

Java Enterprise Edition

Chapitre 03 : Java EE

Sébastien Chèvre, 2019

Plan du module – chapitres

Java Entreprise Edition - JEE

- Chapitre 01 : Introduction et motivations
- Chapitre 02 : Machine Virtuelle Java
- **Chapitre 03 : Java EE**
- Chapitre 04 : *Spring Boot & Spring MVC*
- Chapitre 05 : *ORM & DAO avec Spring Data & JPA*
- Chapitre 06 : Sécurité avec *Spring Security*
- Chapitre 07 : Tests avec *Spring Testing*
- Chapitre 08 : SOA & Micro-services avec *Spring*
- Chapitre 09 : *JMS (Java Messaging Service)*

- Projet

Objectifs du chapitre

Chapitre 03 : JEE

A la fin de ce chapitre, les étudiants peuvent:

- Comprendre les principaux «building blocks» d'une application **JEE**
- Implémenter les principaux «building blocks» d'une application **JEE**
- Implémenter une application simple **JEE** de bout en bout (DB comprise)
- Expliquer les avantages/inconvénients de **JEE**

Plan du chapitre 3

Chapitre 03 : JEE

- **Concept de base**
- **Architecture et Cycle de vie**
- **HTTP/Web servlets**
 - Principe
 - Context
 - Request, Response
 - Paramètres
 - Dispatching
 - Session
 - Cookies
 - Listeners
 - Filtres
 - Connexion DB
- **Mise en pratique**

JEE

Sémantique des diapositives



À savoir théoriquement (TE)



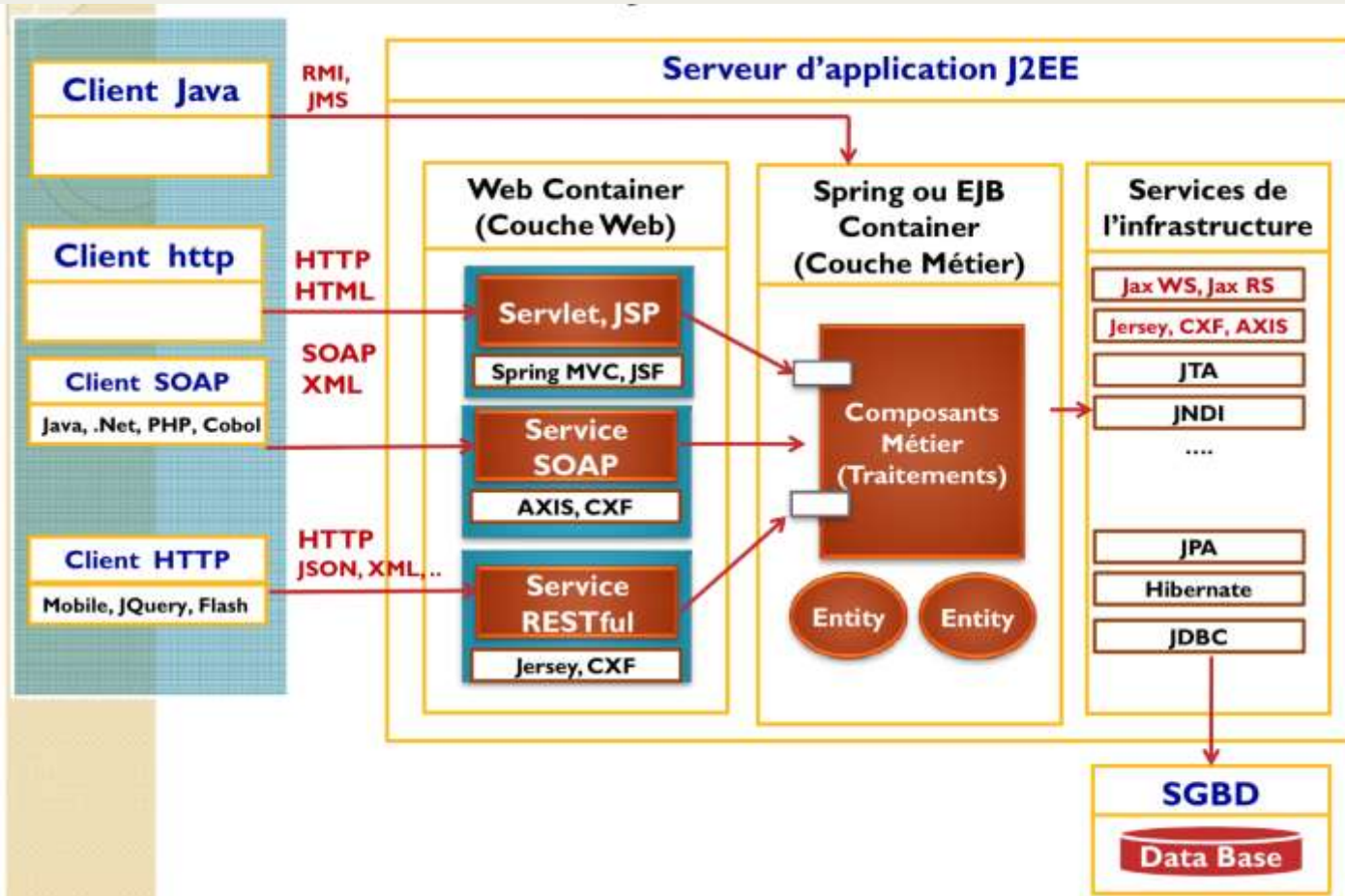
Sensibilisation, illustration de concepts, exemples



