Table des matières

Partie I – Introduction à Ajax

Partie IV – Ressources sur les technologies associées

3

Comment fonctionne Ajax ?

Ajax, un amalgame de technologies

Des ingrédients déjà opérationnels

Contrairement à ce que l'on pourrait croire, Ajax n'est pas une technologie spécifique et innovante mais une conjonction de plusieurs technologies anciennes. Ainsi, les applications Ajax utilisent en général tout ou partie des technologies suivantes :

- Les feuilles de styles CSS qui permettent d'appliquer une mise forme au contenu d'une page XHTML.
- Le DOM qui représente la hiérarchie des éléments d'une page XHTML.
- L'objet XMLHttpRequest de JavaScript qui permet d'assurer des transferts asynchrones (ou quelquefois synchrones) entre le client et le serveur.
- Les formats de données XML ou JSON utilisés pour les transferts entre le serveur et le client.
- Le langage de script client JavaScript qui permet l'interaction de ces différentes technologies.

L'intérêt pour Ajax d'utiliser ces différentes technologies est qu'elles sont déjà intégrées dans la plupart des navigateurs actuels. Elles sont donc immédiatement exploitables – même si quelques différences d'implémentation subsistent d'un navigateur à l'autre.

Ceci représente une véritable aubaine pour les développeurs lorsqu'on connaît les atouts d'Ajax ; et on comprend mieux pourquoi toujours plus de développeurs se rallient à cette technologie.

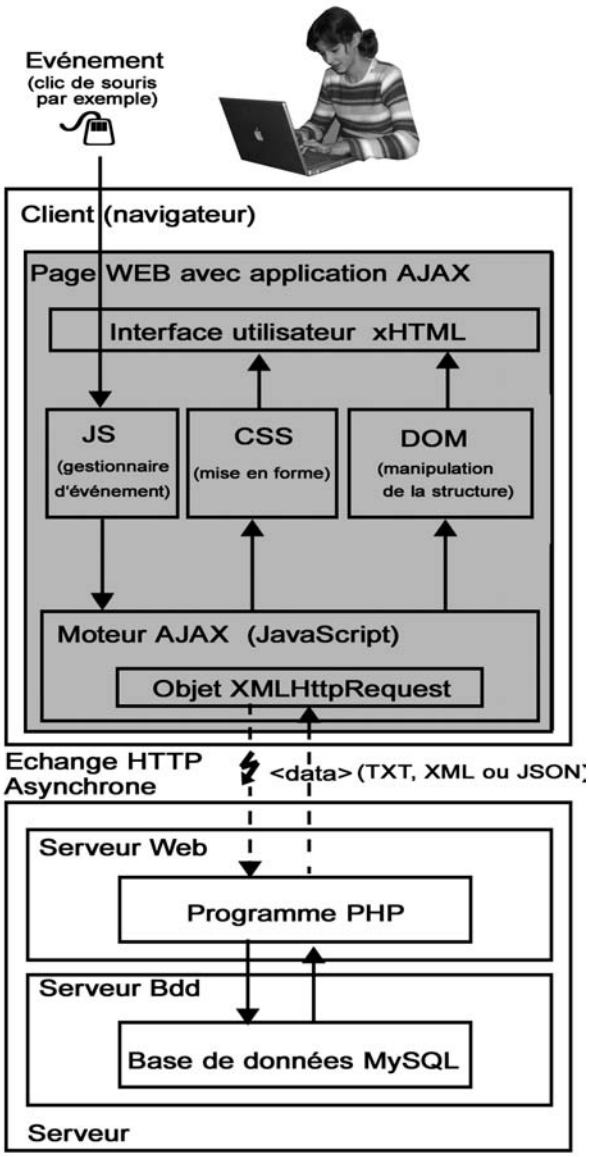
JavaScript, le ciment des fondations d'Ajax

Pour que ces différentes technologies sous-jacentes puissent être exploitées, il faut disposer d'un langage de script capable de les manipuler. Évidemment, dans ce contexte client,

JavaScript est la technologie idéale pour remplir cette mission et faire interagir toutes ces technologies entre elles. Ainsi, dans chaque application Ajax, nous retrouverons un programme JavaScript qui constituera le « moteur » du système, orchestrant à la fois les transferts de données avec l'aide de l'objet XMLHttpRequest et l'exploitation des réponses du serveur en agissant sur les CSS (pour modifier la mise en forme de la page XHTML) et sur le DOM (pour modifier le contenu ou la structure de la page XHTML) (voir figure 3-1).

Figure 3-1

Organisation des principaux composants d'Ajax



En ce qui concerne les données échangées, plusieurs formats peuvent être utilisés selon l'organisation et la complexité des flux d'informations. Les applications les plus simples pourront se contenter de données au format texte (simples couples variable/valeur) alors que les systèmes plus complexes devront choisir de structurer leurs données en XML (le DOM assurant ensuite l'insertion des données XML dans la page XHTML) ou encore dans un format issu de la structure des objets JavaScript, le JSON. À noter que la plupart des requêtes envoyées vers le serveur utilisent le format texte (les couples variable/valeur suffisent dans la majorité des cas), mais sachez qu'elles peuvent éventuellement aussi exploiter les formats XML ou JSON, de la même manière que les résultats retournés par le serveur au navigateur.

Comparatif avec les applications Web traditionnelles

Pour bien comprendre le fonctionnement et connaître les avantages d'un nouveau système, une bonne méthode consiste à le comparer avec l'existant que l'on connaît déjà. Dans cette partie, nous allons utiliser cette méthode en comparant le fonctionnement d'une application Ajax avec celui d'un site Web statique et celui d'un site Web dynamique.

Fonctionnement d'une application Web statique

Avec un site Web statique, la seule interactivité dont dispose l'internaute est de pouvoir passer d'une page HTML à l'autre par un simple clic sur les liens hypertextes présents sur une page. À chaque fois que l'internaute clique sur un lien, une requête HTTP est envoyée, établissant du même coup une communication avec le serveur. Cette communication est de type synchrone, c'est-à-dire que dès l'émission de la requête, la communication reste en place jusqu'à la réception de la réponse du serveur. Pendant le temps de traitement de la requête, le navigateur reste figé, bloquant ainsi toute action possible de l'internaute.

À chaque requête, le serveur retournera une réponse sous la forme d'une page HTML complète. S'il s'agit d'une simple requête, suite à la saisie par l'internaute de l'URL spécifique d'une page dans la barre d'adresse du navigateur ou, plus couramment, lorsque l'internaute clique sur un lien hypertexte, le serveur se contentera de renvoyer la page HTML demandée, ce qui clôturera le traitement côté serveur et débloquera ainsi le navigateur.

Fonctionnement d'une application Web dynamique

Nous avons vu précédemment le traitement d'une simple requête par le serveur mais d'autres cas peuvent se produire, notamment lors de l'envoi d'un formulaire. Dans ce cas, la requête est constituée d'une ligne de requête (précisant la méthode utilisée et le protocole HTTP), d'un corps (qui contient les données envoyées au serveur dans le cas d'une requête émise avec la méthode POST) et d'une série d'en-têtes qui définissent les spécificités de la requête (nature du navigateur utilisé, type d'encodage…) qui permettront au serveur de traiter correctement les informations. En général, lors de l'envoi d'un formulaire, le traitement côté serveur est réalisé par une page contenant un programme (en PHP par exemple). Les données réceptionnées pouvant être traitées directement par le programme ou entraîner un échange avec un

serveur de base de données afin de les mémoriser ou d'émettre une requête SQL. À l'issue de ce traitement, une nouvelle page HTML sera construite à la volée et renvoyée au navigateur, ce qui clôturera le processus, débloquant le navigateur de la même manière qu'avec un site statique.

