.html, extrait)

La Figure 8.9 montre ce qui survient lorsque la première catégorie ("Search Engines", des moteurs de recherche) est sélectionnée. La seconde liste se remplit avec les options disponibles.

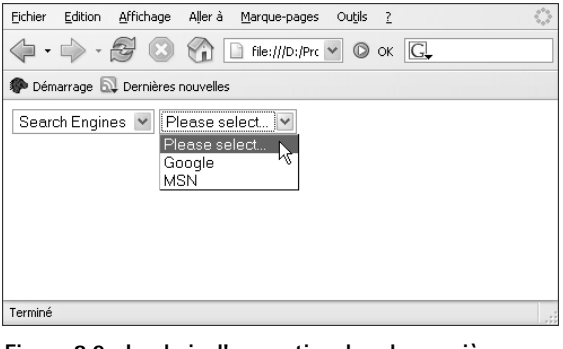

Figure 8.9 : Le choix d'une option dans la première liste conditionne le contenu de la seconde.

Désélection d'un ensemble de boutons radio

```
for (var i=0; i<f.elements["colors"].length; i++) {
  f.elements["colors"][i].checked = false;
}
```

Les boutons radio présentent un inconvénient souvent cité : lorsque l'utilisateur clique sur bouton d'un groupe, il ne peut annuler cette sélection, un bouton au moins restant toujours sélectionné. Avec un peu de JavaScript et un bouton d'annulation, cet effet peut être mitigé. Le code précédent passe en revue le groupe de boutons radio et définit la propriété `checked` de tous les éléments sur `false` (code complet dans le fichier `emptyradio.html`).

Astuce

Ce problème peut aussi être résolu par du HTML pur : proposez simplement un bouton radio (sélectionné par défaut) qui indique "Aucune sélection", de la manière suivante :

```
<input type="radio" name="answer" value="Y" />yes
<input type="radio" name="answer" value="N" />no
<input type="radio" name="answer" value=""
  checked="checked" />no answer
```

Bien entendu, vous devez ensuite modifier le code de validation.

Création de listes de sélection de date préremplies

```
elements["day"].selectedIndex = d.getDate() - 1;
```

La plupart des sites de réservation possèdent un ensemble de trois listes de sélection permettant aux utilisateurs de

saisir une date : une liste pour le jour, une pour le mois et une pour l'année. La génération de ces listes, que ce soit en HTML statique ou côté serveur, est une tâche plutôt simple, mais leur présélection exige un moyen côté serveur (voir le *Guide de Survie PHP et MySQL* pour obtenir une solution en PHP) ou un peu de JavaScript. L'idée consiste à définir la propriété `selectedIndex` de chaque liste sur la valeur de date appropriée, comme indiqué dans le listing suivant (les listes de sélection sont bien entendu abrégées) :

```
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  with (document.forms[0]) {
    var d = new Date();
    elements["day"].selectedIndex = d.getDate() - 1;
    elements["month"].selectedIndex = d.getMonth();
    elements["year"].selectedIndex = 2006 -
    ➥d.getFullYear();
  }
}
</script>
<form>
  <select name="day">
    <option value="1">1</option>
    <!— ... —>
  </select>
  <select name="month">
    <option value="1">January</option>
    <!— ... —>
  </select>
  <select name="year">
    <option value="2006">2006</option>
    <!— ... —>
  </select>
</form>
```

Préremplissage d'une liste de dates (autodate.html, extrait)

Attention

Dans l'exemple, la première année de la liste est 2006 ; cette valeur est également utilisée pour calculer la valeur `selectedIndex` obligatoire pour cette date. Si vous la modifiez, par exemple que vous ajoutez l'année 2007, vous devrez changer la formule pour `elements["year"].selectedIndex` en conséquence.

Création de listes de sélection de date de validation

```
f.elements["day"].options.length = maxDays;
```

La prochaine étape, pour les listes de sélection de dates et en partant de la section précédente, consiste à s'occuper des utilisateurs qui saisissent une date valide. Ainsi, dès qu'il y a changement du mois ou de l'année, la liste de sélection du jour doit être actualisée en conséquence :

```
<select name="month"
  onchange="updateDay(this.form);">
<select name="year"
  onchange="updateDay(this.form);">
```

Pour ce faire, une fonction d'aide (voir le Chapitre 6) détermine si l'année est une année bissextile :

```
function isLeapYear(y) {
  return (y % 4 == 0 && (y % 100 != 0 || y % 400 == 0));
}
```

Grâce à ces informations, la liste de sélection du jour peut être actualisée. S'il y a trop d'entrées, la liste est raccourcie. S'il y a trop peu d'entrées, la liste reçoit des entrées supplémentaires. Tout cela permet de maintenir

la valeur `selectedIndex`, si possible, de sorte que la sélection de l'utilisateur soit maintenue (à moins, par exemple, que le jour `30` ne soit sélectionné et que le mois ne passe à `février`).

```
<script language="JavaScript"
  type="text/JavaScript">
function updateDay(f) {
  var oldDays = f.elements["day"].options.length;
  var month = parseInt(f.elements["month"].value);
  var year = parseInt(f.elements["year"].value);

  var maxDays = 30;

  switch (month) {
    case 1: case 3: case 5: case 7: case 9: case 11:
      maxDays = 31;
      break;
    case 2:
      maxDays = (isLeapYear(year) ? 29 : 28);
      break;
  }

  f.elements["day"].options.length = maxDays;
  if (maxDays > oldDays) {
    for (var i=oldDays; i<maxDays; i++) {
      f.elements["day"].options[i] = new Option(i+1, i+1);
    }
  }
}
</script>
```

Mise à jour automatique de la liste (date.html, extrait)

A la Figure 8.10, vous voyez le résultat : le mois de février de l'année 2000 possède 29 jours.

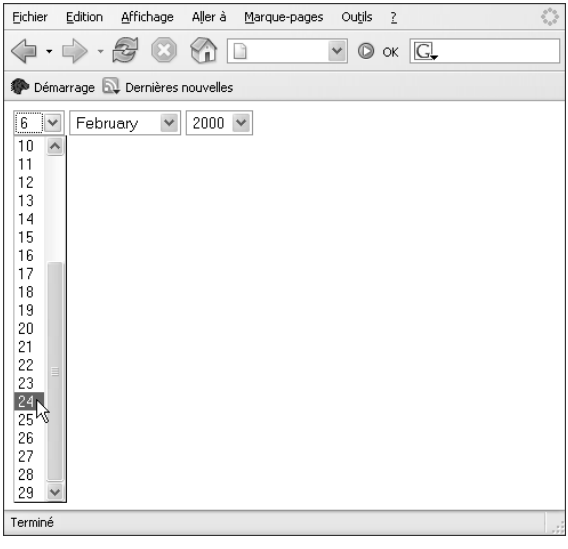

Figure 8.10 : La liste des jours est actualisée en fonction du mois et de l'année.

9

Fenêtres et cadres

Les fenêtres jouent un rôle important dans JavaScript. L'objet `window` est l'objet par défaut et il est utilisé assez souvent (pensez simplement à la méthode `windows.alert()`). De plus, JavaScript peut fonctionner avec les fenêtres (par exemple en en ouvrant de nouvelles) ainsi qu'avec les cadres et les cadres intégrés, qui sont également très proches des objets `window`.

```
window.open("http://www.samspublishing.com/",
          ➥"samsWindow", "");
```

La méthode `window.open()` ouvre une nouvelle fenêtre et prend au moins trois paramètres pour que l'appel soit utile : l'URL à ouvrir, le nouveau nom de la fenêtre et ses options.

Avec le code précédent (fichier `open.html`), une nouvelle fenêtre s'ouvre, affichant la page d'accueil de l'éditeur.

Le deuxième paramètre de la méthode `window.open()` fournit le nom de la nouvelle fenêtre. Cela fonctionne exactement comme un nom de cadre : si vous utilisez le nom de fenêtre comme attribut cible d'un lien, le lien

s'ouvre dans la nouvelle fenêtre. Ainsi, le fichier d'exemple `open.html` contient également un lien qui utilise le nouveau nom de la fenêtre :

```
<a href="http://www.amazon.com/gp/product/0672328801"
   target="samsWindow">This book at Amazon</a>
```

Lorsque vous cliquez sur ce lien, aucune nouvelle fenêtre n'est ouverte mais l'ancienne, précédemment ouverte par JavaScript, est réutilisée.

Sachez que la valeur de retour de `window.open()` est un objet de fenêtre JavaScript qui pointe vers la nouvelle fenêtre. Ainsi, un code ressemblant à ceci ne donne pas le résultat souhaité :

```
<a href="JavaScript:window.open(...)">Open window</a>
```

Puisque la méthode renvoie un objet fenêtre, ce lien écrirait sur la page un élément du type `[object]` (Internet Explorer) ou `[object Window]` (navigateurs Mozilla). Vous devez donc vous assurer que le code JavaScript ne renvoie rien. Pour cela, utilisez le code suivant :

```
<a href="JavaScript:void(window.open(...))">Open
   window</a>
```

Options de fenêtres

```
window.open("http://www.samspublishing.com/",
     "samsWindow",
  "width=640,height=480,directories=no,menubar=no,
    ➥toolbar=no,scrollbars=yes");
```

Le troisième paramètre de `window.open()` est une liste de valeurs, séparées par des virgules, qui définissent l'aspect de la nouvelle fenêtre. Vous pouvez définir la taille (à condition que la largeur et la hauteur soient au moins de 100 pixels), basculer certaines fonctions de la fenêtre

comme la barre d'état, etc. Le Tableau 9.1 montre les options les plus importantes (la liste n'est pas complète, elle laisse plusieurs options au navigateur).

Le code précédent utilise quelques-unes de ces options. Certaines prennent des valeurs numériques, d'autres `yes` ou `no`. Veillez à ne pas insérer d'espaces entre elles.

Tableau 9.1 : Options de fenêtre

Option	Description
`dependent`	Ferme la nouvelle fenêtre si la fenêtre parent est fermée
`directories`	Bascule la barre d'outils personnelle
`height`	Hauteur de la fenêtre
`innerHeight`	Hauteur de la fenêtre, en excluant sa décoration (navigateurs sauf IE)
`innerWidth`	Largeur de la fenêtre, en excluant sa décoration (navigateurs sauf IE)
`left`	Position horizontale de la fenêtre (IE uniquement)
`location`	Bascule la barre d'emplacement
`menubar`	Bascule la barre de menus
`outerHeight`	Hauteur de la fenêtre (navigateurs sauf IE)
`outerWidth`	Largeur de la fenêtre (navigateurs sauf IE)
`resizeable`	Lorsque la fenêtre peut être redimensionnée par l'utilisateur

Tableau 9.1 : Options de fenêtre *(suite)*

Option	Description
`screenX`	Position horizontale de la fenêtre (navigateurs sauf IE)
`screenY`	Position verticale de la fenêtre (navigateurs sauf IE)
`scrollbars`	Bascule les barres de défilement
`status`	Bascule la barre d'état
`toolbar`	Bascule la barre d'outils
`top`	Position verticale de la fenêtre (IE uniquement)
`width`	Largeur de la fenêtre
`z-lock`	Donne à la fenêtre une valeur z-lock (position de l'axe des z) inférieure

Attention

Tous ces effets ne fonctionneront pas dans tous les navigateurs de la même manière. Par exemple, le navigateur Opera ouvre une nouvelle fenêtre, non pas dans une nouvelle fenêtre de navigateur mais, selon les versions, dans un nouvel onglet qui, bien entendu, possède les ornements et la taille de fenêtre par défaut.

La Figure 9.1 montre l'aspect de la fenêtre nouvellement ouverte (fichier `openoptions.html`) avec les options données.

Figure 9.1 : La nouvelle fenêtre n'a que très peu d'ornements.

Ouverture d'une fenêtre modale

```
window.showModalDialog(
  "http://www.samspublishing.com/","samsWindow",
  "dialogWidth=640,dialogHeight=480,status=no,center=yes");
```

Les deux principaux navigateurs acceptent certaines extensions propriétaires de l'objet `window` mais, pour des raisons évidentes, nous ne les traiterons pas ici. Il existe toutefois une exception : Internet Explorer peut créer des fenêtres modales, une chose assez commune dans les applications d'intranet où le type de navigateur peut être choisi par le personnel informatique.

La méthode associée s'appelle `window.showModalDialog()`, elle prend une URL, un nom et un jeu d'options (que vous trouverez au Tableau 9.2).

Le listing suivant ouvre cette boîte de dialogue, la centre et lui attribue une taille de 640 × 480 pixels.

```
<script language="JavaScript"
  type="text/JavaScript">
if (window.showModalDialog) {
  window.showModalDialog(
    "http://www.samspublishing.com/",
    "samsWindow",
    "dialogWidth=640,dialogHeight=480,status=no,
     ➥center=yes");
}
</script>
<a href="http://www.amazon.com/gp/product/
   0672328801" target="samsWindow">This
   book at Amazon</a>
```

Ouverture d'une fenêtre modale (modal.html)

Tableau 9.2 : Options de fenêtre

Option	Description
`center`	Centre la fenêtre
`dialogHeight`	Hauteur de la fenêtre
`dialogLeft`	Position horizontale de la fenêtre
`dialogTop`	Position verticale de la fenêtre
`dialogWidth`	Largeur de la fenêtre
`help`	Bascule le symbole d'aide dans le coin supérieur droit
`status`	Bascule la barre d'état
`resizeable`	Lorsque la fenêtre peut être redimensionnée par l'utilisateur

La Figure 9.2 montre la fenêtre modale produite par le code précédent.

