

重拾丢却的梦 Lv2

2019年06月28日 阅读 5248

[关注](#)

Retrofit使用详解-注解介绍

关于Retrofit的讲解，我将写下面系列篇文章进行总结：

- Get和Post请求的基本使用
- 上传图片 
- 下载文件
- 添加拦截器
- ...(和RxJava联合使用、RxJava+Retrofit+OkHttp简单封装、统一错误处理)

讲解之前我必须先安利个网站[模客](#)，在学习Retrofit之前，各种网络请求如果想学习的话就必须有接口才能访问，Get请求还好说，但遇到Post请求，自己发送的是啥也不清楚，苦于自己不会后端写接口，所以之前对Retrofit的学习要么是各种找现成的API，要么就只是照猫画虎的敲了下并没有实践，有了这个网站你就可以自己写接口自己访问来测测看Retrofit的运用到底对不对。

具体该网站的使用见[模客的使用](#)这篇文章，下面我们来学习Retrofit的使用吧

1. Retrofit的基本使用

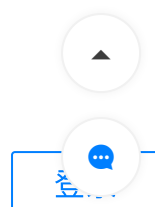
1.1 添加依赖

[复制代码](#)

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

第二个依赖主要是对Json数据进行解析的。

1.2 实现步骤

[探索掘金](#)

- 创建Retrofit实例
- 创建网络请求接口的实例
- 发送网络请求

和[官网](#)一样，可以看下官网介绍的代码，就是上面所写的步骤

[复制代码](#)

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

[复制代码](#)

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("https://api.github.com/")  
    .build();
```

[复制代码](#)

```
GitHubService service = retrofit.create(GitHubService.class);
```

[复制代码](#)

```
Call<List<Repo>> repos = service.listRepos("octocat");
```

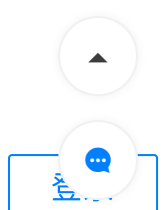
1.3 具体实现

下面我将介绍个具体的例子，让没用过Retrofit的先熟悉下Retrofit如何使用

1.3.1 创建网络请求的接口

[复制代码](#)

```
public interface GetApi {  
  
    /**  
     * 获取用户信息  
     * @return  
     * @Query 注解  
     */  
    @GET("getUserInfo")  
    Call<UserInfo> getUserInfo(@Query("id") String userId);  
  
}
```



UserInfo是我们得到返回数据的数据Bean类

1.3.2 创建Retrofit实例

[复制代码](#)

```
private Retrofit retrofit;
retrofit = new Retrofit.Builder()
    .baseUrl("http://mock-api.com/2vKVbXK8.mock/")
    .addConverterFactory(GsonConverterFactory.create()) //返回的Json数据进行解析
    .build();
```

baseUrl:

这里的baseUrl是自己访问的Url的基类地址，加上刚才@GET("getUserInfo")中的 **getUserInfo** 才是我们真正要访问的地址，因为使用了@Query("id")，所以最终的访问地址为

http://mock-api.com/2vKVbXK8.mock/getUserInfo?id=userid，此处的userid为自己传入的参数。

注意: baseUrl必须要以 / 结尾!!!

addConverterFactory:

加这个 **addConverterFactory(GsonConverterFactory.create())** 固定的，则会将返回的Json数据直接解析为我们的数据Bean类

1.3.3 创建网络请求接口的实例

[复制代码](#)

```
private GetApi getApi;
getApi = retrofit.create(GetApi.class);
```

1.3.4 发送网络请求

[复制代码](#)

```
getApi.getUserInfo(userid).enqueue(new Callback<UserInfo>() {
    @Override
    public void onResponse(Call<UserInfo> call, Response<UserInfo> response) {
        if (response != null && response.body() != null) {
            //此处为获取到的信息
            UserInfo userInfo = response.body();
        }
    }
});
```

```
@Override
public void onFailure(Call<UserInfo> call, Throwable t) {
    Log.i(TAG, "onFailure: " + t);
}
});
```

使用时别忘了申请网络权限哦，使用时可能会遇到以下问题：

CLEARTEXT communication to mock-api.com not permitted by network security policy

这是因为 Android P 不允许明文访问，而前面的 mock 地址是 http 开头的，解决办法是在 AndroidManifest 中的 application 内加入下面这段代码即可：

```
android:usesCleartextTraffic="true"
```

复制代码

以上就是Retrofit最基本的一次网络请求，接下来我们详细看下Retrofit网络请求接口中主要有哪些注解。

2. Get请求

baseUrl为： <http://mock-api.com/2vKVbXK8.mock/>

2.1 @Query

访问地址

```
http://mock-api.com/2vKVbXK8.mock/api/getUserInfo?id=1234
```

复制代码
//1:

实例

```
@GET("api/getUserInfo")
Call<UserInfo> getUserInfo(@Query("id") String userId);
```

复制代码

