

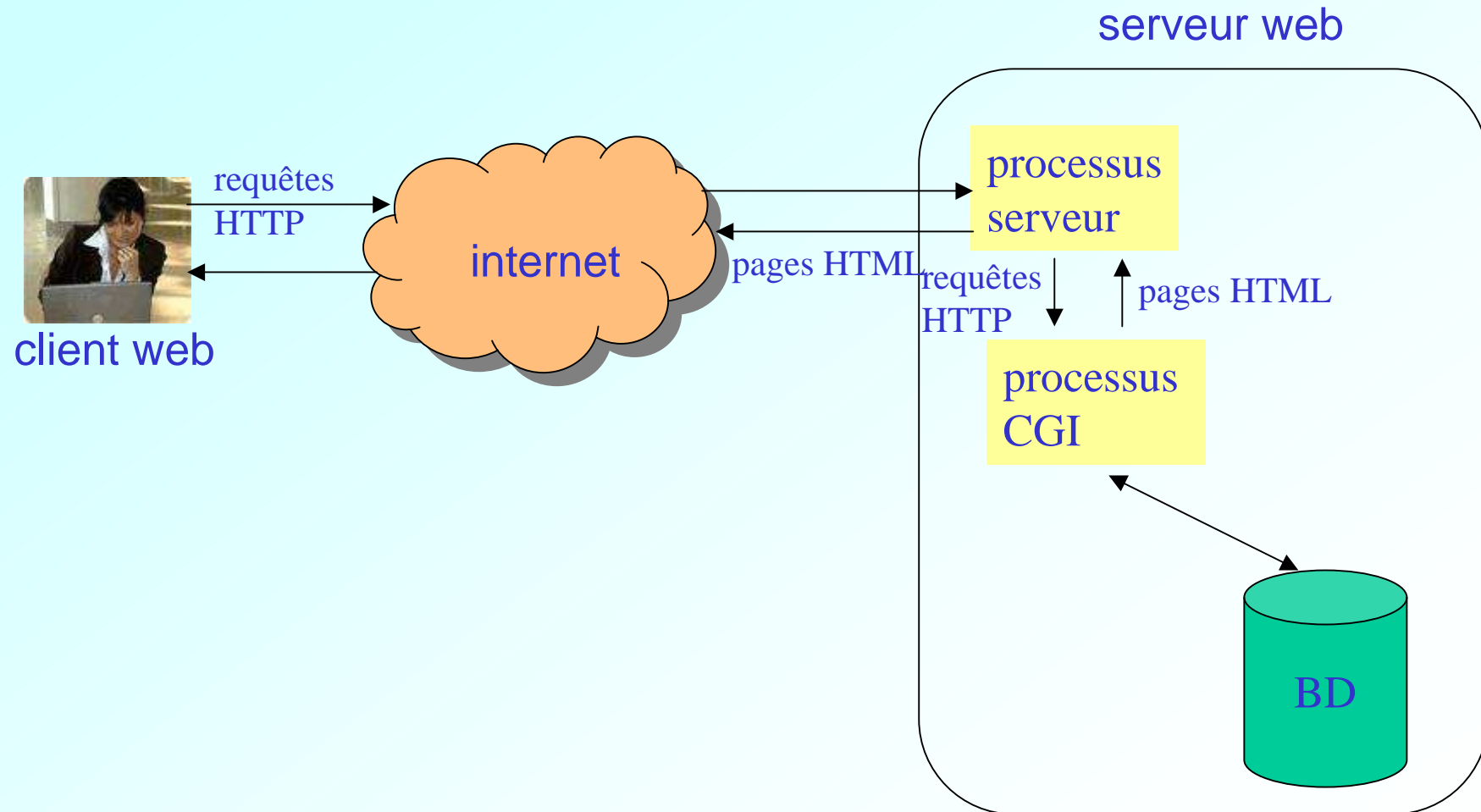
Cours 9 : les servlets Java

- Servlets et applications Web
- Génération de page à partir d'un formulaire
- Déploiement d'une application web
- Cycle de vie d'une servlet
- Servlet, serveur, client, requête : accès aux paramètres

Common Gateway Interface (CGI)

- L'une des premières techniques pour créer du **contenu dynamique**.
- Avec CGI, le serveur Web délègue la requête à un **programme extérieur** (processus fils du processus serveur) qui en retour dirige sa sortie vers le client
- 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 pas 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
- Les scripts CGI peuvent être écrits dans n'importe quel langage, mais le plus souvent en PERL

Architecture CGI



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

Autres solutions

- API d'extension du serveur
 - elles sont propriétaires (WAI pour iPlanet, ISAPI pour IIS)
 - elles sont prises en compte à l'intérieur du processus principal du serveur
 - permettent d'étendre les fonctionnalités du serveur web
- Inconvénients
 - un plantage d'une extension peut entraîner l'arrêt du serveur
 - PB de sécurité: récupération de mots de passe, n° de carte de crédit, ...

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,...)
 - JSP toute plate forme (utilise Java)
- Ces solutions sont uniquement utilisables avec HTTP, ce qui n'est pas le cas des servlets

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

Servlet Java

- Elles font partie intégrante de la plate-forme J2EE
- l'API Servlet contient les paquets `javax.servlet` et `javax.servlet.http`
- Elles n'imposent pas le support de Java sur le navigateur
- 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

Choix des servlets

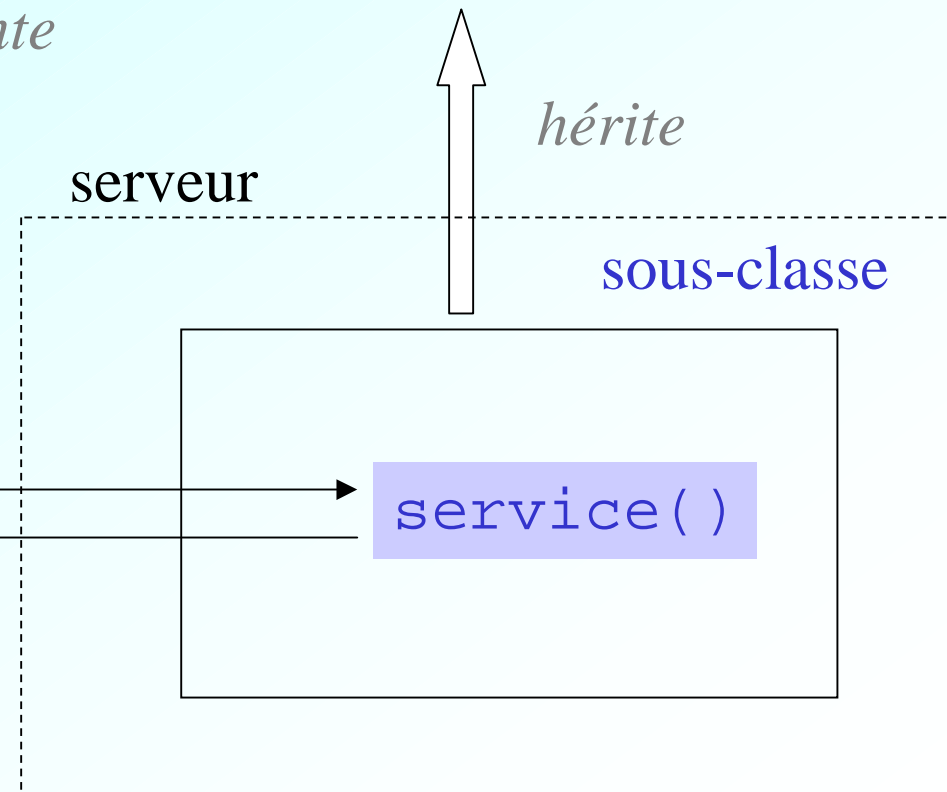
- **Portabilité** : écrite en Java
- **Puissance** : bénéficie des API noyau Java, (réseau, connectivité aux BD, invocation de méthode à distance, multithreading, ...) , de la plate-forme J2EE, de composants JavaBeans du marché, ...
- **Efficacité** : chargée une seule fois, c'est un objet qui reste 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

