

Adv. Java means DURGA SIR..

ADV.JAVA

WITH

SCWCD/OCWCD



India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143, 8096969696

Importance of ADVANCED JAVA:

DURGASOFT

- Every Software Company will do projects on J2EE Technologies
- If you attend any JAVA based Interview, 40% of the Questions are coming from ADVANCED JAVA only.
- If you are Strong in ADVANCED JAVA then it is very easy to learn all the future technologies and frameworks like STRUTS, JSF, SPRING,.....
- If you are Strong in ADVANCED JAVA surveying in software companies is very easy.
- ADVANCED JAVA is basis for all the enterprise applications.

Why DURGA sir ADVANCE JAVA:

- Covered Advance java topics in more depth.
- Provide live execution for each and every application in the class itself.
- Very good notes dictation for each and every topic which contains 500+ pages.
- Certification Oriented training on each and every topic (OCWCD).
- Conducting classes Even in Sundays for the benefit of Students i.e Covering each and every topic from scratch and also to complete course with in the time.
- Reviewing previous day topic before starting the class.
- Conducting interview questions sessions at the end of each and every topic.
- Covering interview questions and answers in detail in the notes dictation.
- Providing 2 mini projects on Advance JAVA.

Java Database Connectivity:

1. Storage Areas

- 1)Temporary Storage Areas
- 2)Permanent Storage Areas

2.Query Processing System

- 1)Query Tokenization
- 2)Query Processing
- 3)Query Optimization
- 4)Query Execution

3.Driver and Driver Types

- 1)Type 1 Driver
- 2)Type 2 Driver
- 3)Type 3 Driver
- 4)Type 4 Driver

4.Steps To design Jdbc Applications

- 1)Load and register the Driver.
- 2)Establish the connection between Java Application.
- 3)Prepare either Statement or preparedStatement or CallableStatement Objects.
- 4)Write and execute SQL Queries.
- 5)close the connection.

6.ResultSet and ResultSet Types

- 1)Read only ResultSet
- 2)UpdatableResultSet
- 3)Forward only ResultSet
- 4)ScrollableResultSets
 - 1)Scroll Sensitive ResultSet
 - 2)Scroll Insensitive ResultSet

7.Prepared Statement

- 1)PreparedStatement with insert sql query
- 2)PreparedStatement with update sql query
- 3)PreparedStatement with select sql query

8.Callable Statement

- 1)CallableStatement with procedure
- 2)CallableStatement with function
- 3)CallableStatement with CURSOR Type Procedure
- 4)CallableStatement with CURSOR type function

9.Transaction Management

1. Atomicity
2. Consistency
3. Isolation
4. Durability

10.Savepoint

11.BatchUpdations

12.Connection Pooling

13.BLOB and CLOB

Servlets:

1.Introduction

- 1) Standalone Applications
- 2) Enterprise Applications

2.Client-Server Arch

- 1)Client
- 2)Server
- 3)Protocol

3.Servlets Design

4.Servlets Lifecycle

5.User Interface

1.Static Form Generation

2.Dynamic Form Generation

6.ServletConfig

7.Servlet Context

8.Servlet Communication

1. Browser-servlet
2. Web-component
3. Applet-Servlet

9.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

10.Servlets Filters

11.Servlets Wrappers

12.Servlets Listeners

Java Server Pages:

1.Introduction

2.JSP Deployment

3.JSPLifeCycle

4.Jsp Elements

 1.Jsp Directives

 2.Scripting Elements

 3.Jsp Actions

5.JSP Directives

1. Page Directive
2. Include Directive
3. Taglib Directive

6.JSP Scripting Elements

1. Declarations
2. Scriptlets
3. Expressions

7.JSP implicit objects

- 1.out
- 2. request
- 3. response
- 4. config
- 5. application
- 6. session
- 7. exception
- 8. page
- 9. pageContext
- 8.JSP Scopes

1.Page Scope

2.request Scope

3.Application Scope

4.SessionScope

9.JSP Standard Actions

10.JSP Custom Actions

11.JSTL

- 1. Core Tags
- 2. XML Tags
- 3. Internationalization or I18N Tags (Formatted tags)
- 4. SQL Tags
- 5. Functions tags

12.Expression Language

- 1)EL operators
- 2)EL implicit objects.
- 3)EL functions.

Storage Areas

As part of the Enterprise Application development it is essential to manage the organizations data like Employee Details, CustomerDetails, Products Details..etc

-->To manage the above specified data in enterprise applications we have to use storage areas (Memoryelements).There are two types of Storage areas.

1) Temporary Storage Areas:

These are the memory elements, which will store the data temporarily
Eg:Buffers,Java Objects

2) Permanent Storage Objects:

These are the memory elements which will store data permanently.
Eg:FileSystems,DBMS,DataWareHouses.

File Systems:

It is a System, it will be provided by the local operating System.

-->Due to the above reason File Systems are not suitable for platform independent technologies like JAVA.

-->File Systems are able to store less volumes of the data.

-->File Systems are able to provide less security.

-->File Systems may increases data Redundancy.

-->In case of File Systems Query Language support is not available. So that all the database operations are complex.

DBMS:

-->Database Management System is very good compare to file System but still it able to store less data when compared to DataWareHouses.

-->DBMS is very good at the time of storing the data but which is not having Fast Retrieval Mechanisms.

DURGASOFT

JDBC

MR.NAGOORBABU

DataWareHouses:

When Compared to File Systems and DBMS it is able to store large and large volumes of data.

-->Data ware houses having fast retrieval mechanisms in the form of data mining techniques.

Q)What is the difference between database and database management system?

Ans:

DataBase is a memory element to store the data.

Database Management System is a Software System,it can be used to manage the data by storing it on database and retrieving it from Database.

Database is a collection of interrelated data as a single unit.

DBMS is a collection of interrelated data and a set of programs to access the data.

There are three types of DBMS:

1)RDBMS(Relational Database Management Systems)

2)OODBMS(Object Oriented DataBase Management Systems)

3)ORDBMS(Object Relational DataBase Management Systems)

1)Relational Database Management Systems:

-->It is a DBMS,it can be used to represent the data in the form of tables.

-->This DBMS will use SQL3 as a Query Language to perform DataBase Operations.

2)Object Oriented DataBase Management System:

-->It is DataBase Management System,it will represents the data in the form of Objects.

-->This database management system will require OQL(Object Query Language)as Query language to perform database operations.

3)Object Relational DataBase Management System:

-->It is a DataBaseManagement System,it will represents some part of data in the form of Tables and some other part of the data in the form of objects

-->This DataBaseManagement System will require SQL3 as Query Language to perform database operations.

Where SQL3 is the combination of SQL2 and OQL

DURGASOFT

JDBC

MR.NAGOORBABU

SQL3=SQL2+OQL

Query Processing System:

When we submit an SQL Query to the Database then Database Engine will Perform the following Steps.

Step1:

Query Tokenization:

This Phase will take SQL Query as an Input, divided into no.of tokens and Generate Stream of tokens as an output.

Step2:

Query Processing:

This phase will take Stream of tokens as an Input, constructs Query Tree with the Tokens, if Query Tree Success then no Syntax error is available in the provided SQL Query. If Query Tree is not Success then there are some syntax errors in the provided SQL Query.

Step3:

Query Optimization:

The main purpose of Query Optimization phase is to perform optimization on Query Tree in order to reduce execution time and to optimize memory utilization.

Step4:

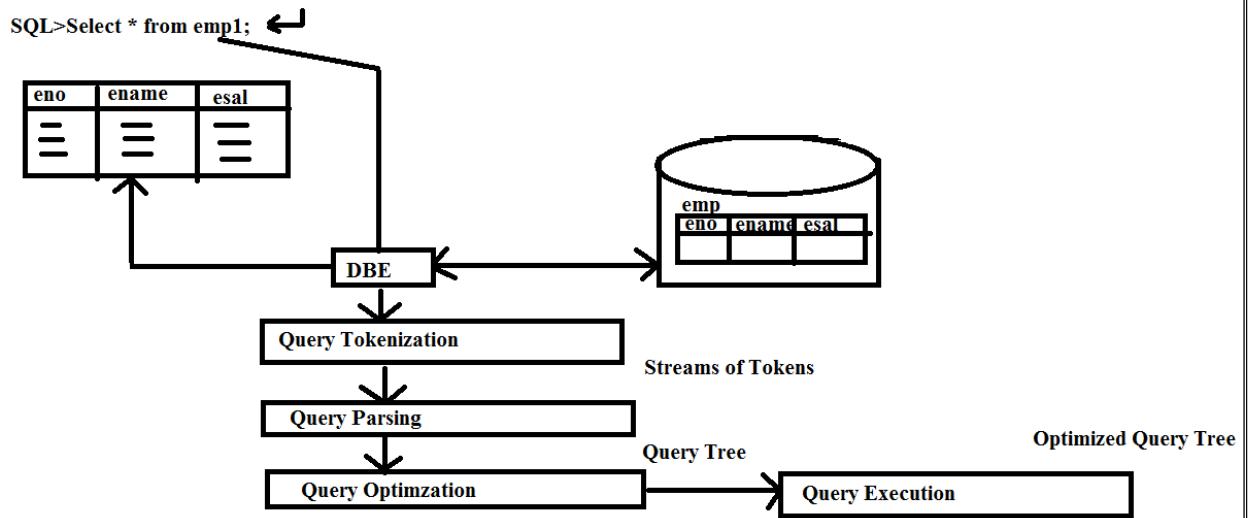
Query Execution:

This phase will take optimized Query Tree as an input and execute the Query by using interpreters.

DURGASOFT

JDBC

MR.NAGOORBABU



JDBC(Java DataBase Connectivity):

-->The process of interacting with the database from Java Applications is called as JDBC.

-->JDBC is an API,which will provide very good predefined library to connect with database from JAVAApplications in order to perform the basic database operations:

-->In case of JDBC Applications we will define the database logic and Java application and we will send a Java represented database logic to Database Engine.But database engine is unable to execute the Java represented database logic,it should required the database logic in Query Language Representations.

-->In the above context, to execute JDBC applications we should require a conversion mechanism to convert the database logic from Java representations to Query language representations and fromQuery language representations to Java representations.

-->In the above situation the required conversion mechanisms are available in the form of a software called as "Driver".

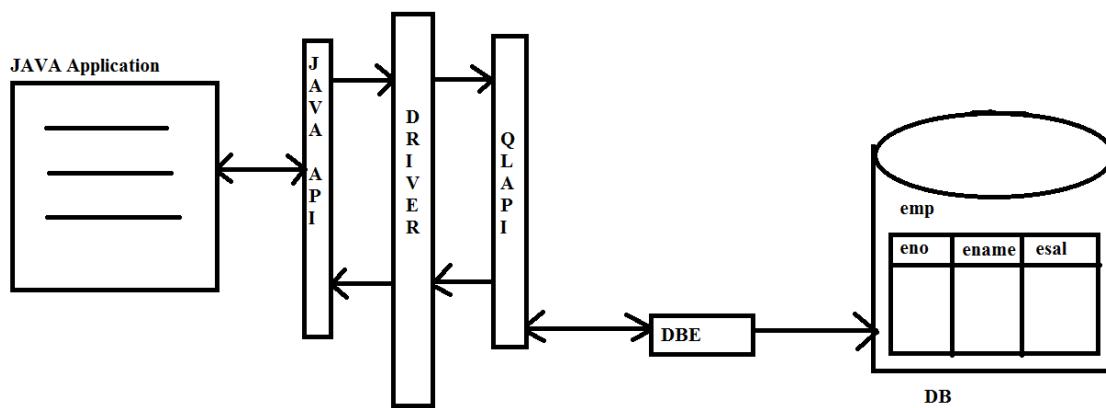
Driver:

-->Driver is an interface existed between Java application and database to map Java API calls to Query language API calls and Query language API calls to Java API calls.

DURGASOFT

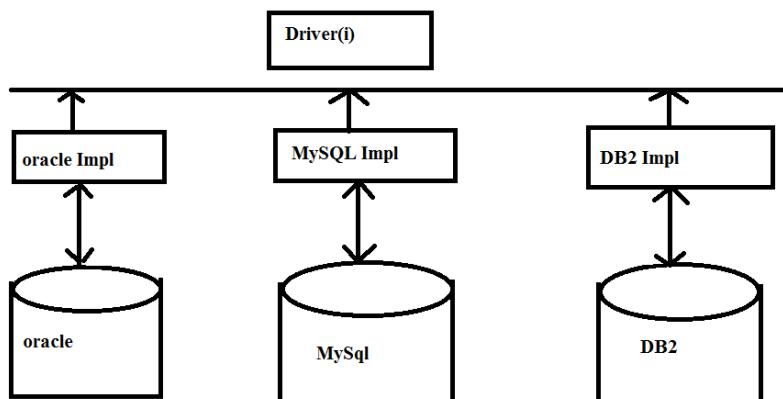
JDBC

MR.NAGOORBABU



-->To provide driver as a product Sun MicroSystems has provided Driver as an interface and Sun MicroSystems lets the database vendors to provide implementation classes to the driver interface as part of their database software's.

-->If we want to use Drivers in JDBC applications then we have to get Driver implementation from the respective database software's.



-->There are 180+ number of drivers but all these drivers could be classified into the following four types

- 1) Type 1
- 2) Type 2
- 3) Type 3
- 4) Type 4

DURGASOFT

JDBC

MR.NAGOORBABU

1) Type 1 Driver:

--> Type 1 Driver is also called as JDBC-ODBC Driver and Bridge Driver.

--> JDBC-ODBC Driver is a driver provided by Sun Micro Systems as an Implementation to Driver Interface.

--> Sun MicroSystems has provided JDBC-ODBC Driver with the inter dependent on the Microsoft's product ODBC Driver.

--> ODBC Driver is a Open Specification, it will provide very good environment to interact with any type of database from JDBC-ODBC Driver.

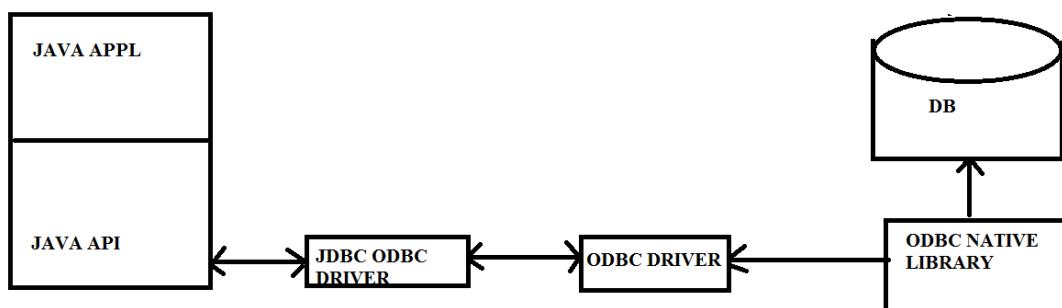
--> If we want to use JDBC-ODBC Driver in our JDBC Applications first we have to install the MicroSoft Product ODBC Driver native library.

--> To interact with the database from Java Application if we use JDBC-ODBC Driver then we should require two types conversions so that JDBC-ODBC Driver is Slower Driver.

--> JDBC-ODBC Driver is highly recommended for stand alone applications, it is not suitable for web applications, distributed applications and so on.

--> JDBC-ODBC Driver is suggestable for Simple JDBC applications, not for complex JDBC applications.

--> The portability of the JDBC-ODBC Driver is very less.



2) Type 2 Driver:

--> Type 2 Driver is also called part java,part native driver that is Type 2 Driver was implemented by using Java implementations and the database vendor provided native library.

--> When compared to Type1 Driver Type2 Driver is faster Driver because it should not require two times conversions to interact with the Database from Java Applications.

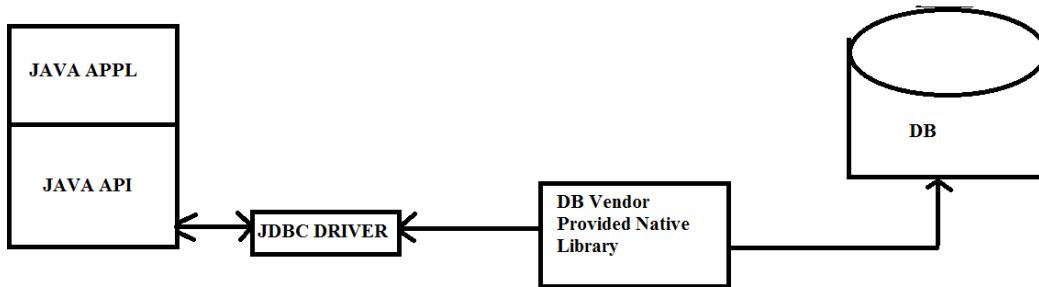
--> When compared to Type1 Driver Type2 driver portability is more.

--> Type2 Driver is still recommended for standalone application not suggestible for web applications and Enterprise applications.

--> If we want to use Type2 Driver in our Jdbc applications then we have to install the database vendor provided native library.

--> Type2 Driver is cast full Driver among all the drivers.

--> Type2 Driver's portability is not good when compared to Type3 Driver and Type4 Driver.



3) Type 3 Driver:

--> Type 3 Driver is also called as MiddleWare DataBase Server Access Driver and NetWorkDriver.

--> Type 3 Driver is purely designed for Enterprise applications it is not suggestible for stand alone applications.

--> Type 3 Driver portability is very good when compared to Type1 and Type2 Driver's.

DURGASOFT

JDBC

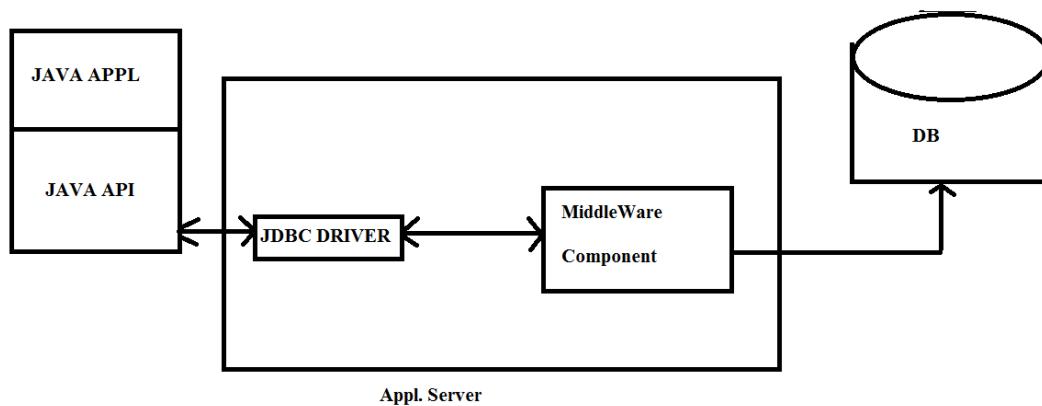
MR.NAGOORBABU

-->Type 3 Driver will provide very good environment to interact with multiple no.of databases.

-->Type 3 Driver will provide very good environment to switch from one database to another database without having modifications in client applications.

-->Type 3 Driver should not require any native library installations, it should require the Compatibility with the application server.

-->Type 3 Driver is fastest Driver when compared to all the Drivers.



4) Type 4 Driver:

-->Type 4 Driver is also called as pure Java Driver and Thin Driver because Type 4 Driver was implemented completely by using java implementations.

-->Type 4 Driver is the frequent used Driver when compared to all the remaining Drivers.

-->Type 4 Driver is recommended for any type application includes standalone applications, Network Applications....

-->Type 4 Driver portability is very good when compared to all the remaining Drivers.

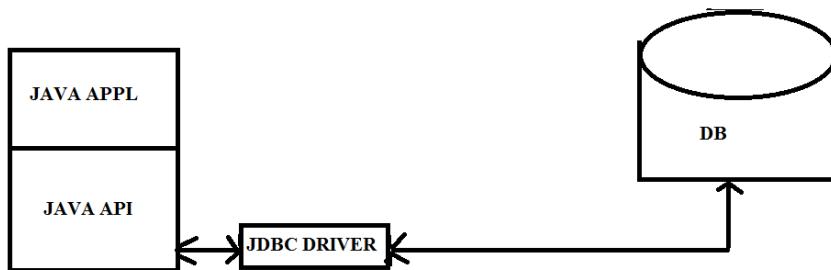
-->Type 4 driver should not require any native library dependences and it should require one time conversion to interact with database from Java Applications.

-->Type 4 is the cheapest Driver among all.

DURGASOFT

JDBC

MR.NAGOORBABU



Steps to design JDBC Application:

- 1)Load and register the Driver.
- 2)Establish the connection between Java Application.
- 3)Prepare either Statement or preparedStatement or CallableStatement Objects.
- 4)Write and execute SQL Queries.
- 5)close the connection.

1)Load and Register the Driver:

In general Driver is an interface provided by Sun Microsystems and whose implementation classes are provided by the Database Vendors as part of their Database Softwares.

-->To load and Register the Driver first we have to make available Driver implementation to JDBC application. For this we have to set classpath environment variable to the location Where we have Driver implementation class.

DURGASOFT

JDBC

MR.NAGOORBABU

-->If we want to use Type1 Driver provided by Sun MicroSystems in our JDBC applications then it is not required to set classpath environment variable because Type1 Driver was provided by Sun MicroSystems as part of Java Software in the form of **sun.jdbc.odbc.JdbcOdbcDriver**

-->If we want to use Type1 Driver in our JDBC applications then before loading we have to Configure the **Microsoft product odbc Driver**.

-->To configure Microsoft product odbc Driver we have to use the following path.

To setting DSN:-

- ```
Start
↓
Control Panel
↓
System and Security
↓
Administrative Tools
↓
Data Sources (ODBC)
↓
user DSN
↓
Click on Add button
↓
Select Microsoft ODBC for the Oracle
↓
Click on Finish button
↓
Provide DSN name (provide any name)
↓
Click on OK
```

-->To load and register Driver in our Jdbc applications we have to use the following method from class 'Class'

```
Public static Class forName(String class_Name)
```

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

-->When JVM encounter the above instruction JVM will pickup the parameter that is **JDBCOdbcDriver**

Class name and JVM will search for its .class file in the current location, if it is not available then JVM will search for it in Java predefined library.

DURGASOFT

JDBC

MR.NAGOORBABU

-->If JVM identify JDBCODBCDriver.class file in Java pre-defined library(rt.jar) then JVM will load

**JdbcOdbcDriver** class byte code to the memory.

-->At the time of loading JdbcOdbcDriver class byte code to the memory JVM will execute a static block,

As part of this JVM will execute a method call like **DriverManager.registerDriver(--);** by the execution of registerDriver() method only JDBCODBCDriver will be available to our Jdbc applications.

-->In case of Type1 Driver if we use either Jdbc 4.0 version or Jdk 6.0 version then it is optional

To perform loading and register the driver step because JVM will perform Driver registration automatically at the time of establishing the connection between Java application and Database.

NOTE:To prepare Jdbc applications Java API has provided the required pre-defined library in the form of java.sql package so that we have to import this package in our Java file.

Import java.sql.\*;

-->java.sql package includes the following pre-defined library to design Jdbc applications.

**java.sql package includes the following predefined library:-**

I-----→interface

C-----→class

1. Driver (I)
2. DriverManager (C)
3. Connection (I)
4. Statement (I)
5. PreparedStatement (I)
6. ResultSet (I)
7. ResultSetMetaData (I)
8. DatabaseMetaData (I)
9. Savepoint(I)

## 2) Establish the Connection between Java application and Database:

---

-->To establish the connection between Java application and Database we have to use the following Method from **DriverManager class**.

Public static Connection getConnection(String url, String db\_user\_name, String db\_password)

DURGASOFT

JDBC

MR.NAGOORBABU

Ex:Connection con=DriverManager.getConnection("jdbc:odbc:dsnName","system","durga");

-->When JVM encounter the above instruction JVM will access getConnection method,as part of the getConnection method JVM will access connect() method to establish virtual socket connection Between Java application and database as per the url which we provided.

-->where getConnection() method will take three parameters

- 1.Driver URL
- 2.Database username
- 3.Database password

-->In general from Driver to Driver Driver class name and Driver url will varied.

-->If we use Type1 Driver then we have to use the following Driver class name and URL

d-class : sun.jdbc.odbc.JdbcOdbcDriver  
url : jdbc:odbc:dsnName

-->In general all the Jdbc Drivers should have an url with the following format.

**main-protocol: sub-protocol**

-->where main-protocol name should be Jdbc for each and every Driver but the sub protocol name should be varied from Driver to Driver.

Q) In Jdbc applications getConnection() method will establish the connection between Java application and Database and return connection object but connection is an interface how it is possible to Create connection object?

Ans:In general in Java technology we are unable to create objects for the interfaces directly,if we want to accommodate interface data in the form of objects then we have to take either an implementation class or Anonymous Inner class.

-->If we take implementation class as an alternative then it may allow its own data a part from the data Declared in interface and implementation class object should have its own identity instead of interface identity.

-->If we want to create an object with only interface identity and to allow only interface data we have to use Anonymous inner class as an alternative.

DURGASOFT

JDBC

MR.NAGOORBABU

-->In jdbc applications getConnection() method will return connection object by returning anonymous Inner class object of connection interface.

NOTE: To create connection object taking an implementation class or anonymous inner class is Completely depending on the Driver Implementation.

### 3)Create either Statement or PreparedStatement or CallableStatement objects as per the requirement:

As part of the Jdbc applications after establish the connection between Java application and Database

We have to prepare SQL Queries,we have to transfer SQL Queries to the databaseEngine and we have to make Database Engine to execute SQL Queries.

-->To write and execute SQL Queries we have to use same predefined library from Statement prepared Statement and callableStatement.

-->To use the above required predefined library we have to prepare either Statement or preparedStatement or CallableStatement objects.

Q)What is the difference between Statement,PreparedStatement and Callable Statement Objects.

Ans:

-->In Jdbc applications when we have a requirement to execute all the SQL Queries independently we have to use Statement.

-->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 application we will use prepared Statement.

-->In jdbc applications when we have a requirement to access stored procedures and functions available At Database from Java application we will use Callable Statement object.

-->To prepare Statement object we have to use the following method from Connection.

Public Statement createStatement()

Ex: Statement st=con.createStatement();

DURGASOFT

JDBC

MR.NAGOORBABU

Where `createStatement()` method will return Statement object by creating Statement interfaces Anonymous inner class object.

## 4) Write and execute SQL Queries:

- 1.`executeQuery()`
- 2.`executeUpdate()`
- 3.`execute()`

Q) What are the differences between `executeQuery()`, `executeUpdate()` and `execute()` method?

Ans: Where `executeQuery()` method can be used to execute “selection group SQL Queries” in order to fetch(retrieve) data from Database.

--> when JVM encounter `executeQuery()` method with selection group SQL query then JVM will pickup

Selection group SQL Query, send to JdbcOdbcDriver, it will send to connection. Now connection will carry that SQL Query to the database engine through Odbc Driver.

--> At database database engine will execute the selection group SQL Query 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 and return to Java Application.

--> As Java technology is pure object oriented, Java application will store the fetched data in the form of an object at heap memory called as “ResultSet”.

--> As per the predefined implementation of `executeQuery` method JVM will return the generated ResultSet object reference as return value from `executeQuery()` method.

Public `ResultSet executeQuery(String sql_Query)` throws `SQLException`

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

--> where `executeUpdate()` method can be used to execute updation group SQL Queries in order to

perform the database operations like create, insert, update, delete, Drop....

--> when JVM encounter updation group SQL Query with `executeUpdate()` method the JVM will pickup

That Sql Query and send to Database through Database connection. At Database side Database engine

DURGASOFT

JDBC

MR.NAGOORBABU

Will execute it, perform updation from Database, identify rowCount value (number of records got updated) and return to Java application.

--> As per the predefined implementation of executeUpdate() method JVM will return row count value From executeUpdate() method.

```
Public int executeUpdate(String sql_Query) throws Exception
```

Ex: int rowCount=st.executeUpdate("update emp1 set esal=esal+500 where esal<1000");

--> In Jdbc applications execute() method can be used to execute both selection group and updation Group SQL Queries.

--> when JVM encounter selection group SQL Query with execute() method then JVM will send selection Group SQL Query to database engine, where database engine will execute it and send back the fetched Data to Java Application.

--> In Java application ResultSet object will be created with the fetched data but as per the predefined implementation of execute() method JVM will return "true" as a Boolean value.

--> when JVM encounter updation group SQL Query as parameter to execute() method then JVM Will send it to the database engine, where database engine will perform updations on database and return row Count value to Java application. But as per the predefined implementation of execute() method JVM will return "false" as Boolean value from execute() method

```
public Boolean execute(String sql_Query) throws SQLException.
```

Ex:

```
boolean b1=st.execute ("select * from emp1");
```

```
boolean b2=st.execute ("update emp1 set esal=esal+500 where esal<10000");
```

DURGASOFT

JDBC

MR.NAGOORBABU

## 5)Close the connection:

In Jdbc applications after the database logic it is convention to close Database connection for this we have to used the following method.

```
Public void close() throws SQLException
```

```
Ex:con.close();
```

**1)The following example demonstrate how to create a table on Database through a JDBC application by taking table name as Dynamic input.**

```
//import section
import java.sql.*;
import java.io.*;
class CreateTableEx{
public static void main(String args[]) throws Exception{

//load a register driver

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

//establish connection between Java application and database
Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");

//prepare Statement
Statement st=con.createStatement();

//create BufferedReader
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

//take table name as dynamic input

System.out.println("Enter table name");

String tname=br.readLine();

//prepare SQLQuery

String sql="create table " + tname + "(eno number,ename varchar2(10),esal number)";

//execute SQL Query

st.executeUpdate(sql);
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
System.out.println("table created successfully");

//close the connection

con.close();

}}
```

**2)The following example demonstrates how to insert no.of records on database table by taking records data as dynamic input.**

```
import java.io.*;
import java.sql.*;
public class JdbcApp2 {
 public static void main(String[] args)throws Exception {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection
 con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","d
urga");
 Statement st=con.createStatement();
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
 while(true)
 {
 System.out.print("Employee Number :");
 int eno=Integer.parseInt(br.readLine());
 System.out.print("Employee Name :");
 String ename=br.readLine();
 System.out.print("Employee Salary :");
 float esal=Float.parseFloat(br.readLine());
 System.out.print("Employee Address :");
 String eaddr=br.readLine();
 st.executeUpdate("insert into emp1
values("+eno+","+ename+","+esal+","+eaddr+ ")");
 System.out.println("Employee Inserted Successfully");
 System.out.print("Onemore Employee[Yes/No]? :");
 String option=br.readLine();
 if(option.equals("No"))
 {
 break;
 }
 }
 con.close();
 }}
```

DURGASOFT

JDBC

MR.NAGOORBABU

In Jdbc applications if we want to used Type1 driver provided by sun micro systems then we have to use the following Driver class and URL

```
driver.class:sun.jdbc.odbc.JdbcOdbcDriver
url:jdbc:odbc:dsnName
```

-->Similarly if we want to use Type4 Driver provided by oracle we have to use the following Driver class and URL

```
driver_class:oracle.jdbc.driver.OracleDriver
url:jdbc:oracle:thin:@localhost:1521:xe
```

-->Oracle Driver is a class provided by oracle software in the form of Ojdbc14.jar file

-->Oracle Software has provided ojdbc14.jar file at the following location

C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc.jar

-->If we want to use Type4 Driver provided by oracle in our Jdbc applications we have to set classpath environment variable to the location where we have ojdbc14.jar

```
D:\jdbc4>set
classpath=%classpath%;C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar
```

### 3)The following example Demonstrate how to perform updations on Database table through Jdbc Application

```
import java.io.*;
import java.sql.*;
public class JdbcApp3 {
 public static void main(String[] args) throws Exception {
 //Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
 System.out.print("Bonus Amount :");
 int bonus_Amt=Integer.parseInt(br.readLine());
 System.out.print("Salary Range :");
 float sal_Range=Float.parseFloat(br.readLine());
 int rowCount=st.executeUpdate
 ("update emp1 set esal=esal+"+bonus_Amt+" where esal<"+sal_Range);
 System.out.println("Employees Updated :" +rowCount);
 con.close();
 }
}
```

**4)The following example demonstrates how to delete no.of records from database table through a Jdbc application**

```
import java.io.*;
import java.sql.*;
public class JdbcApp4 {
 public static void main(String[] args)throws Exception {
 DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
 System.out.print("Salary Range :");
 float sal_Range=Float.parseFloat(br.readLine());
 int rowCount=st.executeUpdate("delete from emp1 where esal<"+sal_Range);
 System.out.println("Records Deleted :" +rowCount);
 con.close();
 }
}
```

-->In jdbc application we will use executeUpdate() method to execute the Updation group SQL queries like create,insert,update,delete,drop,alter and so on.

-->If we execute the SQL Queries like insert,update and delete then really some no.of record will be updated on database table then that number will be return as rowCount value from executeUpdate().

-->If we execute the SQL Queries like create,alter,drop with executeUpdate() method then records manipulation is not available on database,in this context the return value from executeUpdate() method is completely depending on the type of Driver which we used in JDBC application.

-->In the above context if we use type1 Driver provided by SunMicroSystems the executeUpdate() method will return "-1" as rowCount value.

-->For the above requirement if we use Type4 Driver provided by oracle then executeUpdate() method will return "0" as rowCount value

DURGASOFT

JDBC

MR.NAGOORBABU

```
5) import java.sql.*;
public class JdbcApp5 {
 public static void main(String[] args)throws Exception {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
 Statement st=con.createStatement();
 int rowCount1=st.executeUpdate("create table emp1(eno number)");
 System.out.println(rowCount1);
 int rowCount2=st.executeUpdate("drop table emp1");
 System.out.println(rowCount2);
 con.close();
 }
}
```

```
6) import java.sql.*;
public class JdbcApp6 {
 public static void main(String[] args)throws Exception {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 int rowCount1=st.executeUpdate("create table emp1(eno number)");
 System.out.println(rowCount1);
 int rowCount2=st.executeUpdate("drop table emp1");
 System.out.println(rowCount2);
 con.close();
 }
}
```

## ResultSet:

In Jdbc applications if we use Selection group SQL Query as parameter to executeQuery() method then JVM will send that selection group SQL Query to the Database Engine, where Database Engine will execute that SQL Query, fetch the data from Database and send back to Java application.

--->At Java Application the fetched data will be stored in the form of an object at heap memory called as ResultSet.

DURGASOFT

JDBC

MR.NAGOORBABU

-->As per the predefined implementation of executeQuery method JVM will return the generated ResultSet object reference as return value.

```
ReslutSet rs=st.executeQuery("select * from emp1");
```

-->When ResultSet object is created automatically a cursor will be created positioned before the first record.

-->If we want to read the records data from resultset object the for each and every record we have to check whether the next record is available or not from resultset cursor position, if it is available then we have to move resultset cursor to the next record position.

-->To perform the above work we have to use the following method from resultset

```
public boolean next()
```

-->After getting ResultsSet cursor to a particular Record position we have to retrieve the data from respective columns,for this we have to use the following overloaded method

```
public xxx getxxx(int field_No)
```

```
public xxx getxxx(String field_Name)
```

**where xxx may be byte,short,int.....**

```
Ex: while(rs.next())
{
 System.out.println(rs.getInt(1));
 System.out.println(rs.getString(2));
 System.out.println(rs.getFloat(3));
}
```

**7)The following example demonstrate how to fetch the data from database through ResultSet object**

```
import java.sql.*;
import oracle.jdbc.*;
public class JdbcApp7 {
 public static void main(String[] args) throws Exception {
 OracleDriver driver=new OracleDriver();
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 ResultSet rs=st.executeQuery("select * from emp1");
 System.out.println("ENO\tENAME\tESAL\tEADDR");
 System.out.println("-----");
 while(rs.next()){
 System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
```

DURGASOFT

JDBC

MR.NAGOORBABU

```

 +"\\t"+rs.getFloat(3)+"\\t"+rs.getString(4));
 }
con.close();
}
}

```

-->In jdbc applications execute() method can be used to execute both selection group SQL Queries and updation group SQL Queries one at a time.

-->If we use execute() method to execute selection group SQL Query then JVM will send that SQL Query to database engine where Database engine will execute Selection group SQL Query, fetch data from database and return to Java application. At java application the fetched data will be stored in the form of ResultSet object but as per the internal implementation of execute() method JVM will return "true" as a boolean value.

-->In the above context to retrieve the data from resultset object we have to get ResultSetObject reference explicitly.

-->To get ResultSet object reference explicitly we have to use the following method from Statement.

#### **Public ResultSet getResultSet() throws SQLException**

```

8. import java.sql.*;
public class JdbcApp8 {
 public static void main(String[] args) throws Exception {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 boolean b=st.execute("select * from emp1");
 System.out.println(b);
 ResultSet rs=st.getResultSet();
 System.out.println("ENO\\tENAME\\tESAL\\tEADDR");
 System.out.println("-----");
 while(rs.next()){
 System.out.println(rs.getInt(1)+"\\t"+rs.getString(2)
 +"\\t"+rs.getFloat(3)+"\\t"+rs.getString(4));
 }
 }
}

```

DURGASOFT

JDBC

MR.NAGOORBABU

```

 con.close();
 }
}

```

-->If we use updation group SQL Query as parameter to execute() method then JVM will send that updation group SQL Query to Database engine, where database engine will execute it, perform updations on Database table and return the generated row Count value to Java application.

-->As per the predefined implementation of execute() method JVM will return "false" as a boolean value from execute() method

-->In the above context to retrieve the generated rowCount value we have to use the following method from statement

```
public int getUpdateCount() throws SQLException
```

```

9. import java.sql.*;
public class JdbcApp9
{
 public static void main(String[] args) throws Exception
 {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 boolean b=st.execute("update emp1 set esal=esal+500 where esal<10000");
 System.out.println(b);
 int rowCount=st.getUpdateCount();
 System.out.println("Records Updated :"+rowCount);
 con.close();
 }
}

```

Q) If we use updation group SQL Query as parameter to executeQuery() method then what will be the response from JDBC application?

Ans: In Jdbc applications if we use updation group SQL Query as parameter to executeQuery() method then JVM will send updation group SQL Query to DatabaseEngine, where Database Engine will perform updations and Database and return rowCount Value to Java application but as per the predefined implementation of executeQuery() method JVM will expect ResultSet Object.

--> In the above context raising an exception or not to raise an exception is completely depending on the Type of Driver which we used.

DURGASOFT

JDBC

MR.NAGOORBABU

-->For the above requirement if we use Type1 driver then JVM will raise an Exception like java.sql.SQLException: no ResultSet was produced.

```

10.import java.sql.*;
public class JdbcApp12 {
 public static void main(String[] args) {
 Statement st=null;
 try{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:odbc:nag","system","durga");
 st=con.createStatement();
 ResultSet rs=st.executeQuery
 ("update emp1 set esal=esal+500 where esal<10000");
 }
 catch (Exception e){
 e.printStackTrace();
 try{
 int rowCount=st.getUpdateCount();
 System.out.println("Row Count :"+rowCount);
 }catch(Exception e1){
 e1.printStackTrace();
 }
 }
 }
}

```

NOTE:In the above application if we provide getUpdateCount method then we are able to generated the rowCount value in the catch block.

-->For the above requirement if we use Type4 Driver provided by oracle then JVM will not raise any exception,it will prepare a default ResultSet object implicitly.

```

import java.util.*;
class TestTwo{
 public static void main(String args[]){
 Statement st=null;
 try{
 Class.forName("oracle.jdbc.driver.oracleDriver");
 Connection
 con=DriverManager.getConnection(jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 st=con.createStatement();
 ResultSet rs=st.executeQuery("update emp2 set esal=esal+500 where esal<10000");
 }
 }
}

```

DURGASOFT

JDBC

MR.NAGOORBABU

```

int rowCount=st.executeUpdate();
System.out.println("Records Updated.." +rowCount);
}
catch(Exception e){
e.printStackTrace();
}}}

```

Q) In Jdbc applications if we provide selection group SQL Query as parameter to executeUpdate()

Method then what will be the response from Jdbc application?

Ans: In jdbc applications if we provide selection group SQL Query as parameter to executeUpdate()

method then JVM will send that SQL Query to database Engine, where Database engine will fetch the data from database and return to Java application.

--> At java application the returned data will be stored in the form of ResultSet object.

--> As per the predefined implementation of executeUpdate() method JVM will expecting Integer value.

--> In the above context getting an exception or not is completely depends on the Driver which we used in our Jdbc application

--> For the above requirement if we use Type1 Driver then JVM will raise an exception like java.sql.SQLException: no rowCount was produced.

NOTE: In the above situation if we getResultSet() method in catch block then we are able to get Resultset object.

```

11. import java.sql.*;
public class JdbcApp10 {
public static void main(String[] args) {
 Statement st=null;
 try{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
 st=con.createStatement();
 int rowCount= st.executeUpdate("select * from emp1");
 }catch(Exception e){
 e.printStackTrace();
 try{
 ResultSet rs=st.getResultSet();
 System.out.println("ENO\tENAME\tESAL\tEADDR");
 System.out.println("-----");
 }
 }
}

```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 while(rs.next())
 {
 System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
 +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
 }
 }catch(Exception e1){
 e1.printStackTrace();
 }
}
```

-->For the above requirement if we use Type4 Driver then JVM will not raise any Exception, executeUpdate() method will return how many no.of records are retrieved from Database table.

```
12. import java.sql.*;
public class JdbcApp11 {
 public static void main(String[] args) {
 try{
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 int rowCount=st.executeUpdate("select * from emp1");
 System.out.println("Row Count :"+rowCount);
 ResultSet rs=st.getResultSet();
 System.out.println("ENO\tENAME\tESAL\tEADDR");
 System.out.println("-----");
 while(rs.next())
 {
 System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
 +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
 }
 }catch (Exception e)
 {
 e.printStackTrace();
 }
 }
}
```

**13.The following example demonstrated how to retrieve the Data from Database and how to display that data through an HTML page**

```
import java.sql.*;
import java.io.*; public
class JdbcApp14
{
 public static void main(String[] args)throws Exception
 {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 Statement st=con.createStatement();
 ResultSet rs=st.executeQuery("select * from emp1");
 String data="";
 data=data+"<html><body><center><table border='1' bgcolor='lightblue'>";
 data=data+"<tr><td>ENO</td><td>ENAME</td><td>ESAL</td><td>EADDR</td></tr>";
 while(rs.next())
 {
 data=data+"<tr>";
 data=data+"<td>"+rs.getInt(1)+"</td><td>";
 data=data+"<td>"+rs.getString(2)+"</td><td>"+rs.getFloat(3)+"</td><td>";
 data=data+"<td>"+rs.getString(4)+"</td>";
 data=data+"</tr>";
 }
 data=data+"</table></center></body></html>";
 FileOutputStream fos=new FileOutputStream("emp.html",true);
 byte[] b=data.getBytes();
 fos.write(b);
 System.out.println("Open emp.html file to get Employees data");
 fos.close();
 con.close();
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

### Jdbc-awt appl:

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

```
public class LoginFrame extends Frame implements ActionListener {
 Label l1,l2;
 TextField tf1,tf2;
 Button b1;
 String status="";
 public LoginFrame() {
 this.setVisible(true);
 this.setSize(500,500);
 this.setTitle("Login Frame");
 this.setBackground(Color.green);
 this.setLayout(new FlowLayout());
 this.addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent we){
 System.exit(0);
 }
 });
 l1=new Label("User Name");
 l2=new Label("Password");
 tf1=new TextField(20);
 tf2=new TextField(20);
 tf2.setEchoChar('*');
 b1=new Button("Login");
 b1.addActionListener(this);

 Font f=new Font("arial",Font.BOLD,20);
 l1.setFont(f);
 l2.setFont(f);
 tf1.setFont(f);
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 tf2.setFont(f);
 b1.setFont(f);

 this.add(l1);
 this.add(tf1);
 this.add(l2);
 this.add(tf2);
 this.add(b1);
 }

 public void actionPerformed(ActionEvent ae) {
 String uname=tf1.getText();
 String upwd=tf2.getText();
 UserService us=new UserService();
 status=us.checkLogin(uname,upwd);
 repaint();
 }

 public void paint(Graphics g){
 Font f=new Font("arial",Font.BOLD,30);
 g.setFont(f);
 g.drawString("Status :"+status, 50, 250);
 }
}

package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class UserService {
 Connection con;
 Statement st;
 ResultSet rs;
 String status="";
 public UserService() {
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system", "durga");
 st=con.createStatement();
 } catch (Exception e) {
 e.printStackTrace();
 }
 }

 public String checkLogin(String uname, String upwd){
 try {
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
rs=st.executeQuery("select * from registered_Users where
uname='"+uname+"' and upwd='"+upwd+"');
boolean b=rs.next();
if(b==true){
 status="Login Success";
}else{
 status="Login Failure";
}
} catch (Exception e) {
 e.printStackTrace();
}
return status;
}

}

package com.durgasoft;
public class JdbcApp15 {

 public static void main(String[] args) {
 LoginFrame lf=new LoginFrame();
 }
}
```

### Jdbc-awt app2

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
public class EmployeeAddFrame extends Frame implements ActionListener {
 Label l1,l2,l3,l4;
 TextField tf1,tf2,tf3,tf4;
 Button b1;
 String status="";
 public EmployeeAddFrame(){
 this.setVisible(true);
 this.setSize(500, 500);
 this.setTitle("Employee Registration Frame");
 this.setBackground(Color.cyan);
 this.setLayout(null);
 this.addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent we){
 System.exit(0);
 }
 });

 l1=new Label("Employee Number");
 l2=new Label("Employee Name");
 l3=new Label("Employee Salary");
 l4=new Label("Employee Address");

 tf1=new TextField(20);
 tf2=new TextField(20);
 tf3=new TextField(20);
 tf4=new TextField(20);

 b1=new Button("ADD");
 b1.addActionListener(this);

 Font f=new Font("arial",Font.BOLD,20);
 l1.setFont(f);
 l2.setFont(f);
 l3.setFont(f);
 l4.setFont(f);
 tf1.setFont(f);
 tf2.setFont(f);
 tf3.setFont(f);
 tf4.setFont(f);
 b1.setFont(f);

 l1.setBounds(50, 100, 200, 25);
 tf1.setBounds(250, 100, 200, 30);
 l2.setBounds(50, 150, 200, 25);
 tf2.setBounds(250, 150, 200, 30);
 l3.setBounds(50, 200, 200, 25);
 tf3.setBounds(250, 200, 200, 30);
 l4.setBounds(50, 250, 200, 25);
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 tf4.setBounds(250, 250, 200, 30);
 b1.setBounds(50, 300, 100, 30);
 this.add(l1);
 this.add(tf1);
 this.add(l2);
 this.add(tf2);
 this.add(l3);
 this.add(tf3);
 this.add(l4);
 this.add(tf4);
 this.add(b1);
 }
 public void actionPerformed(ActionEvent ae){
 try {
 int eno=Integer.parseInt(tf1.getText());
 String ename=tf2.getText();
 float esal=Float.parseFloat(tf3.getText());
 String eaddr=tf4.getText();

 EmployeeService es=new EmployeeService();
 status=es.add(eno,ename,esal,eaddr);
 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:"+status, 50, 400);
 }
}

package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class EmployeeService {
 Connection con;
 Statement st;
 ResultSet rs;
 String status="";
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
public EmployeeService() {
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");
 st=con.createStatement();
 } catch (Exception e) {
 e.printStackTrace();
 }
}

public String add(int eno, String ename, float esal, String eaddr){
 try {
 rs=st.executeQuery("select * from emp1 where enno="+eno);
 boolean b=rs.next();
 if(b==true){
 status="Employee Existed Already";
 }else{
 st.executeUpdate("insert into emp1
 values("+eno+","+ename+","+esal+","+eaddr+ ")");
 status="Employee Registration Success";
 }
 } catch (Exception e) {
 status="Employee Registration Failure";
 e.printStackTrace();
 }
 return status;
}

}

package com.durgasoft;

public class JdbcApp16 {

 public static void main(String[] args) {
 EmployeeAddFrame f=new EmployeeAddFrame();

 }

}
```

DURGASOFT

JDBC

MR.NAGOORBABU

## Jdbc-Awt App3

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class StudentSearchFrame extends Frame implements ActionListener {
 Label l1;
 TextField tf1;
 Button b1;
 StudentTo sto;
 public StudentSearchFrame() {
 this.setVisible(true);
 this.setSize(500, 500);
 this.setTitle("Student Search Frame");
 this.setLayout(new FlowLayout());
 this.setBackground(Color.green);
 this.addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent we){
 System.exit(0);
 }
 });
 l1=new Label("Student Id");
 tf1=new TextField(20);
 b1=new Button("Search");
 b1.addActionListener(this);

 Font f=new Font("arial",Font.BOLD,20);
 l1.setFont(f);
 tf1.setFont(f);
 b1.setFont(f);

 this.add(l1);
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 this.add(tf1);
 this.add(b1);
 }
 public void actionPerformed(ActionEvent arg0) {
 String sid=tf1.getText();
 StudentService ss=new StudentService();
 sto=ss.search(sid);
 repaint();
 }
 public void paint(Graphics g){
 Font f=new Font("arial",Font.BOLD,30);
 g.setFont(f);
 if(sto==null){
 g.drawString("Student Not Existed", 50, 300);
 }else{
 g.drawString("Student Id :" +sto.getId(), 50, 250);
 g.drawString("Student Name :" +sto.getName(), 50, 300);
 g.drawString("Student Address :" +sto.getAddress(), 50, 350);
 }
 }
}

package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class StudentService {
 Connection con;
 Statement st;
 ResultSet rs;
 StudentTo sto;
 public StudentService() {
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 con=DriverManager.getConnection("jdbc:oracle:oci8:@xe", "system",
 "durga");
 st=con.createStatement();
 } catch (Exception e) {
 }
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
public StudentTo search(String sid){
 try {
 rs=st.executeQuery("select * from student where sid='"+sid+"'");
 boolean b=rs.next();
 if(b==true){
 sto=new StudentTo();
 sto.setSid(rs.getString(1));
 sto.setSname(rs.getString(2));
 sto.setSaddr(rs.getString(3));
 }else{
 sto=null;
 }
 } catch (Exception e) {
 e.printStackTrace();
 }
 return sto;
}
}
```

```
package com.durgasoft;
public class StudentTo {
 private String sid;
 private String sname;
 private String saddr;
 public String getSid() {
 return sid;
 }
 public void setSid(String sid) {
 this.sid = sid;
 }
 public String getSname() {
 return sname;
 }
 public void setSname(String sname) {
 this.sname = sname;
 }
 public String getSaddr() {
 return saddr;
 }
 public void setSaddr(String saddr) {
 this.saddr = saddr;
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
}

}

package com.durgasoft;

public class JdbcApp17 {

 public static void main(String[] args) {
 StudentSearchFrame sf=new StudentSearchFrame();

 }

}
```

## Jdbc-awt app4

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.Label;
import java.awt.TextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.ArrayList;

public class EditorFrame extends Frame implements ActionListener {
 Label l;
 TextArea ta;
 Button b;
 EditorService es;
 boolean bol;
 public EditorFrame(){
 this.setVisible(true);
 this.setSize(700, 800);
 this.setTitle("SQL Editor Frame");
 this.setBackground(Color.green);
 this.setLayout(new FlowLayout());
 this.addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent e){
 System.exit(0);
 }
 });
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 }

 });

l=new Label("Provide SQLQuery");
ta=new TextArea(3,30);
b=new Button("Execute");
b.addActionListener(this);

Font f=new Font("arial",Font.BOLD,20);
l.setFont(f);
ta.setFont(f);
b.setFont(f);

this.add(l);
this.add(ta);
this.add(b);

}

public void actionPerformed(ActionEvent ae){
 String query=ta.getText();
 es=new EditorService();
 bol=es.execute(query);
 repaint();
}

public void paint(Graphics g){
 Font f=new Font("arial",Font.BOLD,30);
 g.setFont(f);
 if(bol==true){
 ArrayList al=es.getEmps();
 g.drawString("ENO ENAME ESAL EADDR",50,200);
 g.drawString("-----", 50, 240);
 int y=300;
 for(int i=0;i<al.size();i++){
 EmployeeTo eto=(EmployeeTo)al.get(i);
 g.drawString(eto.getEno()+" "+eto.getEname()+" "+eto.getEsal()+" "+eto.getEaddr(), 50, y);
 y=y+50;
 }
 }else{
 int rowCount=es.getRowCount();
 g.drawString("Row Count :"+rowCount, 50, 300);
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;

public class EditorService {
 Connection con;
 Statement st;
 ResultSet rs;
 ArrayList al;
 boolean bol;
 int rowCount;
 public EditorService() {
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 con=DriverManager.getConnection("jdbc:oracle:oci8:@xe", "system",
"durga");
 st=con.createStatement();
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
 public boolean execute(String sql_Query){
 try {
 bol=st.execute(sql_Query);
 } catch (Exception e) {
 e.printStackTrace();
 }
 return bol;
 }
 public ArrayList getEmps(){
 al=new ArrayList();
 try {
 rs=st.getResultSet();
 while(rs.next()){
 EmployeeTo eto=new EmployeeTo();
 eto.setEno(rs.getInt(1));
 eto.setEname(rs.getString(2));
 eto.setEsal(rs.getFloat(3));
 eto.setEaddr(rs.getString(4));
 al.add(eto);
 }
 } catch (Exception e) {
 e.printStackTrace();
 }
 return al;
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 }
 }

 catch (Exception e) {
 e.printStackTrace();
 }
 return al;
}

public int getRowCount(){
 try {
 rowCount=st.getUpdateCount();
 } catch (Exception e) {
 e.printStackTrace();
 }
 return rowCount;
}
}
```

```
package com.durgasoft;

public class EmployeeTo {
 private int eno;
 private String ename;
 private float esal;
 private String eaddr;
 public int getEno() {
 return eno;
 }
 public void setEno(int eno) {
 this.eno = eno;
 }
 public String getEname() {
 return ename;
 }
 public void setEname(String ename) {
 this.ename = ename;
 }
 public float getEsal() {
 return esal;
 }
 public void setEsal(float esal) {
 this.esal = esal;
 }
 public String getEaddr() {
 return eaddr;
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 }
 public void setEaddr(String eaddr) {
 this.eaddr = eaddr;
 }

}

package com.durgasoft;

public class JdbcApp18 {

 public static void main(String[] args) {
 EditorFrame f=new EditorFrame();

 }

}
```

## ResultSet Types:

---

In jdbc applications ResultSets can be divided into two types:

-->As per the ResultSet concurrency there are two types of ResultSets.

1)Read only ResultSet

It is a ResultSet object,it will allow the users to read the data only.

To represent this ResultSet object ResultSet interface has provided the following constant

**public static final int CONCUR\_READ\_ONLY**

2)Updatable ResultSet:

It is a ResultSet object,it will allow the users to perform updations on its content.

To represent this resultset object ResultSet interface has provided the following constant.

**public static final int CONCUR\_UPDATABLE**

-->As per the ResultSet cursor movement there are two types of ResultSets

DURGASOFT

JDBC

MR.NAGOORBABU

## 1)Forward only ResultSet:

It is ResultSet object,it will allow the users to iterate the data in forward directiononly.

-->To represent this ResultSet,ResultSet interface has provided the following constant

```
public static final int TYPE_FORWARD_ONLY
```

## 2)Scrollable ResultSet:

These are the Resultset objects,which will allow the users to iterate the data in both forward and backward directions.

There are two types of Scrollable Results:

- 1)Scroll sensitive ResultSet
- 2)Scroll Insensitive ResultSet

Q)What are the differences betwnn Scroll Sensitive ResultSet and Scroll Insensitive ResultSet?

Scroll sensitive ResultSet is a Scrollable resultset object,which will allow the later Database updations.

-->To refer this ResultSet,ResultSet interface has provided the following constant

```
public static final int TYPE_SCROLL_SENSITIVE
```

-->Scroll Insensitive ResultSet is scrollable Resultset object,which will not allow the later database updations after creation.

-->To represent this ResultSet,ResultInterface has provided the following constant

```
public static final int TYPE_SCROLL_INSENSITIVE
```

-->The default ResultSet type in Jdbc applications is Forward only and Read only.

-->In Jdbc applications if we want to specify a particular type to the ResultSet object then we have to provide the above specified ResultSet constants at the time of creating Statement object,for this we have to use the following method

```
public Statement createStatement(int forwardonly)
```

DURGASOFT

JDBC

MR.NAGOORBABU

**(scrollsensitive scrollinsensitive,int Readonly/updatable)**

Ex: Statement

```
st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
```

-->In jdbc applications by using scrollable ResultSet we are able to retrieve the data in both forward direction and backward direction.

-->To retrieve the data in forward direction for each and every record we have to check whether the next record is available or not,if it is available we have to move resultset cursor to the next record position.when we refer a particular record then we have to retrieve the data from the respective columns.To achieve this we have to use the following piece of code:

```
while(rs.next())
{
 System.out.println(rs.getInt(1));
 System.out.println(rs.getString(2));
}
```

-->If we want to retrieve the data in Backward direction then for each and every record we have to check whether the previous record is available or not from the resultset cursor position,if is available then we have to move resultset cursor to the previous record.To achieve this we have to use the following methods:

**public boolean previous()**

-->After moving the resultset cursor to particular record then we have to retrieve the data from the corresponding columns for this we have to use the following methods

```
public xxx getxxx(int field_NUM)
public xxx getxxx(String field_Name)
```

**where xxx may be byte,short,int.....**

DURGASOFT

JDBC

MR.NAGOORBABU

```
Ex: while(rs.previous())
{
 System.out.println(rs.getInt(1));
 System.out.println(rs.getString(2));
 System.out.println(rs.getFloat(3));

}
```

```
14. package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcApp19 {

 public static void main(String[] args) throws Exception {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:oci8:@xe", "system", "durga");
 Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 ResultSet rs=st.executeQuery("select * from emp1");
 System.out.println("Data In Forward Direction");
 System.out.println("ENO\tENAME\tESAL\tEADDR\t");
 System.out.println("-----");
 while(rs.next()){
 System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"
 +rs.getFloat(3)+"\t"+rs.getString(4));
 }
 System.out.println("Data In Backward Direction");
 System.out.println("ENO\tENAME\tESAL\tEADDR");
 System.out.println("-----");
 while(rs.previous()){
 System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"
 +rs.getFloat(3)+"\t"+rs.getString(4));
 }
 con.close();
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
15.
package com.durgasoft;
import java.sql.Connection;
import
java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcApp20 {

 public static void main(String[] args) throws Exception {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con=DriverManager.getConnection
 ("jdbc:oracle:oci8:@xe", "system", "durga");
 Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 ResultSet rs=st.executeQuery("select * from emp1");
 rs.afterLast();
 rs.previous();
 System.out.println(rs.getInt(1));
 rs.beforeFirst();
 rs.next();
 System.out.println(rs.getInt(1));
 rs.last();
 System.out.println(rs.getInt(1));
 rs.first();
 System.out.println(rs.getInt(1));
 rs.absolute(3);
 System.out.println(rs.getInt(1));
 rs.absolute(-3);
 System.out.println(rs.getInt(1));
 rs.first();
 rs.relative(2);
 System.out.println(rs.getInt(1));
 rs.last();
 rs.relative(-2);
 System.out.println(rs.getInt(1));
 }

}
```

DURGASOFT

JDBC

MR.NAGOORBABU

## Jdbc-Awt app5

```
package com.durgasoft;

import java.awt.Button;
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class PlayerFrame extends Frame implements ActionListener {
 Button b1,b2,b3,b4;
 String label;
 EmployeeTo eto;
 EmployeeService es;
 public PlayerFrame() {
 this.setVisible(true);
 this.setSize(500, 500);
 this.setTitle("Player Frame");
 this.setBackground(Color.green);
 this.setLayout(null);
 this.addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent we){
 System.exit(0);
 }
 });
 b1=new Button("First");
 b2=new Button("Next");
 b3=new Button("Previous");
 b4=new Button("Last");

 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 b4.addActionListener(this);

 Font f=new Font("arial",Font.BOLD,20);
 b1.setFont(f);
 b2.setFont(f);
 b3.setFont(f);
 b4.setFont(f);
 }

 public void actionPerformed(ActionEvent ae) {
 if(ae.getSource()==b1)
 eto.getPrevious();
 else if(ae.getSource()==b2)
 eto.getNext();
 else if(ae.getSource()==b3)
 eto.getPrevious();
 else if(ae.getSource()==b4)
 eto.getNext();
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
b1.setBounds(50, 400, 100, 30);
b2.setBounds(160, 400, 100, 30);
b3.setBounds(270, 400, 100, 30);
b4.setBounds(380, 400, 100, 30);
this.add(b1);
this.add(b2);
this.add(b3);
this.add(b4);
es=new EmployeeService();
}
public void actionPerformed(ActionEvent ae){
 label=ae.getActionCommand();
 eto=es.getEmployee(label);
 repaint();
}
public void paint(Graphics g){
 Font f=new Font("arial",Font.BOLD,30);
 g.setFont(f);
 String msg=es.getMsg();
 if(msg.equals("")){
 g.drawString("Employee Number :"+eto.getEno(), 50,100);
 g.drawString("Employee Name :"+eto.getEname(), 50,150);
 g.drawString("Employee Salary :" +eto.getEsal(), 50,200);
 g.drawString("Employee Address :" +eto.getEaddr(), 50,250);
 }else{
 g.drawString(msg, 50,300);
 }
}
}

package com.durgasoft;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class EmployeeService {
 Connection con;
 Statement st;
 ResultSet rs;
 EmployeeTo eto;
 String msg="";
 boolean b=false;
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
public EmployeeService() {
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe", "system","durga");
 st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 rs=st.executeQuery("select * from emp1");
 } catch (Exception e) {
 e.printStackTrace();
 }
}

public EmployeeTo getEmployee(String label){
 try {
 if(label.equals("First")){
 rs.first();
 msg="";
 }
 if(label.equals("Next")){
 b=rs.next();
 if(b==false){
 msg="No More Records In Forward Direction";
 }else{
 msg="";
 }
 }
 if(label.equals("Previous")){
 b=rs.previous();
 if(b==false){
 msg="No More Records In Backward Direction";
 }else{
 msg="";
 }
 }
 if(label.equals("Last")){
 rs.last();
 msg="";
 }
 eto=new EmployeeTo();
 eto.setEno(rs.getInt(1));
 eto.setEname(rs.getString(2));
 eto.setEsal(rs.getFloat(3));
 eto.setEaddr(rs.getString(4));
 } catch (Exception e) {
 e.printStackTrace();
 }
 return eto;
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
public String getMsg(){
 return msg;
}

package com.durgasoft;

public class EmployeeTo {
 private int eno;
 private String ename;
 private float esal;
 private String eaddr;
 public int getEno() {
 return eno;
 }
 public void setEno(int eno) {
 this.eno = eno;
 }
 public String getEname() {
 return ename;
 }
 public void setEname(String ename) {
 this.ename = ename;
 }
 public float getEsal() {
 return esal;
 }
 public void setEsal(float esal) {
 this.esal = esal;
 }
 public String getEaddr() {
 return eaddr;
 }
 public void setEaddr(String eaddr) {
 this.eaddr = eaddr;
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
package com.durgasoft;

public class JdbcApp24 {

 public static void main(String[] args) {
 PlayerFrame pf=new PlayerFrame();
 }

}
```

**Scroll Sensitive ResultSet:**

```
16.
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcApp25 {
 public static void main(String[] args)throws Exception {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
 Statement st=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 ResultSet rs=st.executeQuery("select * from emp1");
 System.out.println("Data Before Updations");
 System.out.println("ENO ENAME ESAL EADDR");
 System.out.println("-----");
 while(rs.next()){
 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getFloat(3)+""
 "+rs.getString(4));
 }
 System.out.println("Application is in Pausing state, please update database");
 System.in.read();
 rs.beforeFirst();
 System.out.println("Data After Updations");
 System.out.println("ENO ENAME ESAL EADDR");
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
System.out.println("-----");
while(rs.next()){
 rs.refreshRow();
 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getFloat(3)+" "+
"rs.getString(4));
}
con.close();
}
```

-->To move ResultSet cursor to before first record we have to use the following method from ResultSet

**public void beforeFirst()**

-->To move ResultSet cursor to After the last record we have to use the following method from ResultSet

**public void afterLast()**

-->To move ResultSet cursor to a particular record we have to use the following method from ResultSet

**public void absolute(int rec\_position)**

-->In case of scroll sensitive ResultSet objects to reflect later database updatations into the ResultSet object we have to refresh each and every record for this we have to use the following method from ResultSet

**public void refreshRow()**

-->where refreshRow() method can be used to refresh only one record.

-->If we use Type4 Driver provided by oracle in the above application then JVM will raise an exception like java.sql.SQLException:unsupportedfeature:refreshrow

-->In jdbc applications scroll sensitive ResultSet object should be supported by Type1 Driver provided by Sun MicroSystems,which could not be supported by Type4 Driver provided by oracle.

-->In Jdbc applications scroll Insensitive ResultSet object could not be supported by both Type1 Driver provided by Sun MicroSystems and Type4 Driver provided by oracle.

-->In jdbc applications the main purpose of UpdatableResultSet object is to perform updatations on its content in order to perform the manipulations with the data available at Database

DURGASOFT

JDBC

MR.NAGOORBABU

-->In jdbc applications Updatable ResultSet objects can be used to insert records on database table to achieve the above requirement we have to use the following steps:

**Step1:get Updatable ResultSet object**

Statement

```
st=con.createStatement(resultSet.TYPE_SCROLL_SENSITIVE,resultSet.CONCUR_UPDATABLE);
```

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

**Step2: After creating ResultSet Object we have to move ResultSet cursor to end of the ResultSet object**

where we have to take a buffer to insert new Record data temporarily to achieve this we have to use the following method from ResultSet

```
public void moveToInsertRow()
```

```
Ex:rs.moveToInsertRow();
```

**Step3:Insert record data on resultset object temporarily to do this we have to use the following method**

```
public void updateXXX(int field_Num,xxx value)
```

```
Ex:rs.updateInt(1,555);
rs.updateString(2,'xyz');
rs.updateFloat(3,9000);
```

**Step4:Make the temporarily insertion as permanent insertion in resultset object as well as on database table to achieve this we have to use the following method**

```
public void insertRow()
```

```
Ex:rs.insertRow();
```

NOTE:The main advantage of this updatable ResultSet object is to perform updations on database table without using SQL Queries.

DURGASOFT

JDBC

MR.NAGOORBABU

17.

The following example demonstrates how to insert no.of records into database table through a Jdbc application

```
package com.durgasoft;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jdbcapp26 {

 public static void main(String[] args) throws Exception{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
 Statement
 st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
);

 ResultSet rs=st.executeQuery("select * from emp1");
 rs.moveToInsertRow();
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

 while(true){
 System.out.print("Employee Number :");
 int eno=Integer.parseInt(br.readLine());
 System.out.print("Employee Name :");
 String ename=br.readLine();
 System.out.print("Employee Salary :");
 float esal=Float.parseFloat(br.readLine());
 System.out.print("Employee Address :");
 String eaddr=br.readLine();
 rs.updateInt(1, eno);
 rs.updateString(2, ename);
 rs.updateFloat(3, esal);
 rs.updateString(4, eaddr);
 rs.insertRow();

 System.out.println("Employee inserted Successfully");
 System.out.print("Onemore Employee[Yes/no] :");
 String option=br.readLine();
 if(option.equals("no")){
 break;
 }
 }
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 con.close();
 }

}
```

NOTE: In jdbc applications updatable ResultSets could not be supported by Type 4 Driver provided by oracle

--> In jdbc applications by using updatable ResultSet object it is possible to update database

--> To perform this we have to use the following steps

#### **Step1: get updatable ResultSet object**

Statement

```
st=con.createStatement(resultSet.TYPE_SCROLL_SENSITIVE,resultSet.CONCUR_UPDATABLE);
```

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

#### **Step2: update Resultset Object Temporarily**

To achieve this we have to use the following method

```
public void updateXXX(int field_Name,xxx value)
```

```
ex: rs.updateFloat(3,7000.0f);
```

#### **Step3: Make temporary updation as permanent updation on resultset obejct as well as on database to achieve this we have to use the following method**

```
public void updateRow()
```

```
Ex:rs.updateRow();
```

18.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jdbcapp27 {
```

DURGASOFT

JDBC

MR.NAGOORBABU

```

public static void main(String[] args) throws Exception {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con=DriverManager.getConnection("jdbc:odbc:nag", "system", "durga");
 Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,
 ResultSet.CONCUR_UPDATABLE);

 ResultSet rs=st.executeQuery("select * from emp1");
 while(rs.next()){
 float esal=rs.getFloat(3);
 if(esal<10000){
 float new_Sal=esal+500;
 rs.updateFloat(3, new_Sal);
 rs.updateRow();
 }
 }
 con.close();
}

}

```

-->In Jdbc applications by using updatable ResultSet object it is possible to delete records on database table to achieve this we have to use the following method

**public void deleteRow()**

Ex:rs.deleteRow();

```

19.
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jdbcapp28 {
 public static void main(String[] args) throws Exception {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection
 con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga"
);
 Statement st=con.createStatement	ResultSet.TYPE_SCROLL_SENSITIVE,
 ResultSet.CONCUR_UPDATABLE);
 ResultSet rs=st.executeQuery("select * from emp1");
 rs.last();
 }
}

```

DURGASOFT

JDBC

MR.NAGOORBABU

```
rs.deleteRow();
st.close();
con.close();
}

}
```

-->To move ResultSet cursor to a particular Record position we have to use the following method from resultset

**public void absolute(int position)**

-->To move ResultSet cursor over some no.of records we have to use the following method from Resultset

**public void relative(int no.of.records)**

-->If we have bulk of records in database table,where if we are trying to retrieve all the records at a time into resultset object automatically Jdbc application performance will be reduced.

-->In the above context to improve the performance of Jdbc application we have to fetch the limited no. of records in multiple attempts.

-->To specify the no.of records which we want to fetch at an attempt then we have to use the following method

**public void setFetchSize(int size)**

-->To get the specified fetch size value from resultset we have to use the following method

**public int getFetchSize**

What is the difference between Statement and PreparedStatement?

Ans:

In Jdbc applications,when we have a requirement to execute the SQL queries independently,we have to use Statement.

In Jdbc applications,when we have a requirement to execute same SQL query in the next sequence where to improve the performance of Jdbc applications,we have to use PreparedStatement.

To achieve the above requirement,if we use Statement,then for every time of executing the same SQL query,DB engine has to perform Query Tokenization,Query Parsing,Query Optimization and Query Execution without having any validation from one time to another time.

DURGASOFT

JDBC

MR.NAGOORBABU

This approach will reduce the performance of Jdbc application.

In the above context,to improve the performance of Jdbc applications,we have to use an alternative where we have to perform Query Processing one time inorder to perform or execute the same SQL Query in the next sequence.

