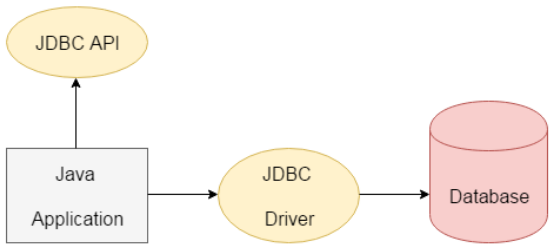
UNIT V: JDBC AND SERVLET

JDBC: JDBC Architecture - JDBC Drivers- Database connectivity in Java- HTML basics - JavaScript basics.
Servlets: Servlet Architecture - Life cycle of a Servlet - The Servlet API- Handling HTTP Request and Response- Cookies - Session Tra
cking-Overview of JSP.

JDBC - Java Database Connectivity

The JDBC stands for Java Database Connectivity. The JDBC project was started in January of 1996, and the specification was frozen in June of 1996. The JDBC is a package, much like other Java packages such as java.awt. It’s not currently a part of the standard Java Developer’s Kit (JDK) distribution, but it is slated to be included as a standard part of the general Java API as the java.sql package.

JDBC provides a common database programming API for Java programs. However, JDBC drivers do not directly communicate with as many database products as ODBC drivers. Instead, many JDBC drivers communicate with database using ODBC. In fact, one of first JDBC drivers was the JDBC -ODBC bridge driver developed by JavaSoft. ODBC is a C language API.
Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



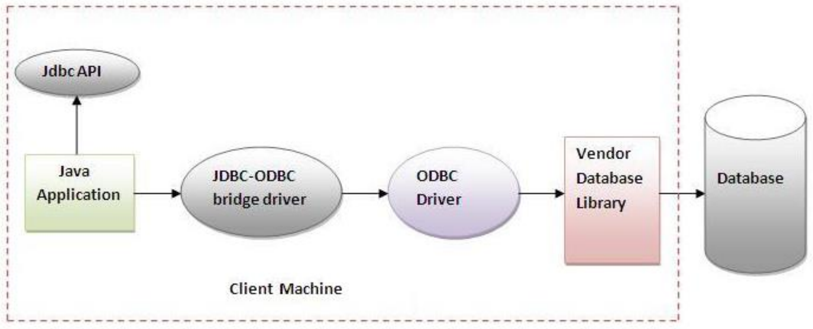
JDBC Driver

JDBC Driver is a software component that enables java application to interact with the database.Thereare 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.



Advantages:

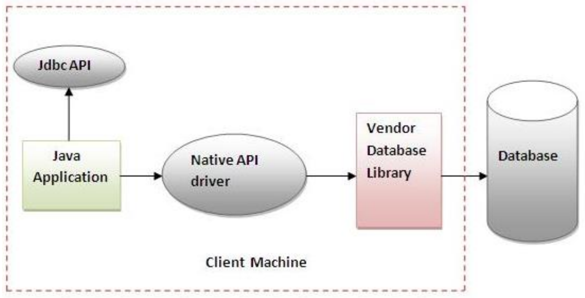
- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.



Advantage:

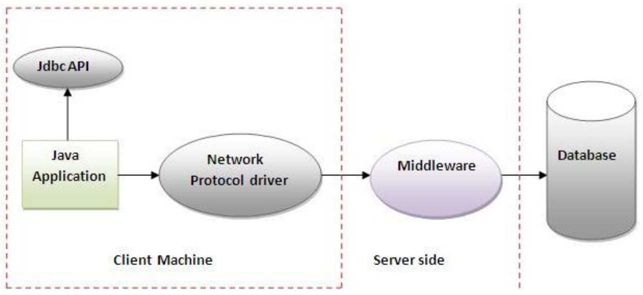
- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



Advantage:

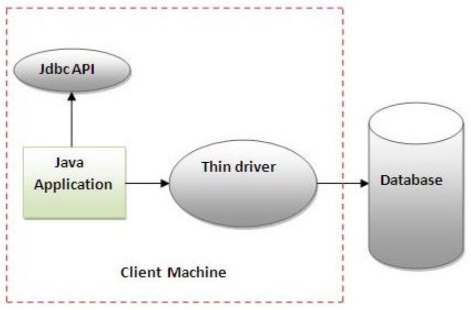
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.



Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depends on the Database.

Database Connectivity in Java

A database is an organized collection of data.A database management system(DBMS) provides mechanisms for storing, organizing, retrieving and modifying data formany users. Database management systems allow for the access and storage of data withoutconcern for the internal representation of data.
Java programs communicate with databases and manipulate their data using the JavaDatabase Connectivity (JDBC™) API. A JDBC driver enables Java applications to connectto a database in a particular DBMS and allows you to manipulate that database usingthe JDBC API.

