



TECHNOLOGIES SERVLET, JSP ET FRAMEWORK DE PRÉSENTATION





PARTIE 1 : SURVOL DE LA PLATEFORME J2EE

PROGRAMMATION WEB AVEC JAVA

- Différentes technologies Java permettent de faire du développement Web à différents niveaux.

	Développement	
	Côté Client	Côté Serveur
.class autonome	applet	servlet
Source Java mixé avec code html	JavaScript	JSP

Pages
statiques

Pages
dynamiques

JAVA FRAMEWORK

- Le Java Framework (Java 2 Platform) est composé de trois éditions, destinées à des usages différents :
 - J2ME : Java 2 Micro Edition est prévu pour le développement d'applications embarquées, notamment sur des PDA (Personal Digital Assistant) et terminaux mobiles (téléphone portables, ...) ;
 - J2SE : Java 2 Standard Edition est destiné au développement d'applications pour ordinateurs personnels ;
 - J2EE : Java 2 Enterprise Edition, destiné à un usage professionnel avec la mise en oeuvre de serveurs.
- Chaque édition propose un environnement complet pour le développement et l'exécution d'applications basées sur Java et
- Comprend notamment une machine virtuelle Java (Java virtual machine) ainsi qu'une bibliothèque de classes.

LA PLATEFORME J2EE

- J2EE (Java 2 Enterprise Edition) est une norme proposée par la société Sun, portée par un consortium de sociétés internationales, visant à définir un standard de développement d'applications d'entreprises multi-niveaux, basées sur des composants.
- La plateforme J2EE désigne l'ensemble constitué des services (API) offerts et de l'infrastructure d'exécution.
- La norme J2EE comprend :
 - Les spécifications du serveur d'application (environnement d'exécution).
 - Des services (au travers d'API) qui sont des extensions Java indépendantes permettant d'offrir en standard un certain nombre de fonctionnalités.

Remarque : Sun fournit une implémentation minimale de ces API appelée J2EE SDK (J2EE Software Development Kit).

LES API DE J2EE

- Les API de J2EE peuvent se répartir en deux grandes catégories :
 - Les composants
 - Les composants web
 - Les composants métier
 - Les services
 - Les services d'infrastructures
 - Les services de communication

LES API DE J2EE

- Les composants :
 - Les composants Web : Servlets et JSP (Java Server Pages). Il s'agit de la partie chargée de l'interface avec l'utilisateur (on parle de logique de présentation).
 - Les composants métier : EJB (Enterprise Java Beans). Il s'agit de composants spécifiques chargés des traitements des données propres à un secteur d'activité (on parle de logique métier ou de logique applicative) et de l'interfaçage avec les bases de données.

LES API DE J2EE

■ Les services :

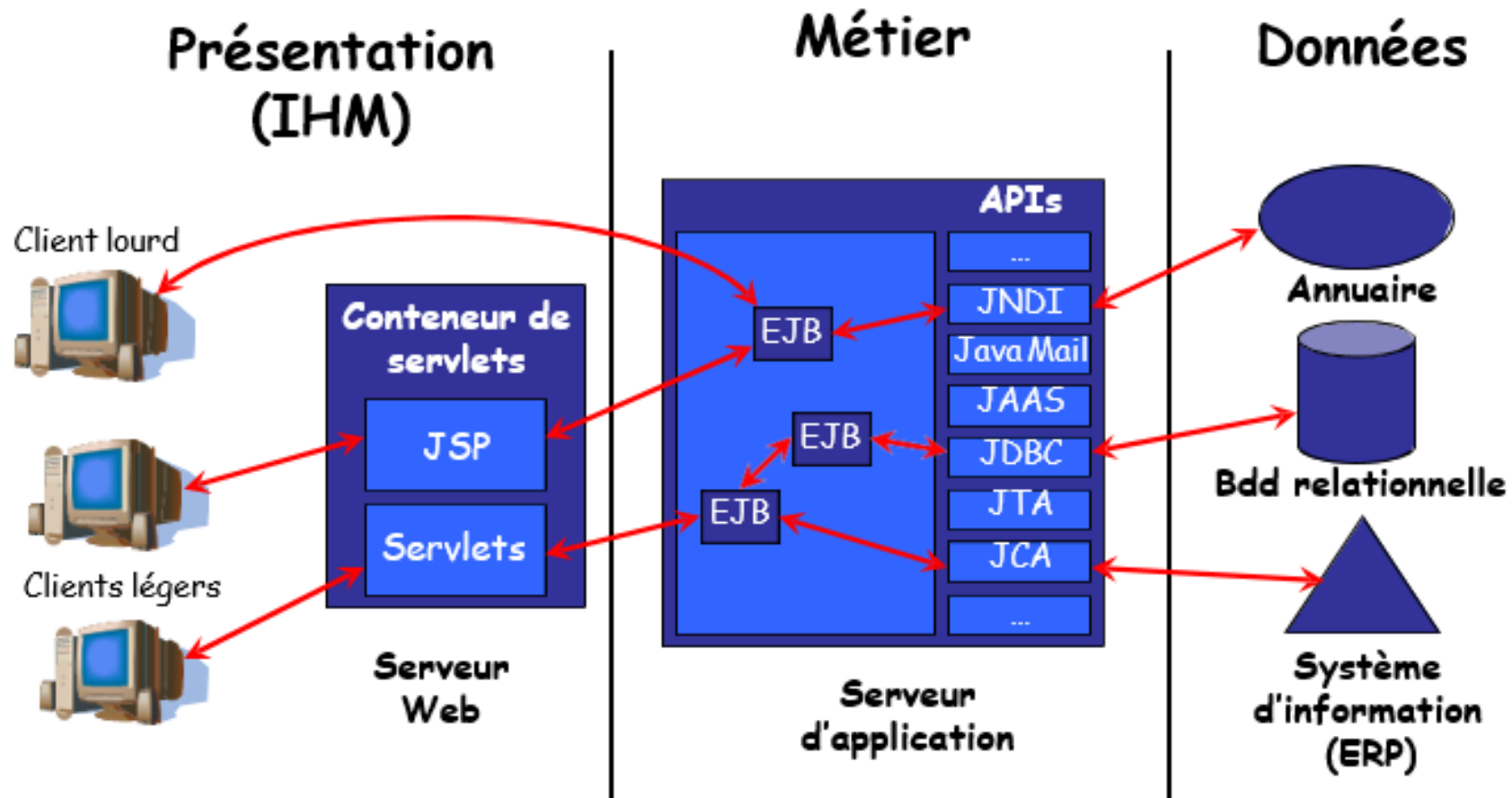
■ Les services d'infrastructures :

- JDBC (Java DataBase Connectivity) est une API d'accès aux bases de données relationnelles.
- JNDI (Java Naming and Directory Interface) est une API d'accès aux services de nommage et aux annuaires d'entreprises tels que DNS, NIS, LDAP, etc.
- JTA/JTS (Java Transaction API/Java Transaction Services) est une API définissant des interfaces standard avec un gestionnaire de transactions.
- JCA (J2EE Connector Architecture) est une API de connexion au système d'information de l'entreprise, notamment aux systèmes dits «Legacy» tels que les ERP.
- JMX (Java Management Extension) fournit des extensions permettant de développer des applications web de supervision d'applications.

