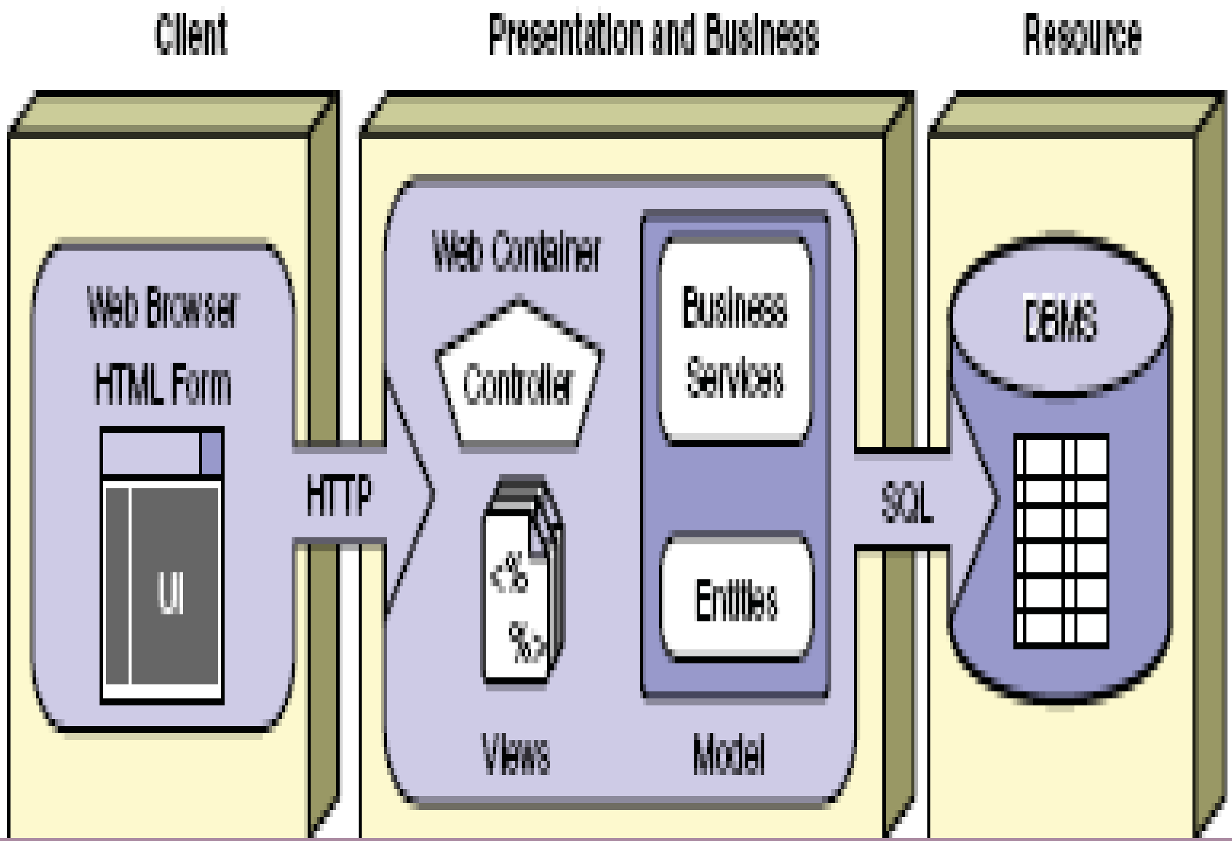
Web Component Development - Servlet

Sujata Batra

Role of Web Component JEE

- Web-centric Java EE application architecture:



Idea of Web Application

- Servlets, JSP pages, HTML files, utility classes, beans, tag libraries, etc. are bundled together in a single directory hierarchy or file

•

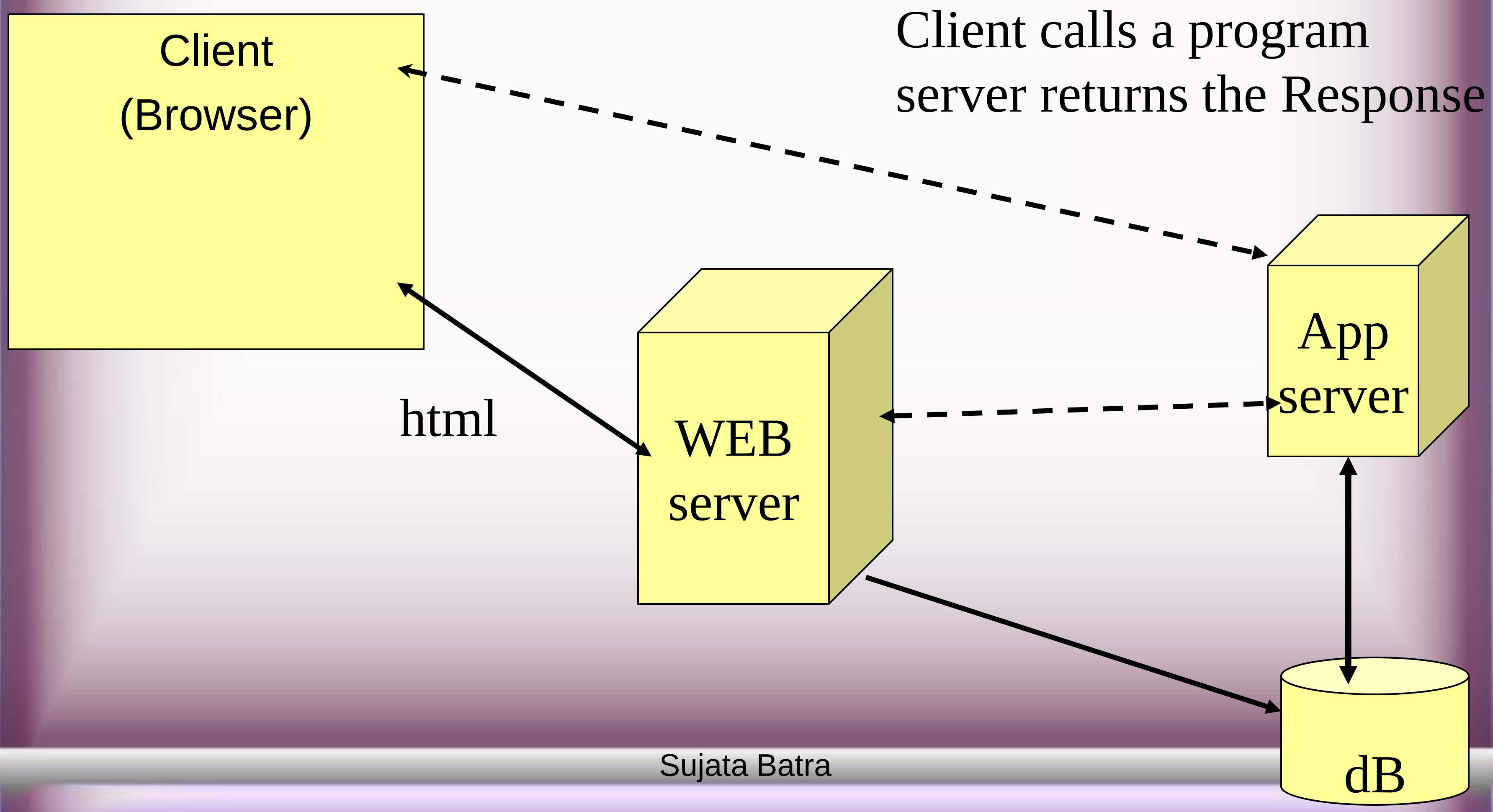
• Access to content in the Web app is always through a URL that has a common prefix

– – http://host/webAppPrefix/abc

– Many aspects of Web application behavior controlled through deployment descriptor- web.xml

APPLICATION SERVER

- A server computer on a computer network dedicated to running certain software applications
- A software engine that delivers applications to client computers.
- Should handle most, if not all, of the business logic and data access of the application.
- Ease of application development and centralization.
- Ex: Weblogic Server, WebSphere, JBOSS


Client
(Browser)
Client calls a program
server returns the Response
html
WEB server
App server
dB

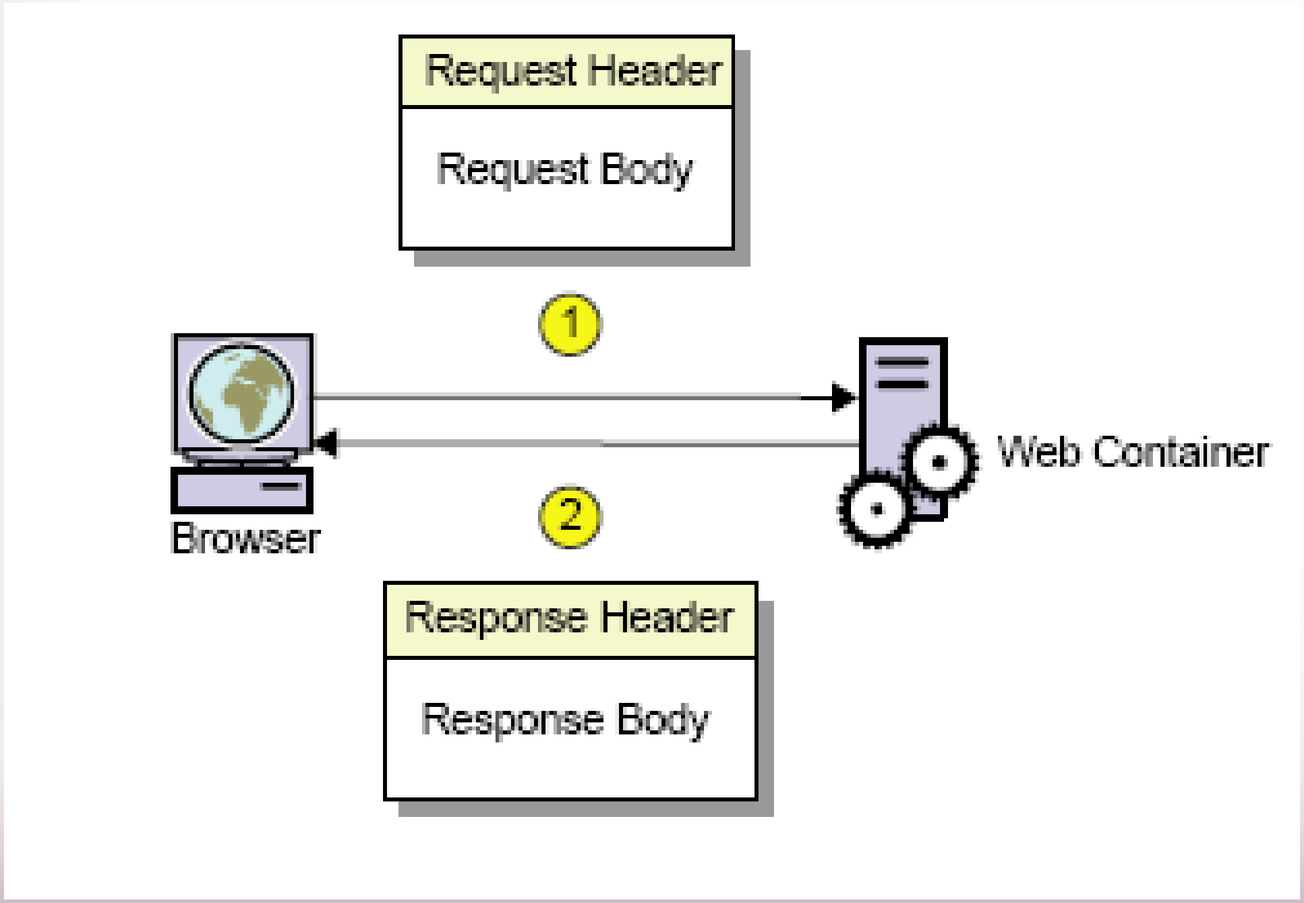
Java Servlets

- Released in 1997 , used to Create web pages with dynamic content
- Standard, server-side Java application that extends the capabilities of a Web Server.
- Mapped to URLs and Deployed in a Web container of an application server which provides a runtime environment for servlets.
- Executed when the J2EE application server receives a client request that is passed to the Web container, which in turn invokes the servlet.
- Platform and server independent

