

Advjava

By
Mr.Ratan

JDBC

JDBC(java to database connectivity):-

- 1) The process of connecting the java application to the particular data base is called java to database connectivity.
- 2) The main purpose of the JDBC is to send the details to the database by using java code.
- 3) Whenever we are executing java application automatically the operations are performed in the database.
- 4) **Jdbc is a technology that enables the java program to manipulate the data stored in the data base.**
- 5) Jdbc is designed only for the java applications it doesn't give support for any other technology.
- 6) The present version of jdbc is JDBC 4.2

Database :-The main purpose of the database is to store the data.

Ex:- Oracle,MySql.....etc

Frequently used Queries in JDBC:-

- 1) create table emp(eno number,ename varchar2(20),esal number);
- 2) insert into emp values(100,'rattaiah',25000);
- 3) select * from emp;
- 4) update emp set esal=esal+2000
- 5) drop table emp;

Note: - in java the the program execution starts from main method called by JVM but in Oracle the database queries are executed by DBE(Database Engine)

For each and every query the database engine perform following steps.

- 1) **Query Tokenization :-**
The query is divided into number of small tokens.
- 2) **Query Parsing:-**
The tokens are arranged in the form of tree structure is called query parsing.
- 3) **Query Optimization:-**
After parsing we can optimize the query to improve execution speed.
- 4) **Query Execution:-**
After optimization we can execute the query and see the output.

Parts of the java language:-

As per the **sun micro system** standard the java language is divided into three types.

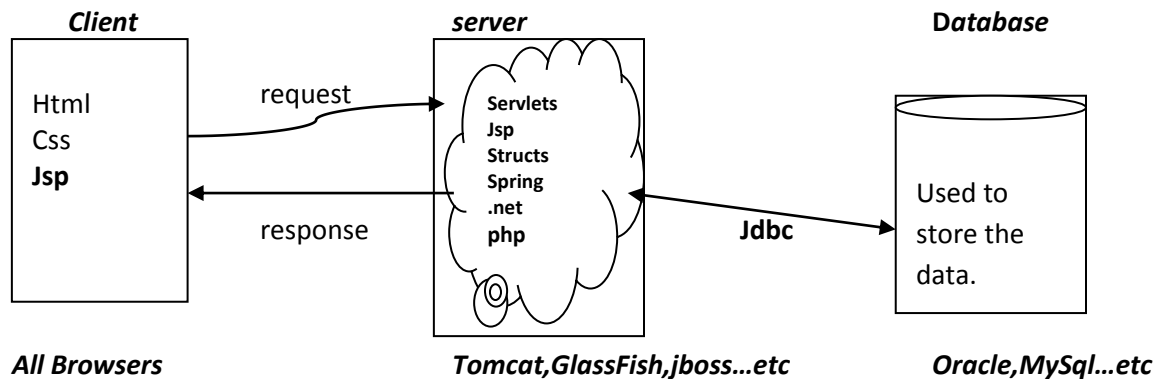
- 1) J2SE/JSE(java 2 standard edition)
- 2) J2EE/JEE(java 2 enterprise edition)
- 3) J2ME/JME(java 2 micro edition)

Adv java Syllabus:-

J2se --->JDBC

J2ee --->Servlets

J2ee --->Jsp

Web-application architecture:-**Client:-**

Who is sending the request and who holds the response is called client.

Ex:- Internet Explorer, Mozilla Firefox, opera.....etc

Server:-

The server contains the applications. The main purpose of the server is

- It will contain the application.
- Take the request from the client.
- Based on the taken request it will identify the project resource and execute that project resource.
- By executing the project some response will be generated that response is dispatched to the client browser.

Ex:- Tomcat, GlassFish, WebLogic, JBOSS, WebSphere.....etc

DataBase:-

Database is used to store the details like client details, application details, registration details.....etc.

Ex:- Oracle, MySql.....etc

Deployment:- The process of placing project into server is called deployment.

Client side technologies:-

The technologies which are used to write the programming at client side is called client side technologies.

Ex :- jsp,html,css,...etc

Server side technologies:-

The technologies which are used to provide the programming at server side is called server side technologies.

Ex:- servlets , jsp , struts , spring....etc

Key interfaces of JDBC:-

1. Connection
2. Statement
3. PreparedStatement
4. CallableStatement
5. ResultSet
6. ResultSetMetaData

7. DatabaseMetaData

8. SavePoint

Note :- All the JDBC main classes and interfaces present in **java.sql** package & in jdbc we are using **javax.sql** package for advanced features like connection pooling.

Servers vs vendor:-

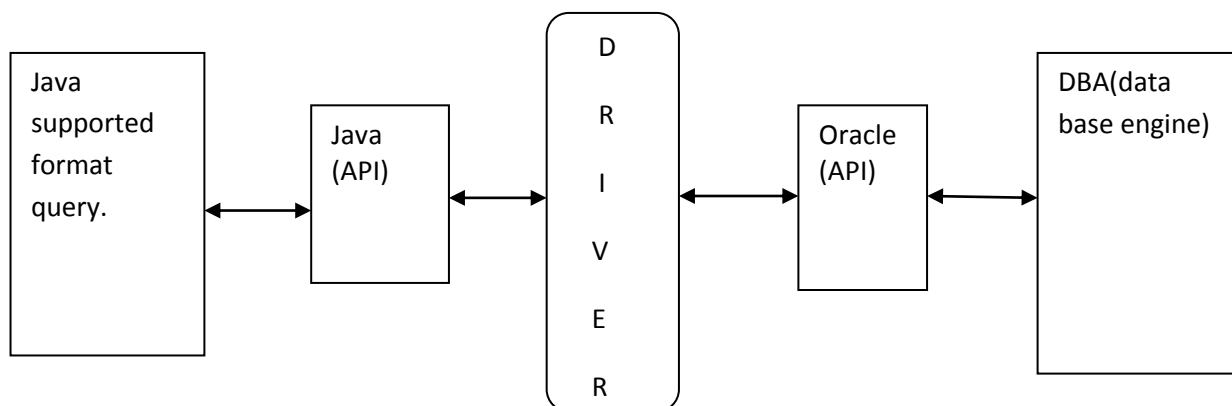
- | | |
|--------------|----------------------------|
| 1) Tomcat | Apache software foundation |
| 2) Glassfish | Sun Micro Systems. |
| 3) JBOSS | jboss community |
| 4) Weblogic | Oracle |
| 5) Websphere | IBM.....etc |

Different type of logics in web application:-

- 1) **presentation layer.**
- 2) **Requested data gathering logics.**
- 3) **Validation logics.**
- 4) **Business logics/service layer**
- 5) **Persistance logic.**
- 6) **Session management logics.**
- 7) **Middleware logics.**

Driver:-

- 1) Driver is interface between the java application and the particular database.
- 2) Driver is acting as a translator between the java application and particular database.
- 3) The driver is converting java formatted SQL queries into the Database supported SQL queries.
- 4) Whenever we are working with JDBC application the first step is load and register the particular driver.
- 5) Number of drivers is available in the market and each and every driver contains its own driver class name and url pattern.



Types of the drivers:-

- | | |
|------------------|---------------------------|
| 1) Type-1 driver | (JDBC-ODBC Bridge Driver) |
| 2) Type-2 driver | (Native API driver) |
| 3) Type-3 driver | (Network protocol Driver) |
| 4) Type-4 driver | (Thin Driver) |

Drivers in JDBC:-

- To use the jdbc Driver in java application we must register that driver into Driver manager service .

- DriverManager service is built in service to manage set of jdbc drivers.

Use the following code to register jdbc driver into DriverManager service.

Step1:-creating driver Object

sun.jdbc.odbc.JdbcOdbcDriver obj = new sun.jdbc.odbc.JdbcOdbcDriver();

step2:registering driver into driver manager service.

DriverManager.registerDriver(obj);

- In present application To load the driver class into application we are using following code,

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

The above statements loads driver class into application and it register driver class into DriverManager service with the help of static block present in JdbcOdbcDriver class.

The these Drivers can be divided into the following four types.

1. Type-1 Driver

2. Type-2 Driver

3. Type-3 Driver

4. Type-4 Driver

1. Type-1 Driver:-

- a. Type-1 Driver is also called as **JdbcOdbcDriver** or **Bridge Driver**.
- b. JdbcOdbcDriver is a reference implementation for Driver interface provided by Sun Microsystems in Java technology in the form of sun.jdbc.odbc.JdbcOdbcDriver. Sun Microsystems has provided JdbcOdbcDriver with the interdependency on the Microsoft product OdbcDriver. Odbc is an open specification, it can be used to interact with any type of database from JdbcOdbcDriver.
- c. Type-1 Driver is highly recommended for Standalone applications.
- d. If we want to use Type-1 Driver in our Jdbc applications then we must install the Microsoft provided Odbc native library in our machine. In case of Type-1 Driver, to interact with database we have to perform 2 types of conversions i.e. from **Java to Odbc** and **Odbc to the respective database**. Due to this reason Type-1 Driver is slower Driver, it will reduce performance of the Jdbc applications and it is less portable Driver.
- e. Type-1 Driver is not suitable for enterprise applications like web applications and so on.
- f. Type-1 Driver is suggestible for simple Jdbc applications, but not suggestible for complex Jdbc applications.

Note :- type-1 JDBC-ODBC Bridge plus ODBC Driver

This combination provides JDBC access via ODBC drivers. ODBC binary code -- and in many cases, database client code -- must be loaded on each client machine that uses a JDBC-ODBC Bridge. Sun provides a JDBC-ODBC Bridge driver, which is appropriate for experimental use and for situations in which no other driver is available.

2. Type-2 driver:-

- a. Type-2 Driver is also called as **Part Java, Part Native Driver** or **Native Driver**.
- b. Type-2 Driver was designed by using Java implementations as well as database vendors provided Native library.
- c. Type-2 Driver is costful Driver when compared with Type-1 Driver.
- d. If we want to use Type-2 Driver in our Jdbc applications then we have to install the database vendor provided Native library.
- e. When compared with Type-1 Driver Type-2 Driver will provide very good performance because Type-1 Driver may require 2 times conversions to interact with database, but Type-2 Driver may take 1 time conversion to interact with database.
- f. When compared with Type-1 Driver Type-2 Driver is more portable Driver because it

should not require Microsoft provided Native library. When compared with Type-1 Driver Type-2 Driver is faster Driver because it should not required two time conversions.

- g. Type-2 Driver is suggestible for only standalone applications, but not suggestible for enterprise applications like web applications and so on.

Note :- type-2 A native API partly Java technology-enabled driver

This type of driver converts JDBC calls into calls on the client API for Oracle, Sybase, Informix, DB2, or other DBMS. Note that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

3. Type-3 driver:-

- a. Type-3 Driver is also called as **Middleware Database Access Driver** or **NetworkDriver**.
- b. Type-1 and Type-2 Drivers are highly recommended for standalone applications, but Type-3 Driver is highly recommended for recommended for enterprise applications.
- c. Type-3 Driver is more portable Driver when compared with Type-1 and Type-2 Drivers because Type-3 Driver should not require Microsoft provided Native library and the respective database vendor provided Native library.
- d. Type-3 Driver is faster Driver when compared with Type-1 and Type-2 Drivers because Type-3 Driver should not require 2 times conversions in order to interact with database.
- e. Type-3 Driver is suggestible for any type of Jdbc applications.
- f. Type-3 Driver is highly recommended for web applications but not for standalone applications because it should require application server environment.
- g. Type-3 Driver will provide very good environment to interact with multiple number of databases from multiple clients at a time.

Note :- type-3 pure java driver for Database middleware:

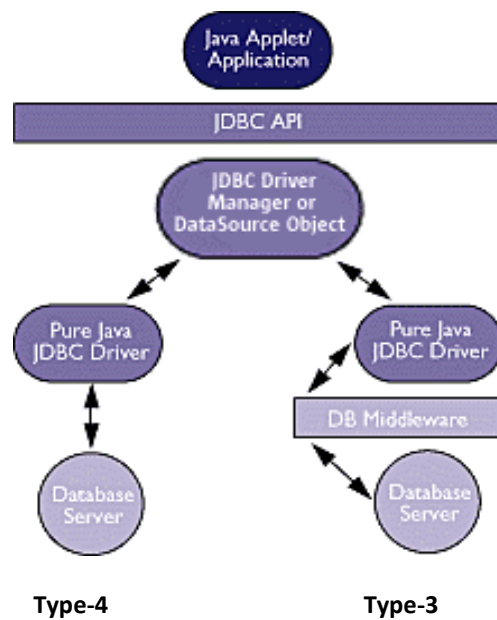
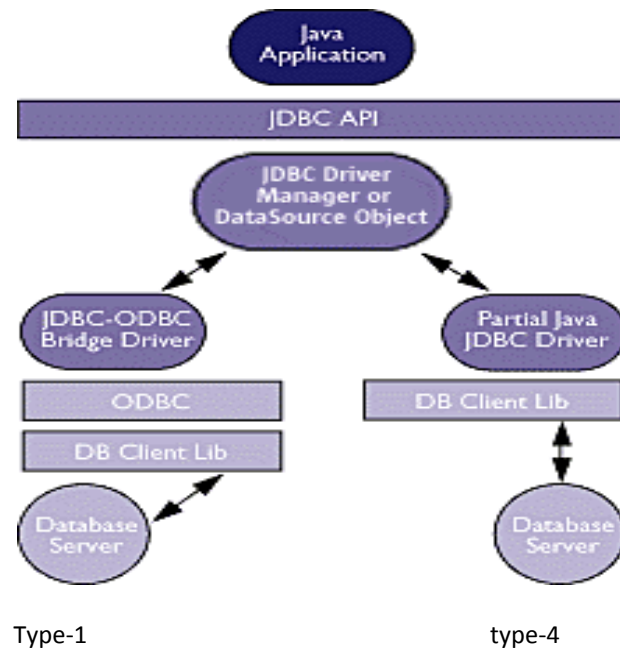
This driver translates the JDBC calls into the middleware vendors protocols, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases.

4. Type-4 driver:-

- a. Type-4 Driver is also called as **Thin Driver** or **Pure Java Driver**.
- b. Type-4 Driver was designed purely on the basis of Java technology.
- c. Type-4 Driver is more portable Driver when compared with Type-1, Type-2 and Type-3 Drivers because Type-4 Driver should not require Odbc Native library, Database vendor provided Native library and Application Server provided Middleware components.
- d. Type-4 Driver is frequently used Driver in Application Development.
- e. Type-4 Driver is recommended for any type of Java, J2EE applications i.e. both standalone applications and enterprise applications.
- f. Type-4 Driver is faster Driver when compared with all the remaining Drivers because Type-4 Driver should not require 2 times conversions in order to interact with database from Java applications.

Note :- type-4 Direct to the database (pure java driver)

This driver is converting jdbc calls into the network protocol used directly by DBMS. allowing a direct call from the client machine to the DBMS server and providing a practical solution for intranet access.



Type-1 driver:-

Driver class name **sun.jdbc.odbc.JdbcOdbcDriver**
 Driver URL pattern **("jdbc:odbc:DSNname","username","password")**
 Ex:-("jdbc:odbc:ratan","system","manager");

The DSN name is **Ratan** to create the DSN name we have to follow the following steps.

Start----->Control Panel----->system and security--->Administrative Tools----->Data Sources (ODBC)

↓
 -----Click the finish<-----Select a driver<-----click add button <-----user DSN
 ↓
 Provide DSN name (provide any name)--->Click on ok button----->ok----->ok

To load and register the driver ----->**Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");**

To provide connection to the particular data base

Connection con=DriverManager.getConnection("jdbc:odbc:first","system","manager");

Type-4 driver:-

Driver class name **oracle.jdbc.driver.OracleDriver**
 Driver URL pattern **("jdbc:oracle:thin:@localhost:1521:xe","username","password")**
 Ex:- ("jdbc:oracle:thin:@localhost:1521:xe","system","manager")

Type-4 driver is pure java driver DSN is not required

To load and register the driver ----->**Class.forName("oracle.jdbc.driver.OracleDriver");**

To provide connection to the particular database

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

Steps to design a first application:-

- Step 1:- Load the driver.**
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
- Step 2:- provide the URL pattern (connect with the particular database by using database username and password).**
Connection connection=DriverManager.getConnection("jdbc:odbc:ratan","ratan","ratan");
- Step 3:- Prepare the Statement object (to execute the SQL query).**
Statement statement=connection.createStatement();
- Step 4:- write the SQL query.**
String q1= "create table emp(eno number,ename varchar2(10),esal number)";
- Step 5:- Execute the SQL query.**
statement.executeUpdate(str);
- Step 6:- close the connection.**
connection.close();

java.sql.Statement object:-

In Jdbc applications, to write and execute the SQL queries we have to use the following methods from Statement.

1. executeQuery() method
2. executeUpdate() method
3. execute() method

The above three methods are present in the Statement object.

1) executeQuery() method:-

- a. In Jdbc applications, executeQuery() method can be used to execute selection group SQL queries in order to fetch the data from database.
- b. When JVM encounters executeQuery() method along with selection group SQL query then JVM will pick up that selection group SQL query and send to database through Jdbc Driver and connection.
- c. Here Database Engine will pick up the selection group SQL query and execute it by performing Query Tokenization, Query Parsing, Query Optimization and Query Execution. By the execution of selection group SQL query Database Engine will fetch the data from database table and send back to Java application through the same connection and Jdbc Driver.
- d. In the above context, Jdbc application will store the fetched data in the form of an object at heap memory is called as ResultSet.
- e. After getting ResultSet object JVM will return that ResultSet object reference as return value from executeQuery() method as per the predefined implementation of executeQuery() method.

public ResultSet executeQuery(String sql_query)throws SQLException

Ex: ResultSet rs=st.executeQuery("select * from emp1");

2) executeUpdate() method:-

- a. In Jdbc applications, executeUpdate() method can be used to execute updation group SQL queries in order to perform the basic database operations like create, insert, update, delete, drop, alter and so on.
- b. When JVM encounters executeUpdate() method along with updation group SQL query then JVM will pickup that updation group SQL query and send to the database through Jdbc Driver and connection.
- c. At database Database Engine will pick up that SQL query, execute it, perform the updations on database table, identifying row count value (the number of records got affected with the provided SQL query) and send that row count value back to java application.
- d. After getting row count value JVM will return that generated row count value as return value from executeUpdate() method as per the predefined implementation of executeUpdate() method.

public int executeUpdate(String sql_query)throws SQLException

Ex: int rowCount=st.executeUpdate("update emp1 set esal=esal+500 where esal<10000");

System.out.println(rowCount); // 3

updation group of sql statement--->use **executeUpdate()**---->return type is(int)

Type-1

create---> -1

drop----> -1

insert----> 1(only one record at a time)

update -->5(based on updated rows)

Type-4

create--->0

drop---->0

insert---->1(only one record at a time)

update-->5(based on updated rows)

3) **execute() method:-**

- In Jdbc applications, execute() method can be used to execute both selection group SQL queries and updation group SQL queries.
- When JVM encounters execute() method along with selection group SQL query then JVM will pick up that SQL query and send to Database Engine, where at database Database Engine will fetch the data from database table and send back to Java application.
- At Java application the fetched data will be stored in the form of ResultSet object, but as per the predefined implementation of execute() method JVM will return true as a boolean value from execute() method.
- When JVM encounters execute() method along with updation group SQL query then JVM will send that SQL query to Database Engine, where Database Engine will perform updations on database and return row count value to Java application, but as per predefined implementation of execute() method JVM will return false as a boolean value from execute() method.

Syntax:- `public boolean execute(String sql_query)throws SQLException`

execute() ---->**updation group of sql queries**---->**false**

```
boolean b2=statement.execute("update emp1 set esal=esal+500 where esal<10000");
System.out.println(b2); //false
```

execute() ----->**selection group of sql queries**-->**true**

```
boolean b1=statement.execute("select * from emp1");
System.out.println(b1); //true
```

Example :-

```
import java.sql.*;
import java.io.*;
import java.util.*;
class Test
{
    public static void main(String[] args) throws Exception
    {
        //TYPE-1 DRIVER
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:first","system","system");
        //TYPE-3 DRIVER
        //Class.forName("ids.sql.IDSServer");
        //Connection con=DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn='accdsn'");
        //TYPE-4 DRIVER
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection
        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","system");
        //TYPE-5 DRIVER
        //Class.forName("com.ddtek.jdbc.oracle.OracleDriver");
        //Connection
        con=DriverManager.getConnection("jdbc:datadirect:oracle://localhost:1521:serviceName='orcl','system','manager");
        Statement st=con.createStatement(); //statement object creation
        Scanner br=new Scanner(System.in); //used to take input from keyboard
        System.out.println("enter table name");
        String tname=br.readLine();
        st.executeUpdate("create table "+tname+"(eno number,ename varchar2(10),esal number)");
        System.out.println("table "+tname+" created successfully");
        System.out.println(rowcount);
        con.close(); //close the connection
    }
}
```

Example :-

```
package com.dss;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class Test {
    public static void main(String[] args) throws ClassNotFoundException, SQLException,
    InterruptedException {
        System.out.println("*****connection creation process*****");
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection connection=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        Statement statement=connection.createStatement();
        System.out.println("*****table creation process*****");
        String query1="create table emp11(eid number,ename varchar2(24),esal number)";
        int a=statement.executeUpdate(query1);
        System.out.println(a);
        System.out.println("table is create successfully");
        System.out.println("*****insertion process*****");
        String query2="insert into emp11 values(111,'ratan',40000)";
        String query3="insert into emp11 values(222,'ratan',50000)";
        String query4="insert into emp11 values(333,'suneel',60000)";
        statement.executeUpdate(query2);
        statement.executeUpdate(query3);
        statement.executeUpdate(query4);
        System.out.println("values are inserted successfully");
        System.out.println("*****retriveing process*****");
        String query5="select * from emp11";
        ResultSet set=statement.executeQuery(query5);
        while(set.next())
        {System.out.println(set.getInt(1)+"---"+set.getString(2)+"---"+set.getInt(3));
        }
        System.out.println("values are printed success fully successfully");
        System.out.println("*****updation process*****");
        String query6="update emp11 set esal=esal+500 where esal>40000";
        int count=statement.executeUpdate(query6);
        System.out.println("updated records----->" +count);
        System.out.println("table is updated successfully successfully");

        Thread.sleep(30000);
        System.out.println("*****deletion process*****");
        String query7="drop table emp11";
        statement.executeUpdate(query7);
        System.out.println("table dropped successfully");
    }
}
```

Output:-

```

*****connection creation process*****
*****table creation process*****
-1
table is create successfully
*****insertion process*****
values are inserted successfully
*****retriveing process*****
111---ratan---40000
222---ratan---50000
333---suneel---60000
values are printed success fully successfully
*****upadation process*****
updated records----->2
table is updated successfully successfully
*****deletion process*****
table dropped successfully

```

Java.util.Scanner(Dynamic Input):-

1. Scanner class present in **java.util** package and it is introduced in 1.5 version.
2. Scanner class is used to take dynamic input from the keyboard.

Scanner s = new Scanner(System.in);

to get int value ----> s.nextInt()

to get float value ---> s.nextFloat()

to get byte value ---> s.nextbyte()

to get String value ---> s.next()

to get single line ---> s.nextLine()

to close the input stream ---> s.close()

import java.util.*;

class Test

```

{
    public static void main(String[] args)
    {
        Scanner s=new Scanner(System.in);    //used to take dynamic input from keyboard
        System.out.println("enter emp hobbies");
        String ehobbies = s.nextLine();
        System.out.println("enter emp no");
        int eno=s.nextInt();
        System.out.println("enter emp name");
        String ename=s.next();
        System.out.println("enter emp salary");
        float esal=s.nextFloat();
        System.out.println("*****emp details*****");
        System.out.println("emp no----->" + eno);
        System.out.println("emp name---->" + ename);
        System.out.println("emp sal----->" + esal);
        System.out.println("emp hobbies----->" + ehobbies);
        s.close();    //used to close the stream
    }
}

```

Ex :- application that contains creation, insertion, updating, deletion.

```
import java.sql.*;
import java.util.*;
class DataBase
{
    public static void main(String[] args)throws Exception
    {
        System.out.println("java developer jdbc is started ");
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        Statement st=con.createStatement();
        Scanner s=new Scanner(System.in);

        System.out.println("*****table creation process*****");
        System.out.println("pls enter table name");
        String tname=s.next();
        String str1="create table "+tname+"(eid number,ename varchar2(23),esal number)";
        st.executeUpdate(str1);
        System.out.println("table " +tname+ " crewated successfully");
        System.out.println("*****insertion process*****");
        while (true)
        {
            System.out.println("please enter emp id");
            int eid=s.nextInt();
            System.out.println("please enter emp name");
            String ename=s.next();
            System.out.println("please enter emp sal");
            int esal=s.nextInt();

            String str2="insert into "+tname+" values("+eid+"','"+ename+"','"+esal+"")";
            st.executeUpdate(str2);
            System.out.println("do u want one more record(yes/no)");
            String option=s.next();
            if (option.equals("no"))
            {
                break;
            }
        }
        System.out.println("valus are inserted successfully");
        System.out.println("*****retrive the data *****");
        String str3="selet * from "+tname;
        ResultSet rs= st.executeQuery(str3);
        while(rs.next())
        {
            System.out.println(rs.getInt(1)+ " "+ rs.getString(2)+ "" + rs.getInt(3));
        }

        System.out.println("*****delete the data *****");
        System.out.println("please enter employee salary which u want delete");
        int esal1=s.nextInt();
        int rowcount=st.executeUpdate("delete "+tname+" where esal>="+esal1+"");
        System.out.println("no of records effected:"+rowcount);
        String str5="select * from "+tname;
```

```
        ResultSet rs1=st.executeQuery(str5);
        while (rs1.next())
        {
            System.out.println(rs1.getInt(1)+ " "+ rs1.getString(2)+ "" + rs1.getInt(3));
        }
        System.out.println("*****update the data *****");
        String str6="update "+tname+" set esal=esal+500 where esal<=60000";
        int rowcount1=st.executeUpdate(str6);
        System.out.println("no of records effected:"+rowcount1);
        String str7="select * from "+tname;
        ResultSet rs2=st.executeQuery(str7);
        while(rs2.next())
        {System.out.println(rs2.getInt(1)+ " "+ rs2.getString(2)+ "" + rs2.getInt(3));
        }
        System.out.println("*****Drop the table*****");
        String str4="drop table "+tname;
        st.executeUpdate(str4);
        System.out.println("table dropped "+tname+" successfully");
    }
};
```

Example:-working with MySql data base

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class Test
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection
        connection=DriverManager.getConnection("jdbc:mysql://localhost:3306/ratandb","root","root");
        Statement statement=connection.createStatement();
        String q1="select * from emp";
        ResultSet resultSet=statement.executeQuery(q1);
        while (resultSet.next())
        {
            System.out.println(resultSet.getInt("eid")+"--"+resultSet.getString("ename")+"--"
            "+resultSet.getFloat("esal"));
        }
        connection.close();
    }
}
```

Java.sql.ResultSet:-

1. ResultSet is a interface present in the java.sql package and it is used to hold the values which are coming from data base
2. We are able to access the data from the ResultSet Object by using cursor is not a data base cursor. This cursor is pointer that points to one row of data in ResultSet, initially the cursor is positioned before the first row.
3. The default ResultSet object is CONCUR_READ_ONLY and TYPE_FORWARD_ONLY

ResultSet concurrency:-

Based on the ResultSet concurrency the ResultSet object is divided into 2 types

- 1) CONCUR_READ_ONLY(it is possible to just read the data)
- 2) CONCUR_UPDATABLE(it is possible to read the data and possible to update the data)

ResultSet sensitivity:-

Based on the ResultSet sensitivity the ResultSet object is divided into 2 types

- 1) TYPE_FORWARD_ONLY (it is possible to read the data only in forward direction)
- 2) SCROLL(it is possible to read the data both forward and backward direction)
 - a. TYPE_SCROLL_SENSITIVE
 - b. TYPE_SCROLL_INSENSITIVE

Ex:-ResultSet Readonly and forward only(default behavior of the ResultSet Object)

```
import java.sql.*;
import java.util.*;
class Test
{
    public static void main(String[] args)throws Exception
    {Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        Statement st = con.createStatement();
        System.out.println("*****retrive the data *****");
        String str3="select * from emp";
        ResultSet rs= st.executeQuery(str3);
            while(rs.next())
            {
                System.out.println(rs.getInt("eno"));
            }
        }
    }
```

Ex:-ResultSet object Scrollable and Readonly.

```
import java.sql.*;
import java.util.*;
class Test
{
    public static void main(String[] args)throws Exception
    {
        System.out.println("java developer jdbc is started ");
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        Statement st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_READ_ONLY);
        System.out.println("*****retrive the data *****");
        String str3="select * from emp";
        ResultSet rs= st.executeQuery(str3);
        rs.afterLast();//the cursor is pointing to the after last record
        while (rs.previous())
        {
            System.out.println(rs.getInt("eno"));
        }
    }
}
```

Ex:- ResultSet object scrollable and UPDATABLE

```
import java.sql.*;
import java.util.*;
class Test
{
    public static void main(String[] args)throws Exception
    {
        System.out.println("java developer jdbc is started ");
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        Statement st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        System.out.println("*****retrive the data *****");
        String str3="select * from emp";
        ResultSet rs= st.executeQuery(str3);
        while(rs.next())
        {
            rs.updateInt("eno",1222);
            rs.updateRow();
        }
    }
}
```


Cursor methods:-when a ResultSet object is first created, the cursor is positioned before the first row.

Next():-

Moves the cursor forward one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned after the last row.

public abstract boolean next() throws java.sql.SQLException;

Previous():-

Moves the cursor backward one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned before the first row.

public abstract boolean previous() throws java.sql.SQLException;

First():-

Moves the cursor to the first row in the ResultSet object. Returns true if the cursor is now positioned on the first row and false if the ResultSet object does not contain any rows.

public abstract boolean first() throws java.sql.SQLException;

Last():-

Moves the cursor to the last row in the ResultSet object. Returns true if the cursor is now positioned on the last row and false if the ResultSet object does not contain any rows.

public abstract boolean last() throws java.sql.SQLException;

beforeFirst():-

Positions the cursor at the start of the ResultSet object, before the first row. If the ResultSet object does not contain any rows, this method has no effect.

public abstract void beforeFirst() throws java.sql.SQLException;

afterLast():- Positions the cursor at the end of the ResultSet object, after the last row. If the ResultSet object does not contain any rows, this method has no effect.

public abstract void afterLast() throws java.sql.SQLException;

absolute(int row):- Positions the cursor on the row specified by the parameter row.

Example:-

```
import java.sql.*;
import java.util.*;
class Test
{
    public static void main(String[] args)throws Exception
    {Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection con=DriverManager.getConnection("jdbc:odbc:ratan ","system","manager");
    Statement st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
    String str3="select * from emp";
    ResultSet rs=st.executeQuery(str3);
    rs.last();
    System.out.println(rs.getString(2));
    rs.first();
    System.out.println(rs.getString(2));
    rs.absolute(3);
    System.out.println(rs.getString(2));
    con.close();
    }
}
```

Ex:-update the first record and last record and specified record.

```
import java.sql.*;
import java.util.*;
class Test
{
    public static void main(String[] args)throws Exception
    {
        System.out.println("java developer jdbc is started ");
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        Statement st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
        System.out.println("*****retrive the data *****");
        String str3="select * from emp";
        ResultSet rs= st.executeQuery(str3);
        rs.first();
        rs.updateInt("eno",999);
        rs.updateRow();

        rs.last();
        rs.updateInt("eno",888);
        rs.updateRow();

        rs.absolute(3);
        rs.updateInt("eno",999);
        rs.updateRow();
    }
}
```

Java.sql.ResultSetMetaData:-

Data about the data available in ResultSet object is called as **ResultSetMetaData**. In Jdbc applications, ResultSetMetaData include the number of columns which are available in ResultSet object, names of each and every column, datatypes of each and every column and so on.

To get ResultSetMetaData object:-

Public ResultSetMetaData getMetaData()

To get the number of columns available in ResultSet object:-

public int getColumnCount()

To get the name of a particular column available in ResultSet object:-

public String getColumnName(int column_index)

To get the datatype of a particular column :-

public String getColumnName(int column_index)

To get the size of the particular column:-

public int getColumnDisplaySize(int column_index)

Example :-

```
import java.sql.*;
public class Test
{public static void main(String[] args) throws Exception
{      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
      Statement st=con.createStatement();
      ResultSet rs=st.executeQuery("select * from emp");
      ResultSetMetaData rsmd=rs.getMetaData();
      int count=rsmd.getColumnCount();
      System.out.println("Number of columns....."+count);
      System.out.println("*****");
      for (int i=1;i<=count;i++)
      {      System.out.println("Column Name....."+rsmd.getColumnName(i));
              System.out.println("Data Type....."+rsmd.getColumnTypeName(i));
              System.out.println("Column Size....."+rsmd.getColumnDisplaySize(i));
              System.out.println("-----");
      }
      con.close();
}
}
```

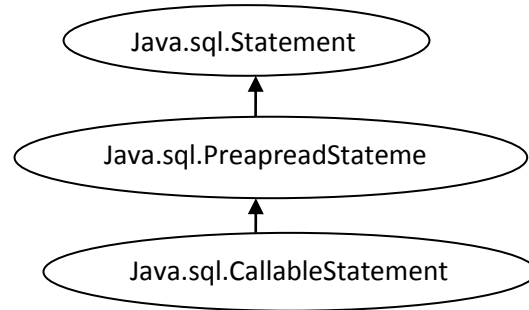
Java.sql.DatabaseMetaData:-

By using DatabaseMetaData we are able to get the database details

- a. Database username
- b. Password
- c. Drivername
- d. Database version.....etc

Example:-

```
import java.sql.*;
class Test
{      public static void main(String args[]){
      try{
              Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","Manager");
      DatabaseMetaData dbmd=con.getMetaData();
      System.out.println("Driver Name: "+dbmd.getDriverName());
      System.out.println("Driver Version: "+dbmd.getDriverVersion());
      System.out.println("UserName: "+dbmd.getUserName());
      System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
      System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());
      con.close();
      }
      catch(Exception e)
      {
              System.out.println(e);
      }
}
}
```

Statement object structure:-**PreparedStatement:-**

- 1) It is sub interface of statement it is used to execute parameterized query.
- 2) In Jdbc applications, we will prefer to use Statement object when we have a requirement to execute all the SQL queries execute independently. In Jdbc applications, when we have a requirement to execute the same SQL query in the next sequence, where to improve the performance of Jdbc applications we will use PreparedStatement.
- 3) Whenever we are executing the query at that situation each and every time at database side the following operations will be performed.
 - a. Query tokenization
 - b. Query parsing
 - c. Query optimization
 - d. Query execution
- 4) If we using Statement interface to execute query every time the above operations are performed at situation the performance of the application will be reduced.
- 5) For the above requirement, if we use Statement then Database Engine will perform Query Tokenization, Query Parsing, Query Optimization and Query Execution without having variation from one attempt to another attempt. This approach will increase burden on the Database Engine, it will reduce the performance of the Jdbc applications.
- 6) To overcome above limitation we should go for PreparedStatement Interface. if we are using preparedStatement the query compiled only one time means the query is loaded at only one time hence the performance of the application will be increased.

Step 1: Create PreparedStatement object with the generalized SQL query format. To create PreparedStatement object we have to use the following method from Connection.

```
public PreparedStatement prepareStatement(String query_fmt)throws SQLException
```

Ex: PreparedStatement pst=con.prepareStatement("insert into emp1 values(?,?,?);");

When JVM encounters the above instruction then JVM will pick up the provided generalized SQL query format and send to Database Engine, where Database Engine will perform Query processing only one time. After the Query processing Database Engine will prepare a buffer with the positional parameters called as **Query Plan**. After getting Query plan at database automatically PreparedStatement object will be created at Java application with the same positional parameters.

Step 2: Set values to the positional parameters available in PreparedStatement object. To set values to the positional parameters we will use the following method from PreparedStatement.

