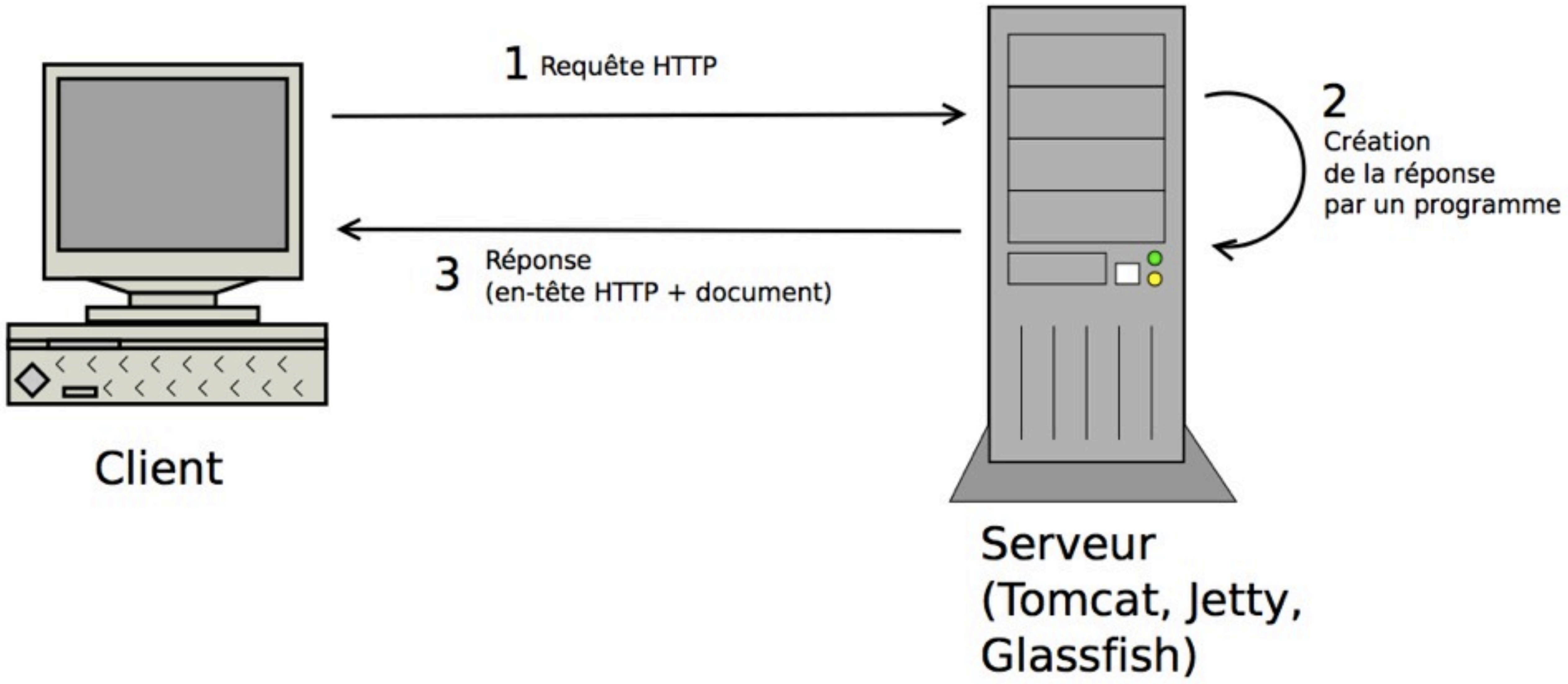
JSP and Servlets

Serge Rosmorduc
serge.rosmorduc@cnam.fr

Content

- WEB architecture : client, server, application server
- Servlets ;
- limitations of servlets ;
- JSP, ; expression language ; beans request ; combination of jsp and servlets, forwarding ;
- How to deal with a form

Web Architecture


1 Requête HTTP
2 Création de la réponse par un programme
3 Réponse (en-tête HTTP + document)
Client
Serveur (Tomcat, Jetty, Glassfish)

Principle of dynamic web sites

- The Web application server receives the request from the client
- it runs a piece of software which creates the page for the client
- it returns the page content to the client

HTTP Protocol Example

```
$ telnet deptinfo.cnam.fr 80
Trying 163.173.228.28...
Connected to deptinfo.cnam.fr.
Escape character is '^]'.
GET /~rosmorse/aisl-chine/ HTTP/1.0

HTTP/1.1 200 OK
Content-Length: 1194
Content-Type: text/html

<!doctype html>
<html>
<head>
</head>
<body>
  <ul>
  <li> <a href="patterns.pdf">Slides on design patterns</a></li>
  <li> <a href="servlets">JSP/Servlets et architecture</a></li>
  </ul>
</body>
</html>
```

