

Build COMPETENCY
across your TEAM



Microsoft Partner

Gold Cloud Platform
Silver Learning

Day 14,15,16. Servlet JSP



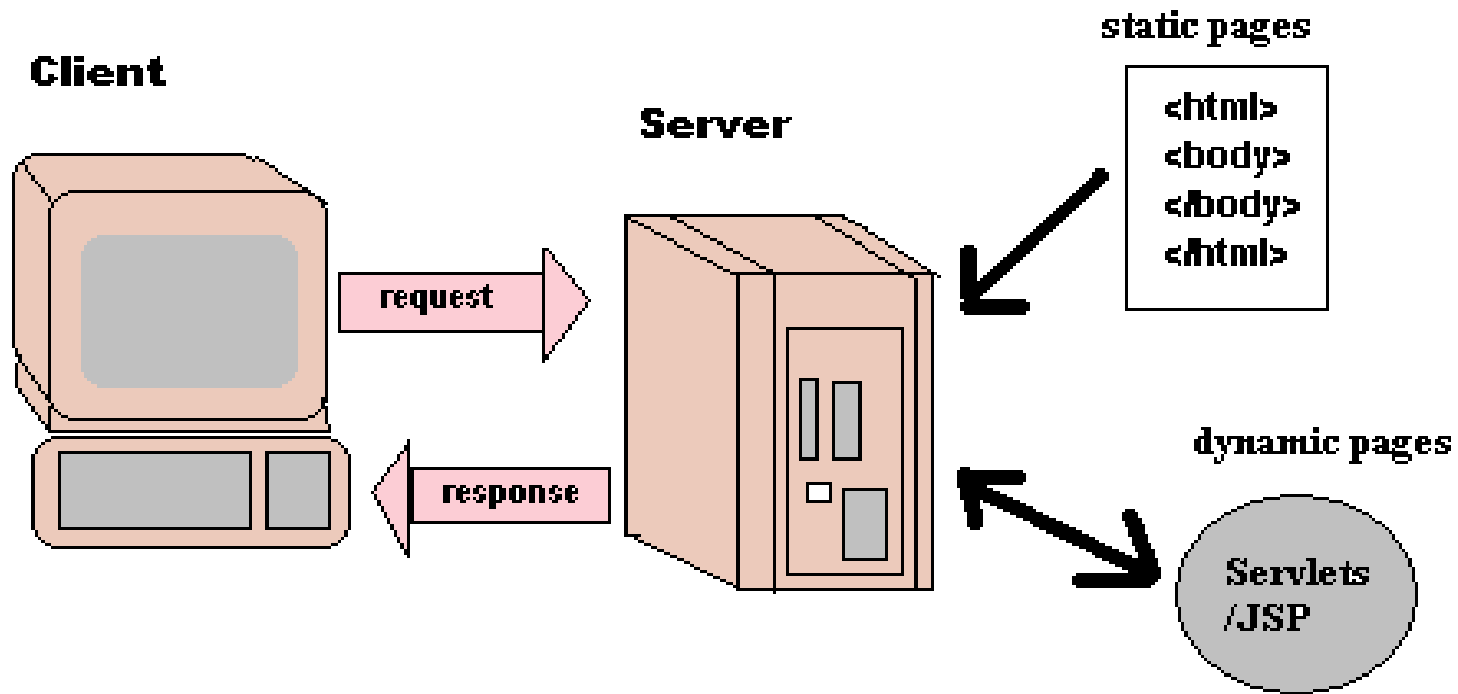
Smita B Kumar



Servlet - Topics Overview

- Introduction to Servlet Technology
- Setting up Servlet programming environment
- Writing and deploying first Servlet
- Role of web server

Client Server Technology



Need of Dynamic Web Page Building

- Data on web page depends on the client request
 - Ex : Search engines
- Data on web pages changes frequently
 - Ex : Weather reports
- Data on web pages uses data from corporate databases
 - Ex : Stock indexes

A Static HTML Page

```
<html>
  <head>
    <title>
      Displaying Date
    </title>
  </head>
  <body>
    <script language="java script">
      <!-- document.write(Date( )); -->
    </script>
  </body>
</html>
```



Writing the First Servlet “GoodMorning.java”

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;
```

```
public class GoodMorning extends HttpServlet {  
    protected void doGet(HttpServletRequest request , HttpServletResponse response)  
        throws IOException {  
        response.setContentType( “text/html” );  
        PrintWriter out = response.getWriter();  
        out.println( “GOOD MORNING EVERYBODY !!!! ” );  
    }  
}
```

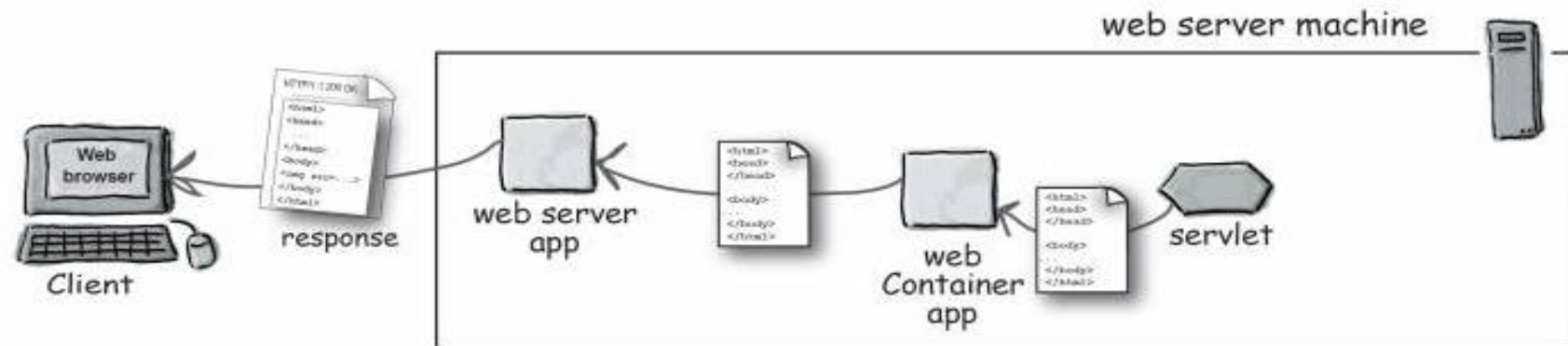
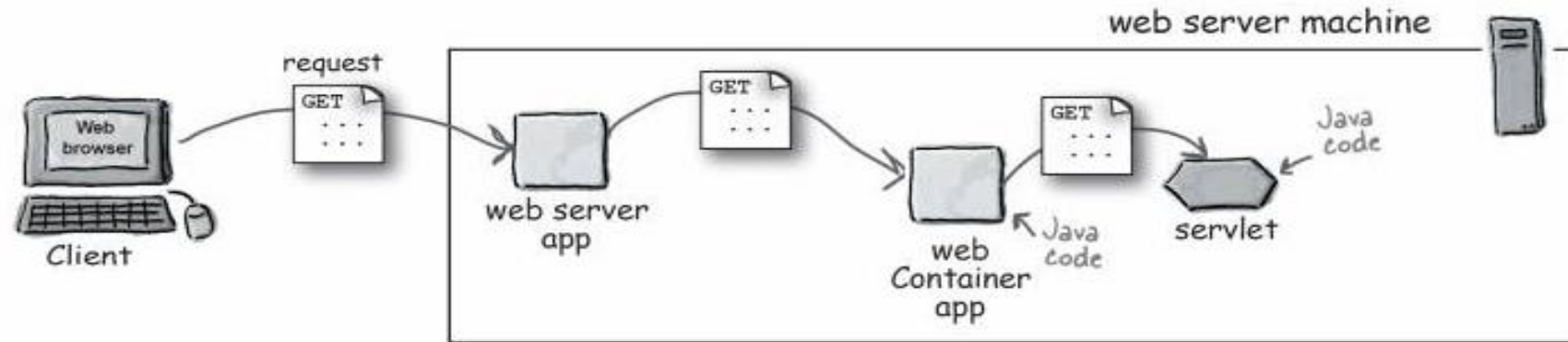
Other Competing Technologies

- HTML (HyperText Markup Language)
- CGI (Common Gateway Interface)
- ASP (Active Server Pages)
- JavaScript
- PHP (HyperText Pre-Processor)
- ColdFusion

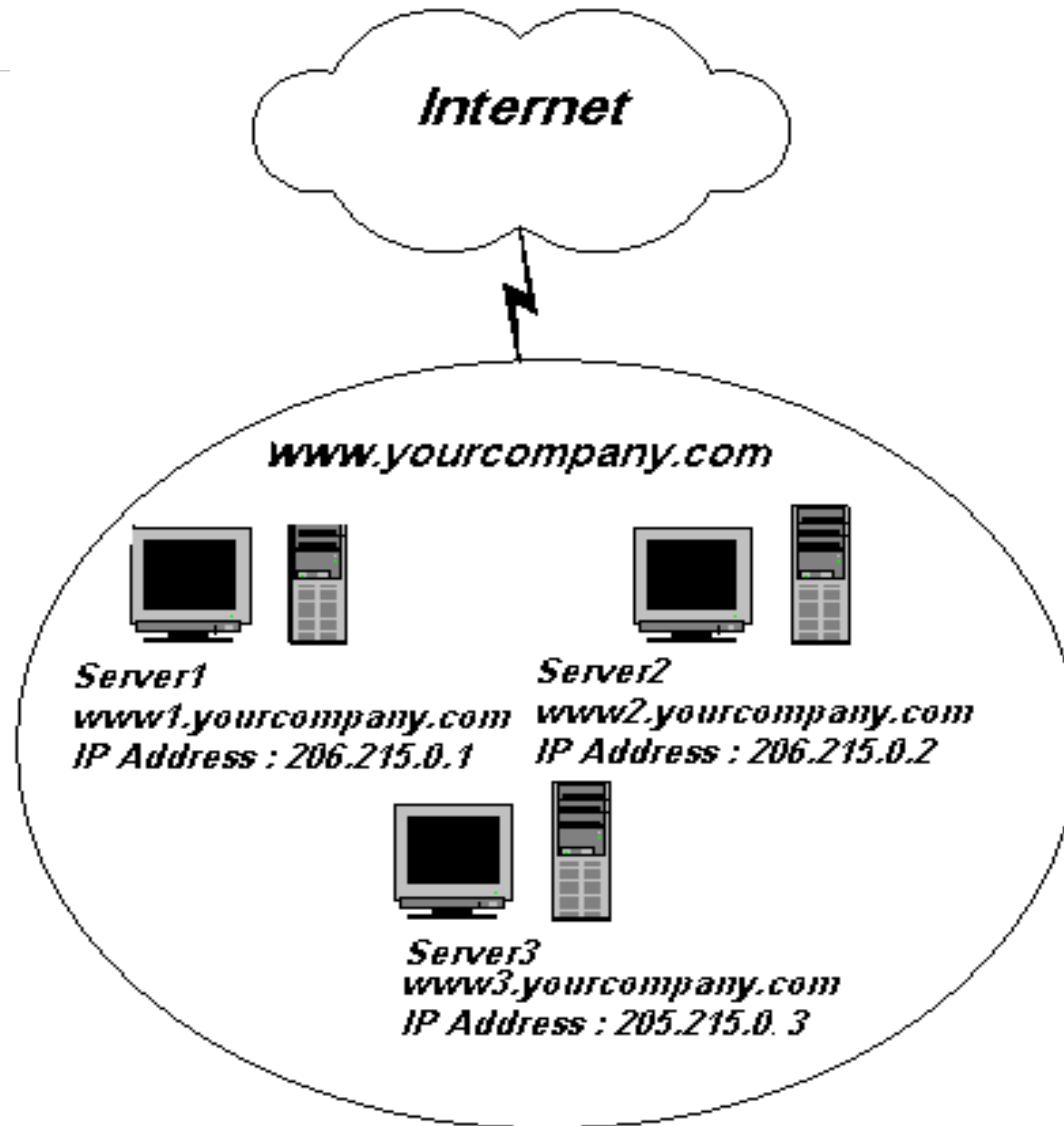
Servlets Features

- Portability
- Powerful
- Efficiency
- Safety
- Integration
- Extensibility
- Inexpensive

Role of a Server

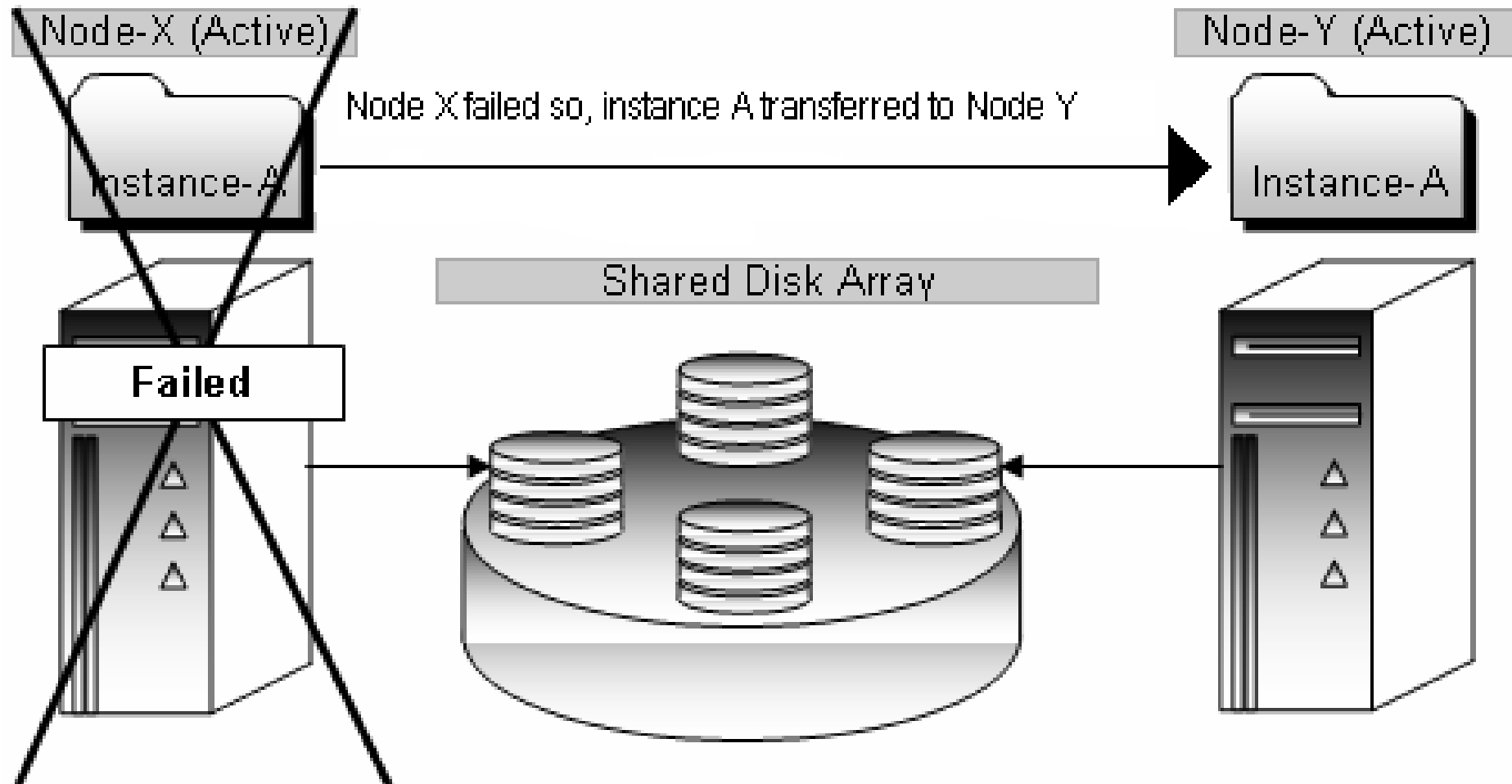


Features of Server



**Web Server Cluster for
www.yourcompany.com**

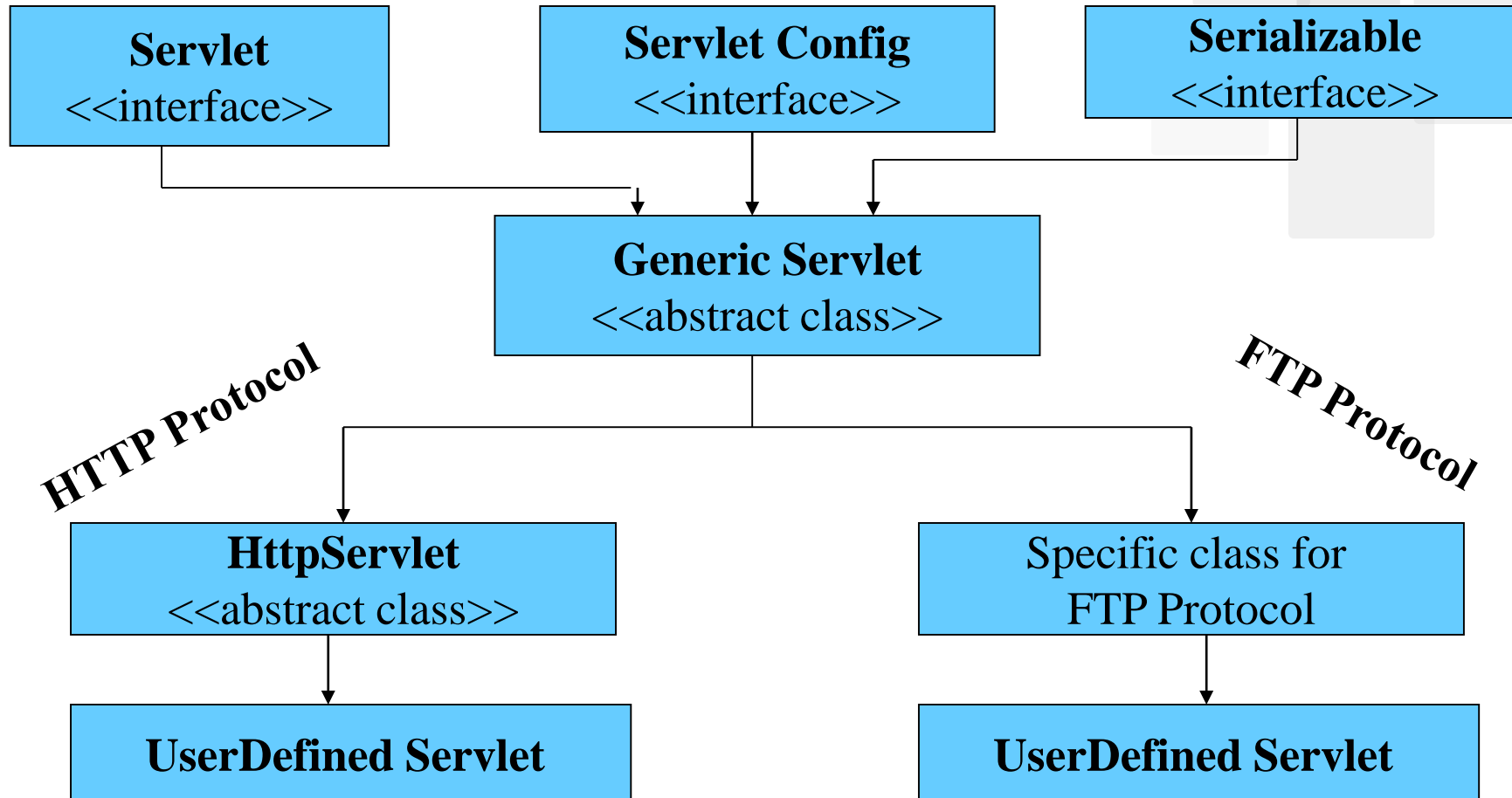
Features of Server (contd...)



Web Servers

- Tomcat (Apache)
- Weblogic (Bea)
- Microsoft Internet Information Server (IIS)
- Websphere (IBM)
- WebStar (StarNine)
- Java Web Server (Sun's)

Servlet Class Hierarchy



Servlet Architecture contd...

Directory Structure :

- **Working Directory**
 - html
 - jsp
 - **WEB-INF**
 - classes
 - lib
 - src

Servlet Architecture contd...

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;
```

```
public class GoodMorning extends HttpServlet {  
    protected void doGet(HttpServletRequest request , HttpServletResponse response) throws IOException {  
        response.setContentType( "text/html" );  
        PrintWriter out = response.getWriter();  
        out.println( "GoodMorning .....Welcome to the Wrold of Servlet!!!!" );  
    }  
  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {  
        doGet( request, response);  
    }  
}
```

