Part 1 - Implementation Summary：https://github.com/tangxiaocheng/SpigotCaseStudy.git

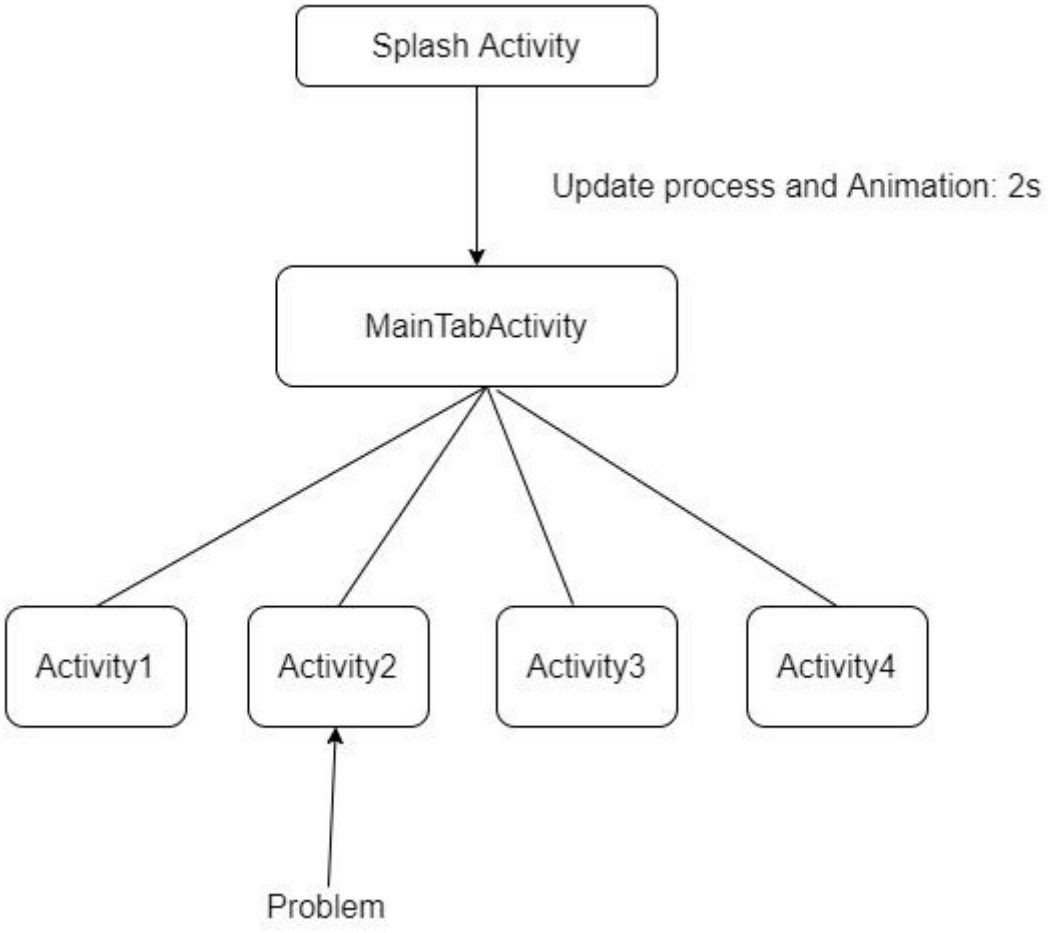
1. For the room data model of DeviceInfo, it has four fields, they are the "baseUrl", the whole parameter "json" string, the "createdTime", and a primary key "id".
2. For the "json" field, it is a pure json string that serialized from a map data structure
3. The RecyclerView will display the data in the DESC order of their model's "createdTime".
4. For the ItemViewType, it depends on the id's odd-even characteristic. IF it's even, it corresponds to "LeftViewHolder".
5. For the RecyclerView adapter, I used PagedListAdapter, DiffUtil, and overridden hashcode and equals method of DeviceInfo for high performance.
6. For the network, I used the Retrofit library in two ways as a comparison, one is combined with RxJava Single. One is a typical CallBack. please check out RightPostActivity and LeftPostActivity.
7. During the process, I added Timber and ChuckerInterceptor for the debugging purpose.
8. One last thought here base on this study case and my experience is that, in a production environment, we could add an interceptor for the OkHttpClient to uniformly add the device info parameters to every request.

Part 2 – Written Prompt

For this part, I would like to share my experience of fixing a critical bug in a project.

Background:

1. It's before Android 4.0, which version name is Honeycomb. There is no NetworkOnMainThreadException at that time. Otherwise, it would be easier to locate the problem.
2. For the application, it's a SplashActivity to MainTabActivity Structure. The SplashActivity has Animation and Update logic in it. The MainTabActivity has four sub Activities. Each of the activities has heavy logic there.



3. For the codebase, we just replaced the outdated network library with the best-practice library which significantly simplified our network request code.

Problem:

1. On the subway to the office, I customarily opened the application for testing around, all of a sudden, I noticed our app totally freeze for about 4 seconds before entering into the home page. However, I was not able to reproduce it.

Initial analyze:

1. Asked around to check if anyone else encountered this appearance. But nobody saw what I said.
2. Checked out the application prototype for the new app launching business demands.
3. Reviewed the code regarding the app launching process. For this, I was able to quickly exclude this part because I just finished this module in the previous version.
4. I excluded the network library because, since the replacement, the app had great improvement in its performance.

Process:

1. Since I wasn't able to reproduce the bug, I moved on to my other tasks, and I marked the problem down as a critical potential issue.
2. In the following two weeks, during daily development, I strongly sensed that something wrong with the application launching process.
3. Right before the product release night, at that moment, I did have too much on my plate, so I decided to find the cause of this abnormal launching process.

How I found it?

1. After searched around in the codebase for hours, I got hungry, so I grab my phone to the snack machine, on my way, I opened the app again, for this time, it was way slower than usual. Normally, the launch time is like 2 seconds but for that time it's about 2.7 seconds. (I was working on this problem the whole time, I could clearly tell the difference).
2. After I finished my snack, I recapped the whole process and compared it with the time on the subway. I drew an initial assumption which is the app is problematic when it's "on the move".(Very funny assumption here)
3. To verify my assumption, I walked around the whole office, eventually, I reproduced the bug in an office corner. (At that moment, I was so excited.), so I moved my computer to the corner and I started a debug process.
4. With debug on the computer, I quickly located the bug, which is a network request on the main thread

Why is it a difficult problem?

1. The manager just verbally asked my coworker to add a post request for statistical purposes without any documentation.
2. The coworker didn't use the standard network module for this particular network request. Instead, he just the old util method without putting it in a thread.
3. He put this online code in a sub-activity of the MainTabActivity, which I didn't check in the first place.
4. The request is right after the launching process, it process time mixed with launching time, this made it very hard to spot.

Afterward

1. After I fixed and reported this bug, our manager initiate a monthly meeting to share and learn new techniques for the new frameworks.
2. Later on, Google introduced "NetworkOnMainThreadException", which avoided this problem in the very first place.