Chapter 10
Servlet

Introduction

- Several years ago, client-server applications have become very popular for building enterprise applications.
- In this model, client application is usually installed on the client's personal computer, which then sends requests to the server application via network.
- In such a model, client applications usually contained most of the presentation and application logic and the interaction with the server application is done through a GUI.
- Such application clients are referred to as fat clients since the application logic is also included in the client application. Any change in the application logic requires reinstallation of the client application on all the computers.
- With the advent of internet, application clients are completely replaced with web clients.

- In this model, both the application logic and presentation logic are physically separated from the clients PC and moved to the server side.
- The Web client delegates all the user interactions to the server side application logic which then processes the requests and uses the presentation tier components to send the response.
- Since the application logic is no longer present on the client PC, web clients are referred to as thin clients.
- Since the presentation logic is also moved to the server side, the layout of the user interface can also be controlled from the server side.
- A web client in this case is nothing but a browser application like Internet Explorer, FireFox etc.
- Applications that use web clients for user interaction are called as Web Applications.

- The way the web applications work is,
  - Web client like Chrome sends a request using HTTP protocol.
  - The server side program takes the HTTP request, processes it and sends a HTTP response back to the web client. The response includes body content in HTML format.
  - Web client then reads the HTTP response, formats the HTML and displays it to the user.
- Since HTTP plays an important role in web applications, let’s first know some basics about it and then look at the server side details.

HTTP

- HTTP stands for Hyper Text Transfer Protocol. Following are some of the important properties of HTTP:
  - It is a stateless protocol, meaning that every HTTP request is independent of each other.
  - It can send data in the request. The server side program reads the data, processes it and sends the response back. This is the most important feature of HTTP, the ability to send data to server side program.
- Based on how the data is sent in the request, HTTP requests are categorized into several types, but the most important and widely used are the GET requests and POST requests.

Difference of Get and Post request

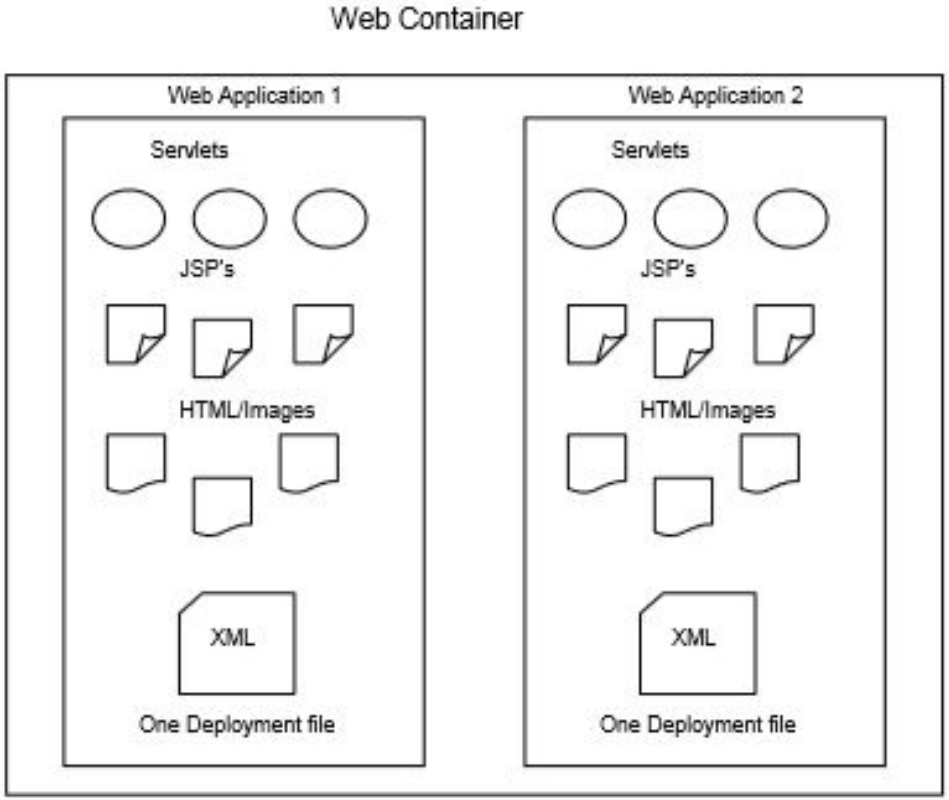
GET	POST
In GET method, values are visible in the URL.	In POST method, values are not visible in the URL.
GET has a limitation on the length of the values, generally 255 characters.	POST has no limitation on the length of the values since they are submitted via the body of HTTP.
GET performs are better compared to POST because of the simple nature of appending the values in the URL.	It has lower performance as compared to GET method because of time spent in including POST values in the HTTP body.
This method supports only string data types.	This method supports different data types, such as string, numeric, binary, etc.
GET results can be bookmarked.	POST results cannot be bookmarked.
GET request is often cacheable.	The POST request is hardly cacheable.
GET Parameters remain in web browser history.	Parameters are not saved in web browser history.

