Java EE Servlets et JSP

Initiation + Approfondissement

Arnaud Delafont

22/07/2019

Objectifs

- Acquérir une bonne connaissance de l'architecture JEE et en particulier des frameworks destinés à la conception des applications pour le Web
- Être capable de concevoir, d'implémenter et de mettre au point des servlets et des JSP
- Mettre en application le Design Pattern MVC
- Être capable de déployer une application WAR

Plan

- Architectures Web
- Plate-forme JEE
- Servlets
- Java Server Pages
- Modèle MVC
- Servlets (suite)
- Taglibs
- Concepts avancés

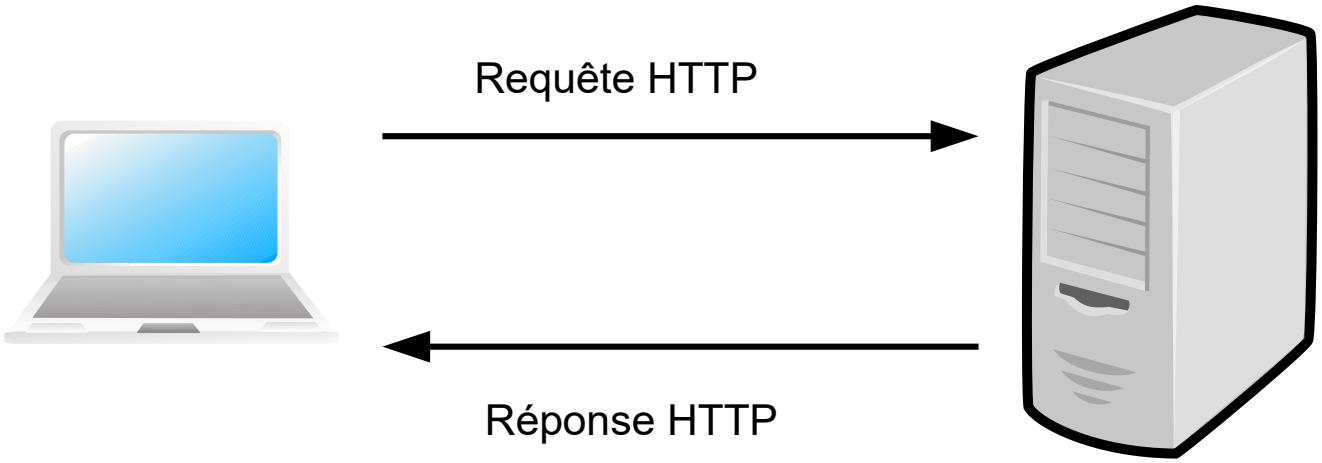
Architectures Web

Protocole HTTP

- Le protocole HTTP(Hypertext Transfer Protocol) a été créé au Cern en 1990 avec les adresses Web et le langage HTML pour créer le World Wide Web
- HTTP est :
  - conçu pour être **simple** et **lisible** par un humain (texte)
  - **extensible**: les en-têtes HTTP permettent d'étendre facilement le protocole
  - **sans état** : il n'y a pas de lien entre deux requêtes qui sont effectuées successivement sur la même connexion.
    les cookies HTTP permettent l'utilisation de sessions avec des états.

Principe

- HTTP est basé sur des couple requête/réponse



Une réponse ne peut exister que si une requête HTTP à été envoyé au serveur

Le protocole **tcp** est utilisé pour véhiculé la requête et la réponse http. Une connexion via tcp est établie pour chaque couple requête/réponse. Depuis http 1.1 plusieurs couple requête / réponse pour une même connexion tcp

Requête HTTP

- Une requête HTTP est un ensemble de lignes envoyé au serveur par le navigateur. Elle contient
  - une **ligne de requête**, elle est composé
    - du type de document demandé
    - du type de requête
    - de la version du protocole
- Les **en-tête** de la requête (obligatoire)
  ↳ fournissent des informations supplémentaire sur la requête et / ou le client
- Le **corps** de la requête (optionnelle)
  ↳ un ensemble de lignes optionnelles permettant par exemple l'envoie de données au serveur par un formulaire

Les types de requêtes

- **GET** → permet de demande une ressource sur le serveur
- **POST** → utilisé lorsqu'une requête modifie la ressource
- **HEAD** → permet de ne demandé que des informations sur le ressource, sans demander la ressource elle-même

Depuis HTTP 1.1

- **PUT** → permet d'ajouter une ressource sur le serveur
- **DELETE** → permet de supprimer une ressource sur le serveur
- **TRACE** → permet de demander au serveur de retourner dans le corps de la réponse un copie de la requête (phase de test)
- **OPTIONS** → obtenir des informations sur les options utilisables pour obtenir une ressource

Les en-tête de requête

- **Accept** : indique au serveur quels type de données sont accepté (liste de type MIME)
- **Accept-Charset** : indique les préférences du navigateur pour les jeux de caractères utilisables ISO-8859,utf-8
- **Accept-Language** : indique au serveur quelle langue il souhaite obtenir sur la ressource demandée
- **Connection** : (http1.1) détermine comment vont se comporter les connexions tcp , keep-alive si le navigateur souhaite conserver la connexion ouverte
- **User-Agent** : permet d'identifié le type de navigateur
- ...

Réponse HTTP

- Les réponses contiennent
  - une ligne de **status**. Elle précise
    - la version HTTP du serveur
    - le code-réponses : 200,404,500...
    - Le texte associé au code OK, FORBIDDEN, NOT FOUND, INTERNAL ERROR
  - les **en-têtes** de la réponse
  - le **corps de le réponse** contient le contenu du fichier (page HTML)
- Le code de réponse : **1xx** → Information, **2xx** → succès, **3xx** → redirection, **4xx** → erreur lié au client, **5xx** → erreur lié au serveur

Les en-têtes de réponse

- **location** : indique le nouvel emplacement où se trouve la ressource (dans les réponses http de redirection 3XX)
- **server** : contient les information sur le type du serveur qui a généré la réponse
- **content-language** : indique la langue du contenu du corps de laréponse Http
- **content-type** : indique type MIME du document contenu dans la réponse
- **date** : la date et l'heure de génération de la réponse HTTP
- …

Les URLs

- **Syntaxe**:
  protocole://identifiant du serveur:numéro de port/ressource?paramètres#signet
- **protocole**: protocole utilisé pour accéder à la ressource. (HTTP, ftp...)
- **identifiant du serveur**: identification de serveur sur le réseau Il doit être transformé en adresse IP (serveur DNS) pour que le protocole TCP puisse effectué la connection avec le serveur.
- **numéro de port**: numéro du port TCP vers laquelle doit être établie la connexion, Il sert a identifier une application sur le serveur ex: 80 pour http, 21 pour ftp)
- **ressource**: Chemin absolue de la ressource. Il commence par / et chaque élément du chemin est séparé par /
- **paramètre**: Données supplémentaires optionnelles, transmise au service lors de la demande à la ressource :
  ? : caractère de séparation obligatoire pour indiquer que des paramètres suivent.

Les URLs

- Les paramètres sont sous la forme du couple **nomDeParamètre=valeurduParamètre**.

