

Servlet & JSP

Servlets et JSP

Intro :

- Servlets et JSP dans J2EE
- Le conteneur Web
- Comparaison avec d'autres technos Web, avantages de Java

Rappels HTTP

Utilisation d'une servlet

Cycle de vie d'une servlet

Servlet Web : requêtes et réponses, exemple

Paramètres, cookies, sessions

Portée des objets, partage des ressources

Applications Web : déploiement

Serveurs de servlet

JSP

Constitution des pages




Directives JSP et actions

JavaBean dans les JSP

Variables prédéfinies

Custom tags

Le paradigme MVC

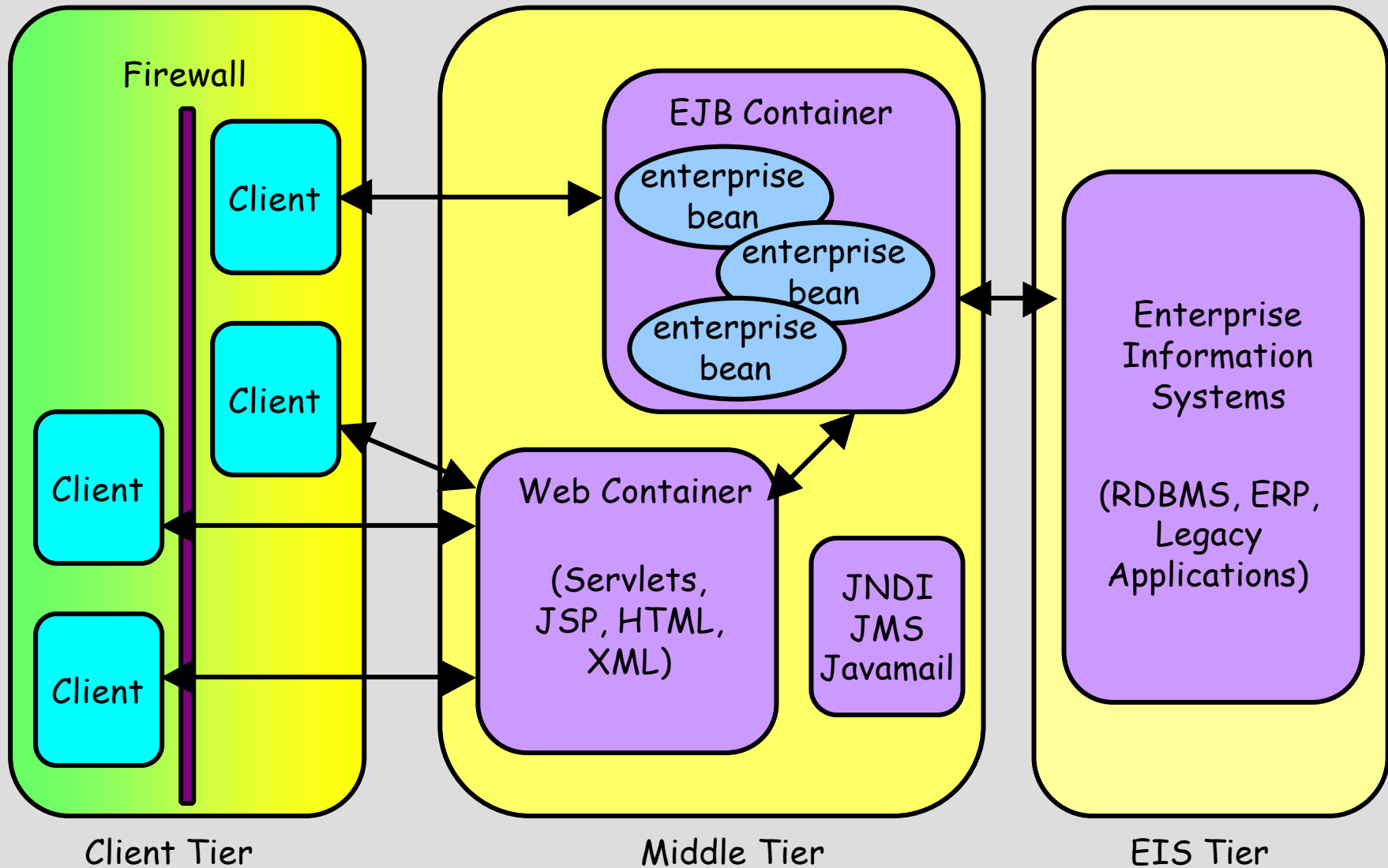
- Génèse
- MVC appliqué au Web
- Le modèle  exemple JavaBean
- La vue  exemple JSP
- Le contrôleur  exemple Servlet

Composants Web de l'architecture J2EE

Spécifications :

- Servlet
- Java Server Pages

<http://java.sun.com/products/servlet/download.html>
<http://java.sun.com/products/jsp/download.html>



Une servlet est un composant qui étend les fonctionnalités d'un serveur web de manière portable et efficace.

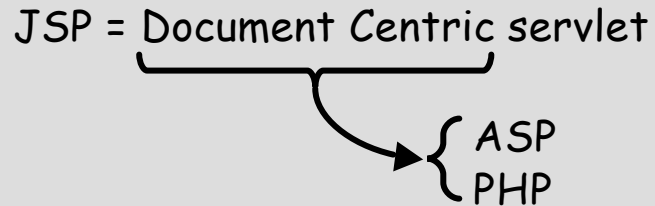
Un serveur web héberge des classes Java servlets qui sont exécutées à l'intérieur du **container web**. Le serveur web associe une ou plusieurs URLs à chaque servlet.

La servlet est invoquée lorsque des requêtes HTTP utilisateur sont soumises au serveur.

Quand la servlet reçoit une requête du client, elle génère une réponse, éventuellement en utilisant la logique métier contenue dans des EJBs ou en interrogeant directement une base de données. Elle retourne alors une réponse HTML ou XML au demandeur.

Un développeur de servlet utilise l'API servlet pour :

- Initialiser et finaliser la servlet
- Accéder à l'environnement de la servlet
- Recevoir ou rediriger les requêtes et envoyer les réponses
- Interagir avec d'autres servlets ou composants
- Maintenir les informations de sessions du client
- Filtrer avant ou après traitement les requêtes et les réponses
- Implémenter la sécurité sur le tiers web



La technologie JSP fournit un moyen simple et extensible pour générer du contenu dynamique pour le client web.

Une page JSP est un document texte qui décrit comment traiter la requête d'un client et comment créer une réponse.

Une page JSP contient :

- Des informations de formatage (modèle) du document web, habituellement en HTML ou XML. Les concepteurs web peuvent modifier cette partie de la page sans affecter les parties dynamiques. Cette approche permet de séparer la présentation du contenu dynamique.
- Des éléments JSP et de script pour générer le contenu dynamique du document Web. La plupart des pages JSP utilisent aussi des JavaBeans et/ou des Enterprise JavaBeans pour réaliser les opérations complexes de l'application. Les JSP permettent en standard d'instancier des beans, de modifier ou lire leurs attributs et de télécharger des applets. La technologie JSP est extensible en utilisant des balises personnalisées qui peuvent être encapsulées dans des bibliothèques de balises personnalisées (taglibs)

