SIMPLIFIED CODING

Dagger 2 Android Example using Retrofit : Injecting Dependency

December 2, 2017 by Belal Khan — 8 Comments

Struggling with Dagger 2? Don't know how to implement it in your Android Project? Don't worry, here am I with this Dagger 2 Android Example.

So, today in this Dagger 2 Android Example we will modify one of our previous project to learn the Dependency Injection Architecture.

The post is for beginners, who just want to get into the design patterns.

In this tutorial, we will work on Retrofit Android Example project that we created in one of the earlier posts. So it is highly recommended that you should go through that tutorial first and get the source code of that project.

Contents [hide]

Dependency Injection is a design pattern, or you can say a concept of Object Oriented Programming, where we don't create an object of another class inside a class using the new keyword (for Java). Instead, we supply the needed object from outside.

In the previous post, **Android Dependency Injection** we learned the conceptual thing about Dependency Injection in detail. So if you are entirely unaware of this stuff, you should go through the previous tutorial first.

What is Dagger 2?

> " Dagger is a fully static, compile-time dependency injection framework for both Java and Android. It is an adaptation of an earlier version created by Square and now maintained by Google.

I guess Dagger 2 is the most popular dependency injection framework currently available. And it is now maintained by Google, so you should learn it.

Working with Dagger 2

Before moving ahead in our Android Project, let's understand first that how we work with Dagger 2.

So we have three major things in Dagger.

#1 Dependency Provider

Dependencies are the objects that we need to instantiate inside a class. And we learned before that we cannot instantiate a class inside a class. Instead, we need to supply it from outside. So the person who will provide us the objects that are called dependencies is called **Dependency Provider**.

Confused? Don't worry it is not a real person :P. It is a regular class that will provide the dependency.

And in **dagger2** the class that you want to make a Dependency Provider, you need to annotate it with the **@Module** annotation.

And inside the class, you will create methods that will provide the objects (or dependencies). With dagger2 we need to annotate these methods with the **@Provides** annotation.

So remember the annotations **@Module and @Provides.**

declaration with **@Inject.**

#3 Component

Finally, we need some connection between our dependency provider and dependency consumer.
For this, we will create an interface by annotating it with **@Component**. And rest of the thing will be done by Dagger.

Is everything confusing? Don't worry you will get things cleared after implementing in the Android Project so keep reading.

Dagger 2 Android Example

Now let's refactor the application that we created in the earlier Retrofit Example Tutorial.

This time we will not create a new Project, so just go to the following link and download the Source Code, then open it in your Android Studio. And then you are ready to move ahead.

Retrofit Android Example – Fetching JSON

Adding Dagger2 to Project

The first step is obvious that we need to add the Dagger2 to our project.

- So in the project that you opened go to **app level build.gradle** and inside the dependencies block add the lines shown below.

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:26.+'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'

    compile 'com.squareup.retrofit2:retrofit:2.3.0'
    compile 'com.squareup.retrofit2:converter-gson:2.2.0'


```

- Sync the project with gradle.

Creating Modules

For this project, we need two modules. The first one is the **App Module**, and the Next one is **API Module.**

App Module

This module will provide the Context. You already know that we need Context everywhere, and in Retrofit as well we need the context object. And as the DI rule says we need an outsider to supply the objects, so here we will create this module that will give us the Context.

- Create a new class named **AppModule** and write the following code.

AppModule.java Java

```java
package net.simplifiedlearning.retrofitexample;

import android.app.Application;

import javax.inject.Singleton;

import dagger.Module;
import dagger.Provides;

/**
 * Created by Belal on 12/2/2017.
 */


@Module
class AppModule {
    private Application mApplication;

    AppModule(Application mApplication) {
        this.mApplication = mApplication;
    }

    @Provides
```

SIMPLIFIED CODING

- The above code is straightforward, and we just use a new thing **@Singleton**. In dagger, we have this annotation, when we want a Single object. I guess you already know about Singleton. With the dagger, you just need to annotate @Singleton when you want a single instance only.

API Module

We need many objects in this Module. You might already know that for Retrofit fit we need a bunch of things. We need **Cache, Gson, OkHttpClient** and the Retrofit itself. So we will define the providers for these objects here in this module.

- So, create the class named **AppModule** and write the following code.

ApiModule.java Java

```java
package net.simplifiedlearning.retrofitexample;

import android.app.Application;

import com.google.gson.FieldNamingPolicy;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

import javax.inject.Singleton;

import dagger.Module;
import dagger.Provides;
import okhttp3.Cache;
import okhttp3.OkHttpClient;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

/**
 * Created by Belal on 12/2/2017.
 */

@Module
class ApiModule {

    String mBaseUrl;