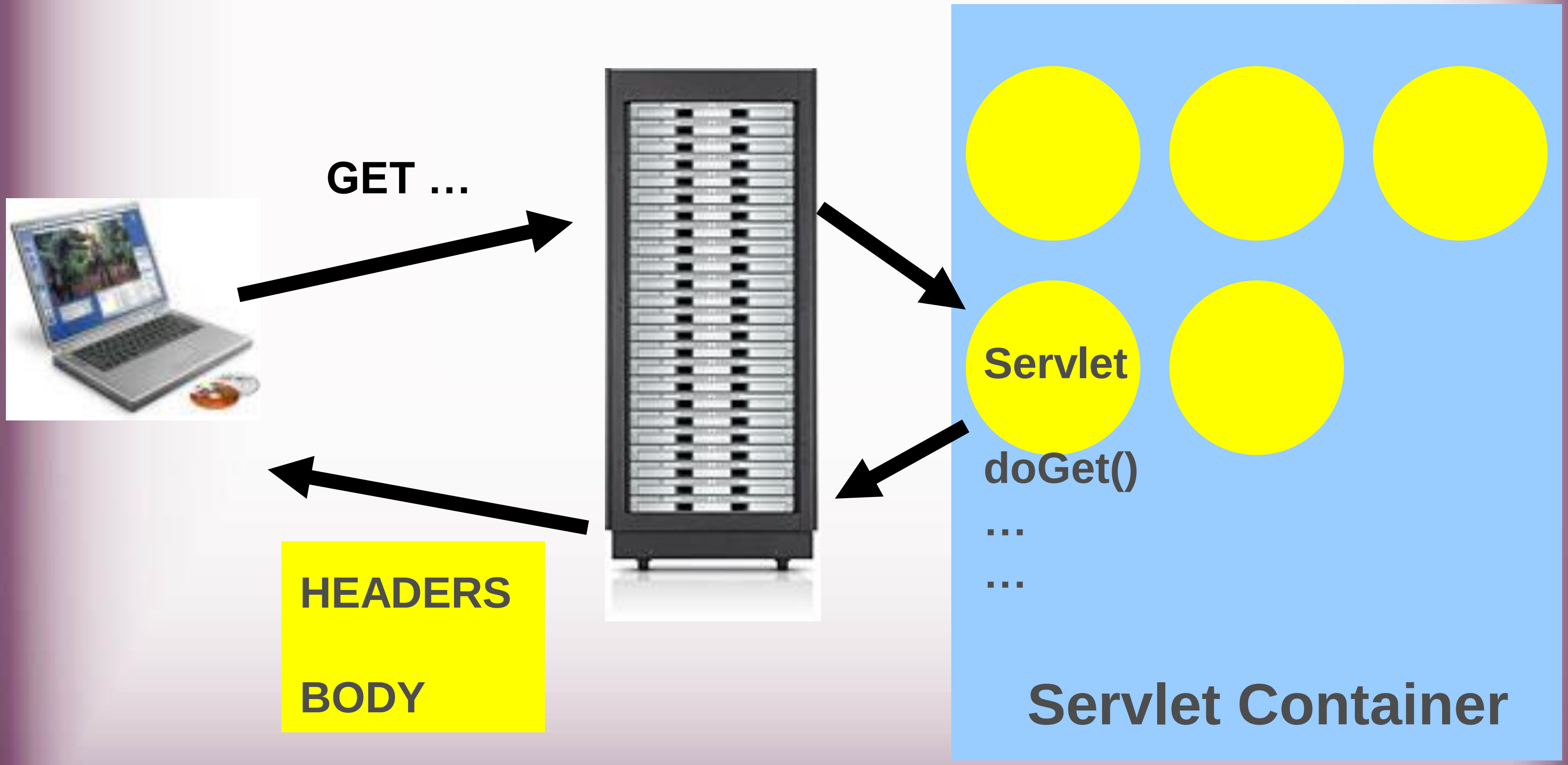
Servlets Features

- All server side frameworks share a common set of features:
  - Read data submitted by the user
  - Generate HTML dynamically based on user input
  - Determine information about the client browser
  - Access Database systems
  - Exploit the HTTP protocol

Request and Response



Web Apps with Servlets



HTML & HTTP

- HTML is a subset of SGML and is administered by the world wide Web Consortium

- Hypertext Transport protocol, specifies how HTML client , Known as browsers or user-agents and web servers communicate

- HTTP defines a few methods to which the web server responds , typically by sending an HTML document

Common HTTP Methods

- **GET -** The user-agent wants to retrieve a resource from the server ,
  - User entered information is appended to the URL in query string

- **POST-** The user-agent wants to pass information to the web server
  - User entered information sent as data (not appended to url)

- **URL** is an address that specifies the location of a resources such as file ,URL’s can be absolute or relative to the Webserver’s “document directory”

- **HTML forms**   let users enter or request information via the Browser

The Key Players

- WebServer
  - Gets the Response from the container
  - Uses HTTP to talk to the Client Browser
  - Knows how to forward to container
- Container
  - Find the Correct Servlet Using URL in DD
  - Manages life cycle of the servlets
  - Creates request and response objects just before starting the thread
  - Starts a Servlet Thread and gives that to the Servlet
  - Call the Service() Method and then doGet or doPost()
  - Destroys the Request,Response Objects
- Servlet
  - Has a public class name
  - Uses request Object to read the parameter from the user
  - Dynamic content for the Client
  - Uses response object to print a response

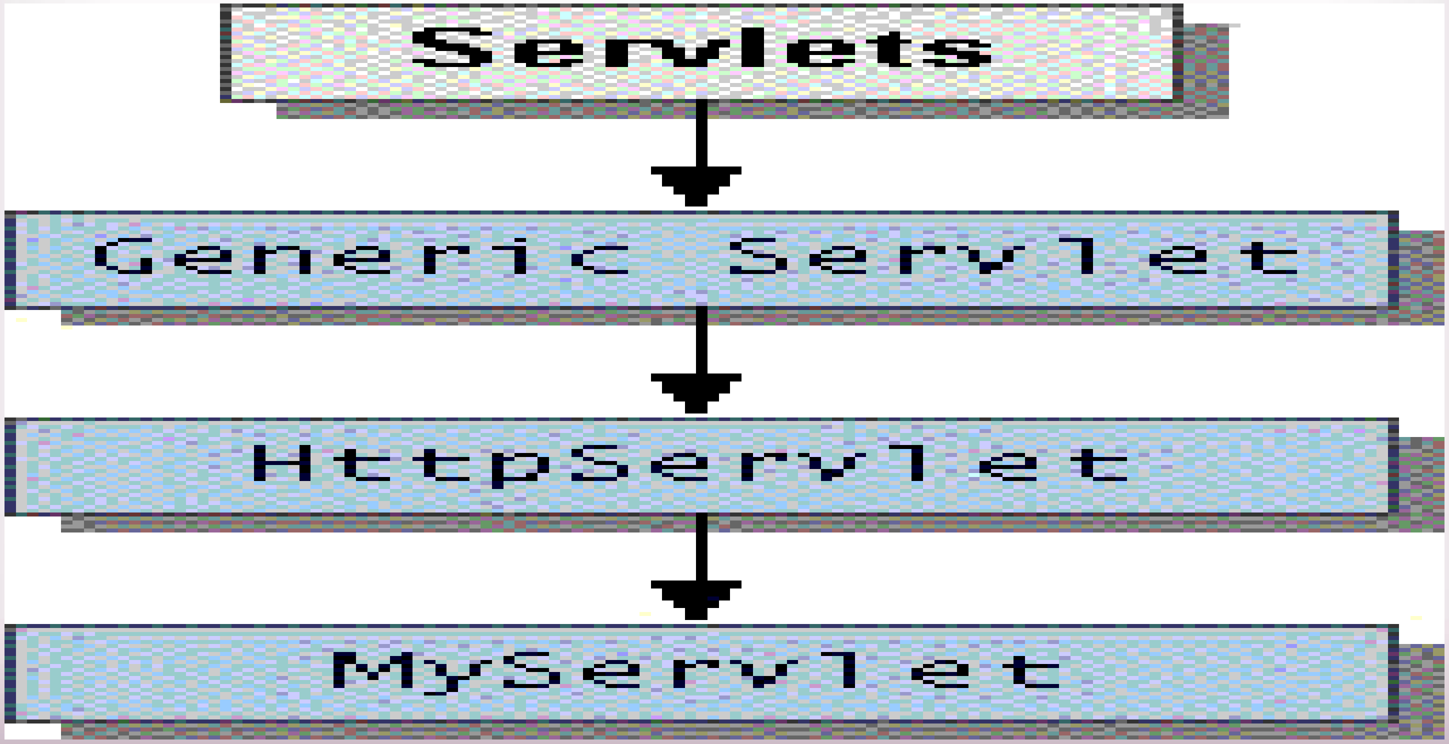
The Servlet Class Hierarchy

- The Servlet Class Hierarchy consists of two top level interfaces which are implemented by the GenericServlet class:
  - javax.servlet.Servlet
  - javax.servlet.ServletConfig

- The GenericServlet class is extended by the HttpServlet class which in turn is extended by a user defined class.

Architecture of the Servlet

- The ***javax.servlet*** package provides interfaces and classes for writing servlets.



The Servlet API

- javax.servlet package contains:
    - ServletRequest Interface
    - ServletResponse Interface
    - ServletContext Interface

- javax.servlet.http package contains:
    - HttpServletRequest Interface
    - HttpServletResponse Interface
    - HttpSession Interface

Servlet Interface

- All servlets implement Servlet interface either directly or more commonly, by extending a class that implements it such as HttpServlet

▪ The Servlet interface declares, methods that manage the servlet and its communications with clients.

▪ Servlet writers provide some or all of these methods when developing a servlet.