Deployment Descriptor and Deployment(web.xml)

1. Write and add web.xml in WEB-INF.

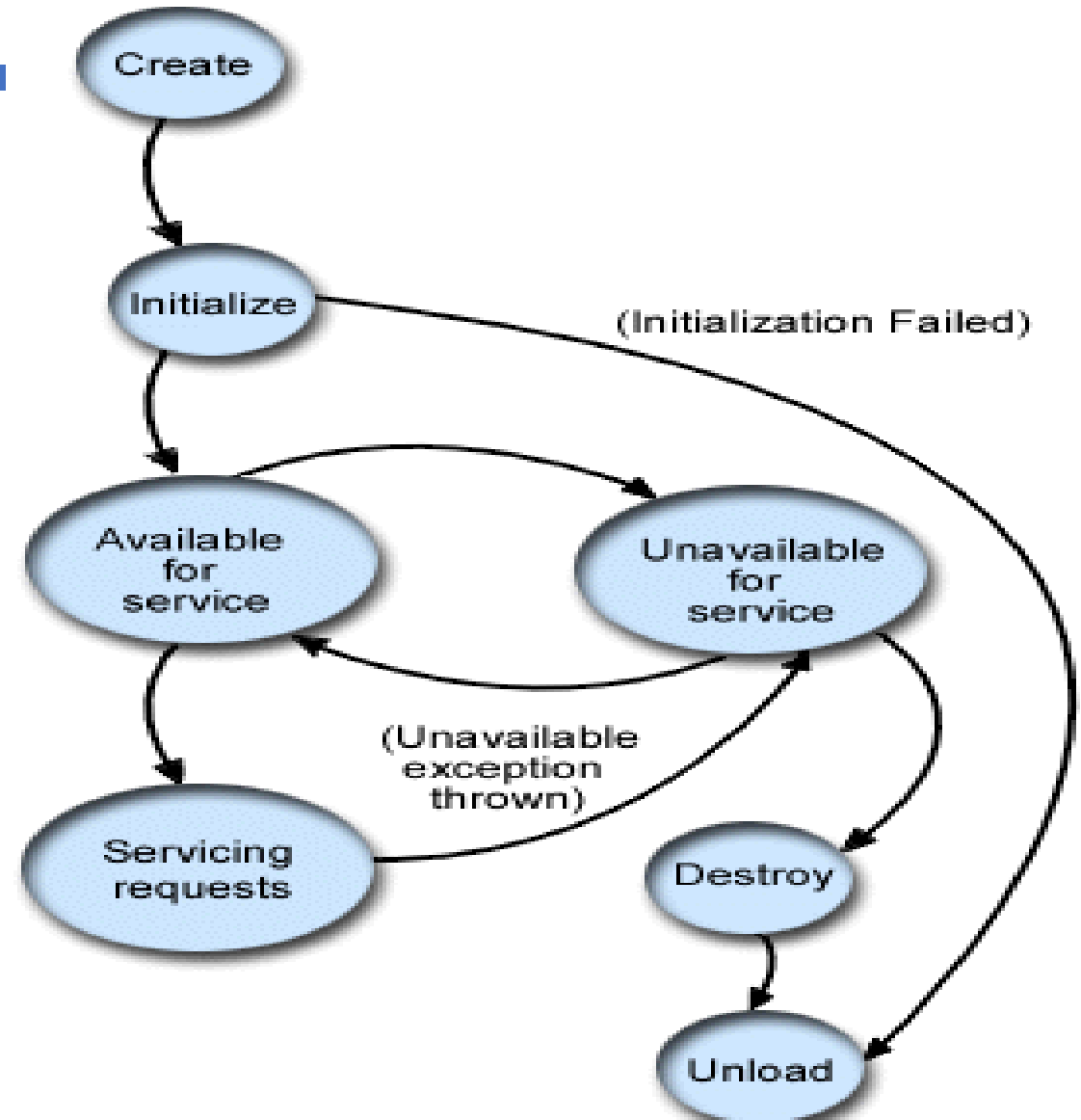
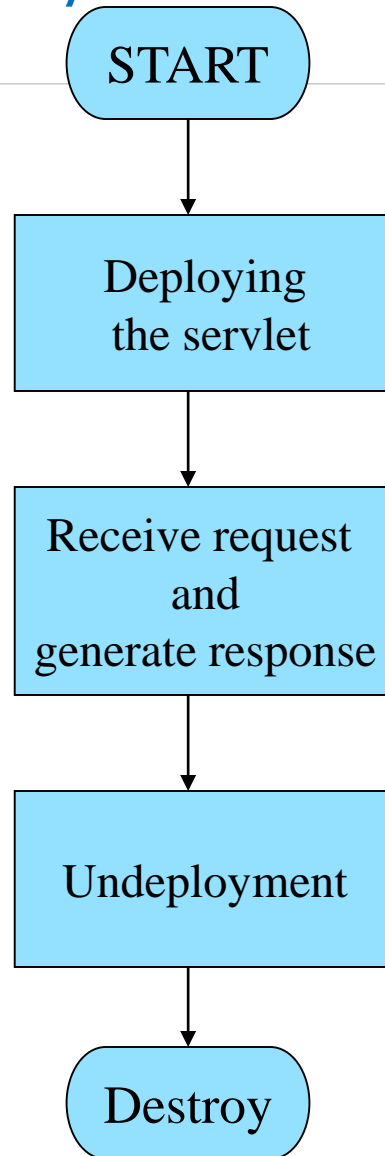
```
<web-app>
  <servlet>
    <servlet-name>GOODMORNING</servlet-name>
    <servlet-class>Module1.packwelcome.GoodMorning</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>GOODMORNING</servlet-name>
    <url-pattern>/goodmorning</url-pattern>
  </servlet-mapping>
</web-app>
```

2. Optional : Convert an application into a WAR file.
3. Mandatory : Deploy application on the server.

Servlet- The Servlet Life Cycle - Topics Overview

- Servlet Lifecycle
- `init()`, `service()` and `destroy()` methods
- The `doGet()` and `doPost()` methods
- The servlet packaging

Servlet Life Cycle



Servlet Life Cycle In Detail

what happens when u say

http://localhost:8080/myapp/ms1

- a) request goes to web container
- b) web container takes url pattern `"/ms1"`
- c) it will open ur `"web.xml"` file
- d) it will check `"url-pattern"` with the name `"/ms1"`
- e) from `"url-pattern"`, it will get `"servlet-name"`
- f) from `"servlet-name"`, it will get `"servlet-class"`.
- g) it will search `"MyServ1.class"` in `"classes"` folder.
- h) if found , it will load `"MyServ1.class"`
- i) instantiate it by invoking `"public no-arg constructor."` (it is compulsory that ur servlet class must be having `"public no-arg constructor"`).
- j) container will now call `"init()"` method
- k) thread will be created or retrieved from the thread pool.

- i) `HttpServletRequest` and `HttpServletResponse` will be created.
- j) `service()` method is called.
- k) `service()` method calls either `"doGet()"` (if request is `"get"`) or `"doPost()"` (if request is `"post"`).
- l) thread will be destroyed or put back into the thread pool.

when u refresh the page (i.e. ur giving second request to the servlet)

- a) thread will be created or retrieved from the thread pool.
- b) `HttpServletRequest` and `HttpServletResponse` will be created.
- c) `service()` method is called.
- d) `service()` method calls either `"doGet()"` (if request is `"get"`) or `"doPost()"` (if request is `"post"`).
- e) thread will be destroyed or put back into the thread pool.

