Session tracking

- HTTP is a stateless protocol that provides no way for a server to recognize that a sequence of requests comm. From the same client.
- But many web applications are not stateless.
- A server can not identify the client by the ip address,

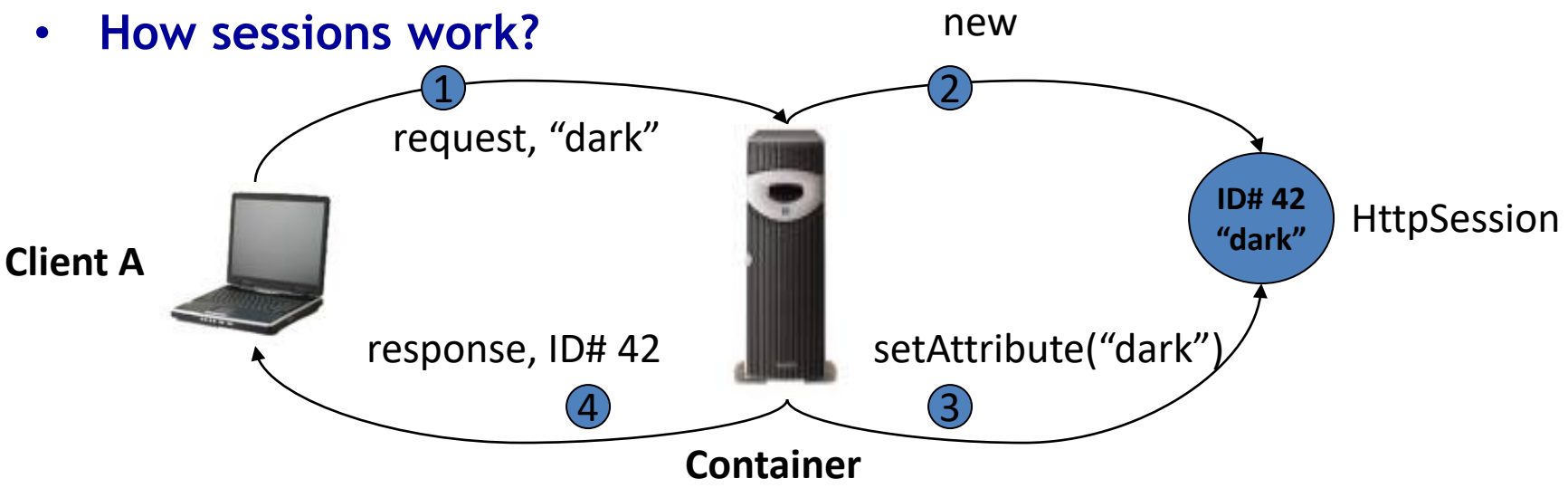
  because the reported ip address may be the address of a proxy server or the address of a server machine that hosts several clients.
- Session tracking gives servlets and other server-side applications the ability to keep track of the user as the user moves through the site.
- Web server maintains user state by creating a session object for each user with a unique session id.
- The session object is passed as part of the request.
- Servlets can add information to session object and read information from them.

