UE INF347

Des APIs Java Web (Servlets et JSP) aux frameworks Java

Sylvie Vignes

Télécom ParisTech
Département Informatique et Réseaux

Les applications du Web selon 2 points de vue complémentaires

- Les aspects fonctionnels liés à l'usage : site, réseaux sociaux, front office d'un SI
- Les aspects d'architecture logicielle
  - Patterns architecturaux : différents raffinements du modèle MVC dont les éléments sont:
    - Le navigateur client
    - Le serveur Web
    - Le serveur d'application
    - Le serveur de données

Ce cours se focalise sur ce point et les implémentations en Java

Du côté du serveur : patterns architecturaux

- Des APIS Java Web : bibliothèques légères, conçues pour un développement de base
  - Servlets (et JSP), JBDC … du JDK
- Aux frameworks : bibliothèques de composants « compactes » et riches en fonctionnalités « génériques »
  - Struts, JSF, **Spring**, Hibernate
  - Chaque framework a un champ d'actions **privilégié**
    - Struts -> Contrôleur
    - JSF -> Vue
    - Spring -> Modèle
    - Hibernate -> persistance, « pont objet-relationnel » vers BD

Du côté du client : patterns architecturaux

- Client Web
  - universel :
    - Navigateur
  - léger
    - Navigateur + JavaScript
  - *alourdi*
    - applets Java, [ActiveX], plug-ins
- Client lourd
    - Application à part entière client/serveur

Plan

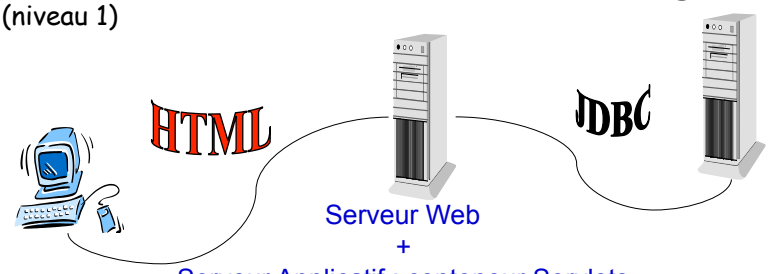
- Rappel sur les applications Web
- Servlets et JSP
  - Compilation/installation/configuration d'un serveur Web compatible
  - Première Servlet ou comment démarrer
  - Gérer les formulaires HTML
  - Gestion de cookies
  - La gestion de session
  - Les Java Server Pages (JSP)
- Principe du framework Struts
- Principe du framework Spring
- L'apport des Java Server Faces (JSF)
- Servlets et accès Bases de Données

Applications Web "3 tiers"

- Niveau 1: présentation
  - Navigateur + serveur Web
- Niveau 2 : applicatif
  - Script ou programme
- Niveau 3 : données
  - Persistance des données

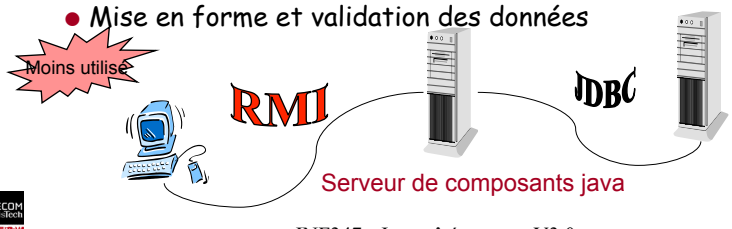
Pattern de base d'une application Web

- Recueille les données utilisateurs (niveau 1)
- Envoie une requête au serveur Web (niveau 1)
- Exécute le programme serveur requis
  (niveau 2 et 3)
- Assemble/renvoie les données vers le navigateur
  (niveau 1)



1- Collecte des données utilisateur

- Quelques solutions pour le client
  - Dynamique, de base: formulaire HTML
    - Saisie de champs puis " submit "
    - Validation par scripts (javaScript)
    - balise ancre avec un exécutable ou interprétable JVM (.class)
  - Dynamique, liée à Java :voir JSP
  - Propriétaire Java : applets
    - 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,programme)
  - Les données de formatage (le cas échéant)
  - Des infos d'en-tête complémentaires
- Requête (*verbe* du protocole) GET
  - Pour extraire des informations sur le serveur
  - Intègre les données de formatage à l'URL
- Requête (*verbe* du protocole) POST
  - pour modifier des informations sur le serveur
  - Données de la page assemblées/envoyées vers le serveur

3- Exécution d'un script ou d'un programme compilé sur le serveur

- Avec la requête http, le serveur Web
  - Identifie le type d'environnement d'éxécution à charger (mapping)
    - En fonction de l'extension du fichier (xxx.cgi,xxx.php,xxx.jsp …)
    - Ou du répertoire où il se trouve (cgi/ php/ servlet/)
  - Charge l'environnement d'éxécution (run-time)
    - JVM pour les Servlets Java
    - Interpréteur Perl
    - …

4- retour des résultats au navigateur

- Le script ou programme côté serveur
- Précise le type de contenu (HTML, XML, images)
- Intègre la réponse dans un flot de sortie

- Le navigateur
  - Reconnaît le type MIME de l'entête (text/html, …)
    - text: données textuelles lisibles.
      - text/rfc822 [RFC822] ; text/plain [RFC2646] ; text/html [RFC2854] .
    - image: données binaires représentant des images numériques
      - image/jpeg ; image/gif ; image/png.
    - audio: données numériques sonores
      - audio/basic ; audio/wav
    - video : données vidéos :
      - video/mpeg
    - application : données binaires autres.
      - application/octet-stream ; application/pdf
  - Et affiche les données en fonction de ce type
    - Duplication de l'environnement (variables, exécution)
    - Retourne (en général) du HTML

Techniques côté serveur

- Programmes s'exécutant sur le serveur retournant des pages construites "à la volée"

- PHP …

- ASP (Microsoft- dotnet) : Active Server Page
  - Des pages HTML avec du code JScript ou VBScript pour accéder aux composants DCOM côté serveur.