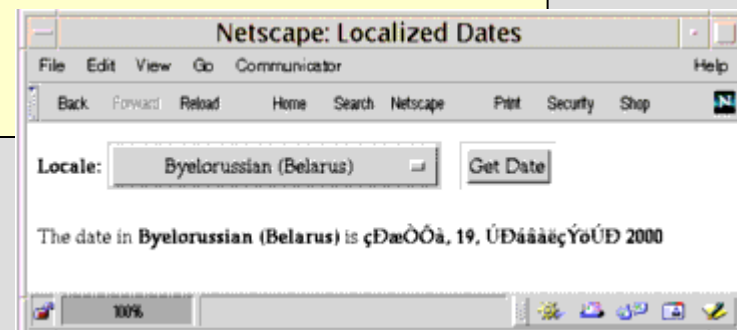
```

<%@ page import="java.util.*,MyLocales" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<html>
  <head>
    <title>Localized Dates</title>
  </head>
  <body bgcolor="white">
    <jsp:useBean id="locales" scope="application" class="MyLocales"/>
    <form name="localeForm" action="index.jsp" method="post">
      <b>Locale:</b>
      <select name="locale">
        <% String selectedLocale = request.getParameter("locale");
           Iterator i = locales.getLocaleNames().iterator();
           while (i.hasNext()) {
             String locale = (String)i.next();
             if (selectedLocale != null && selectedLocale.equals(locale)) {%>
               <option selected><%=locale%></option>
             <% } else { %>
               <option><%=locale%></option>
             <% }
           } %>
      </select>
      <input type="submit" name="Submit" value="Get Date">
    </form>
    <jsp:include page="date.jsp"/>
  </body>
</html>

```

Le résultat
d'une page
peut être :

- HTML
- XML
- SVG
- WML
- ...



Le résultat est une page HTML dynamique

Les composants web sont hébergés dans des conteneurs de servlets, conteneurs de JSP et conteneurs web.

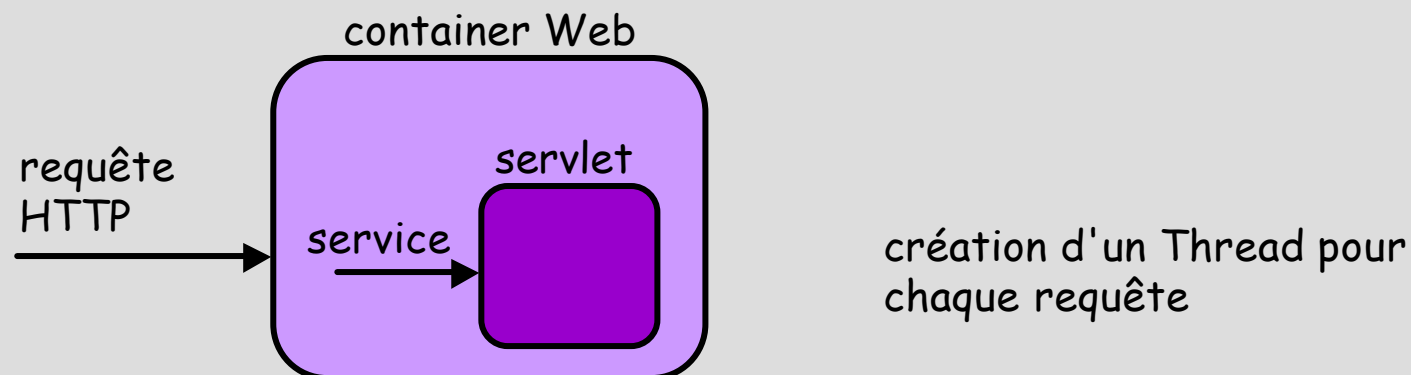
En sus des fonctionnalités normales d'un conteneur de composants, un conteneur de servlets (servlets container) fournit les services réseaux par lesquels les requêtes et réponses sont émises.

Il décode également les requêtes et formate les réponses dans le format approprié.

Tous les conteneurs de servlets doivent supporter le protocole HTTP et peuvent aussi supporter le protocole HTTPS.

Un conteneur de JSP (JSP container) fournit les mêmes services qu'un conteneur de servlets.

Ces conteneurs sont généralement appelés conteneurs web (Web containers).



→ (pas de "bricolage")

La technologie J2EE offre une approche beaucoup plus formelle pour le traitement des **applications Web** que les technologies alternatives apparentées (ASP, PHP, CGI, ISAPI, NSAPI...)

CGI	Servlets
<p>Un processus par requête est lancée sur le serveur</p> <ul style="list-style-type: none"> • gratuit • implémenté sur tous les serveurs Web • supporte tous les langages (les utilisateurs l'utilisent surtout avec PERL) • assez lent • parfois difficile à développer <p>Manque d'évolutivité (plusieurs processus créés, serveur très sollicité)</p> <p>Contournement :</p> <ul style="list-style-type: none"> • FastCGI : instance partagée des programmes CGI • mod_perl (Apache) : script CGI interprété et exécuté dans le serveur Web 	<ul style="list-style-type: none"> • Résidentes • pas de temps de lancement • Multithreads • Gestion de cache • Connexions persistantes (BD) • Plus efficaces • Plus pratiques • Plus puissantes • Portables • Gratuites <p>On peut faire des choses impossibles à réaliser avec des scripts CGI</p> <p>C'est du Java !</p> <p style="text-align: right;">...ça rame</p>

Servlets plus pratiques :

C'est du Java !

API pour gérer :

- les cookies
- le suivi de session
- le protocole HTTP (headers HTTP)

Plus facile à utiliser que CGI/PERL

Portabilité de Java

Supportées sur tous les serveurs :

- Apache
- Microsoft IIS
- WebStar...

Partage de données entre servlets

Chaînage de servlets

Gestion de sessions

Inconvénient :

Comme toutes les technos Web, l'interface graphique utilisateur est limitée à HTML

Contenu d'une requête HTTP

- infos d'en-tête
- URL de la ressource
- données de formatage

Requête GET :

- pour extraire des informations sur le serveur
- intègre les données de formatage à l'URL

```
http://www.inria.fr/hello?param1=value1&...
```

Requête POST :

- pour modifier les données sur le serveur
- données de la page assemblées/envoyées vers le serveur

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
 - ...

Une servlet doit implémenter l'interface `javax.servlet.Servlet`

- soit directement,
- soit en dérivant d'une classe qui implémente déjà cette interface (comme `GenericServlet` ou `HttpServlet`)

L'interface `javax.servlet.Servlet` possède les méthodes pour :

