

JSPs Part 2

JSP 2.0

Goals

Previous versions of JSP limited developers' ability to separate presentation, control, and the data model. The best solution had been the MVC architecture for designing software. MVC splits up Java Servlet/JSP web applications into three parts:

- 1) The Model – Data regarding the state of the world, stored in Java Beans
- 2) The View – The user interface that is presented to the user is done through the JSP.
- 3) The Controller – A servlet handles the request, and updates the model (storing data in the Java Bean), and forwards the request to the JSP that will render the user interface.

Here are some links to **optional** readings regarding JSP 2.0:

<http://www.onjava.com/pub/a/onjava/2003/11/05/jsp.html>

<http://www.onjava.com/pub/a/onjava/2003/12/03/JSP2part2.html>

<http://www.onjava.com/pub/a/onjava/2004/04/21/JSP2part3.html>

<http://www.onjava.com/pub/a/onjava/2004/05/12/jsp2part4.html>

Expression Language

The Expression Language is new to JSP 2.0. It is inspired by JavaScript (which we will learn about soon).

Use it where you would normally use a Java JSP Expression `<%= ... %>`

It's more forgiving about null pointers, so that a small error doesn't "crash" the JSP page.

Converts many data types automatically, so your company's web designers don't have to worry about writing real Java code.

Don't need to worry about absent request parameters, or type conversions.

Expressions are of the form `${expression}`. Easy huh?

Operators (Here are some samples, but mostly they are familiar to you)

`.` -- Access a bean property or Map entry (one of the implicit variables)

`[]` -- Access an array or List element

`%` or `mod` -- Modulo (remainder operator)

`/` or `div` -- Division

`>=` or `ge` -- Test for greater than or equal to

`func(args)` -- function call

`empty` -- tests for empty variable or null, empty String, or an empty array

See the rest of the operators at <http://www.onjava.com/pub/a/onjava/2003/11/05/jsp.html>

Arithmetic Expressions

EL Expression	Result
<code>\${1}</code>	1
<code>\${1 + 2}</code>	3
<code>\${1.2 + 2.3}</code>	3.5
<code>\${1.2E4 + 1.4}</code>	12001.4
<code>\${-4 - 2}</code>	-6
<code>\${21 * 2}</code>	42
<code>\${3/4}</code>	0.75
<code>\${3 div 4}</code>	0.75
<code>\${3/0}</code>	Infinity
<code>\${10%4}</code>	2
<code>\${10 mod 4}</code>	2
<code>\${(1==2) ? 3 : 4}</code>	4

Comparisons

Numeric

EL Expression	Result
<code>\${1 < 2}</code>	true
<code>\${1 lt 2}</code>	true
<code>\${1 > (4/2)}</code>	false
<code>\${1 > (4/2)}</code>	false
<code>\${4.0 >= 3}</code>	true
<code>\${4.0 ge 3}</code>	true
<code>\${4 <= 3}</code>	false
<code>\${4 le 3}</code>	false
<code>\${100.0 == 100}</code>	true
<code>\${100.0 eq 100}</code>	true
<code>\${(10*10) != 100}</code>	false
<code>\${(10*10) ne 100}</code>	false

Alphabetic

EL Expression	Result
<code>\${'a' < 'b'}</code>	true
<code>\${'hip' > 'hit'}</code>	false
<code>\${'4' > 3}</code>	true

Function Calls

Defined by Tag Libraries, Implemented as Static Methods in Java Classes

Of the form: *taglib_name:function(args)*

Change Parameter

foo =

EL Expression	Result
<code>\${param["foo"]}</code>	JSP 2.0
<code>\${my:reverse(param["foo"])}</code>	0.2 PSJ
<code>\${my:reverse(my:reverse(param["foo"]))}</code>	JSP 2.0
<code>\${my:countVowels(param["foo"])}</code>	0

Implicit Variables (predefined for your convenience)

The Expression Language has a lot of “global” predefined variables, that make it more convenient to access things like the session object, or the request or response objects, or any of the variables stored in those objects.

- `pageContext` - the `PageContext` object
- `pageScope` - a Map that maps page-scoped attribute names to their values
- `requestScope` - a Map that maps request-scoped attribute names to their values
- `sessionScope` - a Map that maps session-scoped attribute names to their values
- `applicationScope` - a Map that maps application-scoped attribute names to their values
- `param` - a Map that maps parameter names to a single String parameter value
- `paramValues` - a Map that maps parameter names to a `String[]` of all values for that parameter
- `header` - a Map that maps header names to a single String header value
- `headerValues` - a Map that maps header names to a `String[]` of all values for that header
- `initParam` - a Map that maps context initialization parameter names to their String parameter value
- `cookie` - a Map that maps cookie names to a single `Cookie` object.

Change Parameter

foo =

EL Expression	Result
<code>\${param.foo}</code>	bar
<code>\${param["foo"]}</code>	bar
<code>\${header["host"]}</code>	localhost:8080
<code>\${header["accept"]}</code>	*/*
<code>\${header["user-agent"]}</code>	Mozilla/5.0 (Macintosh; U; PPC Mac OS X; en-us) AppleWebKit/124 (KHTML, like Gecko) Safari/125

See more information at: <http://www.onjava.com/pub/a/onjava/2003/11/05/jsp.html>

JSTL (Standard Tag Library)

Standard Tag Libraries included with Tomcat that you can use. Here’s a snippet from a JSP file:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
  <%@ page import="java.util.Vector" %>
  <h3>Iterating over a range</h3>
  <c:forEach var="item" begin="1" end="10">
    ${item}
  </c:forEach>

  <% Vector v = new Vector();
     v.add("One"); v.add("Two"); v.add("Three"); v.add("Four");
     pageContext.setAttribute("vector", v);
  %>
  <h3>Iterating over a Vector</h3>
  <c:forEach items="${vector}" var="item" >
    ${item}
  </c:forEach>
```

Custom Tag Libraries (Custom Action Elements)

Install in `/WEB-INF/lib/`

They are a way for you to extend JSP. Setting them up is a rather advanced JSP topic, so you do not have to worry about them in this class.

Notes on Tomcat Deployment

WAR Files – Like JAR files, but for distributing Web Apps

From the Tomcat Documentation:

To facilitate creation of a Web Application Archive file in the required format, it is convenient to arrange the "executable" files of your web application (that is, the files that Tomcat actually uses when executing your app) in the same organization as required by the WAR format itself. To do this, you will end up with the following contents in your application's "document root" directory:

- ***.html, *.jsp, etc.** - The HTML and JSP pages, along with other files that must be visible to the client browser (such as JavaScript, stylesheet files, and images) for your application.
- **/WEB-INF/web.xml** - The Web Application Deployment Descriptor for your application. This is an XML file describing the servlets and other components that make up your application, along with any initialization parameters and container-managed security constraints that you want the server to enforce for you. The web.xml is a file you have to edit to tell the servlet container which servlets you have installed, and how to map request paths to the file system. If you install Tomcat, you can read about web.xml at: <http://localhost:8080/tomcat-docs/appdev/web.xml.txt>
- **/WEB-INF/classes/** - Contains any Java class files (and associated resources) required for your application, including both servlet and non-servlet classes, that are not combined into JAR files. If your classes are organized into Java packages, you must reflect this in the directory hierarchy under `/WEB-INF/classes/` (e.g. a Java class named `com.mycompany.mypackage.MyServlet` would need to be stored in a file named `/WEB-INF/classes/com/mycompany/mypackage/MyServlet.class`).
- **/WEB-INF/lib/** - Contains JAR files that contain Java class files (and associated resources) required for your application, such as third party class libraries or JDBC drivers.