Java côté serveur
Servlets et JSP

Patrick Itey
INRIA - Sophia Antipolis

Patrick.Itey@sophia.inria.fr
http://www.inria.fr/acacia/personnel/itey

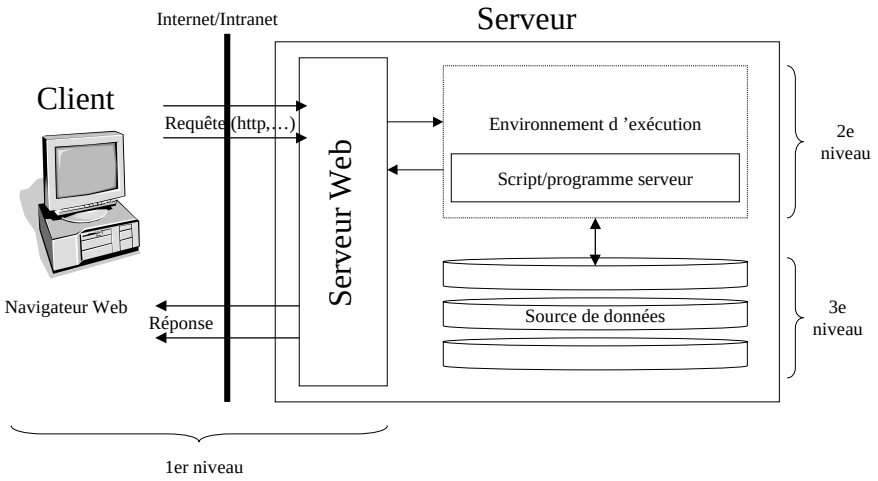
Plan du cours

- Rappel sur les applications Web
- Servlets et JSP : c'est quoi ?
- Compilation / installation / configuration d'un serveur Web compatible
- Première servlet ou comment démarrer ?
- Gérer les formulaires HTML
- Servlets et bases de données
- Gestion des cookies

Plan (suite)

- La gestion de session
- Les Java Server Pages (JSP)
- Des liens
- Des Tds

Applications Web



Architecture

- 3 niveaux :
  - **niveau 1**: **présentation**
    - navigateur + serveur Web
  - **niveau 2**: **applicatif**
    - script ou programme
  - **niveau 3**: **données**
    - données nécessaires au niveau 2

Déroulement

- Une application Web type :
  - **1**: recueille les données utilisateur (**niveau 1**)
  - **2**: envoie une requête au serveur Web
  - **3**: exécute le programme serveur requis (**niveau 2&3**)
  - **4**: assemble/renvoie les données vers le navigateur (**niveau 1**)

1: Collecte des données utilisateur

- Quelques solutions pour le client :
  - très utilisée : formulaire HTML
    - saisie de champs puis « submit »
    - validation par scripts (javaScript)
  - nouvelle : applets Java :
    - connexion socket / RMI avec le serveur Web
    - mise en forme et validation des données
  - ...

2: Requête HTTP vers le serveur Web

- contient :
  - l 'URL de la ressource à accéder (page,script,prog)
  - les données de formatage (le cas échéant)
  - des infos d 'en-tête complémentaires
- requête GET :
  - pour extraire des informations sur le serveur
  - intègre les données de formatage à l 'URL

```
http://www.inria.fr/servlet/hello?key1= value1&…
```

- requête POST :
  - pour modifier les données sur le serveur
  - données de la page assemblées/envoyées vers le serveur

3: Exécution d ’un script/prog. serveur

- Avec la requête http, le serveur Web :
  - identifie le type d ’environnement d ’exploitation à charger (*mapping*)
    - en fonction de l ’extension du fichier (**`.jsp`**, **`.cgi`**, ...)
    - ou du répertoire où il se trouve (**`cgi-bin/`**, **`servlet/`**)
  - charge l ’environnement d ’exécution (*run-time*)
    - interpréteur Perl pour les programmes cgi en perl
    - JVM pour les servlets Java, ...

