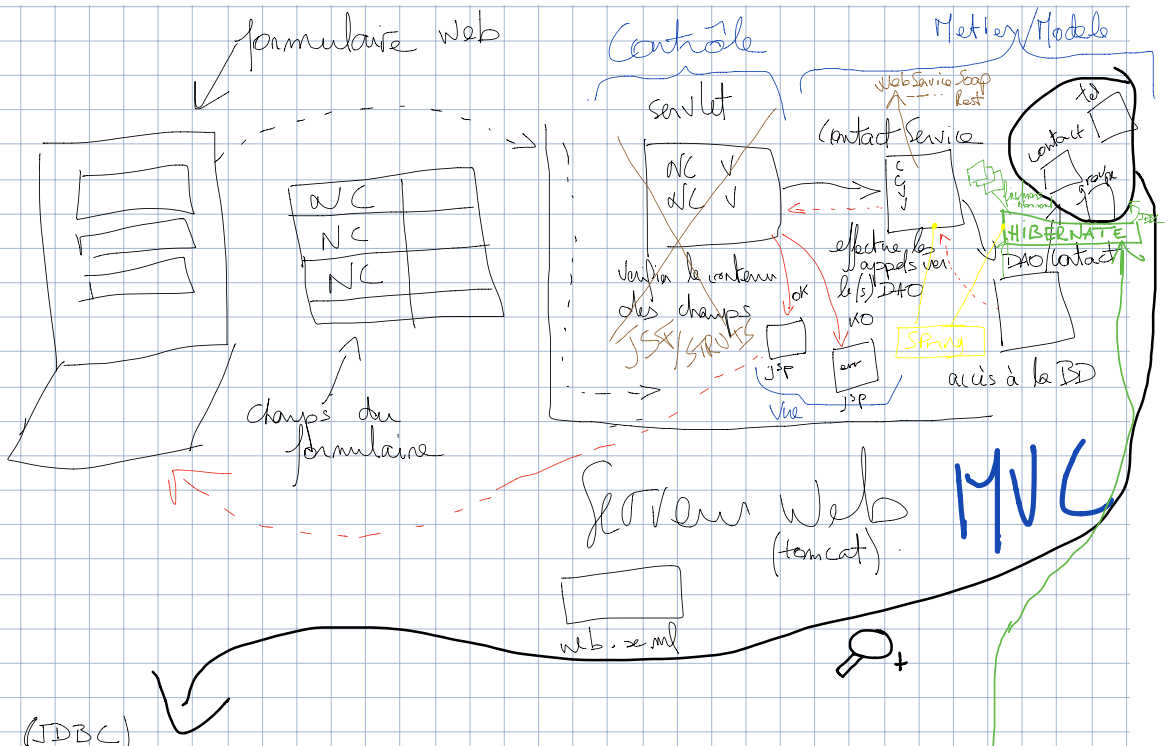


(JDBC)

```
CreateContact(_,_,_)
DriverManager.-
Connection(...)
Statement(INSERT INTO ...)
ExecuteUpdate(...);
```

① Metier ⇒ SQL
② Metier Relationnel et non OO
③ Forte dép vers SGBD
④ 40% du code et du chronomat JAVA ⇒ ↓ lisibilité code
⑤ SQL non optimisé

Remplacer par

```
Contact c = new Contact(_,_,_);
hibernate.save(c);
```

⇒ Plus lisible !!!