5 Steps to connect to the database in java

- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

1) Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

public static void forName(String className)**throws** ClassNotFoundException

Example to register the OracleDriver class

Class.forName("oracle.jdbc.driver.OracleDriver");

2) Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

public static Connection getConnection(String url)**throws** SQLException

public static Connection getConnection(String url,String name,String password)
throws SQLException

Example to establish connection with the Oracle database

```
Connection con=DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe","
system","password");
```

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
public Statement createStatement()throws SQLException
```

Example to create the statement object

```
Statement stmt=con.createStatement();
```

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method:

```
public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
    ResultSet rs=stmt.executeQuery("select * from emp");
    while(rs.next()){
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5)Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method:

```
public void close()throws SQLException
```

Example to close connection

```
con.close();
```

Simple Program to connect to Access Database for Java 8 or above

Note:

Java version after 8 will not support forName() function so need to download the following jar files and place in the lib path of Java Directory

1) commons-lang-2.5
2) commons-logging-1.1.2

3) hsqldb-2.2.8
4) jackcess-2.0.0
5) ucanaccess-3.0.6
   Add these JAR files in Java→ JRE→LIB→EXT folder

Things needed:

1) Access Database Inventory.accdb and table Stud in it saved in path of D:

Program: DatabaseConnection.java

```
importjava.sql.Connection;
importjava.sql.DriverManager;
importjava.sql.ResultSet;
importjava.sql.SQLException;
importjava.sql.Statement;
public class DatabaseConnection {
public static void main(String[] args) {
        // variables
        Connection connection = null;
        Statement statement = null;
ResultSetresultSet = null;
//this step is needed for Java below 8 version
        // Step 1: Loading or registering Oracle JDBC driver class
    /*   try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        }
catch(ClassNotFoundExceptioncnfex) {
System.out.println("Problem in loading MS Access JDBC driver");
cnfex.printStackTrace();
        }*/
        // Step 2: Opening database connection trying with ucanaccess jar for Access Database
try {
            String dbURL = "jdbc:ucanaccess://D:\\Inventory.accdb";
            // Step 2.A: Create and get connection using DriverManager class
connection = (Connection)DriverManager.getConnection(dbURL);
            // Step 2.B: Creating JDBC Statement
statement = connection.createStatement();
            // Step 2.C: Executing SQL &amp; retrieve data into ResultSet
resultSet = statement.executeQuery("SELECT * FROM STUD");
System.out.println("Name\t Age");
```

```
            // processing returned data and printing into console
    while(resultSet.next())
            {
    System.out.println(resultSet.getString(1) + "\t" + resultSet.getString(2));
            }
        }
    catch(SQLExceptionsqlex)
            {
    sqlex.printStackTrace();
        }
    finally
            {
             // Step 3: Closing database connection
    try {
    if(null != connection)
            {
                    // cleanup resources, once after processing
    resultSet.close();
    statement.close();
                    // and then finally close connection
    connection.close();
                }
            }
    catch (SQLExceptionsqlex)
            {
    sqlex.printStackTrace();
            }
        }
      }
    }
```

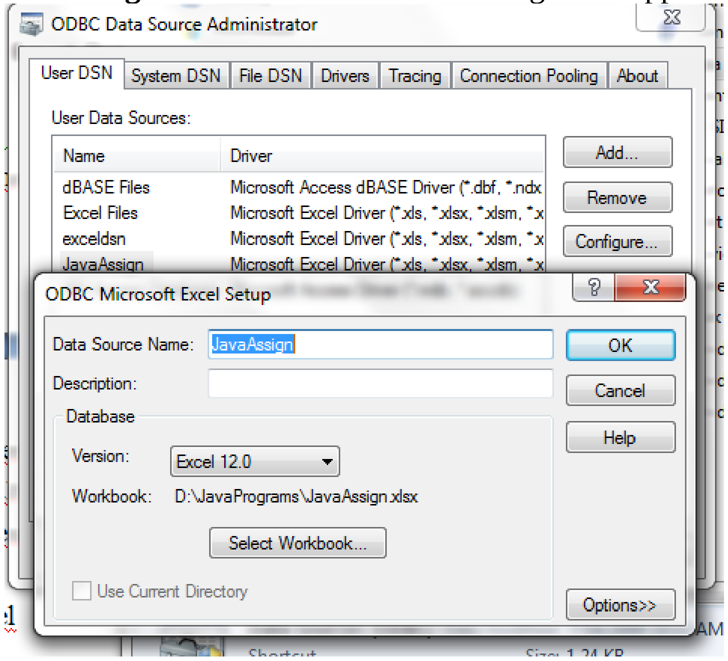
OUTPUT:

```
D:\JavaPrograms>javac DatabaseConnection.java
D:\JavaPrograms>java DatabaseConnection
Name    Age
Kumar   30
Gopal33
```