4: Retour des résultats au navigateur

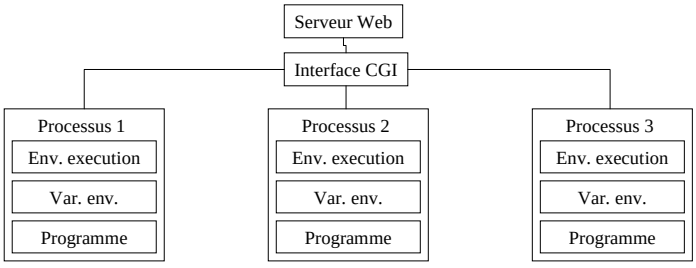
- Le script/prog côté serveur :
  - précise le type de contenu (HTML, XML, images,)
  - intègre la réponse dans un flot de sortie
- Le navigateur :
  - définit le type MIME dans l ’en-tête (text/html,…)
  - et affiche les données en fonction
    - duplication de l ’environnement (variables, exécution), mémoire allouée, copie du programme, …
    - retourne (en général) du HTML

Techniques côté serveur

- CGI (Common Gateway Interface)
- ISAPI, NSAPI (Netscape, Microsoft)
- ASP (Microsoft)
- Servlets Java et JSP (Sun)

CGI (rappel)

- Principe :
  - un processus par requête est lancé sur le serveur



CGI (suite)

- Avantages :
  - gratuit, pris en charge par tous les serveurs Web actuels
  - peut être écrit dans n'importe quel langage (surtout perl)
- Inconvénients :
  - manque d évolutivité (plusieurs processus créés)
    - serveur très sollicité si plusieurs requêtes au même moment
    - amélioré par :
      - » FastCGI : instance partagée des programmes CGI
      - » mod_perl (Apache) : script CGI interprété et exécuté dans le serveur Web
  - assez lent
  - parfois difficile à développer

Servlets et JSP

Java Servlets

- Réponse de Java aux CGI
- Programmes s'exécutant sur le serveur Web et retournant des pages Web dynamiques (à la volée)
- Peuvent être chargés localement ou dynamiquement à travers le réseau

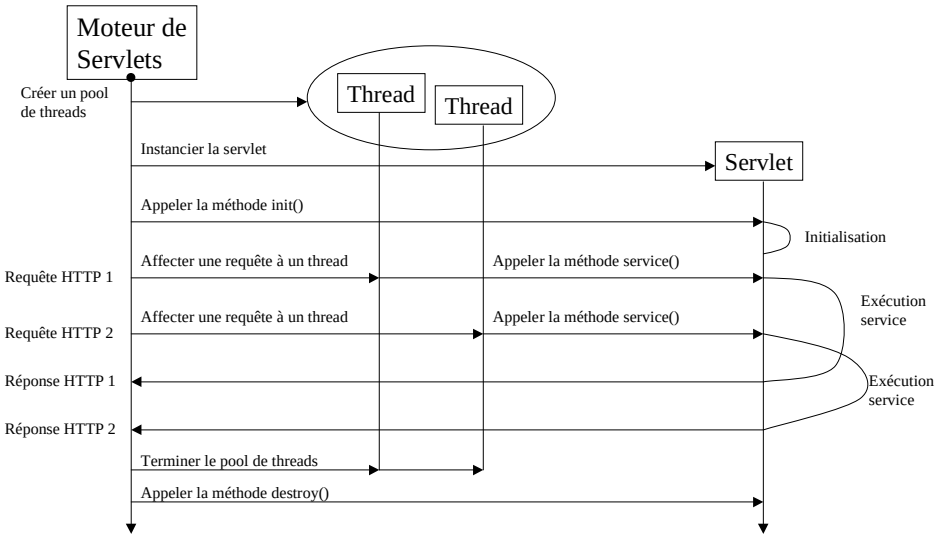
Servlets vs. CGI

- Plus efficaces
- Plus pratiques
- Plus puissantes
- Portables
- Gratuites

Servlets plus efficaces

- Résidentes, pas de fork, pas de temps de lancement
- Multithreads
- Gestion de cache
- Connexions persistantes (BD)
- etc...

Gestion des servlets



Servlets plus pratiques

- Hé, c'est du Java !
- Super API pour gérer les formulaires HTML,
- les cookies, le suivi de session, …
- le protocole HTTP devient facile à manipuler
  - manipuler les headers Http
- Plus facile à utiliser que cgi/perl

Servlet plus puissantes

- On peut faire des choses impossibles à réaliser avec des scripts CGI
  - Parler avec le serveur WWW,
  - Echanger des données via URIs,
  - Partager des données entre servlets,
  - Chaîner des servlets (pool de connections BD),
  - Gestion de sessions (e-commerce),
  - etc...