Fonctionnement d'une application Ajax

Dans le cas d'une application Ajax, si la page contenant la structure XHTML et ses scripts client (moteur Ajax, gestionnaire d'événements…) est chargée de la même manière que pour un site statique, il n'en est pas de même pour les interactions qui suivent entre le navigateur et le serveur. Le moteur Ajax une fois chargé dans le navigateur restera en attente de l'événement pour lequel il a été programmé. Pour cela, un gestionnaire d'événements JavaScript est configuré pour appeler le moteur dès l'apparition de l'événement concerné. Lors de l'appel du moteur, un objet XMLHttpRequest est instancié puis configuré, une requête asynchrone est ensuite envoyée au serveur. À la réception de celle-ci, le serveur démarrera son traitement et retournera la réponse HTTP correspondante. Cette dernière sera prise en charge par la fonction de rappel du moteur Ajax qui exploitera les données pour les afficher à un endroit précis de l'écran.

Chronogrammes des échanges client-serveur

Une des grandes différences entre une application Web traditionnelle et une application Ajax est liée à l'échange asynchrone de données entre le navigateur et le serveur. Pour vous permettre de bien appréhender la différence entre ces deux applications, nous vous proposons de les comparer maintenant à l'aide de leur chronogramme.

Chronogramme d'une application Web dynamique traditionnelle

Lorsqu'un utilisateur sollicite le serveur dans une application Web dynamique traditionnelle (en envoyant un formulaire ou en cliquant sur une URL dynamique), il déclenche une requête HTTP dans laquelle sont imbriqués les paramètres de la demande. À partir de ce moment, le navigateur se fige jusqu'à la réception de la réponse HTTP du serveur, interdisant ainsi à l'utilisateur toute action pendant le temps de traitement de la requête. Dès la réception de la requête, le serveur Web analysera les paramètres et traitera la demande selon son programme. Il pourra interroger un serveur de base de données pour recueillir des données complémentaires si nécessaire. Une fois le traitement terminé, une page HTML complète sera construite à la volée, incluant les résultats du traitement après leur mise en forme. Cette page sera alors retournée au navigateur après son intégration dans le corps de la réponse HTTP. À la réception de la réponse HTTP, le navigateur interprétera la page HTML, comme lors de l'appel d'une page Web dans un site statique, et l'affichera à l'écran, entraînant le rechargement complet de la page. À la fin du chargement de la page, le navigateur est débloqué et l'utilisateur reprend la main sur l'application. Il pourra ainsi éventuellement réitérer une nouvelle demande serveur qui suivra le même cycle de traitement que celui que nous venons de décrire (voir figure 3-2).

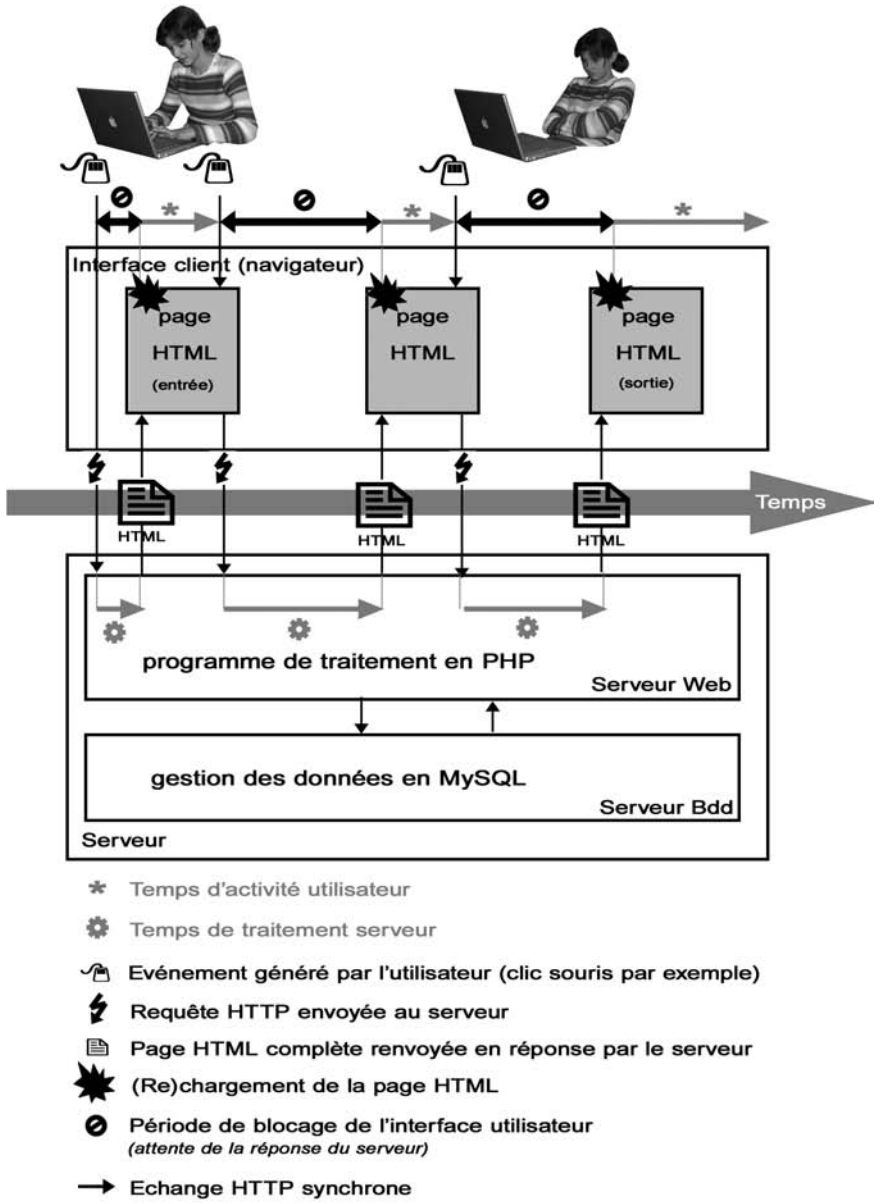


Figure 3-2

Chronogramme des échanges client-serveur d'une application traditionnelle

Chronogramme d'une application Ajax en mode asynchrone

Dans le cas d'une application Ajax en mode asynchrone, le déroulement du traitement est différent. À noter que l'objet XMLHttpRequest peut aussi envoyer des requêtes synchrones, mais dans ce cas le fonctionnement serait semblable à celui d'une application Web dynamique traditionnelle comme celle que nous avons décrite précédemment.

Dans une application Ajax, l'utilisateur doit commencer par appeler la page HTML contenant le moteur Ajax. Une fois la page chargée dans le navigateur, les échanges avec le serveur seront contrôlés par l'application Ajax (voir figure 3-3). L'envoi d'une requête est souvent déclenché par un gestionnaire d'événements JavaScript, mais il peut aussi être généré par un script de temporisation pour actualiser des informations à intervalles réguliers. Quel que soit le mode de déclenchement, le moteur Ajax est appelé par le biais d'une fonction JavaScript. La première action du moteur est la création d'un objet XMLHttpRequest immédiatement suivi de sa configuration (choix de la méthode de transfert GET ou POST, choix du fichier serveur sollicité, activation du mode asynchrone, désignation d'une fonction de rappel, intégration des paramètres…). Une fois l'objet configuré, l'envoi de la requête est déclenché, générant une requête HTTP semblable à celle créée avec une application dynamique traditionnelle. Toutefois, dans le cas de l'envoi d'une requête Ajax, le navigateur n'est pas bloqué et l'utilisateur peut continuer à utiliser son interface comme bon lui semble ; le transfert est asynchrone. Côté serveur, les paramètres seront analysés et le programme pourra aussi solliciter un serveur de base de données si besoin. Mais, contrairement à une application dynamique traditionnelle, le corps de la réponse HTTP retournée au navigateur ne sera pas composé de la page HTML complète : il contiendra seulement les données réclamées par le client. Lorsque le navigateur reçoit la réponse, une fonction de rappel, programmée lors de l'envoi de la requête, se chargera de récupérer les données placées dans le corps de la réponse HTTP, de les mettre en forme et de les insérer dans une zone particulière de la page Web et cela sans nécessiter le rechargement de la page (voir figure 3-3).

