# Retrofit用法详解

### 一、简介

[Retrofit](http://square.github.io/retrofit/" \t "http://duanyytop.github.io/2016/08/06/Retrofit%E7%94%A8%E6%B3%95%E8%AF%A6%E8%A7%A3/_blank)是Square公司开发的一款针对Android网络请求的框架，Retrofit2底层基于OkHttp实现的，[OkHttp](http://square.github.io/okhttp/" \t "http://duanyytop.github.io/2016/08/06/Retrofit%E7%94%A8%E6%B3%95%E8%AF%A6%E8%A7%A3/_blank)现在已经得到Google官方认可，大量的app都采用OkHttp做网络请求，其源码详见[OkHttp Github](https://github.com/square/okhttp)。

本文全部是在Retrofit2.0+版本基础上论述，所用例子全部来自豆瓣Api

首先先来看一个完整Get请求是如何实现：

创建业务请求接口，具体代码如下：

public interface BlueService {  
 @GET("book/search")  
 Call<BookSearchResponse> getSearchBooks(@Query("q") String name,  
 @Query("tag") String tag, @Query("start") int start,  
 @Query("count") int count);  
}

这里需要稍作说明，@GET注解就表示get请求，@Query表示请求参数，将会以key=value的方式拼接在url后面

需要创建一个Retrofit的示例，并完成相应的配置

Retrofit retrofit = new Retrofit.Builder()  
 .baseUrl("https://api.douban.com/v2/")  
 .addConverterFactory(GsonConverterFactory.create())  
 .build();  
BlueService service = retrofit.create(BlueService.class);

这里的baseUrl就是网络请求URL相对固定的地址，一般包括请求协议（如Http）、域名或IP地址、端口号等，当然还会有很多其他的配置，下文会详细介绍。还有addConverterFactory方法表示需要用什么转换器来解析返回值，GsonConverterFactory.create()表示调用Gson库来解析json返回值，具体的下文还会做详细介绍。

调用请求方法，并得到Call实例

Call<BookSearchResponse> call = service.getSearchBooks("小王子", "", 0, 3);

Call其实在Retrofit中就是行使网络请求并处理返回值的类，调用的时候会把需要拼接的参数传递进去，此处最后得到的url完整地址为

[https://api.douban.com/v2/book/search?q=%E5%B0%8F%E7%8E%8B%E5%AD%90&tag=&start=0&count=3](https://api.douban.com/v2/book/search?q=%E5%B0%8F%E7%8E%8B%E5%AD%90&tag=&start=0&count=3" \t "http://duanyytop.github.io/2016/08/06/Retrofit%E7%94%A8%E6%B3%95%E8%AF%A6%E8%A7%A3/_blank)

使用Call实例完成同步或异步请求

同步请求

BookSearchResponse response = call.execute().body();

这里需要注意的是网络请求一定要在子线程中完成，不能直接在UI线程执行，不然会crash

异步请求

call.enqueue(new Callback<BookSearchResponse>() {  
 @Override  
 public void onResponse(Call<BookSearchResponse> call, Response<BookSearchResponse> response) {  
 asyncText.setText("异步请求结果: " + response.body().books.get(0).altTitle);  
 }  
 @Override  
 public void onFailure(Call<BookSearchResponse> call, Throwable t) {  
  
 }  
});

### 二、如何使用

首先需要在build.gradle文件中引入需要的第三包，配置如下：

compile 'com.squareup.retrofit2:retrofit:2.1.0'  
compile 'com.squareup.retrofit2:converter-gson:2.1.0'  
compile 'com.squareup.retrofit2:adapter-rxjava:2.1.0'

