**[好用的网络请求库Retrofit2（入门及讲解）](http://blog.csdn.net/biezhihua/article/details/49232289)**

**首先，在gralde文件中引入后续要用到的库。**

compile 'com.squareup.retrofit:retrofit:2.0.0-beta2'

compile 'com.squareup.retrofit:converter-gson:2.0.0-beta2'

compile 'com.squareup.retrofit:adapter-rxjava:2.0.0-beta2'

compile 'com.squareup.okhttp:okhttp:2.4.0'

compile 'io.reactivex:rxjava:1.0.14'

compile 'io.reactivex:rxandroid:1.0.1'

**接下来创建Retrofit2的实例，并设置BaseUrl和Gson转换。**

Retrofit retrofit = new Retrofit.Builder()

.baseUrl("https://api.github.com")

.addConverterFactory(GsonConverterFactory.create())

.client(new OkHttpClient())

.build();

创建服务类和Bean

public static class Contributor {

public final String login;

public final int contributions;

public Contributor(String login, int contributions) {

this.login = login;

this.contributions = contributions;

}

@Override

public String toString() {

return "Contributor{" +

"login='" + login + '\'' +

", contributions=" + contributions +

'}';

}

}

public interface GitHub {

@GET("/repos/{owner}/{repo}/contributors")

Call<List<Contributor>> contributors(

@Path("owner") String owner,

@Path("repo") String repo);

}

创建请求服务，并为网络请求方法设置参数

GitHub gitHubService = retrofit.create(GitHub.class);

Call<List<Contributor>> call = gitHubService.contributors("square", "retrofit");

最后，请求网络，并获取响应

try{

Response<List<Contributor>> response = call.execute(); // 同步

Log.d(TAG, "response:" + response.body().toString());

} catch (IOException e) {

e.printStackTrace();

}

## Call是Retrofit中重要的一个概念，代表被封装成单个请求/响应的交互行为

通过调用Retrofit2的execute（同步）或者enqueue（异步）方法，发送请求到网络服务器，并返回一个响应（Response）。

1. 独立的请求和响应模块
2. 从响应处理分离出请求创建
3. 每个实例只能使用一次。
4. Call可以被克隆。
5. 支持同步和异步方法。
6. 能够被取消。

由于call只能被执行一次，所以按照上面的顺序执行会得到如下错误。

java.lang.IllegalStateException: Already executed

我们可以通过clone，来克隆一份call，从新调用。

// clone

Call<List<Contributor>> call1 = call.clone();// 5. 请求网络，异步

call1.enqueue(new Callback<List<Contributor>>() {

@Override

public void onResponse(Response<List<Contributor>> response, Retrofit retrofit) {

Log.d(TAG, "response:" + response.body().toString());

}

@Override

public void onFailure(Throwable t) {

}

});

## 参数相关

网络访问肯定要涉及到参数请求，Retrofit为我们提供了各式各样的组合方法。下面以标题＋小例子的方式给出讲解。

#### 固定查询参数

// 服务

interface SomeService {

@GET("/some/endpoint?fixed=query")

Call<SomeResponse> someEndpoint();

}

// 方法调用

someService.someEndpoint();

// 请求头

// GET /some/endpoint?fixed=query HTTP/1.1

#### 动态参数

// 服务

interface SomeService {

@GET("/some/endpoint")

Call<SomeResponse> someEndpoint(

@Query("dynamic") String dynamic);

}

// 方法调用

someService.someEndpoint("query");

// 请求头

// GET /some/endpoint?dynamic=query HTTP/1.1

### 动态参数（Map）

// 服务

interface SomeService {

@GET("/some/endpoint")

Call<SomeResponse> someEndpoint(

@QueryMap Map<String, String> dynamic);

}

// 方法调用

someService.someEndpoint(

Collections.singletonMap("dynamic", "query"));

// 请求头

// GET /some/endpoint?dynamic=query HTTP/1.1

#### 省略动态参数

interface SomeService {

@GET("/some/endpoint")

Call<SomeResponse> someEndpoint(

@Query("dynamic") String dynamic);

}

// 方法调用

someService.someEndpoint(null);

// 请求头

// GET /some/endpoint HTTP/1.1

#### 固定+动态参数

interface SomeService {

@GET("/some/endpoint?fixed=query")

Call<SomeResponse> someEndpoint(

@Query("dynamic") String dynamic);

}

// 方法调用

someService.someEndpoint("query");

// 请求头

// GET /some/endpoint?fixed=query&dynamic=query HTTP/1.1

#### 路径替换

interface SomeService {

@GET("/some/endpoint/{thing}")

Call<SomeResponse> someEndpoint(

@Path("thing") String thing);

}

someService.someEndpoint("bar");

