TD4 : Wikis, Servlets & Projet

L'objet de cette séance est de vous familiariser avec les sockets et les servlets, et d'introduire le projet. Celui-ci consiste à écrire une servlet correspondant à un mini-wiki.

1 pmWiki

Un wiki est un service web d'édition collaborative de pages et documentations en ligne. En général il se présente sous la forme de scripts ou CGI sur serveur, permettant de stocker et d'éditer des pages web dans une syntaxe simplifiée. Le wiki se charge alors de traduire et d'afficher sa syntaxe interne en HTML/XHTML à la volée, au moment de l'accès à une page. Nous proposons ici une familiarisation avec les wikis, par l'intermédiaire de pmWiki, un ensemble de scripts php implémentant ces fonctionnalités.

Exercice 1 : Utilisation de pmwiki

Pour cette question, utiliser le pmwiki installé sur `http://www.labri.fr/~franco/wiki/pmwiki.php`.

1. comment créé-t-on une page ? créer une deuxième page à partir de la page principale par défaut.
2. reproduisez le plus fidèlement possible dans cette deuxième page le texte formaté dans l'image suivante : `/net/cremi/jfranco/tmp/page.png`
3. sur cette page mettre un lien vers la page par défaut du wiki.
4. ajoutez des couleurs aux titres. Créer une liste de liens vers chaque titre de cette page.
5. insérez une image

2 Servlets

Les *servlets* sont des applications Java fonctionnant du côté serveur au même titre que les CGI et les langages de script côté serveur tels que ASP ou bien PHP. Les servlets permettent donc de gérer des requêtes HTTP et de fournir au client une réponse HTTP dynamique (donc de créer des pages web dynamiques). Un des atouts des servlets java c'est la possibilité d'utiliser l'API Java. Les servlets sont indépendantes du serveur web et une servlet s'exécute dans un **conteneur de servlets** qui fait le lien entre les servlets et le serveur web. Le schéma d'exécution est le suivant : le client fait une requête http au serveur, le serveur fait suivre la requete au conteneur de servlets qui lance la servlet concernée et cette derniére exécute la tâche et donne la réponse au serveur. Le serveur renvoie ensuite le résultat au client.

Une ou plusieurs servlets constituent une *web application*. Leur principte de fonctionnement est décrite par une spécification menée par Sun Microsystems `http://java.sun.com/products/servlet/`. Il existe plusieurs conteneurs de servlets (Apache Tomcat, JBoss, ...). Nous allons expliquer comment installer la version 6.0 de Tomcat `http://tomcat.apache.org/` et comment déployer votre première servlet.

Exercice 2 : Installation de tomcat

Tout d'abord la version 6.0 de Tomcat requiert la version 1.5 de java (exécuter dans le terminal java -version pour connaître la version de java installée sur votre machine). Télécharger la version binaire 6.0 de tomcat ici `http://tomcat.apache.org/download-60.cgi` (récupérez le fichier .tar.gz) et mettez le dans un répertoire (je considére qu'il est dans le répertoire ∼/CATALINA). Exécuter ces différentes commandes dans le terminal (`prompt` est le prompt que vous affiche votre terminal) :

```
prompt#: cd bin
prompt#: tar -xzvf apache-tomcat-6.0.14.tar.gz
prompt#: mv apache-tomcat-6.0.14 tomcat6
```

Tomcat a besoin de java pour lancer les servlets et pour cela il lui faut lui indiquer ou se trouve java. Dans votre fichier `.bashrc` ou `.bash_profile` (que l'on va appeler par la suite *fichier de configuration*) ajouter cette ligne (chez moi RepertoireBaseJava=/usr/java/latest :

```
export JAVA_HOME=RepertoireBaseJava
```

Pour programmer des servlets, il faut utiliser l'API des servlets. l'API des servlets n'est pas fournie par défaut dans le jdk. Heureusement cette api est fournie avec tomcat (on peut aussi le télécharger ici `http://java.sun.com/products/servlet/`). Ajouter cette ligne dans votre fichier de configuration :

```
export CLASSPATH=$CLASSPATH:~/CATALINA/tomcat6/lib/servlet-api.jar
```

