**Java Servlets**

A servlet is a java based server-side technology.Java Servlets are programs that run on a web server and used for developing web applications. Using Servlets, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

**Servlets Tasks**

Servlets perform the following major tasks −

Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.

Process the data and generate the results. This process may require talking to a database, executing an RMI invoking a Web service, or computing the response directly.

Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.

Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such task.

**Servlets Packages**

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the javax.servlet and javax.servlet.http packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

**Servlets - Life Cycle**

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

The servlet is initialized by calling the init() method.

The servlet calls service() method to process a client's request.

The servlet is terminated by calling the destroy() method.

Finally, servlet is garbage collected by the garbage collector of the JVM.

=====================================================================================

**The init() Method**

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this −

public void init() throws ServletException {

// Initialization code...

}

The service() Method

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.

Each time the server receives a request for a servlet, the server

spawns a new thread and calls service. The service() method checks

the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet,

doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

}

The service () method is called by the container and service method

invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate.

So you have nothing to do with service() method but you override

either doGet() or doPost() depending on what type of request you

receive from the client.

The doGet() and doPost() are most frequently used methods with in

each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML

form that has no METHOD specified and it should be handled by doGet()

method.

public void doGet(HttpServletRequest request, HttpServletResponse

response)

throws ServletException, IOException {

// Servlet code

}

The doPost() Method

A POST request results from an HTML form that specifically lists POST

as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request, HttpServletResponse

response)

throws ServletException, IOException {

// Servlet code

}

The destroy() Method

The destroy() method is called only once at the end of the life cycle

of a servlet. This method gives your servlet a chance to close

database connections, halt background threads, write cookie lists or

hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked

for garbage collection. The destroy method definition looks like this

−

public void destroy() {

// Finalization code...

}

=====================================================================

**Steps to create a servlet example**

The servlet example can be created by three ways:

By implementing Servlet interface,

By inheriting GenericServlet class, (or)

By inheriting HttpServlet class

**The steps are as follows:**

Create a directory structure

Create a Servlet

Compile the Servlet

Create a deployment descriptor

Start the server and deploy the project

Access the servlet

=====================================================================

**Sample Code**

/ Import required java libraries

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

// Extend HttpServlet class

public class HelloWorld extends HttpServlet {

private String message;

public void init() throws ServletException {

// Do required initialization

message = "Hello World";

}

public void doGet(HttpServletRequest request, HttpServletResponse

response)

throws ServletException, IOException {

// Set response content type

response.setContentType("text/html");

// Actual logic goes here.

PrintWriter out = response.getWriter();

out.println("<h1>" + message + "</h1>");

}

public void destroy() {

}

}

====================================================================

**web.xml**

<servlet>

<servlet-name>HelloWorld</servlet-name>

<servlet-class>HelloWorld</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloWorld</servlet-name>

<url-pattern>/HelloWorld</url-pattern>

</servlet-mapping>

====================================================================

**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.

Servlet Example by inheriting the GenericServlet class

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>");

}

}

====================================================================

**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

**Example of ServletRequest to display the name of the user**

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 in Servlet**

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.

**Methods of RequestDispatcher interface**

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.

Example of RequestDispatcher interface

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.

====================================================================

SendRedirect in servlet

The sendRedirect() method of HttpServletResponse interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

====================================================================

**ServletConfig Interface**

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.

Methods of ServletConfig interface

public String getInitParameter(String name):Returns the parameter

value for the specified parameter name.

public Enumeration getInitParameterNames():Returns an enumeration of

all the initialization parameter names.

public String getServletName():Returns the name of the servlet.

public ServletContext getServletContext():Returns an object of

ServletContext.

====================================================================

Attribute in Servlet

An attribute in servlet is an object that can be set, get or removed

from one of the following scopes:

request scope

session scope

application scope

=====================================================================

Session Tracking in Servlets

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.

Session Tracking Techniques

There are four techniques used in Session tracking:

Cookies

Hidden Form Field

URL Rewriting

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.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using

cookies. It provides a lot of useful methods for cookies.

example of Servlet Cookies

index.html

<form action="servlet1" method="post">

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

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

</form>

FirstServlet.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class FirstServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse

response){

try{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

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

out.print("Welcome "+n);

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

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

//creating submit button

out.print("<form action='servlet2'>");

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

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

out.close();

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

}

}

====================================================================

SecondServlet.java

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class SecondServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse

response){

try{

response.setContentType("text/html");

PrintWriter out = response.getWriter();

Cookie ck[]=request.getCookies();

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

out.close();

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

}

}

====================================================================

**web.xml**

<web-app>

<servlet>

<servlet-name>s1</servlet-name>

<servlet-class>FirstServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>s1</servlet-name>

<url-pattern>/servlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>s2</servlet-name>

<servlet-class>SecondServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>s2</servlet-name>

<url-pattern>/servlet2</url-pattern>

</servlet-mapping>

</web-app>

====================================================================================