9

Utilisation d'AJAX avec Spring

Depuis leurs débuts, les applications Web n'offrent à l'utilisateur final qu'une expérience relativement pauvre. Le cycle traditionnel requête/réponse sur lequel est fondé le Web ne permet pas la création d'interfaces client élaborées, telles que nous pouvons en observer dans les applications bureautiques traditionnelles.

Le problème vient essentiellement du fait que chaque action de l'utilisateur requiert un rechargement complet de la page HTML. Or c'est sur ce point qu'AJAX intervient en permettant le rechargement à chaud de portions d'une page HTML.

Le terme AJAX (Asynchronous JavaScript And XML) renvoie à un ensemble de technologies différentes, toutes existantes depuis longtemps mais qui, combinées intelligemment, permettent de changer radicalement notre mode de création des interfaces Web.

Le concept d'AJAX a été lancé début 2005 par Jesse James Garrett dans un article devenu célèbre, intitulé *AJAX: A New Approach to Web Applications,* disponible sur le site Web de sa société, Adaptive Path, à l'adresse *http://www.adaptivepath.com/publications/essays/archives/000385.php*.

Les technologies utilisées par AJAX, telles que JavaScript et XML, sont connues depuis longtemps, et, en réalité, plusieurs sociétés faisaient déjà de l'AJAX dès 2000. La révolution actuelle provient de l'arrivée à maturité de ces technologies désormais largement répandues et livrées en standard avec les navigateurs Internet récents. Pour preuve, des entreprises comme Google et Microsoft se mettent à utiliser AJAX de manière intensive. Des produits tels que GMail ou Google Maps sont des exemples de technologies AJAX mises à la disposition du grand public.

Ce chapitre se penche en détail sur AJAX et introduit plusieurs techniques permettant de créer une application en suivant ses principes. Nous aborderons ainsi DWR, l'un des frameworks AJAX les plus populaires et qui a la particularité de s'intégrer facilement à Spring. En particulier, ce framework nous permettra de publier des Beans Spring en JavaScript afin de manipuler directement des objets gérés par Spring depuis un navigateur Internet.

Nous soulignerons au passage les principaux écueils qui guettent le développeur AJAX et conclurons le chapitre par une présentation de script.aculo.us, une bibliothèque de scripts Open Source qui permet de réaliser aisément des effets spéciaux.

AJAX et le Web 2.0

Cette section entre dans le détail des technologies et concepts utilisés avec AJAX.

Nous commencerons par démystifier le terme marketing « Web 2.0 », très souvent associé à AJAX. Nous aborderons ensuite le fameux objet `XMLHttpRequest`, qui est au fondement d'AJAX et qui nous permettra de créer une première fonction AJAX simple.

Le Web 2.0

AJAX est souvent associé au Web 2.0. S'il s'agit là d'une campagne marketing très bien orchestrée, force est de reconnaître que derrière les mots se cache une façon radicalement nouvelle de construire des applications Web.

Le Web 2.0 est un concept plus global qu'AJAX, qui regroupe un ensemble d'aspects fonctionnels et métier. L'expression a été conçue par les sociétés O'Reilly et MediaLive International et a été définie par Tim O'Reilly dans un article disponible à l'adresse *http://www.oreillynet.com/lpt/a/6228*.

Nous pouvons retenir de cet article que les applications Web 2.0 s'appuient sur les sept principes suivants :

- Offre d'un service, et non plus d'une application packagée.
- Contrôle d'une banque de données complexe, difficile à créer et qui s'enrichit au fur et à mesure que des personnes l'utilisent.
- Implication des utilisateurs dans le développement de l'application.
- Utilisation de l'intelligence collective, les choix et les préférences des utilisateurs étant utilisés en temps réel pour améliorer la pertinence du site.
- Choix de laisser les internautes utiliser l'application comme un self-service, sans contact humain nécessaire.
- Fonctionnement de l'application sur un grand nombre de plates-formes, et plus seulement sur l'ordinateur de type PC (il y a aujourd'hui plus de terminaux mobiles que de PC ayant accès à Internet).

- Simplicité des interfaces graphiques, des processus de développement et du modèle commercial.

Nous comprendrons mieux cette philosophie en observant des sites emblématiques de ce Web 2.0, tels que les suivants :

- Wikipedia, une encyclopédie en ligne gérée par les utilisateurs eux-mêmes. Chacun est libre d'y ajouter ou de modifier du contenu librement, des groupes de bénévoles se chargeant de relire les articles afin d'éviter les abus.
- Les blogs, qui permettent à chacun de participer et de lier (mécanisme des *trackbacks*) des articles.
- del.icio.us et Flickr, des sites collaboratifs dans lesquels les utilisateurs gèrent leurs données en commun (des liens pour del.icio.us, des images pour Flickr). Ainsi, chacun participe à l'enrichissement d'une base de données géante.

Le Web 2.0 est donc davantage un modèle de fonctionnement collaboratif qu'une technologie particulière. Pour favoriser l'interactivité entre les utilisateurs, ces sites ont besoin d'une interface graphique riche et dynamique. C'est ce que leur propose AJAX, avec sa capacité à recharger à chaud des petits morceaux de pages Web. Ces mini-mises à jour permettent à l'utilisateur d'enrichir la base de données de l'application sans pour autant avoir à recharger une page HTML entière.

Tudu Lists, notre application étude de cas, est un modeste représentant du Web 2.0, puisqu'elle permet le partage de listes de tâches entre utilisateurs, à la fois en mode Web (avec AJAX) et sous forme de flux RSS.

Les technologies d'AJAX

AJAX met en œuvre un ensemble de technologies relativement anciennes, que vous avez probablement déjà rencontrées.

Un traitement AJAX se déroule de la manière suivante :

1. Dans une page Web, un événement JavaScript se produit : un utilisateur a cliqué sur un bouton (événement `onClick`), a modifié une liste déroulante (événement `onChange`), etc.
2. Le JavaScript qui s'exécute alors utilise un objet particulier, le `XMLHttpRequest` ou, si vous utilisez Microsoft Internet Explorer, le composant ActiveX `Microsoft.XMLHTTP`. Cet objet n'est pas encore standardisé, mais il est disponible sur l'ensemble des navigateurs Internet récents. Situé au cœur de la technique AJAX, il permet d'envoyer une requête au serveur en tâche de fond, sans avoir à recharger la page.
3. Le résultat de la requête peut ensuite être traité en JavaScript, de la même manière que lorsque nous modifions des morceaux de page Web en DHTML. Ce résultat est généralement du contenu renvoyé sous forme de XML ou de HTML, que nous pouvons ensuite afficher dans la page en cours.

