



المدرسة العليا للتكنولوجيا - الصويرة  
ⵜⴰⵎⴰⵔⵜ ⵜⴰⵎⴰⵏⴰⵢⵜ ⵜⴰⵔⵉⵎⴰⵙⵜ ⵜⴰⵖⴰⵏⴰⵏⵜ  
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE - ESSAOUIRA

# Java Enterprise Edition

---

## Les Servlets

---

Licence Professionnelle Ingénierie des Systèmes Informatiques et Logiciels

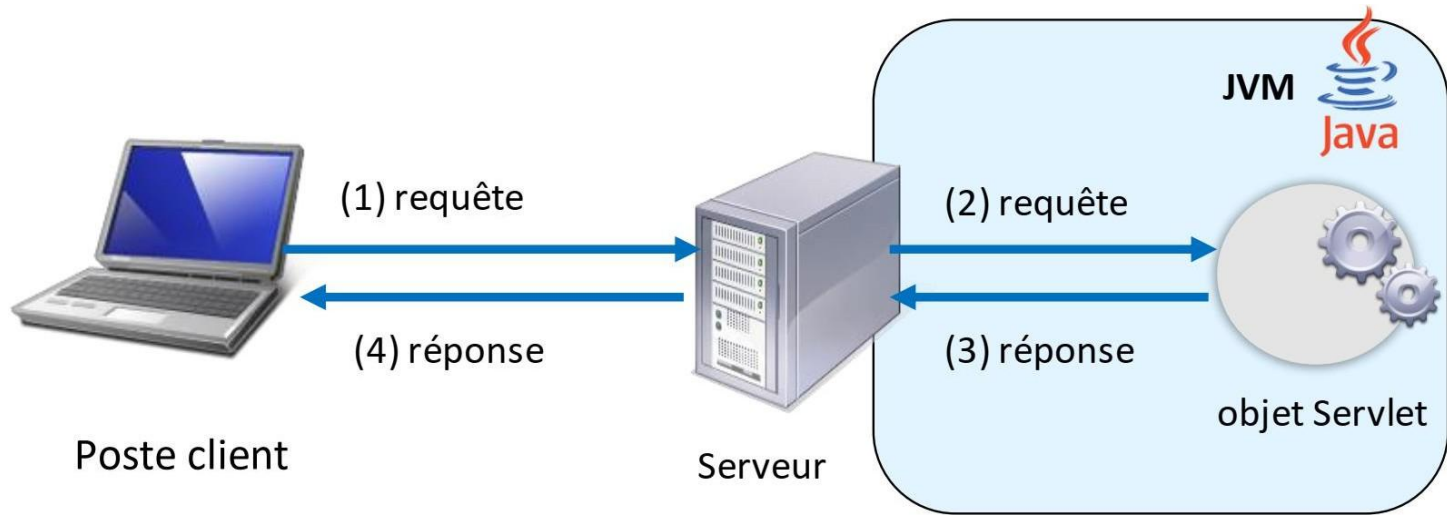
# Introduction

---

- Une servlet est un « objet » Java (classe dérivée de **javax.servlet.GenericServlet**) localisé « côté serveur » qui fournit un service en réponse aux sollicitations de différents « clients ».
- C'est un composant serveur spécialisé fonctionnant dans un « **Web Container** » Java EE.
- Les servlets généralement de type HTTP (classe dérivée de **javax.servlet.http.HttpServlet**) sont utilisées pour étendre des applications hébergées par un serveur web <requêtes HTTP>.
- Quelques avantages :
  - *Portabilité (java),*
  - *Puissance avec l'accès à la totalité de l'API java (accès aux Java Beans, bases de données, annuaires, web services, fonctionnalités réseau, etc ... ),*
  - *Rapidité (la Servlet est chargée une seule fois).*

# Modèle de fonctionnement

---



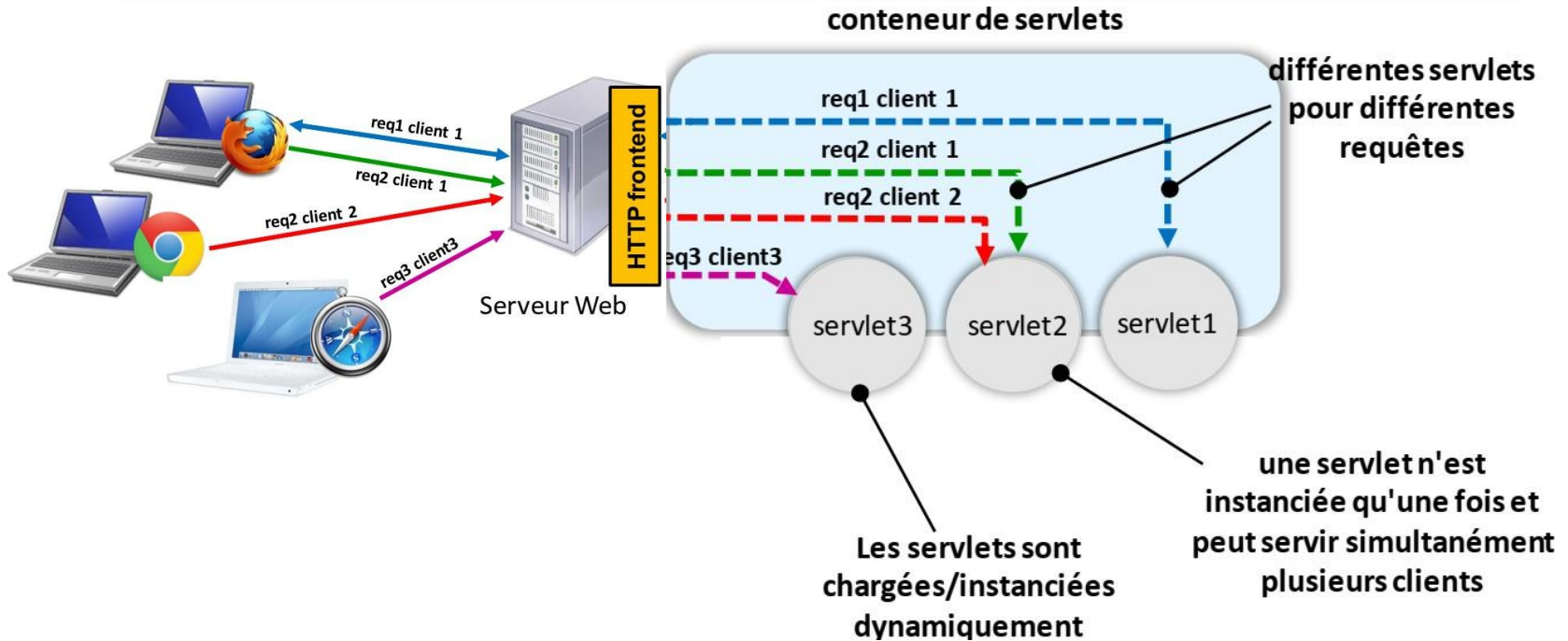
# Conteneur de servlets

---

- JEE propose des classes de servlets spécifiques pour le protocole HTTP.
- Les servlets s'exécutent dans des serveurs dédiés (**Servlets Containers**).
- *Le conteneur de servlets gère les échanges avec le client (**protocole HTTP**) :*
  - *Aiguille les requêtes vers les servlets,*
  - *Charge/décharge les servlets,*
  - *Les servlets sont instanciées une seule fois, ensuite le traitement des requêtes s'effectue de manière concurrente dans des fils d'exécution (threads) différents.*
- Différents types de conteneur de servlets :
  - *Conteneurs légers : Jetty, Resin, Apache Tomcat,*
  - *Conteneurs JEE complets : Glassfish, JBoss, WebSphere....*

