Internet Connectivity: Part 2

Recap

- We studied how to parse JSON data.
- We saw how to open a connection.
- We read in a string (corresponding to JSON) using the Scanner trick.
- We saw the now deprecated AsyncTask framework
- We wrote our AsyncTask replacement

Today

- We’ll learn to fetch data using Google’s Volley library.
- We’ll learn to use DownloadManager to fetch larger amounts of data. This is a kind of service, and it implements a broadcast receiver.
- We’ll also learn how to use JobScheduler to get data in the background.

The Volley Library

Why Volley?

- Automatic scheduling of network request.
- Multiple concurrent network connections.
- Request prioritization.
- Canceling requests easily.
- Works best for smaller amounts of data. Google recommends using DownloadManager/JobScheduler for large amounts of data.
- **Software engineering perspective:**
  - Well-designed library.
  - Gives us an idea of how to implement such a framework.
  - Exposes us to new design patterns.

Volley: Getting started

- Use Volley by creating a RequestQueue and passing to Request objects.
- RequestQueue manages worker threads for:
  - Network ops.
  - Read/Write from cache.
  - Parsing responses.
- Requests parse raw responses.
- Volley dispatches parsed response to UI thread.

Volley: Example 28 (Get an image from Imgur)

- Let’s write simple code to get a single image from Imgur.
- Get the URL from the user.
- Create a RequestQueue.
- Create a new ImageRequest (default request type).
- Add it to the queue.
- Override callbacks onResponse and onErrorResponse.
- How do we handle multiple requests using onResponse and onErrorResponse?

Volley: Handling multiple requests

- We could plant each callback *inline* as an anonymous class.
- This could result in messy code.
- We could create multiple responseListeners and errorListeners.
- The most common way is to use interfaces: https://stackoverflow.com/questions/45212853/handle-multiple-request-in-android-volley.
- Let’s move on.

Volley: NetworkImageView and ImageLoader

- Volley actually gives us two powerful tools to get images from URLs.
- NetworkImageView replaces the standard ImageView.
- ImageLoader sets up a cache to speed up loading multiple images into the NetworkImageView.
- Let’s modify our example to do this.

Volley: Example 29 (NetworkImageView and ImageLoader)

- Let’s modify our Imgur example.
- Get the URL from the user.
- We won’t have to handle the request ourselves now, since this operation is specialized.
- Put the image into the NetworkImageView using the ImageLoader.
- Lrucache holds references to a limited number of values. Each time a value is accessed, it is moved to the head of a queue.
- We have one more step here.

Volley: Writing your own request queue

- This is instructive for beyond just Volley.
- RequestQueue needs:
  - A cache on disk to cache requests.
  - A network connection to actually get stuff.
- **Important:** a RequestQueue must last the lifetime of your app, and you only want one of these floating around for a given data type!
- People often go the route of extending the Application class from Android to create a RequestQueue that outlives Activities and Fragments.
- This is discouraged.

Volley: Singleton pattern for request queues

- We can use the singleton pattern to create a class that is instantiated with the Application context, and outlives Activities.
- Recall what a singleton is:
  - A singleton class has only one instance.
  - It provides a global point of access.
- How do you implement this?

Digression: Singleton Pattern (Approach 1)

- We want only one instance, so you create one inside the class and declare it static.
- Declare the constructor private so nobody can instantiate the class.
- Provide a static method for obtaining the instance.
- Must also be thread-safe so as to not violate the singleton pattern.
  - Use synchronized keyword

Digression: Singleton Pattern (Approach 2)

- In Kotlin, any class created using the “object” keyword is automatically singleton.
- This doesn’t give you a convenient method to access the instance from anywhere, so we’ll usually roll our own solution.

Volley: Singleton pattern for request queues

- In our singleton class, we want both the RequestQueue and the ImageLoader.
- Put anything else here that we don’t want more than one instance of.
- Let’s see an example.

Volley: Example 30 (Singleton pattern for RequestQueue and ImageLoader)

- Let’s modify Example 29.
- We’ll implement a singleton class called ImageFetchSingleton.
- This class will hold both the request queue and the image loader.
- To get those in our app, we’ll have to access through the singleton.
- Remember the singleton pattern for when you want only one instance floating around.
- Our singleton class is in Java (just for fun).
  - Clearly, we can mix Java and Kotlin code.

Volley: Custom Requests

- We might also want to write custom requests if we’re getting a data source that is not a String, Image, or JSON.
- To do so, we need to extend the Request<T> class.
- We also need to implement the following abstract methods:
  - parseNetworkResponse()
  - deliverResponse()
- We won’t be going over this as it’s very tedious.
- Android developer docs have an example of writing a custom GsonRequest.

