ICS 113 (Web Programming)
Reviewer for Final Examination

Web Application – an application program that is stored on a remote server and delivered over the Internet through a browser interface

Advantages

- App runs server side (CPU, disk space, configuration)
- No install, packaging, CDs, upgrades, configurations or tweaking of settings on the client side.
- Greater responsibilities and control placed in the hands of the system administrators (as opposed to the users)
- Data is likely more secure (stored server side, w/ proper security measures and backup)
- Machine independent (any user can log in from any computer.
- One application will run on any and all platforms, assuming standards compliant code and browsers.
- Reduced external network traffic (ex. Database heavy applications)
- Lower client side system requirements (machine only needs network access and the ability to run a compliant web browser)

Disadvantages

- "Who has my data?" Essentially, not you.
- Issues of trust; many users do not trust other people, even within the same company, to keep their data safe and secure.
- Response time. While the actual execution of the app may be much quicker, user response time can be noticeably slower than a local app.
- Internet (or network) connectivity is not (yet) ubiquitous.
- Browser compatibility can still be a problem.
- Some tasks that are simple in traditional application development, are quite complicated from a web application (ex, local printing)
- Security concerns limit what you can accomplish (limited access to the users local machine).
- Depending on the application, usability can be very bandwidth sensitive.

Some advantages are also disadvantages:

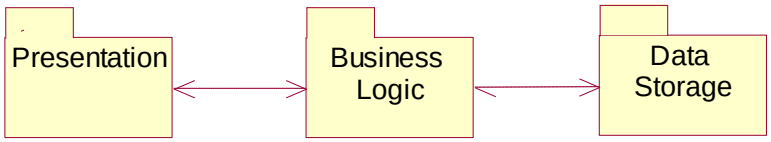
- Running applications server-side requires servers with sufficient power (CPU, memory, disk, bandwidth) to handle multiple simultaneous users.
- Placing greater control in the hands of the system administrators is only an advantage with a sufficient number of competent admins.

Web based applications are ideal for:

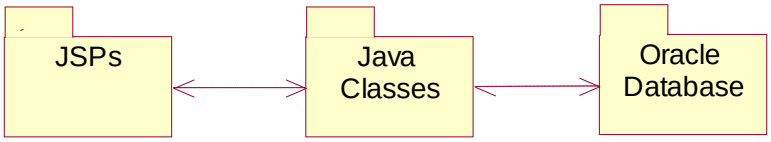
- Database heavy applications
- Applications that must be used remotely
- Varied user base
- Low GUI requirements
- Not performance sensitive (client side)

Information Systems Society - Academics Committee
Institute of Information and Computing Sciences
University of Santo Tomas
By: C. Castodio

Architecture of Web Applications



Example of a possible J2EE implementation



Servlet – Java classes that dynamically process request and response

JSP Pages – text-based documents that execute as Servlets but allow more natural approach to creating static content

Web Container – services that provide for web components (request dispatching, security, concurrency, and life-cycle management) and gives access to APIs such as naming, transaction and email.

N-Tier Architecture

- Client Tier - consists of application logic accessed directly by an end user through a user interface.
- Presentation Tier - consists of application logic that prepares data for delivery to the client tier and processes requests from the client tier for delivery to back-end business logic.
- Business Tier - consists of logic that performs the main functions of the application: processing data, implementing business rules, coordinating multiple users, and managing external resources such as databases or legacy systems.
- Integration Tier - consists of data used by business logic; data can be persistent application data stored in a database management system.

Web Application Life Cycle Composition

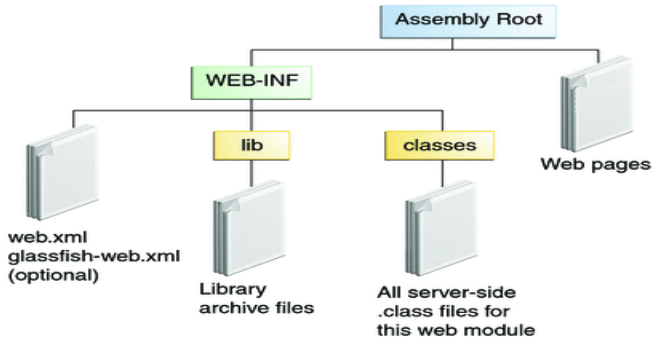
- Web Components
- Static Resource Files
- Helper Classes
- Libraries

