

Développement J2EE



Java **EE**TM

Les Servlets

Principes de base sur les servlets

Construction d'une servlet simple

Fonctionnement détaillé

Les sessions utilisateur

Authentification et habilitation

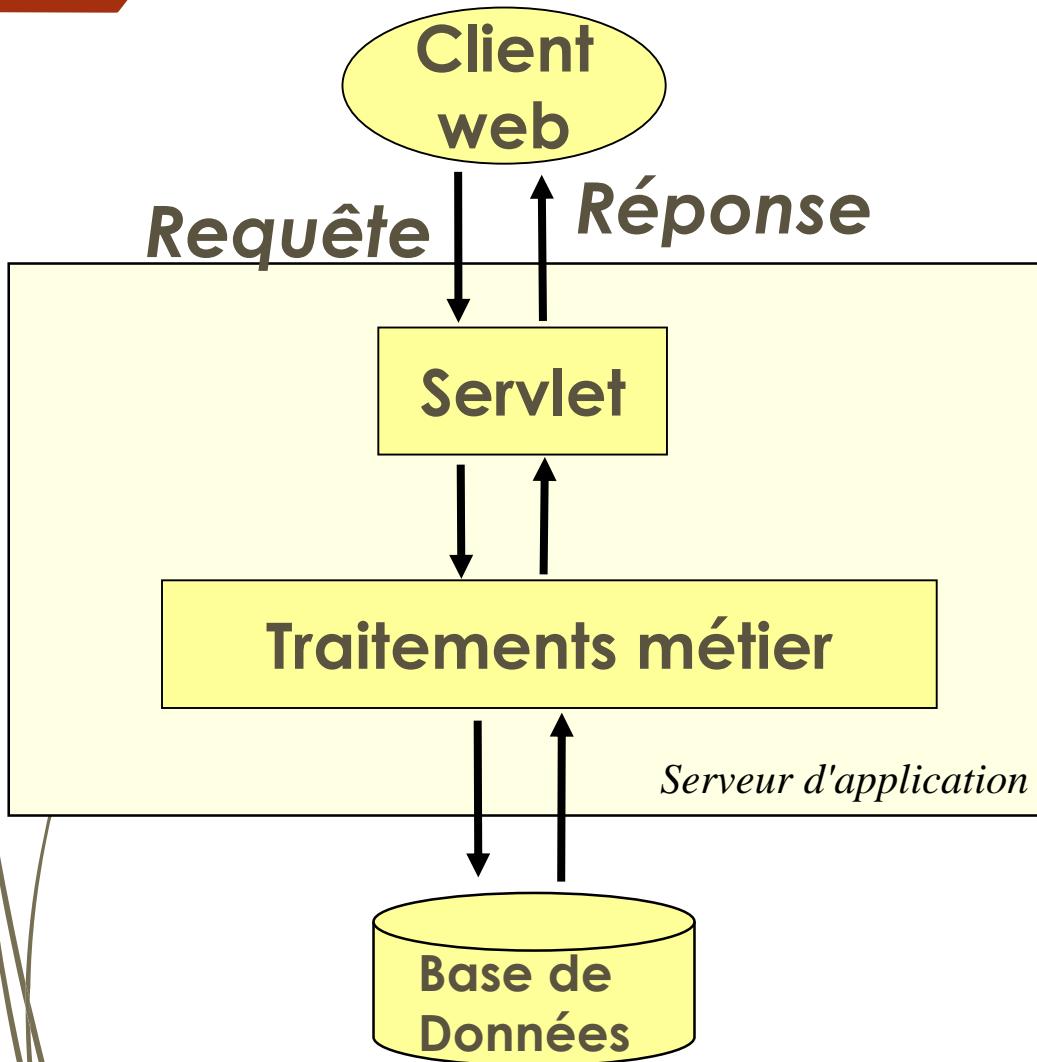
Mise d'informations dans le contexte

Gestion des formulaires

Définition

- Qu'est-ce qu'une Servlet ?
 - ➡ Code java, exécuté suite à un événement utilisateur
 - ❑ Connexion à une url donnée
 - ❑ On parle de 'requête utilisateur'
- La réponse http générée par une servlet ne contient que les fichiers statiques qui sont renvoyés au client.
 - ➡ code HTML, image...

Fonctionnement



La Servlet réceptionne une **requête** client.

La requête contient un ensemble d'informations constituant la demande du client (ex: url demandée, éventuellement paramètres de formulaire...).

Cette requête est traitée dans la servlet.

Une **réponse** est générée.

L'API Java Servlet

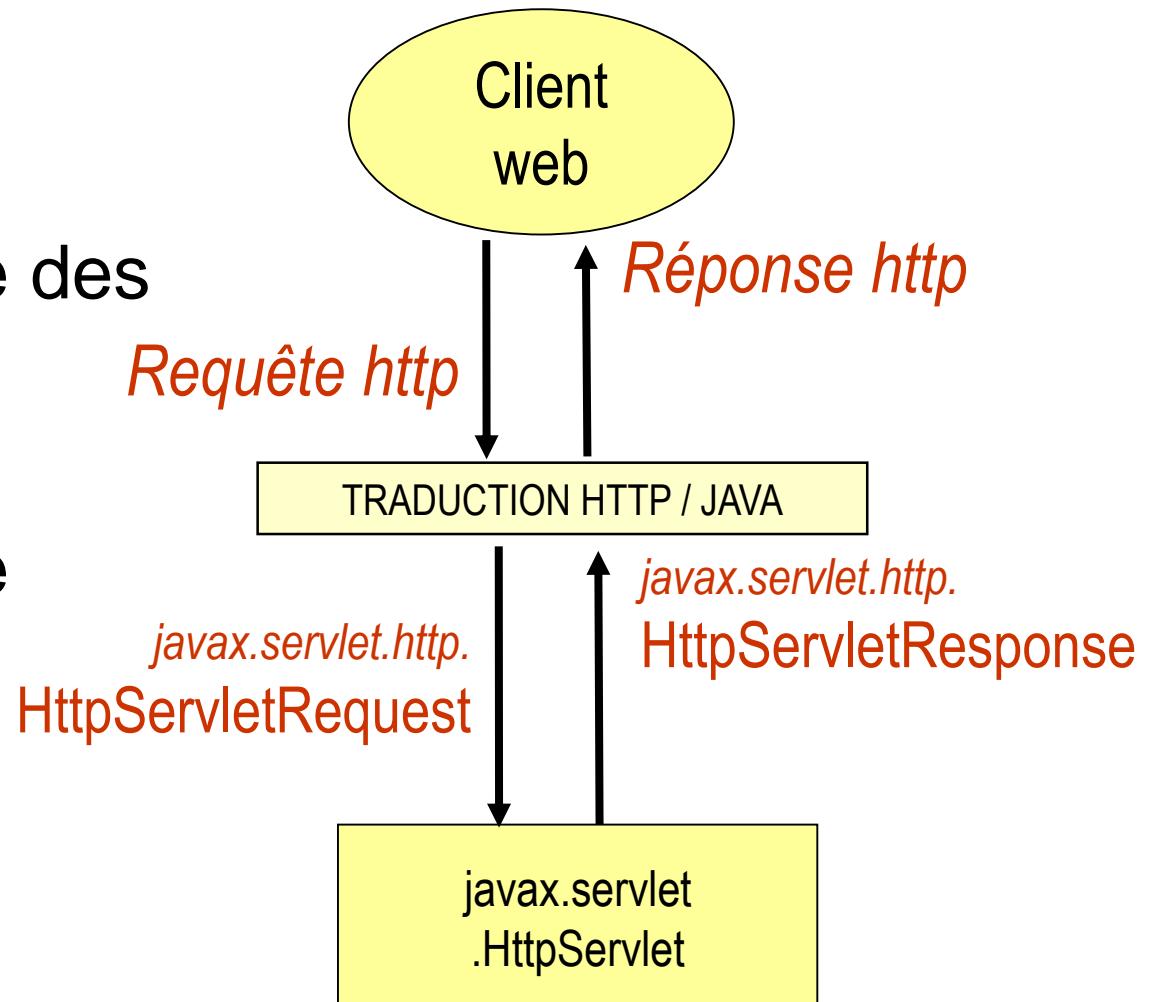
- javax.servlet (définition des objets (Interfaces) standard)
 - ▶ Package générique des servlets
 - ▶ GenericServlet, ServletRequest, ServletResponse...
- javax.servlet.http
 - ▶ spécialisation de javax.servlet pour le protocole HTTP
 - ▶ HttpServlet, HttpServletRequest, HttpServletResponse...



Il est très rare d'utiliser les classes du package javax.servlet directement.

Traduction Java/http

- La traduction est faite par le serveur d'application
 - Le client web perçoit les requêtes/réponses comme des objets http.
 - La servlet perçoit les requêtes/réponses comme des objets Java.



Créer une classe HelloServlet

7

- HelloServlet doit redéfinir la méthode héritée "service".

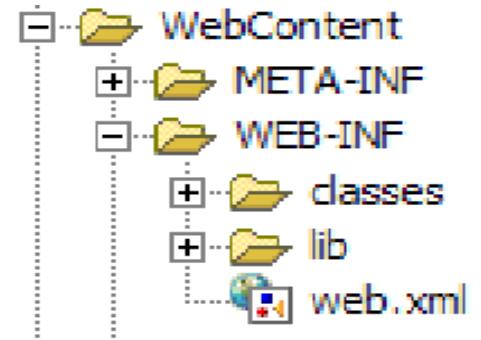
```
public class HelloServlet extends HttpServlet {  
    public void service(HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, IOException {  
        System.out.println("Hello World !");  
    }  
}
```



Redéfinir la méthode "service" est la plus simple des possibilités. Il en existe d'autres qui seront vues plus tard.

Déclarer dans le web.xml (1/2)

