



SERVLET/JSP

Applications Web MVC

SOMMAIRE

- **PARTIE 1 : Rappel sur le contexte des applications Web d'entreprise**
- **PARTIE 2 : Développement Web en Java**
- **PARTIE 3 : Applications Web et servlets**
- **PARTIE 4 : Présentation des Java Server Pages**
- **PARTIE 5 : Les librairies de balises**
- **PARTIE 6 : Accès aux bases de données**
- **PARTIE 7 : Introduction à Struts**

PARTIE 1 : Plan

- Serveur et client Web.
- Protocoles applicatifs (HTTP).
- Accès aux ressources de l'entreprise.
- HTML-XML, applets Java.
- La plate-forme JEE.
- Architecture multi-tiers.

Terminologie: Serveur

- **Serveur:** un ordinateur disposant d'un certain nombre de ressources qu'il met à disposition d'autres ordinateurs (clients) via le réseau.
- **Types de serveurs:**
 - serveur web
 - serveur d'application
 - ...

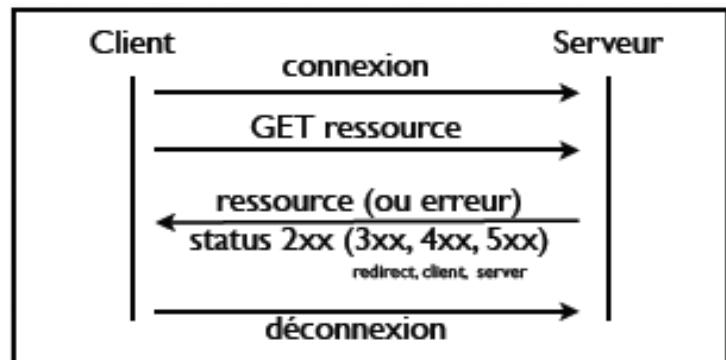
Le protocole HTTP

HTTP: HyperText Transfert Protocol

- RFC 1945, RFC 2616
- protocole de rapatriement de ressources et de soumission de formulaires

Principe (HTTP/1.0)

- connexion,
- demande de ressource (GET),
- renvoi de la ressource ou d'une erreur
- Déconnexion



Le dialogue peut être plus complexe:

- authentification, optimisations...

Format de requêtes

Du client au serveur:

Méthode <URI> HTTP/<version>

[<champs>: <valeur>]

[<tab> suite de la valeur si > 1024]

ligne blanche

[corps de la requête (POST)]

- Exemple de requête GET

GET /index.html HTTP/1.0

Accept: www/source

Accept: text/html

Host: www.w3.org

User-Agent: Mozilla/5.0 (X11; U; Linux i686; fr-FR) Firefox/2.0

- Exemple de requête POST

POST /form HTTP/1.0

Accept: www/source

Accept: text/html

User-Agent: Mozilla/5.0 (X11; U; Linux i686; fr-FR) Firefox/2.0

Content-Length: 24

name1=value1 &

name2=value2

Format de réponse

Du serveur vers le client

HTTP/<Version> <Status> <Commentaire Status>

Content-Type: <Type MIME du contenu>

[<Champ>: <Valeur>]

[<tab> suite de la valeur si >1024]

Ligne vide

Données

- Exemple de réponse

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 2345

Date: Sat, 23 Feb 2008 15:36:29 GMT

Server:Apache/2.2.4 (Ubuntu)

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head> ...

Status des réponses

Code de réponse donné par le serveur au client

- **100-199 Informationnel**
 - 100 : Continue (le client peut envoyer la suite de la requête), ...
- **200-299 Succès de la requête client**
 - 200: OK, 201: Created, 204 : No Content, ...
- **300-399 Redirection de la Requête client**
 - 301: Redirection, 302: Found, 304: Not Modified, 305 : Use Proxy, ...
- **400-499 Requête client incomplète**
 - 400: Bad Request, 401: Unauthorized, 403: Forbidden, 404: Not Found
- **500-599 Erreur Serveur**
 - 500: Server Error, 501: Not Implemented, 502: Bad Gateway, 503: Out Of Resources(Service Unavailable)

Exemples

Récupération d'une page (GET)

```
GET /index.html HTTP/1.0  
Accept: www/source  
Accept: text/html  
Accept: image/jpg  
User-Agent: Toto/0.0  
  
<enter>
```

<- Requête

```
HTTP/1.1 200 OK  
Date: Sat, 23 Feb 2008 16:17:07 GMT  
Server: Apache/2.2.4 (Ubuntu) DAV/2 PHP/5.2.3-1ubuntu6.3  
Last-Modified: Sat, 19 Jan 2008 08:02:12 GMT  
ETag: "42de90-3123-cafd0900"  
Accept-Ranges: bytes  
Content-Length: 12579  
Connection: close  
Content-Type: text/html  
  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//  
EN" "http://www.w3.org/TR/html4/loose.dtd">  
<html><head><title>
```

Réponse ->

Exemples

Soumission d'un formulaire (GET)

```
GET /form?cursus=miage&cours=appliweb HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/jpg

<enter>
```

<- Requête

Réponse ->

```
HTTP/1.1 200 OK
Date: Sat, 23 Feb 2008 16:19:07 GMT
Server: Apache/2.2.4 (Ubuntu) DAV/2 PHP/5.2.3-1ubuntu6.3
Last-Modified: Mon, 21 Jan 2008 07:12:12 GMT
Accept-Ranges: bytes
Content-Length: 1579
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>
```

Exemples

Soumission d'un formulaire (POST)

```
POST /form HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/jpg

cursus=miage&
cours=appliweb
```

<- Requête

Réponse ->

```
HTTP/1.1 200 OK
Date: Sat, 23 Feb 2008 16:19:07 GMT
Server: Apache/2.2.4 (Ubuntu) DAV/2 PHP/5.2.3-1ubuntu6.3
Last-Modified: Mon, 21 Jan 2008 07:12:12 GMT
Accept-Ranges: bytes
Content-Length: 1579
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html><head><title>
```

HTML

- Une page HTML est une ensemble d'éléments,
- Un élément est composé de :
 - balise ouvrante + contenu + balise fermante
 - <title>Titre de page</title>
- Les balises peuvent avoir des attributs:
 -

HTML

- Editable avec un éditeur de texte :-) mais il existe des éditeurs spécialisés:
 - Amaya, DreamWeaver
- HTML est un standard les pages doivent être valides:
 - Validateurs en ligne: W3C, Valet, ...
 - <http://validator.w3.org>, <http://valet.webthing.com>
 - Librairies de validation/nettoyage: Tidy, OpenSP, ...
 - <http://tidy.sourceforge.net>

Applets

Applications Java exécutées par le navigateur

- Restrictions
 - pas d'accès au serveur
 - pas d'accès aux fichiers sur la machine cliente
- Nécessite des plug-ins JAVA
- Possibilité d'écrire des applications complexes, ou d'utiliser des applications existantes

Applets

Le serveur rend disponible:

- une classe (ou un ensemble de classes) JAVA.
- une page HTML référençant la classe (balise `<applet>`)
- *Exemple:*

SimpleApplet.html

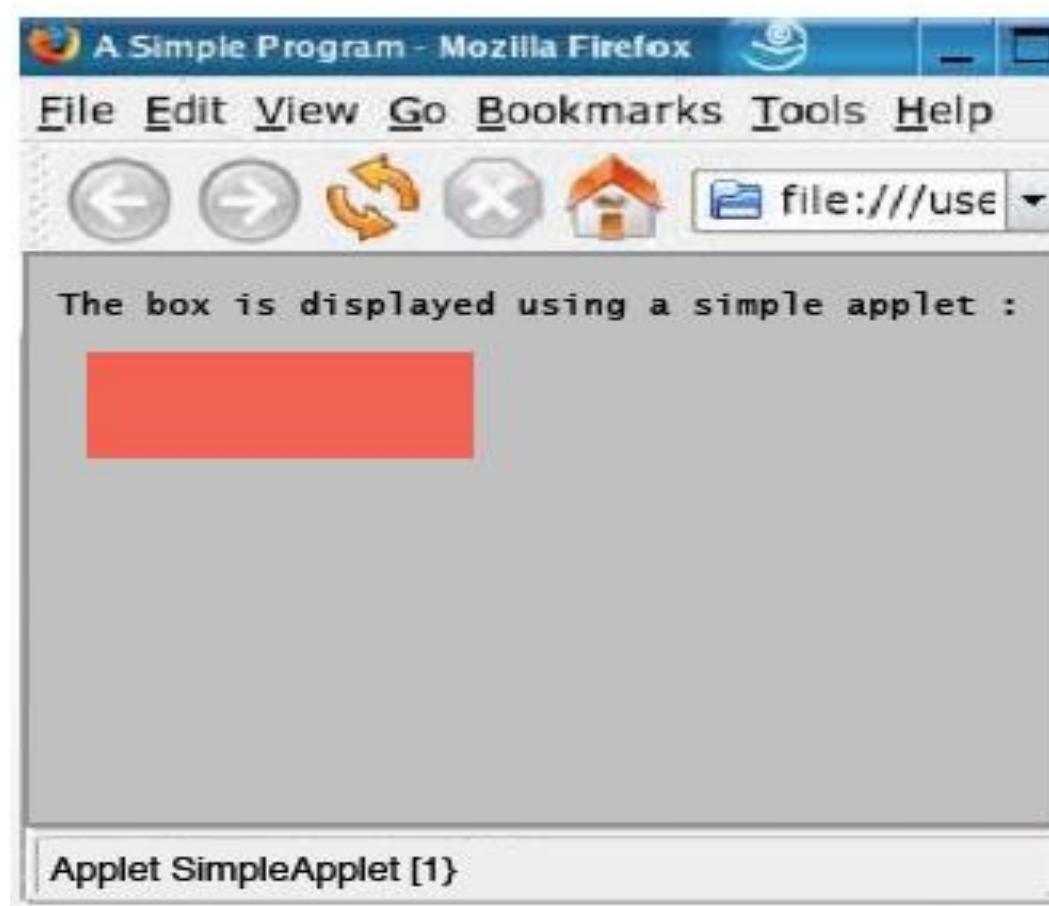
```
<html>
<head>
<title> A Simple Program </title>
</head>
<body>
<p>
The box is displayed using a simple applet : </p>
<applet code="SimpleApplet.class" width="150" height="50" />
</applet>
</body>
</html>
```

SimpleApplet.java

```
import java.applet.Applet;
import java.awt.Color;

public class SimpleApplet extends Applet{
    public void init(){
        this.setBackground(Color.RED);
    }
}
```

Applets

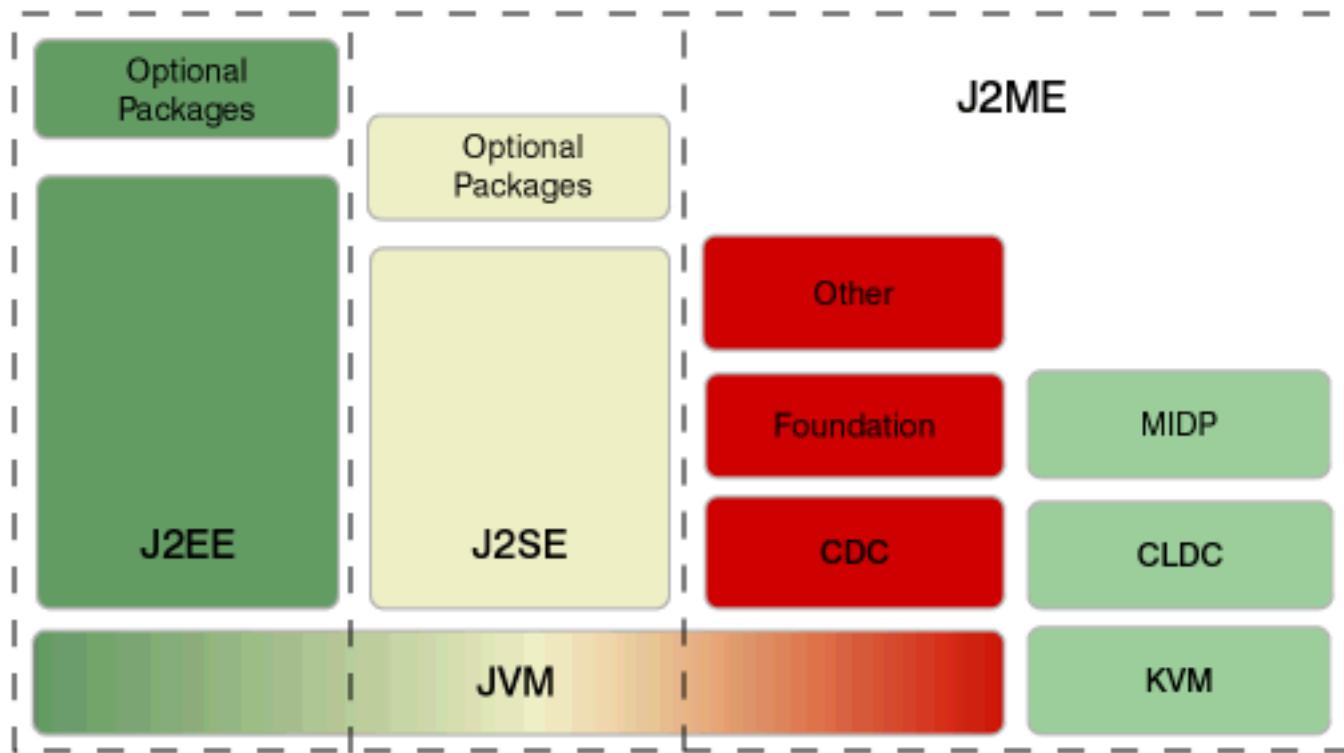


Introduction JEE

Le «**Java Framework**» est composé de trois éditions, destinées à des usages différents :

- **J2ME** : *Java 2 Micro Edition* est prévu pour le développement d'applications embarquées ;
- **JSE** : *Java Standard Edition* est destiné au développement d'applications pour ordinateurs personnels ;
- **JEE** : *Java Enterprise Edition*, destiné à un usage professionnel avec la mise en oeuvre de serveurs.

Introduction JEE



Introduction JEE

- **JEE (Java Enterprise Edition)** : standard de développement d'applications d'entreprises multi-niveaux, basées sur des composants.
- La **plate-forme JEE** présente une solution optimale pour développer des applications robustes, sécurisées et évolutives.
- On parle généralement de **plate-forme JEE** pour désigner l'ensemble constitué des services (**API**) offerts et de **l'infrastructure d'exécution**.

Introduction JEE

JEE comprend notamment :

- Les spécifications du **serveur d'application**, c'est-à-dire de l'environnement d'exécution.
- Des services, au travers d'**API**, extensions Java permettant d'offrir en standard un certain nombre de fonctionnalités.
- Suite(s) de tests
- Label JEE (qualification de plateformes)

API JEE

Les API de **JEE** peuvent se répartir en **trois** grandes catégories :

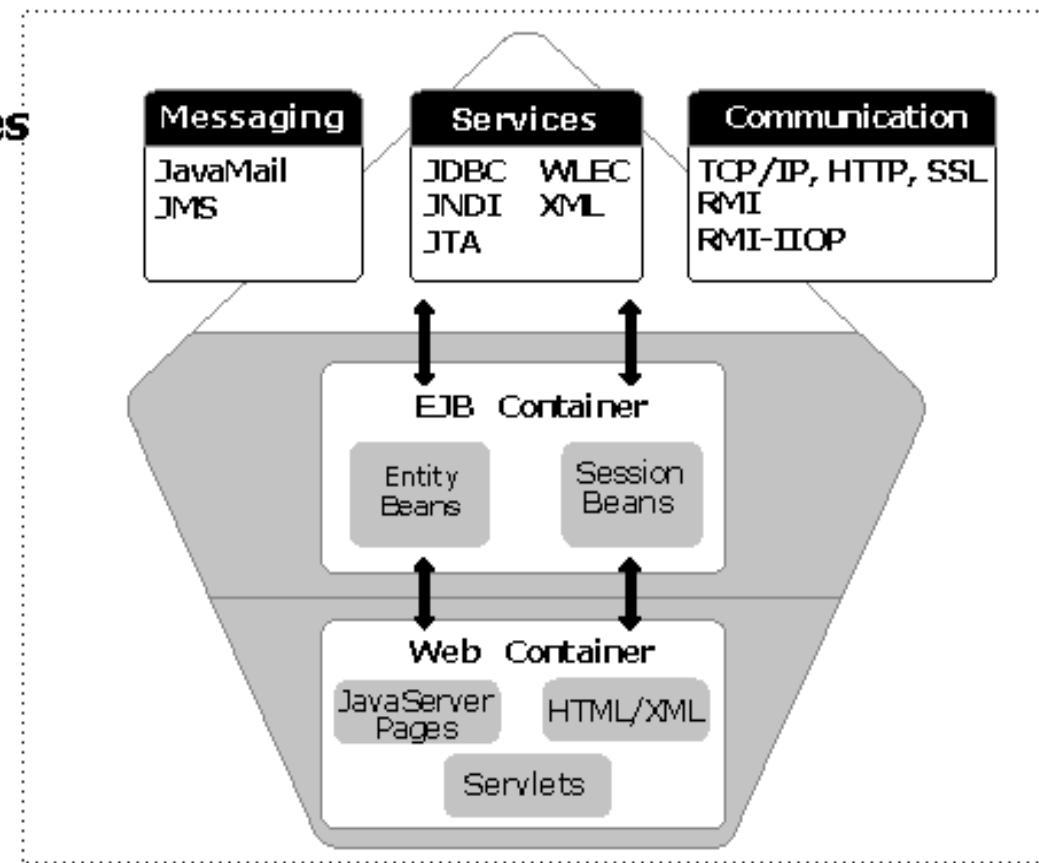
- Les **composants** ;
- Les **services d'infrastructures** ;
- Les **services de communication**.

API JEE

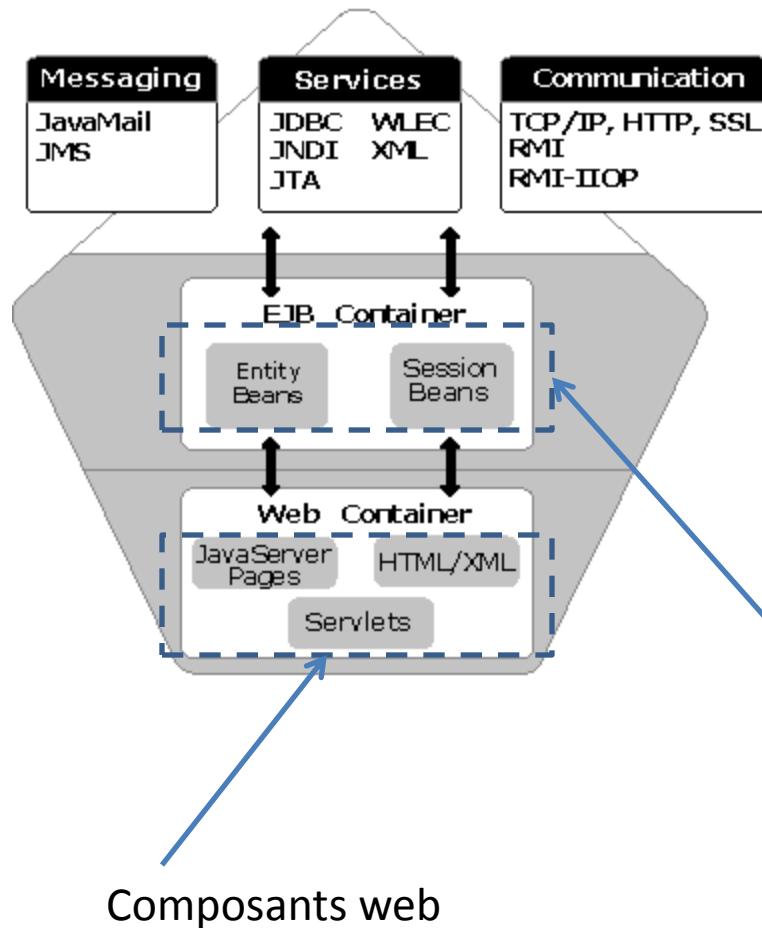
Application Services

Business Logic

Presentation Logic



API JEE: Composants



Les **composants**, on distingue habituellement deux familles de composants :

- Les **composants web** : Servlet et JSP
- Les **composants métier** : EJB

Composants Métier

Composants web

Les composants web

- **Servlet** : Traiter les données envoyées par l'utilisateur et choisir la Vue à retourner à celui-ci.

On appelle cette partie : **Contrôleur**.

- **JSP** : Les JSP sont les pages servant à générer l'ensemble du code HTML de l'interface utilisateur.

On l'appelle généralement : **Vue**.

Elles constituent les solutions techniques de base pour les applications Web en Java.

Les composants web

- Il existe aussi des framework Web qui permettent de concevoir des applications Web Java selon le design pattern MVC (Modèle – Vue – Contrôleur)
- Les deux framework Web Java les plus utilisés sont:
 - JSF
 - STRUTS

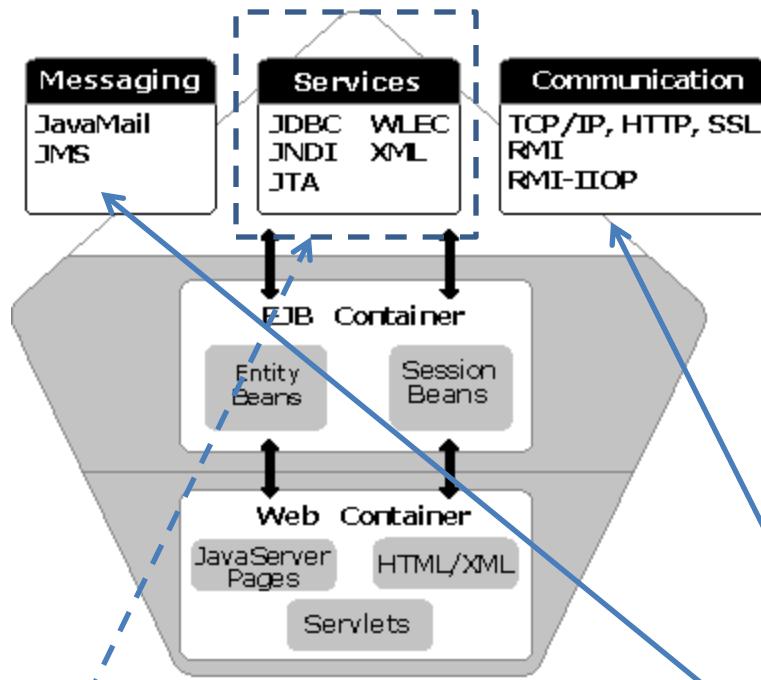
Les composants métier : EJB

- **Les composants métier : EJB** (Enterprise Java Beans).

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.

On parle de la partie : **Modèle**.

API JEE: SERVICES



Les **services**, on distingue habituellement deux familles de services :

- Les services d'infrastructures
- Les services de communication

Les services d'infrastructures

Les services de communication

Les services d'infrastructures

Il en existe un grand nombre, définis ci-dessous :

- **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 un API définissant des interfaces standard avec un gestionnaire de transactions.

Les services d'infrastructures

- **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.
- **JPA**(*Java Persistence API*)

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** est une API permettant la communication synchrone entre objets.

Architecture multi-tiers

L'architecture **JEE** repose sur des composants distincts, interchangeables et distribués, ce qui signifie notamment :

- qu'il est simple d'étendre l'architecture ;
- qu'un système reposant sur **JEE** peut posséder des mécanismes de **haute-disponibilité**, afin de garantir une bonne qualité de service ;
- que la **maintenabilité** des applications est facilitée.

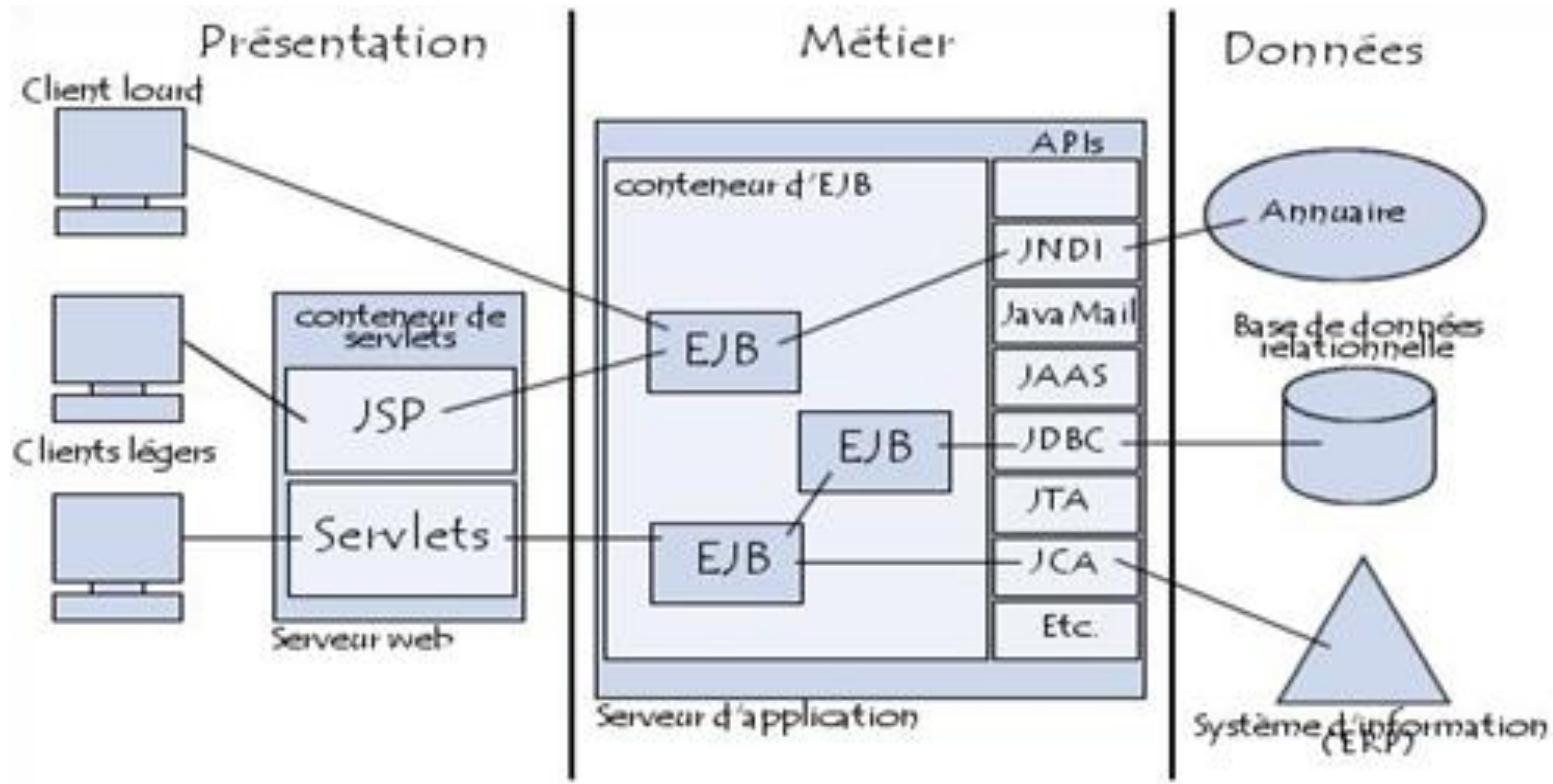


Schéma des relations entre composants et « tiers » dans l'architecture JEE

QUESTIONS ?

SOMMAIRE

- **PARTIE 1 : Rappel sur le contexte des applications Web d'entreprise**
- **PARTIE 2 : Développement Web en Java**
- **PARTIE 3 : Applications Web et servlets**
- **PARTIE 4 : Présentation des Java Server Pages**
- **PARTIE 5 : Les librairies de balises**
- **PARTIE 6 : Accès aux bases de données**
- **PARTIE 7 : Introduction à Struts**

PARTIE 2 : Plan

- **Composants nécessaires à l'utilisation de Java côté serveur**
 - Serveur Web et plate-forme serveur.
 - Moteur de servlet.
 - Java Virtual Machine.
- **Environnement de développement et d'exploitation**
 - Outils: Netbean, Eclipse
 - Architecture du conteneur web Tomcat d'Apache
 - Déploiement des servlets et pages JSP.
- **Développement d'une première servlet**
 - Génération de contenu dynamique.

Traitement d'une requête par le serveur

Avec la requête HTTP, le serveur Web :

- identifie l'environnement d 'exploitation à charger (**mapping**)
 - en fonction de l 'extension du fichier (.jsp , .cgi , .php...)
 - ou du répertoire où il se trouve (cgi-bin/, servlet/)
- charge l'environnement d 'exécution (**run-time**)
 - interpréteur Perl pour les programmes CGI en Perl
 - JVM pour les servlets Java
 - ...

