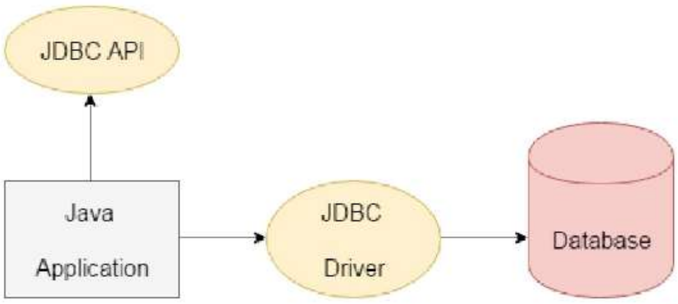
JDBC

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

JDBC Driver

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

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

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

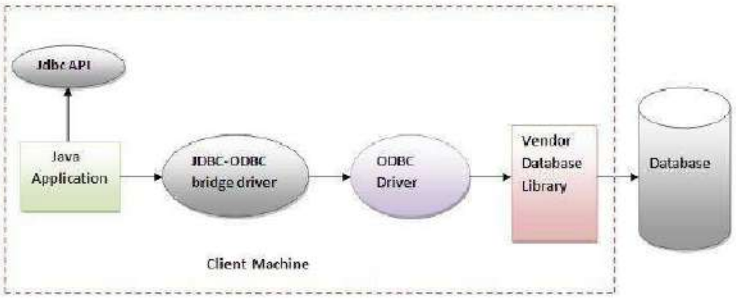


Figure- JDBC-ODBC Bridge Driver

Advantages:

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

Disadvantages:

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

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

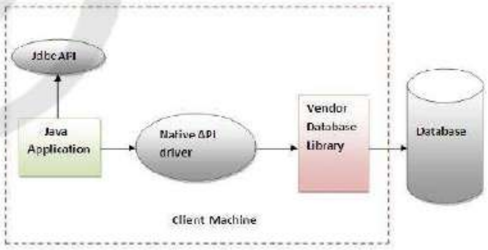


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

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

3) Network Protocol driver

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

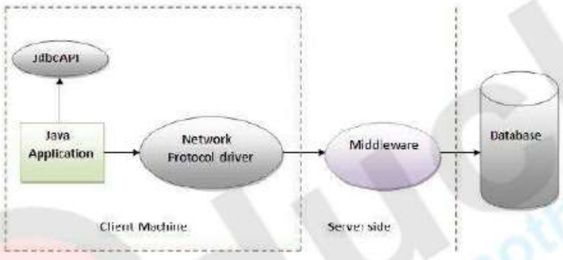


Figure - Network Protocol Driver

Advantage:

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

Disadvantages:

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

4) Thin driver

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

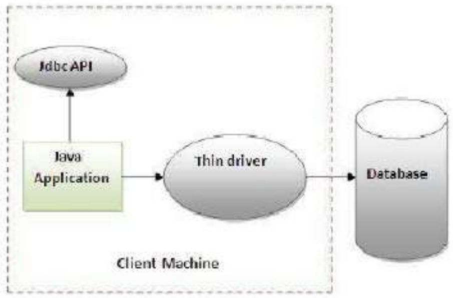


Figure- Thin Driver

Advantage:

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

Disadvantage:

- Drivers depends on the Database.

Steps to connect to the database in java

There are 6 steps to connect any java application with the database in java using JDBC. They are as follows:

- Import SQL Package
- Register the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

1) Import SQL Package

To create JDBC application, you must have to import SQL package in your program.

Eg. import java.sql.*;

2) Register the driver class

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

Syntax of forName() method:

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

Example to register the OracleDriver class

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

3) Create the connection object

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

Syntax of getConnection() method:

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

Example to establish connection with the Oracle database:

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

4) Create the Statement object

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

Syntax of createStatement() method:

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

Example to create the statement object:

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

5) Execute the query

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

Syntax of executeQuery() method:

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

Example to execute query:

```
ResultSet rs=stmt.executeQuery("select * from emp");

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

6) Close the connection object

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

Syntax of close() method:

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

Example to close connection:

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

Example to connect to the Oracle database in java

For connecting java application with the oracle database, you need to follow 5 steps to perform database connectivity. In this example we are using Oracle10g as the database. So we need to know following information for the oracle database:

Driver class: The driver class for the oracle database is oracle.jdbc.driver.OracleDriver.

Connection URL: The connection URL for the oracle10G database is jdbc:oracle:thin:@localhost:1521:xe where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.

Username: The default username for the oracle database is system.

Password: Password is given by the user at the time of installing the oracle database.

Let's first create a table in oracle database.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

Example to Connect Java Application with Oracle database

In this example, system is the username and oracle is the password of the Oracle database.

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");
//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object
con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

Two ways to load the jar file:

1. paste the ojdbc14.jar file in jre/lib/ext folder
2. set classpath

Classes and Interfaces in JDBC

DriverManager class:

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Commonly used methods of DriverManager class:

1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

Connection interface:

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.
2) public Statement createStatement(int resultSetType,int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.
4) public void commit(): saves the changes made since the previous commit/rollback permanent.
5) public void rollback(): Drops all changes made since the previous commit/rollback.
6) public void close(): closes the connection and Releases a JDBC resources immediately.

Statement interface:

The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.
2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.
3) public boolean execute(String sql): is used to execute queries that may return multiple results.
4) public int[] executeBatch(): is used to execute batch of commands.

Example of Statement interface:

Let's see the simple example of Statement interface to insert, update and delete the record.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","s
ystem","oracle");
Statement stmt=con.createStatement();

//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where i
d=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");
System.out.println(result+" records affected");
con.close();
}}
```

PreparedStatement interface:

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Let's see the example of parameterized query:

```
String sql="insert into emp values(?,?,?)";
```

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Syntax:

public PreparedStatement prepareStatement(String query)throws SQLException{}

Methods of PreparedStatement Interface	
Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Example of PreparedStatement interface that inserts the record:

First of all create table as given below:

```
create table emp(id number(10),name varchar2(50));
```

Now insert records in this table by the code given below:

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

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

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();
System.out.println(i+" records inserted");

con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

Example of PreparedStatement interface that updates the record:

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);

int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

Example of PreparedStatement interface that deletes the record:

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);

int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

Example of PreparedStatement interface that retrieve the records of a table:

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

Java CallableStatement Interface:

CallableStatement interface is used to call the stored procedures and functions.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

Difference between Stored Procedure and Function	
Stored Procedure	Function
is used to perform business logic.	is used to perform calculation.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one values.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameter.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

How to get the instance of CallableStatement?

The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:

```
public CallableStatement prepareCall("{ call procedurename(?,?...?)}");
```

The example to get the instance of CallableStatement is given below:

```
CallableStatement stmt=con.prepareCall("{call myprocedure(?,?)}");
```

It calls the procedure myprocedure that receives 2 arguments.

