



## Web Tier : déploiement de servlets

# Plan

- 1 Introduction
- 2 Servlet : Principe de fonctionnement
- 3 Création et développement sur un serveur JEE
- 4 Quelques méthodes de l'API des servlets
- 5 Utilisation des cookies et l'objet HttpSession

# Applications java web

## composants web

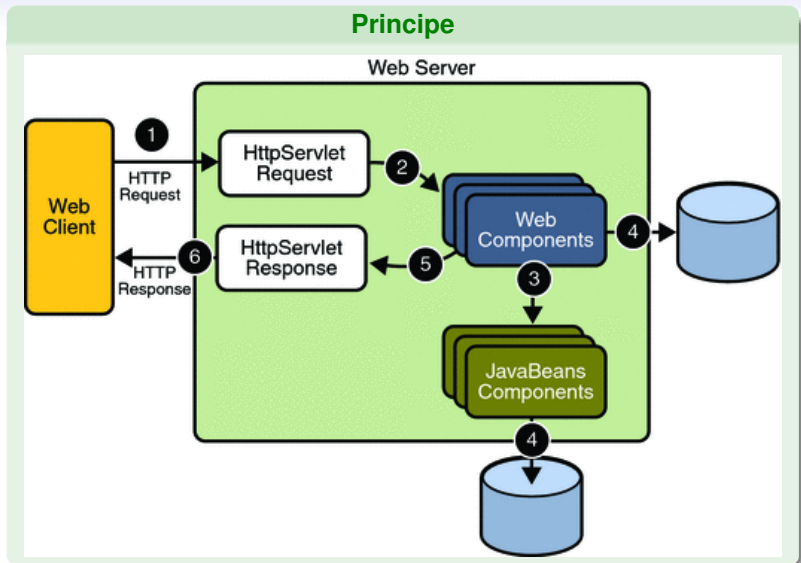
- la plate-forme java 2 fournit en standard les éléments de base permettant la réalisation de pages web dynamiques.
- Ces composants sont soit des Java servlets, des JSP pages, ou des web services.

# Applications java web

## Principe

- 1 Le client envoie une requête HTTP au web server.
- 2 Un web server qui implémente la technologie JEE (Servlets et JSP) convertit la requête dans un objet **HttpServletRequest**.
- 3 Cet objet est délivré à un **web component**, qui peut interagir avec un *JavaBeans component* ou une database pour générer un contenu dynamique.
- 4 Le **web component** peut ensuite générer un objet **HttpServletResponse** ou passer la requête à un autre **web component**.
- 5 Quoi qu'il en soit, un **web component** finira par générer un objet **HttpServletResponse**.
- 6 Cet objet sera converti par le web server pour créer une réponse HTTP qui sera transmise au client.

# Applications java web



# Servlets et JSP en bref

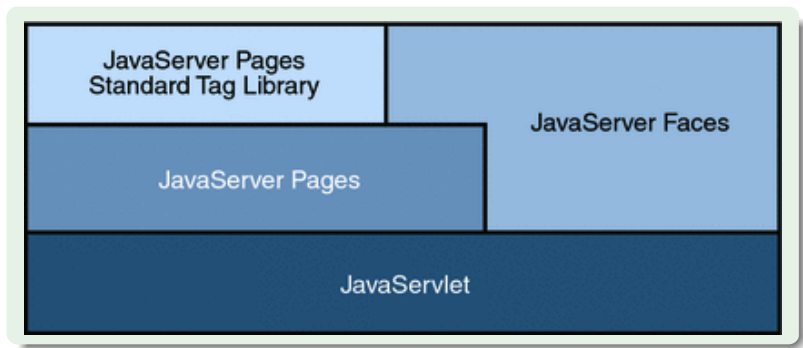
## Servlets

- Programmes java générant dynamiquement des pages web en traitant la requête et en construisant la réponse http. (programmation «service orienté»)

## JavaServer Pages JSP

- document texte qui s'exécute comme une servlet, mais permet une approche plus naturelle pour créer du contenu statique. (pour une programmation plus adaptée pour la génération de fichiers à base de tags (HTML, Scalable Vector Graphics (SVG), Wireless Markup Language (WML), XML, ...))

# Technologie java web



# Cycle de vie d'une application java web

## Cycle de vie d'une application java web

- 1 Développer le code du composant web
- 2 Développer le *web application deployment descriptor*.
- 3 Compiler le composant web et les classes annexes qu'il utilise.
- 4 Optionnellement, packager l'application dans une unité déployable.
- 5 Déployer l'application dans un conteneur web.
- 6 Tester l'URL qui référence l'application web.

