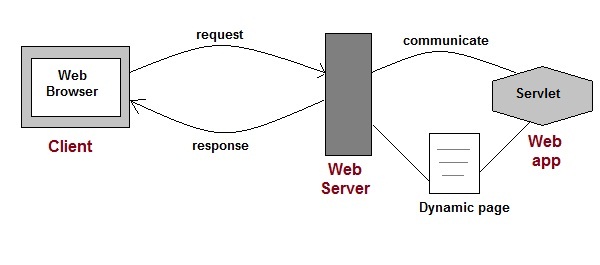
**Servlet**

**Servlet** Technology is used to create web applications. **Servlet** technology uses Java language to create web applications.

Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

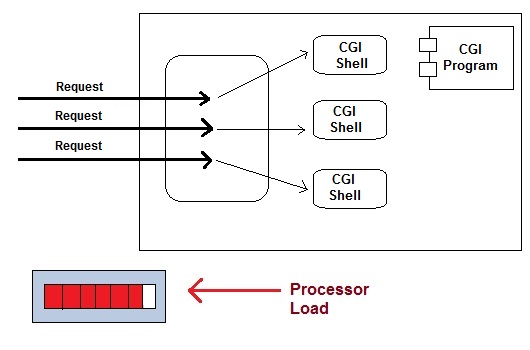


As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.

**CGI (Common Gateway Interface)**

Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works:

* User clicks a link that has URL to a dynamic page instead of a static page.
* The URL decides which CGI program to execute.
* Web Servers run the CGI program in seperate OS shell. The shell includes OS enviroment and the process to execute code of the CGI program.
* The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.



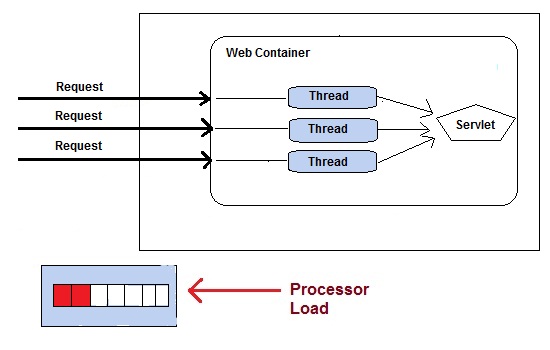
**Drawbacks of CGI programs**

* High resposne time because CGI programs execute in their own OS shell.
* CGI is not scalable.
* CGI programs are not always secure or object-oriented.
* It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

**Advantages of using Servlets**

* Less response time because each request runs in a separate thread.
* Servlets are scalable.
* Servlets are robust and object oriented.
* Servlets are platform independent.



#### Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what the interfaces of javax.servlet package are.

### Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

### Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

### Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

### Classes in javax.servlet.http package

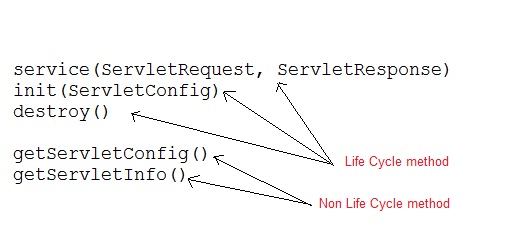
There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

#### Servlet Interface

**Servlet interface** provides common behavior to all the servlets.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.



### Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **public void init(ServletConfig config)** | initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once. |
| **public void service(ServletRequestrequest,ServletResponse response)** | provides response for the incoming request. It is invoked at each request by the web container. |
| **public void destroy()** | is invoked only once and indicates that servlet is being destroyed. |
| **public ServletConfiggetServletConfig()** | returns the object of ServletConfig. |
| **public String getServletInfo()** | returns information about servlet such as writer, copyright, version etc. |

### Servlet Example by implementing Servlet interface

File: First.java

1. import java.io.\*;
2. import javax.servlet.\*;
4. public class First implements Servlet{
5. ServletConfig config=null;
7. public void init(ServletConfig config){
8. this.config=config;
9. System.out.println("servlet is initialized");
10. }
12. public void service(ServletRequest req,ServletResponse res)
13. throws IOException,ServletException{
15. res.setContentType("text/html");
17. PrintWriter out=res.getWriter();
18. out.print("<html><body>");
19. out.print("<b>hello simple servlet</b>");
20. out.print("</body></html>");
22. }
23. public void destroy(){System.out.println("servlet is destroyed");}
24. public ServletConfig getServletConfig(){return config;}
25. public String getServletInfo(){return "copyright 2007-1010";}
27. }

#### GenericServlet Class

GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

**Methods of GenericServlet class**

* public void init(ServletConfig)
* public abstract void service(ServletRequestrequest,ServletResposne response)
* public void destroy()
* public ServletConfiggetServletConfig()
* public String getServletInfo()
* public ServletContextgetServletContext()
* public String getInitParameter(String name)
* public Enumeration getInitParameterNames()
* public String getServletName()
* public void log(String msg)
* public void log(String msg, Throwable t)

#### HttpServlet class

HttpServlet is also an abstract class. This class gives implementation of various service() methods of **Servlet** interface.

