CONFIGURING AND INSTALLING APACHE TOMCAT SERVER

Apache Tomcat server is an open source Java-capable HTTP server and servlet container developed by Apache Software Foundation(ASF).

This could execute special Java programs known as Java Servlet and Java Server Pages (JSP). The sites for Tomcat are http://tomcat.apache.org or http://www.apache.org .

Tomcat was originally written by James Duncan Davison , based on an earlier Sun's server called Java Web Server (JWS). Tomcat is an HTTP application runs over TCP/IP. In other words, the Tomcat server runs on a specific TCP port in a specific IP address. The default TCP port number for HTTP protocol is 80, which is used for the production HTTP server. For test HTTP server, any unused port number between 1024 and 65535 can be chosen.

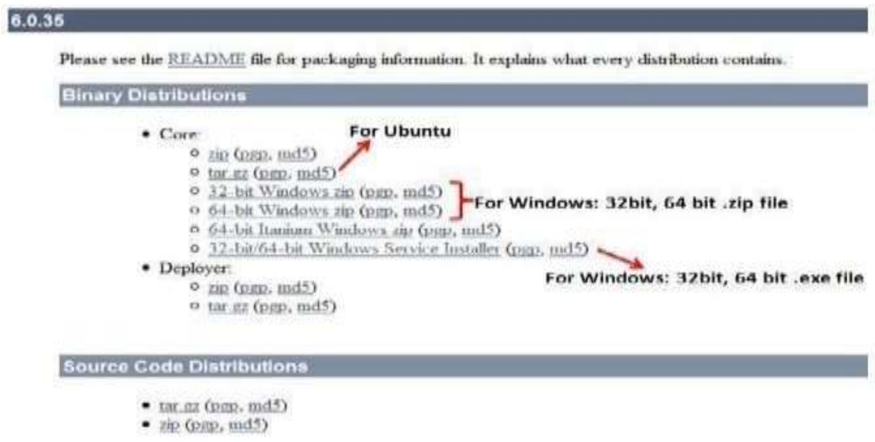
Installing Apache Tomcat Server

The basic environment required for installing Apache Tomcat Server is given below:

- JDK 6 (Java SE 6) (Tomcat 6 requires any installed Java 5 or later JRE (32-bit or 64-bit). We have used JRE 6.)
- Apache Tomcat 6.x
- Windows OS

Step 1: Downloading Apache Tomcat

Download the Apache Tomcat Server from http://tomcat.apache.org/download-60.cgi. Here we have used "Apache Tomcat 6.0.35″ version.



6.0.35

Please see the README file for packaging information. It explains what every distribution contains.

Binary Distributions

- Core:
  - zip (pgp, md5)
  - tar.gz (pgp, md5)
  - 32-bit Windows zip (pgp, md5)
  - 64-bit Windows zip (pgp, md5)
  - 64-bit Itanium Windows zip (pgp, md5)
  - 32-bit/64-bit Windows Service Installer (pgp, md5)
- Deployer:
  - zip (pgp, md5)
  - tar.gz (pgp, md5)

Source Code Distributions

- tar.gz (pgp, md5)
- zip (pgp, md5)

For Ubuntu
For Windows: 32bit, 64 bit .zip file
For Windows: 32bit, 64 bit .exe file

Figure 3.9 Tomcat for various OS

Step 2: Installing Apache Tomcat

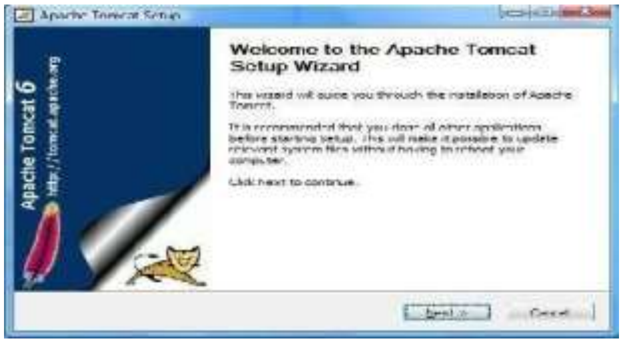
Tomcat can be installed on any operating system that supports the zip or tar formats.To install Apache Tomcat, unzip the downloaded (.zip) file to a safe location. For simplicity and easy access, unzip Tomcat in “C:\Tomcat6\” directory.

Step 3: ***Click on the .exe file***

After downloading windows installer file(.exe file), double click on it and follow the steps given below:

Welcome screen

Simply click on the ‘**Next**‘ button to continue installation process.



Apache Tomcat Setup

Welcome to the Apache Tomcat Setup Wizard

This wizard will guide you through the installation of Apache Tomcat.

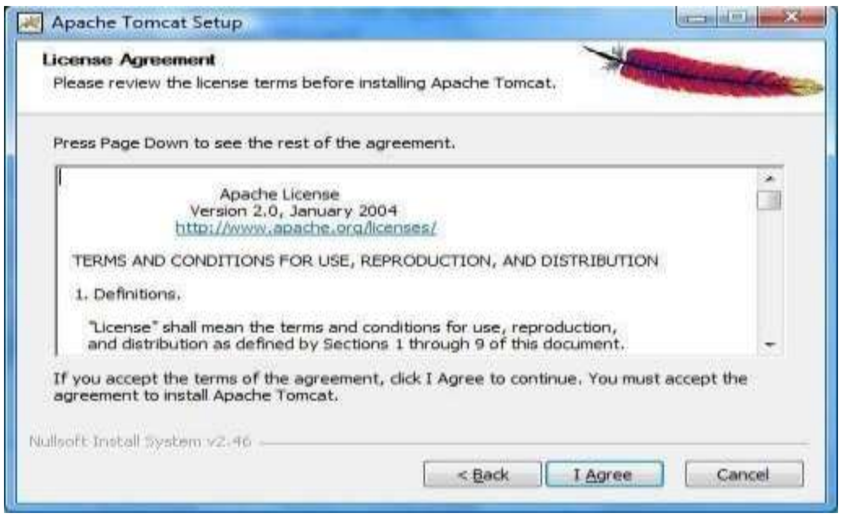
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 3.10 Welcome Screen

License Agreement screen

Accept the terms of the agreement by clicking on ‘**I Agree**‘ button.


Apache Tomcat Setup

License Agreement

Please review the license terms before installing Apache Tomcat.

Press Page Down to see the rest of the agreement.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

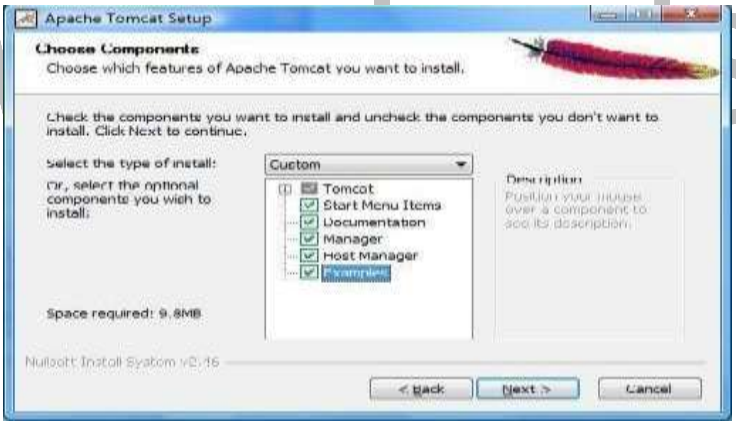
"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Apache Tomcat.

Figure 3.11 License Agreement screen

Choose the components

Choose the features of Apache Tomcat you want to install by checking the components and click‘**Next**‘.


Apache Tomcat Setup

Choose Components

Choose which features of Apache Tomcat you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Custom

Or, select the optional components you wish to install:

- Tomcat
- ✓ Start Menu Items
- ✓ Documentation
- ✓ Manager
- ✓ Host Manager
- ✓ Examples

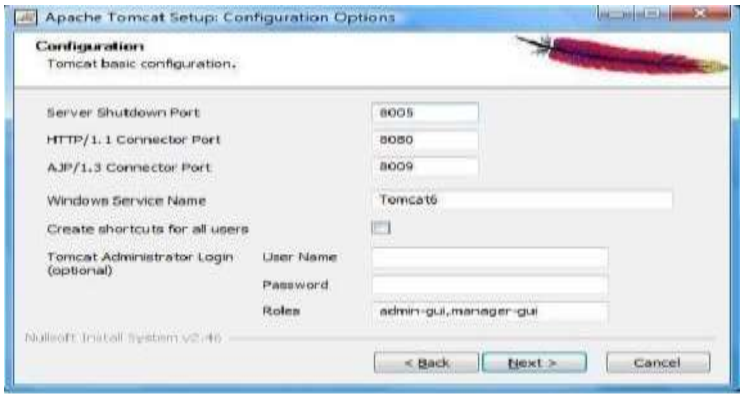
Description: Position your mouse over a component to see its description.

Space required: 9.8MB

Figure 3.12 Choose the components

Tomcat Configuration options

The default port number for Tomcat to process HTTP requests is 8080. The port numbers can be changed after the installation in server.xml which is located in/conf/server.xml. Provide the username and password for Administrator login and click on the ‘**Next’** button.


Apache Tomcat Setup: Configuration Options

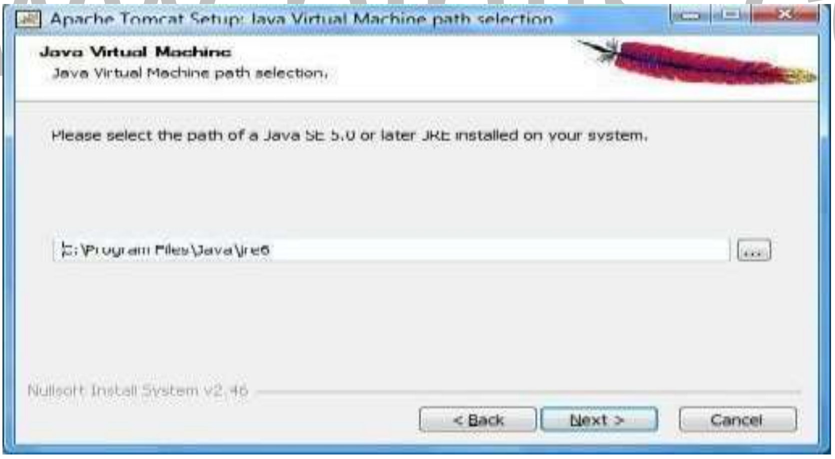
Configuration
Tomcat basic configuration.

