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.

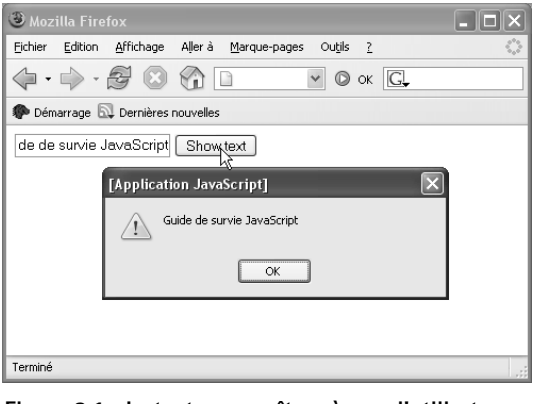

Figure 8.1 : Le texte apparaît après que l'utilisateur a cliqué sur le bouton.

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.

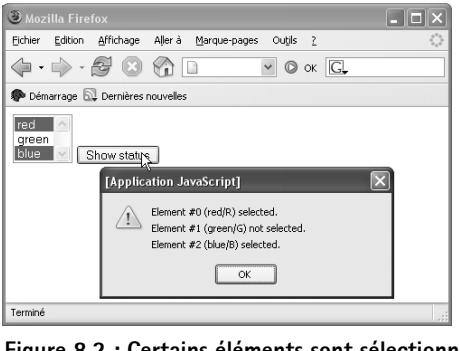

Figure 8.2 : Certains éléments sont sélectionnés.

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.

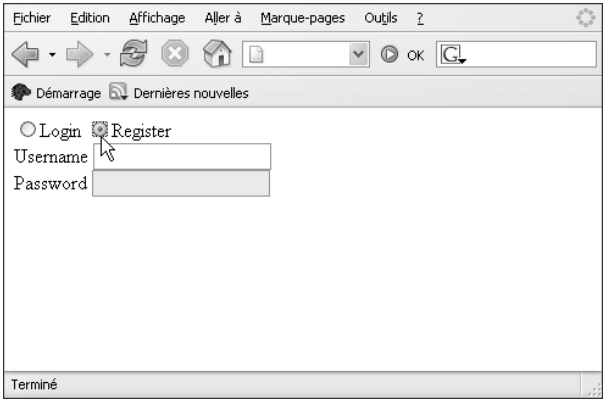

Figure 8.4 : Le champ de mot de passe est maintenant désactivé.

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.

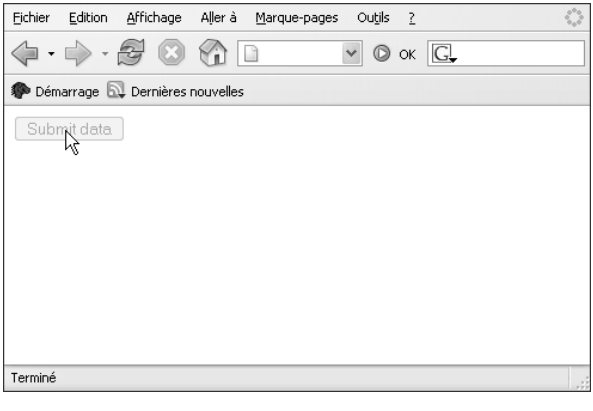

Figure 8.5 : Il est impossible de cliquer deux fois sur le bouton.

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.

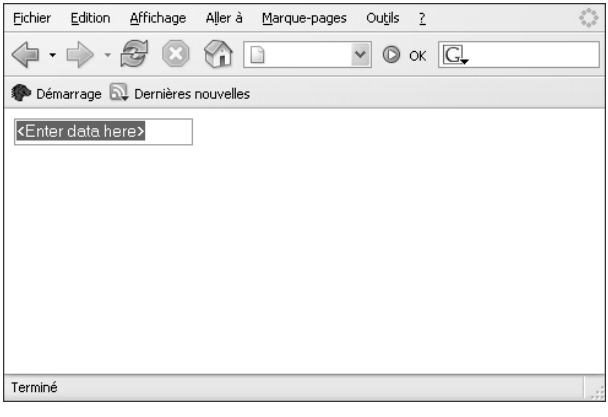

Figure 8.6 : Le texte est sectionné et le champ a le focus.

