JSP

Java Server Pages

Java Server Pages (JSP) are a variant of servlets

- When creating a servlet, we write Java-code, and use Java-statements to write HTML into a stream
- When using JSP we write HTML, and add embedded Java-statements

At runtime a JSP is automatically translated into a servlet and the resulting servlet is then executed

- As a result, the HTML part is transformed into output statements like `out.println("...");`

Servlets are preferred when a complex logic has to implemented (several if-statements, loops)

JSPs are used when the presentation-part is dominating

Comparison

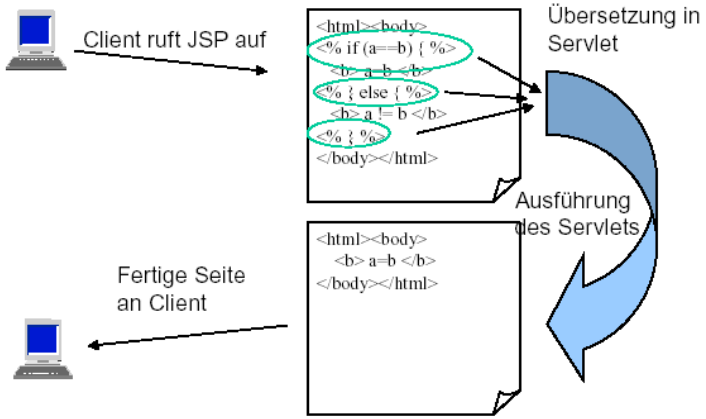
Servlet:

```
public class DatumServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
       HttpServletResponse response) {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><body>");
    out.println("It is: "); out.println(( new
              java.util.Date()).toString());
    out.println("</body></html>");
  }
}
```

JSP:

```
<html>< body>
It is:
<% out.println(( new java.util.Date()).toString()); %>
</ body></ html>
```


Steps when calling a JSP



Creating JSPs

Usually we start with creating the static part of the HTML-page

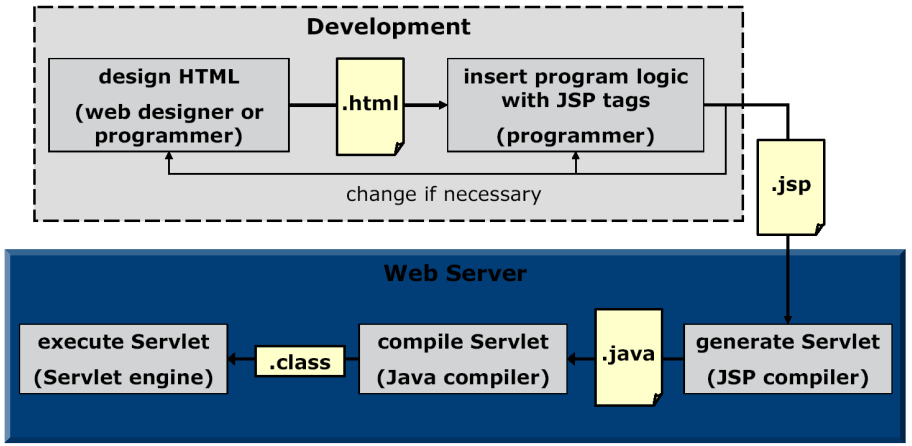
Afterwards we add the JSP-tags for the dynamic part

Available are for example:

- insertion of the value of a Java-expression
- Scriptlets (Java-statements)

In the final result we get a HTML-document in which the JSP-tags are replaced by the result of their execution

Complete Process



Implicitly Declared Objects

Remind: a JSP is executed as a servlet

- thus, we have the possibility to interact with the environment like we do for a servlet
- for this purpose several implicitly declared objects are available that are initialized by the servlet-engine

Example:

- request (subclass of javax.servlet.ServletRequest)
- response (subclass of javax.servlet.ServletResponse)
- out (javax.servlet.jsp.JspWriter)
- session (javax.servlet.HttpSession)
  - by default we always have a session

Page-Directives

Page-Directives set attributes, that are relevant for the complete JSP-page

Original JSP-Notation:

- `<%@ page _directive_= "_value_" %>`

XML-Notation:

- `<jsp:directive.page _directive_= "_value_" />`

Example

- `<jsp:directive.page import="java.utils.* , mypackage.class" />`

The most frequently used page-directive is:

- import="package.class"
- this directive imports classes or packages
- the packeges java.lang.*, javax.servlet.*, javax.servlet.jsp.* and javax.servlet.http.* are automatically imported

Other types of directives

- include (include other files)
- taglib (make available libraries that contain additional tags)

Declarations

Declarations allow to define attributes and methods

These attributes and methods belong to the servlet-object or to the servlet-class

Remind: different calls to the servlet use the same servlet-object

Syntax:

- `<%! Declaration %>` or
- `<jsp:declaration> Declaration </jsp:declaration>`

Example:

```
<jsp:declaration>
public int Sum( int a, int b) { return a + b; }
</jsp:declaration>
```


Expressions

Expressions have to produce a value that can be transformed into a string

Expressions do not have a semicolon at the end!

Expressions become a part of the method, that is executed when the servlet is invoked (like doGet() or doPost())

Syntax:

`<%= Expression %>` or

`<jsp:expression>Expression</jsp:expression>`

- Example:
  - `<%= Sum( oldValue, count) %>`
  - `<jsp:expression>customer.getName()`
    `</jsp:expression>`

Scriplets

Scriplets contain statements in Java-code:

Syntax:

- `<% code fragment %>` or
- `<jsp:scriptlet> code fragment </jsp:scriptlet>`

Similar to expressions, scriplets become a part of the method, that is executed when the servlet is invoked (like doGet() or doPost())

Example:

```
<SELECT name= myselection>
<%for (int i= 0; i< 5; i++) {%>
  <OPTION VALUE= value<%= i%>> entry<%= i%>
  </ OPTION>
<%}%>
</SELECT>
```


JSP and POJOs

POJO = Plain Old Java Object

Simplified variant of Java Beans

POJOs contain a constructor without parameters and properties

Properties have accessor methods (getter + setter)

Names of properties are related to names of methods:

- getter: getProperty
- setter: setProperty

A property usually corresponds to a member variable

- it is not necessary that the member variable has the same name as the property!

Example POJO

```
package user;
public class UserData {
    String name; String email; int age;

    public void setUsername( String value )  { name = value; }
    public String getUsername() { return name; }
    public void setEmail( String value )  { email = value; }
    public String getEmail() { return email; }
    public void setAge( int value )  { age = value; }
    public int getAge() { return age; }
}
```


Using POJOs in JSPs

Create an object

- <jsp:useBean id="user" class="user.UserData"/>
- the default is that the object is only available on the same page
- different calls will use the same object

Set a property

- <jsp:setProperty name="user" property="age" value="49" /> or
- <% user.setAge(49); %>

Read a property

- <jsp:getProperty name="user" property="age" /> or
- <%= user.getAge() %>

Scopes for Beans

The lifetime of a bean need not be restricted to a single page

The element jsp:useBean has an attribute `scope` that specifies the context in which the bean is available

Different scopes:

- page (default): only for the page on which it is created
- request: for the current request
  - This makes a difference to the page-scope if the processing of the request is delegated to a different JSP/servlet
- session: for the current session
- application: for the complete web application

Using Beans across different Pages

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
What's your name? <INPUT TYPE=TEXT NAME=username><BR>
What's your e-mail address? <INPUT TYPE=TEXT NAME=email SIZE=20><BR>
What's your age? <INPUT TYPE=TEXT NAME=age SIZE=4>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

SaveName.jsp

```
<jsp:useBean id="user" class="user.UserData"
scope="session"/>

<jsp:setProperty name="user" property="*"/>
<HTML>
<BODY>
<A HREF="NextPage.jsp">Continue</A>
</BODY>
</HTML>
```


NextPage.jsp

```
<jsp:useBean id="user" class="user.UserData"
scope="session"/>
<HTML>
<BODY>
You entered<BR>
Name: <%= user.getUsername() %><BR>
Email: <%= user.getEmail() %><BR>
Age: <jsp:getProperty name="user" property="age" /> <BR>
</BODY>
</HTML>
```


Recommendations

Java-code within a JSP should be minimized

Result

- simpler integration of programming and Web-design
- better maintenance

Therefore

- program logic should be realized as far as possible within other classes
- additional JSP-techniques can be used
  - Expression language
  - Tag libraries