

# Chapter 5...

---

## Servlets and JSP

---

**Weightage of Marks = 20, Teaching Hours = 08**

### Contents

- 5.1 Introduction
- 5.2 Servlet
  - 5.2.1 Common Gateway Interface (CGI)
  - 5.2.2 Java Servlet Technology
  - 5.2.3 Advantages
  - 5.2.4 Servlet Application Architecture
  - 5.2.5 How a Servlet Works?
  - 5.2.6 Type of Servlets
    - 5.2.6.1 Generic Servlets
    - 5.2.6.2 HTTP Servlets
  - 5.2.7 Servlet Life Cycle
  - 5.2.8 javax.servlet Package
  - 5.2.9 The Java Servlet API
- 5.3 Using Servlets
  - 5.3.1 Interfaces of Servlets
  - 5.3.2 Request and Responses
  - 5.3.3 GenericServlet Class
  - 5.3.4 Http Request Methods
  - 5.3.5 HttpServlet Class
  - 5.3.6 HttpServletRequest Interface
  - 5.3.7 HttpServletResponse Interface
- 5.4 Session Management
  - 5.4.1 Concept of Session
  - 5.4.2 Session Tracking
  - 5.4.3 What is Session Management?
    - 5.4.3.1 URL Rewriting
    - 5.4.3.2 Hidden Fields
    - 5.4.3.3 Cookies
    - 5.4.3.4 Session Objects
  - 5.4.4 Servlet Filters
  - 5.4.5 Servlet Chaining
  - 5.4.6 Servlet Redirection

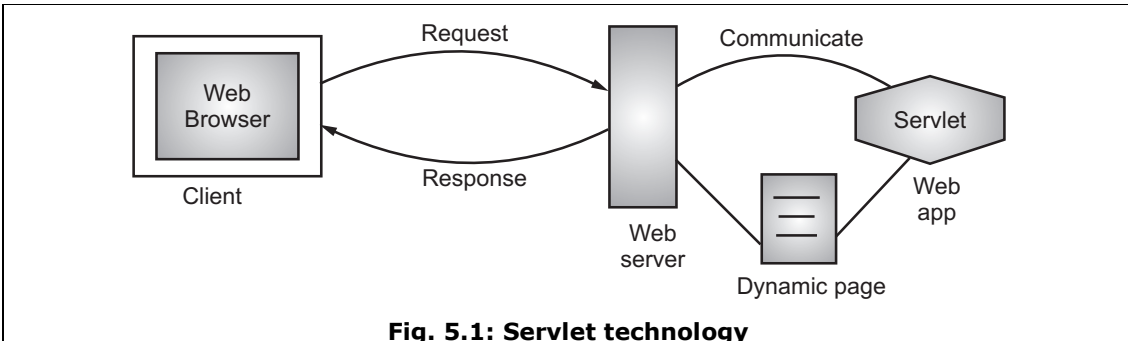
- 5.5 JSP
  - 5.5.1 Advantages of JSP over Servlet
  - 5.5.2 JSP Architecture
  - 5.5.3 Scriptlet
  - 5.5.4 Expression
  - 5.5.5 Declarations
  - 5.5.6 Actions
  - 5.5.7 Implicit Objects
  - 5.5.8 Directives
  - 5.5.9 Life Cycle of a JSP Page
  - 5.5.10 JSP TLD
  - 5.5.11 JSTL
- 5.6 Java Beans
  - Important Points
  - Practice Questions

### **Objectives**

- To Write Web Based Applications using Servlets, JSP and Java Beans
  - To Learn about Cookies, Session Tracking
- 

## **5.1 INTRODUCTION**

- Since, the emergence of the Internet, Web technologies have become more and more important and web applications become more and more common.
- In the earlier days web pages were static i.e. a user request a resource and the server returns it.
- However, with the growth of commercial activities on the web, companies wanted to deliver dynamic content to their customers such as:
  - Performing bank transactions online,
  - Airline or Railway ticket booking,
  - News, and
  - Marketing.
- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Servlet technology is used to create web application. Servlet technology uses Java language to create web application.
- Web application are helper application that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that display current time.



**Fig. 5.1: Servlet technology**

- As Servlet technology uses Java, web application made using Servlet are secured, scalable and robust.

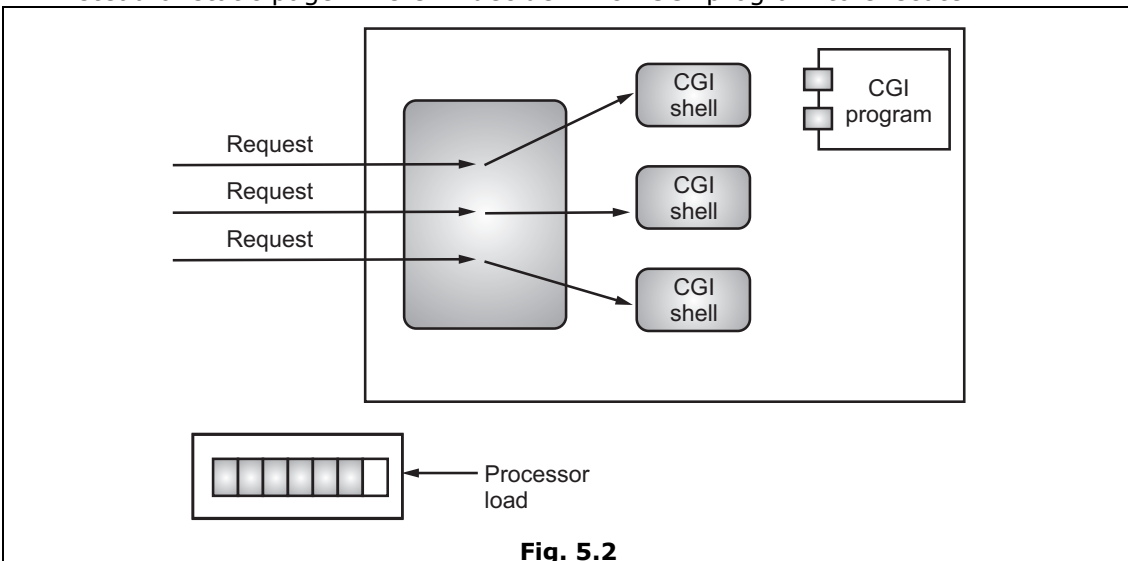
## 5.2 SERVLET

### 5.2.1 Common Gateway Interface (CGI)

- Before Java technology come along, there was a mechanism for writing interactive web applications i.e. CGI script.
- CGI programs provided a simple way to create web applications that accepts user input, queries a database and returns relevant result back to the web browser.
- A CGI program can be written in many languages such as C, C++, VBScript, python etc., although the most popular language for CGI programming is PERL.
- The CGI script to use is specified in the ACTION attribute of the FORM tag. The CGI script is the program that resides on the server.

#### Working of CGI:

- Fig. 5.2 shows working of CGI. User clicks a link that has URL to a dynamic page instead of static page. The URL decide which CGI program to execute.



**Fig. 5.2**

- Web Server run the CGI program in separate OS shell. The shell include OS environment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and sent it back to the web browser.
- CGI is often a good mechanism to use because it is well established and system administrators are familiar with it. But it has a lot of disadvantages.

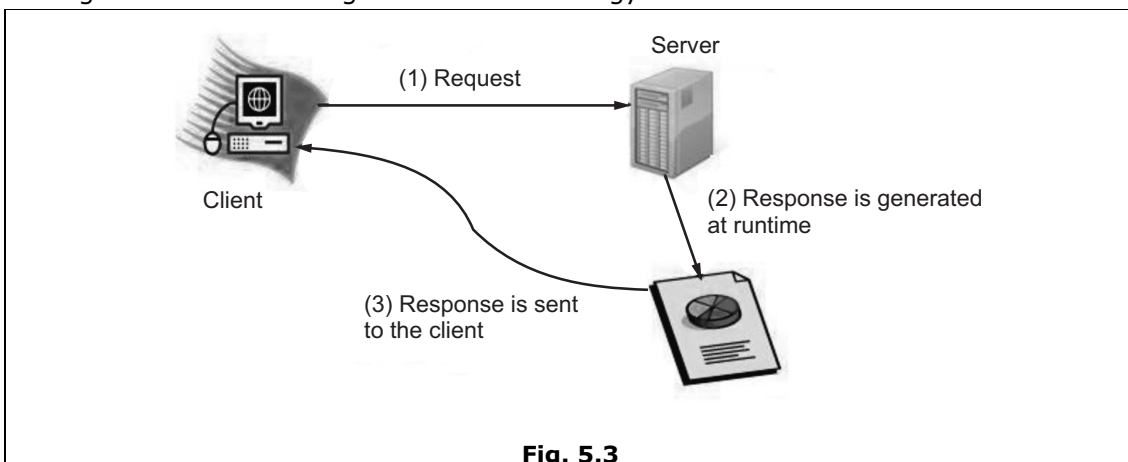
**Disadvantages of CGI:**

1. CGI has lack of scalability and reduced speed. Each time a request is received by the web server, an entirely new process thread is created.
  2. High response time because CGI program execute in their own OS shell.
  3. A process thread consumes a lot of server side resources in multiuser situation.
  4. Difficult for beginners to program.
  5. Sharing of resources such as database connections between scripts or multiple calls to same script is not available.
  6. CGI are not always secure or object-oriented.
  7. CGI is platform-dependent.
- Sun introduced servlets as a way to have thin, dynamic web client. However, it was the Java Servlet Technology which actually replaced CGI almost entirely.
  - When you click or type in a URL the following things happen:
    - The client browser establishes a TCP/IP connection with the server.
    - The browser sends a request to the server.
    - The server sends response to the client
    - The server close the connection.

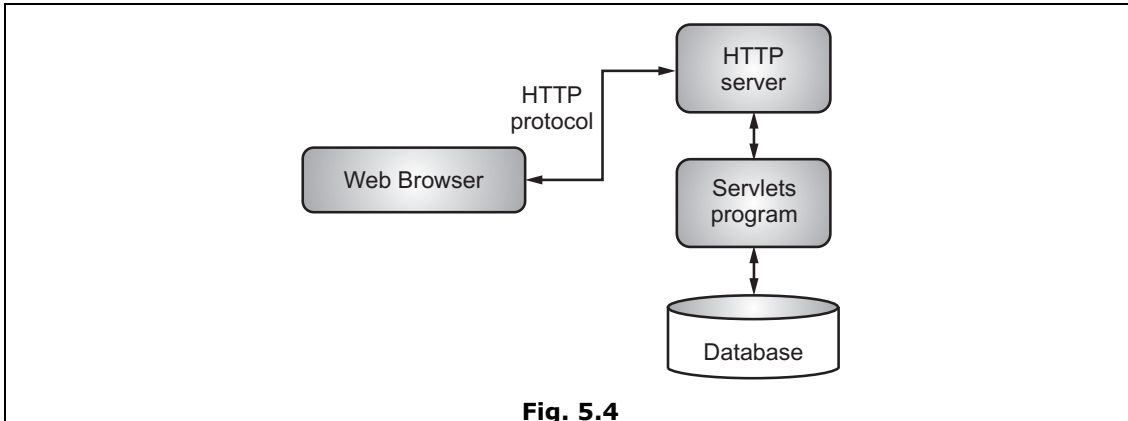
**5.2.2 Java Servlet Technology**

- A java servlet is a server side program that services HTTP requests and returns the result as HTTP responses.
- A servlet:
  - is a Program written using java that runs in a server application to answer clients requests.
  - is supported by virtually all web servers and application servers.
  - solves the performance problem by executing all requests as thread in one process.
  - is a technology i.e. used to create web application.
  - is an API that provides many interfaces and classes including documentations.
- When a user issues a request for a URL that corresponds to a java servlets, the server hands the request off to the servlet (program on server side) for processing. The servlet dynamically produces a response to the request, typically an HTML web page and sends it back to the requesting web browser.

- Servlets provides an object-oriented and extensible middle tier for web server based applications.
- Web servers generally cannot talk to databases – A web server alone cannot create web page content using data held in database. This information cannot make accessible to public through the web server.
- Here, servlets helps to extend the functionality of a web server. Web server uses servlets ability to access a database table, extract the data and convert this data into a format acceptable to a web server for delivery to a client browser through the Internet.
- Servlet technology is used to create web application (resides at server side and generates dynamic web page).
- Servlet technology is robust and scalable as it uses the java language. Before Servlet, CGI (Common Gateway Interface) scripting language was used as a server-side programming language.
- There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.
- 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.
- Fig. 5.3 shows working of servlet technology.



- Fig. 5.4 shows the position of Servlets in a Web Application.



**Fig. 5.4**

### Tasks of Servlets:

1. Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
2. Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
3. Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
4. Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
5. Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

### Java Servlet Development Kit:

- The Java Servlet Development Kit (JSDK) can be used to develop and test server extensions based on the servlet API. Included is a standalone server (called `servletrunner`) that can be used to test servlets before running them in a servlet-enabled web server.
- The JSDK serves as the reference implementation for the Java Servlet API.
- This release will run on top of JDK 1.1.x. If you are interested in developing servlets with JDK 1.2, there is no need to use this JSDK as the servlet API is bundled with JDK1.2.

### 5.2.3 Advantages

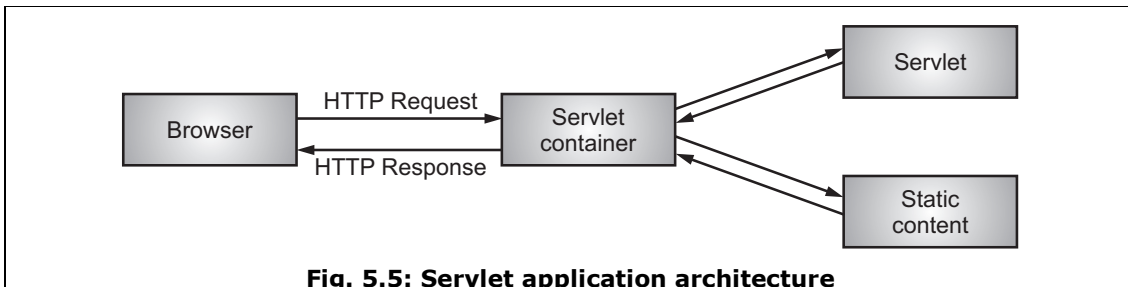
---

- Servlet offers the following benefits (Advantages) that are not necessarily available in other technologies:
  1. **Performance:** The performance of servlets is superior to CGI because there is no process creation for each client request. Instead, each request is handled by the servlet container process. After a servlet is finished processing a request, it stays resident in memory, waiting for another request.
  2. **Portability:** Similar to other Java technologies, servlet applications are portable. You can move them to other operating systems without serious hassles.
  3. **Rapid development cycle:** As a Java technology, servlets have access to the rich Java library, which helps speed up the development process.
  4. **Robustness:** Servlets are managed by the Java Virtual Machine (JVM). As such, you do not need to worry about memory leak or garbage collection, which helps you write robust applications.
  5. **Widespread acceptance:** Java is a widely accepted technology. This means that numerous vendors work on Java-based technologies. One of the advantages of this widespread acceptance is that you can easily find and purchase components that suit your needs, which saves precious development time.
  6. **Secure:** Servlet technology is very secured because of it uses java language.

### 5.2.4 Servlet Application Architecture

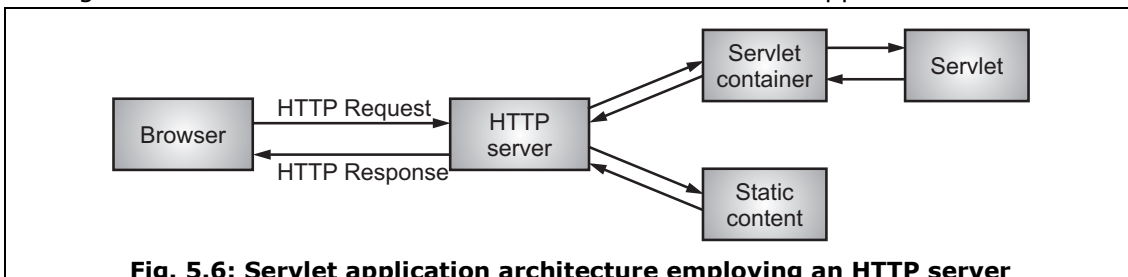
---

- A servlet is a Java class that can be loaded dynamically, (when the request will come from client side) and run by a special web server.
- This servlet-aware web server is called a servlet container, which also was called a servlet engine in the early days of the servlet technology.
- Servlets interact with clients via a request-response model based on HTTP. Because servlet technology works on top of HTTP, a servlet container must support HTTP as the protocol for client requests and server responses.
- However, a servlet container also can support similar protocols, such as HTTPS (HTTP over SSL) for secure transactions.
- Fig. 5.5 provides the architecture of a servlet application.



**Fig. 5.5: Servlet application architecture**

- In a JSP application, the servlet container is replaced by a JSP container. Both the servlet container and the JSP container often are referred to as the web container or servlet/JSP container, especially if a web application consists of both servlets and JSP pages.
- In Fig. 5.5, a servlet application also can include static content, such as HTML pages and image files.
- Allowing the servlet container to serve static content is not preferable because the content is faster if served by a more robust HTTP server, such as the Apache web server or Microsoft Internet Information Server.
- As such, it is common practice to put a web server at the front to handle all client requests. The web server serves static content and passes to the servlet containers all client requests for servlets.
- Fig. 5.6 shows a more common architecture for a servlet application.