Servlets portables

- C'est du Java !
- Supportées par tous les serveurs WWW
  - Apache, Microsoft IIS, WebStar, …
- directement ou via des plugins/patches

Servlets gratuites

- Kit de développement des servlets gratuit
- Nombreuses versions commerciales... (Microsoft, Netscape I-server, Webstar...)
- Mais Apache/Tomcat reste la solution la plus efficace... 100% gratuite !

Servlets vs. applets

- Les *servlets* sont le pendant des *applets* côté serveur
  - mais sans interface graphique utilisateur …
    - elle est limitée à la puissance du langage HTML …
  - par contre, elles ne sont pas astreintes aux mêmes règles de sécurité que les *applets*
    - peuvent établir une connexion avec d'autres clients (RMI, …)
    - peuvent faire des appels système (utilisation pont JDBC-ODBC)
    - ou manipuler des ressources locales (sur le serveur), ...

Avantages et inconvénients

- Avantages :
  - plus facile à développer
  - meilleures performances
  - client « léger "
- Inconvénient :
  - interface graphique utilisateur limitée à HTML

Les JSP (Java Server Pages)

- Réponse aux ASP/PHP/embedded perl, etc...
- Technologie qui permet de mixer Java et HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Welcome to Our Store</TITLE></HEAD>
<BODY>
<H1>Welcome to Our Store</H1>
<SMALL>Welcome,
<!-- User name is "New User" for first-time visitors -->
<% out.println(Utils.getUserNameFromCookie(request)); %>
To access your account settings, click
<A HREF="Account-Settings.html">here.</A></SMALL>
<P>
Regular HTML for all the rest of the on-line store's Web page.
</BODY></HTML>
```


Avantages des JSPs

- Vs ASP : c'est du Java, ça suffit non ?
- Vs Servlets : plus pratique. Séparation du look et du traitement. 100% équivalent.
- Vs JavaScript : Javascript tourne sur le client
- Vs HTML/Dynamic HTML : idem.

- JSP est très facile à mettre en oeuvre !

Servlets/JSP API, configuration d'un serveur WWW compatible

Kits de développement Servlets et JSP

- Récupèrer JSWDK (Servlet 2.1/2.2 & JSP 1.0/1.1)
  - http://java.sun.com/products/servlets
  - Dire à javac où trouver les packages nécessaires :

    ```
    CLASSPATH=.:servlet_dir/servlet.jar:servlet_dir/jsp.jar
    ```
  - Conseil : mettez vos propres servlets dans un package ! (classpath...)
- Installer un serveur WWW qui supporte les servlets

Quel serveur WWW ?

- 2 possibilités
  - installer un serveur qui a le support intégré pour les servlets
  - ajouter à votre serveur WWW actuel un package qui supporte les servlets
- Apache Tomcat (servlets 2.2, JSP 1.1)
- Java Server Web Development Kit (moteur)
- Allaire Jrun
- New Atlanta Server Exec
- Sun Java Web Server...

Premières Servlets

Modèle de programmation

- Une servlet doit implémenter l'interface **`javax.servlet.Servlet`**
  - soit directement,
  - soit en dérivant d'une classe implémentant déjà cette interface comme (**GenericServlet ou HttpServlet**)
- cette interface possède les méthodes pour :
  - initialiser la servlet : **init()**
  - recevoir et répondre aux requêtes des clients : **service()**
  - détruire la servlet et ses ressources : **destroy()**

Structure de base d'une servlet