Les avantages d'Ajax

Économie de la bande passante

Avec Ajax, il n'est plus nécessaire de renvoyer le contenu entier de la page HTML à chaque requête, car l'objet XMLHttpRequest assure la récupération et l'insertion dans la page en cours des seules données à modifier. Ce système permet d'éliminer le transfert de nombreuses informations redondantes, allégeant ainsi fortement le trafic réseau entre le serveur Web et le client (navigateur).

Empêche le rechargement de la page à chaque requête

Le traitement traditionnel d'une requête HTTP entraîne à chaque retour de la réponse du serveur un rechargement complet de la page en cours. Hormis le désagréable « trou blanc » que cela engendre, ce phénomène allonge le temps de traitement d'une requête aux dépens de la réactivité de l'application.

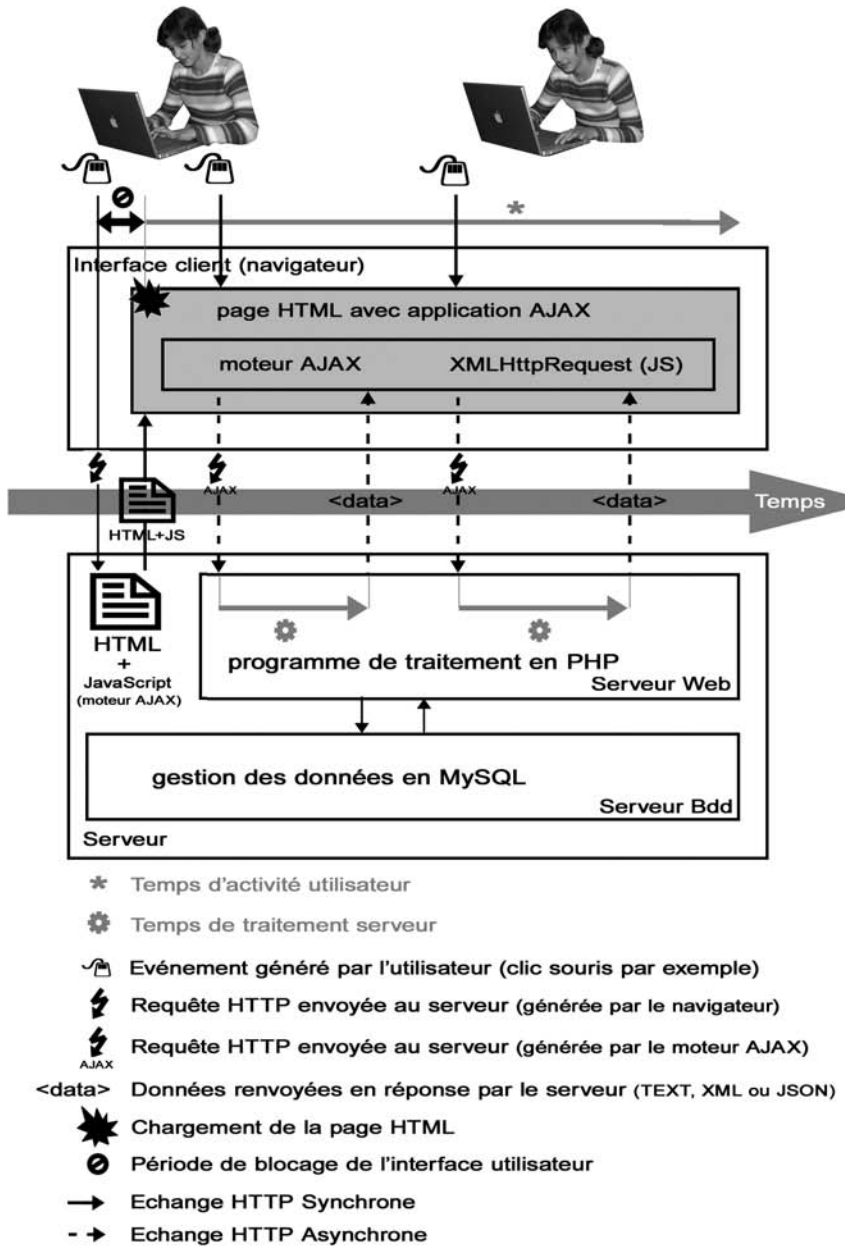


Figure 3-3

Chronogramme des échanges client-serveur d'une application Ajax

Évite le blocage de l'application pendant le traitement de la requête

Contrairement au simple échange HTTP d'une application traditionnelle, dans laquelle l'application cliente est bloquée pendant tout le temps d'attente de la réponse du serveur, l'échange XMLHttpRequest asynchrone d'une application Ajax permet à l'internaute de continuer à travailler pendant le temps de traitement de la requête. Cela ouvre des possibilités nouvelles pour le développement Web, permettant ainsi aux développeurs de créer des applications dont le mode de fonctionnement se rapproche de celui des applications disponibles jusqu'alors sur des ordinateurs de bureau.

Augmente la réactivité de l'application

Les données renvoyées par le serveur étant plus légères (le serveur retournant uniquement les données demandées et non la page HTML entière) et le rechargement de la page complète n'ayant plus lieu à chaque requête, cela améliore considérablement la réactivité du système. De plus, le chargement progressif des données couplé à une méthode prédictive permet de disposer de fonctionnalités graphiques avancées (déplacement d'une carte à l'aide de la souris dans une application de cartographie en ligne par exemple) jusqu'alors réservées aux logiciels autonomes de bureau.

Améliore l'ergonomie de l'interface

Une interface Ajax peut être composée de multiples zones ayant une gestion du contenu indépendante l'une de l'autre. Chaque zone pouvant déclencher ses propres requêtes, il est désormais possible d'avoir une mise à jour ciblée des contenus. Ainsi, grâce aux technologies DHTML associées à Ajax, l'utilisateur peut aménager librement ses différentes zones par un simple glisser-déposer et améliorer l'ergonomie de son interface Web.

Les inconvénients d'Ajax

Pas de mémorisation des actions dans l'historique

Le principal inconvénient d'une application Ajax est lié au fait que les actions de l'utilisateur ne sont pas mémorisées dans l'historique du navigateur. En effet, les différents contenus d'une application Ajax s'affichant toujours dans la même page, ils ne peuvent pas être enregistrés dans l'historique du navigateur comme le seraient les différentes pages HTML d'une application Web traditionnelle.

Par voie de conséquence, les boutons Suivant et Précédent ne sont plus utilisables car ils s'appuient sur l'historique du navigateur pour trouver la page suivante ou précédente. Ceci est évidemment très handicapant pour les internautes qui ont l'habitude d'utiliser ces boutons pour naviguer d'une page à l'autre.

Il existe néanmoins des solutions pour remédier à ce problème en couplant l'application Ajax avec un système d'iframe comme le fait Google dans plusieurs de ses applications Ajax mais cela nécessite un traitement supplémentaire qui complexifie le développement.

Problème d'indexation des contenus

Les différents contenus d'une application Ajax s'affichant dans une seule et même page, les moteurs de recherche pourront indexer uniquement le premier contenu par défaut de la page et non tous les contenus proposés par l'application.

D'autre part, le rappel des différents contenus d'une application Ajax par le biais des favoris sera confronté au même problème. Seul le contenu de la première page pourra être mémorisé dans les signets du navigateur.

Dépendance de l'activation de JavaScript sur le navigateur

Les applications Ajax utilisant JavaScript pour interagir entre les différentes technologies exploitées côté client (CSS, DOM, XML…) sont donc dépendantes de l'activation de JavaScript sur le navigateur, au même titre que tous les autres programmes clients utilisant cette technologie.

Même si les internautes qui désactivent JavaScript se raréfient, il faut toutefois prévoir une version dégradée de l'application en prévision des navigateurs qui ne supporteraient pas ce langage de script.