AJAX est donc un mélange de JavaScript et de DHTML, des technologies utilisées depuis bien longtemps. L'astuce vient essentiellement de ce nouvel objet `XMLHttpRequest`, lui aussi présent depuis longtemps, mais jusqu'à présent ignoré du fait qu'il n'était pas disponible sur suffisamment de navigateurs Internet.

Voici un exemple complet de JavaScript utilisant cet objet `XMLHttpRequest` pour faire un appel de type AJAX à un serveur. C'est de cette manière que fonctionnaient les anciennes versions de Tudu Lists (avant l'arrivée de DWR, que nous détaillons plus loin dans ce chapitre). Cet exemple illustre le réaffichage à chaud d'une liste de tâches, ainsi que l'édition d'une tâche :

```
<script language="JavaScript">
var req;
var fragment;

/**
 * Fonction AJAX générique pour utiliser XMLHttpRequest.
 */
function retrieveURL(url) {
  if (window.XMLHttpRequest) {
    req = new XMLHttpRequest();
    req.onreadystatechange = miseAJourFragment;
    try {
      req.open("GET", url, true);
    } catch (e) {
      alert("<fmt:message key="todos.ajax.error"/>");
    }
    req.send(null);
  } // Utilisation d'ActiveX, pour Internet Explorer
  else if (window.ActiveXObject) {
    req = new ActiveXObject("Microsoft.XMLHTTP");
    if (req) {
      req.onreadystatechange = miseAJourFragment;
      req.open("GET", url, true);
      req.send();
    }
  }
}

/**
 * Met à jour un fragment de page HTML.
 */
function miseAJourFragment() {
  if (req.readyState == 4) { // requête complétée
    if (req.status == 200) { // réponse OK

      document.getElementById(fragment)
        .innerHTML = req.responseText;

    } else {
```

```
      alert("<fmt:message key="todos.ajax.error"/> " +
        req.status);
    }
  }
}

/**
 * Affiche la liste des tâches.
 */
function afficheLesTaches() {
  fragment='todosTable';
  retrieveURL('${ctx}/ajax/manageTodos.action?' +
    'listId=${todoList.listId}&method=render');
}

/**
 * Edite une tâche.
 */
function editeUneTache(id) {
  fragment='editFragment';
  retrieveURL('${ctx}/ajax/manageTodos.action?todoId=' +
    escape(id) + '&method=edit');
}

( ... )

</script>
```

Dans cet exemple, des fragments de page sont mis à jour à chaud, avec du contenu HTML renvoyé par le serveur.

Comme nous pouvons le deviner en regardant les URL envoyées au serveur, ces requêtes HTTP sont traitées par des actions Struts, le paramètre `method` étant utilisé par des actions Struts de type `DispatchAction` *(voir le chapitre 6 pour plus d'informations sur le traitement de ces requêtes).*

Le HTML renvoyé par les actions Struts est simplement affiché dans des éléments HTML. En l'occurrence, la méthode `innerHTML` utilisée plus haut permet de changer le HTML présent dans des éléments `<span>` :

```
<span id="editFragment"></span>

<span id="todosTable"></span>
```

Cette utilisation d'AJAX reste très simple mais réclame une relativement bonne connaissance de JavaScript. Suffisante pour un site Web simple, elle souffre cependant des quelques lacunes suivantes :

- Elle nécessite beaucoup de code JavaScript et devient difficile à utiliser dès lors que nous voulons faire plus que simplement afficher du HTML.
- Elle pose des problèmes de compatibilité d'un navigateur à un autre.

Afin de faciliter l'utilisation de ce type de technique, un certain nombre de frameworks ont vu le jour, tels Dojo, DWR, JSON-RPC, MochiKit, SAJAX, etc.

Nous présentons DWR, l'un des plus populaires, dans la suite de ce chapitre.

Le framework AJAX DWR (Direct Web Remoting)

Parmi la multitude de frameworks AJAX, nous avons choisi DWR pour sa popularité et son intégration à Spring. DWR permet très facilement de présenter en JavaScript des JavaBeans gérés par Spring. Il est ainsi possible de manipuler dans un navigateur Internet des objets s'exécutant côté serveur.

DWR est un framework Open Source développé par Joe Walker, qui permet d'utiliser aisément des objets Java (côté serveur) en JavaScript (côté client).

Fonctionnant comme une couche de transport, DWR permet à des objets Java d'être utilisés à distance selon un principe proche du RMI (Remote Method Invocation). Certains autres frameworks AJAX, comme Rico, proposent des bibliothèques beaucoup plus complètes, avec des effets spéciaux souvent très impressionnants. Pour ce type de résultat, DWR peut être utilisé conjointement avec des bibliothèques JavaScript non spécifiques à J2EE. Nous abordons plus loin dans ce chapitre l'utilisation conjointe de DWR et de script.aculo.us, une bibliothèque d'effets spéciaux très populaire parmi les utilisateurs de Ruby On Rails.

Signe de la reconnaissance de DWR par la communauté Java, des articles ont été publiés sur les sites développeur de Sun, IBM et BEA. Dans le foisonnement actuel de frameworks AJAX, nous pouvons raisonnablement avancer que DWR est l'un des plus prometteurs.

Publié sous licence Apache 2.0, ce framework peut être utilisé sans problème, quelle que soit l'application que vous développez.

Principes de fonctionnement

DWR est composé de deux parties : du JavaScript, qui s'exécute côté client dans le navigateur de l'utilisateur, et une servlet Java, laquelle est chargée de traiter les requêtes envoyées par le JavaScript.

DWR permet de présenter des objets Java en JavaScript et de générer dynamiquement le JavaScript nécessaire à cette fonctionnalité. Les développeurs JavaScript ont ainsi la possibilité d'utiliser de manière transparente des objets Java, qui tournent dans le serveur d'applications et qui donc ont la possibilité d'accéder à une base de données, par exemple. Cela augmente considérablement les possibilités offertes à l'interface graphique d'une application.

La figure 9.1 illustre l'appel d'une fonction Java depuis une fonction JavaScript contenue dans une page Web.