Full example to call the stored procedure using JDBC

To call the stored procedure, you need to create it in the database. Here, we are assuming that stored procedure looks like this.

```
create or replace procedure "INSERTR"
(id IN NUMBER,
name IN VARCHAR2)
is
begin
insert into user420 values(id,name);
end;
/
```

The table structure is given below:

```
create table user420(id number(10), name varchar2(200));
```

In this example, we are going to call the stored procedure INSERTR that receives id and name as the parameter and inserts it into the table user420. Note that you need to create the user420 table as well to run this application.

```
import java.sql.*;
public class Proc {
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","oracle");

CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
stmt.setInt(1,1011);
stmt.setString(2,"Amit");
stmt.execute();

System.out.println("success");
}
}
```

Example to call the function using JDBC:

In this example, we are calling the sum4 function that receives two input and returns the sum of the given number. Here, we have used the registerOutParameter method of CallableStatement interface, that registers the output parameter with its corresponding type. It provides information to the CallableStatement about the type of result being displayed.

The Types class defines many constants such as INTEGER, VARCHAR, FLOAT, DOUBLE, BLOB, CLOB etc.

Let's create the simple function in the database first.

```
create or replace function sum4
(n1 in number,n2 in number)
return number
is
temp number(8);
begin
temp :=n1+n2;
return temp;
end;
/
```

In this example, we are going to call the function sum4 to calculate the addition of two numbers.

```
import java.sql.*;

public class FuncSum {
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","oracle");

CallableStatement stmt=con.prepareCall("{?= call sum4(?,?)}");
stmt.setInt(2,10);
stmt.setInt(3,43);
stmt.registerOutParameter(1,Types.INTEGER);
stmt.execute();

System.out.println(stmt.getInt(1));

}
}
```

ResultSet interface:

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
```

Methods of ResultSet Interface	
1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.

7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

Example of Scrollable ResultSet:

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

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

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","s
ystem","oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");

//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```

Java ResultSetMetaData Interface:

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Methods of ReseulSetMetaData Interface	
Method	Description
public int getColumnCount()throws SQLException	it returns the total number of columns in the ResultSet object.
public String getColumnName(int index)throws SQLException	it returns the column name of the specified column index.
public String getColumnTypeName(int index)throws SQLException	it returns the column type name for the specified index.
public String getTableName(int index)throws SQLException	it returns the table name for the specified column index.

Example of ResultSetMetaData interface :

```
import java.sql.*;
class Rsmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

PreparedStatement ps=con.prepareStatement("select * from emp");
ResultSet rs=ps.executeQuery();
ResultSetMetaData rsmd=rs.getMetaData();

System.out.println("Total columns: "+rsmd.getColumnCount());
System.out.println("Column Name of 1st column: "+rsmd.getColumnName(1));
System.out.println("Column Type Name of 1st column: "+rsmd.getColumnTypeName(1)
);

con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

SERVLETS

Applet vs. Servlets in java		
BASIS FOR COMPARISON	APPLET	SERVLET
Execution	Applet is always executed on the client side.	Servlet is always executed on the server side.

Packages	import java.applet.*;
	import java.awt.*;
		import javax.servlet.*;
		import java.servlet.http.*;

Lifecycle methods	init(), stop(), paint(), start(), destroy().	init( ), service( ), and destroy( ).
User interface	Applets use user interface classes like AWT and Swing.	No User interface required.
Requirement	Requires java compatible browser for execution.	It processes the input from client side and generates the response in terms of the HTML page, Javascript, Applets.
Security	More prone to risk as it is on the client machine.	It is under the server security.

Servlet Technology:

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

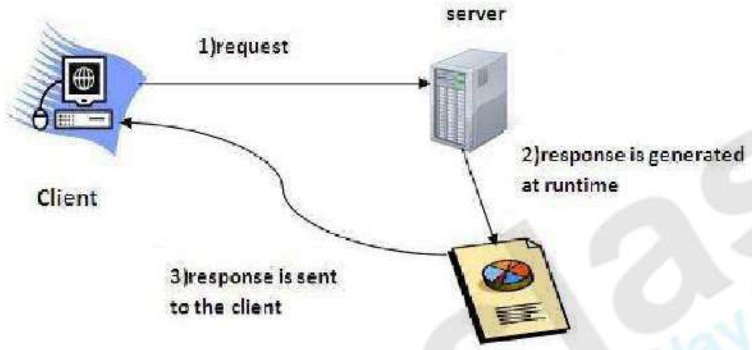
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

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

What is a Servlet?

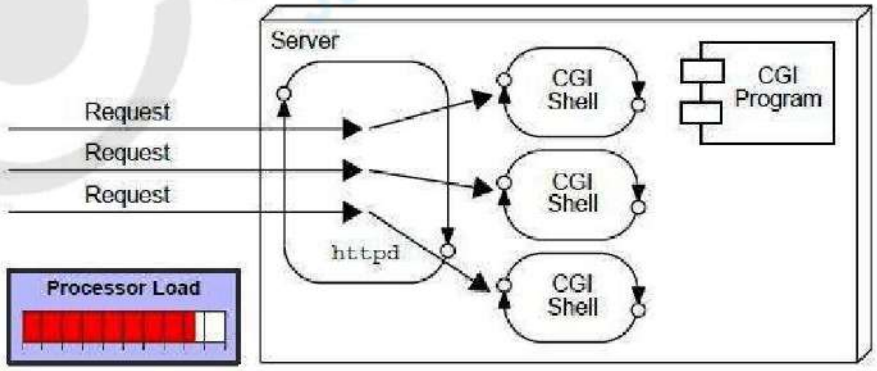
Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.


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

CGI(Commmon Gateway Interface):

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

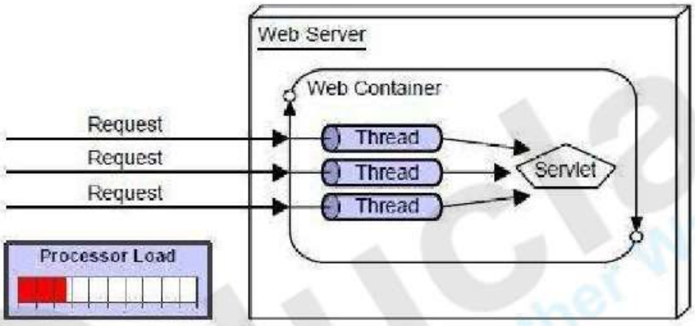


Disadvantages of CGI:

There are many problems in CGI technology:

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

Advantage of Servlet:



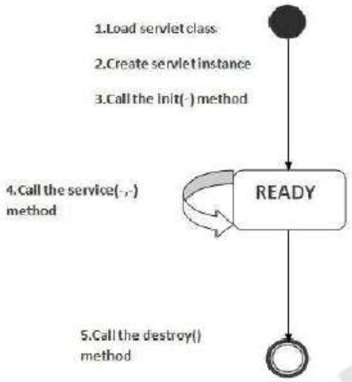
There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits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

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

Life Cycle of a Servlet (Servlet Life Cycle):

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

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


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

As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded:

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

2) Servlet instance is created:

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

3) init method is invoked:

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

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

4) service method is invoked:

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

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

5) destroy method is invoked:

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

```
public void destroy()
```

HTTP Requests

The request sends by the computer to a web server that contains all sorts of potentially interesting information is known as HTTP requests.