Server Shutdown Port: 8005
HTTP/1.1 Connector Port: 8080
AJP/1.3 Connector Port: 8009
Windows Service Name: Tomcat6
Create shortcuts for all users
Tomcat Administrator Login (optional): User Name, Password, Roles: admin-gui,manager-gui

Figure 3.13 Tomcat Configuration options

Installed JRE path

The installer uses the registry to determine the base path of a Java 5 or later JRE, including the JRE installed as part of the full JDK.When running on a 64-bit operating system, the installer will first look for a 64-bit JRE and only look for a 32-bit JRE if a 64-bit JRE is not found.It is not mandatory to use the default JRE detected by the installer. Any installed Java 5 or later JRE (32-bit or 64-bit) may be used by clicking on the browse button and click '**Next**'.


Apache Tomcat Setup: Java Virtual Machine path selection

Java Virtual Machine
Java Virtual Machine path selection.

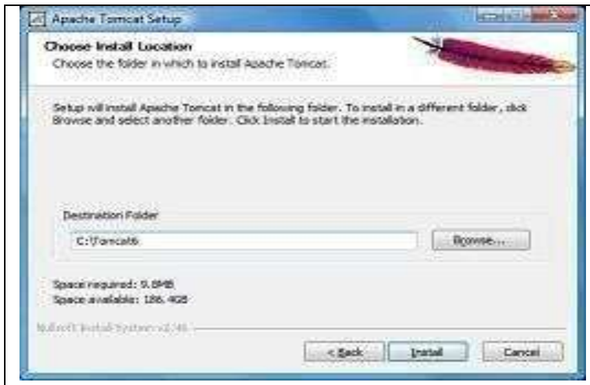
Please select the path of a Java SE 5.0 or later JRE installed on your system.

C:\Program Files\Java\jre6

Figure 3.14 Installed JRE path

Choose Installation Location

In Windows, by default the location will be provided as 'C:\Program Files\Apache Software Foundation\Tomcat 6.0'. But for simplicity, we recommend you to use '**C:\Tomcat6**' as shown below and click '**Install**'.



Choose Install Location

Choose the folder in which to install Apache Tomcat.

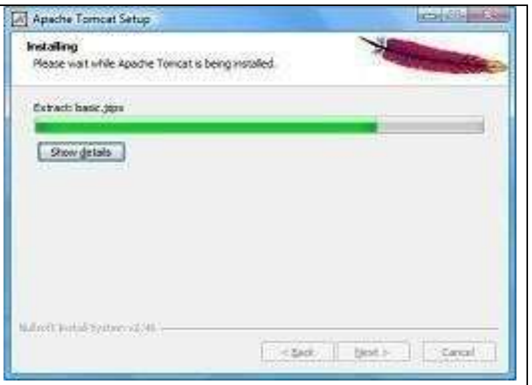
Setup will install Apache Tomcat in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Tomcat6

Space required: 9.8MB

Space available: 186.4GB

a) Choose install location

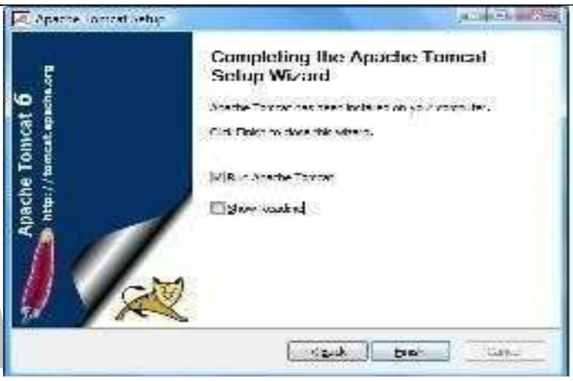


Installing

Please wait while Apache Tomcat is being installed.

Extract: basic.jsps

b)Installing window



Completing the Apache Tomcat Setup Wizard

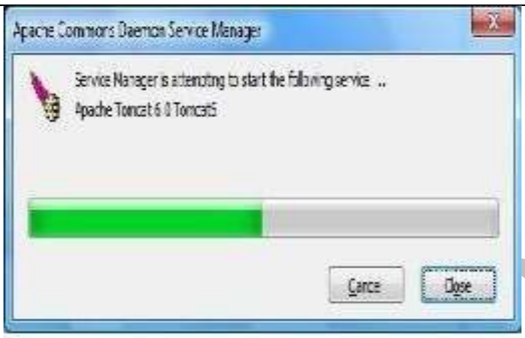
Apache Tomcat has been installed on your computer.

Click Finish to close this wizard.

[✓] Run Apache Tomcat

[ ] Show Readme

c) Setup completion



Service Manager is attempting to start the following service ...

Apache Tomcat 6.0 Tomcat6

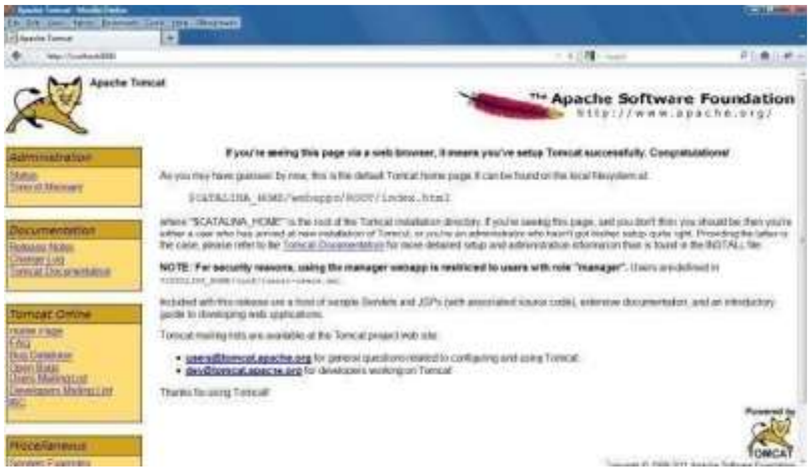
d) Apache Service Manager window

Figure 3.15 Choose Installation Location

Once the service has been started, an **Apache Tomcat icon appears on Windows Taskbar** (bottom right).

Test the Installation

Open browser and type http://localhost:8080 and see the Apache Tomcat home page as shown below.


Apache Tomcat

The Apache Software Foundation
http://www.apache.org/

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the manager webapp is restricted to users with role "manager". Users are defined in [illegible]

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

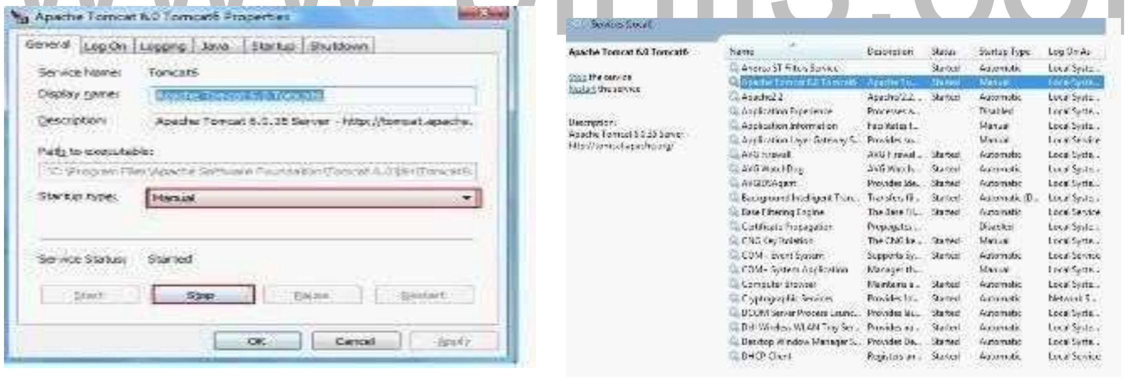
- users@tomcat.apache.org for general questions related to configuring and using Tomcat
- dev@tomcat.apache.org for developers working on Tomcat

Thanks for using Tomcat!

Figure 3.16 Testing the installation

Configuring Apache Tomcat

To manually configure the server, double-click on the Tomcat icon in Taskbar to open the Apache Tomcat properties dialog. Select Manual as startup type and start the server or stop the server and click OK.


Apache Tomcat 6.0 Tomcat6 Properties

Service Name: Tomcat6
Display name: Apache Tomcat 6.0 Tomcat6
Description: Apache Tomcat 6.0.35 Server - http://tomcat.apache.
Path to executable: "C:\Program Files\Apache Software Foundation\Tomcat 6.0\bin\Tomcat6
Startup type: Manual
Service Status: Started

Services (Local)

Apache Tomcat 6.0 Tomcat6

Stop the service
Restart the service

Description:
Apache Tomcat 6.0.35 Server - http://tomcat.apache.org/

Name	Description	Status	Startup Type	Log On As
Andrea ST Filters Service		Started	Automatic	Local Syste...
Apache Tomcat 6.0 Tomcat6	Apache To...	Started	Manual	Local Syste...
Apache2.2	Apache/2.2...	Started	Automatic	Local Syste...
Application Experience	Processes a...		Disabled	Local Syste...
Application Information	Facilitates t...		Manual	Local Syste...
Application Layer Gateway S...	Provides su...		Manual	Local Service
AVG Firewall	AVG Firewal...	Started	Automatic	Local Syste...
AVG WatchDog	AVG Watch...	Started	Automatic	Local Syste...
AVGIDSAgent	Provides Ide...	Started	Automatic	Local Syste...
Background Intelligent Tran...	Transfers fil...	Started	Automatic (D...	Local Syste...
Base Filtering Engine	The Base Fil...	Started	Automatic	Local Service
Certificate Propagation	Propagates ...		Disabled	Local Syste...
CNG Key Isolation	The CNG ke...	Started	Manual	Local Syste...
COM+ Event System	Supports Sy...	Started	Automatic	Local Service
COM+ System Application	Manages th...		Manual	Local Syste...
Computer Browser	Maintains a...	Started	Automatic	Local Syste...
Cryptographic Services	Provides thr...	Started	Automatic	Network S...
DCOM Server Process Launc...	Provides lau...	Started	Automatic	Local Syste...
Dell Wireless WLAN Tray Ser...	Provides a ...	Started	Automatic	Local Syste...
Desktop Window Manager S...	Provides De...	Started	Automatic	Local Syste...
DHCP Client	Registers an...	Started	Automatic	Local Service

Figure 3.17 Manual configuration

Double-click on /bin/Tomcat6w.exe (in our case, it is C:\Tomcat6\bin) and follow the step as above.If it did not open the Apache Tomcat properties window then try to run it as administrator (Right click on exe file->Run as administrator).Using Windows Services: Open Control Panel\Administrative Tools and double-click on Services.Double-click on Apache Tomcat service and do as above.

COOKIES

The cookie is stored on the user's machine, but it is not an executable program and cannot do anything to the stored machine.

A cookie is a short piece of data, not code, which is sent from a web server to a web browser when that browser visits the server's site.

Cookies are embedded in the HTML information flowing back and forth between the user's computer and the servers.Whenever a web browser requests a file from the web server it sends the request along with a cookie, the browser sends a copy of that cookie back to the server along with the request. Thus a server sends a cookie and the browser sends it back whenever it request another file from the same server. In this way, the server knows that the user have visited before and can coordinate the access to different pages on its web site.