LES API DE J2EE

- Les services de communication :
 - JAAS (Java Authentication and Authorization Service) est une API de gestion de l'authentification et des droits d'accès.
 - JavaMail est une API permettant l'envoi de courrier électronique.
 - JMS (Java Message Service) fournit des fonctionnalités de communication asynchrone (appelées MOM pour Middleware Object Message) entre applications.
 - RMI-IIOP (Remote Method Invocation Over Internet Inter-ORB Protocol) est une API permettant la communication synchrone entre objets.

L'ARCHITECTURE J2EE



TYPES DE CLIENTS

- Client lourd (en anglais fat client ou heavy client) : désigne une application cliente graphique exécutée sur le système d'exploitation de l'utilisateur. Un client lourd possède généralement des capacités de traitement évoluées et peut posséder une interface graphique sophistiquée.
- Client léger (en anglais thin client) : désigne une application accessible via une interface web (en HTML) consultable à l'aide d'un navigateur web, où la totalité de la logique métier est traitée du côté du serveur. Pour ces raisons, le navigateur est parfois appelé client universel.
- Client riche est un compromis entre le client léger et le client lourd. L'objectif du client riche est de proposer une interface graphique, décrite avec une grammaire de description basée sur la syntaxe XML, permettant d'obtenir des fonctionnalités similaires à celles d'un client lourd (glisser déposer, onglets, multi fenêtrage, menus déroulants).

SERVEUR D'APPLICATION

- Un serveur d'application est un environnement d'exécution des applications côté serveur.
- Il prend en charge l'ensemble des fonctionnalités qui permettent à N clients d'utiliser une même application
- Exemples :
 - JBoss (www.jboss.org)
 - WebSphere Application Server d'IBM
 - Weblogic de BEA (www.bea.com)
 - ..etc



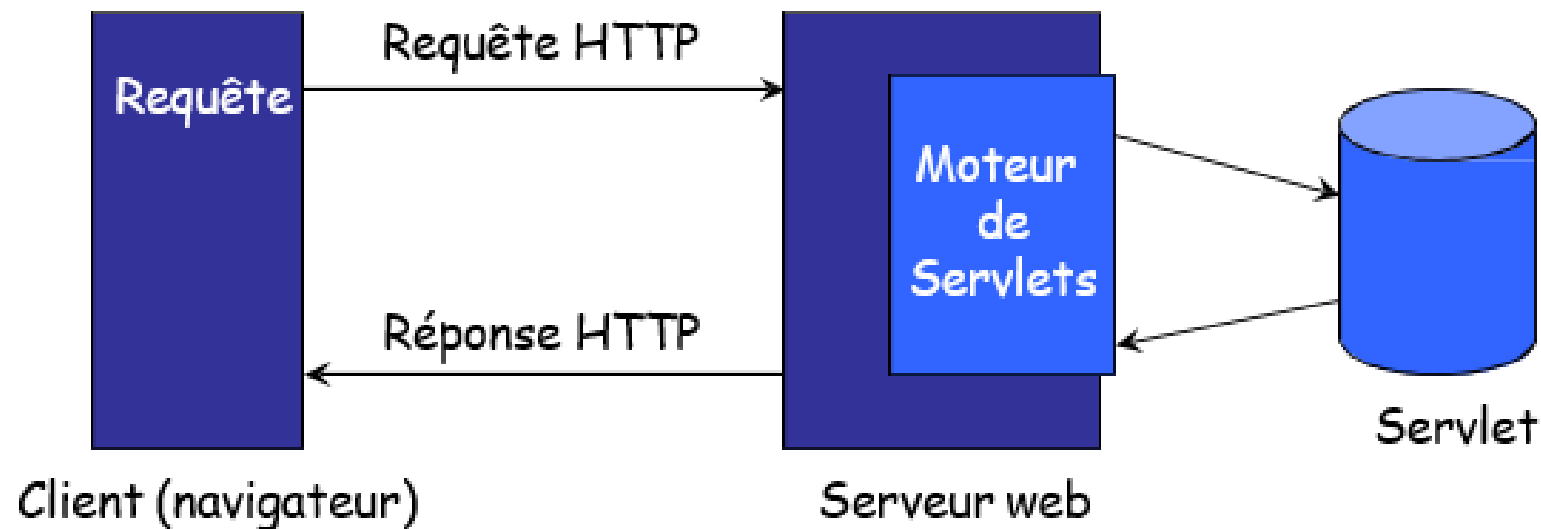
PARTIE 2 : LES SERVLETS |

INTRODUCTION AUX SERVLETS

- Ce sont des applications Java fonctionnant du côté serveur (tels que ASP ou bien PHP).
- Les servlets sont au serveur Web ce que les applets sont au navigateur pour le client.
- Servlet:
 - Reçoit des requêtes HTTP
 - Effectue traitement
 - Fournit une réponse HTTP dynamique au client Web (permet donc de créer des pages web dynamiques).

INTRODUCTION AUX SERVLETS

- les servlets sont indépendantes du serveur web
 - Elles s'exécutent dans un moteur de servlets utilisé pour établir le lien entre la servlet et le serveur Web



AVANTAGES DES SERVLETS

- Les servlets ont de nombreux avantages par rapport aux autres technologies côté serveur:
 - Peut utiliser toutes les API Java afin de communiquer avec des applications extérieures, se connecter à des bases de données, accéder aux entrées-sorties...
 - Sont indépendantes du serveur Web
 - Se chargent automatiquement lors du démarrage du serveur ou bien lors de la connexion du premier client.
 - La résidence en mémoire leur permettent :
 - De traiter les demandes des clients grâce à des threads.
 - D'occuper moins de mémoire et de charge du processeur.
 - À l'opposé, les langages de script traditionnels créent de nouveaux processus pour chaque requête HTTP.
 - permettant de créer des composants réutilisables.