```
import javax.servlet.*;

public class first implements Servlet {

   public void init(ServletConf config)
       throws ServletException {…}

   public void service(   ServletRequest req,
                          ServletResponse rep)
       throws ServletException, IOException {…}

   public void destroy() {…}
}
```

Le cycle de vie

1. la *servlet* est crée puis initialisée (**`init()`**)
   - cette méthode n'est appelée par le serveur qu'une seule fois lors du chargement en mémoire par le moteur de servlet
2. le service du client est implémenté (**`service()`**)
   - cette méthode est appelée automatiquement par le serveur à chaque requête de client
3. la *servlet* est détruite (**`destroy()`**)
   - cette méthode n'est appelée par le serveur qu'une seule fois à la fin
   - permet de libérer des ressources (allouées par `init()`)

Une servlet Web : **`HttpServlet`**

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

Squelette d 'une servlet Http (GET)

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

public class SimpleServlet extends HttpServlet {

   public void init(HttpServletConfig c)
        throws ServletException  {…}

   public void doGet(HttpServletRequest req,
                     HttpServletResponse res)
        throws ServletException, IOException {…}


   public void destroy() {…}

   public String getServletInfo() {…}
}
```


Les méthodes doGet() et doPost()

- Utiliser les objets **`HttpServletRequest`** et **`HttpServletResponse`** passés en paramètres de ces méthodes pour implémenter le service :
  - **`HttpServletRequest`** contient les renseignements sur le formulaire HTML initial (utile pour **`doPost()`**) :
    - la méthode **getParameter()** récupère les paramètres d'entrée
  - **`HttpServletResponse`** contient le flux de sortie pour la génération de la page HTML résultat (**`getWriter()`**)

Structure de base d'une servlet

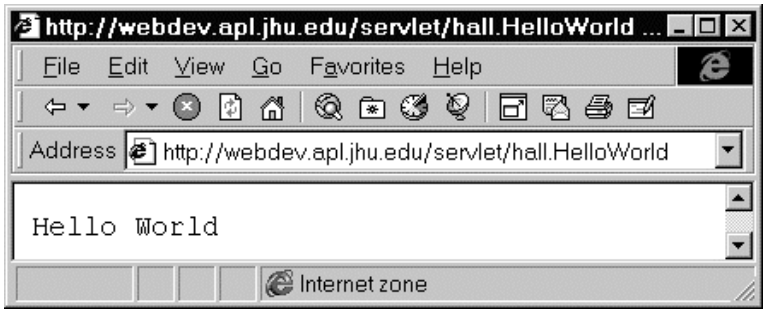
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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

    // Use "request" to read incoming HTTP headers (e.g. cookies)
    // and HTML form data (e.g. data the user entered and submitted)
    ...
    // Use "response" to specify the HTTP response line and headers
    // (e.g. specifying the content type, setting cookies).

     PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
```


Un exemple



Une application Web

- Une application web = un espace virtuel
  - Contient html, images, servlets, jsp...
- Avec Tomcat

  Editer `<tomcat_dir>/server.xml` pour définir une application Web

```
<Context path="pit" docBase="pit"
defaultSessionTimeOut="30" isWARExpanded="true"
isWARValidated="false" isInvokerEnabled="true"
isWorkDirPersistent="false"
/>
```

  - Dans cet exemple, l'application web se nomme pit
  - Mettre les classes dans <tomcat_dir>/**pit**/WEB-INF/classes
  - Editer <tomcat_dir>/**pit**/WEB-INF/web.xml

Une application Web (suite)

- Avec le fichier `<tomcat_dir>/pit/WEB-INF/web.xml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">

<web-app>
    <servlet>
      <servlet-name>
          HelloWorld
      </servlet-name>
      <servlet-class>
          hall.HelloWorld
      </servlet-class>
    </servlet>
</web-app>
```

Une application Web (suite)

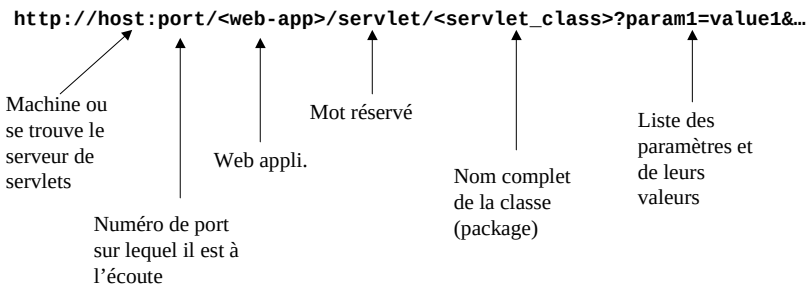
- Une fois une application web créée, on peut mettre autant de servlets que l'on veut...
- Relancer Tomcat à chaque modif des fichiers XML `server.xml` et/ou `web.xml`
- Pour invoquer la servlet, utiliser l'alias :