Example: An Internet shopping site uses a cookie to keep track of which shopping basket belongs to a user.

Components of cookies

Cookies are a plain text data record of 5 variable-length fields:

- **Expiry time**: The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
- **Domain:** The domain name of the site.
- **Path:** The path to the directory or web page that set the cookie.
- **Secure:** If this field contains the word "secure" then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.
- **Name=Value:** Cookies are set and retrieved in the form of key and value pairs.

Uses of cookies

Identifying a user during an e-commerce session, Avoiding username and password, Customizing a site and Focusing advertising

Types of cookies

There are two types of cookies: Session Cookies and persistent cookies

Session Cookies

Session cookies are stored in memory during the applet or application session.Session cookies expire when the applet or application exits and are automatically deleted. These cookies usually store a session ID that is not personally identifiable to users, allowing the user to move from page to page without having to log-in repeatedly. They are widely used by commercial web sites.

Permanent or Persistent Cookies

Permanent cookies are stored in persistent storage and are not deleted when the application exits. They are deleted when they expire. They can retain user preferences for a particular web site, allowing those preferences to be used in future sessions. Permanent cookies can be used to identify individual users, so they may be used by web sites to analyze user's surfing behavior within the web site. These cookies can also be used to provide information about the numbers of visitors, the average time spent on a particular page, and the general performance of the web site.They are usually configured to keep track of users for a prolonged period of time, in some cases many years into the future.

Methods in Servlet Cookies:

Method	Description
public void setDomain(String pattern)	sets the domain to which cookie applies.
public String getDomain()	gets the domain to which cookie applies.
public void setMaxAge(int expiry)	sets how much time (in seconds) should elapse before the cookie expires. By default, the cookie will last only for the current session.
public int getMaxAge()	returns the maximum age of the cookie, specified in seconds, By default,
public String getName()	returns the name of the cookie. The name cannot be changed after creation.
public void setValue(String newValue)	sets the value associated with the cookie.
public String getValue()	gets the value associated with the cookie.
public void setPath(String uri)	sets the path to which this cookie applies. By default, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
public String getPath()	gets the path to which this cookie applies.
public void setSecure(boolean flag)	sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.
public void setComment(String purpose)	specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user.

public String getComment()	returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
public void setDomain(String pattern)	sets the domain to which cookie applies.

Programming cookies

Setting Cookies with Servlet

Setting cookies with servlet involves three steps:

(1) Creating a Cookie object: Call the Cookie constructor with a cookie name and a cookie value, both of which are strings.

Cookie cookie = new Cookie("key","value");

(2) Setting the maximum age:Use setMaxAge() to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 20 hours.

cookie.setMaxAge(60*60*20);

(3) Sending the Cookie into the HTTP response headers: Use response.addCookie() to add cookies in the HTTP response header as follows:

response.addCookie(cookie);

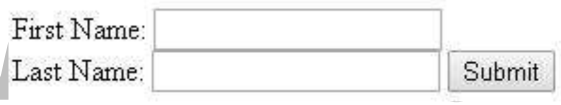
Setting cookies

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
public class HelloForm extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException
  { Cookie firstName = new Cookie("first_name",
request.getParameter("first_name"));
      Cookie lastName = new Cookie("last_name",
request.getParameter("last_name"));
      firstName.setMaxAge(60*60*24);
      lastName.setMaxAge(60*60*24);
      response.addCookie( firstName );
      response.addCookie( lastName );
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
```

```
    out.println( "<html>\n" +"<head><title>" + title + "</title></head>\n" +
          "<body bgcolor=\"#f0f0f0\">\n" + "<h1 align=\"center\">" + title +
"</h1>\n" +
          "<ul>\n" + " <li><b>First Name</b>: " +
request.getParameter("first_name") + "\n" + " <li><b>Last
Name</b>: " + request.getParameter("last_name") + "\n" +
          "</ul>\n" + "</body></html>");
 } }
```

Form.html

```
<html><body>
<form action="HelloForm" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form></body></html>
```



Reading Cookies with Servlet:

To read cookies, create an array of javax.servlet.http.Cookie objects by calling the getCookies( ) method of HttpServletRequest. Then cycle through the array, and use getName() and getValue() methods to access each cookie and associated value.

Reading Cookies

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
public class ReadCookies extends HttpServlet {
  publicvoiddoGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException
  { Cookie cookie = null;
      Cookie[] cookies = null;
```

```
         cookies = request.getCookies();
           response.setContentType("text/html");
           PrintWriter out = response.getWriter();
              out.println( "<html>\n" + "<head><title>" + title + "</title></head>\n" +
                 "<body bgcolor=\"#f0f0f0\">\n" );
          if( cookies != null )
        { out.println("<h2> Found Cookies Name and Value</h2>");
             [illegible]
             {                    cookie = cookies[i];
                    out.print("Name : " + cookie.getName( ) + ", ");
                      out.print("Value: " + cookie.getValue( )+" <br/>");          }     }
 else{            out.println("<h2>No cookies founds</h2>");       }
       out.println("</body>");       out.println("</html>");  }}
```

```
Found Cookies Name and Value
Name : first_name, Value: Priya
Name : last_name, Value: Dharshini
```

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
public class DeleteCookies extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
  {            Cookie cookie = null;
            Cookie[] cookies =null;
                   cookies = request.getCookies();
                  response.setContentType("text/html");
            PrintWriter out = response.getWriter();
          out.println("<html>\n" + "<head><title>" + title + "</title></head>\n" +
          "<body bgcolor=\"#f0f0f0\">\n" );
      if( cookies != null )        {
      out.println("<h2> Cookies Name and Value</h2>");
      for (int i = 0; i < cookies.length; i++)
      {          cookie = cookies[i];
        if((cookie.getName( )).compareTo("first_name") == 0 )
         {    cookie.setMaxAge(0);
              response.addCookie(cookie);
              out.print("Deleted cookie : " + cookie.getName( ) + "<br/>");
         }
        out.print("Name : " + cookie.getName( ) + ", ");
        out.print("Value: " + cookie.getValue(  )+" <br/>");         }     }
else
{  out.println("<h2>No  cookies founds</h2>");      }
    out.println("</body>");      out.println("</html>");  }}
```

```
Cookies Name and Value
Deleted cookie : first_name
Name : first_name, Value: Priya
Name : last_name, Value: Dharshini
```

To delete the cookies in Internet Explorer manually. Start at the Tools menu and select Internet Options. To delete all cookies, press Delete Cookies.

Advantages of cookies:

- Cookies are simple to use and implement.
- Occupies less memory as they do not require any server resources and are stored on the user's computer so no extra burden on server.
- Cookies can be configured to expire when the browser session ends (session cookies) or they can exist for a specified length of time on the client's computer (persistent cookies).
- Cookies persist a much longer period of time than Session state.

Disadvantages of cookies:

- Cookies are not secure as they are stored in clear text they may pose a possible security risk as anyone can open and tamper with cookies.
- Several limitations exist on the size of the cookie text (4kb in general), number of cookies(20 per site in general), etc.
- User has the option of disabling cookies on his computer from browser's settings.
- Cookies will not work if the security level is set to high in the browser.
- Users can delete cookies.
- Users browser can refuse cookies.
- Complex type of data not allowed (e.g. dataset etc). It allows only plain text (i.e. cookie allows only string content.

JAVA SERVER PAGES (JSP)

JavaServer Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications.

The tags in JSP are enclosed within <% and %>.

JSP tags can be used for a variety of purposes, such as

- retrieving information from a database or registering user preferences,
- accessing JavaBeans components,
- passing control between pages and
- sharing information between requests, pages etc.

Advantages of JSP over CGI

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.
- 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.
- 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.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.
- JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages of JSP over Active Server Pages (ASP):

- The dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use.
- It is portable to other operating systems and non-Microsoft Web servers.

Advantages of JSP over Pure Servlets:

- It is an extension to the servlets.
- Easy to maintain thanservlets
- No need for recompilation and redeployment (If JSP page is modified, we don't need to recompile and redeploy the project.).
- Less code.

Architecture of JSP:

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.

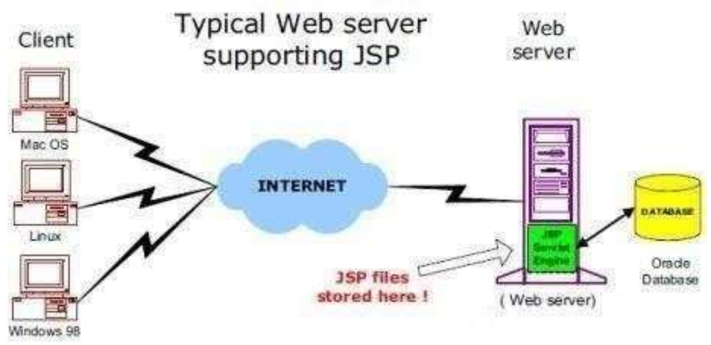


Figure 3.21 JSP-architecture

Processing of JSP

The 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.

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 hasn't 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.

A JSP page is really just another way to write a servlet without having to be a Java programming. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

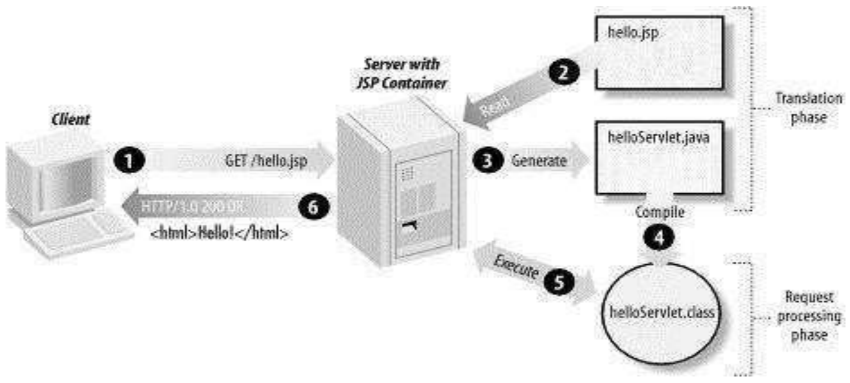


Figure 3.22 Processing of JSP

Difference between JSP and Servlets

JSP	Servlets
JSP is a webpage scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
In MVC, JSP act as a view.	In MVC, servlet act as a controller.
It's easier to code in JSP than in Java Servlets.	More code is needed here.
JSP are generally preferred when there is not much processing of data required.	Servlets are used when there is more processing and manipulation involved.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	There is no such custom tag facility in servlets.
We can achieve functionality of JSP at client side by runningJavaScript at client side.	There are no such methods for servlets.

Lifecycle of JSP

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.The following are the phases followed by a JSP: Compilation, Initialization, Execution and Cleanup

- **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: Parsing the JSP, Turning the JSP into a servlet and Compiling the servlet.

- **JSP Initialization:**

  When a container loads a JSP it invokes the jspInit() method before servicing any requests.In case of 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. The jspInit()initialize database connections, open files, and create lookup tables.

- **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.