```
public void setXxx(int param_index, xxx value)
```

Where xxx may be byte, short, int and so on.

Ex: pst.setInt(1, 111);

```
    pst.setString(2, "aaa");
```

```
    pst.setFloat(3, 5000.0f);
```

When JVM encounters the above piece of code JVM will set the specified values to the respective positional parameters in PreparedStatement object, that values will be reflected to the positional parameters available in Query plan.

Step 3: Make the Database Engine to pick up the values from Query plan and perform the respective database operations.

Example:-

```
package com.dss;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
public class Test {
    public static void main(String[] args) throws ClassNotFoundException, SQLException {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection connection=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        PreparedStatement statement=connection.prepareStatement("insert into emp values(?,?,?)");
        Scanner scanner=new Scanner(System.in);
        while(true){
            System.out.println("enter emp number:-");
            int eno=scanner.nextInt();
            System.out.println("enter emp name:-");
            String ename=scanner.next();
            System.out.println("enter emp sal:-");
            int esal=scanner.nextInt();

            statement.setInt(1,eno);
            statement.setString(2,ename);
            statement.setInt(3,esal);
            statement.executeUpdate();
            System.out.println("record inserted sucessfully");

            System.out.println("do u want one more record");
            String option=scanner.next();
            if(option.equals("no"))
                break;
        }
    }
}
```

Batch Updates:

In general in Jdbc applications, it is required to provide number of SQL queries according to application requirement.

With the above, if we execute the Jdbc application then JVM will send all the SQL queries to the database in a sequential manner.

If we use the above convention to execute SQL queries in Jdbc application we have to spend a lot of time only to carry or transfer SQL queries from Java application to database, this approach will reduce the performance of Jdbc application.

In the above context, to improve the performance of the Jdbc applications we have to use Batch updates.

In batch updates, we will gather or collect all the updation group SQL queries as a single unit called as **Batch** and we will send batch of updation group SQL queries at a time from Java application to database.

At database, Database Engine may execute all the SQL queries and generate respective row count values in the form of an array to Java application.

To add an SQL query to batch we have to use the following method from Statement.

```
public void addBatch(String query)
```

To send batch of updation group SQL queries at a time from Java application to database and to make the Database Engine to execute all the batch of updation group SQL queries we have to use the following method from Statement.

```
public int[] executeBatch()
```

Where int[] will represent all the row count values generated from the updation group SQL queries.

Example:-

```
import java.sql.*;
public class BatchUpdatesDemo
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection
        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan"
        );
        Statement st=con.createStatement();
        st.addBatch("insert into student values(104,'ramesh',70)");
        st.addBatch("update student set smarks=smarks+5 where smarks<90");
        st.addBatch("delete student where sid=102");
        int[] rowCounts=st.executeBatch();
        for (int i=0;i<rowCounts.length;i++)
        {
            System.out.println("Records updated....."+rowCounts[i]);
        }
        con.close();
    }
}
```

Note: If we include selection group SQL query in a batch then JVM will raise an Exception like `ava.sql.BatchUpdateException: invalid batch command: invalid SELECT batch command`.

Batch updations:-we are able to perform by using Statement object.

```
import java.sql.*;
import java.util.*;
class Test
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        Statement st = con.createStatement();
        String str1="insert into emp values(789,'ratansoft',50000)";
        st.addBatch(str1);
        st.addBatch("update emp set esal=esal+200 where esal>45000");
        st.addBatch("delete emp where eno=143");
        int[] a=st.executeBatch();
        for (int i=0;i<a.length;i++ )
        {
            System.out.println("records updated"+a[i]);
        }
    }
}
```

Steps to design standalone applications:- [not required]

Step 1:- prepare the component and add the components to the frame

Step 2:- set the particular layout to the frame.

Step 3:- conversion of static component into the dynamic component.(by adding listeners)

JDBC with Standalone applications:-**Example 1:-**

```
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
class SearchFrame extends Frame implements ActionListener
{
    Label l;
    TextField tf;
    Button b;
    Connection con;
    Statement st;
    ResultSet rs;
    SearchFrame()
    {
        try
        {
            this.setVisible(true);
            this.setSize(500,400);
            this.setBackground(Color.pink);
            this.setTitle("JDBC - AWT Application");
            this.setLayout(new FlowLayout());
            l=new Label("productId");
            tf=new TextField(15);
            b=new Button("Search");
            b.addActionListener(this);
            this.add(l);
            this.add(tf);
            this.add(b);
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
            st=con.createStatement();
        }
        catch (Exception e)
        {e.printStackTrace();}
    }
    public void actionPerformed(ActionEvent ae)
    {try
    {rs=st.executeQuery("select * from emp where eno='"+tf.getText()+"'");
    repaint();
    }
    catch (Exception e)
    {e.printStackTrace();}
    }
```

```
public void paint(Graphics g)
{
    try
    {
        Font f=new Font("arial",Font.BOLD,30);
        g.setFont(f);
        boolean b=rs.next();
        if(b==true)
        {
            g.drawString("emp id....."+rs.getInt(1),50,100);
            g.drawString("emp Name....."+rs.getString(2),50,150);
            g.drawString("emp sal....."+rs.getInt(3),50,200);
        }
        else
        {
            g.drawString("emp does not exists",50,150);
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

class Test
{
    public static void main(String[] args)
    {
        new SearchFrame();
    }
}
```

Example 2:-

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;

class MyFrame extends Frame implements ActionListener
{
    Button b1,b2;
    Connection con;
    ResultSet rs;
    Statement st;
    String label;
    MyFrame()
    {
        try{
            this.setVisible(true);
            this.setSize(500,500);
            this.setBackground(Color.red);
            b1=new Button("NEXT");
            b2=new Button("PREVIOUS");
            this.setLayout(new FlowLayout());
            this.add(b1);
            this.add(b2);
            b1.addActionListener(this);
        }
    }
}
```



```
        b2.addActionListener(this);
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
        rs=st.executeQuery("select * from emp");
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
public void actionPerformed(ActionEvent e)
{
    label=e.getActionCommand();
    repaint();
}
public void paint(Graphics g)
{
    try{
        if (label.equals("NEXT"))
        {
            boolean b=rs.next();
            if (b==true)
            {
                g.drawString("emp number"+rs.getInt(1),50,100);
                g.drawString("emp name"+rs.getString(2),50,200);
                g.drawString("emp sal"+rs.getInt(3),50,300);
            }
            else
            {
                g.drawString("no record",50,500);
            }
        }
        if (label.equals("PREVIOUS"))
        {
            boolean b=rs.previous();
            if (b==true)
            {
                g.drawString("emp number"+rs.getInt(1),50,100);
                g.drawString("emp name"+rs.getString(2),50,200);
                g.drawString("emp sal"+rs.getInt(3),50,300);
            }
            else
            {
                g.drawString("no record",50,500);
            }
        }
    }
    catch(Exception e)
    {
        System.out.println(e);
    }
}
};
class Test
{
    public static void main(String[] args)
    {
        MyFrame f=new MyFrame();
    }
};
```

Transaction Management:-

To change connection's auto commit mode we have to use the following method from Connection.

```
public void setAutoCommit(boolean b)throws SQLException
```

If b==true then the connection will be in auto commit mode else the connection will be in non-auto commit mode.

Ex: con.setAutoCommit(false);

If we change connection's auto commit mode then we have to perform either commit or rollback operations to complete the transactions.

To perform commit and roll back operations we have to use the following methods from Connection.

```
public void commit()throws SQLException
```

```
public void rollback()throws SQLException
```

Note: In case of connection's non-auto commit mode, when we submit SQL query to the connection then connection will send that SQL query to Database Engine and make the Database Engine to execute that SQL query and store the results on to the database table temporarily. In this case, Database Engine may wait for commit or rollback signal from client application to complete the transactions.

Example:-

```
import java.sql.*;
public class TransactionMgmtDemo
{
    public static void main(String[] args)
    {
        Connection con=null;
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
            con.setAutoCommit(false);
            Statement st=con.createStatement();
            st.executeUpdate("insert into emp values(104,'kamal',8000)");
            st.executeUpdate("insert into emp values(105,'nandu',9000)");
            st.executeUpdate("insert into emp values(106,'indhu',9500)");
            con.commit();
            System.out.println("Transaction success");
        }
        catch (Exception e)
        {
            try
            {
                con.rollback();
                System.out.println("Transaction failure");
                e.printStackTrace();
            }
            catch (Exception e1)
            {
                e1.printStackTrace();
            }
        }
    }
}
```

Java.sql.SavePoint:-

One of the features introduced in jdbc3.0 version. And it is a interface Savepoint is a intermediate point and makes it possible to rollback the transaction upto the save point instead of roll backing the entire transaction.

To represent Savepoint Jdbc has provided a predefined interface java.sql.Savepoint.

To set a Savepoint we have to use the following method form Connection.

```
public Savepoint setSavepoint()
```

To perform rollback operation on set of instructions executive w.r.t a particular

Savepoint we have to use the following method from Connection.

```
public void rollback(Savepoint sp)
```

To release Savepoint we will use the following method form Connection.

```
public void releaseSavepoint(Savepoint sp)
```

Note: In Jdbc applications, Savepoint concept could be supported by Type-4 Driver provided by Oracle, which could not supported by Type-1 Driver provided by Sun Microsystems.

Note: Type-4 Driver is able to support Savepoint up to setSavepoint() method and rollback(_) method, not releaseSavepoint(_) method.

Ex:-

```
import java.sql.*;
public class SavepointDemo
{
    public static void main(String[] args)
    {
        Connection con=null;
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
            con.setAutoCommit(false);
            Statement st=con.createStatement();
            st.executeUpdate("insert into student values(105,'srinu',55)");
            Savepoint sp=con.setSavepoint();
            st.executeUpdate("insert into student values(106,'amala',75)");
            con.rollback(sp);
            st.executeUpdate("insert into student values(107,'sania',80)");
            con.commit();
        }
        catch (Exception e)
        {
            try
            {
                e.printStackTrace();
                con.rollback();
            }
            catch (Exception e1)
            {
                e1.printStackTrace();
            }
        }
        System.out.println("Hello World!");
    }
}
```

Example:-

```
import java.sql.*;
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
            Statement st = con.createStatement();
            st.executeUpdate("insert into emp values(123,'baji',10000)");
            Savepoint save=con.setSavepoint("aaa");
            st.executeUpdate("insert into emp values(321,'hassi',20000)");
            con.rollback(save);
            con.commit();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

Java.sql.CallableStatement:-

- ❖ This interface used to execute stored procedures and functions of PL/SQL.
- ❖ We can write business logics in data base with procedures and functions it will improve the performance because these are pre compiled.
- ❖ To get the employee salary based on eid then creates the procedure or function to take the eid as a input and returns salary as a output.

Stored procedures:-

1. Stored procedure return value is optional.
2. Procedure can have both input and output parameters(IN & OUT)
3. Procedure is able to call functions.

Syntax:-

```
create or replace procedure procedure_name([param-list])
as
----- Global declarations
BEGIN
----- Database logic
END procedure_name;
/ (press enter to save and compile the procedure)
```

Functions:-

1. Function must return the value.
2. Function can have only IN(input) parameter.
3. From the functions we are unable to call procedures.

Syntax:-

```
create or replace function function_name([param-list]) return data_type
as
----- Global declarations
BEGIN
----- Database logic
return value;
END function_name;
/ (press enter to save and compile the function)
```

IN :- it is a input value to procedure or function call & set the input value by using setXXX().
OUT :- it is a output value of procedure or function & get the output value by using getXXX().
INOUT :- it is parameter of both input and output values set the input value by using setXXX() and get the value by using getXXX().

Creation of CallableStatement:-

Create the callable statement object by using prepareCall() method of Connection interface.

```
public CallableStatement prepareCall(String pro_cal)throws SQLException
CallableStatement cst=connection.prepareCall("{call getSal(?,?)}");
```

When JVM encounters the above instruction JVM will pick up procedure call and send to Database Engine, where Database Engine will parse the procedure call and prepare a query plan with the positional parameters, as a result CallableStatement object will be created at Java application.

Step 2: If we have IN type parameters in CallableStatement object then set values to IN type parameters.

To set values to IN type parameters we have to use the following method.

```
public void setXxx(int param_position, xxx value)
```

Where xxx may be byte, short, int and so on.

Ex: `cst.setInt (1, 111);`

Step 3: If we have OUT type parameter in CallableStatement object then we have to register OUT type parameter with a particular datatype.

To register OUT type parameter we will use the following method.

```
public void registerOutParameter(int param_position, int data_type)
```

Where data_type may be the constants from Types class like BYTE, SHORT, INTEGER, FLOAT and so on.

Ex: `cst.registerOutParameter(2, Types.FLOAT);`

Step 4: Make Database Engine to pick up the values from Query plan and to execute the respective procedure or function.

To achieve this we have to use The following method.

```
public void execute()throws SQLException
```

Ex: `cst.execute();`

Step 5: Get the values from OUT type parameters available in CallableStatement object.

After executing the respective procedure or function the respective values will be stored in OUT type parameters in CallableStatement object from stored procedure or functions. To access the OUT type parameter values we have to use the following method.

```
public xxx getXxx(int param_position)
```

Where xxx may be byte, short, int and so on.

EX: `float sal=cst.getFloat(2);`

Ex:- execution of procedures

```
create or replace procedure getSal(id IN number, sal OUT number)
as
BEGIN
select esal into sal from emp where eid=id;
END getSal;
/
```

Example:-

```
package com.sravva;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;
public class TestDb {
    public static void main(String[] args) throws Exception
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection
        connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
        CallableStatement cst=connection.prepareCall("{call getSal(?,?)}");
        cst.setInt(1,111);
        cst.registerOutParameter(2, Types.FLOAT);
        cst.execute();
        System.out.println("Salary....."+cst.getFloat(2));
        connection.close();
    }
}
```

Ex:- Execution of functions

```
create or replace function getAvg(id1 IN number, id2 IN number) return number
as
sal1 number;
sal2 number;
BEGIN
select esal into sal1 from emp where eid=id1;
select esal into sal2 from emp where eid=id2;
return (sal1+sal2)/2;
END getAvg;
/
```

Example:-

```
package com.sravva;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;
public class TestDb {
    public static void main(String[] args) throws Exception
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection connection
        =DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
        CallableStatement cst=connection.prepareCall("{?=call getAvg(?,?)}");
        cst.setInt(2,888);
        cst.setInt(3,666);
        cst.registerOutParameter(1, Types.FLOAT);
        cst.execute();
        System.out.println("Average Salary....."+cst.getFloat(1));
        connection.close();
    }
}
```

Connection Pooling:

In general in Jdbc applications, when we have a requirement to perform database operations we will establish the connection with the database from a Java application, at the end of the application we will close the connection i.e. destroying Connection object.

In Jdbc applications, every time establishing the connection and closing the connection may increase burden to the Jdbc application, it will reduce the performance of the jdbc application. In the above context, to improve the performance of Jdbc applications we will use an alternative called as Connection Pooling.

In Connection pooling at the time of application startup we will prepare a fixed number of Connection objects and we will keep them in a separate base object called **Pool object**.

In Jdbc applications, when we have a requirement to interact with the database then we will get the Connection object from Pool object and we will assign it to the respective client application. At the end of the Jdbc application we will keep the same Connection object in the respective Pool object without destroying.

The above mechanism will improve the performance of the application is called as **Connection Pooling**.

If we want to implement Connection pooling in Jdbc application we have to use the following steps.

Step 1: Prepare DataSource object.

DataSource is an object, it is able to manage all the Jdbc parameter which are required to establish the connections.

To represent DataSource object Java API has provided a predefined interface i.e. `javax.sql.DataSource`.

DataSource is an interface provided by Jdbc API, but whose implementation classes are provided by all the database vendors.

With the above convention Oracle has provided an implementation class to DataSource interface in `ojdbc6.jar` file i.e. `oracle.jdbc.pool.OracleConnectionPoolDataSource`.

Ex: `OracleConnectionPoolDataSource ds=new OracleConnectionPoolDataSource();`

Step 2: Set the required Jdbc parameters to DataSource object.

To set the Jdbc parameters like Driver url, database username and password to the DataSource object we have to use the following methods.

`public void setURL(String driver_url)`

`public void setUser(String user_name)`

`public void setPassword(String password)`

Ex: `ds.setURL("jdbc:oracle:thin:@localhost:1521:xe");`

`ds.setUser("system");`

`ds.setPassword("venkat");`

Step 3: Get the PooledConnection object.

PooledConnection is an object provided by DataSource, it can be used to manage number of Connection objects.

To represent PooledConnection object Jdbc API has provided a predefined interface i.e. `javax.sql.PooledConnection`.

To get PooledConnection object we have to use the following method from DataSource.

`public PooledConnection getPooledConnection()`

Ex: `PooledConnection pc=ds.getPooledConnection();`

Step 4: Get Connection object from PooledConnection.

To get Connection object from PooledConnection we have to use the following method.

`public Connection getConnection()`

Ex: `Connection con=pc.getConnection();`

Step 5: After getting Connection prepare Statement or preparedStatement or CallableStatement and perform the respective database operations.

Ex: `Statement st=con.createStatement();`

Ex:-

`import java.sql.*;`

`import javax.sql.*;`

`import oracle.jdbc.pool.*;`

`public class ConnectionPoolDemo`

`{`

`public static void main(String[] args) throws Exception`

`{`

```
OracleConnectionPoolDataSource ds=new OracleConnectionPoolDataSource();
ds.setURL("jdbc:oracle:thin:@localhost:1521:xe");
ds.setUser("system");
ds.setPassword("ratan");
PooledConnection pc=ds.getPooledConnection();
Connection con=pc.getConnection();
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from emp");
System.out.println("EID ENAME ESAL");
System.out.println("-----");
while (rs.next())
{
System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getFloat(3));
}
}
}
```

Note: The above approach of implementing Connection pooling is suggestible up to standalone applications, it is not suggestible in enterprise applications. If we want to implement Connection pooling in enterprise applications we have to use the underlying application server provided Jdbc middleware server.

ResultSet:-

1. rs.next()
2. rs.previous()
3. rs.last()
4. rs.first()
5. rs.updateXXX()
6. rs.updateRow()
7. rs.absolute();
8. rs.beforeFirst()
9. rs.afterLast()
10. rs.getMetaData()

Statement:-

1. st.execute()
2. st.executeUpdate()
3. st.executeQuery()
4. st.addBatch()
5. st.executeBatch()

Connction:-

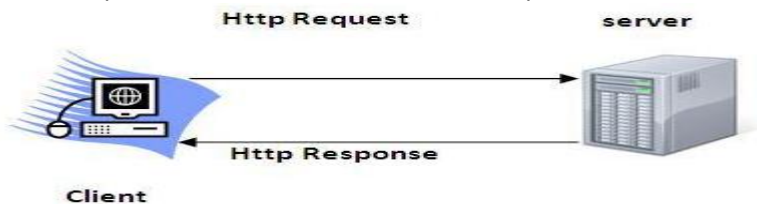
1. con.createStatement()
2. con.preapreStatement()
3. con.preapareCall()
4. con.commit()
5. con.rollback()
6. con.setAutoCommit()
7. con.getMetaData()

Servlets

- ✓ Servlet is a first web technology used to develop the web application.
- ✓ Servlet is an object executed at server side.
- ✓ Servlets is a server side technology used to write the programing at server side.
- ✓ Servlet is a part of the J2EE & these are executed by Servlet container (web container).
- ✓ The present version of the servlet is **Servlet 3.1** introduced in 2013 may compatable with jdk7.
- ✓ The **javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing servlets.

HTTP (Hyper Text Transfer Protocol):-

- ❖ HTTP is the foundation of data communication for the World Wide Web.
- ❖ HTTP functions as a request–response protocol in the client–server computing model.
- ❖ Client & server use HTTP protocol to provide the secure socket connection.
- ❖ It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol).
- ❖ HTTP protocol is stateless means each request is considered as the new request.



When we send the request to server by using client first Http protocol will trap that request it will prepare request format contains header & body fields

The header field contains information about the browser.

The body field contains the actual requested details.

When the server is sending response to client browser first Http protocol will take that response the it will prepare response format contains header & body fields.

The header filed contains information about the browser.

The body field contains actual response.

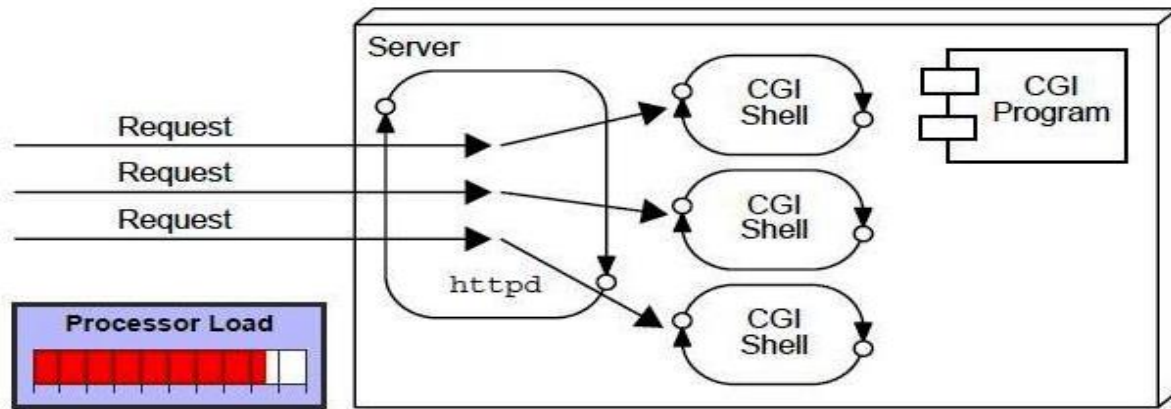
There are seven Http request types but commonly used request types are get & post.

HTTP Methods

- GET** → Request for a web page or an object from server
- PUT** → For sending a document to the server
- POST** → For sending data or information about client to the server
- DELETE** → Request to Delete an object on the server
- HEAD** → Request for information about a web page or a document
- TRACE** → Used to trace the proxies and tunnels in the path from client to server
- OPTION** → Used to determine server's capabilities

CGI(Common Gateway Interface):-

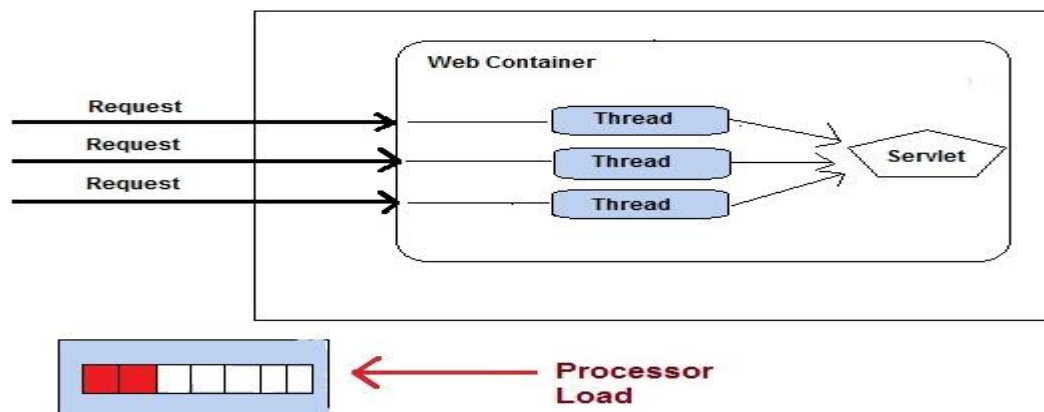
- ✓ Before servlet CGI(Common Gateway Interface) scripting language was popular at server side programming.
- ✓ The URL decides which CGI program to execute.
- ✓ Web Servers run the CGI program in separate OS shell. The shell includes OS environment and the process to execute code of the CGI program.
- ✓ The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.

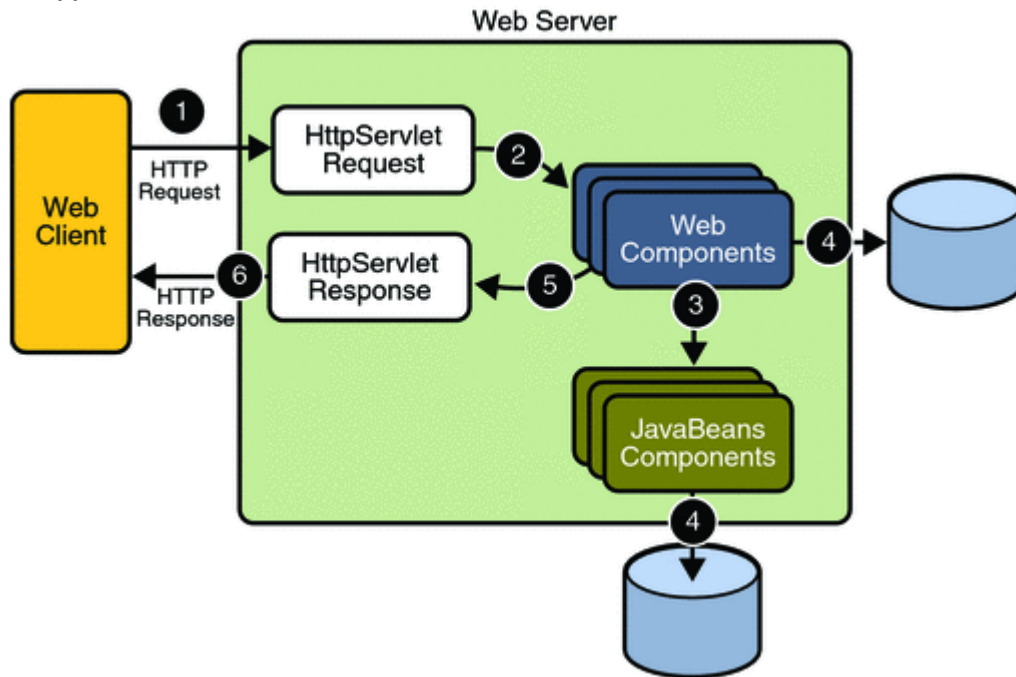
**Drawbacks if CGI:-**

- ❖ If number of clients is increases it will take more time to send response.
- ❖ For each request it starts a new process.
- ❖ It uses the platform dependent languages like C,CPP,Perl..etc

Advantages of using Servlets:-

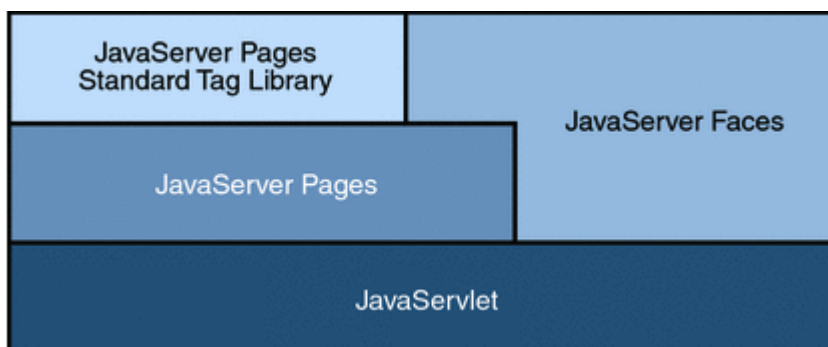
- 1) Less response time because each request runs on separate thread but not process.
 - a. Threads have lot of benefits over process like,
 - i. Threads are sharing common memory area.
 - ii. The thread is light weight.
 - iii. Costs of communication between threads are low.
- 2) Servlets are robust & object oriented.
- 3) Servlets are platform independent.
- 4) Servlets are secure because it uses java language.



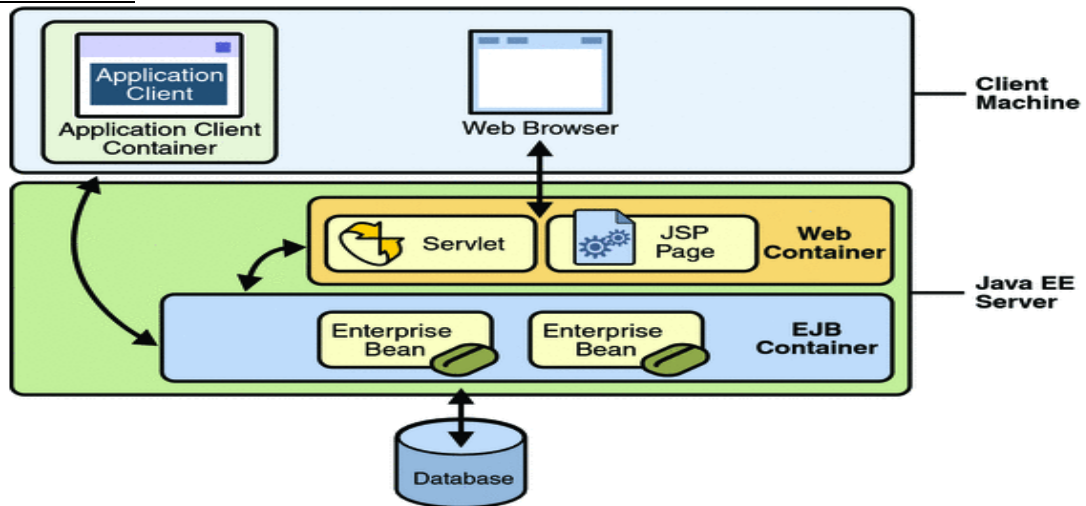
Web applications:-

✓ In above diagram the web components are either Java servlets, JSP pages..etc
The six steps are listed below.

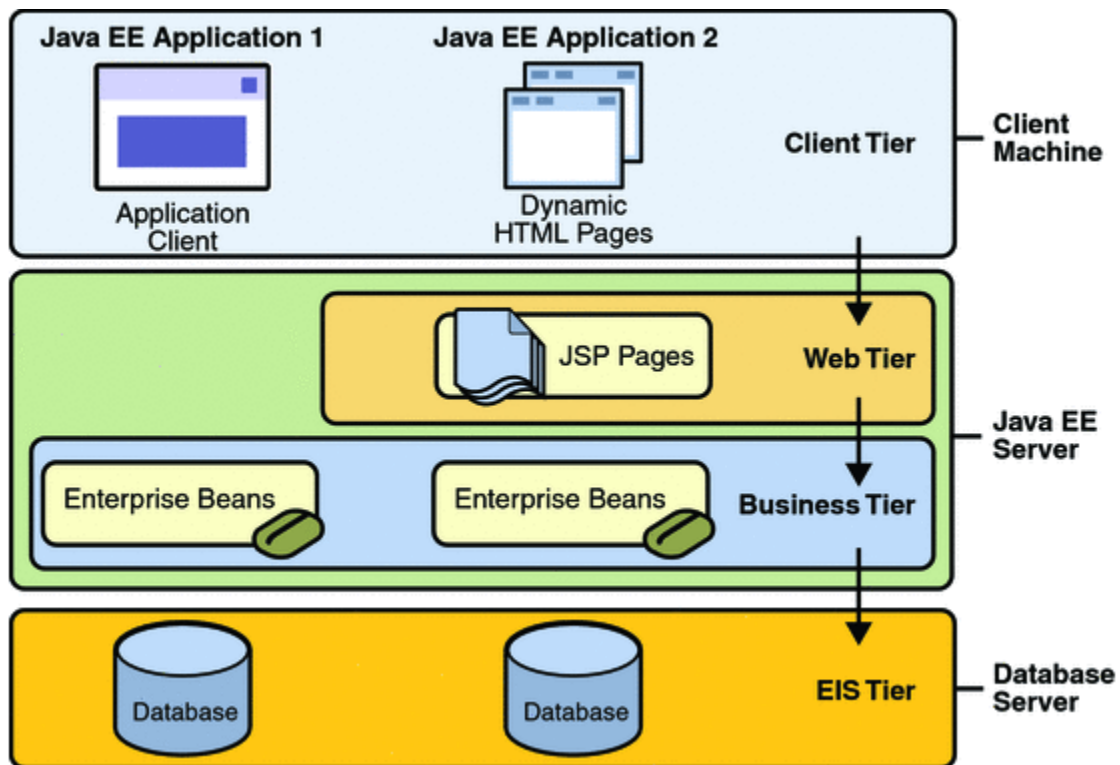
1. Client sends the Http request to the web server.
2. The HttpServletRequest object is delivered to the web component.
3. The web components are interacting with database to generate dynamic content.
4. The web components are interacting with java beans to generate dynamic content.
5. The web component generate the response in HttpServletResponse object.
6. The web server converts the HttpServletResponse into HTTP response and returns it to the client.

Java web applications technologies:-

Notice that Java Servlet technology is the foundation of all the web application technologies.

Container:-

Distributed Multitiered Java EE application:-



- ✓ *Client-tier components run on the client machine.*
- ✓ *Web-tier components run on the Java EE server.*
- ✓ *Business-tier components run on the Java EE server.*
- ✓ *Enterprise information system (EIS)-tier software runs on the EIS server.*

Servlet API history

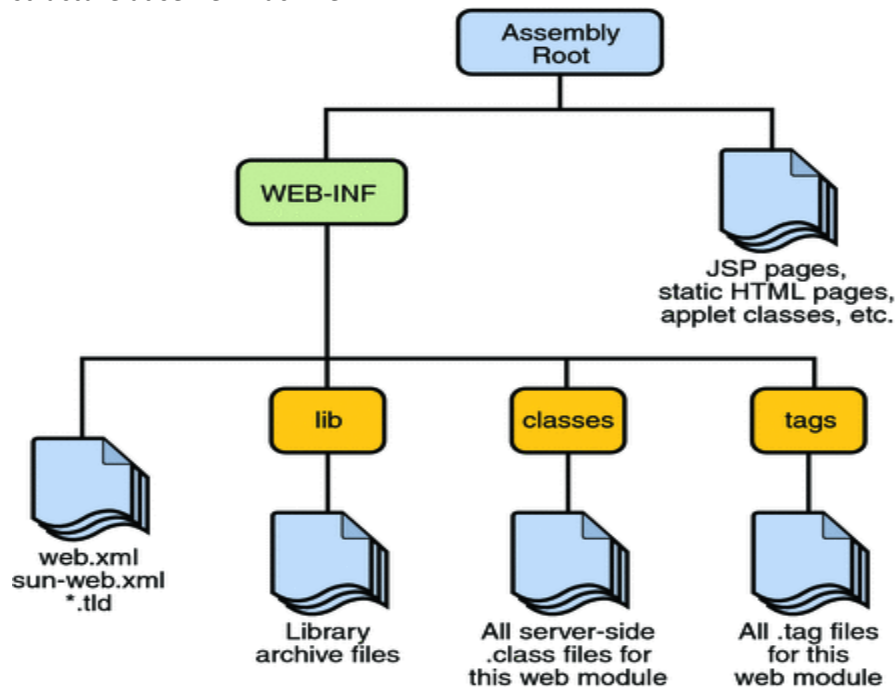
Servlet API version	Released	Platform	Important Changes
Servlet 3.1	May 2013	JavaEE 7	Non-blocking I/O, HTTP protocol upgrade mechanism (WebSocket) ^[5]
Servlet 3.0	December 2009	JavaEE 6, JavaSE 6	Pluggability, Ease of development, Async Servlet, Security, File Uploading
Servlet 2.5	September 2005	JavaEE 5, JavaSE 5	Requires JavaSE 5, supports annotation
Servlet 2.4	November 2003	J2EE 1.4, J2SE 1.3	web.xml uses XML Schema
Servlet 2.3	August 2001	J2EE 1.3, J2SE 1.2	Addition of <code>Filter</code>
Servlet 2.2	August 1999	J2EE 1.2, J2SE 1.2	Becomes part of J2EE, introduced independent web applications in .war files
Servlet 2.1	November 1998	Unspecified	First official specification, added <code>RequestDispatcher</code> , <code>ServletContext</code>
Servlet 2.0		JDK 1.1	Part of Java Servlet Development Kit 2.0
Servlet 1.0	June 1997		

Steps to design a first application:-

- Step 1** Prepare the web application directory structure.
Step 2 Prepare the form pages like html pages.
Step 3 Prepare the servlets.
Step 4 Prepare the web.xml file (Deployment Descriptor file).
Step 5 Deploy the project into server.
Step 6 open client browser access the application by using url.

Step 1: Web Application Directory Structure:

If we want to design any web application then we have to prepare the following directory structure at server machine.



web.xml: The web application deployment descriptor

Tag library descriptor files (see Tag Library Descriptors)

classes: A directory that contains server-side classes: servlets, utility classes, and JavaBeans components

tags: A directory that contains tag files, which are implementations of tag libraries(see Tag File Location)

lib: A directory that contains JAR archives of libraries called by server-side classes

1. Private area:

In web application directory structure, WEB-INF folder and its internal is treated as private area. If we deploy any resource under private area then client is unable to access that resource by using its name directly.

In web application directory structure always access the private area elements by using url pattern specified in web.xml

2. Public area:

The area which is in outside of WEB-INF folder and inside the application folder is treated as public area. If we deploy any resource under public area then client is able to access that resource by using its name directly.

Step 3: Prepare Web Resources (servlets)

There are three approaches to create a servlets in java

- 1) Implementing Servlet interface
- 2) Extending GenericServlet (abstract class)
- 3) Extending HttpServlet (abstract class)

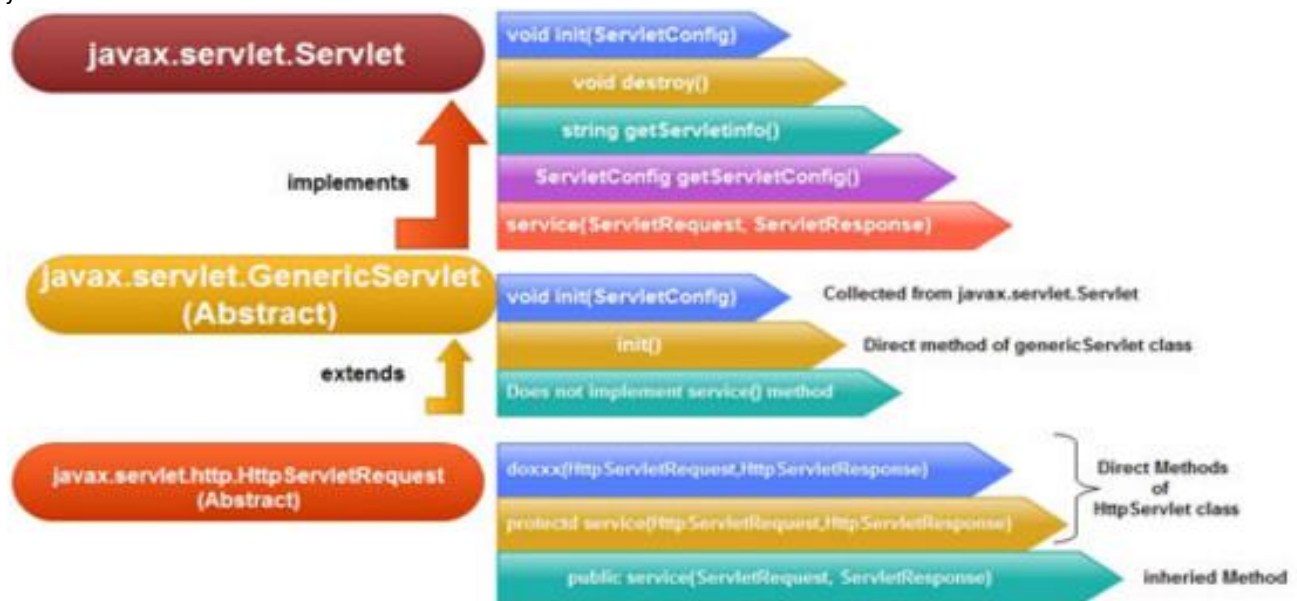
Approach 1: Implementing Servlet interface

```
package javax.servlet;
```

```
import java.io.IOException;
```

```
public interface Servlet
```

```
{ public abstract void init(ServletConfig servletconfig)throws ServletException;
  public abstract ServletConfig getServletConfig();
  public abstract void service(ServletRequest servletrequest, ServletResponse servletresponse)
  throws ServletException, IOException;
  public abstract String getServletInfo();
  public abstract void destroy();
}
```



To design servlets Servlet API has provided the following predefined library as part of `javax.servlet` package and `javax.servlet.http` package.

Q: What is Servlet? and in how many ways we are able to prepare servlets?

Ans: Servlet is an object available at server machine which must implement either directly or indirectly Servlet interface.

As per the predefined library provided by Servlet API, there are 3 ways to prepare servlets.

1. Implementing Servlet interface:

In this approach, if we want to prepare servlet then we have to take an user defined class which must implement Servlet interface.

