



# J2EE

## Servlets et applications web

# J2EE

Composants Java EE :

- **Servlet:** Composant représentant le C ('Controller') du paradigme MVC
- **Portlet:** Conteneur Web (extension de l'API Servlet)
- **JavaServer Pages (JSP):** Framework Web
- **Java Standard Tag Library (JSTL):** bibliothèque de balises pour les JSP
- **JavaServer Faces (JSF):** Java Server Face, Framework Web
- **EJB:** Composants distribués transactionnels
- **JNDI:** API de connexion à des annuaires, notamment des annuaires LDAP, et espace de noms d'objet (ENC)
- **JDBC:** API de connexion à des bases de données
- **Java Message Service (JMS):** API de communication asynchrone par message
- **JCA:** API de connexion, notamment à des PGI
- **JavaMail:** API de gestion des mails

# J2EE

- **JMX:** Extension d'administration des applications
- **JPA:** API de gestion de la persistance des données
- **JTA:** API de gestion des transactions
- **Java API for XML Processing (JAXP):** API d'analyse XML
- **JAXM:** API de communication asynchrone par XML
- **JAX-RPC / JAX-WS:** API de communication synchrone par XML, par exemple à l'aide du protocole SOAP
- **JAXB:** API de sérialisation par XML
- **JAXR:** API de gestion des registres XML, permettant d'enregistrer des Web Services en ebXML
- **Java RMI:** API de communication distante entre des objets Java
- **Java IDL:** API de communication entre objets Java et objets non-Java, via le protocole CORBA

# J2EE: Serveurs certifiés

Une application Java EE s'exécute sur un serveur d'applications.

Sont certifié Java EE 6:

- **Oracle GlassFish Enterprise Server** v3, basé sur le serveur open-source GlassFish
- **Oracle WebLogic Server** 12c de Oracle Corporation
- **JBoss** AS 7.x
- **JEUS** 7
- **Apache Geronimo** 3.0
- **IBM WebSphere Application Server** 8.0
- **IBM WebSphere Application Server Community Edition** 3.0, basé sur Apache Geronimo
- **Fujitsu Interstage Application Server**
- **Caucho Resin** 4.0.17

# J2EE: Servlets et applications web

- Servlet Java
- La servlet HelloWorld
- Déploiement d'applications web
- Cycle de vie d'une servlet

## J2EE: Servlets et applications web

### Common Gateway Interface (CGI) (1/2)

- L'une des premières techniques (1993) pour créer du contenu dynamique.
- CGI n'est pas une application mais un standard industriel
- Le serveur Web délègue la requête à un programme CGI (processus fils du processus serveur)
- Le serveur web transmet des variables au programme CGI (variables d'environnement) lui fournissant des informations sur la requête, le client et le serveur.
- Le programme CGI crée la page résultat (html, image, ...) et la retourne au serveur.
- L'application peut être écrite dans une variété de langages : PERL, C, C++,...

## J2EE: Servlets et applications web

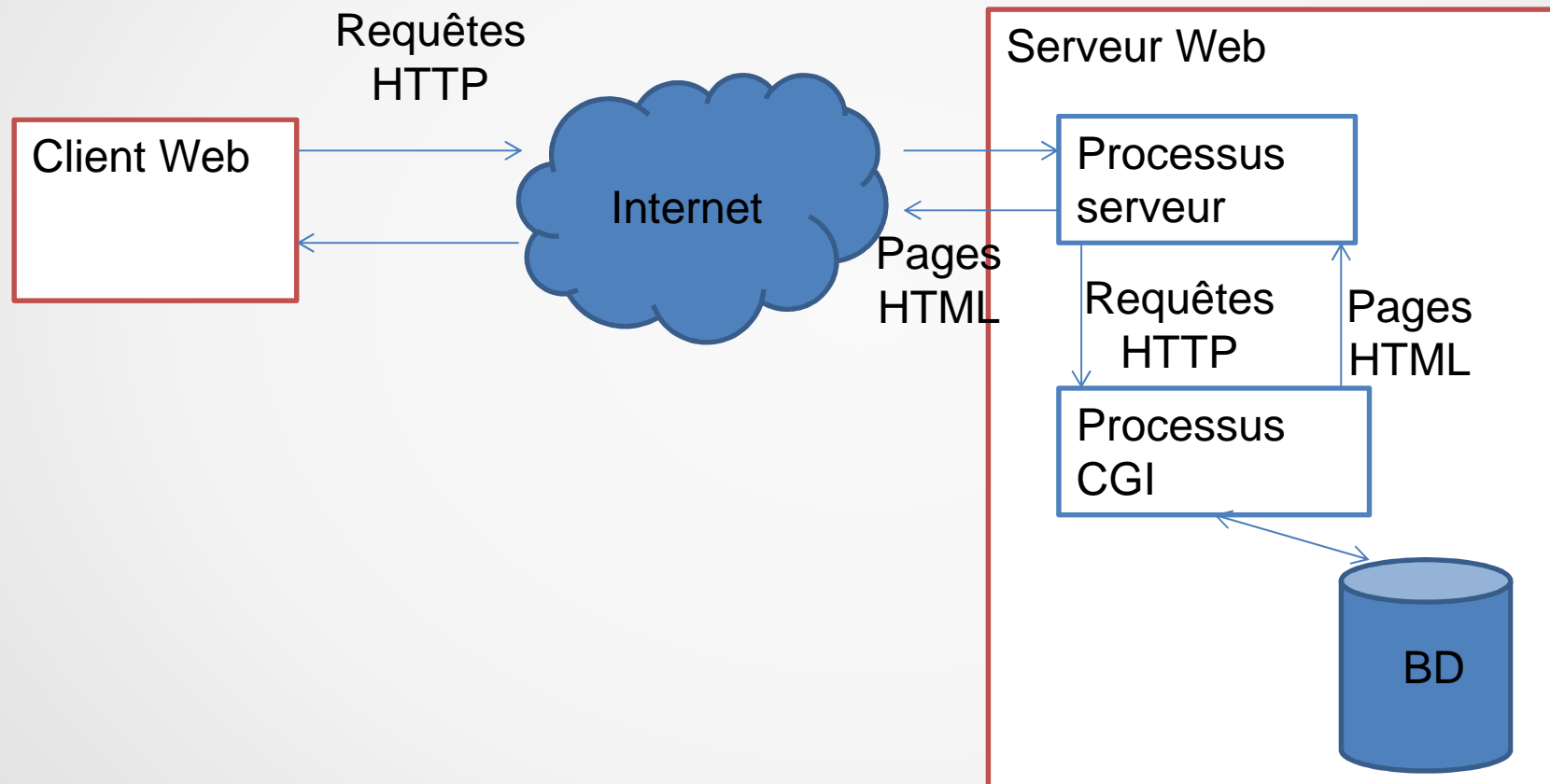
### Common Gateway Interface (CGI) (2/2)

Les Inconvénients :

- Chaque requête donne lieu à la création d'un processus nouveau pour exécuter le programme correspondant. Cela est inefficace en temps et consommateur de ressources serveur
- Ne peut interagir avec le serveur pour, par exemple, écrire dans son fichier de journalisation
- Limitation du nombre de requêtes qu'un serveur peut traiter en concurrence

# J2EE: Servlets et applications web

## Architecture CGI





## J2EE: Servlets et applications web

### Solution PHP

- PHP (Pretty Hypertext Processor), langage de scripts interprété, libre et portable
- Variante de la technique CGI
- Langage de programmation proche de C
- Dédié à la production de pages HTML générées dynamiquement
- Interpréteur PHP intégré à Apache sous la forme d'un module
- Les scripts PHP exécutés au sein d'Apache (=> pas de processus externe contrairement à CGI) produisent du code HTML qui remplacent le code PHP dans le document fourni en sortie

## J2EE: Servlets et applications web

### Scripts coté serveur

- Pour insérer du code dans les pages HTML afin de générer dynamiquement du contenu
- Les pages web peuvent ainsi être précompilées pour améliorer les performances
- Solutions propriétaires:
  - SSJS pour iPlanet ( utilise javaScript )
  - ASP (Active Server Page) de Microsoft,
  - intégrée à IIS
  - intégrée à .NET
  - permet de développer des scripts dans différents langages( utilise Jscript, VBScript,... )
- Ces solutions sont uniquement utilisables avec HTTP, ce qui n'est pas le cas des servlets