  ```
  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 ie. GET, POST, DELETE etc.

- **JSP Cleanup:**

  The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.Override jspDestroy to perform any cleanup, such as releasing database connections or closing open files.

  ```
  public void jspDestroy()
  { //cleanup code }
  ```

JSP Syntax

A JSP document contains the following tags: scriplet, expressions and declaration tags.

- **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.Any text, HTML tags, or JSP elements must be outside the scriptlet.

1. <% code fragment %> OR
2. <jsp:scriptlet> code fragment </jsp:scriptlet> (XML format)

Scriplet

```
<html><head><title>Hello </title></head>
<body>
Hello <br/>
<%out.println("The IP address is " + request.getRemoteAddr());
%>
</body></html>
```

Declaration

This part declares one or more variables or methods that used in Java code later in the JSP file.

1. <%! declaration; [declaration;]+ ... %> OR
2. <jsp:declaration>   code fragment </jsp:declaration> (XML format)

Example: <%! int i = 0; %> ;<%! int a, b, c; %> ;<%! Circle a = new Circle(2.0); %>

- **JSP Expression:**

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, the expression could be used 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 semicolon to end an expression is invalid.

1. <%= expression %> OR
2. <jsp:expression>   expression</jsp:expression> (XML format)

Expression tags

Index.html

```
<html><body>
<form  action="welcome.jsp">
<input type="text" name="uname"><br/>
```

```
<input type="submit" value="go">
</form></body></html>
```

Hai.jsp

```
<html>
<body>
<%= "Hai"+request.getParameter("uname") %>
</form></body></html>
```

In this example, the username is displayed using the expression tag. The index.html file gets the username and sends the request to the hai.jsp file, which displays the username.

- **JSP Comments:**
  - JSP comment marks text or statements that the JSP container should ignore.

    <%-- This is JSP comment --%>

Declarations and Comments:

```
<html><head><title>A Comment Test</title></head>
<body><h2> Comments</h2><p>
  Today's date: <%= (new java.util.Date()).toLocaleString()%>
<%-- Displays today's date --%>
</p></body></html>
```

```
A Comment Test
Today's date: 01-JAN-2015 21:24:25
```

JSP Directives:

A JSP directive affects the overall structure of the servlet class.

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

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. **The page Directive:**

   The page directive is used to provide instructions to the container that pertain to the current JSP page. Page directives can be used anywhere in the JSP page. By convention, page directives are coded at the top of the JSP page.

   1. *<%@ page attribute="value" %> OR*
   2. *<jsp:directive.page attribute="value" /> (XML format)*

2. **The 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.

   1. *<%@ include file="relative url" > OR*
   2. *<jsp:directive.include file="relative url" /> (XML Format)*

3. **The taglib Directive:**

   The JavaServer Pages API allows 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 a JSP page.

   1. *<%@ taglib uri="uri" prefix="prefixOfTag" > OR*
   2. *<jsp:directive.taglib uri="uri" prefix="prefixOfTag" /> (XML Format)*

   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.

JSP Actions:

JSP actions use constructs in XML syntax to control the behavior of the servlet engine.

<jsp:action_name attribute="value" />

Syntax	**Purpose**
jsp:include	Includes a file at the time the page is requested
jsp:useBean	Finds or instantiates a JavaBean
jsp:setProperty	Sets the property of a JavaBean

jsp:getProperty	Inserts the property of a JavaBean into the output
jsp:forward	Forwards the requester to a new page
jsp:plugin	Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin
jsp:element	Defines XML elements dynamically.
jsp:attribute	Defines dynamically defined XML element's attribute.
jsp:body	Defines dynamically defined XML element's body.
jsp:text	Use to write template text in JSP pages and documents.

Attributes:

There are two attributes that are common to all Action elements:

- **Id attribute:** The id attribute uniquely identifies the Action element, and allows the action to be referenced inside the JSP page. If the Action creates an instance of an object the id value can be used to reference it through the implicit object PageContext
- **Scope attribute:** This attribute identifies the lifecycle of the Action element. The id attribute and the scope attribute are directly related, as the scope attribute determines the lifespan of the object associated with the id. The scope attribute has four possible values: (a) page, (b)request, (c)session, and (d) application.

The <jsp:include> Action:

Let us define following two files (a)date.jps and (b) main.jsp as follows:

date.jsp file:

```
    <p>
      Today's date: <%= (new java.util.Date()).toLocaleString()%>
</p>
```

main.jsp file:

```
    <html><head><title>The include Action  Example</title>
    </head>
    <body><center><h2>The include action Example</h2>
```

```
<jsp:include page="date.jsp" flush="true" />
</center></body></html>
```

```
The include action  Example
Today's date: 12-Sep-2010  14:54:22
```

JSP Implicit Objects:

JSP Implicit Objects are the Java objects that the JSP Container makes available to developers in each page and developer can call them directly without being explicitly declared.JSP supports nine automatically defined variables, which are also called implicit objects.

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.

1. out. print(dataType dt)-   Print a data type value
2. config.getServletName()-gets the name of the servlet using config object.
3. pageContext.removeAttribute("attrName", PAGE_SCOPE) -removes the attribute from page scope.

Control-Flow Statements

JSP provides full power of Java to be embedded in the web application. All the APIs and building blocks of Java can be used in JSP programming including decision making statements, loops etc.

Decision-Making Statements

The if...else block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between Scriptlet tags.

If.else

```
<%! int day = 3; %><html>
<head><title>IF...ELSE Example</title></head>
<body><% if (day == 1 | day == 7)
{ %><p> Today is weekend</p>
<% }
else { %><p> Today is not weekend</p>
<% } %></body></html>
```

```
Today is not weekend
```

Switch:

```
<%! int day = 3; %>
<html><head><title>SWITCH...CASE   Example</title></head>
<body>
<% switch(day) {
case 0:out.println("It\'s Sunday."); break;
case 1:out.println("It\'s Monday."); break;
case 2:out.println("It\'s Tuesday."); break;
case 3:out.println("It\'s Wednesday."); break;
case 4:out.println("It\'s Thursday."); break;
case 5:out.println("It\'s Friday."); break;
default:out.println("It's Saturday.");}
%>
</body></html>
```

```
It's Wednesday.
```

Loop Statements:

All three basic types of looping blocks in Java can be used in JSP: for, while,and do…while blocks

For loop

```
<%! int fontSize; %>
<html>
<head><title>FOR LOOP Example</title></head>
<body><%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
<font color="green" size="<%= fontSize %>">
   JSP </font><br /><%}%></body></html>
```

```
JSP
JSP
JSP
```

JSP Operators:JSP supports all the logical and arithmetic operators supported by Java.

JSP Literals: The JSP expression language defines the following literals:Boolean: true and false, Integer: as in Java, Floating point: as in Java, String: with single and double quotes; "is escaped as \", ' is escaped as \', and \ is escaped as \\ and null.

JSP Standard Tag Library (JSTL)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

- Fast developement: JSTL provides many tags that simplifies the JSP.
- Code reusability: We can use the JSTL tags in various pages.
- It avoids the use of scriptlet tag.

There JSTL mainly provides 5 types of tags:

Tag Name	Description
core 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.

sql tags	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.
xml tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is http://java.sun.com/jsp/jstl/xml and prefix is x.
internationalization tags	The internationalization 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.
functions tags	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.

JSTL Core Tags

The JSTL core tags mainly provides 4 types of tags:

- miscellaneous tags: catch and out.
- url management tags: import, redirect and url.
- variable support tags: remove and set.
- flow control tags: forEach, forTokens, if and choose.

Syntax for core tags

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

Tag	**Description**
c:catch	It handles the exception and doesn't propagate the exception to error page. The exception object thrown at runtime is stored in a variable named var.
c:out	It is just like JSP expression tag but it is used for expression. It renders data to the page.
c:import	It is just like jsp include but it can include the content of any resource either within server or outside the server.
c:forEach	It repeats the nested body content for fixed number of times or over collection.
c:if	It tests the condition.
c:redirect	It redirects the request to the given url.

HTML forms with JSP tags

The browser uses two methods to pass some information to web server: GET Method and POST Method. The data entered in the forms reach the server through these methods.

GET method:

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ?character as follows:

Example: http://www.abc.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in the browser's Location:box. Never use the GET method if password or other sensitive information is passed to the server.

The GET method has size limitation: only 1024 characters can be in a request string.This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable which can be handled using getQueryString() and getParameter() methods of request object.

POST method:

A more reliable method of passing information to a backend program is the POST method.This method packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ?in the URL it sends it as a separate message.

This message comes to the backend program in the form of the standard input which can be parsed and use for processing.JSP handles this type of requests using getParameter() method to read simple parameters and getInputStream() method to read binary data stream coming from the client.

Reading Form Data using JSP

JSP handles form data parsing automatically using the following methods depending on the situation:

Methods	**Description**
getParameter()	to get the value of a form parameter.
getParameterValues()	Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
getParameterNames()	Call this method to get a complete list of all parameters in the current request.
getInputStream()	Call this method to read binary data stream coming from the client.

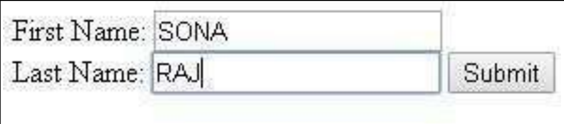
Get():

Form.html

```
<html><body><form action="main.jsp" method="GET">
First Name: <input type="text" name="first_name">
<br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form></body></html>
```

Main.jsp

```
<html><head><title>Using GET Method to Read Form  Data</title>
</head><body><center>
<h1>Using GET Method to Read Form Data</h1>
<ul><li><p><b>First Name:</b>
<%= request.getParameter("first_name")%></p></li>
<li><p><b>Last Name:</b>
<%= request.getParameter("last_name")%>
</p></li>
</ul></body></html>
```


First Name: SONA
Last Name: RAJ

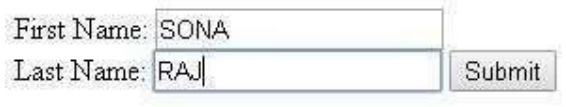
POST Method Using Form:

Form.html

```
<html><body><form  action="main.jsp"  method="POST">
First  Name:  <input  type="text"  name="first_name"><br />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form></body></html>
```

Main.jsp

```
<html><head>
<title>Using GET and POST Method to Read Form Data</title></head>
<body><center>
<h1>Using GET Method to Read Form Data</h1>
<ul><li><p><b>First Name:</b>
<%= request.getParameter("first_name")%></p></li>
<li><p><b>Last Name:</b>
<%= request.getParameter("last_name")%></p></li>
</ul></body></html>
```



Reading All Form Parameters:

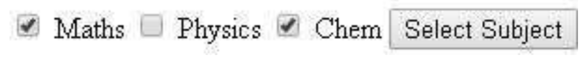
Form.html

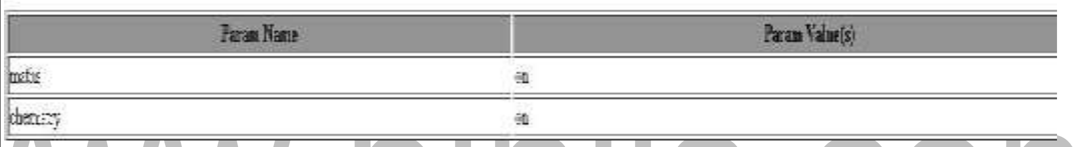
```
<html><body>
<form action="main.jsp" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics" /> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<input type="submit" value="Select Subject" />
</form></body></html>
```

Main.jsp

```
<%@ page import="java.io.*,java.util.*" %>
<html><head><title>HTTP Header Request Example</title>
</head>
<body><center><h2>HTTP Header Request Example</h2>
<table width="100%" border="1" align="center">
```

```
<tr bgcolor="#949494">
<th>Param Name</th><th>Param Value(s)</th></tr>
<%
  Enumeration paramNames = request.getParameterNames();
  while(paramNames.hasMoreElements()) {
    StringparamName=(String)paramNames.nextElement();
    out.print("<tr><td>" + paramName + "</td>\n");
    String paramValue = request.getHeader(paramName);
    out.println("<td> " + paramValue + "</td></tr>\n");
  }%></table></center></body></html>
```





Param Name	Param Value(s)
maths	on
chemistry	on

JDBC CONNECTIVITY

JDBC provides framework to connect to relational databases from java programs.

