Version : **2023.01**.

Dernière mise-à-jour : 2023/12/04 09:17

SER301 - Présentation des Technologies

Contenu du Module

- **SER301 - Présentation des Technologies**
  - Contenu du Module
  - Présentation de Tomcat 8
    - Historique et différentes versions
  - Rappel sur les applications Web en Java
  - Contenu statique, dynamique, Servlets, JSPs et Composants EJB
    - Servlets
    - JSP
    - Enterprise JavaBeans - EJB
  - Le Modèle MVC
  - Les Modules Java EE
    - Modules Web
    - Modules EJB
    - Modules Clients
    - Modules de Connecteurs
  - Positionnement d'Apache Tomcat dans la norme Java EE
    - Structure d'une Application Web
    - Le Descripteur de Déploiement web.xml
    - Les Sessions HTTP

Présentation de Tomcat 8

Historique et différentes versions

Historiquement le serveur d'applications Java **Jakarta Tomcat** appartenait à la première catégorie du projet **Jakarta** issu de la **Fondation Apache**.

Le projet Jakarta est est divisé en trois catégories liées aux technologies **Java** :

- Les serveurs d'application,
- Les bibliothèques, outils et API,
- Les frameworks.

Tomcat a ses origines dans les **Conteneurs Web**, aussi appelés *Moteur de Servlet*, de SUN Microsystems et de la fondation Apache :

- Java Web Server
- JServ

En **1999**, SUN Microsystems donne le code de Java Web Server à la fondation Apache. Tomcat est né de la fusion du projet JServ avec ce dernier.

Tomcat est devenu un projet important et de ce fait a trouvé sa propre place au sein de la Fondation Apache. Quittant le projet Jakarta, Tomcat est maintenant appelé **Apache Tomcat**.

Les différentes versions de Tomcat sont :

Version Tomcat	API Servlet	API JavaServer Pages	Spécification Java JEE
3.x	2.2	1.1	J2EE 1.2
4.x	2.3	1.2	J2EE 1.3
5.x	2.4	2.0	J2EE 1.4
6.x	2.5	2.1	Java EE 5 et +
7.x	3.0	2.2	Java EE 6 et +
8.x	3.1	2.3	Java EE 7 et +
9.x	4.0	2.3	Java EE 8 et +

Version Tomcat	API Servlet	API JavaServer Pages	Spécification Java JEE
10.0.11	5.0	3.0	Java EE 8 et +

Rappel sur les applications Web en Java

Java prend ses origines dans le développement d'applications pour des terminaux mobiles. Débuté en **1991**, le projet **Star 7** a d'abord été basé sur l'utilisation du langage C++. Cependant ce langage, à cause de la gestion mémoire contraignante, a été mis de côté et remplacer par un langage nouveau le **C++–**. Ce langage a été ensuite connu sous le nom **OAK** pour ensuite devenir **Java**.

En **1995** la première version du **JDK** (*Java Development Kit*) a été rendu disponible. Il permettait de développer :

- des applications graphiques,
- des applications client/serveur,
- des applets (applications embarquées dans des pages HTML).

Lors du développement d'une application, la compilation du code source Java donne un format de fichier spécifique appelé **byte-code**. Ce format de fichier est interprété par une **machine virtuelle java**.

Java se décompose en plusieurs plate-formes :

- JSE (*Java Standard Edition*),
  - une plate-forme de base pour le développement d'applications clients/serveurs et graphiques ainsi que des applets qui est en deux formes :
    - JDK (*Java Development Kit*),
      - nécessaire pour le développement sous Java,
    - JRE (*Java Runtime Environment*),
      - exécute des applications java,
- JEE (*Java Enterprise Edition*),
  - une extension de JSE qui permet le développement d'applications qui s'exécutent sur un serveur d'application et qui peuvent être exploitées par :
    - des clients légers, tels des navigateurs web,
    - des clients lourds, tels des applications graphiques fenêtrées.
- JME (*Java Micro Edition*),
  - une plate-forme de développement d'applications mobiles utilisées dans des téléphones mobiles, pocket pc etc.

En **1998**, la version 1.2 de ces plate-formes, connue sous le nom de **Java 2** a donné naissance à l'utilisation des termes **J2SE**, **J2EE** et **J2ME**.

En **2004**, la version 1.5 a été publiée, donnant lieu à **Java 5**.

En **2006**, la version 1.6 ou **Java 6** a été publiée et le chiffre 2 retiré de J2SE et J2EE.

La version **actuelle** de la plate-forme est la version **8**.

