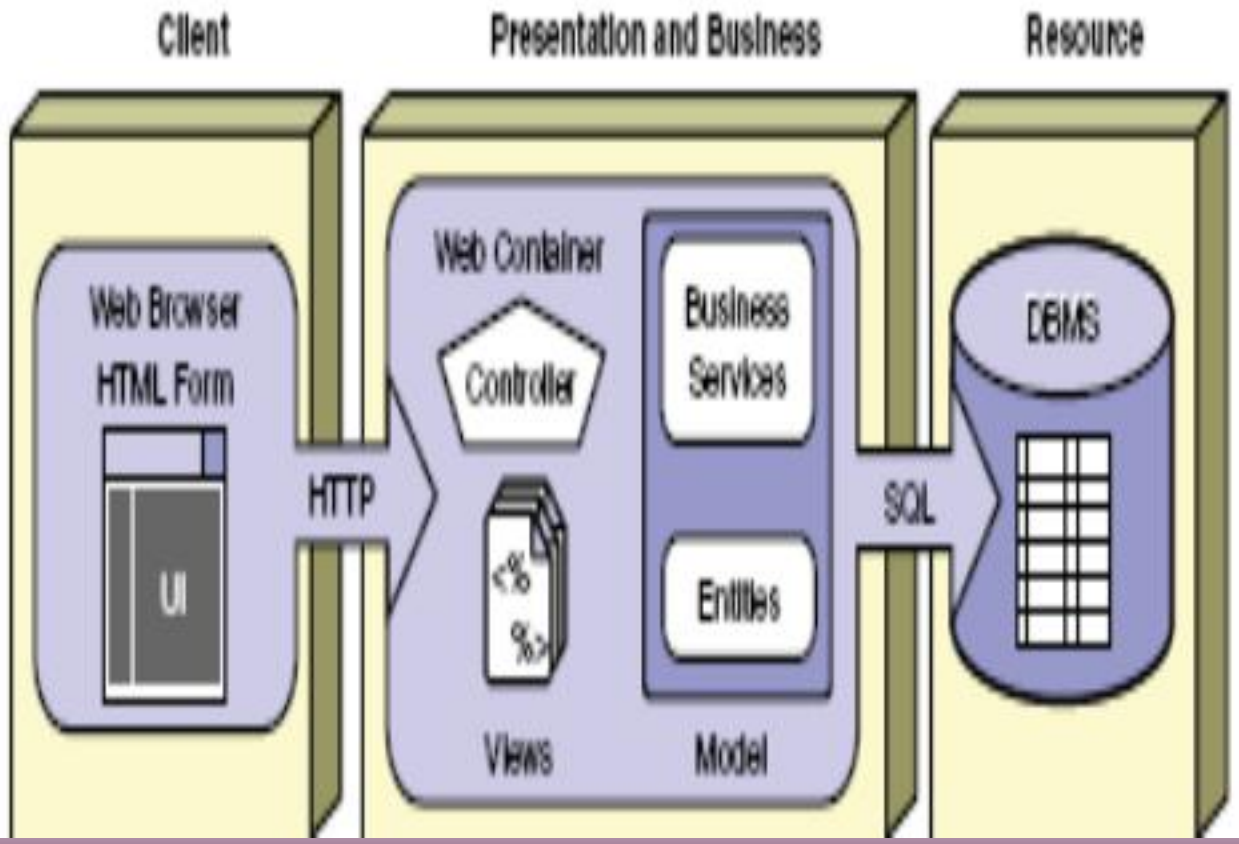
Web Component Development - Servlet

Role of Web Component JEE

- Web-centric Java EE application architecture:



Servlets

- Java technology for serving HTTP requests.
- Used for serving HTTP requests
- Basic servlet functionalities implemented by the basic HTTP servlet classes have the features for accessing HTTP request data and managing cookies and sessions.
- User interface or representation part is done with other technolgies.