MOTEUR DE SERVLETS

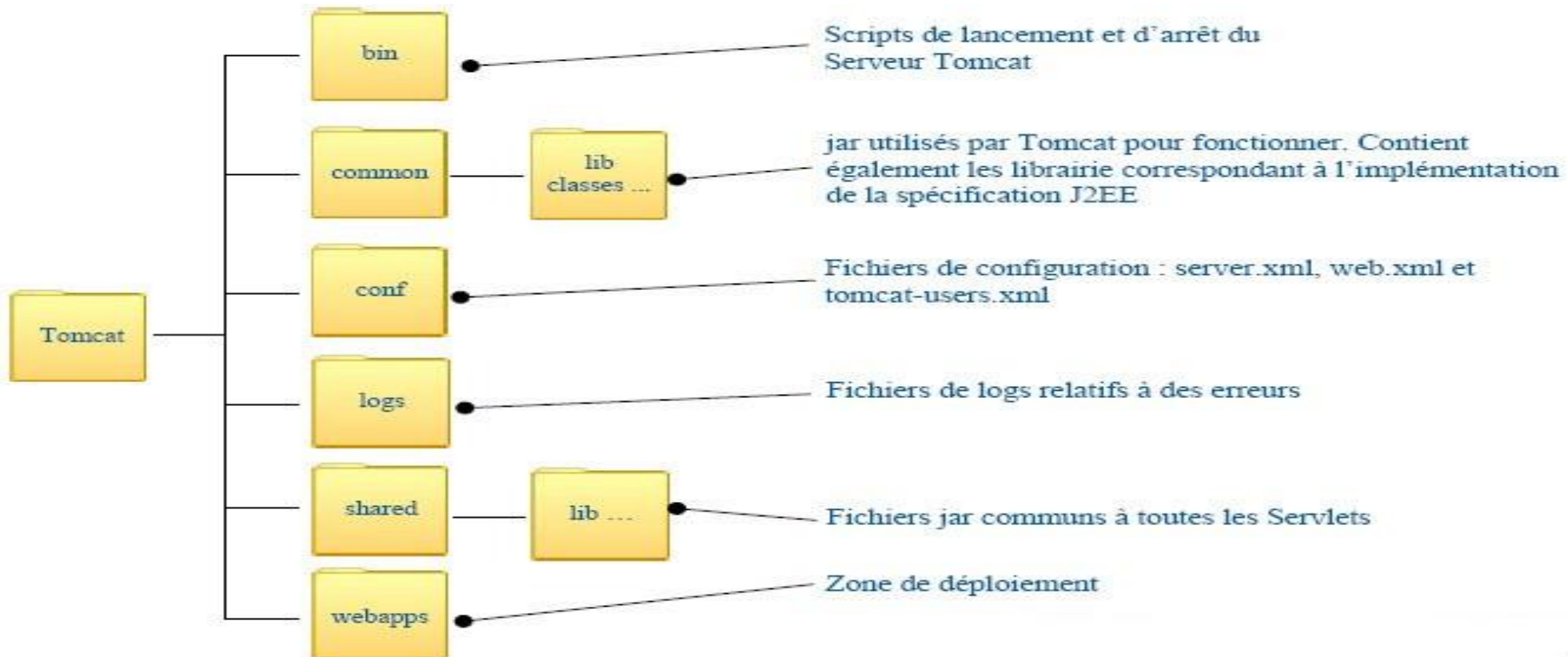
- Un moteur de servlets est connu aussi par conteneur de servlets (en anglais Servlet Container)
- Un moteur de servlets permet d'établir le lien entre la Servlet et le serveur Web
- Il prend en charge et gère les servlets:
 - chargement de la servlet
 - gestion de son cycle de vie
 - passage des requêtes et des réponses
- Deux types de conteneurs
 - Conteneurs de Servlets autonomes : c'est un serveur Web qui intègre le support des Servlets
 - Conteneurs de Servlets additionnels : fonctionnent comme un plug-in à un serveur Web existant

MOTEUR DE SERVLETS

- Nombreux conteneurs de Servlet
 - Jakarta Tomcat
 - JBoss (www.jboss.org)
 - WebSphere Application Server d'IBM
 - Weblogic de BEA (www.bea.com)
- Dans le reste du cours et des TP, nous utiliserons le conteneur Tomcat pour déployer nos servlets.