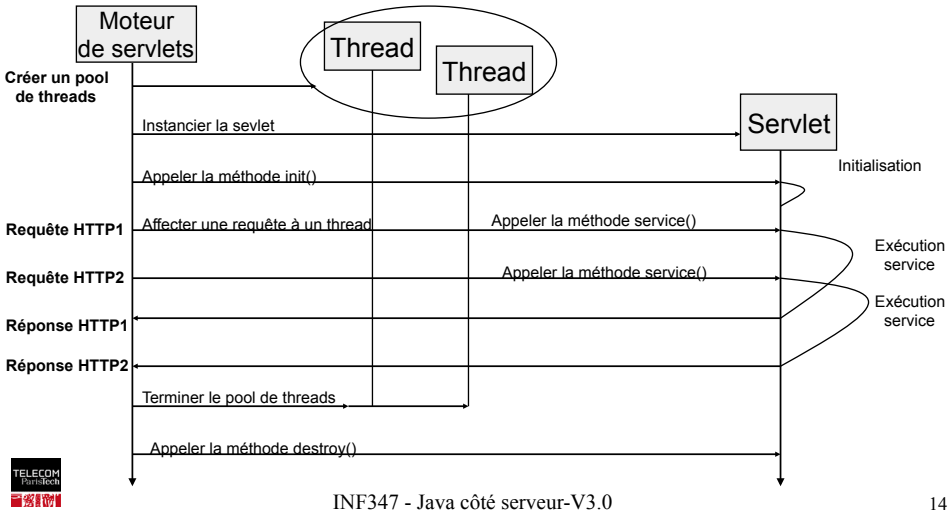
- Servlets Java et JSP

- Et les frameworks …

Efficacité des Servlets

- Résidentes:
  - pas de fork, pas de temps de lancement
- Multithreads
- Gestion du cache
- Connexions persistantes avec les BD
  - par exemple mis en œuvre dans Hibernate

Gestion des servlets



Servlets plus faciles

- C' est du Java ! (standard SUN -> Oracle)
  de la programmation objet
- Des API adaptées pour gérer
  - Les formulaires HTML
  - Les cookies
  - Le suivi de session …
- Le protocole HTTP est facilement manipulable, en particulier les headers

Servlets sont des classes adaptées pour

- "s'interfacer" avec le serveur www
- échanger des données via des URLs
- Partager des données entre servlets
- [Chaîner des servlets (pool de connections BD)]
- Gestion de sessions (e-commerce)

- Mais surtout, ce sont les classes de base dont "héritent" les frameworks

Servlets portables et gratuites

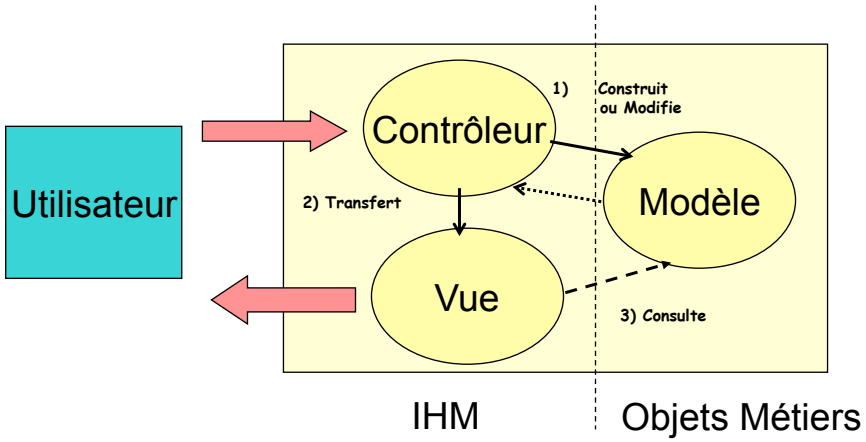
- Java …
- Supportées par tous les serveurs WWW
  - Directement ou via des plugins/add-ons
- Kit de développement des Servlets gratuit JSDK
- Diverses versions commerciales
- Apache/Tomcat ou Tomcat : la solution la plus efficace… 100% gratuite

Servlets vs applets (pour info !)

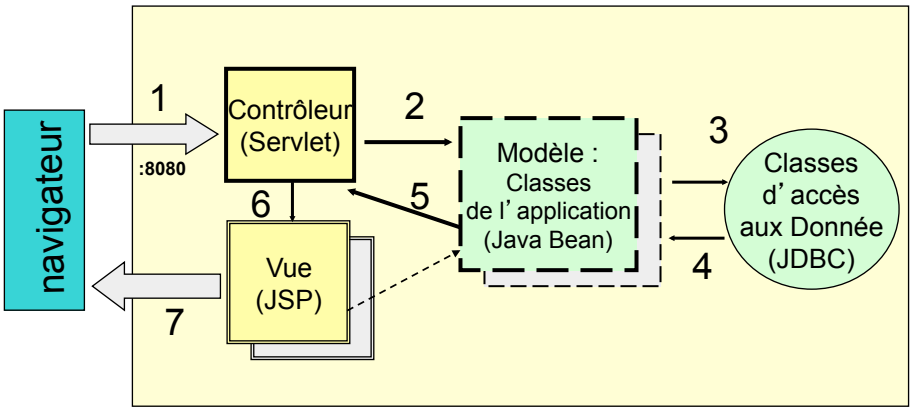
- Côté serveur vs chargées sur le client
- Pas d' interface graphique
- Ne sont pas astreintes aux mêmes règles de sécurité que les applets
  - Peuvent établir une connexion avec d' autres clients RMI
  - Peuvent faire des appels système (utilisation pont JDBC-ODBC)
  - Ou manipuler des ressources locales (sur le serveur)

Le modèle (design pattern) MVC historique

- MVC = Modèle Vue Contrôleur



MVC2 organisation adaptée aux applications Web et à la plateforme JavaEE



Les JSP (Java Server Pages)

une technologie pour mettre du code Java dans un fichier HTML

```
<!doctype html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head><titre>Magasin virtuel</titre></head>
<body>
<h1>Bienvenue dans notre magasin virtuel </h1>
<small>Bonjour,
<!– retrouver le nom depuis la dernière visite -->

<% out.println(Utils.getUserNameFromCookie(request)); %>

Pour accéder à votre compte, cliquez
<a href="Account-Settings.html">ici.</a></small>
<p>
        etc.
</body>
</html>
```

Kits de développement Servlets et JSP

- JSDK (servlet 2.3 et JSP 1.2) tomcat 4.x
  (servlet 2.4 et JSP 2.0) tomcat 5.5, 6, 7
  - http://java.sun.com/products/servlet/index.html
- Indiquer à javac où trouver les packages nécessaires
  - CLASSPATH=./servlet_dir/servlet.jar:servlet-dir/jsp.jar
- Conseil :
  - mettre vos propres servlets dans un package (attention au classpath)
- Installer un serveur
  - qui intègre le support de servlets
  - sinon ajouter à votre serveur Web, un package moteur de servlets
  - Vous pouvez déployer Tomcat « localement »
  - Ou encore intégré à l'IDE eclipse
  - Ou l'IDE Netbeans

