

More on Servlet

- Http related header info
- Initialization -- context vs config scope
- Request Dispatcher
- Session tracking
- **More on annotations and web.xml**

3



3

More on web.xml

- Welcome page list
- Error page
- Exception page
- Load on startup

```
<web-app ...>
...
<servlet>
  <servlet-name>... </servlet-name>...
  <servlet-class>... </servlet-class>...
  <init-param>
    <param-name>databaseURL</param-name>
    <param-value>jdbc:mysql://localhost:3306/ae</param-value>
  </init-param>

  <init-param>
    <param-name>user</param-name>
    <param-value>myuser</param-value>
  </init-param>
</servlet>

<context-param>
  <param-name> user </param-name>
  <param-vlaue> admin </param-value>
</context-param>

<context-param>
  <param-name> adminEmail </param-name>
  <param-vlaue> admin@wrox.com </param-value>
</context-param>

<servlet-mapping>...
</web-app>
```

4



4

```

<web-app xmlns:xs ...
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>default.htm</welcome-file>
  </welcome-file-list>

</web-app>

```

No path or just /
will search in this
order

5



5

```

<web-app xmlns:xs ...
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>Osap</welcome-file>
  </welcome-file-list>

  <error-code>404</error-code>
  <location>/res/my404.html</location>
</error-page>

  <error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/res/myException.html</location>
  </error-page>
</web-app>

```

No path or just /
will search in this
order

or html, jsp

6



6

Load on startup

- By default, Servlet is not loaded until servlet container receives a request for the particular servlet.
- This may cause a delay in accessing the servlet for the first time.
- To avoid the delay in access time, you can use `<load-on-startup>` tag in your web.xml file that allows you to force the servlet container to load (instantiated and have its `init()` called) the servlet as soon as the server starts.

```
<web-app>
...

<servlet>
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>DemoServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
...
</web-app>
```

- value ≥ 0 means that the servlet is loaded when the web-app is deployed or when the server starts, if the value < 0 then servlet would be loaded whenever the container feels like.

Main topics (tentative)

- **Web App Architecture. Preliminary knowledge/Review**
 - Client side: HTML CSS JavaScript
 - Java, UML design patterns, Thread, Serialization. relational databases SQL
- **Client-Server, low level: socket programming RMI**
- **Web applications (server side)**
 - LAMP/CGI
 - Java Servlet
 - JSP, JavaBean, MVC pattern
 - Database access: RDBMS, JDBC
 - More: listener, filter, Ajax, JSON
- **Web (RESTful) services, micro services**
- **Advanced topics (TBD):** Deployments: Docker container, Node JS, React, Angular, Spring Boot
- **Other advanced topics (TBD)**



JSP

- Servlets can be used to generate dynamic Web content.
- One drawback, however, is that we have to embed HTML tags and text inside the Java source code – good at processing data but not so good at presentation.
- Using servlets, you have to modify the Java source code and recompile it if changes are made to the HTML text.
- Writing HTML script is tedious and error-prone. Have a lot of HTML script in a servlet, the program is difficult to write, read and maintain, since the HTML text is part of the Java program.
- “(write) **HTML inside Java**”

```
out.println("Hobbies: ");
for (String h: Hobbies) {
    if (h.equals("Thinking"))
        out.println("<input type='checkbox' name='ice' value='\"
    else out.println("<input type='checkbox' name='ice' value='\"
}
out.println("<hr>");

// Submit and reset buttons
out.println("<input type='reset' value='Clear' />");
out.println("<input type='submit' value='Submit' />");
out.println("</form>");
```

Book ID	AUTHOR	TITLE	PRICE
<input checked="" type="checkbox"/> 3	Tan Ah Teck	Java for Dummies	\$22.0
<input type="checkbox"/> 4	Tan Ah Teck	More Java for Dummies	\$42.0
<input checked="" type="checkbox"/> 5	Joe Suh	Good Java Style	\$12.0

Enter your Name:
 Enter your Phone Number:
 Choose your City:

[Back to Select Menu](#)



