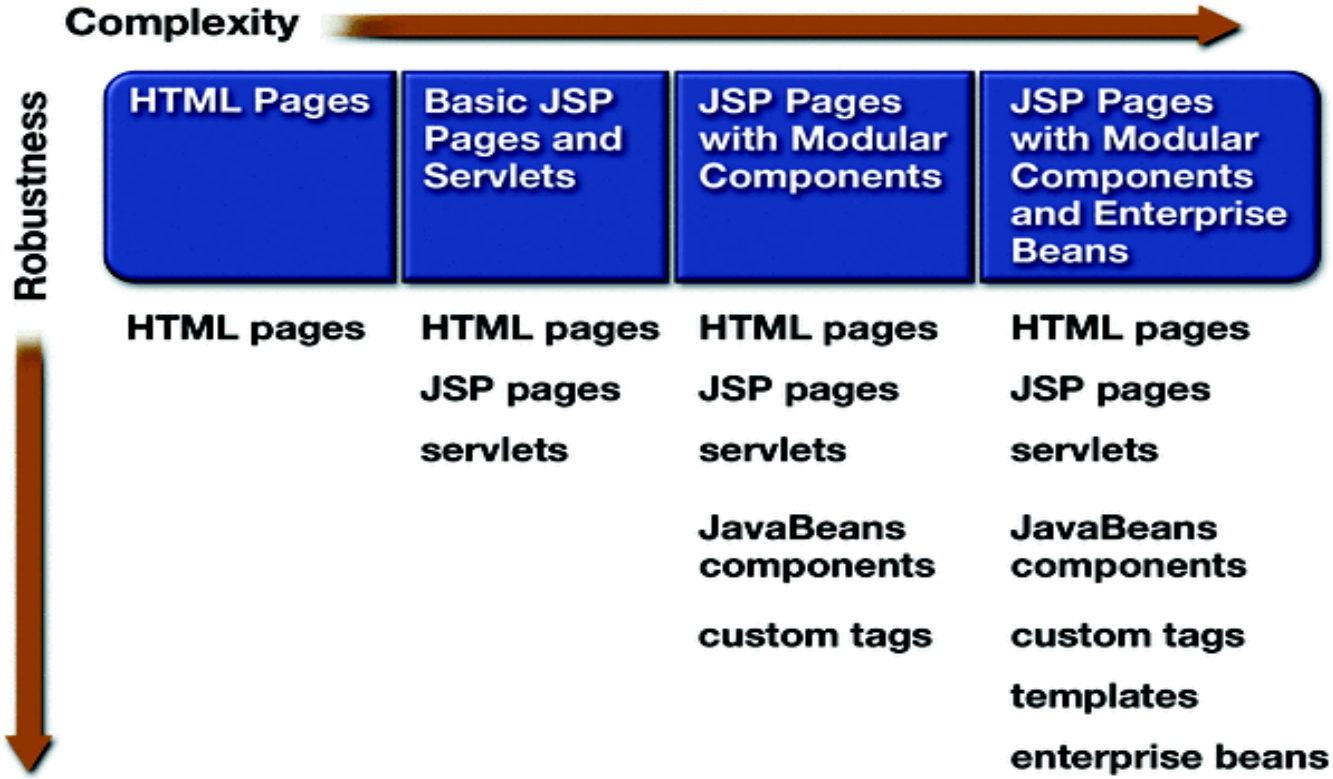
JSP

Rajeev Gupta
M. Tech. CS

Goal

- Simplification of creation and management of dynamic web sites

Web Application Designs



How JSP works ?

- First time JSP is loaded by JSP containers
- Servlet code necessary to fulfill jsp tag automatically generated compiled and loaded into servlet container by JSP container
- Compiled Servlet process any browser request for that JSP page
- If modified Source code of JSP
- It get automatically recompiled