Premières servlets:modèle de programmation

- Une servlet doit implémenter l'interface
  - **`javax.servlet.Servlet`**
  - soit directement,
  - soit en dérivant d'une classe abstraite implémentant déjà cette interface telles que
    - **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(ServletConfig config)
        throws ServletException {…..}

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

    public void destroy() {…..}
}
```

Le cycle de vie

- 1. la *servlet* est créee puis initialisée **`init()`**
  - cette méthode n'est appelée par le serveur qu'une seule fois lors du chargement en mémoire par le moteur (ou conteneur) de servlets
- 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 surchargent **service()** de la classe mère :
  - **doGet()** : pour les requêtes Http de type GET
  - **doPost()** : pour les requêtes Http de type POST
- la classe *HttpServlet* 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 HttpServlet (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
    - String getParameter (String Name) retourne la valeur associée à un paramètre
- **`HttpServletResponse`** contient le flux de sortie pour la génération de la page HTML résultat (**`getWriter()`**)
    - Mais on peut aussi envoyer des données binaires au client (ServletOutputStream ; getOutputStream)

Structure de base d'une HttpServlet

```
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 (sans balises de présentation HTML)

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

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

    PrintWriter out = response.getWriter();
    out.setContentType ("text/html")
    out.println("Hello World");
    }
}
```

Un exemple générant du HTML

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

public class HelloWorld extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                        throws IOException, ServletException        {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Hello World!</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Hello World!</h1>");
            out.println("<a href=/www.monserveur.fr/…/servlet/Hello2>Hello2</a>");
            out.println("</body>");
            out.println("</html>");
   }
```


Compiler et installer cet exemple

- L'installation varie d'un serveur à l'autre
- Pour compiler
  - Positionner le classpath, puis **depuis** le répertoire trivial/

javac -classpath .:<tomcat_dir>/lib/servlet.jar HelloWorld.java

- Deux manières de tester :
  - Avec JSWDK, mettre trivial.HelloWorld dans le répertoire
    <hierarchie_webpages>/WEB-INF/servlets
  - Avec Tomcat : il faut créer une " application web "

Une application Web au sens de Tomcat par exemple

- C' est un espace virtuel qui contient
  - html, images, servlets,jsp…
- Editer <tomcat_dir>/server.xml pour définir l' application Web

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

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

L' application Web (suite)

- le fichier <tomcat_dir>/sv/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>
                  trivial.HelloWorld
            </servlet-class>
      </servlet>
</web-app>
```

L'application Web (suite)

- Une fois une application web créée, on peut mettre autant de servlets que l'on veut...
- Si version antérieure à 3.3.2 relancer Tomcat à chaque modif des fichiers XML
  server.xml et/ou web.xml
- Si version égale ou supérieure, inutile chargement dynamique des classes
- Pour invoquer la servlet, utiliser l'alias :
  `http://host:port/sv/servlet/HelloWorld`
  - ou le nom complet...
    `http://host:port/sv/servlet/trivial.HelloWorld`

Charger et invoquer une servlet

- A partir d'une URL telle que

http://host:port/<web-app>/servlet/<servlet_class>?param1=value1&…

Machine où se trouve le serveur manon

Numero de port sur lequel il est à l'écoute 8080

chemin

Mot réservé

Nom complet de la classe avec le package

Liste des paramètres et de leurs valeurs

A comparer à l'alias :
`http://host:port/sv/servlet/HelloWorld`

Récupération des paramètres passés à la servlet

- Utilisation des méthodes de la classe *HttpServletRequest* :

```
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

```
package trivial;
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>Un autre exemple</title></head>\n" +
        <h1>Hello WWW</h1>\n" +"</body></html>");
    }
}
```

Ecrire une classe utilitaire

Remarque : <DOCTYPE...> et <HEAD...> toujours pareils !

```
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");
      }
}
```

- ne suffit pas ! Bien trop laborieux!
- La vraie solution c' est les JSP

Nouvelle version

```
package trivial;
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>");
}
}
```

Formulaire HTML: Récupérer 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);
        }
}
```


Cookies

- Un cookie est constitué des paires clé=valeur envoyés par le serveur ss forme d'instructions insérées dans l'entête de la réponse HTTP
- Un cookie permet
  - d'identifier des utilisateurs (e-commerce)
  - d'éviter la saisie d'information à répétition
    - Login, password, adresse, téléphone
  - Gérer des «préférences utilisateur» pour les sites «portails»
- Cookie et sécurité
  - Jamais interprété ni exécuté : pas de virus
  - Limité à 4KB; les navigateurs se limitent (300;20 par site)
  - Ne doit pas concerner des données sensibles

Gestion des cookies

- Utiliser les fonctions de l'API des servlets…
  - Pour 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
      - [ ] ( ) = , " / ? @ : ;
- Exemples

  " userperf "," noframes "

Attributs des cookies

- **`getValue/setValue`**
- **`getName/setName`**
- **`getComment/setComment`**
- **`getMaxAge/setMaxAge`** : délai restant avant expiration du cookie (en seconde)
  - par défaut : pour la session courante
- **`getPath/setPath`** : répertoire où s'applique le Cookie dir. courant ou pages spécifiques

Récupérer des cookies

```
Cookie [] cookies = request.getCookies();
String nom = getCookieValue (cookies, "nom", "non trouvé");
...
public static String getCookieValue (Cookie [] cookies,
String cookieName, String defaultValue)
{
for(int i=0; i < cookies.length; i++) {
      Cookie cookie = cookies[i];
      if(cookieName.equals(cookie.getName())
return(cookie.getValue());
}
return (defaultValue);
}
```

Temps d'expiration réglé par le développeur

- 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 : Problématique

- Protocole HTTP = protocole Internet "déconnecté"
  - différent de Telnet, Ftp, …
  - les requêtes et les réponses ne sont pas des transactions
- Certaines applications Web (e-commerce : caddie) ont besoin de maintenir une "mémoire" entre deux requêtes d'un utilisateur
- Une session = ensemble de transactions *apparentées* d'un même utilisateur

Suivi de session avec les servlets Java

- But: transfert des données générées par une requête vers les requêtes suivantes
- 4 méthodes
  - 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)

Récriture d'URL

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