- initialiser la servlet : `init()`
- recevoir et répondre aux requêtes des clients : `service()`
- détruire la servlet et ses ressources : `destroy()`

```
import javax.servlet.*;

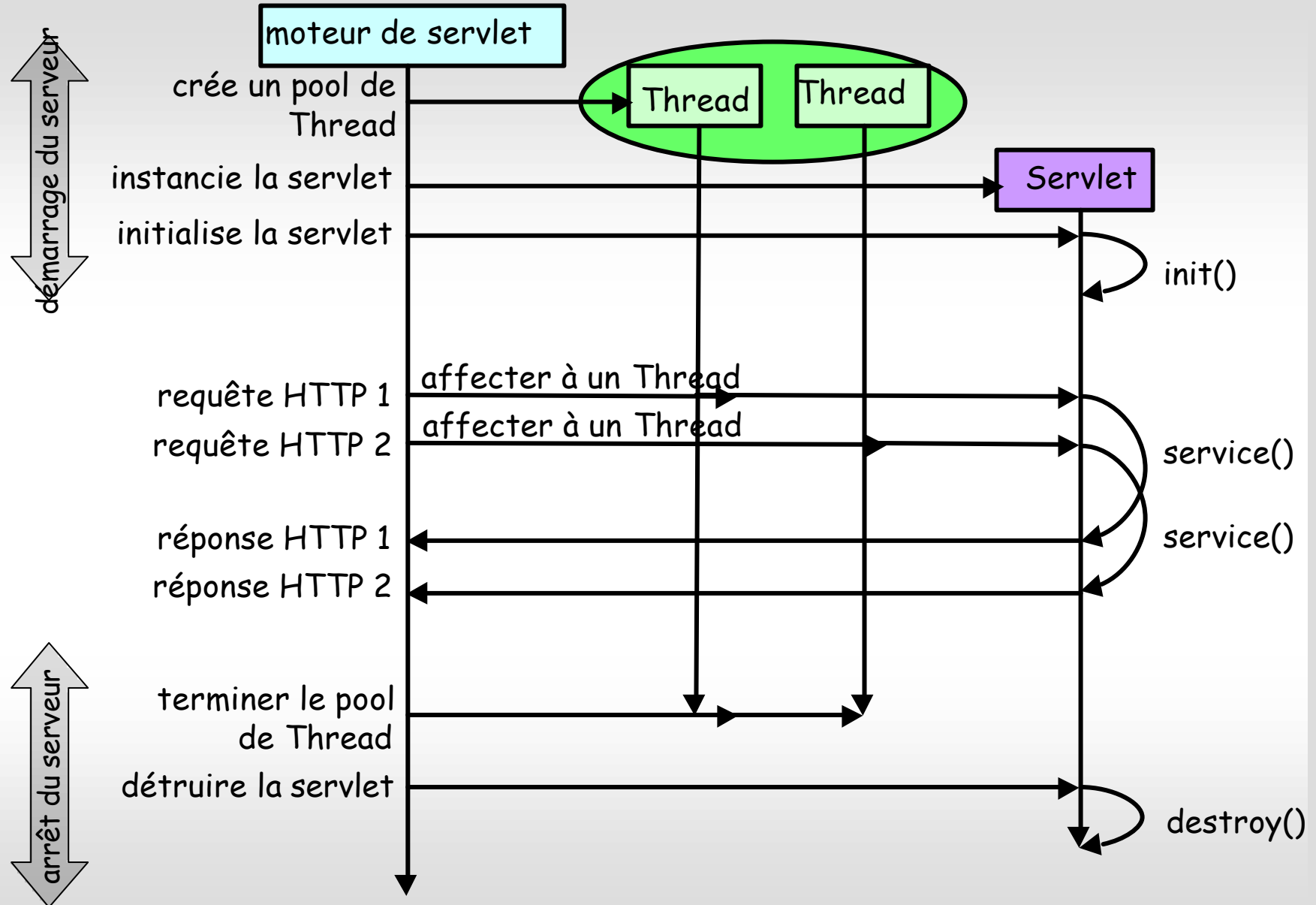
public class MyServlet implements Servlet {

    public void init(ServletConfig config)
        throws ServletException {
        // phase d'initialisation
    }

    public void service( ServletRequest req, ServletResponse rep)
        throws ServletException, IOException {
        // phase de traitement des requêtes
    }

    public void destroy() {
        // phase d'arrêt
    }

}
```



Une servlet Web étend la classe `javax.servlet.http.HttpServlet` (elle implémente `javax.servlet.Servlet`)

Plusieurs méthodes spécifiques au **protocole HTTP** remplacent la méthode `service()`, qui appelle la méthode correspondant au type de requête :

Méthode	Type de requête HTTP
<code>doGet()</code>	GET
<code>doPost()</code>	POST
<code>doPut()</code>	PUT
<code>doDelete()</code>	DELETE
<code>doHead()</code>	HEAD
<code>doOptions()</code>	OPTIONS
<code>doTrace()</code>	TRACE

Une servlet doit redéfinir au moins l'une de ces méthodes

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

public class SimpleServlet extends HttpServlet {

    public void init(ServletConfig c)
        throws ServletException {
        // phase d'initialisation
    }

    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException {
        // phase de traitement des requêtes HTTP GET
    }

    public void destroy() {
        // phase d'arrêt
    }

    public String getServletInfo() {
        // délivre des informations sur la servlet
    }

}
```

Les méthodes `doGet()`, `doPost()`, `doPut()`, `doDelete()`, `doHead()`, `doOptions()` et `doTrace()` utilisent des objets `HttpServletRequest` et `HttpServletResponse` passés en paramètres pour implémenter le service.

`javax.servlet.http.HttpServletRequest` contient les renseignements sur le formulaire HTML initial :

- la méthode `getParameter()` récupère les paramètres d'entrée

`javax.servlet.http.HttpServletResponse` contient le flux de sortie pour la génération de la page HTML résultat

- la méthode `getWriter()` permet de l'obtenir

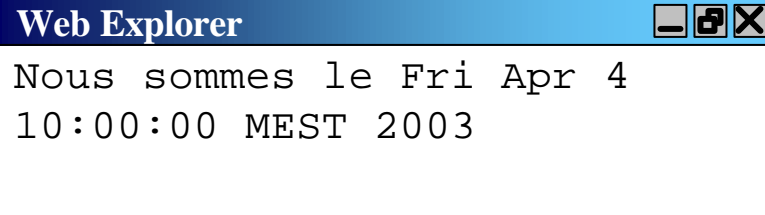
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        // Use "request" to read incoming HTTP headers (e.g. cookies)
        // and HTML form data (e.g. data the user entered and submitted)
        // Use "response" to specify the HTTP response line and headers
        // (e.g. specifying the content type, setting cookies).
        PrintWriter out = response.getWriter();
        // Use "out" to send content to browser
    }
}
```

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

public class DateDuJour extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("plain/text");
        PrintWriter out = response.getWriter();
        out.println("Nous sommes le " + (new java.util.Date()).toString());
        out.flush();
    }
}
```



Les paramètres sont créés par les formulaires HTML

Visibles (GET) ou non (POST)

Ces paramètres doivent être décodés (CGI)... tâche préparée en amont avec les servlets !

```
http://www.inria.fr/hello?param1=value1&...
```

La méthode `getParameter()` de `HttpServletRequest` fonctionne indifféremment avec GET ou POST

```
public void doGet( HttpServletRequest req, HttpServletResponse rep)
    throws ServletException, IOException {
    Enumeration list = req.getParameterNames();
    String value1 = req.getParameter("param1");
    if(value1 == null) {
        // ...
    }
    // ...
}
```


C'est quoi ?

Informations envoyées par le serveur, stockée sur le client
... et renvoyées par le client quand il revient visiter le même URL

- Durée de vie réglable
- Permet d'avoir des données persistantes côté client

Utilisations courantes

- Identification des utilisateurs
- Eviter la saisie d'informations à répétition (login, password, adresse, téléphone...)
- Gérer des préférences utilisateur, profils

Cookie et sécurité

Jamais interprété ou exécuté : pas de virus

Pas partageable : le navigateur ne distribue le cookie qu'au serveur qui l'a émis

Un cookie est limité à 4KB et les navigateurs se limitent à 300 cookies (20 par site) : pas de surcharge de disque

Bien pour rendre privées des données non sensibles (pas n° CB !)

... mais ne constitue pas un traitement sérieux de la sécurité

Une API pour manipuler les cookies :

Classe : `javax.servlet.http.Cookie`

- écrire/lire un cookie : `addCookie(cookie)`, `getCookies()`,
- positionner des attributs d'un cookie : `Cookie#setXxx()`

Exemple d'envoi d'un cookie :

```
Cookie unCookie = new Cookie("nom", valeur);
// ici positionnement des attributs si nécessaire
response.addCookie(unCookie);
```

caractères non autorisés :
espace
[] () = , " / ? @ : ;

Attributs des cookies

- `getValue()/setValue()`
- `getName()/setName()`
- `getComment()/setComment()`
- `getMaxAge()/setMaxAge()` : délai restant avant expiration du cookie (en seconde)
(par défaut : pour la session courante)
- `getPath()/setPath()` : répertoire où s'applique le cookie
(répertoire courant ou chemin spécifique)

