Java Servlet

Introduction

To know the advantage of Servlet we need to know some of the basics related to client-server communication in the web. Consider a request for a static web page. A user enters a Uniform Resource Locator (URL) into a browser. The browser generates an HTTP request to the appropriate web server. The web server maps this request to a specific file. That file is returned in an HTTP response to the browser. The HTTP header in the response indicates the type of the content. The Multipurpose Internet Mail Extensions (MIME) is used for this purpose.

Now consider dynamic content. Assume that an online store uses a database to store information about its business. This would include items for sale, prices, availability, orders, and so forth. It wishes to make this information accessible to customers via web pages. The contents of those web pages must be dynamically generated to reflect the latest information in the database.

Advantages of Servlet Over Applet

Applet is downloaded from the server, runs in the client browser. They will function properly, provided a proper JRE (Java Runtime Environment) is installed in the client browser. If the client uses old version browsers and applet uses advanced features, the browser may fail to execute.

Servlet runs on server machine and usually generates simple HTML code. So, even an old version browser can display these HTML pages correctly.

Applet can’t access local resources such as files and databases. If Servlets are configured properly, you can access system resources. So that Servlets are more powerful than Applet.

There are several alternatives to the Servlets. However, each has its own set of set of problems. Some of the server side technologies are Common Gateway Interface (CGI), Active Server Pages (ASP) and PHP.

Common Gateway Interface (CGI) Vs Servlet

In the early days of the Web, a server could dynamically construct a page by creating a separate process to handle each client request. The process would open connections to one or more databases in order to obtain the necessary information. It communicated with the web server via an interface known as the Common Gateway Interface (CGI). CGI allowed the separate process to read data from the HTTP request and write data to the HTTP response. A variety of different languages were used to build CGI programs. These included C, C++, and Perl.

However, CGI suffered serious performance problems. It was expensive in terms of processor and memory resources to create a separate process for each client request. It was also expensive to open and close database connections for each client request. In addition, the

CGI programs were not platform-independent. Therefore, other techniques were introduced. Among these are servlets.

Servlets offer several advantages in comparison with CGI.

- Performance is significantly better. Servlets execute within the address space of a web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- The Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.
- Finally, the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

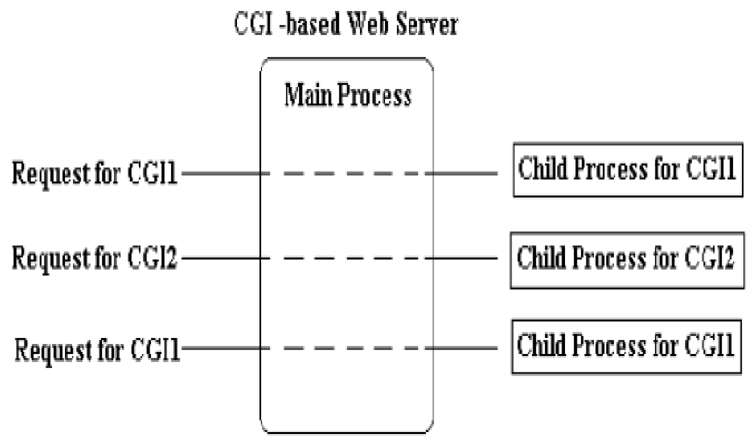
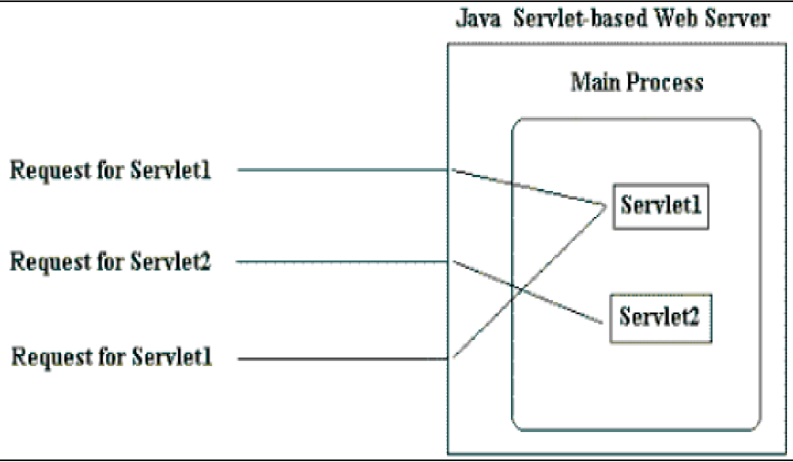
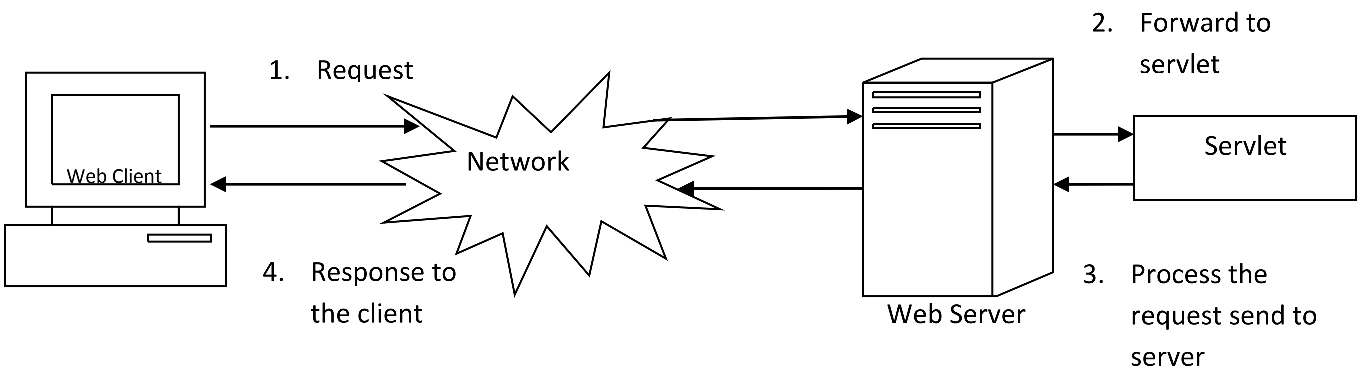
CGI Communication: For each Client request it will create a new process	 CGI -based Web Server Main Process Request for CGI1 Request for CGI2 Request for CGI1 Child Process for CGI1 Child Process for CGI2 Child Process for CGI1
Servlet Communication: After receiving the Client's first request for servlet1, web server create a new instance for that servlet class. If web server receives the second request for the same Servlet, handover the request to same instance.	 Java Servlet-based Web Server Main Process Request for Servlet1 Request for Servlet2 Request for Servlet1 Servlet1 Servlet2

Table 1: Servlet and CGI Communication Process

Servlet

Servlet is a Java technology for server side programming. It is a Java module that runs inside a java enabled web server and services the request obtained from the web server. Servlets are not tied to specific Client-Server protocol, but are mostly used with HTTP. Servlet Execution process consists of four steps:



1) Client sends a request to the web server.
2) Web server interprets it and forwards it to the corresponding Servlet.
3) Servlet process the request, generates the output, and sends it back to the web server.
4) Web server sends the response back to the client, the browser then displays it on the screen.

