Web Component Development Using Java

Session: 3

Session Tracking

Objectives

- Describe the stateless nature of HTTP protocol
- Explain the need of tracking client identity and state
- Explain the URL rewriting method for session tracking
- Explain how to use hidden form fields
- Explain the use of Cookie class and its methods
- Explain how to store and retrieve information in a session
- Describe the use of HTTP session interface and its methods
- Explain how to invalidate a session

Introduction 1-3

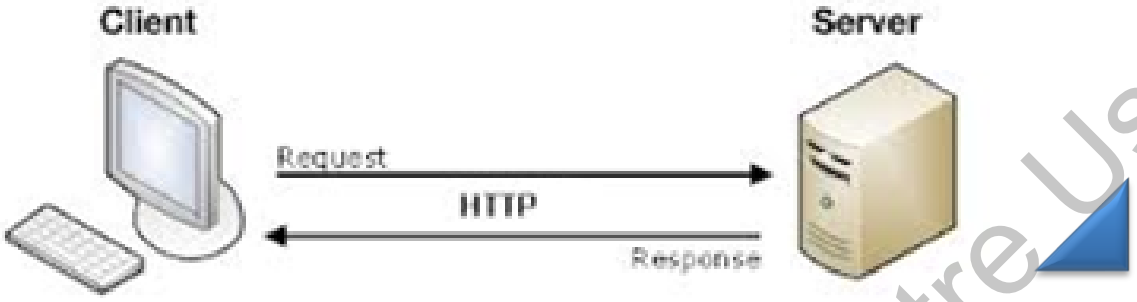
- **Protocol**
  - A set of rules, which governs the syntax, semantics, and synchronization of communication in computers.

- A protocol is said to be stateless when:
  - The configuration setting, transaction, and information are not tracked by a protocol.
  - The connections last for only one transaction.

Example: HTTP protocol.

Introduction 2-3

❖ Why HTTP is referred to as a Stateless Protocol?



HTTP client opens a connection and sends a request message to an HTTP server for a resource

The server returns a response message with the requested resource

Once the resource is delivered, the server closes the connection without storing any information

No connection information is stored, and hence **HTTP is referred to as a stateless protocol**.

Introduction 3-3

❖ **Advantages and Disadvantages of a Stateless Protocol**

Advantages	Disadvantages
• Hosts do not need to retain information about users between requests. • Simplifies the server design.	• Need to include more information in each request which would be interpreted by the server each time. • No acknowledgement of received information.

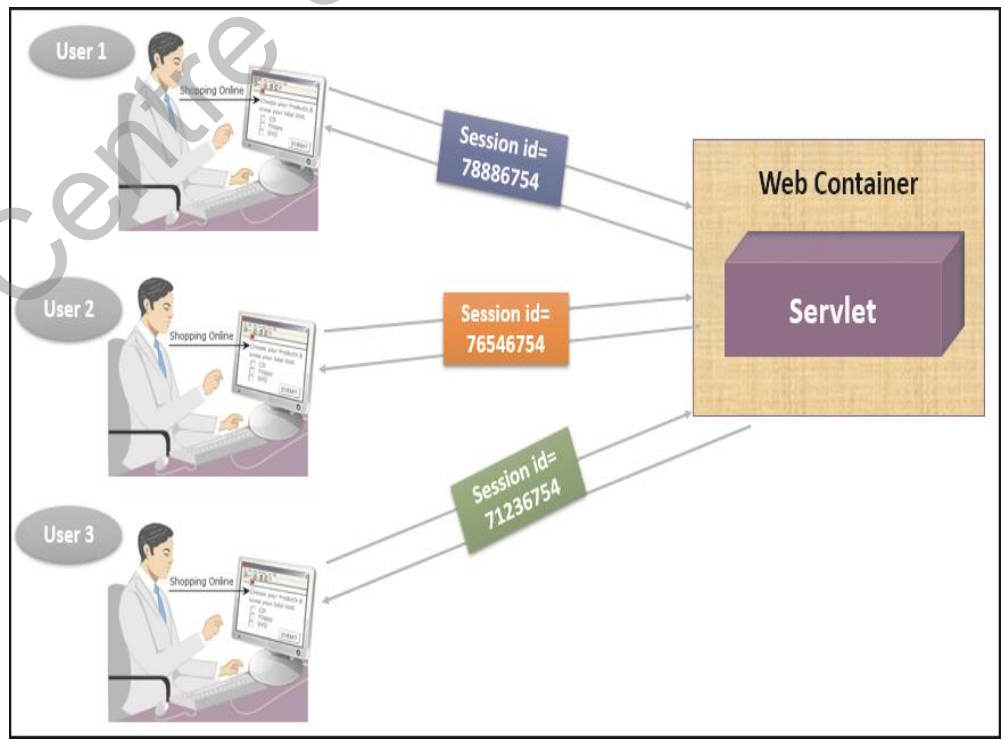
Session Tracking

Problem:

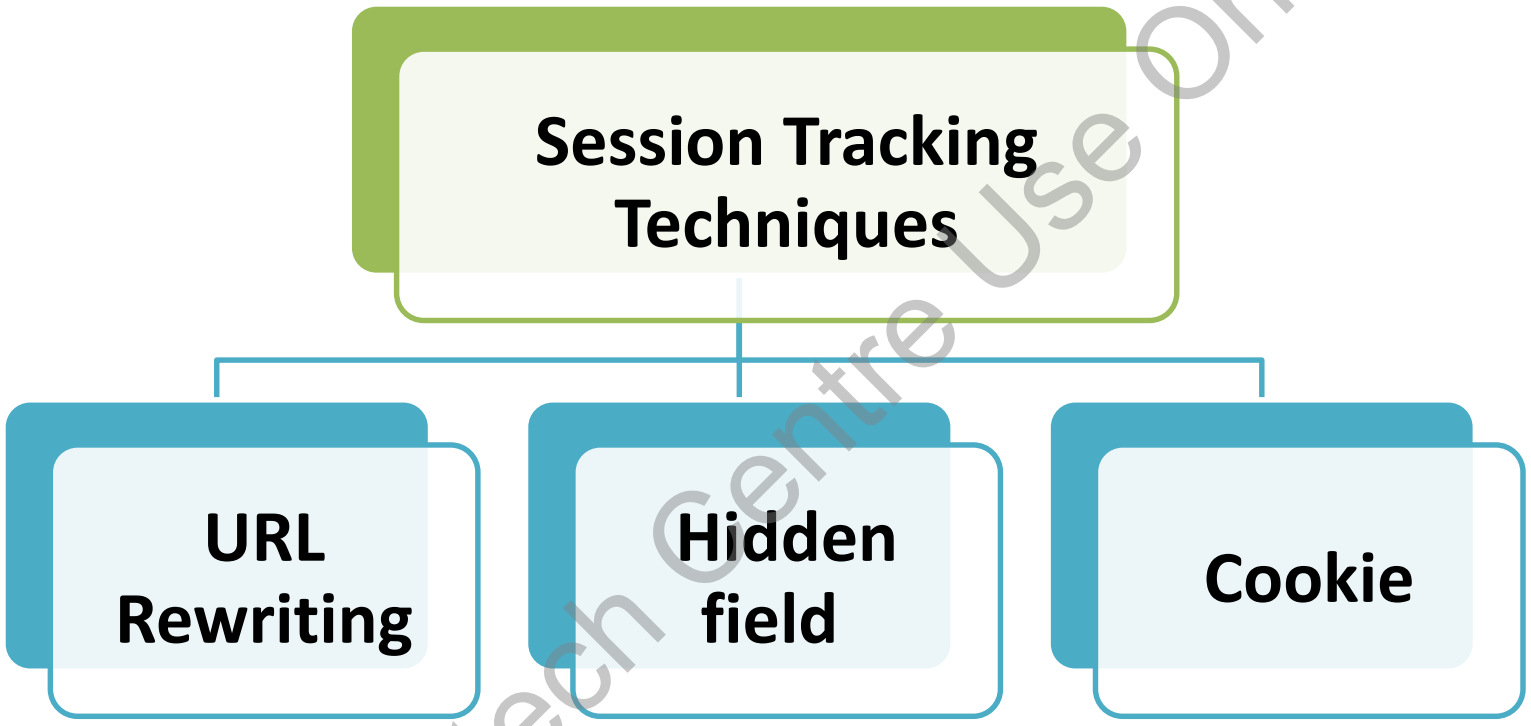
When a customer is doing online shopping, he/she may select items from various pages and put it in the shopping cart. When the customer clicks a new page, the information about the previously selected items is lost due to HTTP being a stateless protocol.

Solution: Session Tracking

- It allows the Web application to maintain information with the server as long as the customer does not log out from the Website.
- It tracks the client identity and other state information required throughout the session.