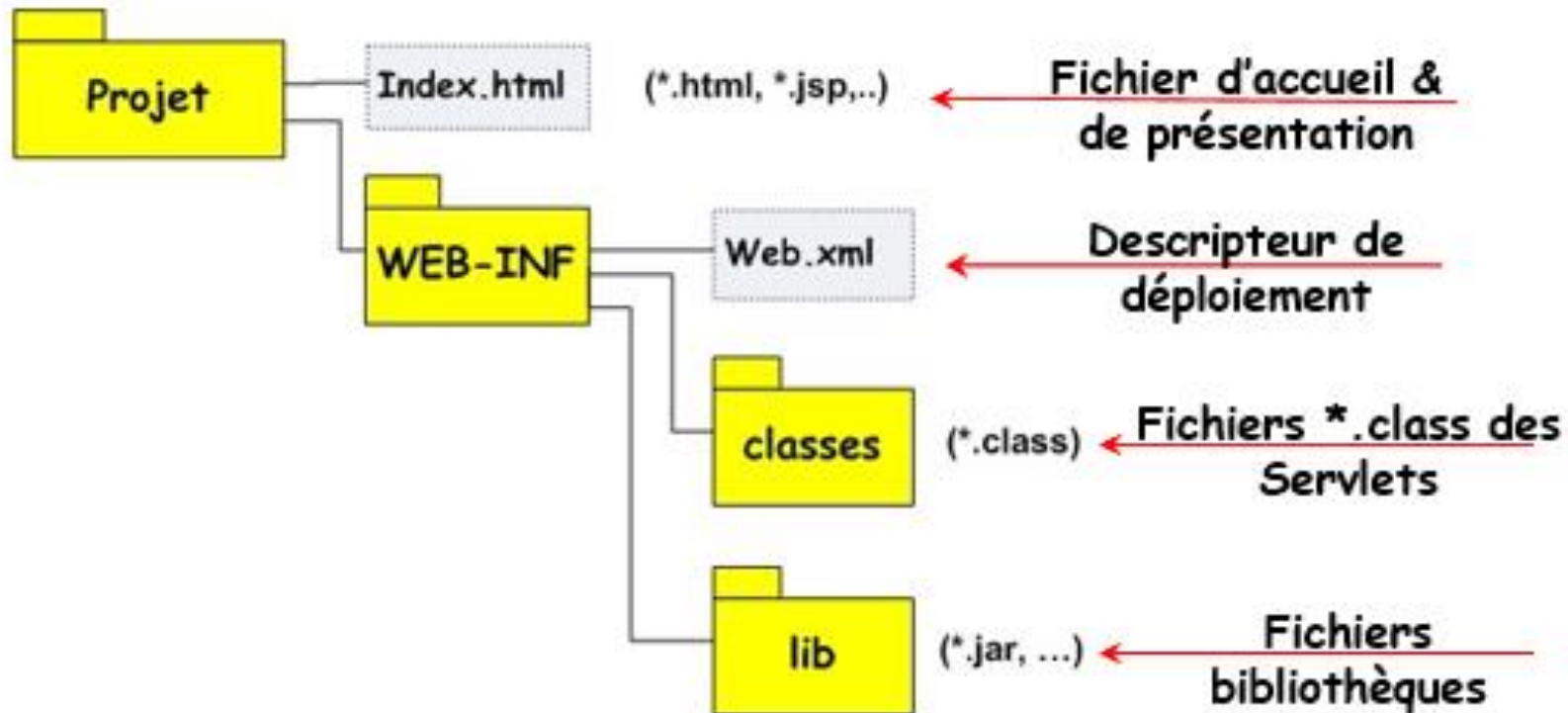
HIÉRARCHIE DES DOSSIERS TOMCAT

■ Organisation partielle des dossiers de Tomcat



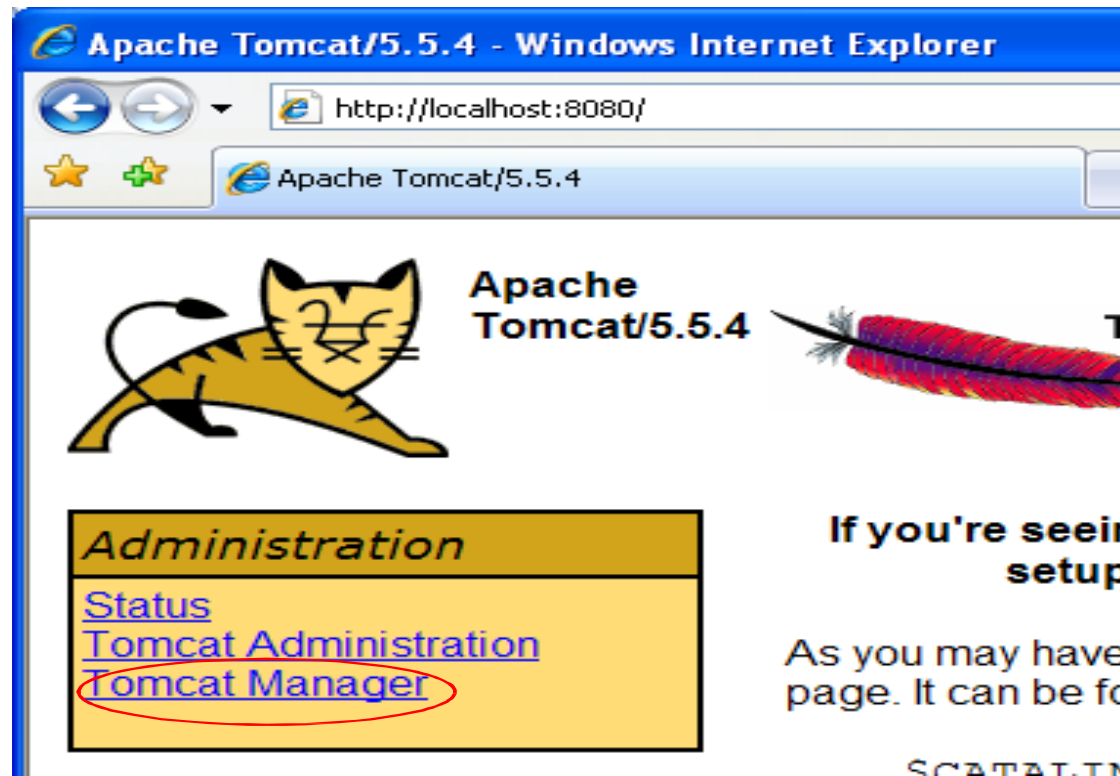
DÉPLOIEMENT D'UNE SERVLET DANS TOMCAT

- Une application Web doit être déployée dans le dossier webapps et avoir la structure suivante:



TOMCAT MANAGER


- Les applications hébergées sur Tomcat peuvent être gérées (démarrées, arrêtées, rechargées) grâce au Tomcat Manager.



TOMCAT MANAGER

- Le lancement de Tomcat Manager demande un login et un mot de passe.
- Par défaut
 - le login est : admin
 - Le mot de passe est vide
- Si vous voulez changer ces valeurs pour pouvez éditer le fichier « tomcat-users.xml » qui se trouve dans le dossier conf de tomcat.

TOMCAT MANAGER


 /manager - Windows Internet Explorer


http://localhost:8080/manager/html

Live Search

/manager

Page Outils

 **The Apache Jakarta Project**
http://jakarta.apache.org/



Gestionnaire d'applications WEB Tomcat

Message: OK

Manager

[List Applications](#) [HTML Manager Help](#) [Manager Help](#) [Etat du serveur](#)

Applications

Chemin	Nom d'affichage	Fonctionnant	Sessions	Commands
/	Welcome to Tomcat	true	0	Démarrer Arrêter Recharger Undeploy
/manager	Tomcat Manager Application	true	0	Démarrer Arrêter Recharger Undeploy
/projet1	Application WEB affichant HelloWorld	true	0	Démarrer Arrêter Recharger Undeploy
/projet2	compteur de visites	true	0	Démarrer Arrêter Recharger Undeploy
/tomcat-docs	Tomcat Documentation	true	0	Démarrer Arrêter Recharger Undeploy

LE FICHIER WEB.XML

- Le fichier « web.xml » est le fichier de configuration de l'application Web qu'on est en cours de construire.
- Il permet de donner des informations de l'application à Tomcat comme :
 - Les noms des classes des Servlets
 - Le nom d'affichage de l'application
 - Le chemin virtuel pour accéder aux différents servlets
 - Les fichiers d'accueils
 - Etc..

LE FICHER WEB.XML :

EXEMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
  <web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd" version="2.4">
```

```
    <display-name>Ma première application Web</display-name>  
    <servlet>
```

```
      <servlet-name>Hello</servlet-name>
```

```
      <servlet-class>HelloWorldServlet</servlet-class>
```

```
    </servlet>
```

```
    <servlet-mapping>
```

```
      <servlet-name>Hello</servlet-name>
```

```
      <url-pattern>/salut</url-pattern>
```

```
    </servlet-mapping>
```

```
  </web-app>
```

LE FICHIER WEB.XML

- Si le dossier contenant notre Servlet se nomme projet , le chemin d'accès à notre servlet sera :
`http://localhost:8080/projet/salut`
- Remarque :
 - Pour compiler une Servlet sous DOS il faudra ajouter la bibliothèque « `servlet-api.jar` » à la variable d'environnement `classpath`.
 - Cette bibliothèque se trouve dans le dossier
- « lib » de Tomcat
 - Elle se compose des packages : « `javax.servlet` » et « `javax.servlet.http` »

EXEMPLE DE SERVLET

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import java.io.*;
```

```
public class HelloWorldServlet extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws  
        ServletException, IOException {  
        res.setContentType("text/html");  
        PrintWriter out = res.getWriter();  
        out.println("<HTML>");  
  
        out.println("<HEAD><TITLE> Titre </TITLE></HEAD>"); out.println("<BODY>");  
  
        out.println(" Hello World"); out.println("</BODY>");  
        out.println("</HTML>"); out.close();  
  
    }  
}
```

ANALYSE DE L'EXEMPLE

- La première étape consiste à importer les packages nécessaires à la création de la servlet
- il faut donc importer `javax.servlet`, `javax.servlet.http` et `java.io`
 - `import javax.servlet.*;`
 - `import javax.servlet.http.*;`
 - `import java.io.*`

ANALYSE D'UNE SERVLET

- Si le type de la requête est GET, alors la méthode de la Servlet qui traite la requête est la suivante :
 - `public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException { //traitement }`
- Si le type de la requête est POST alors la méthode doit être
 - `public void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException { //traitement }`

ANALYSE D'UNE SERVLET

■ Remarque :

- Dans le cas où nous connaissions pas la méthode d'envoi de la requête (GET ou POST) une astuce serait d'écrire les deux méthode comme suit :

```
public void doGet(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException {  
    //traitement  
}
```

```
public void doPost(HttpServletRequest req, HttpServletResponse res) throws IOException,  
ServletException {  
    doGet(req, res);  
}
```

ANALYSE D'UNE SERVLET

- La méthode doGet (resp. doPost) prend deux paramètres :
 - Un paramètre de type HttpServletRequest représentant la requête client
 - Un paramètre de type HttpServletResponse représentant la réponse à renvoyer au client
- Remarque :
 - L'objet HttpServletRequest permet d'extraire toutes les informations sur le client (adresse IP, navigateur, Domaine, paramètres d'un formulaire, etc..)