To create a servlet, we should create a class that extends **HttpServlet** abstract class. The Servlet class that we will create, must not override service() method. Our servlet class will override only the doGet() and/or doPost() methods.

The service() method of **HttpServlet** class listens to the Http methods (GET, POST etc) from request stream and invokes doGet() or doPost() methods based on Http Method type.

**How a Servlet Application works**

**Web container** is responsible for managing execution of servlets and JSP pages for Java EE application. When a request comes in for a servlet, the server hands the request to the Web Container. **Web Container** is responsible for instantiating the servlet or creating a new thread to handle the request. Its the job of Web Container to get the request and response to the servlet. The container creates multiple threads to process multiple requests to a single servlet.

**Servlets don't have a main() method**. Web Container manages the life cycle of a Servlet instance.

|  |
| --- |
| 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(ServletRequestreq,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequestreq, 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(HttpServletRequestreq, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequestreq, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequestreq, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequestreq, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequestreq, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequestreq, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequestreq, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequestreq)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

#### Get vs. Post

There are many differences between the Get and Post request. Let's see these differences:

|  |  |
| --- | --- |
| **GET** | **POST** |
| 1) In case of Get request, only **limited amount of data** can be sent because data is sent in header. | In case of post request, **large amount of data** can be sent because data is sent in body. |
| 2) Get request is **not secured** because data is exposed in URL bar. | Post request is **secured** because data is not exposed in URL bar. |
| 3) Get request **can be bookmarked.** | Post request **cannot be bookmarked.** |
| 4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered | Post request is **non-idempotent.** |
| 5) Get request is **more efficient** and used more than Post. | Post request is **less efficient** and used less than get. |

## GET and POST

Two common methods for the request-response between a server and client are:

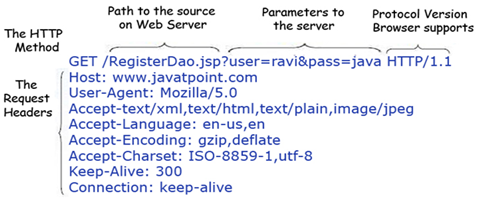
* **GET**- It requests the data from a specified resource
* **POST**- It submits the processed data to a specified resource

## Anatomy of Get Request

The query string (name/value pairs) is sent inside the URL of a GET request:

1. GET /RegisterDao.jsp?name1=value1&name2=value2

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what information is sent to the server.



Some other features of GET requests are:

* It remains in the browser history
* It can be bookmarked
* It can be cached
* It have length restrictions
* It should never be used when dealing with sensitive data
* It should only be used for retrieving the data

## Anatomy of Post Request

The query string (name/value pairs) is sent in HTTP message body for a POST request:

1. POST/RegisterDao.jsp HTTP/1.1
2. Host: www. javatpoint.com
3. name1=value1&name2=value2

As we know, in case of post request original data is sent in message body. Let's see how information is passed to the server in case of post request.

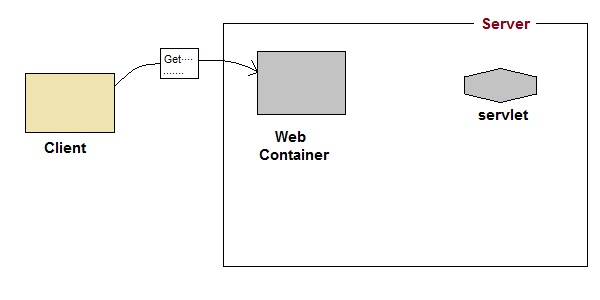


Some other features of POST requests are:

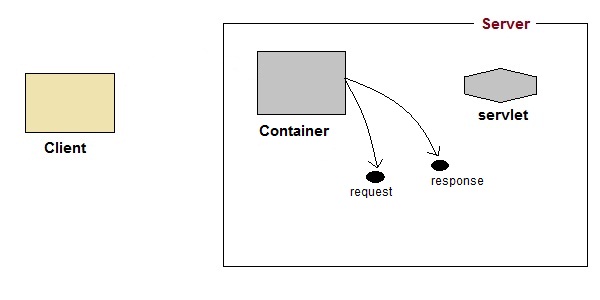
* This requests cannot be bookmarked
* This requests have no restrictions on length of data
* This requests are never cached
* This requests do not remains in the browser history