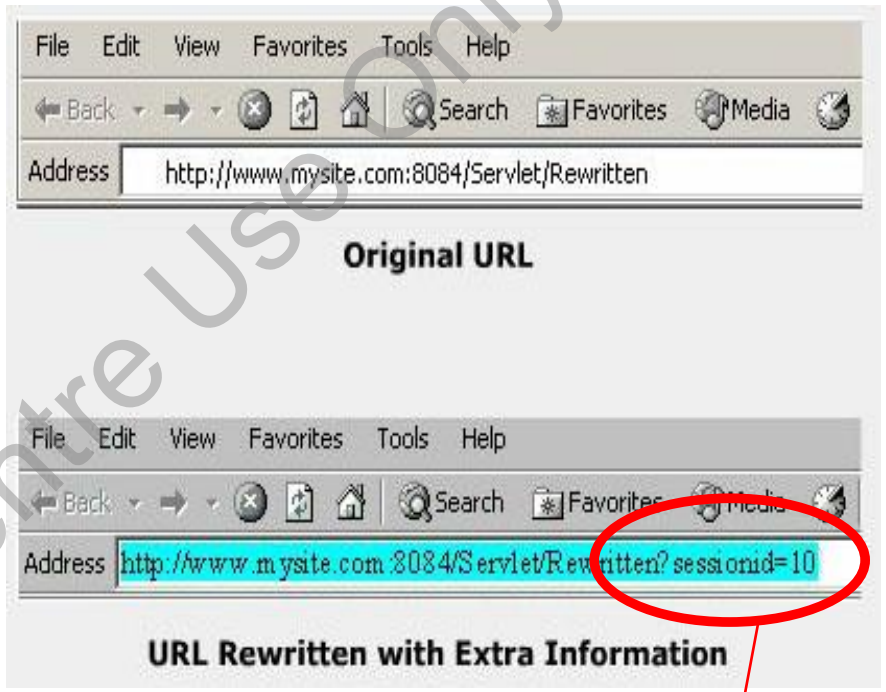
Session Tracking Techniques



Java Servlet API specification also provides a session tracking mechanism through `javax.servlet.http.HttpSession` object.

URL Rewriting

- Uniform Resource Locator (URL) is the address of a resource located on the Web.
- The URL rewriting technique:
  - Adds some extra data at the end of the URL to identify the session.
  - Extra information can be in the form of extra path information or added parameters.
  - When the user clicks a link, the data from the page is appended after the '?' in the URL.
  - Is the lowest priority technique used for session management and is used as an alternative for cookies.