Server Side of the Web Application

- The server- side is the heart of any web application as it comprises of the following:
  - Static resources like HTML files, XML files, Images etc
  - Server programs for processing the HTTP requests
  - A runtime that executes the server side programs
  - A deployment tool for deploying the programs in the server
- In order to meet the above requirements, J2EE offers the following:
  - Servlet and JSP technology to build server side programs
  - Web Container for hosting the server side programs.
  - Deployment descriptor which is an XML file used to configure the web application.

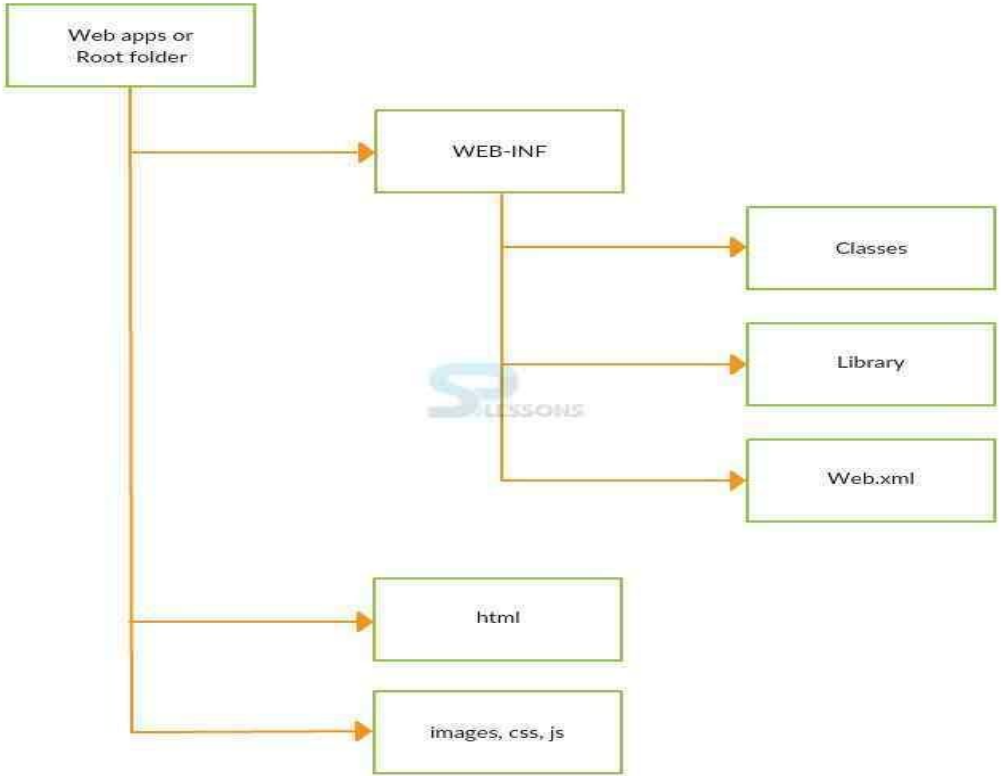
Web Container

- A Web container is the heart of Java based web application which is a sophisticated program that hosts the server side programs like Servlets.
- Once the Servlet programs are deployed in this container, the container is ready to receive HTTP requests and delegate them to the programs that run inside the container.
- These programs then process the requests and generate the responses.
- Also a single web container can host several web applications.
- Look at the following figure that shows how a typical web container looks like:


Web Container
Web Application 1
Servlets
JSP's
HTML/Images
XML
One Deployment file
Web Application 2
Servlets
JSP's
HTML/Images
XML
One Deployment file

- There are several J2EE Web Containers available in the market. One of the most notable one is the Apache’s Tomcat container which is available for free of cost.
- This is what we will install for running web applications.
- Having just the web container is not sufficient. We need to have a programming model to build server side programs that run within this container.
- J2EE supports two key technologies to build server side components in Java. These are Servlet and JSP technologies.