    ApiModule(String mBaseUrl) {
        this.mBaseUrl = mBaseUrl;
    }
```

```java
        Cache cache = new Cache(application.getCacheDir(), cacheSize);
        return cache;
    }

    @Provides
    @Singleton
    Gson provideGson() {
        GsonBuilder gsonBuilder = new GsonBuilder();
        gsonBuilder.setFieldNamingPolicy(FieldNamingPolicy.LOWER_CASE_WITH_UNDERSCORES);
        return gsonBuilder.create();
    }

    @Provides
    @Singleton
    OkHttpClient provideOkhttpClient(Cache cache) {
        OkHttpClient.Builder client = new OkHttpClient.Builder();
        client.cache(cache);
        return client.build();
    }

    @Provides
    @Singleton
    Retrofit provideRetrofit(Gson gson, OkHttpClient okHttpClient) {
        return new Retrofit.Builder()
                .addConverterFactory(GsonConverterFactory.create(gson))
                .baseUrl(mBaseUrl)
                .client(okHttpClient)
                .build();
    }
}
```

We have done with the Modules. Now let’s define the Component and then we will inject the objects.

Building Component

- Now, we will create the Component.

ApiComponent.java Java

```java
package net.simplifiedlearning.retrofitexample;

import javax.inject.Singleton;

import dagger.Component;
```

SIMPLIFIED CODING

```
@Component(modules = {AppModule.class, ApiModule.class})
public interface ApiComponent {
    void inject(MainActivity activity);
}
```

- So you can see we will inject in the **MainActivity.** We also define all the modules using the @Component annotation as you can see in the code.

Before moving ahead Rebuild your project.

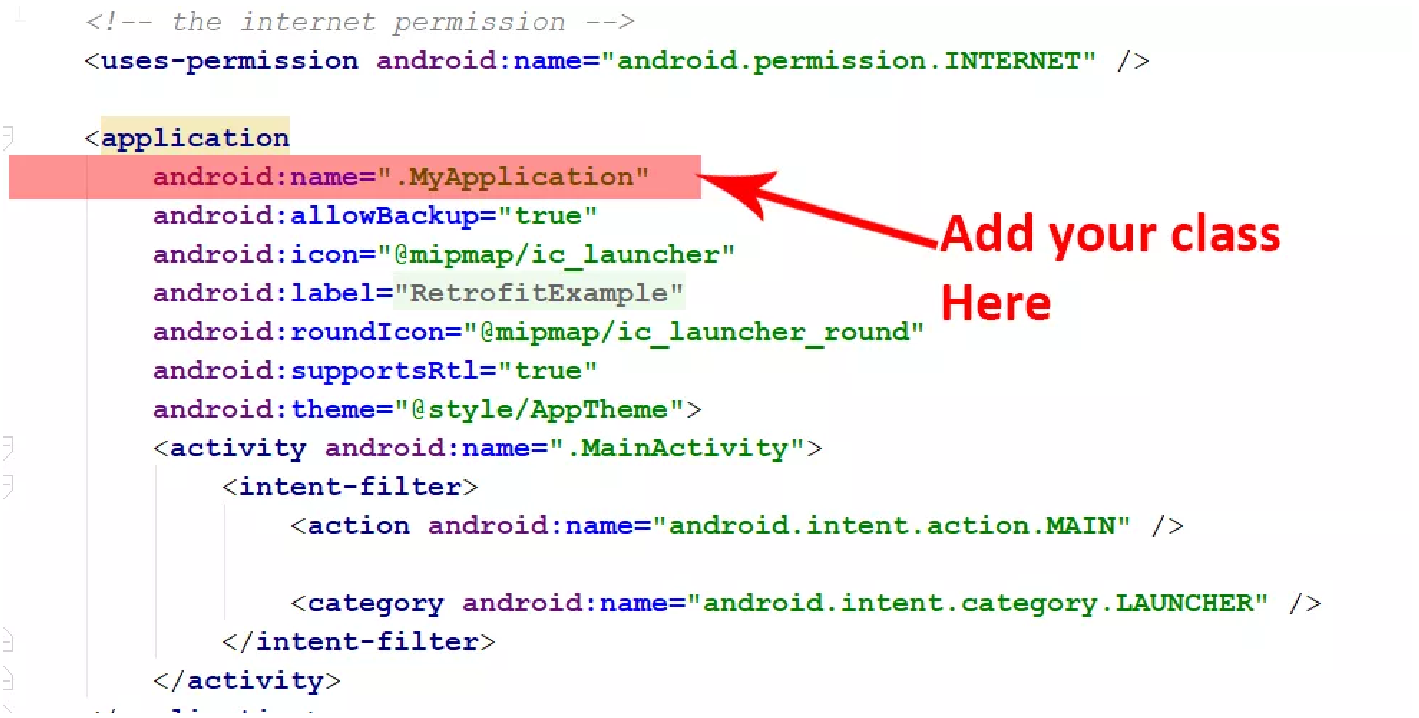
- Now create a class named **MyApplication**. In this class we will build the ApiComponent.

MyApplication.java Java