Web Application Life Cycle

- Develop the web component code
- Develop the web application deployment descriptor
- Compile the web application components and helper classes referenced by the components
- Optionally packaged the application into a deployable unit
- Access a URL that references the web application

Web Module - the smallest deployable and usable unit of web resources; contains images, static web content files

Document Root – top-level directory of web module



WEB-INF

- Web.xml – web application deployment descriptor
- Tag Library Descriptor Files
- Classes – directory that contains server-side classes: servlets, utility classes and JavaBeans components
- Tags – directory that contain tag files, which are implementation of tag libraries
- Lib – a library contains JAR archives of library called by server-side classes

Dynamic Content – content-based generated-based on program parameters, HTTP request and responses and database queries

Static Content – output produced for the consumer is still the same. Typical representation is HTML

Why Build Pages Dynamically?

- The web page is based on data submitted by the user
- The web page is derived from data that changes frequently
- The web page uses information from databases or other server-side sources

What is a Servlet?

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

Common Gateway Interface (CGI) technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

Disadvantages of CGI

- If number of clients increases, it takes more time for sending response.
- For each request, it starts a process and Web server is limited to start processes.
- It uses platform dependent language e.g. C, C++, perl.

Advantages of using a Servlet

- **better performance:** because it creates a thread for each request not process.
- **Portability:** because it uses java language.
- **Robust:** Servlets are managed by JVM so no need to worry about momory leak, garbage collection etc.
- **Secure:** because it uses java language.

HTTP (Hyper Text Transfer Protocol)

- Http is the protocol that allows web servers and browsers to exchange data over the web.
- It is a request response protocol.
- Http uses reliable TCP connections by default on TCP port 80.
- It is stateless means each request is considered as the new request. In other words, server doesn't recognize the user by default.

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Difference Between Get and Post

GET	POST
In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
Get request **can be bookmarked**	Post request **cannot be** bookmarked
Get request is **idempotent**. It means second request will be ignored until response of first request is delivered.	Post request is **non-idempotent**
Get request is **more efficient** and used more than Post	Post request is **less efficient** and used less than get.

Container

- It provides runtime environment for JavaEE (j2ee) applications.
- It performs many operations that are given below:

    1. Life Cycle Management
    2. Multithreaded support
    3. Object Pooling
    4. Security etc.

Server - It is a running program or software that provides services.

Web Server - It contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB.

Application Server – It contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc.

Content Type – It is also known as MIME (Multipurpose internet Mail Extension) Type. It is a **HTTP header** that provides the description about what are you sending to the browser.

Servlet API

- The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.
- The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.
- The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Servlet Interface

- It provides common behavior to all the servlets.
- It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet Interface

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

GenericServlet Class

- It implements **Servlet**, **ServletConfig** and **Serializable** interfaces.
- It provides the implementaion of all the methods of these interfaces except the service method.
- It can handle any type of request so it is protocol-independent.
- It can be created by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet Class

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

HttpServlet Class

- It extends the GenericServlet class and implements Serializable interface.
- It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of Http Servlet Class

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Servet Life Cycle

1. **Servlet class is loaded.**
   The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.
2. **Servlet instance is created.**
   The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.
3. **init method is invoked.**
   The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface.
   Syntax:
   ```
   public void init(ServletConfig config) throws ServletException
   ```
4. **service method is invoked.**
   The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once.
   Syntax:
   ```
   public void service(ServletRequest request, ServletResponse response)
           throws ServletException, IOException
   ```
5. **destroy method is invoked.**
   The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.
   Syntax:
   ```
   public void destroy()
   ```

ServletRequest Interface

- An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Methods of ServletRequest Interface

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

RequestDispatcher Interface

- It provides the facility of dispatching the request to another resource it may be html, servlet or jsp.
- This interface can also be used to include the content of another resource also.
- It is one of the way of servlet collaboration.

Methods of RequestDispatcher Interface

Method	Description
public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException	forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:	includes the content of a resource (servlet, JSP page, or HTML file) in the response.

Example

```
RequestDispatcher rd=request.getRequestDispatcher("servlet2");  //servlet2 is the url-pattern of the second servlet

rd.forward(request, response); //method may be include or forward
```

ServletConfig Interface

- An object of ServletConfig is created by the web container for each servlet.
- This object can be used to get configuration information from web.xml file.

