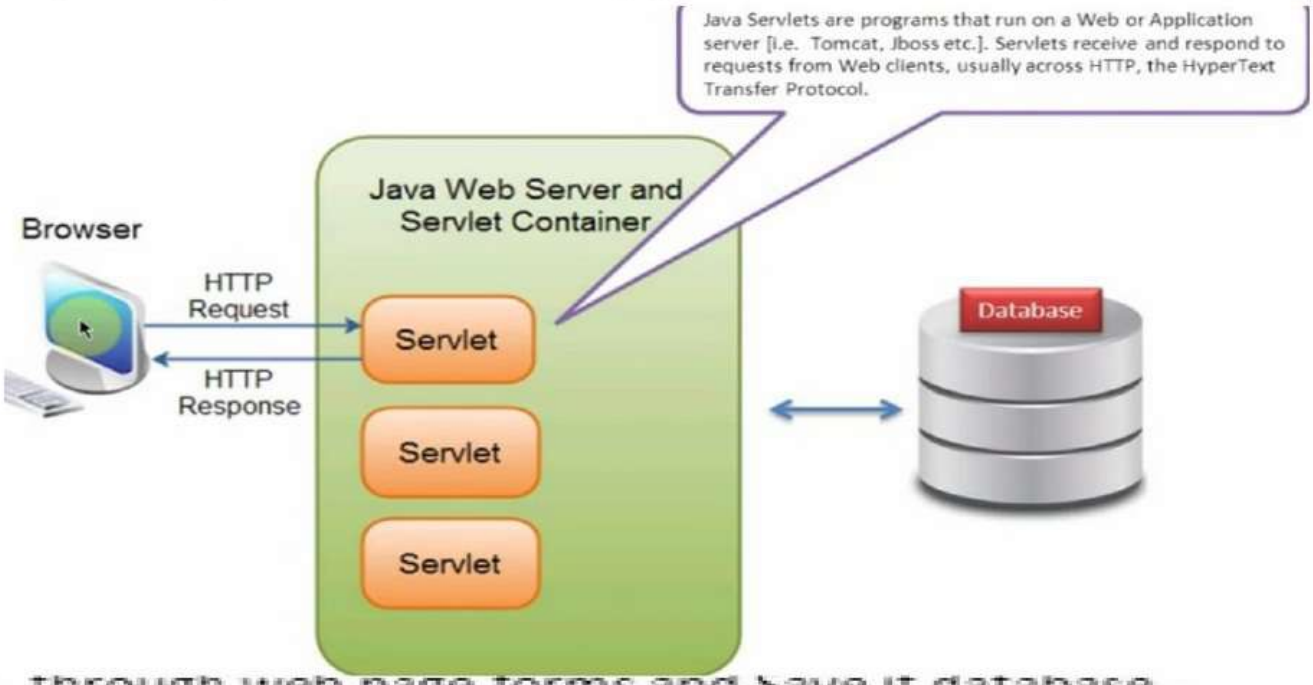
What is servlet

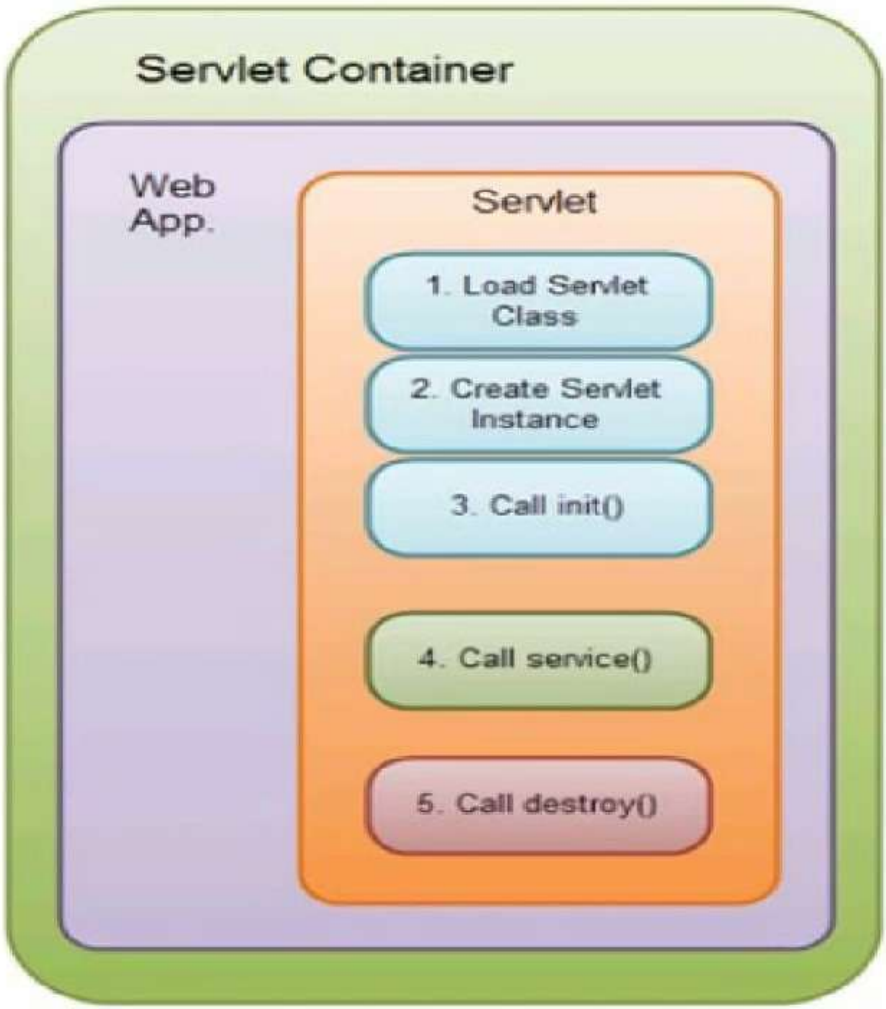
Servlet is a java file which can take the request from the client on the internet and it can process that request it can provide response in the format of html page.



through web page forms and save it database .

- ✓ Get records from a database and present to the
- ✓ Create web pages dynamically.

Servlet Life Cycle



1

Servlet Container load Servlet class

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class classname extends HttpServlet
{
}
```

2

Servlet Container creates an instance of the servlet

