JSP

Basics

Last available official tutorial:
http://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html (2010)

Why JSP?

It is today a deprecated technology, but it is the basis for the current technology (JSF)

So we better understand how it works…

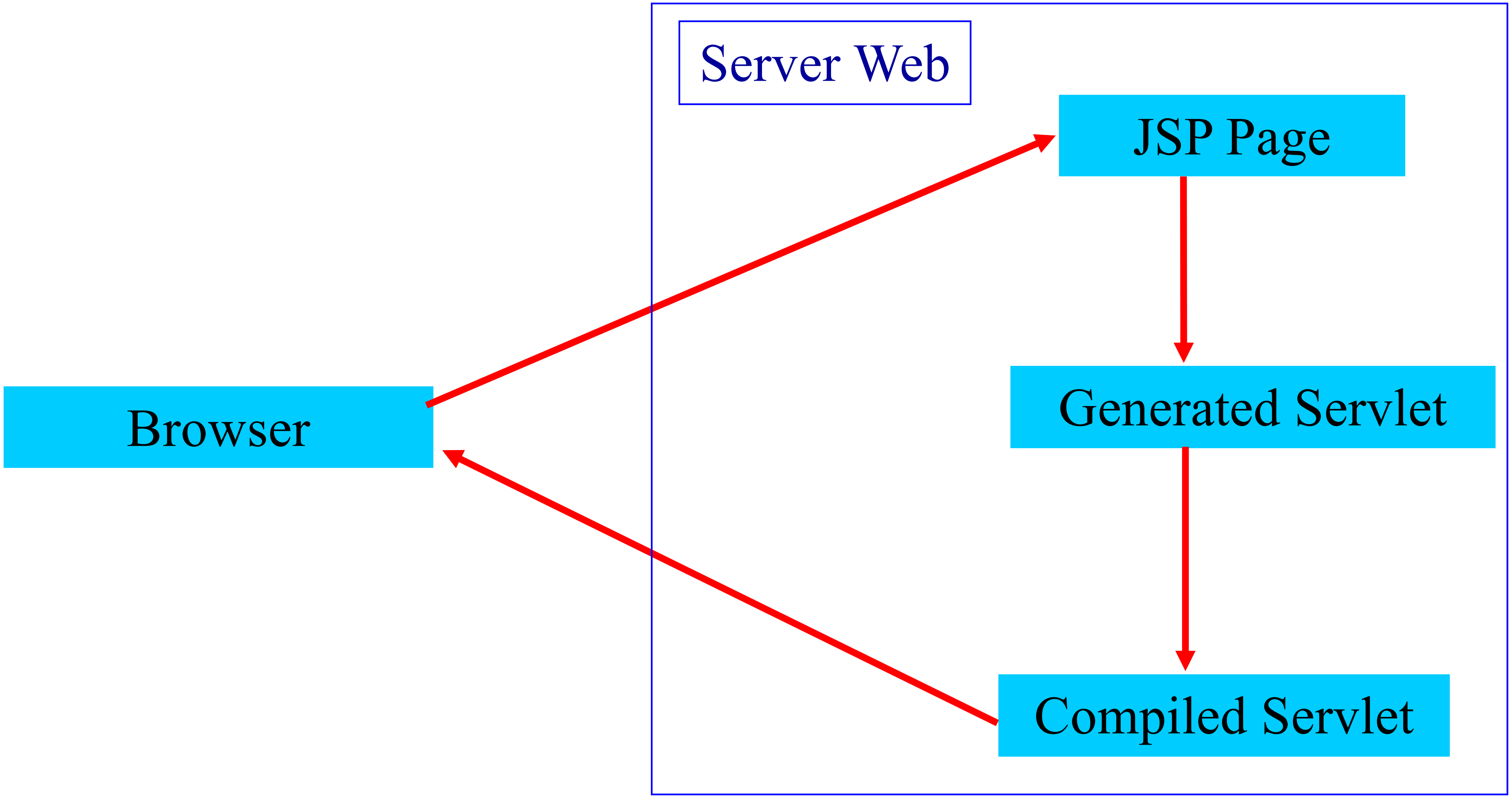
A taste of servlet programming-2

```
import java.util.Calendar;
public class SimpleServlet extends HttpServlet {
   public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
                        throws ServletException, IOException {
       PrintWriter out=response.getWriter();
       response.setContentType("text/html");
       out.println("<HTML><BODY>");
       out.println(Calendar.get(Calendar.HOUR_OF_DAY));
       out.println("</BODY></HTML>");
       out.close();
   }
}
```