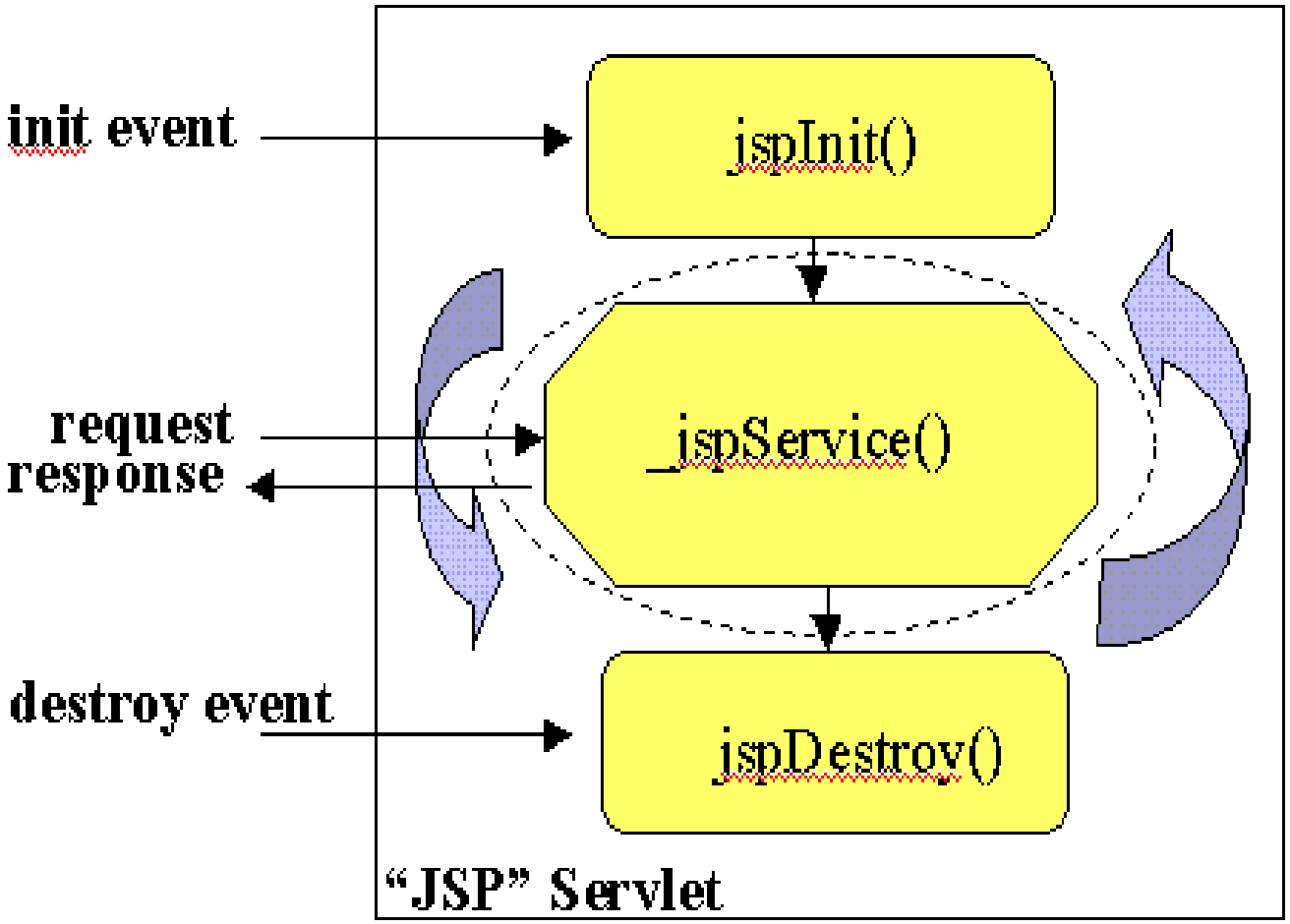
JSP Page Lifecycle Phases

- Translation phase
- Compile phase
- Execution phase

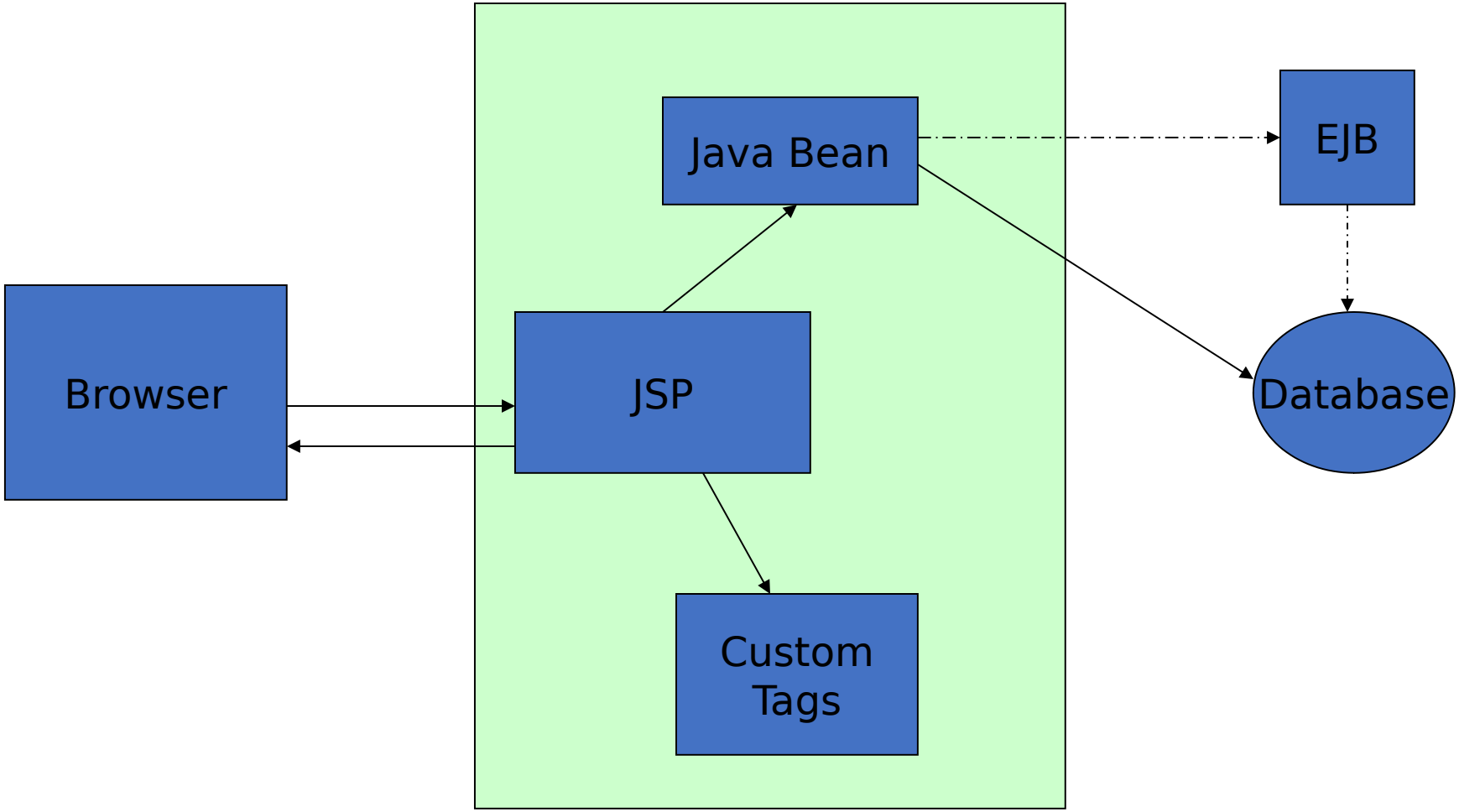
JSP Life Cycle

	Page first written		Server restarted		Page modified	
	Request #1	Request #2	Request #3	Request #4	Request #5	Request #6
JSP page translated into servlet	Yes	No	No	No	Yes	No
JSP's Servlet compiled	Yes	No	No	No	Yes	No
Servlet instantiated and loaded into server's memory	Yes	No	Yes	No	Yes	No
init (or equivalent) called	Yes	No	Yes	No	Yes	No
doGet (or equivalent) called	Yes	Yes	Yes	Yes	Yes	Yes

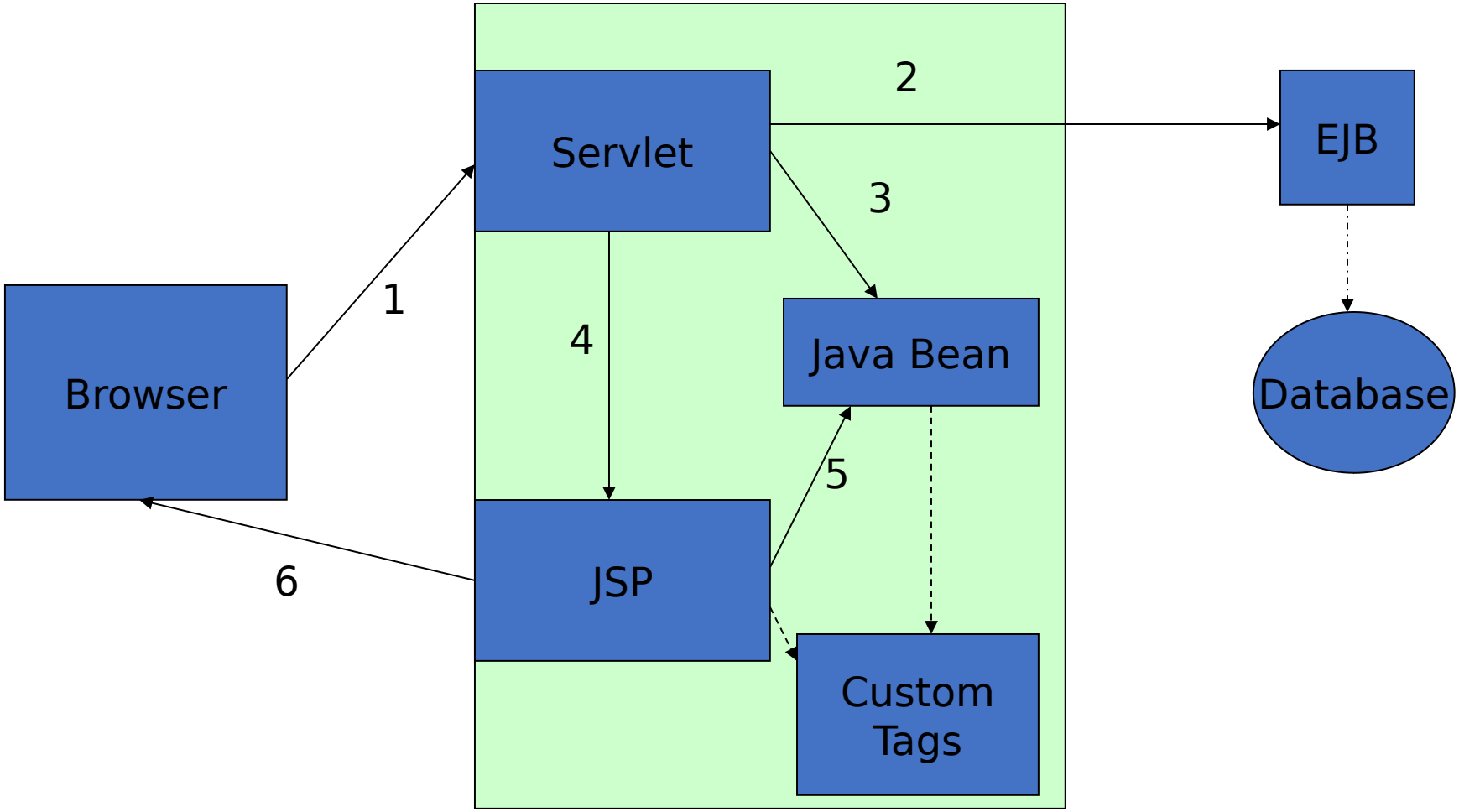
JSP Lifecycle Methods during Execution Phase