Le JRE de la plate-forme JSE est constitué des éléments suivants :

- **JVM** (*Java Virtual Machine*),
  - Il existe une JVM pour la majorité des systèmes d'exploitation, rendant ainsi portables les applications Java,
  - La JVM gère directement la mémoire des applications,
- **Bibliothèque de classe Java**,
  - Des composants logiciels prêt à l'emploi,

Outre les deux éléments précédents, le JDK de la plate-forme JSE contient des **outils de développement** suivants :

- **javac**,
  - un compilateur de code source Java,
- **java**,
  - un interpréteur,
- **javadoc**,
  - un générateur de documentation.

Contenu statique, dynamique, Servlets, JSPs et Composants EJB

Le modèle de développement JEE contient trois types de composants logiciels.

Servlets

Le rôle d'une servlet dans une application est :

- de recevoir les requêtes des clients,
- d'en extraire des informations,
- de formater ces informations,
- de préparer des données nécessaires à la génération d'une réponse.

Les servlets sont :

- des composants logiciels écrits en Java,
- des composants orientés requête/réponse,
- portables et évolutives,
- chargées en mémoire lors de leurs premier appel,
- déchargées de la mémoire lors de l'arrêt de l'application ou du serveur.

Il n'y a qu'une seule instance d'une servlet en mémoire. Le serveur utilise ensuite un **thread** pour traiter chaque requête.

La servlet est une classe Java qui a un cycle de vie spécifique. La servlet est :

- **instancier** et **charger** en mémoire,
- **initialiser** grâce à l'appel de la méthode **init()** par le serveur,
- **prête** - les requêtes sont satisfaites par la méthode **service()**,
- **détruite** par le serveur en appelant la méthode **destroy()**,
- **nettoyée** par la machine virtuelle Java.

JSP

Proches des pages PHP et ASP, les **JavaServer Pages** permettent le développement de contenu mélangé statique et dynamique.

Le rôle d'une page JSP dans une application est de prendre en charge la partie visuelle de l'application en présentant des données au client.

Une page JSP :

- possède une extension **.jsp**,
- est un squelette de code HTML pour la partie statique,
- contient du code Java pour l'intégration des données dynamiques,

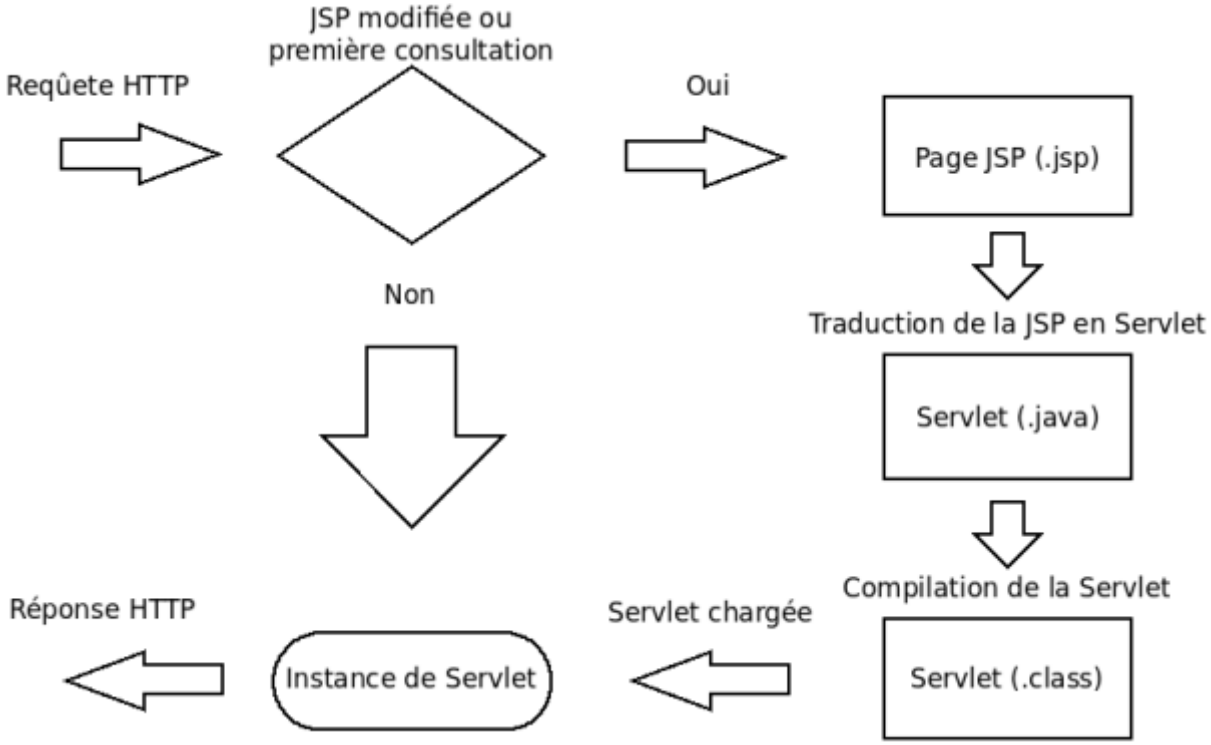
- doit être traitée par le serveur avant d'être renvoyée au client.

