Retrofit

Love Working With APIs
On Android

Marcus Pöhls & Norman Peitek

Retrofit: Love Working with APIs on Android

Take delight in building API clients on Android.

Marcus Pöhls

This book is for sale at http://leanpub.com/retrofit-love-working-with-apis-on-android

This version was published on 2019-02-17

Also By Marcus Pöhls

Contents

Introduction

Due to the popularity of the Retrofit blog post series[1] published in the Future Studio blog, we've decided write a book on Retrofit. We're delighted about the amazing popularity of the Retrofit series! **Thanks a lot for all of the positive feedback, comments and encouragement!** Your kind words go a long way and keep us motivated to publish more content on Retrofit.

This is the second edition of our Retrofit book. We've updated the content including code examples to address **Retrofit 2**. The first edition was completely geared towards Retrofit 1. If you're using Retrofit 1 and want to read the first edition of this book, please visit this book's extras on Leanpub to download the package with `PDF`, `mobi` and `epub` files.

Within this book, we keep the kind of techy style from the tutorials to make this book a great resource for every developer working with Retrofit.

What Topics Are Waiting for You?

You probably scanned the table of contents and know what to expect. Let me describe the chapters in short before we move on and dig deeper into Retrofit and its functionality.

Covered Topics:

- Introduction to Retrofit
- **Quick Start Guide** to jump right into Retrofit
- Create a **Sustainable Android REST Client**
- Extensive manipulation and customization of **requests**
- Comprehensive overview of **response converters and data mapping**
- Handling **Authentication** on Android (Basic, Token, OAuth, Hawk)
- Advanced File Handling, like **File Up- and Download**
- How to **handle errors** in your Android app
- Debug requests and response using **logging**
- App release preparation including ProGuard configuration

Covered Topics: Give Me the Details

The book starts out with an overview about what Retrofit is and how to prepare your Android project to use Retrofit. Further, we'll walk you through the setup on how to create a sustainable REST client basis. Additionally, we'll dive into the basics on how responses get mapped to Java objects and create

[1]https://futurestud.io/blog/retrofit-getting-started-and-android-client

the first client to perform a request against the GitHub API (learning by doing is an effective method :)).

Once we managed the jumpstart, we show you the details about Retrofit's requests: how to perform them synchronous and asynchronous and how to manipulate requests to your personal needs, like adding request parameters, path parameters, request payload and a lot more!

We'll also walk you through Retrofit responses, show you how to change the response converter and how to mock an API on Android itself.

Knowing the basics about Retrofit, we touch a common use case: authentication. Besides basic and token authentication, we'll explain how to use Retrofit for OAuth (including OAuth 2) and how to use OAuth's refresh token to get back a valid access token.

File handling can be kind of tricky, so let's take the road together! We guide you through file up- and downloads with Retrofit and show the actions required to send and receive different types of files like images and compressed packages (e.g. `ZIP`).

Last but not least: once you get your app (using Retrofit) out the door, you need to prepare the release for Google Play. This chapter digs into the correct configuration of ProGuard for Android projects integrating Retrofit and presents examplary rules to keep your app working correctly after distributing via Google Play.

Who Is This Book For?

In short: **this book is for you**. We believe that every developer reading this book will take away new ideas on how to optimize their Android apps in the sense of interacting with API's or webservices using Retrofit. You'll recognize goodies that are applicable for the projects you're working on.

This book is for Android developers who want to receive a substantial overview and reference book on Retrofit. You'll benefit from clearly recognizable code examples related to your daily work with Retrofit.

Rookie

If you're just starting out with Retrofit (or coming from another HTTP library like Android Asynchronous Http Client or even Volley) this book will show you all important parts on how to create sustainable REST clients on Android. The provided code snippets let you jumpstart and create your first successful API client within minutes!

Expert

If you've already worked with Retrofit, you'll profit from our extensive code snippets and apply the learnings to your existing code base. Additionally, the book illustrates various use cases for different functionalities and setups like authentication against different backends, request composition, file up- and downloads, etc.!