Advantage of Using Servlet Config

- The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

Methods of ServletConfig Interface

Method	Description
public String getInitParameter(String name)	Returns the parameter value for the specified parameter name.
public Enumeration getInitParameterNames()	Returns an enumeration of all the initialization parameter names.
public String getServletName()	Returns the name of the servlet
public ServletContext getServletContext()	Returns an object of ServletContext.

Example

```
ServletConfig config=getServletConfig();  //Now we can call the methods of ServletConfig interface
```

ServletContext Interface

- An object of ServletContext is created by the web container at time of deploying the project.
- This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

Advantage of ServletContext

- **Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Usage of ServletContext Interface

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.

Methods of ServletContext

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters.
3. **public void setAttribute(String name,Object object):**sets the given object in the application scope.
4. **public Object getAttribute(String name):**Returns the attribute for the specified name.
5. **public Enumeration getInitParameterNames():**Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. **public void removeAttribute(String name):**Removes the attribute with the given name from the servlet context.

Session Tracking in Servlets

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Why user Session Tracking?

It is used to recognize the particular user.

Session Tracking Techniques

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.

Types of Cookie

- **Non-persistent cookie -** It is valid for single session only. It is removed each time when user closes the browser.
- **Persistent cookie -** It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Useful Methods of Cookie class

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

HttpSession Interface

- The container uses this id to identify the particular user.
- An object of HttpSession can be used to perform two tasks:
    - bind objects
    - view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Methods of HttpSession Interface

Method	Description
public String getId()	Returns a string containing the unique identifier value.
public long getCreationTime()	Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public long getLastAccessedTime()	Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
public void invalidate()	Invalidates this session then unbinds any objects bound to it.

Hidden Form Field

A hidden (invisible) textfield is used for maintaining the state of an user. We store the information in the hidden field and get it from another servlet. Better if we have to submit form in all the pages and we don't want to depend on the browser.

1. <input type="hidden" name="uname" value="Vimal Jaiswal">

Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

```
out.print("<input type='hidden' name='uname' value='"+n+"'>");
```

URL Rewriting

We append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

name and a value is separated using an equal = sign

a parameter name/value pair is separated from another parameter using the ampersand(&)

Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

Java Server Pages

JSP technology is used to create web application just like Servlet technology. It as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc. Processed by JSP Container and Processed by the software on the server.

- A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than servlet because we can separate designing and development.
- Sun Microsystems' solution for developing dynamic content website
- Provides excellent server side scripting support for creating web app
- Enable the developers to directly insert the java code to JSP File
- Efficient, it loads into the web servers memory on receiving request the very first time and the subsequent calls are served within a very short period of time

JSPs and Servlet

- JSP extends and depends on the Java Servlet API and implementation
- JSPs are converted to Java Servlets
- JSP Implementation class is an underlying servlet representation
- All the benefits provided to the Java Servlet component is the same as to the JSP by its web container
- JSP page can also refer to request, response and session management objects created by a web container

Advantage of JSP over Servlet

There are many advantages of JSP over servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

JSP Constructs

Scripting Elements include variable and declarations, expressions to be evaluated, and scriplets.

Directives are interpreted at transition time, inclusion of other files, and defining attributes about a JSP file being translated

Actions are commands that have purely tag-like look to them but are interpreted by the host language transparently to the programmer, inclusion of other files, defining attributes about a JSP file being translated

JSP Comment

<%-- A comment --%>

JSP Scripting Comment

```
<%      /**

            Java-Based JSP Comment Line

        */

%>
```

Java Comment

<% // A Comment %>

JSP Scripting Elements

Declaration is used to declare variables and methods inside a JSP File. Defined as <% ! …. %> character sequences.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

Example:

<%! private double dTaxPay = 0; %>

<%! public double computeTax(){} %>

Expression is an element that defines statements that are evaluated during the request processing time. The output of the expression is the type String object. Defined as <%= %> character sequences.

It is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Example:

<%= new Date(); %>

< %= statement %>

The **Scriplet's** general purpose is for executing statements, declare variables and methods, evaluate expressions, and utilize JSP Objects. It is executed during the processing of JSP Client requests with any output generated to the response output. Defined within <%... %> character sequences.

A scriptlet tag is used to execute java source code in JSP.