Le traitement est effectué au premier appel de la page et à chaque fois que la page est modifiée par un programmeur. C'est pour cette raison que le serveur a besoin d'un JDK car il transforme la JSP en classe Java puis la compile en servlet.

Une requête vers une page JSP peut être résumée ainsi :

- l'arrivée de la première requête,
- la transformation de la page .jsp en code source java (**.java**),
- la compilation du code source en classe Java (**.class**) qui devient ainsi une servlet,
- le départ de la première réponse,
- l'arrivée d'une deuxième requête,
- la délégation de la requête par le serveur *directement* à la servlet précédemment créée,
- le départ de la deuxième réponse,
- etc ...

Le schéma ci-dessous résume ce mécanisme :



Voici un exemple simplifié d'une page jsp :

```
<HTML>
    <HEAD>
        <TITLE>JSP Page Example</TITLE>
    </HEAD>
    <BODY>
        This is a random number:
        <%= Math.random() %>
    </BODY>
</HTML>
```

Enterprise JavaBeans - EJB

Les EJB sont des composants métiers distribués. Il existe deux types d'EJB :

- EJB Session,
- EJB piloté par message, appelé **MDB** (*Message Driven Bean*).

Important - Notez que les **EJB Entités** n'existent plus et sont aujourd'hui remplacés par la nouvelle API de persistance Java : **JPA** (*Java Persistence API*). Les EJB sont hébergés dans une partie spécifique d'un serveur d'applications Java EE. Apache Tomcat ne disposant pas de cet environnement n'utilise pas les EJB.

Le Modèle MVC

La plate-forme Java EE comporte un cycle de conception d'applications :

- le développement des composants applicatifs,
- l'assemblage des composants en modules,
- l'assemblage des modules en applications,
- le déploiement de l'application.

Afin de suivre ce cycle, le développement d'applications utilise le **modèle MVC** (*Model View Controller*) qui :

- a ses origines dans le langage **SmallTalk** au début des années 1980,
- divise l'application en trois parties distinctes :
    - le **modèle**,
    - la **vue**,
    - le **contrôleur**.

Ce modèle, appliqué au développement dans la plate-forme JEE, correspond aux composants logiciels suivants :

- le modèle = **EJB/JPA**,
- la vue = **JSP**,
- le controleur = **servlets**.

Lors d'une requête d'un client vers l'application ainsi développée on constate que :

- la requête http est reçue par la servlet qui extrait des informations,
- ces informations sont utilisées pour appeler des composants EJB/JPA adéquats,
- les composants EJB/JPA manipulent les données (lecture, enregistrement, mise à jour ...),
- les composants EJB/JPA retournent les données ainsi modifiées à la servlet,
- la servlet appelle la JSP adéquate,
- la JSP inclut les données dans la génération de la réponse au client.

Pour simplifier le développement en utilisant le modèle MVC, certains outils sont disponibles :

- **Apache Struts**,
- **JavaServer Faces**.

Les Modules Java EE

Les différentes ressources d'une application sont organisées en 4 **modules** selon le rôle qu'elles jouent :

- modules pour l'interface web,
- modules pour les clients lourds,
- etc ...

Modules Web

Ces modules intègrent :

- des servlets,
- des JSP,
- des pages statiques en html,
- des bibliothèques de classes Java ayant une extension **.jar**.

Ces modules possèdent un **descripteur de déploiement** sous la forme d'un fichier appelé **web.xml** et sont assemblés dans un fichier compressé au

format **zip** ayant une extension **.war**.

Modules EJB

Ces modules intègrent :

- des fichiers de code Java,
- des fichiers de configuration spécifiques à un serveur donné.

