Request Dispatcher

* Interface present in javax.servlet package.
* Used to achieve communication between the Servlets
* Two ways of communication mechanism supported by request dispatcher.
* Forward
* Include
* Communication between the Servlets is done within the same server.

1.Forward

* In this mechanism one Servlet forwards the request and respone to another Servlet.

server

w eb browser

Request request

Servlet2

Servlet3

Servlet1

RequestDispatcher rd=request.getRequestDispatcher(“servletname”);

rd.forward(request,reponse);

2.include

request

Servlet

Web browser

includes response in sourcepage

RequestDispatcher rd=request.getRequestDispatcher(“html filename”);

rd.include(request,reponse);

HttpServlet:

* Present in javax.servlet.http package.
* Httpservlet was designed to work with http protocol only where as GenericServlet was designed to work with any type of protocols.
* The main purpose of introducing httpservlet is
* To achieve sendredirection mechanism(communicating one Servlet to another Servlet which are present in different servers.)
* To make “stateless http protocol ”to “state protocol’ with the help of session tracking mechanism.

**Methods of httpservlet class:**

1. Protected void doget(javax.servlet.http.HttpServletRequest request, (javax.servlet.http.HttpServletResponse response)
2. Protected long getlastmodified(javax.servlet.http.HttpServletRequest request)
3. Protected void doPost(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
4. Protected void doDelete(HttpServletRequest request, HttpServletResponse response)
5. Public void service(ServletRequest request,ServletResponse response)
6. Protected void service(HttpServletRequest request,HttpServletResponse response)

**Execution flow of HttpServlet**

1.public void service(ServletRequest request,ServletResponse response)

2. protected void service(HttpServletRequest request,HttpServletResponse response)

3.doGet(HttpServletRequest request,HttpServletResponse response)

(Or)

doPost(HttpServletRequest request,HttpServletResponse response)

**Syntax to create HttpServlet class**

Import javax.servlet.http.\*;

Class classname extends HttpServlet

{

//override doget or dopost methods

}

**SendRedirection**

* It is a mechanism used to achieve the Servlet communication which are present in same server or different servers.
* HttpServletResponse interface provides the following predefined method to achieve the sendRedirection.
* response.sendRedirect();

**Syntax to call sendRedirect() method**

**Syntax1(within the same server)**

response.sendRedirect(“url pattern of Servlet name”)

ex:response.sendRedirect(“/servlet2”)

**Syntax2(in different servers)**

response.sendRedirect(“any url”);

ex:response.sendRedirect(”<http://www.icicibank.com>”);

**session tracking or session handling:**

* session tracking is a mechanism used to convert stateless http protocol to state protocol i.e., by default http protocol doesnot store any data in server system and can be stored using session tracking until a period of time.
* By using session tracking data of the user(state) can be stored either in client or in server.
* Becoz of maintaining data of the user server can identify that user until the session is closed
* Session tracking is achieved in following 4 ways:-

1. Using cookies
2. Using hiddenformfields
3. Using url rewriting
4. Using http session

1.using cookies

* A cookie is a piece of information stored in the browser to validate a user/client.
* As it is stored in the browser,there is no burden to the server.
* Whenever a client sends first request to the server,it sends the response along with the cookie.
* This cookie received by the client will be stored in the browser so from the second request onwards cookie will be appended to the request object.
* This cookie object is created by the server
* To work with cookies javax.servlet.http package contains a pre-defined class called “cookie”.
* But when ur browser doesnot support cookies or disables cookies this cookie will not work.

Web browser/client server

1st request

cookie

name

response+cookie

password

2nd request+cookie

3rd request+cookie

The above figure shows how a cookie works.