Tomcat est fournie avec deux fichiers : `startup.sh` et `shutdown.sh` qui se trouvent tous les deux dans le répertoire bin de tomcat. Il faut lancer dans le terminal `~/CATALINA/tomcat6/bin/startup.sh` pour le lancer et `~/CATALINA/tomcat6/bin/shutdown.sh` pour le stopper. Vous pouvez aussi créer des liens symboliques vers ces fichiers. Pour vérifier si tomcat est lancé il faut taper dans un navigateur `http://localhost:8080`, vous devriez normalement voir la page d'accueil de tomcat (si ce n'est pas le cas tomcat n'a pas pu se lancer ou n'est pas lancer). Vous trouverez la doc de tomcat ici `http://localhost:8080/docs` (c'est une web-application). On va maintenant voir comment déployer une servlet comme *web application*.

Exercice 3 : Votre première servlet

Pour déployer une web application dans tomcat, il faut le placer dans le répertoire webapps de tomcat (ce répertoire peut-être changé en modifiant l'attribut `appBase` dans le fichier de configuration `context.xml` ou `server.xml`). Chaque web application est dans un répertoire qui porte le nom de la web application (par exemple vous pouvez vérifier que la web application docs est placé dans le répertoire tomcat6/webapps/docs). Supposons que notre première web application aura le nom HelloWorld. Voici un exemple de servlet que l'on va déployer :

```
import javax.servlet.*;
import java.io.*;

public class HelloWorld extends GenericServlet {
     public void service (ServletRequest req, ServletResponse res) {
           try {
                 PrintWriter out = response.getWriter ();
                 out.println ("<html>");
                 out.println ("<head>");
                 out.println ("<title>Hello Everybody !</title>");
                 out.println ("</head>");
                 out.println ("<body>");
                 out.println ("Hello world !");
                 out.println ("</body>");
                 out.println ("</html>");
           } catch (IOException e) {
                 e.printStackTrace();
           }
   }
}
```

Pour déployer une web application nommée HelloWorld il faut créer dans le répertoire `webapps` de tomcat un répertoire `HelloWorld`, qui est le répertoire de base de la web application. Puis dans le répertoire HelloWorld, créer les répertoires `WEB-INF`, `WEB-INF/classes` et `WEB-INF/lib`. Tous les fichiers `.class` doivent être placés dans le répertoire `WEB-INF/classes` et toutes les librairies (les fichiers `.jar`) doivent être placées dans le répertoire `WEB-INF/lib`. Dans le répertoire de base de la web application, on place les fichiers html et autres. En général on crée aussi un répertoire `src` dans le répertoire de base, ce dernier contient tous les fichiers `.java`.

Chaque web application est accompagnée d'un fichier de description, `web.xml` que l'on place dans le répertoire `WEB-INF`. Ce dernier est un fichier `xml` qui décrit la web application, en particulier les différentes classes qui vont être utilisées et comment y accéder. Il y a en particulier deaux balises importantes : *servlet* et *servlet-mapping*. Ce fichier xml commence obligatoirement par ces lignes dans tomcat 6.x.x :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
   javaee/web-app_2_5.xsd" version="2.5">
</web-app>
```

La balise *servlet* décrit le nom de la servlet et la classe java Servlet qui lui est associée. Dans la balise *servlet-mapping* on décrit l'url qui permet d'accéder à la servlet. Par exemple

```
<servlet>
        <servler-name>HelloWorld</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
</servlet>
```

déclare une servlet *HelloWorld* associée à la classe HelloWorld. Maintenant pour dire comment y accéder, il faut utiliser la balise servlet-mapping. Voici un exemple :