Servlet: présentation

- Pour la création d'applications **dynamiques**
- **Classe java** : chargée dynamiquement, elle étend les fonctionnalités d'un serveur web et répond à des requêtes dynamiquement
 - ➔ Permet de gérer des **requêtes HTTP** et de fournir au client une **réponse HTTP**
- S'exécute par l'intermédiaire d'une **JVM**
- S'exécute dans un **moteur de Servlet** ou **conteneur de Servlet** permettant d'établir le lien entre la Servlet et le serveur Web

Servlet: avantages

- **Portabilité**: niveau systèmes et serveurs
- **Efficacité**: semi compilée, multithread, gestion du cache, connexions persistantes
- **Puissance**: communication bidirectionnelle avec le serveur web, partage de données entre servlets, chaînage de servlets
- **Pratique**: gestion des cookies, suivi des sessions, manipulation simple du protocole HTTP

Les containers de servlet

- Les servlets sont des composants
 - Ils s'exécutent dans un container
 - Ils doivent se conformer à une interface pré définie
 - C'est le container qui décide de leur instantiation
- Les containers
 - Environnement d'exécution des servlets
 - Tomcat (jakarta.apache.org/tomcat)
 - Jetty
 - Weblogic
 - Glassfish

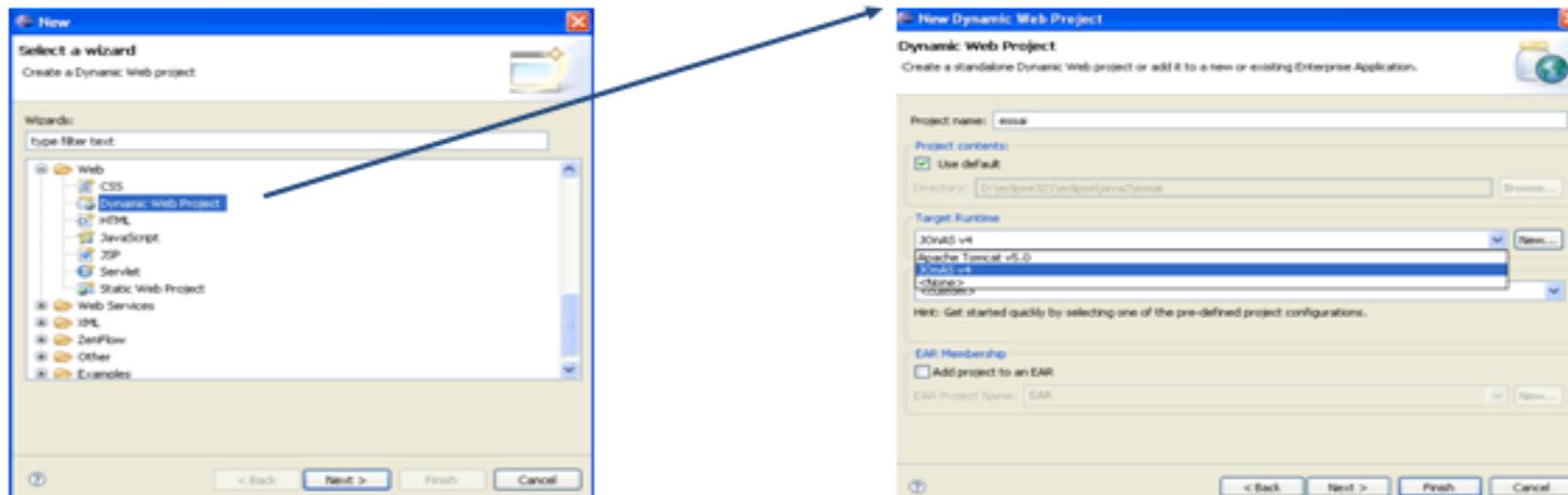
Travaux pratiques

Mise en place de l'environnement.

- *Installation de TOMCAT*

Utilisation de Eclipse

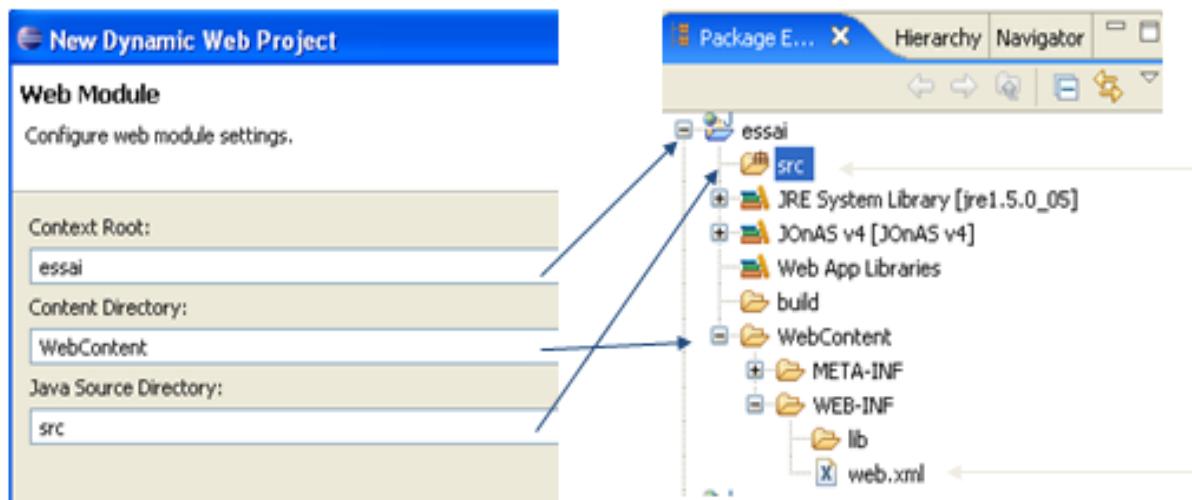
- La création d'un projet WEB avec WTP se fait comme suit :
 - Par file/New/Other...
 - Sélectionnez Dynamic Web Project.
 - Vous pouvez remarquer l'existence de wizards pour créer JSP, Servlets et autres.



- Vous remarquez la possibilité de configurer votre projet avec plusieurs « targets » possibles, comme Tomcat ou JOnAS.
 - La création d'un nouveau « target » se fait via le bouton « new »
Donnez alors la directory d'installation du serveur choisi

Utilisation de Eclipse

- Une fois la « target » configurée, vous validez le nom des directories qui seront créées.
- Vous obtiendrez alors le squelette de votre projet :

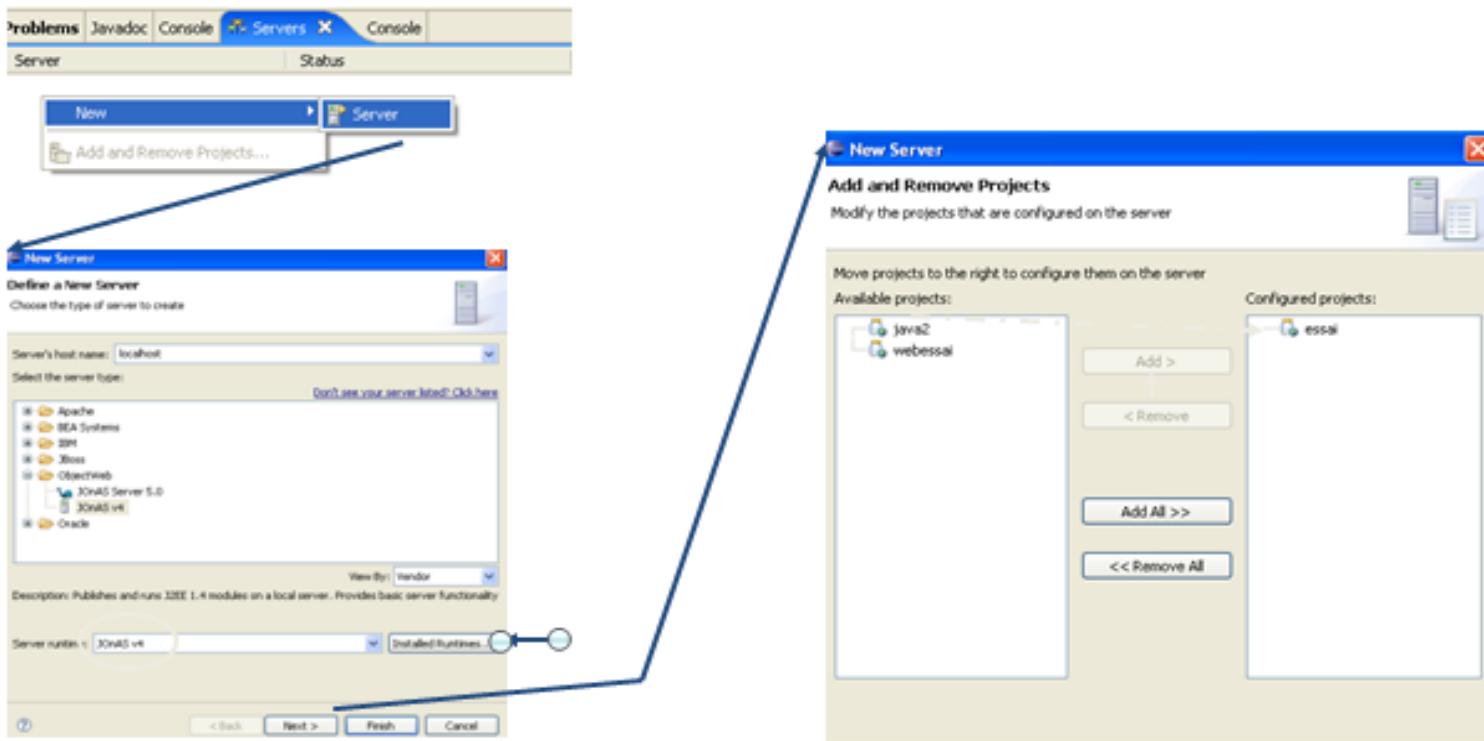


Ici seront
créés les
javabeans,
servlets,
composants

Le fichier
WEB.XML
sera modifié
en fonction
des servlets
ajoutées

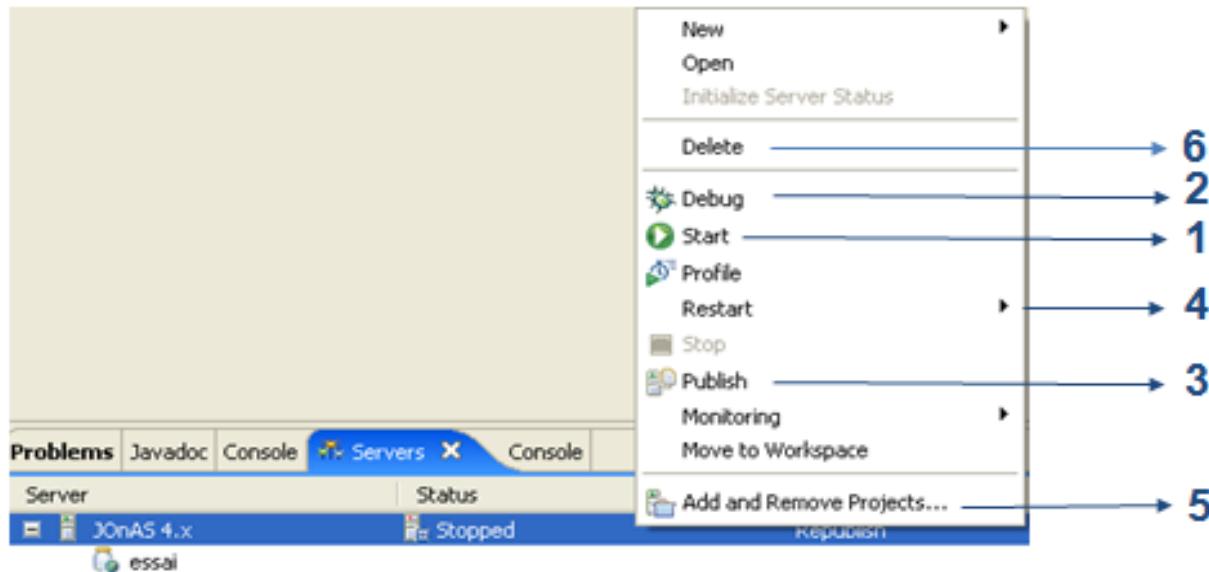
Utilisation de Eclipse

- **Via le menu Window>Show view/Other.../server/servers :**
 - Affichez la fenêtre server, vous permettant d'ajouter un contexte d'exécution à votre application Web :
 - Click droit/New/Server
 - Choisissez alors le runtime configuré
 - Enfin, sélectionnez le projet à tester dans cette configuration



Utilisation de Eclipse

- Vous pouvez alors sélectionner le menu qui vous permettra de :
 - Lancer et déployer votre webapp (1)
 - Lancer en debug votre webapp (2)
 - Re-publier votre webapp suite à une modification (3)
 - Relancer en mode normal ou debug (4)
 - Ajouter/supprimer des webapps (5)
 - Supprimer la configuration (6)
- La vue Internal Web Browser vous permettra de tester votre application sans avoir à quitter Eclipse



Travaux pratiques

Mise en place de l'environnement.

- *Installation de ECLIPSE*
- *Intégration Eclipse_Tomcat*

Outil : Netbean

- L'EDI NetBeans est un environnement de développement intégré, gratuit à l'usage.
- Il fournit du support pour tous les types d'applications Java:
 - client riche
 - Applications d'entreprises multi-couches
 - les applications pour les mobiles

Outil : Netbean

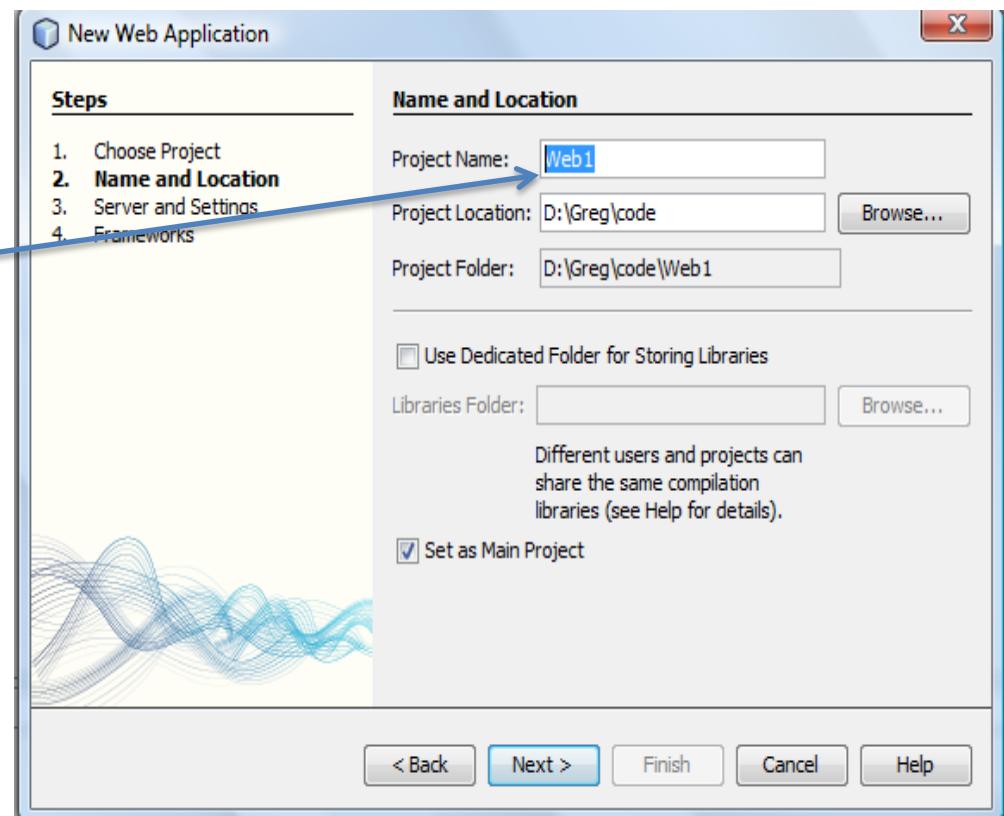
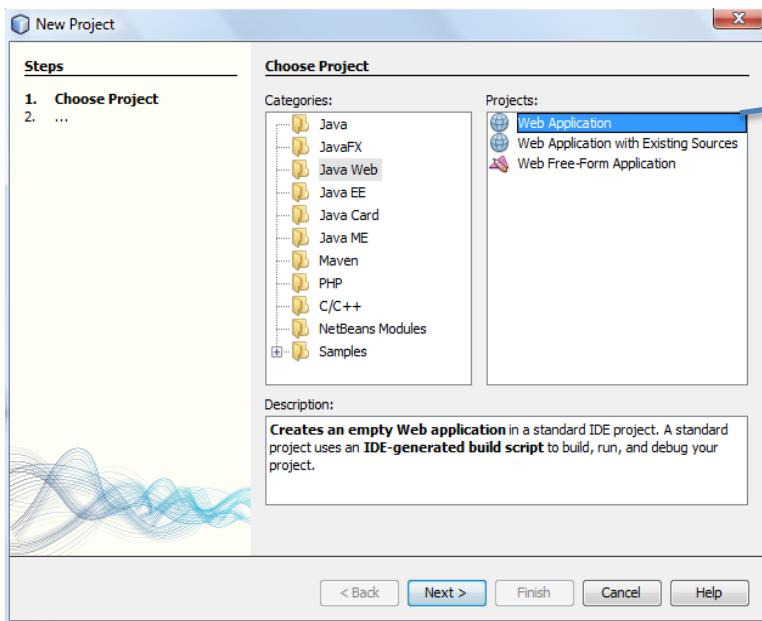
- L'EDI NetBeans a une architecture modulaire qui permet les 'plug-ins'.
- L'EDI est lui-même écrit en Java.
→ tourne (Windows, Unix, Mac ...) est disponible.
- Le job principal de l'EDI est de rendre le cycle d'édition-compilation-débogage plus agréable en intégrant des outils pour ces activités.

Util : Netbean

- **Ce Qui Vient Avec l'EDI NetBeans**
 - Apache Ant
 - Tomcat
 - Junit
 - Le Catalogue de Solutions Java BluePrints

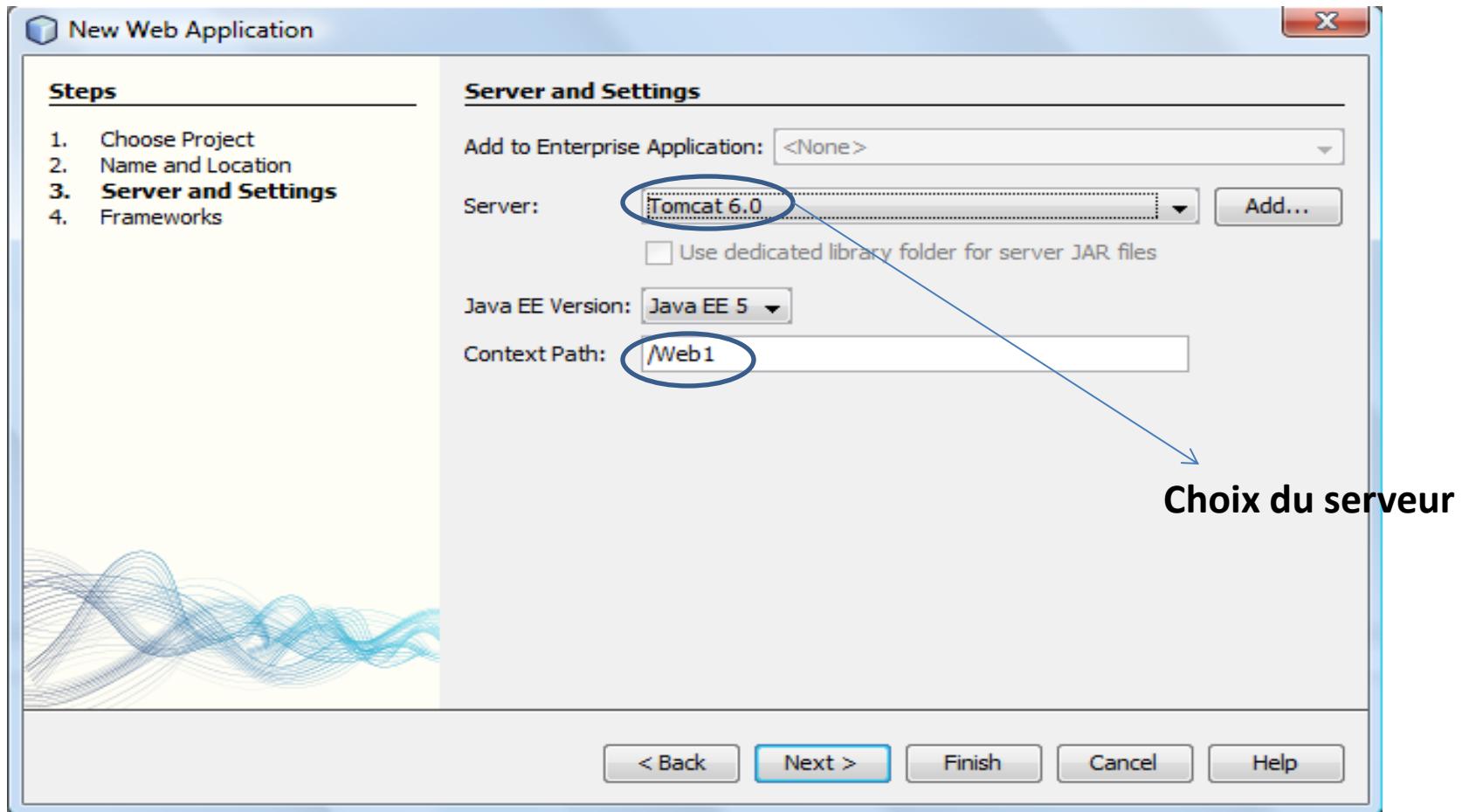
Utilisation de Netbeans

- La création d'un projet WEB avec Netbeans se fait comme suit:
 - Par File/New Project / Java Web
 - Sélectionner Web Application

