Code assignment made by Wouter De Geest on 2014/06/25

Required JDK and language level: 1.8

Quick launch

Unzip king.zip

Run the king.jar file on the command line (java –jar king.jar)

The server will start running on port 8081 and is ready to accept incoming requests.

Following requests will be accepted:

GET /<userid>/login

GET /<levelid>/highscorelist

POST /<levelid>/score?session=<sessionid> with request body {score=<value>}

Successful requests will receive response code 200, failed requests such as parse errors on the URI string or expired sessions will result in error codes above 200 and accompanying error messages.

General comments

Given the importance of concurrency I have tried to take special attention to making this code parallelizable, thread-safe and at the same time avoiding as much as possible heavy locking schemes like synchronized blocks.

The following is a list of choices made to implement these concurrency ideas:

Http server: Uses a thread pool executor which will help in handling a large amount of asynchronous tasks.

Data structures: Using ConcurrentHashMap to avoid locking the whole map but instead locking on the buckets. Also used ConcurrentSkipListSet to have a threadsafe sorted set.

Parallel streaming: Java 8 offers a streaming api that allows lazy streams to be executed in parallel. This is perfect for eg iterative operations over large collections where the work can be divided such as sorting large collections.

Code-style:

1. Wherever possible I have used immutable classes. This by default avoids thread issues.
2. Wherever feasible I have used a java 8 functional programming style again to avoid unexpected thread synchronization on eg iterators.
3. All singletons are written thread-safe using the classloader approach.

Class overview

Package Server

Server [class]: Launces the http server and communicates requests. The server is thread pooled.

Filter [class]: Attaches to incoming requests to allow for preprocessing the request in a more structured manner. In this case, it means creating a Command object.

Command [class]: A command object describes which action to perform and holds the necessary data to perform that action.

RequestMethod [enum]: One of get|put

Action [enum]: one of login|score|highscorelist

Handler [class]: Responsible for forwarding requests to the correct controller with the Command object as input. It retrieves a Response object from the controller and translates this back to the requester using the requestbody.

SessionCleaner [class]: Thread-safe singleton that periodically and in a seperate thread cleans up expired sessions from the SessionStore

Client [class]: Just a test class to send post requests to the server (for sending scores).

Package Controller

Controller [interface]: Describes the contract for calling controllers.

LoginController [class]: Logs in the user by creating a session and storing it into the SessionStore singleton.

ScoreController [class]: Allows to publish scores for a certain level for a user (using its session key) to the ScoreStore singleton.

HighScoreListController [class]: Allows to retreive a cvs structured high score list for a level (having only the highest score per user)

Package Domain

Score [class]: Immutable domain class that represents a score made by a user for a certain level.

It also holds the time the score was achieved. Score implements the Comparable interface as to allow for natural order sorting.

ScoreStore [class]: Thread-safe singleton that stores scores.

I choose to use a multi-level hashtable with the "level" as index for the first level and "user" as index for the second level.

The leaves then contain the actual scores in ascending natural order. This allows for some flexibility in querying different scenarios in time complexity of O(1).

For example to get the high scores of one specific level or to get the scores of specific users for that level. The scores are kept sorted on the leaves. Important to note is that this datastructure is thread-safe by using ConcurrentMap and ConcurrentSkipListSet.

When querying I believe it to be generally a good idea to write it functional especially now that java 8 gave as our new lambda language features and streaming api.

The benefits are huge: you write immutable code and allow for parallel processing over multiple cores using the stream api.

As an example I wrote the getHighScores~() method using these new features.

Session [class]: Immutable domain class that represents a session.

It holds the associated user and the time in milliseconds on creation.

Furthermore it generates a session key using the UUID random generator striped from the underscores fulfulling the regex [A-Za-z0-9].

SessionStore [class]: Thread-safe singleton class that stores sessions.

Sessions become invalid after 10 minutes.

User [class] Immutable domain class that allows to model the actual users using the system. For now it just holds its id.

Package Utils

Contains the StringUtils class that provides some useful static utility methods.