Structure of a Web Application



- All the static resources namely html files, image files can be stored directly under myweb directory.
- Creating this directory structure is one of the steps in the environment configuration section at the beginning of the chapter.
- With the above directory structure, all the components within myweb application should be accessed with the URL starting with:
- http://localhost:8080/myweb/
- If you want to create another web application like mywebproject, you must again create the same directory structure with mywebproject as the root directory and should access the components using the URL starting with:
- http://localhost:8080/mywebproject/
- At this point please make sure you created the directory structure as outlined in environment configuration section.

Servlet Technology

- Servlet technology is a standard J2EE technology used for building dynamic web applications in Java.
- Using Servlet technology is again nothing but using the standard classes and interfaces that it comes with.
- These classes and interfaces form what we call as Servlet API.

Definition of Servlet

- A Servlet is a server side Java program that processes HTTP requests and generates HTTP response.

Servlet API

Though there are several classes and interfaces in this API, we are interested in the most important ones shown in table :

Class/Interface	Description
`HttpServlet`	The base class that represents a HTTP Servlet
`HttpServletRequest`	The class that encapsulates all the HTTP request details
`HttpServletResponse`	The class used for sending HTTP response back to the browser
`ServletConfig`	Class used for dynamically initializing the servlet
`HttpSession`	Class used for session management
`RequestDispatcher`	Class used by the servlet to dispatch the request to various resources.

Deployment Descriptor

- A deployment descriptor is a standard XML file named web.xml that is used by the web container to run the web applications.
- Every web application will have one and only one deployment descriptor (web.xml file). This file defines the following important information pertaining to a Servlet:
  - Servlet name and Servlet class
  - The URL mapping used to access the Servlet
- Let's assume we wrote a servlet named FormProcesssingServlet in a package named myservlets. The definition for this servlet in the web.xml will be as shown below:

```
<servlet>

    <servlet-name>FormProcessingServlet</servlet-name>

     <servlet-class>myservlets.FormProcessingServlet</servlet-class>

</servlet>
```

- The servlet name can be any arbitrary name, but it's a good practice to have the class name as the servlet name. However, the servlet class tag must represent the fully qualified name of the servlet which includes the package name as shown above. This completes Step 1.
- The next thing we need to define is the URL mapping which identifies how the servlet is accessed from the browser. For the above servlet, the url mapping will be as shown below:

  <servlet-mapping>

  <servlet-name>FormProcessingServlet</servlet-name>

  <url-pattern>/FormProcessingServlet</url-pattern>

  </servlet-mapping>

- If you noticed carefully, the servlet name in both the XML snippets is the same. This is how we bind the url mapping with a servlet. The url pattern defines how the servlet will be accessed. With the above mapping, the FormProcessingServlet should be accessed with the following URL:
- http://localhost:8080/myweb/FormProcessingServlet The web container then delegates the request to myservlets.FormProcessing Servlet class to process the request.
-

Steps for Writing a Servlet

- Writing a servlet is very simple. Trust me. You just have to follow a standard process as shown below:
  - Create a class that extends HttpServlet
  - Define 3 methods namely init(), doGet() and doPost().
- These three methods are the standard methods for any servlet. These are called as callback methods that the web container invokes automatically when a HTTP request comes to this servlet.
- For all the GET requests, the web container invokes the doGet() method, and for POST requests it invokes the doPost() method.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class TestServlet extends HTTPServlet {
    public void init(ServletConfig config) throws ServletException {