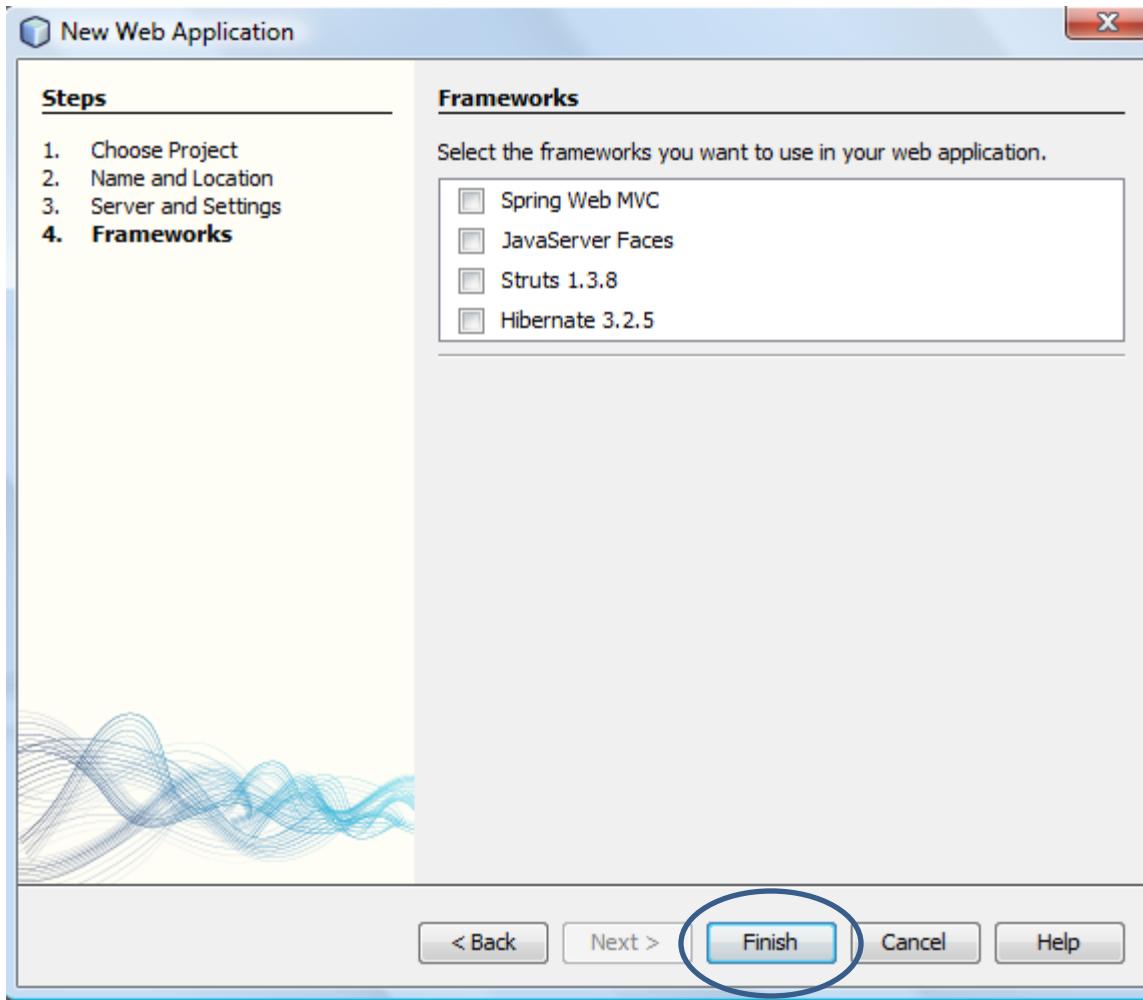


- NEXT

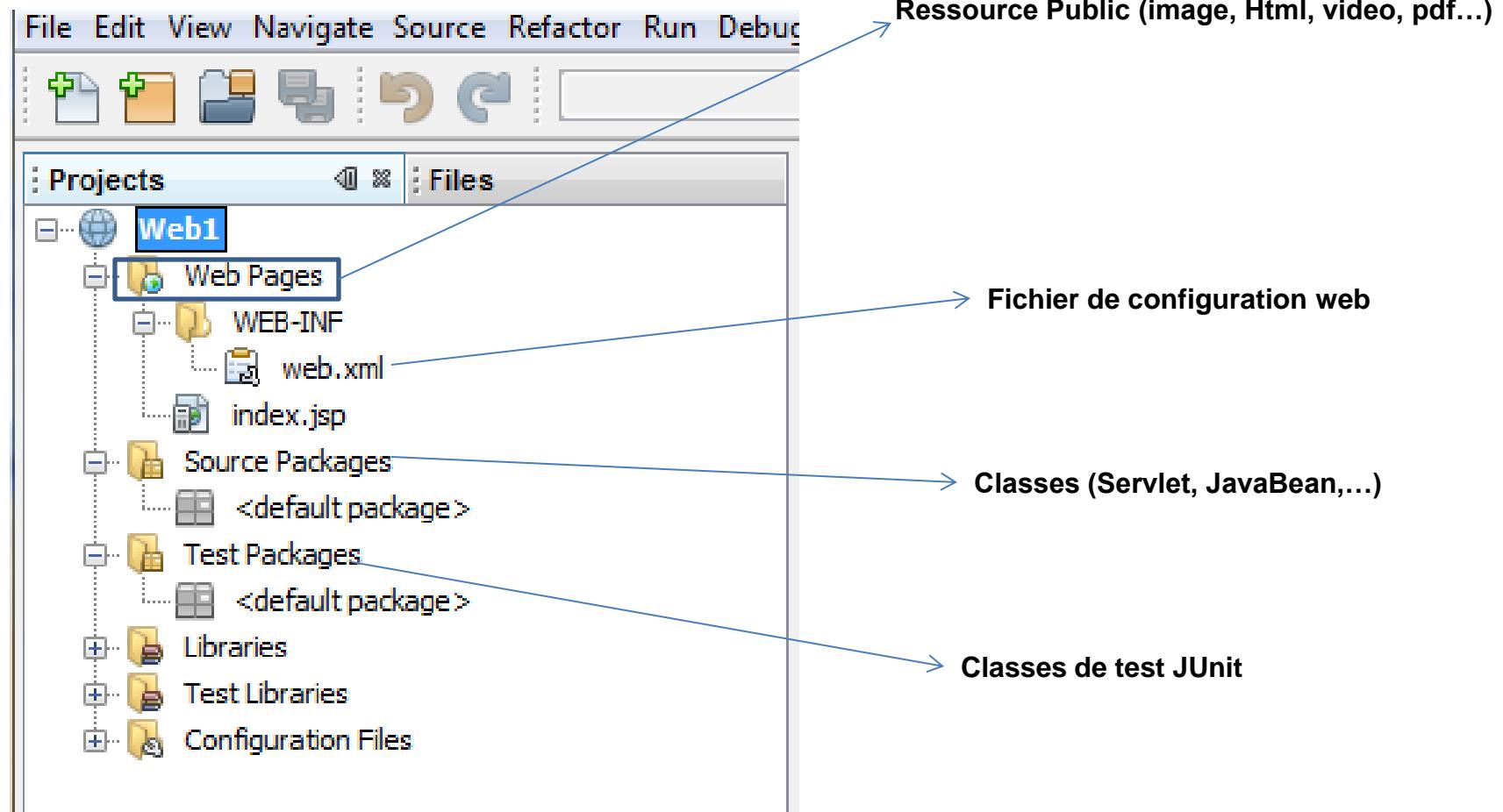
Utilisation de Netbeans



Utilisation de Netbeans

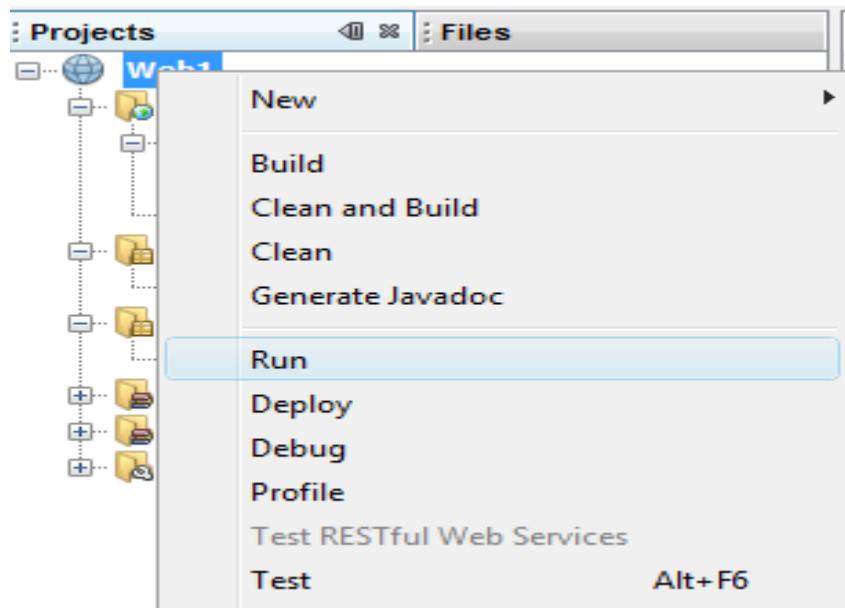


Utilisation de Netbeans

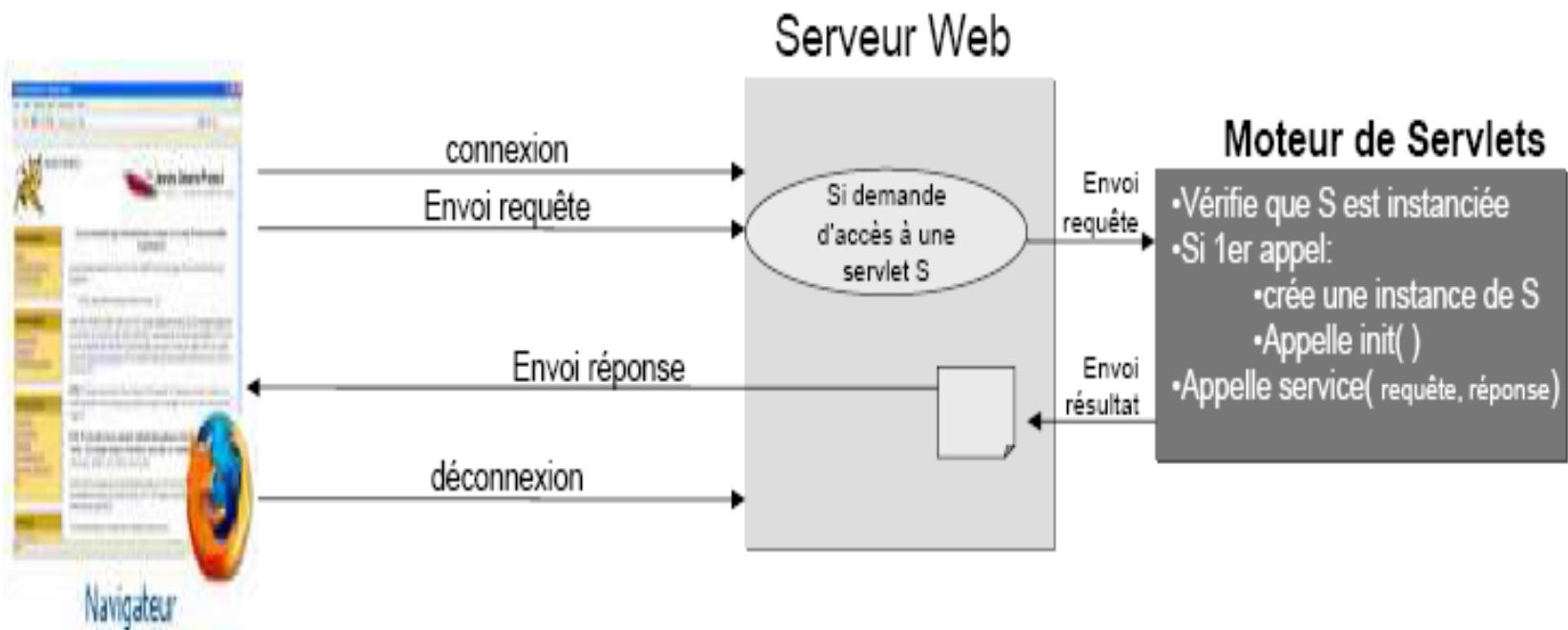


Utilisation de Netbeans

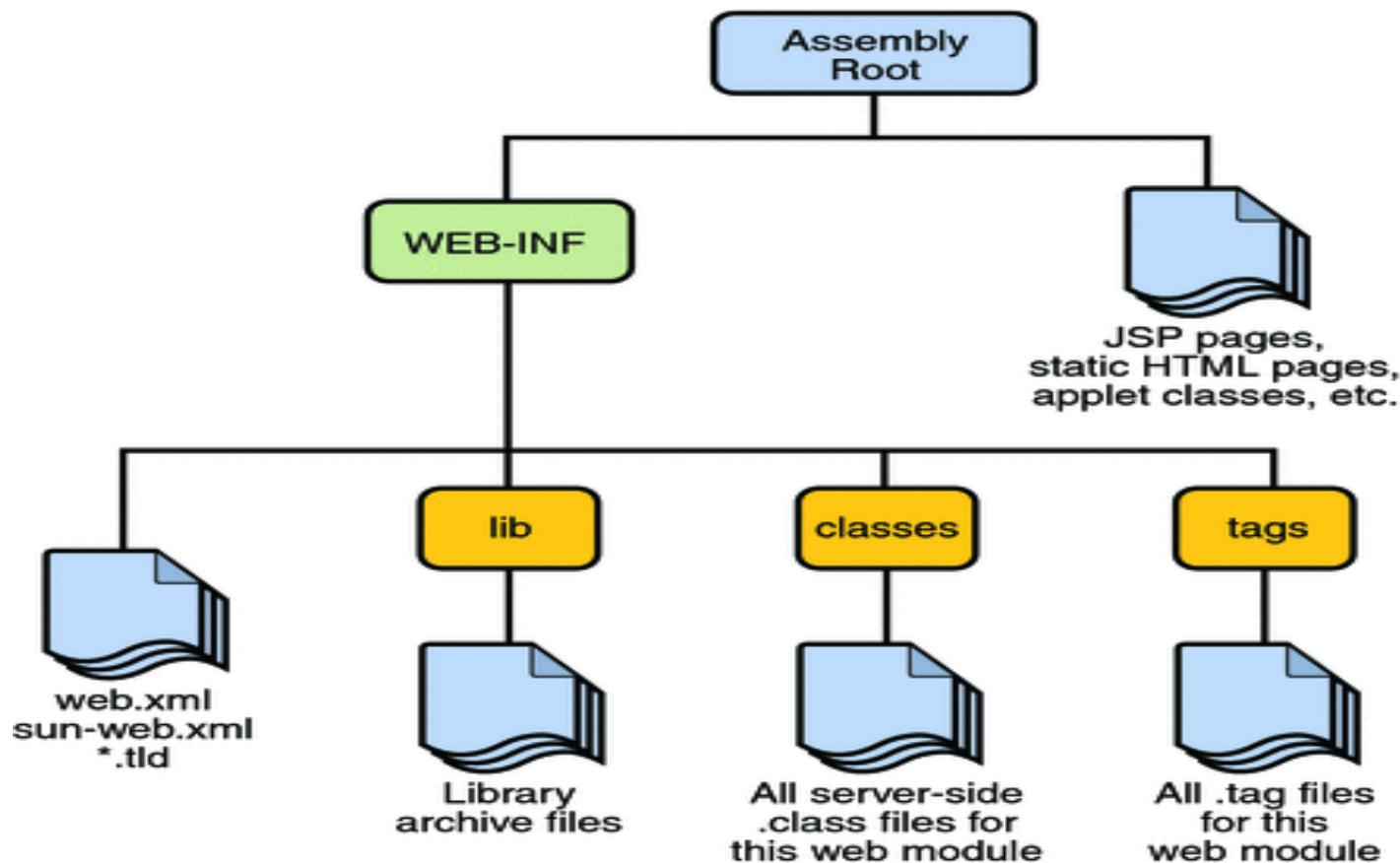
- Exécution
 - Click-droit sur le projet web1 / Run



Servlet : Fonctionnement



Structure d'une application



Structure d'une application

- Package déployable
- Composants Web : Servlet et JSP
- Ressources statiques (images)
- Classes java (helper)
- Librairies
- Descripteurs de déploiement (web.xml)

Servlet et web.xml

- Il existe une norme J2EE pour le déploiement des application WEB incluant des servlets.
- A ce titre, un fichier de configuration, nommé **web.xml** situé sous le répertoire **WEB-INF**, doit être conforme à cette norme afin que le serveur sache quelle servlet il a à exécuter en fonction de l'URL saisie :
 - Il existe donc un mapping entre chaque URL et la servlet que l'on désire exécuter.

Servlet et web.xml



1

Hello world

```
<servlet>
  <description></description>
  <display-name>HelloWorld</display-name>
  <servlet-name>Alpha</servlet-name>
  <servlet-class>com.formation.HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Alpha</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

2

4

```
public class HelloWorld extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request,
        PrintWriter out=response.getWriter());
        out.println("<b>Hello world</b>");
        out.close();
    }
```

3

Servlet et web.xml

1. L'utilisateur valide directement ou indirectement (Bouton de commande) l'URL
2. Dans le conteneur du serveur WEB est recherché le **web.xml**, et à l'intérieur l'entrée <url-pattern > qui correspond à l'url reçue (ici **/hello**)
3. Par indirection dans le fichier **web.xml**, il est trouvé le nom physique de la servlet
4. La servlet est exécutée, par l'appel de la méthode `doPost()` ou `doGet()`, dépendant du type d'appel HTTP

Servlet et web.xml

En-tête du fichier **web.xml**

Balise principale

Balise de description de l'application WEB

Balise de description d'une Servlet

Nom de la Servlet
"Identification"

Classe de la Servlet

Définition d'un chemin virtuel

Nom de la Servlet considéré
"Identification"

Définition du chemin virtuel

```
<?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>Application WEB affichant HelloWorld</display-name>
    <servlet>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>HelloWorld</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/msg.hello</url-pattern>
    </servlet-mapping>
</web-app>
```

Structure générale d'une servlet

- Écrire une servlet implique écrire du code pour gérer la requête HTTP.
- Cela signifie souvent, dans le cas où l'on a pas d'aide d'un framework MVC (STRUTS, JSF, etc..) :
 - traiter (récupérer) les paramètres
 - Contrôler la valeur des paramètres, sont ils acceptables pour l'application ?
 - Formater les paramètres, étant donné qu'ils arrivent en type String, il faut les transformer dans le type attendu (Integer, etc..)

Structure générale d'une servlet

- Stocker éventuellement ces paramètres dans des containers afin qu'ils soient récupérables lors d'un prochain appel
- Appeler l'application
 - Les composants métier
- Préparer la réponse qui peut être :
 - du code HTML en dur dans la servlet
 - Mauvaise idée car complexe à gérer (Aucun outil « WYSIWYG »)
 - Déléguee à une page spécialisée
 - JSP par exemple

Structure générale d'une servlet

- Quand le moteur de servlet appelle doGet() ou doPost(), deux objets importants sont passés en paramètre :
- HttpServletRequest
 - Permet d'accéder à tous les paramètres de la requête et aux autres informations qui ont permis l'invocation de la requête.
- HttpServletResponse
 - Sert de canal de communication vers le client et permet à la servlet de retourner du code en dur au client (des return codes) ou de rediriger vers une autre ressource (via redirect()).

Structure générale d'une servlet

- Ci-dessous, un exemple de servlet qui traite elle-même le retour client :

```
public class HelloWorldServlet extends HttpServlet{
    public void doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
        throws ServletException, IOException {
        try {
            PrintWriter out = response.getWriter();
            out.println("<html>");
            out.println("Hello World");
            out.println("</html>");
        } catch (Throwable t) {
            ...
        }
    }
}
```

- Ci-dessous, un exemple de servlet qui délègue l'affichage à une JSP ; le résultat pour l'utilisateur sera le même.

```
public class HelloWorldServlet extends HttpServlet{
    public void doPost(
        javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response)
        throws ServletException, IOException {
        try {
            RequestDispatcher rd=getServletContext().getRequestDispatcher("helloworld.jsp");
            rd.forward ( request, response );
        } catch (Throwable t) {
            ...
        }
    }
}
```

The diagram illustrates the execution flow of the servlet code. It starts with a large rounded rectangle representing the servlet's processing logic. Inside this rectangle, the code block `rd.forward (request, response);` is highlighted. An arrow points from this code block to a smaller rectangular box on the right, which contains the generated HTML output:

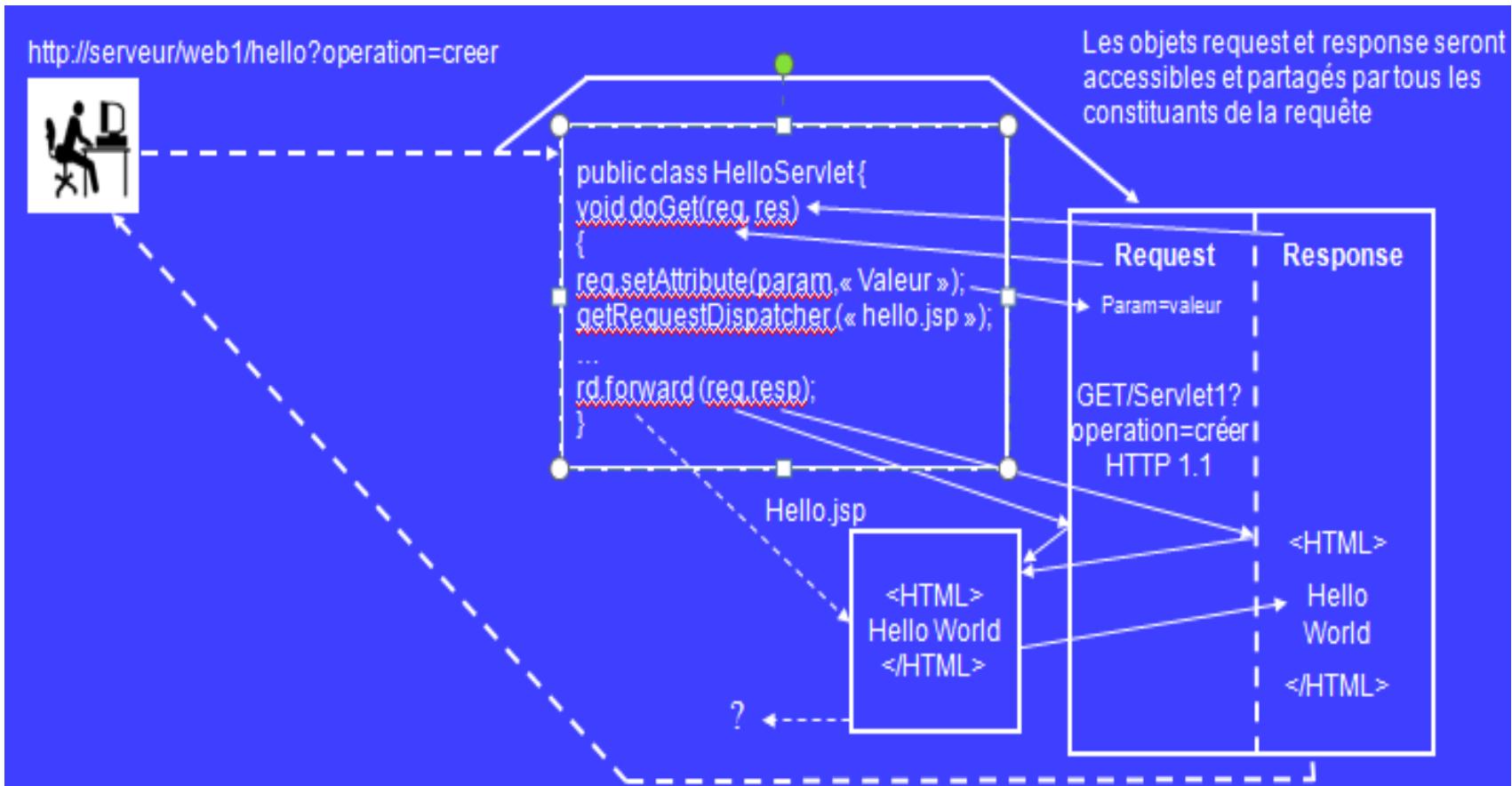
Contiendrait :