Jsp Scriptlet Tag	**Jsp Declaration Tag**
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

JSP Directives

JSP Directives are messages that tell the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive
- taglib directive

General Form: <%@ DirectiveTypeName attribute=”value” …%>

Include Directive

- It used for file insertion to a JSP File.
- Types of File: text, HTML or JSP File
- Has one attribute, “file” – value is the relative URL of the file to be included
- URLs beginning with “/” – relative to a JSP web application context
- Advantage of using this is code reusability

  <%@ include file =”displayWeb/hello.html” %>

Page Directive

- Used to define information about a JSP file and any of its statically included files
- JSP containers use page directive to determine the nature and characteristics of a JSP File
- Has various standard attributes that can be defined only once

Language

Specifies the type of scripting language that is used inside the JSP

Import

Specifies a comma-separated list of classes and packages used by JSP

<%@ page import = “java.io.*” , “java.util.*” %>

Session

Specifies a literal value of “true” or “false” indicating whether this JSP uses HTPP Session. The default value is true.

errorPage

Specifies a URL for a JSP that handles any Throwable exception object tat were uncaught by JSPs and made their way to the JSP container

< %@ page errorPage="myerrorpage.jsp" %>

isErrorPage

Specifies a value of true of false indicating whether this JSP is an error page to handle uncaught JSP exceptions. The default value is false.

< %@ page isErrorPage="true" %>

contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

< %@ page contentType=application/msword %>

info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

< %@ page info="This is the main page" %>

TagLib

JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

Model View Controller - A Design Pattern that **separates** the business logic, presentation logic and data

Model

- the classes that **represent the data being displayed**
- represents the **state of application**
- **business logic**

View

- the JSP Pages that **represent the output the client sees**
- **presentation** i.e. User Interface

Controller

- the part that **handles the requests, decides what logic to invoke** and **what JSP page should apply**
- acts as **interface between view and model**
- **intercepts** all incoming requests

Advantage of MVC Model 2 Architecture

1. Navigation Control is centralize

2. Easy to maintain the large application

Why Combine Servlets and JSP?

Typical picture: use JSP to make it easier to develop and maintain HTML Content

- For simple dynamic code, call servlet code from scripting elements
- For slightly complex applications, use customer classes called from scripting elements
- For moderately complex applications, use beans
- For complex process, starting with JSP is awkward
- Despite the ease of separating the real code into separate classes, beans, and tags, the assumption behind JSP is that a single page gives a single basic look

Possibilities for Handling Single Request

- Servlet only. Works well when
  - output is **binary type** (images)
  - There is **no output** (search engines forwarding and redirection)
  - **Variable** Format/Layout (portal)
- JSP only. Works well when:
  - Output is **character data**
  - Mostly **fixed** format/layout

- Combination (MVC Architecture). Needed when:
    - Single request will result substantially different-looking results.
    - You have large development team with different team members doing the Web Development and the business logic
    - You perform complicated data processing, but have a relatively fixed layout

MVC Misconceptions

- An elaborate framework is necessary such as Struts and JavaServer Faces (JSF)
    - They are not required
    - Implementing MVC with the built-in RequestDispatcher works very well for most simple and moderately complex applications
- MVC totally change your overall system design
    - You can use MVC for individual requests
    - Think of it as the MVC approach, not the MVC architecture

Beans

- Java Classes that follow certain conventions:
    - Must have a zero-argument (empty) constructore
        - Defining or by omitting constructors
    - Should have no public instance variables
        - Use accessor methods instead direct access to fields
    - Persistent values should be access through methods called getXxx and setXxx
        - Boolean properties can use isXxx instead of getXxx
    - Beans and utility classes must always be in packages

Implementing MVC with RequestDispatcher

1. Define beans to represent the data

2. Use a servlet to handle request

    Reads request parameters and checks for missing and malformed data

3. Populate the Beans

    The servlet invokes business logic (application-specific code) or data-access code to obtain the results. Results are placed in the beans that were defined in Step1

4. Store the bean in the request, session, or servlet context

    The servlet calls setAttribute on the request, session, or servlet context objects to store are reference to the beans that represent the results of the request

5. Forward the request to a JSP page

    The servlet determines which JSP page is appropriate to the situation and user the forward method of RequestDispatcher to transfer control to that page.

6. Extract the data from Beans

The JSP pages accesses beans with jsp:useBean and a scope matching the location of step 4. The page then uses jsp:getProperty to output the bean properties.

The JSP does not create or modify bean; it merely extracts and displays data that the servlet created.

Reference:

http://www.javatpoint.com/