Les cadres cachés, une solution alternative à Ajax

Dans le chapitre précédent, nous avons cité d'autres technologies estampillées Web 2.0 (Flash + Flex, application Java) permettant la mise en œuvre d'une application Internet riche (RIA). Nous avions cependant écarté ces solutions car elles ne pouvaient pas fonctionner sur un navigateur sans l'installation d'un plug-in.

Il existe néanmoins une technique nommée « cadre caché » (frameset HTML ou iframe) utilisée bien avant celle de l'objet XMLHttpRequest qui permet d'établir des communications en arrière plan avec le serveur et qui, comme Ajax, ne nécessite pas l'ajout d'un plug-in.

La technique du cadre caché

Cette technique exploite la structure des jeux de cadres HTML dont l'un d'entre eux est invisible et sert de pont pour établir une communication avec le serveur. Le cadre caché est rendu invisible en configurant sa largeur et sa hauteur à zéro pixel. Avec cette technique, il est alors possible d'envoyer des requêtes serveur par le biais du cadre caché sans perturber l'écran de l'utilisateur.

Pour illustrer le fonctionnement de cette technique, nous allons détailler le cycle d'une communication complète entre le navigateur et le serveur. Pour commencer, l'utilisateur déclenche une fonction JavaScript depuis le cadre visible. Cette fonction appellera un script serveur dont le retour sera assigné au cadre caché. Le script serveur analyse alors les paramètres communiqués et traite la demande. Il renvoie ensuite en réponse au cadre caché une page HTML complète contenant le résultat dans une balise `<div>`. Dans cette même page HTML se trouve une fonction JavaScript qui sera invoquée dès que la page sera complètement chargée

dans le cadre caché (avec le gestionnaire d'événements `window.onload` par exemple). Enfin, lorsque la fonction JavaScript s'exécute dans le cadre caché, elle récupère le résultat inséré préalablement dans la balise `<div>` de la même page et l'affecte à une zone définie du cadre visible. L'utilisateur peut alors voir la réponse apparaître dans la page visible du navigateur et cela tout en continuant d'utiliser l'interface pendant le traitement serveur évitant ainsi que la page ne soit rechargée.

Depuis l'apparition des iframes (introduites dans la version 4.0 du HTML), il est possible d'exploiter la même technique mais sans avoir à utiliser la structure contraignante des framesets. En effet, l'iframe peut être placé dans une page HTML traditionnelle et permet de créer ainsi un cadre dans n'importe quelle page existante. Il est même possible de créer des iframes à l'aide d'un programme JavaScript, ce qui permet de mieux contrôler la création et la suppression des flux de communication entre le serveur et le navigateur.

Avantages des cadres cachés

Fonctionne sur les anciens navigateurs

Cette technique étant pratiquée depuis longtemps, elle peut être utilisée sur des navigateurs plus anciens qui ne supportaient pas encore les objets XMLHttpRequest. Il est donc possible d'utiliser la technique des cadres cachés en tant que solution alternative à Ajax si l'on désire que l'application fonctionne sur une plus grande variété de navigateurs.

Conserve l'historique du navigateur

La technique des cadres cachés permet de conserver l'historique du navigateur. Cette caractéristique permet aux internautes de continuer à utiliser les boutons Suivant et Précédent du navigateur contrairement aux applications Ajax. À noter que certaines applications couplent Ajax à la technique des cadres cachés pour remédier au problème des boutons Suivant et Précédent inactifs (comme Gmail et Google Maps par exemple).

Inconvénients des cadres cachés

Manque d'informations sur le traitement de la requête

Le principal inconvénient de la technique des cadres cachés est lié au manque d'informations concernant le traitement de la requête HTTP en arrière-plan. Cela pose de gros problèmes dans le cas où la page du cadre caché n'est pas chargée, car l'internaute peut attendre indéfiniment la réponse sans être informé de l'incident. Même s'il est possible de programmer une temporisation pour interrompre le traitement et informer l'utilisateur au bout d'un temps déterminé, il est préférable désormais d'utiliser un objet XMLHttpRequest qui nous permet de garder le contrôle de toutes les étapes du traitement de la requête HTTP.

8

Applications Ajax-PHP synchrones

Pour commencer simplement, je vous propose une série d'ateliers qui vous permettra de créer progressivement une première application synchrone.

Pour illustrer son fonctionnement nous réaliserons une petite application qui simulera le fonctionnement d'une machine à sous en ligne. Côté client, l'application sera constituée d'une page HTML dans laquelle nous construirons progressivement un moteur Ajax dont l'objectif sera de simuler le fonctionnement d'une machine à sous. Pour cela l'utilisateur devra appuyer sur un bouton pour déclencher une requête depuis son navigateur et relancer ainsi la machine à sous. Côté serveur, un fichier PHP réceptionnera et traitera la requête client puis renverra le montant du gain en retour qui s'affichera ensuite dans la page Web.

Atelier 8-1 : requête synchrone sur un fichier texte sans feuille de styles

Composition du système

Nous allons commencer par mettre en place une structure minimaliste pour tester le fonctionnement d'une première requête synchrone sur un simple fichier texte (voir figure 8-1). Cette première structure est composée :

- d'une page HTML (`index.html`) dans laquelle nous allons intégrer un bouton de formulaire pour déclencher le jeu, une zone d'affichage du résultat et le JavaScript du moteur Ajax ;
- d'un simple fichier texte (`gainMax.txt`) dans lequel nous allons saisir la valeur du gain maximum, soit 100.

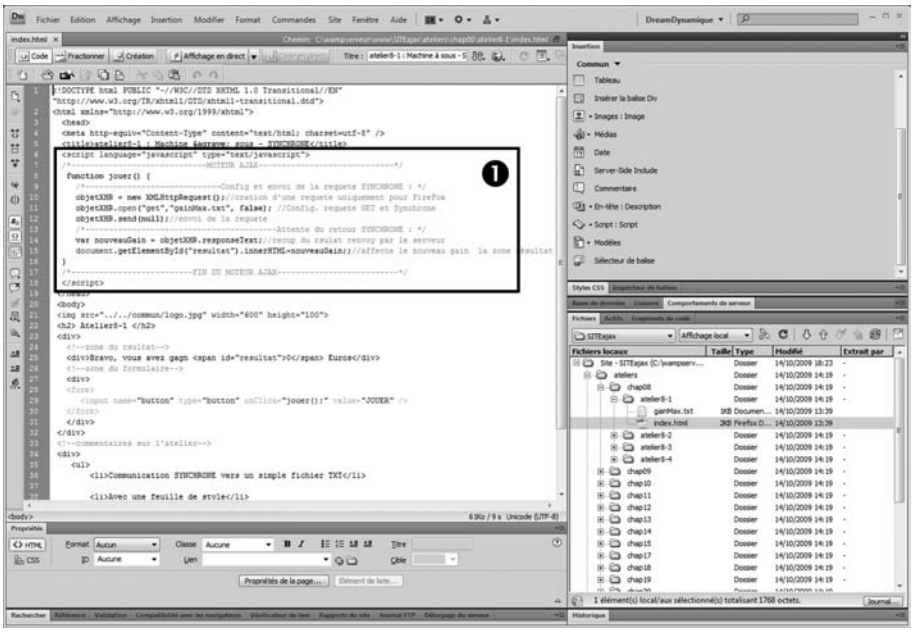


Figure 8-1

Saisie du moteur Ajax dans la balise <head> de la page

> **Prérequis concernant l'objet XMLHttpRequest (XHR)**
>
> Comme vous avez déjà découvert le fonctionnement de l'objet `XMLHttpRequest` dans le chapitre 4 de ce même ouvrage, nous nous appuierons sur ce prérequis dans la rédaction des ateliers de ce chapitre. Si toutefois certaines méthodes ou propriétés de cet objet vous semblent inconnues, n'hésitez pas à vous reporter au chapitre 4 pour revoir son concept.
>
> À noter aussi que nous aurons souvent l'occasion de faire référence à l'objet `XMLHttpRequest` dans cet ouvrage et pour alléger l'écriture, nous utiliserons fréquemment son appellation abrégée XHR.

