Servlets

Why Servlets?

As soon as the Web began to be used for delivering services, service providers recognized the need for dynamic content. Applets, one of the earliest attempts towards this goal, focused on using the client platform to deliver dynamic user experiences. At the same time, developers also investigated using the server platform for this purpose. Initially, Common Gateway Interface (CGI) scripts were the main technology used to generate dynamic content. Though widely used, CGI scripting technology has a number of shortcomings, including platform dependence and lack of scalability. To address these limitations, Java Servlet technology was created as a portable way to provide dynamic, user-oriented content.

What is a Servlet?

A *servlet* is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.
The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

Life cycle of Servlet

The life cycle of a servlet is controlled by the container in which the servlet has been deployed. When a request is mapped to a servlet, the container performs the following steps.
If an instance of the servlet does not exist, the Web container

1) Loads the servlet class.
2) Creates an instance of the servlet class.
3) Initializes the servlet instance by calling the init method.
4) Invokes the service method, passing a request and response object.
5) If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method.

After the Web container loads and instantiates the servlet class and before it listens to the requests of the clients, the Web container initializes the servlet. A servlet that cannot complete its initialization process will throw UnavailableException.The service provided by a servlet is implemented in the service method of a GenericServlet, the do*Method* methods (where *Method* can take the value Get, Delete, Options, Post, Put, Trace) of an

HttpServlet, or any other protocol-specific methods defined by a class that implements the Servlet interface.

The general pattern for a service method is to extract information from the request, access external resources, and then populate the response based on that information. For HTTP servlets, the correct procedure for populating the response is to first fill in the response headers, then retrieve an output stream from the response, and finally write any body content to the output stream. Response headers must always be set before a PrintWriter or ServletOutputStream is retrieved because the HTTP protocol expects to receive all headers before body content.

Getting Information from Requests

A request contains data passed between a client and the servlet. All requests implement the ServletRequest interface. This interface defines methods for accessing the following information:

1) Parameters, which are typically used to convey information between clients and servlets
2) Object-valued attributes, which are typically used to pass information between the servlet container and a servlet or between collaborating servlets.
3) Information about the protocol used to communicate the request and the client and server involved in the request.
4) Information relevant to localization.
5) An input stream can be retrieved from the request and manually parse the data. To read character data, use the BufferedReader object returned by the request's getReader method. To read binary data, use the ServletInputStream object returned by getInputStream.

HTTP servlets are passed an HTTP request object, HttpServletRequest, which contains the request URL, HTTP headers, query string, and so on.

An HTTP request URL contains the following parts:

http://*<host>*:*<port><requestpath>*?*<querystring>*

The request path is composed of the following elements:

Context path: A concatenation of a forward slash (/) with the context root of the servlet's J2EE application.
Servlet path: The path section that corresponds to the component alias that activated this request. This path starts with a forward slash (/).
Path info: The part of the request path that is not part of the context path or the servlet path.

Query strings are composed of a set of parameters and values. Individual parameters are retrieved from a request with the getParameter method.A query string can explicitly appear in a Web page.

Constructing Responses

A response contains data passed between a server and the client. All responses implement the ServletResponse interface. This interface defines methods that allow you to do the following:

1) Retrieve an output stream to use to send data to the client.
2) To send character data, use the PrintWriter returned by the response's getWriter method. To send binary data in a MIME body response, use the ServletOutputStream returned by getOutputStream.
3) To mix binary and text data, for example, to create a multipart response, use a ServletOutputStream and manage the character sections manually.
4) Indicate the content type (for example, text/html), being returned by the response.
5) By default, any content written to the output stream is immediately sent to the client. HTTP response objects, HttpServletResponse, have fields representing HTTP headers such as Status codes, which are used to indicate the reason a request is not satisfied.Cookies, which are used to store application-specific information at the client.

Finalizing a Servlet

When a servlet container determines that a servlet should be removed from service (for example, when a container wants to reclaim memory resources, or when it is being shut down), it calls the destroy method of the Servlet interface. In this method,any resources the servlet is using can be released.

All of a servlet's service methods should be complete when a servlet is removed. The server tries to ensure this completion by calling the destroy method only after all service requests have returned or after a server-specific grace period, whichever comes first.

Advantages over CGI

1) The servlets havefast performance and ease of use combined with more power over traditional CGI (Common Gateway Interface).

2) When an HTTP request is made, a new process is created for each call of the CGI script. This overhead of process creation can be very system-intensive, especially when the script does relatively fast operations. Thus, process creation will take more time than CGI script execution. But in case of Java servlets, a servlet is not a separate process. Each request to be handled by a servlet is handled by a separate Java thread within the Web server process, omitting separate process forking by the HTTP daemon.

3) Simultaneous CGI request causes the CGI script to be copied and loaded into memory as many times as there are requests. However, with servlets, there is the same amount of threads as requests, but there will only be one copy of the servlet class created in memory that stays there also between requests.
4) Only a single instance answers all requests concurrently. This reduces memory usage and makes the management of persistent data easy.
5) A servlet can be run by a servlet engine in a restrictive environment, called a sandbox.

<u>Overview of the execution of a Servlet</u>

1) The container calls the no-arg constructor and instantiates a servlet container (object).
2) The Web container calls the init () method. This method initializes the servlet and must be called before life of a servlet, the init () method is called only once.
3) After initialization, the servlet can service client requests. Each request is serviced in its own separate thread. The Web container calls the service () method of the servlet for every request. The service () method determines the kind of request being made and dispatches it to an appropriate method to handle the request. The developer of the servlet must provide an implementation for these methods. If a request for a method that is not implemented by the servlet is made, the method of the parent class is called, typically resulting in an error being returned to the requester.
4) Finally, the Web container calls the destroy () method that takes the servlet out of service. The destroy () method, like init (), is called only once in the lifecycle of a servlet.