Figure 9.2 : La fenêtre modale (Internet Explorer uniquement).

Détermination de la taille de l'écran

```
window.alert(screen.availWidth + " x " +
             screen.availHeight + " pixels");
```

Dans une situation idéale, toutes les mises en page Web seraient flexibles et s'adapteraient à la solution disponible. Malheureusement, ce n'est pas le cas mais, grâce à JavaScript, vous pouvez vous adapter.

L'objet `screen` (`window.screen` dans les navigateurs récents, `screen` dans les anciens) propose des informations sur la résolution d'écran actuelle. Les propriétés qui vous intéresseront sont `availableHeight` et `availableWidth`, qui indiquent la hauteur et la largeur disponibles. Cela peut, par exemple, être utilisé pour déterminer la résolution actuelle et rediriger l'utilisateur vers une page adaptée à ces dimensions d'écran.

Le code précédent (fichier `screen.html`) détermine l'espace actuellement disponible à l'écran. La Figure 9.3 montre le résultat lorsqu'une solution de 1 024 × 768 pixels est adoptée.

Figure 9.3 : Les dimensions d'écran peuvent ne pas être celles que vous attendez.

Info

Sachez que, dans cet exemple spécifique, il manque 30 pixels ; ils "appartiennent" à la barre Microsoft Windows (on obtient des effets similaires lorsque cette séquence de code est exécutée sur d'autres systèmes avec des barres système). Si vous souhaitez connaître la largeur et la hauteur réelles, utilisez les propriétés `width` et `height` de l'objet `screen`.

Détermination de la taille de la fenêtre

```
with (document.body) {
  window.alert(clientWidth + " x " +
               clientHeight + " pixels");
}
```

L'accès aux propriétés de la fenêtre en cours dépend grandement du navigateur utilisé. Comme c'est souvent le cas avec JavaScript, le monde des navigateurs est divisé en deux : Internet Explorer et les autres. Avec Internet

Explorer, la largeur d'une fenêtre peut être déterminée par `document.body.clientWidth` ; les autres navigateurs utilisent `window.innerWidth` pour la largeur "interne" (qui exclut les ornements de la fenêtre) et `window.outerWidth` pour la largeur "externe" (qui les inclut). Pour déterminer la hauteur, il suffit de remplacer respectivement `width` et `height` par `Width` et `Height`.

Le code suivant produit les dimensions de la fenêtre en cours :

```
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  if (window.innerWidth) {
    window.alert("inner: " + window.innerWidth +
      " x " + window.innerHeight +
      " pixels\nouter: " + window.outerWidth +
      " x " + window.outerHeight + " pixels");
  } else {
    with (document.body) {
      window.alert(clientWidth + " x " +
        clientHeight + " pixels");
    }
  }
}
</script>
<body></body>
```

Détermination des dimensions de la fenêtre actuelle (dimensions.html)

Astuce

Pour que le code fonctionne dans Internet Explorer, sachez que la page a besoin d'une zone `<body>` et que le code ne doit pas être exécuté avant que la page n'ait été totalement chargée.

Redimensionnement d'une fenêtre

```
window.resizeTo(640, 480);
```

Il existe plusieurs manières de définir la taille d'une fenêtre. Lors de son ouverture à l'aide de `window.open()`, il suffit de paramétrer les options `width` et `height`. Pour les fenêtres existantes, JavaScript propose deux méthodes de l'objet `window` :

- `resizeTo()`. Définit la largeur et la hauteur de la fenêtre sur les valeurs fournies, comme dans le code précédent (fichier `resizeto.html` dans l'archive de téléchargement).
- `resizeBy()`. Modifie la largeur et la hauteur de la fenêtre en ajoutant les valeurs fournies, comme le montre le listing suivant (qui réduit de moitié la largeur et la hauteur) :

```
<script language="JavaScript"
  type="text/JavaScript">window.onload = function() {
  var width, height;
  if (window.innerWidth) {
    width = window.outerWidth;
    height = window.outerHeight;
  } else {
    width = document.body.clientWidth;
    height = document.body.clientHeight;
  }
  window.resizeBy(
    -Math.round(width / 2),
    -Math.round(height / 2));
}
</script>
<body></body>
```

Bissection d'une largeur et d'une hauteur de fenêtre (resizeby.html)

Attention

Le redimensionnement d'une fenêtre est généralement considéré peu convivial. L'un des principes du Web stipule que la taille du client n'est pas pertinente et qu'elle peut être ajustée par l'utilisateur. Utilisez donc la technique de cette section avec parcimonie. Sachez également que vous ne pouvez pas redimensionner la fenêtre sur une largeur ou une hauteur inférieure à 100 pixels.

Repositionnement d'une fenêtre

```
window.moveBy(-10, 10);
```

Lors de l'ouverture d'une nouvelle fenêtre, vous devez tenir compte de tous les navigateurs : pour Internet Explorer, paramétrez les options `left` et `top` ; tous les autres navigateurs fonctionnent avec `screenX` et `screenY`. L'appel suivant de `window.open()` crée une fenêtre dans le coin supérieur gauche de l'écran.

```
window.open(
  "anypage.html",
  "name",
  "left=0,top=0,screenX=0,screenY=0");
```

Toutefois, pour les fenêtres existantes, deux méthodes, assez analogues à la section précédente, entrent en jeu :

- `moveTo()`. Positionne la fenêtre à l'endroit spécifié, comme le montre le code du début de cette section (dans le fichier `moveto.html`).
- `moveBy()`. Déplace la fenêtre. Le listing suivant montre la fenêtre déplacée de 10 pixels à gauche et de 10 pixels vers le bas.

```
<script language="JavaScript"
  type="text/JavaScript">
window.moveBy(-10, 10);
</script>
```

Déplacement d'une fenêtre (moveby.html)

Attention

Le déplacement d'une fenêtre est presque aussi peu convivial que le redimensionnement. Une fois de plus, utilisez cette fonction sans excès. Sachez également qu'il est impossible de déplacer la fenêtre de manière à la faire totalement disparaître pour l'utilisateur, pour des raisons de sécurité évidentes.

Ouverture d'une fenêtre contextuelle centrée

```
newwin.moveTo(
  Math.round((screenwidth - windowwidth) / 2),
  Math.round((screenheight - windowheight) / 2));
```

La combinaison de ces expressions permet d'ouvrir une fenêtre, de déterminer ses dimensions ; son repositionnement permet de créer une nouvelle fenêtre centrée sur la page.

L'idée est de calculer la largeur et la hauteur de la nouvelle fenêtre (n'oubliez pas que `window.open()` renvoie une référence à la nouvelle fenêtre) et de déterminer la taille de l'écran ; à partir de là, il suffit de calculer la position requise de la fenêtre, de manière qu'elle soit centrée.

Toutefois, le délai est crucial ici. Sous Internet Explorer, vous pouvez accéder aux dimensions de la fenêtre uniquement après que l'élément `<body>` a été analysé.

On utilise donc une petite astuce : tout d'abord, la nouvelle fenêtre est vide. Puis, `document.write()` crée un élément. La largeur et la hauteur de la fenêtre peuvent alors être lues. Enfin, le document souhaité est chargé dans la nouvelle fenêtre, qui est ensuite repositionnée au centre de l'écran.

```
<script language="JavaScript"
  type="text/JavaScript">
var newwin = window.open("", "samsWindow", "");
newwin.document.write("<body></body>");

var windowwidth, windowheight;
if (window.innerWidth) {
  windowwidth = newwin.outerWidth;
  windowheight = newwin.outerHeight;
} else {
  windowwidth = newwin.document.body.clientWidth;
  windowheight = newwin.document.body.clientHeight;
}
var screenwidth = screen.availWidth;
var screenheight = screen.availHeight;
newwin.moveTo(
  Math.round((screenwidth - windowwidth) / 2),
  Math.round((screenheight - windowheight) / 2));
newwin.location.href =
  "http://www.samspublishing.com/";
</script>
<body></body>
```

Ouverture et centrage d'une fenêtre (center.html)

Info

Parfois, la lecture des propriétés de fenêtre ne fonctionne pas comme prévu dans Internet Explorer et produit des résultats erronés. Dans ce cas, il vaut mieux paramétrer la taille de la nouvelle fenêtre dans l'appel `window.open()`, puis utiliser cette valeur (que l'on sait être vraie) pour centrer la nouvelle fenêtre.

Ouverture d'une fenêtre en plein écran

```
newwin.resizeTo(screen.width, screen.height);
newwin.moveTo(0, 0);
```

Internet Explorer 4 accepte l'option de `window.open()` nommée `fullscreen=yes`, mais une méthode plus générale consiste bien sûr à extraire l'objet de l'écran pour obtenir les dimensions du système, puis à redimensionner la fenêtre en conséquence. N'oubliez pas de la repositionner dans le coin supérieur gauche de l'écran !

```
<script language="JavaScript"
  type="text/JavaScript">
var newwin = window.open(
  "http://www.samspublishing.com/",
  "samsWindow",
  "directories=no,menubar=no,toolbar=no, scrollbars=yes");
newwin.resizeTo(screen.width, screen.height);
newwin.moveTo(0, 0);
</script>
```

Ouverture d'une fenêtre en plein écran (center.hml)

Ouverture d'une nouvelle fenêtre dans un coin de l'écran

```
newwin.moveTo(
  screenwidth - windowwidth,
  screenheight - windowheight);
```

L'ouverture d'une fenêtre dans le coin supérieur gauche de l'écran est assez simple : il suffit de donner la valeur `0` aux positions horizontale et verticale.

```
<script language="JavaScript"
  type="text/JavaScript">
var newwin = window.open(
  "http://www.samspublishing.com/",
  "samsWindow",
  "width=640,height=480,directories=no,menubar=no,
    ➥toolbar=no,scrollbars=yes,left=0,top=0,screenX=0,
    ➥screenY=0");
</script>
```

Ouverture d'une fenêtre dans le coin supérieur gauche (open_topleft.html)

Tous les autres coins de l'écran sont un peu plus difficiles à obtenir car vous devez en calculer les positions. Grâce à l'objet `screen`, ce dilemme peut être résolu avec une petite quantité de code. L'extrait suivant place la nouvelle fenêtre dans le coin inférieur droit ; les listings pour les deux autres coins sont disponibles dans les fichiers `open_topright.html` et `open_bottomleft.html`.

```
<script language="JavaScript"
  type="text/JavaScript">
var newwin = window.open(
  "http://www.samspublishing.com/",
  "samsWindow",
  "width=640,height=480,directories=no,menubar=no,
    ➥toolbar=no,scrollbars=yes");
var windowwidth = 640;
var windowheight = 480;
var screenwidth = screen.availWidth;
var screenheight = screen.availHeight;
newwin.moveTo(
  screenwidth - windowwidth,
  screenheight - windowheight);
</script>
```

Ouverture d'une fenêtre dans le coin inférieur droit (open_bottomright.html)

Création d'une carte de site

```
this.opener.name = "mainWindow";
```

Une carte est un élément très pratique, quel que soit le site Web. Avec un peu de code JavaScript, cette fonction peut encore s'améliorer.

Si vous ouvrez une carte de site dans une nouvelle fenêtre, tous ses liens doivent également s'ouvrir dans la fenêtre initiale. A l'évidence, vous pouvez pour cela paramétrer l'attribut `target` de tous les liens HTML (ou utiliser `<base target="..." />`). Toutefois, une question demeure : quel est le nom de la fenêtre ? Pour répondre à cette question, on peut utiliser JavaScript dans la fenêtre principale et paramétrer la propriété `name` de la fenêtre :

```
this.name = "mainWindow";
```

Une autre manière consiste à utiliser la carte du site et à y référencer la fenêtre d'ouverture. Cela se fait grâce à la propriété `opener` de la fenêtre, paramétrée sur la fenêtre parente de la fenêtre actuelle, le cas échéant. Le code suivant correspond à la carte du site complète :

```
<script language="JavaScript"
  type="text/JavaScript">
this.opener.name = "mainWindow";
</script>
<a href="http://www.samspublishing.com/"
  target="mainWindow">Publisher</a><br />
<a href="http://www.amazon.com/gp/product/
   0672328801" target="mainWindow">This
   book at Amazon</a>
```

Ouverture de la carte du site (sitemap.html, extrait)

Fermeture d'une fenêtre

```
if (!newwindow.closed) newwindow.close();
```

La fermeture d'une fenêtre peut passer par un appel à sa méthode `close()`. Or, cela n'est possible par défaut que pour les fenêtres qui ont été ouvertes à l'aide de JavaScript. Si vous utilisez la méthode sur d'autres fenêtres, l'utilisateur obtient un message d'avertissement analogue à celui affiché à la Figure 9.4 (les navigateurs Mozilla possèdent un message similaire, mais le masquent dans la console JavaScript).

Figure 9.4 : Fermer uniquement les fenêtres ouvertes avec JavaScript ou se retrouver face à cet effrayant message.