- Une application Web a la structure ci-contre
- Le fichier web.xml est le descripteur de déploiement Web.
- HelloServlet est déclarée dans le web.xml comme suit :



```
<web-app>
  <servlet>
    <servlet-name>helloServletName</servlet-name>
    <servlet-class>com.bankonet.servlet.HelloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>helloServletName</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

- A partir de JEE 6, la déclaration d'une servlet peut se faire dans la servlet avec l'annotation @WebServlet

Déclarer dans le web.xml (2/2)

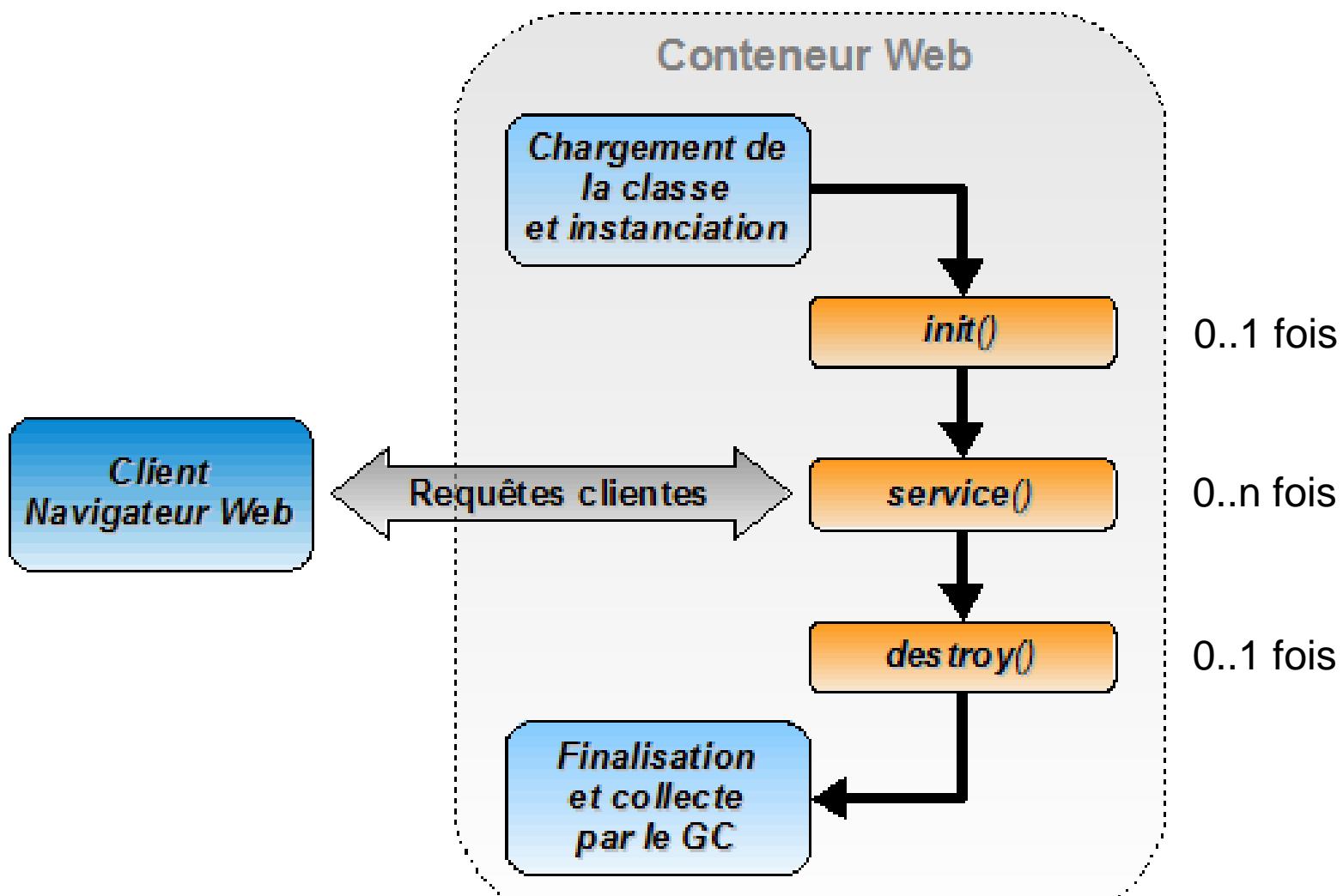
- A chaque application web correspond une url donnée.
 - ➡ ex : <http://www.bankonet.fr> : context
- Dans le web.xml, les servlets sont définies relativement à l'url de l'application web
 - ➡ A "/hello" correspondra <http://www.bankonet.fr/hello>
- Le serveur JEE fait appel à la méthode "service" de la classe définie dans le tag <servlet-class>

Déroulement des opérations

- Un client se connecte à l'url
`http://www.bankonet.fr/hello`
- Le serveur JEE identifie à quelle application web correspond l'url
- Dans le fichier `web.xml`, il identifie quelle est la classe servlet associée au mapping `"/hello"`
- La méthode "service" de `HelloServlet` est exécutée

Cycle de vie de la Servlet

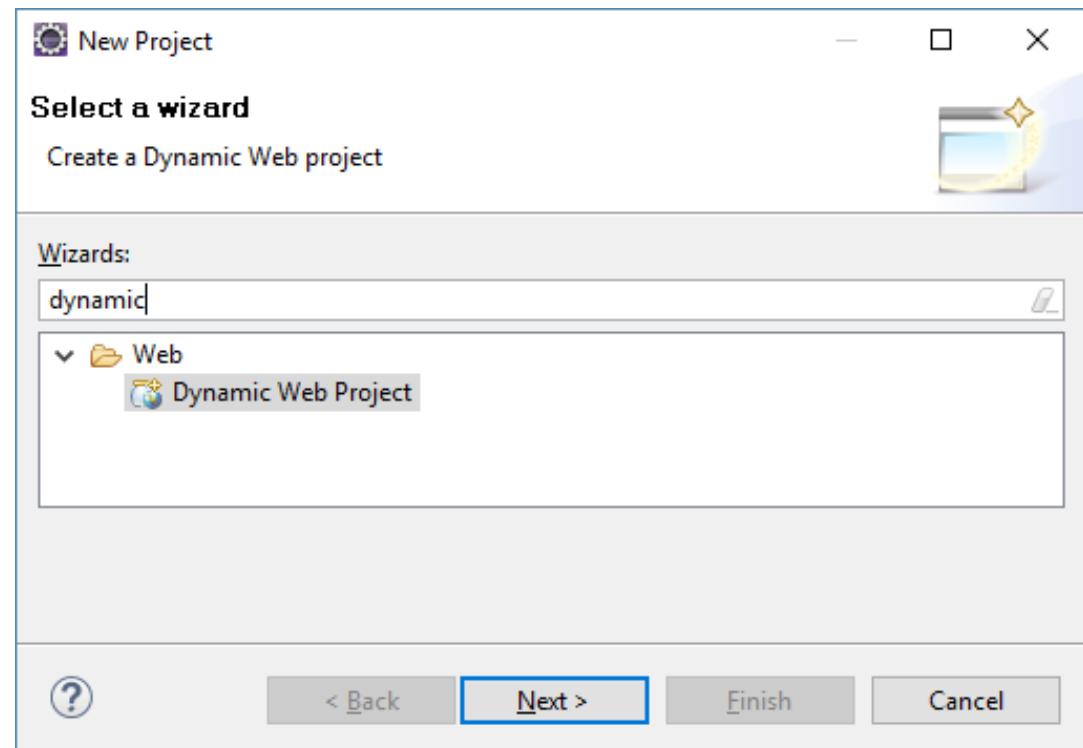
11



Servlets avec Eclipse

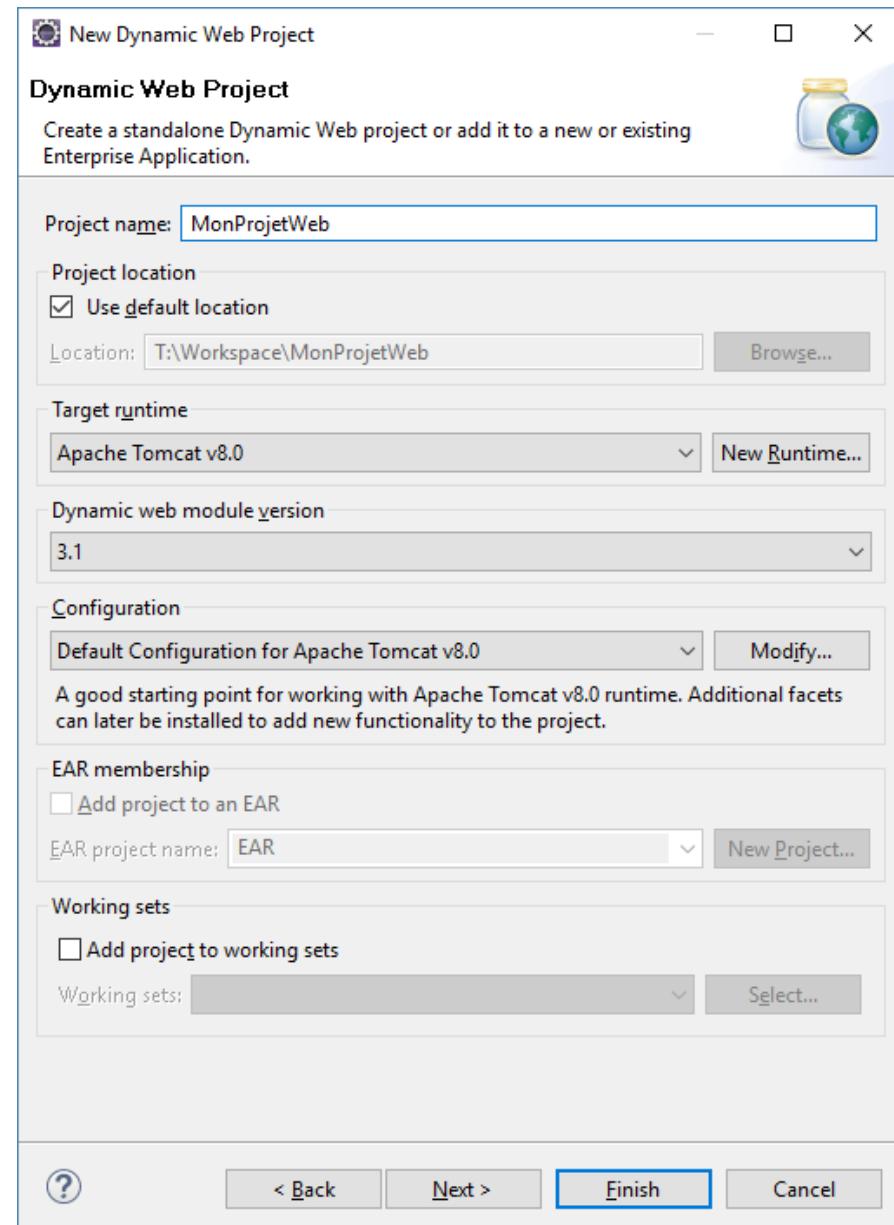
12

- Avant de réaliser une Servlet dans Eclipse vous devrez faire un projet ‘Dynamic Web Project’
- Pour que cela fonctionne vous devrez avoir une version ‘Eclipse For Java EE’
- Vous devez aussi installer un serveur J2EE (Tomcat, Jboss, ...) et faire le lien entre votre serveur et Eclipse



Projets WEBS avec Eclipse

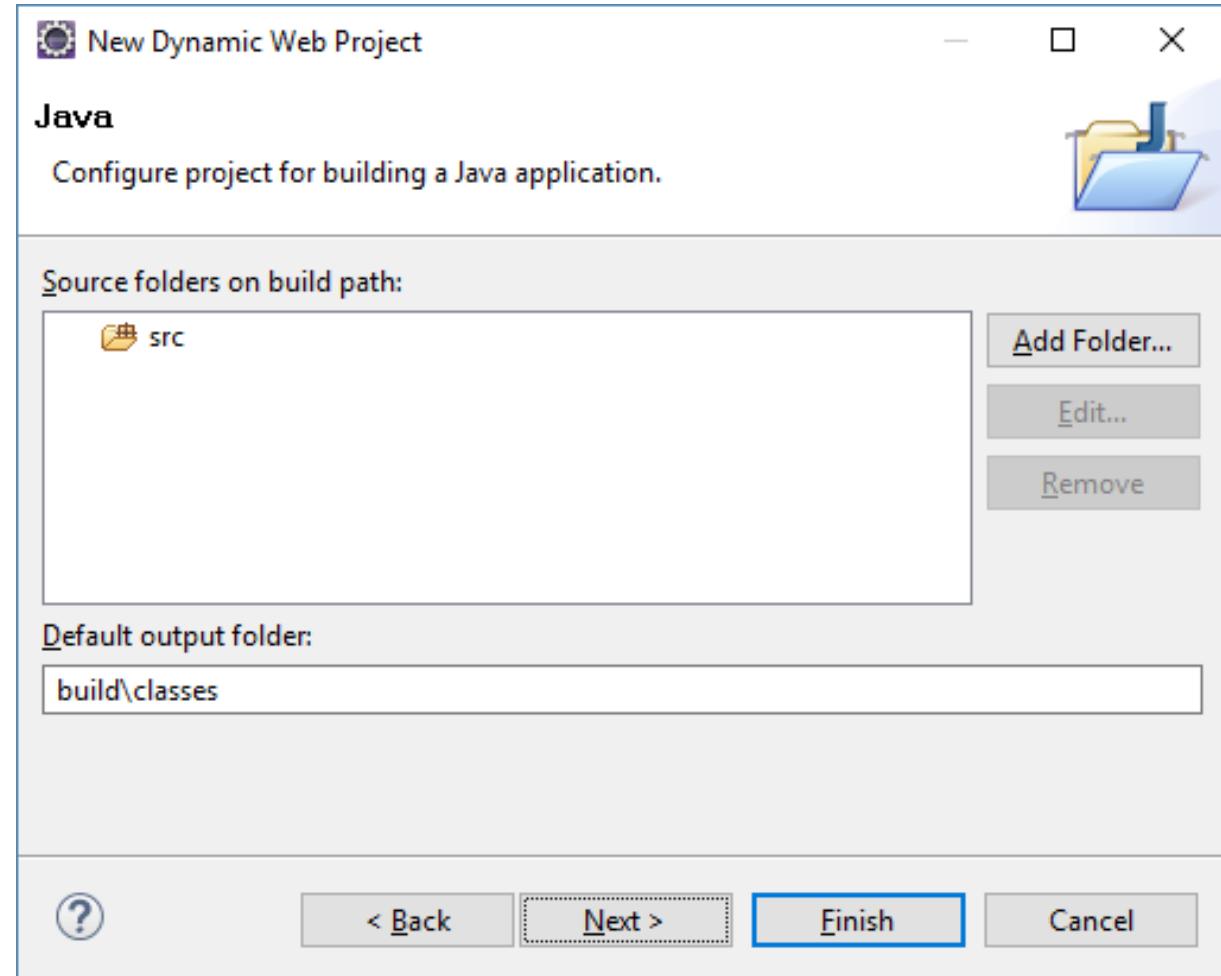
- Indiquez le nom de votre projet
- Indiquez le serveur J2EE cible
 - ➡ Si vous en avez aucun, il faudra en installer un
- Indiquez la version JEE de votre projet
 - ➡ Attention cette version sera en rapport avec
 - Votre serveur
 - Votre JVM



Projets WEBS avec Eclipse

14

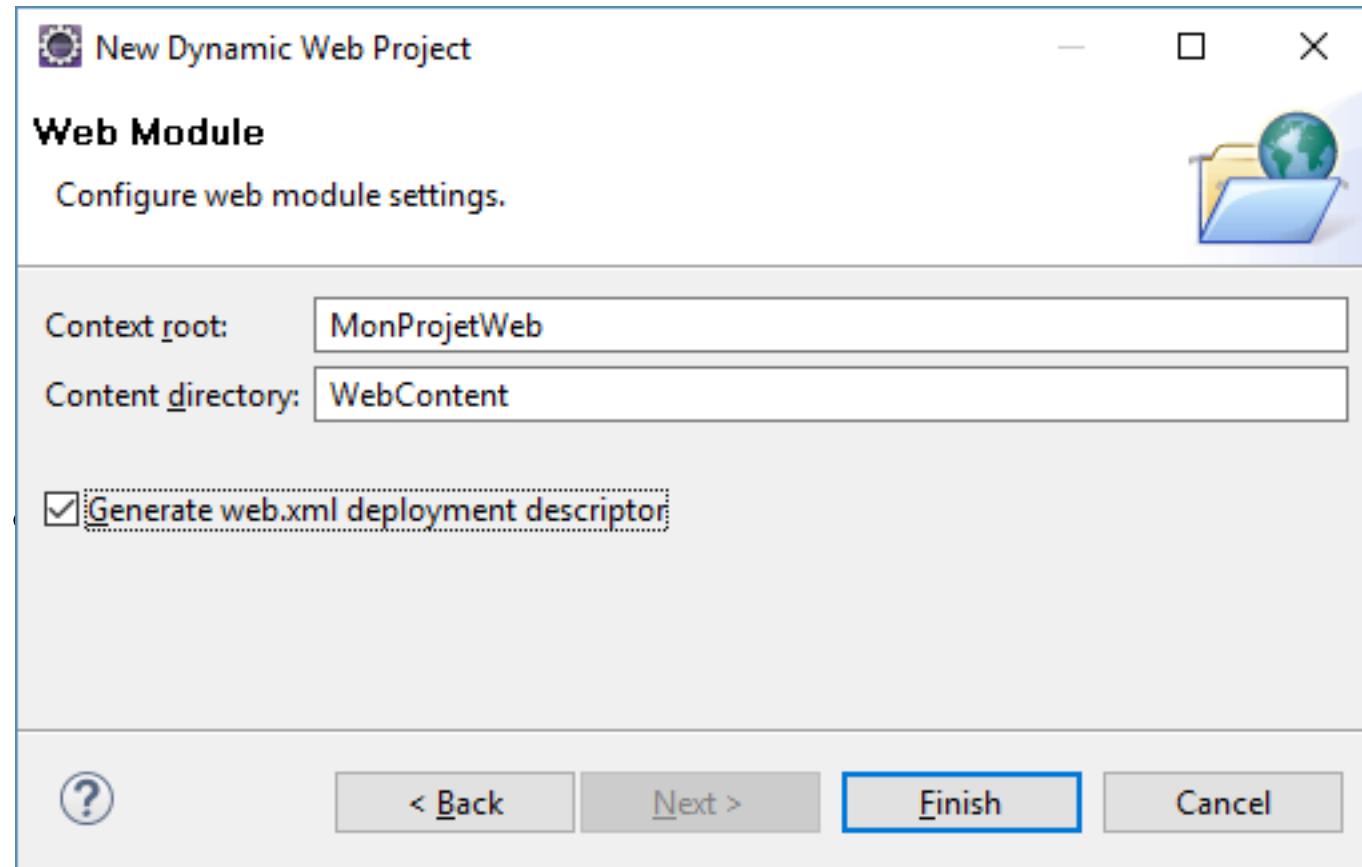
- Comme dans un projet standard vous pouvez indiquer le ou les répertoires qui vont contenir votre code Java



Projets WEBS avec Eclipse

15

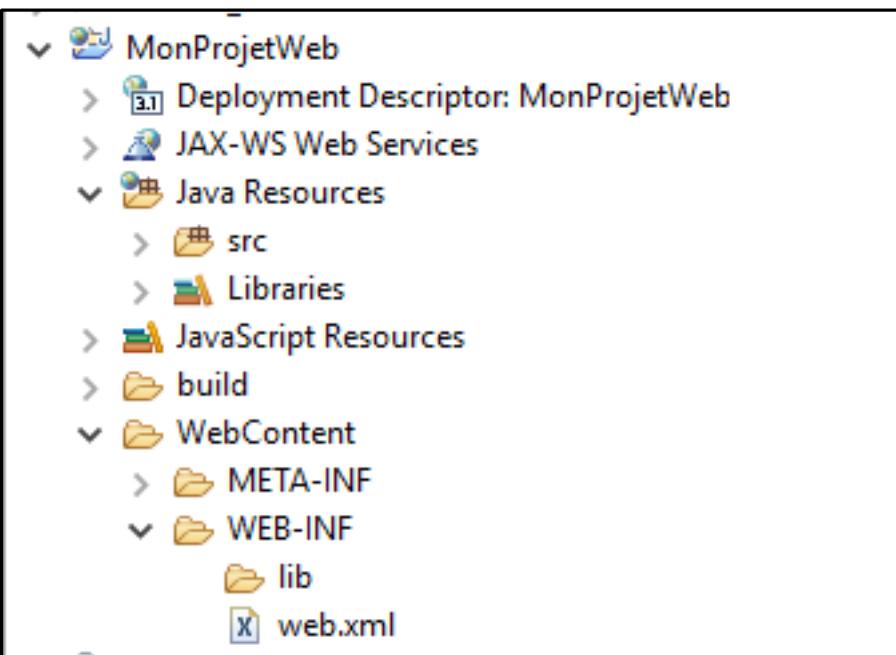
- Dans Context Root vous indiquerez ce qui doit apparaître dans les URLs d'appel à votre site web
 - ➡ Ici :
`http://localhost:8080/MonProjetWeb`
- Attention aux majuscules minuscules
- Dans Content Directory vous retrouverez tout ce qui n'est pas du code Java
 - ➡ Vos page Webs, CSS, Javascript, Images ...
- **Cochez** Generate web.xml afin d'avoir un fichier de configuration par défaut (fichier `web.xml`).



Projets WEBS avec Eclipse

16

- Vous passerez en perspective JEE



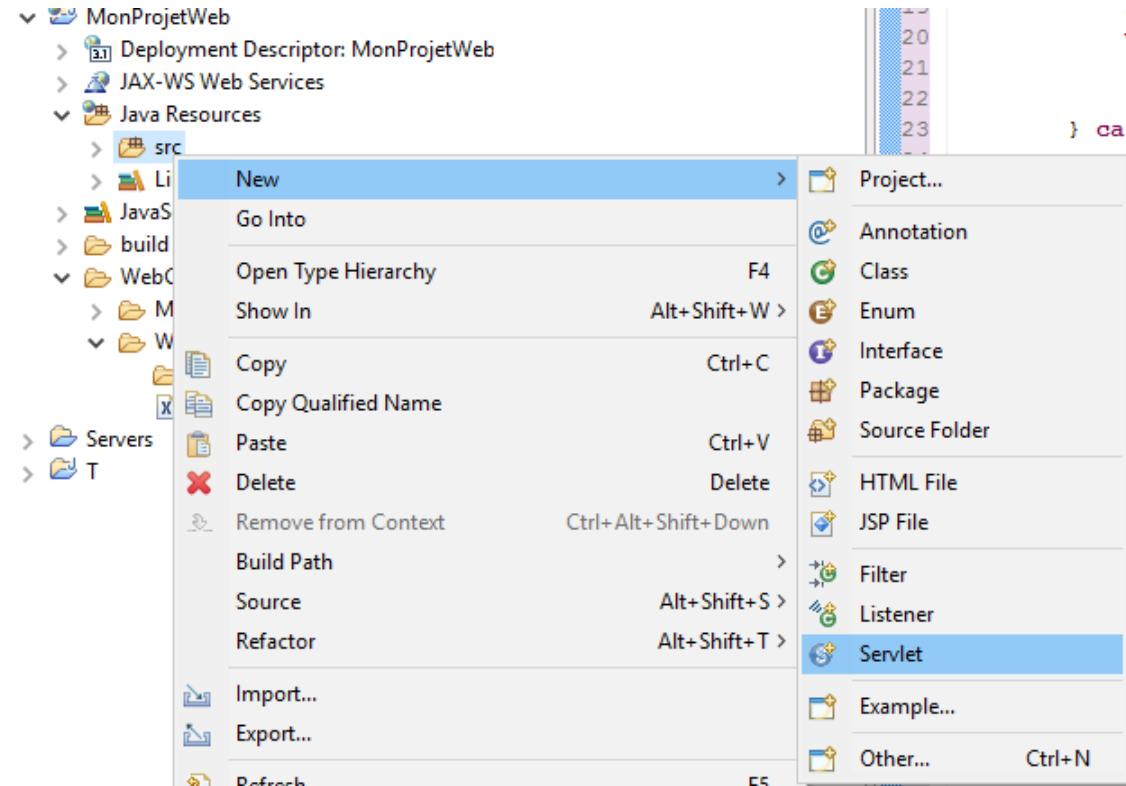
Servlets avec Eclipse

17

➤ Par le menu contextuel, en vous plaçant sur src, New -> Servlet

➡ En passant par l'assistant, la Servlet est automatiquement déclaré dans le descripteur de déploiement web.xml du projet Web

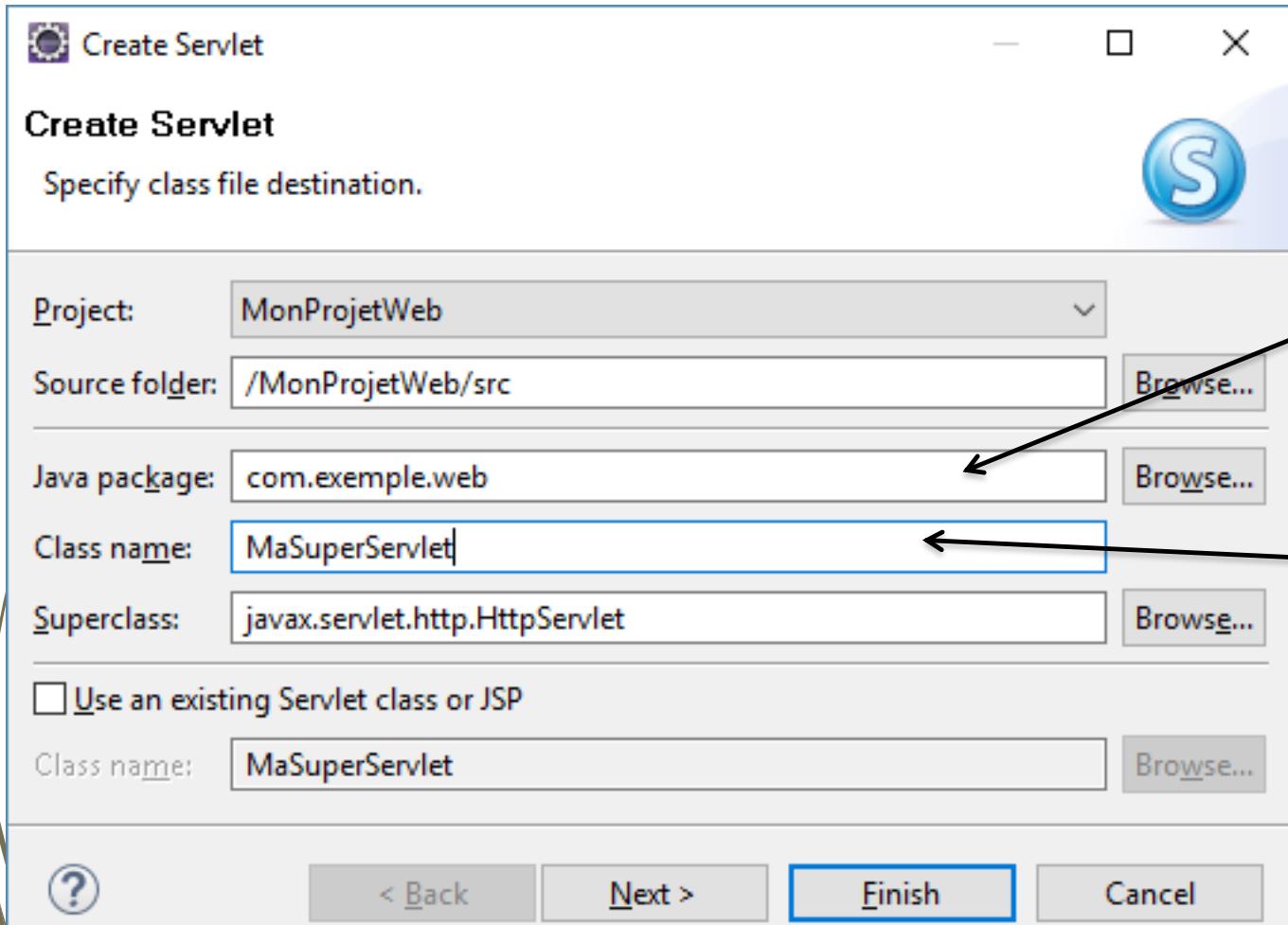
➡ Une Servlet est avant tout une classe Java



Servlets avec Eclipse

18

- Indiquez son nom de classe et le package de votre classe



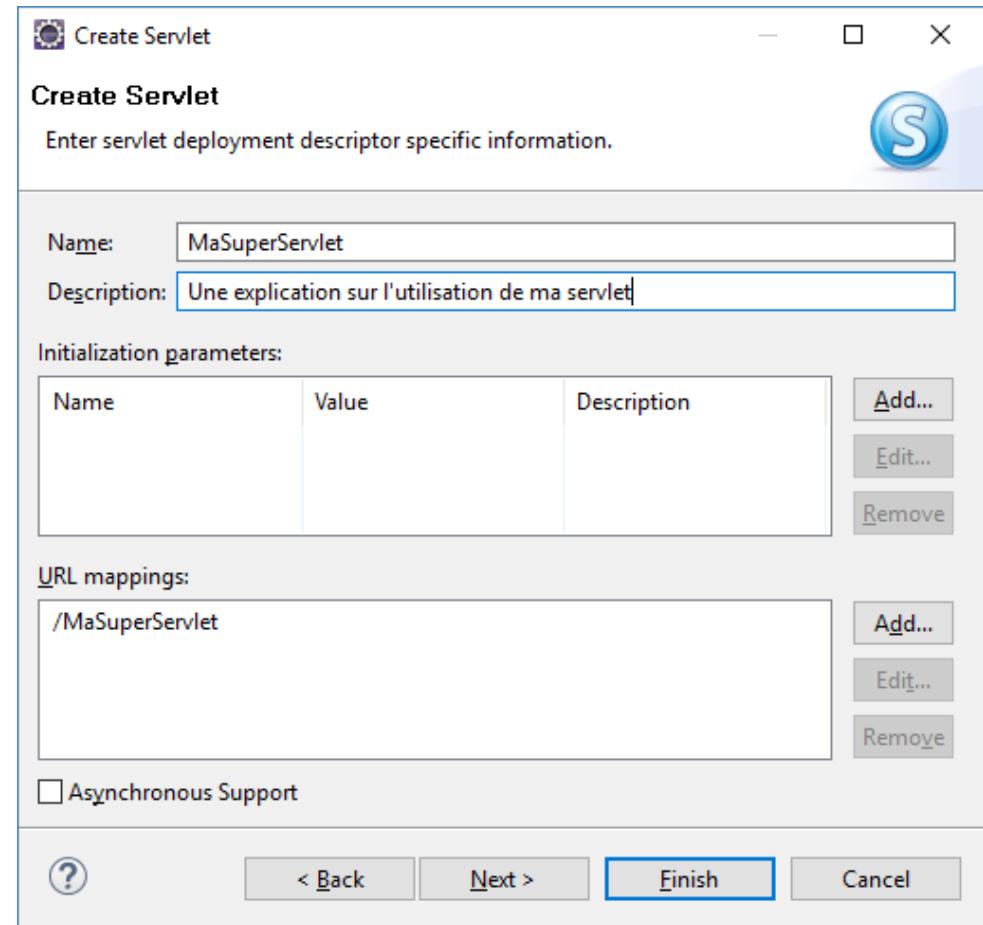
Package en minuscules
séparées par des points

Classe : première lettre en
majuscule, puis CamelCase

Servlets avec Eclipse

19

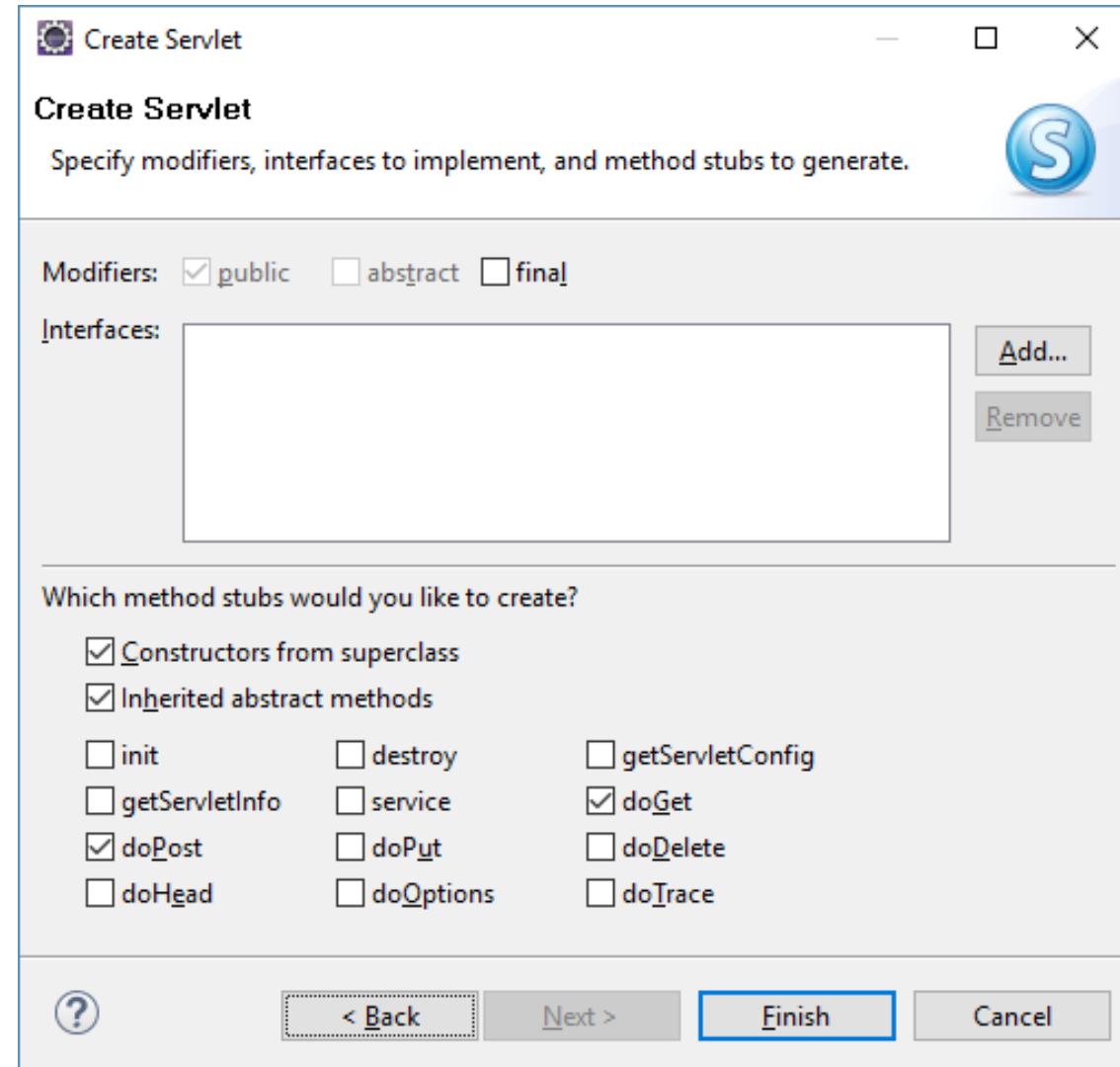
- Le nom de votre servlet sert de clef, il doit être unique dans toute votre application.
- Optionnel : indiquez une description
- Si vous souhaitez faire usage d'informations simples lors de la fabrication de la servlet indiquez le dans les paramètres d'initialisation
- Le plus important étant l'URL mappings (avec un s)
 - ➡ Permet d'indiquer les URL qui permettront d'atteindre votre servlet
 - ➡ Ici :
<http://localhost:8080/MonProjetWeb/MaSuperServlet>



Servlets avec Eclipse

20

- Une classe Servlet hérite de HttpServlet
 - ▶ Vous pouvez faire de l'héritage
 - ▶ Vous pouvez ajouter des interfaces
- A vous de choisir les méthodes qui vont servir réellement
 - ▶ **init** : appeler une fois et une seule par le conteneur après l'instanciation
 - ▶ **destroy** : appeler une fois et une seule par le conteneur lors de l'arrêt de l'application
 - ▶ **doGet**: appeler n fois par le conteneur lors des appels en HTTP get de votre servlet
 - ▶ **doPost**: appeler n fois par le conteneur lors des appels en HTTP post de votre servlet
 - ▶ **doXxx**: appeler n fois par le conteneur lors des appels en HTTP Xxx de votre servlet



Travaux pratiques

Réaliser les travaux pratiques suivants:

TP 00 : Réalisons une Servlet bonjour ensemble

Chargement de la servlet (1/3)

- Une seule fois par application Web
- Une instance de la Servlet est créée et servira pour toutes les requêtes des utilisateurs
- La méthode `init(ServletConfig)` est appelée à l'initialisation de la Servlet

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

- ▶ Exécutée uniquement au moment du chargement de la Servlet
- ▶ Mauvaise pratique : Surcharger cette méthode
 - En cas de surcharge de cette méthode toujours appeler la "super méthode"
- ▶ Bonne pratique : Surcharger **init()**, elle-même appeler par `init(ServletConfig)`
 - `getServletConfig()` pour accéder à la config servlet

Chargement de la servlet (2/3)

- Il existe 2 modes de chargement pour les Servlets :
 - ▶ Chargement au démarrage du serveur
 - ❑ La Servlet est chargée dès le démarrage du serveur.
 - ❑ Spécifié par le paramètre load-on-startup du descripteur de déploiement

```
<servlet>
    <servlet-name>helloServletName</servlet-name>
    <servlet-class>com.bankonet.servlet.HelloServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Chargement de la servlet (3/3)

➤ Chargement à la première demande

► la Servlet est chargée lors de la première utilisation :

- Un utilisateur fait appel à l'URL correspondante
- Un utilisateur soumet un formulaire vers cette servlet
- Par redirection à partir d'une autre servlet

► Comportement par défaut (tag load-on-startup non spécifié)

```
<servlet>
    <servlet-name>helloServletName</servlet-name>
    <servlet-class>com.bankonet.servlet.HelloServlet</servlet-class>
</servlet>
```



La servlet étant chargée à la première demande, le premier utilisateur sera servi plus lentement

Exécution d'une Servlet

25

- Une seule instance de servlet répond à toutes les requêtes client.
 - ➡ une thread pour chaque appel à service()
- DANGER : les variables d'instances (*static* nons *final*) sont partagées.

```
public class LoginServlet extends HttpServlet {  
    private String nom;  
    private String prenom;  
  
    public void service(req, resp)  
        throws ServletException, IOException {  
        nom = ...  
        prenom = ...  
    }  
}
```

```
public class LoginServlet extends HttpServlet {  
    public void service(req, resp)  
        throws ServletException, IOException {  
        String nom = ...  
        String prenom = ...  
    }  
}
```

Déchargement

- Déchargement de la Servlet
 - ▶ Arrêt des traitements puis destruction de l'instance de la Servlet
 - ▶ Cette opération est effectuée lors de l'arrêt 'normal' de l'application Web
 - ❑ Lors de l'arrêt du serveur, toutes les applications sont arrêtées.

```
public void destroy()
```

Méthodes d'appel

- Méthode **doGet** : ne sera appelée que dans le cas d'une requête http de type GET
 - Requête Get : appel par URL (lien `...`. Ou par formulaire en méthode get
- Méthode **doPost** : ne sera appelée que dans le cas d'une requête http de type POST
 - Requête Post : validation d'un formulaire html en méthode post.
- Méthode **doPut**, **doDelete**, **doXxx** : ne sera appelée que dans le cas d'une requête http de type Xxx
 - Requêtes envoyé par un programme. Le doPut est souvent utilisé en Web Service REST

Méthodes d'appel

- Méthode **service** : appelée quel que soit le type de la requête.
 - Pratique si vous ne voulez pas différencier vos appels
- Note : les méthodes service, doPost, doGet sont toujours appelées suite à un événement utilisateur ⇔ une requête HTTP.
On parle de 'méthodes de rappel'.