```
<html>
Hello World
</html>
```

Rôle de request et response

- Dans l'exemple précédent, on remarque que l'on stocke dans response le buffer de sortie (contenu HTML), tandis que dans l'autre, on passe au programme suivant (JSP) les deux références d'objets, request et response :
 - A tout moment, chaque élément (servlet, jsp) peut modifier le contenu de request et de response, afin, soit de constituer la sortie (response), soit de passer des paramètres supplémentaires au composant suivant (request)

Rôle de request et response



QUESTIONS ?

Travaux pratiques

Développement d'une servlet.

- *LAB_S_1*

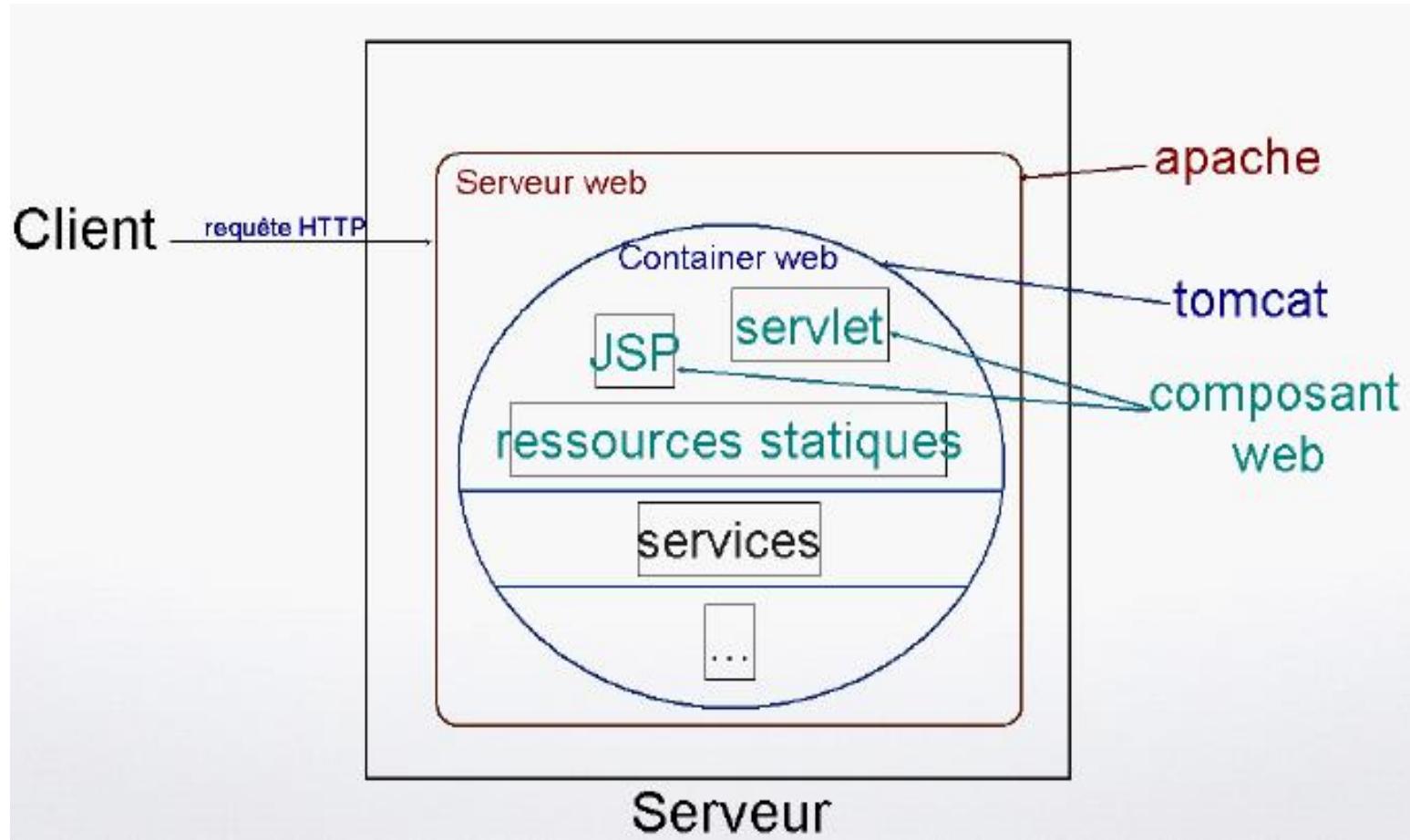
SOMMAIRE

- **PARTIE 1 : Rappel sur le contexte des applications Web d'entreprise**
- **PARTIE 2 : Développement Web en Java**
- **PARTIE 3 : Applications Web et servlets**
- **PARTIE 4 : Présentation des Java Server Pages**
- **PARTIE 5 : Les librairies de balises**
- **PARTIE 6 : Accès aux bases de données**
- **PARTIE 7 : Introduction à Struts**

PARTIE 3 : Plan

- **Développement d'une application avec des servlets**
 - Le conteneur de servlet. Le cycle de vie d'une servlet. Initialiser une servlet. Ecrire les méthodes de services. Gestion des formulaires HTML. Le traitement de la réponse, l'envoi d'information, la génération de HTML. Filtrage des requêtes/réponses. Programmation des filtres. La récupération d'information : du serveur Web, du client et de l'environnement. Invocation d'autres ressources Web. Inclusion et transfert du contrôle.
- **Gestion des erreurs et journalisation des événements**
 - Gestion des erreurs d'exécution. Gestion et emploi des exceptions Java. Envoi d'erreurs http. Journalisation des événements.
- **Suivi de session**
 - Les différentes méthodes. Obtention, consultation et abandon de session. Contexte de session.

Le conteneur de servlet

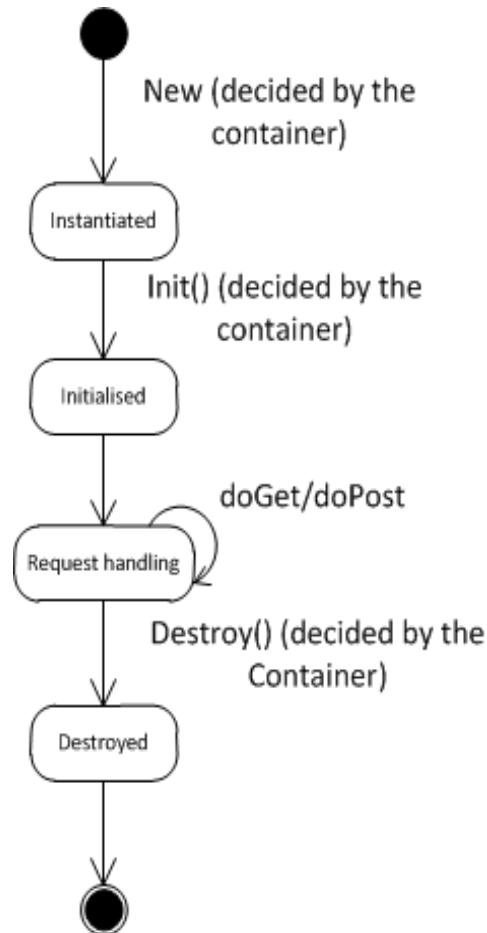


Le conteneur de servlet

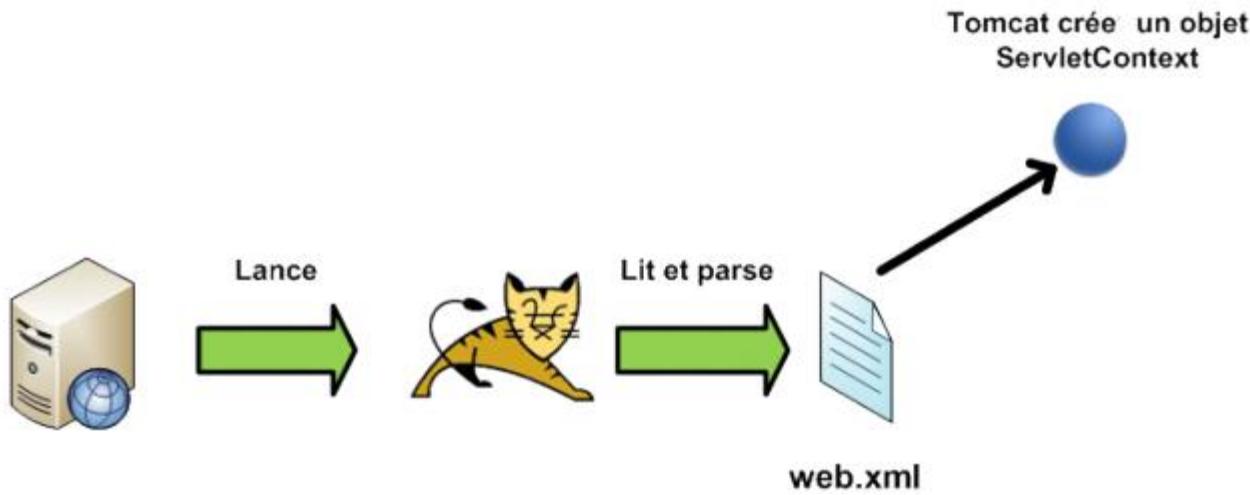
Conteneur ou moteur de Servlet

- Mode Autonome
 - Contient également un serveur web
 - Toutes les requêtes passent par le moteur de Servlet
- Mode lié au serveur Web
 - Sollicité uniquement pour le traitement des Servlet

Servlet : cycle de vie

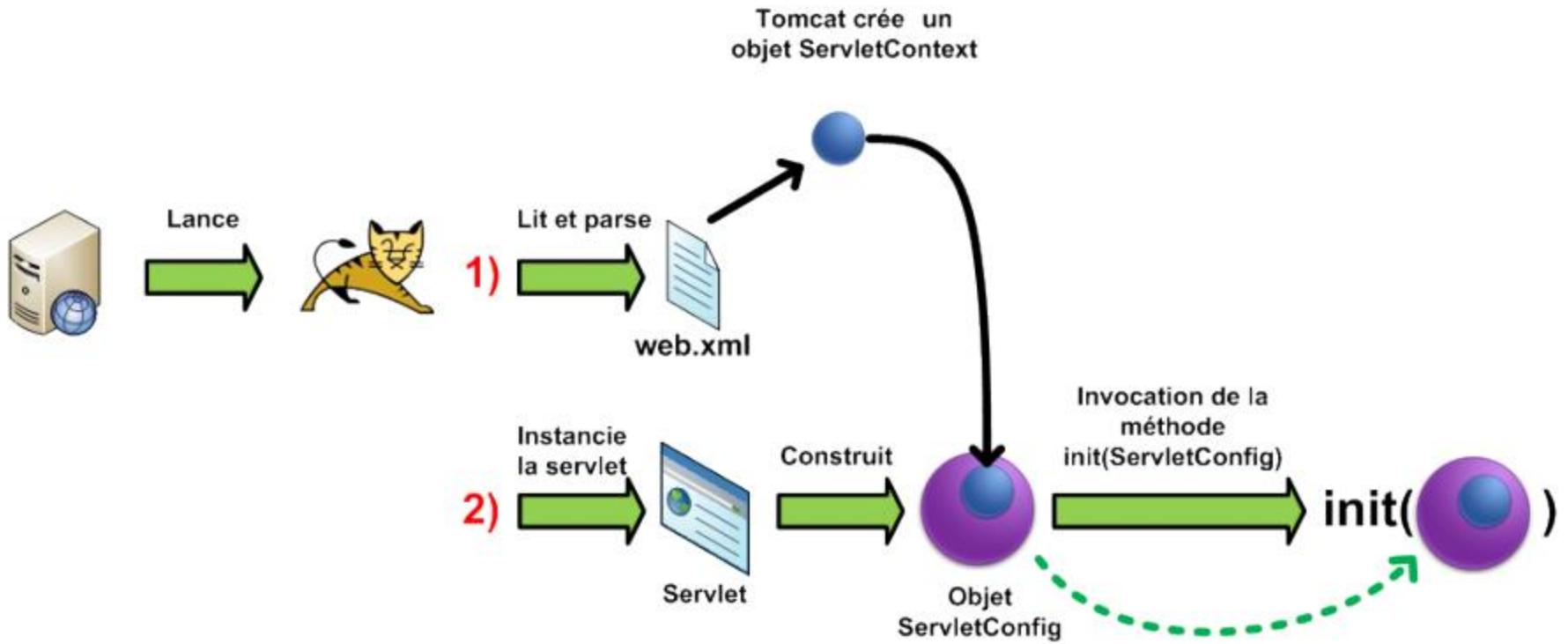


Le conteneur de servlet



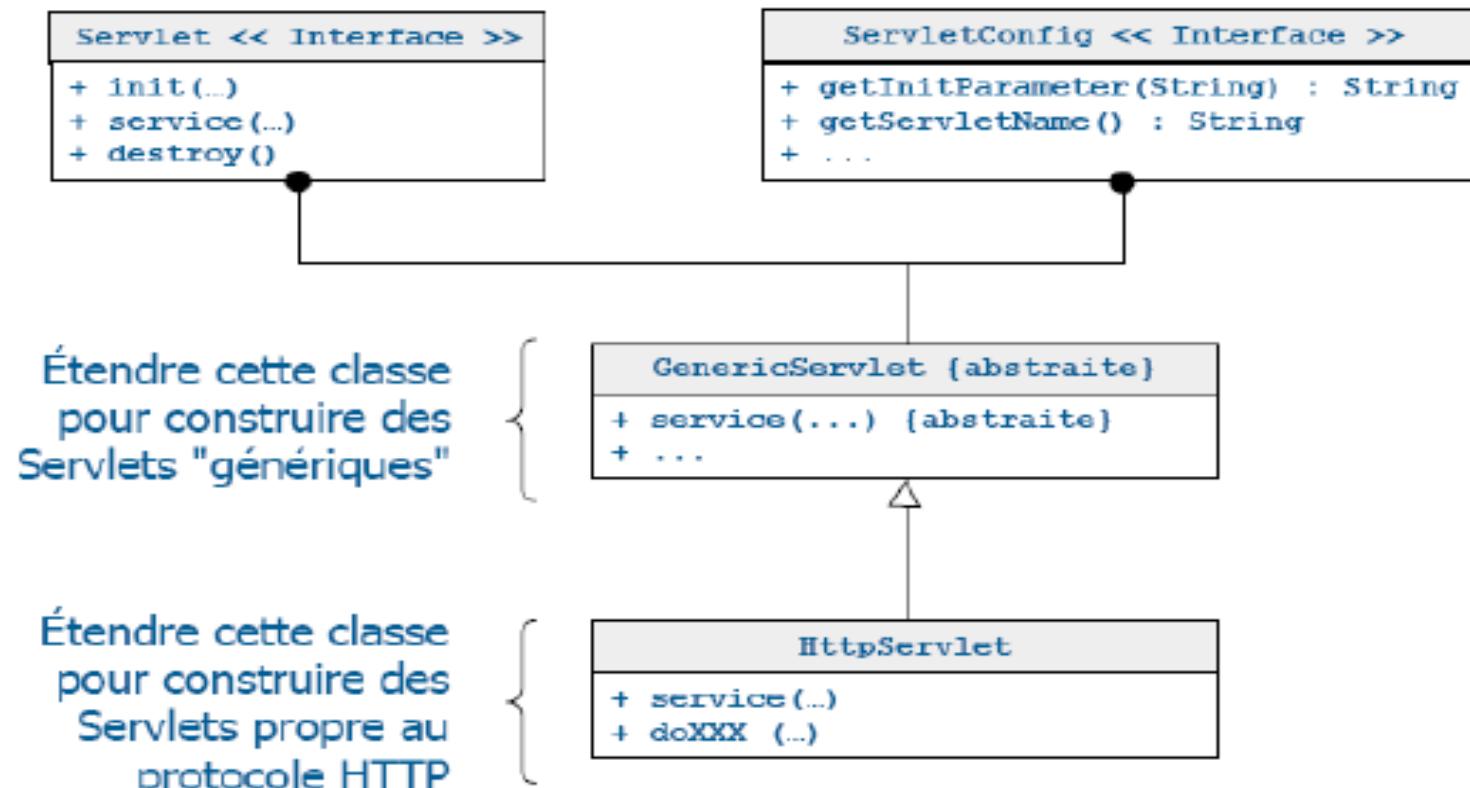
- le conteneur construit un objet **ServletContext** contenant tous les couples "clé - valeur" spécifiés dans le fichier **web.xml** correspondant aux paramètres d'initialisation de contexte
- le conteneur continue son investigation et va construire puis initialiser les servlets présentes
- le conteneur va aussi créer un objet **ServletConfig** contenant les paramètres d'initialisation de la servlet (couple clé - valeur)

Le conteneur de servlet



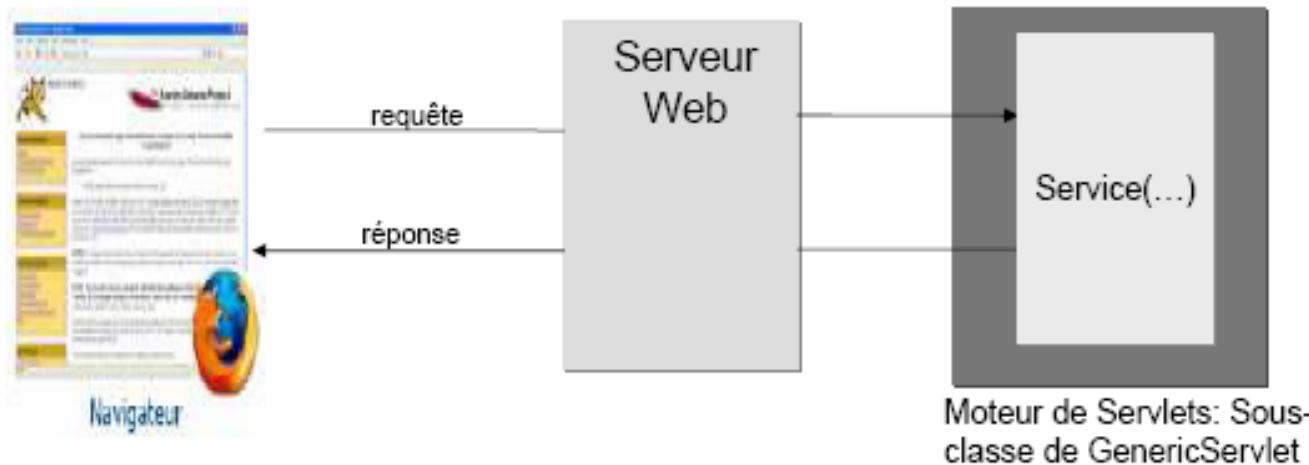
La méthode `init(ServletConfig config)` qu'utilise le conteneur ne doit pas être redéfinie !

Servlet : du générique au http

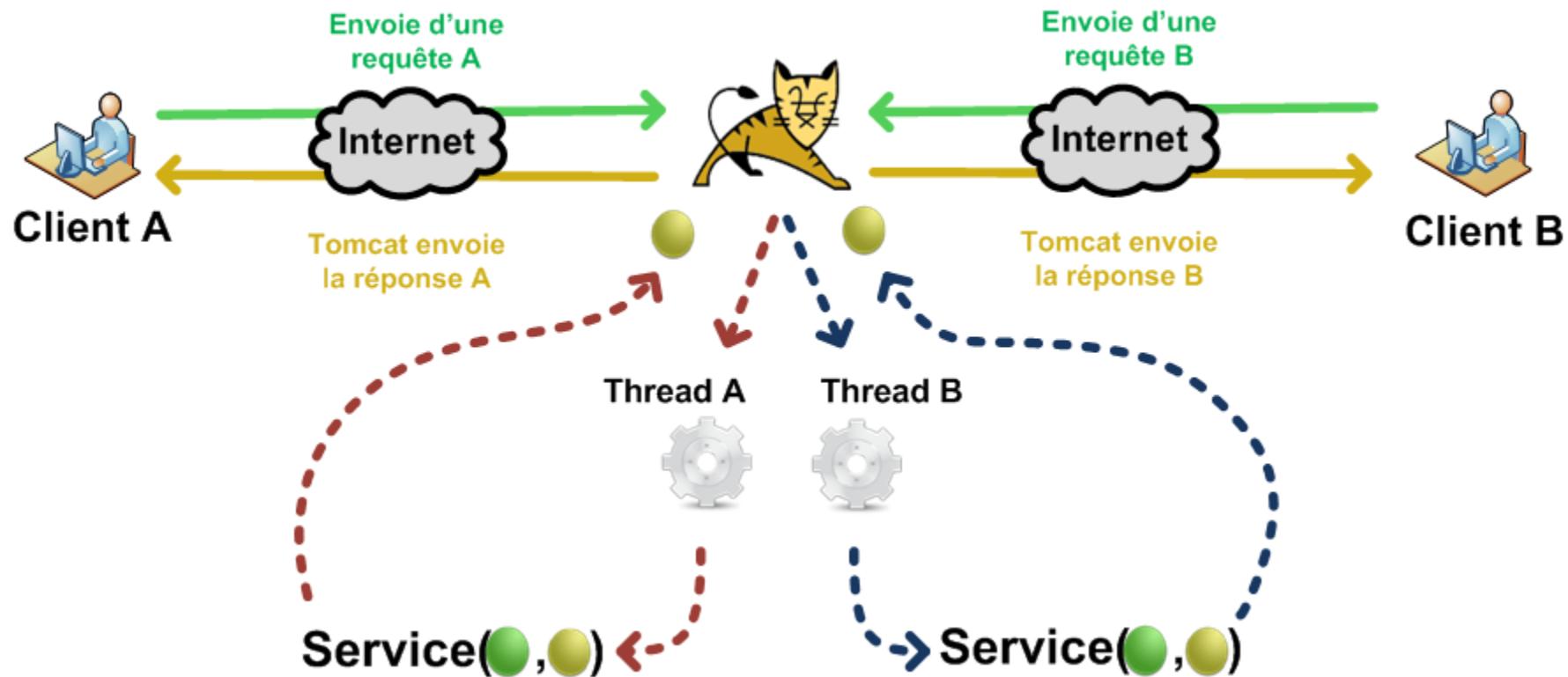


Servlet: GenericServlet

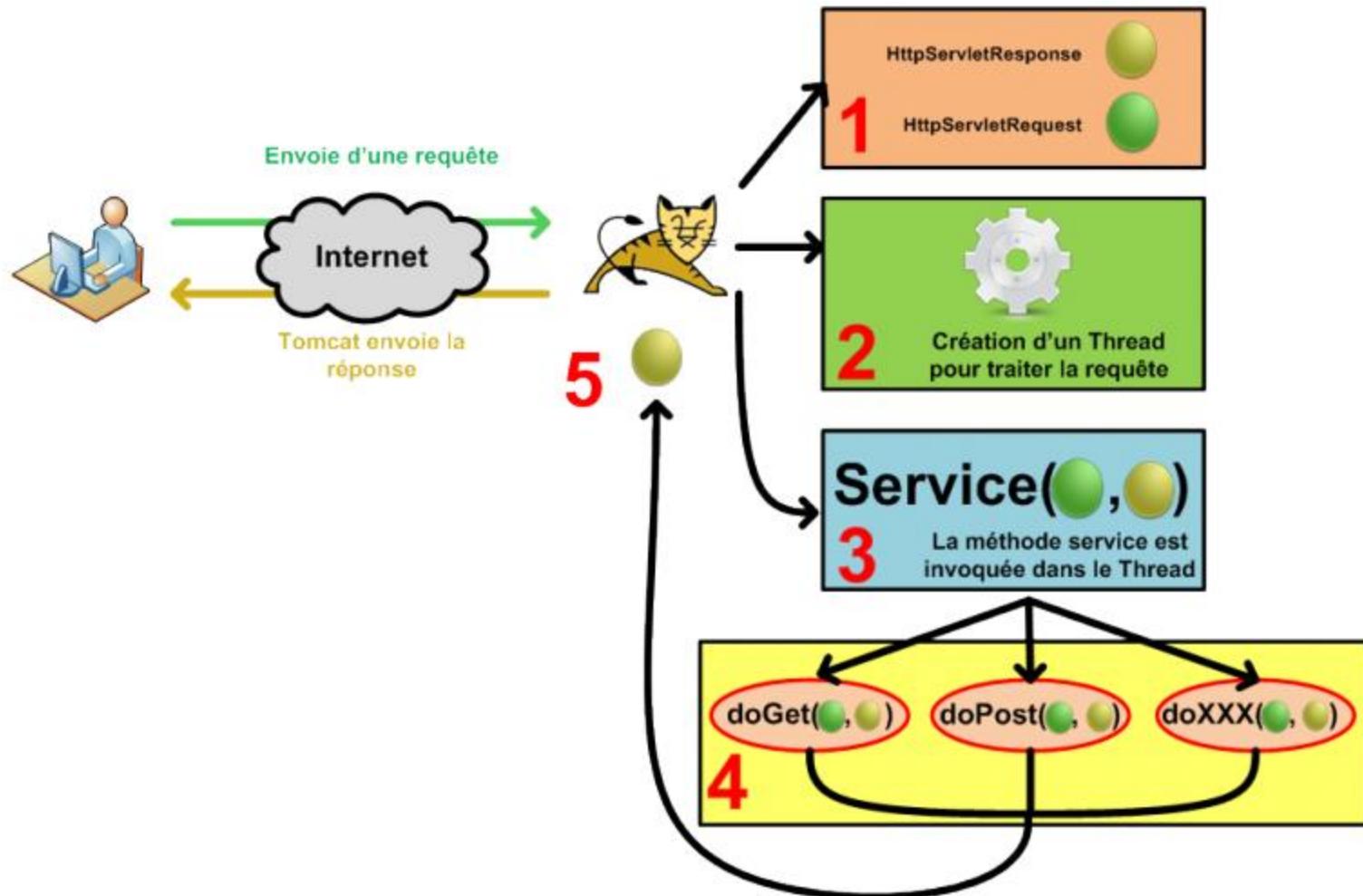
- Indépendante du protocole
- Implémentation de la méthode service(...)



Servlet: GenericServlet



Servlet: HttpServlet



Servlet: HttpServletResponse

Utilisé pour construire un message de réponse HTTP renvoyé au client

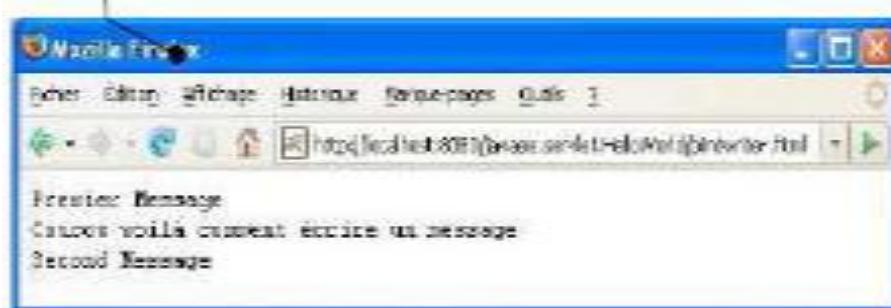
Contient :

- les méthodes nécessaires pour définir le type de contenu, en-tête et code de retour
- un flot de sortie pour envoyer des données (par exemple HTML) au client

- void setStatus(int) : définit le code de retour de la réponse
- void.setContentType(String) : définit le type de contenu MIME
- PrintWriter getWriter() : permet d'envoyer des données texte au client
- ServletOutputStream getOutputStream() : flot pour envoyer des données binaires au client
- void.sendRedirect(String) : redirige le navigateur vers l'URL

Servlet : HttpServletResponse

```
public class HelloWorldPrintWriter extends HttpServlet {  
  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
  
        PrintWriter out = res.getWriter();  
  
        out.println("Premier Message");  
        out.println("Coucou voilà comment écrire un message");  
        out.println("Second Message");  
    }  
}
```



HelloWorldPrintWriter.java

Servlet: HttpServletRequest

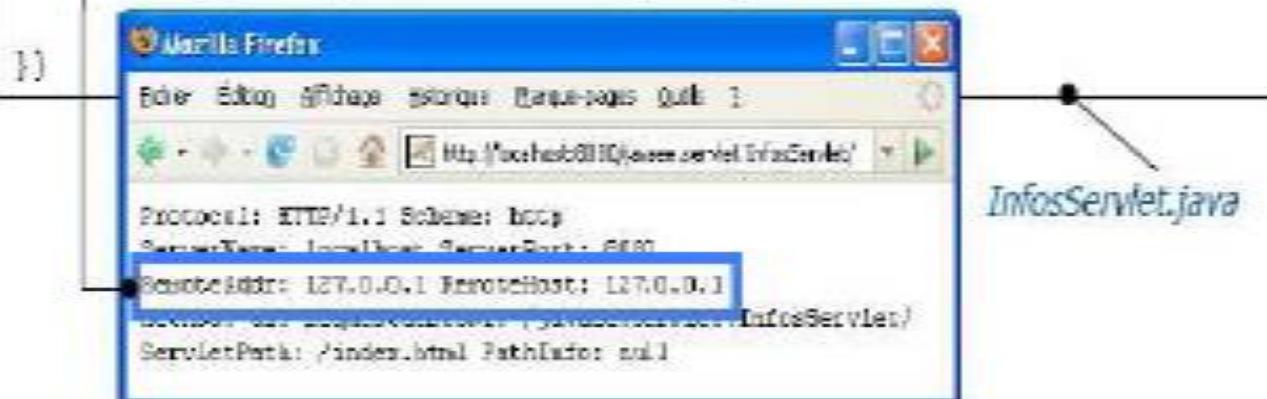
Encapsule la requête HTTP et fournit des méthodes pour :

- récupérer les paramètres passés au serveur par le client
- accéder aux informations du client et de l'environnement du serveur

- `String getMethod()` : retourne le type de requête
- `String getServerName()` : retourne le nom du serveur
- `String getParameter(String name)` : retourne la valeur d'un paramètre
- `String[] getParameterNames()` : retourne le nom des paramètres
- `String getRemoteAddr()` : retourne l'IP du client
- `String getServerPort()` : retourne le port sur lequel le serveur écoute
- ...(voir l'API Servlets)

Servlet : HttpServletRequest

```
public class InfosServlet extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        response.setContentType("text/plain");  
        PrintWriter out= response.getWriter();  
        out.println("Protocol: " + request.getProtocol());  
        out.println("Scheme: " + request.getScheme());  
        out.println("ServerName: " + request.getServerName());  
        out.println("ServerPort: " + request.getServerPort());  
        out.println("RemoteAddr: " + request.getRemoteAddr());  
        out.println("Method: " + request.getMethod());  
    }  
}
```



Les éléments du chemin de requête

- ContextPath : le chemin du contexte de déploiement
- ServletPath : la section du chemin qui a déclenché le mapping
- PathInfo : la partie de la requête qui n'est ni le ContextPath ni le ServletPath
 - Request.getContextPath()
 - Request.getServletPath()
 - Request.getPathInfo()

Exemples

```
out.println("<h1>Servlet URLServlet at " + request.getContextPath () + "</h1>");  
out.println("context path= "+request.getContextPath()+"<br>");  
out.println("servlet path= "+request.getServletPath()+"<br>");  
out.println("path info= "+request.getPathInfo()+"<br>");
```

```
<servlet-mapping>  
    <servlet-name>URLServlet</servlet-name>  
    <url-pattern>/url/*</url-pattern>  
</servlet-mapping>
```



http://localhost:8084/CoursWeb/url/servlet

context path= /CoursWeb
servlet path= /url
path info= /servlet



http://localhost:8084/CoursWeb/url/request

context path= /CoursWeb
servlet path= /url
path info= /request

Travaux pratiques

HTTP Headers

- *LAB_S_2*

Les paramètres d'initialisation

- Récupérer un paramètre d'initialisation d'une servlet

Placer la définition `<init-param>` dans la description de votre servlet dans le fichier `web.xml` comme ceci

```
<servlet>
    <servlet-name>maServlet</servlet-name>
    <display-name>Ma Servlet</display-name>
    <description>Ce que fait ma servlet</description>
    <servlet-class>com.servlet.MaServlet</servlet-class>
    <init-param>
        <param-name>MON_PARAM</param-name>
        <param-value>Bonjour</param-value>
    </init-param>
</servlet>
```

- Puis dans le code de votre servlet, utilisez ceci pour récupérer la valeur de **MON_PARAM**

```
getInitParameter("MON_PARAM");
```

- Annotation attribut `@WebInitParam`

Travaux pratiques

Récupérer un paramètre d'initialisation d'une servlet

- *LAB_S_3*

Les paramètres d'initialisation

- mettre des paramètres de manière globale afin de pouvoir les récupérer dans toutes les servlets

```
<webapp>
    ...
    <context-param>
        <param-name>Author</param-name>
        <param-value>X-plode</param-value>
    </context-param>
</webapp>
```

- Puis dans le code de votre servlet

```
getServletContext().getInitParameter("MON_PARAM");
```

Travaux pratiques

Récupérer un paramètre d'initialisation global

- *LAB_S_4*

Les attributs du context

- Valeurs disponibles pour toute l'application

```
Integer i=(Integer) getServletContext().getAttribute("hit");
if (i==null) {
    getServletContext().setAttribute("hit",1);
} else {
    getServletContext().setAttribute("hit",i+1);
}
out.println("HitCount 1 "+getServletContext().getAttribute("hit"));
    . . . . .
```

Travaux pratiques

Récupérer un attribut du context

- *LAB_S_5*

web.xml

- Le fichier web.xml donne des instructions sur le déploiement du servlet dans le container
- web-app
 - La description d'une Web Application
- Servlet
 - La relation entre le nom du servlet et la classe qui l'implante
- Servlet-mapping
 - La relation entre le nom du servlet et l'url qui permet d'y accéder
- Servlet 3.0 : remplacement possible par annotations @
 - Plus « élégant »
 - Moins « modulaire »

Le servlet mapping

- Permet de construire la relation entre un servlet et son URL

```
<servlet-mapping>
    <servlet-name>Test</servlet-name>
    <url-pattern>/Test/*</url-pattern>
</servlet-mapping>
```

- Tous les urls correspondant à
`http://host:port/webapp/url-pattern` déclencherons l'exécution du servlet

- Exemples

- `/*.do`
- `/Test`
- `/cours/test/*`

Servlet mapping (annotations)

```
@WebServlet("/foo")
```

```
public class A extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res) { ... }  
}
```

```
@WebServlet(name="MyServlet", urlPatterns={"/foo*", "/bar"},  
    initParams = {@WebInitParam(name="x", value="Hello "),  
        @WebInitParam (name="y", value=" World!") })
```

```
public class B extends HttpServlet {
```

```
...  
}
```

Autre propriétés

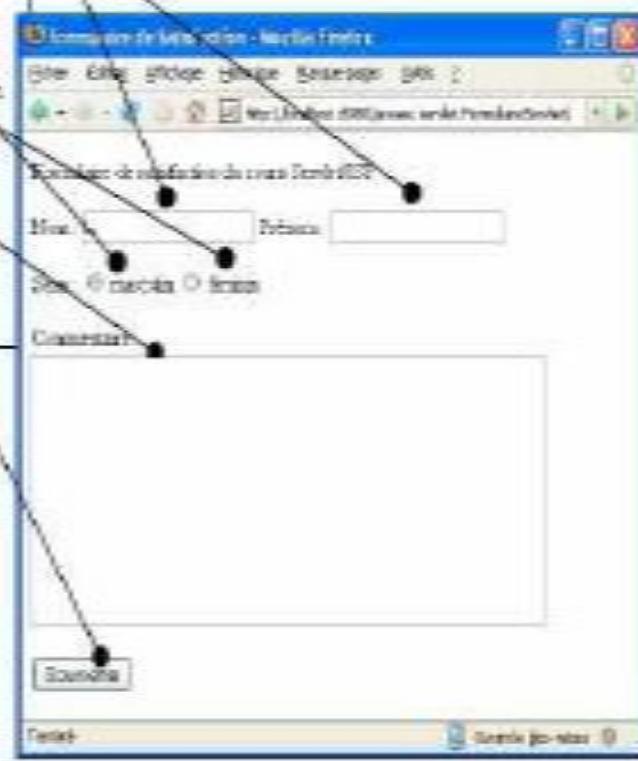
- Pour le mapping mime
 - <mime-mapping>
- Pour les fichiers à charger quand un nom de répertoire est donné
 - <welcome-file-list>
- Pour les pages d'erreur
 - <error-page>
 - Pour chaque code d'erreur on peut fixer une page spécifique
- Il y en a d'autres pour
 - La sécurité
 - Les taglibs
 - Les références aux ressources utilisés par les servlets

Servlet et formulaires

```
<body>
<p>Formulaire de satisfaction du cours Services/FSI </p>
<form name="form1" method="get"
      action="/juveee.servlet.FormulaireServlet/form">
<p>
    Nom : <input type="text" name="nom"> _____
    Prénom : <input type="text" name="prenom"> _____
</p>
<p>
    Sexe :
    <input type="radio" name="radiol" value="sexe" checked="masculin" /> masculin
    <input type="radio" name="radiol" value="sexe" /> féminin
</p>
<p>Commentaire :<br>
    <textarea name="textarea" cols="50" rows="10"></textarea><br>
    <input type="submit" name="Submit" cols="5" value="Soumettre">
</p>
</form>
```

index.html

Le formulaire appelle une Servlet avec une requête de type GET



Servlet et formulaires

Accéder aux paramètres par l'intermédiaire de l'objet
HttpServletRequest :

- String getParameter (String p) : retourne la valeur du paramètre p
- String[] getParamterValues (String p) : retourne les valeurs du paramètre p

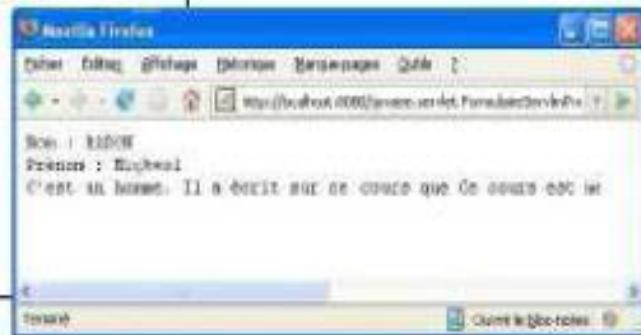
Servlet et formulaires

```
public class FormulaireServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("Nom : " + req.getParameter("nom"));
        out.println("Prénom : " + req.getParameter("prenom"));

        if (req.getParameterValues("radiol") [0].equals("mas")) {
            out.print("C'est un homme. Il");
        } else {
            out.print("C'est une femme. Elle");
        }

        out.print(" a écrit sur ce cours que ");
        out.println(req.getParameter("textarea"));
    }
}
```

FormulaireServlet.java



Travaux pratiques

Récupérer des paramètres utilisateur (Formulaire)

- *LAB_S_6*

Sessions

- Dans une application web, le client a besoin de faire différentes requêtes au serveur:
 - Webmail
 - Site de e-commerce
- Une **session** est l'exécution d'une application web pour un utilisateur donné. C'est un espace de travail limité dans le temps et associé à un espace de stockage d'informations propre à l'utilisateur.

Sessions

HTTP ne gère pas la notion de sessions:

- Pas d'informations sur les requêtes précédentes d'un client
- Pas pratique pour nombre d'applications ayant besoin de gérer des sessions.

Comment implanter les sessions?

Sessions

- Création par le serveur d'identifiant de session fourni au client, qui le renvoie à chaque requête
 - un identifiant de session défini l'état et la durée de validité d'une session
- Plusieurs méthodes pour les échanges d'identifiants entre le client et le serveur:
 - **ré-écriture d'URL**
 - input **HIDDEN** dans les formulaires
 - **Cookies**
 - **HttpSession**

Sessions: ré-écriture d'URL

- L'identifiant de session est encodé dans l'URL des pages retournées par le serveur:

URL originale	identifiant de session
<code>http://localhost/webapp/example;jsessionid=323aed39902</code>	

- Différents codage possibles:

<code>http://server/servlet/Rewritten</code>	URL originale
<code>http://server/servlet/Rewritten/123</code>	information <i>extra-path</i>
<code>http://server/servlet/Rewritten?sessionid=123</code>	ajout de paramètre
<code>http://server/servlet/Rewritten;\$sessionid\$123</code>	spécifique au serveur

- Chaque technique à ces avantages et ces inconvénients

Sessions: input HIDDEN

- L'identifiant de session est “caché” dans une entrée HIDDEN des formulaires retournés par le serveur:

- Exemple:

```
<form method="post" action="/cgi-bin/select">
    <input type="checkbox" name="art123">
        Java™ Servlet Programming, O'Reilly
    ...
</form>
```

réponse de /cgi-bin/select ->