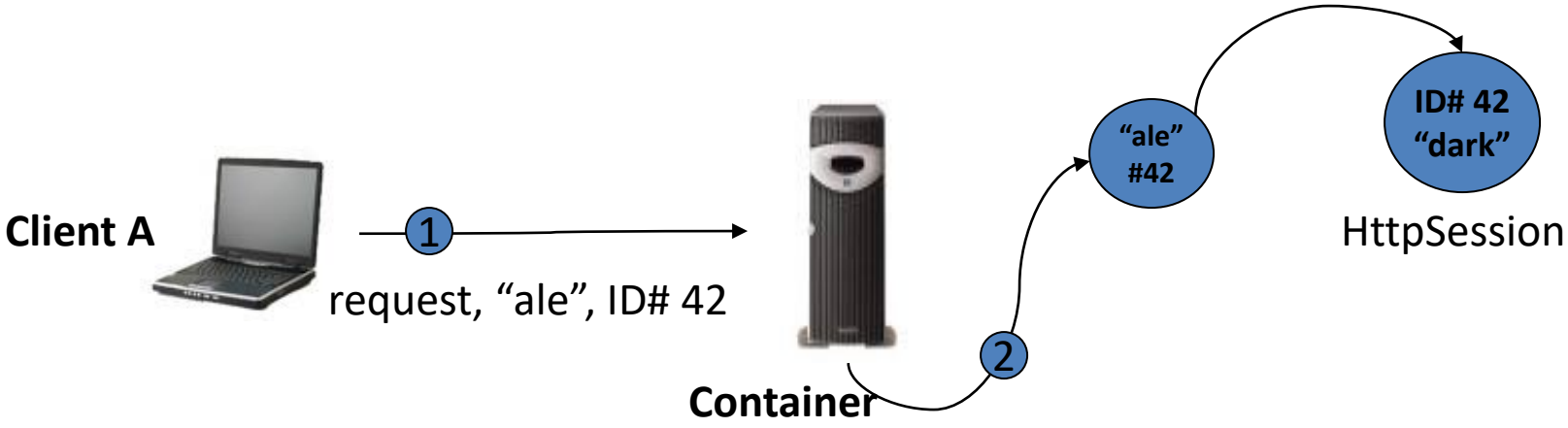
What is Session Tracking?

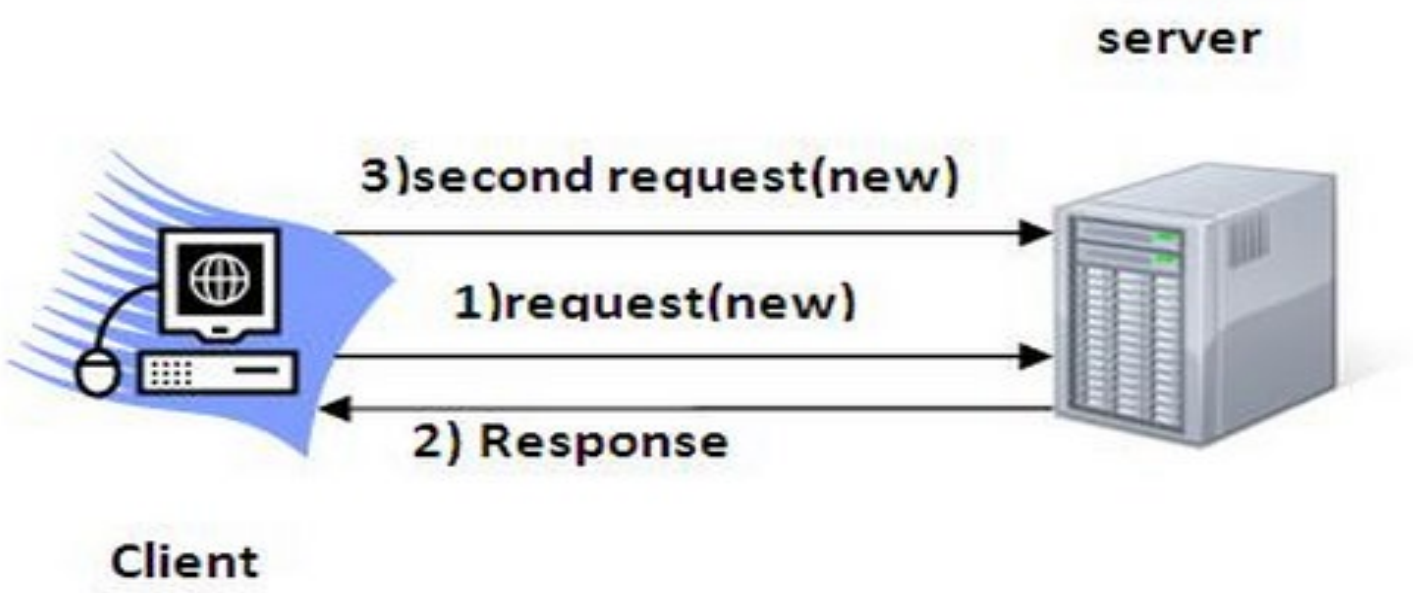
- HTTP is a "stateless" protocol.
- when you are doing on-line shopping,
- the Web server can't easily remember previous transactions.
- This makes applications like shopping carts very problematic:
- when you add an entry to your cart,
- how does the server know what's already in your cart?
- When you move from the page where you specify what you want to buy
- (hosted on the regular Web server)
- to the page that takes your credit card number and
- shipping address (hosted on the secure server that uses SSL), how does the server remember what you were buying?