```
Cookie [] cookies = request.getCookies();
String nom = getCookieValue(cookies, "nom", "non trouvé");
// ...
public static String getCookieValue(Cookie [] cookies,
    String cookieName, String defaultValue) {
    for(int i=0; i < cookies.length; i++) {
        Cookie cookie = cookies[i];
        if(cookieName.equals(cookie.getName()) return(cookie.getValue()));
    }
    return(defaultValue);
}
```

Très simple avec l'API des servlets

Interface `javax.servlet.http.HttpSession`

Un objet "session" peut être associé avec chaque requête

Il va servir à stocker des informations temporairement

Durée de vie limitée et réglable

si la session n'existe pas déjà :
true : crée l'objet
false : renvoie null

```
HttpSession session = request.getSession(true);
Caddy caddy = (Caddy) session.getValue("caddy");
if(caddy != null) {
    // le caddy n'est pas vide !
    afficheLeContenuDuCaddy(caddy);
} else {
    caddy = new Caddy();
    // ...
    caddy.ajouterUnAchat(request.getParameter("NoArticle"));
    session.putValue("caddy", caddy);
}
// ...
```

Quelques méthodes :

```
getID()
isNew()
getCreationTime()/getLastAccessedTime()
getMaxInactiveInterval()
getValue()/removeValue()/putValue()
invalidate()
```

Scope objects	Classe	Accessibilité depuis
Web context	<code>javax.servlet.ServletContext</code>	les composants web d'une application
session	<code>javax.servlet.http.HttpSession</code>	les composants web en rapport avec les requêtes d'une session (maintient de l'état client)
request	<code>javax.servlet.HttpServletRequest</code>	les composants web en rapport avec une requête
page	<code>javax.servlet.jsp.PageContext</code>	la page JSP qui a créé l'objet

Contrôle des accès concurrents

- données en mémoire
- fichiers
- connections aux bases de données
- connections aux réseaux

```
public class Counter {
    private int counter;
    public Counter() {
        counter = 0;
    }
    public synchronized int getCounter() {
        return counter;
    }
    public synchronized int setCounter(int c) {
        counter = c;
        return counter;
    }
    public synchronized int incCounter() {
        return(++counter);
    }
}
```

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \"
            + \"Transitional//EN\">\n"
            + "<html>\n<head><title>Hello World</title></head>\n"
            + "<body bgcolor=\"white\"><h1>Hello World</h1>\n");
        out.println("<p>" + (new java.util.Date()).toString() + "</p>");
        out.println("</body></html>");
    }
}

```

```

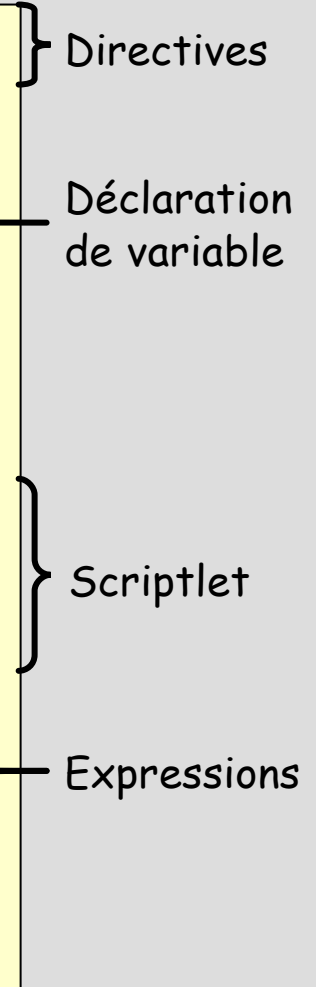
<%@ page contentType="text/html; charset=ISO-8859-1" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " "Transitional//EN">
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body bgcolor="white">
    <h1>Hello World</h1>
    <p><%= (new java.util.Date()).toString() %></p>
  </body>
</html>

```

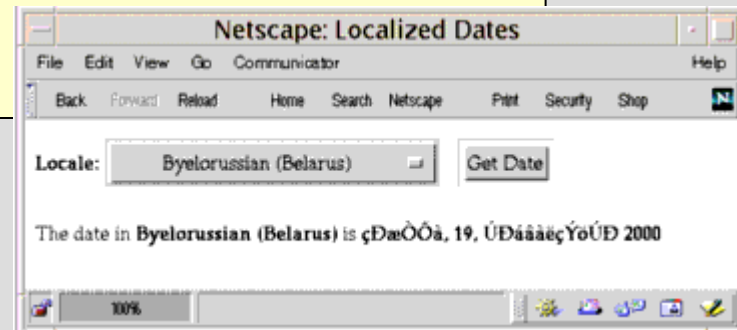
Le conteneur génère une servlet Java à partir du code JSP, puis compile la classe. Cette opération n'est faite qu'une fois, et sera renouvelée en cas de modification du code JSP.

```

<%@ page import="java.util.*,MyLocales" %>
<%@ page contentType="text/html; charset=ISO-8859-5" %>
<%! char c = 0; %>
<html>
  <head>
    <title>Localized Dates</title>
  </head>
  <body bgcolor="white">
    <jsp:useBean id="locales" scope="application" class="MyLocales"/>
    <form name="localeForm" action="index.jsp" method="post">
      <b>Locale:</b>
      <select name="locale">
        <% String selectedLocale = request.getParameter("locale");
           Iterator i = locales.getLocaleNames().iterator();
           while (i.hasNext()) {
             String locale = (String)i.next();
             if (selectedLocale != null && selectedLocale.equals(locale)) { %>
               <option selected><%=locale%></option>
             <% } else { %>
               <option><%=locale%></option>
             <% } %>
           } %>
      </select>
      <input type="submit" name="Submit" value="Get Date">
    </form>
    <jsp:include page="date.jsp"/>
  </body>
</html>
    
```



Elements JSP



Directives

```
<%@ page language="java"  
<%@ page import="java.util.*, java.net.*" %>  
<%@ page contentType="text/plain" %>  
<%@ page session="true|false " %>  
<%@ page errorPage="pathToErrorPage" %>  
<%@ page isErrorPage="true|false" %>
```

```
<%@ include file="chemin relatif du fichier" %>
```

Actions

```
<jsp:include page="relative URL" flush="true" />
```

inclusion au moment où la page est servie, pas au moment où elle est compilée

```
<jsp:usebean id="name" class="package.class" />
```

permet d'instancier un bean depuis une page JSP.

nécessite de connaître le mécanisme des beans...

associé à `<jsp:getProperty />` et `<jsp:setProperty />`

```
<jsp:forward page="/unAutreURI" />
```

redirige vers un autre URI/URL

```
<jsp:usebean id="name"
             class="paquetage.class"
             scope="page|request|session|application"
/>
```

← référence l'instance du composant
 ← nom qualifié de la classe
 ← portée

Lecture d'une propriété du bean :

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

Modification d'une propriété du bean :

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

Initialise tous les attributs de l'objet name avec les paramètres HTTP du même nom

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

```
<html>
  <body>
    <jsp:usebean id="test" class="SimpleBean" />
    <jsp:setProperty name="test" property="message" value="Hello !!" />
    <h1>Le message est : <i><jsp:getProperty name="test" property="message" /></i></h1>
  </body>
</html>
```

Exemple

```
public class SimpleBean {
  private String message = "no message";
  public String getMessage() {
    return message;
  }
  public void setMessage(String message) {
    this.message = message;
  }
}
```