DownloadManager

Services: An Introduction

- Remember that Android has four major components:
  - Activities
  - Services
  - Content providers
  - Broadcast receivers
- This is a great time to understand services.
- What is a service?

Services: Intro

- They perform long-running operations.
- They don’t provide user interfaces.
- They can continue in the background even if the user switches to another app.
- There are 3 types:
  - Foreground services
  - Background services
  - Bound services

Broadcast Receivers: Intro

- This is also a good time to talk about Broadcast Receivers!
- Apps can send and receive broadcast messages triggered by certain events.
- This is a **publish-subscribe** pattern, typically. You publish your broadcast, and any subscriber who wants to can tap into it.
- How does all this fit into internet connectivity?

DownloadManager

- DownloadManager is a system service.
- Allows you to handle long-running HTTP downloads in the background.
- DownloadManager sends a broadcast when the download is finished.
- Any triggering application that uses this system service must implement a broadcast receiver.

DownloadManager

- DownloadManager handles retries, does not require process to be running.
- Requires that downloads be initiated from simple URLs.
- Shows users the download results via the Downloads app.
- Downloads to external storage without trouble.
- Can download only one item at a time (may delay your download).
- Will queue downloads for you.

DownloadManager: Service

- A few steps must be followed.
- First, add DownloadManager to OnCreate using getSystemService.
- Next, create a DownloadManager.Request. This can be used to set properties like:
  - Allow/disallow WiFi downloads.
  - Allow/disallow downloads over roaming.
  - The download destination.
- Then, you must push the request into a queue, and get its unique id.

DownloadManager: Broadcast Receiver

- To listen to DownloadManager, we need to create a BroadcastReceiver.
- We also need to register this receiver.
- This registers an **Intent Filter** for our app that effectively treats the incoming broadcast as an **intent**.
- Of course, intent filters are in general used to grab (or filter out) implicit intents sent to our app.

DownloadManager: Example 31

- Let’s modify Example 28.
- We’ll use DownloadManager to get the file from the internet.
- We’ll store it in the Downloads directory.

JobScheduler

JobScheduler: Introduction

- JobScheduler is the most common way to schedule background work.
- It can be used to schedule work like downloading data, updating network resources, etc.
- More importantly, this can be done while optimizing for memory, power, connectivity.
- Older alternatives are:
  - SyncAdapters.
  - AlarmManager.

JobScheduler: Introduction

- JobScheduler is considered **scalable**: good for both small tasks (clearing local cache) or larger tasks (sync your database with a server).
- It is typically used for tasks that are not time critical.
- Rather, you define conditions for when the job has to be done, and those decide when they job is run.
- Let’s see how to use JobScheduler.

JobScheduler: Setup

- Three steps:
  - Create a JobService to handle your task.
  - Add your JobService to the Android manifest.
  - Schedule your task using a JobInfo object.
    - You can define execution conditions when creating this object.
- As the names imply, JobService is a kind of Service.
- A little different from DownloadManager.

JobScheduler: JobService

- First, have your class extend JobService.
- Override onStartJob() and onStopJob().
  - onStartJob() is called when the job is started.
  - onStopJob() is called only when the job is cancelled prematurely.
- You must also call jobFinished() when everything is done to release resources. If you don’t, you could drain battery.
- It isn’t automatically multithreaded, so you’ll have to use multithreading/coroutines if you want that functionality.

JobScheduler: Example 32

- We’ll write a simple example to show a toast on a specific type of connection.
- The toast is our job.
- Let’s turn off wifi, start the job, then turn on wifi.
- Obviously, you should use background threads or coroutines to schedule more complex tasks that could block the UI thread.
- You can also use this API to connect to the internet and get stuff when required.

Internet Connectivity: Summary

- We saw the HTTPURLConnection way of fetching data.
- We saw how to use background threads and careful software engineering (static classes) to fetch data in a lifecycle-aware fashion inside a fragment.
- We saw how to use Volley to easily enable request queues in our app, and to fetch different kinds of data.
- We saw how to use DownloadManager, a system Service, to download a chunk of data and save it. We also defined a Broadcast Receiver to let us know when the download was done.
- Finally, we saw an example of using JobScheduler to schedule jobs that run when certain conditions are met.

Lab

- Implement background threads in a Fragment to get data from the Openweathermap API.
- Write a singleton class that encapsulates a RequestQueue.
- Use this class to get the JSON string in our Openweathermap API example.
- To do so, simply put the code into the NetworkUtils utility class.