Steps for Java Servlets

1. Subclass of HttpServlet

2. Override doGet(....) method

3. HttpServletRequest
   – read the request parameters

4. HttpServletResponse
   – set Content Type
   – get PrintWriter
   – send text to client via PrintWriter

Web Deployment Descriptor

- /WEB-INF/web.xml
  - Part of the standard
  - Defines servlets used in the web application
  - Maps servlets to URLs
  - A servlet can map to many URLs
- Defines resources available to the web app
- Defines security constraints
- Defines other stuff like
  - Welcome file list
  - Session timeout
  - Error page mapping

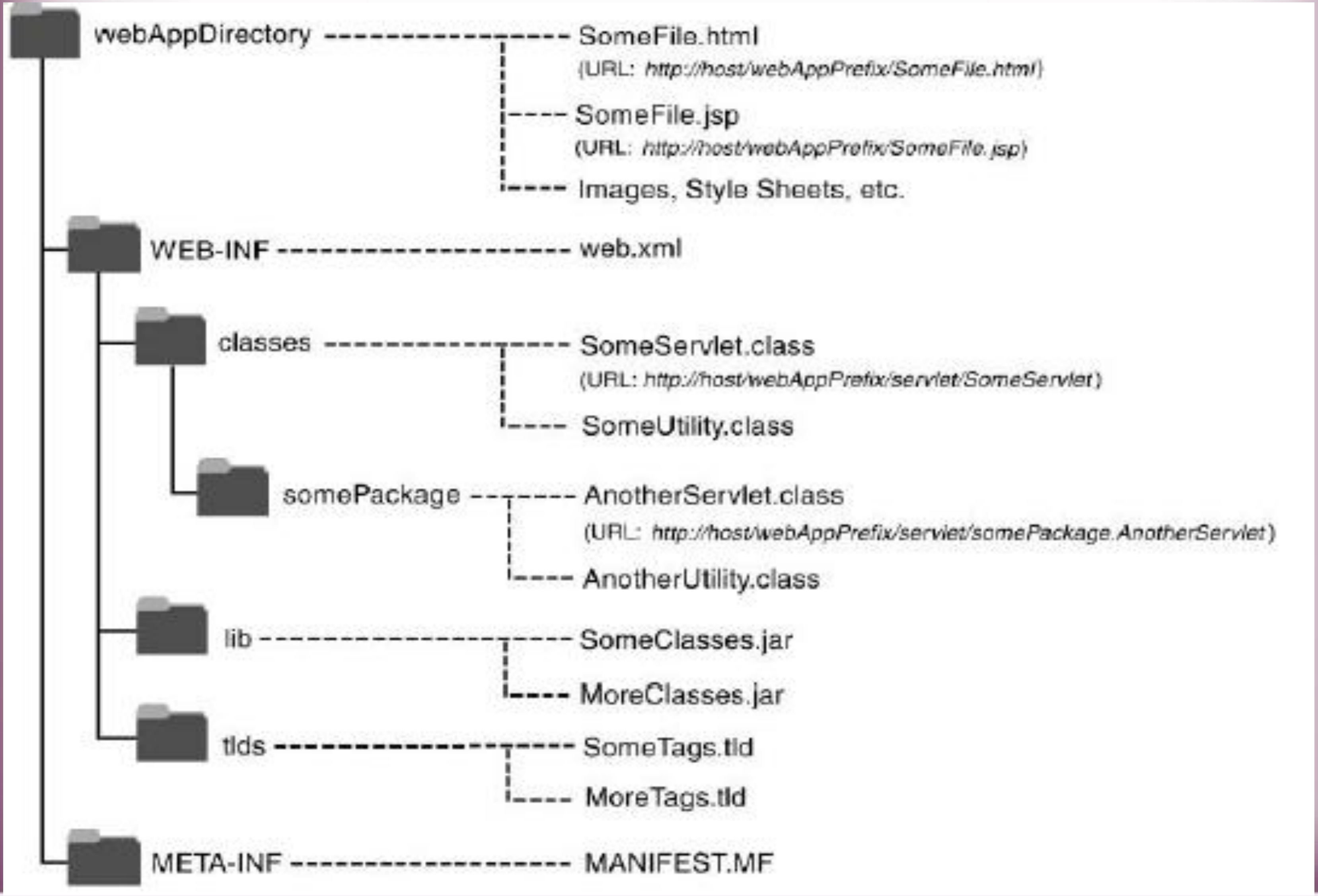
Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems,
  Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <display-name>FirstWebApp</display-name>
  <servlet>
   <servlet-name>FirtstServlet</servlet-name>
   <display-name>FirtstServlet</display-name>
   <servlet-class>com.training.FirtstServlet</servlet-
  class>
  </servlet>
  <servlet-mapping>
   <servlet-name>FirtstServlet</servlet-name>
   <url-pattern>/first</url-pattern>
  </servlet-mapping>
  </web-app>
```

J2EE Web Directory Structure

- Top Directory is normally the context Path
- WEB-INF directory
  - This is a protected directory, can not point browser to any file in this directory
  - /classes – unpacked web application classes, auto-magically added to CLASS_PATH
  - /lib – web application JAR files
  - /taglib – tag library descriptor files

J2EE Web Directory Structure

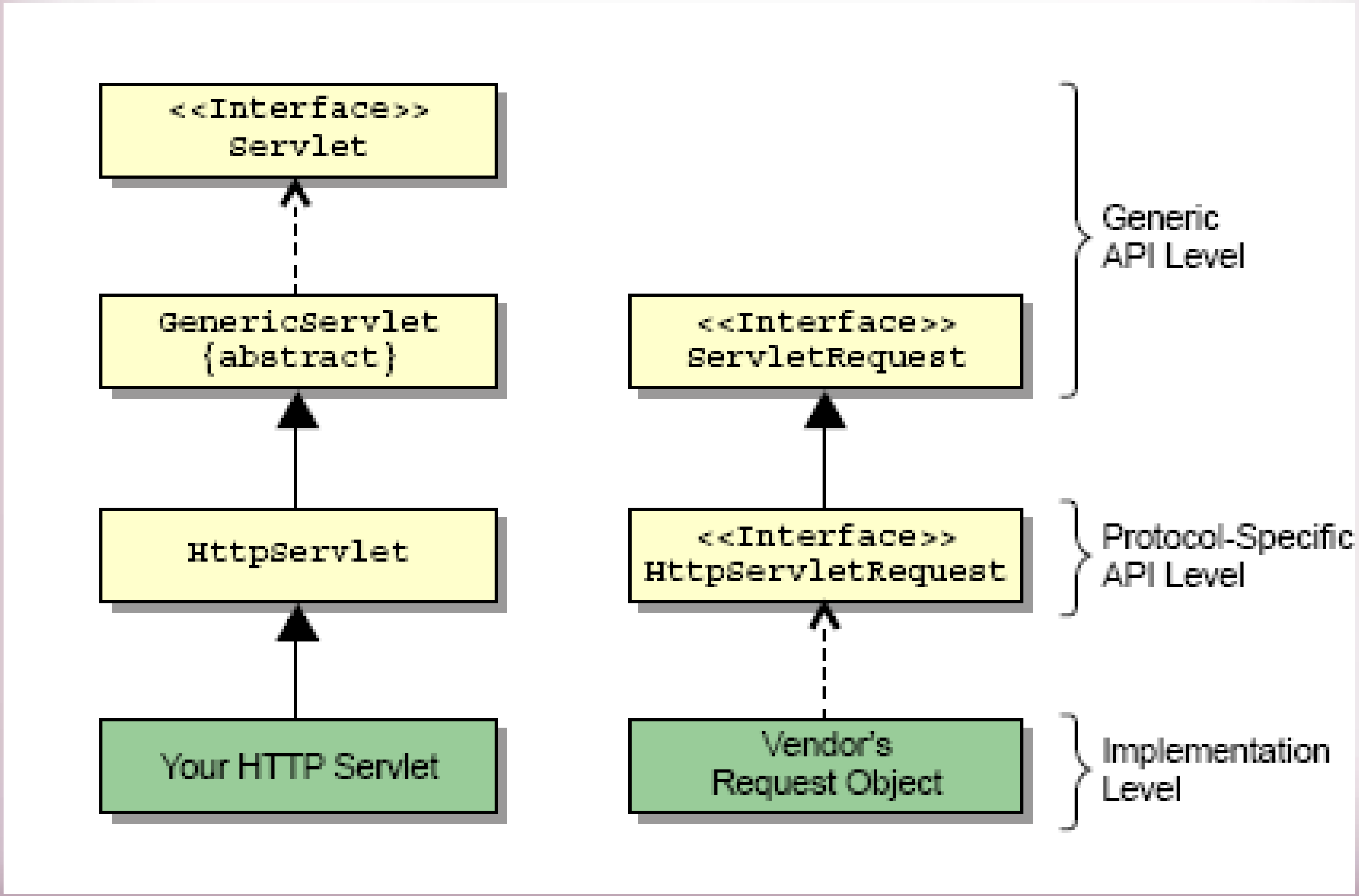

webAppDirectory
SomeFile.html
(URL: http://host/webAppPrefix/SomeFile.html)
SomeFile.jsp
(URL: http://host/webAppPrefix/SomeFile.jsp)
Images, Style Sheets, etc.
WEB-INF
web.xml
classes
SomeServlet.class
(URL: http://host/webAppPrefix/servlet/SomeServlet)
SomeUtility.class
somePackage
AnotherServlet.class
(URL: http://host/webAppPrefix/servlet/somePackage.AnotherServlet)
AnotherUtility.class
lib
SomeClasses.jar
MoreClasses.jar
tlds
SomeTags.tld
MoreTags.tld
META-INF
MANIFEST.MF

Example -1

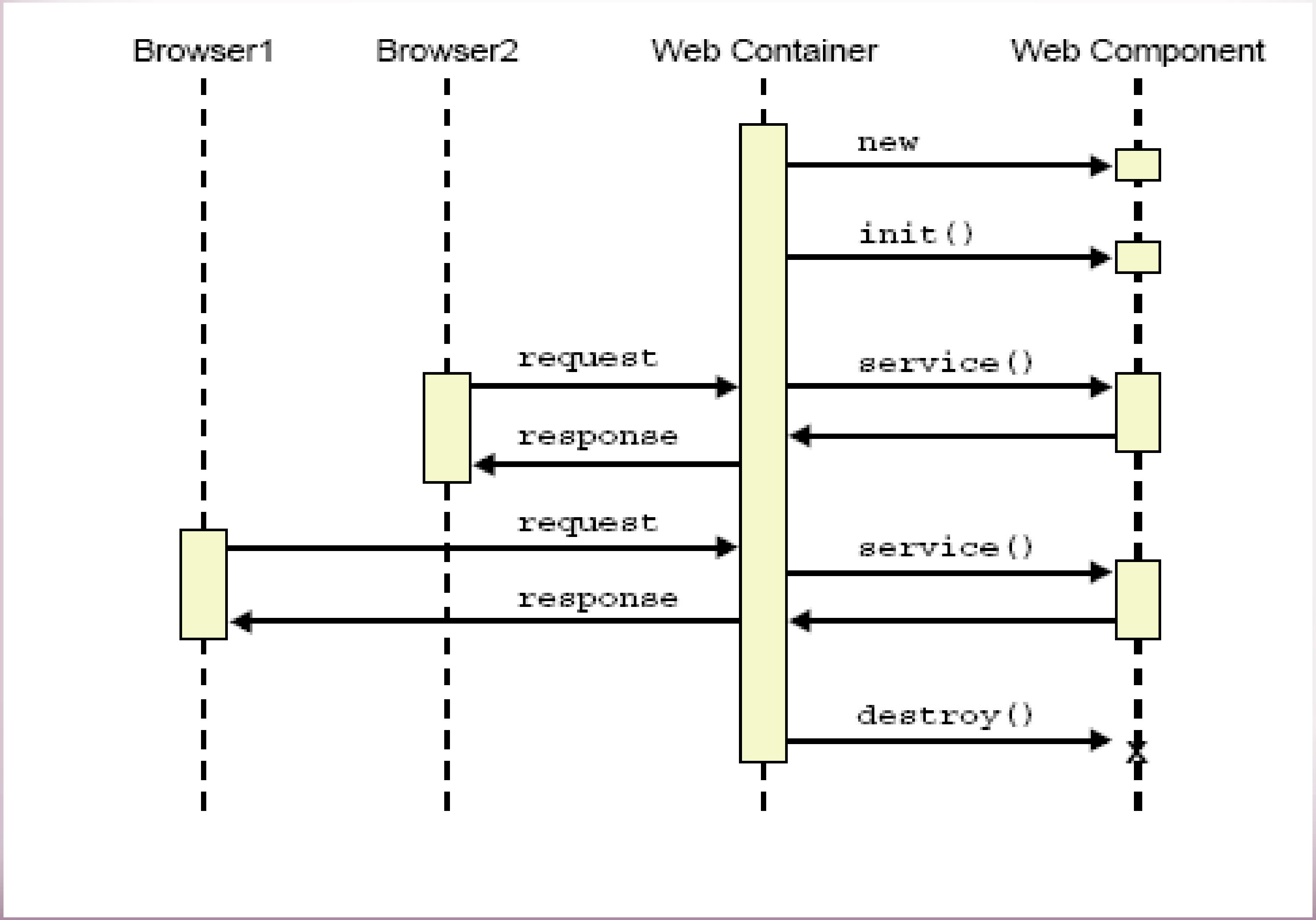
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {

 public void doGet ( HttpServletRequest        request, HttpServletResponse
            response    ) throws ServletException, IOException
{
     PrintWriter out =response.getWriter();
     response.setContentType("text/html");

     out.println("<HTML><HEAD><TITLE>");
     out.println(title);
     out.println("</TITLE></HEAD><BODY>");
     out.println("<P>This is output from SimpleServlet.");
     out.println("</BODY></HTML>");
}
  }
```