Note: Servlet runs with in the Java Virtual Machine. Since it runs on the server and generates simple output. Which is even an old version browser can interpret, So there is no browser compatibility issues.

Advantages of Servlet

- ✓ **Efficient:** When a Servlet gets loaded in the server, it remains in memory as a single object. Each new request is then served by light weight java thread spawned from the Servlet. This is much more efficient technique than creating new process for every request, as done in CGI.
- ✓ **Persistent:** Servlet can maintain session by using session tracking/Cookies. A mechanism that helps, to track information from request to request. Critical information can also be saved to persistent storage and can retrieve from it when the Servlet loaded next time.
- ✓ **Portable (WORA):** Servlets are written in java, so it is portable across operating systems and server implementations. This allows Servlets to be moved across new operating systems. We can develop a Servlet on windows machine running the tomcat on any other Operating System, we can deploy and run the Servlet on that machine. So, Servlets are **W**rite **O**nce **R**un **A**nywhere (WORA).
- ✓ **Robust:** Since Servlets use the entire JDK, they can provide extremely robust solutions. Java has very powerful exception handling mechanism and provides a garbage collector to handle memory leaks. It also has a very large and rich class library with network and file support, database access security and DOM Support.
- ✓ **Extensible:** Servlet can also make use of java's object oriented features, especially inheritance. A sub class inherits all the features of its super class. Additional features and methods can be added and hence it is easily extendable.

- ✓ **Secure:** Servlet run in a java enabled web server. So, they can use the security mechanism from both the web server and java security manager.
- ✓ **Cost effective:** There are number of free web servers available for personal and commercial use. Some of the web servers are

Name of the web server	Organization or Company
Tomcat Server	Apache Software Foundation (ASF)
Web Sphere Application Server	IBM
Web Logic	BEA
JRUN	Adobe
Oracle Application Server	Oracle Corporation

Table 2: List of Popular Servers supports java environment

Servlet Life Cycle

The container in which Servlet has been deployed supervised and controls the life cycle of Servlet. Typically container is nothing but a web server. When the container receives a request from a client and determines that the request should be handled by Servlet, it performs the following steps.

a) If an instance of the target Servlet does not exist, the container does the following
   i) Find and loads the Servlet class
   ii) Create an instance of the Servlet class
   iii) Calls **init()** method on this Servlet instance to initialize it.
b) Otherwise (i.e., target Servlet exist), it invokes the **service()** method on it, passing a ServletRequest type object and ServletResponse type Object.
c) If the container decides that the Servlet is no longer needed, it removes and finalizes the Servlet by calling the Servlet **destroy()** method.

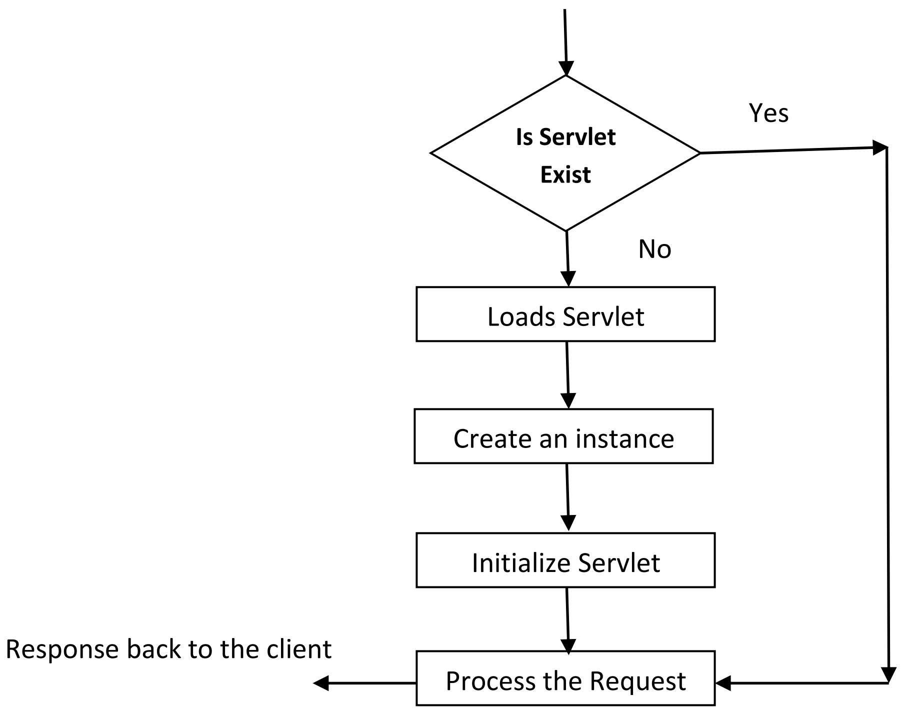


Figure 1: Servlet Life Cycle Flowchart

Life Cycle Methods:

1) **init() Method:**
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. 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. Syntax of the init method is given below:

```
public void init(ServletConfig config) throws ServletException {
  // Initialization code...
}
```

2) **service() Method:**
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{
          // Service Code
}
```

3) **destroy() Method:**
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 such as close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities. The syntax of the destroy method of the Servlet interface is given below:

```
public void destroy()  {
    //Finalize code
}
```

After the destroy() method is called, the servlet object is marked for garbage collection.

Other Methods

4) **getServletConfig() Method:**
Returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet. This is the ServletConfig object passed to the init method; the init method should have stored this object so that this method could return it.

```
public abstract ServletConfig getServletConfig()
```

5) **getServletInfo() Method:**
Returns a string containing information about the servlet, such as its author, version, and copyright. As this method may be called to display such information in an administrative tool that is servlet engine specfic, the string that this method returns should be plain text and not contain markup.

