

# Organisation du cours sur les Servlets

---

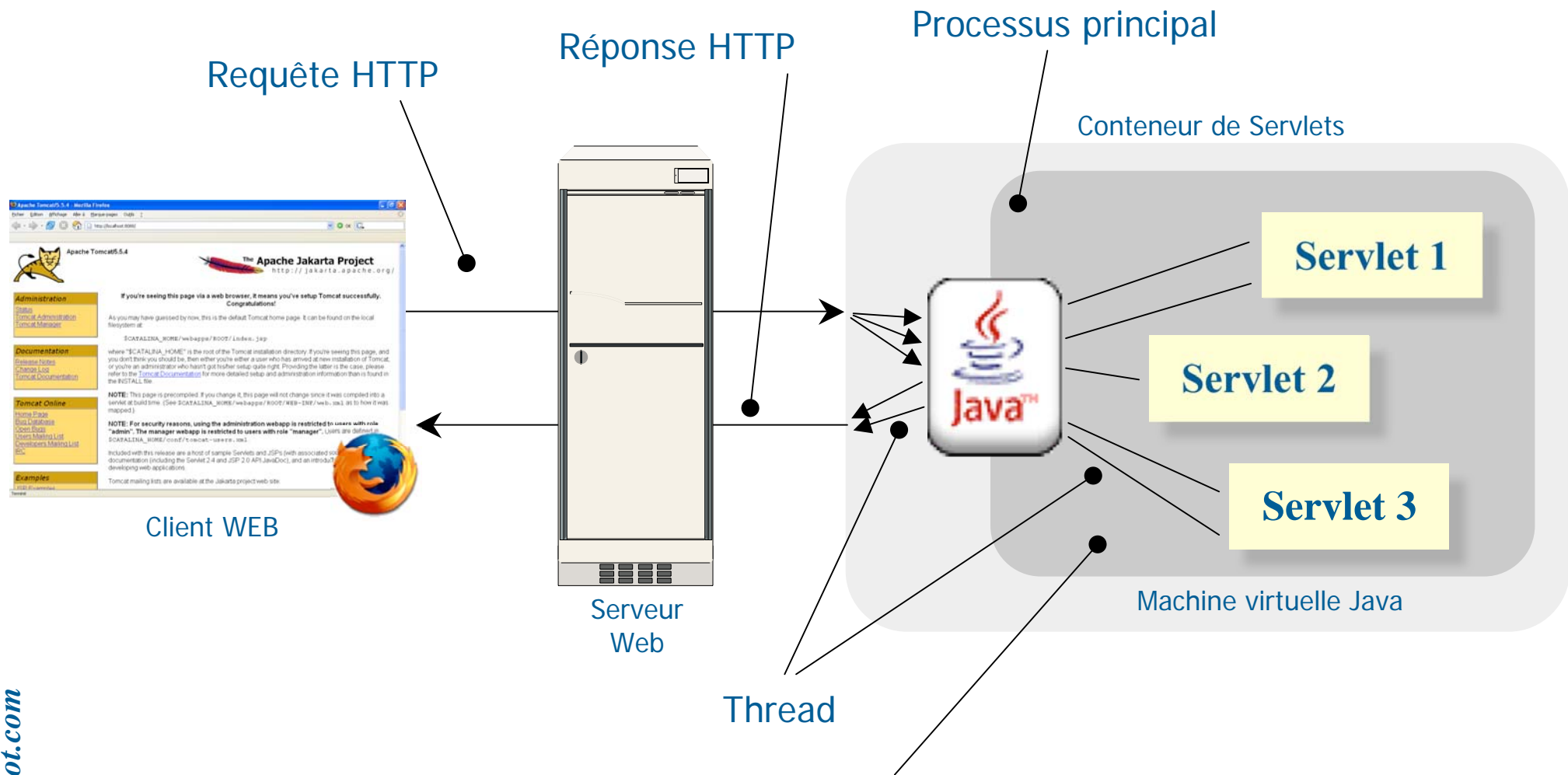
- Servlets et API
- Traitement des données de formulaires
- Architecture de développement
- Cycle de vie
- Suivi de session
- Collaboration de Servlets
- Sécurité : authentification
- Accès aux BD avec JDBC

# Qu'est ce qu'une Servlet

---

- Composant logiciel écrit en Java fonctionnant du coté serveur
- Au même titre nous trouvons
  - CGI (Common Gateway Interface)
  - Langages de script coté serveur PHP, ASP (Active Server Pages)
- Permet de gérer des requêtes HTTP et de fournir au client une réponse HTTP
- Une Servlet s'exécute dans un **moteur de Servlet** ou **conteneur de Servlet** permettant d'établir le lien entre la Servlet et le serveur Web
- Une Servlet s'exécute par l'intermédiaire d'une machine virtuelle

# Architecture Servlets



Les Servlets peuvent être toutes gérées par des thread séparés au sein d'un même processus de machine virtuelle

## Ok, mais à quoi ça sert ?

---

- Créer des pages HTML dynamiques, générer des images, ...
- Effectuer des tâches de type CGI qui sont des traitements applicatifs coté serveur WEB
  - Manipulation d'une base de données
  - Gestion d'un système de surveillance, ...
- Respecter les principes d'une architecture : écrire une application en Java dont l'interface utilisateur est dans le client
  - Applet (SWING)
  - Téléphone portable (WAP)
  - Navigateur (HTML)

# Puissance des Servlets

---

- Portabilité
  - Technologie indépendante de la plate-forme et du serveur
  - Un langage (Java) et plusieurs plate-forme (.NET plusieurs langages et une plate-forme)
- Puissance
  - Disponibilité de l'API de Java
  - Manipulation d'images, connectivité aux bases de données (JDBC), ...
- Efficacité et endurance
  - Une Servlet est chargée une seule fois (CGI chargée puis déchargée après utilisation)
  - Une Servlet conserve son état (connexions à des bases de données)
- Sécurité
  - Typage fort de Java
  - Gestion des erreurs par exception

# Première Servlet : HelloWorld

Ne pas oublier d'importer la bibliothèque Java des Servlets

HelloWorld est un objet de type HttpServlet

Redéfinition de la méthode doGet (traitement d'une requête GET)

```
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 {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML>");
        out.println("<HEAD><TITLE>Bonjour tout le monde</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<BIG>Bonjour tout le monde</BIG>");
        out.println("</BODY></HTML>");
    }
}
```

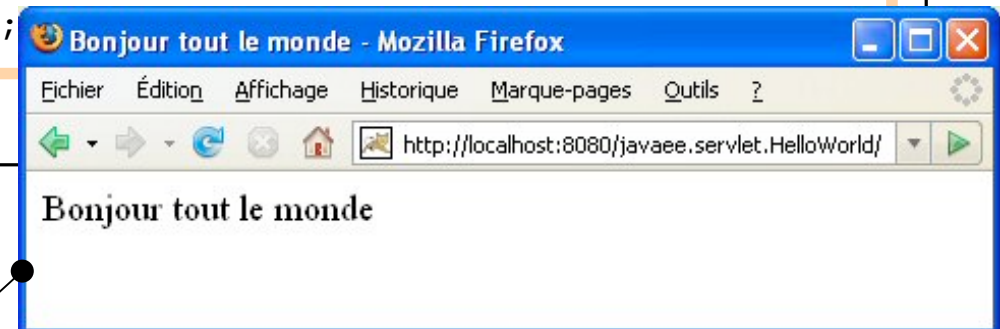
```
res.setContentType("text/html");
PrintWriter out = res.getWriter();

out.println("<HTML>");
out.println("<HEAD><TITLE>Bonjour tout le monde</TITLE></HEAD>");
out.println("<BODY>");
out.println("<BIG>Bonjour tout le monde</BIG>");
out.println("</BODY></HTML>");
```

Réponse sous format HTML

HelloWorld.java du projet HelloWorldServlet

Le résultat sur le client



# L'API Servlet : du générique à l'HTTP

---

- Une Servlet doit implémenter l'interface *javax.servlet.Servlet* et *javax.servlet.ServletConfig*

Servlet << Interface >>
+ <code>init(...)</code>
+ <code>service(...)</code>
+ <code>destroy()</code>

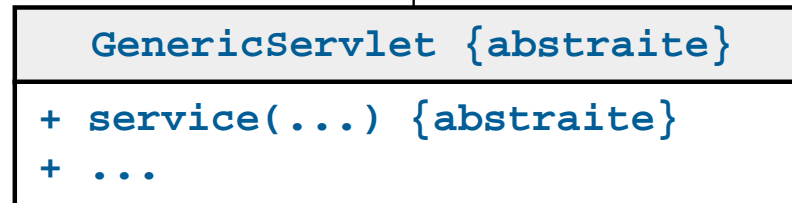
ServletConfig << Interface >>
+ <code>getInitParameter(String) : String</code>
+ <code>getServletName() : String</code>
+ ...

- Plus simplement l'API Servlet fournit deux classes qui proposent déjà une implémentation
  - *GenericServlet* : pour la conception de Servlets indépendantes du protocole
  - *HttpServlet* : pour la conception de Servlets spécifiques au protocole HTTP

# L'API Servlet : du générique à l'HTTP



Étendre cette classe pour construire des Servlets "génériques"



Étendre cette classe pour construire des Servlets propre au protocole HTTP

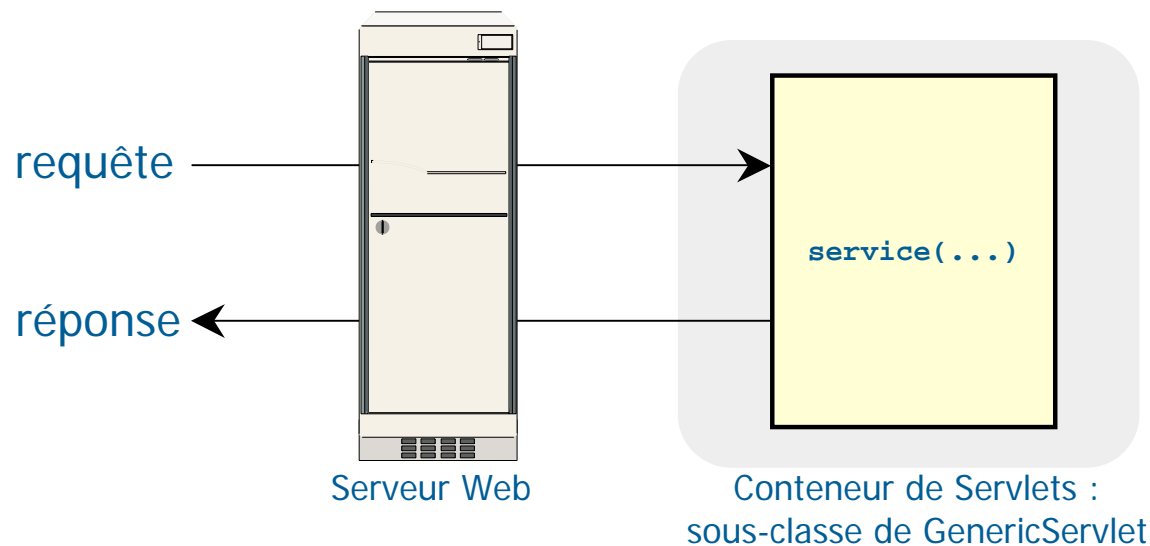


# L'API Servlet : la classe `GenericServlet`

- Une Servlet qui hérite de *GenericServlet* est une Servlet indépendante du protocole