```
<form method="post" action="/cgi-bin/validate">
<INPUT TYPE="hidden" NAME="SessionID" VALUE="54109848932"> ←
    Identifiant: <INPUT TYPE="text" NAME="login">
    Adresse:     <INPUT TYPE="text" NAME="mail">
    N° Client:   <INPUT TYPE="text" NAME="numclient">
    ...
    <INPUT TYPE="hidden" NAME="Language" VALUE="French">
</FORM>
```

- Inconvénients:

- Pas de persistance de l'ID côté client
- Ambiguité en cas d'utilisation de “Back”

Travaux pratiques

Gestions de session (Champ caché)

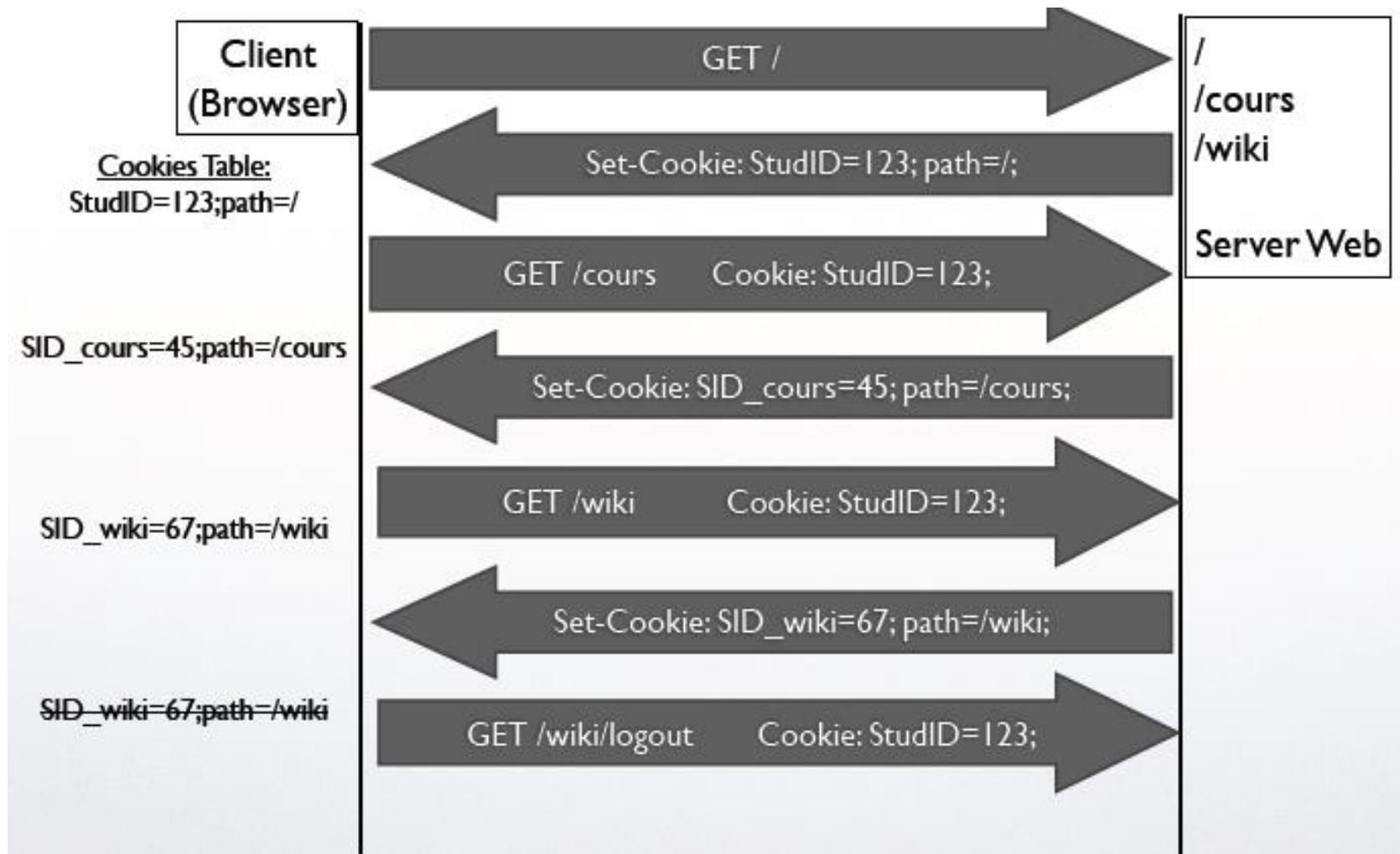
- *LAB_S_7*

Servlet : cookies

Rappel:

- cookie : information envoyée au navigateur (client) par un serveur WEB qui peut ensuite être relue par le client
- Lorsqu'un client reçoit un cookie, il le sauve et le renvoie ensuite au serveur chaque fois qu'il accède à une page sur ce serveur

Cookies



Servlet : cookies

- Classe javax.servlet.http.Cookie pour utiliser 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

Servlet : cookies

- Classe javax.servlet.http.Cookie pour utiliser 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

Servlet : cookies

- Création d'un nouveau cookie
 - ⇒ Ajout à la réponse (HttpServletResponse)
 - `addCookie(Cookie mon_cook)` : ajoute à la réponse un cookie
- Récupération des cookies du client
 - ⇒ Récupération dans la requête (HttpServletRequest)
 - `Cookie[] getCookies()` : récupère l'ensemble des cookies du site

Servlet : 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();
    }
}
```

Servlet: cookies

Exercice:

Implémenter la méthode doGet pour gérer les cookies:

- Si le client est déjà connu affichage d'un message:
« Encore vous »
- Sinon attribution d'un cookie (sessionId) au client et
affichage d'un message: « Bonjour le nouveau »

Utiliser `java.rmi.server.UID().toString` pour générer un identifiant

Servlet: cookies

```
public class CookiesServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        ...  
        String sessionId = null;  
        Cookie[] cookies = req.getCookies();  
        if (cookies != null) {  
            for (int i = 0; i < cookies.length; i++) {  
                if (cookies[i].getName().equals("sessionid")) {  
                    sessionId = cookies[i].getValue();  
                }  
            }  
            if (sessionId == null) {  
                sessionId = new java.rmi.server.UID().toString();  
                Cookie c = new Cookie("sessionid", sessionId);  
                res.addCookie(c);  
                out.println("Bonjour le nouveau");  
            } else {  
                out.println("Encore vous"); ...  
            }  
        }  
    }  
}
```

Génère un identifiant unique pour chaque client

Servlet: HttpSession

API de suivi de session 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 getAttributNames() : retourne les noms de tous les attributs
 - Object getAttribut(String name) : retourne l'objet associé au nom
 - setAttribut(String na, Object va) : modifie na par la valeur va
 - removeAttribut(String na) : supprime l'attribut associé à na
- Destruction (HttpSession)
 - invalidate() : expire la session
 - logout() : termine la session

Servlet: HttpSession

Exercice:

Implémenter la méthode doGet pour gérer le suivi de session:

Affiche un compteur qui est incrémenté à chaque accès sur cette servlet pendant une session.

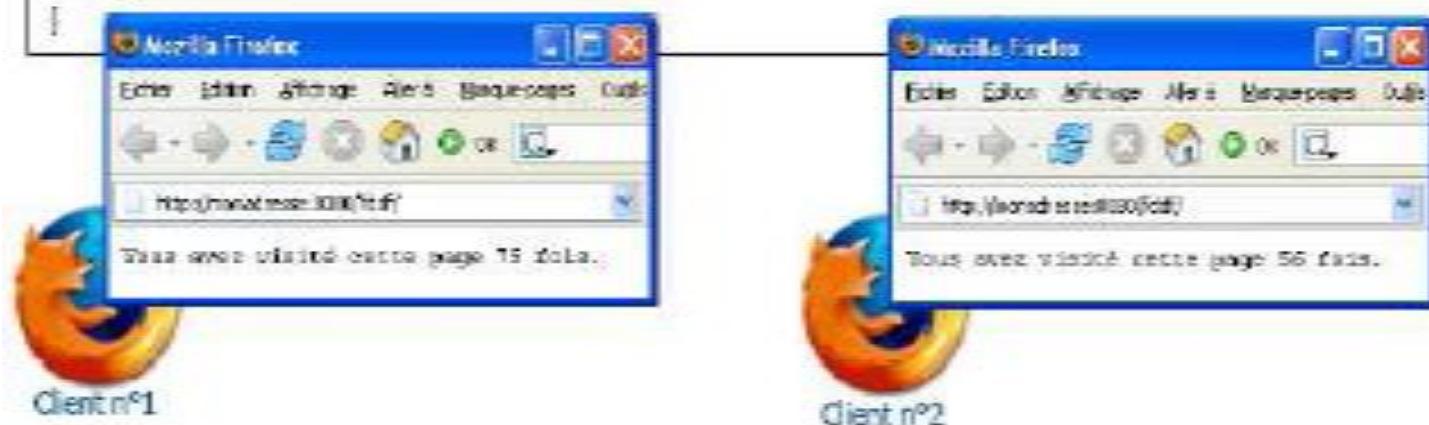
Travaux pratiques

Gestions de session (Cookie)

- *LAB_S_8*

Servlet: HttpSession

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



Servlet: HttpSession

- Modifier le Time Out par défaut de session?

Dans le fichier web.xml situé dans le répertoire WEB-INF de l'application, il suffit d'ajouter les lignes suivantes:

```
<session-config>
    <session-timeout>60</session-timeout>
</session-config>
```

- Modifier le Time Out par de session par programmation?

```
request.getSession().setMaxInactiveInterval(int);
```

- Terminer une session

```
out.println("<h1>Servlet TerminateSession at " + request.getContextPath () + "</h1>");
out.println("Date de création"+new Date(request.getSession().getCreationTime()));
request.getSession().invalidate();
```

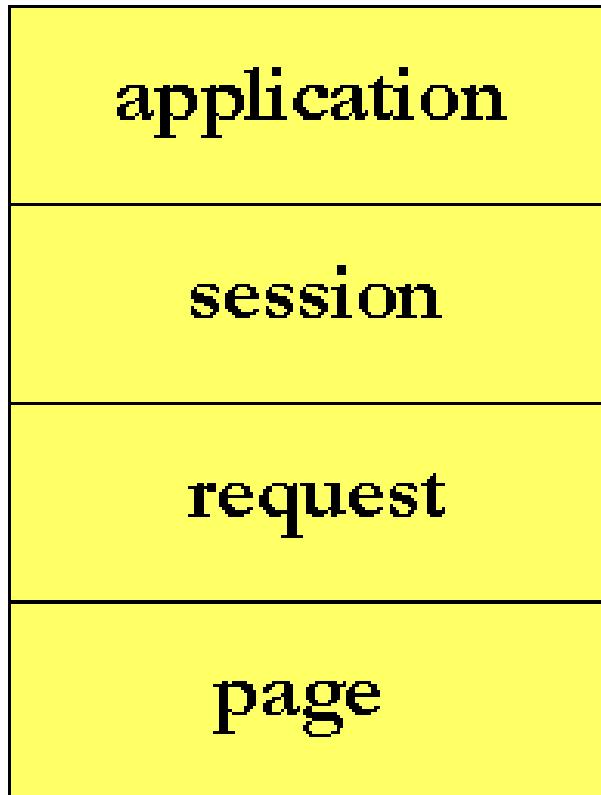
Travaux pratiques

Gestions de session (`HttpSession`)

- *LAB_S_9*

La visibilité

Most
visible



Least
visible

Objects accessible from pages that belong to the same application

Objects accessible from pages belonging to the same session as the one in which they were created

Objects accessible from pages processing the request where they were created

Objects accessible only within pages where they were created

Les filtres

- Introduits dans la version 2.3 de la spec.
- Intercepter les requêtes et les réponses et de les transformer ou pour utiliser les informations qu'elles contiennent.
- Ne créent pas de réponses, mais fournissent des fonctions “universelles” qui peuvent être associées à n’importe quelle servlet.

Les filtres

- Un filtre est un morceau de code exécuté entre la requête et le « endpoint »
- Permettent de faire du pre et post-processing sur une requête
 - Lire la requête, modifier la requête modifier la réponse, retourner des erreurs au client
- Même cycle de vie qu'un servlet
 - init / doFilter / destroy

Intérêt des filtres

- Permettent d'étendre l'application sans changer les servlets
- Services transversaux en entrée et en sortie
- En entrée
 - Log des appels
 - Sécurité
 - Contrôle des paramètres
- En sortie
 - Post processing du flux (XSLT)
 - Compression (gzip)
- Fonctionnent comme un pipe & filter

Ecrire des filtres

- L'API des filtres est définie dans 3 classes:
 - **Filter**,
 - **FilterChain**
 - **FilterConfig** dans javax.servlet
- On définit un filtre en implantant la classe Filter,
- Une chaîne de filtres (FilterChain) décrit comment un série de filtres est exécuté et
- Le FilterConfig contient les données d'initialisation.

Coder un filtre

- Implanter la méthode doFilter()
- Déclarer le filtre
 - web.xml : <Filter> <Filter-mapping>
 - Servlet 3 : @WebFilter("cible")
- Même fonctionnement que pour un servlet
- Le transfert à la suite de la chaîne se fait par la fonction chain.doFilter()
 - Transfert à un autre filtre ou à un servlet ou une page HTML ou une page JSP
 - La suite s'exécute au retour de doFilter

Travaux pratiques

Filtre

- *LAB_S_10*

Les listeners

- Les listeners sont des objets dont les méthodes sont invoquées en fonction du cycle de vie d'un servlet
- A la création et à la destruction d'un contexte (une appli)
 - **Javax.servlet.ServletContextListener**
- Quand on modifie les attributs du contexte
 - **Javax.servlet.ServletContextAttributeListener**
- A la création, la suppression, le timeout d'une session
 - **Javax.servlet.HttpSessionListener**
- A la création, modification, suppression d'un attribut de session
 - **Javax.servlet.HttpSessionAttributeListener**

Les listeners du contexte

ServletContextListener

Void **contextDestroyed**(ServletContextEvent sce)

Notification that the servlet context is about to be shut down.

void **contextInitialized**(ServletContextEvent sce)

Notification that the web application is ready to process requests.

ServletContextAttributeListener

void **attributeAdded**(ServletContextAttributeEvent scab)

Notification that a new attribute was added to the servlet context.

void **attributeRemoved**(ServletContextAttributeEvent scab)

Notification that an existing attribute has been removed from the servlet context.

void **attributeReplaced**(ServletContextAttributeEvent scab)

Notification that an attribute on the servlet context has been replaced.

Les listeners de session

HttpSessionListener

Void sessionCreated(HttpSessionEvent se)

Notification that a session was created.

void sessionDestroyed(HttpSessionEvent se)

Notification that a session was invalidated.

HttpSessionAttributeListener

void attributeAdded(HttpSessionBindingEvent se)

Notification that an attribute has been added to a session.

void attributeRemoved(HttpSessionBindingEvent se)

Notification that an attribute has been removed from a session.

void attributeReplaced(HttpSessionBindingEvent se)

Notification that an attribute has been replaced in a session.

La déclaration d'un listener

```
<listener>
    <description>HttpSessionListener, HttpSessionAttributeListener</description>
    <listener-class>exemple.MyListener</listener-class>
</listener>
```

- Annotation *@WebListener*

Exemple

```
public class MyListener implements HttpSessionListener, HttpSessionAttributeListener {  
  
    public void sessionCreated(HttpSessionEvent arg0) {  
        System.out.println("Session Created"+arg0.getSession());  
        arg0.getSession().setAttribute("newsession", "ma valeur");  
    }  
  
    public void sessionDestroyed(HttpSessionEvent arg0) {  
        System.out.println("Session Destroyed"+arg0.getSession());  
    }  
  
    public void attributeAdded(HttpSessionBindingEvent arg0) {  
        System.out.println("Session Attrbute Added"+arg0.getName());  
        System.out.println("Session Attrbute Added"+arg0.getSession().getAttribute(arg0.getName()));  
    }  
  
    public void attributeRemoved(HttpSessionBindingEvent arg0) {  
        throw new UnsupportedOperationException("Not supported yet.");  
    }  
  
    public void attributeReplaced(HttpSessionBindingEvent arg0) {  
        throw new UnsupportedOperationException("Not supported yet.");  
    }  
}
```

Exemple

```
public class SessionCounterListener implements HttpSessionListener {  
  
    public void sessionCreated(HttpSessionEvent arg0) {  
        Integer i=(Integer) arg0.getSession().getServletContext().getAttribute("count");  
        arg0.getSession().getServletContext().setAttribute("count",i+1);  
        System.out.println("Nombre de sessions = "+(i+1));  
    }  
  
    public void sessionDestroyed(HttpSessionEvent arg0) {  
        Integer i=(Integer) arg0.getSession().getServletContext().getAttribute("count");  
        arg0.getSession().getServletContext().setAttribute("count",i-1);  
        System.out.println("Nombre de sessions = "+(i-1));  
    }  
}
```

Application déployée

Session Created AB4130236FA5B9E054232D468ED9F070

Session Attribute Added newsession

Session Attribute Value ma valeur

Nombre de sessions = 1

Session Attribute Added user

Session Attribute Value john

Nombre de sessions = 0

Session Destroyed AB4130236FA5B9E054232D468ED9F070

Exemple (Servlet 3.0) avec déclarations

```
@WebListener()
public class MyListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent sce) {
        System.out.println(sce.getServletContext().getContextPath()+":activated");
    }

    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println(sce.getServletContext().getContextPath()+":Destroyed");
    }
}
```

Collaboration Inter-Servlet

Les servlets peuvent partager ou déléguer le contrôle de la requête

Deux types de collaborations :

- Délégation : une Servlet peut *envoyer* une requête entière à une autre servlet
- Inclusion : une Servlet peut *inclure* du contenu généré par une autre servlet

Les avantages sont :

Délégation de compétences

Meilleure abstraction et architecture logicielle MVC

Collaboration de servlets : redirection

```
public class Emetteur extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        req.setAttribute("nom", "toto"); // getAttribute(...) pour
                                         // retrouver la valeur
        RequestDispatcher disp =
            req.getRequestDispatcher("/recepteur.do?age=22");
        disp.forward(req, res);
        // Ne rien faire sur req et res
    }
}
```

```
public class Recepteur extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println(req.getParameter("age")); // chaîne de caractères
        out.println(req.getAttribute("nom")); // objet
    }
}
```

Collaboration de servlets : redirection

- Illustration de redirection externe : alternative 1 : redirection différée du client vers une autre URL

```
...
public class ClientMove extends HttpServlet {
    static final String NEW_HOST = "http://www.newhost.com";

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

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String newLocation = NEW_HOST + req.getRequestURI();
        res.setHeader("Refresh", "5; URL=" + newLocation);
        out.println("Changement d'URL : <A HREF=\"\"");
        out.println(NEW_HOST+ "\">"+ NEW_HOST +"</A><BR>");
        out.println("Vous serez redirigé dans 5 secondes...");
    }
...
}
```

- Illustration de redirection externe : alternative 2 : redirection immédiate du client vers une autre URL : sendRedirect(String)

Travaux pratiques

RequestDispatcher (Forwards)

- *LAB_S_11*

Collaboration de servlets : inclusion

- Inclusion du contenu d'une ressource dans la réponse courante (Jsp, servlet, fichier texte ...)
 - void RequestDispatcher.include(ServletRequest req, ServletResponse res)
- La différence avec une redirection est :
 - la servlet appelante garde le contrôle de la réponse
 - elle peut inclure du contenu avant et après le contenu inclus
- Possibilité de transmettre des informations lors de l'inclusion
 - en utilisant les attributs de requête via la méthode setAttribute(...)

Collaboration de servlets: inclusion

```
public class Regroupage extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML><BODY>");

        RequestDispatcher dispat = req.getRequestDispatcher("/inclus.do");
        dispat.include(req, res); out.println("<BR />");

        req.setAttribute("bonjour", "Bonjour");
        dispat.include(req, res); out.println("<BR />");

        req.setAttribute("bonsoir", "Bonsoir");
        dispat.include(req, res); out.println("<BR />");

        out.println("</BODY></HTML>");
    }
}

public class Inclus extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        if(req.getAttribute("bonjour") != null) {
            out.println(req.getAttribute("bonjour"));
            if (req.getAttribute("bonsoir") != null) {
                out.println(req.getAttribute("bonsoir"));
            } else { out.println("Pas Bonsoir"); }
        } else { out.println("Rien"); }
    }
}
```

Travaux pratiques

RequestDispatcher (Include)

- *LAB_S_12*

La gestion des erreurs

- Il est possible de définir les pages à afficher
 - En fonction d'erreurs http
 - En fonction d'exceptions java



Codes HTTP

- 100 : *Continue : Attente de la suite de la requête*
- 200 : *OK : Requête traitée avec succès*
- 201 : *Created : Requête traitée avec succès avec création d'un document*
- 204 : *No Content : Requête traitée avec succès mais pas d'information à envoyer*
- 303 : *See Other : La réponse à cette requête est ailleurs*
- 400 : *Bad Request : La syntaxe de la requête est erronée*
- 401 : *Unauthorized : Une authentification est nécessaire*
- 403 : *Forbidden : L'authentification est refusée*
- **404 : Not Found : Document non trouvé**
- 414 : *Request-URI Too Long : URI trop longue*
- 500 : *Internal Server Error : Erreur interne du serveur*

web.xml : Accès à des paramètres

```
...
<env-entry>
    <env-entry-name>nom</env-entry-name>
    <env-entry-value>toto</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
...

***  

Context ctx = new InitialContext();
String value = (String) ctx.lookup("java:/comp/env/nom");
***
```

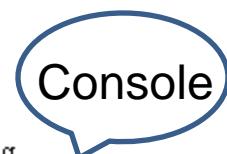
web.xml

- Centralisation de paramètres dans le fichier web.xml accessibles aux servlets de l'application
- Illustration : paramètres d'accès à une base de données

ServletContext: Gestion des traces

- Produire des logs dans Tomcat (<tomcat>/logs/):
 - *void log(String)*
 - *void log(String, Throwable)*

```
public class LogInServlet extends HttpServlet {  
    private ServletContext context;  
  
    public void init(ServletConfig config) throws ServletException {  
        super.init(config);  
        context = getServletContext();  
        context.log("Hello!");  
    }  
    public void doGet(HttpServletRequest req,  
                      HttpServletResponse res) throws IOException {  
        ServletOutputStream out = res.getOutputStream();  
        context.log("Your message has been written. ");  
    }  
}
```



```
Jul 16, 2008 2:14:34 PM org.apache.catalina.core.ApplicationContext log  
INFO: Hello!  
Jul 16, 2008 2:14:34 PM org.apache.catalina.core.ApplicationContext log  
INFO: Your message has been written.  
|
```

QUESTIONS ?

SOMMAIRE

- **PARTIE 1 : Rappel sur le contexte des applications Web d'entreprise**
- **PARTIE 2 : Développement Web en Java**
- **PARTIE 3 : Applications Web et servlets**
- **PARTIE 4 : Présentation des Java Server Pages**
- **PARTIE 5 : Les librairies de balises**
- **PARTIE 6 : Accès aux bases de données**
- **PARTIE 7 : Introduction à Struts**

PARTIE 4: Plan

- **Présentation des objectifs et de l'architecture**
 - Objectifs. Mécanisme de fonctionnement. Exemples de pages JSP.
- **Technique de développement**
 - Les scriplets. Intégration dans la page Web. Directives, déclarations, expressions et actions JSP. Inclusion statique vs dynamique. Versions du langage, syntaxe XML.
- **Utilisation de JavaBeans à partir de page JSP**
 - Définition, création, déploiement et utilisation.
 - Accès et modification à partir d'une page JSP.
- **Développement d'application à l'aide de JSP**
 - Combinaison JSP et servlets. Inclusion d'applets.
 - Accès aux ressources de l'entreprise.

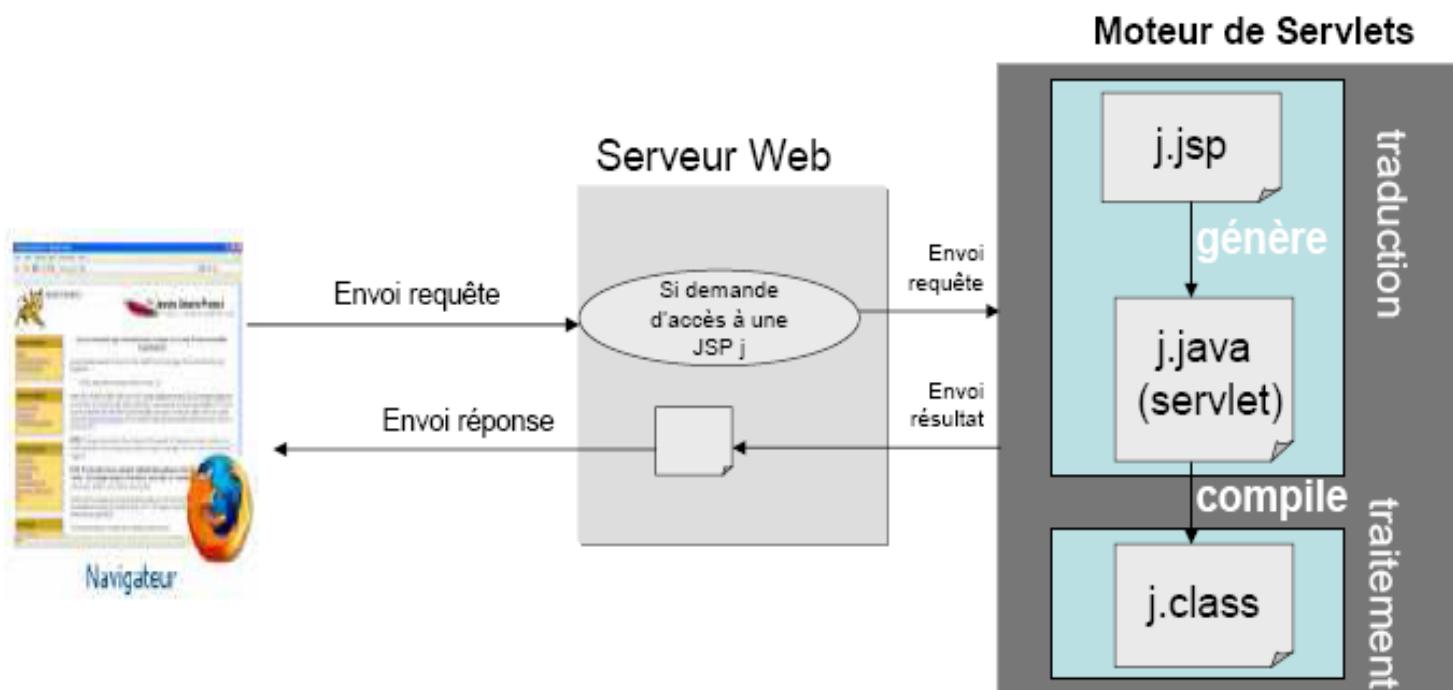
JSP : présentation

- **Servlet:**
 - classe java qui étend javax.servlet.httpServlet
 - Accent mis sur le code java
- **JSP:**
 - **Java Server Pages**
 - Code Java embarqué dans une page HTML entre les balises <% et %>
 - Séparation entre traitement de la requête et génération du flux html

JSP : présentation

- **JSP:**
 - désignés par une URL http://localhost:8080/AppiWeb_JSP/index.jsp
 - fragments de code Java exécutés sur le moteur de Servlets
 - pages JSP sont converties en Servlet par le moteur de Servlets lors du premier appel à la JSP

JSP : fonctionnement



JSP vs Servlet : Exemple

```
public class welcome extends HttpServlet {  
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
  
        out.println("<html>");  
        out.println("<head>");  
        out.println("</head>");  
        out.println("<body>");  
        out.println("<h3>Appli Web</h3>");  
        out.println("Intitulé du cours : JSP<br>");  
        out.println("Date du cours : " + new java.util.Date().toString());  
        out.println("</body>");  
        out.println("</html>");  
        out.close();  
    }  
}
```

Servlet welcome.java

JSP vs Servlet : Exemple

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <h3>Appli Web</h3>
    Intitulé du cours : JSP<br>
    Date du cours : <%= new java.util.Date().toString()%>
  </body>
</html>
```

Page JSP welcome.jsp

JSP vs Servlet : Exemple

```
public final class welcome_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
    public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        try {
            ....
            response.setContentType("text/html;charset=UTF-8");
            pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
            out = pageContext.getOut();
            out.write("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">\n");
            out.write("  \"http://www.w3.org/TR/html4/loose.dtd\"\n");
            out.write("<html>\n");
            out.write("  <head>\n");
            out.write("    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\n");
            out.write("  </head>\n");
            out.write("  <body>\n");
            out.write("    <h3>Appli Web</h3>\n");
            out.write("    Intitulé du cours : JSP<br>\n");
            out.write("    Date du cours : ");
            out.print(new java.util.Date().toString());
            out.write("\n");
            out.write("  </body>\n");
            out.write("</html>\n");
        } catch (Throwable t) {
```

Servlet générée
welcome_jsp.java

JSP : cycle de vie

- Identique au cycle de vie d'une Servlet:
 - Appel de la méthode jsplInit() après le chargement de la page
 - Appel de la méthode _jspService() à chaque requête
 - Appel de la méthode jspDestroy() lors du déchargement
- Rq: Il est possible de redéfinir dans la JSP les méthodes jsplInit() et jspDestroy()

JSP : exemple (modification de jsplnIt())

```
<%@ page import="java.util.Date" %>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <%!
            int mon_compteur;
            Date ma_date;
            public void jspInit() {
                ma_date = new Date();
                mon_compteur=0;
            }
        %>
        <h3>Test : Modification de la méthode jspInit() </h3>
        Date de la première visite: <%= ma_date %><br>
        Nombre de visites : <%= ++mon_compteur %>
    </body>
</html>
```

JSP : éléments du code

- Page JSP:
 - Html: structure statique de la page
 - Code JSP: **éléments dynamiques** de la page
- **4 types d'éléments**

JSP: éléments de script

- 4 types d'éléments de script:
 - Les directives : indiquent à la page les informations globales (par exemple les instructions d'importations)
 - Les déclarations : destinées à la déclaration de méthodes et de variables à l'échelle d'une page
 - Les scriplets : code Java intégré dans la page
 - Les expressions : sous forme de chaîne, en vue de leur insertion dans la sortie de la page

JSP : les directives

<%@.....%>

- **Les directives de jsp 1.2:**

- Page : informations relatives à la page
- Include : fichiers à inclure littéralement
- Taglib : URI d'une bibliothèque de balises utilisée dans