        //initialization here
    }
    public void doGet(HTTPServletRequest req, HTTPServletResponse res) throws ServletException,IOException {
        // Process get request
    }
    public void doPost(HTTPServletRequest req, HTTPServletResponse res) throws ServletException,IOException {
        // Process post request
    }
 }
```

- If you look at the above servlet, it does the following things
  - Imports all the servlet classes
  - The class extends HttpServlet
  - Defined 3 methods init(),doGet() and doPost()
- These three methods are the standard methods for any servlet.
- These are called as callback methods that the web container invokes automatically when a HTTP request comes to this servlet.
- For all the GET requests, the web container invokes the doGet() method, and for POST requests it invokes the doPost() method.

LifeCycle of a Servlet

- The life cycle of a servlet represents how the web container uses the servlet to process the requests. Following is what a web container does with a servlet:
  - Loads the Servlet
  - Instantiates the Servlet
  - Initializes the servlet by executing the init() method.
  - Invokes the doGet() or doPost() methods to process the requests.
  - Repeats Step 4 until all the requests are processed
  - Destroy the servlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class GreetingServlet extends HttpServlet {
        public void doGet(HttpServletRequest req, HttpServletResponse res)       throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        // Send the message
        pw.println("<h1>Welcome to Servlets </h1>");
        }
}
```

- This is a simple servlet which defined just the doGet() method since we will only send a request from the browser which is a GET request.
- This method takes two arguments namely HttpServletRequest and HttpServletResponse.
- The former is used to read the data from the request and the later is used to send the response back to the browser.
- In this example we only use the response object to send a greeting message to the browser.
- To send a response, we need to do three things as outlined below:

1. Define the type of the response data. This is done using the following statement:
   a. res.setContentType("text/html");

      The above line specifies that we are sending HTML response back.

2. Open a stream/channel to the browser to send the response. This is done using the following statement:
   a. PrintWriter pw = res.getWriter();
3. Send the html message to the browser. To send response message to the browser, we use the println() method as shown below:
   a. pw.println("<h1>Welcome to Servlets </h1>");

Servlet Initialization

- Initializing a servlet is one of the most common practices.
- Servlet initialization is done in the init() method of the servlet.
- Defining initialization parameters for servlets in web.xml is a good practice as it eliminates hard coding in the servlet.
- The web container while loading the servlet, invokes the init() method of the servlet as part of the life cycle and passes the parameters defined in the web.xml to it.
- The other advantage with this is, in the future if the initialization values need to be changed, you only have to change the web.xml without having to recompile the entire web application.
- Initialization parameters to a servlet are defined in the web.xml using the <init-param> element as shown below:

```
<init-param>

    <param-name>driver</param-name>

    <param-value>com.mysql.jdbc.Driver</param-value>

</init-param>
```

- With the above definition, a servlet can access the parameter in the init() method as shown below:

```
public void init(ServletConfig config){
    String drivername = config.getInitParameter(“driver”);
}
```

- As you can see from the above code, the ServletConfig class defines a method named getInitParameter() that takes the name of the initialization parameter defined in the web.xml for that servlet, and returns the value.
- Therefore drivername will be initialized with com.mysql.jdbc.Driver.
- As you can see from the above code, the ServletConfig class defines a method named getInitParameter() that takes the name of the initialization parameter defined in the web.xml for that servlet, and returns the value.