10

JSP

- Servlets can be used to generate dynamic Web content.
- One drawback, however, is that we have to embed HTML tags and text inside the Java source code – good at processing data but not so good at presentation.
- Using servlets, you have to modify the Java source code and recompile it if changes are made to the HTML text.
- Writing HTML script is tedious and error-prone. Have a lot of HTML script in a servlet, the program is difficult to write, read and maintain, since the HTML text is part of the Java program.
- “(write) **HTML inside Java**”
- It would be easier for servlet, after calculation, to forward to a HTML page, but HTML page is static.
- JavaServer Pages (JSP) was introduced to remedy this drawback.
- JSP enables you to write regular HTML script in the normal way and, for embed Java code to produce dynamic content.

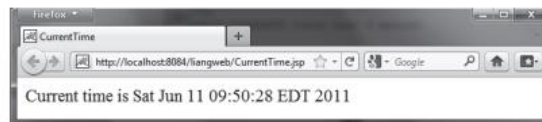
“(write) **Java inside HTML**”



11

Simple example

```
<!-- CurrentTime.jsp -->
<html>
  <head>
    <title>
      CurrentTime
    </title>
  </head>
  <body>
    Current time is <%= new java.util.Date() %>
  </body>
</html>
```



A JSP page must first be processed by a Web server before it can be displayed in a Web browser. The Web server must support JSP, and the JSP page must be stored in a file with a .jsp extension.

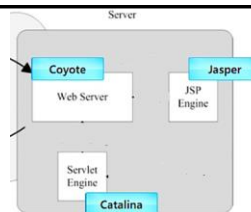
The Web server translates the .jsp file into a Java servlet, compiles the servlet, and executes it. The result of the execution is sent to the browser for display.

12

Little delay

12

Recall:



Catalina

Catalina is Tomcat's servlet container. Catalina implements Sun Microsystems' specifications for servlet and JavaServer Pages (JSP).

Coyote

Coyote is a Connector component for Tomcat that supports the HTTP 1.1 and 2 protocol as a web server. This allows Catalina, nominally a Java Servlet or JSP container, to also act as a plain web server that serves local files as HTTP documents. Coyote listens for incoming connections to the server on a specific TCP port and forwards the request to the Tomcat Engine to process the request and send back a response to the requesting client.

Jasper

Jasper is Tomcat's JSP Engine. Jasper parses JSP files to compile them into Java code as servlets (that can be handled by Catalina). At runtime, Jasper detects changes to JSP files and recompiles them.

13

13

Sample JSP Expression: Random Number

```
<H1>A Random Number</H1>
<%= Math.random() %>
```