Nom de la variable	Classe	Description
application	<code>javax.servlet.ServletContext</code>	l'application Web de la page JSP
config	<code>javax.servlet.ServletConfig</code>	informations d'initialisation de la servlet JSP
exception	<code>java.lang.Throwable</code>	accessible seulement depuis les pages d'erreur
out	<code>javax.servlet.jsp.JspWriter</code>	le flot de sortie
page	<code>java.lang.Object</code>	l'instance de la servlet JSP
pageContext	<code>javax.servlet.jsp.PageContext</code>	les composants web en rapport avec une requête
request	<code>javax.servlet.HttpServletRequest</code>	la requête courante
response	<code>javax.servlet.HttpServletResponse</code>	la réponse
session	<code>javax.servlet.http.HttpSession</code>	la session courante

```
http://java.sun.com/products/jsp/taglibraries.html
```

Des éléments de langage à définir pour les pages JSP

Déclaration :

```
<%@ taglib uri="/WEB-INF/my-tag.tld" prefix="mt" %>
```

Utilisation :

```
<mt:tag> body </mt:tag>
```

JSTL

JSP Standard Tag Library

```
<c:forEach var="item" items="${sessionScope.cart.items}">
  // ...
</c:forEach>
```

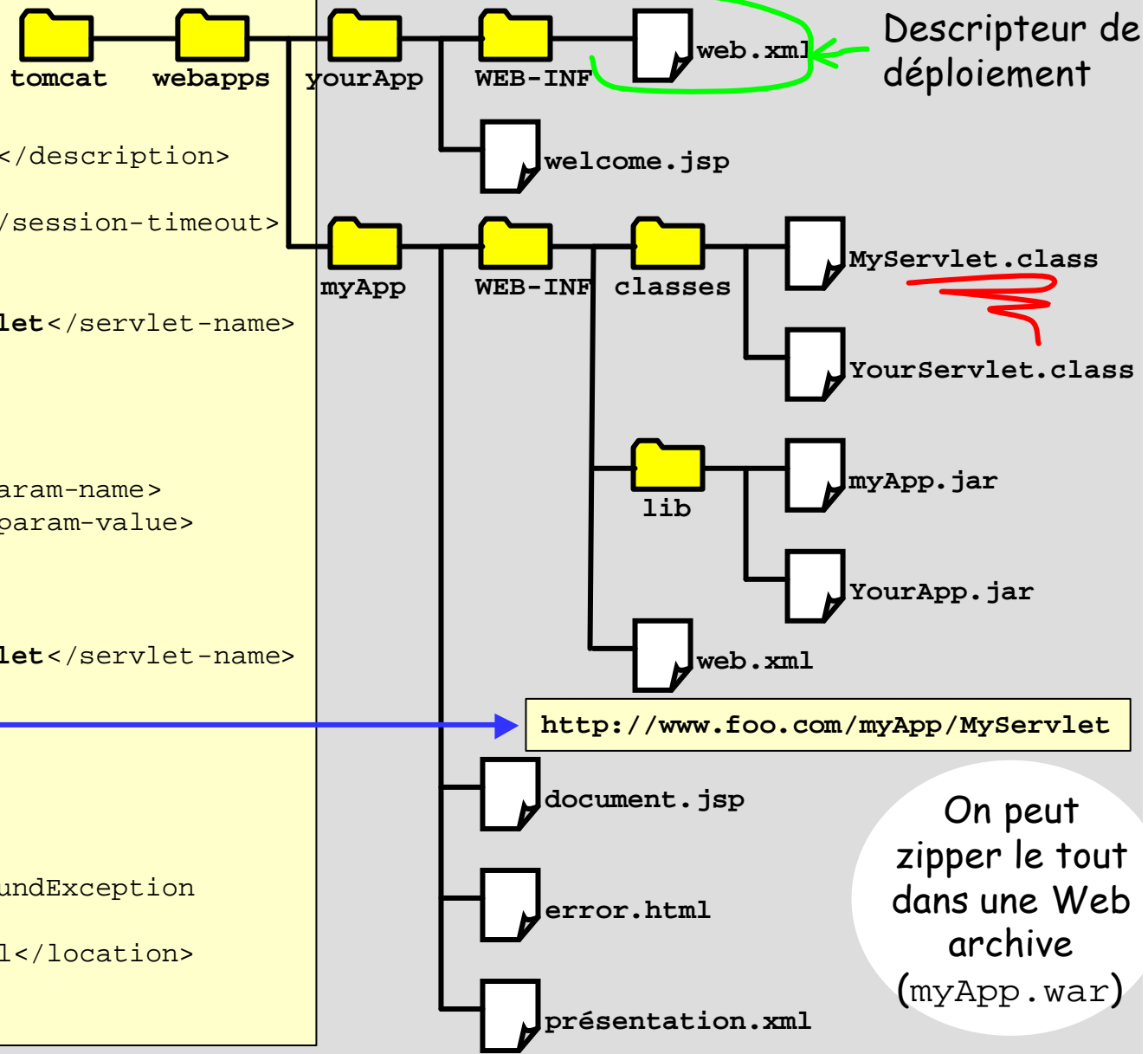
```
<c:set var="bookId" value="${param.Remove}"/>
<jsp:useBean id="bookId" type="java.lang.String" />
<% cart.remove(bookId); %>
<sql:query var="books" dataSource="${applicationScope.bookDS}">
  select * from PUBLIC.books where id = ?
  <sql:param value="${bookId}" />
</sql:query>
```

```
<x:set var="abook"
select="${applicationScope.booklist/
  books/book[@id=$param:bookId]" />
<h2><x:out select="$abook/title"/></h2>
```

```

<web-app>
  <display-name>
    My web app
  </display-name>
  <description>A web app</description>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
  <servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>
      MyServlet.class
    </servlet-class>
    <init-param>
      <param-name>foo</param-name>
      <param-value>bar</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>
      /MyServlet
    </url-pattern>
  </servlet-mapping>
  <error-page>
    <exception-type>
      exception.DocNotFoundException
    </exception-type>
    <location>/error.html</location>
  </error-page>
</web-app>

```



On peut zipper le tout dans une Web archive (myApp.war)

```
String xmlFileName = servletContext.getRealPath("/présentation.xml")
```

/tomcat/webapps/myApp/présentation.xml

Serveurs de servlet

- Apache Tomcat
- Allaire Jrun
- New Atlanta Server Exec
- Sun Java Web Server...

