

M1IF03

Conception d'applications Web



TECHNOLOGIES CÔTÉ SERVEUR (APPLICATIONS WEB EN JAVA)

LIONEL MÉDINI
SEPTEMBRE 2023

Plan du cours

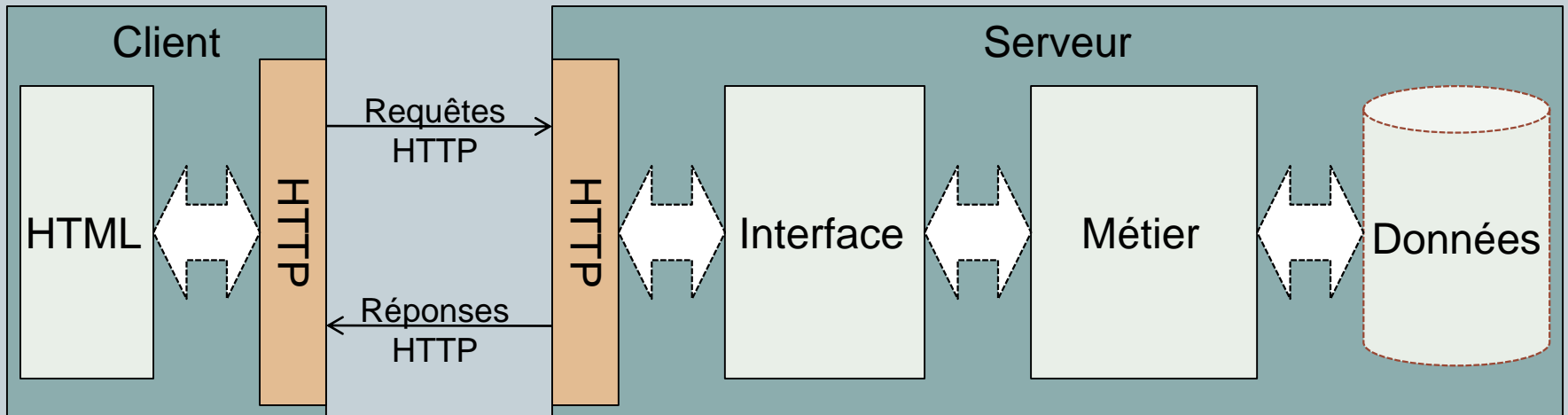


- **Introduction**
 - Application Web : retour sur la définition
 - Différents types d'applications
 - Différentes technologies de programmation côté serveur
- **Programmation côté serveur en Java**
- **Application Web en Java**
- **Conclusion**

Application Web



- Définition
 - Application dont l'interface est visible dans un navigateur
 - ✦ Nécessairement des programmes côté serveur
 - ✦ Parfois une partie côté client
 - Dépendent de l'infrastructure web choisie
 - Exemple



Applications Web



- Différents types d'applications
 - Site Web dynamique
 - ✦ Ensemble de pages dynamiques simples
 - éventuellement inter-reliées
 - mais pas d'inclusion
 - ✦ Exemples
 - Traitement des données d'un formulaire
 - Composition pour l'affichage de données du serveur

Applications Web



- Différents types d'applications
 - Application Web « localisée »
 - ✦ Toute la programmation est sur le serveur
 - ✦ Modèle(s) de structuration de l'application
 - couches, modules, composants, aspects...
 - ✦ Principes de structuration (*cf.* M1IF01)
 - Affectation de responsabilités à chaque ressource
 - Inclusion/appel des ressources les unes dans les autres
 - Utilisation de patterns de conception (MVC, DAO...)
 - ✦ Spécificité des applications Web
 - Aiguillage du flux applicatif par redirection HTTP
 - Choix des types d'objets (classe, servlet, JSP) en fonction du type de composant (métier, interface, données...)

Applications Web



- Différents types d'applications
 - Application Web localisée + AJAX
 - ✦ Cf. transparent précédent
 - ✦ Déport d'une partie de l'application côté client
 - cf. cours suivants...

Applications Web



- Différents types d'applications
 - Application Web côté client
 - ✦ Cf. transparent précédent
 - ✦ Uniquement des ressources statiques côté serveur
 - ✦ Côté client
 - Modèle : scripts métier
 - Vue : moteur de templates
 - Contrôleur : routeur (framework)
 - ✦ Récupération de données en AJAX
 - ✦ Éventuellement, interrogation de différentes sources de données
 - Requêtage cross-domaine → mashup
 - ✦ Cf. M1IF13... Ou pas.

Applications Web



- Différents types d'applications
 - Application Web répartie
 - ✦ Application localisée (AJAX ou non) +
 - ✦ Appel à d'autres ressources / composants
 - sur des machines distantes
 - dont on n'est pas nécessairement propriétaire
 - ✦ Nécessite des mécanismes (« middleware ») de communication
 - Exemples : RPC, CORBA, Services Web, REST...
 - ✦ Nécessite une modélisation du déploiement
 - référencement/connectivité avec les ressources distantes, performances, sécurité...
 - ✦ Cf. cours de M2

Applications Web



- Exemples de technologies
 - Php
 - ✦ Langage interprété
 - ✦ Type de programmation : scripts / fonctions / objets
 - ✦ Moteur : interpréteur existant sur la quasi-totalité des serveurs
 - Java
 - ✦ Bytecode
 - ✦ Type de programmation : classes (servlets), scripts (JSP)...
 - ✦ Moteur : container de servlets Catalina (+ Apache = Tomcat)
 - Microsoft™ .Net Framework
 - ✦ Ensemble de technologies de développement
 - ✦ Type de programmation : dépend du langage VB, C#, J#, ASP...
 - ✦ Moteur : framework sur serveur IIS
 - Python
 - ✦ Langage interprété
 - ✦ Type de programmation : scripts python, scriptlets, DTML...
 - ✦ Moteur : serveur d'applications Zope, Plone
 - ...