Representative Resulting Servlet Code: Random Number

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    HttpSession session = request.getSession();
    JspWriter out = response.getWriter();
    out.println("<H1>A Random Number</H1>");
    out.println(Math.random());
    ...
}
```

You can find the compiled servlet at

workspaceDir/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/work/Catalina/localhost/pr
ojectName/org/apache/jsp



14

14

JSP's role, replace Servlet?

- Writing HTML is a natural way, which is easier than in Servlet.
- Enables separating processing with presentation
- Replace servlet?
- JSP is not good at complicated processing data. Adding lots Java in JSP also mix the presentation and processing.
- Solution: use both Servlet and JSP.
- Servlet focus on data processing, business logic, decide what view to give, and forward request to JSP. JSP focus on presentation.
- This is the MVC or Model 2 architecture.



15

15

JSP Servlet integration

- JSP is not a replacement of Servlet, it is a complement of Servlet. View in MVC architecture.

Model 2 (MVC) Architecture

Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

Model The model represents the state (data) and business logic of the application.

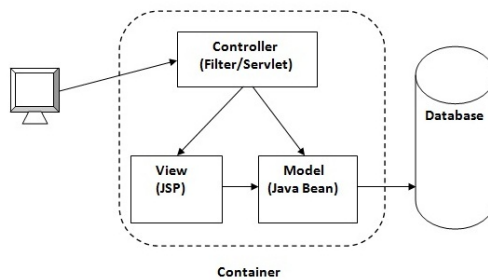
View The view module is responsible to display data i.e. it represents the presentation.

Controller The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

JavaBeans are a logical choice for implementing State

JSPs are a good choice for implementing the View

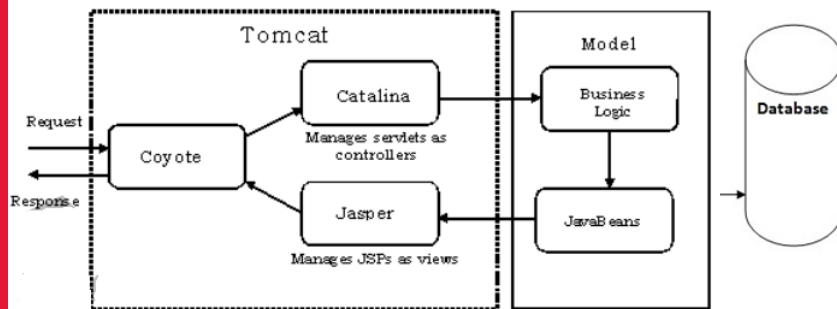
Servlets are an ideal choice for the controller



16

YORK
UNIVERSITY

16



18

YORK
UNIVERSITY

18

JavaBeans? Business Logic Blocks

- A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.
- Following are the unique characteristics that distinguish a JavaBean from other Java classes –
 - It provides a default, no-argument constructor.
 - It should be serializable and that which can implement the Serializable interface.
 - It may have a number of properties which can be read or written.
 - **It may have a number of "getter" and "setter" methods for the properties.**



21

```
getPropertyName()
```

For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property. This method is called accessor.

```
setPropertyName()
```

For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property. This method is called mutator.

21

A JavaBean Example



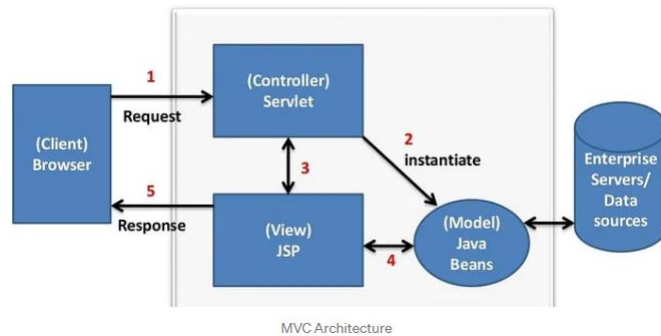
```
public class StudentsBean implements java.io.Serializable {
    private String firstName = null;
    private String lastName = null;
    private int age = 0;