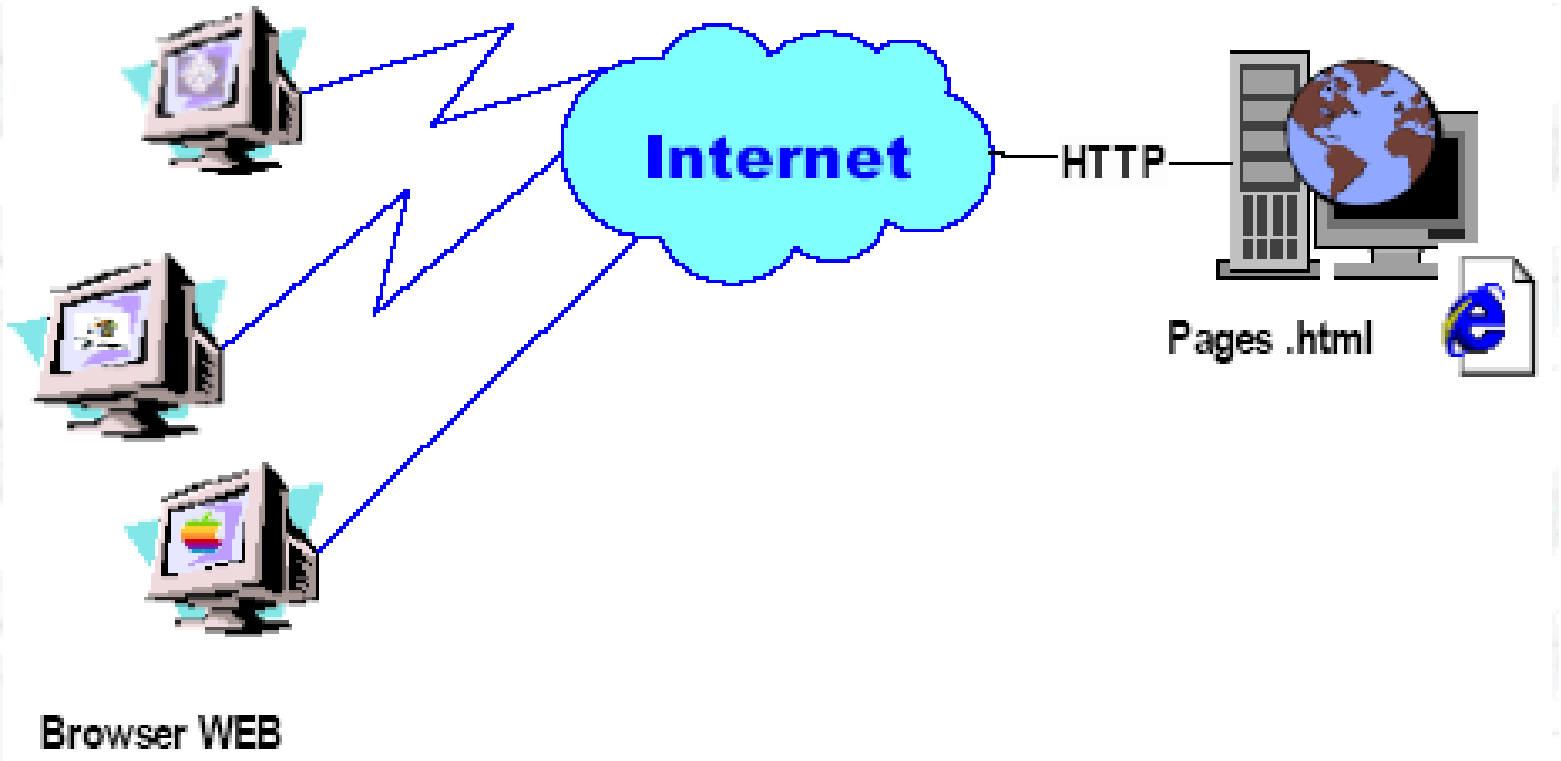
  S'il y a plusieurs paramètres, ils sont séparé par **&** ex: q=req&q2=req2

- **signet** : identificateur du signet ou de la balise. Il s'agit d'un emplacement à l'intérieur de la page web retournée par le service, cette donnée sera traitée par le navigateur web.

  http://www.exemple.com:8888/chemin/d/acc%C3%A8s.php?q=req&q2=req2#signet

- **Encodage d'url**:
  certains caractères ont une signification spécifique dans l'URL (/ ? &). S'ils sont utilisés ils doivent être remplacé par %HH ou HH est le code ASCII en Hexadécimal qui correspond au caractère
  → %20, " → %22, % → %25, & → %26, + →%2B, .→ %2E, / →%2F, ? → %3F, ' → %60