The Source Code

With the purchase of this book, you benefit from the sample project that await your download within the **extras** section on Leanpub. The sample project is an Android project based on gradle. You can directly touch and use classes that are only illustrated in excerpts within this book.

Check your Leanpub Library[2] and select **Retrofit: Love working with APIs on Android** to download the sample code base.

Retrofit Book for Version 1.9

This is the second edition of the Retrofit book. It's fully focussed on Retrofit 2, no hint or code snippet that points to Retrofit 1. If you're interested in the first version of the book that based on Retrofit 1, please visit this book's **extras** page on Leanpub. As a reader of this book, the first version is — of course — also available for you!

Now, let's jump right in and get started with Retrofit!

[2] https://leanpub.com/user_dashboard/library

Chapter 1 — Quick Start & Create a Sustainable REST Client

Within this chapter we're going through the basics of Retrofit, give a brief overview to jump-start with Retrofit in your project and create a REST client foundation that we'll enhance and apply within multiple chapters of this book.

Precisely, we start with a short overview of the Retrofit project and why you should make use of it. Afterwards, we show you how to prepare your Android project to utilize Retrofit. To get you hooked on the Retrofit train, we create a sustainable Android REST client including the assignment of a JSON response converter.

To clarify our usage of the term **Retrofit** throughout this book: we always refer to Retrofit 2. The content completely gears toward Retrofit 2 and we just keep it short by using Retrofit.

Remember, as already mentioned in the Introduction chapter: if you're interested in the previous book version on Retrofit 1.9, we've added the book's PDF, mobi and ePub files within the extras of this book on Leanpub[3].

Retrofit Overview

Retrofit is a »type-safe REST client for Android and Java«[4].

Retrofit abstracts your REST API into Java interfaces. You'll use annotations to describe your individual API endpoints and their HTTP requests. Support for URL parameter replacement (like query and path parameter) is integrated by default, as well as functionality for form-urlencoded and multipart requests. You can of course execute requests using the HTTP methods `GET`, `POST`, `PUT`, `PATCH`, `DELETE`. Anyway, this is literally just the tip of the iceberg and there's a lot more to discover behind Retrofit's scenes.

Why Use Retrofit?

Working with APIs can cause a lot headaches due to various scenarios and edge cases. Problems already arise when just giving the low-level networking some thoughts. Handling connections and interruptions, caching, request retries and SSL handshakes. Also, the worry about worker threads or runnables or even lovely AsyncTasks definitely causes a lot pain in the a**! Think about the pain and effort you need to put into your own implementation.

[3]https://leanpub.com/user_dashboard/library
[4]http://square.github.io/retrofit/

You'll save yourself a lot of time and anger when leveraging a well thought-out and tested library like Retrofit. If you scrolled through the outline of this book, you've already an impression about the functionality and flexibility that comes with Retrofit. Benefit from the library and put your work to pieces where your attention is actually needed!

Retrofit vs. OkHttp

Downright this book, we'll make use of OkHttp and show you how to achieve a solution leveraging the functionality of OkHttp. And the reason is simple: OkHttp is a pure HTTP/SPDY client responsable for any low-level network operation, caching, request and response manipulation, and many more. In contrast, Retrofit is a high-level REST abstraction build on top of OkHttp. Retrofit 2 is strongly coupled with OkHttp and makes intensive use of it. That's the reason why we sometimes need to borrow OkHttp's classes to accomplish the solution.

Quick Start: Add Retrofit to Your Project

Now let's get our hands dirty and back to the keyboard. If you already created your Android project, just go ahead and start from the next paragraph („Internet Permission in Android's Manifest“). If not, create a new Android project in your favorite IDE. We use Android Studio[5] and prefer the Gradle build system, but you surely can use Maven as well.

Internet Permission in Android's Manifest

We use Retrofit to perform HTTP requests against an API running on a server somewhere in the Internet. Executing those requests from an Android application requires the `Internet` permission to open network sockets. You need to define the permission within the `AndroidManifest.xml` file. If you didn't set the Internet permission yet, please add the following line within your manifest definition:

```xml
<uses-permission android:name="android.permission.INTERNET" />
```

A common practice is, to add your app permissions as the first elements in your manifest file. The following snippet is an excerpt from the sample project you can find within the extras of this book on Leanpub.