Servlet générique

`javax.servlet.Servlet` \leftarrow ----- `javax.servlet.GenericServlet`
implante

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

requête
réponse



Servlet HTTP

`javax.servlet.Servlet` ←----- `javax.servlet.HttpServlet`
implante

- pas de méthode `main()`
- `doGet()` et `doPost()`
invoquée par `service()`

requête GET
réponse

requête POST
réponse

serveur

hérite

sous-classe

`service()`

`doGet()`

`doPost()`

La servlet HelloWorld (1/3)

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

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

        //code de la méthode (transparent suivant)
    }
}
```

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 flux 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>");
}
```

La servlet HelloWorld (3/3)

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

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

Le paramètre `req` de type `HttpServletRequest` donne accès aux paramètres de la requête, à l'en-tête HTTP, aux informations du 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`, flux de sortie pour le texte HTML de la réponse

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

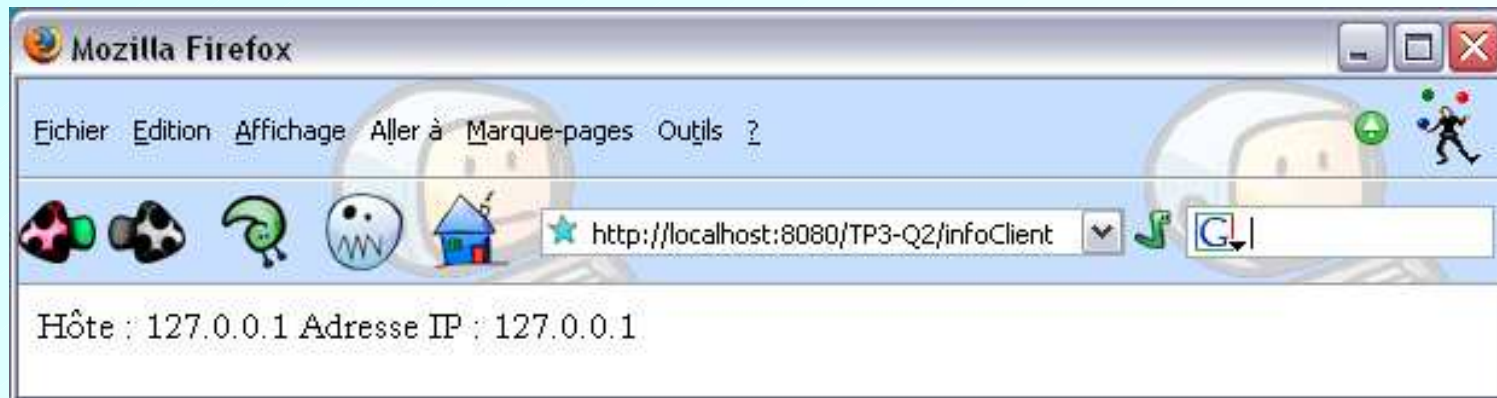
Méthodes permettant d'identifier une machine sur le réseau internet

- `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());
    }
}
```


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



Obtention d'informations sur le serveur (1/3)

Méthode de type `ServletRequest`

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

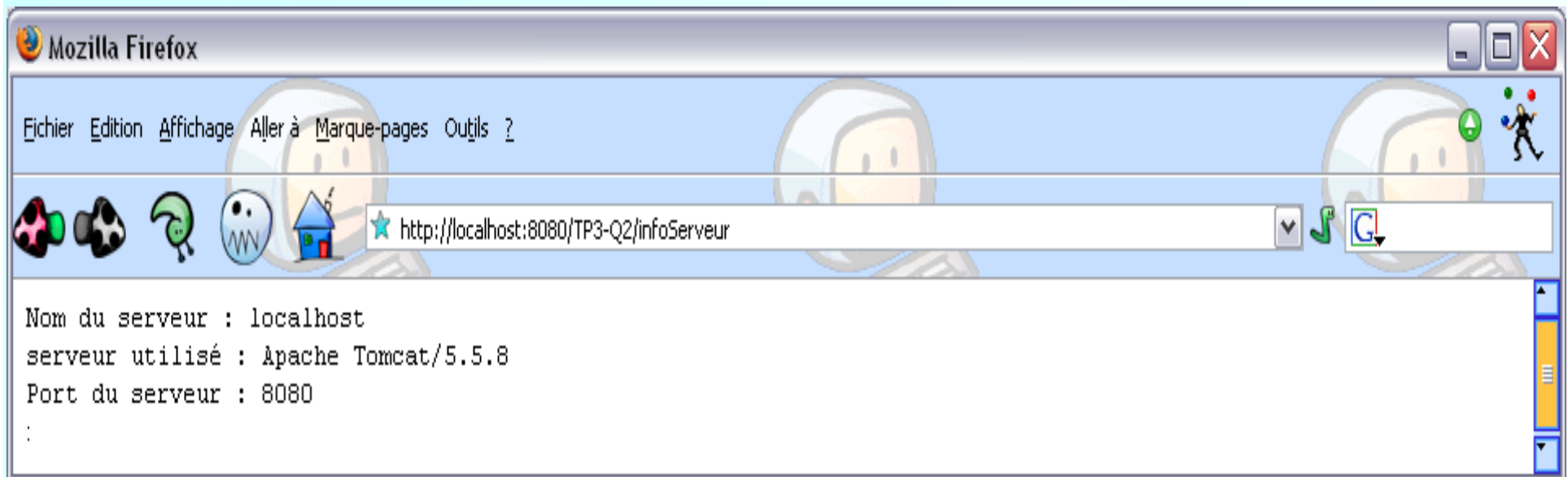
Méthode de type `ServletContext`

- `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 (2/3)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class InfoServeur extends HttpServlet {
    public void doGet
        (HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("Nom du serveur : " + req.getServerName());
        out.print("Serveur utilisé : ");
        out.println(getServletContext().getServerInfo());
        out.println("Port du serveur : " + req.getServerPort());
    }
}
```

