Introduction to Web Engineering

Lecture 4b

Sessions and Cookies

Review

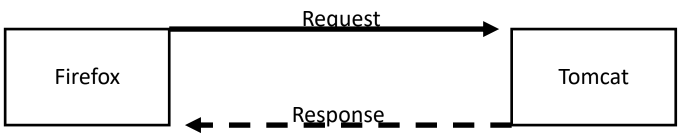
- Java Servlets
- Java Server Pages (JSP)
- Java Beans
- JSP Actions
- JSP Directives

This Lecture

- HyperText Transfer Protocol (HTTP)
- Cookies
- Sessions

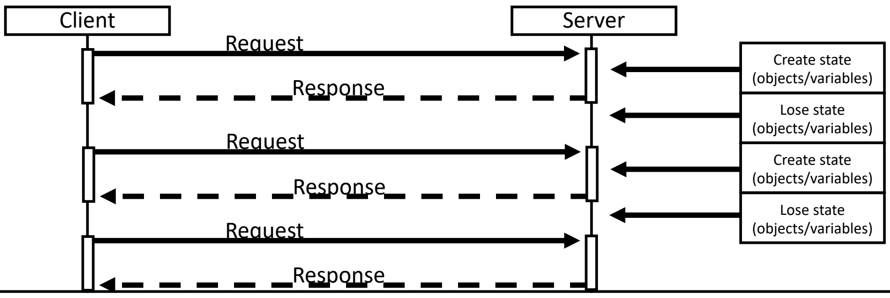
HyperText Transfer Protocol (HTTP)

- Communication protocol for transferring hypertext over a network
- Traditionally in a Client-Server manner
    - Client = Web Browser
    - Server = Tomcat
- Relies on the Request-Response model

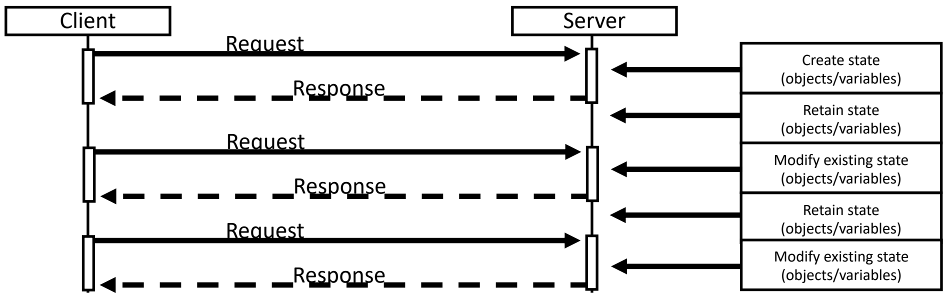


Stateless HTTP

- Each Request-Response pair executes independently from other Request-Response pairs, i.e:
  - Open a connection to the Web server and send request
  - Download the response document (HTML)
  - Close the connection
- No requirement for state to be tracked
- This needs to be done by an application that uses HTTP


Client
Server
Request
Response
Request
Response
Request
Response
Create state (objects/variables)
Lose state (objects/variables)
Create state (objects/variables)
Lose state (objects/variables)

Ideally


Client
Server
Request
Response
Request
Response
Request
Response
Create state (objects/variables)
Retain state (objects/variables)
Modify existing state (objects/variables)
Retain state (objects/variables)
Modify existing state (objects/variables)

Stateless HTTP

- HTTP/1.1 introduced some amount of state for performance reasons
  - Persistent connections
  - To download multiple documents/images/etc.
- These were network optimisations
- This state is transparent to the users of the protocol (the application developers)

HTTP 1.1 persistent connections

- An open connection may be kept open instead of closed
- Requests from the same browser may reuse this connection instead of starting another one
- The connection is closed after a short period of inactivity e.g. 30 seconds
- It doesn't answer the session management problem

Stateless HTTP

- Applications must do their own work to track a user over multiple requests
  - Called “session tracking”
- Tomcat gives us access to the ***Session*** object
- We can use the ***Session*** object to store information about:
  - The user and
  - Its interactions
- Tomcat keeps a reference to this object and **makes it available to our servlets and JSPs**

Cookies

A cookie is…

- Small pieces of textual information that a Web server sends to a client as part of a response
- When the client requests (again) something from the same server (or domain), it also sends the cookie information back to the server
- Allows the server to store user-dependent information across multiple requests
- Data can persist for (milli)seconds up to months