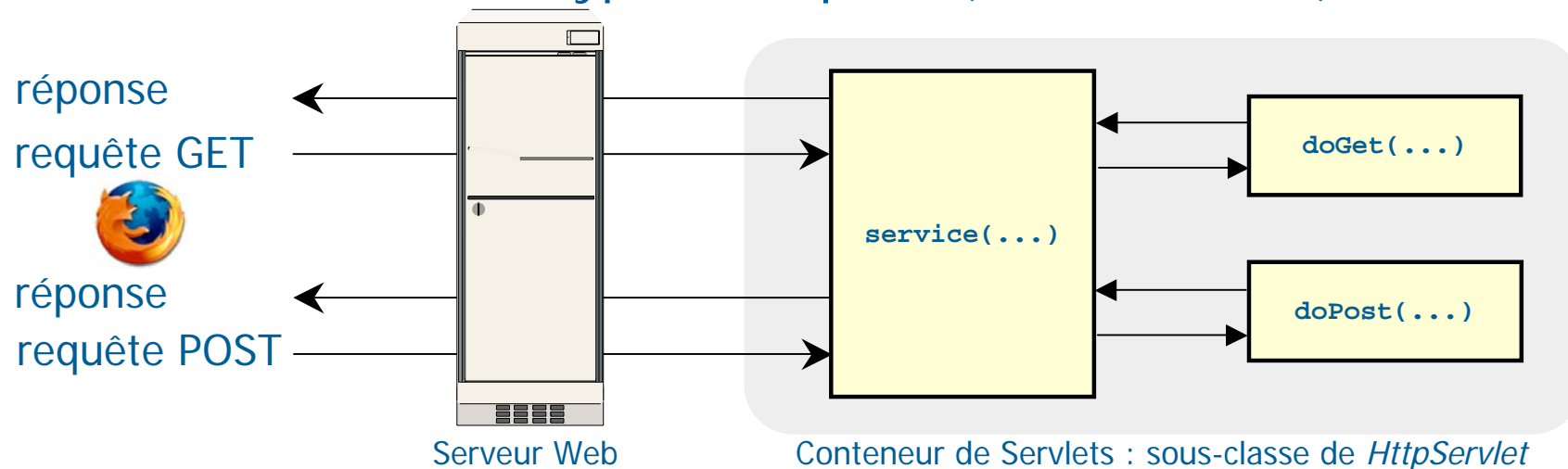
```
GenericServlet {abstraite}
+ service(...) {abstraite}
+ ...
```

- Obligation d'implémenter la méthode `service(...)` qui reste le principal point d'entrée du client vers le serveur
- Besoin de vérifier explicitement le type de protocole



# L'API Servlet : la classe `HttpServlet`

- Dans la suite du cours nous allons utiliser uniquement des Servlets qui réagissent au protocole HTTP d'où l'utilisation de la classe *HttpServlet*
- *HttpServlet* redéfinit la méthode `service(...)`
  - `service(...)` lit la méthode (GET, POST, ...) à partir de la requête
  - Elle transmet la requête à une méthode appropriée de *HttpServlet* destinée à traiter le type de requête (GET, POST, ...)



# HttpServlet : méthodes de traitement de requêtes

---

- Plusieurs méthodes sont fournies pour traiter les différents types de requêtes (GET, POST, ...).
- Elles sont appelées **méthodes de traitement de requêtes**
- Elles ont un en-tête identique *doXXX(...)* où XXX correspond au type de requête
  - *doPost(...)* est la méthode pour traiter les requêtes de type POST
  - *doGet(...)* est la méthode pour traiter les requêtes de type GET
  - *doHead(...)*, *doTrace(...)*, ...
- Selon le type de requête (GET ou POST) le concepteur redéfinit la méthode concernée



Pour les besoins du cours nous utiliserons essentiellement les méthodes *doPost(...)* et *doGet(...)*

# HttpServlet : méthodes de traitement de requêtes

- L'implémentation par défaut des méthodes *doXXX(...)* renvoie une erreur de type HTTP 405
  - Type de requête non supporté par l'URL

Appel du code *doGet(...)* de la super-classe

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorldError extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        super.doGet(req, res);
    }
}
```



*HelloWorldError.java*  
du projet  
**HelloWorldServlet**

**Ne vous trompez pas de  
méthode à redéfinir selon  
le type de requête**

# HttpServlet : requête et réponse

- La méthode *service(...)* et par conséquent les méthodes de traitement de requêtes (ex : *doPost(...)*) sont appelées
  - un objet **requête**
  - un objet **réponse**

Objet de requête

Objet de réponse

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SampleServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ...
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ...
    }
}
```

# HttpServletRequest : objet requête HttpServletRequest

---

- *HttpServletRequest* hérite de *ServletRequest*
- Cet objet encapsule la requête HTTP et fournit des méthodes pour accéder aux informations
  - du client
  - de l'environnement du serveur
- Exemples de méthodes
  - *String getMethod()* : retourne le type de requête
  - *String getServerName()* : retourne le nom du serveur
  - *String getParameter(String name)* : retourne la valeur d'un paramètre
  - *String[] getParameterNames()* : retourne le nom des les paramètres
  - *String getRemoteHost()* : retourne l'IP du client
  - *String getServerPort()* : retourne le port sur lequel le serveur écoute
  - *String getQueryString()* : retourne la chaîne d'interrogation
  - ... (voir l'API Servlets pour le reste)

# HttpServlet : objet requête HttpServletRequest

- Exemple : Servlet qui affiche un certain nombre d'informations issues de *HttpServletRequest*

```
public class InfosServlet extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        response.setContentType("text/plain");  
        PrintWriter out= response.getWriter();  
        out.println("Protocol: " + request.getProtocol());  
        out.println("Scheme: " + request.getScheme());  
        out.println("ServerName: " + request.getServerName());  
        out.println("ServerPort: " + request.getServerPort());  
        out.println("RemoteAddr: " + request.getRemoteAddr());  
        out.println("RemoteHost: " + request.getRemoteHost());  
        out.println("Method: " + request.getMethod());  
    }  
}
```

Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

http://localhost:8080/javaee.servlet.InfosServlet/

Protocol: HTTP/1.1 Scheme: http  
ServerName: localhost ServerPort: 8080  
RemoteAddr: 127.0.0.1 RemoteHost: 127.0.0.1  
Method: GET requestURL: /javaee.servlet.InfosServlet/  
ServletPath: /index.html PathInfo: null  
PathTranslated: null QueryString: null  
RemoteUser: null AuthType: null

*InfosServlet.java* du projet **InfosServlet**

# HttpServletRequest : objet réponse HttpServletResponse

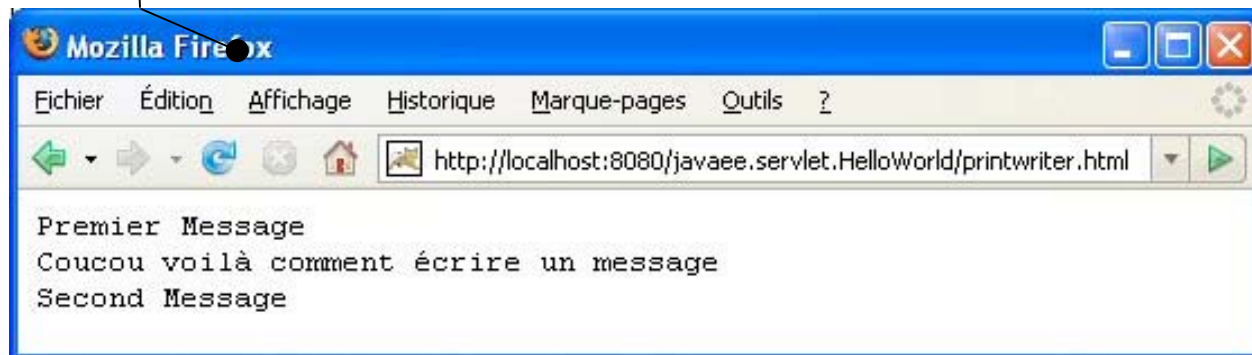
---

- *HttpServletResponse* hérite de *ServletResponse*
- Cet objet est utilisé pour construire un message de réponse HTTP renvoyé au client, il contient
  - les méthodes nécessaires pour définir le type de contenu, en-tête et code de retour
  - un flot de sortie pour envoyer des données (par exemple HTML) au client
- Exemples de méthodes
  - *void setStatus(int)* : définit le code de retour de la réponse
  - *void setContentType(String)* : définit le type de contenu MIME
  - *ServletOutputStream getOutputStream()* : flot pour envoyer des données binaires au client
  - *void sendRedirect(String)* : redirige le navigateur vers l'URL

# HttpServlet : objet réponse HttpServletResponse

- Exemple 1 : écrit un message de type TEXT au client
  - Utilisation de la méthode *PrintWriter* `getWriter()`

```
public class HelloWorldPrintWriter extends HttpServlet {  
  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
  
        PrintWriter out = res.getWriter();  
  
        out.println("Premier Message");  
        ● out.println("Coucou voilà comment écrire un message");  
        out.println("Second Message");  
    }  
}
```



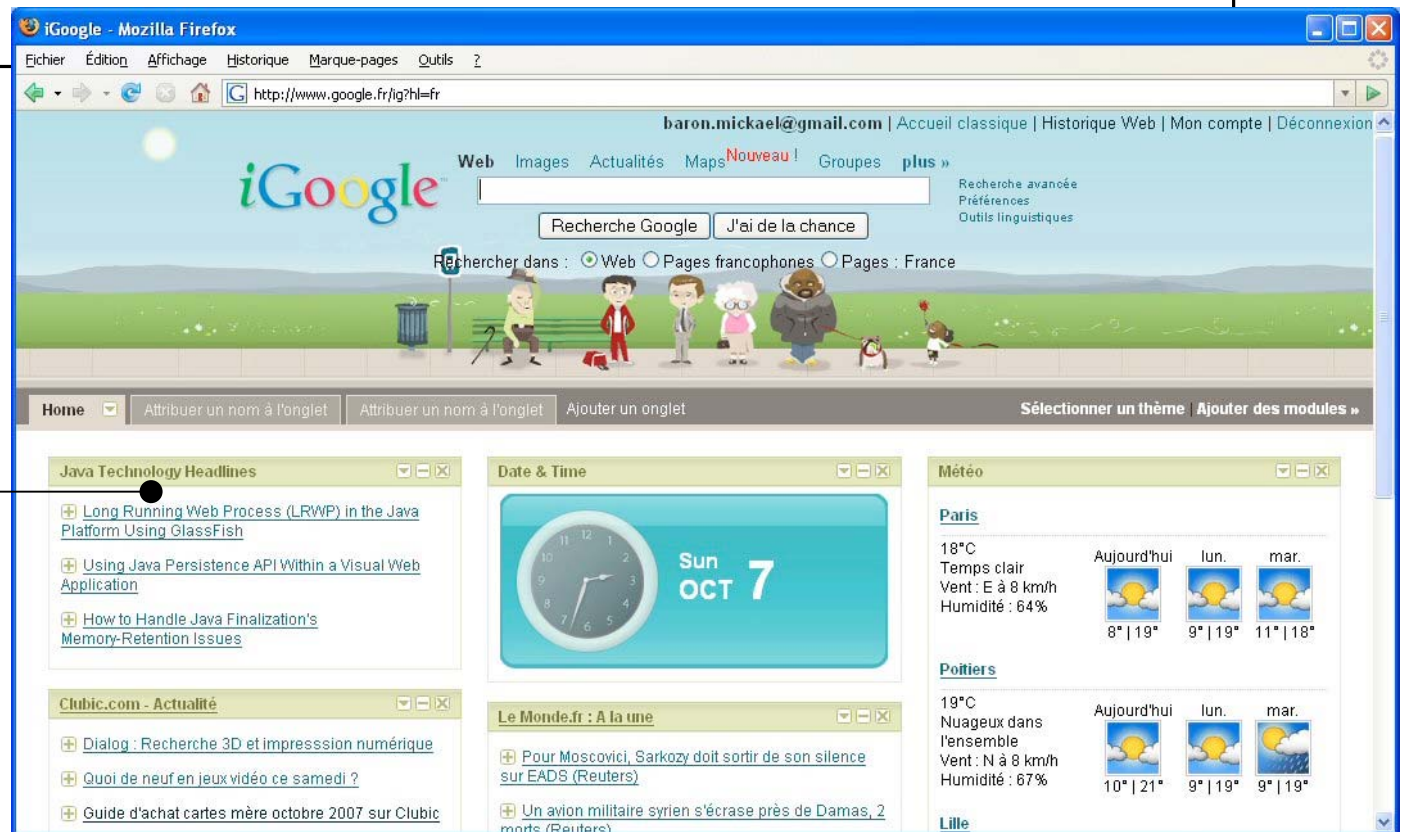
*HelloWorldPrintWriter.java*  
du projet **HelloWorld**

# HttpServlet : objet réponse HttpServletResponse

- Exemple 2 : effectue une re-direction vers un site web
  - Utilisation de la méthode `sendRedirect(String)`

```
public class SendRedirect extends HttpServlet {  
    public void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.sendRedirect("http://www.google.fr");  
    }  
}
```