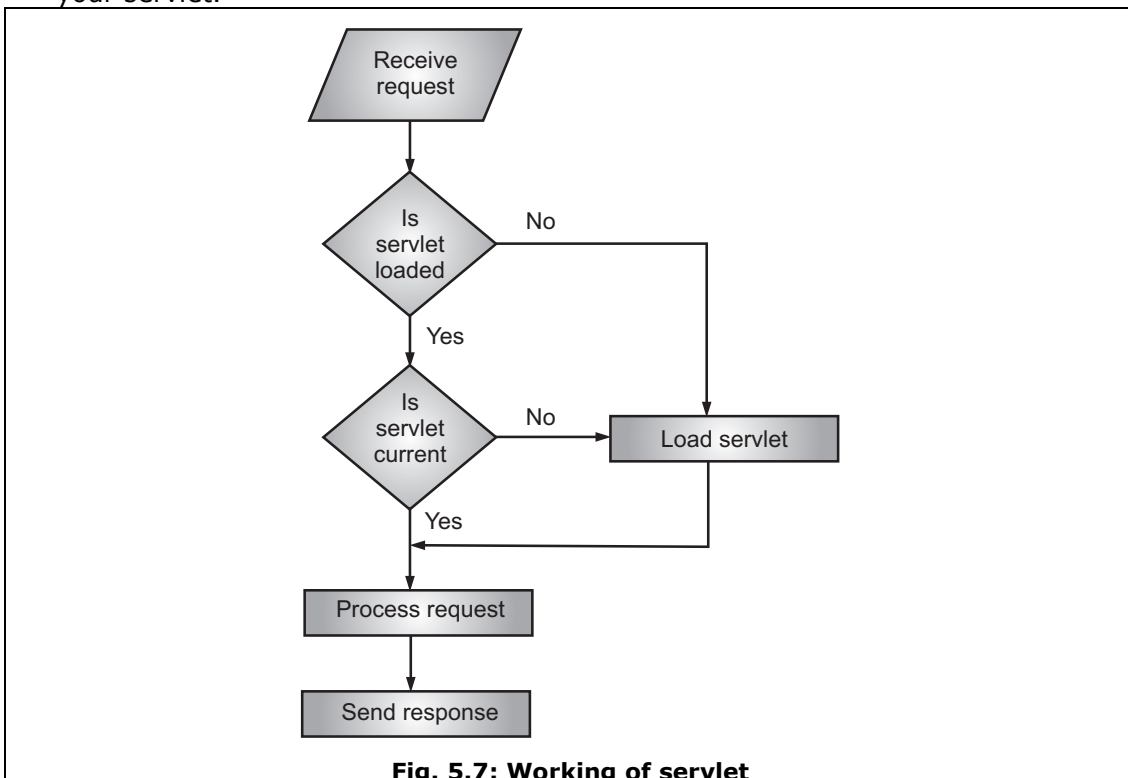


**Fig. 5.6: Servlet application architecture employing an HTTP server**

### 5.2.5 How a Servlet Works?

- A servlet is a java program that run on web server.
- This program will start running when there is request from client side.
- A servlet is loaded by the servlet container the first time the servlet is requested.
- When client sends a request to server, the user request is given to server, the server run servlet and gives user request to servlet, servlet perform operation and generate result and returns the result as response to the server, which in turn sends the response back to the user.
- After that, the servlet stays in memory waiting for other requests - it will not be unloaded from the memory unless the servlet container sees as shortage of memory.

- Each time the servlet is requested, however, the servlet container compares the timestamp of the loaded servlet with the servlet class file. If the class file timestamp is more recent, the servlet is reloaded into memory.
- This way, you do not need to restart the servlet container every time you update your servlet.



**Fig. 5.7: Working of servlet**

## 5.2.6 Types of Servlets

- There are two types of servlets, GenericServlet and HttpServlet.
- GenericServlet defines the generic or protocol independent servlet.
- HttpServlet is subclass of GenericServlet and provides some http specific functionality like doGet and doPost methods.

### 5.2.6.1 Generic Servlets

- It extend javax.servlet.GenericServlet.
- Generic servlets are protocol independent.
- They contain no inherent HTTP support or any other transport protocol.

### 5.2.6.2 HTTP Servlets

- It extend javax.servlet.HttpServlet.
- They have built-in HTTP protocol support and are more useful in a Sun Java System Web Server environment.

- For both servlet types, you implement the constructor method `init()` and the destructor method `destroy()` to initialize or deallocate resources.
- All servlets must implement a `service()` method, which is responsible for handling servlet requests.
- For generic servlets, simply override the `service` method to provide routines for handling requests.
- HTTP servlets provide a `service` method that automatically routes the request to another method in the servlet based on which HTTP transfer method is used.
- So, for HTTP servlets, override `doPost()` to process POST requests, `doGet()` to process GET requests, and so on.

### 5.2.7 A Servlet Life Cycle

- A java program is converted to servlet by implementing servlet interface. This interface in the `javax.servlet` package is the source of all activities in servlet programming.
- Servlet is the central abstraction of the Java servlet technology. Every servlet (java program) you write must implement this `javax.servlet.Servlet` interface, either directly or indirectly.
- A servlet life cycle can be defined as "the entire process from its creation till the destruction". The following are the paths followed by a servlet,
  1. The servlet is initialized by calling the `init()` method.
  2. The servlet calls `service()` method to process a client's request.
  3. The servlet is terminated by calling the `destroy()` method.
  4. Finally, servlet is garbage collected by the garbage collector of the JVM.
- Each servlet has the some life cycle steps as shown in Fig. 5.8.

**Step 1:** A server loads and initializes the servlet.

**Step 2:** The servlet handles zero or more client requests.

**Step 3:** The server removes the servlet (some servers do this step only when they shut down).

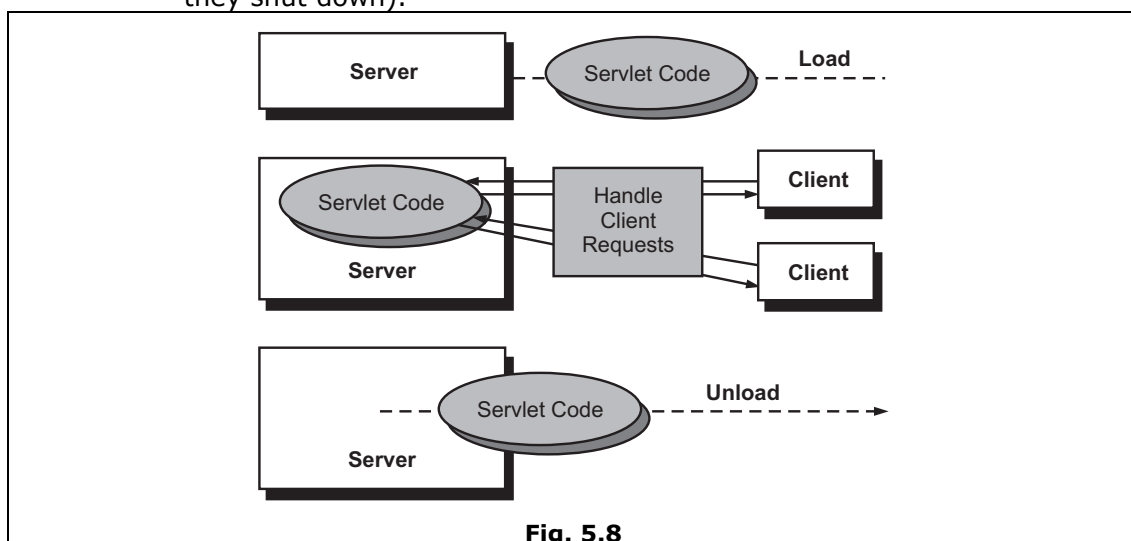
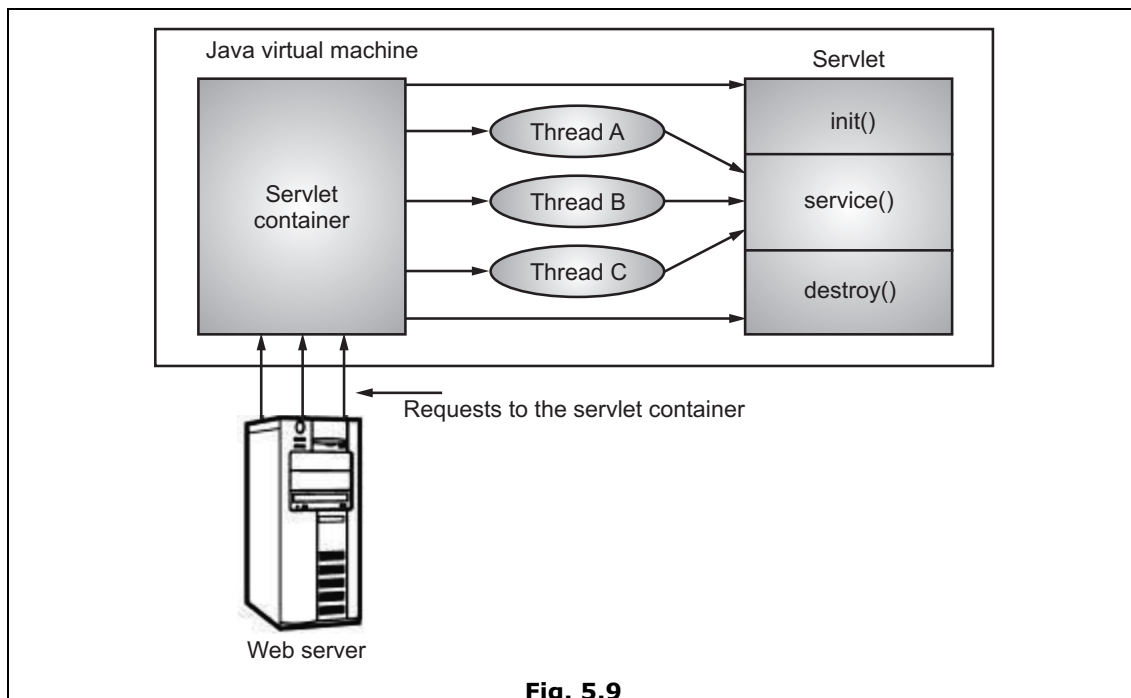


Fig. 5.8

- Fig. 5.9 depicts a typical servlet life-cycle scenario:
  1. First the HTTP requests coming to the server are delegated to the servlet container.
  2. The servlet container loads the servlet before invoking the service() method.
  3. Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.

**Fig. 5.9**

- The life cycle of a servlet is determined by three of its methods i.e., init, service and destroy, (See Fig. 5.9).

### 1. init() Method:

- The init method is called by the servlet container after the servlet class has been instantiated.
- The servlet container calls this method exactly once to indicate to the servlet that the servlet is being placed into service.
- You can override this method to write initialization code that needs to run only once, such as loading a database driver, initializing values, and so on. In other cases, you normally leave this method blank.
- The signature/syntax of this method is as follows:

```
public void init(ServletConfig config) throws ServletException
```

- The init method is important because the servlet container passes a ServletConfig object, which contains the configuration values stated in the web.xml file for this application.

This method also can throw a ServletException.

## 2. Service() Method:

- The service method is called by the servlet container after the servlet's init method to allow the servlet to respond to a request.
- Servlets typically run inside multithreaded servlet containers that can handle multiple requests concurrently.
- Therefore, you must be aware to synchronize access to any shared resources, such as files, network connections and the servlet's class and instance variables.
- This method has the following signature/syntax:

```
public void service(ServletRequest request, ServletResponse  
response) throws ServletException, java.io.IOException
```

- The servlet receives request from server and sends output as response back to client.
- The request of client is converted into the object of ServletRequest and servlet response is sent to client in the form of object of ServletResponse.
- The servlet container passes a ServletRequest object and the ServletResponse object to servlet. The ServletRequest object contains the client's request and the ServletResponse contains the servlet's response.
- These two objects are important because they enable you to write code that determines output of servlet.
- The service method throws a ServletException if an exception occurs that interferes with the servlet's normal operation.
- The service method also can throw a java.io.IOException if an input or output exception occurs during the execution of this method.
- As the name implies, the service method exists so that you can write code that makes the servlet function the way it is supposed to.

## 3. Destroy() Method:

- The servlet container calls the destroy method before removing a servlet instance from service. This normally happens when the servlet container is shut down or the servlet container needs some free memory.
- This method is called only after all threads within the servlet's service method have exited or after a timeout period has passed. After the servlet container calls this method, it will not call the service method again on this servlet.
- The destroy method gives the servlet an opportunity to clean up any resources that are being held, (For example, memory, file handles and threads).

- The signature/syntax of this method is as follows:

```
public void destroy()
```

### Demonstrating Life Cycle of a Servlet:

- Program 5.1 contains the code for a servlet named PrimitiveServlet, a very simple servlet that exists to demonstrate the life cycle of a servlet.
  - The PrimitiveServlet class implements javax.servlet.Servlet, (as all servlets must) and provides implementations for all the methods of servlet.
  - What it does is very simple. Each time any of the init, service or destroy methods is called, the servlet writes the method's name to the console.
- 

### Program 5.1: PrimitiveServlet.java.

```
import javax.servlet.*;
import java.io.IOException;
public class PrimitiveServlet implements Servlet
{
    public void init(ServletConfig config) throws ServletException
    {
        System.out.println("init");
    }
    public void service(ServletRequest request, ServletResponse
        response) throws ServletException, IOException
    {
        System.out.println("service");
    }
    public void destroy()
    {
        System.out.println("destroy");
    }
    public String getServletInfo()
    {
        return null;
    }
    public ServletConfig getServletConfig()
    {
        return null;
    }
}
```

---

- After you compile the source code into the myApp\WEB-INF\classes directory, add the servlet to the **web.xml** under the name Primitive.

- **Web.xml File for PrimitiveServlet:**

```
<web-app>
<servlet>
  <servlet-name>PrimitiveServlet</servlet-name>
<servlet-class>PrimitiveServlet</servlet-class>
</servlet>
  <servlet-mapping>
    <servlet-name>PrimitiveServlet</servlet-name>
    <url-pattern>/PrimitiveServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

- You should then be able to call this servlet from your browser by typing the following URL:

```
http://localhost:8080/myApp/Primitive
```

- The first time the servlet is called, the console displays these two lines:

```
init
service
```

- This tells you that the init method is called, followed by the service method. However, on subsequent requests, only the service method is called. The servlet adds the following line to the console:

```
service
```

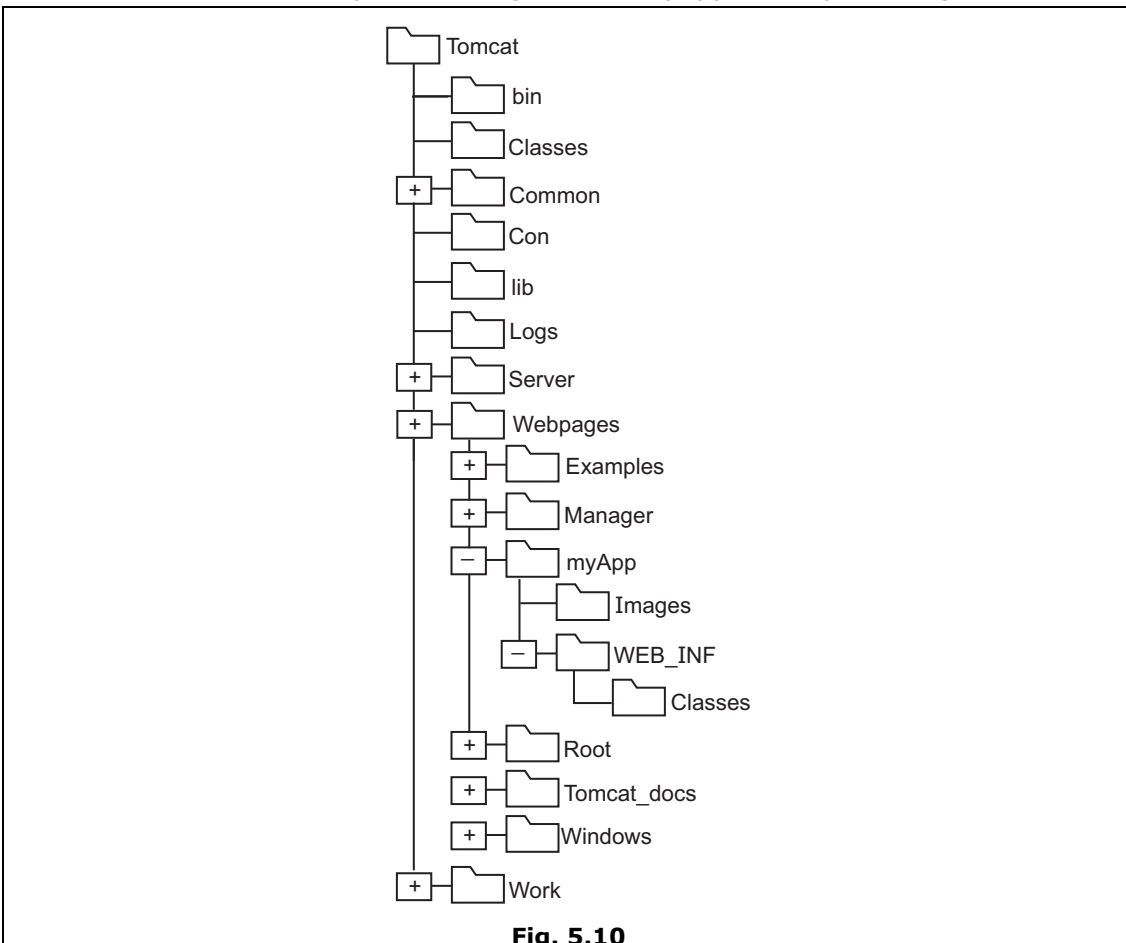
- This proves that the init method is called only once.

**Tomcat Servlet Container (Steps to Run and Save Servlet):**

- A number of servlet containers are available today. The most popular one and the one recognized as the official servlet/JSP container is Tomcat.
- Originally designed by Sun Microsystems, Tomcat by itself is a web server. This means that you can use Tomcat to service HTTP requests for servlets, as well as static files, (HTML, image files and so on).
- To write a servlet, you need at least version 1.2 of the Java Development Kit.
- The reference implementation for both servlets and JSP are not included in J2SE, but they are included in Tomcat. Tomcat is written purely in Java.
- After you have installed and configured Tomcat, you can put it into service.
- Basically, you need to follow six steps to go from writing your servlet to running it.
- These steps are summarized as follows:
  1. Create a directory structure under Tomcat for your application.
  2. Write the servlet source code. You need to import the javax.servlet package and the javax.servlet.http package in your source file.
  3. Compile your source code.
  4. Create a deployment descriptor (web.xml).
  5. Run Tomcat.
  6. Call your servlet from a web browser.

**Step 1: Create a Directory Structure Under Tomcat**

- When you install Tomcat, several subdirectories are automatically created under the Tomcat home directory. One of the subdirectories is webapps. The web apps directory is where you store your web applications.
- A web application is a collection of servlets and other content installed under a specific subset of the server's URL namespace.
- To create a directory structure for an application called myApp, follow these steps:
  1. Create a directory called myApp under the webapps directory. The directory name is important because this also appears in the URL to your servlet.
  2. Create the WEB-INF and Images directories under myApp and create a directory named classes under WEB-INF. The directory structure is shown in Fig. 5.10. The directory classes under WEB-INF is for your Java classes. If you have HTML files, put them directly under the myApp directory. You may also want to create a directory called images under myApp for all your image files.

**Fig. 5.10**

**Step 2: Write the Servlet Source Code:**

- In this step, you prepare your source code. You can write the source code yourself using your favorite text editor.
  - Program 5.2 shows a simple servlet called `TestingServlet`. The file is named `TestingServlet.java`. The servlet sends a few HTML tags and some text to the browser. For now, don't worry if you have not got a clue about how it works.
- 

**Program 5.2: TestingServlet.java.**

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

public class TestingServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Servlet Testing</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Welcome to the ServletTestingCenter");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

---

- Now, save your `TestingServlet.java` file to the `WEB-INF/classes` directory under `myApp`. Placing your source code here will make it accessible from a web browser.
- Static files, such as HTML files and image files, should be placed directly under the `myApp` directory or a directory under it.

**Step 3: Compile Your Source Code:**

- For your servlet source code to compile, you need to include the path to the `servlet.jar` file in your `CLASSPATH` environment variable.
- The `servlet.jar` is located in the `common\lib\` subdirectory of installed Tomcat.
- You can add the complete path to the `servlet.jar` file to your `CLASSPATH` environment variable.

- Again, if you have installed Tomcat under C:\ and named the install directory Tomcat, you must add C:\tomcat\ common\lib\servlet.jar to the CLASSPATH environment variable.
- Afterward, you can compile your source by simply typing the following:

```
javac TestingServlet.java
```

#### Step 4: Create the Deployment Descriptor:

- A deployment descriptor is an optional component in a servlet application. The descriptor takes the form of an XML document called web.xml and must be located in the WEB-INF directory of the servlet application.
- When present, the deployment descriptor contains configuration settings specific to that application. To create the deployment descriptor, you now need to create a web.xml file and place it under the WEB-INF directory under myApp.
- The web.xml for this example application must have the following content.
- Open notepad and type the following:

```
<web-app>
<servlet>
<servlet-name>TestingServlet</servlet-name>
<servlet-class>TestingServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>TestingServlet</servlet-name>
<url-pattern>/TestingServlet</url-pattern>
</servlet-mapping>
</web-app>
```

- Save this as web.xml in WEB-INF folder.
- The web.xml file has one element, web-app. You should write all your servlets under <web-app>.
- For each servlet, you have a <servlet> element and you need the <servlet-name> and <servlet\_class> elements. The <servlet-name> is the name for your servlet, by which it is known Tomcat.
- The <servlet-class> is the compiled file of your servlet without the .class extension. Having more than one servlet in an application is very common. For every servlet, you need a <servlet> element in the web.xml file.

#### Step 5: Run Tomcat:

- If Tomcat is not already running, you need to start it.
- Go to bin folder of Tomcat and click on Tomcat (service Runner).

**Step 6: Call Your Servlet from a Web Browser:**

- Now, you can call your servlet from a web browser. By default, Tomcat runs on port 8080 in the myApp virtual directory under the servlet subdirectory. The servlet that you wrote in the preceding steps is named Testing. The URL for that servlet has the following format:

```
http://localhost:8080/myApp/TestingServlet
```

**Note:** To run each example in this chapter, you need to compile the source code and copy the resulting class file into the classes directory under the WEB-INF directory of your application.

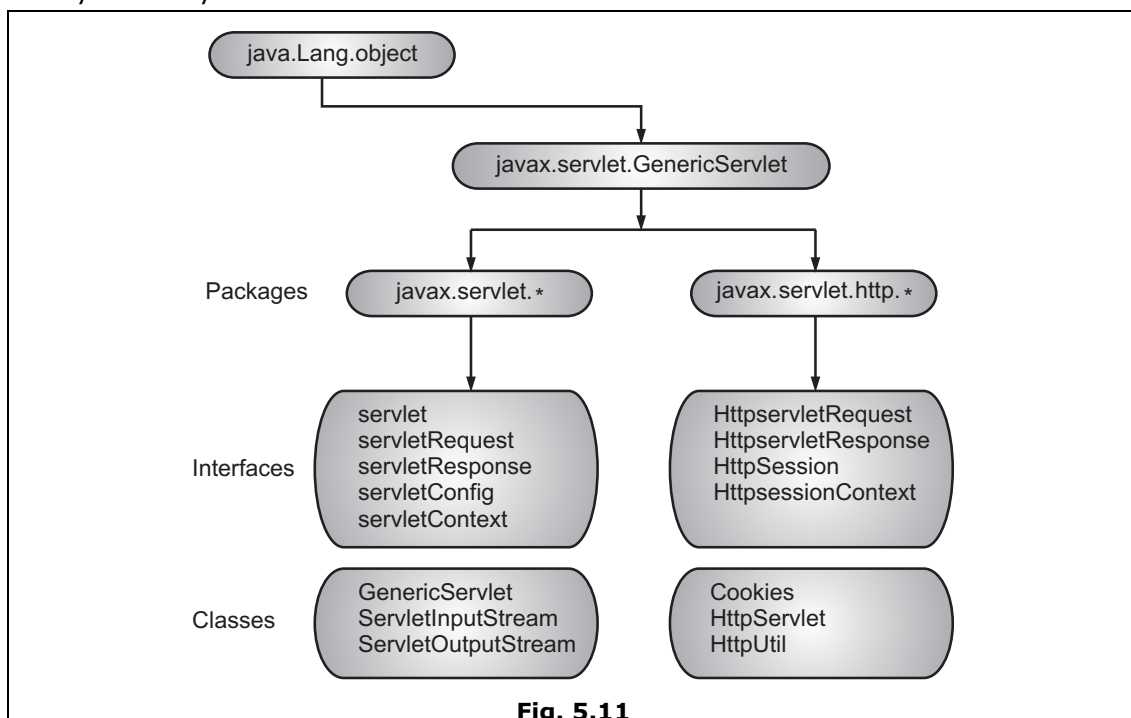
**5.2.8 javax.servlet Package**

- The javax.servlet package is the core of the servlet API.
- It includes the basic servlet interface which all servlets must implement in one form or another and an abstract GenericServlet class for developing basic servlets.
- As shown in Fig. 5.11, this package includes server interfaces and three classes. For communicating with the host server and client.
- The javax.servlet package contains seven interfaces, three classes and two exceptions.
- The seven interfaces are as follows:
  1. RequestDispatcher
  2. Servlet
  3. ServletConfig
  4. ServletContext
  5. ServletRequest
  6. ServletResponse
  7. SingleThreadModel.
- The three classes are as follows:
  1. GenericServlet
  2. ServletInputStream
  3. ServletOutputStream.
- And, finally, the exception classes are:
  1. ServletException.
  2. UnavailableException.

**5.2.9 The Java Servlet API**

- Servlet code specification can be written using any editor. This file will be saved with the extension .java and must be compiled using java compiler.

- Java provides a servlet Application Programming Interface (API) which is a class library for implementing servlets.
- The java API forms a standard interface for developing web applications, regardless of underlying of operating system without the implementation of java API it is impossible to develop dynamic applications.
- Servlets can access the entire family of java APIs. The servlet API is the class library using which requests can be processed and responses can be constructed dynamically.



## 5.3 USING SERVLETS

### 5.3.1 Interfaces of Servlets

#### 1. Servlet Interface:

- Servlet interface is a collection of empty method signatures.
- A servlet must directly or indirectly, (by extending the generic class or httpServletclass) implement the servlet interface.
- This interface hold method signature that bring the following servlet functionalities.
  - Initializing the servlet.
  - Handling a client request.
  - Destroying a servlet.

- Following are the methods available in this interface.

Methods	Description
1. <code>init()</code>	<code>init()</code> is used for initializing the Servlet parameters provided by the <code>ServletConfig</code> object. <code>init()</code> called only once when the Servlet is first loaded. It is commonly used to initialize resources to be used by a Servlet when requests are received.
2. <code>destroy()</code>	<code>destroy()</code> is also called only once immediately before the Servlet is unloaded. <code>destroy()</code> used to clear all retained resources such as database connection, threads, file handles and so on. This method is overridden in order to free up any resources being used by the Servlet.
3. <code>service()</code>	<code>service()</code> is the actual heart of the HTTP Request-Response model. <code>service()</code> called to handle a single client request.
4. <code>getServletConfig()</code>	<code>getServletConfig()</code> object for initializing the Servlet's parameters.
5. <code>getServletInfo()</code>	Provides the Servlet metadata such as author, Servlet version and other copyright information. <code>getServletInfo()</code> needs to be overridden inside the Servlet for it to return the required information.

## 2. ServletContext Interface:

- All servlets belongs to one servlet context. `ServletContext` can only be called at context Initialization time.
- `ServletContext` interface provides a lot of methods which allow communicating with the server such as:
  - finding path information.
  - accessing other servlets running on the server.
  - writing to the server log file.
- `ServletContext` allows enabling applications to load servlets.
- A servlet gets a reference to its `ServletContext` object through the `ServletConfig` object.

## 3. The ServletConfig Interface:

- The `ServletConfig` interface is implemented by the server. It allows a servlet to obtain configuration data when it is loaded.

- The ServletConfig objects help servers pass initialization and context information to servlet such as,
  - A series of initialization parameters.
  - A ServletContext object, which provides information about the server environment.

#### 4. The ServletRequest Interface:

- When a servlet accepts a call from a client, it receives two objects, one is ServletRequest and other is ServletResponse.
- ServletRequest interface encapsulates the communication from the client to the server.
- ServletRequest interface allows the servlet to access information such as:
  - Names of the parameters passed by the client.
  - The protocol such as http POST and PUT methods being used by client.
  - The names of the remote host that made the request.
  - The server that received it.
  - An InputStream for reading binary data from the request body.
- The following are the commonly used methods available in this interface:

Methods	Description
<code>getAttribute()</code>	This method returns the value of the named attribute as an object.
<code>getAttributeNames()</code>	Returns an Enumeration containing the names of the attributes to this request.
<code>getParameter()</code>	Returns the value of a request parameter as a String.
<code>getParameterNames()</code>	Returns an Enumeration of String objects containing the names of the parameters to this request.
<code>getParameterValues()</code>	Returns an array of String objects containing all of the values the given request parameter has.
<code>getRequestDispatcher()</code>	Returns a RequestDispatcher object that acts as a wrapper for the resource located at the given path.

- The ServletRequest interface provides important methods that enable you to access information about the user.
- For example, the `getParameterNames` method returns an Enumeration containing the parameter names for the current request.
- To get the value of each parameter, the ServletRequest interface provides the `getParameter` method.
- The `getRemoteAddress` and `getRemoteHost` methods are two methods that you can use to retrieve the user's computer identity.

### 5. ServletResponse Interface:

- ServletResponse interface provides methods to the servlet for replying to the client.
- ServletResponse interface allows servlet:
  - to set the content length and type of the reply.
  - provides an output stream and a writer.
- Through servletResponse interface, the servlet can send the reply data.
- Subclasses of servletResponse provide the servlet with more protocol-specific capabilities for e.g. HttpServletResponse contains methods that allow the servlet to manipulate Http-specific header information.
- The following are the commonly used methods available in this interface:

Methods	Description
<code>getLocale()</code>	Returns the locale assigned to the response.
<code>getWriter()</code>	Returns a ServletOutputStream suitable for writing binary data in the response.
<code>reset()</code>	Returns a PrintWriter object that can send character text to the client.
<code>setLocale()</code>	Clears any data that exists in the buffer as well as the status code and headers.
<code>setContentType(String)</code>	This method is used to set the content type to be sent to user as output.

- The ServletResponse interface represents the response to the user.
- The most important method of this interface is `getWriter`, from which you can obtain a `java.io.PrintWriter` object that you can use to write HTML tags and other text to the user, (send output of servlet to user).

#### 5.3.2 Request and Responses

- Requests and Responses are what a web application is all about.
- In a servlet application, a user using a web browser sends a request to the servlet container and the servlet container passes the request to the servlet.
- In a servlet paradigm, the user request is represented by the ServletRequest object passed by the servlet container as the first argument to the service method.
- The service method's second argument is a ServletResponse object, which represents the response to the user.

#### Servlet Example:

- In Program 5.3, Listings show a ServletRequest object in action. The example consists of an HTML form in a file named `index.html` that you need to put in the application directory—that is, under `myApp`—and a servlet called `RequestDemoServlet`.

**Program 5.3:** index.html

```
<HTML>
<HEAD>
<TITLE>Sending a request</TITLE>
</HEAD>
<BODY>
<FORM ACTION="DemoServlet">
<BR><BR>
Author: <INPUT TYPE="TEXT" NAME="Author">
<INPUT TYPE="SUBMIT" NAME="Submit">
<INPUT TYPE="RESET" VALUE="Reset">
</FORM>
</BODY>
</HTML>
```

**Program 5.4:** RequestDemoServlet.

```
import javax.servlet.*;
import java.util.Enumeration;
import java.io.IOException;
public class DemoServlet implements Servlet
{
    public void init(ServletConfig config) throws ServletException
    {
    }
    public void destroy()
    {
    }
    public void service(ServletRequest request, ServletResponse
        response) throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter Out = response.getWriter();
        out.println("Server Port: " + request.getServerPort());
        out.println("Server Name: " + request.getServerName());
        out.println("Protocol: " + request.getProtocol());
        out.println("Content Type: " + request.getContentType());
        out.println("Content Length: " + request.getContentLength());
        out.println("Remote Address: " + request.getRemoteAddr());
        out.println("Remote Host: " + request.getRemoteHost());
        Enumeration parameters = request.getParameterNames();
        out.println("Parameter Value: " +
            request.getParameter("Author"));
    }
}
```

```
public String getServletInfo()
{
    return null;
}
public ServletConfig getServletConfig()
{
    return null;
}
}
```

- Save the servlet with extension .java in classes folder of myApp/WEB-INF.
- To run the example, first request the index.html file by using the following URL:  
`http://localhost:8080/myApp/index.html`
- When you submit the form, the request will be forwarded from browser to DemoRequest servlet. you should see the list of attribute names and values in your console.

#### **Problem With Servlet Interface:**

- Everything works fine, but there are two annoying things that you have probably noticed:
  1. You have to provide implementations for all five methods of the Servlet interface, even though most of the time you only need one. This makes your code look unnecessarily complicated.
  2. The ServletConfig object is passed to the init method. You need to preserve this object to use it from other methods. This is not difficult, but it means extra work. This Problem is solved by GenericServlet.

### **5.3.3 GenericServlet Class**

- This is First type of Servlet.
- `javax.servlet.GenericServlet` class provides the basic implementation of the servlet interface and `ServletConfig` interface.
- `GenericServlet` makes writing servlets easier and simpler.
- The `javax.servlet` package provides a wrapper class called `GenericServlet` that implements two important interfaces from the `javax.servlet` package i.e., `Servlet` and `ServletConfig`, as well as the `java.io.Serializable` interface.
- The `GenericServlet` class provides implementations for all methods, most of which are blank. You can extend `GenericServlet` and override only methods that you need to use. Clearly, this looks like a better solution.
- Once, you convert your program from servlet to `GenericServlet` you have to only write service method.

- The code in Program 5.5 is a servlet called SimpleServlet that extends GenericServlet.
- The code provides the implementation of the service method that sends some output to the browser. Because the service method is the only method you need, only this method needs to appear in the class.
- Compared to all servlet classes that implement the javax.servlet.Servlet interface directly, SimpleServlet looks much cleaner and clearer.

---

**Program 5.5:** Extending GenericServlet.