    public StudentsBean() {
    }
    public String getFirstName(){
        return firstName;
    }
    public String getLastName(){
        return lastName;
    }
    public int getAge(){
        return age;
    }
    public void setFirstName(String firstName){
        this.firstName = firstName;
    }
    public void setLastName(String lastName){
        this.lastName = lastName;
    }
    public void setAge(Integer age){
        this.age = age;
    }
}
```

22



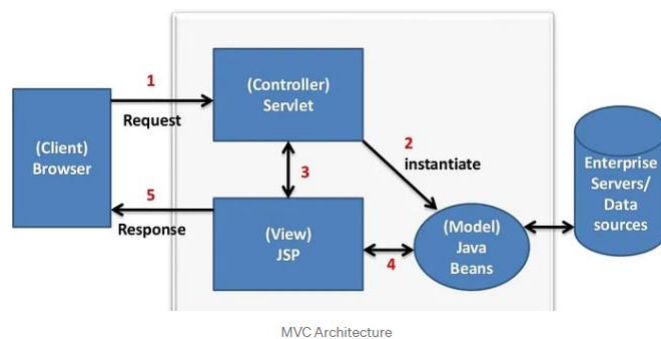
22



- The **controller** interprets requests, and decides on the course of actions based on business-logic. The controller is manifested as a **servlet**.
- The **model** is represented by JavaBeans that expose properties through getter/setter methods. **JavaBeans** represent entities in the business logic
- The **view** is responsible for setting up the presentation of the data and sending the HTML output to the client. The view is implemented as a **JSP**.

23

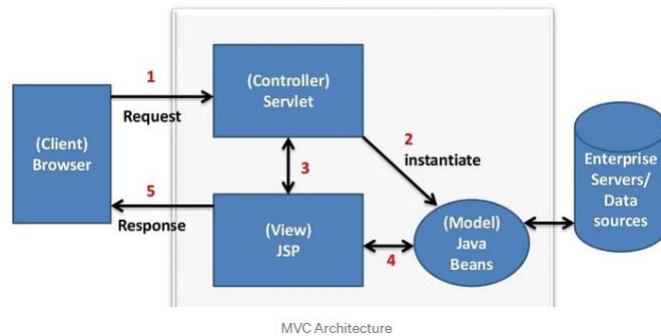
23



- 1. Define beans to represent the data.** Your first step is defining beans to represent the results that will be presented to the user.
- 2. Use a servlet to handle requests.** In most cases, the servlet reads request parameters
- 3. Populate the beans.** The servlet invokes business logic (application specific code) or data-access code to obtain the results. The results are placed in the beans that were defined in step 1.

24

24

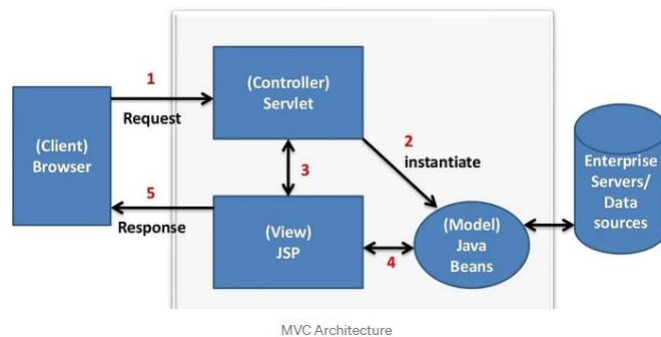


4. **Store the bean in the request, session, or servlet context.** The servlet calls `setAttribute` on the request, session, or servlet context objects to store a reference to the beans that represent the results of the request.
5. **Forward the request to a JSP page.** The servlet determines which JSP page is appropriate to the situation and uses the `forward` method of `RequestDispatcher` to transfer control to that page.
6. **Extract the data from the beans.** The JSP page accesses beans. The page then output the bean properties. The JSP page does not create or modify the bean; it merely extracts and displays data that the servlet created.

25



25



Now that we see the high level role of JSP, let's look at the syntax.

- Older ways: scriptlets
- Newer way: EL, JSTL

27



27

JSP Scripting Constructs

- Scripting elements enable you to specify Java code that will become part of the resultant servlet
- A JSP scriptlet enables you to insert a Java statement into the servlet's *jspService* method, which is invoked by the service method. A JSP scriptlet has the following form:

```
<% Java statement %>
```

- A JSP expression is used to insert a Java expression directly into the output. It has the following form:

```
<%= Java expression %>
```

The expression is evaluated, converted into a string, and sent to the output stream of the servlet.

- A JSP declaration is for declaring methods or fields into the servlet. It has the following form:

```
<%! Java declaration %>
```

28



28

another

```

2  <%@page language="java" contentType="text/html" pageEncoding="UTF-8" %>
3  <!DOCTYPE html>
4  <html>
5  <head>
6    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
7    <title>First JSP</title>
8  </head>
9
10 <body>
11   <%
12     double num = Math.random();
13     if (num > 0.95) {
14       %>
15       <h2>You'll have a luck day!</h2><p><%= num %></p>
16     } else {
17       %>
18       <h2>Well, life goes on ... </h2><p><%= num %></p>
19     }
20   <%
21   %>
22   <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
23 </body>
24 </html>
25


```

Diagram illustrating the mapping of JSP constructs to their corresponding Java code:

- The JSP scriptlet `<% double num = Math.random();` is mapped to the `<% Java statement %>` construct.
- The JSP expression `<%= num %>` is mapped to the `<%= Java expression %>` construct.
- The JSP declaration `<% request.getRequestURI() %>` is mapped to the `<%= Java expression %>` construct.



29