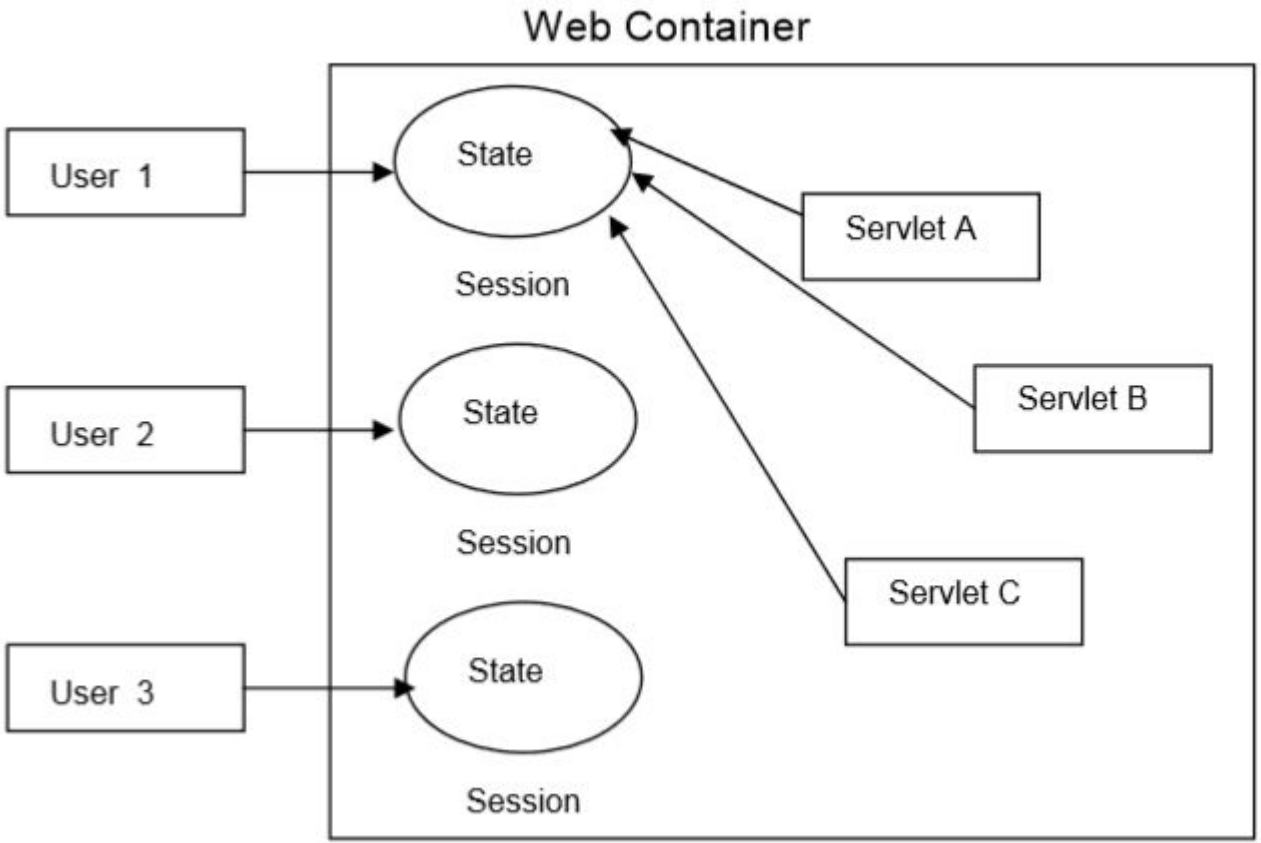
Reading HTML Form Data

- HTML form processing is one of the most common things that any web application does.
- I think you might have noticed on various websites where you fill in html form with all the information and submit it for processing.
- The application then responds with a confirmation message something like “information is successfully processed”.
- Registration pages, Email composing etc are some of the examples.
- Following are the steps for processing a HTML form using a Servlet
    - User fills the data in the html page and submits it.
    - The form data will then be sent to Servlet
    - Servlet reads the form data, processes it and send a confirmation.

Session Management

- The examples we saw until now just deal with one servlet. However, a typical web application comprises of several servlets that require collaborating with each other to give a complete response.
- For instance, if you go online to purchase a book, you need to go through multiple pages like search page, shopping page, billing page etc before you can complete the transaction. In situations like this, it is important that one servlet shares information or data with other servlet.
- In web application terminology we call the shared data or information as state. Having state is just not sufficient. Someone must be able to pass this state from one servlet to other servlet so that they can share the data.
- So, who does this state propagation? Can HTTP do this?

- No, because HTTP is a stateless protocol which means it cannot propagate the state. So, is there anyone to help us out here to make the state available to all the servlets?
- Yes, there is one guy who is always there for our rescue and it’s none other than web container (Tomcat).
- A web container provides a common storage area called as session store the state and provides access to this session to all the servlets within the web application.
- For instance, servlet A can create some state (information) and store it in the session. Servlet B can then get hold of the session and read the state.
- Since the state (data or information) in the session is user specific, the web container maintains a separate session for each and every user as shown in the following diagram.


Web Container
User 1
User 2
User 3
State
State
State
Session
Session
Session
Servlet A
Servlet B
Servlet C

- If you look at the above figure, every user will have his own session object for storing the state pertaining to his transaction and all the servlets will have access to the same session. Also notice, the session objects are present within the container.
- Now that we know what a session is, let’s see how a servlet uses the session for sharing the data across multiple pages in a web application. A servlet can do the following four most important operations to work with sessions.
    - Create the session
    - Store the data in the session
    - Read the data from the session
    - Destroy the session or invalidate the session.
- The above four operations are called as Session Management operations.