Obtention d'informations sur le serveur (3/3)



Obtention d'informations sur la requête (1/3)

Accès aux valeurs de l'en-tête

Méthode de type `HttpServletRequest`

`getHeaderNames()`

retourne le nom de tous les en-têtes sous la forme d'une `Enumeration`

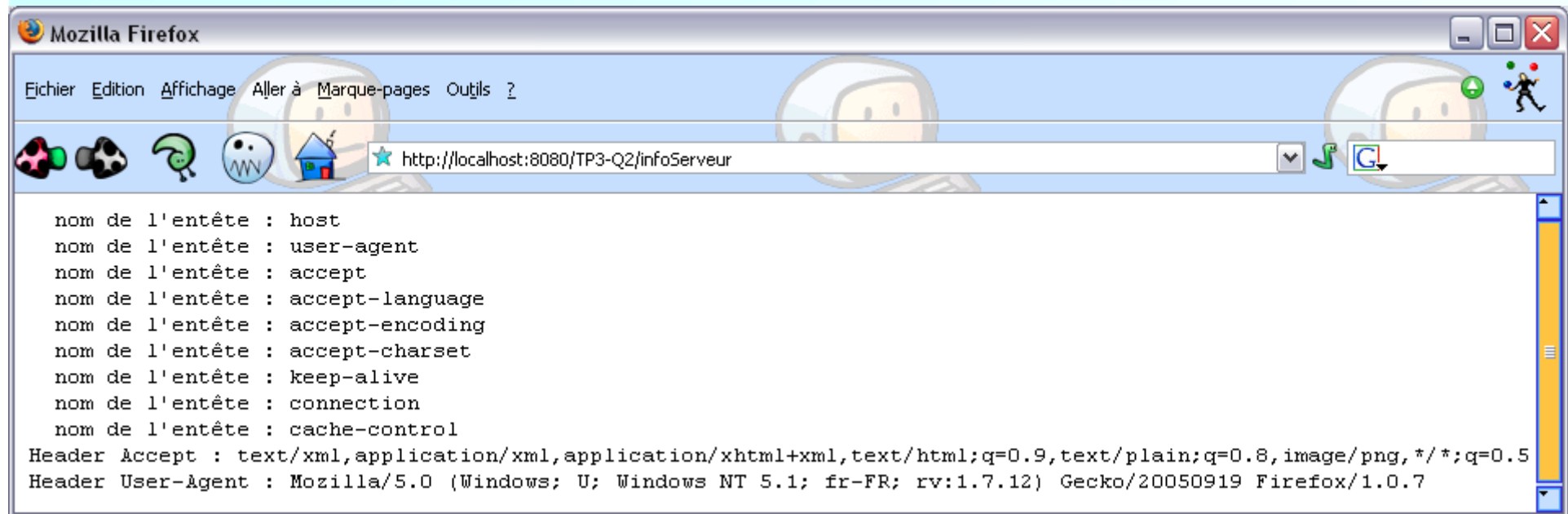
`getHeader(String name)`

retourne la valeur de l'en-tête dont le nom est passé en paramètre

Obtention d'informations sur la requête (2/3)

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class InfoServeur 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"));
    }
}
```

Obtention d'informations sur la requête (3/3)



Générer une page à partir d'un formulaire

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

Production de formulaires : rappels (1/5)

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

Production de formulaires : rappels (2/5)

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 |

Production de formulaires : rappels (3/5)

L'attribut type :

<code><input type = text</code>	→ information de type texte dans un champ de saisie
<code>passwd</code>	→ texte remplacé par des *
<code>file</code>	→ boîte de dialogue pour localiser le fichier
<code>radio</code>	→ choisir une case et une seule
<code>checkbox</code>	→ sélection d'une ou plusieurs options
<code>submit</code>	→ bouton de validation du formulaire
<code>reset</code>	→ efface le contenu d'un formulaire
<code>hidden</code>	→ envoi de données cachées à l'utilisateur

Production de formulaires :

rappels (4/5)

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

Production de formulaires :

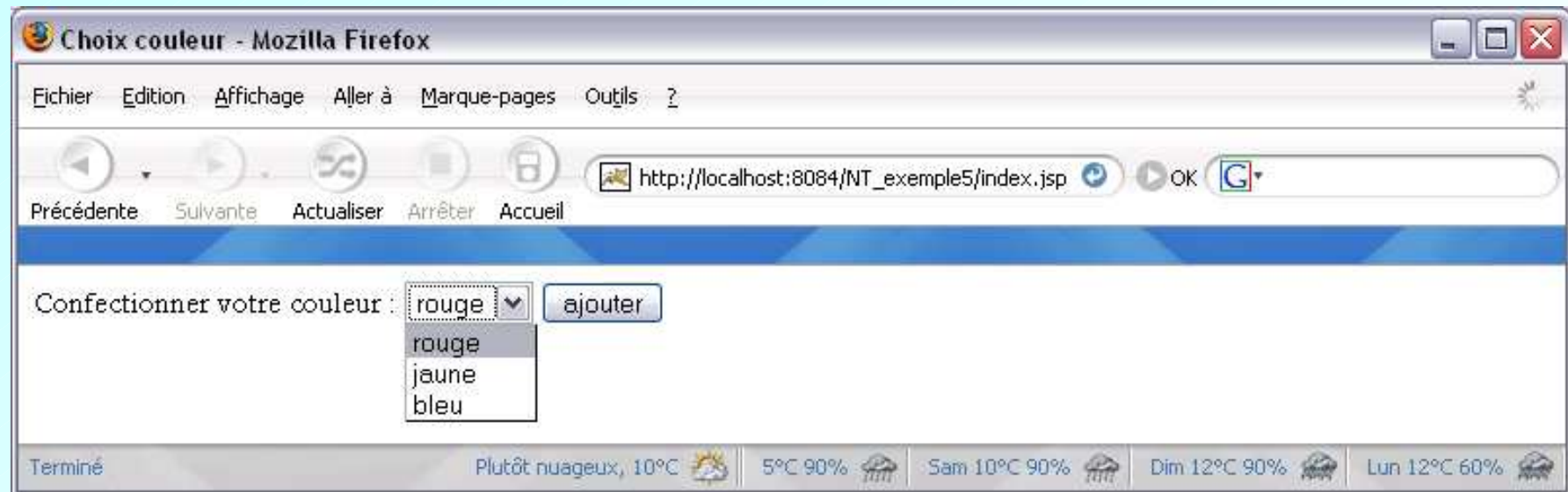
rappels (5/5)

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" size="3">  
  <option value="FF80FF">rose</option>  
  <option value="80FFFF">bleu</option>  
  <option value="FFFF80">jaune</option>  
</select>
```