HTML statique



HTML dynamique

- **Exemple : HTTP + CGI (Common Gateway Interface)**

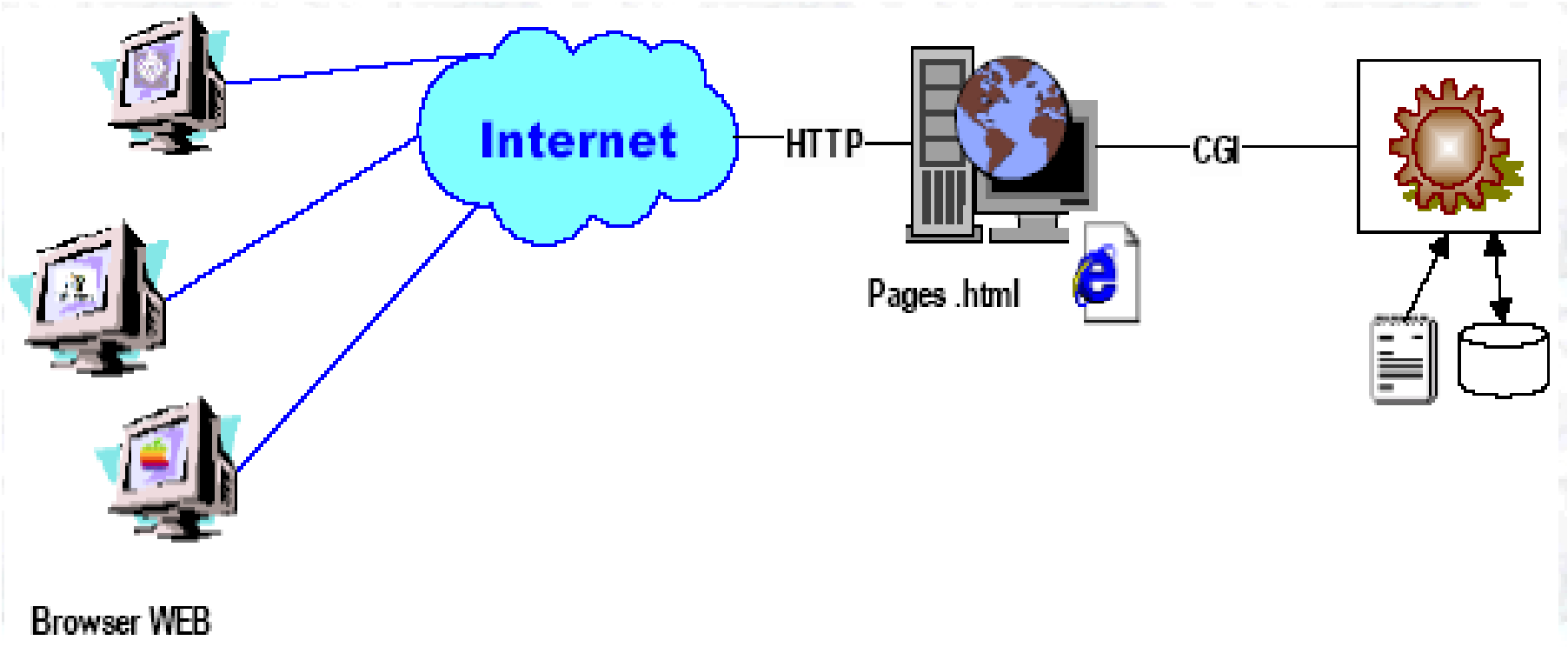


Plate-forme JEE

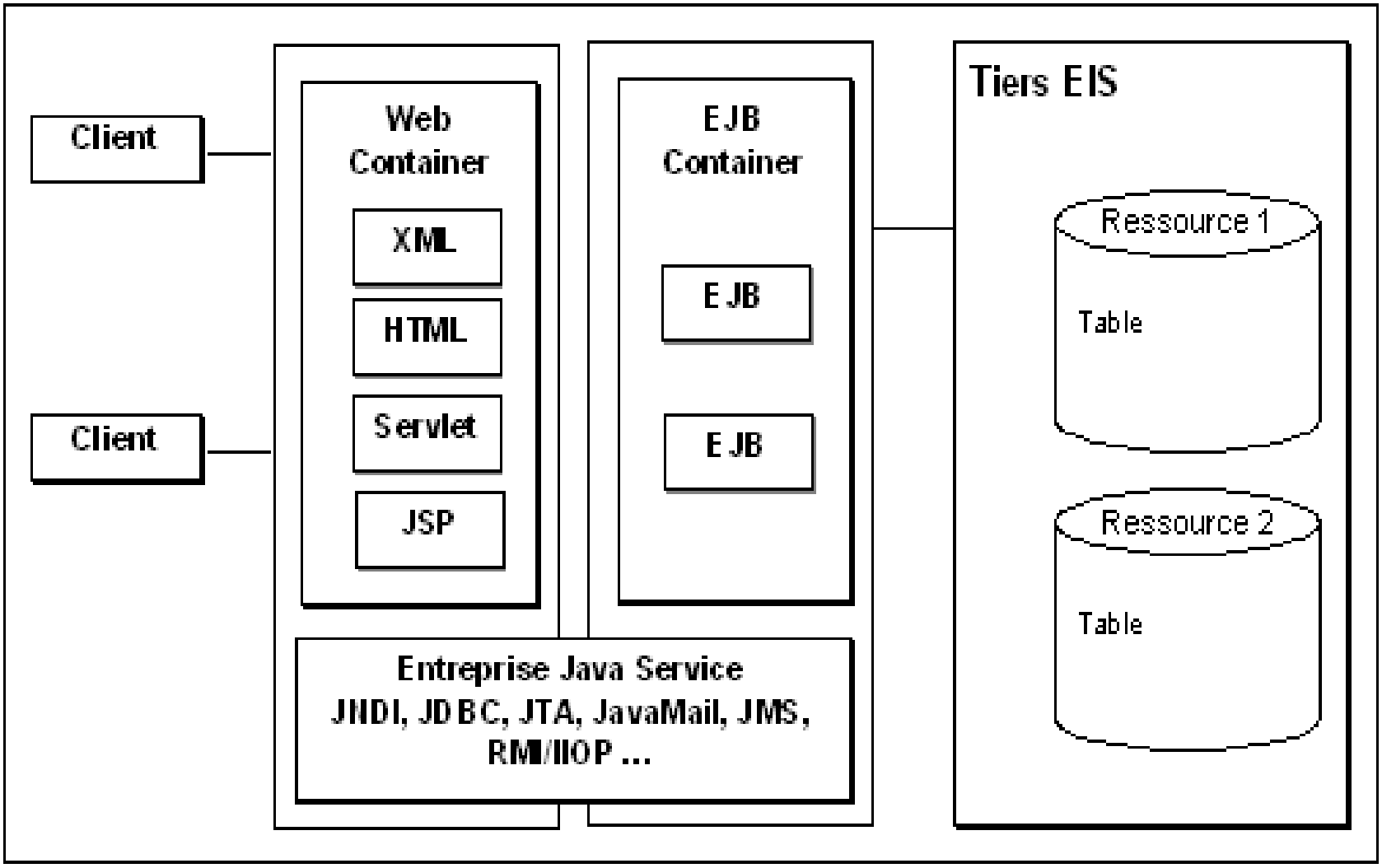
Plate-forme JEE

- **Qu'est-ce que JEE ?**
  Java Entreprise Edition
- Un modèle de programmation basé sur des composants, pour fournir un Standard simple et unifié pour applications distribuées
- JEE est un ensemble de spécifications (et pas un produit) qui, en respectant une architecture multi-tiers, va décrire :
  - L'infrastructure de gestion des applications
  - Les API des services utilisées pour concevoir ces applications

Serveur d'applications

- JEE définit les rôles et les interfaces pour les applications ainsi que l'environnement dans lequel elles seront exécutées
- Ces recommandations permettent à des entreprise de développer des serveurs d'applications conforme aux spécifications
- Serveur d'applications
  - implémente les API JEE
  - héberge des composants applicatifs
  - fournit des services à ces composants au travers d'un conteneur (un environnement d'exécution chargé de gérer des composants applicatif et leur donner accès aux API JEE)

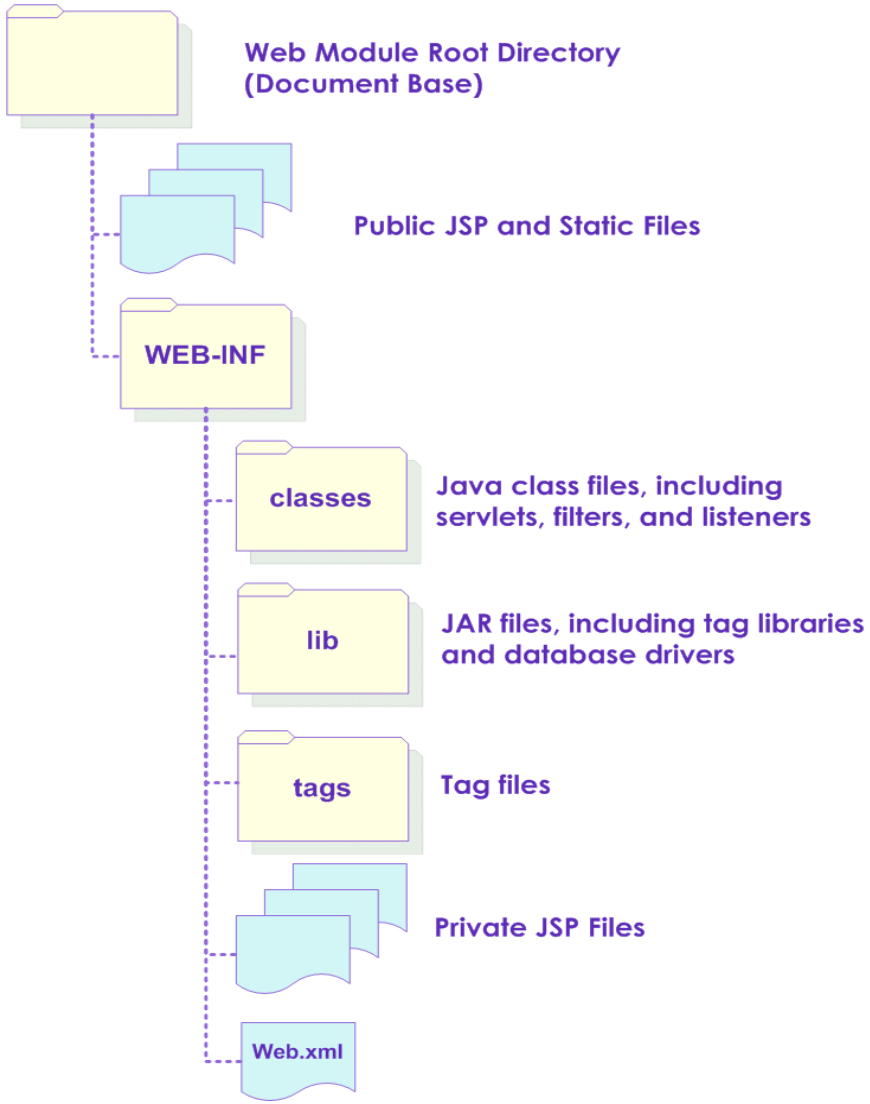
Architecture JEE – Conteneurs



Environnement de developpement

- **Développement, déploiement** :
  - Eclipse JEE: Standard et complet
  - Alternatives : Oracle Netbeans, IntelliJ IDEA...
- **Exécution**
  - Apache Tomcat : Rapide et simple
  - Altenatives : RedHat JBoss App. Server, OW2 JonAS, Oracle Glassfish, IBM Websphere, Oracle WebLogic…
- **Test**
  - Firefox, Chrome... (+ modules)
    Standard et modulaire

Structure d'un module web



Configuration d'eclipse

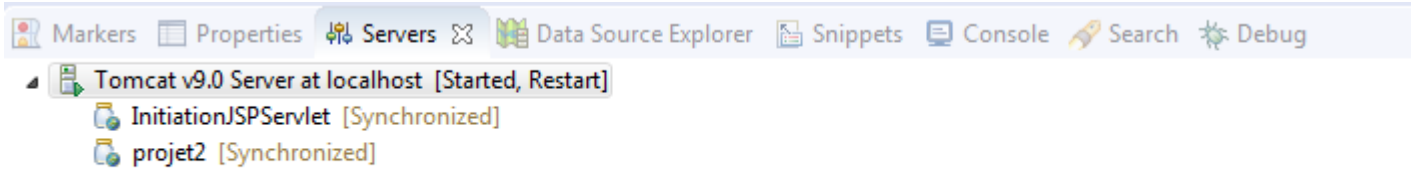
- Menu → Window → Préférences
  - Filtre sur **encoding**
    - CSS Files, HTML Files, JSP Files : sélectionner UTF-8
  - Filtre sur **JRE**
    - Ajouter le répertoire du JDK et le sélectionner
  - Filtre sur **Server**
    - Ajouter le server Tomcat (ou autre)
- Menu → Window → Show View
  - Ajouter la vue package explorer

Création du projet avec Maven

- Menu → File → New → Maven Project
  - Clic droit sur le projet → Propriétés
    - Filtre sur **Java Build Path**
      - Ajouter une librairie / Server Runtime / votre version d'Apache Tomcat (ou autre)
  - Filtre sur **Project Facets**
    - Sélectionner la version de Java correspondant à votre JDK
- Récupérer un web.xml au format 3.1 (penser à modifier le project-name)

Configuration d'eclipse

- Onglet Servers



Ajouter le serveur d'application et associer le projet

Servlets

Qu'est-ce qu'une servlet ?

- Une classe java qui s'exécute coté serveur en tant qu'extension du serveur d'application
- Elle reçoit une requête du client, la traite et renvoie le résultat.

Avantages :

- Efficacité
- Pratique (cookies, session, portabilité, ...)
- Extensible et flexible
- Puissant et robuste (langage Java)

Inconvénients :

- Lourdeur dans une conception graphique
- Inadéquat pour la génération HTML et du code JavaScript

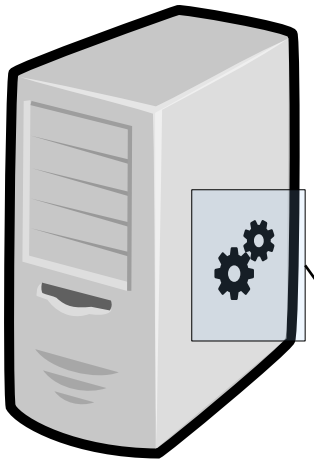
Appel d'une Servlet

Client (navigateur web)

Serveur d'application

1. Requête HTTP (texte)

6. Réponse HTTP (texte)



2. Création des objets requête et réponse

3.appel de service(requête, réponse)

4.appel de doXXX(requête, réponse)

5. Objet réponse

API servlet

L'API Servlet se répartit en deux packages :

- **javax.servlet** : contient les interface et les classes des servlets génériques indépendantes d'un protocole
- **javax.servlet.http** : contient les classes des servlets qui reposent sur le protocole http.