Creating a Session

- The servlet API provides us with a class called HttpSession to work with sessions. To create a session, we do the following:
  - HttpSession session = request.getSession(true);
- The above method returns a new session object if one is not present, otherwise it returns the old session object that is already created before.

Storing the data in Session

- Data in session is stored as key-value pair just like in HashMap or Hashtable. The value can be any Java object and the key is usually a String. To store the data we use the setAttribute() method as shown below:
  - session.setAttribute(“price”,new Double(“12.45”));

Reading the data from the Session

- To read the data, we need to use the getAttribute() method by passing in the key as shown below which then returns the value object:
  - Double d = (Double)session.getAttribute(“price”);

Destroying the Session

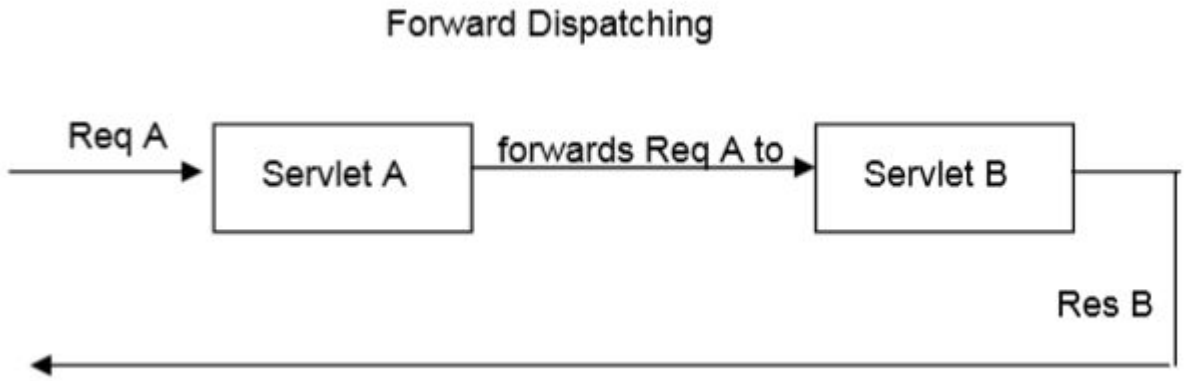
- A session is usually destroyed by the last page or servlet in the web application. A session is destroyed by invoking the invalidate() method as shown below:
  - session.invalidate();

Request Dispatching

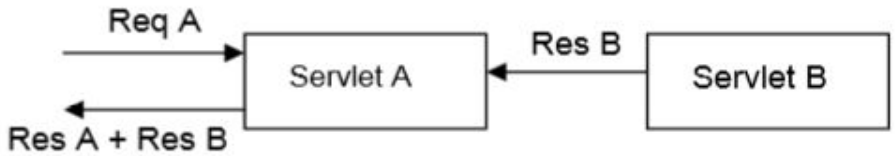
- Request dispatching is the ability of one servlet to dispatch or delegate the request to another servlet for processing.
- In simple words, let's say we have a servlet A which doesn't know how to completely process the request. Therefore, after partially processing the request, it should forward the request to another servlet B. This servlet then does the rest of the processing and sends the final response back to the browser.
- The class used for dispatching requests is the RequestDispatcher interface in Servlet API. This interface has two methods namely forward() and include().

The forward() method

- This method is used for forwarding request from one servlet to another.
- Consider two servlets A and B. Using this method, A gets the request, which then forwards the request to B, B processes the request and sends the response to the browser.
- This method takes HttpServletRequest and HttpServletResponse as parameters.
    - RequestDispatcher rd = req.getRequestDispatcher("shop.html");
    - rd.forward(req, res);


Forward Dispatching
Req A
Servlet A
forwards Req A to
Servlet B
Res B

- With this method, one servlet can include the response of other servlet. The first servlet will then send the combined response back to the browser.
- This method also takes HttpServletRequest and HttpServletResponse as parameters.
    - RequestDispatcher rd=req.getRequestDispatcher("BannerServlet");
    - rd.include(req, res);



Include Dispatching