# Modèle de fonctionnement

# selon HTTP



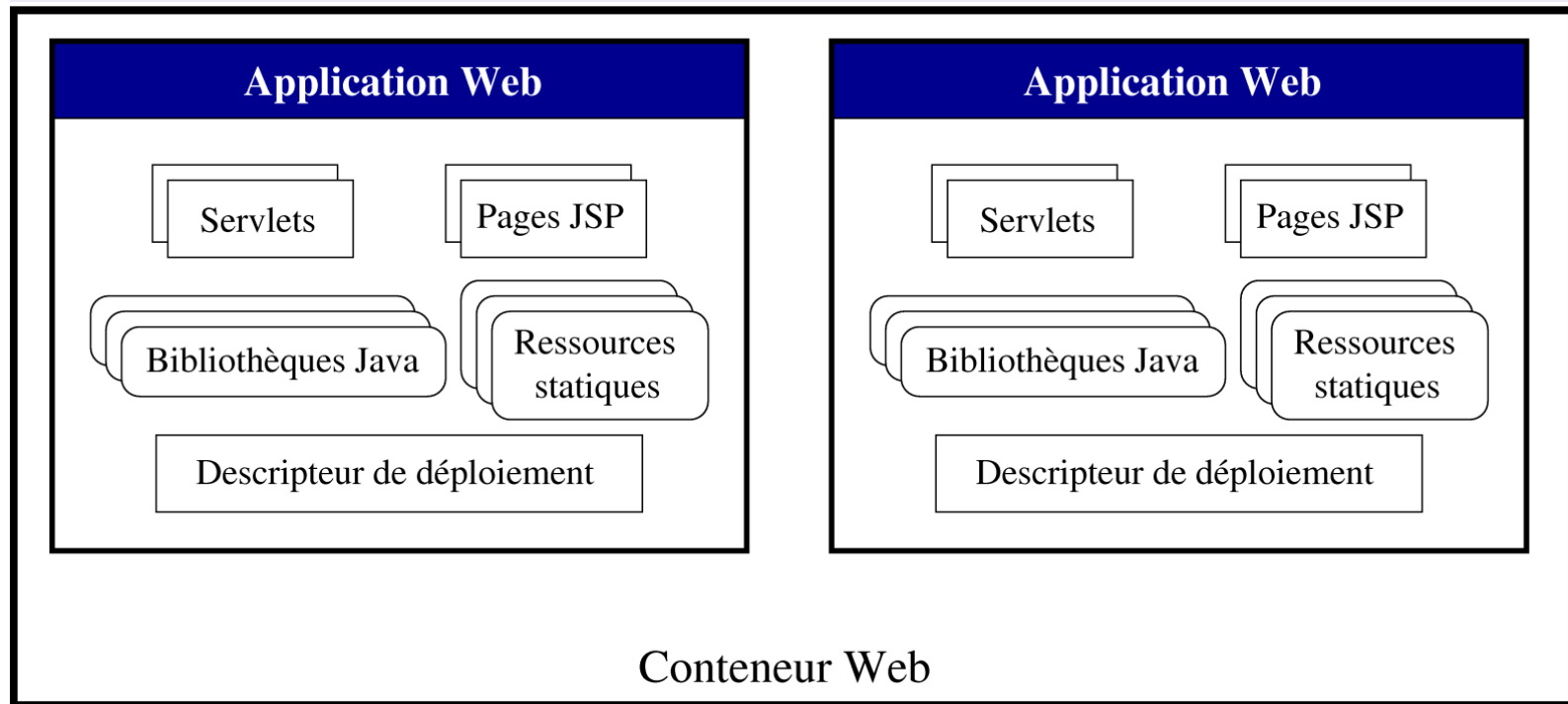
# Versions de l'API Servlet

---

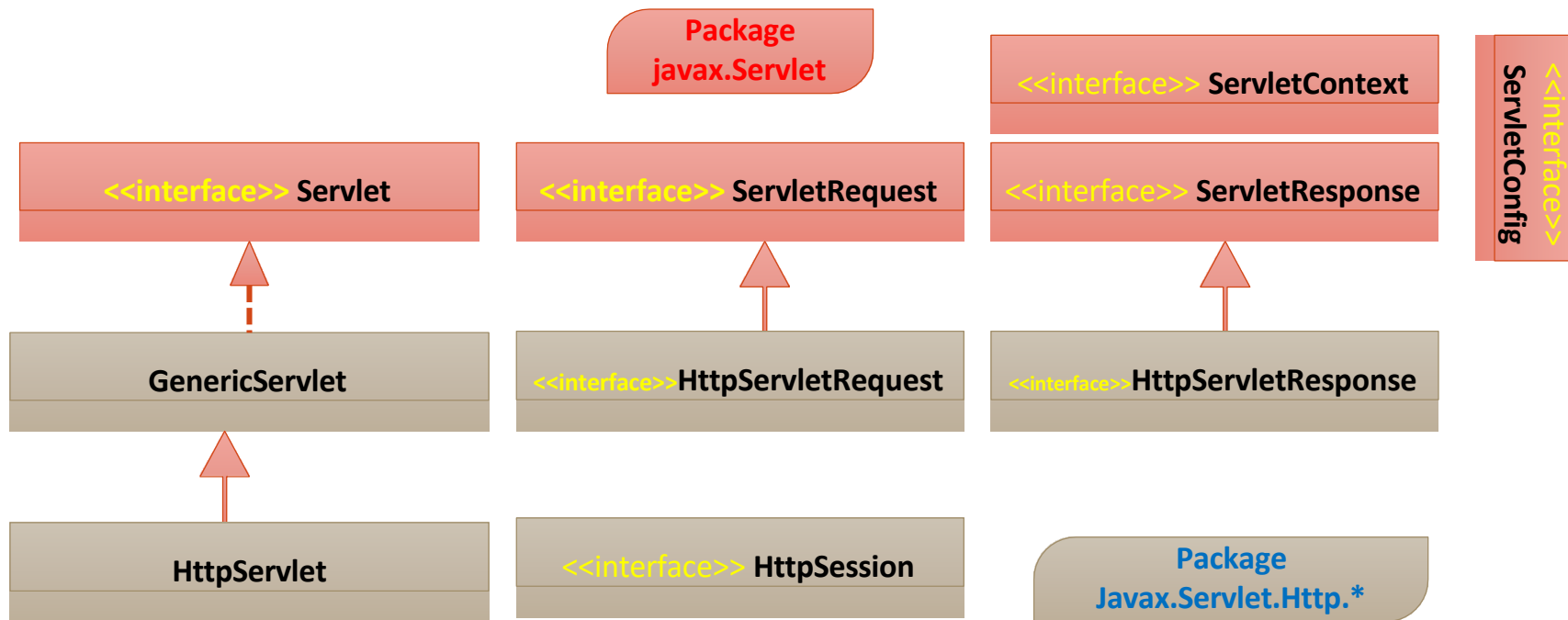
Version	Date de sortie	Plateforme
Servlet 4.0	Courant 2017	JavaEE 8 (fin 2017)
Servlet 3.1	Mai 2013	JavaEE 7
Servlet 3.0	Décembre 2009	JavaEE 6, JavaSE 6
Servlet 2.5	Septembre 2005	JavaEE 5, JavaSE 5
Servlet 2.4	Novembre 2003	J2EE 1.4, J2SE 1.3
Servlet 2.3	Aout 2001	J2EE 1.3, J2SE 1.2
Servlet 2.2	Aout 1999	J2EE 1.2, J2SE 1.2
Servlet 2.1	Novembre 1998	--
Servlet 2.0	--	--
Servlet 1.0	Juin 1997	--

Documentation API Java EE 7 : <http://docs.oracle.com/javaee/7/api/>

# Application Web



# API Servlet





# API Servlet : L'interface `javax.servlet.Servlet`

---

- Il s'agit d'une interface qui sert de corps pour recevoir et répondre aux requêtes des clients.
- Une servlet est exécutée au sein d'un conteneur de servlets.
- L'interface servlet définit les méthodes pour initialiser la servlet, recevoir et répondre aux requêtes des clients et pour détruire aussi la servlet.