Code SQL généré par hibernate est optimisé (fait par des experts)

Serveur web
≠
Serveur application (=WS mais interprète EJB)

```
CreateContact(_,_,_){
DAOContact daoc = new DAOContact();
    daoc.createContact(_,_,_);
}
```

⇓ Spring

```
IDAOContact daoc = (IDAOContact) spring.getBean(uc);
daoc.createContact(uc);
```

⇒ IOC (inversion de contrôle)
ID (injection de dépendances)

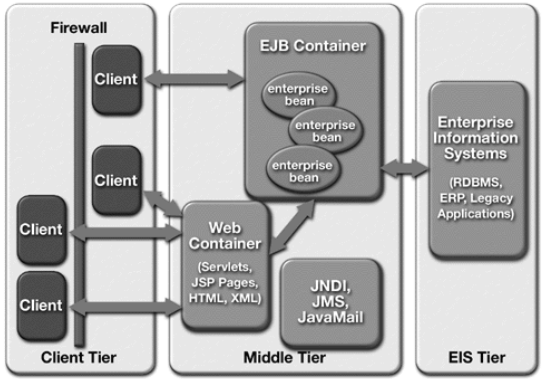
ajoute d'autres méthodes de création

d'instance sans avoir à changer le code métier

Servlets & JSP

Reda Bendraou
Reda.Bendraou@lip6.fr
pagesperso-systeme.lip6.fr/Reda.Bendraou/
(based on a course from Lionel Seinturier)

JEE Platform



Introduction

Servlet & JSP

A Java program executed on the **Server side**

- ✓ servlet : Independent program stored in a **.class** file on the server
- ✓ JSP : Java **source** code embedded in **.html** page

	Client Side	**Server side**
Independent Class	Applet	**Servlet**
Embedded in HTML	JavaScript	**JSP**

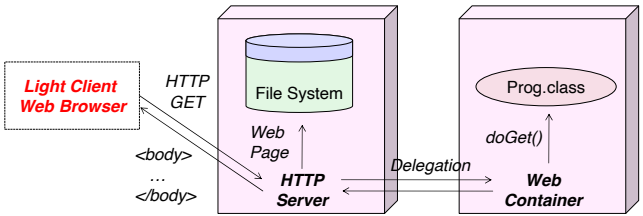
Servlet et JSP

- ✓ Need a web container with a Servlet engine to be executed (ex. **Tomcat**)
- ✓ A JSP page is automatically compiled into Servlet before execution

Introduction

Servlets: Principles

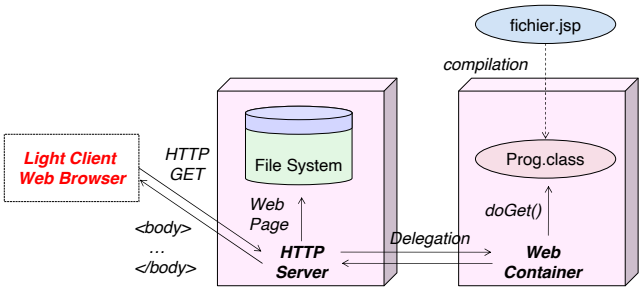
- ✓ .class files are stored on the server side
- ✓ Handled and managed within a Web Container (Tomcat)
- ✓ Accessed through a **URL** http://www.lip6.fr/maservlet/Prog
- ✓ The **Loading of** the l'URL triggers the **execution** of the Prog.class servlet via the web container (usually called Servlet Container)



Introduction

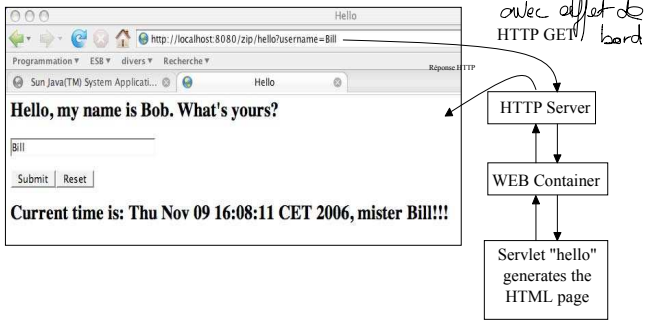
JSP: Principle

A JSP page is compiled into a Servlet in order to be executed!