Si la méthode service est redéfinie, doGet et doPost ne seront jamais appelés.

Renvoi vers une autre ressource

- La Servlet doit pouvoir rediriger le traitement vers une autre ressource
 - ➡ Typiquement une page html, une page jsp et par extension toute ressource qui peut être identifiée par une url.
- Il existe deux méthodes : par **forward** ou par **redirect**.
 - ➡ **Forward** : le navigateur client n'est pas informé du passage de la servlet vers la ressource demandée. Il ne verra que le résultat 'final'.
 - ➡ **Redirect** : le navigateur client est informé du passage de la servlet vers la ressource demandée.

Renvoi vers une autre ressource

30

- Un objet RequestDispatcher permet d'effectuer le forward.

```
public void service(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    //...
    RequestDispatcher disp = request.getRequestDispatcher("/login.html");
    disp.forward(request, response);
}
```

- ➡ le navigateur client n'est pas informé du passage de la servlet vers la ressource demandée.
- ➡ Dans le navigateur, **la barre d'url affiche l'url de la servlet** (et non celle de login.html).

Renvoi vers une autre ressource

- La réponse peut demander une redirection automatique

```
public void service(HttpServletRequest req, HttpServletResponse  
response) throws ServletException, IOException {  
    //...  
    response.sendRedirect(req.getContextPath() + "/login.html");  
}
```

- Le navigateur client est informé du passage de la servlet vers la ressource demandée.
 - Dans le navigateur, **la barre d'url affiche l'url** de login.html.
 - Une **nouvelle requête** http est créée. Les informations présentes dans la précédente requête sont perdues.

Quizz (1/2)

32

- Dans ma servlet AccueilServlet, j'ai redéfini la méthode init(ServletConfig) comme suit :

```
public void init(ServletConfig servconfig) throws ServletException {  
    System.out.println("initialisation de AccueilServlet");  
}
```

- Ma servlet ne fonctionne pas. Quelle en est la cause ?
- Une servlet peut-elle dessiner une image ?

Quizz (2/2)

➤ Dans une servlet, il y a la possibilité de redéfinir plusieurs méthodes de traitement. Quelles sont les possibilités qui vous paraissent pertinentes ?

- Redéfinir service, doGet et doPost
- Redéfinir service
- Redéfinir doPost seul
- Redéfinir doGet et doPost

| Oui | Non |
|--------------------------|--------------------------|
| <input type="checkbox"/> | <input type="checkbox"/> |

Travaux pratiques

Réaliser les travaux pratiques suivants:

TP 01 : Réalisons une Servlet Client et une servlet Compte

Travaux pratiques

35

Les Clients

| Nom | Prenom | Age | |
|--------|--------|-----|------------------|
| Durand | Paul | 34 | Voir ses comptes |
| Dupont | Jean | 52 | Voir ses comptes |
| Martin | Sophie | 23 | Voir ses comptes |
| Tooto | Toto | 89 | Voir ses comptes |

Les Comptes de ce client

| Solde | Taux | Seuil |
|--------|-------------|--------------|
| 2056.0 | Pas de Taux | Pas de Seuil |
| -250.0 | Pas de Taux | -1000.0 |

La session : principe

- Certaines informations doivent pouvoir être mises à disposition pendant toute la durée de connexion d'un utilisateur.
- Exemple classique de l'utilisation des sessions : un panier électronique.
 - ➡ Un objet panier doit être associé à chacun des clients.
 - ➡ Le panier doit être consultable jusqu'à déconnexion de l'utilisateur.

Les sessions : mode de fonctionnement (1/3)

37

- Lors de la première requête http, le serveur ouvre une session pour le client web.
 - Le serveur attribue au client un identifiant de session.
 - L'identifiant sera par la suite stocké sur le poste utilisateur (généralement sous forme de cookie).
- Pour chacune des requêtes suivantes, le client doit présenter son identifiant de session.
 - Si un cookie est utilisé, il est automatiquement joint à chaque requête client.

Les sessions : mode de fonctionnement (2/3)

38

- Le serveur récupère l'identifiant de session du client.
- Il existe une zone mémoire sur le serveur où sont stockées toutes les sessions utilisateur.
 - ➡ Les sessions sont indexées par leur identifiant.
- Le serveur récupère l'objet session en cours.
 - ➡ Il est de type javax.servlet.http.HttpSession

Les sessions : mode de fonctionnement (3/3)

39

- A partir de la requête http, le serveur crée un objet HttpServletRequest.
 - ▶ Porte les informations de la requête http
 - ▶ Mis à disposition dans les servlets
- Le serveur place la session dans l'objet request.
 - ▶ La session doit être récupérée comme suit

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

public class HelloServlet extends HttpServlet {
    public void service(HttpServletRequest request, HttpServletResponse resp)
        throws ServletException, IOException {
        HttpSession session = request.getSession(true);
        // ...
    }
}
```

Les sessions : récupération

- La méthode `request.getSession(boolean)` permet de récupérer une session.
 - `true` : si le client n'a pas de session attribuée, le serveur lui créé une session.
 - `false` : si le client n'a pas de session attribuée, la méthode renvoie '`null`'.

```
public void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    HttpSession session = req.getSession(true);
    // ...
}
```

Fermeture de session

41

- Il est possible d'invalider une session comme suit :

```
public void service(HttpServletRequest request, HttpServletResponse res)
throws ServletException, IOException {

    HttpSession session = request.getSession(true);

    session.invalidate();

    // ...

}
```

- La session du client est supprimée après un délai d'inactivité
 - Positionné par défaut sur le serveur d'application
 - Il est possible de redéfinir cette valeur dans le web.xml :

```
<session-config>
    <session-timeout>2</session-timeout><!-- en minutes -->
</session-config>
```

web.xml

Bonnes pratiques

- Les objets session sont stockés en permanence sur le serveur.
 - ▶ Dans la mesure du possible, il faut stocker un minimum d'informations dans la session.
- Les navigateurs Web peuvent refuser l'utilisation de cookies.
 - ▶ Paramétrable dans les options du navigateur.
 - ▶ Si les utilisateurs d'une application utilisent potentiellement un navigateur sans cookies, il faut encoder les URL comme suit :
 - ▶ <http://www.bankonet.fr/Accueil;JSessionId=12429>

Les scopes

- Il existe 3 scopes dans lesquels nous pouvons temporairement stocker de l'information lorsque cela est requis :
 - ▶ Le scope commun à toutes les servlets d'une application web (objet ServletContext)
 - ▶ Le scope de session (objet HttpSession)
 - ▶ Le scope de la requête (objet HttpServletRequest)
- Ils proposent les mêmes méthodes setAttribute, getAttribute, removeAttribute.
- Choisir le plus adéquat
 - ▶ Pour ne pas conserver l'information plus que nécessaire
 - ▶ Pour libérer les ressources (mémoire) dès que possible

Les scopes : cycle de vie des attributs

- Un *attribut* d'un scope est un objet
 - ➡ Classiquement utilisé pour le transfert d'informations du contrôleur (servlet) vers la vue (page JSP).
- Enregistrer et récupérer des informations d'un scope
 - ➡ `void setAttribute(String, Object)`
 - ➡ `Object getAttribute(String)`
- Supprimer une entrée dans un scope
 - ➡ `void removeAttribute(String)`

Les scopes : requête

- Les informations présentes dans le scope de requête sont utilisables pendant un aller-retour client serveur.
 - Après, la requête est automatiquement détruite par le serveur d'applications.
 - Très utilisé pour communiquer de la servlet vers la page jsp.
 - Chaque client a son propre scope de requête.

```
public void service(HttpServletRequest request, HttpServletResponse resp)
throws ServletException, IOException {
    //...
    request.setAttribute("erreurMessage", "Un message d'erreur");
}
```

Les scopes : session

- Les informations présentes dans le scope de session sont utilisables jusqu'à la suppression de la session utilisateur.
 - ▶ Par déconnexion ou par timeout.
 - ▶ Pour garder de bonnes performances, une bonne pratique est de stocker un minimum de choses dans la session.
 - ▶ Chaque client a son propre scope de session.
 - ▶ Attention : vous ne devriez placer en session que des objets sérialisables.

```
public void service(HttpServletRequest request, HttpServletResponse  
resp) throws ServletException, IOException {  
    //...  
    HttpSession session = request.getSession(true);  
    session.setAttribute("livre", monLivre);  
}
```

Les scopes : application

47

- Les informations présentes dans le scope de servlet sont utilisables tant que l'application est lancée.
 - ▶ Permet de stocker globalement des informations.
 - ▶ Durée de vie
 - jusqu'à l'arrêt du serveur
 - jusqu'au rechargement de l'application (en cas de modification)
 - ▶ Le scope de servlet est partagé par tous les utilisateurs
 - ▶ Attention : placez y des objets thread safe

```
public void service(HttpServletRequest request, HttpServletResponse  
resp) throws ServletException, IOException {  
    //...  
    request.getServletContext().setAttribute("now", new Date());  
}
```

Les contextes : récapitulatif

48

Où a-t-on besoin de l'information ?

| | Plusieurs pages | Les pages liées par le traitement d'une même requête |
|-----------------------|-----------------|--|
| Tous les utilisateurs | ServletContext | ServletContext |
| L'utilisateur courant | HttpSession | HttpServletRequest |

Qui utilise l'information ?

Appel d'une servlet par un formulaire html

49

- L'url de retour est positionnée dans la balise d'en-tête du formulaire.
- La requête est envoyée au serveur quand l'utilisateur clique sur le bouton de type "submit"

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Bienvenu sur Bankonet</title>
</head>

<body>
    <h1>Bankonet</h1>
    <form method="POST" action="/traiterLogin">
        Login: <input type="text" name="j_username" /><br />
        Password: <input type="password" name="j_password" /><br />
        <input type="submit" value="Connexion" />
    </form>
    <br />
    <br />
    Bankonet by SQLI©
</body>
</html>
```

The screenshot shows a web page titled "Bankonet". It contains a login form with fields for "Login" and "Password", and a "Connexion" button. Below the form, the text "Bankonet by SQLI©" is displayed.

Récupération de paramètres dans la Servlet

50

- Les paramètres sont des chaînes de caractères portées par la requête
- Ils peuvent être récupérés dans les méthodes service, doGet, doPost... d'une servlet.

```
public void service(HttpServletRequest request, HttpServletResponse res)
throws ServletException, IOException {
    String monPrenom = request.getParameter("prenom");
    String monNom = request.getParameter("nom");
    String monAdresse = request.getParameter("adresse");
    // ...
}
```

```
<form method="POST" action="/traiterLogin">
    Nom : <input type="text" name="prenom" /><br />
    Prenom : <input type="text" name="nom" /><br />
    Adresse: <input type="text" name="adresse" /><br />
</form>
```

L'importance de POST pour les formulaires

51

- Si l'argument method="post" n'est pas positionné dans le tag <form>, la validation de formulaire renverra une requête de type Get
- Comme dans l'exemple ci-dessous :

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Bienvenu sur Bankonet</title>
</head>

<body>
    <h1>Bankonet</h1>
    <form action="/traiterLogin">
        Login: <input type="text" name="j_username" /><br />
        Password: <input type="password" name="j_password" /><br />
        <input type="submit" value="Connexion" />
    </form>
    <br />
    <br />
    Bankonet by SQLI©
</body>
</html>
```

The screenshot shows a web page titled "Bankonet". It features a login form with two input fields labeled "Login:" and "Password:", and a submit button labeled "Connexion". Below the form, the text "Bankonet by SQLI©" is displayed.

L'importance de POST pour les formulaires

52

- Conséquence : le mot de passe apparaît en clair dans l'url ...
↔ passage en GET

```
http://localhost:8080/bankonet/traiterLogin?j_username=Donald&j_password=Pluto
```

- Pour éviter cela, une validation de formulaire doit toujours renvoyer une requête "POST"

```
<form method="POST" action="/processLogin">  
...  
</form>
```

- Remarque : dans tous les cas le seul moyen de sécuriser ses informations c'est d'utiliser
 - Du SSL ↔ faire de l'https
 - Ou du VPN

Paramètre ou Attribut

53

- Il ne faut pas confondre 'attribut de requête' et 'paramètre de requête'
- Un **paramètre** est une chaîne de caractères transmise par la requête
 - ▶ Après soumission d'un formulaire
 - ▶ Après appel d'URL
- Un **attribut** est un objet qui a été placé explicitement dans le contexte de requête
 - ▶ À partir d'une précédente servlet ou page jsp

```
public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    String monId = req.getParameter("identifiant");
    Client monClient = (Client)req.getAttribute("client");
}
```

Paramètre ou Attribut

- **Un paramètre (vient du client)**
 - ▶ C'est toujours une String
 - ▶ Sera toujours dans le scope request
- **Un attribut (placé par vos soins)**
 - ▶ C'est toujours un Object (il faudra donc le caster)
 - ▶ Sera dans un des scopes (request, session, application)
- **Vous ferez l'erreur au moins une fois :**
 - ▶ Appel à `request.getParameter("clef")` au lieu de `request.getAttribute("clef")`

Développement JEE -Authentification et Habilitation

55

- Authentification
 - ▶ Connaître l'utilisateur de l'application
 - Personnaliser son IHM (langue, thème, ...)
 - Personnaliser ses données
 - ▶ Gérer via un couple login/mot de passe
- Habilitation
 - ▶ Restreindre les fonctionnalités de l'utilisateur
 - ▶ Gérer via des rôles affectés à l'utilisateur
 - Donc pas d'habilitation sans authentification
- Les données d'authentification et d'habilitation peuvent être stockées dans différentes "base de données" utilisateurs
 - ▶ LDAP
 - ▶ Base de données
 - ▶ Fichier

Développement JEE -Authentification et Habilitation

56

- Le mécanisme d'authentification/habilitation peut se faire "à l'ancienne" ...
 - ▶ Création d'une session utilisateur après saisie et vérification du login/mot de passe
 - ▶ Stockage manuel d'un objet utilisateur dans la session
 - ▶ Vérification des droits d'accès aux fonctionnalités
 - ▶ Le tout dans le code Java ➔ Lourd
- ... Ou en appelant le mécanisme d'authentification par formulaire
 - ▶ Configuration dans le descripteur de déploiement web.xml ➔ Beaucoup plus simple

Développement JEE -Authentification et Habilitation

57

- Une "base de données" utilisateurs est appelée "Realm" sur un serveur d'application
- Elle se configure au niveau du serveur d'application
 - ▶ JNDIRealm ➔ Annuaire LDAP
 - ▶ JDBCRealm ➔ Base de données accédé en JDBC
 - ▶ DataSourceRealm ➔ Base de données accédé en JDBC via un DataSource
 - ▶ UserDatabaseRealm ➔ Fichier contenant les utilisateurs et leurs rôles
 - ▶ JAASRealm ➔ Utiliser le framework JavaSE JAAS (nécessite d'avoir écrit et déployé sur le serveur d'application un module JAAS)
 - ▶ ...
 - ▶ Possibilité d'implémenter son propre Realm (non normé entre les différents serveur d'application)

Développement JEE -Authentification et Habilitation

58

- Configuration du descripteur de déploiement web.xml
 - Déclarer les rôles applicatifs via la balise <security-role>

```
<security-role>
    <role-name>client</role-name>
</security-role>
<security-role>
    <role-name>admin</role-name>
</security-role>
```

➤ Configuration du descripteur de déploiement web.xml

► Déclarer les contraintes de sécurité sur l'application via la balise `<security-constraint>`

□ Déclarer des URL pattern

- Ces URL pattern doivent correspondre à des mapping de servlet

□ Contraindre la method HTTP. Optionnel

□ Définir les rôles ayant accès ces URL pattern. Optionnel

□ Définir le niveau de garantie du transport des données. Optionnel

- **NONE** ➔ Aucune contrainte (valeur par défaut)

- **INTEGRAL** ➔ Les données ne peuvent être modifiées (SSL)

- **CONFIDENTIAL** ➔ Les données ne peuvent être modifiées ou observées (SSL + encryptage)

➤ Configuration du descripteur de déploiement web.xml

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Servlets under /espaceclient require role client</web-resource-name>
        <url-pattern>/espaceclient/*</url-pattern>
        <http-method>POST</http-method><!-- Optionnel -->
        <http-method>GET</http-method><!-- Optionnel -->
    </web-resource-collection>

    <auth-constraint>
        <role-name>client</role-name>
    </auth-constraint>

    <user-data-constraint><!-- Optionnel -->
        <transport-guarantee>NONE, INTEGRAL ou CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Développement JEE -Authentification et Habilitation

61

➤ Configuration du descripteur de déploiement web.xml

- ▶ Déclarer le mécanisme d'authentification par formulaire via la balise `<login-config>`
- ▶ `<auth-method>`
 - **BASIC** ➔ Boîte de dialogue gérée par le navigateur
 - **CLIENT-CERT** ➔ Authentification SSL basé sur un certificat client
 - **FORM** ➔ Authentification via une page créée par le développeur (personnalisation possible, intégration de la charte graphique)
 - Le formulaire HTML doit soumettre ses données à la servlet spécifique `j_security_check` (cette servlet existe, pas besoin de la déclarer dans le `web.xml`)
 - Le formulaire HTML doit contenir deux champs login et mot de passe respectant ces noms : `j_username` et `j_password`

```
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/jsp/login.jsp</form-login-page>
        <form-error-page>/jsp/login-failed.jsp</form-error-page>
    </form-login-config>
</login-config>
```

Sécurité programmatic

- Vous pouvez récupérer vos informations d'habilitation via la `HttpServletRequest`
 - ▶ `getRemoteUser()` : retourne l'identifiant de l'utilisateur
 - ▶ `getUserPrincipal()` : retourne l'objet de type `Principal` représentant l'utilisateur connecté
 - ▶ `isUserInRole(String role)` : permet de vérifier automatiquement que l'utilisateur possède le rôle demandé

Quizz (1/2)

63

- Comment tester si une session existe ?
- Faire un bref schéma rappelant le cheminement d'une requête http depuis son envoi sur un navigateur web jusqu'au traitement par la servlet AccueilServlet (montrer le passage 'trame http -> Java' et la récupération de session)

Quizz (2/2)

64

- Il a été vu que la servlet est instanciée une seule fois et que cet unique objet va répondre à toutes les requêtes utilisateur.
- Un utilisateur envoie une requête gourmande qui va nécessiter un traitement de 10 secondes de temps machine.
- Cela veut-il dire que pendant 10 secondes aucun autre utilisateur ne sera servi pendant ce temps ?

Java Server Page

65

- Première approche
- Structure d'une page jsp
- Page d'accueil, page d'erreur
- Inclusion de page
- La bibliothèque de balises JSP
- JSTL
- Ecrire ses propres balises

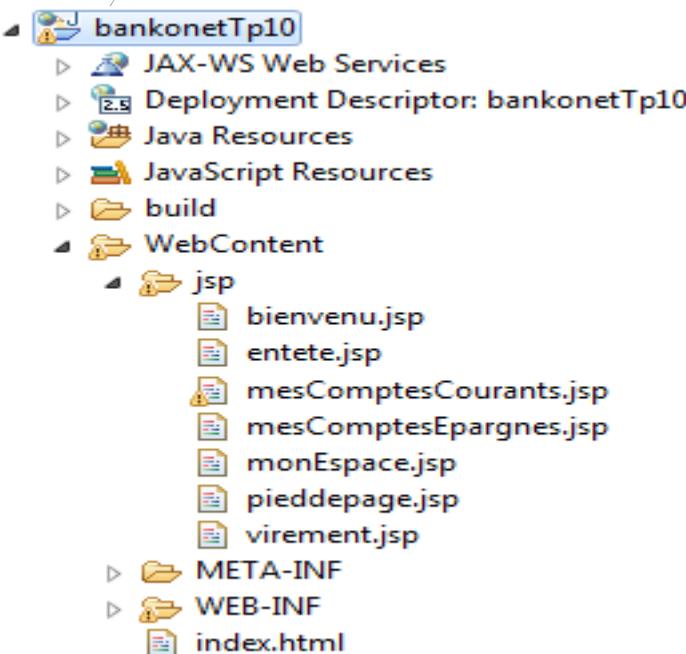
Définition (1/2)

- JSP : JavaServer Pages
 - ▶ Fichier d'extension '.jsp'. Ex : login.jsp, menu.jsp ...
- Une page jsp est un document texte ayant une syntaxe mixte HTML/ Java.
 - ▶ Code HTML = contenu statique, mise en page
 - ▶ Code Java = contenu dynamique
- Une page jsp n'est pas une classe Java
 - ▶ Une page jsp est une facilité de développement
 - ▶ Le serveur d'application génère une classe java à partir de la page JSP
 - ▶ La classe Java générée est compilée, puis exécutée à la demande

Définition (2/2)

➤ Où stocker les pages jsp ?

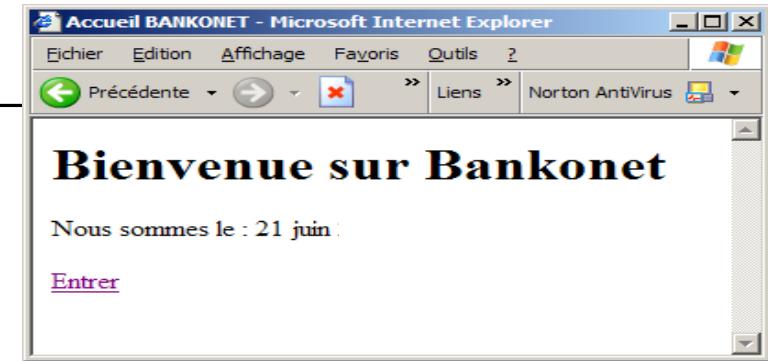
- ▶ Sous le répertoire WebContent (quand on utilise Eclipse)
- ▶ Le répertoire jsp n'est défini dans la norme mais permet de mieux organiser le projet



Exemple

68

```
<html>
<head>
    <title>Accueil BANKONET</title>
</head>
<%@ page language="java" import="java.util.Date,java.text.*" %>
<body>
    <% Date aDate = new Date();
    SimpleDateFormat sdf = new SimpleDateFormat("dd MMMM");%>
    <h1>Bienvenue sur Bankonet</h1>
    <p>Nous sommes le : <%=sdf.format(aDate)%></p>
    <p><a href="<%="request.getContextPath()%>/Connexion">Entrer</a></p>
</body>
</html>
```



accueil.jsp

Cycle de vie d'une page jsp (1/2)

- La page JSP est placée dans l'application JEE.
 - Fichier accueil.jsp

```
<html>
<body>
    <% Date date = new Date(); %>
    La date du jour : <%= date %>
</body>
</html>
```