Demo1.1

Servlet API


<<Interface>>
Servlet
GenericServlet
{abstract}
HttpServlet
Your HTTP Servlet
<<Interface>>
ServletRequest
<<Interface>>
HttpServletRequest
Vendor's
Request Object
Generic
API Level
Protocol-Specific
API Level
Implementation
Level

Life Cycle of a Servlet

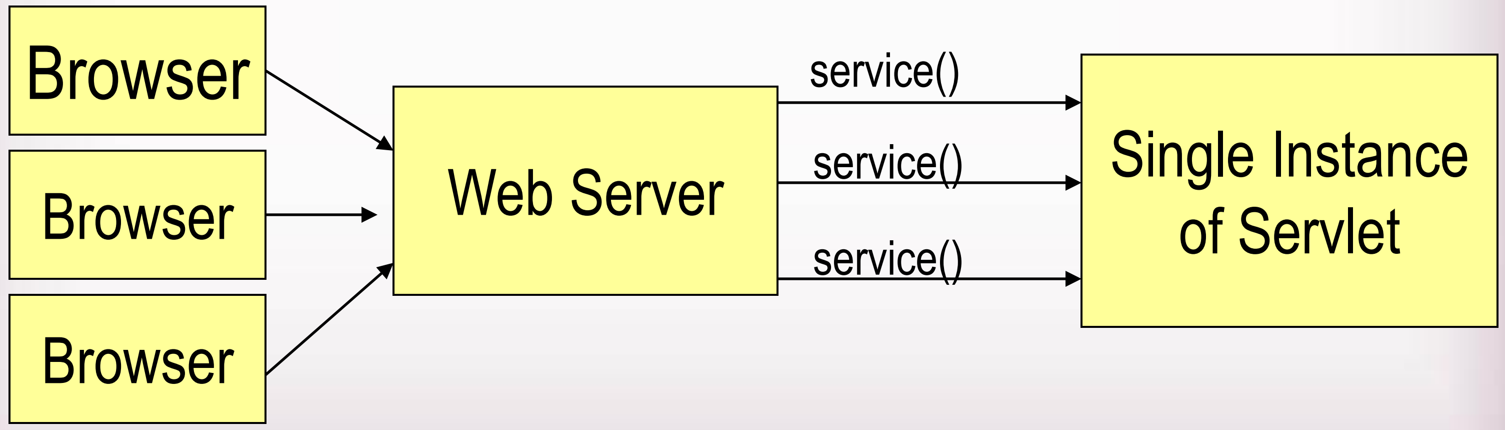


The init() method

- The init() method is called when the servlet is first requested by a browser request.
- It is <u>not</u> called again for each request.
- Used for <u>one-time initialization</u>.
- The init() method is a good place to put any initialization variables

Service() Method

- Each time the server receives a request for a servlet, the server spawns a new thread and calls the servlet's service () method.



The Service Method

- By default the service() method checks the HTTP Header.
- Based on the header, service calls either doPost() or doGet().
- doPost and doGet is where you put the majority of your code.
- If your servlets needs to handle both get and post identically, have your doPost() method call doGet() or vice versa.

Death of a Servlet

- Before a server shuts down, it will call the servlet’s destroy() method.
- You can handle any servlet clean up here. For example:
  - Updating log files.
  - Closing database connections.
  - Closing any socket connections.

ServletRequest interface

getParameter( )

- Used to get the input from the request Object , Returns a String

getParameterValues( )

- Used to get the input from the request Object , Returns a String Array

getParameterNames( )

- Used to get the input from the Request Object ,Return an Enumeration

Parameter and Parameter Values

```
<form action="RegisterServlet" method="get">
User Name:
<input type="text" name="username"/>
<p/>
Subjects of Interest
<p/>
Chenai :
<INPUT type="checkbox" name="subject" value="Java" />
Mumbai
<INPUT type="checkbox" name="subject" value="J2EE"/>
Hydrabad
<INPUT type="checkbox" name="subject" value="Struts"/>
<p/>
<input type="submit" value="Register"/>
</form>
```

Parameter and Parameter Values

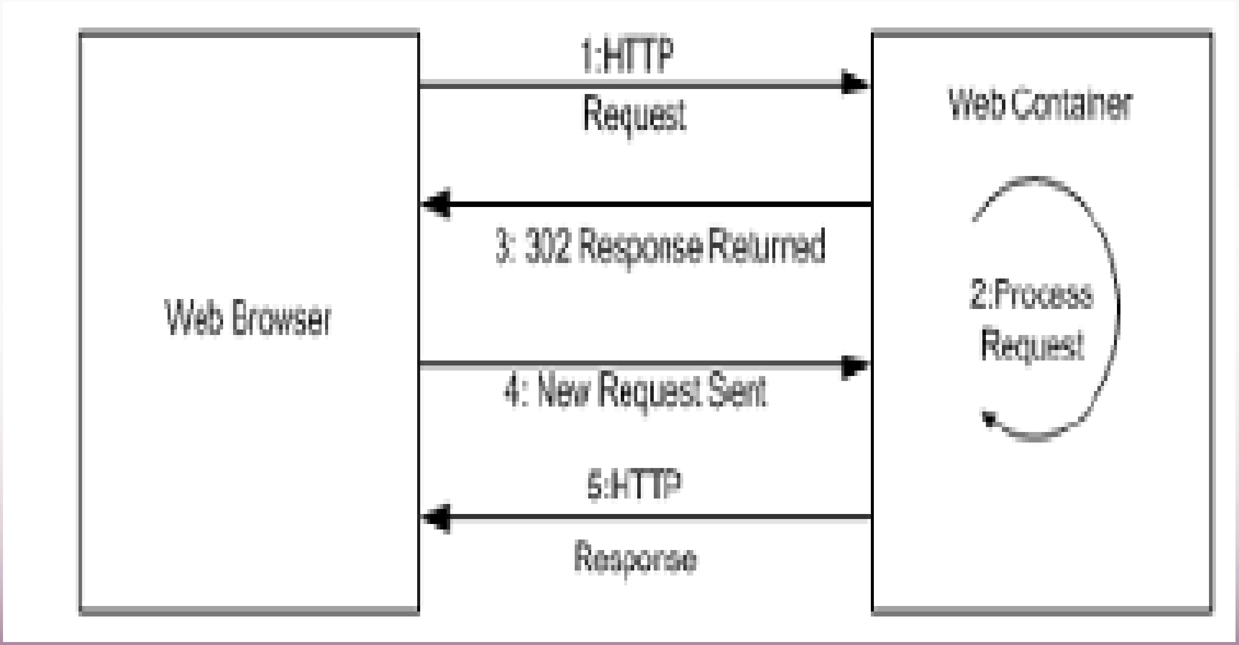
```
public void doGet(HttpServletRequest req,
   HttpServletResponse resp)
    throws ServletException, IOException {

            PrintWriter out = resp.getWriter();

             resp.setContentType("text/html");

      String uname = req.getParameter("username");

       String[] topics = req.getParameterValues("subject");

     out.println("Thanks For Registering ");
     out.println("<h4>"+uname+"</h4>");
     out.println("Yours Topic of Interst Include");

       for(int i=0;i<topics.length;i++) {
             out.println(topics[i]);
     }
   }
```

Inter-Servlet Communication

- A process where two or more servlet communicates with each other to process the client request.

Send Redirect

- When a sendRedirect() method is invoked, it causes the web container to return a response back to the browser with information that a new URL should be requested.
- Because the browser issues a completely new request, any objects that are stored as request attributes before the redirect occurs, will be lost.



Send Redirect

- Send the request with new url back to the client which connects to this URL thereby creating a new pair of request/response objects
- It gives more flexibility because it can redirect the user any URL while forward works only within the same web application
- It takes a String as arguments
- It should be called before writing to the response or before sending the response., else it will throw a IllegalStateException
- response.sendRedirect(http://www.google.com);
- Can also take relative url, with a slash (/) meaning relative to the root of the web app

Demo:sendRedirect

```
String s1=req.getParameter("t1");
String s2=req.getParameter("t2");

if(!((s1.equals("INDIA"))&&(s2.equals("INDIA"))))
{
  res.sendRedirect("http://www.google.com");
}
else
{
 out.println("welcome to our website<br>");
 out.println("visit our Web Site");
out.println("<a href=http://localhost:7001/>Click Here</a>");
}
```

Scope Object

- Enables sharing information among collaborating web components
- Attributes are maintained in scope objects
- Attributes of scope objects are accessed with
  - getAttributes()
  - setAttribute()

- The Four scope object are

- Web Context (Servlet context)
  - Accessible from web components
- Session
  - Accessible from web components handling a request that belong to the session
- Request
  - Accessible from web components handling the request Page
  - Accessible from JSP page that creates the object

ServletContext interface:

- Defines a set of methods that a servlet uses to communicate with its servlet container
- There is one context per "web application" and is accessible to all active resource of that application
- The ServletContext object enables to set, get and change Web application-scope attribute values.
- Used to get Server Information , like name, version of the container and the API being used.

