The Problem

- Writing HTML in Java is cumbersome and inefficient (both runtime and development cycle)
- We would like Web Designers to do design the page and programmers to do the logic (separation of concerns)
Scripting elements make the code messy and don’t allow for good separation of concerns (design/logic). Avoid scripting elements!

But you’re too smart for that...

JSP as a Template Engine

Example (Query Form)

Example (JSP Template)

Example (Servlet)
Example: *include* Tag

- Includes the content of another page (static or dynamic) in the current page

```html
<jsp:page contentType="text/html">
<html>
  <body>
    <jsp:include page="header.html"/>
    <jsp:include page="menu.jsp"/>
    <h1>My special page</h1>
    Here is the content of my special page . . .
    <jsp:include page="footer.html"/>
  </body>
</html>
```

Example

- Using `<jsp:include>` to include a result of a Servlet

```
<%@ page language="java" contentType="text/html"%>
<%@ page import="javairie.stats"%>

<html>
<head>
  <title>Statistics</title>
</head>
<body>
  <h1>Population Statistics</h1>
  <table>
    <tr><th>Population</th><th>Odds</th></tr>
    <tr><td>Population 1</td><td>1/2</td></tr>
    <tr><td>Population 2</td><td>1/3</td></tr>
  </table>
</body>
</html>
```
Example
Using `<jsp:include>` to include a result of a Servlet

```
Example

Multiplication Table

Protected void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException {
        try {
            int rows = Integer.parseInt(
                request.getParameter("rows"));
            int columns = Integer.parseInt(
                request.getParameter("columns"));
            PrintWriter writer = response.getWriter();
            printTable(writer, rows, columns);
        } catch (NumberFormatException e) {
            System.out.println("Illegal parameter value");
        }
    }
```

JSTL

- JSTL - JSP Standard Tag Library
- Collection of tags for functions that are common to many web apps
  - flow control, formatting, IO...
  - Introduces EL expressions

**Note:** JSTL introduce many Tags, but not all make sense. Tags that access the database or read files directly from the JSP page is a bad idea! In general all the logic of fetching the data should be done in lower layer.

```
if Tag

```

```
if Tag

```
foreach Tag

```html
<%@ page contentType="text/html" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<body>
<h1>Course List</h1>
<table border="1" cellpadding="3" cellspacing="0">
<tr>
<th width="80" align="center">Symbol</th>
<th width="400" align="center">Name</th>
<th width="120" align="center">Instructor</th>
</tr>
<c:forEach var="course" items="${courses}"
<tr>
<td align="center">${course.symbol}</td>
<td align="center">${course.name}</td>
<td align="center">${course.instructor}</td>
</tr>
</c:forEach>
</table>
</body>
</html>
```

Custom Tag Example

Repeated HTML code

Tag Libraries

- JSP allows you to add additional tags through tag-libraries
- You can wrap a commonly used presentation code in a new tag and provide it to the web-designer
- You can find numerous tag libraries (notably JSTL) or develop your own

Custom JSP Tag

```java
package examples.web.tags;
import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class RankTag extends TagSupport {
    private static final int MAX_RANK = 5;

    public void setRank(int rank) {
        this.rank = rank;
    }
}
```

Custom JSP Tag

```jsp
<%@ page contentType="text/html" %>
<%@ taglib prefix="examples" uri="http://www.idc.ac.il/j2ee/examples" %>
<h2>Recommended EJB Books</h2>
<table border=0>
<tr>
<td><examples:rank rank="5"/></td>
<td>Enterprise JavaBeans / Reichard Monson-Haefel</td>
</tr>
<tr>
<td><examples:rank rank="5"/></td>
<td>Mastering Enterprise JavaBeans / Ed Roman</td>
</tr>
<tr>
<td><examples:rank rank="4"/></td>
<td>EJB Design Patterns / Floyd Marinescu</td>
</tr>
</table>
```
Custom JSP Tag

```java
public int doStartTag() throws JspException {
    JspWriter writer = pageContext.getOut();
    try {
        writer.println("<table border=0 width=50">
            <tr height=20 width=50>
                for (int i=0; i<MAX_RANK; i++) {
                    writer.println(i<rank ? "<td width=10><image src="res/icons/star.gif"></td>") :
                        "<td width=10>&nbsp;</td>";)
                writer.println("</tr></table>";)
            catch (IOException e) {
                e.printStackTrace();
            }
        return EVAL_PAGE;
    }
```}