```
public void service(HttpServletRequest req,HttpServletResponse res)
{
}
```

3

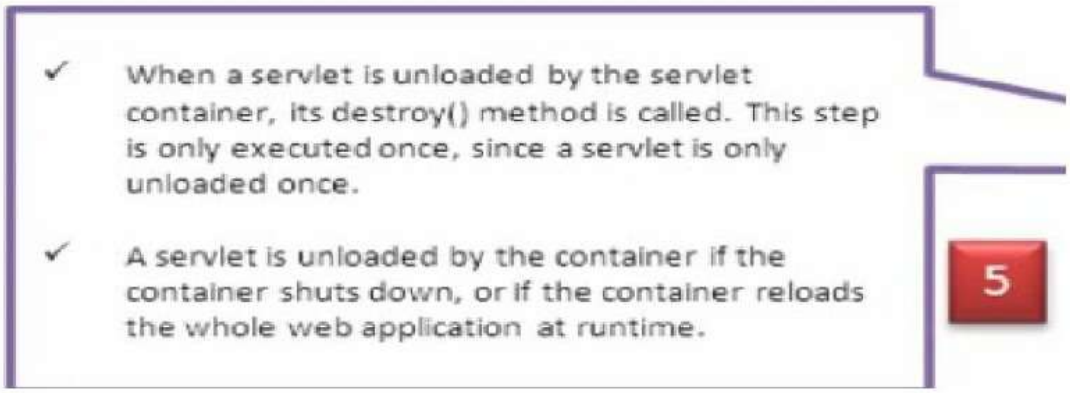
- ✓ Servlet Container will call the init() method of the servlet.
- ✓ init() method used to create or load some data that will be used throughout the life of the servlet.
- ✓ Init() method is executed once.

```
public void init()
{
int a=10;
}
```

4

- ✓ The service() method is the main method to perform the actual task.
- ✓ The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- ✓ It is executed multiple times - once for every HTTP request to the servlet.