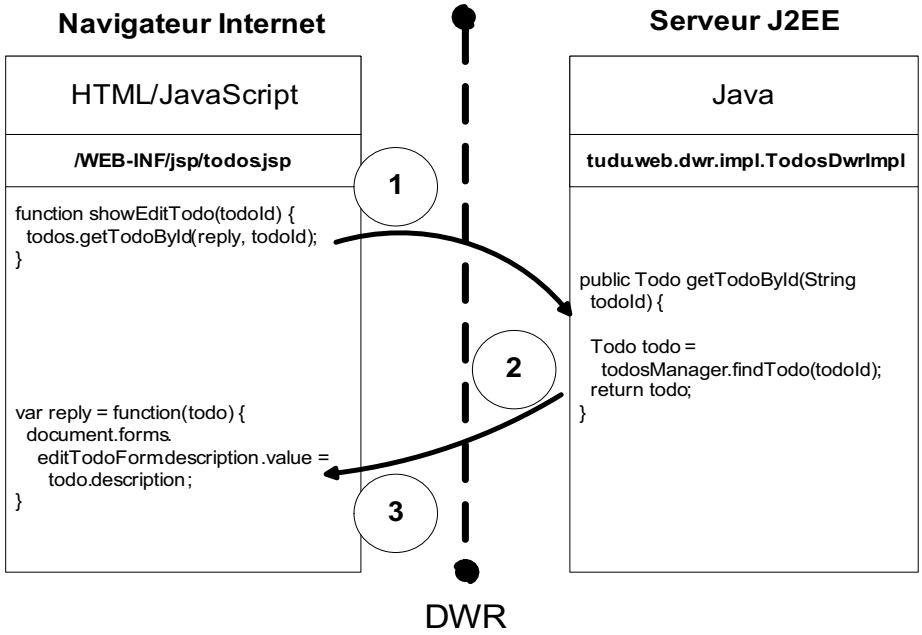


Figure 9.1
Fonctionnement de DWR

Afin de mieux illustrer ce propos, voici comment fonctionne une version simplifiée du code permettant d'éditer une tâche à chaud dans Tudu Lists.

Suite au clic de souris d'un utilisateur sur le texte « edit », à droite d'une tâche, le JavaScript suivant s'exécute :

```
function showEditTodo(todoId) {
 Effect.Appear('editTodoDiv');
 todos.getTodoById(reply, todoId);
 document.forms.editTodoForm.description.focus();
}
```

Ce code affiche le layer DHTML d'édition d'une page (il s'agit d'un effet spécial de script.aculo.us), appelle un objet Java distant et donne le focus à la description de la tâche éditée.

Le code d'appel à la fonction Java est composé d'un objet `todo`, généré automatiquement par DWR, qui prend les deux paramètres suivants en fonction :

- `reply`, la fonction callback JavaScript. Il s'agit donc non pas d'une variable passée en argument, mais d'une fonction JavaScript qui s'exécute après l'exécution de l'appel à l'objet Java.
- `todoId`, l'identifiant de la tâche à éditer, qui est passé en argument à la fonction Java distante.

Ce code appelle la fonction Java distante `getTodoById` de l'objet `todo` présenté par DWR :

```
package tudu.web.dwr.impl;

( ... )

/**
 * Implementation du service tudu.service.TodosManager.
 */
public class TodosDwrImpl implements TodosDwr {

  ( ... )
  public RemoteTodo getTodoById(String todoId) {
    Todo todo = todosManager.findTodo(todoId);
    RemoteTodo remoteTodo = new RemoteTodo();
    remoteTodo.setDescription(todo.getDescription());
    remoteTodo.setPriority(todo.getPriority());
    if (todo.getDueDate() != null) {
      SimpleDateFormat formatter =
        new SimpleDateFormat("MM/dd/yyyy");

      String formattedDate = formatter.format(todo.getDueDate());
      remoteTodo.setDueDate(formattedDate);
    } else {
      remoteTodo.setDueDate("");
    }
    return remoteTodo;
  }
}
```

Ce code Java recherche la tâche demandée dans la couche de service de l'application, laquelle fait à son tour un appel à la couche de persistance afin qu'Hibernate retrouve les informations en base de données. Pour des raisons de sécurité, que nous verrons plus tard dans ce chapitre, ce code crée un objet Java spécifique pour la couche DWR, et cet objet est retourné au client.

Au retour d'exécution du code Java, la fonction callback `reply`, que nous avons vue plus haut, s'exécute :

```
var reply = function(todo) {
 document.forms.editTodoForm.description.value = todo.description;
 document.forms.editTodoForm.priority.value = todo.priority;
 document.forms.editTodoForm.dueDate.value = todo.dueDate;
}
```

Étant une fonction callback, elle prend automatiquement un seul argument, l'objet retourné par la fonction JavaScript `getTodoById`. Cela revient à dire qu'elle utilise l'objet Java retourné par la méthode `getTodoById`, qui représente une tâche. Il est ensuite aisé de recopier les attributs de cet objet dans les champs HTML du layer présentant la tâche ainsi éditée.

Configuration

Vu de l'extérieur, DWR se compose d'une servlet et d'un fichier de configuration, nommé **dwr.xml.** Sa configuration n'est donc pas particulièrement complexe, d'autant que le framework fournit une excellente interface de test.

La servlet DWR

DWR est tout d'abord une servlet, qu'il faut configurer dans l'application Web en cours. Les différents JAR fournis dans la distribution de DWR doivent donc être copiés dans le répertoire **WEB-INF/lib,** et la servlet DWR doit être configurée dans le fichier **WEB-INF/web.xml,** comme toutes les servlets :

```
( ... )
<servlet>
<servlet-name>dwr-invoker</servlet-name>
<servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
<init-param>
  <param-name>debug</param-name>
  <param-value>false</param-value>
</init-param>
</servlet>

( ... )

<servlet-mapping>
  <servlet-name>dwr-invoker</servlet-name>
  <url-pattern>/secure/dwr/*</url-pattern>
</servlet-mapping>
( ... )
```

Dans cette configuration, le paramètre `debug` peut être mis à `true` afin d'avoir des informations plus précises sur le fonctionnement de DWR. Par ailleurs, l'URL d'accès à DWR est `/secure/dwr/*`. Cette URL est bien entendu librement configurable. Dans Tudu Lists, nous préférons la mettre derrière l'URL `/secure/*`, afin qu'elle soit protégée par Acegi Security.

Le fichier *dwr.xml*

DWR est configuré *via* un fichier, qui est par défaut **WEB-INF/dwr.xml.** Ce fichier est configurable grâce aux paramètres d'initialisation de la servlet DWR :

```
<servlet>
  <servlet-name>dwr-user-invoker</servlet-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
    <param-name>config-user</param-name>
    <param-value>WEB-INF/dwr-user.xml</param-value>
  </init-param>
</servlet>
```

Le nom du paramètre doit impérativement commencer par `config`. C'est pourquoi, dans l'exemple ci-dessus, nous l'avons appelé `config-user`. L'intérêt de cette manipulation est d'autoriser plusieurs fichiers de configuration, et ainsi plusieurs instances différentes de la servlet DWR : une instance pour les utilisateurs normaux (`config-user`) et une pour les administrateurs (`config-admin`). En leur donnant des URL différentes, nous pouvons utiliser la sécurité standard de J2EE pour ne permettre qu'aux utilisateurs ayant un rôle donné d'utiliser une instance de DWR.