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

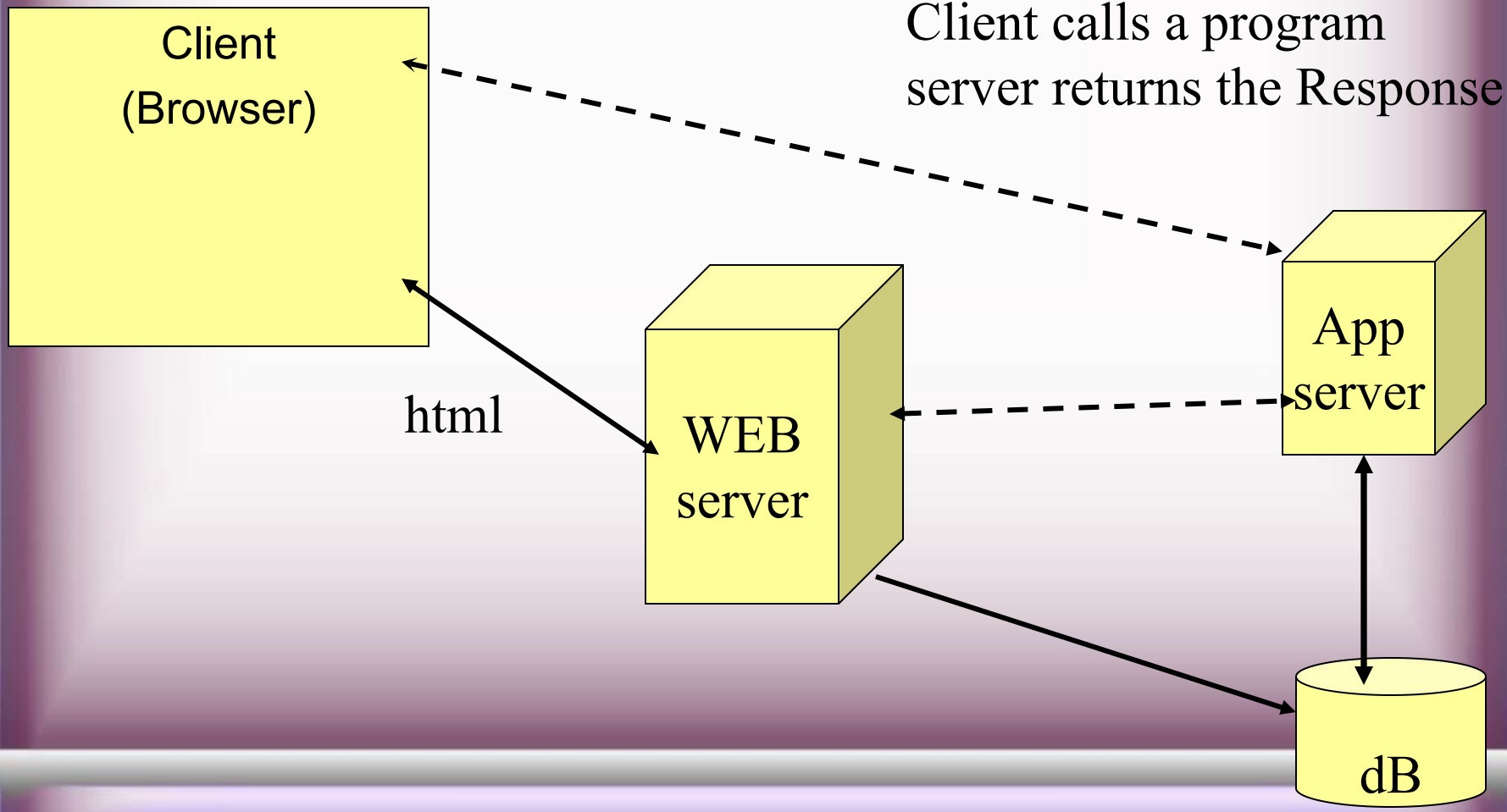
-
- Access to content in the Web app is always through a URL that has a common prefix

  - – http://host/webAppPrefix/abc

  - Many aspects of Web application behavior controlled through deployment descriptor- web.xml till servlet version 2.3
  - Web.xml is replaced with Annotation based Configuration from version 3.0