```
public class UserInfo {
    private String userId;
    private String userName;
```

复制代码

当我们要访问的地址为: `baseUrl + getUserInfo?id=1234` , 以 `?` 形式拼接一个参数这种格式时, 就使用 `@Query` 注解, 该注解就是在`getUserInfo`后面添加 `?` ,并且以 `id=传来的参数userId` 的形式拼接 url

2.2 @QueryMap

访问地址

`http://mock-api.com/2vKVbXK8.mock/api/getArticalInfo?id=405&page=1`

[复制代码](#)

实例



```
@GET("api/getArticalInfo")
Call<ArticalInfo> getArticalInfo(@QueryMap Map<String, String> params);
```

[复制代码](#)

```
public class ArticalInfo {
    private String articalName;
    private String url;
}
```

[复制代码](#)

当要访问的地址是通过 `?` 形式拼接多个参数时就使用 `@QueryMap` 注解

具体使用:

```
Map<String, String> params = new HashMap<>();
params.put("id", "405");
params.put("page", "1");

getApi.getArticalInfo(params).enqueue(new Callback<ArticalInfo>() {
    @Override
    public void onResponse(Call<ArticalInfo> call, Response<ArticalInfo> response) {
        if (response != null && response.body() != null) {
            Log.i(TAG, "onRespons:" + response.body().toString());
        }
    }
    @Override
    public void onFailure(Call<ArticalInfo> call, Throwable t) {
        Log.i(TAG, "onFailure:" + t.getMessage());
    }
})
```

[复制代码](#)

2.3 @Path

访问地址

[复制代码](#)

```
http://mock-api.com/2vKVbXK8.mock/api/getDynamicInfo/1/data  
http://mock-api.com/2vKVbXK8.mock/api/getDynamicInfo/2/data
```

实例

[复制代码](#)

```
@GET("api/getDynamicInfo/{param}/data")  
Call<ResponseBody> getDynamicInfo(@Path("param")int param);
```

当要访问的地址由某个参数动态拼接而成时，使用 `@Path` 注解，上面实例中param这里具体填入的内容是后面调用该方法时传入的参数

[复制代码](#)

```
getApi.getDynamicInfo(param).enqueue(new Callback<ResponseBody>() {  
    @Override  
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {  
        try {  
            String str = new String(response.body().bytes());  
            Log.i(TAG, "onResponse: " + str);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
    @Override  
    public void onFailure(Call<ResponseBody> call, Throwable t) {  
        Log.i(TAG, "onFailure: " + t);  
    }  
});
```

大家仔细看下，你会发现 `Call<ResponseBody>` 这里和上面的不同，这里填入的是 `ResponseBody`，当返回的具体数据不是我们知道的Bean类时可以用这个，这样，我们可以通过上述字符串的形式获取到返回的内容。

2.4 @Url

`http://mock-api.com/2vKVbXK8.mock/api/getDynamicUrlData`

实例

```
@GET
Call<ResponseBody> getDynamicUrl(@Url String url);
```

当要访问的地址不只是动态的变几个参数，而是整个地址都要变化，甚至是基类地址也要变化时，这种动态地址就要用到 **@Url注解**

具体使用

```
String url = "http://mock-api.com/2vKVbXK8.mock/api/getDynamicUrlData"

getApi.getDynamicUrl(url).enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
        try {
            String str = new String(response.body().bytes());
            Log.i(TAG, "onResponse: " + str);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {
        Log.i(TAG, "onFailure: " + t);
    }
});
```

所以说这个注解很方便，上面所有注解所访问的链接均可通过此注解传入完整的Url进行访问。

注意

虽然说最终访问的地址与原先的baseUrl无关，但是baseUrl还是要以 **http://** 或 **https://** 开头，并且后面至少要跟一个字母或者其他东西，不然就会报错。

`http://`

http://a

我实际测试时，这个a这里写什么都可以，没有以 / 结尾也可以

2.5 @Headers("")

静态添加头部信息：包含添加单个头部、添加多个头部

通过@Headers("")注解，内部以key:value的方式填写内容

访问地址



http://mock-api.com/2vKVbXK8.mock/api/staticHeaderInfo

复制代码

实例

静态添加单个头部

```
@Headers("version:1.1")
@GET("api/staticHeaderInfo")
Call<GetBean> getStaticHeadersInfo();
```

复制代码

静态添加多个头部

```
@Headers({"version:1.1",
          "type:android"})
@GET("api/staticHeadersInfo")
Call<GetBean> getStaticMoreHeadersInfo();
```

复制代码

模客后台

2.6 @Header

动态添加单个头部信息

访问地址

`http://mock-api.com/2vKVbXK8.mock/api/dynamicHeadersInfo`

[复制代码](#)

实例