```
public abstract String getServletInfo()
```

Servlet API

Two packages contain the classes and interfaces that are required to build the Servlet. These are **javax.servlet** and **javax.servlet.http**. They constitute the core of the Servlet API. Keep in mind that these packages are not part of the Java core packages. Therefore, they are not included with Java SE. Instead, they are provided by Tomcat. They are also provided by Java EE.

The Servlet API has been in a process of ongoing development and enhancement. The current servlet specification is version 3.1. However, because changes happen fast in the world of Java, you will want to check for any additions or alterations.

- ✓ **javax.servlet** - Protocol independent classes and interfaces exist in javax.servlet package.
- ✓ **javax.servlet.http** - HTTP specific classes and interfaces exist in javax.servlet.http package.

javax.servlet package:

The javax.servlet package contains a number of interfaces and classes that establish the framework in which servlets operate. The following table summarizes several key interfaces that are provided in this package. The most significant of these is Servlet. All servlets must implement this interface or extend a class that implements the interface. The ServletRequest and ServletResponse interfaces are also very important.

Interface	Description
Servlet	Declares life cycle methods for a servlet.
ServletConfig	Allows servlets to get initialization parameters.
ServletContext	Enables servlets to log events and access information about their environment.
ServletRequest	Used to read data from a client request.
ServletResponse	Used to write data to a client response.

Table: 3 Interfaces Defined by javax.servlet package

The following table summarizes the core classes that are provided in the javax.servlet package:

Class	Description
GenericServlet	Implements the Servlet and ServletConfig interfaces.
ServletInputStream	Encapsulates an input stream for reading requests from a client.
ServletOutputStream	Encapsulates an output stream for writing responses to a client.
ServletException	Indicates a servlet error occurred.
UnavailableException	Indicates a servlet is unavailable.

Table: 4 Classes Defined by javax.servlet package

1) The Servlet Interface

All servlets must implement the Servlet interface. It declares the init( ), service( ), and destroy( ) methods that are called by the server during the life cycle of a servlet. A method is also provided that allows a servlet to obtain any initialization parameters. The methods defined by Servlet are shown in the below Table.

The init( ), service( ), and destroy( ) methods are the life cycle methods of the servlet. These are invoked by the server. The getServletConfig( ) method is called by the servlet to obtain initialization parameters. A servlet developer overrides the getServletInfo( ) method to provide a string with useful information (for example, the version number). This method is also invoked by the server.

Method	**Description**
void **destroy**( )	Called when the servlet is unloaded.
ServletConfig **getServletConfig**( )	Returns a ServletConfig object that contains any initialization parameters.
String **getServletInfo**( )	Returns a string describing the servlet.
void **init**(ServletConfig sc) throws ServletException	Called when the servlet is initialized. Initialization parameters for the servlet can be obtained from sc. A ServletException should be thrown if the servlet cannot be initialized.
void **service** (ServletRequest req, ServletResponse res) throws ServletException, IOException	Called to process a request from a client. The request from the client can be read from req. The response to the client can be written to res. An exception is generated if a Servlet or IO problem occurs.

Table: 5 Methods Defined By Servlet Interface

2) The ServletConfig Interface

The ServletConfig interface allows a servlet to obtain configuration data when it is loaded. The methods declared by this interface are summarized here:

Method	**Description**
ServletContext getServletContext( )	Returns the context for this servlet.
String getInitParameter(String param)	Returns the value of the initialization parameter named param.
Enumeration<String> getInitParameterNames( )	Returns an enumeration of all initialization parameter names.
String getServletName( )	Returns the name of the invoking servlet.

Table: 6 Methods Defined By ServletConfig Interface

3) The ServletContext Interface

The ServletContext interface enables servlets to obtain information about their environment. Several of its methods are summarized in following Table (7).

Method	Description
Object getAttribute(String attr)	Returns the value of the server attribute named attr.
void setAttribute(String attr, Object val)	Sets the attribute specified by attr to the value passed in val.
String getServerInfo( )	Returns information about the server.
void log(String s)	Writes s to the servlet log.
void log(String s, Throwable e)	Writes s and the stack trace for e to the servlet log.

Table: 7 Methods Defined By ServletContext Interface

4) The ServletRequest Interface

The ServletRequest interface enables a servlet to obtain information about a client request. Several of its methods are summarized in Table 8.

Method	Description
String getServerName( )	Returns the name of the server.
int getServerPort( )	Returns the port number.
String getProtocol( )	Returns a description of the protocol.
String getRemoteAddr( )	Returns the string equivalent of the client IP address.
String getRemoteHost( )	Returns the string equivalent of the client host name.
String getParameter(String pname)	Returns the value of the parameter named pname.
String[ ] getParameterValues(String name)	Returns an array containing values associated with the parameter specified by name.
Enumeration<String> getParameterNames( )	Returns an enumeration of the parameter names for this request.
Object getAttribute(String attr)	Returns the value of the attribute named attr.
String getCharacterEncoding( )	Returns the character encoding of the request.
int getContentLength( )	Returns the size of the request. The value –1 is returned if the size is unavailable.
String getContentType( )	Returns the type of the request. A null value is returned if the type cannot be determined.
ServletInputStream getInputStream( ) throws IOException	Returns a ServletInputStream that can be used to read binary data from the request. An IllegalStateException is thrown if getReader( ) has been previously invoked on this object.
BufferedReader getReader( ) throws IOException	Returns a buffered reader that can be used to read text from the request. An IllegalStateException is thrown if getInputStream( ) has been previously invoked on this object.

Table: 8 Methods Defined By ServletRequest Interface

5) The ServletResponse Interface

The ServletResponse interface enables a servlet to formulate a response for a client. Several of its methods are summarized in Table 9.

Method	Description
String getCharacterEncoding( )	Returns the character encoding for the response.
ServletOutputStream getOutputStream( ) throws IOException	Returns a ServletOutputStream that can be used to write binary data to the response. An IllegalStateException is thrown if getWriter( ) has been previously invoked on this object.

PrintWriter getWriter( ) throws IOException	Returns a PrintWriter that can be used to write character data to the response. An IllegalStateException is thrown if getOutputStream( ) has been previously invoked on this object.
void setContentLength(int size)	Sets the content length for the response to size.
void setContentType(String type)	Sets the content type for the response to type.

Table: 9 Methods Defined By ServletResponse Interface

Classes in javax.servlet Package

a) The GenericServlet Class

The GenericServlet class provides implementations of the basic life cycle methods for a servlet. GenericServlet implements the Servlet and ServletConfig interfaces. In addition, a method to append a string to the server log file is available. The signatures of this method are shown here:

- void log(String s)
- void log(String s, Throwable e)

Here, s is the string to be appended to the log, and e is an exception that occurred.

b) The ServletInputStream Class

The ServletInputStream class extends InputStream. It is implemented by the Servlet container and provides an input stream that a servlet developer can use to read the data from a client request. In addition to the input methods inherited from InputStream, a method is provided to read bytes from the stream. It is shown here:

int readLine(byte[ ] buffer, int offset, int size) throws IOException

Here, buffer is the array into which size bytes are placed starting at offset. The method returns the actual number of bytes read or –1 if an end-of-stream condition is encountered.

c) The ServletOutputStream Class

The ServletOutputStream class extends OutputStream. It is implemented by the Servlet container and provides an output stream that a servlet developer can use to write data to a client response. In addition to the output methods provided by OutputStream, it also defines the print( ) and println( ) methods, which output data to the stream.

d) The Servlet Exception Classes

javax.servlet defines two exceptions. The first is ServletException, which indicates that a servlet problem has occurred. The second is UnavailableException, which extends ServletException. It indicates that a servlet is unavailable.

First Servlet:

The basic steps are the following:

1. Create and compile the servlet source code. Then, copy the servlet's class file to the proper directory, and add the servlet's name and mappings to the proper web.xml file.
2. Start Tomcat.
3. Start a web browser and request the servlet.

Create and Compile the Servlet Source Code To begin, create a file named HelloServlet.java that contains the following program:

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet {
public void service(ServletRequest request, ServletResponse response)
                throws ServletException, IOException {

        response.setContentType("text/html");
                PrintWriter pw = response.getWriter();
                pw.println("<B>Hello VRSEC!");
                pw.close();
        }
  }
```