Avec la carte du site de la section précédente, on peut utiliser un lien pour fermer la fenêtre :

```
<script language="JavaScript"
  type="text/JavaScript">
this.opener.name = "mainWindow";
</script>
<a href="http://www.samspublishing.com/"
  target="mainWindow">Publisher</a><br />
<a href="http://www.amazon.com/gp/product/
   0672328801" target="mainWindow">This book
   at Amazon</a><br />
<a href="JavaScript:self.close();">Close window</a>
```

La carte du site avec un lien de fermeture (sitemapwindow.html)

Il est également conseillé de fermer la carte du site à partir de la fenêtre principale, si une autre page y est chargée. Dans l'exemple actuel, le chargement de toute autre page dans la fenêtre principale redirige le navigateur vers un autre serveur Web ; vous pouvez donc fermer immédiatement la carte du site. Mais avant cela, vérifiez si la fenêtre est déjà fermée, en étudiant sa propriété `closed` :

```
<script language="JavaScript"
  type="text/JavaScript">
var newwindow = window.open(
  "sitemapwindow.html",
  "sitemap",
  "width=320,height=100,directories=no,menubar=no,
    ➥toolbar=no,scrollbars=yes");
</script>
<body onunload="if (!newwindow.closed)
   newwindow.close();"></body>
```

Ouverture de la carte du site avec fonction de fermeture automatique (sitemapwindow.html)

Vérification de la présence d'un système de blocage des fenêtres contextuelles

```
var newwindow = window.open(
  "",
  "popupcheck",
  "width=100,height=100,directories=no,menubar=no,
  ➥toolbar=no");
```

Toutes les sections précédentes supposent que le navigateur est capable d'ouvrir les fenêtres contextuelles. Par défaut, la plupart des navigateurs modernes autorisent les fenêtres

contextuelles ouvertes à partir du serveur local, mais bloquent celles ouvertes à partir d'Internet. De plus, diverses barres d'outils possédant leurs propres mécanismes de blocage des fenêtres contextuelles compliquent la vie des développeurs qui y font appel.

D'abord la mauvaise nouvelle : il n'existe pas de manière fiable de s'assurer que les fenêtres conceptuelles puissent être *toujours* ouvertes. Certains systèmes acceptent divers niveaux de blocage, d'autres autorisent certaines fenêtres contextuelles, d'autres encore les désactivent toutes. La Figure 9.5 montre un écran de configuration d'un navigateur ordinaire.

Le code suivant tente de détecter la présence d'un système de blocage des fenêtres contextuelles : il ouvre une fenêtre, puis vérifie son existence. Si elle est là, elle est fermée, sinon il existe probablement un système de blocage des fenêtres contextuelles :

```
<script language="JavaScript"
  type="text/JavaScript">
var newwindow = window.open(
  "",
  "popupcheck",
  "width=100,height=100,directories=no,menubar=no,
  ➥toolbar=no");
if (newwindow) {
  newwindow.close();
} else {
  window.alert("Pop-up blocker detected!");
}
</script>
```

Recherche d'un système de blocage des fenêtres contextuelles (popucheck.html)

La Figure 9.6 montre le résultat lorsqu'une fenêtre contextuelle est bloquée (également visible à partir de la barre de notification en haut de la zone de contenu du navigateur).

Figure 9.6 : Détection d'un système de blocage des fenêtres contextuelles.

Astuce

Le comportement par défaut de la plupart des systèmes de blocage des fenêtres contextuelles précise que ne sont autorisées que celles qui sont déclenchées par l'action des utilisateurs. N'essayez donc pas d'appeler `window.open()` lorsque la page s'est chargée, mais plutôt lorsque l'utilisateur clique sur quelque chose. Vous n'avez toutefois aucune garantie que la fenêtre contextuelle apparaîtra réellement.

Utilisation des cadres JavaScript

Travailler avec les cadres dans JavaScript revient un peu à travailler avec l'objet `window`. Sachez qu'il manque aux cadres certaines des fonctions d'un objet `window`, comme la méthode `open()`, ou des manières de les redimensionner ou de les repositionner. En revanche, les propriétés de base d'un cadre sont les mêmes : la propriété `document` pointe vers l'objet `document` chargé dans le cadre et on compte d'autres propriétés, comme `location`.

La propriété `frames` de la fenêtre principale (l'endroit où se trouve le document HTML contenant l'élément `<frameset>`) est un tableau contenant tous les cadres définis dans ce jeu de cadres. Vous pouvez ensuite accéder aux cadres par leurs indices (à partir de 0 : `window.frames[0]`, `window.frames[1]`...) ou par leur attribut `name` : `window.frames["nomDuCadre"]`.

Il est possible d'accéder à la structure du cadre à partir de chaque document de cadre (c'est-à-dire des documents chargés dans un cadre, même s'ils contiennent également des éléments `<frameset>`) : `parent` référence le document de la structure de cadre parent, tandis que `top` passe au document principal.

Modification du contenu de deux cadres en même temps

```
top.frames[frame1].location.href = url1;
top.frames[frame2].location.href = url2;
```

L'une des questions les plus communes de JavaScript concerne la modification simultanée de deux cadres. Même si les cadres deviennent de moins en moins populaires chez les professionnels du Web, cette question demeure un sujet important.

Bien entendu, JavaScript peut encore vous aider dans cette situation. Une fonction prend les noms ou les indices des deux cadres et les deux URL à charger. Puisque les cadres individuels se modifient au fil de l'application, cette fonction doit être placée dans le document du jeu de cadres :

```
<script language="JavaScript"
  type="text/JavaScript">
function changeFrames(frame1, url1, frame2, url2) {
  top.frames[frame1].location.href = url1;
  top.frames[frame2].location.href = url2;
}
</script>
<frameset cols="150,*">
  <frame src="frame1a.html" name="navigation" />
  <frame src="frame2a.html" name="content" />
</frameset>
```

Le jeu de cadres avec la fonction JavaScript (frameset.html)

Le cadre gauche de cet exemple correspond au cadre de navigation ; le cadre droit est le cadre de contenu. Lorsque l'utilisateur clique sur un lien du cadre de navigation, les deux cadres changent, chaque page du cadre de droite possédant une page de navigation associée (la page en cours étant surlignée) à destination du cadre gauche.

```
Frame 1<br />
<a href="JavaScript:top.changeFrames('navigation',
   'frame1b.html', 'content', 'frame2b.html');">
   Frame 2</a>
```

Le jeu de cadres avec les éléments de navigation (frame1a.html)

Il existe une méthode plus générale qui ne limite pas le nombre de cadres à deux, mais accepte un nombre arbitraire de noms de cadres et d'URL :

```
function changeFrames() {
  for (var i=0;
    i < Math.ceil(changeFrames.arguments.length/2);
    i++) {
    top.frames[changeFrames.arguments[2 * i]]
    ➥.location.href = changeFrames.arguments[2 * i + 1];
  }
}
```

Jeu de cadres avec la fonction JavaScript plus flexible (frameset.html, extrait)

Le fichier `frameset.html`, présent dans les archives de téléchargement, contient les deux versions de `changeFrames()`, la moins flexible se trouvant dans un commentaire JavaScript.

Astuce

Il est bien entendu que cet exemple fonctionne mieux si l'on exclut les utilisateurs n'ayant pas JavaScript. Il serait donc envisageable de charger une nouvelle page dans le cadre de contenu mais d'utiliser JavaScript pour modifier également le cadre de gauche, si le navigateur l'accepte :

```
<a href="frame2b" target="content"
  onclick="top.changeFrames('navigation',
     'frame1b.html', 'content', 'frame2b.html');
     return false;">Frame 2</a>
```

Utilisation de cadres masqués pour le stockage des données

Une option assez commode pour les cadres consiste à utiliser les cadres masqués. Ils peuvent servir, par exemple, à stocker des données sur les requêtes HTTP. Toutes les données JavaScript étant perdues si la page n'est pas déchargée, un cadre masqué peut permettre de stocker des données, par exemple lors de la création d'un panier d'achat côté client.

Il faut donc créer un cadre dont la hauteur est égale à 1 pixel par exemple. Par la suite, dès qu'il y a des données à mettre de côté, il suffit d'accéder au cadre et d'enregistrer les données requises dans des variables, toutes accessibles sous forme de propriétés de cadre.

```
top.frames["hiddenJsFrame"].variable = "value";
```

Vous pouvez également placer des fonctions communes dans ce cadre, puis les appeler de la même manière :

```
top.frames["hiddenJsFrame"].functionName();
```

Utilisation des cadres intégrés

```
window.frames["iframe1"].location.href = "iframe1b.html";
document.getElementById("iframe1").src = "iframe1b.html";
```

Les cadres intégrés facilitent la création d'une section dans une page, qui pourra être modifiée indépendamment du reste de la page. Comme leur nom le suggère, ils sont très comparables aux cadres, à quelques différences près. Voici le marquage HTML de ce cadre intégré :

```
<iframe src="iframe1a.html" name="iframe1a"
  id="iframe1"></iframe>
```

Un nom et un identifiant sont fournis pour le cadre intégré, puisqu'il existe deux manières d'y accéder :

- en utilisant le nom et le tableau `window.frames` ;
- en utilisant l'identifiant et `document.getElementById()`.

La première exige que la propriété `location.href` modifie l'URL dans le cadre intégré, la seconde renvoie un objet possédant une propriété `src` pour cette tâche :

- `window.frames["iframe1"].location.href = ➥"iframe1b.html"` ;
- `document.getElementById("iframe1").src = ➥"iframe1b.html"`.

Attention

Même si les cadres intégrés proposent de nouvelles capacités pour les applications Web, ils présentent également des inconvénients. Comme vous le verrez au Chapitre 11, "AJAX et sujets annexes", la création de signets et le fonctionnement des boutons Précédent et Suivant nécessitent quelques contournements, qui ne fonctionnent que lorsque JavaScript est activé.

10

Services Web

Les services Web sont une technologie étonnante, côté serveur. Leur lien avec JavaScript tient à ce qu'il existe une méthode pour appeler des scripts côté serveur depuis le côté client. En réalité, deux méthodes existent : l'une pour les navigateurs fondés sur Mozilla, l'autre pour Internet Explorer. Ces deux options sont traitées dans ce chapitre ; le chapitre suivant propose d'autres solutions intéressantes.

Comprendre les services Web

Certains disent que les services Web ne sont qu'un vieil outil qui a été remis à neuf, mais l'idée qui se cache derrière est loin d'être nouvelle. Le principe de base est que deux machines se parlent l'une à l'autre. A titre d'exemple, une machine contient une logique d'activité ou, plus généralement, des informations, et l'autre demande ces informations (ou veut utiliser la logique d'activité).

Ces opérations sont réalisées depuis longtemps, mais ce n'est qu'il y a quelques années que les principaux acteurs se sont assis autour d'une table pour travailler sur le protocole et placer la communication au-dessus des autres normes. Certaines de ces normes sont maintenant sous l'égide du consortium World Wide Web (W3C), tandis que d'autres sont gérées par le consortium OASIS (Organization for the Advancement of Structured Information Standards, organisation pour l'avancée des normes d'information structurée).

Il existe de nombreux ouvrages sur les services Web, tous généralement assez volumineux. Toutefois, fidèle au concept de ce guide de survie, ce chapitre tente de simplifier les choses, tout en vous donnant tous les éléments essentiels. Ainsi, sachez qu'un protocole sert à transférer la requête vers le service Web et sa réponse en retour. Il existe plusieurs possibilités, mais ces trois sont les plus fréquentes :

- **XML-RPC.** Il s'agit d'un appel de procédure distante XML. XML-RPC utilise un dialecte XML simple pour transférer des appels de fonction, des paramètres et des valeurs de retour.
- **REST** (*Representational state transfer*, changement d'état de représentation). Il utilise pour l'essentiel des requêtes HTTP (`GET` la plupart du temps mais d'autres verbes

HTTP peuvent aussi être employés) sous forme d'appel de service Web. Les données renvoyées par REST sont du XML brut.

- **SOAP** (*Simple Object Access Protocol*)**.** Le sigle désignait à ses origines un protocole simple d'accès aux objets, or ce protocole n'est pas simple et ne traite pas vraiment de l'accès aux objets, l'acronyme a donc aujourd'hui perdu toute sa signification. Il s'agit d'un protocole plutôt complexe mais qui contourne de nombreuses limitations de XML-RPC (qui a d'ailleurs été créé par des personnes ayant travaillé sur SOAP).

Des débats houleux ont lieu quant à savoir lequel est le meilleur, ce protocole ou le suivant. Pour tout dire, tous ont des avantages et des inconvénients.

La plupart du temps, les appels de type REST, XML-RPC ou SOAP sont transportés sur du HTTP qui agit comme un protocole de transport. Mais d'autres protocoles peuvent aussi être utilisés, notamment SMTP (*Simple Mail Transfer Protocol,* protocole simple de transfert de courrier), voire UDP (*User Datagram Protocol*, protocole de datagramme utilisateur).

Grâce à SOAP, lorsque vous savez exactement comment est implémenté un service Web, vous savez également comment l'appeler (y accéder ou l'utiliser). Toutefois, bien souvent, ces informations ne sont pas disponibles, il faut donc qu'il y ait une sorte d'autodescription du service Web contenant toutes les informations importantes, comme les méthodes ou les fonctions exposées, les paramètres et les types de données attendus, ainsi que les éléments renvoyés. Cela peut être réalisé à l'aide d'un fichier XML créé spécifiquement dans ce but. La norme utilisée est WSDL (*Web Service Description Language*, langage de description de services Web). De nos jours, elle est utilisée pour tous

les services Web importants car elle simplifie leur emploi. La plupart des technologies côté serveur proposent d'ailleurs un ou plusieurs moyens de lire du WSDL, puis d'accéder au service Web comme on accéderait à une classe disponible localement.

L'utilisation de tout ce code XML peut vite devenir assez complexe. Le chapitre suivant traite du XML dans le cadre de la technologie AJAX et propose quelques pistes supplémentaires.Cependant, les navigateurs Mozilla et Microsoft Internet Explorer disposent de leur propre méthode pour accéder aux services Web. Cela allège un peu la complexité mais n'est malheureusement pas valable pour tous les navigateurs.