Session Management - Session Tracking

- How sessions work?







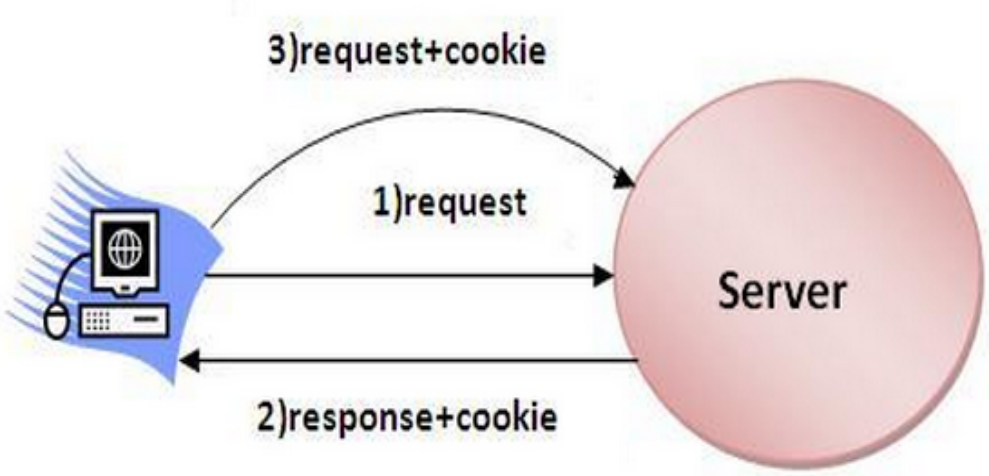
Why use Session Tracking?

To recognize the user.

There are four typical solutions to this problem.

- 1. **Cookies.**
- 2. **URL Rewriting.**
- 3. **Hidden form fields.**
- 4. HTTP Session

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.



javax.servlet.http.Cookie class

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

Constructor of Cookie class

Cookie(String name, String value): Constructs a cookie with a specified name and value.

Commonly used methods of Cookie class

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

1. **public void setMaxAge(int expiry):**Sets the maximum age of the cookie in seconds.
2. **public String getName():**Returns the name of the cookie. The name cannot be changed after creation.
3. **public String getValue():**Returns the value of the cookie.