```
public class MyServlet implements Servlet
{ -----
```

```

}

```

2. Extending GenericServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to GenericServlet abstract class.

```

public class MyServlet implements GenericServlet
{
    -----
}

```

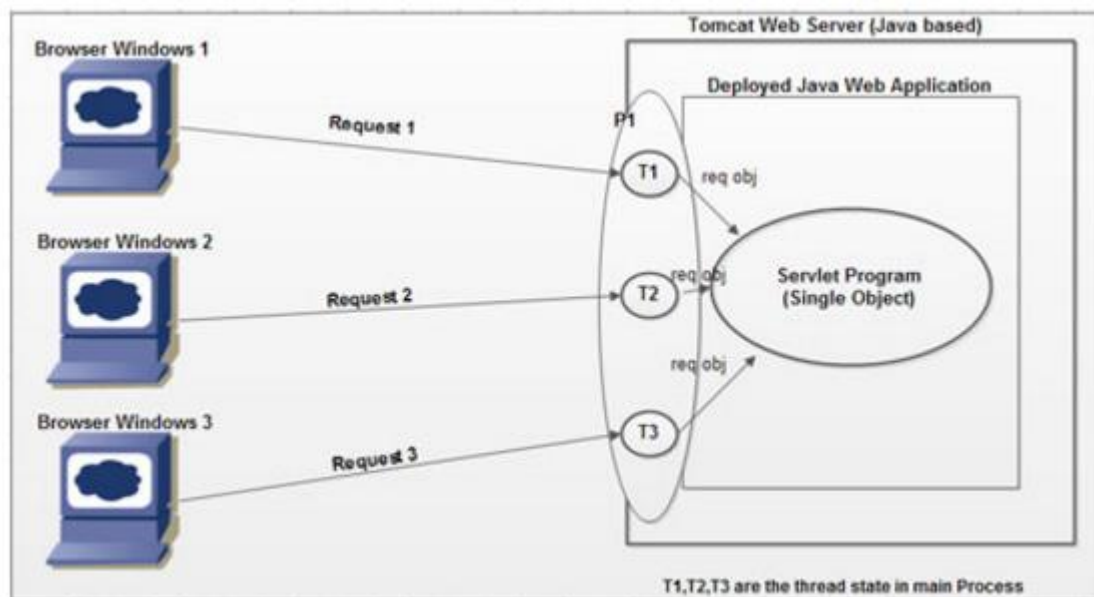
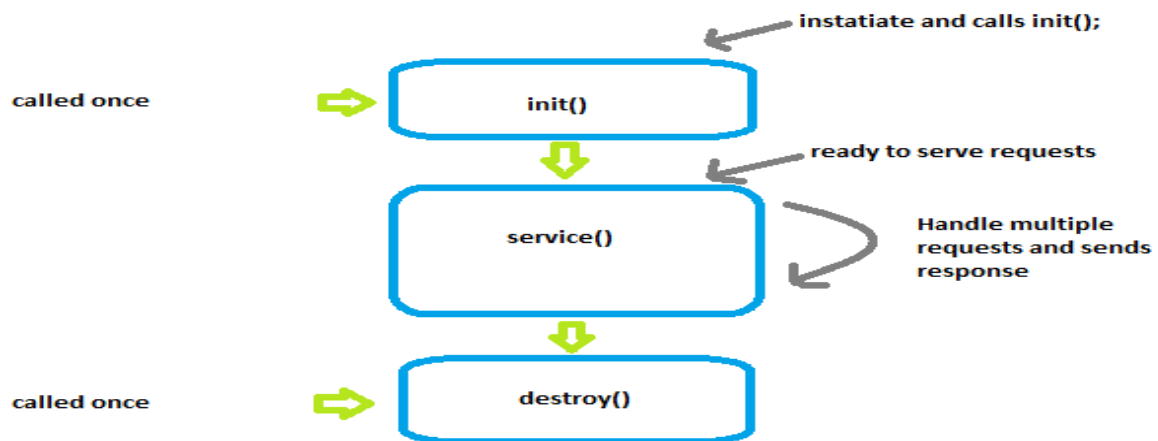
3. Extending HttpServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to HttpServlet abstract class.

```

public class MyServlet implements HttpServlet
{
    -----
}

```



Step 4: Start the Server:

There are 3 ways to start the server.

Execute either startup.bat file or Tomcat7 Service Runner available under bin folder of Tomcat server.

C:\Tomcat 7.0\bin
 Use system program monitor Tomcat
 Start All Programs Apache Tomcat 7.0 Monitor Tomcat →
 Use Apache Tomcat System Service
 Start Type services.MVC in search field Select Apache Tomcat 7.0
 Select Start Service Icon.

Step 5: Access the Web Application:

There are 2 ways to access the web applications.

Open web browser and type the complete URL on address bar.

http://localhost:8080/app1/servlet

Open web browser type the URL up to

http://localhost:8080

If we do the above automatically the Tomcat Home Page will be opened, where we have to select Manager Applications, provide username and password in Security window and click on OK button.

If we do the above automatically list of applications will be opened, where we have to select the required application.

First Approach to Design Servlets (Implementing Servlet interface):

If we want to design a servlet with this approach then we have to take an user defined class it should be an implementation class to Servlet interface.

Where the purpose of init(_) method to provide servlets initialization.

Note: In servlets execution, to perform servlets initialization container has to access init(_) method. To access init(_) method container has to create ServletConfig object.

ServletConfig is an object, it will manage all the configuration details of a particular servlet like logical name of servlet, initialization parameters and so on.

In servlet, the main purpose of service(__,__) method is to accommodate the complete logic and to process the request by executing application logic.

Note: In servlet, service(__,__) method is almost all same as main(_) method in normal Java application.

When we send a request from client to server then container will access service(__,__) method automatically to process the request, where the purpose of getServletConfig() method is to get ServletConfig object at servlet initialization.

Where getServletInfo() method is used to return generalized description about the present servlet.

Where destroy() method can be used to perform servlet reinstantiation.

To prepare application logic in service(__,__) method we have to use the following conventions.

Step 2: Deployment Descriptor or web.xml file:

Deployment Descriptor is web.xml file, it can be used to provide the metadata about the present web application required by the container in order to perform a particular server side action.

In web applications, web.xml file include the following configuration details w.r.t the web application

Welcome Files Configuration

Display Name Configuration

Servlets Configuration

Filters Configuration

Listeners Configuration

Context Parameters Configuration

Initialization Parameters Configuration

Session Time Out Configuration

Load On Startup Configuration

Error Page Configuration
Tag Library Configuration
Security Configuration

In general in web applications, we will deploy the servlets .class files under classes folder of the web application directory structure i.e. private area.

If we deploy any resource under private area then client is unable to access that resource through its name, client is able to access that resource through alias names or locators. In case of servlets, client is able to access servlet classes through the locators called as URL Patterns.

If we provide multiple number of servlets under classes folder and we provide a particular request to a particular servlet available under classes folder with an URL pattern then container should require mapping details between URL patterns and servlets class names as metadata in order to identify w.r.t servlet.

In the above context, to provide the required metadata to the container we have to provide servlet configuration in web.xml file. To provide servlet configuration in web.xml file we have to use the following xml tags.

```
<web-app>
-----
<servlet>
  <servlet-name>logical_name</servlet-name>
  <servlet-class>fully qualified name of servlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>logical_name</servlet-name>
  <url-pattern>urlpattern_name</url-pattern>
</servlet-mapping>
-----
</web-app>
```

Note: In the above servlets configuration, <servlet-name> tag value under <servlet> tag and <servlet-mapping> tag must be same.

Ex: <web-app>

```
<servlet>
  <servlet-name>loginservlet</servlet-name>
  <servlet-class>com.dss.login.LoginServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>loginservlet</servlet-name>
  <url-pattern>/login</url-pattern>
</servlet-mapping>

</web-app>
```

If we want to access the above servlet then we have to provide the following URL at client browser.

http://localhost:8080/loginapp/login

In servlet configuration, there are 3 ways to define URL patterns.

- Exact Match Method

- Directory Match Method
- Extension Match Method

1. Exact Match Method:

In Exact Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with forward slash("/") and pattern name may be anything.

Ex: `<url-pattern>/abc/xyz</url-pattern>`

If we define any URL pattern with exact match method then to access the respective resource we have to provide an URL pattern at client address bar along with URL, it must be matched with the URL pattern which we defined in web.xml file.

Ex: `http://localhost:8080/app1/abc/xyz` Valid
`http://localhost:8080/app1/xyz/abc` Invalid
`http://localhost:8080/app1/xyz` Invalid
`http://localhost:8080/app1/abc` Invalid

Note: In general in web applications, we will prefer to use exact match method to define an URL pattern for a particular servlet when we have a requirement to access respective servlet independently.

2. Directory Match Method:

In Directory Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with forward slash("/") and it must be terminated with "*".

Ex: `<url-pattern>/abc/*</url-pattern>`

If we define any URL pattern with this method then to access the respective resource from client we have to specify an URL pattern at client address bar it should match its prefix value with the prefix value of the URL pattern defined in web.xml file.

Ex: `http://localhost:8080/app1/abc/xyz` Valid
`http://localhost:8080/app1/xyz/abc` Invalid
`http://localhost:8080/app1/abc` Valid
`http://localhost:8080/app1/abc/abc` Valid

Note 1: In general in web applications, we will prefer to use directory match method to define an URL pattern when we have a requirement to pass multiple number of requests to a particular server side resource.

Note 2: In web applications we will use Filters to provide preprocessing and post processing to one or more number of servlets. In this context, when we send a request to respective servlet then container will bypass all the requests to the respective Filter.

To achieve this type of requirement we have to use directory match method to define an URL pattern for the respective Filter.

3. Extension Match Method:

In Extension Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with "*" and it must be terminated with a particular extension.

Ex: `<url-pattern>*.do</url-pattern>`

If we define an URL pattern with this method then to access the respective server side resource from client we have to specify an URL pattern at client address bar, it may start with anything, but must be terminated with an extension which was specified in web.xml file.

Ex: `http://localhost:8080/app1/login.do` Valid
`http://localhost:8080/app1/reg.do` Valid
`http://localhost:8080/app1/add.xyz` Invalid
`http://localhost:8080/app1/search.doo` Invalid

Note 1: In general in web applications, we will prefer to use extension match method to define an URL pattern when we have a requirement to trap all the requests to a particular server side resource and to perform the respective server side action on the basis of the URL pattern name if we provided at client browser.

Note 2: If we design any web application on the basis of MVC then we have to use a servlet as controller, where the controller servlet has to trap all the requests and it has to perform a particular action on the basis of URL pattern name. Here to define URL pattern for the controller servlet we have to use extension method.

In web applications, web.xml file is mandatory or optional is completely depending on the server which we used.

In Apache Tomcat Server, web.xml file is optional when we have not used servlets, filters and so on. In Weblogic Server, web.xml file is mandatory irrespective of using servlets, filters and so on.

Up to servlets2.5 version[J2EE5.0] it is mandatory to provide web.xml file if we use servlets, listeners, filters and so on in our application. But in servlets3.0 version, there is a replacement for web.xml file i.e. Annotations, Annotations will make web.xml file is optional.

In web applications, it is not possible to change the name and location of the deployment descriptor because container will search for deployment descriptor with web.xml name under WEB-INF folder as per its predefined implementation.

1. Specify response content type:

To specify response content type to the client we have to use the following method from ServletResponse.

```
public void setContentType(String MIME_TYPE)
```

where MIME_TYPE may be text/html, text/xml, image/jpeg, img/jpg and so on.

Ex: res.setContentType("text/html");

Note: The default content type in servlets is text/html.

When container encounters the above method then container will pick up the specified MIME_TYPE and container will set that value to content type response header in the response format.

When protocol dispatch the response format to client, before getting the response from response format body part first client will pick up content type response header value i.e. MIME_TYPE, client will prepare itself to hold the response as per the MIME_TYPE.

2. While executing the servlet we have to generate some dynamic response on response object, where to carry the dynamic response to the response object Servlet API has provide a predefined PrintWriter object internally.

To get the predefined PrintWriter object we have to use the following method from ServletResponse.

```
public PrintWriter getWriter()
```

Ex: `PrintWriter out=res.getWriter();`

Http protocol has provided the following http methods along with http1.0 version.

1. GET
2. POST
3. HEAD

Http protocol has provided the following http methods as per http1.1 version.

1. OPTIONS
2. PUT
3. TRACE
4. DELETE

Http1.1 version has provided a reserved http method i.e. CONNECT.

doGet() vs doPost():-

GET:-

1. The default request type is get request.
2. In GET request the user entered information is appended to the URL in a query string.
3. The query string visible in the URL pattern so GET request is able to provide less request.
4. By using GET request we the client is able to send limited amount of data. GET request doesn't have body in the request format hence we are able to send the all parameter values in the form of header part so by using GET request we are able to send limited amount of data(maximum 256 number of characters at a time)
5. In general in web applications GET request can be used to get the information from the server.

POST:-

1. POST is not default request.
2. POST request the user entered information is sent as data not appended to URL.
3. POST is send the data client to server hence it is able to provide very good security .
4. POST request contains body part hence we are able to send the large amount of data client to server by using body part of POST request.
5. In general in web applications POST request can be used to send the data to the server.

Content type:-

It is also known as MIME type. It is part of http header it is providing description about which type of data sending to the browser.

Ex:- text/html

To represent the content type we have to use following method.

`res.setContentType("text/html");`

Approach 1:-

Take user defined class which implements Servlet Interface

Class NyServlet implements Servlet

```
{  
}
```

MyServlet.java:-

```
package com.dss;
```

```
import java.io.IOException;
```

```
import java.io.PrintWriter;
```

```
import javax.servlet.Servlet;
```

```
import javax.servlet.ServletConfig;
```

```
import javax.servlet.ServletException;
```

```
import javax.servlet.ServletRequest;
```

```
import javax.servlet.ServletResponse;
```

```
public class MyServlet implements Servlet {
```

```
    ServletConfig config;
```

```
    public MyServlet() {
```

```
    }
```

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

```
            this.config=config;
```

```
            System.out.println("init method");
```

```
        }
```

```
        public ServletConfig getServletConfig() {
```

```
            System.out.println("getServletConfig method");
```

```
            return config;
```

```
        }
```

```
        public String getServletInfo() {
```

```
            return "config";
```

```
        }
```

```
        public void service(ServletRequest request, ServletResponse response) throws ServletException,  
IOException {
```

```
            System.out.println("service method");
```

```
            response.setContentType("text/html");
```

```
            PrintWriter writer=response.getWriter();
```

```
            writer.print("<html><body>");
```

```
            writer.print("<b>hi ratan</b>");
```

```
            writer.print("</body></html>");
```

```
        }
```

```
        public void destroy() {
```

```
            System.out.println("destroy method");
```

```
        }
```

```
}
```

Web.xml:-

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
    <display-name>app1</display-name>
    <servlet>
        <display-name>MyServlet</display-name>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>
            com.dss.MyServlet</servlet-class>
        </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/first</url-pattern>
    </servlet-mapping>
</web-app>
```

2nd approach to create a servlet:-

abstract class GenericServlet implements Servlet,ServletConfig,Serializable

```
{
}
```

Take user defined class which extends GenericServlet abstract class .

Class MyServlet extends GenericServlet

```
{
}
```

1. GenericServlet is providing all implementation of Servlet interface methods except service() method.
2. Generic servlet is able to handle any type of request hence it is protocol independent

MyServlet.java:-

```
package com.dss;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class MyServlet extends GenericServlet {
    public void service(ServletRequest request, ServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter writer=response.getWriter();
        writer.print("<b>hi ratan how r u</b>");
        writer.print("<b>this is GenericServlet example</b>");
    }
}
```

```

    }

}
Web.xml:-
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
    <display-name>app2</display-name>
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>com.dss.MyServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
</web-app>

```

3rd approach to create a servlet:-

Class HttpServlet extends GenericServlet implements Serializable

```

{
}

```

Take user defined class extends HttpServlet

Class MyServlet extends HttpServlet

```

{
}

```

HttpServletDemo.java:-

GETrequest(<http://localhost:9999/LoginServlet/login?uname=ratan&upwd=ratan>)

package com.dss;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class HttpServletDemo **extends** HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) **throws**
ServletException, IOException {

 response.setContentType("text/html");

 PrintWriter writer=response.getWriter();

 writer.print("hi ratan how r u");

 writer.print("this is HttpServletDemo GET request example");

```

    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer=response.getWriter();
        writer.print("<b>hi ratan how r u</b>");
        writer.print("<b>this is HttpServletDemo POST request example</b>");
    }
}

```

Web.xml:-

```

<web-app>
    <display-name>app3</display-name>
    <servlet>
        <servlet-name>HttpServletDemo</servlet-name>
        <servlet-class> com.dss.HttpServletDemo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HttpServletDemo</servlet-name>
        <url-pattern>/MyServlet</url-pattern>
    </servlet-mapping>
</web-app>

```

POST request:-(http://localhost:9999/LoginServlet/login)

```
package com.dss;
```

```
import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```

public class HttpServletDemo extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer=response.getWriter();
        writer.print("<b>hi ratan how r u</b>");
        writer.print("<b>this is HttpServletDemo GET request example</b>");
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer=response.getWriter();
        writer.print("<b>hi ratan how r u</b>");
        writer.print("<b>this is HttpServletDemo POST request example</b>");
    }
}

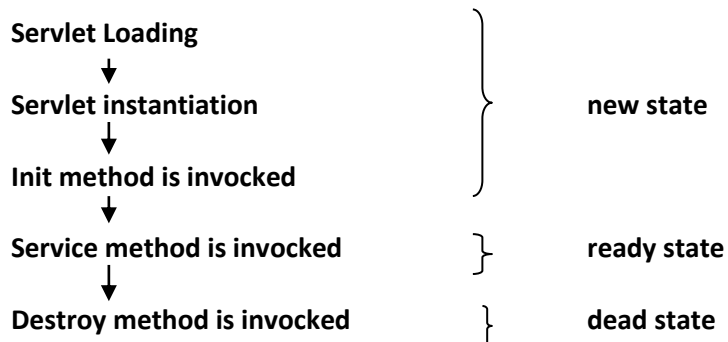
```

Post.html:- To specify the post request

```
<html>
<head><center>POST REQUEST</center></head>
<body>
<form method="POST" action="/first">
<input type="submit" value="lick me" >
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
  <display-name>app3</display-name>
  <servlet>
    <servlet-name>HttpServletDemo</servlet-name>
    <servlet-class>com.dss.HttpServletDemo</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HttpServletDemo</servlet-name>
    <url-pattern>/first</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>post.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Servlet life cycle:-

Servlets Flow of Execution:

When we start the server the main job of container is to recognize each and every web application and to prepare ServletContext object to each and every web application.

While recognizing web application container will recognize web.xml file under WEB-INF folder then perform loading, parsing and reading the content of web.xml file.

While reading the content of web.xml file, if container identifies any context data in web.xml file then container will store context data in ServletContext object at the time of creation.

After the server startup when we send a request from client to server protocol will pick up the request then perform the following actions.

1. *Protocol will establish virtual socket connection between client and server as part of the server IP address and port number which we specified in the URL.*
2. *Protocol will prepare a request format having request header part and body part, where header part will maintain request headers and body part will maintain request parameters provided by the user.*
3. *After getting the request format protocol will carry request format to the main server.*

Upon receiving the request from protocol main server will check whether the request data is in well-formed format or not, if it is in well-formed then the main server will bypass request to container.

Upon receiving the request from main server container will pick up application name and resource name from request and check whether the resource is for any html page or Jsp page or an URL pattern for a servlet.

If the resource name is any html page or Jsp page then container will pick up them application folder and send them as a response to client.

If the resource name is an URL pattern for a particular servlet available under classes folder then container will go to web.xml file identifies the respective servlet class name on the basis of the URL pattern.

After identifying the servlet class name from web.xml file container will recognize the respective servlet .class file under classes folder then perform the following actions.

Step 1: Servlet Loading:

Here container will load the respective servlet class byte code to the memory.

Step 2: Servlet Instantiation: Here container will create a object for the loaded servlet.

Step 3: Servlet Initialization:

Here container will create ServletConfig object and access init(_) method by passing ServletConfig object reference.

Step 4: Creating request and response objects(Request Processing):

After the servlet initialization container will create a thread to access service(__,__) method, for this container has to create request and response objects.

Step 5: Generating Dynamic response:

By passing request and response objects references as parameters to the service(__,__) method then container will access service(__,__) method, executes application logic and generate the required response on response object.

Step 6: Dispatching Dynamic response to Client:

When container generated thread reaching to the ending point of service(__,__) method then container will keep the thread in dead state, with this container will dispatch the dynamic response to main server from response object, where main server will bypass the response to the protocol.

When protocol receives the response from main server then protocol will prepare response format with header part and body part, where header part will manage all the response headers and body part will manage the actual dynamic response.

After getting response format protocol will carry that response format to the client.

Step 7: Destroying request and response objects:

When the response is reached to the client protocol will terminate the virtual socket connection between client and server, with this container destroy the request and response objects.

Step 8: Servlet Deinstantiation:

When container destroy request and response objects then container will go to the waiting state depends on the container implementation, if container identifies no further request for the same resource then container will destroy servlet object.

Note: In servlet execution, container will destroy ServletConfig object just before destroying the servlet object.

Step 9: Servlet Unloading:

After servlet deinstantiation container will eliminate the loaded servlet byte code from operational memory.

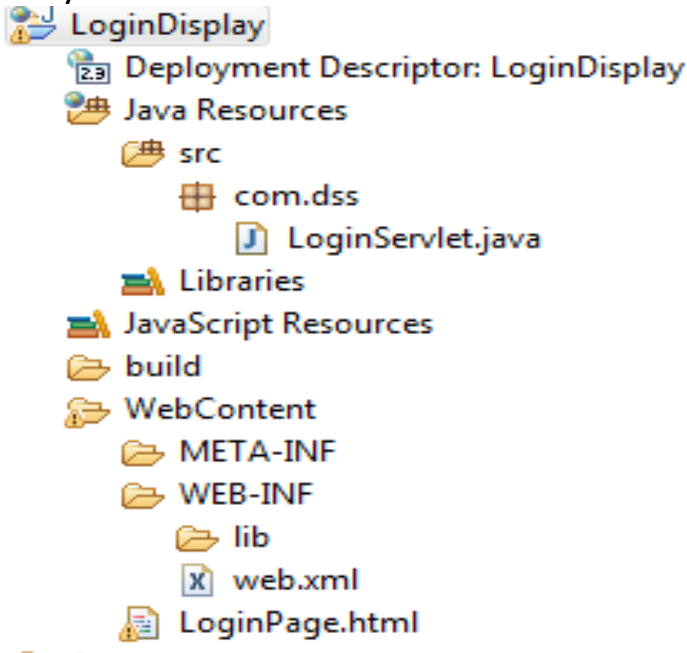
Step 10: Destroying ServletContext object:

In the above servlet life cycle, all the objects like request, response and ServletConfig are destroyed before servlet deinstantiation, but still Servlet object is available in memory.

In general ServletContext object will be destroyed at the time of server shut down.

Display login details:-

Directory structure:-



Web.xml:-

```
<web-app>
  <display-name>App1</display-name>
  <servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.dss.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>LoginPage.html</welcome-file>
  </welcome-file-list>
</web-app>
```

LoginPage.html:-

```
<html>
<head>
<h1><center>login page</center>
</head>
<body>
<form method="get" action="/LoginServlet">
```



```
<pre>
user name<input type="text" name="uname"/>
password<input type="password" name="upwd"/>
<input type="submit" value="login"/>
</pre>
</form>
</body>
</html>
```

LoginServlet.java:-

```
package com.dss;
```

```
import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

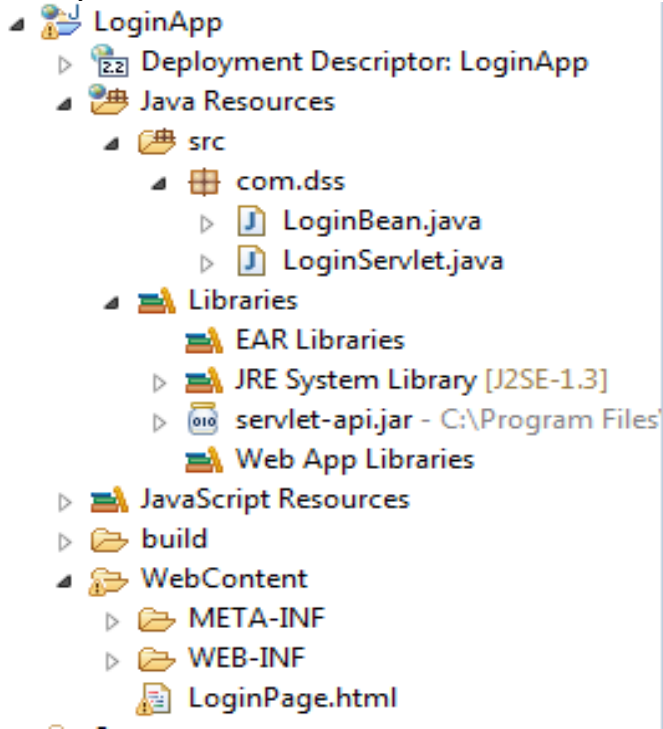
```
public class LoginServlet extends HttpServlet {
    String status;
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    IOException {
        response.setContentType("text/html");
        PrintWriter writer=response.getWriter();

        String uname=request.getParameter("uname");
        String upwd=request.getParameter("upwd");
        writer.println("<html>");
        writer.println("<body>");
        writer.println("username is"+uname);
        writer.println("<br>");
        writer.println("user password is"+upwd);
        writer.println("</body>");
        writer.println("</html>");

    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    }
}
```

LoginServlet application used to store the data into the database:-

Directory structure:-



LoginPage.html:-

```
<html>
<head>
<h1><center>login page</center>
</head>
<body>
<form method="get" action="./LoginServlet">
<pre>
user name<input type="text" name="uname"/>
password<input type="password" name="upwd"/>
<input type="submit" value="login"/>
</pre>
</form>
</body>
</html>
```

Web.xml:

```
<web-app>
  <display-name>App1</display-name>
```

```
<servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.dss.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>LoginPage.html</welcome-file>
</welcome-file-list>
</web-app>
```

LoginServlet.java:-

```
package com.dss;
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
```

```
    String status;
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
```

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

```
    String uname=request.getParameter("uname");
    String upwd=request.getParameter("upwd");
    writer.println("<html>");
    writer.println("<body>");
    writer.println("username is"+uname);
    writer.println("<br>");
    writer.println("user password is"+upwd);
    writer.println("</body>");
    writer.println("</html>");
```

```
    LoginBean lb=new LoginBean();
```

```
    try {
        lb.checkLogin(uname,upwd);
    } catch (SQLException e) {
        e.printStackTrace();
    }
```

```
}
```

```
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
    }
```

```
}
```

LoginBean.java:-

```
package com.dss;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.SQLException;
```

```
import java.sql.Statement;
```

```
public class LoginBean {
```

```
    static Statement statement;
```

```
    static
```

```
    {
```

```
        try {
```

```
            Class.forName("sun.jdbc.odbc.JdbcOdbc");
```

```
            Connection
```

```
connection=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
```

```
            statement=connection.createStatement();
```

```
        } catch (ClassNotFoundException e) {  
            e.printStackTrace();
```

```
        } catch (SQLException e) {  
            e.printStackTrace();
```

```
        }
```

```
        System.out.println("connection is created successfully");
```

```
    }
```

```
    public void checkLogin(String uname, String upwd) throws SQLException {
```

```
        String query="insert into eee values('"+uname+"','"+upwd+"')";
```

```
        statement.executeUpdate(query);
```

```
        System.out.println("vales are inserted successfully");
```

```
    }
```

```
}
```

Registration application:-**Registration.html:-**

```

<html>
<head>
<center>Registration</center>
</head>
<body>
<form method="post" action="/Registration">
<pre>
Name          <input type="text" name="uname"/>
Password      <input type="password" name="upwd"/>
Qualification  <input type="checkbox" name="uqual" value="BSC"/>BSC
               <input type="checkbox" name="uqual" value="MCA"/>MCA
               <input type="checkbox" name="uqual" value="M TECH"/>M
               TECH
Gender         <input type="radio" name="ugen" value="MALE"/>MALE
               <input type="radio" name="ugen" value="FEMALE"/>FEMALE
Technologies   <select name="utech" size="3" multiple>
               <option value="C">C</option>
               <option value="C++">C++</option>
               <option value="JAVA">JAVA</option>
               </select>
Comments      <input type="textarea" name="ucom" rows="5"
cols="25"></textarea>
               <input type="submit" value="Register"/>
</pre>
</form>
</body>
</html>

```

Web.xml:-

```

<web-app>
  <display-name>Registration</display-name>
  <servlet>
    <servlet-name>Registration</servlet-name>
    <servlet-class>com.dss.Registration</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Registration</servlet-name>
    <url-pattern>/Registration</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>Registration.html</welcome-file>
  </welcome-file-list>
</web-app>