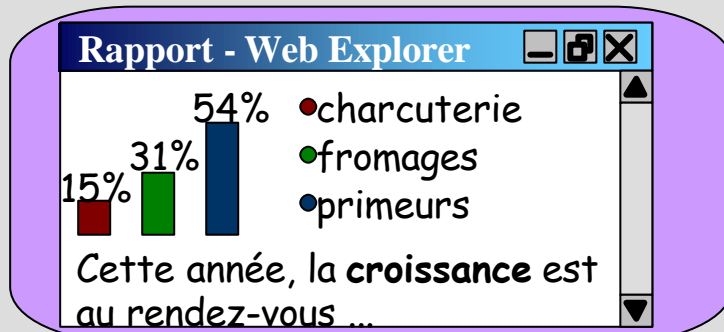
Plate-forme de développement

- Java Web Services Developer Pack

Le paradigme MVC

Servlet

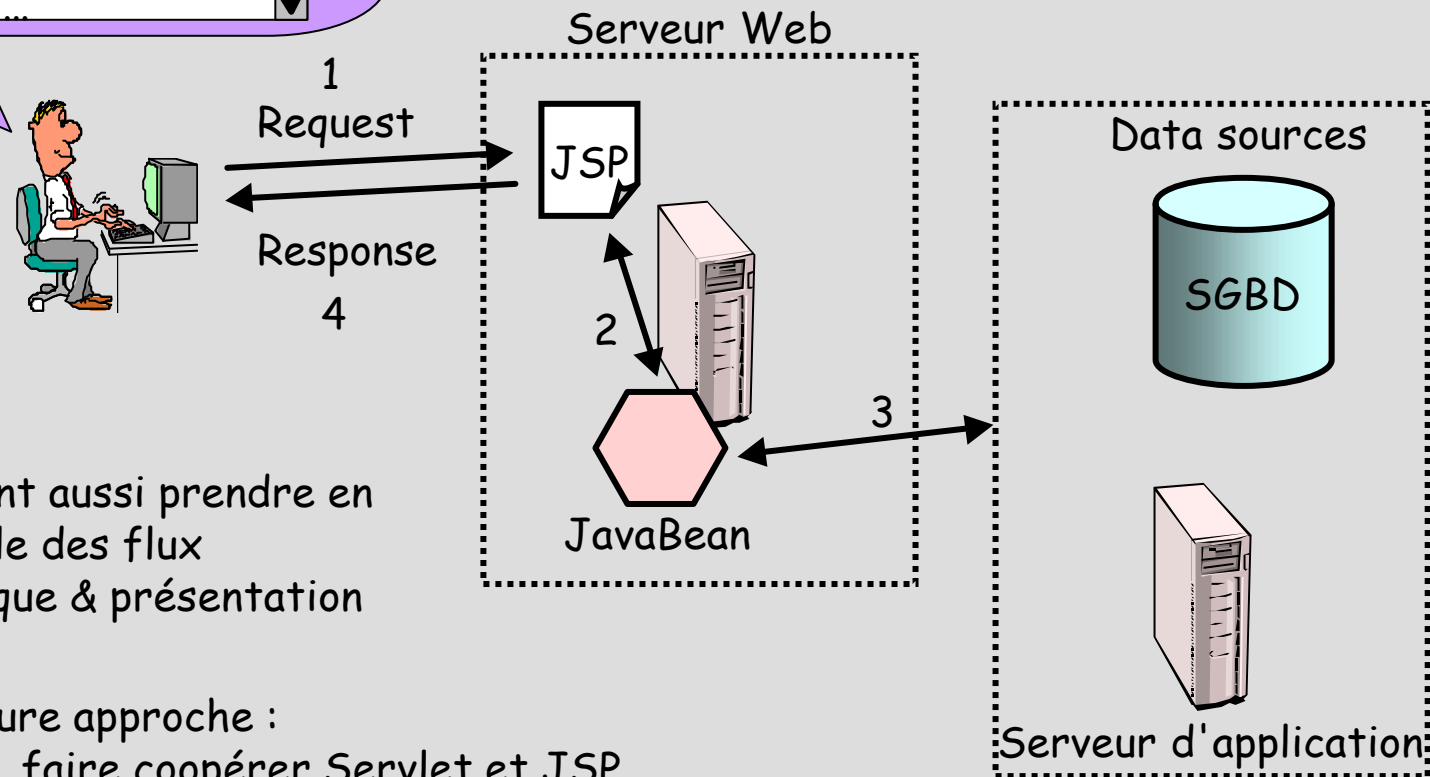
Comme les CGI
Production de HTML avec `println()`



JSP

Les applications deviennent centrées sur les pages (JSP-centric)
Architecture JSP "Model 1"

Architecture JSP "Model 1"

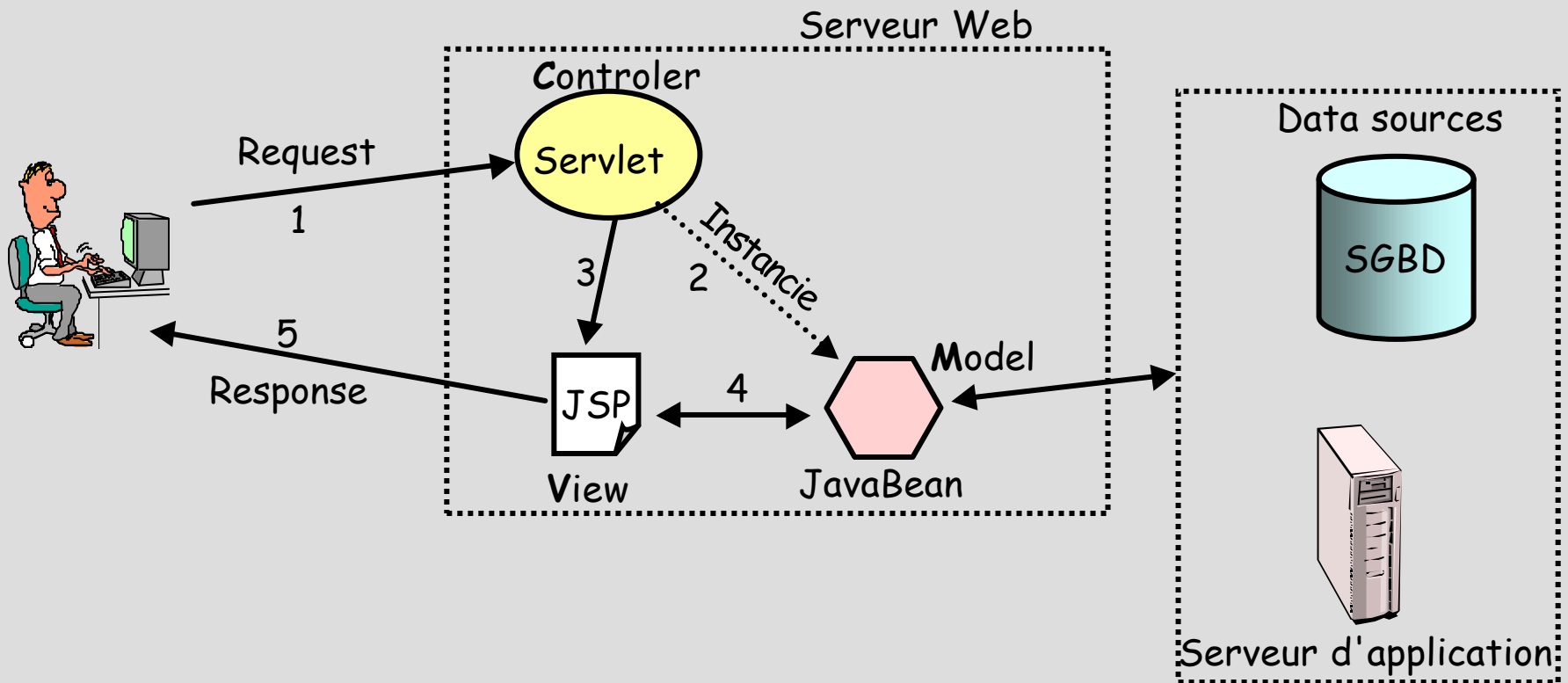


- ☹ Les JSP doivent aussi prendre en charge le contrôle des flux
- ☹ Mixité de logique & présentation

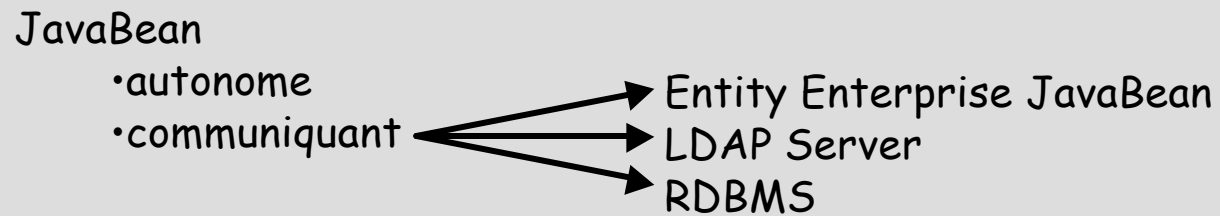
→ Meilleure approche :
faire coopérer Servlet et JSP

- Modularisation
- Couplage faible
- Couvertures fonctionnelles spécifiques

Architecture JSP "Model 2"