Ces modules possèdent un **descripteur de déploiement** sous la forme d'un fichier appelé **ejb-jar.xml** et sont assemblés dans un fichier compressé au format **zip** ayant une extension **.jar**.

Modules Clients

Ces modules permettent l'utilisation des EJB à travers un client lourd ayant été développé avec un API de programmation tel **AWT** ou **SWING**.

Ces modules possèdent un **descripteur de déploiement** sous la forme d'un fichier appelé **application-client.xml** et sont assemblés dans un fichier compressé au format **zip** ayant une extension **.jar**.

Modules de Connecteurs

Ces modules permettent l'accès aux données externes en utilisant des API tels **JDBC**, **JNDI** ou encore **JCA**.

Ces modules possèdent un **descripteur de déploiement** sous la forme d'un fichier appelé **ra.xml** et sont assemblés dans un fichier compressé au format **zip** ayant une extension **.rar**.

Positionnement d'Apache Tomcat dans la norme Java EE

Le serveur Tomcat ne dispose pas de l'environnement spécifique nécessaire pour héberger les EJB. Par conséquent il n'est pas possible d'utiliser les EJB

avec Tomcat. En effet, des modules cités précédemment, seuls les modules web peuvent être exploités par un serveur Tomcat.

Structure d'une Application Web

Chaque application web est stockée dans un répertoire distinct dans le répertoire **/usr/tomcatX/webapps/** où X représente la version de Tomcat.

Le répertoire de l'application est organisé de la façon suivante :

```
|-- MonApplication
|   |-- WEB-INF
|   |   |-- classes
|   |   |-- lib
|   |   `-- web.xml
|   |-- images
|   |-- page.jsp
|   `-- public.html
```

Le répertoire **WEB-INF** est la partie privée de l'application tandis que les fichiers stockés en dessus du répertoire **MonApplication** constituent la partie publique.

Dans le répertoire WEB-INF se trouve :

- un répertoire **classes** pour stocker les classes Java,
- un répertoire **lib** pour stocker les bibliothèques de code Java,
- le descripteur de déploiement **web.xml**.

Chaque application web est accessible par une URL unique composée :

- du nom d'hôte du serveur,
- d'un numéro de port spécifique,
- du chemin du **contexte d'application web** qui représente une vue globale de l'application.

L'URL prend donc la forme **http://nom_d_hote:port/contexte**.

Le Descripteur de Déploiement web.xml

Le **Descripteur de Déploiement** a pour but d'aider le serveur à installer l'application qu'il décrit. Il peut contenir 5 informations principales :

- des paramètres d'initialisation de l'application ou des servlets,
  - informations de type texte consultées par les servlets et JSP de l'application, par exemple, l'adresse email de l'administrateur de l'application,
- la définition des servlets et des JSP,
  - la déclaration de chaque servlet est necéssaire afin qu'elle puisse être utilisée. Les pages JSP doivent aussi être déclarées si elles ont besoin de paramètres d'initialisation particuliers,
- la correspondance des servlets avec des URLs,
  - les classes Java des servlets se trouvent dans la partie privée de l'application : **WEB-INF/classes**. Le serveur d'applications doit associer un URL avec chaque servlet,
- les détails des pages d'accueil et d'erreur de l'application,
  - définition de la page d'acceuil de l'application ainsi que la correspondance entre des erreurs HTTP spécifiques et les pages HTML à afficher,
- des contraintes de sécurité,
  - informations concernant quels utilisateurs ont accès à quelles ressources ainsi que comment ces utilisateurs doivent s'authentifier.