APPLICATION SERVER

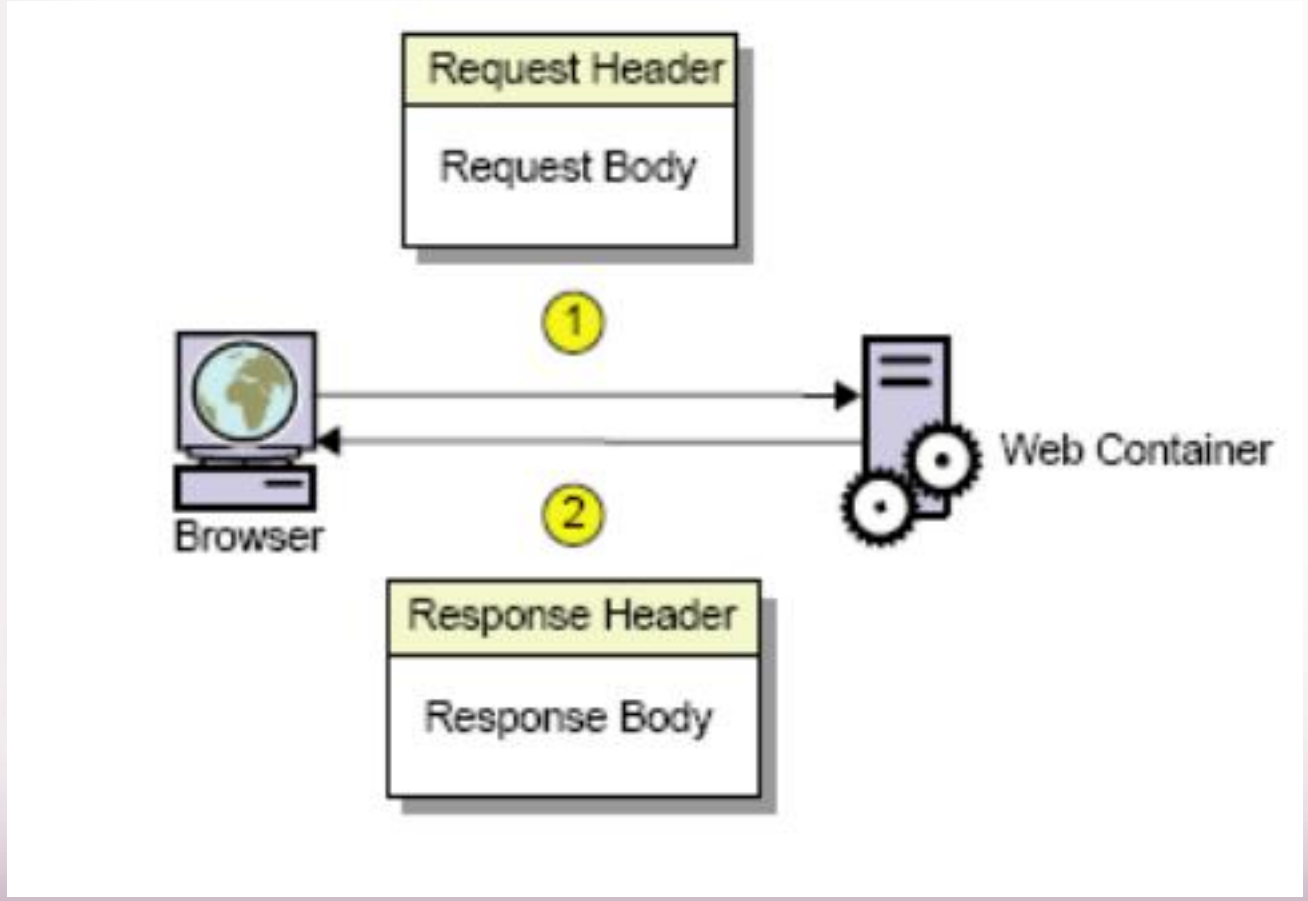
- A server computer on a computer network dedicated to running certain software applications
- A software engine that delivers applications to client computers.
- Should handle most, if not all, of the business logic and data access of the application.
- Ease of application development and centralization.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