Interface	Classe	Description
Servlet	GenericServlet, HttpServlet	Définie une servlet
ServletRequest	HttpServletRequest	Définie une requête
ServletResponse	HttpServletResponse	Définie une réponse
Filter	FilterChain, FilterConfig	Définie un filtre
ServletContext		Obtenir des informations sur le contexte d'exécution de la servlet (l'application web)
ServletConfig		Définie l'ensemble de la configuration d'une servlet (paramètres d'initialisation...)
HttpSession		Définie une session HTTP
	Cookie	Classe représentant un cookie

Interface servlet

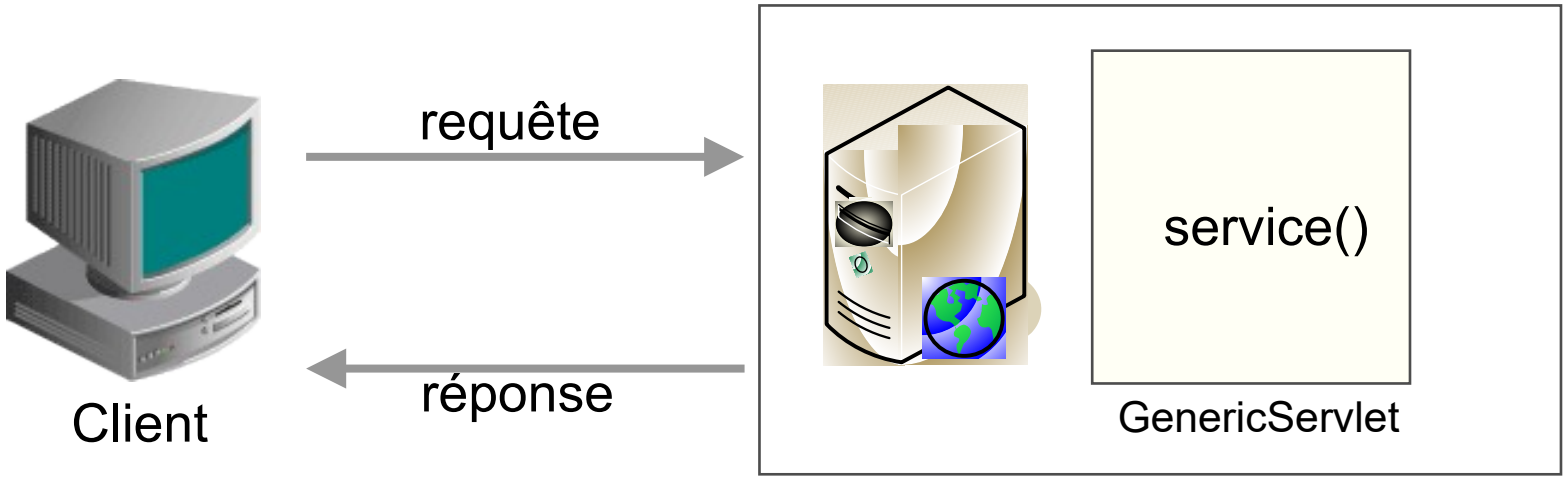
Une servlet est une classe Java qui implémente l'interface **javax.servlet.Servlet**

Les méthodes de l'interface permettent au conteneur web de dialoguer avec la servlet

void **init**(ServletConfig conf)	Appelée une seule fois après l'instanciation de la servlet. Pour l'initialisation de la servlet
void **service** (ServletRequest req, ServletResponse res)	Exécutée par le conteneur lorsque la servlet est sollicitée Chaque requête du client déclenche une seule exécution de cette méthode
void **destroy**()	Appelée lors de la destruction de la servlet.
ServletConfig **getServletConfig**()	Renvoie l'objet ServletConfig passé à la méthode init
String **getServletInfo**()	Renvoie des informations sur la servlet

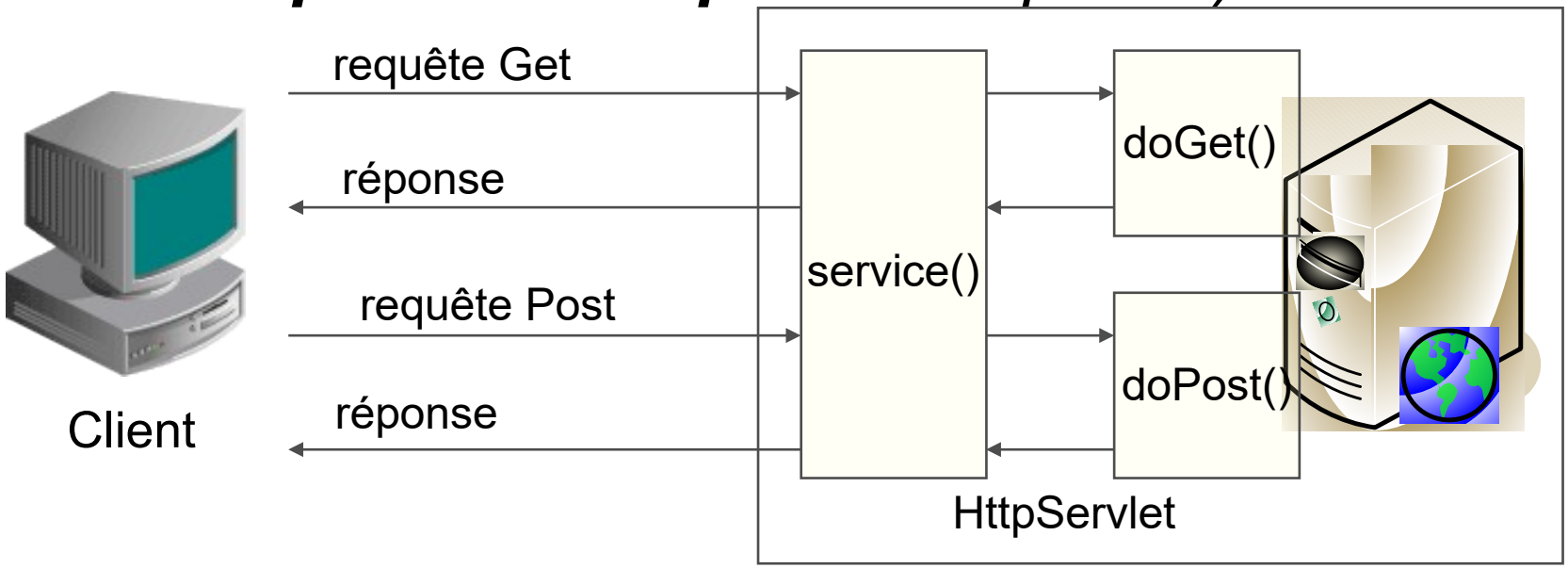
Generic servlet

- Hériter de la classe abstraite **javax.servlet.GenericServlet** et surcharge de la méthode :
  - *void service (**ServletRequest** request, **ServletResponse** response)*