```
out.println("<html>");
out.println("<body>");
Date date = new Date();

out.println("La date du jour : ");

out.println(date);
out.println("</body>");
out.println("</html>");
```

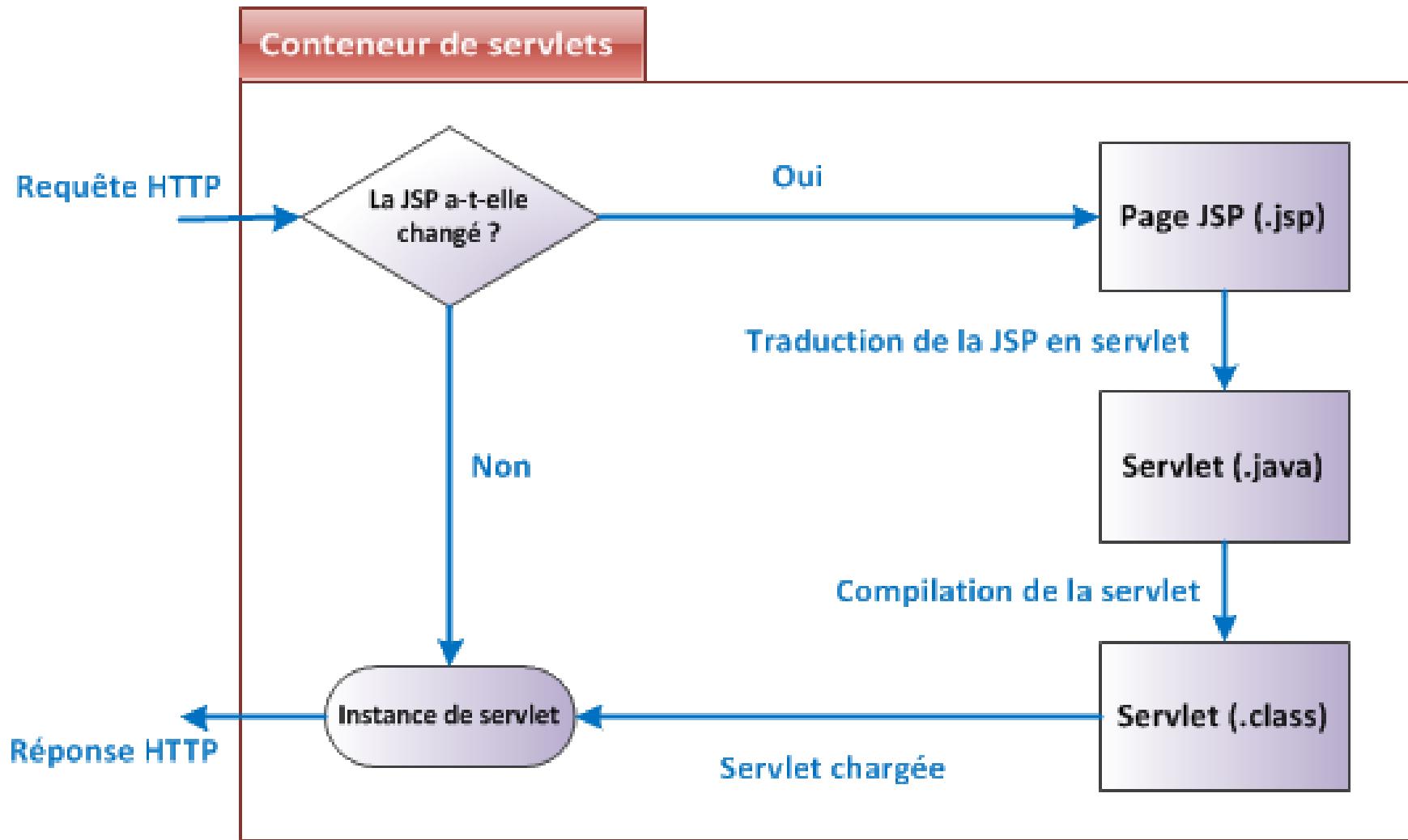
- Au premier appel, le serveur d'application génère une classe java à partir de accueil.jsp .
 - En voici une version simplifiée :

Cycle de vie d'une page jsp (2/2)

- La classe Java générée est une servlet
 - ▶ Elle est compilée à la première utilisation
 - ▶ Génération d'un fichier ".class"
- Le premier appel est plus long
 - ▶ Nécessité de générer la servlet et de la compiler
- Les appels suivants sont bien plus rapides
 - ▶ Appels de servlet classiques
- A chaque modification de la page jsp, le serveur recompile la page automatiquement

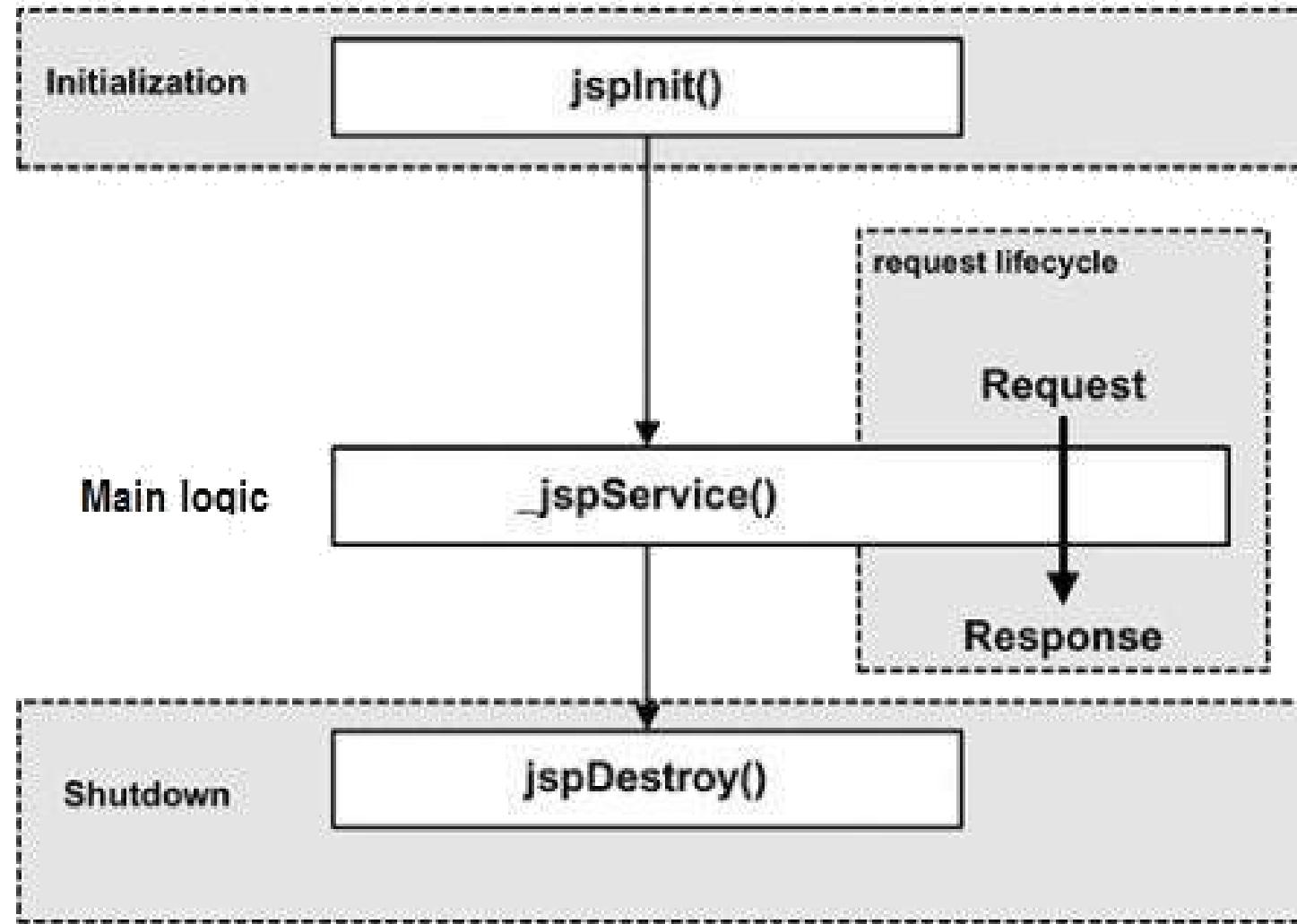
Cycle de vie de la JSP

71



Cycle de vie de la JSP

72



Passer d'une Servlet à une JSP

73

- Par **forward** (on passe tout à la JSP ⇔ ce qui est dans request et response)

```
public void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    req.getRequestDispatcher("/login.jsp").forward(req, resp);
}
```

AccueilServlet

- Par **redirect** (on perd tout ce qui est dans request et response)

```
public void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    resp.sendRedirect(req.getContextPath() + "/login.jsp");
}
```

AccueilServlet

d'une page JSP à une autre page JSP

74

➤ Dans un lien html

```
<a href="<%request.getContextPath() + "/connexion.jsp"%>">  
    Se connecter  
</a>
```

- L'instruction `request.getContextPath()` génère dynamiquement le nom de l'application en cours. Ex : /bankonet
- Après génération par le serveur d'application, le lien ci-dessus permet d'obtenir le résultat html suivant :

```
<a href="/bankonet/connexion.jsp">  
    Se connecter  
</a>
```

La directive 'page' (1/2)

- Définit des attributs spécifiques à la page
- Chaque attribut ne peut être spécifié qu'une seule fois (sauf l'attribut import)
- Exemple :

```
<%@ page language="java" import="java.util.Date,java.text.*" %>
```

La directive 'page' (2/2)

➤ Attributs (valeur par défaut)

- ▶ language (java)
- ▶ extends
- ▶ import
- ▶ session (true)
- ▶ buffer (8 ko)
- ▶ autoFlush (true)
- ▶ contentType (text/html)
- ▶ isThreadSafe (true)
- ▶ info
- ▶ errorPage
- ▶ isErrorPage (false)

Éléments de scripts : Scriptlets (1/2)

77

➤ Fragment de code Java

- ➡ Le code est inséré en l'état dans la classe java générée
 - <% : début de fragment
 - %> : fin de fragment
- ➡ Un fragment peut contenir plusieurs lignes

```
<% Date date = new Date();  
SimpleDateFormat sdf = new SimpleDateFormat("dd MMMM");%>
```



Attention aux ';'. Chaque ligne doit se terminer par un point-virgule ou par une ouverture/fermeture d'accolade.

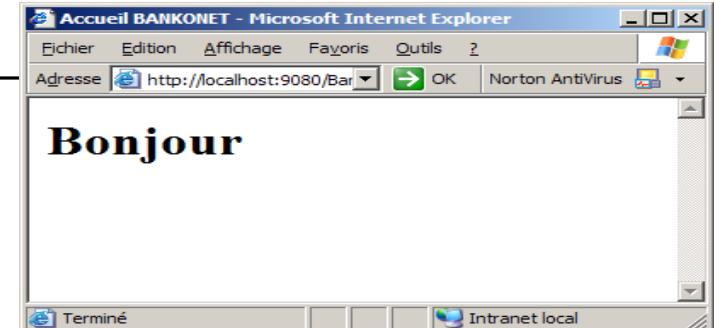
Éléments de scripts : Scriptlets (2/2)

78

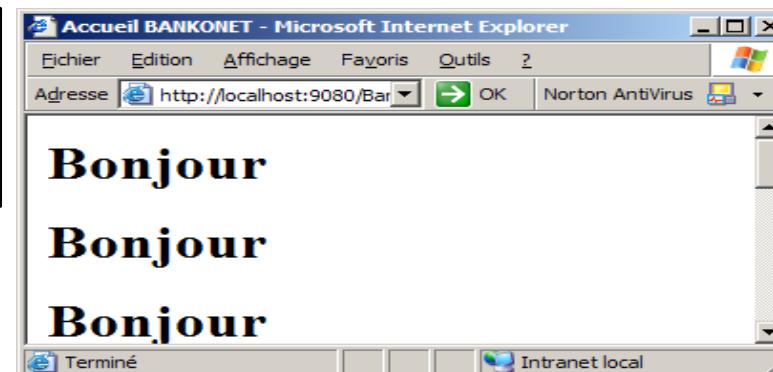
➤ Exemples

```
<%@page language="java" import="java.util.Calendar" %>

<% int i = Calendar.getInstance().get(Calendar.AM_PM);
if (i == Calendar.AM) { %>
    <h1> Bonjour </h1>
<% } else { %>
    <h1> Bonsoir </h1>
<% } %>
```



```
<% for (int i=0; i<10; i++) { %>
<h1> Bonjour </h1>
<% } %>
```



Éléments de scripts : Expressions (1/2)

79

- Fragment de code Java évalué et dont la valeur est insérée dans la réponse
 - ❑ <%= : début de fragment
 - ❑ %> : fin de fragment
- ➡ Un fragment ne contient qu'une seule ligne

```
<%= new Date() %>
```



Attention : pas de ';' à la fin.

Éléments de scripts : Expressions (2/2)

➤ Exemple

```
<%@page language="java" import="java.util.Date" %>
```

Nombre de secondes par heure : <%= 60*60 %>

Date et heure : <%= new Date() %>

➤ Les expressions sont très utilisées pour la récupération de données à afficher

JSP : les objets implicites

- Dans la page jsp, ils sont accessibles directement

| Variable | Type | Représente |
|-------------|---------------------------|------------------------------|
| request | <i>HttpServletRequest</i> | la requête du client |
| application | <i>ServletContext</i> | le contexte de l'application |
| session | <i>HttpSession</i> | la session en cours |

```
<%@page language="java" %>  
<% String nom = (String) session.getAttribute("nom") ; %>
```



Il existe d'autres objets implicites.

Récupération de données : exemple

➤ Mise à disposition d'un objet dans la servlet

```
public class LivreServlet extends HttpServlet {  
    public void service(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        //...  
        request.setAttribute("livre", monLivre); }  
}
```

➤ Utilisation des expressions dans la page JSP

```
<%@page language="java" import="com.bibliotheque.model.Livre" %>  
<% Livre livre = (Livre) request.getAttribute("livre"); %>  
Titre : <%= livre.getTitre() %> </td>  
  
Auteur : <td> <%= livre.getAuteur() %> </td>
```



Le nom des objets implicites est fixe : 'session', 'request'... (et non pas 'ses', 'req'...)

Récupération de données

➤ Mise à disposition d'un objet dans la servlet

```
public class LivreServlet extends HttpServlet {  
    public void service(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        //...  
        request.getSession(true).setAttribute("livre", monLivre); }  
}
```

➤ Utilisation des expressions dans la page JSP

```
<%@page language="java" import="com.bibliothque.model.Livre" %>  
<% Livre livre = (Livre) session.getAttribute("livre"); %>  
...
```



Il faut toujours spécifier le bon scope : un objet placé dans le scope de requête doit être récupéré à partir de ce même scope.

Commentaires

- Les commentaires jsp s'écrivent comme suit :

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

page jsp

- Lors du passage jsp → java, un commentaire jsp génère un commentaire java

```
// ceci est un commentaire
```

classe java générée

- Le commentaire n'apparaît pas dans le code html renvoyé au client
 - ➡ Il reste côté serveur

Gestion du cache pour les JSP

85

- Les pages JSP encapsulent des données dynamiques
- Il est généralement recommandé de désactiver le cache de ces pages pour éviter d'afficher des données obsolètes.
- Dans le cas d'un format HTML :

```
<head>
    <!-- Pour eviter le cache de la page -->
    <META HTTP-EQUIV="Expires" CONTENT="0">
    <META HTTP-EQUIV="Pragma" CONTENT="No-cache">
    <META HTTP-EQUIV="Pragma" CONTENT="No-store">
    <META HTTP-EQUIV="Cache-control", "No-cache">
</head>
```

Page d'accueil

- Dans le web.xml, il est possible de spécifier une (ou plusieurs) pages d'accueil
 - ▶ Le serveur tente de trouver une page de d'accueil en suivant l'ordre de déclaration dans le descripteur de déploiement web.xml
- Dans l'exemple ci-dessus :
 - ▶ si l'url de mon application est `http://localhost:8080/bankonet`, me connecter à cette url invoque la page d'accueil `login.jsp`

```
<web-app id="WebApp">
    <servlet-mapping> ... </servlet-mapping>
    <welcome-file-list>
        <welcome-file>/login.jsp<welcome-file>
    </welcome-file-list>
    <error-page> ... </error-page>
</web-app>
```

web.xml

Page d'erreur

87

- Il est également possible de spécifier une (ou plusieurs) pages d'erreur
 - ▶ Selon les codes erreur http (404, 500, ...)
 - ▶ Ou selon un type d'exception renvoyée (java.lang.Throwable, ...)
- Exemple pour une erreur 404 (fichier non trouvé)

```
<web-app id="WebApp">
    <servlet-mapping> ... </servlet-mapping>
    <welcome-file-list> ... </welcome-file-list>
    <error-page>
        <error-code>404</error-code>
        <location>/erreur.jsp</location>
    </error-page>
</web-app>
```

- Dans l'exemple ci-dessus :
 - ▶ Si l'url `http://localhost:8080/bankonet/aaaabbddzdd` (ressource non-existante) est invoquée, le serveur redirige vers la page /erreur.jsp

Inclusion statique

- Insère le code source d'une ressource (fichier .html, .jsp, ...) dans la page JSP
 - ▶ Juste avant la phase de compilation
 - ▶ la page englobante et la page incluse ne constituent qu'une seule servlet générée
 - ▶ La page incluse voit les imports java qui ont été déclarés dans la page englobante

```
<%@ include file="entete.jsp"%>
```

- Attribut "file"
 - ▶ URL relative de la ressource à insérer

Inclusion dynamique

- Insère le résultat de l'exécution d'une ressource (fichier .html, .jsp, ...) dans la page JSP
 - ▶ Lors de la phase de traitement de la requête
 - ▶ La jsp incluse génère sa propre servlet
 - ▶ La page incluse ne voit pas les imports déclarés dans la page englobante

```
<jsp:include page="entete.jsp" flush="true"/>
```

- Syntaxe
 - ▶ page : URL relative de la ressource à insérer
 - ▶ flush (booléen) : s'il est positionné à true, permet un affichage progressif de la page sur le poste client (le buffer est vidé après l'inclusion)

Inclusion statique ou dynamique ?

- Les inclusions statiques sont plus performantes
 - ➡ Moins de servlets à gérer
- Les inclusions dynamiques se rafraîchissent mieux
- Les inclusions dynamiques permettent un affichage progressif de la page sur le poste client
 - ➡ Attribut flush="true"

Les bibliothèques de balises jsp

- Exécutées lors de la phase de traitement de la requête
 - Peuvent
 - ▶ Modifier le flot de sortie
 - ▶ Utiliser, modifier et/ou créer des objets
 - Interprétées lors de la phase de traitement de la requête
 - ▶ Les valeurs de certains attributs peuvent être des expressions JSP

Les bibliothèques de balises jsp

- <jsp:useBean> : permet de récupérer un objet porteur de données à afficher
 - L'objet doit respecter la norme JavaBean
- Le composant peut être
 - créé ou récupéré dans un contexte existant
 - ❑ page (PageContext) → local à la page
 - ❑ request (ServletRequest) → dans la requête
 - ❑ session (HttpSession) → dans la session utilisateur
 - ❑ application (ServletContext) → dans le contexte de l'application

Les bibliothèques de balises jsp

93

➤ Exemple : <jsp:useBean>

```
<jsp:useBean id="societe" scope="session"
              type="com.sysdeo.bean.Societe" />

<html>
  <body>
    <h1>Présentation</h1>
    <ul>
      <li>Société : <%= societe.getRaisonSocial() %></li>
      <li>Numéro de société : <%= societe.getId() %></li>
    </ul>
  </body>
</html>
```

Les bibliothèques de balises jsp

➤ <jsp:getProperty> : Place la valeur de la propriété d'un Bean dans le flot de sortie

► Convertie sous forme de chaîne de caractère (String)

➤ Syntaxe

```
<jsp:useBean id="societe" scope="session" type="com.sysdeo.bean.Societe" />
```

```
Société : <jsp:getProperty name="societe" property="raisonSocial" />
```

```
Numéro de société : <jsp:getProperty name="societe" property="id" />
```

Les bibliothèques de balises jsp

- Il existe d'autres balises JSP beaucoup moins utilisées
 - ➡ <jsp:forward> → Redirige une requête (fichier HTML, JSP, Servlet...)
 - ➡ <jsp:setProperty> → Affecte une propriété d'un objet JavaBean
 - ➡ <jsp:root> → Peut définir la racine d'un JSP document (page JSP respectant le format XML)
 - ➡ ...

Configuration JSP (1/2)

- Elément regroupant la configuration des JSP
 - ▶ <jsp-config>
- Élément pour regrouper plusieurs JSP
 - ▶ <jsp-property-group>
 - ▶ Permet d'indiquer des propriétés communes à un groupe de jsp regroupées par un pattern spécifique
 - <el-ignored> → Ignorer les Expression Language
 - <scripting-invalid> → Scriptlet interdit
 - <page-encoding> → Encoding des JSP
 - <include-prelude> → En-tête (insérer comme directive @include)
 - <include-coda> → Pied de page (insérer comme directive @include)
 - <is-xml> → La JSP doit-elle être respecter le format XML (depuis JSP 2.1), on parle de "JSP document"

Configuration JSP (2/2)

97

➤ Exemple

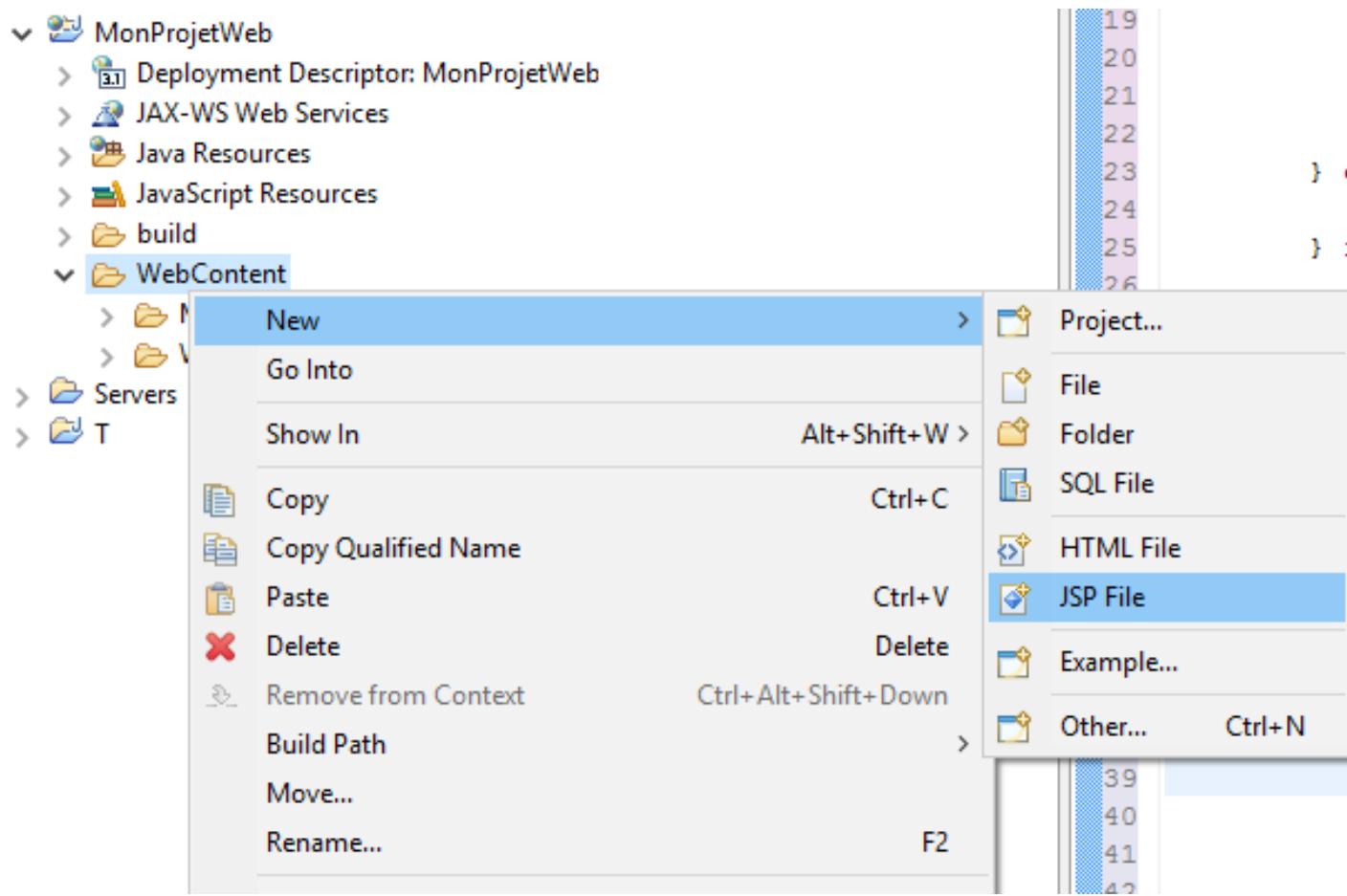
```
<web-app ...>
...
<jsp-config>
  <jsp-property-group>
    <!-- Toutes les JSP -->
    <url-pattern>*.jsp</url-pattern>
    <!-- Scriptlet interdit -->
    <scripting-invalid>true</scripting-invalid>