```

Registration.java:-

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

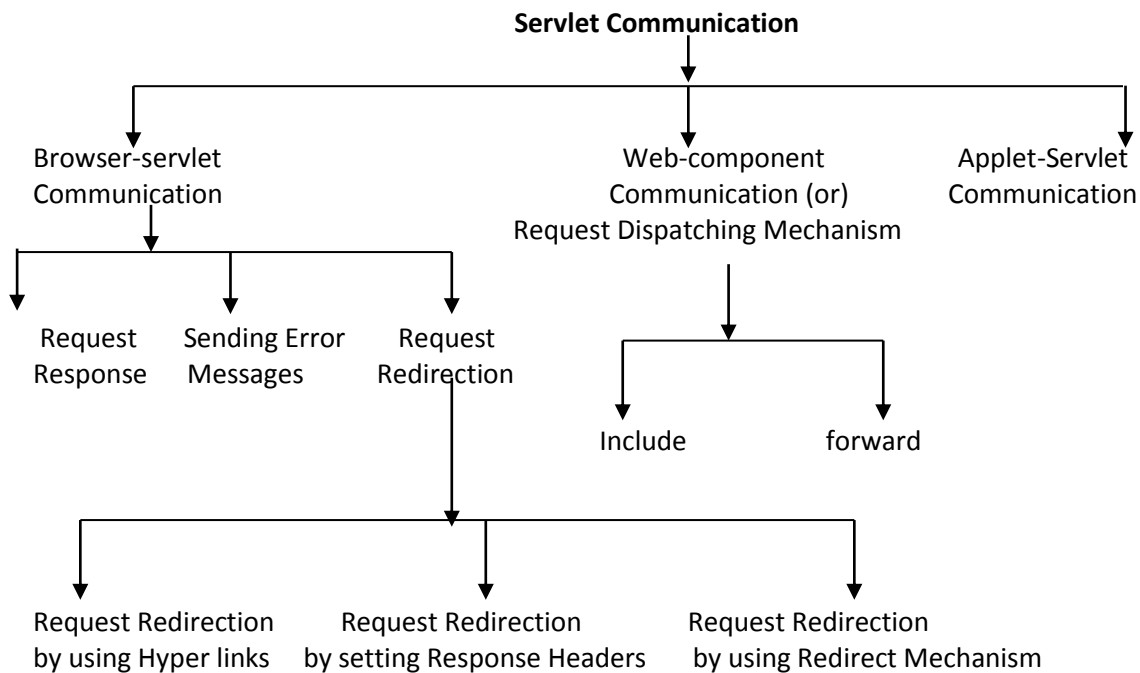
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Registration extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String uname=request.getParameter("uname");
        String upwd=request.getParameter("upwd");
        String[] uqual=request.getParameterValues("uqual");
        String ugen=request.getParameter("ugen");
        String[] utech=request.getParameterValues("utech");
        String ucom=request.getParameter("ucom");
        out.println("<html>");
        out.println("<body bgcolor='lightgreen'>");
        out.println("<center><b><font size='6'>");
        out.println("Name....."+uname);
        out.println("<br><br>");
        out.println("Password....."+upwd);
        out.println("<br><br>");
        out.println("Qualification<br><br>");
        for(int i=0;i<uqual.length;i++)
        {
            out.println(uqual[i]);
            out.println("<br><br>");
        }
        out.println("Gender...."+ugen);
        out.println("<br><br>");
        out.println("Technologies<br><br>");
        for(int i=0;i<utech.length;i++)
        {
            out.println(utech[i]);
            out.println("<br><br>");
        }
        out.println("Comments....."+ucom);
    }
}
```

```

        out.println("<br><br>");
        out.println("Congratulations....."+uname);
        out.println("<br><br>U R Registration Success");
        out.println("</font></b></center></body></html>");
    }
}

```



request redirection by using Hyper-link mechanism:-

welcome.html:-

```

<HTML>
<HEAD>
Request Redirection
</HEAD>
<BODY>
<form method="post" action="hutch">
to send the request:-<input type="submit" value="click me">
</BODY>
</HTML>

```

Hutch.java:-

```

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

```

```

        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println("<html>");
        out.println("<body bgcolor='lightyellow'>");
        out.println("<center><b><font size='6'>");
        out.println("<br><br>");
        out.println("To Get Services From Hutch Click");

        out.println("<a
href='http://localhost:9999/vodaphoneapp/welcome.html'>CustomerCare@www.vodaphone.com</a>"
);
        out.println("</font></b></center></body></html>");
    }
}

```

Web.xml:-

```

<web-app>
<welcome-file-list>
<welcome-file>welcome.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>h</servlet-name>
<servlet-class>Hutch</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>h</servlet-name>
<url-pattern>/hutch</url-pattern>
</servlet-mapping>
</web-app>

```

Send redirect by using response-header mechanism:-

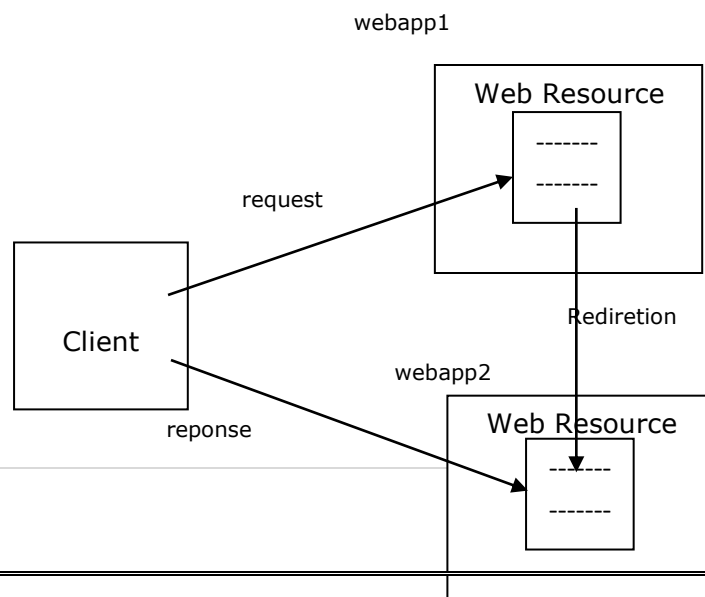
```

res.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
res.setHeader("location","http://www.facebook.com");

```

Send redirect by using send-redirect mechanism:-

By using send-redirect we are able to redirect the request to within the server and outside of the server.




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

LoginServlet.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // TODO Auto-generated method stub
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        String uname = request.getParameter("uname");
        String upwd = request.getParameter("upwd");

        if(uname.equals("sravva")&& upwd.equals("infotech"))
        {
            /*writer.println("login success");
            writer.println("<a href='http://facebook.com'>click the link to redirect req to fb</a>");
            */
            /*response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
            response.setHeader("location", "https://www.google.co.in");*/

            response.sendRedirect("http://facebook.com");
        }
        else
        {
            response.sendError(408, "login fail try once again");
        }
    }
}
```

Login.html:-

```
<html>
<body>
<form action="/LoginServlet">
user name: <input type="text" name="uname"/><br/>
password : <input type="password" name="upwd"/><br/>
<input type="submit" value="login"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <display-name>LoginServlet</display-name>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.sravya.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

RequestDispatcher:-

The requestDispatcher is used to dispatch the request to another resources(html,servlets,jsp) and it is include the content of another resource.

There are two methods are available in the RequestDispatcher Object

1. Public void Include(ServletRequest request,ServletResponse response)throws ServletException,IOException

By using above method we are able to transfer the request from one resource to another resources like html,servlets,jsp

2. Public void Forward (ServletRequest request,ServletResponse response)throws ServletException,IOException

By using above method we are able to include the target resource response.

Step 1: get RequestDispatcher object

RequestDispatcher is an object, it will provide very good environment either to include the target resource response into the present resource response or to forward request from present resource to the target resource.

To get RequestDispatcher object we will use the following 2 ways.

1. ServletContext
 1. getRequestDispatcher(_) method
 2. getNamedDispatcher(_) method
2. ServletRequest
 1. getRequestDispatcher(_) method

getRequestDispatcher(_) vs getNamedDispatcher(_):-

Both the methods are used to get the RequestDispatcher object.

To get RequestDispatcher object, if we use getRequestDispatcher(_) method then we should pass the locator of target resource as parameter.

Note : In case of the servlet, url pattern is treated as locator.

public RequestDispatcher getRequestDispatcher(String path)

To get RequestDispatcher object, if we use `getNamedDispatcher(_)` method then we have to pass logical name of target resource as parameter.

Note : In case of the servlet, logical name is a name specified along with `<servlet-name>` tag in web.xml file.

```
public RequestDispatcher getNamedDispatcher(String path)
```

What is the difference between `getRequestDispatcher(_)` method ServletContext and ServletRequest?

Both the methods can be used to get the RequestDispatcher object.

To get RequestDispatcher object, if we use `getRequestDispatcher(_)` method from ServletContext then we must pass the relative path of target resource.

To get RequestDispatcher object, if we use `getRequestDispatcher(_)` method from ServletRequest then we have to pass either relative path or absolute path of target resource.

Note: In general, relative path should prefix with forward slash("/") and absolute path should not prefix with forward slash("/").

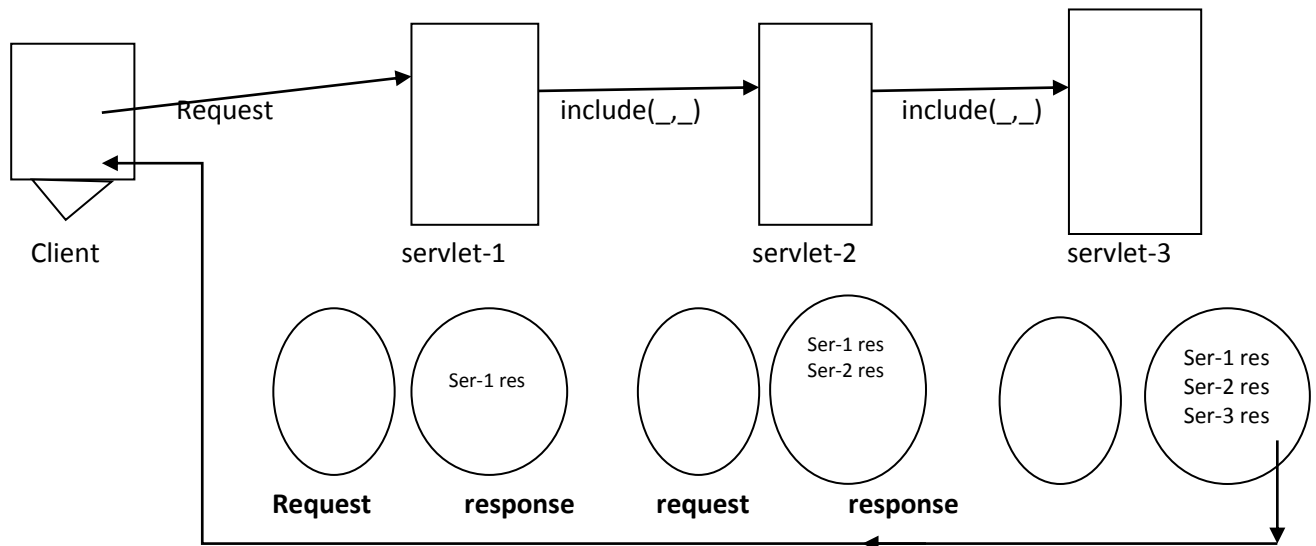
Step 2: Apply either Include mechanism or Forward mechanism to achieve Web-Component Communication:

To represent Include and Forward mechanisms RequestDispatcher has provided the following methods.

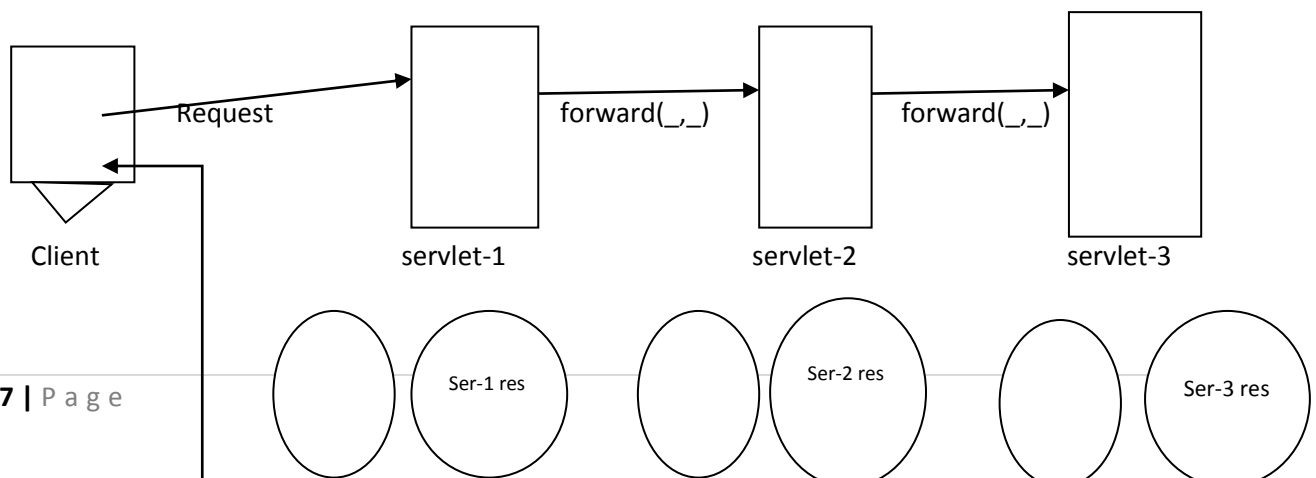
```
public void include(ServletRequest req, ServletResponse res)throws SE, IOE
```

```
public void forward(ServletRequest req, ServletResponse res)throws SE, IOE
```

Include():-



forward():-





Login.html:-

```
<html>
<body bgcolor='red'>
<form action="./login" method="post">
Name:<input type="text" name="uname"/><br/>
Password:<input type="password" name="upwd"/><br/>
<input type="submit" value="login"/>
</form>
</body>
</html>
```

Servlet1.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;

public class Servlet1 extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

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

        String upwd=request.getParameter("upwd");
        if(upwd.equals("ratan")){
            RequestDispatcher rd=request.getRequestDispatcher("success");
            rd.forward(request, response);
        }
        else{
            writer.print("oops ! Sorry username or password error! try once again !");
            RequestDispatcher rd=request.getRequestDispatcher("login.html");
            rd.include(request, response);
        }
    }
}
```

Servlet2.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;
public class Servlet2 extends HttpServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String n=request.getParameter("uname");
        out.println("Welcome to ratansoft world : "+n);
        out.println("ratan sir welcomes u : "+n);
    }
}
```

Web.xml:-

```
<web-app>
<servlet>
    <servlet-name>sss</servlet-name>
    <servlet-class>Servlet1</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>sss</servlet-name>
    <url-pattern>/login</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>Welcome</servlet-name>
    <servlet-class>Servlet2</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Welcome</servlet-name>
    <url-pattern>/success</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>login.html</welcome-file>
</welcome-file-list>
</web-app>
```

SecondServlet.java

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class SecondServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();

        ServletConfig config = this.getServletConfig();

        writer.println(config.getInitParameter("ccc"));
        writer.println(config.getInitParameter("ddd"));

        ServletContext context = config.getServletContext();
        writer.println(context.getInitParameter("username"));
        writer.println(context.getInitParameter("password"));
    }
}
```

Web.xml:-

```
<web-app>
  <display-name>App1</display-name>
  <context-param>
    <param-name>username</param-name>
    <param-value>system</param-value>
  </context-param>

  <context-param>
    <param-name>password</param-name>
    <param-value>manager</param-value>
  </context-param>
</web-app>

<servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class>com.sravya.FirstServlet</servlet-class>
  <init-param>
    <param-name>aaa</param-name>
    <param-value>apple</param-value>
  </init-param>
  <init-param>
    <param-name>bbb</param-name>
    <param-value>ratan</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>SecondServlet</servlet-name>
```

```
<servlet-class>com.sravva.SecondServlet</servlet-class>
  <init-param>
    <param-name>ccc</param-name>
    <param-value>orange</param-value>
  </init-param>
  <init-param>
    <param-name>ddd</param-name>
    <param-value>anu</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>SecondServlet</servlet-name>
  <url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>
</web-app>
```

Application:-**Login.html:-**

```
<html>
<head>
<body>
<form action="/LoginServlet">
user name: <input type="text" name="uname"/><br/>
password : <input type="password" name="upwd"/><br/>
<input type="submit" value="login"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
  <display-name>App2</display-name>
  <welcome-file-list>
    <welcome-file>login.html</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>LoginServlet</servlet-name>
    <servlet-class>com.sravva.LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>LoginServlet</servlet-name>
    <url-pattern>/LoginServlet</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>SuccessServlet</servlet-name>
    <servlet-class>com.sravva.SuccessServlet</servlet-class>
  </servlet>
```

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

LoginServlet.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        String uname = request.getParameter("uname");
        String upwd = request.getParameter("upwd");
        if(uname.equals("sravva")&&upwd.equals("infotech"))
        {
            //RequestDispatcher dispatcher = request.getRequestDispatcher("SuccessServlet");
            //dispatcher.forward(request, response);
            request.getRequestDispatcher("SuccessServlet").forward(request, response);//project level code
        }
        else
        {
            writer.println("!OOPS login failed can u please try again");
            //RequestDispatcher dispatcher = request.getRequestDispatcher("login.html");
            //dispatcher.include(request, response);
            request.getRequestDispatcher("login.html").include(request, response);//project level code
        }
    }
}
```

SuccessServlet.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SuccessServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
```



```
        PrintWriter writer = response.getWriter();
        writer.println("login success this is success servlet");
        writer.println("<br>");
        writer.println("we are using forward mach");
    }
}
```

3. Applet-Servlet Communication:

In general in web application, we will use a browser as client, we will send request from client browser to a servlet available at server, by executing the respective servlet some response will be send back to the client browser.

Similarly in case of Applet-Servlet Communication, we will use applet as client, from the applet only we will send request to the respective servlet available at server machine, where by executing the respective servlet the required response will be generated and send back to the applet.

In above situation, the communication which we provided between applet and servlet is called as Applet-Servlet Communication.

If we want to achieve Applet-Servlet Communication in web applications we have to use the following steps.

Step 1: Prepare URL object with the respective url.

URL u=new

URL("http://localhost:8080/loginapp/login?uname=abc&upwd=abc123");

Step 2: Establish connection between applet and server by using URLConnection object.

URLConnection uc=u.openConnection();

Step 3: Send request from applet to servlet.

uc.setDoInput(true);

Note: If we do the above step a request will be send to servlet from applet where container will execute the respective servlet, generate the response and send that response to applet client. But, the response is available on URLConnection object.

Step 4: Get InputStream from URLConnection.

InputSream is=uc.getInputStream();

Step 5: Read the response from InputStream.

BufferedReader br=new BufferedReader(new InputStreamReader(is));

String res=br.readLine();

LoginApplet.java:-

import java.applet.Applet;

import java.awt.Button;

import java.awt.Color;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.Label;

import java.awt.TextField;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;

```
import java.net.URL;
import java.net.URLConnection;

public class LoginApplet extends Applet implements ActionListener {
    Label l1,l2;
    TextField tf1,tf2;
    Button b;
    String res="";
    public void init(){
        this.setBackground(Color.pink);
        this.setLayout(new FlowLayout());
        l1=new Label("User Name");
        l2=new Label("Password");
        tf1=new TextField(20);
        tf2=new TextField(20);
        tf2.setEchoChar('*');
        b=new Button("Login");
        b.addActionListener(this);
        this.add(l1);
        this.add(tf1);
        this.add(l2);
        this.add(tf2);
        this.add(b);
    }
    public void actionPerformed(ActionEvent ae) {
        try{
            URL u=new
            URL("http://localhost:2011/loginapp1/login?uname="+tf1.getText()+"&upwd="+tf2.getText());
            URLConnection uc=u.openConnection();
            uc.setDoInput(true);
            InputStream is=uc.getInputStream();
            BufferedReader br=new BufferedReader(new InputStreamReader(is));
            res=br.readLine();
            repaint();
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
    public void paint(Graphics g){
        Font f=new Font("arial", Font.BOLD, 30);
        g.setFont(f);
        g.drawString("Status :"+res, 50, 250);
    }
}

loginApplet.html:-
<applet code="LoginApplet" width="500" height="500"></applet>
```

Q: What are the differences between ServletConfig and ServletContext?

1. ServletConfig is an object, it will manage all the configuration details of a particular servlet, where the configuration details include logical name of the servlet, initialization parameters and so on.

ServletContext is an object, it will manage all the context details of a particular web application, where the context details include logical name of web application and context parameters and so on.

2. ServletConfig is an object, it will provide the complete view of a particular servlet.

ServletContext is an object, it will provide the complete view of particular web application.

3. ServletConfig object will be prepared by the container immediately after servlet instantiation and just before calling init(_) method in servlet initialization.

ServletContext object will be prepared by the container the moment when we start the server i.e. the time when we deploy the web application.

4. ServletConfig object will be destroyed by the container just before servlet deinstaniation.

ServletContext object will be destroyed by the container when we shutdown the server i.e. the time when we undeploy the web application.

5. Due to the above reasons, the life of ServletConfig object is almost all the life of the respective servlet object.

The life of ServletContext object is almost all the life of the respective web application.

6. If we declare any data in ServletConfig object then that data will be shared upto respective servlet.

If we declare any data in ServletContext object then that data will be shared to all the no. of resources which are available in the present web application.

7. Due to the above reason, ServletConfig object will provide less sharability where as ServletContext object will provide more sharability.

8. In web applications, container will prepare ServletConfig object when it receives the request from client only except in load-on-startup case.

In web applications, container will prepare ServletContext object irrespective of getting request from client.

9. In web applications, ServletConfig object will allow only parameters data but ServletContext object will allow both parameters and attributes data.

10. Due to the above reason, ServletConfig object will allow only Static Inclusion of data where as ServletContext object will allow both Static Inclusion and Dynamic Inclusion of data.

To get the ServletContext object we have to use the following method from ServletConfig.

```
public ServletContext getServletContext();
```

Ex: `ServletContext context=config.getServletContext();`

Note: In servlet3.0 version, it is possible to get `ServletContext` object even from `ServletRequest`.

Ex: `ServletContext context=req.getServletContext();`

If we want to get the logical name of the web application from `ServletContext` object first of all we have to declare it in `web.xml` file.

To declare a logical name in `web.xml` file we have to use the following xml tag.

```
<web-app>
-----
<display-name>logical_name</display-name>
-----
</web-app>
```

To get the logical name of web application from `ServletContext` object we will use the following method.

`Public String getServletContextName()`

Ex: `String lname=context.getServletContextName();`

If we want to provide context parameters on `ServletContext` object first we have to declare them in `web.xml` file.

FirstServlet.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();

        ServletConfig config = this.getServletConfig();

        writer.println(config.getInitParameter("aaa"));
        writer.println(config.getInitParameter("bbb"));

        ServletContext context = config.getServletContext();
        writer.println(context.getInitParameter("username"));
        writer.println(context.getInitParameter("password"));
```

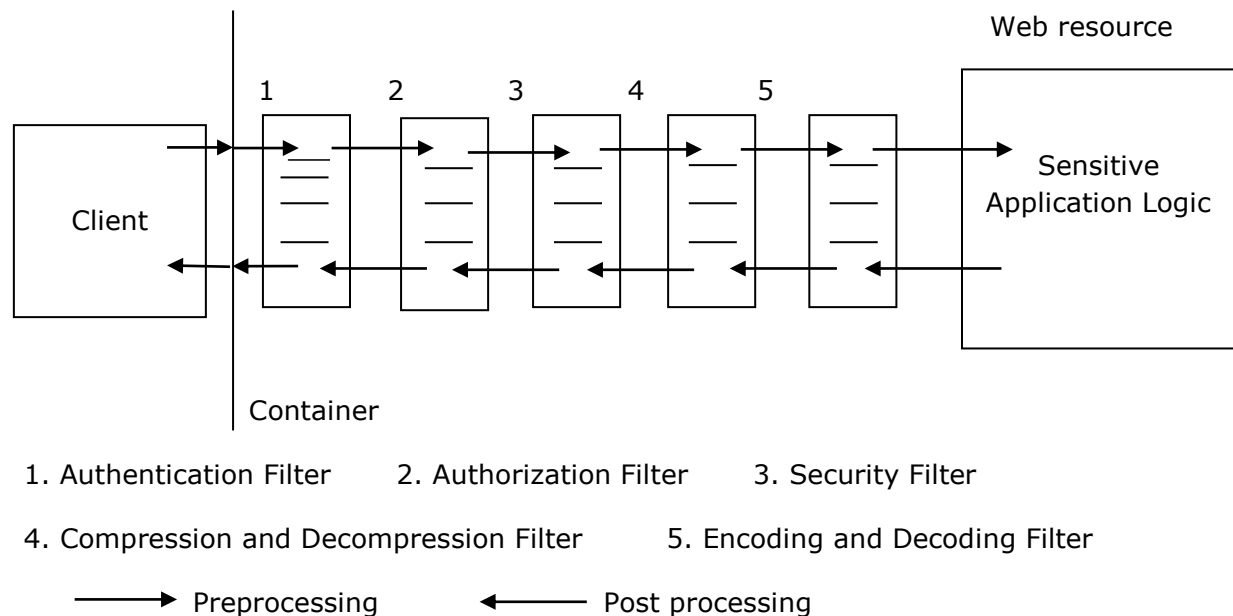
```
}  
}
```

Filters:

In general in web application development, we will provide the complete application logic in the form of the web resources like servlets, jsp's and so on.

As part of the web application development some web resources may require the services like Authentication, Authorization, Security, Data compression and decompression and so on as preprocessing and post processing.

In the above context, to implement all the above preprocessing and post processing services Servlet API has provided a separate component called **Filter**.



From the above representation when we send a request from client to server for a particular web resource then container will pick up that request, container will check whether any Filter is associated with the respective web resource, if container identify any Filter or Filters then container will execute that Filters first.

While executing a Filter if the present request is satisfied all the Filter constraints then only container will bypass that request to next Filter or next web resource.

If the present request is not satisfied the Filter constraints then container will generate the respective response to client.

Filter is a server side component, it will be executed by the container automatically when it receives request from client for a particular web resource.

If we want to use Filters in our web applications, we have to use the following steps.

Step 1: Prepare Filter class.

Step 2: Configure Filter class in web.xml file.

Step 1: Prepare Filter class:

Filter is an object available at server machine, it must implement Filter interface either directly or indirectly.

```
public interface Filter {  
    public void init(FilterConfig config)throws ServletException;  
    public void doFilter(ServletRequest req, ServletResponse res, FilterChain fc)throws SE, IOE  
    public void destroy(); }
```

```
public class MyFilter implements Filter { ----- }
```

Where init(_) method can be used to perform Filter Initialization.

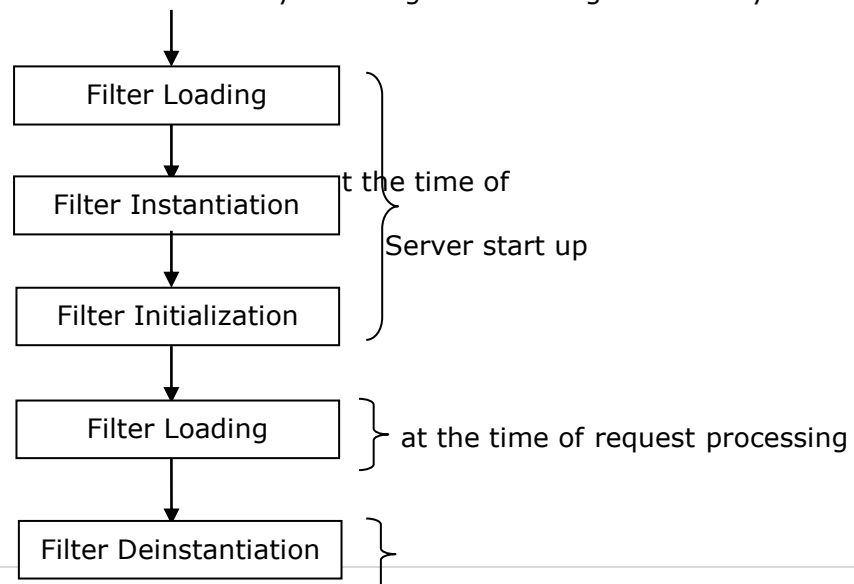
Where doFilter(_____) method is same as service(_____) method in servlets it is able to accommodate actual Filter logic.

Where destroy() method can be used to perform Filter Deinstantiation.

While executing a particular Filter in web applications, if we satisfy all the Filter constraints then we need to bypass the request from present Filter to the next Filter web resource, for this we need to use the following method from FilterChain.

```
public void doFilter(ServletRequest req, ServletResponse res)throws SE, IOE
```

While executing a particular web application, when container identify a particular Filter to execute then container will execute that Filter by following the following Filter life cycle.



at the time of server shutdown



In web applications, by default all the Filters are auto-loaded, auto-instantiated, auto-initialized at the time of server start up. So that Filters should not require load-on-startup configuration in web.xml file.

Step 2: Filter class Configuration:

To configure a Filter in web.xml file we have to use the following xml tags.

```
<web-app>
  <filter>
    <filter-name>logical_name</filter-name>
    <filter-class>Fully Qualified name of Filter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>logical_name</filter-name>
    <url-pattern>pattern_name</url-pattern>
    or
    <servlet-name>logical name of servlet</servlet-name>
  </filter-mapping>
  -----
</web-app>
```

If we want to provide mapping between a Filter and Servlet then we have to provide the same url-pattern for both Filter and Servlet or provide the respective servlet logical name along with <servlet-name> tag in place of <url-pattern> tag under <filter-mapping>.

In web applications, it is possible to use a single Filter for multiple number of web resources.

To achieve this we have to use " /* " (Directory match) as url-pattern to the respective Filter.

In web applications, it is possible to provide multiple number of Filters for a single web resource, in this case container will execute all the Filters as per the order in which we provided <filter-mapping> tags in web.xml file.

Form.html:-

```
<html>
<body>
<form action="/SuccessServlet">
User name : <input type="text" name="uname"/><br>
user age  : <input type="text" name="uage"/><br>
user address : <input type="text" name="uaddr"/><br>
<input type="submit" value="registration"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<display-name>App3</display-name>
<welcome-file-list>
  <welcome-file>form.html</welcome-file>
</welcome-file-list>

<filter>
  <filter-name>Filter1</filter-name>
  <filter-class>com.sravya.Filter1</filter-class>
</filter>
<filter-mapping>
  <filter-name>Filter1</filter-name>
  <url-pattern>/SuccessServlet</url-pattern>
</filter-mapping>

<filter>
  <filter-name>Filter2</filter-name>
  <filter-class>com.sravya.Filter2</filter-class>
</filter>
<filter-mapping>
  <filter-name>Filter2</filter-name>
  <url-pattern>/SuccessServlet</url-pattern>
</filter-mapping>

<servlet>
  <servlet-name>SuccessServlet</servlet-name>
  <servlet-class>com.sravya.SuccessServlet</servlet-class>
</servlet>
<servlet-mapping>
```



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

Filter1.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class Filter1 implements Filter {
    public void destroy() {
        // TODO Auto-generated method stub
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        int age = Integer.parseInt(request.getParameter("uage"));

        if(age>20)
        {
            // pass the request along the filter chain
            chain.doFilter(request, response);
        }
        else
        {
            writer.println("u r not eligible for mrg u age is below 20 years");
            request.getRequestDispatcher("form.html").include(request, response);
        }
    }
    public void init(FilterConfig fConfig) throws ServletException {
    }
}
}
```

Filter2.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
```

```
import javax.servlet.ServletException;
import javax.servlet.ServletResponse;
public class Filter2 implements Filter {
    public void destroy() {
        // TODO Auto-generated method stub
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        String uaddr = request.getParameter("uaddr");
        if(uaddr.equals("hyderabad"))
        {
            chain.doFilter(request, response);
        }
        else
        {
            writer.println("this application only for hyd person");
            request.getRequestDispatcher("form.html").include(request, response);
        }
    }
    public void init(FilterConfig fConfig) throws ServletException {
        // TODO Auto-generated method stub
    }
}
```

SuccessServlet.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SuccessServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        String uname = request.getParameter("uname");
        String uage = request.getParameter("uage");
        String uaddr = request.getParameter("uaddr");

        writer.println("***** U r registration success*****");
        writer.println("<br>");
        writer.println("user name="+uname);
        writer.println("<br>");
        writer.println("<br>");
        writer.println("user age="+uage);
        writer.println("<br>");
        writer.println("user naddress="+uaddr);
    }
}
```

```
        writer.println("<br>");  
        writer.println("user registration id=13456");  
        writer.println("<br>");  
        writer.println("we will find one girl for u soon.....keep smiling");  
    }  
}
```

Session Tracking Mechanisms:

As part of the web application development it is essential to manage the clients previous request data at the time of processing later request.

To achieve the above requirement if we use request object then container will create request object when it receives request from client and container will destroy request object when it dispatch response to client.

Due to this reason request object is not sufficient to manage clients previous request data at the time of processing later request.

To achieve the above requirement we able to use ServletContext object, but ServletContext object will share its data to all the components which are used in the present applications and to all the users of the present web application.

Due to this reason ServletContext object is unable to provide clear cut separation between multiple users.

In web applications, to manage clients previous request data at the time of processing later request and to provide separation between multiple users we need a set of mechanisms explicitly at server side called as **Session Tracking Mechanisms**.

Session:

Session is a time duration, it will start from the starting point of client conversation with server and will terminate at the ending point of client conversation with the server.

The data which we transferred from client to server through multiple number of requests during a particular session then that data is called **State of the Session**.

In general in web applications, container will prepare a request object similarly to represent a particular user we have to prepare a separate session.

If we allow multiple number of users on a single web application then we have to prepare multiple number of session objects.

In this context, to keep track of all the session objects at server machine we need a set of explicit mechanisms called as **Session Tracking Mechanisms**.

In web applications, there are 4 types of Session Tracking Mechanisms.

- 1) **HttpSession Session Tracking Mechanism**
- 2) **Cookies Session Tracking Mechanism**
- 3) **URL-Rewriting Session Tracking Mechanism**
- 4) **Hidden Form Fields Session Tracking Mechanism**

From the above Session Tracking Mechanisms Servlet API has provided the first 3 Session Tracking Mechanisms as official mechanisms, Hidden Form Fields Session Tracking Mechanism is purely developers creation.

1. HttpSession Session Tracking Mechanism:

In **HttpSession Session Tracking Mechanism**, we will create a separate HttpSession object for each and every user, at each and every request we will pick up the request parameters from request object and we will store them in the respective HttpSession object for the sake of future reusability.

After keeping the request parameters data in HttpSession object we have to generate the next form at client browser by forwarding the request to particular static page or by generating dynamic form.

In HttpSession Session Tracking Mechanism, to create HttpSession object we will use either of the following methods.

```
req.getSession();  
req.getSession(false);
```

Q: What is the difference between getSession() method and getSession(false) method?

Ans: Both the methods can be used to return HttpSession object.

To get HttpSession object if we getSession() method then container will check whether any HttpSession object existed for the respective user or not, if any HttpSession object is existed then container will return the existed HttpSession object reference.

If no HttpSession object is existed for the respective user then container will create a new HttpSession object and return its reference.

```
public HttpSession getSession()
```

Ex: HttpSession hs=req.getSession();

To get HttpSession object if we getSession(false) method then container will check whether any HttpSession object existed w.r.t. user or not, if any HttpSession object is existed then container will return that HttpSession object reference.

If no HttpSession object is existed then container will return null value without creating new HttpSession object.

```
public HttpSession getSession(boolean b)
```

Ex: HttpSession hs=req.getSession(false);

Note: getSession(true) method functionality is almost all same as getSession() method.

Q: If we allow multiple number of users to access present web application then automatically container will create multiple number of HttpSession objects. In this case how container will identify user specific HttpSession object in order to put user specific attributes and to get attributes?

Ans: In HttpSession Session Tracking Mechanism, when we create HttpSession object automatically container will create an unique identification number in the form of hexadecimal number called as **Session Id**. Container will prepare session id in the form of Cookie with the name **JSESSIONID**.

In general the basic nature of Cookie is to transfer from server to client automatically along with response and it will be transferred from client to server automatically along with request.

Due to the above nature of Cookies session id Cookie will be transferred from server to client and from client to server automatically.

In the above context, if we use getSession() method or getSession(false) method first container will pick up session id value from request and it will identify the user specific HttpSession object on the basis of session id value.

To destroy HttpSession object explicitly we will use the following method from HttpSession.

```
public void invalidate()
```

If we want to destroy HttpSession object after a particular ideal time duration then we have to use the following method.

```
public void setMaxInactiveInterval(int time)
```

In web applications, HttpSession object will allow only attributes data, it will not allow parameters data.

To set an attribute on to the HttpSession object we have to use the following method.

```
public void setAttribute(String name, Object value)
```

To get a particular attribute value from HttpSession object we have to use the following method.

```
public Object getAttribute(String name)
```

To get all attribute names from HttpSession object we have to use the following method.

```
public Enumeration getAttributeNames()
```

To remove an attribute from HttpSession object we have to use the following method.

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

Drawbacks:

1. In HttpSession Session Tracking Mechanism, we will create a separate HttpSession object for each and every user, where if we increase number of users then automatically number of HttpSession object will be created at server machine, it will reduce the overall performance of the web application.