*SendRedirect.java* du  
projet **HelloWorld**



# HttpServlet : objet réponse HttpServletResponse

## ➤ Exemple 3 : effectue un téléchargement de fichier sur le client

```
public class DownloadFileServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        try {
            InputStream is = new FileInputStream("c:/dd.txt");
            OutputStream os = response.getOutputStream();

            response.setContentType("text/plain");
            response.setHeader("Content-Disposition", "attachment; filename=toto.txt");

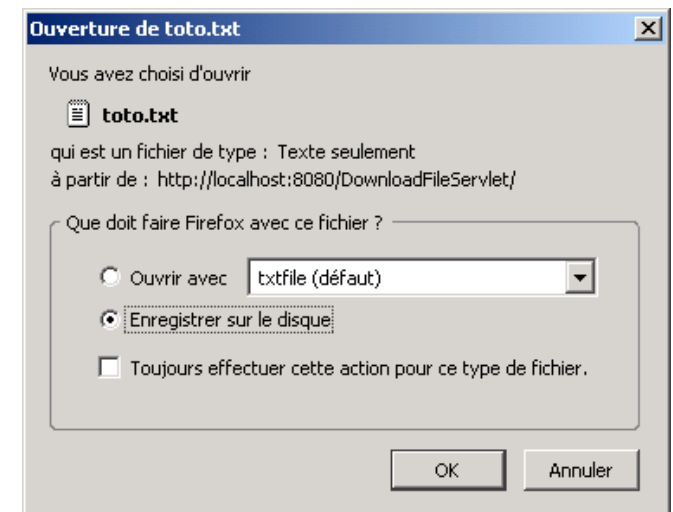
            int count;
            byte buf[] = new byte[4096];
            while ((count = is.read(buf)) > -1)
                os.write(buf, 0, count);
            is.close();
            os.close();
        } catch (Exception e) {
            // Y a un problème.
        }
    }
}
```

Le fichier à télécharger se trouve sur le serveur

Flux de sortie = client

En-tête de la réponse adaptée pour retourner un fichier

*DownloadFileServlet.java* du projet **DownloadFileServlet**



# HttpServlet : objet réponse HttpServletResponse

## ➤ Exemple 4 : effectue un pull client

```
public class PullClientServlet extends HttpServlet {
    private int count = 10;

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/plain");
        PrintWriter out = resp.getWriter();

        if (count > 0) {
            resp.setHeader("Refresh", "1"); ●
            count--;
            out.println(count + "...");
        } else {
            out.println("Fin");
        }
    }
}
```

Toutes les 1 seconde  
la page est rechargée  
et cela 10 fois de suite

*PullClientServlet.java* du projet  
**PullClient**

# HttpServlet : objet réponse HttpServletResponse

## ➤ Exemple 5 : effectue un push serveur

```
public class PushServerServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        ServletOutputStream out = res.getOutputStream();
        res.setContentType("multipart/x-mixed-replace;boundary=End");
        out.println();
        out.println("--End");

        for (int i = 10; i > 0; i--) {
            out.println("Content-Type: text/plain");
            out.println();
            out.println(i + "...");
            out.println();
            out.println("--End");
            out.flush();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        out.println("Content-Type: text/plain");
        out.println();
        out.println("Fin");
        out.println("--End--");
        out.flush();
    }
}
```

Utilisation de  
« multipart/mixed »  
pour coder le pushing  
serveur

Connexion non fermée  
mais réponse envoyée  
au client

Connexion fermée  
réponse envoyée au  
client

*PushServerServlet.java* du  
projet **PushServer**

# Servlets et formulaires : du coté HTML

---

- Utilisation de la balise `<FORM>` `</FORM>`
  - Option `METHOD` : type de requête GET ou POST
  - Option `ACTION` : l'URL où envoyer les données
- Utilisation de composants IHM pour saisir des informations
  - Contenu à l'intérieur de la balise `FORM`

A screenshot of an HTML form containing several input elements: a text input with the value "0549498070", a text area containing "blabla", a checked checkbox labeled "case à cocher", an unchecked radio button labeled "bouton radio", and a dropdown menu with the text "Element 1".

- Chaque composant est défini au moyen d'une balise particulière `SELECT`, `INPUT`, `TEXTAREA`, ...
- A l'intérieur de chaque balise du composant (`SELECT` par exemple) plusieurs options et notamment une (`NAME`) qui permet d'identifier le composant : `NAME="mon_composant"`
- Les données sont envoyées quand l'utilisateur clique sur un bouton de type `SUBMIT`

# Servlets et formulaires : du coté HTML

```
<body>
<p>Formulaire de satisfaction du cours Servlet/JSP </p>
<form name="form1" method="get" action="form.html">
  <p>
    Nom : <input type="text" name="nom">
    Prénom : <input type="text" name="prenom">
  </p>
  <p>
    Sexe :
    <input type="radio" name="radiol" value="sexe" checked>masculin
    <input type="radio" name="radiol" value="sexe">féminin
  </p>
  <p>Commentaire :<br>
    <textarea name="textarea" cols="50" rows="10"> </textarea><br>
    <input type="submit" name="Submit" rows="5" value="Soumettre">
  </p>
</form>
```

Le formulaire appelle une Servlet avec une requête de type GET

Formulaire de satisfaction du cours Servlet/JSP

Nom: BARON Prénom: Mickaël

Sexe :  masculin  féminin

Commentaire

Ce cours est magnifiquement réalisé, toutefois les étudiants discutent un peu trop

Soumettre

*index.html* du projet  
**UIForm**

**Il se peut que des composants portent le même identifiant. Exemple : composant bouton radio**

# Servlets et formulaires : du côté Servlet

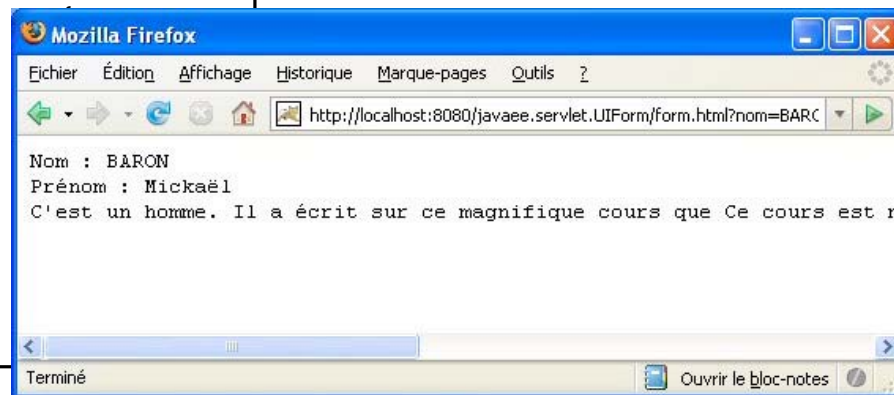
- Pour lire les données du formulaire : traiter la requête
- Accéder par l'intermédiaire de l'objet *HttpServletRequest* aux paramètres
  - *String getParameter(String p)* : retourne la valeur du paramètre p
  - *String[] getParameterValues(String p)* : retourne les valeurs du paramètre p

```
public class UIFormServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println("Nom : " + req.getParameter("nom"));
        out.println("Prénom : " + req.getParameter("prenom"));

        if (req.getParameterValues("radio1")[0].equals("mas"))
            out.print("C'est un homme. Il");
        else {
            out.print("C'est une femme. Elle");
        }

        out.print(" a écrit sur ce magnifique cours que ");
        out.println(req.getParameter("textarea"));
    }
}
```

Cette méthode est utile lorsque dans un formulaire vous avez plusieurs composants qui portent le même nom

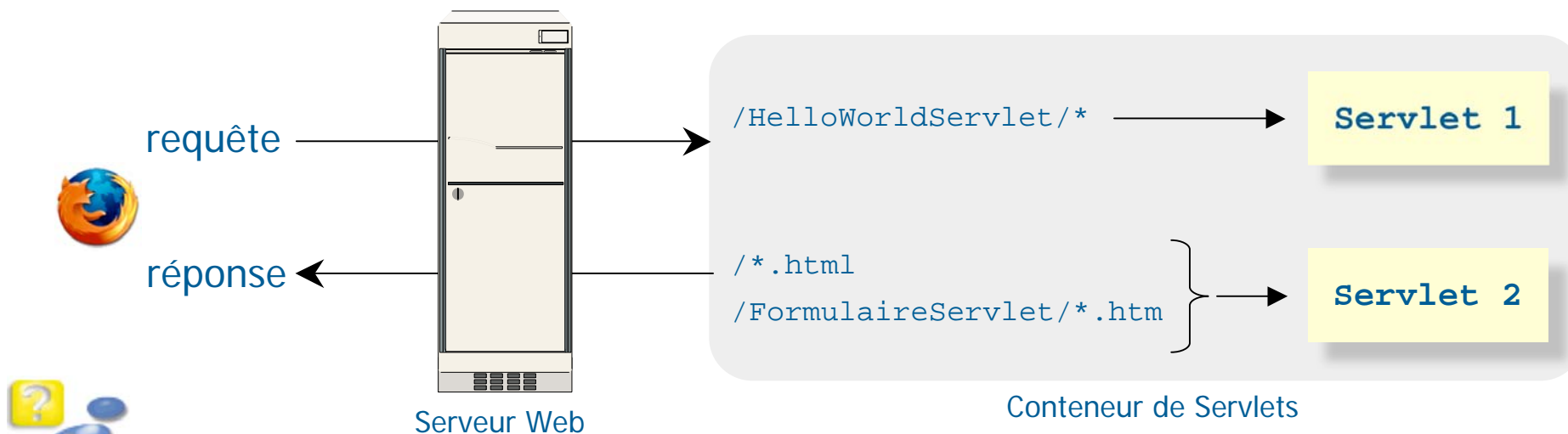


UIFormServlet.java du projet

UIForm

# Architecture de développement d'une application WEB

- Une application WEB peut contenir plusieurs Servlets
- Pour tester et déployer une Servlet, il faut un système d'exécution appelé **conteneur de Servlets** ou **moteur de Servlets**
- Le conteneur réalise le lien entre la Servlet et le serveur WEB
  - Transforme code Java (bytecode) en HTML
  - Associe à des URL's virtuels une Servlet

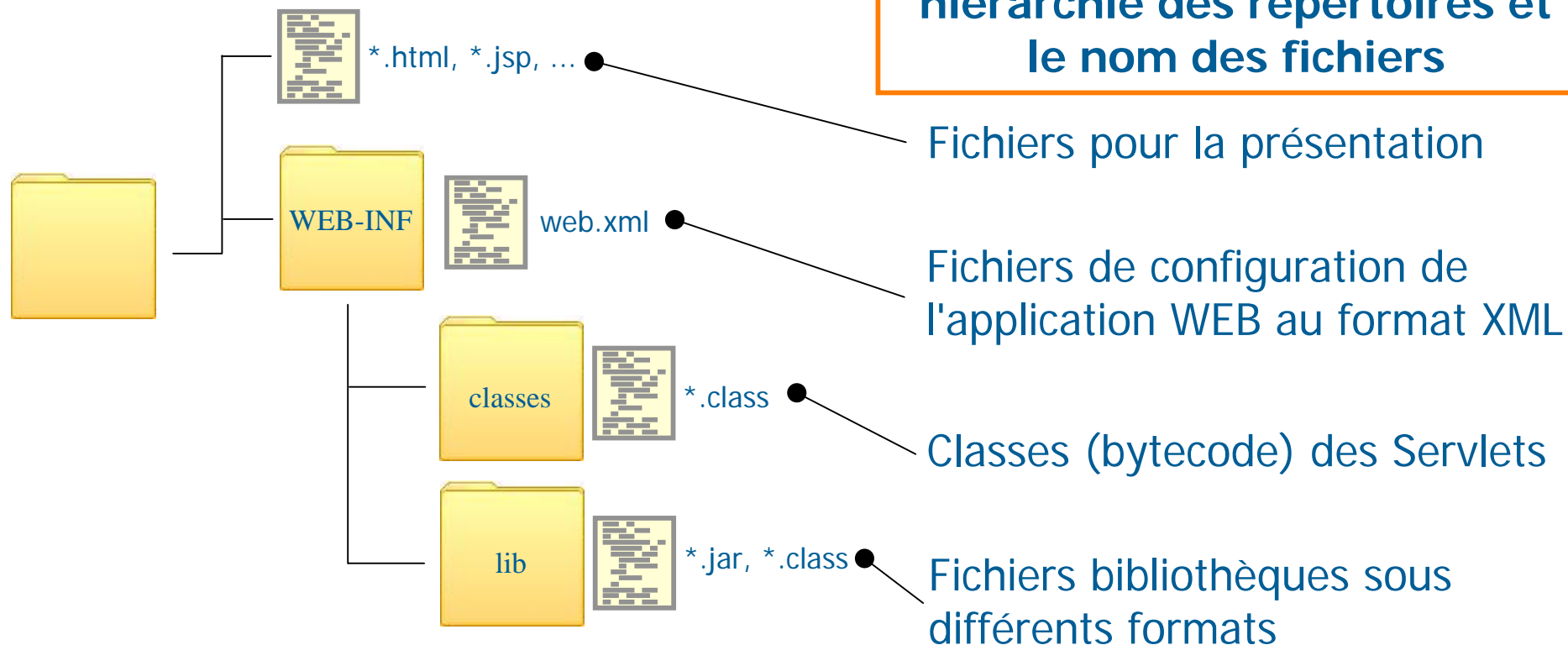


Il existe de nombreux conteneurs de Servlets présentés dans la partie suivante

# Architecture de développement d'une application WEB

- Une application WEB est contenue dans un répertoire physique sur le serveur
- Une application WEB possède une hiérarchie de répertoires et de fichiers

**Respecter scrupuleusement la hiérarchie des répertoires et le nom des fichiers**



## Les contextes ...

---

- Un **contexte** constitue une vue sur le fonctionnement d'une même application web
- Possibilité d'accéder à chacune des ressources de l'application web correspondant au contexte

- Servlets ●

Étudier à la fin de cette partie  
« Collaboration de Servlets »

- Pages JSP ●

Approfondissement sur la partie JSP  
« Collaboration JSP Servlets : MVC »

- ...