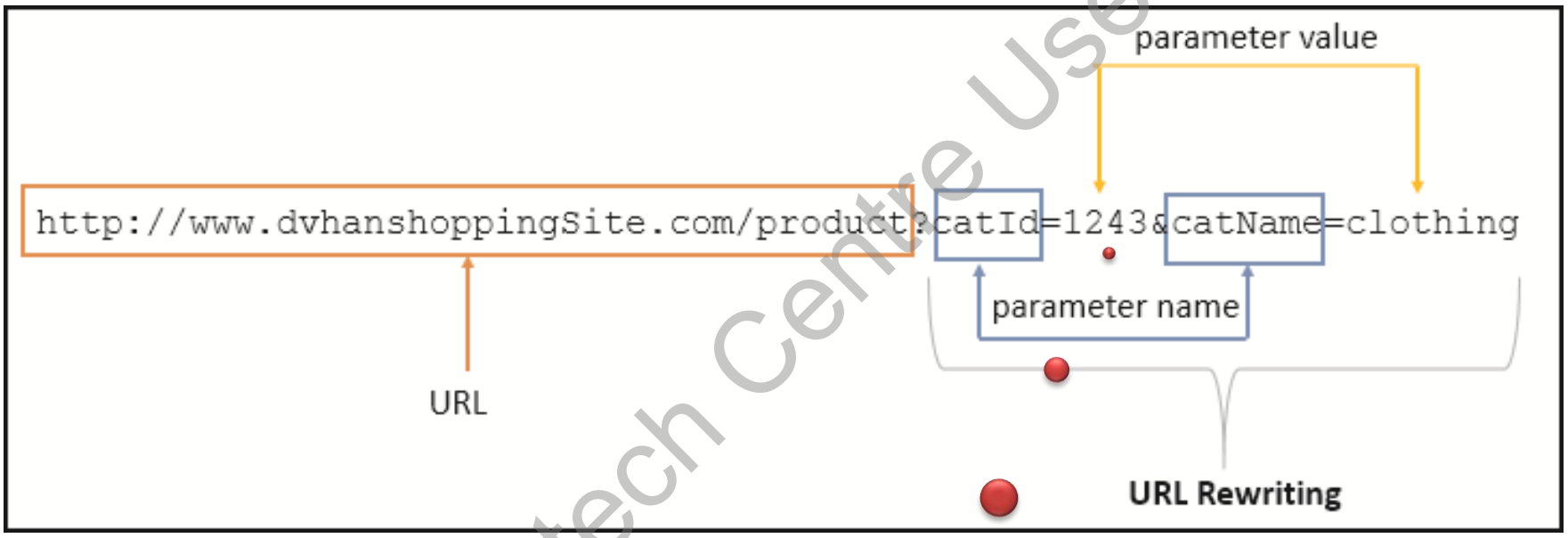


Original URL is appended with the parameter sessionid=10 at the end

This parameter is sent to the server as part of the client's request and helps the server to identify the client.

Information in URL 1-4

- Figure shows a parameter or token attached at the end of the query string sent in the request.



The token consist of name-value pair.

Information in URL 2-4

- The code snippet demonstrates the working of URL rewriting technique using two servlets.

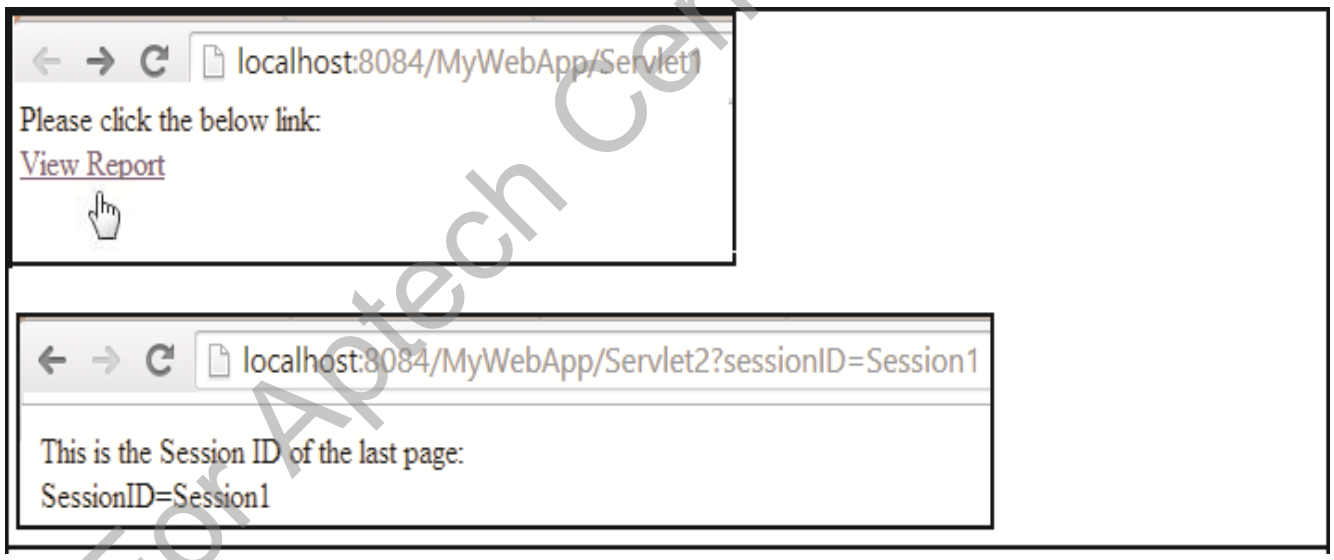
```
/* Servlet1.java */

. . .
public class Servlet1 extends HttpServlet {
   protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {

        response.setContentType("text/html;charset=UTF-8");

        String sessionID = "Session1";
         PrintWriter pw = response.getWriter();

                pw.println("<html>");
                pw.println("<head></head>");
                pw.println("<body>");
                pw.println("Please click the below link:<br>");

pw.println("<a href=/MyWebApp/Servlet2?sessionID="+sessionID+">
View Report</a><br>");

                pw.println("</body>");
                pw.println("</html>");

        }
}
```

Information in URL 3-4

```
/*Servlet2.java*/

. . .
public class Servlet2 extends HttpServlet {

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

        response.setContentType("text/html;charset=UTF-8");

        String sessionID = request.getParameter("sessionID");

                PrintWriter pw = response.getWriter();

                pw.println("<html>");
                pw.println("<head></head>");
                pw.println("<body>");
                pw.println("This is the Session ID of the last page:
<br>");
                pw.println("SessionID="+sessionID+"<br>");
                pw.println("</body>");
                pw.println("</html>");

    }
}
```