Le fichier **dwr.xml** est de la forme suivante :

```
<dwr>
  <init>
    <creator id="..." class="..."/>
    <converter id="..." class="..."/>
  </init>

  <allow>
    <create creator="..." javascript="..." scope="...">
      <param name="..." value="..."/>
    </create>

    <convert convertor="..." match="..."/>
  </allow>

  <signatures>
    ( ... )
  </signatures>
</dwr>
```

La partie `<init></init>` n'est généralement pas utilisée. Elle sert à instancier les classes utilisées plus bas dans le fichier de configuration, dans le cas où le développeur souhaiterait utiliser ses propres classes au lieu de celles fournies en standard par DWR. Nous allons voir que les classes livrées avec DWR sont cependant largement suffisantes pour une utilisation de l'outil, fût-elle avancée.

La section `<allow></allow>` indique à DWR quelles classes il a le droit d'instancier et de convertir. Cette section est la plus importante du fichier. Nous allons détailler les éléments Creator et Convertor qui la composent.

Pour instancier une classe, DWR utilise un Creator, défini dans le fichier de configuration par la balise `<create>`. Ce Creator peut instancier une classe de plusieurs manières et les associer à un JavaScript généré à la volée :

```
<allow>
  <create creator="new" javascript="Example">
    <param name="class" value="tudu.web.dwr.Example"/>
  </create>
</allow>
```

Dans cet exemple, le Creator `new` permet d'instancier une classe et de l'associer à un JavaScript nommé `Example`. Il s'agit du Creator le plus simple, mais il en existe d'autres,

en particulier pour accéder à des Beans Spring, ce que nous verrons dans l'étude de cas. L'objet `Example` une fois présenté ainsi en JavaScript, ses méthodes peuvent être appelées depuis le navigateur client. Cependant, il est tout à fait possible que lesdites méthodes utilisent comme arguments, ou comme objets de retour, des classes complexes, telles que des collections ou des JavaBeans. Ces classes vont donc devoir être converties d'un langage à l'autre.

Pour convertir une classe Java en JavaScript et *vice versa,* DWR utilise des Converters. En standard, DWR est fourni avec des Converters permettant la conversion des types Java primaires, des dates, des collections et des tableaux, des JavaBeans, mais aussi d'objets plus complexes, comme des objets DOM ou Hibernate.

Pour des raisons de sécurité, certains Converters complexes, comme celui gérant les JavaBeans, ne sont pas activés par défaut. Pour convertir un JavaBean en JavaScript, il faut donc autoriser sa conversion :

```
<allow>
  <convert converter="bean"
           match="tudu.web.dwr.bean.RemoteTodoList"/>

  <convert converter="bean"
           match="tudu.web.dwr.bean.RemoteTodo"/>
</allow>
```

Utilisation du JavaScript dans les JSP

Une fois la servlet DWR correctement configurée, il est nécessaire d'accéder au code JavaScript généré dynamiquement par le framework depuis les pages JSP. Pour cela, il faut importer dans les JSP les deux bibliothèques propres à DWR :

```
<script type="text/javascript"
        src="${ctx}/secure/dwr/engine.js"></script>

<script type="text/javascript"
        src="${ctx}/secure/dwr/util.js"></script>
```

Dans Tudu Lists, cet import est réalisé dans le fichier **WEB-INF/jspf/header.jsp.**

Notons que ces deux fichiers JavaScript sont fournis par la servlet DWR et qu'ils n'ont pas à être ajoutés manuellement dans l'application Web.

Il faut ensuite importer les fichiers JavaScript générés dynamiquement par DWR et qui représentent les classes Java décrites dans **dwr.xml.** Voici l'exemple de la classe `tudu.web.dwr.impl.TodosDwrImpl`, qui est configurée dans **dwr.xml** pour être présentée en tant que todos :

```
<script type='text/javascript'
        src='${ctx}/secure/dwr/interface/todos.js'></script>
```

Ce fichier doit donc être importé depuis la servlet DWR mappée dans Tudu Lists sur l'URL `/secure/dwr/*`, à laquelle nous ajoutons le suffixe `interface`.

Test de la configuration

La configuration étudiée ci-dessus est relativement complexe, l'import et l'utilisation du JavaScript généré posant généralement problème aux nouveaux utilisateurs. C'est pourquoi DWR est fourni avec une interface de tests très bien conçue. Pour la mettre en place, il faut modifier la configuration de DWR dans le fichier **web.xml,** afin de passer en mode debug :

```
<servlet>
  <servlet-name>dwr-invoker</servlet-name>
  <servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>true</param-value>
  </init-param>
</servlet>
```

Notons bien que cette configuration ne doit pas être utilisée en production, car elle faciliterait la tâche d'une personne malveillante.

Une fois ce paramétrage effectué, nous pouvons accéder à des pages de test *via* l'URL de la servlet DWR, qui, dans notre étude de cas, est *http://127.0.0.1:8080/Tudu/secure/dwr/* (si vous avez changé la racine de contexte du projet, remplacez *Tudu* dans l'URL par la nouvelle racine, en respectant les majuscules). Ces pages listent les objets présentés par DWR, ainsi que leurs méthodes. Ces méthodes peuvent être appelées directement. Le code source de ces pages fournit une aide précieuse pour l'utilisation de DWR.

La figure 9.2 illustre la page affichant la classe `TodosDWRImpl`, l'une des deux classes configurées avec DWR dans Tudu Lists.

Figure 9.2
Interface de test de DWR

Utilisation de l'API servlet

DWR permet de s'abstraire de l'API servlet, ce qui simplifie généralement grandement le travail à effectuer. En comparaison de Struts, nous utilisons directement des arguments passés en paramètres au lieu de les extraire depuis un objet de formulaire.

L'API servlet reste toutefois incontournable pour peu que nous voulions stocker des attributs dans la session HTTP, utiliser JAAS pour la sécurité ou accéder au contexte de l'application Web. Pour toutes ces utilisations, DWR stocke dans une variable `ThreadLocal` les objets `HttpServletRequest`, `HttpServletResponse`, `HttpSession`, `ServletContext` et `ServletConfig`.

Pour accéder à la requête HTTP en cours, il suffit d'écrire :

```
HttpServletRequest request
  = uk.ltd.getahead.dwr.WebContextFactory.get()
                .getHttpServletRequest();
```

L'accès à cette variable en `ThreadLocal` n'est bien entendu possible que pour les fils d'exécution correspondant à des requêtes traitées avec DWR.

Gestion des performances

À première vue, l'utilisation d'AJAX améliore les performances. Au lieu de demander au serveur de recharger des pages complètes, nous nous contentons de lui demander le strict minium.