Aspects pratiques

Java Entreprise Edition

Architecture de référence



Concepts de base



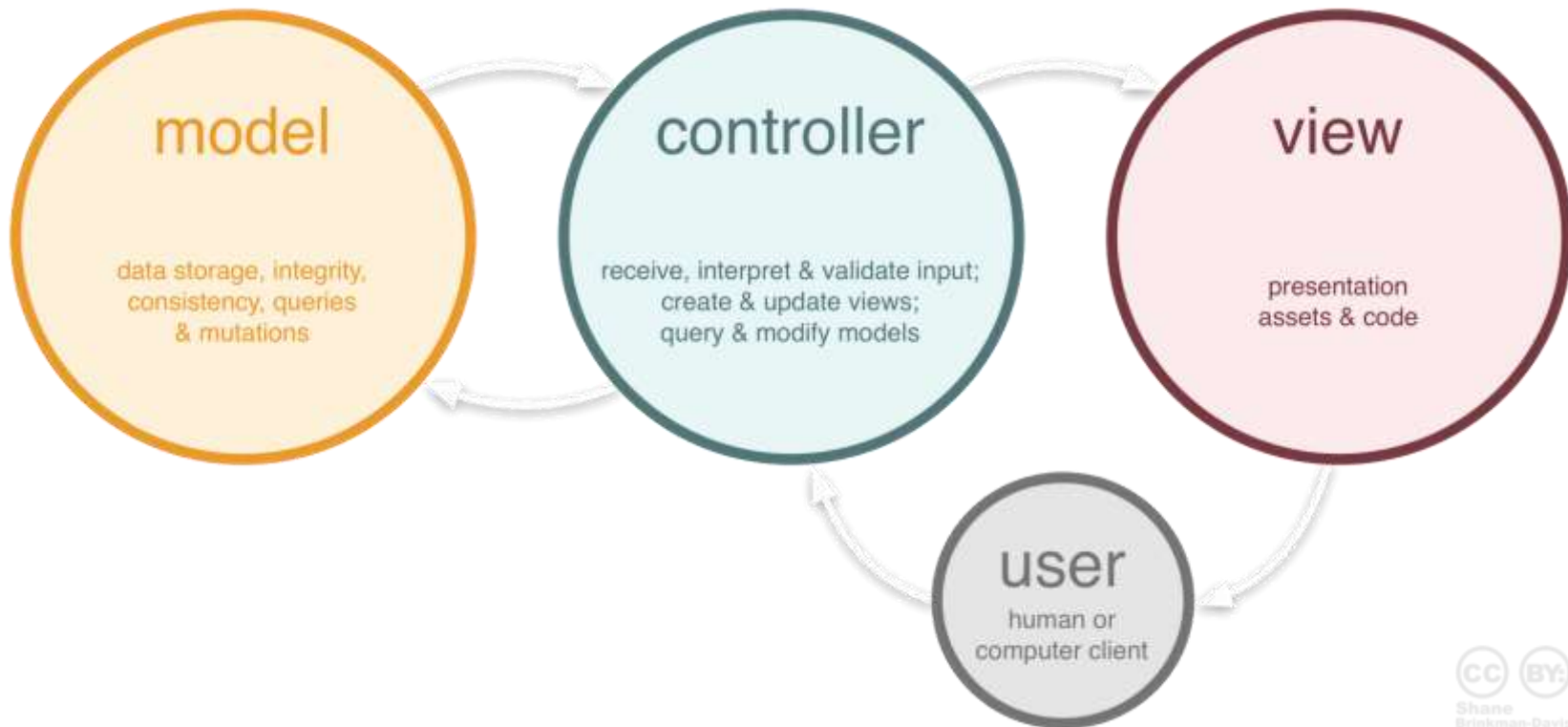
MVC

Modèle – Vue - Contrôleur



- **MVC** est un *Design Pattern* qui signifie Modèle-Vue-Contrôleur
- C'est une façon de structurer votre application en couches logiques:
 - les éléments qui touchent au **Modèle** (la persistance, en gros la base de données, mais aussi le métier)
 - la **Vue** (la façon dont les données sont affichées, les templates)
 - et enfin les **Contrôleurs** (gestion de la cinématique, lancement des traitements)

MVC interactions



JEE

Servlets



■ Une Servlet...

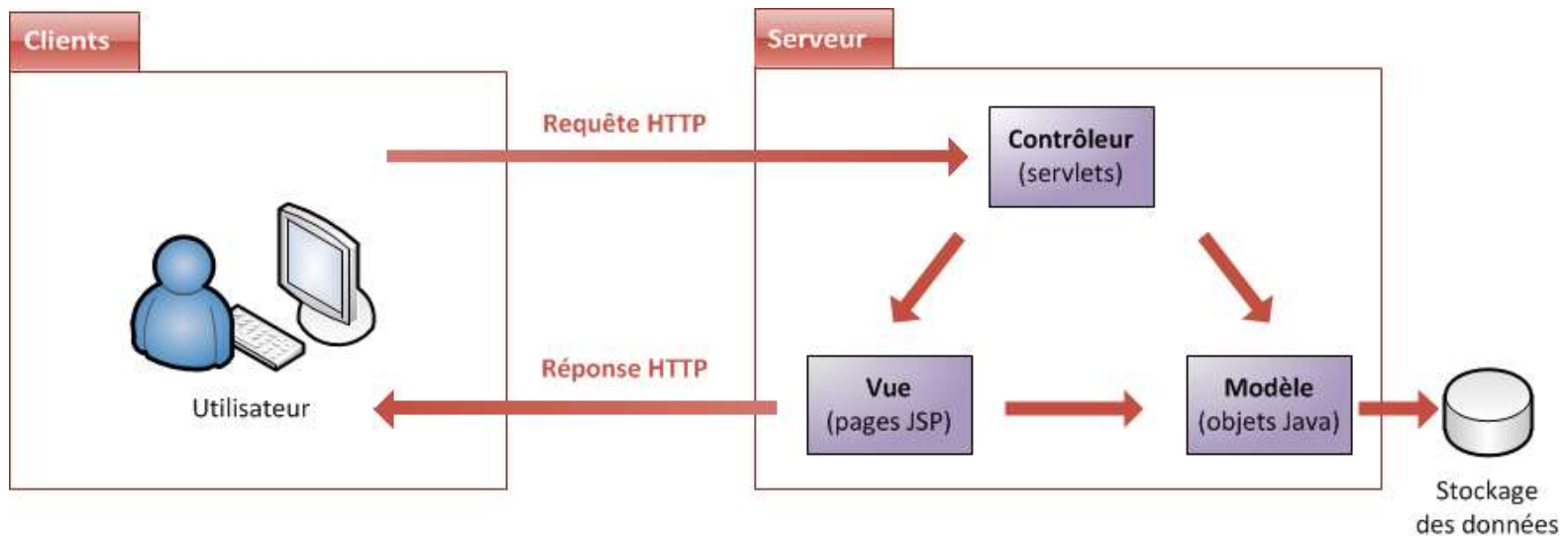
- est une classe héritant de ***HttpServlet***
- permet la programmation d'applications coté serveur
- permet l'extension d'un serveur Web en java
- permet la construction d'application Web dynamique
- équivalent des CGI en java

JEE

Servlets – MV Contrôleur



- La servlet joue le rôle de contrôleur : gestion de la cinématique, lancement des traitements

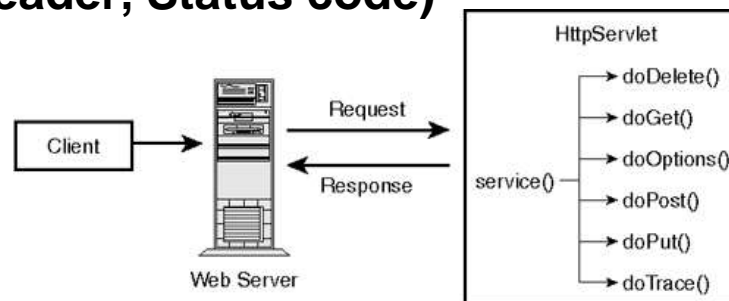


JEE

Fonctionnement des Servlets



- Une **servlet** lit les données envoyées par un client Web (transmises par le serveur)
 - Données explicites (Formulaire, données)
 - Données implicites (Request Header)
 - Implicites mais modifiables
- Elle génère un résultat
- Elle envoie le résultat au client
 - Données explicites (Page HTML, données)
 - Données implicites (Response Header, Status code)
 - Implicites mais modifiables



