Yi Li (yl2326)
Patrick Berens(pcb73)
Max Spector (ms2786)

Table of Contents

1) Overview
2) Functionality
3) Program Properties and Operations
 a. Concurrency
 b. Removing stale sessions
 c. Storing sessions
 d. Canonicalizing the cookie values
 e. Canonicalizing the cookie values
 f. Creating session IDs (for now and future sessions)
 g. Limiting the size of the session state value
3) Testing
4) How to run our code
5) Using the deployable .war file

1) Overview

In this project, we developed a scalable website. This site is done by writing Servlet code that runs on a single machine. This site maintains and displays a small session state with session timeout implemented. The site has three buttons: 'Refresh', 'Replace', and 'Logout'. Session management is done through passing a cookie between the client and server.

2) Functionality

In the Servlet code, the function 'doGet' is called when a GET request is made. First, the Servlet checks if the client has an existing, non-expired session. If this is not the case, the Servlet creates a new session and sends the client a cookie. If the client has an existing, non-expired session, then the Servlet updates the state of the session in a thread safe manner. Once these operations are done, it returns to the client the updated cookie. The website is updated to reflect any changes in state (i.e. the message) as well as the new expiration time of the session.

3) Program Properties and Operations

a. Concurrency

First, we used a ConcurrentHashMap to make sure that all hash table operations are atomic. Next, we used locks (Java's Reentrant Lock) to make sure our program is concurrent. Each session has a corresponding lock that is associated with it. When there are changes being made in regards to that specific session, then the server acquires the corresponding lock. When the work is done, the lock is released. This prevents the contents of the state table from being trashed. Furthermore, this allows for parallel threads to be safely modify the contents of the state table. There is no global lock on the table or on the locks table. This is to allow optimistic parallelization.

We chose to use this implementation because it allows two unique sessions to write to the session data table, while making sure we do not update update the value associated with the same key in the hash table concurrently. This implementation was chosen over using a 'synchronized(this)' over the critical parts of the program because using the 'synchronized(this)' would prevent different threads from accessing the critical parts of the program even if the threads were unrelated to each other.

There is also a lock that is acquired when a cookie is created for the first time (this allows for unique sessionIDs to be generated).

In using session based locks to ensure that the critical sections of the code are thread-safe as well as using the ConcurrentHashMap data structure's properties, we have built a program that is safe and concurrent.

b. Removing stale sessions

We use a "daemon" thread that wakes up every 60 seconds and removes timed-out session from the table. We also use the same session level locks to make sure this is thread safe.

c. Storing sessions

We store our sessions in Java's ConcurrentHashMap data structure. This data structure has a mapping of sessionID to version number, message, and expiration-timestamp.

d. Cookie details

Session management is done through using a Cookie to store information, not an HttpSession. The cookie containing the sessionID, version number, and location metadata is stored on the client, while the server stores information such as the sessionID, version number, message, and expiration date.

e. Canonicalizing the cookie values

When the msg is read by the Servlet, a regular expression is used to make sure the input is restricted to “safe” characters such as letters and “0123456789.-_”.

f. creating session IDs (for now and future sessions)

Currently, a counter begins at 0 and increments whenever the servlet receives a new request for a session. When there are multiple servers, a string that corresponds to the server is appended to the counter value so newly generated session IDs are globally unique.

g. Limiting the size of the session state value

The message field input is restricted to 400 characters. This ensures that the session state value is less than 512 bytes.

3) Testing

a) Testing the ‘Refresh’

To test ‘Refresh’, run the Servlet and press ‘Refresh’. The expiration time of the session should be updated with a new expiration time.

b) Testing the ‘Replace’

To test ‘Replace’, run the Servlet, enter some text to the left of ‘Replace’ and press Replace. The expiration time of the session should be updated with a new expiration time and the top of the screen should be replaced with the new text.

c) Testing the ‘Logout’

To test ‘Logout’, start by running the Servlet, entering some text to the left of ‘Replace’ and pressing Replace. The expiration time of the session should be updated with a new expiration time and the top of the screen should be replaced with the new text. Next, press ‘Logout’. This should update the expiration time as well as return the default page.

d) Testing a website refresh

To test a website refresh, run the Servlet, enter some text to the left of ‘Replace’ and press Replace. The expiration time of the session should be updated with a new expiration time and the top of the screen should be replaced with the new text. Now, refresh the page (F5 or Ctrl-R). The expiration time of the session should be updated with a new expiration time.

e) Testing the session expiration

To test the cleaning of old session states, start by running the Servlet, entering some text to the left of ‘Replace’ and pressing Replace. The expiration time of the session should be updated with a new expiration time and the top of the screen should be replaced with the new text. Next wait for 60 seconds; afterwards refresh the website. This should update the expiration time as well as return the default page.

f) Testing the concurrent access of one cookie

To test this, start by running the Servlet, entering some text to the left of ‘Replace’ and pressing Replace. The expiration time of the session should be updated with a new expiration time and the top of the screen should be replaced with the new text. Next, open a new tab and access the Servlet. The modified text should exist in the newly opened tab.

4) How to run our code

See the section ‘Using the deployable .war file’.

5) Using the deployable .war file

Place the “cs5300p1.war” file into the web_app directory of Apache Tomcat server. Make sure the Apache Tomcat server has started. Next, run: “http://localhost:8080/cs5300p1/msgcookieservlet” in a browser. The Servlet is now live.