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

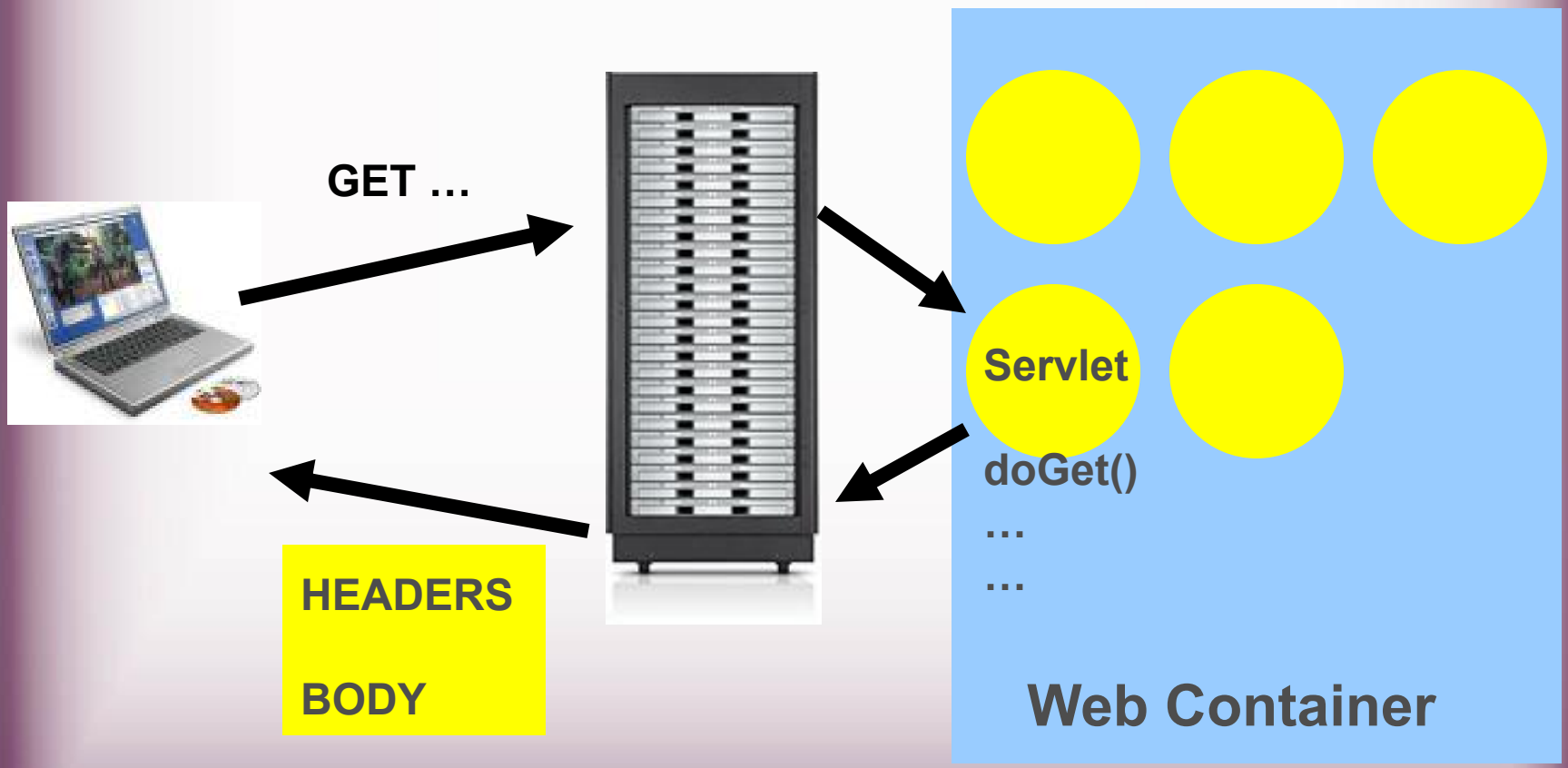
Java Servlets

- 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

Request and Response



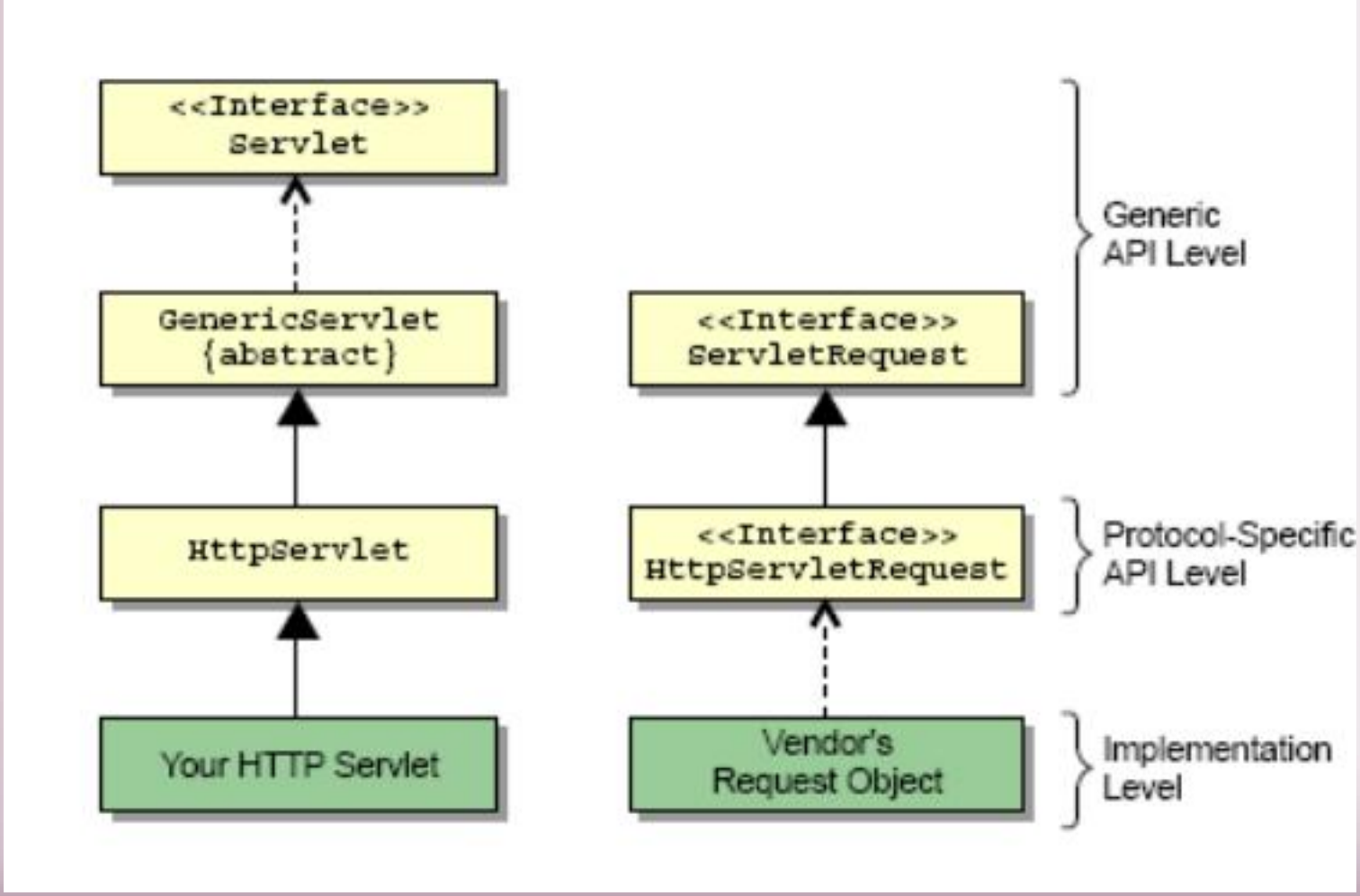
Web Apps with Servlets



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

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

Steps for Creating 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

Example

```
public class GreetingServlet extends HttpServlet {
public GreetingServlet() {
        super();
   }
@Override
public void init(ServletConfig config) throws ServletException {
}
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {   }

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

}
```

@WebServlet

- Used to define a **Servlet** component in a web application.
- Specified on the Servlet class
- Contains metadata about the **Servlet** being declared.