Architecture et cycle de vie



REPRODUCTION

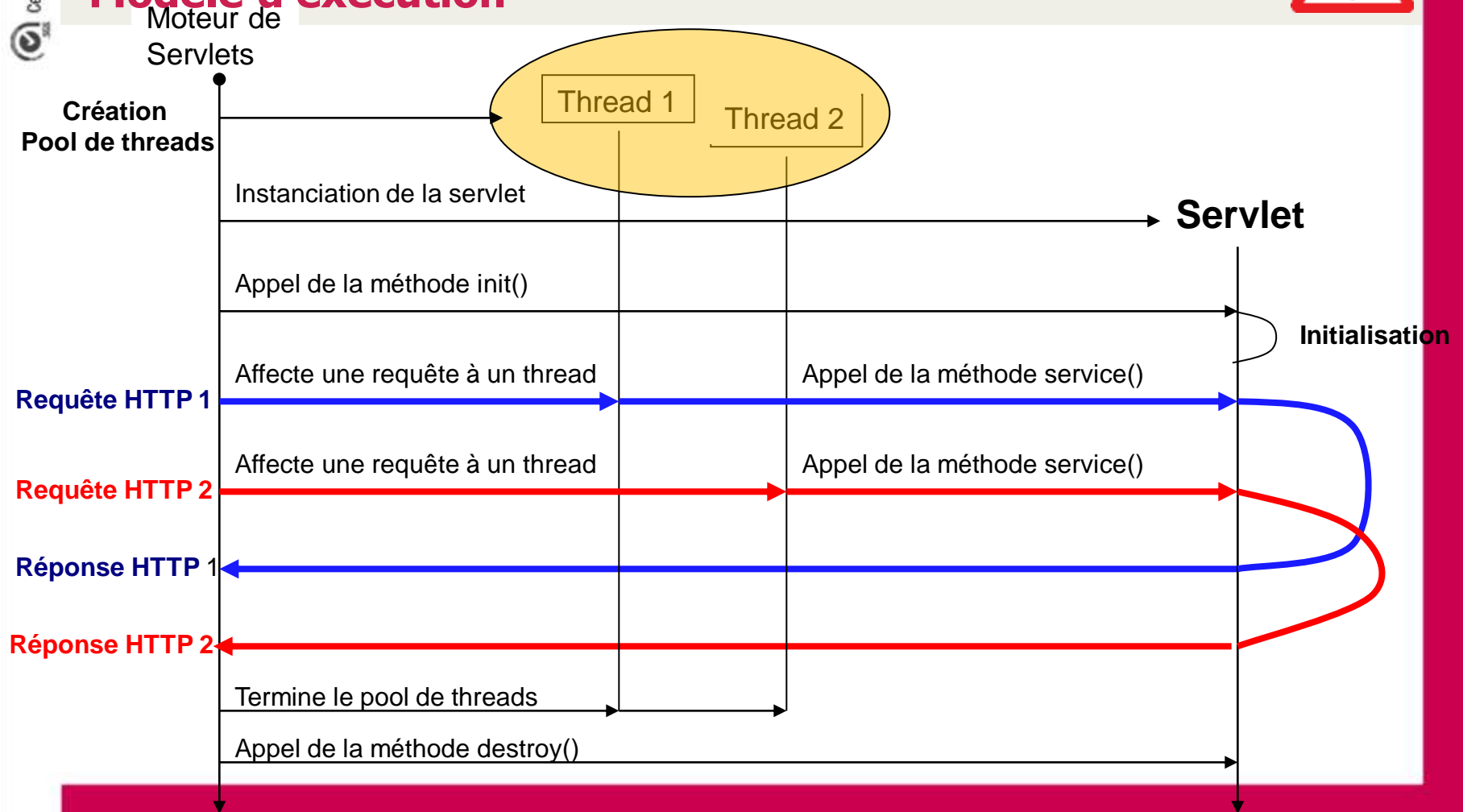
Servlets composants java



- Les servlets sont des **composants** java
 - Ils s'exécutent dans **un container** (container de servlets)
 - Ils doivent se conformer à **une interface** prédéfinie
 - C'est le container qui décide de leur **instantiation**
- Les containers
 - Environnement d'exécution des servlets
 - Tomcat (jakarta.apache.org/tomcat)
 - Jetty
 - Weblogic
 - Glassfish

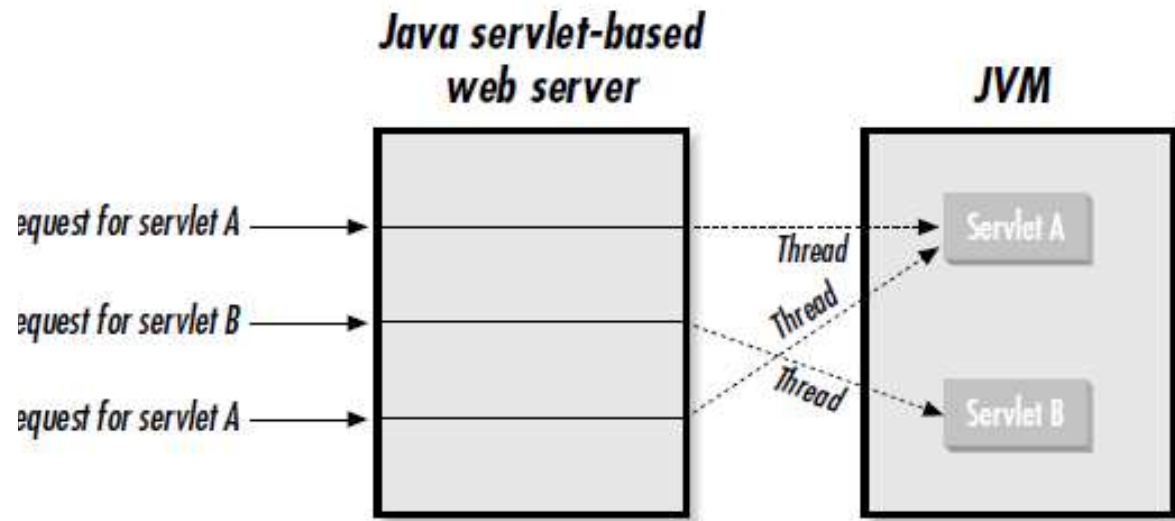
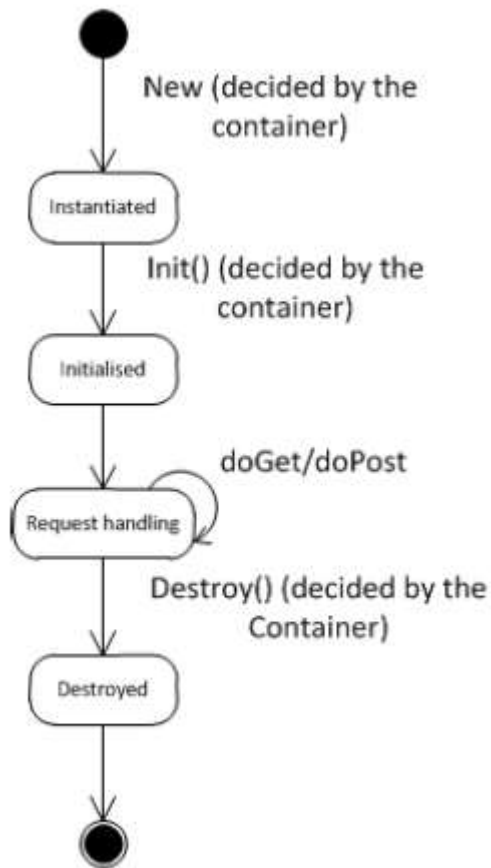
Servlets