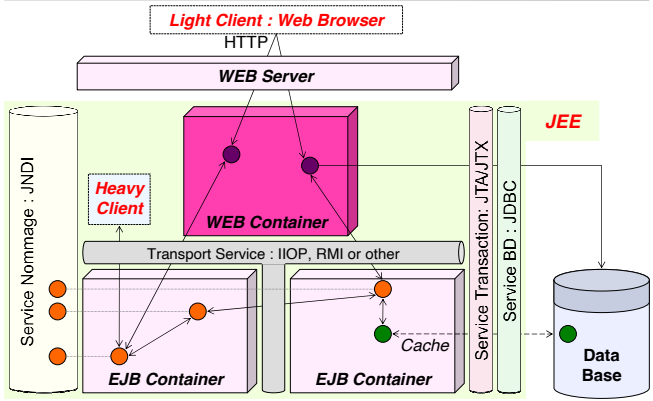


Introduction

Servlet: How it works!



Introduction



First Step : The Servlets

Servlet : Realization

Writing a servlet = Writing a **Java class**

On the initial loading of the servlet (or after a modification), the **Web Container**

- ✓ **instantiates** (and initializes) the servlet
- ✓ servlet = Java objet present within the servlet engine

Then, for the other invocations, the **Web Container**

- ✓ Executes the servlet code in a separate ***thread***
- ✓ The result of executing the code is sent back to the client
- ✓ In case of an exception in the servlet's java code, a message is displayed on the client's browser

Servlet : Realization

Servlet Development (coding)

Use of the packages: Java **javax.servlet.*** & **javax.servlet.http.***

- ✓ You must extend the **`javax.servlet.http.HttpServlet`** class
- ✓ You must redefine the **doGet()** or **doPost()** of this class
  - doGet : corresponds to a HTTP GET request
  - doPost : corresponds to a HTTP POST request
- ✓ Include the code to be executed when the servlet is invoked
  Automatically called by the Web container upon a requeste

void doGet(HttpServletRequest request , HttpServletResponse response);

Request sent by the client !
Automatically filled by the Web Container

Response HTML outcome of the servlet!
To fill within the servlet code

Servlet : Realization

Insights of the servlet API

Most important methods of the **request** object:

- ✓ **`String getParameter(String param)`**
  Returns the **value** of the field param extracted from the form data
- ✓ **`java.util.Enumeration getParameterNames()`**
  Returns the set of parameter names transferred to the servlet from the html form (client side)
- ✓ **`String getMethod()`**
  Returns the HTTP method (GET or POST) used to invoke the servlet (the attribute "method" in the html form, on the client side)

Servlet : Realization

Insights of the servlet API

Most important methods of the **response** object:

- ✓ **`void setContentType(String type)`**
  Defines the **MIME type** of the document generated by the servlet execution
- ✓ **`PrintWriter getWriter()`**
  Returns the **output flow** allowing the servlet to generate its outcome
  The servlet writes the HTML code on this output flow

Servlet : Realization

Example of a servlet :

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

public class HelloServlet extends HttpServlet {
   public void doGet(HttpServletRequest request,
        HttpServletResponse response ) throws
        ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        // écriture du HTML pour le client
        out.println("<html><body>Current time is: " +
              new Date() + ".</body></html>");
        out.close(); } }
```


Servlet Life Cycle

Each Servlet is instantiated only once
⇒ persistency of instance variables between 2 invocations