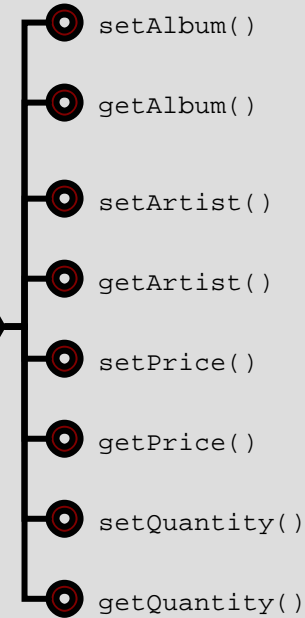


- Etat du système
- Logique métier



CD.java

```
public class CD {
    private String album="";
    private String artist="";
    private float price=0;
    private int quantity=0;
    public CD() {}
    public void setAlbum( String album ) {
        this.album = album;
    }
    public String getAlbum() {
        return this.album;
    }
    public void setArtist( String artist ) {
        this.artist = artist;
    }
    public String getArtist() {
        return this.artist;
    }
    public void setPrice( float price ) {
        this.price = price;
    }
    public float getPrice() {
        return this.price;
    }
    public void setQuantity( int quantity ) {
        this.quantity = quantity;
    }
    public int getQuantity() {
        return this.quantity;
    }
}
```

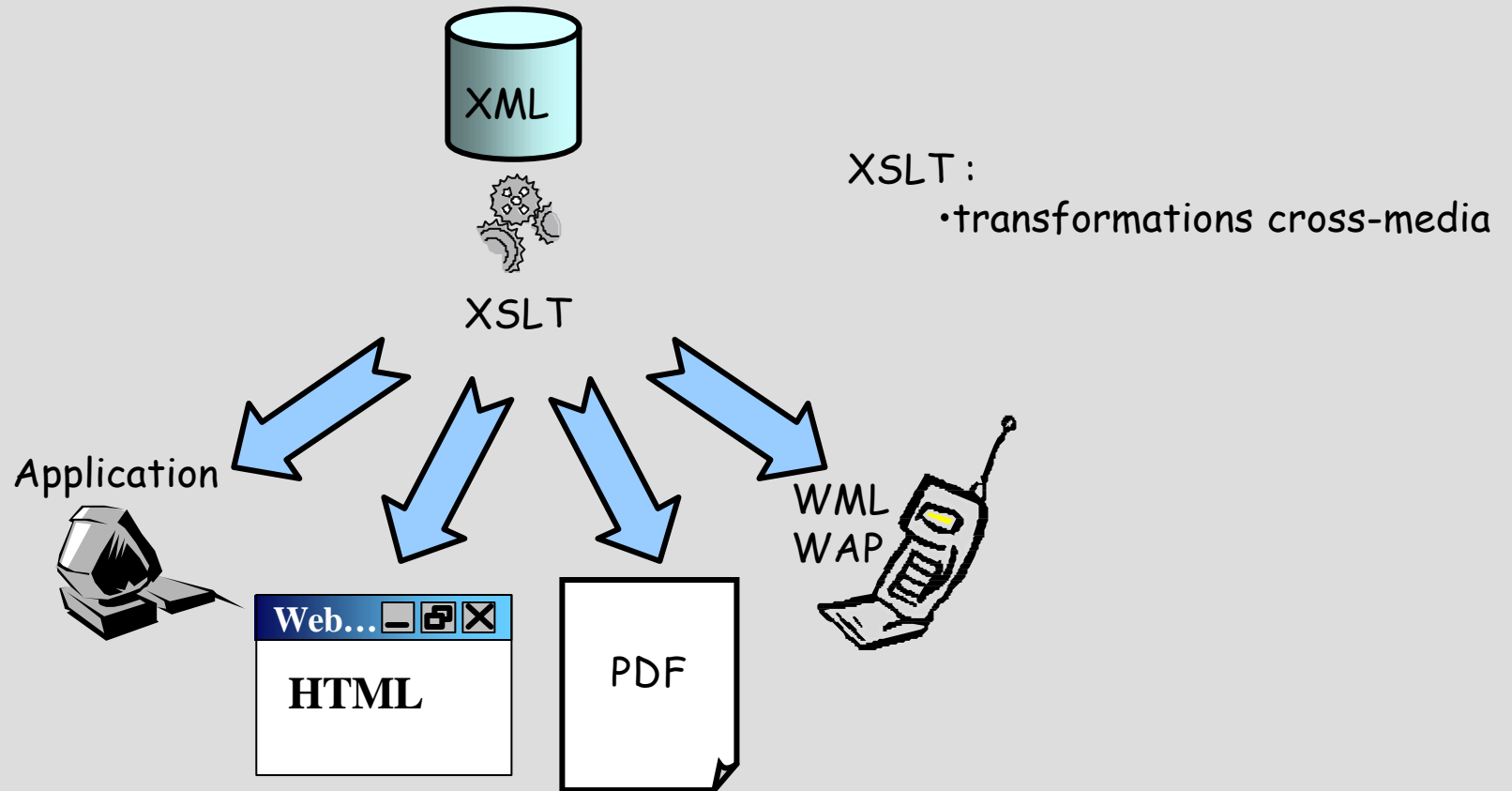


```
public boolean equals(Object o) {
    if (o instanceof CD) {
        CD cd = (CD) o;
        return this.album.equals(cd.album)
            && this.artist.equals(cd.artist);
    }
    return false;
}
public int hashCode() {
    return this.album.hashCode() ^ this.artist.hashCode();
}
}
```

- JSP
- Velocity
- XSLT

→ Template text + contenu dynamique

Tags tels que `<jsp:useBean>`
+ custom tag libraries JSTL



```
<%@page session="true" import="java.util.*, CD"%>
<% Vector buyList = (Vector) session.getValue("shopping.cdCart");
if (buyList != null && (buyList.size())>0) {%>
<table border="2">
<tr><th>Album</th>
<th>Artiste</th>
<th>Prix</th>
<th>Quantité</th>
<th></th></tr>
<% for (int i=0; i<buyList.size();i++) {
CD cdOrder = (CD) buyList.elementAt(i);%>
<tr><td><%=cdOrder.getAlbum()%></td>
<td><%=cdOrder.getArtist()%></td>
<td><%=cdOrder.getPrice()%></td>
<td><%=cdOrder.getQuantity()%></td>
<td><form name="deleteForm"
action="/cd/CdShopServlet"
method="post">
<input type="submit"
value="Supprimer">
<input type="hidden"
name="delIndex"
value="<%=i%>">
<input type="hidden"
name="action" *
value="DELETE">
</form></td></tr>
<% }%>
</table>
<p>
<form name="checkout" action="/cd/CdShopServlet"
method="post">
<input type="hidden" name="action" value="CHECKOUT">
<input type="submit" value="Commander">
</form>
</p>
<% }%>
```

Cart.jsp

Shop.jsp

```
<%@page session="true"%>
<html>
<head><title>Music shopping</title></head>
<body>
<h1>Music shopping</h1>
<form name="shopping" action="/cd/CdShopServlet" method="post">
<b>CD :</b>
<select name="CD">
<option>Red hot chili peppers|Californication|16.95</option>
<option>U2 |The joshua tree|14.95</option>
<option>Rolling stones |best of |18.95</option>
<option>Radiohead |OK computer |15.95</option>
</select>
<b>quantité :</b>
<input type="text" name="qty" size="3" value="1">
<input type="hidden" name="action" value="ADD">
<input type="submit" name="submit" value="Ajouter au panier">
</form>
<jsp:include page="Cart.jsp" flush="true"/>
</body>
</html>
```