# J2EE: Servlets et applications web

## Servlets et applications Web

- Les servlets sont une technologie Java en réponse à la programmation CGI
- Les servlets sont des programmes qui s'exécutent dans une machine virtuelle Java sur un serveur Web (le plus souvent) et construisent des pages Web.
- Construire des pages Web "à la volée" est utile pour plusieurs raisons :
  - Elles dépendent des données saisies par l'utilisateur (site e-commerce, résultat d'une recherche, ...)
  - Les données changent fréquemment ( titres, éléments graphiques, ...)
  - Les données à afficher utilisent des bases de données

# J2EE: Servlets et applications web

## Servlet Java

- Elles font partie intégrante de la plate-forme J2EE
- L'API Servlet contient les paquetages `javax.servlet` et `javax.servlet.http`
- Elles n'imposent pas le support de Java sur le navigateur (contrairement aux applets)
- Elles sont portables et sûres
- Le lien bidirectionnel entre servlets et serveur permet une interaction étroite
- A chaque requête correspond un thread. Tous les threads sont gérés par le même processus

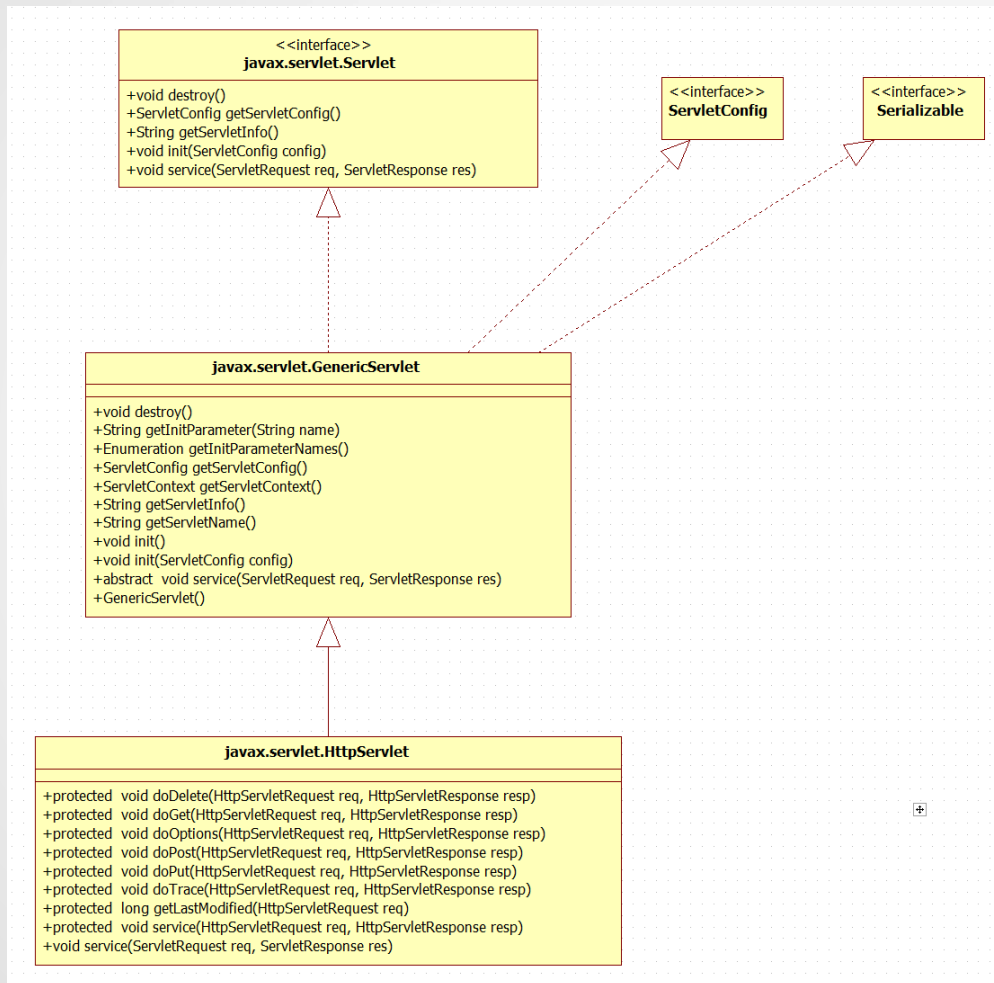
# J2EE: Servlets et applications web

## Choix des servlets

- Portabilité : écrite en Java
- Puissance : bénéficie des API noyau Java, (réseau, connectivité aux BD, invocation de méthode à distance, multithreading, ...) , de la plate-forme J2EE, de composants JavaBeans du marché, ...
- Efficacité : chargée une seule fois, c'est un objet qui reste en mémoire du serveur qui l'utilise par invocation de ses méthodes
- Sûreté (typage fort, exceptions, gestionnaire de sécurité Java)
- Orientées Objet : extensibilité des API, modularité

# J2EE: Servlets et applications web

## Diagramme UML



- pas de méthode main()
- service() invoquée par le serveur
- request et response paramètres implicites
- doGet() et doPost() invoquée par service()

# J2EE: Servlets et applications web

## Diagramme UML



# J2EE: Servlets et applications web

## Diagramme UML

### **javax.servlet.GenericServlet**

```
+void destroy()  
+String getInitParameter(String name)  
+Enumeration getInitParameterNames()  
+ServletConfig getServletConfig()  
+ServletContext getServletContext()  
+String getServletInfo()  
+String getServletName()  
+void init()  
+void init(ServletConfig config)  
+abstract void service(ServletRequest req, ServletResponse res)  
+GenericServlet()
```



# J2EE: Servlets et applications web

## Diagramme UML

### **javax.servlet.HttpServlet**

```
+protected void doDelete(HttpServletRequest req, HttpServletResponse resp)
+protected void doGet(HttpServletRequest req, HttpServletResponse resp)
+protected void doOptions(HttpServletRequest req, HttpServletResponse resp)
+protected void doPost(HttpServletRequest req, HttpServletResponse resp)
+protected void doPut(HttpServletRequest req, HttpServletResponse resp)
+protected void doTrace(HttpServletRequest req, HttpServletResponse resp)
+protected long getLastModified(HttpServletRequest req)
+protected void service(HttpServletRequest req, HttpServletResponse resp)
+void service(ServletRequest req, ServletResponse res)
```

# J2EE: Servlets et applications web

## La servlet HelloWorld (1/3)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet{
    public void doGet
        (HttpServletRequest req,HttpServletResponse res)
        throws ServletException, IOException{

        //code de la méthode (transparent suivant)

    }
}
```

# J2EE: Servlets et applications web

## La servlet HelloWorld (2/3)