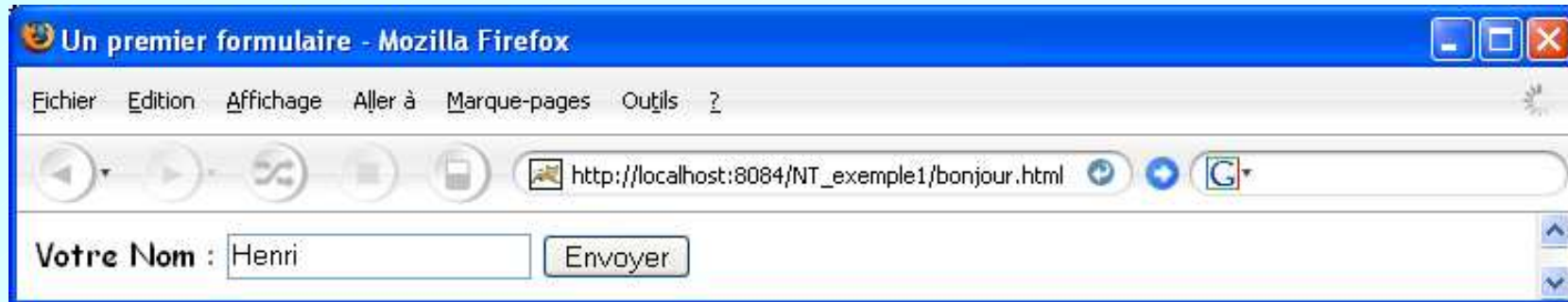
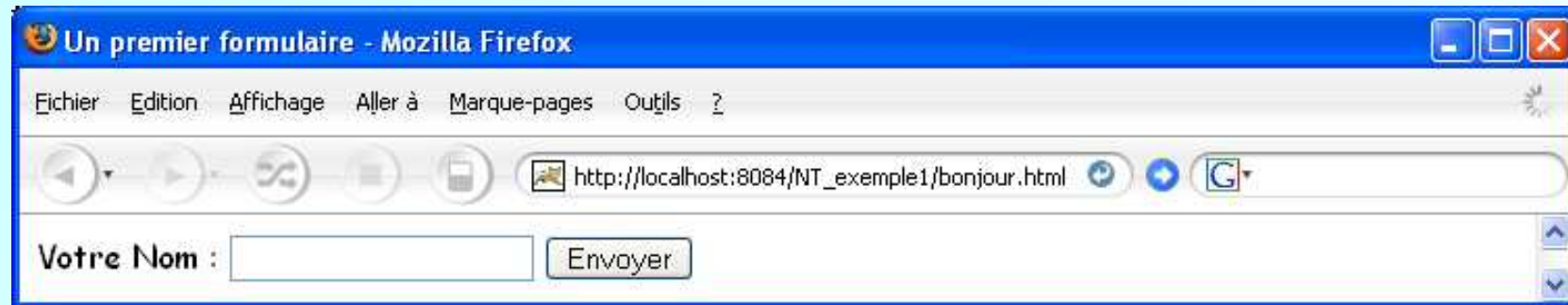


Exemple 1(1/2)

```
<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>
```

Exemple 1(2/2)

L'application `NT_exemple1` consiste en une simple requête à la page `bonjour.html`



Exemple 2 (1/4)

Page retournée par la requête : `http://localhost:8084/NT_exemple2/`
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>
```

Exemple 2 (2/4)

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

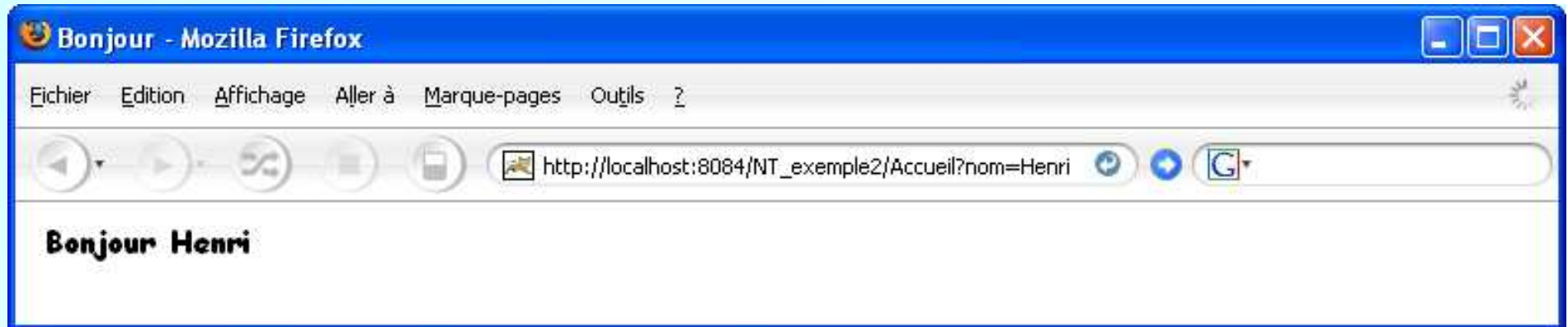
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Accueil extends HttpServlet{
    public void doGet
        (HttpServletRequest req,HttpServletResponse res)
            throws ServletException,IOException{
        res.setContentType("text/html");
        PrintWriter 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>");
    }
}
```

Exemple 2 (3/4)

Le fichier `web.xml` est le descripteur de déploiement de l'application NT_exemple2

```
<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>
</web-app>
```

Exemple 2 (4/4)



Autre exemple de formulaire

```
<form action='/servlets/simplecookie' 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></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></hr>
<b>Carte de Credit:</b><br>
<input type='radio' name='TypeCarte' value='Visa' />Visa <br>
<input type='radio' name='TypeCarte' value='Master Card' />
Master Card <br>
<input type='radio' name='TypeCarte' value='Java SmartCard' />
Java SmartCard<br>
<b>Numéro de la carte: </b>
<input type='password' name='numéroCarte' /> <br>
<b>Répéter (Numéro de la carte):</b>
<input type='password' name='numéroCarte' /> <br><br>
<hr></hr>
<input type='submit' value='valider commande' /><br>
</form>
```

formulaire - Mozilla Firefox

Fichier Edition Affichage Aller à Marque-pages Outils ?