The init() method

```
public void init( ) throws ServletException{
    f= new File("C:/Count.txt");
    FileInputStream fi=null;
    try {
        if (f.exists( )){
            fi = new FileInputStream(f);
            hitCount = fi.read( );
            fi.close( );
        }
        else
            hitCount = 0;
    }
    catch(IOException ie){
        throw new ServletException("Input file corrupted.", ie);
    }
}
```

The service() method

```
protected void doGet ( HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {
```

```
    System.out.println("Servlet receiving subsequent requests" );
```

```
    PrintWriter out = response.getWriter();
```

```
    out.println(" synergetics-india Software Private Limited" );
```

```
    out.println(" This website is hit "+(hitCount++)+" no. of times." );
```

```
}
```

```
protected void doPost (HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException{
```

```
    doGet(arg0, arg1);
```

```
}
```

The destroy() method

```
public void destroy(){
    System.out.println( " Object being undeployed from server.." ) ;
    try
    {
        FileOutputStream fo = new FileOutputStream(f);
        fo.write(hitCount);
        fo.close();
    }
    catch(IOException io){
        io.printStackTrace();
    }
}
```

Servlet Packaging

Various ways in which Packaging can be done.....

- .jar file : Java Archive
- .war file : Web Archive
- .ear file : Enterprise Archive

FormData, Config and Context Parameters - Topics Overview

- The Form Data
- Collecting form data for different HTML components
- Roll of XML descriptor
- The 'context' and 'config' parameters
- The Context workspace

The Form Data

The HTML elements

- The FORM ACTION tag
- The INPUT like component tags
- The SUBMIT button tag

The method types

- The GET type of method

`http://localhost:7001/CollectParameters/collect?fname=abc&sname=xyz`

- The POST type of method

The form data is sent on the separate line

Different HTML Components

Creating Form

<FORM ACTION = "/CollectParameters22/param" METHOD = "GET">

Creating Text Box

First Name : <INPUT TYPE="TEXT" NAME = "fname" ">

Creating Password Box

PassWord : <INPUT TYPE="PASSWORD" NAME = "pass">

Creating Radio Buttons

Married<INPUT TYPE="RADIO" NAME="marital" VALUE="Married">

Unmarried<INPUT TYPE="RADIO" NAME="marital" VALUE="Unmarried">

Creating Text Area

<TEXTAREA name='address' COLS="50" ROWS="4"></TEXTAREA>

Different HTML Components (Contd...)

Creating Combo Box

```
<SELECT NAME="city" SIZE="1">  
    <OPTION VALUE="Mumbai">Mumbai</ OPTION >  
    < OPTION VALUE ="Pune">Pune</ OPTION >  
    < OPTION VALUE ="Nasik">Nasik</ OPTION >  
</Select>
```

Creating Check Box

```
Times <INPUT TYPE="CHECKBOX" NAME="news" VALUE="Times ">  
DNA   <INPUT TYPE="CHECKBOX" NAME="news" VALUE="DNA">
```

Creating Submit Button

```
<INPUT TYPE="RESET" VALUE = "RESET">  
<INPUT TYPE="SUBMIT" NAME="choice" VALUE = "Tie">  
<INPUT TYPE="SUBMIT" NAME="choice" VALUE = "Necklace">
```

The HttpServletRequest

The ServletRequest methods :

- String Request.getParameter(String)
- String [] request.getParameterValues(String)
- Enumeration request.getParameterNames()
- BufferedReader request.getReader()
- ServletInputStream request.getInputStream()

The HttpServletRequest methods :

- String getHeader (String)
- Enumeration getHeaderNames()
- String getMethod()

The HttpServletResponse

The Servlet Response methods :

- PrintWriter getWriter()
- void setContentType(String type)
- void setBufferSize(int size)
- ServletOutputStream getOutputStream()

The HttpServletResponse methods :

- void addHeader(String name, String value)
- void setHeader(String name, String value)
- void setStatus(int code)

Collecting Form Data

```
private void doPost(HttpServletRequest arg0, HttpServletResponse arg1) {  
    String[] firstName = arg0.getParameter("fname");           // Text box  
    String surName = arg0.getParameter("sname");               // Text box  
    String passWord = arg0.getParameter("pass");               // Password box  
    String maritalStatus = arg0.getParameter("marital");        // Radio Buttons  
    String yourAddress = arg0.getParameter("address");          // Text Area  
    String yourCity = arg0.getParameter("city");                // Combo box  
    String yourChoice = arg0.getParameter("choice");            // Submit Button  
  
    String [ ]news = arg0.getParameterValues("news");          // Check boxes  
  
    if (news != null){  
        out.println("<TR><TD>News Paper/s");  
        out.println("<TD>"+news[0]);  
        for(int i=1; i<news.length; i++){  
            out.println(" & "+news[i]);  
        }  
    }  
}
```

Collecting Form data contd...

```
Enumeration e = arg0.getParameterNames( );  
while(e.hasMoreElements( )){  
    String field = (String)e.nextElement( );  
    String value = arg0.getParameter(field);  
    out.println(field+" : "+value+"<BR>");  
}
```

The ServletContext

- One per application
- Facilitates communication between server and the servlet
- Provides access to resources and facilities common to all servlets and JSPs in the application
- `getServletContext()` returns a reference to the ServletContext

The ServletConfig

- One per Servlet
- Used to store information specific to a particular servlet
- Carries servlet specific data, which is not accessible to any other servlet
- `getServletConfig ()` returns the reference to the ServletConfig

Initializing Parameters

<!-- Declaring Context Parameters -->

<context-param>

<param-name>drv</param-name>

<param-value>oracle.jdbc.driver.OracleDriver</param-value>

</context-param>

<!-- Declaring Servlet Tag -->

<servlet>

<servlet-name>BankEntry</servlet-name>

<servlet-class>packbank02.BankEntry</servlet-class>

<!-- Declaring Config Parameters -->

<init-param>

<param-name>user</param-name>

<param-value>scott</param-value>

</init-param>

</servlet>

Accessing initial parameters and other server side information.

- `getInitParameter()`
- `getInitParameterNames()`
- `getMajorVersion()`
- `getMinorVersion()`
- `getServerInfo()`

To access server side file resources.

- `getMimeType()`
- `getResourceAsStream()`
- `getRequestDispatcher(String path)`

Handling server side log

- `log()`

ServletContext methods contd...

Accessing context workspace

- `setAttribute()`
- `getAttribute()`
- `getAttributeNames()`
- `removeAttribute()`

ServletConfig methods

Obtaining Config Parameters

- `getInitParameter()`
- `getInitParameterNames()`

Obtaining Context reference

- `getServletContext()`

Obtaining Servlet Name

- `getServletName()`

ServletConfig v/s ServletContext

- **Accessibility :**

- **ServletContext :** One per Application , Context parameters are available across servlets under same application.
- **ServletConfig :** One per servlet. The config parameters are private to the servlet and cannot be accessed by any other servlet.

- **Getting the parameter values :**

- **ServletContext sct = this.getServletContext();**
String driverName = sct.getInitParameter("drv");
- **ServletConfig sc = this.getServletConfig();**
String passwd = sc.getInitParameter("pass");

- **Setting attributes :**

- **ServletConfig has only Parameters.**
 - Cannot set the config parameters via methods hence, only getter methods are available.
- **ServletContext has both Paramters and Attributes.**
 - Context Parameters can be set via the setter methods provided

Analysing Request Header - Topics Overview

- The GET and POST requests
- HTTP Request Parameter
- The Request Headers
- Analyzing request header

Typical Http Request

For GET method :

GET /requestheadersdemo?name=abc&surname=xyz http/1.1
Host : <http://localhost:8081/Myervlets/collectparameters>
User-Agent: Mozilla/4.0
Accept : */*
Accept Encoding : gzip,deflate

For POST method :

POST /requestheadersdemo http/1.1
Host : <http://localhost:8081/Myervlets/collectparameters>
User-Agent: Mozilla/4.0
Accept : */*
Accept Encoding : gzip,deflate
name=abc&surname=xyz

Reading Request Header

General :

getHeader("HeaderName")

getHeaderNames()

Specialized :

getCookies()

getRemoteUser()

getContentLength()

getDateHeader()

getIntHeader()

Related Info :

getMethod()

getRequestURI()

getQueryString()

getProtocol()

Analysing Request Header

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) {  
    PrintWriter out = response.getWriter( );  
    out.println("<B>Request Method : </B>" + request.getMethod( ) + "<BR>");  
    out.println("<B>Request Protocol : </B>" + request.getProtocol( ) + "<BR>");  
    out.println("<B>Request URI : </B>" + request.getRequestURI( ) + "<BR>");  
  
    Enumeration headerNames = request.getHeaderNames( ); while(headerNames.hasMoreElements( )){  
        String headerName =(String)headerNames.nextElement( );  
        out.println("<TR><TD>" + headerName);  
        out.println("    <TD>" + request.getHeader(headerName));  
    }  
}
```

Module 5 : Setting Response Header

Overview :

Typical HttpResponse - Topics Overview

- HTTP Response Header
- Setting MIME types
- Status Code, Refreshing, Sending Error Page, Logging in
- Setting Encoding Type

Setting Response Header

```
import javax.servlet.*;
import javax.servlet.http.*;
public class ResponseHeaders extends HttpServlet{
    public void doGet(HttpServletRequest req , HttpServletResponse res)throws ServletException , IOException{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter( );
        res.addHeader("MyHeader" , "This is My custom header");
        out.println("<html>");
        out.println("<body>");
        out.println("My Custom Header Set");
        out.println("</body>");
        out.println("<html>");
    }
}
```

Various Response Headers

- Allow
- Cache-Control
- Connection
- Content-Encoding
- Content-Language
- Content-Length
- Content-Type
- Refresh

Setting MIME type

Method to set MIME type :

setContentTypes(Mime type)

List of all mime types :

- text/plain : Plain text
- text/html : HTML document
- text/xml : XML document
- audio/basic : Sound file in .snd format
- image/gif : GIF image
- image/jpeg : JPEG image
- application/msword : Microsoft Word Document
- application/pdf : Acrobat (.pdf) file .
- application/x-java-archive : JAR file
- video/mpeg : MPEG video clip

Setting Encoding Type

```
public class Gzipping extends HttpServlet {
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {
        String encodings = arg0.getHeader("Accept-Encoding");
        PrintWriter out;
        String title;
        arg1.setContentType("text/plain");
        if ((encodings != null) && (encodings.indexOf("gzip") != -1)) {
            title = "Page Encoded with GZip";
            OutputStream out1 = arg1.getOutputStream();
            out = new PrintWriter(new GZIPOutputStream(out1), false);
            arg1.setHeader("Content-Encoding", "gzip");
        }
        else {
            title = "Un-encoded Page.";
            out = arg1.getWriter();
        }
        for(int i = 0; i < 10000; i++)
            out.println("-foo-faa");
        out.close();
    }
}
```


The Redirection - Topics Overview

- Redirection
- Redirection by setting response header
- Linking to another websites

Redirection in HTML

```
public class Redirect extends HttpServlet{
    protected void doGet(HttpServletRequest request, HttpServletResponse response)throws ServletException,
    IOException{
        String url1 = "http://localhost:8081/MyServlet/readword";
        String url2 = "http://www.synergetics-india.com";
        String url3 = "/MyServlet/goodmorning";

        // HTML Hyper Link Way of Redirection
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><head><title>301 Moved Permanently</title></head>");
        out.println("<body>");
        out.println("Website moved to <a href=\""+url1+"\">here.</a>");
        out.println("</body></html>");
        out.close();
    }
}
```

Redirection using Servlet API

```
protected void doGet(HttpServletRequest request , HttpServletResponse response)  
    throws ServletException, IOException{
```

```
    PrintWriter out = res.getWriter();  
    out.println("<B><CENTER> *****WELCOME*****");
```

```
// Redirection by specifying the absolute path of the site to whom the request is directed  
    res.sendRedirect(url2);
```

```
// Redirection by specifying the relative path of the site to whom the request is directed  
    res.sendRedirect(url3);
```

```
// Redirection by setting response header  
    res.setStatus(HttpServletResponse.SC_MOVED_TEMPORARILY);  
    res.setHeader("Location", url1);  
}
```

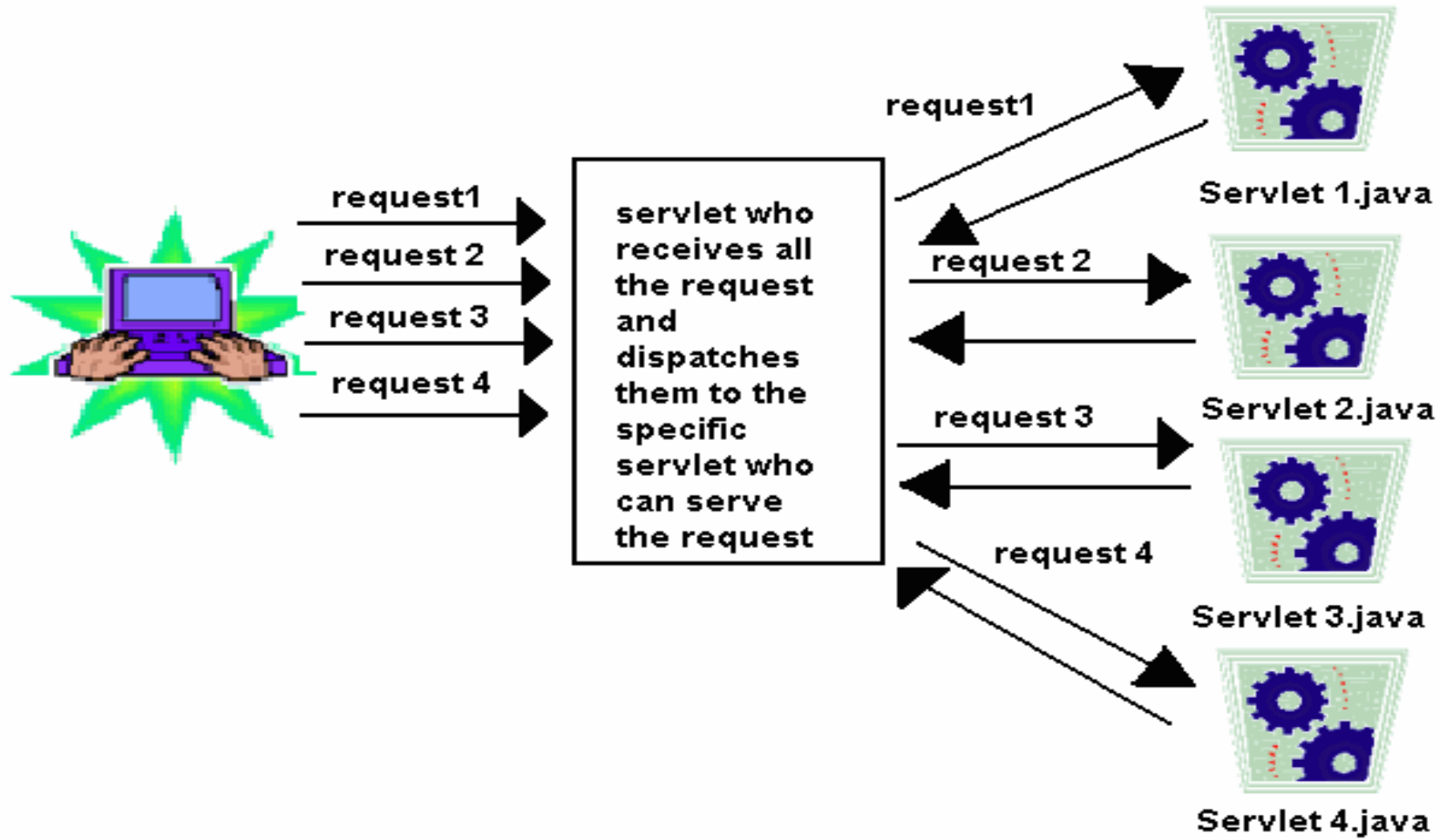
Setting the Refresh Header