**How a Servlet works**

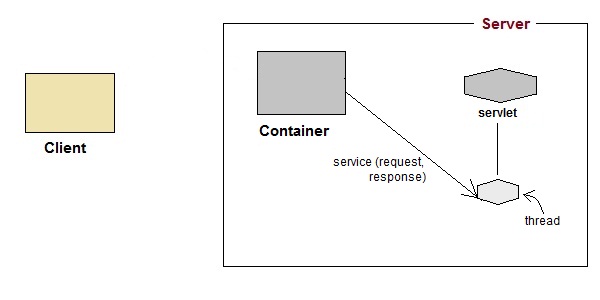
1. User sends request for a servlet by clicking a link that has URL to a servlet.



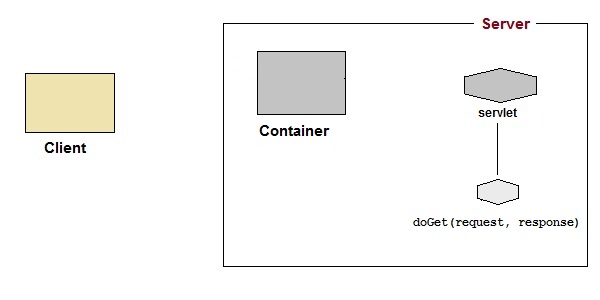
1. The container finds the servlet using **deployment descriptor** and creates two objects :
   1. **HttpServletRequest**
   2. **HttpServletResponse**



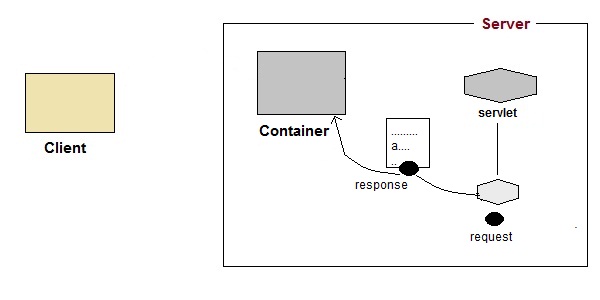
1. Then the container creates or allocates a thread for that request and calls the Servlet's service()method and passes the **request, response** objects as arguments.



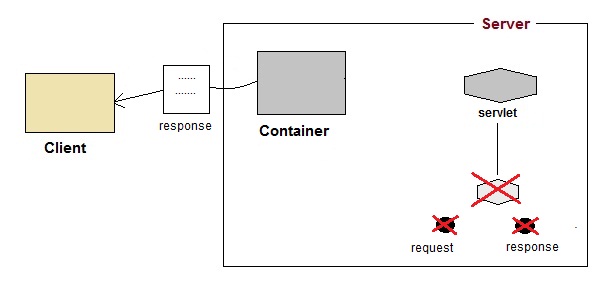
1. The service() method, then decides which servlet method, doGet() or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.



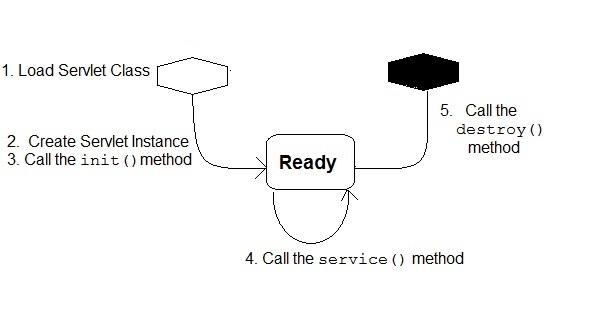
1. Then the Servlet uses response object to write the response back to the client.



1. After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.



#### Servlet Life Cycle



1. **Loading Servlet Class:** A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation:**After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. **Call to the init() method :** init() method is called by the Web Container on servlet instance to initialize the servlet.

**Signature of init() method :**

public void init(ServletConfigconfig) throws ServletException

1. **Call to the service() method :** The containers call the service() method each time the request for servlet is received. The service() method will then call the doGet() or doPost() methos based ont eh type of the HTTP request, as explained in previous lessons.

**Signature of service() method :**

public void service(ServletRequest request, ServletResponse response) throws

ServletException, IOException

1. **Call to destroy() method:** The Web Container call the destroy() method before removing servlet instance, giving it a chance for cleanup activity.