Model 1 Architecture



Model 2 Architecture



JSP tags

- HTML code: *<html-tag>content</html-tag>*
- JSP Comments: <%-- *comment* --%>
- Expressions: <%= *expression* %>
- Scriptlets (statements): <% *code* %>
- Declarations: <%! *code* %>
- Directives: <%@ *directive attribute*="*value*" %>
- Actions: <jsp:forward.../>, <jsp:include.../>
- Expression-Language Expressions: ${*expression*

JSP tags

JSP tags are classified as:-

1. Directive:
   - **Affect overall structure of Servlet that result from translation.**
2. Scripting Elements:
   - **Allows inserting Java code into JSP pages**
3. Action:
   - **Affect Runtime behavior of JSP**

JSP Directive

JSP Directive

Used to set global values such as class declaration, methods, O/P content type

Page directive have scope for entire page

<%@ .....%>

3 type of JSP Directive are:-

1. Page directive
2. Include directive
3. Taglib directive

page-Directive

- import attribute: A comma separated list of classes/packages to import

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

- contentType attribute: Sets the MIME-Type of the resulting document (default is text/html)

```
<%@ page contentType="text/plain" %>
```

- session="**true**|false" specifies if to use a session?

- buffer="*sizekb*|none|**8kb**"
  - Specifies the content-buffer (out) size in kilo-bytes

- autoFlush="**true**|false"
  - Specifies whether the buffer should be flushed when it fills, or throw an exception otherwise

- isELIgnored ="**true**|false"
  - Specifies whether *JSP expression language* is used

Page Directive Example…

<%@ page language="java" import="java.util.Date(), java.util.Dateformate()" iserrorpage="false"%>

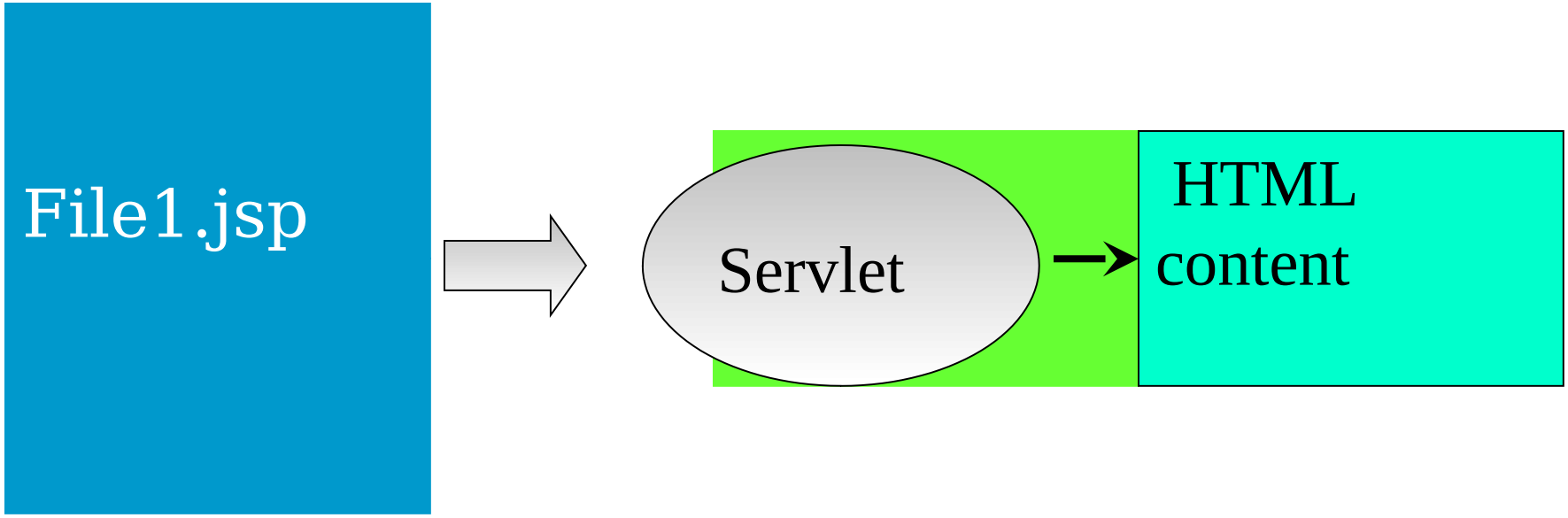
Include directive

Instruct container to include content of resources in current JSP (inserting inline)

Ex: consider a.jsp contains
<%@ include file="abc.jsp">
All code of abc.jsp is copied to a.jsp

As of Tomcat 5.x, generated Servlets are updated when included files change (unlike older versions…)

Include Directive



Include directive :Example

Demo.jps

```
<html>
<head></head>
<body>
  <h1>include directive</h1>
  <%@ include file="include.jsp"%>
</body>
<body>
</html>
```

include.jsp

```
<%page import="java.util.Date"%>
<%="Current date is"+new Date()%>
```

Error Pages

- We can set one JSP page to be the handler of uncaught exceptions of another JSP page, using JSP directives

- <%@ page errorPage="*url* " %>
  - Defines a JSP page that handles uncaught exceptions
  - The page in *url* should have true in the page-directive:

- <%@ page isErrorPage="true|false" %>
  - The variable exception

Error page Example

```
<html>
  <head><title>Reading From Database </title></head>
  <body>
    <%@ page import="java.sql.*" %>
    <%@ page errorPage="errorPage.jsp" %>

    <%
      Class.forName(“com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection(“.....”);

     %>
    <h2>Can Connect!!</h2>
  </body>
</html>
```

```
<html>
  <head><title>Connection Error</title></head>
  <body>
      <%@ page import="java.io.*" %>
      <%@ page isErrorPage="true" %>
     <h1>Oops. There was an error when you accessed the
         database.</h1>
    <h2>Here is the stack trace:</h2>
    <pre style="color:red">
    <% exception.printStackTrace(new PrintWriter(out)); %>
    </pre>
  </body>
</html>
```

Scripting Elements

Declaration

Block of Java code in JSP that used to define class wide variable and methods in generated servlet..

Ex:

```
<%! Int numtimes=3;
  public String sayHello(String name)
  {
      return (“Hello”+name);
  }
<html>
<head>Declaration test</head>
<body>
<p>The value of num time is <%=numtime %><?p>
<p>Hello to :<%=sayHello(“foo and bar…”)%></p>
</body>
</html>
```

Decleration Ex: counter

```
<%! private int accessCount = 0; %>
<%! private synchronized int incAccess() {
      return ++accessCount;
       } %>
<h1>Accesses to page since Servlet init:
<%= incAccess() %> </h1>
```

Scriptlets

Block of Java code that is executed during request processing time <%...%>

All code b/w <%...%> tags in JSP gets put into service() method

EX:

```
<%
  String un=request.getParameter("un");
%>
<input type="text" value="<%=un%>">
....
.....
```

Example: Scriptlet.jsp

Scriptlet.jsp

```
<html>
<head><title>hi it is scriptlet test </title></head>
<%
  for(int i=0;i<10;i++)
  {
  out.println("Scriptlet test"+i+"<br>");
  System.out.println("Goes to Console of server");
  }
%>
</body>
</html>
```