引入完第三包接下来就可以使用Retrofit来进行网络请求了。接下来会对不同的请求方式做进一步的说明。

#### Get方法

##### 1. @Query

Get方法请求参数都会以key=value的方式拼接在url后面，Retrofit提供了两种方式设置请求参数。第一种就是像上文提到的直接在interface中添加@Query注解，还有一种方式是通过Interceptor实现，直接看如何通过Interceptor实现请求参数的添加。

|  |
| --- |
| public class CustomInterceptor implements Interceptor {  @Override  public Response intercept(Chain chain) throws IOException {  Request request = chain.request();  HttpUrl httpUrl = request.url().newBuilder()  .addQueryParameter("token", "tokenValue")  .build();  request = request.newBuilder().url(httpUrl).build();  return chain.proceed(request);  } } |

addQueryParameter就是添加请求参数的具体代码，这种方式比较适用于所有的请求都需要添加的参数，一般现在的网络请求都会添加token作为用户标识，那么这种方式就比较适合。

创建完成自定义的Interceptor后，还需要在Retrofit创建client处完成添加

|  |
| --- |
| addInterceptor(new CustomInterceptor()) |

##### 2. @QueryMap

如果Query参数比较多，那么可以通过@QueryMap方式将所有的参数集成在一个Map统一传递，还以上文中的get请求方法为例

|  |
| --- |
| public interface BlueService {  @GET("book/search")  Call<BookSearchResponse> getSearchBooks(@QueryMap Map<String, String> options); } |

调用的时候将所有的参数集合在统一的map中即可

|  |
| --- |
| Map<String, String> options = new HashMap<>(); map.put("q", "小王子"); map.put("tag", null); map.put("start", "0"); map.put("count", "3"); Call<BookSearchResponse> call = mBlueService.getSearchBooks(options); |

##### 3. Query集合

假如你需要添加相同Key值，但是value却有多个的情况，一种方式是添加多个@Query参数，还有一种简便的方式是将所有的value放置在列表中，然后在同一个@Query下完成添加，实例代码如下：

|  |
| --- |
| public interface BlueService {  @GET("book/search")  Call<BookSearchResponse> getSearchBooks(@Query("q") List<String> name); } |

最后得到的url地址为

|  |
| --- |
| https://api.douban.com/v2/book/search?q=leadership&q=beyond%20feelings |

##### 4. Query非必填

如果请求参数为非必填，也就是说即使不传该参数，服务端也可以正常解析，那么如何实现呢？其实也很简单，请求方法定义处还是需要完整的Query注解，某次请求如果不需要传该参数的话，只需填充null即可。

针对文章开头提到的get的请求，加入按以下方式调用

|  |
| --- |
| Call<BookSearchResponse> call = service.getSearchBooks("小王子", null, 0, 3); |

那么得到的url地址为

|  |  |
| --- | --- |
| 1 | https://api.douban.com/v2/book/search?q=%E5%B0%8F%E7%8E%8B%E5%AD%90&start=0&count=3 |

##### 5. @Path

如果请求的相对地址也是需要调用方传递，那么可以使用@Path注解，示例代码如下：

|  |
| --- |
| @GET("book/{id}") Call<BookResponse> getBook(@Path("id") String id); |

业务方想要在地址后面拼接书籍id，那么通过Path注解可以在具体的调用场景中动态传递，具体的调用方式如下：

|  |
| --- |
| Call<BookResponse> call = service.getBook("1003078"); |

此时的url地址为

|  |
| --- |
| https://api.douban.com/v2/book/1003078 |

@Path可以用于任何请求方式，包括Post，Put，Delete等等

### Post请求

##### 1. @field

Post请求需要把请求参数放置在请求体中，而非拼接在url后面，先来看一个简单的例子

|  |
| --- |
| @FormUrlEncoded @POST("book/reviews") Call<String> addReviews(@Field("book") String bookId, @Field("title") String title, @Field("content") String content, @Field("rating") String rating); |

这里有几点需要说明的

@FormUrlEncoded将会自动将请求参数的类型调整为application/x-www-form-urlencoded，假如content传递的参数为Good Luck，那么最后得到的请求体就是

|  |
| --- |
| content=Good+Luck |

FormUrlEncoded不能用于Get请求

@Field注解将每一个请求参数都存放至请求体中，还可以添加encoded参数，该参数为boolean型，具体的用法为

|  |
| --- |
| @Field(value = "book", encoded = true) String book |

encoded参数为true的话，key-value-pair将会被编码，即将中文和特殊字符进行编码转换

##### 2. @FieldMap

上述Post请求有4个请求参数，假如说有更多的请求参数，那么通过一个一个的参数传递就显得很麻烦而且容易出错，这个时候就可以用FieldMap

|  |
| --- |
| @FormUrlEncoded @POST("book/reviews") Call<String> addReviews(@FieldMap Map<String, String> fields); |