- **String[] value**
  - Array of URL patterns
  - **Either this or urlPatterns should be present**
- **String[] urlPatterns** -
  - Array of URL patterns to which this Filter applies
  - Required Annotation
- **int loadOnStartup**
  - Integer value denoted the startup ordering hint

@WebServlet

```
@WebServlet(urlPatterns = {"/greeting","/greet"})
public class GreetingServlet extends HttpServlet {



}
```

Inter-Servlet Communication

- A process where two or more servlet communicates with each other to process the client request.
- A servlet can **forward** the request to another servlet to process the client request.
- A servlet can **include** the output of another servlet to process the client request.
- Implemented using Request Dispatcher
- Is an object of the **javax.servlet.RequestDispatcher** interface

GET Method

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



RequestDispatcher dispatcher =
    request.getRequestDispatcher("/welcome.jsp");

dispatcher.forward(request, response);

}
```

Welcome.jsp

- <%@ page language="java" contentType="text/html; charset=ISO-8859-1 pageEncoding="ISO-8859-1"%>
- <!DOCTYPE html >

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Nalanda Library</title>
</head>
<body>

<h1>Nalanda Library</h1>

</html>
```

Request Attributes –Request Dispatching

- Stores data as attribute to the request object in the first servlet.
- Forwards the request to the second servlet using the RequestDispatcher.
- Retrieves the data from the request object and displays the result using the second servlet.

```
request.setAttribute("foundAuthor", author);

RequestDispatcher dispatcher=
request.getRequestDispatcher("/ShowAuthor.jsp");

dispatcher.forward(request, response);
```

- The path is a String and can use relative addressing too.

Using Request Parameter

```
<form action="LibraryServlet" method="post">

<label>Author Id</label>

<input type="text" name="authorId" placeholder="Author Id to
    Search">

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

</form>
```

Post Method

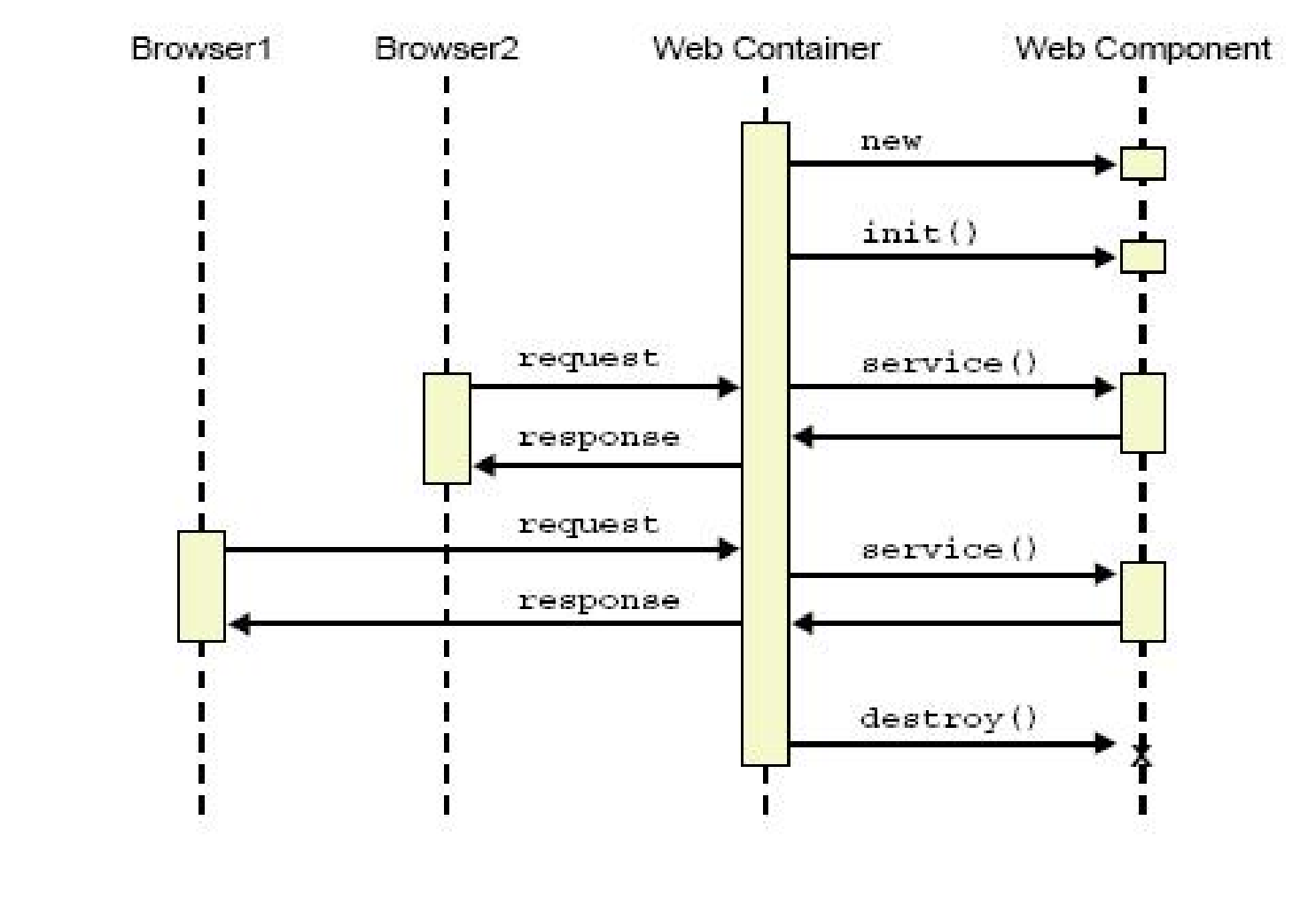
```
protected void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {

String strAuthorId = request.getParameter("authorId");
     long authorId = Long.parseLong(strAuthorId);

          AuthorDao dao =new AuthorDao();

           author =dao.findAuthor(authorId);
         request.setAttribute("foundAuthor",author);
RequestDispatcher dispatcher =
   request.getRequestDispatcher("/welcome.jsp");
         dispatcher.forward(request, response);
}
```