```
public class RefreshHeader extends HttpServlet {  
    String url = "www.synergetics-india.com";  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
        PrintWriter out = response.getWriter();  
        out.println("<H1>HELLO !!!!!<H1>");  
  
        //The Browser asks for an updated page after 30 seconds  
        response.setIntHeader("Refresh",10);  
  
        //The browser goes to the specified page after specified time delay  
        response.setHeader("refresh","5; URL=http://www.synergetics-india.com");  
    }  
}
```

The Request Dispatching - Topics Overview

- The Request Dispatcher
- The forward and include requests
- Sending request between servlets
- The request workspace

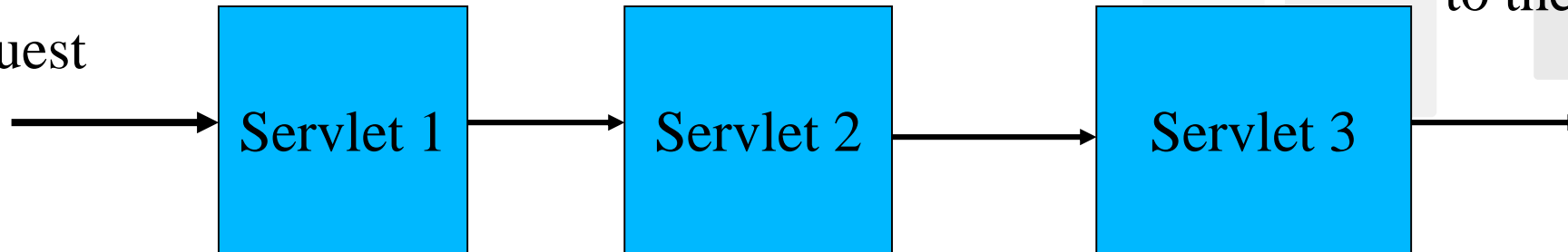
Request Dispatcher



Dispatching using forward() method

The request is received by one servlet and response is generated by another.

Client
request



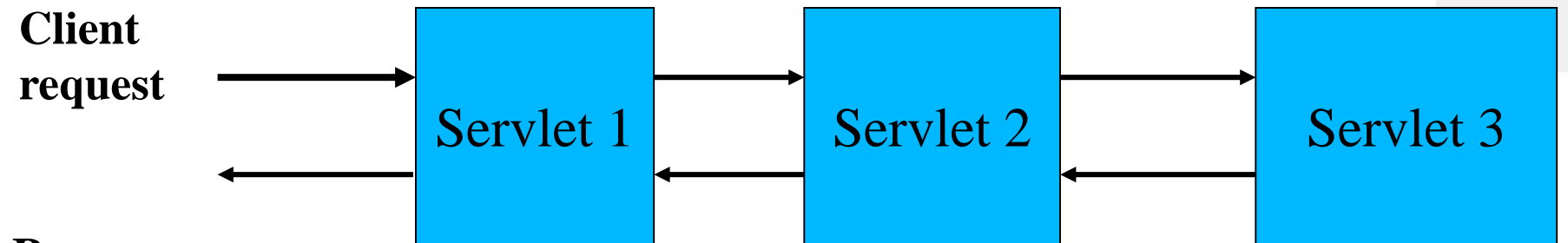
Receives the request and
dispatches it to other servlet

It may manipulate the
request coming from
servlet1 and propagate
it to next servlet. It
will overwrite the
response generated by
servlet1

Generates a
response for the
client. It will
overwrite the
response generated
by servlet2

Dispatching using include() method

The request is received by one servlet and response is generated by same after including responses from other servlets .



**Response
generated**

Receives the request and
dispatches it to other servlet

Generates an response
and dispatches to servlet
3

Adds some more
data to the
response and
sends back to
servlet2

MainPage.java

```
protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1) throws ServletException, IOException {
    PrintWriter out = arg1.getWriter();
    out.println( "synergetics-india Software Pvt Ltd." );
    String accounttype = arg0.getParameter("accounttype");
    RequestDispatcher rd;
    if (accounttype.equalsIgnoreCase("Current")){
        rd = arg0.getRequestDispatcher("/current");
        rd.forward(arg0, arg1);
    }
    else {
        rd = arg0.getRequestDispatcher("/saving");
        rd.include(arg0, arg1);
    }
    out.println("Lok Center , Marol - Maroshi Road , Marol , Andheri(E) .");
}
```

The Cookies - Topics Overview

- Pros and Cons of using Cookies
- Session and Persistent Cookies
- Sending and Receiving Cookies
- Application of Cookies

Preserving client data

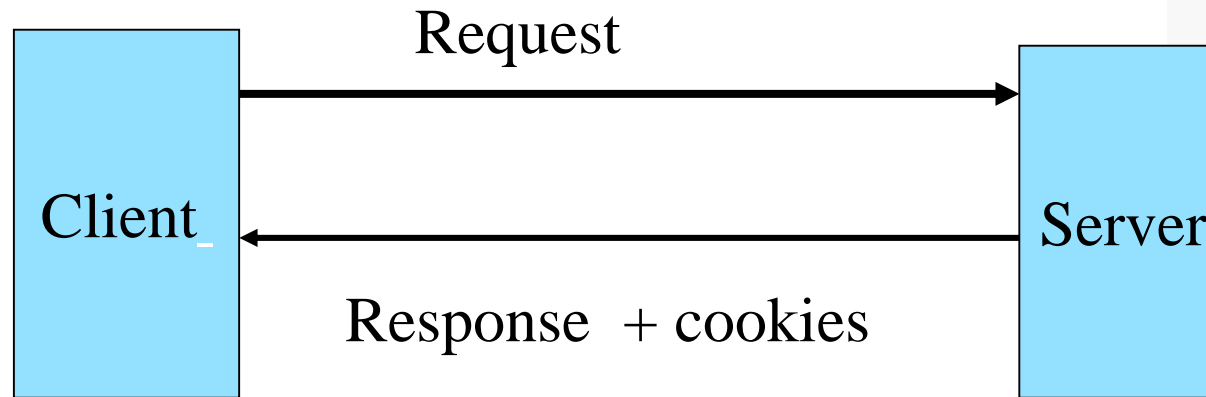
- At the server side in database
- At the server side in any other storage
 - Text files
 - Serialized form
- At the client side in the form of cookies

More about Cookies

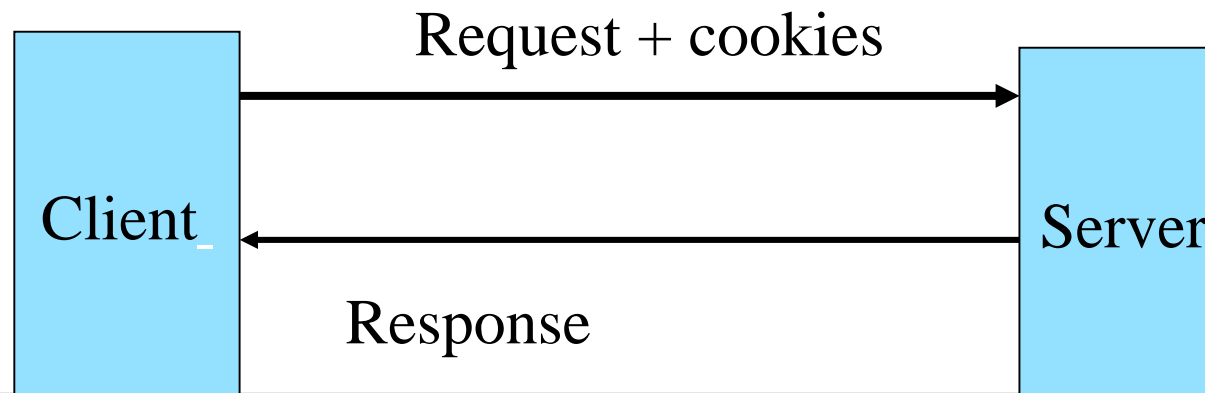
- Cookies is:
 - a textual information stored at client's machine by the server.
 - neither a virus nor carrier of a virus.
 - cannot contain Executable code.
 - carries less confidential client specific information.
- Browser restricts:
 - 20 cookies for a website.
 - 300 cookies for a browser instance.
 - 4kb size for a single cookie.

Working of Cookies :

A cookie generation:



The next request:



Creating Cookies

- Three steps to create a new cookie:
 - 1) Create a new Cookie Object
 - **Cookie cookie = new Cookie (name, value);**
 - 2) Set cookie attributes
 - **cookie.setMaxAge (60);**
 - 3) Add your cookie to the response object:
 - **Response.addCookie (cookie)**

Creating Cookies (contd...)

```
public class CreateCookies extends HttpServlet{
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1) throws ServletException, IOException{
        for(int i=0; i<3; i++){
            Cookie cook = new Cookie("Session"+i, "Value "+i);
            arg1.addCookie(cook);// This is a cookie with age for a session only.
            Cookie cook1 = new Cookie("Persistent"+i, "Value "+i);
            if (i==0)
                cook1.setMaxAge(60);
            if (i==1)
                cook1.setMaxAge(120);// Age set for two minutes.
            if (i==2)
                cook1.setMaxAge(60*60);
            arg1.addCookie(cook1);// This is a cookie persistant for an hour.
        }
        PrintWriter out = arg1.getWriter();
        out.println("To see the cookies, visit the");
        out.println("<AHREF = \"/MyServlets/showcookies\">SHOWCOOKIES</A>");
    }
}
```

Reading Cookies

```
public class ShowCookies extends HttpServlet {  
    protected void doGet ( HttpServletRequest arg0, HttpServletResponse arg1) throws ServletException,  
        IOException {  
        PrintWriter out = arg1.getWriter( );  
        Cookie [ ] cookies = arg0.getCookies( );  
        for(int i= 0 ; i < cookies.length; i++){  
            out.println( "<TR>"+" <TD>" + cookies[i].getName()+  
                " <TD>" + cookies[i].getValue() ) ;  
        }  
    }  
}  
  
protected void doPost(HttpServletRequest arg0, HttpServletResponse arg1) throws ServletException,  
    IOException{  
    doGet(arg0, arg1);  
}  
}
```


Applications of cookies

- Customizing the sites
- Focused advertising
- Storing information about the client

Applications of cookies (contd...)

```
public class RepeatVisitor extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,  
        IOException {  
        boolean newVisitor = true;  
        Cookie[ ] cookies = request.getCookies();  
        if(cookies != null){  
            for(int i =0;i<cookies.length;i++){  
                Cookie c = cookies[i];  
                if(c.getName().equals("repeatVisitor") && c.getValue().equals("yes")){  
                    newVisitor = false;  
                    break;  
                }  
            }  
        }  
    }  
}
```

Applications of cookies (contd...)

```
String title , color;  
if(newVisitor) {
```

```
    Cookie returnVisitorCookie = new Cookie("repeatVisitor","yes");  
returnVisitorCookie.setMaxAge(60*60*24*365); // persistent for 1 year  
response.addCookie(returnVisitorCookie);  
    title = "WELCOME !!!!!";  
    color = "skyblue";
```

```
}
```

```
else {
```

```
    title = "Nice to see you again !!!!!";  
    color = "pink";
```

```
}
```

```
PrintWriter out = response.getWriter();
```

```
out.println("<html>\n" + "<head><title>" + title + "</title></head>\n" +
```

```
    "<body bgcolor = "+color+">\n" + "<h1 align=\"center\">" + title + "</h1>\n" +  
    "</body></html>");
```

```
}
```

Disadvantages of cookies

- Browsers block cookies
- Threat to privacy

The Session Tracking - Topics Overview

- Session Object
- Setting and getting attributes
- Session Tracking API
- Browser and Server sessions

Need of Session Tracking

- To identify a user if he re-visits your site.
- To maintain the state , when there are series of requests from the same user.

Session Handling

- Accessing session object associated with current request :
 - `request.getSession();`
- Looking up information associated with a session :
 - `session.getAttribute();`
- Storing information in a session :
 - `session.setAttribute();`
- Discarding Session :
 - `session.invalidate();`

Various Ways of Session Tracking

- Cookies
- URL Rewriting
- Hidden Form Fields

The Hidden Form Fields

The HTML :

```
<html> <body>
<form action = "/MyServlet/hiddenfield" Method = POST align =Center>
USERNAME : <input type = text name = "user" align = center>
<input type = submit value = "Login" align = center>
</form> </body> </html>
```

```
public class HiddenField extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
```

```
        String username = request.getParameter("user");
        PrintWriter out = response.getWriter();
        out.println("Hello !! click Submit to proceed further");
        out.println("<form action =\"/MyServlet/secondservlet\">");
        out.println("<input type =\"hidden\" name =\"user\" value =\""+username+"\">");
        out.println("SURNAME : <input type =\"text\" name = \"surname\" );
        out.println("<input type =\"Submit\" value = \"Submit\"</form>");
```

```
}
```

The Hidden Form Fields (contd...)