```
public void service(HttpServletRequest req,HttpServletResponse res)
  {
        PrintWriter out=res.getWriter();
        c=a+b;
        out.println("addition is"+c);
  }
```



Interfaces in javax.servlet package

There are many interfaces in javax.servlet package.

They are as follows:

- Servlet
- ServletRequest
- ServletResponse
- RequestDispatcher

1. 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.

2. 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.
public int getContentLength()	Returns the size of the request entity data, or -1 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.

3. Servlet Response

Servlet API provides two important interfaces **ServletResponse** and **HttpServletResponse** to assist in sending response to client.

Some Important Methods of ServletResponse

Methods	Description
PrintWriter getWriter()	returns a PrintWriter object that can send character text to the client.
void setBufferSize(int size)	Sets the preferred buffer size for the body of the response
void setContentLength(int len)	Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header

void setContentType(String type)	sets the content type of the response being sent to the client before sending the respond.
void setBufferSize(int size)	sets the preferred buffer size for the body of the response.
boolean isCommitted()	returns a boolean indicating if the response has been committed
void setLocale(Locale loc)	sets the locale of the response, if the response has not been committed yet.

4. RequestDispatcher

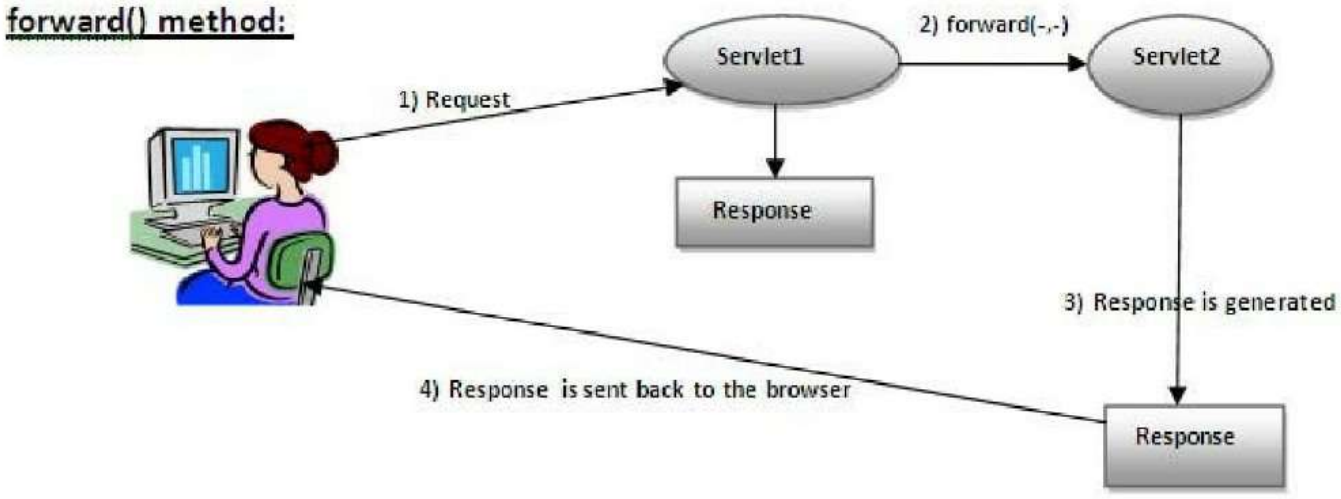
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

There are two methods defined in the RequestDispatcher interface.

Methods of RequestDispatcher interface

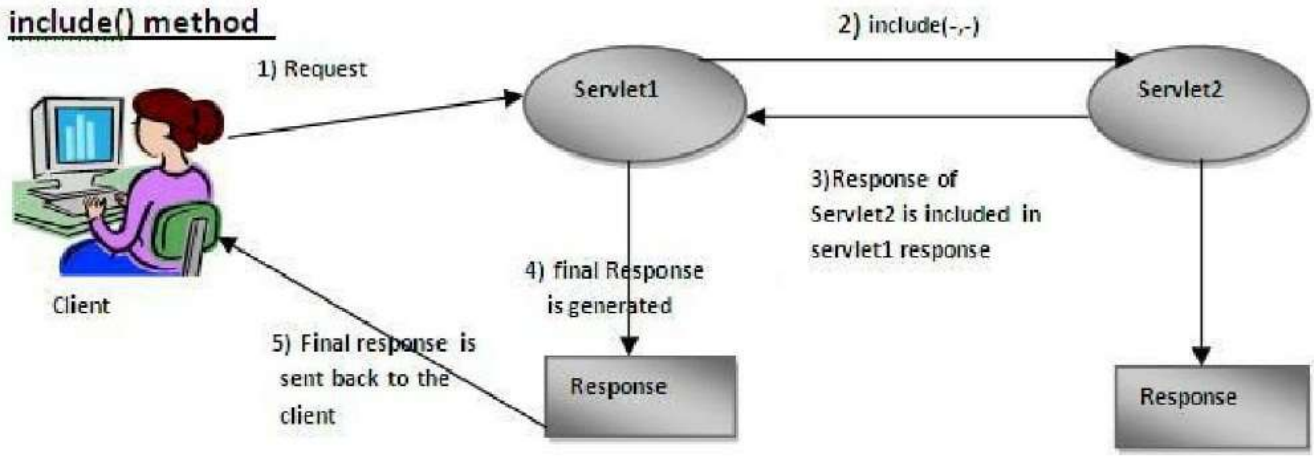
The RequestDispatcher interface provides two methods. They are:

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.



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.

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.



Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

- HttpServletRequest
- HttpServletResponse
- HttpSession

1. HttpServletRequest interface

HttpServletRequest interface adds the methods that relates to the **HTTP** protocol.

Some important methods of HttpServletRequest

Methods	Description
String getContextPath()	returns the portion of the request URI that indicates the context of the request
Cookies getCookies()	returns an array containing all of the Cookie objects the client sent with this request
String getQueryString()	returns the query string that is contained in the request URL after the path
HttpSession getSession()	returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session
String getMethod()	Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