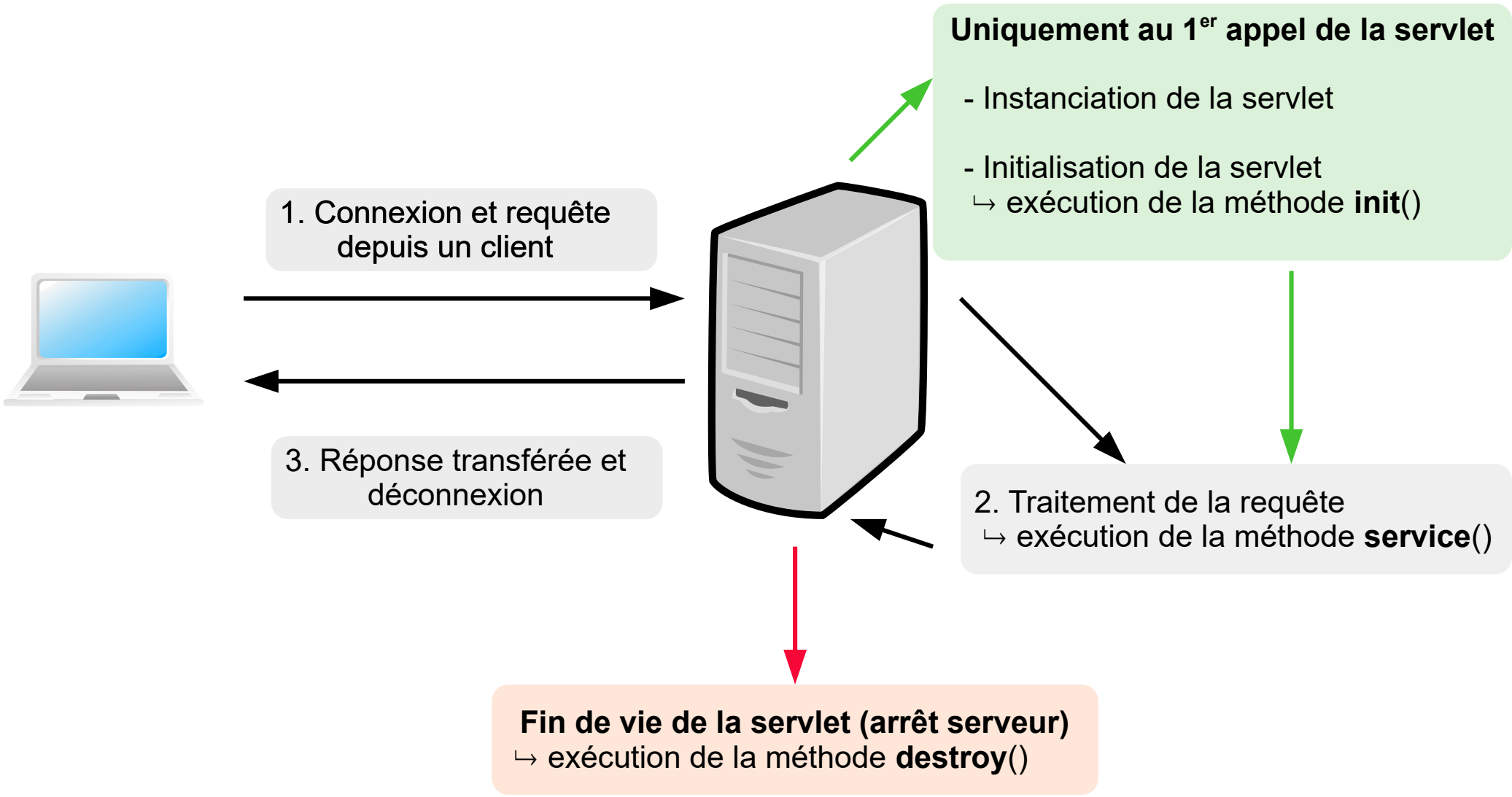


HttpServlet

- Hériter de **javax.servlet.http.HttpServlet** et surcharge des méthodes :
  - *void doGet(**HttpServletRequest** request, **HttpServletResponse** response)*
  - *void doPost(**HttpServletRequest** request, **HttpServletResponse** response)*



Cycle de vie d'une servlet



Déclaration de la servlet

- Dans le descripteur de déploiement : web.xml

```
<servlet>
   <description>Ma première servlet</description>          ← Commentaire sur la servlet
   <servlet-name>FirstServlet</servlet-name>                ← Nom de la servlet
   <servlet-class>tp.controlers.FirstServlet</servlet-class> ← Classe java de la servlet
</servlet>
```

établir le lien entre une servlet et une URL

```
<servlet-mapping>
   <servlet-name>FirstServlet</servlet-name>     ← Nom de la servlet
   <url-pattern>/firstservlet</url-pattern>      ← URL pour contacter la servlet
</servlet-mapping>
```

Chargement de la servlet au démarrage du serveur

```
<servlet>
   ...
   <load-on-startup>1<load-on-startup>   ← Le nombre indique l'ordre de chargement
</servlet>
```

Déclaration de la servlet

- Avec une annotation dans le code source (depuis servlet 3.0)

```
@WebServlet("/firstservlet")
public class FirstServlet extends HttpServlet {
}
```

```
@WebServlet(name="FirstServlet",
                    urlPatterns="/firstservlet" )
public class FirstServlet extends HttpServlet {
}
```

Chargement de la servlet au démarrage du serveur

```
@WebServlet(name="FirstServlet",
                    urlPatterns="/firstservlet"
                    loadOnStartup=0 )          ← Le nombre indique l'ordre de chargement
public class FirstServlet extends HttpServlet {
}
```

Paramètre d'initialisation

- Dans le descripteur de déploiment (web.xml)

```
...
<servlet>
  <servlet-name>...</servlet-name>
  <servlet-class>...</servlet-class>
  <init-param>
      <description>...</description>          ← commentaire sur le paramètre
      <param-name>annee</param-name>          ← nom du paramètre
      <param-value>2012</param-value>         ← valeur du paramètre
  </init-param>
  ...
</servlet>
```

- Avec les annotations

```
@WebServlet(name="FirstServlet",
                urlPatterns="/firstservlet"
                initParams={@webInitParam(name="annee", value="2012")}
}
```

Paramètre d'initialisation

- On récupère les paramètres dans la méthode **init** avec la méthode **getInitParameter(String name)**

  retourne null, si le paramètre n'existe pas

```
public void init() throws ServletException{
    super.init() ;
    String a = getServletConfig().getInitParameter("annee");
    ...
}
```

Réponse à une requête

- À la réception d'une requête la servlet la traite et crée une réponse

- 2 objets sont utilisés:
  - Interface : **javax.servlet.ServletResponse**
  - Interface : **javax.servlet.ServletRequest**

Écriture de la réponse

- L'interface **javax.servlet.ServletResponse** est utilisée pour l'écriture de la réponse
- Définir le statut de la réponse

setStatus(int sc) → Fixer la valeur du statut de la réponse

1xx→ Informatif
2xx→ Succés
3xx→ Redirection
4xx→ Erreur coté client
5xx→ Erreur coté serveur

Par défaut, le code de statut est 200
SC_xxx → constante de l'interface représentant les codes des status
ex : SC_INTERNAL_SERVER_ERROR pour 500

Écriture de la réponse

sendError(int sc)	signaler une erreur ou un problème dans le traitement de la requête.
sendError(int sc,String msg)	Idem avec une personnalisation du message

Écriture de la réponse

- Ajouter des en-têtes
  Utiliser pour ajouter à la réponse htttp des informations supplémentaires

setContentType(String type)	Fixer le type MIME du contenu du corps de la requête
setCharacterEncoding(String charset)	Fixer le codage des caractères du corp de la requete
setLocale(Locale loc)	contrôler la langue dans laquelle la réponse est envoyée.
setHeader(String name,String value)	Ajouter une en-tête de nom *name* et la valeur *value*. Elle est écrasé si elle existe déjà
addHeader(String name,String value)	Idem sauf si l'en-tête existe déjà, la valeur lui est ajouté
sendRedirect(String location)	Rediriger de la requête vers une autre URL. Le type de réponse est obligatoirement 302 (redirection temporaire)

Écriture de la réponse

- La modification des en-têtes de réponse doit se faire avant l'écriture d'information dans le corps de la requête
- Écrire le corps de la réponse
  Cela consiste à envoyer les informations sur le flux de sortie
  - PrintWriter getWriter()
    ↳ écrire du texte dans le corps de la reponse HTTP
  - ServletOutputStream getOutputStream()
    ↳ écrire des données binaire dans le corps de la reponse HTTP
- On ne peut utiliser que l'un ou l'autre (sinon on déclenche une exception IllegalStateException)

Lecture de la requête

Pour lire la requête client, on utilise l'interface :
javax.servlet.ServletRequest

- Lire les paramètres

String **getParameter**(String name)	Lire la valeur du paramètre *name*
String[] **getParameterValues**(String name)	Lire les valeurs du paramètre *name* exemple : un ensemble de checkboxes ayant le même *name*
Enumeration **getParameterNames**()	Récupérer un objet énumération qui contient le nom de tous les paramètres de la requête
Map **getParameterMap**()	Récupérer un Map:clés nom des paramètres valeurs : tableau de string contenant les valeurs

Lecture de la requête

- Lire les en-têtes

String **getContentType**()	Accéder au type MIME du contenu de la requête
String **getCharacterEncoding**()	Obtenir l'encodage du corps de la requête
Locale **getLocale**()	Récupérer la langage par défaut du client
Enumeration **getLocales**()	Récupérer tous les langages préférés du clients ordonnés par préférences sinon le langage du serveur est retourné.
String **getHeader**(String name)	Récupérer l'en-tête qui correspond à name
Enumeration **getHeaders**(String name)	Récupérer les valeurs de toutes les en-têtes correspondant à name
Enumeration **getHeaderNames**()	Récupérer la liste de toutes les en-têtes utilisées dans la requête

Lecture de la requête

- Lire les information de l'URL
  - Information sur le serveur

String **getServerName**()	Récupérer le nom du serveur
int **getServerPort**()	Récupérer le n° de port du serveur
String **getContextPath**()	Récupérer le nom de l'application qui héberge la servlet (Eclipse → nom du projet)
String **getServletPath**()	Récupérer la valeur qui identifie la servlet dans l'application (web.xml → servlet-mapping)
StringBuffer **getRequestURL**()	Récupérer l'URL utiliser pour contacter la servlet

