CS2030 Lab #9: Bus API

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Bus

Topic Coverage

- Asynchronous programming
- CompletableFuture
- Lambda functions

Problem Description

We have a Web API online for querying bus services and bus stops in Singapore. You can go ahead and try:

- https://cs2030-bus-api.herokuapp.com/bus_services/96 returns the list of bus stops (id followed by description) served by Bus 96.
- https://cs2030-bus-api.herokuapp.com/bus_stops/16189 returns the description of the stop followed by a list of bus services that serve the stop.

(note: our database is two years old though -- don't rely on this for your daily commute!)

In this lab, we will write a program that uses the Web API to do the following:

Given the current stop `S`, and a search string `Q`, returns the list of buses serving `S` that also serves any stop with description containing `Q`. For instance, given `16189` and `Clementi`, the program will output

```
Search for: 16189 <-> Clementi:
From 16189
- Can take 96 to:
  - 17171 Clementi Stn
  - 17091 Aft Clementi Ave 1
  - 17009 Clementi Int
- Can take 151 to:
  - 17091 Aft Clementi Ave 1
- Can take 151e to:
  - 17091 Aft Clementi Ave 1

Took 11,084ms
```

The pairs of `S` and `Q` are entered through the standard input with every pair of `S` and `Q` in a separate line.

```
08031 Orchard
17009 NUS
17009 MRT
15131 Stn
08031 Int
```

A working program has been written and is available here. Download and study the program to understand what it does and how it works. Keep a copy of program around for comparison and reference later.

The given program is written synchronously. Every query to the Web API is done one-by-one, and the program has to wait until one query completes before it can continue execution of the program. As a result, the program is slower than it should be.

The Task

Your task, for this lab, is to change the given program so that it executes asynchronously. Doing so can significantly speed up the program.

The root of synchronous Web API access can be found in the method `httpGet` in `BusAPI.java`, in which the invocation of method `send` from the class `HttpClient` is done synchronously (i.e., it blocks until the response returns).
`HttpClient` also provides an asynchronous version of `send` called `sendAsync`, which is exactly the same as `send` except that it is asynchronous and returns a `CompletableFuture>` instead of `HttpResponse`. (You do not need to get into the nitty-gritty details of the `HttpClient` and `HttpResponse` for this lab -- but they are still good to know, so read up about them at your leisure).

To make the program synchronous, you should first change the invocation of `send` in `BusAPI` to `sendAsync`. All other changes will be triggered by this.

It is important to note that your should not call `CompletableFuture::join` prematurely, so that everything that has been **promised** so far, from the lower-level Web API calls to the higher-level logic of searching for bus services, are done asynchronously. The only place where you should `join` is in the `main()` method after waiting for all the `CompletableFuture` objects to complete using `allOf`. The final step would then be to print out the description of the bus routes found.

The speed-up you would experience for the asynchronous version depends on the complexity of the inputs. For the following test input:

```
08031 Orchard
17009 NUS
17009 MRT
15131 Stn
08031 Int
12345 Dummy
```

a typical reduction in the time from around 150-180 seconds to 10-15 seconds (i.e. more than 10 times speed up) is expected. Your milage may vary, but you should see some speed up in the total query time.

This task is divided into several levels. Read through all the levels to see how the different levels are related.

Remember to: * always compile your program files first before either using `jshell` to test your program, or using `java` to run your program * run `checkstyle` on your code

For this lab, you are also required to write `javadoc` comments and make sure you can generate the `javadoc` HTML files. You are not required to submit the `javadoc` HTML files, but they will be generated later based on your code.

Level 1: `BusAPI.java`

Start by changing the method call `HttpClient::send` to `HttpClient::sendAsync` in the method `httpGet`. In the skeleton code provided, notice that as soon as the `send` method returns with a `HttpResponse`, it proceeds to check the status. Since the `send` could take a long time to complete, we would have to wait for it to complete first.

On the other hand, using `sendAsync` returns a response wrapped within a `CompletableFuture`; it is a **promise** that a response will be returned (not now, but eventually). The part of the code that deals with the response can now be attached as a callback to this `CompletableFuture`.

Modify the `BusAPI.java` program such that sending a `http` request is now done asynchronously. You will also need to modify the `getBusStopsServedBy` and `getBusServicesAt` methods accordingly.