Création d'un service Web avec PHP

```
$soap = new SoapServer(
  'RandomNumberService.wsdl',
  array('uri' =>
    'http://JavaScript.phrasebook.org/')
);
```

Pour utiliser un service Web, il faut d'abord le trouver. La technologie côté serveur la plus populaire pour l'heure s'intitule PHP (*Hypertext Preprocessor,* préprocesseur hypertexte), c'est donc le choix évident pour implémenter un service Web. Il faut pour cela un serveur Web sur lequel PHP soit installé. De nombreux modules d'hébergement Web abordables le contenant existent aujourd'hui, mais PHP peut être installé sur la plupart des serveurs Web. Vous pouvez aussi télécharger sa distribution et son source sur sa page d'accueil : **http://www.php.net/**.

Pour que tout fonctionne, procurez-vous PHP 5 ou une version ultérieure et activez la bibliothèque SOAP livrée avec. Sous UNIX/Linux, vous devez compléter PHP avec l'argument de configuration ; les utilisateurs de Windows ajouteront la ligne suivante au fichier de configuration `php.ini` :

```
extension=php_soap.dll
```

Il faut également disposer d'un fichier WSDL, c'est-à-dire d'une description du service Web comprenant les méthodes qu'il expose et les paramètres qu'il attend. Le fichier `RandomNumberService.wsdl` contient toutes ces informations. N'oubliez pas de consulter la section `<soap:address location="http://localhost/js/webservice.php" />` du WSDL et d'adapter l'URL à votre système local.

Le script suivant implémente un service Web SOAP simple avec une méthode (qui renvoie un nombre aléatoire dans un intervalle donné) :

```
<?php
  $soap = new SoapServer(
    'RandomNumberService.wsdl',
    array('uri' =>
      'http://JavaScript.phrasebook.org/')
  );
  $soap->setClass('ServiceClass');
  $soap->handle();

  class ServiceClass {
    function randomNumber($lower, $upper) {
      return rand($lower, $upper);
    }
  }
?>
```

Un service Web PHP (webservice.php)

Création d'un service Web avec ASP.NET

```
<%@ WebService language="C#"
  class="RandomNumberService" %>
```

ASP.NET de Microsoft est aujourd'hui l'une des technologies Web dont le développement est parmi les plus rapides. Elle présente un aspect intéressant : elle simplifie considérablement la création et l'utilisation de services Web.

Pour implémenter un service Web, il faut créer un fichier présentant l'extension `.asmx` sur le serveur Web. La directive `<%@ WebService %>` doit ensuite être placée en haut de la page. Toutes les méthodes qui seront exposées comme méthodes de services Web doivent être préfixées par l'attribut `[WebMethod]`. En voici un exemple complet :

```
<%@ WebService language="C#"
  class="RandomNumberService" %>

using System;
using System.Web.Services;

[WebService(Namespace=
   "http://JavaScript.phrasebook.org/")]
public class RandomNumberService {

  [WebMethod]
  public int randomNumber(int lower, int upper) {
    Random r = new Random();
    return r.Next(lower, upper + 1);
  }
}
```

Un service Web ASP.NET (webservice.asmx)

Lorsque vous appelez cette page à partir d'un serveur Web activé pour ASP.NET dans votre navigateur Web, vous obtiendrez (selon votre configuration) une jolie page d'informations présentant des détails sur le service. Si vous ajoutez `?WSDL` à l'URL, vous obtenez une description WDSL, automatiquement générée à partir de ASP.NET. La Figure 10.1 montre le résultat.

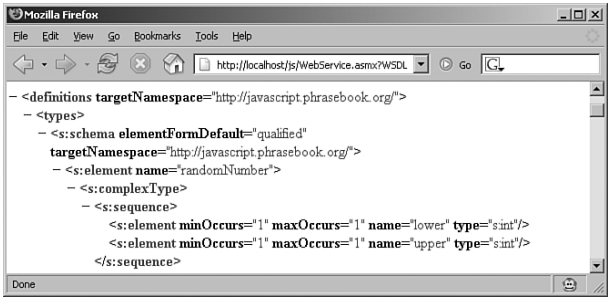

Figure 10.1 : Page de détails automatiquement générée pour le service Web d'ASP.NET.

Appel d'un service Web à partir d'Internet Explorer

```
WebService.useService("webservice.php?WSDL",
  "RandomNumberService");
```

Il y a quelques années, Microsoft proposait un script d'aide pour accéder à un service Web depuis une page HTML utilisant JavaScript. Pour que cela fonctionne, le service Web devait se trouver sur le même serveur que le code JavaScript.

Malheureusement, Microsoft a cessé de développer ce composant et l'a masqué dans MSDN (*réseau des développeurs de Microsoft*). Il est pourtant toujours disponible et fonctionne bien. Rendez-vous à l'adresse **http://msdn.microsoft.com/archive/en-us/samples/Internet/behaviors/library/webservice/default.asp**, téléchargez le fichier `webservice.htc` et placez-le dans le répertoire où se trouve votre fichier HTML (le fichier `webservice.htc` ne fait pas partie du code source téléchargeable pour cet ouvrage). Le marquage suivant charge alors le composant :

```
<div id="WebService"
  style="behavior:url(webservice.htc);">
```

Puis, `WebService` référence le composant et permet de charger un service Web distant (méthode `useService()`), en fournissant sa description WSDL. Le composant permet alors au code JavaScript d'appeler une méthode Web (méthode `callService()`, en fournissant une fonction de rappel, un nom de méthode et des paramètres).

Dans la fonction de rappel, le résultat peut être analysé ou simplement affiché, comme le montre la séquence de code suivante :

```
<html>
<head>
  <title>JavaScript</title>
  <script language="JavaScript"
    type="text/JavaScript">
  function callWebService(f) {
    WebService.useService("webservice.php?WSDL",
      "RandomNumberService");
    WebService.RandomNumberService.callService(
      callbackFunction,
      "randomNumber",
      parseInt(f.elements["lower"].value),
```

```
      parseInt(f.elements["upper"].value));
  }

  function callbackFunction(result) {
    document.getElementById("random").innerHTML =
      result.value;
  }
  </script>
</head>
<body>
  <div id="WebService"
    style="behavior:url(webservice.htc);">
  </div>
  <form>
    A number between
    <input type="text" name="lower" size="3"
      value="1" /> and
    <input type="text" name="upper" size="3"
      value="49" /> is
    <span id="random" /><br />
    <input type="button" value="Retrieve"
      onclick="callWebService(this.form);" />
  </form>
</body>
</html>
```

Appel de services Web à partir d'Internet Explorer (webservice-ie-php.html)

La Figure 10.2 montre le résultat : le nombre aléatoire provient du service Web sur le serveur Web.

Info

Le listing précédent traite de l'appel du service Web PHP ; le fichier `webservice-ie-aspnet.html` traite le service Web ASP.NET. La seule différence réside dans le fait que le nom de l'URL du service Web a été modifié ; le reste du code est inchangé.

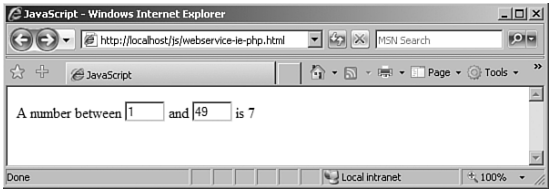

Figure 10.2 : Le nombre aléatoire provient du service Web.

Appel d'un service Web à partir d'un navigateur Mozilla

```
var soapcall = new SOAPCall();
```

Les navigateurs Mozilla ont leur propre manière d'appeler des services Web. La classe JavaScript `SOAPCall` peut d'ailleurs se charger de créer la requête SOAP. Après avoir instancié la classe, vous devez paramétrer deux propriétés : l'action SOAP (un champ d'en-tête SOAP, souvent l'espace de nom du service Web plus le nom de la méthode) et l'URI (*Uniform Resource Identifier*) absolue (comprenant le nom du serveur) du service Web.

```
var soapcall = new SOAPCall();
soapcall.actionURI =
  "http://JavaScript.phrasebook.org/randomNumber";
soapcall.transportURI =
  "http://localhost/js/webservice.php";
```

Vous créez ensuite un tableau de tous les paramètres à utiliser dans l'appel du service Web, en instanciant la classe `SOAPParameter` :

```
var parameters = [
  new SOAPParameter(1, "lower"),
  new SOAPParameter(49, "upper")
];
```

Puis, la méthode `encode()` prépare l'appel du service Web. Elle prend les paramètres suivants :

- la version SOAP (0 désigne SOAP 1.1, c'est la valeur recommandée) ;
- le nom de la méthode du service Web appelé ;
- l'espace de nom du service Web ;
- d'autres en-têtes SOAP envoyés (sous forme de tableau) ;
- le nombre de paramètres à envoyer ;
- les paramètres à envoyer (sous forme de tableau).

La méthode `asyncInvoke()` appelle le service Web ; vous pouvez fournir une fonction de rappel qui s'exécute lorsque les données reviennent du service Web. Il s'agit d'un document XML, vous devez donc utiliser DOM pour accéder aux informations importantes. Le code suivant fonctionne généralement bien :

```
function callbackFunction(result) {
  var returnData =
     result.body.firstChild.firstChild.firstChild.data;
}
```

Voici le code complet :

```
<html>
<head>
  <title>JavaScript</title>
  <script language="JavaScript"
    type="text/JavaScript">
  function callWebService(f) {
    var soapcall = new SOAPCall();
    soapcall.actionURI =
       "http://JavaScript.phrasebook.org/randomNumber";
    soapcall.transportURI =
      "http://localhost/js/webservice.php";
```

```
    var parameters = [
      new SOAPParameter(parseInt(
        f.elements["lower"].value), "lower"),
      new SOAPParameter(parseInt(
        f.elements["upper"].value), "upper")
    ];
    soapcall.encode(
      0,
      "randomNumber",
      "http://JavaScript.phrasebook.org/",
      0,
      [],
      parameters.length,
      parameters
    );
    soapcall.asyncInvoke(callbackFunction);
  }

  function callbackFunction(result) {
    document.getElementById("random").innerHTML =
      result.body.firstChild.firstChild.firstChild.data;
  }
  </script>
</head>
<body>
  <form>
    A number between
    <input type="text" name="lower" size="3"
      value="1" /> and
    <input type="text" name="upper" size="3"
      value="49" /> is
    <span id="random" /><br />
    <input type="button" value="Retrieve"
      onclick="callWebService(this.form);" />
  </form>
</body>
</html>
```

Appel d'un service Web à partir de Mozilla (webservice-mozilla-php.html)

Info

Une fois de plus, sachez que vous devrez peut-être modifier l'URL pour l'appel du service Web afin qu'elle corresponde à votre système.

Appel d'un service Web ASP.NET à partir d'un navigateur Mozilla

```
var schematype = schemacoll.getType(
  "integer",
  "http://www.w3.org/2001/XMLSchema"
);
```

Il existe un problème potentiel avec le code précédent : les types de données des paramètres envoyés au service Web ne sont pas fournis. Malheureusement, certaines implémentations de services Web nécessitent ces informations pour fonctionner. Et surtout, certains services Web .NET ne fonctionnent pas sans ces informations.

Il est difficile de dire qui est responsable de la situation, Microsoft ou Mozilla, mais il existe au moins une manière de faire coopérer ces deux systèmes. Pour cela, vous devez modifier le moyen de création des paramètres. Après avoir instancié la classe `SOAPCall`, vous devez charger le codage correct pour chaque paramètre. Dans l'exemple, le type de données utilisé est `integer`, c'est donc lui que nous allons employer. Le tableau `parameters` est maintenant créé de la manière suivante :

```
var l = new SOAPParameter(parseInt(
  f.elements["lower"].value), "lower");
var u = new SOAPParameter(parseInt(
  f.elements["upper"].value), "upper");
```

```
var soapenc = new SOAPEncoding();
assenc = soapenc.getAssociatedEncoding(
  "http://schemas.xmlsoap.org/soap/encoding/",
  false
);
var schemacoll = assenc.schemaCollection;
var schematype = schemacoll.getType(
  "integer",
  "http://www.w3.org/2001/XMLSchema"
);
l.schemaType = schematype;
u.schemaType = schematype;
var parameters = [l, u];
```

Appel d'un service Web avec des types de données distinctifs depuis Mozilla (webservice-mozilla-php.html, extrait)

Astuce

Vérifiez d'abord si le service Web fonctionne sans ce codage supplémentaire. Les anciennes versions de Mozilla n'acceptent pas la méthode `getAssociatedEncoding()`, tandis que certaines versions plus récentes parviennent à coopérer avec certains services Web .NET sans qu'il soit nécessaire de fournir le type de données.

11

AJAX et sujets annexes

AJAX (*Asynchronous JavaScript And XML*) a beaucoup fait parler de lui lorsque le terme a été inventé en février 2005. Et même s'il y a eu beaucoup de critiques (justifiées), concernant à la fois le terme et le mélange technologique qu'il promet, tout le bruit fait autour d'AJAX a mené à la renaissance de JavaScript. Non seulement les capacités sous-estimées de JavaScript s'intègrent dans le travail quotidien des développeurs Web, mais les fonctions plus avancées sont également de nouveau à la mode. Ce chapitre présente les faits les plus importants sur JavaScript et les technologies annexes, en particulier la gestion du XML.

