

Chapter 2 Networking

Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

Java Networking Terminology

The widely used java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1) IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP

- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communication.

Visit next page for java socket programming.

Classes in java.net

- ContentHandler
- DatagramPacket
- DatagramSocket
- DatagramSocketImpl
- HttpURLConnection
- InetAddress
- MulticastSocket
- ServerSocket
- Socket
- SocketImpl
- URL
- URLConnection
- URLEncoder
- URLStreamHandler

Exceptions in java.net

- BindException
- ConnectException
- MalformedURLException
- NoRouteToHostException
- ProtocolException
- SocketException
- UnknownHostException
- UnknownServiceException

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.

Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient
{
    public static void main(String[] args)
    {
        Try
        {
            Socket s=new Socket("localhost",6666);
            DataOutputStream dout=new DataOutputStream(s.getOutputStream());
            dout.writeUTF("Hello Server");
            dout.flush();
            dout.close();
            s.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

MyServer.java

```
import java.io.*;
import java.net.*;
public class MyServer
{
    public static void main(String[] args)
    {
        Try
        {
            ServerSocket ss=new ServerSocket(6666);
            Socket s=ss.accept();//establishes connection
            DataInputStream dis=new DataInputStream(s.getInputStream());
            String str=(String)dis.readUTF();
            System.out.println("message= "+str);
            ss.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

Java URL

The **Java URL** class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example:

A URL contains many information:

1. **Protocol:** In this case, http is the protocol.
2. **Server name or IP Address:** In this case, www.javatpoint.com is the server name.
3. **Port Number:** It is an optional attribute. If we write http://www.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. **File Name or directory name:** In this case, index.jsp is the file name.

Commonly used methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class are given below.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated

Example of Java URL class

```
import java.io.*;
import java.net.*;

public class URLLDemo
{
    public static void main(String[] args)
    {
        Try
```

```

{
    URL url=new URL("http://www.javatpoint.com/sonoojaiswal/index.jsp");
    System.out.println("Protocol: "+url.getProtocol());
    System.out.println("Host Name: "+url.getHost());
    System.out.println("Port Number: "+url.getPort());
    System.out.println("File Name: "+url.getFile());
}catch(Exception e){System.out.println(e);}
}
}

```

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com etc.

Commonly used methods of InetAddress class

Method	Description
Public static InetAddress getByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	it returns the instance of InetAddress containing local host name and address.
public String getHostName()	it returns the host name of the IP address.
public String getAddress()	it returns the IP address in string format.

Example of Java InetAddress class

```

import java.io.*;
import java.net.*;
public class InetDemo
{
    public static void main(String[] args)
    {
        Try
        {
            InetAddress ip=InetAddress.getByName("www.javatpoint.com");
            System.out.println("Host Name: "+ip.getHostName());
            System.out.println("IP Address: "+ip.getAddress());
        }catch(Exception e){System.out.println(e);}
    }
}

```

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets. A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

- **DatagramSocket()** throws **SocketEeption**: it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- **DatagramSocket(int port)** throws **SocketEeption**: it creates a datagram socket and binds it with the given Port Number.
- **DatagramSocket(int port, InetAddress address)** throws **SocketEeption**: it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class

- **DatagramPacket(byte[] barr, int length)**: it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] barr, int length, InetAddress address, int port)**: it creates a datagram packet. This constructor is used to send the packets.

```
// DReceiver
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class DReceiver{
    public static void main(String[] args) throws Exception
    {
        DatagramSocket ds = new DatagramSocket(3000);
        byte[] buf = new byte[1024];
        DatagramPacket dp = new DatagramPacket(buf, 1024);
        ds.receive(dp);
        String strRecv = new String(dp.getData(), 0, dp.getLength());
        System.out.println(strRecv);
        ds.close();
    }
}