Lecture de la requête

- Information sur le client

String **getRemoteAddr**()	Obtenir l'adresse IP du client
int **getRemotePort**()	Obtenir le port du client

- information concernant le protocole de communication

String **getProtocol**()	Protocol**/**majorVersion**.**minorVersion ex : (HTTP/1.1)
boolean **isSecure**()	ex : true si HTTPS

Java Server Pages

Présentation

- JSP → pages web dynamiques
- Une page JSP contient :
  - de l'HTML pour la partie statique de la page
  - du code Java pour générer la partie dynamique de la page
- Elle constitué de :
  - donné et de tag HTML
  - tag Jsp
  - scriplet (code java intégré à la jsp)
- 2 syntaxes : standard ou XML pur

Fonctionnement interne des JSP

1. Pour chaque JSP, le serveur génère le code source d'une servlet (.java)
2. Le serveur traite toutes les balises HTML pour générer un code source java pour envoyer dans la réponse HTTP les balises
3. Il insère le contenu des balises JSP dans le code source java
4. Il compile la servlet générée (.class)
5. Il crée une instance de la classe et exécute la méthode service
6. La même instance est utilisée si une requête HTTP concerne la page JSP arrive sur le serveur

Scriplets

- **< % %> Insertion d'instruction Java**
  Peut contenir des déclarations de variables, mais de déclarations de méthodes
  Les instructions seront placées dans la méthode **_jspService()** de la classe générée.
- **<%! %> Déclaration de variables ou de méthodes**
  L'emplacement de la balise n'a pas d'importance
  Ils seront insérés comme membre de la classe générée
- **<%= %> Évaluation d'une expression**
  Pas de point virgule
  C'est un raccourcit pour <% out.println();%>
  <%= XXX%> équivaut à <% out.println(XXX);%>
- **< %-- --%> Commentaires**

Exemple de Scriplets

```
<% String s1 = new String("Bonjour !"); %>
<%= s1 %>

<%= s2 %> <%-- Pas d'erreur de compilation -->
<%! String s2 = new String("Bonjour à tous !"); %>
```

Affichage :

Bonjour ! Bonjour à tous !

Objets implicites

Pour l'accès automatique aux éléments de la sejsprvlet, il existe des objets implicites utilisables depuis une JSP

request	référence vers l'objet HttpServletRequest utilisé pour contacter la page jsp
response	référence sur le flux sortant de la servlet. Une JSP ne modifie en général pas explicitement le flux sortant
session	référence vers l'objet HttpSession associé au client
out	flux de sortie sous forme d'un objet JspWriter
pageContext	accès à l'objet pageContext associé à la page JSP
application	référence à l'objet SeverContext associé à l'application web contenant la page jsp
config	référence sur l'objet de type ServletConfig utilisée pour initialiser la page
exception	référence sur l'objet exception, disponible uniquement pour les pages dédiées au traitement d'erreur

Objets implicites

```
<%-- page1.jsp --%>
<form method="post" action="page2.jsp">
<input type="text" name="message" />
<input type="submit" />
</form>
```

```
<%-- page2.jsp --%>
<%
out.println(request.getParameter("message"));
%>
```

Directives

- Les directives permettent de préciser des informations globales sur la page JSP
- La syntaxe est : <%@ directive attribut="valeur" ... %>
- **Directive include**
  ↳ permet d'inclure une autre ressource (html, jsp ou xml) dans une page jsp

  <%@include file="/logo.html " %>
- **Directive taglib**
  ↳ permet le référencement de bibliothèque de balise externes

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

Directives

- **Directive page**
  ↳ elle permet de définir des options qui s'appliquent à toute la JSP
  elle ne doit pas redéfinir un attribut déjà définit sauf import

language="langage"	Indique le langage utilisé dans la page jsp par défaut java
extends="package.nomDeClasse"	Fixer le nom de la classe dont va hériter la classe généré par la page jsp
import="package.nomDeClasse"	Permet de définir les packages importés dans le code généré par le serveur
session="boolean"	Indique si la session est accessible depuis la page jsp par défaut true
errorPage="url"	Lorsqu'une exception est déclenché et qu'elle n'est pas traité dans la page le serveur appelle la page définie par l'url

Directives

isErrorPage="boolean"	Indique si la page est conçue pour le traitement des exceptions
contentType="type mime"	Indique le type de document contenu dans la réponse HTTP ex : text/html
pageEncoding="type d 'encodage"	Type d'encodage des caractères de la réponse HTTP

```
<%@page contentType="text\html"
        pageEncoding="UTF-8"
        errorPage="erreur.jsp"
        import="java.util.*"
%>
```

Éléments d'action

- Permettent :
  - Inclusions & Redirections…
  - Instanciation d'objects
- Syntaxe : <jsp:action…>
- Actions principales :
  - Include
  - UseBean
  - SetProperty
  - GetProperty
  - text...

Utilisations de beans

- Un JavaBean est une classe ayant :
  - Un constructeur public sans arguments
  - Des getters et setters normés pour accéder aux propriétés
  - Une sérialisation possible

```
public class Bateau
      implements java.io.Serializable {
   private String voile;
   public Bateau(){  voile = null;  }
   public String getVoile() {
      return voile;
   }
   public void setVoile(String voile) {
      this.voile = voile;
   }
}
```

Utilisations de beans

Utilisation d'un javabean dans une JSP : <jsp:useBean …/>

Attributs de jsp:useBean

- id : nom de l'instance du bean
- class : nom de la classe Java
- scope : {page|request|session|application}
- type : type du bean
  (optionnel, remplace class si le JavaBean existe)
- beanName : Nom de la classe ou de l'objet sérialisé (optionnel)

Utilisations de beans

```
<jsp:useBean id="myBean"
    class="mybeans.ClientBean"
    scope="session" />
```

```
<%
    ClientBean myBean = null;
    MyBean = (ClientBean)session.
        getAttribute("myBean");

    if(myBean==null){
        myBean = new ClientBean();
    }
%>
```

getProperty, setProperty

- Récupérer ou modifier les attributs du bean

```
<jsp:setProperty name="nameBean"
property="propName" value="Terry"
property="propName"
param="nameParameter"
property="*"
/>

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

getProperty, setProperty

```
<jsp:useBean id="unFormateur"
class="beans.BeanFormateur"/>
<jsp:setProperty name="unFormateur"
property= "prenomF" value="Benjamin"/>
<jsp:getProperty name="unFormateur"
property=" prenomF"/>
```

Affichage :

```
Benjamin
```

Include, param, forward

- **<jsp:include…>**
  - Inclure le résultat d'une page (HTML ou JSP) dans une page JSP.
- **<jsp:param…>**
  - Spécifier des paramètres pour une autre balise (include, forward…).
- **<jsp:forward…>**
  - Redirige vers une autre page (HTML ou JSP)

Exemples

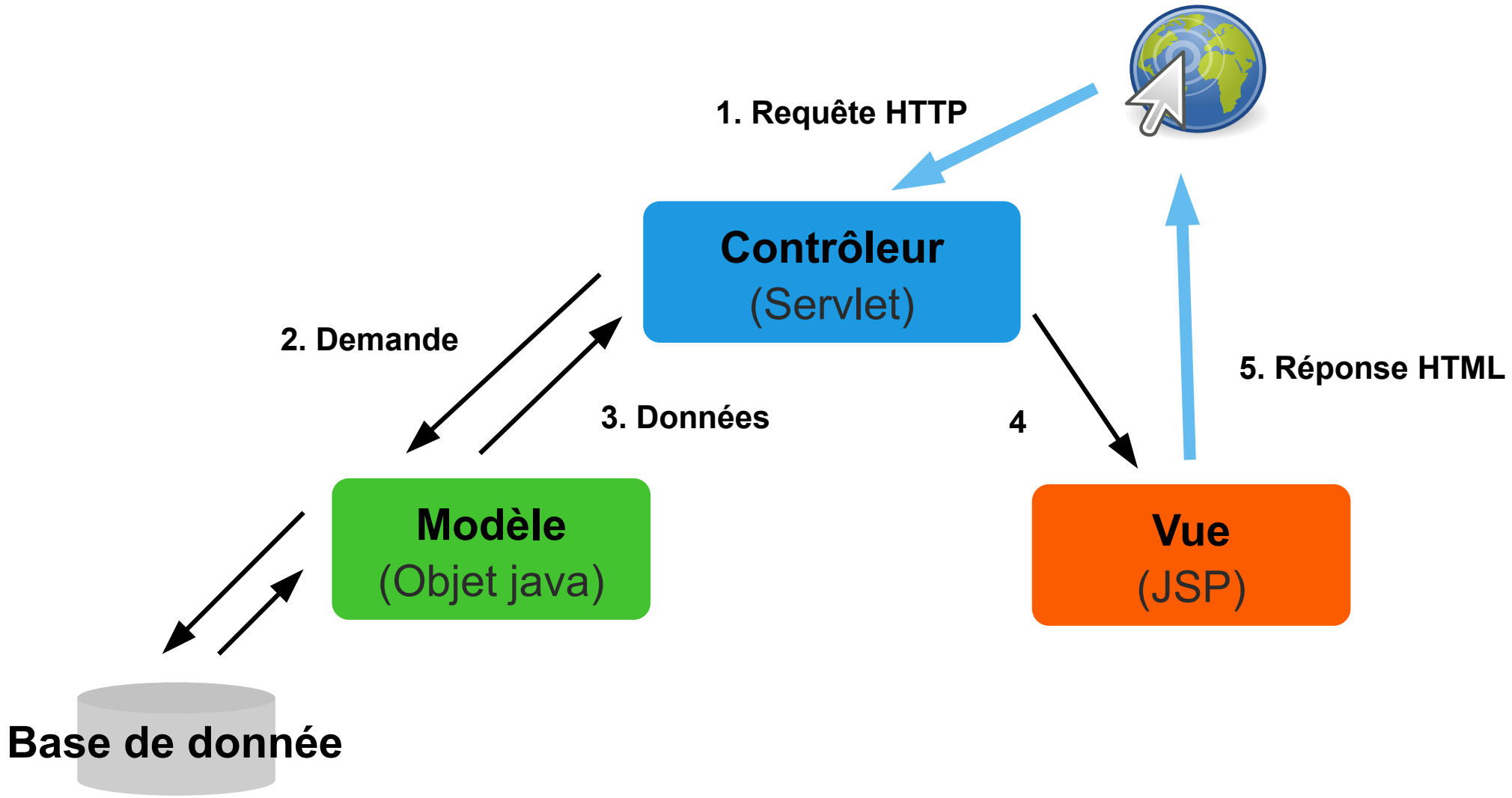
```
<jsp:include page="login.jsp" flush="true">
  <jsp:param name="login" value="coursJSP"/>
  <jsp:param name="pass" value="actionJava"/>
</jsp:include>
```