- Utilisation du fichier **web.xml** pour la description de déploiement du contexte

**Une application WEB ne  
contient qu'un seul  
contexte**

# Le fichier web.xml : c'est quoi ?

---

- Le fichier **web.xml** regroupe les informations de fonctionnement de l'application WEB (description de déploiement du contexte)
- Utilisation du format XML
  - eXtensible Markup Language
  - Syntaxe universelle pour la structuration des données créée par le World Wide Web Consortium (W3C) en 1996
  - Langage basé sur des balises permettant de donner une signification au document (HTML s'intéresse essentiellement à l'affichage)
  - Extensible par la possibilité de créer de nouvelles balises
  - Balises XML sensibles à la case et à la rigueur (balise ouvrante = obligation d'une balise fermante)
  - La structure du document est défini par un fichier XSD (XML Schema Description)

**Des informations sur le format  
XML peuvent être trouvées à  
[www.w3.org/XML](http://www.w3.org/XML)**



# Le fichier web.xml : c'est quoi ?

## ► Fichier web.xml typique

La présence du fichier web.xml est obligatoire pour que vos servlets WEB fonctionnent

En-tête du fichier web.xml

Balise principale

Balise de description de l'application WEB

Balise de description d'une Servlet

Nom de la Servlet "Identification"

Classe de la Servlet

Définition d'un chemin virtuel

Nom de la Servlet considéré "Identification"

Définition du chemin virtuel

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <display-name>Application WEB affichant HelloWorld</display-name>
  <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/msg.hello</url-pattern>
  </servlet-mapping>
</web-app>
```

# Le fichier web.xml : c'est quoi ?

- Possibilité de décrire dans le fichier **web.xml** le fonctionnement de plusieurs Servlets

*web.xml* du projet  
**HelloWorld**

L'application WEB est  
ici composée de deux  
Servlets

Deux façon différentes  
d'appeler la Servlet  
*HelloWorldServlet*

Une seule façon d'appeler  
la Servlet  
*HelloWorldPrintWriter*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>

  <display-name>Servlets affichant différemment le message
    HelloWorld
  </display-name>

  <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>HelloWorldPrintWriter</servlet-name>
    <servlet-class>HelloWorldPrintWriter</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>*.toutpourservlet</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/msg.hello</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>HelloWorldPrintWriter</servlet-name>
    <url-pattern>/printwriter.html</url-pattern>
  </servlet-mapping>
</web-app>
```

# Le fichier web.xml : c'est quoi ?

- Le fichier **web.xml** permet la définition de chemin virtuel : mais comment sont-ils utilisés pour appeler les Servlets ?

web.xml du projet  
**HelloWorld**

Trois chemins virtuels ont été définis pour exécuter la Servlet *HelloWorldServlet*

Adresse du  
Serveur

Port

Contexte de  
l'application  
WEB

```
...  
<servlet-mapping>  
  <servlet-name>HelloWorldServlet</servlet-name>  
  <url-pattern>/HelloWorldServlet/msg.hello</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
  <servlet-name>HelloWorldServlet</servlet-name>  
  <url-pattern>*.toutpourservlet</url-pattern>  
</servlet-mapping>  
<servlet-mapping>  
  <servlet-name>HelloWorldServlet</servlet-name>  
  <url-pattern>/index.html</url-pattern>  
</servlet-mapping>  
...
```

*http://localhost:8080/HelloWorldServlet*

*http://localhost:8080/HelloWorldServlet/***bonjour.toutpourservlet**

*http://localhost:8080/HelloWorldServlet/***hello.toutpourservlet**

*http://localhost:8080/HelloWorldServlet/***HelloWorldServlet/msg.hello**



La définition du contexte d'une application WEB sera présentée dans la partie suivante

# Cycle de vie d'une Servlet

---

- Création et initialisation de la Servlet
  - Utilisation de paramètres initiaux
- Traitement des services demandés par les clients (au travers de la méthode *service(...)* notamment), le cas échéant
  - Persistance d'instance
- Destruction de la Servlet et passage au ramasse-miettes de la machine virtuelle
  - Déconnexion avec une base de données



**Au contraire les applications serveur de type CGI sont créées à chaque requête et détruites après le traitement des réponses**

## Cycle de vie d'une Servlet : persistance d'instance

---

- Entre chaque requête du client les Servlets **persistent** sous forme d'**instances d'objet**
- Au moment où le code d'une Servlet est chargé, le serveur ne crée qu'**une seule instance** de classe
- L'instance (unique) traite chaque requête effectuée sur la Servlet
- Les avantages (rappels)
  - L'empreinte mémoire reste petite
  - Le surcoût en temps lié à la création d'un nouvel objet pour la Servlet est éliminé
  - La persistance est possible c'est-à-dire la possibilité de conserver l'état de l'objet à chaque requête (un exemple le compteur)

# Cycle de vie d'une Servlet : persistance d'instance

- Exemple : Servlet qui incrémente un compteur à chaque requête du client

```
public class SimpleCounterServlet extends HttpServlet {  
    private int count = 0;  
  
    protected void doGet(HttpServletRequest req, HttpServletResponse res)  
        throws ServletException, IOException {  
        res.setContentType("text/plain");  
        PrintWriter out = res.getWriter();  
        count++;  
  
        out.println("Depuis son chargement, on a accédé à cette Servlet " +  
            count + " fois.");  
    }  
}
```

**Pensez à prévoir des accès  
synchronisés au cas où plus  
d'une requête est traitée en  
même temps par le Serveur**



*SimpleCounterServlet.java* du  
projet **Counter**

# Cycle de vie d'une Servlet : rechargement d'une Servlet

---

- A chaque rechargement d'une Servlet par le conteneur de Servlet, il y a **création d'une nouvelle instance** et donc destruction de l'ancienne
- Le rechargement d'une Servlet a lieu quand il y a :
  - Modification d'au moins une **classe** de l'application WEB
  - Demande explicite de l'administrateur du serveur WEB
  - Redémarrage du conteneur de Servlets

**Le conteneur de Servlets ne s'intéresse qu'au Servlet et par conséquent si vous modifiez autre chose que des classes il n'y aura aucun rechargement implicite de Servlets**



## Cycle de vie d'une Servlet : méthodes `init()`

---

- Un constat : il n'y a **pas de constructeur** dans une Servlet
- L'initialisation des attributs se fait par l'intermédiaire de la méthode *init()*
  - Elle ne possède pas de paramètre
  - Définie et implémentée dans la classe abstraite *GenericServlet*
- La méthode *init()* peut être appelée à différents moments
  - Lorsque le conteneur de Servlets démarre
  - Lors de la première requête à la Servlet
  - Sur demande explicite de l'administrateur du serveur WEB

**N'essayez pas de placer des constructeurs à la mode POO ça ne sert à rien**



# Cycle de vie d'une Servlet : méthode init()

- Exemple : Servlet qui incrémente un compteur à chaque requête du client avec une valeur initiale de 6

```
public class InitCounterServlet extends HttpServlet {
    private int count;

    public void init() throws ServletException {
        count = 6;
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        count++;

        out.println("Depuis son chargement, on a accédé à cette Servlet " +
            count + " fois.");
    }
}
```

*InitCounterServlet.java* du  
projet **Counter**

# Cycle de vie d'une Servlet : méthode `init()`

- Possibilité d'utiliser des paramètres d'initialisation exploités exclusivement par la méthode `init()`
- Les paramètres d'initialisation sont définis dans le fichier **web.xml** de l'application WEB

Balise qui détermine le nom du paramètre

Balise qui détermine la valeur du paramètre

Balise qui explique le rôle du paramètre (optionnelle)

```
...
<web-app ...>
  <display-name>Servlet simulant un compteur</display-name>

  <servlet>
    <servlet-name>InitConfigFileCounterServlet</servlet-name>
    <servlet-class>InitConfigFileCounterServlet</servlet-class>
    <init-param>
      ● <param-name>initial_counter_value</param-name>
      ● <param-value>50</param-value>
      ● <description>Valeur initiale du compteur</description>
    </init-param>
  </servlet>
  ...
</web-app>
```

*web.xml* du projet Counter

**Plusieurs paramètres peuvent être définis mais attention à l'ordre des balises !!**



# Cycle de vie d'une Servlet : méthode init()

- Exemple : Servlet qui incrémente un compteur et qui est initialisée par le fichier **web.xml**

```
public class InitConfigFileCounterServlet extends HttpServlet {
    private int count;

    public void init() throws ServletException {
        String initial = this.getInitParameter("initial_counter_value");
        try {
            count = Integer.parseInt(initial);
        } catch (NumberFormatException e) {
            count = 0;
        }
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ...
        count++;

        out.println("Depuis son chargement, on a accédé à cette Servlet " +
            count + " fois.");
    }
}
```

*InitConfigFileCounterServlet.java*  
du projet **Counter**

## Cycle de vie d'une Servlet : méthode `destroy()`

---

- Une Servlet doit libérer toutes les ressources qu'elle a acquises et qui ne pourront être passées au ramasse-miettes
- Exemples de ressources
  - Connexion à une base de données
  - Ouverture d'un fichier sur le disque serveur
- La méthode `destroy()` donne une dernière chance d'écrire les informations qui ne sont pas encore sauvegardées
- La méthode `destroy()` est également utilisée pour écrire les informations persistantes qui seront lues lors du prochain appel à `init()`

## Cycle de vie d'une Servlet : méthode `destroy()`

- Exemple : Servlet qui incrémente un compteur qui sauvegarde l'état avec `destroy()` et qui recharge avec `init()`

```
public class InitDestroyCounterServlet extends HttpServlet {
    private int count;

    public void destroy() {
        FileWriter fileWriter = null; PrintWriter printWriter = null;
        try {
            fileWriter = new FileWriter("InitCounter.initial");
            printWriter = new PrintWriter(fileWriter);
            printWriter.println(count);
        } catch (IOException e) {
            ...
        } finally {
            if (printWriter != null) printWriter.close();
        }
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ...
    }
}
```

*InitDestroyCounterServlet.java*

du projet **Counter**

## Cycle de vie d'une Servlet : méthode `destroy()`

- Exemple (suite) : Servlet qui incrémente un compteur qui sauvegarde l'état avec `destroy()` et qui recharge avec `init()`

```
...
public void destroy() {
    // Voir transparent précédent
}

public void init() throws ServletException {
    FileReader fileReader = null;
    BufferedReader bufferedReader = null;
    try {
        fileReader = new FileReader("InitCounter.initial");
        bufferedReader = new BufferedReader(fileReader);
        String initial = bufferedReader.readLine();
        count = Integer.parseInt(initial);
    } catch (IOException e) {
        ...
    } finally {
        if (bufferedReader != null)
            bufferedReader.close();
    }
}
}
```

*InitDestoryCounterServlet.java*

du projet **Counter**

## Envoyer un contenu multimédia

---

- Pour l'instant nous avons écrit des Servlets qui retournaient des contenus HTML
- Besoin de retourner des contenus différents :
  - Protocole WAP et langage WML utilisés par les téléphones portables
  - Génération de contenus multimédias (création de graphes, manipulation d'images)
- L'API Java facilite la gestion des contenus multimédias en proposant des bibliothèques
  - Encodage d'images sous différents formats (GIF, JPEG)
  - Manipulation et traitement d'images