RajKiran 22

Example 2: To connect Excel Database

Steps: Create a DSN : **JavaAssign** and connect it to D:\JavaPrograms\supplier.xls file



User Data Sources:

Name	Driver
dBASE Files	Microsoft Access dBASE Driver (*.dbf, *.ndx
Excel Files	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.x
exceldsn	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.x
JavaAssign	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.x

Data Source Name: JavaAssign

Description:

Database

Version: Excel 12.0

Workbook: D:\JavaPrograms\JavaAssign.xlsx

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
 import java.sql.*;
  public class Dbexcel
  {
    static
    {
       try  {
           Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
       }
       catch (Exception e) {
           System.err.println(e);
       }
    }
    public static void main(String args[]) {
       Connection conn=null;
```

```
        Statement stmt=null;
        String sql="";
        ResultSet rs=null;
        try {
            conn=DriverManager.getConnection("jdbc:odbc:JavaAssign","","");
            stmt=conn.createStatement();
            sql="select * from [supplier$]";
            rs=stmt.executeQuery(sql);

            while(rs.next()){
                System.out.println(rs.getString("sno")+" "+ rs.getString("sname"));
            }
        }
        catch (Exception e){
            System.err.println(e);
        }
        finally {
            try{
                rs.close();
                stmt.close();
                conn.close();
                rs=null;
                stmt=null;
                conn=null;
            }
            catch(Exception e){}
        }
    }
}
```

OUTPUT:

```
D:\JavaPrograms>javac Dbexcel.java
D:\JavaPrograms>java Dbexcel
S1          Hyndai
S2          Ford
S3          Tata
```

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

1. create database sonoo;
2. use sonoo;
3. create table emp(id int(10),name varchar(40),age int(3));

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password both.

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

The above example will fetch all the records of emp table.

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behavior of web pages

Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

JavaScript Example

1. JavaScript Example
2. Within body tag
3. Within head tag

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let's create the first JavaScript example.

```
<script type="text/javascript">
```

```
document.write("JavaScript is a simple language for javatpoint learners");
</script>
```

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

---

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

---

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let’s see the simple example of JavaScript that displays alert dialog box.