Modèle d'exécution



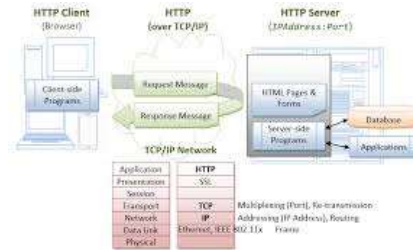
Servlets

Cycle de vie



- **Modèle I/O bloquant**
- **Voir chapitre 1, aspect réactif non bloquant**

HTTP Servlet



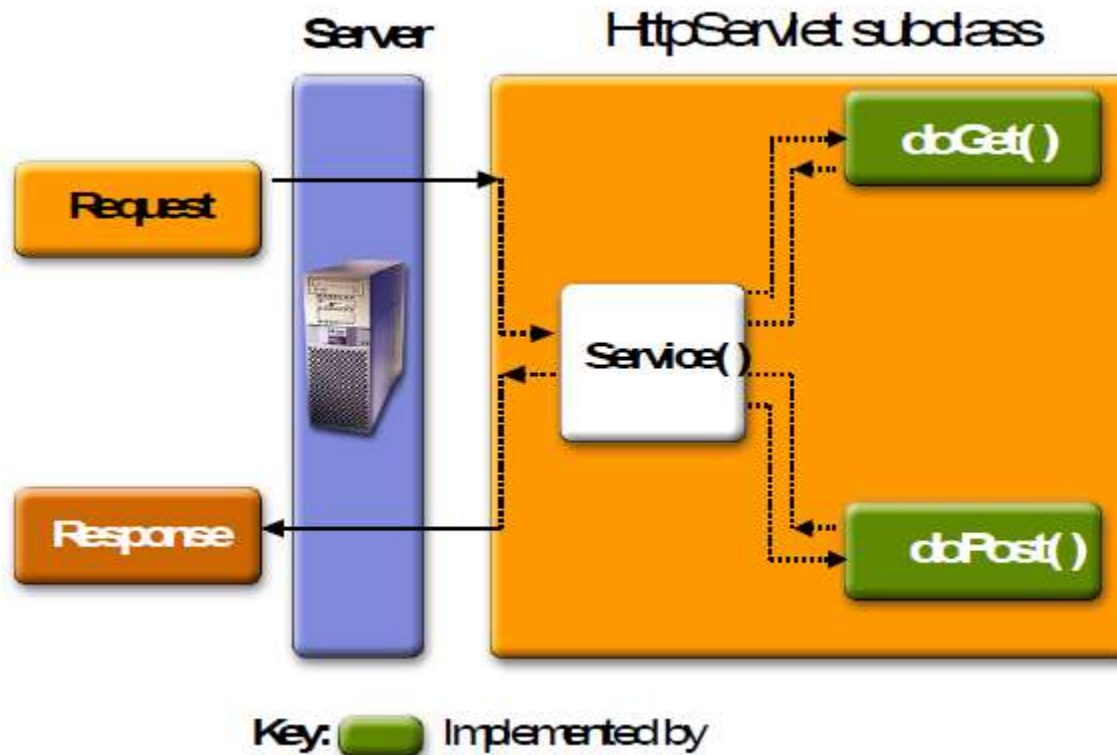
HTTP Servlet

Introduction



- Pour faciliter le traitement particulier des serveurs Web, la classe **Servlet** doit hérité de ***javax.servlet.http.HttpServlet***
 - 2 méthodes remplacent ***service()*** de la classe mère :
 - **doGet()** : pour les requêtes Http de type GET
 - **doPost()** : pour les requêtes Http de type POST
 - la classe ***servlet*** doit obligatoirement contenir l'une ou l'autre de ces 2 méthodes redéfinie, choisie selon le mode d'envoi du formulaire HTML qui l'exécute
 - ***service()*** de ***HttpServlet*** appelle automatiquement la bonne méthode en fonction du type de requêtes Http

HTTP Servlet Introduction



HTTP Servlet Javadoc



■ HTTP Servlet Javadoc

Modifier and Type	Method and Description
protected void	doDelete (<code>HttpServletRequest req, HttpServletResponse resp</code>) Called by the server (via the <code>service</code> method) to allow a servlet to handle a DELETE request.
protected void	doGet (<code>HttpServletRequest req, HttpServletResponse resp</code>) Called by the server (via the <code>service</code> method) to allow a servlet to handle a GET request.
protected void	doHead (<code>HttpServletRequest req, HttpServletResponse resp</code>) Receives an HTTP HEAD request from the protected <code>service</code> method and handles the request.
protected void	doOptions (<code>HttpServletRequest req, HttpServletResponse resp</code>) Called by the server (via the <code>service</code> method) to allow a servlet to handle a OPTIONS request.
protected void	doPost (<code>HttpServletRequest req, HttpServletResponse resp</code>) Called by the server (via the <code>service</code> method) to allow a servlet to handle a POST request.
protected void	doPut (<code>HttpServletRequest req, HttpServletResponse resp</code>) Called by the server (via the <code>service</code> method) to allow a servlet to handle a PUT request.
protected void	doTrace (<code>HttpServletRequest req, HttpServletResponse resp</code>) Called by the server (via the <code>service</code> method) to allow a servlet to handle a TRACE request.
protected long	getLastModified (<code>HttpServletRequest req</code>) Returns the time the <code>HttpServletRequest</code> object was last modified, in milliseconds since midnight January 1, 1970 GMT.
protected void	service (<code>HttpServletRequest req, HttpServletResponse resp</code>) Receives standard HTTP requests from the public <code>service</code> method and dispatches them to the <code>doXXX</code> methods defined in this class.
void	service (<code>ServletRequest req, ServletResponse res</code>) Dispatches client requests to the protected <code>service</code> method.



HTTP Servlet Javadoc



Hierarchy For Package javax.servlet.http

Package Hierarchies:
All Packages

■ javax.servlet.http

Class Hierarchy

- java.lang.Object
 - javax.servlet.http.Cookie (implements java.lang.Cloneable, java.io.Serializable)
 - java.util.EventObject (implements java.io.Serializable)
 - javax.servlet.http.HttpSessionEvent
 - javax.servlet.http.HttpSessionBindingEvent
 - javax.servlet.GenericServlet (implements java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig)
 - javax.servlet.http.HttpServlet
 - javax.servlet.http.HttpUtils
 - javax.servlet.ServletRequestWrapper (implements javax.servlet.ServletRequest)
 - javax.servlet.http.HttpServletRequestWrapper (implements javax.servlet.http.HttpServletRequest)
 - javax.servlet.ServletResponseWrapper (implements javax.servlet.ServletResponse)
 - javax.servlet.http.HttpServletResponseWrapper (implements javax.servlet.http.HttpServletResponse)

Interface Hierarchy

- java.lang.AutoCloseable
 - javax.servlet.http.WebConnection
- java.util.EventListener
 - javax.servlet.http.HttpSessionActivationListener
 - javax.servlet.http.HttpSessionAttributeListener
 - javax.servlet.http.HttpSessionBindingListener
 - javax.servlet.http.HttpSessionIdListener
 - javax.servlet.http.HttpSessionListener
- javax.servlet.http.HttpSession
- javax.servlet.http.HttpSessionContext
- javax.servlet.http.HttpUpgradeHandler
- javax.servlet.http.Part
- javax.servlet.ServletRequest
 - javax.servlet.http.HttpServletRequest
- javax.servlet.ServletResponse
 - javax.servlet.http.HttpServletResponse

Servlets

Première servlet



```
@WebServlet(  
    name="basicServlet",  
    urlPatterns = "/info/*"  
)  
public class BasicController extends HttpServlet{  
  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
        throws ServletException, IOException {  
  
        //récupération de l'objet permettant d'écrire dans le flux de sortie  
        PrintWriter out = resp.getWriter();  
  
        out.append("<p>Hello World</p>");  
  
    }  
}
```

- Répond à toutes les chemins **/info**, avec la méthode http **GET**

Servlet Mapping

web.xml



- Permet de construire la relation entre un servlet et son URL (*web.xml*)

```
<servlet-mapping>
```

```
  <servlet-name>Test</servlet-name>
```

```
  <url-pattern>/Test/*</url-pattern>
```

```
</servlet-mapping>
```