##### 3. @Body

如果Post请求参数有多个，那么统一封装到类中应该会更好，这样维护起来会非常方便

|  |
| --- |
| @FormUrlEncoded @POST("book/reviews") Call<String> addReviews(@Body Reviews reviews);  public class Reviews {  public String book;  public String title;  public String content;  public String rating; } |

#### 其他请求方式

除了Get和Post请求，Http请求还包括Put,Delete等等，用法和Post相似，所以就不再单独介绍了。

#### 上传

上传因为需要用到Multipart，所以需要单独拿出来介绍，先看一个具体上传的例子

首先还是需要新建一个interface用于定义上传方法

|  |
| --- |
| public interface FileUploadService {   // 上传单个文件  @Multipart  @POST("upload")  Call<ResponseBody> uploadFile(  @Part("description") RequestBody description,  @Part MultipartBody.Part file);   // 上传多个文件  @Multipart  @POST("upload")  Call<ResponseBody> uploadMultipleFiles(  @Part("description") RequestBody description,  @Part MultipartBody.Part file1,  @Part MultipartBody.Part file2); } |

接下来我们还需要在Activity和Fragment中实现两个工具方法，代码如下：

|  |
| --- |
| public static final String MULTIPART\_FORM\_DATA = "multipart/form-data";  @NonNull private RequestBody createPartFromString(String descriptionString) {   return RequestBody.create(  MediaType.parse(MULTIPART\_FORM\_DATA), descriptionString); }  @NonNull private MultipartBody.Part prepareFilePart(String partName, Uri fileUri) {   File file = FileUtils.getFile(this, fileUri);   // 为file建立RequestBody实例  RequestBody requestFile =  RequestBody.create(MediaType.parse(MULTIPART\_FORM\_DATA), file);   // MultipartBody.Part借助文件名完成最终的上传  return MultipartBody.Part.createFormData(partName, file.getName(), requestFile); } |

好了，接下来就是最终的上传文件代码了

|  |
| --- |
| Uri file1Uri = ... // 从文件选择器或者摄像头中获取  Uri file2Uri = ...   // 创建上传的service实例 FileUploadService service =   ServiceGenerator.createService(FileUploadService.class);  // 创建文件的part (photo, video, ...) MultipartBody.Part body1 = prepareFilePart("video", file1Uri);  MultipartBody.Part body2 = prepareFilePart("thumbnail", file2Uri);  // 添加其他的part RequestBody description = createPartFromString("hello, this is description speaking");  // 最后执行异步请求操作 Call<ResponseBody> call = service.uploadMultipleFiles(description, body1, body2);  call.enqueue(new Callback<ResponseBody>() {   @Override  public void onResponse(Call<ResponseBody> call,  Response<ResponseBody> response) {  Log.v("Upload", "success");  }  @Override  public void onFailure(Call<ResponseBody> call, Throwable t) {  Log.e("Upload error:", t.getMessage());  } }); |

### 三、其他必须知道的事项

#### 1. 添加自定义的header

Retrofit提供了两个方式定义Http请求头参数：静态方法和动态方法，静态方法不能随不同的请求进行变化，头部信息在初始化的时候就固定了。而动态方法则必须为每个请求都要单独设置。

静态方法

|  |
| --- |
| public interface BlueService {  @Headers("Cache-Control: max-age=640000")  @GET("book/search")  Call<BookSearchResponse> getSearchBooks(@Query("q") String name,   @Query("tag") String tag, @Query("start") int start,   @Query("count") int count); } |

当然你想添加多个header参数也是可以的，写法也很简单

|  |
| --- |
| public interface BlueService {  @Headers({  "Accept: application/vnd.yourapi.v1.full+json",  "User-Agent: Your-App-Name"  })  @GET("book/search")  Call<BookSearchResponse> getSearchBooks(@Query("q") String name,   @Query("tag") String tag, @Query("start") int start,   @Query("count") int count); } |