```
import javax.servlet.*;
import java.io.IOException;
import java.io.PrintWriter;
public class SimpleServlet extends GenericServlet
{
    public void service(ServletRequest request, ServletResponse
        response)throws ServletException, IOException
    {response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD>");
    out.println("<TITLE>");
    out.println("Extending GenericServlet");
    out.println("</TITLE>");
    out.println("</HEAD>");
    out.println("<BODY>");
    out.println("Extending GenericServlet makes your code simpler.");
    out.println("</BODY>");
    out.println("</HTML>");
    }
}
```

---

**Problem with GenericServlet:**

1. The GenericServlet is protocol depended servlet.
2. Protocol depended means it works with all protocol.
3. It gives services when the request is coming from any protocol ex FTP,SMTP,POP3.
4. But most of the time the user sends request through http protocol.
5. GenericServlet provide services to http protocol also but doesnot support all functionality(methods) of http protocol.

So we need a servlet that support full functionality of http protocol.

### 5.3.4 HTTP request Methods

- HttpServlet servlet is used for such cases where we require all the functionality (methods) of http protocol.
- Each HTTP request can use one of the many request methods as specified in the HTTP standards.
- The HTTP request methods and the descriptions of each method are given in the following table:

Method	Description
GET	GET is the simplest and probably, most used HTTP method. GET simply retrieves the data identified by the URL. If the URL refers to a script (CGI, servlet, and so on), it returns the data produced by the script.
HEAD	The HEAD method provides the same functionality as GET, but HEAD only returns HTTP headers without the document body.
POST	Like GET, POST is also widely used. Typically, POST is used in HTML forms. POST is used to transfer a block of data to the server in the entity body of the request.
OPTIONS	The OPTIONS method is used to query a server about the capabilities it provides. Queries can be general or specific to a particular resource.
DELETE	The DELETE method is used to delete a document from the server. The document to be deleted is indicated in the URI (Uniform Resource Identifier) section of the request.
PUT	The PUT method is a complement of a GET request and PUT stores the entity body at the location specified by the URI. It is similar to the PUT function in FTP.
TRACE	The TRACE method is used to trace the path of a request through firewall and multiple proxy servers. TRACE is useful for debugging complex network problems and is similar to the trace route tool.

#### Javax.servlet.http Package:

- This is second type of Servlet:
- This package contains a number of interfaces and classes that are commonly used by servlet developers. This package is used by servlet developers. This package is used to develop servlets that supports the http protocol.
- Some core interfaces provided by this package are:

Interface	Description
HttpServletRequest	Enables servlets to read data from an HTTP request.
HttpServletResponse	Enables servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written .
HttpSessionBindingListener	Informs an object that it is bound to or unbound from a session.

- Following are the some of the core classes provided by this package.

Class	Description
HttpServlet	Provides methods to handle HTTP requests and responses.
Cookie	Allow state information to be stored on a client machine.
HttpEventSession	Encapsulates session charged event.
HttpSessionBindingEvent	Indicates when a listener is bound to or unbound from a session value or that a session attribute is changed.

### 5.3.5 HttpServlet Class

- The HttpServlet class extends the javax.servlet.GenericServlet class. The HttpServlet class also adds a number of interesting methods for you to use.
- The most important are the six doxxx methods that get called when a related HTTP request method is used.
- The six methods are doPost, doPut, doGet, delete, doOptions and doTrace.
- Each doxxx method is invoked when a corresponding HTTP method is used. For instance, the doGet method is invoked when the servlet receives an HTTP request that was sent using the GET method.
- Of the six doxxx methods, the doPost and the doGet methods are the most frequently used.
- The doPost method is called when the browser sends an HTTP request using the POST method.
- The POST method is one of the two methods that can be used by an HTML form.
- Consider the following HTML form at the client side:

```
<FORM ACTION="Register" METHOD="POST">
  <INPUT TYPE=TEXT Name="firstName">
  <INPUT TYPE=TEXT Name="lastName">
  <INPUT TYPE=SUBMIT>
</FORM>
```
- When the user clicks the Submit button to submit the form, the browser sends an HTTP request to the server using the POST method.
- The web server then passes this request to the Register servlet and the doPost method of the servlet is invoked.
- Using the POST method in a form, the parameter name/value pairs of the form are sent in the request body.

- For example, if you use the preceding form as an example and enter Tanmay as the value for firstName and Gurunani as the value for lastName, you will get the following result in the request body:

```
firstName=Tanmay
lastName=Gurunani
```

- An HTML form can also use the GET method; however, POST is much more often used with HTML forms.
- The doGet method is invoked when an HTTP request is sent using the GET method. GET is the default method in HTTP. When you type a URL, such as www.yahoo.com, your request is sent to Yahoo! using the GET method.
- If you use the GET method in a form, the parameter name/value pairs are appended to the URL.
- Therefore, if you have two parameters named firstName and lastName in your form, and the user enters Tanmay and Gurunani, respectively, the URL to your servlet will become something like the following:  

```
http://yourdomain/myApp/Register?firstName=Tanmay&lastName=Gurunani
```
- Upon receiving a GET method, the servlet will call its doGet method.
- The HTTP method used by the client request. Knowing the HTTP method, the service method simply calls the corresponding doxxx method.
- The following are the commonly used methods available in this class.

Methods	Description
doGet ()	Is called in response to an HTTP GET request [including conditional GET requests]. Overriding this method to support a GET request also automatically supports an HTTP HEAD request. A HEAD request is a GET request that returns no body in the response, only the request header fields.
doHead ()	Is called in response to an HTTP HEAD request. The client sends a HEAD request when it wants to see only the headers of a response such as Content-Type or Content-Length.
doPost ()	Is called in response to an HTTP POST request. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information such as credit card numbers.
doPut ()	Is called in response to an HTTP PUT request. The PUT operation allows a client to place a file on the server and is similar to sending a file by FTP.
doDelete ()	Is called in response to an HTTP DELETE request. The DELETE operation allows a client to remove a document or Web page from the server.

**Handling Http POST Request:**

- The doPost() method is called when the browser sends an HTTP request using the post method.
- Now lets develop one servlet that will handle the HTTP post request. The servlet is invoked when a form on a web page is submitted.
- The example contains two files. A web page is defined in getvalue.html and a servlet is defined in Register.java.

```
<html>
<body>
<Form Name = "form"
      ACTION = "RegisterServlet"
      METHOD = POST>
<INPUT TYPE = TEXT NAME = FIRST>
<INPUT TYPE = TEXT NAME = LAST>
<INPUT TYPE = SUBMIT VALUE = "SUBMIT">
</FORM>
</BODY>
</HTML>
```

- The above code defines a form that contains two text fields and a submit button. When user clicks the submit button to submit the form, the browser sends an HTTP request to server using the POST method.
- The source code for RegisterServlet.java is as follows:

**Program 5.6:**

```
import java.servlet.*;
import java.servlet.http.*;
import java.io.*;
public class RegisterServlet extends HttpServlet
{
    public void doPost(HttpServletRequest req,
        HttpServletResponse res)throws
        ServletException, IOException
    {
        string F1 = res.getParameter("FIRST");
        string L1 = res.getParameter("LAST");
        res.setContentType("text/html");
        PrintWriter PW = res.getWriter();
        pw.println("The centered name is:");
        pw.println(F1 + " " + L1);
        pw.close()
    }
}
```

- Using the POST method in a form, the parameter values of the form are sent to the servlet.
- For example, if you use the preceding form as an example and enter RATI as firstName and Yemul as lastName you will get the following result.

```
The Entered Name is
      RATI YEMUL
```

### Handling Http get Request:

- Now let develop one more servlet that handles an HTTP Get request.
- The servlet is invoked when a form on webpage is submitted. Again here also we have two files one html file and other is java source file.

```
<html>
<body>
<Form name = "Form1"
      ACTION = "RegisterServlet"
      METHOD = POST>
<INPUT TYPE = TEXT NAME = FIRST>
<INPUT TYPE = TEXT NAME = LAST>
<INPUT TYPE = SUBMIT VALUE = "SUBMIT">
</FORM>
</BODY>
</HTML>
```

- The java source code that handles get request is as follows:

---

### Program 5.7:

```
import java.servlet.*;
import java.servlet.http.*;
import java.io.*;
public class RegisterServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req, HttpServletResponse
res)throws ServletException, IOException
    {
        string F1 = res.getParameter("FIRST");
        string L1 = res.getParameter("LAST");
        res.setContentType("text/html");
        PrintWriter PW = res.getWriter();
        pw.println("The centered name is:");
        pw.println(F1 + " " + L1);
        pw.close()
    }
}
```

---

- When you use Get method in a form the parameter values are appended to the URL. Therefore, if you have two parameters name FirstName and LastName in our form and the user enters RATI and YEMUL respectively, the URL to your servlet will become something like the following:

```
http://Yourdomain/myapp/Register?
    firstName = RATI and lastName = YEMUL
```

Upon receiving a Get method, servlet will call its doGet() method.

### 5.3.6 HttpServletRequest Interface

- This interface is implemented by server. It enables servlet request to obtain information about a client request.
- This interface provides method for extracting HTTP parameters from the query string or the request body depending on the types of request such as Get or POST.
- HttpServletRequest Interface extends ServletRequest Interface to provide request information for HTTP servlet.
- This interface includes support for:
  - Cookies,
  - Session tracking, and
  - Access to HTTP header information.
- Some of the commonly available methods in this interface are:

Methods	Description
getCookies()	Returns an array containing all of the Cookie objects the client sent with this request.
getQueryString()	Returns the query string that is contained in the request URL after the path.
getSession()	Returns the current session associated with this request or if the request does not have a session, creates one.

### 5.3.7 HttpServletResponse Interface

- The HttpServletResponse interface is implemented by the server. It enables a servlet to formulate an HTTP response to a client.
- HttpServletResponse interface extends servletResponse interface to provide HTTP protocol specific functionality including response header and status codes.
- The HttpServletResponse interface provides several protocol-specific methods not available in the javax.servlet.ServletResponse interface.
- The HttpServletResponse interface extends the javax.servlet.ServletResponse interface.

- In the examples, in this chapter so far, you have seen that you always use two of the methods in `HttpServletResponse` when sending output to the browser: `setContentType` and `getWriter`.

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

- There is more to it, however, the `addCookie` method sends cookies to the browser. You also use methods to manipulate the URLs sent to the browser.
- Another interesting method in the `HttpServletResponse` interface is the `setHeader` method. This method allows you to add a name/value field to the response header.
- The Program 5.8 displays Login page that prompts the user to enter a user name and a password. If both are correct, the user will get Welcome message. If not, the user will see the error message.
- When the servlet is first requested, the servlet's `doGet` method is called. The `doGet` method then outputs the form.
- The user can then enter the user name and password and submit the form.

---

**Program 5.8: A Login Page.**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class LoginServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse
                response) throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String userName = request.getParameter("userName");
String password = request.getParameter("password");
if (userName != null && password != null &&
    userName.equals("jamesb") && password.equals("007"))
{
Out.println("WEL COME");
}
else
{
out.println("<HTML>");
out.println("<HEAD>");
```

```
out.println("<TITLE>Login</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("<BR>");
out.println("<BR>Please enter your user name and password.");
out.println("<BR><FORM METHOD=GET>");
out.println("<BR>User Name: <INPUT TYPE=TEXT NAME=username>");
out.println("<BR>Password: <INPUT TYPE=PASSWORD NAME=password>");
out.println("<BR><INPUT TYPE=SUBMIT VALUE=Submit>");
out.println("</FORM>");
out.println("</BODY>");
out.println("</HTML>");
}
}
}
```

---

**Programs:****1. Servlet to find factorial of number.**

- Now let develop one more servlet that handles an HTTP Get request.
- The servlet is invoked when a form on webpage is submitted. Again here also we have two files one html file and other is java source file.

```
<html>
<body>
<Form name = "Form1"
      ACTION = "FactorialServlet"
      METHOD = POST>
<INPUT TYPE = TEXT NAME =number>
<INPUT TYPE = SUBMIT VALUE = "SUBMIT">
</FORM>
</BODY>
</HTML>
```

- The java source code that handles get request is as follows:

```
import java.servlet.*;
import java.servlet.http.*;
import java.io.*;
public class FactorialServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)throws ServletException, IOException
```

```
{ int fact=1;
string F1 = res.getParameter("number");
int num=Integer.parseInt(F1);
res.setContentType("text/html");
PrintWriter out = res.getWriter();
for(int i=num;i>1;--i)
    fact=fact*I;
out.println("The factorial of "+num+" is:");
out.println(fact);
out.close()
}
```

---

## 2. Servlet to find largest of two number.

- Now let develop one more servlet that handles an HTTP Get request.
- The servlet is invoked when a form on webpage is submitted. Again here also we have two files one html file and other is java source file.

```
<html>
<body>
<Form name = "Form1"
    ACTION = "LargeServlet"
    METHOD = POST>
<INPUT TYPE = TEXT NAME =first>
<INPUT TYPE = TEXT NAME =second>
<INPUT TYPE = SUBMIT VALUE = "SUBMIT">
</FORM>
</BODY>
</HTML>
```

- The java source code that handles get request is as follows:

```
import java.servlet.*;
import java.servlet.http.*;
import java.io.*;
public class LargeServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)throws ServletException, IOException
    {
        string F1 = res.getParameter("first");
        int num=Integer.parseInt(F1);
        string F2 = res.getParameter("second");
```

```
        int num1=Integer.parseInt(F2);
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        if(num <num1)
            out.println("The largest is "+num1);
        else
            out.println("The largest is "+num);
        out.close()
    }
}
```

---

### 3. Servlet to perform addition of two number.

- Now let develop one more servlet that handles an HTTP Get request.
- The servlet is invoked when a form on webpage is submitted. Again here also we have two files one html file and other is java source file.

```
<html>
<body>
<Form name = "Form1"
      ACTION = "AddServlet"
      METHOD = POST>
<INPUT TYPE = TEXT NAME =first>
<INPUT TYPE = TEXT NAME =second>
<INPUT TYPE = SUBMIT VALUE = "SUBMIT">
</FORM>
</BODY>
</HTML>
```

- The java source code that handles get request is as follows:

```
import java.servlet.*;
import java.servlet.http.*;
import java.io.*;
public class AddServlet extends HttpServlet
{
    public void doGet(HttpServletRequest req,
        HttpServletResponse res)throws
        ServletException, IOException
    {
        string F1 = res.getParameter("first");
        int num=Integer.parseInt (F1);
        string F2 = res.getParameter("second");
        int num1=Integer.parseInt (F2);
```

```
res.setContentType("text/html");
PrintWriter out = res.getWriter();
out.println("The addition is "+(num1+num));
out.close()
}
}
```

## 5.4 SESSION MANAGEMENT

- Session management is a mechanism used by the Web container to store session information for a particular user.
- There are four different techniques used by Servlet application for session management. They are Cookies, Hidden field, URL Rewriting and Session Object.

### 5.4.1 Concept of Session

- Session simply means a particular interval of time.
- Session is used to store everything that we can get from the client from all the requests the client makes.
- Fig. 5.12 shows how sessions work.

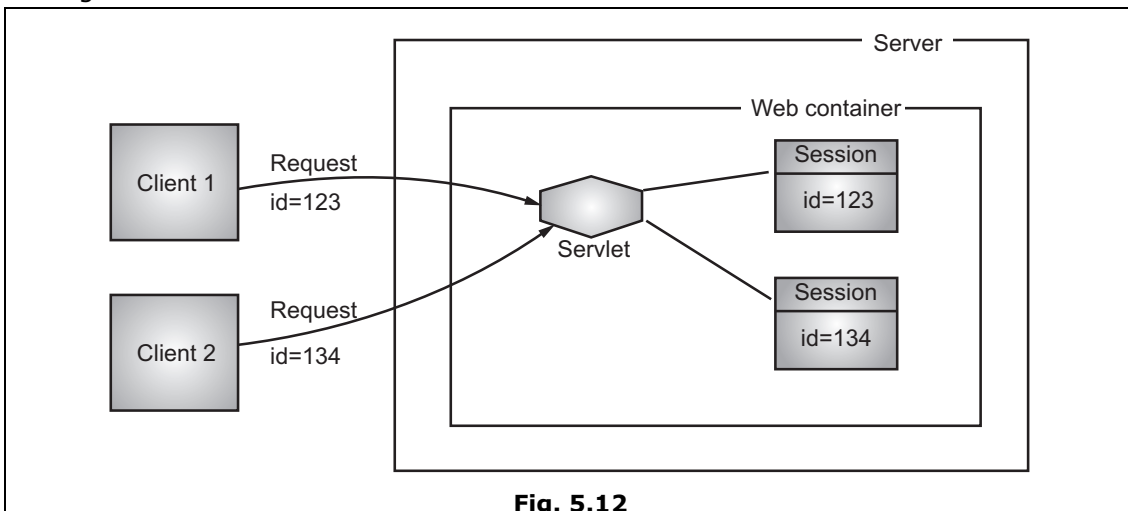
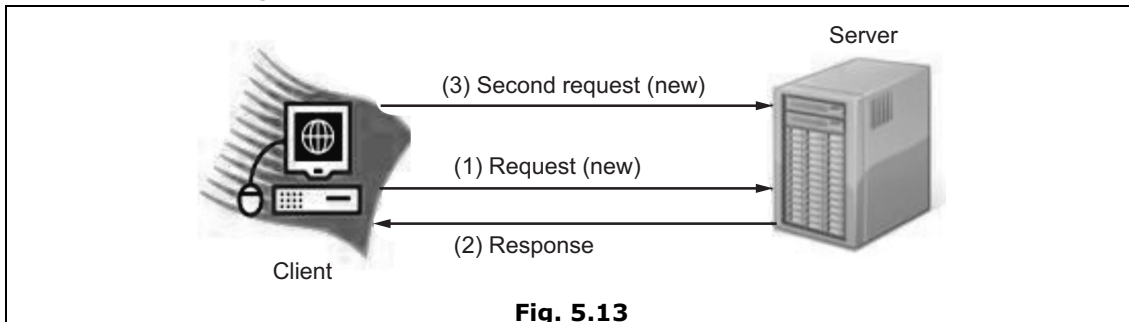


Fig. 5.12

### 5.4.2 Session Tracking

- Session tracking is a way to maintain state (data) of an user.
- It is also known as session management in servlet.
- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

- HTTP is stateless that means each request is considered as the new request. It is shown in the Fig. 5.13.