To achieve the above alternative,we have to use PreparedStatement over Statement.

If we want to use PreparedStatement in Jdbc applications we have to use the following steps.

## 1.Create PreparedStatement object by providing generalised SQL Query:

To create PreparedStatement object,we have to use the following method from Connection.

```
public PreparedStatement prepareStatement(String SQL_format)
```

Eg:

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

When JVM encounter the above instruction,JVM will pickup the provided SQL query and send to DB engine through Jdbc Driver and connection.

Upon receiving SQL query,DB engine will perform query processing and prepare query plan with the parameters,as a result PreparedStatement object will be created at Java application with the parameters

## 2.Set values to the parameters available in PreparedStatement object as per the applicationrequirement.

To set values to the PreparedStatement object,we have to the following method from PreparedStatement

```
public void setXXX(int parameter_Numb,xxx value)
where xxx may be byte,short,int....
```

Eg:

```
pst.setInt(1,111);
pst.setString(2,"Laddu");
pst.setFloat(3,30000.0f);
```

When JVM encounter the above instructions then JVM will set the specified values to the respective parameters in PreparedStatement object,where these values are reflected to Query plan parameters automatically through the Jdbc driver and connection.

DURGASOFT

JDBC

MR.NAGOORBABU

### 3. Make Database engine to pickup the values from Query plan and perform the respective operations per the generalised SQL query which we provided

If the generalised SQL query belongs to selection group, then we have to use the following method.

```
public ResultSet executeQuery() throws Exception
```

If the generalised SQL query belongs to updation group, then we have to use the following method

```
public int executeUpdate() throws SQLException
```

```
Eg: int rowCount=pst.executeUpdate();
```

```
20. package com.durgasoft;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp31 {

 public static void main(String[] args) throws Exception {
 Class.forName("com.mysql.jdbc.Driver");
 Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
 PreparedStatement pst=con.prepareStatement("insert into emp1 values(?, ?, ?, ?)");
 BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
 while(true){
 System.out.print("Employee Number :");
 int eno=Integer.parseInt(br.readLine());
 System.out.print("Employee Name :");
 String ename=br.readLine();
 System.out.print("Employee Salary :");
 float esal=Float.parseFloat(br.readLine());
 System.out.print("Employee Address :");
 String eaddr=br.readLine();

 pst.setInt(1, eno);
 pst.setString(2, ename);
 pst.setFloat(3, esal);
 pst.setString(4, eaddr);

 pst.executeUpdate();
 System.out.println("Employee Inserted Successfully");
 System.out.print("Onemore Employee [yes/no] :");
 }
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
String option=br.readLine();
if(option.equals("no")){
 break;
}
con.close();
}}
```

21.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp32 {

 public static void main(String[] args) throws Exception {
 Class.forName("com.mysql.jdbc.Driver");
 Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
 PreparedStatement pst=con.prepareStatement("update emp1 set esal=esal+? where esal<?");
 pst.setInt(1, 500);
 pst.setFloat(2, 10000.0f);
 int rowCount=pst.executeUpdate();
 System.out.println("Records Updated : "+rowCount);
 con.close();
 }
}
```

22.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class JdbcApp33 {
 public static void main(String[] args) throws Exception{
 Class.forName("com.mysql.jdbc.Driver");
 Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
 PreparedStatement pst=con.prepareStatement("select * from emp1 where esal<?");
 pst.setFloat(1, 10000.0f);
 ResultSet rs=pst.executeQuery();
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
System.out.println("ENO ENAME ESAL EADDR");
System.out.println("-----");
while(rs.next()){
 System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getFloat(3)+""
"+rs.getString(4));
}
con.close();
}

}
```

23.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Properties;

public class JdbcApp35 {
 public static void main(String[] args) throws Exception{
 Class.forName("com.mysql.jdbc.Driver");
 Properties p=new Properties();
 p.setProperty("user", "root");
 p.setProperty("password", "root");
 Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb",p);
 PreparedStatement pst=con.prepareStatement("insert into student values(?,?,?,?,?)");
 pst.setString(1, "S-111");
 pst.setString(2, "Durga");
 pst.setString(3, "Java");
 Date d=Date.valueOf("2015-01-25");
 pst.setDate(4, d);
 pst.executeUpdate();

 pst.setString(1, "S-222");
 pst.setString(2, "Anil");
 pst.setString(3, "Oracle");
 java.util.Date date=new java.util.Date();
 int val=date.getDate();
 java.sql.Date dt=new java.sql.Date(val);
 pst.setDate(4, dt);
 pst.executeUpdate();
 con.close();
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
}
```

```
}
```

## Batch Updations:

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 updations.

In batch updations, 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.

### Batch updations with Statement:

```
24.package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
public class JdbcApp30 {
 public static void main(String[] args) throws Exception {
 Class.forName("oracle.jdbc.OracleDriver");
 Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
 "system", "durga");
 Statement st = con.createStatement();
 st.addBatch("insert into emp1 values(666,'FFF',9000,'Hyd')");
 st.addBatch("update emp1 set esal=esal-500 where esal<10000");
 st.addBatch("delete from emp1 where eno=555");
 // st.addBatch("select * from emp1");--> java.sql.BatchUpdateException
 int[] rowCounts = st.executeBatch();
 for (int i = 0; i < rowCounts.length; i++) {
 System.out.println("Records Manipulated : " + rowCounts[i]);
 }
 st.close();
 con.close();
 }
}
```

## Batch Updations with PreparedStatement:

```
25.
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp34 {
 public static void main(String[] args) throws Exception {
 Class.forName("com.mysql.jdbc.Driver");
 Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb",
 "root", "root");
 PreparedStatement pst = con.prepareStatement("insert into emp1 values(?, ?, ?, ?)");
 pst.setInt(1, 666);
 pst.setString(2, "FFF");
 pst.setFloat(3, 6000);
 pst.setString(4, "Hyd");
 pst.addBatch();

 pst.setInt(1, 777);
 pst.setString(2, "GGG");
 pst.setFloat(3, 7000);
 pst.setString(4, "Hyd");
 pst.addBatch();
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
 pst.setInt(1,888);
 pst.setString(2, "HHH");
 pst.setFloat(3, 8000);
 pst.setString(4, "Hyd");
 pst.addBatch();
 int[] rowCounts=pst.executeBatch();
 for(int i=0;i<rowCounts.length;i++){
 System.out.println("Records Manipulated :"+rowCounts[i]);
 }
 con.close();
 }

}
```

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

## Transaction Management:-

Transaction:- Transaction is an unit of work performed by the front end application on back end system.

1. Deposit some amount in an account.
2. Withdraw some amount from an account.
3. Transfer some amount from one account to another account.

In database applications, every transaction must satisfy the following 4 properties.

1. Atomicity
2. Consistency
3. Isolation
4. Durability

### 1. Atomicity:-

In general we are able to perform multiple number of operations in a particular transaction, where performing all the operations or performing none of the operations is called as Atomicity property. If we perform all the operations successfully in a transaction then the state of the transaction should be success.

If we perform none of the operations in a transaction then state of the transaction should be failure.

Note: While performing operations in a transaction, if we encounter a problem with any operation then we should perform rollback operation over all the operations which are available in the transactions.

DURGASOFT

JDBC

MR.NAGOORBABU

## 2. Consistency:

In database applications, before the transaction and after the transaction the state of the database must be stable is called as Consistency property.

To achieve consistency we will perform some checkings called as Consistency Checkings.

Consistency checkings will check correctness of the results after the transactions.

## 3. Isolation:

In database applications, if we perform more than one transaction on a single data item then that transactions are called as Concurrent Transactions. In concurrent transactions, one transaction execution should not be effect to the another transaction. This nature of the transaction is called as Isolation.

While performing concurrent transactions, there may be a chance to get concurrency problems, to resolve these problems we will use Isolation Levels in database application.

## 4. Durability:

In database applications, after committing the transaction if we have any catastrophic failures like power failure, operating system crashing and so on then we have to preserve the modifications which we performed on the database during respective transaction. This nature of the transaction is called as Durability.

In Jdbc applications, when we establish connection then automatically that connection will have a default mode i.e. auto commit mode. In case of auto commit mode, when we submit SQL query to the connection, where connection will carry 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 permanently.

The above auto commit nature of connection may not satisfy the transaction's atomicity property. To preserve transaction atomicity property in Jdbc applications we have to change connection's auto commit mode.

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
```

DURGASOFT

JDBC

MR.NAGOORBABU

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.

```
26.
package com.durgasoft;
import java.sql.Connection;
import
java.sql.DriverManager;
import java.sql.Statement;

public class JdbcApp40 {

 public static void main(String[] args) {
 Connection con=null;
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 con=DriverManager.getConnection("jdbc:oracle:thin:
@localhost:1521:xe","system", "durga");
 con.setAutoCommit(false);
 Statement st=con.createStatement();
 st.executeUpdate("insert into student values(111,'AAA',78)");
 st.executeUpdate("insert into student values(222,'BBB',87)");
 st.executeUpdate("insert into student values(333,'CCC',96)");
 con.commit();
 System.out.println("Transaction Success");
 } catch (Exception e) {
 try{
 con.rollback();
 System.out.println("Transaction Failure");
 }catch(Exception e1){
 e1.printStackTrace();
 }
 //e.printStackTrace();
 }
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
27.
package com.durgasoft;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class JdbcApp41 {

 public static void main(String[] args) {
 Connection oracle_conn=null;
 Connection mysql_conn=null;
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 Class.forName("com.mysql.jdbc.Driver");
 oracle_conn=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
 mysql_conn=DriverManager.getConnection
 ("jdbc:mysql://localhost:3306/durgadb","root","root");

 oracle_conn.setAutoCommit(false);
 mysql_conn.setAutoCommit(false);

 Statement oracle_st=oracle_conn.createStatement();
 Statement mysql_st=mysql_conn.createStatement();

 BufferedReader br=new BufferedReader
 (new InputStreamReader(System.in));
 System.out.print("Source Account :");
 String source_Account=br.readLine();
 System.out.print("Target Account :");
 String target_Account=br.readLine();
 System.out.print("Amount To Transfer :");
 int trans_Amt=Integer.parseInt(br.readLine());

 int oracleRowCount=oracle_st.executeUpdate
 ("update account set balance=balance-"+trans_Amt+" where
 accNo='"+source_Account+"'");
 int mysqlRowCount=mysql_st.executeUpdate("update account set
 balance=balance+"+trans_Amt+" where accNo='"+target_Account+"'");
 if((oracleRowCount==1) && (mysqlRowCount==1)){
 oracle_conn.commit();
 mysql_conn.commit();
 System.out.println(trans_Amt+" Transferred Successfully from
 "+source_Account+" to "+target_Account);
 System.out.println("Transaction Success");
 }
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

```
 }else{
 oracle_conn.rollback();
 mysql_conn.rollback();
 System.out.println("Transaction Failure");
 }
 } catch (Exception e) {
 try {
 oracle_conn.rollback();
 mysql_conn.rollback();
 System.out.println("Transaction Failure");
 } catch (Exception e2) {
 e2.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.

```
28.
package com.durgasoft;
import java.sql.Connection;
import
java.sql.DriverManager;
import java.sql.Savepoint;
import java.sql.Statement;

public class JdbcApp42 {

 public static void main(String[] args) {
 Connection con=null;
 try {
 Class.forName("oracle.jdbc.OracleDriver");
 con=DriverManager.getConnection
 ("jdbc:oracle:thin:@localhost:1521:xe","system", "durga");
 con.setAutoCommit(false);
 Statement st=con.createStatement();
 st.executeUpdate("insert into student values(111,'AAA',76)");
 Savepoint sp=con.setSavepoint();
 st.executeUpdate("insert into student values(222,'BBB',88)");
 con.rollback(sp);
 st.executeUpdate("insert into student values(333,'CCC',99)");
 con.commit();
 System.out.println("Transaction Success");
 } catch (Exception e) {
 try {
 con.rollback();
 System.out.println("Transaction Failure");
 } catch (Exception e2) {
 e2.printStackTrace();
 }
 }
 }
}
```

## Stored Procedures And Functions:

What is the difference between Stored procedures and functions?

Ans: Stored procedure is a block of instructions defined at database to represent a particular action.  
Stored procedures will not use return statement to return a value.

Syntax: create or replace procedure procedure\_name([param-list])  
as

DURGASOFT

JDBC

MR.NAGOORBABU

```

----- Global declarations

BEGIN

----- Database logic

END procedure_name;
/ (press enter to save and compile the procedure)
```

Stored function is a block of instructions defined at database to represent a particular action. Stored functions will use return statement to return a value.

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 Jdbc applications, to access stored procedures and functions defined at database from Java application then we have to use CallableStatement object.

To represent CallableStatement object Jdbc API has provided the interface in the form of java.sql.CallableStatement.

If we want to use CallableStatement object in Jdbc applications then we have to use the following steps.

#### **Step 1: Get CallableStatement object.**

To create CallableStatement object in Jdbc applications we have to use the following method from Connection.

```
public CallableStatement prepareCall(String pro_cal) throws SQLException
```

```
Ex: CallableStatement cst=con.prepareCall("{call getSal(?,?)}");
```

DURGASOFT

JDBC

MR.NAGOORBABU

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.

Note: In case of stored procedures and functions we are able to pass the parameters in the following 3 ways.

## 1. IN Type Parameter:

This parameter will get the value from procedure call or function call and make available to the procedure body or function body.

Syntax: var\_name IN data\_type

## 2. OUT Type Parameter:

This parameter will get the value from procedure body or function body and send that value to the respective procedure call or function call.

Syntax: var\_name OUT data\_type

## 3. INOUT Type Parameter:

This parameter is acting as both IN type and OUT type parameters.

Syntax: var\_name INOUT data\_type

**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.

DURGASOFT

JDBC

MR.NAGOORBABU

```
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 eno=id;
END getSal;
/
```

29.

```
import java.sql.*;
public class JdbcApp34
{
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");
CallableStatement cst=con.prepareCall("{call getSal(?,?)}");
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
cst.setInt(1,101);
cst.registerOutParameter(2, Types.FLOAT);
cst.execute();
System.out.println("Salary....."+cst.getFloat(2));
con.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 eno=id1;
select esal into sal2 from emp where eno=id2;
return (sal1+sal2)/2;
END getAvg;
/
```

Ex:-

```
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");
CallableStatement cst=con.prepareCall("{?=call getAvg(?,?)}");
cst.setInt(2,888);
cst.setInt(3,6666);
cst.registerOutParameter(1, Types.FLOAT);
cst.execute();
System.out.println("Average Salary....."+cst.getFloat(1));
con.close();
}
}

/*
create or replace procedure getEmps(sal IN number, emps OUT SYS_REFCURSOR)
AS
BEGIN
open emps for
select * from emp1 where esal<sal;
END getEmps;
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
/
*/
package com.durgasoft;

import java.io.FileInputStream;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.Properties;

import oracle.jdbc.internal.OracleTypes;

public class JdbcApp38 {

 public static void main(String[] args) throws Exception {
 FileInputStream fis = new FileInputStream("db.properties");
 Properties p = new Properties();
 p.load(fis);
 String driver_class = p.getProperty("driver_class");
 String driver_url = p.getProperty("driver_url");
 Class.forName(driver_class);
 Connection con = DriverManager.getConnection(driver_url, p);
 CallableStatement cst = con.prepareCall("{call getEmps(?,?)}");
 cst.setFloat(1, 10000);
 cst.registerOutParameter(2, OracleTypes.CURSOR);
 cst.execute();
 Object obj = cst.getObject(2);
 ResultSet rs = (ResultSet) obj;
 System.out.println("ENO\tENAME\tESAL\tEADDR");
 System.out.println("-----");
 while (rs.next()) {

 System.out.println(rs.getInt(1) + "\t" + rs.getString(2) + "\t" + rs.getFloat(3) + "\t" + rs.getString(4));
 }
 con.close();
 }
}
```

**db.properties**

---

```
driver_class=oracle.jdbc.OracleDriver
driver_url=jdbc:oracle:thin:@localhost:1521:xe
user=system
password=durga
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
/*
create or replace function getEmployees(no1 IN number,no2 IN number) return SYS_REFCURSOR
AS
employees SYS_REFCURSOR;
BEGIN
open employees for
 select * from emp1 where eno>=no1 and eno<=no2;
return employees;
END getEmployees;
/
*/
package com.durgasoft;

import java.io.FileInputStream;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.Properties;

import oracle.jdbc.internal.OracleTypes;

public class JdbcApp39 {

 public static void main(String[] args)throws Exception {
 FileInputStream fis=new FileInputStream("db.properties");
 Properties p=new Properties();
 p.load(fis);
 String driver_class=p.getProperty("driver_class");
 String driver_url=p.getProperty("driver_url");
 Class.forName(driver_class);
 Connection con=DriverManager.getConnection(driver_url, p);
 CallableStatement cst=con.prepareCall("{?=call getEmployees(?,?)}");
 cst.setInt(2, 111);
 cst.setInt(3, 555);
 cst.registerOutParameter(1, OracleTypes.CURSOR);
 cst.execute();
 ResultSet rs=(ResultSet)cst.getObject(1);
 System.out.println("ENO\tENAME\tESAL\tEADDR");
 System.out.println("-----");
 while(rs.next()){

 System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
 }
 con.close();
 }
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
db.properties
driver_class=oracle.jdbc.OracleDriver
driver_url=jdbc:oracle:thin:@localhost:1521:xe
user=system
password=durga
```

## 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.

DURGASOFT

JDBC

MR.NAGOORBABU

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();

```
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();
```

DURGASOFT

JDBC

MR.NAGOORBABU

```

ds.setURL("jdbc:oracle:thin:@localhost:1521:xe");
ds.setUser("system");
ds.setPassword("durga");
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.

## BLOB and CLOB:

---

**BLOB (Binary Large Object):**

Up to now in Jdbc applications, we are able to interact with the database in order to insert a record, retrieve a record and so on with the varchar data or number data and so on.

As per the application requirement if we want to insert an image or a document in the database table then Oracle provided datatypes numbers, varchar are not sufficient, we have to use BLOB and CLOB datatypes provided by Oracle.

The main purpose of the BLOB and CLOB datatypes is to represent large volumes of binary data and large volumes of character data in a database table.

To insert large volumes of binary data (an image) on to the database table we have to use the following steps.

**Step 1: Prepare a table at database with blob data type.**

Ex: create table emp\_details(eno number, image blob);

**Step 2: Represent an image file in the form of File class object.**

Ex: File f=File("Desert.jpg");

DURGASOFT

JDBC

MR.NAGOORBABU

**Step 3: Get File class object content in the form of FileInputStream.**

Ex: FileInputStream fis=new FileInputStream(f);

**Step 4: Create preparedStatement object with insert SQL query format.**

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

**Step 5: Set BinaryStream to the blob type positional parameter in PreparedStatement.**

To set a BinaryStream with the blob type positional parameter we have to use the following method from PreparedStatement.

```
public void setBinaryStream(int param_index, InputStream is, int length);
```

Ex: pst.setBinaryStream(2, fis, (int)f.length());

**Step 6: Execute PreparedStatement.**

Ex: pst.executeUpdate();

```
import java.sql.*;
import java.io.*;
public class BLOBDemo
{
 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");
 File f=new File("Desert.jpeg");
 FileInputStream fis=new FileInputStream(f);
 PreparedStatement pst=con.prepareStatement("insert into employee values(?,?)");
 pst.setInt(1,105);
 pst.setBinaryStream(2,fis,(int)f.length());
 pst.executeUpdate();
 System.out.println("Employee Image inserted Successfully");
 con.close();
 }
}
```

Steps to retrieve blob data from database table to Jdbc application:

**Step 1: Prepare ResultSet object with blob data.**

Ex: ResultSet rs=st.executeQuery("select \* from emp\_details");

**Step 2: Read normal data from ResultSet object.**

DURGASOFT

JDBC

MR.NAGOORBABU

```
Ex: rs.next();
```

```
int eno=rs.getInt(1);
```

**Step 3: Get BinaryStream from blob datatype available at ResultSet object**

To get a BinaryStream from blob type parameter available at ResultSet object we have to use the following method.

```
public InputSteram getBinaryStream(int param_index)
```

```
Ex: InputSteram is=rs.getBinaryStream(2);
```

**Step 4: Prepare the target resource to hold up the retrieved blob data by using FileOutputStream.**

```
Ex: FileOutputStream fos=new FileOutputStream("myimage.jpeg");
```

**Step 5: Read bit by bit from InputStream and write the same bit by bit on FileOutputStream to store the retrieved data on target file.**

```
Ex: int i=read();
```

```
while(i!=-1) {
 fos.write(i);
 i=is.read();
}
```

```
import java.sql.*;
import java.io.*;
public class BLOBDemo1
{
 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","durga"
);
 Statement st=con.createStatement();
 ResultSet rs=st.executeQuery("select * from employee");
 rs.next();
 System.out.println("Employee Id..... "+rs.getInt(1));
 InputStream is=rs.getBinaryStream(2);
 FileOutputStream fos=new FileOutputStream("myimage.jpg");
 int i=is.read();
```

DURGASOFT

JDBC

MR.NAGOORBABU

```
while (i != -1)
{
fos.write(i);
i=is.read();
}
System.out.println("Image created Successfully with the name
myimage.jpeg");
fos.close();
con.close();
}
}
```

## CLOB (Character Large Object):

clob is a datatype provided by Oracle, it can be used to represent large values of character data like documents, pdf files and so on.

If we want to perform operations with clob datatype then we have to use the same steps what we have used with blob datatype, but we need to provide the following replacements.

```
import java.sql.*;
import java.io.*;
public class CLOBDemo
{
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","durga"
);
PreparedStatement pst=con.prepareStatement("insert into webinfo
values(?,?)");
pst.setString(1, "app");
File f=new File("web.xml");
FileReader fr=new FileReader(f);
pst.setCharacterStream(2, fr, (int)f.length());
pst.executeUpdate();
System.out.println("web application stored in database successfully");
fr.close();
con.close();
}
}
```

DURGASOFT

JDBC

MR.NAGOORBABU

Ex:-

```
import java.sql.*;
import java.io.*;
public class CLOBDemo1
{
 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","durga"
);
 Statement st=con.createStatement();
 ResultSet rs=st.executeQuery("select * from webinfo");
 rs.next();
 System.out.println("Application name is....."+rs.getString(1));
 FileWriter fw=new FileWriter("myweb.xml");
 Reader r=rs.getCharacterStream(2);
 int i=r.read();
 while (i != -1)
 {
 fw.write(i);
 i=r.read();
 }
 System.out.println("web.xml is retrieved successfully with the name
 myweb.xml");
 fw.close();
 con.close();
 }
}
```

## JDBC INTERVIEW QUESTIONS:

1. What is the difference between Database and Database management system?
2. How a query could be executed when we send a query to Database?
3. What is Driver? How many Drivers are available in JDBC? What are the types?
4. What is JDBC and What are the steps to write a JDBC application?
5. How to load a JDBC driver?
6. How to establish a Database connection between java application and Database?
7. Basically Connection is an interface, how getConnection() will create an object for Connection interface?
8. What is the requirement to use Statement object?
9. How to execute SQL Queries from a java application?
10. What are the differences between executeQuery(...), executeUpdate(...) and execute(...) methods?
11. How to create a table dynamically from a jdbc application?.
12. How to insert records into a table from a JDBC application?

DURGASOFT

JDBC

MR.NAGOORBABU

13. How to update a table from a jdbc application?.
14. How to delete records from a table from jdbc application?.
- 15.What is ment by ResultSet object and How to Fetch the Data from Database?.
- 16.In general execute() method can be used to execute selection group SQL queries for getting the data from Database , but execute() return a boolean value true so here how it possible to fetch the data from database?
- 17.In general execute() method can be used to execute updatation group SQL queries for updating the data on Database , but execute() return a boolean value false so here how it possible to get the records updated count value(int value)?
18. If we use selection group SQL query to executeUpdate() ,what happened?
19. If we use updatation group SQL query to executeQuery() ,what happened?
20. What is ment by ResultSet and What are the types of ResultSets are available in JDBC application?
21. What is the difference between ScrollSensitive ResultSet and ScrollInsensitive ResultSets?
22. What is the default ResultSet type in JDBC application and How it is possible to create a specific type of ResultSet object?
23. How to iterate the data from Scrollable ResultSet object in both forward and backward direction?
24. How to generate ScrollSensitive Result Set and how to reflect the later updations from database automatically to the ResultSet object?
25. How to insert records into Database throws Updatable ResultSet?
26. How to perform updations on Database throws Updatable ResultSet?
27. What is meant by ResultSetMetaData ?How to get The ResultSet metadata of a ResultSet object?
28. How to display the data with the respective field names
29. What are the differences between Statement and PreparedStatement? (or) Tell me the situations where we should go for PreparedStatement over Statement object.
30. How to insert number of records into a table through Prepared Statement object.
31. How to update the database through PreparedStatement object.
32. How to fetch the data from database through PreparedStatement object.
33. What is meant by Transaction? How it is possible to maintain Transactions in JDBC applications?
34. What is meant by SavePoint?How to use Savepoints in JDBC applications?

**Adv. Java means DURGA SIR..**

# **ADV.JAVA With SCWCD / OCWCD Servlets Material**

## **1. Servlet Technology Model**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

**Ex. IBM Employee**

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

**DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

## Servlet Technology Model

### Agenda:

#### 1) Servlet API and life cycle

- javax.servlet Package :
  - Interfaces :(14)
  - Classes : (9)
  - Exceptions : (2)
- javax.servlet.Servlet interface :
- load-on-startup
- Life Cycle of the Servlet that implements Servlet interface :
- GenericServlet(AC):
- javax.servlet.http package :
  - Interfaces :(8)
  - Classes :(7)
- Structure of HttpServletRequest :
- Structure of HttpServletResponse :

#### 2) Http Methods

- GET
- POST
- HEAD
- OPTIONS
- PUT
- DELETE
- TRACE
- HttpServlet
- Life Cycle of HttpServletRequest

#### 3) HttpServletRequest

- To retrieve request parameters
- To retrieve request headers
- To retrieve request cookies
- Retrieving client & Server information from the request

#### 4) HttpServletResponse

- To set response headers
- To set ContentType of response
- To get Text Stream for response
- To get Binary stream for response
- To perform redirecting
- To add Cookies to the response
- Differences between send-Redirection and forward mechanism :

**www.durgasoftonlinetraining.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

 **USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

## Introduction :

### 2-tier:

known and fixed number of clients we can access 2-tier.

**Web Application:** collection of web resources.

**Web Resource:** each web resource is capable of generate one web page.

(Ex: html , servlets, Jsp's , java script )

### Web resources :

2 types

- static
- Dynamic

A web page whose content is fixed i.e., static web page

Ex: html , js , images

A web page whose content will be changed dynamically based on time of request generate input values of the request.

Ex:Servlets, JSP's , Server side java script(ssjs) , PHP

Based on the place client side web resources program executed.(in browser)

Based on the place server side web resources program executed but not resides .(in server side)

## WEB APPLICATION REQUIREMENTS :

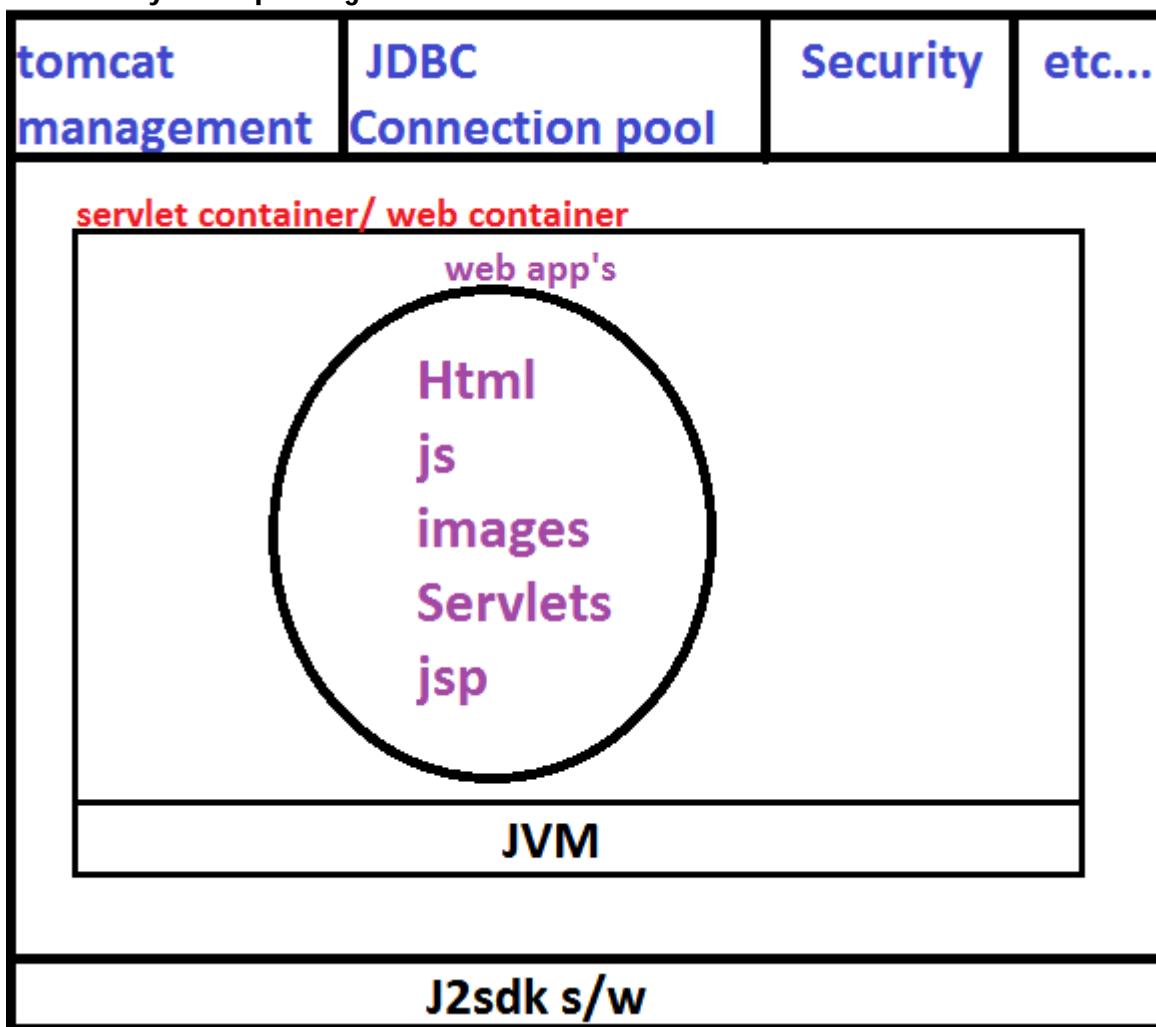
1. Browser software.
2. Technology to develop client side.
3. Technology to develop server side.
4. User software.
5. DataBase software.

### WEB Server Software :

Web server software is special piece of s/w (or) special s/w which can manages the web applications and web resources we executes this web resources automatically or dynamically



when ever your requesting .



#### Responsibilities of web server :

1. It collects all client http requests.
2. It passes to appropriate web resources of web application.
3. Web server proceed built middle ware software.
4. container built in controller software.
5. Gives resultant out put.

#### Responsibilities of Servlet Container :

1. To provide the environment to manage web resource of Web Application.
2. It Perform total life cycle operations.
3. In case of JSP to generate corresponding Generated Servlet.

J2SDK software is instalable.

(through cmd prompt )

J2EE is not a instalable software , It is a specification,(not through cmd prompt )

Specification is a document , it contain set of RULES(interfaces) and GUIDELINES(classes).

SERVLET ia not a technology , It is a specification.

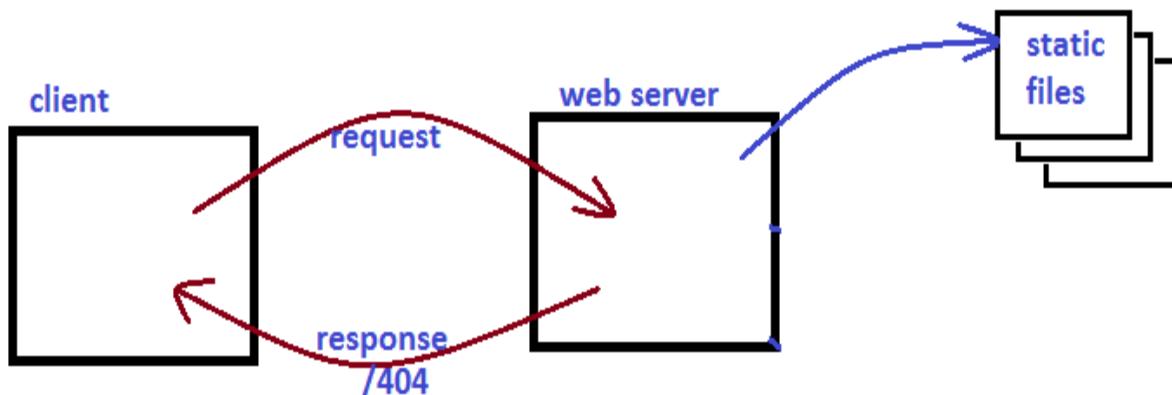
Specification related package consists of more number of INTERFACES and less number of CLASSES.

RULES --->interfaces(method declarations)

GUIDELINES---> classes(concrete methods )

In servlets we use the instance variables are not Thread Safe.

## Web Programming for Static information :



Client sends a request for a static file.

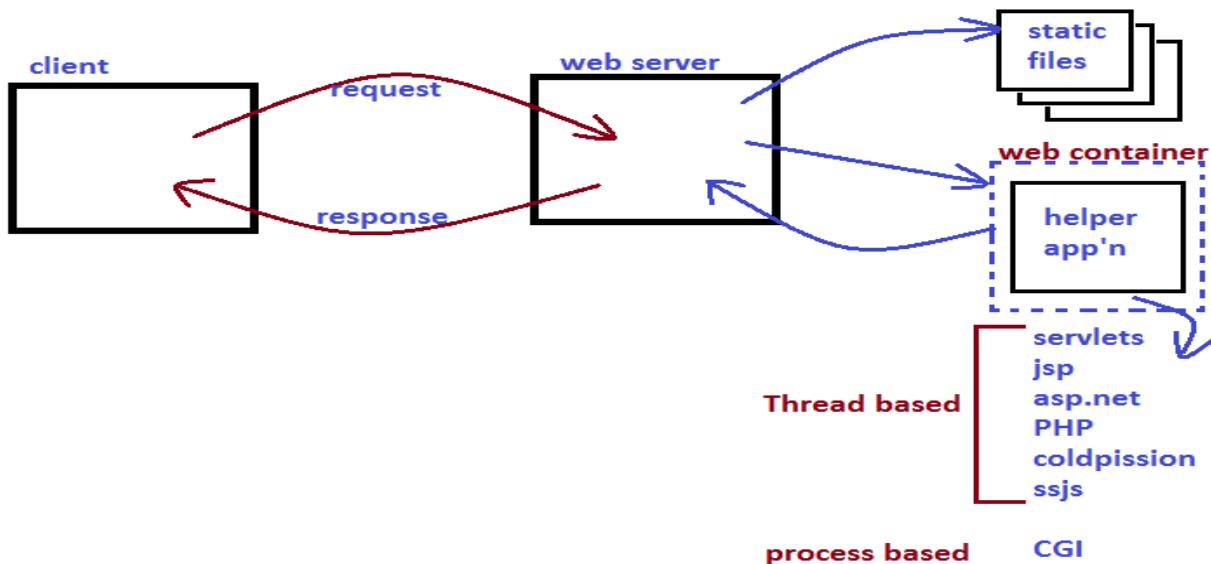
Web server searches whether the requested static file is available or not at Server side.

If the requested resource is available then it will return that static file as response.

If it is not available ,then web server sends 404 status code saying RequestResource is not available.

To serve static file, no processing is required at server side .Hence webserver always loves to serve static information.

## Web Programming for Dynamic information :



**Client sends a request to the Web server.**

**Web server checks whether the request is for static or Dynamic information.**

**If the request is for static information , web server searches for the required Static file.**

**If it is available it returns that file , other wise it returns 404 status code.**

**If the request is for Dynamic information web server forwards the request to some Helper Application.**

**Helper Application analyzes and process the request and generate required dynamic information.**

**Helper Application forwards that response to the web server and web server forwards that response to the client.**

**The following are various possible Helper Applications at Server side.**

- 1.Servlet
- 2.Jsp
- 3.Asp.net
- 4.PHP
- 5.cold fusion
- 6.CGI
- 7.server side Java Script.

**A servlet is a Server side web component managed by web container for generation of**

dynamic information.

Web container is the best assistant to the programmer . It maintains entire life cycle of the servlet.

So that programmer has to concentrate only on Business logic.The remaining operations Instantiation of Servlet , Executing life cycle, Destroying the Servlet object and etc, taken care by Web container.

### **TYPES OF WEB CONTAINER:**

There are 3 types of web containers are possible.

1. **Stand alone :**

Both web server and web container are available in a single integrated component , such type of web container are called Stand-alone web container.

Ex : Tomcat

This type of web containers are best-suitable for small scale applications and rarely used.

2. **In-Process Web container :**

If the web container runs in same address space(same machine) of web server and it is available as plug-in such type of web containers are called In-Process web container.

In this case both web server and web container need not be from the same vendor.

3. **Out-Process Web container:**

Web server and web container both are running on different machines , Web container is attached to the web server externally.Such type of web containers are called Out-process web containers.

we can configure front-end apache has to forward the request to the back-end weglogic server. These type of web containers are industry using.

This type of web Containers are most commonly used web containers.

LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...

**JAVA MEANS DURGASOFT**

INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE

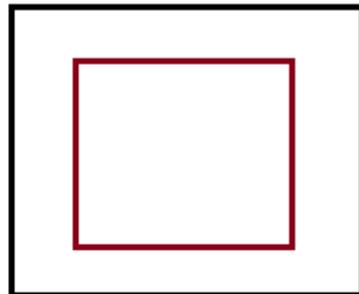
AN ISO 9001:2008 CERTIFIED

**DURGA**  
SOFTWARE SOLUTIONS

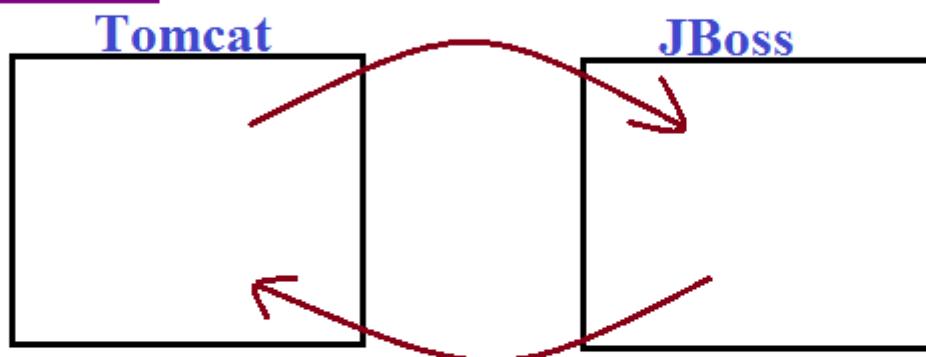
#202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

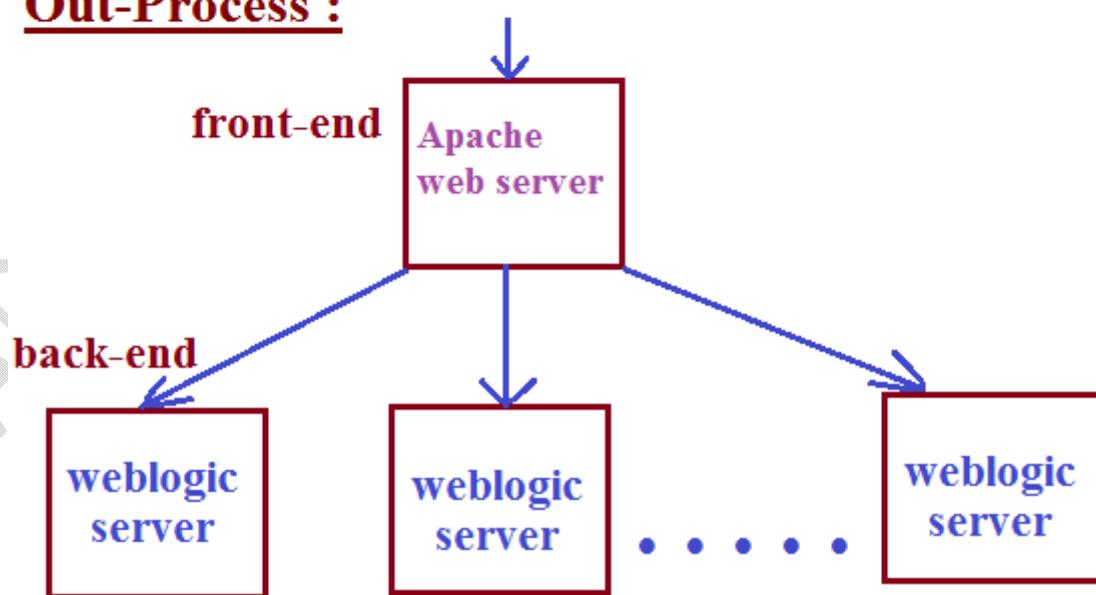
### Stand alone :



### In-Process :



### Out-Process :



## Differences between CGI and Servlets :

| <u>CGI</u>                                                                                                                                                                                          | <u>SERVLETS</u>                                                                                                                                                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| It is Process based i.e, for every request a separate process will be created and it is responsible to generate required response.                                                                  | It is thread-based i.e, for every request a new thread will be created and it is responsible to process the request.                                                                                    |
| Creation and Destruction of process for every request is costly. If the number of requests increase it will effects performance of the system.Hence CGI technology fail to the destroy of scalable. | Creation and Destruction of new thread for every request is not costly . Hence there is no effect on performance even though number of requests increases due to this it succeeds to delevary scalable. |
| Two process never share same address space(memory).Hence there is no chance of concurrency, inconsistency problems and syncronization is not required.                                              | All threads share common address space hence there may be a chance of concurrency, inconsistency problems.                                                                                              |
| CGI programs can be written in multiple languages.But most commonly used language is PERL.                                                                                                          | Servlets can be written only in Java.                                                                                                                                                                   |
| Most of the CGI languages are not object oriented.Hence we are missing benefits of OOPs.                                                                                                            | Java language itself is Object-Oriented . Hence we can get all benefits of OOPs.                                                                                                                        |
| CGI technology is platform dependent.                                                                                                                                                               | Servlet technology is platform independent.                                                                                                                                                             |

### **FAST CGI :**

It improves performance when compared with traditional CGI. In this case web container maintains a pool of process so that a single process can serve multiple requests one by one.

## Servlet Technology Model :

1. servlet API and life cycle
2. Http methods
3. HttpServletRequest
4. HttpServletResponse

**Servlet defination :** Servlet is a single instance multiple thread based server side Technology to develop Dynamic web resources of web application.

## Servlet API :

We can develop Servlets by using the following 2 packages.

1. **javax.servlet:** This package defines several classes and interfaces to develop servlets from scratch irrespective of any protocol.
2. **javax.servlet.http :** It is the sub package of javax.servlet and contains several convenient classes and interfaces to develop http based servlets.

## javax.servlet Package :

### Interfaces of javax.servlet Package :(14)

- 1) **Servlet :**  
Every servlet in java should implement servlet interface either directly or indirectly. ie, Servlet interface acts as a root interface for all java Servlets.  
This interface defines the most common methods (including life cycle methods) which are applicable for any servlet object.
- 2) **ServletRequest :**  
ServletRequest object can be used to hold client data.  
ServletRequest interface defines several methods to acts as end users provided data from the request object.  
**Ex:** getParameter()
- 3) **ServletResponse :**  
ServletResponse object can be used to prepare and send responds to the client.  
ServletResponse interface defines several methods which are required for preparation of

response.

Ex: getWriter(), setContentType()

4) **ServletConfig :**

For every servlet , web container creates one ServletConfig object to hold its configuration information like logical name of the Servlet , instantiation parameters etc.

ServletConfig interface defines several methods to access servlets configuration information.

Ex: getServletName()  
getInitParameter()

5) **ServletContext :**

For every web application, web container will create one ServletContext object to hold application level configuration information like name of the app'n and ServletContext parameter etc.

*Note : ServletConfig is per Servlet , where as ServletContext is per web application.*

6) **RequestDispatcher :**

By using RequestDispatcher object we can dispatch the request from one component to another component.

This interface defines two methods.

- 1. forward()
- 2.include()

7) **SingleThreadModel :**

- Servlet technology is single instance multi threaded model. i.e, multi threads can operate simultaneously on the servlet object . Hence there may be a chance of data inconsistency problems.
- we can resolve these problems by using SingleThreadModel interface. If a servlet implements SingleThreadModel interface, then a single thread can access servlet object at a time i.e, Servlet can process only one request at a time. i.e, the main objective of SingleThreadModel interface is to provide thread-safety.
- But the problem with SingleThreadModel is Servlet can process only one request at a time which impacts performance of the system. Because of this problem SingleThreadModel interface is not recommended to use and it is deprecated in Servlet 1.3 version without introducing any replacement.
- we can provide thread-safety to the Servlet by using synchronized keyword.
- SingleThreadModel interface doesn't contain any methods . It is a marker interface.
- **Note:** In addition to above interfaces javax.servlet package defines the following interfaces also .

8) **Filter**

9) **FilterConfig**

10) **FilterChain**

----- to implements Filter Concept.

11) **ServletRequestListener**

12) **ServletRequestAttributeListener**

13) **ServletContextListener**

**14) ServletContextAttributeListener**

----- to implements Listener Concept.

**Classes of javax.servlet package : (9)****1) GenericServlet :**

This class implement Servlet interface and provides default implementation for every method except service(). Hence it is an abstract class.

we can use this class as a base class to develope protocol independent servlets.

**2) ServletOutputStream :**

we can use this class object to send binary data( pdf files , image files , video/audio files etc ) as response to the client.

**3) ServletInputStream:**

we can use ServletInputStream objects to read binary data send by the client.

Note:

In addition to above classes javax.servlet package defines the following classes also

**4) ServletRequestWrapper****5) ServletResponseWrapper**

-----to implement wrapper concept.

**6) ServletRequestEvent****7) ServletRequestAttributeEvent****8) ServletContextEvent****9) ServletContextAttributeEvent**

-----to define Events for Listeners.

All Events are classes and Listeners are interfaces

**Exceptions of javax.servlet package :**

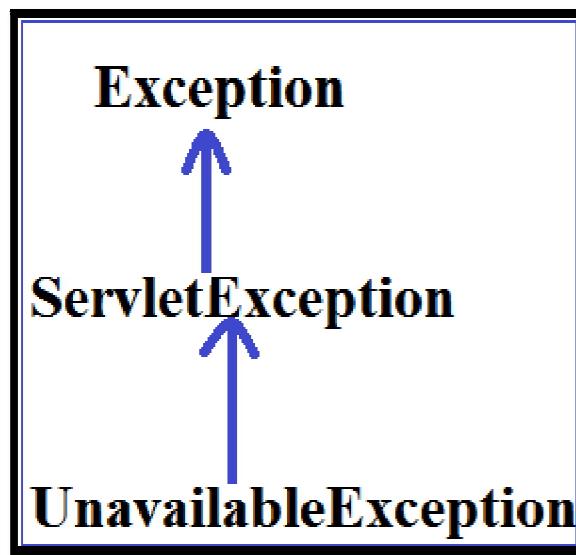
javax.servlet package defines the following 2 exceptions.

**1. ServletException.**

Servlet can throw this exception whenever it faces any difficulty while processing client request.

**2. UnavailableException :**

It is the child class of ServletException and it is deprecated .

Note :

In total javax.servlet package defines

14 - Interfaces

9--Classes

2--Exceptions

**javax.servlet.Servlet interface :**

Every servlet in java should compulsary implement Servlet interface either directly or indirectly i.e, Servlet interface acts as a root interface for all servlets.

This interface defines the most common methods which can be applicable for any Servlet object.

The following are the methods defined in Servlet interface.

1. init()
2. service()
3. destroy()
4. getServletConfig()
5. getServletInfo()

**1. init():**

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

This method will be executed only once by the web container immediately after Servlet instantiation to perform initialization activities .

Web container won't place Servlet object into service() method in the following cases :

1. If init() throws ServletException.
2. If init() doesn't return with in the time period specified by web container.

In the above cases web container makes that servlet object eligible for Garbage Collection without calling any other life cycle method .

In this case web container creates a new Servlet object to provide service.

## 2. service():

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

web container calls this method for every request to provide response.

Entire servicing logic/business logic , we have to define in this method only.

## 3. destroy():

```
public void destroy();
```

This method will be executed only once by the web container to perform clean up activities , when ever web container takes servlet object from out of service. This is usually happens at the time of application undeployed (or) at the time of server shut down (or) Web Container requires some free memory .

when ever web container calls destroy() method , it may not be executed immediately. It will wait until completing all currently executing threads.

**NOTE :** init() , service() , destroy() are called life cycle methods of Servlet.  
we can call destroy() explicitly from the init() and service() , in this case destroy() will be executed just like a normal method call and servlet object won't be destroyed .

#### 4. getServletConfig :

```
public ServletConfig getServletConfig();
```

This method can be used to get ServletConfig object .  
By using this config object Servlet can get its configuration information.

#### 5. getServletInfo () :

```
public void getServletInfo()
```

This method returns information about Servlet like Author, version , copy right information etc.

---

Demo program the develope a servlet by implementing Servlet interface

#### FirstServlet.java

```
1) package com.jobs4times;
2) import java.io.IOException;
3) import java.io.PrintWriter;
4) import javax.servlet.Servlet;
5) import javax.servlet.ServletConfig;
6) import javax.servlet.ServletException;
7) import javax.servlet.ServletRequest;
8) import javax.servlet.ServletResponse;
9)
10) public class FirstServlet implements Servlet {
11)
12) static {
13) System.out.println("Servlet class loading");
14) }
15) public FirstServlet () {
```

```

16) System.out.println("servlet instantiation");
17)
18)
19) ServletConfig config;
20) public void init(ServletConfig config) throws ServletException {
21) this.config=config;
22) System.out.println(" we are in init() method ");
23) }
24) public void service(ServletRequest request, ServletResponse response)
25) throws ServletException, IOException {
26) System.out.println("We are in service() method ");
27) PrintWriter out=response.getWriter();
28) out.println("welcome to SCWCD");
29) }
30) public void destroy() {
31) System.out.println("We are in destroy() method ");
32) }
33) public ServletConfig getServletConfig() {
34) return null ;
35) }
36) public String getServletInfo() {
37) return "Written by Jobs4Times ";
38) }
39)

```

web.xml

```

1) <web-app>
2) <servlet>
3) <servlet-name>first</servlet-name>
4) <servlet-class>com.jobs4times.FirstServlet</servlet-class>
5) <load-on-startup>5</load-on-startup>
6) </servlet>
7) <servlet-mapping>
8) <servlet-name>first</servlet-name>
9) <url-pattern>/fs</url-pattern>
10) </servlet-mapping>
11) </web-app>

```

<http://localhost:8080/SCWCD1A/fs>

- When ever we are sending the request ,webserver checks whether this request is for static or dynamic information by using URL pattern.

2. If the request is for static information , Webserver searches for the required static file and provides required response.
3. If the request is for dynamic information , Webserver forwards that request to webcontainer .  
The webcontainer identifies the corresponding servlet class by using web.xml
4. Webcontainer loads that .class file , perform instantiation , and execute init() and service() methods and provide required response to the webserver
5. webserver inturn forwards that response to end-user.

**Without <load-on-startup> :**

First Request:

1. loading .class file (/ servlet loading)
2. Instantiation
3. init()
4. service()

Second Request: service()

**With <load-on-startup> :**

At server startup/Application deployment

1. loading .class file (/ servlet loading)
2. Instantiation
3. init()

First Request : service()

Second Request : service()

**Note:**

The allowed values for <load-on-startup> tag is an integer i.e., +ve , zero , -ve

In the case of <load-on-startup> in deployment descriptor

- Highest value to give the least priority
- Lowest value give high Priority
- Zero will give last priority
- -ve value means ignore the <load-on-startup> concept.

If we give the two servlets having the same <load-on-startup> value we can't expect execution order or behaviour .

***What is the advantage of <load-on-startup> :***

We can equalize the response time of first request and remaining requests.

The main disadvantage of <load-on-startup> is it increases server start up time without any specific requirement don't configure this <load-on-startup> on servlet.

***Note:***

From servlet 2.5v onwards a single servlet can be mapped with multiple <url-pattern>tags i.e., we can take multiple <url-pattern> tags with in single <servlet-mapping> tag.

Ex:

- 1) <servlet-mapping>
- 2)     <url-pattern>/test </url-pattern>
- 3)     <url-pattern>/test </url-pattern>
- 4) </servlet-mapping>

***Note :***

Whenever we are writing 2 servlets having same url-pattern , if we are sending a request to particular url-pattern

The order of evaluation of web.xml is decided by the underling webserver , there is no specification rules

- In the case of Tomcat the order of evaluation of web.xml is Bottom to Top.
- In the case of Weblogic the order of evaluation of web.xml is Top to Bottom.

**Life Cycle of the Servlet that implements Servlet interface :**

1. Servlet class loading by class loader.(It is part of Jvm)
2. Servlet Instantiation by webcontainer , For this web container always calls public-no-argument constructor . Hence Every servlet should compulsary contain public-no-argument constructor.Otherwise we will get RuntimeException saying java.lang.InstantiationException exception .
3. Execution of init() by webcontainer  
***Note:*** The above 3 steps will be performed Generally at the time of first request.If <load-on-startup> is configured these will be executed at the time of either server startup or at the time of application deployment.
4. Execution of service() by web-container.
5. Execution of destroy()

***Note:***If we are invoking destroy() explicitly inside service() that time any exception will be raised the exception will show to the end-user, when we won't handle this situation .

If web-container calls `destroy()` then that time the exception will be raised the exception that suppressed by the web-container.

### constructor Vs init() :

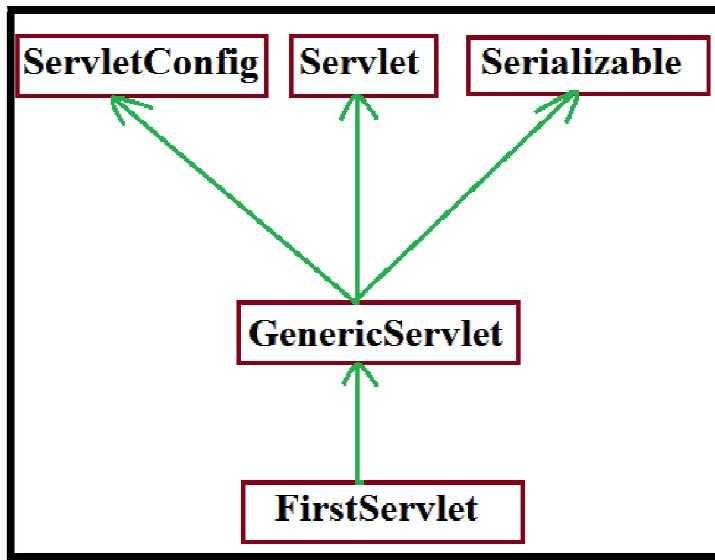
- In general , we can use constructor to perform initialization activities but in old versions of java, constructor cannot accept dynamically generated class name as argument.
- To perform initialization of servlet compulsory we should provide `ServletConfig` object as argument. whose class name is dynamically generated by web-container
- As constructor cannot accept these dynamically generated names, Sun people ignores constructor concept and introduced a specific method `init()` to perform initialization activities , which can take dynamically generated class names as the arguments.
- In the older versions , constructor cannot take any arguments but to perform initialization of a servlet compulsory we should provide `ServletConfig` as argument.
- Hence we can't use constructor to perform initialization activities , Sun people introduced a specific methods `init()` . for this , which can take `ServletConfig` as argument.

### destroy() Vs finalise() :

- Before destroying any object Garbage collector always calls `finalise()` to perform cleanup activities. But we can't expect exact behaviour of Garbage collector. which is vendor dependent.
- Hence instead of depending on `finalise()` , Sun people introduced a specific method `destroy()` to perform cleanup activities , which should be executed always.

### GenericServlet(AC):

- We can develop servlet by implementing `Servlet` interface directly. In this approach compulsory we should provide implementation for all methods of servlet interface whether it is required or not . This approach increase length of the code and reduces readability. We can resolve this problem by using `GenericServlet`.
- `GenericServlet` implements `Servlet` interface and provide default implementation for every method except `service()`. Hence it is an abstract class.
- We can develop servlets very easily by extending `GenericServlet`, instead of implementing `Servlet` interface directly. In this approach we have to provide implementation only for required methods instead of implementing all. It reduces length of the code and improves readability.
- `GenericServlet` is protocol independent servlet.
- `GenericServlet` implements `ServletConfig` and `Serializable` interfaces also.



Demo program to develop Servlet by extending GenericServlet :

```

1) public class FirstServlet extends GenericServlet {
2) public void service(ServletRequest request,ServletResponse response)
3) throws ServletException, IOException {
4) PrintWriter pw=response.getWriter();
5) pw.println("Developing Servlet by GenericServlet");
6) System.out.println("This is Service() in First Servlet ");
7) }
8) }

```

Internal implementation of GenericServlet :

```

1) public abstract class GenericServlet implements Servlet , ServletConfig , Serializable {
2) private transient ServletConfig config ;
3) public void init(ServletConfig config) throws ServletException {
4) this.config=config ; //webcontainer purpose
5) init();
6) }
7) public void init() throws ServletException { //programmer purpose
8) }
9) public abstract service(ServletRequest request,ServletResponse response)
10) throws ServletException, IOException ;
11) public void destroy() {
12) }
13) public ServletConfig getServletConfig() {
14) return config;
15) }

```

```

16) public String getServletInfo() {
17) return "this is GenericServlet";
18) }
19) -----
20) -----
21) }
```

GenericServlet contains 2 init() , we can override init() in our Servlet as follows .

#### 1<sup>st</sup> way :

```

1) public void init(ServletConfig config) throws ServletException {
2) // my own initialization
3) }
```

This approach is not recommended , because we are not saving config object for the future purpose . Hence in our servlet any one calls getServletConfig(), this method returns null .

If we are calling getServletName() then we will get Runtime Exception saying java.lang.NullPointerException

#### 2<sup>nd</sup> way :

```

1) public void init(ServletConfig config) throws ServletException {
2) super.init(config);
3)
4) }
```

This approach is valid , but not recommended to override in Servlet , because internally 3 init () are executed , which impacts performance of the system .

#### 3<sup>rd</sup> way :

```

1) public void init() throws ServletException {
2) // my own initialization activities
3) }
4)
```

This way is highly recommended to define init() in our servlet.



**How many init() present in GenericServlet and explain the need of it ?**

2 init() methods are available in GenericServlet.

One is for web-containers purpose and the Second one is programmers purpose.

**In how many ways we can override init() in our Servlet and which is the best way ?**

3 ways 3<sup>rd</sup> one is the best way.

**In GenericServlet why config variable declared as transient ?**

1. Every Servlet in java is Serializable. At the time of Serialization of servlet object the corresponding config object is serialized automatically . Because it is part of object-graph of Servlet object.
2. If we are serializing config object , from that config object hacker may get corresponding context object reference. Once hacker got context object he can able to perform any operation on our application , which is never be recommended security wise. Hence at the time of serialization of servlet object , we can't Serialize config object. Due to this reason config variable declared as transient.

**2.javax.servlet.http package :**

This package defines more convenient classes and interfaces to define http based Servlets .  
//protocol dependent

**Interfaces of javax.servlet.http : (8)****1) HttpServletRequest :**

It is the child interface of ServletRequest .

HttpServletRequest object can be used to hold client information.

**2) HttpServletResponse :**

It is the child interface of ServletResponse.

This object can be used to prepare and send response to the client.

**3) HttpSession :**

We can use HttpSession object to remember the client information across multiple requests. i.e., we can use session object for session management purpose.

- 4) **HttpSessionListener :**  
To implement http based Listener
- 5) **HttpSessionAttributeListener :**  
To implement http based Listener
- 6) **HttpSessionBindingListener :**  
To implement http based Listener
- 7) **HttpSessionActivationListener :**  
To implement http based Listener
- 8) **HttpSessionContext :**  
deprecated and hence not recommended to use

### Classes of javax.servlet.http package : (7)

- 1) **HttpServlet :**  
It is the child class of GenericServlet . It can be used for developing http based Servlets .
- 2) **Cookie :**  
We can use Cookie objects to implement Session management.
- 3) **HttpSessionEvent :**  
To define Events for http based Listeners.
- 4) **HttpSessionBindingEvent :**  
To define Events for http based Listeners.
- 5) **HttpServletRequestWrapper :**  
To define http based Wrappers.
- 6) **HttpServletResponseWrapper :**  
To define http based Wrappers.
- 7) **HttpUtils :**  
Deprecated , not recommended to use .

#### Note:

In total javax.servlet.http package defines

8 ----->interfaces

7----->classes

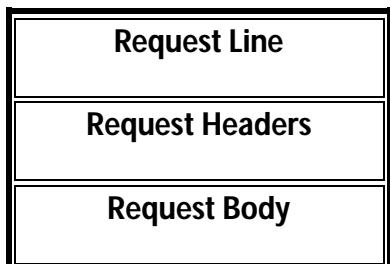
Webserver and webclient communicate by using some common language ,which is nothing

but HTTP .

Http defines a standard structures for HttpRequest and HttpResponse .

### Structure of HttpRequest :

Browser always sends the request in the following format :



### Request Line :

|                         |                             |                                                |
|-------------------------|-----------------------------|------------------------------------------------|
| GET<br>(Request method) | /login.jsp<br>(Request URI) | HTTP/1.0<br>(protocol version used by browser) |
|-------------------------|-----------------------------|------------------------------------------------|

### Request Headers :

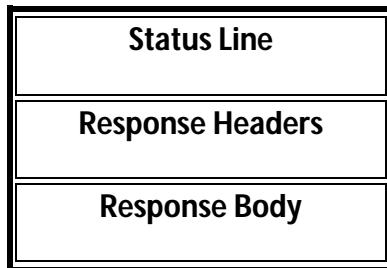
The request headers describes configuration information of the browser like

1. media types accepted by browser , language accepted by browser , encoding types supported by browser etc.
2. Web server use this information to send customized responses to the client .

### Request Body :

1. It contains end-user provides information for the GET request it is optional where as for the POST request it is mandatory.
2. This is the structure of HttpRequest , If the browser sends request in this format , then only webserver understands the request . i.e, webserver understandable form.

## Structure of HttpResponse :



### Status Line :

|                      |                     |                                               |
|----------------------|---------------------|-----------------------------------------------|
| 200<br>(Status code) | OK<br>(Description) | HTTP/1.1<br>(protocol version used by server) |
|----------------------|---------------------|-----------------------------------------------|

### Status Codes :

1xx -----> Informational

2xx----->Successful

3xx----->Redirectionaal

4xx----->client error

5xx----->Server error

### Response Headers :

These will provide configuration information of the server and information about the response [meta data] like content type of response , content length , last modified data etc., Browser will use these response headers to represent properly the response to the end-user.

### Response Body :

It contains the original response provided by webserver.

If the server sends the response in the above form , then only browser understands the response i.e., it is browser understandable form.

**Types of HttpRequest methods :**

Based on type of information requested by the browser HttpRequest methods are divided as follows

1. GET
2. POST
3. HEAD
4. OPTIONS
5. PUT
6. DELETE
7. TRACE
8. CONNECT
9. MOVE
10. LOCK
11. PROFOUND

1 - 3 methods introduced in http 1.0

4 - 11 methods introduced in http 1.1

1 - 7 are Big Http methods

**GET :**

1. We can use get request if we are expecting information from the server .
2. Usually for the get request read operation will be performed at server side . Hence status of application won't be changed in get request.
3. In GET request end-users provided information will be appended to the url as the part of Query string.
4. As the end-users information is visible in url. there may be a chance of security problems will raise. Hence sensitive data we can't send by using get request.
5. The length of the url is fixed . Hence we can send only limited amount of information by get request .
6. Only character data is allowed in url . Hence we can't send binary data by using get request.
7. As extra client provided information is available in the url , Bookmarking of url is possible .

**Idempotent request :**

By repeating the request multiple times , if there is no change in response such type of requests are Idempotent requests.

Ex: GET requests are Idempotent and POST requests are not Idempotent.

**Safe request :**

By repeating the same request multiple times , if there is no side-effect at server side , such type of requests are called safe-requests .

Ex: GET requests are safe , where as POST requests are not safe to repeat.

**Triggers to send GET request :**

- 1) Type url in the address bar and submit is always GET request.
- 2) Clicking hyperlink is always GET request.
- 3) Submitting the form , where method attribute specify with GET value is always GET request.

```
1) <form action="/test" method="GET">
2) -----
3) </form>
```

- 4) Submitting the form without method attribute is always GET request i.e., default method for form is GET.

```
1) <form action="/test">
2) -----
3) -----
4) </form>
```

**POST :**

1. If we want to post huge amount of information to the server then we should go for POST.  
Ex: uploading our resume in job portal.
2. Usually in post requests update operation will be performed. The state of operation will be changed.
3. In post request , client information will be encapsulated in the request body instead of appending to the url . Hence we can send sensitive data by using POST request.
4. There is no limit on size of request body . Hence we can send huge amount of information to the Server.

5. We can send binary data also in addition to text data.
6. Bookmarking of POST request is not possible.
7. POST requests are not Idempotent and not-safe to repeat.

### *Triggers to send POST request :*

There is only one-way to send POST request that is to use the form with method attribute value is POST .

- 1) `<form action="/test" method="POST">`
- 2) -----
- 3) -----
- 4) `</form>`

I.e., without having the form there is no chance of sending POST request.

### *Differences between GET & POST :*

| GET                                                                       | POST                                                                         |
|---------------------------------------------------------------------------|------------------------------------------------------------------------------|
| If we are expecting information from server , then we should go for GET . | If we want to post huge information to server , then we should go for POST . |
| Usually read operation will be performed.                                 | Write & update operators will be performed .                                 |
| Client data will be appended to url in the form of Query-String .         | Client data will be encapsulated in request body .                           |
| We can't send sensitive information .                                     | We can send sensitive information .                                          |
| We can send only limited information.                                     | We can send Huge information.                                                |
| We can send only text data.                                               | We can send text & binary data.                                              |
| Bookmarking is possible .                                                 | Bookmarking is not- possible .                                               |
| GET requests are Idempotent & Safe.                                       | POST requests are not-Idempotent & not-Safe.                                 |
| Multiple ways to send GET request.                                        | Only one way to send POST request.                                           |

### **HEAD :**

- We can use this method to get only response header information like content type , content length , last modified date etc., but no response body i.e., head request retrieves always response header but not response body.

- For HEAD request , internally doGet() will be executed. HEAD request is part of GET request.
- HEAD requests are Idempotent & Safe.

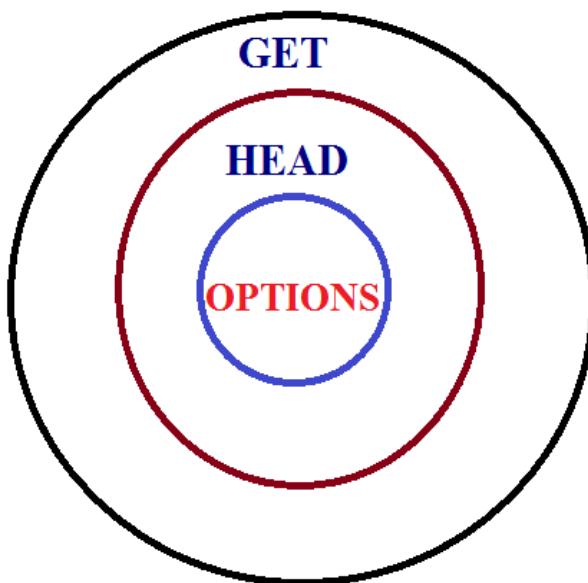
```

1) HEAD request -----> doHead {
2) doGet {
3) take only header part of GET response and
4) provide that as response to HEAD request.
5)
6) }

```

### OPTIONS :

- This method is for getting supporting http methods to retrieve a particular resource from the Server side .
- OPTIONS method is Idempotent and Safe.
- **Note:** HEAD response is the part of GET response and OPTIONS is the part of HEAD response .



response from OPTIONS method request :

Http 1.1 | 200 | OK

Host : www.jobs4times.com  
 Server : Apache  
 Date : Fri, 08 Nov

Allow : OPTIONS , HEAD , GET , POST

Content Length : 0

### PUT :

1. We can use PUT method for placing a resource at Server side where the location is specified by URL.
2. At the specified location , if already another resource present then the old resource is replaced with provided new resource .
3. By means of status code we can identify whether replacement is happen or not .
4. 200 means replacement happen , 201 means replacement not happen.
5. PUT method is Idempotent but not Safe .

### DELETE :

- We can use this method for deleting a particular resource from Server side. It is exactly counter part of PUT method.
- DELETE is Idempotent , but not-Safe.

**Note:** As the PUT & DELETE methods are not safe. Most of the web-servers won't allow these methods by default.

To allow these methods at server side some configuration changes are required.

### TRACE :

We can use this method for debugging purposes. If we want to know what request , server getting exactly as response, then we should go for TRACE method.

TRACE method is Idempotent and Safe.

| Method  | Is Idempotent ? | Is Safe ? |
|---------|-----------------|-----------|
| GET     | Yes             | Yes       |
| POST    | No              | No        |
| HEAD    | YES             | YES       |
| OPTIONS | Yes             | Yes       |
| PUT     | Yes             | No        |
| DELETE  | Yes             | No        |

TRACE

Yes

Yes

**Note :** The only non-idempotent method is POST .

The following methods are not safe : POST , PUT , DELETE .

### **HttpServlet :**

We can use HttpServlet class as a base class to develop Http based Servlets.

It is the child class of GenericServlet.

For every Http method XXX , HttpServlet class contains the corresponding doXxx() methods.