Compile your java program as follows:

```
$ javac -Xlint:rawtypes BusAPI.java
```

Once compiled, you can update the given jar file and test your program.

```
$ jar uf level1.jar *.class
$ echo "16189 Clementi" | java -jar level1.jar
Search for: 16189  <-> Clementi:
From 16189
- Can take 96 to:
```

```
  - 17171 Clementi Stn
  - 17091 Aft Clementi Ave 1
  - 17009 Clementi Int
- Can take 151 to:
  - 17091 Aft Clementi Ave 1
- Can take 151e to:
  - 17091 Aft Clementi Ave 1

Took 3,131ms
```

You may now proceed to submit `BusAPI.java` and test if this level passes. CodeCrunch will provide the asynchronous versions of the other classes.

Level 2: `BusService.java` and `BusStop.java`

Now compile the `BusService.java` class:

```
$ javac -Xlint:rawtypes BusService.java
```

and take note of the compilation errors. Specifically, the `getBusStops` method should now return a **promise** of a `Set`.

In this case, as soon as `BusAPI::getBusStopsServedBy` method completes, a `Scanner object` can then be created to proceed with reading user input.

Likewise, modify the `findStopsWith` method accordingly.

Compile your java program as follows:

```
$ javac -Xlint:rawtypes BusService.java
```

Once compiled, you can update the given jar file and test your program.

```
$ jar uf level2.jar *.class
$ echo "16189 Clementi" | java -jar level2.jar
Search for: 16189  <-> Clementi:
From 16189
- Can take 96 to:
  - 17171 Clementi Stn
  - 17091 Aft Clementi Ave 1
  - 17009 Clementi Int
- Can take 151 to:
  - 17091 Aft Clementi Ave 1
- Can take 151e to:
  - 17091 Aft Clementi Ave 1

Took 3,131ms
```

You may now proceed to submit `BusAPI.java` and `BusService.java`, in order to test if this level passes. CodeCrunch will provide the asynchronous versions of the other classes.

Level 3: `BusSg.java` and `BusRoutes.java`

Within the `BusSg` class, modify the `getBusServices` method to return a **promise** of a `Set`. In addition, the `findBusServicesBetween` method needs to be modified accordingly. This will trigger a change in the constructor of the `BusRoutes` class.

Within the `BusRoutes` class, the `Map` of bus services should now map a bus service to a **promise** of a `Set`. When generating the description of the bus route, the original method goes through the bus services with the route, and for each bus service, it obtains a set of matching bus-stops and calls `decribeService` to return a `String` representing the bus service and its matching stops. With the new implementation, the set of matching bus-stops is now a **promise**. You will need to be able to combine these **promises** together, and return the string representation of the bus route as a **promise**.

Compile your java program as follows:

```
$ javac -Xlint:rawtypes BusSg.java
```

Once compiled, you can update the given jar file and test your program.

```
$ jar uf level3.jar *.class
$ echo "16189 Clementi" | java -jar level3.jar
Search for: 16189  <-> Clementi:
From 16189
```

```
- Can take 96 to:
  - 17171 Clementi Stn
  - 17091 Aft Clementi Ave 1
  - 17009 Clementi Int
- Can take 151 to:
  - 17091 Aft Clementi Ave 1
- Can take 151e to:
  - 17091 Aft Clementi Ave 1

Took 3,131ms
```

You may now proceed to submit `BusAPI.java`, `BusService.java`, `BusRoutes.java` and `BusSg.java`, in order to test if this level passes.

Level 4: `Main.java`

In the `Main` class, you will now need to construct all the **promises** as `CompletableFuture` objects and wait for all of them to complete. Finally, output the description of the bus routes.

A sample run of your java program as follows:

```
$ javac -Xlint:rawtypes Main.java
$ echo "16189 Clementi" | java Main
Search for: 16189  <-> Clementi:
From 16189
- Can take 96 to:
  - 17171 Clementi Stn
  - 17091 Aft Clementi Ave 1
  - 17009 Clementi Int
- Can take 151 to:
  - 17091 Aft Clementi Ave 1
- Can take 151e to:
  - 17091 Aft Clementi Ave 1

Took 3,131ms

$ cat test.in
16189 Clementi
17009 NUS

$ cat test.in | java Main
Search for: 16189  <-> Clementi:
From 16189
- Can take 96 to:
  - 17171 Clementi Stn
  - 17091 Aft Clementi Ave 1
  - 17009 Clementi Int
- Can take 151 to:
  - 17091 Aft Clementi Ave 1
- Can take 151e to:
  - 17091 Aft Clementi Ave 1

Search for: 17009  <-> NUS:
From 17009
- Can take 196 to:
  - 17191 NUS High Sch
- Can take 156 to:
  - 41011 NUS Bt Timah Campus
- Can take 96 to:
  - 16169 NUS Raffles Hall
  - 16199 NUS Fac Of Design & Env
  - 16149 NUS Fac Of Architecture
  - 16159 NUS Fac Of Engrg

Took 3,924ms
```

You may proceed to submit your java files.