```
<jsp:forward page="login.jsp">
  <jsp:param name="login" value="coursJSP"/>
  <jsp:param name="pass" value="actionJava"/>
</jsp:forward>
```

```
Vous êtes : <%= request.getParameter("login")
%>
```

Modéle MVC
(Model View Controler)

Servlet + JSP = MVC



Chaînage de pages

- L'objet **RequestDispatcher** :

```
RequestDispatcher rd =
   request.getRequestDispatcher("pageCible");
```

- Inclusion : utilisation de la méthode ***include()*** sur le **RequestDispatcher** :

```
rd.include(request, response);
```

- Redirection : utilisation de la méthode ***forward()*** sur le **RequestDispatcher** :

```
rd.forward(request, response);
```

Ajouter des informations à la requête

setAttribute(String name,Object o)	Stocker un objet dans HttpServletRequest, name identifie l'objet dans la liste d'attribut
Object **getAttribute**(String name)	Lire un objet dans HttpServletRequest identifié par name
removeAttribute(String name)	Supprimer l'objet identifié par name
Enumeration **getAttributeNames**()	Obtenir la liste des noms des attributs de la requête

Cookies

- Un cookie représente un petite quantité d'information envoyé au client , puis renvoyée au serveur par le client
- Création de cookies

Cookie(String name,String value)	Constructeur qui permet de déterminer le nom *name* et la valeur stocké *value* dans le cookie
setMaxAge(int expiry)	Fixer la durée de vie du cookie en seconde **-1** → ne pas stocker le cookie de façon permanente, reste en mémoire jusqu'à la fermeture du client **0** → le client doit supprimer le cookie
setValue(String value)	Modifier la valeur contenue dans le cookie

- Incorporer un cookie à la réponse HTTP

addCookie(Cookie c)	Associer le cookie à la réponse HTTP

Cookies

Lorsque le client génère une requête HTTP, il ajoute automatiquement les cookies que l'application avait transféré

- Lecture des Cookies

Cookies[] **getCookies**()	Retourne tous les cookies de la requête (null s'il n'y en a pas)
String **getValue**()	Retourne la valeur contenue dans le cookie
String **getName**()	Retourne le nom du cookie

```
// Création cookie
Cookie c = new Cookie("login_cookie" , "admin");
response.addCookie(c);
// Lecture des cookies
Cookie[] listeCookies =request.getCookies();
String name = listeCookies [0].getName();
String value = listeCookies [0].getValue();
```

Session

- La notion de session crée donc une notion de persistance, durant une certaine période fixée à l'avance.
- La gestion d'une session peut se faire de 2 manières:
  - Stockage de l'identifiant de session dans un cookie
  - L'identifiant de session est ajouté à l'URL pour GET ou aux paramètres interne pour POST
- Obtenir une session

HttpSession **getSession**()	retourne la session associée à la requête HTTP ou une nouvelle session si elle n'existe pas
HttpSession **getSession**(boolean create)	idem sauf retourne null si *create* est à false et si la session n'existe pas

Session

- Stocker, extraire et supprimer des éléments

setAttribute(String name, Object value)	Ajouter ou remplacer un objet *value* dans la session, *name* nom utilisé pour identifier l'objet dans la session
Object **getAttribute**(String name)	retourne l'objet placé dans la session identifié par *name*, null s'il n'existe pas
removeAttribute(String name)	Supprime l'élément de la session identifié par *name*
Enumeration **getAttributeNames**()	Retourne le nom de tous les éléments de la session
String **getId**()	Retourne l'identifiant de session généré par le serveur

```
HttpSession session = request.getSession();
Object obj = new Object();
session.setAttribute("table", obj);
Object obj = session.getAttribute("table");
```

Session

- Mettre à fin à la session

invalidate()	Destruction de la session
Int **getMaxInactiveInterval**()	Obtenir la durée de vie de la session en secondes
setMaxActiveInterval(int interval)	Fixer la durée de vie uniquement pour cette session

La valeur de la durée de vie (en minute) par défaut est configurable par le descripteur de déploiement web.xml

```
<session-config>
   <session-timeout>300</session-timeout>
</session-config>
```

Gestion des erreurs

- Déclarer une page d'erreur dans la JSP

```
<%@ page errorPage="ShowError.jsp" %>
```

- Définir une JSP comme étant une page d'erreur

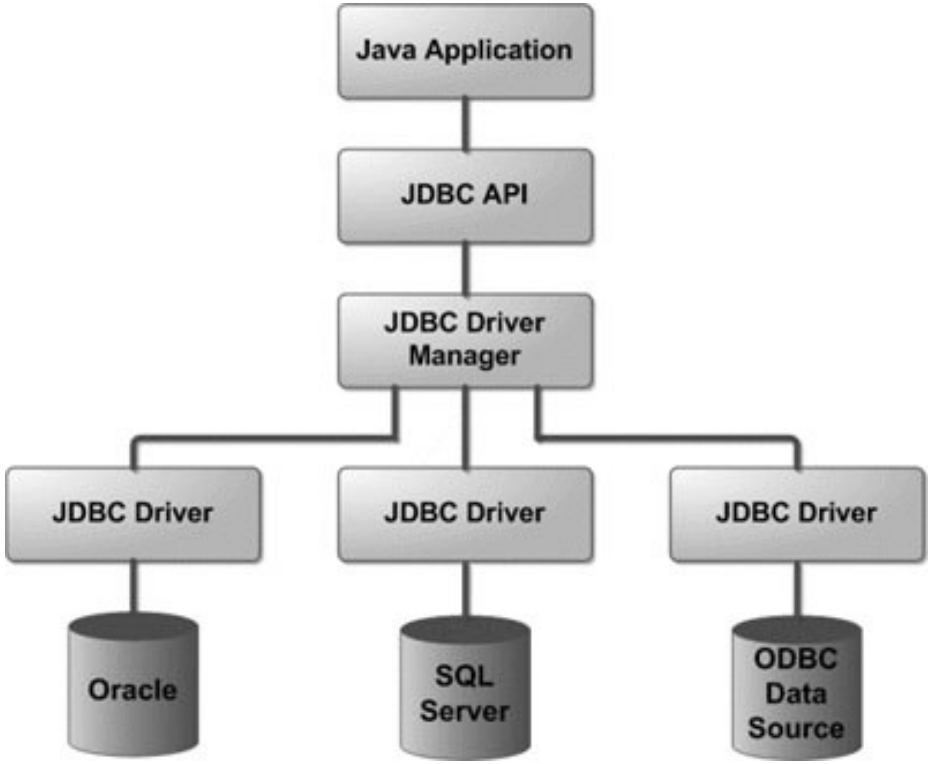
```
<%@ page isErrorPage="true" %>
```

- Variable implicite : exception