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.

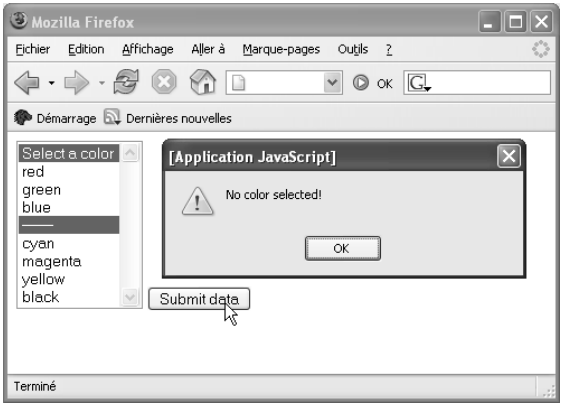

Figure 8.7 : Le formulaire ne peut être envoyé que si un élément "réel" a été sélectionné.

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.

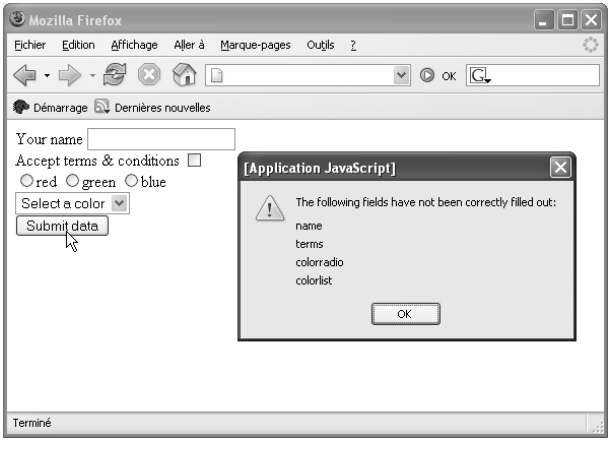

Figure 8.8 : Le formulaire est automatiquement validé.

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.

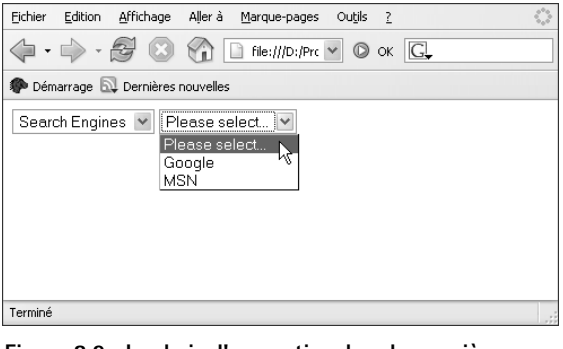

Figure 8.9 : Le choix d'une option dans la première liste conditionne le contenu de la seconde.

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.

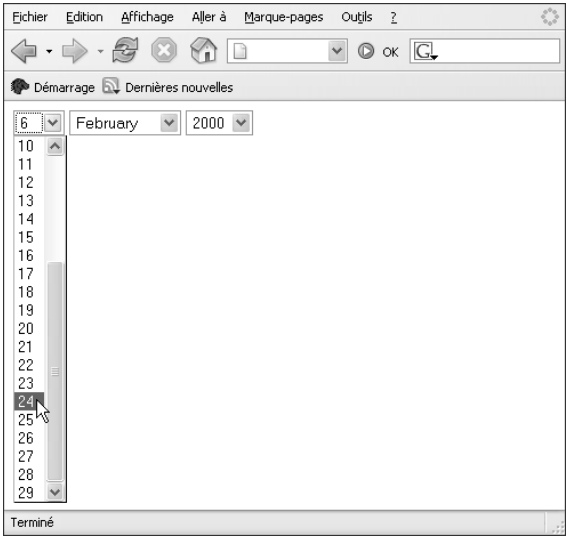

Figure 8.10 : La liste des jours est actualisée en fonction du mois et de l'année.