```
<script type="text/javascript">
 alert("Hello Javatpoint");
</script>
```

---

2) JavaScript Example : code between the head tag

Let’s see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.

To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
<html>
<head>
<script type="text/javascript">
```

```
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

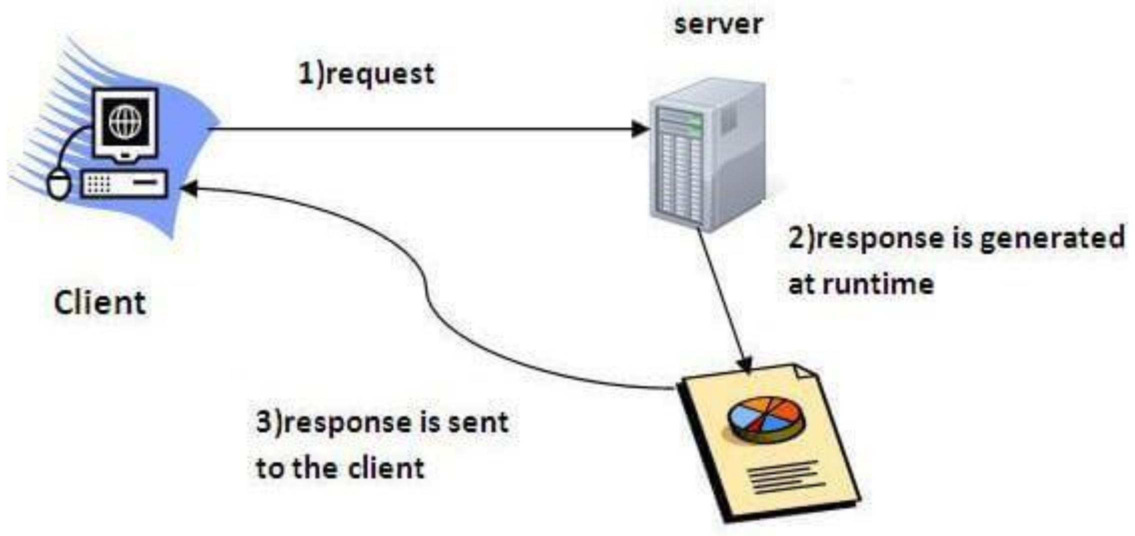
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.


server
1)request
2)response is generated at runtime
Client
3)response is sent to the client

What is the web application and what is the difference between Get and Post request?

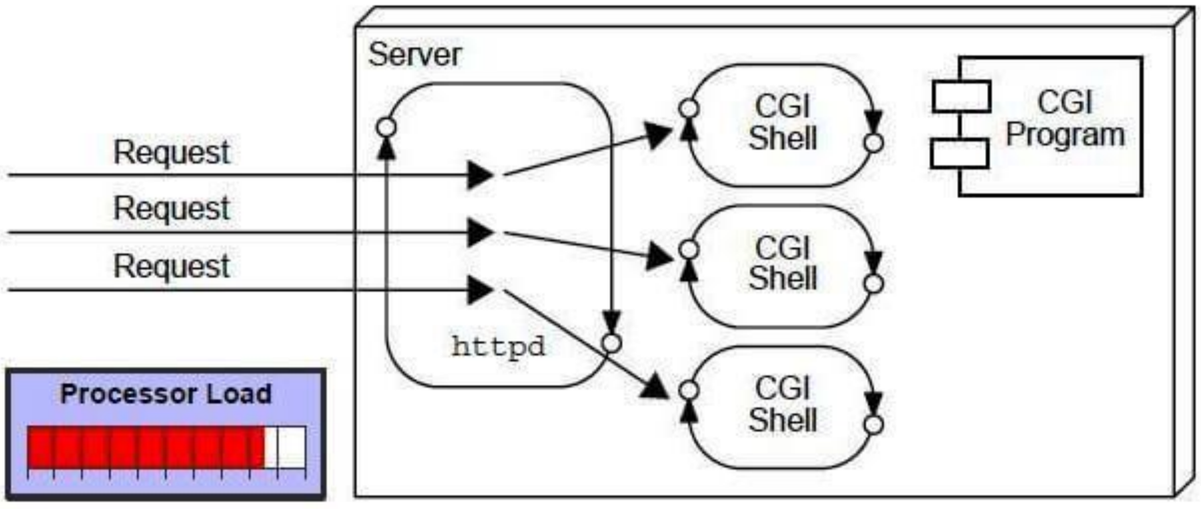
- What information is received by the web server if we request for a Servlet?
- How to run servlet in Eclipse, MyEclipse and Netbeans IDE?
- What are the ways for servlet collaboration and what is the difference between RequestDispatcher and sendRedirect() method?
- What is the difference between ServletConfig and ServletContext interface?
- How many ways can we maintain the state of a user? Which approach is mostly used in web development?
- How to count the total number of visitors and whole response time for a request using Filter?
- How to run servlet with annotation?
- How to create registration form using Servlet and Oracle database?
- How can we upload and download the file from the server?

What is a web application?

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

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

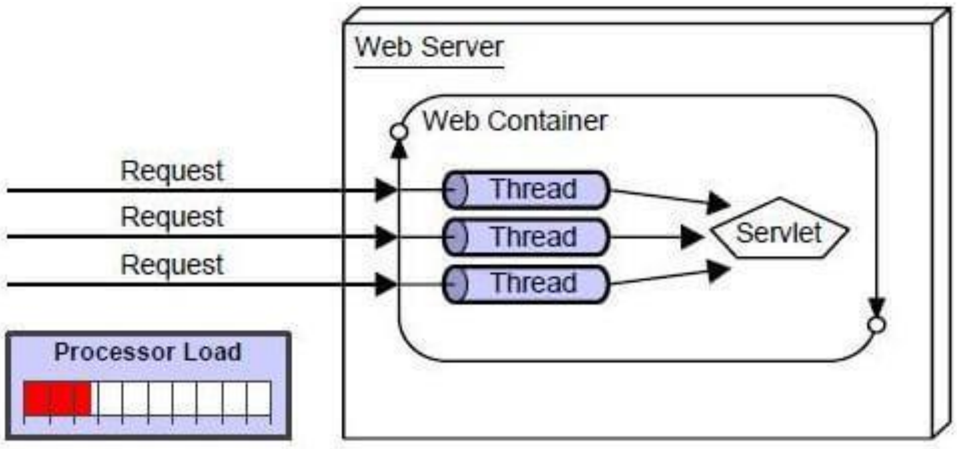


Disadvantages of CGI

There are many problems in CGI technology:

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

Advantages of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Properties of Servlets :

- Servlets work on the server-side.
- Servlets capable of handling complex request obtained from web server.

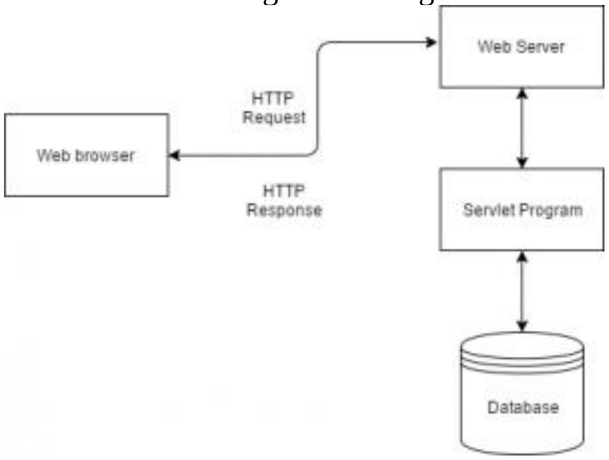
Execution of Servlets :

Execution of Servlets involves the six basic steps:

1. The clients send the request to the web server.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generate the response in the form of output.