Voici un exemple abrégé :

```
<file>
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">

  <display-name>Tomcat Manager Application</display-name>
  <description>
    A scriptable management web application for the Tomcat Web Server;
    Manager lets you view, load/unload/etc particular web applications.
  </description>
```

```
  <servlet>
    <servlet-name>Manager</servlet-name>
    <servlet-class>org.apache.catalina.manager.ManagerServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>HTMLManager</servlet-name>
    <servlet-class>org.apache.catalina.manager.HTMLManagerServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <!-- Uncomment this to show proxy sessions from the Backup manager or a
         StoreManager in the sessions list for an application
    <init-param>
      <param-name>showProxySessions</param-name>
      <param-value>true</param-value>
    </init-param>
    -->
    <multipart-config>
      <!-- 50MB max -->
      <max-file-size>52428800</max-file-size>
      <max-request-size>52428800</max-request-size>
      <file-size-threshold>0</file-size-threshold>
    </multipart-config>
  </servlet>
  <servlet>
    <servlet-name>Status</servlet-name>
    <servlet-class>org.apache.catalina.manager.StatusManagerServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
```

```
    <param-value>0</param-value>
  </init-param>
</servlet>

<servlet>
  <servlet-name>JMXProxy</servlet-name>
  <servlet-class>org.apache.catalina.manager.JMXProxyServlet</servlet-class>
</servlet>

<!-- Define the Manager Servlet Mapping -->
<servlet-mapping>
  <servlet-name>Manager</servlet-name>
    <url-pattern>/text/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>Status</servlet-name>
  <url-pattern>/status/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>JMXProxy</servlet-name>
    <url-pattern>/jmxproxy/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>HTMLManager</servlet-name>
  <url-pattern>/html/*</url-pattern>
</servlet-mapping>

<filter>
  <filter-name>SetCharacterEncoding</filter-name>
  <filter-class>org.apache.catalina.filters.SetCharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
```

```
  </filter>

  <filter-mapping>
    <filter-name>SetCharacterEncoding</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>

  <filter>
    <filter-name>CSRF</filter-name>
    <filter-class>org.apache.catalina.filters.CsrfPreventionFilter</filter-class>
    <init-param>
      <param-name>entryPoints</param-name>
      <param-value>/html,/html/,/html/list,/index.jsp</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>CSRF</filter-name>
    <servlet-name>HTMLManager</servlet-name>
  </filter-mapping>

  <!-- Define a Security Constraint on this Application -->
  <!-- NOTE:  None of these roles are present in the default users file -->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>HTML Manager interface (for humans)</web-resource-name>
      <url-pattern>/html/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
       <role-name>manager-gui</role-name>
    </auth-constraint>
  </security-constraint>
  <security-constraint>
    <web-resource-collection>
```

```
    <web-resource-name>Text Manager interface (for scripts)</web-resource-name>
    <url-pattern>/text/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
     <role-name>manager-script</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>JMX Proxy interface</web-resource-name>
    <url-pattern>/jmxproxy/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
     <role-name>manager-jmx</role-name>
  </auth-constraint>
</security-constraint>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Status interface</web-resource-name>
    <url-pattern>/status/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
     <role-name>manager-gui</role-name>
     <role-name>manager-script</role-name>
     <role-name>manager-jmx</role-name>
     <role-name>manager-status</role-name>
  </auth-constraint>
</security-constraint>

<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Tomcat Manager Application</realm-name>
</login-config>
```

```
<!-- Security roles referenced by this web application -->
<security-role>
  <description>
    The role that is required to access the HTML Manager pages
  </description>
  <role-name>manager-gui</role-name>
</security-role>
<security-role>
  <description>
    The role that is required to access the text Manager pages
  </description>
  <role-name>manager-script</role-name>
</security-role>
<security-role>
  <description>
    The role that is required to access the HTML JMX Proxy
  </description>
  <role-name>manager-jmx</role-name>
</security-role>
<security-role>
  <description>
    The role that is required to access to the Manager Status pages
  </description>
  <role-name>manager-status</role-name>
</security-role>

<error-page>
  <error-code>401</error-code>
  <location>/WEB-INF/jsp/401.jsp</location>
</error-page>
<error-page>
  <error-code>403</error-code>
  <location>/WEB-INF/jsp/403.jsp</location>
</error-page>
```

```
  <error-page>
    <error-code>404</error-code>
    <location>/WEB-INF/jsp/404.jsp</location>
  </error-page>

</web-app>
```

On peut constater dans ce fichier la présence d'une en-tête qui spécifie la version **XML** :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Le fichier contient ensuite l'ouverture de la balise **<web-app…>** qui constitue l'élément racine du fichier. Il contient notamment le **schéma XML** utilisé pour valider le contenu du fichier. Un schéma XML est un fichier ayant une extension **.xsd**. L'élément **web-app** est fermé en fin de fichier avec la balise **</web-app>** :

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">
...
</web-app>
```

L'élément **<display-name>** donne ensuite un nom à l'application :

```
  <display-name>Tomcat Manager Application</display-name>
```

L'élément **<description>** indique une description de l'application :

```
  <description>
    A scriptable management web application for the Tomcat Web Server;
    Manager lets you view, load/unload/etc particular web applications.
```

```
  </description>
```

L'élément **<filter>** permet de spécifier le code de caractères à utiliser pour l'application :

```
  <filter>
    <filter-name>SetCharacterEncoding</filter-name>
    <filter-class>org.apache.catalina.filters.SetCharacterEncodingFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
  </filter>
```

Ensuite l'élément **<filter>** déclaré doit être associé à une URL grâce à l'élément **<filter-mapping>** :