- Tous les urls correspondant à

http://host:port/webapp/url-pattern

- déclencherons l'exécution de la servlet
- Exemple:

http //localhost:8080/MyFirstApp/Test/accueil

Servlet Mapping annotations



- Permet de construire la relation entre un servlet et son URL

```
@WebServlet(  
    name = "test",  
    urlPatterns = "/test/*"  
)
```

- Tous les urls correspondant à **http://host:port/webapp/url-pattern** déclencherons l'exécution de la servlet

Servlet

Les éléments du chemin de requête



- **ContextPath** : le chemin du contexte de déploiement
- **ServletPath** : la section du chemin qui a déclenché le mapping
- **PathInfo** : la partie de la requête qui n'est ni le *ContextPath* ni le *ServletPath*

- Méthodes:
 - `request.getContextPath()`
 - `request.getServletPath()`
 - `request.getPathInfo()`

Servlet

Les éléments du chemin de requête



- *Code dans la servlet*

```
String contextPath = req.getContextPath();
String servletPath = req.getServletPath();
String pathInfo = req.getPathInfo();

PrintWriter out = resp.getWriter();

out.append("<p>ContextPath: " + contextPath + "</p>");
out.append("<p>ServletPath: " + servletPath + "</p>");
out.append("<p>PathInfo: " + pathInfo + "</p>");
```

- *Mapping*

```
@WebServlet(
    name="basicServlet",
    urlPatterns = "/info/*"
)
```

- ***http://localhost:8080/ServletBasics/info/test***

- *contextPath: /ServletBasics*
- *servletPath: /info*
- *pathInfo: /test*

- ***http://localhost:8080/ServletBasics/info/test/test2***

- *contextPath: /ServletBasics*
- *servletPath: /info*
- *pathInfo: /test/test2*

Servlet

Paramètres de requête



- Les paramètres d'une requête sont les paramètres envoyés **par le client** au serveur
 - ***Toujours de type chaîne de caractère***
 - ***Immuable, le serveur ne peut pas les modifier***

```
String test = req.getParameter("test");  
String toto = req.getParameter("toto");
```

```
@WebServlet(  
    name="paramServlet",  
    urlPatterns = "/param"  
)
```

- ***http://localhost:8080/ServletBasics/param?test=2&toto=ok***
 - *contextPath: /ServletBasics*
 - *servletPath: /param*
 - *pathInfo: /null*

Servlet

Attributs de requête



- Les attributs de requêtes sont des objets Java, **utilisé uniquement pour le côté serveur** (servlet, JSP)
 - *De n'importe quelle type Java (serialisable)*

```
req.setAttribute("test", new Personne(nom, prenom));  
req.setAttribute("toto", "toto");  
  
...  
Personne personne = (Personne) req.getAttribute("test");  
  
String toto = req.getAttribute("toto");
```

HTTP Servlet

ServletContext



- Correspond au contexte de l'application Web
- Maintient des données pour toute l'application
- Fournit le ***request dispatcher***
- Données d'initialisations pour toute l'application
- Un ***ServletContext*** par application et par JVM
- Attribut de context → Valeurs disponibles pour toute l'application
 - Peut être un objet (sérialisable!)

```
Integer i = (Integer)getContext().getAttribute("test")

if(null == i){
    getContext().setAttribute("test",1)
}
else{
    getContext().setAttribute("test",i + 1)
}

out.append(String.format("test count: %d" ,i))
```

HTTP Servlet

HttpServletRequest



- interface ***HttpServletRequest***
- Fournit les informations sur la requête du client au serveur
- Principales méthodes:
 - ***String getParameter(String name)***
 - ***Enumeration getParameterNames()***
 - Retourne une énumération de tous les noms de paramètres
 - ***String[] getParameterValues()***
 - Retourne un tableau contenant toutes les valeurs des paramètres
 - ***String getHeader(String name)***
 - ***Enumeration getHeaderNames()***
 - Retourne une énumération de tous les noms des propriétés du header
 - ***String[] getHeaderValues()***
 - Retourne un tableau de toutes les valeurs du header

HTTP Servlet

RequestHeader



```
out.append("<h2> Header values </h2>");

Enumeration headerNames = req.getHeaderNames();

while(headerNames.hasMoreElements()) {

    String headerName = (String) headerNames.nextElement();
    String headerValue = req.getHeader(headerName);
    out.append(String.format("<p>Header: %s, valeur: %s", headerName, headerValue ));

}
```

Header values

Header: accept, valeur: image/gif, image/jpeg, image/pjpeg, application/x-ms-application, application/xhtml+xml, application/x-ms-xbap, application/msword, application/vnd.ms-powerpoint, application/vnd.ms-excel, */*

Header: accept-language, valeur: fr-CH

Header: ua-cpu, valeur: AMD64

Header: accept-encoding, valeur: gzip, deflate

Header: user-agent, valeur: Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.2; Win64; x64; Trident/7.0)

Header: host, valeur: localhost:8080

Header: connection, valeur: Keep-Alive

HTTP Servlet

RequestParameters



```
out.append("<h2> Param values </h2>");

Enumeration paramNames = req.getParameterNames();

while(paramNames.hasMoreElements()) {

    String paramName = (String) paramNames.nextElement();
    String paramValue = req.getParameter(paramName);
    out.append(String.format("<p>Paramètres: %s, valeur: %s", paramName, paramValue ));

}
```

- <http://localhost:8080/ServletBasics/info/test?nom=Mouse&prenom=Mickey>
 - nom: **Mouse**
 - prenom: **Mickey**

HTTP Servlet

HttpServletResponse



- Objet fournissant les services pour envoyer la réponse à un client
- Les principales méthodes sont:
 - ***java.io.PrintWriter getWriter()***
 - Pour récupérer un objet `PrintWriter` qui permet d'écrire dans le flux de sortie
 - ***public void setContentLength(int len)***
 - Positionne le header `Content-Length`
 - ***public void setContentType(String type)***
 - Position le header `Content-Type` (exemple `text/html`)
 - ***public void sendError(int sc, String msg) throws java.io.IOException***
 - Envoi un message d'erreur au client (cf code dans l'API)
 - ***public void setHeader(java.lang.String name, java.lang.String value)***
 - Définis un header dans la réponse

HTTP Servlet

HttpServletResponse



```
@WebServlet(  
    name="basicServlet",  
    urlPatterns = {"/info/*"}  
)  
  
public class BasicController extends HttpServlet{  
  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  
  
        //récupération de l'objet permettant d'écrire dans le flux de sortie  
        PrintWriter out = resp.getWriter();  
  
        //header content length  
        resp.setContentLength(1024);  
  
        //header content-type  
        resp.setContentType("text/html");  
  
        //définition d'un header custom  
        resp.setHeader("monCustomHeader", "customHeader");  
  
        out.append("<p>Hello World</p>");  
    }  
}
```



HTTP Servlet

RequestDispatcher



■ L'objet *RequestDispatcher* est utilisé pour

- Transférer la requête à une autre ressource du serveur (servlet, jsp...)
- Inclure la réponse d'un autre programme
- **La requête peut être transférée**
 - `requestDispatcher.forward(request, response)`
 - Une utilisation classique consiste à avoir un servlet contrôleur qui transmet les commandes à des servlets spécialisés (jsp)
 - Le contrôle ne revient jamais au servlet
- **La réponse peut être incluse dans la réponse en cours de construction**
 - `requestDispatcher.include(request, response)`