Fonctionnement du système

Le bouton JOUER de la page permettra à l'utilisateur d'afficher la valeur maximum des gains. Ce bouton sera relié à un gestionnaire d'événements `onclick` qui appellera une fonction `jouer()` (voir code 8-1, à noter que pour cette première application le gestionnaire sera intégré dans la balise HTML `<input>` mais nous verrons par la suite qu'il est préférable de déclarer les gestionnaires d'événements directement dans le code JavaScript afin de bien séparer le programme de la structure de la page).

Cette fonction déclenchera le moteur Ajax (voir code 8-2) qui créera un objet XMLHttpRequest (par la suite, nous utiliserons son appellation abrégée XHR) puis le configurera (à l'aide de la méthode `open()` de l'objet : choix de la méthode `GET` et ciblage du fichier `gainMax.txt` en mode Synchrone) avant d'envoyer la requête (à l'aide de la méthode `send()` de l'objet).

L'information contenue dans le fichier texte `gainMax.txt` (soit la valeur 100) sera ensuite retournée au navigateur dans le corps de la réponse. Cette valeur sera enregistrée dans la variable `nouveauGain` par le biais de la propriété `responseText` de l'objet puis affectée à la zone de résultat à l'aide de la propriété `innerHTML` de l'élément résultat.

Conception du système

Téléchargement des codes sources des ateliers

Le moment est venu de passer à l'action. Les explications des différents ateliers vous permettront de créer vos différents scripts à partir de zéro. Cependant, vous avez la possibilité de télécharger tous les fichiers des exemples de cet ouvrage à votre disposition dans l'extension du livre sur le site *www.editions-eyrolles.com* (utilisez le nom de l'auteur comme mot clé pour accéder à l'extension de cet ouvrage).

Vous pourrez ainsi vous reporter à ces fichiers pour les comparer avec les vôtres en cas de problème, ou encore tester directement le fonctionnement de tous les ateliers directement sur ces ressources si vous ne désirez pas saisir vous-même les codes.

Ouvrez Dreamweaver (ou l'éditeur de votre choix) et sélectionnez le site Ajax que nous avons créé lors de l'installation de l'environnement de développement. Créez un nouveau fichier HTML et enregistrez-le tout de suite dans le répertoire `/ateliers/chap8/atelier8-1/` en le nommant `index.html`.

Organisation de vos ateliers

Nous vous suggérons de créer chaque atelier dans un répertoire différent portant le nom de l'atelier afin de bien isoler les fichiers utilisés dans le cadre de nos essais. Les deux fichiers (`index.html` et `gainMax.txt`) de ce premier atelier seront donc enregistrés dans un répertoire nommé « atelier8-1 ». Ainsi chaque atelier sera indépendant de l'autre, le seul élément qui ne sera pas dans le répertoire est l'image `logo.jpg` placée en haut de chaque page `index.html`, cette dernière étant commune à tous les ateliers, nous l'avons placé dans un répertoire nommé `/commun/`.

Dans ce premier exemple, nous avons choisi de ne pas lier la page HTML à une feuille de styles par souci de simplicité pour la mise en œuvre de votre première application. Néanmoins, nous allons structurer les différentes zones de la page HTML avec des balises `<div>` en prévision de la future feuille de styles que nous allons appliquer ensuite à cette page. Les deux éléments de la page (la zone `<span>` du résultat et le formulaire contenant le bouton JOUER) sont tous les deux insérés dans des balises `<div>` différentes et l'ensemble est regroupé dans une troisième balise `<div>` qui servira de conteneur pour la page (voir code 8-1).

Code 8-1 :

```
<div>
  <!--zone du résultat-->
  <div>
  Bravo, vous avez gagné <span id="resultat">0</span> euros
  </div>
  <!--zone du formulaire-->
  <div>
```

```
  <form>
   <input name="button" type="button" onclick="jouer();" value="JOUER" />
  </form>
  </div>
 </div>
```

Ressources sur les technologies associées

Nous avons regroupé dans la partie 4 de cet ouvrage plusieurs chapitres sur chacune des technologies utilisées dans les applications des ateliers. Nous vous invitons à vous y reporter pour obtenir des compléments d'informations si les explications qui accompagnent ces ateliers ne vous suffisent pas.

Placez-vous ensuite dans la balise `<head>` de la page et saisissez le code 8-2 ci-dessous.

Code 8-2 :

```
<script language="javascript" type="text/javascript">
/*----------------------------MOTEUR AJAX-----------------------*/
  function jouer() {
 /*-------------------Config et envoi de la requête SYNCHRONE : */
  //création d'une requête uniquement pour Firefox
  objetXHR = new XMLHttpRequest();
  //Config. requête GET et Synchrone
  objetXHR.open("get","gainMax.txt", false);
  //envoi de la requête
  objetXHR.send(null);
/*--------------------------------Attente du retour SYNCHRONE : */
  //récupération du résultat renvoyé par le serveur
  var nouveauGain = objetXHR.responseText;
  //Affecte le nouveau gain à la zone résultat
  document.getElementById("resultat").innerHTML=nouveauGain;
 }
/*-------------------------FIN DU MOTEUR AJAX------------------*/
 </script>
```

Enregistrez le fichier `index.html` après avoir terminé la saisie puis ouvrez un fichier texte (Depuis le menu de Dreamweaver : Fichier>Nouveau cliquez sur le bouton Autres à gauche de la fenêtre puis sélectionnez le type Texte dans la liste). Saisissez la valeur 100 dans le contenu du fichier et enregistrez-le sous le nom `gainMax.txt` (voir figure 8-2).

Pour ce premier script, il est intéressant d'expliquer d'une façon détaillée le code de cette page HTML afin de bien comprendre le mécanisme de cette application Ajax.

Dans la partie visible de page HTML (balise `<body>`, voir le code 8-1), la zone dans laquelle s'affichera le résultat est délimitée par une balise `<span>` à laquelle nous avons ajouté un identifiant `resultat` qui permettra ensuite d'en modifier le contenu à l'aide de la propriété `innerHTML` de l'élément ainsi constitué. Lors de son premier affichage, le contenu de cette balise est initialisé avec la valeur 0. Cette valeur sera ensuite remplacée par la valeur contenue dans le fichier texte (100).

```
<span id="resultat">0</span>
```