```
public void doGet
(HttpServletRequest req,HttpServletResponse res)
    throws ServletException, IOException {
    // quel type de contenu pour la réponse ?
    res.setContentType("text/html");
    // construire un flot de sortie
    PrintWriter out = res.getWriter();
    // envoi de la page générée
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
    out.println("<BODY>");
    out.println("<H1>Hello World</H1>");
    out.println("</BODY></HTML>");
}
```

# J2EE: Servlets et applications web

## La servlet HelloWorld (3/3)

L'exécution de cette servlet répond à une requête HTTP.

Par exemple: `http://localhost:8080/hello/HelloWorld`

Chaque fois que le serveur web reçoit une requête `GET`, il invoque la méthode `doGET()` en lui transmettant 2 objets traduisant la requête émise et la réponse à fournir

Le paramètre `req` de type `HttpServletRequest` donne accès aux paramètres de la requête, à l'en-tête HTTP, aux informations sur le client

`res` de type `HttpServletResponse` contient les données de la réponse aux client. Elles peuvent être de n'importe quel type

Le type des réponses est précisé par la méthode `setContentType()`

La méthode `getWriter` retourne un objet de type `PrintWriter`, flot de sortie du texte créant la page HTML

# Déploiement d'une application web

- Structure d'une application web
- Descripteur de déploiement

# Déploiement d'une application web

Une application web est composée de servlets, JSP, documents html et d'autres ressources telles que des fichiers image ou son, ...

La portabilité de telles applications impose :

- une structure commune
- une description standard de leur déploiement

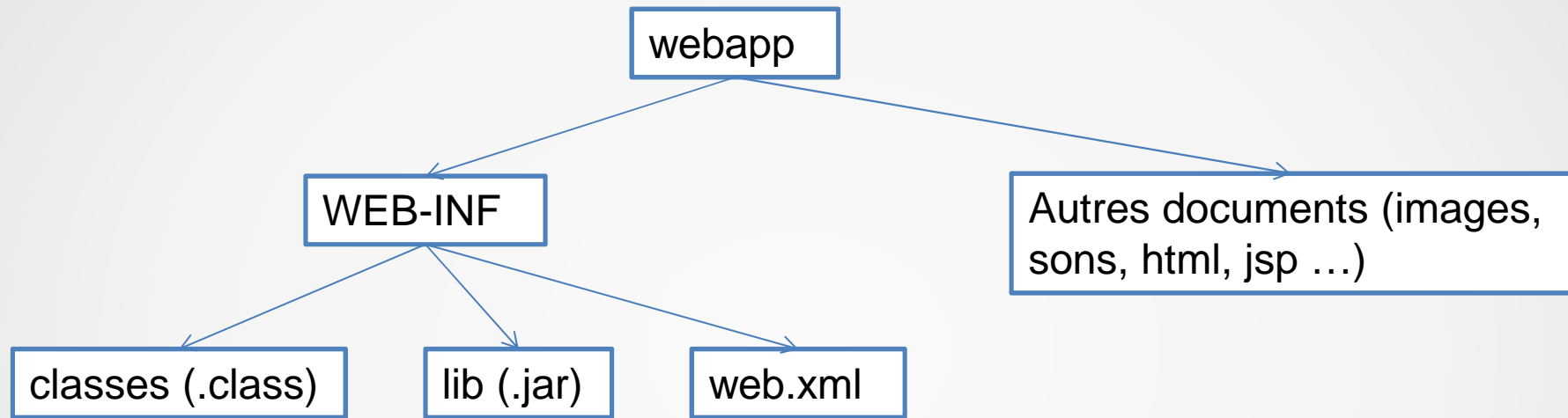
Toutes les ressources d'une application web sont réunies dans un même répertoire dont la structure est strictement définie. L'ensemble des fichiers peut être réuni dans un fichier d'archive (`.war`) sous forme compressée et signé numériquement

La description du déploiement est contenue dans un fichier XML de nom `web.xml`

L'ensemble de ces informations est indépendante du serveur et simplifie le portage et le déploiement d'une application web

# Déploiement d'une application web

## Structure d'une application web



# Déploiement d'une application web

## Le descripteur de déploiement

C'est un fichier XML accompagné d'une DTD (Document Type Definition) ou un schéma XML qui spécifie la structure des balises autorisées pour décrire la configuration d'une application web.

Les balises permettent

- de donner un nom à une servlet et de l'associer à une classe java
- d'associer une ou plusieurs URL à une servlet
- de spécifier des paramètres d'initialisation
- de définir un contexte général à l'application
- de définir des fichiers de bienvenue, des pages d'erreur
- de spécifier la sécurité

La DTD (Document Type Definition) contient plus de 50 balises:



# Déploiement d'une application web

## Structure générale de déploiement

```
<web-app>
<context-param> ... </context-param>
<servlet> ... </servlet>
<servlet-mapping> ... </servlet-mapping>
<session-config> ... </session-config>
<mime-mapping> ... </mime-mapping>
<error-page> ... </error-page>
<welcome-file-list> ... </welcome-file-list>
</web-app>
```

Paramètres  
d'initialisation du  
contexte de l'app.

Déclaration des  
données de la  
servlet

Paramètres de session

Association entre  
codes d'erreur et  
traitement

Liste des fichiers  
requis par défaut

Association entre types  
mime et extension

# Déploiement d'une application web

## Structure de déploiement d'une application web (1/2)

```
<web-app>
  <context-param>
    <param-name>...</param-name>
    <param-value>...</param-value>
  </context-param>
  <servlet>
    <servlet-name>...</servlet-name>
    <servlet-class>...</servlet-class>
    <init-param>
      <param-name>...</param-name>
      <param-value>...</param-value>
    </init-param>
  </servlet>
  .... // autres servlets
```

valeurs partagées par  
plusieurs servlets

valeurs  
d'initialisation  
d'une servlet

# Déploiement d'une application web

## Structure de déploiement d'une application web (2/2)

```
<servlet-mapping>
    <servlet-name>...</servlet-name>
    <url-pattern>...</url-pattern>
</servlet-mapping>
<session-config>
    <session-timeout>...</session-timeout>
</session-config>
<mime-mapping>
    <extension>...</extension> // exemple pdf
    <mime-type>...</mime-type> // exemple application/pdf
</mime-mapping>
<welcome-file-list>
    <welcome-file>...</welcome-file> // exemple index.jsp
    <welcome-file>...</welcome-file> // exemple index.htm
</welcome-file-list>
<error-page>
    <error-code>...</error-code>
    <location>...</location>
</error-page>
</web-app>
```

Association entre  
servlet et URL

redéfinit le  
comportement par  
défaut pour une erreur

# Cycle de vie d'une servlet

- Création/Initialisation
- Traitement des requêtes
- Destruction

# Cycle de vie d'une servlet

## Introduction

Le conteneur de servlet exécute le plus souvent l'ensemble des servlets dans une même JVM.

Les avantages sont :

- Performance, partage des informations entre les servlets
- Persistance, une servlet persiste en mémoire sous la forme d'un objet Java (instance d'une classe). Les données persistent à travers les requêtes. Par exemple, une seule connexion à une base de données est nécessaire pour être utilisée par toutes les requêtes successives. Les informations récupérées sur une base sont disponibles par la suite par toutes les requêtes
- Sécurité, chaque servlet possède sa propre partie privée

Le rechargement d'une servlet est automatique lorsque son code est modifié (à condition qu'elle se trouve dans `WEB-INF/classes`)

# Cycle de vie d'une servlet

## Création/initialisation

Après chargement d'une servlet (création de l'instance), le serveur appelle la méthode `init()` avant tout traitement de requête

La méthode `init()` a pour rôle d'initialiser la servlet, pour définir des valeurs initiales ou par défaut.

Les paramètres d'initialisation ne sont pas associés à une requête. Ils sont toujours disponibles pour la servlet.

Ils se trouvent dans le descripteur de déploiement `web.xml`

# Cycle de vie d'une servlet

## La classe ServletConfig

La classe ServletConfig fournit 3 méthodes:

- Accès aux paramètres d'initialisation de la servlet :

```
public String ServletConfig.getInitParameter(String nom)
```

- Accès aux noms des paramètres d'initialisation de la servlet

```
public Enumeration ServletConfig.getInitParameterNames()
```

- Accès au nom de la servlet

```
String ServletConfig.getServletName()
```

# Cycle de vie d'une servlet

## Exemple 3 (1/2)

```
<servlet>
  <servlet-name>Exemple</servlet-name>
  <servlet-class>servlets.Exemple3</servlet-class>
    <init-param>
      <param-name>user</param-name>
      <param-value>moi</param-value>
    </init-param>
</servlet>
```



# Cycle de vie d'une servlet

## Exemple 3 (2/2)

```
public void init(ServletConfig config)
    throws ServletException {
    String user = config.getInitParameter("user");
}
```