<u>Method used in Servlet Execution</u>

Three methods are central to the life cycle of a servlet. These are ***init ( )***, ***service ( )***, and ***destroy ( )***. They are implemented by every servlet and are invoked at specific times by the server.

init ()

The user enters a Uniform Resource Locator (URL) to a web browser.The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server. The HTTP request is received by the web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address space of the server. The server invokes the **init ()** method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.

service ()

The server then invokes the **service()** method of the servlet.This method is called to process the HTTP request.The servlet can read data that has been provided in the HTTP request, in form of a query string.It may also formulate an HTTP response for the client. The servlet remains in the server’s address space and is available to process any other

HTTP requests received from clients. The service () method is called for each HTTP request. The doMethods call the service/process methods.

destroy()

The server will unload the servlet from its memory when it is no longer required.The server calls the destroy() method to relinquish any resources such as file handles that are allocated for the servlet; important data may be saved to a persistent store.The memory allocated for the servlet and its objects can then be garbage collected.

Examples of Servlets:

Example 1: Servlet prints a "Hello world" HTML page.

The HttpServlet is a subclass of GenericServlet, an implementation of the Servlet interface.The service () method dispatches requests to the methods doGet (), doPost (), doPut (), doDelete (), and so on; according to the HTTP request.

```
import java.io.IOException;
public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("<html>\n" +
                "<head><title>Hello World</title></head>\n" +
                "<body>\n" +
                "<h1>Hello, world!</h1>\n" +
                "</body></html>");
  }
}
```

Example 2: Servlet that displays the data obtained from Database

Communication between Servlet and Database

```
import java.io.*;
import java.net.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class s1 extends HttpServlet {
    private Connection conn;
    private Statement st;
    private ResultSet res;

    protected void processRequest(HttpServletRequest request,
                              HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try
        {

         Class.forName("com.mysql.jdbc.Driver");
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/e","root"
,"amirtha");
           st=conn.createStatement();
         res=st.executeQuery("select * from employee2");
         int i=0;

           out.println("<html>");
           out.println("<body>");
         while(res.next())
         {
                out.println(res.getInt(1) );
                out.println( res.getString(2)+ "<br> ");


         }
         out.println("</body></html>");
         out.close();
         st.close();
         conn.close();
}catch (ClassNotFoundException err) { }
              catch (SQLException err) {  }

   }
  protected void doGet(HttpServletRequest request, HttpServletResponse
  response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }


}
```

<u>Example 3: Servlet to display processingclient form information and displaying the output in client window</u>

<u>Client Server Communication- CLIENT FORM TO SERVER SERVLET</u>
<u>HTML Form</u>

```
<html>
<head>
<title>New Page 1</title>
</head>
<body>
<h2>Login</h2>
<p>Please enter your username and password</p>
<form method="POST" action="vs">
  <p> Username  <input type="text" name="username" size="20"></p>
  <p> Password  <input type="text" name="password" size="20"></p>
  <p><input type="submit" value="Submit" name="B1"></p>
</form>
<p>&nbsp;</p>
</body>
</html>
```

vs is the sevlet class file

SERVLET PROCESSING CLIENT REQUEST

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

public class vs extends HttpServlet{
  public void doPost(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException, IOException {
  response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String name = request.getParameter("username");
  String pass = request.getParameter("password");

  out.println("<html>");
  out.println("<body>");
  out.println("Thanks  Ms./Mr." + "  " + name + "  "
  + "from VIT<br>" );
  out.println("your password is : "
  + "  " + pass + "<br>");
  out.println("</body></html>");

  }
}
```

EXAMPLE 4: Client Form Data stored in database using Servlet
CLIENT FORM TO SERVER SERVLET TO DATABASE

CLIENT FORM WITH VALIDATION USING JAVASCRIPT

```
<html>
<head>
<script>
var flag = false;
var flag1 = false;
function validate()
{
   var s = form1.upwd.value;
   var repwd1 = /^\w{6,8}$/;
   if ( !repwd1.test(s) )
   {
      alert("enter a alphanumric set of min 6 or max 8 chars");
       form1.upwd.value= "  ";
   }
   else
      flag = true;
}
function check()
{
   if ( form1.upwd.value != form1.cpwd.value )
   {
        alert("password doesn't match")
        form1.cpwd.value= " ";
   }
   else
   flag = true;
}
function check()
{
   if ( form1.upwd.value != form1.cpwd.value )
   {
        alert("password doesn't match")
        form1.cpwd.value= " ";
   }
   else
     flag1 = true;
}
</script>
</head>
<body>
<form name ="form1" action="Registration" method="GET">
<table border ="1" width=30px height=30px>
<tr><td>
<strong>User_Name:</strong>
<input type="text" name="user"></td></tr>
<tr><td><strong>Password:</strong>
<input type="password" name="upwd" onblur="validate()" ></td></tr>
<tr><td><strong>Confirm_Password:</strong>
<input type="password" name="cpwd" onblur="check()"></td></tr>
</table>
<input type="submit" name="button" value ="finalize">
</form>
</body>
</html>
```

SERVLET CODE

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

public class Registration extends HttpServlet {


    protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/e","root",
"amirtha");
         PreparedStatement st=con.prepareStatement("insert into re
values (?,?)" );
         st.setString( 1 , request.getParameter("user") );
         st.setString( 2 , request.getParameter("upwd"));

         st.executeUpdate();
         con.commit();
         st.close();
         con.close();
       }catch (ClassNotFoundException err) { System.out.println(err); }
        catch (SQLException err) { System.out.println(err); }


        out.close();
    }

    }
```