5. The servlet send the response back to the web server.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Servlet Architecture

The following diagram shows the servlet architecture:



Need For Server-Side Extensions

The **server-side extensions** are nothing but the technologies that are used to create dynamic Web pages. Actually, to provide the facility of dynamic Web pages, Web pages need a container or Web server. To meet this requirement, independent Web server providers offer some proprietary solutions in the form of **APIs**(Application Programming Interface). These **APIs** allow us to build programs that can run with a Web server. In this case , **Java**

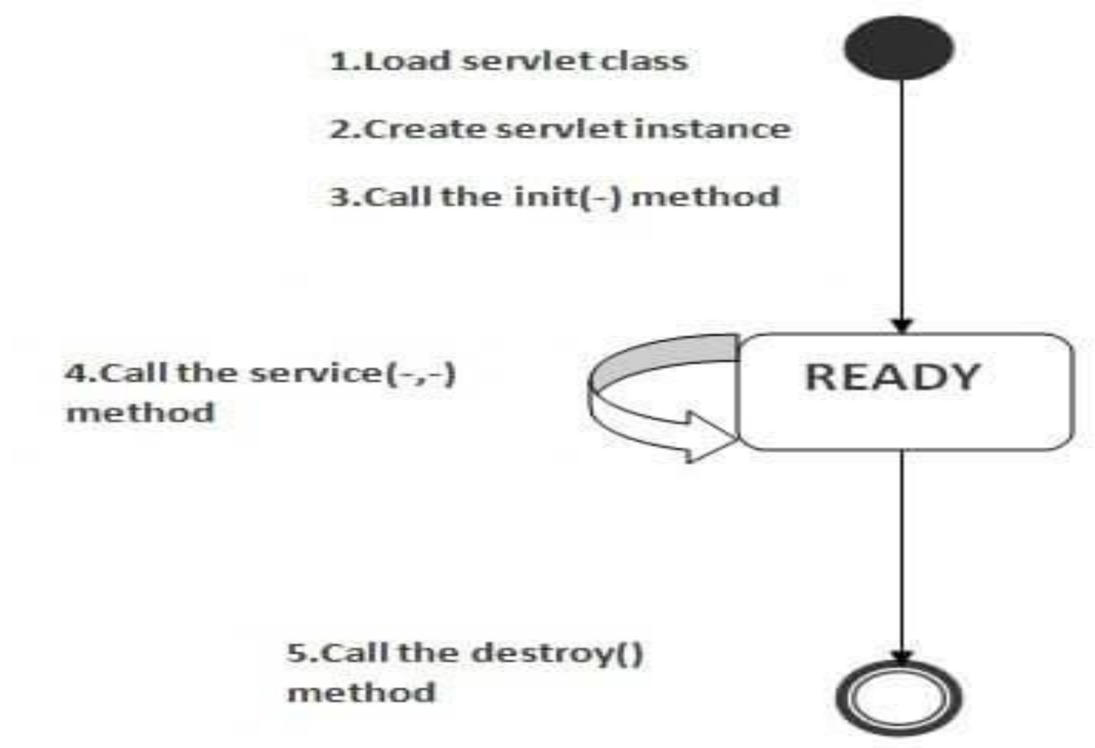
Servlet is also one of the component APIs of **Java Platform Enterprise Edition** which sets standards for creating dynamic Web applications in Java.
Before learning about something its important to know the need for that something, its not like that this is the only technology available for creating dynamic Web pages. The Servlet technology is similar to other Web server extensions such as **Common Gateway Interface**(CGI) scripts and **Hypertext Preprocessor** (PHP). However, Java Servlets are more acceptable since they solve the limitations of **CGI** such as low performance and low degree scalability.

Life Cycle of a Servlet (Servlet Life Cycle)

1. Life Cycle of a Servlet
1. Servlet class is loaded
2. Servlet instance is created
3. init method is invoked
4. service method is invoked
5. destroy method is invoked

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

As displayed in the above diagram, 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) Servlet class is loaded

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

2) Servlet instance is created

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

3) init method is invoked

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

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

---

4) service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
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()
```