Figure 3.18 JDBC Architecture

- ***JDBC API provides industry-standard and database-independent connectivity between java applications and database servers.***

The JDBC library includes APIs for each of the tasks commonly associated with database usage:

- Making a connection to a database
- Creating SQL or MySQL statements
- Executing that SQL or MySQL queries in the database
- Viewing & Modifying the resulting records

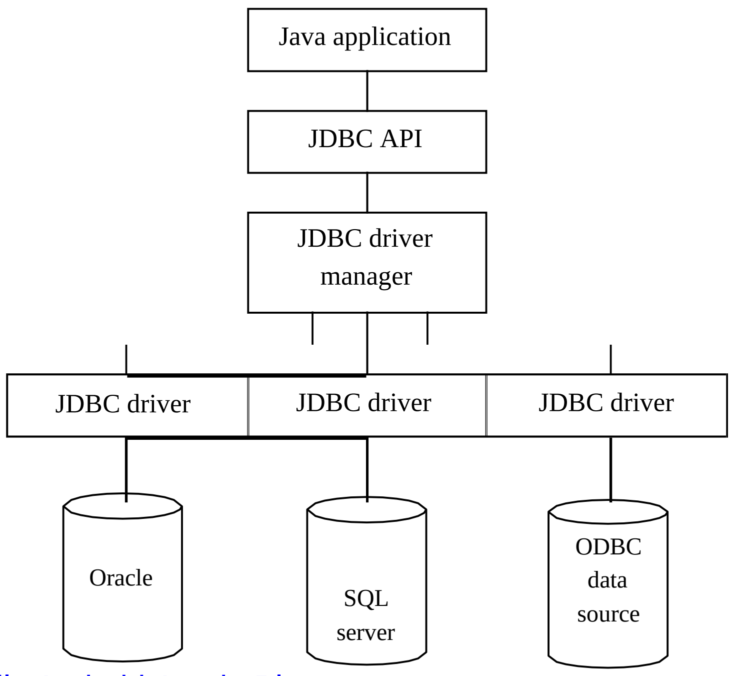
JDBC can be used with Java Applications, Java Applets, Java Servlets, Java ServerPages (JSPs) and Enterprise JavaBeans (EJBs)

JDBC Architecture

JDBC Architecture consists of two layers:

– **JDBC API:** This provides the application-to-JDBC Managerconnection.

– **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.



The JDBC API supports both two-tier and three-tier processing models for database access.The JDBC API uses a driver manager and database-specific drivers to provide connectivity to heterogeneous databases.

The **JDBC driver manager** ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Components of JDBC

The following are some of the important components of JDBC:

DriverManager:

This class manages a list of database drivers. It is responsible for matching the connection requests from the java application with the proper database driver using communication sub protocol.

Driver:

This interface handles the communications with the database server.The DriverManager manages these objects.

Connection:

This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

Statement:

The objects created from this interface are used to submit the SQL statements to the database.

ResultSet:

These objects hold data retrieved from a database after executing an SQL query using Statement objects. It acts as an iterator that moves through its data.

SQLException:

This class handles any errors that occur in a database application**.**

JDBC Driver

JDBC drivers implement the defined interfaces in the JDBC API for interacting with the database server.The JDBC drivers open database connections and interact with it by sending SQL or database commands then receiving results with Java.

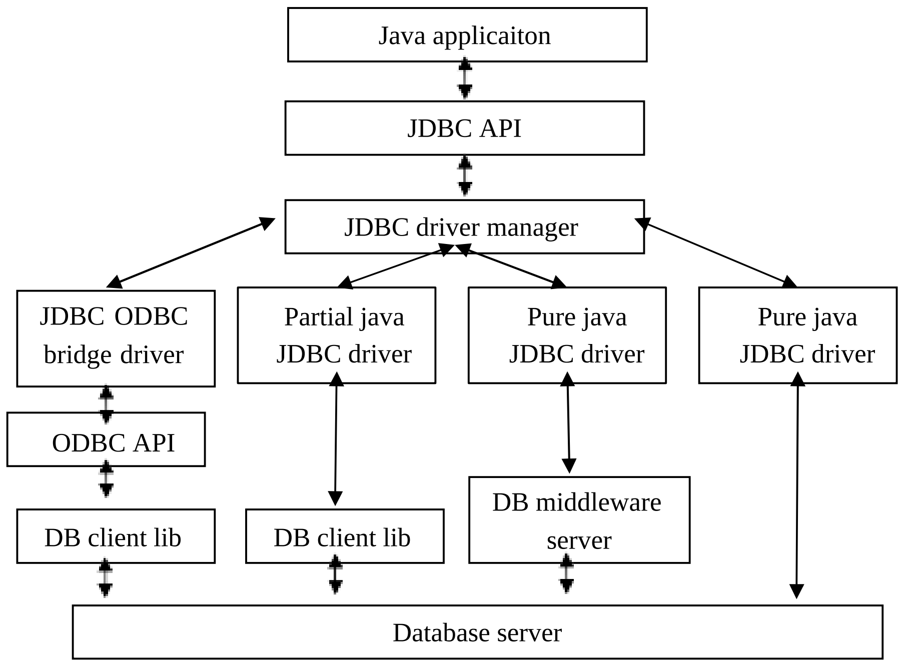


Figure 3.19 JDBC Drivers

1. **JDBC-ODBCBridge plus ODBC Driver (Type 1):**

   This driver uses ODBC driver to connect to database servers. The ODBC drivers must be installed in the machines from where the connection is established to JDBC drivers.This driver is almost obsolete and should be used only when other options are not available.

2. **Native API partly Java technology-enabled driver (Type 2):**

   This type of driver converts JDBC class to the client API for the RDBMS servers.The database client API should be installed at the machine from which we want to make database connection. Because of extra dependency on database client API drivers, this is also not preferred driver.

3. **Pure Java Driver for Database Middleware (Type 3):**

   This type of driver sends the JDBC calls to a middleware server that can connect to different type of databases. A middleware server must be installed to work with this kind of driver. This adds to extra network calls and slow performance. Hence this is also not widely used JDBC driver.

4. **Direct-to-Database Pure Java Driver (Type 4):**

   This is the preferred driver because it converts the JDBC calls to the network protocol understood by the database server. This solution doesn't require any extra APIs at the client side and suitable for database connectivity over the network. However for this solution, we should use database specific drivers, for example OJDBC jars provided by Oracle for Oracle DB and MySQL Connector/J for MySQL databases.

Creating a JDBC application

There are following six steps involved in building a JDBC application:

1. **Import the packages -** Include the packages containing the JDBC classes needed for database programming.
2. **Register the JDBC driver -** Initialize a driver so to open a communications channel with the database.
3. **Open a connection -** Using DriverManager.getConnection() method create a Connection object, which represents a physical connection with the database.
4. **Execute a query -** Statement for building and submitting an SQL statement to the database.
5. **Extract data from result set -** Use ResultSet.getXXX() method to retrieve the data from the result set.
6. **Clean up the environment -** Explicitly closes all database resources.

Connections using JDBC

The JDBC connections can be established in two ways:

– By using JDBC-ODBC bridge

– By using vendor specific JDBC driver

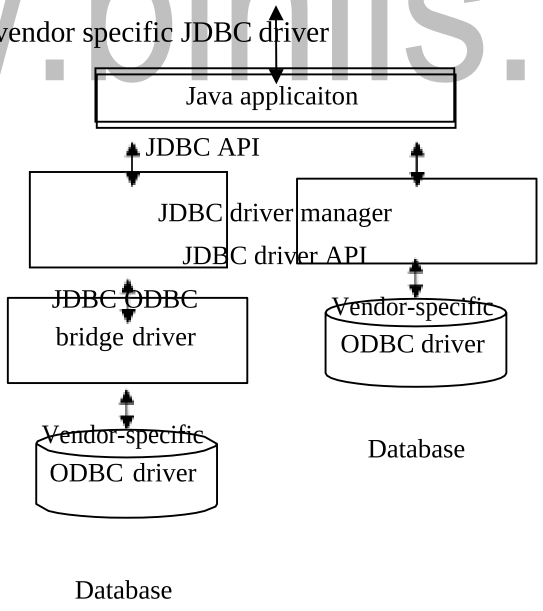


Figure 3.20 JDBC drivers

Define the Connection URL

URLs referring to databases use the jdbc: protocol and embed the server host, port, and database name (or reference) within the URL.

Establish the Connection

To make the actual network connection, pass the URL, database username, and database password to the getConnection method of the DriverManager class.The getConnection throws an SQLException.

Connection connection = DriverManager.getConnection(oracleURL, username, password);

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname: port Number/databaseName

Methods of connection class:

1. prepareStatement- Creates precompiled queries for submission to the database.
2. prepareCall- Accesses stored procedures in the database.
3. rollback/commit- Controls transaction management.
4. close- Terminates the open connection.
5. isClosed- Determines whether the connection timed out or was explicitly closed.

JDBC - Statements, PreparedStatement and CallableStatement

Once a connection is obtained we can interact with the database. The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable to send SQL or PL/SQL commands and receive data from the database. They also define methods that help bridge data type differences between Java and SQL data types used in a database

Interfaces	Recommended Use
Statement	Use for general-purpose access to thedatabase. Useful when using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Used when the SQL statements are to be executed many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use when database stored procedures are to be executed. The CallableStatement interface can also accept runtime input parameters.