Applications Web



- Dans ce cours
 - Applications Web localisées
 - Patterns / bonnes pratiques
 - ✦ Client-serveur
 - ✦ Programmation déclarative
 - ✦ Chaîne de responsabilités
 - ✦ Couches
 - ✦ MVC
 - Programmation en Java
 - ✦ Servlet API
 - ✦ Maven Webapp Archetype

Les spécifications Java « entreprise »



- Spécifications de haut niveau
 - ✦ Web, objets distants, objets transactionnels distribués...
 - ✦ Les packages qui commençaient par `javax`.
- Historique
 - 1999 (Java 2 SE)
 - ✦ l'ensemble des spécifications est « sorti » de Java 2 SE -> Java 2 EE
 - 2006 (Java 5 SE)
 - ✦ le 2 disparaît -> Java EE
 - ✦ les numéros de versions « suivent » ceux de Java SE : JEE5
 - 2017 (Java 7 SE)
 - ✦ Oracle « libère » la spécification Java EE -> Fondation Eclipse
 - ✦ La Fondation Eclipse renomme la spec en **Jakarta**

Les spécifications Java « entreprise »



- Historique (suite)
 - Sept. 2019 (Java SE 8)
 - ✦ Jakarta EE 8 :
 - Compatible avec l'ancienne spec Java EE 8 → Tomcat 9
 - Déc. 2020 (Java SE 8)
 - ✦ Jakarta EE 9 :
 - Incompatible avec l'ancienne spec → Tomcat 10.0
 - Renommage des packages : `javax` → `jakarta`
 - Mai 2022 (Java SE 11)
 - ✦ Jakarta EE 10 :
 - « Nettoyage » de la spec → Tomcat 10.1
 - Suppression de certaines spec obsolètes

Plan du cours



- Introduction
- **Programmation côté serveur en Java**
 - Principes
 - Servlets
 - JSP
 - Javabeans
 - Taglibs
 - Filtres
- Application Web en Java
- Conclusion

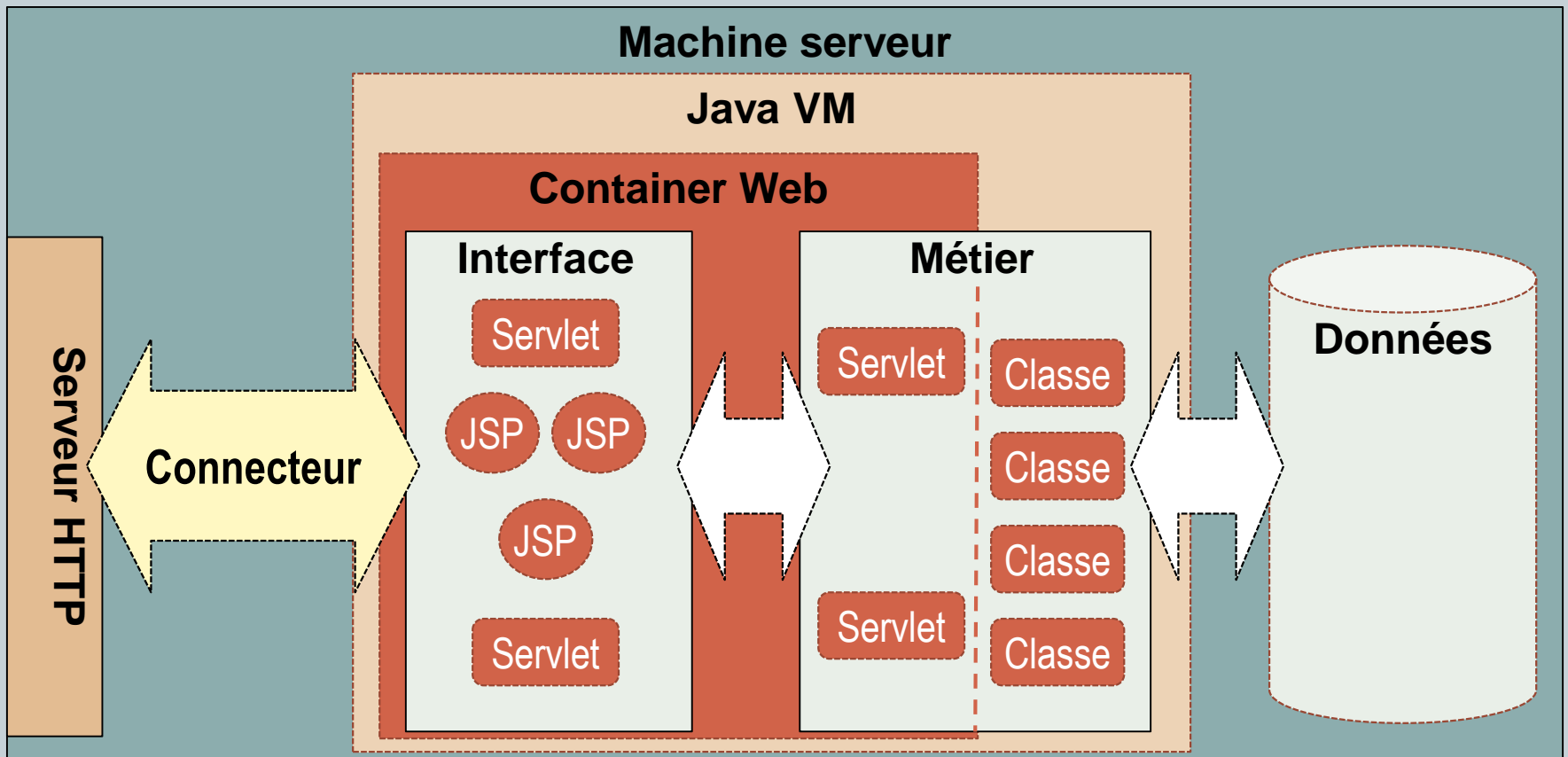
Programmation côté serveur en Java



- **Principes de la programmation côté serveur en Java**
 - Réception de la requête du client → Serveur Web
 - Encapsulation de la requête client dans un objet Java
`HttpServletRequest` → Moteur de servlets
 - Traitement de la requête et génération de la réponse sous forme d'un objet Java
`HttpServletResponse` → Composants Java (servlets, JSP, classes, interfaces, JavaBeans...)
 - Désencapsulation de la réponse → Moteur de servlets
 - Envoi de la réponse au client → Serveur Web

Programmation côté serveur en Java

- Principes de la programmation côté serveur en Java



Programmation côté serveur en Java



- Quelques outils disponibles

- [Tomcat](#)

- ✦ Projet d'Apache issu de Apache Jakarta™ (arrêté en 2011)

- Référence en matière de moteurs de servlets

- ✦ Contenu

- Serveur web : Apache

- Connecteur : mod_jk (Jakarta)

- Protocole : Apache JServ (AJP1.3)

- Moteur de servlets : Catalina

- Compilateur de JSP : Jasper

- [Jetty](#)

- ✦ Serveur + conteneur de servlets : « léger », issu d'Eclipse

- ...