```
protected void doXxx() throws ServletException, IOException
```

**Ex:**

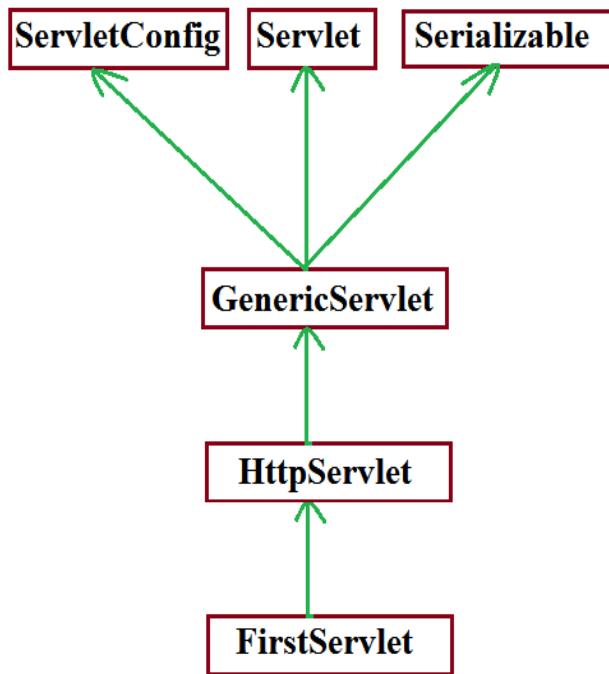
```
protected void doGet(HttpServletRequest request , HttpServletResponse response) throws
ServletException, IOException
```

doGet(), doPost(), doHead(), doOption(), doPut(), doDelete(), doTrace() .

**HttpServlet contains 2 service() methods.**

1. public void service(ServletRequest request , ServletResponse response ) throws ServletException, IOException
2. protected void service(HttpServletRequest request , HttpServletResponse response ) throws ServletException, IOException





**Ex:** Demo program for HttpServlet.

**login.html**

We can use this to send post request to the server

```

1) <html>
2) <body><h1>This is HttpServlet Demo</h1>
3) <form action="/scwcd1c/test" method="post">
4) Enter Name : <input type="text" name="uname" >
5) <input type="submit">
6) </form>
7) </body>
8) </html>

```

**FirstServlet.java**

```

1) import javax.servlet.*;
2) import javax.servlet.http.*;
3) import java.io.*;
4) public class FirstServlet extends HttpServlet {
5) public void doGet(HttpServletRequest request, HttpServletResponse response)
6) throws ServletException, IOException {
7) PrintWriter out=response.getWriter();
8) String name=req.getParameter("uname");
9) out.println("Hello"+name+"Good Morning , This is doGet method");

```

```

10) }
11) public void doPost(HttpServletRequest request, HttpServletResponse response)
12) throws ServletException, IOException {
13) PrintWriter out=response.getWriter();
14) String name=req.getParameter("uname");
15) out.println("Hello "+name+" Good Morning , This is doPost method");
16) }
17)

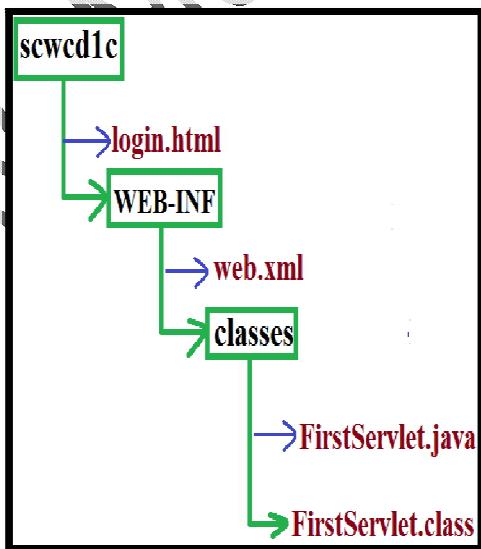
```

**web.xml**

```

1) <web-app>
2) <servlet>
3) <servlet-name>first</servlet-name>
4) <servlet-class>FirstServlet</servlet-class>
5) </servlet>
6) <servlet-mapping>
7) <servlet-name>first</servlet-name>
8) <url-pattern>/fs</url-pattern>
9) </servlet-mapping>
10) </web-app>

```

**Deployment Structure :****Life Cycle of HttpRequest :**

- 1) When ever we are submitting the form , browser prepares HttpRequest and send to the Server.

- 2) Webserver checks whether the request is for static or for dynamic information, by using URL.
- 3) If the request is for static information , webserver provide required response.
- 4) If the request is for dynamic information , then webserver forwards that request to web-container.
- 5) Webcontainer identifies the corresponding matched servlet by using web.xml
- 6) Webcontainer checks whether the corresponding Servler object is already available or not . If it is not already available , then webcontainer loads the corresponding .class file and perform instantiation by executing public no argument constructor .
- 7) After instantiation , immediately webcontainer creates the ServletConfig object and invoke init() by passing that config object as argument.
- 8) Webcontainer creates ServletRequest and ServlerResponse object and by passing those as arguments , It will invoke public service().
- 9) With in that service() , request and response objects will be type casted into HttpServletRequest and HttpServletResponse and invoke protected service() by passing those as arguments.

```

1) public void service(ServletRequest req , ServletResponse res)
2) throws ServletException, IOException {
3) HttpServletRequest req=(HttpServletRequest)req ;
4) HttpServletResponse res=(HttpServletResponse)res ;
5) service(req1 , res1);
6) }
```

- 10) With in the protected service() we will identify HttpRequest method and corresponding doXxx() will be invoked. If it is not a valid Http method then it is method will return 501 status code response saying invalid Http method .

```

1) protected void service(HttpServletRequest request,HttpServletResponse response)
2) throws ServletException, IOException{
3) String method=request.getMethod();
4) if(method.equals("GET")) {
5) doGet(req , res);
6) }
7) else if(method.equals("POST")) {
8) doPost(req , res);
9) }
10) else if(method.equals("HEAD")) {
11) doHead(req , res);
12) }
13)
14)
15) else {
16) prepare response with 501 Status code saying Invalid Http Method.
17) }
```

18) }

11) If our Servlet class contain required doXxx() , then it will be executed . Otherwise parent class doXxx() will be executed.

**NOTE :**

1. For every web application, web container creates one ServletContext object at the time of application deployment.
2. For every Servlet , web container creates one ServletConfig object just before calling init().
3. For every request web container creates one ServletRequest and one ServletResponse objects just before calling service() . Once service() completes both request and response objects will be eligible for Garbage Collector(GC).

**Case 1 :** If we are writing public service() in our Servlet , then for any type of request [GET or POST] only service() will be executed.

**Case 2 :** If our Servlet contains both public service() and protected service() , then for every request only public service() will be executed.

**Case 3 :** If our Servlet contains both service() and doGet() , then for any request including GET , only service() will be executed. And there is no chance of executing doGet().

**Note:** It is never recommended to place service() in our Servlet. i.e., defining service() in our Servlet is stupid kind of activity.

**Case 4 :** If we are sending get request , but our Servlet doesn't contain doGet() , then HttpServlet doGet() will be executed which provides response with 405 status code saying Http method GET is not Supported by this url.

**Case 5 :** If we are sending post request , but our servlet doesn't contain doPost() , then HttpServlet doPost() will be executed , which provide 405 status code saying Http method POST is not Supported by this url.

**Case 6 :** HttpServlet class doesn't contain any abstract method , still it is declared as abstract class what is the reason ?

For majority methods in HttpServlet , proper implementation is not available and these are just to send error information .

Hence if we are creating HttpServlet object directly and calling these methods , we will get just error information , with that we can't do anything.

To prevent instantiation of HttpServlet , Sun people declared HttpServlet class as abstract .

**Case 7 :** To provide same response for both GET and POST request , we have to implement doGet() and doPost() as follows .

Ex:

```

1) public class FirstServlet extends HttpServlet {
2) public void doGet(HttpServletRequest request,HttpServletResponse response)
3) throws ServletException, IOException {
4) // implemented required information
5) }
6) public void doPost(HttpServletRequest request,HttpServletResponse response)
7) throws ServletException, IOException {
8) doGet();
9) }
10) }
```

**Case 8 :** In our Servlet , It is not recommended to override the following methods . Because these methods are properly implemented inside HttpServlet.

1. service(ServletRequest request, ServletResponse response)
2. service(HttpServletRequest request, HttpServletResponse response)
3. doHead()
4. doOptions()
5. doTrace()

**Case 9 :** In our Servlet , it is highly recommended to override the following methods , because these methods doesn't have proper implementation in HttpServlet.

6. doGet()
7. doPost()
8. doPut()
9. doDelete()

**Case 10 :** To provide response to HEAD request , we have to override either doHead() or doGet() , otherwise we will get 405 status code saying Http method GET is not supported by this url.

## HttpServletRequest :(I)

Using HttpServletRequest , write code to

1. Retrieve Form parameters
  1. getParameter()
  2. getParameterValues()

- 3. `getParameterNames()`
- 4. `getParameterMap()`
- 2. Retrieve request Headers
  - 1. `getHeader()`
  - 2. `getHeaders()`
  - 3. `getHeaderNames()`
  - 4. `getIntHeader()`
  - 5. `getDateHeader()`
- 3. Retrieve Cookies
  - 1. `getCookies()`

### **Retrieve Form parameters :**

- Form parameters are key-value pairs , where both key and values are String objects only.
- A parameter can be associated with either a single value or with multiple values.
- ServletRequest interface defines the following methods to retrieve Form parameters at server side .
  - 1. `getParameter()`
  - 2. `getParameterValues()`
  - 3. `getParameterNames()`
  - 4. `getParameterMap()`

#### **1. getParameter():**

```
public String getParameter(String pname)
```

- It returns the value associated with specified parameter.
- If the parameter associated with multiple values then this method simply returns only first value.
- If the specified parameter not available then we will get null argument is case-sensitive .

```
String user=request.getParameter("uname");
```

#### **2. getParameterValues()**

```
public String[] getParameterValues(String pname)
```

- It returns all values associated with specified parameter.

- If the parameters not available , we will get null .
- Argument is Case-Sensitive.
- Ex:

```
1) String[] course=request.getParameterValues("course");
2) for(String s:course) {
3) out.println(s);
4) }
```

### 3. getParameterNames()

**public Enumeration getParameterNames()**

- Returns all Form parameters names.
- If the request is not associated with any form parameters. Then we will get empty enumeration object but not null.

Ex:

```
1) Enumeration e= request.getParameterNames();
2) while(e.hasMoreElements()) {
3) String pname=(String)e.nextElement();
4) String pvalue=request.getParameter(pname);
5) out.println(pname+" "+pvalue);
6) }
```

### 4. getParameterMap()

**public Map getParameterMap()**

- Returns the map object containing parameter names as keys and parameter values as map values.
- Keys are strings and values are string[ ] .
- If the request doesn't associated with any parameters , this method returns empty map object , but not null.

| key(String) | value(String[ ]) |
|-------------|------------------|
| subject     | {SCJP , SCWCD }  |
| user        | { arun }         |
| .....       | .....            |
| -----       | -----            |

Ex:

```
1) Map m = request.getParameterMap();
2) for(Object o : m) {
3) Map.Entry m1=(Map.Entry)o;
4) String pname=(String)m1.getKey();
5) String[] pvalue=(String[])m1.getValue();
6) out.print(pname+".....");
7)
8) for(String s1 : pvalue) {
9) out.println(s1 + " , ");
10}
11) out.println(" ");
12}
13) output :
14) subjectSCJP , SCWCD
15) user.....arun
```

Example :

```
1) Map m=req.getParameterMap();
2) Set keySet=m.entrySet();
3) Iterator itr=keySet.iterator();
4) while(itr.hasNext()) {
5) Map.Entry entry=(Map.Entry)itr.next();
6) String pname=(String)entry.getKey();
7) out.println(pname);
8) String[] pvalues=(String[])entry.getValue();
9) out.println(pvalues);
```

Example :

```
1) public void doPost(..)..
2) response.setContentType();
3) PrintWriter pw=res.getWriter();
4) out.println("");
5) out.println("name :" + request.getParameter("uname"));
6)
7)
8) }
```

## Retrieving Request headers :

- For every request browser sends its configuration information in the form of request headers. These may include the media types accepted by browser , encoding types supported by browser the type of browser etc.
- Server uses these request headers to send customized responses to the client.

*The following are some of the important request headers.*

- 1) **Accept :**  
media types accepted by browser.[ like txt/html , pdf , ppt , jar ]
- 2) **Accept-Encoding:**  
Encoding types supported by browser.
- 3) **User-agent:**  
It represents the type of the browser.
- 4) **Content-length:**  
It represents the length of request body.
- 5) **Cookie :**  
Used to send cookies for Session management.

## Utilities of Request headers :

1. By using Accept-Encoding request header at Server side we can identify whether browser supports compression from or not.
2. If browser supports compressed form , we can send response in compressed form. So that we can save download time and bandwidth.
3. By using User-agent request header at server side we can identify the type of browser which sends the request. Based on that we can send browser compatible customized responses.
4. By using Cookie request header we can send Cookie to the server so that we can achieve session management.

**HttpServletRequest defines the following methods to retrieving header information at server side**

1. **getHeader()**

```
public String getHeader(String hname)
```

- It returns the value associated with specified request header.
- If the header associated with multiple values then this method returns only 1<sup>st</sup> value .
- If the specified request header is not available then we will get null.
- Argument is not-case sensitive .

Ex:

```
String hvalue=request.getHeader("accept");
output :
accept -- text/html , application/xhtml+xml , application/xml ;
```

## 2. getHeaders()

**public Enumeration getHeaders(String hname)**

- Returns all values associated with specified header .
- If the specified header not available , then this method returns empty Enumeration object, but not null.
- Argument is not-case sensitive .

## 3. getHeaderNames()

**public Enumeration getHeaderNames()**

Returns all request header names associated with that request object.

HttpServletRequest defines the following more convenient methods to retrieve int and date values(Headers) directly .

## 4. getIntHeader()

**public int getIntHeader(String hname);**

Ex:

```
String length=request.getIntHeader("content-length");
int l=Integer.parseInt(length);
(OR)
int l=request.getIntHeader("content-length");
```

If the specified request header is not associated with int value , then this method returns NumberFormatException.

```
req.getIntHeader("user-agent");//RE:java.lang.NumberFormatException
```

If the specified request header is not available in getIntHeader() , this method returns "-1" but not null.

#### 5. getDateHeader()

```
public long getDateHeader(String hname);
```

- Returns the date value associated with specified header as no. of milliseconds Since Jan 1<sup>st</sup> 1970.
- If we pass these milliseconds as arguments to Date constructor. We will get exact date

Ex:

```
long l=request.getDateHeader("date");
Date d=new Date(l);
```

- If the specified header is not associated with the date , then we will get RuntimeException saying IllegalArgumentException Exception.
- If the specified request header is not available in getDateHeader() , this method returns "-1" but not null.

***Write a Servlet to print all request-headers associated with the request.***

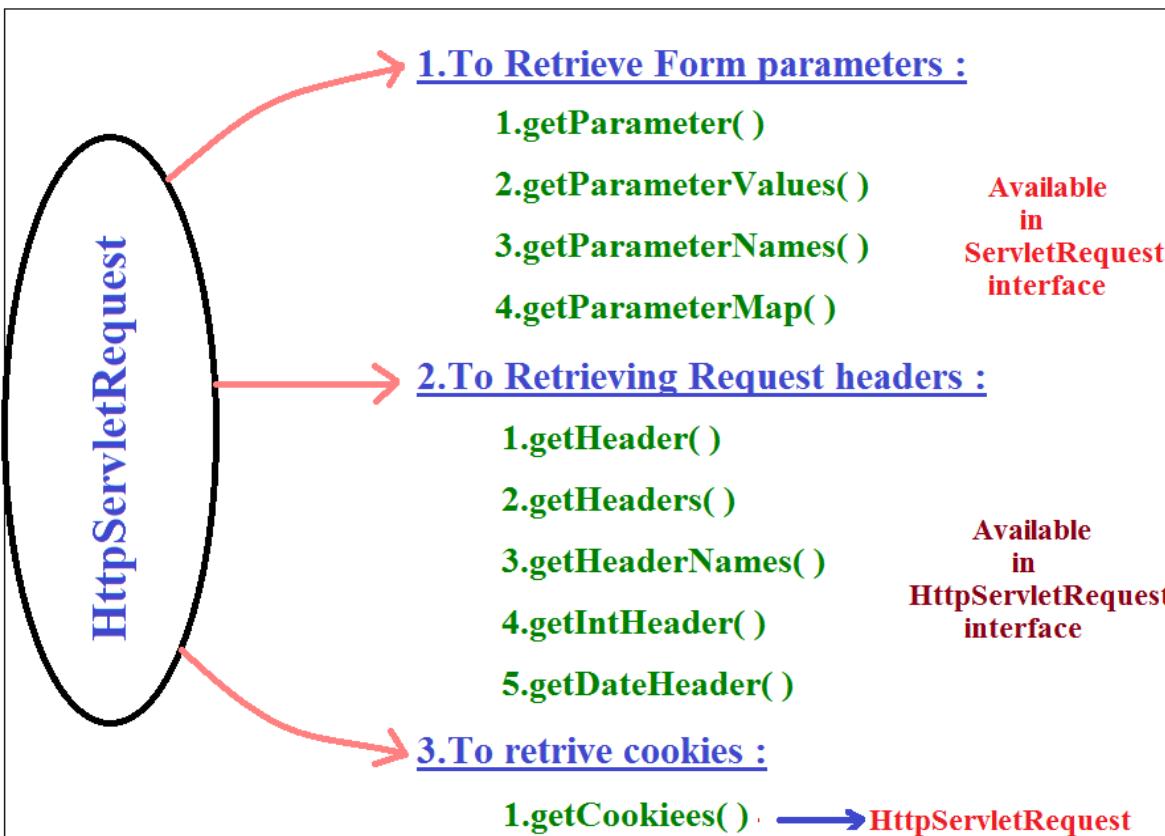
EX:

```
1) public class ReqHeader extends HttpServlet {
2) public void doGet(HttpServletRequest request,HttpServletResponse response)
3) throws ServletException, IOException {
4) PrintWriter out = response.getWriter();
5) Enumeration e = request.getHeaderNames();
6) while(e.hasMoreElements()) {
7) String hname=(String)e.nextElement();
8) String hvalue=request.getHeader(hname);
9) out.println(hname+"-----"+hvalue);
10) }
11) }
12) }
```

### Retrieving Cookies from the request :

HttpServletRequest interface defines the following methods to retrieve cookies from the request object .

```
Cookie[] c = request.getCookies();
```



### Retrieving client & Server information from the request :

ServletRequest interface defines the following methods to retrieve client and server information from request.

```
public String getRemoteHost()
```

Returns fully qualified name of the client which sends the request.

```
public String getRemoteAddr()
```

Returns IP-address of the client .

**public String getRemotePort()**

Returns the port no. on which the client is running .

**public String getServerName()**

Returns the name of the Server to which the request has been sent .

**public String getServerPort()**

Returns the port no. on which server is running .

A request triggered from **www.jobs4times.com** with an IP-address **192.168.203.32** running on port **1012**. Place the appropriate ServletRequest methods to their corresponding values.



**getRemotePort( )**  
**getServerHost( )**  
**getRemoteHost( )**  
**getRemoteAddr( )**  
**getServerAddr( )**  
**getServerName( )**  
**getServerPort( )**

### HttpServletResponse :

By using **HttpServletResponse** interface , write code

1. To set Response Headers .
2. To set content type of Response .
3. To acquire text stream for response.
4. To acquire binary stream for response .
5. To redirect the request to another URL .
6. To add cookies to response.

### Setting HttpServletResponse Headers :

- **Http response header describes the configuration information of the server and information about the response like content-type , content-length , last modified date etc.,**
- **Browser using these response headers to display response body properly to the end user.**

**HttpServletResponse** defines the following methods to add headers to the response .

**public void addHeader(String hname , String hvalue )**

- If the specified header name is already available , to the existing values , this new value will also be added.
- Replacement will not Occur .

**public void setHeader(String hname , String hvalue )**

- If the specified header is already available . Then the old value will be replaced with new value .

Some times headers associated with int & date values .

1. **public void addIntHeader(String hname , int hvalue )**
2. **public void setIntHeader(String hname , int value )**
3. **public void addDateHeader(String hname , long ms )**
4. **public void setDateHeader(String hname , long ms )**

#### Note:

If the header associated with multiple values . Then we have to use addHeader()

If the header associated with only one value , then we should go for setHeader()

### **Set Content type of Response :**

Content type header represents **MIME** type (multipurpose internet mail extension) of the response.

#### **Common MIME Types :**

1. **text/html -----> Html text as response**
2. **text/xml -----> xml document as response**
3. **image/jpeg -----> JPEG as response**
4. **image/png -----> png as response**

5. application/ms-excel -----> excel sheet as response
6. application/ms-word -----> word document as response
7. application/pdf -----> PDF file as response
8. application/jar -----> jar file as response

We can set MIME type by using the following 2 ways

**By ServletResponse interface :**

It contains the following method to set the response.

```
public void setContentType(String mimeType)
```

Ex:

```
response.setContentType("application/PDF")
```

**By HttpServletResponse interface :**

It contains setHeader() to set content type.

```
public void setHeader(String hname , String hvalue)
```

Ex:

```
response.setHeader("content type" , "text/html");
```

Note: In general response.setContentType() is recommended to use.

To acquire text stream for the response :

1. We can send text data as the response by using PrintWriter object .
2. We can get PrintWriter object by using getWriter() of ServletResponse .

```
public PrintWriter getWriter() throws IOException
```

Ex:

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

To acquire Binary stream as response :

1. We can send Binary information(video , image files) by using ServletOutputStream object.
2. By using getOutputStream() of ServletResponse we can get this object .

```
public ServletOutputStream getOutputStream() throws IOException
```

Ex:

```
ServletOutputStream sos=response.getOutputStream();
```

Demo program to send image file as response to the client

```

1) import javax.servlet.*;
2) import javax.servlet.http.*;
3)
4) public class BinaryStream extends HttpServlet {
5) public void doGet(HttpServletRequest request,HttpServletResponse response)
6) throws ServletException,IOException{
7) response.setContentType("image/jpeg");
8) ServletOutputStream sos = response.getOutputStream();
9)
10) //File f= new File("c:\\tomcat\\sunset.jpeg");
11)
12) String path=getServletContext().getRealPath("sunset.jpeg");
13) File f= new File(path);
14) FileInputStream fis=new FileInputStream(f);
15)
16) byte[] b=new byte[(int)f.length()];
17) //f.length() returns long so we have to convert into int.
18) //[] --> accepts int only
19)
20)
21) fis.read(b); //reading the image & place into byte[]
22) sos.write(b); //write byte[] to response
23) sos.flush();
24) sos.close();
25) }
26}
```

### To send XML data as a response :

```

1) response.setHeader("content-Type", "text/xml");
2) PrintWriter out=response.getWriter();
3) out.print("<?xml version=\"1.0\"?>");
4) out.print("<greeting language=\"en_US\">");
5) out.print("hello world");
6) out.print("</greeting>");
```

### Note :

At any point of time we can get either PrintWriter object (or) ServletOutputStream object but not both , simultaneously.

Otherwise we will get RuntimeException saying IllegalStateException

Because from the Servlet we can send only one response at a time.

```

1) public void doGet() {
2) PrintWriter out=response.getWriter();
3) ServletOutputStream sos =response.getOutputStream();
4) -----
```

- 5) -----  
6) }

**Output:** RuntimeException saying **IllegalStateException** getWriter() has already been called for this response.

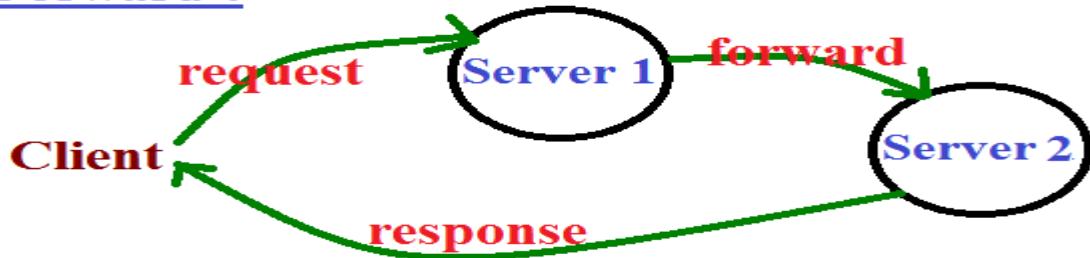
**IQ:** By using which of the following stream we can send both binary & text data as response from the Servlet ?

1. PrintWriter( )
2. ServletOutputStream( )
3. ServletInputStream( )
4. None of the above ( )

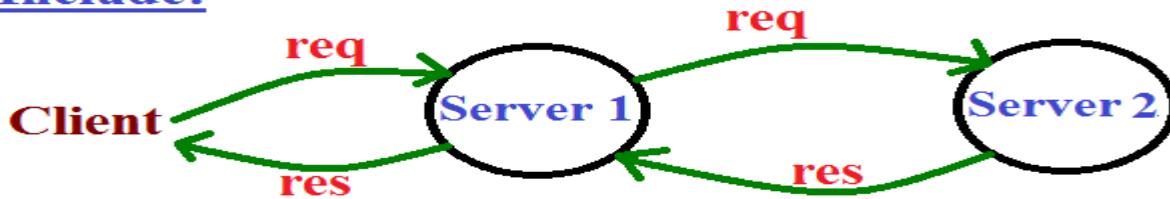


### Re-directing HttpServletRequest to another url :

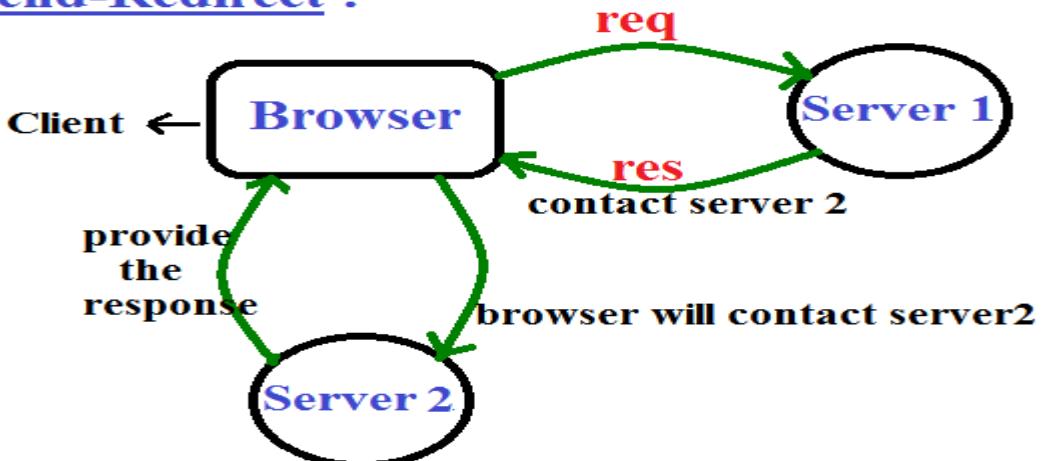
#### Forward :



#### Include:



#### send-Redirect :

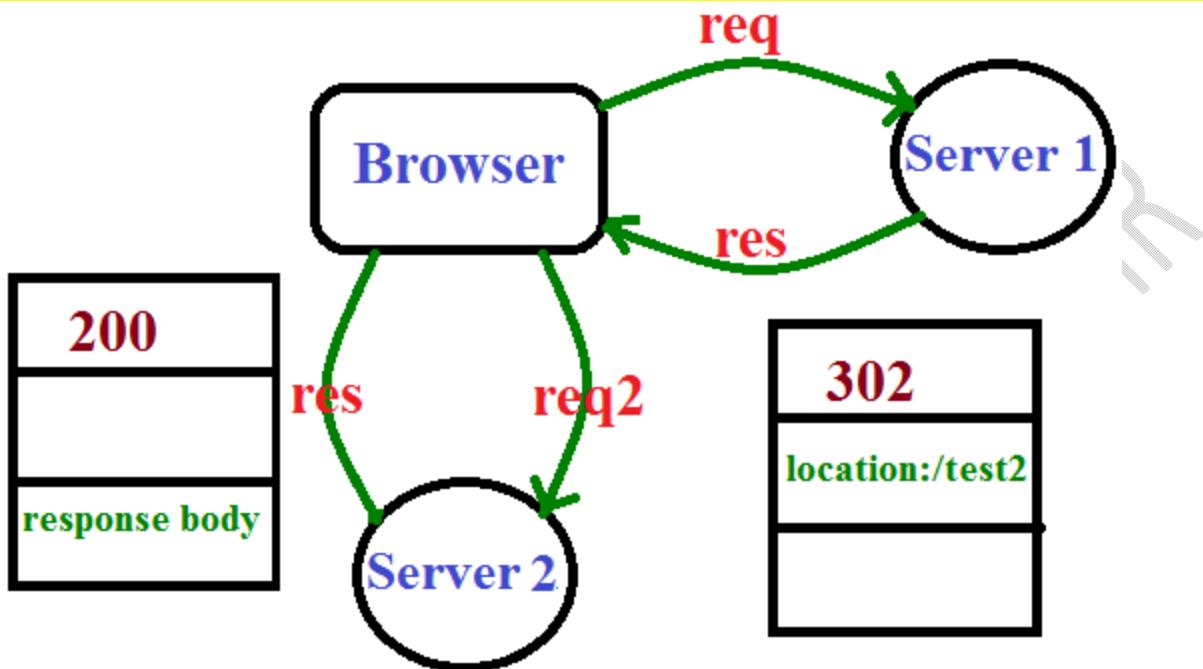


Forward & include should work when both servlets should be in same server .

Some times we have to re-direct the request to another url we can achieve this by using `sendRedirect()` of `HttpServletResponse` .

```
public void sendRedirect(String targetpath) throws IOException
```

**Process of send Redirection :**



**Here operation takes place at browser  
(i.e., client-side)**

1. Browser sends request to Servlet1
2. If servlet1 is not responsible to provide response , then it will update browser to contact servlet2. This can be informed by setting status code 302 & location header with servlet2.
3. Browser by seeing status code 302 creates a new request object & sends to Servlet2.
4. Servlet2 provides desired response to the browser & the browser inturn display that response to the end user .

**Demo program for send Re-direct :**

```

1) public class FirstServlet extends HttpServlet {
2) public void doGet(HttpServletRequest request,HttpServletResponse response)
3) throws ServletException, IOException {
4) response.sendRedirect("test2");
5) (or)
6) response.setStatus("302");
7) response.setHeader("location" , "scwcd/test2");
8) }
9) }
```

```

1) public class SecondServlet extends HttpServlet {
2) public void doGet(HttpServletRequest request,HttpServletResponse response)
3) throws ServletException, IOException {
4) PrintWriter out=response.getWriter();
5) out.println("2nd Servlet ");
6) }
7) }
```

- When ever we are sending request to FirstServlet , SecondServlet will provide the response through re-direction
- After committing the response we are not allow to perform re-direction. Otherwise we will get RuntimeException saying IllegalStateException(Tomcat people doesn't provide support for this).

```

1) public void doGet(HttpServletRequest request,HttpServletResponse response)
2) throws ServletException, IOException {
3) -----
4) -----
5) out.flush(); //committing the response
6) response.sendRedirect("/test2"); //RE:IllegalStateException
7) }
```

**www.durgasoftonlinetraining.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

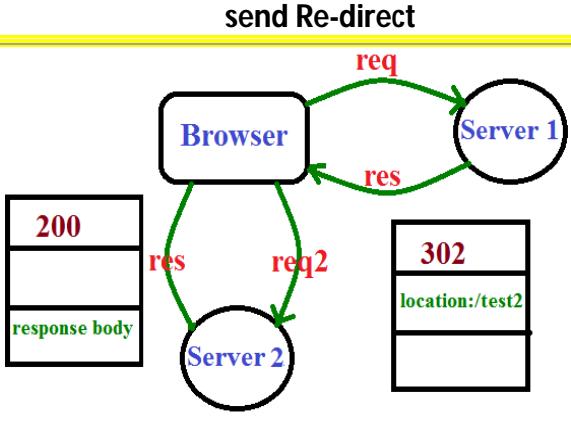
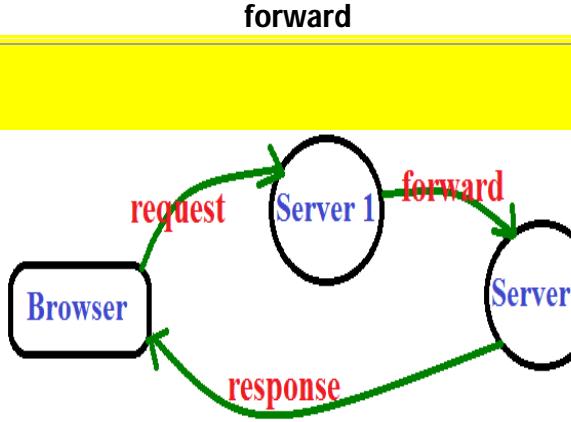
**Ph: +91-8885252627, 7207212427  
+91-7207212428**

**USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

Applicable for forward also.

### Differences between send-Redirection and forward mechanism :

| send Re-direct                                                                                                                                     | forward                                                                                                                                                     |
|----------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  <p>Here operation takes place at browser (i.e., client-side)</p> |                                                                           |
| This mechanism will work at client side. Hence client aware of which Servlet is providing the required response.                                   | This mechanism will work at server side and it is not visible to the client. Hence client is not aware of which Servlet is providing the required response. |
| In this approach an extra trip is required to the client side . Hence network traffic increases & effects the performance of the system.           | No extra trip is required to the client & hence there are no network traffic & performance problem.                                                         |
| This is only possible mechanism to communicate with resources which are present outside of web-container.                                          | forward mechanism will work with in the web-container only and we can't use this mechanism to communicate with the resources present outside of container.  |
| Separate new request object will be created in send Redirection .                                                                                  | Same request object will be forwarded .                                                                                                                     |
| By using HttpServletResponse object . we can achieve send redirection.<br><code>response.sendRedirect("/test2");</code>                            | By using Request Dispatcher object we can achieve forward mechanism .<br><code>rd.forward(req , res);</code>                                                |

After committing the Response we are not allow to perform send Redirection .Otherwise we will get RuntimeException saying IllegalStateException.

After Commiting Response , we can't perform forward otherwise IllegalStateException.

### Adding Cookies to the response :

HttpServletResponse contains the following method to addCookie to the Response object.

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

Ex:

```
Cookie c=new Cookie(String name , String value);
response.addCookie(c);
```

### NOTE :

1. We can use redirection to communicate either with in the same server or outside the server. But recommended to use communicate outside.
2. forward mechanism is applicable to communicate only with in the same server . But outside of server we can't use.
3. To communicate with in the same server we can use either forward or send redirection but recommended to use forward.





H  
Y

**LEARN FROM EXPERTS ...**

# **COMPLETE JAVA**

CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES....

# **COMPLETE .NET**

C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

# **TESTING TOOLS**

MANUAL + SELENIUM

# **ORACLE | D2K**

# **MSBI | SHARE POINT**

# **HADOOP | ANDROID**

# **C, C++, DS, UNIX**

# **CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED



# 202, 2nd Floor, HUDA Maitrivanam,

Ameerpet, Hyd. Ph: 040-64512786,

**9246212143, 8096969696**

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA With SCWCD / OCWCD Servlets Material**

## **3. The Web Container Model**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

**Ex. IBM Employee**

**Trained Lakhs of Students  
for last 14 years across INDIA**

**India's No.1 Software Training Institute**

**DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

# Web Container Model

## Agenda:

- 1) **ServletContext parameter**
  - Comparisons between Servlet init & context initialization parameters
- 2) **Servlet Scopes**
  - Request Scope
  - Session Scope
  - Application Scope (or) Context Scope
  - Multithreading issues associated with each scope
  - Differences between parameters and attributes
  - To print all context scoped attributes:
  - To print hit count of the web-application:
  - To print no. of requests send in each(current) Session
  - To print no.of Session objects created in application
  - no. of requests triggered Ip-address wise
  - How do we make Context scoped attributes Thread safe ?
- 3) **RequestDispatcher**
  - Differences between getting RequestDispatcher by ServletRequest & ServletContext
  - RequestDispatcher's methods
    - Forward mechanism
    - Include mechanism
  - Differences between forward() and include() :
  - Foreign RequestDispatcher
- 4) **FILTERS (servlet 2.3 v)**
  - FILTER- API :
    - Filter
    - FilterConfig
    - FilterChain
  - Configuring Filter in web.xml
  - Mapping of Filter :
    - Mapping to a particular url-pattern
    - Mapping to a particular Servlet
    - Mapping to entire web-application
  - dispatcher tag
  - Difference between Filter's doFilter() and FilterChain's doFilter()
- 5) **Wrappers**

- RequestWrapper
- ResponseWrappers
- Demo program for request wrapper
- Demo program for response wrapper

### **ServletContext parameter :**

For the ServletContext initialization parameter.

1. Write Servlet code to access initialization parameters.
2. Create the deployment descriptor elements for the initialization parameter.
1. For every Servlet , web container will create one ServletConfig object to maintain Servlet level initialization parameter . By using this object Servlet can get its configuration information.
2. Similarly for every web-application webcontainer creates one ServletContext object to maintain application level configuration information
3. Servlet can get application level configuration information through this context object only.
4. ServletConfig per Servlet , where as ServletContext per Web-application.
5. If initialization parameters are common for all Servlets then it is not recommended to declare those parameters at Servlet level , we have to declare such type of parameters at application -level by using <context-param>

```
<web-app>
 <context-param>
 <param-name>username</param-name>
 <param-value>scott</param-value>
 </context-param>
</web-app>
```

6. We can declare any number of context parameters but one <context-param> for each Servlet.
7. <context-param> is the direct child tag of <web-app> & hence we can declare any where with in <web-app>
8. The context initialization parameters are available through out the web-application any where.
9. If we want to use same init-parameters then we can declare at context-level as

### **<context-param>**

```
<param-name>username</param-name>
<param-value>scott</param-value>
</context-param>
```

10. With in the Servlet we can access these context initialization parameters by using ServletContext object.

11. We can get ServletContext object by using getServletContext() of ServletConfig interface.

With in our Servlet the following 3 possible ways to get the ServletContext object.

```
ServletContext context=getServletContext()
```

```
ServletContext context=getServletConfig().getServletContext()
```

```
ServletContext context=this.getServletConfig().getServletContext()
```

We can retrieve ServletContext parameters by using ServletContext object for that purpose. ServletContext interface defines the following methods for accessing context initialization parameters.

1. String getInitParameter(String name)
2. Enumeration getInitParameterNames()

Demo program to display context initialization parameters :

```
public class FirstServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 PrintWriter out = response.getWriter();
 ServletContext context = this.getServletConfig().getServletContext();
 Enumeration enum = context.getInitParameterNames();
 while(enum.hasMoreElements()) {
 String paramValue = (String)enum.getInitParameter(paramName);
 out.println(paramName+ " ... "+paramValue);
 }
 }
}
```

**web.xml**

```
<web-app>

 <context-param>
 <param-name>user</param-name>
 <param-value>scott</param-value>
 </context-param>

 <context-param>
 <param-name>pwd</param-name>
 <param-value>tiger</param-value>
 </context-param>

 <servlet>
 <servlet-name>first</servlet-name>
 <servlet-class>FirstServlet</servlet-class>
 <init-param>
 <param-name></param-name>
 <param-value></param-value>
 </init-param>
 </servlet>

 <servlet-mapping>
 <servlet-name>first</servlet-name>
 <url-pattern>/fs</url-pattern>
 </servlet-mapping>

</web-app>
```

**Note :** With in the Servlet if we can call `getInitParameter()` directly then it provides Servlet initialization parameters but not `ServletContext` parameters , hence initparameters by default Servlet initialization parameters.

**Note:** We can Access Servlet initialization parameters in the following ways .

```
String value=getInitParameter("movie");
String value=getServletConfig().getInitParameter("movie");
```

We can Access context initialization parameters in the following ways .

```
String value=getServletContext().getInitParameter("movie");
String value=getServletConfig().getServletContext().getInitParameter("movie");
```

**Note:** Whether Servlet or context init-parameters , these are deploy time constants i.e., from the Servlet , we can read their values but we are not allow to modify i.e., we have only getter()'s but not setter()'s

### Comparision between Servlet init & context initialization parameters :

Property	Servlet init-param	Servlet context-param
Declaration	By using init-param with in Servlet. <servlet> <init-param> <param-name> <param-value> </init-param> </servlet>	By using init-param with in web-app. <servlet> <context-param> <param-name> <param-value> </context-param> </servlet>
Servlet Code to Access the Parameter	String value = getInitParameter("pname"); (or) String value = getServletConfig(). getInitParameter("pname");	String value = getServletContext(). getInitParameter("pname"); (or) String value= getServletConfig(). getServletContext(). getInitParameter("pname");
Availability (Scope)	Available only for a particular Servlet in which <init-param> is declared.	Available for all Servlet's & Jsp's with in the web-application.

### Servlet Scopes :

Objective : For the fundamental Servlet attribute scopes. (Request, session & context )

1. Write Servlet code to add retrieve and remove attributes.
2. For the given scenario identify the proper scope.
3. Identify multithreading issues associated with each scope.

There are 3 types of parameters available.

1. Form parameters or Request parameters
2. Servlet Initialization parameters
3. ServletContext parameters

**Attribute :** Place holder to store the information with in the web-application.

The main purpose of these parameters are to bring information from outside environment into the Servlet , and these are read-only .

Based on our requirement we can't create a new parameter , we can't modify and we can't remove existing parameters with in the Servlet. Here we can't use parameters to store and share information between the components of application.

To handle this requirement sun people introduced attributes concept.

Based on our requirement we can create a new attribute , we can modify and remove existing attributes. Hence attributes concept is best suitable to store and share information between the components of web-application.

Based on our requirement , we have to store this attributes into the proper scopes.

There are 3 scopes are possible in the Servlets .

1. Request Scope
2. Session Scope
3. Application (or) context scope

#### Request Scope :

1. We can maintain request scope by using `ServletRequest` ( or ) `HttpServletRequest` object .
2. Request scope will start at the time of request object creation (i.e., just before starting `service()` ) and end at the time of request object destruction i.e., just after completing `service()`
3. The information stored in request scope is available for all components which are processing that request.

`ServletRequest` interface defines the following methods to perform attribute management in request scope :

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

- To add an attribute.
- If the specified attribute is already available , then the old value replaced with new value.

`public Object getAttribute(String name)`

- Returns the value associated with specified attribute.
- If the attribute is not available then we will get null.

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

This method remove an specified attribute from request scope.

```
public Enumeration getAttributeNames();
```

- This method returns all attribute names associated with request object.
- The most common area where we can use request scope is RequestDispatcher forward and include mechanisms.

**Note:**

1. Adding an attribute in a scope is called attribute binding , where as removing an attribute from a scope is called attribute unbinding.
2. In any scope , attribute name should be unique i.e., there is no chance of existing 2 attributes with in same name in the same scope.

***Session Scope :***

1. This scope is maintained by HttpSession object.
2. Session scope will start at the time of session object creation & ends at the time of session object destruction.
3. Information stored in the session scope is available for all components which are participating in that Session.

HttpSession interface defines the following methods to perform attribute management in session scope :

1. public void setAttribute(String name , Object value)
2. public Object getAttribute(String name)
3. public void removeAttribute(String name )
4. public Enumeration getAttributeNames()

**Note :**

- Once session expires we can't call these methods , Other wise we will get "IllegalStateException".

```
session.invalidate();
session.getAttribute("raja");
```

- **EX:** Login information should be available for entire session . Hence we have to store this information in the Session scope .

### Application Scope (or) Context Scope :

1. This Scope is maintained by ServletContext object.
2. Application scope will start at the time of context object creation and end at the time of Context object destruction. i.e., application scope will start at application deployment time/server start up and ends at application undeployment/server shutdown.
3. The information stored in the application scope will be available for all components of web application irrespective of end-user.

ServletContext defines the following methods for attribute management in application scope:

1. public void setAttribute(String name , Object value)
2. public Object getAttribute(String name )
3. public void removeAttribute(String name )
4. public Enumeration getAttributeNames()

### Multithreading issues associated with each scope :

1. For every request , a new request object will be created which is accessed by only current thread. Other threads are not allowed to access request scoped attributes . Hence request scoped attribute are always Thread-safe.
2. With in the same session , we can send multiple requests simultaneously by opening new browser window . Hence Session object can be accessed simultaneously by multiple threads. Hence Session scoped attributes are not Thread-safe .
3. Context scoped attributes can be accessed simultaneously by multiple threads. Hence these are not Thread-safe.
4. Instance & static variables of Servlet can be accessed simultaneously by multiple threads, hence these are not Thread-safe.
5. For every thread a separate copy of local variable will be created. Hence these local variables are Thread-safe.

Member	is Thread Safe ?
Request Scoped attributes	YES
Session Scoped attributes	NO
Context Scoped attributes	NO
Static Variables	NO
Instance Variables	NO
Local Variables	YES

Parameters are key-value pairs .

Both key & value are String objects.

Hence at the time of retrieval it is not required to perform any type -casting .

We can assign directly parameter value to the String type variable without performing any type-casting.

```
String pvalue=request.getParameter("user");
String pvalue=request.getInitParameter("user");
```

Attributes also key-value pairs but keys are String type and values can be any Object type. Hence at the time of retrieval. Compulsory we should perform type-casting.

```
String value = request.getAttribute("user");
CompiletimeError:Incompatible types
 found:java.lang.Object
 required:java.lang.String
```

```
String value=(String)request.getAttribute("user");
```

IIC: To access the value of request scoped attribute user, which of the following is valid ?

1. String user=getParameter("user"); -----> invalid
2. String user=request.getParameter("user"); -----> invalid
3. String user=request.getInitParameter("user"); -----> invalid
4. String user=request.getAttribute("user"); -----> invalid
5. String user=(String)request.getAttribute("user");-----> valid

**Example :**

```
public class FirstServlet extends HttpServlet {
 StringBuffer buffer1=new StringBuffer("raja");
 static StringBuffer buffer2=new StringBuffer();
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 StringBuffer filter=new StringBuffer();
 HttpSession session=request.getSession();
 }
}
```

The above example which of the following are Thread Safe ?

1. buffer1 X
2. buffer2 X
3. request ✓
4. filter ✓
5. session X

### Differences between parameters and attributes :

Parameters	Attributes
We can use parameters to bring information from outside environment into Servlet.	We can use attributes to share information with in the application between components.
Parameters are read-only. i.e., with in the Servlet we can perform only read-operation and we can't modify their values i.e., we have only getter methods but not Setter.	Attributes are not read-only. We have both getter & setter methods. Based on our requirement we can create a new attribute remove & replace already existing attributes.
Parameters are deployment time constants.	Attributes are not deployment time constants
Parameters are key-value pairs where both key & value String objects only. key ---->String value ---> String	Attributes are key-value pairs where key is String type but value can be any Object type . key ---->String value ---> Object
At the time of retrieval it is not required to perform any type-casting.	At the time of retrieval compulsory we should perform type-casting.

#### CustRequest.java

```
public class CustRequest extends HttpServletRequestWrapper {
 public CustRequest(HttpServletRequest request) {
 super(request);
 }
 public String getString(String word) {
 String word1=super.getParameter(word);
 if(word1.equals("java")|| word1.equals("scjp"))
 return word;
 else
 return word1;
 }
}
```

#### TargetServlet.java

```
public class TargetServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 PrintWriter out = response.getWriter();
 String word=((ServletRequest) request).getParameter("word");
 out.println("You typed :"+word);
 }
}
```

1. Usage Filters Concept in our web application is considered as following "Intercepting Filter Design pattern".
2. Usage of Wrapper concept in our web-application is considered as "decorator design pattern".

**Demo program to print all context scoped attributes:**

```
public class ContextAttributeDemo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 PrintWriter out = response.getWriter();
 out.println("<h1>Context Attributes </h1>");
 ServletContext context=getServletContext();
 context.setAttribute("course", "SCWCD");
 Enumeration e=context.getAttributeNames();
 while (e.hasMoreElements()) {
 String aname = (String) e.nextElement();
 Object avalue=context.getAttribute(aname);
 out.println(aname+.....+avalue+
);
 }
 }
}
```

**Output :**

```
org.apache.catalina.resources.....
org.apache.naming.resources.ProxyDirContext@50a649
```

```
course.....SCWCD
```

```
com.sun.faces.config.WebConfiguration.....
com.sun.faces.config.WebConfiguration@d507e9
```

```
org.apache.AnnotationProcessor.....
org.apache.catalina.util.DefaultAnnotationProcessor@1fa6d18
```

For every web-application , web-container always add some attributes in application Scope for its internal purpose. Here context Scoped attributes never be empty.

**Ex 2: Write a program to print hit count of the web-application:  
(i.e., to print no. of requests to the web-application)**

```
public class HitCountDemo extends HttpServlet {
```

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 PrintWriter out = response.getWriter();
 ServletContext context = getServletContext();
 Integer count = (Integer) context.getAttribute("hitcount");
 if (count == null) {
 count = 1;
 } else {
 count++;
 }
 context.setAttribute("hitcount", count);
 out.println("Hit Count is :" + count);
 System.out.println("count : " + count);
}
}

```

**Note:** The main limitation of this program is count value can't persist across application re-deployment and restarts.

Incrementing the count value code, we have to place in every Servlet which may result duplicate code and cause several problems. We can resolve these problems by using Listeners.

Write a program to print no. of requests send in each(current) Session.

```

public class SessionCount extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 PrintWriter out = response.getWriter();
 HttpSession session = request.getSession();
 Integer count = (Integer) session.getAttribute("hitcount");
 if (count == null)
 count = 1;
 else
 count++;
 session.setAttribute("hitcount", count);
 out.println("The no.of counts in Session is :" + count);
 }
}

```

- For every browser new Session will be created.
- Per every user count value is : 1 (first browser)

- Other browser value : 2  
(From one browser if we send any no.of requests, count value won't be increase.)
- If we open new browser(tab) from old----->file -----> new -----> window

**Write a program to print no.of Session objects created in application  
(i.e., to print total no. of users login to the application):**

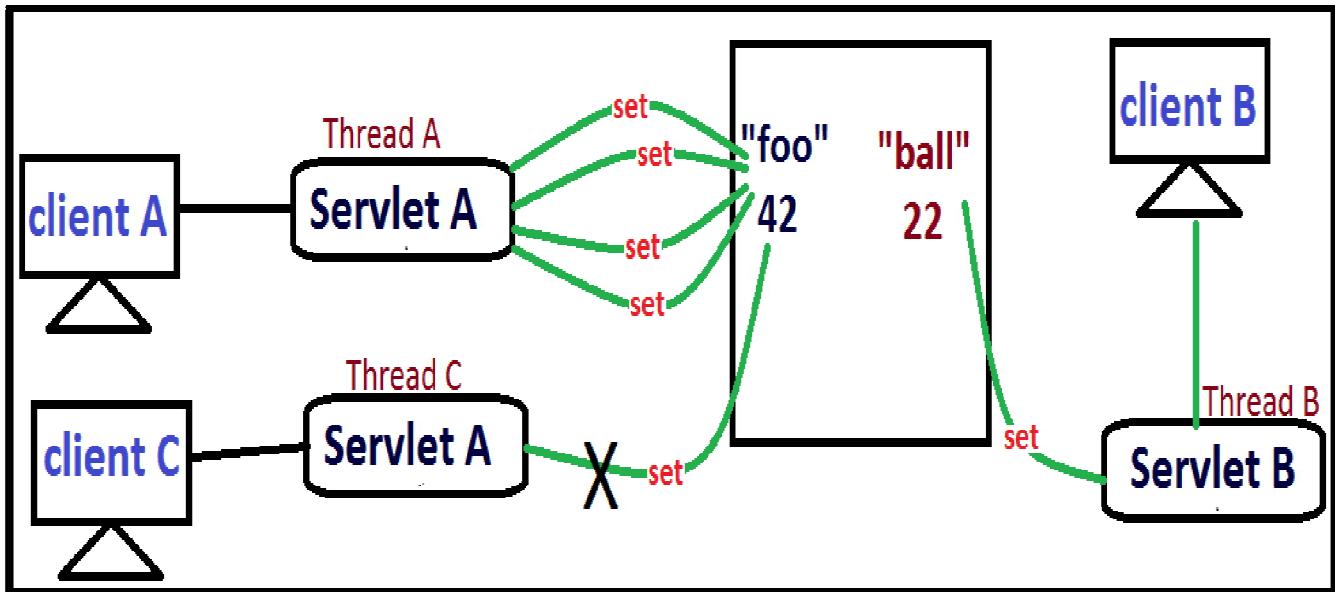
```
public class UserCount extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 PrintWriter out = response.getWriter();
 ServletContext context=getServletContext();
 Integer count=(Integer)context.getAttribute("hitcount");
 HttpSession session=request.getSession();
 if(session.isNew()) {
 if(count == null)
 count = 1;
 else
 count++;
 context.setAttribute("hitcount", count);
 }
 out.println("No of Users logged in : "+ count);
 }
}
```

**WAP to display no. of requests triggered Ip-address wise. (from same machine):**

```
public class IPAddress extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 ServletContext context=getServletContext();
 String ipaddress=request.getRemoteAddr();
 Integer count=(Integer)context.getAttribute(ipaddress);
 if(count == null) {
 count = 1;
 }
 else {
 count++;
 }
 context.setAttribute(ipaddress, count);
 out.println("The no. of requests : "+ count);
 }
}
```

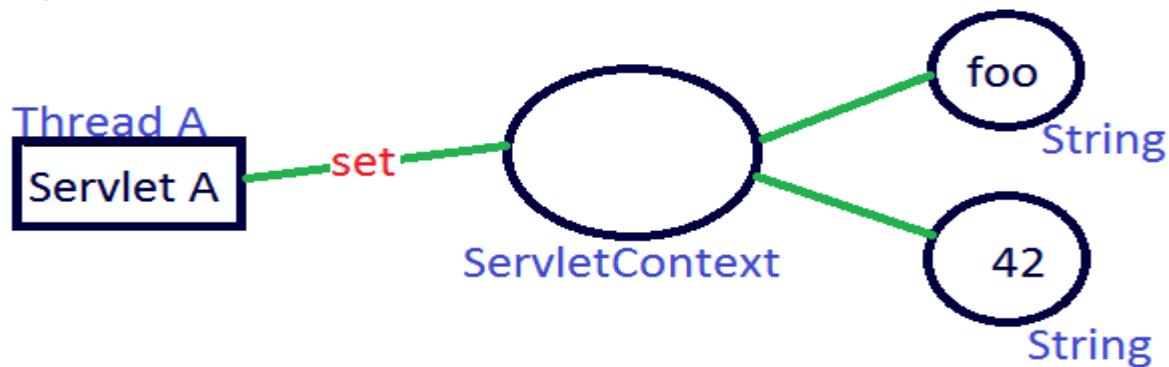
```
}
```

### How do we make Context scoped attributes Thread safe ?

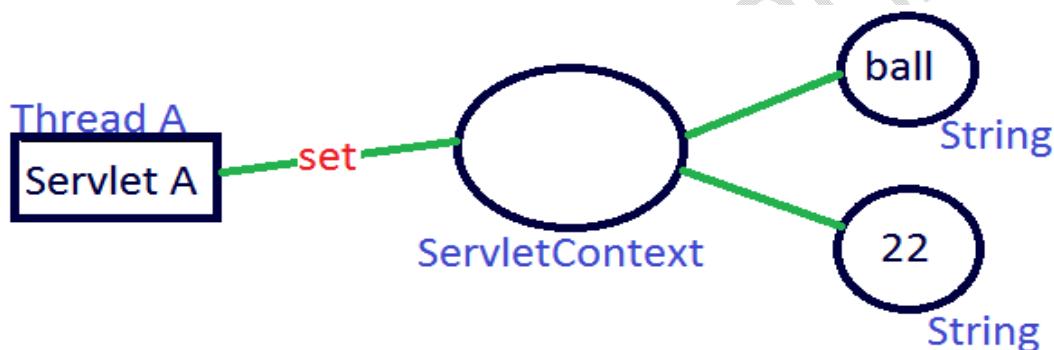


```
public class ContextScopeThreadSafe extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("The Context coped Attributes : ");
 getServletContext().setAttribute("foo", "42");
 getServletContext().setAttribute("bar", "22");
 out.println(getServletContext().getAttribute("foo"));
 out.println(getServletContext().getAttribute("bar"));
 }
}
```

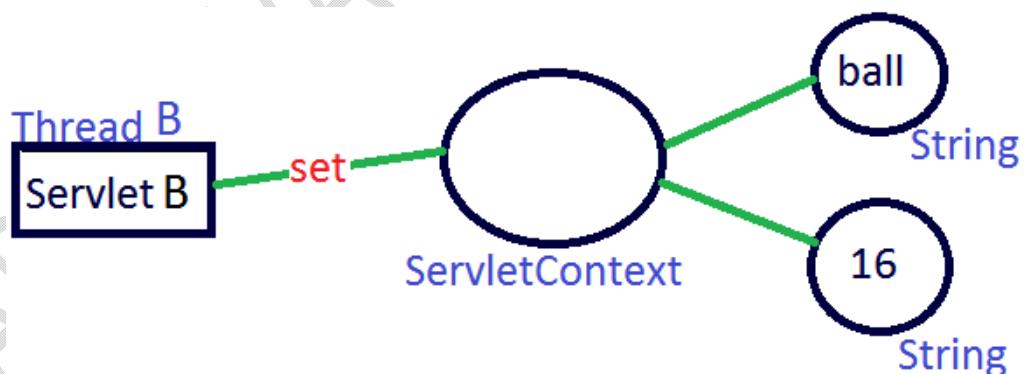
**Step 1:** Servlet A set the ContextAttribute foo with the value 42



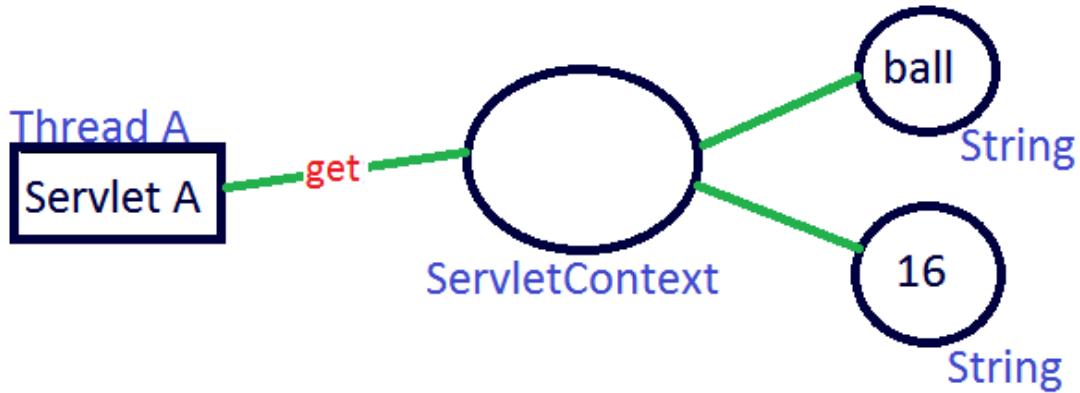
**Step 2:** Servlet A set the ContextAttribute ball with the value 22



**Step 3:** Thread B becomes the running Thread(Thread A goes back to runnable but not running) and set the context attributes ball with value of 16.



**Step 4:** Thread A become the running Thread again and get the value of ball and print it's to the response.



```

public synchronized void doGet(HttpServletRequest request,
 HttpServletResponse response) throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("The Context coped Attributes : ");

}

```

synchronized a doGet() won't product Context attributes.

synchronizing doGet() means that only one Thread of Servlet A can be running at a time it doesn't stop other servlets or jsp's from accessing context scoped attributes.

synchronizing doGet() would stop other Threads from the same Servlet from accessing context scoped attributes but it won't do anything to stop a completely different servlets or jsp's

#### Note :

We don't need lock on the servlet object , we need to lock on ServletContext.The typical way to protect context attributes synchronize on context object it self.

If every one accessing context attributes get the lock on context object then you are guarantee that only one method at a time can be getting or setting context scoped attributes.

```

public class ContextSyncronize extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Test the Context scoped Attributes :");
 }

```

```
synchronized (getServletContext()) {
 getServletContext().setAttribute("foo", "42");
 getServletContext().setAttribute("ball", "22");
 out.println(getServletContext().getAttribute("foo"));
 out.println(getServletContext().getAttribute("ball"));
}
}
}
```

Now we are getting lock on Context object , in this way to protect Context scoped attributes  
(you don't want to lock on Servlet object , synchronized(this) )



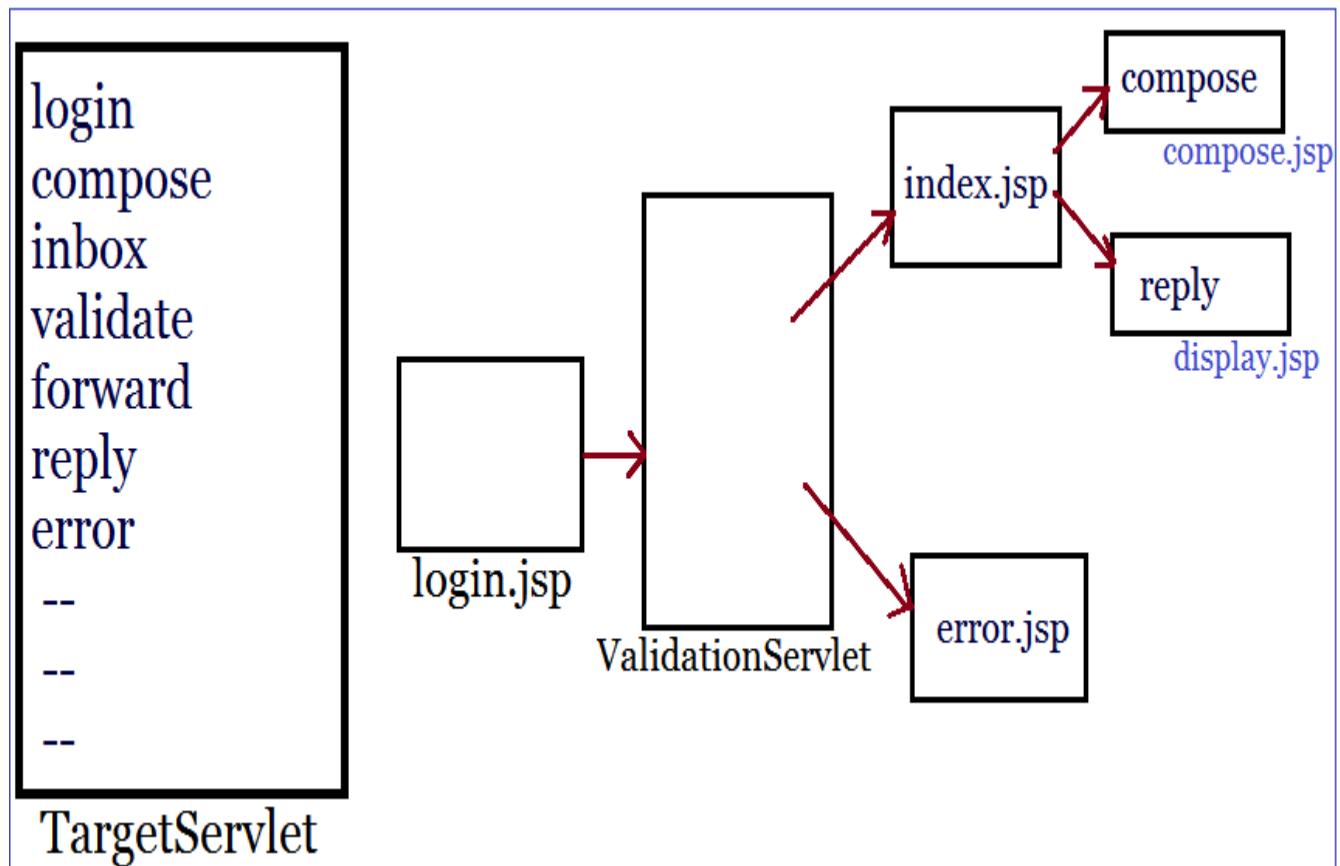
### RequestDispatcher:

#### Objective:

1. Describes RequestDispatcher mechanism.
2. Write Servlet code to create RequestDispatcher.
3. Write Servlet code to forward or include target resource.
4. Identify and describe the attributes added by web-container while forwarding and including.

It is not recommended to define entire functionality in a single component. It has several serious disadvantages:

1. For every change re-design of component is required. Hence enhancement will become very complex and creates maintainability problems.
2. It doesn't promote re-usability of the code.  
We can resolve these problems by maintaining a separate component for each task.



**The main advantages of this approach is :**

1. With out effecting remaining components , we can modify any component. Hence enhancement will become very easy and improves maintainability.
2. This approach promotes re-usability of the code.

**Example:**

1. Where ever validation logic is required we can reuse same validation logic without re-writing.
2. If the functionality is distributed across several components, these components have to communicate with each other to provide response to end-user. We can achieve this communication by using RequestDispatcher.

Servlet code to get RequestDispatcher :

We can get RequestDispatcher either by ServletRequest object (or) by ServletContext object.

By ServletRequest :

ServletRequest interface defines the following method to get RequestDispatcher object.

```
public RequestDispatcher getRequestDispatcher("String target")
```

The target resource path can be specified either by absolute path (or) relative path.

Example:

```
RequestDispatcher rd=request.getRequestDispatcher("/test2"); //absolute path
RequestDispatcher rd=request.getRequestDispatcher("test2"); //relative path
```

If the target resource is not available then we will get 404 status code saying requested resource is not available.

By ServletContext object :

ServletContext interface defines the following 2 methods to get the RequestDispatcher object.

```
public RequestDispatcher getRequestDispatcher("String target");
```

Target resource path compulsory we should specifying by using absolute path. If we are using relative path then we will get IllegalArgumentException.

Ex:

```
RequestDispatcher rd=context.getRequestDispatcher("/test2"); //valid because absolute path
RequestDispatcher rd=context.getRequestDispatcher("test2"); //IllegalArgumentException
because relative path
```

If the target resource is not available , then we will get 404 status code.

```
public RequestDispatcher getNamedDispatcher("String servletName") ; //in this servletName is
logicalname
```

1. If URL pattern is not defined for the target Servlet then we have to use this method.

2. The argument represents the value associated with Servlet name tag in web.xml which represents logical name of the Servlet.
3. If the target Servlet is not available then we will get null but not 404 status code .On that null if we are trying to perform forward (or) including we will NullPointerException.

Ex:

```
RequestDispatcher rd=context.getNamedDispatcher("targetServlet"); //in this targetServlet
is logicalname
```

#### Differences between getting RequestDispatcher by ServletRequest & ServletContext :

RequestDispatcher from ServletRequest	RequestDispatcher from ServletContext
<pre>RequestDispatcher rd = request.getRequestDispatcher("/test2");</pre> <pre>RequestDispatcher rd = request.getRequestDispatcher("test2");</pre> The target resource can be specified either by absolute path (or) relative path.	<pre>RequestDispatcher rd = context.getRequestDispatcher("/test2");</pre> The target resource should be specified only by absolute path but not relative path. Otherwise we will get IllegalArgumentException.
We can use this RequestDispatcher to communicate only with in the same application	We can use this to communicate either with in the same or outside of the application . But recommended to use to communicate outside of the web-application
We can't get RequestDispatcher by <servlet-name> tag	We can get RequestDispatcher by <servlet-name> for this we have getNamedDispatcher()

Note : <servlet-mapping> tag is optional to configure servlet with in the web.xml , that time nobody can access that servlet directly from browser, in that case the remaining servlets can communicate by using it's logical name.

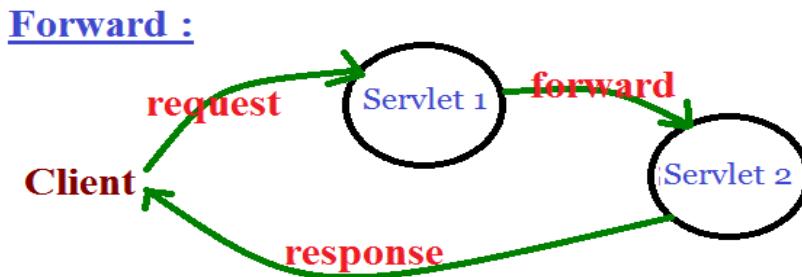
#### RequestDispatcher's methods :

RequestDispatcher interface defines the following 2 methods.

1. public void forward(ServletRequest request, ServletResponse response) throws ServletException, IOException

2. `public void include(ServletRequest request, ServletResponse response) throws ServletException, IOException`

### Forward mechanism :



1. If the 1<sup>st</sup> Servlet is responsible for some preliminary processing & 2<sup>nd</sup> Servlet is responsible to provide requirement response then we should go for forward mechanism.
2. Just before forwarding response object will be cleared automatically by the web-container . Hence if any response added by the first servlet won't be delivered to the end-user. Only second servlet response will be displayed.
3. In forward mechanism second servlet has complete control on response object. Hence it can change response headers like `contentType()` .
4. In forward mechanism , the same request object will be forwarded to the second servlet. Hence information sharing between the components is always possible in the form of request scoped attributes.
5. After forwarding the request to second servlet the control will come to the first servlet again to execute remaining statements. In the remaining statements if we are trying to write anything to the response these statements will be ignored by the web-container. In the remaining statements if any exception raised only that exception information will be displayed for end user instead of second servlet response . i.e., Second servlet response will be delivered after completing remaining statements execution only.

Ex:

```

public class ForwardServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Hello");
 RequestDispatcher rd=request.getRequestDispatcher("/forward2");
 rd.forward(request,response);
 }
}

```

```

 out.println("Hi");
 System.out.println(10/0); //-->1
 }
}

public class ForwardTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("This is Second Forward Servlet ");
 }
}

```

- If we are sending the request to the FirstServlet, ArithmeticException will be displayed for the end-user instead of second Servlet response.
- If we are commenting line1, then only 2<sup>nd</sup> Servlet response will be displayed for the end-user.
- Recursive forward call always raises RuntimeException saying StackOverflowError.

After committing the response , we can't perform forward (or) redirection . Otherwise we will get RuntimeException saying java.lang.IllegalStateException

Ex:

```

public class ForwardTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("This is Forward Servlet ");
 out.flush();
 RequestDispatcher rd=request.getRequestDispatcher("/forward");
 rd.forward(request, response);
 // ---life cycle will start to the second servlet
 }
}

```

We can pass Query String as part of forward mechanism

```

public class ForwardTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("This is Forward Servlet ");
 RequestDispatcher rd = request.getRequestDispatcher("/forward?uname=raja&pwdscwcd");
 rd.forward(request, response);
 }
}

```

```

 }
}

public class ForwardServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 String name=request.getParameter("uname");
 String pwd=request.getParameter("pwd");
 out.println(name+"....."+pwd);
 }
}

```

**login.html**

```

<form action="/servlet/vs">
 <table>
 <tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
 <tr><td>Password : </td><td> <input type="text" name="pwd"></td></tr>
 <tr><td><input type="submit" value="submit"></td></tr>
 </table>
</form>

```

**ValidateServlet.java**

```

public class ValidateServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Validate Servlet");
 String name=request.getParameter("uname");
 String pwd=request.getParameter("pwd");
 if(name.equalsIgnoreCase("raja")&&pwd.equalsIgnoreCase("scwcd")) {
 RequestDispatcher rd=getServletContext().getRequestDispatcher("/inbox.jsp");
 rd.forward(request, response);
 }
 else {
 RequestDispatcher rd=request.getRequestDispatcher("error.jsp");
 rd.forward(request, response);
 }
 }
}

```

**error.jsp**

Invalid Credentials , please login here  
[LOGIN](login.html)

**index.jsp**

## <h1>Successful Login</h1>

- Attributes added by web-container while forwarding the request.
- While forwarding the request from one Servlet to another , web-container will always add some attributes in request scope to provide Original request information to the second Servlet.