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

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)
- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header

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

The HTTP request methods are:

HTTP Request	Description
GET	Asks to get the resource at the requested URL.
POST	Asks the server to accept the body info attached. It is like GET request with extra info sent with the request.
HEAD	Asks for only the header part of whatever a GET would return. Just like GET but with no body.
TRACE	Asks for the loopback of the request message, for testing or troubleshooting.
PUT	Says to put the enclosed info (the body) at the requested URL.
DELETE	Says to delete the resource at the requested URL.
OPTIONS	Asks for a list of the HTTP methods to which the thing at the request URL can respond

Difference between Get and Post requests	
GET	POST
1) In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
2) Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar.
3) Get request can be bookmarked.	Post request cannot be bookmarked.
4) Get request is idempotent . It means second request will be ignored until response of first request is delivered	Post request is non-idempotent.
5) Get request is more efficient and used more than Post.	Post request is less efficient and used less than get.
6) It requests the data from a specified resource	It submits the processed data to a specified resource

Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The javax.servlet.http package contains interfaces and classes that are responsible for http requests only.

Let's see what are the classes and interfaces of javax.servlet package.

Interfaces in javax.servlet package	Classes in javax.servlet package	Interfaces in javax.servlet.http package	Classes in javax.servlet.http package

Servlet ServletRequest ServletResponse RequestDispatcher ServletConfig ServletContext	GenericServlet ServletInputStream ServletOutputStream ServletException UnavailableException	HttpServletRequest HttpServletResponse HttpSession HttpSessionListener	HttpServlet Cookie HttpServletRequest Wrapper HttpServletResponse Wrapper HttpSessionEvent

Servlet Interface

1. Servlet Interface
2. Methods of Servlet interface

Servlet interface provides common behaviour to all the servlets.

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

Methods of Servlet interface:

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

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

Example:

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

public class First implements Servlet{
ServletConfig config=null;

public void init(ServletConfig config){
this.config=config;
System.out.println("servlet is initialized");
}

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

res.setContentType("text/html");

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello simple servlet</b>");
out.print("</body></html>");

}
public void destroy(){System.out.println("servlet is destroyed");}
public ServletConfig getServletConfig(){return config;}
public String getServletInfo(){return "copyright 2007-1010";}

}
```

GenericServlet Class:

GenericServlet class implements Servlet, ServletConfig and Serializableinterfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. public void init(ServletConfig config) is used to initialize the servlet.
2. public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. public ServletConfig getServletConfig() returns the object of ServletConfig.
5. public String getServletInfo() returns information about servlet such as writer, copyright, version etc.
6. public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. public ServletContext getServletContext() returns the object of ServletContext.
8. public String getInitParameter(String name) returns the parameter value for the given parameter name.
9. public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.
10. public String getServletName() returns the name of the servlet object.
11. public void log(String msg) writes the given message in the servlet log file.
12. public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

Example:

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

public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException{
```

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

PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");

}
}
```

HttpServlet class:

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

Methods of HttpServlet class:

There are many methods in HttpServlet class. They are as follows:

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

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

ServletRequest interface:

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

Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

Method	Description
public String getParameter(String name)	is used to obtain the value of a parameter by name.
public String[] getParameterValues(String name)	returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
java.util.Enumeration getParameterNames()	returns an enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or -1 if not known.
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

Example of ServletRequest to display the name of the user:

In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the getParameter method that returns the value for the given request parameter name.

index.html

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

DemoServ.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServ extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException {
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
String name=req.getParameter("name");//will return value
pw.println("Welcome "+name);
pw.close();
}}
```

RequestDispatcher interface:

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

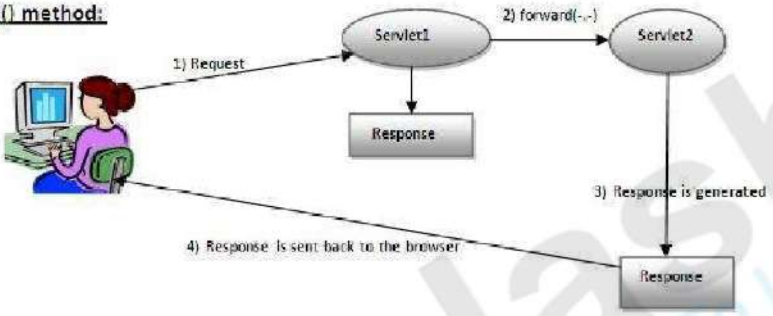
Methods of RequestDispatcher interface:

The RequestDispatcher interface provides two methods. They are:

1. public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.

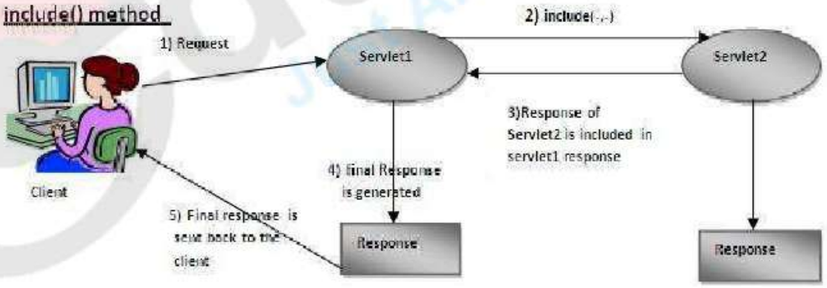
2. public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException: Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

forward() method:



As you see in the above figure, response of second servlet is sent to the client. Response of the first servlet is not displayed to the user.

include() method



2) include(-,-)
1) Request
Servlet1
Servlet2
3)Response of Servlet2 is included in servlet1 response
4) final Response is generated
Client
5) Final response is sent back to the client
Response
Response

As you can see in the above figure, response of second servlet is included in the response of the first servlet that is being sent to the client.