➔ Ces outils implémentent [Jakarta Servlet API](#)

Servlets



- **Définition (officielle)**

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

- Servlets are protocol- and platform-independent server side components, written in Java, which dynamically extend Java enabled servers.
- They provide a general framework for services built using the request-response paradigm.
- Their initial use is to provide secure web-based access to data which is presented using HTML web pages, interactively viewing or modifying that data using dynamic web page generation techniques.
- Since servlets run inside servers, they do not need a graphical user interface.

Servlets



- Définition (courte)
 - Implémentation Java d'un mécanisme de requête/réponse
 - ✦ Initialement : indépendant d'un protocole
 - ✦ Avec encapsulation des données dans des objets
 - Générique
 - Requête
 - Réponse
 - Contexte applicatif
 - Spécifique HTTP
 - Méthode
 - Type MIME de la réponse
 - Headers
 - Session
 - Cookies

Servlets

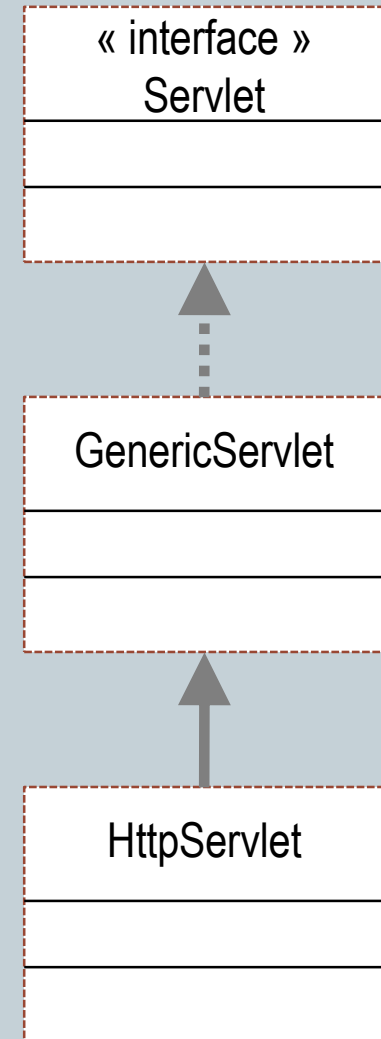


- **Concrètement**
 - **Objet (classe) Java**
 - ✦ Composant d'application
 - ✦ Derrière un serveur (Web, mais pas seulement)
 - ✦ Mappée à une URL sur le serveur
 - **Dans un « Container »**
 - ✦ Pas d'accès direct au serveur
 - ✦ Accès protégé aux autres objets métier de l'application
 - ✦ Gestion avancée par le container

Servlets



- L'API Servlet
 - Packages Java
 - ✦ jakarta.servlet
 - Servlet : interface
 - GenericServlet : classe abstraite
 - ✦ jakarta.servlet.http
 - HttpServlet : classe d'implémentation
 - Méthodes
 - ✦ Gestion du cycle de vie
 - ✦ Service



Servlets



- Méthodes de gestion du cycle de vie
 - Sont appelées par le conteneur
 - ✦ après l'instanciation (pour rendre une servlet opérationnelle) ou
 - ✦ en fin de service (avant le garbage collecting)
 - Permettent des traitements spécifiques à l'application
 - ✦ Chargement / déchargement de données de configuration
 - ✦ Activation de services annexes (logs, persistance...)

Servlets



- **Méthodes de gestion du cycle de vie**
 - `jakarta.servlet.GenericServlet`
 - ✦ `public void init(ServletConfig config) throws ServletException`
 - Il faut appeler `super.init(config)` en surchargeant cette méthode
 - ✦ `public void init() throws ServletException`
 - Inutile d'appeler `super.init()` ; il vaut mieux surcharger celle-ci
 - ✦ `public void destroy()`

Servlets



- Méthodes de service
 - Permettent de rendre le service
 - ✦ traitement de la requête
 - ✦ génération de la réponse
 - Implémentation différente avec/sans protocole HTTP
 - ✦ GenericServlet : une seule méthode
 - ✦ HttpServlet : une méthode (de classe) par méthode (HTTP)
 - Utilisation
 - ✦ GenericServlet
 - surcharger la méthode de service (abstraite)
 - ✦ HttpServlet
 - surcharger au moins une méthode de service

Servlets



- **Méthodes de service**

- `jakarta.servlet.GenericServlet`

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

- `jakarta.servlet.http.HttpServlet`

- ✦ `protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException`

- ✦ `protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException`

- ✦ `... doDelete, doHead, doOptions, doPut, doTrace`

Servlets



- **Accès aux données encapsulées**

- *Via les objets requête et réponse passés en paramètres des méthodes de service*

- ServletRequest

- ✦ getParameter

- HttpServletRequest

- ✦ getCookies

- ✦ getHeader

- ✦ getMethod

- ✦ getSession

- ServletResponse

- ✦ getWriter

- HttpServletResponse

- ✦ addCookie

- ✦ addHeader

- ✦ sendError

- ✦ sendRedirect

Servlets



- Exemple de code (HTTPServlet, Servlet API V2)

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class NewServlet extends HttpServlet {