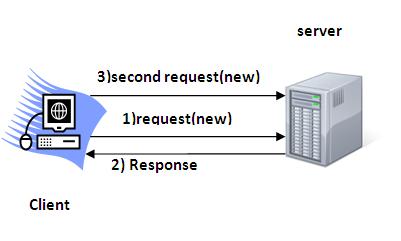
Session Tracking in Servlets

**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**
5. 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.



### Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

### Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

### Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

### 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.

### Cookie class

**javax.servlet.http.Cookie** class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

### Constructor of Cookie class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| Cookie() | constructs a cookie. |
| Cookie(String name, String value) | constructs a cookie with a specified name and value. |

### Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void setMaxAge(int expiry) | Sets the maximum age of the cookie in seconds. |
| public String getName() | Returns the name of the cookie. The name cannot be changed after creation. |
| public String getValue() | Returns the value of the cookie. |
| public void setName(String name) | changes the name of the cookie. |
| public void setValue(String value) | changes the value of the cookie. |

### Other methods required for using Cookies

|  |
| --- |
| For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:   1. **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object. 2. **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser. |

### 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

### How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

1. Cookie ck=new Cookie("user","");//deleting value of cookie
2. ck.setMaxAge(0);//changing the maximum age to 0 seconds
3. response.addCookie(ck);//adding cookie in the response

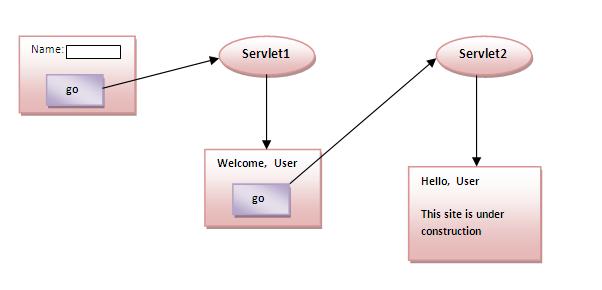
### How to get Cookies?

Let's see the simple code to get all the cookies.

1. Cookie ck[]=request.getCookies();
2. for(int i=0;i<ck.length;i++){
3. out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
4. }

### 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

1. <form action="servlet1" method="post">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>

### FirstServlet.java

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;

6. public class FirstServlet extends HttpServlet {
8. public void doPost(HttpServletRequest request, HttpServletResponse response){
9. try{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. Cookie ck=new Cookie("uname",n);//creating cookie object
18. response.addCookie(ck);//adding cookie in the response
20. //creating submit button
21. out.print("<form action='servlet2'>");
22. out.print("<input type='submit' value='go'>");
23. out.print("</form>");
25. out.close();
27. }catch(Exception e){System.out.println(e);}
28. }
29. }

### SecondServlet.java

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;
5. public class SecondServlet extends HttpServlet {
7. public void doPost(HttpServletRequest request, HttpServletResponse response){
8. try{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. Cookie ck[]=request.getCookies();
14. out.print("Hello "+ck[0].getValue());
16. out.close();
18. }catch(Exception e){System.out.println(e);}
19. }

22. }

### web.xml

1. <web-app>
3. <servlet>
4. <servlet-name>s1</servlet-name>
5. <servlet-class>FirstServlet</servlet-class>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>s1</servlet-name>
10. <url-pattern>/servlet1</url-pattern>
11. </servlet-mapping>
13. <servlet>
14. <servlet-name>s2</servlet-name>
15. <servlet-class>SecondServlet</servlet-class>
16. </servlet>
18. <servlet-mapping>
19. <servlet-name>s2</servlet-name>
20. <url-pattern>/servlet2</url-pattern>
21. </servlet-mapping>
23. </web-app>

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 VimalJaiswal 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

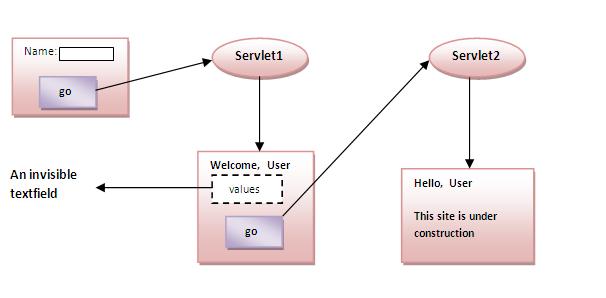
1. It will always work whether cookie is disabled or not.

### 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.

### Example of using Hidden Form Field

In this example, we are storing the name of the user in a hidden text field and getting that value from another servlet.