2. In case of HttpSession Session Tracking Mechanism, we are able to identify user specific HttpSession object among multiple number of HttpSession objects by carrying Session Id value from client to server and from server to client.

In the above context, if the client browser disable Cookies then HttpSession Session Tracking Mechanism will not execute its functionality.

Form1.html:

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="/FirstServlet">
<pre>
Name          <input type="text" name="uname"/>
Age           <input type="text" name="uage"/>
<input type="submit" value="Next"/>
</pre>
</form></font></b></body></html>
```

Form2.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br><hr>
<body bgcolor="lightblue"><b>
```

```
<font size="7">
<form method="post" action="/SecondServlet">
<pre>                Qualification        <input type="text" name="uqual"/>
                Designation            <input type="text" name="udes"/>
                <input type="submit" value="Next"/>
</pre>
</form></font></b></body></html>
```

Form3.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="/DisplayServlet">
<pre>                Email            <input type="text" name="email"/>
                Mobile            <input type="text" name="mobile"/>
                <input type="submit" value="display"/>
</pre>
</form></font></b></body></html>
```

Web.xml:-

```
<web-app >
<display-name>HttpSession</display-name>
<welcome-file-list>
  <welcome-file>form1.html</welcome-file>
</welcome-file-list>

<servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class>com.sravva.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>SecondServlet</servlet-name>
  <servlet-class>com.sravva.SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>SecondServlet</servlet-name>
  <url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>DisplayServlet</servlet-name>
```

```
<servlet-class>com.sravva.DisplayServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>DisplayServlet</servlet-name>
  <url-pattern>/DisplayServlet</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java:-

```
package com.sravva;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class FirstServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String uname = request.getParameter("uname");
        String uage = request.getParameter("uage");

        HttpSession session = request.getSession();//sid-cookie

        session.setAttribute("uname",uname);
        session.setAttribute("uage",uage);

        RequestDispatcher dispatcher = request.getRequestDispatcher("form2.html");
        dispatcher.forward(request, response);
    }
}
```

SecondServlet.java:-

```
package com.sravva;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class SecondServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String uqual = request.getParameter("uqual");
        String udes = request.getParameter("udes");
        HttpSession session = request.getSession();//sid already available
```

```
session.setAttribute("uqual", uqual);  
session.setAttribute("udes", udes);
```

```
RequestDispatcher dispatcher = request.getRequestDispatcher("form3.html");  
dispatcher.forward(request, response);
```

```
}
```

```
}
```

DisplayServlet.java:-

```
package com.sravya;  
import java.io.IOException;  
import java.io.PrintWriter;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import javax.servlet.http.HttpSession;  
public class DisplayServlet extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter writer = response.getWriter();  
  
        String mobile = request.getParameter("mobile");  
        String email = request.getParameter("email");  
  
        HttpSession session = request.getSession();//located  
  
        writer.println("*****Complete details*****");  
        writer.println("<br>");  
        writer.println("user name: "+session.getAttribute("uname"));  
        writer.println("<br>");  
        writer.println("user age: "+session.getAttribute("uage"));  
        writer.println("<br>");  
        writer.println("user qualification: "+session.getAttribute("uqual"));  
        writer.println("<br>");  
        writer.println("user designation: "+session.getAttribute("udes"));  
        writer.println("<br>");  
        writer.println("user mobile: "+mobile);  
        writer.println("<br>");  
        writer.println("user email: "+email);  
        writer.println("<br>");  
    }  
}
```


URL-Rewriting Session Tracking Mechanism:

In case of HttpSession Session Tracking Mechanism, when we create HttpSession object automatically Session Id will be created in the form of the Cookie, where Session Id Cookie will be transferred from server to client and from client to server along with response and request automatically.

In HttpSession Session Tracking Mechanism, we are able to identify user specific HttpSession object on the basis of Session Id only.

In this context, if we disable Cookies at client browser then HttpSession Session Tracking Mechanism will not execute its functionality.

In case of Cookies Session Tracking Mechanism, the complete client conversation will be stored at the respective client machine only in the form of Cookies, here the Cookies data will be opened to every user of that machine. So that Cookies Session Tracking Mechanism will not provide security for the application data.

To overcome the above problem, we have to use URL-Rewriting Session Tracking Mechanism.

In case of URL-Rewriting Session Tracking Mechanism, we will not maintain the clients conversation at the respective client machine, we will maintain the clients conversation in the form of HttpSession object at server machine. So that URL-Rewriting Session Tracking Mechanism is able to provide very good security for the application data.

URL-Rewriting Session Tracking Mechanism is almost all same as HttpSession Session Tracking Mechanism, in URL-Rewriting Session Tracking Mechanism we will not depending on a Cookie to maintain Session Id value, we will manage Session Id value as an appender to URL in the next generated form.

In this context, if we send a request from the next generated form automatically the appended Session Id value will be transferred to server along with the request.

In this case, even though if we disable Cookies at client browser, but still we are able to get Session Id value at server machine and we are able to manage clients previous request data at the time of processing the later request.

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session Id value in the next generated form. So that this mechanism is called as **URL-Rewriting Session Tracking Mechanism**.

In URL-Rewriting Session Tracking Mechanism, it is mandatory to append Session Id value to the URL by getting Session Id value explicitly.

To perform this work HttpServletResponse has provided a separate method like,

public String encodeURL(String url)

Ex: out.println("<form method='get'

```
action="" + res.encodeURL("./second") + ">");
```

Drawback:

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session Id value in the generated form, for this we must execute encodeURL() method. So that URL-Rewriting Session Tracking Mechanism should require dynamically generated forms, it will not execute its functionality with static forms.

Form.html:-

```
<html>
<body>
<form action="./FirstServlet">
User name : <input type="text" name="uname"/><br>
user age  : <input type="text" name="uage"/><br>
<input type="submit" value="next"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<display-name>url</display-name>
<welcome-file-list>
  <welcome-file>form.html</welcome-file>
</welcome-file-list>

<servlet>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class>com.sravva.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>SecondServlet</servlet-name>
  <servlet-class>com.sravva.SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>SecondServlet</servlet-name>
  <url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>

<servlet>
  <servlet-name>DisplayServlet</servlet-name>
  <servlet-class>com.sravva.DisplayServlet</servlet-class>
</servlet>
<servlet-mapping>
```

```
<servlet-name>DisplayServlet</servlet-name>
<url-pattern>/DisplayServlet</url-pattern>
</servlet-mapping>

</web-app>
```

FirstServlet.java:-

```
package com.sravva;
```

```
import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

```
public class FirstServlet extends HttpServlet {
    @SuppressWarnings("deprecation")
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String uname = request.getParameter("uname");
        String uage = request.getParameter("uage");

        HttpSession session = request.getSession();

        session.setAttribute("uname", uname);
        session.setAttribute("uage", uage);

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

        writer.println("<html>");
        writer.println("<body>");
        writer.println("<form method='get' action='"+response.encodeUrl("./SecondServlet")+">");
        writer.println("<br>");
        writer.println("user qualificatins :<input type='text' name='uqual'/>");
        writer.println("<br>");
        writer.println("user designation : <input type='text' name='udes'/>");
        writer.println("<br>");
        writer.println("<input type='submit' value='next'/>");
        writer.println("</form>");
        writer.println("</body>");
        writer.println("</html>");
    }
}
```

SecondServlet.java:-

```
package com.sravva;
```

```
import java.io.IOException;  
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import javax.servlet.http.HttpSession;
```

```
public class SecondServlet extends HttpServlet {
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {
```

```
        String uqual = request.getParameter("uqual");  
        String udes = request.getParameter("udes");
```

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

```
        session.setAttribute("uqual", uqual);  
        session.setAttribute("udes", udes);
```

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

```
        writer.println("<html>");  
        writer.println("<body>");  
        writer.println("<form method='get'  
action="+response.encodeUrl("./DisplayServlet")+ ">");  
        writer.println("<br>");  
        writer.println("user email :<input type='text' name='email'/>");  
        writer.println("<br>");  
        writer.println("user mobile : <input type='text' name='mobile'/>");  
        writer.println("<br>");  
        writer.println("<input type='submit' value='display'/>");  
        writer.println("</form>");  
        writer.println("</body>");  
        writer.println("</html>");
```

```
}  
}
```

DisplayServlet.java:-

```
package com.sravva;
```

```
import java.io.IOException;  
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import javax.servlet.http.HttpSession;
```

```
public class DisplayServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  
ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter writer = response.getWriter();  
  
        HttpSession session = request.getSession();//locate  
  
        writer.println("user name =" + session.getAttribute("uname"));  
        writer.println("<br>");  
        writer.println("user age =" + session.getAttribute("uage"));  
        writer.println("<br>");  
        writer.println("user qualifications =" + session.getAttribute("uqual"));  
        writer.println("<br>");  
        writer.println("user designations =" + session.getAttribute("udes"));  
        writer.println("<br>");  
        writer.println("user email id =" + request.getParameter("email"));  
        writer.println("<br>");  
        writer.println("user mobile =" + request.getParameter("mobile"));  
    }  
}
```

Cookies Session tracking machanism:-

Cookie is a small object, it can be used to represent a single name value pair and which will be maintained permanently at client machine.

In HttpSession Session Tracking Mechanism, all the clients conversations will be maintained at server machine in the form of HttpSession objects.

In HttpSession Session Tracking Mechanism, if we increase number of users then automatically number of HttpSession objects will be created at server. So that HttpSession Session Tracking Mechanism will increase burden to server machine.

To overcome the above problem we have to use an alternative mechanism, where we have to manage all the clients conversations at the respective client machines only.

To achieve the above requirement we have to use **Cookies Session Tracking Mechanism**.

In Cookies Session Tracking Mechanism, at each and every client we will pick up all the request parameters, prepare a separate Cookie for each and every request parameter, add all the Cookies to response object.

In the above context, when container dispatch response to client automatically all the added Cookies will be transferred to client and maintain at client machine permanently.

In the above context, when we send further request from the same client automatically all the Cookies will be transferred to server along with request.

By repeating the above process at each and every request we are able to manage clients previous data at the time of processing later request.

Drawbacks:

If we disable the Cookies at client browser then Cookies Session Tracking Mechanism will not execute its functionality.

In case of Cookies Session Tracking Mechanism, all the clients data will be maintain at the respective client machine only, which is open to every user of that machine. So that Cookies Session Tracking Mechanism will not provide security for the application data.

To create Cookie object with a particular name-value pair we have to use the following Cookie class constructor.

```
public Cookie(String name, String value)
```

To add a Cookie to the response object we have to use the following method from HttpServletResponse.

```
public void addCookie(Cookie c)
```

To get all the Cookies from response object we need to use the following method.

```
public Cookie[] getCookies()
```

To get name and value of a Cookie we have to use the following methods,

```
public String getName()
```

```
public String getValue()
```

In web applications, it is possible to provide comments to the Cookies. So that to set the comment to Cookie and get the comment from Cookie we need to use the following methods.

```
public void setComment(String comment)
```

```
public String getComment()
```

In web applications, it is possible to provide version numbers to the Cookies. So that to set a version number to Cookie and get a version number from Cookie we need to use the following methods.

```
public void setVersion(int version_no)
```

```
public int getVersion()
```

In web applications, it is possible to specify life time to the Cookies. So that to set max age to Cookie and get max age from Cookie we need to use the following methods.

```
public void setMaxAge(int age)
```

```
public int getMaxAge()
```

In web applications, it is possible to provide domain names to the Cookies. So that to set domain name to Cookie and get domain name from Cookie we need to use the following methods.

```
public void setDomain(String domain)
```

```
public String getDomain()
```

In web applications, it is possible to provide a particular path to the Cookies to store. So that to set a path to Cookie and get a path from Cookie we need to use the following methods.

```
public void setPath(String path)
```

```
public String getPath()
```

Form1.html:

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="/FirstServlet">
<pre>                Name                <input type="text" name="uname"/>
                        Age                <input type="text" name="uage"/>
                        <input type="submit" value="Next"/>

</pre>
</form></font></b></body></html>
```

Form2.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br><hr>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="/SecondServlet">
<pre>                Qualification        <input type="text" name="uqual"/>
                        Designation      <input type="text" name="udes"/>
                        <input type="submit" value="Next"/>

</pre>
</form></font></b></body></html>
```

Form3.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="/DisplayServlet">
```



```
<pre>                Email        <input type="text" name="email"/>
                Mobile        <input type="text" name="mobile"/>
                <input type="submit" value="display"/>
</pre>
</form></font></b></body></html>
```

Web.xml:-

```
<web-app >
  <display-name>Cookies</display-name>
  <welcome-file-list>
    <welcome-file>form1.html</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>com.sravva.FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/FirstServlet</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>SecondServlet</servlet-name>
    <servlet-class>com.sravva.SecondServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SecondServlet</servlet-name>
    <url-pattern>/SecondServlet</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>DisplayServlet</servlet-name>
    <servlet-class>com.sravva.DisplayServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>DisplayServlet</servlet-name>
    <url-pattern>/DisplayServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

FirstServlet.java:-

```
package com.sravva;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        String uname = request.getParameter("uname");
        String uage = request.getParameter("uage");

        Cookie cookie1 = new Cookie("uname", uname);
        Cookie cookie2 = new Cookie("uage", uage);

        response.addCookie(cookie1);
        response.addCookie(cookie2);

        request.getRequestDispatcher("form2.html").forward(request, response);
    }
}
```

SecondServlet:-

```
package com.sravva;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SecondServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String uqual = request.getParameter("uqual");
        String udes = request.getParameter("udes");
    }
}
```

```
        Cookie cookie3 = new Cookie("uqual",uqual);
        Cookie cookie4 = new Cookie("udes",udes);

        response.addCookie(cookie3);
        response.addCookie(cookie4);
        request.getRequestDispatcher("form3.html").forward(request, response);
    }
}
```

DisplayServlet.java:-

```
package com.sravva;
```

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class DisplayServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();

        String mobile = request.getParameter("mobile");
        String email = request.getParameter("email");

        Cookie[] c = request.getCookies();
        writer.println("user name="+ c[0].getValue());
        writer.println("<br>");
        writer.println("user name="+ c[1].getValue());
        writer.println("<br>");
        writer.println("user name="+ c[2].getValue());
        writer.println("<br>");
        writer.println("user name="+ c[3].getValue());
        writer.println("<br>");
        writer.println("user name="+ mobile);
        writer.println("<br>");
        writer.println("user name="+ email);
        writer.println("<br>");
    }
}
```

Hidden Form Field Session Tracking Mechanism:

Hidden Form Field Session Tracking Mechanism is not official Session Tracking Mechanism from Servlet API, it was purely developers creation.

In Hidden Form Field Session Tracking Mechanism, at each and every request we will pick up all the request parameters, generate dynamic form, in dynamic form generation we have to maintain the present request parameters data in the form of hidden fields.

In the above context, if we dispatch the response to client then we are able to get a dynamic form with visible fields and with invisible fields.

If we send a request from dynamic form then automatically all the visible fields data and invisible fields data will be send to server as request parameters.

By repeating above process at each and every request we are able to manage the clients previous request data at the time of processing the later request.

studentform.html:-

```
<html>
<head>
<body bgcolor="lightgreen">
    <form method="get" action="./first">
        <center><b><br><br>
        Student Name:<input type="text" name="sname"/><br><br>
        Student Id:<input type="text" name="sid"/><br><br>
        Student Address:<input type="text" name="saddr"/><br><br>
        <input type="submit" value="Submit">
        </b></center>
    </form>
</body>
</html>
```

web.xml:-

```
<web-app>
  <display-name>hiddenformfieldsapp</display-name>
  <welcome-file-list>
    <welcome-file>studentform.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>FirstServlet</servlet-name>
```

```
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/first</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>SecondServlet</servlet-name>
  <servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>SecondServlet</servlet-name>
  <url-pattern>/second</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>ThirdServlet</servlet-name>
  <servlet-class>ThirdServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>ThirdServlet</servlet-name>
  <url-pattern>/third</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FirstServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        PrintWriter out=response.getWriter();

        String sname=request.getParameter("sname");
        String sid=request.getParameter("sid");
        String saddr=request.getParameter("saddr");

        out.println("<html><body bgcolor='lightyellow'>");
        out.println("<center><b><br><br>");
        out.println("Welcome to Student Application");
        out.println("<br><br>");
        out.println("<form method='get' action='/hiddenformfieldsapp/second'>");
        out.println("<input type='hidden' name='sname' value='"+sname+"'>");
        out.println("<input type='hidden' name='sid' value='"+sid+"'>");
```

```
        out.println("<input type='hidden' name=saddr value='"+saddr+"'>");
        out.println("<br><br>");
        out.println("Student Age:");
        out.println("<input type='text' name='sage'>");
        out.println("<br><br>");
        out.println("<input type='submit' value='Submit'>");
        out.println("</form></b></center></body></html>");
    }
}
```

SecondServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class SecondServlet extends HttpServlet {
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
```

```
        PrintWriter out=response.getWriter();
        String sname=request.getParameter("sname");
        String sid=request.getParameter("sid");
        String saddr=request.getParameter("saddr");
        String sage=request.getParameter("sage");
        out.println("<html><body bgcolor='lightyellow'>");
        out.println("<center><b><br><br>");
        out.println("Student Details Are...");
        out.println("<br><br>");
        out.println("Student Name....."+sname+"<br><br>");
        out.println("Student Id....."+sid+"<br><br>");
        out.println("Student Address....."+saddr+"<br><br>");
        out.println("<a href='/hiddenformfieldsapp/third?sage="+sage+"'>
                                SHOW STUDENT AGE</a>");
```

```
    }
}
```

ThirdServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ThirdServlet extends HttpServlet {
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
```

```
        PrintWriter out=response.getWriter();
        String sage=request.getParameter("sage");
        out.println("<html><body bgcolor='lightpink'>");
        out.println("<center><b><br><br>");
        out.println("Student Age is....."+sage);
        out.println("</b></center></body></html>");
    }
}
```

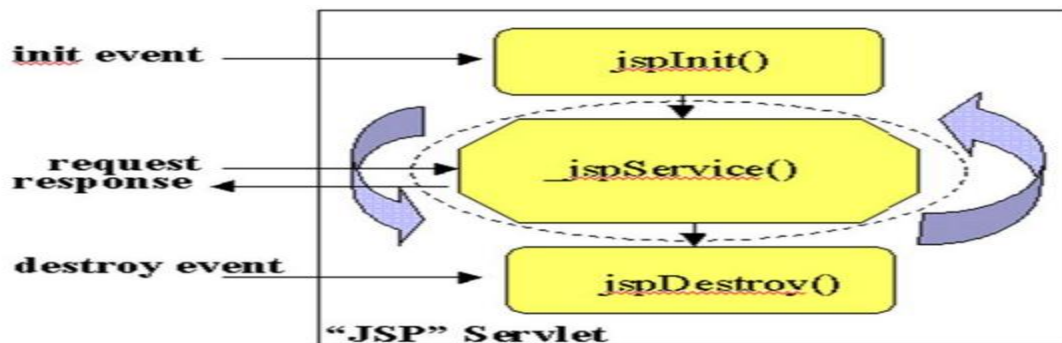
JSP (Java Server Pages)

- ✓ Jsp is extension of servlets so we can use all features of servlets in addition we can use implicit objects, custom tags, predefined tags.
- ✓ If we down modifications on servlets for every modification need to recompiled and redeployed but in jsp refresh button is enough to reflect the modifications.
- ✓ Servlets runs faster compere to jsp because the jsp are internally converted into servlets that converted servlets are executed then we will get the response.
- ✓ To write the servlets we required more java knowledge but to write the jsp code less java knowledge is sufficient.
- ✓ In MVC jsp acts as a view part & servlets acts as a controller part.
- ✓ Servlets are best for more processing logics but jsp are best for more presentation logics rather than processing logics.
- ✓ The present version of servlets is **3.1** & present version of jsp is **2.2**
- ✓ In servlets we are mixing both presentation logics and business logics but in jsp we can separate our business logics with presentation logics.
- ✓ We must place the servlets in private area of directory structure so to access the private area elements **web.xml** is mandatory but it is possible to place the jsp pages both public & private area hence to access the jsp **web.xml** is optional.
- ✓ The servlets predefined support **servlet-api.jar** & jsp predefined support **jsp-api.jar**.
- ✓ The life cycle methods of servlets
 - `Inti()`
 - `Service()`
 - `Destroy()`



The life cycle methods of jsp

`_jspInt()`
`_jspService()`
`_jspDestroy()`



Servlet is a Thread based technology, if we deploy it at server then container will create a separate thread instead of the process for every request from the client.

Due to this Thread based technology at server side server side application performance will be increased.

In case of the servlet, we are unable to separate both presentation logic and business logic.

If we perform any modifications on servlets then we must perform recompilation and reloading.

If we want to design web applications by using servlets then we must require very good knowledge on Java technology.

JSP is a server side technology provided by Sun Microsystems to design web applications in order to generate dynamic response.

The main intention to introduce Jsp technology is to reduce java code as much as possible in web applications.

Jsp technology is a server side technology, it was designed on the basis of Servlet API and Java API.

In web application development, we will utilize Jsp technology to prepare view part or presentation part.

Jsp technology is very good at the time of generating dynamic response to client with very good look and feel.

If we want to design any web application with Jsp technology then it is not required to have java knowledge.

In case of Jsp technology, we are able to separate presentation logic and business logic because to prepare presentation logic we will use html tags and to prepare business logic we will use Jsp tags separately.

If we perform any modifications on Jsp pages then it is not required to perform recompilation and reloading because Jsp pages are auto-compiled and auto-loaded.

Jsp Deployment:

In web application development, it is possible to deploy the Jsp pages at any location of the web application directory structure, but it is suggestible to deploy the Jsp pages under application folder.

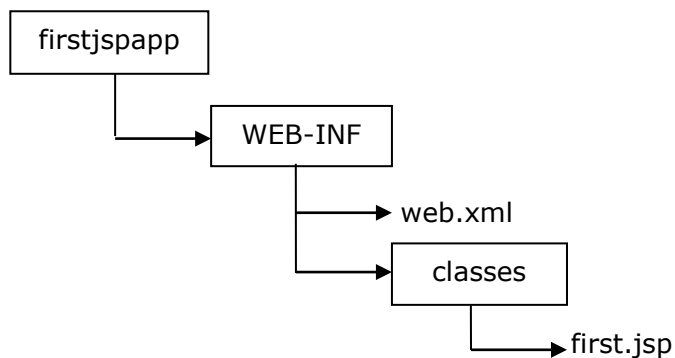
If we deploy the Jsp pages under application folder i.e. public area then we are able to access that Jsp page from client by using its name directly in the url.

If we deploy the Jsp pages under private area(WEB-INF, classes) then we must define url pattern for the Jsp page in web.xml file and we are able to access that Jsp page by specifying url pattern in client url.

To configure Jsp pages in web.xml file we have to use the following xml tags.

```
<web-app>
```

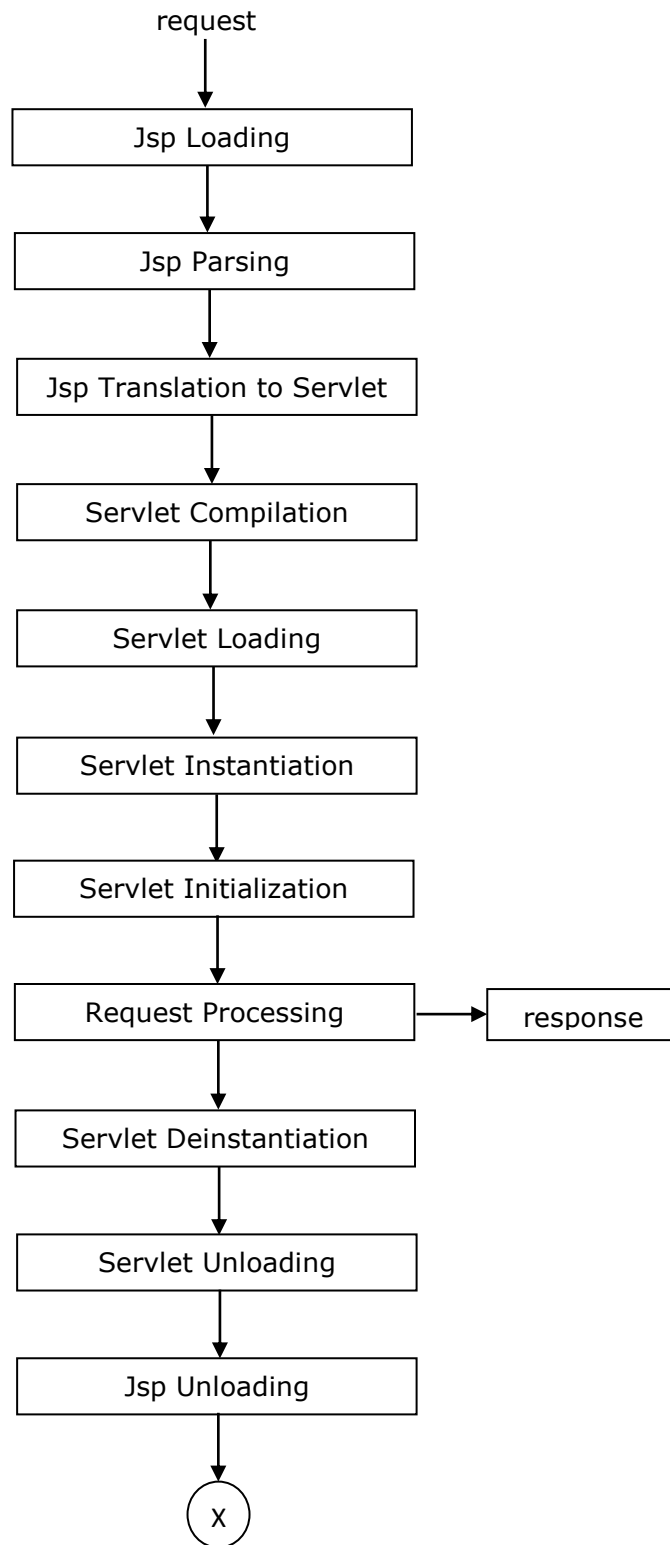
```
-----  
<servlet>  
  <servlet-name>logical_name</servlet-name>  
  <Jsp-file>context relative path of Jsp page</Jsp-file >  
</servlet>  
<servlet-mapping>  
  <servlet-name>logical_name</servlet-name>  
  <url-pattern>pattern_name</url-pattern>  
</servlet-mapping>  
-----  
</web-app>
```

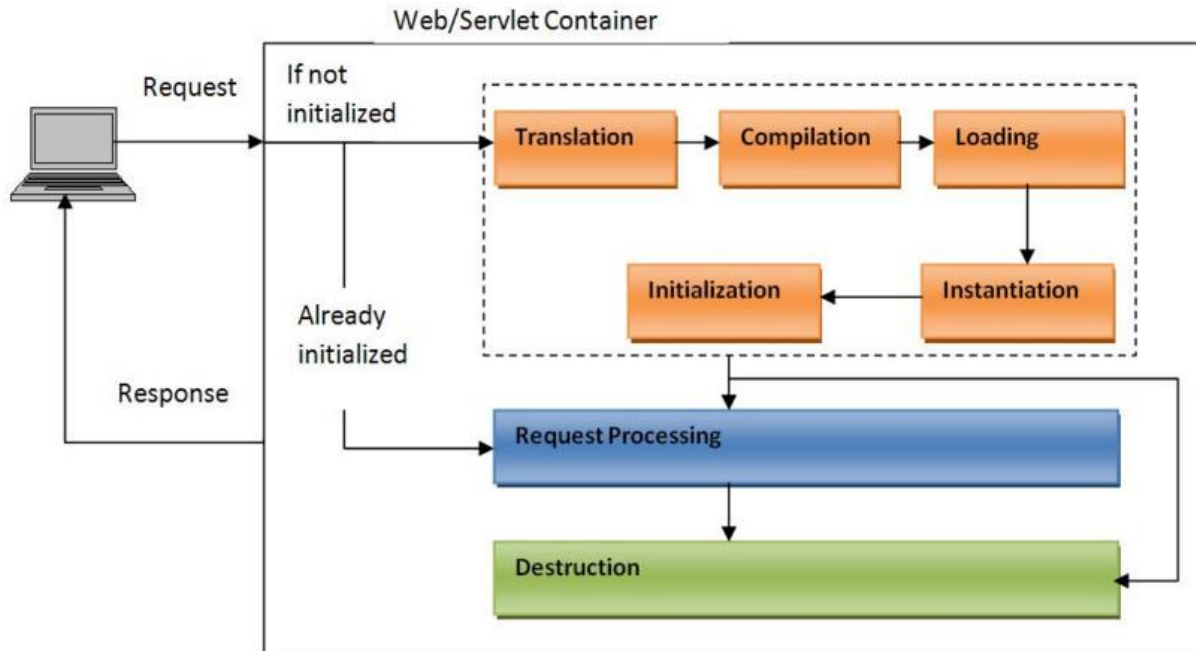
Application:**Directory Structure:****first.jsp:**

```
<html>  
  <body bgcolor="lightgreen">  
    <center><b><font size="7" color="red">  
      <br><br>  
      First Jsp Application Deployed in Classes  
    </font></b></center>  
  </body>  
</html>
```

Web.xml:

```
<web-app>  
  <servlet>  
    <servlet-name>fj</servlet-name>  
    <jsp-file>/WEB-INF/classes/first.jsp</jsp-file>  
  </servlet>  
  <servlet-mapping>  
    <servlet-name>fj</servlet-name>  
    <url-pattern>/jsp</url-pattern>  
  </servlet-mapping>  
</web-app>
```

Jsp Life Cycle:



When we send request from client to server for a particular Jsp page then container will pick up the request, identify the requested Jsp pages and perform the following life cycle actions.

1. Jsp Loading:

Here container will load Jsp file to the memory from web application directory structure.

2. Jsp Parsing:

Here container will check whether all the tags available in Jsp page are in well-formed format or not.

3. Jsp Translation to Servlet:

After the Jsp parsing container will translate the loaded Jsp page into a particular servlet.

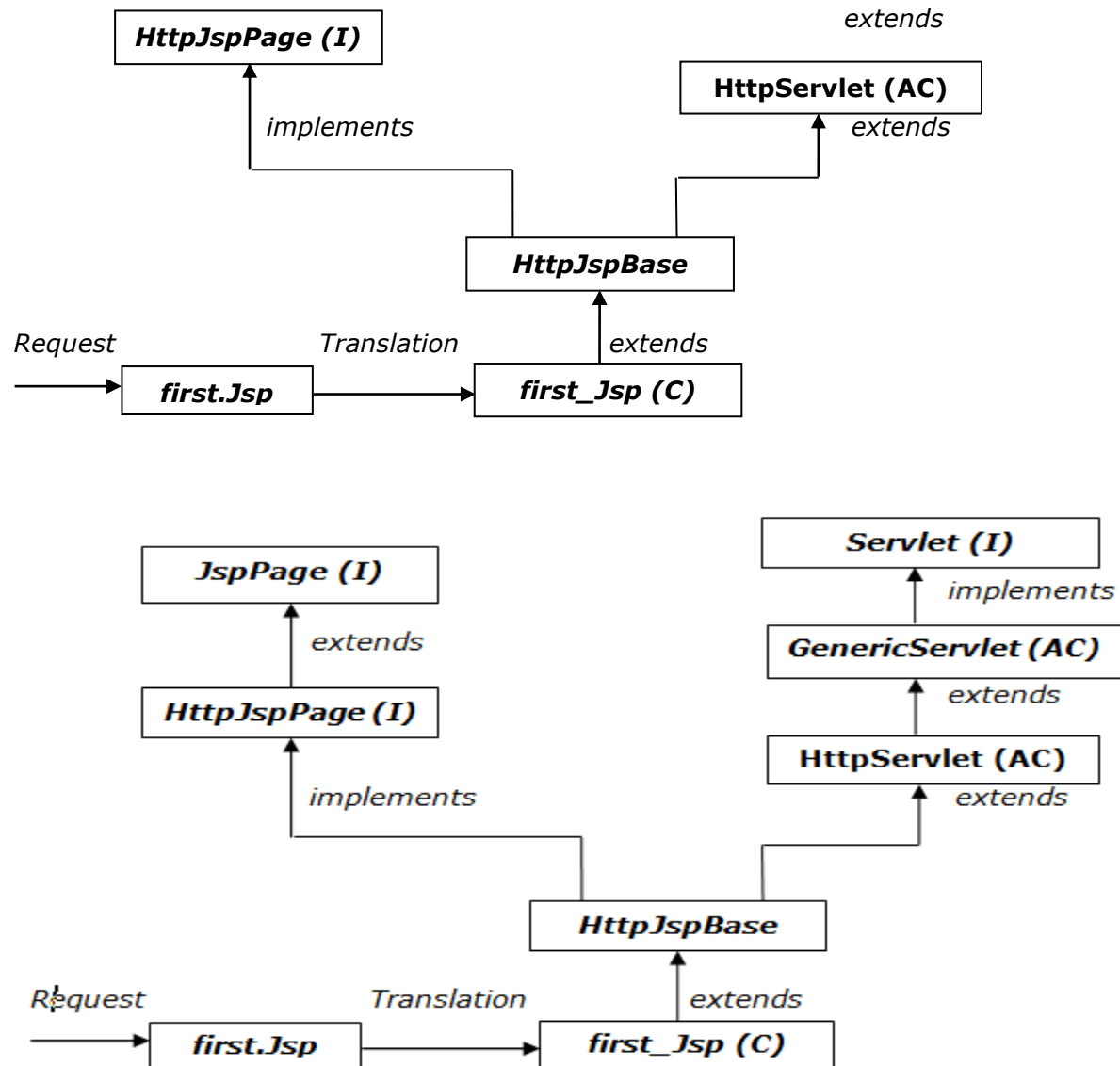
While executing a Jsp page Tomcat container will provide the translated servlet in the following location at Tomcat Server.

C:\Tomcat7.0\work\catalina\localhost\org\apache\Jsp\first_Jsp.java

If the Jsp file name is first.jsp then Tomcat Server will provide a servlet with name first_jsp. By default all the translated servlets provided by Tomcat container are final.

The default super class for translated servlet is **HttpJspBase**.





Where *JspPage* interface has declared the following methods.

```
public void _jspInit()
public void _jspDestroy()
```

Where *HttpJspPage* interface has provided the following method.