```
public class SecondServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
        ServletException, IOException {  
        doPost(request,response);  
    }  
  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
        ServletException, IOException {  
        String uname = request.getParameter("user");  
        String surname = request.getParameter("surname");  
        PrintWriter out = response.getWriter();  
        out.println("Hello !!! "+uname);  
    }  
}
```

The URL Rewriting

```
public class URLRewriting extends HttpServlet{

    public void doGet(HttpServletRequest request,HttpServletResponse response)throws IOException, ServletException{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String str = "/MyServlet/SessionTracking";
        HttpSession ses = request.getSession();
        String encodedURL = response.encodeURL(str);
        out.println("test session");
        out.println("<html><body>");
        out.println("<form action =" + encodedURL + ">");
        out.println("<input type =\"Submit\" value = \"Submit\"</form>");
        out.println("</body></html>");

    }
}
```

The Cookies

- Can pass information:
 - For both GET and POST type of requests.
 - Even if application has static web pages.
- Cannot pass information:
 - If Browser blocks cookies.
 - If web page is bookmarked

The Session Tracking API

Provides a simple interface to manage the session automatically.

- **The API manages session through:**
 - Cookies
 - URL rewriting
- **The API can do:**
 - Provides implicit session object
 - Creates unique session ID
 - Tracks session using session ID
 - Provides listeners for implicit session objects
- **API Hierarchy :**
 - public interface HttpSession

Methods of HttpSession

- isNew()
- invalidate()
- getAttribute(String name)
- setAttribute(String name)
- getMaxInactiveInterval()
- setMaxInactiveInterval()

The Session creation

```
public class SessionTracking extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("Test session");  
        HttpSession ses = request.getSession();  
        if(ses.isNew()){  
            out.println("This is a new Sesion");  
        }  
        else{  
            out.println("Welcome back");  
        }  
    }  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        doGet(request , response);  
    }  
}
```

Filters and Listeners - Topics Overview

- Introduction to JSP Technology
- Writing and deploying first .jsp program
- Basic Syntax of JSP Elements
- JSP Life Cycle
- JSP scripting elements: Declarations, Scriptlets and Expressions

Agenda

- Exploring the need of filter
- Exploring filter API
- Creating a Filter
- Configuring a Filter
- Using initialization parameter in Filters



The Need for Filter

- Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:
 - To intercept requests from a client before they access a resource at back end.
 - To manipulate responses from server before they are sent back to the client.

Types of Filter suggested

- Authentication Filters.
- Data compression Filters
- Encryption Filters .
- Filters that trigger resource access events.
- Image Conversion Filters .
- Logging and Auditing Filters.
- MIME-TYPE Chain Filters.
- Tokenizing Filters .
- XSL/T Filters That Transform XML Content



Exploring the Filter API

- Defined in javax.servlet.* package
 - Filter
 - FilterChain
 - FilterConfig
- We define the filter by implementing the Filter interface
- FilterChain is passed to a filter by Container for invoking the series of Filter
- FilterConfig contains initialization data if required



Exploring the Filter interface

- Filter interface contains doFilter method which is the heart of filter
- It usually performs the following action
 - Examines the request header
 - Customizes the request Object (like header,data etc)
 - Customizes the response object(like header, data etc)

Creating a Filter

- **Basic format**

```
public void doFilter(ServletRequest request,  
ServletResponse response,  
FilterChain chain)  
throws ServletException, IOException {  
...  
chain.doFilter(request,response);  
}
```

- **Note on first two arguments**

- They are of type ServletRequest and ServletResponse, not HttpServletRequest and HttpServletResponse.
 - Do a typecast if you need HTTP-specific capabilities

- **Note on final argument**

- It is a FilterChain, not a Filter. Its doFilter method is different – two arguments only.

Simple Report Filter

```
public class ReportFilter implements Filter {  
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws  
        ServletException, IOException {  
        HttpServletRequest req = (HttpServletRequest)request;  
        System.out.println(req.getRemoteHost() + " tried to access " + req.getRequestURL() +  
            " on " + new Date() + ".");  
        chain.doFilter(request,response);  
    }  
    public void init(FilterConfig config) throws ServletException { }  
    public void destroy() {}  
}
```

Registering the Filter in web.xml

```
<web-app>
```

```
<!-- Register the name "Reporter" for ReportFilter. -->
```

```
<filter>
```

```
<filter-name>Reporter</filter-name>
```

```
<filter-class>moreservlets.filters.ReportFilte</filter-class>
```

```
</filter>
```

Important note

Servers load filters into memory when the Web app first comes up. So, if that filter is not found, your *entire Web app is disabled*.

Associating Reporting with given URL's

<!-- Apply Reporter filter to home page. -->

<filter-mapping>

<filter-name>Reporter</filter-name>

<url-pattern>/index.jsp</url-pattern>

</filter-mapping>

<!-- Also apply Reporter filter to servlet named "TodaysSpecial". -->

<filter-mapping>

<filter-name>Reporter</filter-name>

<servlet-name>TodaysSpecial</servlet-name>

</filter-mapping>

</web-app>

Using initialization Parameter for Filter

- Filter can be configured to use initialization parameters like a normal servlet like
 - <init-param>
 - <context-param>
- <init-param>
 - Use FilterConfig inside init method to read the parameters
- <context-param>
 - Using FilterConfig need to obtain ServletContext.
 - Read the parameter from the ServletContext thus obtained

Advantages of Filter

- **Encapsulate common behavior.**
 - Have 30 different servlets or JSP pages that need to compress their content to decrease download time? Make 1 compression filter and apply it to all 30 resources.
- **Separate high-level access decisions from presentation code.**
 - Want to block access from certain sites without modifying the individual pages to which these access restrictions apply? Create an access restriction filter and apply it to as many pages as you like.
- **Apply wholesale changes to many different resources.**
 - Have a bunch of existing resources that should remain unchanged except that the company name should be changed? Make a string replacement filter and apply it wherever appropriate.

Listeners - Topics Overview

- Understanding Listener
- Monitoring creation and destruction in ServletContext
- Recognizing session creation and destruction
- Recognizing request creation and destruction

Understanding Listeners

- JSP is a template page technology
 - High level abstraction of Servlets
- Separation of presentation from logic
- Even non java programmer can create JSP pages with reasonable ease



Available Listeners

- **Servlet context listeners.**
 - These listeners are notified when the servlet context (i.e., the Web application) is initialized and destroyed.
- **Servlet context attribute listeners.**
 - These listeners are notified when attributes are added to, removed from, or replaced in the servlet context.
- **Session listeners.**
 - These listeners are notified when session objects are created, invalidated, or timed out.
- **Session attribute listeners.**
 - These listeners are notified when attributes are added to, removed from, or replaced in any session.

Creating a Listeners

- **Implement the appropriate interface.**
 - Use ServletContextListener, ServletContextAttributeListener, HttpSessionListener, or HttpSessionAttributeListener.
- **Override the methods needed to respond to the events of interest.**
 - Provide empty bodies for the other methods in the interface.
- **Access the important Web application objects.**
 - Six objects that you are likely to use in event-handling methods:
 - The servlet context
 - The name of the servlet context attribute that changed
 - The value of the servlet context attribute that changed
 - The session object
 - The name of the session attribute that changed

Creating a Listeners

- **Use these objects.**
 - This process is application specific, but there are some common themes. For example, with the servlet context, you are most likely to read initialization parameters (`getInitParameter`), store data for later access (`setAttribute`), and read previously stored data (`getAttribute`).
- **Declare the listener.**
 - You do this with the `listener` and `listener-class` elements of the general Web application deployment descriptor (*web.xml*) or of a tag library descriptor file.
- **Provide any needed initialization parameters.**
 - Servlet context listeners commonly read context initialization parameters to use as the basis of data that is made available to all servlets and JSP pages. You use the context-param *web.xml* element to provide the names and values of these initialization parameters.

Monitoring Creation and Destruction

- **The ServletContextListener class responds to the Initialization and destruction of the servlet context.**
 - These events correspond to the creation and shutdown of the Web application itself.
- **ServletContextListener is most commonly used to**
 - Set up application-wide resources like database connection pools
 - Read the initial values of application-wide data that will be used by multiple servlets and JSP pages.

Implementing ServletContextListener

- **Implement the ServletContextListener interface.**
- **Override contextInitialized and contextDestroyed.**
 - **contextInitialized is triggered when the Web application is first** loaded and the servlet context is created. Most common tasks:
 - Creating application-wide data (e.g., by reading context init params)
 - Storing that data in an easily accessible location .
 - **contextDestroyed is triggered when the Web application is being** shut down and the servlet context is about to be destroyed. Most common task:
 - Releasing resources (e.g. closing connections).
- **Obtain a reference to the servlet context.**
 - The contextInitialized and contextDestroyed methods each take a ServletContextEvent as an argument.
 - The ServletContextEvent class has a getServletContext method that returns the servlet context

Implementing ServletContextListener

- **Use the servlet context.**
 - Read initialization parameters: `getInitParameter`
 - Store data: `setAttribute`
 - Make log file entries: `log`.
- **Declare the listener.**

```
<listener>  
  <listener-class>package.Listener</listener-class>  
</listener>
```
- **Provide needed initialization parameters.**

```
<context-param>  
  <param-name>name</param-name>  
  <param-value>value</param-value>  
</context-param>
```

Implementing ServletContextAttributeListener

- **Implement ServletContextAttributeListener**
- **Override attributeAdded, attributeReplaced, and attributeRemoved.**
 - attributeAdded is triggered when a new attribute name is first added to the servlet context.
 - attributeReplaced is triggered when a new value is assigned to an existing name. attributeAdded is *not triggered in this case*. The old value is obtained via event.getValue and the new value is obtained via context.
 - getAttribute. attributeRemoved is triggered when a servlet context attribute is removed altogether.
- **Obtain references to the attribute name, attribute value, and servlet context.**
 - Call the following methods of the event object: getName,getValue, and getServletContext

Implementing ServletContextAttributeListener

- **Use the objects.**

- You normally compare attribute name to a stored name to see if it is the one you are monitoring. The attribute value is used in an application-specific manner. The servlet context is usually used to read previously stored attributes (getAttribute), store new or changed attributes (setAttribute), and make entries in the log file (log).

- **Declare the listener.**

- Use the listener and listener-class elements to list the fully qualified name of the listener class,