```
<%
exception.printStackTrace(response.getWriter(
)); %>
```

Base de données

- Rappel sur JDBC



Intercepteur de requête

- L'interface Filter permet d'implémenter un intercepteur de requête.
- Utilisations possibles : gestion des rôles, redirection, enrichissement de la requête, préparation de la réponse...
- Deux configurations possibles :
  - web.xml
  - annotations

Interface Filter : web.xml

- Interface Filter : web.xml

```
<filter>
	<description>Description de mon filtre</description>
	<display-name>LoginFilter</display-name>
	<filter-name>LoginFilter</filter-name>
	<filter-class>fr.dawan.projet2.controllers.LoginFilter</filter-class>
</filter>
<filter-mapping>
	<filter-name>LoginFilter</filter-name>
	<url-pattern>/backoffice/*</url-pattern>
	<dispatcher>REQUEST</dispatcher>
	<dispatcher>FORWARD</dispatcher>
	<dispatcher>INCLUDE</dispatcher>
	<dispatcher>ERROR</dispatcher>
</filter-mapping>
```

Interface Filter : annotations

```
@WebFilter(urlPatterns = { "/backoffice/*" },
                dispatcherTypes = {
DispatcherType.FORWARD,
DispatcherType.INCLUDE,
DispatcherType.REQUEST,
DispatcherType.ERROR })
public class LoginFilter implements Filter {
// ...
}
```

Ecouteurs d'événements

- Des interfaces « listener » permettent « d'écouter » des événements (création, destruction, modification des attributs...) au niveau :
  - de la session : HttpSessionListener, HttpSessionAttributeListener
  - de la servlet : ServletContextListener, ServletContextAttributesListener
  - de la requête : ServletRequestListener, ServletRequestAttributeListener
  - ...

Configuration des listeners

- 2 configurations possibles
  - Par le web.xml :

```
<listener>
    <listener-class>
        momDomaine.maClasseListener
    </listener-class>
</listener>
```

  - Par annotation sur la classe implémentant le listener :
    @WebListener Configuration des listeners

Taglibs

Définition : Taglibs

- Bibliothèques de balises personnalisées
- Comme des JavaBeans, elles permettent une séparation du code Java et du code de la JSP
- Facilitent la gestion d'une application web
- Il existe plusieurs bibliothèques populaires :
  - JSTL
  - Taglibs de Struts
  - DisplayTag
  - Jakarta TagLib

Fonctionnement des Taglibs

- Un identificateur pour chaque balise personnalisée : Une classe Java implémentant javax.servlet.jsp.tagext.Tag

- Un fichier TLD (Tag Library Descriptor) : lien entre la JSP et l'identificateur du tag. Il décrit également un ensemble de balises

- Le chemin vers le TLD peut être spécifié dans le descripteur de déploiement (optionnel)

Déploiement : web.xml

Pour un code optimisé, et de l'aide de la part de l'éditeur, on peut indiquer le chemin vers le TLD dans le descripteur de déploiement web.xml

```
<jsp-config>

   <!-- taglibs -->
   <taglib>
      <taglib-uri>uriName</taglib-uri>
      <taglib-location>TLDPath</taglib-location>
   </taglib>

</jsp-config>
```

Utilisation de Taglibs dans une JSP

- Déclaration de la bibliothèque, 2 attributs nécessaires :
  - "uri" (le nom, lire dans la TLD)
  - "prefix" (pour l'utilisation dans la JSP)

```
<%@taglib uri="uriName" prefix="pr" %>
```

- Utilisation par le "tagName" depuis la bibliothèque

```
<pr:tagName attribute="blue"…>
```

Java Standard Tag Library

- Spécifications par SUN (implémentation par Apache)
- 4 bibliothèques de balise (+ Functions, pour l'EL) :
  - Core : fonctions de base
  - XML : traitements XML
  - Format : internationalisation de JSP
  - Database : requêtes SQL
- Installation :
  - Téléchargement : https://jstl.dev.java.net/download.html
  - Copie des TLD (depuis le JAR d'implémentation, répertoire META-INF/) dans le répertoire /WEB-INF/tlds et des JAR dans le répertoire /WEB-INF/lib

JSTL : Configuration de Maven

```
<dependency>
	<groupId>javax.servlet</groupId>
	<artifactId>jstl</artifactId>
	<version>1.2</version>
</dependency>
```

Le langage d'expressions

- Langage permettant de faciliter l'acces aux objets Java depuis une page JSP (lecture uniquement)

- Syntaxe : ***${expression }***

```
${unFormateur.prenomF}
${chaises[2].pieds['gauchedevant'].taille}
${soleil.luminosite * 1830}
```

- Utilisable dans les balises autorisant les expressions

```
<c:out value='${texte}' default='rien' />
```

Langage d'expressions: objet implicite

Objet implicite	Rôle
PageScope	variable contenue dans la portée de la page (PageContext)
RequestScope	variable contenue dans la portée de la requête (HttpServletRequest)
SessionScope	variable contenue dans la portée de la session (HttpSession)
ApplicationScope	variable contenue dans la portée de l'application (ServletContext)
Param	paramètre de la requête http
ParamValues	paramètres de la requête sous la forme d'une collection
Header	en−tête de la requête
HeaderValues	en−têtes de la requête sous la forme d'une collection
InitParam	paramètre d'initialisation
Cookie	Cookie
PageContext	objet PageContext de la page

Langage d'expression: opérateurs

Operateur	Rôle
.	Obtenir une propriété d'un objet ${param.nom}
[]	Obtenir une propriété par son nom ou son indice
empty	Teste si un objet est null ou vide pou une chaîne de caractère. ${empty param.nom}
== ou eq	teste l'égalité de deux objets
!= ou ne	teste l'inégalité de deux objets
< ou lt	test strictement inférieur
> ou gt	test strictement supérieur
<= ou le	test strictement supérieur
>= ou ge	test supérieur ou égal
+ − * / ou div % ou mod	Opérateur mathématiques
&& ou and, || ou or, ! ou not	Opérateurs logiques

JSTL: bibliothèque de base

Balise de gestion des variables

- Affectation d'une variable → **<c:set>**

  Attributs :

  - **var** : le nom de la variable à créer ou à modifier
  - **scope** : page (par défaut), request, session, application
  - **value** : la valeur à enregistrer dans la variable
  - **target** : attribut utilisé pour modifier une propriété d'un objet
  - **property** : attribut utilisé avec **target** pour identifier la propriété de l'objet

JSTL: bibliothèque de base

- Destruction d'une variable → **<c:remove>**

  Attributs :

  - **name** : le nom de la variable
  - **scope** : page (par défaut), request, session, application

- Envoyer dans le flux de sortie de la JSP le résultat de l'évaluation de l'expression → **<c:out>**
  Attributs :

  - **value** : valeur à afficher (obligatoire)
  - **default** : définir une valeur par défaut si la valeur est null
  - **escapeXml** : booléen qui précise si les caractères particuliers (< > & ...) doivent être convertis en leurs équivalents HTML (< > &amp ; ...)

JSTL: bibliothèque de base

Opération conditionnelles

- **<c:if>**
  attributs
  - **test** : expression à évaluer
  - **var** et **scope** : permet de stocker le résultat du test dans une variable (var) et à un emplacement (scope)
- **<c:choose>** équivaut à un switch en java
  - **<c:when>** correspond au case
    attribut test donne l'expression à évaluer
  - **<c:otherwise>** correspond au default

JSTL: bibliothèque de base

Les itérations

- **<c:foreach>**
  - Parcours d'une liste d'éléments (tableau, collection)
    Attribut :
    - **Items** (obligatoire) : défini le tableau ou la collection
    - **begin** et **end** : index de début et de fin d'itération
  - Boucle équivalente à un for en java
    Attribut :
    - **begin** et **end** (obligatoire) : borne de départ et de fin de l'itération
    - **step** : pas de l'itération par défaut 1

JSTL: bibliothèque de base

- Attribut **var** : référence l'élément courant ou la valeur courante du compteur
- Attribut **varStats** : nom de la variable pour suivre l'évolution de la boucle. Elle a pour propriétés
  - **index** : indice courant de l'itération
  - **current** : objet courant de l'itération
  - **count** : indique le nombre de passage dans la boucle
  - **first** : true, si c'est la première itération
  - **last** : true, si c'est la dernière itération

JSTL: bibliothèque de base

- **<c:forTokens>** équivalent à split pour un string en java
  Attributs :
  - **items** : contient la chaîne de caractère
  - **delims** : contient le caractère de délimitation
  - **var** : contient les chaine de caractère contenant les mots successivement
  - **begin**, **end** et **step** : idem foreach

JSTL: bibliothèque de base

Manipulation des url

- **<c:import>** équivaut à <jsp:include>
  Attribut :
  - **url** : nom de la ressource
- **<c:redirect>** redirection
  - **url** : url de la redirection
- **<c:url>** écrire une URL
  Attribut :
  - **value** : l'url
- **<c:param>** paramètre de l'url (enfant des balises précédentes)
  attribut **name** : nom du paramètre et **value** valeur du paramètre