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
```

For the above 3 abstract methods *HttpJspBase* class has provided the default implementation but *_jspService(,)* method would be overridden in *first_jsp* class with the content what we provided in *first.jsp* file.

4. Servlet Compilation:

After getting the translated servlet container will compile servlet java file and generates the respective .class file.

5. Servlet Loading:

Here container will load the translated servlet class byte code to the memory.

6. Servlet Instantiation:

Here container will create object for the loaded servlet.

7. Servlet Initialization:

Here container will access `_jspInit()` method to initialize the servlet.

8. Creating request and response objects:

After the servlet initialization container will create a thread to access `_jspService(____)` method, for this container has to create `HttpServletRequest` and `HttpServletResponse`.

9. Generating Dynamic response:

After getting request and response objects container will access `_jspService(____)` method, by executing its content container will generate some response on response object.

10. Dispatching Dynamic response to Client:

When container generated thread reached to the ending point of `_jspService(____)` method then that thread will be in Dead state, with this container will dispatch dynamic response to client through the Response Format prepared by the protocol.

11. Destroying request and response objects:

When the dynamic response reached to client protocol will terminate its virtual socket connection, with this container will destroy request and response objects.

12. Servlet Deinstantiation:

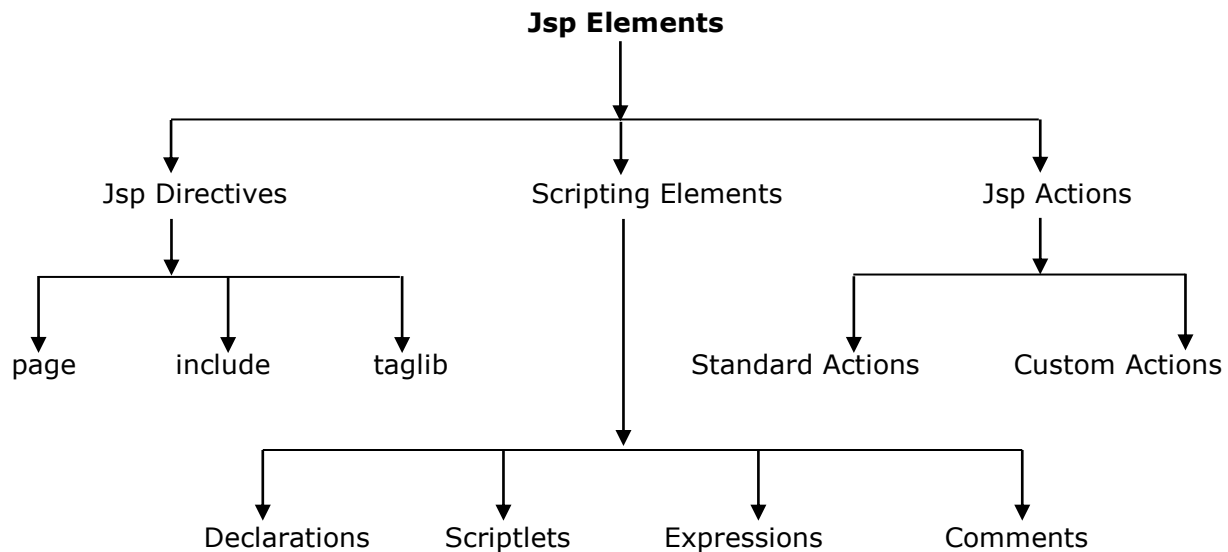
After destroying request and response objects container will be in waiting state depends on the container, then container identifies no further request for the same resource then container will destroy servlet object, for this container will execute `_jspDestroy()` method.

13. Servlet Unloading and Jsp Unloading:

After the servlet deinstantiation container will eliminate the translated servlet byte code and Jsp code from memory.

Jsp Elements:

In web applications to design Jsp pages we have to use the following elements.



Q: What are the differences between Jsp Directives and Scripting Elements?

Ans: 1. In web applications, Jsp Directives can be used to define present Jsp page characteristics, to include the target resource content into the present Jsp page and to make available user defined tag library into the present Jsp page.

In web applications, Jsp Scripting Elements can be used to provide code in Jsp pages.

2. All the Jsp Directives will be resolved at the time of translating Jsp page to servlet.

All the Jsp Scripting Elements will be resolved at the time of request processing.

3. Majority of the Jsp Directives will not give direct effect to response generation, but majority of Scripting Elements will give direct effect to response generation.

Q: To design Jsp pages we have already Jsp Scripting Elements then what is requirement to go for Jsp Actions?

Ans: In Jsp applications, Scripting Elements can be used to allow java code inside Jsp pages but the main theme of Jsp technology is not to allow java code inside the Jsp pages.

In the above context, to preserve the theme of Jsp technology we have to eliminate scripting elements from Jsp pages, for this we have to provide an alternative i.e. Jsp Actions provided by Jsp technology.

In case of Jsp Actions, we will define scripting tag in place of java code, in Jsp pages and we will provide the respective java code inside the classes folder.

In this context, when Jsp container encounter the scripting tag then container will execute the respective java code and perform a particular action called as **Jsp Action**.

1. Jsp Directives:

To provide Jsp Directives in Jsp pages we have to use the following syntaxes.

1. Jsp-Based Syntax:

`<%@Directive_name [attribute-list]%%>`

Ex: `<%@page import="java.io.*"%>`

2. XML-Based Syntax:

`<jsp:directive.directiveName[attribute-list]%/>`

Ex: `<jsp:directive.page import="java.io.*"/>`

There are 3 types of Directives in Jsp technology.

1. Page Directive
2. Include Directive
3. Taglib Directive

Jsp Directives :-

The jsp directives are messages that tells the web container how to translate a jsp page into the corresponding servlet.

There are three types of directives in jsp.

Directive**Description**

<%@ page attribute="value" %>

Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.

<%@ include attribute="value" %>

Includes a file during the translation phase.

<%@ taglib attribute="value" %>

Declares a tag library, containing custom actions, used in the page

Page Directive:-

Page Directive can be used to define the present Jsp page characteristics like to define import statements, specify particular super class to the translated servlet, to specify metadata about present Jsp pages and so on.

<%@page [attribute-list]%>

Where attribute-list in Jsp page directive may include the following list.

Language	contentType	import	extends
Info	buffer	autoFlush	errorPage
isErrorPage	session	isThreadSafe	isELIgnored

✓ **language:-**

This attribute can be used to specify language.

The default value of this attribute is java.

<%@page language="java"%>

✓ **contentType:**

This attribute will take a particular MIME type in order to give an intimation to the client about to specify the type of response which Jsp page has generated.

<%@page contentType="text/html"%>

The default value of this attribute is text/html.

✓ **import:**

This attribute can be used to import a particular packages into the present Jsp pages.

<%@page import="java.io.*"%>

If we want to import multiple number of packages into the present Jsp pages then we have to use either of the following 2 approaches.

Specify multiple number of packages with comma(,) .

<%@page import="java.io.*,java.util.*,java.sql.*"%>

Provide multiple number of import attributes for the list of packages.

<%@page import="java.io.*" import="java.util.*" import="java.sql.*"%>

The default values of this attribute are java.lang, javax.servlet, javax.servlet.http, javax.servlet.jsp.

Note: Among all the Jsp page attributes only import attribute is repeatable attribute, no other attribute is repeatable.

✓ **extends:**

This attribute will take a particular class name, it will be available to the translated servlet as super class.

<%@page extends="com.dss.MyClass"%>

Where MyClass should be an implementation class to `HttpJspPage` interface and should be a subclass to `HttpServlet`.

The default value of this attribute is `HttpJspBase` class.

✓ **info:**

This attribute can be used to specify some metadata about the present Jsp page.

<%@page info="First Jsp Application"%>

If we want to get the specified metadata programmatically then we have to use the following method from `Servlet` interface.

`public String getServletInfo()`

The default value of this attribute is `Jasper JSP2.2 Engine`.

✓ **buffer:**

This attribute can be used to specify the particular size to the buffer available in `JspWriter` object.

Note: Jsp technology is having its own writer object to track the generated dynamic response, `JspWriter` will provide very good performance when compared with `PrintWriter` in servlets.

<%@page buffer="52kb"%>

The default value of this attribute is `8kb`.

✓ **autoFlush:**

It is a boolean attribute, it can be used to give an intimation to the container about to flush or not to flush dynamic response to client automatically when `JspWriter` buffer filled with the response completely.

If `autoFlush` attribute value is `true` then container will flush the complete response to the client from the buffer when it reaches its maximum capacity.

If `autoFlush` attribute value is `false` then container will raise an exception when the buffer is filled with the response.

**org.apache.jasper.JasperException:An exception occurred processing JSP page/first.jsp at line:9
root cause: java.io.IOException:Error:Jsp Buffer Overflow.**

**Note: if we provide `0kb` as value for buffer attribute and `false` as value for autoFlush attribute then container will raise an exception like `org.apache.jasper.JasperException:/first.jsp(1,2)
jsp.error.page.badCombo`**

The default value of this attribute is `true`.

```
<%@ page language="java" contentType="text/html" buffer="1kb" autoFlush="true"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
```

```
</head>
<body>
    <%
        for(int i=0; i<1000; i++) {
            out.println("RAMA");
        }
    %>
</body>
</html>
```

Observation :-

`<%@ page language="java" contentType="text/html" buffer="1kb" autoFlush="false"%>`
Caused by: java.io.IOException: Error: JSP Buffer overflow

✓ **errorPage:**

This attribute can be used to specify an error page to execute when we have an exception in the present Jsp page.

`<%@page errorPage="error.jsp"%>`

✓ **isErrorPage:**

It is a boolean attribute, it can be used to give an intimation to the container about to allow or not to allow exception implicit object into the present Jsp page.

If we provide value as true to this attribute then container will allow exception implicit object into the present Jsp page.

If we provide value as false to this attribute then container will not allow exception implicit object into the present Jsp page.

The default value of this attribute is false.

Ex: `<%@page isErrorPage="true"%>`

first.jsp:

```
<%@ page language="java" contentType="text/html" import="java.util.*" errorPage="error.jsp"%>
<html>
<body>
<%!Date d=null;%>
<%
    out.println(d.toString());
%>
</body>
</html>
```

error.jsp:

```
<%@ page language="java" contentType="text/html" isErrorPage="true"%>
<html>
<body>
    <%=exception%>
</body>
</html>
```

10. session:

It is a boolean attribute, it is give an intimation to the container about to allow or not to allow session implicit object into the present Jsp page. The default value of this attribute is true.

Ex: `<%@page session="true"%>`

11. isThreadSafe:

It is a boolean attribute, it can be used to give an intimation to the container about to allow or not to allow multiple number of requests at a time into the present Jsp page.

If we provide true as value to this attribute then container will allow multiple number of requests at a time.

If we provide false as value to this attribute then automatically container will allow only one request at a time and it will implement `SingleThreadModel` interface in the translated servlet.

The default value of this attribute is true.

Ex: `<%@page isThreadSafe="true"%>`

12. isELIgnored:

It is a boolean attribute, it can be used to give an intimation to the container about to allow or not to allow Expression Language syntaxes in the present Jsp page.

Note: Expression Language is a Scripting language, it can be used to eliminate java code completely from the Jsp pages.

If `isELIgnored` attribute value is true then container will eliminate Expression Language syntaxes from the present Jsp page.

If we provide false as value to this attribute then container will allow Expression Language syntaxes into the present Jsp pages.

The default value of this attribute is false.

Ex: `<%@page isELIgnored="true"%>`

Include Directive:

Include Directive can be used to include the content of the target resource into the present Jsp page.

```
<%@include file="--"%">
```

Where file attribute can be used to specify the name and location of the target resource.

logo.jsp:

```
<html>
  <body><center>
    <table width="100%" height="20%" bgcolor="red">
      <tr><td colspan="2"><center><b><font size="7" color="white">
        Ratan Software Solutions
      </font></b></center></td></tr>
    </table></center></body>
</html>
```

footer.jsp:

```
<html>
  <body><center>
    <table width="100%" height="15%" bgcolor="blue">
      <tr><td colspan="2"><center><b><font size="6" color="white">
        copyrights2010-2020@Ratansoftwareolutions
      </font></b></center></td></tr>
    </table></center></body>
</html>
```

body.jsp:

```
<html>
  <body bgcolor="lightyellow">
    <center><b><font size="7">
      <p><br>
        Ratan Software Solutions is one of the Training Institute.
      <br><br></p>
    </font></b></center></body>
</html>
```

mainpage.jsp:

```
<%@include file="logo.jsp"%>
<%@include file="body.jsp"%>
< %@include file="footer.jsp"%>
```

Note : in including mechanism all the dependent file content is included in main file during translation page.

Mainpage_jsp.java:- (Translation File)

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class main_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
    private static final javax.servlet.jsp.JspFactory _jspxFactory =
        javax.servlet.jsp.JspFactory.getDefaultFactory();
    private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;
    static {
        _jspx_dependants = new java.util.HashMap<java.lang.String,java.lang.Long>{3};
        _jspx_dependants.put("/body.jsp", Long.valueOf(1458693914551L));
        _jspx_dependants.put("/footer.jsp", Long.valueOf(1458693895602L));
        _jspx_dependants.put("/header.jsp", Long.valueOf(1458693869263L));
    }
    private javax.el.ExpressionFactory _el_expressionfactory;
    private org.apache.tomcat.InstanceManager _jsp_instancemanager;
    public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
        return _jspx_dependants;
    }
    public void _jspInit() {
        _el_expressionfactory =
        _jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
        _jsp_instancemanager =
        org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
    }
    public void _jspDestroy() { }
    public void _jspService(final javax.servlet.http.HttpServletRequest request, final
    javax.servlet.http.HttpServletResponse response)
        throws java.io.IOException, javax.servlet.ServletException {
        final javax.servlet.jsp.PageContext pageContext;
        javax.servlet.http.HttpSession session = null;
        final javax.servlet.ServletContext application;
        final javax.servlet.ServletConfig config;
        javax.servlet.jsp.JspWriter out = null;
        final java.lang.Object page = this;
        javax.servlet.jsp.JspWriter _jspx_out = null;
        javax.servlet.jsp.PageContext _jspx_page_context = null;
        try {
            response.setContentType("text/html");
            pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
            _jspx_page_context = pageContext;
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;
            out.write("\r\n");
            out.write("<html>\r\n");
            out.write("<body>\r\n");
            out.write(" ");
        }
    }
}
```

```

out.write("\r\n");
out.write("<html>\r\n");
out.write("  <body><center>\r\n");
out.write("    <table width=\"100%\" height=\"20%\" bgcolor=\"red\">\r\n");
out.write("      <tr><td colspan=\"2\"><center><b><font size=\"7\" color=\"white\">\r\n");
out.write("        Ratan Software Solutions\r\n");
out.write("      </font></b></center></td></tr>\r\n");
out.write("    </table></center></body>\r\n");
out.write("</html>\r\n");
out.write("\r\n");
out.write("  ");
out.write("\r\n");
out.write("<html>\r\n");
out.write("  <body bgcolor=\"lightyellow\">\r\n");
out.write("    <center><b><font size=\"7\">\r\n");
out.write("      <p><br>\r\n");
out.write("      Ratan Software Solutions is one of the Training Institute.\r\n");
out.write("    <br><br></p>\r\n");
out.write("    </font></b></center></body>\r\n");
out.write("</html>\r\n");
out.write("\r\n");
out.write("  ");
out.write("\r\n");
out.write("<html>\r\n");
out.write("  <body><center>\r\n");
out.write("    <table width=\"100%\" height=\"15%\" bgcolor=\"blue\">\r\n");
out.write("      <tr><td colspan=\"2\"><center><b><font size=\"6\" color=\"white\">\r\n");
out.write("        copyrights2010-2020@Ratansoftwareolutions\r\n");
out.write("      </font></b></center></td></tr>\r\n");
out.write("    </table></center></body>\r\n");
out.write("</html>\r\n");
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>");
} catch (java.lang.Throwable t) {
if (!(t instanceof javax.servlet.jsp.SkipPageException)){
  out = _jspx_out;
  if (out != null && out.getBufferSize() != 0)
    try {
      if (response.isCommitted()) {
        out.flush();
      } else {
        out.clearBuffer();
      }
    } catch (java.io.IOException e) {}
  if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
  else throw new ServletException(t);
}
} finally {
  _jspxFactory.releasePageContext(_jspx_page_context);
}
}
}

```

Login Application :**Login.html**

```
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action=../Login.jsp method="post">
User name:<input type="text" name="username" /><br>
Password:<input type="password" name="password" /><br>
<input type="submit" value="login">
</form>
</body>
</html>
```

Login.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>

<%!
String uname;
String upwd;
%>

<%
uname = request.getParameter("username");
upwd = request.getParameter("password");
if(uname.equals("ratan") && upwd.equals("ratan"))
{
out.println("login success");
}
else
{
out.println("login failure");
}
%>
<br/>
user name : <%=uname %>
<br/>
user Password : <%=upwd %>
</body>
</html>
```

Note : check the converted servlet **Login_jsp.java** in the following location.

1. Declaration tag elements present in servlet class outside of the methods.
2. Script lets tag data present in `_jspService()` method.
3. Expression tag data present as a parameter of `out.print(eid)` method.

3. Taglib Directive:

The main purpose of **Taglib Directive** is to make available user defined tag library into the present Jsp pages.

Syntax: `<%@taglib uri="__ __" prefix="__ __"%>`

Where uri attribute can be used to specify the name and location of user defined tag library.

Where prefix attribute can be used to define prefix names for the custom tags.

Ex: `%@taglib uri="/WEB-INF/db.tld" prefix="connect"%`

JSP Scripting Elements:

Jsp scripting elements enable you to write the java code inside the jsp file. There are 3 types of Scripting Elements.

Expressions : `<%= expression%>` that are evaluated and inserted into output,

Scriptlets : `<% code %>` that are inserted into the servlets service method,

Declarations : `<%! code %>` that are inserted into the body of the servlet class, outside of any methods.

1) Declarations:

It can be used to declare the variable declarations, method definitions, classes declaration...etc

The code written inside the jsp declaration tag is placed outside the service() method of servlet.

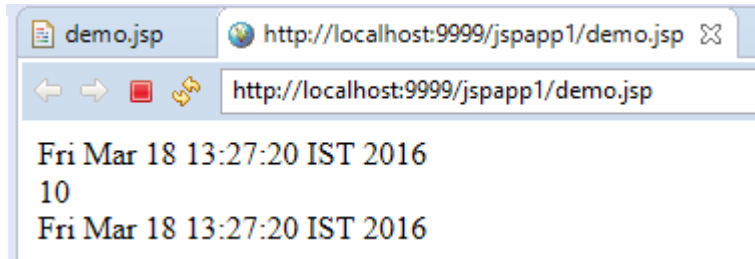
Syntax : `<%! Field or method Declarations; %>`

Example : `<% int i = 0;%>`
`<%! int add(int a,int b){`
`return a+b; }`
`%>`

Application:-

```
<%@ page language="java" contentType="text/html" import="java.util.*"%>
<html>
<body>
```

```
<%! Date d;%>
<%    d=new Date();
      out.println(d);
      out.println("<br>");
      out.println(2*5);
      out.println("<br>");
%>
<%=d%>
</body>
</html>
```



2. Scriptlets: This Scripting Element can be used to provide a block of java code.

Syntax: `<% Block of java code %>`

If we provide any block of java code by using scriptlets then that code will be available to translated servlet inside `_jspService(_)` method.

3. Expressions:

This is scripting element can be used to evaluate the single java expression and display that expression resultant value onto the client browser.

Syntax: `<%=Java Expression%>`

If we provide any java expression in expression scripting element then that expression will be available to translated servlet inside `_jspService(_)` method as a parameter to `out.write()` method.

Ex: `out.write(Java Expression);`

first.jsp:

```
<%@page import="java.util.*"%>
<%!
Date d=null;
String date=null;
%>
<%
d=new Date();
date=d.toString();
%>
<html>
<body bgcolor="lightyellow">
<center><b><font size="6" color="red"><br><br>
Today Date : <%=date%>
</font></b></center></body>
</html>
```

Translated Servlet:

```
-----  
-----  
import java.util.*;  
public final class first_jsp extends HttpJspBase implements JspSourceDependent  
{  
    Date d=null;  
    String date=null;  
    public void _jspInit()throws ServletException  
    { }  
    public void _jspDestroy()  
    { }  
    public void _jspService(HttpServletRequest req, HttpServletResponse res)throws SE, IOE  
    {  
        d=new Date();  
        date=d.toString();  
        out.write("<html>");  
        out.write("<body bgcolor='lightyellow'>");  
        out.write("<center><b><font size='6' color='red'><br><br>");  
        out.write("Today Date.....");  
        out.write(date);  
        out.write("</font></b></center></body></html>");  
    }  
}
```

4. Jsp Comments:

In Jsp pages, we are able to use the following 3 types of comments.

1. XML-Based Comments
2. Jsp-Based Comments
3. Java-Based Comments inside Scripting Elements

1. XML-Based Comments:

```
<!--  
    -----  
    -----  
    -----  
-->
```

} Description

2. Jsp-Based Comments:

```
<%--  
    -----  
    -----  
    -----  
--%>
```

} Description

3. Java-Based Comments inside Scripting Elements:

1. Single Line Comment:

```
//-----Description-----
```

2. Multiline Comment:

```
/*  
-----  
-----  
-----  
*/
```

} Description

3. Documentation Comment:

```
/**  
-----  
-----  
-----  
*/
```

} Description

Jsp Implicit Objects:

In web applications, all the web developers may require some objects like request, response, config, context, and session and so on are frequent requirement, to get these objects we have to write some piece of java code.

Once the above specified objects are as frequent requirement in web applications, Jsp technology has provided them as predefined support in the form of Jsp Implicit Objects in order to reduce burden on the developers.'

1. out -----> javax.servlet.jsp.JspWriter
2. request -----> javax.servlet.HttpServletRequest
3. response -----> javax.servlet.HttpServletResponse
4. config -----> javax.servlet.ServletConfig
5. application -----> javax.servlet.ServletContext
6. session -----> javax.servlet.http.HttpSession
7. exception -----> java.lang.Throwable
8. page -----> java.lang.Object
9. pageContext -----> javax.servlet.jsp.PageContext

Application :**Login.html:**

```
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<form action=../Login.jsp method="post">
User name:<input type="text" name="username" /><br>
Password:<input type="password" name="password" /><br>
<input type="submit" value="login">
</form>
</body>
</html>
```

Login.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>

<%!
String uname;
String upwd;
%>

<%
uname = request.getParameter("username");
upwd = request.getParameter("password");
if(uname.equals("ratan") && upwd.equals("ratan"))
{
    response.sendRedirect("http://www.facebook.com");
}
else
{
    response.sendError(777,"login fail try with valid user name & password");
}
%>
<br/>
user name : <%=uname %>
<br/>
user Password : <%=upwd %>
</body>
</html>
```

Q: What is the difference between PrintWriter and JspWriter?

Ans: PrintWriter is a writer in servlet applications, it can be used to carry the response. PrintWriter is not BufferedWriter so that its performance is very less in servlets applications.

JspWriter is a writer in Jsp technology to carry the response. JspWriter is a BufferedWriter so that its performance will be more when compared with PrintWriter.

Q: What is PageContext? and What is the purpose of PageContext in Jsp Applications?

Ans: PageContext is an implicit object in Jsp technology, it can be used to get all the Jsp implicit objects even in non-jsp environment.

To get all the Jsp implicit objects from PageContext we have to use the following method.

```
public Xxx getXxx()
```

Where Xxx may be out, request, response and so on.

Ex: JspWriter out=pageContext.getOut();

```
HttpServletRequest request=pageContext.getRequest();
```

```
ServletConfig config=pageContext.getServletConfig();
```

Note: While preparing Tag Handler classes in custom tags container will provide pageContext object as part of its life cycle, from this we are able to get all the implicit objects.

In Jsp applications, by using pageContext object we are able to perform some operations with the attributes in Jsp scopes page, request, session and application like adding an attribute, removing an attribute, getting an attribute and finding an attribute.

To set an attribute onto a particular scope pageContext has provided the following method.

```
public void setAttribute(String name, Object value, int scope)
```

Where scopes may be

```
public static final int PAGE_SCOPE=1;  
public static final int REQUEST_SCOPE=2;  
public static final int SESSION_SCOPE=3;  
public static final int APPLICATION_SCOPE=4;
```

Ex: <%
pageContext.setAttribute("a", "aaa", pageContext.REQUEST_SCOPE);
%>

To get an attribute from page scope we have to use the following method.

```
public Object getAttribute(String name)
```

Ex: `String a=(String)pageContext.getAttribute("a");`

If we want to get an attribute value from the specified scope we have to use the following method.

```
public Object getAttribute(String name, int scope)
```

Ex: `String uname=(String)pageContext.getAttribute("uname",
pageContext.SESSION_SCOPE);`

To remove an attribute from a particular we have to use the following method.

```
public void removeAttribute(String name, int scope)
```

Ex: `pageContext.removeAttribute("a", pageContext.SESSION_SCOPE);`

To find an attribute value from page scope, request scope, session scope and application scope we have to use the following method.

```
Public Object findAttribute(String name)
```

Ex: `String name=pageContext.findAttribute("uname");`

3. Jsp Actions:

In Jsp technology, by using scripting elements we are able to provide java code inside the Jsp pages, but the main theme of Jsp technology is not to allow java code inside Jsp pages.

To eliminate java code from Jsp pages we have to eliminate scripting elements, to eliminate scripting elements from Jsp pages we have to provide an alternative i.e. Jsp Actions.

In case of Jsp Actions, we will define a scripting tag in Jsp page and we will provide a block of java code w.r.t. scripting tag.

When container encounters the scripting tag then container will execute respective java code, by this an action will be performed called as Jsp Action.

In Jsp technology, there are 2 types of actions.

1. Standard Actions
2. Custom Actions

1. Standard Actions:

Standard Actions are Jsp Actions, which could be defined by the Jsp technology to perform a particular action.

Jsp technology has provided all the standard actions in the form of a set of predefined tags called Action Tags.

1. `<jsp:useBean---->`
2. `<jsp:setProperty---->`
3. `<jsp:getProperty---->`

4. <jsp:include---->
5. <jsp:forward---->
6. <jsp:param---->
7. <jsp:plugin---->
8. <jsp:fallback---->
9. <jsp:params---->
10. <jsp:declaration---->
11. <jsp:scriptlet---->
12. <jsp:expression---->

Java Beans:

Java Bean is a reusable component.

Java Bean is a normal java class which may declare properties, setter and getter methods in order to represent a particular user form at server side.

If we want to prepare Java Bean components then we have to use the following rules and regulations.

1. Java Bean is a normal java class, it is suggestible to implement Serializable interface.
2. Always Java Bean classes should be public, non-abstract and non-final.
3. In Java Bean classes, we have to declare all the properties w.r.t. the properties define in the respective user form.
4. In Java Bean classes, all the properties should be private.
5. In Java Bean classes, all the behaviours should be public.
6. If we want to declare any constructor in Java Bean class then that constructor should be public and zero argument.

<jsp:useBean>:

- ✓ The main purpose of <jsp:useBean> tag is to interact with bean object from a particular Jsp page.
- ✓ This tag is used to locate or instantiate bean class. If the bean object is already created it doesn't create a bean based on scope. But if the bean is not created it instantiates the bean.

<jsp:useBean id="--" class="--" type="--" scope="--"/>

Id = instance name

Class = fully qualified name of the bean class

Scope =page request session application (scope of the bean object)

Where type attribute will take the fully qualified name of Bean class to define the type of variable in order to manage Bean object reference.

1. Page Scope:

If we declare the data in page scope by using pageContext object then that data should have the scope up to the present Jsp page.

The default scope is page scope

2. Request Scope:

If we declare the data in request object then that data should have the scope up to the number of resources which are visited by the present request object.

3. Session Scope:

Specifies that you can use this bean from any JSP page in the same session whether processes the same request or not.

4. Application Scope:

specifies that you can use this bean from any JSP page in the same application

Note: In <jsp:useBean> tag, always it is suggestible to provide either application or session scope to the scope attribute value.

Ex: <jsp:useBean id="e" class="Employee" type="Employee" scope="session"/>

When container encounters the above tag then container will pick up class attribute value i.e. fully qualified name of Bean class then container will recognize Bean class .class file and perform Bean class loading and instantiation.

After creating Bean object container will assign Bean object reference to the variable specified as value to id attribute.

After getting Bean object reference container will store Bean object in a scope specified as value to scope attribute.

2. <jsp:setProperty>:

The main purpose of <jsp:setProperty> tag is to execute a particular setter method in order to set a value to a particular Bean property.

Syntax: <jsp:setProperty name="--" property="--" value="--"/>

Where name attribute will take a variable which is same as id attribute value in <jsp:useBean> tag.

Where property attribute will take a property name in order to access the respective setter method.

Where value attribute will take a value to pass as a parameter to the respective setter method.

3. <jsp:getProperty>:

The main purpose of <jsp:getProperty> tag is to execute a getter method in order to get a value from Bean object.

Syntax: <jsp:getProperty name="--" property="--"/>

Where name attribute will take a variable which is same as id attribute value in <jsp:useBean> tag.

Where property attribute will take a particular property to execute the respective getter method.

Note: In case of <jsp:useBean> tag, in general we will provide a separate <jsp:setProperty> tag to set a particular value to the respective property in Bean object.

In case of <jsp:useBean> tag, it is possible to copy all the request parameter values directly onto the respective Bean object.

To achieve this we have to provide "*" as value to property attribute in <jsp:setProperty> tag.

Ex: <jsp:setProperty name="e" property="*" />

If we want to achieve the above requirement then we have to maintain same names in the request parameters i.e. form properties and Bean properties.

Note: The above "*" notation is not possible with <jsp:getProperty> tag.

```
<jsp:useBean id="person" class="PersonBean"
              scope="request" />
```

Form.html

```
<html>
<h5> *****Employee Details*****</h5>
<body>
<form method="post" action="main.jsp">
    Employee id :<input type="text" name="eid"/><br>
    Employee name : <input type="text" name="ename"/><br>
    Employee salary :<input type="text" name="esal"/><br>
    <input type="submit" value="submit"/>
</form>
</body>
</html>
```

Main.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
    <%!    int eid;
           String ename;
           double esal;
    %>
```

```

<%      eid = Integer.parseInt(request.getParameter("eid"));
        ename = request.getParameter("ename");
        esal = Double.parseDouble(request.getParameter("esal"));
    %>
    <jsp:useBean id="eb" class="com.sravva.EmpBean" scope="page"/>
    <jsp:setProperty property="ename" name="eb" value="<%=ename%>"/>
    <jsp:setProperty property="eid" name="eb" value="<%=eid%>"/>
    <jsp:setProperty property="esal" name="eb" value="<%=esal%>"/>
    <h5>*****Employee Details*****</h5>
    Employee name : <jsp:getProperty property="ename" name="eb"/> <br>
    Employee id : <jsp:getProperty property="eid" name="eb"/> <br>
    Employee salary: <jsp:getProperty property="esal" name="eb"/> <br>
</body>
</html>

```

EmpBean.java

```

package com.sravva;
import java.io.Serializable;
public class EmpBean implements Serializable{
    int eid;
    String ename;
    double esal;
    public int getId() {
        return eid;
    }
    public void setId(int eid) {
        this.eid = eid;
    }
    public String getName() {
        return ename;
    }
    public void setName(String ename) {
        this.ename = ename;
    }
    public double getEsal() {
        return esal;
    }
    public void setEsal(double esal) {
        this.esal = esal;
    }
}

```

4. <jsp:include>:

Q: What are the differences between include directive and <jsp:include> action tag?

Ans: 1. In Jsp applications, **include directive** can be used to include the content of the target resource into the present Jsp page.

In Jsp pages, **<jsp:include>** action tag can be used to include the target resource response into the present Jsp page response.

2. In general include directive can be used to include static resources where the frequent updations are not available.

<jsp:include> action tag can be used to include dynamic resources where the frequent updations are available.

3. In general directives will be resolved at the time of translation and actions will be resolved at the time of request processing. Due to this reason include directive will be resolved at the time of translation but **<jsp:include>** action tag will be resolved at the time of request processing.

If we are trying to include a target Jsp page into present Jsp page by using include directive then container will prepare only one translated servlet.

To include a target Jsp page into the present Jsp page if we use **<jsp:include>** action tag then container will prepare 2 separate translated servlets.

In Jsp applications, include directives will provide static inclusion, but **<jsp:include>** action tag will provide dynamic inclusion.

In Jsp technology, **<jsp:include>** action tag was designed on the basis of Include Request Dispatching Mechanism.

Syntax: **<jsp:include page="--" flush="--"/>**

Where page attribute will take the name and location of the target resource to include its response.

Where flush is a boolean attribute, it can be used to give an intimation to the container about to autoFlush or not to autoFlush dynamic response to client when JspWriter buffer filled with the response at the time of including the target resource response into the present Jsp page.

-----Application3-----

includeapp:

addform.html:

```
<html>
  <body bgcolor="lightgreen">
    <form action="add.jsp">
      <pre>
        <u>Product Details</u>
        Product Id : <input type="text" name="pid"/>
        Product Name : <input type="text" name="pname"/>
        Product Cost : <input type="text" name="pcost"/>
        <input type="submit" value="ADD"/>
      </pre>
    </form>
  </body>
```

</html>

add.jsp:

```
<%@page import="java.sql.*"%>
<%!
```

```
    String pid;
    String pname;
    int pcost;
    static Connection con;
    static Statement st;
    ResultSet rs;
    ResultSetMetaData rsmd;
    static{
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"ratan");
            st=con.createStatement();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
%>
<%
    try{
        pid=request.getParameter("pid");
        pname=request.getParameter("pname");
        pcost=Integer.parseInt(request.getParameter("pcost"));
        st.executeUpdate("insert into product
values('"+pid+"','"+pname+"','"+pcost+"')");
        rs=st.executeQuery("select * from product");
        rsmd=rs.getMetaData();
        int count=rsmd.getColumnCount();
%>
        <html><body><center>
        <table border="1" bgcolor="lightyellow">
        <tr>
<%
            for (int i=1;i<=count;i++){
%>
                <td><b><font size="6" color="red">
                <center><%=rsmd.getColumnName(i) %></center>
                </font></b></td>
<%
            }
%>
        </tr>
<%
        while(rs.next()){
%>
        <tr>
```

```
<%
    for(int i=1;i<=count;i++){
%>
        <td><b><font size="6">
<%=rs.getString(i) %>
        </font></b></td>
<%
    }
%>
</tr>
<%
    }
%>
</table></center></body></html>
<%
    }
    catch(Exception e){
        e.printStackTrace();
    }
%>
    <hr>
    <jsp:include page="addform.html" flush="true"/>
```

Include Directive:-

- 1) It is used to include the target jsp page into present jsp page.
- 2) Include directive include the target page data into present jsp file at translation time(jsp-servlet).
- 3) If any modification done in jsp file will not be visible until jsp file compiles again.
- 4) Include directive is static import.
- 5) It is better for static pages, if we are using for dynamic pages for every change it requires translation.
- 6) By using include directive we are unable to pass parameters to target page.

Syntax:-

```
<%@ include file="file_name" %>
```

Example:-

```
<%@ include file="hello.jsp" %>
```

Include Action:-

- 1) It is used to include the target jsp into present jsp page.
- 2) The content of the target jsp included at runtime instead of translation time.
- 3) If you done the modifications these are effected when we send the request to jsp.
- 4) Include action is dynamic import.
- 5) Instead of including original data it is calling include method at runtime.
- 6) By using include action we are able to pass parameters to the included page.