Référence:
Quantité :
Prix HT : euros

Prénom:
Nom :
Adresse:

Carte de Credit:
☐ Visa
☐ Master Card
☐ Java SmartCard
Numéro de la carte:
Répéter (Numéro de la carte):

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 :

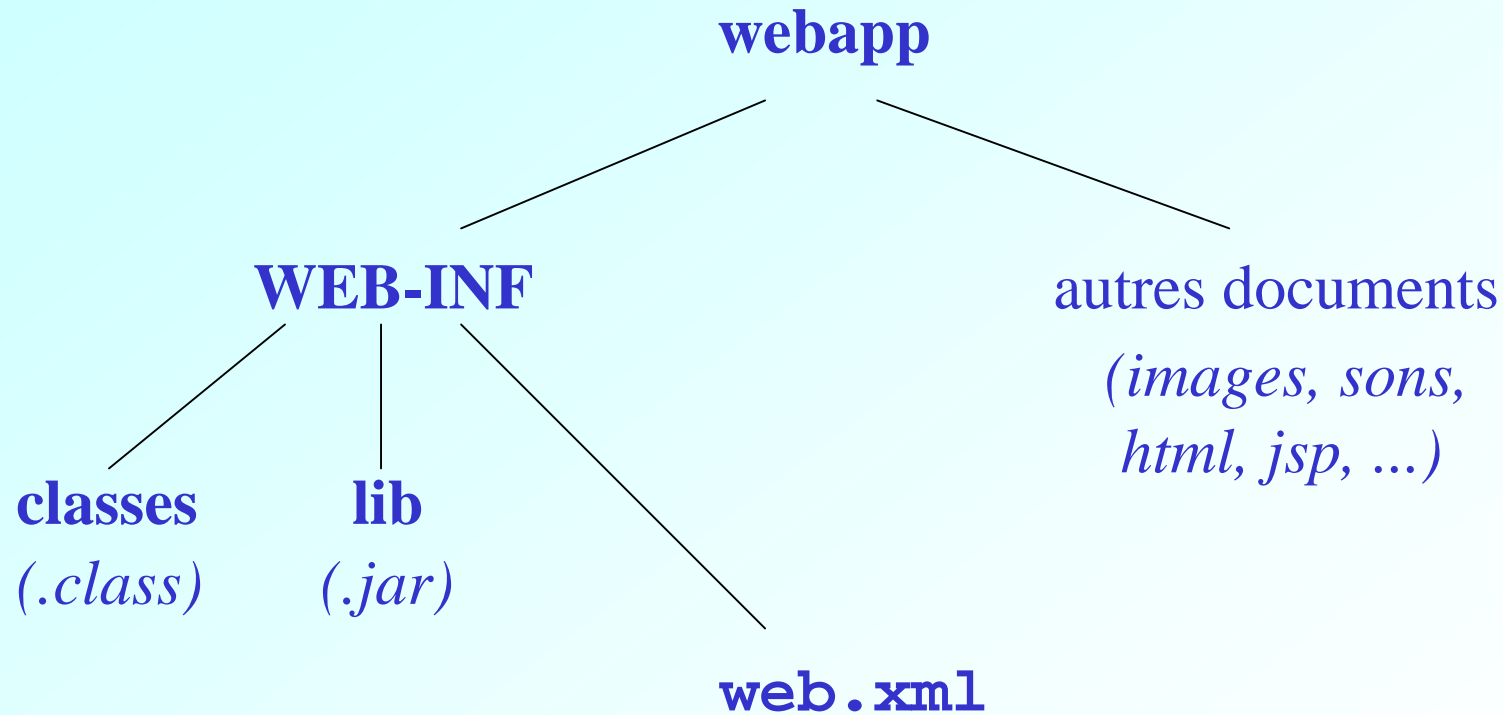
- une structure commune
- une description standard de leur déploiement

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

Structure d'une application web



Le descripteur de déploiement

C'est un fichier XML accompagné d'une DTD 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 contient plus de 50 balises

(http://edocs.bea.com/wls/docs61/webapp/web_xml.html#1014756)

Structure générale de déploiement d'une application web

```
<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 la servlet

Déclaration des
données de la servlet

Paramètres de
session

Liste des fichiers
requis par défaut

Association entre
codes d'erreur et
ressource de
traitement

Association entre
types mime et
extension

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

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

Introduction

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

Les avantages sont :

- **performance**, le 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)

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`

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()
```

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>
```

Exemple 3 (2/2)

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

```
public void init() throws ServletException{  
    ServletConfig config = getServletConfig();  
    Enumeration iter     = config.getInitParameterNames();  
    while( iter.hasMoreElements() ){  
        String nom = (String)iter.nextElement();  
        if( nom.equals("user") )  
            out.println(nom+": "+config.getInitParameter("user"));  
    }
```

Exemple 4 (1/3)

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>
```

Exemple 4 (2/3)

La servlet requise produisant la page HTML en retour

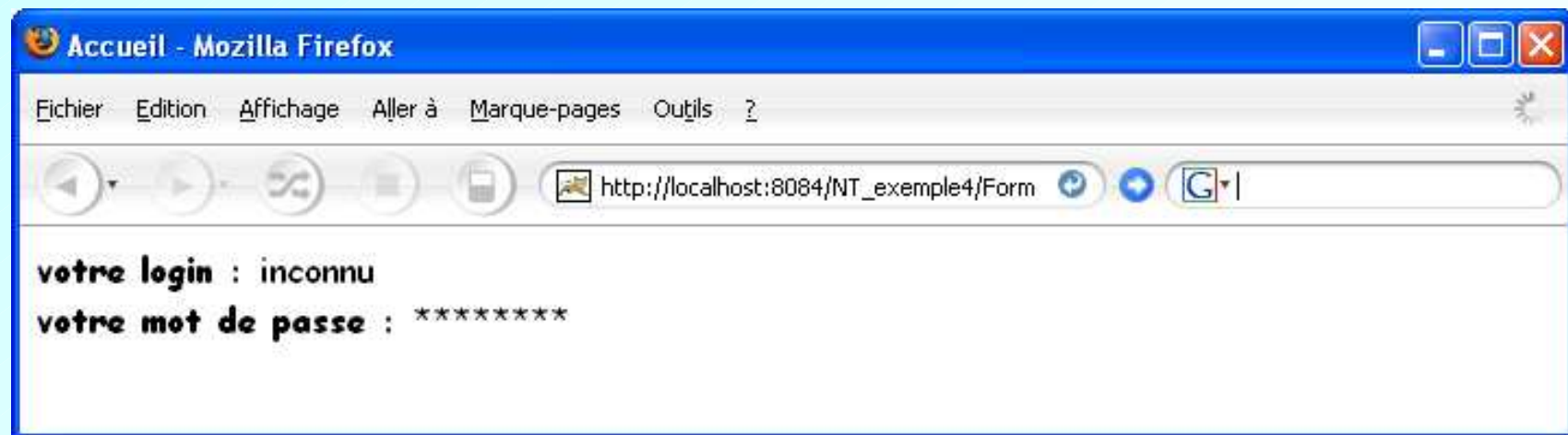
```
public class Form extends HttpServlet {
    private String nomParDefault = null;
    private String passParDefault = null;

    public void init() throws ServletException{
        ServletConfig config = getServletConfig();
        nomParDefault = config.getInitParameter("nomParDefault");
        passParDefault = config.getInitParameter("passParDefault");
    }

    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 = nomParDefault; }
        if (passwd == null) { passwd = passParDefault; }
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Accueil</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<b>votre login          : </b>"+nom+"<br>");
        out.println("<b>votre mot de passe : </b>"+passwd);
        out.println("</BODY></HTML>");
    }
}
```