ANALYSE D'UNE SERVLET

- L'objet `HttpServletResponse` doit être complété d'informations par la servlet avant de le renvoyer au client.
- La première étape consiste à définir le type de données qui vont être envoyées au client (généralement il s'agit d'une page HTML).
- La méthode `setContentType()` de l'objet `HttpServletResponse` prend donc comme paramètre le type MIME associé au format HTML (`text/html`) :
 - `res.setContentType("text/html");`

ANALYSE D'UNE SERVLET

- La création d'un objet `PrintWriter` grâce à la méthode `getWriter()` de l'objet `HttpServletResponse` permet d'envoyer du texte formaté au navigateur
- `PrintWriter out = res.getWriter();`
- La méthode `println()` de l'objet `PrintWriter` permet d'envoyer les données textuelles au navigateur
 - `out.println("<HTML>");`
 - ...
 - `out.println("</HTML>");`
- L'objet `PrintWriter` doit être fermé lorsqu'il n'est plus utile avec sa méthode `close()`
 - `out.close();`

NOTION DE CONTEXTE

- Un contexte constitue pour chaque Servlet d'une même application une vue sur le fonctionnement de cette application.
- Une application web peut être composée de :
 - Servlets
 - JSP
 - Classes utilitaires
 - Documents statiques (pages html, images, sons, etc.)
 - Etc..
- Grâce à ce contexte, il est possible d'accéder à chacune des ressources de l'application web correspondant au contexte.
- Dans le code source d'une servlet, un contexte est représenté par un objet de type ServletContext.
- Exemple :
 - `getServletContext().getServerInfo()` retourne le nom du logiciel utilisé pour prendre en charge la requête , par exemple Tomcat/3.2.1.

NOTION DE CONTEXTE

- Chaque contexte est propre à une application et qu'il n'est pas possible de partager des ressources entre applications différentes.
- A chaque contexte correspond une arborescence dans le système de fichiers qui contient les ressources accédées lors des requêtes vers le moteur de servlets comme nous avons vu pour Tomcat.
- Le fichier web.xml est donc un descripteur de déploiement du contexte. Il peut contenir entre autres:
 - Les paramètres d'initialisation du contexte.
 - Les définitions des servlets et des JSPs.
 - La liste des fichiers de bienvenue.
 - Les pages d'erreur.
 - Etc..

NOTION DE CONTEXTE

- Exemple de fichiers web.xml:

```
<web-app>
<servlet>
  <servlet-name>maServletToto</servlet-name>
  <servlet-class>Toto</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>maServletToto</servlet-name>
  <url-pattern>/maServlet</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>accueil.html</welcome-file>
</welcome-file-list>
<error-page>
  <error-code>404</error-code>
  <location>/404.html</location>
</error-page>
</web-app>
```

L'API SERVLET

- L'API Servlet fournit un ensemble de classes et d'interfaces pour la manipulation des servlets
- Cet API est fourni sous forme d'un kit appelé JSDK (Java Servlet Development Kit)
- L'API servlet regroupe un ensemble de classes dans deux packages :
 - `javax.servlet` : contient les classes pour développer des servlets génériques indépendantes d'un protocole
 - `javax.servlet.http` : contient les classes pour développer des servlets qui reposent sur le protocole http utilisé par les serveurs web.

LE PACKAGE JAVAX.SERVLET

- Le package javax.servlet définit plusieurs interfaces, méthodes et exceptions :

- Les interfaces :

- [RequestDispatcher](#) : définit un objet qui reçoit les requêtes du client et les envoie à n'importe quelle ressource (par exemple servlet, fichiers HTML ou JSP) sur le serveur.
- [Servlet](#) : interface de base d'une servlet
- [ServletConfig](#) : Définit d'un objet utilisé par le conteneur de la servlet pour passer de l'information à une servlet pendant son initialisation.
- [ServletContext](#) : Définit un ensemble de méthodes qu'une servlet utilise pour communiquer avec le conteneur de servlets
- [ServletRequest](#) : Définit un objet contenant la requête du client.
- [ServletResponse](#) : Définit un objet qui contient la réponse renvoyée par la servlet
- [SingleThreadModel](#) : Permet de définir une servlet qui ne répondra qu'à une seule requête à la fois

LE PACKAGE JAVAX.SERVLET

■ Les classes :

- **GenericServlet** : Classe définissant une servlet indépendante de tout protocoles
- **ServletInputStream** : permet la lecture des données de la requête cliente
- **ServletOutputStream** : permet l'envoi de la réponse de la servlet

■ Les exceptions :

- **ServletException** : Exception générale en cas de problème durant l'exécution de la servlet
- **UnavailableException** : Exception levée si la servlet n'est pas disponible

LE PACKAGE JAVAX.SERVLET.HTTP

- Le package `javax.servlet.http` définit plusieurs interfaces et méthodes :

- **Les interfaces :**

- `HttpServletRequest` : Hérite de `ServletRequest` : définit un objet contenant une requête selon le protocole http
- `HttpServletResponse` : Hérite de `ServletResponse` : définit un objet contenant la réponse de la servlet selon le protocole http
- `HttpSession` : Définit un objet qui représente une session

- **Les classes :**

- `Cookie` : Classe représentant un cookie (ensemble de données sauvegardées par le navigateur sur le poste client)
- `HttpServlet` : Hérite de `GenericServlet` : classe définissant une servlet utilisant le protocole http
- `HttpUtils` : Classe proposant des méthodes statiques utiles pour le développement de servlet http (classe devenue obsolète)

L'INTERFACE D'UNE SERVLET

- Pour pouvoir être gérée par le conteneur Web, toute
- servlet doit implémenter l'interface Servlet
- appartenant au package javax.servlet
- Cette interface permet ainsi au conteneur Web de gérer le cycle de vie de la servlet.
- L'interface d'une servlet se compose des méthodes suivantes :
 - la méthode `init()`
 - la méthode `service()`
 - la méthode `getServletConfig()`
 - la méthode `getServletInfo()`
 - la méthode `destroy()`

servlet
<i>init()</i> <i>service()</i> <i>getServletConfig()</i> <i>getServletInfo()</i> <i>destroy()</i>