**Fig. 5.13**

### 5.4.3 What is Session Management?

- The Login servlet is used to require users to enter a valid user name and password before they can see some information.
- When a user first requests the servlet, the Login form is displayed. The user then can enter a user name and password and submit the form.
- Assuming that the form uses the POST method, the user information is captured in the doPost method, which does the authentication by calling the login method. If the login was successful, the information is displayed. If not, the Login form is sent again.
- What if you have another servlet that also only allows authorized users to view the information?
- This second servlet does not know whether, the same user has successfully logged in to the first servlet. Consequently, the user will be required to log in again.
- This is, of course, not practical. Every time a user goes to request a protected servlet, he or she has to login again even though all the servlets are part of the same application.
- This easily pushes the user to the edge of frustration and most likely results in a lost customer. Fortunately, there are ways to get around this, using techniques for remembering a user's session.
- Once, users have logged in they do not have to login again. The application will remember them. This is called session management.
- Session management, also called session tracking, goes beyond simply remembering a user who has successfully logged in.
- Anything that makes the application remember information that has been entered or requested by the user can be considered session management.
- Session management does not change the nature of HTTP statelessness, it simply provides a way around it.

- By principle, you manage a user's session by performing the following to servlets pages that need to remember a user's state:
  1. When the user requests a servlet, in addition to sending the response, the servlet also sends a token or an identifier.
  2. If the user does not come back with the next request for the same or a different servlet, that is fine. If the user does come back, the token or identifier is sent back to the server. Upon encountering the token, the next servlet should recognize the identifier and can do a certain action based on the token. When the servlet responds to the request, it also sends the same or a different token. This goes on and on with all the servlets that need to remember a user's session.
- You will use four techniques for session management. They operate based on the same principle, although what is passed and how it is passed is different from one to another.
- The techniques areas follows:
  1. URL rewriting,
  2. Hidden fields,
  3. Cookies, and
  4. Session objects.
- Which technique you use depends on what you need to do in your application.
- Each of the techniques is discussed in the sections below.

#### **5.4.3.1 URL Rewriting**

- With URL rewriting, you append a token or identifier to the URL of the next servlet or the next resource.
- You can send parameter name/value pairs using the following format:

```
url?name1=value1&name2=value2&...
```
- A name and a value is separated using an equal sign (=) a parameter name/value pair is separated from another parameter name/value pair using the ampersand (&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server.
- From a servlet, you can use the `HttpServletRequest` interface's `getParameter` method to obtain a parameter value.
- For instance, to obtain the value of the second parameter, you write the following:

```
request.getParameter(name2);
```

- The use of URL rewriting is easy and simple. When using this technique, however, you need to consider several things:
  1. The number of characters that can be passed in a URL is limited. Typically, a browser can pass up to 2,000 characters.
  2. The value that you pass can be seen in the URL. Sometimes, this is not desirable. For example, some people prefer their password not to appear on the URL.
  3. You need to encode certain characters, such as & and ? characters and white spaces, that you append to a URL.

#### URL Rewriting for Session Management:

- If the client has disabled cookie in the browser then session management using cookie wont work. In that case URL rewriting can be used as a backup.
- In URL rewriting, a token (parameter) is added at the end of the URL. The token consist of name/value pair seperated by an equal (=) sign.
- Fig. 5.14 shows an example of URL rewriting.

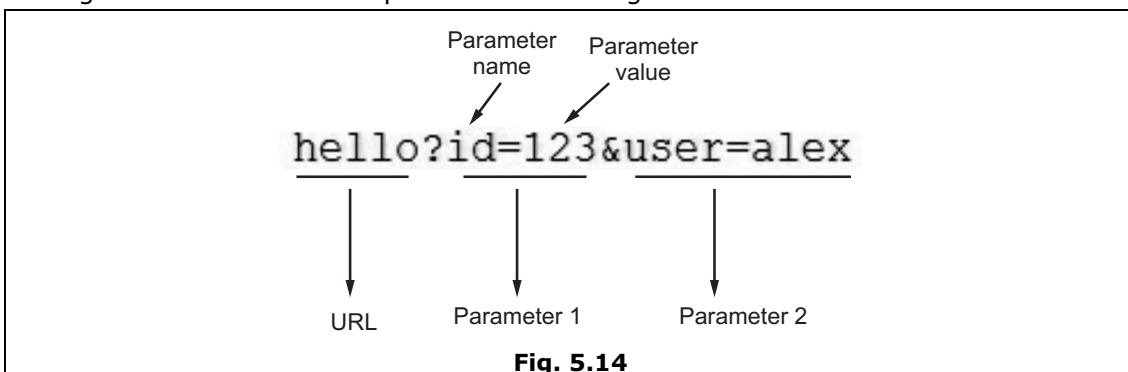


Fig. 5.14

- When the user clicks the URL links having parameters, the request goes to the Web Container with extra bit of information at the end of URL. The Web Container will fetch the extra part of the requested URL and uses it for session management.
- The `getParameter()` method is used to get the parameter value.

#### 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.

#### 5.4.3.2 Hidden Fields

- Another technique for managing user sessions is by passing a token as the value for an HTML hidden field.

- Unlike the URL rewriting, the value does not show on the URL but can still be read by viewing the HTML source code.
- Although this method also is easy and simple to use, an HTML form is always required.

### Hidden Form Fields:

- Hidden form field can also be used to store session information for a particular client. In case of hidden form field a hidden field is used to store client state. In this case user information is stored in hidden field value and retrieve from another servlet.
- In short, in Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.
- Let's see the code to store value in hidden field.

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

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

- In Fig. 5.15 we are storing the name of the user in a hidden textfield and getting that value from another servlet.

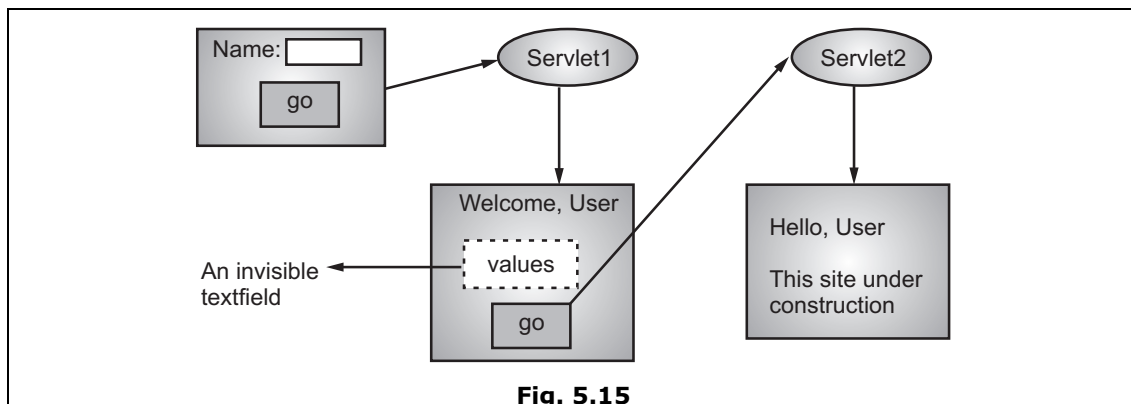


Fig. 5.15

### Advantage of Hidden Form Field:

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

### Disadvantages 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.

### 5.4.3.3 Cookies

- The third technique that you can use to manage user sessions is by using cookies.

- A cookie is a small piece of information that is passed back and forth in the HTTP request and response.
- Eventhough a cookie can be created on the client side using some scripting language such as JavaScript, it is usually created by a server resource, such as a servlet.
- The cookies sent by a servlet to the client will be passed back to the server when the client requests another page from the same application.

### 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.
- Fig. 5.16 shows working of cookies.

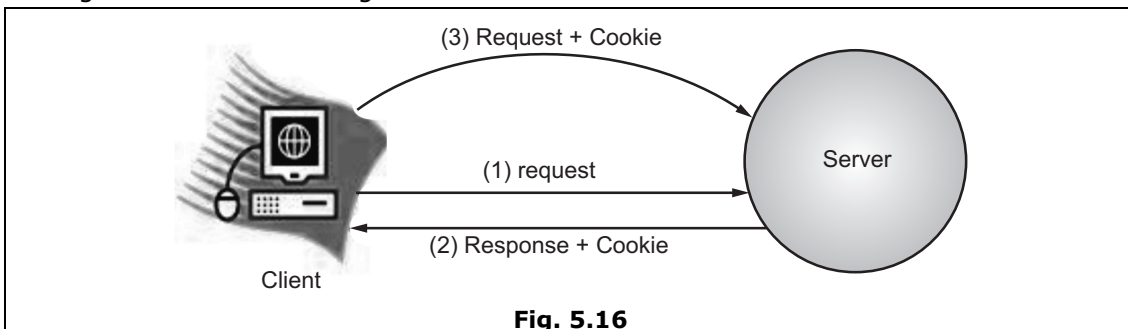


Fig. 5.16

- Types of cookies in servlets:
  - 1. Non-persistent Cookie:** It is valid for single session only. It is removed each time when user closes the browser.
  - 2. 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.
- In servlet programming, a cookie is represented by the Cookie class in the javax.servlet.http package.
- You can create a cookie by calling the Cookie class constructor and passing two String objects i.e., the name and value of the cookie.
- For instance, the following code creates a cookie object called c1.
- The cookie has the name "myCookie" and a value of "secret":

```
Cookie c1 = new Cookie("myCookie", "secret");
```

- You then can add the cookie to the HTTP response using the addCookie method of the HttpServletResponse interface:

```
response.addCookie(c1);
```

- The following example shows how you can create two cookies called `userName` and `password` and illustrates how those cookies are transferred back to the server. The servlet is called `CookieServlet` and its code is given below.
- When it is first invoked, the `doGet` method of the servlet is called. The method creates two cookies and adds both to the `HttpServletResponse` object, as follows:

```
Cookie c1 = new Cookie("userName", "tanmay");
Cookie c2 = new Cookie("password", "rashmi");
response.addCookie(c1);
response.addCookie(c2);
```

- Next, the `doGet` method sends an HTML form that the user can click to send another request to the servlet:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>Cookie Test</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("Please click the button to see the cookies sent to
                                                    you.");

out.println("<BR>");
out.println("<FORM METHOD=POST>");
out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
out.println("</FORM>");
out.println("</BODY>");
out.println("</HTML>");
```

- The form does not have any element other than a submit button. When the form is submitted, the `doPost` method is invoked.
- To retrieve cookies, you use the `getCookies` method of the `HttpServletRequest` interface. This method returns a `Cookie` array containing all cookies in the request. It is your responsibility to loop through the array to get the cookie you want, as follows:

```
Cookie[] cookies = request.getCookies();
int length = cookies.length;
for (int i=0; i<length; i++)
{
    Cookie cookie = cookies[i];
    out.println("<B>Cookie Name:</B> " + cookie.getName() + "<BR>");
    out.println("<B>Cookie Value:</B> " + cookie.getValue() + "<BR>");
}
```

**Program 5.9:** Sending and Receiving Cookies.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class CookieServlet extends HttpServlet {
    /*Process the HTTP Get request*/
    public void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException
    {
        Cookie c1 = new Cookie("userName", "tanmay");
        Cookie c2 = new Cookie("password", "rashmi");
        response.addCookie(c1);
        response.addCookie(c2);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Cookie Test</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Please click the button to see the cookies sent to
            you.");

        out.println("<BR>");
        out.println("<FORM METHOD=POST>");
        out.println("<INPUT TYPE=SUBMIT VALUE=Submit>");
        out.println("</FORM>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
    /**Process the HTTP Post request*/
    public void doPost(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Cookie Test</TITLE>");
```

```
out.println("</HEAD>");
out.println("<BODY>");
out.println("<BR><BR><H2> here are all the cookies.</H2>");
Cookie[] cookies = request.getCookies();
int length = cookies.length;
for (int i=0; i<length; i++) {
    Cookie cookie = cookies[i];
    out.println("<B>Cookie Name:</B> " + cookie.getName() + "<BR>");
    out.println("<B>Cookie Value:</B> " + cookie.getValue() +
        "<BR>");
}
out.println("</BODY>");
out.println("</HTML>");
}
}
```

---

**Advantages of Cookies:**

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

**Disadvantages of Cookies:**

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

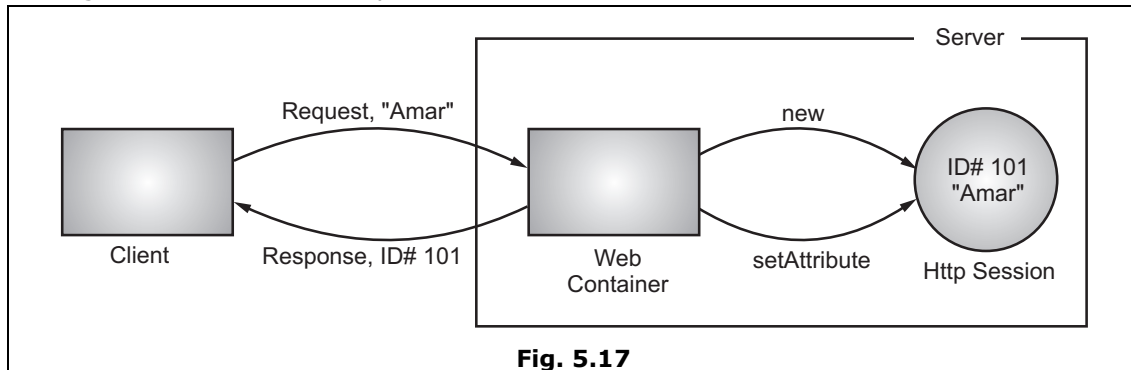
**Persisting Cookies:**

- The cookies you created in the previous last as long as the browser is open. When the browser is closed, the cookies are deleted.
- You can choose to persist cookies so that they last longer.
- The `javax.servlet.http.Cookie` class has the `setMaxAge` method that sets the maximum age of the cookie in seconds.

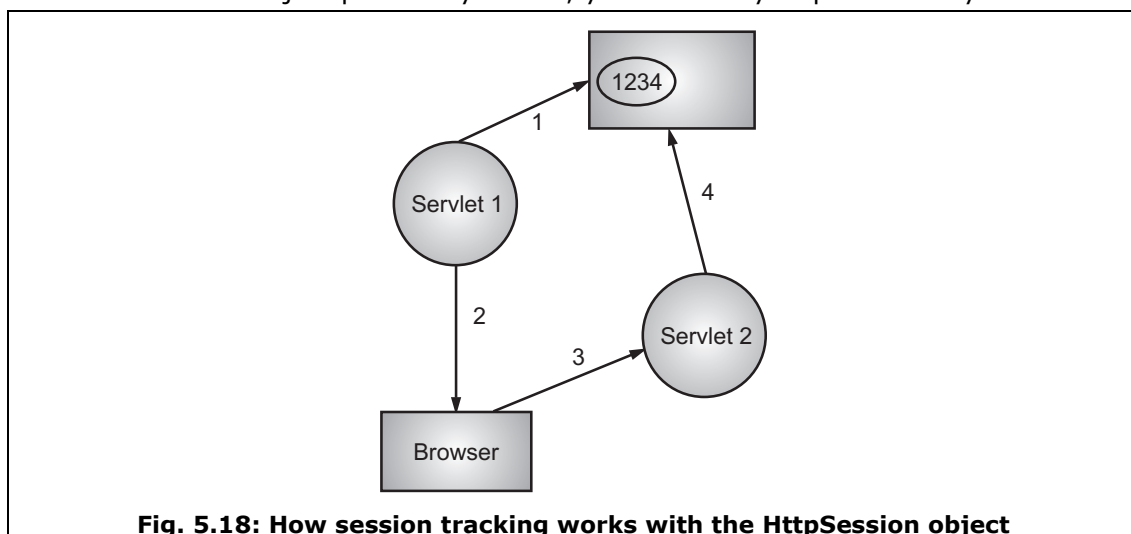
**5.4.3.4 Session Objects**

- The Session object, represented by the `javax.servlet.http.HttpSession` interface, is the easiest to use and the most powerful.
- For each user, the servlet can create an `HttpSession` object that is associated with that user only and can only be accessed by that particular user.
- The `HttpSession` object acts like a Hash table into which you can store any number of key/object pairs.
- `HttpSession` object is used to store entire session with a specific client. We can store, retrieve and remove attribute from `HttpSession` object.
- Any servlet can have access to `HttpSession` object through the `getSession()` method of the `HttpServletRequest` object.