Exemple 4 (3/3)

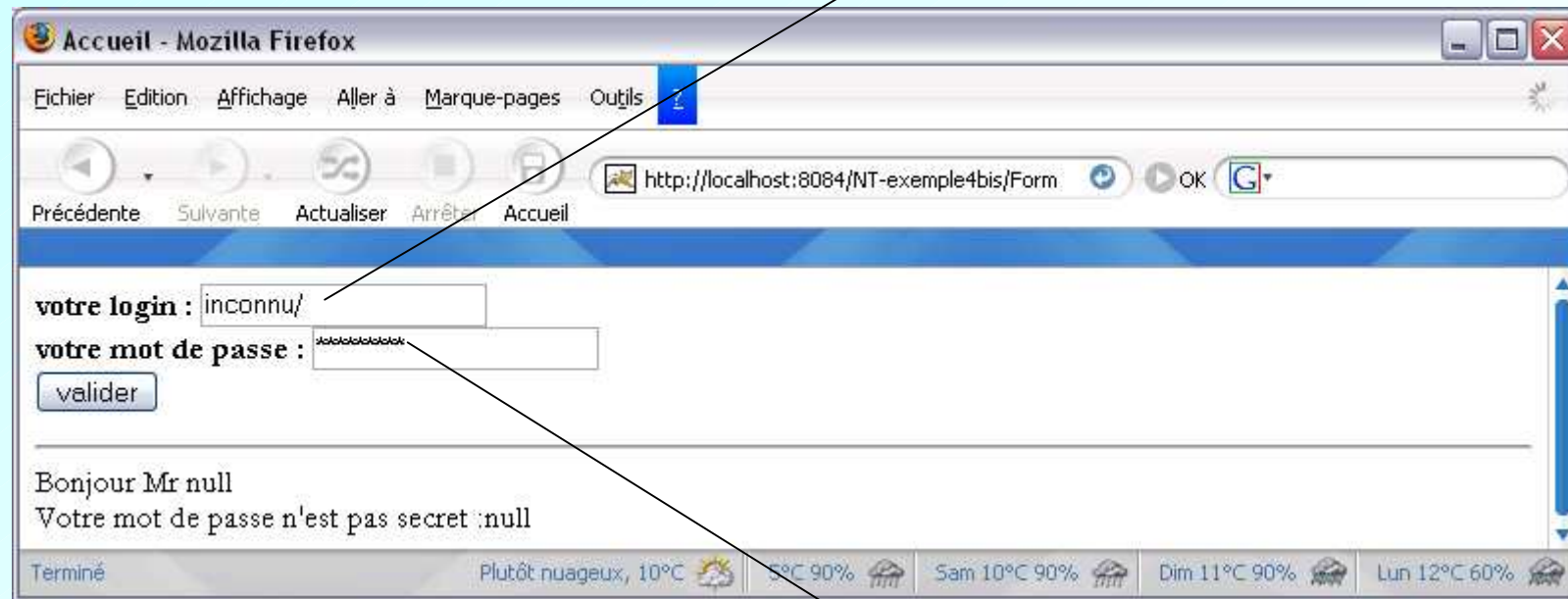
Affichage de la page produite par la servlet `Form`



Exemple 4 bis (1/5)

La servlet `Form` affiche cette page

nom par défaut



The screenshot shows a Mozilla Firefox browser window titled "Accueil - Mozilla Firefox". The address bar displays "http://localhost:8084/NT-exemple4bis/Form". The page content includes a login form with the following elements:


- Label: **votre login :** followed by a text input field containing "inconnu/".
- Label: **votre mot de passe :** followed by a text input field containing "xxxxxxxxxx".
- A "valider" button.
- A message: "Bonjour Mr null".
- A message: "Votre mot de passe n'est pas secret : null".

The browser's status bar at the bottom shows "Terminé" and a weather forecast for the next few days.

mot de passe par défaut

Exemple 4 bis (2/5)

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



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

suite → transparent suivant

Exemple 4 bis (3/5)

partie du fichier web.xml dans laquelle sont définis les paramètres par défaut de la servlet

```
<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>passwdParDefaut</param-name>
    <param-value>*****</param-value>
  </init-param>
</servlet>
```