```
@GET("api/dynamicHeadersInfo")
Call<ResponseBody> getDynamicHeaderInfo(@Header("version") String version);
```

[复制代码](#)

```
getApi.getDynamicHeaderInfo("1.1.1").enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
    }
    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {
    }
});
```

[复制代码](#)

模客后台

可以看到我们访问时确实是在头部加入了version=1.1.1的信息。

2.7 @HeaderMap

动态添加多个头部信息

访问地址

`http://mock-api.com/2vKVbXK8.mock/api/dynamicHeadersInfo`

[复制代码](#)

[复制代码](#)

```
@GET("api/dynamicHeadersInfo")
Call<ResponseBody> getDynamicHeadersInfo(@HeaderMap Map<String, String> headers);
```

[复制代码](#)

```
Map<String, String> headers = new HashMap<>();
headers.put("version", "2.2.2");
headers.put("type", "Android");

getApi.getDynamicHeadersInfo(headers).enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
    }
    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {
    }
});
```



模客后台

可以看到我们在访问的时候两个头部信息已经加载header中

3. Post请求

对比着Get请求中的注解，Post请求中的注解就很好记了。

@Field 对应 @Query

@FieldMap 对应 @QueryMap

@Body 对应 @Url

注意

在Post请求中，尽量不要使用@Query和@QueryMap，因为它传入的参数是直接拼接在url上的，不安全，而@Field和@FieldMap是写入到Body中的。

3.1 @FormUrlEncoded

`@FormUrlEncoded` 用于修饰 `@Field` 注解和 `@FieldMap` 注解，将会自动将请求参数的类型调整为

`application/x-www-form-urlencoded`

3.2 @Field

访问地址

`http://mock-api.com/2vKVbXK8.mock/api/fieldParam`

复制代码

实例



```
@FormUrlEncoded
@POST("api/fieldParam")
Call<ResponseBody> postFieldFun(@Field("key") String key);
```

复制代码

记得加上 `@FormUrlEncoded`，否则会报以下错误：

```
@Field parameters can only be used with form encoding
```

复制代码

使用

```
private PostApi postApi;
postApi = retrofit.create(PostApi.class);

postApi.postFieldFun("myfittinglife").enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
        try {
            String str = new String(response.body().bytes());
            Log.i(TAG, "onResponse: " + str);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
})

@Override
public void onFailure(Call<ResponseBody> call, Throwable t) {
    Log.i(TAG, "onFailure: " + t);
}
```

复制代码



首页 ✓

探索掘金



模客后台

可以看到我们通过@Field注解添加的参数是写在body中而不是直接拼接在Url后面

这里是根据我们设立的规则来进行匹配的，当然我们也可以选择文本或其他的方式进行匹配。

3.3 @FieldMap

访问地址



<http://mock-api.com/2vKVbXK8.mock/api/fieldMapParam>

复制代码

实例

```
@FormUrlEncoded
@POST("api/fieldMapParam")
Call<ResponseBody> postFildMapFun(@FieldMap Map<String, String> params);
```

复制代码

@FieldMap时候于多个相同类型参数的传递

使用

```
Map<String, String> params = new HashMap<>();
params.put("key", "myfittinglife");
params.put("password", "123456");

postApi.postFildMapFun(params).enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {
        try {
            String str = new String(response.body().bytes());
            Log.i(TAG, "onResponse: " + str);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
})
```


复制代码

```
public void onFailure(Call<ResponseBody> call, Throwable t) {  
    Log.i(TAG, "onFailure: " + t);  
}  
});
```

模客后台

可以看到我们传入body内的两个参数

3.4 @Body

3.3 可以看到  `@FieldMap` 注解适合多个相同类型参数的传递，如果多个不同类型传递的话，总不能写多个 `@Field` 吧

```
Call<ResponseBody> postFieldFun(@Field("key") String key,@Field("num")int num);
```

如果要更多种类类型的话那就更繁琐了，所以这里我们可以用 `@Body` 注解，直接传入一个对象过去，对象内可包含多种类型数据。

访问地址

<http://mock-api.com/2vKVbXK8.mock/api/bodyParam>

复制代码

实例

```
@POST("api/bodyParam")  
Call<ResponseBody> postBodyFun(@Body PostBodyBean postBodyBean);
```

复制代码

```
public class PostBodyBean {  
    private String key;  
    private int num;  
    private boolean isTrue;  
}
```

复制代码

使用

```
PostBodyBean postBodyBean = new PostBodyBean("myfittinglife",1,true);
postApi.postBodyFun(postBodyBean).enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response)    {
        try {
            String str = new String(response.body().bytes());
            Log.i(TAG, "onResponse: " + str);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void onFailure(Call<ResponseBody> call, Throwable t) {
        Log.i(TAG, "onFailure: " + t);
    }
});
```



模客后台

可以看到我们传的Body

注意

使用@Body注解一定在创建Retrofit的时候加

上 `.addConverterFactory(GsonConverterFactory.create())` ,目的是将对象转化为json字符串进行传递,否则会报以下错误