Welcome.jsp

- <%@ page language="java" contentType="text/html; charset=ISO-8859-1 pageEncoding="ISO-8859-1"%>
- <!DOCTYPE html >

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Nalanda Library</title>
</head>
<body>
${foundAuthor}
</html>
```

Life Cycle of a Servlet

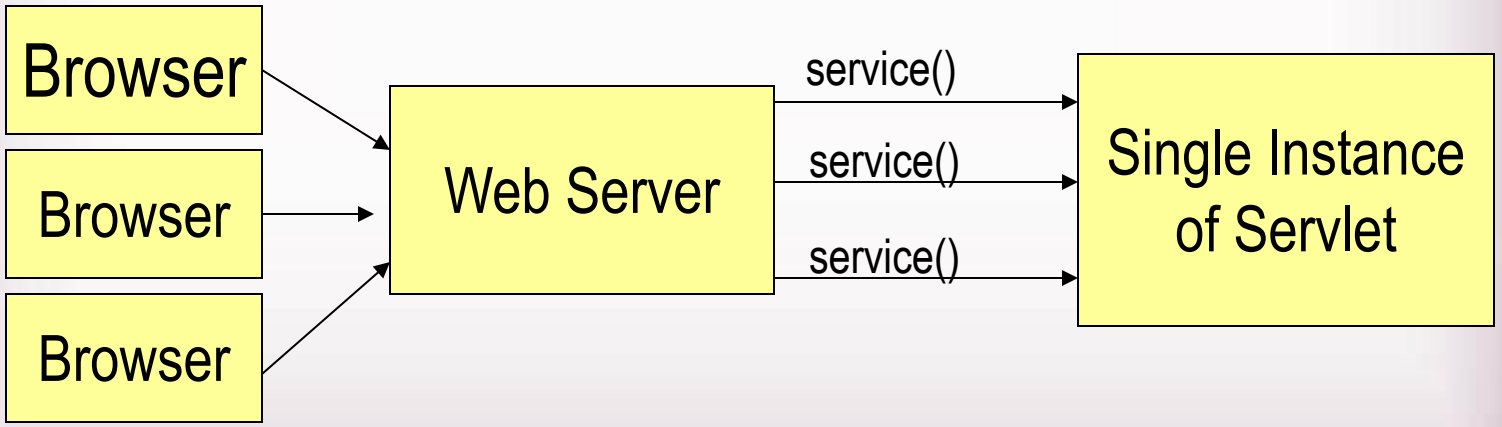


The init() method

- The init() method is called when the servlet is first requested by a browser request.
- It is not called again for each request.
- Used for one-time initialization.
- 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.

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.

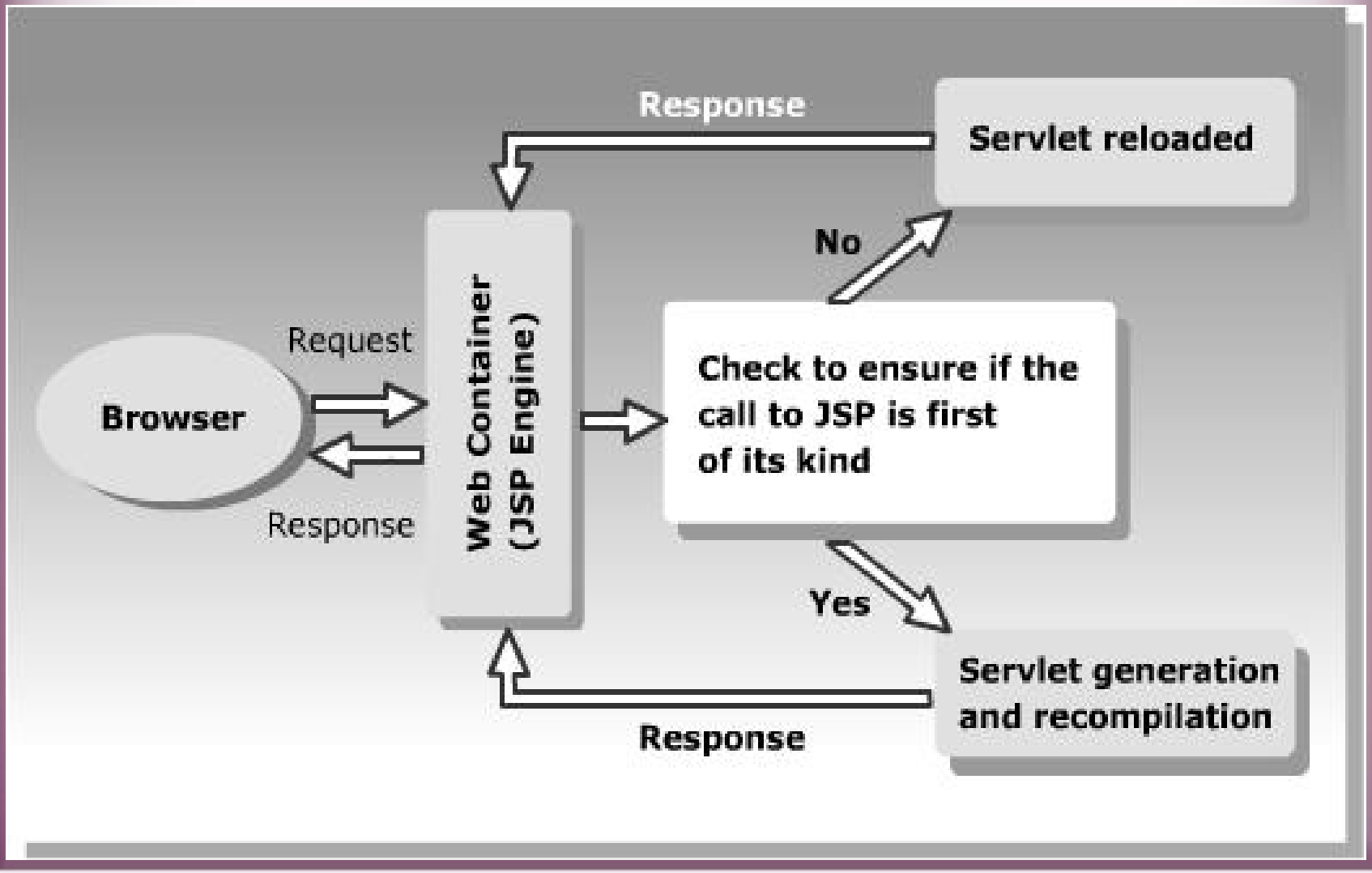
  –A Web designer can design and formulate the layout for a Web page by using HTML.

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

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



Jsp Scripting

- Scriptlets
  - Similar to code placed inside the service method

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

Expressions & Declarations

- `<%= code %>`

- Expressions begins with the JSP start tag followed by an equals sign
- *Does not end in a semicolon*, unlike other Java Statements
- Declarations
  - `<%! code %>`
- Used initialize variables and methods
- *Scriptlets, Expressions. Variables and methods created within Declaration elements are global*

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%>
</body>
</html>
```