// GET /some/endpoint/bar HTTP/1.1

#### 固定头

interface SomeService {

@GET("/some/endpoint")

@Headers("Accept-Encoding: application/json")

Call<SomeResponse> someEndpoint();

}

someService.someEndpoint();

// GET /some/endpoint HTTP/1.1

// Accept-Encoding: application/json

#### 动态头

interface SomeService {

@GET("/some/endpoint")

Call<SomeResponse> someEndpoint(

@Header("Location") String location);

}

someService.someEndpoint("Droidcon NYC 2015");

// GET /some/endpoint HTTP/1.1

// Location: Droidcon NYC 2015

#### 固定+动态头

interface SomeService {

@GET("/some/endpoint")

@Headers("Accept-Encoding: application/json")

Call<SomeResponse> someEndpoint(

@Header("Location") String location);

}

someService.someEndpoint("Droidcon NYC 2015");

// GET /some/endpoint HTTP/1.1

// Accept-Encoding: application/json

// Location: Droidcon NYC 2015

#### Post请求，无Body

interface SomeService {

@POST("/some/endpoint")

Call<SomeResponse> someEndpoint();

}

someService.someEndpoint();

// POST /some/endpoint HTTP/1.1

// Content-Length: 0

#### Post请求有Body

interface SomeService {

@POST("/some/endpoint")

Call<SomeResponse> someEndpoint(

@Body SomeRequest body);

}

someService.someEndpoint();

// POST /some/endpoint HTTP/1.1

// Content-Length: 3// Content-Type: greeting

//

// Hi!

#### 表单编码字段

interface SomeService {

@FormUrlEncoded

@POST("/some/endpoint")

Call<SomeResponse> someEndpoint(

@Field("name1") String name1,

@Field("name2") String name2);

}

someService.someEndpoint("value1", "value2");

// POST /some/endpoint HTTP/1.1

// Content-Length: 25

// Content-Type: application/x-www-form-urlencoded

//

// name1=value1&name2=value2

#### 表单编码字段（Map）

interface SomeService {

@FormUrlEncoded

@POST("/some/endpoint")

Call<SomeResponse> someEndpoint(

@FieldMap Map<String, String> names);

}

someService.someEndpoint(

// ImmutableMap是OKHttp中的工具类

ImmutableMap.of("name1", "value1", "name2", "value2"));

// POST /some/endpoint HTTP/1.1

// Content-Length: 25

// Content-Type: application/x-www-form-urlencoded

//

// name1=value1&name2=value2

## 动态Url(Dynamic URL parameter)

interface GitHubService {

@GET("/repos/{owner}/{repo}/contributors")

Call<List<Contributor>> repoContributors(

@Path("owner") String owner,

@Path("repo") String repo);

@GET

Call<List<Contributor>> repoContributorsPaginate(

@Url String url);

}

// 调用

Call<List<Contributor>> call = gitHubService.repoContributors("square", "retrofit");

Response<List<Contributor>> response = call.execute();

// 响应结果

// HTTP/1.1 200 OK

// Link: <https://api.github.com/repositories/892275/contributors?

page=2>; rel="next", <https://api.github.com/repositories/892275/

contributors?page=3>; rel="last"

// 获取到头中的数据

String links = response.headers().get("Link");

String nextLink = nextFromGitHubLinks(links);

// https://api.github.com/repositories/892275/contributors?page=2

### 可插拔的执行机制(Multiple, pluggable execution mechanisms)

interface GitHubService {

@GET("/repos/{owner}/{repo}/contributors")

// Call 代表的是CallBack回调机制

Call<List<Contributor>> repoContributors(

@Path("owner") String owner,

@Path("repo") String repo);

@GET("/repos/{owner}/{repo}/contributors")

// Observable 代表的是RxJava的执行

Observable<List<Contributor>> repoContributors2(

@Path("owner") String owner,

@Path("repo") String repo);

@GET("/repos/{owner}/{repo}/contributors")

Future<List<Contributor>> repoContributors3(

@Path("owner") String owner,

@Path("repo") String repo);

}

注意，要在构建Retrofit时指定适配器模式为RxJavaCallAdapterFactory

Retrofit retrofit = new Retrofit.Builder()

.addConverterFactory(GsonConverterFactory.create())

.addCallAdapterFactory(RxJavaCallAdapterFactory.create())

.baseUrl("http://www.duitang.com")