    <include-prelude>entete.jsp</includle-prelude>
    <include-coda>piedDePage.jsp</include-coda>
  </jsp-property-group>
</jsp-config>
...
</web>
```

JSP dans Eclipse

98

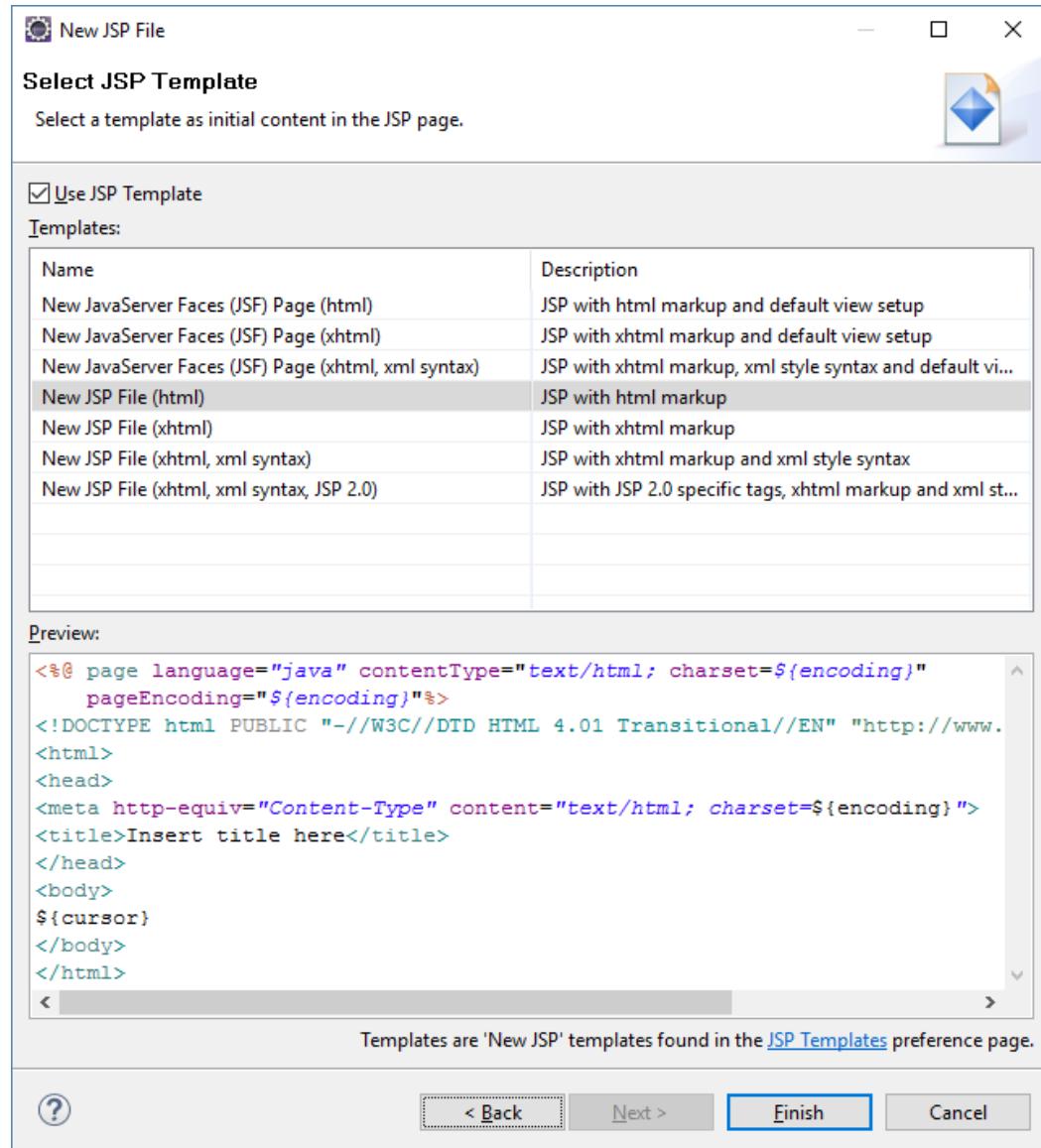
- Toujours en perspective JEE, via le menu contextuel, New -> JSP File en se plaçant dans WebContent



JSP dans Eclipse

99

- Vous pouvez choisir un template pour vos JSP
- Pas encore d'HTML5



JSP dans Eclipse

- Vous pouvez utiliser la compléction de code dans vos JSP
- Elle fonctionne pour l'HTML, le CSS et le Java
- Cependant, l'outil peut assez rapidement ne plus comprendre ce que vous êtes et indiquer des erreurs où il n'y en a pas.
- Des points d'arrêt peuvent être ajoutés
 - ▶ Dans le code des scriptlets
 - ▶ A l'appel des taglib
 - ▶ Pas dans le code HTML
- En cas de rencontre d'un point d'arrêt, Eclipse switch vers la perspective Debug
- Les points d'arrêt sont pris en compte seulement lorsque le serveur est démarré en mode Debug

101

Travaux pratiques

Réaliser les travaux pratiques suivants:

TP 00 bis : Réalisons une JSP bonjour ensemble

JSTL, c'est quoi ? (1/2)

- JSTL est un ensemble de bibliothèques de balises correspondant aux besoins courants de développement de pages JSP
 - ▶ Parcours d'une collection d'objets
 - ▶ Accès à un JavaBean
 - ▶ Formatage de nombres, dates
 - ▶ Accès en base de données
 - ▶ ...

JSTL, c'est quoi ? (2/2)

➤ Une norme pérenne

- ▶ Historiquement, simple projet Apache
- ▶ Depuis, intégrés à JEE depuis la version J2EE 1.4

❑ <http://www.oracle.com/technetwork/java/index-jsp-135995.html>

➤ Mise en place

- ▶ Conteneur web : Ajouter l'API et son implémentation (ajouter un JAR)

Versions de JSTL

- Dernière version de JSTL : 1.2
- Historique et compatibilité
 - ➡ JSTL 1.0 ➔ Servlet 2.3, JSP 1.2 (obsolète)
 - ➡ JSTL 1.1 ➔ Servlet 2.4, JSP 2.0
 - ➡ JSTL 1.2 ➔ Servlet 2.5, JSP 2.1

Les 5 librairies de tags

➤ Documentation des taglibs

► <https://jstl.java.net/>

| Taglib | Préfixe | Description | Exemple |
|-----------|---------|--|---------------------|
| Core | c | Action de base : affichage, gestion des variables de scope, boucle, condition... | If, forEach, Out... |
| I18N | fmt | Formatage, localisation | Bundle, Message... |
| XML | x | Manipulation de fichier XML | Parse, Transform... |
| Database | sql | Accès en base de données | Query, Update... |
| Functions | fn | Depuis JSP 2.0, Expression Language sur des String | Contains, Length... |

Utilisation des JSTL

- Import / déclaration dans chaque page jsp, en JSTL 1.0

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

- ▶ Ou, si vous utilisez JSLT 1.2 et supérieur

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

- Vous devez aussi ajouter le JAR qui va bien dans votre projet
 - ▶ jstl-xxx.jar
- Si la déclaration a été omise, il n'y a pas d'erreur de compilation.
 - ▶ Fonctionnement anormal en phase d'exécution

Utilisation des JSTL

107

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

Bonjour <c:out value="\${client.prenom}" />

c:out écrit dans le flux de la JSP
(comme <%= %>)

Expression dynamique évaluée à
l'exécution : expression language

EL : l'Expression Language

➤ Langage simple et flexible d'accès aux données

- ▶ \${...} pour encadrer les expressions dynamiques

`${expression}`

- ▶ Tout objet présent au moins dans le scope de la JSP peut être utilisé comme variable dans une expression EL
- ▶ Pas de typage
- ▶ Conversions automatiques (nombres ⇄ String)
- ▶ Evaluation des expressions à l'exécution



Pas de vérification syntaxique, attention aux fautes de frappe :
il y aura une erreur à l'exécution

EL: l'Expression Language

➤ Accès à une variable

- ▶ Recherche l'attribut client dans les contextes page, request, session, application dans cet ordre
- ▶ Retourne l'attribut dès qu'il est trouvé dans un des scopes, sinon **retourne null**

```
 ${client}
```

➤ Accès à une méthode

- ▶ Invocation de la **méthode** getter sur l'objet
- ▶ Renvoie **une erreur** si la propriété n'existe pas

```
 ${client.nom}
```

```
client.getNom()
```

EL: l'Expression Language

➤ Accès à une Map

- Retourne la valeur associée à la clé, null s'il n'y en a pas

```
 ${maMap["uneClé"]}
```

➤ Accès à une List ou un tableau

- Retourne la valeur associée à l'index, null s'il n'y en a pas

```
 ${maListe[1]}  
 ${monTableau[i]}
```

➤ On peut enchaîner les appels

```
 ${mesClients[i].adresse.ville}  
 ${client.comptes[0].solde}
```

EL : Objets implicites

111

- Objets implicites EL pour faciliter l'accès aux paramètres
 - ▶ Paramètre de requête (valeur simple) **param**
 - ▶ Paramètre de requête (valeur multiple) **paramValues**
- Accès à un contexte particulier
pageScope, **requestScope**, **sessionScope**, **applicationScope**

```
 ${sessionScope.client}  
 ${param["login"]}
```

- Attention : si vous ne précisez pas le contexte d'un attribut, il sera cherché dans tous les contextes en partant du plus proche vers le plus éloigné
 - ▶ **pageScope** -> **requestScope** -> **sessionScope** -> **applicationScope**

EL : Objets implicites

- Objets implicites EL pour faciliter l'accès aux principaux objets
 - ▶ Page **pageContext**
 - ▶ Requête **pageContext.request**
 - ▶ Réponse **pageContext.response**
 - ▶ Session **pageContext.session**
 - ▶ Application **pageContext.servletContext**

EL : Opérateurs

| Opérateur | Description |
|------------------|--|
| . | Accès à une propriété de Bean |
| [] | Accès à un élément d'une liste |
| () | Sous-expression |
| + | Addition |
| - | Soustraction ou nombre négatif |
| / ou div | Division |
| % ou mod | Modulo (Reste) |
| == ou eq | Egalité |
| != ou ne | Inégalité |
| < ou lt | Test d'infériorité |
| > ou gt | Test de supériorité |
| <= ou le | Inférieur ou égal |
| >= ou ge | Supérieur ou égal |
| && ou and | ET logique |
| ou or | OU logique |
| ! ou not | Inversion de booléen |
| empty | Test de valeur vide (null, chaîne vide, collection vide) |

➤ Services génériques

- ▶ Écriture JSP
- ▶ Boucles
- ▶ Conditions
- ▶ Inclusions
- ▶ URL
- ▶ ...

| Description | Préfixe |
|-------------|---------|
| Core | c |

JSTL – CORE - <c:out />

115

- <c:out /> renvoie une expression après évaluation

- ▶ Exemple

```
<c:out value="${user.name}" default="Inconnu" />  
Dupont
```

- ▶ Évaluation et chaîne de caractère

```
<c:out value="Bonjour, ${user.name}" />  
Bonjour, Dupont
```

- ▶ L'attribut escapeXml

```
<c:out value=<tag>" />  
&lt;tag&gt;  
  
<c:out value=<tag>" escapeXml="false" />  
<tag>
```

- ▶ Une erreur fréquente : Oublier le \${ }

```
<c:out value="name" />  
name
```

JSTL – CORE - <c:if />

116

- L' expression est exécutée sous certaine(s) condition(s)
 - Si plusieurs conditions se succèdent, elles ne sont pas mutuellement exclusives.

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

<c:if test="${article.price == 15}" >
    Prix de 15
</c:if>

<c:if test="${article.price > 40 || article.price < 10 }" >
    Prix trop grand ou trop petit
</c:if>
```

- JSTL ne dispose pas de structure if/else. Mais il dispose de choose/when/otherwise.

JSTL – CORE - <c:choose>

117

- La balise <c:choose> est utilisée pour des conditions mutuellement exclusives
 - Dans l'exemple ci-dessous, si le prix est supérieur à 100, les deux derniers tests ne sont pas effectués.

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

<c:choose>
    <c:when test="${article.price > 100}">
        Gamme luxe
    </c:when>
    <c:when test="${article.price > 0 }">
        Catégorie normale
    </c:when>
    <c:otherwise >
        Cet article n'est pas en vente
    </c:otherwise>
</c:choose>
```

JSTL – CORE - <c:forEach/>

118

```
<%@page language="java" %>
<%@page import="com.shopping.cart.bean.ShoppingCart,
com.shopping.cart.bean.Article, java.util.*" %>
<html>
  <body>
<table border="1">
  <% ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");
List articleList = cart.getArticles();
Iterator articleIte = articleList.iterator();

while (articleIte.hasNext()) {
Article tempArticle = (Article) articleIte.next(); %>
<tr>
  <td> <%= tempArticle.getTitle()%> </td>
  <td> <%= tempArticle.getAuthor()%> </td>
  <td> <%= tempArticle.getPrice()%> </td>
</tr>
<% } %>
</table>
</body>
</html>
```



JSTL – CORE - <c:foreach>

119

➤ Exemple d'implémentation JSTL

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

<html>
<body>
<table>
<c:forEach items="${cart.articles}" var="tempArticle">
  <tr>
    <td> <c:out value="${tempArticle.title}" /> </td>
    <td> <c:out value="${tempArticle.author}" /> </td>
    <td> <c:out value="${tempArticle.price}" /> </td>
  </tr>
</c:forEach>
</table>
</body>
</html>
```



JSTL – CORE - <c:forEach/>

120

➤ Gestion du cas d'une collection vide

```
<c:choose>
    <c:when test="${!empty cart.articles}">
        <table>
            <c:forEach items="${cart.articles}" var="tempArticle">
                <tr>
                    <td> <c:out value="${tempArticle.title}" /> </td>
                    <td> <c:out value="${tempArticle.author}" /> </td>
                    <td> <c:out value="${tempArticle.price}" /> </td>
                </tr>
            </c:forEach>
        </table>
    </c:when>
    <c:otherwise>
        Votre panier est vide
    </c:otherwise>
</c:choose>
```



JSTL - CORE - <c:forEach>

121

- On peut fixer le début et la fin d'une boucle.
- La variable définie dans varStatus a des propriétés permettant de suivre le parcours de la boucle

```
<table>
<c:forEach items="${cart.articles}" var="tempArticle"
begin="${lineNumberStart}" end="${lineNumberEnd}"
varStatus="loopStatus">
    <tr>
        <td> Ligne no <c:out value="${loopStatus.count}" /> </td>
        <td> Article no <c:out value="${loopStatus.index}" /> dans le panier</td>
        <td> <c:out value="${tempArticle.title}" /> </td>
            <td> <c:out value="${tempArticle.author}" /> </td>
            <td> <c:out value="${tempArticle.price}" /> </td>
    </tr>
</c:forEach>
</table>
```

JSTL – CORE - <c:url/>

122

➤ <c:url /> construit une URL (n'affiche **RIEN**)

Exemple depuis une page `http://localhost/one/dir/default.jsp`

```
<c:url value="page.jsp" />  
page.jsp
```

```
<c:url value="/page.jsp" />  
/one/page.jsp
```

```
<c:url value="/page.jsp" context="/two" />  
/two/page.jsp
```

```
<c:url value="/page.jsp">  
    <c:param name="id" value="${id}" />  
    <c:param name="title" value="modifier page" />  
</c:url>  
/one/dir/page.jsp?id=2&title=modifier+page
```

Exemple d'utilisation:

```
<a href="
```

Exemple d'utilisation avec changement de contexte:

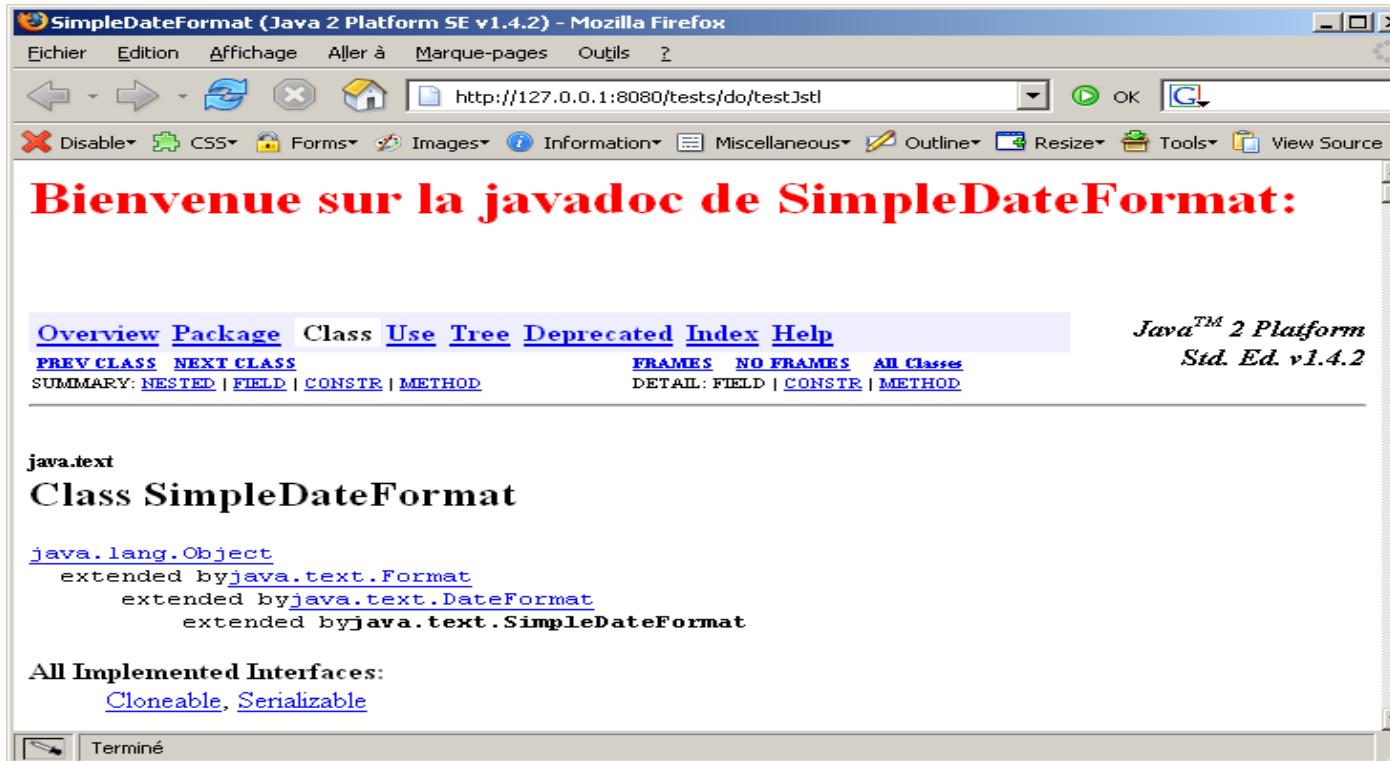
```
<a href="
```

JSTL – CORE - <c:import/>

123

- Importe le contenu d'une ressource accédée via une url

```
<h1>Bienvenue sur la javadoc de SimpleDateFormat:</h1>
<c:import url="http://java.sun.com/.../SimpleDateFormat.html" />
```



JSTL – CORE - <c:redirect />

124

- Redirige vers une autre url

```
<c:choose>
    <c:when test="${empty user}">
        <c:redirect url="/login.jsp" />
    </c:when>
    <c:otherwise >
        une page...
    </c:otherwise>
</c:choose>
```

JSTL : Librairie Format

125

➤ Formatage et internationalisation

- ▶ Messages internationalisés
- ▶ Formatage de dates
- ▶ Formatage de nombres
- ▶ ...

| Description | Préfixe |
|-------------|---------|
| Format | fmt |

➤ Formatage nombres, monnaie

```
<%@ taglib prefix= "fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  
  
<fmt:formatNumber value="12.3" pattern=".00"/>  
  
<fmt:formatNumber value="${product.price}" type="currency" />  
  
<fmt:formatNumber value="${stats.result}" type="percent" />
```

➤ Formatage de dates

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

<%-- Cr ation d'une variable aDay contenant la date du 5/3/65 --%>
<fmt:parseDate var="aDay" value="05/03/1965" pattern="dd/MM/yyyy" />

<%-- Formatage d'une variable aDay --%>
<fmt:formatDate value="${aDay}" type="date" dateStyle="full"/>
<fmt:formatDate value="${aDay}" type="date" pattern="dd MMM yyyy ' ' HH:mm"/>
```

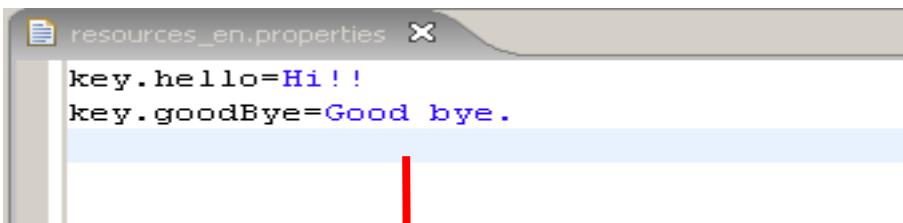


Voir `java.text.SimpleDateFormat`

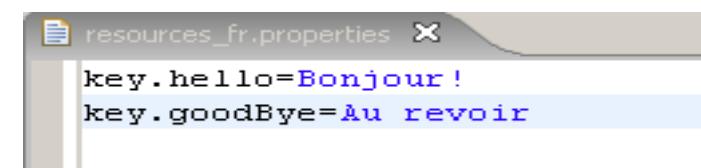
➤ Internationalisation

```
<%@ taglib prefix= "fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  
  
<fmt:setLocale scope="session" value="${myLocal}" />  
  
<fmt:bundle basename="com.site.resources.resources">  
  
    <fmt:message key="key.hello" />  
  
</fmt:bundle>
```

myLocal=en



myLocal=fr



Travaux pratiques

129

Réaliser les travaux pratiques suivants:

TP 02 : Réalisons les JSP client et compte

Bibliothèques de balises : Définition

130

- Les bibliothèques de balises permettent de définir de nouvelles actions associées à une balise personnalisée
 - ▶ portable (indépendant du conteneur de JSP)
 - ▶ mécanisme simple
 - ▶ une large gamme d'actions peut être décrite par ce mécanisme
- Pour cela on doit :
 - ▶ Créer un descripteur de bibliothèque de balise (Tag Library Descriptor)
 - ▶ Créer des classes gestionnaires (handler) associées aux balises
 - ▶ Déclarer la bibliothèque dans le descripteur de déploiement (web.xml)
 - ▶ Déclarer les bibliothèques utilisées dans la JSP

Bibliothèques de balises

- Tag Library Descriptor
 - ▶ document XML d'extension *.tld
 - ▶ informations générales de la bibliothèque
 - ▶ description de la syntaxe de chaque balise
 - Nom de la balise
 - Nom des attributs (obligatoires ou optionnels)
 - Contenu de la balise (body)
 - ▶ association d'une balise à sa classe gestionnaire (TagHandler)
- Permet aux IDE de prendre connaissance des nouvelles balises disponibles

Bibliothèques de balises

132

➤ Descripteur de bibliothèque (2/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
 1.1//EN" "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>bankonet</shortname>
  <uri>http://www.bankonet.fr/tags</uri>
  <tag>
    <name>link</name>
    <tagclass>com.bankonet.taglib.LinkTag</tagclass>
    <bodycontent>empty</bodycontent>
    <attribute>
      <name>href</name>
      <required>true</required>
    </attribute>
    <attribute>
      <name>label</name>
      <required>true</required>
    </attribute>
  </tag>
</taglib>
```

bankonetTags.tld

Bibliothèques de balises

- 3 classes possibles à hériter pour écrire ses propres tags :
 - ➡ javax.servlet.jsp.tagext.TagSupport
 - Tag sans corps
 - ➡ javax.servlet.jsp.tagext.BodyTagSupport
 - Tag avec corps
 - ➡ javax.servlet.jsp.tagext.SimpleTagSupport
 - Tag avec ou sans corps et gestion simplifiée
- Pour chaque attribut de la balise, on a un membre dans le handler avec le même nom et ses accesseurs

Bibliothèques de balises

➤ Les méthodes à surcharger

► TagSupport et BodyTagSupport

- ❑ doStartTag() appelé à l'analyse de la balise de début
- ❑ doEndTag() appelé à l'analyse de la balise de fin
- ❑ doInitBody() appelé avant le traitement éventuel du contenu de la balise
 - Seulement pour BodyTagSupport
- ❑ doAfterBody() appelé après le traitement éventuel du contenu de la balise

► Les retours possibles

- ❑ EVAL_BODY_INCLUDE ➔ Evalue le corps du tag
- ❑ EVAL_PAGE ➔ Evalue la suite de la page
- ❑ SKIP_BODY ➔ Ignore le corps du tag
- ❑ SKIP_PAGE ➔ Ignore la suite de la page

Bibliothèques de balises

➤ Les méthodes à surcharger

► SimpleTagSupport

□ Simplement doTag() à surcharger

➤ Sans traitement du corps de la balise

```
public void doTag() throws JspException, IOException {
    JspWriter out = getJspContext().getOut();
    out.print("Hello World!");
}
```

➤ Avec traitement du corps de la balise

```
public void doTag() throws JspException, IOException {
    JspWriter out = getJspContext().getOut();

    StringWriter flux = new StringWriter(); // flux.append()...
    JspFragment body = getJspBody();
    body.invoke(flux); // écrit dans flux

    out.print(flux.toString());
}
```

Bibliothèques de balises

136

➤ Exemple : génération d'un lien html avec son contexte

```
public class LinkTag extends TagSupport {  
    private String href;  
    private String label;  
  
    public void setHref(String pHref) { href = pHref; }  
  
    public void setLabel(String pLabel) { label = pLabel; }  
  
    public int doStartTag() throws JspException {  
        try {  
            HttpServletRequest request = (HttpServletRequest)pageContext.getRequest();  
            JspWriter out = pageContext.getOut();  
            out.println("<a href=\"" + request.getContextPath() + href + "\">"  
                      + label + "</a>");  
            return EVAL_BODY_INCLUDE;  
        }  
        catch (IOException e) { e.printStackTrace(); }  
        return SKIP_BODY;  
    }  
}
```

Bibliothèques de balises

137

- L'interpréteur de pages jsp invoque la création d'une instance LinkTag.
- Les méthodes setHref(...) et setLabel(...) sont invoquées (setters correspondant aux attributs de la balise).
- La méthode doStartTag(...) est invoquée. Son contenu sera ajouté à la réponse http.
- La méthode doEndTag(...) est invoquée. Son contenu sera ajouté à la réponse http.