```
public void init(ServletConfig config)
    throws ServletException {
    Enumeration enum = config.getInitParameterNames();
    while( enum.hasMoreElements() ){
        String nom =(String)enum.nextElement();
        out.println(nom);
    }
}
```

# Traitement des requêtes

- A chaque requête correspond un thread Java indépendant.
- Le moteur de servlet appelle ensuite la méthode `service()` responsable du traitement de la requête
- Plusieurs threads peuvent exécuter simultanément les méthodes d'une même instance de servlet
- Dans le modèle par défaut, il appartient au programmeur de gérer la concurrence d'accès aux données

# Destruction d'une requête

- Après le traitement de toutes les requêtes et le déchargement de la servlet, le container de servlet appelle la méthode `destroy()`
- La destruction libère les ressources acquises par la servlet. Il est donc encore possible de sauvegarder des informations.
- En pratique, on utilise la méthode `destroy()` pour fermer des fichiers, des connexions à des bases de données ou sauvegarder un état

# Gestion du cache : coté client

- Les pages téléchargées sont généralement stockées dans le cache
- En cas de téléchargement de la même page, pas d'appel à `doGet()`, la page est retirée du cache
- Pour gérer le cache, la servlet envoie dans sa réponse un en-tête `Last-Modified`
- Le client envoie un en-tête `If-Modified-Since` (date de la dernière modification de la page téléchargée)

# Gestion du cache : coté serveur

- La servlet peut intercepter sa sortie et la mettre en cache sur le serveur
- Le contenu est renvoyé vers le client à partir du cache si aucune modification n'a eu lieu sur la réponse
- La servlet doit être sous-classe de `CacheHttpServlet` au lieu de `HttpServlet`
- Et implémenter la méthode:  
`getLastModified(HttpServletRequest)`

# Obtenir des informations

- Sur le serveur
- Sur le client
- Sur la requête

# Obtention d'informations sur le client (1/3)

Méthodes permettant d'identifier une machine sur le réseau internet

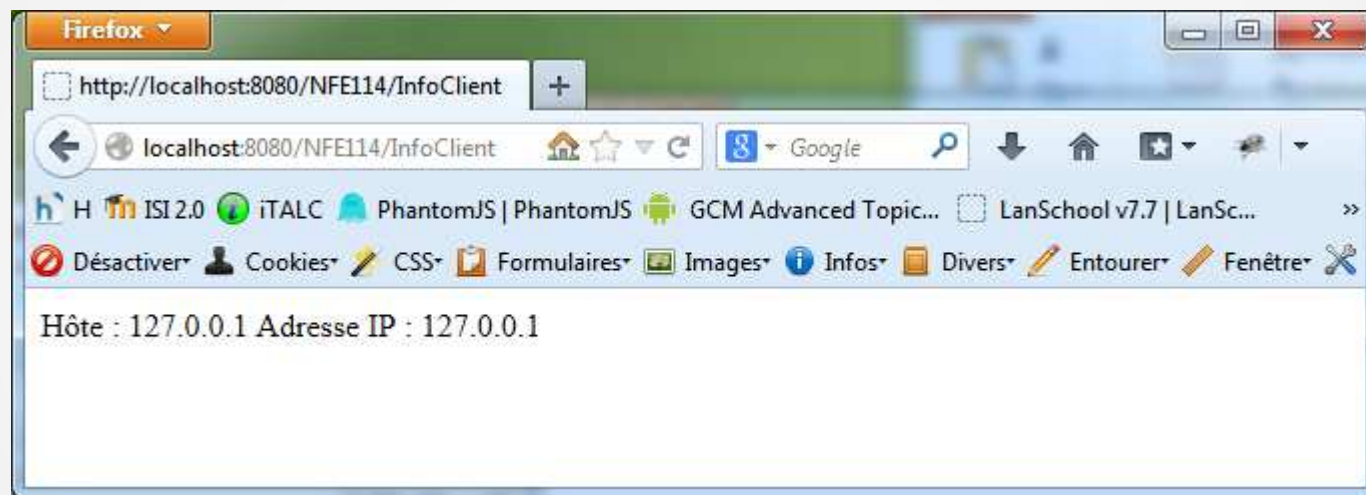
- `getRemoteAddr()` : récupère l'adresse IP de l'hôte qui a initié la requête.
- `getRemoteHost()` : renvoie le nom de la machine hôte qui a initié la requête. Si le nom d'hôte ne peut pas être récupéré, la représentation de l'adresse IP du client est renvoyé sous forme de chaîne de caractères.

## Obtention d'informations sur le client (2/3)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class InfoClient extends HttpServlet{
    public void doGet
        (HttpServletRequest req,HttpServletResponse res)
            throws ServletException, IOException{
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("Hôte : "+req.getRemoteHost());
        out.println("Adresse IP : "+req.getRemoteAddr());
        out.close();
    }
}
```



# Obtention d'informations sur le client (3/3)



# Obtention d'informations sur le serveur

## Méthode de type `ServletRequest`

- `getServerName()` : permet d'obtenir le nom du serveur
- `getServerPort()` : permet d'obtenir le numéro du port

## Méthode de type `ServletContext`

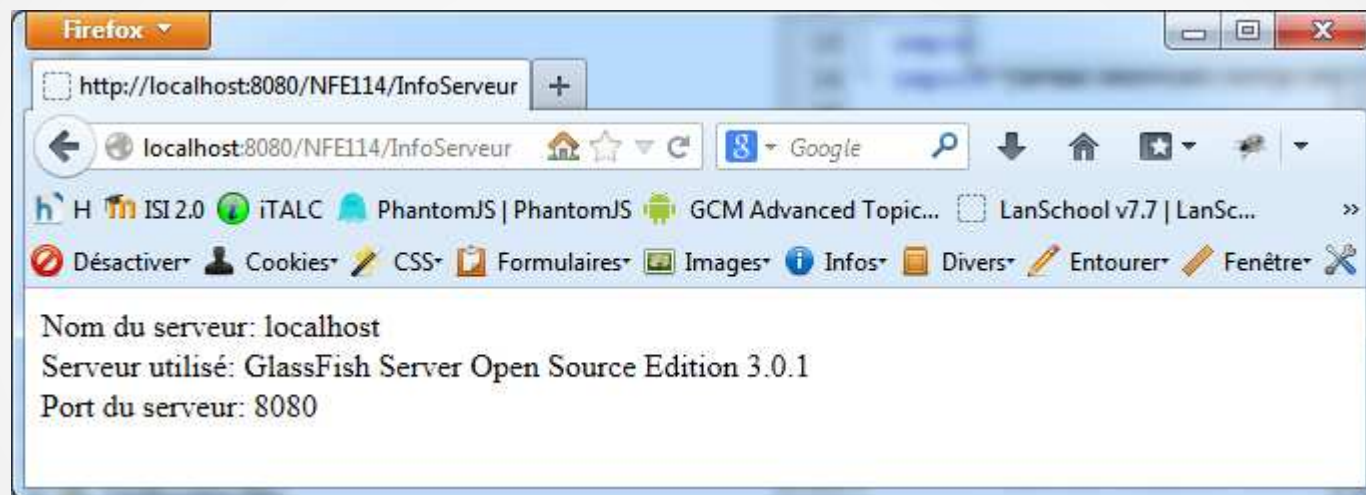
- `getServerInfo()` : retourne le nom et la version du logiciel serveur
- `getAttribute(String name)` : retourne la valeur de l'attribut passé en paramètre

# Obtention d'informations sur le serveur

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class InfoServeur extends HttpServlet {
    public void doGet
        (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println("Nom du serveur: "
            +request.getServerName()+"<br/>");
        out.println("Serveur utilisé: "
            +getServletContext().getServerInfo()+"<br/>");
        out.println("Port du serveur: "
            +request.getServerPort()+"<br/>");
        out.close();
    }
}
```

# Obtention d'informations sur le serveur



# Informations sur la requête (1/3)

Quelques méthodes d'accès à l'en-tête de la requête :

// retourne la valeur de l'en-tête de nom name

```
public String HttpServletRequest.getHeader(String name)
```

// retourne les valeurs multiples d'un en-tête de nom name

```
public Enumeration HttpServletRequest.getHeaders(String  
name)
```

// retourne le nom de chaque en-tête