```
<a href="http://www.monserveur.fr/servlet/foo?uid=sv">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 :
  - cacher les données de session dans des champs "hidden" :

<INPUT TYPE="HIDDEN" NAME="**uid**" VALUE=**sv**">

- Limitations :
  - idem la "récriture 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 *à chaque requête d'un utilisateur*
  - Il permet de stocker des informations *persistantes*
  - Durée de vie limitée et réglable

Modèle de base

Dans le code de doPost

```
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);
}....
```


Methodes (public) de la classe HttpSession

- **String getID ()**
  - Génère l'ID de la session
- **boolean isNew ()**
  - true si la session est crée mais pas encore envoyée au client
- **Object getValue (String name)**
  - Génère la valeur correspondant au nom spécifié
- **void putValue (String name, Object value)**
  - Ajoute un élément dans la session
- **void removeValue(String name),**
  - Enlève un élément de la session
- **long getCreationTime () / long getLastAccessedTime()**
- **int setMaxInactiveInterval (int interval)**
- **Void invalidate()**
  - Supprime la session en cours
- **…**

JSP

- Du java dans une page html !

  http://www.monserveur.fr/jsp/Test.**jsp**?titre=ESSAI

  . . .

  <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"

Les JSP (Java Server Pages)

une technologie qui permet de mélanger Java et HTML (ou XML)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head><title>Magasin virtuel</title></head>
<body>
<h1>Bienvenue dans notre magasin virtuel </h1>
<small>Bonjour,
<!– retrouver le nom depuis la dernière visite -->
<% out.println(Utils.getUserNameFromCookie(request)); %>
Pour accéder à votre compte, cliquez
<a href="Account-Settings.html">ici.</a></small>
<p>
        etc.
</body>
</html>
```

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) -->
    <%
     <!- - génère les majuscules de l'alphabet et change la première lettre -->
        for(int i = 0; i < 26; i++){
        for(int j = 0; j < 26; j++){
            c = (char)(0x41 + (26 - i + j)%26);
    %>
    <!- - génère la valeur de c.toString() dans la page HTML -->
    <%=  c %>
    <%                      } %>
<br>
<%                          } %>
</body></html>
```


Balises JSP dans 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(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 :
  - page : informations relatives à la page
  - include : fichiers à inclure littéralement
  - Taglib: URI de bibliothèque de balises utilisées dans la page

La directive page - exemples

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

La directive include

- exemples :

<%@ include file="chemin relatif du fichier" %>

<%@ include file="/top.html" %>

correspond au *home dir* du serveur

- Interprété littéralement, le fichier peut contenir:
  - HTML, scripting elements, directives, actions, …
- L'insertion se fait au moment de la traduction de la page …

Objets implicites

- Sont des variables prédéfinies accessibles dans les *scripting elements* :
  - request : *ref d'objet de type* **`HttpServletRequest`**
  - response : **`HttpServletResponse`**
  - session **`:  HttpSession`**
  - out : flot de sortie (idem **`response.getWriter()`**)
  - application : **`ServletContext`** (idem **`getServletConfig().getContext()`**)

Actions (1)

- Syntaxe XML
- Permettent de faire des actions au moment où la page est demandée par un client
  - Inclure dynamiquement un fichier
  - Rediriger vers une autre page
  - Utiliser des ***java beans***
  - 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:forward page="/unAutreURI" />`
  - redirige vers un autre URI/URL
- `<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.../>`

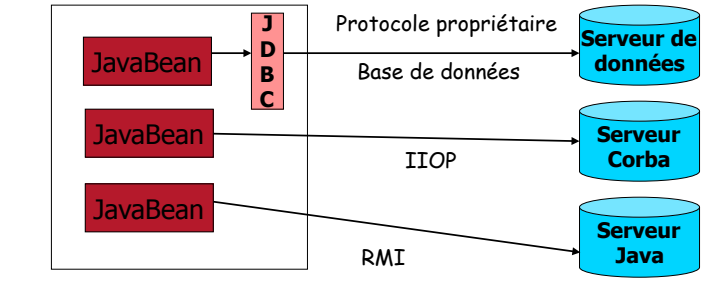
Les "Java Beans": Premiers pas de Sun vers les composants logiciels

- Définition initiale:
  - « java beans is a reusable software component that can be manipulated visually in a builder tool »
- Sémantique:
  - essentiellement graphique

par exemple le package Swing est composé de JavaBeans

  - Mais aussi des Beans « invisibles »; peu connus, pourtant…

Java Beans: exemples d'utilisation



JMV + Conteneur de servlets

Java Beans: Caractéristiques

- Supporter les fonctions:
  - Introspection: réflexion, bean Info
  - Customisation:
    - Arranger/Déployer
    - Règle d ' écriture
  - Événement:
    - Communique par événement, « *listener* »
  - Persistance:
    - *serializable*
    - sauvegarde l' état courant
    - rendre des attributs non sauvegardables

Exemple d'utilisation d'un bean

- La page JSP :

```
<html> ...
<jsp:usebean id="test" class= "mon_package.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 mon_package;

public class SimpleBean {
        private String message = "no message";

        public String getMessage() {
        return message;
        }

        public void setMessage(String message) {
        this.message = message;
        }
}
```

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 !

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" />
```

Un exemple complet
«consulter en intranet la liste des étudiants d'AWEB »

- En utilisant une servlet, rechercher l'adresse électronique, le numéro de casier d'un étudiant et l'afficher
- 3 parties
  - La page HTML pour le formulaire
  - La servlet effectuant la requête
  - Et la page HTML résultat générée par la servlet

recherche.html

Le formulaire

```
<HTML>
<HEAD><TITLE>Liste AWEB</TITLE></HEAD><BODY>
<CENTER><H1> Liste AWEB </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://manon:8080/servlet/Liste>