Bibliothèques de balises

138

➤ Déclaration dans la page jsp

```
<%@ taglib prefix="bankonet" uri="http://www.bankonet.fr/tags"%>
```

- ▶ uri : URI déclarée dans le fichier tld
- ▶ prefix : préfixe à utiliser pour marquer une balise personnalisée

➤ Appel

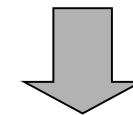
```
<bankonet:link href="/Virement" label="Effectuer un virement"/>
```

Bibliothèques de balises : Cheminement

139

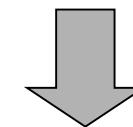
```
<bankonet:link href="/virement" label="Effectuer un virement"/>
```

Dans la page jsp



```
<a href="/BankonetWeb/Virement">Effectuer un virement</a>
```

Code html généré



- Effectuer un virement

Dans le navigateur

Bibliothèques de balises

140

- Des taglibs utiles existent dans le monde JEE, avant d'écrire vos propres taglibs, vérifier qu'elles n'existent pas déjà
- DisplayTag ➔ Affiche une liste d'objets sous forme d'un tableau <http://www.displaytag.org/1.2>
 - ➡ Pagination
 - ➡ Tri
 - ➡ Internationalisation
 - ➡ Export (pdf, Excel, XML, CSV)
- CKEditor ➔ Edition de texte riche <http://ckeditor.com/>

| Amount | Project | Task | City |
|--------|------------|------------------------------------|----------|
| 587.0 | Arts | et gubergren ut et | Olympia |
| 72.0 | Gladiators | duo sit erat justo | Neapolis |
| 277.0 | Gladiators | justo tempor consetetur consetetur | Roma |
| 598.0 | Gladiators | magna erat tempor justo | Roma |
| 953.0 | Gladiators | voluntua nonum et sedinoccina | Olympia |

```
<% request.setAttribute( "test", new TestList(10, false) ); %>
display:table name="test" />
```

141

MVC

Définition

Mise en pratique

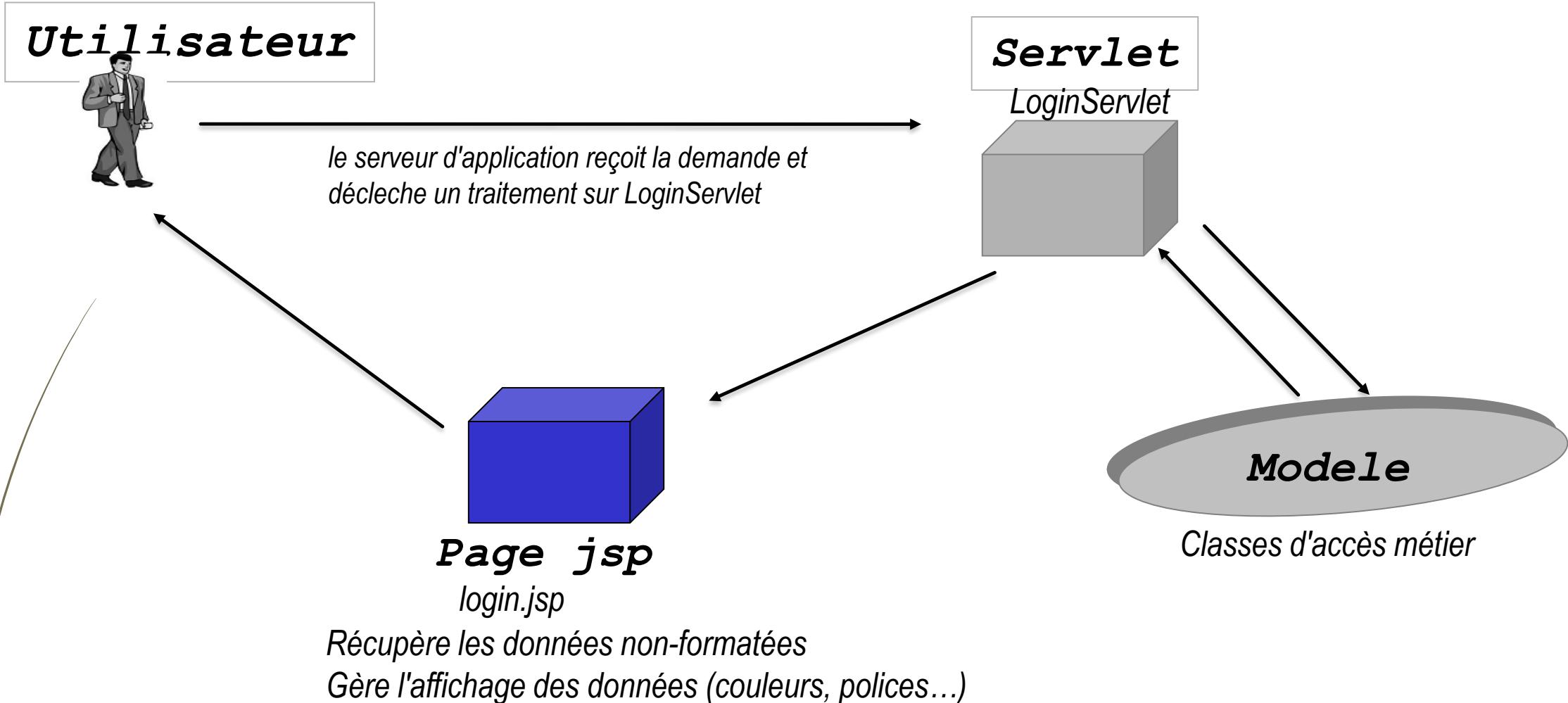
Définition

142

- Architecture découpée en couches :
 - Modèle : des objets Java qui servent pour l'échange d'information entre la Vue et le Contrôleur. C'est du code Java.
 - Vue : Ce que voit l'utilisateur. C'est de l'HTML, du CSS, du Javascript, et potentiellement des JSPs
 - Contrôleur : Objet qui permet de gérer les actions/demandes de l'utilisateur. Peut être une Servlet ou un objet de traitement templatisé.

Rappel : structure MVC avec JEE

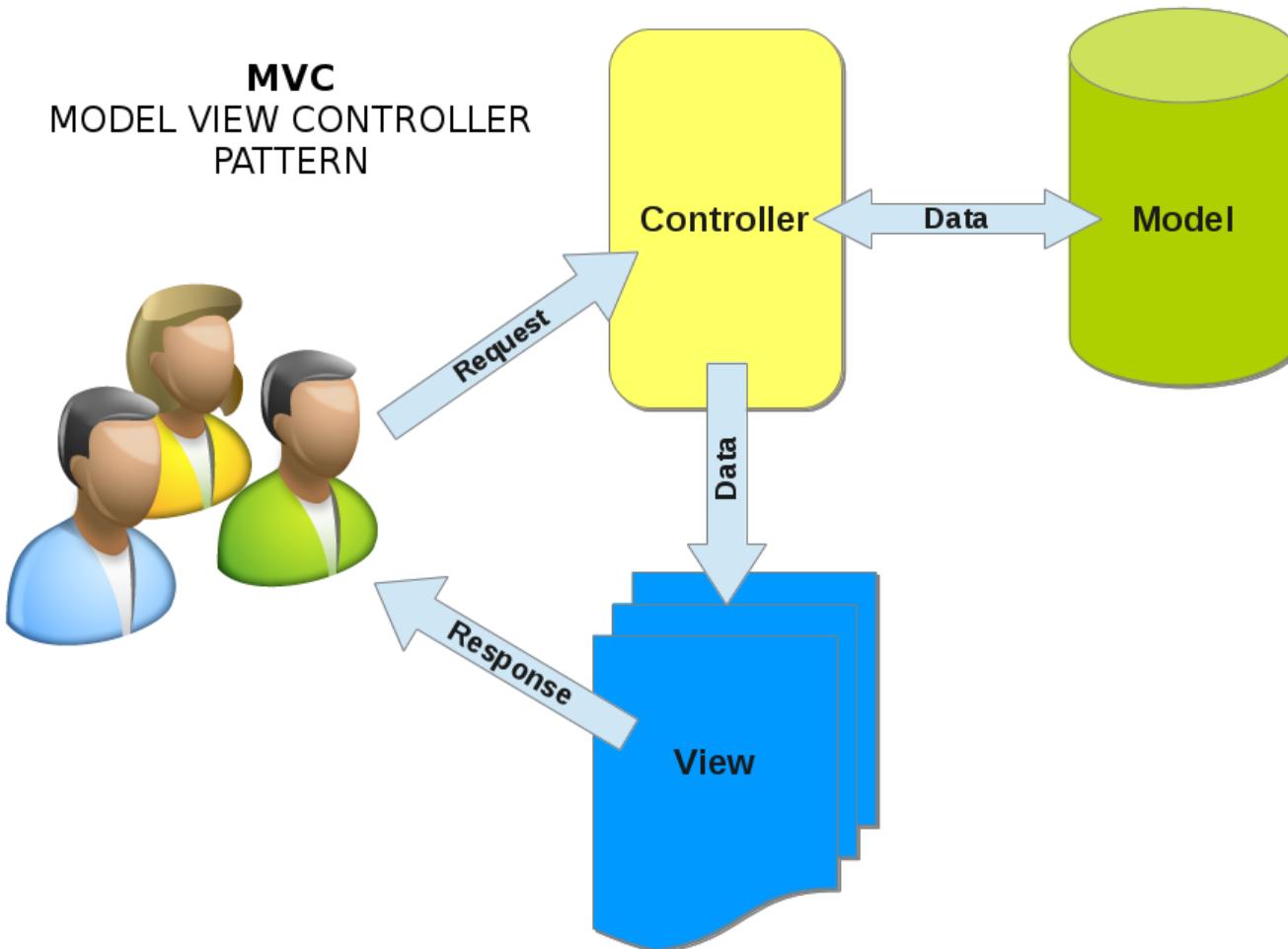
143



Définition

144

➤ MVC



En JSP - Servlet

145

- Vues : Vos pages JSPs
- Contrôleurs : Vos servlets
- Modèles : Vos objets qui viennent de la base + d'autres objets spécifiques aux vues
- Vous allez avoir autant de Contrôleurs que d'action dans votre site web
 - ➡ Vous pouvez cependant faire usage de l'héritage ou d'astuces webs
- Action =
 - ➡ Voir une page
 - ➡ Valider un formulaire
 - ➡ Cliquer sur un bouton

En JSP - Servlet

- Chaque élément a un rôle bien défini
- La Vue affiche
- Le Modèle transporte l'information
- Le Contrôleur récupère ou fabrique du modèle en fonction des demandes des Vues
- C'est de l'architecture : on range de manière standard pour simplifier la maintenance

En JSP - Servlet

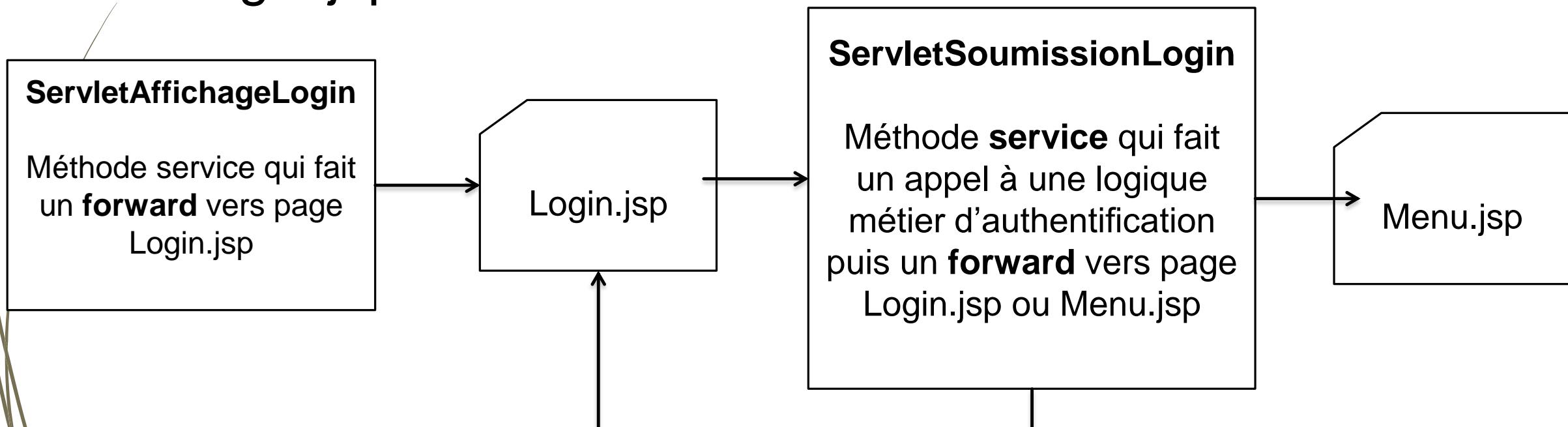
147

- Selon les framework vous pouvez avoir des règles d'architectures complémentaires.
- Typiquement, le cheminement des appels devraient toujours suivre le schema
 - ➡ Servlet -> JSP -> Servlet -> JSP -> Servlet -> JSP -> ...
- En aucun cas on doit avoir :
 - ➡ JSP -> JSP ou Servlet -> Servlet

En JSP - Servlet

148

- Action 1 : Afficher la page Login.jsp
- Action 2 : soumettre le formulaire de la page Login.jsp, aller vers Menu.jsp si tout est ok sinon aller vers Login.jsp



En JSP - Servlet

149

- Dans notre exemple, pour accéder au site, on tapera l'url de la servlet et pas celui de la JSP
- On peut aussi coder un seul contrôleur et utiliser les méthodes doGet et doPost afin de différencier les deux traitements.
 - Ou faire usage de la présence de paramètre dans la request
- Dans cette exemple, le modèle c'est
 - Un entier ou un objet utilisateur que l'on place dans le scope session afin de s'en servir plus tard.
 - Un String qui nous sert de message d'erreur ou de bienvenue
 - Un objet LoginBean qui représente le formulaire de la page Login.jsp avec un attribut login et password

En JSP - Servlet

➤ Avantages

- ▶ Si vous respectez bien le modèle MVC vous ne devriez jamais avoir d'URLs de JSPs visibles dans le navigateur
- ▶ Vous restez maître de vos enchainements d'écran via votre code Java

➤ Inconvénients

- ▶ Peut entraîner des lourdeurs ou du copier/coller selon les frameworks

Travaux pratiques

Réaliser les travaux pratiques suivants:

TP 03 : Couplons nos servlets et nos JSPs

TP 04 : Réalisons un vrai site web en MVC

152

Filtres

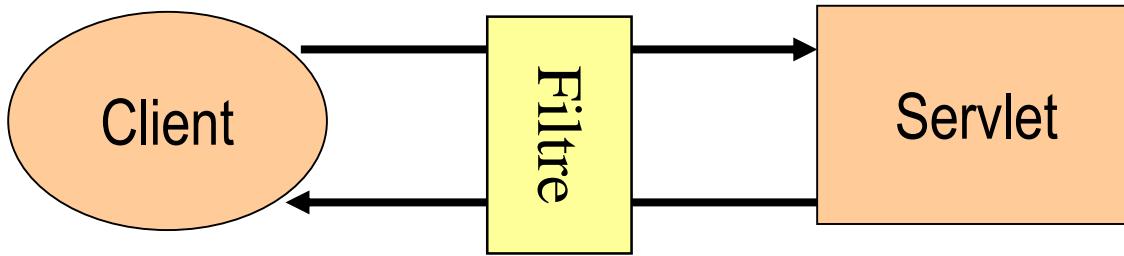
Définition

Déclaration

Définition

153

- Un filtre est un intermédiaire de passage entre le client et la servlet



- Il peut notamment
 - ➡ Intercepter la requête avant traitement
 - ➡ Intercepter la réponse après traitement
- Un filtre est une classe Java
 - ➡ Il doit implémenter l'interface javax.servlet.Filter

Référencement dans le web.xml

154

- Dans le web.xml, un filtre est positionné sur un type d'URI

- ▶ Filtre portant sur un seul URL

```
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/Accueil</url-pattern>
</filter-mapping>
```

- ▶ Filtre portant sur l'ensemble des URLs de l'application web

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

- ▶ Filtre portant sur certains URLs

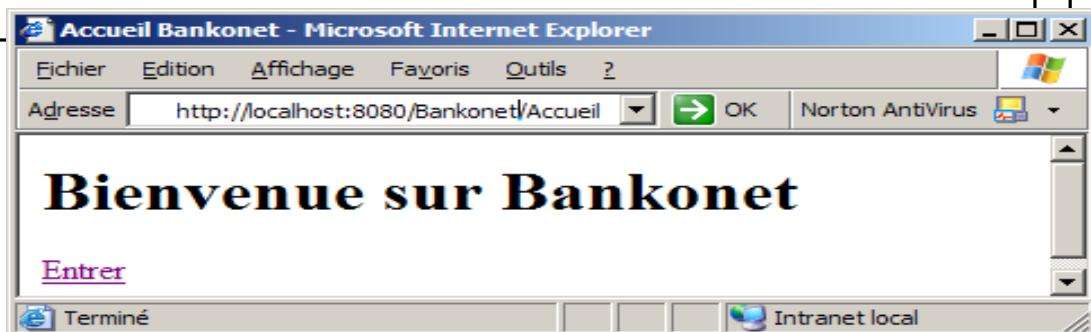
```
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/filtered/*</url-pattern>
</filter-mapping>
```

Filtres : Exemple

155

1 - LogFilter.java

```
public class LogFilter implements Filter {  
    public void doFilter(ServletRequest req,  
                         ServletResponse resp, FilterChain chain)  
        throws ServletException, IOException {  
        System.out.println("passage dans le filtre");  
        chain.doFilter(req, resp);  
    } ... }
```



2 - web.xml

```
<filter>  
    <filter-name>LogFilter</filter-name>  
    <filter-class>com.bankonet.filter.LogFilter  
    </filter-class>  
</filter>  
<filter-mapping>  
    <filter-name>LogFilter</filter-name>  
    <url-pattern>/*</url-pattern>  
</filter-mapping>
```

3 - Appel à la Servlet "AccueilServlet"

4 - Traces (console)

```
Console [BankonetFilteredServer (WebSphere v5.1)]  
[12/07/04 16:45:04:945 CEST] 537b7cb1 WebGroup I SRVE0180I: [Bankonet_Web] [/Be  
[12/07/04 16:45:04:945 CEST] 537b7cb1 SystemOut O passage dans le filtre  
[12/07/04 16:45:04:945 CEST] 537b7cb1 SystemOut O passage dans AccueilServlet
```

Chain.doFilter(...)

- Le filtre englobe l'appel de ressource (servlet, fichier statique...)
- La ressource est appelée lors de l'invocation de la méthode Chain.doFilter(...)
- Le code placé avant cette invocation sera exécuté avant l'appel
- Le code placé après cette invocation sera exécuté après l'appel

```
public class LogFilter implements Filter {  
    public void doFilter(ServletRequest req, ServletResponse resp,  
        FilterChain chain) throws ServletException, IOException {  
        System.out.println("Avant appel de servlet");  
        chain.doFilter(req, resp);  
        System.out.println("Après appel de servlet");  
    } ... }
```



Si le chain.doFilter(...) est omis, l'élément suivant n'est pas traitée et la réponse n'est pas transmise au client

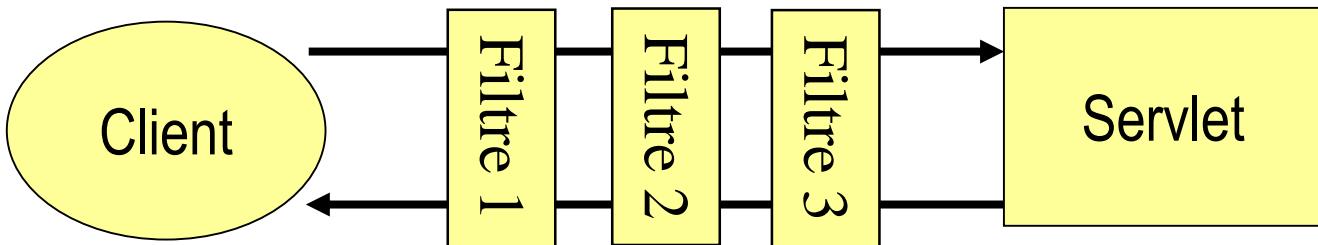
Exemples d'utilisation

- Suppression des blancs html
 - ➡ Pour limiter le trafic du serveur web vers le client
- Authentification/sécurisation
- Journalisation (logs) et d'audit
- Conversion d'images
- Traduction de format (XML → HTML, XML → WML)
- Cache
- Ré-écriture d'url (création de répertoires virtuels)
- ...