How to get the object of RequestDispatcher:

The getRequestDispatcher() method of ServletRequest interface returns the object of RequestDispatcher. Syntax:

Syntax of getRequestDispatcher method

```
public RequestDispatcher getRequestDispatcher(String resource);
```

Example of using getRequestDispatcher method

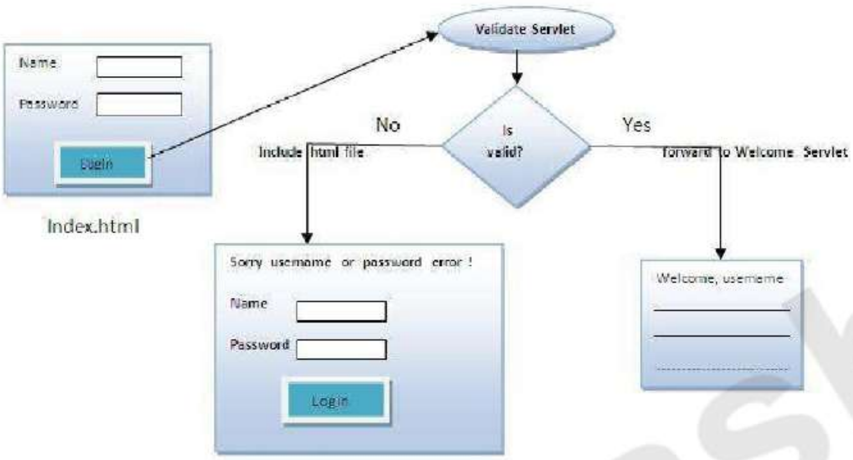
```
RequestDispatcher rd=request.getRequestDispatcher("servlet2");
//servlet2 is the url-pattern of the second servlet

rd.forward(request, response);//method may be include or forward
```

Example of RequestDispatcher interface:

In this example, we are validating the password entered by the user. If password is servlet, it will forward the request to the WelcomeServlet, otherwise will show an error message: sorry username or password error!. In this program, we are cheking for hardcoded information. But you can check it to the database also that we will see in the development chapter. In this example, we have created following files:

- index.html file: for getting input from the user.
- Login.java file: a servlet class for processing the response. If password is servet, it will forward the request to the welcome servlet.
- WelcomeServlet.java file: a servlet class for displaying the welcome message.
- web.xml file: a deployment descriptor file that contains the information about the servlet.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
Password:<input type="password" name="userPass"/><br/>
<input type="submit" value="login"/>
</form>
```

Login.java

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


public class Login extends HttpServlet {

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

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

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

    String n=request.getParameter("userName");
    String p=request.getParameter("userPass");

    if(p.equals("servlet"){
       RequestDispatcher rd=request.getRequestDispatcher("servlet2");
       rd.forward(request, response);
    }
    else{
       out.print("Sorry UserName or Password Error!");
       RequestDispatcher rd=request.getRequestDispatcher("/index.html");
       rd.include(request, response);

       }
    }

}
```

WelcomeServlet.java

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

public class WelcomeServlet extends HttpServlet {

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

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

    String n=request.getParameter("userName");
    out.print("Welcome "+n);
    }

}
```

```
web.xml
<web-app>
 <servlet>
   <servlet-name>Login</servlet-name>
   <servlet-class>Login</servlet-class>
  </servlet>
  <servlet>
   <servlet-name>WelcomeServlet</servlet-name>
   <servlet-class>WelcomeServlet</servlet-class>
  </servlet>


  <servlet-mapping>
   <servlet-name>Login</servlet-name>
   <url-pattern>/servlet1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
   <servlet-name>WelcomeServlet</servlet-name>
   <url-pattern>/servlet2</url-pattern>
  </servlet-mapping>

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

ServletConfig:

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

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

1. public String getInitParameter(String name):Returns the parameter value for the specified parameter name.
2. public Enumeration getInitParameterNames():Returns an enumeration of all the initialization parameter names.
3. public String getServletName():Returns the name of the servlet.
4. public ServletContext getServletContext():Returns an object of ServletContext.

How to get the object of ServletConfig:

getServletConfig() method of Servlet interface returns the object of ServletConfig.

Syntax of getServletConfig() method:

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

Example of getServletConfig() method:

```
ServletConfig config=getServletConfig();
//Now we can call the methods of ServletConfig interface
```

Syntax to provide the initialization parameter for a servlet:

The init-param sub-element of servlet is used to specify the initialization parameter for a servlet.

```
<web-app>
 <servlet>
  ......

  <init-param>
   <param-name>parametername</param-name>
   <param-value>parametervalue</param-value>
  </init-param>
  ......
 </servlet>
</web-app>
```

Example of ServletConfig to get initialization parameter:

In this example, we are getting the one initialization parameter from the web.xml file and printing this information in the servlet.

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

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

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

  ServletConfig config=getServletConfig();
  String driver=config.getInitParameter("driver");
  out.print("Driver is: "+driver);

  out.close();
  }

}
```

web.xml
```
<web-app>

<servlet>
<servlet-name>DemoServlet</servlet-name>
<servlet-class>DemoServlet</servlet-class>

<init-param>
<param-name>driver</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>
```

```
<servlet-mapping>
<servlet-name>DemoServlet</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

</web-app>
```

ServletContext:

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the <context-param> element.

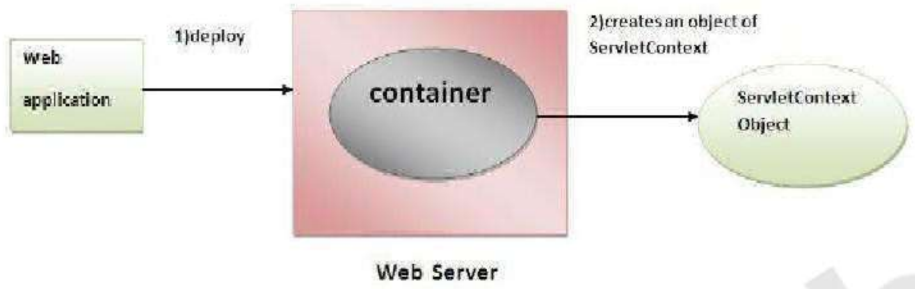
Advantage of ServletContext

Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.



Commonly used methods of ServletContext interface:

There is given some commonly used methods of ServletContext interface.

1. public String getInitParameter(String name):Returns the parameter value for the specified parameter name.
2. public Enumeration getInitParameterNames():Returns the names of the context's initialization parameters.
3. public void setAttribute(String name,Object object):sets the given object in the application scope.
4. public Object getAttribute(String name):Returns the attribute for the specified name.
5. public Enumeration getInitParameterNames():Returns the names of the context's initialization parameters as an Enumeration of String objects.
6. public void removeAttribute(String name):Removes the attribute with the given name from the servlet context.

How to get the object of ServletContext interface:

1. getServletContext() method of ServletConfig interface returns the object of ServletContext.
2. getServletContext() method of GenericServlet class returns the object of ServletContext.

Syntax of getServletContext() method:

public ServletContext getServletContext()

Example of getServletContext() method:

```
//We can get the ServletContext object from ServletConfig object
ServletContext application=getServletConfig().getServletContext();

//Another convenient way to get the ServletContext object
ServletContext application=getServletContext();
```

Syntax to provide the initialization parameter in Context scope:

The context-param element, subelement of web-app, is used to define the initialization parameter in the application scope. The param-name and param-value are the sub-elements of the context-param. The param-name element defines parameter name and and param-value defines its value.