Comprendre AJAX

Dans les années 1990, Microsoft ajoute un nouvel objet ActiveX à son navigateur Web Internet Explorer (version 5.0) : `XMLHttpRequest`. Cet objet pouvait envoyer des requêtes HTTP à un serveur et évaluer les données qu'il renvoyait, un peu comme le concept des services Web (mais à l'aide d'une autre méthode appelée REST, *Representational state transfer*). L'idée est née d'une requête de l'équipe Outlook qui travaillait sur la dernière version du frontal Web pour son logiciel.

Il y a quelques années, ce sont les développeurs du projet Mozilla qui ont découvert cet objet, décidant de créer leur propre version de `XMLHttpRequest`, bien entendu sous forme d'objet JavaScript natif, ActiveX ne fonctionnant pas sur toutes les plates-formes. Les développeurs Safari ont eux aussi implémenté `XMLHttpRequest`. Puisque le navigateur était basé sur le moteur de rendu KHTML de Konqueror, le code (donné par Apple) pourrait aussi être appliqué au navigateur KDE. Enfin, les navigateurs Opera et iCab ont également ajouté une prise en charge de `XMLHttpRequest`.

De plus en plus d'applications Web utilisant `XMLHttpRequest`, Jesse James Garrett a écrit un article en février 2005 intitulé "Ajax: A New Approach to Web Applications" (*Ajax : une nouvelle approche des applications Web*, disponible en ligne à l'adresse **http://www.adaptivepath.com/publications/essays/archives/000385.php**). Dans cet article, il inventait le terme AJAX qui signifie "JavaScript asynchrone + XML". Il avançait que sa société utilisait `XMLHttpRequest` et JavaScript, ainsi que DOM, XML, XSLT, XHTML et CSS et qu'il était de plus en plus difficile de l'expliquer aux clients, en particulier s'ils n'étaient pas au fait de la technique. Il a donc inventé ce terme global qui a pourtant fait naître de nombreuses critiques (par exemple, aucun

codage XML n'est nécessaire pour AJAX). Toutefois, c'est l'invention du terme "AJAX" qui a donné le départ de la technologie, c'est à partir de là qu'elle a véritablement décollé.

D'un point de vue technique, AJAX consiste à utiliser `XMLHttpRequest` pour envoyer des requêtes HTTP au serveur Web (qui doit se trouver dans le même domaine que le script, pour des raisons de sécurité) et à utiliser JavaScript pour évaluer et afficher les données sur le client. Son utilité se limite à cela ce qui n'empêche pas que se tiennent des conférences entières consacrées au sujet.

Initialisation d'une application AJAX

```
XMLHttp = new XMLHttpRequest();
XMLHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

La base de toutes les applications AJAX est l'objet `XMLHttpRequest` déjà mentionné. Tous les navigateurs activés pour AJAX le prennent en charge en natif mais, dans Internet Explorer, l'objet ActiveX doit être présent. Il existe une exception : Internet Explorer 7 assure la prise en charge native de `XMLHttpRequest`. La meilleure technique pour créer l'objet consiste à utiliser `try...catch` et à instancier d'abord l'objet natif (pour mettre Internet Explorer 7 sur la bonne piste, même si ce navigateur prend toujours en charge ActiveX), puis la version ActiveX :

```
if (window.XMLHttpRequest) {
  // instantie l'objet natif
} else if (window.ActiveXObject) {
  // instantie l'objet ActiveX
}
```

A noter qu'il existe plusieurs possibilités d'instancier l'objet ActiveX : Microsoft commercialise en effet plusieurs versions de sa bibliothèque XML dans lesquelles l'objet est caché. Une solution tout à fait sûre consisterait à procéder à une vérification de toutes les versions, en utilisant une partie de code compliquée. Toutefois, la solution suivante ne porte que sur les versions les plus importantes et fonctionne avec Internet Explorer versions 5 et suivantes et sur tous les autres navigateurs compatibles avec AJAX.

```
function getXMLHttp() {
  var XMLHttp = null;
  if (window.XMLHttpRequest) {
    try {
      XMLHttp = new XMLHttpRequest();
    } catch (e) { }
  } else if (window.ActiveXObject) {
    try {
      XMLHttp = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
      try {
        XMLHttp = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (e) { }
    }
  }
  return XMLHttp;
}
```

Création de l'objet `XMLHttpRequest` (xmlhttp.js)

Envoi d'une requête GET

```
XMLHttp.open("GET", "phrasebook.txt");
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.send(null);
```

L'envoi d'une requête HTTP à un serveur à l'aide de `XMLHttpRequest` passe par les étapes suivantes :

1. Fournir l'URL et le verbe HTTP à utiliser.
2. Définir une fonction de rappel lorsque les résultats arrivent.
3. Envoyer la requête.

L'étape 1 peut être réalisée avec la méthode `open()` de l'objet `XMLHttpRequest`. A la différence de ce que suggère son nom, la méthode n'ouvre pas réellement de connexion HTTP, elle se contente d'initialiser l'objet. Vous fournissez le verbe HTTP à utiliser (généralement `GET` ou `POST`) ainsi que l'URL.

A l'étape 2, la fonction de rappel est fournie à la propriété `onreadystatechange` de l'objet. Dès que la propriété `readyState` de `XMLHttpRequest` change, cette fonction de rappel est appelée. Enfin, la méthode `send()` envoie la requête HTTP.

Dans la fonction de rappel, la valeur `4` de `readyState` représente l'état de l'objet que nous souhaitons : l'appel réalisé. Dans ce cas, la propriété `responseText` contient des données renvoyées du serveur.

En voici un exemple tout à fait fonctionnel, l'envoi d'une requête `GET` au serveur (un fichier appelé `phrasebook.txt`

avec un contenu de texte simple) et l'évaluation de la réponse de cet appel :

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
 type="text/JavaScript">
var XMLHttp = getXMLHttp();
XMLHttp.open("GET", "phrasebook.txt");
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.send(null);

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    window.alert("Returned data: " + XMLHttp.responseText);
  }
}
</script>
```

Envoi d'une requête GET (xmlhttpget.html)

Comprendre les états de XMLHttpRequest

Dans l'ensemble, l'objet `XMLHttpRequest` accepte cinq états, indiqués au Tableau 11.1 Selon l'implémentation, tous ces états peuvent survenir pendant l'exécution d'un script AJAX. C'est la raison pour laquelle il est tellement important de toujours interroger l'état avant d'essayer d'accéder à d'autres données `XMLHttpRequest`.

Tableau 11.1 : États de l'objet XMLHttpRequest

Etat	Description
0	Non initialisé
1	Chargement

Tableau 11.1 : États de l'objet XMLHttpRequest *(suite)*

Etat	Description
2	Chargé
3	En attente
4	Terminé

Envoi d'une requête POST

```
XMLHttp.setRequestHeader("Content-type",
  "application/x-www-form-urlencoded");
XMLHttp.send("word1=JavaScript&word2=Phrasebook");
```

Lorsqu'on envoie une requête `GET`, tous les paramètres font partie de l'URL. Pour `POST`, toutefois, ces données sont envoyées dans le corps de la requête HTTP. Pour procéder ainsi avec l'objet `XMLHttpRequest`, les paramètres doivent être fournis dans la méthode `send()`. Un inconvénient apparaît : si vous voulez accéder à ces paramètres côté serveur, il faut définir l'en-tête HTTP `Content-type` correct, avec la méthode `setRequestHeader()`, de la manière suivante :

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
var XMLHttp = getXMLHttp();
XMLHttp.open("POST", "post.php");
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.setRequestHeader("Content-type",
  "application/x-www-form-urlencoded");
XMLHttp.send("word1=JavaScript&word2=Phrasebook");
```

```
function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    window.alert("Returned data: " + XMLHttp.responseText);
  }
}
</script>
```

Envoi d'une requête POST (xmlhttppost.html)

Le script pointe vers l'URL `post.php`, un simple script PHP renvoyant les données :

```
<?php
  if (isset($_POST['word1']) && isset($_POST['word2'])) {
    echo $_POST['word1'] . ' ' . $_POST['word2'];
  } else {
    echo 'No data sent.';
  }
?>
```

Le script PHP ciblé par la requête POST (post.php)

Bien entendu, vous pouvez fournir tout autre script de n'importe quelle autre technologie côté serveur ou simplement pointer (`POST`) vers un fichier de texte brut.

Envoi d'une requête synchrone

```
XMLHttp.open("GET", "phrasebook.txt", false);
```

Par défaut, les requêtes HTTP *via* `XMLHttpRequest` sont asynchrones, ce qui explique la nécessité d'une fonction de rappel. Toutefois, lorsque vous paramétrez le troisième

paramètre de la méthode `open()` sur `false`, la requête est synchrone, ce qui signifie que l'exécution de script est arrêtée jusqu'à ce que les données reviennent du serveur. Le code suivant en fait usage :

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
window.onload = function() {
  var XMLHttp = getXMLHttp();
  XMLHttp.open("GET", "phrasebook.txt", false);
  XMLHttp.send(null);
  document.getElementById("output").innerHTML =
    "Returned data: " + XMLHttp.responseText;
}
</script>
<p id="output">Calling the server ...</p>
```

Envoi d'une requête synchrone (xmlhttpsync.html)

Sachez que l'ensemble du code n'est exécuté qu'après le chargement complet de la page, faute de quoi l'accès à l'élément HTML de résultat peut échouer.

Attention

Les appels synchrones peuvent être pratiques mais il vaut mieux les éviter pour des raisons de simplicité d'utilisation ou de performances. L'exécution du script s'arrête totalement lorsque l'appel est réalisé, ce qui peut alors se transformer en véritable cauchemar sur des connexions lentes ou des serveurs chargés.

Réception de données multiples du serveur

```
var data = XMLHttp.responseText.split("\n");
```

Par défaut, la propriété `responseText` n'existe qu'une fois ; l'ensemble des données renvoyées par le serveur sera donc placé dans la chaîne. Toutefois, il est souvent nécessaire que plusieurs données soient renvoyées par l'appel `XMLHttpRequest`. Il existe plusieurs solutions à ce scénario, la plus simple étant probablement de renvoyer plusieurs données et de les séparer à l'aide d'un caractère de séparation qui n'apparaît pas dans les données elles-mêmes (par exemple, un tabulateur ou une nouvelle ligne ou le symbole de la barre verticale). Il est alors possible d'utiliser JavaScript pour accéder à ces informations.

Pour cette expression, les fichiers texte suivants, côté serveur, sont interrogés à l'aide de HTTP. Dans un scénario concret, il y aurait probablement un script dynamique côté serveur mais, pour la démonstration, le fichier statique suffit.

```
JavaScript
Phrasebook
Sams Publishing
```

Données multiples dans un fichier (phrasebook-multiple.txt)

Le code suivant accède ensuite à chaque information des données renvoyées ; la Figure 11.1 montre le résultat.

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
```

```
var XMLHttp = getXMLHttp();
XMLHttp.open("GET", "phrasebook-multiple.txt");
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.send(null);

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    var data = XMLHttp.responseText.split("\n");
    window.alert(data[0] + " " + data[1] + " by " +
    ➥data[2]);
  }
}
</script>
```

Travailler avec plusieurs données à partir du serveur (xmlhttpmultiple.html)

Figure 11.1 : Le serveur renvoie plusieurs données.

Interruption d'une requête HTTP

```
XMLHttp.abort();
```

Selon le navigateur, seul un nombre limité de requêtes HTTP peut être exécuté au même moment. Vous risquez d'avoir des problèmes, notamment si vous disposez d'une page incluant plusieurs composants AJAX. L'interruption d'une requête HTTP peut donc devenir une nécessité.

La méthode utilisée pour cela est `abort()`. Le code suivant interrompt la requête si elle n'a pas été totalement exécutée au bout de cinq secondes. Pour démontrer ce comportement, le script PHP `delay.php` qui est appelé par le code demande dix secondes à s'exécuter. Sachez également qu'on vérifie en premier la propriété `readyState` : si elle est égale à 0 ou 4, il n'y a rien à interrompre.

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
var XMLHttp = getXMLHttp();
XMLHttp.open("GET", "delay.php?" + Math.random());
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.send(null);
window.setTimeout("timeout();", 5000);

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    window.alert("Returned data: " +
                ➥XMLHttp.responseText);
  }
}

function timeout() {
```

```
  if (XMLHttp.readyState != 4 && XMLHttp.readyState != 0) {
    XMLHttp.onreadystatechange = function() { };
    XMLHttp.abort();
    window.alert("Request aborted");
  }
}
</script>
```

Interruption de requête HTTP (xmlhttpabort.html)

Cinq secondes après le chargement de cette page, la requête est interrompue.

Astuce

Ajouter le résultat de `Math.random()` à l'URL à appeler entraîne la mise en cache du script `delay.php` à désactiver, puisque l'URL est différente à chaque fois.