[5] https://developer.android.com/sdk/index.html

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="android.retrofitbook.futurestud.io.fsretrofitbook"
          xmlns:android="http://schemas.android.com/apk/res/android">

    <uses-permission android:name="android.permission.INTERNET"/>

    …
```

Define Dependency: Gradle or Maven

Now let's define Retrofit as a dependency for your project. Depending on your used build system, define Refrofit in your `build.gradle` or `pom.xml` file. When running the command to build your code, the build system will download and provide the library for your project.

At the time of this book being written, the latest Retrofit version is `2.0.0`. There has been a lot activity in the Retrofit repository on GitHub during the last weeks to finish the second major release. We'll keep this book update to future versions of Retrofit.

Ok, now let's add Retrofit as a dependency to our project:

build.gradle

```
dependencies {
    // Retrofit & OkHttp
    implementation 'com.squareup.retrofit2:retrofit:2.5.0'
}
```

pom.xml

```
<dependency>
    <groupId>com.squareup.retrofit2</groupId>
    <artifactId>retrofit</artifactId>
    <version>2.5.0</version>
</dependency>
```

Sync your Gradle or Maven project to import the required packages for Retrofit. Retrofit 2 doesn't ship with an integrated response converter anymore. We need to manually add the desired response converters as a dependency to our project. The following section will show you how to add Gson for JSON response mapping to your project.

JSON Response Mapping

As already mentioned, by default Retrofit 2 doesn't come with any response converter integrated. To add Google's Gson for JSON to Java object mapping, we need to add another Gradle or Maven dependency to our project. Update your `build.gradle` or `pom.xml` file appropriately to import Retrofit's Gson converter besides Retrofit.

build.gradle

```
dependencies {
    // Retrofit & Gson
    implementation 'com.squareup.retrofit2:retrofit:2.5.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
}
```

pom.xml

```
<dependency>
    <groupId>com.squareup.retrofit2</groupId>
    <artifactId>retrofit</artifactId>
    <version>2.5.0</version>
</dependency>
<dependency>
    <groupId>com.squareup.retrofit2</groupId>
    <artifactId>converter-gson</artifactId>
    <version>2.5.0</version>
</dependency>
```

Once again, synchronize your project to import the Gson converter for Retrofit. In the following, you'll learn how to define a service interface to map a given API endpoint.

Declare an API Interface

This part of the quick start guide is intended to get you kickstarted and how applicable Retrofit is for your Android app. Let's directly jump in and use a code example to make things approachable:

```
public interface GitHubService {
    @GET("users/{user}/repos")
    Call<List<GitHubRepo>> reposForUser(@Path("user") String username);
}
```

The snippet above defines a Java interface called `GitHubService` having only one method `reposForUser(…)`. The method and its parameters have Retrofit annotations which describe the behavior of this method.

The `@GET()` annotation explicitly defines that a GET request will be executed once the method gets called. Further, the `@GET()` definition takes a string parameter representing the endpoint url of your API. Additionally, the endpoint url can be defined with placeholders which get substituted by path parameters.

The method signature contains a `@Path()` annotation for the `user` parameter. This `@Path` annotation maps the provided parameter value during the method call to the path within the request url. The declared `{user}` part within the endpoint url will be replaced by the provided value of `username`. We'll catch up path parameters in more detail in a later chapter.

The snippet above only describes how to define your API on the client side. That's fine for now. We'll have a look at the practical example in a second. There, we're going to execute the actual API request.

Sustainable Android REST Client

Retrofit offers a wide range of functionalities and there are a lot of possible configurations. A lot of larger applications will require some specific setup, for example for OAuth authentication. In order to achieve a clean and stable project, we'll introduce you our idea of a sustainable Android client: the `ServiceGenerator`.

The `ServiceGenerator`

As you know from our the quick start above, the `Retrofit` object and its builder are the heart of all requests. Here you configure and prepare your requests, responses, authentication, logging and error handling. Unfortunately, we've seen too many developers just copy-and-pasting these parts instead of separating into one clean class. The `ServiceGenerator` will give you our solution, which is based on Bart Kiers' idea[6].

Let's start with the simple code. In its current state, it only defines one method to create a basic REST client for a given class/interface, which returns a service class from the interface.

[6]https://github.com/bkiers/retrofit-oauth/tree/master/src/main/java/nl/bigo/retrofitoauth