Declaration

Expression

JSTL

- Sun and JCP, initiated the **JSP-Standard Tag Library** (JSTL) project.
- The tag libraries are very elegant and simple to use,
- However non standard tags and there proliferation creates lot of confusion among the developers
- JSTL was introduced in 2003, and now incorporated into JSP-2.
- It helps in taking away Java code! From jsp page .
- It also provide a standard implementation for typical application functionality
  - Reusability
  - Avoid reinventing the wheel

What Is the JSTL?

- A collection of tag libraries implementing common JSP functionality
  - Core functions
    - Variables, I/O, conditionalization, iteration
  - Formatting/I18n
    - Message bundles, numbers, dates
  - XML operations
    - parsing, transformations
  - Database operations
    - SQL

Declaration of JSTL Tag Libraries

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

- Delimiters are “**${**” and “**}**”
- The EL can only be used for specifying attribute values in JSTL tags

```
<c:out value=”${firstName}”/>
```

- Multiple expressions can be combined and mixed with static text (*i.e.*, implicit string concatenation)

```
<c:out value=”Hello ${firstName} ${lastName}!”/>
```

<c:out> & <c:set>

- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
- <c:set var="message" value="JSTL Programming " scope="page"/>
- <c:out value="${message}"/>

Scope 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.

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**

pageContext

- This represent the entire JSP page.
- Access information about the page
- Stores references to the request and response objects for each request.
  - The application, config, session,
  - out objects are derived by accessing attributes of this object.
- Contains information about the directives issued to the JSP page, including the buffering information, the errorPageURL, and page scope.
- Performing work involved with the forward and include actions.
- The pageContext object also handles uncaught exceptions.

PageContext to Set and get attributes

- Setting the page-scoped attribute
  - <% float one = new float(45.2) ; %>
  - <%pageContext.setAttribute(“atname”,one); %>
  - <%=pageContext.getAttribute(“atname”)%>
- Using pageContext to set a Session-scoped attribute

  - <% float two = new float(45.2) ; %>
  - <%pageContext.setAttribute(“atname”,two,PageContext.SESSION_SCOPE); %>
  - <%=pageContext.getAttribute(“atname”, PageContext.SESSION_SCOPE)%>

- APPLICATION_SCOPE, PAGE_SCOPE,
- REQUEST_SCOPE,SESSION_SCOPE

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>
```

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.

<c:if>

- If the value of attribute is true the body is executed.
  - With out body
    - **<c:if test="testCondition" var="varName" scope="{page|request|session| application}"/>**
  - With body
    - **<c:if test="testCondition" [var="varName"] [scope="{page|request|session|application}]">**

      body content

      </c:if>

Example

```
<c:if test="${empty param.tname}“ var=“test” >

    <jsp:forward page="error.jsp" />
</c:if>

  Hello ${param.tname}

Test Result : ${test}
```

Decision and Iteration

```
<c:if test="${studName eq 'Ramesh' }">
    <c:out value="Correct Answer"/>
 </c:if>
```