# Une Servlet hérite de *javax.servlet.http.HttpServlet*

## HttpServlet est une classe abstraite définissant :

- **doGet** : pour les HTTP GET requests
- **doPost**, pour les HTTP POST requests
- *doPut*, pour les HTTP PUT requests
- *doDelete*, pour les HTTP DELETE requests
- *init* et *destroy* : gestion du cycle de vie de la servlet (e.g. des ressources)
- *getServletInfo*, informations à propos de la Servlet

## Hello.java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Hello extends HttpServlet {

    protected void doGet(HttpServletRequest request ,
        HttpServletResponse response)
        throws ServletException , IOException {
        response.setContentType (" text/html ");
        PrintWriter out = response.getWriter ();
        out.println ("<HTML><HEAD><TITLE>Hello Client!</TITLE>"+
            "</HEAD><BODY>Hello Client!</BODY></HTML>");
        out.close ();
    }
}
```

## Hello dans le détail :

### Hello.java : import nécessaires

```
//Exception IO
import java.io.IOException;

// pour le flux d'écriture
import java.io.PrintWriter;

//Exception liée au cycle de vie de la Servlet
import javax.servlet.ServletException;

//classes obligatoires
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

## Hello.java : héritage et méthode doGet

```
// La servlet herite de la classe HttpServlet
public class Hello extends HttpServlet {

    // Cette Servlet peut être appelée par une requête HTTP GET
    protected void doGet(
        HttpServletRequest request, // contient les infos de la requête
        HttpServletResponse response) // contient la réponse à construire
        throws ServletException, IOException { // exceptions
```

# Hello dans le détail

## Avant toute chose : spécification du type de la réponse

```
response.setContentType("text/html");
```

## Récupération du flux d'écriture

```
PrintWriter out = response.getWriter();
```

## Écriture de la réponse html

```
out.println("<HTML><HEAD><TITLE>Hello Client!</TITLE>"+  
" </HEAD><BODY>Hello Client!</BODY></HTML>");
```

```
// optionnel mais utile s'il y a vait des traitement additionnels  
out.close();
```

# Déploiement de fichiers html simples

## Avec Eclipse

- Créer un nouveau projet de type de type "*dynamic web project*".
- Créer un nouveau fichier html (contenu indifférent) dans le répertoire WebContent : c'est lui qui contient les pages html statiques et le JSP.
- Déployer le fichier html sur un serveur JEE avec Eclipse (*run on server*).

# Déploiement d'une première servlet

## Dans le même projet

- Créer une nouvelle Servlet (First.java) : elle affichera une page simple avec une image. Utiliser les fichiers suivants : [▶ First.java](#) [▶ duke.waving.gif](#)
- Déployer la servlet sur un serveur JEE.
- Modifier la place de la servlet sur le serveur en modifiant le *web application deployment descriptor* et tester.

# Déploiement d'une servlet utilisant des fichiers annexes

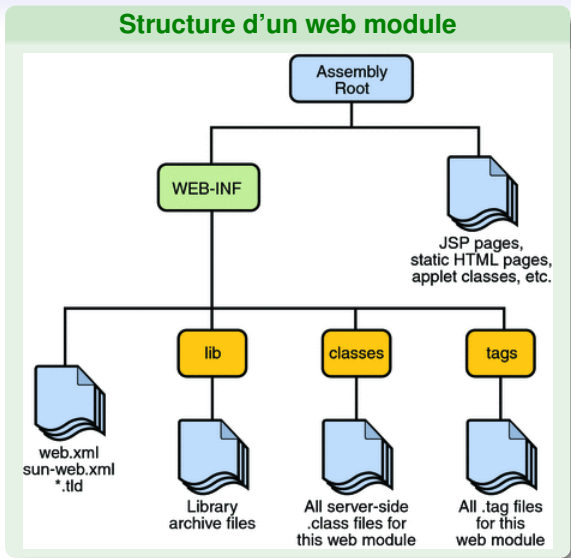
## Dans votre projet

- 1. Rajouter les classes suivantes dans votre projet.
- 2. Déployer la servlet correspondante sur un serveur JEE avec Eclipse.
- 3. Déployer la servlet correspondante sur le serveur à l'aide d'un fichier war.

## code source

- ▶ TimePage.java
- ▶ GiveMeTime.java

# Mise en place du web module



# Mise en place d'un web module

## Structure d'un web module

- **web.xml** : *The web application deployment descriptor*
- **Tag library descriptor files**
- **classes** : un répertoire contenant les classes utilisées côté serveur : servlets, classes annexes, composants JavaBeans.
- **tags** : Un répertoire contenant des fichiers tag : des implémentations des librairies de tag.
- **lib** : Un répertoire contenant des fichiers JAR nécessaires aux classes côté serveur.

## Si le module web ne contient que des pages statiques ou des jsp :

- Alors l'application deployment descriptor n'est pas obligatoire (web-xml)

# Déploiement d'un web module

## première solution : tel quel

- Copier la structure du répertoire du module web tel quel sur le serveur

## deuxième solution : sous forme d'un fichier archive

- Créer une archive web, un fichier WAR, à déployer sur le serveur.
- Pour que cela soit possible le fichier doit contenir un *runtime deployment descriptor* : *sun-web.xml*

# runtime deployment descriptor

- Ce fichier xml décrit des informations sur le module (notamment le **context root** : racine de l'application web par rapport à l'adresse http du serveur).