```java
public class ServiceGenerator {

    private static final String BASE_URL = "https://api.github.com/";

    private static Retrofit.Builder builder =
            new Retrofit.Builder()
                    .baseUrl(BASE_URL)
                    .addConverterFactory(GsonConverterFactory.create());

    private static Retrofit retrofit = builder.build();

    private static OkHttpClient.Builder httpClient =
            new OkHttpClient.Builder();

    public static <S> S createService(
        Class<S> serviceClass) {
        return retrofit.create(serviceClass);
    }
}
```

The `ServiceGenerator` class uses Retrofit's `Retrofit` builder to create a new REST client with the given API base url (`BASE_URL`). For example, GitHub's API base url is located at `https://api.github.com/` and you must update the provided example url with your own url to call your API instead of GitHub's.

The `createService` method takes a `serviceClass`, which is the annotated interface for API requests, as a parameter and creates a usable client from it. On the resulting client you'll be able to execute your network requests.

Why Is Everything Declared Static Within the `ServiceGenerator`?

You might wonder why we use static fields and methods within the `ServiceGenerator` class. Actually, it has one simple reason: we want to use the same objects (`OkHttpClient`, `Retrofit`, ...) throughout the app to just open one socket connection that handles all the request and responses including caching and many more. It's common practice to just use one `OkHttpClient` instance to reuse open socket connections. That means, we either need to inject the `OkHttpClient` to this class via dependency injection or use a static field. As you can see, we chose to use the static field. And because we use the `OkHttpClient` throughout this class, we need to make all fields and methods static.

Additionally to speeding things up, we can save a little bit of valuable memory on mobile devices when we don't have to recreate the same objects over and over again.

Using the `ServiceGenerator`

Have you seen Retrofit request code? It would look like this:

```
String API_BASE_URL = "https://api.github.com/";

OkHttpClient.Builder httpClient = new OkHttpClient.Builder();

Retrofit.Builder builder =
    new Retrofit.Builder()
            .baseUrl(API_BASE_URL)
            .addConverterFactory(
                GsonConverterFactory.create()
            );

Retrofit retrofit =
    builder
        .client(
            httpClient.build()
        )
        .build();

GitHubClient client = retrofit.create(GitHubClient.class);
```

For one request, this looks fine. But if you have dozens of network requests throughout your app, it'll be a nightmare to manage. With our `ServiceGenerator`, you only need a single line:

```
GitHubClient client = ServiceGenerator.createService(GitHubClient.class);
```

All of the preparations were moved into our `ServiceGenerator`.

Unfortunately, in most cases the `ServiceGenerator` cannot stay this simple. Thus, the code from above only gives you a starting point. You'll need to adapt it to your needs just like we'll do in other chapters. Nevertheless, in the next two sections we'll explore a few possible changes.

Preparing Logging

One of the most common wishes for developers is to know what kind of data is actually being sent and received by Retrofit. We have an entire chapter dedicated on logging with Retrofit, where you can learn more.

Logging with Retrofit 2 is done by an interceptor called `HttpLoggingInterceptor`. You'll need to add an instance of this interceptor to the `OkHttpClient`. For example, you could solve it the following way:

```java
public class ServiceGenerator {

    private static final String BASE_URL = "https://api.github.com/";

    private static Retrofit.Builder builder =
            new Retrofit.Builder()
                    .baseUrl(BASE_URL)
                    .addConverterFactory(GsonConverterFactory.create());

    private static Retrofit retrofit = builder.build();

    private static HttpLoggingInterceptor logging =
            new HttpLoggingInterceptor()
                    .setLevel(HttpLoggingInterceptor.Level.BODY);

    private static OkHttpClient.Builder httpClient =
            new OkHttpClient.Builder();

    public static <S> S createService(
        Class<S> serviceClass) {
        if (!httpClient.interceptors().contains(logging)) {
            httpClient.addInterceptor(logging);
            builder.client(httpClient.build());
            retrofit = builder.build();
        }

        return retrofit.create(serviceClass);
    }
}
```

There are a few things you've to be aware of. Firstly, make sure you're not accidentally adding the interceptor multiple times! We check with `httpClient.interceptors().contains(logging)` if the logging interceptor is already present. Secondly, make sure to not build the `retrofit` object on every `createService` call. Otherwise the entire purpose of the `ServiceGenerator` is defeated.

Prepare Authentication

The requirements for authentication are a little bit different. You can learn more in our sections on Basic Authentication, Token Authentication, OAuth, or even Hawk Authentication. While the details are a little different for each authentication implementation, you probably will have to change the `ServiceGenerator`. One of the changes is that you need to pass additional parameters to `createService` to create a client.

Let's look at an example for Hawk authentication:

```java
public class ServiceGenerator {

    private static final String BASE_URL = "https://api.github.com/";

    private static Retrofit.Builder builder =
            new Retrofit.Builder()
                    .baseUrl(BASE_URL)
                    .addConverterFactory(GsonConverterFactory.create());

    private static Retrofit retrofit = builder.build();

    private static HttpLoggingInterceptor logging =
            new HttpLoggingInterceptor()
                    .setLevel(HttpLoggingInterceptor.Level.BODY);

    private static OkHttpClient.Builder httpClient =
            new OkHttpClient.Builder();

    public static <S> S createService(
            Class<S> serviceClass, final HawkCredentials credentials) {
        if (credentials != null) {
            HawkAuthenticationInterceptor interceptor =
                    new HawkAuthenticationInterceptor(credentials);

            if (!httpClient.interceptors().contains(interceptor)) {
                httpClient.addInterceptor(interceptor);

                builder.client(httpClient.build());
                retrofit = builder.build();
            }
        }

        return retrofit.create(serviceClass);
    }
}
```

Our `createService` now has a second parameter for the `HawkCredentials`. If you pass a non-null value, it'll create the necessary Hawk authentication interceptor and add it to the Retrofit client. We also need to rebuild Retrofit to apply our changes to the next request.

One more heads-up, you probably will see slightly different versions of the `ServiceGenerator` in future chapters. Don't be confused! We recommend that you also keep your `ServiceGenerator` slim and specialized for the use case!

In this section you've learned why centralizing your Retrofit client generation makes sense and is recommended. You've seen one approach how you can implement it with the `ServiceGenerator` class. Nevertheless, you probably will have to adjust it to your purposes.

Retrofit in Use

Ok, let's use an example and define a REST client to request data from GitHub. First, we have to create an interface and define the required methods.

GitHub Client

We'll use the already defined `GitHubService` interface from above that has only one method to request the repositories for a given user. Remember that we're replacing the path parameter placeholder (`{user}`) with the actual value of `user` when calling this method.

```java
public interface GitHubService {
    @GET("users/{user}/repos")
    Call<List<GitHubRepo>> reposForUser(@Path("user") String user);
}
```

The interface above defines a `GitHubRepo` class. This class includes the required object properties to map the response data. For illustration purposes, we just define the repository's `id` and `name` properties. GitHub's API response for this endpoint has a lot more data, but is sufficient to show you how things work.

```java
public class GitHubRepo {
    private int id;
    private String name;

    public GitHubRepo() {}

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

With regard to the previously mentioned JSON mapping: the created `GitHubService` defines a method named `reposForUser` with the return type `List<GitHubRepo>`. The `List<GitHubRepo>` is wrapped into a `Call`. We'll cover the `Call` within the next chapter. For now it's ok to know that you need wrap your response into a call object.

If an appropriate response converter is present, Retrofit ensures that the server's JSON response gets mapped correctly to Java objects (assuming that the JSON data matches the given Java class).