Example: login.jsp

```
<%
   if((request.getParameter("un").equals("raj"))
  &&(request.getParameter("pw").equals("java")))
  {
%>
  <jsp:forward page="forward2.jsp"/>
<%
    }
      else
    {
%>
  <%@include file="index.jsp"%>
<%

  }
%>
```

Scriptlet Misuse

```
<%@page language="java" import="java.sql.*"%>
<% String name=request.getParameter("name");
   String pass=request.getParameter("pw");
   try
   {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   connection con=null;
   con=DriverManager.getConnection("jdbc:odbc:rajconn","root","root");
   PreparedStatement st=con.prepareStatement("select * from pass where username=? And password=?");
   st.setString(1,un);
   st.setString(2,pw);
   ResultSet rs=st.executeQuery();
   if(rs.next())
   {
%>
   <jsp:forward page="forward2.jsp"/>
<%
   }
   else
   {
   out.println("Invalid un and password");
%>
   <%@include file="index.jsp"%>

<%
   }
   catch(Exception e)
   {
   out.println(e.getMessage());
%>
```

Expression

<%= and %>

Short cut notation for script let that sends value off a java expression to the client

Expression.jsp

```
<html>
<head><title>hi it is expression test </title></head>
<%! int i=0;%>
<%i++; %>
<%=“This jsp is accessed “+i+” times”%>

</body>
</html>
```

Predefined Variables (Implicit Objects)

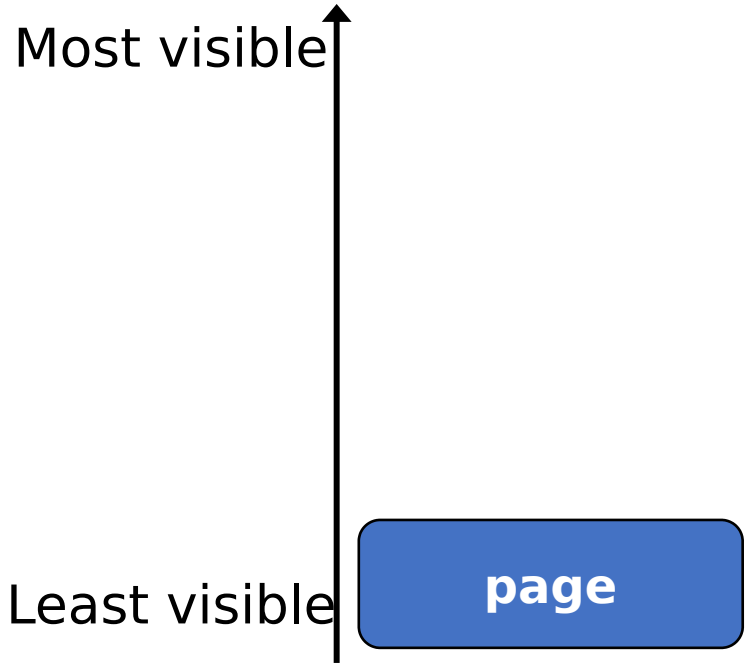
- The following predefined variables can be used:
  - **request**: the HttpServletRequest
  - **response**: the HttpServletResponse
  - **session**: the HttpSession associated with the request
  - **out**: the PrintWriter (a buffered version of type JspWriter) used to fill the response content
  - **application**: The ServletContext
  - **config**: The ServletConfig

```
<html>
 <head>
  <title>JSP Expressions</title>
 </head>
        <body>

       <li>Current time: <%= new java.util.Date()
      %></li>
            <li>Your hostname:<%=
      request.getRemoteHost() %></li>
            <li>Your session ID: <%= session.getId()
      %></li>
            <li>The <code>testParam</code> form
      parameter:
                  <%= request.getParameter("testParam")
      %></li>
          </body>
      </html>
```