Request (ends with empty line)

Header

Empty line

Content

Response

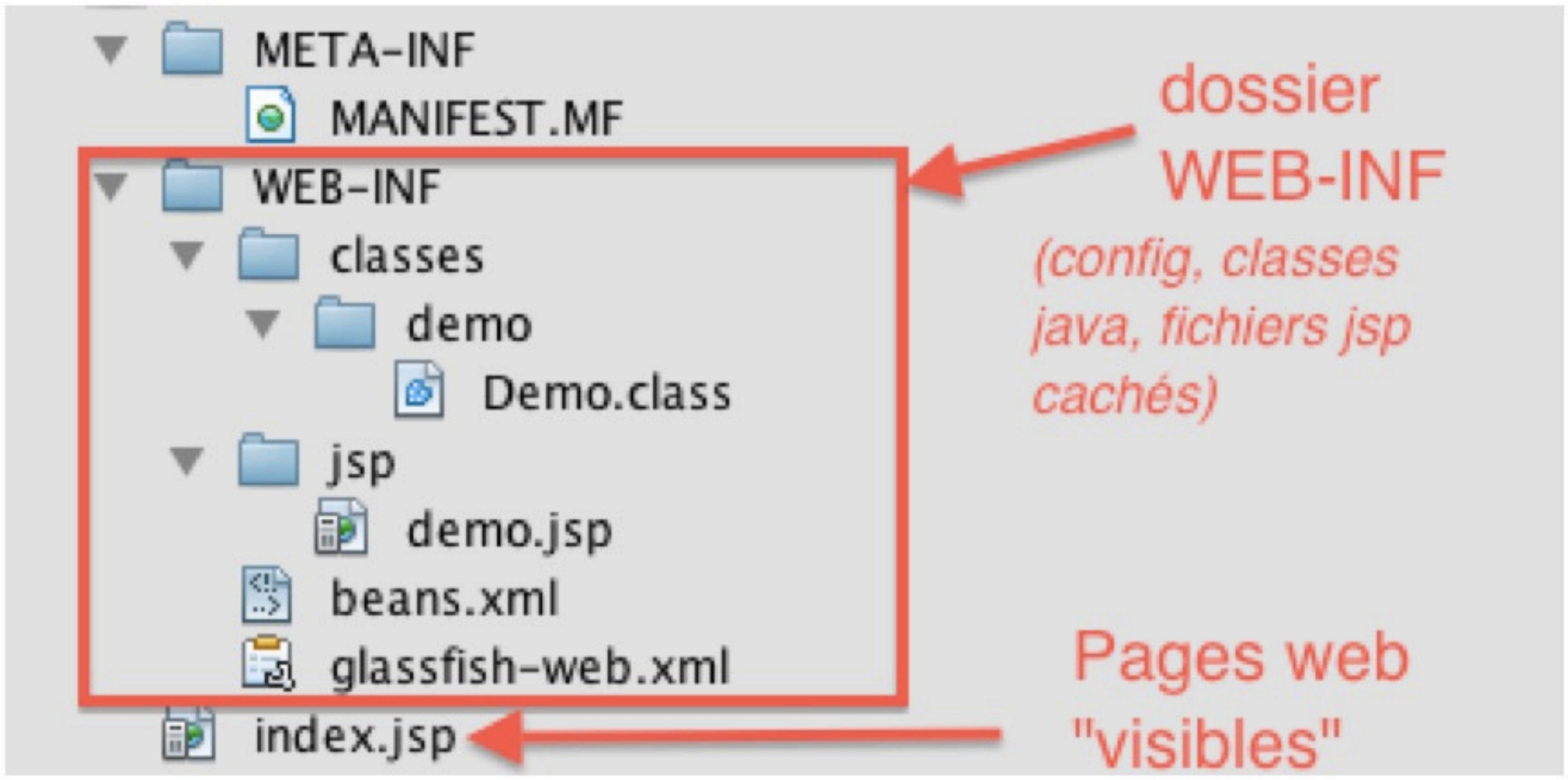
Content-type

POST and GET mode

J2EE stack

- Lots of possible variants and modules
- Web part : Servlets/JSP ; Wicket, JSF ;
- + service directory (JNDI) ;
- + transaction management (JTA) ; persistance (JTA) ; + load balancing...
- Simple servers : Tomcat, Jetty...
- full-stack servers : JBoss, Glassfish...
- EJB3 vs. Spring : converging systems

Files layout in a Web application



Files layout in a web app

- Usually compressed (zipped) in a .war file
- the root of the application folder contains:
  - directly accessible ressources (HTML files, pictures, some jsps)
  - a WEB-INF folder which contains
    - the web.xml configuration file
    - a «classes» folder for compiled java code
    - a «lib» folder for java «.jar» libraries
  - WEB-INF is **not** visible by web clients. It’s suitable for storing hidden ressources.