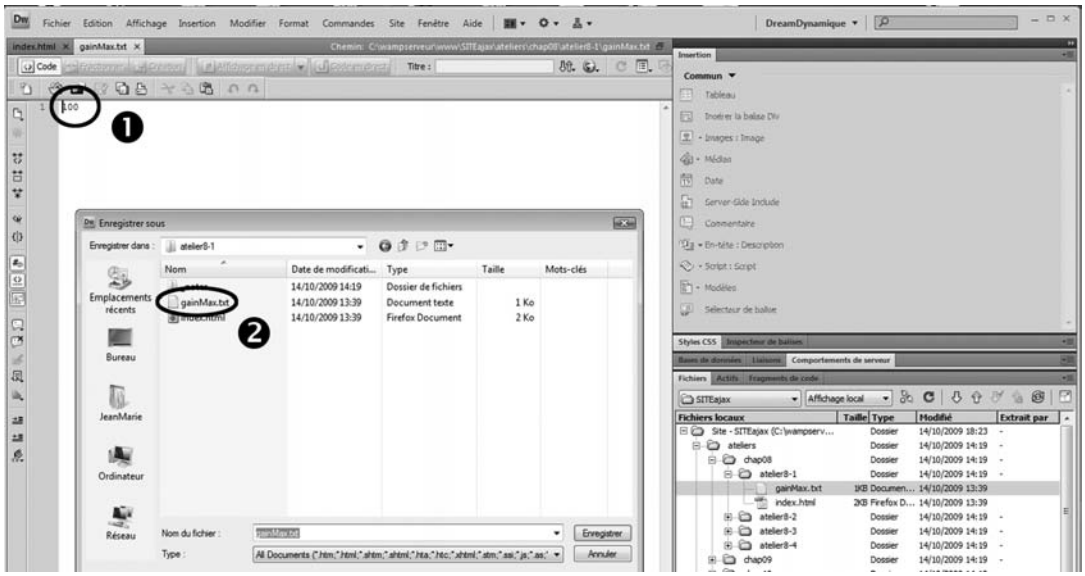


Figure 8-2

Création du fichier texte gainMax.txt

Un peu plus bas, un formulaire a été ajouté afi n d'insérer un bouton de commande JOUER dans la page HTML. L'élément `<input>` est lié à un gestionnaire d'événements qui p ermettra d'appeler la fonction contenant le moteur Ajax (`onclick="jouer()"`). L'utilisateur pourra ainsi afficher le contenu retourné par le fi chier texte par un simple clic sur ce bouton . À noter que la valeur du fichier te xte étant toujours la même, il est nécessaire d'appuyer sur le bouton d'actualisation du navigateur (ou d'utiliser le raccourci clavier F5) pour revenir à l'état initial de la page avant d'appuyer de nouveau sur le bouton JOUER.

```
<form>
 <input type="button" onclick="jouer();" value="JOUER" />
</form>
```

Passons maintenant à la fonction `jouer()` contenant le moteur Ajax. La première instruction de cette fonction permet d'instancier la classe XMLHttpRequest et de créer un objet nommé `objetXHR`.

```
objetXHR = new XMLHttpRequest();
```

Une fois l'objet créé, il f aut ensuite le confi gurer avec sa méthode `open()`. Trois paramètres seront nécessaires à sa configuration. Le premier permet d'indiquer que nous désirons utiliser la méthode `GET` pour émettre la requête HTTP. Le second précise le fichier ciblé par la requête, soit le fichier te xte `gainMax.txt` pour ce premier e xemple. Enfi n le troisième paramètre est initialisé avec la valeur `false` afin d'indiquer que la requête devra être en mode synchrone.

```
objetXHR.open("get","gainMax.txt", false);
```

Maintenant que l'objet est créé et configuré, il ne reste plus qu'à l'envoyer. Pour cela, nous utiliserons la méthode `send()` de l'objet. À noter que l'argument de cette méthode sera utilisé lorsque nous aurons une méthode `POST` avec des paramètres à communiquer au serveur. Comme ce n'est pas le cas de notre exemple, ce paramètre sera configuré avec la valeur `null`.

```
objetXHR.send(null);
```

La requête étant synchrone, la communication restera ouverte dans l'attente de la réponse comme dans le cas d'une requête HTTP traditionnelle. Nous pouvons donc placer les instructions de traitement de la réponse immédiatement après l'envoi de la requête. La première instruction de ce traitement permet d'affecter la réponse texte à une variable nommée `nouveauGain`. Nous utilisons pour cela la propriété `responseText` de l'objet XHR.

```
var nouveauGain = objetXHR.responseText;
```

Nous arrivons maintenant au terme de la fonction du moteurAjax. En effet, nous disposons de la valeur de la réponse côté client, il ne nous reste plus qu'à l'affecter à la zone résultat afin qu'elle remplace la valeur 0 configurée par défaut. Pour cela, nous utiliserons la méthode `getElementById()` qui permet de référencer l'élément de la balise `<span>` par son identifiant `resultat`. Puis nous exploiterons la propriété `innerHTML` qui permettra de remplacer le contenu de l'élément par la valeur 100 sauvegardée précédemment dans la variable `nouveauGain`.

```
document.getElementById("resultat").innerHTML=nouveauGain;
```

Test du système

Pour tester le système, vous devez commencer par ouvrir la page `index.html` dans le Web Local avec le navigateur Firefox. Pour cela, plusieurs solutions s'offrent à vous. La première est la plus rapide mais nécessite d'avoir configuré le serveur d'évaluation dans la définition initiale du site Ajax (revoir si besoin le chapitre 7). Assurez-vous avant tout que WampServer 2 est bien démarré (un icône en forme de demi-cercle doit apparaître dans la zone d'état en bas à droite de l'écran de votre ordinateur).

Ouvrez la page à tester (`index.html` dans notre cas) dans Dreamweaver puis cliquez sur l'icône Aperçu/Débogage (bouton ayant la forme d'une planète bleue) situé dans le menu de l'éditeur de fichier puis sélectionnez Firefox dans la liste des navigateurs (par la suite, nous vous conseillons d'utiliser le raccourci clavier F12). Le navigateur Firefox doit alors s'ouvrir et afficher la page concernée.

L'autre solution est plus longue mais pourra être utilisée dans tous les cas, même si vous n'utilisez pas Dreamweaver ou si le serveur d'évaluation n'a pas encore été configuré. Déroulez le menu du manager Wamp et sélectionnez l'option `localhost`. La page d'accueil du Web Local de Wamp doit alors s'ouvrir dans le navigateur Firefox (préalablement configuré comme le navigateur par défaut dans la procédure d'installation de WampServer 2). Dans la zone Vos projets cliquez sur le petit répertoire nommé `SITEajax`. La racine de notre site Ajax n'ayant pas de fichier d'index, la liste des fichiers et répertoires s'affiche dans la nouvelle page. Cliquez successivement sur les répertoires `atelier`, `chap8` puis `atelier8-1`. La page `index.html` doit alors s'ouvrir comme dans le cas de la première solution.

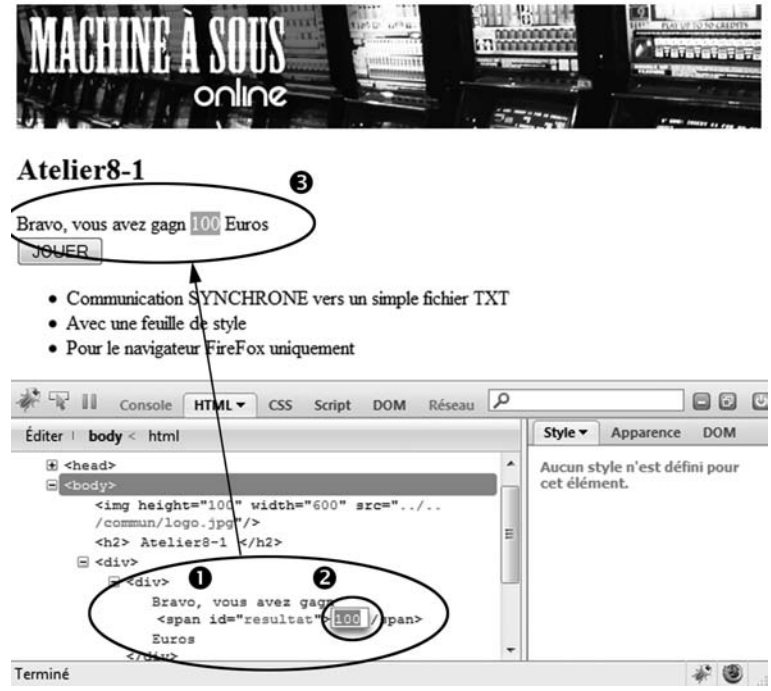
Fonctionne uniquement sur Firefox

L'exemple de cet atelier fonctionne uniquement sur le navigateur Firefox (ou les autres navigateurs compatibles avec l'objet `XMLHttpRequest`). Si toutefois vous désirez le faire fonctionner dès maintenant avec Internet Explorer (version supérieure à 5.0), il suffit de remplacer la syntaxe d'instanciation de l'objet XHR, soit actuellement `new XMLHttpRequest()`, par celle-ci : `new ActiveXObject("MSXML2.XMLHttp")`. Soyez rassuré, nous présenterons plus loin le script à utiliser pour que vos applications puissent fonctionner avec tous les types de navigateur.