- Fig. 5.17 shows how HttpSession works?



- On client's first request, the Web Container generates a unique session ID and gives it back to the client with response.
- The client sends back the session ID with each request.
- The Web Container uses this ID, finds the matching session with the ID and associates the session with the request.
- The HttpSession object is accessible from other servlets in the same application. To retrieve an object previously stored, you need only to pass the key.



- An HttpSession object uses a cookie or URL rewriting to send a token to the client. If cookies are used to convey session identifiers, the client browsers are required to accept cookies.
- Unlike previous techniques, however the server does not send any value. What it sends is simply a unique number called the session identifier.
- This session identifier is used to associate a user with a Session object in the server.

- Therefore, if there are 10 simultaneous users, 10 Session objects will be created in the server and each user can access only his or her own HttpSession object.
- The way an HttpSession object is created for a user and retrieved in the next requests is illustrated in Fig. 5.18.
- Fig. 5.18 shows that there are four steps in session tracking using the HttpSession object.
  1. An HttpSession object is created by a servlet called Servlet1. A session identifier is generated for this HttpSession object. In this example, the session identifier is 1234, but in reality, the servlet container will generate a longer random number that is guaranteed to be unique. The HttpSession object then is stored in the server and is associated with the generated session identifier. Also the programmer can store values immediately after creating an HttpSession.
  2. In the response, the servlet sends the session identifier to the client browser.
  3. When the client browser requests another resource in the same application, such as Servlet2, the session identifier is sent back to the server and passed to Servlet2 in the javax.servlet.http.HttpServletRequest object.
  4. For Servlet2 to have access to the HttpSession object for this particular client, it uses the getSession method of the javax.servlet.http.HttpServletRequest interface. This method automatically retrieves the session identifier from the request and obtains the HttpSession object associated with the session identifier.
- The getSession method of the javax.servlet.http.HttpServletRequest interface has two overloads. They are as follows:
  1. HttpSession getSession()
  2. HttpSession getSession(boolean create)
- The first method returns the current session associated with this request or if the request does not have a session identifier, it creates a new one.
- The second method returns the HttpSession object associated with this request if there is a valid session identifier in the request. If no valid session identifier is found in the request, whether a new HttpSession object is created depends on the create value.
- If the value is true, a new HttpSession object is created if no valid session identifier is found in the request. Otherwise, the getSession method will return null.
- Now that you know how session management works using the HttpSession object, let's have a look at the javax.servlet.http.HttpSession interface in more detail.

**javax.servlet.http.HttpSession Interface:**

- This interface has the following methods:
  - `getAttribute`
  - `getAttributeNames`
  - `getCreationTime`
  - `getId`
  - `getLastAccessedTime`
  - `getMaxInactiveInterval`
  - `getServletContext`
  - `getSessionContext`
  - `getValue`
  - `getValueNames`
  - `invalidate`
  - `isNew`
  - `putValue`
  - `removeAttribute`
  - `removeValue`
  - `setAttribute`
  - `setMaxInactiveInterval`
- Each of the methods is discussed below:
  1. **getAttribute:** This method retrieves an attribute from the `HttpSession` object. The return value is an object of type `Object`; therefore you may have to downcast the attribute to its original type. To retrieve an attribute, you pass the name associated. The signature/syntax for this method is as follows:

```
public Object getAttribute(String name) throws IllegalStateException
```
  2. **getAttributeNames:** The `getAttributeNames` method returns a `java.util.Enumeration` containing all attribute names in the `HttpSession` object. This method returns an `IllegalStateException` if it is called upon an invalidated `HttpSession` object. The syntax is as follows:

```
public java.util.Enumeration getAttributeNames() throws  
                                IllegalStateException
```
  3. **getCreationTime:** The `getCreationTime` method returns the time that the `HttpSession` object was created, in milliseconds since January 1, 1970 00:00:00 GMT. This method returns an `IllegalStateException` if it is called upon an invalidated `HttpSession` object. The syntax for this method is as follows:

```
public long getCreationTime() throws IllegalStateException
```
  4. **getId:** The `getID` method returns the session identifier. The syntax for this method is as follows:

```
public String getId()
```

5. **getLastAccessedTime:** The `getLastAccessedTime` method returns the time the `HttpSession` object was last accessed by the client. The return value is the number of milliseconds lapsed since January 1, 197000:00:00 GMT. The following is the method syntax:

```
public long getLastAccessedTime()
```

6. **removeAttribute:** The `removeAttribute` method removes an attribute bound to this `HttpSession` object. This method returns an `IllegalStateException` if it is called upon an invalidated `HttpSession` object. Its syntax is as follows:

```
public void removeAttribute(String name) throws
```

```
IllegalStateException
```

7. **setAttribute:** The `setAttribute` method adds a name/attribute pair to the `HttpSession` object. This method returns an `IllegalStateException` if it is called upon an invalidated `HttpSession` object. The method has the following syntax:

```
public void setAttribute(String name, Object attribute) throws
```

```
IllegalStateException
```

#### 5.4.4 Servlet Filters

- A filter is an object that can transform the header and content (or both) of a request or response.
- Filters differ from web components in that filters usually do not themselves create a response. Instead, a filter provides functionality that can be “attached” to any kind of web resource.
- Consequently, a filter should not have any dependencies on a web resource for which it is acting as a filter; this way, it can be composed with more than one type of web resource.
- The main tasks that a filter can perform are as follows:
  1. Query the request and act accordingly.
  2. Block the request-and-response pair from passing any further.
  3. Modify the request headers and data. You do this by providing a customized version of the request.
  4. Modify the response headers and data. You do this by providing a customized version of the response.
  5. Interact with external resources.

##### How Filters Works?

- Filters are components that you can use and configure to perform some filtering task.
- Filter is used for pre-processing of request and post-processing of response. You can have any number of filters for pre-processing of a request and post-processing of a response.
- Filters are configured in the deployment descriptor of a web application.

- Fig. 5.19 shows working of filters. The process as follows:
  - When a request reaches the Web Container, it checks if any filter has URL patterns that matches the requested URL.
  - The Web Container locates the first filter with a matching URL pattern and filter's code is executed.
  - If another filter has a matching URL pattern, its code is then executed. This continues until there are no filters with matching URL patterns.
  - If no errors occurs, the request passes to the target servlet.
  - The servlet returns the response back to its caller. The last filter that was applied to the request is the first filter applied to the response.
  - At last the response will be passed to the Web Container which passes it to the client.

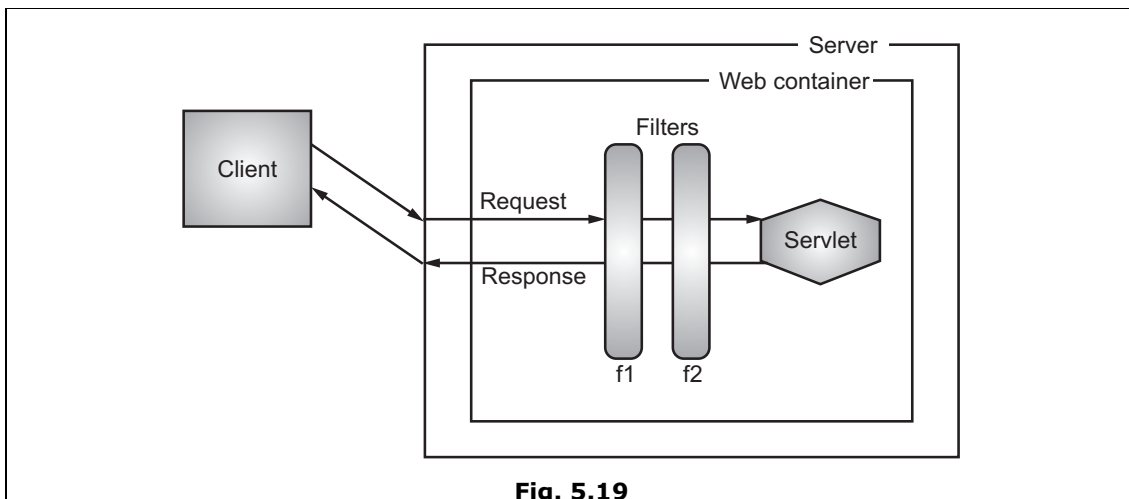


Fig. 5.19

### Servlet Filter Methods:

- A filter is simply a Java class that implements the `javax.servlet.Filter` interface.
- The `javax.servlet.Filter` interface defines three methods:

Sr. No.	Method and Description
1.	<code>public void doFilter (ServletRequest, ServletResponse, FilterChain):</code> This method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
2.	<code>public void init(FilterConfig filterConfig):</code> This method is called by the web container to indicate to a filter that it is being placed into service.
3.	<code>public void destroy():</code> This method is called by the web container to indicate to a filter that it is being taken out of service.

- Applications of filters include authentication, logging, image conversion, data compression, encryption, tokenizing streams, XML transformations and so on.

### 5.4.5 Servlet Chaining

- In servlet chaining, multiple servlets are called for a single client HTTP request, each servlet providing part of the HTML output.
- Each servlet receives the original client HTTP request as input, and each servlet produces its own output independently.
- Fig. 5.20 shows the servlet chaining process flow.

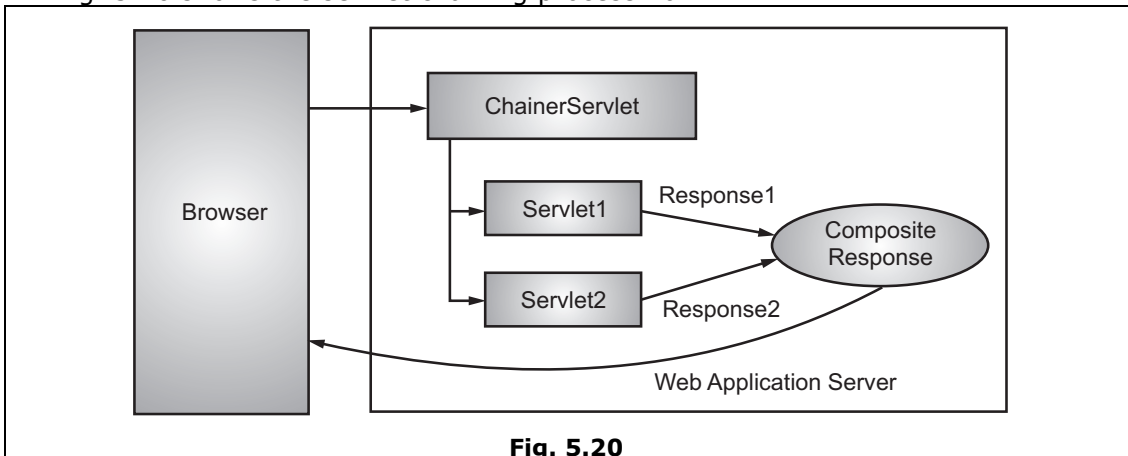


Fig. 5.20

- This servlet is specified on the original request, and multiple servlets are specified in an initialization parameter as the target:  
 Parameter name: `chainer.pathlist`  
 Parameter value: `/chainFirst /chainSecond`
- Each servlet is called in the order specified on the alias and the output HTML is made up of the output from all of the servlets.
- Servlet filtering and chaining have the advantage of allowing the Web developer to create modular servlets that can, for example, output standard HTML headers and footers or provide common dynamic content for pages.

### 5.4.6 Servlet Redirection

- Servlet redirection can be used to communicate between two servlet present in different servers. Some time the response of certain request may be present on two servers or given by two servlet.
- In such scenario the request from user goes to one server or one servlet then from that server it is forwarded to another server or servlet.
- The `sendRedirect()` method of `HttpServletResponse` interface can be used to redirect response to another resource.

- **Syntax of `sendRedirect()` method:**

```
public void sendRedirect(String URL) throws IOException;
```

- **Example:**

```
response.sendRedirect("http://www.google.com");
```

## 5.5 JSP

- JSP (Java Server Pages) is a standard for developing interactive Web applications (pages containing dynamic content).
- A JSP web page may display different content based on certain parameters (information stored in a database, the user preferences etc.), while a classic webpage (with the .htm or .html extension) will continuously display the same information.
- JSP is actually a powerful scripting language executed on the server side (like CGI, PHP, ASP) and not on the client side (Java applets which run in the browser of the user connected to a site).
- JSPs are integrated in a web page in HTML using special tags which will notify the Web server that the code included within these tags are to be interpreted. The result (HTML codes) will be returned to the client browser.
- JSP pages are opposite of Servlets. Servlet adds HTML code inside Java code while JSP adds Java code inside HTML. Everything a Servlet can do, a JSP page can also do it.
- JSP enables us to write HTML pages containing tags that run powerful Java programs. JSP separates presentation and business logic as Web designer can design and update JSP pages without learning the Java language and Java Developer can also write code without concerning the web design.
- JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, JSTL etc.
- 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. It provides some additional features such as Expression Language, Custom Tag etc.

### Why Use JSP?

1. Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
2. JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
3. JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.
4. JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

### 5.5.1 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 and simple.
  - 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.

#### Comparison between JSP and Servlet:

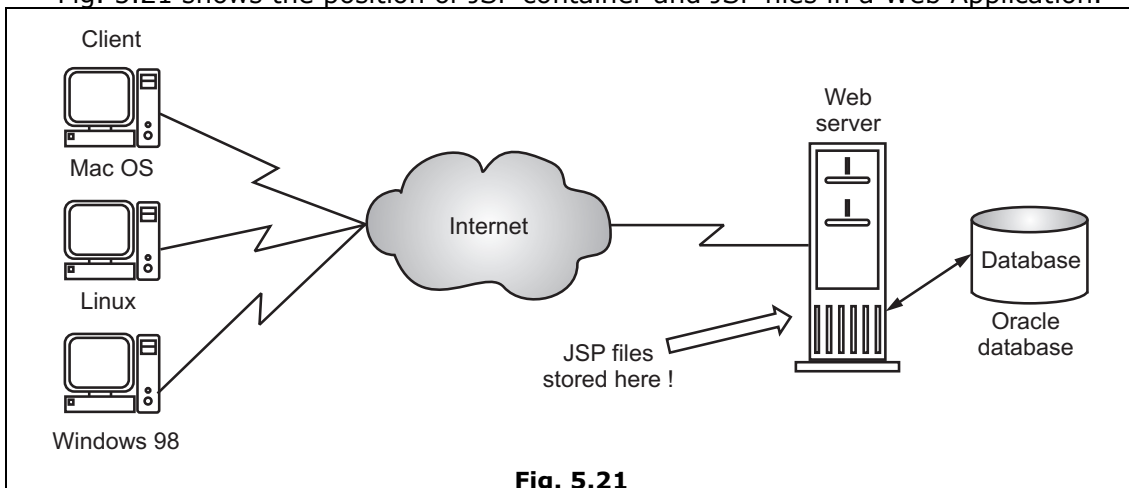
Terms	JSP	Servlet
What are they?	JSP is a webpage scripting language, generally used to create the dynamic web content.	Servlets are Java programs that are already compiled and which also create dynamic web content.
Typically	JSP is typically more oriented towards displaying information.	Servlet is more oriented towards processing information.
Role in MVC (Model View Controller)	JSP acts as a viewer.	Servlet acts as a controller.
Applicable at the time of	They are generally preferred when there is not much processing of data required.	They are generally preferred when there is more processing and manipulation involved.
Running speed	JSP runs slower as compared to a Servlet. JSP compiles into Java Servlets.	Servlets run faster as compared to JSP.

*contd. ...*

Code complications	The code programming is easy as compared to that of Servlets.	The code programming is difficult as compared to that of JSP.
Facility	Here, we can build custom tags which can directly call Java beans.	No such facility is available in servlets.
Consists of	JSP are Java HTML representation mixed with JAVA scriptlets.	Servlet are full functional Java codes.
Consistence of objects	JSP has Implicit objects.	Servlets does not have such type of objects.
Examples	To display a report.	To process a user submitted form.

### 5.5.2 JSP Architecture

- The web server needs a JSP engine i.e., container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages.
- A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.
- Fig. 5.21 shows the position of JSP container and JSP files in a Web Application.