```

1 <%@page language="java" contentType="text/html" pageEncoding="UTF-8" %>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
6 <title>First JSP</title>
7 </head>
8 <body>
9 <%
10 double num = Math.random();
11 if (num > 0.95) {
12 <h2>You'll have a luck day!</h2><p><%= num %></p>
13 } else {
14 <h2>Well, life goes on ... </h2><p><%= num %></p>
15 }
16 <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
17 </body>
18 </html>

```

workspaceDir/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/work/Catalina/localhost/projectName/org/apache/jsp

```

public final class first_jsp extends org.apache.jasper.runtime.HttpJspBase
implements org.apache.jasper.runtime.JspSourceDependent {

    public void _jspInit() { ..... }

    public void _jspDestroy() { ..... }

    public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, ServletException {

        // JSP pre-defined variables (in service() method)
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;

        JspWriter _jspx_out = null;
        PageContext _jspx_page_context = null;

        try {
            response.setContentType("text/html");
            pageContext = _jspxFactory.getPageContext(this, request, response,
            null, true, 8192, true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;

            out.write("<!DOCTYPE HTML>\r\n");
            out.write("<html>\r\n");
            out.write("<head>\r\n");
            out.write("    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\r\n");
            out.write("    <title>First JSP</title>\r\n");
            out.write("</head>\r\n");
            out.write("<body>\r\n");
            out.write("    ");

            double num = Math.random();
            if (num > 0.95) {
                out.write("\r\n");
                out.write("        <h2>You will have a luck day!</h2><p>(");
                out.print(num);
                out.write("</p>\r\n");
                out.write("    ");
            } else {
                out.write("\r\n");
                out.write("        <h2>Well, life goes on ... </h2><p>(");
                out.print(num);
                out.write("</p>\r\n");
                out.write("    ");
            }

            out.write("<a href=\"");
            out.write(request.getRequestURI());
            out.write("><h3>Try Again</h3></a>\r\n");
            out.write("</body>\r\n");
            out.write("</html>");

```

30

JSP Predefined variables (implicit objects)

These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared.

Object & Description	
request This is the HttpServletRequest object associated with the request. <% request.getParameter(..) request.setAttribute(..) request.getProtocol() > request.getSession() >	application This is the ServletContext object associated with the application context. application.setAttribute() , application.getAttribute() , application.getInitParameter()
response This is the HttpServletResponse object associated with the response to the client. response.setCookie(..)	config This is the ServletConfig object associated with the page. config.getInitParameter()
out This is the PrintWriter object used to send output to the client. out.println(..)	pageContext This encapsulates use of server-specific features like higher performance JspWriters .
session This is the HttpSession object associated with the request. session.setAttribute() , session.getAttribute()	page This is simply a synonym for this , and is used to call the methods defined by the translated servlet class.
	Exception The Exception object allows the exception data to be accessed by designated

31

<%= "ABC" %> same as <% out.print("ABC") %>

31

JSP Predefined variables (implicit objects)

Implicit Objects and their corresponding classes:

IMPLICIT OBJECT	CLASS
out	javax.servlet.jsp.JspWriter
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
exception	javax.servlet.jsp.JspException
page	java.lang.Object
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig

32



32

```
<%@ page import = "java.io.*,java.util.*" %>

<html>
  <head>
    <title>HTTP Header Request Example</title>
  </head>

  <body>
    <center>
      <h2>HTTP Header Request Example</h2>

      <table width = "100%" border = "1" align = "center">
        <tr bgcolor = "#949494">
          <th>Header Name</th>
          <th>Header Value(s)</th>
        </tr>
        <%
          Enumeration headerNames = request.getHeaderNames();
          while(headerNames.hasMoreElements()) {
            String paramName = (String)headerNames.nextElement();
            out.print("<tr><td>" + paramName + "</td><n>");
            String paramValue = request.getHeader(paramName);
            out.println("<td> " + paramValue + "</td></tr><n>");
          }
        <%>
      </table>
    </center>