next →← prev

HTTP Requests

The request sent by the computer to a web server, contains all sorts of potentially interesting information; it is known as HTTP requests.

The HTTP client sends the request to the server in the form of request message which includes following information:

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)

- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header

The HTTP request method indicates the method to be performed on the resource identified by the **Requested URI (Uniform Resource Identifier)**. This method is case-sensitive and should be used in uppercase.

The HTTP request methods are:

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET r
HEAD	Asks for only the header part of whatever a GET would return. J
TRACE	Asks for the loopback of the request message, for testing or trou
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the rec

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

Get vs. Post

There are many differences between the Get and Post request. Let's see these differences:

GET and POST

Two common methods for the request-response between a server and client are:

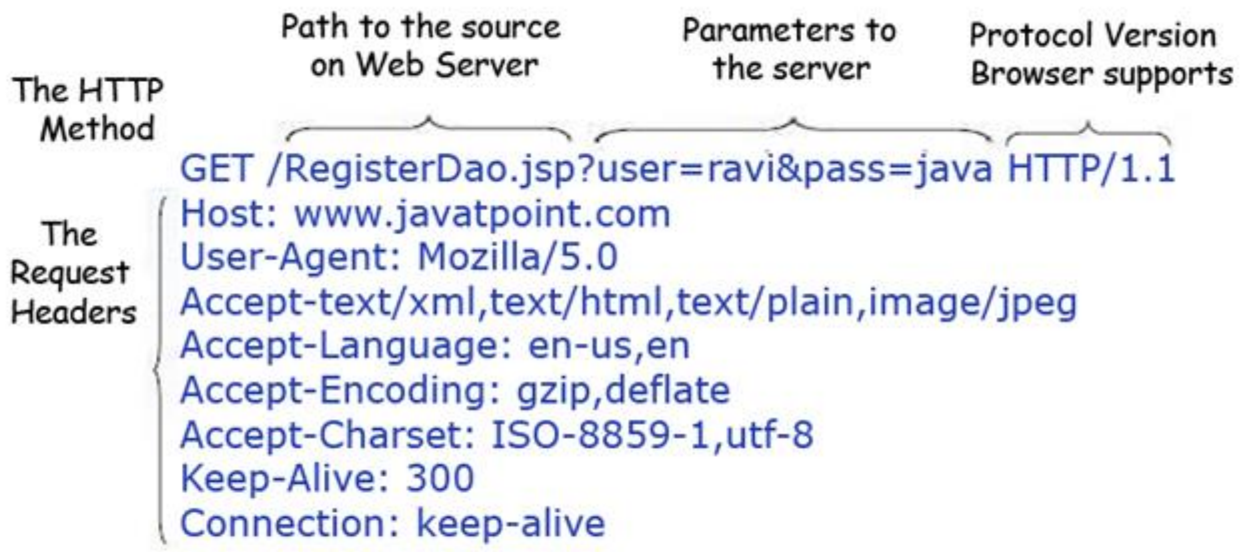
- **GET**- It requests the data from a specified resource
- **POST**- It submits the processed data to a specified resource

Anatomy of Get Request

The query string (name/value pairs) is sent inside the URL of a GET request:

```
GET /RegisterDao.jsp?name1=value1&name2=value2
```

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what information is sent to the server.


The HTTP Method
Path to the source on Web Server
Parameters to the server
Protocol Version Browser supports
The Request Headers

```
GET /RegisterDao.jsp?user=ravi&pass=java HTTP/1.1
Host: www.javatpoint.com
User-Agent: Mozilla/5.0
Accept-text/xml,text/html,text/plain,image/jpeg
Accept-Language: en-us,en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8
Keep-Alive: 300
Connection: keep-alive
```

Some other features of GET requests are:

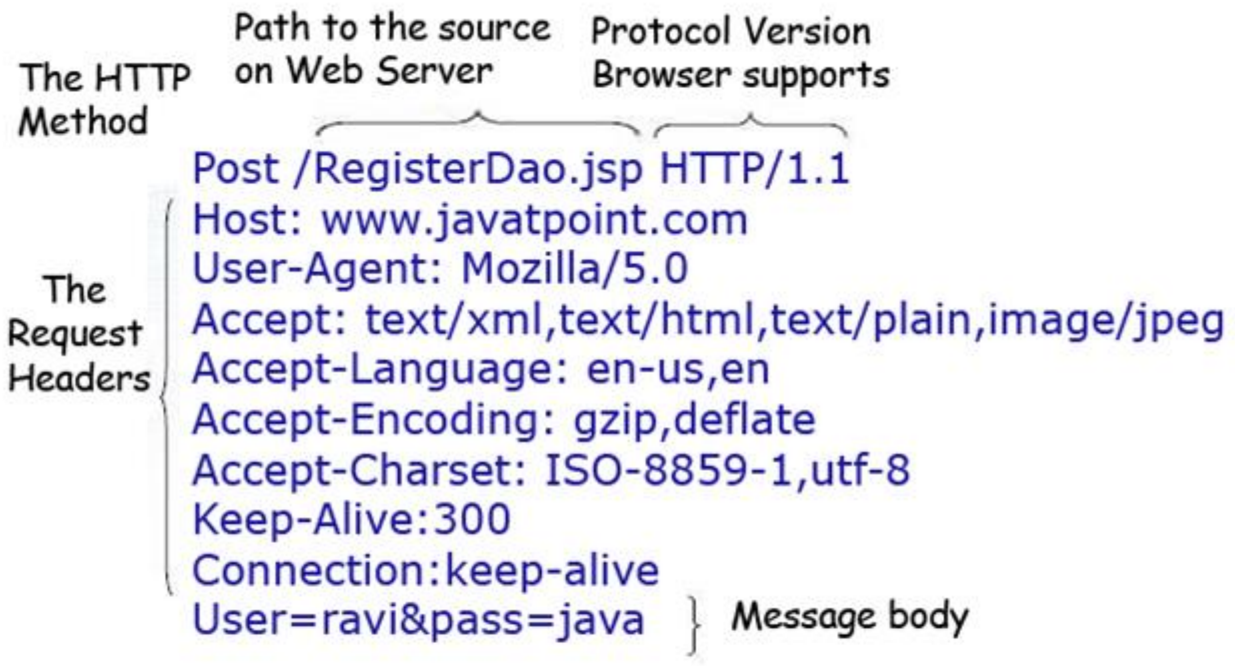
- It remains in the browser history
- It can be bookmarked
- It can be cached
- It have length restrictions
- It should never be used when dealing with sensitive data
- It should only be used for retrieving the data

Anatomy of Post Request

The query string (name/value pairs) is sent in HTTP message body for a POST request:

```
POST/RegisterDao.jsp HTTP/1.1
Host: www. javatpoint.com
name1=value1&name2=value2
```

As we know, in case of post request original data is sent in message body. Let's see how information is passed to the server in case of post request.


The HTTP Method
Path to the source on Web Server
Protocol Version Browser supports
The Request Headers
Message body

```
Post /RegisterDao.jsp HTTP/1.1
Host: www.javatpoint.com
User-Agent: Mozilla/5.0
Accept: text/xml,text/html,text/plain,image/jpeg
Accept-Language: en-us,en
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8
Keep-Alive:300
Connection:keep-alive
User=ravi&pass=java
```

Some other features of POST requests are:

- This requests cannot be bookmarked
- This requests have no restrictions on length of data
- This requests are never cached
- This requests do not retain in the browser history

Cookies

Cookies are text files stored on the client computer and they are kept for various information tracking purpose. Java Servlets transparently supports HTTP cookies.

There are three steps involved in identifying returning users −

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.

- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set or reset cookies, how to access them and how to delete them.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A servlet that sets a cookie might send headers that look something like this −

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name = xyz; expires = Friday, 04-Feb-07 22:03:38 GMT;
   path = /; domain = tutorialspoint.com
Connection: close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this −

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name = xyz
```

A servlet will then have access to the cookie through the request method *request.getCookies()* which returns an array of *Cookie* objects.

Servlet Cookies Methods

Following is the list of useful methods which you can use while manipulating cookies in servlet.

Sr.No.	Method & Description
1	**public void setDomain(String pattern)** This method sets the domain to which cookie applies, for example tutorialspoint.com.
2	**public String getDomain()** This method gets the domain to which cookie applies, for example tutorialspoint.com.
3	**public void setMaxAge(int expiry)** This method sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session.
4	**public int getMaxAge()** This method returns the maximum age of the cookie, specified in seconds, By default, -1 indicating the cookie will persist until browser shutdown.
5	**public String getName()** This method returns the name of the cookie. The name cannot be changed after creation.
6	**public void setValue(String newValue)** This method sets the value associated with the cookie
7	**public String getValue()** This method gets the value associated with the cookie.
8	**public void setPath(String uri)**

	This method sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories.
9	**public String getPath()** This method gets the path to which this cookie applies.
10	**public void setSecure(boolean flag)** This method sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.
11	**public void setComment(String purpose)** This method specifies a comment that describes a cookie's purpose. The comment is useful if the browser presents the cookie to the user.
12	**public String getComment()** This method returns the comment describing the purpose of this cookie, or null if the cookie has no comment.

Setting Cookies with Servlet

Setting cookies with servlet involves three steps −

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

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

Keep in mind, neither the name nor the value should contain white space or any of the following characters −

```
[ ] ( ) = , " / ? @ : ;
```

(2) Setting the maximum age − You use setMaxAge to specify how long (in seconds) the cookie should be valid. Following would set up a cookie for 24 hours.