The following is the list of attributes added by web-container:

1. javax.servlet.forward.request-uri
2. javax.servlet.forward.context-path
3. javax.servlet.forward.servlet-path
4. javax.servlet.forward.path-info
5. javax.servlet.forward.query-string

Note:

1. If we are getting RequestDispatcher by using getNamedDispatcher , the web-container won't add any attributes in request scope.
2. With in the same Servlet we can't call forward more than once directly . Otherwise we will get IllegalStateException.

## Include mechanism :

1. In several components required some common functionality, Its never recommended to hardcode that functionality in every component. We have to separate component and we have to make that functionality available by using include mechanism for every component.
2. This approach promotes code re-usability and reduces maintainability problems.
3. Hence the main objective of include mechanism is to include the response of other resources in the current response . This include is best suitable for JSP to include Headers and Footers , Banner information and Logos etc.
4. In the include mechanism the first Servlet is responsible to provide total response . But it can include the response of some other components.
5. In the Include, second servlet doesn't have complete control on the response object. It is not allowed to change response headers , If it is trying to change these changes will be ignored by web-container.

Ex:

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

```

 out.println("First Include Servlet
");
 RequestDispatcher rd=request.getRequestDispatcher("/include2");
 rd.include(request, response);
 out.println("First Include Servlet Again ");
 }
}

public class IncludeTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Include Second Servlet
");
 }
}

```

When ever we are sending the request to IncludeServlet the following is the output :

First Include Servlet  
 Include Second Servlet  
 First Include Servlet Again

After committing the response we can call include() But we can't call forward() and sendRedirect() .

With in the Servlet we can call include any no. of times . But we can call forward() only once and mostly as last statement.

#### Attributes added by Web-container while performing include :

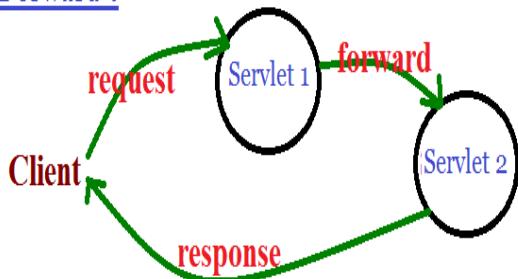
While performing include web-container will add the following attributes in request scope to make original request information available to the second Servlet.

1. javax.servlet.include.request-uri
2. javax.servlet.include.context-path
3. javax.servlet.include.servlet-path
4. javax.servlet.include.path-info
5. javax.servlet.include.query-string

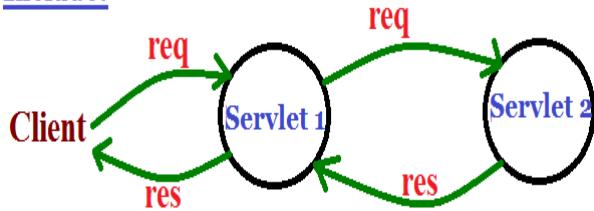
Note: When ever we are getting RequestDispatcher by using getNamedDispatcher() , then web-container won't add these attributes in request scope.

#### Differences between forward() and include () :

forward()	include()
-----------	-----------

Forward :

s1 ---->forwarding Servlet  
s2 ---->forwarded Servlet

Include:

s1 ----> including Servlet  
s2 ----> included Servlet

Once first Servlet forwards the request to the second Servlet , second Servlet is completely control on response object , provide the response.

In the case of include first Servlet is responsible to provide required response.

While performing forward call the response object will be cleared automatically. Hence FirstServlet response won't be displayed for the end-user.

While performing include, the response object won't be cleared. Hence first servlet response also will be displayed for the end-user.

After committing the response we can't perform forward. Otherwise we will get IllegalStateException.

After committing the response we can call include mechanism.

With in the same Servlet we can call forward() only once and mostly as the last statement.

With in the Servlet we can call include( ) any no. of times.

In the case of forward() , second Servlet has complete control on the response object.

In the case of include , 2<sup>nd</sup>Servlet doesn't have complete control on the response object. It is not allowed to change response headers.

forward mechanism we can use frequently in Servlets because it is associated with processing.

Include mechanism we can use frequently in JSP's , Because it is associated with view part.

**Foreign RequestDispatcher :****Inside Servlet1 :**

```
ServletContext context=getServletContext();
ServletContext fcontext=context.getServletContext("/webapp2");
RequestDispatcher rd=fcontext.getRequestDispatcher("/test2");
rd.forward(req , res);
```

1. By default most of the webservers including Tomcat won't provide support for cross context communication.
2. In this case getRequestDispatcher() returns null, on that null if we are trying call forward() (or ) include() we will get NullPointerException.
3. To provide support for cross context communication at Server level some configuration changes are required.
4. As RequestDispatcher mechanism will work with in the same server. Hence both the applications should be deployed in the same Server.

**Important Interview Questions :**

1. What is the purpose of RequestDispatcher ?
2. Explain RequestDispatcher mechanism ?
3. Differences between context.getRequestDispatcher() & context.getNamedDispatcher( ) ?
4. Various possible ways to get RequestDispatcher ?
5. Differences betwee forward & sendRedirection ?
6. What is the Difference betwee forward & include ?
7. What is foreign RequestDispatcher and how we can get ?
8. Explain RequestDispatcher mechanism between 2 applications of the same Server ?
9. What are the attributes added by the web-container while forwarding and including the request ?
10. What is the purpose of these attributes & explain its meaning ?

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

# **JAVA MEANS DURGASOFT**

**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED

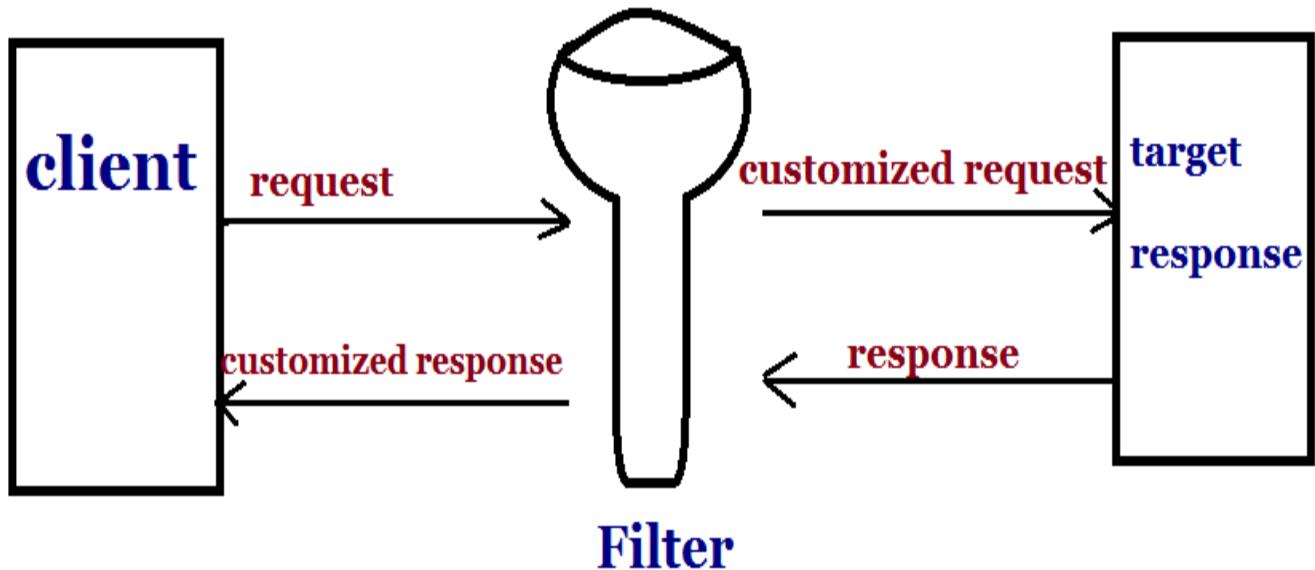
**DURGA**  
SOFTWARE SOLUTIONS

# 202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

**FILTERS (servlet 2.3 v) :****Objective :**

1. Describe web-container request processing model.
2. Write and configure a filter.
3. Create a request and response Wrapper for the given design problem.
4. If we want to perform any activity at the time of pre-processing and post-processing of the request. Then we should go for Filters.
5. This concept introduced in Servlet 2.3 version.

**The most common application areas of Filter are...**

1. Logging
2. Security checks like Authentication & Authorization
3. Altering request information
4. Altering response information
5. Compression of response
6. Encryption of response etc.

**FILTER- API :**

We can develop Filters concept by using the following 3 interfaces.

- 1) Filter
- 2) FilterConfig
- 3) FilterChain

### Filter :

**Every Filter in java should implement Filter interface either directly or indirectly.**

Filter interface defines the following 3 methods.

1. init() :

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

This method will be executed only once to perform initialization activity after instantiation immediately.

2. destroy() :

```
public void destroy()
```

This method will be executed only once to perform CleanUp activities , just before taking the Filter from out of Service.

3. doFilter() :

```
public void doFilter(ServletRequest request , ServletResponse response, FilterChain chain) throws ServletException, IOException
```

This method will be executed for every request, Entire Filtering logic we have to define in this method only.

We can use FilterChain object to forward the request to the next level . It can be another Filter (or) Servlet.

### FilterConfig :

- For every Filter, web-container creates one FilterConfig object to hold its configuration information.
- Web-container handover FilterConfig object to the Filter as argument to init() method

FilterConfig interface defines the following 4 methods :

1. **public String getFilterName();**  
Returns logical name of the Filter configured in web.xml by using <filter-name> tag .
2. **public String getInitParameter(String parameter)**
3. **public Enumeration getInitParameterNames()**
4. **public ServletContext getServletContext()**

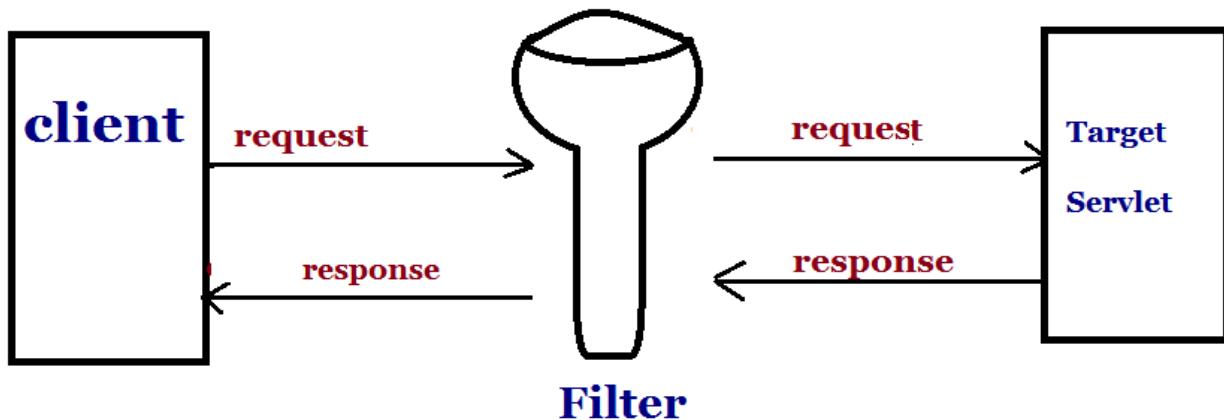
**FilterChain :**

- We can use FilterChain to forward the request to the next level.[It may be another filter (or) Servlet ].
- FilterChain interface defines the following doFilter().

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

```
public class FirstFilter implements Filter {
 public void init(FilterConfig config) throws ServletException {}
 public void doFilter(ServletRequest request, ServletResponse response,
 FilterChain chain) throws IOException, ServletException {
 PrintWriter out = response.getWriter();
 out.println("This is First Filter before processing the request i.e., pre-processing");
 chain.doFilter(request, response);
 out.println("This is First Filter after processing the request i.e., post-processing");
 }
 public void destroy() {}
}

public class TargetServletFilter extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("This is Target Servlet Filter ");
 }
}
```

**Analysis:**

1. Whenever we are Sending the request to the Servlet , web-container checks is any Filter configured for this Servlet (or) not
2. If any Filter is configured web-container forwards the request to the Filter instead of Servlet.
3. After completing Filtering logic Filter forwards the request to the TargetServlet.
4. After processing the request by TargetServlet the response will be forwarded to the Filter instead of browser.
5. After executing Filtering logic , Filter forwards the total response to the end-user .

***Configuring Filter in web.xml :***

```

<servlet>
 <servlet-name>TargetServletFilter</servlet-name>
 <servlet-class>jobs.TargetServletFilter</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>TargetServletFilter</servlet-name>
 <url-pattern>/tsfilter</url-pattern>
</servlet-mapping>
<filter>
 <filter-name>firstfilter</filter-name>
 <filter-class>jobs.FirstFilter</filter-class>
</filter>
<filter-mapping>
 <filter-name>firstfilter</filter-name>
 <url-pattern>/tsfilter</url-pattern>
</filter-mapping>

```

**Note:** The life cycle of a Filter will start at the time of application deployment (or) Server startup.i.e., Filter class loading, instantiation and execution of init() will be performed automatically at the time of either application deployment or Server StartUp. Hence load-on-

startup concept is not applicable for Filters.

Web-container is responsible to perform instantiation of the Filter for this it always calls public no-argument constructor . Hence every Filter class should compulsory contain public no-argument constructor.

### ***Mapping of Filter :***

We can map a Filter either for a particular url-pattern (or) to a particular Servlet (or) to the whole web-application.

#### ***Mapping to a particular url-pattern :***

```
<filter-mapping>
 <filter-name>DemoFilter</filter-name>
 <url-pattern>/test</url-pattern>
</filter-mapping>
```

When ever we are sending a request for with specified url-pattern then only Filter will be executed.

#### ***Mapping to a particular Servlet :***

```
<filter-mapping>
 <filter-name>DemoFilter</filter-name>
 <servlet-name>TargetServlet</servlet-name>
</filter-mapping>
```

Once Servlet got the request automatically Filter will be executed.

#### ***Filter mapping to entire web-application :***

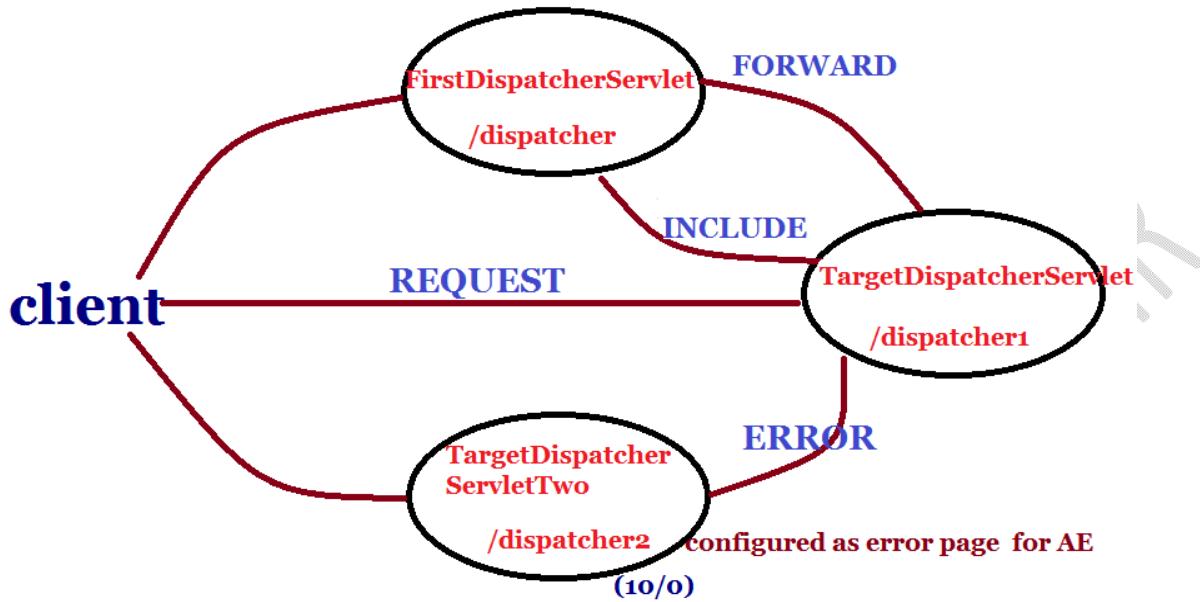
For any request to the web-application , whether it is for Servlet or JSP , this Filter will be executed.

**Note:** Mapping a Filter to the whole web-application is possible from Servlet 2.5 version onwards only.



**<dispatcher> :**

A servlet can get the request in one of the following possible ways.



**A request directly from the browser. (REQUEST)**

**By RequestDispatcher's forward call. (FORWARD)**

**By RequestDispatcher's include call. (INCLUDE )**

**By RequestDispatcher's error call. (ERROR)**

- By default Filter will be executed only for direct end-user request, and won't be executed for RequestDispatcher's forward & include and error-page calls.
- But in Servlet 2.4 version Sun people introduced <dispatcher> to extend Filter concepts for remaining cases also. i.e., for which type of request Filter will be executed is decided by < dispatcher > tag.

The allowed values for <dispatcher > tag are...

- REQUEST
- FORWARD
- INCLUDE
- ERROR

**Ex:** If we want to execute Filter for direct end-user request and for RequestDispatcher's forward call. We have to configure <filter-mapping> tag as follows.

```

<filter-mapping>
 <filter-name>DemoFilter</filter-name>
 <url-pattern>/test</url-pattern>
 <dispatcher>REQUEST</dispatcher>

```

```
<dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

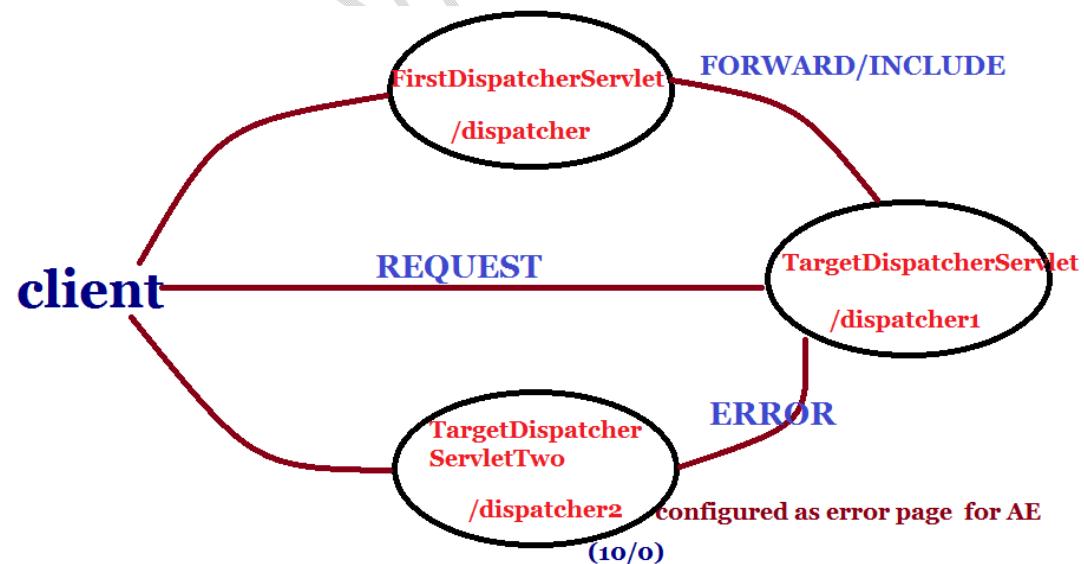
In this case Filter won't be executed for RequestDispatcher's INCLUDE call and ERROR page call.

Ex 2 : To execute a Filter only for RequestDispatcher's include call we have to configure <filter-mapping> as follows.

```
<filter-mapping>
 <filter-name>
 <url-pattern>
 <dispatcher>INCLUDE</dispatcher>
 </filter-mapping>
```

When ever we are Using <dispatcher> tag, then default behaviour won't be reflected. i.e., we are Overriding default-behaviour by using <dispatcher> tag.

#### Demo Program :



**FirstDispatcherServlet.java**

```
public class FirstDispatcherServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("First Dispatcher Servlet Pre-processing
");
 RequestDispatcher rd=request.getRequestDispatcher("/dispatcher1");
 rd.include(request, response);
 out.println("
First Dispatcher Servlet Post-processing
");
 }
}
```

**TargetDispatcherServlet.java**

```
public class TargetDispatcherServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 for(int i=0;i<5;i++)
 out.println("
This is Target Dispatcher Servlet"+i+"
");
 }
}
```

**TargetDispatcherServletTwo.java**

```
public class TargetDispatcherServletTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println(10/0);
 }
}
```

**DispatcherFilter.java**

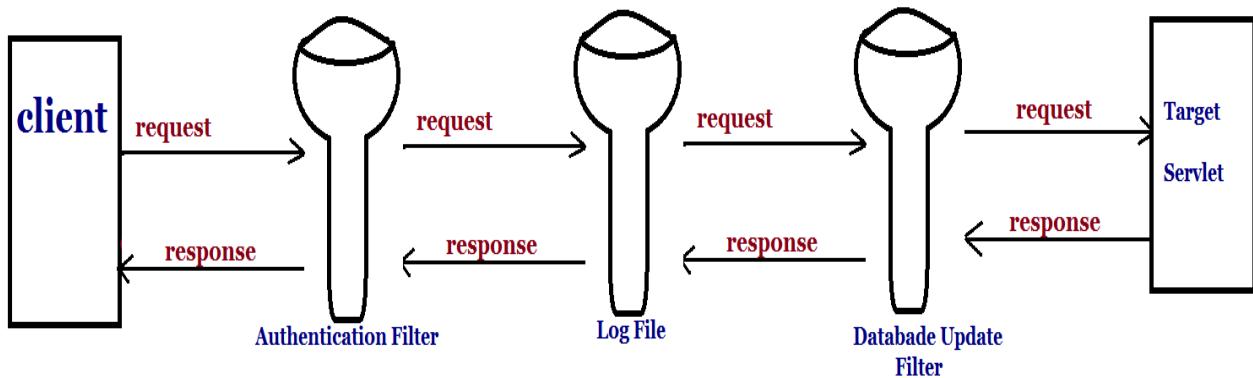
```
public class DispatcherFilter implements Filter {
 public void destroy() {
 System.out.println("Destroy method");
 }
 public void doFilter(ServletRequest request, ServletResponse response,
 FilterChain chain) throws IOException, ServletException {
 PrintWriter out=response.getWriter();
 out.println("
This is First Filter before processing the request i.e., pre-processing
");
```

```

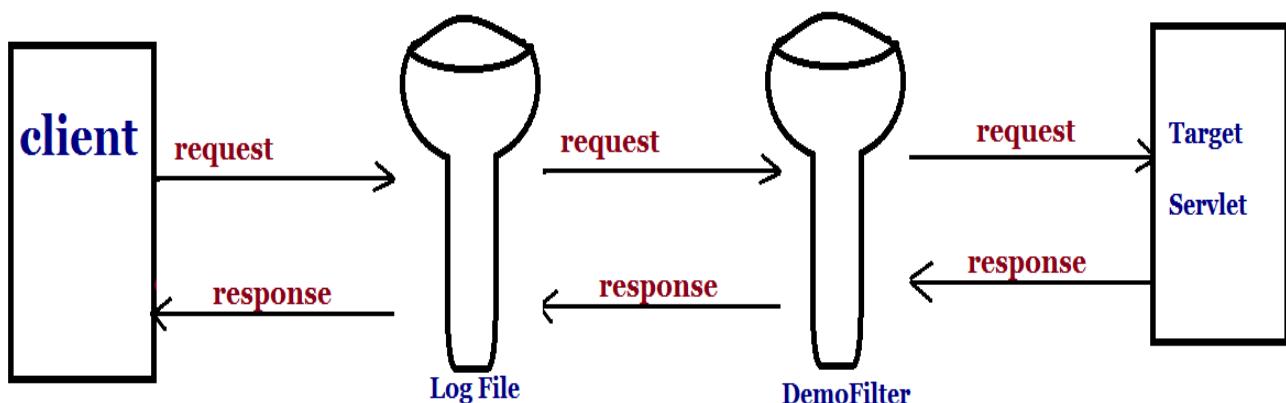
 chain.doFilter(request, response);
 out.println("
This is First Filter after processing the request i.e., post-processing
");
 }
 public void init(FilterConfig config) throws ServletException {
 System.out.println("Filter init method");
 }
}

```

We can configure more than one Filter for a TargetServlet and all these Filters will be executed one by one and forms FilterChain.



Demo program :



Note : The log information will be stored in the following file locating in logs folder of Tomcat.(localhost.2013.04.16.txt.....date)

Web-container's rule for ordering of Filters in FilterChain :

1. Identify all Filters which are configured according to url-pattern and Execute all these filters from top to bottom .
2. Identify all Filters which are configured according to servlet-name and execute all these Filters from top- to- bottom.

i.e., The Filters which are configured by url patterns will get high priority when compared with Filters which are configured by servlet-name.

Ex:

```
<filter-mapping>
 <filter-name>Filter1</filter-name>
 <url-pattern>/Recipes/*</url-pattern>
</filter-mapping>

<filter-mapping>
 <filter-name>Filter2</filter-name>
 <url-pattern>/Recipes/HotList.do</url-pattern>
</filter-mapping>

<filter-mapping>
 <filter-name>Filter3</filter-name>
 <url-pattern>/Recipes/Add/*</url-pattern>
</filter-mapping>

<filter-mapping>
 <filter-name>Filter4</filter-name>
 <url-pattern>/Recipes/Modify/Modify.do</url-pattern>
</filter-mapping>

<filter-mapping>
 <filter-name>Filter5</filter-name>
 <url-pattern>*</url-pattern>
</filter-mapping>
```

Request URI	Order of Execution
/Recipes/HotList.do	Filter 1,5,2
/Recipes/HotReport.do	Filter 1,5
/HotList.do	Filter 5
/Recipes/Modify/Modify.do	Filter 1,5,4
/Recipes/Add/AddRecipes.do	Filter 1,3,5

**Difference between Filter's doFilter() and FilterChain's doFilter() :**

Filter's doFilter()	FilterChain's doFilter()
public void doFilter(ServletRequest request , ServletResponse response , FilterChain fc)	public void doFilter(ServletRequest request , ServletResponse response ) throws

<b>throws ServletException, IOException</b>	<b>ServletException, IOException</b>
We can use this method to define entire filtering logic.	We can use this method to forward request to the next level.
This doFilter() is a callback method because web-container will call this method automatically for every request.	It is a inline method because we have to call this method explicitly then only it will be executed .

**www.durgasoftonlinetraining.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428  
USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

### Wrappers :

- Some times it is required to alter request and response information in filters , we can achieve this by using Wrapper class.  
 Ex 1: With in the filter we have to convert end-users resume format from PDF to DOC file , We can achieve this by Wrapper.  
 Ex 2: In the filter we have to compress the response and we can send that compressed response to the end-user. So that we can reduce download time, We can achieve this by using Wrapper.

There are 2 types of Wrapper classes

#### 1. RequestWrapper

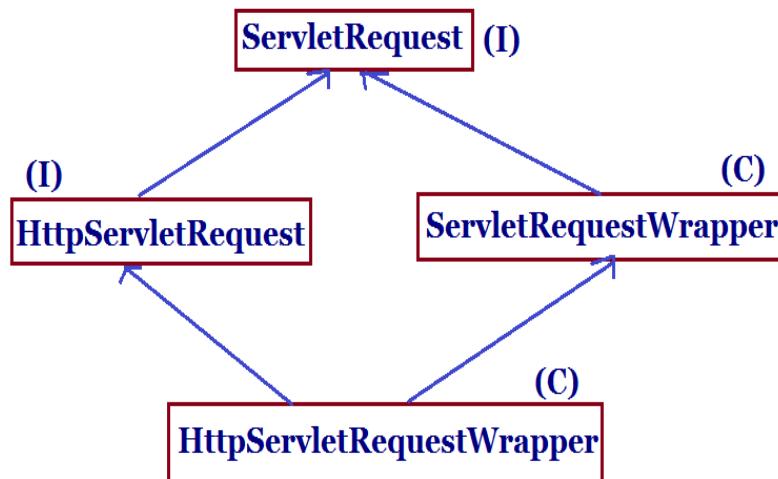
## 2. ResponseWrapper

### RequestWrapper :

- To alter request information

There are 2 types of RequestWrapper classes

1. ServletRequestWrapper
2. HttpServletRequestWrapper

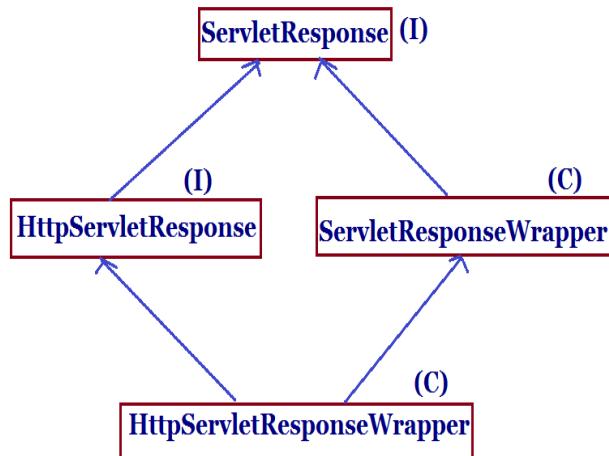


### ResponseWrappers :

- To alter response information

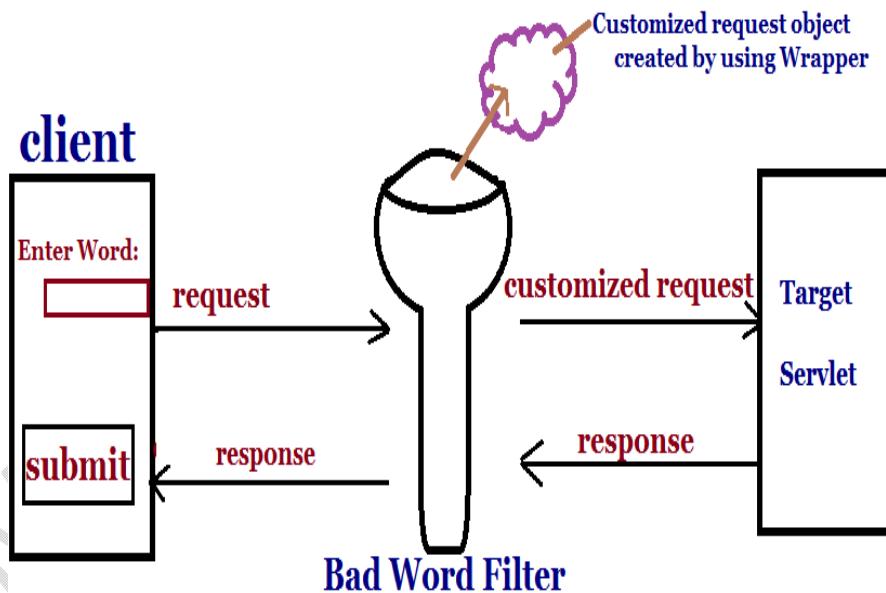
There are two types of Response Wrappers.

1. ServletResponseWrapper
2. HttpServletResponseWrapper



### Demo program for RequestWrapper :

(Sending the duplicate manager instead of original manager to the estate )



1. End user enter the word and click the submit button.
2. Web-container forwards the request to the filter instead of Servlet.
3. Filter creates a customized request object by using Wrapper.
4. Filter forwards that customized request object to the target Servlet instead of Original request.
5. With in the target Servlet if we are applying any operation on the request object our own customized behaviour will be reflected. but not original request behaviour.
6. Target Servlet prepares the response & forward to filter instead of browser.

## 7. Filter forwards that response to the end-user.

### **Demo Program :**

wrapper.html

```
<form action=".(wrapper">
 Enter Word : <input type="text" name="word">

 <input type="submit" value="submit">
</form>
```

BadWordWrapperFilter.java

```
public class BadWordWrapperFilter implements Filter {
 public void destroy() {
 System.out.println("destroy method");
 }
 public void doFilter(ServletRequest request, ServletResponse response,
 FilterChain chain) throws IOException, ServletException {
 CustomizeWrapperRequest myRequest = new
 CustomizeWrapperRequest((HttpServletRequest) request);
 chain.doFilter(myRequest, response);
 }
 public void init(FilterConfig config) throws ServletException {
 System.out.println("init method");
 }
}
```

CustomizeWrapperRequest.java

```
public class CustomizeWrapperRequest extends HttpServletRequestWrapper {
 public CustomizeWrapperRequest(HttpServletRequest request) {
 super(request);
 }
 public String getParameter(String x) {
 String word1=super.getParameter(x);
 if(word1.equalsIgnoreCase("java") || word1.equalsIgnoreCase("scjp")) {
 return "SLEEP";
 }
 else {
 return word1;
 }
 }
}
```

TargetWrapperServlet.java

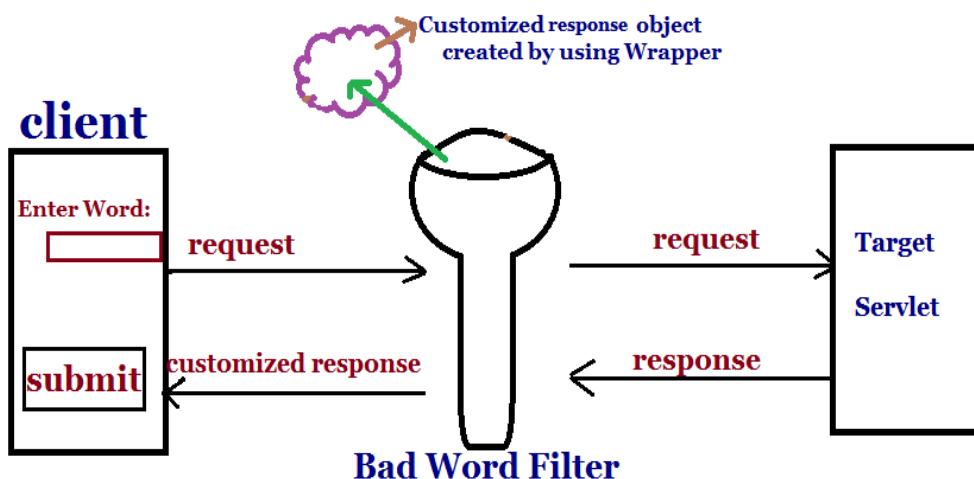
```
public class TargetWrapperServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
```

```

 PrintWriter out = response.getWriter();
 String word=request.getParameter("word");
 out.println("The Entered word is :" + word);
 }
}

```

### Write a Demo program for response wrapper :



### **ResponseWrapperFilter.java**

```

public class ResponseWrapperFilter implements Filter {
 public void destroy() {
 System.out.println("Response Wrapper destroy");
 }
 public void doFilter(ServletRequest request, ServletResponse response,
 FilterChain chain) throws IOException, ServletException {
 CustomizeWrapperResponse myReponse=new CustomizeWrapperResponse
 ((HttpServletResponse) response);
 chain.doFilter(request, myReponse);
 String text=myReponse.toString();
 if(text != null)
 text = text.toUpperCase();
 response.getWriter().write(text);
 }
 public void init(FilterConfig config) throws ServletException {
 System.out.println("Response Wrapper init");
 }
}

```

### **CustomizeWrapperResponse.java**

```

public class CustomizeWrapperResponse extends HttpServletResponseWrapper {

```

```

protected CharArrayWriter charWriter;
protected PrintWriter writer;
public CustomizeWrapperResponse(HttpServletRequest response) {
 super(response);
 charWriter=new CharArrayWriter();
}
public PrintWriter getWriter() throws IOException {
 if(writer!=null){
 return writer;
 }
 return writer=new PrintWriter(charWriter);
}
public String toString() {
 String text=null;
 if(writer!=null) {
 text=charWriter.toString();
 }
 return text;
}
}

```

**TargetResponseWrapperServlet.java**

```

public class TargetResponseWrapperServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter(); //customize response
 out.println("Hi Ashok , Wrappers are very easy ");
 }
}

```

In the above program some implementations required

- We have to override `getOutputStream()` method.
- Suppose if we are getting `PrintWriter` and `ServletOutputStream` , we have to handle "IllegalStateException".
- Usage Filter concept in our web application is considered as following "Intercepting Filter Design Pattern".
- Usage of wrapper concept in our web application is considered as "Decorator Design Pattern".

***all programs web.xml***

<web-app>

<servlet>

```
<servlet-name>FirstServlet</servlet-name>
<servlet-class>jobs.FirstServlet</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>FirstServlet</servlet-name>
 <url-pattern>/fs</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>CustRequest</servlet-name>
 <servlet-class>jobs.CustRequest</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>CustRequest</servlet-name>
 <url-pattern>/cr</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>TargetServlet</servlet-name>
 <servlet-class>jobs.TargetServlet</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>TargetServlet</servlet-name>
 <url-pattern>/ts</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>ContextAttributeDemo</servlet-name>
 <servlet-class>jobs.ContextAttributeDemo</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>ContextAttributeDemo</servlet-name>
 <url-pattern>/cad</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>HitCountDemo</servlet-name>
 <servlet-class>jobs.HitCountDemo</servlet-class>
</servlet>
```

```
<servlet-mapping>
 < servlet-name>HitCountDemo</servlet-name>
 < url-pattern>/hcd</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name>SessionCount</servlet-name>
 < servlet-class>jobs.SessionCount</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name>SessionCount</servlet-name>
 < url-pattern>/sc</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name>UserCount</servlet-name>
 < servlet-class>jobs.UserCount</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name>UserCount</servlet-name>
 < url-pattern>/uc</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name>IPAddress</servlet-name>
 < servlet-class>jobs.IPAddress</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name>IPAddress</servlet-name>
 < url-pattern>/ip</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name>ContextScopeThreadSafe</servlet-name>
 < servlet-class>jobs.ContextScopeThreadSafe</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name>ContextScopeThreadSafe</servlet-name>
```

```
<url-pattern>/csts</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>ContextSyncronize</servlet-name>
 <servlet-class>jobs.ContextSyncronize</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>ContextSyncronize</servlet-name>
 <url-pattern>/cs</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>ForwardServlet</servlet-name>
 <servlet-class>jobs.ForwardServlet</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>ForwardServlet</servlet-name>
 <url-pattern>/forward</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>ForwardTwo</servlet-name>
 <servlet-class>jobs.ForwardTwo</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>ForwardTwo</servlet-name>
 <url-pattern>/forward2</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>ValidateServlet</servlet-name>
 <servlet-class>jobs.ValidateServlet</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>ValidateServlet</servlet-name>
 <url-pattern>/vs</url-pattern>
</servlet-mapping>
```

```
<servlet>
 < servlet-name >IncludeServlet</servlet-name>
 < servlet-class >jobs.IncludeServlet</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name >IncludeServlet</servlet-name>
 < url-pattern >/include</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name >IncludeTwo</servlet-name>
 < servlet-class >jobs.IncludeTwo</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name >IncludeTwo</servlet-name>
 < url-pattern >/include2</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name >TargetServletFilter</servlet-name>
 < servlet-class >jobs.TargetServletFilter</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name >TargetServletFilter</servlet-name>
 < url-pattern >/tsfilter</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name >FirstDispatcherServlet</servlet-name>
 < servlet-class >jobs.FirstDispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
 < servlet-name >FirstDispatcherServlet</servlet-name>
 < url-pattern >/dispatcher</url-pattern>
</servlet-mapping>

<servlet>
 < servlet-name >TargetDispatcherServlet</servlet-name>
 < servlet-class >jobs.TargetDispatcherServlet</servlet-class>
```

```
</servlet>

<servlet-mapping>
 <servlet-name>TargetDispatcherServlet</servlet-name>
 <url-pattern>/dispatcher1</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>TargetDispatcherServletTwo</servlet-name>
 <servlet-class>jobs.TargetDispatcherServletTwo</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>TargetDispatcherServletTwo</servlet-name>
 <url-pattern>/dispatcher2</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>TargetWrapperServlet</servlet-name>
 <servlet-class>jobs.TargetWrapperServlet</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>TargetWrapperServlet</servlet-name>
 <url-pattern>/wrapper</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>ResponseWrapperFilter</servlet-name>
 <servlet-class>jobs.ResponseWrapperFilter</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>ResponseWrapperFilter</servlet-name>
 <url-pattern>/responsefilter</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>TargetResponseWrapperServlet</servlet-name>
 <servlet-class>jobs.TargetResponseWrapperServlet</servlet-class>
</servlet>

<servlet-mapping>
```

```
<servlet-name>TargetResponseWrapperServlet</servlet-name>
<url-pattern>/responsewrapper</url-pattern>
</servlet-mapping>

<filter>
 <filter-name>firstfilter</filter-name>
 <filter-class>jobs.FirstFilter</filter-class>
</filter>

<filter-mapping>
 <filter-name>firstfilter</filter-name>
 <url-pattern>/tsfilter</url-pattern>
</filter-mapping>

<filter>
 <filter-name>dispatcherfilter</filter-name>
 <filter-class>jobs.DispatcherFilter</filter-class>
</filter>

<filter-mapping>
 <filter-name>dispatcherfilter</filter-name>
 <servlet-name>FirstDispatcherServlet</servlet-name>
 <dispatcher>INCLUDE</dispatcher>
 <dispatcher>REQUEST</dispatcher>
 <dispatcher>FORWARD</dispatcher>
 <dispatcher>ERROR</dispatcher>
</filter-mapping>

<filter>
 <filter-name>BadWordWrapperFilter</filter-name>
 <filter-class>jobs.BadWordWrapperFilter</filter-class>
</filter>

<filter-mapping>
 <filter-name>BadWordWrapperFilter</filter-name>
 <url-pattern>/wrapper</url-pattern>
</filter-mapping>

<filter>
 <filter-name>ResponseWrapperFilter</filter-name>
 <filter-class>jobs.ResponseWrapperFilter</filter-class>
</filter>
```

```

<filter-mapping>
 <filter-name>ResponseWrapperFilter</filter-name>
 <url-pattern>/responsewrapper</url-pattern>
</filter-mapping>

<error-page>
 <exception-type>java.lang.ArithmaticException</exception-type>
 <location>/dispatcher1</location>
</error-page>

</web-app>

```

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

# **JAVA MEANS DURGASOFT**

**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
 SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
 +91 9246212143  
 +91 8096969696

**[www.durgasoftonlinetraining.com](http://www.durgasoftonlinetraining.com)**



**Online Training  
 Pre Recorded Video  
 Classes Training  
 Corporate Training**

**Ph: +91-8885252627, 7207212427  
 +91-7207212428**

**USA Ph : 4433326786**

**E-mail : [durgasoftonlinetraining@gmail.com](mailto:durgasoftonlinetraining@gmail.com)**

**LEARN FROM EXPERTS ...**

# **COMPLETE JAVA**

CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

# **COMPLETE .NET**

C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

# **TESTING TOOLS**

MANUAL + SELENIUM

# **ORACLE D2K**

# **MSBI SHARE POINT**

# **HADOOP ANDROID**

# **C, C++, DS, UNIX**

# **CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED

**DURGA**  
Software Solutions®

# 202, 2nd Floor, HUDA Maitrivanam,  
Ameerpet, Hyd. Ph: 040-64512786,

**9246212143, 8096969696**

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA**

**With**

# **SCWCD / OCWCD**

**Servlets Material**

## **4. Session Management**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

# **DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

## Session Management

### Agenda:

#### 1) Session API

- Creation of Session object
- Invalidating a Session
  - invalidate()
  - By Session timeout mechanism
- Important Methods of HttpSession
- Attribute Management in Session Scope
- Exchanging Session Id between Client and Server

#### 2) Cookies

- Important methods of Cookie class
- Persistent (Vs) Non-Persistent Cookies
- Advantages of Cookies
- Limitations of Cookie
- Differences between Session-API and Cookies

#### 3) URL Rewriting

- HttpServletResponse two methods to append Session id to the url
- Advantage of URL Re-writing
- Limitations

#### 4) Hidden Variables

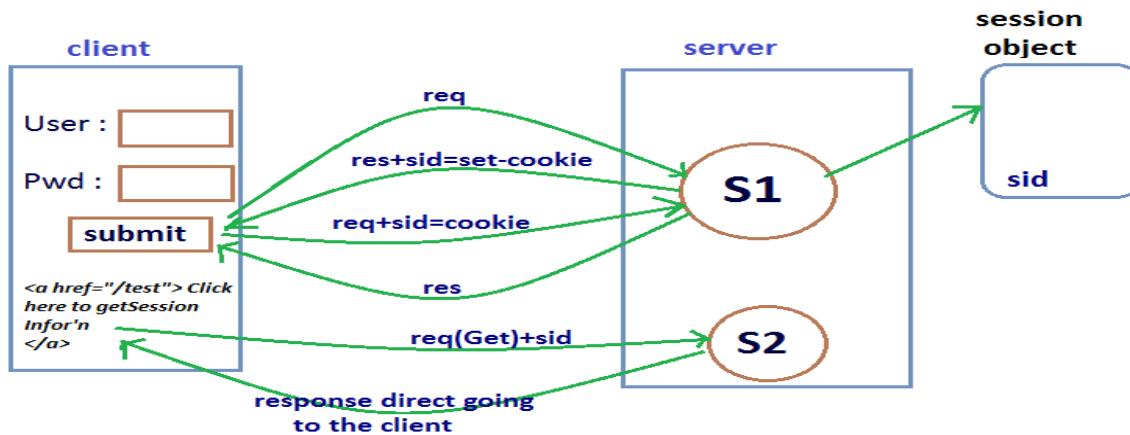
#### 5) Listeners

- Request Listeners
  - ServletRequestListener
  - ServletRequestAttributeListener
- Context Listeners
  - ServletContextListener
  - ServletContextAttributeListener
- Session Listeners
  - HttpSessionListener
  - HttpSessionAttributeListener
  - HttpSessionBindingListener
  - HttpSessionActivationListener

**Objective:**

1. For the given scenario describe the Session API.
2. Explain the process of creating a Session object.
3. What are various different mechanisms to invalidate a session
4. The basic limitation of Http is , Its a stateless protocol i.e., it is unable to remember client state across multiple request.
5. Every request to the server is considered as a new request. Hence some mechanism is required to remember client information across multiple requests. This mechanism is nothing but Session management /Session Tracking Mechanism.
6. The following are various Session management mechanisms.
  1. Session-API
  2. Cookies
  3. URL Rewriting
  4. Hidden Variables

It is not official Session management from Sun. It is just a programmers trick to remember client information across multiple requests.

**1.Session-API :**

1. When ever client sends 1<sup>st</sup> request to the server, If the server wants to remember client information for the future purpose then it will create a Session object stores the required Session information in the form of Session scoped attributes.
2. Server sends the corresponding session-id as the part of 1<sup>st</sup> response.
3. Client saves that Session-id and send back to the Server with every consecutive request.
4. By accessing Session-id and corresponding Session object , Server can able to remember client information accross multiple requests. This mechanism is nothing but Session management by using Session-API.

5. In this mechanism entire Session information will be stored at Server side , and only Session-id will be maintained by Client.

Ex:Bank Locker

#### Creation of Session object :

HttpServletRequest interface defines the following methods for the creation of Session object.

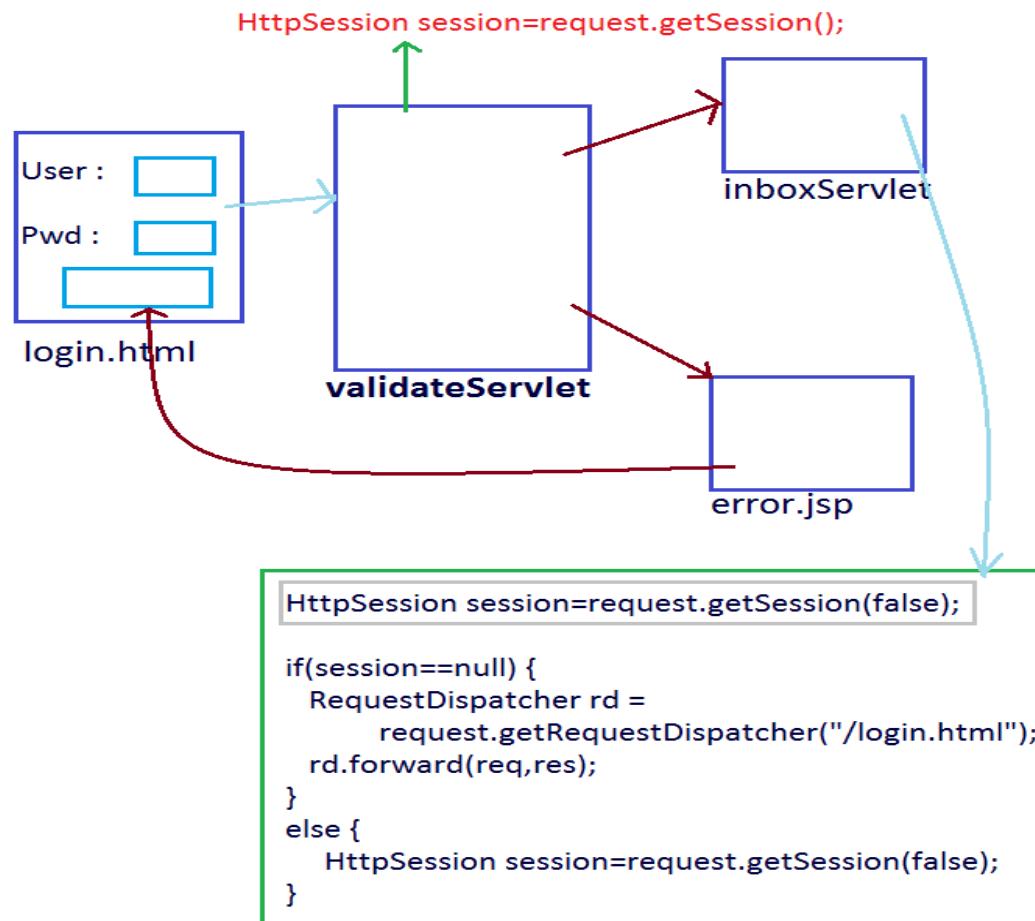
##### public HttpSession getSession()

- HttpSession session=request.getSession();
- First this method will check whether the request is already associated with any Session or not. If the request associated with any session then existing Session object will be returned.
- If the request is not associated with any Session , then only a new Session object will be created.

##### public HttpSession getSession(boolean b)

- if the argument is true , then this method simply acts as getSession()
- If the argument is false , then this method first checks whether the request is already associated with any Session or not, If it is already associated with a Session , then existing Session object will be returned.
- If the request is not associated with a Session then this method simply returns null , without creating new object.
- Note: There is always guarantee that first method returns a Session object. It may be newly created or already existing one.
- There is no guarantee that getSession(false) method will return Session object.





- After providing credentials in the login page the request will be forwarded to the ValidateServlet.
- Within the ValidateServlet, we will check whether the credentials are valid or not. If the credentials are valid, a new Session object will be created and forward the request to inbox.jsp, Hence within the Validate Servlet. We have to use `request.getSession()` method because we can create a new Session object, if it is not already there.
- To access inbox.jsp, compulsory request should be associated with Session.
- If the Session is not already there inbox.jsp is not responsible to create new Session object and just simply forwards the request to login page.
- Hence to meet this requirement inside inbox.jsp we have to use `request.getSession(false);`

#### Which of the two statements are equal ?

- `HttpSession session = request.getSession();`
- `HttpSession session=context.getSession(true);`

- HttpSession session = request.getSession(false);
- HttpSession session = request.getSession(true);

**Answer : 1 & 4**

### Invalidating a Session :

We can invalidate a Session by using the following 2 ways

1. By invalidate method
2. By timeout mechanism

#### 1.invalidate() :

HttpSession interface contains invalidate() to invalidate a Session explicitly.

```
public void invalidate()
```

When ever we click logout button , internally this method will be executed.

**Ex: session.invalidate()**

### LogoutServlet.java

```
public class LogoutServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 HttpSession session=request.getSession(false);
 if(session != null)
 session.invalidate();
 else
 out.println("No session is invalidate()");
 }
}
```

### web.xml

```
<servlet>
 <servlet-name>LogoutServlet</servlet-name>
 <servlet-class>session.LogoutServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>LogoutServlet</servlet-name>
 <url-pattern>/logout</url-pattern>
</servlet-mapping>
```

## 2.By Session timeout mechanism :

1. If we are not performing any operation for a predefined amount of time on the session object, then session will be expired automatically
2. This pre-defined amount of time is called "Session time out".
3. We can configure Session time out at Server level or a particular web-application level or for a particular Session object level.

- ***Automatic Support from the WebServer :***

Most of the Web-Servers provide default support for Session time out mostly it is 30 minutes. We can customize this value based on our requirement. This Session timeout is application for all sessions , Which are created in that Server irrespective of application.

- ***Configuring Session timeout at application level :***

1. We can configure Session timeout for entire web-application in web.xml as follows

```
<web-app>
 <session-config>
 <session-timeout> 10 </session-timeout>
 </session-config>
</web-app>
```

2. The session-config is direct child tag of web-app .Hence we can take anywhere with in web-app
3. The unit to the session timeout is in minutes.
4. Zero (or) negative value indicates that session never expires.(until we click the logout button)
5. This session timeout value is applicable for all sessions , which are created as part of web application.

### Configuring Session timeout for a particular Session object :

6. We can set the Session timeout by using setMaxInactiveInterval() method for a particular Session object.

```
public void setMaxInactiveInterval(int seconds)
```

7. **Ex: session.setMaxInactiveInterval(120);**

8. The argument is in seconds, -ve value indicates session never expires, Zero value indicates session expires immediately .
9. This session time out is applicable only for a particular session object on which this method has called.

### Comparision between two Session timeout mechanism :

Property	<session-timeout>	setMaxInactiveInterval()
Scope	It is applicable for all the Sessions which are created in that application.	It is applicable only for a particular Session object on which we called this method
Units	Minutes	Seconds
Zero value	Session never expires	Session expires immediately
- ve	Session never expires	Session never expires

### Important Methods of HttpSession :

#### 1) public boolean isNew()

We can use this method to check whether the Session object is newly created or not.

#### 2) public void invalidate()

To expires a session forcefully.

#### 3) public void setMaxInactiveInterval(int seconds)

To set session timeout for a particular Session object .

#### 4) public void getMaxInactiveInterval()

It returns the session timeout value in seconds.

#### 5) public String getId()

Returns the session id.

#### 6) public long getCreationTime()

- Returns the time when the Session was created in milliseconds , Since Jan 1st1970.
- If we are passing this long value to the Date constructor , we will get exact Date and time.  
Date d = new Date( l );

#### 7) public long getLastAccessedTime()

Returns the time when the client accessed recently the .....

### 8) public ServletContext getServletContext()

Returns the ServletContext object to which this Session belongs.

#### ***Methods in HttpSession to perform Attribute Management in Session Scope :***

1. public void setAttribute(String name, Object value)
2. public Object getAttribute(String name )
3. public void removeAttribute(String name)
4. public Enumeration getAttributeNames()

Note : Once session expires we can't call any of the above methods , validation leads to RuntimeException saying "IllegalStateException".But this rule is not applicable for getServletContext() method .

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

session.isNew(); //java.lang.IllegalStateException

out.println(session.getServletContext()); //valid
```

***Ex: Demo program for Session Management by session API***

login.html

```
<form action="../sessionone">
<table>
 <tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
 <tr><td>Value : </td><td> <input type="text" name="value"></td></tr>
 <tr><td><input type="submit" value="submit"></td></tr>
</table>
</form>
Session Information
```

SessionServletOne.java

```
public class SessionServletOne extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 String name=request.getParameter("uname");
 String value=request.getParameter("value");
 HttpSession session=request.getSession();
 if(session.isNew()) {
 out.println("New Session got created "+session.getId());
```

```

 }
 else {
 out.println("With existing Session id : "+session.getId());
 }
 session.setAttribute(name, value);
 session.setMaxInactiveInterval(120);
 RequestDispatcher rd=request.getRequestDispatcher("login.html");
 rd.include(request, response);
 }
}

```

**SessionServletTwo.java**

```

public class SessionServletTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 HttpSession session=request.getSession(false);
 if(session == null) {
 out.println("No Session id associated with request ");
 }
 else {
 Enumeration e=session.getAttributeNames();
 out.println("<table border=3><tr><th>Session Attribute Name</th>
 <th>Session Attribute Value</th></tr>");
 while(e.hasMoreElements()) {
 String name=(String)e.nextElement();
 String value=(String)session.getAttribute(name);
 out.println("<tr><td>" +name+ "</td> <td>" +value+ "</td></tr>");
 }
 out.println("</table>");
 out.println("
The Session creation Time is :" + new Date(session.getCreationTime()));
 out.println("
The session last accessed time is :" + new Date(session.getLastAccessedTime()));
 out.println("
The Session max inactive interval is :" + session.getMaxInactiveInterval());
 }
 out.println("
 login page ");
 }
}

```

**web.xml**

```

<web-app>
 <servlet>
 <servlet-name>SessionServletOne</servlet-name>
 <servlet-class>session.SessionServletOne</servlet-class>
 </servlet>

 <servlet>

```

```
<servlet-name>SessionServletTwo</servlet-name>
<servlet-class>session.SessionServletTwo</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>SessionServletOne</servlet-name>
 <url-pattern>/sessionone</url-pattern>
</servlet-mapping>

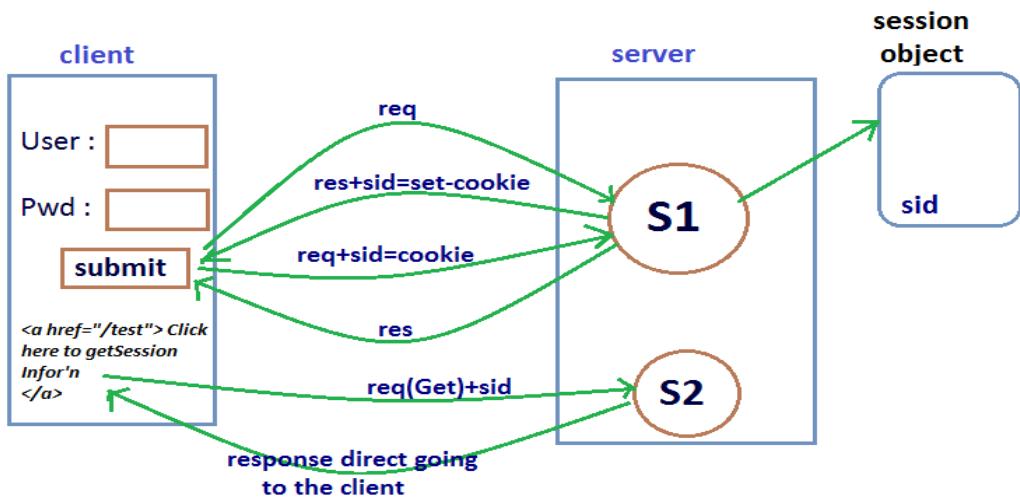
<servlet-mapping>
 <servlet-name>SessionServletTwo</servlet-name>
 <url-pattern>/sessiontwo</url-pattern>
</servlet-mapping>

<session-config>
 <session-timeout>1</session-timeout>
</session-config>

</web-app>
```

### Exchanging Session Id between Client and Server :

1. Whenever Client sends 1<sup>st</sup> request to the Sever , If any request information is required to remember for the future purpose then server creates a Session object and score the required information in the form of Session scoped attributes.
2. Server sends corresponding session id to the client as the part of 1<sup>st</sup> response. For this server uses "set-cookie" response header.
3. Whenever client got the response , it retrieves the session id and stored in the local file System.
4. With every consecutive request client send back the corresponding Session id to the server, for this client uses "cookie" request header.
5. By accessing Session id and corresponding Session object Server can able to remember client information across multiple requests.
6. By using set-cookie response header and cookie request header session id will be exchanged between client and server.



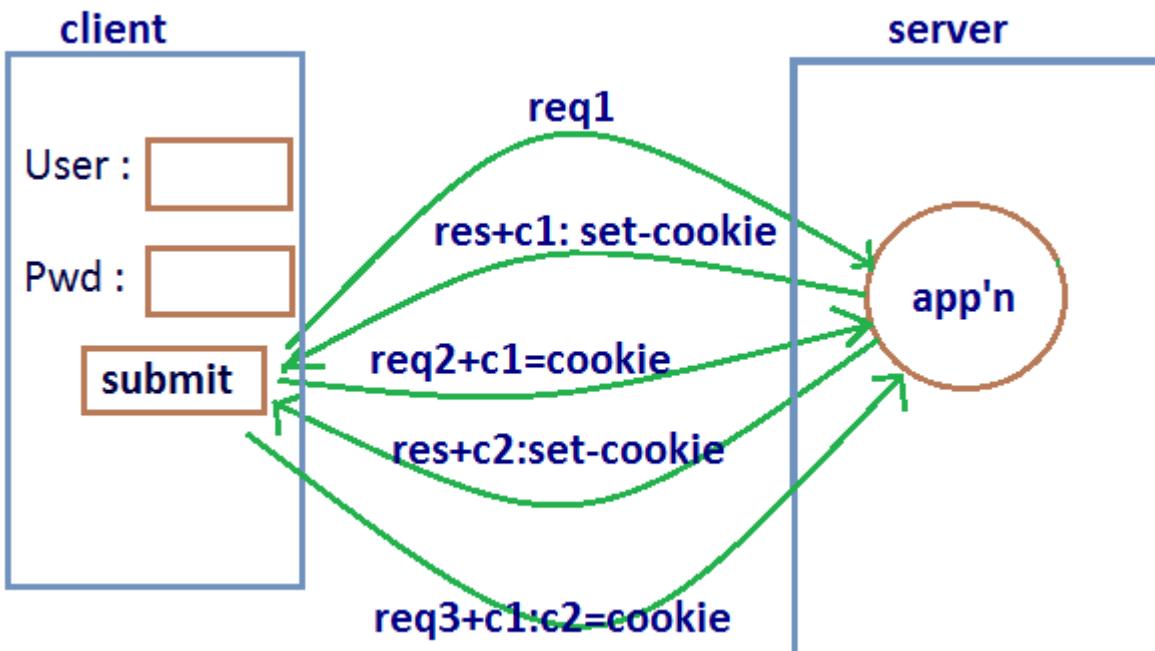
Session id will be stored as per application in the client side.

header	value
cookie	sid

Ex: Note:(Bank locker):

1. If the required Session information is very less than creating a separate Session object and maintaining that object at Server side is not recommended because it impacts performance of the System.
2. To resolve this, we should go for cookie Session management mechanism , where required Session information is maintained at client side and Server is not responsible to maintain any Session information.



**Cookies :**

1. Cookie is a small amount of information (key , value) pair.
2. Whenever client sends a request to the Server , if any information required to remember for future purpose then server creates a Cookie object with that information and send that Cookie back to the client as the part of response . For, this , server uses set-cookie response header.
3. Whenever client got the response it retrieves the cookies send by the Server and stores those cookies in the local file system.
4. Client sendback all the Cookies send by the server with every consecutive request for this client use cookie request header.
5. By accessing those cookies server can able to remember client information across multiple requests.
6. We can create Cookie object by using Cookie class constructor.

```
Cookie cookieobject = new Cookie(String key , String value);
```

7. After creating the Cookie object we have to add that Cookie to the response by using addCookie().

```
response.addCookie(cookieobject);
```

8. At server side , we can retrieve all cookies send by the client by using getCookies( ).

```
Cookies[] cookieobject = request.getCookies();
```

9. If the request doesn't associated with any Cookies then this method returns null.

### Important methods of Cookie class :

#### public String getName()

This method returns the name of the Cookie.

#### public String getValue()

Returns value of the Cookie.

#### public int getMaxAge()

Returns the maximum age of Cookie in seconds.

#### public void setMaxAge(int seconds )

To set max age of cookie.

- "+ve" value indicates that cookie will be expires after that how many seconds have passed.
- Note that the value is max age of the cookie, when the cookie will expires , not the cookie current age.
- "-ve" value indicates the cookie is not stored permanently and it will be deleted when browser exists/close.
- "0" value indicates cookies to be deleted.

### Demo Program for session management by cookies

#### login.html

```
<form action=".//cookieone">
 <table>
 <tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
 <tr><td>Value : </td><td> <input type="text" name="value"></td></tr>
 <tr><td><input type="submit" value="submit"></td></tr>
 </table>
</form>
Cookie Information
```

#### CookieServletOne.java

```
public class CookieServletOne extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 String name=request.getParameter("uname");
```

```

String value=request.getParameter("value");
Cookie c=new Cookie(name, value);
c.setMaxAge(120);
response.addCookie(c);
out.println("Cookie added Successfully");
out.println("domain"+c.getDomain());
out.println("comment"+c.getComment());
out.println("maxage"+c.getMaxAge());
out.println("version"+c.getVersion());
out.println("hashcode"+c.hashCode());
RequestDispatcher rd=request.getRequestDispatcher("login.html");
rd.include(request, response);
}
}

```

**CookieServletTwo.java**

```

public class CookieServletTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 Cookie[] c = request.getCookies();
 if(c == null) {
 out.println("No cookies associated with this request");
 }
 else {
 out.println("<table border=3><tr><th>Cookie Name</th><th>Cookie Value</th></tr>");
 for(Cookie c1:c) {
 String name = c1.getName();
 String value = c1.getValue();
 out.println("<tr><td>" + name + "</td><td>" + value + "</td></tr>");
 }
 out.println("</table>");
 }
 }
}

```

**web.xml**

```

<web-app>

 <servlet>
 <servlet-name>CookieServletOne</servlet-name>
 <servlet-class>session.CookieServletOne</servlet-class>
 </servlet>

 <servlet>
 <servlet-name>CookieServletTwo</servlet-name>

```

```

<servlet-class>session.CookieServletTwo</servlet-class>
</servlet>

<servlet-mapping>
 <servlet-name>CookieServletOne</servlet-name>
 <url-pattern>/cookieone</url-pattern>
</servlet-mapping>

<servlet-mapping>
 <servlet-name>CookieServletTwo</servlet-name>
 <url-pattern>/cookietwo</url-pattern>
</servlet-mapping>

</web-app>

```

### Persistent (Vs) Non-Persistent Cookies :

Persistent Cookies	Non-Persistent Cookies
<ol style="list-style-type: none"> <li>If we set max age for the Cookie , such type of Cookies are called "Persistent Cookies".</li> <li>These cookies will be stored permanently in the local file system.</li> <li>Once the time expires , these Cookies will be disabled automatically.</li> </ol>	<ol style="list-style-type: none"> <li>If we are not setting max age for the Cookie , such type of Cookies are called Non-Persistent/temporary Cookies.</li> <li>These will be stored in the browsers cache and disabled automatically Once browser will be closed.</li> </ol>

### Advantages of Cookies :

- It is very easy to implement.
- Persist across browser shutdowns, server shutdowns and application redeployments.
- If very less Session information is available or if huge no. of end-users are available then the best suitable mechanism is Cookies.

### Limitations of Cookie :

- To meet security constraints there may be a chance of disabling cookies at client side. In this case Session management by Cookies won't work .
- The maximum no. of Cookies supported by the browser is fixed.(maximum of 30, based on browser)
- The size of Cookie is also fixed. Hence we can't store huge amount of information by using Cookies.
- The Cookies should be travelled every time across the network. Hence there may be a chance of network overheads. i.e., it impacts performance

### Differences between Session-API and Cookies :

Session-API	Cookies
If huge amount of Session information is available , then we should go for Session-API .	If very less amount of Session information is available , then we should go for Cookie.
Session information will be maintained at Server side.	Session information will be maintained at client side.
The Session information can be any type and need not be String type.	Session information should be String type
Session information won't persist across server shutdown & application re-deploys.	Session information will be persist across Server shutdowns & application deployments.(The cookies should be persist)

**Note:** To meet Security constraints there may be a chance of disabled cookies at client side . In this case Session management by using Session-API and Cookies won't work. To handle this requirement we should go for url-rewriting .

### URL - Rewriting :

1. Whenever Cookies are disabled at client side , browser unable to see "set-cookie" response header and hence browser unable to get session id and cookies send by the Server .
2. Due to this browser can't send session id & Cookies to the server and hence server is unable to remember client information across multiple requests . So that Session management fails.
3. To resolve this we should go for url rewriting technique.
4. The central idea in this technique is append required Session information to the url , instead of appending to "set-cookie" response header.
5. Whenever client clicks url for further communication server can get required Session information with the url . So that Server can able to remember client information across multiple requests.

URL Re-writing = URL + session information

OR

URL Re-writing = URL ; jsessionid=1234

**HttpServletResponse defines the following two methods to append Session id to the url**

- 1) **public String encodeURL(String url)**

Returns url by appending jsession id

## 2) **public String encodeRedirectURL(String url)**

Returns the url by appending session id . This URL can be used as an argument to send `sendRedirect()` method

- The above 2 methods will append jsessionid to the url iff cookies are disabled at client side.
- If cookies are enabled then these methods returns the same url without appending jsessionid .
- `HttpServletRequest` defines the following methods to identify whether the sessionid is coming as the part of url or with cookie request header.
  - 1) `public boolean isRequestedSessionIdFromURL()`
  - 2) `public boolean isRequestedSessionIdFromCookie()`
- By using these methods we can identify underlying Session management technique.

### Advantage of URL Re-writing :

There is no chance of disabling url re-writing technic hence this technique will work always . It is universally supported.

### Limitations :

1. It is very difficult to rewrite every url to append Session information .
2. Url rewriting works only for dynamic document i.e., response should contain at least one url .
3. To keep our web-application as robust we can use both cookies and url rewriting together.
4. If cookies are enabled then cookies will work otherwise url rewriting will work.