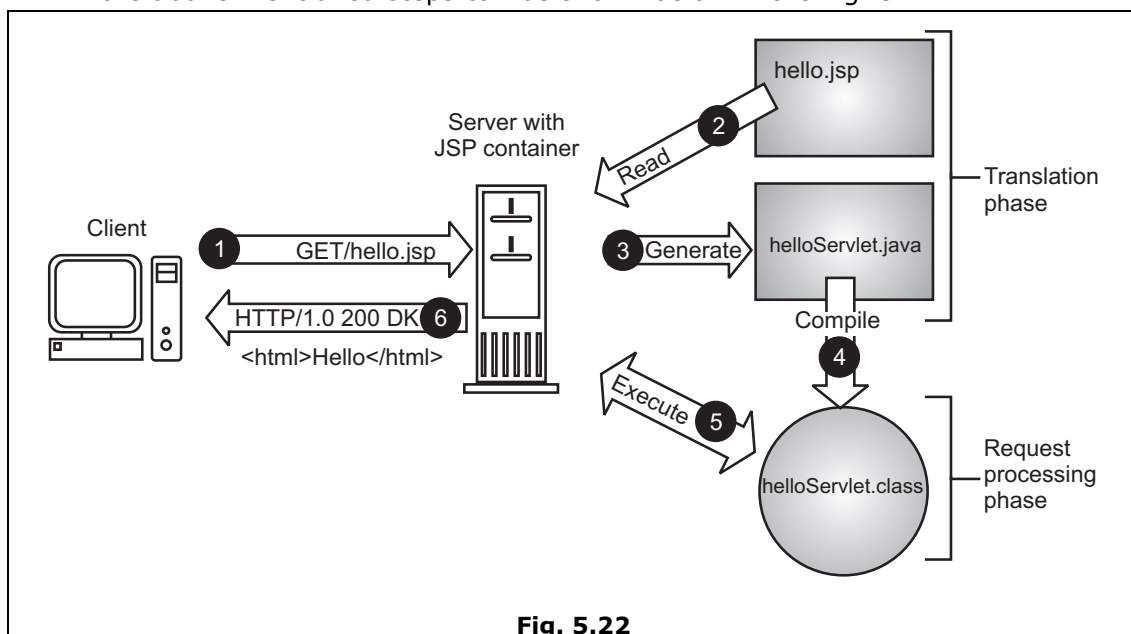


**Fig. 5.21**

#### JSP Processing:

- The following steps explain how the web server creates the web page using JSP:
  - As with a normal page, your browser sends an HTTP request to the web server.
  - The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.

- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to `println( )` statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.
  - The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
  - A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.
  - The web server forwards the HTTP response to your browser in terms of static HTML content.
  - Finally, web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page.
- All the above mentioned steps can be shown below in the Fig. 5.22.

**Fig. 5.22**

- Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet.
- If the JSP is older than its generated servlet, the JSP container assumes that the JSP has not changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

### 5.5.3 Scriptlet

- A scriptlet can contain any number of java language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- JSP Scriptlets begins with `<%` and ends `%>`.
- We can embed any amount of java code in the JSP Scriptlets.

- **Syntax for Scriptlet:**

```
<% // any java source code here %>
```

- A scriptlet is a fragment of Java code that is run when the user requests the page.
- For example, any Java if/for/while blocks opened in one scriptlet element must be correctly closed in a later element for the page to successfully compile. Markup which falls inside a split block of code is subject to that code, so markup inside an if block will only appear in the output when the if condition evaluates to true; likewise, markup inside a loop construct may appear multiple times in the output depending upon how many times the loop body runs.
- The following would be a valid for loop in a JSP page:

```
<p>Counting to three:</p>
<% for(int i=1; i<4; i++){ %>
<p>This number is <%= i %> </p>
<% } %>
<p>OK</p>
```

- The output displayed in the user's web browser would be:

```
Counting to three:
This number is 1.
This number is 2.
This number is 3.
OK.
```

### 5.5.4 Expression

- Expression is used to insert values directly to the output.
- An expression tag places an expression to be evaluated inside the java servlet class.
- A JSP expression element contains a scripting language expression that is evaluated, converted to a String and inserted where the expression appears in the JSP file.
- Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

- The expression element can contain any expression that is valid according to the Java language specification but you cannot use a semicolon to end an expression.
- Following is the syntax of JSP Expression:

```
<%= expression %>
```

- You can write XML equivalent of the above syntax as follows:

```
<jsp:expression>
    expression
</jsp:expression>
```

- Following is the simple example for JSP Expression:

```
<html>
<head><title>A Comment Test</title></head>
<body>
<p>
Today's date: <%= (new java.util.Date()).toLocaleString() %>
</p>
</body>
</html>
```

- This would generate following result:

```
Today's date: 11-nov-2014 21:24:25
```

### **5.5.5 Declarations**

- A declaration declares one or more variables or methods that you can use in Java code later in the JSP file.
- You must declare the variable or method before you use it in the JSP file.
- Following is the syntax of JSP Declarations:

```
<%! declaration; [ declaration; ]+... %>
```

- You can write XML equivalent of the above syntax as follows:

```
<jsp:declaration>
    code fragment
</jsp:declaration>
```

- Following is the simple example for JSP Declarations:

```
<%!int i =0; %>
<%!int a, b, c; %>
<%!Circle a =newCircle(2.0); %>
```

### 5.5.6 Actions

- JSP actions use constructs in XML syntax to control the behavior of the servlet engine.
- You can dynamically insert a file, reuse JavaBeans components, forward the user to another page or generate HTML for the Java plugin.
- There is only one syntax for the Action element, as it conforms to the XML standard:

```
<jsp:action_nameattribute="value"/>
```

- Action elements are basically predefined functions and there are following JSP actions available:

Syntax	Purpose
<code>jsp:include</code>	Includes a file at the time the page is requested.
<code>jsp:useBean</code>	Finds or instantiates a JavaBean.
<code>jsp:setProperty</code>	Sets the property of a JavaBean.
<code>jsp:getProperty</code>	Inserts the property of a JavaBean into the output.
<code>jsp:forward</code>	Forwards the requester to a new page.
<code>jsp:plugin</code>	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
<code>jsp:element</code>	Defines XML elements dynamically.
<code>jsp:attribute</code>	Defines dynamically defined XML element's attribute.
<code>jsp:body</code>	Defines dynamically defined XML element's body.
<code>jsp:text</code>	Use to write template text in JSP pages and documents.

### 5.5.7 Implicit Objects

- These objects are available for the JSP developer and they can use the objects in the JSP files without declaring the objects in the JSP.
- JSP container provides a list of instantiated objects for you to access different kind of data in a web application.
- These objects are called implicit objects as they are automatically available for the implement.

- JSP supports nine automatically defined variables, which are also called implicit objects. These variables are:

Objects	Description
request	This is the HttpServletRequest object associated with the request.
response	This is the HttpServletResponse object associated with the response to the client.
out	This is the PrintWriter object used to send output to the client.
session	This is the HttpSession object associated with the request.
application	This is the ServletContext object associated with application context.
config	This is the ServletConfig object associated with the page.
pageContext	This encapsulates use of server-specific features like higher performance JspWriters.
page	This is simply a synonym for this, and is used to call the methods defined by the translated servlet class.
Exception	The Exception object allows the exception data to be accessed by designated JSP.

### 5.5.8 Directives

- JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing.
- JSP directive tag gives special information about the page to the JSP engine.
- A JSP directive affects the overall structure of the servlet class. It usually has the following form/syntax:

```
<%@ directive attribute="value" %>
```

- Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.
- The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.
- There are three types of directive tag:

Directive	Description
<%@ page ... %>	Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
<%@ include ... %>	Includes a file during the translation phase.
<%@ taglib ... %>	Declares a tag library, containing custom actions, used in the page

**1. Let see above directives in detail:**

- The page directive is used to provide instructions to the container that pertain to the current JSP page.
- You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.
- Following is the basic syntax of page directive:

```
<%@ page attribute="value" %>
```

- You can write XML equivalent of the above syntax as follows:

```
<jsp:directive.pageattribute="value"/>
```

**Attributes:** Following is the list of attributes associated with page directive:

Attribute	Purpose
buffer	Specifies a buffering model for the output stream.
autoFlush	Controls the behavior of the servlet output buffer.
contentType	Defines the character encoding scheme.
errorPage	Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
isErrorPage	Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
extends	Specifies a superclass that the generated servlet must extend.
import	Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
info	Defines a string that can be accessed with the servlet's <code>getServletInfo()</code> method.
isThreadSafe	Defines the threading model for the generated servlet.
language	Defines the programming language used in the JSP page.
session	Specifies whether or not the JSP page participates in HTTP sessions.
isELIgnored	Specifies whether or not EL expression within the JSP page will be ignored.
isScriptingEnabled	Determines if scripting elements are allowed for use.

**2. include Directive:**

- The include directive is used to includes a file during the translation phase.
- This directive tells the container to merge the content of other external files with the current JSP during the translation phase.
- You may code include directives anywhere in your JSP page.

- The general usage form/syntax of this directive is as follows:

```
<%@ include file="relative url">
```

- The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.
- You can write XML equivalent of the above syntax as follows:

```
<jsp:directive.includefile="relative url"/>
```

### 3. taglib Directive:

- The JavaServer Pages API allows you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.
- The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides a means for identifying the custom tags in your JSP page.
- The taglib directive follows the following syntax:

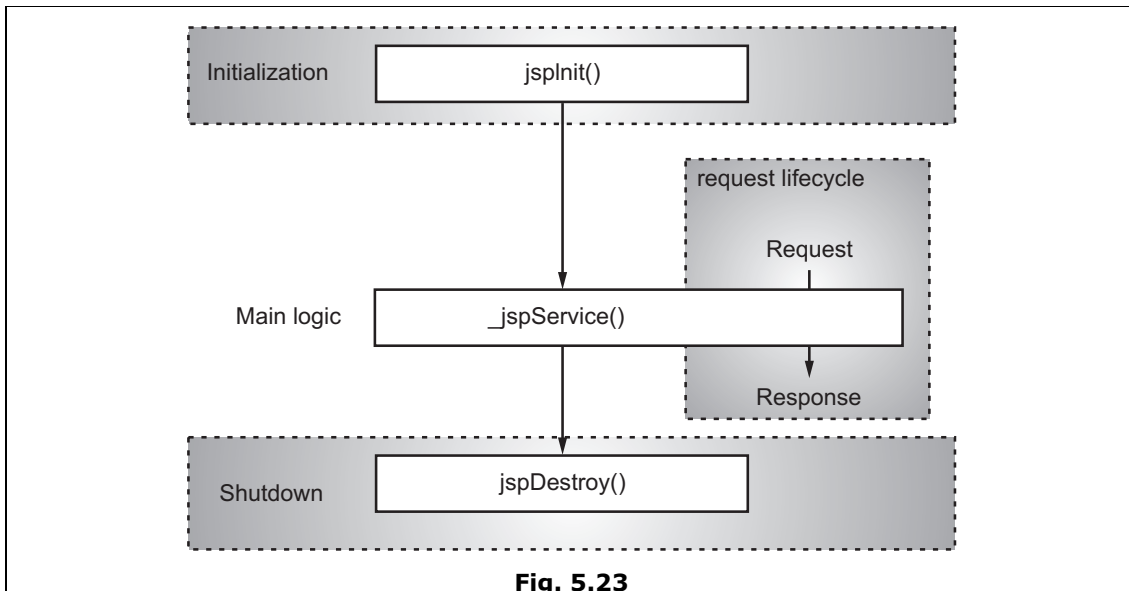
```
<%@ taglib uri="uri" prefix="prefixOfTag">
```

Where the uri attribute value resolves to a location the container understands and the prefix attribute informs a container what bits of markup are custom actions.

## **5.5.9 Life Cycle of a JSP Page**

---

- A JSP life cycle can be defined as "the entire process from its creation till the destruction which is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet".
- A JSP page services requests as a servlet. Thus, the life cycle and many of the capabilities of JSP pages (in particular the dynamic aspects) are determined by Java Servlet technology.
- The following are the paths followed by a JSP:
  1. Compilation,
  2. Initialization,
  3. Execution, and
  4. Cleanup.
- The four major phases of JSP life cycle are very similar to Servlet Life Cycle as shown in Fig. 5.23.

**Fig. 5.23**

- Fig. 5.23 shows following parts of a JSP page:

### 1. JSP Compilation:

- When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page.
- If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
- The compilation process involves three steps:
  - (i) Parsing the JSP.
  - (ii) Turning the JSP into a servlet.
  - (iii) Compiling the servlet.

### 2. JSP Initialization:

- When a container loads a JSP it invokes the `jspInit()` method before servicing any requests.
- If you need to perform JSP-specific initialization, override the `jspInit()` method:

```
public void jspInit(){  
    // Initialization code...  
}
```

- Typically initialization is performed only once and as with the servlet `init` method, you generally initialize database connections, open files and create lookup tables in the `jspInit()` method.

### 3. JSP Execution:

- This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

- Whenever, a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the `_jspService()` method in the JSP.
- The `_jspService()` method takes an `HttpServletRequest` and an `HttpServletResponse` as its parameters as follows:

```
void _jspService(HttpServletRequest request,
    HttpServletResponse response)
{
    // Service handling code...
}
```

- The `_jspService()` method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods i.e. GET, POST, DELETE etc.

#### 4. JSP Cleanup:

- The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.
- The `jspDestroy()` method is the JSP equivalent of the `destroy` method for servlets.
- Override `jspDestroy()` when you need to perform any cleanup, such as releasing database connections or closing open files.

### 5.5.10 JSP TLD

- A custom tag is a user-defined JSP language element.
- When a JSP page containing a custom tag is translated into a servlet, the tag is converted to operations on an object called a tag handler i.e., Tag Library Descriptor (TLD).
- A TLD is an XML document that contains information about a library as a whole and about each tag contained in the library.
- TLDs are used by a web container to validate the tags and by JSP page development tools.
- Tag library descriptor file names must have the extension `.tld`.
- The Web container then invokes those operations when the JSP page's servlet is executed.
- JSP tag extensions let you create new tags that you can insert directly into a JavaServer Page just as you would the built-in tags.
- The JSP 2.0 specification introduced Simple Tag Handlers for writing these custom tags.
- Custom tags are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page.
- The same business logic can be used many times by the use of custom tag.

**Advantages of Custom Tags:**

1. **Eliminates the need of scriplet tag:** The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
  2. **Separation of business logic from JSP:** The custom tags separate the the business logic from the JSP page so that it may be easy to maintain.
  3. **Reusability:** The custom tags makes the possibility to reuse the same business logic again and again.
- **Syntax** to use custom tag: There are two ways to use the custom tag. They are given below:

```
<prefix:tagname attr1=value1....attrn=valuen />
```

OR

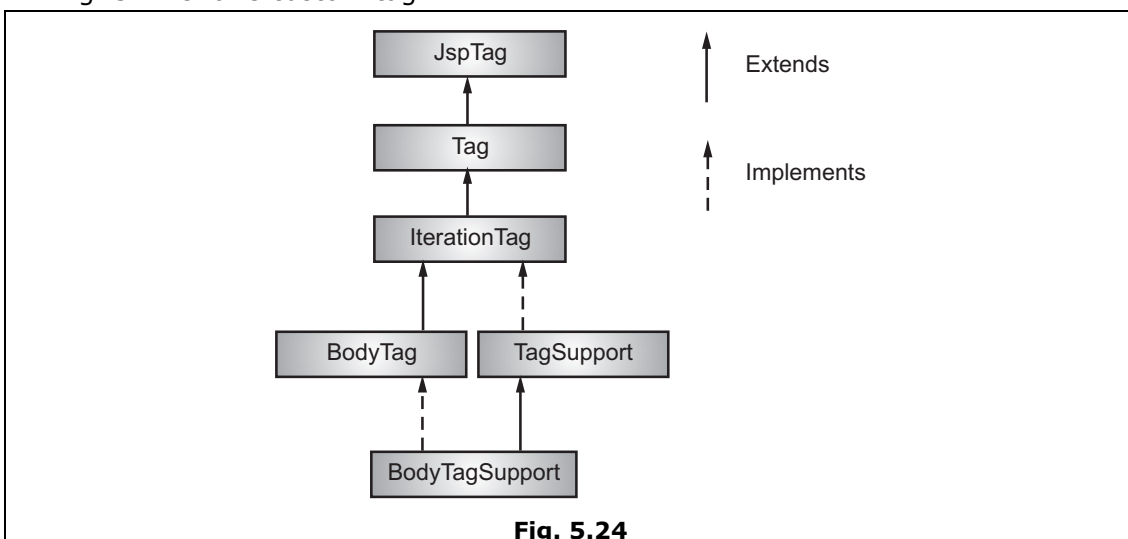
```
<prefix:tagname attr1=value1....attrn=valuen >
```

```
body code
```

```
</prefix:tagname>
```

**Custom Tag API:**

- The javax.servlet.jsp.tagext package contains classes and interfaces for JSP custom tag API. The JspTag is the root interface in the Custom Tag hierarchy.
- Fig. 5.24 shows custom tag API.



- To write a customer tab you can simply extend SimpleTagSupport class and override the doTag() method, where you can place your code to generate content for the tag.