```
<jsp:include page="file_name" />
```

```
<jsp:param name="parameter_name" value="parameter_value" />
```

```
</jsp:include>
```

Syntax:-

```
<jsp:include page="file_name" />
```

Example:-

```
<jsp:include page="hello.jsp"/>
```

main.jsp:-

```
<%@ page language="java" contentType="text/html" %>
```

```
<html>
```

```
<body>
```

```
This is info about main.jsp file<br><br><br>
```

```
<jsp:include page="hello.jsp">
```

```
<jsp:param name="a" value="10"/>
```

```
</jsp:include>
```

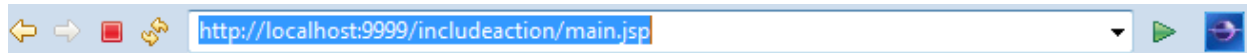
```
</body>
```

```
</html>
```

hello.jsp:-

```
<%@ page language="java" contentType="text/html" %>
```

```
<%@page import="java.util.*"%>
<html>
<head>
<body>
    hi this is hello.jsp file content<br>
    welcome to Sravyainfotech<br>
    <%= "Today date =" + new Date() %><br>
    <%= "param tag value=" + request.getParameter("a") %>
</body>
</html>
```



This is info about main.jsp file

hi this is hello.jsp file content
welcome to Sravyainfotech
Today date =Thu Aug 06 12:18:08 IST 2015
param tag value=10

5. <jsp:forward>:

Q: What are the differences between <jsp:include> action tag and <jsp:forward> action tag?

Ans: 1. <jsp:include> action tag can be used to include the target resource response into the present Jsp page.

<jsp:forward> tag can be used to forward request from present Jsp page to the target resource.

2. <jsp:include> tag was designed on the basis of Include Request Dispatching Mechanism.

<jsp:forward> tag was designed on the basis of Forward Request Dispatching Mechanism.

3. When Jsp container encounter <jsp:include> tag then container will forward request to the target resource, by executing the target resource some response will be generated in the response object, at the end of the target resource container will bypass request and response objects back to first resource, at the end of first resource execution container will dispatch overall response to client. Therefore, in case of <jsp:include> tag client is able to receive all the resources response which are participated in the present request processing.

When container encounters <jsp:forward> tag then container will bypass request and response objects to the target resource by refreshing response object i.e. by eliminating previous response available in response object, at the end of target resource container will dispatch the generated dynamic response directly to the client without moving back to first resource. Therefore, in case of <jsp:forward> tag client is able to receive only target resource response.

Syntax: <jsp:forward page="--"/>

Where page attribute specifies the name and location of the target resource.

login.html

```
<html>
<body>
<form action="main.jsp">
    <h5> *****Login Details*****</h5>
    User name : <input type="text" name="uname"/><br>
    Password : <input type="password" name="upwd"/><br>
    <input type="submit" value="login"/>
</form>
</body>
</html>
```

Main.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
    <%!    String uname;
           String upwd;
    %>

    <%
    String uname = request.getParameter("uname");
    String upwd = request.getParameter("upwd");
    if(uname.equals("sravya"))
    {
    %>
    <jsp:forward page="success.jsp"/>
    <%
    }
    else
    {out.println("****U R Login is fail try with another username & password*****");
    }%>

    <jsp:include page="login.html"/>

</body>
</html>
```

success.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
    Hi this is Success jsp<br>
    U r login is Success
</body>
</html>
```

6. <jsp:param>:

This action tag can be used to provide a name value pair to the request object at the time of by passing request object from present Jsp page to target page either in include mechanism or in forward mechanism or in both.

This tag must be utilized as a child tag to <jsp:include> tag and <jsp:forward> tags.

Syntax: <jsp:param name="--" value="--"/>

-----Application4-----

forwardapp:

registrationform.html:

```
<html>
  <body bgcolor="lightgreen">
    <form action="registration.jsp">
      <pre>
        <u>Registration Form</u>
        Name : <input type="text" name="pname"/>
        Age : <input type="text" name="uage"/>
        Address : <input type="text" name="uaddr"/>
        <input type="submit" value="Registration"/>
      </pre>
    </form>
  </body>
</html>
```

existed.jsp:

```
<center><h1>User Existed</h1></center>
```

success.jsp:

```
<center><h1>Registration Success</h1></center>
```

failure.jsp:

```
<center><h1>Registration Failure</h1></center>
```

registration.jsp:

```
<%@page import="java.sql.*"%>
<%!
    String uname;
    int uage;
    String uaddr;
    static Connection con;
    static Statement st;
    ResultSet rs;
    static{
```

```
        try{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"ratan");
            st=con.createStatement();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
%>
<%
    try{
        uname=request.getParameter("uname");
        uage=Integer.parseInt(request.getParameter("uage"));
        uaddr=request.getParameter("uaddr");
        rs=st.executeQuery("select * from reg_users where uname='"+uname+"'");
        boolean b=rs.next();
        if(b==true)
        {
%>
            <jsp:forward page="existed.jsp"/>
<%
                }
                else
                {
                    int rowCount=st.executeUpdate("insert into reg_users values
('"+uname+"','"+uage+"','"+uaddr+"')");
                    if(rowCount == 1)
                    {
%>
                        <jsp:forward page="success.jsp"/>
<%
                            }
                            else
                            {
%>
                                <jsp:forward page="failure.jsp"/>
<%
                                    }
                                }
                            }
                        }
                    catch(Exception e){
%>
                        <jsp:forward page="failure.jsp"/>
<%
                            e.printStackTrace();
                        }
%>
```

7. <jsp:plugin>:

This tag can be used to include an applet into the present Jsp page.

Syntax: <jsp:plugin code="--" width="--" height="--" type="--"/>

Where code attribute will take fully qualified name of the applet.

Where width and height attributes can be used to specify the size of applet.

Where type attribute can be used to specify which one we are going to include whether it is applet or bean.

Ex: <jsp:plugin code="Logo" width="1000" height="150" type="applet"/>

8. <jsp:params>:

In case of the applet applications, we are able to provide some parameters to the applet in order to provide input data.

Similarly if we want to provide input parameters to the applet from <jsp:plugin> tag we have to use <jsp:param> tag.

<jsp:param> tag must be utilized as a child tag to <jsp:params> tag.

<jsp:params> tag must be utilized as a child tag to <jsp:plugin> tag.

Syntax:

```
<jsp:plugin>
  <jsp:params>
    <jsp:param name="--" value="--"/>
    <jsp:param name="--" value="--"/>
    -----
  </jsp:params>
  -----
</jsp:plugin>
```

If we provide any input parameter to the applet then that parameter value we are able to get by using the following method from Applet class.

```
public String getParameter(String name)
```

Ex: String msg=getParameter("message");

-----Application5-----**pluginapp:****LogoApplet.java:**

```
import java.awt.*;
import java.applet.*;
public class LogoApplet extends Applet
{
    String msg;
    public void paint(Graphics g)
```

```

        {
            msg=getParameter("message");
            Font f=new Font("arial",Font.BOLD,40);
            g.setFont(f);
            this.setBackground(Color.blue);
            this.setForeground(Color.white);
            g.drawString(msg,150,70);
        }
    }

```

logo.jsp:

```

<jsp:plugin code="LogoApplet" width="1000" height="150" type="applet">
    <jsp:params>
        <jsp:param name="message" value="ratan software solutions"/>
    </jsp:params>
</jsp:plugin>

```

9. <jsp:fallback>:

The main purpose of <jsp:fallback> tag is to display an alternative message when client browser is not supporting <OBJECT----> tag and <EMBED----> tag.

Syntax: <jsp:fallback>-----Description-----</jsp:fallback>

In Jsp applications, we have to utilize <jsp:fallback> tag as a child tag to <jsp:plugin> tag.

Ex: <jsp:plugin code="LogoApplet" width="1000" height="150" type="applet">
 <jsp:fallback>Applet Not Allowed</jsp:fallback>
 </jsp:plugin>

10. <jsp:declaration>:

This tag is almost all same as the declaration scripting element, it can be used to provide all the Java declarations in the present Jsp page.

Syntax: <jsp:declaration>

 ----- } Java Declarations

 </jsp:declaration>

11. <jsp:scriptlet>:

This tag is almost all same as the scripting element scriptlets, it can be used to provide a block of Java code in Jsp pages.

Syntax: <jsp:scriptlet>

 ----- } Block of Java code

 </jsp:scriptlet>

12. <jsp:expression>:

This tag is almost all same as the scripting element expression, it can be used to provide a Java expression in the present Jsp page.

Syntax: <jsp:expression> Java Expression </jsp:expression>

Ex:

```
<%@page import="java.util.*"%>
<jsp:declaration>
    Date d=null;
    String date=null;
</jsp:declaration>
<jsp:scriptlet>
    d=new Date();
    date=d.toString();
</jsp:scriptlet>
<html>
    <body bgcolor="lightyellow">
        <center><b><font size="6" color="red"><br><br>
            Today Date : <jsp:expression>date</jsp:expression>
        </font></b></center></body>
</html>
```

2. Custom Actions:

In Jsp technology, by using Jsp directives we are able to define present Jsp page characteristics, we are unable to use Jsp directives to perform actions in the Jsp pages.

To perform actions if we use scripting elements then we have to provide Java code inside the Jsp pages.

The main theme of Jsp technology is not to allow Java code inside Jsp pages, to eliminate Java code from Jsp pages we have to eliminate scripting elements.

If we want to eliminate scripting elements from Jsp pages we have to use actions.

There are 2 types of actions in Jsp technology.

1. Standard Actions
2. Custom Actions

Standard Actions are predefined actions provided by Jsp technology, these standard actions are limited in number and having bounded functionalities so that standard actions are not sufficient to satisfy the complete client requirement.

In this context, there may be a requirement to provide Java code inside the Jsp pages so that to eliminate Java code completely from Jsp pages we have to use Custom Actions.

Custom Actions are Jsp Actions which could be prepared by the developers as per their application requirements.

In Jsp technology, standard actions will be represented in the form of a set of predefined tags are called as **Action Tags**.

Similarly all the custom actions will be represented in the form of a set of user defined tags are called as **Custom Tags**.

To prepare custom tags in Jsp pages we have to use the following syntax.

Syntax: <prefix_Nmae:tag_Name>

```
    -----  
    -----  
    -----  
    ----- } Body  
</prefix_Name>
```

If we want to design custom tags in our Jsp applications then we have to use the following 3 elements.

1. Jsp page with taglib directive
2. TLD(Tag Library Descriptor) file
3. TagHandler class

Where TagHandler class is a normal Java class it is able to provide the basic functionality for the custom tags.

Where TLD file is a file, it will provide the mapping between custom tag names and respective TagHandler classes.

Where taglib directive in the Jsp page can be used to make available the tld files into the present Jsp page on the basis of custom tags prefix names.

Internal Flow:

When container encounters a custom tag container will pick up the custom tag name and the respective prefix name then recognize a particular taglib directive on the basis of the prefix attribute value.

After recognizing taglib directive container will pick up uri attribute value i.e. the name and location of tld file then container will recognize the respective tld file.

After getting tld file container will identify the name and location of TagHandler class on the basis of custom tag name.

When container recognize the respective TagHandler class .class file then container will perform TagHandler class loading, instantiation and execute all the life cycle methods.

1. Taglib Directive:

In custom tags design, the main purpose of taglib directive is to make available the required tld file into the present Jsp page and to define prefix names to the custom tags.

Syntax: <%@taglib uri="--" prefix="--"%>

Where prefix attribute can be used to specify a prefix name to the custom tag, it will have page scope i.e. the specified prefix name is valid up to the present Jsp page.

Where uri attribute will take the name and location of the respective tld file.

2. TLD File:

The main purpose of TLD file is to provide the mapping between custom tag names and the respective TagHandler classes and it is able to manage the description of the custom tags attributes.

To provide the mapping between custom tag names and the respective TagHandler class we have to use the following tags.

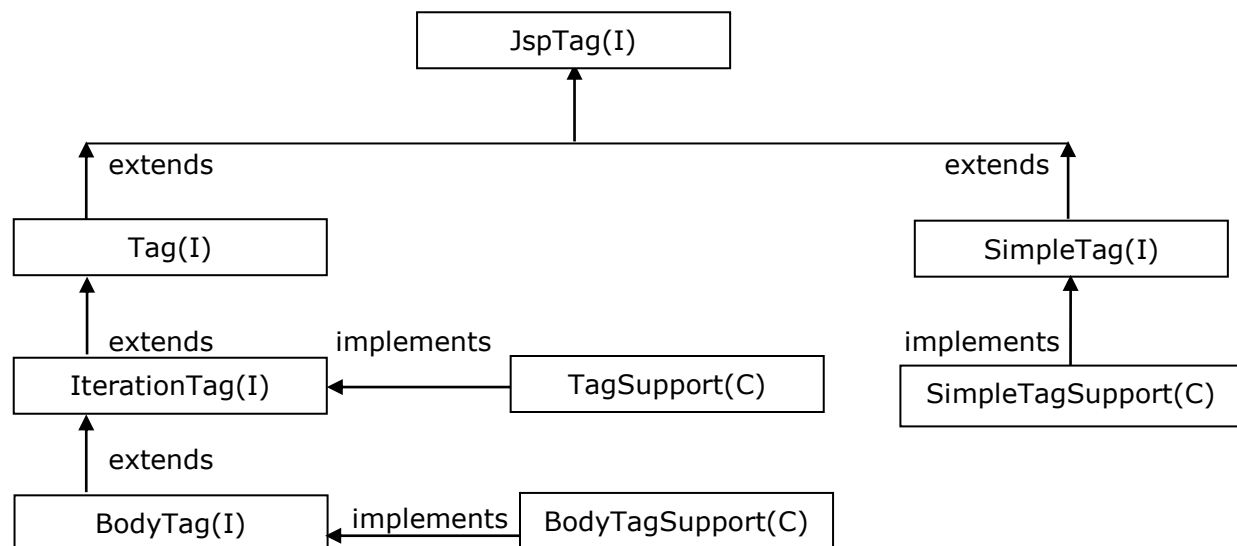
```
<taglib>
  <jsp-version>jsp version</jsp-version>
  <tlib-version>tld file version</tlib-version>
  <short-name>tld file short name</short-name>
  <description>description about tld file</description>
  <tag>
    <name>custom tag name</name>
    <tag-class>fully qualified name of TagHandler class</tag-class>
    <body-content>jsp or empty</body-content>
    <short-name>custom tag short name</short-name>
    <description>description about custom tags</description>
  </tag>
  -----
</taglib>
```

3. TagHandler class:

In custom tags preparation, the main purpose of TagHandler class is to define the basic functionality for the custom tags.

To design TagHandler classes in custom tags preparation Jsp API has provided some predefined library in the form of javax.servlet.jsp.tagext package (tagext-→tag extension).

javax.servlet.jsp.tagext package has provided the following library to design TagHandler classes.



The above Tag library was divided into 2 types.

1. Classic Tag library
2. Simple Tag library

As per the tag library provided by Jsp technology there are 2 types of custom tags.

1. Classic tags
2. Simple Tags

1. Classic Tags:

Classic Tags are the custom tags will be designed on the basis of Classic tag library provided by Jsp API.

As per the classic tag library provided by Jsp API there are 3 types of classic tags.

1. Simple Classic Tags
2. Iterator Tags
3. Body Tags

1. Simple Classic Tags:

Simple Classic Tags are the classic tags, which should not have body and attributes list.

To design simple classic tags the respective TagHandler class must implement Tag interface either directly or indirectly.

Predefined support:-

```
package javax.servlet.jsp.tagext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
public interface Tag extends JspTag
{
    public abstract void setPageContext(PageContext pagecontext);
    public abstract void setParent(Tag tag);
    public abstract Tag getParent();
    public abstract int doStartTag()throws JspException;
    public abstract int doEndTag()throws JspException;
    public abstract void release();
    public static final int SKIP_BODY = 0;
    public static final int EVAL_BODY_INCLUDE = 1;
    public static final int SKIP_PAGE = 5;
    public static final int EVAL_PAGE = 6;
}
```

Userdefined class:-

```
public class MyHandler extends Tag
{
    //write the code here
}
```

Where the purpose of `setPageContext(_)` method is to inject `pageContext` implicit object into the present `TagHandler` class.

Where the purpose of `setParent(_)` method is to inject parent tags `TagHandler` class object into the present `TagHandler` class.

Where the purpose of `getParent()` method is to return the parent tags `TagHandler` class object from the `TagHandler` class.

Where the purpose of `doStartTag()` method is to perform a particular action when container encounters the start tag of the custom tag.

After the custom tags start tag evaluating the custom tag body is completely depending on the return value provided by `doStartTag ()` method.

There are 2 possible return values from `doStartTag()` method.

1. `EVAL_BODY_INCLUDE`
2. `SKIP_BODY`

If `doStartTag()` method returns `EVAL_BODY_INCLUDE` constant then container will evaluate the custom tag body.

If `doStartTag()` method returns `SKIP_BODY` constant then container will skip the custom tag body and encounter end tag.

Where the purpose of `doEndTag()` method is to perform an action when container encounters end tag of the custom tag.

Evaluating the remaining the Jsp page after the custom tag or not is completely depending on the return value provided by `doEndTag()` method.

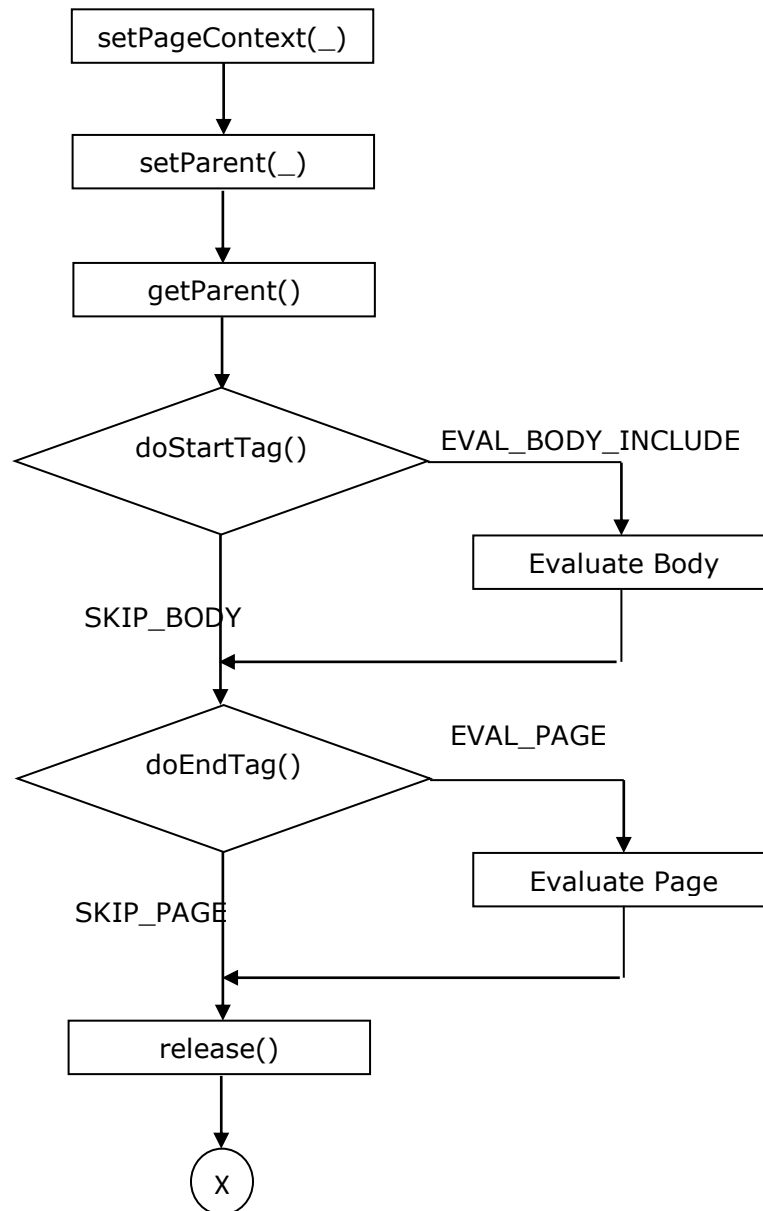
There are 2 possible return values from `doEndTag()` method.

1. `EVAL_PAGE`
2. `SKIP_PAGE`

If `doEndTag()` method returns `EVAL_PAGE` constant then container will evaluate the remaining Jsp page.

If `doEndTag()` method returns `SKIP_PAGE` constant then container will not evaluate the remaining Jsp page.

Where `release()` method can be used to perform `TagHandler` class deinstantiation.

Life Cycle of Tag interface:

Note: In Tag interface life cycle, container will execute `getParent()` method when the present custom tag is child tag to a particular parent tag otherwise container will not execute `getParent()` method.

Steps to design custom tag application:-

Step 1:- create the TagHandler class to provide the action

Step 2:- create the Tag Library Descriptor file(tld) file to configure custom tags.

Step 3:- write the jsp file to declare the custom tag configured in tld file.

Application:-**hello.jsp:-**

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
```

```
<mytags:hello>
```

```
    hi ratan how r u
```

```
</mytags:hello>
```

```
<br>
```

this is remaining page of jsp

hello.tld:-

```
<taglib>
```

```
    <jsp-version>2.1</jsp-version>
```

```
    <tlib-version>1.0</tlib-version>
```

```
        <tag>
```

```
            <name>hello</name>
```

```
            <tag-class>com.tcs.Customtag</tag-class>
```

```
            <body-content>jsp</body-content>
```

```
        </tag>
```

```
</taglib>
```

Customtag.java:-

```
package com.tcs;
```

```
import java.io.IOException;
```

```
import javax.servlet.jsp.JspException;
```

```
import javax.servlet.jsp.JspWriter;
```

```
import javax.servlet.jsp.PageContext;
```

```
import javax.servlet.jsp.tagext.Tag;
```

```
public class Customtag implements Tag{
```

```
    PageContext context;
```

```
    public int doEndTag() throws JspException {
```

```
        System.out.println("doEndTag method");
```

```
        return SKIP_PAGE;
```

```
    }
```

```
    public int doStartTag() throws JspException {
```

```
        JspWriter writer = context.getOut();
```

```
        try {
```

```
            writer.println("this is custom tag application by Ratan");
```

```
            writer.print("<br>");
```

```
        } catch (IOException e) {
```

```
            e.printStackTrace();
```

```
        }
```

```
        return EVAL_BODY_INCLUDE;
```

```
    }
```

```
    public Tag getParent() {
```

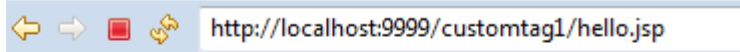
```
        System.out.println("getParent method");
```

```

        return null;
    }
    public void release() { }
    public void setPageContext(PageContext arg0) {
        this.context=arg0;
        System.out.println("page context method");
    }
    public void setParent(Tag arg0) {
        System.out.println("set parent method");
    }
}

```

The above example output:-



this is custom tag application by Ratan
hi ratan how r u

Server console check the life cycle methods:-

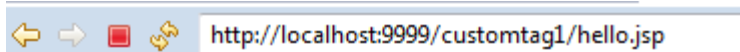
```

INFO: Server startup in 796 ms
page context method
set parent method
doEndTag method

```

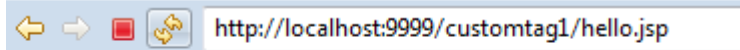
Observations :-

```
return EVAL_BODY_INCLUDE;           return SKIP_PAGE;
```



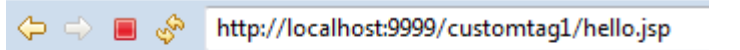
this is custom tag application by Ratan
hi ratan how r u

```
return SKIP_BODY;                   return SKIP_PAGE;
```



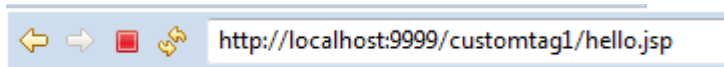
this is custom tag application by Ratan

```
return EVAL_BODY_INCLUDE;           return EVAL_PAGE;
```



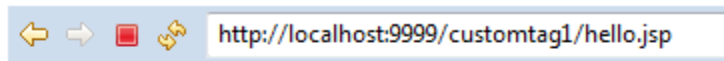
this is custom tag application by Ratan
hi ratan how r u
this is remaining page of jsp

```
return EVAL_BODY_INCLUDE;           return SKIP_PAGE;
```



this is custom tag application by Ratan
hi ratan how r u

return 5; **return 1;**



this is custom tag application by Ratan
hi ratan how r u

Note: To compile above code we need to set the classpath environment variable to the location of jsp-api.jar file.

Observations:

Case 1: In the above application, if we provide <body-content> type is empty in the tld file and if we provide body to the custom tag then container will raise an Exception like

org.apache.jasper.JasperException: /hello.jsp(2,0) According to TLD, tag mytags:hello must be empty, but is not

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello>
    hi ratan how r u
</mytags:hello>
```

hello.tld:-

```
<taglib>
    *****
    <body-content>empty</body-content>
    *****
</taglib>
```

Case 2: If we provide <body-content> value as jsp in tld file, if we provide body to custom tag in jsp page and if we return SKIP_BODY constant in the respective TagHandler class then container won't raise any Exception but container won't evaluate custom tag body.

Attributes in Custom Tags:-

In custom tag it is possible to provide multiple attributes.

If we want to provide attributes in custom tags then we have to perform the following steps.

Step 1: Define attribute in the custom tag.

```
<mytags:hello name="Ratan"/>
```

Step 2: Provide attributes description in the respective tld file.

To provide attributes description in tld file we have to use the following tags in tld file.

```
<taglib>
    *****
    <tag>
        *****
        <attribute>
            <name>attribute_name</name>
            <required>true/false</required>
            <rtexprvalue>true/false</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

Where <attribute> tag can be used to represent a single attribute in the tld file.

Where <name> tag will take attribute name.

Where <required> tag is a boolean tag, it can be used to specify whether the attribute is mandatory or optional.

Where <rtexprvalue> tag can be used to specify whether the attribute accept runtime values or not.

Step 3: Declare a property and setter method in TagHandler class with the same name of the attribute defined in custom tag.

```
public class MyHandler implements Tag {
    private String name;
    public void setName(String name) {
        this.name=name;
    }
    *****
}
```

2. Iterator Tags:

- Iterator tags are the custom tags, it will allow to evaluate custom tag body repeatedly.
- If we want to prepare iterator tags the respective TagHandler class must implement javax.servlet.jsp.tagext.IterationTag interface.

Predefined support:-

```
package javax.servlet.jsp.tagext;
import javax.servlet.jsp.JspException;
public interface IterationTag extends Tag
{
    public abstract int doAfterBody()throws JspException;
    public static final int EVAL_BODY_AGAIN = 2;
}
```

Userdefined class:-

```
class MyHandler implements IterationTag
{
    //write the body here
}
```

In general there are 2 possible return values from **doStartTag()** method.

1) EVAL_BODY_INCLUDE

If we return EVAL_BODY_INCLUDE constant from doStartTag() method then container will execute the custom tag body.

2) SKIP_BODY

If we return SKIP_BODY constant from doStartTag() method then container will skip the custom tag body.

Note: In case of iterator tags, we must return EVAL_BODY_INCLUDE from doStartTag() method.

After evaluating the custom tag body in case of iterator tags, container will access doAfterBody() method.

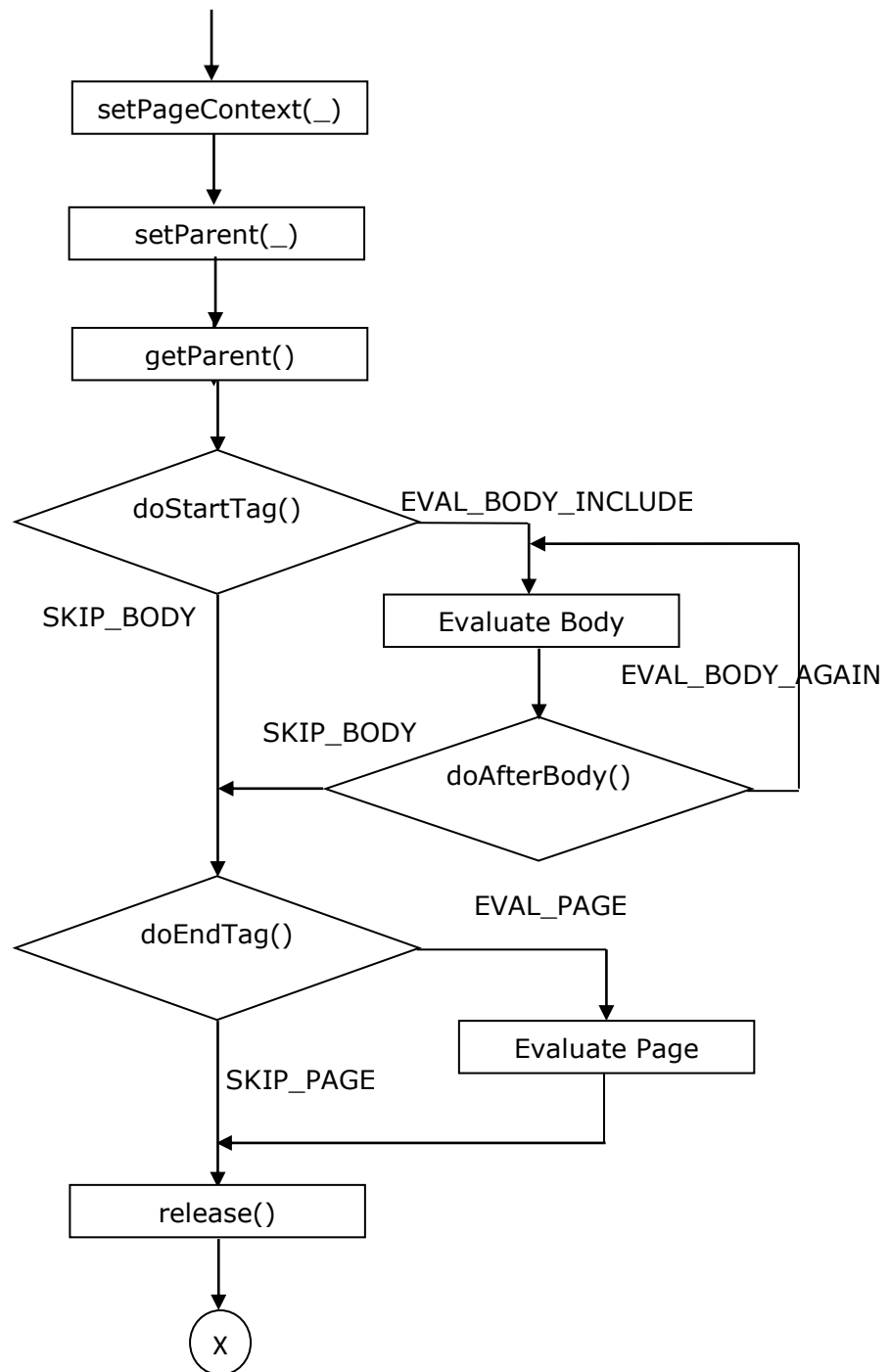
In the above context, evaluating the custom tag body again or not is completely depending on the return value which we are going to return from doAfterBody() method.

1) EVAL_BODY_AGAIN

If we return EVAL_BODY_AGAIN constant from doAfterBody() method then container will execute the custom tag body again.

2) SKIP_BODY

If we return SKIP_BODY constant from doAfterBody() method then container will skip out custom tag body evaluation and encounter end tag of the custom tag.

Life Cycle of IterationTag interface:

If we want to design custom tags by using above approach then the respective TagHandler class must implement Tag interface and IterationTag interface i.e. we must provide the implementation for all the methods which are declared in Tag and IterationTag interfaces in our TagHandler class.

This approach will increase burden to the developers and unnecessary methods in TgaHandler classes.

To overcome the above problem Jsp API has provided an alternative in the form of TagSupport class.

TagSupport is a concrete class, which was implemented Tag and IterationTag interfaces with the default implementation.

If we want to prepare custom tags with the TagSupport class then we have to take an user defined class, which must be a subclass to TagSupport class.

Predefined support:-

```
public interface TagSupport implements IterationTag {  
    public static final int EVAL_BODY_INCLUDE;  
    public static final int SKIP_BODY;  
    public static final int EVAL_PAGE;  
    public static final int SKIP_PAGE;  
    public static final int EVAL_BODY_AGAIN;  
    public PageContext pageContext;  
    public Tag t;  
    public void setPageContext(PageContext pageContext) {  
        this.pageContext=pageContext;  
    }  
    public void setParent(Tag t) {  
        this.t=t;  
    }  
    public Tag getParent() {  
        return t;  
    }  
    public int doStartTag()throws JspException {  
        return SKIP_BODY;  
    }  
    public int doAfterBody()throws JspException {  
        return SKIP_BODY;  
    }  
    public int doEndTag()throws JspException {  
        return EVAL_PAGE;  
    }  
    public void release() {}  
}
```

User defined class:-

```
public class MyHandler implements TagSupport  
{  
}
```

Application:-**hello.jsp:-**

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:iterate times="10"><br>
    Hi Ratan how r u
</mytags:iterate>
```

hello.tld:-

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>iterate</name>
        <tag-class>com.sravva.Iteration</tag-class>
        <body-content>jsp</body-content>
        <attribute>
            <name>times</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

Iteration.java:-

```
package com.sravva;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class Iteration extends TagSupport{
    private int times;
    private int count=1;
    public void setTimes(int times) {
        this.times = times;
    }
    @Override
    public int doAfterBody() throws JspException {
        if(count<times)
        {
            count++;
            return EVAL_BODY_AGAIN;
        }
        else
        {
            return SKIP_BODY;
        }
    }
    @Override
    public int doStartTag() throws JspException {
        return EVAL_BODY_INCLUDE;
    }
}
```

Application:-**Hello.jsp:-**

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
```

```
<mytags:loop start="1" end="20"><br>
```

```
    Hi Ratan how r u
```

```
</mytags:loop>
```

remaining page of the JSP

hello.tld:-

```
<taglib>
```

```
    <jsp-version>2.1</jsp-version>
```

```
    <tlib-version>1.0</tlib-version>
```

```
    <tag>
```

```
        <name>loop</name>
```

```
        <tag-class>com.sravva.Iteration</tag-class>
```

```
        <body-content>jsp</body-content>
```

```
        <attribute>
```

```
            <name>start</name>
```

```
            <required>true</required>
```

```
        </attribute>
```

```
        <attribute>
```

```
            <name>end</name>
```

```
            <required>true</required>
```

```
        </attribute>
```

```
    </tag>
```

```
</taglib>
```

Iteration.jsp:-

```
package com.sravva;
```

```
import javax.servlet.jsp.JspException;
```

```
import javax.servlet.jsp.tagext.TagSupport;
```

```
public class Iteration extends TagSupport{
```

```
    private int start;
```

```
    private int end;
```

```
    public void setStart(int start) {  
        this.start = start;    }
```

```
    public void setEnd(int end) {  
        this.end = end; }
```

```
    @Override
```

```
    public int doAfterBody() throws JspException {  
        if(end>start)  
        {  
            start++;  
            return EVAL_BODY_AGAIN;    }  
        else  
        {  
            return EVAL_PAGE;    }  
    }
```

```
    @Override
```

```
    public int doStartTag() throws JspException {  
        return EVAL_BODY_INCLUDE;    }
```

```
}
```

Nested Tags:-

- Defining a tag inside a tag is called as Nested Tag.
- In custom tags application, if we declare any nested tag then we have to provide a separate configuration in tld file and we have to prepare a separate TagHandler class under classes folder.