```
  <filter-mapping>
    <filter-name>SetCharacterEncoding</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

Chaque servlet doit ensuite être déclarée en donnant la correspondance entre un **nom logique** et la classe Java de la servlet. La déclaration peut également contenir des paramètres d'initialisation propre à la servlet. Si les paramètres d'initialisation sont déclarées en dehors d'un élément **<servlet>** dans un élément **<context-param>**, ces paramètres s'appliquent à toutes les servlets et à toutes les JSP de l'application :

```
    <servlet>
    <servlet-name>Manager</servlet-name>
    <servlet-class>org.apache.catalina.manager.ManagerServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
  </servlet>
  <servlet>
    <servlet-name>HTMLManager</servlet-name>
```

```
    <servlet-class>org.apache.catalina.manager.HTMLManagerServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <multipart-config>
      <max-file-size>52428800</max-file-size>
      <max-request-size>52428800</max-request-size>
      <file-size-threshold>0</file-size-threshold>
    </multipart-config>
  </servlet>
  <servlet>
    <servlet-name>Status</servlet-name>
    <servlet-class>org.apache.catalina.manager.StatusManagerServlet</servlet-class>
    <init-param>
      <param-name>debug</param-name>
      <param-value>0</param-value>
    </init-param>
  </servlet>

  <servlet>
    <servlet-name>JMXProxy</servlet-name>
    <servlet-class>org.apache.catalina.manager.JMXProxyServlet</servlet-class>
  </servlet>
```

Ensuite chaque servlet déclarée doit être associée à une URL grâce à l'élément **<servlet-mapping>** :

```
  <servlet-mapping>
    <servlet-name>Manager</servlet-name>
      <url-pattern>/text/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Status</servlet-name>
    <url-pattern>/status/*</url-pattern>
```

```
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>JMXProxy</servlet-name>
      <url-pattern>/jmxproxy/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>HTMLManager</servlet-name>
    <url-pattern>/html/*</url-pattern>
  </servlet-mapping>
```

Finalement l'élément **<security-constraint>** est utilisé pour restreindre l'accès à certaines parties de l'application à certains utilisateurs :

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>HTML Manager interface (for humans)</web-resource-name>
     <url-pattern>/html/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>manager-gui</role-name>
   </auth-constraint>
 </security-constraint>
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>Text Manager interface (for scripts)</web-resource-name>
     <url-pattern>/text/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>manager-script</role-name>
   </auth-constraint>
 </security-constraint>
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>JMX Proxy interface</web-resource-name>
     <url-pattern>/jmxproxy/*</url-pattern>
```

```
    </web-resource-collection>
    <auth-constraint>
       <role-name>manager-jmx</role-name>
    </auth-constraint>
  </security-constraint>
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Status interface</web-resource-name>
      <url-pattern>/status/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
       <role-name>manager-gui</role-name>
       <role-name>manager-script</role-name>
       <role-name>manager-jmx</role-name>
       <role-name>manager-status</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Tomcat Manager Application</realm-name>
  </login-config>
  <security-role>
    <description>
      The role that is required to access the HTML Manager pages
    </description>
    <role-name>manager-gui</role-name>
  </security-role>
  <security-role>
    <description>
      The role that is required to access the text Manager pages
    </description>
    <role-name>manager-script</role-name>
  </security-role>
  <security-role>
```

```
    <description>
      The role that is required to access the HTML JMX Proxy
    </description>
    <role-name>manager-jmx</role-name>
  </security-role>
  <security-role>
    <description>
      The role that is required to access to the Manager Status pages
    </description>
    <role-name>manager-status</role-name>
  </security-role>
```

Les Sessions HTTP

Les **Sessions HTTP** permettent au serveur d'identifier un client lors de sa navigation. Chaque session est associée à un identifiant unique qui généré par le serveur et est envoyé vers le client. Quand le client envoie une requête, il inclut l'identifiant de la session.

L'identifiant de session peut être envoyé au client soit par le biais d'un cookie nommé **jsessionid**, soit inclus dans l'URLs envoyées par le serveur vers le client dans le cas où ce dernier n'accepte pas les cookies.

Les cookies :

- sont des informations transmises sous format texte,
- sont limités à 20 par site et 300 par navigateur,
- sont limités en taille à 4Ko.

La session expire à la fermeture du navigateur web ou après 30 minutes d'inactivité.