此外也可以通过Interceptor来定义静态请求头

|  |
| --- |
| public class RequestInterceptor implements Interceptor {  @Override  public Response intercept(Chain chain) throws IOException {  Request original = chain.request();  Request request = original.newBuilder()  .header("User-Agent", "Your-App-Name")  .header("Accept", "application/vnd.yourapi.v1.full+json")  .method(original.method(), original.body())  .build();  return chain.proceed(request);  } } |

添加header参数Request提供了两个方法，一个是****header(key, value)****，另一个是****.addHeader(key, value)****，两者的区别是，header()如果有重名的将会覆盖，而addHeader()允许相同key值的header存在

然后在OkHttp创建Client实例时，添加RequestInterceptor即可

|  |
| --- |
| private static OkHttpClient getNewClient(){  return new OkHttpClient.Builder()  .addInterceptor(new RequestInterceptor())  .connectTimeout(DEFAULT\_TIMEOUT, TimeUnit.SECONDS)  .build(); } |

动态方法

|  |
| --- |
| public interface BlueService {  @GET("book/search")  Call<BookSearchResponse> getSearchBooks(  @Header("Content-Range") String contentRange,   @Query("q") String name, @Query("tag") String tag,   @Query("start") int start, @Query("count") int count); } |

#### 2. 网络请求日志

调试网络请求的时候经常需要关注一下请求参数和返回值，以便判断和定位问题出在哪里，Retrofit官方提供了一个很方便查看日志的Interceptor，你可以控制你需要的打印信息类型，使用方法也很简单。

首先需要在build.gradle文件中引入logging-interceptor

|  |
| --- |
| compile 'com.squareup.okhttp3:logging-interceptor:3.4.1' |

同上文提到的CustomInterceptor和RequestInterceptor一样，添加到OkHttpClient创建处即可，完整的示例代码如下：

|  |
| --- |
| private static OkHttpClient getNewClient(){  HttpLoggingInterceptor logging = new HttpLoggingInterceptor();  logging.setLevel(HttpLoggingInterceptor.Level.BODY);  return new OkHttpClient.Builder()  .addInterceptor(new CustomInterceptor())  .addInterceptor(logging)  .connectTimeout(DEFAULT\_TIMEOUT, TimeUnit.SECONDS)  .build(); } |

HttpLoggingInterceptor提供了4中控制打印信息类型的等级，分别是NONE，BASIC，HEADERS，BODY，接下来分别来说一下相应的打印信息类型。

NONE没有任何日志信息

Basic打印请求类型，URL，请求体大小，返回值状态以及返回值的大小

|  |
| --- |
| D/HttpLoggingInterceptor$Logger: --> POST /upload HTTP/1.1 (277-byte body)  D/HttpLoggingInterceptor$Logger: <-- HTTP/1.1 200 OK (543ms, -1-byte body) |

Headers打印返回请求和返回值的头部信息，请求类型，URL以及返回值状态码

|  |
| --- |
| <-- 200 OK https://api.douban.com/v2/book/search?q=%E5%B0%8F%E7%8E%8B%E5%AD%90&start=0&count=3&token=tokenValue (3787ms) D/OkHttp: Date: Sat, 06 Aug 2016 14:26:03 GMT D/OkHttp: Content-Type: application/json; charset=utf-8 D/OkHttp: Transfer-Encoding: chunked D/OkHttp: Connection: keep-alive D/OkHttp: Keep-Alive: timeout=30 D/OkHttp: Vary: Accept-Encoding D/OkHttp: Expires: Sun, 1 Jan 2006 01:00:00 GMT D/OkHttp: Pragma: no-cache D/OkHttp: Cache-Control: must-revalidate, no-cache, private D/OkHttp: Set-Cookie: bid=D6UtQR5N9I4; Expires=Sun, 06-Aug-17 14:26:03 GMT; Domain=.douban.com; Path=/ D/OkHttp: X-DOUBAN-NEWBID: D6UtQR5N9I4 D/OkHttp: X-DAE-Node: dis17 D/OkHttp: X-DAE-App: book D/OkHttp: Server: dae D/OkHttp: <-- END HTTP |