### Demo program on URL rewriting

#### **login.html**

```
<form action=".//redirectone">
<table>
 <tr><td>Name :</td><td> <input type="text" name="uname"></td></tr>
 <tr><td><input type="submit" value="submit"></td></tr>
</table>
</form>
```

#### **UrlRedirectServletOne.java**

```
public class UrlRedirectServletOne extends HttpServlet {
 @SuppressWarnings("deprecation")
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 String name = request.getParameter("uname");
 HttpSession session = request.getSession();
 session.setAttribute("uname", name);
 out.println("Welcome to Aksahay");
 // out.println("
 click here to get User ");
 }
}
```

```
out.println("
 click here to get User ");
 }
}
```

**UrlRedirectServletTwo.java**

```
public class UrlRedirectServletTwo extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 HttpSession session = request.getSession(false);
 String name = (String)session.getAttribute("uname");
 //String name = request.getParameter("name");
 out.println("Good Morning :" + name);
 }
}
```

**web.xml**

```
<web-app>

 <servlet>
 <servlet-name>UrlRedirectServletOne</servlet-name>
 <servlet-class>session.UrlRedirectServletOne</servlet-class>
 </servlet>

 <servlet>
 <servlet-name>UrlRedirectServletTwo</servlet-name>
 <servlet-class>session.UrlRedirectServletTwo</servlet-class>
 </servlet>

 <servlet-mapping>
 <servlet-name>UrlRedirectServletOne</servlet-name>
 <url-pattern>/redirectone</url-pattern>
 </servlet-mapping>

 <servlet-mapping>
 <servlet-name>UrlRedirectServletTwo</servlet-name>
 <url-pattern>/redirecttwo</url-pattern>
 </servlet-mapping>

</web-app>
```

## Listeners

### Objective :

- Describe the web-container event life cycle model for the request, Session and webapplication(context).
- Create and configure Listener class for each Scope.
- Create and configure attribute Listeners for each scope.
- For the given scenario identify proper attribute Listener.
- In the webapplication there may be a chance of occurring several events like ...
  1. Request object creation/destruction
  2. Session object creation/destruction
  3. Context object creation/destruction
  4. Attribute addition in request/session Scope
  5. Removal of attribute in request/session Scope
- We can configure Listeners classes to listen these events and they can do appropriate things whenever that event occurs.

When ever a particular event occurs if we want to perform certain operation then we should go for Listeners.

All the Listeners are divided into 3 groups.

#### 1) RequestListeners :

- These Listen the events related to request.
- There are 2 types of RequestListeners.
  - ServletRequestListener
  - ServletRequestAttributeListener

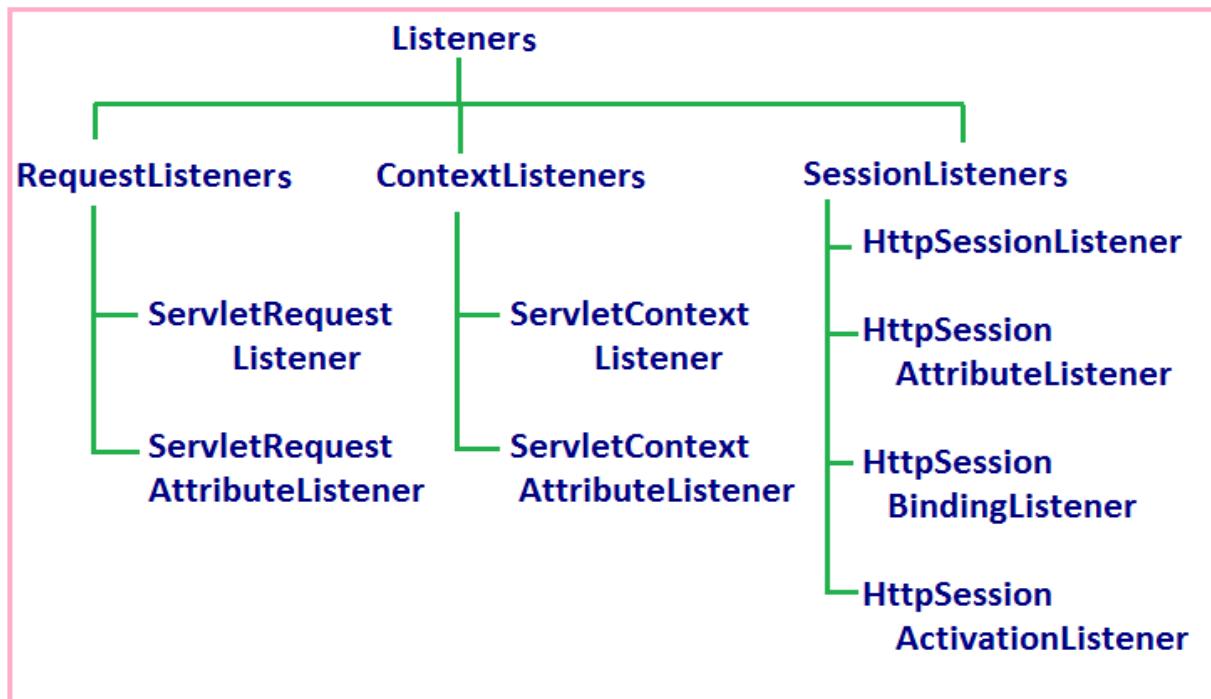
#### 2) Session Listeners :

- Listens the events related to Sessions.
- There are 4 types of Session Listeners.
  - HttpSessionListener
  - HttpSessionAttributeListener
  - HttpSessionBindingListener
  - HttpSessionActivationListener

#### 3) ContextListener :

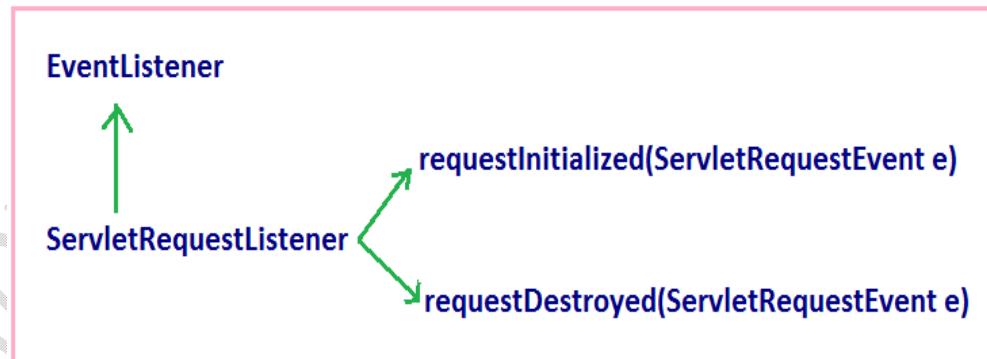
- Listens the events related to context.
- There are 2 types of context Listeners.
  - ServletContextListener
  - ServletContextAttributeListener

**Note:** `java.util.EventListener` is the super interface for all the Listeners in our servlets. This Listener doesn't contain any method and acts as marker interface.



### RequestListeners :

`ServletRequestListener`



This Listener life cycle events of request object like request object creation and destruction.

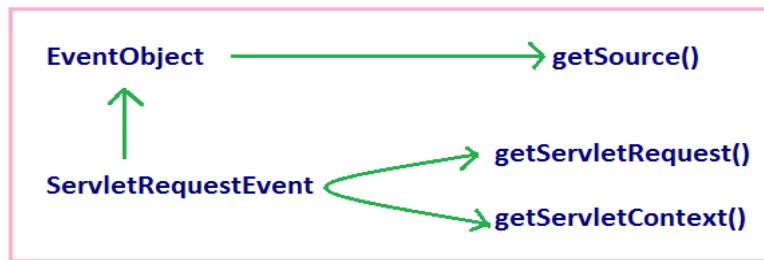
This interface defines 2 methods

**public void requestInitialized(ServletRequestEvent e)**

This method will be executed automatically by the WebContainer, at the time of request object creation, i.e., just before executing service()

**public void requestDestroyed(ServletRequestEvent e)**

This method will be executed automatically by the WebContainer at the time of request object destruction, i.e., just after completing service().

**ServletRequestEvent(C) :**

It defines the following 2 methods

1. `public ServletRequest getServletRequest()`
2. `public ServletContext getServletContext()`

**java.util.EventObject**

ServletRequestEvent is the child class of EventObject, it contains only one method getSource()

`public Object getSource()`

It returns source of the event, in this case the source of the event is web application which is represented by ServletContext object hence getSource() method returns ServletContext object.

**ServletRequestListenerDemo.java**

```

public class ServletRequestListenerDemo implements ServletRequestListener {
 static int count;
 static {
 System.out.println("ServletRequestListener class is loading ");
 }
 public ServletRequestListenerDemo() {
 System.out.println("ServletRequestListener Object is created ");
 }
 public void requestDestroyed(ServletRequestEvent event) {
 System.out.println("The request object destroyed at :" + new java.util.Date());
 System.out.println("The source of destroyed request is :" + event.getSource());
 }
 public void requestInitialized(ServletRequestEvent event) {
 count++;
 System.out.println("new request object is created at :" + new java.util.Date());
 System.out.println("The source of creation request is :" + event.getSource());
 }
}

```

```

 System.out.println("The context is :" + event.getServletContext());
 System.out.println("The request is :" + event.getServletRequest());
 }
}

```

**RequestServlet.java**

```

public class RequestServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("The hit count of the web application is :" + ServletRequestListenerDemo.count);
 }
}

```

**web.xml**

```

<web-app>
 <listener>
 <listener-class>listener.ServletRequestListenerDemo</listener-class>
 // fully qualified name
 </listener>
 <servlet>
 <servlet-name>RequestServlet</servlet-name>
 <servlet-class>listener.RequestServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>RequestServlet</servlet-name>
 <url-pattern>/request</url-pattern>
 </servlet-mapping>
</web-app>

```

- By using listener tag we can configure listener class in web.xml
- listener tag is the direct child tag of web-app, hence we can take anywhere with in the web-app.
- web container is responsible for the creation of listener object for this WC always calls public no-arg constructor, hence every listener class should compulsory contains public no-arg constructor otherwise Instantiation Exception.
- Instantiation of the listener will be happen at the time of web-application deployment or server start-up.

**Note:** In the above program count value won't persist across server shutdown or application undeployment but we can persist count value by using ServletContextListener.

**ServletRequestAttributeListener**

This Listener listens to events related to request scoped attributes like attribute addition, attribute removal, attribute replacement.

This interface defines following methods

1. **public void attributeAdded(ServletRequestAttributeEvent e)**  
This method will be executed automatically by the by the WC, when ever we are adding an attribute an request scope.
2. **public void attributeRemoved(ServletRequestAttributeEvent e)**  
This method will be executed automatically by the by the WC, when ever we are removing an attribute from request scope.
3. **public void attributeReplaced(ServletRequestAttributeEvent e)**  
This method will be executed automatically by the by the WC, when ever we are replacing an existing attribute named with new value. This method returns old value, but not new value.

### ServletRequestAttributeEvent

It is the child class of ServletRequestEvent, this class defines the following 2 methods

1. **public String getName()**  
It returns name of the attribute which is adding request scope.
2. **public Object getValue()**  
It returns the value of attribute which is added or removed , In this case of replacement this method returns old value.

#### ServletRequestAttributeListenerDemo.java

```
public class ServletRequestAttributeListenerDemo implements ServletRequestAttributeListener {
 public ServletRequestAttributeListenerDemo() {
 System.out.println("ServletRequestAttributeListenerDemo obj created ");
 }
 public void attributeAdded(ServletRequestAttributeEvent event) {
 System.out.println("Attribute added name :" + event.getName());
 System.out.println("Attribute added value :" + event.getValue());
 }
 public void attributeRemoved(ServletRequestAttributeEvent event) {
 System.out.println("Attribute removed name :" + event.getName());
 System.out.println("Attribute removed value :" + event.getValue());
 }
 public void attributeReplaced(ServletRequestAttributeEvent event) {
 System.out.println("Attribute replaced name :" + event.getName());
 System.out.println("Attribute replaced value :" + event.getValue());
 }
}
```

#### ServletRequestAttribute.java

```
public class ServletRequestAttribute extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```

 throws ServletException, IOException {
 response.setContentType("text/html");
 //PrintWriter out = response.getWriter();
 request.setAttribute("Ashok", "SCJP");
 request.setAttribute("Ashok", "SCWCD");
 request.removeAttribute("Ashok");
}
}

web.xml
<listener>
 <listener-class>listener.ServletRequestAttributeListenerDemo</listener-class>
</listener>
<servlet>
 <servlet-name>ServletRequestAttribute</servlet-name>
 <servlet-class>listener.ServletRequestAttribute</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>ServletRequestAttribute</servlet-name>
 <url-pattern>/requestattribute</url-pattern>
</servlet-mapping>

```

### ContextListener:

#### ServletContextListener

This Listener listens life cycle events of context object like creation & destruction. This Listener defines the following 2 methods

1. public void contextInitialized(ServletContextEvent e)  
This method will be executed at the time of context object created or application deployment.
2. public void contextDestroyed(ServletContextEvent e)  
This method will be executed at the time of context object destruction. i.e., at the time of application un-deployment.

#### ServletContextEvent:

It is the child class `java.util.EventObject` and it contains only one method `public ServletContext getServletContext()`.

Demo program to ContextListener to print hitcount of the web-application which should persist across server shutdown.

#### ServletRequestListenerDemo.java

#### ServletContextListenerDemo.java

```

public class ServletContextListenerDemo implements ServletContextListener {
 public void contextDestroyed(ServletContextEvent event) {
 System.out.println("context destroyed ");
 String path = event.getServletContext().getRealPath("abc.txt");
 }
}

```

```

try {
 PrintWriter out=new PrintWriter(path);
 out.print(ServletRequestListenerDemo.count);
 out.flush();
} catch (Exception e) {}

public void contextInitialized(ServletContextEvent event) {
 System.out.println("context initialized ");
 String path = event.getServletContext().getRealPath("abc.txt");
 try {
 BufferedReader br = new BufferedReader(new FileReader(path));
 String s = br.readLine();
 if(s != null) {
 int c = Integer.parseInt(s);
 ServletRequestListenerDemo.count = c;
 }
 } catch (Exception e) {}
}
}

```

**ServletContext.java**

```

public class ServletContext extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Hitcount persist across server shutdown
in Servlet context listener :" + ServletRequestListenerDemo.count);
 }
}

```

**web.xml**

```

<listener>
 <listener-class>listener.ServletContextListenerDemo</listener-class>
</listener>
<servlet>
 <servlet-name>ServletContext</servlet-name>
 <servlet-class>listener.ServletContext</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>ServletContext</servlet-name>
 <url-pattern>/context</url-pattern>
</servlet-mapping>
<listener>
 <listener-class>listener.ServletRequestListenerDemo</listener-class>
</listener>

```

- We can configure more than one listener of the same type but the order of the execution is based on the order of Listener tag in the web.xml
- If both listeners are 2 different types the order is not important.

### Demo Program on ServletContextListener

It is not a Dog object in realtime it is DataSource object.

#### Dog.java

```
public class Dog {
 public String breed;
 public Dog(String breed) {
 this.breed = breed;
 }
 public String getBreed() {
 return breed;
 }
}
```

#### ServletContextListenerDog.java

```
public class ServletContextListenerDog implements ServletContextListener {
 public void contextInitialized(ServletContextEvent event) {
 System.out.println("context listener dog initialized");
 ServletContext context=event.getServletContext();
 String dogBreed = context.getInitParameter("breed");
 Dog d = new Dog(dogBreed);
 context.setAttribute("dog",d);
 }
 public void contextDestroyed(ServletContextEvent event) {
 System.out.println("context listener dog destroyed");
 }
}
```

#### ServletContextDog.java

```
public class ServletContextDog extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.println("Servlet Context Listener Dog ");
 Dog d = (Dog) getServletContext().getAttribute("dog");
```

```

 out.println("The dog breed is : "+d.getBreed());
 }
}

```

**web.xml**

```

<listener>
 <listener-class>listener.ServletContextListenerDog</listener-class>
 //The whole point is to be initialized the app'n before any servlet is initialized
</listener>
<context-param>
 <param-name>breed</param-name>
 <param-value>Great Puppy</param-value>
</context-param>
<servlet>
 <servlet-name>ServletContextDog</servlet-name>
 <servlet-class>listener.ServletContextDog</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>ServletContextDog</servlet-name>
 <url-pattern>/contextdog</url-pattern>
</servlet-mapping>

```

1. Container read the deployment descriptor for the application including `<listener>` and `<context-param>` elements.
2. Container creates a new `ServletContext` object for this application that all parts of application will share.
3. Container creates name/value pairs of strings for each context init-parameters.
4. Container gives the `ServletContext` reference to the name value parameters.
5. Container creates a new instance of the `ServletContextListenerDemo` class.
6. Container calls the listener context initialized method by passing `ServletContextEvent`, the `EventObject` has a reference to the `ServletContext` so the Eventhandling code can get the context form the event & get the `ContextInitParameters` from the Context.
7. Listener asks `ServletContextEvent` for a reference to `ServletContext`.
8. Listener asks `ServletContext` for the `ContextInitParameter(breed)`.
9. Listener uses this `initParameter` to construct a new `Dog` object.
10. Listener sets the `Dog` object as attributes in the servlet context scope.
11. Container makes a new servlet(i.e., makes a new `ServletConfig`) with `initParameter` gives a `ServletConfig` reference to the `ServletContext`, when it calls servlet init method.
12. Servlet gets a request and ask the `ServletContext` for the attribute "dog".
13. Servlets calls `getBreed()` on the `Dog` object.

**ServletContextAttributeListener**

This listener listens events related to context scoped attributes , i.e., attribute added in the context scope or replacement or removal. This interface defines the following methods

1. `public void attributeAdded(ServletContextAttributeEvent e)`

2. public void attributeRemoved(ServletContextAttributeEvent e)
3. public void attributeReplaced(ServletContextAttributeEvent e)

### ServletContextAttributeEvent

It is child class of ServletContextEvent, it contains the following 2 methods.

1. public String getName()
2. public Object getValue()

### Session Listeners:

#### 1) HttpSessionListener

HttpSessionListener listens lifecycle events of session object like session object creation and destruction, this interface defines the following 2 methods.

- public void sessionCreated(HttpSessionEvent e)
- public void sessionDestroyed(HttpSessionEvent e)

#### HttpSessionEvent

It is the child class of java.util.EventObject, it contains only one method.

public HttpSession getSession()

#### HttpSessionListenerDemo.java

```
public class HttpSessionListenerDemo implements HttpSessionListener {
 static int count;
 public void sessionCreated(HttpSessionEvent event) {
 System.out.println("new session object created at :" + new java.util.Date());
 count++;
 }
 public void sessionDestroyed(HttpSessionEvent event) {
 System.out.println("session object is destroyed :" + new java.util.Date());
 count--;
 }
}
```

#### SessionServlet.java

```
public class SessionServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 HttpSession session = request.getSession();
 //session.setMaxInactiveInterval(120);
 out.println("The no. of users online is :" + HttpSessionListenerDemo.count);
 }
}
```

```
}
```

```
web.xml
<listener>
 <listener-class>listener.HttpSessionListenerDemo</listener-class>
</listener>
<session-config>
 <session-timeout>3</session-timeout>
</session-config>
<servlet>
 <servlet-name>SessionServlet</servlet-name>
 <servlet-class>listener.SessionServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>SessionServlet</servlet-name>
 <url-pattern>/session</url-pattern>
</servlet-mapping>
```

### HttpSessionAttributeListener

This Listener listens events related to session scoped attributes like attribute addition in the session scope, or removal or replacement.

This interface defines the following methods

1) public void attributeAdded(HttpSessionBindingEvent e)

This method will be executed automatically by the WC, when ever we are adding any type of objects in the session scope.

2) public void attributeReplaced(HttpSessionBindingEvent e)

This method will be executed automatically by the WC, when ever we are replacing an existing object with new object.

3) public void attributeRemoved(HttpSessionBindingEvent e)

This method will be executed automatically by the WC, when ever we are removed any type of attribute in the session scope.

**Note:** There is no class named with HttpSessionAttributeEvent for this requirement the request class is HttpSessionBindingEvent.

### HttpSessionBindingEvent

It is the child class of HttpSessionEvent , and it contains 2 methods

- public String getName()
- public Object getValue()

### HttpSessionBindingListener

When ever we are trying to add or remove or replacement a particular type of object in session scope, if we want to perform any operation then we should go for HttpSessionBindingListener.

This interface defines the following 2 methods

- 1) public void valueBound(HttpSessionBindingEvent e)

This method will be executed automatically by the WC, when ever we are trying to add a particular type of object in session scope.

- 2) public void valueUnbound(HttpSessionBindingEvent e)

This method will be executed automatically by the WC, when ever we are trying to remove a particular type of object in session scope.

- For replacement operations both methods will be executed but valueBound() first and followed by valueUnbound().
- It is not required to configuire HttpSessionBindingListener in web.xml
- When ever we are trying to add an attribute in session scope , WC will check whether the corresponding class implements HttpSessionBindingListener or not , if it is implements HttpSessionBindingListener then valueBound() will be executed automatically by the WC.
- WC follows the same approach for attributeRemoval and attributeReplacement also.
- If we configure both attribute & binding listeners then binding listener will executed first followed by attribute listener.

### Demo program

HttpSessionAttributeListenerDemo.java

```
public class HttpSessionAttributeListenerDemo implements HttpSessionAttributeListener {
 public void attributeAdded(HttpSessionBindingEvent event) {
 System.out.println("Attributes added");
 }
 public void attributeRemoved(HttpSessionBindingEvent event) {
 System.out.println("Attribute removed");
 }
 public void attributeReplaced(HttpSessionBindingEvent event) {
 System.out.println("Attribute replaced");
 }
}
```

HttpSessionBindingListenerDemo.java

```
public class HttpSessionBindingListenerDemo implements HttpSessionBindingListener {
 public void valueBound(HttpSessionBindingEvent event) {
 System.out.println("Session object has added to the Session scope : bound ");
 }
 public void valueUnbound(HttpSessionBindingEvent event) {
 System.out.println("Session object has removed to the Session scope : unbound ");
 }
}
```

ServletSessionAttributeBind.java

```
public class ServletSessionAttributeBind extends HttpServlet {
```

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 HttpSession session=request.getSession();
 session.setAttribute("a1", "SCJP");
 session.setAttribute("a1", "SCWCD");
 session.setAttribute("a2", new HttpSessionBindingListenerDemo());
 session.setAttribute("a3", new HttpSessionBindingListenerDemo());
 session.setAttribute("a2", new HttpSessionBindingListenerDemo());
 session.removeAttribute("a2");
 session.setAttribute("a1", new HttpSessionBindingListenerDemo());
 out.println("HttpSessionBindingListener not required to configure");
}
}

```

**web.xml**

```

<listener>
 <listener-class>listener.HttpSessionAttributeListenerDemo</listener-class>
</listener>
<servlet>
 <servlet-name>ServletSessionAttributeBind</servlet-name>
 <servlet-class>listener.ServletSessionAttributeBind</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>ServletSessionAttributeBind</servlet-name>
 <url-pattern>/sessionbind</url-pattern>
</servlet-mapping>

```

**HttpSessionActivationListener**

- If a web-application is distributed across several JVM's such type of web-application is called **distributed web applications**.
- The main advantage of distributed web-applications are --
  - By Load Balancing:** we can improve performance of the application.
  - By Handling:** fail over situations we can keep our application has robust.
- In distributed web-applications the session object is required to migrate from one JVM to another JVM.
- When ever a session object is migrating from one JVM to another JVM , the corresponding session scoped attributes also will migrate across the network , hence session scoped attributes should serializable.
- At the time of session object migration , if want to perform any operation then we should go for **HttpSessionActivationListener**.

This Listener defines the following 2 methods.

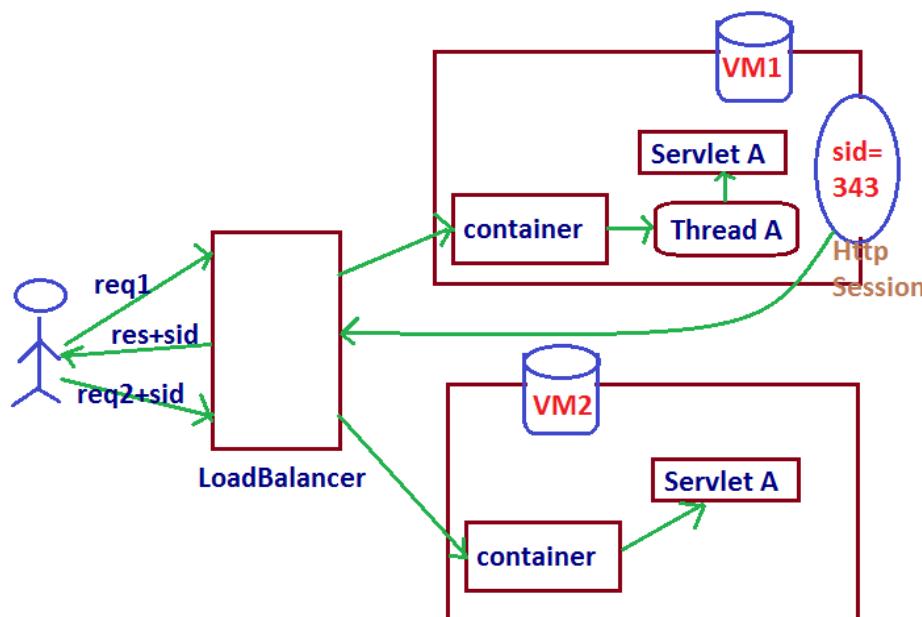
1) `public void sessionWillPassivate(HttpServletRequestEvent e)`

This method is called on each implementing object bound to the session just before serialization.

2) `public void sessionDidActivate(HttpServletRequestEvent e)`

This method will be executed on each implements object bound to the session just after de-serialization.

The session id 343 migrates from one VM1 to another VM2, in other words it is no longer exists on VM1 once it moves to VM2. This migration means the session was passivated on VM1 and activated on VM2.



The container gets the request the corresponding session id and realize this session id is on a different virtual machine i.e., VM1

O : requestA for ServletA could happen for VM1 and requestB for ServletA could end up on different VM , what happens to the things ServletContext, ServletConfig and HttpSession objects ?

Only HttpSession object (and their attributes) moves from one VM to another VM.

There is one ServletContext per one JVM, there is one ServletConfig per Servlet, per VM but there is only one HttpSession object for a given session id per web-app'n, regardless of how many virtual machines the app'n distributed across.

Note: HttpSessionActivationListener is also not required to configure in web.xml

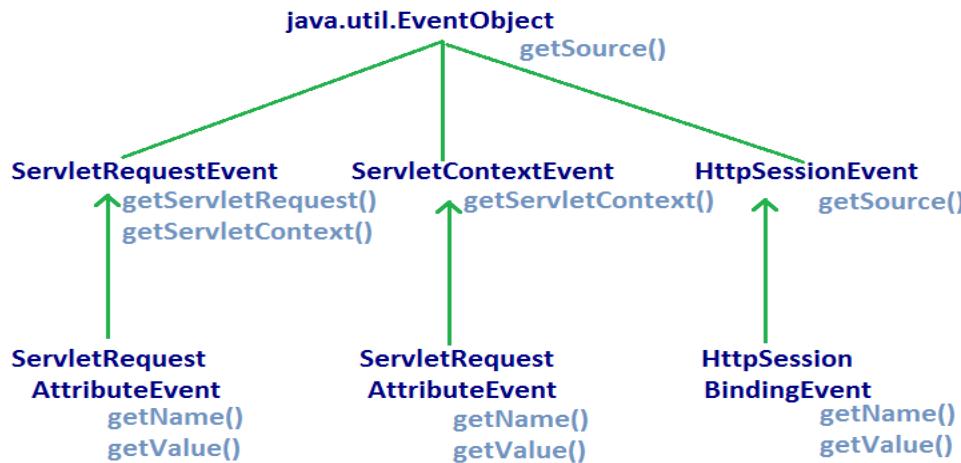
**HttpSessionActivationListenerDemo.java**

```
public class HttpSessionActivationListenerDemo implements HttpSessionActivationListener {
```

```

public void sessionDidActivate(HttpSessionEvent arg0) {
 System.out.println("object bound to the session just after De-Serialization");
}
public void sessionWillPassivate(HttpSessionEvent arg0) {
 System.out.println("object bound to the session just before Serialization");
}
}

```



Listener	Purpose	Corresponding Methods	Corresponding Events	Corresponding Event Methods	Is required to configure web.xml
Servlet Request Listener	To listens life cycle events of request object. i.e., request object creation & destruction	requestInitialized() requestDestroyed()	ServletRequest Event	getServletRequest() getServletContext()	Yes
Servlet Request Attribute Listener	To listens events related request scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	ServletRequest AttributeEvent	getName() getValue()	Yes
Servlet Context Listener	To listens life cycle events of context object. i.e., context object creation & destruction	contextInitialized() contextDestroyed()	ServletContext Event	getServletContext()	Yes

Servlet Context Attribute Listener	To listen events related to context scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	ServletContext AttributeEvent	getName() getValue()	Yes
HttpSession Listener	To listens life cycle events of session object. i.e., session object creation & destruction	sessionCreated() sessionDestroyed()	HttpSession Event	getSession()	Yes
HttpSession Attribute Listener	To listen events related to session scoped attributes	attributeAdded() attributeReplaced() attributeRemoved()	HttpSession AttributeEvent HttpSession BindingEvent	getName() getValue()	Yes
HttpSession Binding Listener	When ever we are adding or removing or replacing a particular type of object in session scope , to perform certain activity then we should go for this Listener	valueBound() valueUnbound()	HttpSession BindingEvent	getName() getValue()	not required
HttpSession Activation Listener	If we want to perform any activity just before serialization and just after de-serialization in distributed web-applications	sessionWillPassivate() sessionDidActivate()	HttpSession Event	getSession()	not required

LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...

**JAVA MEANS DURGASOFT**  
INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
SOFTWARE SOLUTIONS

# 202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

**LEARN FROM EXPERTS ...**

# **COMPLETE JAVA**

CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

# **COMPLETE .NET**

C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

# **TESTING TOOLS**

MANUAL + SELENIUM

# **ORACLE | D2K**

# **MSBI | SHARE POINT**

# **HADOOP | ANDROID**

# **C, C++, DS, UNIX**

# **CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED

**DURGA**  
Software Solutions®

# 202, 2nd Floor, HUDA Maitrivanam,

Ameerpet, Hyd. Ph: 040-64512786,

**9246212143, 8096969696**

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA**

**With**

# **SCWCD / OCWCD**

**Servlets Material**

## **5. Web Application Security**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

# **DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

## Web Application Security

- 1) Basic Terminology
  - Authentication
  - Authorization
  - Data Integrity
  - Confidentiality
- 2) Types of Authentication
  - BASIC
  - DIGEST
  - Form-based
  - Https client-cert
- 3) Declarative Security
- 4) Programmatic Security

### Basic Terminology :

Based on Servlet specification compare the following Security mechanisms

- 1) **Authentication:**  
It is process of validating the user, we can implement authentication by using userName and Password  
**Ex:** providing userName and password to login into bank site is called Authentication.
- 2) **Authorization :**  
It is process of validating access permissions of a user i.e, It is process of checking whether user allowed to access a particular resource or not after authentication we have to perform authorization by using Access Control List (ACL) we can implement authorization  
**Ex:** we are not allowed to access some other accounts information even though we are valid member of the bank i.e., we are not authorized to access some other account information.
- 3) **Data Integrity:**  
It is process of ensuring that data should not be changed in transformation from client to server by using Secure Socket Layer(SSL) , we can implement Data Integrity .
- 4) **Confidentiality :**  
It is process of ensuring that no one except intended user is able to understand our information. We can achieve this confidentiality by using encryption mechanisms .

**What is difference between Authorization and Confidentiality ?**

Authorization prevents information reaching and unintended users at beginning only , where as Confidentiality ensure that even though information falls in wrong hand it stills remains unreachable.  
Ex:Beer website application

1. should have userName and password to access this application(**authentication**).
2. Only premium users can get 10% discount (**authorization**).
3. When ever user places an order some sort of confermation is required(**Data Integrity**).
4. When ever customer places or type credit card information should send encrypted form and should not be misused (**Confidentiality**).

**www.durgasoftonlinetraining.com**

**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

**USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

**JAVA MEANS DURGASOFT**

**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
SOFTWARE SOLUTIONS

**# 202, 2<sup>nd</sup> FLOOR**  
**www.durgasoft.com**

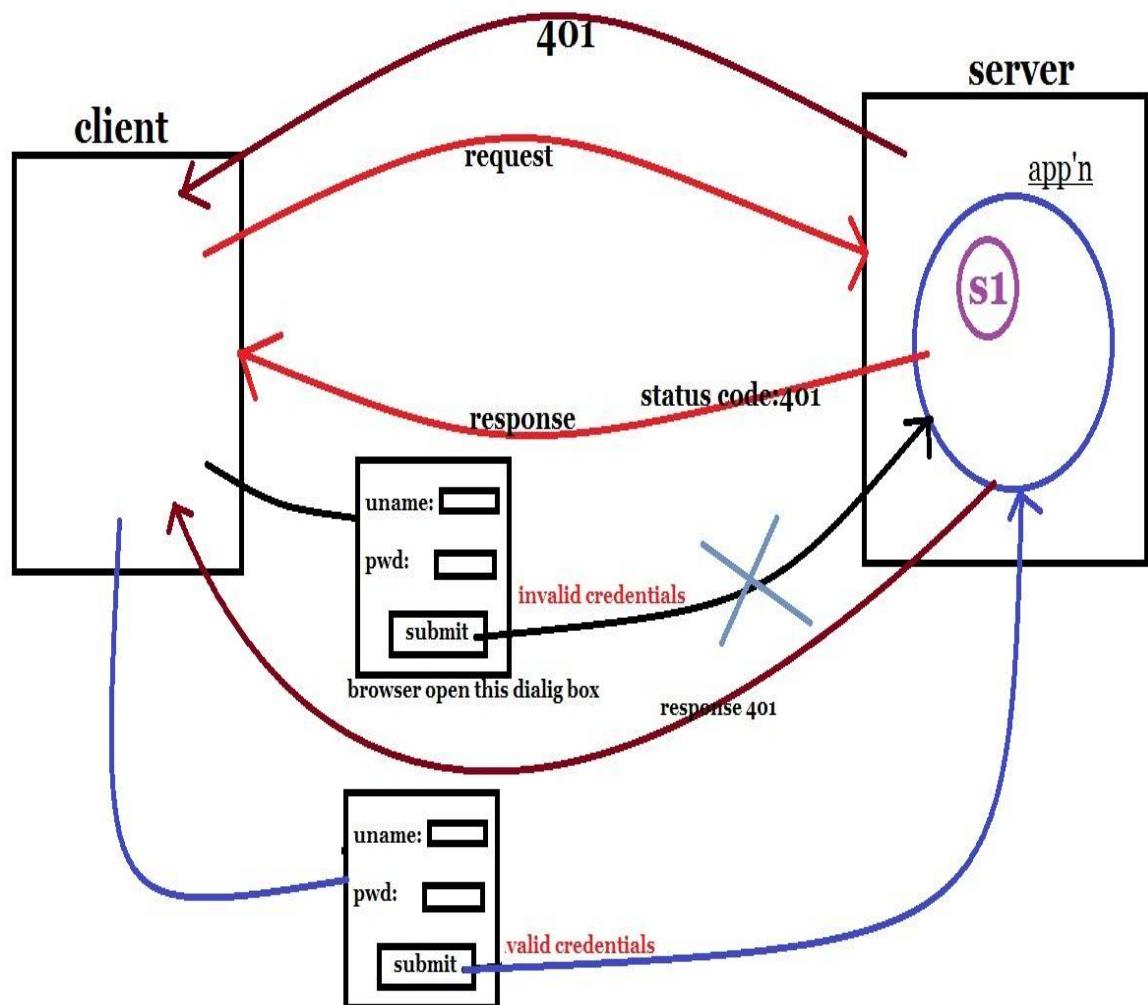
**040-64512786**  
**+91 9246212143**  
**+91 8096969696**

***Types of Authentication Mechanism :***

According to servlet specification 4 types of authentication mechanism possible.

**HTTP BASIC Authentication Mechanism :**

It is the most simplest and commonly used Authentication mechanism.  
The Basic Authentication Mechanism introduced in http 1.1 specification process.

**Basic Authentication Mechanism graphical representation:**

- 1) Browser sends a request to the server at this time browser don't know whether the request is protected or not , hence it sends normal request .

- 2) Server identifies the requested resource is secure hence instead of sending required response hence it will send 401 status code saying it requires authentication.
- 3) By receiving the 401 status code browser opens a dialog box prompting user Name and password , once the end user enter user Name and password browser resend the request to server with credentials.
- 4) When the server receives the request it validate the user Name and password the credentials are valid server sends proper required response otherwise it will send 401 status code application. (this process may happen maximum 3 number of times if still if you are giving wrong user Name and password this time 401 status code sending to the end user . )

**ADVANTAGES :**

- 1) It is very easy to implement and setup.
- 2) All browsers and all web servers can provide support for this mechanism.

**LIMITATIONS :**

- 1) user-name and password will send in plain text form from client to server , hence security is very less in this authentication mechanisms, this authentication mechanism follows BASE64 encoding technique/algorithum
- 2) We can't customize LookUP and Feel of dialog box

**DIGEST AUTHENTICATION MECHANISM:**

It is exactly similar to Basic except the password is sending encrypted form, this encryption makes more secure.

**ADVANTAGE:**

when compare to Basic Authentication mechanism DIGEST Authentication mechanism is more secure.

**LIMITATIONS:**

- 1) Only few browsers can provide support for this mechanism
- 2) Browser is responsible for this mechanism
- 3) Most of the browsers doesn't provide support for this DIGEST mechanism because servlet specification doesn't tell it is mandatory tecnic .  
(My web server doesn't know which encryption algorithm followed by browser , so that it gives decryption problems )
- 4) We can't customize Look and Feel of dialog box.

**FORM-BASED AUTHENTICATION MECHANISM:**

- 1) This mechanism is exactly similar to Basic authentication mechanism except that instead of depending on browser's dialog box , we can provide our own login frm or our own dialog box.
- 2) Developer is responsible to provide login and error pages , so that we can customize LOOK and FEEL based on our requirement.
- 3) The only requirement for the login form is :
  - The value of action attribute should be 'J\_Security\_Check'

- The form compulsory should contains 2 text fields with the names 'J\_username' and 'J\_password' except these all the remaining things are customizable.

**ADVANTAGES:**

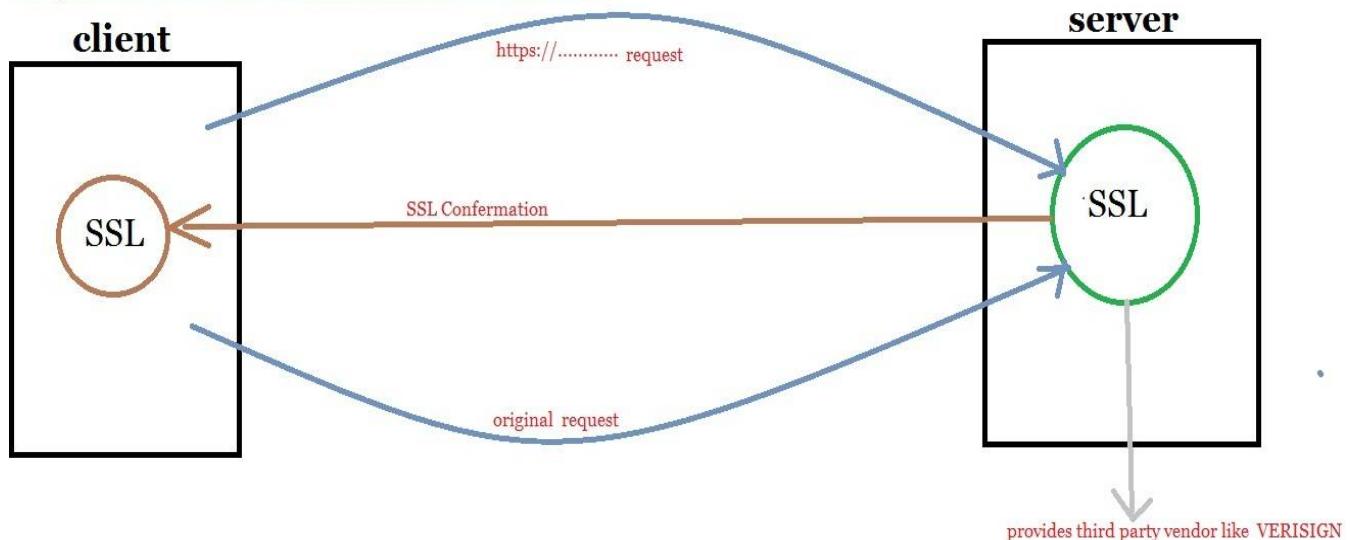
It is very easy to setup ,All browsers and all web servers can provide support for this mechanism.  
we can customize login form based on our requirement.

**LIMITATIONS:**

In this case also user-name and password will send in plain text form from client to server , hence security is very less .

**HTTPS CLIENT-CERT AUTHENTICATION MECHANISM:**

This is most secure type of authentication and it is most commonly used type of real time.  
HTTP means Http over secure socket layer.

**Https client-cert Authentication Mechanism**

SSL is a protocol to ensure the privacy of sensitive data transmitted over the internet.

In the mechanism Authentication is perform when SSL connection is established between client and server.

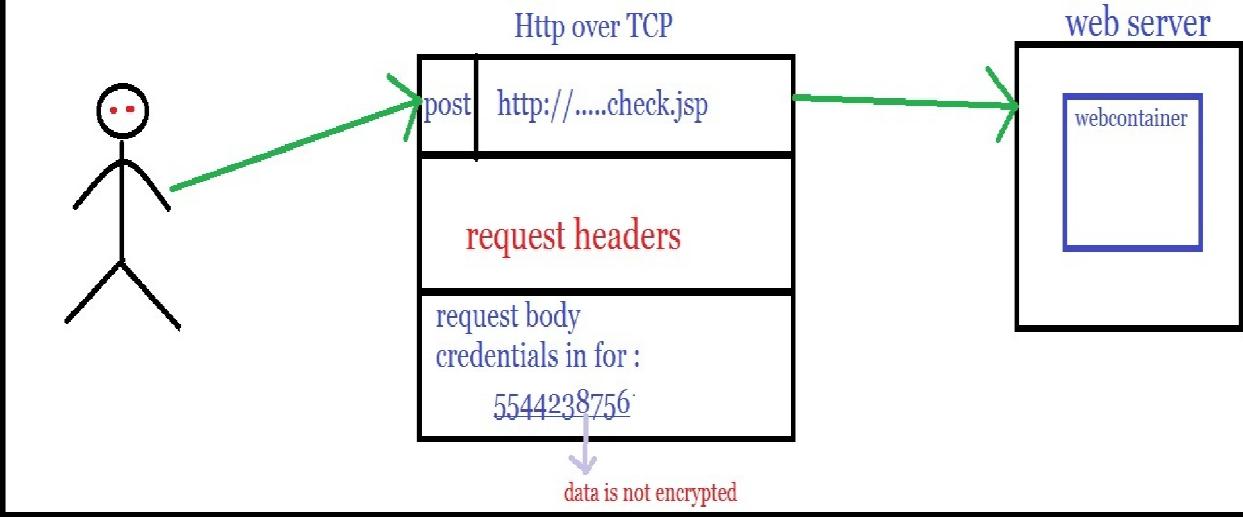
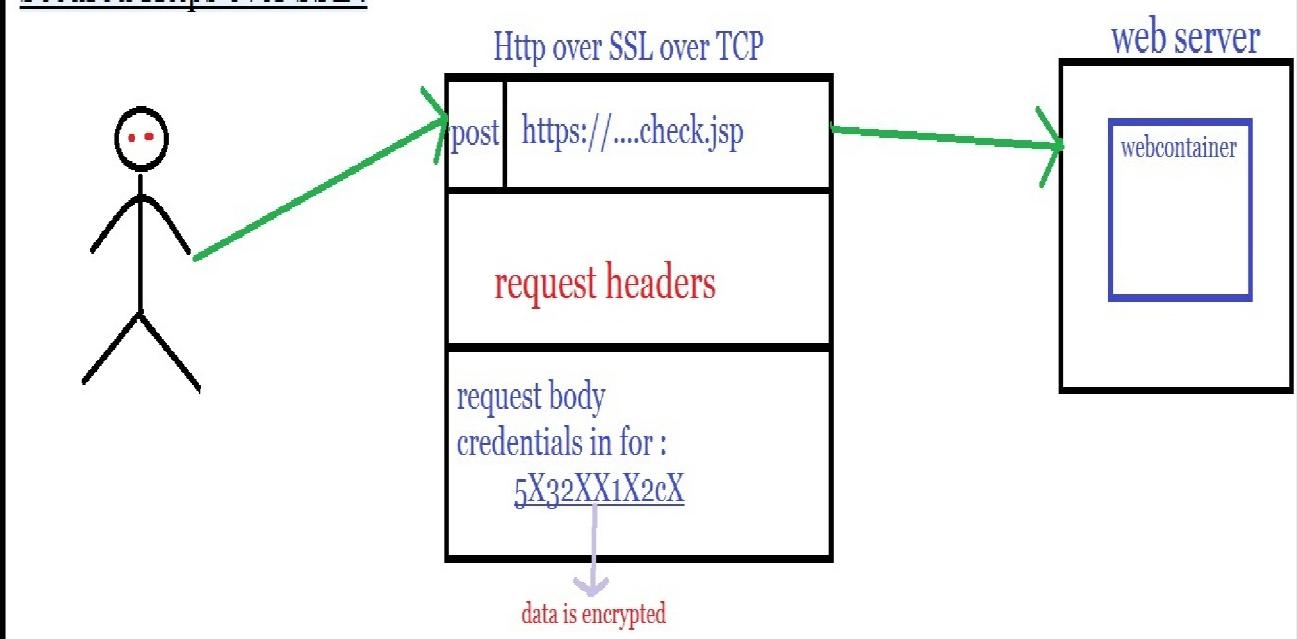
**ADVANTAGES:**

It is most secure type of authentication.

**DIS ADVANTAGES:**

It is costly to implement and maintain.

It receives a certificate from 3rd party vendors like verisign.

Http request not secured :Secured Https over SSL :

Type	Specification	DataIntegrity	Comments
BASIC	Http	Base 64-encoding	All browsers can provide support.
DIGEST	Http	Strong - but not SSL	Browsers and servers may not be support.
FORM BASED	J2EE	Very less security/weak encryption	We can customize login page .
CLIENT-CERT	J2EE	Strong-SSL(public key)	Strong , Users must have certificates .

## DECLARATIVE SECURITY:

**objective:** In the deployment descriptor declare .....

- 1) < security-constraint >
- 2) < login-config >
- 3) < security-role >

We can implement web security mainly by using following 3 tags.

1) **< security-constraint >**

It defines which resource has to be protected, which roles are allowed to access and how the resource is transported across the network

2) **< login-config >**

It defines the type of authentication we are using

3) **< security-role >**

It defines Security roles which are allowed in web application

All the above 3 tags are direct child tags of < web-app >

hence we can take anywhere within the < web-app > tag but it is convention to take the above order only.

### < security-constraint >

It defines the following 3 child tags.

1) **<web-resource-collection >**

It defines which resource has to be protected

2) **<auth-constraint >**

Authorization constraints which determine what roles are allowed to access these resources .

3) **< user-data-constraint >**

What type of protection is required when transporting the resource across network.

**<web-resource-collection >**

It contains there are 4 child tags.

- 1) <web-resource-name >
- 2) < description >
- 3) < url-pattern >
- 4) < http-method >

If we are specifying this tag then security constraint is applicable for all http methods (like GET, POST , HEAD , DELETE , TRACE , PUT , OPTIONS )

**< auth-constraint >**

It defines the following 2 child tags.

- 1) < description >
- 2) < role-name >

**< user-data-constraint >**

It defines the following 2 child tags.

- 1) < description >
- 2) < transport-guarantee >

This tag specifies what type of guarantee we have to provide while transporting the resource across network.

The allowed values for these tags are

NONE: it means the data is transported in plain-text form , it is default value .

INTEGRAL: it means the data should not be changed in transmission .

CONFIDENTIAL: it means the data is transported in encrypted form.

The priority order is : CONFIDENTIAL > INTEGRAL > NONE

**Note:** we have only one < user-data-constraint > per < security-constraint >

**< login-config >**

This tag specifies what type of authentication , we are using it contain the following 3 child tags.

- 1) < auth-method >

It represents authentication methods the allowed values are BASIC , DIGEST , FORM , CLIENT-CERT

- 2) < realm-name >

It represents location where we are storing authentication information. It is required only for BASIC authentication.

- 3) < form-login-config >

This tag is required to specify login page and error page in case of form based authentication mechanism.

This tag contains 2 child tags .

- < form-login-page >
- < form-error-page >

< security-role >

It can be used to defines security roles in web-applications

This tag contains the following 2 child tags.

- < description >
- < role-name >

Note: if we want to configure multiple roles that time each role a separate < security-role > tag is required.

Example:

```
< security-role >
 < role-name > jobs4times < / role-name >
< / security-role >
< security-role >
 < role-name > ashok < / role-name >
< / security-role >
```

**Write a program BASIC authentication mechanism:**

<http://localhost:7001/scwcd/login.jsp>

login.jsp

```
<form action="/BasicAuthentication" method="post">
 Enter name : <input type="text" name="uname">

 <input type="submit" value="submit">
</form>
```

BasicAuthentication

```
package com.jobs4times;
public class BasicAuthentication implements HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException {
 System.out.println("We are in doGet() method ");
 System.out.println("Get: After Authentication only we can access this servlet ");
 }
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException {
 System.out.println("We are in doPost() method ");
 System.out.println("Post: After Authentication only we can access this servlet ");
 }
}
```

web.xml

```
<web-app>
```

```
 <servlet>
 <servlet-name>BasicAuthentication</servlet-name>
 <servlet-class>com.jobs4times.BasicAuthentication</servlet-class>
 </servlet>
```

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

```
 <security-constraint>
 <web-resource-collection>
 <web-resource-name>basic</web-resource-name>
 <url-pattern>/BasicAuthentication</url-pattern>
 <http-method>GET</http-method>
 <http-method>POST</http-method>
 </web-resource-collection>
 <auth-constraint>
 <role-name>adminrole</role-name>
 <role-name>ashokrole</role-name>
 </auth-constraint>
 <user-data-constraint>
 <transport-guarantee>CONFIDENTIAL</transport-guarantee>
 </user-data-constraint>
 </security-constraint>
```

```
 <login-config>
 <auth-method>BASIC </auth-method>
 </login-config>
```

```
 <security-role>
 <role-name>ashokrole</role-name>
 </security-role>
```

```
 <security-role>
 <role-name>adminrole</role-name>
 </security-role>
```

```
</web-app>
```

tomcat-users.xml

```
<tomcat-users>
 <role rolename="adminrole"/>
 <role rolename="ashokrole"/>
 <user name="admin" password="scjp" roles="adminrole" />
 <user name="ashok" password="scwcd" roles="ashokadmin" />
 <!-- <user name="admin" password="scjp" roles="adminrole,ashokarole" />
 //both roles same username , pwd -->
</tomcat-users>
```

#### Rules for auth-constraint :

- 1) Within a security-constraint element auth-constraint element is an optional.
- 2) If an auth-constraint exists the container must perform authentication for associated URL's
- 3) If an auth-constraint exists doesn't exist the container must allow unauthenticated access for the URL's

Ex:

- 1) < auth-constraint >
 

```
< role-name > guest < /role-name >
 < role-name > admin < /role-name >
 < /auth-constraint >
```

Note: guest-role and admin-role can access
- 2) < auth-constraint >
 

```
< role-name >*< /role-name >
 < /auth-constraint >
```

Note: any role can access.
- 3) < auth-constraint / >
 

```
Note: no body can access.
```
- 4) If no < auth-constraint > in web.xml , that time every body can access , container must allow unauthenticated access for the URL's .

#### Rules for < role-name > tag :

1. Within the < auth-constraint > element < role-name > element is optional.
2. If < role-name > element exists, they tell to the container , which roles are allowed.
3. If < auth-constraint > element exists with no < role-name > element , then no users are allowed.
4. if < role-name > \* < role-name > Then all users are allowed.

#### Key points for <web-resource-collection> :

1. The <web-resource-collection> element to primary sub elements
  1. url-pattern
  2. http-method(optional , zero or more)

2. The <url-pattern> and <http-method> together define resource request that are constraints to be accessible by the only those roles defined in <auth-constraint>
3. web-resource-name tag is mandatory (even though you probably won't use it , but it is using web-container )
4. description tag is optional.
5. The <url-pattern> element follows same servlet standard naming conventions and rules.
6. You must specify at least one url-pattern but you can have meaning
7. Valid methods for http-method is GET,POST , PUT , HEAD, DELETE, TRACE, OPTION.
8. If no http-method's are specified then all methods will be constraints which means they can be accessed by only roles in <auth-constraint>
9. we can has more than one <web-resource-collection> with in the same <security-constraint> element.
10. The <auth-constraint> element applies all <web-resource-collection> element in the <security-constraint>

#### Writa a Program for FORM-BASED AUTHENTICATION MECHANISM:

##### login.html

```
<h3>Form Based Authentication Example :</h3>
<form action="j_security_check" method="post">
 <table border="2" bgcolor="lightgrey"><tr><td>
 UserName: <input type="text" name="j_username">
</td></tr><tr><td>
 Password:<input type="password" name="j_password">
</td></tr>
 <tr><td align="center"><input type="submit" value="Submit"></td></tr>
 </table>
</form>
```

##### error.html

```
<body>
 <pre>Your credentials are not correct

 please provide valid credentials</pre>
<body>
```

##### post.html

```
<form method="post" action=".//post">
 <input type="submit" >
</form>
```

##### FormServlet.java

```
package com.jobs4times;
public class FormServlet implements HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException {
 System.out.println("We are in doGet() method ");
 System.out.println("Get: After Authentication only we can access this servlet ");
```

```
 }
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException {
 System.out.println("We are in doPost() method ");
 System.out.println("Post: After Authentication only we can access this servlet ");
 }
}
```

[web.xml](#)

```
<web-app>

 <servlet>
 <servlet-name>FormServlet</servlet-name>
 <servlet-class>com.jobs4times.FormServlet</servlet-class>
 </servlet>

 <servlet-mapping>
 <servlet-name>FormServlet</servlet-name>
 <url-pattern>/FormServlet</url-pattern>
 </servlet-mapping>

 <security-constraint>
 <web-resource-collection>
 <web-resource-name>form</web-resource-name>
 <url-pattern>/FormServlet</url-pattern>
 <http-method>GET</http-method>
 <http-method>POST</http-method>
 </web-resource-collection>
 <auth-constraint>
 <role-name>ashokrole</role-name>
 </auth-constraint>
 </security-constraint>

 <login-config>
 <auth-method>FORM</auth-method>
 <form-login-config>
 <form-login-page>/login.html</form-login-page>
 <form-error-page>/error.html</form-error-page>
 </form-login-config>
 </login-config>

 <security-role>
 <role-name>ashokrole</role-name>
 </security-role>

```

</web-app>

[tomcat-users.xml](#)

```
<tomcat-users>
 <role rolename="ashokrole"/>
 <user name="ashok" password="scwcd" roles="ashokadmin" />
</tomcat-users>
```

**Programmatic Security:**

Some times declarative security may not enough to meet programming requirement.

**Ex:**

Based on the user role we have to provide customized response , if the user is admin then we have to provide admin related response.

If the user is manager , then we have to provide manager related response.

To meet this requirement compulsory we should go for programmatic security and Declarative security is notenough.

We can implement programmatic security by using the following methods of HttpServletRequest().

1) [public boolean isUserInRole\(String roleName\)](#)

If the authentication user belongs to specified role then this method returns true , if authenticated user not belongs specified role or if the user has not been authenticated then this method returns false.

2) [public string getRemoteUser\(\)](#)

This method returns authenticated user Name(loginName) , if the user has been authenticated then this method returns null .

**Example:**

```
public class FirstServlet implements HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException {
 System.out.println("We are in doGet() method ");
 if(request.isUserInRole("admin")) {
 //this is admin related response
 } else {
 //this is general response
 }
 }
}
```

The main problem is this approach is .....

we are hardcode in servlet ,

there is any change in the role-name modify the servlet code is costly operation and creates maintainability problems ,

to resolve this we should go for <security-role-ref> in web.xml,

by using this tag we can configure/map hardcoded role-name with original value.

[Write a demo program for programmatic security:](#)[login.html](#)

```
<h3>programmatic security:</h3>
<form action="/test" method="get">
 <table border="2" bgcolor="lightgrey"><tr><td>
 UserName: <input type="text" name="uname">
</td></tr><tr><td>
 Password:<input type="password" name="pwd">
</td></tr>
 <tr><td align="center"><input type="submit" value="Submit"></td></tr>
 </table>
</form>
```

[FirstServlet.java](#)

```
public class FirstServlet implements HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws
 ServletException, IOException {
 System.out.println("We are in doGet() method ");
 if(request.isUserInRole("hero")) {
 out.println("this is hero home page");
 } else {
 out.println("this is others home page");
 }
 }
}
```

[web.xml](#)

```
<web-app>

 <servlet>
 <servlet-name>first</servlet-name>
 <servlet-class>com.jobs4times.FirstServlet</servlet-class>
 <security-role-ref>
 <role-name>hero</role-name> // hardcoded rolename in servlet
 <role-link>adminrole</role-link> //original roleName
 </security-role-ref>
 </servlet>

 <servlet-mapping>
 <servlet-name>first</servlet-name>
 <url-pattern>/test</url-pattern>
 </servlet-mapping>

 <security-constraint>
 <web-resource-collection>
 <web-resource-name>checked</web-resource-name>
 <url-pattern>/test</url-pattern>
 </web-resource-collection>
 </security-constraint>

```

```

<http-method>GET</http-method>
<http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
 <role-name>adminrole</role-name>
 <role-name>ashokrole</role-name>
</auth-constraint>
</security-constraint>

<login-config>
 <auth-method>BASIC </auth-method>
</login-config>

<security-role>
 <role-name>adminrole</role-name>
</security-role>

</web-app>

```

[tomcat-users.xml](#)

```

<tomcat-users>
 <role rolename="adminrole"/>
 <role rolename="ashokrole"/>
 <user name="admin" password="scjp" roles="adminrole" />
 <user name="ashok" password="scwcd" roles="ashokadmin" />
</tomcat-users>

```

---

Multiple Security constraints elements with same name url-pattern ( or partially matching url-pattern and http-method elements )

---

[web.xml](#)

```

<web-app >
.....
<security-constraint>
 <web-resource-collection>
 <web-resource-name>display</web-resource-name>
 <url-pattern>/beer/updateReceipts/*</url-pattern>
 <url-pattern>/beer/displayReceipts/*</url-pattern>
 <http-method>POST</http-method>
 </web-resource-collection>

 <auth-constraint>
 //Authorization constraints with different roles
 </auth-constraint>
</security-constraint>

```

```

<security-constraint>
 <web-resource-collection>
 <web-resource-name>update</web-resource-name>
 <url-pattern>/beer/updateReceipts/*</url-pattern>
 <url-pattern>/beer/updateUser/*</url-pattern>
 <http-method>POST</http-method>
 </web-resource-collection>

 <auth-constraint>
 //Authorization constraints with different roles
 </auth-constraint>

</security-constraint>

</web-app>

```

How should container handle authorization when the same resource used by more than one

< security-constraint >

Content of A	Content of B	Whose has Access "Update Receipts"
< auth-constraint >         <role-name>guest</role-name>     </auth-constraint >	< auth-constraint >         <role-name>admin</role-name>     < auth-constraint >	Both guest , admin
< auth-constraint >         <role-name>guest</role-name>     </auth-constraint >	< auth-constraint >         <role-name>*</role-name>     </auth-constraint>	Every Body
< auth-constraint />	< auth-constraint >         <role-name>admin</role-name>     </auth-constraint>	No Body
no < auth-constraint > specified	< auth-constraint >         <role-name>admin</role-name>     </auth-constraint >	Every Body can Access

**Note:** When 2 different non-empty auth-constraint elements apply to the same constraint resource access is granted to the union of all roles from both of the auth-constraint elements .

#### Implementing Authentication:

4 login-config Examples :

web.xml

```
<web-app>
<login-config>
 <auth-method>BASIC</auth-method>
</login-config>
-----OR-----
<login-config>
 <auth-method>DIGEST</auth-method>
</login-config>
-----OR-----
<login-config>
 <auth-method>CLIENT-CERT</auth-method>
</login-config>
-----OR-----
<login-config>
 <auth-method>FORM</auth-method>
 <form-login-config>
 <form-login-page>/login.html</form-login-page>
 <form-error-page>/error.html</form-error-page>
 </form-login-config>
</login-config>
</web-app>
```

---

Except for form once we have declared login-config element in the deployment description(web.xml) , implementing authentication is done(assume we have already configured userName or password or roles configured at server level).

web.xml

```
<web-app>
<security-constraint>
 <web-resource-collection>
 <web-resource-name>
 <url-pattern>
 <http-method>
 <description>
 </description>
 </http-method>
 </url-pattern>
 </web-resource-collection>
 <auth-constraint>
 <description>
 <role-name>
 </role-name>
 </auth-constraint>
 <user-data-constraint>
 <transport-guarantee>
 </user-data-constraint>
</security-constraint>
<login-config>
```

```
<auth-method>
<relian-name>
<form-login-config>
 <form-login-page>
 <form-error-page>
</form-login-config>
</login-config>
<security-role>
 <role-name>
</security-role>
</web-app>
```

**FREE TRAINING VIDEOS**

**You Tube** **3000+  
VIDEOS**

**[www.youtube.com/durgasoftware](http://www.youtube.com/durgasoftware)**

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

**JAVA MEANS DURGASOFT**

**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
[www.durgasoftware.com](http://www.durgasoftware.com)

040-64512786  
+91 9246212143  
+91 8096969696

**LEARN FROM EXPERTS ...**

**COMPLETE JAVA**  
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,....

**COMPLETE .NET**  
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

**TESTING TOOLS**  
MANUAL + SELENIUM

**ORACLE D2K**

**MSBI | SHARE POINT**

**HADOOP | ANDROID**

**C, C++, DS, UNIX**

**CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED  
**DURGA** Software Solutions® # 202, 2nd Floor, HUDA Maitrivanam,  
Ameerpet, Hyd. Ph: 040-64512786,  
**9246212143, 8096969696**

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA**

**With**

# **SCWCD / OCWCD**

**JSP Material**

**1. The Java Server pages (JSP) Technology model**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

# **DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

## JSP's 2.1v

- 1) The Java Server pages (JSP) Technology model
- 2) Building JSP Pages Using Standard Actions
- 3) Building JSP Pages Using the Expression Language(EL)
- 4) Building JSP Pages using Tag Libraries(JSTL)
- 5) Building a Custom Tag Library

### **JSP Technology Model**

#### 1) JSP API and life cycle

- Translation Phase
- JSP-API
- javax.servlet.jsp.JspPage()
- javax.servlet.jsp.HttpJspPage()
- Internal implementation of HttpJspBase class ?
- LifeCycle of JSP
- PreCompilation of JSP

#### 2) JSP ELEMENTS :

- Template Text
- JSP Directives :

##### 1) page directive

- import
- session
- isELIgnored
- isThreadSafe
- contentType
- language
- buffer
- autoFlush
- extends
- info
- isErrorHandler
- errorPage
- pageEncoding

\* Summary of the page directive attributes

\* Is Servlet is ThreadSafe ?

\* Is Jsp is ThreadSafe by default ?

##### 2) include directive

Difference between include directive and include action :

### 3) taglib directive

### 3) Scripting Elements

#### ❖ Traditional Scripting elements

- Scriptlets
- Declarations
- Expressions
- Comments
  - Jsp comments
  - Html comments
  - Java comments
  - VISIBILITY of comments
  - Comparision of Scripting Elements

#### ❖ Modern Scripting elements

- EL
- JSTL

### 4) JSP Actions

### 5) JSP Implicit Objects/elements :

- ❖ request
- ❖ response
- ❖ config
- ❖ application
- ❖ session
- ❖ out :
  - How to write response in Jsp ?
  - What is the difference between PrintWriter and JspWriter ?
  - What is the difference between out.write(100) and out.print(100) ?
- ❖ page
- ❖ pageContext
  - To get all other Jsp implicit objects :
  - To perform RequestDispatcher Mechanism :
  - To perform Attribute Management in any scope (Jsp scopes)
- ❖ exception
  - Declarative Approach
  - Programmatic Approach
  - Which approach is best approach to configure error pages ?

### 6) JSP Scopes

- request scope
- session scope
- application scope (or) context scope
- page scope

## JSP API and life cycle

Describes the purpose and event sequence of JSP page life cycle.

1. Translation phase
2. Jsp page compilation
3. Load the class
4. Create an Instance (Instantiation)
5. Call `jsplInit()`
6. Call `_jspService()`
7. Call `jspDestroy()`

### Translation Phase :

1. The process of translating Jsp page (.jsp file) into corresponding Servlet (.java file) is called translation phase.
2. This can be done by Jsp container(engine) , In tomcat this engine is called "JASPER"
3. The generated .java file is available in the following location(Tomcat6.0\work\catalina\contextrootname\jsp1\org\apache\jsp)  
In weblogic : user-projects\domain\mydomain\myserver\.....

### JSP-API :

1. Every class which is generated for the jsp , must implements`javax.servlet.jsp.JspPage(I)` Or `javax.servlet.jsp.HttpJspPage` either directly or indirectly
2. Tomcat people provided a base class `org.apache.jasper.runtime.HttpJspBase` for implementing the above 2 interfaces. Hence any Servlet which is generated by tomcat is extending this `HttpJspBase` class.

**www.durgasoftonlinetraining.com**

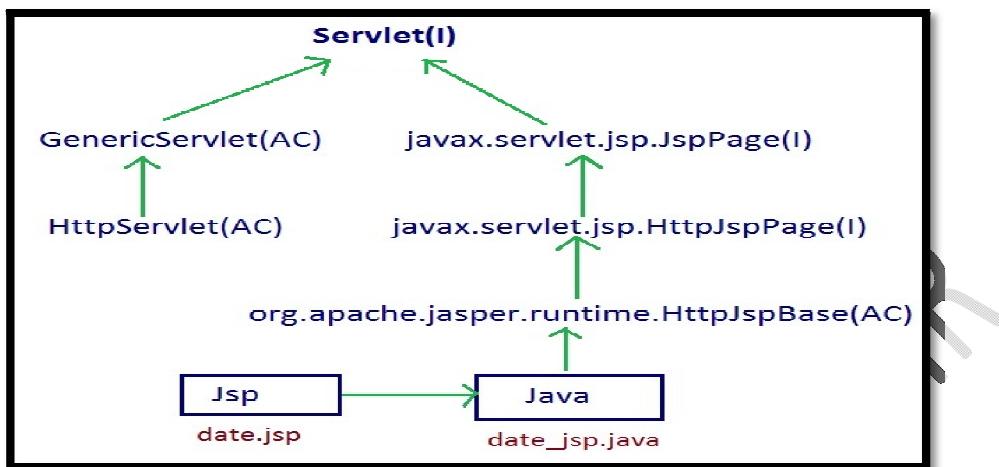


**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

**USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**



HttpJspBase class name will be valid from vendor to vendor (server to server)

#### **javax.servlet.jsp.JspPage(I) :**

This interface defines the following 2 methods

1. jsplInit()
2. jsplDestroy()

#### [jsplInit\(\) :](#)

**public void jsplInit()**

1. This method will be executed only once to perform initialization activities , whenever we are sending first request to the JSP
2. Web-container always calls init(ServletConfig sc) presents in HttpJspBase class , which intern calls jsplInit()
3. If we have any initialization activities we have to override jsplInit() in our Jsp

```

public class HttpJspBase --- {
 public final void init(ServletConfig config) throws ServletException {
 jsplInit();
 }
}

```

Various possibilities of overriding jsplInit() in our JSP

**date.jsp**

<%!

```

public void jsplInit() {
 system.out.println("jsp Init");

```

```
//our own initialization
}
%>
The server time is : <%=new java.util.Date()%>
```

This method will be executed at the time of first request. We can't override init(ServletConfig sc) in jsp, because it is declared as final in HttpJspBase class

### jspDestroy()

**public void jspDestroy()**

1. This method will be executed only once to perform cleanup activities just before taking jsp from out of service.
2. Web-container always calls destroy() available in HttpJspBase which intern calls jspDestroy()

```
class HttpJspBase {
 final destroy() {
 jspDestroy();
 }
 jspDestroy() {}
}
```

If we have any cleanUp activity we should override jspDestroy() & we can't override destroy() , because it is declared as final in HttpJspBase class.

### **javax.servlet.jsp.HttpJspPage()** :

1. It is the child interface of JspPage(), it contains only one method i.e., \_jspService()
 

```
public void _jspService(HttpServletRequest req , HttpServletResponse res) throws
ServletException, IOException
```
2. This method will be executed for every request.
3. Web-container always calls service(HSR,HSR) of HttpJspBase class, which intern calls our \_jspService(HSR,HSR)
4. \_jspService() should be generated automatically by the web-container at translation phase and we can't override.
5. If we write \_jspService() in the jsp explicitly then generated Servlet class contains two \_jspService() , which causes compile time error.(with in the same class 2 methods with same signature is not possible)
6. In our jsp we can't override service() , because these are declared as final in base class.

### Internal implementation of HttpJspBase class ?