HTTP Servlet

RequestDispatcher



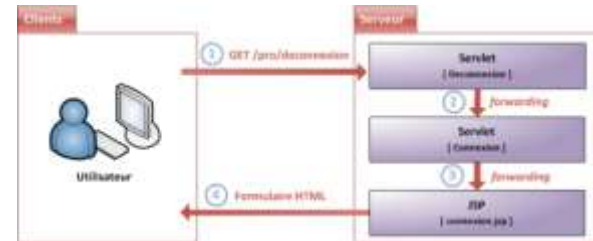
```
@WebServlet(  
    name="basicServlet",  
    urlPatterns = "/info/*"  
)  
  
public class BasicController extends HttpServlet{  
  
    @Override  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,  
    IOException {  
  
        //récupération de l'objet requestDispatcher pour transfert sur une autre servlet  
        RequestDispatcher requestDsipatcher = request.getRequestDispatcher("/userServlet");  
  
        //Transfert de la suite du traitement à la servlet suivante  
        dispatcher.forward(request,response);  
    }  
}
```

HTTP Servlet

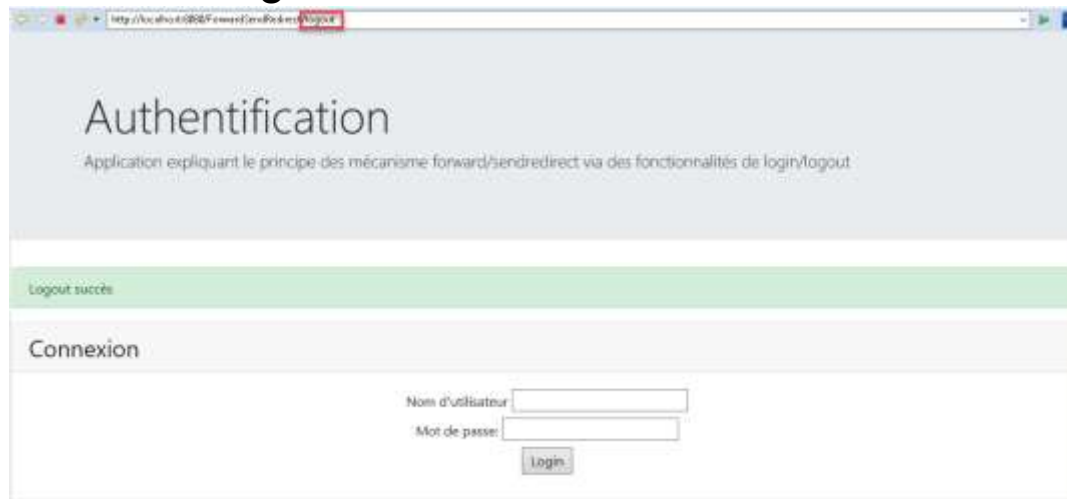
Forward vs sendRedirect



- **requestDispatcher.forward**
 - Exemple avec fonctionnalité de login/logout



- Problème: le rechargement de l'url relance le traitement de logout



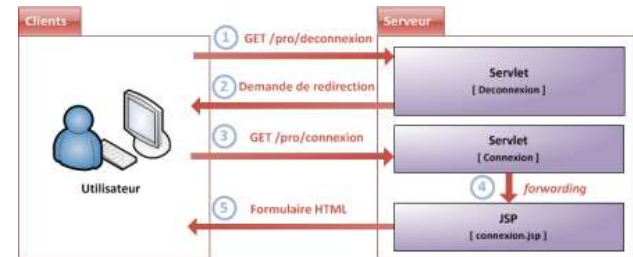
HTTP Servlet

Forward vs sendRedirect



■ `response.sendRedirect`

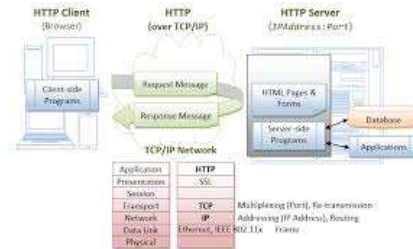
- Exemple avec fonctionnalité de login/logout



- Logout → Url propre peut être relancé sans impact (idempotence)

The screenshot shows a web browser window with the URL <http://localhost:8080/ForwardAndRedirect/>. The page title is "Authentification" and the subtitle is "Application expliquant le principe des mécanisme forward/sendredirect via des fonctionnalités de login/logout". Below the title, there is a green bar indicating "Logout réussi". Underneath, there is a section titled "Connexion" with a form containing two input fields: "Nom d'utilisateur" and "Mot de passe", and a "Login" button.

Java Server Page - JSP



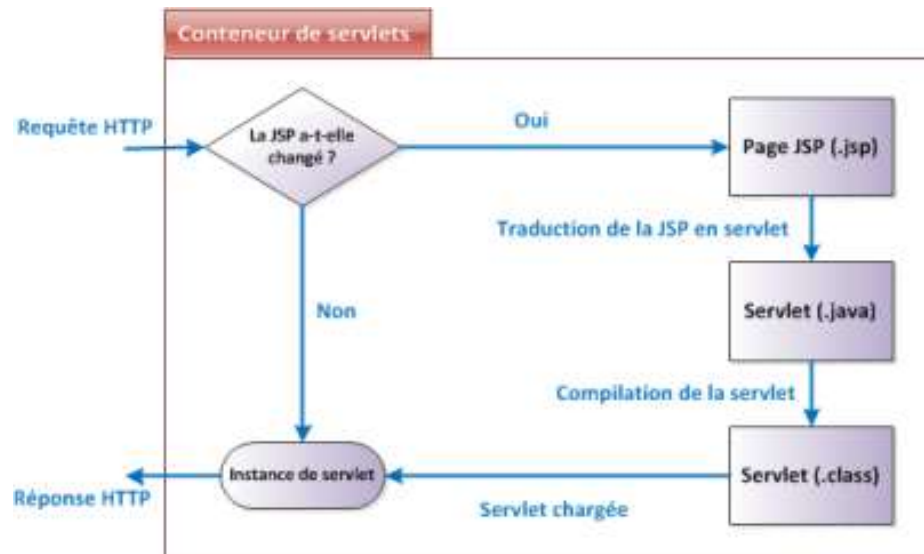
Servlet

Servlet vs Java Server Page (JSP)



Servlet

- Cinématique globale
- Routage et transfert de de traitement
- Pas adapté pour la construction de vue (html)



Servlet

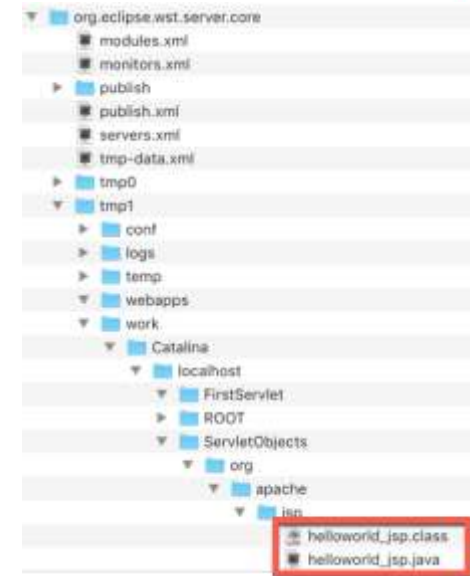
Servlet vs Java Server Page (JSP)



JSP

- Page html...
- Extension .jsp
- Contient des balises web standards et des balises Java (scriptlet)
- Compilé par le serveur en servlet
- Du coup, une *jsp* fournit les objets hérités de *HttpServlet*

→ *Rechercher dans le répertoire work de Tomcat, dans workspace les pages jsp compilés*