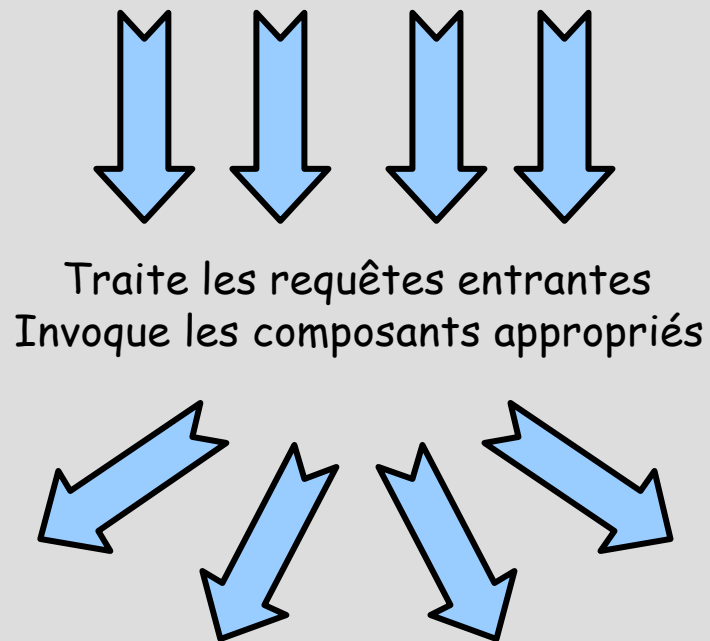
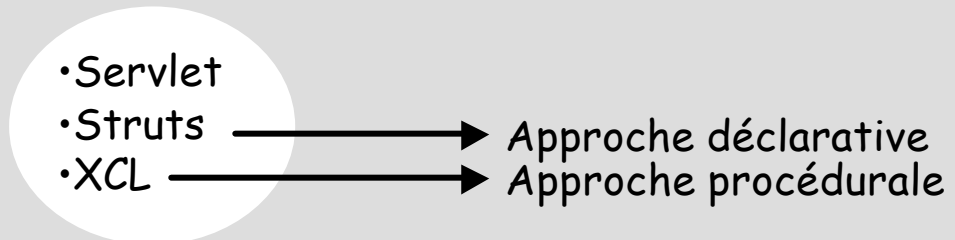
Music shopping - Web Explorer

Music shopping

CD: Red hot chili peppers|Californication|16.95 ▼ quantité: 1

[Ajouter au panier](#)

Album	Artiste	Prix	Quantité	
best of	Rolling stones	18.95	2	Supprimer
OK computer	Radiohead	15.95	1	Supprimer



CdShopServlet.java

```

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import CD;

public class CdShopServlet extends HttpServlet {
    public void doPost(HttpServletRequest req,
        HttpServletResponse res)
        throws ServletException, IOException {
        HttpSession session=req.getSession(false);
        if (session==null) {
            res.sendRedirect("error.html");
        }
        Vector buyList= (Vector) session.getValue("shopping.cdCart");
        String action = req.getParameter("action");
        if ("CHECKOUT".equals(action)) {
            // order stuff here
        } else {
            if ("DELETE".equals(action)) {
                String del = req.getParameter("delIndex");
                int d = Integer.parseInt(d);
                buyList.removeElementAt(d);
            } else if ("ADD".equals(action)) {
                CD newCd = getCD(req);
                int index = buyList.indexOf(newCd);
                if ( index!=-1) {
                    CD cd = (CD) buyList.elementAt(index);
                    cd.setQuantity(cd.getQuantity()+newCd.getQuantity());
                } else {
                    buyList.addElement(newCd);
                }
            }
            session.putValue("shopping.cdCart", buyList);
            String url="/cd/Shop.jsp";
            ServletContext ctxt = getServletContext();
            RequestDispatcher rd = ctxt.getRequestDispatcher(url);
            rd.forward(req, res);
        }
    }
}

```

```

private CD getCD(HttpServletRequest req) {
    String theCd = req.getParameter("CD");
    String qty = req.getParameter("qty");
    StringTokenizer t =
        new StringTokenizer(theCd, "|");
    CD cd = new CD();
    cd.setArtist(t.nextToken());
    cd.setAlbum(t.nextToken());
    int p = Integer.parseInt(t.nextToken());
    cd.setPrice(p);
    cd.setQuantity(Integer.parseInt(qty));
    return cd;
}
}

```

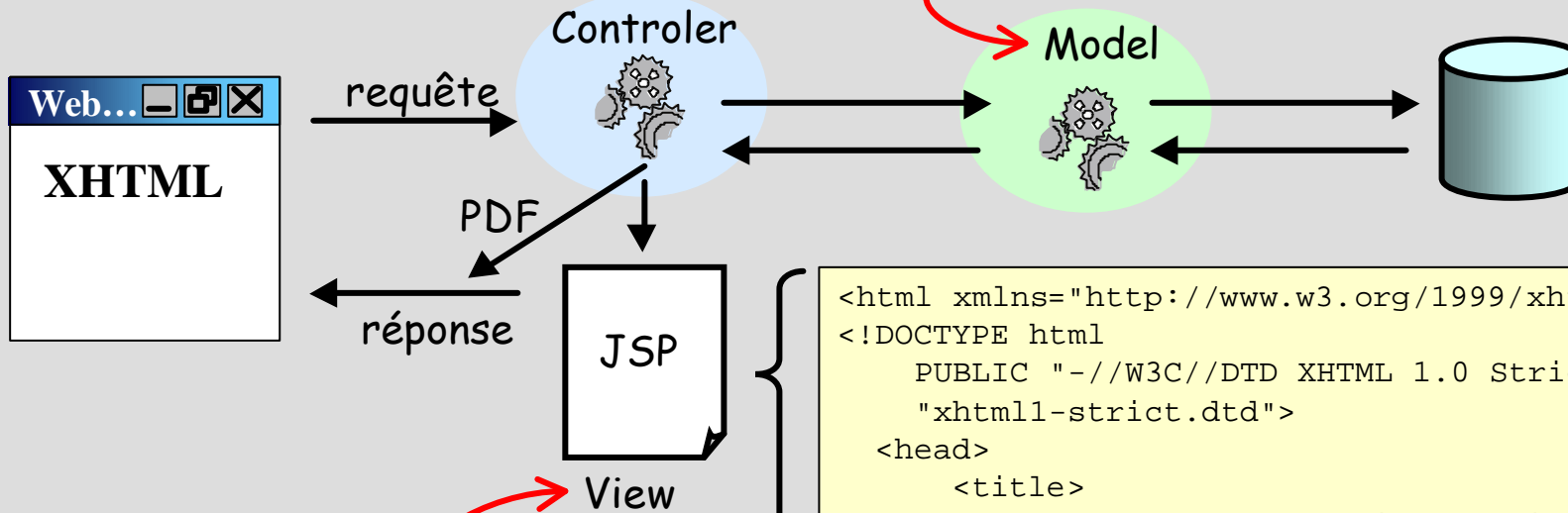
Model : logique d'accès aux données
View : page JSP qui assemble en HTML le résultat
Controler : servlet qui interprète les requêtes du client

- Décode les requêtes
- Récupère les données brutes
- Applique les transformations XSLT

Développeur Web

- Se connecte aux sources de données
- Accède aux données

Programmeur du Système d'Information



- Récupère les données calculées
- Les insère dans la page HTML

Web designer

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "xhtml1-strict.dtd">
<head>
  <title>
    <%=request.getAttribute("title")%>
  </title>
  <%=request.getAttribute("meta-datas")%>
</head>
<body background="fond.jpg">
  
  <%=request.getAttribute("toolbar")%>
  <%=request.getAttribute("content")%>
</body>
</html>
```