## sun-web.xml

?? xml	version="1.0" encoding="UTF-8"
DOCTYPE	sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Application
☑ sun-web-app	(context-root?, security-role-mapping*, servlet*, idempotent-url-
@ error-url	
☑ context-root	/test
▶ ☑ class-loader	(property*)
▶ ☑ jsp-config	(property*)

# Déploiement d'un module web sur un serveur

## Plusieurs solutions :

- Mettre le fichier war dans le répertoire *domain-dir/autodeploy/* du serveur.
- Utiliser la console d'administration (asadmin graphique) du serveur port 4848.
- En utilisant un ide.

# Exemple : administration d'un serveur

## avec GlassFish v2.0

The screenshot shows the Sun Java System Application Server Admin Console. At the top, it displays the user 'admin', domain 'domain1', and server 'localhost'. The main title is 'Sun Java™ System Application Server Admin Console'. On the left, a tree view shows the navigation structure: 'Tâches courantes', 'Enregistrement', 'Application Server', 'Applications', 'Applications Entreprise', 'Applications Web' (selected), 'Modules EJB', 'Modules connecteur', 'Modules de cycle de vie', 'Modules client d'application', 'Services Web', 'JBI', and 'Assemblages de services'. The main content area is titled 'Applications > Applications Web' and features a section 'Redéployer les applications/modules d'entreprise'. Below this, there is a description: 'Permet d'indiquer l'emplacement d'une application à redéployer. Le fichier de package sélectionné doit correspondre au type...'. The form includes fields for 'Nom' (tests) and 'Type' (Application Web (.war)). Under 'Emplacement', there are two radio buttons: 'Fichier de package à télécharger sur le serveur' (selected) and 'Fichier de package local ou répertoire accessible à partir du serveur d'applications'. The first option has a 'Browse...' button, and the second has a 'Parcourir les fichiers' button.

## Guide d'administration du serveur sun



► Sun Java System Application Server 9.1 Administration Guide

# Servlets mapping

## Dans le fichier web.xml (le web application deployment descriptor)

▷ <input type="checkbox"/>	servlet	(((description*, display-name*, icon*)), servlet-r
▷ <input type="checkbox"/>	servlet-mapping	(servlet-name, url-pattern+)
▽ <input type="checkbox"/>	servlet	(((description*, display-name*, icon*)), servlet-r
	<input type="checkbox"/> description	
	<input type="checkbox"/> display-name	GreetingServlet
	<input type="checkbox"/> servlet-name	GreetingServlet
	<input type="checkbox"/> servlet-class	servlets.GreetingServlet
▽ <input type="checkbox"/>	servlet-mapping	(servlet-name, url-pattern+)
	<input type="checkbox"/> servlet-name	GreetingServlet
	<input type="checkbox"/> url-pattern	/hello
▷ <input type="checkbox"/>	servlet	(((description*, display-name*, icon*)), servlet-r
▷ <input type="checkbox"/>	servlet-mapping	(servlet-name, url-pattern+)
▷ <input type="checkbox"/>	servlet	(((description*, display-name*, icon*)), servlet-r

## Paramètres contenus dans une url

### La méthode `getParameter` de l'objet `HttpServletRequest`

- Il permet de récupérer la valeur d'un paramètre contenu dans une url.
- par dans exemple :  
*http://localhost:8080/hello?username=toto&firstName=bob* obtenu avec un formulaire de type get
- `→String username = request.getParameter("username");`

# L'objet RequestDispatcher

## javax.servlet.RequestDispatcher

- Il permet d'utiliser d'autres servlets pour la construction de la réponse.
- sur le principe include : une portion de la réponse
- ou forward : une autre servlet s'occupe de la réponse

## Récupération de l'objet en fonction de l'url souhaitée

- RequestDispatcher dispatcher =  
getServletContext().getRequestDispatcher("/response");

## include ou forward. Attention : l'url de la page ne change pas

- dispatcher.include(request, response);
- dispatcher.forward(request, response);

# Utilisation des cookies

## Les cookies

- L'envoi d'un cookie vers le navigateur du client se fait grâce à la méthode `addCookie()` de l'objet `HttpServletResponse` :  
`void AddCookie(Cookie cookie)`
- Il est nécessaire de créer le cookie avant tout envoi de données :  
`Cookie MonCookie = new Cookie("nom", "valeur");`
- Pour récupérer les cookies provenant de la requête du client, il suffit d'utiliser la méthode `getCookies()` de l'objet `HttpServletRequest` :  
`Cookie[] getCookies()`
- La durée de vie d'un cookie est réglable ; par défaut un cookie disparaît à la fermeture du navigateur, mais on peut le rendre persistant :
- `int getMaxAge()` : Retourne la durée de validité du cookie (en secondes)
- `void setMaxAge(int duree)` : Définit la durée de validité du cookie (en secondes)

# L'objet `javax.servlet.http.HttpSession`

## Principe

- Il permet de stocker des données côté serveur pendant une durée déterminée :
  - ▶ API JEE
- Il peut être créé à partir d'un objet `HttpServletRequest` en utilisant la méthode `getSession` (cf. API)
- Voici des exemples de code basé sur ce principe :
  - ▶ `BookStoreServlet.java`
  - ▶ `ShoppingCart.java`

# Bibliographie

## Références

- Ce cours reprend largement le tutoriel JEE proposé par Sun :
- [▶ JEE tutoriel](#)