Récupération d'en-têtes HTTP

```
var headers = XMLHttp.getAllResponseHeaders();
```

La méthode `getAllResponseHeaders()` (voir le code précédent) renvoie tous les en-têtes HTTP de la réponse HTTP, tandis que la méthode `getResponseHeader()` ne renvoie qu'un seul en-tête spécifique. Le code suivant montre comment récupérer des informations sur le type de serveur Web utilisé :

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
```

```
var XMLHttp = getXMLHttp();
XMLHttp.open("GET", "phrasebook.txt");
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.send(null);

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    var servertype =
      XMLHttp.getResponseHeader("Server");
    window.alert("Web server used: " + servertype);
  }
}
</script>
```

Récupération d'un en-tête de réponse HTTP (responseheader.html)

Sachez que tous les serveurs n'envoient pas cet en-tête, certains y insèrent des données fausses pour compliquer le profilage du serveur.

Réception de XML en provenance du serveur

```
var xml = XMLHttp.responseXML;
```

La propriété `responseText` fonctionne bien pour une quantité limitée de données non structurées. Une méthode plus élégante pour traiter des données complexes et structurées dans une application AJAX consiste à utiliser la propriété `responseXML`. Lorsqu'on y accède, la réponse de la requête HTTP se fait sous forme d'objet DOM XML JavaScript, mais uniquement si le serveur renvoie du XML valable. Ce sera `null` dans le cas contraire.

Accéder aux détails de l'objet XML revient en quelque sorte à accéder aux éléments DOM depuis JavaScript. Pour cette section, nous utiliserons l'extrait de fichier XML suivant :

```
<books>
  <book pubdate="2006">
    <title>JavaScript Phrasebook</title>
    <publisher>Sams Publishing</publisher>
  <book>
</books>
```

Exemple de données XML (phrasebook.xml)

Pour que les navigateurs Web puissent lire ce code XML (Internet Explorer notamment est très strict à ce sujet), il faut envoyer au client le type MIME correct : `text/xml`. Si vous passez par Apache, optez plutôt pour la configuration suivante dans `mime.types`, déjà présente par défaut :

```
text/xml
```

Sous Internet Information Services (IIS), vous pouvez configurer les types MIME dans la console d'administration (`Start`, `Run`, `inetmgr`). Vous pouvez aussi permettre à un script côté serveur de présenter le fichier avec le type MIME correct ; c'est généralement obligatoire si vous utilisez une technologie côté serveur pour générer le XML :

```
<?php
  header('Content-type: text/xml');
  readfile('phrasebook.xml');
?>
```

Paramétrage du type MIME correct pour la réponse (phrasebook.xml.php)

Le code suivant accède aux informations du fichier XML, en employant la structure DOM et des méthodes comme `getElementsByTagName()` et `getAttribute()`. La Figure 11.2 présente le résultat.

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
var XMLHttp = getXMLHttp();
XMLHttp.open("GET", "phrasebook.xml");
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.send(null);

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    var xml = XMLHttp.responseXML;
    var book = xml.getElementsByTagName("book")[0];
    var pubdate = book.getAttribute("pubdate");
    var title, publisher;
    for (var i=0; i<book.childNodes.length; i++) {
      if (book.childNodes[i].nodeName == "title") {
        title = book.childNodes[i].firstChild .nodeValue;
      } else if (book.childNodes[i].nodeName ==
      ➥"publisher") {
        publisher = book.childNodes[i].firstChild
        ➥.nodeValue;
      }
    }
    window.alert(title + " by " + publisher +
                ➥" (" + pubdate + ")");
  }
}
</script>
```

Extraction d'informations de la réponse HTTP (xmlhttpxml.html)

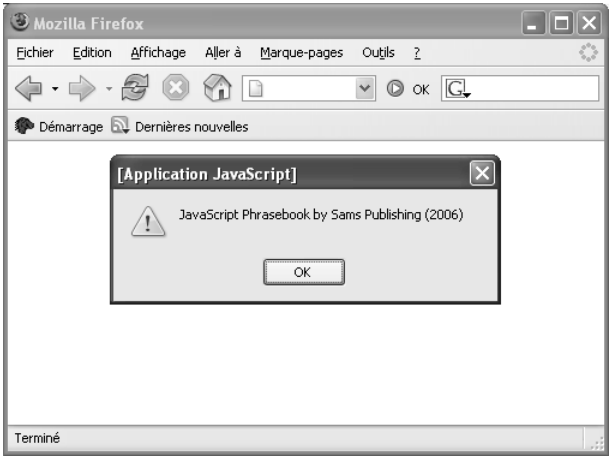

Figure 11.2 : Les données du fichier XML.

Astuce

Autre propriété utile du document DOM XML : **`documentElement`**, qui est un raccourci vers l'élément racine des données XML. Ainsi, si le nœud racine contient des attributs qui vous intéressent (comme le font certains services Web), **`documentElement`** est assez pratique.

Comprendre JSON

JSON (*JavaScript Object Notation,* notation d'objet JavaScript) est, tout comme AJAX, plutôt un terme qu'une technologie mais il possède lui sa propre page d'accueil, **http://www.json.org/**.

JSON est une caractéristique négligée et sous-estimée de JavaScript, une notation assez compacte pour les tableaux et les objets. De nombreux ouvrages définissent les tableaux JavaScript de la manière suivante, par exemple :

```
var ajax = new Array(
  "Asynchronous", "JavaScript", "+", "XML");
```

Il existe une manière plus courte, qui utilise des crochets :

```
var ajax = [
  "Asynchronous", "JavaScript", "+", "XML"];
```

Il en va de même pour les objets (qui sont des tableaux plus ou moins détaillés dans JavaScript), cette fois en utilisant des accolades. Le code suivant définit un objet avec trois propriétés :

```
var book = {
  "title": "JavaScript Phrasebook",
  "publisher": "Sams Publishing",
  "pubdate": 2006
};
```

Les deux "raccourcis" peuvent être associés pour représenter des données complexes sous forme de chaîne. Et c'est là l'objet de JSON : puisqu'il peut être joliment exprimé dans une chaîne, c'est un très bon format de sérialisation de données. C'est assez facile de créer une représentation JSON d'un objet côté serveur et c'est encore plus simple de transformer cette chaîne en un objet JavaScript côté client, comme le montre la section suivante.

Utilisation de JSON pour la (dé)sérialisation de données

```
var json = XMLHttp.responseText;
var book = eval("(" + json + ")");;
```

JSON devient de plus en plus fréquemment le format d'échange de données standard pour les applications AJAX. De nombreuses structures AJAX le prennent en charge, de nombreux services Web proposent une interface JSON et PHP 6 assurera probablement une prise en charge de JSON au cœur du langage.

Utiliser JSON dans JavaScript est également assez simple. Le code qui précède évalue JSON et le transforme en objet JavaScript, une simple fonction `eval()` y parvient.

La notation JSON de l'encadré précédent, "Comprendre JSON", figure dans un fichier intitulé `phrasebook.json` ; le code suivant lit ce fichier en utilisant `XMLHttpRequest`, puis en extrait quelques données :

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
var XMLHttp = getXMLHttp();
XMLHttp.open("GET", "phrasebook.json");
XMLHttp.onreadystatechange = handlerFunction;
XMLHttp.send(null);

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    var json = XMLHttp.responseText;
    var book = eval("(" + json + ")");
    var pubdate = book.pubdate;
    var title = book.title;
```

```
    var publisher = book.publisher;
    window.alert(title + " by " + publisher +
                ➥" (" + pubdate + ")");
  }
}
</script>
```

Utiliser JSON pour la désérialisation des données (xmlhttpjson.html)

Attention

Il est généralement déconseillé d'utiliser `eval()`, car il introduit une vulnérabilité de sécurité importante si JSON provient d'une source non fiable. Du fait de la restriction liée au domaine identique de `XMLHttpRequest`, on peut généralement faire confiance au code JSON mais, pour plus de sécurité, téléchargez la bibliothèque JSON `json.js` à l'adresse http://www.json.org/js.html. Le code suivant remplace l'appel `eval()` :

```
var book = json.parseJSON();
```

Création d'un écran d'attente

```
document.getElementById("loading").style.visibility
   = "hidden";
```

L'un des plus gros obstacles pour les applications Web modernes est l'attente : une opération a lieu, mais en arrière-plan. Or, il vous faut en informer les utilisateurs, faute de quoi ils risquent de s'impatienter. On peut pour cela modifier le curseur de la souris (voir Chapitre 4, "CSS") ; il est aussi possible d'afficher une bannière d'attente. De nombreuses applications font graduellement disparaître une bannière contenant la mention "veuillez patienter" ou "chargement en cours" lors de l'exécution d'un appel `XMLHttpRequest` ; l'un des premiers sites Web à utiliser cela a été Google Mail.

En fait, cette expression nécessite plus d'éléments DHTML qu'AJAX. Lorsque l'appel (asynchrone !) est envoyé au serveur, l'écran de chargement s'affiche, il est placé dans le coin supérieur droit (vous pouvez bien sûr utiliser la position que vous préférez). Lorsque les données reviennent du serveur, la bannière disparaît à nouveau. Le code suivant implémente ceci, indépendamment du navigateur :

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
var XMLHttp = getXMLHttp();
window.onload = function() {
  XMLHttp.open("GET", "delay.php?" + Math.random());
  XMLHttp.onreadystatechange = handlerFunction;
  XMLHttp.send(null);
  with (document.getElementById("loading")) {
    style.visibility = "visible";
    if (navigator.appName ==
      "Microsoft Internet Explorer") {
      style.posLeft = document.body.clientWidth - 75;
      style.posTop = 0;
    } else {
      style.left = (window.innerWidth - 75) + "px";
      style.top = "0px";
    }
  }
}

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    document.getElementById("loading").style
    ➥.visibility = "hidden";
    window.alert("Returned data: " +
                ➥XMLHttp.responseText);
```

```
  }
}
</script>
<span id="loading" style="position: absolute;
   visibility: hidden; background-color: red;
   width: 75px;">Loading ...</span>
```

Implémentation d'un écran d'attente (waiting.html)

La Figure 11.3 présente le navigateur qui attend des résultats : la bannière apparaît (et disparaît à nouveau lorsque les données de la requête HTTP sont arrivées).

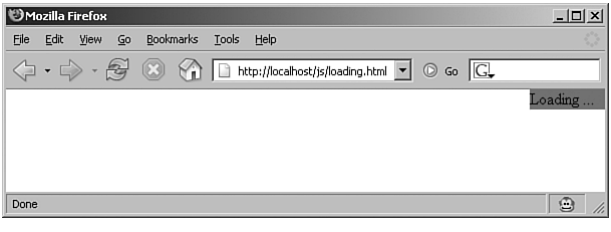

Figure 11.3 : L'écran d'attente.

Attention

L'objet `XMLHttpRequest` ne remarquera pas un éventuel problème avec un élément de votre requête HTTP, y compris avec les temporisations. Utilisez donc le motif présenté à la section "Interruption d'une requête HTTP" pour vérifier le statut de la requête après une certaine période. Si la requête ne s'est pas encore terminée, redémarrez-la ou affichez un message d'erreur et laissez disparaître la bannière d'attente.

Résolution du problème de signet

```
var data = unescape(location.hash.substring(1));
```

L'un des principaux problèmes rencontrés avec les applications AJAX actuelles réside dans le fait qu'il est impossible de placer un signet sur une page. Le contenu pouvant changer grâce aux appels `XMLHttpRequest`, mais l'URL restant la même, le signet ne fonctionne pas.

Il existe toutefois un contournement. Attention, ce n'est que l'emballage du code que vous devez écrire, le véritable travail à faire (selon votre application) consiste à faire perdurer les données, puis à les réappliquer à la page.

L'astuce se traduit par l'ajout des données qui identifient l'état actuel de la page à l'URL, non pas sous la forme d'un paramètre `GET` (puisque cela rechargerait la page), mais après le dièse :

```
http://server/file.html#info
```

Les données concernées ici par `info` doivent identifier le statut de la page en cours. Les détails de l'implémentation dépendent grandement de la manière dont AJAX est employé sur la page.

Dès que l'état actuel de la page change, le dièse (`location.hash` dans JavaScript) doit être mis à jour. L'extrait de code suivant lit ensuite ces informations une fois que la page est chargée. Vous devez implémenter la fonction `applyData()` qui s'occupe de transformer les informations de `location.hash` en contenu réel sur la page.

```
window.onload = function() {
  if (location.hash.length >= 2) {
    var data = unescape(location.hash.substring(1));
    applyData(data);
  }
};
```

Cela augmente bien entendu la quantité de travail à réaliser mais vos utilisateurs bénéficieront grandement de cette fonction commode. Malheureusement, cette méthode ne fonctionne pas (encore) avec Safari et Konqueror car ils gèrent `location.hash` d'une manière différente.

Attention

N'essayez pas de placer simplement JSON après le dièse, puis d'appeler `eval()` car un pirate pourrait, par malveillance, placer du code JavaScript dans l'URL qui serait ensuite exécutée dans le contexte de votre site. Ce type d'attaque est appelé *Cross-Site Scripting* (XSS) et se révèle très dangereux. Validez donc toujours les données avant de les utiliser.

Résolution du problème du bouton Précédent

```
window.frames["loader"].window.location.search =
  "?" + escape(data);
```

Il existe un problème lié au thème traité à la section précédente et qui concerne le bouton Précédent (et, bien entendu aussi le bouton Suivant). Lorsque l'URL de la page ne change pas mais que son contenu est modifié, le bouton Précédent ne fonctionne pas comme prévu.

Pour résoudre ce problème et, une fois de plus, cette section ne présente qu'une manière générique et non une solution complète (de nombreux détails dépendent de l'implémentation), il faut d'abord traiter deux sous-problèmes :

- Vérifier qu'au moment du chargement de la page, les informations figurant dans `location.hash` sont appliquées à la page (si vous appliquez la section précédente, ce travail a déjà été effectué).

- Vous assurer que les diverses pages du nouveau dièse se trouvent dans l'historique du navigateur, faute de quoi les boutons Précédent et Suivant ne fonctionneront pas. Cette tâche est effectuée automatiquement par les navigateurs Mozilla, mais pas par Internet Explorer.

