CodeCrunch

CS2030 (2310) Exercise #6: Bus

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

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://raw.githubusercontent.com/nus-cs2030/2324-s1/main/bus_services/96 returns the list of bus stops (id followed by description) served by Bus 96.
- https://raw.githubusercontent.com/nus-cs2030/2324-s1/main/bus_stops/16189 returns the description of the stop followed by a list of bus services that serve the stop.

Note: only data necessary for the completion of this exercise is provided.

Given a bus stop `S`, and a search string `Q`, we can find the list of buses serving `S` that also serves any stop with a description containing `Q`. For instance, given `S = 16189` and `Q = "Clementi"`,

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

Task

In this lab, we will write a program that first reads pairs of `S` and `Q` from user input, with every pair of `S` and `Q` in a separate line, and then answering each query by using the Web API.

```
$ cat test.in
17009 NUS
17009 MRT
15131 Stn

$ cat test.in | java Main
Search for: 17009 <-> NUS:
From 17009
- Can take 156 to:
  - 41011 NUS Bt Timah Campus
- Can take 196 to:
  - 17099 Yale-NUS College
  - 17191 NUS High Sch
- Can take 96 to:
  - 16169 NUS Raffles Hall
  - 17099 Yale-NUS College

Search for: 17009 <-> MRT:
From 17009
- Can take 173 to:
  - 28101 Opp SMRT Ulu Pandan Depot
  - 28109 SMRT Ulu Pandan Depot

Search for: 15131 <-> Stn:
From 15131
- Can take 200 to:
  - 11361 Buona Vista Stn Exit C
  - 11369 Buona Vista Stn Exit D
  - 15131 Kent Ridge Stn
  - 15139 Kent Ridge Stn Exit B
  - 16011 Opp Haw Par Villa Stn
  - 16019 Haw Par Villa Stn
- Can take 92 to:
  - 11361 Buona Vista Stn Exit C
  - 11369 Buona Vista Stn Exit D
  - 15131 Kent Ridge Stn
  - 15139 Kent Ridge Stn Exit B

Took 2,448ms
```

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.

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

Level 0

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. *You do not need to get into the nitty-gritty details of the `HttpClient` and `HttpResponse` for this lab -- you can read up about them at your leisure if you are interested.*

Let us test the synchronous `send` using the `Main` class below:

```
$ cat Main.java
class Main {
    public static void main(String[] args) {
        System.out.println(BusAPI.getBusStopsServedBy("95"));
        System.out.println(BusAPI.getBusStopsServedBy("96"));
    }
}

$ javac Main.java

$ java Main
11181,Opp Blk 43
11189,Blk 43
11259,Opp Blk 21
11261,Holland Village
11361,Buona Vista Stn Exit C
11369,Buona Vista Stn Exit D
11381,Blk 10A
11401,Holland V Stn/Blk 12
16009,Kent Ridge Ter
16009,Kent Ridge Ter
16171,Yusof Ishak Hse
16179,Opp Yusof Ishak Hse
16181,Ctrl Lib
16189,Information Technology
16191,Aft Kent Ridge Dr
```

```
16199,Bef Architecture Dr
18121,Opp Ayer Rajah Ind Est
18129,Ayer Rajah Ind Est
18141,Aft Anglo-Chinese JC
18149,Essec Business Sch
18221,NUH
18239,Opp NUH
18301,Lim Seng Tjoe Bldg (LT 27)
18309,Blk S17
18311,Blk S12
18319,Opp University Hall
18321,Opp University Health Ctr
18329,University Health Ctr
18331,Kent Ridge Stn Exit A/NUH
18339,Opp Kent Ridge Stn Exit A

16149,SDE3
16159,Coll of Design & Engrg
16169,NUS Raffles Hall
16179,Opp Yusof Ishak Hse
16189,Information Technology
16199,Bef Architecture Dr
17009,Clementi Int
17009,Clementi Int
17091,Aft Clementi Ave 1
17099,Yale-NUS College
17151,Blk 410
17159,Blk 365
17161,Blk 455
17171,Clementi Stn Exit A
17239,NTUC Fairprice
```

Due to the synchronicity of computation, the bus stops for service 95 are always output before those of service 96.