```
<servlet-mapping>
        <servler-name>HelloWorld</servlet-name>
        <url-pattern>/project/example</url-pattern>
</servlet-mapping>
```

dit que pour accéder à la servlet il faut taper dans le navigateur `http://monURL/HelloWorld/project/example` (en local `monURL` c'est `localhost:8080`).

On peut aussi donner des paramétres à la servlet (ces paramétres sont utilisés lors de l'initialisation de la servlet, ils sont utilisés par la méthode *init* de la servlet). Voici un exemple de fichier web.xml :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
    javaee/web-app_2_5.xsd" version="2.5">

    <display-name>Hello World </display-name>
    <description> Ma premiére Servlet </description>
    <context-param>
      <param-name>webmaster</param-name>
      <param-value>webmaster@mycompany.com</param-value>
      <description>Adresse email du webmaster.</description>
    </context-param>

    <servlet>
```

```
      <servlet-name>HelloWorld</servlet-name>
      <servlet-class>HelloWorld</servlet-class>
      <init-param>
        <param-name>count</param-name>
        <param-value>0</param-value>
      </init-param>
      <init-param>
        <param-name>max</param-name>
        <param-value>50</param-value>
      </init-param>
    </servlet>

    <servlet-mapping>
      <servlet-name>HelloWorld</servlet-name>
      <url-pattern>/project/example</url-pattern>
    </servlet-mapping>

    <session-config>
      <session-timeout>30</session-timeout>    <!-- 30 minutes maximum par session -->
    </session-config>
</web-app>
```

On peut avoir plusieurs servlets dans une web application, toutes décrites dans le même fichier `WEB-INF/web.xml`.

Tester le bon fonctionnement de cette Servlet `HelloWorld`.

3 Le projet : la servlet Wiki

L'objet du projet est de créer un wiki sous forme d'une servlet java (feuille de projet distribuée bientôt). Un squelette de servlet vous est donné pour le projet, qui implémente quelques fonctionnalités de base du wiki. Nous nous familiarisons ici avec ce code, que vous compléterez par la suite dans le cadre du projet pour rajouter des fonctionnalités plus avancées.

Exercice 4 : Déploiement de la servlet du Wiki

Si vous avez déjà fait une copie locale du dépôt subversion `inf157`, metter le simplement à jour via la commande `svn update`, et le code fourni sera dans `trunk/projets/sujet/code/`. Sinon récupérez une copie du dépot avec `svn --username login co https://services.emi.u-bordeaux1.fr/projet/svn/inf157/`.

Le script `deploy.sh` se charge de compiler la servlet et de la déployer dans tomcat. Si votre variable `CATALINA_HOME` ne pointe pas vers votre répertoire de tomcat, vous devez éditer ce script pour lui affecter la bonne valeur. Ce script effectue ces opérations :

1. création de la structure des répertoires de la servlet (via le script `createDirectories.sh`) ;
2. recopie des sources java dans `webapps/src`, compilation, et déplacement des binaires dans `webapps/deploy/classes` (via le script `compil.sh`) ;
3. recopie du fichier `web.xml` dans `WEB_INF` ;
4. recopie du fichier `params.txt` dans `webapps/deploy` ;
5. création d'une archive `.war` pour l'importation dans tomcat ;
6. déploiement de la servlet (à partir de l'archive) dans tomcat (et relancement de celui-ci).

Déployez la servlet du wiki sur votre serveur tomcat, et testez les fonctionnalités implémentées du wiki. Pour cela, éditer le fichier `params.txt` pour indiquer le répertoire qui stockera les pages de votre wiki (le script ne le crée pas) et éditer le fichier `web.xml` si vous souhaitez redéfinir l'URL d'accès à la Servlet.

Par défaut, l'URL d'une page `X` du wiki est : `http ://localhost :8080/MaServletWiki/X`

Exercice 5 : Rajout d'éléments syntaxiques

Le wiki comporte une classe `WikiParser` qui interprète la syntaxe du wiki. Seulement quelques éléments syntaxiques sont actuellement interprétés. La classe se base sur les classes outils de reconnaissances d'expressions régulières de Java pour effectuer des remplacements d'éléments syntaxiques par leur code équivalent en XHTML. Après avoir examiné la classe ainsi que la documentation de la classe `java.util.regex.Pattern` pour comprendre la syntaxe des expressions régulières interprétées, compléter cette classe pour inclure l'interprétation de titres avec la même syntaxe que pmWiki : `! ! Titre` devient alors `<h1>Titre</h1>`, `! ! ! Titre 2` devient `<h2>Titre</h2>`, etc.