// DSender
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
```

```

public class DSender{
    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket();
        String str = "hello world";
        InetAddress ia = InetAddress.getByName("127.0.0.1");
        DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ia, 3000);
        ds.send(dp);
        ds.close();
    }
}

```

Chapter-3 Servlet and JSP

Servlet technology is used to create web application (resides at server side and generates dynamic web page).

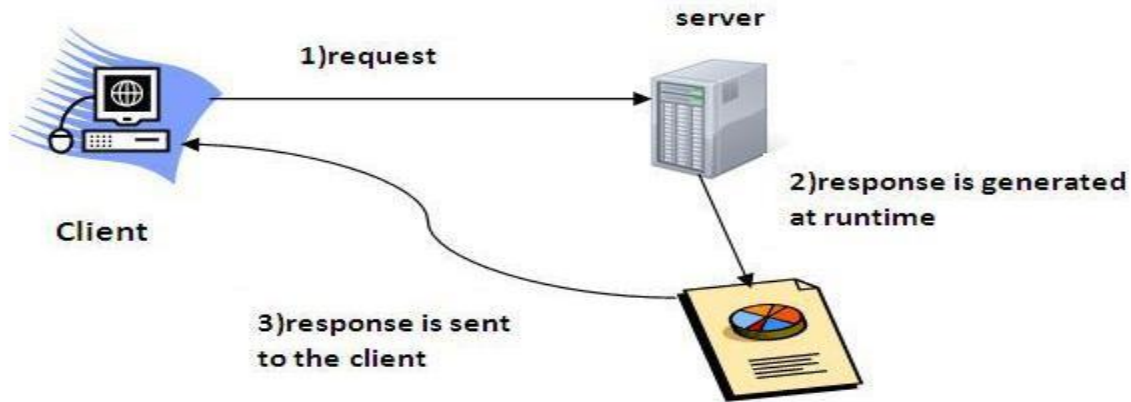
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

What is a Servlet?

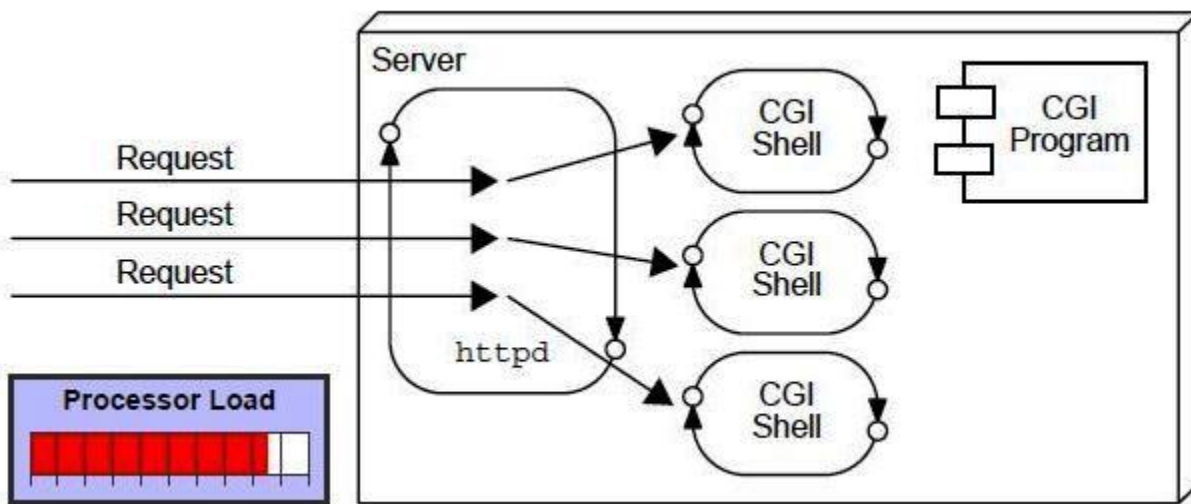
Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.



CGI(Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



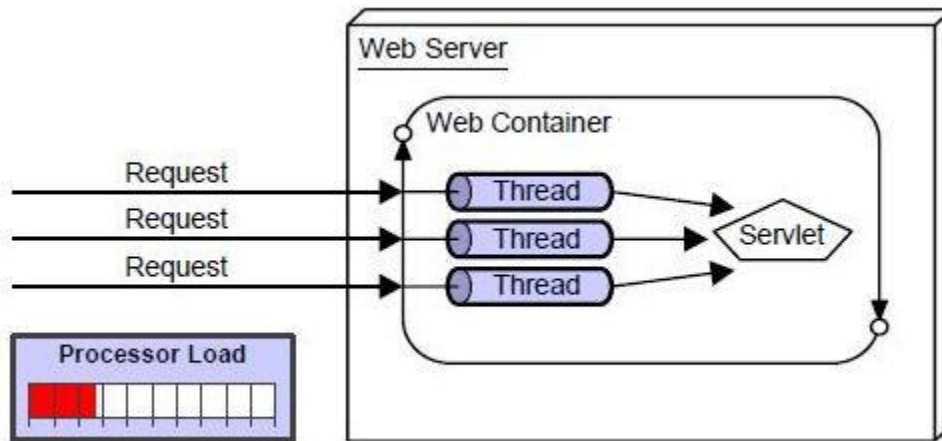
Disadvantages of CGI

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.

3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet



There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language..

Let's differentiate Servlet and CGI –

Servlet	CGI (Common Gateway Interface)
Servlets are portable	CGI is not portable.
In Servlets each request is handled by lightweight Java Thread	IN CGI each request is handled by heavy weight OS process
In Servlets, Data sharing is possible	In CGI, data sharing is not available.
Servlets can link directly to the Web	CGI cannot directly link to Web server.

Servlet	CGI (Common Gateway Interface)
server	
Session tracking and caching of previous computations can be performed	Session tracking and caching of previous computations cannot be performed
Automatic parsing and decoding of HTML form data can be performed.	Automatic parsing and decoding of HTML form data cannot be performed.
Servlets can read and Set HTTP Headers	CGI cannot read and Set HTTP Headers
Servlets can handle cookies	CGI cannot handle cookies
Servlets can track sessions	CGI cannot track sessions
Servlets is inexpensive than CGI	CGI is more expensive than Servlets

Types of servlet

GenericServlet class

GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **public abstract void service(ServletRequest request, ServletResponse response)** provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.

5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

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>");
    }
}
```

HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

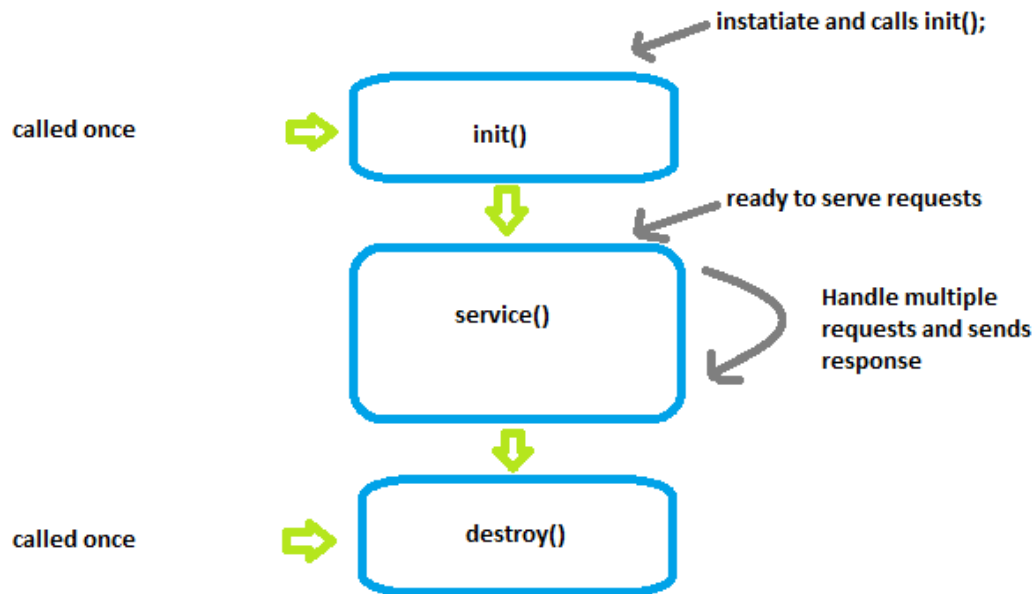
There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void delete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Life cycle of servlet

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.