  `http://host/pit/servlet/HelloWorld`

  ou le nom complet...
  `http://host/pit/servlet/hall.HelloWorld`

Charger et invoquer une servlet

- D'une manière générale, une URL du type :

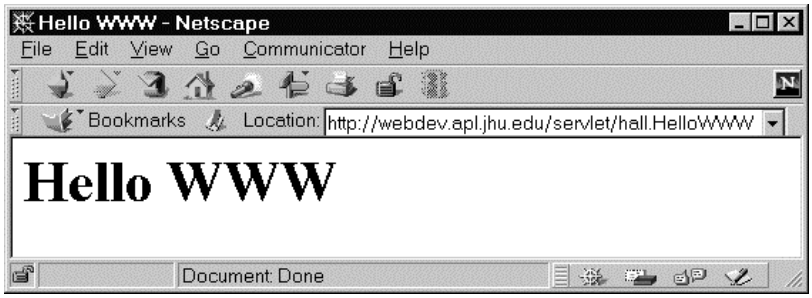


Récupération des paramètres passés à la servlet

- Utilisation des méthodes de **`ServletRequest`** :

```
public void doGet(   HttpServletRequest req,
                     HttpServletResponse rep)
   throws ServletException, IOException
{
       String[] values = req.getParameterValues();
       Enumeration list = req.getParameterNames();
       String value1 = req.getParameter("param1");
       if(value1 == null) ...
}
```


Un autre exemple



Un autre exemple

```
package hall;

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class HelloWWW extends HttpServlet {

  public void doGet(HttpServletRequest request,
                              HttpServletResponse response)
                       throws ServletException, IOException {

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
                "Transitional//EN\">\n" +
                "<HTML>\n" +
                "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                "<H1>Hello WWW</H1>\n" +"</BODY></HTML>");

  }

}
```


Quelques trucs

- Bon, génèrer du HTML... la vraie solution c'est JSP !
- Néanmoins : <DOCTYPE...> et <HEAD...> toujours pareils !
- Faire une classe utilitaire !

```
public class ServletUtilities {
  public static final String DOCTYPE =
    "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
   Transitional//EN\">";

  public static String headWithTitle(String title) {
    return(DOCTYPE + "\n" +
           "<HTML>\n" +
           "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n");
  }
}
```

Nouvelle version

```
package hall;

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

public class HelloWWW2 extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(ServletUtilities.headWithTitle("Hello WWW") +
                "<BODY>\n" +
                "<H1>Hello WWW</H1>\n" +
                "</BODY></HTML>");
  }
```


Gestion des formulaires HTML

Introduction

- Gestion des formulaires HTML

```
http://host/path?user=Marty+Hall&origin=bwi&dest=lax
```

- La partie compliquée = paramètres du formulaire
- Visibles ou non dans l'URL (GET/POST)
- Ces paramètres doivent être décodés !
  - Partie la plus difficile. Encodage = norme CGI
- Avec les servlets : un vrai plaisir !

Récupèrer les paramètres

- Méthode getParameter() de HttpServletRequest
  - Fonctionne avec GET ou POST

```
public class ShowParameters extends HttpServlet {
   public void doGet(HttpServletRequest request,...) {
     out.println(request.getParameter("param1") );
     Enumeration paramNames=request.getParameterNames();
     String[] paramValues =
     request.getParameterValues(paramNames);
   }
}
```

Gestion des cookies

C’est quoi ?

- Morceaux d’informations envoyés par le serveur
  … et renvoyés par le client quand il revient visiter le même URL
- Durée de vie réglable
- Permet la persistance

A quoi ça sert ?

- Identification des utilisateurs (e-commerce)
- Eviter la saisie d'informations à répétition
  - login, password, adresse, téléphone…
- Gérer des « préférences utilisateur »
  - sites portails…
- ...

Cookie et sécurité ?

- Jamais interprété ou exécuté : pas de virus
- Un cookie est limité à 4KB et les navigateurs se limitent à 300 cookies (20 par site) : pas de surcharge de disque
- Bien pour rendre privées des données non sensibles
  - nom, adresse, … mais pas No CB !
- … mais ne constitue pas un traitement sérieux de la sécurité

Gestion des cookies ?