<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>
```

Structure de la servlet Liste (1)

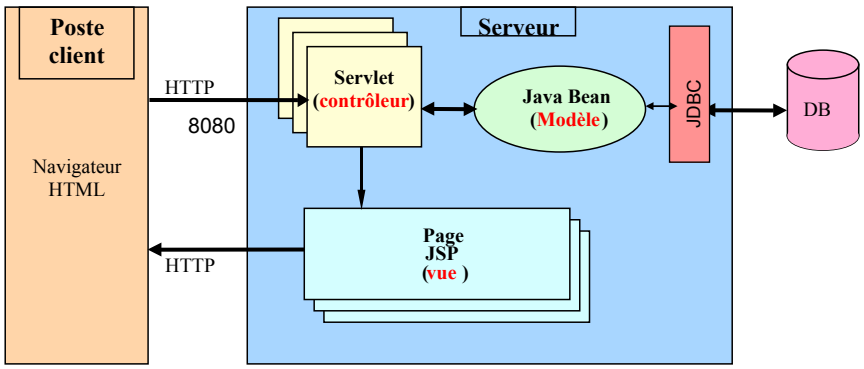
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;

public class Liste extends HttpServlet
{
public void doPost(HttpServletRequest req,
                   HttpServletResponse res)
            throws ServletException, IOException {
      ...
      }
public String getServletInfo() {
      return "La servlet Liste";
      }
}
```


La servlet Liste (2)

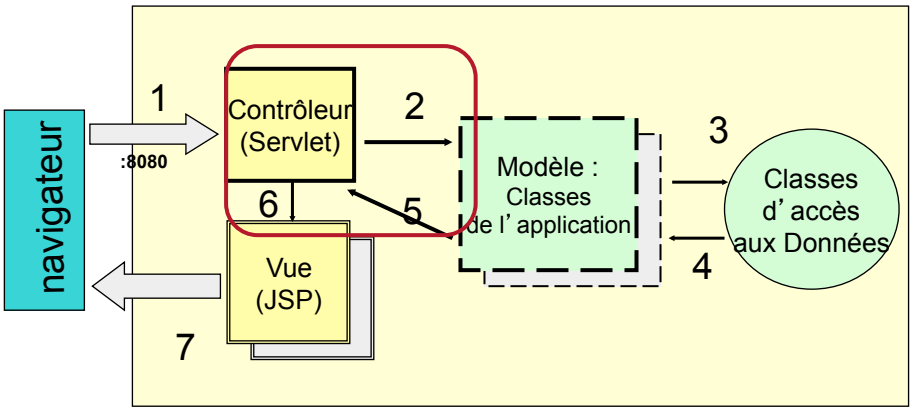
```
public void doPost(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {
  res.setContentType("text/html");
  PrintWriter out = res.getWriter();
  out.println("<HTML><BODY>");
  out.Println("<CENTER><H1>Voici les coordonnées :</H1></CENTER>");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

Le modèle de conception modèle-vue-contrôleur



Le contrôleur = servlet unique
traite les demandes du client = traite les paramètres [action] de la requête HTTP
Le modèle est réalisé par une ou des archives .jar
Le contrôleur choisit la vue

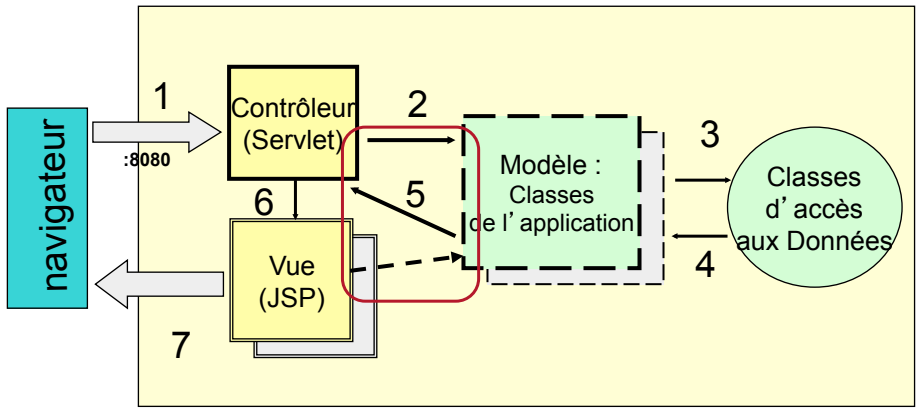
MVC avec Struts



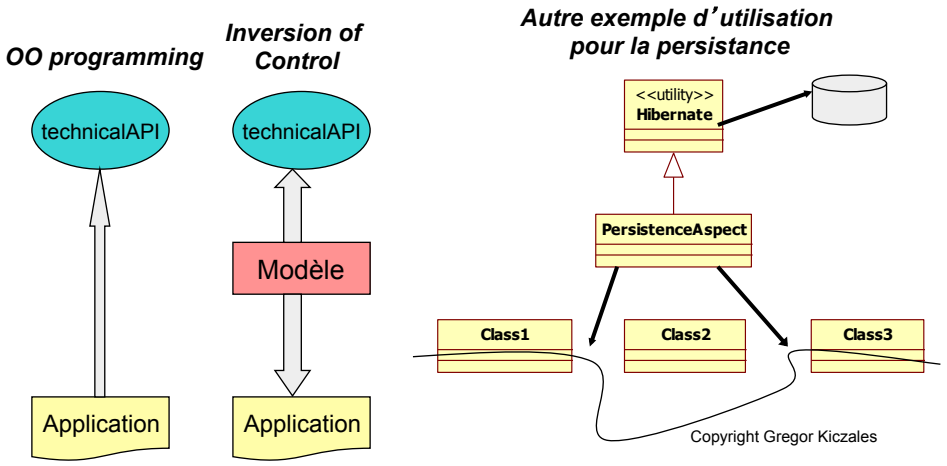
MVC avec Struts

- Les méthodes particulières de la servlet/contrôleur sont traitées par des instances dérivées de la classe *Action* de Struts
- Il faut écrire des fichiers de configuration
  - web.xml
  - struts-config.xml
- + quelques modifications des pages JSP
- Modèle : mêmes archives .jar

MVC avec Spring (principe)



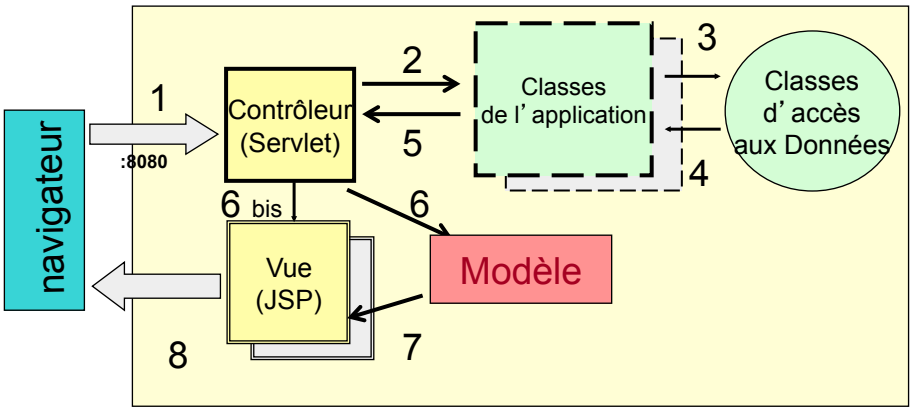
En plus, dans Spring modèle de conception IoC: Inversion of Control



MVC avec Spring

- Le contrôleur est assuré par une servlet fournie par Spring *DispatcherServlet* qui accède à la couche métier
- Les vues = pages JSP (quelques détails à changer)
- Le modèle = mêmes archives .jar
- Fichiers de configuration
  - *web.xml* ce fichier contient un listener qui au moment de l'application va exploiter
  - *applicationContext.xml* qui crée des beans d'accès aux modèles

Implémentation MVC par Spring



Et JSF ?

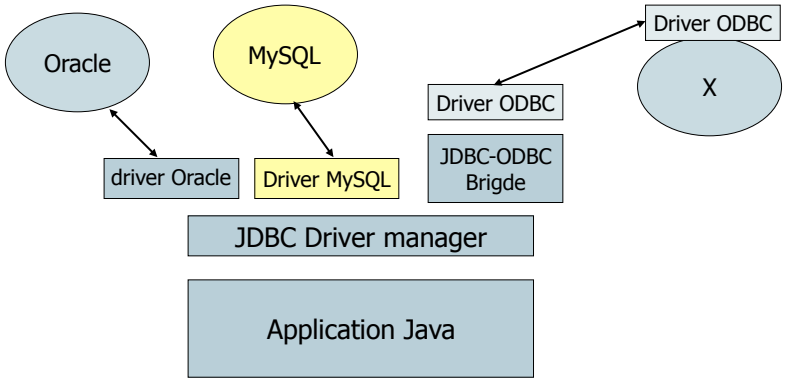
- Java Server Faces
- Spécification Java EE de composants *Faces*
- S' appuie sur MVC2
  - JSF est dédié à l' amélioration de la partie Vue du modèle
- Implémentation Apache MyFaces
- Gère les Vues en forme d' arbre de composants graphiques (UIComponents)
- avec des classes *renderers* de rendu graphique
- Permet de traiter des services communs :
  - navigation ou gestion de messages

Java et les Bases de données JDBC

- Java Data Base Connectivity
- API Java (indépendant de la plate-forme) pour interagir avec des BD relationnelles
  - lancer des requêtes SQL
  - récupèrer les résultats
- Tentative de standardiser l'accès aux BD

JBDC

le driver manager est un objet Java.
Une application peut accéder à plusieurs BD simultanément



Les interfaces JDBC

- Driver : tous les pilotes respectent cette interface
- Statement : la méthode execute sert à exécuter des requêtes
  - 2 sous-interfaces :
    - PreparedStatement : instructions prédéfinies
    - CallableStatement : procédures stockées
- ResultSet permet de stocker sous forme ensembliste le résultat d'une requête SQL
- (Accès aux Meta Données)

Accès aux données

1. Enregistrer le *driver*
   - *utiliser la méthode* forName *de la classe* Class
   - Class.forName ("com.mysql.jdbc.Driver")
2. Accès à la base
   - via une URL
     - Syntaxe générale jdbc:nomprotocole:paramètres
     - jdbc:mysql://serveur/nombase
   - **jdbc:mysql://mysql.tp.enst.fr/magasini** (port par défaut de MySQL 3306)
3. Ouverture de la connexion
   Connection C = DriveManager.getConnection(url,user,password);
4. Création d'un *statement*
5. Exécution de la requête
6. Lecture des résultats

Toutes les méthodes doivent prendre en compte l'exception SQLException

1-Enregistrement du driver

- Disposer d'un driver propriétaire : xxx.jar
- Charger le driver :