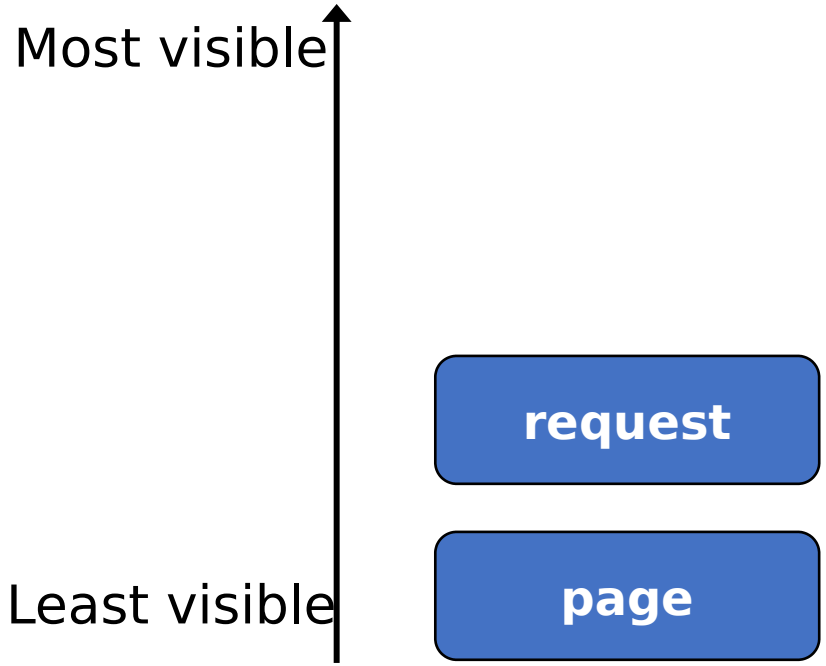
Understanding Scopes

Different Object Scopes



Objects may be accessed only within Pages where they are created

Different Object Scopes



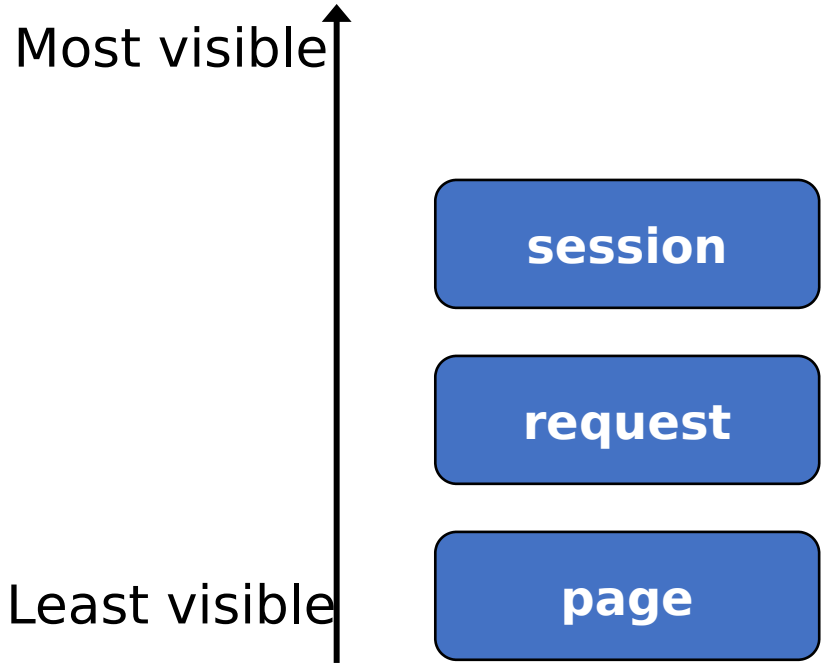
request

Only within pages processing the Request in which they are created

page

Objects may be accessed only within Pages where they are created

Different Object Scopes

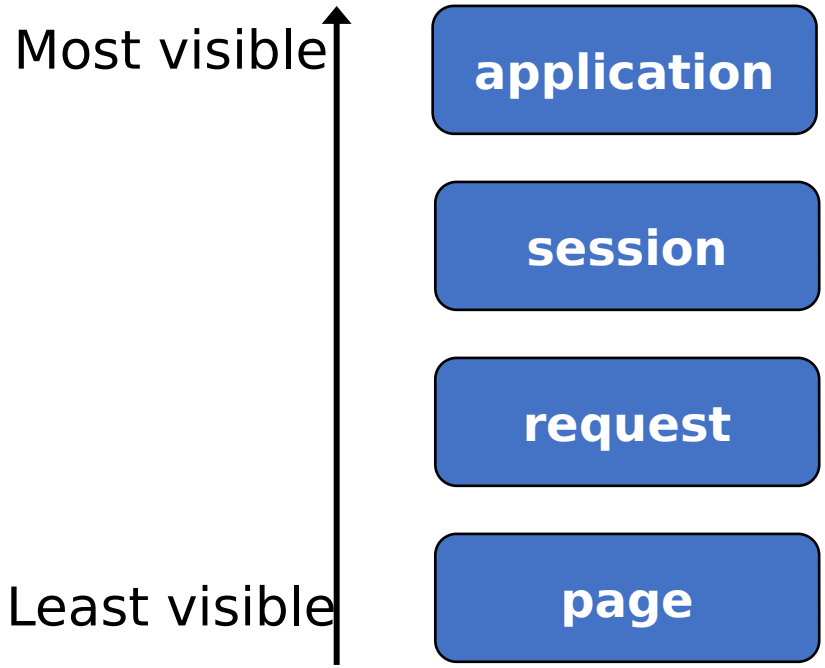


Only from pages belonging to same session as the one in which they were created

Only within pages processing the request in which they are created

Objects may be accessed only within pages where they are created

Different Object Scopes



Within all pages belonging to same application

Only from pages belonging to same session as the one in which they were created

Only within pages processing the request in which they are created

Objects may be accessed only within pages where they are created

Object Scopes
session, request & page

Object Scopes
session vs. application

Standard Actions
JSP beans

Standard Actions

Tags that affect runtime behavior of JSP and response send back to client

Std action types:

<jsp:useBean>

<jsp:setProperty>

<jsp:getProperty>

<jsp:param>

<jsp:include>

<jsp:forward>

<jsp:plugin>

<jsp:useBean>

To separate code from presentation …

Encapsulate code in a
JavaBean (POJO) and then instantiate and use this object within our jsp

<jsp:useBean> used to create instance of JavaBean and assign to a variable name(or id)

<jsp:useBean id="ob" scope="scopename" class="Emp"/>

Scope: request. session.,application or page

<jsp:setProperty property="name" name="ob" value="raj"/>
<jsp:getProperty property="name" name="ob"/>

Small Examples on JavaBean

```
<form action="MyJspController.jsp">
    Name: <input type="text" name="name"/>
    Pass: <input type="password" name="pass"/>
<input type="submit"/>
</form>
```

User.java i.e. bean

```
package com;
        import java.io.Serializable;
        public class User implements Serializable{
        private String name;
        private String pass;
        public String getName() {
        return name;
        }
        public void setName(String name) {
        this.name = name;
        }
        public String getPass() {
        return pass;
        }
        public void setPass(String pass) {
        this.pass = pass;
        }

        public boolean isValid(){
        if(this.getName().equalsIgnoreCase("raj")&&this.getPass().equalsIgnoreCase("raj"))
        return true;
        else
        return false;

        }

}
```

Bean.jsp

```
<jsp:useBean id="ob" class="com.User" scope="page">
<jsp:setProperty property="name" name="ob"/>
<jsp:setProperty property="pass" name="ob" />
</jsp:useBean>


<jsp:getProperty property="name" name="ob"/>
<jsp:getProperty property="pass" name="ob"/>


<%= ob.isValid() %>
```

<jsp:include> and <jsp:forward>

This action allows a static/dynamic resources specified by URL to be included in current JSP at processing time

An included page has only access to JspWriter object

Equivalent to javax.servlet.RequestDispatcher include()method..

If page O/P is buffered then buffered is flushed prior to inclusion

<jsp: include page="my.jsp" flush="true">

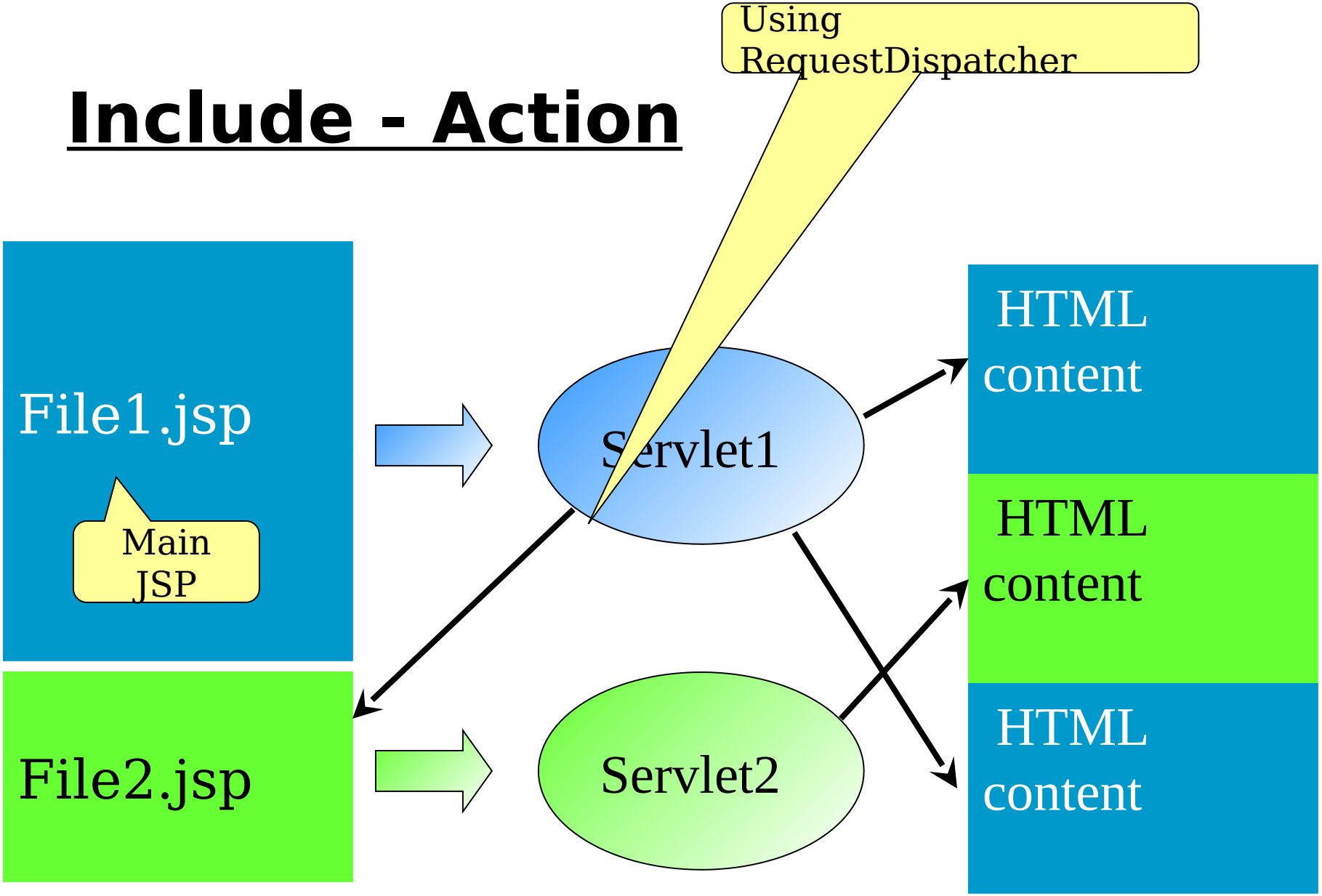
Include page my.jsp and buffer is flushed