- Utiliser les fonctions de l ’API des servlets…
  - créer un cookie : classe **`Cookie`**,
  - écrire/lire un cookie : **`addCookie(cookie), getCookies()`**,
  - positionner des attributs d’un cookie : **`cookie.setXxx`**(…)
- Exemple d’envoi d’un cookie :

```
...
String nom = request.getParameter("nom");
Cookie unCookie = new Cookie("nom", nom);
...ici positionner des attributs si on le désire
response.addCookie(unCookie);
...
```


Création d'un cookie

- **`Cookie unCookie = new Cookie(name, value);`**
  - 2 arguments de type **`java.lang.String`** :
    - **`name`** et **`value`**
  - caractères non autorisés :
    - espace blanc
    - [ ] ( ) = , " / ? @ : ;

Temps d'expiration

- Par défaut, durée de vie d'un cookie = la connexion.
- Si on veut que le cookie soit sauvé sur disque, modifier sa durée de vie :

```
public static final int SECONDS_PER_YEAR =
  60*60*24*365;
cookie.setMaxAge(SECONDS_PER_YEAR);
```


Suivi de session en java

Problématique

- Protocole HTTP = protocole Internet déconnecté
  - différent de Telnet, Ftp, …
  - traite les requêtes et les réponses comme transactions simples et isolées (requêtes non apparentées)
- Certaines applications Web (e-commerce : caddie) ont besoin de maintenir une "mémoire" entre deux requêtes
  - ie. maintenir une connexion de l'utilisateur sur le serveur
  - pour se faire : concept de "suivi de sessions"

Suivi de session : qu'est-ce que c'est ?

- Mémoire de ce que fait l'utilisateur d'une page à l'autre
  - consiste au transfert de données générées par une requête vers les requêtes suivantes
- 4 méthodes avec les servlets Java
  - 1) utilisation des cookies (déjà vu)
  - 2) réécriture d'URL : passage de paramètres
  - 3) utilisation des champs de formulaire "hidden"
  - 4) utilisation du JSDK (HttpSession API)

Réécriture d'URL

- Principe :
  - ajouter dans la chaîne de requête de la servlet des informations supplémentaires identifiant la session

```
<a href="http://leo.inria.fr/servlet/foo?uid=itey">Acheter</a>
```

  - l'ID utilisateur est transmis en même temps que la requête; il est accédé par chaque servlet mentionnée qui récupère les informations persistantes (BD, fichiers) à partir de cet ID
- Limitations :
  - données volumineuses, caractères autorisés, longueur URL, données visibles (sécurité)

Champs de formulaires cachés

- Principe :
  - on cache les données de session dans des champs "hidden" :

```
<INPUT TYPE="HIDDEN" NAME="uid" VALUE=itey">
```

- Limitations :
  - idem la "réécriture d'URL" sauf pour la sécurité (utilisation de POST)

Servlet session ?

- Très simple avec l'API des servlets (JSDK)
  - objet **`HttpSession`**
- Principe :
  - Un objet "session" peut être associé *avec chaque requête*
  - Il va servir de "container" pour des informations persistantes
  - Durée de vie limitée et réglable

Modèle basique

```
HttpSession session = request.getSession(true);
Caddy caddy = (Caddy) session.getValue("caddy");

if(caddy != null) {
   // le caddy n'est pas vide !
  afficheLeContenuDuCaddy(caddy);
} else {
   caddy = new Caddy();
   ...

   caddy.ajouterUnAchat(request.getParameter("NoArticle"));
   session.putValue("caddy", caddy);
}....
```

Méthodes de la classe HttpSession

- **`getID()`**
- **`isNew()`**
- **`getCreationTime() / getLastAccessedTime()`**
- **`getMaxInactiveInterval()`**
- **…**
- **`getValue(), removeValue(), putValue()`**
- **…**
- **`invalidate()`**

Java Server pages (JSP)

C'est quoi ?