Let's look closely at this program. First, note that it imports the **javax.servlet** package. This package contains the classes and interfaces required to build servlets. Next, the program defines HelloServlet as a subclass of GenericServlet. Inside HelloServlet, the **service( )** method (which is inherited from GenericServlet) is overridden. This method handles requests from a client. Notice that the first argument is a **ServletRequest** object. This enables the servlet to read data that is provided via the client request. The second argument is a **ServletResponse** object. This enables the servlet to formulate a response for the client.

The call to **setContentType( )** establishes the MIME type of the HTTP response. In this program, the MIME type is text/html. This indicates that the browser should interpret the content as HTML source code. Next, the **getWriter( )** method obtains a PrintWriter. Anything written to this stream is sent to the client as part of the HTTP response. Then println( ) is used to write some simple HTML source code as the HTTP response.

Compile this source code and place the HelloServlet.class file in the proper Tomcat directory as described in the previous section. Also, add HelloServlet to the web.xml file.

Reading Servlet Parameters

The ServletRequest interface includes methods that allow you to read the names and values of parameters that are included in a client request. We will develop a servlet that illustrates their use. The example contains two files. A web page is defined in **PostParameters.html** and a servlet is defined in **PostParametersServlet.java**.

The HTML source code for PostParameters.html is shown in the following listing. It defines a table that contains two labels and two text fields. One of the labels is Employee and the other is Phone. There is also a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

PostParameters.html

```
<html>
  <body>
    <center>
        <form method="post" action="http://localhost:8080/Test/PostParametersServlet">
            <table>
              <tr>
                        <td><B>Employee</td>
                        <td><input type=textbox name="e" size="25" value=""></td>
              </tr>
               <tr>
                        <td><B>Phone</td>
                        <td><input type=textbox name="p" size="25" value=""></td>
               </tr>
            </table>
                        <input type=submit value="Submit">
            </body>
            </html>
```

The source code for PostParametersServlet.java is shown in the following listing. The service() method is overridden to process client requests. The getParameterNames() method returns an enumeration of the parameter names. These are processed in a loop. You can see that the parameter name and value are output to the client. The parameter value is obtained via the getParameter( ) method.

PostParametersServlet.java

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

            public class PostParametersServlet extends GenericServlet {
```

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

    PrintWriter pw = response.getWriter();
    Enumeration e = request.getParameterNames();

        while(e.hasMoreElements())
        {
                String pname = (String)e.nextElement();
                pw.print(pname + " = ");
                String pvalue = request.getParameter(pname);
                pw.println(pvalue);
        }
        pw.close();
        }
        }
```

Compile the servlet. Next, copy it to the appropriate directory, and update the web.xml file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat (if it is not already running).
2. Display the web page in a browser.
3. Enter an employee name and phone number in the text fields.
4. Submit the web page.

After following these steps, the browser will display a response that is dynamically generated by the servlet.

Example# 2

The ServletRequest interface includes methods that allow you to read the names and values of parameters that are included in a client request. We will develop a servlet that illustrates their use. The example contains two files. A web page is defined in **Login.html** and a servlet is defined in AccessParam.java.

The HTML source code for **Login.html** is shown in the following listing. It contains two text fields and One login Button (Submit). One of the textbox is for UserName and the other is for Password. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

Login.html

```
<html>
  <head>
       <title> MY Login </title>
  </head>
  <body>
          <form name="login" method="POST" action="http://localhost:8080/TEST/Login">
                    User Name : <input type="text" name="uid" value=""/><br><br>
                    Password    : <input type="password" name="pwd" value=""/><br><br>
                    <input type="submit" name="sub" value="login">
          </form>
  </body>
</html>
```

The source code for **AccessParam.java** is shown in the following listing. The service() method is overridden to process client requests. The getParameter(ParameterName) method returns a String. By using this method userName and password values are retrieved from the ServletRequest Object. And compare these two values with predefined values, if those two are equal then returns welcome screen otherwise returns an error message.

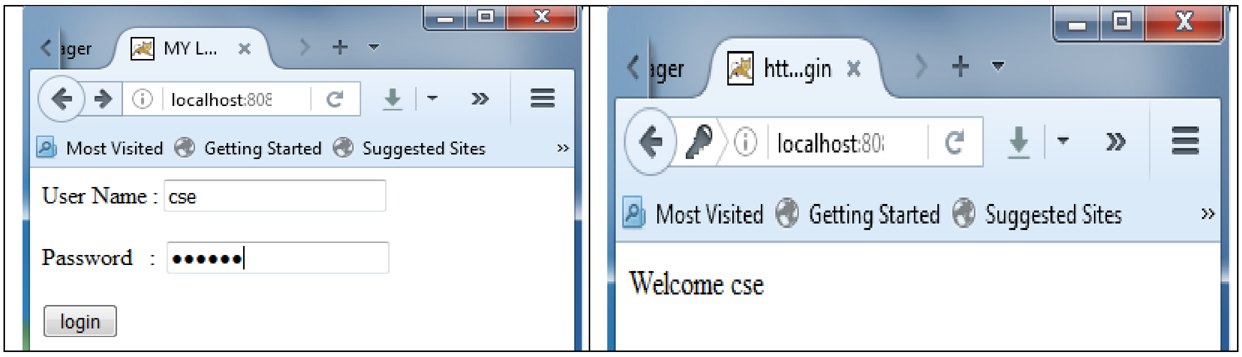
Accessing Parameter:

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

public class AccessParam extends GenericServlet
{
     public void service(ServletRequest req, ServletResponse res) throws ServletException,
IOException
     {
          res.setContentType("text/html");
          PrintWriter pw = res.getWriter();

          String userName = req.getParameter("uid");
          String passKey = req.getParameter("pwd");

          if(userName.equals("cse") && passKey.equals("abc123"))
            pw.println("Welcome " +userName);
          else
            pw.println("Invalid User Name or Password ....!");
          pw.close();

     }
}
```

Web.xml

```
<web-app>
    <servlet>
        <servlet-name>Login1</servlet-name>
        <servlet-class>AccessParam</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Login1</servlet-name>
        <url-pattern>/Login</url-pattern>
    </servlet-mapping>
</web-app>
```

OUTPUT:

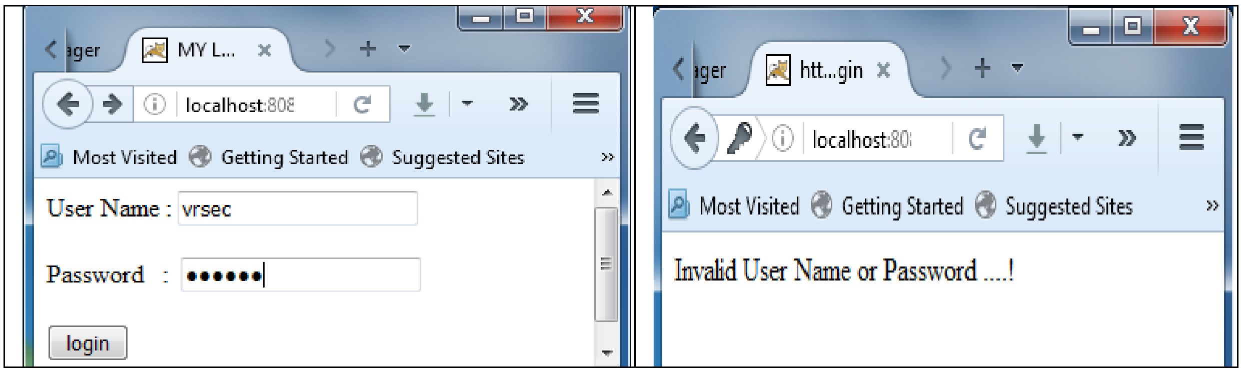


User Name : cse

Password : ••••••

Welcome cse

Reading Servlet Parameters: Validate the user (valid User)



User Name : vrsec

Password : ••••••

Invalid User Name or Password ....!

Reading Servlet Parameters: Validate the user (In-valid User)

Reading Initialization Parameters:

The Servlet container provides the initialization parameters for a Servlet or filter within the configuration object the container passes to the init method. The configuration object provides a **getInitParameter()** method that takes a parameter- Parameter name as a string and returns the contents of the initialization parameter.

Login.html

```
<html>
  <head>
        <title> MY Login </title>
  </head>
  <body>
       <form name="login" method="POST" action="http://localhost:8080/TEST/parameters">
                        User Name : <input type="text" name="uid" value=""/><br><br>
                        Password  &nbsp;&nbsp;:&nbsp; <input type="password" name="pwd"
value=""/><br><br>
                        <input type="submit" name="sub" value="login">
            </form>
  </body>
</html>
```

Initialization Parameters: (Servlet Class)

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

public class InitParm extends GenericServlet
{
      String default_user,default_pwd;
            public void init(ServletConfig sc)
                  {
                        default_user = sc.getInitParameter("userName");
                        default_pwd = sc.getInitParameter("userpwd");
                    }

           public void service(ServletRequest req, ServletResponse res) throws
                                                  ServletException, IOException
               {
                        res.setContentType("text/html");
                        PrintWriter pw = res.getWriter();
                        String uid = req.getParameter("uid");
                        String pkey = req.getParameter("pwd");

           if(default_user.equals(uid) && default_pwd.equals(pkey))
                  pw.println("Welcome " +uid);
            else
                  pw.println("Invalid Userid or password...........!");
            pw.close();
      }
}
```

Reading initialization parameters using **getInitParameter(pName)** method from ServletConfig interface Object. Read users/Client input using **getParameter(pName)** method from ServletRequest interface Object. In our example **default_user** and **default_pwd** are read by using getInitParameter() method and users input i.e., **uid** and **pwd** are read by using getParameter(). Compare these values and validate the users input. If validation fails then gets error message otherwise get welcome screen.

Web.xml:

```
<web-app>
        <servlet>
                    <servlet-name>init</servlet-name>
                    <servlet-class> InitParm</servlet-class>
                    <init-param>
                              <param-name>userName</param-name>
                              <param-value>csephani</param-value>
                    </init-param>
                    <init-param>
                              <param-name>userpwd</param-name>
                              <param-value>abc123</param-value>
                    </init-param>
          </servlet>
          <servlet-mapping>
                     <servlet-name>init</servlet-name>
                     <url-pattern>/parameters</url-pattern>
          </servlet-mapping>
</web-app>
```

The javax.servlet.http Package

The preceding examples have used the classes and interfaces defined in **javax.servlet**, such as ServletRequest, ServletResponse, and GenericServlet, to illustrate the basic functionality of servlets. However, when working with HTTP, you will normally use the interfaces and classes in **javax.servlet.http**. As you will see, its functionality makes it easy to build Servlets that work with HTTP requests and responses.

Interface	**Description**
HttpServletRequest	Enables Servlets to read data from an HTTP request.
HttpServletResponse	Enables Servlets to write data to an HTTP response.
HttpSession	Allows session data to be read and written.

The following table summarizes the classes used in this chapter. The most important of these is HttpServlet. Servlet developers typically extend this class in order to process HTTP requests.

Class	**Description**
Cookie	Allows state information to be stored on a client machine.
HttpServlet	Provides methods to handle HTTP requests and responses.

1) The HttpServletRequest Interface

The HttpServletRequest interface enables a servlet to obtain information about a client request. Several of its methods are shown in Table 10.