Information in URL 4-4

- In the code, **`Servlet1`** generates a hyperlink element which when clicked by the user will send the request to **`Servlet2`**.
- A name-value parameter **`sessionID=Session1`** is appended with the URL link specified with the `<a>` element.
- **`Servlet2`** accesses the parameter using `getParameter()` method and displays its value on the page.
- **Output:**



URL Rewriting - Advantages and Disadvantages

Advantages

- Can be appended when sending the data from the HTML form.
- Can be sent along with the dynamic generated content from a Servlet.
- Is a preferred way to maintain the session when the browser doesn't support cookies or user disables the support for cookies.

Disadvantages

- URL can only be send through hyperlinks on the Web page.
- Long URLs cannot store that much information because of the URLs length limitation.
- URLs containing data information is visible, so it is not safe to be shared with others.

Hidden Form Fields 1-3

- **Hidden fields**:
  - They are placed within an HTML form.
  - They are either a part of the static HTML form or dynamic form generated through Servlets.
  - They can be used to hold any kind of data.
  - They are not visible to the user and hence are not interpreted by the browser.
- **Advantages**:
  - User can pass much more information to the server.
  - Character encoding is not necessary.
- **Syntax**:

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

Hidden Form Fields 2-3

- The code snippet demonstrates the use of hidden field.

```
<!-- index.html -- >
. . .
<body>
  <form action = "MyServlet" method="POST">

   <input type="hidden" name="job" value="Developer"/>

    Name: <input type="text" name="firstname"/>

    <input type="submit" name="Submit"/>
  </form>

</body>
. . .
```

```
/* MyServlet.java */
. . .
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {

    response.setContentType("text/html;charset=UTF-8");

    PrintWriter out = response.getWriter();
```

Hidden Form Fields 3-3

```
        try {
            // Retrieves the name entered on the form
            String name = request.getParameter("firstname");
            // Retrieves the value from the hidden field
            String job = request.getParameter("job");
            out.println("<h3> Welcome: " + name + "<br>");
            out.println("\n Your Job: " + job + "</h3>");
            out.close();
        } finally {
            out.close();
        }
    }
. . .
```

- **Output:**

Hidden Form Fields - Advantages and Disadvantages

Advantages

- Supported in all browsers
- No special server requirements from clients
- Not visible directly to the user
- Works with or without cookies

Disadvantages

Works only when the page receives request through a submission of a form

Cookies

- **Cookie**:
  - Small piece of information sent by a server to the client Web browser.
  - Stored on client's machine and is read back by the server on receiving request for the same page.
  - Contains one or more name-value pairs which are exchanged in request and response headers.
  - Stored for a limited life span on client's machine.
  - Are automatically deleted after a specified time period is completed.
  - Value of the cookie can uniquely identify a client.

- HTTP request header contains:
  - Information such as method, URL path, and HTTP protocol version.

- HTTP response header contains:
  - Date, size, and type of the file that server is sending back to the client.

Cookie API 1-9

- Figure depicts the concept of cookie.

As the value of the cookie can uniquely identify a client, cookies are commonly used in session tracking.

- **Drawback:**
  - Most browsers allow the users to deactivate cookies.

Cookie API 2-9

- The code snippet demonstrates how to create and add cookie in the Servlet response.

```
//This snippet remember an added item by adding to a cookie

public void doGet (HttpServletRequest request,
HttpServletResponse response) throwsServletException,
IOException
{
  //If the user wants to add an item in a cookie

      if (values != null) {
          ItemId = values[0];

          Cookie getItem = new Cookie("Buy", ItemId);

           getItem.setComment("User has indicated a desire "
        + "to buy this book from the bookstore.");

          response.addCookie(getItem);
      }
}
```

Cookie API 3-9

- The Servlet API provides `javax.servlet.http.Cookie` class for creating and working with cookies.
- The `Cookie` class provides several methods which help in cookie management:
  - **`public void setMaxAge(int expiry)`**
    - This method sets the maximum age of the cookie in seconds.
    - If the value is positive, then the cookie will expire after that many seconds which is specified by the expiry.
    - For example, `demoCookie.setMaxAge(600);`
  - **`public int getMaxAge()`**
    - It returns the maximum age of the cookie.
    - It returns an integer which specifies the maximum age of the cookies in seconds.