Simple.jsp

```
<%@ page import=java.util.*  %>
<html>
 <body>
    <% out.println(Calendar.get(Calendar.HOUR_OF_DAY)); %>
  </body>
</html>
```

JSP Lifecycle



JSP nuts and bolts

Syntactic elements:

<%@ directives %>

<%! declarations %>

<% scriptlets %>

<%= expressions %>

<jsp:actions/>

<%-- Comment --%>

Implicit Objects:

- request
- response
- pageContext
- session
- application
- out
- config
- page

JSP nuts and bolts

Syntactic elements:

<%@ directives %> → Interaction with the CONTAINER

<%! declarations %> → In the initialization of the JSP

<% scriptlets %> → In the service method
<%= expressions %> → (Syntactic sugar) same as scriptlets

<jsp:actions/>

Scriptlets

A scriptlet is a block of Java code executed during the request-processing time.

In Tomcat all the scriptlets gets put into the service() method of the servlet. They are therefore processed for every request that the servlet receives.

Scriptlet

Examples :

```
<% z=z+1; %>
```

```
<%
       // Get the Employee's Name from the request
       out.println("<b>Employee: </b>" +
       request.getParameter("employee"));
       // Get the Employee's Title from the request
       out.println("<br><b>Title: </b>" +
       request.getParameter("title"));
  %>
```

Declarations

A declaration is a block of Java code used to:

define class-wide variables and methods in the generated servlet.

They are initialized when the JSP page is initialized.

<%! DECLARATION %>

Examples:

<%! String nome="pippo"; %>

<%! public String getName() {return nome;} %>

Directives

A directive is used as a message mechanism to:

pass information from the JSP code to the container

Main directives:

page

include (for including other STATIC resources at compilation time)

taglib (for including custom tag libraries)

Directives

```
<%@ DIRECTIVE{attributo=valore} %>
```

main attributes:

```
<%@ page language=java  session=true %>
<%@ page import=java.awt.*,java.util.*  %>
<%@ page isThreadSafe=false  %>
<%@ page errorPage=URL %>
<%@ page isErrorPage=true %>
```

Standard actions

Standard action are tags that affect the runtime behavior of the JSP and the response sent back to the client.

<jsp:include page="URL" />

For including STATIC or DYNAMIC resources at request time

<jsp:forward page="URL" />

Java Bean

A bean is a Java class that:

- Provides a public zero-arguments constructor
- Implements java.io.Serializable
- Follows JavaBeans design patterns
  - Has Set/get methods for properties
- Is thread safe/security conscious

```
public class SimpleBean implements Serializable {
      private int counter;
      SimpleBean() {counter=0;}
      int getCounter() {return counter;}
      void setCounter(int c) {counter=c;}
}
```

See http://docs.oracle.com/javase/tutorial/javabeans/

Standard actions involving beans

```
<jsp:useBean id=“name” class=“fully_qualified_pathname”
scope=“{page|request|session|application}” />

<jsp:setProperty name=“nome” property=“value” />

<jsp:getProperty name=“nome” property=“value” />
```

Predefined Objects

out	Writer
request	HttpServletRequest
response	HttpServletResponse
session	HttpSession
page	this nel Servlet
application	servlet.getServletContext area condivisa tra i servlet
config	ServletConfig
exception	solo nella errorPage
pageContext	sorgente degli oggetti, raramente usato

request

```
<%@ page errorPage="errorpage.jsp" %>
<html>
  <head>
    <title>UseRequest</title>
  </head>
  <body>
    <%
        // Get the User's Name from the request
        out.println("<b>Hello: " + request.getParameter("user") + "</b>");
    %>
  </body>
</html>
```