    public void init(ServletConfig config) throws ServletException {
        super.init(config); ... }
    public void destroy() { ... }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>Hello page</title></head>");
        out.println("<body><h1>Hello "+ request.getParameter("name") +
            </h1></body></html>");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException { ... }
}
```

Servlets



- Conclusion sur les servlets
 - Avantages
 - ✦ Composants simples...
 - Classes Java
 - ✦ ...pratiques...
 - Codage minimum : cycle de vie, traitement de la requête
 - Tous les autres aspects sont pris en charge par le conteneur
 - ✦ ...et sûrs
 - Isolation du serveur par le conteneur
 - « rigueur » de l'orienté-objet
 - Inconvénients
 - ✦ Beaucoup de `out.println()`
 - ✦ Difficile de comprendre le code HTML généré

Jakarta Server Pages



- **Principe**
 - Écrire une page Web dynamique comme si elle était statique
 - Ne mettre du code que quand nécessaire
 - ➔ Scripting à la Php
- **Même fonctionnalités que HttpServlet**
 - Implémentation du mécanisme requête/réponse
 - ✦ Accéder aux mêmes données/objets qu'une servlet
 - ✦ Inclure ou rediriger la requête vers une autre servlet/JSP
 - Spécifique à HTTP
 - ✦ Génération de différents types de contenus : HTML, XML, SVG...
 - ✦ Gestion des méthodes, headers, cookies, sessions...

Jakarta Server Pages



- **Format simplifié**
 - Programmation descriptive
 - ✦ (X)HTML classique
 - ✦ Scripts : code « HTML-like » qui doit être compilé en code Java
 - Bibliothèques de tags spécifiques
 - Définition de balises personnalisées
 - Programmation impérative
 - ✦ Code Java à traiter directement par la JVM du serveur

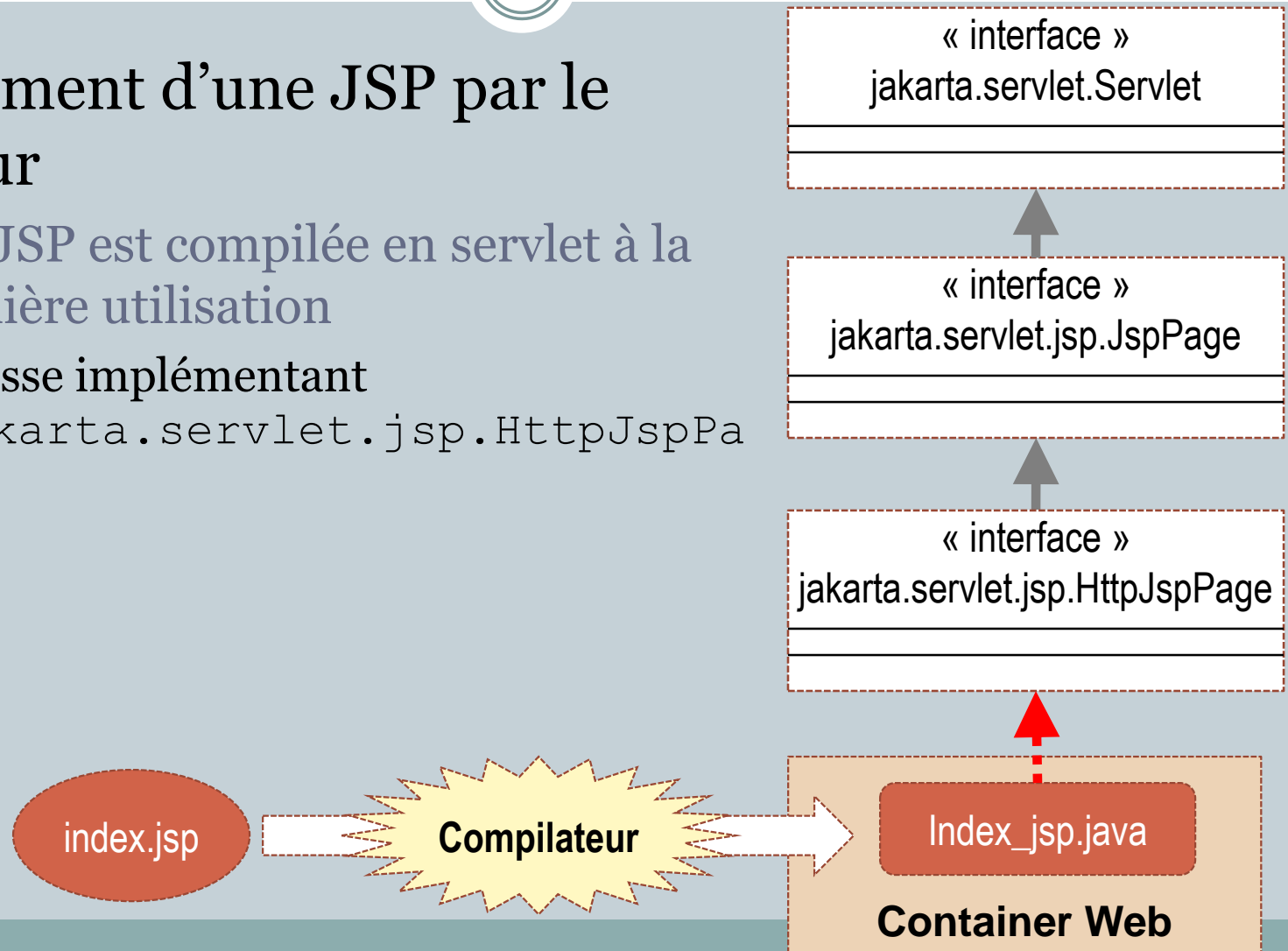
Jakarta Server Pages



- L'API JSP
 - Packages Java
 - ✦ jakarta.servlet.jsp
 - ✦ jakarta.servlet.jsp.el
 - ✦ jakarta.servlet.jsp.tagext
 - Méthodes identiques à celles de l'API Servlet
 - ✦ Gestion du cycle de vie
 - ✦ Service

Jakarta Server Pages

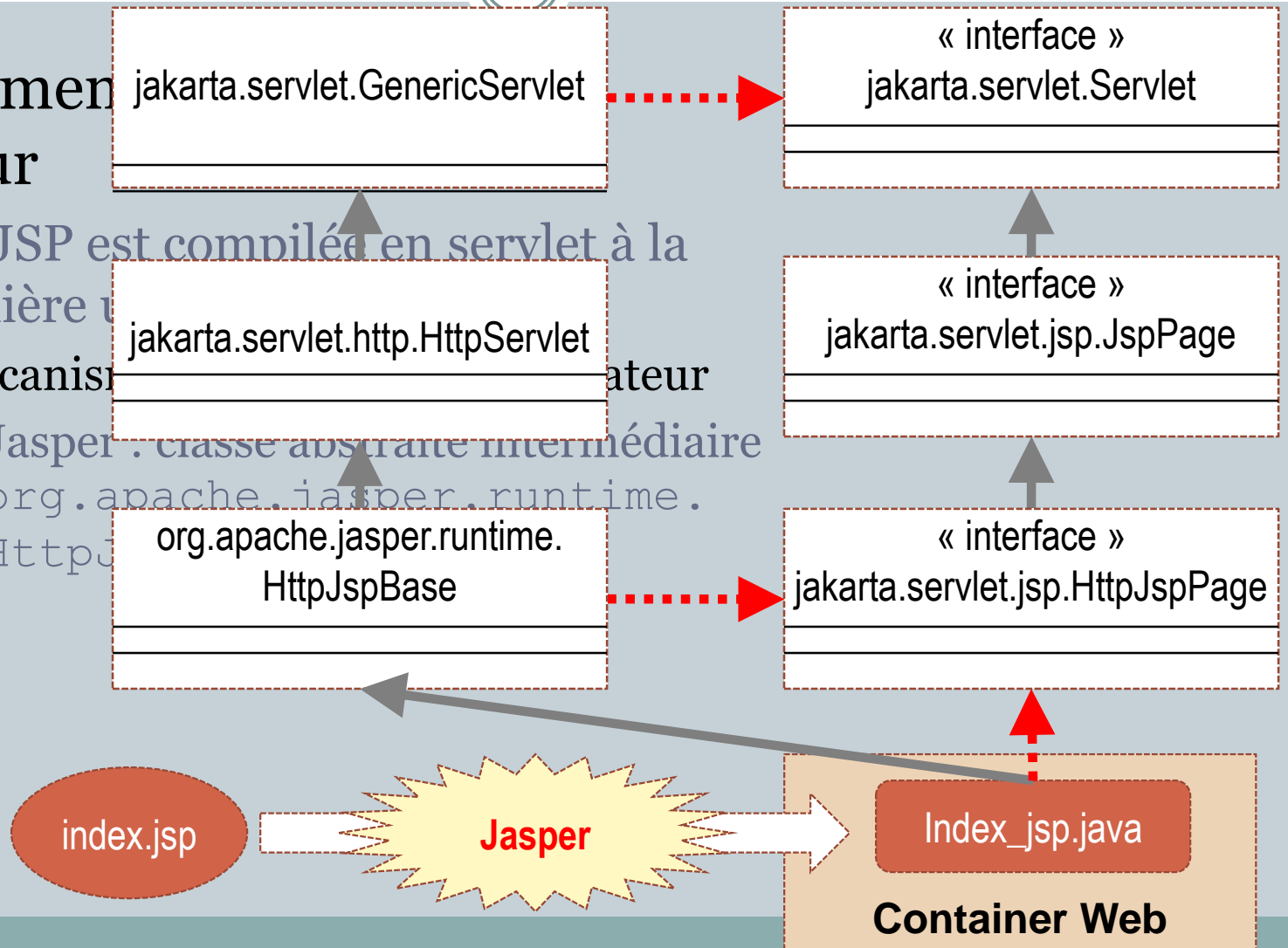
- Traitement d'une JSP par le serveur
 - Une JSP est compilée en servlet à la première utilisation
 - ✦ Classe implémentant `jakarta.servlet.jsp.HttpJspPage`



Jakarta Server Pages

- **Traitement serveur**

- Une JSP est compilée en servlet à la première utilisation
 - ✦ Mécanisme de compilation
 - Jasper : classe abstraite intermédiaire
 - `org.apache.jasper.runtime.HttpJspBase`



Jakarta Server Pages



- **Syntaxe**
 - **Syntaxes « classiques »**
 - ✦ Balises de scripts
 - ✦ XML : plusieurs espaces de noms
 - jsp
 - user-defined (JSTL, etc.)
 - ✦ Expression language
 - ➔ **Peuvent être combinées/imbriquées**

Jakarta Server Pages



- **Scriptlets : `<% code %>`**
 - Morceaux de code (blocs d'instructions) Java dans la page
 - Syntaxe XML