Cookie API 4-9

- The code snippet demonstrates how to get the cookie age.

```
/* Prints the cookie age */
  PrintWriter out = response.getWriter();
  Cookie demoCookie = new Cookie("FavColor", "Blue");
  demoCookie.setMaxAge(600);
  int result = demoCookie.getMaxAge();
  out.println("Cookie Age: " +result);
. . .
```

- The code returns the maximum age of the cookie, which is specified in seconds.

- **`public void setValue(java.lang.String newValue)`**
  - This method assigns a new value to a cookie after the cookie is created.

Cookie API 5-9

- The code snippet demonstrates how to set the value of the cookie.

```
// Sets the value of the cookie
public void setCookieValue(String value)
{
   if (value == null || (value.equals("")))
   throw new IllegalArgumentException("Invalid
cookie value set in: " + getClass().getName());
    if (cookie != null)
       cookie.setValue(value);
}
```

- **public java.lang.String getValue()**
  - Returns a string containing the cookie's present value.

Cookie API 6-9

- **public java.lang.String getName()**
  - Returns the name of the cookie.
  - Once, the cookie has been created its name cannot be changed.
  - The code snippet demonstrates how to retrieve the name and value of the cookie.

```
// Retrieves the name of the cookie and its value
for (int i = 0; i<cookies.length; i++) {
   String name = cookies[i].getName();
   String value = cookies[i].getValue();
   out.println("name = " + name +"; value = " +
value);
```

- **public void setPath(String uri)**
  - Sets the path for the cookie.
  - Is available to all the pages specified in the directory and its subdirectories.

Cookie API 7-9

- The code snippet demonstrates how to set the path for the cookie.

```
// Sets the path for the cookie

   cookie = new Cookie("sessionId", "erased");
   cookie.setPath("/servlet/SessionCookie");
   resp.setHeader("Set-Cookie", cookie.toString());
```

- **public java.lang.String getPath()**
  - Returns the path on the server to which the client returns the cookie.
  - Returns a string specifying a path that contains a servlet name.
  - Is available to all sub paths on the server.

For example, `/AptechDemo`

Cookie API 8-9

- **public Cookie[] getCookies()**
  - Reads the cookies from a request by using the `HttpServletRequest.getCookies()` method.
  - Returns an array containing all of the Cookie objects that the client sends with the request.
  - The code snippet demonstrates how to read cookies received in the client request.

```
// Retrieves cookies from the request object
   Cookie[] cookies = request.getCookies();
// Iterates through the array
   for(int i=0; i<cookies.length; i++) {
      Cookie = cookies[i];
    // Print cookie details
    out.println("Cookie Name: "+ cookie.getName();
    out.println("Cookie Value: " +
cookie.getValue();
}
```

Cookie API 9-9

- **`void addCookie (Cookie cookie)`**
  - Is Sent using `HttpServletResponse` object.
  - Adds field to the HTTP response headers to send cookies to the browser, one at a time.
  - Adds specified cookie to the response and can be called multiple times to set more than one cookies.

For example, `response.addcookie(new Cookie("cookiename","cookievalue"));`

Securing Cookies 1-4

- **Problems with Cookies**:
  - JavaScript can be used to access the cookies from the machine.
  - Cookies are available across scripts and may be accessed by hackers for manipulation.

- To secure the cookies from hackers on the Web, user can configure cookies with two security settings namely, `secure` and `HttpOnly`.

- `secure` flag:
  - Informs the Web browser that cookies should be sent only on the SSL connection.

- `HttpOnly` flag:
  - Informs the browser that the content of the cookie are not accessible within JavaScript.
  - Is included in the HTTP response header and prevents the cookies from certain kind of attacks.

Securing Cookies 2-4

- The `Cookie` class provides the following methods:

`public void setSecure()`

- This method informs the browser to send the cookie only through secured protocol, such as `HTTPS`.

`public void setHttpOnly(boolean)`

- This method can be used to mark or unmark the cookie.
- If set as true, then cookie is marked as **`HttpOnly`** and are not exposed to client-side scripting code.

`public boolean isHttpOnly()`

- This method is used to check whether the cookie has been marked as **`HttpOnly`**.