- Du java dans une page WWW !

```
http://fkeiko.inria.fr:8080/jsp/Test.jsp?titre=Les+JSP
...
<I> <%= request.getParameter("titre"); %> </I>
...
```

  - Entre les balises JSP <% … %>
- On peut mettre des pages `.jsp` partout où on met des pages HTML
- Elles sont converties "au vol" en servlet par le moteur de JSP

Un exemple simple

```
<html><head><title>Un exemple de page JSP</title></head><body>
<!-- définit les informations globales a la page -->
<%@page language="java" %>

<!-- Déclare la variable c -->
<%! char c = 0; %>

<!-- Scriplet (code java) %>
<%
   for(int i = 0; i < 26; i++){
        for(int j = 0; j < 26; j++){
          c = (char)(0x41 + (26 - i + j)%26);
%>
<%= c %>
<%      } %>
<br>
<% } %>
</body></html>
```

Balises JSP dans le HTML

- Trois types :
  - 1 - *Scripting elements* : du code java
  - 2 - *directives* : pour le contrôle de la structure
  - 3 - *actions* : importation de composants existants

Scripting elements (1)

- <%= expression %>

```
Il est <%= new java.util.Date() %> <P>
et votre hostname est <%= request.getRemoteHost() %>
```

  - permet d'intégrer des valeurs dans le code HTML
  - ces valeurs sont évaluées, converties en chaînes de caractères et affichées
  - les objets implicites (request, response, session, out, ...) disponibles

Scripting elements (2)

- <% code Java %> (scriplets)

```
<%
   String nom = request.getParameter("nom");
   ...
   out.println("Nom de l'utilisateur " + nom);
%>
```

  - c'est un bloc de code Java
  - placé dans **`_jspService()`** de la servlet générée
  - ayant accès :
    - aux variables et *beans* déclarés (**`<%! … %>`**)
    - aux objets implicites (voir plus loin)

Scripting elements (3)

- <%! déclarations %>

```
<%!
  private int accessCount = 0;
  private int incrementCount() {accessCount++;}
%>
...
<H2>Nombre et liste des articles</H2>
Nombre d'articles : <%= incrementCount() %>
```

  - définitions des méthodes et variables de classe

  à utiliser dans toute la page
  - définit les méthodes **`jspInit()`** et **`jspDestroy()`**

Directives

- <%@ directive attribut1="valeur"
  attribut2="valeur"... %>
- 2 directives possibles (jsp1.0) :
  - page : informations relatives à la page
  - include : fichiers à inclure littéralement

La directive : page

- Valeurs possibles :
  - <%@ page language="java"
  - <%@ page import="java.util.*, java.net.*" %>
  - <%@ page contentType="text/plain" %>
  - <%@ page session="true|false " %>
  - <%@ page errorPage="pathToErrorPage" %>
  - <%@ page isErrorPage="true|false" %>
  - <%@ page …

Actions (1)

- Syntaxe XML
- Permettent de faire des actions au moment où la page est demandée par un client :
  - inclure dynamiquement un fichier
  - utiliser des *beans*
  - rediriger vers une autre page
  - etc...

Actions (2)

- **`<jsp:include page="relative URL" flush="true" />`**
  - inclusion au moment où la page est servie, pas au moment où elle est traduite en servlet.
- **`<jsp:usebean id="name" class="package.class" />`**
  - permet d'instancier un *bean* depuis une page JSP.
  - nécessite de connaître le mécanisme des *beans*...
  - associé à **`<jsp:getProperty.../>`** et **`<jsp:setProperty.../>`**
- **`<jsp:forward page="/unAutreURI" />`**
  - redirige vers un autre URI/URL

Usebean et getProperty

- Mécanisme très puissant !

```
<jsp:usebean
   id="name"  (référence l'instance du composant)
   class="paquetage.class" (nom qualifié de la classe)
   scope="page|request|session|application" (portée)
 />
```

- Pour lire une propriété du *bean* :

```
<jsp:getProperty name="name" property="property" />
```


Usebean et setProperty

- Pour modifier une propriété du *bean* :

```
<jsp:setProperty name="name"
                 property="property"
                 value="value" />

<jsp:setProperty name="name"
                 property="*" />
```

  - Initialise tous les attributs de l 'objet **name** avec les paramètres HTTP du même nom
  - En 2 lignes !