```java
package net.simplifiedlearning.retrofitexample;

import android.app.Application;

/**
 * Created by Belal on 12/2/2017.
 */

public class MyApplication extends Application {

    private ApiComponent mApiComponent;

    @Override
    public void onCreate() {
        super.onCreate();

        mApiComponent = DaggerApiComponent.builder()
                .appModule(new AppModule(this))
                .apiModule(new ApiModule("https://simplifiedcoding.net/demos/"))
                .build();
    }

    public ApiComponent getNetComponent() {
        return mApiComponent;
    }
}
```

- **Note that we are using the same project, with no additional changes. So make sure you are working, on the same project. The URL we are using in the above code to fetch JSON is working and you can**

SIMPLIFIED CODING



```
<!-- the internet permission -->
<uses-permission android:name="android.permission.INTERNET" />

<application
    android:name=".MyApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="RetrofitExample"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

Injecting Dependency with Dagger 2

Now finally, we can inject the dependency.

- Come inside **MainActivity.java** and modify it as below.

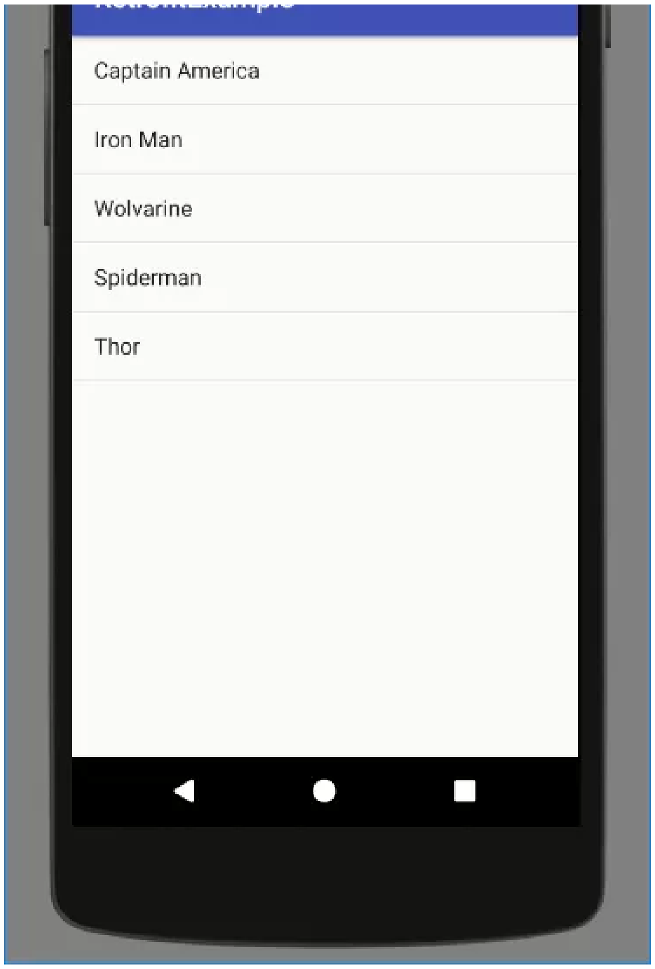
MainActivity.java Java

```java
package net.simplifiedlearning.retrofitexample;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.util.List;

import javax.inject.Inject;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;

```

SIMPLIFIED CODING

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

        ((MyApplication) getApplication()).getNetComponent().inject(this);


        listView = (ListView) findViewById(R.id.listViewHeroes);

        getHeroes();
    }

    private void getHeroes() {
        Api api = retrofit.create(Api.class);
        Call<List<Hero>> call = api.getHeroes();

        call.enqueue(new Callback<List<Hero>>() {
            @Override
            public void onResponse(Call<List<Hero>> call, Response<List<Hero>> response) {
                List<Hero> heroList = response.body();
                String[] heroes = new String[heroList.size()];

                for (int i = 0; i < heroList.size(); i++) {
                    heroes[i] = heroList.get(i).getName();
                }

                listView.setAdapter(new ArrayAdapter<String>(getApplicationContext(), android.R.lay
            }

            @Override
            public void onFailure(Call<List<Hero>> call, Throwable t) {
                Toast.makeText(getApplicationContext(), t.getMessage(), Toast.LENGTH_SHORT).show();
            }
        });
    }