### index.html

1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>

### FirstServlet.java

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;
5. public class FirstServlet extends HttpServlet {
6. public void doGet(HttpServletRequest request, HttpServletResponse response){
7. try{
9. response.setContentType("text/html");
10. PrintWriter out = response.getWriter();
12. String n=request.getParameter("userName");
13. out.print("Welcome "+n);
15. //creating form that have invisible textfield
16. out.print("<form action='servlet2'>");
17. out.print("<input type='hidden' name='uname' value='"+n+"'>");
18. out.print("<input type='submit' value='go'>");
19. out.print("</form>");
20. out.close();
22. }catch(Exception e){System.out.println(e);}
23. }
25. }

### SecondServlet.java

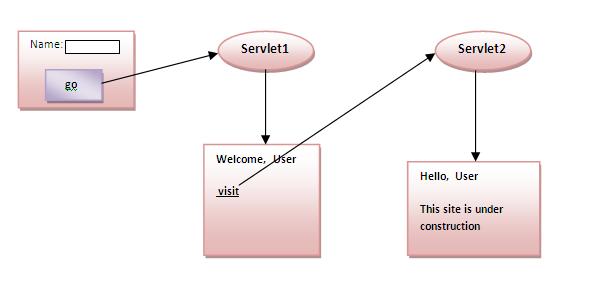
1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;
4. public class SecondServlet extends HttpServlet {
5. public void doGet(HttpServletRequest request, HttpServletResponse response)
6. try{
7. response.setContentType("text/html");
8. PrintWriter out = response.getWriter();
10. //Getting the value from the hidden field
11. String n=request.getParameter("uname");
12. out.print("Hello "+n);
14. out.close();
15. }catch(Exception e){System.out.println(e);}
16. }
17. }

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.



### Advantage of URL Rewriting

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

### Disadvantage of URL Rewriting

1. It will work only with links.
2. 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.

### index.html

1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>

### FirstServlet.java

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;

6. public class FirstServlet extends HttpServlet {
8. public void doGet(HttpServletRequest request, HttpServletResponse response){
9. try{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. //appending the username in the query string
18. out.print("<a href='servlet2?uname="+n+"'>visit</a>");
20. out.close();
22. }catch(Exception e){System.out.println(e);}
23. }
25. }

### SecondServlet.java

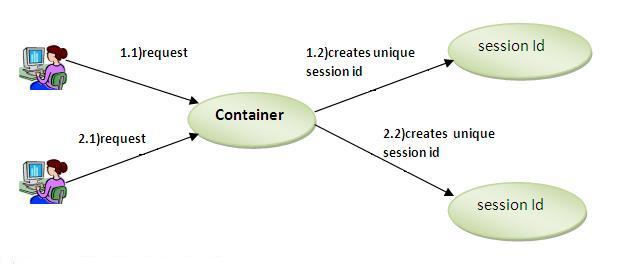
1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;
5. public class SecondServlet extends HttpServlet {
7. public void doGet(HttpServletRequest request, HttpServletResponse response)
8. try{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. //getting value from the query string
14. String n=request.getParameter("uname");
15. out.print("Hello "+n);
17. out.close();
19. }catch(Exception e){System.out.println(e);}
20. }

23. }

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. **publicHttpSessiongetSession():**Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **publicHttpSessiongetSession(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.

### Commonly used methods of HttpSession interface

1. **public String getId():**Returns a string containing the unique identifier value.
2. **public long getCreationTime():**Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **public long getLastAccessedTime():**Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

### Example of using HttpSession

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the setAttribute() method of HttpSession interface and to get the attribute, we have used the getAttribute method.

### index.html

1. <form action="servlet1">
2. Name:<input type="text" name="userName"/><br/>
3. <input type="submit" value="go"/>
4. </form>

### FirstServlet.java

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;

6. public class FirstServlet extends HttpServlet {
8. public void doGet(HttpServletRequest request, HttpServletResponse response){
9. try{
11. response.setContentType("text/html");
12. PrintWriter out = response.getWriter();
14. String n=request.getParameter("userName");
15. out.print("Welcome "+n);
17. HttpSession session=request.getSession();
18. session.setAttribute("uname",n);
20. out.print("<a href='servlet2'>visit</a>");
22. out.close();
24. }catch(Exception e){System.out.println(e);}
25. }
27. }

### SecondServlet.java

1. import java.io.\*;
2. import javax.servlet.\*;
3. import javax.servlet.http.\*;
5. public class SecondServlet extends HttpServlet {
7. public void doGet(HttpServletRequest request, HttpServletResponse response)
8. try{
10. response.setContentType("text/html");
11. PrintWriter out = response.getWriter();
13. HttpSession session=request.getSession(false);
14. String n=(String)session.getAttribute("uname");
15. out.print("Hello "+n);
17. out.close();
19. }catch(Exception e){System.out.println(e);}
20. }
21. }