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:if condition='<%=10<20%>'>
    <mytags:true>condition is true</mytags:true> <br>
    <mytags:false>condition is false</mytags:false><br>
</mytags:if>
rest of the JSP code
```

hello.tld:-

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>if</name>
        <tag-class>com.sravva.If</tag-class>
        <body-content>jsp</body-content>
        <attribute>
            <name>condition</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
    <tag>
        <name>>true</name>
        <tag-class>com.sravva.True</tag-class>
        <body-content>jsp</body-content>
    </tag>
    <tag>
        <name>false</name>
        <tag-class>com.sravva.False</tag-class>
        <body-content>jsp</body-content>
    </tag>
</taglib>
```

If.java:-

```
package com.sravva;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class If extends TagSupport{
    private boolean condition;
    public void setCondition(boolean condition) {
        this.condition = condition;
    }
    public boolean getCondition()
```

```
        {
            return condition;
        }
        @Override
        public int doStartTag() throws JspException {
            return EVAL_BODY_INCLUDE;
        }
        @Override
        public int doEndTag() throws JspException {
            return EVAL_PAGE;
        }
    }
}
```

True.java:-

```
package com.sravva;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class True extends TagSupport {
    @Override
    public int doStartTag() throws JspException {
        If f = (If)getParent();
        boolean b = f.getCondition();
        if(b==true)
        {
            return EVAL_BODY_INCLUDE;
        }
        else {
            return SKIP_BODY;
        }
    }
}
```

False.java:-

```
package com.sravva;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
public class False extends TagSupport {
    @Override
    public int doStartTag() throws JspException {
        If f = (If)getParent();
        boolean b = f.getCondition();
        if(b==true)
        {
            return SKIP_BODY;
        }
        else
        {
            return EVAL_BODY_INCLUDE;
        }
    }
}
```

empdetails.jsp:

```
<%@taglib uri="/WEB-INF/emp.tld" prefix="emp"%>
<emp:empDetails/>
```

emp.tld:

```
<taglib>
  <jsp-version>2.1</jsp-version>
  <tlib-version>1.0</tlib-version>
  <tag>
    <name>empDetails</name>
    <tag-class>com.dss.EmpDetails</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

EmpDetails.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.sql.*;
public class EmpDetails extends TagSupport
{
    Connection con;
    Statement st;
    ResultSet rs;
    public EmpDetails()
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
            st=con.createStatement();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    public int doStartTag() throws JspException
    {
        try
        {
            JspWriter out=pageContext.getOut();
            rs=st.executeQuery("select * from emp");
            ResultSetMetaData rsmd=rs.getMetaData();
            int count=rsmd.getColumnCount();
            out.println("<html><body bgcolor='pink'>");
        }
    }
}
```

```
        out.println("<center><br><br>");
        out.println("<table border='1' bgcolor='lightyellow'>");
        out.println("<tr>");
        for (int i=1;i<=count;i++)
        {
            out.println("<td><b><font size='6'
color='red'><center>" + rsmd.getColumnNames(i) + "</center></font></b></td>");
        }
        out.println("</tr>");
        while (rs.next())
        {
            out.println("<tr>");
            for (int i=1;i<=count;i++)
            {
                out.println("<td><b><font
size='5'>" + rs.getString(i) + "</font></b></td>");
            }
            out.println("</tr>");
        }
        out.println("</table></center></body></html>");
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    return SKIP_BODY;
}
```


3. Body Tags:

- ❖ Up to now in our custom tags (simple classic tags, iteration tags) we prepared custom tags with the body, where we did not perform updations over the custom tag body, just we scanned custom tag body and displayed on the client browser.
- ❖ If we want to perform updations over the custom tag body then we have to use Body Tags.
If we want to design body tags in Jsp technology then the respective TagHandler class must implement BodyTag interface either directly or indirectly.

BodyTag:-

```
package javax.servlet.jsp.tagext;
```

```
import javax.servlet.jsp.JspException;
```

public interface BodyTag extends IterationTag

```
{ public abstract void setBodyContent(BodyContent bodycontent);  
  public abstract void doInitBody()throws JspException;  
  public static final int EVAL_BODY_TAG = 2;  
  public static final int EVAL_BODY_BUFFERED = 2; }
```

All methods at inherited level:-

public interface BodyTag extends IterationTag

```
{ public static final int EVAL_BODY_INCLUDE;  
  public static final int SKIP_BODY;  
  public static final int EVAL_PAGE;  
  public static final int SKIP_PAGE;  
  public static final int EVAL_BODY_AGAIN;  
  public static final int EVAL_BODY_BUFFERED;  
  public void setPageContext(PageContext pageContext);  
  public void setParent(Tag t);  
  public Tag getParent();  
  public int doStartTag()throws JspException;  
  public void doInitBody()throws JspException;  
  public void setBodyContent(BodyContent bodyContent);  
  public int doAfterBody()throws JspException;  
  public int doEndTag()throws JspException;  
  public void release(); }
```

User defined class:-

```
public class MyHandler implements BodyTag  
{ //write the logics here; }
```

In case of body tags, there are 3 possible return values from doStartTag() method.

a) EVAL_BODY_INCLUDE

If we return EVAL_BODY_INCLUDE constant from doStartTag() method then container will evaluate the custom tag body i.e. display as it is the custom tag body on client browser.

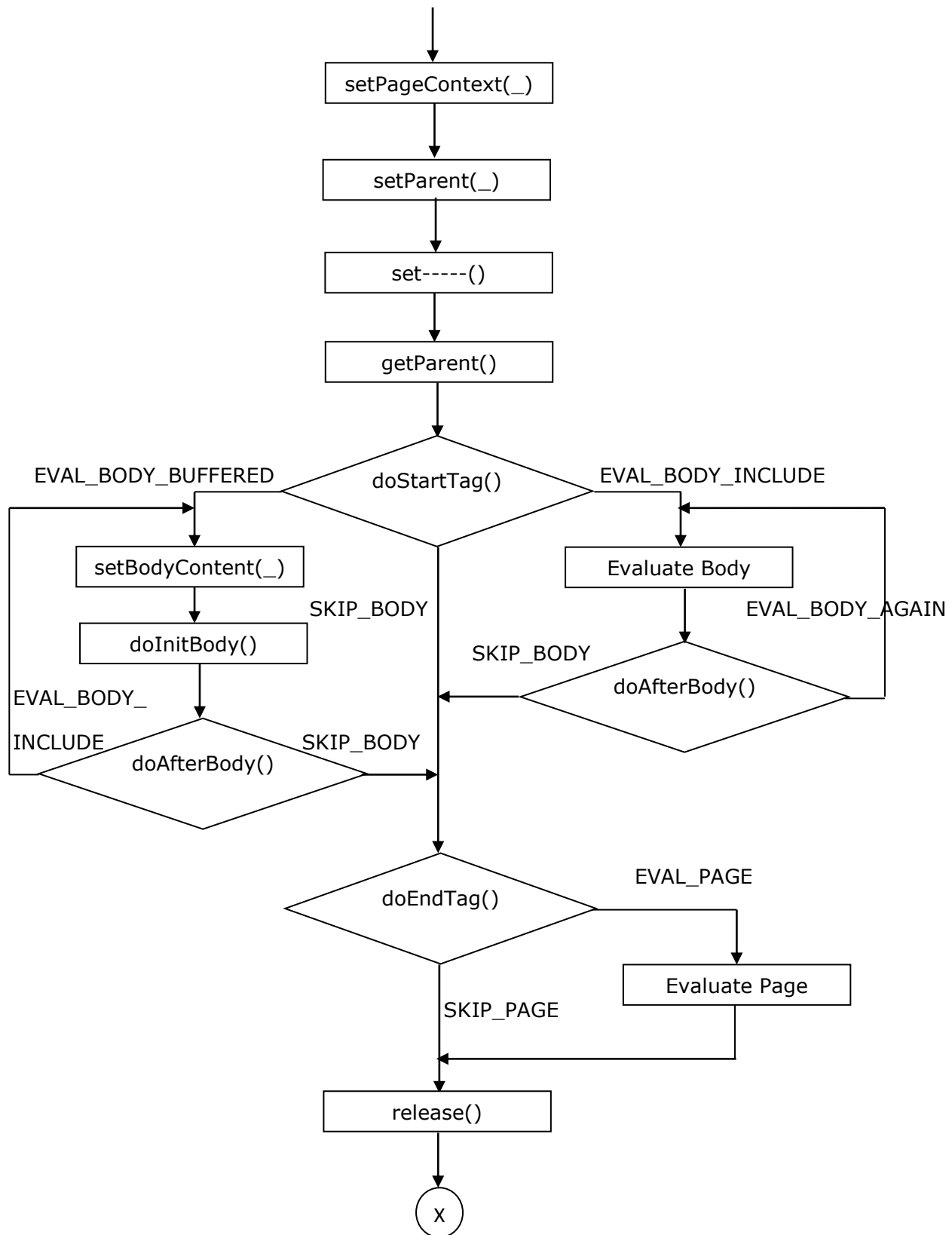
b) SKIP_BODY

If we return SKIP_BODY constant from doStartTag() method then container will skip the custom tag body.

c) EVAL_BODY_BUFFERED

If we return EVAL_BODY_BUFFERED constant from doStartTag() method then container will store custom tag body in a buffer then access setBodyContent(_) method.

To access setBodyContent(_) method container will prepare BodyContent object with the buffer. After executing setBodyContent(_) method container will access doInitBody() method in order to prepare BodyContent object for allow modifications.

Life Cycle of BodyTag interface:

- ❖ To prepare custom tags if we use above approach then the respective TagHandler class must implement all the methods declared in BodyTag interface irrespective of the application requirement.

This approach may increase burden to the developers and unnecessary methods in the custom tag application.

- ❖ To overcome this problem we will use an alternative provided by Jsp technology i.e. ***javax.servlet.jsp.tagext.BodyTagSupport*** class.

BodyTagSupport class is a concrete class, a direct implementation class to BodyTag interface and it has provided the default implementation for all the methods declared in BodyTag interface.

If we want to prepare custom tags with BodyTagSupport class then the respective TagHandler class must extend BodyTagSupport and overrides the only required methods.

```
public class BodyTagSupport implements BodyTag {  
    public static final int EVAL_BODY_INCLUDE;  
    public static final int SKIP_BODY;  
    public static final int EVAL_PAGE;  
    public static final int SKIP_PAGE;  
    public static final int EVAL_BODY_AGAIN;  
    public static final int EVAL_BODY_BUFFERED;  
    public PageContext pageContext;  
    public Tag t;  
    public BodyContent bodyContent;  
    public void setPageContext(PageContext pageContext) {  
        this.pageContext=pageContext;  
    }  
    public void setParent(Tag t) {  
        this.t=t;  
    }  
    public Tag getParent() {  
        return t;  
    }  
    public int doStartTag()throws JspException {  
        return EVAL_BODY_BUFFERED;  
    }  
    public void setBodyContent(BodyContent bodyContent) {  
        this.bodyContent=bodyContent;  
    }  
    public void doInitBody()throws JspException  
    { }  
    public int doAfterBody()throws JspException {  
        return SKIP_BODY;  
    }  
    public int doEndTag()throws JspException {  
        return EVAL_PAGE;    }  
    public void release() { }  
}
```

- ❖ In case of body tags, custom tag body will be available in BodyContent object, to get custom tag body from BodyContent object we have to use the following method.

public String getString()

- ❖ To send modified data to the response object we have to get JspWriter object from BodyContent, for this we have to use the following method.

public JspWriter getEnclosingWriter()

Application:-

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:reverse>
    Sravya Infotech
</mytags:reverse>
```

hello.tld:-

```
<taglib>
  <jsp-version>2.1</jsp-version>
  <tlib-version>1.0</tlib-version>
  <tag>
    <name>reverse</name>
    <tag-class>com.sravya.Reverse</tag-class>
    <body-content>jsp</body-content>
  </tag>
</taglib>
```

Reverse.java:-

```
package com.sravya;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyTagSupport;
public class Reverse extends BodyTagSupport{
    public int doEndTag()throws JspException
    {
        try
        {
            String data=bodyContent.getString();
            StringBuffer sb=new StringBuffer(data);
            StringBuffer rdata=sb.reverse();
            bodyContent.getEnclosingWriter().println(rdata);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return EVAL_PAGE;
    }
}
```

-----Application11-----

custapp6:**editor.html:**

```
<html>

    <body bgcolor="lightgreen">
        <b><font size="6" color="red">
            <form action="/result.jsp">
                <pre>
                    Enter SQL Query
                    <textarea name="query" rows="5" cols="50"></textarea>
                    <input type="submit" value="GetResult"/>
                </pre>
            </form></font></b></body>
</html>
```

result.jsp:

```
<%@taglib uri="/WEB-INF/result.tld" prefix="dbtags"%>
<jsp:declaration>
    String query;
</jsp:declaration>
<jsp:scriptlet>
    query=request.getParameter("query");
</jsp:scriptlet>
<dbtags:query>
    <jsp:expression>query</jsp:expression>
</dbtags:query>
```

result.tld:

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>query</name>
        <tag-class>com.dss.Result</tag-class>
        <body-content>jsp</body-content>
    </tag>
</taglib>
```

Result.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.sql.*;
public class Result extends BodyTagSupport
{
```

```
Connection con;
Statement st;
ResultSet rs;
public Result()
{
    try
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");

        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","
ratan");
        st=con.createStatement();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
public int doEndTag() throws JspException
{
    try
    {
        JspWriter out=pageContext.getOut();
        String query=bodyContent.getString();
        boolean b=st.execute(query);
        if (b==true)
        {
            rs=st.getResultSet();
            ResultSetMetaData rsmd=rs.getMetaData();
            out.println("<html>");
            out.println("<body bgcolor='lightblue'>");
            out.println("<center><br><br>");
            out.println("<table border='1' bgcolor='lightyellow'>");
            int count=rsmd.getColumnCount();
            out.println("<tr>");
            for (int i=1;i<=count;i++)
            {
                out.println("<td><center><b><font size='6'
color='red'>" +rsmd.getColumnName(i)+"</font></b></center></td>");
            }
            out.println("</tr>");
            while (rs.next())
            {
                out.println("<tr>");
                for (int i=1;i<=count;i++)
                {
                    out.println("<td><h1>" +rs.getString(i)+"</h1></td>");
                }
                out.println("</tr>");
            }
            out.println("</table></center></body></html>");
        }
    }
}
```

```
        else
        {
            int rowCount=st.getUpdateCount();
            out.println("<html>");
            out.println("<body bgcolor='lightyellow'>");
            out.println("<center><b><font size='7' color='red'>");
            out.println("<br><br>");
            out.println("Record Updated : "+rowCount);

            out.println("</font></b></center></body></html>");
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    return EVAL_PAGE;
}
```

2. Simple Tags:

Classic tags Vs Simple tags:-

- ❖ Classic tags are more API independent, but Simple tags are less API independent.
- ❖ If we want to design custom tags by using classic tag library then we have to remember 3 types of life cycles.
If we want to design custom tags by using simple tag library then we have to remember only 1 type of life cycle.
- ❖ In case of classic tag library, all the TagHandler class objects are cacheable objects, but in case of simple tag library, all the TagHandler class objects are non-cacheable objects.
- ❖ In case of classic tags, all the custom tags are not body tags by default, but in case of simple tags, all the custom tags are having body tags capacity by default.

If we want to design custom tags by using simple tag library then the respective TagHandler class must implement SimpleTag interface either directly or indirectly.

Predefine support:-

```
package javax.servlet.jsp.tagext;
import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
public interface SimpleTag extends JspTag
{ public abstract void doTag()throws JspException, IOException;
  public abstract void setParent(JspTag jsptag);
  public abstract JspTag getParent();
  public abstract void setJspContext(JspContext jspcontext);
  public abstract void setJspBody(JspFragment jspfragment);
}
```

User defined class:-

```
public class MyHandler implements SimpleTag
{ //write the logics here
}
```

Where jspContext is an implicit object available in simple tag library, it is same as pageContext, it can be used to make available all the Jsp implicit objects.

Where jspFragment is like bodyContent in classic tag library, it will accommodate custom tag body directly.

Where setJspContext(_) method can be used to inject JspContext object into the present web application.

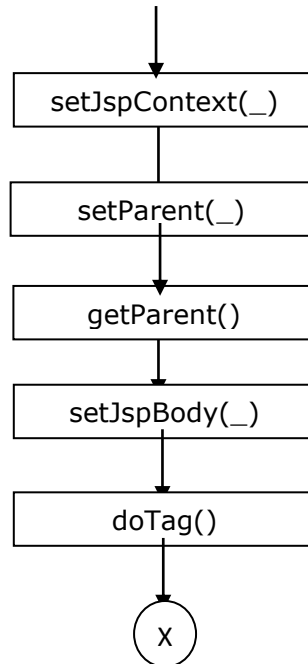
Where setParent(_) method can be used to inject parent tags TagHandler class object reference into the present TagHandler class.

Where getParent() method can be used to get parent tags TagHandler class object.

Where setJspBody(_) method is almost all equal to setBodyContent(_) method in order to accommodate custom tag body.

Where doTag() method is equivalent to doStartTag() method and doEndTag() method in order to perform an action.

Life Cycle of SimpleTag interface:



- ❖ To design custom tags if we use approach then the respective TagHandler class must implement SimpleTag interface i.e. it must implement all the methods which are declared in SimpleTag interface.
- ❖ This approach will increase burden to the developers and unnecessary methods in the custom tags. To overcome the above problem Jsp technology has provided an alternative in the form of javax.servlet.jsp.tagext.SimpleTagSupport class.
- ❖ SimpleTagSupport is a concrete class provided by Jsp technology as an implementation class to SimpleTag interface with default implementation.
- ❖ If we want to design custom tags by using SimpleTagSupport class then the respective TagHandler class must be extended from SimpleTagSupport class and where we have to override the required methods.

Predefined support:-

```
public interface SimpleTag extends JspTag {  
    private JspContext jspContext;  
    private JspFragment jspFragment;  
    private JspTag jspTag;  
    public void setJspContext(JspContext jspContext) {  
        this.jspContext=jspContext;  
    }  
    public void setParent(JspTag t) {  
        this.jspTag=jspTag;  
    }  
    public void setJspBody(JspFragment jspFragment) {  
        this.jspFragment=jspFragment;  
    }  
    public JspTag getParent() {
```

```
        return jspTag;
    }
    public JspFragment getJspBody() {
        return jspFragment;
    }
    public JspContext getJspContext() {
        return jspContext;
    }
    public void doTag()throws JspException, IOException
    { }
```

User defined class:-

```
public class MyHandler extends SimpleTagSupport
{    //write the logics here
}
```

Application:**hello.jsp:-**

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello/>
```

hello.tld:-

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>hello</name>
        <tag-class>com.sravva.Hello</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

Hello.java:-

```
package com.sravva;
import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class Hello extends SimpleTagSupport{
    @Override
    public void doTag() throws JspException, IOException {
        JspContext context = getJspContext();
        JspWriter writer = context.getOut();
        writer.println("this is Sample Tag application");
    }
}
```

Accessing Tag Body:-

To print custom tag body use below code.

Application:-**hello.jsp:-**

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello>
    hi this is body of custom tag<br>
</mytags:hello>
```

hello.tld:-

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>hello</name>
        <tag-class>com.sravva.Hello</tag-class>
        <body-content>scriptless</body-content>
    </tag>
</taglib>
```

Hello.java:-

```
package com.sravva;
import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
public class Hello extends SimpleTagSupport{
    @Override
    public void doTag() throws JspException, IOException {
        JspWriter writer = getJspContext().getOut();
        StringWriter stringWriter = new StringWriter();
        getJspBody().invoke(stringWriter);
        writer.println("this is Sample Tag application");
        writer.println(stringWriter.toString());
    }
}
```

Attributes in custom tags:-

To provide the attributes in custom tags use below code.

Application:-**hello.jsp:-**

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello msg="ratansoft">
    hi this is body of custom tag<br>
</mytags:hello>
```

hello.tld:-

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>hello</name>
        <tag-class>com.sravva.Hello</tag-class>
        <body-content>scriptless</body-content>
        <attribute>
            <name>msg</name>
        </attribute>
    </tag>
</taglib>
```

Hello.java:-

```
package com.sravva;

import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class Hello extends SimpleTagSupport{
    private String msg;
    public void setMsg(String msg) {
        this.msg = msg;
    }
    @Override
    public void doTag() throws JspException, IOException {
        JspWriter writer = getJspContext().getOut();

        StringWriter stringWriter = new StringWriter();
        getJspBody().invoke(stringWriter);
        writer.println("this is Sample Tag application");
        writer.println(stringWriter.toString());
        writer.println("<br>");
        writer.println(msg);
    }
}
```

Custom URI in jsp custom tags:-

- We can use custom uri in jsp custom tags by specifying **tld** file location in web.xml file.
- The web container will get the information about tld file from web.xml file.

Application:-**hello.jsp:-**

```
<%@taglib uri="mytags" prefix="xxx"%>
<xxx:hello>
    Hi Ratan how r u <br>
</xxx:hello>
this is rest of the JSP Application
```

web.xml:-

```
<web-app>
    <jsp-config>
        <taglib>
            <taglib-uri>mytags</taglib-uri>
            <taglib-location>/WEB-INF/hello.tld</taglib-location>
        </taglib>
    </jsp-config>
</web-app>
```

hello.tld:-

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <uri>mytags</uri>
    <tag>
        <name>hello</name>
        <tag-class>com.sravva.TaglibUri</tag-class>
        <body-content>jsp</body-content>
    </tag>
</taglib>
```

TaglibUri.java:-

```
package com.sravva;
import java.io.IOException;
import java.util.Date;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;
public class TaglibUri extends TagSupport{
    @Override
    public int doStartTag() throws JspException {
        JspWriter writer = pageContext.getOut();
        try {
            Date d = new Date();
            writer.println("this is custom tag application URI attribute");
            writer.println("<br/>");
            writer.println("today date is="+d);
        }
    }
}
```

```
        writer.println("<br/>");
    } catch (IOException e) {
        e.printStackTrace();
    }
    return EVAL_BODY_INCLUDE;
}
@Override
public int doEndTag() throws JspException {
    return EVAL_PAGE;
}
}
```

Jsp Standard Tag Library(JSTL):

In Jsp technology, by using scripting elements we are able to provide Java code inside the Jsp pages.

To preserve Jsp principles we have to eliminate scripting elements, for this we have to use Jsp Actions.

In case of Jsp Actions, we will use standard actions as an alternative to scripting elements, but which are limited in number and having bounded functionality so that standard actions are not specified the required application format.

Still it is required to provide Java code inside the Jsp pages.

In the above context, to eliminate Java code completely from Jsp pages we have to use custom actions.

In case of custom actions, to implement simple Java syntaxes like if condition, for loop and so on we have to provide a lot of Java code internally.

To overcome the above problem Jsp technology has provided a separate tag library with simple Java syntaxes implementation and frequently used operations.

JSTL is an abstraction provided by Sun Microsystems, but where implementations are provided by all the server vendors.

With the above convention Apache Tomcat has provided JSTL implementation in the form of the jar files as standard.jar, jstl.jar.

Apache Tomcat has provided the above 2 jar files in the following location.

C:\Tomcat7.0\webapps\examples\WEB_INF\lib

If we want to get JSTL support in our Jsp pages then we have to keep the above 2 jar files in our web application lib folder.

JSTL has provided the complete tag library in the form of the following 5 types of tags.

1. Core Tags
2. XML Tags

3. Internationalization or I18N Tags (Formatted tags)
4. SQL Tags
5. Functions tags

To get a particular tag library support into the present Jsp page we have to use the following standard URL's to the attribute in the respective taglib directive.

`http://java.sun.com/jstl/core`

`http://java.sun.com/jstl/xml`

`http://java.sun.com/jstl/fmt`

`http://java.sun.com/jstl/sql`

`http://java.sun.com/jsp/jstl/functions`

1. Core Tags:

In JSTL, core tag library was divided into the following 4 types.

1. General Purpose Tags
 1. `<c:set----->`
 2. `<c:remove----->`
 3. `<c:catch----->`
 4. `<c:out----->`
2. Conditional Tags
 1. `<c:if----->`
 2. `<c:choose----->`
 3. `<c:when----->`
 4. `<c:otherwise----->`
3. Iterate Tags
 1. `<c:forEach----->`
 2. `<c:forTokens----->`
4. Url-Based Tags
 1. `<c:import----->`
 2. `<c:url----->`
 3. `<c:redirect----->`

1. General Purpose Tags:

1. `<c:set----->`:

This tag can be used to declare a single name value pair onto the specified scope.

Syntax: `<c:set var="--" value="--" scope="--"/>`

Where var attribute will take a variable i.e. key in key-value pair.

Where value attribute will take a particular value i.e. a value in key-value pair.

Where scope attribute will take a particular scope to include the specified key-value pair.

2. `<c:out----->`:

This tag can be used to display a particular value on the client browser.

Syntax: <c:out value="--"/>

Where value attribute will take a static data or an expression.

To present an expression with value attribute we have to use the following format.

Syntax: \${expression}

Ex: <c:out value"\${a}"/>

If the container encounters above tag then container will search for 'a' attribute in the page scope, request scope, session scope and application scope.

-----Application13-----

jstlapp1:

core.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <c:set var="a" value="AAA" scope="request"/>
            <br>
            <c:out value="core tag library"/>
            <br><br>
            <c:out value="${a}"/>
            </font></b></center>
    </body>
</html>
```

3. <c:remove----->:

This tag can be used to remove an attribute from the specified scope.

Syntax: <c:remove var="--" scope="--"/>

Where scope attribute is optional, if we have not specified scope attribute then container will search for the respective attribute in the page scope, request scope, session scope and application scope.

-----Application14-----

core1.jsp:


```

<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <c:set var="a" value="AAA" scope="request"/>
            <br>
            a-----> <c:out value="{a}"/>
            <br><br>
            <c:remove var="a" scope="request"/>
            a-----> <c:out value="{a}"/>
            </font></b></center>
        </body>
    </html>

```

4. <c:catch----->:

This tag can be used to catch an Exception raised in its body.

Syntax: <c:catch var="--">
 -----</c:catch>

Where var attribute will take a variable to hold the generated Exception object reference.

-----Application15-----

core2.jsp:

```

<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <c:catch var="e">
                <jsp:scriptlet>
                    java.util.Date d=null;
                    out.println(d.toString());
                </jsp:scriptlet>
            </c:catch>
            <c:out value="{e}"/>
            </font></b></center>
        </body>
    </html>

```

2. Conditional Tags:

1. <c:if----->: This tag can be used to implement if conditional statement.

Syntax: <c:if test="--"/> Where test attribute is a boolean attribute, it may take either true or false values.

-----Application16-----

core3.jsp:

```

<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <c:set var="a" value="10"/>
            <c:set var="b" value="20"/>
            <c:if test="{a<b}">
                condition is true
            </c:if>
            <br>
            out of if
        </font></b></center>
    </body>
</html>

```

2. <c:choose----->, <c:when-----> and <c:otherwise----->:

These tags can be used to implement switch programming construct.

Syntax: <c:choose>
 <c:when test="--">

 </c:when>

 <c:otherwise>

 </c:otherwise>
 </c:choose>

-----Application17-----

core4.jsp:

```

<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <c:set var="a" value="10"/>
            <c:choose>
                <c:when test="{a==10}">
                    TEN
                </c:when>
                <c:when test="{a==15}">
                    FIFTEEN
                </c:when>
                <c:when test="{a==20}">
                    TWENTY
                </c:when>
            </c:choose>
        </font></b></center>
    </body>
</html>

```

```

                <c:otherwise>
                    Number is not in 10,15 and 20
                </c:otherwise>
            </c:choose>
        </font></b></center>
    </body>
</html>

```

3. Iterator Tags:

1. <c:forEach----->:

This tag can be used to implement for loop to provide iterations on its body and it can be used to perform iterations over an array of elements or Collection of elements.

Syntax 1: <c:forEach var="--" begin="--" end="--" step="--">

</c:forEach>

Where var attribute will take a variable to hold up the loop index value at each and every iteration.

Where begin and end attribute will take start index value and end index value.

Syntax 2: <c:forEach var="--" items="--">

</c:forEach>

Where var attribute will take a variable to hold up an element from the respective Collection at each and every iteration.

Where items attribute will take the reference of an array or Collection from either of the scopes page, request, session and application.

-----Application18-----

core5.jsp:

```

<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <c:forEach var="a" begin="0" end="10" step="2">
                <c:out value="{a}"/><br>
            </c:forEach>
        </b>
    </center>
    </body>
</html>

```

```

String[] s={"A","B","C"};
request.setAttribute("s",s);
%>
<br>
<c:forEach var="x" items="{s}">
  <c:out value="{x}"/><br>
</c:forEach>
</font></b></center>
<body>
<html>

```

2. <c:forTokens----->:

This tag can be used to perform String Tokenization on a particular String.

Syntax : <c:forTokens var="--" items="--" delims="--">

</c:forTokens>

Where var attribute will take a variable to hold up token at each and every iteration.

Where items attribute will take a String to tokenize.

Where delims attribute will take a delimiter to perform tokenization.

-----Application19-----

core6.jsp:

```

<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
  <body>
    <center><b><font size="7">
      <c:forTokens var="token" items="Ratan Software Solutions" delims="
">
        <c:out value="{token}"/><br>
      </c:forTokens>
    </font></b></center>
  <body>
<html>

```

4. Url-Based Tags:

1. <c:import----->:

This tag can be used to import the content of the specified target resource into the present Jsp page.

-----Application20-----

second.jsp:

```
<center><h1>This is second.jsp</h1></center>
```

core7.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
  <body>
    <center><b><font size="7">
      Start
      <br>
      <c:import url="second.jsp"/>
      <br>
      End
    </font></b></center>
  </body>
</html>
```

2. <c:url----->:

This tag can be used to represent the specified url.

Syntax : <c:url value="--"/>

-----Application21-----

core8.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
  <body>
    <center><b><font size="7">
      <a href='<c:url value="http://localhost:2020/loginapp"/>'>Login
Application</a>
    </font></b></center>
  </body>
</html>
```

2. <c:redirect----->:

This tag can be used to implement Send Redirect Mechanism from a particular Jsp page.

Syntax: <c:redirect url="--"/>

-----Application22-----

core9.jsp:

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

<html>
    <body>
        <center><b><font size="7">
            <c:redirect url="http://localhost:2020/registrationapp"/>
        </font></b></center>
    </body>
</html>
```

4. SQL Tags:

The main purpose of SQL tag library is to interact with the database in order to perform the basic database operations.

JSTL has provided the following set of tags as part of SQL tag library.

1. <sql:setDataSource----->
2. <sql:update----->
3. <sql:query----->
4. <sql:transaction----->
5. <sql:param----->
6. <sql:dateParam----->

1. <sql:setDataSource----->:

This tag can be used to prepare the Jdbc environment like Driver loading, establish the connection.

Syntax: <sql:setDataSource driver="--" url="--" user="--" password="--"/>

Where driver attribute will take the respective Driver class name.

Where url attribute will take the respective Driver url.

Where user and password attributes will take database user name and password.

2. <sql:update----->:

This tag can be used to execute all the updation group SQL queries like create, insert, update, delete, drop and alter.

Syntax 1: <sql:update var="--" sql="--"/>

Syntax 2: <sql:update var="--"> ----- query ----- </sql:update>

In the above <sql:update> tag we are able to provide the SQL queries in 2 ways.

1. Statement style SQL queries, which should not have positional parameters.
2. PreparedStatement style SQL queries, which should have positional parameters.

If we use PreparedStatement style SQL queries then we have to provide values to the positional parameters, for this we have to use the following SQL tags.

1. <sql:param----->:

This tag can be used to set a normal value to the positional parameter.

Syntax 1: <sql:param value="--"/>

Syntax 2: <sql:param> ----- value ----- </sql:param>

2. <sql:dateParam----->:

This tag can be used to set a value to parameter representing data.

Syntax 1: <sql:dateParam value="--"/>

Syntax 2: <sql:dateParam>value</sql:dateParam>

-----Application23-----

jstlapp2:

sql.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
            <sql:update var="result" sql="create table emp1(eid number, ename
varchar2(10), esal number)"/>
            Row Count ..... <c:out value="${result}"/>
        </font></b></center>
    </body>
</html>
```

-----Application24-----

sql1.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
            <sql:update var="result" sql="insert into emp1
values(101,'aaa',5000)"/>

            Row Count ..... <c:out value="${result}"/>
        </font></b></center>
    </body>
</html>
```

-----Application25-----

sql2.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
            <sql:update var="result" sql="insert into emp1 values(?,?,?)">
                <sql:param value="103"/>
                <sql:param>ccc</sql:param>
                <sql:param value="7000"/>
            </sql:update>
            Row Count ..... <c:out value="${result}"/>
        </font></b></center>
    </body>
</html>
```

-----Application26-----

sql3.jsp:

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



```

<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
            <sql:update var="result">
                update emp1 set esal=esal+? where esal>?
                <sql:param>500</sql:param>
                <sql:param>5000</sql:param>
            </sql:update>
            Row Count ..... <c:out value="${result}"/>

        </font></b></center></body>
</html>

```

-----Application27-----

sql4.jsp:

```

<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
            <sql:update var="result">
                delete emp where esal>1000
            </sql:update>
            Row Count ..... <c:out value="${result}"/>
        </font></b></center>
    </body>
</html>

```

3. <sql:query----->:

This tag can be used to execute selection group SQL queries in order to fetch the data from database table.

Syntax 1: <sql:query var="--" sql="--"/>

Syntax 2: <sql:query var="--"> ----- query ----- </sql:query>

If we execute selection group SQL queries by using <sql:query> tag then SQL tag library will prepare result object to hold up fetched data.

In SQL tag library, result object is a combination of ResultSet object and ResultSetMetaData object.

In result object, all the column names will be represented in the form a single dimensional array referred by columnNames predefined variable and column data (table body) will be represented in the form of 2-dimensionnal array referred by rowsByIndex predefined variable.

-----Application28-----

sql5.jsp:

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

<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
            <sql:query var="result" sql="select * from emp"/>
            <table border="1" bgcolor="lightyellow">
                <tr>
                    <c:forEach var="columnName"
items="${result.columnNames}">
                        <td><center><b><font size="6" color="red">
                            <c:out value="${columnName}"/>
                        </font></b></center></td>
                    </c:forEach>
                </tr>
                <c:forEach var="row" items="${result.rowsByIndex}">
                    <tr>
                        <c:forEach var="column" items="${row}">
                            <td><b><font size="5">
                                <c:out value="${column}"/>
                            </font></b></td>
                        </c:forEach>
                    </tr>
                </c:forEach>
            </table>
        </font></b></center>
    </body>
</html>
```

4. <sql:transaction----->:

This tag will represent a transaction, which includes collection of <sql:update> tags and <sql:query> tags.

3. I18N Tags(Formatted Tags):

1. <fmt:setLocale----->:

This tag can be used to represent a particular Locale.

Syntax: <fmt:setLocale value="--"/>

Where value attribute will take Locale parameters like en_US, it_IT and so on.

2. <fmt:formatNumber----->:

This tag can be used to represent a number w.r.t the specified Locale.

Syntax: <fmt:formatNumber var="--" value="--"/>

Where var attribute will take a variable to hold up the formatted number.

Where value attribute will take a number.

3. <fmt:formatDate----->:

This tag can be used to format present system date w.r.t. a particular Locale.

Syntax: <fmt:formatDate var="--" value="--"/>

Where var attribute will take a variable to hold up the formatted date.

Where value attribute will take the reference of Date object.

4. <fmt:setBundle----->:

This tag can be used to prepare ResourceBundle object on the basis of a particular properties file.

Syntax: <fmt:setBundle var="--" basename="--"/>

Where var attribute will take a variable to hold up ResourceBundle object reference.

Where basename attribute will take base name of the properties file.

5. <fmt:message----->:

This tag can be used to get the message from ResourceBundle object on the basis of provided key.

Syntax: <fmt:message var="--" key="--"/>

Where var attribute will take a variable to hold up message.

Where key attribute will take key of the message defined in the respective properties file.

-----Application29-----

jstlapp3:**abc_en_US.properties:**

```
#abc_en_US.properties
#-----
welcome=Welcome to US user
```

abc_it_IT.properties:

```
#abc_it_IT.properties
#-----
welcome=Welcome toe Italiano usereo
```

fmt.jsp:

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

<%@taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<%@page isELIgnored="true"%>
<html>
    <body>
        <center><b><font size="7">
            <fmt:setLocale value="it_IT"/>
            <fmt:formatNumber var="num" value="123456.789"/>
            <c:out value="${num}"/><br><br>
            <jsp:useBean id="date" class="java.util.Date">
                <fmt:formatDate var="fdate" value="${date}"/>
                <c:out value="${fdate}"/>
            </jsp:useBean>
            <br><br>
            <fmt:setBundle basename="abc"/>
            <fmt:message var="msg" key="welcome"/>
            <c:out value="${msg}"/>
        </font></b></center>
    </body>
</html>
```

JSTL Functions:

The main purpose of functions tag library is to perform all the String operations which are defined in the String class.

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

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

    <c:set var="a" value="Ratan Software Solutions"/>

    ${fn.length(a)}

    ${fn.concat(a, "Hyderabad")}
```

```
${fn.toLowerCase(a)}
```

```
${fn.toUpperCase (a)}
```

```
${fn.contains(a, "Software")}
```

```
${fn.startsWith(a, "Ratan")}
```

```
${fn.endsWith(a, "Solutions")}
```

```
${fn.substring(a, 4, 20)}
```