```
<c:if test="${name.contains('a') }">
          <c:out value="${name}"></c:out>
        </c:if>
```

Decision and Iteration

```
<c:if test="${!(name eq 'Umang') }">
    <c:out value="${name}"/>
</c:if>
```

```
<c:forEach var="n"  begin="10"  end="20" step="1">
    <c:out value="${n}"/>
</c:forEach>
```

Iterate over Objects in Scope

```
<c:forEach var="name" items="${sessionScope.nameList}">
<c:out value="${name}"/>

</c:forEach>
```

EL Identifiers

- Identifiers are resolved against the four JSP scopes
  - Using `PageContext.findAttribute(name)`
  - Scopes are searched sequentially: `page`, `request`, `session`, `application`
- Reserved identifiers for the 11 JSTL implicit objects:
  - `pageContext`,
  - `pageScope`,
  - `requestScope`,
  - `sessionScope`,
  - `applicationScope`
  - `Param`
  - `paramValues`
  - `header`
  - `headerValues`
  - `initParam`
  - `cookie`

The Form

```
<form action="Welcome.jsp">

 Best Java Book:

  <input type="text" name="bookName">
  Best Java Student
  <input type="text" name="studentName">

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

</form>
```

JSTL Implicit Object

```
<c:out value="${param.bookName}"/>
 <br/>
 <c:out value="${param.studentName}"/>

 <c:set var="studName" value="${param.studentName}"
                                        scope="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 import ="java.io.* , java.util.*" %>

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 jsp page , but it’s a kind of runtime call
  - This inserts the RESPONSE of the “Header.jsp” at run time.
- Both the Include directlves are position sensitive.

<jsp:forward>

- Can forward the request to another jsp page or servlet by using this tag

- It can be based on condition also

  – <jsp:forward page =“getvalues.jsp” />
  – < %
    if (request.getParmeter(“username”)==null) { %>
    <jsp:forward page=“reenter.jsp”/>
  <%} %>

  Hello <%=request.getParameter(“userName”) %>

  <jsp:forward page=“user.jsp">
  <jsp:param name="p1" value="Invalid userid/password" />
  </jsp:forward>

LISTENERS AND FILTERS

Servlet Listeners

- Monitoring and reacting to events in a servlet's life cycle can be done by defining listener objects
- The Methods get invoked when life-cycle events occur.
- **Defining the Listener Class**
- Defined as an implementation of a listener interface.
- Created by implementing the corresponding interface for a Particular event

Listner Interfaces and Event Class

- **Web context - Initialization and destruction**
  - ServletContextListener and ServletContextEvent
- **Attribute added, removed, or replaced**
  - ServletContextAttributeListener and ServletContextAttributeEvent
- **Session Creation, invalidation, activation, passivation, and timeout**
  - HttpSessionListener, HttpSessionActivationListener, and HttpSessionEvent
- **Attribute added, removed, or replaced**
  - HttpSessionAttributeListener and HttpSessionBindingEvent

Listeter Interface and Event classes

- **Request - request start and process**
  - ServletRequestListener and ServletRequestEvent
- **Attribute added, removed, or replaced**
  - ServletRequestAttributeListener and
  - ServletRequestAttributeEvent

Listener

```
@WebListener
public class ContextListener implements
    ServletContextListener {

    Logger log = Logger.getRootLogger();

    public void contextInitialized(ServletContextEvent sce) {

      log.info("Context Initialzed");
        ServletContext ctx = sce.getServletContext();

            log.info("Context Info"+ ctx.toString());
        }
    }
```

Filters

- Transform the content of HTTP requests, response, and header information
- Do not generally create response or responding to requests
- Attached to one or more servlets or JSP
- Modify or adapt the request/response for a resource
- Act on dynamic or static content – web resources

Examples of Filters

- Examples of Filters
  - Authentication
  - Blocking requests based on user identity
- Logging and auditing
  - Tracking user of a web application
- Image conversion
  - Scaling maps, and so on
- Localization
  - Targeting the request and response to a particular locale

Uses

- Access a resource before a request is invoked
- Process a request before it is invoked
- Modify request headers and data
- Modify response headers and data before sending them to client
- Intercept an invocation of a resource
- Act on a servlet, or servlets or static content by one or more filters in a specific order

Specifying Filter Mappings

- A web container uses filter mappings to decide how to apply filters to web resources.
- A filter Mapping matches a filter to a web component by name, or to web resources by URL pattern.
  - Filter is mapped by annotating filter class with @WebFilter

```
@WebFilter("/*")
public class MyFilter implements Filter {
    ...
}
```

Basic Steps to Create Filters

- Create a class that implements the Filter interface
  - Implement three methods doFilter(), init(), and destroy()
- Put filtering behavior in doFilter() method
- Call doFilter() method of FilterChain object
  - When doFilter() is invoked from FilterChain object, the next associated filter is invoked
  - If no other filter is associated, servlets/JSP themselves invoked

Filter

```
@WebFilter("/*")
public class LogFilter implements Filter {

Logger log = Logger.getRootLogger();

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

log.info("Log filter  PRE CALLED ");
chain.doFilter(request, response);
log.info("Log filter POST CALLED ");
}

}
```