```
public class CompteurServlet extends HttpServlet {
    int compteur = 0;
    public void doGet( HttpServletRequest request, HttpServletResponse response )
          throws ServletException, IOException {
          response.setContentType( "text/html" );
          PrintWriter out = response.getWriter(); out.println( "<html><body>" );
          out.println( "<h1> "+ compteur++ + "</h1>" );
          out.println( "</body></html>" );
     }
}
```

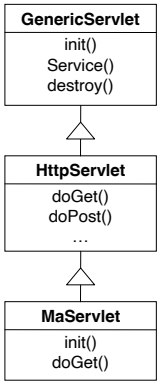

Servlet Life Cycle

Servlet life Cycle

- ✓ void init(ServletConfig conf)
  - Method called by the engine when initializing the servlet
  - Can be used to set some parameters of (used by) the servlet
  - Never use the constructor to initialize a Servlet
- ✓ void destroy()
  - A method called when the servlet is destroyed

HTTP methods:

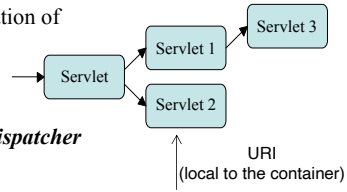
- ✓ service() handles all HTTP requests
- ✓ doGet(), doHead(), doPost(), doPut() doDelete(), doTrace()
  Proper to each HTTP request type



Chaining the Servlets

Results outcome from the aggregation of different servlets

- ✓ Better modularity
- ✓ Better reusability



Important: Use of the *RequestDispatcher*

- ✓ Obtained via the request object

```
RequestDispatcher rd = request.getRequestDispatcher( "servlet2" );
```

Inclusion of another servlet's result

- ✓ **rd.include(request, response);**

Delegation towards another servlet

- ✓ **rd.forward(request, response);**

Servlets API

Other methods of the request object

- ✓ String getProtocol() :
- ✓ String getServerName() / String getServerPort() :
- ✓ String getRemoteAddr() / String getRemoteHost() :
- ✓ String getScheme() :
- ✓ java.io.BufferedReader getReader() :

Cookies & Sessions with Servlets

Cookie = data stored by a Web server on a client's machine

- ✓ To save data about user/customer preferences
- ✓ The user has the possibility to forbidden their use by configuring the browser

Defined by class **javax.servlet.http.Cookie**

- ✓ Give a name and a value to the cookie
- ✓ **uneCookie = new Cookie("sonNom", "saValeur");**
- ✓ Use the response object to set the cookie **response.addCookie(uneCookie);**
- ✓ Extracted via the request object **Cookie[] desCookies = request.getCookies();**
- ✓ Some methods : String getName() / String getValue()

Cookies & Sessions with Servlets

Following a user session

- ✓ HTTP protocol is non-connected (stateless)
- 2 successive requests from the same user are considered independently by the server

Session : **following the user activities along its page browsing**

- ✓ A Session object associated to all user's **requests** (= IP @ + browser)
- ✓ Sessions **expire** (no request for n seconds ⇒ the session expires)

Created/Consulted from request object

- ✓ **HttpSession session = request.getSession(true);** returns the current session for this user or create a new session
- ✓ **HttpSession session = request.getSession(false);** returns the current session for this user or null

Cookies & Sessions with Servlets

Most important methods of the HttpSession object

- ✓ void **setAttribute**(String name, Object value); adds a pair of (name, value) for this session
- ✓ Object **getAttribute**(String name); returns the object associated to the key name ou null
- ✓ void **removeAttribute**(String name); removes the pair identified by the key name
- ✓ java.util.Enumeration **getAttributeNames**(); returns all attribute names associated to the session
- ✓ void **setMaxInactiveInterval**(int seconds); Specifies the remaining time before closing a session
- ✓ long **getCreationTime**(); / long **getLastAccessedTime**(); returns the creation date/ last access of the session in ms since 1/1/1970, 00h00 GMT new Date(long);

3 objets pour s'envoyer des objets : request session

Cookies & Sessions with Servlets

Important: sharing data between servlet

Execution Context = a set of pairs (name, value) shared by all the instantiated servlets

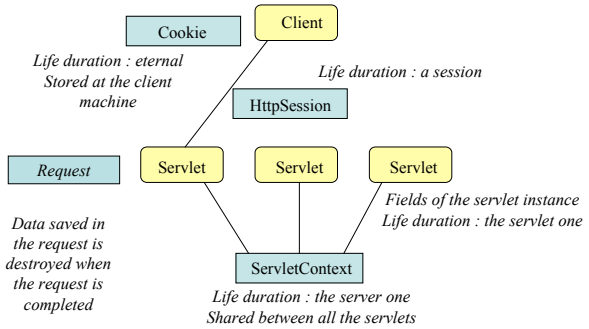
ServletContext ctx = **getServletContext**()

Methods of the ServletContext object

- ✓ void **setAttribute**(String name, Object value)
  adds a pair of (name, value) within the context
- ✓ Object **getAttribute**(String name)
  returns the object associated to the key name ou null
- ✓ void **removeAttribute**(String name)
  removes the pair identified by the key name
- ✓ java.util.Enumeration **getAttributeNames**()
  returns all attribute names associated to the context

Cookies et Sessions dans les Servlets

Summary of data object with servlets



Servlet : conclusion

Servlets :

- ✓ Portability, easy to write (**Java**)
- ✓ Executed in // using (***threads***)

But :

- ✓ Hard to write HTML code within Java code (mix)
  Introduction to the technology Java Server Pages (JSP)
- ✓ No integrated mechanism for dealing with distribution
  Introduction to the technology Enterprise Java Beans (EJB)

Second Step : JSP

JSP : how it works!