```
<listener>
```

```
  <listener-class>
```

```
    somePackage.SomeListener
```

```
  </listener-class>
```

```
</listener>
```

Recognizing Session Creation and destruction

- **Implement the HttpSessionListener interface.**
- **Override sessionCreated and sessionDestroyed.**
 - sessionCreated is triggered when a new session is created.
 - sessionDestroyed is triggered when a session is destroyed. This destruction could be due to an explicit call to the invalidate method or because the elapsed time since the last client access exceeds the session timeout.
 - Multithreaded access is possible. Synchronize if necessary.
- **Obtain a reference to the session and possibly to the servlet context.**
 - Each of the two HttpSessionListener methods takes an HttpSessionEvent as an argument. The HttpSessionEvent class has a getSession method that provides access to the session object. You almost always want this reference; you occasionally also want a reference to the servlet context. If so, first obtain the session object and then call getServletContext on it

Recognizing Session Creation and destruction

- **Use the objects.**

- One of the only methods you usually call on the session is `setAttribute`. Do this in `sessionCreated` if you want to guarantee that all sessions have a certain attribute.
- Wait! What about `getAttribute`? Nope. In `sessionCreated`, there is nothing in the session yet, so `getAttribute` is pointless. In addition, all attributes are removed before `sessionDestroyed` is called, so calling `getAttribute` is also pointless there. If you want to clean up attributes that are left in sessions that time out, you use the `attributeRemoved` method of `HttpSessionAttributeListener`. So, `sessionDestroyed` is mostly reserved for listeners that are simply keeping track of the number of sessions in use.

- **Declare the listener.**

- In `web.xml` or the TLD file, use `listener` and `listener-class` to list fully qualified name of listener class, as below.

`<listener>`

`<listener-class>package.SomeListener</listener-class>`

`</listener>`

Using HttpSessionAttributeListener

- **Implement HttpSessionAttributeListener.**
- **Override attributeAdded, attributeReplaced, and attributeRemoved.**
 - attributeAdded is triggered when a new attribute name is first added to a session.
 - attributeReplaced is triggered when a new value is assigned to an existing name. attributeAdded is *not triggered in this case*. The old value is obtained via event.getValue and the new value is obtained via session.getAttribute.
 - attributeRemoved is triggered when a session attribute is removed altogether. This removal can be due to an explicit programmer call to removeAttribute, but is more commonly due to the system removing all attributes of sessions that are about to be deleted because their timeout expired.

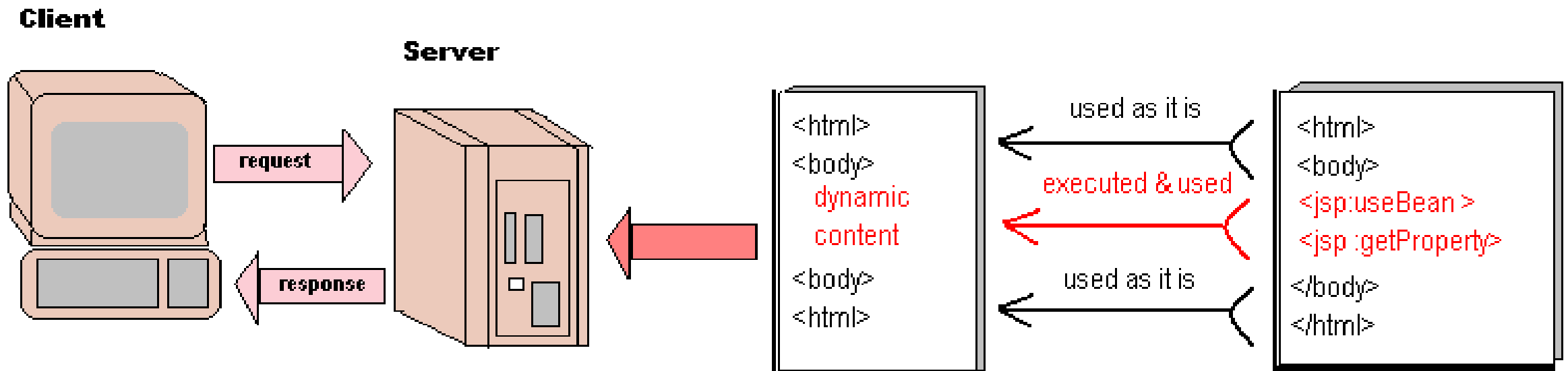
Using HttpSessionAttributeListener

- **Obtain references to the attribute name, attribute value, session, & ServletContext.**
 - The HttpSessionAttributeListener methods take an HttpSessionBindingEvent as args. HttpSessionBindingEvent has three useful methods: getName (name of attribute that was changed), getValue (value of changed attribute—new value for attributeAdded and previous value for attribute Replaced and attributeRemoved), and getSession (the HttpSession object). If you want access to the servlet context, first obtain the session and then call getServletContext on it.
- **Use the objects.**
 - The attribute name is usually compared to a stored name to see if it is the one you are monitoring. The attribute value is used in an application-specific manner. The session is usually used to read previously stored attributes (getAttribute) or to store new or changed attributes (setAttribute).
- **Declare the listener.**
 - Use listener and listener-class in *web.xml* as before.

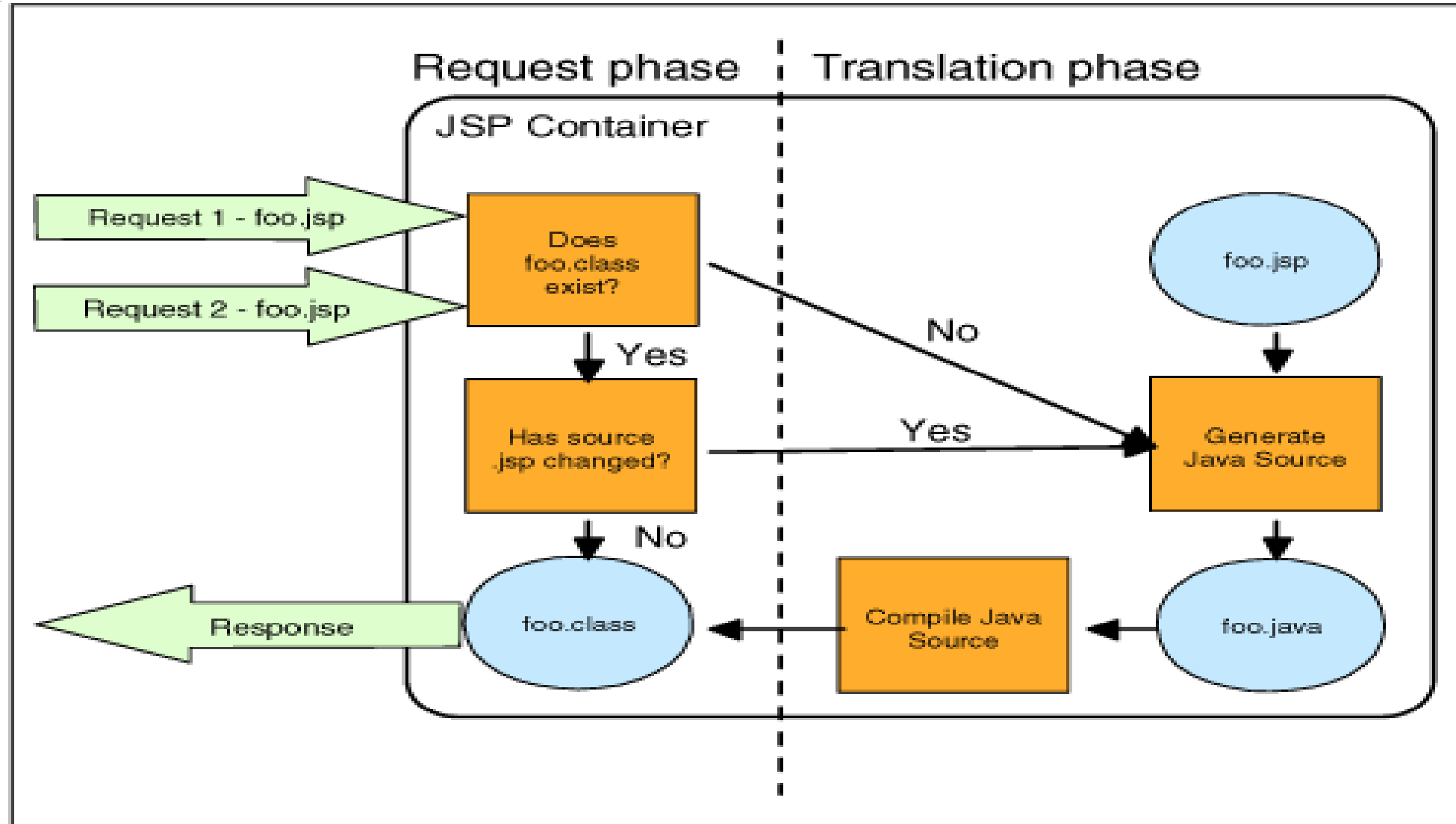
The JSP Technology - Topics Overview

- Introduction to JSP Technology
- Writing and deploying first .jsp program
- Basic Syntax of JSP Elements
- JSP Life Cycle
- JSP scripting elements: Declarations, Scriptlets and Expressions

JSP (Java Server Pages)



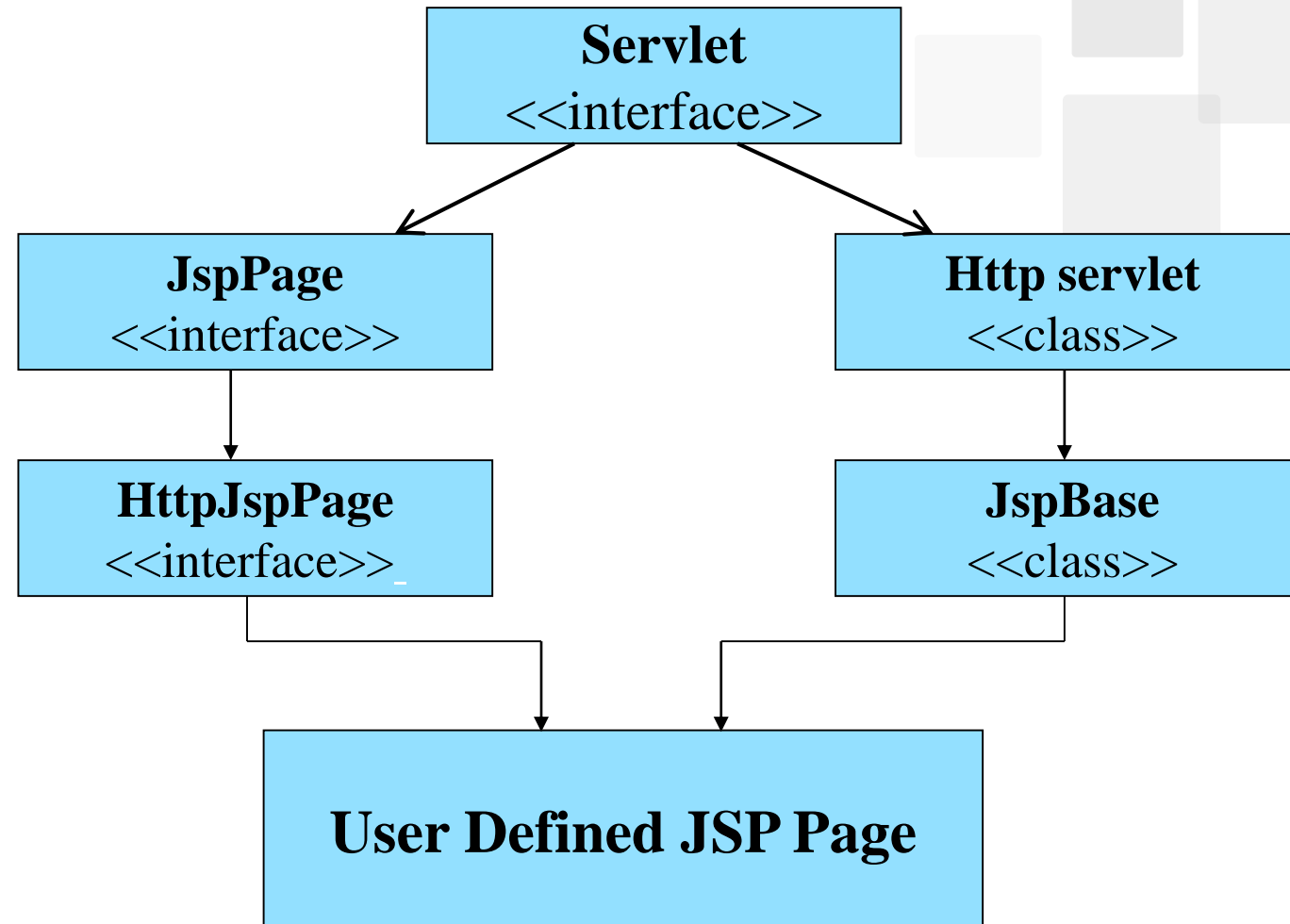
How JSP works ?



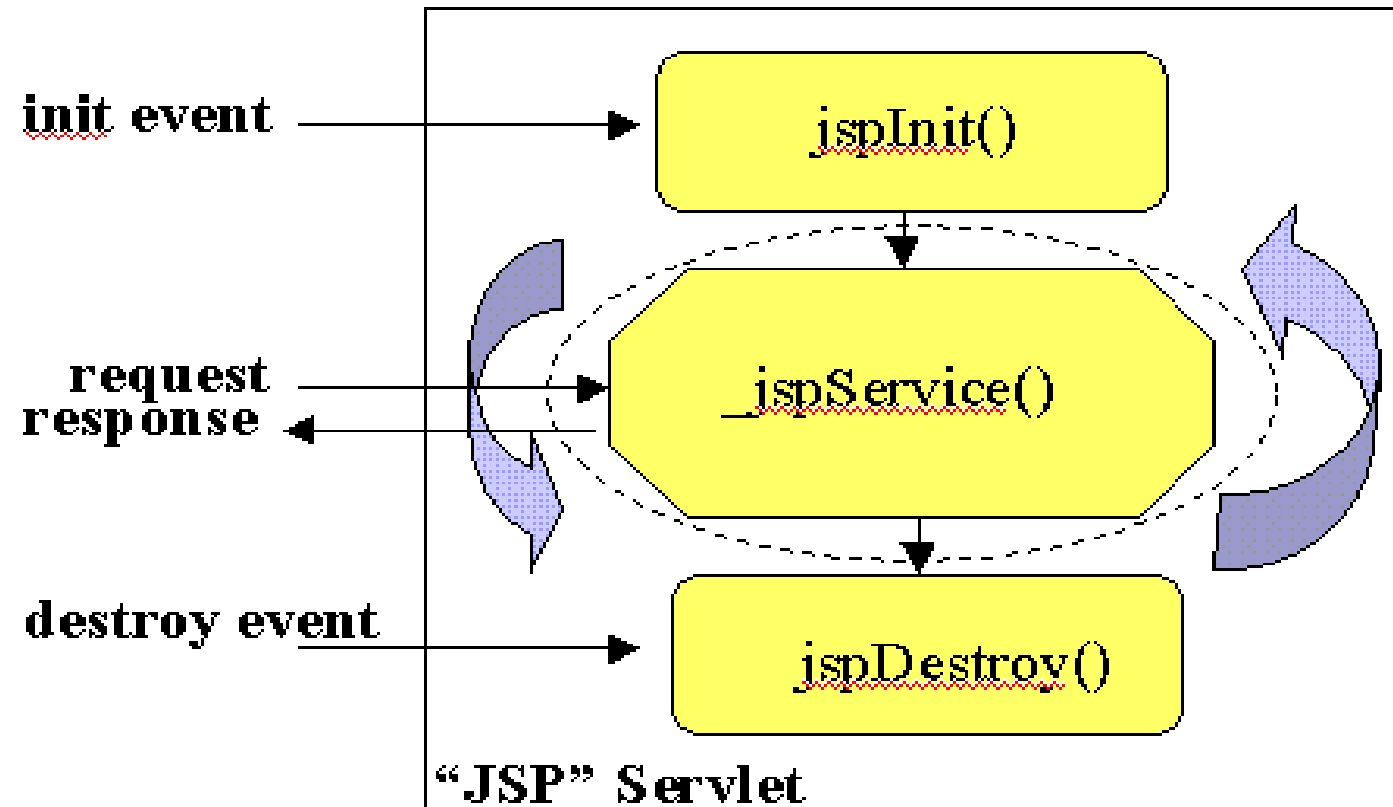
The First.jsp

```
<html>  
<body>  
  WELCOME!!!!!!  
</body>  
</html>
```

The JSP API



The JSP LifeCycle



The Elements of JSP

```
<%@ page import = "packjava.Counter" %>
```

Directives

```
<html>
```

```
<body>
```

The page count is

```
<%
```

```
    out.println(Counter.getCount());
```

```
%>
```

```
</body>
```

```
</html>
```

Scriptlets

```
package packjava;
    public class Counter{
        private static int count;
        public static int getCount(){
            count++;
            return count;
        }
    }
```


The Elements of JSP (Contd...)

```
<%@ page import ="packjava.*" %>
```

```
<html>
```

```
<body>
```

```
<%
```

The page count is

```
<%= Counter.getCount() %>
```

```
%>
```

```
</body>
```

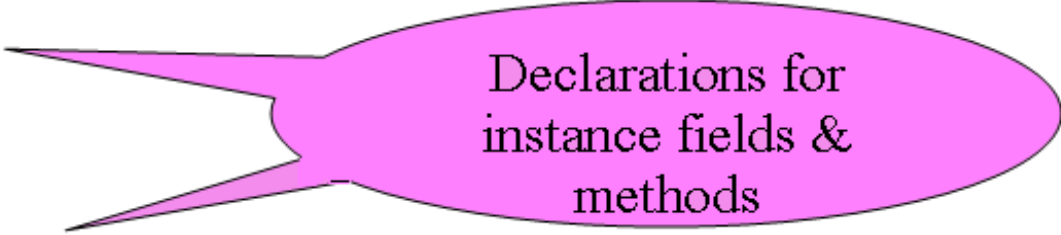
```
</html>
```

Expressions

```
package packjava;  
public class Counter{  
    private static int count;  
    public static int getCount(){  
        count++;  
        return count;  
    }  
}
```

The Elements of JSP (Contd...)