Java Server Page

objets à disposition



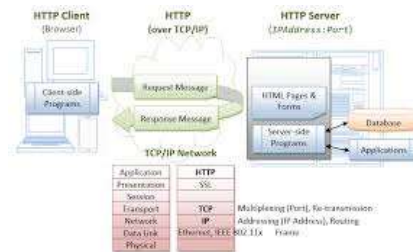
```
<!DOCTYPE html>
...
<!-- scriptlet afin d'importer une dépendance java -->
<%@ page import="java.util.List" %>
...
<%
String loginFailed = (String)request.getAttribute("login-failed");

if(null != loginFailed){
%>
<div class="alert alert-danger" role="alert">
    Authentification échouée
</div>
<%
}
%>
```

■ Objets à dispositions (héritée de *HttpServlet*)

```
final javax.servlet.jsp.PageContext pageContext;
javax.servlet.http.HttpSession session = null;
final javax.servlet.ServletContext application;
final javax.servlet.ServletConfig config;
javax.servlet.jsp.JspWriter out = null;
final java.lang.Object page = this;
javax.servlet.jsp.JspWriter _jspx_out = null;
javax.servlet.jsp.PageContext _jspx_page_context = null;
```

HTTPSession et Cookies



HTTP Servlet

HttpSession



- Comment maintenir l'état d'un client au cours d'une série de requêtes d'un même utilisateur pendant un temps donné ?
- HTTP est stateless
- Comment identifier le client ?
- Où mettre les données de la session ?

HTTP Servlet

HttpSession



- C'est le paramètre *request* qui maintient les informations sur la session
- Les méthodes sont
 - ***request.getSession(boolean flag)***
 - flag=true : retourne l'objet session courant ou en crée un s'il n'y en a pas.
 - flag=false : Retourne l'objet session courant ou null s'il n'y en a pas
 - ***isRequestedSessionIdValid()***
 - Vrai si l'id de la session est valide dans le contexte courant
 - ***isRequestedSessionIdFromCookie()***
 - Vrai si l'id de la session vient d'un cookie
 - ***isRequestedSessionIdFromURL()***
 - Vrai si l'id de la session vient d'un URL

HTTP Servlet

HttpSession



- La session est matérialisé par l'interface **HttpSession**
- Il est donc possible de l'implémenter afin de disposer d'un objet session «custom»

Method Summary	
<code>java.lang.Object</code>	<code>getAttribute(java.lang.String name)</code> Returns the object bound with the specified name in this session, or <code>null</code> if no object is bound under the name.
<code>java.util.Enumeration<java.lang.String></code>	<code>getAttributeNames()</code> Returns an Enumeration of String objects containing the names of all the objects bound to this session.
<code>long</code>	<code>getCreationTime()</code> Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
<code>java.lang.String</code>	<code>getId()</code> Returns a string containing the unique identifier assigned to this session.
<code>long</code>	<code>getLastAccessedTime()</code> Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked the time the container received the request.
<code>int</code>	<code>getMaxInactiveInterval()</code> Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
<code>ServletContext</code>	<code>getServletContext()</code> Returns the ServletContext to which this session belongs.
<code>HttpSessionContext</code>	<code>getSessionContext()</code> <i>Deprecated. As of Version 2.1, this method is deprecated and has no replacement. It will be removed in a future version of the Java Servlet API.</i>
<code>java.lang.Object</code>	<code>getValue(java.lang.String name)</code> <i>Deprecated. As of Version 2.2, this method is replaced by <code>getAttribute(java.lang.String)</code>.</i>
<code>java.lang.String[]</code>	<code>getValueNames()</code> <i>Deprecated. As of Version 2.2, this method is replaced by <code>getAttributeNames()</code>.</i>
<code>void</code>	<code>invalidate()</code> Invalidates this session then unbinds any objects bound to it.
<code>boolean</code>	<code>isNew()</code> Returns <code>true</code> if the client does not yet know about the session or if the client chooses not to join the session.
<code>void</code>	<code>putValue(java.lang.String name, java.lang.Object value)</code> <i>Deprecated. As of Version 2.2, this method is replaced by <code>setAttribute(java.lang.String, java.lang.Object)</code>.</i>
<code>void</code>	<code>removeAttribute(java.lang.String name)</code> Removes the object bound with the specified name from this session.
<code>void</code>	<code>removeValue(java.lang.String name)</code> <i>Deprecated. As of Version 2.2, this method is replaced by <code>removeAttribute(java.lang.String)</code>.</i>
<code>void</code>	<code>setAttribute(java.lang.String name, java.lang.Object value)</code> Binds an object to this session, using the name specified.

HTTP Servlet

HttpSession – maintient de la session



- La session est maintenue, suivant les contextes, de plusieurs manières différentes:

- **URL rewriting**

- Utilisé par les navigateurs ne supportant pas les cookies
- Ajout de données supplémentaires à l'url par le serveur

- **Hidden Form Fields**

- Méthode «old school» du champ caché
- Similaire à URL Rewriting

- **Cookies**

- La plus utilisée
- Utilisation de localStorage ou sessionStorage possible

- **Session SSL (Secure Socket Layer)**

- Session maintenue à travers la session SSL
- Pour les navigateurs supportant SSL

Name	Value	Domain	Path	Expires	Size
JSESSIONID	D75767BCAA7B9048EE041800FA3454B3	localhost	/Servlet	1999-1...	

HTTP Servlet

HttpSession – maintient de la session



```
//on vide la session de l'attribut logout-ok
request.getSession().removeAttribute("logout-ok");

Utilisateur loginUtilisateur = new
Utilisateur(req.getParameter("username"),req.getParameter("pass"));

Optional<Utilisateur> utilisateur = udao.authentifieUtilisateur(loginUtilisateur);

if(utilisateur.isPresent()) {

    //définit l'attribut userLogged dans la session
    request.getSession().setAttribute("userLogged", utilisateur.get());
    req.getRequestDispatcher("authenticated.jsp").forward(req, resp);
}
```


HTTP Servlet

Cookies



- **Cookies = Témoins de connexion**
- défini par le protocole de communication *HTTP* comme étant une suite d'informations envoyée par un serveur *HTTP* à un client *HTTP*, que ce dernier retourne lors de chaque interrogation du même serveur *HTTP* sous certaines conditions.
- Les cookies peuvent être utilisés pour maintenir les données relatives à l'utilisateur durant sa navigation, mais aussi à travers plusieurs visites
- **Max 20 cookies par site/utilisateur, 300 cookies au total**
- Le navigateur envoie les cookies au serveur via les requêtes *HTTP* (dans le header)

HTTP Servlet

Cookies

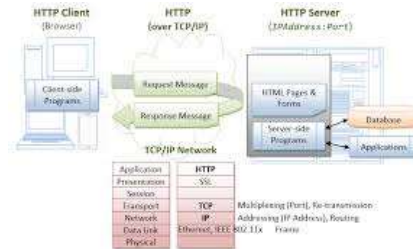


- Un cookie a un nom, une valeur, des attributs : comment, path et domain, un âge maximum, et un numéro de version.
- Une Servlet envoie les cookies au navigateur à l'aide de la méthode : `HttpServletResponse.addCookie(javax.servlet.http.Cookie)`
- Cette méthode ajoute les cookies à la réponse,
- Les Cookies sont récupérées à l'aide de la méthode `HttpServletRequest.getCookies()`.
- Plusieurs cookies peuvent avoir le même nom mais un "path" différent.

```
Cookie cookie = new Cookie("monCookie", "cookie-ok.");  
cookie.setMaxAge(100 * 1000 * 60); //100 secondes  
resp.addCookie(cookie);
```