String getServletPath()	returns the part of this request's URL that calls the servlet

2. HttpServletResponse Interface

HttpServletResponse interface adds the methods that relates to the **HTTP** response.

Some Important Methods of HttpServletResponse

Methods	Description
void addCookie(Cookie cookie)	adds the specified cookie to the response.
void sendRedirect(String location)	Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer
int getStatus()	gets the current status code of this response
String getHeader(String name)	gets the value of the response header with the given name.
void setHeader(String name, String value)	sets a response header with the given name and value
void setStatus(int sc)	sets the status code for this response
void sendError(int sc, String msg)	sends an error response to the client using the specified status and clears the buffer

3. HttpSession interface

HttpSession object is used to store entire session with a specific client. We can store, retrieve and remove attribute from **HttpSession** object. Any servlet can have access to **HttpSession** object throughout the getSession() method of the **HttpServletRequest** object.

The HttpSession object is used for session management. A session contains information specific to a particular user across the whole application. When a user enters into a website (or an online application) for the first time HttpSession is obtained via request.getSession(), the user is given a unique ID to identify his session. This unique ID can be stored into a cookie or in a request parameter.

The HttpSession stays alive until it has not been used for more than the timeout value specified in tag in deployment descriptor file(web.xml). The default timeout value is 30 minutes, this is used if you don't specify the value in tag. This means that when the user doesn't visit web application time specified, the session is destroyed by servlet container. The subsequent request will not be served from this session anymore, the servlet container will create a new session.

Creating a new session

getSession() method returns a session. If the session already exist, it return the esisting session else create a new sesion

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

Methods of HttpSession

1.setAttribute()

public void setAttribute(String name, Object value): Binds the object with a name and stores the name/value pair as an attribute of the HttpSession object. If an attribute already exists, then this method replaces the existing attributes.

You can store the user information into the session object by using setAttribute() method and later when needed this information can be fetched from the session. This is how you store info in session. Here we are storing username, emailid and userage in session with the attribute name uName, uemailId and uAge respectively.