Now let us discuss the life cycle methods in detail.

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 doGet, 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

There are given 6 steps to create a **servlet example**. These steps are required for all the servers.

The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

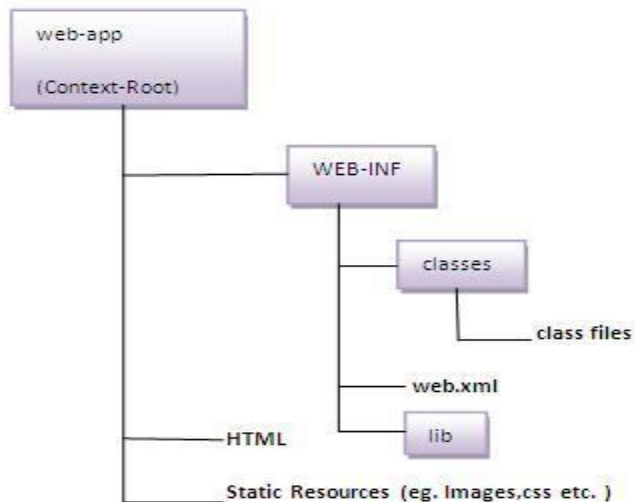
Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1) Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystems defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2) Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request.

```
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");
        pw.println("Welcome "+name);
    }
}

```

3)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```

<web-app>
    <servlet>
        <servlet-name>sonoojaiswal</servlet-name>
        <servlet-class>DemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>sonoojaiswal</servlet-name>
        <url-pattern>/welcome</url-pattern>
    </servlet-mapping>
</web-app>

```

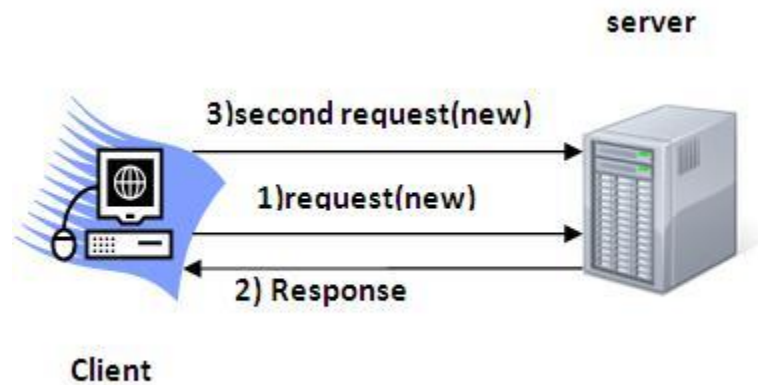
Session Tracking in Servlets

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.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

To recognize the user It is used to recognize the particular user.

Session Tracking Techniques

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

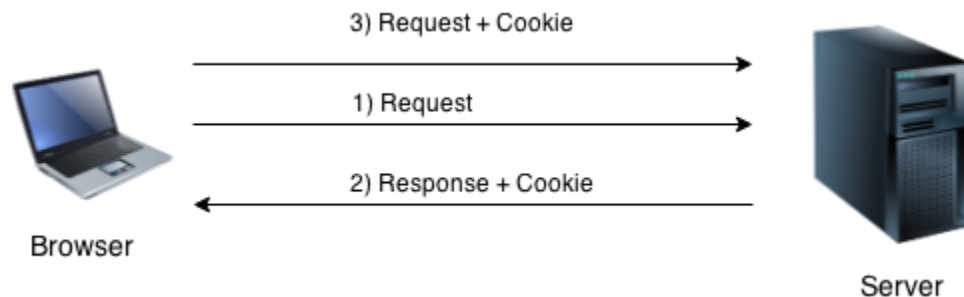
Cookies in Servlet

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.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

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

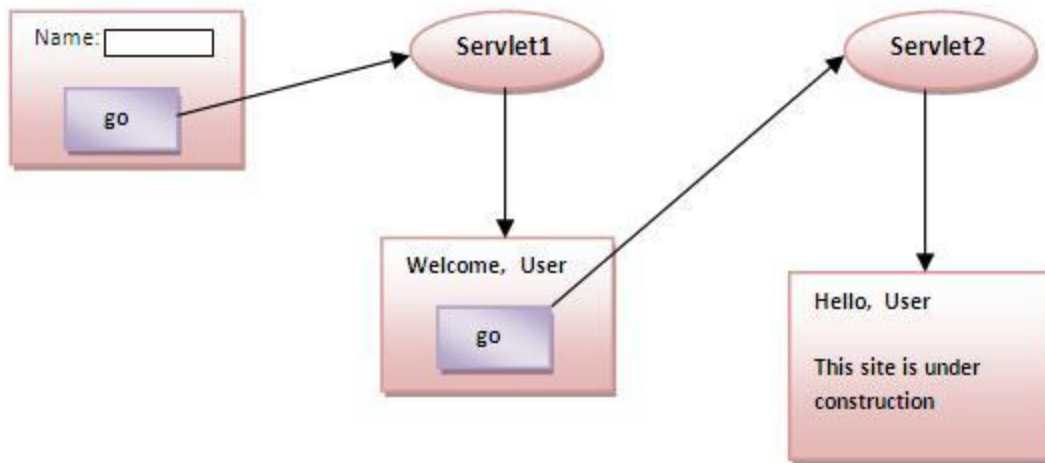
How to create Cookie?

Let's see the simple code to create cookie.

1. `Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object`
2. `response.addCookie(ck);//adding cookie in the response`

Simple example of Servlet Cookies

In this example, we are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



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);}
    }
}