```
<web-app>
 ......

  <context-param>
    <param-name>parametername</param-name>
    <param-value>parametervalue</param-value>
  </context-param>
 ......
</web-app>
```

Example of ServletContext to get the initialization parameter:

In this example, we are getting the initialization parameter from the web.xml file and printing the value of the initialization parameter. Notice that the object of ServletContext represents the application scope. So if we change the value of the parameter from the web.xml file, all the servlet classes will get the changed value. So we don't need to modify the servlet. So it is better to have the common information for most of the servlets in the web.xml file by context-param element. Let's see the simple example:

DemoServlet.java

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

```
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");
PrintWriter pw=res.getWriter();

//creating ServletContext object
ServletContext context=getServletContext();

//Getting the value of the initialization parameter and printing it
String driverName=context.getInitParameter("dname");
pw.println("driver name is="+driverName);

pw.close();

}}
```

web.xml

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/context</url-pattern>
</servlet-mapping>
```

```
</web-app>
```

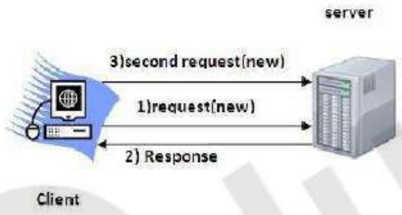
Session Tracking:

Session simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

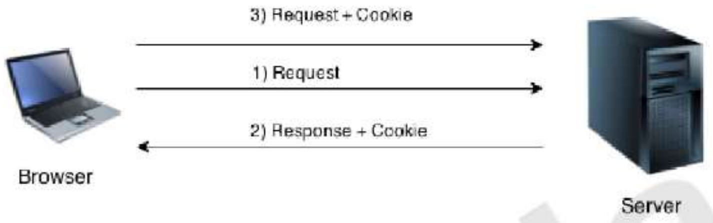
Cookies in Servlet

A cookie is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is valid for single session only. It is removed each time when user closes the browser.

Persistent cookie

It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

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

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.

2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Methods of Cookie class

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. public void addCookie(Cookie ck): method of HttpServletResponse interface is used to add cookie in response object.
2. public Cookie[] getCookies(): method of HttpServletRequest interface is used to return all the cookies from the browser.

How to create Cookie?

Let's see the simple code to create cookie.

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
```

```
response.addCookie(ck);//adding cookie in the response
```

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

Let's see the simple code to get all the cookies.

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
 out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of coo
kie }
```

Hidden Form Field:

In case of Hidden Form Field a hidden (invisible) textfield is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

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

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

Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

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

Disadvantage of Hidden Form Field:

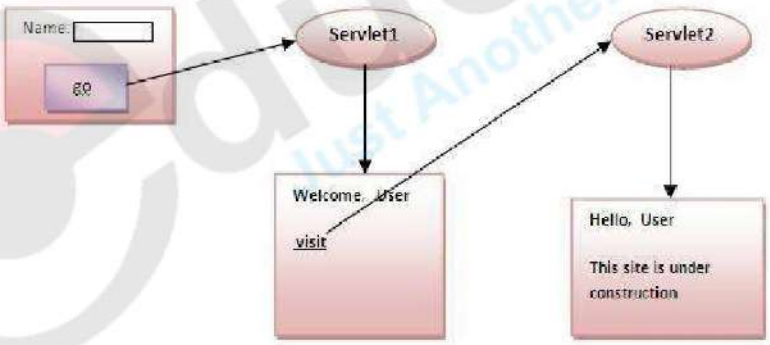
1. It is maintained at server side.
2. Extra form submission is required on each pages.
3. Only textual information can be used.

URL Rewriting:

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



Advantage of URL Rewriting

1. It will always work whether cookie is disabled or not (browser independent).
2. Extra form submission is not required on each pages.

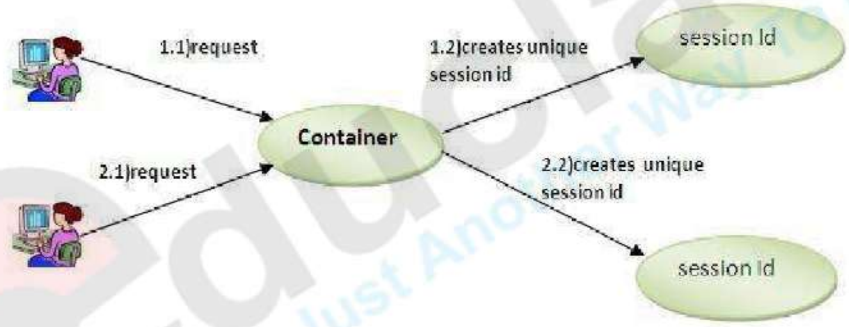
Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

HTTPSession Interface:

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

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

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

Commonly used methods of HttpSession interface

1. public String getId(): Returns a string containing the unique identifier value.
2. public long getCreationTime(): Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. public long getLastAccessedTime(): Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. public void invalidate(): Invalidates this session then unbinds any objects bound to it.

Web Development using JSP

JSP

What is JSP?

- **JSP** technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.
- A JSP page consists of HTML tags and JSP tags.
- The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

Advantage of JSP over Servlet

- There are many advantages of JSP over servlet. They are as follows:

  1) Extension to Servlet

  – JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

  2) Easy to maintain

  – JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

  3) Fast Development: No need to recompile and redeploy

  – If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

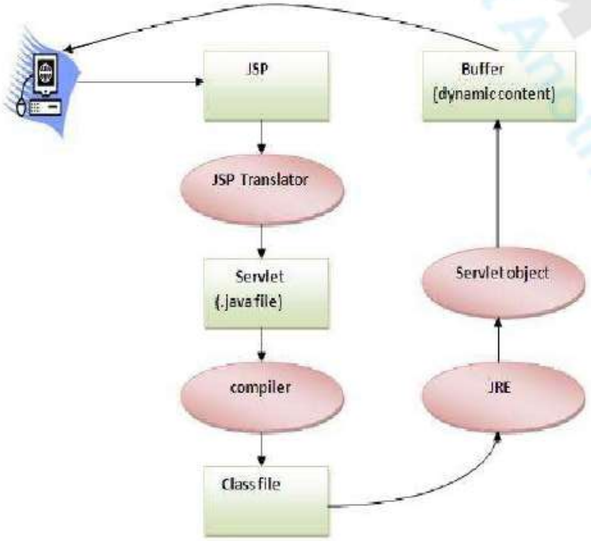
  4) Less code than Servlet

  – In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life Cycle of a JSP page