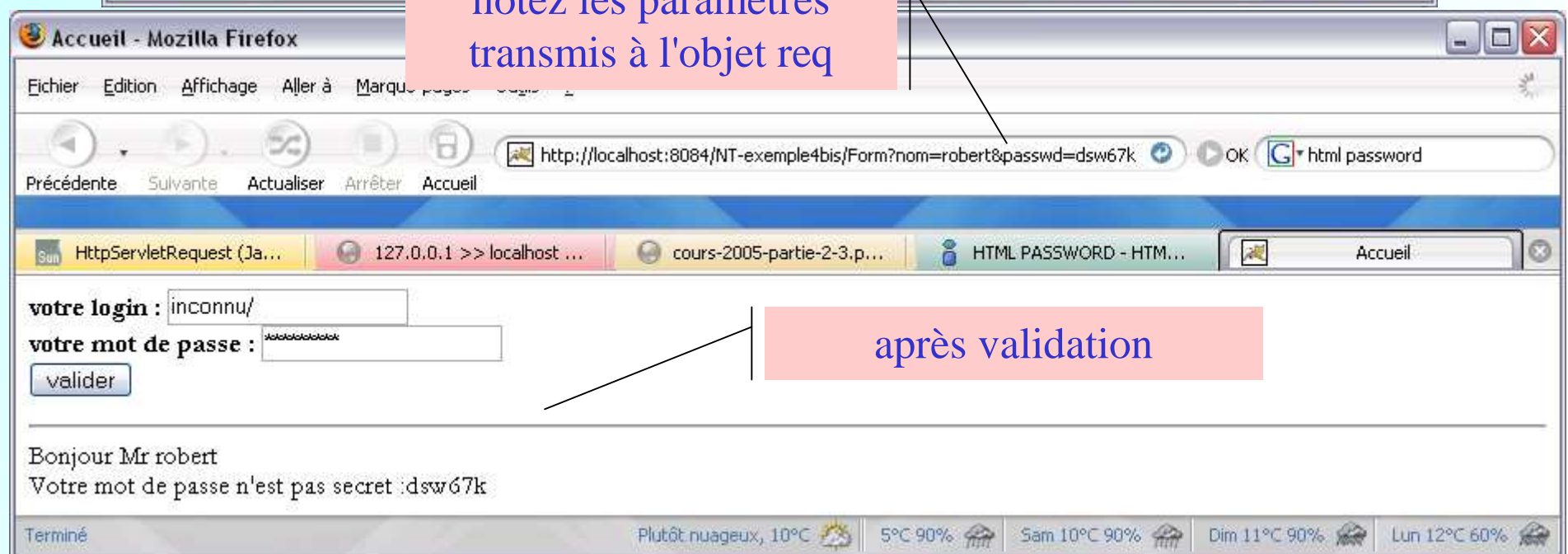
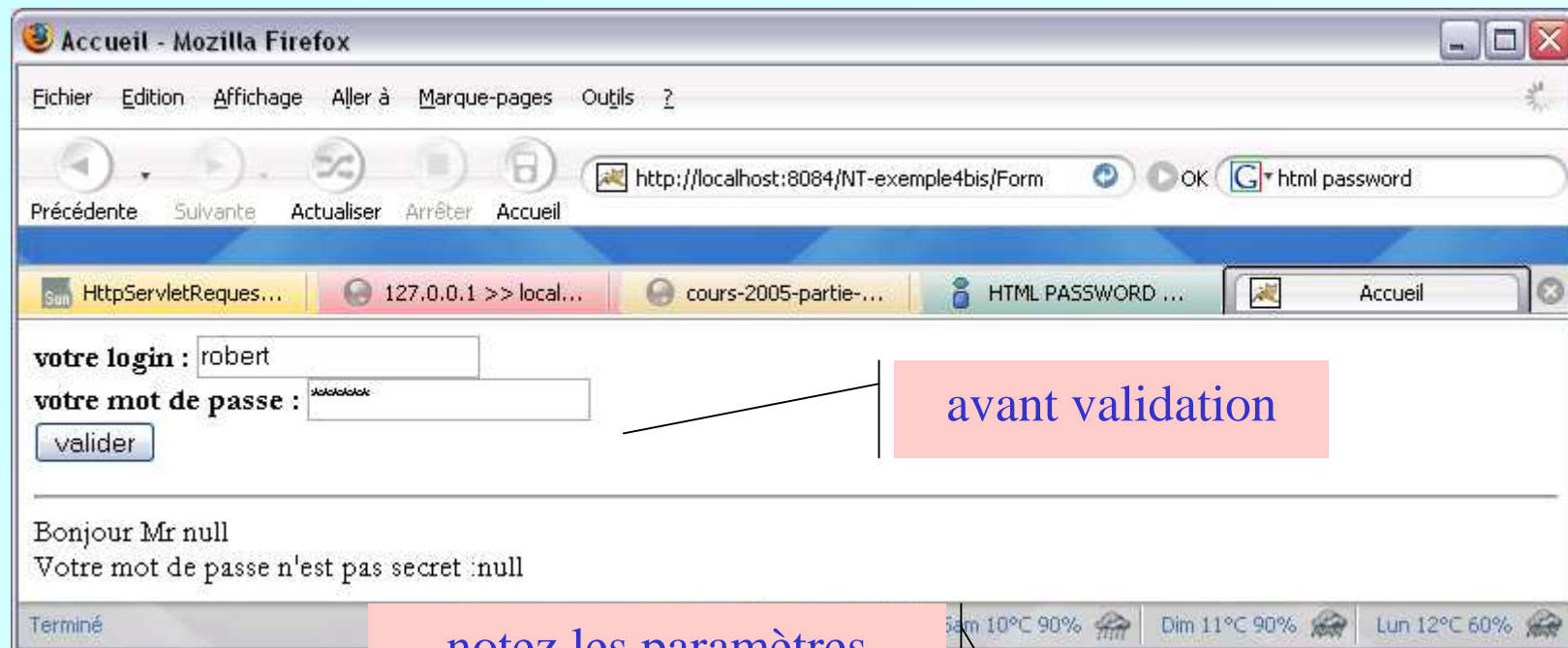
Exemple 4 bis (4/5)

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException{
    res.setContentType("text/html"); PrintWriter out=res.getWriter();
    out.println( "<HTML>" );
    out.println( "<HEAD><TITLE>Accueil</TITLE></HEAD><BODY>" );
    out.println( "<FORM ACTION=' ' method='get'>" );
    out.println( "<b>votre login          : </b>" );
    out.println( "<input type='text' name='nom' value=" +
                  nomParDefaut + "><br>" );
    out.println( "<b>votre mot de passe : </b>" );
    out.println( "<input type='password' name='passwd' value=" +
                  passwdParDefaut + "><br>" );
    out.println( "<input type='submit' value='valider' /><br>" );
    out.println( "</FORM><hr />" );
    String nom = req.getParameter( "nom" );
    String passwd = req.getParameter( "passwd" );
    out.println( "Bonjour Mr " + nom + "<br>" );
    out.println( "Votre mot de passe n'est pas secret :" + passwd );
    out.println( "</BODY></HTML>" );
}
```

Exemple 4 bis (5/5)

<http://localhost:8084/NT-exemple4bis/Form>

1. si première utilisation → chargement et initialisation de la servlet
2. requête HTTP → création de l'objet `req` paramètre de la méthode `doGet`
3. exécution de la méthode `doGet`
4. 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
5. après saisie les login et mot de passe, le formulaire actionne la même servlet de nouveau
6. exécution une nouvelle fois de la méthode `doGet`
7. les valeurs par défaut sont rétablies dans les zones de texte et les valeurs saisies s'affichent,



La classe ServletContext

Permet aux ressources (servlets, jsp, etc...) d'une même application web de partager des informations (contexte commun).
La classe `ServletContext` fournit 2 méthodes :

- Accès aux paramètres d'initialisation du contexte :

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

- Accès aux noms des paramètres d'initialisation du contexte

```
public Enumeration  
    ServletContext.getInitParameterNames()
```

Déploiement du contexte

Exemple : informations d'authentification pour l'accès à une BD