# Envoyer un contenu multimédia

- Exemple : Servlet qui génère et retourne une image JPEG affichant le message « Bonjour tout le monde »

```
public class GenerateImageServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ServletOutputStream out = res.getOutputStream();
        Frame frame = null; Graphics2D g = null; BufferedImage bim;

        frame = new Frame(); frame.addNotify();
        buffer_image = new BufferedImage(620, 60, BufferedImage.TYPE_3BYTE_BGR);

        g = buffer_image.createGraphics();
        g.setFont(new Font("Serif", Font.ITALIC, 48));
        g.drawString("Bonjour tout le monde !", 10,50);

        JPEGImageEncoder encode = JPEGCodec.createJPEGEncoder(out);
        encode.encode(buffer_image);
        out.close();
    }
}
```



*GenerateImageServlet.java* du  
projet **Multimedia**

## Suivi de session

---

- Le protocole HTTP est un protocole sans état
- Impossibilité alors de garder des informations d'une requête à l'autre (identifier un client d'un autre)
- Obligation d'utiliser différentes solutions pour remédier au problème d'état
  - Authentifier l'utilisateur
  - Champs de formulaire cachés
  - Réécriture d'URL
  - Cookies persistants
  - Suivi de session

## Cookies persistants : Cookie

---

- Un cookie est une information envoyée au navigateur (client) par un serveur WEB qui peut ensuite être relue par le client
- Lorsqu'un client reçoit un cookie, il le sauve et le renvoie ensuite au serveur chaque fois qu'il accède à une page sur ce serveur
- La valeur d'un cookie pouvant identifier de façon unique un client, ils sont souvent utilisés pour le suivi de session
- Les cookies ont été introduits par la première fois dans Netscape Navigator
  - *[home.netscape.com/newsref/std/cookie\\_spec.html](http://home.netscape.com/newsref/std/cookie_spec.html)*
  - *[www.cookie-central.com](http://www.cookie-central.com)*

## Cookies persistants : Cookie

---

- L'API Servlet fournit la classe *javax.servlet.http.Cookie* pour travailler avec les Cookies
  - *Cookie(String name, String value)* : construit un cookie
  - *String getName()* : retourne le nom du cookie
  - *String getValue()* : retourne la valeur du cookie
  - *setValue(String new\_value)* : donne une nouvelle valeur au cookie
  - *setMaxAge(int expiry)* : spécifie l'âge maximum du cookie
- Pour la création d'un nouveau cookie, il faut l'ajouter à la réponse (*HttpServletResponse*)
  - *addCookie(Cookie mon\_cook)* : ajoute à la réponse un cookie
- La Servlet récupère les cookies du client en exploitant la réponse (*HttpServletRequest*)
  - *Cookie[] getCookies()* : récupère l'ensemble des cookies du site

# Cookies persistants : Cookie

---

- Code pour créer un cookie et l'ajouter au client

```
Cookie cookie = new Cookie("Id", "123");  
res.addCookie(cookie);
```

- Code pour récupérer les cookies

```
Cookie[] cookies = req.getCookies();  
if (cookies != null) {  
    for (int i = 0; i < cookies.length; i++) {  
        String name = cookies[i].getName();  
        String value = cookies[i].getValue();  
    }  
}
```

**Il n'existe pas dans l'API Servlet  
de méthode permettant de récupérer  
la valeur d'un cookie par son nom**



# Cookies persistants : Cookie

- Exemple : gestion de session (identifier un client d'un autre) par l'intermédiaire des cookies persistants

```
public class CookiesServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ...
        String sessionId = null;
        Cookie[] cookies = req.getCookies();
        if (cookies != null) {
            for (int i = 0; i < cookies.length; i++) {
                if (cookies[i].getName().equals("sessionId")) {
                    sessionId = cookies[i].getValue();
                }
            }
        }
        if (sessionId == null) {
            sessionId = new java.rmi.server.UID().toString();
            Cookie c = new Cookie("sessionId", sessionId);
            res.addCookie(c);
            out.println("Bonjour le nouveau");
        } else {
            out.println("Encore vous"); ... }
        }
    }
}
```

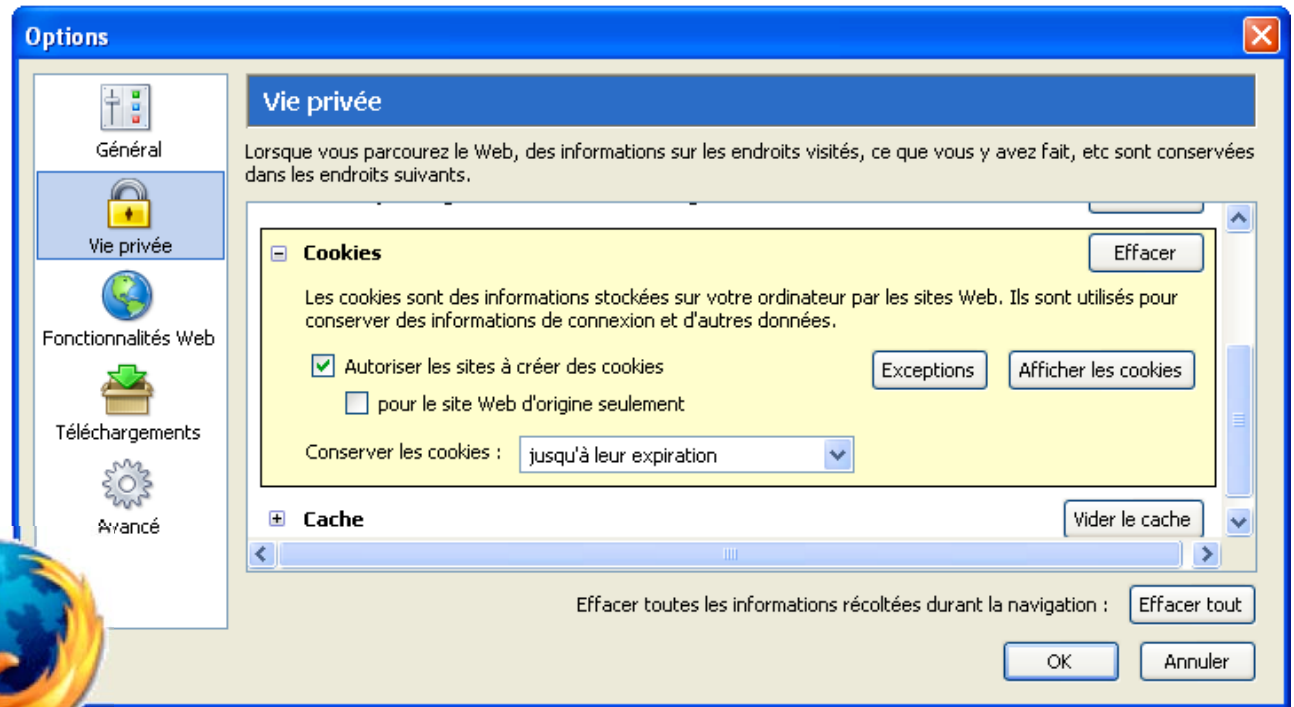
Génère un identifiant unique pour chaque client

*CookiesServlet.java* du projet  
Cookies

# Suivi de session : HttpSession

- Le plus gros problème des cookies est que les navigateurs ne les acceptent pas toujours

- L'utilisateur peut configurer son navigateur pour qu'il refuse ou pas les cookies



- Les navigateurs n'acceptent que 20 cookies par site, 300 par utilisateur et la taille d'un cookie peut être limitée à 4096 octets

# Suivi de session : HttpSession

- Solutions : utilisation de l'API de suivi de session *HttpSession*
- Méthodes de création liées à la requête (*HttpServletRequest*)
  - *HttpSession getSession()* : retourne la session associée à l'utilisateur
  - *HttpSession getSession(boolean p)* : création selon la valeur de *p*
- Gestion d'association (*HttpSession*)
  - *Enumeration getAttributeNames()* : retourne les noms de tous les attributs
  - *Object getAttribute(String name)* : retourne l'objet associé au nom
  - *setAttribute(String na, Object va)* : modifie *na* par la valeur *va*
  - *removeAttribute(String na)* : supprime l'attribut associé à *na*
- Destruction (*HttpSession*)
  - *invalidate()* : expire la session
  - *logout()* : termine la session

**Mécanisme très puissant permettant de stocker des objets et non de simples chaînes de caractères comme les cookies**

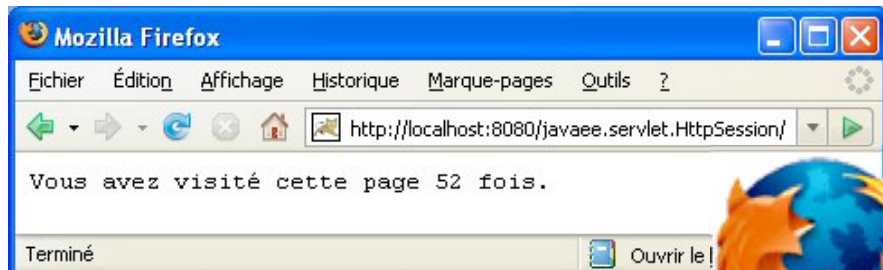


# Suivi de session : HttpSession

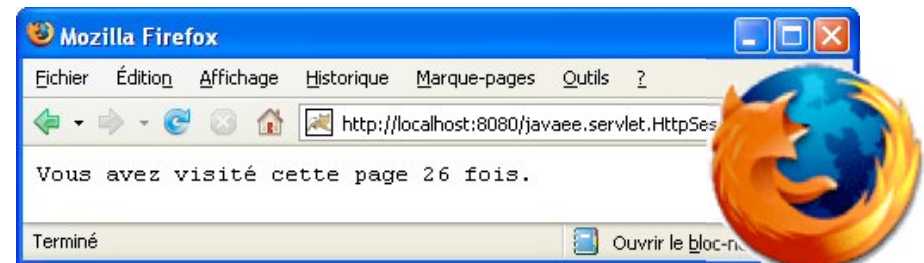
## ➤ Exemple : suivi de session pour un compteur

```
public class HttpSessionServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain"); PrintWriter out = res.getWriter();
        HttpSession session = req.getSession();
        Integer count = (Integer)session.getAttribute("count");
        if (count == null)
            count = new Integer(1);
        else
            count = new Integer(count.intValue() + 1);
        session.setAttribute("count", count);
        out.println("Vous avez visité cette page " + count + " fois.");
    }
}
```

*HttpSessionServlet.java*  
du projet HttpSession



Client n°1



Client n°2

# Collaboration de Servlets

---

- Les Servlets s'exécutant dans le même serveur peuvent dialoguer entre elles
- Deux principaux styles de collaboration
  - *Partage d'information* : un état ou une ressource.  
Exemple : un magasin en ligne pourrait partager les informations sur le stock des produits ou une connexion à une base de données
  - *Partage du contrôle* : une requête.  
Réception d'une requête par une Servlet et laisser l'autre Servlet une partie ou toute la responsabilité du traitement

**N'essayez pas de communiquer avec des Servlets de serveurs différents cette solution ne fonctionnera pas**



## Collaboration de Servlets : partage d'information

---

- La collaboration est obtenue par l'interface *ServletContext*
- L'utilisation de *ServletContext* permet aux applications web de disposer de son propre conteneur d'informations unique
- Une Servlet retrouve le *ServletContext* de son application web par un appel à *getServletContext()*
- Exemples de méthodes
  - *void setAttribute(String name, Object o)* : lie un objet sous le nom indiqué
  - *Object getAttribute(String name)* : retrouve l'objet sous le nom indiqué
  - *Enumeration getAttributeNames()* : retourne l'ensemble des noms de tous les attributs liés
  - *void removeAttribute(String name)* : supprime l'objet lié sous le nom indiqué

# Partage d'information

- Exemple : Servlets qui vendent des pizzas et partagent une spécialité du jour

Création de deux attributs

```
public class PizzasAdmin extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        ServletContext context = this.getServletContext();
        context.setAttribute("Specialite", "Jambon Fromage");
        context.setAttribute("Date", new Date());
        out.println("La pizza du jour a été définie.");
    }
}
```