```
<html><body>
<h1>Table des factorielles</h1>
<% int i,fact;
   for ( i=1,fact=1 ; i<4 ; i++, fact*=i ) {
     out.print( (i-1) + "! = " + fact + "<br>" );
   } %>
</body></html>
```

Invocation
⇒
Execution:
Server Side

JSP : how it works!

```
<html><body>
<h1>Table des factorielles</h1>
<% int i,fact;
   for ( i=1,fact=1 ; i<4 ; i++, fact*=i ) {
     out.print( (i-1) + "! = " + fact + "<br>" );
   } %>
</body></html>
```

HTML Code
Java Code
Generated HTML
HTML Code

Code
Sent to
the
Client

```
<html><body>
<h1>Table des factorielles</h1>
0!=1<br>
1!=1<br>
2!=2<br>
3!=6<br>
</body></html>
```


JSP : how it works!

```
<html><body>
<h1>Table des factorielles</h1>
<% int i,fact;
   for ( i=1,fact=1 ; i<4 ; i++, fact*=i ) {
     out.print( (i-1) + "! = " + fact + "<br>" );
   } %>
</body></html>
```

After
Compilation

```
public final class fact_jsp /* ... */ {
  public void _jspService(HttpServletRequest request,
                          HttpServletResponse response) /{
    * ... */
    PrintWriter out = response.getWriter();
    response.setContentType("text/html");
    /* ... */
    out.write("<html><body>\n");
    out.write("<h1>Table des factorielles</h1>\n");
    int i,fact;
    for ( i=1,fact=1 ; i<4 ; i++, fact*=i) {
      out.print( (i-1) + "! = " + fact + "<br>" );
    }
    out.write("\n");
    out.write("</html></body>\n");  /* ... */ }
```


JSP : Realization

Several sections <% … %> may be used within the same page

During the first loading of a jsp

- ✓ A servlet is generated from the JSP
- ✓ Compilation of the servlet
- ✓ Instantiation of the servlet
- In case of syntactical error, a message is sent to the browser (errors detected at runtime)

For the following invocations:

Execution of the servlet within a thread

JSP : Realization

Implicit Objects within a JSP page

Objects usable within the Java code of JSPs

- ✓ out — output flow to generate the HTML code
- ✓ request — the request that caused the loading of the JSP
- ✓ response — the response to the request of loading the JSP
- ✓ Page — the servlet instance associated with the current JSP
- ✓ exception — exception generated in case of an exception in the JSP page
- ✓ session — session tracking for the same customer/user
- ✓ application — a data space shared between all JSPs

JSP : Realization

Equivalent implicit Objects in Servlets

Objects usable within the Java code of JSPs

- ✓ out — response.getWriter()
- ✓ request — le paramètre HttpServletRequest
- ✓ response — le paramètre HttpServletResponse
- ✓ Page — this
- ✓ exception — *pas d'équivalent immédiat*
- ✓ session — request.getSession(true)
- ✓ application — getServletContext()

JSP : Realization

Directive <%= ... %>

<%= expr %> displays the result of evaluating the expression expr

<%= expr %> shortcut for <% out.print(expr); %>