## Servlet

```
+init():void  
+destroy():void  
+service(req: ServletRequest, rep: ServletResponse)  
. . .
```

# API Servlet : javax.servlet.GenericServlet

---

- Implémentation de l'interface Servlet définissant une servlet indépendante de tout protocole.
- Généralement, les développeurs de servlets utilisent des classes qui héritent de **GenericServlet** ou bien de ses classes descendantes comme **HttpServlet**.

## GenericServlet

```
+getServletInfo() : String  
+getServletContext : ServletContext  
+getServletConfig() : ServletConfig  
... 
```

# API Servlet : javax.servlet.**ServletRequest**

---

- Il définit un objet qui fournit les informations de la requête du client à la servlet.
- Le conteneur crée un objet **ServletRequest** et le passe comme argument à la méthode « **service ()** » de la Servlet.

## **ServletRequest**

```
+getAttribute(name : String) : Object  
+setAttribute(name : String, o:Object) : void  
.  
.  
.
```

# API Servlet : javax.servlet.**ServletResponse**

---

- Définition d'un objet qui contient la réponse renvoyée par la servlet au client.

## **ServletResponse**

```
+setContentType(type : String) : void  
+getWriter : PrintWriter  
... 
```

# API Servlet : javax.servlet.**ServletContext**

- Définit les méthodes qu'utilise une servlet pour communiquer avec le conteneur de servlets. Par exemple, pour obtenir le type MIME, pour rediriger les requêtes, pour écrire dans des fichiers de logs.
- Il y a un **seul contexte** par web application par JVM.
- L'objet **ServletContext** est contenu dans l'objet **ServletConfig**. Ce dernier est *fourni à la servlet lors dans son initialisation par le serveur web*.

## ServletContext

```
+getContextPath() : String  
+getAttribute(name : String) : Object  
+setAttribute(name : String, o:Object) : void  
... 
```

# API Servlet : javax.servlet.**ServletConfig**

---

- Objet de configuration de servlet qui est utilisé par le conteneur de servlet pour passer des informations à la servlet lors de son initialisation.

## **ServletConfig**

```
+ getServletName() : String  
+ getServletContext() : ServletContext  
+ getInitParameter(name: String) : String  
+ getInitParameterNames() : Enumeration  
... 
```

# API Servlet : javax.servlet.http.HttpServlet

---

- Hérite de **GenericServlet** : classe définissant une servlet utilisant le protocole http.
- Une sous classe de **HttpServlet** doit au moins réécrire une des méthodes suivantes: doGet, doPost, doPut, doDelete, ...

## HttpServlet

```
+service(req: ServletRequest, rep: ServletResponse)
+service(req: HttpServletRequest, rep: HttpServletResponse)
+doGet(req: HttpServletRequest, rep: HttpServletResponse)
+doPost(req: HttpServletRequest, rep: HttpServletResponse)
+doPut(req: HttpServletRequest, rep: HttpServletResponse)
+doDelete(req: HttpServletRequest, rep: HttpServletResponse)
. . .
```

# API Servlet : javax.servlet.http.HttpServletRequest

---

- Hérite de **ServletRequest** : définit un objet contenant une requête selon le protocole http.

HttpServletRequest

+getHeader(name:String):String

+getSession():HttpSession

...



# API Servlet :

## `javax.servlet.http.HttpServletResponse`

---

- Hérite de **ServletResponse** : définit un objet contenant la réponse de la servlet selon le protocole HTTP.
- Elle dispose par exemple de méthodes permettant d'accéder aux entête HTTP et aux cookies.

### HttpServletResponse

```
+addCookie(c: Cookie):void  
+addHeader(name, value : String):void  
...
```

# API Servlet : javax.servlet.http.HttpSession

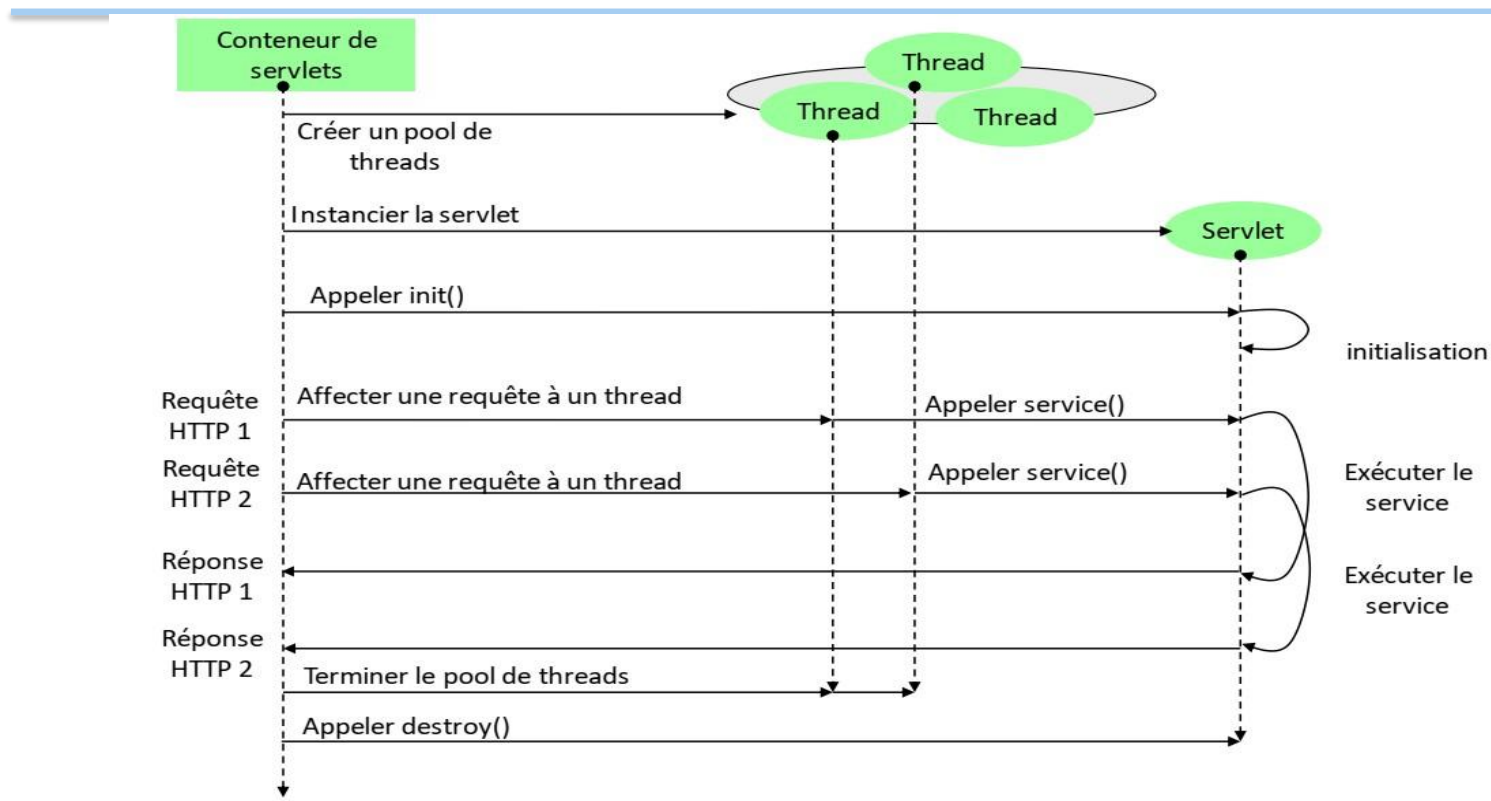
---

- Définit un objet qui représente une session utilisateur.