Securing Cookies 3-4

- The code snippet shows how to set the cookies as `HttpOnly` and `secure` flag programmatically in servlet.

```
protected void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException {
. . .

  Cookie = new Cookie("Color", "Cyan");
  Response.addCookie(cookie);
  cookie.setHttpOnly(true);
  cookie.setSecure(true);

. . .
  boolean status = cookie.isHttpOnly();
  out.println("<br>Status of Cookie - Marked as HttpOnly
= " + status);
}
```

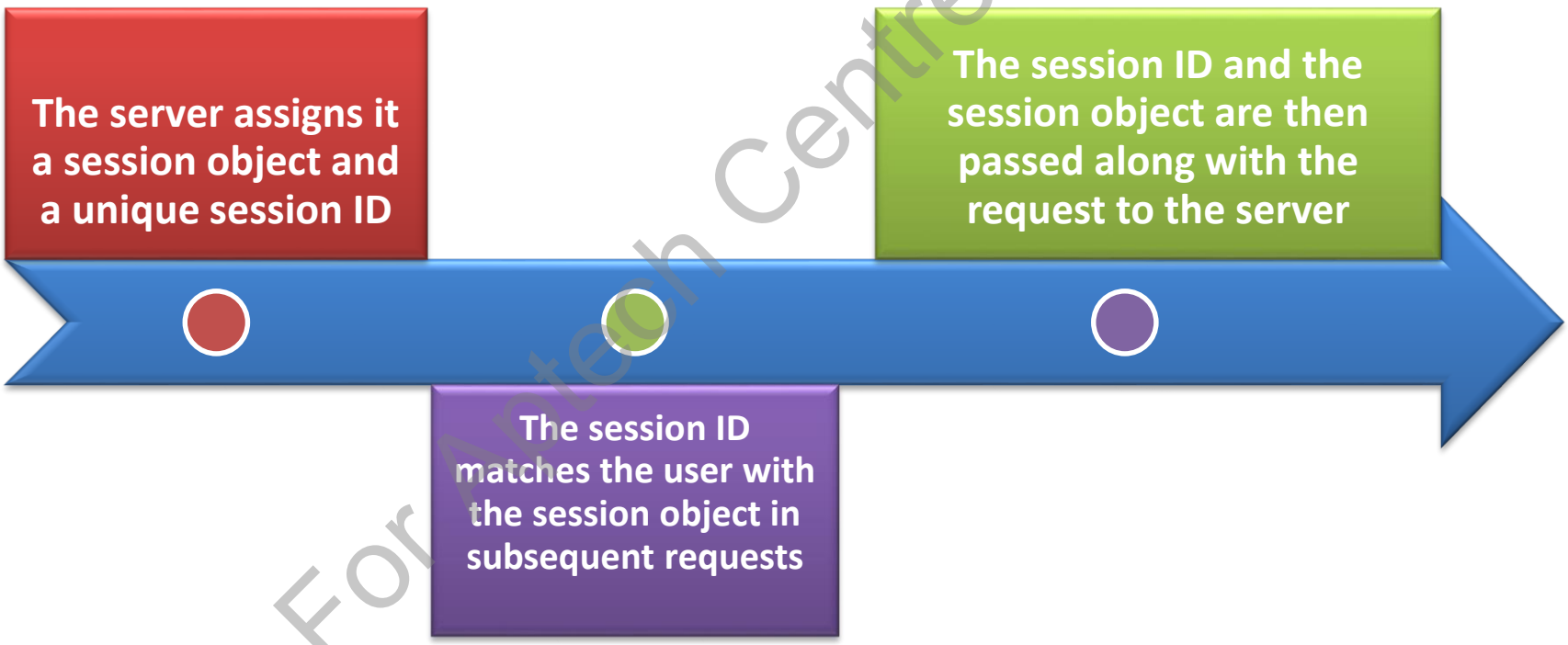
Securing Cookies 4-4

- Alternatively, the user can provide the declaration settings in the `web.xml` configuration file.
- The code snippet shows the `web.xml` file with security setting for a cookie.

```
. . .
<session-config>
   <cookie-config>
     <http-only>true</http-only>
     <secure>true</secure>
   </cookie-config>
<session-config>
. . .
```

HttpSession 1-3

- Session is created between an HTTP client and an HTTP server by the servlet container using this interface.
- `HttpSession` interface is used to create a session between the client and server.
- When users make a request:



HttpSession 2-3

- The methods of `HttpSession` interface used to create a session are:
- **`public Object getAttribute(String name)`**
  - Returns the object which is bound with the specified name in the session.
  - Returns null in case there is no object bound under the name.
- **`public String getId()`**
  - Returns a string containing the unique identifier assigned to this session.
- **`public int getMaxInactiveInterval()`**
  - Returns the maximum time interval, in seconds, for which the servlet container will keep the session alive between the client accesses.
- **`public ServletContext getServletContext()`**
  - Returns the `ServletContext` object to which the current session belongs.
- **`public void invalidate()`**
  - Invalidates the session and the objects bound to the session are unbounded.