```
<%@ page
[language="java"] [extends="package.class"]
[import="{package.class|package.*}, ..."] [session="true|false"]
[buffer="none|8kb|sizekb"] [autoflush="true|false"]
[contentType="mimeType [charset=characterSet]" |
    "text/html; charset=ISO-8859-17"]
[iserrorPage="true|false"]
%>
```

JSP : les directives de page

- Définir les "import" nécessaires au code Java de la JSP
 - `<%@ page import="java.io.*"%>`
- Définir le type MIME du contenu retourné par la JSP
 - `<%@ page contentType="text/html"%>`
- Fournir l'URL de la JSP à charger en cas d'erreur
 - `<%@ page errorPage="err.jsp"%>`
- Définir si la JSP est une page invoquée en cas d'erreur
 - `<%@ page isErrorPage="true" %>`
- Déclarer si la JSP peut être exécutée par plusieurs clients à la fois
 - `<%@ page isThreadSafe="false" %>`

JSP : les directives de page

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@ page errorPage="ErrorDiv.jsp" %>
<html>
    <head>
        <title>Exemple @ errorPage</title>
    </head>
    <body>

        <h3>Division de 2 nombres au hasard compris entre 0 et 5</h3>
        <int denom = (int)(Math.random() * 5);
         int numer = (int)(Math.random() * 5);%>
        Le résultat de la division de
        <%=numer%> par <%=denom%> est : <%=numer/denom%>
    </body>
</html>
```

Exp_errorPage.jsp

```
<%@ page isErrorPage="true" %>
<html>
    <head>
    </head>
    <body>
        <h2>Erreur : Division par zéro </h2>
    </body>
</html>
```

ErrorDiv.jsp

http://localhost:8084/AppiWeb_JSP/Exp_errorPage.jsp

Division de 2 nombres au hasard compris entre 0 et 5

Le résultat de la division de 3 par 1 est : 3

http://localhost:8084/AppiWeb_JSP/Exp_errorPage.jsp

Erreur : Division par zéro

JSP : les directives d'inclusion

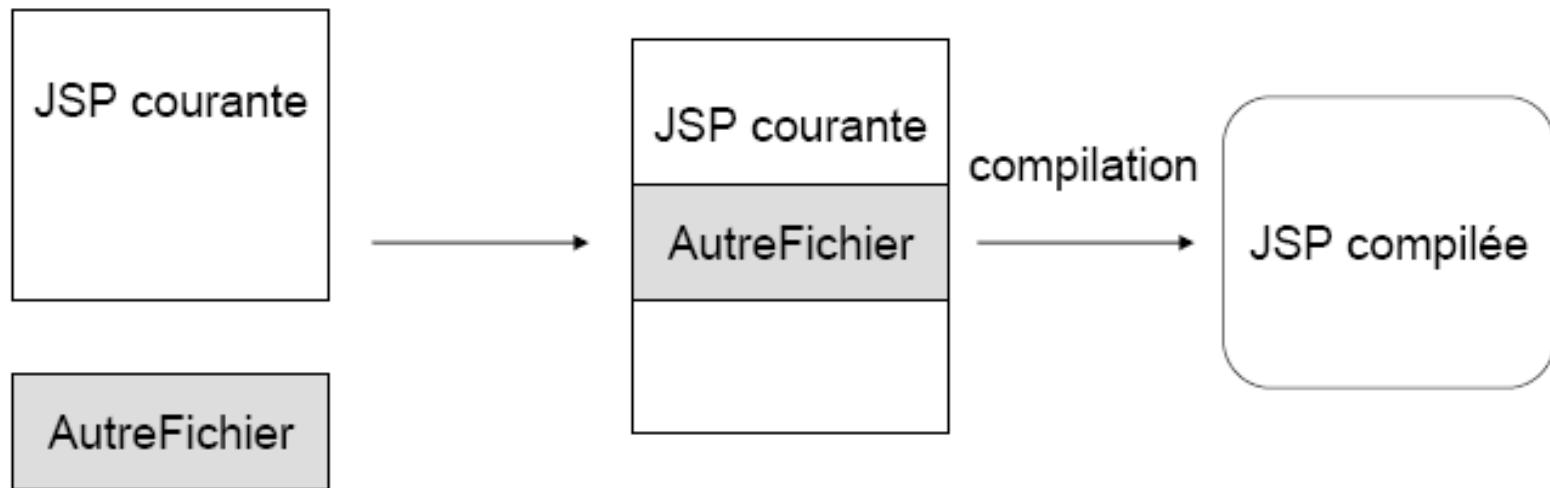
Les directives d'inclusion:

<%@ include%>

- Permettent d'inclure le contenu d'un autre fichier dans la page JSP courante
- Inclusion effectuée avant la compilation de la jsp

JSP : les directive d'inclusion

<%@ include file="AutreFichier"%>



JSP : les directives d'inclusion

```
<html>
    <head>
        <title>Exp @ include</title>
    </head>
    <body>
        <%>
        String le_nom = "Dupont";
        String le_prenom="toto";
        String l_adresse="Nice";
        %>
        <%@ include file="ficheInfo.jsp" %>
    </body>
</html>
```

Exp_include.jsp

```
<h3> Informations sur la personne </h3>
Nom : <%=le_nom%><br>
Prénom : <%=le_prenom%><br>
Adresse : <%=l_adresse%><br>
```

ficheInfo.jsp



JSP : les balises personnalisées

<%@ taglib%>

- Permettent d'indiquer une bibliothèque de balises : adresse et préfixe, pouvant être utilisées dans la page

<%@ taglib prefix="pref" uri="taglib.tld" %>

JSP : les déclarations

<%! %>

- Permettent de déclarer des méthodes et des variables d'instance connus dans toute la page JSP

```
<%!
private int mon_entier;
private int somme(int a, int b) {return (a+b);}
%>
```

JSP : les scriptlets

<%.....%>

- Permettent d'insérer des blocs de code java (*qui seront placés dans `_jspService(...)`*)

```
<% int som=0;  
    for (int i=1; i< 15; i++)  
        som=somme(som,i);%>
```

JSP : les scriplets

Donnent accès à une liste d'objets implicites à partir de l'environnement de la servlet :

- request : requête du client (classe dérivée de HttpServletRequest)
- response : réponse de la page JSP (classe dérivée de HttpServletResponse)
- session : session HTTP correspondant à la requête
- out : objet représentant le flot de sortie
- Etc.

JSP : les expressions

<%=.....%>

- Permettent d'évaluer une expression et renvoyer sa valeur (string)
- Correspond à *out.println(...);*

Nous sommes le : <%=new java.util.Date()%>

JSP : les commentaires

<%--.....--%>

- Permettent d'insérer des commentaires (*qui ont l'avantage de ne pas être visibles pour l'utilisateur*)

```
<%-- ceci est un commentaire --%>
```

Travaux pratiques

JSP Calculateur

- *LAB_J_1*

JSP : les éléments d'action

- permettent de faire des traitements au moment où la page est demandée par le client
 - utiliser des Java Beans
 - inclure dynamiquement un fichier
 - rediriger vers une autre page
- Constitués de balises pouvant être intégrées dans une page jsp (syntaxe XML)

`<jsp:/>`

JSP : les éléments d'action

- Actions jsp standards:
 - jsp:include et jsp:param
 - jsp:forward
 - jsp:useBean
 - jsp:setProperty et jsp:getProperty

JSP : include/param

- `jsp:include` : identique à la directive `<%@ include ...` sauf que l'inclusion est faite au moment de la requête
 - Donc après compilation...
- `jsp:param` : permet de passer des informations à la ressource à inclure

JSP : include/param

jsp:include et jsp:param

```
<jsp:include page="ficheInfo.jsp" flush="true">
    <jsp:param name="le_nom" value="Dupont"/>
    <jsp:param name="le_prenom" value="toto"/>
    <jsp:param name="l_adresse" value="Nice"/>
</jsp:include>
```

JSP : forward

- Permet de passer le contrôle de la requête à une autre ressource
- `jsp:param` permet ici aussi de passer des informations à la ressource de redirection

```
<jsp:forward page="Redirect.jsp">
    <jsp:param name="monParam" value="maValeur"/>
</jsp:forward>
```

Travaux pratiques

JSP Include

- *LAB_J_2*

JSP : éléments de script-objets implicites

- **request** : requête courante (HttpServletRequest)
- **response** : réponse courante (HttpServletResponse)
- **out** : flot de sortie permet l'écriture sur la réponse
- **session** : session courante (HttpSession)
- **application** : espace de données partagé entre toutes les JSP (ServletContext)
- **page** : l'instance de servlet associée à la JSP courante (this)

Travaux pratiques

JSP Implicit Object

- *LAB_J_3*

JSP : useBean

- Permet de séparer la partie traitement de la partie présentation
- Permet d'instancier un composant JavaBean (classe java) qui pourra être appelé dans la page JSP

```
<jsp:useBean id="testBean" class="exemplesjsp.MonBean"/>
```

JSP : useBean-JavaBean

- Permet de coder la logique métier de l'application WEB
- L'état d'un Bean est décrit par des attributs appelés propriétés

JSP : useBean-JavaBean

- classe Java respectant un ensemble de directives
 - Un constructeur public sans argument
 - Des propriétés « prop » accessibles au travers de getteurs et setteurs: getProp (lecture) et setProp (écriture) portant le nom de la propriété

JSP : useBean-JavaBean

type getNomDeLaPropriété()

- ⇒ pas de paramètre et son type est celui de la propriété

void setNomDeLaPropriété(type)

- ⇒ un seul argument du type de la propriété et son type de retour est void

JSP : useBean-JavaBean

Java Bean: Exemple

- Utilise le constructeur par défaut ne possédant aucun paramètre

```
| import java.util.Date;
| public class BeanPersonne {
|
|     private String leNom,lePrenom;
|     private Date dateNaiss;
|
|     public String getLeNom()
|     {return leNom;}
|     public void setLeNom(String leNom)
|     {this.leNom=leNom;}
|
|     public String getLePrenom()
|     {return lePrenom;}
|     public void setLePrenom(String lePrenom)
|     {this.lePrenom=lePrenom;}
|
|     public Date getDateNaiss()
|     {return dateNaiss;}
|     public void setDateNaiss(Date dateNaiss)
|     {this.dateNaiss=dateNaiss;}
| }
```

JSP : useBean

```
<jsp:useBean id="personne" class="mesBeans.BeanPersonne" scope="session"/>
```

Nom de l'instance

package.class du bean

Champ d'existence de
l'objet Bean:

- request
- page
- session
- application

JSP : get/setproperty

jsp:setProperty et jsp:getProperty

- Permet de récupérer ou de modifier les valeurs d'une instance de JavaBean

- **Récupération :**

```
<jsp:getProperty name= "testBean" property= "maProp" />
```

Équivalent à: <*=testBean.getMaProp ()%>

JSP : get/setproperty

jsp:setProperty et jsp:getProperty

- Modification :

- Attribuer automatiquement aux attributs les valeurs récupérés de la requête

```
<jsp:setProperty name= "testBean" property= "*" />
```

- Attribuer directement une valeur de paramètre à un attribut

```
<jsp:setProperty name= "testBean" property= "maProp" param="monParam"/>
```

- Attribuer directement une valeur à un attribut

```
<jsp:setProperty name= "testBean" property= "maProp" value="3"/>
```

JSP : get/setproperty

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<jsp:useBean id="personne" class="beans.BeanPersonne" scope="session"/>
<jsp:setProperty name="personne" property="leNom" value="Martin"/>
<jsp:setProperty name="personne" property="lePrenom" value="toto"/>
<jsp:setProperty name="personne" property="dateNaiss" value="<%=new java.util.Date()%"/>
<html>
    <head>
        <title>Fiche Personne</title>
    </head>
    <body>
        nom: <jsp:getProperty name="personne" property="leNom"/><br>
        prénom: <jsp:getProperty name="personne" property="lePrenom"/> <br>
        date de naissance: <jsp:getProperty name="personne" property="dateNaiss"/>
    </body>
</html>
```

nom: Martin
prénom: toto
date de naissance: Mon Mar 31 10:21:43 CEST 2008

Travaux pratiques

JSP et JAVABEAN

- *LAB_J_4*

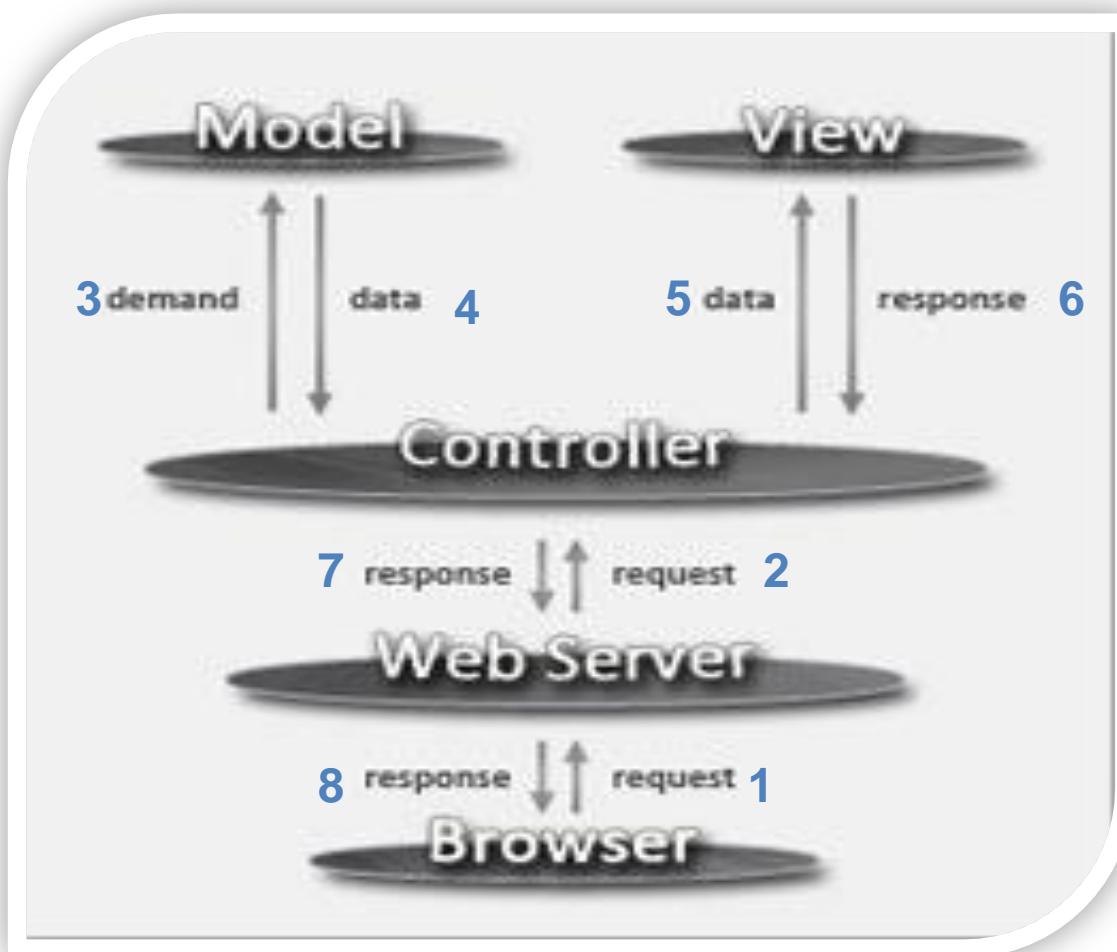
Combinaison Servlet et JSP

Transmission de requête vers une jsp ou inclusion de la jsp dans la servlet : Utilisation de RequestDispatcher