Filtres en série

158

- Il est possible d'associer plusieurs filtres à une URI donnée



- Ils sont exécutés dans l'ordre de déclaration dans le web.xml

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

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

Modification de la réponse

159

- Dans l'exemple de filtre "LogFilter" vu précédemment, la réponse http n'est pas modifiée par le filtre
- Certains filtres de servlet modifient la réponse http
 - ➡ Ex : Suppression des blancs
 - ➡ Ex : Zipper un fichier proposer en téléchargement

Annotation

160

- Vous pouvez aussi utiliser l'annotation de classe @WebFilter au lieu du mapping web.xml
 - C'est l'un **ou** l'autre (pas les deux)

```
@WebFilter(urlPatterns = { "/MonF" })
public class MonSuperFiltre implements Filter {
```

- Notez que comme pour une servlet, un filtre peut avoir plusieurs URLs de mapping

Quizz

161

- Que se passe-t-il si j'oublie le "Chain.doFilter(...)" dans la méthode doFilter d'un filtre ?
- J'utilise un filtre qui ne me sert qu'en développement. Je ne souhaite pas qu'il soit activé en environnement de production. Que dois-je modifier lors du passage développement → production ?
- Un projet comporte un filtre d'authentification AuthFilter et 10 servlets. Comment faire pour que le filtre ne soit appliqué que pour 2 des 10 servlets ?

Filtres et RequestDispatcher

162

➤ Paramétrage dans web.xml

- ▶ Depuis servlet 2.4
- ▶ Par défaut, filtre activé seulement par accès direct
- ▶ Valeurs possibles
 - ❑ REQUEST → Accès direct
 - ❑ FORWARD → RequestDispatcher.forward()

```
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Travaux pratiques

163

Réaliser les travaux pratiques suivants:

TP 05 – p1 : Réaliser un filtre pour sécuriser le site web

164

Les listeners

Définition

Déclaration

Programmes d'écoute - ServletContext

165

- Un système d'abonnement permet à certains objets d'être notifiés lorsque des évènements particuliers ont lieu (8 évènements possibles)
- ServletContextListener : Cycle de vie de l'application
 - ▶ contextInitialized (création)
 - ▶ contextDestroyed (destruction)
- ServletContextAttributeListener : Modification du contenu des attributs application
 - ▶ attributeAdded (ajout)
 - ▶ attributeRemoved (suppression)
 - ▶ attributeReplaced (remplacement)

Programmes d'écoute - ServletRequest

- **ServletRequestListener** : Cycle de vie des requêtes
 - ▶ requestInitialized (création)
 - ▶ requestDestroyed (destruction)
- **ServletRequestAttributeListener** : Modification du contenu des attributs requête
 - ▶ attributeAdded (ajout)
 - ▶ attributeRemoved (suppression)
 - ▶ attributeReplaced (remplacement)

Programmes d'écoute - HttpSession

167

- HttpSessionListener : Cycle de vie des sessions
 - ▶ sessionCreated (création)
 - ▶ sessionDestroyed (destruction)

- HttpSessionAttributeListener : Modification du contenu des attributs session
 - ▶ attributeAdded (ajout)
 - ▶ attributeRemoved (suppression)
 - ▶ attributeReplaced (remplacement)

Programmes d'écoute - HttpSession

- HttpSessionBindingListener : Les objets implémentant cette interface seront notifiés lorsqu'ils seront stockés ou supprimés d'une session
 - ▶ valueBound (ajout en session)
 - ▶ valueUnbound (suppression de la session)
- HttpSessionActivationListener : Les objets implémentant cette interface seront notifiés lorsque la session sera activé ou désactivé (utile en environnement avec répartition de charge, fail-over)
 - ▶ sessionDidActivate (activé)
 - ▶ sessionWillPassivate (désactivé)

Programmes d'écoute

169

➤ Déclaration dans le web.xml

```
<!-- déclarations précédentes : filter*, filter-mapping* -->
<listener>
    <listener-class>com.bk.listener.SysdeoSessionListener</listener-class>
</listener>
<!-- déclarations suivantes : servlet*, servlet-mapping* -->
```

- Pour chaque élément `<listener>`, une instance de la classe est créée au démarrage du serveur.
- Si plusieurs listeners du même type (`HttpSessionListener` par ex.) sont déclarés, ils sont notifiés dans l'ordre de déclaration dans le descripteur.

Programmes d'écoute

170

- Le container prend en compte la déclaration du programme d'écoute dans le web.xml
- Il instancie SysdeoSessionListener au démarrage du serveur
- A chaque création/destruction de session, les méthodes correspondantes sont invoquées

```
public class SysdeoSessionListener
    implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent event) {
        System.out.println("Session created");
        System.out.println(event.getSource());
    }
    public void sessionDestroyed(HttpSessionEvent event) {
        System.out.println("session destroyed");
    }
}
```

```
<listener>
    <listener-class>
com.bankonet.listener.SysdeoSessionListener
    </listener-class>
</listener>
```

Dans web.xml



HttpSessionEvent permet de récupérer des informations sur la session en cours et sur l'objet déclencheur de l'événement

Annotation

171

- Vous pouvez aussi utiliser l'annotation de classe @WebListener au lieu de sa déclaration dans le fichier web.xml
 - C'est l'un ou l'autre (pas les deux)

```
@WebListener  
public class MonSuperListener implements HttpSessionAttributeListener {
```

- Notez qu'un listener peut implémenter plusieurs interfaces.

Travaux pratiques

172

Réaliser les travaux pratiques suivants:

TP 05 – p2 : Réaliser un listener qui nous permettra de savoir qui est connecté au site

Ajax

173

Définition

- Asynchronous Javascript and XML
- Requêter un serveur via un l'objet JavaScript XMLHttpRequest sans recharger toute la page
 - ➡ Pré-valider les données d'un formulaire
 - ➡ Charger les informations d'un type d'objet dans un template HTML existant
 - ➡ ...
- Traiter la réponse du serveur via une fonction JavaScript "callback" afin de mettre à jour la page

Ajax

175

- La requête Ajax va appeler une Servlet
- La servlet envoie une réponse formatée pour pouvoir être interprétée par du code JavaScript côté navigateur
 - ▶ XML (désaffecté au profit du format JSON)
 - ▶ JSON