```

```
}  
}
```

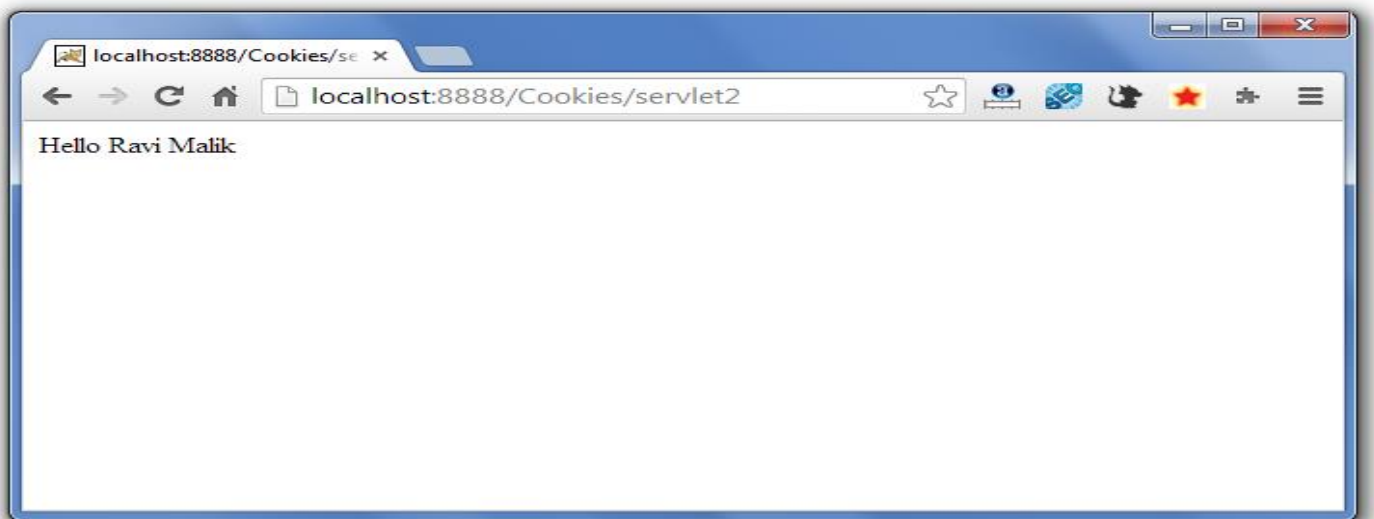
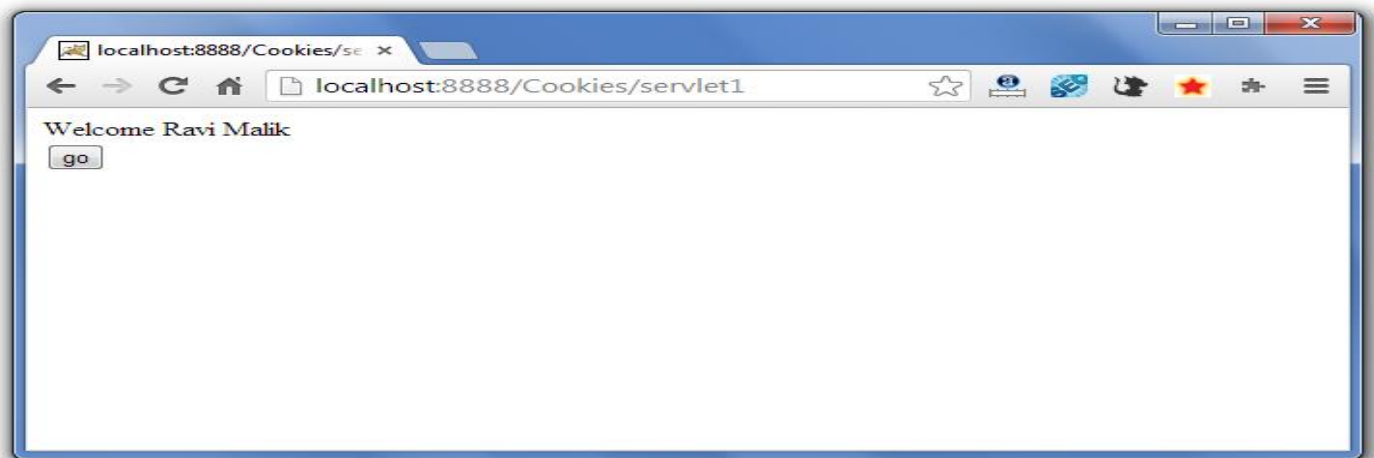
SecondServlet.java

```
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>
```