```
RequestDispatcher MyJspDispat =  
getServletContext().getRequestDispatcher("/folder/page.jsp");
```

- MyJspDispat.include(req,res) : la ressource est insérée dans la servlet active.
- MyJspDispat.forward(req,res) : le flux est complètement redirigé vers la nouvelle ressource (l'appelant ne peut plus effectuer de sortie au client, cette tâche est transmise à la nouvelle ressource uniquement)

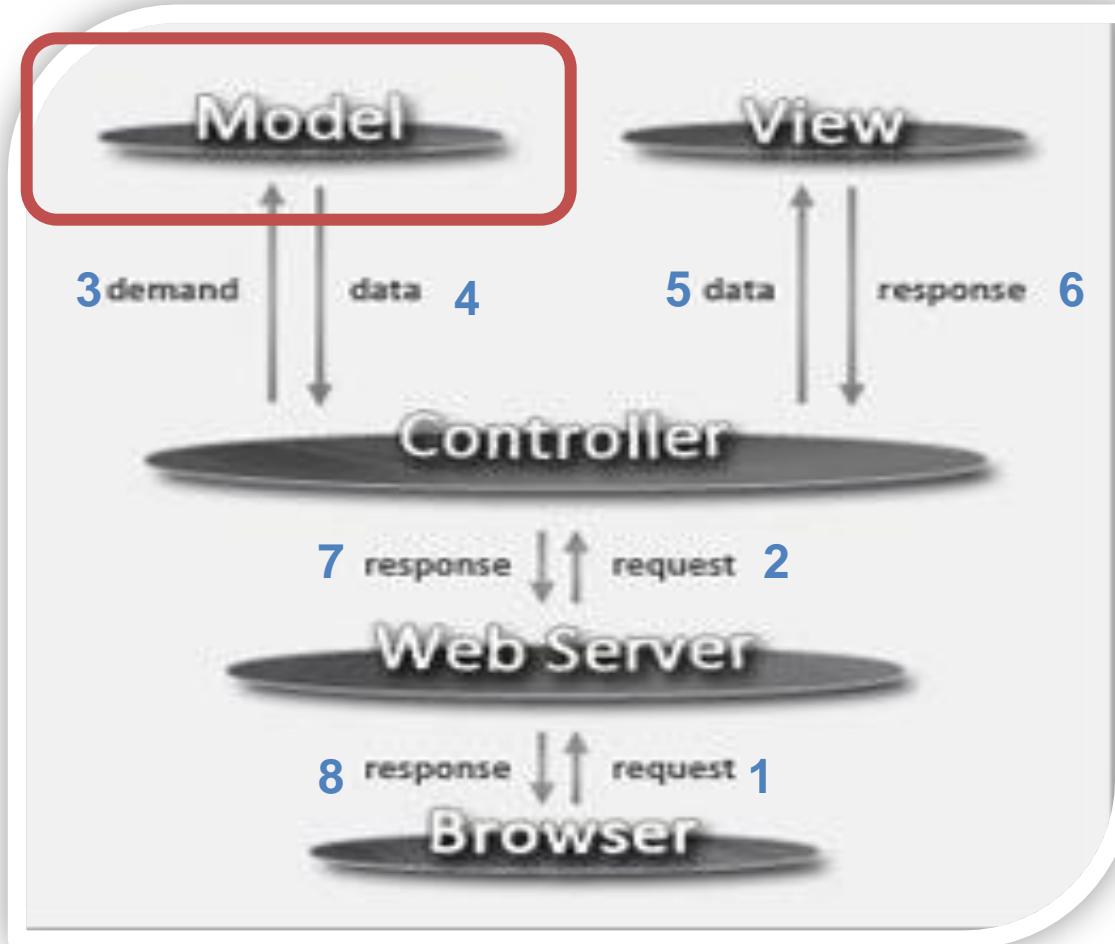
MVC : WEB



MVC WEB: Fonctionnement

1. Le client agit sur la vue.
2. Le contrôleur intercepte l'action.
3. Le contrôleur traite la demande en utilisant le modèle.
4. Le contrôleur reçoit une réponse du modèle et effectue les actions associées.
5. Le contrôleur sélectionne une vue pour présenter les résultats (mise à jour si besoin).
6. {
7. } La vue est renvoyée au client
8. }

LE MODÈLE



LE MODÈLE: RÔLE

- Encapsule les propriétés d'une donnée
- Être indépendant des vues et contrôleurs
- Disposer de méthodes pour récupérer ou mettre à jour les données
 - insertion,
 - suppression,
 - modification.

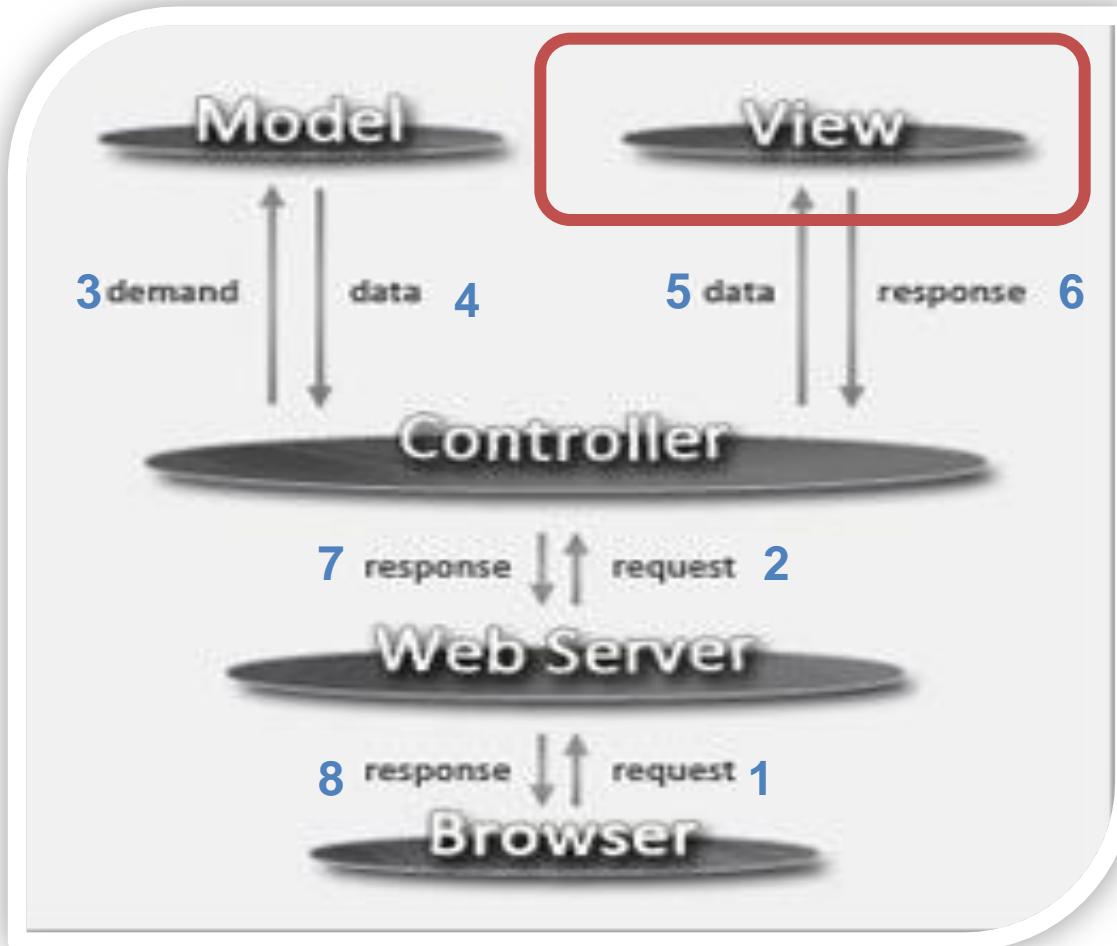
LE MODÈLE: RÔLE

- Autoriser éventuellement des vues partielles.
 - Exemple : méthodes faisant des select sur une base de données.
- Gestion des données de l'application, garantie de leur intégrité.
 - Exemple : concurrence en cas d'accès simultanés sur une BD ou des fichiers.

MODÈLE = DONNÉES + TRAITEMENTS

- Les résultats renvoyés ne sont pas mis en forme :
 - Retour XML, Json, objet Java, etc.
 - Pas de HTML
- Cas typique : base de données + méthodes SQL + classes Java
 - Exemple : classe panier sur un site de vente en ligne et stockage du panier en BD.

LA VUE



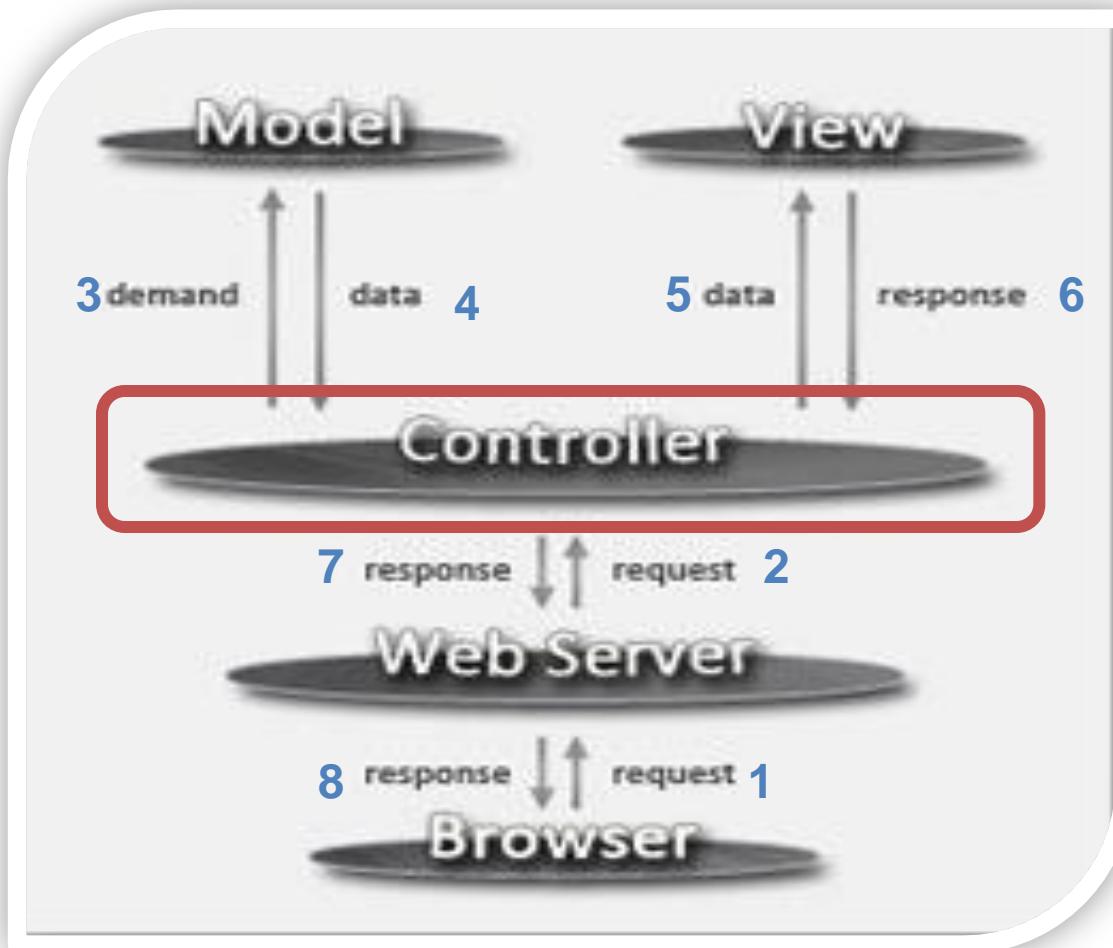
LA VUE

- La vue fait l'interface avec l'utilisateur.
- Sa première tâche est d'afficher les données qu'elle a récupérées auprès du modèle.
- Sa seconde tâche est de recevoir tous les actions de l'utilisateur
 - clic de souris,
 - sélection d'une entrées,
 - boutons, ...

LA VUE

- Ses différents événements sont envoyés au contrôleur.
- La vue peut aussi donner plusieurs vues, partielles ou non, des mêmes données.
- La vue peut aussi offrir la possibilité à l'utilisateur de changer de vue.
- La vue contient le code HTML. C'est la seule partie qui doit en contenir.

LE CONTRÔLEUR



LE CONTRÔLEUR

- Rôle
 - Le contrôleur est chargé de la synchronisation du modèle et de la vue.
 - Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer.
 - Le contrôleur est souvent scindé en plusieurs parties dont chacune reçoit les événements d'une partie des composants.

LE CONTRÔLEUR

- Le contrôleur n'effectue aucun traitement, ne modifie aucune donnée, ne fait aucun affichage.
- Le contrôleur doit donc :
 1. Analyser la requête, soit extraire les informations dans l'URL
 2. Faire une mise à jour du modèle si nécessaire, en lui passant des paramètres
 3. Déterminer la vue à afficher et demander son affichage

MVC: AVANTAGES

- Le modèle est totalement indépendant de la vue.
 - Possibilité de changer la partie IHM.
- Le modèle étant totalement autonome, il peut être modifié beaucoup plus facilement.
 - Résiste aux évolutions des règles métier et/ou au changement du mode de persistance des données.
 - La vue n'a pas besoin d'être modifiée dans ce cas.

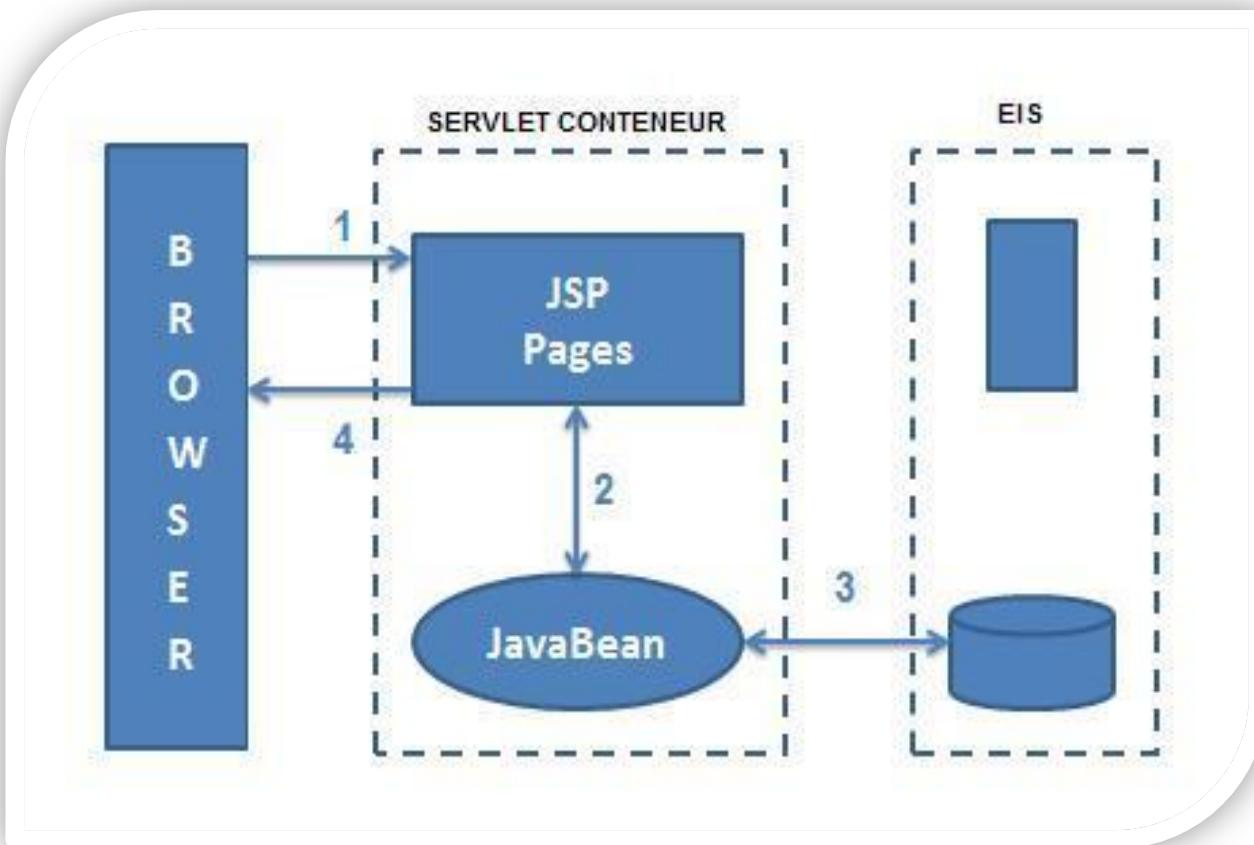
MVC: AVANTAGES

- Plusieurs vues peuvent utiliser le même modèle.
 - Gain en coût de développement important.
- Deux équipes peuvent travailler en parallèle.
 - Une équipe d'infographistes peut travailler sur les vues.
 - Une équipe de développeurs peut travailler sur le modèle et le contrôleur.
- Les trois couches doivent être réellement indépendantes et ne doivent communiquer que par des interfaces.

MVC: SERVLET/JSP

- Pour construire notre application en respectant le MVC deux solutions s'offre à nous:
 - Une architecture centrée sur des pages JSP
 - Une architecture centrée sur des SERVLET

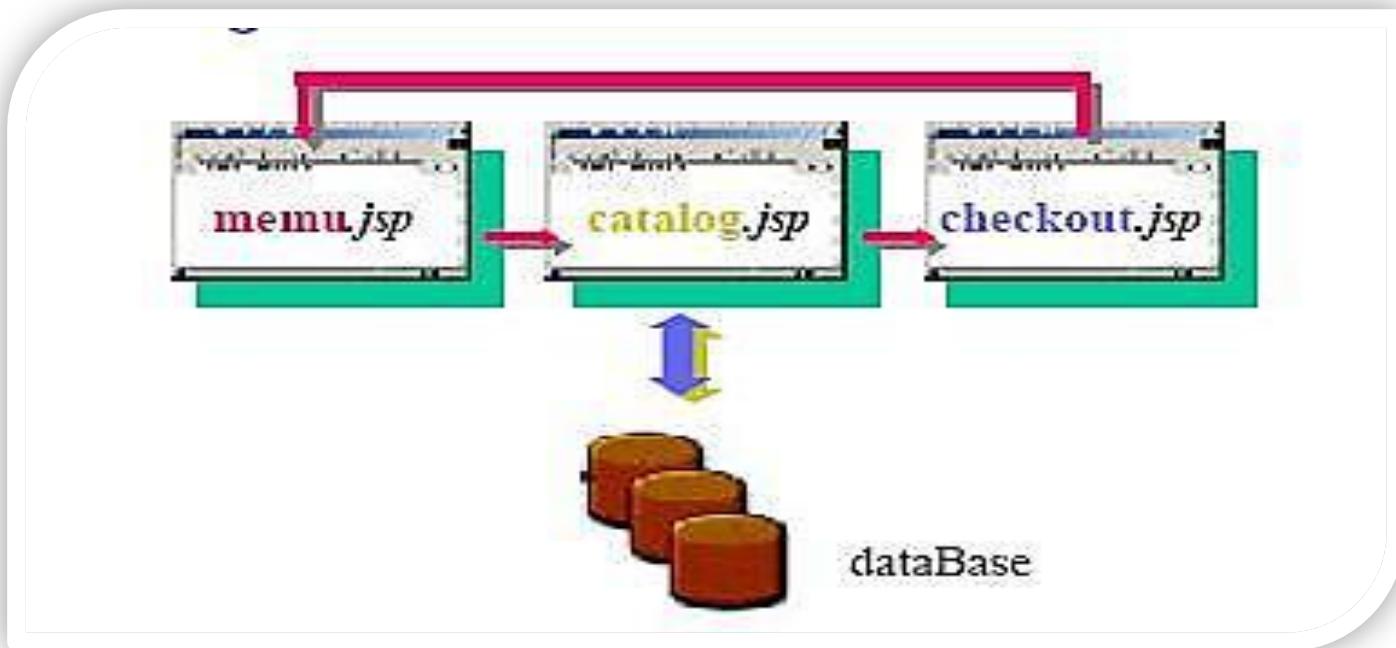
ARCHITECTURECENTRÉE JSP



ARCHITECTURE CENTRÉE JSP

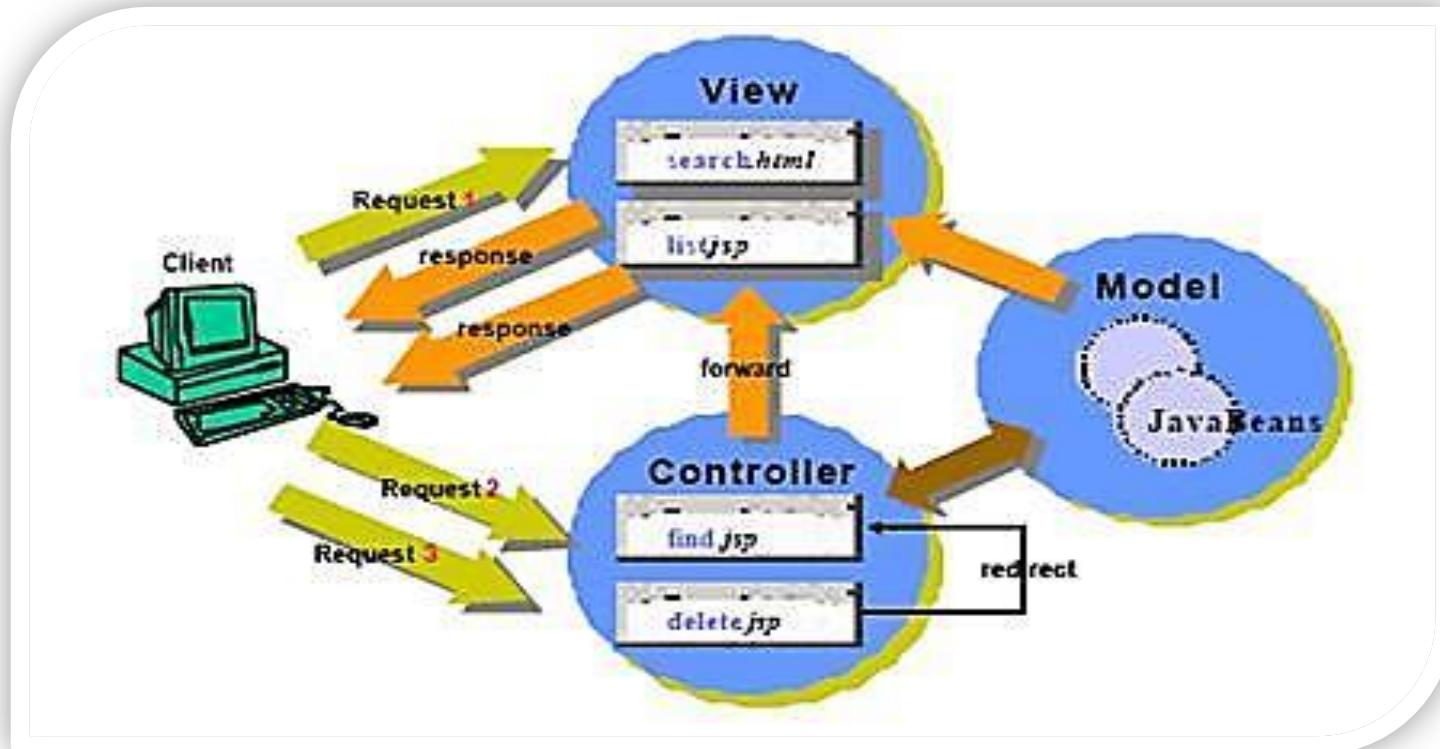
- Très facile à mettre en œuvre, ce modèle ne permet pas la construction de grosses applications web, la logique étant mélangée dans toutes les pages JSP.
- Les tâches complexes sont difficilement implantées dans ce modèle qui tend a produire du code difficile à étendre et à maintenir.
- Convient pour faire rapidement un prototype de notre application web.

ARCHITECTURE CENTRÉE JSP



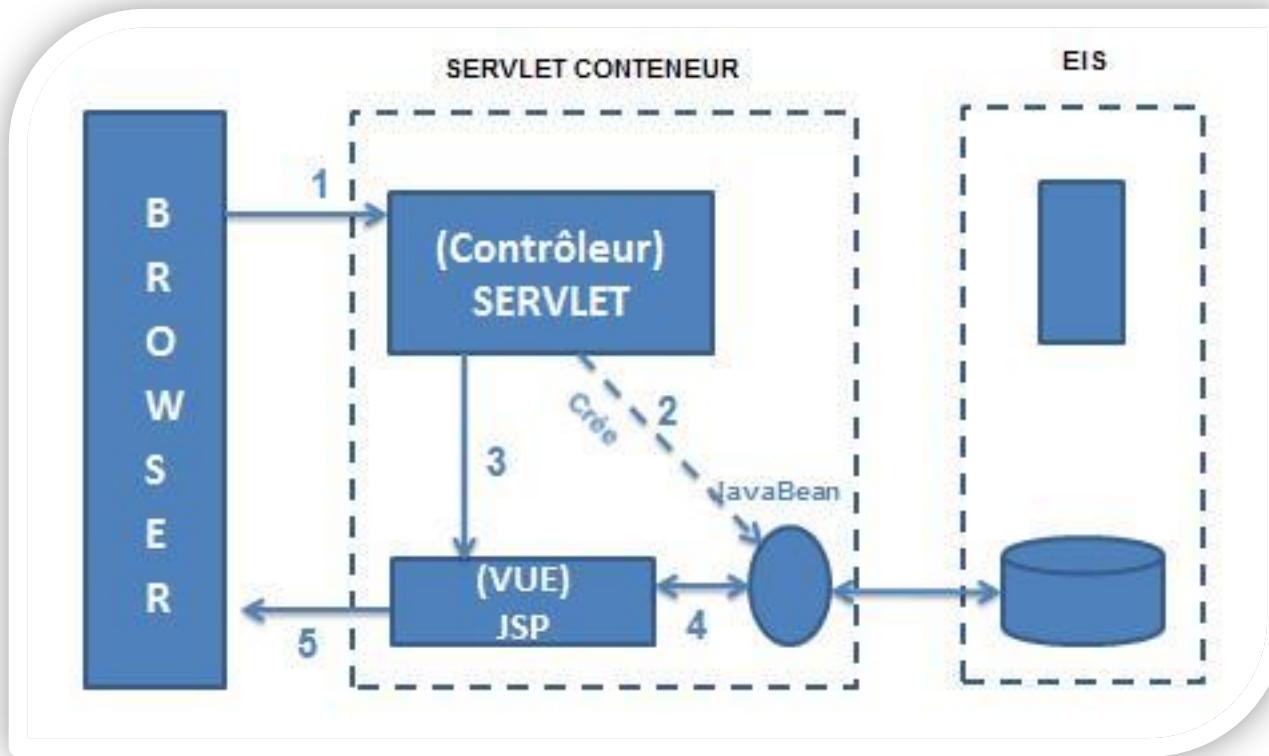
- Le contrôleur est distribuer sur plusieurs pages.
- Les grosses pages JSP sont difficilement réutilisables.
- Le code généré n'est pas modulable.
- Une mise à jour met en branle l'ensemble du projet plutôt qu'un seul point précis.

ARCHITECTURE JSP Scénario

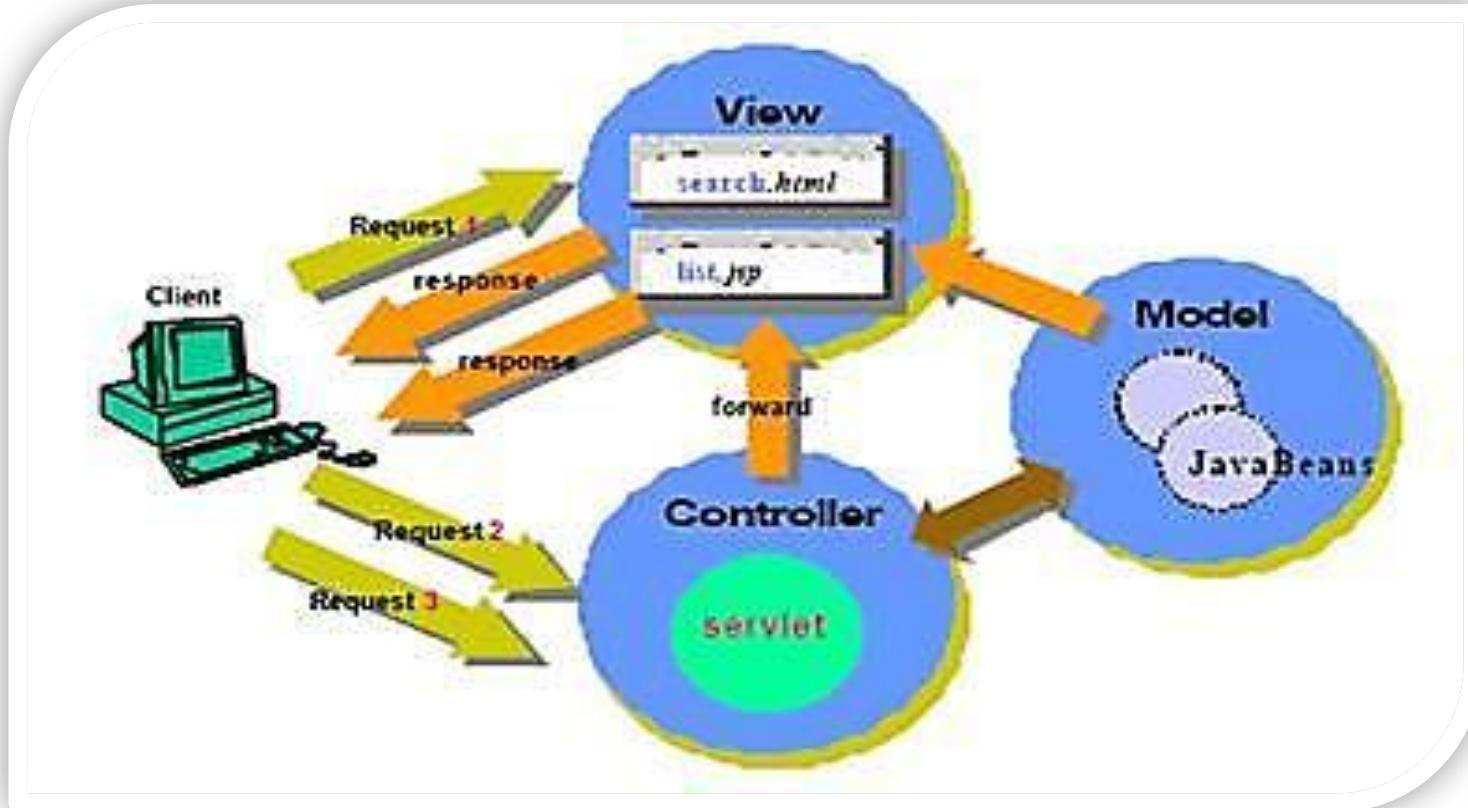


- Des JSP jouent des rôles de contrôleurs.
- Jsp:forward permet de chainer les requêtes pour invoquer une autre page.
- Ex: <jsp:forward page="AutrePage.jsp"/>

ARCHITECTURE CENTRÉE SERVLET



ARCHITECTURE CENTRÉE SERVLET Scénario



ARCHITECTURE CENTRÉE SERVLET: Code

- Syntaxe dans la servlet pour lancer la JSP

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)

    String url="/jsp/shopping/EShop.jsp";
    ServletContext sc = getServletContext(); // Héritée de GenericServlet
    RequestDispatcher rd = sc.getRequestDispatcher(url);
    rd.forward(request, response);

}
```

- L' élément central est le RequestDispatcher

ARCHITECTURE CENTRÉE SERVLET: Code

- La servlet peut passer des valeurs à la JSP appelé grâce à `setAttribute()`.

```
//appelle les méthodes sur les objets métiers
List<Produit> produits= objetMetier.getAllProduit();

//ajoute à la requête
request.setAttribute("resultat",produits);
```

- La JSP extrait les objets de `request` grâce à `getAttribute()`.

```
<%>
List<Produit> resultat=(List<Produit>) request.getAttribute("resultat");
%>
```

Limites du modèle MVC

- Le MVC se révèle trop complexe pour de petites applications.
- Le temps accordé à l'architecture peut ne pas être rentable pour le projet.
- Pages plus lentes à afficher (hors cache).
- Plus de ressources consommées.

Travaux pratiques

MVC

- *LAB_J_5*
- *LAB_J_6*

QUESTIONS ?

SOMMAIRE

- **PARTIE 1 : Rappel sur le contexte des applications Web d'entreprise**
- **PARTIE 2 : Développement Web en Java**
- **PARTIE 3 : Applications Web et servlets**
- **PARTIE 4 : Présentation des Java Server Pages**
- **PARTIE 5 : Les librairies de balises**
- **PARTIE 6 : Accès aux bases de données**
- **PARTIE 7 : Introduction à Struts**

Généralités

- Dans les pages JSP, on remarque qu'il y a mélange entre le code Java et le code HTML
- Le développeur a tendance à mettre trop de codes dans un document qui ne devrait que fournir de l'affichage (HTML)
- Difficile à maintenir des pages JSP par un non programmeur
- Il faut donc fournir des outils qui encapsulent le code Java
 - Cacher le code Java dans des balises personnalisées

Qu'est-ce qu'un tag JSP

- Un tag JSP est une simple balise XML associée à une classe Java
- A la compilation d'une JSP, les balises sont remplacées par le résultat des classes Java
- Exemple:

Balise ouvrante

```
<jsp:useBean id="monBean" class="Package.MonObjet" scope="attribut" >
    <jsp:setProperty name="monBean" property="*" />
</jsp:useBean>

<jsp:forward page="/page.jsp" />
    <jsp:param name="defaultparam" value="nouvelle" />
</jsp:forward>
```

Balise fermante

Attributs présents

Qu'est-ce qu'un tag JSP

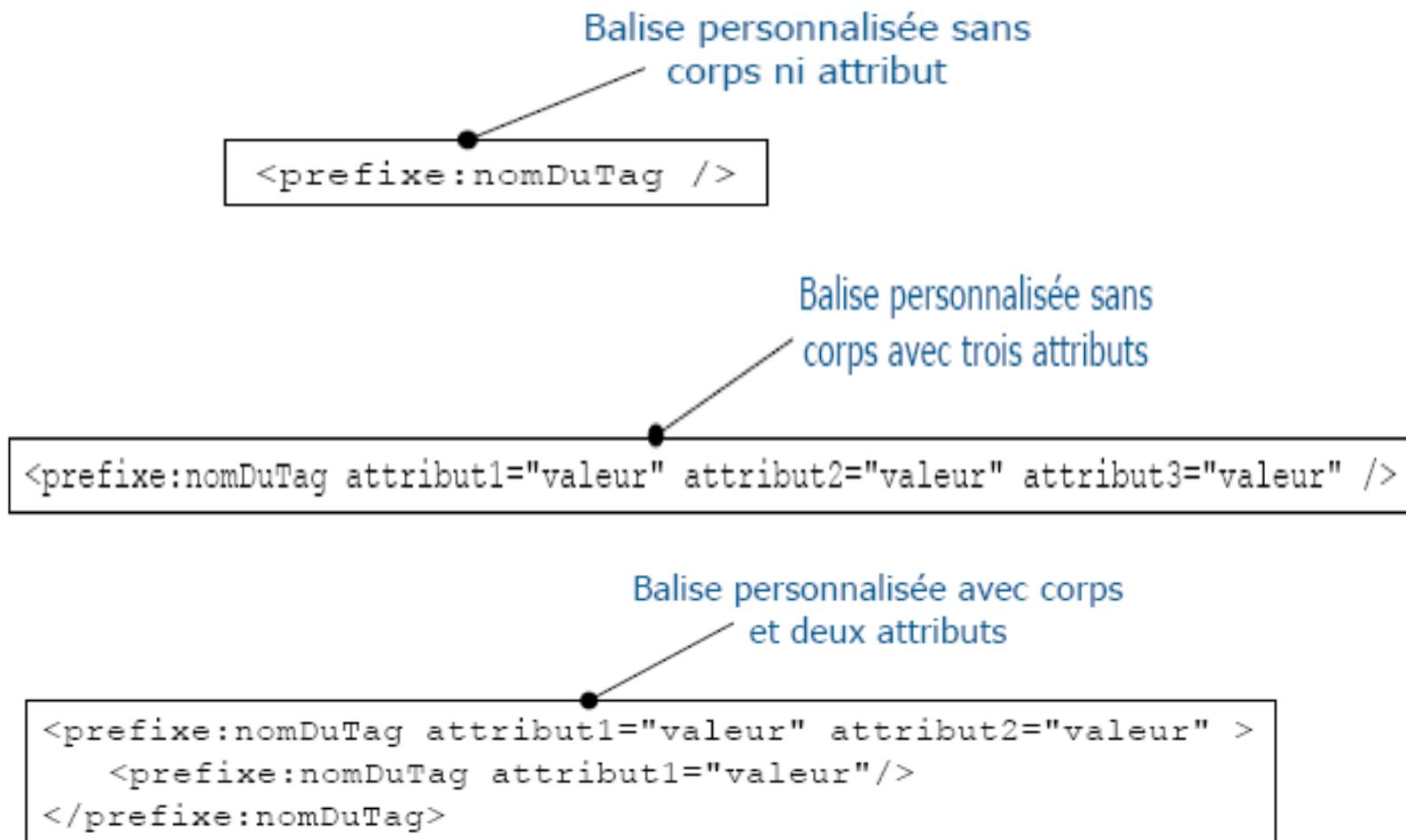
- La syntaxe d'un tag JSP est variable selon s'il dispose d'attributs et/ou d'un corps

```
<prefix:nomDuTag attribut1="valeur" attribut2="valeur" >  
    Corps du Tag  
</prefix:nomDuTag>
```

- Nous retrouvons les éléments suivants
 - préfixe : permet de distinguer les différents tags utilisés
 - nomDuTag : identifie le nom du tag de la librairie « préfixe »
 - Un certain nombre de couples d'attribut/valeur (peut-être au nombre de zéro)
 - Un corps (peut ne pas exister)

Qu'est-ce qu'un tag JSP

Exemple : différentes formes de balises JSP (syntaxe XML)



Qu'est-ce qu'un tag JSP

Un tag personnalisé est **composé de trois éléments**

- Tag Library Descriptor (TLD) ou description de la bibliothèque de balises effectue le mapping entre les balises et les classes Java (obligatoire)
 - Fichier de type XML
 - Le format porte obligatoirement l'extension « tld »
- Une classe appelée « handler » pour chaque balise qui compose la bibliothèque (obligatoire)
- Une classe permettant de fournir des informations supplémentaires sur la balise personnalisée au moment de la compilation de la JSP (facultatif)

Utilisation dans une page JSP

- Pour chaque bibliothèque de balise à utiliser dans une JSP, il faut la déclarer en utilisant la directive taglib
 - uri : l'URI de la description de la bibliothèque (fichier *.tld)
 - prefix : espace de noms pour les tags de la bibliothèque dans la JSP

```
<%@ taglib uri="/taglib/mytag.tld" prefix="myprefix" %>
```

- Deux façons de définir l'uri
 - directe (le nom du fichier TLD avec son chemin relatif)

```
<%@ taglib uri="/taglib/mytag.tld" prefix="myprefix" %>
```

- indirecte (correspondance avec le fichier de description du contexte web.xml)

```
<%@ taglib uri="myTagLib" prefix="myprefix" %>
```

```
<taglib>
  <taglib-uri>myTagLib</taglib-uri>
  <taglib-location>/taglib/mytag.tld</taglib-location>
</taglib>
```

Ajout dans le fichier
web.xml

JSTL: Introduction

- **JSTL** : Java server page Standard Tag Library.
 - <http://jstl.java.net/>
- Ensemble standard d'actions personnalisées (Custom Tags) développé par la JSR (Java Specification Request) 052
- Propose fonctionnalités souvent rencontrées dans les JSP :
 - *Tags de structure (itération, conditionnement ...)*
 - *Internationalisation*
 - *Exécution de requêtes SQL*
 - *Utilisation de documents XML*

Introduction

- Le but de JSTL est de simplifier le travail des auteurs de page JSP.
- JSTL permet de développer des pages JSP en utilisant des balises XML.
- JSTL se base sur l'utilisation des EL en remplacement des scriptlets Java.
- Il existe différentes versions des JSTL(1.0, 1.1, 1.2). La plus récente se base sur les JSP 2.0 qui intègre un moteur d'EL.

Langage d'expression (EL)

En plus JSTL propose un langage d'expression (EL) permettant de référencer facilement les objets java accessibles dans le contexte de la JSP.

- Depuis la version 2.0 des JSP, il est possible de placer à n'importe quel endroit d'une page JSP des expressions qui sont évaluées et remplacées par le résultat de leur évaluation : les Expressions Languages(EL) .

Utilisation

- Pour utiliser les JSTL, il suffit de copier dans le répertoire WEB-INF/lib de votre application WEB les fichiers **jstl-api-1.2.jar** et **jstl-impl-1.2.jar** qui se trouvent dans le répertoire lib de la distribution JSTL.
- Les JSTL disposent d'un ensemble de librairies de fonctionnalités différentes
- Pour inclure une librairie jstl à une page JSP il faut ajouter une déclaration en tête de page (directive taglib).

Utilisation

- Déclaration en tête de page (directive taglib) de cette façon:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

- Cette ligne permet d'inclure la librairie "core" de la JSTL qui contient les actions de base d'une application web.

Utilisation

Librairie	URI	Préfixe
Core	http://java.sun.com/jsp/jstl/core	c
Format	http://java.sun.com/jsp/jstl/fmt	fmt
XML	http://java.sun.com/jsp/jstl/xml	x
SQL	http://java.sun.com/jsp/jstl/sql	sql
Fonctions	http://java.sun.com/jsp/jstl/functions	fn

Travaux pratiques

- TP: Installation bibliothèque balises JSTL

Langage d'expression (EL)

- **Objectif**
 - *fournir un moyen simple d'accéder aux données nécessaires à une JSP en s'affranchissant de la syntaxe Java*
- **EL**
 - permet d'utiliser et de faire référence à des objets java accessible dans les différents contextes (page, requete, session ou application) d'une JSP
- **Syntaxe**
 - \${**xxx**} où **xxx** est le nom désignant un objet java défini dans un contexte particulier

Langage d'expression (EL)

- La chaîne **XXX** correspond à l'expression à interpréter.
- Une expression peut être composée de plusieurs termes séparés par des opérateurs.
- Ex :
 - \${ (10 + 2)*2 } => 24
 - \${ a && b } => true/false

Langage d'expression (EL)

Les différents **termes** peuvent être :

- Un type primaire (null, Long, Double, String, Boolean, ...)
- Un objet implicite (requestScope, sessionScope, param, paramValues, header, headerValues, ...)
- Un attribut d'un des scopes de l'application. Cet attribut pouvant être stocké dans n'importe quel scope (page, request, session ou application). L'EL cherchera dans tous les scopes dans cet ordre.