```
public Enumeration HttpServletRequest.getHeaderNames()
```

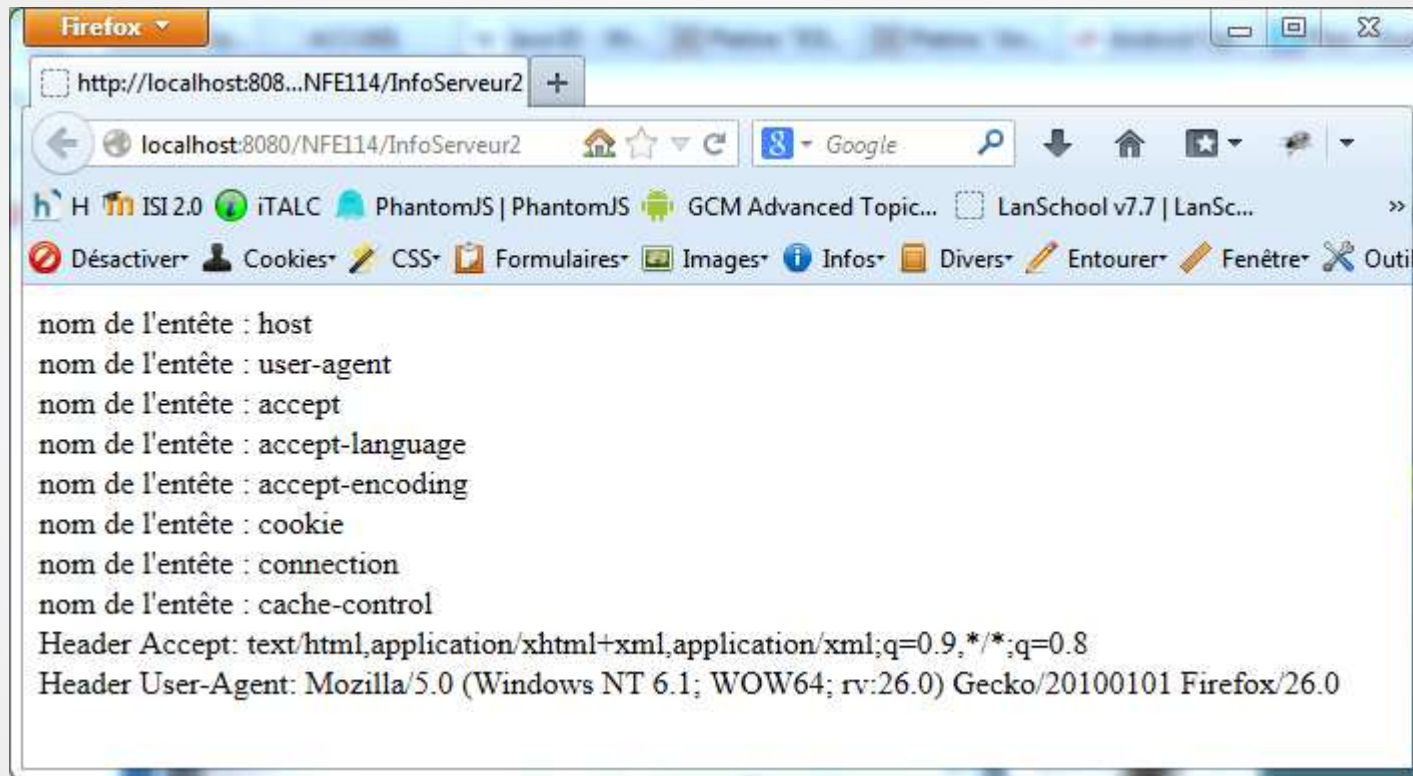
# Informations sur la requête (2/3)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class InfoServeur2 extends HttpServlet {
    public void doGet
        (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        Enumeration e = req.getHeaderNames();
        while(e.hasMoreElements())
            out.println(" nom de l'entête : " + e.nextElement());
        out.println("Header Accept: " + req.getHeader("Accept"));
        out.println("Header User-Agent: "+req.getHeader("User-Agent"));
    }
}
```

# Informations sur la requête (3/3)



# Paramètres de requête

- Des paramètres peuvent être ajoutés à une requête.
- Ils sont différents des paramètres d'initialisation associés à la servlet
- La servlet (HTTP) récupère ces paramètres dans les requêtes HTTP GET ou POST
- La servlet accède à la valeur d'un paramètre connaissant son nom par la méthode :

```
public String ServletRequest.getParameter(String nom)
```

ou bien s'il la valeur retournée n'est pas atomique par

```
public String[] ServletRequest.getParameterValues(String nom)
```



# Paramètres de requête

## Exemple (1/3)

```
public class ChoixCouleur extends HttpServlet {
    protected void doGet
        (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        String melange = req.getParameter("couleur");
        if(melange!=null){
            out.println( melange );
        }
    }

    protected void doPost
        (HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        doGet(request,response);
    }
}
```

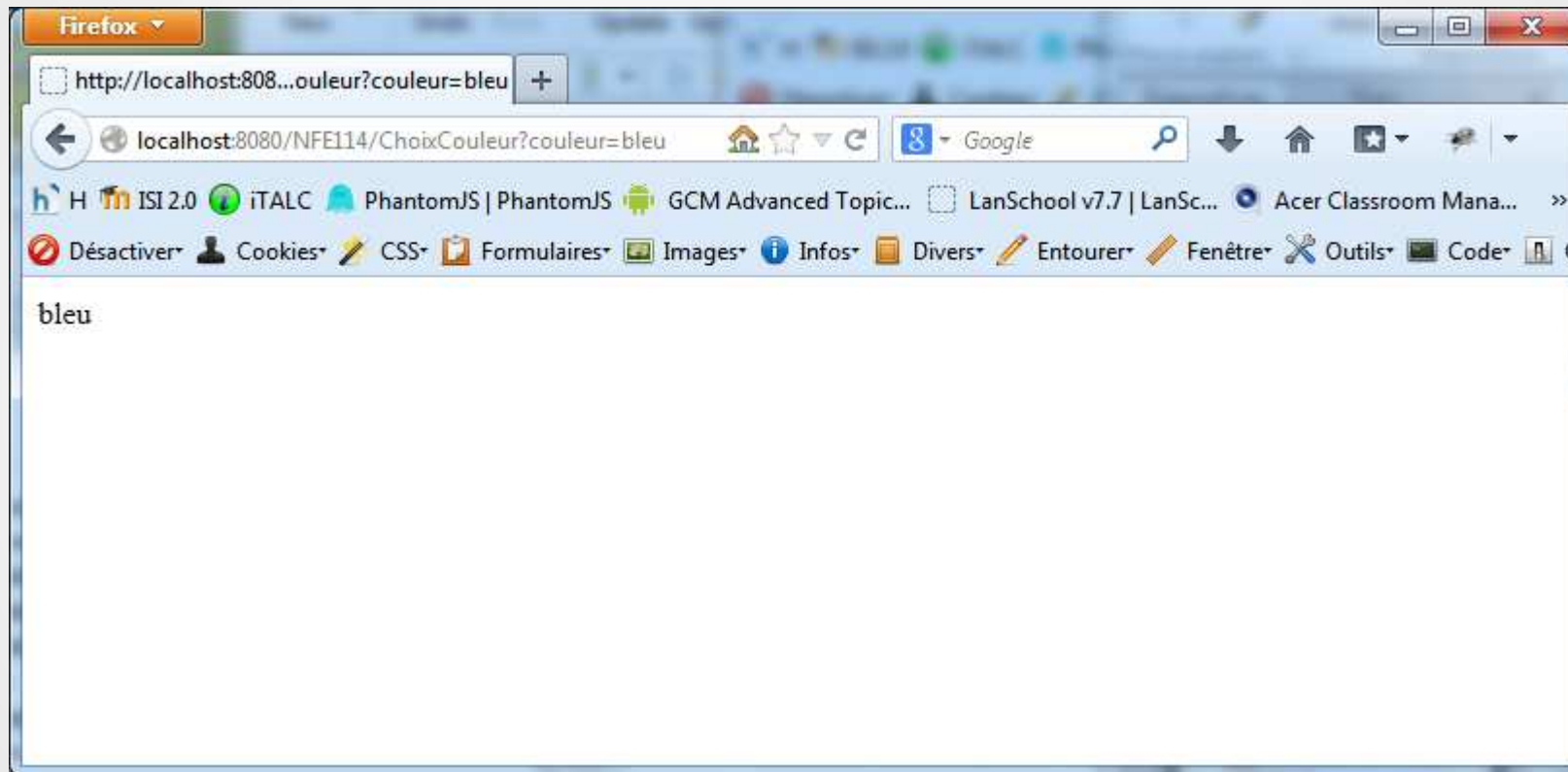
# Paramètres de requête

## Exemple (2/3): web.xml

```
<servlet>
    <servlet-name>ChoixCouleur</servlet-name>
    <servlet-class>LesServlets.ChoixCouleur</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ChoixCouleur</servlet-name>
    <url-pattern>/ChoixCouleur</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>
        index.jsp
    </welcome-file>
</welcome-file-list>
```

# Paramètres de requête

## Exemple (2/3): web.xml



# Formulaires

- Production de formulaires : rappels
- Exemple de formulaire HTML
- La servlet
- Fonctionnement de l'application

# Formulaires

## Production de formulaires : rappels (1/6)

La balise `<form>` au début d'un formulaire

3 attributs:

- **action** page à exécuter après validation des infos saisies
- **method** POST ou GET
- **enctype** spécifie le format des données à envoyer

exemple :

```
<form action=http://localhost:8080/servlet/HelloWorld method=post">
```

- les données sont transmises par la méthode `post`
- la servlet `HelloWorld` est exécutée après validation du formulaire