HttpSession

```
+getAttribute(name:String):Object  
+setAttribute(name:String, o:Object):void  
...
```

# Cycle de vie d'une servlet



# Cycle de vie d'une servlet

---

- Le serveur crée un pool de threads auxquels il va pouvoir affecter chaque requête.
- La servlet est chargée au démarrage du serveur ou lors de la première requête.
- La servlet est instanciée par le serveur.
- La méthode `init()` est invoquée par le conteneur.
- Lors de la première requête, le conteneur crée les objets **Request** et **Response** spécifiques à la requête.
- La méthode `service()` est appelée à chaque requête dans un nouveau thread. Les objets **Request** et **Response** lui sont passés en paramètre.
- Grâce à l'objet **Request**, la méthode `service()` va pouvoir analyser les informations en provenance du client.
- Grâce à l'objet **Response**, la méthode `service()` va fournir une réponse au client.
- La méthode `destroy()` est appelée lors du déchargement de la servlet, c'est-à-dire lorsqu'elle n'est plus requise par le serveur. La servlet est alors signalée au « garbage collector ».

# Déclaration d'une servlet

- Créer une classe qui hérite de l'interface **HttpServlet**
- Redéfinir une ou plusieurs méthodes **doXXX()**
- Ajouter une description dans le descripteur du déploiement.

```
package mypackage;
import java.io.IOException;
import ...
...

public class MyServlet extends HttpServlet {
    /**
     * Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        ...
    }

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

# Hello World Servlet

```
import java.io.IOException;

public class HelloWorldServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head> <title>Hello</title> </head> ");
        out.println(" ");
        out.println("<body>");
        out.println("<h1>Hello World!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Redéfinition de la méthode doGet traitant les requêtes HTTP GET.

Construction de la réponse à la requête en utilisant l'objet HttpServletResponse.

# Hello World Servlet

- Dans le fichier WEB-INF/web.xml

```
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>com.mypackage.HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

Le libelle de  
la classe

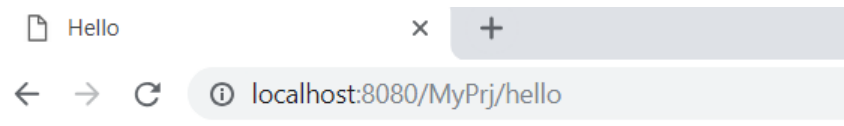
La classe Java

URI pour invoquer la  
servlet

# Scénario d'exécution

---

- Depuis un navigateur, l'on invoque l'url suivante : <http://localhost:8080/MyPrj/hello>
- Il s'agit d'un appel GET où l'on demande le document situé dans l'URL affiché ci-dessus.
- L'URI correspond à une servlet « **HelloWorldServlet** » qui sera exécutée par le conteneur Web.
- Son exécution permet le rendu d'un document HTML qui affiche le message « Hello World ».

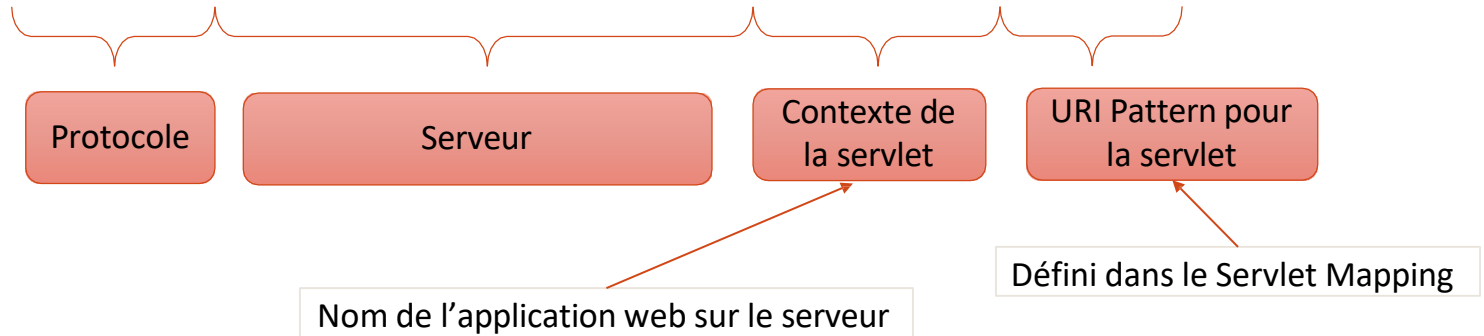


**Hello World!**



# Servlet URL

http://localhost:8080/MyPrj/hello

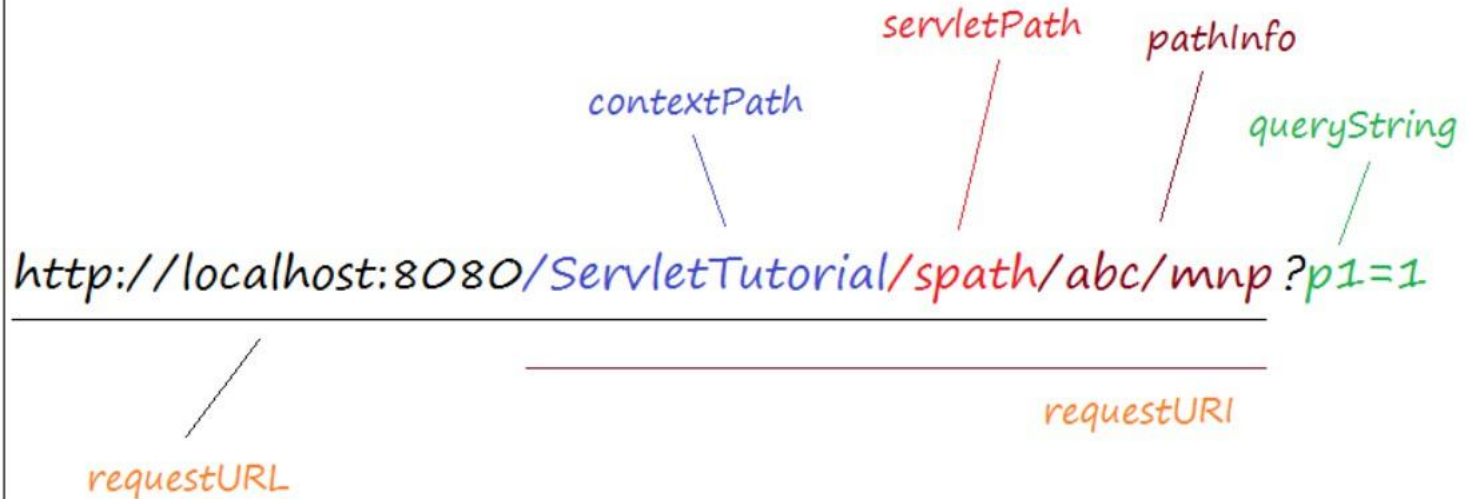


\* un serveur peut héberger plusieurs applications

\*\* une application peut être composée de plusieurs servlets

# Servlet URL Pattern

Servlet `url-pattern = /spath/*`



# Servlet URL Pattern

- Il y a 4 manières pour configurer le chemin de Servlet:

URL Pattern	Examples
/*	http://example.com/contextPath
	http://example.com/contextPath/status/abc
/status/abc/*	http://example.com/contextPath/status/abc
	http://example.com/contextPath/status/abc/mnp
	http://example.com/contextPath/status/abc/mnp?date=today
	<del>http://example.com/contextPath/test/abc/mnp</del>
*.map	http://example.com/contextPath/status/abc.map
	http://example.com/contextPath/status.map?date=today
	<del>http://example.com/contextPath/status/abc.MAP</del>
/	Đây là Servlet mặc định.

# Arborescence d'une application web

## webapp1/

- \*.html, \*.js, \*.gif, ...
- \*.jsp
- repertoire/
  - \*.html, \*.js, \*.gif, \*.jsp

ressources web  
« classiques »

## ■ WEB-INF/

- web.xml
- classes/
  - \*.class
  - package/
    - \*.class
- lib/
  - \*.jar

**WEB-INF :**  
répertoire spécial,  
inaccessible depuis  
le client http

descripteur de la webapp

classes Java compilées  
(hiérarchie de packages)

librairies nécessaires au  
fonctionnement de la webapp

L'arborescence d'une application web est le résultat  
de la décompression du fichier « .war » ( zip )

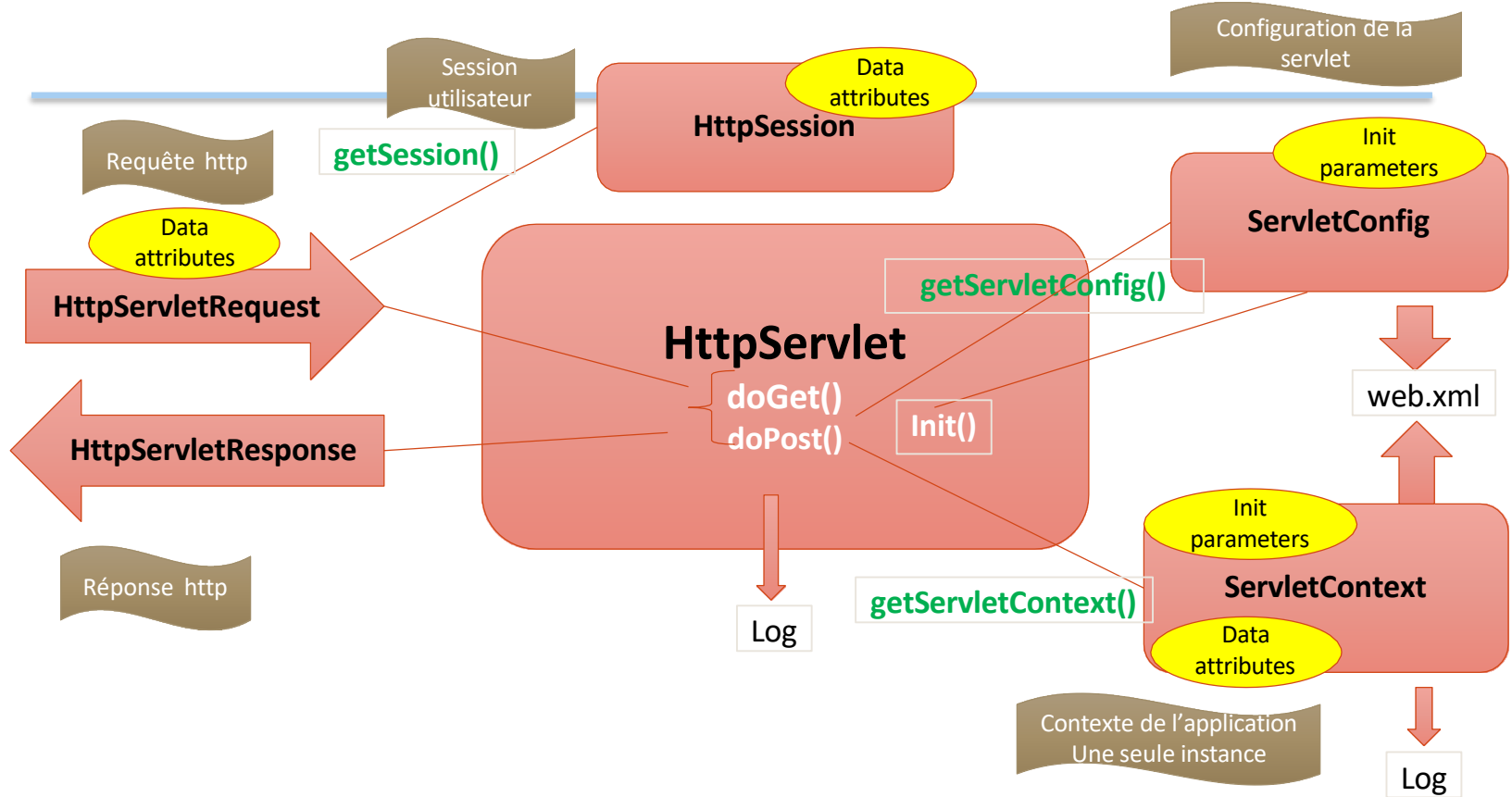
webapp2/  
etc...

2 modes de déploiement :

"**exploded**" = .war décompressé

"**non exploded**" = .war non décompressé

# Environnement d'une servlet



# Initialisation et destruction d'une servlet