```
<jsp:scriptlet>  
response.setContentType("text/plain");  
</jsp:scriptlet>
```
 - Syntaxe script

```
<% response.setContentType("text/plain"); %>
```

Jakarta Server Pages



- Variables prédéfinies dans les scriptlets
 - **request**
 - **response**
 - **out**
 - **session**
 - **application**
 - **page**
 - ...

Jakarta Server Pages



- Expressions : `<%= code %>`
 - Des expressions, qui sont évaluées et insérées dans la méthode de service de la servlet
 - Syntaxe XML

```
<jsp:expression>
  new java.util.Date()
</jsp:expression>
```
 - Syntaxe script

```
<%= new java.util.Date() %>
```
 - Equivalent à

```
<% out.println(new java.util.Date()); %>
```

Jakarta Server Pages



- Déclarations : `<%! code %>`
 - Permettent de définir des méthodes ou des champs qui seront insérés dans le corps de la servlet
 - Syntaxe XML

```
<jsp:declaration>  
private int VariableGlobale = 0;  
</jsp:declaration>
```
 - Syntaxe script

```
<%! private int VariableGlobale = 0; %>
```

Jakarta Server Pages



- Directives : `<%@ code %>`
 - Informations globales relatives à la page
 - Trois types de directives
 - ✦ **page** : modifier les données de la page (import de packages, spécification d'un type de contenu, gestion des sessions)
`<%@ page import="java.util.*" %>`
 - ✦ **include** : inclure des fichiers ou autres servlets/JSP
`<%@ include page="/monJSP.jsp" flush="true" %>`
 - ✦ **taglib** : utiliser des bibliothèques de balises personnalisées
`<%@ taglib uri="..." prefix="..." %>`

Jakarta Server Pages



- Exemple de code simple : une JSP qui compte le nombre de fois où elle a été appelée

```
<html>
  <head><title>Déclarations et
expressions</title></head>
  <body>
    <h1>Déclarations JSP</h1>

    <%! private int accessCount = 0; %>
    <p>Cette page a été accédée <%= ++accessCount %>
fois depuis le démarrage du serveur</p>

  </body>
</html>
```

Jakarta Server Pages



- **Les bibliothèques de tags**

- Permettent de prendre en charge différentes fonctionnalités sans écrire de code Java
- Sont reliées à des classes Java, mappées à l'exécution des tags
- Exemples

- ✦ **La Java Standard Tag Library (JSTL)**

- **Plusieurs bibliothèques de tags**

- **Core**

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

- **XML**

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

- **Internationalisation**

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

- **SQL**

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

- **Fonctions**

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

Jakarta Server Pages



- Les bibliothèques de tags
 - Permettent de prendre en charge différentes fonctionnalités sans écrire de code Java
 - Sont reliées à des classes Java, mappées à l'exécution des tags
 - Exemples
 - ✦ Autres bibliothèques de code
 - « Quasi-standards » disponibles sur Internet
 - `<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean" prefix="bean" %>`
 - `<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html" %>`
 - `<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic" prefix="logic" %>`
 - Que vous définirez pour une application donnée
 - Non présenté en cours
 - Pointeur : <http://adiguba.developpez.com/tutoriels/j2ee/jsp/taglib/>

Jakarta Server Pages



- **La syntaxe EL (Expression Language)**
 - Depuis JSP 2.0
 - Depuis JSP 2.1 (mai 2006) : Unified EL (commun avec JSF)
 - S'utilise
 - ✦ dans les attributs de tags JSP
 - ✦ directement dans le corps de la page
 - **Syntaxe**
 - ✦ `${ expression_el }`
 - **Exemple**