# Formulaires

## Production de formulaires : rappels (2/6)

La balise `<input>` permet la saisie d'informations à travers plusieurs interfaces graphiques

Les informations saisies peuvent être de la forme:

- ligne de texte
- nom de fichier
- case à cocher

5 attributs :

- **name** nom du paramètre transmis après validation
- **value** texte saisi
- **maxlength** nombre maximal de caractères pouvant être saisis
- **size** taille visible du champ de saisie
- **type** type de l'information à saisir

# Formulaires

## Production de formulaires : rappels (3/6)

L'attribut type :

`<input type = text` information de type texte dans un champ de saisie

`passwd` texte remplacé par des \*

`file` boîte de dialogue pour localiser le fichier

`radio` choisir une case et une seule

`checkbox` sélection d'une ou plusieurs options

`submit` bouton de validation du formulaire

`reset` efface le contenu d'un formulaire

`hidden` envoi de données cachées à l'utilisateur

>

# Formulaires

## Production de formulaires : rappels (4/6)

La balise `<texarea>` insère une zone de saisie de texte

### 3 attributs:

- **name** fournit un nom à la zone de texte pour récupérer les données transmises
- **rows** nombre de lignes de la zone de texte
- **cols** nombre de colonnes de la zone de texte



# Formulaires

## Production de formulaires : rappels (5/6)

La balise `<select>` insère une liste déroulante d'options à sélectionner

attributs :

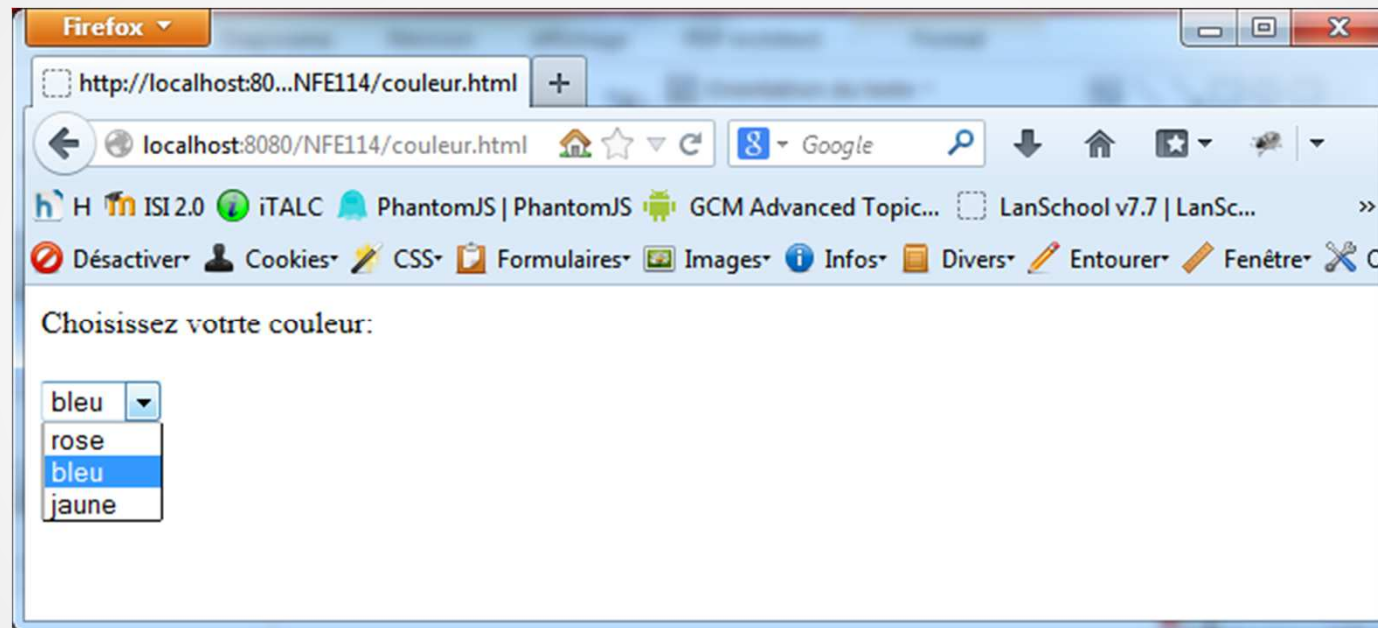
- `name` nom donné au paramètre liste
- `size` nombre de lignes affichées
- `<option value = ... ></option>`

exemple:

```
<select name="liste">  
  <option value="FF80FF">rose</option>  
  <option value="80FFFF">bleu</option>  
  <option value="FFFF80">jaune</option>  
</select>
```

# Formulaires

## Production de formulaires : rappels (6/6)



# Formulaires

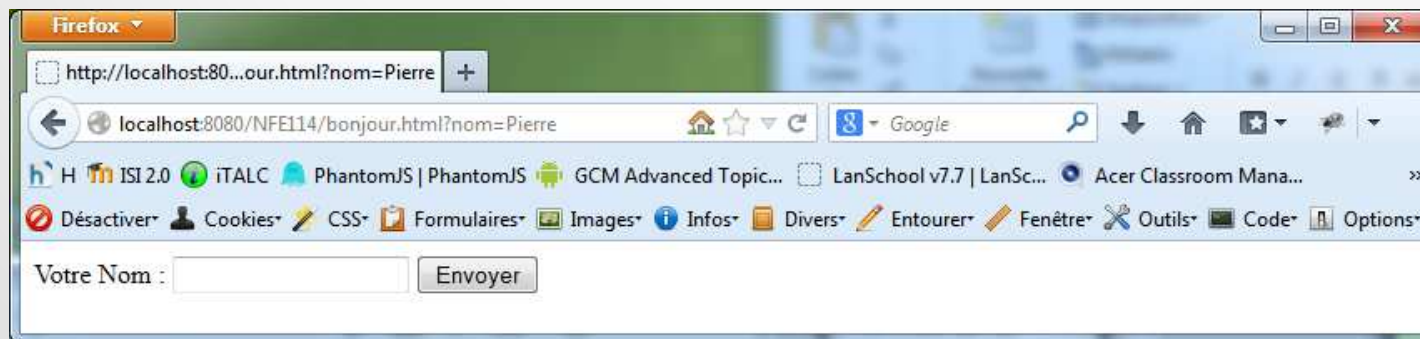
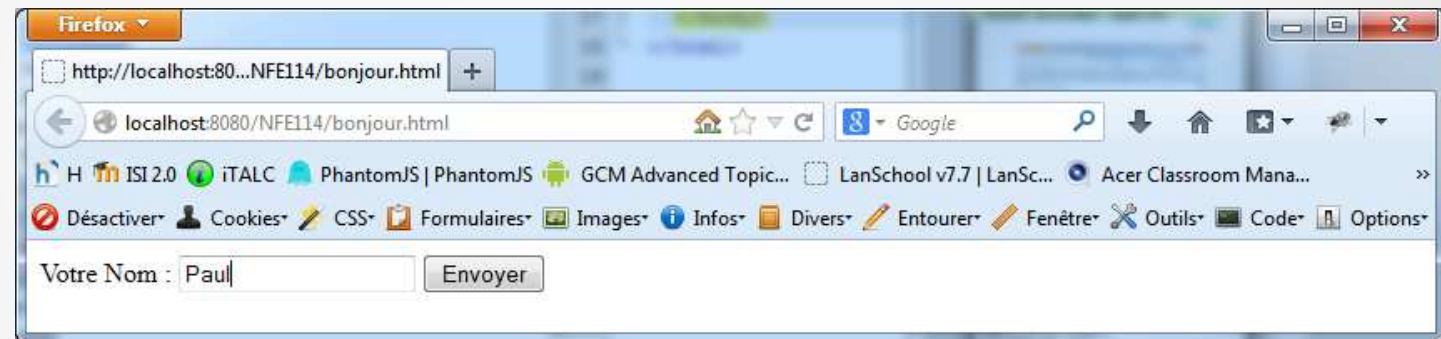
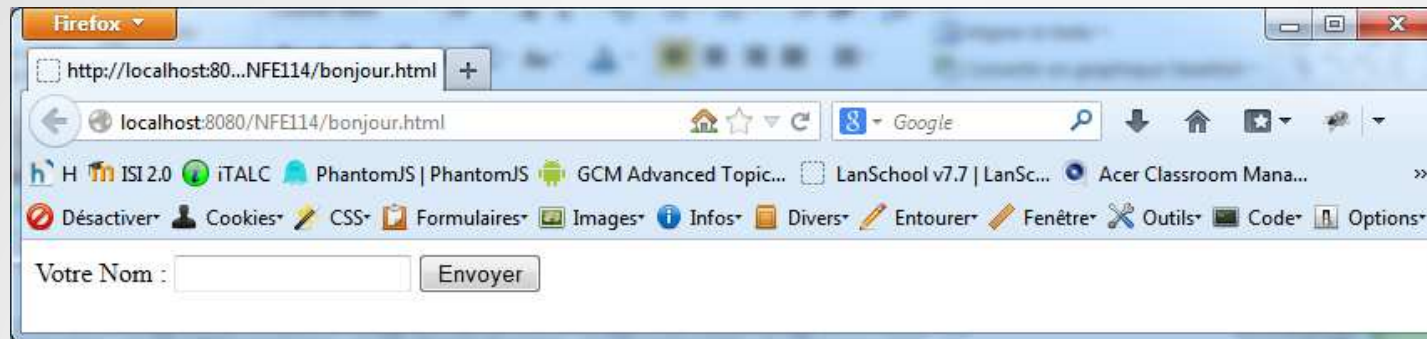
## Exemple 1(1/2)