```
pilote = "com.mysql.jdbc.Driver"
try {
Class.forName(pilote).NewInstance() ;
}
catch (ClassNotFoundException e) {
System.err.println("Exception: " + e.toString());
}
```


2 et 3 - Ouvrir une connexion

- Utilisation d'une méthode *static* de la classe *DriverManager*

```
try
{
connect = DriverManager.getConnection
                    (urlBase,"nomLogin", "nomPassword");
      if (connect ==null) … lever une exception pb avec le serveur
      catch (SQLException e) … récupérer l'exception connexion à votre BD

}
```

- En général, on met les paramètres de connexion dans un fichier texte

```
jdbc.url= jdbc:mysql://mysql.tp.enst.fr/magasini
jdbc.user=…
dbc.password=…
```

4- L'INTERFACE STATEMENT

- permet de définir les requêtes SQL

```
Statement stmt = connect.createStatement();
```

- Permet d'exécuter deux types de requêtes :
  - Les requêtes de modification de la base
  - Les requêtes de consultation de la base

5 - Modification de la base

- API :

```
int executeUpdate(String requeteSQL)

stmt.executeUpdate("create table tester +" (num integer, ch text)") ;
stmt.executeUpdate(insert into tester values (1,'dupont')) ;
stmt.executeUpdate(insert into tester values (2,'durant')) ;
```

- Valable pour les requêtes du type ***select * into ...***
- retourne le nombre de lignes créées, modifiées par la commande

Consultation de la base

- API : on utilise la méthode executeQuery et on récupère le résultat dans un objet ResultSet

```
ResultSet executeQuery(String requeteSQL)

ResultSet rs = stmt.executeQuery("select * from tester") ;
while (rs.next ())
 {
   //parcours de chaque ligne
   int sonAge=rs.getInt("Age")
 }
```


Récupération de données pour chaque type de base

- La classe ResultSet dispose des méthodes suivantes :
  - int getInt(int i)
    - retourne un entier de la colonne i
  - int getInt(String col)
    - retourne un entier de la colonne col
  - String getString(int i)
    - retourne l'objet String de la colonne i
  - String getString(String col)
    - retourne l'objet String de la colonne col
  - boolean next()
    - retourne true s'il reste un n-uplet à lire
  - Date getDate(int i)
    - retourne l'objet Date de la colonne i
  - Date getDate(String col)
    - retourne l'objet Date de la colonne col

Récupération des données

- Un exemple :