**<**form action=”FirstServlet"method="post">

Name:<input type="text" name="userName"/><br/>

<input type="submit" value="go"/>

</form>

</body>

</html>

Servlet class

public classFirstServlet extendsHttpServlet

{

public voiddoPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException

{ 1

response.setContentType("text/html");

PrintWriter out = response.getWriter();

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

out.print("Welcome "+n);

out.print("<form action='SecondServlet' method=’GET’>");

out.print(<input type="hidden" name="uname" value=n>);

out.print("<input type='submit' value='go'>");

Cookie ck=**new** Cookie("username",n);//creating cookie object

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

out.print("<form action='SecondServlet' method=’GET’>");

out.print("<input type='submit' value='go'>");

out.print("</form>");

}}

Secondservlet

public classSecondServlet extendsHttpServlet {

public voiddoPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException

{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

Cookie ck[]=request.getCookies();

//request.getParameter(uname);

out.print("Hello "+ck[0].getValue());

}}

**2.using HIDDEN FORMFIELDS.**

* In this approach a hidden text field is used for maintaining the state of user.
* This hidden text field is accessed by the Servlet and the state of the user is stored in the server.
* <input type="hidden" name="uname" value="state\_of\_user">

Servlet1

Servlet2

browser/client

uname

request

creates form fields

V=

submit

uname is passed to servlet2

value

submit

3.**using URLREWRITING:**

Browser/client Servlet1 Servlet2

request

name

Submit

link

creates

user data/state

* In this technique a identifier/token will be appended to the url of next servelt or next resource.
* In this technique state of the user is passed in the form of key-value pairs.
* Syntax: url?a1=value&a2=value&……
* The above url has to be passed in the form of link(use anchor tag).
* In the above diagram whenever user clicks the link ,a parameter (key-value) pair is passed to the servelet2 using request.getparameter(“key”).
* It works only with hyper\_links.

4.Using **http session:**

* It is a predefined interface in javax.servlet.http package used to maintain the authenticated data of user(state of user) in server machine.
* Whenever a new request is coming to the Server, web container will create httpsession object in which the state of the user is stored and can be used multiple times by all the Servlets present in a webapplication.

server

creates

creates

creates

Re request

request

request

servlet

C client3

Client2

Client1

**Getting HttpSession Object:**

* HttpServletRequest interface provides following methods to get the http session object.

1. Public HttpSession getSession()

* The above method returns HttpSession object if already exists,else creates a new object if not existed.

Syntax:

HttpSession hs=request.getHttpSession();

2.public HttpSession getSession(boolean b)

* In the above method if the Boolean value is true it returns the existing HttpSession object or creates a new object if it is not existed.
* If Boolean value is false returns HttpSession object if existed else returns null.

Syntax:

HttpSession hs=request.getHttpSession(false);

**HttpSession interface methods:**

1.setAttribute()

Used to set the value in httpsession object in the form of object

**Syntax**

hs.setAttribute(String key,Object data);

2.getAttribute()

Used to get the httpsession object data.

**syntax**

hs.getAttribute(String key);

3.getCreateTime()

Returns the httpsession object created time in milliseconds.

4.getlastAccessedTime()

Used to get the last accessed time by the client in milliseconds.

5.invalidate()

Used to destroy the httpsession object.

Design a web application to work with httpsession with the following requirements.

Hello uname

Welcome uname

**clickhere**

Servlet2

S Servlet1

submit

uname

Servlet1.java

Public class MyServlet1 extends HttpServlet

{

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

{

Response .setContentType(“text/html”);

PrintWriter pw=reponse.getWriter();

String uname=request.getParameter(“name”);

HttpSession hs=request.getSession();

Hs.setAttribute(“name”,uname);

Pw.println(“welcome”+uname);

Pw.println(“<html><body><a href=’servlet2’>click here</a></body></html>”);

}}

Servlet2.java

Public class MyServlet2 extends HttpServlet

{

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

{

Response .setContentType(“text/html”);

PrintWriter pw=reponse.getWriter();

HttpSession hs=request.getSession();

Object o=hs.getAtrribute(“name”);

String un=(String)o;

Pw.println(“hello”+un);

**Java JDBC(java database connectivity)**

* Java jdbc is an api which is used to connect our java application to a database.
* As java is a object oriented programming language and database contains tables,queries inorder to provide communication in between them a java api uses jdbc drivers.
* Jdbc driver acts as a bridge between a java application and a database.