Cependant, dès que nous maîtrisons cette technique, nous sommes rapidement poussés à multiplier les requêtes côté serveur. Le site devient alors plus riche, plus « Web 2.0 », mais, en contrepartie, les requêtes HTTP se trouvent multipliées.

Coût de DWR en matière de performances

En lui-même, DWR n'ajoute qu'un impact négligeable en matière de performances côté serveur. Le serveur d'applications, le temps réseau ou les traitements métier sont normalement bien plus importants.

Joe Walker, l'auteur de DWR, a effectué plusieurs tests qui lui permettent de proposer les optimisations suivantes :

- Les appels de moins de 1 500 octets peuvent tenir dans un seul paquet TCP, ce qui améliore sensiblement l'utilisation du réseau. Il est donc important de n'échanger que le strict minimum de données.
- L'utilisation de paramètres de la JVM privilégiant les objets à faible durée de vie apporte un gain de performances, tout du moins avec Tomcat. Avec une JVM Sun, il s'agit de l'option -XX:NewSize.

Utilisation de batchs pour les requêtes

Il est possible de grouper des requêtes DWR afin de diminuer le nombre d'allers-retours entre le client et le serveur. Nous retrouvons cette optimisation à plusieurs endroits dans Tudu Lists, en particulier lors de l'ajout d'un todo :

```
DWREngine.beginBatch();
todos.addTodo(replyRenderTable,
  listId, description, priority, dueDate);

todos.getCurrentTodoLists(replyCurrentTodoLists);
DWREngine.endBatch();
```

La fonction `beginBatch()` permet de placer les requêtes suivantes en queue et de ne les envoyer qu'une fois la méthode `endBatch()` exécutée.

Étude des performances avec JAMon

Afin de mieux étudier les performances des appels à DWR, nous avons utilisé JAMon, un outil Open Source de monitoring des performances disponible à l'adresse *http://www.jamonapi.com/.*

Pour cela, nous avons créé un filtre de servlet, disponible dans Tudu Lists mais que vous pouvez réutiliser sans modification dans votre propre application. Cette classe, `tudu.web.filter.JAMonFilter`, permet d'accumuler des statistiques sur l'ensemble des requêtes HTTP envoyées au serveur. Ces statistiques sont ensuite visualisables *via* **JAMonAdmin.jsp,** une JSP fournie sur le site de JAMon et que nous avons intégrée dans Tudu Lists. Il s'agit de l'onglet Monitoring, qui est uniquement accessible aux administrateurs de l'application.

Grâce à ce filtre et à la JSP permettant d'étudier les statistiques, il est possible de trouver les requêtes AJAX qui prennent le plus de temps ou celles qui sont appelées le plus souvent.

Dans le cas particulier de Tudu Lists, la requête la plus utilisée en production est l'appel à la méthode `renderTodos` du Bean Spring `todosDwr`. Elle est appelée plusieurs dizaines de milliers de fois par jour et a un temps de réponse inférieur à 10 millisecondes.

Intégration de Spring et de DWR

DWR propose une ingénieuse intégration à Spring, qui permet d'utiliser directement des Beans Spring en JavaScript.

Il faut pour cela utiliser un Creator particulier, nommé `spring`, qui indique à DWR que le Bean en question est géré par Spring. Ce créateur prend en paramètre l'ID du Bean Spring. C'est cette configuration que nous retrouvons dans Tudu Lists :

```
<dwr>
  <allow>
    <create creator="spring"
```

```
            javascript="todo_lists" scope="application">

      <param name="beanName" value="todoListsDwr" />
    </create>
    <create creator="spring"
            javascript="todos" scope="application">

      <param name="beanName" value="todosDwr" />
    </create>

( ... )

  </allow>
</dwr>
```

Dans cet exemple, nous publions en JavaScript les Beans Spring nommés `todoListsDwr` et `todosDwr`, par l'intermédiaire de DWR. Ces Beans ont bien entendu été configurés au préalable dans un contexte Spring, nommé **WEB-INF/applicationContext-dwr.xml.**

Voici la configuration du premier Bean, `todoListsDwr` :

```
<bean id="todoListsDwr"
      class="tudu.web.dwr.impl.TodoListsDwrImpl">

  <property name="todoListsManager">
    <ref bean="todoListsManager" />
  </property>
  <property name="userManager">
    <ref bean="userManager" />
  </property>
</bean>
```

Les Beans Spring ainsi configurés sont accessibles en JavaScript. En voici un exemple d'utilisation, tiré de la page JSP **WEB-INF/jsp/todo_lists.jsp,** qui sert à gérer les listes de tâches :

```
<script type='text/javascript'
        src='${ctx}/secure/dwr/interface/todo_lists.js'>
</script>

<script type="text/javascript">
( ... )

// Ajout d'un utilisateur à la liste sélectionnée.
function addTodoListUser() {
  var listId = document.forms.editListForm.listId.value;
  var login = document.forms.editListForm.login.value;
  todo_lists.addTodoListUser(replyAddTodoListUser, listId, login);
}
```

Tudu Lists : utilisation d'AJAX

Nous utilisons la technologie AJAX dans les deux pages principales de l'étude de cas Tudu Lists, la page de gestion des listes de todos et la page de gestion des todos.

Tudu Lists utilise DWR, intégré à Spring de la manière étudiée précédemment, afin d'éditer, ajouter, supprimer ou afficher des entités gérées dans la couche de service de l'application et persistées avec Hibernate.

Fichiers de configuration

Les fichiers de configuration sont ceux étudiés précédemment :

- **WEB-INF/web.xml,** qui sert à configurer la servlet DWR.xml.
- **WEB-INF/applicationContext-dwr.xml,** qui gère les Beans Spring présentés avec DWR.
- **WEB-INF/dwr.xml,** qui est le fichier de configuration de DWR.

Chargement à chaud d'un fragment de JSP

Nous allons commencer par l'exemple le plus simple, le chargement à chaud d'un fragment HTML généré côté serveur par une JSP. L'utilisation d'AJAX ne nécessite pas obligatoirement l'utilisation et la transformation de données en XML. Il est possible de générer une chaîne de caractères côté serveur et de l'afficher directement dans la page HTML en cours. Dans le monde J2EE, le moyen naturel pour générer du HTML étant un fichier JSP, voici de quelle manière transférer une JSP en AJAX.

Cette technique s'applique aux deux fichiers **WEB-INF/jspf/todo_lists_table.jsp** et **WEB-INF/jspf/todos_table.jsp.** Nous allons étudier plus précisément ce deuxième fichier, qui est essentiel à la génération de la principale page de l'application. Voyons pour cela le JavaScript contenu dans la page **WEB-INF/jsp/todos.jsp** qui va utiliser ce fragment de JSP :

```
function renderTable() {
  var listId = document.forms.todoForm.listId.value;
  todos.renderTodos(replyRenderTable, listId);
}

var replyRenderTable = function(data) {
  DWRUtil.setValue('todosTable',
    DWRUtil.toDescriptiveString(data, 1));
}
```

La fonction `renderTable()` utilise l'objet `todos`, qui est un Bean Spring présenté par DWR, pour générer du HTML. La variable `replyRenderTable` est une fonction callback prenant automatiquement en paramètre l'argument de retour de la fonction `renderTodos`.