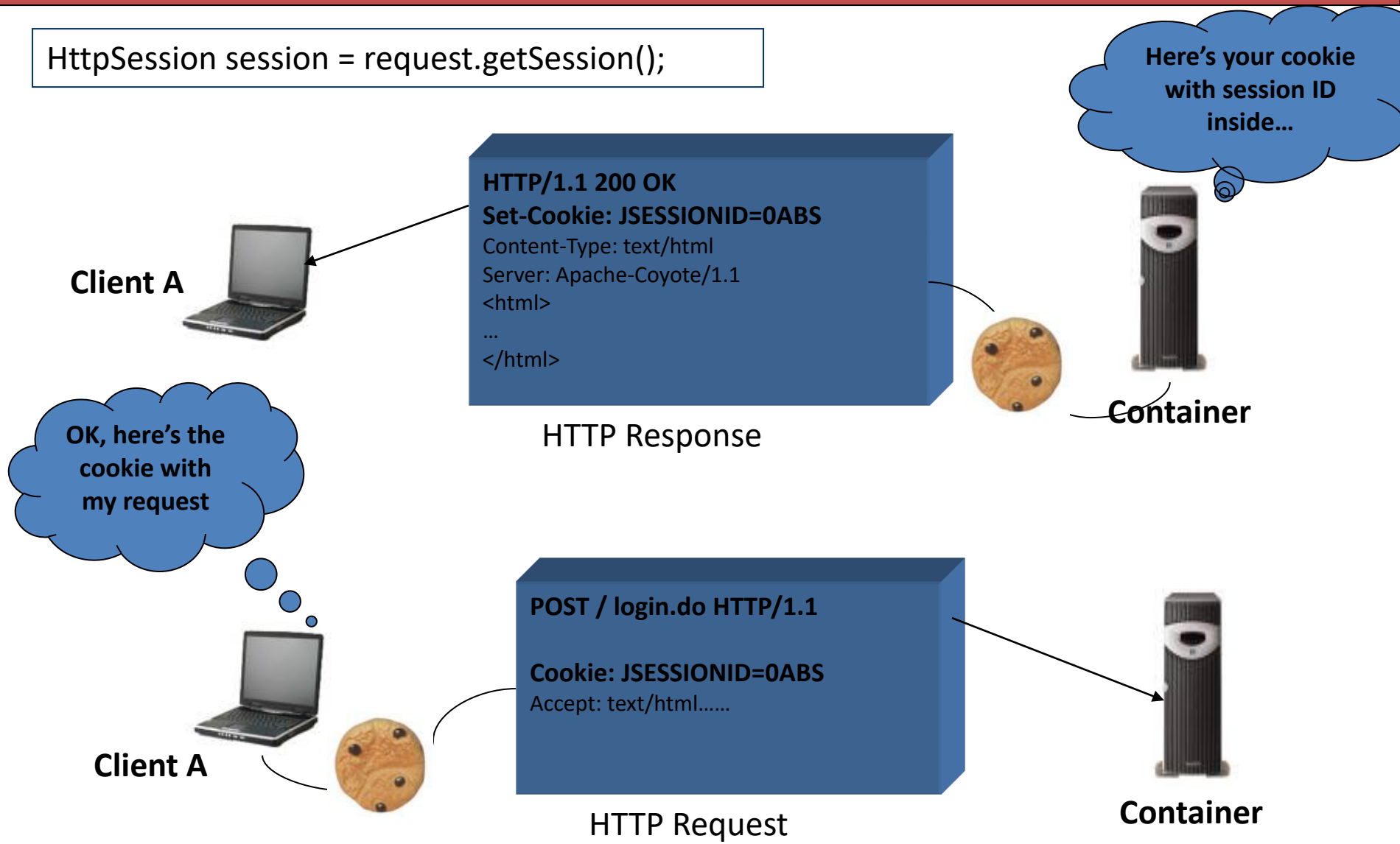
Other methods required for using Cookies

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.

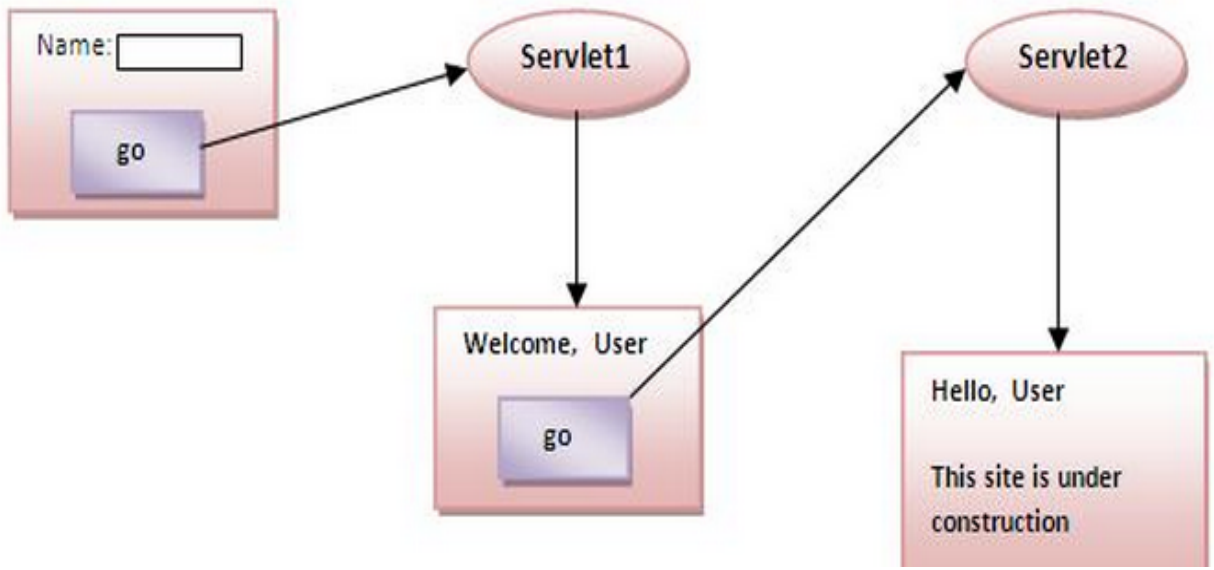
Session Tracking - Cookies

```
HttpSession session = request.getSession();
```