}
```

- Now you can try running your application.



Dagger 2 Android Example

- So, it is working fine.

You might be thinking **"WHAT THE HELL? I did all these big changes to do nothing"**. 😛 But trust me it makes life easier when working on some big projects.

What we learned in this post is following an architecture. You will get the usefulness of this thing when I post about some Advanced Android App Design Pattern like MVP or MVVM.

Dagger 2 Android Example Source Code

So, guys, I hope you understand the basic idea that how we work with Dagger 2. But still, if you are having any trouble following the codes you can get my source code from GitHub.
To get the link, you need to unlock it using one of the SHARE buttons.

So that's all for this Dagger 2 Android Example friends. I hope you found it useful if you did, then please SHARE this post with your friends.
If you have any question regarding this Dagger 2 Android Example, then leave your comments, and I will try to help you. Thank You 🙂

Related

Android Dependency Injection : What it is and How to do it?
December 1, 2017
In "Android Advance"

Retrofit Android Example – Fetching JSON from URL
November 3, 2015
In "Android Advance"

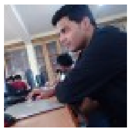
Android Firebase Tutorial – User Registration with Authentication
July 15, 2016
In "Android Advance"

Some more tutorials for you

1. **Login With Facebook Android Studio using Facebook SDK 4**
2. **WordPress to Android App using WP REST API Tutorial**

SIMPLIFIED CODING

Filed Under: Android Advance, Android Application Development



About Belal Khan

I am Belal Khan, I am currently pursuing my MCA. In this blog I write tutorials and articles related to coding, app development, android etc.

Comments

Pritish Sawant says

December 3, 2017 at 6:01 am

I am getting error in MyApplication class. There is nothing called DaggerApiComponent.

Reply

your project as Dagger generates it on compile time. So go to build -> rebuild the project then you will find the DaggerApiComponent.

Reply

Uttam Kumar says
December 9, 2017 at 3:36 am

Really this is awesome article for dragger 2. Thanks @Belal sir

Reply

buya says
December 9, 2017 at 11:16 am

amazing explanation man hats off

Reply

Pratik Gondil says
January 2, 2018 at 11:16 am

Your tutorial is awesome.thank u so much for this please share the next tutorial for MVP and MVVM architecture.i am eagerly waiting for that.

Reply

Muhamad Arief says

SIMPLIFIED CODING

Anon says
January 23, 2018 at 5:51 pm

.appModule(new AppModule(this))
.apiModule(new ApiModule("https://simplifiedcoding.net/demos/"))
Both module are deprecated, any suggestions?

Reply

akash says
April 5, 2018 at 2:37 am

Thank you very much for your post on android . keep going ...

Reply

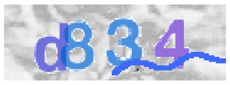
Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

SIMPLIFIED CODING



CAPTCHA Code
*

POST COMMENT

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.

SEARCH

Search this website ...

SIMPLIFIED CODING

DOWNLOAD OUR ANDROID APP

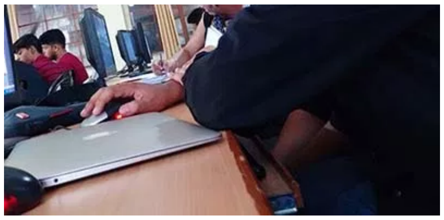
GET IT ON Google Play

Simplified Coding

YouTube 22K

ABOUT ME

SIMPLIFIED CODING



Hello I am Belal Khan, founder and owner of Simplified Coding. I am currently pursuing MCA from St. Xavier's College, Ranchi. I love to share my knowledge over Internet.

CONNECT WITH ME

Belal Khan

G+ Follow

1,495 followers

Follow @codesimplified

Simplified Coding

Like Page

Be the first of your friends to like this

Simplified Coding

G+ Follow

POPULAR TUTORIALS

Android JSON Parsing – Retrieve From MySQL Database

Android Login and Registration Tutorial with PHP MySQL

SIMPLIFIED CODING

Android TabLayout Example using ViewPager and Fragments

Android Upload Image to Server Using PHP MySQL

Retrieve Data From MySQL Database in Android using Volley

Firebase Cloud Messaging Tutorial for Android

Android Volley Tutorial – User Registration and Login

Android MySQL Tutorial to Perform Basic CRUD Operation

SIMPLIFIED CODING

SIMPLIFIED CODING

tutorials related to programming and application development. You can get various nice and simplified tutorials related to programming, app development, graphics designing and animation. We are trying to make these things simplified and entertaining. We are writing text tutorial and creating video and visual tutorials as well. You can check about the admin of the blog here and check out our sitemap

Privacy Policy

Disclaimer

About

Contact Us

Write for Us

CATEGORIES

Android Advance Android Application Development Android Beginners Android Intermediate Ionic Framework Tutorial JavaScript Kotlin Android Others PHP Advance PHP Tutorial React Native