*PizzasAdmin.java* du projet  
**ServletContext**

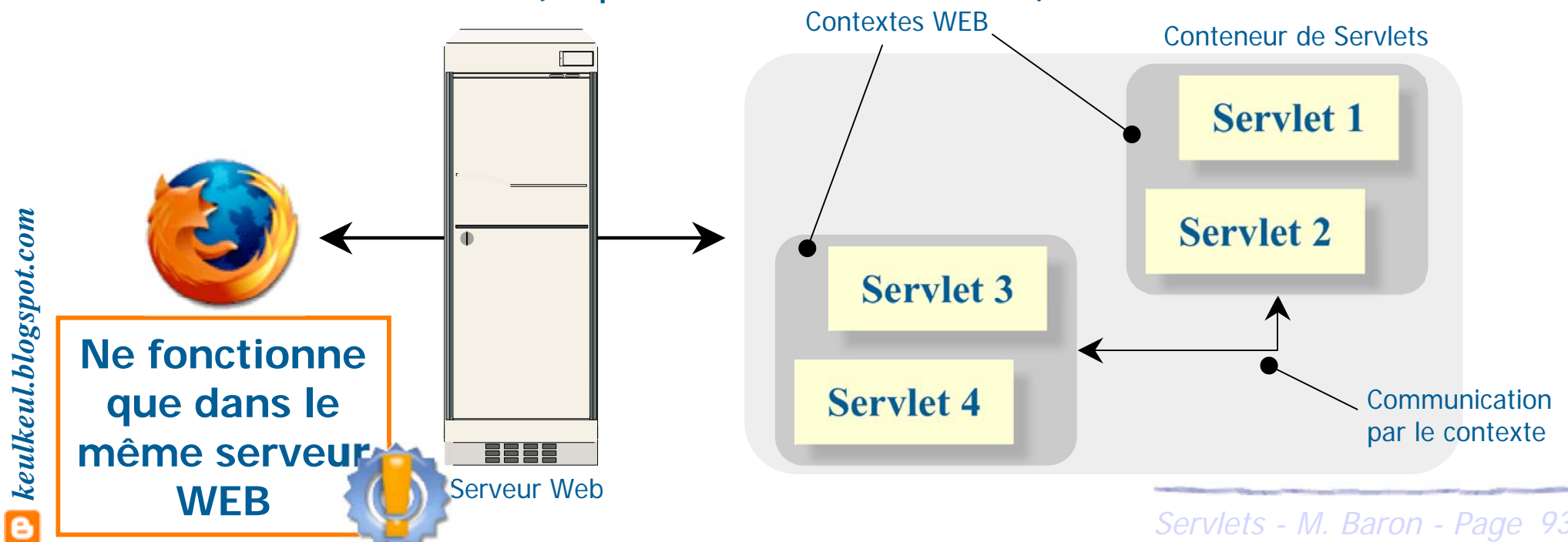
```
public class PizzasClient extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ...
        ServletContext context = this.getServletContext();
        String pizza_spec = (String)context.getAttribute("Specialite");
        Date day = (Date)context.getAttribute("Date");
        DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
        String today = df.format(day);
        out.println("Aujourd'hui (" + today + "), notre specialite est : " + pizza_spec);
    }
}
```

Lecture des attributs

*PizzasClient.java* du projet  
**ServletContext**

# Partage d'information

- Possibilité de partager des informations entre contextes web
- Première solution : utilisation d'un conteneur d'informations externes (une base de données par exemple)
- Seconde solution : la Servlet recherche un autre contexte à partir de son propre contexte
  - *ServletContext getContext(String uripath)* : obtient le contexte à partir d'un chemin URI (uripath = chemin absolu)



# Partage d'information

- Exemple : permet d'afficher la spécialité du jour de l'application web précédente

```
public class ReadSharePizzas extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        ServletContext my_context = this.getServletContext();
        ServletContext pizzas_context = my_context.getContext("/ServletContext");
        String pizza_spec = (String)pizzas_context.getAttribute("Specialite");
        Date day = (Date)pizzas_context.getAttribute("Date");

        DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
        String today = df.format(day);

        out.println("Aujourd'hui (" + today + "), notre specialite est : " + pizza_spec);
    }
}
```

Contextes de l'application  
web précédente

*ReadSharePizzas.java* du projet  
**CrossServletContext**



**Pour communiquer entre contextes,  
il faut autoriser la communication  
inter-contextes (voir partie suivante)**

# Collaboration de Servlets : partage du contrôle

---

- Les Servlets peuvent partager ou distribuer le contrôle de la requête
- Deux types de distribution
  - Distribuer un renvoi : une Servlet peut *renvoyer* une requête entière
  - Distribuer une inclusion : une Servlet peut *inclure* du contenu généré
- Les avantages sont
  - La délégation de compétences
  - Une meilleure abstraction et une plus grande souplesse
  - Architecture logicielle MVC (Servlet = contrôle et JSP = présentation)



**Collaboration entre Servlets /  
JSP dans la partie JSP**

# Collaboration de Servlets : partage du contrôle

---

- Le support de la délégation de requête est obtenu par l'interface *RequestDispatcher*
- Une Servlet obtient une instance sur la **requête**
  - *RequestDispatcher getRequestDispatcher(String path)* : retourne une instance de type *RequestDispatcher* par rapport à un composant
  - Un composant peut-être de tout type : Servlet, JSP, fichier statique, ...
  - *path* est un chemin relatif ou absolu ne pouvant pas sortir du contexte
- Pour distribuer en dehors du contexte courant il faut :
  - Identifier le contexte extérieur (utilisation de *getContext()*)
  - Utiliser la méthode *getRequestDispatcher(String path)*
  - Le chemin est uniquement en absolu



**De préférence utilisez la méthode  
*getRequestDispatcher* de  
*ServletRequest***

## Partage du contrôle : distribuer un renvoi

---

- La méthode *forward(...)* de l'interface *RequestDispatcher* renvoie une requête d'une Servlet à une autre ressource sur le serveur
  - *void forward(ServletRequest req, ServletResponse res)* : redirection de requête

```
RequestDispatcher dispat =  
    req.getRequestDispatcher("/index.html");  
dispat.forward(req, res);
```

- Possibilité de transmettre des informations lors du renvoi
  - en attachant une chaîne d'interrogation (au travers de l'URL)
  - en utilisant les attributs de requête via la méthode *setAttribute(...)*
- Les choses à ne pas faire ...
  - ne pas effectuer de modification sur la réponse avant un renvoi
  - ne rien faire sur la requête et la réponse après une distribution d'un renvoi

# Partage du contrôle : distribuer un renvoi

## ➤ Exemple : distribuer un renvoi de *Emetteur* à *Recepteur*

```
public class SenderServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        req.setAttribute("seconde", "au revoir");

        RequestDispatcher dispat = req.getRequestDispatcher("/recepteur.html?mot=bonjour");
        dispatcher.forward(req, res);
        // Ne rien faire sur req et res
    }
}
```

Transmission d'informations par attributs

Le chemin est absolu par rapport au contexte de l'application web

Transmission d'informations par chaîne d'interrogation

*SenderServlet.java* du projet  
**ForwardInclude**

```
public class ReceiverServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        out.println(req.getParameter("mot"));
        out.println(req.getAttribute("seconde"));
    }
}
```

Affichage des informations stockées dans la requête

*ReceiverServlet.java* du projet  
**ForwardInclude**

L'utilisation des attributs à la place des paramètres donne la possibilité de passer des objets et non des chaînes de caractères



## Partage du contrôle : distribuer un renvoi

---

- Nous avons vu au début de cette partie qu'il existait une méthode de redirection
  - *sendRedirect(...)* est une redirection effectuée par le client
  - *forward(...)* est une redirection effectuée par le serveur
- Est-il préférable d'utiliser *forward(...)* ou *sendRedirect(...)* ???
  - *forward(...)* est à utiliser pour la partage de résultat avec un autre composant sur le même serveur
  - *sendRedirect(...)* est à utiliser pour des redirections externes car aucune recherche *getContext(...)* n'est nécessaire



**Préférez *forward(...)* pour des redirections dans le contexte et *sendRedirect(...)* pour le reste**

## Partage du contrôle : distribuer une inclusion

---

- La méthode *include(...)* de l'interface *RequestDispatcher* inclut le contenu d'une ressource dans la réponse courante

```
RequestDispatcher dispat =  
    req.getRequestDispatcher("/index.html");  
dispat.include(req, res);
```

- La différence avec une distribution par renvoi est :
  - la Servlet appelante garde le contrôle de la réponse
  - elle peut inclure du contenu avant et après le contenu inclus
- Possibilité de transmettre des informations lors de l'inclusion
  - en attachant une chaîne d'interrogation (au travers de l'URL)
  - en utilisant les attributs de requête via la méthode *setAttribute(...)*
- Les choses à ne pas faire ...
  - ne pas définir le code d'état et en-têtes (pas de *setContentType(...)*)
  - supprimer les balises `<HTML>` et `<BODY>`

# Partage du contrôle : distribuer une inclusion

## ➤ Exemple : permet de distribuer une inclusion

```
public class IncluderServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML><BODY>");

        RequestDispatcher dispat = req.getRequestDispatcher("included.html");
        dispat.include(req, res); ●
        out.println("<br>");

        req.setAttribute("bonjour", "Bonjour");
        dispat.include(req, res); ●
        out.println("<br>");

        req.setAttribute("bonsoir", "Bonsoir");

        dispat.include(req, res); ●
        out.println("<br>");

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

Cette Servlet réalise trois inclusions

*IncluderServlet.java* du projet

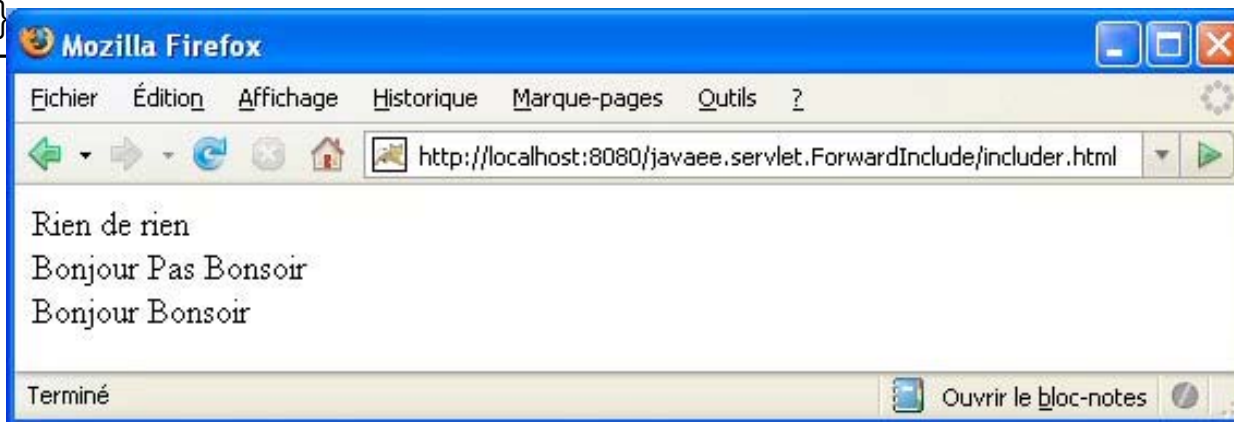
**ForwardInclude**

# Partage du contrôle : distribuer une inclusion

## ► Exemple (suite) : permet de distribuer une inclusion