La solution au second sous-problème consiste à utiliser un cadre intégré masqué, qui chargera une page invisible, dont le seul but est d'obtenir une entrée dans l'historique de la page. Voici le cadre intégré :

```
<iframe src="loader.html" name="loader"
  style="display:none"></iframe>
```

Dès qu'un événement survient sur la page (lié à AJAX), vous devez forcer le cadre de chargement à se charger à nouveau, en ajoutant des données à l'URL (à la différence du code précédent, ici, le rechargement est obligatoire) :

```
if (window.ActiveXObject) {
  window.frames["loader"].window.location.search =
    "?" + escape(data);
}
```

Enfin, le cadre de chargement doit appliquer toutes les données chargées au document principal ; l'opération est nécessaire, au moins pour Internet Explorer, car les boutons Précédent et Suivant y modifient le contenu du cadre intégré !

```
window.onload = function() {
  if (location.search.length >= 2) {
    var data = unescape(location.search.substring(1));
    top.applyData(data);
  }
};
```

Cela complique un peu plus la vie du programmeur, en particulier si plusieurs effets AJAX cohabitent sur la page. Mais, nous l'avons déjà dit, vos utilisateurs bénéficieront grandement de cette fonction. Et, une fois de plus, ce contournement ne fonctionne pas (encore) avec Safari et Konqueror.

XSLT

```
var process = new XSLTProcessor();
```

Les navigateurs Internet Explorer et Mozilla prennent en charge XSLT de JavaScript, selon des procédures très différentes. Pour Microsoft Internet Explorer, il vous faudra une fois de plus ActiveX (avec également Internet Explorer 7). Vous chargez d'abord le fichier XSLT, puis le XML (provenant bien sûr de `XMLHttpRequest`). Enfin, le résultat de la transformation XSL est disponible sous forme de chaîne et peut, par exemple, être annexé à la page.

Les navigateurs Mozilla, quant à eux, adoptent une autre méthode ; ils se reposent exclusivement sur un objet natif. Le résultat est différent de l'implémentation d'Internet Explorer. Au final, vous obtenez un fragment DOM, que vous pouvez ajouter au code DOM de la page en cours.

Comprendre XSLT

En 1999, le W3C a publié la spécification XSLT 1.0 (*eXtended Stylesheet Language Transformations*). Le sigle signifie *transformations XSL*. Il s'agit d'un langage fondé sur XML pour transformer des données sous une autre forme. La plupart du temps, les données produites le sont en HTML, mais d'autres formats sont aussi possibles. Sur le Web toutefois, le HTML est généralement préférable.

XSLT fonctionne avec des modèles, grâce auxquels vous accédez, entre autres choses, à des valeurs de nœud et d'attributs du fichier source XML. Le code suivant présente un exemple de fichier XSLT :

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<ul>
  <xsl:for-each select="books/book">
    <li>
      <xsl:value-of select="title" />
      by
      <xsl:value-of select="publisher" />
      (<xsl:value-of select="@pubdate" />)
    </li>
  </xsl:for-each>
</ul>
</xsl:template>
</xsl:stylesheet>
```

Le fichier XSLT (phrasebooks.xsl)

Le code suivant couvre les deux principaux types de navigateurs :

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script language="JavaScript"
  type="text/JavaScript">
var XMLHttp = getXMLHttp();
window.onload = function() {
  XMLHttp.open("GET", "phrasebooks.xml");
  XMLHttp.onreadystatechange = handlerFunction;
  XMLHttp.send(null);
}

function handlerFunction() {
  if (XMLHttp.readyState == 4) {
    var xml = XMLHttp.responseXML;
    if (window.ActiveXObject) {
      var xslt = new ActiveXObject(
      ➥"MSXML2.FreeThreadedDOMDocument");
      xslt.async = false;
      xslt.load("phrasebooks.xsl");
      var template = new ActiveXObject(
      ➥"MSXML2.XSLTemplate");
      template.stylesheet = xslt;
      var process = template.createProcessor();
      process.input = xml;
      process.transform();
      var para = document.createElement("p");
      para.innerHTML = process.output;
      document.body.appendChild(para);
    } else if (window.XSLTProcessor) {
      var xslt = document.implementation
       ➥.createDocument("", "", null);
      xslt.async = false;
      xslt.load("phrasebooks.xsl");
      var process = new XSLTProcessor();
```

```
      process.importStylesheet(xslt);
      var result = process.transformToFragment(
       ➥xml, document);
      document.body.appendChild(result);
    }
  }
}
</script>
```

Utilisation de XSLT avec JavaScript (xmlhttpxsl.html)

La Figure 11.4 montre le résultat de la transformation : les données du fichier XML sont présentées sous forme de liste à puces.

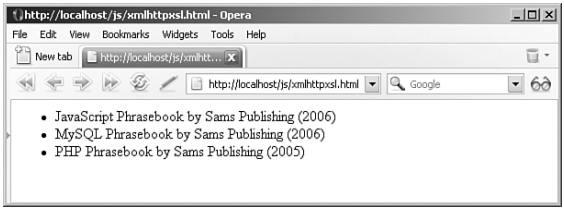

Figure 11.4 : Le résultat de la transformation XSL.

Comme le montre également la Figure 11.4, le navigateur Opera (à partir de la version 9) possède une implémentation XSLT compatible, utilisant la manière Mozilla.

Utilisation d'une bibliothèque XML

```
var result = xsltProcess(xml, xslt);
```

Vous le voyez, l'utilisation de XSLT à partir de JavaScript peut être assez compliquée lorsque l'objectif est de la faire fonctionner avec autant de navigateurs que possible.

Dans ce cas, pour gagner du temps et s'épargner des efforts, il est recommandé d'utiliser un cadre externe.

Google a créé une structure de ce type ("AJAXSLT"), disponible gratuitement (sous licence BSD) à l'adresse **http://goog-ajaxslt.sourceforge.net/**. La section suivante utilise la version 0.4 de la structure, que vous pouvez également découvrir à partir des chemins employés dans ce code. La structure elle-même ne fait pas partie des téléchargements de codes pour cet ouvrage.

L'expression suivante procède à la transformation XSLT mais repose cette fois-ci sur les fonctionnalités proposées par AJAXSLT.

```
<script language="JavaScript"
  type="text/JavaScript" src="xmlhttp.js"></script>
<script src="ajaxslt-0.4/misc.js"
  type="text/JavaScript"></script>
<script src="ajaxslt-0.4/dom.js"
  type="text/JavaScript"></script>
<script src="ajaxslt-0.4/xpath.js"
  type="text/JavaScript"></script>
<script src="ajaxslt-0.4/xslt.js"
  type="text/JavaScript"></script>
<script language="JavaScript"
  type="text/JavaScript">
var xml = null;
var xslt = null;
var XMLHttp1 = getXMLHttp();
var XMLHttp2 = getXMLHttp();
window.onload = function() {
  XMLHttp1.open("GET", "phrasebooks.xml");
  XMLHttp1.onreadystatechange = handlerFunction1;
  XMLHttp1.send(null);
  XMLHttp2.open("GET", "phrasebooks.xsl");
  XMLHttp2.onreadystatechange = handlerFunction2;
  XMLHttp2.send(null);
}
```

```
function handlerFunction1() {
  if (XMLHttp1.readyState == 4) {
    xml = xmlParse(XMLHttp1.responseText);
    if (xslt != null) {
      transform();
    }
  }
}

function handlerFunction2() {
  if (XMLHttp2.readyState == 4) {
    xslt = xmlParse(XMLHttp2.responseText);
    if (xml != null) {
      transform();
    }
  }
}

function transform() {
  var result = xsltProcess(xml, xslt);
  document.getElementById("output").innerHTML =
    result;
}
</script>
<div id="output"></div>
```

Utilisation de XSLT avec JavaScript et AJAXSLT (ajaxslt.html)

Cela fonctionne bien (vous le voyez, la majorité du code s'occupe du téléchargement des fichiers XML et XSL) mais l'opération marche aussi dans plusieurs navigateurs, comme le montre la Figure 11.5 (capturée dans Konqueror). Autre avantage, vous pouvez également utiliser XPath avec cette bibliothèque. Les navigateurs Internet Explorer et Mozilla et Opera fonctionnent avec XPath, mais de manière encore plus incompatible qu'avec XSLT. La prise en charge de AJAXSLT par XPath permet aussi de gérer des navigateurs plus anciens.

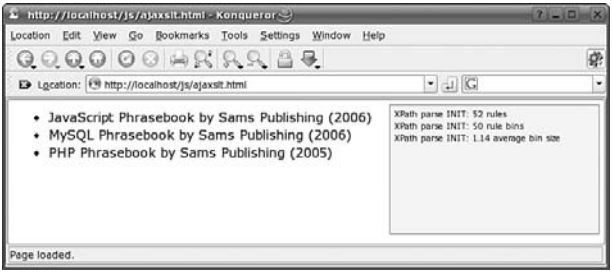

Figure 11.5 : Avec AJAXSLT, la transformation fonctionne même dans Konqueror.

Utilisation du service Web Yahoo!

```
<script language="JavaScript" type="text/JavaScript"
  src="http://api.search.yahoo.com/WebSearchService/V1/
  webSearch?appid=XXX&query=JavaScript&output=
  json&callback=showResults">
</script>
```

De plus en plus de services Web proposent une interface JSON ; les services Web Yahoo! ont d'ailleurs été parmi les premiers à le faire. Après une inscription (gratuite) à **http://api.search.yahoo.com/webservices/register_application**, vous recevez votre identifiant personnel. Il est lié à votre application et doit remplacer les `XXX` de cette expression. La balise précédente `<script>` appelle le service Web et attend du JSON en retour et fournit également le nom d'une fonction de rappel, appelée après que les données y sont placées. Votre application reçoit donc le code JavaScript du serveur Yahoo! et l'exécute, ce qui signifie que vous devez faire confiance à Yahoo! pour l'utiliser. La fonction de rappel récupère donc un objet JavaScript avec les résultats de la recherche de Yahoo!.

Le code suivant crée alors une liste à puces avec les données provenant du service Web. Vous trouverez de plus amples informations sur le format spécifique des données renvoyées de Yahoo! dans la documentation en ligne à l'adresse **http://developer.yahoo.com/search/Web/V1/webSearch.html**. La Figure 11.6 montre le résultat de ce code.

```
<script language="JavaScript"
  type="text/JavaScript">
function showResults(data) {
  var ul = document.getElementById("output");
  for (var i=0; i < data.ResultSet.Result.length;
     i++) {
    var text = document.createTextNode(
    ➥data.ResultSet.Result[i].Title + " - " +
    ➥data.ResultSet.Result[i].Url);
    var li = document.createElement("li");
    li.appendChild(text);
    ul.appendChild(li);
  }
}
</script>
<body>
  <p><ul id="output"></ul></p>
</body>
<script language="JavaScript" type="text/JavaScript"
  src="http://api.search.yahoo.com/WebSearchService/
     ➥V1/webSearch?appid=XXXXX&query=JavaScript&
     ➥output=json&callback=showResults">
</script>
```

Appel du service Web Yahoo! avec JavaScript (yahoowebservice.html)

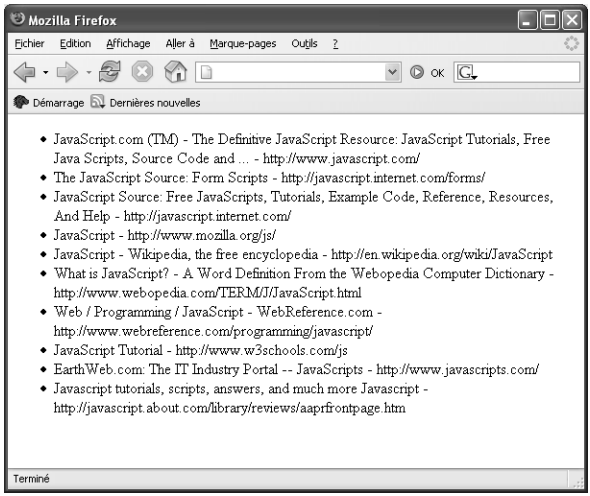

Figure 11.6 : Les résultats de la recherche Yahoo!.

Utiliser un cadre d'applications AJAX

Plus vous utilisez AJAX sur votre site Web, plus vous devez chercher les cadres d'applications AJAX disponibles, susceptibles de vous faciliter la tâche avec `XMLHttpRequest` et d'autres fonctions JavaScript avancées.

Le cadre d'applications à employer dépend largement de la technologie côté serveur. Il existe plusieurs cadres d'applications AJAX pour Java (comme Zimbra, à l'adresse **http://www.zimbra.com/**) et ASP.NET 2.0 (comme Atlas, à l'adresse **http://atlas.asp.net/**) ; PHP connaît trop de cadres d'applications pour que nous puissions les énumérer (par exemple le package PEAR à l'adresse **http://pear.php.net/package/HTML_AJAX**).

Il existe également des cadres d'applications AJAX assez intéressants, indépendants des technologies, et qui se concentrent plutôt sur l'aspect JavaScript. Dojo notamment (**http://www.dojotoolkit.org/**) est assez utile car il propose des contournements analogues à ceux mentionnés dans ce chapitre pour faire fonctionner les boutons Précédent et Suivant.

On considère que Tibet est l'une des plus anciens cadres d'applications (**http://www.technicalpursuit.com/**). Le travail a démarré (au moins du point de vue conceptuel) en 1997 et le produit est sorti en 1999.