Nous pouvons remarquer dans la page `index.html` qui est maintenant affichée dans le navigateur que la valeur du gain est égale à 0. Si vous cliquez sur le bouton JOUER, vous déclenchez alors la requête synchrone et la valeur du gain doit être immédiatement remplacée par celle du gain maximum stockée dans le fichier `gainMax.txt` (soit 100, voir repère 3 de la figure 8-3).

Figure 8-3

Affichage du contenu des éléments de la page HTML en mémoire à l'aide Firebug



L'intérêt de ces ateliers est surtout d'observer le fonctionnement du système afin de mieux comprendre les rouages du mécanisme d'une application Ajax et pour vous permettre par la suite de diagnostiquer un éventuel problème et dépanner une application plus complexe. Pour ces raisons nous allons activer l'extension Firebug à chaque test en cliquant sur l'icône placé en bas et à droite du navigateur.

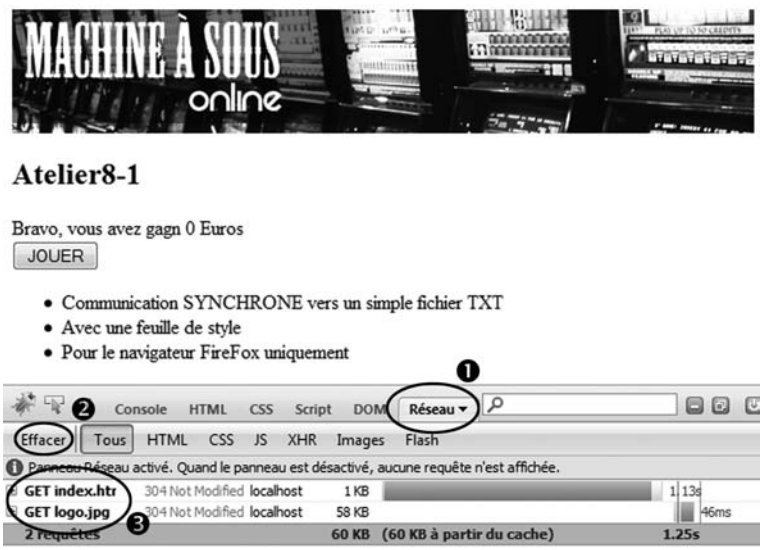
Comme nous l'avons déjà vu dans le chapitre consacré à Firefox et à ses extensions, Firebug permet d'effectuer de multiples opérations lors du test d'une applicationAjax. Par exemple, si

vous cliquez sur l'onglet HTML, vous pouvez afficher le contenu des éléments en mémoire en déroulant successivement les différents éléments de la page. Attention, il s'agit des contenus en mémoire (représentation du DOM sous forme de balises) et non d'un simple affichage du code initial de la page. Ainsi, dans notre cas, après l'envoi de la requête, la valeur dans la zone de résultat est égale à 100 (voir repères 1 et 2 de la figure 8-3) et non à 0 (valeur qui serait visible dans cette même fenêtre avant l'action sur le bouton JOUER ou dans le cas de l'affichage du code source traditionnel d'une page). De plus, si vous survolez avec votre souris l'un de ces éléments dans la fenêtre de Firebug, la zone correspondante dans l'écran du navigateur est alors mise en évidence.

Une autre fonctionnalité très intéressante de Firebug est de pouvoir observer les informations échangées entre le navigateur et le serveur. Vous pouvez ainsi faire apparaître tous les objets externes qui constituent votre page HTML (structure brute de la page HTML, images, feuille CSS, fichier JS externe…) lors de son appel initial et connaître le temps de chargement de chacun des objets individuellement. Mais cela est encore plus intéressant avec une application Ajax lors de l'envoi d'une requête car, si vous pouvez aussi connaître le temps de traitement de la requête, vous pouvez surtout afficher le contenu de la requête HTTP (et de tous ses entêtes) ainsi que le contenu de la réponse HTTP correspondante. Pour cela, il faut cliquer sur l'onglet Réseau de Firebug (voir repère 1 de la figure 8-4) puis sur le bouton Effacer (voir repère 2 de la figure 8-4) pour nettoyer les chargements précédents. Pour simuler le chargement initial de la page, nous allons cliquer sur le bouton de réactualisation de la page (ou plus simplement à l'aide du raccourci F5). Vous devriez voir apparaître les deux objets constituant la page Web de notre exemple avec leurs temps de chargement respectifs, soient la structure brute de la page HTML et le chargement de l'image placée en tête de notre page (voir repère 3 de la figure 8-4).

Figure 8-4

Affichage des temps de chargement des objets constituant une page Web à l'aide de Firebug



15

Plug-ins jQuery

Ce dernier chapitre de la troisième partie vient compléter le chapitre 14 sur jQuery en vous présentant quelques exemples de plug-ins jQuery qui trouveront certainement une place dans vos futures applications.

Les plug-ins jQuery sont des bibliothèques complémentaires à la bibliothèque de base que nous avons utilisée dans le chapitre précédent. D'installation très facile, ils vous permettront, en quelques lignes de code, de disposer d'applications avancées que vous pourrez personnaliser à votre guise.

Mise en œuvre d'un plug-in jQuery

Localisation des plug-ins jQuery

La sélection des 5 plug-ins jQuery présentés dans ce chapitre est évidemment très loin d'être exhaustive. Le but de ce chapitre est principalement de vous initier à la mise en œuvre de quelques plug-ins afin que vous puissiez ensuite appliquer la même procédure pour en installer d'autres selon les besoins de vos futures applications.

Vous trouverez une liste conséquente de plug-ins jQuery à l'adresse ci-dessous. Vous pourrez les classer selon leur famille (Ajax, Effets graphiques, Menus de navigation, Widget…), leur nom, leur date ou encore selon leur popularité.

http://jquery.com/plugins/

Comment installer un plug-in jQuery ?

L'installation d'un plug-in jQuery est très simple. Vous devez commencer par télécharger les ressources du plug-in jQuery sur votre ordinateur puis vous devez placer un ou plusieurs fichiers JavaScript et CSS dans un répertoire du site en cours de développement.

Figure 15-1

Annuaire des plug-ins jQuery

Il faut ensuite ajouter des balises `<script>` ou `<link>` pour référencer les ressources nécessaires dans la page sur laquelle vous désirez mettre en place l'application (voir exemple du code 15-1). Évidemment, le fichier de la bibliothèque de base `jquery.js` doit être préalablement référencé de la même manière dans cette même page.

Code 15-1 : exemple de balises de référence à des fichiers JavaScript et CSS d'un plug-in jQuery :

```
<link rel="stylesheet" href="ui.tabs.css" type="text/css" />
<script src="ui.tabs.js" type="text/javascript"></script>
```

Une fois les ressources disponibles dans la page, il ne vous reste plus qu'à créer un objet jQuery, en référençant l'élément qui accueille l'application à mettre en place, pour ensuite lui appliquer une des méthodes du plug-in jQuery (voir exemple du code 15-2).

Code 15-2 : exemple d'appel d'une méthode de plug-in jQuery :

```
$(document).ready(function() {
 $('#menuOnglet ul').tabs({ remote: true });
});
```

Dans l'exemple du code ci-dessus, le sélecteur utilisé (`#menuOnglet ul`) fait référence aux balises `<ul>` enfants de l'élément portant l'identifiant `menuOnglet`. L'application concernée (en l'occurrence ici, un menu à onglets) prend donc place sur ces balises ainsi ciblées (voir exemple du code 15-3).