The JSP pages follows these phases:

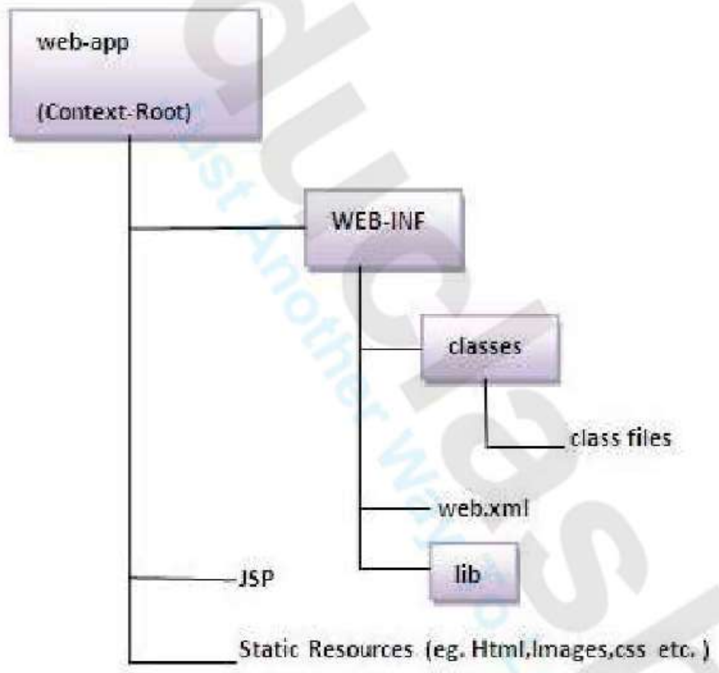
1. Translation of JSP Page
2. Compilation of JSP Page
3. Classloading (class file is loaded by the classloader)
4. Instantiation (Object of the Generated Servlet is created).
5. Initialization (jspInit() method is invoked by the container).
6. Reqeust processing (_jspService() method is invoked by the container).
7. Destroy (jspDestroy() method is invoked by the container).



- JSP page is translated into servlet by the help of **JSP translator**. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet.
- Afterthat Servlet page is compiled by the compiler and gets converted into the **class file.**
- Moreover, all the processes that happens in servlet is performed on JSP later like initialization, committing response to the browser and destroy.

Directory Structure of JSP page

- The directory structure of JSP page is same as servlet. We contains the jsp page outside the WEB-INF folder or in any directory.



JSP API

The JSP API consists of two packages:

- javax.servlet.jsp

javax.servlet.jsp package

- The javax.servlet.jsp package has two interfaces and classes.
- **The two interfaces are as follows:**
  - JspPage
  - HttpJspPage
- **The classes are as follows:**
  - JspWriter
  - PageContext
  - JspFactory
  - JspEngineInfo
  - JspException
  - JspError

JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag.

JSP Scripting elements

- The scripting elements provides the ability to insert java code inside the jsp.
- There are three types of scripting elements:
  - scriptlet tag
  - expression tag
  - declaration tag

- **JSP scriptlet tag**
  - A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:
    `<%  java source code %>`

- **Example of JSP scriptlet tag**

  In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

JSP Scriptlet tag (Scripting elements)

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

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

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");

out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP Expression tag

- The code placed within JSP expression tag is written to the output stream of the response. So you need not write out.print() to write data.
- It is mainly used to print the values of variable or method.
- Syntax of JSP expression tag

```
<%= statement %>
```

Example of JSP expression tag that prints welcome message

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Example of JSP expression tag that prints username (index.jsp)

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

Example of JSP expression tag that prints current time

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInst
ance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that gets username and prints welcome (welcome.jsp)

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

JSP declaration tag

- The **JSP declaration tag** is used *to declare fields and methods*.
- The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.
- Syntax of JSP declaration tag
  - The syntax of the declaration tag is as follows:

    <%! field or method declaration %>

Difference between JSP Scriptlet tag and Declaration tag	
Jsp Scriptlet Tag	**Jsp Declaration Tag**
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

JSP declaration tag

Example of JSP declaration tag that declares field (index.jsp)

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares Method (index.jsp)

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSP Implicit Objects

- There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.
- The available implicit objects are out, request, config, session, application etc.

JSP Implicit Objects	
Object	**Type**
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

JSP Implicit Objects

1) JSP out implicit object

- For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter.
- **Example:**

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

2) JSP request implicit object

- The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

Example of JSP request implicit object (index.html)

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

Example of JSP request implicit object (welcome.jsp)

```
<%
String name=request.getParameter("uname");
 out.print("welcome "+name);
%>
>
```

JSP Implicit Objects

3) JSP response implicit object

- In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Example of JSP response implicit object (index.html)

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

Example of JSP response implicit object (welcome.jsp)

```
<%
response.sendRedirect("http://www.google.co
m");
%>
```

4) JSP config implicit object

- In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page.
- The config object is created by the web container for each jsp page.

JSP Implicit Objects

Example of JSP config implicit object

File index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

File: welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));

 String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

File: web.xml

```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-
name>
<jsp-file>/welcome.jsp</jsp-file>

<init-param>
<param-name>dname</param-name>
<param-
value>sun.jdbc.odbc.JdbcOdbcDriver</para
m-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-
name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

JSP Implicit Objects

5) JSP application implicit object

- In JSP, application is an implicit object of type *ServletContext*.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.
- This initialization parameter can be used by all jsp pages.

JSP Implicit Objects

Example of JSP application implicit object

File index.html
```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

File: welcome.jsp
```
<%
out.print("Welcome "+request.getParameter("uname"));

 String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);

%>
```

File: web.xml
```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-
name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-
name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
<param-name>dname</param-name>
<param-
value>sun.jdbc.odbc.JdbcOdbcDriver</para
m-value>
</context-param>

</web-app>
```

JSP Implicit Objects

6) session implicit object

- In JSP, session is an implicit object of type HttpSession. The Java developer can use this object to set, get or remove attribute or to get session information.

File index.html

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

File: welcome.jsp

```
<html>
<body>
<%

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

out.print("Welcome "+name);

session.setAttribute("user",name);

<a href="second.jsp">second jsp page</a>

%>
</body>
</html>
```

File: second.jsp

```
<html>
<body>
<%
 String name=(String)session.getAttribute("user");
out.print("Hello "+name);
%>
</body>
</html>
```

JSP Implicit Objects

7) pageContext implicit object

- In JSP, pageContext is an implicit object of type PageContext class. The pageContext object can be used to set,get or remove attribute from one of the following scopes: (1) page (2) request (3) session (4) application
- In JSP, page scope is the default scope.