To make the program asynchronous, let us change the invocation of `send` in `BusAPI`. `HttpClient` provides an asynchronous version of `send` called `sendAsync`, which is exactly the same as `send` except that it is asynchronous and returns a `CompletableFuture<HttpResponse>` instead. By returning a response wrapped within a `CompletableFuture` — it is a **promise** that a response will be returned (not now, but eventually).

```
...
import java.util.concurrent.CompletableFuture;

class BusAPI {
    ...

    private static CompletableFuture<String> httpGet(String url) {
        HttpClient client = HttpClient.newBuilder()
            .build();
        HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(url))
            .build();

        return client.sendAsync(request, BodyHandlers.ofString())
            .thenApply(r -> {
                if (r.statusCode() != HTTP_RESPONSE_STATUS_OK) {
                    System.out.println(r + " " + r.statusCode());
                    return "";
                }
                return r.body();
            });
    }

    public static CompletableFuture<String> getBusStopsServedBy(String serviceId) {
        ...

    public static CompletableFuture<String> getBusServicesAt(String stopId) {
        ...
```

Note that unlike `HttpClient::send` which requires the checked exceptions `IOException` and `InterruptedException` to be caught or thrown, `sendAsync` no longer requires them to be caught.

We now modify our `main` method as follows:

```
$ cat Main.java
import java.util.concurrent.CompletableFuture;

class Main {
    public static void main(String[] args) {
        CompletableFuture<Void> cf1 = BusAPI.getBusStopsServedBy("95")
            .thenAccept(x -> System.out.println(x));
        CompletableFuture<Void> cf2 = BusAPI.getBusStopsServedBy("96")
            .thenAccept(x -> System.out.println(x));
        cf1.join();
        cf2.join();
    }
}

$ javac Main.java
```

Now, run `java Main` several times and you will notice that the output of bus stops of service 96 *may* come before those of service 95, or vice-versa. You may also try running program by removing the `join` statements and see what happens.

In the remainder of this exercise, you are to refine the higher-level logic of searching for bus services such that computation proceeds asynchronously. It is important to note that you 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 are done asynchronously.

Level 1

Now compile the `BusService.java` class and take note of the compilation errors.

Specifically, the `getBusStops` method should now return `CompletableFuture<List<BusStop>>` — a **promise** of a `List<BusStop>`. As soon as the `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 BusAPI.java BusStop.java BusService.java
```

and submit the three files to CodeCrunch of testing.

Level 2

Within the `BusSg` class, modify the `getBusServices` method to return a **promise** of a `List<BusServices>`. 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 `List<BusStops>`. 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 BusAPI.java BusStop.java BusService.java BusRoutes.java BusSg.java
```

and submit the five files to CodeCrunch of testing.

Level 3

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

You should first create a `List<CompletableFuture<BusRoutes>>` and have the individual queries run asynchronously. Thereafter, collate the responses and output the description of the bus routes.

A sample run of your java program as follows:

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

Took 729ms

$ cat test.in
17009 NUS
17009 MRT
15131 Stn

$ cat test.in | java Main
Search for: 17009 <-> NUS:
From 17009
- Can take 156 to:
  - 41011 NUS Bt Timah Campus
- Can take 196 to:
  - 17099 Yale-NUS College
  - 17191 NUS High Sch
- Can take 96 to:
  - 16169 NUS Raffles Hall
  - 17099 Yale-NUS College

Search for: 17009 <-> MRT:
From 17009
- Can take 173 to:
  - 28101 Opp SMRT Ulu Pandan Depot
  - 28109 SMRT Ulu Pandan Depot

Search for: 15131 <-> Stn:
From 15131
- Can take 200 to:
  - 11361 Buona Vista Stn Exit C
  - 11369 Buona Vista Stn Exit D
  - 15131 Kent Ridge Stn
  - 15139 Kent Ridge Stn Exit B
  - 16011 Opp Haw Par Villa Stn
  - 16019 Haw Par Villa Stn
- Can take 92 to:
  - 11361 Buona Vista Stn Exit C
  - 11369 Buona Vista Stn Exit D
  - 15131 Kent Ridge Stn
  - 15139 Kent Ridge Stn Exit B

Took 882ms
```

You may proceed to submit your java files to CodeCrunch for testing.

Submission (Course)

Select course: CS2030 (2023/2024 Sem 1) - Programming Methodology II

Your Files:

BROWSE

SUBMIT (only .java, .c, .cpp, .h, .jsh, and .py extensions allowed)

To submit multiple files, click on the Browse button, then select one or more files. The selected file(s) will be added to the upload queue. You can repeat this step to add more files. Check that you have all the files needed for your submission. Then click on the Submit button to upload your submission.