Create a Statement Object

A Statement object is used to send queries and commands to the database.It is created from the Connection using createStatement.

Statement statement = connection.createStatement();

Most, but not all, database drivers permit multiple concurrent Statement objects to be open on the same connection.

Execute a Query or Update

The Statement object can be uses to send SQL queries by using the executeQuery method, which returns an object of type ResultSet.

Example: String query = "SELECT col1, col2, col3 FROM sometable";

ResultSet resultSet = statement.executeQuery(query);

The methods in the Statement class are:

Methods	Description
executeQuery	Executes an SQL query and returns the data in a ResultSet. The ResultSet may be empty, but never null.
executeUpdate	Used for UPDATE, INSERT, or DELETE commands. Returns the number of rows affected, which could be zero. Also provides support for Data Definition Language (DDL) commands, for example, CREATE TABLE, DROP TABLE, and ALTERTABLE.

executeBatch	Executes a group of commands as a unit, returning an array with the update counts for each command. Use addBatch to add a command to the batch group.
setQueryTimeout	Specifies the amount of time a driver waits for the result before throwing a SQLException.
getMaxRows/setMaxRows	Determines the number of rows a ResultSet may contain. Excess rows are silently dropped. The default is zero for no limit.

Process theResults

The simplest way to handle the results is to use the next method of ResultSet to move through the table a row at a time. Within a row, ResultSet provides various getXxx methods that take a column name or column index as an argument and return the result in a variety of different Java types. For instance, use getInt if the value should be an integer, getString for a String, and so on for most other data types. To just display the results, use getString for most of the column types.

```
while(resultSet.next())
{System.out.println(resultSet.getString(1) + " " + resultSet.getString(2) + " " +
resultSet.getString("firstname") + " " resultSet.getString("lastname"));}
```

Methods	Description
next/previous	Moves the cursor to the next (any JDBC version) or previous (JDBC version 2.0 or later) row in the ResultSet, respectively.
relative/absolute	The relative method moves the cursor a relative number of rows, either positive (up) or negative (down). The absolute method moves the cursor to the given row number. If the absolute value is negative, the cursor is positioned relative to the end of the ResultSet (JDBC 2.0).
getXxx	Returns the value from the column specified by the column name or column index as an Xxx Java type (see java.sql.Types). Can return 0 or null if the value is an SQL NULL.
wasNull	Checks whether the last getXxx read was an SQL NULL.
findColumn	Returns the index in the ResultSet corresponding to the specified column name.
getRow	Returns the current row number, with the first row starting at 1 (JDBC 2.0).

getMetaData	Returns a ResultSetMetaData object describing the ResultSet.
ResultSetMetaData	Gives the number of columns and the column names.

Close the Connection

To close the connection explicitly close() is used.

connection.close();

Closing the connection also closes the corresponding Statement and ResultSet objects.

The PreparedStatement Objects

The PreparedStatement interface extends the Statement interface which gives added functionality with a couple of advantages over a generic Statement object. This statement gives the flexibility of supplying arguments dynamically.

Example:

```
 PreparedStatement pstmt = null;
Try{ String SQL = "Update Employees SET age = ? WHERE id = ?";
pstmt = conn.prepareStatement(SQL);}
```

The CallableStatement Objects

A Connection object creates the CallableStatement object which would be used to execute a call to a database stored procedure.

Simple JDBC connection in a servlet

```
import java.io.IOException;import java.io.PrintWriter;import java.sql.Connection;
import java.sql.DriverManager;import java.sql.ResultSet;import java.sql.SQLException;
import java.sql.Statement;import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;importjavax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class  JDBCServlet  extends HttpServlet
{   public  void  doGet(HttpServletRequest  inRequest,   HttpServletResponse outResponse)
throws  ServletException,    IOException
    { PrintWriter out = null;
      Connection connection = null;
      Statement  statement;
      ResultSet rs;
```

```
        try {
              Class.forName("com.mysql.jdbc.Driver");
              connection =
DriverManager.getConnection("jdbc:mysql://localhost/products");
             statement = connection.createStatement();
             outResponse.setContentType("test/html");
             out = outResponse.getWriter();
              rs = statement.executeQuery("SELECT ID, title, price FROM product");
              out.println("<HTML><HEAD><TITLE>Products</TITLE></HEAD>");
              out.println("<BODY>");
              out.println("<UL>");
            while (rs.next()) {
                       out.println("<LI>" + rs.getString("ID") + " " + rs.getString("title")
+ " " + rs.getString("price"));
                          }
         out.println("</UL>");
         out.println("</BODY></HTML>");        }
catch (ClassNotFoundException e)
{    out.println("Driver Error");    }
catch (SQLException e)
{    out.println("SQLException: " + e.getMessage());    } }
 public void doPost(HttpServletRequest inRequest,    HttpServletResponse outResponse)
throws ServletException, IOException
{   doGet(inRequest, outResponse);  }}
```

ID	Title	Price
56	Soap	40
98	Shampoo	15

Batch Processing

Batch Processing allows grouping related SQL statements into a batch and submitting them with one call to the database. When several SQL statements are sent to the database at once, they can be clubbed together to reduce the amount of communication overhead. This is called **batch processing.** The following methods are used in batch processing:

S.No	Method	Description
1.	DatabaseMetaData. supportsBatchUpdates()	This method determines if the target database supports batch update processing.
2.	addBatch()	This method is used to add individual statements to the batch.
3.	executeBatch()	This method is used to start the execution of all the statements grouped together.
4.	clearBatch()	This method removes all the statements added with the addBatch() method. The statements cannot be selectively chosen.

The following code provides an example of a batch update using Statement object:

```
Statement stmt = conn.createStatement();
conn.setAutoCommit(false); // Set auto-commit to false
String SQL = "INSERT INTO Employees (id, first, last, age) " +
"VALUES(200,'Zia', 'Ali', 30)";
stmt.addBatch(SQL); // Add above SQL statement in the batch.
String SQL = "INSERT INTO Employees (id, first, last, age) " +
"VALUES(201,'Raj', 'Kumar', 35)"; // Create one more SQL statement
stmt.addBatch(SQL); // Add above SQL statement in the batch.
String SQL = "UPDATE Employees SET age = 35 " + "WHERE id = 100";
stmt.addBatch(SQL);
int[] count = stmt.executeBatch();// Create an int[] to hold returned values
conn.commit();//Explicitly commit statements to apply changes
```

Unit-3

SERVER SIDE PROGRAMMING

SERVLETS

Servlets are effective for developing Web-based solutions

CGI (Common Gateway Interface)

The common gateway interface (CGI) is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user. CGI does not communicate directly with the browser.

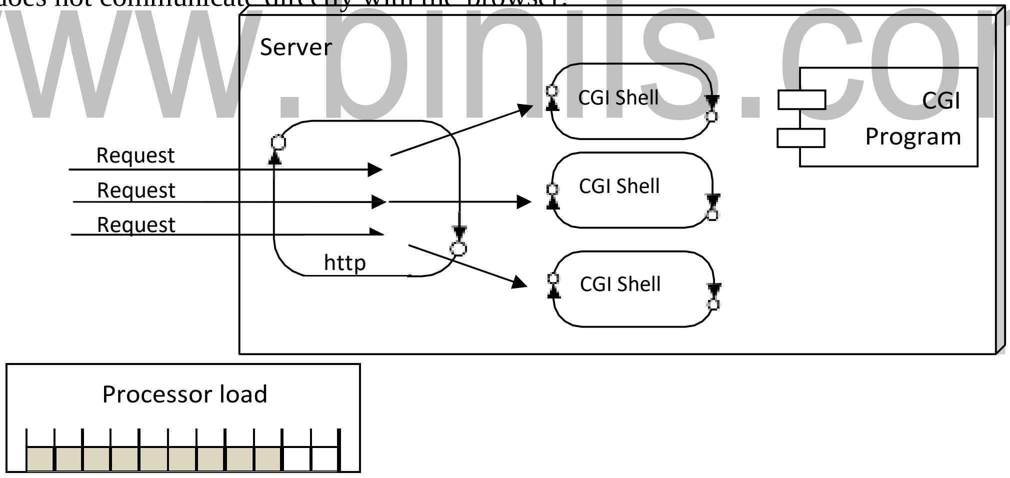


Figure 3.1 CGI

Advantages of Servlets over CGI

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

Capabilities of Servlet

Servlet is a web component that is deployed on the server to create dynamic web page.

Differences between process and threads

Process	**Threads**
Process run in separate address space	Threads of a process share address space.
Process have their own copy of data segment of parent process.	Threads have direct access to data segment of its process
Processes must use inter-process communication to communicate with sibling processes	Threads can directly communicate with other threads of that process.
Process carry more state information	Threads carry less state information
New process require duplication of parent process, and allocation of memory and resources for it are costly	New threads are easily created.

Servlet Architecture

The server that executes a servlet is called as the **servlet container or servlet engine.**

A client sends an HTTP request to the server or servlet container.

The server or servlet container receives the request and directs it to be processed by the appropriate servlet.

The servlet does its processing, which may be interacting with a database or other server-side components.

The servlet returns its results to the client.

From the programming perspective, all servlets must implement the Servlet interface of the packagejavax.servlet.

The methods of interface Servlet are invoked automatically by the servlet container.

This interface defines the following five methods:

void init(ServletConfig config)

ServletConfig getServletConfig()

String getServletInfo()

void service(ServletRequest request, ServletResponse response)

void destroy()

1. **void init(ServletConfig config)**

   The servlet container calls the init method only once after creating the servlet instance.

   The init method is used to initialize the servlet.

   The ServletConfig argument is supplied by the servlet container that executes the servlet.

2. **ServletConfig getServletConfig()**

   This method returns a reference to an object that implements interface ServletConfig.

   This object provides access to the servlet's configuration information such as servlet initialization parameters and the servlet's ServletContext, which provides the servlet with access to its environment

3. **String getServletInfo()**

   This method is defined by a servlet programmer to return a String containing servlet information such as the servlet's author and version.

4. **void service(ServletRequest request, ServletResponse response)**

   The servlet container calls this method to respond to a client request to the servlet.

5. **void destroy()**

   This "cleanup" method is called by the web container before removing the servlet instance from the service.

   Resources used by the servlet, such as an open file or an open database connection, should be deallocated here.