```
public class ExempleServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
  
    private String message; // variable d'instance de la servlet  
  
    public ExempleServlet() {  
        super();  
    }  
    public void init() throws ServletException{  
        ServletConfig config = getServletConfig();  
        message = config.getInitParameter("msg");  
    }  
}
```

Récupération du  
paramètre à partir  
d'une ServletConfig

web.xml

```
<servlet>  
    <servlet-name>Example</servlet-name>  
    <servlet-class>com.mypackage.ExempleServlet</servlet-class>  
    <init-param>  
        <param-name>msg</param-name>  
        <param-value>Initialization Parameter</param-value>  
    </init-param>  
</servlet>  
<servlet-mapping>  
    <servlet-name>Example</servlet-name>  
    <url-pattern>/example</url-pattern>  
</servlet-mapping>
```

# Initialisation et destruction d'une servlet

```
public void init (ServletConfig config) throws ServletException {  
    super.init(config);  
    message = config.getInitParameter("msg");  
}
```

```
public void destroy() {  
    message = null;  
}
```

```
<!-- Configuration de la servlet Example -->  
<servlet>  
    <servlet-name>Example</servlet-name>  
    <servlet-class>com.mypackage.ExempleServlet</servlet-class>  
  
    <!-- Paramètre d'initialisation -->  
    <init-param>  
        <param-name>msg</param-name>  
        <param-value>Initialization Parameter</param-value>  
    </init-param>  
  
    <!-- Mapping de l'URL à la servlet -->  
    <servlet-mapping>  
        <servlet-name>Example</servlet-name>  
        <url-pattern>/example</url-pattern>  
    </servlet-mapping>
```

# Environnement

---

- Information sur la servlet
  - **getServletInfo** ()-> {auteur, version, copyright, ...}
  - **getServletName** ()-> {nom figurant dans le web.xml}
- Paramètre de configuration de la servlet
  - ServletConfig **getServletConfig**()
- Contexte d'exécution de la servlet
  - ServletContext **getServletContext** ()



# Les objets **ServletContext**

---

- Un objet **ServletContext** est partagé par toutes les servlets de l'application.
- Peut contenir des paramètres d'initialisation de l'application.
- Permet de récupérer le chemin du contexte (Context Path) de l'application nécessaire pour construire des URLs.
- Peut contenir des données accessibles par les différentes servlets.

```
context.setAttribute(name,obj);  
obj = context.getAttribute(name) ;  
enum = context.getAttributeNames();  
context.removeAttribute(name);
```

# Les objets ServletContext

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    ServletContext context = getServletContext();
    String messageContext = context.getInitParameter("ctxMsg");
    String path = request.getContextPath();
    String samePath = context.getContextPath();
}
```

C'est possible de récupérer aussi le « context path » directement à partir de l'objet ServletContext (API Servlet 2.5)

ContextPath

Ne pas confondre avec **getServletContextName()** qui renvoie le nom symbolique de la web app ( <display-name> dans le fichier "web.xml" )

# Accès aux ressources de l'application web

---

- Récupération et utilisation des fichiers contenus dans le .war (toujours accessibles même si le .war n'est pas décompressé lors du déploiement)
- Liste des ressources d'un répertoire :
  - **Set<String> paths = servletContext.getResourcePaths("/dir");**  
"/dir/index.html"  
"/dir/file.txt"  
"/dir/subdir"
- Accès à une ressource par son nom :
  - **URL url = servletContext.getResource("/dir/file");**
- Récupération d'un "InputStream" pour une ressource :
  - **InputStream is = servletContext.getResourceAsStream("/dir/file");**
- Toutes ces méthodes peuvent renvoyer **null** ( si la ressource n'existe pas).