```
session.setAttribute("uName", "ChaitanyaSingh");
session.setAttribute("uemailId", "myemailid@gmail.com");
session.setAttribute("uAge", "30");
```

This First parameter is the attribute name and second is the attribute value. For e.g. uName is the attribute name and ChaitanyaSingh is the attribute value in the code above.

2. getAttribute():

Returns the String object specified in the parameter, from the session object. If no object is found for the specified attribute, then the getAttribute() method returns null.

TO get the value from session we use the getAttribute() method of HttpSession interface. Here we are fetching the attribute values using attribute names.

```
String userName = (String) session.getAttribute("uName");
String userEmailId = (String) session.getAttribute("uemailId");
String userAge = (String) session.getAttribute("uAge");
```

Example:

index.html

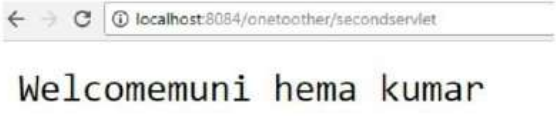
```
<form method="get" action="firstservlet">
        Enter name<input type="text" name="na" /> <br/>
      <input type="submit" value="submit"/>
     </form>
```

firstservlet.java

```
public class firstservlet extends HttpServlet
{
 public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,
IOException
 {
    String str=req.getParameter("na");
    HttpSession ses=req.getSession();
    ses.setAttribute("na", str);
    res.sendRedirect("secondservlet");
 }
}
```

secondservlet.java

```
public class secondservlet extends HttpServlet
{
 public void service(HttpServletRequest req,HttpServletResponse res) throws IOException
 {
    HttpSession ses=req.getSession();
    String str=ses.getAttribute("na").toString();
    PrintWriter out=res.getWriter();
```



```
        out.print("Welcome"+str);
    }
}
```

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

- GenericServlet
- ServletInputStream
- ServletOutputStream

1. GenericServlet class

GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. 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 String getServletName()** returns the name of the servlet object.

2. ServletInputStream class

ServletInputStream class provides stream to read binary data such as image etc. from the request object. It is an abstract class.

The **getInputStream()** method of **ServletRequest** interface returns the instance of ServletInputStream class. So can be get as:

1. ServletInputStream sin=request.getInputStream();

Method of ServletInputStream class

There are only one method defined in the ServletInputStream class.

1. **int readLine(byte[] b, int off, int len)** it reads the input stream.

3. ServletOutputStream class

ServletOutputStream class provides a stream to write binary data into the response. It is an abstract class.

The **getOutputStream()** method of **ServletResponse** interface returns the instance of ServletOutputStream class. It may be get as:

```
ServletOutputStream out=response.getOutputStream();
```

Methods of ServletOutputStream class

The ServletOutputStream class provides print() and println() methods that are overloaded.

```
void print(datatype name){}

void println(datatype name){}
```

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

- HttpServlet
- Cookie

1. 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.

2. Cookie class

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

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

Method	Description
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.

Open any shopping website.

Select any product and add it to cart.

Again select any other product (this is new request means we are in other page)

Our previous selected product is stored in cart. Here cart behave like a cookie now.

In order to save your data for this session either u can use sesssion or cookie.

Sending

Cookie is class

Cookie obj=new Cokiee("lable",ourdatavariable);

responseobject.addCookie(cookieobject);

Receiving (In second servlet)

Here the client will send all cookies.

Becauase client doesn't know which cookie client a server it is.

To receive all the cookies we use getCookies() method.

```
String str=null;
Cookie obj[]=requestobject.getCookies();
for(Cookie variable: cookiearrayobject)
{
  if(variable.getName().equals("textboxname"))
   {
```

```
            stringvaribale=variable.getValue();
        }
    }
```

Example:

index.html

```
<form method="get" action="firstservlet">
        Enter name<input type="text" name="na" /> <br/>
    <input type="submit" value="submit"/>
    </form>
```

firstservlet.java

```
public class firstservlet extends HttpServlet
{
public void service(HttpServletRequest req,HttpServletResponse res) throws ServletException,
IOException
 {
   String str=req.getParameter("na");
   Cookie ck=new Cookie("na",str);
   res.addCookie(ck);
   res.sendRedirect("secondservlet");
 }
}
```

secondservlet.java

```
public class secondservlet extends HttpServlet
{
 public void service(HttpServletRequest req,HttpServletResponse res) throws IOException
 {
   Cookie cook[]=req.getCookies();
   String str=null;
   for(Cookie c:cook)
   {
     if(c.getName().equals("na"))
     {
       str=c.getValue();
     }
   }
   PrintWriter out=res.getWriter();
   out.print("Welcome"+str);
 }
```

Java Database Connectivity Steps(JDBC Connectivity using Servlet)

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

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

- Closing connection

1) Register the driver class

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

Syntax

Example to register the OracleDriver class

```
Class.forName("com.mysql.jdbc.Driver");
```

2) Create the connection object

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

Syntax

```
Connection con=DriverManager.getConnection
                ("jdbc:mysql://localhost:3306/databasename","userame","password");
```

3) Create the Statement object

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

Syntax:

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

4) Execute the query

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

Syntax:

```
ResultSet rs=stmt.executeQuery(queryobject);
while(rs.next())
{
  //records will be here
  // To receive database column use rs.getString(1)
1 means first column
}
```

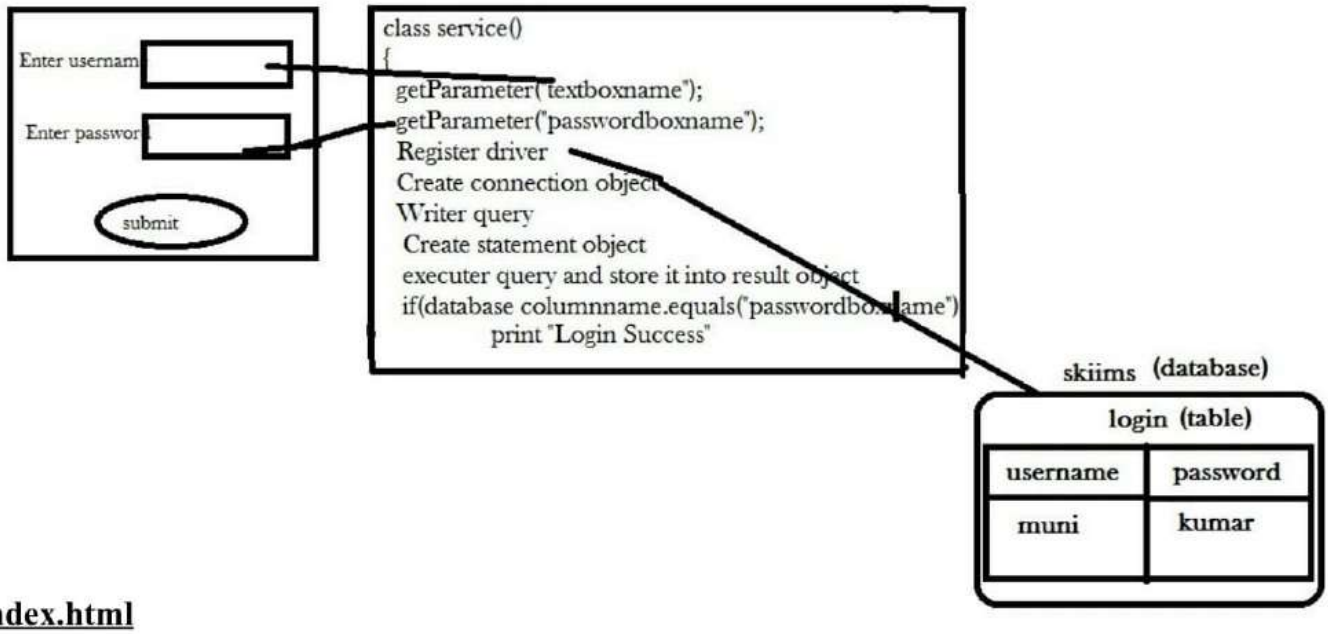
5) Close the connection object

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

Syntax

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

Login Page using Servlet (Creating Login Page using Servlet with JDBC)



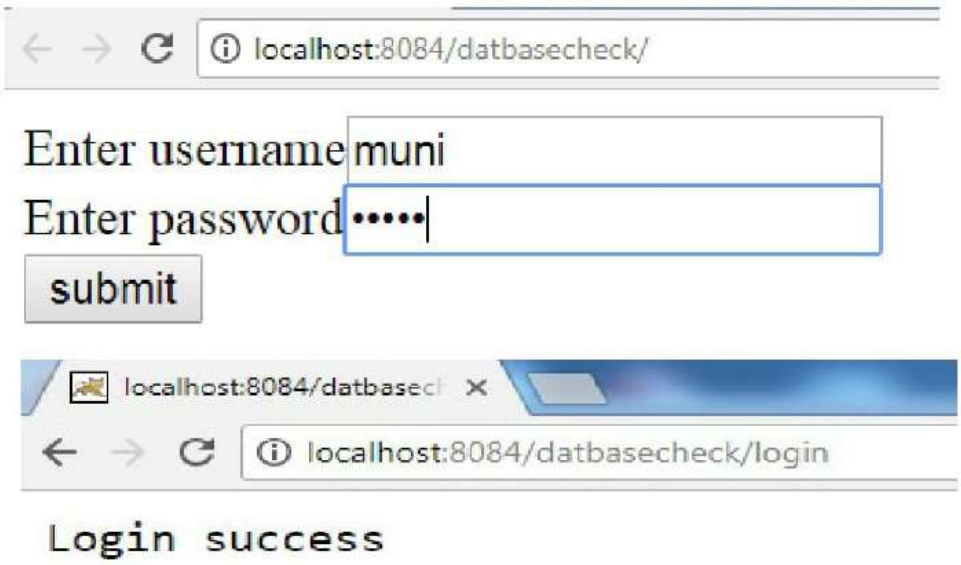
index.html