Method	**Description**
String getAuthType( )	Returns authentication scheme.
Cookie[ ] getCookies( )	Returns an array of the cookies in this request.
String getHeader(String field)	Returns the value of the header field named field.
int getIntHeader(String field)	Returns the int equivalent of the header field named field.
String getMethod( )	Returns the HTTP method for this request.
String getPathInfo( )	Returns any path information that is located after the servlet path and before a query string of the URL.
String getQueryString( )	Returns any query string in the URL.
String getRemoteUser( )	Returns the name of the user who issued this request.
String getRequestedSessionId( )	Returns the ID of the session.
String getRequestURI( )	Returns the URI.
StringBuffer getRequestURL( )	Returns the URL.
String getServletPath( )	Returns that part of the URL that identifies the servlet.
HttpSession getSession( )	Returns the session for this request. If a session does not exist, one is created and then returned.
HttpSession getSession(boolean new)	If new is true and no session exists, creates and returns a session for this request. Otherwise, returns the existing session for this request.
boolean isRequestedSessionIdFromCookie( )	Returns true if a cookie contains the session ID. Otherwise, returns false.
boolean isRequestedSessionIdFromURL( )	Returns true if the URL contains the session ID. Otherwise, returns false.
boolean isRequestedSessionIdValid( )	Returns true if the requested session ID is valid in the current session context.

Table: 10 Methods Defined By HttpServletResquest Interface

2) The HttpServletResponse Interface

The HttpServletResponse interface enables a servlet to formulate an HTTP response to a client. Several constants are defined. These correspond to the different status codes that can be assigned to an HTTP response.

Method	Description
void addCookie(Cookie cookie)	Adds cookie to the HTTP response.
boolean containsHeader(String field).	Returns true if the HTTP response header contains a field named field
String encodeURL(String url)	Determines if the session ID must be encoded in the URL identified as url. If so, returns the modified version of url. Otherwise, returns url. All URLs generated by a servlet should be processed by this method.
void sendError(int c) throws IOException	Sends the error code c to the client.
void sendError(int c, String s) throws IOException	Sends the error code c and message s to the client.
void sendRedirect(String url) throws IOException	Redirects the client to url.
void setHeader(String field, String value)	Adds field to the header with value equal to value.
void setIntHeader(String field, int value)	Adds field to the header with value equal to value.
void setStatus(int code)	Sets the status code for this response to code.

Table: 11 Methods Defined By HttpServletResponse Interface

3) HTTPSession interface

The HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session. Several of its methods are summarized in Table 12. All of these methods throw an IllegalStateException if the session has already been invalidated.

Method	Description
Object getAttribute(String attr)	Returns the value associated with the name passed in attr. Returns null if attr is not found.
Enumeration<String> getAttributeNames( )	Returns an enumeration of the attribute names associated with the session.
long getCreationTime( )	Returns the creation time (in milliseconds since midnight, January 1, 1970, GMT) of the invoking session.
String getId( )	Returns the session ID.
long getLastAccessedTime( )	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the client last made a request on the invoking session.
void invalidate( )	Invalidates this session and removes it from the context.
boolean isNew( )	Returns true if the server created the session and it has not yet been accessed by the client.
void removeAttribute(String attr)	Removes the attribute specified by attr from the session.
void setAttribute(String attr, Object val)	Associates the value passed in val with the attribute name passed in attr.

Table: 12 Methods Defined By HttpSession Interface

HttpServlet Class

The HttpServlet class extends GenericServlet. It is commonly used when developing servlets that receive and process HTTP requests. The methods defined by the HttpServlet class are summarized in Table 13.

Method	Description
void **doDelete**(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP DELETE request.
void **doGet**(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP GET request
void **doHead**(HttpServletRequest req,HttpServletResponse res) throws IOException, ServletException	Handles an HTTP HEAD request.
void **doOptions**(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP OPTIONS request
void **doPost**(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP POST request.
void **doPut**(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP PUT request
void **doTrace**(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Handles an HTTP TRACE request.
long **getLastModified**(HttpServletRequest req)	Returns the time (in milliseconds since midnight, January 1, 1970, GMT) when the requested resource was last modified.
void **service**(HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Called by the server when an HTTP request arrives for this servlet. The arguments provide access to the HTTP request and response, respectively.

Table: 13 Methods Defined By HttpServlet Class

The Cookie Class

The Cookie class encapsulates a cookie. A **cookie** is stored on a client and contains state information. Cookies are valuable for tracking user activities. For example, assume that a user visits an online store. A cookie can save the user's name, address, and other information. The user does not need to enter this data each time he or she visits the store.

A servlet can write a cookie to a user's machine via the addCookie( ) method of the HttpServletResponse interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser. The names and values of cookies are stored on the user's machine. Some of the information that can be saved for each cookie includes the following:

- ✓ The name of the cookie
- ✓ The value of the cookie
- ✓ The expiration date of the cookie
- ✓ The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user’s machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends.

Method	**Description**
Object clone( )	Returns a copy of this object.
String getComment( )	Returns the comment.
String getDomain( )	Returns the domain.
int getMaxAge( )	Returns the maximum age (in seconds).
String getName( )	Returns the name.
String getPath( )	Returns the path.
String getValue( )	Returns the value.
int getVersion( )	Returns the version.
boolean isHttpOnly( )	Returns true if the cookie has the HttpOnly attribute.
void setComment(String c)	Sets the comment to c.
void setDomain(String d)	Sets the domain to d.
void setMaxAge(int secs)	Sets the maximum age of the cookie to secs. This is the number of seconds after which the cookie is deleted.
void setPath(String p)	Sets the path to p.
void setSecure(boolean secure)	Sets the security flag to secure.
void setValue(String v)	Sets the value to v.
void setVersion(int v)	Sets the version to v.

Table: 14 Methods Defined By Cookie Class

The domain and path of the cookie determine when it is included in the header of an HTTP request. If the user enters a URL whose domain and path match these values, the cookie is then supplied to the web server. Otherwise, it is not. There is one constructor for Cookie. It has the signature shown here:

Cookie(String name, String value)

Here, the name and value of the cookie are supplied as arguments to the constructor. The methods of the Cookie class are summarized in Table 14.

Develop a servlet that illustrates how to use cookies. The servlet is invoked when a form on a web page is submitted. The example contains three files as summarized here:

File	Description
AddCookie.html	Allows a user to specify a value for the cookie named MyCookie.
AddCookieServlet.java	Processes the submission of AddCookie.html.
GetCookiesServlet.java	Displays cookie values.