✓*Example of using 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>
```

First servlet

```
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);}
}
```

Second servlet

```
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].getName());

        out.close();

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

Web.xml

```
<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>
```

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.

URL Rewriting.

1. You can append some extra data on the end of each URL that identifies the session,
2. and the server can associate that session identifier with data it has stored about that session.
3. This is also an excellent solution, and
4. even has the advantage that it works with browsers that don't support cookies or where the user has disabled cookies.
5. However, it has most of the same problems as cookies,
6. namely that the server-side program has a lot of straightforward but tedious processing to do.
7. In addition, you have to be very careful that every URL returned to the user
8. (even via indirect means like Location fields in server redirects) has the extra information appended.
9. And, if the user leaves the session and comes back via a bookmark or link, the session information can be lost.

Session tracking-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.

Session Tracking - URL Rewriting

URL + ;jsessionid=1234567

Client A

HTTP/1.1 200 OK
Content-Type: text/html
Server: Apache-Coyote/1.1
<html>
<body>
< a href =" http:// www.sharmanj.com/Metavante;jsessionid=0AAB">
**click me **
</html>

HTTP Response

Container

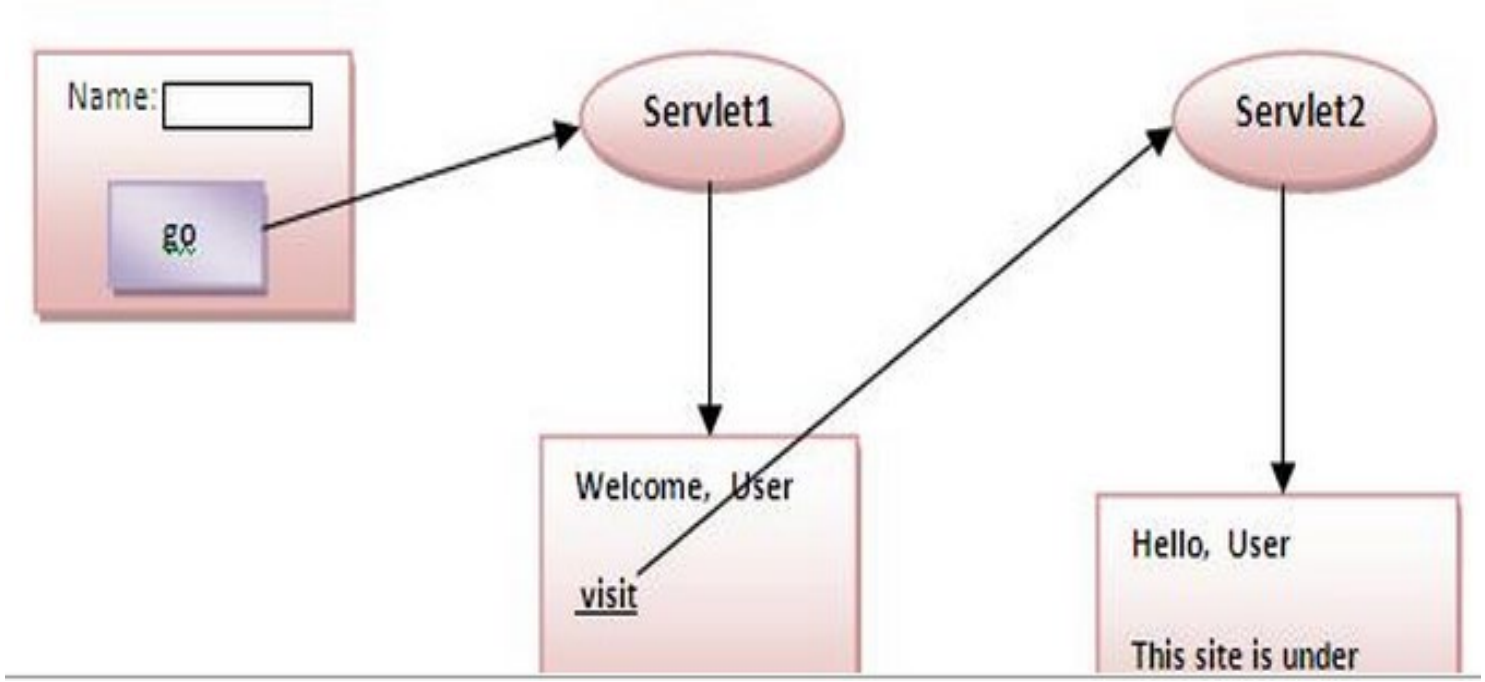
GET /Metavante;jsessionid=0AAB

HTTP / 1.1
Host: www.sharmanj.com
Accept: text/html

HTTP Request

Client A

Container


Name:
go
Servlet1
Servlet2
Welcome, User
visit
Hello, User
This site is under

✓*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

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

First servlet

```
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);}
        }
```

second servlet

```
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-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>
```

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.

Hidden form fields.

HTML forms have an entry that looks like the following:

<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">.

when the form is submitted,

the specified name and value are included in the GET or POST data.

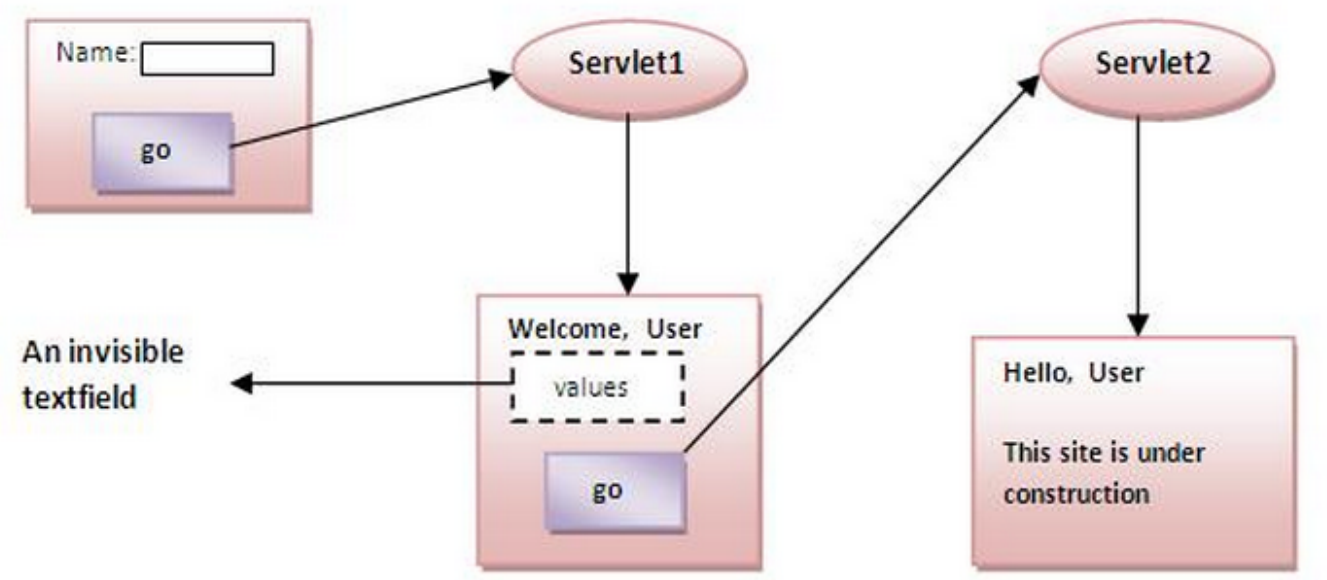
This can be used to store information about the session. Disadvantage

that it only works if every page is dynamically generated,

since the whole point is that each session has a unique identifier.