Java api

database

Jdbc driver

Java application

* The above figure depicts how a java application is connected to database.
* These jdbc drivers are classified into 4 types:

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:**

* It was the first jdbc driver designed to communicate a java application with a database.
* This jdbc-odbc bridge driver uses a odbc driver and vendor specific library jars to connect to the database.
* Odbc driver converts jdbc method calls into odbc function calls which is written in c language.

**2.Native-API driver (partially java driver):**

* If any driver software connects the java application with database through vendor specific driver comes under type2 driver.
* Database vendors designed their own driver software using c/c++ .so these type of drivers are called native api driver.



**3.Network protocol driver(fully java driver):**

****

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

4.**Thin driver(fully java driver):**

* If any driver s/w given by database vendor is going to communicate java application with the database directly then those drivers comes under type4 driver.
* All type4 drivers are designed by using java languages so these drivers are called as pure driver.



**JDBC API:**

This jdbc api contains many predefined interfaces and classes. Following are the interfaces and classes present in the api.

|  |  |
| --- | --- |
| Interfaces  1.connection  2.statement  3.preparestatement  4.resultset  5.callablestatement  6.resultsetmetadata  7.databasemetadata | Classes  1.drivermanager |

**5 steps to connect a java application to a database:**

1. Register the driver class
2. Creating connection
3. Creating statement
4. Executing queries
5. Closing connection

**Register the driver class :**

A call to Class.forName(“fully\_qualified\_classname”) is used to register the specified driver class.

(1)To register the OracleDriver class

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

**(2)** To register the MySqlDriver class

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

**Creating connection:**

To create a connection object we need to call getConnection method present in DriverManager class

Syntax of getConnection() method

1. public staticConnection getConnection(String url)throwsSQLExce

ption

2. public staticConnection getConnection(String url,String name,Strin

g password)**throws** SQLException

**Example:**

1)To establish connection with the Oracle database

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","password");

**2)** To establish connection with the MySql database

Connection con=DriverManager.getConnection(

"jdbc:mysql://localhost:3306/test","root","root");

**3.Creating statement :**

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

1. publicStatement createStatement()throwsSQLException

Example to create the statement object

1. Statement stmt=con.createStatement();

**4.Execute the query:**

The executeQuery() method of Statement interface is used to execute queries in 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

publicResultSet executeQuery(String sql)throwsSQLException

Example to execute query

ResultSet rs=stmt.executeQuery("select \* from Emp"); //executeUpdate()

**while**(rs.next()){

System.out.println(rs.getInt(1)+" "+rs.getString(2));

}

**5.Close the connection**

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 voidclose()throwsSQLException

Example to close connection

con.close();

mvc architecture:



* MVC is an architecture or design pattern that separates business logic, presentation and data. In MVC,
* M stands for Model (represented as java bean class)
* V stands for View(represented as html,jsp)
* C stands for controller(represented as servlet)

**import** java.io.\*;

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**public class** ControllerServlet **extends** HttpServlet {

**protected void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out=response.getWriter();

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

String password=request.getParameter("password");

LoginBean bean=**new** LoginBean();

bean.setName(name);

bean.setPassword(password);

request.setAttribute("bean",bean);

**boolean** status=bean.validate();

**if**(status){

RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");

rd.forward(request, response);

}

**else**{

RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");

rd.forward(request, response);

}

**File: login-success.jsp**

<%@page **import**="com.java.LoginBean"%>

<p>You are successfully logged in!</p>

<%

LoginBean bean=(LoginBean)request.getAttribute("bean");

out.print("Welcome, "+bean.getName());

%>

**File: login-error.jsp**

<p>Sorry! username or password error</p>

<%@ include file="index.jsp" %>