  </body>
</html>
```

HTTP Header Request Example

Header Name	Header Value(s)
accept	*/
accept-language	en-us
user-agent	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1 Trident/4.0; InfoPath.2; MS-RTC LM 8)
accept-encoding	gzip, deflate
host	localhost:8080
connection	Keep-Alive
cache-control	no-cache

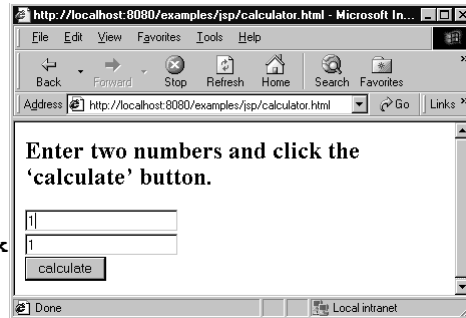
33

Another JSP

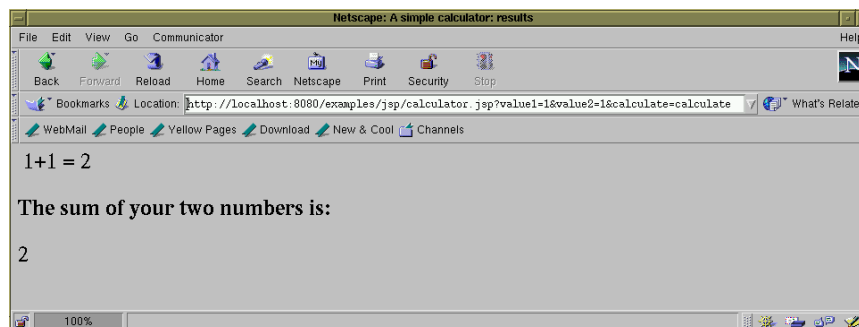
```
<html>
<head></head>
<body>
<p>Enter two numbers and click
'calculate' button.</p>

<form action="calculator.jsp" method="get">
  <input type="text" name="value1"><br>
  <input type="text" name="value2" ><br>
  <input type="submit" name="calculate" value="calculate">
</form>
</body>
</html>
```

calculator.html



34

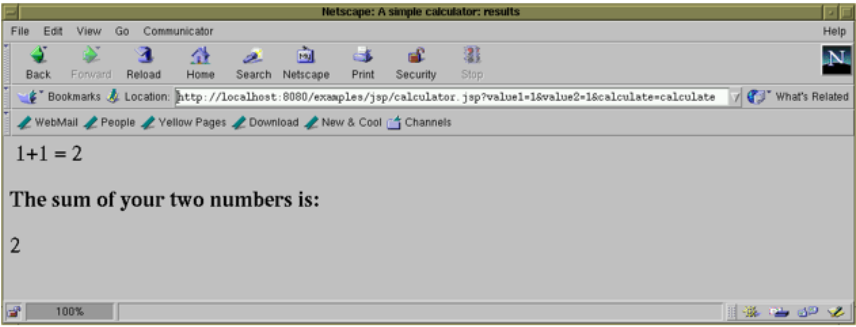


```
<html>
<head><title>A simple calculator: results</title></head>
<body>
<%-- A simpler example 1+1=2 --%>
1+1 = <%= 1+1 %> // or <% out.print(1+1) %>
<%-- A simple calculator --%>
<h2>The sum of your two numbers is:</h2>
<%= Integer.parseInt(request.getParameter("value1")) +
Integer.parseInt(request.getParameter("value2"))
%>
</body>
</html>
```

calculator.jsp



35



```

<html>
<head><title>A simple calculator: results</title></head>
<body>
<!-- A simpler example 1+1=2 -->
1+1 = <%= 1+1 %> // or <% out.print(1+1) %>
<!-- A simple calculator -->
<h2>The sum of your two numbers is:</h2>
<%= Integer.parseInt(request.getParameter("value1")) +
      Integer.parseInt(request.getParameter("value2"))
%>
</body>
</html>

```

YORK UNIVERSITY

calculator.jsp

36