```
public class IncludedServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter out = res.getWriter();
        if(req.getAttribute("bonjour") != null) {
            out.println(req.getAttribute("bonjour"));
            if (req.getAttribute("bonsoir") != null) {
                out.println(req.getAttribute("bonsoir"));
            } else {
                out.println("Pas Bonsoir");
            }
        } else {
            out.println("Rien de rien");
        }
    }
}
```

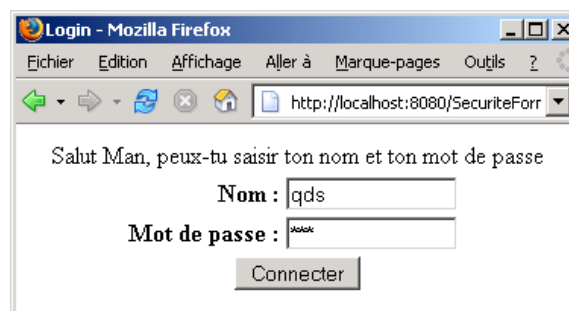
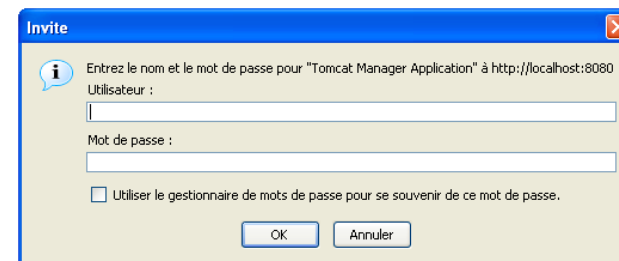
Retour à l'appelant



*IncludedServlet.java* du projet  
**ForwardInclude**

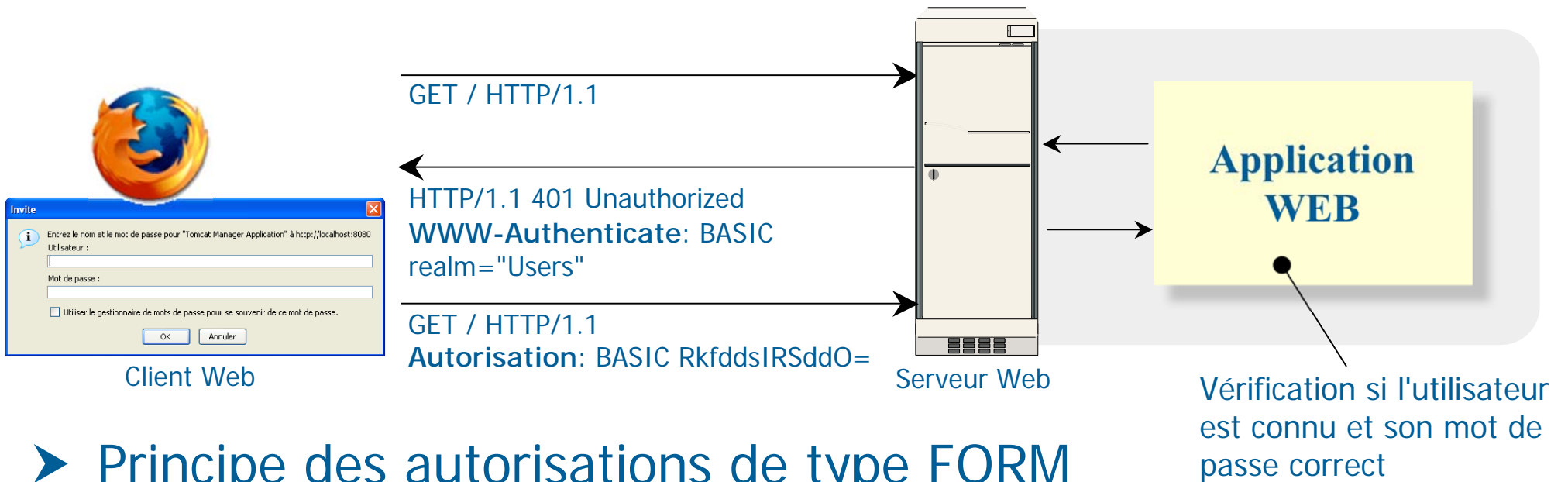
# Sécurité : authentification

- La sécurité consiste à conserver les informations sensibles dans les mains des utilisateurs
  - *Authentification* : capable de vérifier l'identité des parties impliquées
  - *Habilitation* : limiter l'accès aux ressources à un ensemble d'utilisateurs
  - *Confidentialité* : garantir la communication des parties impliquées
- Nous distinguons plusieurs types d'autorisation :
  - **BASIC** : fournit par le protocole HTTP basé sur un modèle simple de demande/réponse (codage Base64)
  - **FORM** : authentification ne reposant pas celle du protocole HTTP

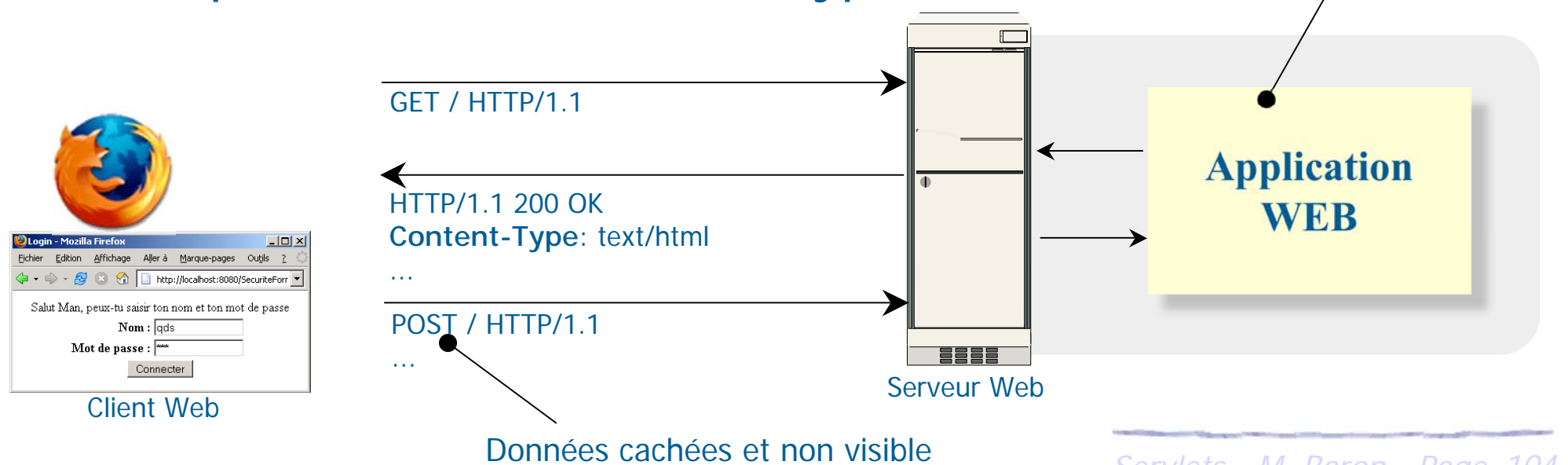


# Sécurité : types d'autorisation

## ► Principe des autorisations de type BASIC



## ► Principe des autorisations de type FORM



# Sécurité : deux familles d'identification pour les Servlets

---

- Gérée par le conteneur de Servlets (**Ident 1**) :
  - Spécification d'un domaine de sécurité dans le fichier de configuration **web.xml**
  - Les utilisateurs sont gérés (l'utilisateur existe-il, le mot de passe est-il correct, ...) **uniquement** par le conteneur de Servlets
    - Basée sur les rôles (BASIC)
    - A base de formulaire (FORM)
- Effectuée à l'intérieur des Servlets (**Ident 2**) :
  - Les utilisateurs sont stockés dans une base de données, un fichier, ...
  - La vérification est effectuée dans les Servlets (besoin d'un codage)
    - Personnalisée (BASIC)
    - Personnalisée à base de formulaire (FORM)

# Authentification basée sur les rôles : Ident 1

- Exemple : Servlet qui après identification affiche un ensemble d'informations

Aucune vérification dans le code de la Servlet pour l'identification

```
public class SecurityRolesServlet extends HttpServlet
protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();

    out.println("Bonjour : " + req.getRemoteUser());
    out.println("Information Top-Secrête");
    out.println("Type d'authentification : " + req.getAuthType());

    out.println("Est-un administrateur : " + req.isUserInRole("admin"));
}
}
```

*SecurityRolesServlet.java* du projet  
**AuthentificationByRoles**



Conteneur de Servlets

Invite

Entrez le nom et le mot de passe pour "Tomcat Manager Application" à http://localhost:8080

Utilisateur :

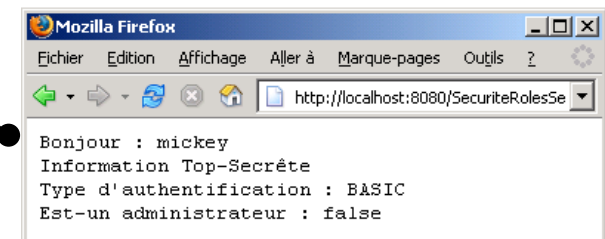
Mot de passe :

Utiliser le gestionnaire de mots de passe pour se souvenir de ce mot de passe.

OK Annuler

Identification réussie ?

Oui



# Authentification basée sur les rôles : Ident 1

- Besoin de modification du fichier de configuration **web.xml** de l'application WEB considérée

*web.xml* du projet  
**AuthentificationByRoles**

```
...
<web-app ...>
  <display-name>Gestion d'une authentification par "Roles"</display-name>
  <servlet>
    <servlet-name>SecuriteRolesServlet</servlet-name>
    <servlet-class>SecuriteRolesServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SecuriteRolesServlet</servlet-name>
    <url-pattern>/SecuriteRolesServlet</url-pattern>
  </servlet-mapping>

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>SecretProtection</web-resource-name>
      <url-pattern>/SecuriteRolesServlet</url-pattern>
      <http-method>GET</http-method>
    </web-resource-collection>

    <auth-constraint>
      <role-name>test</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Authentification de SecuriteRoleServlet</realm-name>
  </login-config>
</web-app>
```

Définition des Servlets  
contenus dans  
l'application WEB et des  
chemins virtuels

Définit pour quelles URL la  
contrainte d'identification  
doit être mise en œuvre

Protection des  
ressources pour la  
méthode GET

Rôle(s) ayant le droit  
d'accéder aux ressources de  
l'application WEB

« Habillage » de la  
boîte d'identification

La définition des rôles sera  
présentée dans la partie suivante



# Authentification personnalisée basée sur les rôles : Ident 1

- Exemple : Servlet qui après identification personnalisée affiche un message de bienvenue

Aucune vérification dans le code de la Servlet pour l'identification

```
public class SecurityFormRolesServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("Bonjour : " + req.getRemoteUser());
        req.getSession().invalidate();
    }
}
```

Termine explicitement la session. Obligation de s'identifier à chaque fois que la Servlet est appelée

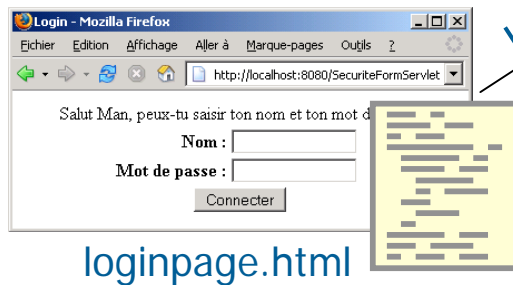
*SecurityFormRolesServlet.java* du projet **AuthentificationFormByRoles**

Conteneur de Servlets

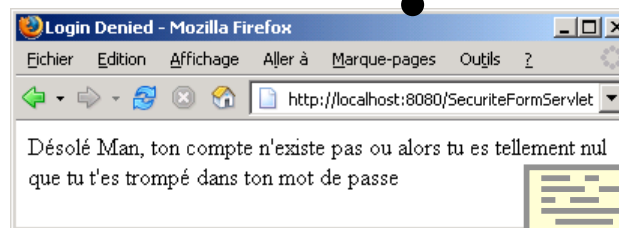
Identification réussie ?

Non

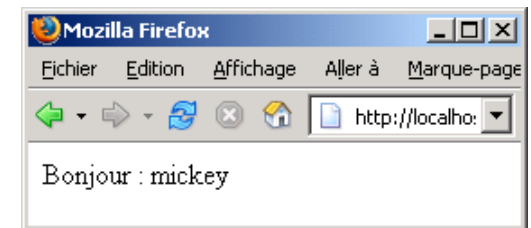
Oui



loginpage.html



errorpage.html



Deux fichiers au format HTML gèrent les pages de connexion et d'erreur

# Authentification personnalisée basée sur les rôles : Ident 1

- Besoin de modification du fichier de configuration **web.xml** de l'application WEB considérée

```
...
<web-app ...>
  ...
  <servlet> ... </servlet>
  <servlet-mapping> ... </servlet-mapping>
  <security-constraint> ... </security-constraint>

  <login-config>
    <auth-method>
      FORM
    </auth-method>
    <form-login-config>
      <form-login-page>
        /loginpage.html
      </form-login-page>
      <form-error-page>
        /errorpage.html
      </form-error-page>
    </form-login-config>
  </login-config>
</web-app>
```

Le début du fichier web.xml est identique au précédent modulo le nom de la classe *SecuriteFormServlet*

Balise qui stocke la page de connexion

Balise qui stocke la page des erreurs de connexion

web.xml du projet

## AuthentificationFormByRoles

Les fichiers HTML sont placés à la racine de l'application WEB et au même niveau que le répertoire WEB-INF



# Authentification personnalisée basée sur les rôles : Ident 1

- Le formulaire de la page *loginpage.html* doit employer
  - la méthode POST pour la transmission des données
  - des valeurs spéciales pour les noms des composants

Rend invisible la transmission des données

Le moteur de Servlet avec les informations contenues dans le fichier *web.xml* se charge de traiter l'identification

```
<FORM METHOD=POST ACTION="j_security_check">
```

```
  Nom d'utilisateur : <INPUT TYPE=TEXT NAME="j_username"><br>
```

```
  Mot de passe : <INPUT TYPE=PASSWORD NAME="j_password"><br>
```

```
  <INPUT TYPE=SUBMIT>
```

```
</FORM>
```

Valeur pour le nom d'utilisateur

Valeur pour le mot de passe

*loginpage.html* du projet  
**AuthenticationFormByRoles**

C'est du JSP



# Authentification personnalisée : Ident 2

## ➤ Exemple : Servlet qui traite les autorisations de type BASIC