Note:

Use include directive : if resource is not changing frequently

Use include action: if resource is changing frequently

Include - Action



Forward.jsp

```
<%
   if((request.getParameter(“un”).equals(“raj”))
  &&(request.getParameter(“pw”).equals(“java”)))
  {
%>
  <jsp:forward page=“forward2.jsp”/>
<%
    }
      else
    {
%>
  <%@include file=“index.jsp”%>
<%

  }
%>
```

Forward2.jsp

```
<html>
<head><title>usebean action test
 page</title></head>
<body>
  <h1> forward action test: login sucessful</h1>
<p>welcome<%=request.getParameter(“un”)%>
</body>
</html>
```

Using <jsp:param> to pass parameter while doing Request Dispaching…

```
html>
  <head>
    <title>Include (action) Example</title>
  </head>

  <body>
    <h2>Included part begins:<h2><hr/>
    <jsp:include page="/requestParams2.jsp" >
   <jsp:param name="sessionID" value="<%= session.getId() %>" />
    </jsp:include>
    <hr/><h2>Included part ends<h2>
  </body>
</html>
```

requestParams2.jsp

```
-----------------------------
<%@ page import="java.util.*" %>
<%
Enumeration parameterNames = request.getParameterNames();
while (parameterNames.hasMoreElements())
  {
  String name = (String)parameterNames.nextElement();
  out.println(name);
  out.println(request.getParameter(name) );
  }
%>
```

initialize servlet via JSP

```
<web-app ...>
  <servlet>
    <servlet-name>myTestJSPInit</servlet-name>
    <jsp-file>/TestInit.jsp</jsp-file>
    <init-param>
      <param-name>email</param-name>
      <param-value>rgupta.mtech@gmail.com</param-value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>myTestJSPInit</servlet-name>
    <url-pattern>/TestInit.jsp</url-pattern>
  </servlet-mapping>

</web-app>
```

ways to get the web.xml init parameter

```
<%-- override the jspInit() method %>
<%!
  public void jspInit() {

    ServletConfig sConfig = getServletConfig();
    String emailAddr = sConfig.getInitParameter("email");
    ServletContext ctx = getServletContext();
    ctx.setAttribute("mail", emailAddr);
  }
%>
```

Getting things in body

```
<%= "Mail Attribute is: " + application.getAttribute("mail") %>
<br>
<%= "Mail Attribute is: " + pageContext.findAttribute("mail") %>
<%
  ServletConfig sConfig = getServletConfig();
  String emailAddr = sConfig.getInitParameter("email");
  out.println("<br><br>Another way to get web.xml
 attributes: " + emailAddr );
%>
<%
  out.println(getServletConfig
 ().getInitParameter("email") );
%>
```

Using <jsp:useBean ....>

Servlet code:

```
foo.Person p = new foo.Person();
p.setName("Paul");
request.setAttribute("person", p);

RequestDispatcher view = request.getRequestDispatcher("result.jsp");
view.forward(request, response);
....
```

Retriving in JSP

```
<html><body>
  Person is: <%= request.getAttribute("person") %>
</body></html>
```

Correct way

- <% foo.Person p = (foo.Person) request.getAttribute("person");%>
- Person is: <%= p.getName() %>

Better way

```
<jsp:useBean id="person" class="foo.Person"
  scope="request" />
```

```
Person is: <jsp:getProperty name="person"
  property="name" />
```

- What if we wanted the reference type to be different from the actual object type

- in other words the Person class is an abstract class and make a concrete subclass called Employee

```
Person
 |
Employee
```

```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee"
  scope="page">
```

Expression Language

What is EL

- EL (Expression Language), it is now part of the JSP 2.0 spec.
- EL offers a simpler way to invoke Java code

- EL example

```
# Java Expression (old way don't do now)
Please contact: <%= application.getAttribute("mail")
%>
# EL way
please contact: ${applicationScope.mail}
```

Stoping all JSP pages from using any scripting elements

```
<web-app ...>
  ...
    <jsp-config>
      <jsp-property-group>
      <url-pattern>*.jsp</url-pattern>
      <scripting-invalid>true</scripting-invalid>
      <el-ignored>true</el-ignored>
      </jsp-property-group>
    </jsp-config>
  ...
  </web-app>
```

stop using EL

```
<%@ page isELIgnored="true" %>
```

Note: this takes priority over the DD tag above

How EL make life Easy

Consider this code in controller servlet

```
foo.Person p = new foo.Person();
p.setName("Paul");
foo.Dog dog = new foo.Dog();
dog.setName("Spike");
p.setDog(dog);
request.setAttribute("person", p);
```

using tags

```
<%= ((foo.Person) request.getAttribute("person")).getDog().getName()
%>
```

JSP Code using EL

```
<html><body>
Dog's name is: ${person.dog.name}
</body></html>
```

❖Servlet code

```
String[] footballTeams = { "Liverpool", "Manchester Utd", "Arsenal",
  "Chelsea" }
  request.setAttribute("footballList", footballTeams);
```

❖JSP Code

```
Favorite Team: ${footballList[0]}
  Worst Team: ${footballList["1"]}

  <%-- using the arraylist toString()
  All the teams: ${footballList}
```

❖Note: The index value in the brackets is a String literal

❖ which means it gets coerced into an int, which means ["one"] would not work but ["10"] would.

❖ Servlet code

```
java.util.Map foodMap = new java.util.HashMap();

  foodMap.put("Fruit", "Banana");
  foodMap.put("TakeAway", "Indian");
  foodMap.put("Drink", "Larger");
  foodMap.put("Dessert", "IceCream");
  foodMap.put("HotDrink", "Coffee");

  String[] foodTypes = {"Fruit", "TakeAway", "Drink", "Dessert",
  "HotDrink"}
  request.setAttribute("foodMap", foodMap);
```

❖ JSP Code

```
Favorite Hot Drink is: ${foodMap.HotDrink}

  Favorite Take-Away is: ${foodMap["TakeAway"]}

  Favorite Dessert is: ${foodMap[foodTypes[3]]}
```

Using paramValues

- Use **paramValues** when you want multiple values for one given parameter name

- Example

HTML Form

```
<html><body>

  <form action="TestBean.jsp">
    name: <input type="text" name="name">
    ID: <input type="text" name"empID">

    First food: <input type="text" name="food">
    Second food: <input type="text" name="food">

    <input type="submit">
  </form>

  </body></html>
```

❖JSP Code

Request param name is: **${param.name}**

Request param empID is: **${param.empID}**

<%-- you will only get the first value -->
Request param food is: **${param.food}**

First food is: **${paramValues.food[0]}**

Second food is: **${paramValues.food[1]}**

Getting other parameters with EL

- host header

  Host is: <%= request.getHeader("host) %>
  Host is: ${header["host"]}
  Host is: $header.host}

- Request method (Post or Get)

  Method is: ${pageContext.request.method}

- Cookie information

  Username is: ${cookie.userName.value}

- Context init parameter

  email is: <%= application.getInitParameter("mainEmail") %>

  email is: {$initParam.mainEmail}

  note: you need to configure the parameter in the DD

Quick recap…

Element Type	Example
directive	<%@ page import="java.util.*" %>
declaration	<%! int y = 3; %>
EL Expression	email: ${applicationScope.mail}
scriptlet	<% Float one = new Float(42.5); %>
expression	<%= pageContext.getAttribute(foo") %>
action	<jsp:include page="foo.html" />