```
int i = 0;
while (rs.next()) {
int age = rs.getInt("age");
String nom = rs.getString(« nom");
System.out.println("ligne " + i + ": " + age + ", " + nom);
i++;
}
```

- next() est obligatoire même pour lire une seule ligne

- JDBC 1.0 oblige à parcourir la table résultat
- JDBC 2.0 déplacement amélioré

JDBC 2.0

- Déplacement amélioré
- Mise a jour de la base en utilisant des méthodes Java au lieu de commandes SQL
- Envoi de plusieurs commandes SQL (batch)
- Quelques adaptations à SQL 3
- Package java : javax.sql

en JDBC2 Scrollable ResultSets

- Nouvelle définition pour obtenir une table résultat :

```
Statement stmt ; ResultSet srs ;
stmt = db.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                          ResultSet.CONCUR_UPDATABLE);
srs = stmt.executeQuery("SELECT * from joueur");
```

- Constantes prédéfinies au niveau de ***ResultSet*** :
  - TYPE_FORWARD_ONLY
    - parcours dans un seul sens
  - TYPE_SCROLL_SENSITIVE
    - sensible aux changements
  - TYPE_SCROLL_INSENSITIVE
    - non sensible aux changements
  - CONCUR_READ_ONLY
    - non modifiable
  - CONCUR_UPDATABLE
    - modifiable

Les possibilités de déplacement

srs.next()	ligne suivante
srs.previous()	ligne précédente
srs.absolute(i)	aller a la ième ligne
srs.absolute(-i)	aller a la ième ligne en partant de la dernière
srs.relative(i)	descendre de i lignes
srs.relative(-i)	remonter de i lignes
srs.afterlast()	fin de la table
srs.isAfterLast()	retourne vrai si fin de table
srs.last()	aller à la dernière ligne

Fermeture de la connexion

- Fermer le dialogue SQL
- Fermer la session avec la base

```
stmt.close() ;
connect.close() ;
```


Création d'un statement

- 3 types de *statement*
  - statement : requêtes simples
  - (prepared statement : requêtes précompilées)
  - (callable statement : procédures stockées)

- Création d 'un statement

```
Statement s = c.createStatement ();
```

Exécution d'une requête

- 3 types d'exécutions
  - executeQuery quand les requêtes retournent un ResultSet
  - executeUpdate pour les requêtes
    - INSERT, UPDATE, DELETE, CREATE TABLE et DROP TABLE
  - execute pour les procédures stockées

- Exécution de la requête

```
String myQuery = « SELECT prenom, nom, email » +
      «FROM employe » +
       «WHERE (nom= 'lenom' ) AND (email IS NOT NULL) +
       «ORDER BY nom »;

ResultSet rs = s.executeQuery (myQuery);
```

6 - Lecture des résultats

- executeQuery renvoie Resultset (itérateur)

- donc rs se parcourt itérativement row par row

```
Statement s = c.createStatement ();
String myQuery = «SELECT num,nom FROM matable»)

ResultSet rs = s.executeQuery (myQuery);
while (rs.next())
{
        int i = rs.getInt(«num»)
        String s = rs.getString(«nom»)
        System.out.println («ligne=» + num + «  » + nom)
}
```

INF347 – Web 2 : concepts et outils

GWT Google Web Toolkit

Sylvie.Vignes@telecom-paristech.fr

École Nationale Supérieure des Télécommunications
Département Informatique et Réseaux

Le Web 2 …

- enrichir des applications Web avec des composants Ajax
  - Asynynchronous JavaScript XML (+API DOM)
- pour créer des applications « similaires à des applications bureautiques locales »
- mais qui soient accessibles dans un navigateur Web
- Conséquences du *style Ajax*
  - Interaction sophistiquée : écran partiellement mis à jour
  - Session client:
    - états temporaires stockées dans le client
    - états utilisateur persistants sur le serveur

Problèmes de JavaScript?

- Difficile à écrire, à mettre au point
- Interprétation différente selon navigateur
- …
- Update du back button seulement quand une nouvelle page remplace l'ancienne
- Java Script ne peut *downloader* que des données en provenance de la machine qui possédait le document HTML contenant le tag <script>

Pourquoi GWT?

- Pour suppléer à JavaScript
- GWT permet de générer du code JavaScript à partir de code Java
- Pour la communauté Java
  - "Write Once, Run Anywhere"
  - IDE : Plug-in Eclipse
  - debug pas à pas
  - 1 code source pour tous les navigateurs
  - Optimisaton
    - génération de code Javacript «utile» et compacté
- GWT ou … AJAX chez Google
  - ne résout pas tout mais fournit une « abstraction » plus manipulable plus proche de SWING plus que DOM!
  - Résout back button, compatibilité navigateurs, RPC/serialization
  - http://code.google.com/webtoolkit

GWT

- Framework hybride Java/Ajax
- Modèle de programmation simple
- Maintenu par Google

- Avantages développeur
  - Java …
  - Shell pour tester des modules : debug sans compilation
  - Composants UI « Object Oriented » ni AWT ni SWING
  - RPC simple asynchrone navigateur-serveur

- Avantages pour utilisateur
  - Réponse rapide aux actions
  - Compatibilité navigateurs
  - Pas besoin de plug-ins ou JVM
  - Gestion de l'historique sur le navigateur

Principes

- Construction d'interfaces graphiques web en langage Java (java.util.*, java.lang.* …)

- GWT fournit une API Widgets Graphiques Panel, Button, TextBox … sont des classes extensibles

- Compilation du code java en Javascript

- Application web = code javascript + html + css + …

- JavaScript téléchargé et exécuté par le navigateur

Bibliothèques de Widgets

- Projet de widget graphique
  - Ext GWT = GXT (commercial)
  - GWanted (open source)
  - GWT-Widget (open source)
- Projet encapsulant des librairies JS
  - SmartGWT : SmartClient (open source)
  - GWT-EXT : ext ≤ 2.0.3 (open source)
- Tatami : dojo (open source)

Interaction client- serveur

- Définition de services coté serveur
  - Interface Java de définition du service (méthodes)
  - Implémentation du service par une Servlet
- Appel des services par le code client
  - RPC Asynchrone sur http
  - Interface Java de réception du retour d' appel (CallBack)

RPC GWT exemple

- Définition de l'interface du service

```
public interface MyService extends RemoteService {
String myMethod(String s);
}
```

- Implémentation de l'interface coté Serveur