WAFs

- Such a framework is referred to as WAF (Web Application Framework)
- There are many Java based WAFs, on top of JSP and/or Servlets
- We’ll describe a general design for working with JSP & Servlets. It is used in Struts2 and other WAFs

Web Application Framework

- Real applications consists of many Web-pages both dynamic & static
- Pages forward/redirect request to other pages, include other pages, and call components to perform logic
- We need a consistent framework for organizing all presentation and logic components and the interaction between them

Model2

- Model2* applies a design pattern known as MVC to the problem
  - The client directs all requests to a single controller Servlet
  - The controller process the request, acts on the model and forwards the request to another Servlet/JSP for rendering
  - The target Servlet/JSP render the response according to attributes passed by the controller via request/session attributes

Model2 (MVC)

- Forward request after setting attributes in request or session
Passing parameters

```java
public class ControllerServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, ...) {
        ... request.setAttribute("price", new Float(price));
        ... forward
    }
}
```

Rendering JSP page

```html
<%@ page contentType="text/html" %>
Price: ${price}
```

Using JavaBeans

- JSP allows to separate presentation from logic by using JavaBeans
- You ‘embed’ a JavaBean in your page
- Set its properties, typically according to request parameters values
- The bean calculates the value of other properties accordingly
- Retrieve the value of the calculated property and include it in the page

Mortgage Calculator Example

```java
/**
 * A JavaBean component for calculating a mortgage
 * monthly payment
 */
public class MortgageCalcBean implements Serializable {
    // The requested amount of the loan
    private double loanAmount;
    // The annual interest
    private double interest;
    // The requested period of the load in years.
    private int years;
    // Hold the value of the properties of the bean
}
```
Settter/Getter methods define the properties of the bean.

MortgageCalcBean

```java
public double getLoanAmount() {
    return loanAmount;
}

public void setLoanAmount(double loanAmount) {
    this.loanAmount = loanAmount;
}

public double getInterest() {
    return interest;
}

public void setInterest(double interest) {
    this.interest = interest;
}

public double getMonthlyPayment() {
    double monthlyPayment = calculateMonthlyPayment();
    return monthlyPayment;
}

private double calculateMonthlyPayment() {
    double monthlyInterest = interest * 0.01 / 12;
    int months = years * 12;
    double debt = Math.pow(1+monthlyInterest, months);
    double monthlyPayment =
        loanAmount*debt*monthlyInterest/(debt-1);
    monthlyPayment = Math.floor(monthlyPayment*100)/100.0;
    return monthlyPayment;
}
```

Read only property to return the result.

**Packaging**

- A web-application consists of many elements
  - Static HTML pages and resources (images, clips, ...)
  - JSP pages and tag-libraries
  - Servlets, Java classes, JARs
  - Configuration files (descriptors)

---

**mortgage_calculator.jsp**

```jsp
<h1 align="center">Simple Mortgage Calculator</h1>

<p>Below is a simple calculator for calculating the monthly payment of the mortgage. Please enter the required loan amount, the interest you expect to get for the loan and the period of the loan in years; then press the 'calculate' button.</p>

... tabs, tables to enhance the look of the page

```
Packaging

Example

Apache Ant and other insects...

- Manual packaging is tedious & error-prone
- Application server tools & IDEs provide varying level of support for packaging, deployment and editing descriptor files - often not enough
- We’ll use Apache Ant, which is a generic build tool and de-facto standard building Java applications
- More on Ant in a separate presentation...

web.xml

```xml
WEB-INF"

<web-app>
  <servlet>
    <servlet-name>CourseListServlet</servlet-name>
    <servlet-class>CourseListServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>CourseListServlet</servlet-name>
    <url-pattern>/course_list</url-pattern>
  </servlet-mapping>
  <env-entry>
    <env-entry-name>jdbc/IDCDB</env-entry-name>
    <env-entry-value>
      !org.gjt.mm.mysql.Driver!jdbc:mysql://localhost:...
    </env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
</web-app>
```