```
<%! static int count = 0 %>
<%!      public void sayHello(){
          out.println("Hello you are the
          count + "visitor to the site");
          count++;
        }
%>
```



Declarations for
instance fields &
methods

```
public class Counter extends HttpServlet {
    static int count =0;
    public void sayHello(){
        ---- ;
    }
    public void _jspService(HttpServletRequest req,HttpServletResponse res)
        throwsServletException{
        ----- ;
    }
}
```

Various Elements in a JSP page

- **HTML Comment** : `<!-- -->`
- **JSP Comment** : `<%-- --%>`
- **JSP Scriptlets** : `<% %>`
- **JSP Directives** : `<% @ %>`
- **JSP Expressions** : `<% = %>`
- **JSP Declaration** : `<%! %>`

Configuring init parameters for JSP

```
<web-app>
```

```
....
```

```
<servlet>
```

```
    <servlet-name>MyJsp</servlet-name>
```

```
    <jsp-file>/MyJsp.jsp</jsp-file>
```

```
    <init-param>
```

```
        <param-name>email</param-name>
```

```
        <param-value>admin@yahoo.com</param-value>
```

```
    </init-param>
```

```
</servlet>
```

```
.....
```

```
</web-app>
```

Predefined Variables and Page Directives - Topics Overview

- Using predefined variables
- Handling response using response variable
- Using Page directives

JSP predefined variables

1. JspWriter	→	out
2. HttpServletRequest	→	request
3. HttpServletResponse	→	response
4. HttpSession	→	session
5. ServletContext	→	application
6. ServletConfig	→	config
7. PageContext	→	pageContext
8. Object	→	page

The page Directives

- Attributes to the page directive
 - import
 - errorPage
 - isErrorPage
 - contentType
 - extends

Including Files - Topics Overview

- Compile time and Runtime inclusion
- The 'include' action and 'include' directive
- Forwarding request/including response
- Pros and cons of Compile and Runtime inclusions

Compile Time File Inclusion

Include Directive :

Includes the specified page during translation phase.

➤ Syntax :

```
<%@include file="/JSPPageToBeIncluded.jsp" %>
```

Including pages at runtime

- **The jsp:include Action :**
 - `<jsp:include page="URL" flush="true">`
- **The page Attribute :**
 - It specifies the relative URL of the page whose output should be included.
- **The flush Attribute :**
 - This attribute is optional.
 - Specifies whether the output stream of the main page should be flushed before the inclusion of page.

The <jsp:forward> Tag

- **Propagates the request object from one JSP to another JSP or servlet.**
- **Syntax :-**
 - `<jsp:forward page = “ URL ” >`
- **Example :-**
 - `<jsp:forward page="/servlet/login" />`
 - `<jsp:forward page="/servlet/login">
 <jsp:param name="username" value="jsmith" />
</jsp:forward>`

The “param” element

- **Syntax :**

- `<jsp:param name= “paramName” value=“paramValue”>`

- **Used with include and forward actions**

- `<jsp:include page=“URL” flush=“true”>`

`<jsp:param name=“paramname” value=“paramvalue”>`

`</jsp:include>`

Differences in compile time and runtime file inclusion

	jsp : include Action	Include Directive
Syntax	<code><jsp:include page=“...”/></code>	<code><%@ include file=“..”%></code>
Time of inclusion	Request Time	Page Translation time
What is included	Output of included page	Actual code of the file
Number of Servlets generated	2 (main page and included page each become separate servlet)	1 (included file is inserted into main page, that page is translated into a servlet)

The Pro's and Cons

- Maintenance
- Speed

The 'useBean' Tag in JSP - Topics Overview

- Building and accessing Beans
- Collecting Form Data using Beans
- Sharing Beans

Collecting Form Parameters using JSP

```
<HTML>
<BODY>
  <TITLE>JSP Collect Parameters</TITLE>
  <%= request.getParameter("flights")%>
  <%
    String val1 = request.getParameter("emailAddress");
    String val2 = request.getParameter("origin");
    String val3 = request.getParameter("desti");
    String val4 = request.getParameter("totCost");
    String val5 = request.getParameter("advance");
  %>
  Information Details : <BR>
  Email Address : <%=val1%>
  Origin : <%=val2%>
  Destination : <%=val3%>
  Total Cost : <%=val4%>
  Advance Paid : <%=val5%>
</BODY>
</HTML>
```


The “useBean” Tag

- The “useBean” Tag :
 - **<jsp : useBean id : “idName” class = “beanClass” type = “refType” scope = “variableScope” />**
- The setProperty :
 - **<jsp:setProperty name= “idName” property= “propertyName” value = “value” >**
- The getProperty :
 - **<jsp:getProperty name = “ idName” property = “propertyName” >**

Collecting Form Data using “useBean” tag

```
<jsp:useBean id = “ entry ” class = “ pack020param.TravAgentBean ” />
```

Information Details :

Email Address : <jsp : getProperty name = “ entry ” property= “ emailAddress ” />
Origin : <jsp : getProperty name = “ entry ” property= “ origin ” />
Destination : <jsp : getProperty name = “ entry ” property= “ destination ” />
Total Cost : <jsp : getProperty name = “ entry ” property= “ totalCost ” />
Advance Paid : <jsp : getProperty name = “ entry ” property= “ advance ” />

Setting values using "useBean" Tag

```
<jsp:useBean id = "entry" class = "pack020param.TravAgentBean" />
```

```
<% -- 1 --- %>
```

```
<jsp:setProperty name = "entry" property = "emailAddress"  
    value = '<%=request.getParameter("emailAddress") %>' />
```

```
<%-- 2 --%>
```

```
<% float tcost = 1.0f; tcost = Float.parseFloat(request.getParameter("totalCost")); %>
```

```
<jsp:setProperty name = "entry" property = "totalCost" value = "<%= tcost %>" />
```

```
<%-- 3 --%>
```

```
<jsp:setProperty name = "entry" property = "destination" param = "desti" />
```

```
<jsp:setProperty name = "entry" property = "totalCost" param = "totCost" />
```

```
<%-- 4 --%>
```

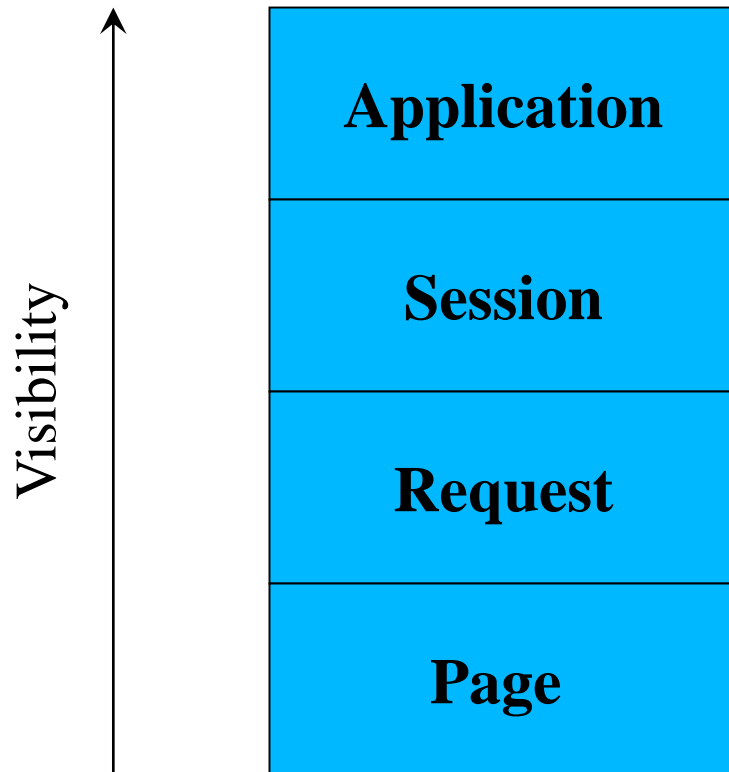
```
<jsp:setProperty name = "entry" property = "advance" />
```

```
<jsp:setProperty name = "entry" property = "origin" />
```

```
<%--5--%>
```

```
<jsp:setPropert name ="entry" property =" * " />
```

Sharing Beans



Objects accessible from pages that belong to same application

Objects accessible to pages belonging to same session in which they were created

Objects accessible to the pages processing the request where they were created

Objects accessible only within the pages where they are created. **This is the default scope**

```
<jsp:setProperty name = "entry" property = "advance" scope = "request"/>
```

The Expression Language (EL) - Topics Overview

- Accessing properties, arrays and collections
- The EL Implicit objects
- The EL Scopes
- Few other EL Operators

Importance of EL

- It simplifies the presentation layer by replacing hard-to-maintain Java scripting elements with short and readable entries.
- **Syntax :**
\$ { expression }

Why EL?

- Concise access to stored objects.
- Shorthand notation for bean properties.
- Simple access to collection elements.
- Access to request parameters, cookies and other request data.
- Small but useful set of simple operators.
- Conditional output.
- Automatic type conversion.
- Empty values instead of error messages.

The Implicit Object references in EL

- **Map of Scope Attributes :**
 - pageScope
 - requestScope
 - sessionScope
 - applicationScope
- **Maps of request parameter :**
 - param
 - paramValues
- **Maps of RequestHeader :**
 - header
 - headerValues
- **Map of context init parameter :**
 - initParam
- **PageContext**
 - Actual reference to pageContext object

Accessing Implicit Objects

Accessing Context Information

```
${pageContext.servletContext.serverInfo }  
${pageContext.servletContext.majorVersion }  
${pageContext.servletContext.minorVersion }  
${pageContext.servletContext.servletContextName }
```

Accessing session information

```
${pageContext.session.id }  
${pageContext.session.lastAccessedTime }  
${pageContext.session.creationTime }
```

Accessing config information

```
${pageContext.servletConfig.servletName }
```

Accessing request information

```
${ pageContext.request.method }  
${ pageContext.request.contextPath }
```

Accessing Bean Properties

<%@ page isELIgnored='false' %>

Accessing Bean properties using the “Bean” object with session scope

Email Address : \${ Bean.emailAddress }
Destination : \${ Bean.destination }
Total Cost : \${ Bean.totalCost }
Advance Paid : \${ Bean.advance }

Accessing an ArrayList using the “WeakNames” reference with session scope

\${ WeekNames[0] } \${ WeekNames[1] }
\${ WeekNames[2] } \${ WeekNames[3] }
\${ WeekNames[4] } \${ WeekNames[5] }
\${ WeekNames[6] } \${ WeekNames[7] }

Accessing an Array using the “MonthNames” reference with session scope

\${ MonthNames[0] } \${ MonthNames[1] }
\${ MonthNames[2] } \${ MonthNames[3] }
\${ Friends["Deshpande"] } \${ Friends["Mishra"] }
\${ Friends["Dawane"] } \${ Friends["Govindraj"] }

The EL Operators

- **Arithmetic Operators**

- +
- -
- *
- / or div
- % or mod

- **Relational Operators**

- == or eq
- != or ne
- < or lt
- < or gt
- <= or le
- >= or ge

- **Logical Operators**

- &&
- and
- ||
- or
- !
- not

- **Empty Operator**

- empty

Using EL Operators

```
<html>
<body>
  ${num>3}<br><br>
  ${integer le 12 }<br><br>
  ${list[0] || list["1"] and true}<br><br>
  ${num > integer }<br><br>
  ${num == integer-1 }<br><br>
  ${42 div 0}
  ${list[0]} ${list["1"]}
  ${requestScope[integer] ne 4 and 6 le
    num || false }<br><br>
</body>
</html>
```

```
protected void doGet(HttpServletRequest request,
  HttpServletResponse response) {
    String num="2";
    request.setAttribute("num",num);
    Integer i = new Integer("3");
    request.setAttribute("integer",i);

    ArrayList list = new ArrayList();

    list.add("true"); list.add("false");
    list.add("2"); list.add("10");

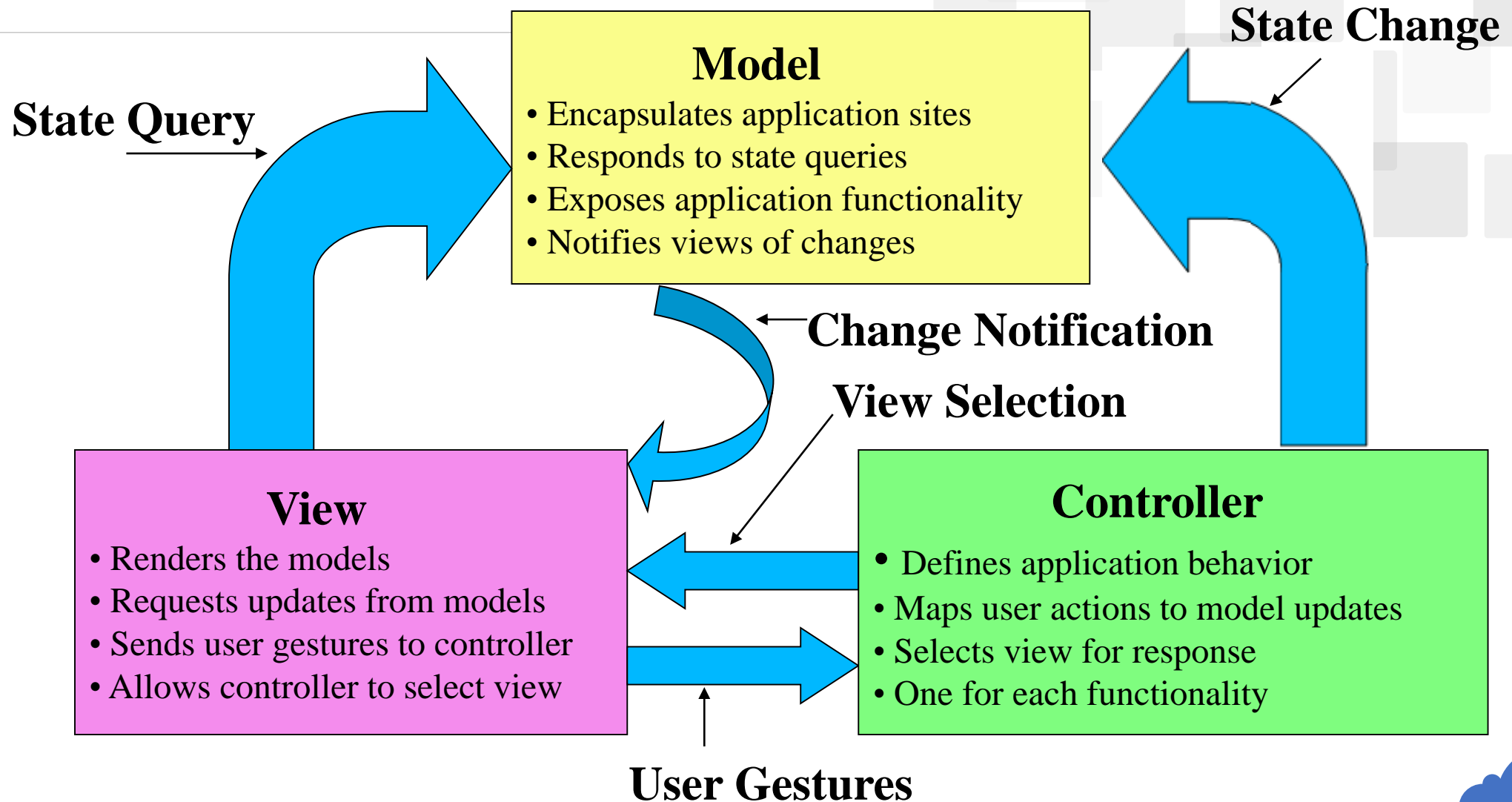
    request.setAttribute("list",list);