Code 15-3 : exemple de balises qui supportent l'application du plug-in jQuery :

```
<div id="menuOnglet">
  <ul>
   <li><a href="infoCss.html"><span>CSS</span></a></li>
   <li><a href="infoXml.html"><span>XML</span></a></li>
  </ul>
</div>
```

Atelier 15-1 : plug-in UI Tabs : menu à onglets

Composition du système

Ce plug-in permet de mettre en œuvre un système de navigation dynamique par onglets. Dans notre exemple, le contenu de chaque onglet est stocké dans une page HTML spécifique mais il pourrait tout aussi bien être généré dynamiquement par un fichier PHP, si besoin.

Cette structure est composée :

- d'une page HTML (`index.html`) qui contient l'application ;
- du fichier de la bibliothèque de base jQuery (`jquery.js`) ;
- du fichier de la bibliothèque du plug-in UI Tabs (`ui.tabs.js`) ;
- du fichier de la feuille de styles du plug-in UI Tabs (`ui.tabs.css`) ;
- du fichier d'une feuille de styles complémentaire pour le navigateur IE (`ui.tabs-ie.css`) ;
- du fichier d'une feuille de style de personnalisation du contenu de chaque onglet qui est à définir selon la mise en forme désirée (`styleContenu.css`) ;
- d'un fichier élément graphique pour la construction des onglets du menu (`tab.png`) ;
- d'un ensemble de 4 pages contenant le contenu (fragments de codes HTML) de chaque onglet (`infoCss.html`, `infoXml.html`, `infoJavaScript.html`, `infoDom.html`) ;
- d'une animation indiquant que le traitement est en cours (`chargeur.gif`).

Fonctionnement du système

Le fonctionnement du système de navigation est très simple, dès que l'utilisateur clique sur un des onglets du menu, la page HTML correspondante est chargée et les fragments de codes HTML qu'elle contient s'affichent dans la zone située en dessous du menu.

Chaque chargement de la page est signalé par l'affichage d'une animation dans l'onglet. Dès que le chargement d'une page est terminé, la représentation des onglets est modifiée afin d'indiquer quel est l'onglet actuellement actif.

Conception du système

Ouvrez une nouvelle page HTML et sauvegardez-la sous le nom `index.html` dans un nouveau répertoire nommé `/chap15/atelier15-1/`.

Commencez par télécharger le fichier zip regroupant les différentes ressources sur le site de l'auteur (voir adresse ci-dessous).

http://stilbuero.de/jquery/tabs/

Copiez ensuite les différents fichiers nommés dans la composition du système indiquée précédemment dans ce nouveau dossier. À noter que le kit de l'auteur contient un fichier de la bibliothèque de base, vous pouvez donc utiliser ce fichier (dans ce cas renommez le `jquery.js`) ou récupérer celui que nous avons déjà utilisé dans les ateliers du chapitre 14.

Ouvrez ensuite la page `index.html` et ajoutez une balise de script faisant référence à la bibliothèque `jquery.js`.

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

Dans ce même fichier, ajoutez ensuite une autre balise de script afin de référencer le plug-in `ui.tabs.js`.

```
<script src="ui.tabs.js" type="text/javascript"></script>
```

Puis ajoutez trois liens `link` pour pouvoir disposer des feuilles de styles CSS du plug-in (`ui.tabs.css`) et son complément pour le navigateur Internet Explorer (`ui.tabs-ie.css`) ainsi que de la feuille de styles personnalisée (`styleContenu.css`) qu'il convient de créer spécialement pour mettre en forme le contenu de chaque onglet.

Ouvrez ensuite une balise `<script>` et insérez un gestionnaire `.ready()` afin de détecter le chargement complet des éléments du DOM.

Code 15-4 : gestionnaire de test du chargement du DOM :

```
<script type="text/javascript">
$(document).ready(function() {
 …
});
</script>
```

Ajoutez à l'intérieur une instruction qui permet de créer un objet jQuery sélectionnant les balises avec lesquelles l'application va être reliée. Dans notre cas, le sélecteur de l'objet référence la (ou les) balise(s) `<ul>` enfant(s) de l'élément dont l'identifiant est `memoOnglet`.

Code 15-5 : ajout de l'instanciation de l'objet jQuery :

```
<script type="text/javascript">
$(document).ready(function() {
 $('#menuOnglet ul');
});
</script>
```

Appliquez ensuite la méthode `tabs()` du plug-in jQuery à l'objet sélecteur précédemment configuré. Cette méthode possédant plusieurs options de configuration, nous allons utiliser dans notre exemple l'option `cache` qui permet de bloquer la mise en mémoire des données.

Code 15-6 : ajout de l'appel de la méthode `tabs()` :

```
<script type="text/javascript">
$(document).ready(function() {
 $('#menuOnglet ul').tabs({ cache: true });
});
</script>
```

Pour terminer la configuration de la page `index.html`, il faut maintenant créer les balises sur lesquelles le plug-in jQuery va s'appliquer. Pour cela, placez-vous dans la balise `body` de la page et ajoutez une balise `div`, son `id` étant configuré avec la valeur `menuOnglet`.

Code 15-7 : création de la zone d'affichage du menu :

```
<div id="menuOnglet">
 …
</div>;
```

Insérez ensuite à l'intérieur de cette balise une structure de listes non ordonnées (balise `<ul>`). Le menu comportera autant d'onglets qu'il y a de balises `<li>` dans cette structure. Dans notre exemple, nous allons ajouter 4 balises `<li>` afin de créer le même nombre d'onglets.

Code 15-8 : ajout des balises `<li>` :

```
<div id="menuOnglet">
 <ul>
   <li>…</li>
   <li>…</li>
   <li>…</li>
   <li>…</li>
 </ul>
</div>;
```

Ajoutez à l'intérieur de chaque balise `<li>` le nom qui apparaît sur l'onglet. Encadrez ensuite ce nom par un lien hypertexte qui pointe sur le fichier contenant les fragments HTML du contenu de l'onglet concerné.

Code 15-9 : intégration des informations des 4 onglets :

```
<div id="menuOnglet">
 <ul>
   <li><a href="infoCss.html" ><span>CSS</span></a></li>
   <li><a href="infoJavascript.html" ><span>JavaScript</span></a></li>
   <li><a href="infoDom.html" ><span>DOM</span></a></li>
   <li><a href="infoXml.html" ><span>XML</span></a></li>
 </ul>
</div>;
```

Les modifications du fichier `index.html` sont maintenant terminées, vous pouvez l'enregistrer et passer à la création de la feuille de styles dédiée à la mise en forme du contenu. Celle-ci

peut être personnalisée selon les contenus mis dans chaque onglet. Nous vous proposons ci-dessous quelques styles très simples pour créer ce fichier en guise d'exemple.

Code 15-10 : création de la feuille de styles `styleContenu.css` :

```
body {
 font-size: 16px;
 font-family: Verdana, Helvetica, Arial, sans-serif;
}
h1 {
 margin: 1em 0 1.5em;
 font-size: 18px;
}
p {
 margin: 0;
 font-size: 12px;
}
```

Enregistrez ce fichier sous le nom `styleContenu.css` avant de passer aux tests du système dans le navigateur.

Test du système

Ouvrez la page `index.html` dans le navigateur Firefox en appuyant sur la touche F12 dans Dreamweaver. Le système doit afficher le menu avec ses 4 onglets. Si la structure de la liste s'affiche sans mise en forme, vérifier la présence des différents fichiers JavaScript et CSS indiqués précédemment ainsi que la configuration des différents liens qui les référencent dans la page `index.html`.

Le premier onglet nommé CSS doit être ouvert par défaut. Cliquez ensuite sur l'onglet de votre choix. L'animation `chargeur.gif` doit apparaître dans l'onglet pendant le temps du chargement de la page de l'onglet sélectionné. Dès que le chargement est terminé, le contenu doit s'afficher dans la zone d'affichage et le menu doit changer d'apparence pour indiquer l'onglet actuellement sélectionné (voir figure 15-2).

Figure 15-2

Test du plug-in UI Tabs : menu à onglets dynamiques