On a donc :

`${name} ⇔ <%= pageContext.findAttribute("name"); %>`

- Une fonction EL (cf JSTL)

Exemple

`${sessionScope.utilisateur.nom}`

accès à la propriété nom de l'objet utilisateur situé dans la session

équivaut à

```
<%=session.getAttribute("utilisateur").getNom()%>
```

Langage d'expression (EL)

Les **EL** apportent trois avantages :

- Une meilleure **lisibilité** : le code se limite quasiment au nom du bean et de sa propriété.
- Pas de **cast**, et de ce fait pas d'import (on accède aux propriétés par réflexion).
- Gestion des valeurs **null** (Voir la section "Gestion des exceptions").

Variables implicites définies par EL

- **pageContext** : Accès à l'objet **PageContext** de la page JSP.
- **pageScope** : Map permettant d'accéder aux différents attributs du scope '**page**'.
- **requestScope** : Map permettant d'accéder aux différents attributs du scope '**request**'.
- **sessionScope** : Map permettant d'accéder aux différents attributs du scope '**session**'.
- **applicationScope** : Map permettant d'accéder aux différents attributs du scope '**application**'.
- **param** : Map permettant d'accéder aux paramètres de la requête HTTP sous forme de **String**.

Variables implicites définies par EL

- **paramValues** : Map permettant d'accéder aux paramètres de la requête HTTP sous forme de **tableau de String**.
- **header** : Map permettant d'accéder aux valeurs du Header HTTP sous forme de **String**.
- **headerValues** : Map permettant d'accéder aux valeurs du Header HTTP sous forme de **tableau de String**.
- **cookie** : Map permettant d'accéder aux différents Cookies.
- **initParam** : Map permettant d'accéder aux **init-params** du web.xml.

Quelques exemples d'utilisations

- `${ pageContext.response.contentType }` affiche le content type de la réponse.
- `${ pageScope["name"] }` affiche l'attribut "name" du scope page.
- `${ param["page"] }` affiche la valeur du paramètre "page".
 `${ header["user-agent"] }` affiche l'header "user-agent" envoyé par le navigateur.

Opérateurs définis par EL

Opérateurs arithmétiques :		Opérateurs relationnels :	
+	Addition	<code>==</code> ou <code>eq</code>	Egalité
-	Soustraction	<code>!=</code> ou <code>ne</code>	Inégalité
*	Multiplication	<code><</code> ou <code>lt</code>	Plus petit que
/ ou <code>div</code>	Division	<code>></code> ou <code>gt</code>	Plus grand que
% ou <code>mod</code>	Modulo	<code><=</code> ou <code>le</code>	Plus petit ou égal
		<code>>=</code> ou <code>ge</code>	Plus grand ou égal
Opérateurs logiques :		Autres :	
<code>&&</code> ou <code>and</code>	ET logique (<code>true</code> si les deux opérandes valent <code>true</code> , <code>false</code> sinon)	<code>empty</code> [*]	<code>true</code> si l'opérande est <code>null</code> , une chaîne vide, un tableau vide, une Map vide ou une List vide. <code>false</code> sinon.
<code> </code> ou <code>or</code>	OU logique (<code>true</code> si au moins une des deux opérandes vaut <code>true</code> , <code>false</code> sinon)	() [*]	Modifie l'ordre des opérateurs.
<code>!</code> ou <code>not</code> [*]	NON logique (<code>true</code> si l'opérande vaut <code>false</code> , et inversement)	? :	Opérateur conditionnel en ligne, format particulier : <code>condition ? valeur_si_true : valeur_si_false</code>

Accès aux propriétés des objets

- Trois catégories d'**objets** sont définis par les **EL** :
 - Les **beans mappés** (objet de type **Map**)
 - Les **beans indexées** (**tableau Java**, ou objet de type **List**)
 - Les **beans standards** (tout autres objets).

Les propriétés des beans standards

- Il y a deux manière d'accéder aux propriétés d'un bean : en utilisant l'opérateur point (.) ou l'opérateur crochet ([...]).
- Ainsi, pour accéder à la propriété **name** de notre bean, on peut utiliser les syntaxes suivantes :

Opérateur 'point'

```
 ${ bean.name }
```

Opérateur 'crochet'

```
 ${ bean["name"] }  
ou  
 ${ bean['name'] }
```

Les propriétés des beans standards

- Dans les deux cas le résultat sera le même et est équivalent au code suivant :

Code Java

```
<%@ page import="package.MonBean" %>
<%
    MonBean bean = (MonBean) pageContext.findAttribute ("bean");
    if (bean != null)
        out.print ( bean.getName () );
%>
```

Les propriétés indexées (List et tableau)

- Les propriétés indexées permettent d'accéder aux différents éléments d'un tableau ou d'une **List**.
- On peut accéder aux différents éléments en utilisant l'opérateur 'crochet' avec un index, par exemple :

```
 ${ list[0] }  
 ${ list[1] }  
 ${ list["2"] }  
 ${ list["3"] }  
 ${ list[ config.value ] }  
 etc.
```

- On accède alors aux différentes valeurs via la méthode **get(int)** de l'interface **List** .

Les propriétés mappés (Map)

- De la même manière que pour les propriétés indexées, on utilise l'opérateur 'crochet' afin d'accéder aux différentes valeurs d'une **Map**, par exemple :

```
 ${ map["clef1"] }
 ${ map['clef2'] }
 ${ map[ config.key ] }
 etc.
```

Gestion des exceptions

- Les **EL** gèrent un certains nombres d'exceptions afin de ne pas avoir à les traiter dans les JSP.
- **NullPointeurException** (et les valeurs **null**) :
 - lors de l'affichage dans la page JSP, toutes les valeurs **null** sont remplacées par des chaînes vides, afin de ne pas afficher le texte **null** à l'utilisateur...
- **ArrayOutOfBoundsException** :
 - De la même manière, l'accès à un index incorrect d'un élément d'un tableau ou d'une **List** ne provoquera pas d'exception mais renverra une valeur **null**.

La bibliothèque Core

Catégorie	Tag	Description
Utilisation de EL	out	envoie dans flux de sortie la valeur d'une expression EL
	remove	retire une variable d'un contexte donné
	catch	capture d'exceptions
	set	stocke une variable dans un contexte donné
Contrôle (conditions et itération)	if	évalue contenu de son corps si condition vraie
	choose	traitement de différents cas mutuellement exclusifs
	forEach	parcours des éléments d'une collection
	forTokens	parcours des éléments (tokens) d'une chaîne
Gestion des URL	import	accès à une ressource via son url
	url	permet de formater une url
	redirect	permet de faire une redirection vers une nouvelle url

Exemple

```
<!-- Afficher tous les paramètres de la requête HTTP (param est une Map)-->
<c:forEach var="entry" items="${param}" >
    Le paramètre "${entry.key}" vaut "${entry.value}".<br/>
</c:forEach>
```

Exemple

```
<!-- Afficher séparément des mots séparés par un point-virgule -->
<c:forTokens var="p" items="mot1;mot2;mot3;mot4" delims=";">
    ${p}<br/>
</c:forTokens>
```

Exemple

```
<!-- La forme suivante : -->
<c:url value="/mapage.jsp?paramName=paramValue"/>

<!-- est équivalente à : -->
<c:url value="/mapage.jsp">
    <c:param name="paramName" value="paramValue"/>
</c:url><br/>
```

Exemple

```
<!-- Cration d'un lien dont les paramtres viennent d'une MAP -->
<c:url value="/index.jsp" var="variableURL">
    <c:forEach items="${parameterMap}" var="entry">
        <c:param name="${entry.key}" value="${entry.value}"/>
    </c:forEach>
</c:url>
<a href="#">variableURL>Mon Lien</a>
```

Exemple

```
<!-- Ainsi le code suivant : -->
<c:url value="/page.jsp?param=value">

<!-- Affichera pour le contexte "contextPath" :
     /contextPath/page.jsp?param=value
     ou si les cookies sont dsactiv :
     /contextPath/page.jsp;jsessionid=XXXXXXXXXX?param=value
-->

<!-- Et le code suivant crera une variable "url" dans le scope page -->
<c:url var="url" scope="page" value="/page.jsp?param=value">
    <c:param name="id" value="1"/>
</c:url>
<!-- La variable de scope "url" contiendra donc :
     /contextPath/page.jsp?param=value&id=1
     ou si les cookies sont dsactiv :
     /contextPath/page.jsp;jsessionid=XXXXXXXXXX?param=value&id=1
-->
```

La bibliothèque Format(I18N)

Catégorie	Tag
Définition de la langue	setLocale
Formattage de messages	bundle
	message
	setBundle
Formattage de dates et nombres	formatNumber
	parseNumber
	formatDate
	parseDate
	setTimeZone
	timeZone

Facilite l'internationalisation d'une page JSP

Internationalisation (i18n)

- La classe **java.util.Locale** permet de représenter les spécificités régionales.
- Un **ResourceBundle** permet de gérer un ensemble de fichier ***.properties** contenant les ressources localisées.
- Par exemple pour gérer les langues françaises, anglaises et italiennes, on pourrait avoir les fichiers suivants :
 - **Message_fr.properties**
 - **Message_en.properties**
 - **Message_it.properties**

Internationalisation (i18n)

- <fmt:setLocale/> : Définir la Locale

Exemple

```
<!-- Force l'affichage de la page en anglais : -->
<fmt:setLocale value="en" scope="page"/>

<!-- Force l'affichage en français pour un utilisateur : -->
<fmt:setLocale value="fr" scope="session"/>
```

- <fmt:setBundle/> : Définir le ResourceBundle

Exemple

```
<!-- Change le ResourceBundle par défaut de la page : -->
<fmt:setBundle basename="package.MyRessource"/>

<!-- Création d'un autre Bundle : -->
<fmt:setBundle basename="package.MyOtherRessource" var="bundleName"/>

<!-- Affichage d'un message du ResourceBundle par défaut : -->
<fmt:message key="keyName"/>

<!-- Affichage d'un message du second Bundle : -->
<fmt:message key="keyName" bundle="${bundleName}"/>
```

La bibliothèque XML

Catégorie	Tag
Fondamentale	parse
	set
	out
Gestion du flux (condition et itération)	if
	choose
	forEach
Transformation XSLT	transform

Permet de manipuler des données en provenance d'un document XML

La bibliothèque Database

- facilite l'accès aux bases de données.
 - *solution simple mais non robuste pour accéder à des bases de données.*
 - *utile pour développer des pages de tests ou des prototypes mais n'a pas vocation à remplacer les accès réalisés grâce à des beans ou des EJB (Enterprise Java Beans).*

Catégorie	Tag
Définition de la source de données	setDataSource
Execution de requête SQL	query
	transaction
	update

Travaux pratiques

- TP: mise en œuvre JSTL / langage EL

QUESTIONS ?

Travaux pratiques

- *Développement de librairies de balises.
Intégration de JSTL aux applications
développées.*

SOMMAIRE

- **PARTIE 1 : Rappel sur le contexte des applications Web d'entreprise**
- **PARTIE 2 : Développement Web en Java**
- **PARTIE 3 : Applications Web et servlets**
- **PARTIE 4 : Présentation des Java Server Pages**
- **PARTIE 5 : Les librairies de balises**
- **PARTIE 6 : Accès aux bases de données**
- **PARTIE 7 : Introduction à Struts**

PARTIE 6 : Plan

- **Etude d'une application avec accès aux bases de données relationnelles**
 - Mise en place de la base et de l'interface JDBC. Connexion à la base, récupération d'information, mise à jour de données. Transaction. Pool de connexions. Les DataSources.
- **Correspondance BDR/Modèles objet**
 - Objectifs. Approches et outils Java.
 - Présentation de JPA et les différentes solutions du marché (Hibernate...).

Librairies JDBC

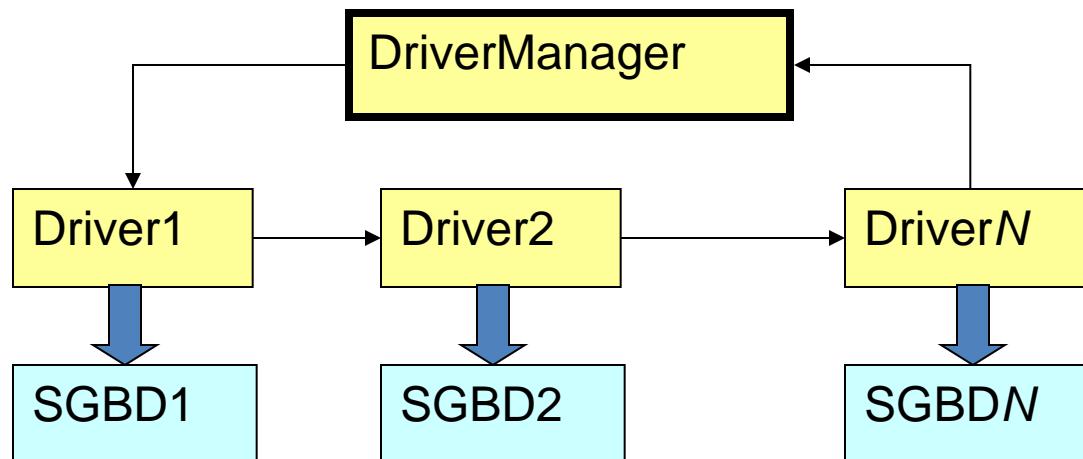
- JDBC : Java DataBase Connectivity
- Qui : Sun
- Quoi : deux librairies (accessibles dans Java 1.4)
 - `java.sql` : *JDBC 2.0 Core API*
 - `javax.sql` : *JDBC 2.0 Optional Package API*
- Objectif :
 - Accéder aux SGBD à l'aide de requêtes SQL
 - Récupérer les résultats en Java
 - Le type de la base est transparent (Sybase, Oracle, MySql, Access ...) *généricité, standardisation*

JDBC Généricité - standardisation

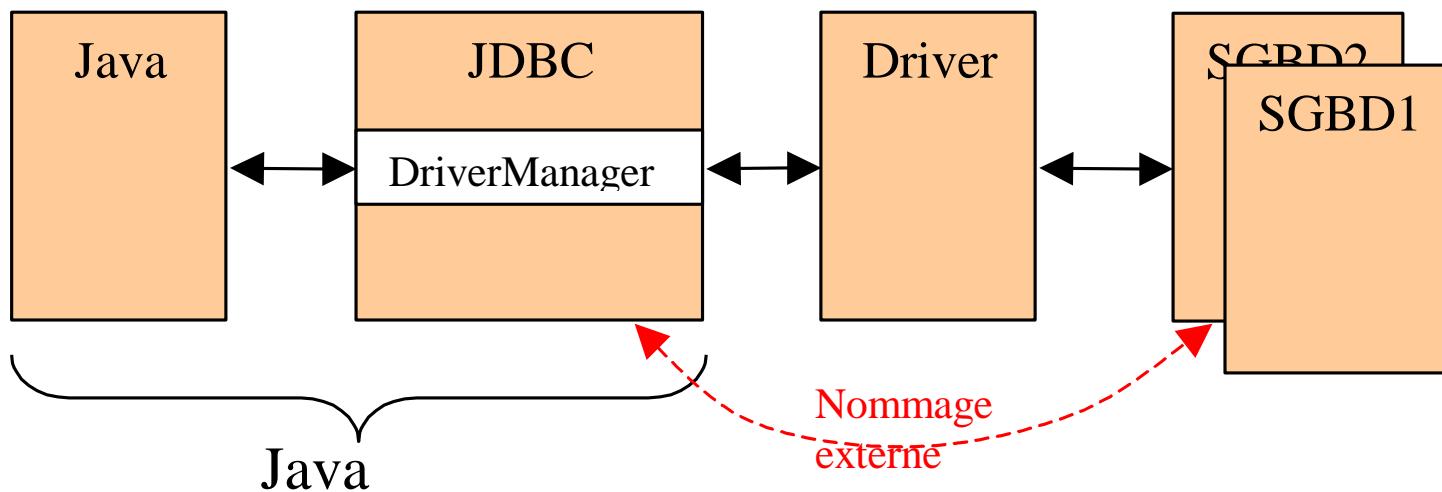
- **Abstraction des SGBDRs :**
 - JDBC définit un ensemble d'interfaces
 - Driver, Connection, Statement, ResultSet ...
 - JDBC définit une classe : DriverManager
- **Implementations** : Chaque base particulière possède son implémentation des interfaces : c'est la notion de "*pilote*"

Pilotes (Drivers) et gestionnaire de pilotes (DriverManager)

- Classe `java.sql.DriverManager` :
 - Le gestionnaire de pilotes offert par la JVM
- Interface `java.sql.Driver`
 - Définit le point d'entrée de tout accès à une base



Un même driver peut piloter plusieurs bases



Interface java.sql.Driver

- Chaque vendeur fournit une implémentation
 - Certaines sont fournies avec le Java core
 - Accès à ODBC `sun.jdbc.odbc.JdbcOdbcDriver`
 - Les autres doivent être importées
 - postGresql : `org.postgresql.Driver`
- Généralement on trouve ces classes dans des fichiers jar dans les distributions du constructeur
- Dans le pire des cas :
 - On peut trouver une liste de plus de 160 pilotes à <http://industry.java.sun.com/products/jdbc/drivers>

Travaux pratiques

Installation du serveur WAMP

Accès aux données - 5 étapes

1. Charger un pilote
2. Créer une connexion à la base
3. Créer une requête (Statement)
4. Exécuter une requête
5. Présenter les résultats

Première étape
"charger un pilote"

Charger un pilote

- Classe `java.sql.DriverManager`
 - Fournit les utilitaires de gestion des connexions et des pilotes
 - Elle ne contient que des méthodes statiques
- Les pilotes sont chargés dynamiquement
 - *implicitement* en instanciant la "system property" `jdbc.drivers`
 - *explicitement* par la méthode statique `forName()` de la classe `Class` (voir ci-après)

Conventions à propos du chargement des pilotes

- *Classe* `DriverManager`
 - Elle inspecte la propriété système `jdbc.drivers` au moment de son instantiation
- *Classes implémentant l'interface* `Driver`
 - elles doivent s'enregistrer d'elles mêmes auprès du `DriverManager` au moment de leur instantiation

Exemples de chargement

- Initialisation avant appel de l'API

```
jdbc.drivers=nom-de-classe-Driver
```

- Initialisation au moment de l'appel de l'API

```
java -Djdbc.drivers=nom-de-classe
```

- Initialisation dans le programme

```
try {  
    Class.forName("nom-de-classe");  
} catch (ClassNotFoundException ex)  
{ . . . }
```

Remarques

- L'utilisation de propriétés (systèmes ou privées à l'API) permet de ne pas "*cabler*" le nom de classes dans le code java
- La propriété `jdbc.drivers` peut contenir plusieurs noms complets de classes, il suffit de séparer les noms par des caractères ":"
- Exemple:

– `foo.bah.Driver:wombat.sql.Driver`

The diagram shows the string "foo.bah.Driver:wombat.sql.Driver" with two curly braces underneath. The first brace covers the portion "foo.bah.Driver" and is labeled "nom1". The second brace covers the portion "wombat.sql.Driver" and is labeled "nom2".

Seconde étape

"Créer une connexion à la base"

Classe `java.sql.Connection`

- Demande de connexion
 - méthode statique `getConnection(String)` classe `DriverManager`
- Le driver manager essaye de trouver un driver approprié d'après la chaîne passée en paramètre
- Structure de la chaîne décrivant la connexion

`jdbc:protocole:URL`

Exemples

`jdbc:odbc:epicerie`

`jdbc:mysql://athens.imaginaire.com:4333/db`

`jdbc:oracle:thin:@blabla:1715:test`

Classe java.sql.Connection (suite)

- Connexion sans information de sécurité

```
Connection con =  
    DriverManager.getConnection  
    ("jdbc:odbc:epicerie");
```

- Connexion avec informations de sécurité

```
Connection con =  
    DriverManager.getConnection  
    ("jdbc:odbc:epicerie", user, password);
```

- Dans tous les cas faut récupérer l'exception
java.sql.SQLException

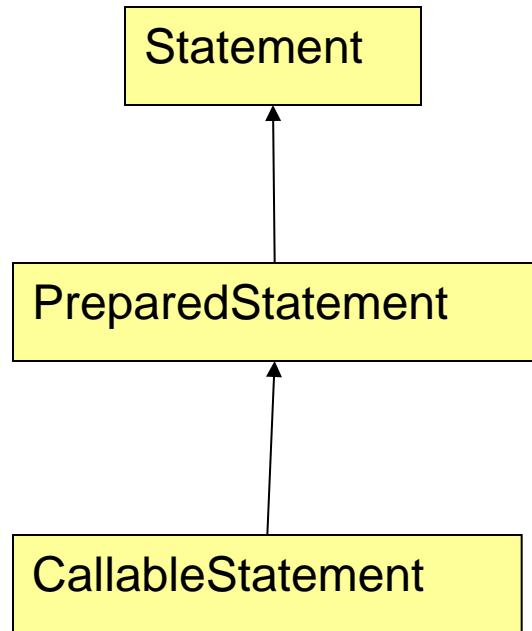
Troisième étape

"Créer une requête"

Trois types de requêtes

- Requêtes simples
 - `java.sql.Statement`
- Requêtes Précompilées
 - `java.sql.PreparedStatement`
 - Pour une opération réalisée à plusieurs reprises
- Procédures stockées
 - `java.sql.CallableStatement`
 - Pour lancer une procédure du SGBD

Héritage entre interfaces requêtes



Interface `java.sql.Statement`

- C'est le type d'objet
 - pour exécuter une requête SQL simple
 - pour recevoir ses résultats
- Exemple de création

```
Connection con =  
    DriverManager.getConnection  
    ("jdbc:odbc:epicerie");
```

```
Statement stmt = con.createStatement();
```

Statement

La requête sql n'est pas liée
à l'objet Statement

- **Initialisation**

```
Connection con = getConnection();
```

```
Statement stmt = con.createStatement();
```

- **Exécuter une requête**

```
String sql = "SELECT * FROM fournisseur";
```

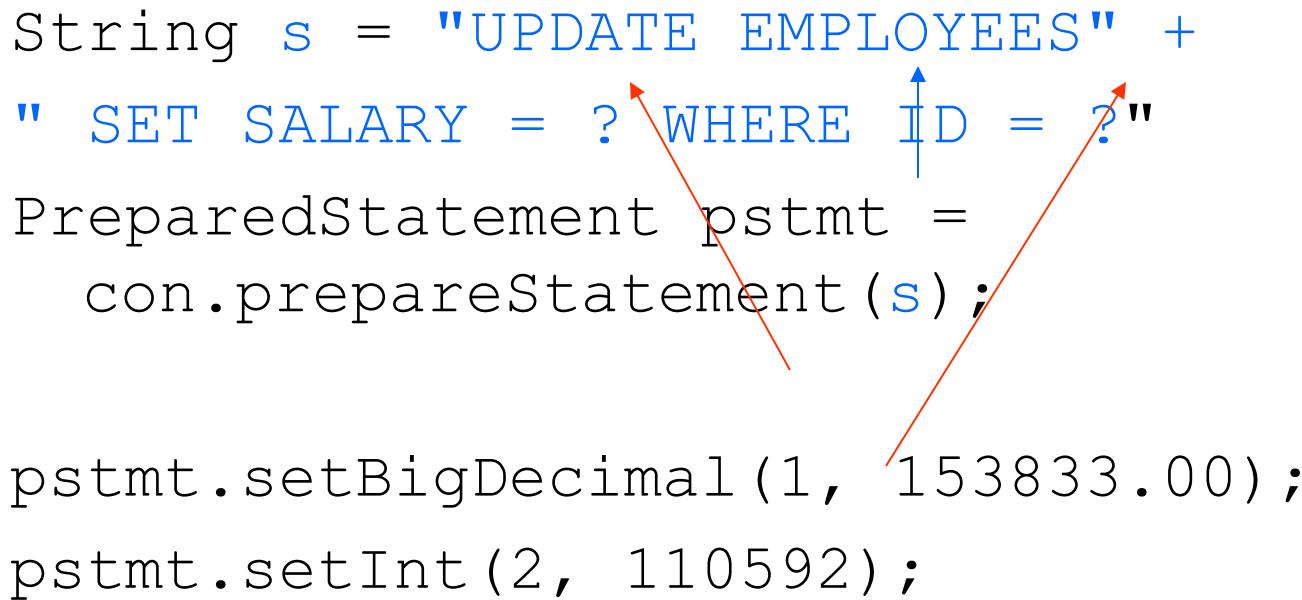
```
ResultSet rs = stmt.executeQuery(sql);
```

PreparedStatement

La requête sql est liée à l'objet PreparedStatement

- Requêtes paramétrées

```
String s = "UPDATE EMPLOYEES" +  
" SET SALARY = ? WHERE ID = ?"  
PreparedStatement pstmt =  
con.prepareStatement(s);  
  
pstmt.setBigDecimal(1, 153833.00);  
pstmt.setInt(2, 110592);
```



Quatrième étape

"Exécuter une requête"

Interrogation de la base

- Instruction SQL « select »

```
String sql = "SELECT * FROM  
fournisseur";
```

```
ResultSet rs =  
stmt.executeQuery(sql);
```

Mise à jour : effacer une table

- Instruction SQL « drop table »

```
Statement stmt =  
    con.createStatement();  
  
int i = stmt.executeUpdate(  
    "drop table fournisseur");
```

Statement - Deux types d'exécution

- Requêtes "select" retournant un tableau

```
Statement stmt = con.createStatement();  
String sql = "SELECT * FROM fournisseur";  
ResultSet rs = stmt.executeQuery(sql);
```

- Requêtes de mise à jour et de création

```
Statement stmt = conn.createStatement();  
String sql = "INSERT INTO Customers " +  
    "VALUES (1001, 'Simpson', 'Mr.', " +  
    "'Springfield', 2001)";  
int i = stmt.executeUpdate(sql);
```

executeQuery ()

- Pour lancer une requête "SELECT" statique

```
Statement stmt = con.createStatement();
```

```
String s = "select * from employes";
```

```
ResultSet rs = stmt.executeQuery(s);
```

- Un ResultSet est un objet qui modélise un tableau à deux dimensions
 - Lignes (*row*)
 - Colonnes (valeurs d'attributs)

executeUpdate () - 1

INSERT INTO

- Pour lancer une requête INSERT

```
Statement stmt = con.createStatement();
```

```
String s = "INSERT INTO test (code, val) " +  
          "VALUES (" + valCode + ", '" +val +"' )";  
int i = stmt.executeUpdate(s);
```

- Le résultat est un entier donnant le nombre de lignes créées

executeUpdate () - 2

UPDATE

- Pour lancer une requête UPDATE

```
Statement stmt = con.createStatement();
```

```
String s = "UPDATE table  
SET column = expression  
WHERE predicates";
```

```
int i = stmt.executeUpdate(s);
```

- Le résultat est un entier donnant le nombre de mises-à-jour

executeUpdate () - 3

DELETE

- Pour lancer une requête DELETE

```
Statement stmt = con.createStatement();
```

```
String s = "DELETE FROM table  
WHERE predicates";
```

```
int i = stmt.executeUpdate(s);
```

- Le résultat est un entier donnant le nombre de d'effacements

PreparedStatement

```
Class.forName(driverClass);
java.sql.Connection connection =
    java.sql.DriverManager.getConnection(url, user
, password);

PreparedStatement pstmt =
connection.prepareStatement(
    "insert into users values(?, ?, ?, ?)");
pstmt.setInt(1, user.getId());
pstmt.setString(2, user.getName());
pstmt.setString(3, user.getEmail());
pstmt.setString(4, user.getAddress());
pstmt.execute();
```

Cinquième étape

"Présenter les résultats"

Récupération des résultats

```
java.sql.Statement stmt =  
    conn.createStatement();  
String s = "SELECT a, b, c FROM Table1";  
ResultSet rs = stmt.executeQuery(s);  
while (rs.next()) {  
    int i = rs.getInt("a");  
    String s = rs.getString("b");  
    byte b[] = rs.getBytes("c");  
    System.out.println("ROW = " + i +  
        " " + s + " " + b[0]);  
}
```

Objets ResultSet

```
ResultSet rs = stmt.executeQuery(s);
```

- Se parcourt itérativement *row* par *row*
- Les colonnes sont référencées par leur numéro ou leur nom
- L'accès aux valeurs des colonnes se fait par des méthodes getXXX() où XXX représente le type de l'attribut se trouvant dans la colonne

Objets ResultSet (suite)

- Une rangée (*row*) est un tuple
- un RS contient un curseur pointant sur une rangée du résultat
- au départ le pointeur est positionné *avant la 1ère rangée*
- La méthode `next()` fait passer à l'enregistrement suivant.
- Le premier appel `resultat.next()` positionne sur la première rangée
- `next()` renvoie un booléen `true` en général, et `false` lorsque l'on a dépassé le dernier enregistrement.

types SQL - méthodes JDBC

BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
CHAR	getString()
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()
LONGVARBINARY	getBytes()

types SQL - méthodes JDBC (bis)

LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

Travaux pratiques

Connexion database à partir d'une servlet

- *LAB_D_1*
- *LAB_D_2*

QUESTIONS ?