Fichier bonjour.html

```
<html>
<head>
<title>Un premier formulaire</title>
</head>
<body>
<form method="get" action="">
Votre Nom :
<input type="text" name="nom" />
<input type="submit" />
</form>
</body>
</html>
```

# Formulaires

## Exemple 1(2/2)



# Formulaires

## Exemple 2 (1/4)

Fichier bonjour.html

Page retournée par la requête : <http://localhost:8080/NFE114/bonjour.html>  
et interprétée par le navigateur

```
<html>
  <head>
    <title>Un premier formulaire</title>
  </head>
  <body>
    <form method="get" action="Accueil">
      Votre Nom :
      <input type="text" name="nom" />
      <input type="submit" />
    </form>
  </body>
</html>
```

# Formulaires

## Exemple 2 (2/4)

La servlet Accueil est invoquée à la suite d'une validation du formulaire précédent

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Accueil extends HttpServlet{
    public void doGet
        (HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String nom = req.getParameter("nom");
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Bonjour</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<b>Bonjour </b><b>" + nom + "</b>");
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

# Formulaires

## Exemple 2 (3/4)

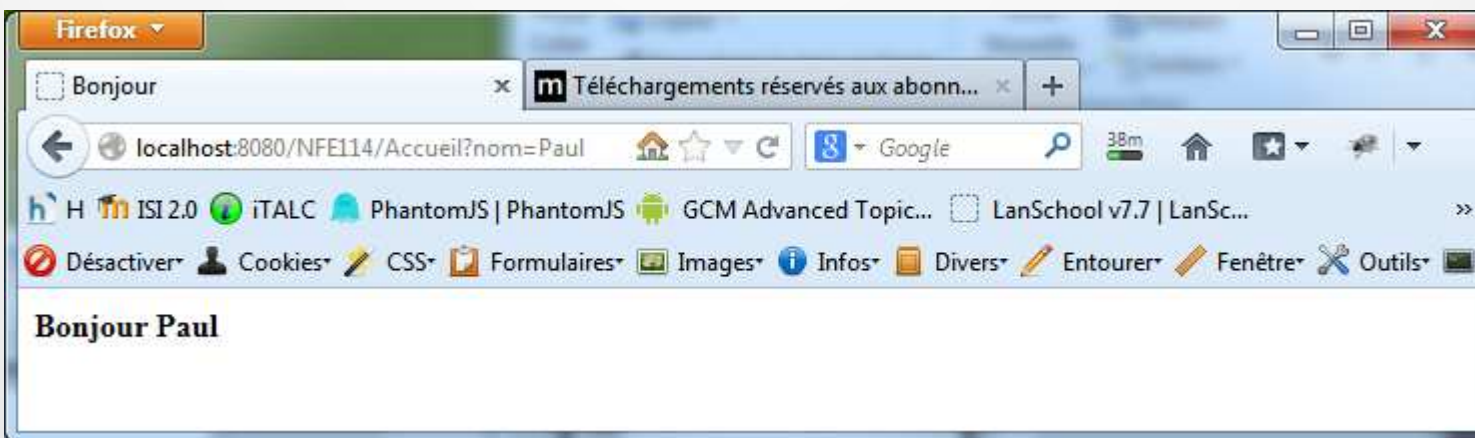
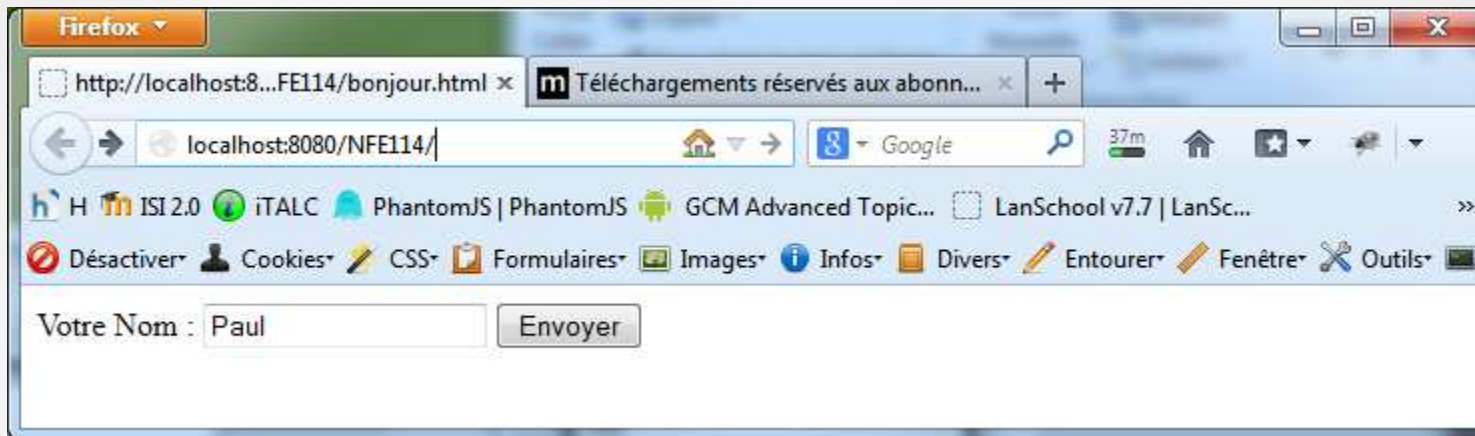
```
<web-app version="2.4" xmlns=http://java.sun.com/xml/ns/j2ee ...>
<servlet>
    <servlet-name>Accueil</servlet-name>
    <servlet-class>servlets.Accueil</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Accueil</servlet-name>
    <url-pattern>/Accueil</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>
        bonjour.html
    </welcome-file>
</welcome-file-list>
```

Le fichier web.xml est le  
descripteur de déploiement de  
l'application



# Formulaires

## Exemple 2 (4/4)





# Formulaires

## Exemple 3 (1/3)

```
<form action='CB' method='post'>

    <b>Référence: </b><input type='text' name='ref' /><br/>
    <b>Quantité : </b><input type='text' name='quantité' /><br/>
    <b>Prix HT : </b><input type='text' name='prix'
value='euros' /><br/>
    <hr/>
    <b>Prénom :</b> <input type='text' name='prénom' /><br/>
    <b>Nom: </b> <input type='text' name='nom' /><br/>
    <b>Adresse:</b><br/>
    <textarea name='adresse' rows='3' cols='40'></textarea><br/>
    <hr/>
```

# Formulaires

## Exemple 3 (2/3)

```
<b>Carte de Credit:</b><br/>
<input type='radio' name='TypeCarte'
value='Visa'>Visa</input><br/>
<input type='radio' name='TypeCarte' value='Master
Card'>Master Card</input><br/>
<input type='radio' name='TypeCarte' value='Java
SmartCard'>Java SmartCard</input><br/>
<b>Numéro de la carte:</b>
<input type='password' name='numéroCarte' /><br/>
<hr/>

<input type='submit' value='valider commande' />
</form>
```

# Formulaires

## Exemple 3 (3/3)

The screenshot shows a Firefox browser window displaying a web form at the URL `http://localhost:8080/NFE114/cartes.html`. The browser's address bar and tabs are visible at the top. The form itself is organized into several sections:

- Top Section:** Contains three input fields labeled "Référence:", "Quantité:", and "Prix HT : euros".
- Personal Information Section:** Contains three input fields labeled "Prénom:", "Nom:", and "Adresse:".
- Carte de Credit Section:** Includes three radio buttons for "Visa", "Master Card", and "Java SmartCard", followed by an input field for "Numéro de la carte:".
- Action:** A "valider commande" button is located at the bottom of the form.