# Accès aux ressources de l'application web

---

```
Set paths = context.getResourcePaths("/WEB-INF/");
if (paths != null) {
    for (Object item : paths) {
        out.print("<li>" + item + "</li>");
    }
}
URL url = context.getResource("/META-INF/MANIFEST.MF");
if (url != null) {
    InputStream ist = url.openStream();
}
```

# Accès aux ressources de l'application web

---

```
InputStream is = context.getResourceAsStream("/WEB-INF/conf.properties");
if (is != null) {
    Properties props = new Properties();
    try {
        props.load(is);
        String min = props.getProperty("min");
        String max = props.getProperty("max");
    } catch (IOException e) {
        System.out.println("error reading properties");
    }
}
```

# Accès au système de fichier du serveur

---

- Bien que fonctionnant dans un "conteneur", les servlets ont accès au système de fichiers.
- Les ressources (fichiers) peuvent être adressées:
  - Avec leur chemin absolu, ex : `"/tmp/dir/myfile.txt"`,
  - Avec un chemin relatif au répertoire de déploiement de la WebApp (via le **ServletContext** ).
- `String s = servletContext.getRealPath("/dir/file");`

NB : ne fonctionne que si le .war est décompressé ( mode "exploded" )  
Sinon, renvoie null

# Récupération des paramètres de la requête:

---

- Les paramètres d'une requête HTTP peuvent être fournis par :
  - « query string » : <http://xxxxxxx/ScreenTest?action=INSERT&level=n>
  - données de formulaire envoyé par « GET » ou « POST »  
`<form action="xxx" > <input ...> </form>`
- Obtenir la valeur d'un paramètre HTTP :
  - String **getParameter**(String name)
- Obtenir la liste des paramètres HTTP :
  - Enumeration **getParameterNames**()
  - Map **getParameterMap**()
  - String[] **getParameterValues**(String name)

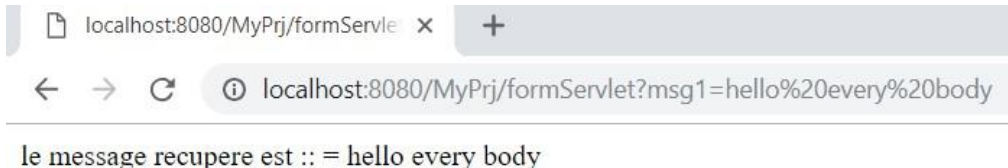
Si plusieurs paramètres avec le même nom => plusieurs valeurs

Exemples :

- . 2 champs de même nom
- . 1 liste à sélection multiple

# Exemple : soumission de l'URL suivante:

```
public class FormServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        String message = request.getParameter("msg1");  
        PrintWriter out = response.getWriter();  
        out.println("<html>");  
        out.println("<head>");  
        out.println("<body>");  
        out.println("le message recupere est :: = " + message + "<br>");  
        out.println("</body>");  
        out.println("</head>");  
        out.println("</html>");  
    }  
}
```





# Exemple 2 : Méthode POST

- Soit le document **form.html** qui sera sollicité via l'URL:
  - <http://localhost:8080/MyPrj/form.html>

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="ISO-8859-1">
5 <title>Insert title here</title>
6 </head>
7 <body>
8 <form action="/MyPrj/formServlet" method="post">
9 <label>Nom:</label>
10 <input name="nom" value="" /><br>
11 <label>Prenom:</label>
12 <input name="prenom" value="" />
13 <button type="submit">Valider</button>
14 </form>
15 </body>
16 </html>
```

Action à exécuter lors de la  
soumission du document

← → ↻ ⓘ localhost:8080/MyPrj/form.html

Nom:

Prenom:

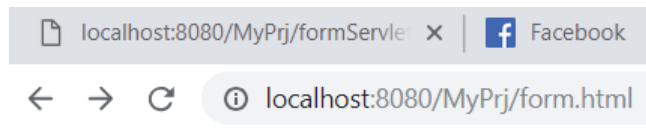
# Exemple 2 : Méthode POST

- L'exécution de la servlet dont le code de la méthode **doPOST** est le suivant :

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String nom = request.getParameter("nom");
    String prenom = request.getParameter("prenom");

    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<body>");
    out.println("votre nom est :: = " + nom + "<br>");
    out.println("votre prenom est :: = " + prenom + "<br>");
    out.println("</body>");
    out.println("</head>");
    out.println("</html>");
}
```

- Affichera ->

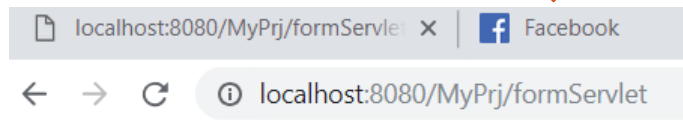
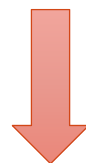


localhost:8080/MyPrj/form.html

Nom: yoyo

Prenom: saadi

Valider



localhost:8080/MyPrj/formServlet

votre nom est :: = yoyo  
votre prenom est :: = saadi

# Gestion des attributs de la requête

---

- L'objet « **Request** » dispose d'une collection d'attributs qui permet de stocker les objets utiles pour toute la durée de la requête.
- Les attributs sont utilisés lorsqu'une requête est traitée par succession de servlets (et non par une seule servlet). Par exemple dans le cas d'un « forward » vers une JSP.
- Exemple d'utilisation :
  - Création d'un attribut -> `Personne p = new Personne();`
  - Accrochage de l'objet à la requête : `request.setAttribute("prs" , p);`
  - Récupération de l'objet à partir d'une autre servlet dans le chainage de la requête initiale : `Personne p = (Personne) request.getAttribute("prs ");`

# Récupération des « headers »

---

- Méthodes générales :
  - String **getHeader**(String name)
  - Enumeration **getHeaders**(String name)
  - Enumeration **getHeaderNames**()
  - int **getIntHeader**(String name)
- Méthode spécialisées :
  - long **getDateHeader**(java.lang.String name)
- Cookie [] **getCookies**()
- etc...

# Informations " réseau " & "user"

- Adresse IP du client\* : String **getRemoteAddr()**
- Le nom de hôte du client \* : String **getRemoteHost()**
- Protocole : String **getProtocol()**-> Ex : "HTTP/1.1«
- Méthode HTTP : String **getMethod()** -> "GET", "POST", "PUT", "DELETE", etc...
- Authentification :
  - String **getRemoteUser()**

(\*) ou du dernier proxy  
ayant envoyé la requête

- Renvoie le nom de l'utilisateur si la page est protégée par mot de passe et si l'utilisateur est authentifié
- Renvoie « null » si la page n'est pas protégée par un mot de passe

# Lecture des données envoyées

- Données de type texte ( Texte, XML, ... ) :

- avec la classe « **java.io.BufferedReader** »

```
java.io.BufferedReader in = request.getReader();
String s ;
while ( ( s = in.readLine() ) != null) {
    System.out.println( s );
}
```

- Données binaires ( image, fichier, ... ) :

- avec la classe « **ServletInputStream** » dérivée de « **java.io.InputStream** »

```
javax.servlet.ServletInputStream in = request.getInputStream();
int ch ;
while( (ch = in.read()) != -1) {
    //Lecture « binaire » octet par octet
}
```

```
javax.servlet.ServletInputStream in = request.getInputStream();
byte[] buf = new byte[4 * 1024];
while ( in.read(buf) != -1 ) {
    //Lecture « binaire » avec un buffer
}
```

# Les URL et URI de la requête

- Mapping de la servlet dans le web.xml

```
<servlet-mapping>
  <servlet-name>RequestInfo</servlet-name>
  <url-pattern>/req/*</url-pattern>
</servlet-mapping>
```

- Requête http <http://myhost:8080/mywebapp/req/aaa/bbb?A=123&B=zzz>
  - request.getRequestURL() = '<http://myhost:8080/mywebapp/req/aaa/bbb>'
  - request.getRequestURI() = '/mywebapp/req/aaa/bbb'
  - request.getScheme() = 'http'
  - request.getServerName() = 'myhost'
  - request.getServerPort() = '8080'
  - request.getContextPath() = '/mywebapp' (1)
  - request.getServletPath() = '/req'
  - request.getPathInfo() = '/aaa/bbb' (2)
  - request.getQueryString() = 'A=123&B=zzz' (3)

(1) : "" (vide) si l'application est la "root application" du serveur (pas de nom de contexte)  
(2) : **null** si aucun "path" après l' "url pattern" de la servlet  
(3) : **null** si pas de "query string"

# Les objets HttpServletResponse

`response.setStatus(code)`

Code Status

Headers

```
out = response.getWriter();  
out.print(...);  
out.println(...);
```

Body

HTTP/1.1 **200 OK**

Date: Wed, 19 May 2000 13:12:34 GMT

Server: Apache/1.3.6 (Unix) PHP/3.0.7

Etag: "2da0dc-2870-374039cd"

Accept-Ranges: bytes

Content-Length: 10352

Connection: Close

Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML .....