3.Hidden form field

In case of Hidden Form Field an invisible textfield is used for maintaing 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.



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 textfield and getting that value from another servlet.

index.html

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

```
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='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);}
        }
```

```
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

```
<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>
```

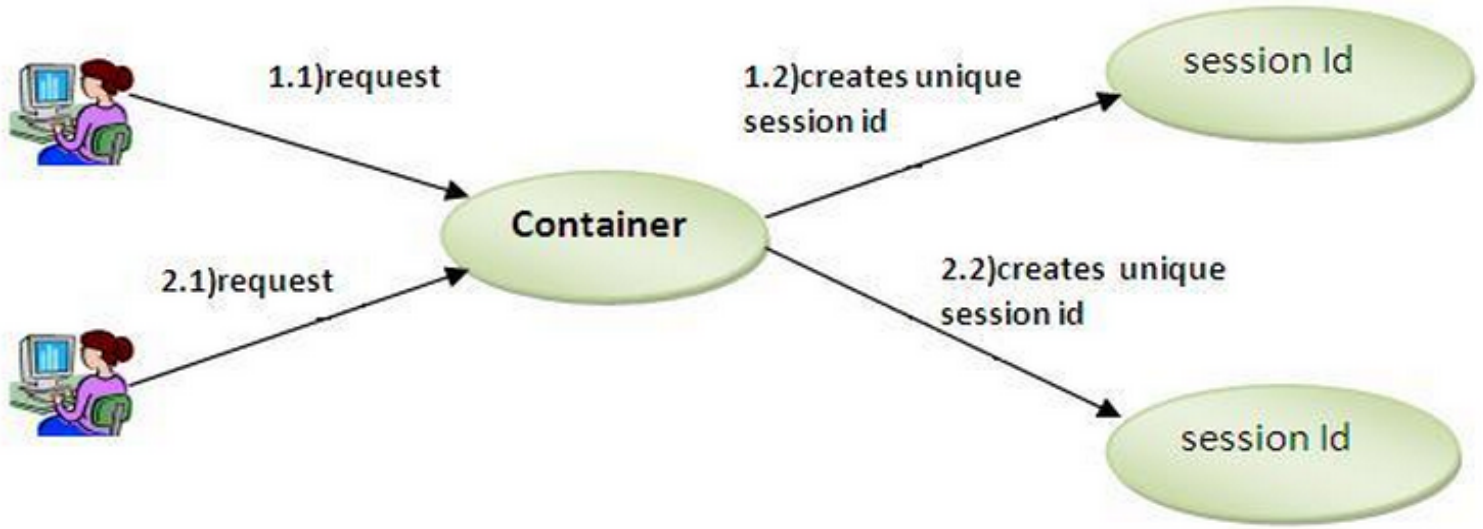
HttpSession API.

- This is a high-level interface built on top of cookies or URL-rewriting.
- In fact, on many servers, they use cookies if the browser supports them,
- but automatically revert to URL-rewriting when cookies are unsupported or explicitly disabled. But the servlet author doesn't need to bother with many of the details,
- doesn't have to explicitly manipulate cookies or information appended to the URL, and
- is automatically given a convenient place to store data that is associated with each session.

4.HttpSession

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.

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

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

First servlet

```
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);

                HttpSession session=request.getSession();
                session.setAttribute("uname",n);

                out.print("<a href='servlet2'>visit</a>");

                out.close();

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

}
```

Second servlet

```
public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
                try{

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

                HttpSession session=request.getSession(false);
                String n=(String)session.getAttribute("uname");
                out.print("Hello "+n);

                out.close();

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


}
```

```
<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>
```