```
<c:forEach var="customer" items="${customers}"  
    Customer: <c:out value="${customer}"/>  
</c:forEach>
```

Jakarta Server Pages



- La syntaxe EL (Expression Language)
 - Permet d'accéder facilement à
 - ✦ des objets de la page
 - `${totoBean}`
 - ✦ des propriétés de beans
 - `<c:out value="${totoBean.nom}" />`
 - ✦ des objets implicites
 - `${header['User-Agent']}`
 - ✦ des opérateurs, des fonctions, des variables, etc.

Jakarta Server Pages



- La syntaxe EL (Expression Language)
 - Les objets implicites
 - ✦ Accès aux différents composants d'une page JSP
 - pageContext, initParam
 - param, paramValues, header, headerValues, cookie
 - pageScope, requestScope, sessionScope, applicationScope
 - ✦ Exemples
 - `${ pageContext.response.contentType }`
 - `${ pageScope["name"] }`
 - `${ param["page"] }`
 - `${ header["user-agent"] }`

Jakarta Server Pages



- **La syntaxe EL (Expression Language)**
 - **Lien avec Java**
 - ✦ **Types primaires**
 - Convertis en objet du type de la classe wrapper correspondante
 - `java.lang.Long`, `java.lang.String`...
 - ✦ **Opérateurs**
 - Unaires et binaires classiques : `+`, `-`, `&&`, `and`, `!`, `not`
 - **Ne pas interpréter les EL**
 - ✦ **Dans toute la page**

```
<%@ page isELIgnored="true" %>
```
 - ✦ **Au coup par coup**

```
\${ ceci ne sera pas interprété comme une EL }
```

Les JavaBeans



- **Définition (1996)**
 - *Composants logiciels réutilisables* d'applications
 - En pratique : des classes Java
- **Structure**
 - Un constructeur sans paramètre
 - Des propriétés cachées et accessibles par des méthodes publiques
 - ✦ `public String getNom()` et `public void setNom(String valeur)`
 - Les autres méthodes sont privées
- **Intérêt**
 - Composants possédant une forme standardisée
 - ➔ Utilisation facilement déductible par introspection
 - En Web

Déporter de la logique (métier, données) hors de la présentation

Les JavaBeans



- Utilisation avec des JSP

- Définir le bean

- ✦

```
<jsp:useBean id="toto" class="package.TotoBean" scope="request" />
```

- L'utiliser

- ✦

```
<jsp:setProperty name="toto" property="nom" value="<%= request.getParameter(\"nom\") %>" />
```

```
<h1> hello <jsp:getProperty name="toto" property="nom" /></h1>
```

- ✦ **ou**

```
<h1> hello <%= toto.getName() %></h1>
```

Les JavaBeans



- **Notion de scope (portée)**
 - Précisée dans la déclaration JSP
 - Modifie la portée de la variable dans la servlet générée
 - Valeurs autorisées
 - ✦ page
 - Variable interne à la méthode de réponse
 - Non transmise en cas d'inclusion ou de redirection
 - ✦ request
 - Variable interne à la méthode de réponse
 - Transmise en cas d'inclusion mais pas de redirection (attribut de requête)
 - ✦ session
 - Attribut de session
 - ✦ application
 - Variable globale (définie dans le contexte applicatif)

Retour sur JSP / Servlets



- **Quoi mettre où ? (travailler en servlets ou en JSP ?)**
 - Dépend de la quantité de code Java / HTML
 - Dépend de la couche dans laquelle on se trouve
 - Plus objectivement
 - ✦ **Servlets**
 - Aiguillage des données
 - Accès aux autres méthodes que celles de service
 - Définition d'autres méthodes
 - ✦ **JSP**
 - Inclusions d'autres JSP / intégration de composants
 - Appel à des JavaBeans
 - Utilisation de taglibs

Pattern Web MVC



- **Principe**
 - Mise en place du pattern MVC (voir CM M1IF01) en Java côté serveur
 - Implémentation en Java : servlets / JSP / JavaBeans
- **Modèle**
 - Contient le domaine de l'application
 - Peut utiliser d'autres patterns
 - Implémenté sous forme de classes / interfaces simples
 - ✦ Beans
 - ✦ POJOs

Pattern Web MVC



- **Contrôleur**
 - Rôle
 - ✦ Gère les paramètres des requêtes
 - ✦ Lie modèle et vue
 - ✦ Peut effectuer de la conversion simple de paramètres (non recommandé)
 - Implémentation
 - ✦ Servlet
 - Remarques
 - ✦ Dans les applications complexes, un contrôleur est lié à une **action** spécifique
 - ✦ Plusieurs contrôleurs délégués (en-dessous d'un contrôleur principal)
 - ✦ Dans certains frameworks, existence d'une classe Action
 - Formalise les relations entre modèle et vue,
 - Adaptation des paramètres

Pattern Web MVC



- **Vue**
 - **Rôle**
 - ✦ Encapsulent la création des pages Web de réponses
 - ✦ Soit à partir d'un ensemble d'objets et de valeurs « préparés » par l'action liée au contrôleur
 - ✦ Soit en faisant appel au modèle (beans) pour récupérer des données
 - **Implémentation**
 - ✦ JSP
- **Remarques**
 - MVC « pull-based » : la vue interroge le modèle (beans)
 - MVC « push-based » : le contrôleur passe les données à la vue (actions)
 - ✦ Dans ce cas, les données sont passées sous forme de paires clés-valeurs dans les attributs de la requête :
`request.setAttribute("une clé", monObjetValeur);`

Pattern Web MVC



- L'interface RequestDispatcher
 - Pour « Passer la main » à une autre servlet / JSP
 - ✦ Du contrôleur à la vue
 - ✦ Du contrôleur principal à un contrôleur délégué (MVC type 2)
 - ✦ D'une partie de la vue à une sous-partie
 - Récupérée par la requête ou par le contexte applicatif
 - ✦ `request.getRequestDispatcher("relativePath");`
 - À partir du chemin relatif de la requête
 - ✦ `getServletContext.getRequestDispatcher("absolutePath");`
 - À partir du chemin racine de l'application
 - ✦ `getServletContext.getNamedDispatcher("servletName");`
 - Cf. nom déclaré dans la configuration (web.xml, annotation)

Pattern Web MVC



- L'interface [RequestDispatcher](#)
 - 2 méthodes exposées