Table: 14 List of Files to create and Access the Cookie using Servlets

The HTML source code for AddCookie.html is shown in the following listing. This page contains a text field in which a value can be entered. There is also a submit button on the page. When this button is pressed, the value in the text field is sent to AddCookieServlet via an HTTP POST request.

AddCookie.html

```
<html>
<body>
<form name="Form1" method="post" action="http://localhost:8080/Test/AddCookieServlet">
<B>Enter a value for MyCookie:</B>
            <input type=textbox name="data" size=25 value="">
            <input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for AddCookieServlet.java is shown in the following listing. It gets the value of the parameter named "data". It then creates a Cookie object that has the name "MyCookie" and contains the value of the "data" parameter. The cookie is then added to the header of the HTTP response via the addCookie( ) method. A feedback message is then written to the browser.

AddCookieServlet.java

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

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

String data = request.getParameter("data");
Cookie cookie = new Cookie("MyCookie", data);          // Create cookie.
response.addCookie(cookie);                            // Add cookie to HTTP response.

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

```
                    pw.println("<B>MyCookie has been set to");
                    pw.println(data);
                    pw.close();
          }
}
```

The source code for GetCookiesServlet.java is shown in the following listing. It invokes the getCookies( ) method to read any cookies that are included in the HTTP GET request. The names and values of these cookies are then written to the HTTP response. Observe that the getName( ) and getValue( ) methods are called to obtain this information.

GetCookiesServlet.java

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

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

Cookie[] cookies = request.getCookies();       // Get cookies from header of HTTP request.

          response.setContentType("text/html");
          PrintWriter pw = response.getWriter();
          pw.println("<B>");
                    for(int i = 0; i < cookies.length; i++) {
                              String name = cookies[i].getName();
                              String value = cookies[i].getValue();
                              pw.println("name = " + name + "; value = " + value);
                         }
                    pw.close();
           }
 }
```

Compile the servlets. Next, copy them to the appropriate directory, and update the web.xml file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat, if it is not already running.
2. Display AddCookie.html in a browser.
3. Enter a value for MyCookie.
4. Submit the web page.

After completing these steps, you will observe that a feedback message is displayed by the browser. Next, request the following URL via the browser:

http://localhost:8080/Test/GetCookiesServlet

Observe that the name and value of the cookie are displayed in the browser. In this example, an expiration date is not explicitly assigned to the cookie via the setMaxAge( ) method of Cookie. Therefore, the cookie expires when the browser session ends. You can experiment by using setMaxAge( ) and observe that the cookie is then saved on the client machine.

Handling HTTP Requests and Responses

The HttpServlet class provides specialized methods that handle the various types of HTTP requests. A servlet developer typically overrides one of these methods. These methods are doDelete( ), doGet( ), doHead( ), doOptions( ), doPost( ), doPut( ), and doTrace( ). However, the GET and POST requests are commonly used when handling form input. Therefore, this section presents examples of these cases.

Handling HTTP GET Requests

Here we will develop a servlet that handles an HTTP GET request. The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in **ColorGet.html**, and a servlet is defined in ColorGetServlet.java. The HTML source code for ColorGet.html is shown in the following listing. It defines a form that contains a select element and a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies a servlet to process the HTTP GET request.

ColorGet.html

```
<html>
<body>
        <form name="Form1" action="http://localhost:8080/Test/ColorGetServlet">
<B>Color:</B>
        <select name="color" size="1">
                <option value="Red">Red</option>
                <option value="Green">Green</option>
                <option value="Blue">Blue</option>
        </select>
        <input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for ColorGetServlet.java is shown in the following listing. The doGet( ) method is overridden to process any HTTP GET requests that are sent to this servlet. It uses the getParameter( ) method of HttpServletRequest to obtain the selection that was made by the user. A response is then formulated.

ColorGetServlet.java

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

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

                    response.setContentType("text/html");
                    PrintWriter pw = response.getWriter();
                    String color = request.getParameter("color");
                    pw.println("<B>The selected color is: ");
                    pw.println(color);
                    pw.close();
          }
}
```

Compile the servlet. Next, copy it to the appropriate directory, and update the web.xml file, as previously described. Then, perform these steps to test this example:

1. Start Tomcat, if it is not already running.
2. Display the web page in a browser.
3. Select a color.
4. Submit the web page.

After completing these steps, the browser will display the response that is dynamically generated by the servlet. One other point: Parameters for an HTTP GET request are included as part of the URL that is sent to the web server. Assume that the user selects the red option and submits the form. The URL sent from the browser to the server is

http://localhost:8080/Test/ColorGetServlet?color=Red

The characters to the right of the question mark are known as the query string.

Handling HTTP POST Requests

Here we will develop a servlet that handles an HTTP POST request. The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in ColorPost.html, and a servlet is defined in ColorPostServlet.java.

The HTML source code for ColorPost.html is shown in the following listing. It is identical to ColorGet.html except that the method parameter for the form tag explicitly specifies that the POST method should be used, and the action parameter for the form tag specifies a different servlet.

ColorPost.html

```
<html>
<body>
 <form name="Form1" method="post" action="http://localhost:8080/Test/ColorPostServlet">

<B>Color:</B>
                    <select name="color" size="1">
                    <option value="Red">Red</option>
                    <option value="Green">Green</option>
                    <option value="Blue">Blue</option>
          </select>
<br><br>
     <input type=submit value="Submit">
</form>
</body>
</html>
```

The source code for ColorPostServlet.java is shown in the following listing. The doPost( ) method is overridden to process any HTTP POST requests that are sent to this servlet. It uses the getParameter( ) method of HttpServletRequest to obtain the selection that was made by the user. A response is then formulated.