<HTML>

<BODY>

...

</BODY>

</HTML>

Type :

« texte » (XML,CSV,..) ou

« binaire » (objets sérialisés, ...)

`response.setHeader(name, value)`  
`response.addHeader(name,value)`

`response.setContentLength(len)`

`response.setContentType(type)`

```
out = response.getOutputStream();  
out.print(...); out.println(...);  
out.write(...);
```



# Codes d'erreur

---

- Les «codes de statuts » **400 à 599** correspondent à des erreurs ou des impossibilités de traiter la requête.
- Ils peuvent être renvoyés avec la méthode « **sendError()** » :
  - void **sendError ( int sc )**
  - void **sendError ( int code, String message )** le serveur envoie le message dans une page HTML ( content type = « text/html » ).

```
response.setStatus(response.SC_NOT_FOUND);
```

Ou

```
response.sendError(response.SC_NOT_FOUND, "ressource introuvable");
```

# Rediriger une requête

---

- Le code de statut « **301** » ou « **302** » et le header « **Location** » permettent de demander au navigateur de rediriger sa requête sur une autre URL.
- Avec **setStatus()** + **setHeader()** :

ou

```
response.setStatus(response.SC_MOVED_PERMANENTLY);
```

Code 301

```
response.setStatus(response.SC_MOVED_TEMPORARILY);
```

Code 302

```
response.setHeader("Location", "http://localhost:8080/MyPrj/redirection1.html");
```

- Ou, plus simplement, avec **sendRedirect()** :

```
response.sendRedirect("http://localhost:8080/MyPrj/redirection1.html");
```

Code 302

# Gestion des cookies

---

- Cookie (témoin de connexion) : petit élément d'information envoyé depuis un serveur vers un client HTTP et que ce dernier retourne lors de chaque interrogation du même serveur HTTP sous certaines conditions.
- Les cookies sont stockée localement sur le poste client.
- Un cookie comporte (entre autres) :
  - un **nom**
  - une **valeur**
  - une **date d'expiration** (durée de vie)
  - un “**domaine**” (généralement le serveur qui l’a créé)
  - un “**chemin**” (path) indiquant pour quelle URL il est valable.

# javax.servlet.http.Cookie

- Une fois le cookie créé, son nom ne peut plus être changé (pas de méthode `setName()`).

## Cookie

```
+Cookie(String name, String value)
+getName():String
+getMaxAge():int
+getValue():String
+getDomain():String
...
+setValue(String v):void
+setMaxAge(int expiry):void
+setdomain(String domain):void
...
```

La valeur ne doit pas contenir le caractère '='

# Récupération et dépôt de cookies

---



récupérer un cookie

- *via l'objet* `HttpServletRequest`
- `Cookie[] getCookies()`

déposer un cookie

- *via l'objet* `HttpServletResponse`
- `void addCookie(Cookie c)`

# Gestion des sessions

---

- De nombreuses applications nécessitent de relier entre elles une série de requêtes issues d'un même client :
  - *ex : application de E-commerce doit sauvegarder l'état du panier du client*
- **Session** : Période de temps correspondant à navigation continue d'un utilisateur sur un site.
- HTTP étant sans état (**stateless**) c'est de la responsabilité des applications web de conserver un tel état, i.e de gérer les sessions :
  - *Identifier l'instant où un nouvel utilisateur accède à une des pages du site,*
  - *Conserver des informations sur l'utilisateur jusqu'à ce qu'il quitte le site,*
  - *Cookies  $\simeq$  variables permettant de contrôler l'exécution de l'application Web. Mais ...*
    - *Stockage côté client*
    - *Possibilité de modifier les variables côté client*
    - *DANGEREUX*
- *Les données liées aux sessions doivent rester côté serveur :*
  - *Seul l'identifiant de session transmis sous forme de cookie,*
  - *En Java utilisation de l'interface : **javax.servlet.http.HttpSession***

# *javax.servlet.http.HttpSession*

- L'objet Session est un espace de stockage permettant de référencer des objets de tout type en leur associant un identifiant de type String (principe de dictionnaire, comme « **Hashtable** »)

## HttpSession

```
+getAttribute(name:String):Object  
+getAttributeNames():Enumeration<String>  
+getId():int  
+invalidate():void  
+isNew():boolean  
+setAttribute(name:String, o:Object):void  
+setMaxInactiveInterval(interval:int):void  
...
```

# Objet HttpSession

---

- Pas de classe d'implémentation dans l'API Java EE (HttpSession est une interface)
- Classe d'implémentation dépend du conteneur web JEE (TomCat, Jetty, ....) :
  - Instanciée par le conteneur
  - Dans le code des servlets on n'utilise que l'interface
- Associé à l'objet **HttpServletRequest** qui permet de le récupérer ou de le créer :
  - logique : car identifiant de session transmis dans requête http (cookie ou paramètre de requête).



# Utilisation : création et récupération de session

---

- On demande simplement la session de l'utilisateur qui émet la requête:
  - Obtenir la session, la créer si elle n'existe pas encore :

```
HttpSession session = request.getSession(true);  
// ou request.getSession() qui est true par défaut
```

- Rechercher la session, sans la créer si elle n'existe pas :

```
HttpSession session = request.getSession(false);  
if(session != null) {  
    //ToDo  
}
```


# Utilisation : gestion des attributs de session

---

- **Societe societe = new Societe(..., ..., ... ) ;**
- Création ou récupération de session
  - **HttpSession session = request.getSession();**
- Stockage un objet :
  - **session.setAttribute("soc", societe );**
- Récupération de l'objet :
  - **Societe soc = (Societe) session.getAttribute("soc");**  
**if ( soc != null ) ... ...**
- Récupération de tous les noms d'attributs
  - **Enumeration e = session.getAttributeNames();**
- Suppression de l'objet :
  - **session.removeAttribute("soc");**

# Utilisation : durée de vie

---

- La durée de vie (temps maximum de conservation entre 2 accès) d'une session est limitée.
- Paramétrage dans le fichier « **web.xml** »  
**<session-config>**  
    **<session-timeout>60</session-timeout>**  
**</session-config>**  


En minutes
- La fin d'une session peut être ...
  - **Implicite** ( délai dépassé => « time out » ),
  - **Explicite** ( sur action utilisateur => code Java ) ☐ **session.invalidate();**

# Utilisation : informations et cycle de vie session

---

- Quel est l'identifiant de la session ?  
**String sId = session.getId();**
- Quand la session a-t-elle été créée ?  
**long t = session.getCreationTime();**
- Dernier accès à la session  
**long lastAccess = session.getLastAccessedTime();**
- Invalider la session  
**session.invalidate(); // Dé-référence les objets de la session**
- Durée de conservation maximum entre 2 accès  
**int max = session.getMaxInactiveInterval() ;**  
**session.setMaxInactiveInterval(max) ;**
- S'agit-il d'une nouvelle session ?  
**if ( session.isNew() ) { ... ... }**

# Inclusion et redirection

---

- Le traitement d'une requête HTTP par une servlet peut nécessiter l'utilisation d'autres ressources :
  - *Inclusion d'une page HTML statique,*
  - *Redirection vers une autre servlet ou page JSP.*
- Réalisé à l'aide d'un objet **javax.servlet.RequestDispatcher**
- Obtention d'un objet **RequestDispatcher**
  - *via un objet **javax.servlet.ServletContext***
  - *via un objet **javax.servlet.http.HttpServletRequest***
- **RequestDispatcher getRequestDispatcher(java.lang.String path)**
  - **path** : chaîne de caractères précisant le chemin de la ressource vers laquelle la requête doit être transférée.
  - *doit commencer par un '/' et est interprété comme relatif au contexte de l'application.*

# javax.servlet.RequestDispatcher

- **include ()**: souvent utilisé pour insérer dans la réponse des fragments de code HTML
- **forward ()**: établit une redirection vers une autre ressource pour produire la réponse.