Using JSTL

JSTL

- JSP 1.2 introduced supported for a special tag library called the JSP Standard Tag Library (JSTL)
- Version 1.0 released in June 2002
  Version 1.1 released in June 2004
- The JSTL saves programmers from having to develop custom tag libraries for a range of common tasks, such as if statements, conditional loops etc.
- Enables developers to produce more maintainable and simpler JSP code
- **Important development for JSP technology**

JSTL

- The JSP Standard Tag Library groups actions into four libraries as follows:

Library	Contents
Core	Core functions such as conditional processing and looping, important data from external environments etc
Formatting	Format and parse information
SQL	read and write relational database data
XMI	Processing of XML data

JSTL

- To use any of these libraries in a JSP, need to declare using the taglib directive in the JSP page, specifying the URI and the Prefix

Library	Prefix	URI
Core	c	http://java.sun.com/jsp/jstl/core
Formatting	fmt	http://java.sun.com/jsp/jstl/fmt
SQL	sql	http://java.sun.com/jsp/jstl/sql
XMI	xml	http://java.sun.com/jsp/jstl/xml

Example of declaring use of core library:

```
<%@ taglib  prefix = "c" uri = "http://java.sun.com/jsp/jstl/core %>
```

JSTL: Example

Example: JSP page using JSTL that outputs 1 to 10 on a webpage using the <c:forEach> and <c:out> tags of the **core library**

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
 <head>
  <title>Count to 10 Example (using JSTL)</title>
 </head>

 <body>
  <c:forEach var="i" begin="1" end="10" step="1">
   <c:out value="${i}" />

   <br />
  </c:forEach>
 </body>
</html>
```

A taglib directive declare use of core library

JSTL tag examples

JSTL: Example <c:foreach>

Looking more closely at <c:forEach tag> …..

```
<c:forEach var="i" begin="1" end="10" step="1">
   <c:out value="${i}" />

   <br />
  </c:forEach>
 </body>
</html>
```

The `<forEach>` tag enables loop logic. In this case, will look through 10 times. Equivalent to java "`for`" loop
Closing tag `<c:forEach>` placed after the end of body of loop

JSTL: Example <c:foreach>

All JSTL tags have a set of attributes (similar to HTML tags..)

e.g. <c:foreach> tag has 6 attributes:

```
var, items, varStatus,  begin, end, step
```

The full details for each attribute is in the JSTL specification document.

Willl need to use this document to verify WHICH tag should be used and HOW is should be used

JSTL: Example <c:out>

<c:out> .. outputs a value to webpage.

Usually uses just one attribute `value`

Examples:

```
<c:out value="${i}" />

<c:out value="The result of 1 + 2 is ${1+2}" />

<c:out value="param.userName" />
```

JSTL: Example <c:if>

<c:if> .. evaluates a condition. Uses an attribute `test` to hold the condition

Example :

```
<%-- Simple if conditions --%>
<c:if test='${param.p == "someValue"}'>
    Generate this template text if p equals
    someValue
</c:if>
```

Example 2

```
<c:if test='${param.p}'>
    Generate this template text if p equals "true"
</c:if>
```

JSTL: Multiple 'if' conditions

An if/else action requires the use of the <c:choose> tag

Syntax :

```
<c:choose>
   body content (<when> and <otherwise> subtags)
</c:choose>
```

JSTL: Multiple 'if' conditions

Uses <c:choose>, <c:when> and <c:otherwise>

Example:

```
<c:choose>
  <c:when test='${param.p == "0"}'>
      <c:out value = "zero recorded"/>
  </c:when>
  <c:when test='${param.p == "1"}'> Generate this
      <c:out value = "single value"/>
  </c:when>
  <c:otherwise>
    <c:out value = "Set to ${param.p}"/>
  </c:otherwise>
</c:choose>
```

JSTL: Other core <c:..> actions

Other examples: (NOT a complete list!)

```
<c:set>    ….sets the value of a variable
<c:remove> ….removes a scoped variable
<c:catch> ….catches an exception
<c:url> ….. encodes a URL
<c:import>… imports the content of a resource
<c:redirect>.. redirects to another URL
<c:param>.. adds a request parameter to other
actions
```

JSTL: <fmt:.....> example

Library	Prefix	URI
Core	c	http://java.sun.com/jsp/jstl/core
Formatting	**fmt**	**http://java.sun.com/jsp/jstl/fmt**
SQL	sql	http://java.sun.com/jsp/jstl/sql
XMI	xml	http://java.sun.com/jsp/jstl/xml

JSTL contains a set of actions in the Formatting library - these tags are useful for formatting numbers, times and dates

e.g. `<fmt:parseDate>`

JSTL: <fmt:parseDate> example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt"
prefix="fmt" %>


<html>  etc etc

<fmt:parseDate value= ${param.empDate}"  var =
"parsedEmpDate" pattern = "yyyy-MM-dd" />

etc etc

</html>
```

The fmt:parseDate action takes the date or time string specified by the value attribute (e.g. 2001-09-28) , interprets it according to the pattern defined by the pattern attribute and saves it in a variable called "parsedEmpDate"

JSTL: other <fmt> actions

Other examples:

`<fmt:formatNumber>` – formats a numeric value
e.g. number of digits, currency, decimal place
e.g. `<fmt:formatNumber value="12.3" pattern=".000"/>`
will output “12.300”

`<fmt:formatDate>` --formats a date and time

JSTL: Expression language

- Up to now, could only use Java expressions to assign dynamic values → syntax errors common
- JSTL now provides an expression language (EL) to support the tags → simpler syntax, less errors
- The EL is now part of the JSP specification (as of versions JSP 2.0) - can be used in JSTL tags or directly in JSP pages.

JSTL: Expression language

- All EL expressions are evaluated at runtime
- The EL usually handles data type conversion and null values --→ easy to use
- An EL expression always starts with a ${ and ends with a }

JSTL: Expression language

- The expression can include
  - literals (“1”, “100” etc)
  - variables
  - implicit variables

Examples:

```
<c:out value  = “${1+2+3}” />
```

expression

```
<c:if test = “${param.Address == “D6”}” />
```

JSTL: Expression language - operators

==	>	+
!=	<=	-
<	>=	*
		/ or div

- Logical operators consist of &&, ||, and !
- The **empty** operator is a prefix operator that can used to determine if a value is null or empty. For example:

```
<c:if test="${empty param.name}">
     Please specify your name.
</c:if>
```

JSP Implicit objects

- In JSP, need to be able to access information about the environment in which the page is running e.g. the parameters passed in a request for a form, the browser type of the user, etc.

- Implicit objects are a set of Java objects that the JSP Container makes available to developers in each page. These objects may be accessed as built-in variables via scripting elements

JSTL implicit variables

- The JSTL EL allows these objects to be accessed as 'Implicit Variables'
- Implicit variable are just pre-agreed fixed variable names that can be used in JSTL Expressions
-
- --→ Think of as "variables that are automatically available to your JSP page"..