API Example Request

The snippet below illustrates the usage of the `ServiceGenerator` to instantiate your client, specifically the `GitHubService`, and the method call to get a user's repositories via that client. This snippet is a modified version of provided Retrofit github-client example[7].

```
@Override
protected void onCreate(final Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Change base url to GitHub API
    ServiceGenerator.changeApiBaseUrl("https://api.github.com/");

    // Create a simple REST adapter which points to GitHub's API
    GitHubService service =
            ServiceGenerator.createService(GitHubService.class);

    // Fetch and print a list of repositories for user "fs-opensource"
    Call<List<GitHubRepo>> call = service.reposForUser("fs-opensource");
    call.enqueue(new Callback<List<GitHubRepo>>() {
        @Override
        public void onResponse(Call<List<GitHubRepo>> call,
                                       Response<List<GitHubRepo>> response) {
            if (response.isSuccess()) {
                for (GitHubRepo repo : response.body()) {
                        Log.d("Repo: ",
                                repo.getName() + " (ID: " + repo.getId() + ")");
                }
            } else {
                    Log.e("Request failed: ",
                            "Cannot request GitHub repositories");
            }
        }
```

[7]https://github.com/square/retrofit/blob/master/samples/src/main/java/com/example/retrofit/SimpleService.java

```

        @Override
        public void onFailure(Call<List<GitHubRepo>> call, Throwable t) {
            Log.e("Error fetching repos", t.getMessage());
        }
    });
}
```

You've got a first impression of Retrofit and know how to define an interface which represents your API endpoints on client side. Besides that, you know how to create the API client with the help of Retrofit's `Retrofit` class and how to create a generic `ServiceGenerator` for static service creation.

We'll update the `ServiceGenerator` in the following chapters within this book with examples for authentication.

The next chapter shows you how to define and manipulate requests with Retrofit.

Chapter Summary

You've mastered your first steps to become a proficient developer using Retrofit. Within this chapter, you got an overview on Retrofit and we walked through a practice related example. You should have learned

- [x] What is Retrofit
- [x] Why use Retrofit
- [x] How to define a service interface representing an API endpoint
- [x] How to create an instance of the service interface
- [x] How to execute an API request

The next chapter is all about requests. We'll guide you through the setup of Retrofit to receive and send data with your request.

Outro

Our goal is to truly help you getting started and ultimately master Retrofit. We hope you learned many new things throughout this book. We want you to save time while learning the basics and details about Retrofit. The existing Retrofit documentation lacks various information and this book should help you to gain extensive in-depth knowledge without loosing time searching StackOverflow for correct answers.

We're currently planning new chapters and sections on Retrofit that will be added within the upcoming months. We feel the need for an extra chapter about reactive extensions on Android using RxAndroid/RxJava. That is the current high priority item on our idea list. Besides that, we plan to add content on the testing chapter.

Nonetheless, we'll update the content of this book to later Retrofit versions as new releases become available. However, it will take some time to update the code for breaking changes introduced to Retrofit. Of course, we'll let you about any book updates.

As a reader of this book, we'll always reward your loyalty by publishing exclusive content and any future update will —of course— be free for you!

For us it's really important to exceed our reader's expectations. In all our products and guides we aim for a high quality. If you feel like a section or chapter in this book wasn't clear or extensive enough, please let us know at info@futurestud.io[8]. We always love hearing back from you, so if you have anything to say, don't hesitate to shoot us an email. We welcome any feedback, critic, suggestions for new topics or whatever is currently on your Retrofit mind!

Don't forget, we're publishing new tutorials every Wednesday and Thursday, mainly about Android and Node.js within the Future Studio University. Feel free to visit our homepage[9] and the University[10] :)

Finally, we're also releasing YouTube videos in 4k resolution every Monday and Friday. We just started publishing a 20-part series on Retrofit. Feel free to check out our Future Studio YouTube channel[11].

Thanks a lot for reading this book! We truly appreciate your interest and hope you learned a lot from reading this book! <3

— Marcus & Norman

[8] mailto:info@futurestud.io
[9] https://futurestud.io
[10] https://futurestud.io/blog
[11] https://www.youtube.com/c/FutureStudio