L'INTERFACE D'UNE SERVLET

- Afin de mettre en place l'interface Servlet nécessaire au conteneur de servlet, il existe plusieurs possibilités :
 1. Définir manuellement chaque méthode
 2. Dériver la classe GenericServlet et redéfinir les méthodes dont on a besoin
 3. Dériver la classe HttpServlet et redéfinir les méthodes dont on a besoin
- C'est la 3eme solution que nous utilisons presque tout le temps

LA MÉTHODE `init()`

- Signature :
 - `public void init(ServletConfig config) throws ServletException`
- Est appelée par le conteneur à chaque instantiation de la servlet
- Lors de l'instanciation, le conteneur de servlet passe en argument à la méthode `init()` un objet `ServletConfig` permettant de charger des paramètres de configuration propres à la servlet.
- En cas d'anomalie lors de l'appel de la méthode `init()`, celle-ci renvoie une exception de type `ServletException` et la servlet n'est pas initialisée.

LA MÉTHODE init()

■ Exemple :

- Écrire une servlet qui compte le nombre d'utilisation d'une servlet depuis son chargement.

■ Solution :

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

```
public class Compteur1 extends HttpServlet { private int compteur;  
  
    public void init() throws ServletException { compteur = 0;  
}
```

LA MÉTHODE init()

```
protected void doGet(HttpServletRequest req, HttpServletResponse  
res) throws ServletException, IOException
```

```
{
```

```
    res.setContentType("text/plain");    PrintWriter out =
```

```
    res.getWriter();    compteur++;
```

```
    out.println("Depuis son chargement, on a accédé à cette Servlet "  
+compteur+" fois.");
```

```
}
```

```
}
```

LA MÉTHODE init()

- Il est possible de faire des initialisations au niveau du fichier web.xml et de les utiliser dans la méthode init().
- Exemple :

```
<servlet>
  <servlet-name>Cmp</servlet-name>
  <servlet-class>Compteur2</servlet-class>
  <init-param>
    <param-name>compteur_initial</param-name>
    <param-value>50</param-value>
    <description>Valeur init du compteur</description>
  </init-param>
</servlet>
```

LA MÉTHODE `init()`

```
public class Compteur2 extends HttpServlet { private int compteur;
```

```
public void init() throws ServletException {
```

```
    String initial = this.getInitParameter("compteur_initial"); try {
```

```
        compteur = Integer.parseInt(initial);
```

```
    }
```

```
    catch(NumberFormatException e) { compteur= 0;
```

```
    }
```

```
}
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException {
```

```
    //même traitement que l'exemple dernier...
```

```
}}
```

LA MÉTHODE `service()`

- Signature :

- `public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException`

- Cette méthode est exécutée par le conteneur lorsque la servlet est sollicitée.

- Elle détermine le type de requête dont il s'agit, puis transmet la requête et la réponse à la méthode adéquate (`doGet()` ou `doPost()`).

- Chaque requête du client déclenche une seule exécution de cette méthode.

LA MÉTHODE `getServletConfig()`

- Signature :
 - `public ServletConfig getServletConfig()`
- Renvoie un objet `ServletConfig` qui constitue un intermédiaire permettant d'accéder au contexte d'une application.
- On peut aussi utiliser `ServletConfig` pour récupérer un paramètre du fichier `web.xml` :
- Exemple :
- String param;

```
public void init(ServletConfig config)
{
    param = config.getInitParameter("param");
}
```

LA MÉTHODE `getServletInfo()`

- Signature:

- `public String getServletInfo()`

- Lorsqu'elle est surchargée permet de retourner des informations sur la servlet comme l'auteur, la version, et le copyright.

- Ces informations peuvent être exploitées pour affichage par des outils dans les conteneurs Web.

- Exemple :

```
public String getServletInfo() {  
    return " servlet écrite par x (x@y.com)" ;  
}
```

LA MÉTHODE `destroy()`

- Signature :
 - `void destroy()`
- La méthode `destroy()` est appelée par le conteneur lors de l'arrêt du serveur Web.
- Elle permet de libérer proprement certaines ressources (fichiers, bases de données ...).
- C'est le serveur qui appelle cette méthode.

LE CYCLE DE VIE D'UNE SERVLET

- Le cycle de vie d'une servlet est assuré par le conteneur de servlet (grâce à l'interface Servlet).
- Cette interface permet à la servlet de suivre le cycle de vie suivant :
 1. le serveur crée un **pool de threads** auxquels il va pouvoir affecter chaque requête
 2. La servlet est chargée au démarrage du serveur ou lors de la première requête
 3. La servlet est instanciée par le serveur
 4. La méthode `init()` est invoquée par le conteneur
 5. Lors de la première requête, le conteneur crée les objets `Request` et `Response` spécifiques à la requête

LE CYCLE DE VIE D'UNE SERVLET

6. La méthode `service()` est appelée à chaque requête dans une nouvelle thread. Les objets `Request` et `Response` lui sont passés en paramètre
7. Grâce à l'objet `Request`, la méthode `service()` va pouvoir analyser les informations en provenance du client
8. Grâce à l'objet `Response`, la méthode `service()` va fournir une réponse au client
9. La méthode `destroy()` est appelée lors du déchargement de la servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La servlet est alors signalée au garbage collector.

DÉVELOPPER UNE SERVLET HTTP

- Les étapes de développement d'une servlet sont les suivantes:
 1. Lecture de la requête (représentée par l'objet *HttpServletRequest*)
 2. Traitement
 3. Création de la réponse (représentée par l'objet *HttpServletResponse*)

DÉVELOPPER UNE SERVLET HTTP

Lecture d'une requête

- L'objet *HttpServletRequest* fournit un ensemble de méthodes pour avoir toutes les informations concernant une requête.
- Ces méthodes sont comme suit :
 - **String `getMethod()`** : Récupère la méthode HTTP utilisée par le client
 - **String `getHeader(String name)`** : Récupère la valeur de l'en-tête demandée
 - **String `getRemoteHost()`** : Récupère le nom de domaine du client
 - **String `getRemoteAddr()`** : Récupère l'adresse IP du client

DÉVELOPPER UNE SERVLET HTTP

- **String `getParameter(String name)`** : Récupère la valeur du paramètre `name` d'un formulaire. Lorsque plusieurs valeurs sont présentes, la première est retournée
- **String[] `getParameterValues(String name)`** : Récupère les valeurs correspondant au paramètre `name` d'un formulaire, c'est-à-dire dans le cas d'une sélection multiple (cases à cocher, listes à choix multiples) les valeurs de toutes les entités sélectionnées
- **Enumeration `getParameterNames()`** : Retourne un objet *Enumeration* contenant la liste des noms des paramètres passés à la requête
- **String `getServerName()`** : Récupère le nom du serveur
- **String `getServerPort()`** : Récupère le numéro de port du serveur