ServletContext interface:

- **<u>public void setAttribute(String, Object)</u>**
- Binds the object with a name and stores the name/value pair as an attribute of the ServletContext object. If an attribute already exists, then this method replaces the existing attribute.

- **<u>public Object getAttribute(String attrname)</u>**
- Returns the Object stored in the ServletContext object with the name passed as a parameter.

Methods of ServletContext interface

- public Enumeration getAttributeNames()
  - Returns an Enumeration of String object that contains names of all the context attributes.
- public String getInitParameter(String pname)
  - Returns the value of the initialization parameter with the name passed as a parameter.
- public Enumeration getInitParameterNames()
  - Returns an Enumeration of String object that contains names of all the initialization parameters.

Attributes and Parameters

- **Attributes :**
  - Types: Application/context , Request, Session
- Method :
  - setAttribute(String Name,Object Value)
  - Object getAttribute(String)
- **Parameters**
  - Application/Context/Init parameters
  - Request Parameters
  - Servlet Init Parameters
- Methods:
  - getInitParameter(String name);
  - Cannot Set Attributes
  - Return Type is String

Session Management

- Session management is the technique by which the link between different request and responses of different pages of a session are maintained.

- The Option for Tracking Client Details:
  - Use a Stateful Session Bean
  - Use a Database
  - Use an HttpSession

Session

•A Persistent Network Connection Between Two Hosts

•A Session will be created When a connection to the server is established and will be closed when the connection is closed

•Every Session will have a Unique Session-Id

•A Session is timed out Based on the Time out Value that depends upon the Server

Session Tracking

<u>With Cookie</u>

A Session object is created to hold all the values that are selected and also encapsulates the session ID Details

<u>Without Cookie</u>

When a Cookie is not available the state is maintained by Rewriting the URL

The State can be Maintained by Passing the values Using Hidden Fields

Life Cycle of a Cycle of Session

- The client request a resource from the server.
- The server returns a valid session id that allows the client to be uniquely identified.The session id is sent in a cookie that will be returned by the client with every request.
- The client issues any number of requests to the server. Every request must accompany the cookie.
- The client logs out.
- The session id will be removed from the cookie file.
- This concludes the session.

Sending a Session Cookie

- HttpSession session = request.getSession();
- To Know whether the session exisits or new

  If(session.isNew()) {

  }

  – Request.getSession(true)

    Returns a Session if the session preexist, or creates a new session and returns it
  – Request.getSession(false)

    Return a preexisting session, or null if there was no session.

URL Based Session ID

• Some browsers do not support cookies or allow the user to disable cookie

• Session Management resorts to a second method, URL rewriting,

•With URL rewriting, links returned to the browser or redirect have the session ID appended to them.

•***response.encodeURL("/myservlet");***

•The rewritten form of the URL is sent to the server as part of the client's request.

•Session ID information embedded in the URL, which is received by the application through HTTP GET requests

URL Rewriting

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

     HttpSession sess = request.getSession(true);
     PrintWriter out = response.getWriter();
     out.println("Inside First Servlet");
     out.println("New Session  = ");
     out.println(sess.isNew());
     out.println("<a href=\"");
     out.println(response.encodeURL ("Second"));
       out.println("\">Second</a>");
     }
```

URL Rewriting

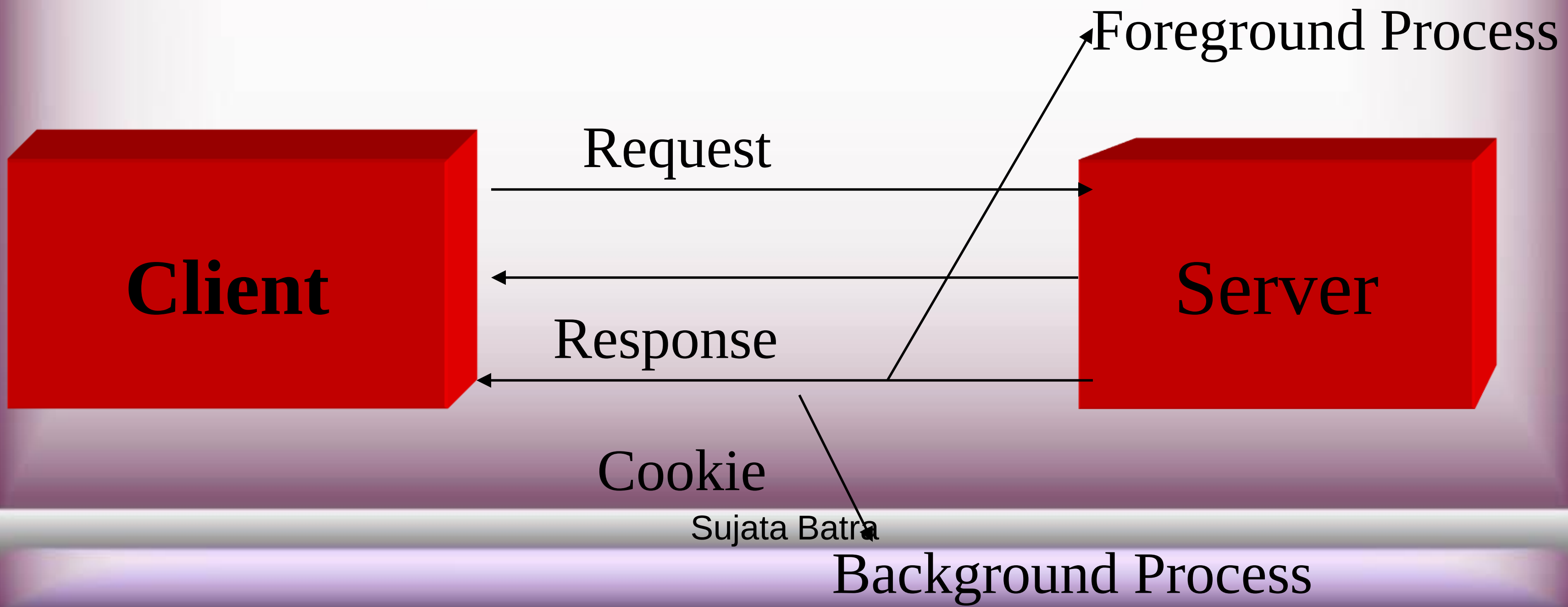
```
protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {

HttpSession sess = request.getSession(true);

PrintWriter out = response.getWriter();

out.println("Inside Second  Servlet");

out.println("New Session  = ");
out.println(sess.isNew());
}
```

COOKIE

•HTTP Mechanism that handles specific user settings and manages state

•Useful in tracking individual users as they traverse across the website

•Used to store and retrieve user specific information



Methods defined in the Cookie class

- public String getName()
  - Returns the name of the cookie.
- public void setMaxAge(int expiry)
  - Sets the maximum time for which the client browser retains the cookie value.
- public int getMaxAge()
  - Returns the maximum age of the cookie in seconds.
- public void setValue(String value)
  - Sets a new value to the cookie.
- public String getValue()
  - Returns the value of the cookie.

HTTPSession Methods

- getCreationTime()
  - Return the time session was created
- getLastAccessedTime()
  - Return the last time the container got a request with this session id , in milliseconds
- setMaxInactiveInterval()
  - Specifies the maximum time in seconds, that you want to allow between client requests for the session
- getMaxInactiveInterval()
  - Returns the maximumtime in seconds , that is allowed between client requests for the session
- Invalidate()
  - Ends the session, unbinds all the attributes with the session

Methods of HttpSession interface

- public Enumeration getAttributeNames()

- Returns the name of all the objects that are bound to the session object.

- public void removeAttribute(String name)

- Unbinds the session object from the attribute, name specified in the method.

- public String getId()
- Returns a string that contains the unique identifier associated with the session.

Session Timeout

- **times out**
- Configuring in web.xml

  <session-config>

  <session-timeout>15</session-timeout>
- The time is set in minutes

- Setting through program – only for that particular session instance
  - session.setMaxInactiveInterval(20*60)
- The time is set in seconds

- **Call Invalidate on the seesion Object**
  - session.invalidate()

- The application crashes or undeployed

Need for Connection Pooling

- Database connection is an expensive and limited resource
  - Using connection pooling, a smaller number of connections are shared by a larger number of clients
- Creating and destroying database connections are expensive operations
  - Using connection pooling, a set of connections are pre-created and are available as needed basis cutting down on the overhead of creating and destroying database connections

JDBC *connection pool*:

- Is a ready-to-use pool of established connections to a JDBC database.
- Is accessed via a *DataSource*, which is bound into the JNDI tree.
- Eliminates the overhead of creating separate database connections for each application. A Pool of connection is shared and re-used across applications .
- It determines when a connection is in use and It creates more connections when needed.
- When the pool is closed all the connections are released

Connection Pool….

Features of a connection pool include:

- Configurable initial and maximum number of connections in the pool, set by: `InitialCapacity` and `MaxCapacity`.
- *Automatic* connection testing and rebuilding to ensure connections remain healthy,
    - Test newly created connections.
    - Test unused connections periodically.
    - Test reserved connections.
    - Test released connections.
- The ability to *manually* test a connection pool to ensure they remain healthy.

JNDI

- A generic mechanism for Java components to find other components, resources, or services indirectly at runtime.
- It provide an *indirection layer*, so that components can find required resources without being particularly aware of the indirection.
- A driver that is accessed via a DataSource object does not register itself with the DriverManager
- Direct references to the data source, JDBC driver class names, user names and passwords or eliminated
- DataSource object is registered to JNDI naming service by the container and then retrieved by a client though a lookup operation

Data source

- A data source object enables JDBC applications to obtain a DBMS connection from a connection pool.
- Each data source object binds to the JNDI tree and points to a Connection pool
- Applications look up the data source on the JNDI tree and then request a connection from the data source.
- Data Source objects that implement connection pooling also produce a connection to the particular data source that the Data Source class represents
- The connection object that the getConnection method returns is a handle to a Pooled Connection object rather than being a physical connection

Data Source

- Driver vendor implements the interface

- Data Source object is the factory for creating database connections

- A Data Source object has properties that can be modified when necessary – these are defined in a container's configuration file

  - location of the database server
  - name of the database
  - network protocol to use to communicate with the server

- These properties can be changed, any code accessing that data source does not need to be changed

Context and Initial Context

- A context Interface represents a set of bindings within a naming service that share the same naming convention.
- A Context object provides the methods for binding, unbinding, listing names to objects
- JNDI performs all naming operations relative to a context.
- To assist in finding a place to start, the JNDI specification defines an InitialContext class.
- This class is instantiated with properties that define the type of naming service in use and, for naming services that provide security, the ID and password to use when connecting.

META-INF/context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/JNDI_JDBC" docBase="JNDI_JDBC">
 <Resource name="jdbc/ds1"
   auth="Container" type="javax.sql.DataSource"
           url="jdbc:mysql://localhost:3306/myexamples"
           username="root" password="srivatsan"
         driverClassName="org.gjt.mm.mysql.Driver"
  maxActive="20" maxIdle="10"/>
</Context>
```