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

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

          response.setContentType("text/html");
          PrintWriter pw = response.getWriter();
          String color = request.getParameter("color");

          pw.println("<B>The selected color is: ");
          pw.println(color);
          pw.close();
     }
}
```

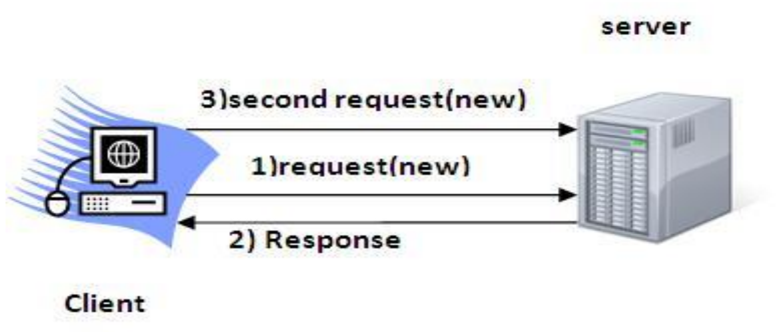
Compile the servlet and perform the same steps as described in the previous section to test it.

NOTE: Parameters for an HTTP POST request are not included as part of the URL that is sent to the web server. In this example, the URL sent from the browser to the server is http://localhost:8080/Test/ColorPostServlet. The parameter names and values are sent in the body of the HTTP request.

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 is a stateless protocol. Each request is independent of the previous one. However, in some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a server. Sessions provide such a mechanism. HTTP is stateless that means each request is considered as the new request. It is shown in the below figure.



A session can be created via the getSession( ) method of HttpServletRequest. An HttpSession object is returned. This object can store a set of bindings that associate names with objects. The setAttribute( ), getAttribute( ), getAttributeNames( ), and removeAttribute( ) methods of HttpSession manage these bindings. Session state is shared by all servlets that are associated with a client.

Session Tracking Techniques

Basically there are four techniques which can be used to identify a user session.

1. Cookies
2. Hidden Fields
3. URL Rewriting
4. HttpSession

a) Cookies

Cookie is a key value pair of information, sent by the server to the browser and then browser sends back this identifier to the server with every request there on. There are two types of cookies:

- ✓ **Non Persistent cookies**: are temporary cookies and are deleted as soon as user closes the browser. The next time user visits the same website, server will treat it as a new client as cookies are already deleted.
- ✓ **Persistent cookies:** remains on hard drive until we delete them or they expire.

If cookie is associated with the client request, server will associate it with corresponding user session otherwise will create a new unique cookie and send back with response. Simple code snippet to create a cookie with name sessionId with a unique value for each client:

```
Cookie cookie = new Cookie(“sessionID”, “some unique value”);
response.addCookie(cookie);
```

User can disable cookie support in a browser and in that case server will not be able to identify the user so this is the major disadvantage of this approach.

Advantage of Cookies

- ✓ Simplest technique of maintaining the state.
- ✓ Cookies are maintained at client side.

Disadvantage of Cookies

- ✓ It will not work if cookie is disabled from the browser.
- ✓ Only textual information can be set in Cookie object.

Example on Cookies:

index.html

```
<html>
<body>
<form action="http://localhost:8080/Test/Create" method="post">
        Name:<input type="text" name="userName"/><br/>
                <input type="submit" value="go"/>
</form>
</body>
</html>
```

CreateCookie.java

```
   import java.io.*;
   import javax.servlet.*;
   import javax.servlet.http.*;
       public class CreateCookie 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

                      out.print("<form action=' http://localhost:8080/Test/Access'>");
                      out.print("<input type='submit' value='go'>");
                      out.print("</form>");
                      out.close();
                }
catch(Exception e){
System.out.println(e);
  }  }  }
```

AccessCookie.java

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

public class AccessCookie 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>Servlet1</servlet-name>
                <servlet-class>CreateCookie</servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>Servlet1</servlet-name>
                <url-pattern>/Create</url-pattern>
        </servlet-mapping>

        <servlet>
                <servlet-name>Servlet2</servlet-name>
                <servlet-class>AccessCookie</servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name>Servlet2</servlet-name>
                <url-pattern>/Access</url-pattern>
        </servlet-mapping>
</web-app>
```

b) Hidden Field

Hidden fields are the input fields which are not displayed on the page but its value is sent to the servlet as other input fields. For example

<input type=”**hidden**” name=”sessionId” value=”unique value”/>

is a hidden form field which will not displayed to the user but its value will be send to the server and can be retrieved using request.getParameter (“sessionId”) in Servlet. As we cannot hardcode the value of hidden field created for session tracking purpose, which means we cannot use this approach for static pages like HTML. In short with this approach, HTML pages cannot participate in session tracking with this approach.

Advantages:

- ✓ Does not have to depend on browser whether the cookie is disabled or not.
- ✓ Inserting a simple HTML Input field of type hidden is required. Hence, its easier to implement.

Disadvantages:

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

Example:

index.html

```
<form action="http://localhost:8080/Test/servlet1">
        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 doGet(HttpServletRequest request, HttpServletResponse response){
     try{
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();

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

          //creating form that have invisible textfield
          out.print("<form action=' http://localhost:8080/Test/servlet2'>");
          out.print("<input type='hidden' name='uname' value='"+n+"'>");
```

```
                    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 doGet(HttpServletRequest request, HttpServletResponse response)
            try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();

            //Getting the value from the hidden field
            String n=request.getParameter("uname");
            out.print("Hello "+n);
            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>
```

c) 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

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

Disadvantage of URL Rewriting

- ✓ It will work only with links.
- ✓ It can send Only textual information.

Example of using URL Rewriting

In this example, we are maintaning the state of the user using link. For this purpose, we are appending the name of the user in the query string and getting the value from the query string in another page.

<u>index.html</u>

```
<form action="servlet1">
    Name:  <input type="text" name="userName"/><br/>
           <input type="submit" value="go"/>
</form>
```

<u>FirstServlet.java</u>

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

public class FirstServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{
                response.setContentType("text/html");
                PrintWriter out = response.getWriter();
                String n=request.getParameter("userName");
                out.print("Welcome "+n);
                //appending the username in the query string
                out.print("<a href='servlet2?uname="+n+"'> visit </a>");
                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 doGet(HttpServletRequest request, HttpServletResponse response)
            try{
                    response.setContentType("text/html");
                    PrintWriter out = response.getWriter();

                    //getting value from the query string
                    String n=request.getParameter("uname");
                    out.print("Hello "+n);
                    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>
```

Security issues:

Different authors write Servlets that are run using the resources of the server under supervision of web server. So, before deploying Servlets in the web server, make sure that they comes from trusted source. Certain access constraints should be imposed on the Servlet depending on their sources. The Servlet container restricts the users, who can access and who are not access the resources which are available in the web server.

In addition to these security Issues, the author of the Servlet should consider the following:

- ✓ Take sufficient care while writing the **file upload code**. If not implanted carefully, users may fill the total hard disk of the server by uploading large files.
- ✓ Review the code that **access files/database** based on the user input. For example, don't allow the users to execute arbitrary SQL Commands. If allowed, user may fire some harmful SQL Commands that can delete database tables.
- ✓ Make sure that the request comes from an **authorized user**. Don't rely on the existence of a session variable.
- ✓ Make sure that you have not used **System.exit()** method anywhere in your program. It will terminate the web server.
- ✓ Don't display **sensitive parameter** values in the web page.