```
<form method="post" action="login">
        Enter username<input type="text" name="un" /><br/>
        Enter password<input type="password" name="up" /><br/>
        <input type="submit" value="submit" />
</form>
```

login.java

```
public class login extends HttpServlet
{
 public void service(HttpServletRequest req,HttpServletResponse res) throws IOException
{
 String uname=req.getParameter("un");
 String upass=req.getParameter("up");
 PrintWriter out=res.getWriter();
  try
  {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/muni","root","admin");
    String q="select * from login where username='"+uname+"' ";
    Statement ps=con.createStatement();
    ResultSet rs = ps.executeQuery(q);
    if(rs.next())
    {
      if(rs.getString(2).equals(upass))
        out.println("Login success");
      }
  }
```

```
    catch (ClassNotFoundException | SQLException ex)
    {
        Logger.getLogger(login.class.getName()).log(Level.SEVERE, null, ex);
    }
  }
}
```



JSP (Java Server Pages)

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.

JSP technology is used to create web application just like Servlet technology.

JavaServer Pages (**JSP**) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.

- JSPs are normal HTML pages with embedded Java code. To process a JSP file, developers need a JSP engine, which is connected to a Web server.
- The JSP page is then compiled into a servlet, which is handled by the servlet engine. This phase is known as translation.
- The servlet engine then loads the servlet class and executes it to create dynamic HTML, which is then sent to the browser.
- When the next page is requested, the JSP page is precompiled into the servlet and executed, unless the JSP page is changed.

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

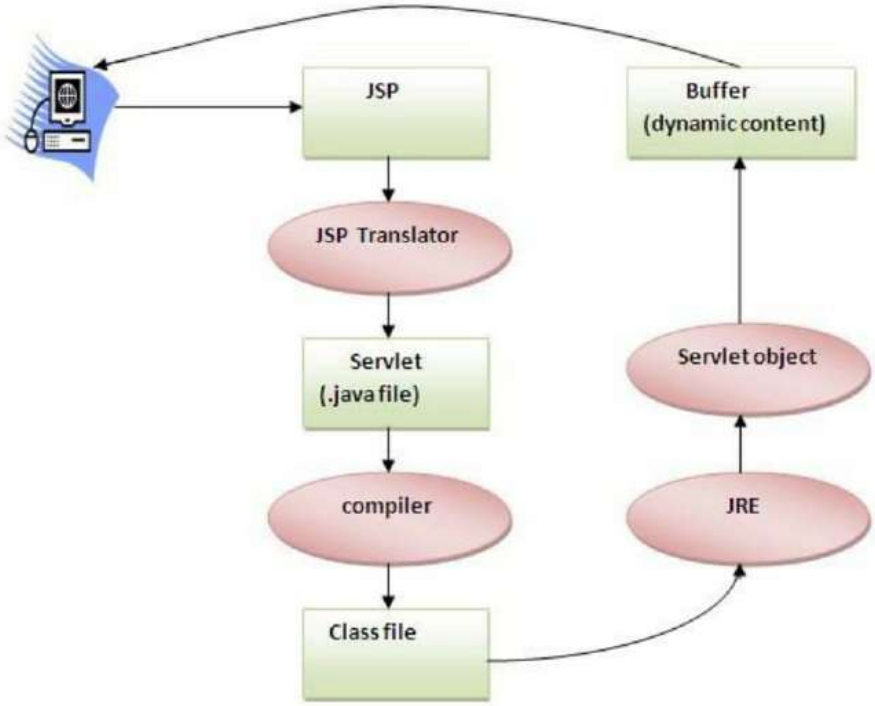
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life cycle of a JSP Page

The JSP pages follows these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.



As depicted in the above diagram, 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.

JSP Scripting elements

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

JSP Scripting elements

The scripting elements provide the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriplet 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>
```

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

It is mainly used to print the values of variable or method.

Syntax

```
<%=  variable or method %>
```

Example:

```
<%
  int a=10;
 int b=20;
 int c=a+b;
%>
<%=  c

 %>
```

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

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

JSP Declaration Tag

The **JSP declaration tag** is used *to declare variables 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:

```
<%!
        variable 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.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

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

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

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

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

JSP Implicit Objects

Thee are no.of **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.

Object	**Type**
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
session	HttpSession
exception	Throwable