```
<web-app>
<!-- le contexte de l'application web -->
  <context-param>
    <param-name>user</param-name>
    <param-value>samia</param-value>
  </context-param>
  <context-param>
    <param-name>passwd</param-name>
    <param-value>2dsw67k</param-value>
  </context-param>
  . . . . .
</web-app>
```

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 variables d'instance ou de classe de la servlet

Destruction d'une requête

Le serveur de servlet appelle la méthode `destroy()` après déchargement de la servlet et le traitement de toutes requêtes qui lui ont été adressées

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

Requête

- Paramètres d'une requête
- accès à l'en-tête d'une requête

Paramètres de requête

Des paramètres peuvent être ajoutés à une requête.

Ils sont différents des paramètres d'initialisation associés à la servlet (et non à la requête)

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
```

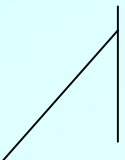
```
    ServletRequest.getParameter(String nom)
```

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

```
public String[]
```

```
    ServletRequest.getParameterValues(String nom)
```

Exemple 5 (1/4)



la servlet qui
traite la requête

```
<form action="ChoixCouleur" method="post" >  
Confectionner votre couleur :  
  <select name= "couleur">  
    <option value="FF80FF">rouge</option>  
    <option value="FFFF80">jaune</option>  
    <option value="80FFFF">bleu</option>  
  </select>  
  <input type="submit" value="ajouter"></input>  
</form>
```

Exemple 5 (2/4)

```
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.getParameterValues("couleur");  
        if (melange!=null){  
            for( int i=0; i<melange.length;i++ )  
                out.println( melange[i] );  
        }  
    }  
    protected void doPost  
        (HttpServletRequest request, HttpServletResponse response)  
            throws ServletException, IOException {  
        doGet(request,response);  
    }  
}
```

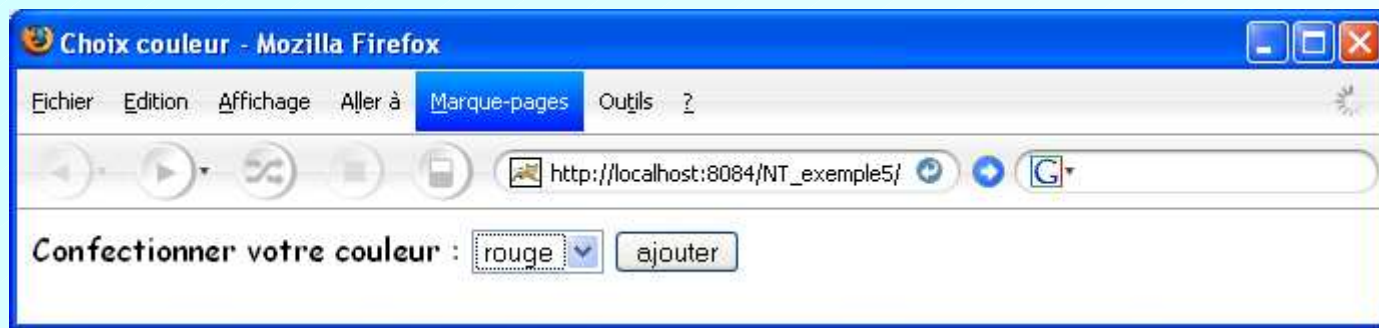
en fait une seule valeur dans le
tableau

Exemple 5 (3/4)

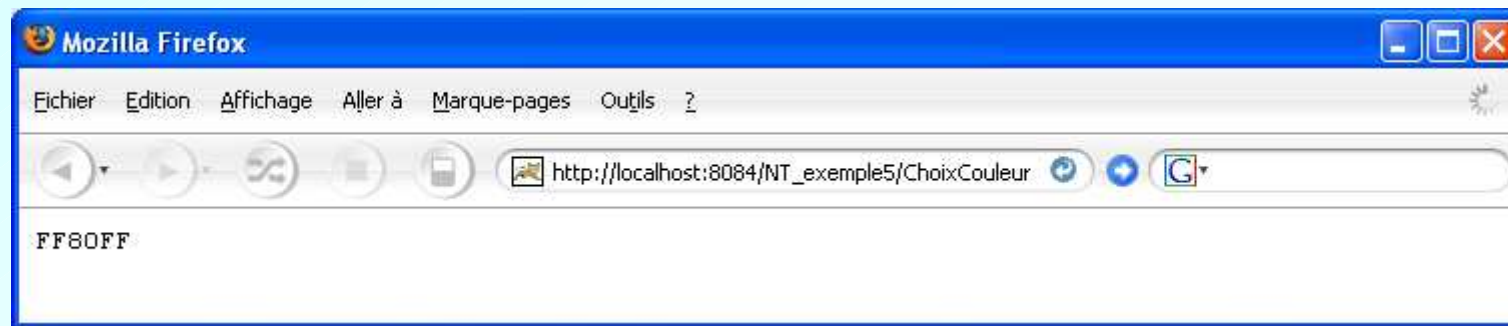
```
<servlet>
    <servlet-name>ChoixCouleur</servlet-name>
    <servlet-class>servlets.ChoixCouleur</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ChoixCouleur</servlet-name>
    <url-pattern>/ChoixCouleur</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>
     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

Exemple 5 (4/4)

Interprétation du fichier de bienvenue `index.jsp`



Interprétation de la servlet `ChoixCouleur`



Accès à l'en-tête de la requête

Quelques méthodes :

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

```
public String
```

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

```
// retourne les valeurs multiples d'une en-tête
```

```
// de nom name
```

```
public Enumeration
```

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

```
// retourne le nom de chaque en-tête
```

```
public Enumeration
```

```
    HttpServletRequest.getHeaderNames(    )
```

Exemple 6 (1/2)

```
public void doGet
(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException{
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println( "liste des en-têtes de la requête :" );
    Enumeration noms = req.getHeaderNames();
    String nom = null;
    String valeur = null;
    while(noms.hasMoreElements()){
        nom = (String) noms.nextElement();
        Enumeration valeurs = req.getHeaders(nom);
        if (valeurs != null){
            while(valeurs.hasMoreElements()){
                valeur = (String)valeurs.nextElement();
                out.println( nom+" : "+valeur );
            }
        }
    }
}
```


Exemple 6 (2/2)

liste des en-têtes de la requête :

host : localhost:8080

user-agent : Mozilla/5.0 (Windows; U; Windows NT 5.1; fr-FR; rv:1.7.5)
Gecko/20041108 Firefox/1.0

accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9,
text/plain;q=0.8, image/png, */*;q=0.5

accept-language : fr,fr-fr;q=0.8,en-us;q=0.5, en;q=0.3

accept-encoding : gzip,deflate

accept-charset : ISO-8859-1,utf-8;q=0.7,*;q=0.7

keep-alive : 300

connection : keep-alive

referer : http://localhost:8080/bonjour/form.html