ReadMe

dh626 Di Huang

1. **The overall structure of my solution.**

There are three source files in my solution. They are SessionManagementServlet.java, session.java and message.jsp.
SessionManagementServlet.java is used to realize the functionality of the servlet.
session.java is used to create session object that is used in SessionManagementServlet.java.
message.jsp is used to display the main web page and realize interaction with user.

(1) session.java

This class describe the session object. A session has a sessionID, a version number, a message and an expire time. So they are four private fields of session. Each filed has a get method and a set method. This class also includes a constructor.

(2) SeesionManagementServlet.java

This class is used to realize the functionality of the servlet.
The main logic of this class is:

```
boolean  newCookie = false; // a flag. If newCookie is true, that means a new session and a new cookie have been created to response to this request. There is no need to do other implementation later (e.g. update present session)
presentSession; //record present session
sessionCookie; // record cookie for present session

Clear time-out session in map;
Check if this request has cookie;

If (request doesn't have cookie){
    Create a new session;
    Create a new cookie;
    Put new session in map;

    newCookie = true;
}else (request has cookie){
    Check if map has the session indicated by the cookie
    If (map has this session){
        Check if this session in map is expired
    }

    if (map doesn't has this session || map has this session but this session has been timed out){
        Create a new session;
```

```
            Create a new cookie;
            Put new session in map;

            newCookie = true;
        }else (map has this session & this session still effective){
            retrieve the session from map as present session;
        }
    }

    if (replace is pressed){
        if (newCookie == false){
            get new message & check if it exceeds 512 bytes;
            get new version number;
            get new expire time;

            update cookie for this session;
            update this session in map;
        }
    }

    if (refresh is pressed){
        if (newCookie == false){
            get new version number;
            get new expire time;

            update cookie for this session;
            update this session in map;
        }
    }

    if (logout is pressed){
        expire the cookie;
        remove the session from map;
        display "log out" page;
        return;
    }

    attach cookie to the response;
    pass session info into message.jsp and direct there;
```

(3) <u>message.jsp</u>

Display the dynamic web page. Interact with user.
(just designed as the example in instruction)

2. **cookie format**
   cookie name = “CS5300PROJ1SESSION”
   cookie value = String “sessionID_Version_Location”

3. **session table**
   The session table is realized by a CocurrentHashMap to solve the synchronization issue. The key for this map is sessionID (UUID), and the value of the map is the session associated with the sessionID.

4. **Clear time-out session**
   The time-out sessions are cleared each time the doPost() method is evoked. So it is implemented on the request-processing threads.
   Each time it iterates the whole map and removes sessions whose expiration time is earlier than present time.
   Also, if a session associated with the cookie is found in map but it has been expired, it will be removed as well.

5. **Deploy .war file**
   Drop the .war file into path/Tomcat/webapps
   Then run the startup.sh file in Tomcat/bin with terminal.
   Now Tomcat should start.
   Then tap http://localhost:8080/cs5300/ in browser, carriage return. The web page should be displayed.