```
public abstract class HttpJspBase extends HttpServlet implements HttpJspPage {
```

```

public final void init(ServletConfig config) throws ServletException {
 super.init(config);
 jsplInit();
}
public final void destroy() {
 jspDestroy();
}
public void jsplInit() {}
public void jspDestroy() {}
public final void service(HttpServletRequest req, HttpServletResponse res) throws
 ServletException, IOException {
 _jspService(req, res);
}
public abstract void _jspService(HttpServletRequest req, HttpServletResponse res) throws
 ServletException, IOException;
public String getServletInfo() {
 return "Jsp information";
}
}

```

**What is the signature of Underscore in \_jspService() ?**

We can't write this method explicitly and it should be generated automatically by the web-container.

In our Jsp , which of the following methods we can write explicitly ?

1. init(ServletConfig sc) //X-- final so wrong
2. jsplInit() //right
3. destroy() //wrong
4. jspDestroy() //right
5. service() //wrong
6. \_jspService() //wrong

**LifeCycle of JSP:**

1. jsplInit()
2. \_jspService(-,-)
3. jspDestroy()

These methods are called life cycle methods of a Jsp.

.jsp -----> .java -----> .class -----> load .class file -----> instantiation -----> jsplInit() --  
-----> \_jspService(-,-) ----->jspDestroy()

1. jsplInit() and jspDestroy() methods will be executed only once, But \_jspService() will be executed for every request.

2. If any problem occurs at the time of translation Or at the time of compilation we will get 500 status code response.
3. JspPage will be participated into translated in the following cases :
  1. At the time of 1<sup>st</sup> request
  2. When compared with earlier request , Source code of the jsp got modified . For this web-container will use ARAXIS tool to compare time stamps of .jsp and .class file

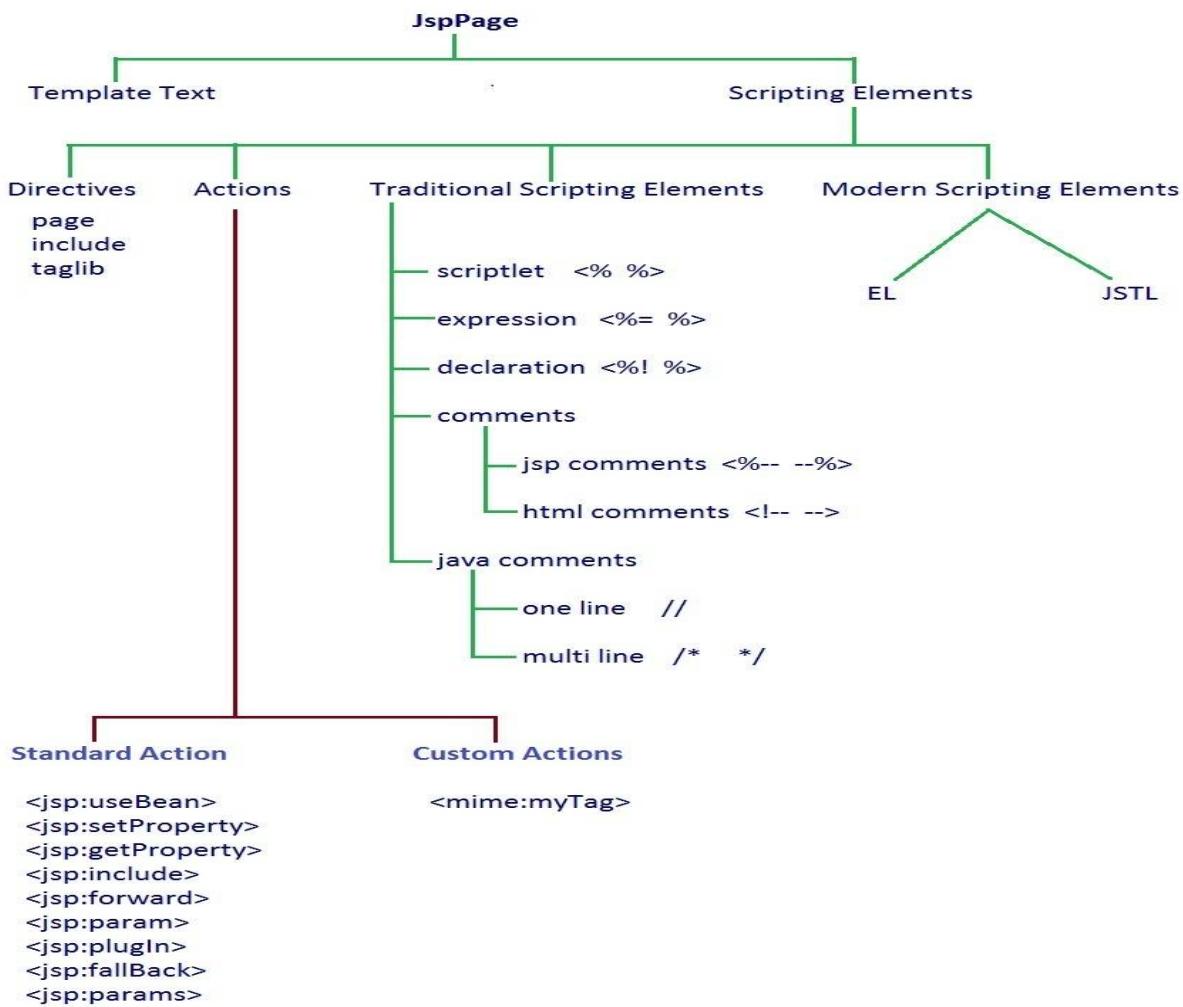
```
date.jsp (date_jsp.java)
<%!
 static {
 System.out.println("class is loading");
 }
 public date_jsp() {
 System.out.println("The generated java class object is created");
 }
 public void jsplnit() {
 System.out.println("jsplnit() will be called");
 }
 public void jspDestroy() {
 System.out.println("jspDestroy() will be called");
 }
%>
The server time is : <%= new java.util.Date0 %>
```

#### PreCompilation of JSP :

1. We can initiate translation phase explicitly by using pre-compilation process( upto jsplnit() ).
2. Jsp specification provides a special request parameter "jsp-precompile" to perform precompilation . We can use this parameter to hit the jsp for compilation without processing any request.
3. We can invoke pre-compilation as follows  
: http://localhost:8080/jsp1/date.jsp?jsp\_precompile=true [ upto jsplnit() will be executed ]
4. The main advantages of pre-compilation are
  1. All requests will be processed with uniform response time
  2. We can capture translation time errors and compilation time errors before processing first request.

While translation and compilation any problem or exception occurs then we will get http status code 500 response.

#### **JSP ELEMENTS :**



### Objective : Identify , describe and write Jsp code for the following elements

1. Template Text
2. Scripting elements
3. Standard and Custom Actions
4. Expression Language Elements

#### Template Text :

1. It consists of html and xml tags and raw data.
2. For the template text no translation is required , And it will become argument to out.write() present in \_jspService() of generated servlet.

**date.jsp**

The server time is :

```
<%= new java.util.Date() %>
```

**date\_jsp.java**

```
public final class date_jsp extends HttpJspBase {
 public void _jspService(HttpServletRequest request, HttpServletResponse response) {
 try {
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 out.write("The server time is:");
 out.print(new Date());
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}
```

The Template text will become argument to write() where as Expression value will become argument to print() , what is the reason.

write() can take only character data as argument and template text is always character data. Hence template text will become argument to out.write().

Expression value can be any data type , hence we can't use write() , compulsory we should go for print() which can take any type of argument.

### JSP Directives :

A directive is a translation time instruction to the jsp engine by using directives.  
we can specify whether current jsp error page or not, whether the current jsp participating into session traffic or not.

Syntax: <%@directive-name attribute-name=attribute-value1,... %>

There are 3 types of directives

#### 1) Page directive :

- It can be used to specify overall properties of jsp page to the jsp engine
- <%@ page isErrorPage="true" %>
- This directive specifies that the current jsp is error page and hence jsp engine makes exception implicit object available to the jsp page.

#### 2) Include directive :

We can use this directive to include the content of some other page into current page at translation time(static include)

<%@ include file="header.jsp" %>

#### 3) taglib directive :

We can use this directive to make custom tag functionality available to the jsp

<%@ taglib prefix="mime" uri="http://www.jobs4times.com" %>

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

**JAVA MEANS DURGASOFT**  
**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

**JSP Actions :**

Actions are commands to Jsp engine , They direct jsp engine to perform certain task at execution time(Runtime).

The following are various standard actions available in jsp

- 1) <jsp:useBean>
- 2) <jsp:getProperty>
- 3) <jsp:setProperty>
- 4) <jsp:include>
- 5) <jsp:forward>
- 6) <jsp:param>
- 7) <jsp:plugIn>
- 8) <jsp:fallBack>
- 9) <jsp:params>

Ex: <jsp: include page="header.jsp" />

It includes the response of header.jsp in the current page response at runtime.(Dynamic include).

**Scripting Elements :**

There are 2 types of Scripting elements

1. Traditional Scripting elements
  1. Expressions
  2. Scriptlets
  3. Declarative
  4. Comments
2. Modern Scripting elements
  1. EL
  2. JSTL

**Expressions :(<%= %>)**

- Expression can be used to print java expression to the jsp.  
Example : <%= 2+3 %> (OR) out.print(2+3);
- Expression will become argument to out.print() present in \_jspService()
- Inside jsp expression , expression value should not ends with semicolon otherwise we will get http status 500 error.  
Ex : <%= newDate(); %> (OR) out.print(new Date()); // 500 status code

- Inside expressions we can take method calls also but void return type method calls are not allowed.  
`<%= math.random() %> //valid  
<%=(new AL()).clear() %> // invalid  
because void return type methods are not allowed`
- Inside expression space is not allowed between % and = , otherwise it will be treated as scriptlet which causes compile time error .

```
<% =20 %>
_jspService() {
=20 ; // invalid statememt
}
```

Declarations are not allowed in expressions i.e., declaring variables,methods etc.,  
`<%= String s="durga" %>` OR

```
out.println(String s="durga"); //invalid
```

#### Which of the following Expressions are valid ?

- `<%= 2*3 %> //valid`
- `<%= 2<3 %> //valid`
- `<%= math.sqrt(4) %> //valid`
- `<%= 2*3 ;%> //invalid`
- `<%= new Student() %> // sop(new Student()); //valid`

#### Scriptlet : (<% %>)

- We can use scriptlet to place java code in the jsp

```
<% any java code %>
// this java code in _jspService()
```

- Scriptlet code will be placed directly inside `_jspService()` of generated servlet.
- Every java statement present inside scriptlet should ends with semicolon(:).

#### Write Jsp to print hit count :

```
<%
int count = 0;
count++;
//every request count initialized to 0 and print 1
out.println(count);
%>
```

This Jsp is invalid and generates '1' as response every time, here count variable is local.

```
<%
 int count = 0 ;
%>

<%
 count++;
 out.println(count);
%>

<%
/*
 any java code;
 it will be placed in generated servlet _jspService()
 */
%>

<%
 System.out.println("hello");
 // valid
%>

<%
 public int x=20; // invalid, x is not final
 out.println(x*2);
%>

<%
 final int x = 20; // valid, x is final
 out.println(x*2);
%>

<%
 public int m1() {
 int x = 10;
 int y = 20; //invalid
 return x*y;
 }
%>
```

Result is :<%= m1() %>

What is the difference between the following ?

<% int count=0 ; %>	<%! int count=0 ; %>
---------------------	----------------------

It is the local variable present inside  
\_jspService()

It is the instance variable present in generated Servlet , out  
side of \_jspService()

Note : It's never recommended to write scriptlets in the jsp.

### Declarations : (<%! %>)

1. We can use declaration tag to declare instance variables , instance methods, static variables , static methods , instance and static blocks , constructors, inner classes etc.,

```
<%!
 // any java declarations
%>
```

2. These declarations will be placed directly in the generated Servlet but outside of \_jspService()

```
<%!
 int x = 10 ;
 static int y = 20;
 public void m1() {
 System.out.println("allowed");
 }
%>
```

3. Every statement present in declaration tag should compulsory ends with semicolon.

```
<%!
 public void m1() {
 out.println("hello");
 }
%>
<%
 m1();
%>
```

```
class {
 public void m1() {
 out.println("Hello");
 }
 _jspService(- -) - {
 JspWriter out ;

 }
}
```

CE: out cannot be resolved

out is local variable to \_jspService() we can't access outside .

4. All implicit objects available in Jsp's are local variables inside \_jspService()
5. Declaration tag code will be placed outside of \_jspService() , Hence implicit objects we can't use in declaration tags. Otherwise we will get Compile time error.
6. But we can use implicit objects inside scriptlets and expressions because their code will be placed inside \_jspService()

### Comments :

There are 3 types of comments are allowed in Jsp

1. Jsp comments  
<%-- Jsp comments --%>
2. Html comments  
<!-- html/xml comments -->
3. Java comments
  1. Single line
  2. Multi line

### Jsp Comments :

<%-- Jsp comment --%>

1. Also known as hidden comments , because these are not visible in the remaining phases of jsp execution (like .java, .class, and response).
2. This comments are highly recommended to use in the Jsp.

### Html Comments :

<!-- Html/XML comments -->

1. Also known as template text comments
2. These are visible to the end-user as the part of generated response source code, hence these are not recommended to use with in the Jsp.

### Java Comments :

```
<%
// single line java comment
/* multi line java comment */
%>
```

1. Also known as scripting Comments
2. These comments are visible in the generated Servlet source code(.java) but in all other phases these are not visible.

### **VISIBILITY :**

Comments	Syntax	in JSP	in generated servlet source code(.java)	in end-user response
JSP	<%-- --%>	visible	not visible	not visible
Html	<!-- -->	visible	visible	visible

Java	// /* ..... */	visible	visible	not visible
------	-------------------	---------	---------	-------------

Translator can perform only one level translation , hence among declaration,scriptlet,expressions and comments we can't take another i.e., nesting of scripting elements are not allowed otherwise we will get compiletime error.

```
<%!
<%-- Jsp comments --%> // invalid
 int a=10;
%>
<%
 <% out.println("hello"); %>//invalid
%>
<%
 out.print(<%= 10+2+3 %>);//invalid
%>
<%--
 <% out.print("It is valid because i'm in comments"); %>
--%>
```

### Comparision of Scripting Elements :

Elements	Syntax	is ends with simicolon	code will be _jspService()
Scriptlet	<% %>	Yes	Yes
Expression	<%= %>	No	Yes
Declaration	<%! %>	Yes	No
Jsp Comments	<%-- --%>	not applicable	not applicable

### Directives :(<%@ %>)

Write a jsp code that uses the following directives are

1. page
2. include
3. taglib

Directives are translation time instruction to the jsp engine, by using these directives we can specify whether Jsp is an error page or not, whether Jsp is participating or not and etc.,

Syntax:

```
<%@ directive-name attribute-name=attribute-value1,..... %>
attribute-values should encode with singlecode(' ') or doublecode(" ")
```

**page :**

This page directive specifies overall properties of Jsp page to Jsp engine, we can use this directive no.of times and any where.

Syntax:

```
<%@ page attribute-name=attribute-value %>
```

The following are list of possible attributes of the page directive (13):

1. import
2. session
3. isELIgnored
4. isThreadSafe
5. contentType
6. language
7. buffer
8. autoFlush
9. extends
10. info
11. isErrorPage
12. errorPage
13. pageEncoding

**import :**

We can use this import attribute to import classes and interfaces in our Jsp, this is exactly similar to core java import statement.

date.jsp (with import attribute)

```
<%@ page import="java.util.Date" %>
The Server Time is : <%= new Date() %>
```

date.jsp (without import attribute)

```
The server time is: <%= new java.util.Date() %>
```

To import multiple packages the following various possibilities :

```
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
or
<%@ page import="java.util.*" import="java.io.*" %>
or
<%@ page import="java.util.*,java.io.*" %>
```

With in the same Jsp we are not allowed to take any attribute except import attribute of page directive multiple times with different values but we can take same attribute multiple times with same values for other attributes.

**Q : Which of the following are valid ?**

```
<%@ page import="java.util.*" import="java.io.*" %>
//valid
<%@ page session="true" session="false" %>
//invalid
<%@ page isELIgnored="true" isELIgnored="false" %>
<%@ page session="true" session="true" %>
//valid
<%@ page language=java %>
//invalid
```

The following packages are not required to import by default available in every Jsp.

1. `java.lang.*;`
2. `javax.servlet.*;`
3. `javax.servlet.http.*;`
4. `javax.servlet.jsp.*;`

**session :**

- We can use session attribute to make session object unavailable to the jsp by default session object is available to every jsp page.
- If we don't want to session object we have to declare page directive as follows

```
<%@ page session="false" %>
```

- In this case session object is not available in the jsp, in the rest of the jsp when ever we are trying to use this session object that time we will get an error like session can't be resolved.

```
<%@ page session="false" %>
<% out.println(session.getId()); %>
//session can't be resolved
%>
```

- If we are not declaring session attribute explicitly (or) declaring session attribute with true value , then the equivalent generated servlet code as follows
- 

```
HttpSession session=null;
session=pageContext.getSession();
```

If we take session attribute with false value then the generated servlet above 2 lines of code won't be available.

- The allowed values for session attribute are True/true/TRUE/TruE or False/false/FALSE/FalsE.
- In this case case is not important, consider only content.

- If we are taking other than true or false , then we will get translation time error.

```
<%@ page session="ashok" %> //invalid
```

**Which of the following make session object is available to Jsp ?**

```
<%@ page session="true" %> //valid
<%@ page session="false" %> //invalid
<%@ page session=true %> //invalid
<%@ page contentType="text/html" %> //valid
```

### isELIgnored

Expression Language introduced in jsp 1.2v

The main objective of EL is to eliminate java code from jsp

```
<%= request.getParameter("user") %>
Or
${param.user}
```

we can use isELIgnored attribute to enable or disable in our Jsp.

**isELIgnored="true"**

EL syntax just treated as template text without any processing

**isELIgnored="false"**

EL syntax will be evaluated and print its value.

```
<%@ page isELIgnored="true" %>
The result is : ${2+3}
output : The result is : ${2+3}
```

```
<%@ page isELIgnored="false" %>
The result is : ${2+3}
output : The result is : 5
```

The default value of Jsp 1.2v is "true".

But from Jsp 2.0v onwards default value is "false".

### isThreadSafe

We can use this attribute to provide Thread Safety to the Jsp

**isThreadSafe="true"**

It means the jsp is already thread safe and it is not required to implement SingleThreadModel interface by the generated sevlet

**isThreadSafe="false"**

The current Jsp is not Thread Safe hence to provide thread safety generated servlet should implement SingleThreadModel interface, In this case the jsp can process only one request at a time.

The default value for isThreadSafe value is "true".

**Which of the following to provide Thread Safety to the Jsp ?**

- <%@ page isThreadSafe="true" %> // not provide
- <%@ page isThreadSafe="false" %> // provide
- <%@ page isThreadSafe="true" isThreadSafe="false" %> // not provide
- No special arrangement is required because jsp is Thread Safe by default.

**contentType**

- We have to use this attribute to specify content type of the response(mime type)
- The default value of the contentType is "text/html".

**If we want to send JSON object as response to the client ,what is the content type required in the Jsp ?**

```
<%@ page contentType="application/json" %>
```

**language**

- Defines the language used in scriptlets, expressions, declarations, write now the only possible value is Java.
- The default value of the language is "java".

**Ex:** <%@ page language="java" %>

**buffer**

Defines how buffering is handled by the implicit out object (reference to JspWriter)

**autoFlush**

Defines whether the buffered output is flushed automatically , the default value is "true".

**extends**

Defines the super class of the class this jsp will become

```
<%@ page extends="com." %>/compiler error
```

**info**

Defines a string that gets put in to the translated page just show that we can get it using the generated servlet inherited `getServletInfo()` .

**isErrorPage**

It is defined in exception implicit object.

**errorPage**

It is defined in exception implicit object.

**pageEncoding**

Defines the character encoding for the jsp's the default is "ISO-8859-1"

**Summary of the page directive attributes :**

Attribute name	Purpose	default value
import	To import classes and interfaces of package	There is no default value, but default packages are java.lang.*; javax.servlet.*; javax.servlet.http.*; javax.servlet.jsp.*;
session	To make session object unavailable to Jsp	true
contentType	To specify mime type of the response	text/html
isELIgnored	To enable or disable Expression Language in our Jsp	false
isThreadSafe	To provide Thread Safety to the Jsp	true
language	If can be used to specify the scripting language which is used in Jsp	java
pageEncoding	Defines character encoding for the Jsp	ISO-8859-1
outoFlush	Defines whether buffered output is flush automatically	true

**Is Servlet is ThreadSafe ?**

No, by default Sevlet is not ThreadSafe and we can provide Thread Safety by implementing SingleThreadModel interface.

### Is Jsp is ThreadSafe by default ?

No, Jsp is not ThreadSafe by default and we provide Thread Safety by taking page directive.

```
<%@ page isThreadSafe="false" %>
```

### **Include Mechanism :**

- several Jsp's required for common functionality then it's recommended to write that common functionality separately in every jsp.
- We have to write that common code into separate file and we have to include that file wherever it is required.

The main advantages in this approach is

1. code re-usability
2. improves maintainability
3. Enhancements will be come very easy

We can achieve this include mechanism either by using include directive or include action.

### **include directive :**

Syntax : <%@ include file="header.jsp" %>

This inclusion will happen at translation time hence this inclusion is also called static include or translation time inclusion

### **include action :**

Syntax : <jsp: include page="footer.jsp" flush="true" />

This inclusion will happen at request processing time and hence it is considered as dynamic include or run time inclusion

header.jsp

### **Master in java certification**

footer.jsp

learning subject is not enough,  
utilization is very important

main.jsp

```
<%@ include file="header.jsp" %>
welcome to learners , it is enough
<jsp:include page="footer.jsp" flush="true" />
```

### Difference between include directive and include action :

include directive	include action
<%@ include file="header.jsp" %> It contains only one attribute and it is mandatory	<jsp: include page="footer.jsp" flush="true" /> It contains 2 attributes i.e., page,flush. page is mandatory,flush is optional
This inclusion will happen at translation time hence it's considered as static include.	This inclusion will happen at request processing time and hence it is considered as dynamic include.
For both including and included Jsp's a single servlet will be generated hence code sharing between the components is possible.	For including and included Jsp's separate servlets will be generated, hence code sharing between components is not possible.
Relatively performance is high	Relatively performance is low
There is no guarantee for the inclusion of latest version of included Jsp, it is a vendor dependent.	Always latest version of included page will be included.
If the target resource won't change frequently then it is recommended to use static include.	The target resource will change frequently then recommended to use dynamic include.

- To include header.jsp at translation time the required code is  
`<%@ include file="header.jsp" %>`
- To include header.jsp at request processing time the required code is  
`<jsp: include page="header.jsp" />`

1. `<%@ include page="footer.jsp" %> //invalid`
2. `<%@ include file="footer.jsp" flush="true" %> //invalid`
3. `<%@ include file="footer.jsp" /> //invalid`
4. `<%@ include file="footer.jsp" %> //valid`
5. `<jsp:include file="footer.jsp" flush="true" /> //invalid`
6. `<jsp:include page="footer.jsp" flush="true" %> //invalid`
7. `<jsp:include page="footer.jsp" flush="true" /> //valid`

- page attribute applicable for include action but not for include directive where as  
the, file attribute applicable for include directive but not for include action.
- flush attribute applicable only for include action but not for include directive.

#### Example :

header.jsp

```
<%!
int x=10 ;
%>
```

**main.jsp**

```
<%@ include file="header.jsp" %>
The result of x is : <%= x*100 %> //1000
```

**main.jsp**

```
<jsp:include page="header.jsp" />
The result of x : <%= x*100 %> //x can't be resolved
```

**taglib directive :**

**syntax :**

```
<%@ taglib prefix="mime" uri="www.jobs4times.com" %>
```

- We can use Taglib directive to make custom tag functionality available to the jsp
- taglib directive contains 2 attributes
  1. prefix
  2. uri
- prefix can be used with in the Jsp, uri can be used to specify location of TLD file.
- For every custom tag in the Jsp seperate taglib directive is required.

**JSP Implicit Objects :**

- Objective: For the given scenario write a jsp code by using appropriate jsp implicit objects.
- When compared with Servlet programming developing jsp's is very easy. Because the required mandatory stuff automatically available in every jsp.
- Implicit objects also one such area , implicit objects also bydefault available .

**The following is the list of all implicit objects :**

Implicit Objects	Type
request	javax.servlet.http.HttpServletRequest(I)
response	javax.servlet.http.HttpServletResponse(I)
config	javax.servlet.ServletConfig(I)
application	javax.servlet.ServletContext(I)
session	javax.servlet.http.HttpSession(I)
out	javax.servlet.jsp.JspWriter(AC)
page	java.lang.Object(CC)
pageContext	javax.servlet.jsp.PageContext(AC)
exception	java.lang.Throwable(CC)

**request & response implicit objects:**

- request and response implicit objects are available to the jsp are arguments to \_jspService()

- All the methods present in HttpServletRequest and HttpServletResponse can be applicable on this request and response implicit objects.

**index.html**

```
<form action="test.jsp">
 Enter UserName : <input type="text" name="uname" />

 Enter Course : <input type="text" name="course" />

 Enter Place : <input type="text" name="place" />

 <input type="submit" value="submit">
</form>
```

**test.jsp**

```
<%@page import="java.util.*" %>
The request method:<%= request.getMethod() %>
The User Name : <%=request.getParameter("uname") %>
<%
Enumeration e=request.getParameterNames();
while(e.hasMoreElements()) {
%
<%= request.getParameter((String)e.nextElement()) %>
<% } %>
<%
response.setContentType("text/xml");
response.setHeader("ashok","SCWCD");
%>
```

### application : (javax.servlet.ServletContext)

This implicit object of type javax.servlet.ServletContext , which provides environment of the web-application, what ever the methods available in ServletContext those methods are available application implicit object also.

**Ex:**

```
<%
RequestDispatcher rd = application.getRequestDispatcher("/header.jsp");//absolute path
rd.include(request,response); //OR
rd.forward(request,response);
%>
```

**application.jsp**

```
<%
java.util.Enumeration e=application.getInitParameterNames();
while(e.hasMoreElements()) {
%
<%= application.getInitParameter((String)e.nextElement()) %>
<% } %>
```

**web.xml**

```
<web-app>
 <display-name>MyJsp</display-name>
 <context-param>
 <param-name>uname</param-name>
 <param-value>ashok</param-value>
 </context-param>

</web-app>
```

**config**

This implicit object is of the type javax.servlet.ServletConfig what ever the methods present in ServletConfig interface, we can apply all these methods on config implicit object also.

The following are the methods available in ServletConfig interface

1. `public String getServletName()`
2. `public String getInitParameter(String pname)`
3. `public Enumeration getInitParameterNames()`  
<http://localhost:2018/app/config.jsp>
4. `public ServletContext getServletContext()`  
<http://localhost:2018/app/test>

**config.jsp**

The init parameter is :

```
<%= getServletConfig.getInitParameter("uname") %>
```

The servlet parameter is :

```
<%= config.getServletParameter("course") %>
```

The servlet name is :`<%= config.getServletName() %>`

**web.xml**

```
<web-app>
 <servlet>
 <servlet-name>configDemo</servlet-name>
 <jsp-file>/config.jsp</jsp-file>
 <init-param>
 <param-name>uname</param-name>
 <param-value>ashok</param-value>
 </init-param>
 </servlet>
 <servlet-mapping>
 <servlet-name>configDemo</servlet-name>
 <url-pattern>/test</url-pattern>
 <url-pattern>/test1</url-pattern>
 <url-pattern>/test2</url-pattern>
 </servlet-mapping>
</web-app>
```

If we are placing jsp with in the context root in the time, we can access that jsp the following 2 ways

1. By using it's name
2. By using it's url-pattern

`http://localhost:2018/jspApp/config.jsp`

Answer : null

```
null
jsp
(the default logical name of the jsp's is jsp)
http://localhost:2018/jspApp/test
Answer : ashok
test
configDemo
```

To result servlet level configuration for the Jsp's compulsory we should access the jsp by using it's url-pattern.

## page

page implicit object always pointing to current servlet object in the generated servlet it is declared as follows

```
public class
public void _jspService(,-,-)-
{
 Object page=this;
}
```

Which of the following are valid ?

1. `<%= page.getServletInfo() %>` //invalid
2. `<%= ((Servlet)page).getServletInfo() %>` //valid
3. `<%= ((HttpServlet)page).getServletInfo() %>` //valid
4. `<%= this.getServletInfo() %>` //valid
5. `<%= getServletInfo() %>` //valid

page implicit object is declared as `java.lang.Object` type, hence we are allowed to call only the methods present in the `Object` class but not Servlet specific methods.

Hence this implicit object is most rarely used implicit object.

## session

session implicit object is by default available to every jsp and it is of the type of `HttpSession`.

`<%@page session="true" %>`

The session timeout is : `<%= session.getMaxInactiveInterval() %>`

The session id is : `<%= session.getId() %>`

The session new is : `<%= session.isNew() %>`

We make a session object unavailable to the jsp

`<%@page session="false" %>`

```
<%= session.getMaxInactiveInterval() %>
//session can't be resolved
```

**Which of the following make session object available to the Jsp ?**

1. <%@page session=true %> //invalid
2. <%@page session="false" %> //invalid
3. <%@page session="true" session="false" %> //invalid
4. <%@page session="true" %> //valid
5. <%@page contentType="text/xml" %> //valid

**session.jsp**

```
<%@page session="true" %>
<% session.setAttribute("ashok","SCWCD"); %>
```

**date.jsp**

```
<%@page session="true" %>
<%= session.getId() %>
<%= session.getAttribute("ashok") %>
```

<http://localhost:2018/scwdcApp/session.jsp>

<http://localhost:2018/scwdcApp/date.jsp>

**same browser : (date.jsp)**

**output : same sessionid, SCWCD**

**different browser : (date.jsp)**

**output : different sessionid, null**

**out : javax.servlet.jsp.JspWriter(AC)**

- This implicit object of type javax.servlet.jsp.JspWriter, This object is specially designed for Jsp's to write character data to the response object.
- The main difference between PrintWriter and JspWriter in the case of PrintWriter there is no buffering concept is available . hence whenever we are writing any data by using PrintWriter it will be return directly into the response object.
- But in the case of JspWriter buffer concept is available, hence whenever we are writing by using JspWriter it will be written to the buffer and at the end total buffer data will be added to the response object.
- Except this buffer difference , there is no other difference between JspWriter and PrintWriter.  
**JspWriter=PrintWriter+buffer**

**out.jsp**

```
<%@page autoFlush="true" %>
<%
 java.io.PrintWriter pw=response.getWriter();
 pw.println("Good Morning Ashok");
 pw.println("Good Morning Akshay");
```

```

pw.println("Good Morning Arun");
pw.println("Good Morning Sai");
%>

```

**output:**

Good Morning Ashok  
 Good Morning Sai  
 Good Morning Akshay  
 Good Morning Arun

**Note:**

1. **autoFlush="true"** A value of true indicates automatically buffer flushing , A value of false indicates and value of false throws an exception.
2. It causes the servlet throws an exception when the servlet output buffer is full.
3. In Jsp can't set the false value for autoFlush attribute , if the buffer is "none".
4. The default buffer size is 8kb and none is also valid for the buffer attribute , In this case whenever we are writing any data by using JspWriter that time it will be return directly to the response object.

**out.jsp**

```

<%@page autoFlush="true" buffer="none" %>
<%
 java.io.JspWriter jw=response.getWriter();
 jw.println("Good Morning Ashok");
 jw.println("Good Morning Akshay");
 jw.println("Good Morning Arun");
 jw.println("Good Morning Sai");
%>

```

**output:**

Good Morning Ashok  
 Good Morning Akshay  
 Good Morning Arun  
 Good Morning Sai

- Within the Jsp we can use either PrintWriter or JspWriter but not recommended to use both simultaneously.
- JspWriter is the child class of Writer, hence all methods available in writer are available by default to the JspWriter through Inheritance.
- In addition to these methods JspWriter contains print(), println() methods, add any type of data add to these methods.

**Writer(AC) write(int)**  
**write(char)**  
**write(String)**

```
JspWriter(AC) print()
 println()
```

//AC = abstract class

### Example:

```
<%
 out.print("The PI value is :");
 out.println(3.14);
 out.print("This exactly ");
 out.println(true);
```

### output:

The PI value is : 3.14  
 This exactly true  
 %>

### How to write response in Jsp

By using out implicit object which is of the type Writer.

### What is the difference between PrintWriter and JspWriter ?

#### Buffering

JspWriter = PrintWriter + buffer

### What is the difference between out.write(100) and out.print(100) ?

- In the case of write(100), the corresponding character "d" will be added to the response.
- In the case of print(100) , the int value 100 will be added to the response.

#### Ex:

```
<%
 out.write(100); //d
 out.print(100); //100
%>
```

### **pageContext : (javax.servlet.jsp.PageContext)**

- The pageContext implicit object is a type of javax.servlet.jsp.pageContext(AC).
- It is an abstract class and vendor is responsible to provide implementation ,  
 PageContext is the child class of JspContext(jsp 2.0v)
- PageContext class is useful in ClassicTagModel.

We can pageContext implicit object for the following 3 purposes

1. To get all other Jsp implicit objects.
2. To perform RequestDispatcher Mechanism
3. To perform Attribute Management in any scope (Jsp scopes)

### How to get all other Jsp implicit objects ?

By using pageContext implicit object we can get any other jsp implicit objects, hence pageContext implicit object act as a single point of contact for the remaining implicit objects.

PageContext class defines the following methods to retrieve remaining all other implicit objects.

```
request --> getRequest()
response --> getResponse()
config --> getServletConfig()
application --> getServletContext()
session --> getSession()
page --> getPage()
out --> getOut()
exception --> getException()
```

- These methods are not useful within the Jsp because all jsp implicit objects already available to every Jsp,
- These methods are useful outside of Jsp mostly in custom tags(Tag Handler Classes).

### To perform RequestDispatcher Mechanism :

We can use pageContext implicit object to perform RequestDispatcher activity also, for this purpose pageContext class defines the following methods

```
public void forward(String targetPath);
public void include(String targetPath);
```

Note : All the rules of ServletRequestDispatcher will be applicable for pageContext implicit object based on RequestDispatcher mechanism.

#### first.jsp

This is RD mechanism using pageContext implicit object

```
<%
pageContext.forward("second.jsp");
System.out.println("control come back");
%>
```

#### second.jsp

This is second jsp

```
<% java.util.Enumeration e=request.getAttributeNames(); %>

```



### To perform Attribute Management in any scope (Jsp scopes) :

We can use pageContext implicit object to perform attribute management in any scope (will be covered in Jsp scopes)

**exception :**

configuring error pages in Jsp.

We can configuring the following 2 approaches

1. Declarative Approach
2. Programmatic Approach

### Declarative Approach :

We can configure error pages according to a particular exceptions or perticular error code in web.xml as follows

```
<web-app>
 <error-page>
 <exception-type>java.lang.ArithmetricException</exception-type>
 <location>/error.jsp</location>
 </error-page>
 <error-page>
 <error-code>404</error-code>
 <location>/error404.jsp</location>
 </error-page>
</web-app>
```

**Note :** The error pages configured in this approach is applicable entire web-application(not a particular servlet Or not a particular Jsp).

### Programmatic Approach :

We can configure error pages for a particular Jsp by using error page attribute of page directive.

**demo.jsp**

```
<%@page errorPage="error.jsp" %>
<% out.println(10/0); %>
```

- In demo.jsp any problem or exception occurs then error.jsp is responsible to report that error.
- This way of configuring error pages for a particular jsp we can declare jsp as a error page by using isErrorPage attribute of page directive.
- In error pages only exception implicit object is available.

**error.jsp**

```
<%@page isErrorPage="true" %>
The raised Exception is : <%= exception %>
//internally toString() calling
```

If Jsp is not configuring/declared as a error page and if we are trying to use exception implicit object then we will get compile time error saying that exception can't be resolved.

**error.jsp**

```
The raised Exception is : <%= exception %>
```

If we are trying to access error page directly without taking any exception then exception implicit object value refers null.

Ex: <http://localhost:2018/jspApp/error.jsp>

If we are configuring both declarative and programmatic approach on web-application, then programmatic approach will be effected.

**Which approach is best approach to configure error pages ?**

Declarative approach is best approach because we can customize error pages based on exception type and error code.

- jsp implicit objects are local variables of \_jspService() in the generated servlet, hence we are allowed to use these implicit objects inside scriptlet and expression tag because the corresponding code will be placed inside \_jspService(), but we are not allowed to access declaration tag because it's code will be placed outside of \_jspService().
- The most powerful implicit object is pageContext and rarely used implicit object is page.
- Among 9 implicit objects except session & exception , all remaining implicit objects are always available in every Jsp we can't make them unavailable.
- session implicit object is by default available in every jsp but we can make it's unavailable by using session attribute of page directive.

```
<%@page session="false" %>
```

- exception implicit object is by default not available to every jsp but we can make it's available by using isErrorPage attribute of page directive.

```
<%@page isErrorPage="true" %>
```

## JSP Scopes

In our servlets we have following 3 scopes

- request scope
- session scope
- application scope (or) context scope
- page scope (In addition to these scopes we have page scope also in our jsp's)

### request scope :

- In servlets this scope will be maintained by using ServletRequest object but in jsp's it is maintained by request implicit object.
- Information stored in request scope is available or applicable for all components which are processing the same request.
- The request scope will be started at the time of request object creation. i.e., just before starting service() and will be lost at the time of request object destruction, i.e., just after completion service().

We can perform attribute management in request scope by using the following methods of ServletRequest interface

- public void setAttribute(String aname, Object avalue)
- public Object getAttribute(String aname)
- public void removeAttribute(String aname)
- public Enumeration getAttributeNames()

### session scope :

- session scope will be maintained in our servlet by using HttpSession object but in Jsp's maintained by session implicit object.
- Information stored in the session scope is available or applicable for all the components which are participating in the session scope.
- session scope will be started at the time of session object creation and will be lost once session will be expire either by invalidate() or session time out mechanism.

We can perform attribute management in session scope by using the following methods of HttpSession interface

- public void setAttribute(String aname, Object avalue)

2. public Object getAttribute(String aname)
3. public void removeAttribute(String aname)
4. public Enumeration getAttributeNames()

#### application scope :

- We can maintain application scope in servlet by using ServletContext object but in Jsp's by using application implicit object.
- Information stored in the application scope is available or applicable for all the components of web-application.
- application scope will be started at the time of ServletContext object creation. i.e., at the time of application deployment or server startup, and ends at the time of ServletContext destruction i.e., application undeployment or server shutdown.

We can perform attribute management in application scope by using the following methods of ServletContext interface

1. public void setAttribute(String aname, Object avalue)
2. public Object getAttribute(String aname)
3. public void removeAttribute(String aname)
4. public Enumeration getAttributeNames()

#### page scope :

- This scope is applicable only for Jsp's but not for servlets, This scope will be maintained by pageContext implicit object (but not page implicit object).
- Information stored in the page scope is by default available with in the translation unit only.
- page scope is most commonly used scope in custom tag share information between TagHandler classes.

We can perform attribute management in page scope by using the following methods of PageContext class

1. public void setAttribute(String aname, Object avalue)
2. public Object getAttribute(String aname)
3. public void removeAttribute(String aname)



We can perform attribute management in any scope by using the following methods of pageContext implicit object :

We can use pageContext implicit object to perform attribute management in any scope for this purpose PageContext class following these methods

**JspContext(AC)**

```

1)public void setAttribute(String aname, Object avalue, int scope)
2)public Object getAttribute(String aname, int scope)
3)public void removeAttribute(String aname, int scope)
4) public Enumeration getAttributeNamesInScope(int scope)
5) public Object findAttribute(String aname) // more methods

```



**PageContext(AC)**

```

PAGE_SCOPE
REQUEST_SCOPE
SESSION_SCOPE
APPLICATION_SCOPE

```

// public static final fields

`public void setAttribute(String aname, Object value, int scope)`

The allowed values for the scope is :

pageContext.PAGE\_SCOPE or 1  
 pageContext.REQUEST\_SCOPE or 2  
 pageContext.SESSION\_SCOPE or 3  
 pageContext.APPLICATION\_SCOPE or 4

If we are passing other than these values we will get Runtime exception saying `java.lang.IllegalArgumentException(invalid scope)`

`public Object findAttribute(String aname)`

First this method searches in page scope for the required attribute if it is available it return the corresponding value, if it is not available then it will search in request scope followed by session and application scope(order is important).

page > request > session > application

suppose that attribute is not available even in application then this method returns null.

**demo.jsp**

```
<%
 pageContext.setAttribute("ashok","scwcd",2);
 pageContext.setAttribute("ashok","scjp",4);
 pageContext.forward("header.jsp");
 pageContext.setAttribute("akshay","scjp");
%>
```

- In this case , if we are not mention in any scope by default scope in Jsp is page scope.
- The information is available in current jsp but not outside of the Jsp.

**header.jsp**

```
<%
 out.println(pageContext.getAttribute("akshay"));
 out.println(pageContext.findAttribute("ashok"));
 out.println(pageContext.findAttribute("abc"));
%>
```

**Output:**

```
null
scwcd
null
```

#### Difference between getAttribute(-) and findAttribute(-) ?

- getAttribute(-) method it searches by default in page scope if we are not mention any scope.
- findAttribute(-) method it will search all scopes for the required attribute.

#### Which of the following are valid to store attribute in page scope ?

1. page.setAttribute("ashok","scjp"); //invalid
2. pageContext.setAttribute("ashok","scjp",2); //invalid
3. pageContext.setAttribute("ashok","scjp",pageContext.PAGE\_SCOPE); //valid
4. pageContext.setAttribute("ashok","scwcd"); //valid
5. pageContext.setAttribute("ashok","scjp", pageContext.PAGE\_CONTEXT\_SCOPE); //invalid

```
<%!
 // java comments are allowed in declarations
%>

<%
 // java comments are allowed in scriptlet and expression also
 /* java comments are allowed in scriptlet and expression also */
%>
```

**demo.jsp**

```
<%
 pageContext.setAttribute("ashok","SCWCD",3);
 pageContext.setAttribute("ashok","SCJP",4);
 pageContext.forward("header.jsp");
%>
```

**header.jsp**

```
<%@page session="false" %>
<% out.println(pageContext.findAttribute("ashok")); %>
// session scope is not available so then go for application scope
```

scope	in Servlets	in Jsp's(using jsp implicit objects)
application/context	getServletContext. setAttribute("ashok","scwcd");	application. setAttribute("ashok","scwcd");
session	request.getSession(). setAttribute("ashok","scwcd");	session. setAttribute("ashok","scwcd");
request	request. setAttribute("ashok","scwcd");	request. setAttribute("ashok","scwcd");
page	Not Applicable	pageContext. setAttribute("ashok","scwcd");

**www.durgasoftonlinetraining.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**



**USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

**LEARN FROM EXPERTS ...**

# **COMPLETE JAVA**

CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

# **COMPLETE .NET**

C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

# **TESTING TOOLS**

MANUAL + SELENIUM

# **ORACLE | D2K**

# **MSBI | SHARE POINT**

# **HADOOP | ANDROID**

# **C, C++, DS, UNIX**

# **CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED

**DURGA**  
Software Solutions®

# 202, 2nd Floor, HUDA Maitrivanam,  
Ameerpet, Hyd. Ph: 040-64512786,

**9246212143, 8096969696**

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA With SCWCD / OCWCD JSP Material**

**2. Building JSP Pages Using Standard Actions**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

# **DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

173

DURGASOFT, # 202, 2<sup>nd</sup> Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,  
040 - 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | [www.durgasoft.com](http://www.durgasoft.com)

## JSP Standard Actions

### Agenda:

- 1) <jsp:useBean> tag
- 2) <jsp:setProperty> tag
- 3) <jsp:getProperty> tag
- 4) <jsp:include> tag
- 5) <jsp:forward> tag
- 6) <jsp:param> tag
- 7) <jsp:plug-in> tag
- 8) <jsp:params> tag
- 9) <jsp:fallback> tag
- 10) Summary of Standard actions
- 11) JSP Documentation

### **Introduction :**

case 1: problem

```
public class JspDemoServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 String uname = request.getParameter("userName");
 request.setAttribute("UserName", uname);
 RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
 rd.forward(request, response);
 }
}
```

view.jsp

Welcome to : <%= (String)request.getAttribute("UserName") %>

OR

case 2 :

```
public class JspDemoServlet extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 String uname = request.getParameter("userName");
 foo.Person p = new foo.Person();
 p.setName(uname);
 request.setAttribute("person", p);
 RequestDispatcher rd = request.getRequestDispatcher("view1.jsp");
 rd.forward(request, response);
 }
}
```

**view1.jsp**

```
Welcome to :
<%@page import="foo.Person"%>
Welcome to :
<%
 Person p=(Person)request.getAttribute("person");
 out.println(p.getName());
%>
```

**Person.java**

```
package foo;
public class Person {
 String name;
 public void setName(String name) {
 this.name = name;
 }
 public String getName() {
 return name;
 }
}
```

**Solution : view1.jsp**

```
<jsp:useBean id="person" class="foo.Person" scope="request" />
Welcome to : <jsp:getProperty property="name" name="person" />
```

**Problem : demo.jsp**

```
<%!
 public int squareIt(int n) {
 return n*n;
 }
%>
<h1>The square of 4 is : <%= squareIt(4) %></h1>
<h1>The square of 2 is : <%= squareIt(2) %></h1>
```

**This approach has several serious dis-advantages :**

- There is no clear separation of presentation logic and business logic. The person who is writing the jsp should have compulsory knowledge on both java & html which may not be possible always, this approach doesn't promote re-usability .
- It reduces readability.
- We can resolve these problem by separating entire business logic inside java bean.

**javabean :**

```
package foo;
public class Calculator {
 public int squareIt() {
 return n*n;
 }
}
```

view.jsp

```
<jsp:useBean id="calc" class="foo.Calculator" scope="session" />
```

```
The square of 4 is : <%= calc.squareIt(4) %>
```

**This approach has several advantages :**

- Separation of presentation and business logic (presentation logic available in Jsp and business logic available in java bean) , so that readability is improved.
- Java developer can concentration on business logic where as Html developer can concentration on presentation logic as both can work simultaneously so that we can reduce development time and increases productivity.
- It promotes re-usability of the code i.e., where ever this squareIt() functionality is used we can reuse the same bean with rewriting.
- We can purchase bean for file uploading and we can start uploading of the file with in minutes.

**Note :** It is never recommended to write scriptlets, expressions, declarations and business logic with in the Jsp.

**java bean:** java bean is a simple java class

**To use useBean inside jsp the bean class has to follow the following rules**

- Jsp engine is responsible to perform instantiation of Java bean for this always executes public no argument constructor otherwise <jsp:useBean> tag won't be work. (because it internally calls that setter method only)
- For every property bean class should contains public getter methods otherwise<jsp:getProperty> tag won't be work.(because it internally calls corresponding getter method only)
- It is highly recommended to keep bean class as part of some package.

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

**JAVA MEANS DURGASOFT**  
INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

**Jsp Standard Actions :****<jsp:useBean> :**

We can use this tag to make bean functionality available to the Jsp, There are 2 forms of <jsp:useBean>

**without body :**

```
<jsp:useBean id="c" class="CustomerBean" />
```

**with body :**

```
<jsp:useBean id="c" class="CustomerBean" >

<jsp:useBean>
```

The main objective of body is to perform initialization for the newly created bean object, The bean object is already available then it won't be created new bean object it will reuse existing object only , in this case body won't be executed.

**Attribute list of <jsp:useBean> :**

1. id
2. class
3. type
4. scope
5. beanName

**id :**

- This id attribute represents name of the reference variable of the bean object.
- By means of this id attribute only we can access bean in rest of the Jsp's
- This attribute is a mandatory attribute.

```
<jsp:useBean id="c" class="CustomerBean" />
// the appropriate equivalent java code
CustomerBean c = new CustomerBean();
```

**class :**

- This attribute represents fully qualified name of the Java bean class, for this class only jsp engine perform instantiation hence the value of class attribute should be concrete class.

- abstract class and interfaces are not allowed.
- The bean class compulsory should contain public no-arg constructor otherwise we will get instantiation problem.
- This attribute is optional but when ever we are not specify class attribute compulsory we should specify type attribute.

**type :**

- We can use type attribute to specify the type of reference variable.
- The value of type attribute can be concrete class or abstract class or interface.
- This attribute is optional but when ever we are not specify type attribute compulsory we should specify class attribute.

```
<jsp:useBean id="p" type="foo.Person" class="foo.Student" />
// equivalent java code
foo.Person p = new foo.Student();
```

**scope :**

- This attribute specifies in which scope jsp engine has to search for the required bean object, in that scope if the bean object is not already available then Jsp engine creates a bean object and store into specified scope for the future purpose.
- The allowed values for the scope attributes are page, request, session, application.
- This attribute is optional and default scope is "page".

```
<jsp:useBean id="c" class="CustomerBean" scope="session" />
//equivalent java code
CustomerBean c=null;
synchronized(session) {
 c = (CustomerBean)pageContext.getAttribute("c", pageContext.SESSION_SCOPE);
 if(c == null) {
 c = new CustomBean();
 pageContext.setAttribute("c", c, pageContext.SESSION_SCOPE);
 }
}
```

If we specify only type attribute without class attribute then bean object should be already available with in the specified scope if it is not already available then we will get instantiation exception.

```
<jsp:useBean id="c" type="CustomerBean" scope="session" />
```

To use session scope compulsory session object should be available otherwise we will get translation time error.

```
<%@page session="false" %>
<jsp:useBean id="c" class="CustomerBean" scope="session" />
```

**Exception:** Illegal for useBean to use session scope when jsp page declare that it doesn't participate into the session.

```
package foo;
public class CalculatorBean {
 static int i = 0;
 public CalculatorBean() {
 System.out.println("calculator bean object created"+ (++i)+"times");
 }
 public int squareIt(int n) {
 return n*n;
 }
}
```

test.jsp

```
<jsp:useBean id="calc" class="foo.CalculatorBean" scope="session" />
The square of 4 is : <%= calc.squareIt(4) %>
The square of 5 is : <%= calc.squareIt(5) %>
Click Here
```

beanTest.jsp

```
<jsp:useBean id="calc" type="foo.CalculatorBean" scope="session" />
The square of 8 is : <%= calc.squareIt(8) %>
The square of 9 is : <%= calc.squareIt(9) %>
```

case 2 :

```
<jsp:useBean id="calc" type="foo.CalculatorBean" scope="session" >
 The square of 8 is : <%= calc.squareIt(8) %>
 The square of 9 is : <%= calc.squareIt(9) %>
</jsp:useBean>
```

case 3 :

```
<jsp:useBean id="c" type="foo.CalculatorBean" scope="session" >
 The square of 8 is : <%= calc.squareIt(8) %>
 The square of 9 is : <%= calc.squareIt(9) %>
</jsp:useBean>
```

exception: javax.servlet.ServletException:java.lang.InstantiationException

person.java

```
package foo;
public abstract class Person {
 public abstract String getMessage();
}
```

Student.java

```
package foo;
public class Student extends Person {
 public String getMessage() {
 return "Ashok";
 }
 public String getName() {
 return "Aggidi";
 }
}
```

**test.jsp**

```
<jsp:useBean id="p" type="foo.Person" class="foo.Student" scope="session" />
The person name is : <%= p.getMessage() %>
Click Here
```

**beanTest.jsp**

```
<jsp:useBean id="person" class="foo.Student" scope="session" />
The student name is : <%= person.getName() %>
```

**beanName :**

There may be a chance of using serialized bean object from local file system in that case we have to used beanName attribute.

**Conclusions for <jsp:useBean> :**

1. 'id' attribute is mandatory
2. 'scope' attribute is optional and default scope is page.
3. The attributes class, type, beanName can be used in the following combinations
  1. class
  2. type
  3. class and type
  4. type and beanName
4. beanName is always associated with type attribute but not class attribute.
5. If we are using class attribute whether we are using type attribute or not compulsory it should be concrete class and it should contains public no-arg constructor.
6. If we are using only type attribute without class attribute then the specified bean object compulsory should be available specified scope otherwise this tag won't create new bean object and raises instantiation exception.

Consider the following code

```
package pack1;
public class CustomerBean {

}
```

**Assume that no CustomerBean object is already created then which of the following standard actions create a new bean object and store in request scope ?**

1. <jsp:useBean id="c" class="pack1.CustomerBean" /> //invalid
2. <jsp:useBean id="c" type="pack1.CustomerBean" scope="request" /> //invalid
3. <jsp:useBean name="c" class="pack1.CustomerBean" scope="request" /> //invalid
4. <jsp:useBean id="c" class="pack1.CustomerBean" scope="request" /> //valid

### **<jsp:getProperty> :**

This tag can be used to get the properties of bean object

```
<%
 CustomerBean c = new CustomerBean();
 out.println(c.getName());
%>
```

//equalant jsp

```
<jsp:useBean id="c" class="CustomerBean" />
<jsp:getProperty name="c" property="name" />
<jsp:getProperty> tag contains the following 2 attributes
```

1. name
2. property

#### **name :**

It represents name of the bean object reference variable from which the required property has to retrieve , it is exactly equal to id attribute of <jsp:useBean>.

#### **property :**

It represents the name of the java bean property whose value has to retrieve.

Note : both attributes are mandatory attributes.

```
package foo;
public class CustomerBean {
 private String name = "ashok";
 private String mail = "ashok@jobs4times.com";
 public String getName() {
 return name;
 }
 public String getMail() {
 return mail;
 }
}
```

index.jsp

```
<%@page import="foo.CustomerBean" %>
<%
 CustomerBean c=new CustomerBean();
 out.println("The customer name is :" +c.getName());
 out.println("The customer mail is :" +c.getMail());
%>
```

(OR)

```
<jsp:useBean id="c" class="foo.CustomerBean"/>
```

The Customer name is : <jsp:getProperty property="name" name="c"/>  
The Customer mail is : <jsp:getProperty property="mail" name="c"/>

#### **<jsp:setProperty> :**

We can use this tag to set the properties of bean object , **<jsp:setProperty>** tag contains the following forms

form 1 :

```
<jsp:setProperty property="age" name="c" value="30" />
(OR)
c.setAge("30");
```

form 2 :

```
<jsp:setProperty property="age" name="c" param="age" />
(OR)
c.setAge(request.getParameter("age"));
```

**Note :** The param attribute to retrieve the request parameter value "age" and assign in to java bean property age.

form 3 :

```
<jsp:setProperty property="age" name="c" />
(OR)
c.setAge(request.getParameter("age"));
```

**Note:** To get this type of advantage it is highly recommended to maintain bean property names same as form parameter names.

form 4 :

```
<jsp:setProperty property="*" name="c" />
// "*" specifies all the properties of java bean
```

**Note :** It iterates all request parameters and if any request parameter name match with java bean property name then assign request parameter value to the java bean property.

<jsp:setProperty> contains the following attributes

1. name
2. property
3. value
4. param

[name :](#)

It represents name of the reference variable of the bean object whose property has to set, this is exactly same as id attribute of <jsp:useBean>, It is mandatory attribute.

[property :](#)

It represents name of the java bean property which has to set, it is mandatory attribute.

[value :](#)

It specifies the value which has to set to the java bean property, it is an optional and should not come combination with param attribute.

[param :](#)

This attribute specifies name of the request parameter whose value has to set to the java bean property, it is an optional attribute and should not come combination with value attribute.

**CustomerBean.java**

```
package foo;
public class CustomerBean {
 private String name = null;
 private String mail = null;
 private String age = null;
 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 public String getMail() {
 return mail;
 }
 public void setMail(String mail) {
 this.mail = mail;
 }
}
```

```

public String getAge() {
 return age;
}
public void setAge(String age) {
 this.age = age;
}
}

```

**index.jsp**

```

<form action = "test.jsp" method="post">
 <h2>Enter Your Details :</h2>
 Enter Name : <input type="text" name="uname">
 Enter Email : <input type="text" name="mail">
 Enter Age : <input type="text" name="age">
 <input type="submit" value="submit">
</form>

```

**test.jsp**

```

<h1>Set Property Example : </h1>
<jsp:useBean id="c" class="foo.CustomerBean" />
<jsp:setProperty property="*" name="c"/>
Entered Name is :
<jsp:getProperty property="name" name="c" />
Entered Mail is :
<jsp:getProperty property="mail" name="c" />
Entered Age is :
<jsp:getProperty property="age" name="c" />

```

**Note :** All required type conversions(String to int) will be performed automatically by **<jsp:setProperty>**.

**Case 1 :**

```

test.jsp
<h1>Set Property Example : </h1>

<jsp:useBean id="c" class="foo.CustomerBean" />

<jsp:setProperty property="mail" name="c"
 value="<% request.getParameter("mail") %>" /> //valid
<jsp:setProperty property="mail" name="c"
 value="<% request.getParameter("age") %>" /> //invalid

```

Entered Name is :

```

<jsp:getProperty property="name" name="c" />

```

Entered Mail is :

```
<jsp:getProperty property="mail" name="c" />
```

Entered Age is :

```
<jsp:getProperty property="age" name="c" />
```

**Note :** Automatic String to primitive conversions doesn't work if we use scripting elements, it fails even if we use expression inside `<jsp:setProperty>` standard actions.

```
<jsp:useBean id="c" class="foo.CustomerBean" />
<jsp:setProperty property="*" name="c" /> //valid
<jsp:setProperty property="emplId" name="c" param="emplId" /> //valid
<jsp:setProperty property="emplId" name="c" /> //valid
<jsp:setProperty property="emplId" name="c" value="1123" /> //valid
```

**Note :** If we are using scripting elements inside jsp standard actions automatic conversions doesn't work but it is possible scripting elements inside standard actions.

```
<jsp:useBean id="c" class="foo.CustomerBean" />
<jsp:setProperty property="emplId" name="c"
value="<% request.getParameter("emplId") %>" /> //invalid
```

### Developing reusable web-components :

We can develop reusable web-components by using the following 3 standard actions

1. `<jsp:include>`
2. `<jsp:forward>`
3. `<jsp:param>`

#### `<jsp:include>` :

The response of include page will be included in current response at request processing time, hence it is a dynamic include.

This tag representation of `pageContext.include("header.jsp");`

This standard action contains the following 2 attributes

1. `page`
2. `flush`

- **page** : page attribute represents name of the included page and it is mandatory.
- **flush** : It specifies whether the response will be flushed before inclusion or not , it is an optional attribute and default value is false.

#### index.jsp

```
<jsp:include page="header.jsp" />
```

```
<h2>Welcome to Jobs4Times.com</h2>
<jsp:include page="footer.jsp" />
```

header.jsp

```
<h2> We are Master in Java Certification </h2>
```

footer.jsp

```
<h4>copyright © www.jobs4times.com </h4>
```

**Following 4 possible ways to implement include mechanism in JSP's :**

- 1) By using include directive :

```
<%@include file="header.jsp" %>
```

- 2) By using include action :

```
<jsp:include page="header.jsp" flush="true" />
// by default flush is false
```

- 3) By using pageContext implicit object :

```
<% pageContext.include("header.jsp"); %>
```

- 4) By using RequestDispatcher :

```
<%
RequestDispatcher rd=request.getRequestDispatcher("header.jsp");
rd.include(request,response);
%>
```

**<jsp:forward> :**

If the first jsp is responsible to perform some preliminary processing activities and second jsp is responsible for providing complete response then we should go for forward mechanism.

**Syntax :**

```
<jsp:forward page="second.jsp" />
```

It contains only one attribute, i.e., page , and it is a mandatory attribute.

first.jsp

```
<h2>This is first Jsp page </h2>
<jsp:forward page="second.jsp" />
```

```
<h2>This is after forwarding </h2>
```

second.jsp

```
<h2>This is second Jsp page </h2>
```

**Following 3 possible ways to implement forward mechanism in JSP's :**

1. By using forward action :

```
<jsp:forward page="second.jsp" />
```

2. By using pageContext implicit object :

```
<% pageContext.forward("second.jsp"); %>
```

3. By using RequestDispatcher :

```
<%
 RequestDispatcher rd=request.getRequestDispatcher("header.jsp");
 rd.forward(request,response);
%>
```

**<jsp:param> :**

While forwarding or including if we want to send parameters to the target Jsp we can achieve this by using **<jsp:param>** tag

**<jsp:param>** tag defines the following 2 mandatory attributes

1. **name** : It represents name of the parameter.
2. **value** : It represents value of the Parameter.

**Note :** The parameters which are sending by using **<jsp:param>** tag are available as form parameters in target Jsp.

first.jsp

```
<jsp:include page="second.jsp">
<jsp:param value="ashok" name="username"/>
<jsp:param value="25" name="age"/>
</jsp:include>
```

second.jsp

```
<%@page isELIgnored="false" %>
<h2>Hi i'm getting the values from Jsp param</h2>
The UserName is : <%=request.getParameter("username") %>
The Age is : <%=request.getParameter("age") %>
```

**Conclusions :**

**case 1 :**

```
<jsp:forward page="http://localhost:2020/jspApp/second.jsp"/> //invalid
<jsp:include page="http://localhost:2020/jspApp/second.jsp"/> //invalid
<%@include file="http://localhost:2020/jspApp/second.jsp"%> //invalid
```

The value of page and file attributes should be a relative path or absolute path but not server port, protocol and etc.,

**case 2 :**

```
<jsp:forward page="/test" /> //valid
<jsp:include page="/test" /> //valid
<%@include file="/test" %> //invalid
```

In the case of forward and include actions page attribute can pointing to a servlet.

But in the case of include directive file attribute can't pointing to a servlet but it can pointing to a Jsp,html,xhtml,xml and etc.,

**case 3 :**

```
<jsp:forward page="/test?name=ashok" /> //valid
<jsp:forward page="second.jsp?name=ashok" /> //valid
<jsp:include page="second.jsp?name=ashok" /> //valid
<%@include file="second.jsp?name=ashok" %> //invalid
```

In the case of include and forward actions we are allowed to pass query string,

But in the case of include directive we are not allowed to pass query string.

**<jsp:plugin> :**

- Using **<jsp:plugin>** action provides the support for including java applet in a jsp page, If we have java applets that are part of your web-applications.
- While it's possible to have appropriate html code embedded in your Jsp page, the use of **<jsp:plugin>** allows the functionality to the browser is neutral.

There are the following 2 optional support tags those work with **<jsp:plugin>**. i.e., **<jsp:params>** and **<jsp:fallback>**

**<jsp:params> :**

We can pass any additional parameters to the target applet or bean.

**<jsp:fallback> :**

Which specifies any content that should be display to the browser, if the plugin is not started or because of some runtime issues.

A plugin specific message will be presented to the end user.

A translation time error will occurs if we use `<jsp:params>`, `<jsp:fallback>` any other context other than child of `<jsp:plugin>`.

**Syntax :**

```
<jsp:plugin code="classFileName .class extentions"
codebase="The directory of class file name" type="applet/bean" >
{align="alignment"}
{height=""}
{width=""}

{ <jsp:params>
 { <jsp:param value="pvalue" name="pname"/> }
</jsp:params> }

<jsp:fallback> Arbitrary test </jsp:fallback>
</jsp:plugin>

{} ---->optional attributes
```

**type :**

The type of object plugin will execute you must specify either applet or bean, as this attribute there is no default value.

It is a mandatory attribute.

`type="applet/bean"`

**code : (class="ClassName")**

The name of the java class file that plugin will execute we must include .classexension, file name is relative to the directory named in codeBase attribute , it is also mandatory attribute.

**codeBase : (codeBase="ClassFileDirectoryName")**

This is a absolute path or relative path to the directory that contains applet code, it is also mandatory attribute.

index.jsp

```
<jsp:forward page="plugIn.jsp"></jsp:forward>
```

plugIn.jsp

```
<h2>Welcome to Jsp Application</h2>
```

**First Applet :**

```
<jsp:plugin code="ClockApplet.class" codebase="applets" type="applet" width="260" height="150">
<jsp:fallback>Plug in tag not supported </jsp:fallback>
</jsp:plugin>
```

**Second Applet :**

```
<jsp:plugin code="edu.com.MessageApplet.class"
 codebase="applets" type="applet" width="500" height="100">
<jsp:params>
 <jsp:param value="Welcome to SCWCD" name="msg"/>
</jsp:params>
<jsp:fallback>No support to this Applet </jsp:fallback>
</jsp:plugin>
```

**ClockApplet.java**

```
public class ClockApplet extends Applet implements Runnable {
 String time;
 public void init() {
 Thread t = new Thread(this);
 t.start();
 }

 public void Paint(Graphics g) {
 g.setColor(Color.lightGray);
 g.fillRect(40, 40, 150, 100, true);
 g.setFont(new Font("monotypeCorsiva",Font.ITALIC,35));
 g.setColor(Color.red);

 g.drawString(time, 60, 100);
 }

 public void run() {
 while(true) {
 time = String.valueOf (new SimpleDateFormat("hh:mm:ss").format(new Date()));
 repaint();
 try {
 Thread.sleep(1000);
 } catch (InterruptedException e) {
 e.printStackTrace();
 }
 }//run
 }
}//class
```

**MessageApplet.java**

```
package edu.com;
```

```

public class MessageApplet extends Applet implements Runnable {
 String message;
 public void init() {
 message = getParameter("msg");//available in applet class
 if(message == null) {
 message = "This is default message";
 }
 }

 public void Paint(Graphics g) {
 g.setColor(Color.BLUE);
 g.setFont(new Font("monotype Corsiva",Font.ITALIC,40));
 g.drawString(message, 20, 50);
 }

 public void run() {}

}

```

first.jsp

```

<h2>This is first Jsp Page</h2>
<%! String nextPage="second.jsp"; %>
<jsp:forward page="<%= nextPage %>" /> //valid
 // static or dynamic
<h2>This is after forwarding</h2>

```

second.jsp

```
<h2>This is second Jsp Page</h2>
```

### Summary of Standard actions :

Standard action	Purpose	attributes
<jsp:useBean>	To make bean object available to the Jsp	id, class, type, scope, beanName
<jsp:getProperty>	To get and print properties of Java bean	name, property
<jsp:setProperty>	To set the properties of bean	name, property, value, param
<jsp:forward>	To forward a request from one Jsp to another Jsp	page
<jsp:include>	To include the response of some other Jsp into current Jsp at request processing time	page, flush
<jsp:param>	To send parameters to the target Jsp while forwarding or including	name, value
<jsp:plugin>	To embedded a java applet/bean into your Jsp	type, code, codeBase, {height,width,align}

<jsp:params>	To pass additional data or parameters to the target applet or java bean	no
<jsp:fallback>	A plugin specific message will be display to the end-user when a plugin is not started or due to some runtime issue	no



#### JSP Documentation :

They are 2 types syntaxes are possible to write a Jsp

1. Jsp Standard syntax
2. Xml based syntax

- Jsp Standard Syntax : If we are writing a Jsp by using Jsp standard syntax such type of Jsp's are called Jsp pages.
- Xml based syntax : If we are writing Jsp by using Xml based syntax such type of Jsp's are called Jsp document.

	Standard Syntax	XML based Syntax (all are case sensitive)
scriptlet :	<% %>	<jsp:script>
declararion :	<%! %>	<jsp:declaration>
expression :	<%= %>	<jsp:expression>

#### directives :

directives	Standard Syntax	XML based Syntax (all are case sensitive)
page	<%@page import="java.util.*" %>	<jsp:directive.page import="java.util.*" />

include	<%@include file="second.jsp" %>	<jsp:directive.include file="second.jsp" />
taglib	<%@taglib prefix="mime" uri="www.jobs4times.com" %>	For the taglib directive there is no equivalent syntax in XML but we can provide its purpose by using <jsp:root>

**Note :**

```
<jsp:root xmlns:mime="www.jobs4times.com" version="4.3" />
//version is mandatory attribute
//xmlns:mime is an optional attribute
//and we can take any no.of xml name spaces
```

**Standard actions :**

There is no difference in standard actions representation between standard syntax and XML based syntax.

**Ex :**

	Standard Syntax	XML based Syntax
useBean	<jsp:useBean id="c" class="CustomerBean" />	// same

**Comments :**

JSP specification doesn't provide any specific syntax for writing comments, hence we can use normal syntax

Jsp comment	XML comment
<%-- --%>	<!-- -->

**Template text :**

JSP Standard syntax doesn't provide any specific way to write template text but XML based syntax it provides a special tag <jsp:text>

Jsp standard syntax	XML based syntax
This is second JSP	<jsp:text> This is second JSP </jsp:text>

**How the web-container identifies JSP document ?**

1. Save the JSP file with .jspx extension.
2. Enclose entire JSP with in <jsp:text> tag.
3. Use <jsp-config> element in web.xml as follows

```
<web-app>
 <jsp-config>
 <jsp-property-group>
 <url-pattern>/jspx/*</url-pattern>
 <is-xml>true</is-xml>
 </jsp-property-group>
 </jsp-config>
</web-app>
```

**Note :** From servlet 2.4v onwards web-container can identify Jsp document automatically , no special arrangement is required.

### Write a Jsp document to print hit count of the Jsp ?

index.jsp (servlet 2.5v)

```
<jsp:directive.page isELIgnored="false" />
<!--
 including header.jsp to current jsp
-->
<jsp:directive.include file="header.jsp" />
<jsp:declaration> int count=0; </jsp:declaration>
<jsp:scriptlet> count++; </jsp:scriptlet>
<jsp:text>The hit count of this page is : </jsp:text>
<jsp:expression> count </jsp:expression>
```

- It is never recommended to use both standard and XML based syntax simultaneously.
- The main advantage of XML based syntax is we can use XML editors like XML-spy for writing and debugging JSP's
- All XML based tags and attributes are case-sensitive and attributes enclosed either single single quote('') or double quote("") .

### Which of the following valid way of importing java.util.ArrayList in our Jsp ?

```
<jsp:page import="java.util.ArrayList" /> //invalid

<jsp:page.directive import="java.util.ArrayList" /> //invalid

<jsp:directive page import="java.util.ArrayList" /> //invalid

<jsp:directive.page import="java.util.ArrayList" /> //valid
```

Person.java

package foo;

```
public class Person {
 private String name;
 private Dog dog;
```

```

public String getName() {
 return name;
}
public void setName(String name) {
 this.name = name;
}
public Dog getDog() {
 return dog;
}
public void setDog(Dog dog) {
 this.dog = dog;
}
}

```

**Dog.java**

```

package foo;
public class Dog {
 private String name;
 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
}

```

**JspDemoServlet.java**

```

public class JspDemoServlet extends HttpServlet {

 public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 foo.Person p = new Person();
 p.setName("ashok");

 foo.Dog d = new Dog();
 d.setName("spike");

 p.setDog(d);

 request.setAttribute("person", p);
 RequestDispatcher rd=request.getRequestDispatcher("view.jsp");
 rd.forward(request, response);
 }
}

```

**view.jsp**

The Person's Dog name is :

```
<%=(foo.Person)request.getAttribute("person")).getDog().getName()%>
```

It is never recommended to use Scripting elements in Jsp , hence the above solution is not proper approach

```
<jsp:useBean id="person" class="foo.Person" scope="request" />
```

The Person's Dog name is :

```
<jsp:getProperty property="dog" name="person" />
```

output:

The Person's Dog name is : foo.Dog@1234...

//If request scope is not specified the output is null

Solution : \${Person.dog.name}

The <Jsp:getProperty> tag lets you access only property of bean attributes, there is no capability for nested properties, where you want a property of property rather than property of attribute.

#### Limitations of Standard Actions :

Standard Actions can handle String and primitive properties , if we know the standard actions can deal with in a attribute of any type as long as all the attribute properties are String/primitives. there is no capability for nested properties.

**www.durgasoftonlinetraining.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

**USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

**LEARN FROM EXPERTS ...**

**COMPLETE JAVA**  
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

**COMPLETE .NET**  
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

**TESTING TOOLS**  
MANUAL + SELENIUM

**ORACLE D2K**

**MSBI SHARE POINT**

**HADOOP ANDROID**

**C, C++, DS, UNIX**

**CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
Software Solutions® # 202, 2nd Floor, HUDA Maitrivanam,  
Ameerpet, Hyd. Ph: 040-64512786,  
**9246212143, 8096969696**

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA With SCWCD / OCWCD JSP Material**

**3. Building JSP Pages Using the Expression Language(EL)**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

# **DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

**173**

DURGASOFT, # 202, 2<sup>nd</sup> Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,  
040 - 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | [www.durgasoft.com](http://www.durgasoft.com)

## Building JSP Pages Using Expression Language (EL)

Agenda:

1) Introduction

2) EL Implicit Objects :

- pageScope, requestScope, sessionScope, applicationScope
- param & paramValues
- header & headerValues
- initParam
- cookie
- pageContext
- handling errors

3) EL Operators :

- property access operator ( . )
- collection access operator [ ]
- Arithmetic operators ( +, -, \*, /, % )
- Relational operators ( lt, gt, eq, ne ) OR ( <, >, <=, >=, ==, != )
- Logical operators ( and, or, not )
- conditional operator ( ?: )
- empty operator ( empty )
- EL operator presidency
- EL reserved words
- EL Vs null

4) EL Functions :

- Write a java class
- Write a TLD file : (tag library descriptor file)
- Write a taglib directive
- Write EL function call
- Work Flow : (in EL function application)

### Introduction

- E-L introduced in Jsp 2.0v , the main objective of EL is to eliminate java code from the jsp.
- In general we can use EL with JSTL and custom tags for complete elimination of java code from the jsp.
- EL expressions are always with in { } and prefixed with \$ (dollar) sign
- syntax:
- \${leftVatable.rightVariable}

- The leftVariable can be any EL implicit objects/attributes stored in some scopes (page scope, request scope, session scope, application scope)

- EL Implicit Objects (11)
- EL Operators
- EL Functions

To print the value of request parameter "user" in jsp, write the code ?