Elle utilise des méthodes utilitaires fournies par DWR pour afficher cet argument de retour dans l'entité HTML possédant l'identifiant `todosTable`.

Ces fonctions utilitaires, fournies par le fichier **util.js,** ont été importées *via* le header de la JSP **WEB-INF/jspf/header.jsp.** Ce ne sont pas des fonctions AJAX, et elles ne servent qu'à faciliter l'affichage sans être pour autant obligatoires. Les deux fonctions utilisées ici sont les suivantes :

- `DWRUtil.setValue(id, value)`, qui recherche un élément HTML possédant l'identifiant donné en premier argument et lui donne la valeur du second argument.
- `DWRUtil.toDescriptiveString(objet, debug)`, une amélioration de la fonction `toString` qui prend en paramètre un niveau de debug pour plus de précision.

La partie la plus importante du code précédent concerne la fonction `todos.renderTodos()`, qui prend en paramètre la fonction callback que nous venons d'étudier ainsi qu'un identifiant de liste de todos.

Comme nous pouvons le voir dans le fichier **WEB-INF/dwr.xml,** cette fonction est en fait le Bean Spring `todosDwr` :

```
<create creator="spring" javascript="todos" scope="application">
  <param name="beanName" value="todosDwr" />
</create>
```

Ce JavaBean est lui-même configuré dans **WEB-INF/applicationContext-dwr.xml :**

```
<bean id="todosDwr" class="tudu.web.dwr.impl.TodosDwrImpl">
```

La fonction appelée est au final la fonction `renderTodos(String listId)` de la classe `TodosDwrImpl`.

Cette méthode utilise la méthode suivante pour retourner le contenu d'une JSP :

```
return WebContextFactory.get().forwardToString(
                    "/WEB-INF/jspf/todos_table.jsp");
```

Cette méthode permet donc de recevoir le résultat de l'exécution d'une JSP sous forme de chaîne de caractères, que nous pouvons ensuite insérer dans un élément HTML de la page grâce à la fonction `DWRUtil.setValue()`, que nous avons vue précédemment.

Cette technique évite d'utiliser du XML, qu'il faudrait parser et transformer côté client. Elle permet d'utiliser un résultat qui a en fait été obtenu côté serveur. En ce sens, ce n'est donc pas une technique AJAX « pure ». Elle a cependant l'avantage d'être simple et pratique.

Modification d'un tableau HTML avec DWR

Dans l'étape suivante de notre étude de cas, nous utilisons plus complètement l'API de DWR en modifiant les lignes d'un tableau HTML. Cette technique permet de créer des tableaux éditables à chaud par l'utilisateur. Ce dernier peut en changer les lignes et les cellules sans pour autant avoir à subir le rechargement de la page Web en cours.

La page utilisée pour gérer les todos, **WEB-INF/jsp/todos.jsp,** possède un menu sur la gauche, qui contient un tableau contenant les listes de todos possédées par l'utilisateur. Ce tableau est géré en AJAX grâce au JavaScript suivant :

```
function renderMenu() {
  todos.getCurrentTodoLists(replyCurrentTodoLists);
}

var replyCurrentTodoLists = function(data) {
  DWRUtil.removeAllRows("todoListsMenuBody");
  DWRUtil.addRows("todoListsMenuBody", data,
    [ selectTodoListLink ]);
}

function selectTodoListLink(data) {
  return "<a href=\"javascript:renderTableListId('"
    + data.listId + "')\">" + data.description + "</a>";
}
```

Nous utilisons les deux éléments importants suivants :

- `renderMenu()`, qui est la fonction appelée à d'autres endroits de l'application pour générer le tableau. En l'occurrence, cette fonction est appelée au chargement de la page, lors de la mise à jour des todos, ainsi que par une fonction lui demandant un rafraîchissement toutes les deux minutes. En effet, les listes de todos pouvant être partagées avec d'autres utilisateurs, les données doivent être régulièrement rafraîchies. Cette fonction appelle un Bean Spring présenté avec DWR, dont nous connaissons maintenant bien le fonctionnement, et utilise une variable callback.
- `replyCurrentTodoLists`, qui est une fonction callback prenant comme paramètre le résultat de la méthode `todos.getCurrentTodoLists()`. Cette méthode, qui appartient à la classe `tudu.web.dwr.impl.TodosDwrImpl`, retourne un tableau de JavaBeans, de type `tudu.web.dwr.bean.RemoteTodoList`. Afin de retourner un tableau de listes de todos, nous n'utilisons pas directement le JavaBean `tudu.domain.model.TodoList`. Présenter en JavaScript un objet de la couche de domaine pourrait en effet présenter un risque en matière de sécurité. Pour cette raison, seuls des Beans spécifiques à la présentation sont autorisés dans DWR, *via* la section `<allow></allow>` du fichier **WEB-INF/dwr.xml.**

Cette dernière fonction fait appel aux classes utilitaires de DWR suivantes, qui permettent de gérer les éléments de tableau :

- `DWRUtil.removeAllRows(id)`, qui enlève toutes les lignes du tableau HTML possédant l'identifiant passé en argument.
- `DWRUtil.addRows(id, data, cellFuncs)`, qui ajoute des lignes au tableau HTML possédant l'identifiant passé en premier argument. Les données utilisées pour construire ce tableau sont passées dans le deuxième paramètre de la fonction et sont un tableau d'objets. Le troisième paramètre est un tableau de fonctions. Chacune de ces fonctions prend en paramètre un élément du tableau, ce qui permet de créer les cellules.

Dans notre exemple, le tableau n'ayant qu'une colonne, une seule fonction, `selectTodoListLink(data)`, prend donc en paramètre un JavaBean, `tudu.web.dwr.bean.RemoteTodoList`, converti au préalable en JavaScript.

Nous pouvons ainsi utiliser le JavaScript pour obtenir l'identifiant et la description de la liste à afficher dans la cellule du tableau.

Utilisation du pattern session-in-view avec Hibernate

Le pattern session-in-view, présenté au chapitre 11, permet d'utiliser l'initialisation tardive, ou lazy-loading, pour de meilleures performances. Cette méthode, utile dans le cadre des JSP, reste entièrement valable pour des composants AJAX.