HttpSession 3-3

- `public boolean isNew()`
  - Returns true if the client is unaware about the session or chooses not to be part of the session.
- `public void setAttribute(String name, Object value)`
  - Binds the object to the current session by using the specified name.
- `public void removeValue(String name)`
  - Removes the object bound with the specified name from the session.
- `public void setMaxInactiveInterval(int interval)`
  - Specifies the time, in seconds, between the client requests before the servlet container invalidates the current session.

Storing Information in a Session 1-2

- The data can be stored in an `HttpSession` object using the name-value pairs.
- The data which is stored is available throughout the current session.
- The method **`setAttribute()`** is used to store the data in a session.
- The code snippet demonstrates how to create a new session object and set object in it.

```
public void doGet( HttpServletRequest request,
HttpServletResponse ) throws IOException,
ServletException
{
  HttpSession httpSession = request.getSession(true);
// Gets current session or create a new one if not exist
    if (httpSession.isNew())
    {
     // Set the maximum interval for the session
         httpSession.setMaxInactiveInterval(60);
```

Storing Information in a Session 2-2

```
    // Sets the name attribute name as Jami
    httpSession.setAttribute("name", "Jenny");
    // Sets the attribute background colour to "#FFFFFF"
    httpSession.setAttribute("age", new Integer(20));

    }
}
```

- In the code,
  - `getSession()` returns the current session object associated with the request. If the session does not exists, then the boolean value true indicates to create a new session.
  - `isNew()` returns a boolean value indicating whether it is a new session or not. If it returns true, then the objects are bounded to the new session through `setAttributes()` method.
  - `setMaxInactiveInternal()` specifies the time between the requests from the client before the servlet container invalidates the session.

Retrieving Information Stored in a Session

- The code snippet explains the procedure for retrieving name and age stored in the session.

```
// Retrieves name and age from session
 String myText = (String)session.getAttribute("name");

 int myNumber = ((Integer)

 session.getAttribute("age")).intValue();
```

- The stored values in the session can be retrieved using `getAttribute()` method.
- Since, the return type is an object, typecasting of data associated with that attribute name in the session is done.

Invalidate a Session

- The `invalidate()` method is used to avoid the hacker from causing any harm to the Web application.
- It destroys the data in a session that another servlet or JSP might require in future.
- Sessions should be invalidated cautiously as they are associated with the client, not with individual servlets or JSP pages.
- The code snippet demonstrates invalidating the session.

```
// Returns current session or a new session if it does not exist
     HttpSession session = request.getSession (true);
// Checks the session
   if (session.isNew() == false) {
// Invalidates the session if it is not a new session
   session.invalidate();
// Creates a new session
   session = request.getSession (true);
  }
```

- The code directs the session to invalidate itself if it is not created newly.
- To invalidate the session manually, the `invalidate()` method should be called.

Session Timeout

- After a certain time period of inactivity the session is destroyed to prevent the number of sessions increasing infinitely.
- It happens if the user remains inactive for a period greater than the set inactive time period.
- The session timeout period can be set either in the web.xml file or can be set by the method `setMaxInactiveInterval()`.
- The setting for session time-out should be written in `web.xml` file.
- **Syntax**:

```
<session-config>
        <session-timeout>N</session-timeout>
</session-config>
```

  - `N` in the fragment is session timeout period.

Summary

- Session tracking allows the server to keep track of successive requests made by the same client.
- Some of the session tracking techniques are namely, URL rewriting, hidden field, and cookie.
- Java Servlet specification provides a session tracking mechanism through javax.servlet.http.HttpSession object.
- The URL rewriting technique adds some extra data at the end of the URL to identify the session.
- Hidden form fields are used to pass data to the server-side resource invisibly from the user.
- A cookie is a small piece of information sent by a server to the client Web browser. The cookies are stored on client's machine and are read back by the server on receiving request for the same page.
- To secure the cookies from hackers on the Web, you can configure cookies with two security settings namely, secure and HttpOnly.
- The HttpSession interface is used to create a session between the client and server. The session is created between an HTTP client and an HTTP server by the servlet container using this interface.