Exemple d'utilisation d'un bean

- La page JSP :

```
<html> ...
<jsp:usebean id="test" class="inria.SimpleBean" />
<jsp:setProperty name="test" property="message"
                 value="Hello !!" />
<h1>Le message est : <i>
<jsp:getProperty name="test" property="message" />
</i></h1>…
</html>
```


Exemple d'utilisation d'un bean

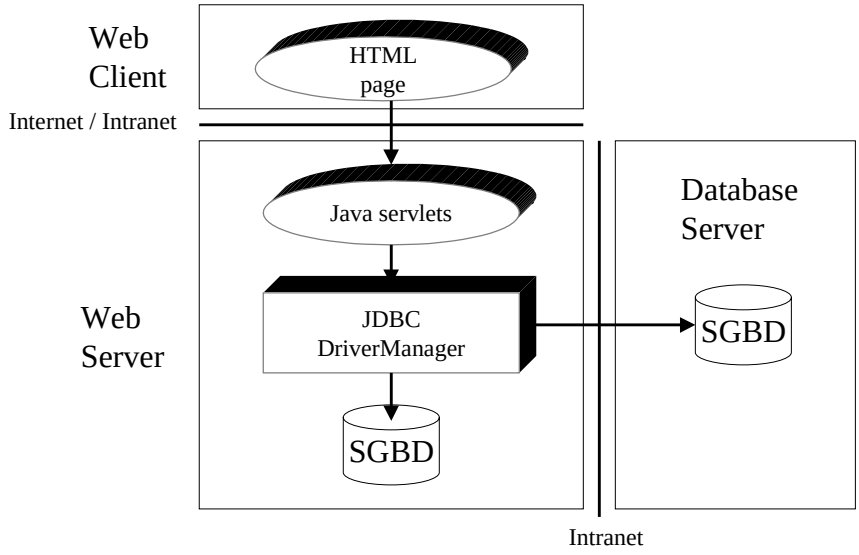
- Le code source Java du *bean* :

SimpleBean.java

```
package inria;
public class SimpleBean {
   private String message = "no message";
   public String getMessage() {
       return message;
   }
   public void setMessage(String message) {
       this.message = message;
   }
}
```

Servlets et bases de données

Servlets et bases de données



Un exemple complet : "publier sur le Web un annuaire d'une société"

- L'objectif :
  - publier sur le Web en utilisant une *servlet* les coordonnées d'un employé en le recherchant par son nom
- L'approche en 3 parties :
  - la page HTML pour le formulaire d 'interrogation
  - la servlet effectuant la requête
  - la page HTML résultat générée par la servlet

Le formulaire d' interrogation

Fichier : annuaire.html

```
<HTML>
<HEAD><TITLE>Annuaire YETI</TITLE></HEAD>
<BODY>
<CENTER><H1> Annuaire de la société YETI </H1></CENTER>
<HR WIDTH="75%">
<CENTER><H2>Recherche de coordonnées</H2></CENTER>
<P>Tapez les premières lettres de la personne désirée
<P><FORM
   METHOD=POST
   ACTION=http://fkeiko.inria.fr:8090/servlet/Annuaire>
<INPUT TYPE=TEXT NAME="nom" SIZE=10 MAXLENGTH=20 VALUE="">
<P><INPUT TYPE=SUBMIT NAME="go" VALUE="Rechercher">
<INPUT TYPE=RESET NAME="reset" VALUE="Reset">
</BODY>
</HTML>
```

La servlet Annuaire (1)

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

public class Annuaire extends HttpServlet
{
   public void doPost(HttpServletRequest req, HttpServletResponse
   res)
        throws ServletException, IOException {
        ...
   }

   public String getServletInfo() {
        return "La servlet Annuaire";
   }
}
```


La servlet Annuaire (2)

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><BODY>");
    out.Println("<CENTER><H1>Voici les coordonnées :</H1></CENTER>");
    try{
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
          String url = "jdbc:odbc:ANNUAIRE";
          Connection c = DriverManager.getConnection(url, "itey", "admin");
          Statement s = c.createStatement();
          String query =       "SELECT name,phone FROM Annuaire" +
                               "WHERE name LIKE '" + req.getParameter("nom") +  "%'";
          ResultSet rs = s.executeQuery(query);
          rs.next();
          out.println("<P>NOM: " + rs.getString("name"));
          out.println("<P>TELEPHONE: " + rs.getInt("phone"));
          rs.close(); s.close(); c.close();
    } catch(Exception e) {…}
    out.println("</BODY>/HTML>");
    out.close();
}
```