```
public class MyServiceImpl
extends RemoteServiceServlet implements MyService {
public String myMethod(String s) {
return s + "." + s;
}
}
```


RPC exemple (suite)

- Définition de l'interface de callback

```
public interface MyServiceAsync {
    void myMethod(String s, AsyncCallback<String> a);
}
```

- Appel du service depuis du code coté navigateur

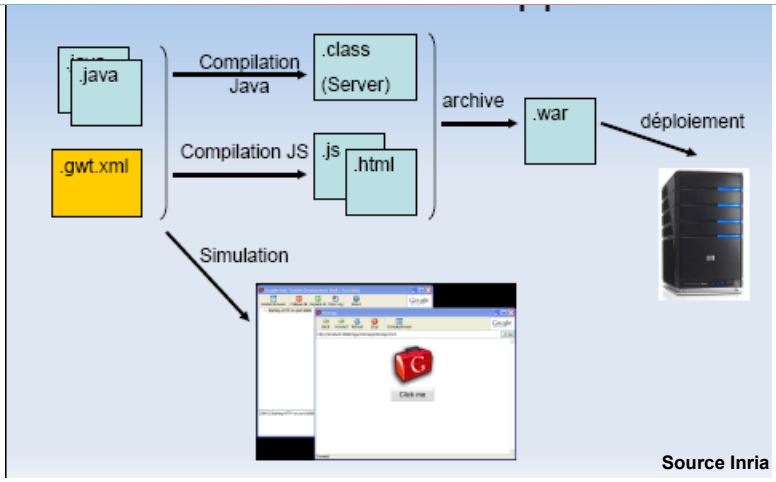
```
MyServiceAsync myService = (MyServiceAsync)
    GWT.create(MyService.class);
((ServiceDefTarget) myService).setServiceEntryPoint(
GWT.getModuleBaseURL() + "myservice");
AsyncCallback<String> callback = new AsyncCallback<String>() {
      public void onSuccess(String result) { // do some UI stuff to show success }
      public void onFailure(Throwable caught) { // do some UI stuff to show failure }
};

myService.emptyMyInbox("toto", callback);
```

Internationalisation

- pour :
  - Supporter plusieurs langues (fr, en, …)
  - Permettre le choix des terminologies par un non développeur
- Séparer le code du texte
- Externaliser les textes dans un fichier .properties
  - Utilisation simple via une interface Java + Annotations
- Support de messages construits (avec paramètres)

Processus de développement



Test avec Junit

- Tests unitaires
- Hériter de la classe GWTTestCase
- Implémenter getModuleName() pour récupérer son module
- Utilisation d'assertions
- Test du modèle, pas de l'interface

Les concurrents

- Flex :Adobe
  - Seul le SDK en open source (réponse à GWT)
    - Outils chers
    - Bibliothèque de widgets très chers
  - Description xml de l'interface
  - Lecteur flash nécessaire
- Microsoft Silverlight(C# , VB.NET)

Les références

- Le site officiel : http://code.google.com/webtoolkit
  - La démo de GWT
    - http://gwt.google.com/samples/KitchenSink/KitchenSink.html
  - Smart GWT : http://code.google.com/p/smartgwt/
  - GXT (ext GWT) : http://extjs.com/products/gxt/
  - Ongwt.com

L'architecture REST

Technologies de communication inter-applications

- API : on embarque directement l'application à appeler (dépendance directe, JAR)
- EJB : on appelle l'application avec un protocole de niveau transport (TCP/IP) ou directe (si même JVM)
- REST : échange niveau HTTP, on exploite à 100% le protocole
- SOAP : au dessus d'HTTP, on encapsule les messages

REST

- **Re**presentational **S**tate **T**ransfer
- Roy Fielding http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- Pour tirer profit de la sémantique du Web
  ET
  du protocole HTTP

Principe

- On utilise les méthodes HTTP sur les ressources exposées afin d' effectuer toutes les opérations voulues. On est donc toujours sans état.
- Technologies utilisées
  - HTTP
  - URI
  - XML / JSON (représentation des ressources)
  - Types MIME

JSON

- **J**ava**S**cript **O**bject **N**otation
- Utilisé pour les services exposé directement vers le navigateur (Mashup)
- Structure :
  - Paires clé / valeurs
- Les valeurs peuvent être des listes de paires, des tableaux de paires, des objets, chaines de caractères…

Exemple de JSON

```
{"menu": {
   "id": "file",
   "value": "File",
   "popup": {
     "menuitem": [
       {"value": "New", "onclick": "CreateNewDoc()"},
       {"value": "Open", "onclick": "OpenDoc()"},
       {"value": "Close", "onclick": "CloseDoc()"}
     ]
   }
 }}
```

Utilisation de JSON

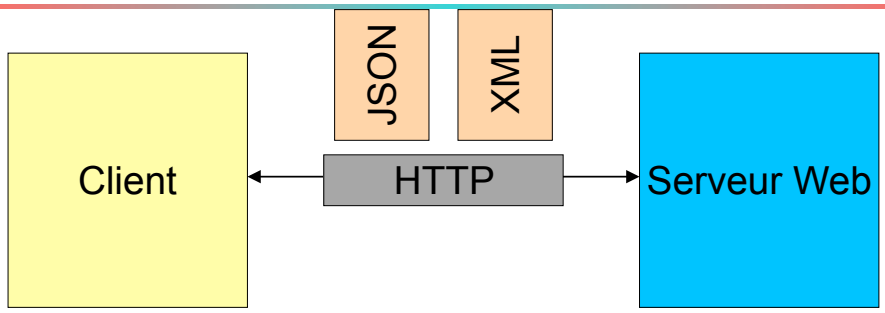
- JSON est une structure de donnée JavaScript.
- Pour l'interpréter, on utilisait la fonction JavaScript **`eval()`**, dangereuse :

```
var donnees = eval('('+donnees_json+')');
```

- Natif dans les navigateurs depuis 2009 (IE8, Firefox 3.5, Opera 10.5) :

```
var donnees = JSON.parse(donnees_json);
```

Communication REST



CRUD

CRUD	REST	
CREATE	POST	Create a sub resource
READ	GET	Retrieve the current state of the resource
UPDATE	PUT	Initialize or update the state of a resource at the given URI
DELETE	DELETE	Clear a resource, after the URI is no longer valid

2009 - Cesare Pautasso

Utilisation de REST

Paul Downey, BT