```
void forward(ServletRequest request, ServletResponse response)  
    throws ServletException, java.io.IOException
```

- ✦ “This method allows one servlet to do preliminary processing of a request and another resource to generate the response.”
- ✦ “forward should be called before the response has been committed to the client (before response body output has been flushed). If the response already has been committed, this method throws an `IllegalStateException`.
- ✦ “Uncommitted output in the response buffer is automatically cleared before the forward.”

Pattern Web MVC



- L'interface RequestDispatcher
 - 2 méthodes exposées

```
void include(ServletRequest request, ServletResponse response)  
    throws ServletException, java.io.IOException
```

- ✦ “Includes **the content** of a resource (servlet, JSP page, HTML file) in the response. In essence, this method enables programmatic server-side includes.”
- ✦ “The included servlet cannot change the response status code or set headers; **any attempt to make a change is ignored.**”

Pattern « chain of responsibility »



- Principe
 - Séparation des préoccupations
 - ✦ Une servlet traite le métier de l'application
 - ✦ Des objets dédiés gèrent d'autres aspects non fonctionnels
 - Logs
 - Sécurité
 - ...
 - Interception des objets requête et réponse par des **filtres**
 - Patterns liés
 - ✦ Intercepteur
 - ✦ Décorateur / proxy

Pattern « chain of responsibility »



- L'interface **Filter**

- Représente un handler (filtre) spécifique à une préoccupation
- Prototype

- ✦ Méthodes de gestion du cycle de vie

```
public void init(FilterConfig filterConfig)
                throws ServletException;
public void destroy();
```

- ✦ Méthode de service

```
public void doFilter (ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
                throws IOException, ServletException;
```

- Remarque : les implémentations récentes de la Servlet API fournissent une classe HttpFilter

Pattern « chain of responsibility »



- L'interface **FilterChain**
 - Responsable de la gestion de la chaîne
 - ✦ Instanciée par le conteneur en fonction des paramètres de configuration
 - ✦ Utilisée par les filtres pour transmettre la requête et la réponse à l'élément suivant (filtre, servlet) de la chaîne
 - Ordre des éléments dans la chaîne
 - ✦ Ordre des déclarations dans le fichier de configuration de l'application (web.xml)
 - ✦ Aucune spécification pour la configuration par annotations (API V3)
 - Une seule méthode exposée

```
public void doFilter (ServletRequest request,  
                     ServletResponse response)  
    throws IOException, ServletException;
```

Pattern « chain of responsibility »



- Exemple de filtre (HTTP)

```
import jakarta.servlet.*;
import jakarta.servlet.http.*;

public class IpFilter extends HttpFilter {
    String ipPrefix = "";

    public void init(FilterConfig config) throws ServletException {
        String ipPrefix = config.getInitParameter("ipPrefix");
    }

    public void doFilter(HttpServletRequest request, HttpServletResponse response,
        FilterChain chain) throws IOException, ServletException {
        String ip = request.getRemoteAddr();
        if(ip.startsWith(ipPrefix)) {
            chain.doFilter(request, response);
        } else {
            response.sendRedirect(SC_UNAUTHORIZED);
        }
    }

    public void destroy() {
    }
}
```

Remarque : il est possible de « commencer » par `chain.doFilter(request, response)` pour réaliser des post-traitements

Pattern « Wrapper » (Décorateur)



- **Utilité**
 - Modifier une requête pour la passer à une classe réalisant un traitement particulier
 - ✦ Ex : rajouter des attributs
 - Permettre à une classe de générer une réponse sans altérer la réponse principale
 - ✦ Ex : vérifier le flux de sortie sans l'inclure dans la « vraie » réponse
- **Classes**
 - [ServletRequestWrapper](#)
 - [ServletResponseWrapper](#)

Servlet API V3



- Principe
 - Configuration par annotations dans le code
 - Apparition en 2008 (avec Java 6)
- Exemple de code (servlet)

```
@WebServlet (  
    urlPatterns="/MyApp",  
    name="MyServlet",  
    initParams= @WebInitParam(name="toto", value="titi")  
)  
public class MyServlet extends HttpServlet {  
    public void doGet (HttpServletRequest req,  
        HttpServletResponse res) {  
        .....  
    }  
}
```

Servlet API V3



- Principe
 - Configuration par annotations dans le code
 - Apparition en 2008 (avec Java 6)
- Exemple de code (filtre)

```
@WebFilter(servletNames = "MyServlet")
public class MyFilter extends HttpFilter {
    public void doFilter(HttpServletRequest req,
                        HttpServletResponse res,
                        FilterChain chain)
        throws IOException, ServletException {
        ...
    }
}
```

Servlet API V3



- Ajout de paramètres de configuration
 - Identique pour servlets (et JSP) et filtres
 - ✦ XML
 - `<init-param>`, `<param-name>`, `<param-value>`...
 - ✦ Annotation
 - `@WebInitParam(name="toto", value="Bonjour")`
 - ➔ Ils sont injectés à travers un objet `ServletConfig` / `FilterConfig` qui permet de récupérer
 - ✦ Le nom de servlet / filtre déclaré dans la config
 - ✦ Les paramètres déclarés dans la config
 - ✦ Le contexte applicatif initialisé par le conteneur

Servlet API V4



- Hors du scope de ce cours
 - Prise en compte des protocoles HTTP/2 et HTTP/3
 - ✦ API de push serveur (*vs.* client-serveur)
 - HttpServletMapping
 - ✦ Permet de savoir durant l'exécution l'URL qui a effectivement été appelée pour accéder à une servlet
 - Trailer response header
 - ✦ Pour rajouter des headers dans les contenus chunked
 - ...

Plan du cours



- Introduction
- Programmation côté serveur en Java
- **Application Web en Java**
 - Packaging et contenus
 - Création
 - Déploiement
- Conclusion

Application Web en Java



- **Packaging**

- Un fichier **.war** (Web ARchive)

- Contenu

- ✦ Fichiers Web de l'application (HTML, JSP, JS, CSS...)

- ✦ Répertoire « META-INF » : fichiers de configuration

- MANIFEST.MF : informations sur le zip

- ✦ Répertoire « WEB-INF » : contenu de l'application

- **web.xml** : descripteur de déploiement

- Répertoire « classes » : autres classes de l'application (beans...)

- Répertoire « **webapp** » : ressources Web statiques (HTML, CSS...)

- Répertoire « lib » : bibliothèques supplémentaires (jars)

- Répertoire « src » : sources Java

- Ne contient pas nécessairement

- ✦ Les sources des classes Java

- ✦ Les jars nécessaires à l'exécution de l'application

Création d'une application Web



- Dans un IDE
 - Créer un nouveau projet « de type Web » (dépend de l'IDE)
 - Indiquer au serveur où déployer l'application

- Avec Maven