```
cookie.setMaxAge(60 * 60 * 24);
```

(3) Sending the Cookie into the HTTP response headers – You use response.addCookie to add cookies in the HTTP response header as follows –

```
response.addCookie(cookie);
```

Example

Let us modify our Form Example to set the cookies for first and last name.

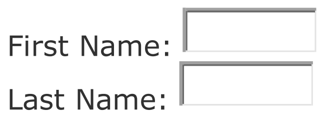
```
// Import required java libraries

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;



// Extend HttpServlet class

public class HelloForm extends HttpServlet {



   public void doGet(HttpServletRequest request, HttpServletResponse response)

      throws ServletException, IOException {



      // Create cookies for first and last names.

      Cookie firstName = new Cookie("first_name",
request.getParameter("first_name"));

      Cookie lastName = new Cookie("last_name", request.getParameter("last_name"));



      // Set expiry date after 24 Hrs for both the cookies.

      firstName.setMaxAge(60*60*24);

      lastName.setMaxAge(60*60*24);



      // Add both the cookies in the response header.

      response.addCookie( firstName );
      response.addCookie( lastName );

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Setting Cookies Example";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

      out.println(docType +
         "<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>"
      );
   }
```

```
}
```

Compile the above servlet **HelloForm** and create appropriate entry in web.xml file and finally try following HTML page to call servlet.

```
<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>
```

Keep above HTML content in a file Hello.htm and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access *http://localhost:8080/Hello.htm*, here is the actual output of the above form.



First Name:

Last Name:

Try to enter First Name and Last Name and then click submit button. This would display first name and last name on your screen and same time it would set two cookies firstName and lastName which would be passed back to the server when next time you would press Submit button.

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and web server −

Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the recieved cookie.

This may not be an effective way because many time browser does not support a cookie, so I would not recommend to use this procedure to maintain the sessions.

Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows −

```
<input type = "hidden" name = "sessionid" value = "12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

This could be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

URL Rewriting

You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.

For example, with http://tutorialspoint.com/file.htm;sessionid = 12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and it works even when browsers don't support cookies. The drawback of URL re-writing is that you would have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

The HttpSession Object

Apart from the above mentioned three ways, servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.

You would get HttpSession object by calling the public method **getSession()**of HttpServletRequest, as below −

```
HttpSession session = request.getSession();
```

You need to call *request.getSession()* before you send any document content to the client. Here is a summary of the important methods available through HttpSession object −

Sr.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.

4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last accessed time of the session, in the format of milliseconds since midnight January 1, 1970 GMT
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval (seconds), that the servlet container will keep the session open between client accesses.
7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public boolean isNew(** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
// Import required java libraries

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

import java.util.*;


// Extend HttpServlet class

public class SessionTrack extends HttpServlet {


   public void doGet(HttpServletRequest request, HttpServletResponse response)

      throws ServletException, IOException {


      // Create a session object if it is already not  created.

      HttpSession session = request.getSession(true);


      // Get session creation time.

      Date createTime = new Date(session.getCreationTime());


      // Get last access time of this web page.

      Date lastAccessTime = new Date(session.getLastAccessedTime());


      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);

      // Set response content type
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";

      out.println(docType +
         "<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>"
      );
   }
}
```

Compile the above servlet **SessionTrack** and create appropriate entry in web.xml file. Now running *http://localhost:8080/SessionTrack* would display the following result when you would run for the first time −

```
Welcome to my website

Session Infomation
```

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

Now try to run the same servlet for second time, it would display following result.

```
Welcome Back to my website

Session Infomation
```

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

Deleting Session Data

When you are done with a user's session data, you have several options −

- **Remove a particular attribute** − You can call *public void removeAttribute(String name)* method to delete the value associated with a particular key.
- **Delete the whole session** − You can call *public void invalidate()*method to discard an entire session.
- **Setting Session timeout** − You can call *public void setMaxInactiveInterval(int interval)* method to set the timeout for a session individually.

- **Log the user out** – The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.
- **web.xml Configuration** – If you are using Tomcat, apart from the above mentioned methods, you can configure session time out in web.xml file as follows.

```
<session-config>

   <session-timeout>15</session-timeout>

</session-config>
```

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.

The getMaxInactiveInterval( ) method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, getMaxInactiveInterval( ) returns 900.

What is JavaServer Pages?

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

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.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the **Common Gateway Interface (CGI)**. But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are 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.

Finally, 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

Following table lists out the other advantages of using JSP over other technologies −

vs. Active Server Pages (ASP)

The advantages of JSP are twofold. First, the dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use. Second, it is portable to other operating systems and non-Microsoft Web servers.

vs. Pure Servlets

It is more convenient to write (and to modify!) regular HTML than to have plenty of println statements that generate the HTML.

vs. Server-Side Includes (SSI)

SSI is really only intended for simple inclusions, not for "real" programs that use form data, make database connections, and the like.

vs. JavaScript

JavaScript can generate HTML dynamically on the client but can hardly interact with the web server to perform complex tasks like database access and image processing etc.

vs. Static HTML

Regular HTML, of course, cannot contain dynamic information.