```
<%= request.getParameter("user") %>
```

// is equivalent code

`${param.user}`

To print the value of session scoped attribute "x" in jsp ?

```
<%= session.getAttribute("x") %>
```

// is equivalent code

`${sessionScope.x}`

To use any variable x in EL compulsory it should be an attribute of some scope ?

`${x}`

It prints the value of x attribute present in any scope(pageScope, requestScope, sessionScope, applicationScope) and if there is no such attribute then it prints BlankSpace but not null

**Ex 1:**

```
<%!
 int x = 10;
%>
The value of x is : ${x}
```

**output :**

The value of x is : //BlankSpace

**Ex 2:**

```
<%@page isELIgnored="false" %>
<%! int x=10; %>
<% pageContext.setAttribute("x",x); %>
The value of x is : ${x}
```

**output :**

The value of x is : 10

**EL Implicit Objects (11) :**

EL contains 11 implicit objects

The power of EL is just because of these implicit objects only.

The following is the list of EL implicit objects

1. pageScope	----> map objects to retrieve scoped attributes
2. requestScope	
3. sessionScope	
4. applicationScope	
5. param	----> map objects to retrieve form parameters
6. paramValues	
7. header	----> map objects to retrieve request headers
8. headerValues	
9. cookie	----> map objects to retrieve cookies
10. initParam	----> map objects to retrieve context parameters
11. pageContext	----> java bean object (it is not a map)

**pageScope, requestScope, sessionScope, applicationScope**

1. pageScope	----> pageContext.getAttribute()
2. requestScope	----> request.getAttribute()
3. sessionScope	----> session.getAttribute()
4. applicationScope	----> application.getAttribute()

**Ex : To print the value of session scoped attribute "x"**

`${sessionScope.x}`

**Ex :** To print the value of request scoped attribute "x"

`${requestScope.x}`

**Ex :**

`${sessionScope.x}`

It prints the value of session scoped attribute "x" , in sessionScope if there is no such attribute then we will get BlankSpace

**Ex :**

`${x}`

jsp engine first checks in pageScope for attribute "x" if it is available then it prints the value, then it's not available it will check in requestScope followed by sessionScope and applicationScope, it simply act as `pageContext.findAttribute(x);`

**scopes.jsp**

**Note :**  `${leftVariable.rightVariable}`

- If leftVariable is a map then rightVariable should be "key", If the specified key is not available then we will get BlankSpace as a output.
- If leftVariable is a bean then rightVariable should be "property", If the specified property is not available then we will get "PropertyNotFoundException".

`${person.empld}`

// here person attribute stored in some scope

In Servlet code :

```
package foo;

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;

public class ELServletDemo extends HttpServlet {

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

```

foo.Person p = new foo.Person();
p.setName("Akshay");

foo.Dog d = new Dog();
d.setName("puppy");

p.setDog(d);

request.setAttribute("person", p);
request.setAttribute("arun", "SCWCD");

RequestDispatcher rd = request.getRequestDispatcher("demo.jsp");
rd.forward(request, response);
}
}

```

in Jsp code :

```

${requestScope.arun}
${requestScope.person.dog.name}

<%-- ${person.dog.name} --%>

```

Note :

- use `requestScope` implicit object to get request scoped attributes but not request parameters.
- If we want request properties then we should go for `pageContext` EL implicit object.

**param & paramValues :**

We can use these implicit objects to retrieve form parameter values

1. param	---->	request.getParameter()
2. paramValues	---->	request.getParameterValues()

`${param.x}`  // it prints the value of form parameter x, if the specified parameter is not available then we will get BlankSpace

If parameter associated with multiple values then we will get first value.

param :  
Map

String	String
uname	ashok

course	scwcd
place	hyd

paramValues :

Map

String	String[]
uname	{ashok, arun, akshay}
course	{scjp, scwcd}
place	{hyd}

`${paramValues.uname}`

// String[] will be returns on that Object class toString() is called.

<code> \${ paramValues.uname[0] }</code>	--->	ashok
<code> \${ paramValues.uname[100] }</code>	--->	BlankSpace
<code> \${ paramValues.user }</code>	--->	BlankSpace

Note : EL handles null and ArrayIndexOutOfBoundsException very nicely and prints BlankSpace.

form.html

```
<h3>Enter Your Details :</h3>
<form action="formParam.jsp" method="post">
 Enter Name :<input type="text" name="name" >

 Enter Mail :<input type="text" name="mail" >

 Enter Age:<input type="text" name="age" >

 Enter Food 1 :<input type="text" name="food" >

 Enter Food 2 :<input type="text" name="food" >

 <input type="submit" value="Submit">

</form>
```

formParam.jsp

```
<%@page isELIgnored="false"%>
```

```

<h3>Form Parameter Values :</h3>
Name : ${param.name}

Mail : ${param.mail}

Age : ${param.age}

Food : ${param.food}

Sex : ${param.sex}

ParamValues.name : ${paramValues.name}

ParamValues.name[0] : ${paramValues.name[0]}

ParamValues.name[1] : ${paramValues.name[1]}

ParamValues.food : ${paramValues.food}

ParamValues.food[0] : ${paramValues.food[0]}

ParamValues.food[1] : ${paramValues.food[1]}

ParamValues.food[100] : ${paramValues.food[100]}


```

### header & headerValues :

These are exactly similar to param and paramValues except that these are retrieving request headers.

1. header	---->	request.getHeader()
2. headerValues	---->	request.getHeaders()

If the specified request header is not available you will get BlankSpace as a output.

\${ header.x }	--->	BlankSpace
	--->	localhost:2020
	--->	java.lang.String@1234

header.jsp

```

<%@page isELIgnored="false"%>

<h3>Header Information :</h3>

Accept : ${header.accept}


```

```
Accept Values [0] : ${headerValues.accept[0]}

Accept Values [1] : ${headerValues.accept[1]}

Host : ${header.host}

Host [0] : ${headerValues.host[0]}

Some x is : ${header.x}

Accept Language : ${header['accept-language']}

```

### initParam :

By using initParam implicit object we can access ServletContext parameters but not servlet initialization parameters.

initParam ---> application.getInitParameter()

`${initParam.user}` // it prints the value associated with context parameter user.

If the specified parameter is not available then we will get BlankSpace.

`${initParam.x}` //BlankSpace

### initParam.jsp

```
<%@page isELIgnored="false" %>

<h3>Init Parameters are :</h3>

Init Param Name : ${initParam.name}

Init Param Mail : ${initParam.mail}

Init Param Age : ${initParam.age}

The X value is : ${initParam.x}

The Y value is : ${initParam.y}

```

### web.xml

```
<display-name>EL Implicit Objects</display-name>

<context-param>
 <param-name>name</param-name>
 <param-value>Ashok</param-value>
</context-param>
```

```
<context-param>
<param-name>mail</param-name>
<param-value>admin@jobs4times.com</param-value>
</context-param>
<context-param>
<param-name>age</param-name>
<param-value>2022</param-value>
</context-param>
<context-param>
<param-name>x</param-name>
<param-value>12345</param-value>
</context-param>
<context-param>
<param-name>y</param-name>
<param-value>54321</param-value>
</context-param>
```

To print the value of "userName" cookie ?

```
<%
Cookie[] c=request.getCookies();
for(int i=0;i<c.length;i++) {
 if(c[i].getName().equals("userName")) {
 out.println(c[i].getValue());
 }
}
%>
```

(OR)

```
<%@page isELIgnored="false" %>
Cookie value is : ${cookie.userName.value}

```

### cookie :

By using cookie implicit object we can retrieve cookies associated with the request

cookie ----> request.getCookies()

cookie :

Map

String	String
--------	--------

JSESSIONID	1234
c1	SCJP

**cookie.jsp**

```
<%@page isELIgnored="false" %>

<%
Cookie c1=new Cookie("c1","SCJP");
Cookie c2=new Cookie("c2","SCWCD");

response.addCookie(c1);
response.addCookie(c2);
%>

<h3>The Cookie Id is :</h3>
${cookie.JSESSIONID.name}

${cookie.JSESSIONID.value}

${cookie.JSESSIONID}

Cookie c1 is : ${cookie.c1}

Cookie c1 name is : ${cookie.c1.name}

Cookie c1 value is : ${cookie.c1.value}

Cookie c2 name is : ${cookie.c2.name}

Cookie c2 value is : ${cookie.c2.value}

```

**pageContext :**

- This is only one EL implicit object which matches with jsp implicit object.
- This is only one EL implicit object which is a non-map object by using this implicit object we can bring all jsp implicit objects into EL.

```
<%@page isELIgnored="false" %>

<% session.setAttribute("course","SCJP"); %>

${pageContext.request.method}

${pageContext.session.id}

${pageContext.request.Cookies[0]}

${course}

```

```
 ${request.method}

 ${session.id}

```

### handling errors :

- There is an exception and it was not caught then in this case the request will be forwarded along with exception to the error page URL, here the error page is specified `errorPage` attribute of page directive and by specified URL.
- This is so simple because the error page itself is identified by using the implicit object `exception`, more over in addition to exception object an error page also has the following request attributes available.
  - `javax.servlet.error.status-code`
  - `javax.servlet.error.request-URI`
  - `javax.servlet.error.servlet-name`
  - When ever we use the `pageContext.getErrorData()` method, it is possible to get an instance of `javax.servlet.jsp.Error-Data` class
  - This will provide to find a simple way to access the above attributes. (`status-code`, `request-URI`, `servlet-name`)

### index.jsp

```
<%@page isELIgnored="false" %>

<%@page errorPage="error.jsp" %>
<%
out.println(10/0);
%>
```

### error.jsp

```
<%@page isErrorPage="false"%>

The Raised Exception is : ${pageContext.exception.message}

The Status code is : ${pageContext.errorData.statusCode}

The Request URI is : ${pageContext.errorData.requestURI}

The Servlet Name is : ${pageContext.errorData.servletName}

All EL implicit objects are map object type except pageContext.
```

### EL Operators :

EL contains it's own specific operators the following is list of all possible operators

1. property access operator ( `.` )
2. collection access operator `[]`
3. Arithmetic operators ( `+, -, *, /, %` )
4. Relational operators ( `lt, gt, eq, ne` ) OR ( `<, >, <=, >=, ==, !=` )
5. Logical operators ( `and, or, not` )

6. conditional operator (?:)
7. empty operator (empty)

#### property access operator :

```

${leftVariable.rightVariable}
// map ----key
// bean ---property
// left attribute stored in some scope

```

- If the expression has a variable followed by . (dot) , the left variable must be a map or bean.
- The thing to the right of the . (dot) must be a key or bean property.

The thing on the right must follow normal java naming rules for the identifies.

```

${person.name}

// name must starts with letter or underscore or $
// after the first character we can include the numbers
// can't be a java keyword
${initParam.age}

//map ---- key
${person.dog}

//bean ---- property
${person.1}

//javax.el.ELEception:ErrorParsing

```

#### collection access operator :

```

${leftVariable.rightVariable}
// Map -----key //quotes are mandatory
// Bean -----property //quotes are mandatory

// List ----- index //quotes are optional
// Array ----- index //quotes are optional

```

When we use the . operator the thing on the left can be only a Map or Bean and thing on the right must follow java naming rules for the identifies but with collection access operator, the thing on the left can also be a List or Array that also meaning the thing on the right can be a number or any thing that resolves to a number or an identifier that doesn't java naming rules.

**Ex :**

```

${musicMap[MusicTypes[0]]}

```

(OR)

```
 ${musicMap[MusicTypes["0"]]}
```

But you can't

```
 ${musicMap[MusicTypes["zero"]]}
 // It can't convert into int
```

**Map Example :**

```
 ${initParam['mail']}

 ${initParam['age']}

 ${initParam['name']}

 ${initParam[age]}
 <%-- blank space --%>
```

**Note :** If we are not using code symbol then it is treated as attribute if the specified attribute is not available then we will get blank space.

**Bean in servlet code :**

```
 package foo;

 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;

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

 foo.Person p = new foo.Person();
 p.setName("Akshay");

 foo.Dog d = new Dog();
 d.setName("puppy");

 p.setDog(d);

 request.setAttribute("person", p);
```

```
 RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
 rd.forward(request, response);
 }

}
```

### view.jsp

```
<%@ page isELIgnored="false"%>
${person['dog']['name']}
${person[dog][name]} //blankspace
```

#### *List Example :*

Servlet code :

```
package foo;

import java.io.IOException;
import java.util.ArrayList;

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 ELServletDemo extends HttpServlet {

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

 java.util.ArrayList list=new ArrayList();
 list.add("Ashok");
 list.add("Arun");
 list.add("Akshay");
 list.add("Saicharan");

 request.setAttribute("x", list);
 request.setAttribute("index", "2");

 RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
 rd.forward(request, response);
 }
}
```

```
}
```

```
}
```

```
view.jsp
```

```
<%@ page isELIgnored="false"%>
The first name is : ${x[0]}

The second name is : ${x[1]}

```

```
List Names are : ${x}
```

**Array Example :**

Servlet code :

```
package foo;

import java.io.IOException;
import java.util.HashMap;

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 ELServletDemo extends HttpServlet {

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

 java.util.Map musicMap = new HashMap();
 musicMap.put("melody", "song1");
 musicMap.put("DJ", "song2");
 musicMap.put("fastBeat", "song3");
 musicMap.put("slowBeat", "song4");

 request.setAttribute("musicMap", musicMap);
 String[] musicTypes = {"melody", "DJ", "fastBeat", "slowBeat"};
 request.setAttribute("musicType", musicTypes);

 RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
 rd.forward(request, response);
 }

}
```

```
view.jsp
```

```
<%@ page isELIgnored="false"%>
```

```
 ${ musicMap[musicType[1]] }
```

```
 ${ musicMap["DJ"] }
```

note : first inner most brackets will be evaluated.

### Ex 2 :

```
<%@ page isELIgnored="false"%>
<%
 String[] s={"A","B","C","D"};
 pageContext.setAttribute("s",s);
%>
${s[0]}

${s[1]}

${s[100]}

```

### Example :

#### Person.java

```
package foo;

public class Person {
 private String name;
 private Dog dog;

 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 public Dog getDog() {
 return dog;
 }
 public void setDog(Dog dog) {
 this.dog = dog;
 }
}
```

#### Dog.java

```
package foo;

public class Dog {
 private String name;
```

```

private Toy[] toys;

public String getName() {
 return name;
}

public void setName(String name) {
 this.name = name;
}

public Toy[] getToys() {
 return toys;
}

public void setToys(Toy[] toys) {
 this.toys = toys;
}

}

```

Toy.java

```

package foo;

public class Toy {
 private String name;

 public String getName() {
 return name;
 }

 public void setName(String name) {
 this.name = name;
 }
}

```

In Servlet code :

```

package foo;

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;

public class ELServletDemo extends HttpServlet {

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

 foo.Person p = new Person();

```

```

 p.setName("SAI");

 foo.Dog d = new Dog();
 d.setName("puppy");

 foo.Toy t = new Toy();
 t.setName("bear1");
 foo.Toy t1 = new Toy();
 t1.setName("bear2");
 foo.Toy t2 = new Toy();
 t2.setName("bear3");

 p.setDog(d);
 d.setToys(new foo.Toy[]{t,t1,t2});

 request.setAttribute("person", p);

 RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
 rd.forward(request, response);
}

}

```

**view.jsp**

```

<%@ page isELIgnored="false"%>

${person.name} 's dog ${person.dog.name} , toys are
${person.dog.toys[0].name} , ${person.dog.toys[1].name} , and
${person.dog.toys[2].name}
SAI 's dog puppy , toys are bear1 , bear2 , and bear3

```

**Note :** Where ever property access operator is required there we can use collection access operator but where ever collection access operator is required may not be use property access operator.

**Arithmetic operators : (+, -, /, \*, %)**

EL doesn't support operator overloading and + operator always meant for Arithmetic addition but not for String Concatenation.

+ operator :

```
<%@ page isELIgnored="false"%>
```

```

2+3= ${2+3}
"2"+3= ${"2"+3}
"2"+'3'= ${"2"+'3'}
abc+'3'= ${abc+'3'}

```

```
"abc"+'3'= ${"abc"+'3'}
// java.lang.NullPointerException:For input string: "abc"

"""+3= ${"""+'3'}
// java.lang.NullPointerException:For input string: ""

null+'3'= ${null+'3'}
```

**- operator :**

```
<%@ page isELIgnored="false"%>
```

```
2-3= ${2-3}
"2"-3= ${"2"-3}
"2"-'3'= ${"2"-3'}
abc-'3'= ${abc-'3'}
```

```
"abc"-3= ${"abc"-3'}
// java.lang.NullPointerException:For input string: "abc"

"""-3= ${"""-3'}
// java.lang.NullPointerException:For input string: " "

null-'3'= ${null-'3'}
```

**\* operator, / operator :** all rules are exactly similar to + operator except that division operator always follows floating point Arithmetic.

**syntax :**

```
 ${a/b} or ${a div b}
```

**Ex :**

```
 ${10/2} = 5.0
```

```
 ${10/0} = Infinity
```

```
 ${0/0} = NaN
```

**% operator :**

All rules are exactly similar to + operator here both integral and floating Arithmetics are possible.

**syntax :**

```
 ${a%b} or ${a mod b}
```

**Ex:**

```
 ${10%3} // 1
 ${10%0} //java.lang.ArithmetricException: / by zero
 ${10.0%0} //NaN
 ${"""/'3'}
 //java.lang.NumberFormatException: empty String
```

**Note :** In Arithmetic operator null evaluates Zero, the presidency of Arithmetic operators \*, /, %, +, -.

**In side Servlet code :**

```
package foo;

import java.io.IOException;
import java.util.ArrayList;

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 ELServletDemo extends HttpServlet {

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

 java.util.ArrayList numList = new ArrayList();
 numList.add("1");
 numList.add("2");
 numList.add("3");
 numList.add("4");
 request.setAttribute("numbers", numList);

 String[] s = {"melody", "DJ", "fast", "slow"};
 request.setAttribute("musicList", s);

 RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
 rd.forward(request, response);
 }
}
```

**view.jsp**

```
<%@ page isELIgnored="false"%>

${musicList[numbers[0]]}

${musicList[numbers[1]+2]}

${musicList[numbers[numbers[0]]+1]}

${musicList[numbers["2"]]}

```

**Relational operators :**

<	or	lt
<=	or	le
>	or	gt
>=	or	ge
==	or	eq
!=	or	ne

**Ex :**

```
<%@ page isELIgnored="false"%>
```

**Relational operators :**

```
 ${3 < 4} //true
 ${3 > 4} //false
 ${3 >= 4} //false
 ${3 <= 4} //true
 ${"abc" == "abc"} //true
 ${2!=1} //true
 ${sai == sai} //true
 ${3 < sai} //false
 ${String < String} //false
```

In Relational Operators , the unknown variable evaluates as a false.

**Logical operators :**

&&	or	and
	or	or

**Ex :**

```
<%@ page isELIgnored="false"%>

${true and true} //true
${false || false} //false
${!true} //false
${ashok && ashok} //false
${not ashok} //true
```

**Note :** In logical operators unknown variable evaluates false.

### conditional operator : (?:)

```
<%@ page isELIgnored="false"%>

${(3 < 4)?"yes":"no"} //yes
${(true == false)?"Right":"wrong"} //wrong
${(2 < 3)?"correct":"wrong"} //correct
```

In the case of conditional operator unknown variable treats a variable stored in some scope.

### empty operator :

**syntax :**

`${empty obj}`

returns true iff object doesn't exists/ an empty array/ object is an empty collection/ object as a empty String in all other cases it returns false.

**Ex :**

```
<%@ page isELIgnored="false"%>
<%@page import="java.util.ArrayList"%>

<%ArrayList l=new ArrayList(); %>

${empty test} //true
${empty "durga"} //false
${empty null} //true
${empty " "} //false
${empty l} //true
```

In Servlet code :

```
package foo;
```

```

import java.io.IOException;
import java.util.ArrayList;

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 ELServletDemo extends HttpServlet {

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

 int num = 2;
 request.setAttribute("num", num);

 Integer i = new Integer(3);
 request.setAttribute("integer", i);

 java.util.ArrayList list = new ArrayList();
 list.add("true");
 list.add("false");
 list.add("5");
 list.add("6");
 request.setAttribute("list", list);

 RequestDispatcher rd = request.getRequestDispatcher("jspDemo.jsp");
 rd.forward(request, response);
 }
}

```

**view.jsp**

```

<%@ page isELIgnored="false"%>

<jsp:useBean id="myDog" class="foo.Dog">
<jsp:getProperty name="myDog" property="name" />
</jsp:useBean>

${myDog.name and true} //false

${requestScope["integer"] ne 4 and 6 le num or false} //false
//EL executes left to right

```

### EL operator presidency :

1. unary (! , empty )
2. Arithmetic ( \*, /, %, +, - )
3. Relational ( <, >, <=, >=, ==, != )
4. Logical ( &&, || )
5. conditional operator ( ?: )

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

# **JAVA MEANS DURGASOFT**

## **INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

### EL reserved words :

It	true	instanceof	div
gt	false	empty	mod
le	and	null	
ge	or		
ne	not		
eq			

### EL Vs null :

1. EL behavior very nicely with null, in Arithmetic operator null is treated as a zero.
2. With String evaluation null is treated as empty String (" ")
3. With logical operators null is treated as false
4. In the case of empty operator null is treated as true
5. In the case of relational operator null is treated as false

Assume that there is no attribute named with "foo" but there is an attribute named with "bar" but bar doesn't have any property or key name "foo"

- \${foo} // BlankSpace
- \${foo[bar]} // BlankSpace
- \${bar[foo]} // BlankSpace
- \${foo.bar} // BlankSpace
- \${7/foo} // Infinity
- \${7%foo} // java.lang.ArithmeticException
- \${true and false} // false
- \${true or false} // true

- \${not foo} // true

### EL Functions :

1. The main objective of EL is separate code from the jsp
2. If we are having any business functionality separate in to a java class and we can invoke this functionality through EL syntax.
3. The page designer has to know only function name, tld file uri to get its functionality hence EL is alternative to custom tags.

Designing EL function application contains the following stages :

1. Writing a java class with required functionality.
2. Writing a TLD file to map a java class to Jsp
3. Write a taglib directive to make business functionality available to Jsp.
4. Write EL function call.

### Writing a java class :

1. Any java class can act as a repository for EL functions , the only requirement of the method that act as a functions it should be public and static.
2. The method can take parameter also, there are no restriction on return type even void return type is also allowed.

### Writing a TLD file : (tag library descriptor file)

TLD file is a xml file which provides mapping between java class (where functionality is available) and jsp (where functionality is required)

We can configure EL function by using function tag in TLD file , it contains the following 4 steps :

1. description : It is an optional tag
2. name : By means of this name only we can call EL functionality in jsp ( this name need not be same as original method name, it is a logical name)  
There is no <function-name> tag to configure EL function .
3. function-class : The fully qualified name of java class where EL function defined.
4. function-signature : The signature of EL function which is defined in java class , here we have to specified return type, method name and argument list.

### Writing a taglib directive :

We have to declare taglib directive in the jsp to provide location of the TLD file

```
<%@taglib prefix="mime" uri="www.jobs4times.com" %>
```

### Writing EL function call :

```
 ${mime.getBalance(435678)}
```

demo.jsp

```
<%@ page isELIgnored="false"%>
<%@taglib prefix="mime" uri="DiceFunctions"%>
```

Your 3 - Digit Lucky Number :

```
 ${mime:rollIt()}

 ${mime:rollIt()}

 ${mime:rollIt()}

```

DiceRoller.java

```
 package foo;
```

```
 public class DiceRoller {
```

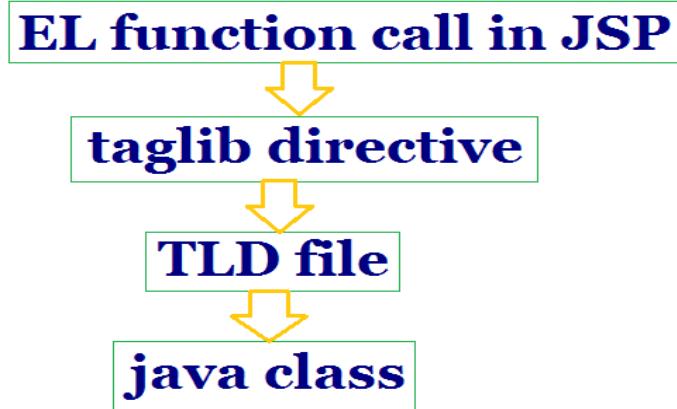
```
 public DiceRoller() {
 System.out.println("DiceRoller");
 }
 public static int rollDice() {
 return (int) (Math.random() *6+1);
 }
 }
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
 <taglib version="2.1" >
 <tlib-version>1.0</tlib-version>
 <uri>DiceFunctions</uri>
 <functions>
 <name>rollIt</name>
 <function-class>foo.DiceRoller</function-class>
 <function-signature>int rollDice()</function-signature>
 </functions>
 </taglib>
```



**Work Flow : (in EL function application)**

1. When ever jsp engine encounters EL function call in jsp first it identifies the prefix and check for corresponding taglib directive with matched prefix.
2. From the taglib directive jsp engine identifies uri and checks for corresponding TLD file with matched uri.
3. From the TLD file jsp engine identifies the corresponding java class and method.
4. Jsp engine execute that method and returns its result to jsp.

**Note :**

- The common thing between EL function call and taglib directive is "prefix".
- The common thing between taglib directive and TLD file is "uri".
- The common thing between TLD file and java class is "class name" and "method" .

Write a program to invoke Math class sqrt() as a EL function call with name m1

```
public static double sqrt(double) //method signature
```

index.jsp

```
<%@ page isELIgnored="false"%>
<%@taglib prefix="mime" uri="MathFunctions"%>
```

The square root of 4 is : \${mime.m1(4)}

The square root of 5 is : \${mime.m1(5)}

mathFunction.tld

```
<taglib version="2.1" >
<tlib-version>1.0</tlib-version>
<uri>MathFunctions</uri>
<functions>
<name>m1</name>
<function-class>java.lang.Math</function-class>
```

```
<function-signature>double sqrt(double)</function-signature>
</functions>
</taglib>
```

**www.durgasoftonlinetraining.com**



**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428  
USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

# **JAVA MEANS DURGASOFT**

**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED

**DURGA**  
SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

**LEARN FROM EXPERTS ...**

**COMPLETE JAVA**  
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

**COMPLETE .NET**  
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

**TESTING TOOLS**  
MANUAL + SELENIUM

**ORACLE D2K**

**MSBI SHARE POINT**

**HADOOP ANDROID**

**C, C++, DS, UNIX**

**CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED  
**DURGA** # 202, 2nd Floor, HUDA Maitrivanam,  
Software Solutions Ameerpet, Hyd. Ph: 040-64512786,  
9246212143, 8096969696

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA With SCWCD / OCWCD JSP Material**

**4. Building JSP Pages using Tag Libraries(JSTL)**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

**DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

**173**

DURGASOFT, # 202, 2<sup>nd</sup> Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,  
040 - 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | [www.durgasoft.com](http://www.durgasoft.com)

**Building JSP Pages Using Java Standard Tag Library (JSTL)****Agenda :**

## ❖ Core Library (14 tags)

## 1) General purpose tags :

- <c:out>
- <c:set>
- <c:remove>
- <c:catch>
- Summary of General Purpose tags

## 2) Conditional tags :

- <c:if>
- <c:when>
- <c:choose>
- <c:otherwise>
- Summary of Conditional tags

## 3) Iteration tags :

- <c:forEach>
- <c:forTokens>
- Summary of Iteration Tags

## 4) Re-directional tags : (URL related tags)

- <c:url>
- <c:redirect>
- <c:import>
- <c:param>
- Summary of url tags

## ❖ Sql Tag Library

## ❖ XML Tag Library

## ❖ Formatting Tag Library

## ❖ Functional Tag Library

**Introduction :**

1. Sun people encapsulated the core functionality which is common to many web application in the form of JSTL, programmer can use this predefined library without writing on his own.
2. The main objective of EL is to eliminate java code from the jsp but it fails to replace java code complete elimination.
3. Which process some functionality , we can resolve this problem by using JSTL hence the main objective of JSTL is to remove java code from the Jsp.

Entire JSTL divided into 5 parts :

1. Core Library (14 tags)
  - It defines some standard actions to perform general programming like implementing condition and iterational statements.
  - It can also perform jsp fundamental tasks such as setting attributes, writing output to the jsp page and redirect in the request to other pages and etc.,
2. Sql Tag Library : It defines several standard actions it can be used for DataBase operations.
3. XML Tag Library : It defines several standard actions useful for writing and formatting XML data.
4. Formatting Tag Library : It defines several standard actions which are useful for I18N purpose.
5. Functional Tag Library : It defines several standard actions which are used for manipulating for Collection and Objects.

**Core Library :**

JSTL core library divided into the following 4 parts

- 1) General purpose tags :
  - <c:out>
  - <c:set>
  - <c:remove>
  - <c:catch>
- 2) Conditional tags :
  - <c:if>
  - <c:when>
  - <c:choose>
  - <c:otherwise>
- 3) Iterational tags :
  - <c:forEach>
  - <c:forTokens>
- 4) Re-directional tags : (URL related tags)
  - <c:url>
  - <c:redirect>

- <c:import>
- <c:param>

### Installing JSTL :

By default JSTL functionality is not available to the jsp, we can provide JSTL functionality by placing the following jar files in web-application lib folder.

1. jstl.jar : defines several classes and interfaces provided by Sun.
2. standard.jar : provide library implementation classes by vendor.

**Note :** The above 2 jar files we have to download from net and place it either in web-application lib folder or at server level lib folder( i.e., ex : tomcat/lib )

To make core library available to the jsp , we have to declare taglib directive as follows

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

OR

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

### **General Purpose tags :**

<c:out>

We can use this tag for writing template text and expressions to the jsp

```
<c:out>
```

c --- prefix  
out ---- tagname

form 1 :

```
<c:out value="ashok" />
```

It prints template text ashok to the Jsp

```
<c:out value="${param.uname}" />
// runtime expressions
```

form 2 :

```
<c:out value="${param.uname}" default="ashok" />
```

- If the main value is not possible or if it is not then we can provide default value by using default attribute.

- If the specified request parameter 'uname' is not available then returns a null , in that case only default will get the chance.

form 3 :

```
<c:out value="${result}" escapeXml="false" />
```

If the result contains xml data , if we want to xml data then take `escapeXml="false"` , if we don't want to xml data then take `escapeXml="true"` , in that case the result value considered template text.

**Ex 1 :**

```
<%@page isELIgnored="false" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:out value="ashok" />
The Entered name :
<c:out value="${param.uname}" default="charan" />
```

**Ex 2 :**

```
<%@page isELIgnored="false" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
 <head>
 <title>The attribute escapeXml in c:out </title>
 </head>
 <body>
 <c:set var="test" scope="session">
 We are going to use the attribute escapeXml of c:out to show the difference
 <table border="5">
 <tr><td bgcolor="green">Arun </td>
 <td bgcolor="yellow">Akshay </td>
 <td bgcolor="red">SaiChran </td></tr>
 </table>
 </c:set>

 <h1>out with escapeXml="false"</h1>
 <c:out value="${test}" escapeXml="false" />

 <h1>out with escapeXml="true"</h1>
 <c:out value="${test}" escapeXml="true" />

 </body>
</html>
```

**<c:out> defines following 3 attributes :**

- value** : It is a mandatory attribute to provide required value , it can be String literal or runtime expression.
- default** : It is an optional attribute and it is for providing default value, Jsp engine consider it's value if and only if the value attribute evaluates null.

3. **escapeXml** : It is an optional attribute the default value is true.  
 "true" if the tag should escape special xml characters,  
 "false" it process that xml data.

### <c:set>

The <jsp:setProperty> tag can do only one thing set property of bean but

- 1) if we want to set a value in map (or)
- 2) if we want to make one entry in map (or)
- 3) if we simply want to create new request scope attribute.

We can get all these things by using <c:set>

<c:set> comes in 2 flavors var and target

1. **var** : The var version is for setting attribute variable.
2. **target** : The target version is for setting bean properties or map values.

Each of the 2 flavors comes in 2 variations with or without body.

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

<c:set var="x" value="13"/>
<c:set var="y" value="23"/>
<c:set var="result" scope="application" >
${x*y}
</c:set>
```

The result of the 2 numbers is :

<c:out value="\${result}"/>

**Ex :**

```
<c:set var="fedo" value="${person.dog}" />
// value -- need not be a String
```

**Ex :**

```
<c:set var="x" scope="session">
ashok, arun
</c:set>
```

### setting a target property or value with <c:set> :

Without body :

<c:set target="\${map}" property="raja" value="SCWCD"/>

bean :

```
<c:set target="${person}" property="name">
${person.name}
</c:set>
```

**Example 1 :****In Servlet Code :**

```
package com.jstl;

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;

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

 foo.Person person = new foo.Person();
 foo.Dog dog = new foo.Dog();

 dog.setName("spike");
 person.setDog(dog);

 System.out.println(person);

 request.setAttribute("person", person);

 RequestDispatcher rd = request.getRequestDispatcher("pages/demo.jsp");
 rd.forward(request, response);
 }

}

demo.jsp

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

<c:set target="${person}" property="name" value="Ashok"/>
<c:out value="${person.name}" />
```

**Example 2 :**

```
package com.jstl;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

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 ServletDemo extends HttpServlet {

 public void service(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 Map map = new HashMap();
 request.setAttribute("map", map);
 RequestDispatcher rd = request.getRequestDispatcher("pages/demo.jsp");
 rd.forward(request, response);
 }
}
```

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

<c:set target="${map}" property="ashok" value="SCWCD"/>
<c:out value="${map}"/>

<c:out value="${map.ashok}"/>
```

```
web.xml

<web-app>
<servlet>
 <servlet-name>ServletPerson</servlet-name>
 <servlet-class>com.jstl.ServletDemo</servlet-class>
</servlet>
```

```
<servlet-mapping>
 <servlet-name>ServletPerson</servlet-name>
 <url-pattern>/fs</url-pattern>
</servlet-mapping>
```

```
</web-app>
```

```
output :
```

```
{ashok=SCWCD}
```

**SCWCD**

<http://localhost:8080/jsp/fs>

**<c:set> conclusions :**

- We can never have both var, target attributes <c:set> it will gives unpredictable results.
- Scope is an optional attribute, default scope is page.
- If the "value" is null, the attribute named by var will be removed.

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

<c:set var="x" value="${param.user}" />
<c:out value="${x}" />
//Assume there is no request parameter user
output : blankspace
```

- If the attribute var doesn't exist <c:set> will be created but if value is not null.
- If the target expression is null then container throws an Exception.
- If the target expression is not a map/bean then container throws an Exception.
- If the target expression is a bean but the bean doesn't have a property matches with property attribute then container throws an exception saying Invalid property in <set>.

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

<c:set target="${person}" property="name" value="Ask" scope="session" />
<c:out value="${person.name}" />
output :
```

**org.apache.jasper.JasperException:**  
Illegal scope attribute without var in "c:set" tag.

**note :** In <c:set> tag all attributes are optional , when ever we are taking scope attribute compulsory we can take var attribute otherwise we will get exception.

**<c:remove>**

We can use this tag to remove attribute in the specified scope this tag contains the following 2 attributes.

1. var : represents name of the attributes.
2. scope : represents the scope in which attribute present.

**Ex :**

```
<c:remove var="x" />
```

If the scope is not specified then JSP engine will search page scope for the specified attribute, if it is available then JSP engine will removes the attribute, If the specified attribute is not available it will search in request scope followed by session, application.

Note : The `<c:remove>` compulsory should be a var attribute but not expression.

```
<c:remove var="x"/> //valid
<c:remove var="${x}"> //invalid
```

Note 2 : In `<c:remove>` tag var attribute is the mandatory attribute and scope attribute is a optional attribute.

**Ex :** remove.jsp

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

<c:set var="x" value="10"/>
<c:set var="y" value="20"/>
<c:set var="result" value="${x*y}" scope="application"/>
Before removal of result : <c:out value="${result}" /> //200

<c:remove var="x"/>
<c:remove var="y"/>
removal of x & y : <c:out value="${result}" /> //200

<c:remove var="result"/>
After removal of result is :
<c:out value="${result}" default="8888"/> //8888
Output :
```

```
Before removal of result : 200 //200
removal of x & y : 200 //200
After removal of result is : 8888 //8888
```

### `<c:catch>`

This can be used to catch and suppress that exception, so that the result of the code will be executed normally

We have to place risky code as a body of `<c:catch>` tag.

**Syntax :**

```
<c:catch>
Risky Code
```

```
</c:catch>
```

- If an exception raised an risky code when this tag suppress that expression and rest of the code will be executed normally.
- We can hold the raised exception object by using var attribute, which is page scoped attribute.

Note : in <c:catch> var attribute is optional.

Ex 1 :

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

<c:catch var="e"> //page scope we can't access outside
<% int age=Integer.parseInt("ten"); %>
The Raised Exception : ${e}
</c:catch>
<c:if test="${e!=null}">
<h1>OOPS! -- Exception Raised </h1> ${e}
</c:if>
output :
```

```
//page scope we can't access outside
OOPS! -- Exception Raised
```

java.lang.NumberFormatException: For input string: "ten"

Ex 2 :

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

The UserName is : ${param.uname}
<c:catch var="e">
<%int age=Integer.parseInt(request.getParameter("age"));%>
The Age is : ${param.age}
</c:catch>

<c:if test="${e!=null}">
<h1>OOPS!--Exception Raised</h1> <h3>${e}</h3>
</c:if>
```

```
The Height is : ${param.height}
output :
```

<http://localhost:8081/jstl/WebRoot/pages/jstl.jsp?uname=Ashok&age=ten&height=5.5>

```
The UserName is : Ashok
OOPS!--Exception Raised
```

`java.lang.NumberFormatException: For input string: "ten"`

The Height is : 5.5

Ex 3 :

```
<c:catch var="e">
${Person.age}
</c:catch>
```

Raised Exception is : \${e}

age is not a property of Person hence it raises `PropertyNotFoundException` but `<c:catch>` handles that exception and continue rest of the JSP normally.

#### *Summary of General Purpose tags :*

Tag	Description	Attributes
<code>&lt;c:out&gt;</code>	For writing template text and expression to the JSP page.	<code>value, default, escapeXml</code> <b>value is mandatory attribute</b>
<code>&lt;c:set&gt;</code>	To set some attribute in some scope and to set bean property and add to entries in the Map.	<code>var, target, value, scope, property</code>
<code>&lt;c:remove&gt;</code>	To remove an attribute in the specified scope, if we are not mentioning any scope page followed request, session, application.	<code>var, scope</code> <b>var is mandatory attribute</b>
<code>&lt;c:catch&gt;</code>	For suppress an Exception and continue rest of the JSP normally.	<code>var</code>

#### Conditional Tags :

##### <c:if>

If we can use this tag to implement core java if statement , there are 2 forms are `<c:if>` available.

##### Without body:

```
<c:if test="testcondition" var="x" scope="session"/>
```

In this case test condition will be evaluated and result store into var x , If the rest of the Jsp page where ever the same test condition is required , we can use its directly without re-evaluated once again.

- In this case test attribute is mandatory.
- var, scope attributes are optional.
- But when ever the scope attribute is specified compulsory we should take var attribute.

#### With body :

```
<c:if test="testcondition" var="x" scope="session">

</c:if>
```

The test condition is true then the body will be executed otherwise without executing the body , the rest of the JSP will be executed.

- In this case also we can store test results into var variable.
- scope, var attributes are optional.

Ex :

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

<c:if test="${uname eq 'Ashok'}">
<jsp:forward page='demo.jsp'/>
</c:if>
```

if.jsp

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

<h1>If Conditional Tag with Body</h1>
<c:set var="x" value="10" scope="request"/>
<c:if test="${x eq 10}" var="y" scope="session">
 x value is : ${x}

 The result is : ${y}
</c:if>
```

output :

If Conditional Tag with Body

x value is : 10  
The result is : true

<c:choose>, <c:when>, <c:otherwise> we can use these tags for implements if-else , switchstatements.

**implementing if-else :**

JSTL doesn't contain any tag for else , we can implement if-else statement by using the above tags

```
<c:choose>
<c:when test="testcondition">
 //Action 1 (if)
</c:when>

<c:otherwise>
 //Action 2 (else)
</c:otherwise>
</c:choose>
```

If test condition is true Action 1 will be executed else Action 2 will be executed.

**implementing switch statement :**

```
<c:choose>
<c:when test="testCondition1">
 //Action 1
</c:when>
<c:when test="testCondition2">
 //Action 2
</c:when>

<c:otherwise>
 //default Action
</c:otherwise>
</c:choose>
```

- <c:choose> should compulsory contains atleast one <c:when> , but <c:otherwise> is optional.
- Every <c:when> implicitly contains break stastement hence there is no chance fall-through inside switch.
- We have to take <c:otherwise> as a last statement only.
- <c:choose> and <c:otherwise> won't take any attribute but <c:when> tag can contains only one mandatory attribute i.e., test.

**Ex :**

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

<html>
<body>
<form>
Select the Number
<select name="day">
<option>1</option>
```

```
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
<option>6</option>
<option>7</option>
</select>

<input type="submit"/>
</form>

<c:set var="day" value="${param.day}"/>
Today is :
<c:choose>
<c:when test="${day eq 1}">
<c:out value="Sunday"/>
</c:when>
<c:when test="${day eq 2}">
<c:out value="Monday"/>
</c:when>
<c:when test="${day eq 3}">
<c:out value="Tuesday"/>
</c:when>
<c:when test="${day eq 4}">
<c:out value="Wednesday"/>
</c:when>
<c:when test="${day eq 5}">
<c:out value="Thursday"/>
</c:when>
<c:when test="${day eq 6}">
<c:out value="Friday"/>
</c:when>
<c:when test="${day eq 7}">
<c:out value="Saturday"/>
</c:when>

<c:otherwise>
<c:out value="Select the values between 1 to 7"/>
</c:otherwise>
</c:choose>

</body>
</html>
```

**Summary of Conditional tags :**

Tag	Description	Attributes
<c:if>	To implement core java if statement	test, var, scope <b>test</b> attribute is mandatory
<c:choose> <c:when> <c:otherwise>	To implement if-else and switch statements	<c:choose> and <c:otherwise> won't take any attributes, <c:when> take one attribute i.e., <b>test</b> .

**Iteration Tags :****<c:forEach> tag :**

<c:forEach> to implement general purpose **for loop**.

**form 1 :**

```
<c:forEach begin="0" end="11" step="1">
Learning Jstl (11 times)
</c:forEach>
```

Here

- This loop internally maintain one counter variable , which is incremented by step attribute value.
- The default value for the step attribute is "1" , and it is optional attribute.

**form 2 : <c:forEach> with var attribute**

```
<c:forEach begin="0" end="11" var="count" step="2">
${count}
</c:forEach>
```

output :

0 2 4 6 8 10

**form 3 : </c:forEach> with items attribute**

```
<c:forEach items="${Array/Collections}" var="obj">
.....
</c:forEach>
```

- **items** attribute should contains either Collection Obj (OR) Arrays.
- This action will integrate over each item in the collection until all the elements.
- We can represent current collection obj by using var attribute.

Types of items attribute	Types of var attribute
primitive array Ex : int[]	Corresponding wrapper class Integer

Collection	java.lang.Collection
Map	Map.entry
Object Array	Corresponding object class type
Student[]	Student
List of String separated by ","	String

**Ex :**

```
<%@page isELIgnored="false" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%
 String[] s = {"A","B","C","D"};
 session.setAttribute("s", s);
%>

<c:forEach items="${s}" var="obj">
 The Current object is : ${obj}

</c:forEach>
```

**output :**

The Current object is : A  
The Current object is : B  
The Current object is : C  
The Current object is : D

**header.jsp**

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

<table>
<tr><th>Header name</th><th>Header value</th></tr>
<tr><td><c:forEach items="${header}" var="x"></td></tr>
 <tr><td>${x.key}</td><td>${x.value}</td></tr>
<tr><td></c:forEach></td></tr>
</table>
```

**cookie.jsp**

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

<%
 Cookie c1 = new Cookie("uname", "Agastya");
 Cookie c2 = new Cookie("mail", "info@jobs4times.com");
 Cookie c3 = new Cookie("mobile", "9822334455");
```

```

Cookie c4 = new Cookie("address", "IND");
response.addCookie(c1);
response.addCookie(c2);
response.addCookie(c3);
response.addCookie(c4);

%>
<c:forEach items = "${cookie}" var="x">
 <h2>${x.value.name} ---- ${x.value.value}</h2>
</c:forEach>

output :
address ---- IND
uname ---- Agastya
mail ---- info@jobs4times.com
mobile ---- 9822334455
JSESSIONID ---- 62a82c98e954f45a4f5967a745ff

```

**Write a program to print all the session scoped attributes (attribute names and attribute values)**

in Servlet code :

```

package info;
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 ServletDemo extends HttpServlet {
 private static final long serialVersionUID = 1L;

 protected void processRequest(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 HttpSession session = request.getSession();
 session.setAttribute("Ashok", "SCJP");
 session.setAttribute("Arun", "SCWCD");

 RequestDispatcher rd = request.getRequestDispatcher("myJsp.jsp");
 rd.forward(request, response);
 }

 @Override
 protected void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 processRequest(request, response);
 }
}

```

```

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException {
 processRequest(request, response);
}
}

```

**myJsp.jsp**

```

<%@page isELIgnored="false" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:forEach items="${sessionScope}" var="obj">
<h2>${obj.key} ---- ${obj.value}</h2>
</c:forEach>

```

**output :**  
Arun ---- SCWCD  
Ashok ---- SCJP

#### Example :

**In Servlet Code :**

```

String[] movies1={"A", "B", "C"};
String[] movies2={"MovieA", "MovieB", "MovieC"};

java.util.ArrayList list=new java.util.ArrayList();
list.add(movies1);
list.add(movies2);

request.setAttribute("moviesList", list);

RequestDispatcher rd=request.getRequestDispatcher("myJsp.jsp");
rd.forward(request, response);

```

**myJsp.jsp**

```

<c:forEach items="${moviesList}" var="listElements">
<c:forEach items="${listElements}" var="movie">
${movie}
</c:forEach>
</c:forEach>

```

**output :**  
A B C MovieA MovieB MovieC

#### form 4 : <c:forEach> with varStatus attribute

- This attribute discuss status of the iteration like current iteration number is 1<sup>st</sup> iteration or not.
- This attribute is the type of javax.servlet.jsp.jstl.core.LoopTagStatus
- This class contains several methods, which are useful during iterations.



<code>public Object getCurrent()</code>	it returns the current item in the iteration.
<code>public int getIndex()</code>	returns current index
<code>public int getCount()</code>	returns the no. of iterations that have already perform including current iteration.
<code>public boolean isFirst()</code>	returns information about whether the current iteration is first , then it returns "true" else returns "false"
<code>public boolean isLast()</code>	returns information about whether the current iteration is last , then it returns "true"
<code>public Integer getBegin()</code>	returns the value of begin attribute for the associate tag, (OR) null if no begin attribute is specified.
<code>public Integer getEnd()</code>	returns the value of end attribute for the associate tag, (OR) null if no end attribute is specified.
<code>public Integer getStep()</code>	returns the value of step attribute for the associate tag, (OR) null if no step attribute is specified. (i.e., there is no default value)

myJsp.jsp

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

<c:forEach items="Sai,Shiva,Vishnu" varStatus="status">
The current Item : ${status.current}

The index : ${status.index}

The count : ${status.count}

Is it first Item ? ${status.first}

Is it last Item ? ${status.last}

The begin Item : ${status.begin}

```

```
The end Item : ${status.end}

The step Item : ${status.step}

</c:forEach>
```

**output :**

```
The current Item : Sai
The index : 0
The count : 1
Is it first Item ? true
Is it last Item ? false
The begin Item :
The end Item :
The step Item :
The current Item : Shiva
The index : 1
The count : 2
Is it first Item ? false
Is it last Item ? false
The begin Item :
The end Item :
The step Item :
The current Item : Vishnu
The index : 2
The count : 3
Is it first Item ? false
Is it last Item ? true
The begin Item :
The end Item :
The step Item :
```

#### Example 2 :

```
myJsp.jsp
<c:forEach begin="0" end="10" step="1" varStatus="status">
The begin Item : ${status.begin}

The end Item : ${status.end}

The step Item : ${status.step}

</c:forEach>
```

**output :**

```
The begin Item : 0
The end Item : 10 //11 times
The step Item : 1
```

#### **<c:forTokens> :**

- It is specialized version of forEach to perform String tokenization based on some delimiter(seperator).

- It behaves exactly same as StringTokenizer in core Java.

```
<c:forTokens items="ask,sai,raja,raki" delims=";" var="obj">
.....
</c:forTokens>
```

For each token according to the separator, the body will be executed we can store the current Token by using var attribute.

**Ex :**

myJsp.jsp

```
<c:forTokens items="ask,sai" delims="," var="obj">
The current object : ${obj}

</c:forTokens>

output :
The current object : ask
The current object : sai
```

<c:forTokens> can take the following attributes :

<b>begin :</b>	specifies the index where iteration should start the index of first token is zero.
<b>end :</b>	specifies the index where iteration should terminates.
<b>step :</b>	counter increments value between iterations.
<b>varStatus :</b>	It specifies status of the iteration like it is a first iteration or not.

**Ex :**

```
<c:forTokens items="ask,sai,raja,raki" delims=";" >
No one is Good !!!
</c:forTokens>
```

**Ex :**

```
<c:forTokens items="ask,sai,raki" delims=";" varStatus="status">
Is it first element ? ${status.first}

Is it last element ? ${status.last}

The begin index : ${status.begin}

The end index : ${status.end}

The step index : ${status.step}

No one is Good !!!
</c:forTokens>
output :
Is it first element ? true
Is it last element ? true
```

**The begin index :****The end index :****The step index :****No one is Good !!!**

- In case of <c:forTokens> items attribute should be String only but in the case of <c:forEach> items can be "Collection/Array, Map or String".
- Hence <c:forTokens> is considered as a specialized version of forEach loop.

**Ex :****The combination of varStatus, var is allowed.**

```
<c:forTokens items="ask,sai,raki" delims="," varStatus="status" var="x">
The Current Element : ${status.current} -- ${x}

</c:forTokens>
```

**output :**

The Current Element : ask -- ask

The Current Element : sai -- sai

The Current Element : raki -- raki

**Summary of Iteration Tags :**

Tags	description	attribute
<c:forEach>	general purpose for loop and enhanced for loop	begin, end, items, step, var, varStatus

**begin, end** ---> These are mandatory in the case of normal for loops.**items** ---> This is mandatory in the case of enhanced for loops.

(According to jsp specification all attributes are optional)

<c:forTokens>	Specialized version of StringTokenizer	begin, end, step, var, varStatus, items, delims
---------------	----------------------------------------	-------------------------------------------------

**items, delims** ---> These are mandatory attributes.**URL related tags :****c:import**

- By using <c:import> to include the response of some other JSP into Current JSP at request processing time,  
Hence this inclusion is called Dynamic Include.  
It is exactly equal to <jsp:include> standard action.
- <jsp:include> and include directive applicable with in the same server/container  
but <c:import> can be applicable either with in the same server or outside of the server.  
It is always recommended to use outside of the web server.

```
<jsp:include page="http://localhost:8080/jstl/myJsp.jsp"/> //invalid
<%@include file="http://localhost:8080/jstl/myJsp.jsp"%> //invalid
<c:import url="http://localhost:8080/jstl/myJsp.jsp"/> //valid
```

#### form 1 :

demo.jsp

Hello Demo Jsp World!

myJsp.jsp

Hello, this is from myJsp.jsp on GlassFish server.

```
<c:import url="http://localhost:8080/jstl/demo.jsp"/>
```

output :

Hello, this is from myJsp.jsp on GlassFish server.

Hello Demo Jsp World!

#### form 2 :

myJsp.jsp

Hello, this is from myJsp.jsp on GlassFish server.

```
<c:import url="/demo.jsp" context="/jstl" />
```

// absolute paths

We can import the resources from outside of current application also

(i.e., cross context communication also possible)

#### form 3 :

We can store the result of imported page into a variable specified by var attribute,

Where ever the rest of the JSP, we can use directly that variable without import once again.

myJsp.jsp

Hello,

```
<c:import url="/demo.jsp" var="result" scope="session"/>
```

The result is :\${result}

output :

Hello, The result is :

Hello Demo Jsp World!

Whenever we are using var attribute the result of target JSP store into var attribute , if we want that result we have to retrieve from that var attribute.

#### form 4 :

The more convinient way to store the result of <c:import> is to use Reader object, it is alternative to var attribute.

Hence var and varReader should not come symultaneously.

Hello,

```
<c:import url="demo.jsp" varReader="myReader"/>

<%
java.io.Reader myReader=(java.io.Reader)pageContext.getAttribute("myReader");
int i=myReader.readLine();
write(i!=null){
//you can perform your own operations.
//once checked again
}
%>
```

**form 5:**

While performing import we can send parameters to the target jsp for this we should use `<c:param>` tag these parameters are available in the target Jsp in the form of request parameters (or) form parameters.

demo.jsp  
Hello Demo Jsp World! <br>

The form parameter is : \${param.c1} <br>

The form parameter is : \${param.c2}

myJsp.jsp

Hello, <br>

```
<c:import url="http://localhost:8080/jstl/demo.jsp">
<c:param name="c1" value="SCJP"/>
<c:param name="c2" value="SCWCD"/>
</c:import>
```

output :

Hello,

Hello Demo Jsp World!

The form parameter is : SCJP

The form parameter is : SCWCD

**<c:redirect>:**

We can use this tag to redirect the request to another page, it is exactly equal to sendRedirect of ServletResponse.

**form 1:**

Hello, <br>

```
<c:redirect url="/demo.jsp" />
// here absolute path is optional
```

output :

Hello Demo Jsp World!

**form 2:**

```
<c:redirect url="/demo.jsp" context="/jstl" />
```

We can redirect request to some other web application resources also.

### form 3 :

While performing redirection we can pass parameters of target resources for this we have to use `<c:param>` tag.

demo.jsp

Hello Demo Jsp World! <br>

The form parameter is : \${param.c1} <br>

The form parameter is : \${param.c2}

myJSP.jsp

Hello, <br>

```
<c:redirect url="/demo.jsp" context="/jstl">
```

```
 <param name="c1" value="SCJP"/>
```

```
 <param name="c2" value="SCWCD"/>
```

```
</c:redirect>
```

output :

Hello Demo Jsp World!

The form parameter is :

The form parameter is :

### <c:url> :

We can use this tag to rewrite the url by appending the session information and form parameters to the URL.

In servlet code :

```
PrintWriter out=response.getWriter();
HttpSession session=request.getSession();
out.println(" <a href=\" "+
```

```
 response.encodeURL("test.do")+
```

```
 "\"> click ");
```

In Jsp :

```
<a href=<c:url value="demo.do" />>click me
```

### form 1:

```
<c:url value="demo.jsp" var="x"/>
```

```
Click Me
```

### form 2:

```
<c:url value="/demo.jsp" var="x" context="/jstl" scope="session" />
```

**form 3 :**

```
<c:url value="demo.jsp" var="x" scope="request">
<c:param name="c1" value="SCJP"/>
<c:param name="c2" value="SCWCD"/>
</c:url>
```

- <c:url> tag rewrite the value of var attribute by appending session id, iff cookies are disabled and store into the var attribute.
- Suppose if we are not disable in the cookie <c:url> won't append the session id to the url.

```
<c:url value="demo.jsp" var="x">
<c:param name="c1" value="SCJP"/>

<c:param name="c2" value="SCWCD"/>

</c:url>
```

The value of x : \${x} <br>  
[Click Me](#)

**output :**

The value of x : demo.jsp?c1=SCJP&c2=SCWCD  
[Click Me](#)

Hello Demo Jsp World!

The form parameter is : SCJP

The form parameter is : SCWCD

- URL-encoding means replacing the unsafe (or) reserved characters with other characters and the whole thing is decoded again on the server side.
- Ex : spaces are not allowed in URL but we can substitute "+" sign for the spaces.
- The problem in <c:url> is doesn't automatically encode your URLs.
- We can encode the URLs by using <c:param> tag
- <c:url> tag can do only rewrite the URL but not encode the URL.

**Using <c:url> with a query string :**

myJsp.jsp

```
<c:set var="first" value="Ashok"/>
<c:set var="last" value="Agg"/>

<a href=<c:url value='demo.jsp?first=${first}&last=${last}' var='x'>
Click Me

The URL using param is :${x}

```

Using Param tag in the body url-rewriting and Url-encoding<br>

```
<c:url value="demo.jsp" var="y">
```

```
<c:param name="first" value="${first}" />

<c:param name="last" value="${last}" />

</c:url>
```

The value of y : \${y} <br>
[Click Me](#)

**demo.jsp**

Hello Demo Jsp World! <br>
The form parameter is : \${param.first} <br>
The form parameter is : \${param.last}

**output :**

Click Me  
The URL using param is : demo.jsp?first=Ashok&last=Agg  
Using Param tag in the body url-rewriting and Url-encoding  
The value of y : demo.jsp?first=Ashok&last=Agg  
Click Me

Hello Demo Jsp World!  
The form parameter is : Ashok  
The form parameter is : Agg

#### Summary of url tags :

Tag	Description	Attributes
<c:import>	For importing the response of other page into current page into request processing time.	url, var, scope, varReader, context, charEncoding
<c:redirect>	To redirect the request to other web components.	url, context
<c:url>	To rewrite url by appending session information and parameters.	value, var, scope, context
<c:param>	To send the parameters while implementing and Redirecting.	name, value



**LEARN FROM EXPERTS ...**

# **COMPLETE JAVA**

CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,....

# **COMPLETE .NET**

C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

# **TESTING TOOLS**

MANUAL + SELENIUM

# **ORACLE D2K**

# **MSBI SHARE POINT**

# **HADOOP ANDROID**

# **C, C++, DS, UNIX**

# **CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED

**DURGA**  
Software Solutions®

# 202, 2nd Floor, HUDA Maitrivanam,

Ameerpet, Hyd. Ph: 040-64512786,

**9246212143, 8096969696**

**www.durgasoft.com**

**Adv. Java means DURGA SIR..**

# **ADV.JAVA With SCWCD / OCWCD JSP Material**

**5. Building a Custom Tag Library**



**DURGA M.Tech**

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students  
for last 14 years across INDIA

**India's No.1 Software Training Institute**

**DURGASOFT**

**www.durgasoft.com Ph: 9246212143 ,8096969696**

173

DURGASOFT, # 202, 2<sup>nd</sup> Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,

040 - 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | [www.durgasoft.com](http://www.durgasoft.com)

## JSP Custom Tag Library

### Agenda :

- ❖ Introduction
  - ❖ Components of Custom tag applications
  - ❖ Tag Handler Class
  - ❖ TLD(Tag Library descriptor)
  - ❖ Taglib directive
  - ❖ Flow of Custom Tag Application
  - ❖ Tag Extension API
- 1) Classic Tag Model
  - Tag interface
  - IterationTag (I)
  - TagSupport (C)
  - BodyTag (I)
  - BodyContent (AC)
    - Comparision between TagSupport and BodyTagSupport classes
    - Nested Tags (or) co-operative tags
    - Getting Arbitrary Ancestor Object
- 2) SimpleTag Model : (jsp 2.0v)
  - SimpleTag interface
  - Jsp implicit objects and attributes in SimpleTag Model
  - Key differences between ClassictagModel and SimpleTagModel
  - DynamicAttributes
- 3) TagFiles (jsp2.0v)
  - Building & Using a Simple tag file
  - Declaring body-content for tag file
  - TagFiles with DynamicAttributes

### **Introduction :**

Standard Actions, EL, JSTL are not succeeded to complete elimination of java code from Jsp.

**Ex:** Our requirement is to communicate with EJP or DB there is no standard action is defined for this requirement, we can defined our own tag from Jsp 1.1v onwards to meet our programming requirement such types of tags are nothing but Custom Tags.

All Custom tags can be divided into 3 parts

#### 1. Classic Tag Model (Jsp1.1)

2. Simple Tag Model(2.0)
3. Tag files (Jsp2.0)

### Components of Custom tag applications :

Custom tag application contains the following 3 components

**1) Tag Handler Class :**

- It is a simple java class which defines entire required functionality.
- Every Tag Handler class should compulsory implements Tag interface either directly or indirectly.
- Web-container is responsible for creation of Tag handler class object for this always invokes public no-arg constructor , hence every tag handler class should compulsory contains public no-arg constructor.

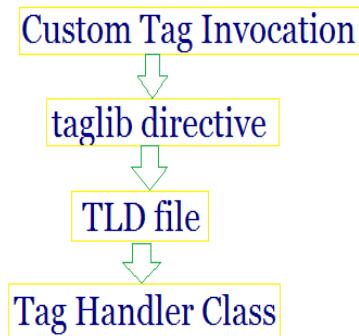
**2) TLD(Tag Library descriptor) :**

It is an xml file which provides mapping between Jsp(where custom tag functionality is required) and Tag handler class (where custom tag functional available).

**3) Taglib directive :**

We can use Taglib directive to make custom tag functionality available to the Jsp, it provides location of TLD file.

### Flow of Custom Tag Application :



- When ever Jsp engine encounters a custom tag it identifies prefix and checks for corresponding taglib directive with matched prefix from the taglib directive Jsp engine identifies location of TLD file.
- From the TLD file Jsp engine identifies corresponding Tag handler class Jsp engine execute the THC and provides required functionality to the Jsp.

### Tag Extension API :

We can define custom tags by using only one package `javax.servlet.jsp.tagext`;

This package contains the following interfaces and class:

- `Tag (I)`
- `IterationTag (I)`
- `BodyTag (I)`
- `TagSupport (C)`
- `BodyTagSupport (C)`
- `BodyContext (C)`

#### **Tag (I) :**

- It act as a base interface for all THC
- Every THC should implement Tag interface either directly or indirectly.
- This interface defines 6 methods , these methods can be applicable on any THC object.
- If we want to include Tag body at most once without manipulation then we should go for Tag interface.

#### **IterationTag (I) :**

- It is the child interface of Tag , if we want to include Tag body multiple times without any manipulation then we should go for IterationTag interface.
- It defines one extra method `doAfterBody()`.

#### **BodyTag (I) :**

It is the child interface of IterationTag, if we want to manipulate the TagBody then we should go for BodyTag interface.

It defines the following 2 methods

1. `setBodyContent()`
2. `doInitBody()`

#### **TagSupport (C) :**

1. It implements IterationTag interface and provides default implementation for all methods and hence this class act as base class to develop Tag Handler class where TagBody can be included any no.of times without manipulation.
2. This class act as a adaptor for Tag and IterationTag interfaces.

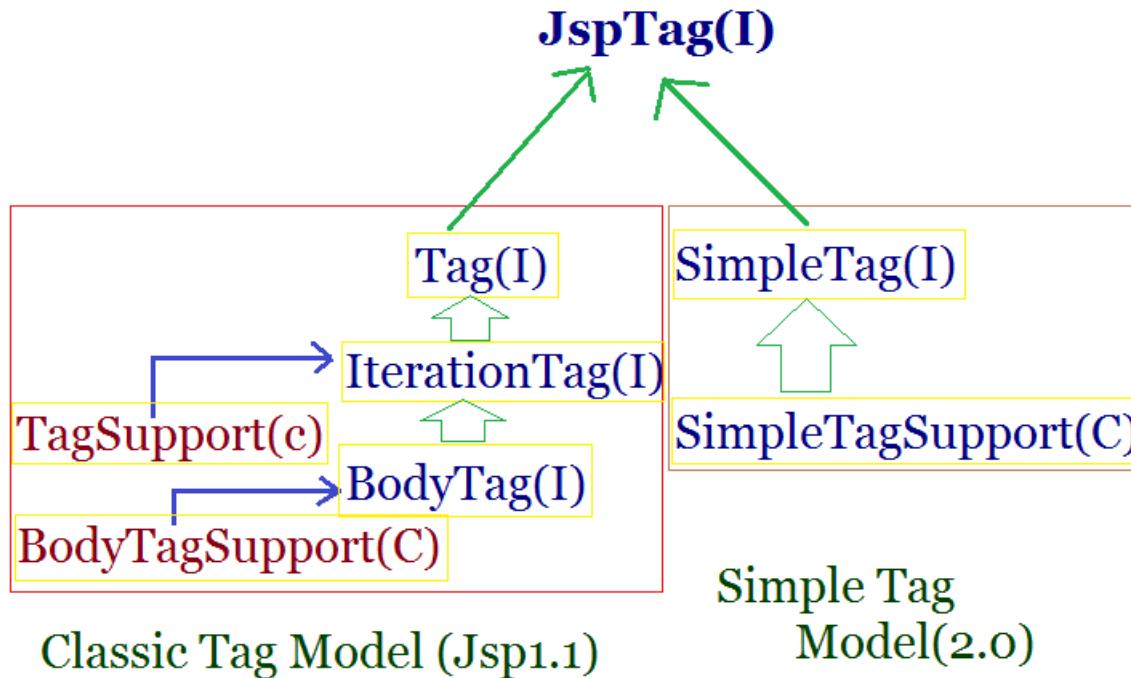
#### **BodyTagSupport (C) :**

1. This class implements BodyTag interface and provides default implementation for all its methods and it is child class of TagSupport class.

2. We can use this class as a base class for implementing custom tag that can process the tag body.
3. This class act as a adaptor for BodyTag interface.

### BodyContext (C) :

1. BodyContext object act as a buffer for Tag body, it is the child class of JspWriter.
2. We can use this BodyContext class only with in BodyTag interface and BodyTagSupport class.



JspTag interface just for polymorphism purpose and doesn't contain any methods.

### Classic Tag Model

#### Tag interface :

Tag interface defines the following 6 methods :

1. setPageContext()
2. setParent()
3. doStartTag()
4. doEndTag()
5. getParent()
6. release()

Tag interface defines the following 4 constants :

1. EVAL\_BODY\_INCLUDE
2. SKIP\_BODY
3. EVAL\_PAGE
4. SKIP\_PAGE

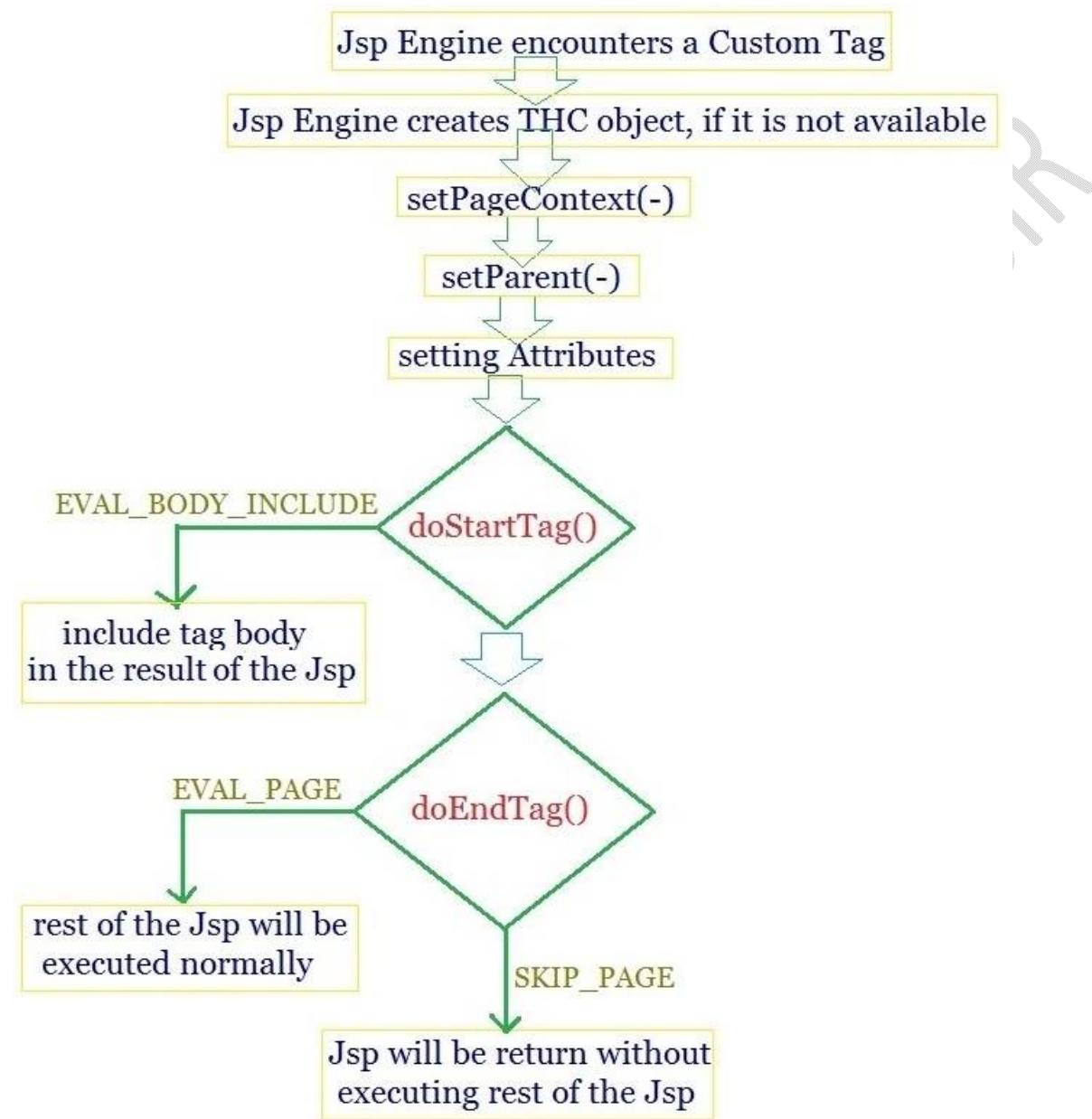
### LifeCycle of Tag Handler class that implements Tag interface :

- 1) When ever Jsp engine encounters a Custom tag invocation it will identify corresponding tag handler class through taglib directive and TLD file.
- 2) Web-container will checks whether the corresponding Tag handler class object is available or not , if it is not available Jsp engine creating an object by executing public no-arg constructor , hence every Tag handler class should compulsory contains public no-arg constructor , violation leads we will get instantiation exception.
- 3) Jsp engine calls setPageContext(-) to make PageContext object available to the Tag handler class.  
`public void setPageContext(PageContext pcontext)`
- 4) THC can use this PageContext object to get all other Jsp implicit attributes.
- 5) Jsp engine calls setParent() to make parent tag object available to Tag Handler class  
`public void setParent(Tag tag)`  
This method is useful in nesting

### Setting Attributes :

- A custom tag can be invoked with attributes also for every attribute the corresponding THC should contains one instance variable and corresponding setter methods like a bean , these are exactly same as properties of bean class for every custom tag attributes Jsp engine will execute corresponding setter method to make attribute values make its available to THC.
- Jsp engine will invoke doStartTag()  
`public int doStartTag() throws JspException`
- Entire custom tag functionality we have to define in this method only , this method can return either EVAL\_BODY\_INCLUDE or SKIP\_BODY
- If this method return EVAL\_BODY\_INCLUDE then Tag Body will be included in the result of Jsp (output of the Jsp).
- If this method return a SKIP\_BODY , then Jsp engine don't consider Tag Body.
- Jsp engine calls doEndTag() `public int doEndTag() throws JspException` doEndTag() can returns if this method return EVAL\_PAGE the rest of the Jsp will be executed normally.
- If the returns SKIP\_PAGE then Jsp engine will return without executing rest of the Jsp.
- Finally Jsp engine calls release() to perform cleanup activities when ever Tag Handler Object is no longer required.  
`public void release()`

For every custom tag invocation the following sequence of events will be happened (Flow Chart)



But release() method will not be called for every custom tag invocation.

#### custom tag example :

index.jsp