The browser's toolbar shows various icons for navigation and search, and the status bar at the bottom indicates the page is loaded.

# Formulaires

## Exemple 4 (1/5)

Extrait du fichier web.xml

```
<servlet>
  <servlet-name>form</servlet-name>
  <servlet-class>servlets.Form</servlet-class>
  <init-param>
    <param-name>nomParDefaut</param-name>
    <param-value>inconnu</param-value>
  </init-param>
  <init-param>
    <param-name>passParDefaut</param-name>
    <param-value>*****</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>form</servlet-name>
  <url-pattern>/Form</url-pattern>
</servlet-mapping>
```


# Formulaires

## Exemple 4 (2/5)

La servlet requise produisant la page HTML en retour

```
public class Form extends HttpServlet {  
    private String nomParDefault= null;  
    private String passParDefault = null;  
  
    public void init(ServletConfig config) throws ServletException {  
        nomParDefault = config.getInitParameter("nomParDefault");  
        passParDefault= config.getInitParameter("passParDefault");  
    }  
}
```

...



Ces paramètres par défaut sont définis dans le fichier web.xml

# Formulaires

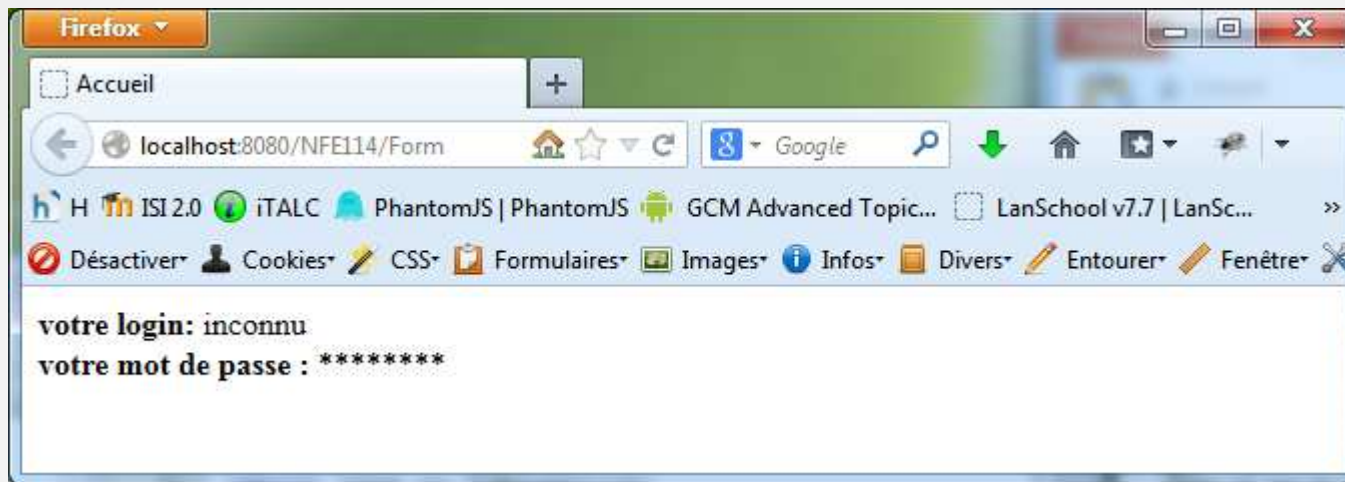
## Exemple 4 (3/5)

La servlet requise produisant la page HTML en retour

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    String nom = req.getParameter("nom");
    String passwd = req.getParameter("passwd");
    if(nom == null)
        nom=nomParDefaut;
    if(passwd == null)
        passwd = passParDefaut;
    out.println("<HTML><HEAD><TITLE>Accueil</TITLE></HEAD><BODY>");
    out.println("<b>votre login: </b>" + nom + "<br>");
    out.println("<b>votre mot de passe : </b>" + passwd);
    out.println("</BODY></HTML>");
}
}
```

# Formulaires

## Exemple 4 (4/5)



# Formulaires

## Exemple 4 (5/5)

Avec l'utilisation des annotations:

```
@WebServlet(name="Form", urlPatterns={"/Form"},
    initParams = {
        @WebInitParam(name="nomParDefaut", value="inconnu"),
        @WebInitParam(name="passParDefaut", value="*****")
    }
)
public class Form extends HttpServlet {
...
}
```



# Formulaires

## Exemple 4 modifiée

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        String nom = request.getParameter("nom");
        String passwd = request.getParameter("passwd");
        if(nom == null)
            nom=nomParDefault;
        if(passwd == null)
            passwd = passParDefault;
        ...
    }
```

# Formulaires

## Exemple 4 modifiée

...

```
out.println("<HTML>");
out.println("<HEAD><TITLE>Accueil</TITLE></HEAD>");
out.println("<BODY>");
```

```
out.println("<form action='' method='get'>");
out.println("<b>votre login: </b>"+nom+"<br><br/>");
out.println("<input type='text' name ='nom' value='"+nom+"' /><br/>");
out.println("<b>votre mot de passe : </b><br/>");
out.println("<input type='password' name ='passwd' value='"+passwd+"' /><br/>");
out.println("<input type='submit' value='valider' /><br/>");
out.println("</form><br/>");
```

```
out.println("Bonjour "+nom+"<br/>");
out.println("Votre mot de passe n'est pas secret: "+passwd+"<br/>");
```

```
out.println("</BODY></HTML>");
} finally {
    out.close();
}
```

```
}
```

# Formulaires

## Exemple 4 modifié

Firefox

Accueil

localhost:8080/NFE114/Form

Google

ISI 2.0 iTALC PhantomJS | PhantomJS GCM Advanced Topic... LanSchool v7.7 | LanSc... robot4.jpg (Image JPE...

Désactiver Cookies CSS Formulaires Images Infos Divers Entourer Fenêtre Outils Code Options

**votre login:** inconnu

inconnu

**votre mot de passe :**

.....

valider

Bonjour inconnu

Votre mot de passe n'est pas secret: \*\*\*\*\*

# Formulaires

## Exemple 4 modifié

Firefox

Accueil

localhost:8080/NFE114/Form

Google

ISI 2.0 iTALC PhantomJS | PhantomJS GCM Advanced Topic... LanSchool v7.7 | LanSc... robot4.jpg (Image JPE...

Désactiver Cookies CSS Formulaires Images Infos Divers Entourer Fenêtre Outils Code Options

**votre login:** inconnu

Pierre

**votre mot de passe :**

.....

valider

Bonjour inconnu

Votre mot de passe n'est pas secret: \*\*\*\*\*

# Formulaires

## Exemple 4 modifié

Notez l'URL

Firefox

Accueil

localhost:8080/NFE114/Form?nom=Pierre&passwd=12345678

Google

H ISI 2.0 iTALC PhantomJS | PhantomJS GCM Advanced Topic... LanSchool v7.7 | LanSc... robot4.jpg (Image JPE...

Désactiver Cookies CSS Formulaires Images Infos Divers Entourer Fenêtre Outils Code Options

**votre login:** Pierre

Pierre

**votre mot de passe :**

.....

valider

Bonjour Pierre

Votre mot de passe n'est pas secret: 12345678

# Formulaires

## Exemple 4 modifiée

`http://localhost:8084/NFE114/Form`

- si première utilisation => chargement et initialisation de la servlet
- requête HTTP => création de l'objet req paramètre de la méthode doGet
- exécution de la méthode doGet
- affichage du formulaire avec les valeurs par défaut des paramètres et la valeur null pour les variables locales puisqu'aucune validation du formulaire n'a été effectuée
- après saisie des login et mot de passe, le formulaire actionne la même servlet de nouveau
- exécution une nouvelle fois de la méthode doGet
- les valeurs par défaut sont rétablies dans les zones de texte et les valeurs saisies s'affichent