Cookies

- The information is stored on the user’s computer
  - Web browser might ask you when it is asked to set a cookie – unless this is disabled (often the default!)
- Restrictions on cookies – defined in RFC2109
  - 20 cookies per domain
  - 4096 bytes per cookie description
  - 300 cookies overall
  - FIFO removing system

Cookies

Uses

- Identifying a user during an e-commerce session – putting items into a shopping cart
- Avoiding username and password – popular with low-security sites
- Customizing a site - used by portals to remember look and feel selections
- Targeted advertising - directed rather than random ads

Cookies

Cookies have basic attributes…

- **name**
  - An identifier, e.g., my_yahoo_cookie
- **value**
  - String of characters
  - Same "encoding" as URLs, e.g., name=Joe%20Blog
- **domain**
  - Different web sites shouldn't see each other's cookies
  - Can share cookies with different URLs on same site

Cookies

- **path**
  - Restricts the cookies visibility to a part of the web server's directory tree
  - Useful for large sites that want multiple cookies
- **expiry time**
  - Dictates how long the client should keep the cookie
  - No expiry time (or 0) – discarded when the browser shuts down
- **secure flag**
  - Boolean – tells the browser to use Secure Socket Layer (SSL) requests when sending this cookie

Cookies

- When requesting a URL from an HTTP server, the browser will match the URL against all cookies and if any of them match, a line containing the name/value pairs of all matching cookies will be included in the HTTP request
  - Cookie: *NAME1=OPAQUE_STRING1*; *NAME2=OPAQUE_STRING2* ...

Cookies

- Example…
- Client requests a document, and receives in the response:
  ```
  Set-Cookie: CUSTOMER=WILE_E_COYOTE; path=/;
   expires=Wednesday, 09-Nov-19 23:12:40 GMT
  ```
- When client requests a URL in path / on this server, it sends:
  ```
  Cookie: CUSTOMER=WILE_E_COYOTE
  ```

Cookies

- Client requests a document, and receives in the response:
  ```
  Set-Cookie:   PART_NUMBER=ROCKET_LAUNCHER_0001;
   path=/
  ```
- When client requests a URL in path `/` on this server, it sends:
  ```
  Cookie:CUSTOMER=WILE_E_COYOTE;PART_NUMBER=ROCK
   ET_LAUNCHER_0001
  ```

Cookies

- Client receives:
  ```
  Set-Cookie: SHIPPING=FEDEX; path=/foo
  ```
- When client requests a URL in path `/` on this server, it sends:
  ```
  Cookie: CUSTOMER=WILE_E_COYOTE;
   PART_NUMBER=ROCKET_LAUNCHER_0001
  ```
- When client requests a URL in path `/foo` on this server, it sends:
  ```
  Cookie: CUSTOMER=WILE_E_COYOTE;
   PART_NUMBER=ROCKET_LAUNCHER_0001; SHIPPING=FEDEX
  ```

Cookies

1. Create a cookie using the constructor `Cookie`

   `Cookie c = new Cookie("userID", "c3014254");`

1. Set life span for the cookie

   ```
   c.setMaxAge(60*60*24*7); // one week
   c.setMaxAge(0); //To discard cookie
   ```

3. Add a cookie

   `response.addCookie(c);`

Cookies

- Java provides `javax.servlet.http.Cookie`
  - Public methods for manipulating cookies in both Java Servlets and JSPs
- `Cookie(String` *name*`, String` *value*`)`
  - Create a new cookie with ***name*** = ***value***
- `Cookie[] request.getCookies()`
- `response.addCookie(Cookie` *cookie*`)`
  - Pass cookies to and from the browser

Cookies

- The name of the cookie
  - `String getName()`
  - `void setName(String` *`name`*`)`
- Sets the value of the cookie
  - `String getValue()`
  - `void setValue(String` *`value`*`)`
- Time in seconds before cookie expires
  - `int getMaxAge()`

Cookies

- Sets the domain for which the cookie applies
  - `String getDomain()`
  - `void setDomain(String` *`pattern`*`)`
  - You can use this method to instruct the browser to return cookies to other hosts within the same domain

    ```
    cookie.setDomain(".vacations.com");
    ```

Cookies

- A path on the server to which the client should return the cookie
  - `String getPath()`
  - `void setPath(String` *`uri`*`)`
  - If not specified, the cookie is returned for all the URIs in the same directory as the current page as well as all subdirectories
- A secure encrypted protocol is indicated to the client for sending the cookie (false by default)
  - `boolean getSecure()`
  - `void setSecure(boolean` *`flag`*`)`