Deploy /oracle-ds.xml

- <datasources>
- <local-tx-datasource>
- <jndi-name>DefaultDS</jndi-name>
- <connection-url>jdbc:oracle:thin:@localhost:1521:xe</connection-url>
- <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
- <user-name>SYSTEM</user-name>
- <password>jboss</password>
- <valid-connection-checker-classname>
- org.jboss.resource.adapter.jdbc.vendor.OracleValidConnectionChecker</validconnection-
- checker-class-name>
- <metadata>
- <type-mapping>Oracle9i</type-mapping>
- </metadata>
- </local-tx-datasource>
- </datasources>

Deploy/mysql-ds.xml

- <datasources>
- <local-tx-datasource>
- <jndi-name>DefaultDS</jndi-name>
- <connection-url>jdbc:mysql://localhost:3306/test</connection-url>
- <driver-class>com.mysql.jdbc.Driver</driver-class>
- <user-name>root</user-name>
- <password>jboss</password>
- <valid-connection-checker-classname>
- org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker</validconnection-
- checker-class-name>
- <metadata>
- <type-mapping>mySQL</type-mapping>
- </metadata>
- </local-tx-datasource>
- </datasources>

Obtaining Initial Context

-
- **try {**
- IntitialContext ic = **new InitialContext();**
- DataSource ds = (DataSource)ic.lookup("java:/DefaultDS");
- Connection con = ds.getConnection();
- PreparedStatment pr = con.prepareStatement("SELECT USERID, PASSWD FROM JMS_USERS");
- ResultSet rs = pr.executeQuery();
- **while (rs.next()) {**
- out.println("<br> " +rs.getString("USERID") + " | " +rs.getString("PASSWD"));
- }

Two Ways of Output

- There are two streams to choose from
  - PrintWriter for character output
  - ServletOutputStream for bytes output

  - ServletOutputStream out=response.getOutputStream()
  - Out.Write(abyteArray);

Demo9.1

Two Ways of Output

```
BufferedInputStream buf = null;
 try {
   ServletOutputStream stream =     response.getOutputStream();
   File doc = new File(“/eval.doc”);
   response.setContentType("application/msword");
   FileInputStream input = new FileInputStream(doc);
   buf = new BufferedInputStream(input);
   int readBytes = 0;
   while ((readBytes = buf.read()) != -1)
     stream.write(readBytes);
```

Methods in the HttpServletRequest interface:

- public String getHeader(String fieldname)
- Returns the value of the request header field such as Cache-Control
- and Accept-Language specified in parameter

- public Enumeration getHeaders(String sname)
- Returns all the values associated with a specific request header as
- an Enumeration of String objects

- public Enumeration getHeaderNames()
- Returns the names of all the request headers that a servlet can access as an Enumeration of String objects.

Demo: getHeader()

```
public class HostInfo extends HttpServlet {

    public void doPost(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
    {
            String rhost = req.getRemoteHost();
            String raddr = req.getRemoteAddr();

            PrintWriter pw = res.getWriter();
            res.setContentType("text/html");
            pw.println("<H3>Details of client machine from which the request is
    coming:</H3>");
            pw.println("<B>Hostname of Client Machine : </B>" + rhost + "<BR>");
            pw.println("<B>IP Address of Client Machine : </B>" + raddr);
    }
}
```

getHeaderNames()

```
public class RequestHeaderExample extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Enumeration e = request.getHeaderNames();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = request.getHeader(name);
            out.println(name + " = " + value          );
        }
    }
}
```

Java Server Pages - JSP

Introduction to JSP Technology

- JSP pages, by virtue of the separate placement of the static and dynamic content, facilitates both Web developers and the Web designer to work independently.

- Facilitates the segregation of the work profiles of a Web designer and a Web developer.

  –A Web designer can design and formulate the layout for a Web page by using HTML.

  –A Web developer, working independently, can use Java code and other JSP specific tags to code the business logic.

Servlets and JSP

- Servlets tie up files to independently handle the static presentation logic and the dynamic business logic.
- Servlet programming involves extensive coding.
- Changes made to the code requires identification of the static code content and dynamic code content to facilitate incorporation of the changes.

- JSP allows Java to be embedded directly into an HTML page by using special tags.
- JSP pages, by virtue of the separate placement of the static and dynamic content, facilitates both Web developers and the Web designer to work independently.

JSP is also a Servlet

1. Jsp file is Written by the web page author

2. Jsp file is TRANSLATED to file.java by the container

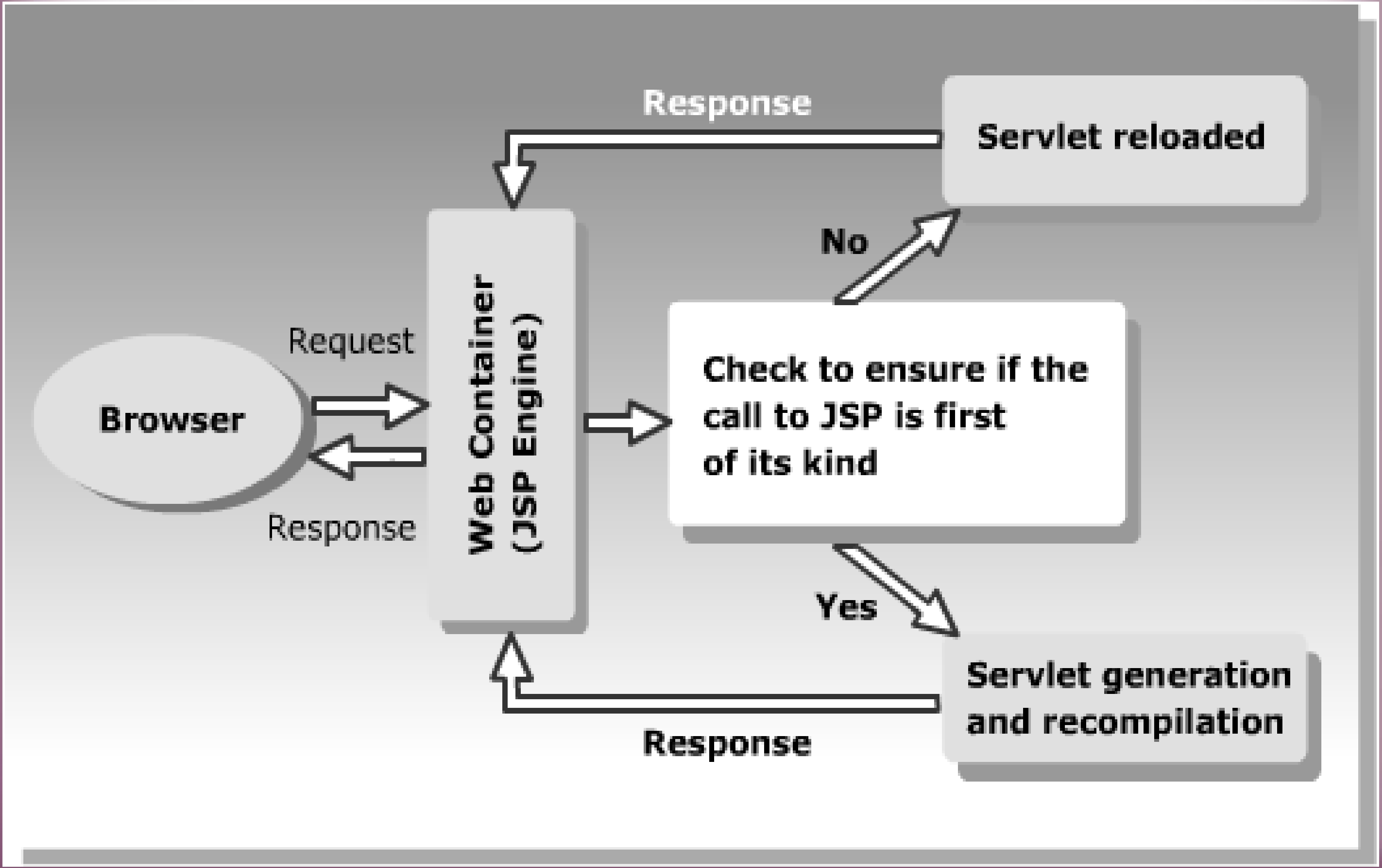
3. File.java COMPILED to file.class container

4. File class is loaded by the container an initialized as Servlet

5. The container instantiates the servlet and cause the servlets is jspInit() method

6. The container creates a new thread to handle the clients request and the servlets _jspService() method runs

JSP Life Cycle


Response
Servlet reloaded
No
Request
Browser
Response
Web Container (JSP Engine)
Check to ensure if the call to JSP is first of its kind
Yes
Servlet generation and recompilation
Response

sample JSP Page:

```
<html>
<head>
<title>Java Server Page</title>
</head>
<body>
<center>
<%
out.println("<b>Hello From Simple JSP Page</b>");
%>
</center>
</body>
</html>
```

JSP Content

The three components of a JSP page are:

- **JSP life cycle methods**

- jspInit(): Is invoked at the time when the servlet is initialized.
- _jspService(): Is invoked when request for the JSP page is received.
- jspDestroy(): Is invoked before the servlet is removed from the service

- **Jsp Components**

- JSP scripting
- JSP directives
- JSP actions

Jsp Scripting

- Scriptlets
  - Similar to code placed inside the service method

- Declarations

  - Similar to code placed globally for a class to use

- Expressions
  - Similar to print() method calls

JSP Scripting Elements

- Valid Java Code Embedded directly into an HTML page.
- **JSP Syntax:**
    - ***<% code %>***
- Java code is encapsulated within the opening and closing JSP tags.
- If It needs to produce output ,can use the implicit **out** object.

JSP Scripting Elements

```
<html>
<head>
</head>
<body>
<%
out.println("Hello From Jsp Standard Scriptlet ");
%>


</body>
</html>
```

Scriptlet-JSP Syntax

Scriptlet-XML Syntax

Expressions

- **JSP Syntax:**
  - ***`<%= code %>`***

- Expressions begins with the JSP start tag followed by an equals sign

- *Does not end in a semicolon*, unlike other Java Statements

Declaration Element Syntax

- **JSP Syntax:**
  - ***<%! code %>***

- Used initialize variables and methods

- *Scriptlets, Expressions. Variables and methods created within Declaration elements are global*

- Begins exclamation point (<%!) and ends with a semicolon.