Choisissez donc votre cadre d'applications avec mesure, selon votre technologie côté serveur : déterminez au préalable si vous utilisez exclusivement une technologie côté serveur ou si vous devez vous conformer aux choix de votre clientèle. Selon ces informations, les caractéristiques des divers cadres d'applications et notamment l'activité de développement (puisque de nombreux projets ne voient jamais le jour ou meurent au bout de quelques mois), votre choix se portera sur un cadre d'applications ; mais les sections de ce chapitre vous indiqueront souvent les éléments nécessaires pour transformer votre application Web en application AJAX.

12

Médias intégrés

HTML et JavaScript sont une combinaison à succès mais qui ne suffit pas toujours à créer une application Web moderne. HTML permet donc aux développeurs d'intégrer un contenu différent dans un site. Généralement, il faut pour cela disposer d'un plug-in pour navigateur. Ce chapitre montre comment accéder à ce contenu et propose plusieurs pointeurs vers les types de contenu les plus largement utilisés. Sachez que la plupart des codes de ce chapitre ne figurent pas dans les téléchargements de code.

Accès aux médias intégrés

`document.media`

L'élément HTML "officiel" qui permet d'intégrer les médias s'intitule `<object>`. Toutefois, pour que le site fonctionne dans plusieurs navigateurs, il est généralement conseillé d'intégrer un objet `<embed>` dans l'élément `<object>`.

Et pour que l'accès à l'objet intégré soit aussi simple que possible, l'attribut `id` de l'élément `<object>` doit correspondre à l'attribut `name` de l'objet `<embed>` :

```
<object classid="..." id="media">
  <embed name="media"></embed>
</object>
```

Vous pouvez ensuite accéder aux médias intégrés en utilisant le raccourci figurant au début de cette section. Bien entendu, `document.getElementById()` sert aussi à accéder à l'élément `<object>` ; tous les éléments `<embed>` se trouvent dans le tableau `document.embeds`. Vous le voyez, le raccourci est beaucoup plus pratique ici.

Vérification des modules complémentaires dynamiques

```
if (new ActiveXObject(
  "ShockwaveFlash.ShockwaveFlash.8")) { }
```

Dans les navigateurs Mozilla, l'URL spéciale `about:plugins` propose la liste de tous les modules complémentaires (*plug-in*) disponibles, comme le montre la Figure 12.1 ; cela fonctionne également dans Opera.

Selon le navigateur choisi, il existe deux manières pour vérifier activement la présence d'un module complémentaire donné :

- Dans les navigateurs Mozilla et Opera (qui utilisent la même API), `navigator.plugins` donne un tableau contenant tous les modules complémentaires. Chaque élément indique le nom, le nom de fichier et la description de chaque module. De même, `navigator.mimeTypes` propose une liste de tous les types MIME pris en charge et acceptés par le module.

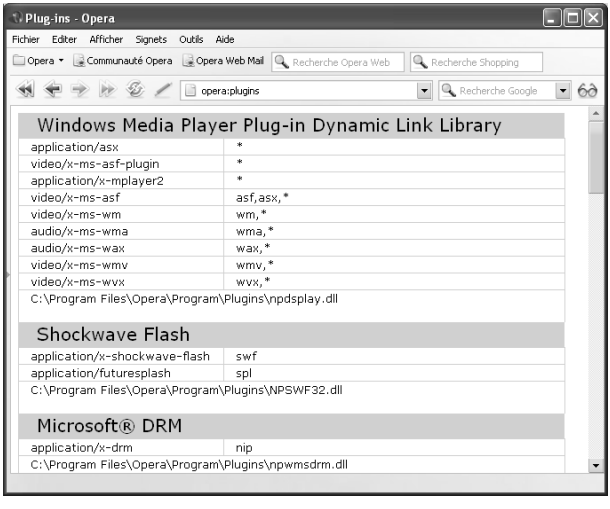

Figure 12.1 : Opera présente une liste de tous les modules complémentaires installés.

- Sous Internet Explorer (Windows uniquement), vous pouvez essayer d'instancier la classe associée, présentée par le module complémentaire, en utilisant le constructeur `ActiveXObject`.

Le code suivant tente de détecter le module complémentaire Flash. Pour Internet Explorer, on fait appel au nom de l'objet COM associé, pour les autres navigateurs, le code recherche un module appelé "Shockwave Flash".

```
<script language="JavaScript" type="text/JavaScript">
var hasFlash = false;
if (window.ActiveXObject) {
  try {
    if (new ActiveXObject(
      "ShockwaveFlash.ShockwaveFlash.8")) {
```

```
      hasFlash = true;
    }
  } catch (e) {
  }
} else if (navigator.plugins) {
  if (navigator.plugins["Shockwave Flash"]) {
      hasFlash = true;
  }
}
window.alert(hasFlash);
</script>
```

Détection du module complémentaire Flash (plugin.html)

Attention

Le tableau `navigator.plugins` est également disponible sous Internet Explorer, mais il est vide. Vous devez donc d'abord rechercher `window.ActiveXObject`, puis `navigator.plugins`. L'opération ne fonctionne pas dans l'autre sens.

Gestion des versions récentes d'Internet Explorer

Une décision de justice a obligé Microsoft à sortir une mise à jour de son navigateur en mars et avril 2006, modifiant le comportement des médias intégrés. Si l'utilisateur "n'active pas" les données intégrées en cliquant dessus, de nombreux événements ne se déclenchent pas et d'autres restrictions apparaissent. La Figure 12.2 montre les informations présentées par le navigateur dans ce cas.

Toutefois, à l'adresse **http://msdn.microsoft.com/workshop/author/dhtml/overview/activating_activex.asp**, Microsoft propose un contournement. L'idée consiste à

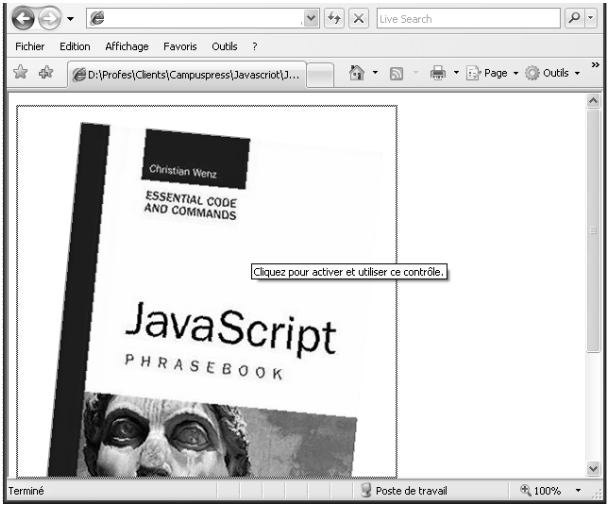

Figure 12.2 : Les médias intégrés (ici, une animation Flash) sont inactifs.

utiliser JavaScript pour inclure l'élément `<object>` sur la page. Les précisions techniques apportées par la décision de justice stipulent que le code JavaScript qui crée (*via* la méthode `document.write()`) l'élément `<object>` doit résider dans un fichier externe. Il faut donc d'abord utiliser un élément `<script>` de la manière suivante :

```
<script language="JavaScript"
  type="text/JavaScript" src="externalFile.js">
</script>
```

puis produire, dans le fichier `externalFile.js`, le marquage qui charge le fichier média intégré :

```
document.write("<object classid=\"...\" ...></object>");
```

Astuce

Bien entendu, le média intégré ne fonctionne que lorsque JavaScript est activé, vous devez donc également disposer d'une solution dans un élément `<noscript>`. Du fait du brevet logiciel à l'origine de la décision de justice, il est également possible que d'autres fournisseurs de navigateurs aient à implémenter un nouveau comportement du même type pour les médias intégrés.

Accès au contenu multimédia

L'accès au contenu multimédia (données audio ou vidéo) dans le navigateur est assez difficile. Il existe d'ailleurs de nombreux modules complémentaires, et chaque version semble modifier un élément par rapport à son prédécesseur. Pour obtenir des informations actualisées ou spécifiques à une version concernant l'accès aux divers modules, consultez le site Web du fabricant.

Le système présente tout de même de bons côtés, les principaux modules complémentaires liés aux données multimédias, actuellement Windows Media Player, Real Player et Apple QuickTime présentent des noms de méthode assez similaires. Ainsi, lorsque vous avez accédé au module complémentaire, comme cela est indiqué au début du chapitre, vous obtenez, grâce aux méthodes du Tableau 12.1, la fonctionnalité essentielle dont vous avez besoin pour travailler avec les médias intégrés.

Tableau 12.1 : Méthodes pour la fonctionnalité multimédia

Action	Windows Media Player	Real Player	QuickTime
Lecture	`play()`	`DoPlay()`	`play()`
Arrêt	`stop()`	`DoStop()`	`stop()`
Pause	`pause()`	`DoPause()`	`pause()`
Volume	`volume`	`GetVolume()/ SetVolume()`	`GetVolume()/ SetVolume()`

Accès au contenu Java

```
document.JavaApplet.setString(field.value);
```

On a imaginé pendant un temps que Java était le format Web dominant mais sa place est de nos jours plutôt en arrière-plan. Toutefois, vous rencontrerez toujours des applets Java intégrées dans les pages Web, même si le module complémentaire Java n'a pas la même présence que son homologue Flash.

Lorsqu'une applet Java est intégrée dans une page, il est possible d'appeler toutes ses méthodes publiques. L'applet suivante, très simple, présente sa méthode `setString()`, qui définit une variable privée, puis se redessine :

```
import java.applet.*;
import java.awt.*;

public class JavaApplet extends Applet
{
  private String _s = "Welcome!";
```

```
  public void paint(Graphics g) {
    g.setColor(Color.black);
    g.drawString(_s, 20, 20);
  }

  public void setString(String s) {
    this._s = s;
    this.repaint();
  }
}
```

Une applet Java simple (JavaApplet.java)

Une fois cette applet compilée dans une classe, le code JavaScript suivant appelle la méthode `setString()` dès que le contenu du champ de texte est modifié. La Figure 12.3 montre le résultat.

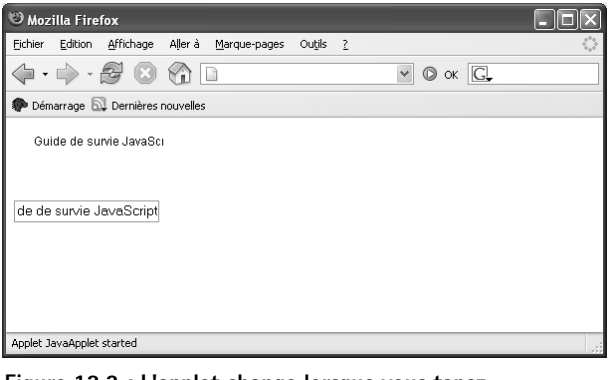

Figure 12.3 : L'applet change lorsque vous tapez.

```
<script language="JavaScript"
  type="text/JavaScript">
function paint(field) {
  document.JavaApplet.setString(field.value);
}
</script>
<applet code="JavaApplet.class" name="JavaApplet"
  width="150" height="75"></applet><br />
<input type="text" onkeyup="paint(this);" />
```

Accès à Java avec JavaScript (java.html)

Accès au contenu Flash

```
document.flash.Play();
document.flash.Rewind();
document.flash.StopPlay();
```

Les animations Flash peuvent être contrôlées *via* du JavaScript. Entre autres choses, vous pouvez démarrer, arrêter et mettre en pause l'animation. Le module complémentaire présente donc les méthodes `Play()`, `Rewind()` et `StopPlay()`. Le listing suivant implémente une simple console de contrôle pour l'animation Flash.

```
<object
  classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
  codebase="http://fpdownload.macromedia.com/pub/
  ➥shockwave/cabs/flash/swflash.cab #version=7,0,0,0"
  width="400" height="550" id="flash">
  <param name="allowScriptAccess"
    value="sameDomain" />
  <param name="movie" value="flash.swf" />
```

```
  <embed src="flash.swf" width="400" height="550"
    name="flash"
    allowScriptAccess="sameDomain"
    type="application/x-shockwave-flash"
    pluginspage="http://www.macromedia.com/go/
      ➥getflashplayer" />
</object>
<a href="JavaScript:document.flash.Play();">
     Play</a>
<a href="JavaScript:document.flash.Rewind();">
     Stop</a>
<a href="JavaScript:document.flash.StopPlay();">
     Pause</a>
```

Accès à une animation Flash à partir de JavaScript (flash.html

Info

Les options **allowScriptAccess** définies à la fois dans les éléments **<object>** et **<embed>** permettent à JavaScript d'accéder réellement à l'animation Flash.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

LE GUIDE DE SURVIE

JavaScript®

L'ESSENTIEL DU CODE ET DES COMMANDES

Ce *Guide de survie* vous propose l'ensemble du code JavaScript, Ajax, DHTML et CSS dont vous avez besoin pour créer des applications Web, des pages et des sites interactifs, rapidement et efficacement.

CONCIS ET MANIABLE

Facile à transporter, facile à utiliser — finis les livres encombrants !

PRATIQUE ET FONCTIONNEL

Plus de 100 séquences de codes personnalisables pour profiter au maximum de JavaScript et d'Ajax.

Christian Wenz est un rédacteur professionnel, mais aussi auteur, formateur et consultant spécialisé dans les technologies Web, qui s'enorgueillit d'avoir utilisé JavaScript avant qu'il ne soit à la mode et Ajax bien avant que le terme ne soit inventé !

Niveau : Intermédiaire
Catégorie : Web
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 : 978-2-7440-4003-0

9 782744 040030