```
Unable to create @Body converter for class PostBodyBean
```

3.5 @Part/@PartMap

这两个注解和文件上传相关, 在后面的文章中会进行详细讲解。

4. 注意

- 访问地址http开头, 记得在AndroidManifest内的application中加入

- GET请求不能使用@Body注解
- baseUrl网址末尾一定加 / 斜杠
- @Body注解一定要加 `.addConverterFactory(GsonConverterFactory.create())`
- 当@GET或@POST注解的url为全路径时（可能和baseUrl不是一个域），会直接使用注解的url的域。
- 使用@Path注解时，path对应的路径不能包含 / ，否则会将其转化为%2F，报如下错误。在遇到想动态的拼接多节url时，还是使用@Url吧。

Attempt to invoke virtual method 'byte[] okhttp3.ResponseBody.bytes()' on a null object

复制代码



5. 总结

以上就是关于Retrofit的一些注解的基本使用介绍，具体代码见[Github](#).

为了简便观察，文章中只是使用了一个注解来建立接口，其实在实际使用时，可以多个注解连用，例如下面所示：

```
@GET("api/{param}/getUserInfo")
Call<UserInfo> getUserInfo(@Path("params") String param, @Query("id") String userId);
```

复制代码

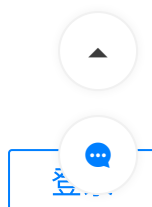
具体怎么结合还是要根据实际项目提供的url来使用。以上就是全部内容，能力有限，不对的地方还望指出，有帮助的话还望点个start，下篇文章见。

6. 参考文章

[Retrofit官方Github地址](#)

[Retrofit官网](#)

[说说Retrofit中的注解](#)



重拾丢却的梦 Lv2 Android开发

获得点赞 181 · 获得阅读 35,567

[关注](#)**安装掘金浏览器插件**

打开新标签页发现好内容，掘金、GitHub、Dribbble、ProductHunt 等站点内容轻松获取。快来安装掘金浏览器插件获取高质量内容吧！

空气转 

模拟网络请求 推荐 postman这个软件 😊

1年前



回复

相关推荐

张风捷特烈 2天前 Android Flutter

我的 FlutterUnit 开源项目竟然被套壳商用了

今天有个哥们加我微信，说我的 FlutterUnit 被抄了，还是商业收费的，我下载看了一下，...

6326 131 81

Battler 1天前 Android

Android修炼系列（二十），由系统字号来调整 App 字体大小

在平时开发中，一般 App 的界面布局都只会适配标准字体的尺寸，如果用户在设置中修改了系统字号大小，那么 Ap...

1075 30 2

付十一 1天前 Android

25岁，应该是什么样子？ | 2021 年中总结

过去十八岁 没戴表 不过有时间 够我 没有后顾 野性贪玩 霎眼廿七岁 时日无多方不敢偷懒 宏...

1047 25 16

九心 21小时前 Android

Android转场动画的前世今生

前一段时间做图片查看器的升级时，在打开图片查看器的时，找不到好的过渡方式。于是，...

[首页](#) [探索掘金](#)[登录](#)

究极逮虾户 1天前 Android

妖怪般的VerifyError | 奇形怪状的bug

如果你是因为这个bug，不幸点入这篇文章，我想说你运气属实不好，那么让我们掌声欢迎...

764 18 3

安安安安卓 2天前 Android

android Livedata最详尽的使用场景分析，让你爱上Livedata

本文基本上覆盖了Livedata的大部分用法和场景，如果您有更好的场景分析欢迎评论或私信我，帮助我我来充实文章...

669 21 14

i听风逝夜 1天前 Android

高仿微信下拉小程序入口动画

前言 突然发现微信下拉小程序入口动画非常细腻，比较好奇，所以仿照他做了一个，并不是很完美，部分效果还没完...

766 22 评论

JunBin 1天前 架构 Android

移动应用遗留系统重构（13）-🔥🔥🔥一镜到底！MVP重构示例篇

随着业务演进，代码中存在很多Activity及Controller的上帝类。今天我们将拿File Bundle作...

784 9 评论

付十一 2天前 Android

【Kotlin篇】差异化分析，let，run，with，apply及also

作用域函数是Kotlin比较重要的一个特性，共分为以下5种：let、run、with、apply 以及 als...

1235 41 6

易冬 2天前 OpenCV Android

Android OpenCV（四十九）：图像积分图

图像积分图 积分图是Crow在1984年首次提出，是为了在多尺度透视投影中提高渲染速度...

322 18 1