```
<html> <body>
<% int aleat = (int) (Math.random() * 5); %>
<h1> Nombre aléatoire : <%= aleat %> </h1>
</body> </html>
```

JSP : Realization

Instance Methods and variables

Instance methods and variables can be associated (defined) to a JSP

Between **<%!** and **%>**

Instance methods and variables **of the generated servlet**

```
<html> <body>
<h1>Compteur</h1>
<%!
  int cpt = 0;
  int getCpt() {     return cpt++;    }
%>

<h1> <%= getCpt() %> </h1>
</body> </html>
```

Instance Variable
initialized at instantiation time of the JSP
persistent between 2 invocations

Instance Method
attached to the object corresponding to the JSP

JSP : Realization

Instance Methods and variables

⇒ no access to implicit objects (out, request, page…) : objects defined within the principal servlet method (_jspService()) (or doGet, doPost for HttpServlet)

Important!

```
<%! int cpt = 0 %>
```

Instance Variable
⇒ assigned at initialization time

```
<% int cpt = 0 %>
```

local Variable
⇒ assigned for each invocation

<%! … %> : defines an instance variable (persistent between 2 invocations)
<% … %> : defines a local variable to the jsp (reinitialized at each invocation)

JSP : handling exceptions

Syntactical errors

- ✓ In HTMLcode
- ✓ In JSP directives (ex. : missing directive %>)
- ✓ In Java code (ex : missing a ";")

Java exceptions (ex. : NullPointerException)

In all cases, error displayed in client's browser

- ✓ One can realize a customized error page for the purpose of the application
- ✓ Use of the directives :
  - <%@ page errorPage="..." %> : URL of the exception handler
  - <%@ page isErrorPage="..." %> : true if the page is an exception handler

JSP : handling exceptions

Example

```
<html><body>
<h1>Test d'une erreur</h1>
<% int rand = (int)(Math.random() *2); %>
<h1>Resultat: <%= 12/rand %></h1>
</html></body>
```

if rand = 0,
Default page error

if rand ≠ 0,
Default page error

JSP : handling exceptions

Example of exception handling

```
<html><body>
<h1>Test d'une erreur</h1>
<%@ page
      errorPage="err.jsp"
%>
<% int rand =
  (int)(Math.random() *2);%>
<h1>Resultat:
    <%= 12/rand %></h1>
</html></body>
```

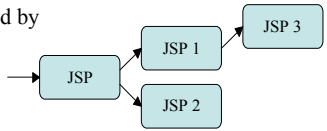
if rand = 0,
delegation of the request to err.jsp
Extracting the exception via the predefined object "exception"

```
<html><body>
<%@ page isErrorPage="true"%>
<h1>Division par zero</h1>
<h1>Exception:
  <%= exception.getMessage() %>
</h1> </html></body>
```

Chaining the JSPs

Aggregation of the results produced by several JSPs

- ✓ Better modularity
- ✓ Better reusability



Directive <jsp:include page="…"/>

URI
(local to the container)

main.jsp

```
<html> <body>
<h1>JSP principale</h1>

<jsp:include
page="inc.jsp" />
</body> </html>
```

inc.jsp

```
<b>JSP incluse</b>

<p>
<%= (int) (Math.random()*5) %>
</p>
```

File included
Do not use <HTML> <BODY>

Chaining the JSPs

JSP Inclusion

2 kinds of inclusions

- ✓ <jsp:include page="…"/> dynamic inclusion
  (delegation of servlets: **two servlets**)
- ✓ <%@ include file="..." %> static inclusion
  (inclusion at the source level : **only one servlet**)

statique Inclusion