Declaration & Expression

```
<html>
<head>
</head>
<body>
<%! public String showdata()
{
return "JSP Pages ";
}
%>
<font size=2><b><%=showdata()%></b></font>
<%! String str="welcome";
int a=100;
int b=a+200;
%>
The given string is <%=str%>
The given integer is <%=a%>
Computed integer is <%=b%>
```

Declaration

Expression

Decision-Making Statements

```
<body>
<form action="ConditionChecking.jsp">
  Option :
   <select name="option">
          <option value="First">First</option>
          <option value="Second">Second</option>
    </select>

  <input type="submit" value="Select">

</form>
```

Decision-Making Statements

```
<body>
<%
  String option =request.getParameter("option");
    if(option.equalsIgnoreCase("First"))
    {
 %>
    <a href="First.jsp">First</a>
    <% }   else   {    %>
    <a href="Second.jsp">Second</a>
    <%} %>
</body>
```

Loop Statements

```
<body>
<%
 for(int i=0;i<=10;i++) {
%>

<font size="<%=i%>">
          Welcome to Jsp Programming
</font>


<% }  %>


</body>
```

Two Types of Comments

- **<!– HTML Comment -- >**
  - The Container passes this to the client.
  - The browser interprets it as comment and hide them while displaying the page,
  - Ignored by both the JSP engine and the browser, and is left intact.
- **<% -- JSP Comment --% >**
  - This is a JSP or "hidden" comment.

<!– HTML Comment -- >

```
<html>
<head>
<title>HTML Comment </title>
</head>
<body>

<!--  A Simple Jsp Comment -->

<!--
  Commented Code <%=request.getParameter(“username”) %>
 -->

</body>
</html>
```

Html Comment Output– View Source

```
<html>
<head>
<title>HTML Comment </title>
</head>
<body>

<!--  A Simple Jsp Comment -->

<!--
    Commented Code null
 -->

</body>
</html>
```

<% -- JSP Comment --% >

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

<%-- A Simple Jsp Comment --%>

<%--
  Commented Code <%=request.getParameter(“username”) %>
--%>

</body>
</html>
```

JSP Comment Output-View Source

```
<html>
<head>
<title>JSP Comment </title>
</head>

<body>




</body>
</html>
```

JSP Implicit Objects

- Pre-defined variables that can be included in JSP expressions and scriptlets.
- Implemented from servlet classes and interfaces.

- **The out Object**

  – This is output stream is exposed to the JSP author through the implicit out object.

  – The out object is an instantiation of a javax.servlet.jsp.JspWriter object.

Request Object

- When a client requests a page a new object to represent that request.
- Its an instance of **javax.servlet.http.HttpServletRequest** and is given parameters describing the request.
- Stored in special name/value pairs that can be retrieved
  - **`request.getParameter(name)`**
- The request object also provides methods to retrieve header information and cookie data.
  - **`request.getRequestURI() and`**
  - **`request.getServerName() to identify the server).`**

Response Object

- This object is used to represent the response to the client.
- Its an **javax.servlet.http.HttpServletResponse**
- Deals with the stream of data sent back to the client.
- The **out** object is related to the response object.
- Defines the interfaces that deals
  - **HTTP headers.**
  - **cookies**
  - **change the MIME content type of the page**
  - **HTTP status codes**

The session object

- Used to track information about a particular client while using stateless connection protocols, such as HTTP.
- Can be used to store arbitrary information between client requests.
- Each session should correspond to only one client and can exist throughout multiple requests.
- Sessions are often tracked by URL rewriting or cookies,
- The session object is an instance of javax.servlet.http.HttpSession and behaves exactly the same way that session objects behave under Java Servlets.

Session – Set/Get Attribute

```
<html>
<head>
</head>
<body>
<%if((session.getAttribute("validUser"))==null) {

session.setAttribute("validUser","yes");
%>
<form action="Validate.jsp">
user name
<input type="text" name="username"/>
<input type="submit" value="Register"/>
</form>

<%} else {%>out.println("Invalid Access-Try Again") <%}
  %>
</body>
</html>
```

Session – Set/Get Attribute

```
<body>
<% if((session.getAttribute("validUser"))!=null)
{
String sessAtt=
  (String)(session.getAttribute("validUser"));
if(sessAtt.equalsIgnoreCase("yes"))
{ out.println("Valid Access");
session.invalidate();
}  }
else  {
out.println("Invalid Access"); %>
<a href="Login.jsp">Login</a>
<%} %>
<p>Validate Page </p>
</body>
```

The application Object

- This is an wrapper around the ServletContext object for the generated Servlet.
- This object is a representation of the JSP page through its entire lifecycle.
- This object is created when the JSP page is initialized and will be removed when the JSP page is removed by the jspDestroy() method, the JSP page is recompiled, or the JVM crashes.
- Information stored in this object remains available to any object used within the JSP page.

config

- The config object is an instantiation of javax.servlet.ServletConfig.
- This object is a direct wrapper around the ServletConfig object for the generated servlet.
- It has the same methods and interfaces that the ServletConfig object does in programming Java Servlets.
- This object allows the JSP author access to the initialization parameters for the Servlet or JSP engine. This can be useful in deriving standard global information, such as the paths or file locations.

Scope Objects- Application and Config

```
<body>
<center>
<h1> Scope Objects- Application and  Config </h1>
Java Version to be Used
<%=application.getInitParameter("java_version") %>
<p/>
Web Server Version
<%=application.getMajorVersion() %>
<%=application.getMinorVersion() %>
<p/>
Server Info
<%=application.getServerInfo() %>
<hr>
Contact our Administator at
  <%=config.getInitParameter("email_id") %>
</center>
</body>
```

Page

- This object is an reference to the instance of the page.
- An object that represents the entire JSP page.
- When page is first instantiated the page object is created by obtaining a reference to the this object.
- The page object is really a direct synonym for the “this” object.
- During the JSP lifecycle, the this object may not refer to the page itself.
- Within the context of the JSP page, The page object will remain constant and will always represent the entire JSP page.

Directive Statement

Components of a JSP Page-Directives

- A directive element in a JSP page provides global information about a particular JSP page and is of three types:
    - page Directive
    - taglib Directive
    - include Directive
- The syntax for defining a directive is:
- <%@ directive attribute=”value” %>

Page Directive Attribute

- Defines attributes that notify the Web container about the general settings of a JSP page.
- The syntax of the page directive is:
  - <%@ page attribute_list %>

This can be used to import packages too – To import a single package

<%@ page import=“java.io.*” %>

To import multiple packages

Page Directive Attribute

- errorPage
  - Specifies that any un-handled exception generated will be directed to the URL.
- isErrorPage
  - Specifies that the current JSP page is an error page, if the attribute value is set to true. The default value of isErrorPage attribute is false.

Error Handling

```
<BODY>

   <form action="Validate.jsp">

     User Name :
        <input type="text" name="username"/>
     PassWord :
         <input type="password" name="password"/>

     <input type="submit" value="Register"/>

   </form>

</BODY>
```

Error Handling - Attribute - errorPage

```
<HEAD>
        <%@ page errorPage="MyExceptionHandler.jsp"%>
</HEAD>

   <BODY>

        <%
        String a = request.getParameter("username");
        int b = Integer.parseInt(a);
        %>

   </BODY>
```

Error Handling - Attribute -isErrorPage

```
<HEAD>

   <%@ page isErrorPage="true"%>

</HEAD>

   <BODY>

     Error Page

     <%=exception.getMessage()%>
     <%=exception.getClass()%>

   </BODY>
```

The include Directive

- Specifies the names of the files to be inserted during the translation of the JSP page.
- Creates the contents of the included files as part of the JSP page.
- Inserts a part of the code that is common to multiple pages.
- There are two include directive in JSP
    - **`<%@ include file =" " %>`**
    - **`<jsp:include page =" " />`**
- Include directive happens at translation time and <jsp:include> happens at runtime.

<include> and <jsp:include>

- `<%@ include` ***file***`="Header.jsp %>`
  - It takes the contents of the Header.jsp and places it into the calling jsp BEFORE it does the translation of the jsp page
  - This inserts the SOURCE of "header.jsp" at translation time
- `<jsp:include` ***page***`="Header.jsp />`
  - In case of the <jsp:include> the orginal header.jsp file is NOT inside the generated

The taglib Directive

- Imports a custom tag into the current JSP page.
- Associates itself with a URI to uniquely identify a custom tag.
- Associates a tag prefix string that distinguishes a custom tag with the other tag library used in a JSP page.
- The syntax to import a taglib directive in the JSP page is:
  - <%@ taglib uri=“tag_lib_URI” prefix=”prefix” %>

Scope of JavaBean in a JSP page

- **Page :**
  - Default scope is to page scope.
- **request:**
  - Specifies that the JavaBean object is available for the current request.
- **session:**
  - Specifies that the JavaBean object is available only for the current session.
- **application:**
  - Specifies that the JavaBean object is available for the entire Web application.

EL
(Expression Language)

Scriptless JSP

- A JSP page without any scriptlets is a script-less JSP.
- Our attempt is to reduce the clutter of java code in the JSP
- With standard jsp action tags we can achieve some script-less-ness.
  - But surely this is not enough.
  - *For instance how will you print request parameters in a script-less way?*

Need for EL

Take another example-

Suppose we have a class Car which has an attribute called engine of class Engine.

Let us attempt to display the attribute of the engine object given the Car object but in a scriptless way.

```
public class Car{
Engine engine;
String model;
//other attributes …
//getters and setters methods…
}
class Engine{
String engine_type;
int  engine_cc;
//other attributes….
public getEngine_type(){…}
public getEngine_cc
// other getter and setter methods….
}
```

Attempts to display engine’s engine_type attribute

- Using standard actions

```
<jsp:useBean id=“car” class=“Car” scope=“request”
/>
Engine Type:<jsp:getProperty name=“car”
property=“engine.engine_type”>
```

NO THIS WILL RESULT IN AN ERROR

- Using scripting

```
<%=
(Car)request.getAttribute(“car”).getEngine.getEngi
ne_type() %>
```

THIS IS FINE BUT REMEMBER WE SAID WE WILL ATTEPT SCRIPTLESS JSP.

SO WE CONCLUDE

STANDARD ACTIONS ARE NOT ENOUGH TO MAKE JSP SCRIPTLESS

EL to rescue!

- Using EL to solve the problem

```
<body>
<jsp:useBean id="car" class="Car"
scope="request" />
 ${car.engine.engine_type}
</body>
```

instead of `<jsp:getProperty>`

`But what is EL?`

EL

- Expression language
- Primary feature of JSP technology version 2.0
- EL expressions can be used:
  - In static text
    - The value of an el expression in static text is computed and inserted into the current output
  - In any standard or custom tag attribute that can accept an expression
    - If the static text appears in a tag body, the expression will not be evaluated if the body is declared to be tagdependent

EL Syntax

${E1.E2}

EL implicit Object OR an attribute either in page, request, session or application scope (searched in that sequence)

If E1 is implicit object , then E2 is Attribute

If E1 is Attribute, then E2 is the property of the object.

Example

Bean