JSTL: Expression language – Implicit Objects

- Very common implicit object is `param`
- `param` refers to parameter passed in a request message (e.g. information entered into a form by a user).
- e.g. `<c:out value = "${param.userName}"/>`
- Further Examples of using `param` in next topic

Comparing JSTL and Scriptlets

JSTL removes complexity by using tags instead of java code (abstraction)

JSP pages using JSTL usually easier to maintain

JSTL allows HTML 'tag' developers to 'program'

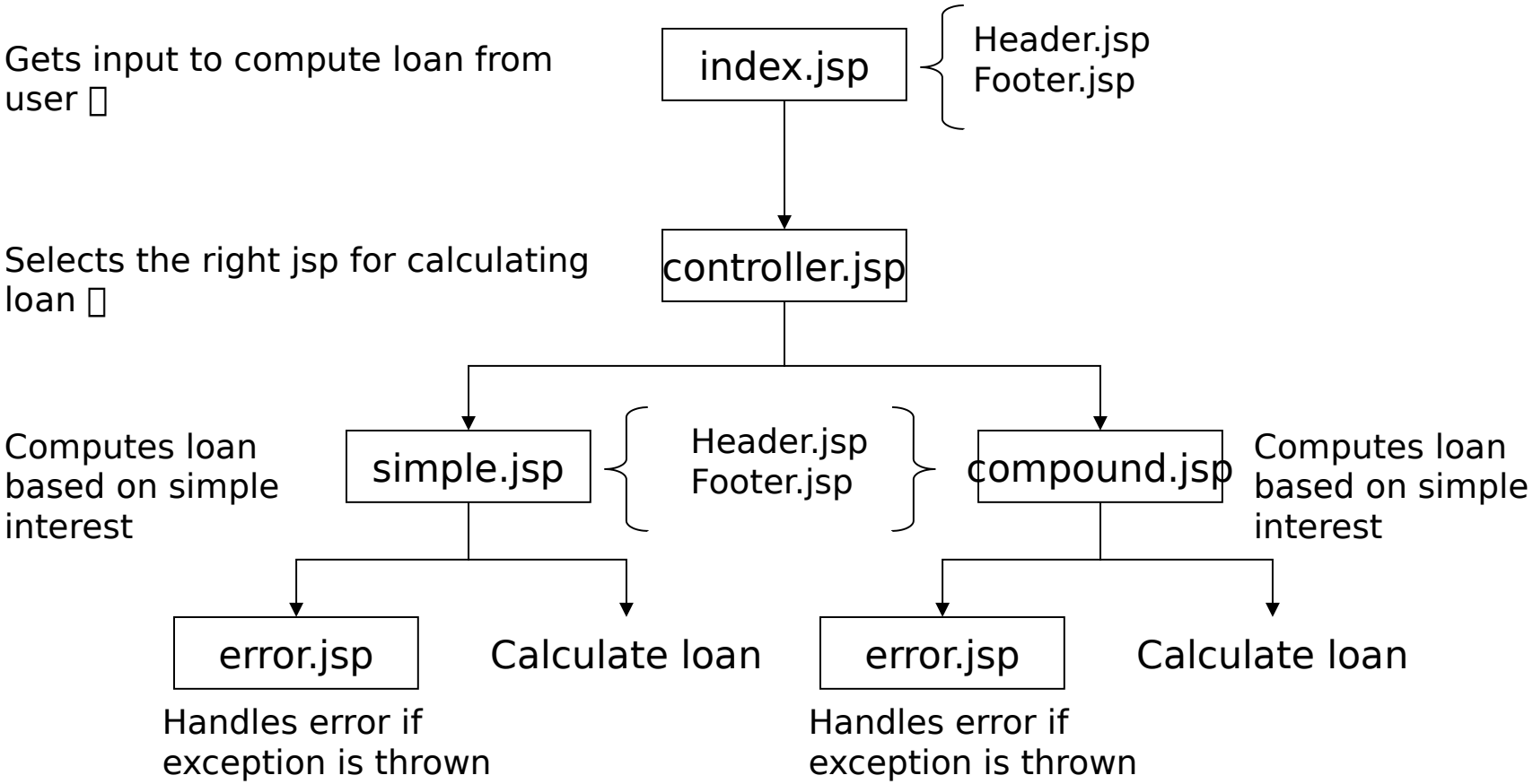
JSTL often more difficult to debug

Note: Using JSTL *does not* eliminate scriplets entirely.. may still need them sometimes for more complex logic

Loan Calculator application

Example

Loan Calculator



Loan Calculator

index.jsp

```
<html>
  <head>
    <title>Include</title>
  </head>
  <body style="font-family:verdana;font-size:10pt;">
    <%@ include file="header.html" %>
    <form action="controller.jsp">
      <table border="0" style="font-family:verdana;font-
            size:10pt;">
        <tr>
          <td>Amount:</td>
          <td><input type="text" name="amount" />
        </tr>
        <tr>
          <td>Interest in %:</td>
          <td><input type="text" name="interest"/></td>
        </tr>
        <tr>
          <td>Compound:</td>
          <td><input type="radio" name="type" value="C"
              checked/></td>
        </tr>
        <tr>
          <td>Simple:</td>
          <td><input type="radio" name="type"
            value="S" /></td>
        </tr>
        <tr>
          <td>Period:</td>
          <td><input type="text"
            name="period"/></td>
        </tr>
      </table>
      <input type="submit" value="Calculate"/>
    </form>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

Loan Calculator

Miscelaneous

controller.jsp

```
<%
  String type = request.getParameter("type");
  if(type.equals("S")) {
%>
<jsp:forward page="/simple.jsp"/>
<%
  } else {
%>
 <jsp:forward page="/compound.jsp"/>
<%
  }
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<html>
  <head>
    <title>Simple</title>
  </head>
  <body style="font-family:verdana;font-
        size:10pt;">
    <%@ include file="header.html" %>
    <p style="color=#FF0000"><b><%=
        exception.getMessage() %></b></p>
    <jsp:include page="footer.jsp"/>
  </body>
</html>
```

header.jsp

```
<h3>Loan Calculator</h3>
```

footer.jsp

```
<%= new java.util.Date() %>
```

Loan Calculator

simple.jsp

```
<%@ page errorPage="error.jsp" %>
<%!
 public double calculate(double amount, double
        interest, int period) {
  if(amount <= 0) {
   throw new IllegalArgumentException("Amount
        should be greater than 0: " + amount);
  }
  if(interest <= 0) {
   throw new IllegalArgumentException("Interest
        should be greater than 0: " + interest);
  }
  if(period <= 0) {
   throw new IllegalArgumentException("Period should
        be greater than 0: " + period);
  }
  return amount*(1 + period*interest/100);
 }
%>
```

```
<html>
 <head>
  <title>Simple</title>
 </head>
 <body style="font-family:verdana;font-size:10pt;">
  <%@ include file="header.html" %>
  <%
   double amount =
        Double.parseDouble(request.getParameter("a
        mount"));
   double interest =
        Double.parseDouble(request.getParameter("int
        erest"));
   int period =
        Integer.parseInt(request.getParameter("period"
        ));
  %>
  <b>Pincipal using simple interest:</b>
  <%= calculate(amount, interest, period) %>
  <br/><br/>
  <jsp:include page="footer.jsp"/>
 </body>
</html>
```

Loan Calculator

compound.jsp

```
<%@ page errorPage="error.jsp" %>
<%!
 public double calculate(double amount, double
        interest, int period) {
  if(amount <= 0) {
   throw new IllegalArgumentException("Amount
        should be greater than 0: " + amount);
  }
  if(interest <= 0) {
   throw new IllegalArgumentException("Interest
        should be greater than 0: " + interest);
  }
  if(period <= 0) {
   throw new IllegalArgumentException("Period should
        be greater than 0: " + period);
  }
  return amount*Math.pow(1 + interest/100, period);
 }
%>
```

```
<html>
 <head>
  <title>Compound</title>
 </head>
 <body style="font-family:verdana;font-size:10pt;">
  <%@ include file="header.html" %>
  <%
   double amount =
        Double.parseDouble(request.getParameter("a
        mount"));
   double interest =
        Double.parseDouble(request.getParameter("in
        terest"));
   int period =
        Integer.parseInt(request.getParameter("period
        "));
  %>
  <b>Pincipal using compound interest:</b>
  <%= calculate(amount, interest, period) %>
  <br/><br/>
  <jsp:include page="footer.jsp"/>
 </body>
</html>
```

Any questions?