Body打印请求和返回值的头部和body信息

|  |
| --- |
| <-- 200 OK https://api.douban.com/v2/book/search?q=%E5%B0%8F%E7%8E%8B%E5%AD%90&tag=&start=0&count=3&token=tokenValue (3583ms) D/OkHttp: Connection: keep-alive D/OkHttp: Date: Sat, 06 Aug 2016 14:29:11 GMT D/OkHttp: Keep-Alive: timeout=30 D/OkHttp: Content-Type: application/json; charset=utf-8 D/OkHttp: Vary: Accept-Encoding D/OkHttp: Expires: Sun, 1 Jan 2006 01:00:00 GMT D/OkHttp: Transfer-Encoding: chunked D/OkHttp: Pragma: no-cache D/OkHttp: Connection: keep-alive D/OkHttp: Cache-Control: must-revalidate, no-cache, private D/OkHttp: Keep-Alive: timeout=30 D/OkHttp: Set-Cookie: bid=ESnahto1\_Os; Expires=Sun, 06-Aug-17 14:29:11 GMT; Domain=.douban.com; Path=/ D/OkHttp: Vary: Accept-Encoding D/OkHttp: X-DOUBAN-NEWBID: ESnahto1\_Os D/OkHttp: Expires: Sun, 1 Jan 2006 01:00:00 GMT D/OkHttp: X-DAE-Node: dis5 D/OkHttp: Pragma: no-cache D/OkHttp: X-DAE-App: book D/OkHttp: Cache-Control: must-revalidate, no-cache, private D/OkHttp: Server: dae D/OkHttp: Set-Cookie: bid=5qefVyUZ3KU; Expires=Sun, 06-Aug-17 14:29:11 GMT; Domain=.douban.com; Path=/ D/OkHttp: X-DOUBAN-NEWBID: 5qefVyUZ3KU D/OkHttp: X-DAE-Node: dis17 D/OkHttp: X-DAE-App: book D/OkHttp: Server: dae D/OkHttp: {"count":3,"start":0,"total":778,"books":[{"rating":{"max":10,"numRaters":202900,"average":"9.0","min":0},"subtitle":"","author":["[法] 圣埃克苏佩里"],"pubdate":"2003-8","tags":[{"count":49322,"name":"小王子","title":"小王子"},{"count":41381,"name":"童话","title":"童话"},{"count":19773,"name":"圣埃克苏佩里","title":"圣埃克苏佩里"} D/OkHttp: <-- END HTTP (13758-byte body) |

#### 3. 为某个请求设置完整的URL

​ 假如说你的某一个请求不是以base\_url开头该怎么办呢？别着急，办法很简单，看下面这个例子你就懂了

|  |
| --- |
| public interface BlueService {   @GET  public Call<ResponseBody> profilePicture(@Url String url); }  Retrofit retrofit = Retrofit.Builder()   .baseUrl("https://your.api.url/");  .build();  BlueService service = retrofit.create(BlueService.class);  service.profilePicture("https://s3.amazon.com/profile-picture/path"); |

​ 直接用@Url注解的方式传递完整的url地址即可。

#### 4. 取消请求

Call提供了cancel方法可以取消请求，前提是该请求还没有执行

|  |
| --- |
| String fileUrl = "http://futurestud.io/test.mp4";  Call<ResponseBody> call =   downloadService.downloadFileWithDynamicUrlSync(fileUrl); call.enqueue(new Callback<ResponseBody>() {   @Override  public void onResponse(Call<ResponseBody> call, Response<ResponseBody> response) {  Log.d(TAG, "request success");  }   @Override  public void onFailure(Call<ResponseBody> call, Throwable t) {  if (call.isCanceled()) {  Log.e(TAG, "request was cancelled");  } else {  Log.e(TAG, "other larger issue, i.e. no network connection?");  }  } });  }  // 触发某个动作，例如用户点击了取消请求的按钮 call.cancel();  } |