    RequestDispatcher rd =
        request.getRequestDispatcher("/eloperator");
    rd.forward(request,response);
}
```

The MVC Architecture - Topics Overview

- The MVC Architecture
- Dispatching requests between Servlet and JSP
- Packaging elements of Application

Model View Controller (MVC)



Model View Controller (MVC) contd...

Model :

- Encapsulates the application logic.
- Could be more than one for an application.
- Can be a POJO or EJB or DAO or combination of these.
- Is the only part of the application which interacts with the DB.

View :

- Forms the presentation layer of your application.
- Could be GUI or JSP or HTML or any other presentation type.

Controller :

- Processes and responds to events, typically user actions, and may invoke changes on the model.

JSTL and Custom Tags - Topics Overview

- JSTL
- Simplifying JSP using Custom Tags.
- The TagSupport and BodyTagsupport classes.
- Designing Custom Tags and using them

JSP Standard Tag Library

- Introduction to JSTL
- Exploring the features of JSTL
- Working with core Tag Library
- Working with SQL Tag library

Introduction to JSTL

- Contains Standard tag library
- Library is available in all compliant containers
- Wide range of custom action functionality
- Readability and Maintainability

Function Overview of JSTL

- Core functionality
- XML Manipulation
- SQL
- Internationalization and Formatting
- Note
 - Makes use of expression language



JSTL expression language

- Accessed via `${expression}`
- Provides shorthand notation to access
 - Attributes of standard servlet Objects
 - Bean properties
 - Map, List and Array Element
- Standard part of JSP 2.0
 - In JSTL EL can be used only in attributes
 - In JSP 2.0 EL can be used anywhere

Logic tags

- One choice: if
`<c:if test="{someTest}">Content`
`</c:if>`
- Lots of choices: choose
`<c:choose>`
`<c:when test="test1">Content1</c:when>`
`<c:when test="test2">Content2</c:when>`
`<c:when test="testN">ContentN</c:when>`
`<c:otherwise>Default Content</c:otherwise>`
`</c:choose>`

Caution: resist use of business logic!

The "if" tag

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

<c:forEach var="i" begin="1" end="10">

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

<c:if test="\${i > 7}">

(greater than 7)

</c:if>

</c:forEach>

"Choose" tag

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<UL>
<c:forEach var="i" begin="1" end="10">
<LI> <c:out value="${i}"/>
<c:choose> <c:when test="${i < 4}">

```

Database Access Tag

- **<sql:setDataSource>**
 - Specifies data source (can also be set by config settings)
- **<sql:query>**
 - Queries database and stores ResultSet in variable
 - Warning: this usage violates rule of keeping business logic out of presentation layer. Instead, do database access in servlet and pass results to JSP via MVC.

Database Access Tag

- **<sql:update>**
- **<sql:param>**
- **<sql:dateParam>**
- **<sql:transaction>**
 - Performs the enclosed **<sql:query>** and **<sql:update>** actions as a single transaction

sql:setDataSource

- **You can set a data source globally via configuration settings or application-scoped variables.**

Preferred approach in real applications

Or, you can set it on a specific page

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>  
<sql:setDataSource driver="com.mysql.jdbc.Driver"  
url="jdbc:mysql://localhost:3306/test" user="root"  
password="root"/>
```

sql:query

- **Form 1: use the sql attribute**
 - `<sql:query var="results" sql="SELECT * FROM ..."/>`
- **Form 2: put the query in the body of the tag**
 - `<sql:query var="results">`
 - `SELECT * FROM ...`
 - `</sql:query>`

sql:query

- **Options**

- dataSource
- maxRows
- startRow

- **Caution**

- Embedding SQL directly in JSP may be hard to maintain



Simple Example

```
<%@ taglib prefix="c"
uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="sql"
uri="http://java.sun.com/jstl/sql" %>
<sql:setDataSource
driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/test"
user="root"
password="root"/>
```

Simple Example Continued

```
<sql:query var="employees">
```

```
SELECT * FROM employees
```

```
</sql:query>
```

```
<UL>
```

```
<c:forEach var="row" items="${employees.rows}">
```

```
<LI> <c:out value="${row.firstname}"/>
```

```
<c:out value="${row.lastname}"/>
```

```
</c:forEach>
```

```
</UL>
```



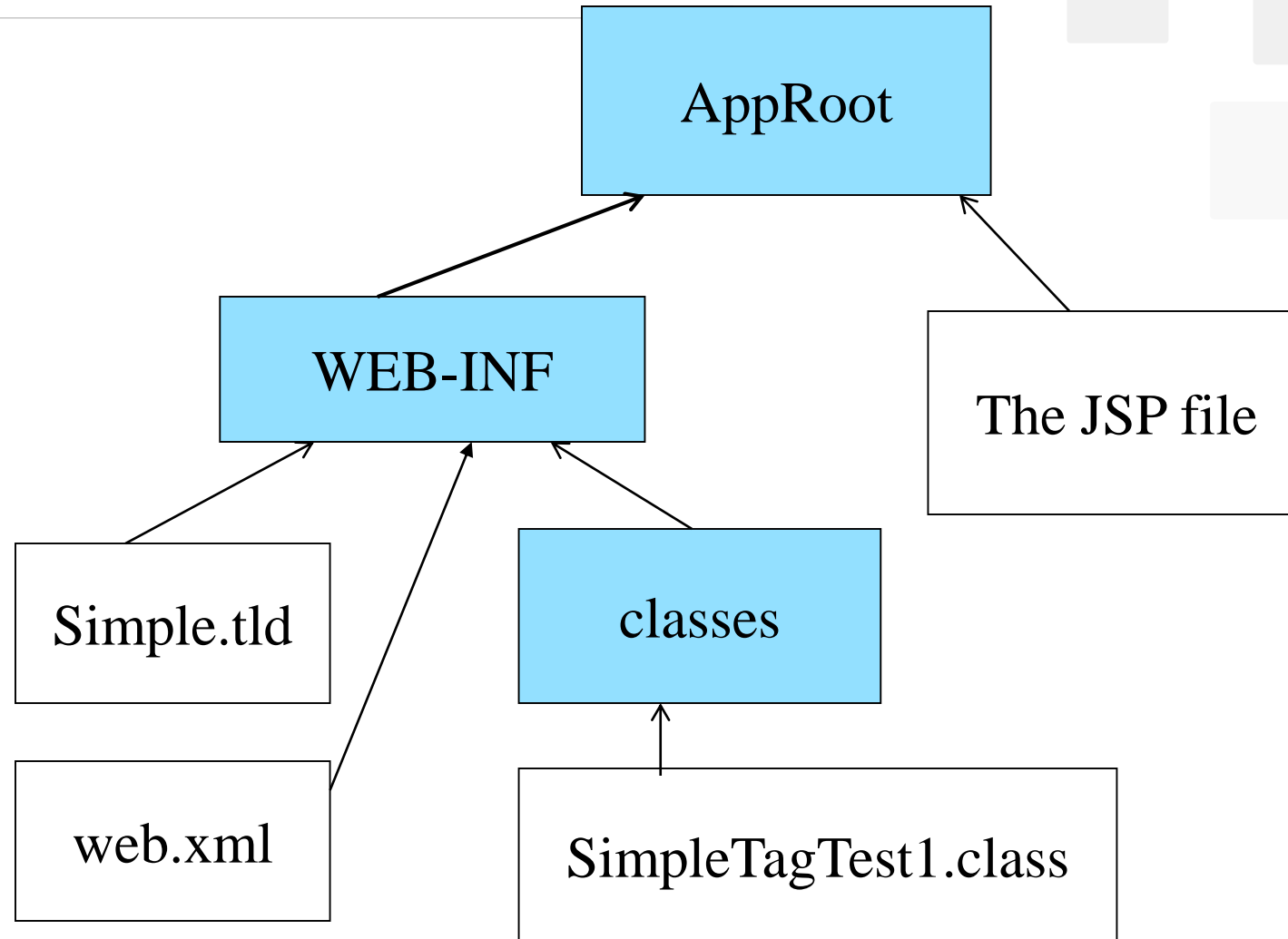
URL-Handling Tags

- **<c:import>**
 - Read content from arbitrary URLs
 - Insert into page
 - Store in variable
 - Or make accessible via a reader
 - Unlike <jsp:include>, not restricted to own system
- **<c:redirect>**
 - Redirects response to specified URL
- **<c:param>**
 - Encodes a request parameter and adds it to a URL
 - May be used within body of <c:import> or <c:redirect>

Simplifying JSP using Custom Tags.

- These are easy to use elements for JSP, but represents complex server side behavior.
- You can define your own tag, group them into collection called tag library that can be used in any number of JSP files.
- Steps necessary to build your Custom tag :
 - Create a “ TAG HANDLER CLASS ”.
 - Add a tag description into “TAG LIBRARY DESCRIPTOR FILE ” (.tld file).
 - Add <taglib> tag in web.xml
 - Use the tag in the JSP File

Making a Simple Tag Handler



Use of JSP custom tag

- Custom tag is a user defined JSP language element
- It is translated into servlet
- Tag is converted into operation on tag handler
- Container invokes those operation on the JSP page
- Three types of tag
 - Simple tag
 - Tags with attributes
 - Tags with body content



Use of JSP custom tag

- **Java-based tags**
 - Components of a tag library
 - Basic tags
 - Tags that use attributes
 - Tags that use body content
 - Tags that optionally use body content
- **JSP-based tags (tag files)**
 - Components of a tag library
 - Basic tags
 - Tags that use attributes
 - Tags that use body content



Components that make up a tag library

- **The Tag Handler Class**

- Java code that says what to output
- Must implement `javax.servlet.jsp.tagext.SimpleTag`
- Usually extends `SimpleTagSupport`
- Goes in same directories as servlet class files and bean
- **The Tag Library Descriptor File**
- XML file describing tag name, attributes, and implementing tag handler class
 - Goes under WEB-INF
- **The JSP File**
 - Imports a tag library (referencing URL of descriptor file)
 - Defines tag prefix
 - Uses tags

Simple illustration

- Create a java class that acts as a tag handler

package mytag;

import javax.servlet.jsp.tagext.*;

import javax.servlet.jsp.*;

import java.io.*;

public class HelloTag extends SimpleTagSupport

{

public void doTag() throws JspException, IOException {

JspWriter out = getJspContext().getOut(); out.println("Hello Custom Tag!");

}

}

Simple illustration

- compile above class and copy it in a directory available in environment variable CLASSPATH. Finally create following tag library file: <Tomcat-Installation-Directory>webapps\ROOT\WEB-INF\custom.tld.

```
<taglib>
```

```
<tlib-version> 1.0</tlib-version>
```

```
<jsp-version> 2.0</jsp-version>
```

```
<short-name>Example TLD</short-name>
```

```
<tag> <name>Hello</name>
```

```
<tag-class>mytag.HelloTag</tag-class>
```

```
<body-content>empty</body-content> </tag>
```

```
</taglib>
```

Simple illustration

- Now it's time to use above defined custom tag **Hello** in our JSP program as follows

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
```

```
<html> <head>
```

```
<title>A sample custom tag</title>
```

```
</head>
```

```
<body>
```

```
<ex:Hello/>
```

```
</body>
```

```
</html>
```

Accessing tag with body content

- Consider you want to define a custom tag named `<ex:Hello>` and you want to use it in the following fashion with a body:

`<ex:Hello>`

This is message body

`</ex:Hello>`

Tag file

```
package mytag;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;

public class HelloTag extends SimpleTagSupport {
    StringWriter sw = new StringWriter();
    public void doTag() throws JspException, IOException {
        getJspBody().invoke(sw);
        getJspContext().getOut().println(sw.toString());
    }
}
```

TLD

```
<taglib>
<tlib-version> 1.0</tlib-version>
<jsp-version> 2.0</jsp-version>
<short-name>Example TLD with Body</short-name>
<tag> <name>Hello</name>
<tag-class>com.tutorialspoint.HelloTag</tag-class>
<body-content>scriptless</body-content>
</tag>
</taglib>
```

JSP

```
<%@ taglib prefix="ex" uri="WEB-INF/custom.tld"%>
<html>
<head>
<title>A sample custom tag</title>
</head>
<body>
<ex:Hello> This is message body </ex:Hello>
</body>
</html>
```

Elements in TLD

Property	Purpose
name	The name element defines the name of an attribute. Each attribute name must be unique for a particular tag.
required	This specifies if this attribute is required or optional. It would be false for optional.
rtexprvalue	Declares if a runtime expression value for a tag attribute is valid
type	Defines the Java class-type of this attribute. By default it is assumed as String
description	Informational description can be provided.
fragment	Declares if this attribute value should be treated as a JspFragment .

Q & A

Contact: smitakumar@synergetics-india.com

Build COMPETENCY
across your TEAM



Microsoft Partner
Gold Cloud Platform
Silver Learning

Thank You