```
public class SecurityServlet extends HttpServlet {
    private Hashtable users = new Hashtable();
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        users.put("mickael:baron","allowed");
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();

        String auth = req.getHeader("Authorization");

        if (!allowUser(auth)) {
            res.setHeader("WWW-Authenticate", "BASIC realm=\"users\"");
            res.sendError(HttpServletResponse.SC_UNAUTHORIZED);
        } else out.println("Page Top-Secret");
    }

    protected boolean allowUser(String auth) throws IOException {
        if (auth == null) return false;
        if (!auth.toUpperCase().startsWith("BASIC "))
            return false;

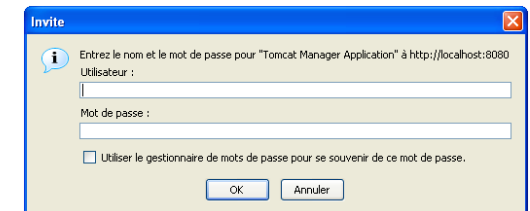
        String userpassEncoded = auth.substring(6);
        String userpassDecoded = Base64Encoder.decode(userpassEncoded);

        return ("allowed".equals(users.get(userpassDecoded)));
    }
}
```

*SecurityServlet.java* du projet  
**AuthentificationByServlet**

Récupère dans la  
requête le HEADER  
nommé **Authorization**

S'il s'agit du premier essai ou  
d'une mauvaise saisie => envoie  
d'une réponse avec l'en-tête  
**WWW-Authenticate** et l'état  
**SC\_UNAUTHORIZED**

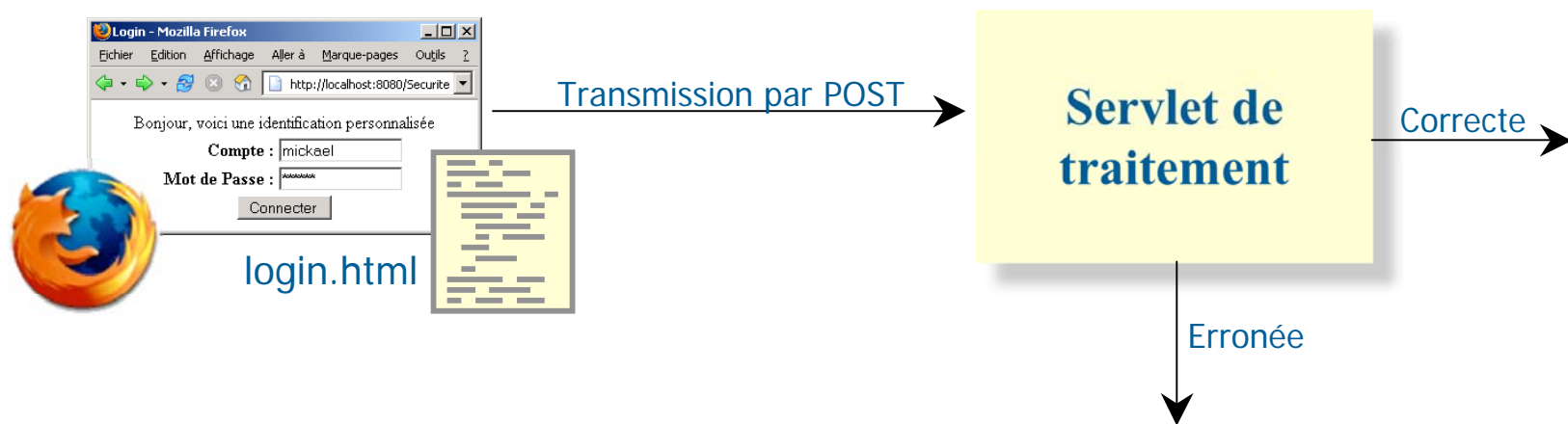


Méthode qui traite  
le résultat de l'en-  
tête **Authorization**

La vérification de l'identification est  
effectuée dans le code de la Servlet

# Authentification personnalisée à base de formulaire : Ident 2

- Différences avec les autres types d'identification
  - Cette identification part d'un **formulaire basique** avec une protection minimum des transmissions grâce à la méthode POST
  - Cette identification comparée à *l'identification personnalisée basée sur les rôles* est « démarrée » directement par la page WEB de connexion (login.html)  
Au contraire *l'identification personnalisée basée sur les rôles* est démarrée au travers de son URL virtuelle



# Authentification personnalisée à base de formulaire : Ident 2

## ➤ Exemple : Servlet qui traite les autorisations de type FORM

```
public class SecurityFormServlet extends HttpServlet {
    private Hashtable users = new Hashtable();

    public void init(ServletConfig config) throws ServletException {
        users.put("mickael:baron","allowed");
    }

    public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        String account = req.getParameter("account");
        String password = req.getParameter("password");

        if (!allowUser(account, password)) {
            out.println("<HTML><HEAD><TITLE>Access Denied</TITLE></HEAD>");
            out.println("<BODY>Votre compte et mot de passe sont incorrects.<BR>");
            out.println("<A HREF=\"\"/SecuritePersoFormServlet/login.html\">
                Retenter</A>");
            out.println("</BODY></HTML>");
        } else {
            req.getSession().setAttribute("logon.isDone", account);
            out.println("Bravo : " + account);
        }
    }
    ...
}
```

Le mot de passe est  
caché dans la requête  
de type POST

Récupère les variables  
qui ont été transmises  
par le formulaire de  
login.html

Génère une page  
d'erreur avec la  
possibilité de s'identifier  
une nouvelle fois

L'identification est  
réussie et on modifie la  
session en ajoutant le  
nom de l'utilisateur

*SecurityFormServlet.java* du projet  
**AuthentificationFormByServlet**

# Se connecter aux bases de données

---

- Utilisation de l'API JDBC (Java DataBase Connectivity)
  - JDBC est une API du niveau SQL, elle permet d'exécuter des instructions SQL et de retrouver les résultats (s'il y en a)
  - L'API est un ensemble d'interfaces et de classes conçues pour effectuer des actions sur toute base de données (mySQL, ORACLE, SYBASE, ODBC, Derby)
- Utilisation d'un gestionnaire de pilotes JDBC
  - Un pilote JDBC spécifique à une base de données implémente l'interface *java.sql.Driver*
  - Peut dialoguer avec tout pilote conforme à l'API JDBC où les pilotes sont disponibles à [java.sun.com/products/jdbc](http://java.sun.com/products/jdbc)
- JDBC en quatre étapes
  - Charger le pilote
  - Se connecter à la base
  - Créer et exécuter une requête SQL
  - Traiter le résultat si nécessaire

# Se connecter aux bases de données : charger un pilote

- Le pilote est obligatoire, il convertit les appels JDBC en appels natifs. Il est nécessaire de connaître le nom de la classe du pilote JDBC que l'on veut utiliser

- Pilote ORACLE : `oracle.jdbc.driver.OracleDriver`
- Pilote JDBC/ODBC : `sun.jdbc.odbc.JdbcOdbcDriver`
- Pilote mySQL : `com.mysql.jdbc.Driver`
- Pilote Derby : `org.apache.derby.jdbc.ClientDriver`



**La librairie (généralement .jar) désignant le pilote doit être placée dans le répertoire WEB-INF/lib**

- Le chargement du pilote se fait en utilisant la méthode `Class.forName(String Pilote) throws ClassNotFoundException`

Chargement du pilote Derby

```
Class.forName("org.apache.derby.jdbc.ClientDriver");
```

Exception levée sur la classe pilote n'est pas trouvée

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Chargement du pilote ODBC

# Se connecter aux bases de données : se connecter à la base

- Pour se connecter à une base de données il faut fournir une URL qui indique où se trouve la base
  - URL ORACLE : *jdbc:oracle:thin:host:port:ibase*
  - URL ODBC : *jdbc:odbc:IDDSN*
  - URL mySQL : *jdbc:mysql:host*
  - URL Derby : *jdbc:derby:host*
- La connexion à la base se fait en utilisant la méthode *DriverManager.getConnection("URL","user","pass")* throws *SQLException*

Exception levée si la connexion à la base est impossible

Connexion à une base mySQL nommée Espoir

```
Connection ma_connexion =  
DriverManager.getConnection("jdbc:mysql://localhost/Espoir","michael","baron");
```

```
Connection ma_connexion =  
DriverManager.getConnection("jdbc:oracle:thin:@dbhost:1528:ORCL","michael","baron");
```

Connexion à une base ORACLE nommée ORCL

# Se connecter aux bases de données : créer et exécuter une requête SQL

- La requête ne peut être créée et exécutée que si le pilote et la connexion à la base se sont valides
- Il faut avant tout créer une instruction SQL sur la base *createStatement() throws SQLException*

Exception levée si l'instruction SQL ne peut être créée

```
Statement mon_statement = ma_connexion.createStatement();
```

La référence de la connexion à une base de données

- Il faut enfin exécuter la requête en écrivant concrètement le requête SQL *executeQuery(String requete) throws SQLException*

Exception levée si le requête SQL n'est pas correcte

```
ResultSet mon_resultat = mon_statement.executeQuery("SELECT * FROM `table`");
```

Référence de l'instruction SQL

La requête SQL

# Se connecter aux bases de données : traiter le résultat

- Enfin il faut traiter un objet de type *ResultSet* il existe de nombreuses méthodes
  - *boolean next()* : avancer d'une ligne de la table résultante
  - *String getString(String columnName)* : interroger un champ *String* par son nom
  - *String getString(int columnIndex)* : interroger un champ *String* par son index
  - *int getInt(...)* : interroger un champ *Integer* par son nom ou index
  - ...

```
while (mon_resultat.next()) {  
    String colonnel = mon_resultat.getString(1);  
    out.println("Nom:" + colonnel);  
}
```

Accède à la prochaine ligne  
s'il y en a

Affiche le résultat en ligne

# Se connecter aux bases de données

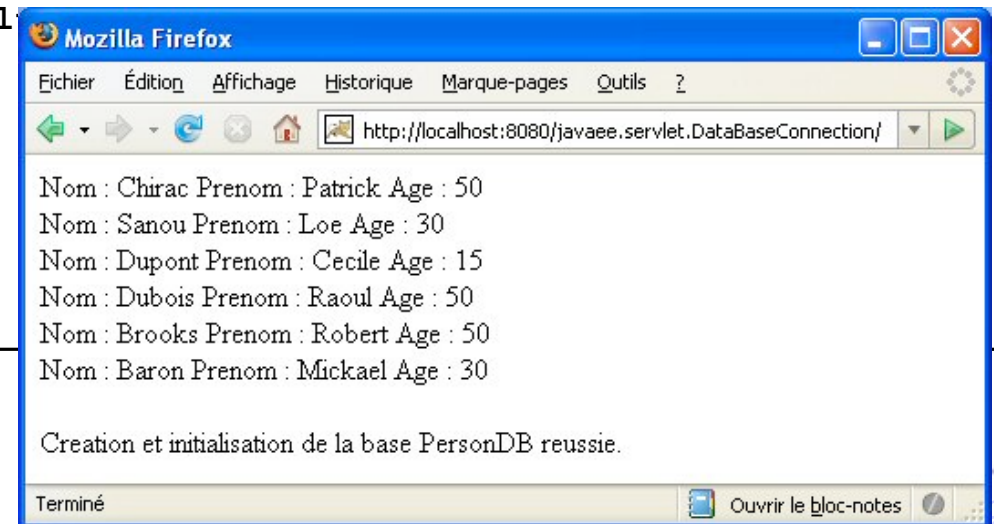
## ► Exemple : Servlet qui interroge une base de données mySQL

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

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        try {
            Class.forName("org.apache.derby.jdbc.ClientDriver");
            Connection ma_connexion =
                DriverManager.getConnection("jdbc:derby://localhost:1527/PersonDB");
            Statement mon_statement = ma_connexion.createStatement();
            ResultSet mon_resultat = mon_statement.executeQuery("SELECT NAME,
                FIRSTNAME, OLDYEAR FROM PERSON");
            while (mon_resultat.next()) {
                out.print("Nom : " + mon_resultat.getString(1));
                out.print(" Prénom : " + mon_resultat.getString(2));
                out.println(" Age : " + mon_resultat.getString(3));
                out.println("<br>");
            }
        } catch (SQLException e) {
            ...
        }
    }
}
```

*DataBaseServlet.java* du projet  
**DataBaseConnection**



**N'oubliez pas dans la requête  
SQL les cotes `` entre les  
différentes tables**