```
<HTML> <BODY>
<H1>JSP déléguée</H1>
<P>
<%= (int) (Math.random()*5) %>
</P>
</BODY> </HTML>
```


Chaining the JSPs

Delegation of JSPs

A JSP can delegate the execution of a request to another JSP

Directives <jsp:forward page="…"/>

main.jsp

```
<html> <body>
<h1>Code ignoré</h1>

<jsp:forward
page="frow.jsp" />
```

forw.jsp

```
<html><body>
<b>JSP déléguée</b>
…
</body></html>
```

delegated file
with<HTML> <BODY>

Chaining the JSPs

Delegation of JSPs

The original JSP is completely ignored

Forward

```
<html><body>
<b>JSP déléguée</b>
…
</body></html>
```

sent
To the
Client

Chaining the JSPs

Delegation and inclusion of JSPs

Transferring parameters to included and delegated JSPs

- ✓ Use of pairs of (name, value)
- ✓ Directive `<jsp:param name="..." value="..." />`

  Extracting the parameters : **`request.getParameter("name")`**

```
<html> <body>
<h1>JSP principale</h1>
<jsp:include page="inc.jsp">
   <jsp:param name="nom" value="Bill" />
</jsp:include>
</body> </html>
```

```
Hello, <%= request.getParameter("nom") %>
```


JSP: more

Importing packages :

```
<%@ page import="java.util.Iterator, java.util.HashMap" %>
```

Concurrency

(i.e. servlets must `implements SingleThreadModel`)

```
<@ page isThreadSafe="false" %>
```

Initialization & destruction of a jsp

Redefine methods `jspInit()` et `jspDestroy()`

Servlet/JSP: Comparison

JSP : compiled into a servlet
Servlet : possibility to distinguish between HTTP requests (doGet, doPost,..)

JSP : lot of HTML, few of Java
Servlet : lot of Java, few of HTML

session, chaining, redirection : yes for both cases : API vs directives

servlet : pure Java : easily editable within an IDE
JSP : HTML editor

servlet compilation before deployment / JSP after
JSP need to be re deployed in case of an error

Deployment & execution

Packaging

Web Component : entity corresponding to a URL

- ✓ a servlet
- ✓ a jsp file

Web Application : deployment unit

- ✓ A set of Web components

deployment file (IMPORTANT)

- ✓ Description of the different components of the Web application

  Standardized, within a **web.xml**

Packaging

Description of Web components (web.xml, standard)

```
<web-app>
   <servlet>
        <servlet-name>Hello</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
   </servlet>
   <servlet-mapping>
        <servlet-name>Hello</servlet-name>
        <url-pattern>/hello</url-pattern>
   </servlet-mapping>
</web-app>
```

Servlet name

Path in your packages to the HelloServlet.class

http://server:port/???/**hello**

Localisation de l'application Web dans le conteneur

Structure of the War

Packaging

Packaging the web application Web : a War file

(Web Archive, standardized)

/ : (at the root) put Ressources used by the application

/index.html corresponds to http://…/test/index.html

/hello.jsp corresponds to http://…/test/hello.jsp

/WEB-INF/ : descriptor of the Web application

/WEB-INF/web.xml

/WEB-INF/classes/ : your classes here (servlets, Java Beans, etc.)

/WEB-INF/classes/HelloServlet.class corresponds to http://…/test/hello

Look to the previous slide for servlet name mappings

Deployment

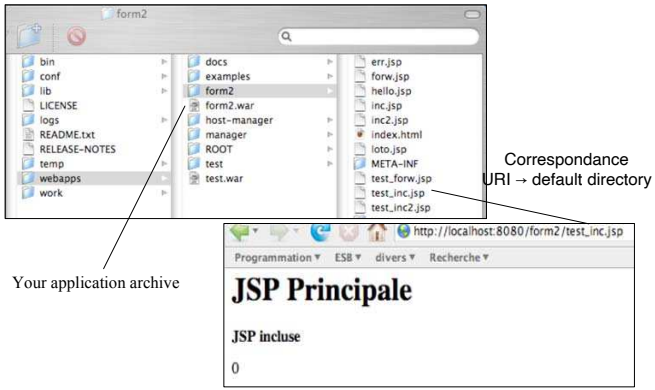
Tomcat Installation

- ✓ Download the archive from the web site
- ✓ Unzip the archive
- ✓ It works!

Content of the archive

- ✓ bin Scripts for starting/stopping the server
- ✓ conf Configuration files (server.xml)
- ✓ doc Documentation lib Libraries used by Tomcat
- ✓ logs Directory for logs
- ✓ src Tomcat sources
- ✓ webapps IMPORTANT: directory where to put your web-archives (war)

Deployment



HTML Form

Tag **<form>**

- ✓ Attribute method :
  - get : method HTTP GET, parameters via the URI
  - post : method HTTP POST, the HTML with parameters sent to the server
- ✓ Attribute action :
  - URI of the Web component that receives the form
- ✓ tag <input> : definition of a parameter
  - Attribute type
    - Defines the type of the parameter (text, password, reset, submit…)
  - Attribute name
    - Defines the names of the parameter

HTML Form

Example of an HTML Form