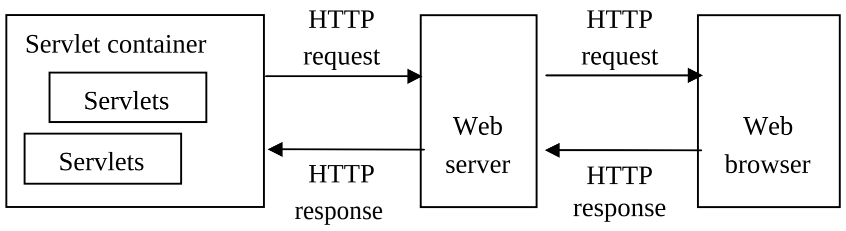


Figure: 3.3 Servlet Architecture

Servlet lifecycle

The web container maintains the life cycle of a servlet instance. The following are the phases in the life of a servlet:

- Servlet class is loaded.
- Servlet instance is created.
- init method is invoked.
- service method is invoked.
- destroy method is invoked

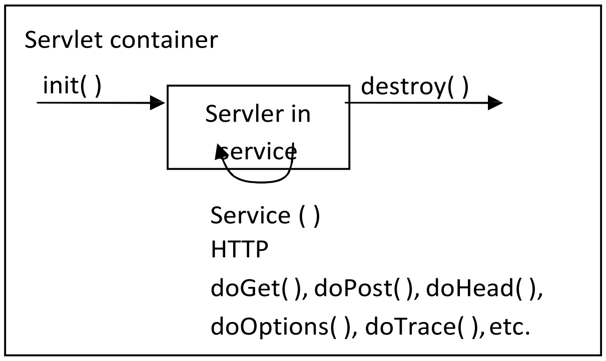


Figure 3.4 Servlet lifecycle

There are three states of a servlet: new, ready and end.

The servlet is in new state if servlet instance is created.

After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks.

When the web container invokes the destroy() method, it shifts to the end state.

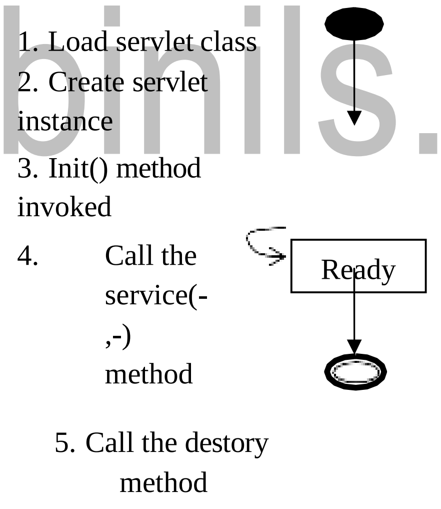

1. Load servlet class
2. Create servlet instance
3. Init() method invoked
4. Call the service(-,-) method
Ready
5. Call the destory method

Figure 3.5 State diagram of lifecycle of servlet

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance.The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface.

public void init(ServletConfig config) throws ServletException

4) service method is invoked

The web container calls the service method each time when request for the servlet is received.

If servlet is not initialized, it follows the first three steps as described above then calls the service method.

If servlet is initialized, it calls the service method. Remember that servlet is initialized only once.

The service() method is the main method to perform the actual task.

The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.The doGet() and doPost() are most frequently used methods with in each service request.

Syntax:public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

The servlet can be created by three ways:

1. By implementing Servlet interface
2. By inheriting GenericServlet class
3. By inheriting HttpServlet class

Servlet Interface

Servlet interface provides common behaviour to all the servlets. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods (init, service and destroy) that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods in servlet interface

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

Creating a servlet using servlet interface

```
import java.io.*;
import
javax.servlet.*;
public class First implements Servlet
     ServletConfig config=null;
      public void init(ServletConfig config)
      {                this.config=config;                    }
         public void service(ServletRequest req,ServletResponse res)
                              throws IOException,ServletException
         {  res.setContentType("text/html");
         PrintWriter out=res.getWriter();
         out.print("<html><body>");
         out.print("<b>This is a simple servlet</b>");
         out.print("</body></html>");            }
        public void destroy()
        {              System.out.println("servlet is  destroyed");
                       }public ServletConfig getServletConfig()
      {return config;         }
      public String getServletInfo()
```

The above program creates a servlet by implementing the servlet interface.

The servlet is initialized with its configuration in the init().

The service() does the real operation of the servlet.

In this example, the servlet creates a web page that displays “This is a simple servlet”.

The service method creates two objects in its parameter field: req and res.

The req object is created for ServletRequest class. All the requests will be issues by this object.

The res object is created for ServletResponseclass. This is responsible for the output of the servlet.

res.ContentType() describes the content of the output.

PrintWriter is the output stream with object out.

The print() is called using the object of the PrintWriter class.

The print() contains the HTML tags to create the web page.

The servlet is destroyed using destroy(). In this example, getServletInfo () is optional and returns the copyrights information.

ServletConfig Interface

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

public ServletConfig getServletConfig(); //This returns the object of the ServletConfig.

Methods of ServletConfig interface

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

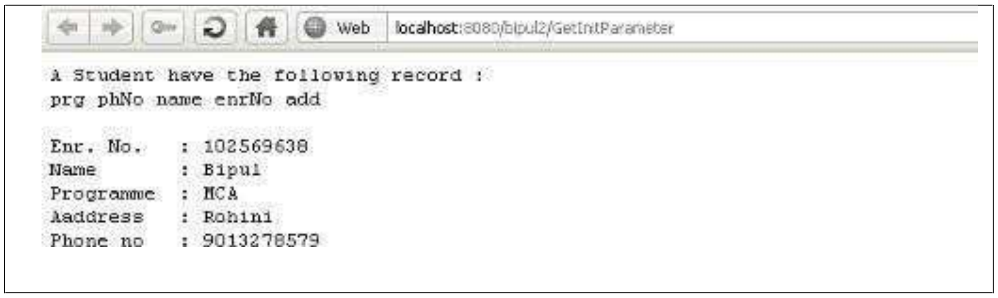
Servlet configuration

```
import java.io.*;import java.util.*;import javax.servlet.*;
import javax.servlet.http.*;
public class GetInitParameter extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
PrintWriter out = response.getWriter();
out.println("A Student have the following record : ");
Enumeration enm =
getServletConfig().getInitParameterNames();while
(enm.hasMoreElements()) {
out.print(enm.nextElement() + " ");}
out.println("\nEnr. No.:"+getServletConfig().getInitParameter("enrNo"));
out.println("Name : "+ getServletConfig().getInitParameter("name"));
out.println("Programme : " + getServletConfig().getInitParameter("prg"));
```

web.xml

```
<web-app><servlet><init-param>
<param-name>enrNo</param-name>
<param-value>102569638</param-value></init-param>
<init-param><param-name>name</param-name>
<param-value>Bipul</param-value></init-param>
<init-param><param-name>prg</param-name>
<param-value>MCA</param-value></init-param>
<init-param><param-name>add</param-name>
<param-value>Rohini</param-value></init-param>
<init-param><param-name>phNo</param-name>
<param-value>9013278579</param-value></init-param>
<servlet-name>GetInitParameter</servlet-name>
<servlet-class>GetInitParameter</servlet-class></servlet>
<servlet-mapping>
<servlet-name>GetInitParameter</servlet-name>
<url-pattern>/GetInitParameter</url-pattern>
```



```
A Student have the following record :
prg phNo name enrNo add

Enr. No.    : 102569638
Name        : Bipul
Programme   : MCA
Aaddress    : Rohini
Phone no    : 9013278579
```

GenericServlet class

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces.

It provides the implementaion of all the methods of these interfaces except the service method.GenericServlet class can handle any type of request so it is protocol-independent.

Methods of GenericServlet class

Method	Description
public void init(ServletConfig config)	To initialize the servlet.
public abstract void service(ServletRequest request, ServletResponse response)	Provides service for the incoming request. It is invoked at each time when user requests for a servlet.
public void destroy()	Invoked only once throughout the life cycle and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	Returns the object of ServletConfig.
public String getServletInfo()	Returns information about servlet such as writer, copyright, version etc.
public void init()	It is a convenient method for the servlet programmers, now there is no need to call super.init(config)
public ServletContext getServletContext()	Returns the object of ServletContext.
public String getInitParameter(String name)	Returns the parameter value for the given parameter name.
public Enumeration getInitParameterNames()	Returns all the parameters defined in the web.xml file.
public String getServletName()	Returns the name of the servlet object.

Gereric servlet

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

public class First extends GenericServlet{

public void service(ServletRequest req,ServletResponse res) throws IOException,
        ServletException{

res.setContentType("text/html");

PrintWriter out=res.getWriter();out.print("<html><body>");

out.print("<b>hello generic servlet</b>");out.print("</body></html>");}}
```

HttpServlet Class

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

Methods of HttpServlet class

Method	Description
public void service(ServletRequest req,ServletResponse res)	dispatches the request to the protected service method by converting the request and response object into http type.
protected void service(HttpServletRequest req, HttpServletResponse res)	receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
protected void doGet(HttpServletRequest req, HttpServletResponse res)	handles the GET request. It is invoked by the web container.
protected void doPost(HttpServletRequest req, HttpServletResponse res)	handles the POST request. It is invoked by the web container.
protected void doHead(HttpServletRequest req, HttpServletResponse res)	handles the HEAD request. It is invoked by the web container.
protected void doOptions(HttpServletRequest req, HttpServletResponse res)	handles the OPTIONS request. It is invoked by the web container.
protected void doPut(HttpServletRequest req, HttpServletResponse res)	handles the PUT request. It is invoked by the web container.

protected void doTrace(HttpServletRequest req, HttpServletResponse res)	handles the TRACE request. It is invoked by the web container.
protected void doDelete(HttpServletRequest req, HttpServletResponse res)	handles the DELETE request. It is invoked by the web container.
protected long getLastModified(HttpServletRequest req)	returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

HTTP servlet

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  private String message;
 public void init() throws ServletException
 {      message = "Hello World"; }
 publicvoiddoGet(HttpServletRequestrequest, HttpServletResponse response)
          throws ServletException, IOException
 {     response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<h1>" + message +"</h1>"); }
 public void destroy() { }}
```

ServletRequest Interface

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

Methods of ServletRequest interface

Method	Description
public String getParameter(String name)	Usedto obtain the value of a parameter by name.
public String[] getParameterValues(String name)	Returnsan array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.

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

ServletRequest to display the name of the user

index.html

```
<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login"></form>
<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login"></form>
```

DemoServ.java

```
import javax.servlet.http.*; import javax.servlet.*; import java.io.*;
public class DemoServ extends HttpServlet{
```

```
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{ res.setContentType("text/html"); PrintWriter pw=res.getWriter();
String name=req.getParameter("name");//will return value
pw.println("Welcome"+name); pw.close(); }}
```

FORM GET AND POST ACTIONS

The browser uses two methods to pass this information to web server. These methods are GET () and POST ().

GET(): The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the '?' character.

Example: http://www.test.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from browser to web server.GET method should be avoided while passing password or other sensitive information to the server.The GET method can hold only 1024 characters in a request string.This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet() method.

Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation:

1. getParameter()-to get the value of a form parameter.
2. getParameterValues()-this method is used if the parameter appears more than once and returns multiple values(example checkbox).
3. getParameterNames()-this method is used when a complete list of all parameters is needed in the current request.