session

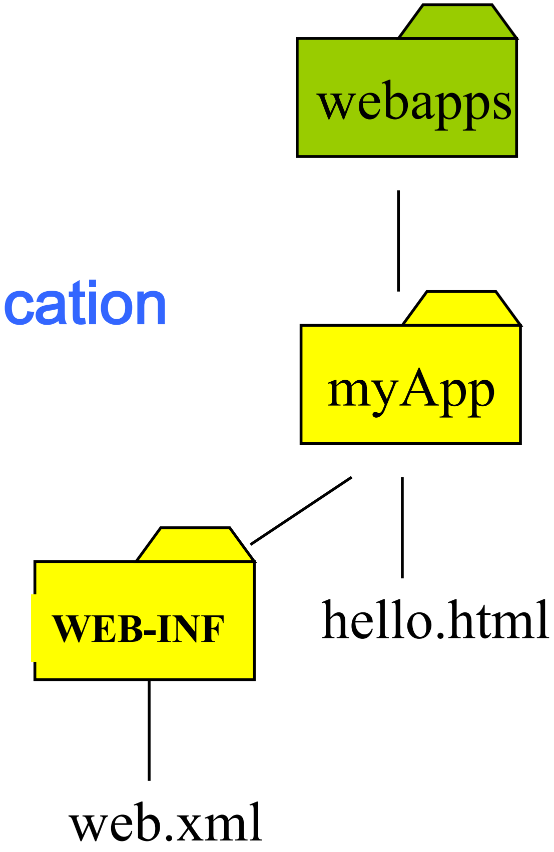
```
<%@ page errorPage="errorpage.jsp" %>
<html> <head> <title>UseSession</title> </head> <body>
  <%
   // Try and get the current count from the session
   Integer count = (Integer)session.getAttribute("COUNT");
   // If COUNT is not found, create it and add it to the session
   if ( count == null ) {
     count = new Integer(1);
     session.setAttribute("COUNT", count);
   } else {
     count = new Integer(count.intValue() + 1);
     session.setAttribute("COUNT", count);
   }
   // Get the User's Name from the request
   out.println("<b>Hello you have visited this site: " + count + " times. </b>");
  %>
 </body> </html>
```

WebApps
(Tomcat configuration)

Static pages

A web.xml file **MUST** be provided:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN“
   "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
</web-app>
```



JSP pages

To let Tomcat serve JSP pages, we follow the same procedure that we described for static pages.

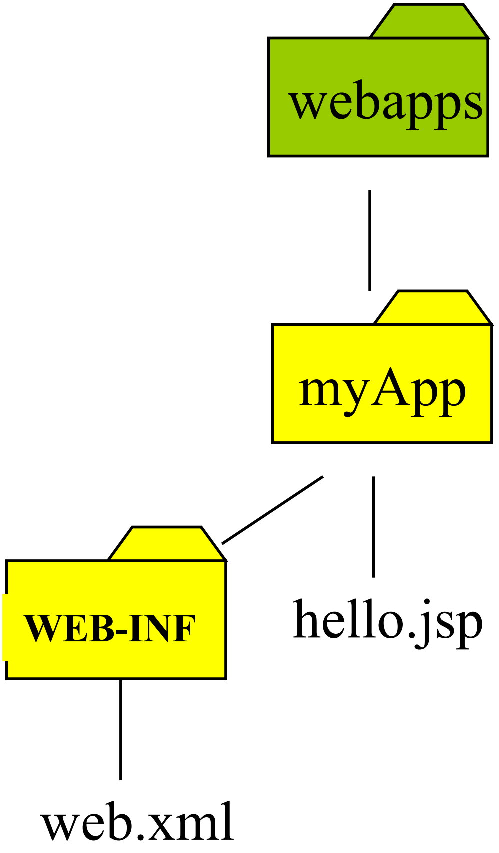
In the myApp folder we can depost the JSP files.

On our Tomcat server, the URL for the hello.jsp file becomes:

http://*machine/port*/myApp/hello.jsp

The WEB-INF directory is still empty.

The same web.xml file as in the static case must be provided.


webapps
myApp
WEB-INF
hello.jsp
web.xml