```
package c;
import java.io.Serializable;
public class Emp implements Serializable{
private String empno;
private String name;
public Emp(){
empno="402";
name="Rama";}
public void setEmpno(String empno){
  this.empno=empno;}
public void setName(String name){
  this.name=name; }
public String getEmpno(){return empno;}
public String getName(){
  return name;
}}
```

JSP:

```
<jsp:useBean class="c.Emp" id="a"/>
<jsp:setProperty name="a"  property="empno"
value="Mr. ${a.name} ${a.empno}"/>
<jsp:getProperty name="a" property="empno"/>
```

The expressions are evaluated from left to right. Each expression is evaluated to a String and then concatenated with any intervening text. The resulting String is then coerced to the attribute's expected type.

Displays

```
Mr. Rama 402
```

EL Implicit Objects

- **`pageScope`**
- **`requestScope`**
- **`sessionScope`**
- **`applicationScope`**
- **`param`**
- **`paramValues`**
- **`header`**
- **`headerValues`**
- **`cookie`**
- **`initParam`**
- **`pageContext`**

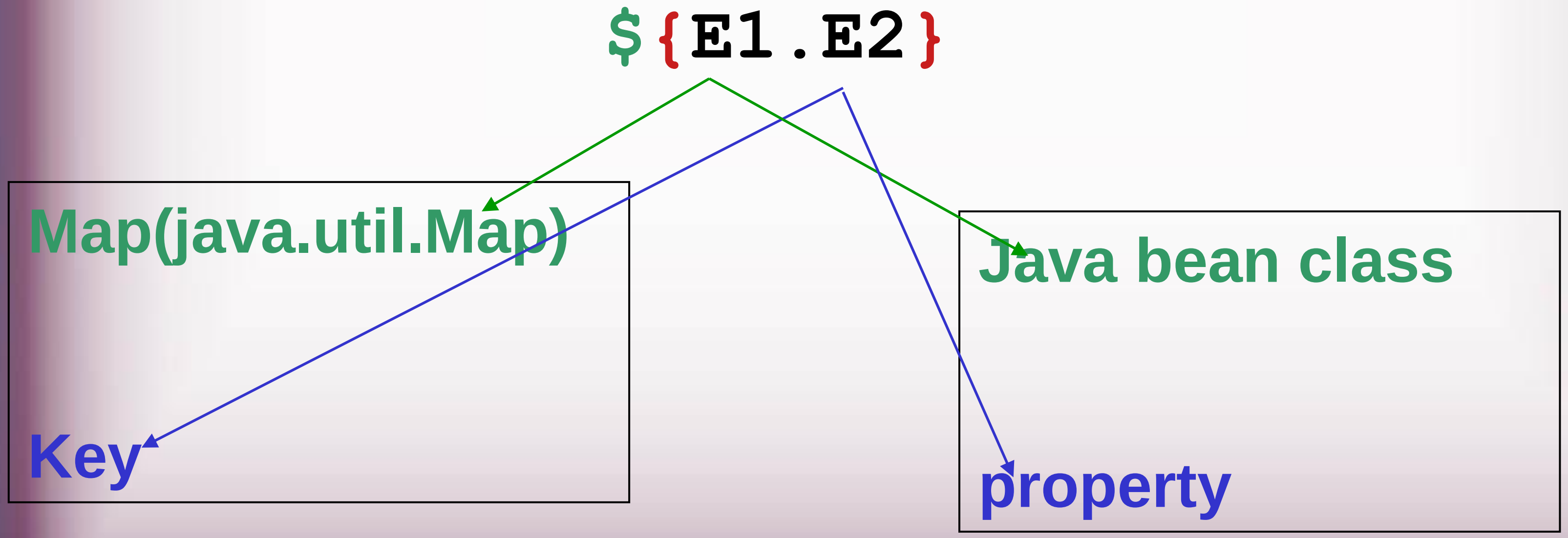
•Note that these objects are not the same as JSP implicit object

Map objects

Like JSP implicit object

Using "." Operator

- When the variable is on the left side of the dot, it is either a Map(Key/Value pair) or a bean(properties)



Examples

- *We have seen an example for java bean with property*

```
<jsp:useBean class="c.Emp" id="a"/>
 ${a.name}
```

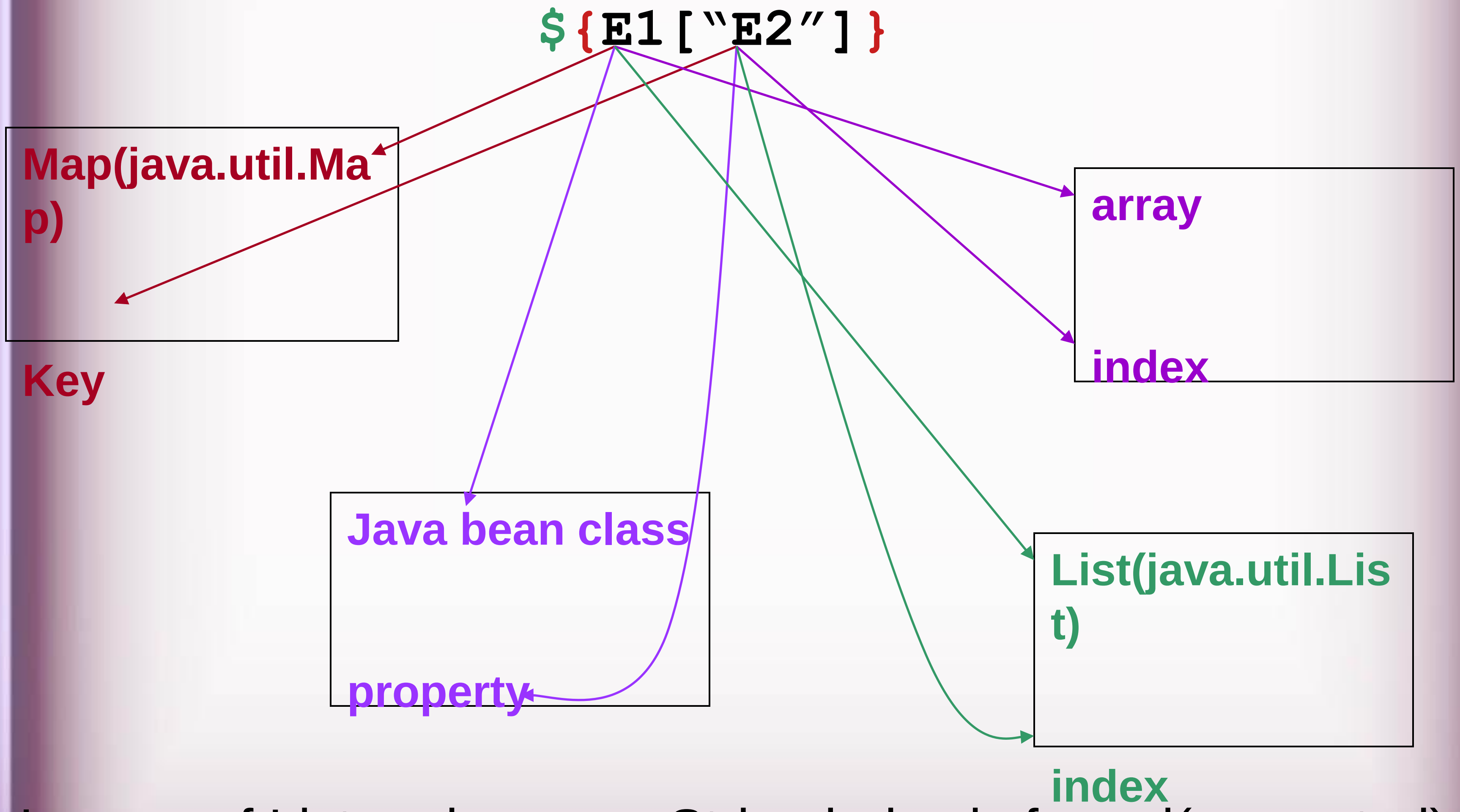
- Map Example

```
<jsp:useBean class="c.Emp" id="a"/>
 ${pageScope.a.name}
```

  – Where 'pageScope' is implicit object of map type.

Using “[ ]” operator

- When the variable is on the left side of the [], it can be
  - ➢ Map
  - ➢ bean
  - ➢ List
  - ➢ array



•In case of List and arrays , String index is forced(converted) to an int.

Example using [] and .

```
protected void doGet(…){
java.util.Map carmap=new
   java.util.HashMap();
            carmap.put("Large","Mercedes");
            carmap.put("Medium","Getz");
            carmap.put("Small","Alto");
String[] cartype={"Large","Medium","Small"};

request.setAttribute("carmap",carmap);
request.setAttribute("cartype",cartype);
request.setAttribute("size","Medium");

request.getRequestDispatcher("i2.jsp").forwa
   rd(request,response);
}
```

In JSP

```
 <body>
    My car is: ${carmap.Large}
    <br> or  <br>
    My car is: ${carmap["Large"]}
 <br>

Car type : ${cartype[0]} or Car type :
  ${cartype["1"]}
<br>
   My car is: ${carmap[size]}
 <br>

 My car is:
 ${carmap[cartype[0]]}
   </body>
```

Both print **Mercedes**

prints **Large**

prints **Medium**

prints **Getz.** Without double quotes inside the square brackets the string value is treated as attribute. The attribute evaluates and its value is substituted.

prints **Mercedes**

Question?

- Write an EL to print the session id.

~~`${session.id}`~~ No 'session' in EL implicit object

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

EL implicit object

Question?

- Suppose that there is a ‘size’ attribute set in the session scope as well. Write an EL Expression to print the size attribute in the session scope?

~~`${pageContext.session.size}`~~

~~`${size}`~~ This returns the request scope’s ‘size’

`${sessionScope.size}`

Getting other parameters

- **`${cookie.uname.value}`**
  - Returns the value of uname cookie
- **`${initParam.color}`**
  - **`<%=application.getInitParameter(“color”)%>`**

- Can you write EL to get the parameters from the form.

Form Params

- **`${param.eno}`**
  - Gets the request parameter matching 'eno'
- **`${paramValues.hobbies}`**
  - Gets the first value from the multiple value request attribute
- **`${paramValues.hobbies[0]}`**
  - Same as the above
- You could also say **`${param.eno[0]}`** for a single value parameter.

EL Operators

- Arithmetic Operators
  - **+**
  - **-**
  - *****
  - **/,div**

  ${42/0} →gives INFINITY as output and not Exception
  - **% ,mod**

  ${42%0}→ gives an Exception

  In arithmetic expressions, EL treats the unknown variable (null value) as Zero(0).

- Relational Operators
  - ==, eq
  - !=, ne
  - < , lt
  - > , gt
  - <=, le
  - >=, ge

- Logical Operators

– && , and	AND
– || , or	OR
– !, not	NOT

In Logical expressions (using logical operators and/or relational operator, EL treats the unknown variable (null value) as false.

EL reserved words

and	**eq**	**gt**	**true**	**instanceof**	
or	**ne**	**le**	**false**	**empty**	
not	**lt**	**ge**	**null**	**div**	**mod**

Note:

`${empty obj}` returns true if obj is null or empty.