Post()

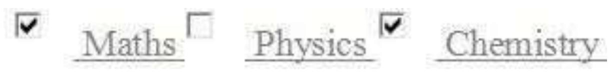
Post() is a reliable method of passing information to a backend program.This handles the information in exactly the same way as GET methods, but instead of sending it as a text string after a ?in the URL, it sends it as a separate message. This message comes to the backend program in the form of the standard input which can be parsed and processed. Servlet handles this type of requests using doPost() method.

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

```
public  class   ReadParams   extends HttpServlet  {     // Method to handle GET method
request.
  publicvoiddoGet(HttpServletRequestrequest, HttpServletResponseresponse)
            throws ServletException, IOException
  {
      response.setContentType("text/html");       PrintWriter out = response.getWriter();
  String title = "Reading All Form Parameters";
         out.println( "<html>\n" + "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor=\"#f0f0f0\">\n"  +  "<h1  align=\"center\">"  +  title  +  "</h1>\n"
+"<table      width=\"100%\"      border=\"1\"      align=\"center\">\n"      +      "<tr
bgcolor=\"#949494\">\n"    +"<th>Param    Name</th><th>Param    Value(s)</th>\n"+
"</tr>\n");
      Enumeration paramNames = request.getParameterNames();
     while(paramNames.hasMoreElements())             {
         String paramName = (String)paramNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n<td>");
         String[] paramValues= request.getParameterValues(paramName);
         if (paramValues.length== 1)          {
           String paramValue = paramValues[0];
           if  (paramValue.length() ==  0)            out.println("<i>No Value</i>");
            else            out.println(paramValue);         }
        else {            // Read multiple valued data
             out.println("<ul>");
              for(int i=0; i < paramValues.length; i++) {
                  out.println("<li>" +paramValues[i]);
             }            out.println("</ul>");         }      }
        out.println("</tr>\n</table>\n</body></html>");
  } // Method to handle POST method request.
  publicvoiddoPost(HttpServletRequestrequest, HttpServletResponse response)
      throws ServletException, IOException
 {     doGet(request, response);  }}
```

Form.html

```
<html><body>
<form action="ReadParams" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics" /> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<input type="submit" value="Select Subject" />
</form></body></html>
```



Servlet output:

Param Name	Param Value(s)
maths	on
Chemistry	on

In the above example, the subjects checked in form.html is retrieved in the servlet code using get() and post(). The getParameterValues() returns the values of more than one parameter and is a enumeration type. The hasMoreElements() checks whether there are more parameters to be passed and the nextElement() fetches the next parameter value.

SESSION HANDLING

Session simply means a particular interval of time.Session Tracking is a way to maintain state (data) of a user. It is also known as **session management** in servlet.Http is a stateless protocol that means each request is considered as the new request. So the states are maintained using various 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 a user to recognize to particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:Cookies, Hidden Form Field, URL Rewriting and HttpSession Interface

Cookies

A cookie is a small piece of information that persist between the multiple client requests.In HTTP each request is considered as a new request even if two requests are issued by the same user.A cookie is added with response from the servlet which serves as an identifier to the user. This is done when the first request is made.So cookie is stored in the cache of the browser. After that if any request is sent by the user, the stored cookie is added with request by default. Thus, the server recognizes the user as the old user.

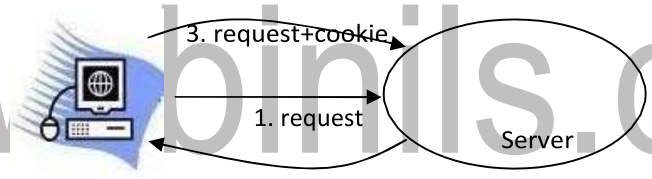


Figure 3.6 Cookies

Differences between Cookie and Session

Session	Cookie
Sessions are **server-side** files that contain user information	Cookies are **client-side** files that contain user information
Any amount of data can be stored within sessions.	Official MAX Cookie size is 4KB
Session ends when user close his browser.	Cookie ends depends on the life time you setfor it.

Hidden Form field

Here, a hidden (invisible) textfield is used for maintaining the state of end user. In such case, the information is stored in the hidden field. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

<input type="hidden" name=”Field name" value="value">

Example: <INPUT TYPE="HIDDEN" NAME="session" VALUE="a1234">

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data.

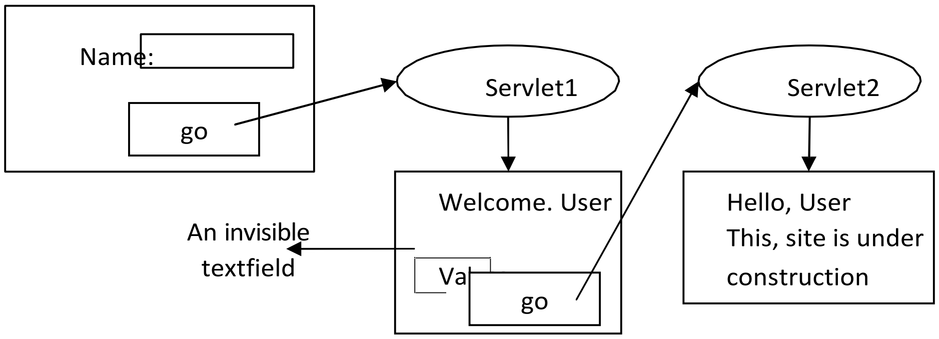


Figure 3.7 Hidden Form field

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.

URL Rewriting

Appending the name of the user in the query string and getting the value from the query string in another page is called URL rewriting. The parameter **name-value pairs** are passed 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 using the ampersand(&). With URL rewriting, every local URL the user might click on is dynamically modified, or rewritten, to include extra information. The extra information can be in the form of extra path information, added parameters, or some custom, server-specific URL change.

Due to the limited space available in rewriting a URL, the extra information is usually limited to a unique session ID. When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a, the getParameter() method to obtain a parameter value.

http://server:port/servlet/Rewritten	Original URL
http://server:port/servlet/Rewritten/123	extra path information
http://server:port/servlet/Rewritten?sessionid=123	added parameter
http://server:port/servlet/Rewritten;$sessionid$123	custom change

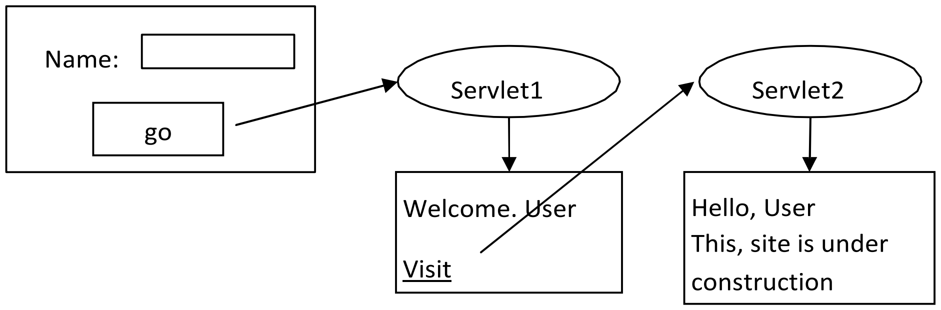


Figure 3.7(a) URL rewriting

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.

HTTP Session Interface

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

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

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

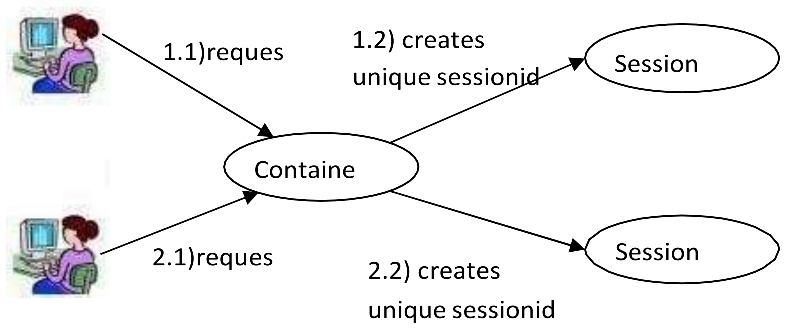


Figure 3.8 Creating session

Methods of HttpSession interface

Method	Description
public Object getAttribute(String name)	returns the object bound with the specified name in this session, or null if no object is bound under the name.
public Enumeration getAttributeNames()	returns an Enumeration of String objects containing the names of all the objects bound to this session.
public long getCreationTime()	returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public String getId()	returns a string containing the unique identifier assigned to this session.
public long getLastAccessedTime()	returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
public int getMaxInactiveInterval()	returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
public void invalidate()	invalidates this session and unbinds any objects bound to it.
public boolean isNew()	returns true if the client does not yet know about the session or if the client chooses not to join the session.
public void removeAttribute(String name)	removes the object bound with the specified name from this session.
public void setAttribute(String name, Object value)	binds an object to this session, using the name specified.
public void setMaxInactiveInterval(int interval)	specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Session tracking

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
import java.util.*;
 public class SessionTrack extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
          throws ServletException, IOException
 {      HttpSession session = request.getSession(true); //session id is created
    Date createTime = new Date(session.getCreationTime()); // Get session creation time.
    Date lastAccessTime = new Date(session.getLastAccessedTime());
    // Get last access time of this web page.
    String title = "Welcome Back to my website";
    Integer visitCount = new Integer(0);
    String visitCountKey = new String("visitCount");
    String userIDKey  =  new String("userID");
    String userID = new String("ABCD");
    // Check if this is new comer on your web page.
    if  (session.isNew()){        title = "Welcome to my website";
      session.setAttribute(userIDKey, userID);
    } else {
      visitCount = (Integer)session.getAttribute(visitCountKey);
      visitCount = visitCount + 1;
      userID = (String)session.getAttribute(userIDKey);
    }
    session.setAttribute(visitCountKey, visitCount);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>\n" + "<head><title>" + title + "</title></head>\n" +
            "<body   bgcolor=\"#f0f0f0\">\n"   +      "<h1 align=\"center\">" + title +
"</h1>\n" +
             "<h2 align=\"center\">Session Infomation</h2>\n" +
            "<table border=\"1\" align=\"center\">\n" + "<tr bgcolor=\"#949494\">\n"  +
            " <th>Session info</th><th>value</th></tr>\n" + "<tr>\n"+
            " <td>id</td>\n" + " <td>" + session.getId() + "</td></tr>\n" +
            "<tr>\n" + " <td>Creation Time</td>\n" + "  <td>"  + createTime +
            " </td></tr>\n" + "<tr>\n" +” <td>Time of Last Access</td>\n"  +
            "   <td>"  + lastAccessTime +    " </td></tr>\n"  +    "<tr>\n" + "  <td>User
ID</td>\n" +
            " <td>" + userID + " </td></tr>\n" + "<tr>\n" + " <td>Number of
 visits</td>\n" +
            "     <td>"    +    visitCount    +    "</td></tr>\n"    +"</table>\n"    +
 "</body></html>");
  }}
```

Welcome to my website

Session Information

Session info	Value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

info type	Value
Id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	1