```
<html> <body>
<h3>Hello, What's your name?</h3>
<form method="post" action="url-servlet">
  Name: <input type="text" name="username" size="25">
  Password: <input type="password" name="password" size="25">
  <p></p>
  <input type="submit"
         value="Submit">
  <input type="reset"
         value="Reset">
</form>
</body> </html>
```

HTML Form

Récupération des données d'un formulaire dans une servlet

```
public class FormulaireServlet extends HttpServlet {
    public void doPost( HttpServletRequest request, HttpServletResponse response )
            throws ServletException, IOException {
        response.setContentType( "text/html" );
        PrintWriter out = response.getWriter();
        out.println( "<html><body>" );
        String name = request.getParameter("username");
        String password = request.getParameter("password");
        out.println( "<h2>Hello, " + name + ", your password is \""
                                        + password + "\"</body></html>");
    }}
```


HTML Form

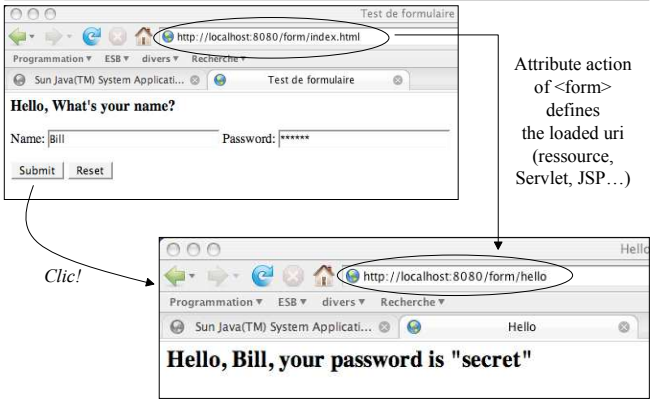
Récupération des données d'un formulaire dans une JSP

```
<html><body>

<h2>Hello,
      <%= request.getParameter("username") %>,
    your password is "
      <%= request.getParameter("password") %>
    "
</h2>

</body></html>
```


HTML Form



GET Vs. POST

1. GET requests should not modify the state of your application
   1. No side effects! Exp. Data from the form is used just to extract data
   2. If your request aims to update the data/state of your application, use POST
2. GET: Form's data is displayed in the URL of the request. Using POST, the data is hidden
3. GET: form's data size is limited in the URL (255 characters). No limit for the POST (e.g. uploading a file)

Conclusion

Servlet & Java Server Pages :

Execution behavior on the server side

Summary of functionalities

- ✓ JAVA embedded within HTML or HTML embedded within JAVA
- ✓ Portability, easy to write (Java)
- ✓ Notion of session over HTTP
- ✓ Persistency of data between two calls
- ✓ JSP loaded and instanciated only once
- ✓ JSP executed within a thread
- Be carful of concurrent accesses!