File index.html

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

File: welcome.jsp

```
<html>
<body>
<%

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

out.print("Welcome "+name);

pageContext.setAttribute("user",name,PageC
ontext.SESSION_SCOPE);

<a href="second.jsp">second jsp page</a>

%>
</body>
</html>
```

File: second.jsp

```
<html>
<body>
<%
 String name=(String)pageContext.getAttribute("user",P
ageContext.SESSION_SCOPE);
out.print("Hello "+name);
%>
</body>
</html>
```

JSP Implicit Objects

8) page implicit object:

- In JSP, page is an implicit object of type Object class. This object is assigned to the reference of auto generated servlet class. It is written as:

  Object page=this;

  For using this object it must be cast to Servlet type.For example:

  <% (HttpServlet)page.log("message"); %>

  Since, it is of type Object it is less used because you can use this object directly in jsp. For example:
  <% this.log("message"); %>

9) exception implicit object

- In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages. It is better to learn it after page directive. Let's see a simple example: Example of exception implicit object:

  File name: error.jsp

```
<%@ page isErrorPage="true" %>
<html>
<body>
 Sorry following exception occured:<%= exception %>
 </body>
</html>
```

JSP Directives

- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- There are three types of directives:
  - page directive
  - include directive
  - taglib directive
- Syntax of JSP Directive

  <%@ directive attribute="value" %>

1) JSP page directive

- The page directive defines attributes that apply to an entire JSP page.
- Syntax of JSP page directive

  <%@ page attribute="value" %>

- Attributes of JSP page directive:

  (1) import (2) contentType (3) info (4) buffer (5) language

JSP Directives

2) JSP include directive

- The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).
- **Advantage of Include directive**
  - Code Reusability
- Syntax of include directive

  <%@ include file="resourceName" %>
- Example of include directive

  In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
</html>
```

JSP Directives

3) JSP taglib directive

- The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags.
- In the custom tag section we will use this tag so it will be better to learn it in custom tag.).
- **Syntax JSP Taglib directive**

```
<%@ taglib uri="urioftheta glibrary" prefix="prefixoftaglibrary" %>
```

- **Example of JSP Taglib directive**
- In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.timscdr.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

Exception Handling in JSP

- The exception is normally an object that is thrown at runtime.
- Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:
  - By **errorPage** and **isErrorPage** attributes of page directive
  - By **<error-page>** element in web.xml file

Example of exception handling in jsp by the elements of page directive

- In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.
- There are 3 files:
- index.jsp for input values
- process.jsp for dividing the two numbers and displaying the result
- error.jsp for handling the exception

Exception Handling in JSP Example

File index.jsp

```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

File: process.jsp

```
<%@ page errorPage="error.jsp" %>
<%

String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);

%>
```

File: error.jsp

```
<%@ page isErrorPage="true" %>

<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```

JSP Action Tags

- There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.
- The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

SP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

JSP with Beans

- A Java Bean is a java class that should follow following conventions:
  - It should have a no-arg constructor.
  - It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

jsp:useBean action tag

- The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.
- Syntax of jsp:useBean action tag

```
<jsp:useBean id= "instanceName" scope= "page | request | session | application"
class= "packageName.className" type= "packageName.className"
beanName="packageName.className | <%= expression >" >
</jsp:useBean>
```

JSP with Beans

Example of jsp:useBean action tag

File calculator.java

```
public class Calculator{

public int cube(int n){return n*n*n;}

}
```

File: index.jsp

```
<jsp:useBean id="obj" class="com.javatp
oint.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

Session Tracking in JSP

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

Maintaining Session Between Web Client And Server

- Let us now discuss a few options to maintain the session between the Web Client and the Web Server –

  1) Cookies

  - A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.
  - This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

  2) Hidden Form Fields

  - A web server can send a hidden HTML form field along with a unique session ID as follows –
  - <input type = "hidden" name = "sessionid" value = "12345"> This entry means that, when the form is submitted, the specified name and value are automatically included in the **GET** or the **POST** data. Each time the web browser sends the request back, the **session_id** value can be used to keep the track of different web browsers.
  - This can be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

Session Tracking in JSP

3) URL Rewriting

- You can append some extra data at the end of each URL. This data identifies the session; the server can associate that session identifier with the data it has stored about that session.
- For example, with **http://timscdr.in/file.htm;sessionid=12345**, the session identifier is attached as **sessionid = 12345** which can be accessed at the web server to identify the client.
- URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies. The drawback here is that you will have to generate every URL dynamically to assign a session ID though page is a simple static HTML page.

4) The session Object

- Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.
  - a one page request or
  - visit to a website or
  - store information about that user
- By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows –
- <%@ page session = "false" %>
- The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Session Tracking in JSP

4) The session Object (Contd.)

Sr. No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last time the client sent a request associated with the this session, as the number of milliseconds since midnight January 1, 1970 GMT.
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
8	**public boolean isNew()** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

Introduction to Custom tags

- **Custom tags** are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page.
- The same business logic can be used many times by the use of custom tag.

Advantages of Custom Tags:

- **Eliminates the need of scriptlet tag** The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
- **Separation of business logic from JSP** The custom tags separate the the business logic from the JSP page so that it may be easy to maintain.
- **Re-usability** The custom tags makes the possibility to reuse the same business logic again and again.

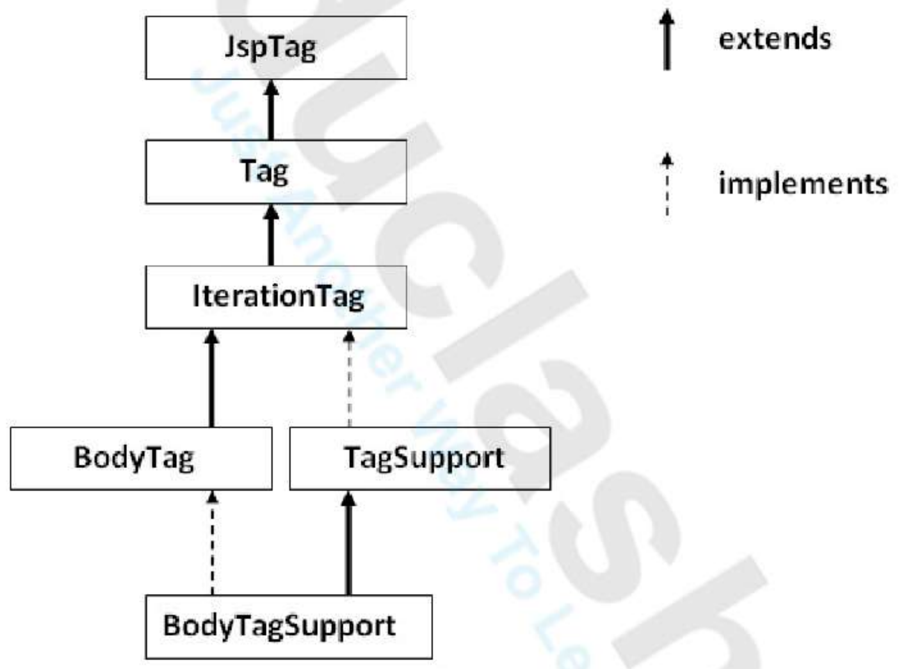
Syntax to use custom tag:

There are two ways to use the custom tag. They are given below:

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

Custom tag API

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



Introduction to Custom tags

- **JspTag interface**
  - The JspTag is the root interface for all the interfaces and classes used in custom tag. It is a marker interface.
- **Tag interface**
  - The Tag interface is the sub interface of JspTag interface. It provides methods to perform action at the start and end of the tag.
- Fields of Tag interface
  - There are four fields defined in the Tag interface. They are:

Field Name	Description
public static int EVAL_BODY_INCLUDE	it evaluates the body content.
public static int EVAL_PAGE	it evaluates the JSP page content after the custom tag.
public static int SKIP_BODY	it skips the body content of the tag.
public static int SKIP_PAGE	it skips the JSP page content after the custom tag.

Introduction to Custom tags

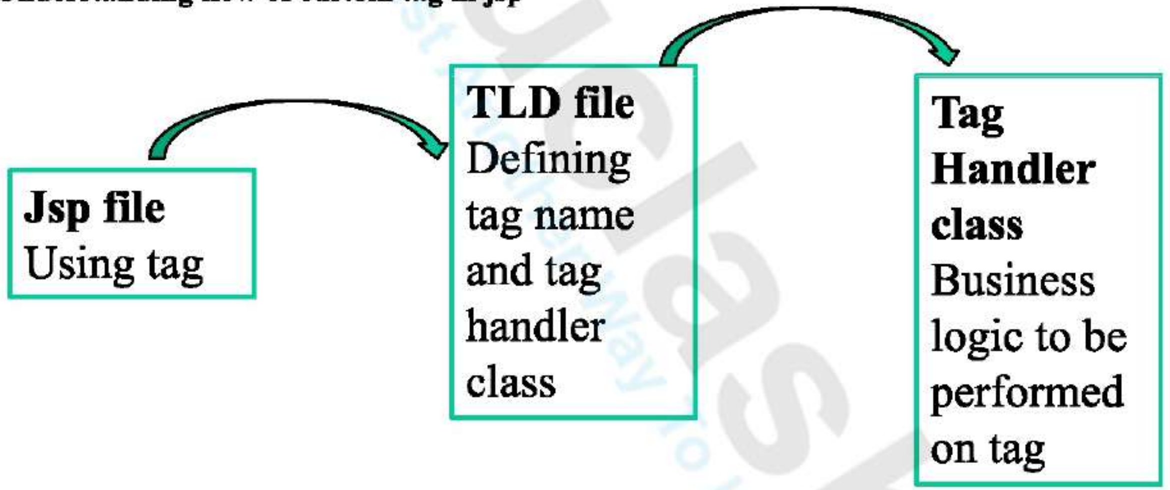
- **Methods of Tag interface**

Method Name	Description
public void setPageContext(PageContext pc)	it sets the given PageContext object.
public void setParent(Tag t)	it sets the parent of the tag handler.
public Tag getParent()	it returns the parent tag handler object.
public int doStartTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the start of the tag.
public int doEndTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the end of the tag.
public void release()	it is invoked by the JSP page implementation object to release the state.

Example of Custom tag

- In this example, we are going to create a **custom tag that prints the current date and time**. We are performing action at the start of tag.
- For creating any custom tag, we need to follow following steps:
  - **Create the Tag handler class** and perform action at the start or at the end of the tag.
  - **Create the Tag Library Descriptor (TLD) file** and define tags
  - **Create the JSP file that uses the Custom tag defined in the TLD file**

– **Understanding flow of custom tag in jsp**


Jsp file
Using tag
TLD file
Defining tag name and tag handler class
Tag Handler class
Business logic to be performed on tag

Example of Custom tag

File: MyTagHandler.java

```
import java.util.Calendar;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
public class MyTagHandler extends TagSupp
ort{

public int doStartTag() throws JspException
{
   JspWriter out=pageContext.getOut();//retur
ns the instance of JspWriter
   try{
    out.print(Calendar.getInstance().getTime()
);//printing date and time using JspWriter
   }catch(Exception e){System.out.println(e);
}
   return SKIP_BODY;//will not evaluate the
 body content of the tag
}
}
```

File: mytags.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-
//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2
//EN"
   "http://java.sun.com/j2ee/dtd/web-
jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <uri>http://tomcat.apache.org/example-
taglib</uri>

<tag>
<name>today</name>
<tag-
class>com.javatpoint.sonoo.MyTagHandler</tag-
class>
</tag>
</taglib>
```

Note: **Tag Library Descriptor** (TLD) file contains information of tag and Tag Hander classes. It must be contained inside the **WEB-INF**directory.

Example of Custom tag

File: index.jsp

```
<%@ taglib uri="WEB-
INF/mytags.tld" prefix="m" %>
Current Date and Time is: <m:today/>
```

JSTL (JSP Standard Tag Library)

- The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.
- Advantage of JSTL
  - **Fast Developement** JSTL provides many tags that simplifies the JSP.
  - **Code Reusability** We can use the JSTL tags in various pages.
  - **No need to use scriptlet tag** It avoids the use of scriptlet tag.

JSTL Core Tags

- The JSTL core tag provides variable support, URL management, flow control etc. The syntax used for including JSTL core library in your JSP is:
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

JSTL (JSP Standard Tag Library)

JSTL Core Tags

- The JSTL core tag provides variable support, URL management, flow control etc. The syntax used for including JSTL core library in your JSP is:
- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

JSP Standard Tag Library	
Tags	**Description**
c:out	It display the result of an expression, similar to the way <%=...%> tag work.
c:import	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
c:set	It sets the result of an expression under evaluation in a 'scope' variable.
c:remove	It is used for removing the specified scoped variable from a particular scope.
c:catch	It is used for Catches any Throwable exceptions that occurs in the body.
c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
c:forEach	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
\c:param	It adds a parameter in a containing 'import' tag's URL.
c:redirect	It redirects the browser to a new URL and supports the context-relative URLs.
c:url	It creates a URL with optional query parameters.

JSTL (JSP Standard Tag Library)

JSTL Core Tags

- Examples:

```
1. <c:out value="${'Welcome to javaTpoint'}"/>
2. <c:import var="data" url="http://www.javatpoint.com"/>
   <c:out value="${data}"/>
3. <c:set var="Income" scope="session" value="${4000*4}"/>
   <c:out value="${Income}"/>
4. <c:remove var="income"/>
5. <c:if test="${income > 8000}">

    <p>My income is: <c:out value="${income}"/><p>
```