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP
monCookie	cookie-ok.	localhost	/Cookies	2018-12-06T20:43:52.446Z	19	

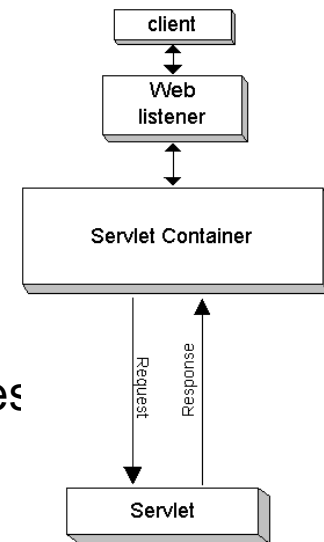
Listeners et filtres



Servlet Listeners



- Les listeners sont des objets dont les méthodes sont à l'écoute et sont invoquées en fonction du cycle de vie d'un servlet ou d'une application
- A la création et à la destruction d'un contexte (une appli)
 - **Javax.servlet.ServletContextListener**
- Quand on modifie les attributs du context
 - **Javax.servlet.ServletContextAttributeListener**
- A la création, la suppression, le timeout d'une session
 - **Javax.servlet.HttpSessionListener**
- A la création, modification, suppression d'un attribut de ses
 - **Javax.servlet.HttpSessionAttributeListener**



Servlet

Listeners – utilisations, besoins ?



- Pour une application Web:
 - En l'absence d'un point d'entrée (main) de l'appli, les listener permettent d'effectuer des opérations d'initialisation au démarrage de l'application
 - Exemple: Initialisation d'une connexion à la base de données lors du démarrage et les fermetures de ces connexions lors de «shut down» de l'appli
- Pour une session
 - Initialisation de la session
- Aspects Asynchrones

Servlet Listeners – types



Create Listener

Select the application lifecycle events to listen to.

Servlet context events	
<input type="checkbox"/> Lifecycle	? javax.servlet.ServletContextListener
<input type="checkbox"/> Changes to attributes	? javax.servlet.ServletContextAttributeListener
HTTP session events	
<input type="checkbox"/> Lifecycle	? javax.servlet.http.HttpSessionListener
<input type="checkbox"/> Changes to attributes	? javax.servlet.http.HttpSessionAttributeListener
<input type="checkbox"/> Session migration	? javax.servlet.http.HttpSessionActivationListener
<input type="checkbox"/> Object binding	? javax.servlet.http.HttpSessionBindingListener
Servlet request events	
<input type="checkbox"/> Lifecycle	? javax.servlet.ServletRequestListener
<input type="checkbox"/> Changes to attributes	? javax.servlet.ServletRequestAttributeListener

[?](#)

Servlet

Exemple : *ServletRequestListener*



```
@WebListener
public class TestListener implements ServletRequestListener {

    public TestListener() {
        System.out.println("Listener instancié");
    }

    public void requestDestroyed(ServletRequestEvent sre) {
        System.out.println("Listener détruit");
    }

    public void requestInitialized(ServletRequestEvent sre) {
        System.out.println("Listener initialisé");
    }

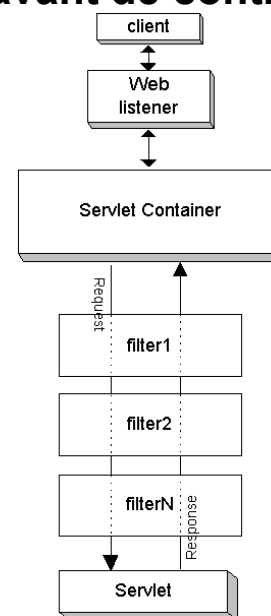
}
```

Servlet

Filtres



- Un filtre est un morceau de code exécuté entre la requête et le « endpoint » (Servlet)
- Permettent de faire du **pre et post-processing** sur une requête
 - Lire la requête, modifier la requête, modifier la réponse, retourner des erreurs au client
 - Exemple: vérifier que l'utilisateur est authentifié avant de continuer
- Même cycle de vie qu'un servlet
 - Init / doFilter / destroy



Servlet Filtres



```
@WebFilter(urlPatterns="",servletNames="mainServlet") //répond à l'url / (root) de mainServlet
public class TestFilter implements Filter {

    public TestFilter() {
    }

    public void destroy() {
    }

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

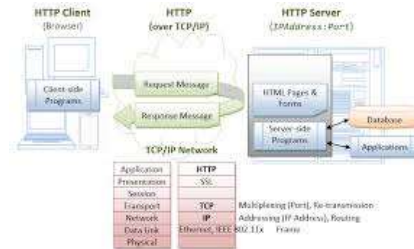
        request.setAttribute("filter", "ajoute par TestFilter");

        chain.doFilter(request, response); //ne pas oublier sous peine de casser le chaînage
    }

    public void init(FilterConfig fConfig) throws ServletException {
    }

}
```

Connection DB avec JEE



JEE

Connexion DB



- Il est toujours recommandé d'utiliser les **Connection Pool** pour accéder à une base de données à partir d'une application Web
- Néanmoins pour des exercices simples, il est possible de créer des connexion directement à la base et exécuter des requêtes SQL à partir d'un Servlet par exemple
- Procédure:
 - **Instancier le driver JDBC correspondant à la DB:**
`Class.forName("com.mysql.jdbc.Driver").newInstance();`
 - **Créer une connexion :**
`Connection conn = DriverManager.getConnection(url, userName, password);`
`String url = "jdbc:mysql://localhost:3306/world";`
 - **Préparer les statements:**
`String url = "jdbc:mysql://localhost:3306/world";`
`PreparedStatement ps = conn.prepareStatement("select * from country where Name = ?")`
 - **Exécuter la requête**
`ResultSet rs = ps.executeQuery();`

JEE

Connexion DB



```
private static Connection getConnection() {  
  
    Connection connection = null;  
  
    try {  
        Class.forName("com.mysql.jdbc.Driver").newInstance();  
    } catch (InstantiationException | IllegalAccessException | ClassNotFoundException e) {  
        e.printStackTrace();  
    }  
  
    try {  
        connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
  
    return connection;  
  
}
```

JEE

Connexion DB



```
public List<Tache> getAllTaches(){  
  
    List<Tache> taches = new ArrayList<>();  
  
    try {  
        PreparedStatement preparedStatement = getConnection()  
            .prepareStatement("select * from taches");  
  
        ResultSet resultSet = preparedStatement.executeQuery();  
  
        while(resultSet.next()) {  
            String description = resultSet.getString("description");  
            String dueDate = resultSet.getString("due_date");  
            taches.add(new Tache(description,dueDate));  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
  
    return taches;  
}
```

JEE










Connexion DB



```
public void saveTaches(Tache tache){  
  
    try {  
  
        String insertTableSQL = "INSERT INTO taches (description, due_date) VALUES (?,?)";  
  
        PreparedStatement preparedStatement = getConnection().prepareStatement(insertTableSQL);  
        preparedStatement.setString(1,tache.getDescription());  
        preparedStatement.setString(2, tache.getDueDate());  
  
        preparedStatement .executeUpdate();  
  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Connexion DB

- Ne pas oublier de fournir la librairie jdbc au container web

- >  JavaScript Resources
- >  Referenced Libraries
- >  > build
- ▼  > WebContent
 - >  META-INF
 - ▼  > WEB-INF
 - ▼  > lib
 -  mysql-connector-java-5.1.47.jar
 -  NewFile.html

→ WEB-INF→lib