C'est de cette manière que le Bean Spring `tudu.web.dwr.impl.TodoListsDwrImpl` peut rechercher la liste des utilisateurs dans sa méthode `getTodoListsUsers()`. En effet, la liste des utilisateurs est une collection en initialisation tardive, c'est-à-dire qu'elle n'est réellement recherchée en base de données qu'au moment où elle appelée.

Pour ce faire, il nous faut configurer le filtre de servlets d'Hibernate afin qu'il traite les requêtes envoyées à DWR de la même manière qu'il traite les requêtes envoyées à Struts ou à Spring MVC. Cela se traduit par un double mapping dans le fichier **WEB-INF/web.xml :**

```
<filter>
  <filter-name>Hibernate Session In View Filter</filter-name>
  <filter-class>
org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
  </filter-class>
</filter>

<!-- configuration pour Struts -->
<filter-mapping>
  <filter-name>Hibernate Session In View Filter</filter-name>
  <url-pattern>*.action</url-pattern>
</filter-mapping>

<!-- configuration pour DWR -->
<filter-mapping>
  <filter-name>Hibernate Session In View Filter</filter-name>
  <url-pattern>/secure/dwr/*</url-pattern>
</filter-mapping>
```

Améliorations apportées par script.aculo.us

script.aculo.us est une bibliothèque d'effets spéciaux qui n'est pas spécifique à DWR ou même à J2EE, mais qui s'intègre bien à ces technologies.

Le site Web officiel de script.aculo.us *(http://script.aculo.us)* propose de nombreux exemples et démonstrations, dont beaucoup sont réalisées avec Ruby On Rails, un framework aujourd'hui très populaire, qui intègre script.aculo.us en standard.

Nous allons voir comment intégrer script.aculo.us à DWR dans le cadre des Tudu Lists afin de réaliser un site Web esthétiquement plus agréable.

Installation

Pour installer script.aculo.us, il faut en télécharger la dernière version et l'installer dans l'application Web comme n'importe quel fichier JavaScript. Pour Tudu Lists, nous avons choisi de l'installer dans le répertoire **WebContent/scripts/scriptaculous.** Afin de suivre les évolutions de cette bibliothèque, nous avons ajouté un fichier **version.txt**, qui stocke la version de la distribution utilisée.

Nous avons ensuite importé les deux fichiers principaux de script.aculo.us dans le header de Tudu Lists, le fichier **WEB-INF/jspf/header.jsp :**

```
<script type="text/javascript"
        src="${ctx}/scripts/scriptaculous/prototype.js">
</script>
<script type="text/javascript"
        src="${ctx}/scripts/scriptaculous/scriptaculous.js">
</script>
```

Le premier fichier, **prototype.js,** correspond à Prototype, un framework simplifiant le développement JavaScript sur lequel script.aculo.us est fondé.

Le deuxième fichier, **scriptaculous.js,** importe l'ensemble des fichiers nécessaires à l'utilisation de script.aculo.us.

Nous avons choisi de placer ces deux fichiers dans le header de Tudu Lists afin de simplifier le développement, sachant qu'une fois téléchargés ils seront stockés dans le cache du navigateur client. Les importer dans chaque page JSP n'est donc pas un handicapant en terme de charge serveur.

Utilisation des effets spéciaux

L'utilisation des effets fournis par script.aculo.us est très simple. En voici un exemple, provenant de la page de gestion des todos, **WEB-INF/jsp/todos.jsp :**

```
function renderMenu() {
  todos.getCurrentTodoLists(replyCurrentTodoLists);
  Effect.Appear("todoListsMenuBody");
}
```

La bibliothèque d'effets spéciaux s'utilise avec l'objet `Effect`, lequel possède un grand nombre d'effets, qui prennent en paramètre l'identifiant du composant HTML à impacter. Dans l'exemple ci-dessus, nous demandons un effet d'apparition sur le composant HTML `todoListsMenuBody`.

La liste complète de ces effets est disponible sur le site de script.aculo.us. Il existe notamment des effets de fondu (Effect.Fade) et de disparition (Effect.Puff), qui ressemblent à ce que nous trouvons dans Microsoft PowerPoint pour afficher ou faire disparaître des éléments de présentation.

Ces effets peuvent aussi être utilisés dans les événements JavaScript, comme dans l'exemple suivant :

```
<div onclick="new Effect.Fade(this)">
    Cliquez ici pour que cet élément disparaisse.
</div>
```

Ces effets spéciaux présentent les deux utilités majeures suivantes dans la construction d'une page AJAX :

- Ils permettent à l'utilisateur d'avoir une notification visuelle de ses actions. Les personnes habituées aux sites Web classiques peuvent être désorientées devant un site Web AJAX. Grâce à des effets tels que `Effect.Shake`, `Effect.Pulsate` ou `Effect.Highlight`, elles ont mieux conscience de ce qui se passe dans l'application.
- Ils permettent à l'utilisateur d'attendre que les fonctions AJAX s'exécutent. Lorsque nous éditons un todo, il est plus agréable d'avoir une boîte de dialogue qui s'affiche en douceur, avec les éléments déjà prérenseignés. Sans effet spécial, nous n'aurions le choix qu'entre un site semblant peu réactif, si nous recherchons les informations avant d'afficher la boîte de dialogue, ou un site avec des boîtes de dialogues vides se remplissant avec un temps de retard, dans le cas inverse.

Ces effets spéciaux sont donc très importants pour la qualité de l'expérience utilisateur.

Utilisation avancée

script.aculo.us permet de proposer des solutions qui n'étaient jusqu'à présent disponibles que dans les applications de type client lourd. Parmi ces fonctionnalités avancées, script.aculo.us propose le glisser-déplacer, les listes à trier graphiquement et les champs HTML qui se remplissent *via* des appels côté serveur.

Dans sa version 1.5, que nous utilisons dans le cadre de cet ouvrage, script.aculo.us s'intègre mal à DWR pour ces fonctionnalités avancées. C'est la raison pour laquelle nous allons les étudier en dehors de l'étude de cas et de DWR.

Édition de texte à chaud

Cette fonctionnalité permet de transformer une chaîne de caractères normale en un champ de saisie HTML. Nous pouvons ainsi modifier certains éléments d'une page en cliquant simplement dessus.

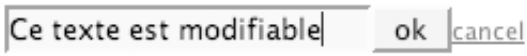
En voici un exemple de code :

```
<p id="texteModifiable">Ce texte est modifiable</p>
```

```
<script language="JavaScript">
 new Ajax.InPlaceEditor('texteModifiable',
  '${ctx}/tudu/example/saveText');
</script>
```

Le résultat de ce code est illustré à la figure 9.3.

Figure 9.3
Texte modifiable à chaud



La fonction `Ajax.InPlaceEditor` prend en premier paramètre l'identifiant du bloc HTML à modifier (un paragraphe dans notre exemple) et en second une URL qui sera chargée de traiter la requête. Cette fonction contourne donc DWR, comme indiqué précédemment.