```
{  
    "id": "1",  
    "nom": "Compte PEL",  
    "clé": "valeur"  
}
```
 - ▶ HTML directement à intégrer dans la page
- De nombreux frameworks Ajax permettent de créer des requêtes Ajax et de manipuler la réponse simplement
 - ▶ jQuery → <http://jquery.com>
 - ▶ Ext JS → <http://www.sencha.com/products/extjs>
 - ▶ Prototype → <http://www.prototypejs.org>
 - ▶ Angular JS → <https://angularjs.org/>

Framework connexes

Struts

JSF

Hibernate

EJB

Présentation des frameworks

177

- Traduction
 - ▶ Littéralement "cadre de travail", ou "squelette d'application" ou "plate-forme"
- Définition
 - ▶ Les frameworks sont des structures logicielles qui définissent des cadres dans lesquels viennent s'insérer les objets et concepts spécifiques à une application.
- En pratique, un framework est un ensemble de classes qui fournissent un ensemble de services (techniques ou métiers) aux applications qui s'appuient dessus.

Deux types de framework

➤ Frameworks techniques

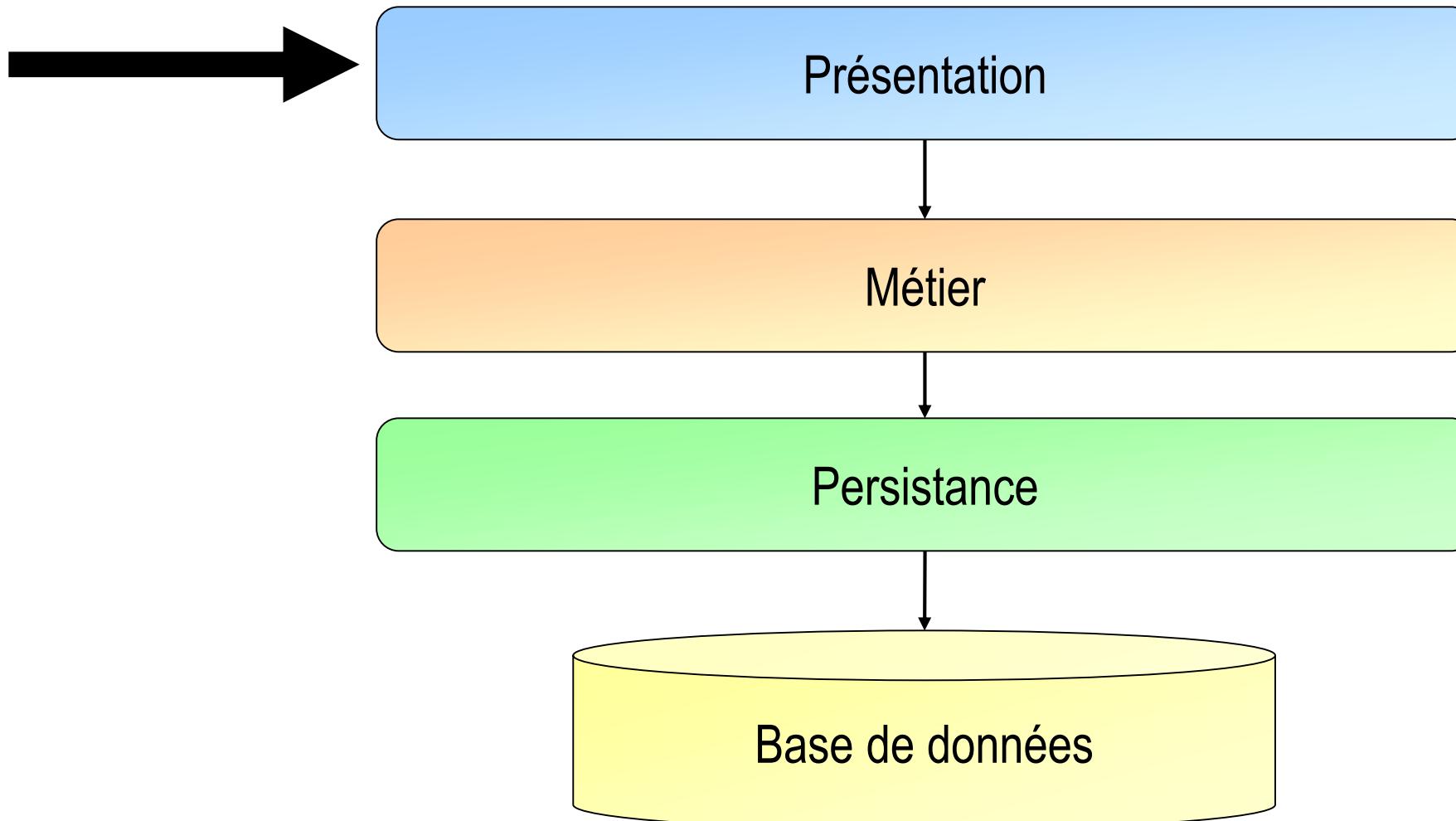
- ▶ Apporte les concepts et mécanismes qui permettent, dans le cadre d'une architecture logicielle retenue, de s'abstraire d'un certain nombre de problématiques conceptuelles et techniques récurrentes.
- ▶ Ex : journalisation, validation, internationalisation, ...

➤ Frameworks métiers

- ▶ Fournit des services à forte plus value fonctionnelle
- ▶ Ex : framework de gestion de compte (compte courant, compte d'épargne, PEL, ...), framework de gestion d'abonnement (de téléphone, de télévision, de cinéma, ...), ...

Framework de présentation WEB

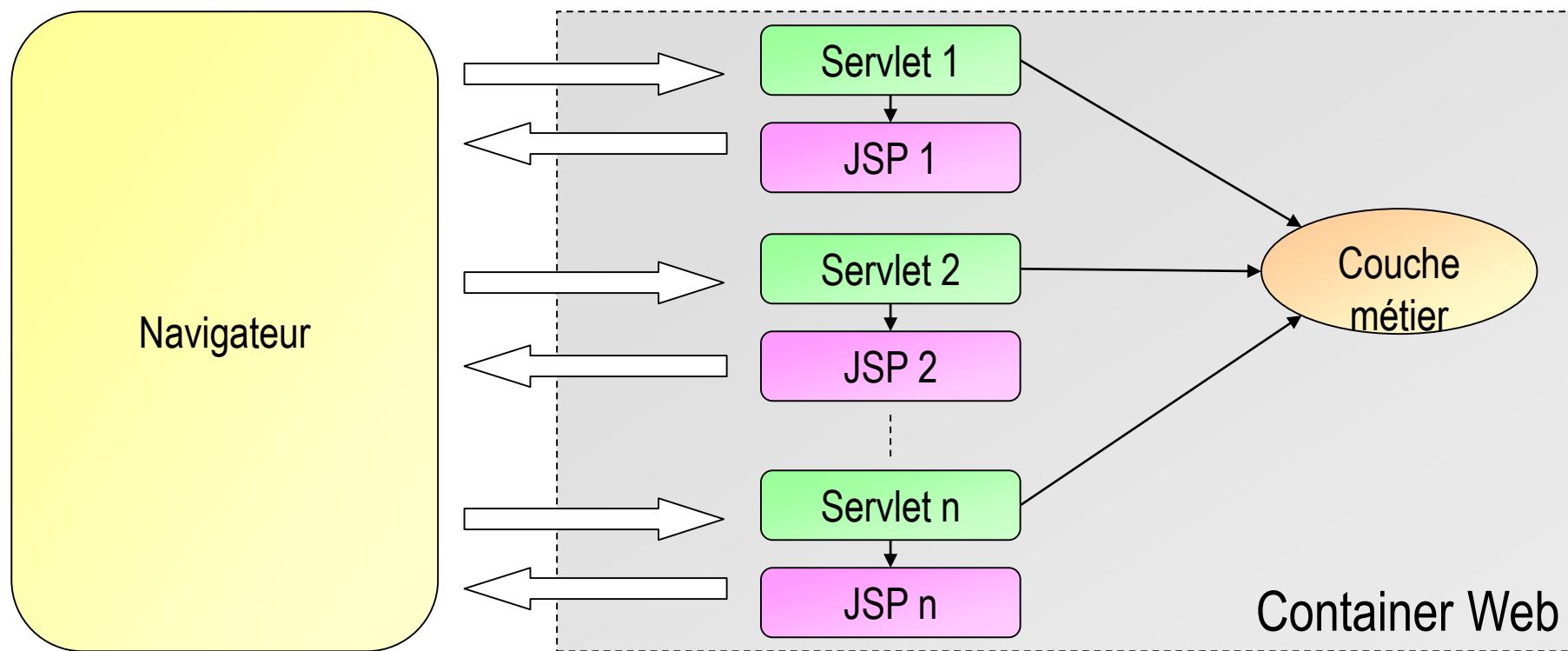
179



Limites du pattern MVC

180

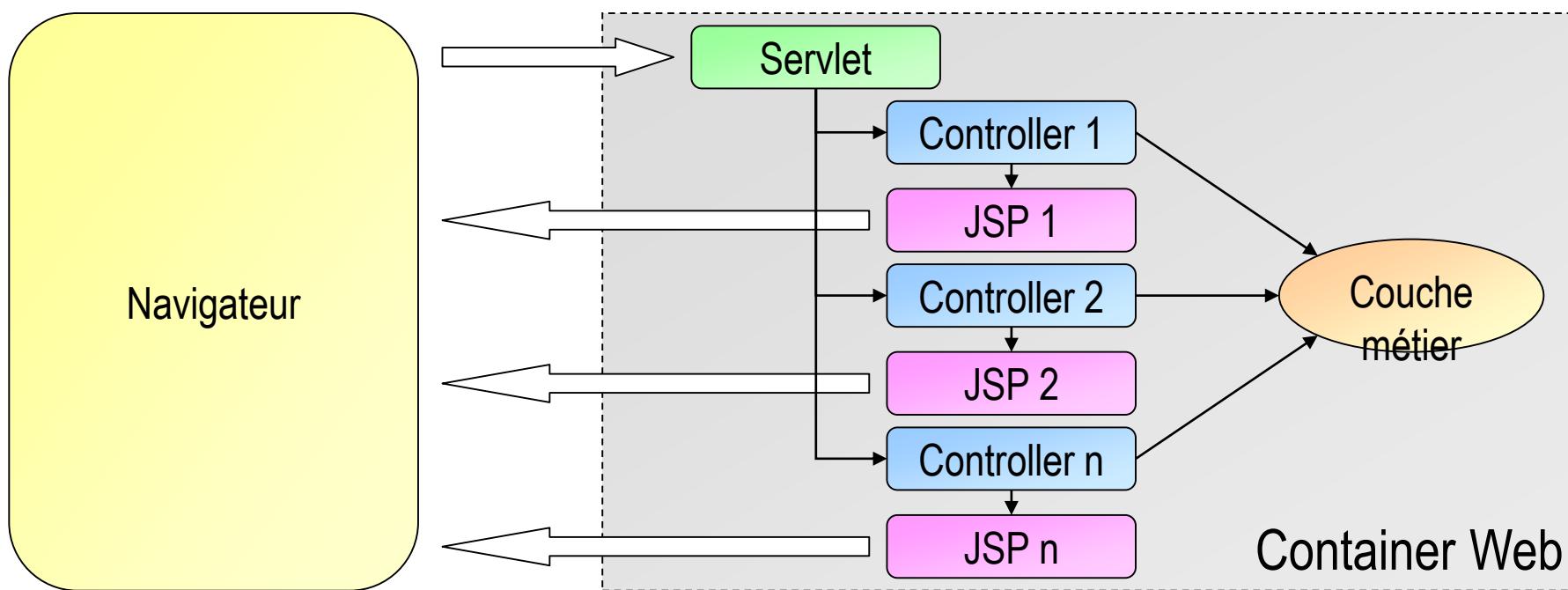
- Multitude de Servlets (une pour chaque vue)
- La mise en place de traitements communs (ex : validation) n'est pas aisée avec une telle architecture



Les apports de MVC2

181

- Le navigateur agit sur une unique Servlet (Macro-contrôleur)
- Le Macro-contrôleur réalise les traitements communs et délègue la partie variable au Micro-contrôleur (classe Java)
 - ➡ Intégration possible de services factorisé : gestion des erreurs, authentification, autorisation, validation, ...



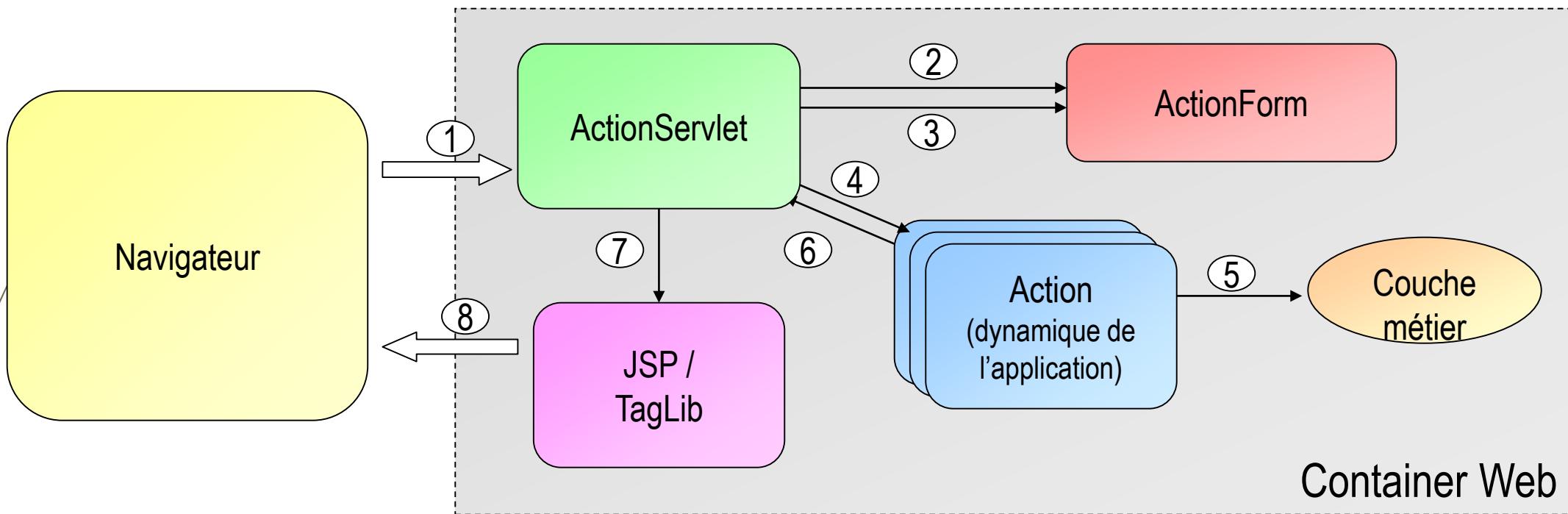
Présentation de Struts

182

- Framework web Java implémentant le pattern MVC2
 - ▶ Basé sur les API Servlet et JSP
 - ▶ Open source
 - ▶ Jakarta / Apache Software Foundation (<http://struts.apache.org>)
- Fournit les fonctionnalités suivantes
 - ▶ Templating (tiles)
 - ▶ Gestion des exceptions améliorées
 - ▶ Prise en charge de la soumission de formulaire
 - ▶ Validation des formulaires
 - ▶ Prise en charge de la navigation entre pages
 - ▶ I18N (internationalisation)
 - ▶ ...

Fonctionnement de Struts 1

183



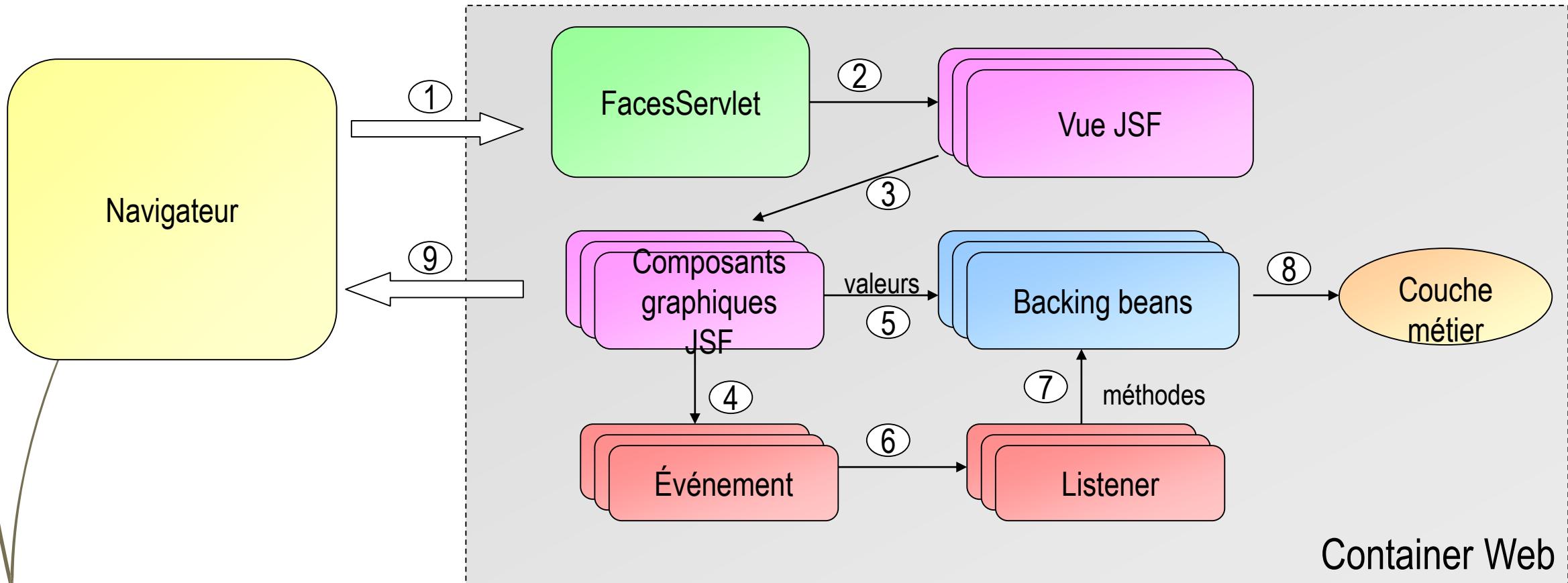
Présentation de JavaServer Faces

184

- Framework web Java améliorant le pattern MVC2
 - ▶ Suit le principe d'un client lourd : composants, événements
 - ▶ Spécification Java Community Process (JSR 127)
 - ▶ Implémentation de Sun et de Apache (MyFaces)
- Fournit les fonctionnalités suivantes
 - ▶ Composants graphiques persistant entre les requêtes
 - ▶ Notion de "binding" entre les composants et le modèle
 - ▶ Abstraction de la notion de formulaire
 - ▶ Abstraction par rapport aux API JSP/Servlet (Vue client lourds)
 - ▶ Conversion des données saisies
 - ▶ Validation des données saisies
 - ▶ Prise en charge de la navigation entre pages
 - ▶ I18N (internationalisation)
 - ▶ Extensibilité

Fonctionnement de JavaServer Faces

185



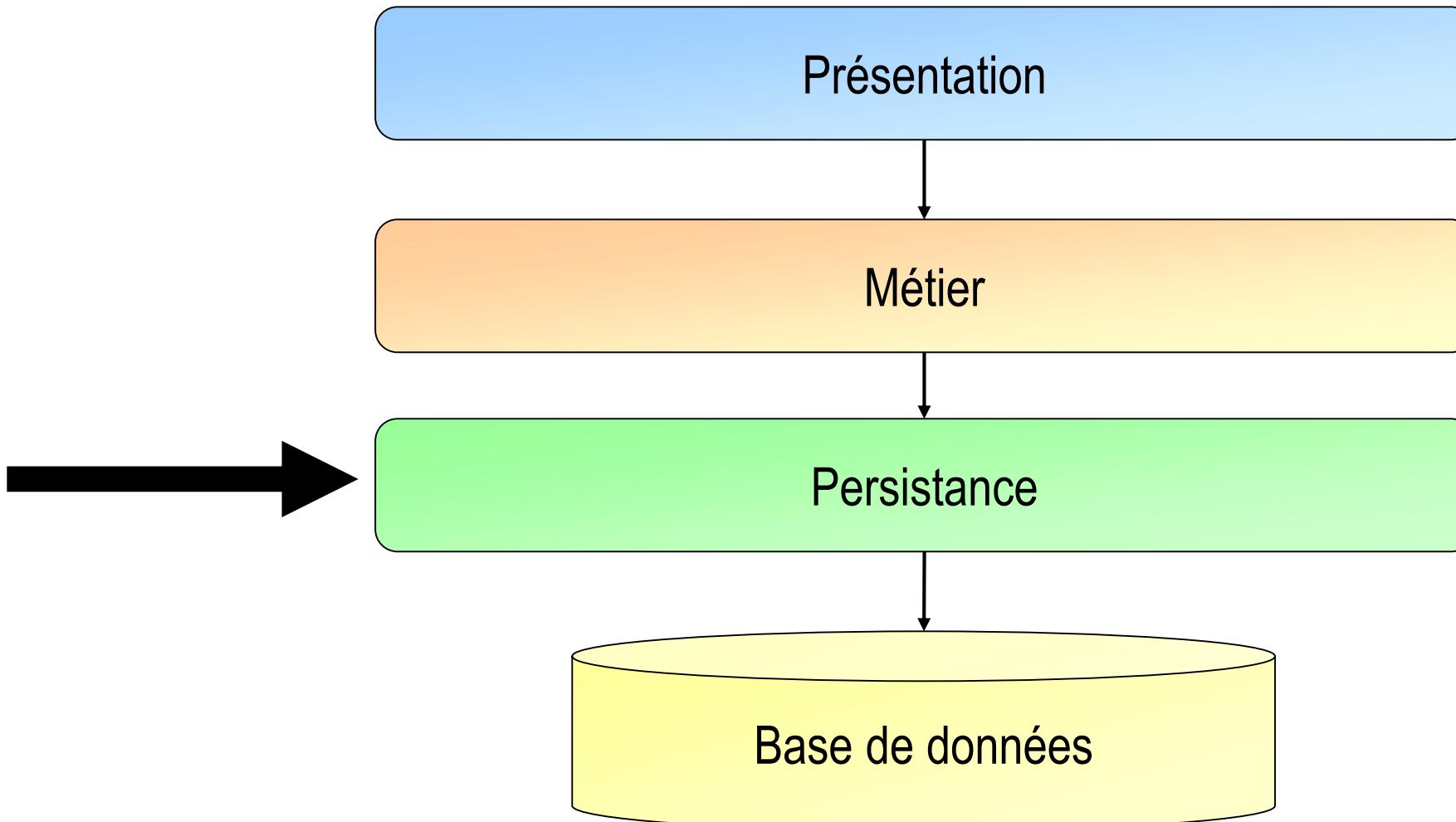
Positionnement des frameworks Web

186

- L'API des Servlets permet
 - ▶ de traiter une requête HTTP dans le monde Java sans formalisme ni abstraction
- Struts permet
 - ▶ de formaliser le traitement d'une requête HTTP...
 - Ex : phase de remplissage d'un formulaire, phase de validation d'un formulaire, traitement de l'action utilisateur, annulation, ...
 - ▶ ...sans abstraction par rapport aux API des Servlets en implémentant le pattern MVC2
- JSF permet
 - ▶ de formaliser le traitement d'une requête utilisateur au sens large en s'abstrayant des API des Servlets en étendant le pattern MVC2

Les frameworks de persistance

187



Pourquoi ?

- L'utilisation directe de JDBC présente de nombreuses limites
 - ▶ Adhérence dans le code aux noms des schémas, des tables, des colonnes
 - ▶ Répétitivité de l'écriture des requêtes SQL qui doivent toutes être écrites
 - ▶ Évolution et maintient du code difficile
 - ▶ Pas de fonctionnalité haut niveau (ex : liens entre enregistrements, chargement à la demande, héritage, ...)
 - ▶ Problèmes d'optimisation (ex : nombre de requêtes sur les transactions longues)

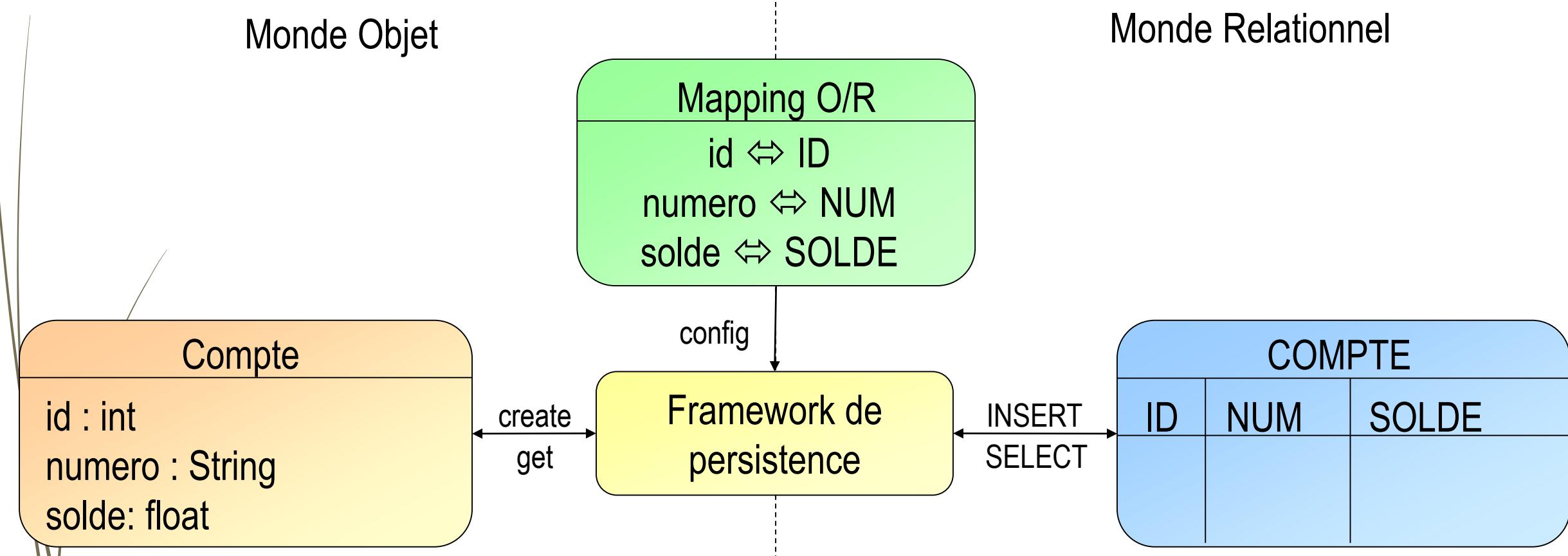
Pourquoi ?

➤ L'idée

- ▶ Définir une fois pour toutes la correspondance entre le monde objet et le monde relationnel puis laisser le framework générer lui même les requêtes SQL (INSERT, DELETE, SELECT, UPDATE)
- ▶ Cette correspondance se nomme "Mapping Objet/Relationnel" (ORM)

Mapping Objet/Relationnel

190



Présentation d'Hibernate

191

- Framework de persistance Java focalisé sur les bases de données relationnelles
 - ➡ développé par Gavin King en 2001
 - ➡ Version actuelle 5
- Download accessible depuis www.hibernate.org
- Hibernate est gratuit, et disponible sous Licence LGPL
- Hibernate a rejoint le JBoss Group
 - ➡ Moyens plus importants
 - ➡ S'inscrit dans la stratégie globale de JBoss

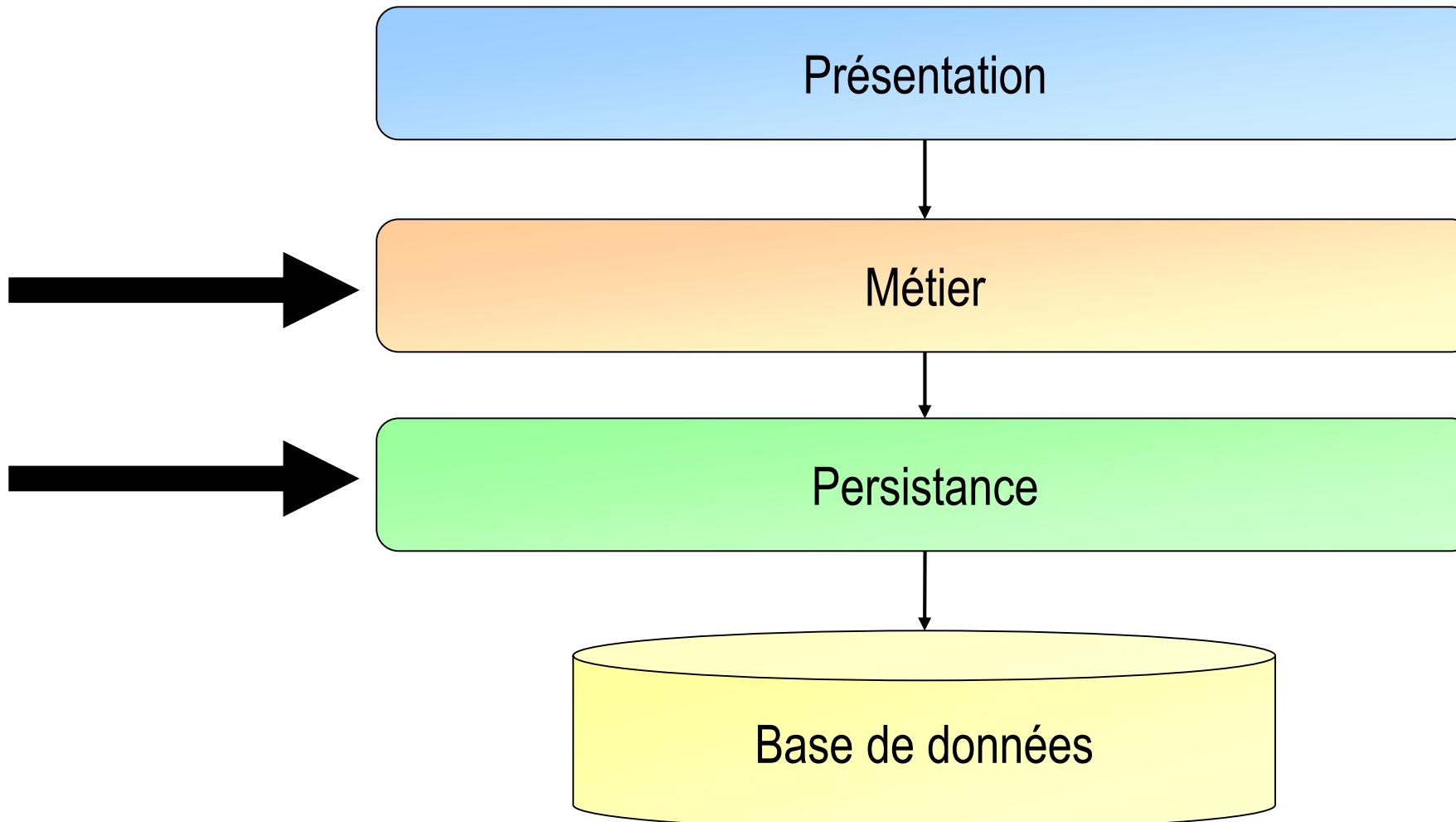
Description d'Hibernate

192

- Hibernate manipule et fait persister des POJOs
 - ▶ POJO → Plain Old Java Objects "bons vieux objets Java"
 - ▶ Pas d'interface particulière à implémenter
- Hibernate est implémenté en Java et possède des API Java
- Hibernate est compatible avec les standards Java SE et JEE
 - ▶ Compatibilité avec tous les serveurs d'application JEE du marché
 - Websphere, Weblogic, JBoss, JOnAS, etc..
 - ▶ JNDI, JDBC, JTA
 - ▶ S'intègre dans une architecture à base d'EJB Session en façade
 - ▶ Rentre en concurrence directe avec les EJBs Entité CMP
- L'API Java d'Hibernate est propriétaire
- Compatibilité avec la plupart des SGBDR
 - ▶ Plus de 20 (ex: Oracle, MySQL, Sybase, PostgreSQL, DB2, ...)

La norme EJB

193



La norme EJB

- Les EJBs sont des composants Java qui résident dans le container EJB des serveur d'application JEE
- Les EJBs sont des composants
 - ▶ Distribués (accessibles à distance, ex: RMI)
 - ▶ Portables
 - ▶ Persistants
 - ▶ Transactionnels
- Actuellement en version 3
 - ▶ EJB 3.0 intégré à JEE5
 - ▶ EJB 3.1 intégré à JEE 6

La notion de composant

- Les composants sont des morceaux de programme développés et déployés de façon indépendante et qui présentent une interface permettant leur assemblage
 - ➡ Analogie avec l'électronique, notion de boîte noire
- La spécification EJB détaille comment implémenter ces composants mais elle ne remplace pas l'analyse qui permet de les identifier

Objectifs des EJBs

- Permettre aux développeurs de se focaliser sur la logique métier sans avoir à se préoccuper des problèmes liés aux architectures distribuées
 - ▶ Gestion des communications réseaux
 - ▶ Gestion du multi-threading et des accès concurrents
 - ▶ Gestion des transactions
 - ▶ Gestion de la persistance
- Offrir une solution standard
 - ▶ indépendante des OS et des fournisseurs de serveur d'applications
- Couvrir les phases de développement, déploiement et exploitation

Les différents types d'EJB

- Trois utilisations de composants
 - ▶ l'application cliente accède à la logique métier résidant sur le serveur
 - ▶ l'application cliente manipule un objet métier représentant des données persistantes
 - ▶ l'application cliente dépose une demande de traitement asynchrone. Ce traitement sera réalisé par un récepteur asynchrone.
- Trois types d'EJB pour répondre à ces besoins
 - ▶ Session Bean
 - ▶ Entity Bean
 - ▶ Message Driven Bean

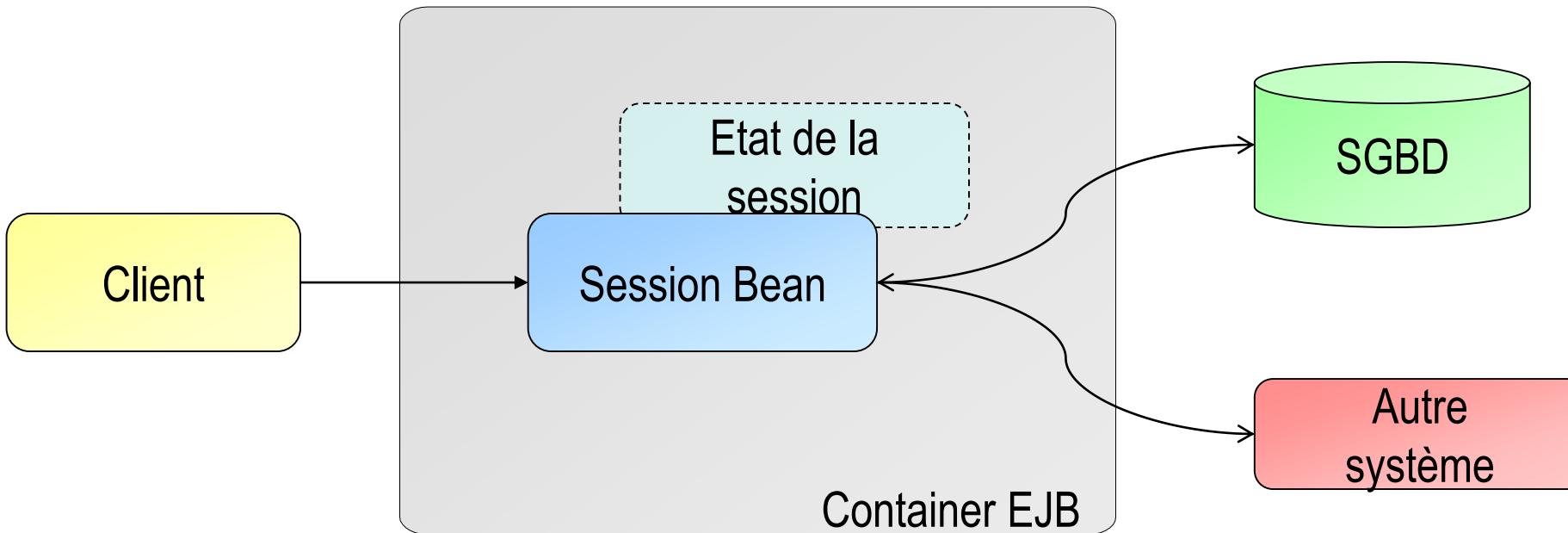
Présentation des EJB Session

198

- Représente une communication avec un client
- Est accédé par un seul client à la fois
- Peut être vu comme une extension du client dans certains cas
- Manipule généralement à des objets métier persistés
- Peut participer à une transaction, mais son état n'est pas transactionnel
- Deux types
 - ▶ Sans état (Stateless)
 - ▶ Avec état (Stateful)

Architecture à base d'EJB Session

199



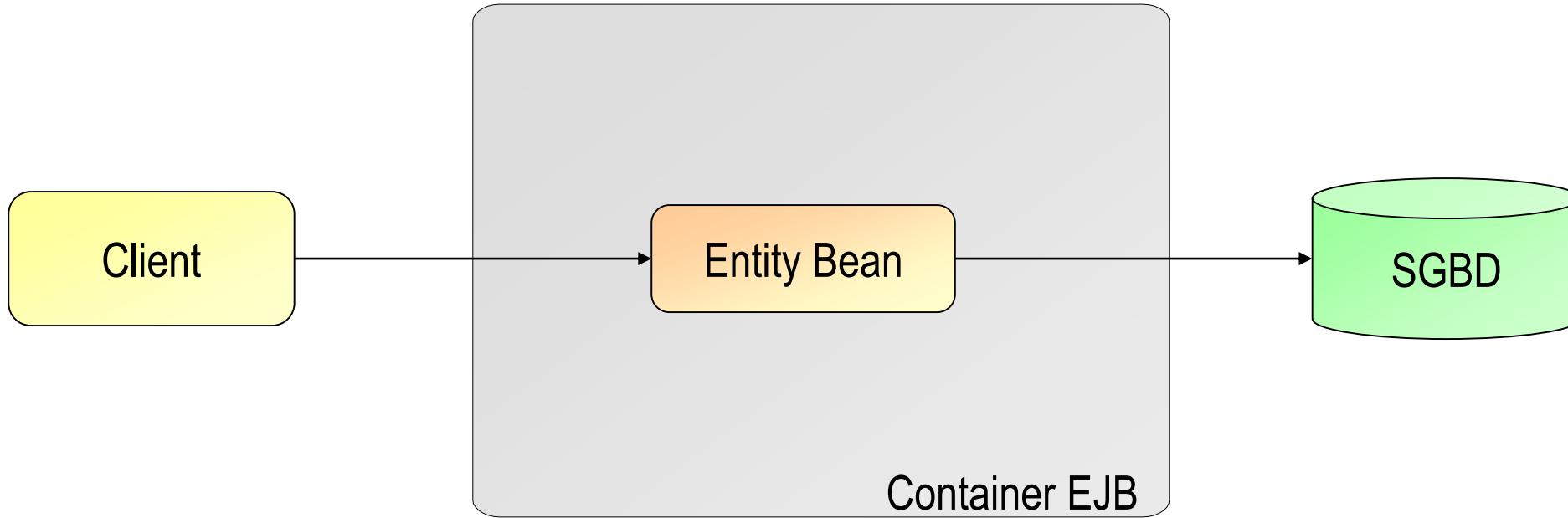
Présentation des EJB Entités

200

- Représente des données persistantes
 - ➡ Ex : une entrée dans une table d'un SGBDR
- A une durée de vie indépendante du serveur d'EJB
 - ➡ Les données en base permettent de reconstruire l'EJB
- Géré de façon transactionnelle
- Deux types
 - ➡ BMP (Bean Managed Persistence)
 - ➡ CMP (Container Managed Persistence)
- Concurrent des frameworks de persistance
 - ➡ Ex : Hibernate

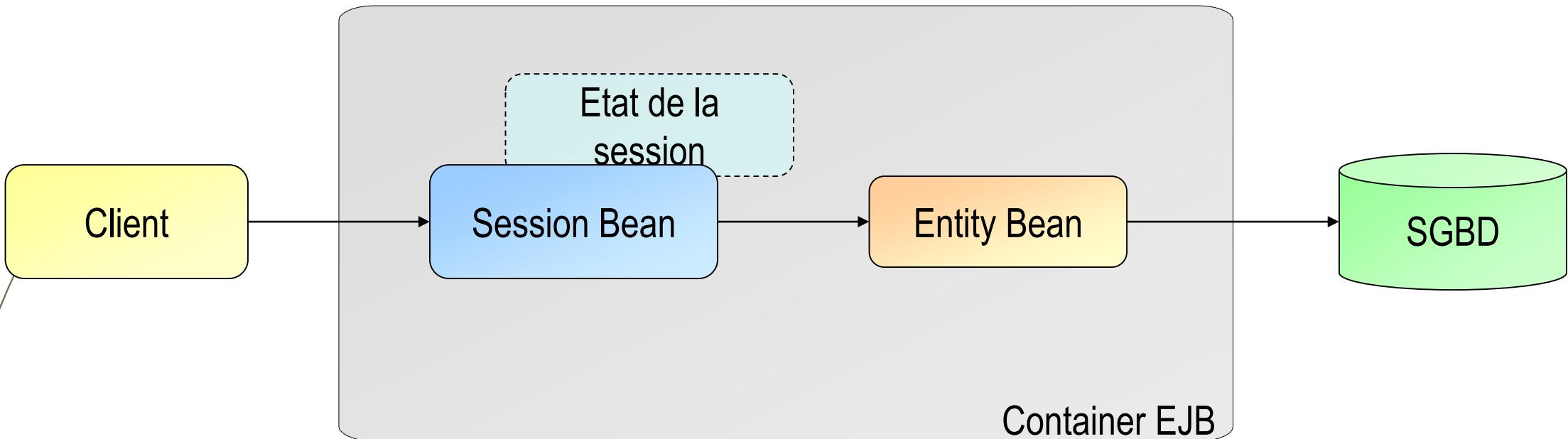
EJB Session et EJB Entités

201



Architecture à base d'EJB Entités

202



Présentation des Message Driven Bean

203

- Un Message Bean est un bean qui peut être activé par un message provenant d'un middleware orienté message (ex : MQSeries)
 - ▶ Récepteur asynchrone suite à l'envoi d'un message sur une file d'attente ou à propos d'un sujet pré-déterminé
 - ▶ Utilisation de JMS (Java Messaging Service)
- Traitements asynchrones de messages JMS
- L'utilisateur demande un travail, mais n'attend pas la réponse
 - ▶ Demande d'impression
 - ▶ Demande de production de carte sim

Architecture MDB

204