DÉVELOPPER UNE SERVLET HTTP

Création de la réponse

- Après lecture et traitement d'une requête, la réponse à fournir à l'utilisateur est représentée sous forme d'objet *HttpServletResponse*.
- Les méthodes de l'objet *HttpServletResponse* sont comme suit :
 - **void setHeader(String Nom, String Valeur)** : Définit une paire clé/valeur dans les en-têtes
 - **void setContentType(String type)** : Définit le type MIME de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur

DÉVELOPPER UNE SERVLET HTTP

- **void setContentLength(int len)** : Définit la taille de la réponse
- **PrintWriter getWriter()** : Retourne un objet *PrintWriter* permettant d'envoyer du texte au navigateur client. Il se charge de convertir au format approprié les caractères Unicode utilisés par Java
- **ServletOutputStream getOutputStream()** : Définit un flot de données à envoyer au client, par l'intermédiaire d'un objet *ServletOutputStream*, dérivé de la classe *java.io.OutputStream*
- **void sendredirect(String location)** : Permet de rediriger le client vers l'URL *location*

DÉVELOPPER UNE SERVLET HTTP

- **String setStatus(int StatusCode)** : Définit le code de retour de la réponse
- Rappelons quelques codes de retour:
 - Code 202 (**SC_ACCEPTED**) : Requête acceptée.
 - Code 204 (**SC_NO_CONTENT**) : pas d'information à retourner.
 - Code 301 (**SC_MOVED_PERMANENTLY**) : la ressource demandée a été déplacée.
 - Code 400 (**SC_BAD_REQUEST**) : La requête est syntaxiquement incorrecte.
 - Code 403 (**SC_FORBIDDEN**) : le serveur comprend la requête mais refuse de la servir.
 - Code 404 (**SC_NOT_FOUND**) : la ressource demandée n'est pas disponible.
 - etc...

ENVOYER UN CONTENU MULTIMÉDIA

- Pour l'instant nous avons écrit des Servlets qui retournaient des contenus HTML
- Besoin de retourner des contenus différents :
 - Protocole WAP et langage WML utilisés par les téléphones portables
 - Génération de contenus multimédias (création de graphes, manipulation d'images)
- L'API Java facilite la gestion des contenus multimédias en proposant des bibliothèques
 - Encodage d'images sous différents formats (GIF, JPEG)
 - Manipulation et traitement d'images

ENVOYER UN CONTENU MULTIMÉDIA

- Exemple : Servlet qui génère et retourne une image JPEG affichant le message « Bonjour monde ! »

```
import com.sun.image.codec.jpeg.*; import javax.servlet.http.*;
```

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

```
public class ImageServlet extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws  
        ServletException, IOException {  
        // le contenu produit est une image au format jpeg  
        res.setContentType("image/jpeg");  
    }  
}
```

ENVOYER UN CONTENU MULTIMÉDIA

// l'objet enc va encoder les données et les envoyer sur

// le flux de sortie de type ServletOutputStream

```
JPEGImageEncoder enc = JPEGCodec.createJPEGEncoder(out);
```

// création d'une nouvelle image d'une résolution de 1024 par 768

```
BufferedImage image = new BufferedImage(1024,768,BufferedImage.TYPE_BYTE_INDEXED);
```

// récupération du contexte graphique lié à l'image

```
Graphics2D g = image.createGraphics();
```

// la prochaine opération s'effectuera avec la couleur rouge

```
g.setColor(Color.red);
```

// affichage de la célèbre phrase

```
g.drawString("Bonjour monde !", 400, 500);
```

// transformation des données au format jpeg et envoi

// de celles-ci sur le flux standard de sortie (le navigateur)

```
enc.encode(image); }
```

SUIVI DE SESSION

- Le protocole HTTP est un protocole sans état
- Impossibilité alors de garder des informations d'une requête à l'autre (identifier un client d'un autre)
- Obligation d'utiliser différentes solutions pour remédier au problème d'état dont :
 - L'utilisation de cookies
 - L'utilisation de sessions

SUIVI DE SESSION : COOKIES

- Les cookies représentent un moyen simple de stocker temporairement des informations chez un client, afin de les récupérer ultérieurement.
- Les cookies ont été introduits par la première fois dans Netscape Navigator
- Les cookies font partie des spécifications du protocole HTTP.
- L'en-tête HTTP réservé à l'utilisation des cookies s'appelle Set- Cookie, il s'agit d'une simple ligne de texte de la forme:
 - Set-Cookie : NOM=VALEUR; domain=NOM_DE_DOMAINE; expires=DATE
- La valeur d'un cookie pouvant identifier de façon unique un client, ils sont souvent utilisés pour le suivi de session

SUIVI DE SESSION : COOKIES

- L'API Servlet fournit la classe *javax.servlet.http.Cookie* pour travailler avec les Cookies
 - *Cookie(String name, String value)* : construit un cookie
 - *String getName()* : retourne le nom du cookie
 - *String getValue()* : retourne la valeur du cookie
 - *setValue(String new_value)* : donne une nouvelle valeur au cookie
 - *setMaxAge(int expiry)* : spécifie l'âge maximum du cookie en secondes
- Pour la création d'un nouveau cookie, il faut l'ajouter à la réponse (*HttpServletResponse*)
 - *addCookie(Cookie mon_cook)* : ajoute à la réponse un cookie
- La Servlet récupère les cookies du client en exploitant la requête (*HttpServletRequest*)
 - *Cookie[] getCookies()* : récupère l'ensemble des cookies du site

SUIVI DE SESSION : COOKIES

- Code pour créer un cookie et l'ajouter au client : `Cookie cookie = new Cookie("Id", "123"); res.addCookie(cookie);`
- Code pour récupérer les cookies
`Cookie[] cookies = req.getCookies(); if (cookies != null) {`
`for (int i = 0; i < cookies.length; i++) { String name = cookies[i].getName(); String value =`
`cookies[i].getValue();`
`...`
`}`
`}`
- Remarque :
 - Il n'existe pas dans l'API Servlet de méthode permettant de récupérer la valeur d'un cookie par son nom

SUIVI DE SESSION : HTTPSESSION

- Le plus gros problème des cookies est que les navigateurs ne les acceptent pas toujours
- L'utilisateur peut configurer son navigateur pour qu'il refuse ou pas les cookies
- Les navigateurs n'acceptent que 20 cookies par site, 300 par utilisateur et la taille d'un cookie peut être limitée à 4096 octets (4 ko)

SUIVI DE SESSION : HTTPSESSION

- Solutions : utilisation de l'API de suivi de session

javax.servlet.http.HttpSession

- Méthodes de création liées à la requête (*HttpServletRequest*)
 - *HttpSession getSession()* : retourne la session associée à l'utilisateur
 - *HttpSession getSession(boolean p)* : création selon la valeur de *p*
- Gestion d'association (*HttpSession*)
 - *Enumeration getAttributeNames()* : retourne les noms de tous les attributs
 - *Object getAttribute(String name)* : retourne l'attribut name
 - *setAttribute(String an, Object av)* : associe l'objet av à la chaîne an
 - *removeAttribute(String na)* : supprime l'attribut associé à na
- Destruction (*HttpSession*)
 - *invalidate()* : expire la session