```
<%@taglib uri="customtags" prefix="mine" %>
Hello this is Tag demo Jsp
```

```
<mine:welcome>
This is body of the custom tag
</mine:welcome>
This is after tag
```

### TagDemo.java

```
package com.tag;

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 TagDemo implements Tag {

 public PageContext pageContext;

 static {
 System.out.println("class loading");
 }

 public TagDemo() {
 System.out.println("instantiation");
 }

 public void setPageContext(PageContext pageContext) {
 this.pageContext = pageContext;
 System.out.println("pagecontext object setted");
 }

 public void setParent(Tag tag) {
 System.out.println("parent tag setted");
 }

 public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");
 JspWriter out = null;
 try {
 out = pageContext.getOut();
 out.println("Welcome to custom tag developers");
 out.println("Welcome to custom Tag handler class");
 } catch (IOException e) {
 e.printStackTrace();
 }
 //return SKIP_BODY; // output 1
 return EVAL_BODY_INCLUDE; //output 2
 }
}
```

```

 }

 public int doEndTag() throws JspException {
 System.out.println("This is doEndTag() method");
 return EVAL_PAGE; //output 1
 //return SKIP_PAGE; //output 2
 }
 public Tag getParent() {
 System.out.println("getParent() method");
 return null;
 }

 public void release() {
 System.out.println("release() method");
 pageContext = null;
 }
}

```

same program using annotations :

```

package com.tag;

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 TagDemo implements Tag {

 private PageContext pageContext;

 static {
 System.out.println("class loading");
 }

 public TagDemo() {
 System.out.println("instantiation");
 }

 @Override
 public void setPageContext(PageContext pageContext) {
 this.pageContext = pageContext;
 System.out.println("pagecontext object setted");
 }

 @Override
 public void setParent(Tag tag) {

```

```
 System.out.println("parent tag setted");
 }

 @Override
 public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");

 @SuppressWarnings("UnusedAssignment")
 JspWriter out = null;
 try {
 out = getPageContext().getOut();
 out.println("Welcome to custom tag developers
");
 out.println("Welcome to custom Tag handler class
");
 } catch (IOException e) {
 }
 //return SKIP_BODY; // output 1
 return EVAL_BODY_INCLUDE; //output 2
 }

 @Override
 public int doEndTag() throws JspException {
 System.out.println("This is doEndTag() method");
 return EVAL_PAGE; //output 1
 //return SKIP_PAGE; //output 2
 }

 @Override
 public Tag getParent() {
 System.out.println("getParent() method");
 return null;
 }

 @Override
 public void release() {
 System.out.println("release() method");
 setPageContext(null);
 }

 public PageContext getPageContext() {
 return pageContext;
 }
}
```

myTld.tld

```
<taglib version="2.1">
 <tlib-version>1.2</tlib-version>
```

```
<uri>customtags</uri>
<tag>
 <name>welcome</name>
 <tag-class>com.tag.TagDemo</tag-class>
</tag>
</taglib>
```

In the above program , if doStartTag() returns EVAL\_BODY\_INCLUDE and doEndTag returns EVAL\_PAGE then the following is output :

Hello this is Tag demo Jsp  
 Welcome to custom tag developers  
 Welcome to custom Tag handler class  
 This is body of the custom tag  
 This is after tag

in console :

Info: class loading  
 Info: instantiation  
 Info: pagecontext object setted  
 Info: parent tag setted  
 Info: This is doStartTag() method  
 Info: This is doEndTag() method

#### Mapping of the Jsp with tld file :

##### approach 1 :

We can hard code the location of tld file for the uri attribute of taglib directive

Ex:

```
<%@taglib uri="/WEB-INF/myTld.tld" prefix="mine" %>
```

The problem in this approach is if there is change in name of the tld file or location of the tld file we have perform that which is complex to the programmer.

##### approach 2 :

Instead of hard coding the location of tld file we can define mapping through web.xml

Ex:

```
<%@taglib uri="http://jobs4times.com/scwcd/tags" prefix="mine" %>
```

web.xml

```
<web-app>
 <jsp-config>
```

```
<taglib>
 <taglib-uri>http://jobs4times.com/scwcd/tags</taglib-uri>
 <taglib-location>/WEB-INF/myTld.tld</taglib-location>
</taglib>
</jsp-config>
</web-app>
```

This approach is highly recommended to use because there is any change in location of tld, just change in web.xml is enough not required to change in all Jsp's

### approach 3 :

We can map taglib uri attribute directly with uri attribute of tld file

```
<%@taglib prefix="mine" uri="customtags" %>
```

myTld.tld

```
<taglib version="2.1">
 <tlib-version>1.2</tlib-version>
 <uri>mycustomtags</uri>

</taglib>
```

This approach is not recommended because all web-servers may not be supported.

### Structure of TLD file :

```
<taglib version="2.1">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>welcome</name>
 <tag-class>com.tag.CustomTagDemo</tag-class>
 <body-content>XXXX</body-content>
 <attribute>
 <name>msg</name>
 <required>true</required>
 <rteprvalue>true</rteprvalue>
 </attribute>

 </tag>
</taglib>
```

In the above tld file we can write EL functions also.

### <body-content> :

It describes the type of content allow inside tag body the allowed values are

1. empty : The body of custom tag should be empty i.e., we can't take any tag body in this case we can invoke custom tag as follows <mine:mytag/>
  2. tagdependent : Entire tag body will be treated plain text, Jsp engine sends tag body to the tag handler class without any processing.
  3. scriptless : Tag body should not contains any scripting elements scriptlet, expression, declarations are not allowed but standard actions and EL expressions are allowed.
  4. jsp : There are no restrictions on tag body whatever allowed in Jsp by default allowed in tag body also(including scripting elements).
- Note : The default value of <body-content> is "jsp".

#### <attribute> :

A custom tag can be invoked with attribute also we have to declare these attributes by using attribute tag , this tag contains the following child tags

1. <name> : name of the attribute
2. <required> : true means mandatory attribute, false means optional attribute, the default value is false
3. <rexprvalue> (runtime expression value) :true means runtime expressions are allowed, false means runtime expressions are not allowed

```
<mime:mytag color="#${param.color}" />
```

we have to provide only literals

```
<mime:mytag color="red" />
```

The default value is false

#### Write a demo program : empty custom tag with mandatory attribute :

A tag contain attribute also for each attribute we have to do the following things

- We have to declare that attribute in the TLD file by using <attribute> tag
- For each attribute THC should contain one instance variable and corresponding setter methods.
- In the case of optional attribute there may be a chance of NullPointerException will raise hence we have to handle carefull.

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
<%@page isELIgnored="false"%>

<mine:double number="3"/>
<mine:double number="#${param.number}" />
```

**TagAttributeDemo.java**

```
package com.tag;

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 TagAttributeDemo implements Tag {

 public PageContext pageContext = null;
 private int number = 0;

 @Override
 public void setPageContext(PageContext pageContext) {
 this.pageContext = pageContext;
 System.out.println("pagecontext object setted");
 }

 @Override
 public void setParent(Tag tag) {
 System.out.println("parent tag setted");
 }

 public void setNumber(int number) {
 this.number = number;
 }

 @Override
 @SuppressWarnings("CallToPrintStackTrace")
 public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");

 @SuppressWarnings("UnusedAssignment")
 JspWriter out = null;
 try {
 out = pageContext.getOut();
 out.println("The double of the given no "+number+" is "+(2*number));
 } catch (IOException e) {
 e.printStackTrace();
 }
 return SKIP_BODY;
 // return EVAL_BODY_INCLUDE;
 }

 @Override
```

```

public int doEndTag() throws JspException {
 System.out.println("This is doEndTag() method");
 //return EVAL_PAGE;
 return SKIP_PAGE;
}

@Override
public Tag getParent() {
 System.out.println("getParent() method");
 return null;
}

@Override
public void release() {
 System.out.println("release() method");
 pageContext = null;
}

}

```

**myTld.tld**

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <description>double of number</description>
 <name>double</name>
 <tag-class>com.tag.TagAttributeDemo</tag-class>
 <body-content>empty</body-content>
 <attribute>
 <name>number</name>
 <required>true</required>
 <rtpvalue>true</rtpvalue>
 </attribute>
 </tag>
</taglib>

```

**web.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
 http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

```

```
<jsp-config>
<taglib>
<taglib-uri>http://jobs4times.com/tags</taglib-uri>
<taglib-location>/WEB-INF/myTld.tld</taglib-location>
</taglib>
</jsp-config>
</web-app>
```

<http://localhost:8080/jstl/test.jsp?number=7>

**Write a demo program : empty custom tag with optional attribute :**

demo.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
<%@page isELIgnored="false"%>

<mine:welcome name="Ashok"/>

<mine:welcome name="Arun"/>

<mine:welcome />

<mine:welcome name="Agastya"/>

```

TagOptional.java

```
package com.tag;

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 TagOptional implements Tag {

 public PageContext pageContext=null;
 private String name=null;

 @Override
 public void setPageContext(PageContext pageContext) {
 this.pageContext=pageContext;
 System.out.println("pagecontext object setted");
 }

 @Override
 public void setParent(Tag tag) {
 System.out.println("parent tag setted");
 }

 public void setName(String name){
```

```
this.name=name;
}

@Override
@SuppressWarnings("CallToPrintStackTrace")
public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");

 @SuppressWarnings("UnusedAssignment")
 JspWriter out=null;
 try{
 out=pageContext.getOut();
 if(name==null){
 out.println(
 Hi Guest welcome to Custom Tags");
 }
 else{
 out.println(
 Hi "+name+ welcome to Custom Tags");
 }
 }catch (IOException e) {
 e.printStackTrace();
 }
 return SKIP_BODY;
// return EVAL_BODY_INCLUDE;
}

@Override
public int doEndTag() throws JspException {
 System.out.println("This is doEndTag() method");
 return EVAL_PAGE;
//return SKIP_PAGE;

}

@Override
public Tag getParent() {
 System.out.println("getParent() method");
 return null;
}

@Override
public void release() {
 System.out.println("release() method");
 pageContext=null;
}
```

```
}
```

**myTld.tld**

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
<tlib-version>1.2</tlib-version>
<uri>www.jobs4times.com</uri>
<tag>
<name>welcome</name>
<tag-class>com.tag.TagOptional</tag-class>
<body-content>empty</body-content>
<attribute>
<name>name</name>
<required>false</required>
</attribute>
</tag>
</taglib>
```

**web.xml**

```
same as previous
```

**output:**

```
Hi Ashok welcome to Custom Tags
Hi Arun welcome to Custom Tags
Hi Guest welcome to Custom Tags
Hi Agastya welcome to Custom Tags
```

**IterationTag()** :

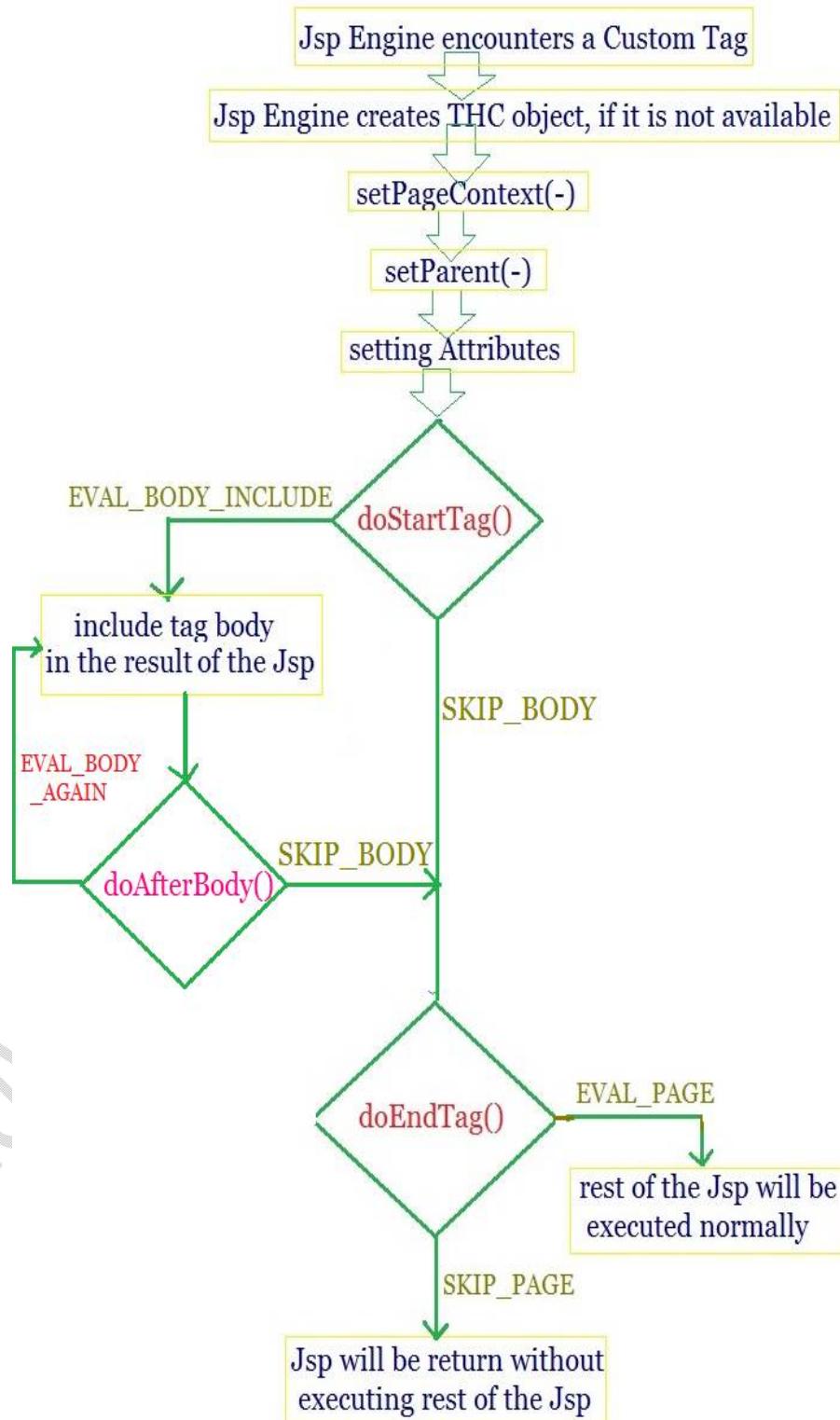
1. It is child interface of Tag
2. If we want to include tag body multiple times without manipulation then we should go for IterationTag.
3. IterationTag interface contains one extra method **doAfterBody()** one extra constant is **EVAL\_BODY\_AGAIN**

**doAfterBody()** :

```
public int doAfterBody() throws JspException
```

- This method will be executed after **doStartTag()** and it can return either **EVAL\_BODY\_AGAIN** or **SKIP\_BODY**
- If it returns **EVAL\_BODY\_AGAIN** then tag body will be consider once again followed by execution of **doStartBody()**

## Flow Chart of IterationTag :



**Write a program for IterationTag :****test.jsp**

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag count="3">
This is Iteration Tag Demo

</mine:myTag>
Hi, This is afterBody
```

**IterationTagDemo.java**

```
package com.tag;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.IterationTag;
import javax.servlet.jsp.tagext.Tag;

public class IterationTagDemo implements IterationTag {

 @SuppressWarnings("PublicField")
 public PageContext pageContext=null;
 private int count=0;
 private Tag tag=null;

 @Override
 public void setPageContext(PageContext pageContext) {
 this.pageContext=pageContext;
 System.out.println("pagecontext object setted");
 }

 @Override
 public void setParent(Tag tag) {
 this.tag=tag;
 System.out.println("parent tag setter");
 }

 public void setCount(int count){
 this.count=count;
 }

 @Override
 public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");

 if(count>0){
 return EVAL_BODY_INCLUDE;
 }
 }
}
```

```
}

else{
 return SKIP_BODY;
}

}

@Override
public int doAfterBody() throws JspException{
if(-count>0)
 return EVAL_BODY_AGAIN;
else
 return SKIP_BODY;
}

@Override
public int doEndTag() throws JspException {
System.out.println("This is doEndTag() method");
return EVAL_PAGE;
}

@Override
public Tag getParent() {
System.out.println("getParent() method");
return tag;
}

@Override
public void release() {
System.out.println("release() method");
pageContext=null;
count=0;
tag=null;
}

}
```

#### myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
<tlib-version>1.2</tlib-version>
<uri>www.jobs4times.com</uri>
<tag>
<name>myTag</name>
<tag-class>com.tag.IterationTagDemo</tag-class>
<body-content>tagdependent</body-content>
<attribute>
<name>count</name>
```

```
<required>true</required>
</attribute>
</tag>
</taglib>
```

**web.xml**

```
<jsp-config>
<taglib>
<taglib-uri>http://jobs4times.com/tags</taglib-uri>
<taglib-location>/WEB-INF/myTld.tld</taglib-location>
</taglib>
</jsp-config>
```

**output :**

This is Iteration Tag Demo  
This is Iteration Tag Demo  
This is Iteration Tag Demo  
Hi, This is afterBody

**TagSupport (C) :**

- The main drawback of implements Tag and IterationTag interfaces directly is we have to provide implementation for all methods even though most of the times we have to consider doStartTag(), doAfterBody(), doEndTag()
- We can resolve this problem by using TagSupport class can implement IterationTag and provides default implementation for all its methods and it is very easy to extend TagSupport class and override required methods instead of implementing all methods.

**Internal implementation of TagSupport class :**

```
import java.io.Serializable;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.IterationTag;
import javax.servlet.jsp.tagext.Tag;

public class TagSupport implements IterationTag,Serializable {
 private static final long serialVersionUID = 1L;

 private Tag parent;
 protected PageContext pageContext;

 @Override
 public void setPageContext(PageContext pageContext) {
 this.pageContext=pageContext;
 System.out.println("pagecontext object setted");
 }
}
```

```
}

@Override
public void setParent(Tag parent) {
 this.parent=parent;
 System.out.println("parent tag setter");
}

@Override
public int doStartTag() throws JspException {
 System.out.println("This is doStartTag() method");
 return SKIP_BODY;
}

@Override
public int doAfterBody() throws JspException{
 System.out.println("This is doAfterBody() method");
 return SKIP_BODY;
}

@Override
public int doEndTag() throws JspException {
 System.out.println("This is doEndTag() method");
 return EVAL_PAGE;
}

@Override
public Tag getParent() {
 System.out.println("getParent() method");
 return parent;
}

@Override
public void release() {
 System.out.println("release() method");
 pageContext=null;
 parent=null;
}
}
```

**Note :**

1. The default return type of doStartTag(), doAfterBody() is SKIP\_BODY
2. The default return type of doEndTag() is EVAL\_PAGE

pageContext variable is by default variable to the child classes hence we can use this variable directly in our tag handler classes

**Write a demo program for TagSupport class :**

**test.jsp**

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

This is before custom tag

<mine:myTag>
This is Tag body

</mine:myTag>
This is after custom tag
```

**TagSupportDemo.java**

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class TagSupportDemo extends TagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public int doStartTag() throws JspException {
 JspWriter out=null;
 try{
 out=pageContext.getOut();
 out.println("This is THC using TagSupport class
");
 }
 catch(IOException e){
 e.printStackTrace();
 }

 return EVAL_BODY_INCLUDE;
 }
}
```

**myTld.tld**

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
```

```
<name>myTag</name>
<tag-class>com.tag.TagSupportDemo</tag-class>
<body-content>tagdependent</body-content>
</tag>
</taglib>
```

**web.xml**

```
<jsp-config>
<taglib>
<taglib-uri>http://jobs4times.com/tags</taglib-uri>
<taglib-location>/WEB-INF/myTld.tld</taglib-location>
</taglib>
</jsp-config>
```

<http://localhost:8080/jstl/test.jsp?>

**output :**

This is before custom tag  
This is THC using TagSupport class  
This is Tag body  
This is after custom tag

**Example : 2****test.jsp**

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag count="3">
This is Iteration body

</mine:myTag>
This is after Iteration
```

**TagSupportDemo.java**

```
package com.tag;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;

public class TagSupportDemo extends TagSupport {
private static final long serialVersionUID = 1L;

private int count=0;

public void setCount(int count){
this.count=count;
}
```

```

@Override
public int doStartTag() throws JspException {
if(count>0)
return EVAL_BODY_INCLUDE;
else
return SKIP_BODY;
}

@Override
public int doAfterBody() throws JspException{
if(--count>0)
return EVAL_BODY_AGAIN;
else
return SKIP_BODY;
}

```

**myTld.tld**

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
<tlib-version>1.2</tlib-version>
<uri>www.jobs4times.com</uri>
<tag>
<name>myTag</name>
<tag-class>com.tag.TagSupportDemo</tag-class>
<body-content>tagdependent</body-content>
<attribute>
<name>count</name>
<required>true</required>
</attribute>
</tag>
</taglib>

```

**web.xml**

```

<jsp-config>
<taglib>
<taglib-uri>http://jobs4times.com/tags</taglib-uri>
<taglib-location>/WEB-INF/myTld.tld</taglib-location>
</taglib>
</jsp-config>

```

**output :**

This is Iteration body  
This is Iteration body  
This is Iteration body

This is after Iteration

### BodyTag (I) :

- It is the child interface of IterationTag
- If we want to manipulate the body then we should go for BodyTag interface.
- BodyTag interface defines the following 2 extra methods
  1. public void setBodyContent(BodyContent bodyContent)
  2. public void doInitBody() throws JspException
- BodyTag interface defines the following 2 extra constants
  1. EVAL\_BODY\_BUFFERED
  2. EVAL\_BODY\_TAG (deprecated)

### BodyContent (AC) :

- We can use this BodyContent object to hold tag body
- BodyContent class is the child class of JspWriter
- BodyContent class is an abstract class and vendor is responsible to provide implementation

Note : BodyContent class object act as a buffer for tag body , if we want to manipulate that tag body, first we have to retrieve that tag body from the BodyContent object , for that BodyContent class defines the following methods

1. public String getString() : returns tag body in the form of String.
2. public Reader getReader() : return a Reader object to extract tag body.
3. public JspWriter getEnclosingWriter() : It returns JspWriter to print data to the jsp page.
4. public void ClearBody() : To clear body present in BodyContent object i.e., entire tag body present in BodyContent will be removed.

**www.durgasoftonlinetraining.com**



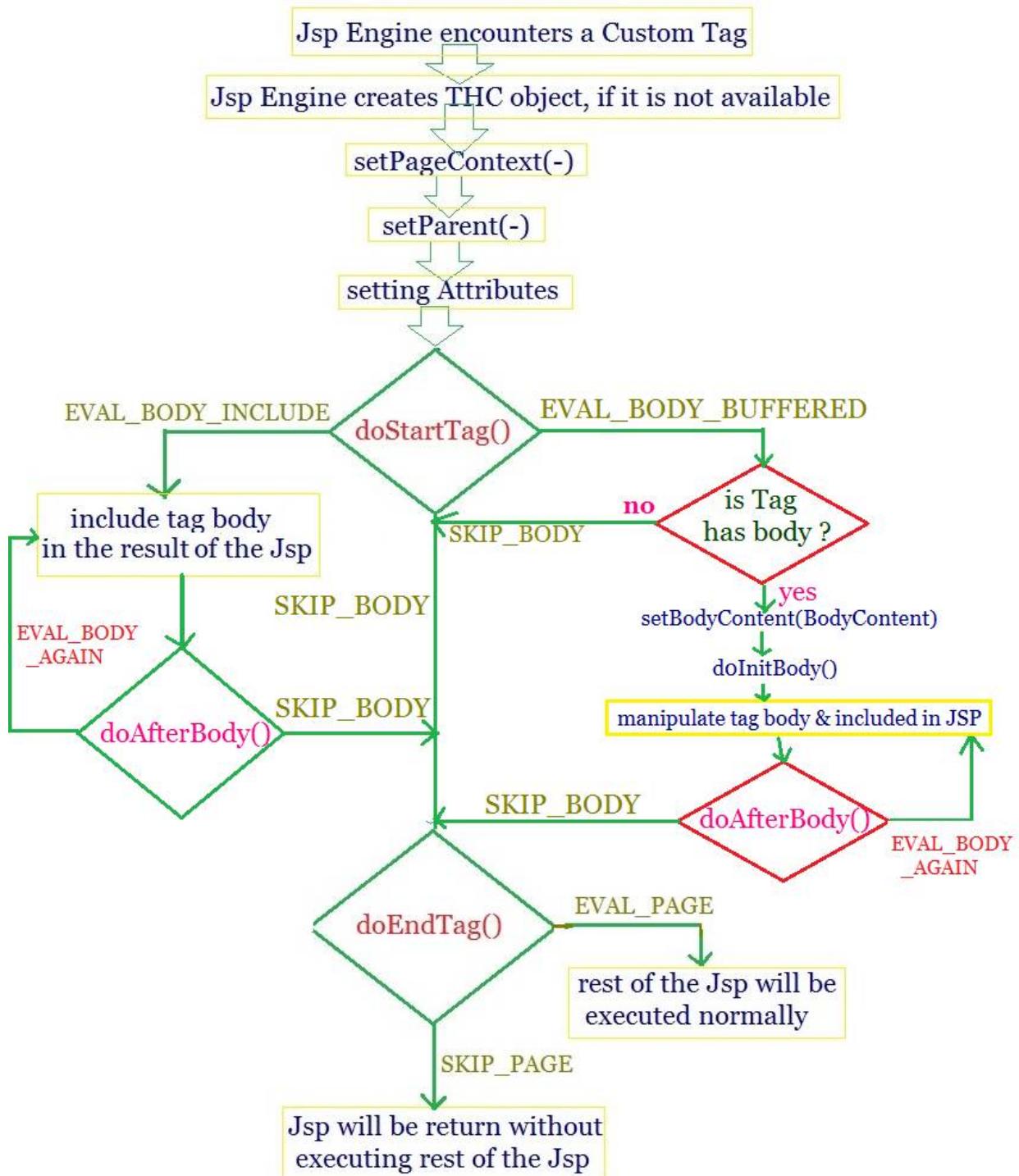
**Online Training  
Pre Recorded Video  
Classes Training  
Corporate Training**

**Ph: +91-8885252627, 7207212427  
+91-7207212428**

**USA Ph : 4433326786**

**E-mail : durgasoftonlinetraining@gmail.com**

## Flow Chart of BodyTag interface :



### Life Cycle of BodyTag interface :

- Life Cycle of BodyTag interface is exactly similar to IterationTag handler, and difference is doStartTag() returns either EVAL\_BODY\_INCLUDE or EVAL\_BODY\_BUFFERED.
- If the method returns EVAL\_BODY\_BUFFERED and tag contains body then jsp engine creates BodyContent object and invoke setBodyContent() by passing BodyContent object as a argument followed by doInitBody().

Note : we can write a logic to manipulate the tag body with in doAfterBody()

**setBodyContent(-) and doInitBody() won't be executed in the following cases**

**case 1 :** If doStartTag() returns either EVAL\_BODY\_INCLUDE or SKIP\_BODY.

**CASE 2 :** If doStartTag() returns either EVAL\_BODY\_BUFFERED and tag doesn't contain body.

### Implementation of BodyTagSupport class :

- This class extends TagSupport class and implements BodyTag interface.
- This class provides default implementation for all 9 methods available in BodyTag interface.
- It is very easy to write Tag handler class by extending BodyTagSupport class instead of implementing BodyTag interface directly.
- In this case we have to provide implementation only for required methods but not for all 9 methods.

### Internal Implementation of BodyTagSupport class :

#### BodyTagSupport.java

```
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTag;
import javax.servlet.jsp.tagext.TagSupport;

public class BodyTagSupport extends TagSupport implements BodyTag{
 private static final long serialVersionUID = 1L;
 protected BodyContent bodyContent;

 @Override
 public void setBodyContent(BodyContent bodyContent){
 this.bodyContent=bodyContent;
 }

 @Override
```

```

public void doInitBody(){
 System.out.println(" doInitBody method");
}

@Override
public int doStartTag() throws JspException{
 return EVAL_BODY_BUFFERED;
}

}

```

- `pageContext`, `bodyContent` variables are by default available to our tag handler classes we can use directly.
- The default return type `doStartTag()` in `BodyTagSupport` is `EVAL_BODY_BUFFERED`, `doAfterBody()` is `SKIP_BODY` and `doEndTag` is `EVAL_PAGE`.

#### Write a demo program for the usage of `BodyTagSupport` class

`test.jsp`

```

<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag>
This is Body Tag Support Demo body

</mine:myTag>

```

`BodyTagSupportDemo.java`

```

package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.BodyTagSupport;

public class BodyTagSupportDemo extends BodyTagSupport{
 private static final long serialVersionUID = 1L;

 @Override
 public int doAfterBody() throws JspException{
 JspWriter out=null;
 try{
 String data=bodyContent.getString();
 data=data.toUpperCase();
 out=bodyContent.getEnclosingWriter();
 out.println(data);
 }
 catch(IOException e){

```

```

e.printStackTrace();
}
return SKIP_BODY;
}
}

```

**myTld.tld**

```

<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.BodyTagSupportDemo</tag-class>
 </tag>
</taglib>

```

**web.xml**

Same as previous

**output :**

THIS IS BODY TAG SUPPORT DEMO BODY

**Comparision between TagSupport and BodyTagSupport classes :**

Method	TagSupport (C)	BodyTagSupport (C)
<b>doStartTag() :</b>		
Possible return values	EVAL_BODY_INCLUDE SKIP_BODY	EVAL_BODY_BUFFERED EVAL_BODY_INCLUDE SKIP_BODY
Default return values from implemented classes	SKIP_BODY	EVAL_BODY_BUFFERED
no.of times invoke per tag	only once	only once
<b>doAfterBody() :</b>		
Possible return values	EVAL_BODY_AGAIN SKIP_BODY	EVAL_BODY_AGAIN SKIP_BODY
Default return values from implemented classes	SKIP_BODY	SKIP_BODY
no.of times invoke per tag	0 or more	0 or more
<b>doEndTag() :</b>		
Possible return values	EVAL_PAGE SKIP_PAGE	EVAL_PAGE SKIP_PAGE

Default return values from implemented classes	EVAL_PAGE	EVAL_PAGE
no.of times invoke per tag	only once	only once
<b>setBodyContent() &amp; doInitBody()</b> :		
circumstances under which these methods no.of times call per tag ?	not applicable	executed only once iff doStartTag() return EVAL_BODY_BUFFERED and tag has body.

### Nested Tags (or) co-operative tags :

Some times a group of tags work together will perform certain functionality such type of tags are called Nested tags

Ex : In JSTL <c:choose>, <c:when>, <c:otherwise> tags work together to implement core java if-else and switch statement such type of tags are called Co-operative & Nested Tags

Ex:

```
<c:choose>
<c:when>
 Action
</c:when>
<c:otherwise>
 Default Action
</c:otherwise>
</c:choose>
```

Ex:

```
<mine:myTag>
<mine:myDetails />
</mine:myTag>
```

From child tag handler if we want to get parent tag object we have to use getParent() method

### Write a demo program for nested custom tags :

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:myTag>
<mine:myTag>
 <mine:myTag>
 <mine:myTag />
 </mine:myTag>
</mine:myTag>
</mine:myTag>
```

NestedTagDemo.java

```
package com.tag;
```

```

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.Tag;
import javax.servlet.jsp.tagext.TagSupport;
public class NestedTagDemo extends TagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public int doStartTag() throws JspException{
 int nestedLevel=0;
 JspWriter out=null;
 Tag tag=getParent();
 while(tag!=null){
 nestedLevel++;
 tag=tag.getParent();
 }
 try{
 out=pageContext.getOut();
 out.println("
Nested Level : "+nestedLevel);
 }
 catch(IOException e){
 e.printStackTrace();
 }
 return EVAL_BODY_INCLUDE;
 }

}

```

**myTld.tld**

```

<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.NestedTagDemo</tag-class>
 </tag>
</taglib>

```

**web.xml**

Same as previous

**output :**

Nested Level : 0  
Nested Level : 1

**Nested Level : 2****Nested Level : 3****Example :****menu.jsp**

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<mine:menu>
<mine:menuItem item="chicken"/>
<mine:menuItem item="mutton"/>
<mine:menuItem item="fish"/>
<mine:menuItem item="noodles"/>
</mine:menu>
```

**Write a Tag handler class(THC) that takes the menu items from its child and print it**

**MenuTag.java**

```
package com.tag;

import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class MenuTag extends TagSupport {
 private static final long serialVersionUID = 1L;

 ArrayList<String> menuList=null;

 public void addItem(String item){
 menuList.add(item);
 }

 @Override
 public int doStartTag() throws JspException{
 menuList=new ArrayList<String>();
 return EVAL_BODY_INCLUDE;
 }

 @Override
 public int doEndTag() throws JspException{
```

```

JspWriter out=null;
try{
 out=pageContext.getOut();
 out.println("The menu items are : "+menuList);
}
catch(IOException e){
 e.printStackTrace();
}
return EVAL_PAGE;
}
}

```

**Write a program to accept the menu items and add to menu list of its parent**

#### MenuItemTag.java

```

package com.tag;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;

public class MenuItemTag extends TagSupport {
 private static final long serialVersionUID = 1L;

 public String item=null;

 public void setItem(String item){
 this.item=item;
 }

 @Override
 public int doStartTag() throws JspException{
 MenuTag menuTag=(MenuTag)getParent();
 menuTag.addItem(item);
 return SKIP_BODY;
 }
}

```

#### myTld.tld

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>menu</name>

```

```
<tag-class>com.tag.MenuTag</tag-class>
</tag>
<tag>
<name>menuItem</name>
<tag-class>com.tag.MenuItemTag</tag-class>
<attribute>
<name>item</name>
<required>true</required>
</attribute>
</tag>
</taglib>
```

**web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
 http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

<jsp-config>
<taglib>
<taglib-uri>http://jobs4times.com/tags</taglib-uri>
<taglib-location>/WEB-INF/myTld.tld</taglib-location>
</taglib>
</jsp-config>
</web-app>
```

**output :**

The menu items are : [chicken, mutton, fish, noodles]

**Getting Arbitrary Ancestor Object :**

We can get immediate parent by using `getParent()`, `TagSupport` class defines the following method to get an arbitrary ancestor class object

[public static Tag findAncestorWithClass\(Tag t, Class c\);](#)

- Accessing jsp implicit objects and attributes in tag handler class with in the tag handler class we can get jsp implicit objects by using `PageContext` object.
- Tag handler class can get `PageContext` object an argument to `setPageContext()`.

`PageCotext` class defines the following methods to get jsp implicit objects :

request	---->	getRequest()
response	---->	getResponse()

config	---->	getServletConfig()
application	---->	getServletContext()
session	---->	getSession()
out	---->	getOut()
page	---->	getPage()
exception	---->	getException()

All the methods we have to call on PageContext object

**Note :** exception implicit object is available only in error pages , if the enclosing jsp is not error page then getException() returns null.

#### Accessing attributes by using PageContext object :

PageContext class defines the following methods to perform attribute management in any scope (we can perform attribute management in tag handler classes also )

1. public void setAttribute(String name, Object value);
2. public void setAttribute(String name, Object value, int scope);
3. public Object getAttribute(String name);
4. public Object getAttribute(String name, int scope);
5. public void removeAttribute(String name);
6. public void removeAttribute(String name, int scope);
7. public Object findAttribute(String name);
8. public Enumeration getAttributeNamesInScope(int scope);

#### myJsp.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
```

JSP Implicit Objects from Custom Tags<br>

```
<mine:jsplImplicitObjects/>
```

#### JspImplicitObj.java

```
package com.tag;
```

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
```

```
public class JsplImplicitObj extends TagSupport {
 private static final long serialVersionUID = 1L;
 private String mail="Ask";

 @Override
 public int doStartTag() throws JspException{
 JspWriter out=null;
 try{
 ServletRequest request=pageContext.getRequest();
 ServletResponse response=pageContext.getResponse();
 ServletConfig config=pageContext.getServletConfig();
 ServletContext application=pageContext.getServletContext();
 HttpServlet page=(HttpServlet)pageContext.getPage();
 HttpSession session=pageContext.getSession();
 out=pageContext.getOut();
 Throwable exception=pageContext.getException();

 out.println("
The Server details:"+request.getServerName()+" "
 +request.getServerPort());
 out.println("
The content-type:"+response.getContentType());
 out.println("
The Session Id:"+session.getId());
 out.println("
The Current Servlet:"+page);
 out.println("
The Context Parameter value:"
 +application.getInitParameter("uname"));

 if(exception==null){
 out.println("
exception is null because there is no exception code");
 }
 else{
 out.println("
The exception value :" +exception);
 }

 String formParamValue=config.getInitParameter("mail");

 if(formParamValue==null || mail==null){

 if(formParamValue==null){
 out.println("
If you want to access form parameter, "
 + "you send param name in the form of query String");
 }
 else{
 out.println("
The form parameter value:" +formParamValue);
 }

 if(mail==null){
```

```

 out.println("
If want to get the init parameter values"
 + " then you can send a request using url-pattern");
 }
 else{
 out.println("
The init parameter value:"+mail);
 }

}
else{
 out.println("
The form param value:"+formParamValue);
 out.println("
The init param value:"+mail);
}
}
catch(Exception e){
 e.printStackTrace();
}

return EVAL_BODY_INCLUDE;
}

}

```

**myTld.tld**

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>jspImplicitObjects</name>
 <tag-class>com.tag.JspImplicitObj</tag-class>
 </tag>
</taglib>

```

**web.xml**

```

<web-app>
 <jsp-config>
 <taglib>
 <taglib-uri>http://jobs4times.com/tags</taglib-uri>
 <taglib-location>/WEB-INF/myTld.tld</taglib-location>
 </taglib>
 </jsp-config>

 <context-param>
 <param-name>uname</param-name>
 <param-value>SaiCharan</param-value>
 </context-param>

```

```

<servlet>
<servlet-name>implicitObj</servlet-name>
<jsp-file>/myJsp.jsp</jsp-file>

<init-param>
<param-name>mail</param-name>
<param-value>jobs4times@gmail.com</param-value>
</init-param>
</servlet>

<servlet-mapping>
<servlet-name>implicitObj</servlet-name>
<url-pattern>/test</url-pattern>
</servlet-mapping>

</web-app>

```

**http://localhost:8080/jstl/test**

**output :**

#### JSP Implicit Objects from Custom Tags

The Server details:ashok 8080  
The content-type:text/html  
The Session Id:089ea14fea88884de1b6f490bafe  
The Current Servlet:org.apache.jsp.test\_jsp@63c3024d  
The Context Parameter value:SaiCharan  
exception is null because there is no exception code  
The form param value:jobs4times@gmail.com  
The init param value:Ask

#### SimpleTag Model : (jsp 2.0v)

Implementing Custom tags by using Classic Tag Model (i.e., Tag, IterationTag, BodyTag, TagSupport, BodyTagSupport) is very complex because each tag has its own life cycle and different possible return types for every method to resolve this complexity Sun people introduced SimpleTagModel in jsp 2.0 version

In SimpleTag model we can built custom tags by using SimpleTag interface and its implementation class SimpleTagSupport.

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**  
**JAVA MEANS DURGASOFT**  
**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

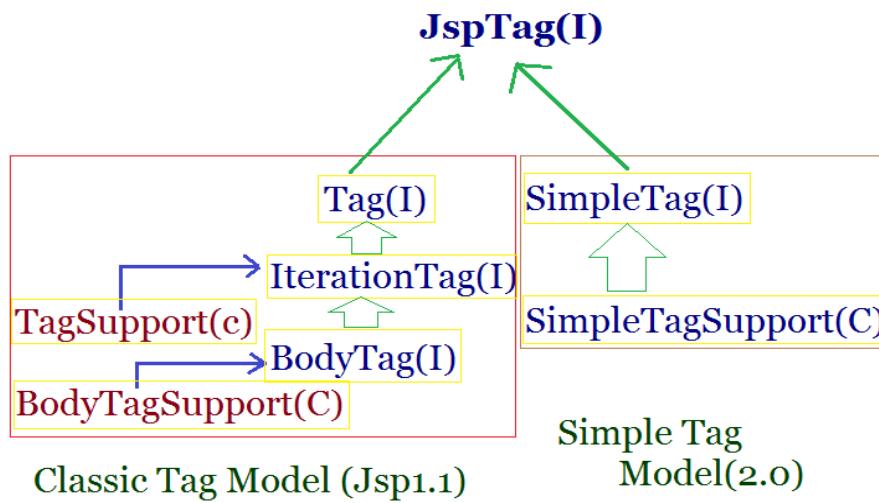


# 202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
+91 9246212143  
+91 8096969696

**212**

DURGASOFT, # 202, 2<sup>nd</sup> Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,  
**040 - 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com**



### SimpleTag interface :

It is child interface of JspTag and defines the following 5 methods

1. `public void setJspContext(JspContext context)`  
by using `setJspContext()` method we can make `JspContext` object available to Tag Handler class  
by using this object we can get all jsp implicit objects and attributes in our tag handler classes.
2. `public void setParent(JspTag parent)`  
This method will be executed iff the tag has another tag (nested tag)
3. `public void setJspBody(JspFragment jspBody)`
4. `public void doTag() throws JspException, IOException`
5. `public JspTag getParent()`

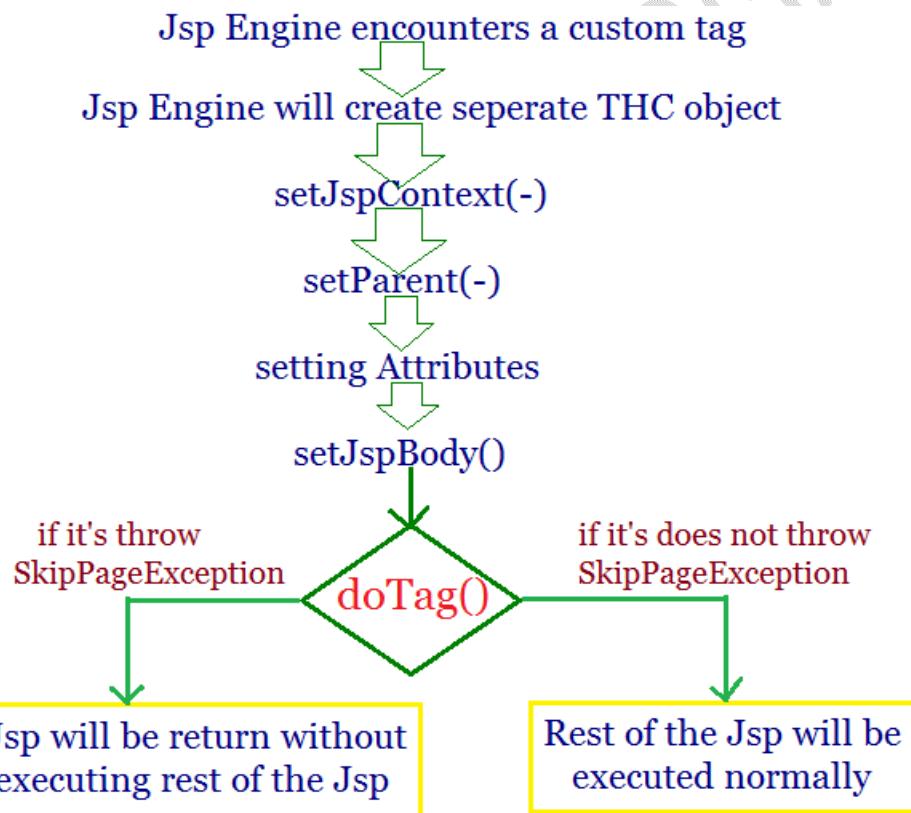
These methods are life cycle methods Jsp engine calls automatically for every custom tag invocation , but not `getParent()` method.

### Life Cycle of SimpleTag Model :

1. When ever the jsp engine encounters a custom tag in jsp it will identify corresponding tag handler class by using taglib directive and TLD file.
2. It creates a new instance of TH class by executing public no-argument constructor.
3. SimpleTag handler objects are never reused by the WC for each tag invocation in new tag handler class created.
4. Web-container executes `setJspContext()` to make `JspContext` object available to tag handler class by using this `JspContext` object tag handler class gets all jsp implicit objects and attributes.
5. Jsp engine will call `setParent()` to make parent tag object available to tag handler class , this method is useful in nested tags( `setParent()` is only called if the element is nested in another tag invocation )
6. If a custom tag invoked with attribute then for each attribute jsp engine will call corresponding setter methods to make attribute value available to tag handler class (for every attribute one instance variable and corresponding setter method should required)

7. If a custom tag invoke with body then `setJspBody()` will be executed by taking `JspFragment` object as argument, `JspFragment` object represents tag body in `SimpleTagModel` tag body should not contains scripting elements i.e., the allowed values for the body-content tag is empty, tagdependent, scriptless from Jsp2.0v onwards body-content tag is optional, in `SimpleTagModel` default value is scriptless.
8. If a custom tag is empty then `setJspBody()` won't be called , there is no default value in body-content in `SimpleTagModel`.
9. Finally jsp engine will invoke `doTag()` to required functionality this method is equivalent to `doStartTag()`, `doEndTag()`, `doAfterBody()`
10. Once `doTag()` completes tag handler class object will be destroy by the WC.

### Flow chart for SimpleTagModel :



### Internal implementation of SimpleTagSupport class :

`SimpleTagSupport.java`

```
package com.tag;
```

```
import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
```

```
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.JspTag;
import javax.servlet.jsp.tagext.SimpleTag;

public class SimpleTagSupport implements SimpleTag {
 private static final long serialVersionUID = 1L;

 @Override
 public void setJspContext(JspContext pc) {
 throw new UnsupportedOperationException("Not supported yet.");
 //To change body of generated methods, choose Tools | Templates.
 }

 @Override
 public void setParent(JspTag parent) {
 throw new UnsupportedOperationException("Not supported yet.");
 //To change body of generated methods, choose Tools | Templates.
 }

 @Override
 public JspTag getParent() {
 throw new UnsupportedOperationException("Not supported yet.");
 //To change body of generated methods, choose Tools | Templates.
 }

 @Override
 public void setJspBody(JspFragment jspBody) {
 throw new UnsupportedOperationException("Not supported yet.");
 //To change body of generated methods, choose Tools | Templates.
 }

 @Override
 public void doTag() throws JspException, IOException {
 throw new UnsupportedOperationException("Not supported yet.");
 //To change body of generated methods, choose Tools | Templates.
 }
}
```

(OR)

Sample code :

```
import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.JspTag;
```

```
import javax.servlet.jsp.tagext.SimpleTag;

public class SimpleTagSupport implements SimpleTag {
 private static final long serialVersionUID = 1L;

 private JspContext jspContext;
 private JspFragment jspBody;
 private JspTag parent;

 @Override
 public void setJspContext(JspContext jspContext) {
 this.jspContext=jspContext;
 }

 protected JspContext getJspContext(){
 return jspContext;
 }

 @Override
 public void setParent(JspTag parent) {
 this.parent=parent;
 }

 @Override
 public JspTag getParent() {
 return parent;
 }

 @Override
 public void setJspBody(JspFragment jspBody) {
 this.jspBody=jspBody;
 }

 public JspFragment getJspBody() {
 return jspBody;
 }

 @Override
 public void doTag() throws JspException, IOException {
 }

 public static final JspTag findAncestorWithClass(JspTag tag,Class c){
 //return null;
 }
}
```

**Write a demo program to SimpleTagSupport class :**

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

This is before Custom Tag

<mine:myTag>
This is the body of the tag

</mine:myTag>
This is rest of the Jsp page
```

Tag Handler Class

SimpleTagSupportDemo.java

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.SkipPageException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagSupportDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out = getJspContext().getOut();
 out.println("This is SimpleTagSupport class
");
 //throw new SkipPageException();
 }
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.SimpleTagSupportDemo</tag-class>
 <body-content>scriptless</body-content>
 </tag>
</taglib>
```

**web.xml**

```
<web-app>
<jsp-config>
<taglib>
<taglib-uri>http://jobs4times.com/tags</taglib-uri>
<taglib-location>/WEB-INF/myTld.tld</taglib-location>
</taglib>
</jsp-config>
</web-app>
```

**Note :** if doTag() is not throwing a SkipPageException then following is the output

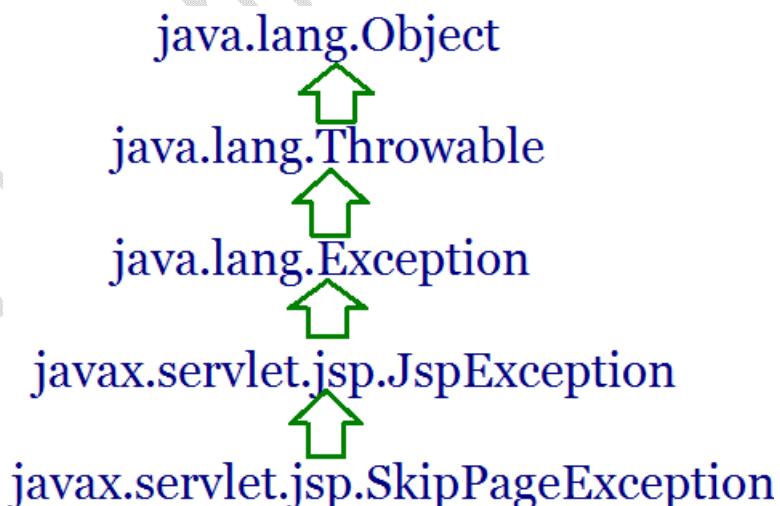
This is before Custom Tag  
This is SimpleTagSupport class  
This is rest of the Jsp page

If doTag() method throws SkipPageException()

This is before Custom Tag  
This is SimpleTagSupport class

By default the Tag Body should not be include of JSP if we want to include then we have to arrange some extra arrangement.( i.e., getterMethods, ....)

**SkipPageException hierarchy :**



**Accessing Tag Body in SimpleTagModel**

We can access tag body simple tag handler by using getJspBody() this method returns JspFragment object

public JspFragment getJspBody()

JspFragment is an abstract class at translation time, the container generate the implementation of the JspFragment abstract class capable of executing the defined fragment(tag body)

1. public JspContext getJspContext()
2. public void invoke(java.io.Writer out)
  - It causes evaluation of tag body and return to the supplied writer
  - If we pass null argument to invoke() then it will write directly to the jsp page

**index.jsp**

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<title>Tag body in Simple tag model</title>
This is before Custom Tag

<mine:myTag>
This is the body of the tag

</mine:myTag>
This is rest of the Jsp page
```

**SimpleTagSupportBodyDemo.java**

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagSupportBodyDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println("This is SimpleTagSupport class
");
 JspFragment body=getJspBody();
 body.invoke(null);
 }
}
```

**myTld.tld**

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.SimpleTagSupportBodyDemo</tag-class>
 <body-content>scriptless</body-content>
 </tag>
</taglib>
```

**web.xml****same as previous****<http://localhost:8080/jstl/index.jsp>****output :**

This is before Custom Tag  
 This is SimpleTagSupport class  
 This is the body of the tag  
 This is rest of the Jsp page

**Manipulating tag body in SimpleTagModel :****index.jsp**

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<title>Manipulating Tag body in Simple tag model</title>
This is before Custom Tag

<mine:myTag>
 This is body of the tag

</mine:myTag>
This is rest of the Jsp page
```

**ManipulateSimpleTagSupportBodyDemo.java**

```
package com.tag;

import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.SimpleTagSupport;
```

```
public class ManipulateSimpleTagSupportBodyDemo extends SimpleTagSupport{
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println("This is SimpleTagSupport class
");
 JspFragment body=getJspBody();
 StringWriter stringWriter=new StringWriter();
 body.invoke(stringWriter);
 //it returns tag body into StringWriter object
 out.println(stringWriter.toString().toUpperCase());
 }
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.ManipulateSimpleTagSupportBodyDemo</tag-class>
 <body-content>scriptless</body-content>
 </tag>
</taglib>
```

web.xml

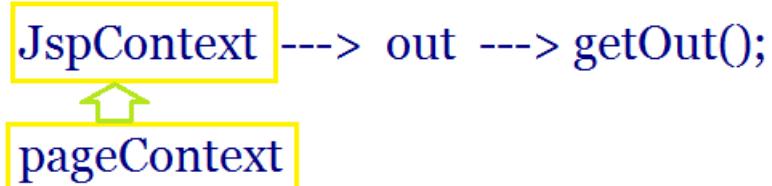
Same as previous

output :

This is before Custom Tag  
 This is SimpleTagSupport class  
 THIS IS BODY OF THE TAG  
 This is rest of the Jsp page

### Jsp implicit objects and attributes in SimpleTag Model :

JspContext class contains getOut() but not remaining jsp implicit object methods



request ---> getRequest();  
 response ---> getResponse();  
 application ---> getServletContext();  
 config ---> getServletConfig();  
 session ---> getSession();  
 page ---> getPage()  
 exception ---> getException();

In Tag Handler class :

```

package com.tag;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagSupportImplicitDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
 PageContext pageContext=(PageContext)getJspContext();
 HttpSession session=pageContext.getSession();
 JspWriter out=pageContext.getOut();
 out.println("The session id :" + session.getId() + "
");
 ServletConfig config=pageContext.getServletConfig();
 }
}

```

```

ServletContext application=pageContext.getServletContext();
ServletRequest request=pageContext.getRequest();
ServletResponse response=pageContext.getResponse();
Object page=pageContext.getPage();
Throwable exception=pageContext.getException();
}

}

```

**index.jsp**

```

<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
<%@page isELIgnored="false" %>

<title>Setting attributes in Simple tag handler </title>
This is before Tag invocation

<mine:myTag>
${movie}

</mine:myTag>
This is after tag invocation

```

**AttributeSimpleTagDemo.java**

```

package com.tag;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class AttributeSimpleTagDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 String movies[]={ "movieA", "MovieB", "MovieC" };
 @Override
 public void doTag() throws JspException, IOException {
 System.out.println("Welcome to Attribute Mgt");
 JspWriter out=getJspContext().getOut();
 for(int i=0;i<movies.length;i++){
 getJspContext().setAttribute("movie", movies[i]);
 }
 }
}

```

```

 getJspBody().invoke(null);
}
}
}

```

Each loop of the tag handler resets the "movie" attribute value and calls `getJspBody().invoke()` again.

#### myTld.tld

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.AttributeSimpleTagDemo</tag-class>
 <body-content>scriptless</body-content>
 </tag>
</taglib>

```

#### web.xml

Same as previous

#### output :

This is before Tag invocation

movieA

MovieB

MovieC

This is after tag invocation

#### What happens when the tag is invoked from on included page ?

#### index.jsp

```

<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

<title>includes test Jsp </title>
This is before Jsp inclusion

<jsp:include page="test.jsp"/>

This is after Jsp inclusion

```

#### test.jsp

```

<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

```

```
<title>Included file</title>
This is before Custom Tag

<mine:myTag>
 This is the body of the tag

</mine:myTag>
This is rest of the Jsp page
```

**SimpleTagDemo.java**

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.SkipPageException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleTagDemo extends SimpleTagSupport {
 private static final long serialVersionUID = 1L;

 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println("Hello, this is from tag handler
");
 //throw new SkipPageException();
 }
}
```

**myTld.tld**

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.SimpleTagDemo</tag-class>
 <body-content>scriptless</body-content>
 </tag>
</taglib>
```

**web.xml**

Same as previous

**output :**

This is before Jsp inclusion  
 This is before Custom Tag  
 Hello, this is from tag handler

This is rest of the Jsp page  
This is after Jsp inclusion

remove comment on above THC

output :

This is before Jsp inclusion  
This is before Custom Tag  
Hello, this is from tag handler

This is after Jsp inclusion

#### What is the difference between ClassicTagModel and SimpleTagModel with respect to tag body :

- In ClassicTagModel the tag body can contains scripting elements hence the allowed values for body-content types are empty, tagdependent, scriptless, jsp and default value is jsp.
- But in SimpleTagModel the tag body should not contains scripting elements hence the allowed values for body-content types are empty, tagdependent, scriptless and default value is scriptless.



#### Key differences between ClassictagModel and SimpleTagModel :

Property	ClassictagModel	SimpleTagModel
key interfaces	Tag(I) IterationTag(I) BodyTag(I)	SimpleTag(I)
supporting implementation classes	TagSupport(C) BodyTagSupport(C)	SimpleTagSupport(C)
key life cycle methods that we	doStartTag() doEndTag()	doTag()

have to implement	doAfterBody()	
how to write response to jsp output stream	pageContext().getOut().println() we should enclose this statement by using try, catch	getPageContext(). getOut().println() it is not required enclose try, catch
how to access jsp implicit objects and attributes	by using PageContext pageContext.getOut();	by using JspContext
how to include tag body in the result	in the case of Tag, IterationTag interfaces doStartTag() should return EVAL_BODY_INCLUDE but in BodyTag interface doStartTag() should return EVAL_BODY_BUFFERED	getJspBody().invoke(null);
how to stop current jsp execution	doEndTag() should return SKIP_PAGE	in side doTag() we should throw SkipPageException

### DynamicAttributes :

- In general tags can contain attributes for which we have to declare in the tld file by using attribute tag for these attributes we have to maintain one instance variable and corresponding setter methods in tag handler class such type of attributes are called static attributes.
- But we can use attributes even though tld file doesn't contain any <attribute> tag declaration such type of attributes are called dynamic attributes.
- Dynamic attributes concept applicable for both classic and simple tag models
- Dynamic attributes concept introduced jsp2.0v

To support Dynamic attribute concept we have to do the following things :

1. In the tld file we have to declare <dynamic-attributes> tag

```
<taglib version="2.1">
<tlib-version>1.2</tlib-version>
<uri>www.jobs4times.com</uri>
<tag>
.....
<dynamic-attributes>true</dynamic-attributes>
.....
</tag>
</taglib>
```

2. The corresponding tag handler class should implements DynamicAttributes interface, this interface introduced jsp2.0v and contains only one method.

```
public void setDynamicAttribute(String nameSpace, String name, Object value)
throws JspException{}
```

For every DynamicAttribute this method will be executed

test.jsp

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>
```

This is before Custom Tag <br>

```
<mine:myTag name="Nandu" wife="Renu" brother="Tinku" habbits="sleep,lunch"/>
```

This is rest of the Jsp page

DynamicAttributesDemo.java

```
package com.tag;

import java.io.IOException;
import java.util.HashMap;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.DynamicAttributes;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class DynamicAttributesDemo extends SimpleTagSupport
 implements DynamicAttributes{
 private static final long serialVersionUID = 1L;

 HashMap h=new HashMap();

 @Override
 public void setDynamicAttribute(String nameSpace,
 String name, Object value) throws JspException{
 h.put(name, value);
 }
 @Override
 public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println(h + "
");
 }
}
```

myTld.tld

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.DynamicAttributesDemo</tag-class>
 <body-content>scriptless</body-content>
 <dynamic-attributes>true</dynamic-attributes>
 </tag>
</taglib>
```

```
</tag>
</taglib>
```

**web.xml**

Same as previous

**output :**

```
This is before Custom Tag
{wife=Renu, name=Nandu, brother=Tinku, habbits=sleep,lunch}
This is rest of the Jsp page
```

#### Combination of static & dynamic attributes :

**index.jsp**

```
<%@taglib uri="http://jobs4times.com/tags" prefix="mine" %>

This is before Custom Tag

<mine:myTag num="2" min="5" max="10" pow="3"/>
This is rest of the Jsp page
```

**DynamicAttributesDemo.java**

```
package com.tag;

import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.DynamicAttributes;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class DynamicAttributesDemo extends SimpleTagSupport
 implements DynamicAttributes{
 private static final long serialVersionUID = 1L;

 String output="";
 private int num;

 public void setNum(int num){
 this.num=num;
 output=output+" The Given No is : "+num;
 }

 @Override
 public void setDynamicAttribute(String nameSpace,
 String name, Object value) throws JspException{
 int n=Integer.parseInt((String) value);
```

```

if(name=="min"){
 output=output+"
minimum value of "+num+","+n+" is : "+Math.min(num, n);
}
else if(name=="max"){
 output=output+"
maximum value of "+num+","+n+" is : "+Math.max(num, n);
}
else if(name=="pow"){
 output=output+"
power value of "+num+","+n+" is "+Math.pow(num, n);
}

}

@Override
public void doTag() throws JspException, IOException {
 JspWriter out=getJspContext().getOut();
 out.println(output + "
");
}
}

```

**myTld.tld**

```

<?xml version="1.0" encoding="UTF-8"?>
<taglib version="2.1" xmlns="http://java.sun.com/xml/ns/javaee">
 <tlib-version>1.2</tlib-version>
 <uri>www.jobs4times.com</uri>
 <tag>
 <name>myTag</name>
 <tag-class>com.tag.DynamicAttributesDemo</tag-class>
 <body-content>scriptless</body-content>
 <dynamic-attributes>true</dynamic-attributes>
 <attribute>
 <name>num</name>
 <required>true</required>
 <rteprvalue>true</rteprvalue>
 </attribute>
 </tag>
</taglib>

```

**web.xml**

Same as previous

**output :**

This is before Custom Tag  
The Given No is : 2  
minimum value of 2,5 is : 2  
maximum value of 2,10 is : 10  
power value of 2,3 is 8.0  
This is rest of the Jsp page

**TagFiles (jsp2.0v) :****objectives:**

1. Describes the semantics of tag file model
2. Describes application structure of tag file
3. Write a tag file and explain the constraints on jsp content in tag body
4. TagFiles concept introduced in Jsp2.0v
5. Tagfile is a jsp page or jsp document designed to be used as a custom action
6. The main advantage of tag files when compared with classic and simple tag model is we can built custom tag very easily and we are not required to write tld file and the corresponding tag handler classes
7. The main limitation of tag files is won't suggestible will doing must processing

**Building & Using a Simple tag file :**

1. Write a Jsp page or Jsp document and save it with .tag extension.
2. Place this tag file inside /WEB-INF/tags folder
3. Write a taglib directive in Jsp with "tagdir" attribute

**index.jsp**

```
<%@taglib prefix="mine" tagdir="/WEB-INF/tags"%>
<%@taglib prefix="wish" tagdir="/WEB-INF/tags/myTags"%>

<html>
<title>Tag Files Demo</title>
<body>
<h1>THis is Tag File Demo</h1>
<mine:myTag />
<wish:welcome/>
</body>
</html>
```

**welcome.tag**

```
<h1>This is Welcome tag file</h1>
```

**myTag.tag**

```
<h1>This is MyTag File</h1>
```

**output:**

This is Tag File Demo

This is MyTag File

This is Welcome tag file

Note : Internally tag files will be converted into simple Tag Handlers , which are generated by JSP Engine, and it is available in work folder.

**Declare tag files with attributes :**

We can declare attributes for Tag files by using attribute directive.

```
<%@attribute name="title" required="true" rtexprvalue="true"%>
```

attribute directive is not part of JSP, It's part of Tag files

test.jsp

```
<%@taglib prefix="mine" tagdir="/WEB-INF/tags"%>

<title>Tag Files Demo</title>
<h1>THis is Tag File Demo</h1>
<mine:myTag title="Ashok"/>
```

myTag.tag

```
<%@attribute name="title" required="true" rtexprvalue="true"%>

<h1>This is MyTag File</h1>
title : ${title}
```

output :

```
This is Tag File Demo
This is MyTag File
title : Ashok
```

**Declaring body-content for tag file :**

Tag file invocation can contain body but we are not use scripting elements i.e., inside tag body doesn't contain scriptlets, expressions, declarations but it contains EL-expressions ans Jsp standard actions.

```
<%@tag body-content="tagdependent"%>
```

- tag directive is not part of JSp, it is part of Tag files.
- body-content allowed values are empty, tagdependent, scriptless.
- The default value is scriptless.

index.jsp

```
<%@taglib prefix="wish" tagdir="/WEB-INF/tags/myTags"%>
```

This is Tag Files with attributes and tag body <br>

```
<wish:welcome fontColor="#660099">
```

This is Tag Body

```
</wish:welcome>
```

welcome.tag

```
<%@attribute name="fontColor" required="true"%>
```

```
<%@tag body-content="tagdependent"%>

<jsp:doBody/>

```

**output :**

This is Tag Files with attributes and tag body

This is Tag Body //colour

With in the tag file, we can access tag body by using <jsp:doBody/>

Classic Tag Model	EVAL_BODY_INCLUDE, EVAL_BODY_BUFFERED
Simple Tag Model	getJspBody().invoke(null)
Tag Files	<jsp:doBody/>

- Either directly or indirectly inside WEB-INF/tags folder
- Either directly or indirectly inside META-INF/tags present in jar file WEB-INF/libfolder.

**Note :** If we deploy Tag file in some jar file compulsory we have to write TLD file , we have to place it inside META-INF folder.

**myTld.tld (in META-INF)**

```
<taglib version="2.1">
<tlib-version>1.2</tlib-version>
<uri>www.jobs4times.com</uri>
<tag-file>
<name>myTag</name>
<path>/META-INF/tags/myTag.tag</path>
</tag-file>
</taglib>
```

**Ex : test.jsp**

```
<%@taglib prefix="wish" uri="jobs4times"%>
```

Tag Files deployed in the form of Jar files <br><wish:welcome />

welcome.tag

This is Welcome tag file  
deploying in some jar file

**myTld.tld**

```
<taglib version="2.1">
<tlib-version>1.2</tlib-version>
<uri>jobs4times</uri>
```

```
<tag-file>
<name>welcome</name>
<path>/META-INF/tags/welcome.tag</path>
</tag-file>
</taglib>
```

### TagFiles with DynamicAttributes :

- TagFiles can also include DynamicAttributes the mechanism is basically same ClassicTagModel and SimpleTagModel but with TagFiles.
- The jsp engine provide the map object for you, you can inspect and iterate over the map of attribute key-value pair using for-each JSTL .

```
<%@tag body-content="scriptless" dynamic-attributes="${mapObj}"%>
```

The value of dynamic attributes is page scoped variable that holds map or Hashmap reference.

```
<c:forEach items="mapObj" var="x">
 ${x.key} and ${x.value}
</c:forEach>
<tag>
<name>myTag</name>
<tag-class>com.tag.SimpleTagDemo</tag-class>
<body-content>scriptless</body-content>
</tag>
<mine:myTag>
 ${2*3}
</mine:myTag>
```

output :

6

**LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...**

# **JAVA MEANS DURGASOFT**

**INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
 SOFTWARE SOLUTIONS

#202 2<sup>nd</sup> FLOOR  
[www.durgasoft.com](http://www.durgasoft.com)

040-64512786  
 +91 9246212143  
 +91 8096969696

**If it works, which of the following Simple tag Model in tag handler life cycle methods will be executed?**

1. void doTag()
2. void setParent()
3. void setJspContext()
4. JspTag getParent()
5. void setJspBody()

Ans : 1, 3, 5

- The setParent() is called only when tag is involved from with in another tag, since this tag was not nested setParent() was not executed.
- In the case of Classic Tag Model whether the tag contains nested or not in all cases setParent() will be executed i.e., setParent() is part of lifecycle method.

#### Disabling scripting language in web.xml

- we can disable scripting language <scripting-invalid> tag
- If we are disabling EL that time jsp engine will consider EL syntax as template text.

```
<web-app>
<jsp-config>
<jsp-property-group>
<url-pattern>*.jsp</url-pattern>
<scripting-invalid>true</scripting-invalid>
</jsp-property-group>
</jsp-config>
</web-app>
```

Once the <scripting-invalid> if we are not allowed to use scripting elements in jsp violation leads translation time error we will get.

#### Disabling expression language globally

We can disable EL for a particular jsp by using isELIgnored attribute of page directive.

```
<%@page isELIgnored="true"%>
```

We can disable expression language at application level as follows.

```
<web-app>
<jsp-config>
<jsp-property-group>
<url-pattern>*.jsp</url-pattern>
<el-ignored>true</el-ignored>
</jsp-property-group>
</jsp-config>
</web-app>
```

**LEARN FROM EXPERTS ...**

**COMPLETE JAVA**  
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...

**COMPLETE .NET**  
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF

**TESTING TOOLS**  
MANUAL + SELENIUM

**ORACLE | D2K**

**MSBI | SHARE POINT**

**HADOOP | ANDROID**

**C, C++, DS, UNIX**

**CRT & APTITUDE TRAINING**

AN ISO 9001:2008 CERTIFIED  
**DURGA**  
Software Solutions® # 202, 2nd Floor, HUDA Maitrivanam,  
Ameerpet, Hyd. Ph: 040-64512786,  
**9246212143, 8096969696**

**www.durgasoft.com**