**Create "Hello" Tag:**

- Consider you want to define a custom tag named <ex:Hello> and you want to use it in the following fashion without a body:

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

- To create a custom JSP tag, you must first create a Java class that acts as a tag handler. So let us create HelloTag class as follows:

```
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class HelloTag extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
        JspWriter out = getJspContext().getOut();
        out.println("Hello Custom Tag!");
    }
}
```

- Above code has simple coding where doTag() method takes the current JspContext object using getJspContext() method and uses it to send "Hello Custom Tag!" to the current JspWriter object.
- Let us compile above class and copy it in a directory available in environment variable CLASSPATH.
- Finally, create following tag library file: <Tomcat-Installation-Directory>webapps\ROOT\WEB-INF\custom.tld.

```
<taglib>
<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>Example TLD</short-name>
<tag>
<name>Hello</name>
<tag-class>HelloTag</tag-class>
<body-content>empty</body-content>
</tag>
</taglib>
```

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

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

- Try to call above JSP and this should produce following result:

```
HelloCustomTag!
```

### Accessing the Tag Body:

- You can include a message in the body of the tag as you have seen with standard tags.
- Consider you want to define a custom tag named `<ex:Hello>` and you want to use it in the following fashion with a body:

```
<ex:Hello>
```

- This is message body,

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

- Let us make following changes in above our tag code to process the body of the tag:

```
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class HelloTag extends SimpleTagSupport {
    StringWriter sw = new StringWriter();
    public void doTag()
        throws JspException, IOException
    {
        getJspBody().invoke(sw);
        getJspContext().getOut().println(sw.toString());
    }
}
```

- In this case, the output resulting from the invocation is first captured into a `StringWriter` before being written to the `JspWriter` associated with the tag.
- Now accordingly we need to change TLD file as follows:

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

- Now let us call above tag with proper body as follows:

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

- This is message body

```
</ex:Hello>
</body>
</html>
```

- This will produce following result:

```
This is message body
```

#### Custom Tag Attributes:

- You can use various attributes along with your custom tags.
- To accept an attribute value, a custom tag class needs to implement setter methods, identical to JavaBean setter methods as shown below:

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

public class HelloTag extends SimpleTagSupport {
    private String message;

    public void setMessage(String msg) {
        this.message = msg;
    }

    StringWriter sw = new StringWriter();

    public void doTag()
        throws JspException, IOException
    {
        if (message != null) {
            /* Use message from attribute */
            JspWriter out = getJspContext().getOut();
            out.println( message );
        }
    }
}
```

```
else{
    /* use message from the body */
        getJspBody().invoke(sw);
        getJspContext().getOut().println(sw.toString());
    }
}
}
```

- The attribute's name is "message", so the setter method is setMessage().
- Now let us add this attribute in TLD file using <attribute> element as follows:

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

- Now let us try following JSP with message attribute as follows:

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

- This will produce following result:
- This is custom tag.

### 5.5.11 JSTL

- The Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications.
- JSTL represents a set of tags to simplify the JSP development.
- JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

#### Advantage of JSTL:

- 1. Fast Development:** JSTL provides many tags that simplifies the JSP.
  - 2. Code Reusability:** We can use the JSTL tags in various pages.
  - 3. No need to use scriptlet tag:** It avoids the use of scriptlet tag.
- The JSTL tags can be classified, according to their functions, into following JSTL tag library groups that can be used when creating a JSP page:
    1. Core Tags,
    2. Formatting tags,
    3. SQL tags, and
    4. JSTL Functions.

#### Install JSTL Library:

- If you are using Apache Tomcat container then follow the following two simple steps:
  1. Download the binary distribution from Apache Standard Taglib and unpack the compressed file.
  2. To use the Standard Taglib from its Jakarta Taglibs distribution, simply copy the JAR files in the distribution's 'lib' directory to your application's webapps\ROOT\WEB-INF\lib directory.
- To use any of the libraries, you must include a <taglib> directive at the top of each JSP that uses the library.

#### 1. Core Tags:

- The core group of tags are the most frequently used JSTL tags.
- The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is <http://java.sun.com/jsp/jstl/core> . The prefix of core tag is c.
- Following is the syntax to include JSTL Core library in your JSP:

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

- There are following Core JSTL Tags:

Tag	Description
<c:out>	Like <%= ... >, but for expressions.
<c:set>	Sets the result of an expression evaluation in a 'scope'.
<c:remove>	Removes a scoped variable (from a particular scope, if specified).
<c:catch>	Catches any Throwable that occurs in its body and optionally exposes it.
<c:if>	Simple conditional tag which evaluates its body if the supplied condition is true.
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>.
<c:when>	Subtag of <choose> that includes its body if its condition evaluates to 'true'.
<c:otherwise>	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'.
<c:import>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
<c:forEach>	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality.
<c:forEachTokens>	Iterates over tokens, separated by the supplied delimiters.
<c:param>	Adds a parameter to a containing 'import' tag's URL.
<c:redirect>	Redirects to a new URL.
<c:url>	Creates a URL with optional query parameters.

## 2. Formatting Tags:

- The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Web sites.
- These tags provide support for message formatting, number and date formatting etc. The url for the internationalization tags is <http://java.sun.com/jsp/jstl/fmt> and prefix is fmt.
- Following is the syntax to include Formatting library in your JSP:

```
<%@ taglib prefix="fmt"
uri="http://java.sun.com/jsp/jstl/fmt" %>
```

- Following is the list of Formatting JSTL Tags:

Tag	Description
<fmt:formatNumber>	To render numerical value with specific precision or format.
<fmt:parseNumber>	Parses the string representation of a number, currency or percentage.
<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern.
<fmt:parseDate>	Parses the string representation of a date and/or time.
<fmt:bundle>	Loads a resource bundle to be used by its tag body.
<fmt:setLocale>	Stores the given locale in the locale configuration variable.
<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.
<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable.
<fmt:message>	To display an internationalized message.
<fmt:requestEncoding>	Sets the request character encoding.

### 3. SQL Tags:

- The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, MySQL or Microsoft SQL Server.
- The JSTL sql tags provide SQL support. The url for the sql tags is <http://java.sun.com/jsp/jstl/sql> and prefix is sql.
- Following is the syntax to include JSTL SQL library in your JSP:

```
<%@ taglib prefix="sql"
uri="http://java.sun.com/jsp/jstl/sql" %>
```

- Following is the list of SQL JSTL Tags:

Tag	Description
<code>&lt;sql:setDataSource&gt;</code>	Creates a simple DataSource suitable only for prototyping.
<code>&lt;sql:query&gt;</code>	Executes the SQL query defined in its body or through the sql attribute.
<code>&lt;sql:update&gt;</code>	Executes the SQL update defined in its body or through the sql attribute.
<code>&lt;sql:param&gt;</code>	Sets a parameter in an SQL statement to the specified value.
<code>&lt;sql:dateParam&gt;</code>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<code>&lt;sql:transaction &gt;</code>	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

#### 4. JSTL Functions:

- JSTL includes a number of standard functions, most of which are common string manipulation functions.
- The functions tags provide support for string manipulation and string length. The url for the functions tags is <http://java.sun.com/jsp/jstl/functions> and prefix is fn.
- Following is the syntax to include JSTL Functions library in your JSP:

```
<%@ taglib prefix="fn"
    uri="http://java.sun.com/jsp/jstl/functions" %>
```

- Following is the list of JSTL Functions:

Function	Description
<code>fn:contains()</code>	Tests if an input string contains the specified substring.
<code>fn:containsIgnoreCase()</code>	Tests if an input string contains the specified substring in a case insensitive way.
<code>fn:endsWith()</code>	Tests if an input string ends with the specified suffix.
<code>fn:escapeXml()</code>	Escapes characters that could be interpreted as XML markup.
<code>fn:indexOf()</code>	Returns the index within a string of the first occurrence of a specified substring.

**contd. ...**

<code>fn:join()</code>	Joins all elements of an array into a string.
<code>fn:length()</code>	Returns the number of items in a collection, or the number of characters in a string.
<code>fn:replace()</code>	Returns a string resulting from replacing in an input string all occurrences with a given string.
<code>fn:split()</code>	Splits a string into an array of substrings.
<code>fn:startsWith()</code>	Tests if an input string starts with the specified prefix.
<code>fn:substring()</code>	Returns a subset of a string.
<code>fn:substringAfter()</code>	Returns a subset of a string following a specific substring.
<code>fn:substringBefore()</code>	Returns a subset of a string before a specific substring.
<code>fn:toLowerCase()</code>	Converts all of the characters of a string to lower case.
<code>fn:toUpperCase()</code>	Converts all of the characters of a string to upper case.
<code>fn:trim()</code>	Removes white spaces from both ends of a string.

## 5.6 JAVA BEANS

- A Java Bean is a software component that has been designed to be reusable in a variety of different environments.
- In short JavaBeans are reusable software components for Java.
- They are classes that encapsulate many objects into a single object (the bean). They are serializable, have a 0-argument constructor and allow access to properties using getter and setter methods.
- It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio.
- A Bean may be visible to an end user. Software to generate a piechart from a set of data points is an example of a Bean that can execute locally.
- **Definition:** A Java Bean is "a reusable software component that can be visually manipulated in builder tools".

### Why Use Java Bean?

1. It is a reusable software component.
2. A bean encapsulates many objects into one object, so we can access this object from multiple places.
3. It provides the easy maintenance.

**Characteristics of Bean:**

- Following are the unique characteristics that distinguish a JavaBean from other Java classes:
  1. It provides a default, no-argument constructor.
  2. It should be serializable and implement the Serializable interface.
  3. It may have a number of properties which can be read or written.
  4. It may have a number of "getter" and "setter" methods for the properties.

**Advantages of Beans:**

1. A Bean obtains all the benefits of Java's "Write-Once, Run-Anywhere" paradigm.
2. The properties, events, and methods of a Bean that are exposed to an application builder tool can be controlled.
3. A Bean may be designed to operate correctly in different locales, which makes it useful in global markets.
4. Auxiliary software can be provided to help a person configure a Bean. This software is only needed when the design-time parameters for that component are being set. It does not need to be included in the run-time environment.
5. The configuration settings of a Bean can be saved in persistent storage and restored at a later time.
6. A Bean may register to receive events from other objects and can generate events that are sent to other objects.

**Introspection:**

- Introspection is the process of analyzing a Bean to determine its capabilities. This is an essential feature of the Java Beans API, because it allows an application builder tool to present information about a component to a software designer.
- Without introspection, the Java Beans technology could not operate.
- Introspection is the automatic process of analyzing a bean's design patterns to reveal the bean's properties, events, and methods. This process controls the publishing and discovery of bean operations and properties.
- There are two ways in which the developer of a Bean can indicate which of its properties, events, and methods should be exposed by an application builder tool.
  1. With the first method, simple naming conventions are used. These allow the introspection mechanisms to infer information about a Bean.
  2. In the second way, an additional class is provided that explicitly supplies this information. The first approach is examined here.
- The following topics indicate the design patterns for properties and events that enable the functionality of a Bean to be determined.

**Design Patterns for Properties:**

- A property is a subset of a Bean's state. The values assigned to the properties determine the behavior and appearance of that component.
- Properties are a Bean's appearance and behavior attributes that can be changed at design time.
- This point discusses three types of properties: simple, Boolean, and indexed.

**1. Simple Properties of Beans:**

- A simple property has a single value. It can be identified by the following design patterns, where N is the name of the property and T is its type.

```
public T getN( );  
public void setN(T arg);
```

- A read/write property has both of these methods to access its values. A read-only property has only a get method. A write-only property has only a set method.

**2. Boolean Properties:**

- A Boolean property has a value of true or false.
- It can be identified by the following design patterns, where N is the name of the property:

```
public boolean isN( );  
public boolean getN( );  
public void setN(boolean value);
```

- Either the first or second pattern can be used to retrieve the value of a Boolean property. However, if a class has both of these methods, the first pattern is used.

**3. Indexed Properties:**

- An indexed property consists of multiple values.
- It can be identified by the following design patterns, where N is the name of the property and T is its type:

```
public T getN(int index);  
public void setN(int index, T value);  
public T[ ] getN( );  
public void setN(T values[ ]);
```

**JavaBeans Example:**

```
public class StudentsBean implements java.io.Serializable  
{  
    private String firstName = null;  
    private String lastName = null;  
    private int age = 0;  
    public StudentsBean() {  
    }  
}
```

```
public String getFirstName(){
    return firstName;
}
public String getLastName(){
    return lastName;
}
public int getAge(){
    return age;
}
public void setFirstName(String firstName){
    this.firstName = firstName;
}
public void setLastName(String lastName){
    this.lastName = lastName;
}
public void setAge(Integer age){
    this.age = age;
}
}
```

- The useBean action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP.
- The full syntax for the useBean tag is as follows:

```
<jsp:useBean id="bean's name" scope="bean's scope" typeSpec/>
```

- Following example shows its simple usage:

```
<html>
<head>
<title>useBean Example</title>
</head>
<body>

<jsp:useBean id="date" class="java.util.Date" />
<p>The date/time is <%= date %>

</body>
</html>
```

- This would produce following result:

```
The date/time is Thu Sep 30 11:18:11 GST 2010
```

**Important Points**

- CGI programs provided a simple way to create web applications that accepts user input, queries a database and returns relevant result back to the web browser.
- HTTP is the protocol that allows web servers and browsers to exchange data over the web.
- An HTTP transaction begins with a request from the client browser and ends with a response from the server.
- A well-designed software application is partitioned into separate logical parts called layers.
- A java servlet is a server side program that services HTTP requests and returns the result as HTTP responses.
- This servlet-aware web server is called a servlet container, which also was called a servlet engine in the early days of the servlet technology.
- A servlet is loaded by the servlet container the first time the servlet is requested.
- The servlet then is forwarded the user request, processes it and returns the response to the servlet container, which in turn sends the response back to the user.
- The init method is called by the servlet container after the servlet class has been instantiated.
- The service method is called by the servlet container after the servlet's init method to allow the servlet to respond to a request.
- The servlet container calls the destroy method before removing a servlet instance from service.
- A web application is a collection of servlets and other content installed under a specific subset of the server's URL namespace.
- The java API forms a standard interface for developing web applications, regardless of underlying of operating system without the implementation of java API it is impossible to develop dynamic applications.
- The javax.servlet package is the core of the servlet API.
- Servlet interface is a collection of empty method signatures.
- ServletContext allows enabling applications to load servlets and filters at runtime that are needed.
- The ServletConfig interface is implemented by the server. It allows a servlet to obtain configuration data when it is loaded.
- The user request is represented by the ServletRequest object passed by the servlet container as the first argument to the service method.
- The service method's second argument is a ServletResponse object, which represents the response to the user.
- The ServletRequest interface defines an object used to encapsulate information about the user's request, including parameter name/value pairs, attributes and an input stream.

- The `getRemoteAddress` and `getRemoteHost` methods are two methods that you can use to retrieve the user's computer identity.
- `ServletResponse` interface provides methods to the servlet for replying to the client.
- The `ServletResponse` interface represents the response to the user.
- `javax.servlet.GenericServlet` class provides the basic implementation of the servlet interface and `ServletConfig` interface.
- A servlet container allows multiple requests for the same servlet by creating a different thread to service each request.
- A functional `HttpServlet` must override at least one of the methods typically `service()`, `doGet()`, `doPost()`.
- The `doPost()` method is called when the browser sends an HTTP request using the post method.
- This interface is implemented by server. It enables servlet request to obtain information about a client request.
- `HttpServletRequest` Interface extends `ServletRequest` Interface to provide request information for HTTP servlet.
- The `getParameterNames` method returns an Enumeration containing the parameter names.
- `HttpServletResponse` interface extends `ServletResponse` interface to provide HTTP protocol specific functionality including response header and status codes.
- The Hypertext Transfer Protocol (HTTP) is the network protocol that web servers and client browsers use to communicate with each other.
- With URL rewriting, you append a token or identifier to the URL of the next servlet or the next resource.
- A cookie is a small piece of information that is passed back and forth in the HTTP request and response.
- In servlet programming, a cookie is represented by the `Cookie` class in the `javax.servlet.http` package.
- The `HttpSession` object is accessible from other servlets in the same application.
- JavaBeans are reusable software components for Java.

### **Practice Questions**

---

1. What is Servlet?
2. Explain `GenericServlet`?
3. Explain Servlet interface?
4. Explain `HttpServlet`?
5. Explain `SingleThreadModel`?
6. Differentiate between Put and Get?
7. Explain various Session Management or Session Tracking?
8. Explain `HttpSession` class?

9. Explain Cookie? Explain Persistent cookies
10. Write any three methods of HttpSession class?
11. Write a Servlet program which will display Login Form and verify the login?
12. Write a servlet program that will display the total number of hits to user?
13. Explain the advantages of Servlet technology over other technology.
14. Explain the benefits of Servlet?
15. Write a servlet program which will take a number from user and display the factorial of that on client side.
16. Write a servlet program that will take a number from the user and display the table of that number to user?
17. Write a servlet program that will take two numbers from the user and display the addition, subtraction, multiplication of that number to user?
18. Explain the following methods:
  - (a) setContentType( )
  - (b) getWriter( )
  - (c) getParameter( )
  - (d) getParameterNames( )
  - (e) getCookies( )
19. Explain the Life Cycle method of servlet.
20. Write a servlet program that will display the details of Student table using JDBC in tabular form on user side?
21. Explain various Http doXXX methods?
22. Write a servlet program that will take data of student and store it in of Student table using JDBC?
23. Write a servlet program that will search particular data in Student table using JDBC, by taking some key from user.
24. Write a servlet for demonstrating the generic servlet class.
25. Write a servlet for demonstrating the generic servlet class.
26. Write a servlet to demonstrate the Http Servlet class using do Get ( ).
27. Write a servlet to demonstrate the Http Servlet class using do Post ( ).
28. Write a servlet to demonstrate the cookie.
29. What is JSP?
30. Explain the following terms of JSP:
  - (i) Scriptlet
  - (ii) Expression
  - (iii) Directives
  - (iv) Declaration
31. Compare JSP and Servlet.
32. With the help of diagram describe life cycle of JSP page.
33. What is javabeans? Explain with example.