SUIVI DE SESSION : HTTPSESSION

- Exemple : Servlet qui permet d'utiliser la suivi de session pour un compteur

```
public class HttpSessionServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,  
        IOException {
```

```
        res.setContentType("text/plain"); PrintWriter out = res.getWriter(); HttpSession  
        session = req.getSession();
```

```
        Integer count = (Integer)session.getAttribute("compteur");
```

```
        if (count == null)
```

```
            count = new Integer(1); else
```

```
            count = new Integer(count.intValue() + 1); session.setAttribute(" compteur ", count);  
        out.println("Vous avez visité cette page " + count + " fois.");
```

```
    }}
```

COLLABORATION DE SERVLETS

- Les Servlets s'exécutant dans le **même** serveur peuvent dialoguer entre elles
- Deux principaux styles de collaboration :
 - **Partage d'information** : un état ou une ressource.
Exemple : un magasin en ligne pourrait partager les informations sur le stock des produits ou une connexion à une base de données
 - **Partage du contrôle** : une requête.
Réception d'une requête par une Servlet et laisser l'autre Servlet une partie ou toute la responsabilité du traitement

COLLABORATION DE SERVLETS : PARTAGE D'INFORMATION

- La collaboration est obtenue par l'interface

ServletContext

- L'utilisation de *ServletContext* permet aux applications web de disposer de son propre conteneur d'informations unique
- Une Servlet retrouve le *ServletContext* de son application Web par un appel à *getServletContext()*
- Exemples de méthodes :
 - *void setAttribute(String nom, Object o)* : lie un objet sous le nom indiqué
 - *Object getAttribute(String nom)* : retrouve l'objet sous le nom indiqué
 - *Enumeration getAttributeNames()* : retourne l'ensemble des noms de tous les attributs liés
 - *void removeAttribute(String nom)* : supprime l'objet lié sous le nom indiqué

PARTAGE D'INFORMATION

Exemple : Servlets
qui vendent des
pizzas et partagent
une spécialité du
jour

```
public class PizzasAdmin extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        ServletContext context = this.getServletContext();  
        context.setAttribute("Specialite", "Anchois");  
        context.setAttribute("Date", new Date());  
        out.println("La pizza du jour a été définie.");  
    }  
}
```

Création de deux attributs

```
public class PizzasClient extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {  
        ...  
        ServletContext context = this.getServletContext();  
        String pizza_spec = (String)context.getAttribute("Specialite");  
        Date day = (Date)context.getAttribute("Date");  
        DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);  
        String today = df.format(day);  
        out.println("Aujourd'hui (" + today + "), notre specialite est : " + pizza_spec);  
    }  
}
```

Lecture des attributs

COLLABORATION DE SERVLETS : PARTAGE DU CONTRÔLE

- Pour une collaboration dynamique entre servlets, deux possibilités existent:
 - Déléguer entièrement la requête à une autre servlet : méthode **forward()**
 - Inclure la réponse d'une autre servlet dans la servlet en cours : méthode **include()**
- Ces deux méthodes appartiennent à l'interface `RequestDispatcher` du package `javax.servlet`
 - *`RequestDispatcher getRequestDispatcher(String path)`* : retourne une instance de type *`RequestDispatcher`* par rapport à un composant
 - Un composant peut-être de tout type : Servlet, JSP, fichier statique, ...
 - *`path`* est un chemin relatif ou absolu ne pouvant pas sortir du contexte

PARTAGE DU CONTRÔLE (FORWARD)

■ Soit l'exemple suivant :

- Une servlet (Servlet1) reçoit une requête
- Elle y place un attribut *attr1* qu'elle y met la chaîne "salut"
- Elle renvoie ensuite cette requête à une autre Servlet (Servlet2).
- Servlet2 récupère cet attribut et se charge de créer la réponse qu'elle renvoie à l'utilisateur.

■ Attention:

- Servlet1 ne doit pas toucher à la réponse car c'est Servlet2 qui s'en charge.
- Après le renvoi de la requête à Servlet2, Servlet1 ne doit plus toucher à la requête.

PARTAGE DU CONTRÔLE (FORWARD)

Code pour servlet1

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

public class Servlet1 extends HttpServlet {

    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
    {

        req.setAttribute("attr1", "salut");

        RequestDispatcher dispat = req.getRequestDispatcher("/maServlet2Path"); dispat.forward(req,res);

    }

}
```

PARTAGE DU CONTRÔLE (FORWARD)

Code pour servlet2

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

```
public class Servlet2 extends HttpServlet {
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException {
```

```
res.setContentType("text/plain"); PrintWriter out = res.getWriter();
```

```
out.println("l'attribut que j'ai récupéré de servlet 1 est: "+req.getAttribute(" attr1 "));
```

PARTAGE DU CONTRÔLE (INCLUDE)

■ Soit l'exemple suivant :

- Une servlet (Servlet1) reçoit une requête
- Elle y place un attribut *attr1* qu'elle y met la chaîne "bonjour"
- Elle inclut une autre servlet dans le traitement (Servlet2)
- Servlet2 récupère cet attribut et l'affiche
- Servlet1 reprend le contrôle, elle modifie *attr1* en "bonsoir" et l'affiche
- Servlet2 récupère encore une fois cet attribut et l'affiche
- Servlet1 reprend le contrôle

■ Remarque:

- Servlet2 ne doit pas fermer la réponse par `</body>` car c'est Servlet1 qui s'en charge.
- C'est Servlet1 qui se charge de préciser le type mime de la réponse.

PARTAGE DU CONTRÔLE (INCLUDE)

Code pour servlet1

```
public class Servlet1 extends HttpServlet {  
  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,  
        IOException {  
  
        res.setContentType("text/html"); PrintWriter out = res.getWriter(); out.println("<HTML><BODY>");  
        out.println("<br>");  
  
        out.println("je suis servlet1, je place un attribut attr1=bonjour dans la requete"); out.println("<br>inclusion  
: "); req.setAttribute("attr1", "bonjour"); RequestDispatcher dispat =  
        req.getRequestDispatcher("/maServlet2Path"); dispat.include(req,res); out.println("<br>");  
  
        out.println("je suis la servlet 1, je modifie l'attribut attr1=bonsoir dans la requete"); req.setAttribute("attr1",  
        "bonsoir"); out.println("<br>inclusion : "); dispat.include(req,res); out.println("<br>");  
        out.println("</BODY></HTML>");  
    }  
}
```

PARTAGE DU CONTRÔLE (INCLUDE)

Code pour Servlet2

```
public class Servlet2 extends HttpServlet {  
  
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws  
ServletException, IOException {  
  
        PrintWriter out = res.getWriter();  
        out.println("je suis servlet2 attr1 vaut "+req.getAttribute("attr1"));  
    }  
}
```