Demo with Netbeans

GET Method

- User query data sent in the URL

  `http://www.google.com/search`**`?q=cnam&sourceid=chrome`**

  - protocole and server http://www.google.com
  - page : search
  - variable(s) :
    - **q**, value « cnam»
    - **sourceid**, value « chrome »
- A get request is very easy to reproduce on a web browser

POST method

- Request data sent in the body of the HTTP request
- Not visible

Servlet

- Java class extending HTTP servlet
- a servlet answers for a particular URL
- it has two methods, doGet and doPost for resp. GET and POST requests
- those methods will answer for GET and POST on this particular URL.

A Hello World Servlet

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                          HttpServletResponse response)
              throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        out.println("Bonjour " +
                          request.getParameter("nom"));
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

A Hello World Servlet

Annotation

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                              HttpServletResponse response)
              throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        out.println("Bonjour " +
                         request.getParameter("nom"));
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

A He

Sets the URL

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
              throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        out.println("Bonjour " +
                         request.getParameter("nom"));
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

A Hello World Servlet

```java
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
              throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        out.println("Bonjour " +
                         request.getParameter("nom"));
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

for GET requests

A Hello World Servlet

```java
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
              throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        out.println("Bonjour " +
                         request.getParameter("nom"));
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

We will send HTML with UTF-8 encoding

A Hello World Servlet

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
              throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        out.println("Bonjour " +
                         request.getParameter("nom"));
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

We write the answer

A Hello World Servlet

```
@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
              throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<body>");
        out.println("Bonjour " +
                         request.getParameter("nom"));
        out.println("</body>");
        out.println("</html>");
        out.close();

    }
}
```

Use of a parameter from the URL

Servlet

- Annotation : @WebServlet(name = "HelloServlet", urlPatterns = {"/hello"}) : says the servlet is associated with URL «/hello»
- request object: represents the request; gives access to form parameters
- useful methods for request :
  - String getParameter(String): returns the value of one parameter
  - String [] getParameterValues(String) : returns the values of a multiple values parameters (lists, checkboxes...)
- response object: represents the request result; allows to write the result, send redirections...

Limits of servlets

- HTML generation is cumbersome;
- Mixing HTML and Java is bad
  - no separation between display and processing
  - difficult to read anyway

JSP

- HTML with bits of Java in them
- Specific markup:
  - <% ... %> java *statements*;
  - <%= ... %> java *expression*.
- Specific objects :
  - out : allows writing (JspWriter)
  - request
  - response
  - session
  - application

JSP, example

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>JSP</title>
</head>
<body>
  <p>Hello <%= request.getParameter("nom") %></p>

<p>
<%
  for (int i= 0; i < 100; i++) {
    out.print("-");
  }
%>
</body>
</html>
```

Expression

java statements

Proper use of those

- Servlet
  - Java code (good)
  - receives the request
  - delegates the processing to business classes (we want to minimize the web aspect)
  - delegates display to a JSP
- JSP
  - mix of HTML and Java
  - **computes nothing at all**
  - **only displays data sent by the servlet**
- Advantage : separation of concerns, separate development, can be tested separately
- In practice...

Forwarding between servlet and JSP

- Example : display of a person name, capitalized. We use a RequestDispatcher

```
@WebServlet(name = "Hello2", urlPatterns = {"/hello2"})
public class Hello2 extends HttpServlet {
@Override
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
              throws ServletException, IOException {
   String nom = request.getParameter("nom");
   nom = NomHelper.capitaliser(nom);
   request.setAttribute("nomCapitalise", nom);
   RequestDispatcher r =
      request.getRequestDispatcher("/WEB-INF/jsp/hello2.jsp");
   r.forward(request, response);
   }
}
```

Forwarding between servlet and JSP

```
// get the parameters values...
String nom = request.getParameter("nom");
// delegate processing to business class
nom = NomHelper.capitaliser(nom);
// store the result in attributes (request beans)
request.setAttribute("nomCapitalise", nom);
// create and forward to a dispatcher
RequestDispatcher r =
   request.getRequestDispatcher("/WEB-INF/jsp/hello2.jsp");
r.forward(request, response);
```

Comments

- Capitalizing is not that simple : null, empty string...
- we get it out of the servlet, which should be as little smart as possible: creation of Helper class NomHelper
- this class can be tested on its own (without JSP/Servlet environment)
- computed data is temporarily stored as beans with request.setAttribute(name, value).
    - **name** is a String used to identify the attribute
    - **value** is any **object** (not necessarily a String)

RequestDispatcher

- Asks a JSP to manage the end of the request's processing
- Use:

```
String path= "/WEB-INF/jsp/hello2.jsp";

RequestDispatcher r=
request.getRequestDispatcher(path);

r.forward(request, response);
```

- path: path to the jsp (usually) for display. Often in WEB-INF

A bit of expression language

- In JSP, you can write:

  <%= request.getAttribute("name") %>

- it's long and not readable
- plus, we would like to remove java code from JSP
- special micro-language: you can write

  ${name}

  instead

Expression language and properties

- When a bean is an object with accessors (getters)
- like a «person» bean with methods getName() and getSurname()
- then you can access the properties with:
  - ${person.name}
  - ${person.surname}
- it calls person.getName() and person.getSurname()

hello2.jsp

```
<!DOCTYPE html>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>JSP Page</title>
</head>
<body>
 hello ${name}
</body>
</html>
```

Summary

- Servlets receives a request, extracts arguments and decides what to do
- it delegates actual processing to Helper/ Business classes, which don't know anything about the web
- it stores the values to display in beans, and handle the end of the processing to a JSP for display

How to deal with forms

Example

- Software in which:
  - the user will type the name and the surname of a person in a form
  - the fields will be checked (non empty)
  - will compute a greeting message
  - will display the message with a JSP
  - In case of errors, the form will be redisplayed with error messages and *existing data*
- The **same servlet** deals with form display and form processing
- **two** jsp : one for form display, one for result display
- often, we use doGet() for the initial display, and doPost() for subsequent processing

```
@WebServlet(name = "Saluer", urlPatterns = {"/saluer"})
public class Saluer extends HttpServlet {

   protected void doGet(HttpServletRequest req, HttpServletResponse
   resp) throws ServletException, IOException {
      req.getRequestDispatcher(
            "/WEB-INF/jsp saluerForm.jsp" ).forward(req, resp);
   }

   protected void doPost(HttpServletRequest req, HttpServletResponse
   resp) throws ServletException, IOException {
      String jsp;
      String nom= req.getParameter("nom");
      String prenom= req.getParameter("prenom");
      if (nom == null || prenom == null || "".equals(nom) ||
      "".equals(prenom)) {
         req.setAttribute("message", "les champs doivent être remplis");
            jsp= "/WEB-INF/jsp/saluerForm.jsp";
      } else {
         req.setAttribute("nom", nom.toUpperCase());
         req.setAttribute("prenom", prenom);
         jsp= "/WEB-INF/jsp/saluer.jsp";
      }
      req.getRequestDispatcher(jsp).forward(req, resp);
   }
}
```

JSP

```
<!DOCTYPE html>
<html>
    <head>
        <title>Saluer</title>
    </head>
    <body>
        <p style="color:red">${message}</p>
        <form method="POST">
            <p>nom : <input type="text" name="nom"
                            value="${param.nom}"/></p>
            <p>prenom : <input type="text" name="prenom"
                            value="${param.prenom}"/></p>
            <p><input type="submit"/></p>
        </form>
    </body>
</html>
```

- Note that if ${message} is not set, it's not a problem.
- ${param.nom} : value of *parameter* «nom» (not attribute).

Improvement: error handling with Maps

```
protected void doPost(....) throws .... {
   String jsp;
   HashMap<String,String> erreurs=
               new HashMap<String,String>();
   String nom= req.getParameter("nom");
   String prenom= req.getParameter("prenom");

   if (nom == null || "".equals(nom)) {
       erreurs.put("nom", "ce champ doit être rempli");
   }
   if (prenom == null || "".equals(prenom)) {
       erreurs.put("prenom", "ce champ doit être rempli");
   }
   if (! erreurs.isEmpty()) {
       req.setAttribute("erreurs", erreurs);
       jsp= "/WEB-INF/jsp/saluerForm1.jsp";
   } else {
       req.setAttribute("nom", nom.toUpperCase());
       req.setAttribute("prenom", prenom);
       jsp= "/WEB-INF/jsp/saluer.jsp";
   }
   req.getRequestDispatcher(jsp).forward(req, resp);
}
```

New JSP form

```
<!DOCTYPE html>
<html>
    <head>
        <title>Saluer</title>
        <style>
            .erreur {color: red;}
        </style>
    </head>
    <body>
        <form method="POST">
            <p>nom : <input type="text" name="nom"
                            value="${param.nom}"/>
                <span class="erreur"> ${erreurs.nom}</span>
            </p>
            <p>prenom : <input type="text" name="prenom"
                               value="${param.prenom}"/>
                <span class="erreur"> ${erreurs.prenom}</span>
            </p>
            <p><input type="submit"/></p>
        </form>
    </body>
</html>
```