```
mvn archetype:generate
    -DarchetypeGroupId=org.apache.maven.archetypes
    -DarchetypeArtifactId=maven-archetype-webapp
    -DarchetypeVersion=1.3
```

- Structure générée par Maven

```
|-- pom.xml
|-- src
    |-- main
        |-- webapp
            |-- WEB-INF
            |-- web.xml
            |-- index.jsp
```

/>\ Il manque le répertoire java

Création d'une application Web



- Le pom.xml généré contient des configurations standard de plugins
 - maven-war-plugin
 - ✦ Modifier la configuration de votre projet
 - ✦ Utiliser des buts Maven (goals) spécifiques
 - ➔ Doc : <http://maven.apache.org/plugins/maven-war-plugin/>
- Il est possible d'en rajouter d'autres
 - Tomcat
 - Jetty
 - ...

Descripteur de déploiement (web.xml)



- **Principe**

- Fichier XML décrivant les principales caractéristiques d'une application Web
- Élément racine : `<web-app>`
- Principaux sous-éléments : `<servlet>`, `<servlet-mapping>`, `<filter>`, `<welcome-file-list>`...
- Facultatif depuis la spécification Servlet 3.0
 - ➔ Annotation `@WebServlet`
- Docs :
 - ✦ http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml1.html
 - ✦ <https://developers.google.com/appengine/docs/java/config/webxml?hl=fr>

Descripteur de déploiement (web.xml)



- Exemple 1 : servlet

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <display-name>HelloWorld Application</display-name>
  <description> This is a simple web application with a source code
organization based on the recommendations of the Application
Developer's Guide. </description>
  <servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>examples.Hello</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Source : <http://pubs.vmware.com/vfabric5/index.jsp?topic=/com.vmware.vfabric.tc-server.2.6/getting-started/tutwebapp-web-xml-file.html>

Descripteur de déploiement (web.xml)



- Exemple 2 : JSP, welcome file list

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>HelloJSP</servlet-name>
    <jsp-file>/hello/hello.jsp</jsp-file>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloJSP</servlet-name>
    <url-pattern>/hello/*</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Descripteur de déploiement (web.xml)



- Exemple 3 : filtre

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <filter>
    <filter-name>logger</filter-name>
    <filter-class>monApplication.XMLLogFilter</filter-class>
    <init-param>
      <param-name>logFile</param-name>
      <param-value>log.xml</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>logger</filter-name>
    <url-pattern>*.jsp</url-pattern>
  </filter-mapping>
</web-app>
```


Déploiement



- **Déploiement d'une application Web Java**
 - Consiste à permettre à un conteneur Web d'exécuter l'application
 - Dépôt dans un répertoire *ad hoc* du serveur
 - ✦ Exemple : répertoire « webapps » de Tomcat
 - Lecture des fichiers war au (re)démarrage du serveur
 - ✦ Analyse du fichier war et des paramètres de configuration du descripteur de déploiement
 - ✦ Création du répertoire correspondant dans webapps
 - ✦ Mapping des URL de l'application vers le répertoire créé
 - Autres méthodes de déploiement
 - ✦ <http://tomcat.apache.org/tomcat-4.0-doc/appdev/deployment.html>

Tests



- **Une problématique importante**

- Client particulier (difficile à émuler)

- Distribution des ressources

- Réseau

- ...

- ➔ Différents outils, différentes méthodes

- Références :

<https://softwareengineering.stackexchange.com/questions/175856/how-to-test-java-web-applications>

<https://www.guru99.com/web-application-testing.html>

<https://jwebunit.github.io/jwebunit/>

<https://spring.io/guides/gs/testing-web/>

Plan du cours



- Introduction
- Programmation côté serveur en Java
- Application Web en Java
- **Conclusion**

Conclusion



- Dans ce cours
 - Aperçu des technos de programmation côté serveur en Java
 - ✦ Servlets, JSP, JSTL, taglibs
 - ➔ Ce ne sont pas les seules (JSF, EJB...)
 - À mixer avec les technos côté client
 - ✦ CSS, JavaScript, XML (SVG, transformation...)
 - ✦ AJAX...
 - Ne dispensent pas de réfléchir à la structuration de l'application (au contraire !)
 - ✦ Utilisation de frameworks
 - Permettent de mettre en place facilement des services complexes
 - Assurent (?) un minimum de rigueur dans le développement

Bibliographie utilisée pour ce cours



- Développement d'applications Web en Java :
http://fr.wikipedia.org/wiki/JavaServer_Pages
- Java Servlet API :
<https://tomcat.apache.org/tomcat-10.0-doc/servletapi/>
- Java EE API :
<http://java.sun.com/javae/5/docs/api/>
- JSP :
http://fr.wikipedia.org/wiki/JavaServer_Pages
<http://www.commentcamarche.net/contents/jsp/jspcarac.php3>
- JSTL et EL :
<http://java.sun.com/developer/technicalArticles/javaserverpages/faster/>
- EL :
<http://adiguba.developpez.com/tutoriels/j2ee/jsp/el/>
<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro7.html>
- Taglibs :
<http://adiguba.developpez.com/tutoriels/j2ee/jsp/taglib/>
- API V3 :
<https://today.java.net/pub/a/today/2008/10/14/introduction-to-servlet-3.html>