L'URL appelée a pour rôle de sauvegarder le changement de texte. Elle reçoit en POST le paramètre `value`, contenant le nouveau texte à enregistrer, et doit renvoyer la chaîne de caractères à afficher. Cette dernière peut donc être différente du texte envoyé, si celui-ci n'est pas valide, par exemple. Une simple servlet est parfaitement adaptée pour ce type de traitement.

Création d'une liste déroulante dynamique

Cette fonctionnalité est souvent appelée Google Suggest, Google ayant popularisé ce type de liste déroulante dans une version bêta de son site.

La figure 9.4 en illustre un exemple avec une recherche sur le mot-clé « j2ee ».

Figure 9.4
Interface de Google Suggest

De même que la fonction précédente, cet exemple court-circuite DWR et s'utilise de la manière suivante :

```
<input type="text" id="utilisateurs" name="utilisateurs"/>
<div id="utilisateurs_autocomplete"></div>
```

```
<script language="JavaScript">
  new Ajax.Autocompleter("utilisateurs",
    "utilisateurs_autocomplete",
    "${ctx}/tudu/example/findUsers",
    {});
</script>
```

La fonction `Ajax.AutoCompleter` prend en premier paramètre l'identifiant de la zone de texte éditable, en deuxième paramètre l'identifiant du layer DHTML servant à afficher les valeurs dynamiques, en troisième paramètre l'URL chargée de traiter la requête et en quatrième paramètre des options.

L'URL traitant la requête va recevoir en méthode POST le contenu du champ HTML. La clé utilisée en paramètre est l'attribut `name` de la balise `input`. Cette URL devra retourner le contenu affiché dans le layer DHTML, nommé `utilisateurs_autocomplete` dans notre exemple, sous forme de liste HTML :

```
<ul>
  <li>j2ee</li>
  <li>j2ee tutorial</li>
</ul>
```

Cette fonctionnalité améliore l'utilisabilité du site, mais elle est également extrêmement gourmande en ressources. Nous ne pouvons que conseiller l'utilisation d'un cache côté serveur. Il existe plusieurs solutions adaptées pour ce type de besoin, en particulier Ehcache (*http://ehcache.sourceforge.net),* que nous présentons au chapitre 11, ou OSCache *(http://www.opensymphony.com/oscache/),* qui propose un filtre de servlet intéressant pour cette situation.

Si notre but est la création d'un vrai moteur de recherche, l'utilisation de bases de données n'est pas recommandée, car elles sont lentes et mal adaptées à ce type de traitement. L'utilisation de Lucene *(http://lucene.apache.org/)* paraît plus appropriée, d'autant que cette bibliothèque peut être intégrée à Spring. Nous avons pu rencontrer l'architecte d'un site Internet français à fort trafic, qui utilise la combinaison Spring/Lucene pour obtenir des temps de réponse inférieurs à 100 millisecondes par requête. De tels résultats rendent possible la création d'un site Web de type Google Suggest en utilisant uniquement des technologies Java Open Source.

Utilisation de Prototype

Prototype est une bibliothèque JavaScript, disponible à l'adresse *http://prototype.conio.net/.* Elle est utilisée en interne par script.aculo.us, car elle fournit un ensemble d'utilitaires simplifiant la manipulation de l'arbre DOM et les appels AJAX. Pour un développeur JavaScript, cette bibliothèque d'un investissement minime en apprentissage est certainement utile.

Ses fonctions les plus simples et les plus utiles, qui ne sont pas spécifiques à J2EE ou même à AJAX, sont détaillées dans les sections suivantes.

La fonction *$()*

La fonction `$()` est un raccourci vers la fonction JavaScript `document.getElementById()`, qui est très souvent utilisée en AJAX. En voici un exemple d'utilisation :

```
<div id="exempleDiv">
 <p>Bonjour!</p>
</div>

<script language="JavaScript">
 var exemple = $('exempleDiv');
 alert(exemple.innerHTML);
</script>
```

Cette fonction est capable de retourner un tableau d'éléments si nous lui donnons en paramètres plusieurs identifiants :

```
var exemples = $('exempleDiv', 'exempleDiv2', 'exempleDiv3');

for(i=0; i<exemples.length; i++) {
  alert(exemples[i].innerHTML);
}
```

La fonction *$F()*

Il s'agit là aussi d'un raccourci, capable de retrouver n'importe quel élément d'un formulaire HTML, quel que soit son type.

Cela donne avec l'exemple précédent :

```
<input type="text" id="exempleInput" value="Bonjour!"/>

<script language="JavaScript">
 var exemple = $F('exempleInput');
 alert(exemple);
</script>
```

La fonction *Try.these()*

Cette fonction est utile pour créer des sites Web robustes, capables d'afficher correctement des pages Web malgré les problèmes de compatibilité posés par le JavaScript. Nous en aurons certainement besoin si nous voulons utiliser AJAX sur un site disponible sur Internet.

La fonction `Try.these()` permet de tester plusieurs fonctions à la suite, jusqu'à ce que l'une d'elles marche correctement. En voici un exemple d'utilisation :

```
function exemple(){
  return Try.these(
    function() {
      // fonctionnement normal.
    },
```

```
    function() {
      // mode dégradé, utilisé si la première fonction a échoué
    )
  );
}
```

Conclusion

Après une introduction à AJAX et au Web 2.0, nous nous sommes arrêtés sur deux grandes bibliothèques JavaScript complémentaires, DWR et script.aculo.us. DWR permet d'utiliser côté client des Beans Spring provenant d'une application Java/J2EE en back-end. script.aculo.us permet d'ajouter des effets spéciaux aux pages Web. La combinaison de ces différentes technologies permet de développer des sites Web attractifs, radicalement différents de ceux d'avant 2005. C'est là un changement important en terme de fonctionnalités et d'utilisabilité, qu'il va être important de maîtriser dans les années à venir.

Nous avons vu qu'avec cette boîte à outils il n'était pas très difficile de créer des pages Web utilisant AJAX. Les difficultés rencontrées se trouvent plutôt du côté de l'utilisation d'AJAX, qui présente deux inconvénients : cette technique bombarde les serveurs de requêtes HTTP, mettant en danger leur montée en charge, et risque de troubler les utilisateurs habitués à des applications plus traditionnelles. À ces deux problèmes, nous avons vu qu'il existait des solutions : monitoring des performances et tuning pour le premier, utilisation d'effets spéciaux pour le second.

AJAX est une technique simple d'emploi et bien outillée. Nous ne saurions trop conseiller aux concepteurs de sites Web de l'ajouter à leur arsenal technique.