## RequestDispatcher

```
+ forward(servletRequest req, ServletResponse rep) : void  
+ include(servletRequest req, ServletResponse rep) : void . . .
```

```
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/includes/header.html");  
dispatcher.include(request, response);
```

```
if (noServlet == 1) {  
    getServletContext().getRequestDispatcher("/demoForward1").forward(request, response);  
} else {  
    getServletContext().getRequestDispatcher("/index.html").forward(request, response);  
}
```

# Forward vs Include

- Le « **forward** » est totalement transparent pour le client HTTP (il ne peut pas savoir s'il y a eu une redirection ou non).
- Ne pas confondre avec **response.sendRedirect("...")** qui provoque un échange http avec le navigateur.
- Possibilité de réaliser plusieurs « **include** »
  - de différentes ressources,
  - d'une même ressource.
- **Exemple :**
  - **`rd1 = getServletContext().getRequestDispatcher("/debut.html");`**  
**`rd1.include(request,response);`**  
**`rd1.include(request,response); // 2eme include de la même ressource`**  
**`rd2 = getServletContext().getRequestDispatcher("/fin.html");`**  
**`rd2.include(request,response);`**

# Forward vs Include

---

- **Forward :**

- **avant** : ne pas affecter le type de contenu de la réponse  
(`setContentType( "....." ) ;`),
- **après** : ne pas utiliser le flux de sortie.

- **Include :**

- le fichier à inclure est un « **fragment** » de document à produire en sortie,
- dans le cas d'un document HTML il ne doit donc contenir que la « partie à inclure »( par exemple : pas de tags `<html>`, `<head>`, `<body>`, etc... )
- pas de `setContentType` après un « **include** »



# Stockage et transfert de données entre servlets

---

- Il est possible de transférer des objets d'une servlet à une autre. Ce transfert sera limité à la portée de l'objet sur lequel est attaché l'objet de transfert.
- On peut localiser ces objets selon trois niveaux:
  - Niveau **requête** : durant tout le chainage de la requête
  - Niveau **session** : durant toute la session de l'utilisateur
  - Niveau **application** : durant l'application et donc partagés par toutes les sessions utilisateur.

# Niveau requête

---

- Durée de vie : celle de la requête
  - Accrocher l'objet à la requête : nom d'objet -> référence sur l'objet
  - Faire suivre à une autre servlet
  - Récupérer l'objet dans la requête
- **MonObjet obj = new MonObjet("test");**  
**request.setAttribute("mon\_objet", obj);**

```
rd = getServletContext().getRequestDispatcher("/autre");  
rd.forward(request,response);
```

```
MonObjet obj = (MonObjet) request.getAttribute("mon_objet");
```

# Niveau session

---

- Durée de vie : celle de la session de l'utilisateur
- Visibilité : limitée à l'utilisateur de la session
  - Stocker l'objet dans la session : nom d'objet -> référence sur l'objet
  - Récupérer l'objet dans la session
- **MonObjet obj = new MonObjet("test");**  
.. ..  
**session.setAttribute("mon\_objet", obj);**  
**MonObjet obj = (MonObjet) session.getAttribute("mon\_objet");**

# Niveau application

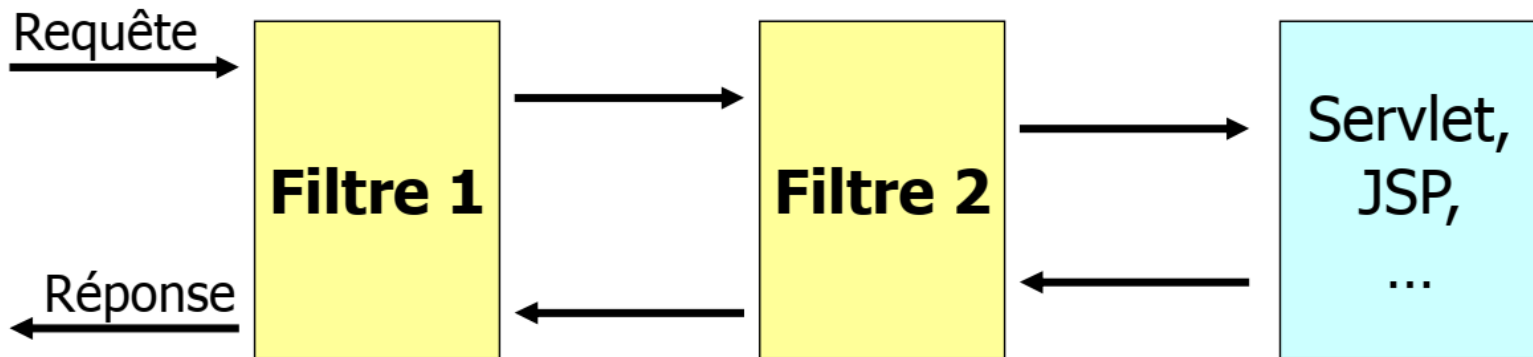
---

- Durée de vie : celle de l'application
- Visibilité : dans toutes les sessions
  - Stocker l'objet dans le contexte : nom d'objet -> référence sur l'objet
  - Récupérer l'objet dans le contexte :
- **MonObjet obj = new MonObjet("test");**  
**ServletConfig servConfig = getServletConfig();**  
**ServletContext application = servConfig.getServletContext();**  
**application.setAttribute("mon\_objet", obj);**  
**MonObjet obj = (MonObjet)application.getAttribute("mon\_objet");**

# Les filtres

---

- Ils filtrent les requêtes cherchant à accéder à certaines ressources de l'application web.
- Un filtre peut être « mappé » avec plusieurs ressources Web.
- Une ressource Web peut être « mappée » avec plusieurs filtres.



# Les filtres : mapping

```
<filter>
    <filter-name> filter1 </filter-name>
    <filter-class>package.Filter1</filter-class>
</filter>
<filter-mapping>
    <filter-name> filter1 </filter-name>
    <url-pattern>/doc/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name> filter1 </filter-name>
    <servlet-name>test</servlet-name >
</filter-mapping>
```

Classe java et non  
symbolique du filtre

Ressources filtrées

# Les filtres : implémentation

---

- Créer un filtre = créer une classe qui implémente l'interface « **javax.servlet.Filter** ».
- Méthodes :
  - **init()**,
  - **doFilter()**,
  - **destroy()**.
- La méthode "**doFilter**" est le point de passage dans le filtre. Elle dispose des paramètres :
  - **ServletRequest** ( comme les servlets )
  - **ServletResponse** ( comme les servlets )
  - **FilterChain** : instance de la "chaîne de filtrage".

# Les filtres : cas d'usage

---

- Un filtre est donc utilisable pour les **traces**, les **statistiques**, etc...
- Un filtre ( dans la méthode « **doFilter()** » ) peut bloquer l'accès à une ressource (ne pas laisser la requête atteindre la servlet ou JSP adressée).
- Pour cela, il peut :
  - Générer intégralement la réponse à la place de la ressource adressée,
  - Utiliser le **Request Dispatcher** (par exemple pour faire un "**forward**" ),
  - Faire un « **sendRedirect** »
  - etc ...
- Les filtres sont aussi utilisés pour les **contrôles d'accès** aux ressources (authentification, ... ).