Output



2) Hidden Form Field

In case of Hidden Form Field a **hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

1. `<input type="hidden" name="uname" value="Vimal Jaiswal">`

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

Real application of hidden form field

It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Advantage of Hidden Form Field

1. It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

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

3)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&??`

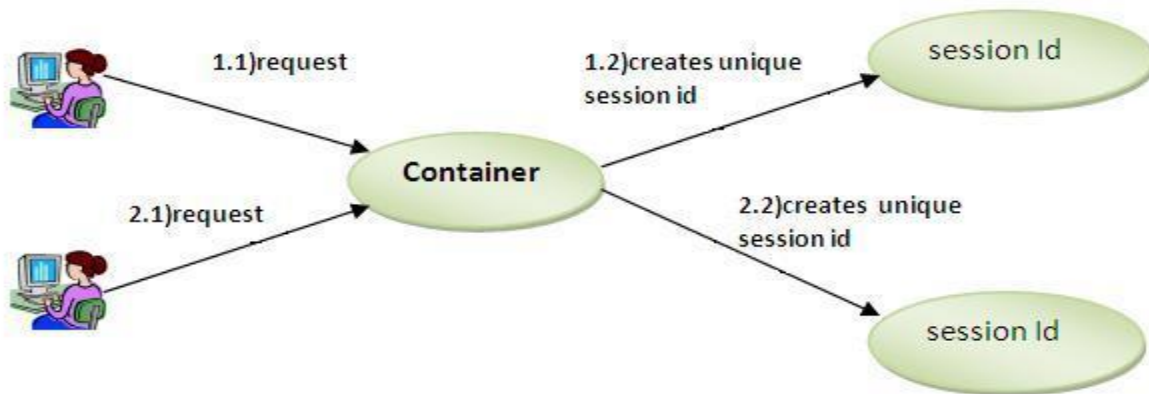
A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use `getParameter()` method to obtain a parameter value.

4) HttpSession interface

In such case, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

1. bind objects

2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):**Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.