Cookies and Java – Example

```
<%
  String name = request.getParameter("name");
  String value = request.getParameter("value");
  Cookie cookie = new Cookie(name, value);
  cookie.setPath("/");
  cookie.setDomain("flame.newcastle.edu.au");
  cookie.setMaxAge(60);
  cookie.setSecure(false);
  response.addCookie(cookie);
%>
```

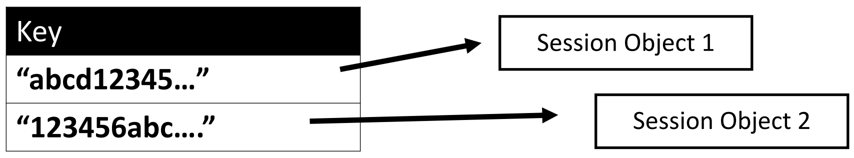
Cookies and Java – Example

```
<%
  Cookie[] cookies = request.getCookies();
  for (int i = 0; i < cookies.length; i++) {
%>
<tr>
  <td><%= cookies[i].getName() %></td>
  <td><%= cookies[i].getValue() %></td>
  <td><%= cookies[i].getDomain() %></td>
  <td><%= cookies[i].getPath() %></td>
  <td><%= cookies[i].getMaxAge() %> secs</td>
  <td><%= cookies[i].getSecure() %></td>
</tr>
<% } %>
```

Session

1. Client issues a request
2. Server creates a unique ***session id*** which it maps to a ***session object***
3. The server includes this ***session id*** with the response
   - Normally in a ***cookie***
4. The client sends the ***session id*** with each consecutive request
   - Normally in a ***cookie***
5. ***session object*** exists until it is invalidated, times out, or server shuts down

- This way the server can look up the correct ***session object*** for the request



Bringing it all Together

- Java web servers use cookies to store the ***session id***
  Set-Cookie:*JSESSIONID=A46A602EAEBBF9EFC07A2FC0634BCCAA; Path=/demo02/; HttpOnly*
- It is possible for an attacker to "steal" another person's ***session id***
  - Gaining access to another user's session object
- To avoid "Session Hijacking":
  - The ***session id*** should be long
    - Harder to guess
  - The ***session id*** should only ever be sent via an encrypted channel
    - HTTPS – more on this later

Session

- Using sessions in servlets is quite straightforward, and involves:
  1. Accessing the ***session object*** associated with the current request
  2. Looking up information associated with the session
  3. Storing information in the session
- To access the ***session object***:
  - In a servlet → *request.getSession();*
  - In a JSP → *getSession();*
  - Both return a *HttpSession* object

Session

- Once a session object is created on the server, it has a unique session id
  - Returns true if the session id of this request submitted, came in as part of a cookie
    `boolean isRequestSessionIdFromCookie()`
  - Returns true if the session id of this request submitted, came as part of the URL
    `boolean isRequestSessionIdFromUrl()`
  - Does this request have a valid session associated with it?
    `boolean isRequestSessionIdValid()`

Session

- `String getId()`
  - Return a string containing the unique identifier assigned to this session
- `long getCreationTime()`
  - Return the time this session object was created
- `boolean isNew()`
  - Returns true if the server has just created a session and the client is yet to use it

Session

- `long getLastAccessTo()`
  - Returns the last time the client sent a request associated with the session
- `int getMaxInactiveInterval()`
- `setMaxInactiveInterval(int` ***interval***`)`
  - The maximum number of inactive seconds that the server keeps this session open for
- `void invalidate()`
  - Expires the session and unbinds any objects bound to it

Session

- `Object getAttribute(String` ***name***`)`

  ```
  Class value = (Class) session.getAttribute("Identifier");
  ```
- `void setAttribute(String` ***name***`, Object` ***value***`)`

  ```
  session.setAttribute("Identifier", value);
  ```
  - Setting and getting attributes of a session
- `Object removeAttribute(String` ***name***`)`
  - Removes an attribute from the session
- `String[] getAttributeNames(String` ***name***`)`
  - Returns an array that includes the names of all the attributed stored in the session

Sessions – Example

```
HttpSession session = request.getSession();
 // Servlet only
Cart cart = (Cart)session.setAttribute("cart");
if (cart == null)
 cart = new Cart();
cart.addItem(id, quantity);
session.setAttribute("cart", cart);
```

Resources

- Tomcat HttpSession API
  - https://tomcat.apache.org/tomcat-8.0-doc/servletapi/javax/servlet/http/HttpSession.html
- Tomcat Cookies API
  - https://tomcat.apache.org/tomcat-8.0-doc/servletapi/javax/servlet/http/Cookie.html