```

1 <!-- ComputeLoan.html -->
2 <html>
3 <head>
4 <title>ComputeLoan</title>
5 </head>
6 <body>
7 <form method = "get" action = "ComputeLoan.jsp">
8   Compute Loan Payment<br />
9   Loan Amount
10  <input type = "text" name = "loanAmount" /><br />
11  Annual Interest Rate
12  <input type = "text" name = "annualInterestRate" /><br />
13  Number of Years
14  <input type = "text" name = "numberOfYears" size = "3" /><br />
15  <p><input type = "submit" name = "Submit"
16    value = "Compute Loan Payment" />
17    <input type = "reset" value = "Reset" /></p>
18  </form>
19 </body>
20 </html>

```

LISTING 38.3 ComputeLoan.jsp

```

1 <!-- ComputeLoan.jsp -->
2 <html>
3 <head>
4 <title>ComputeLoan</title>
5 </head>
6 <body>
7 <% double loanAmount = Double.parseDouble(
8   request.getParameter("loanAmount"));
9   double annualInterestRate = Double.parseDouble(
10    request.getParameter("annualInterestRate"));
11   double numberOfYears = Integer.parseInt(
12    request.getParameter("numberOfYears"));
13   double monthlyInterestRate = annualInterestRate / 1200;
14   double monthlyPayment = loanAmount * monthlyInterestRate /
15     (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
16   double totalPayment = monthlyPayment * numberOfYears * 12; %>
17   Loan Amount: <%= loanAmount %><br />
18   Annual Interest Rate: <%= annualInterestRate %><br />
19   Number of Years: <%= numberOfYears %><br />
20   <b>Monthly Payment: <%= monthlyPayment %><br />
21   Total Payment: <%= totalPayment %><br /></b>
22 </body>
23 </html>

```

YORK UNIVERSITY

37

application

```
<web-app>
...
<context-param>
  <param-name>contextParameter1</param-name>
  <param-value>ValueOfContextParameter1</param-
value>
</context-param>
</web-app>
```

```
<html>
<head> <title> Config Implicit Object</title>
</head>
<body>
<%
  String sname=(String) application.getInitParameter("contextParameter1");
  out.print("Context parameter is: " + sname);
%>
</body>
</html>
```

38



38

config

web.xml file

```
<web-app>

<servlet>
  <servlet-name>sonoojaiswal</servlet-name>
  <jsp-file>/welcome.jsp</jsp-file>

  <init-param>
    <param-name>dname</param-name>
    <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
  </init-param>

</servlet>
```

index.html

```
<form action="welcome">
  <input type="text" name="uname">
  <input type="submit" value="go"> <br/>
</form>
```

welcome.jsp

```
<%
  out.print("Welcome "+request.getParameter("uname"));

  String driver=config.getInitParameter("dname");
  out.print("driver name is="+driver);
%>
```

39



39

- We address the problem of *writing HTML inside Java*
- By *writing Java (scriptlet) inside HTML*
- Can be very messy and unreadable too
- Still intermingle processing and presentation

```

<%
String category = request.getParameter("category");
if (category != null){
    %>
<div>
    <span class="label" style="margin-left: 15px;"> List of <%= category%>
    Books
</span>
</div>
<%> %>
<table id="grid">
    <thead>
        <tr>
            <th id="th-title">Book Title</th>
            <th id="th-author">Author</th>
        </tr>
    </thead>

    <tbody>
        <%
        List<Book> books = (List<Book>)request.getAttribute("booklist");
        Iterator<Book> iterator = books.iterator();

        while (iterator.hasNext()) {
            Book book = (Book)iterator.next();
            Long bookId = book.getId();
            List<Author> authors = book.getAuthors();

            %>
            <tr>
                <th scope="row" id="r100"><%=book.getBookTitle()%></th>
                <% for (Author author: authors){
                    if ( book.getId().equals(author.getBookId())){
                        %><td><%=author.getFirstName()+" " +author.getLastName()%></td>
                    <%>
                }>
            </tr>
        }
    </tbody>
</table>

```



40