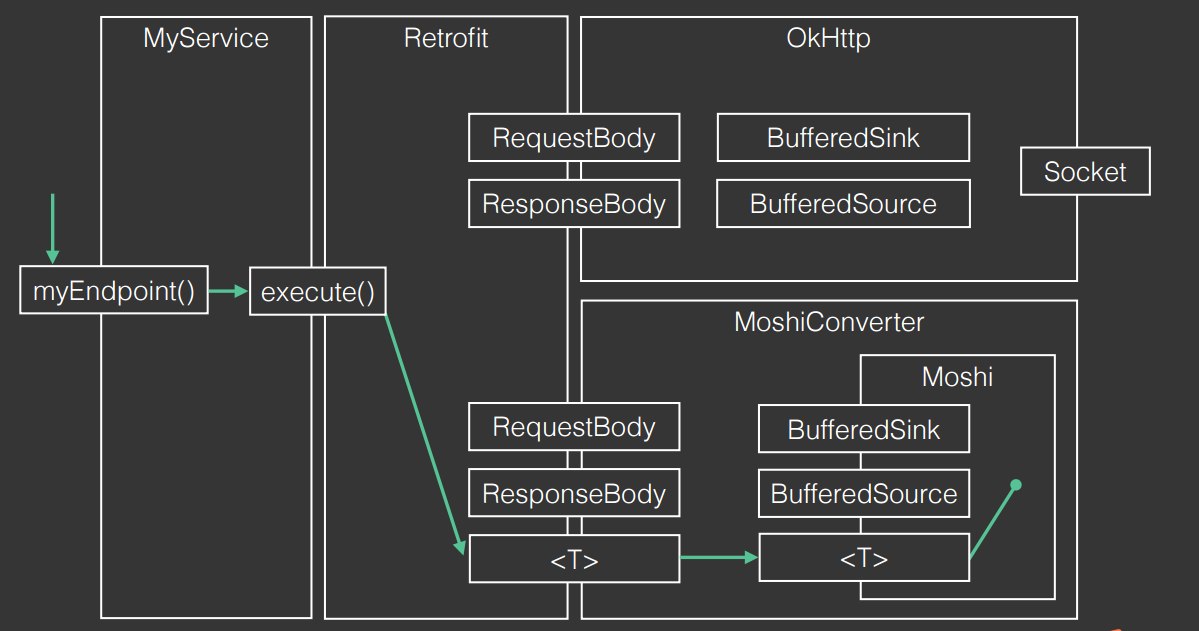
.build();

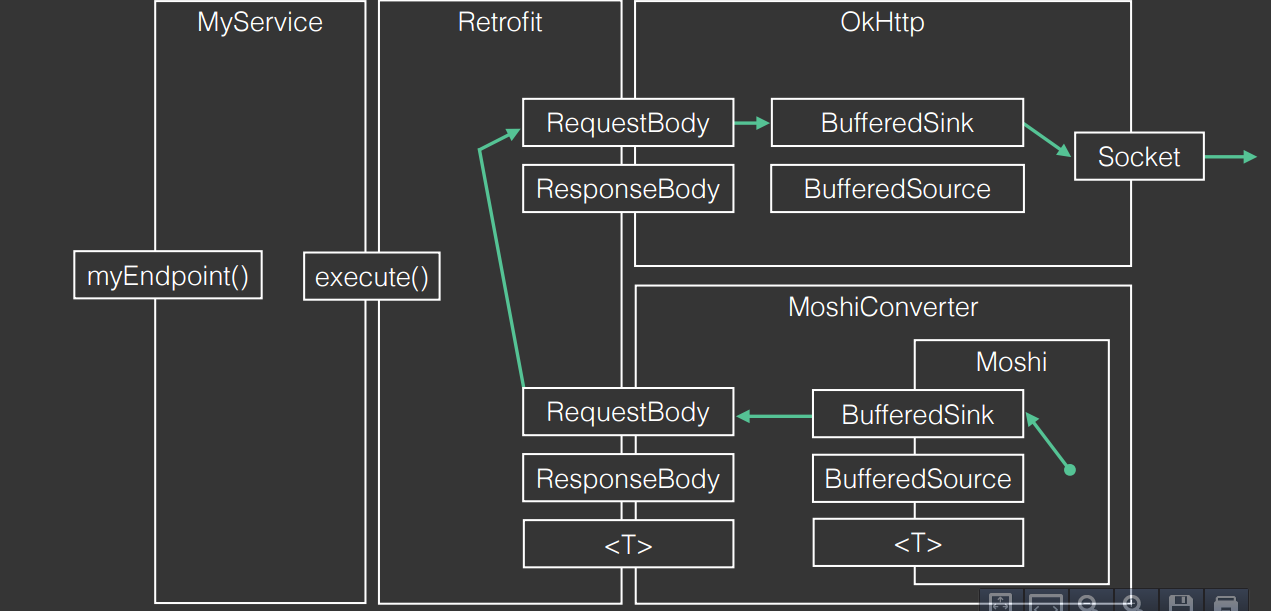
否则，会报出如下错误：

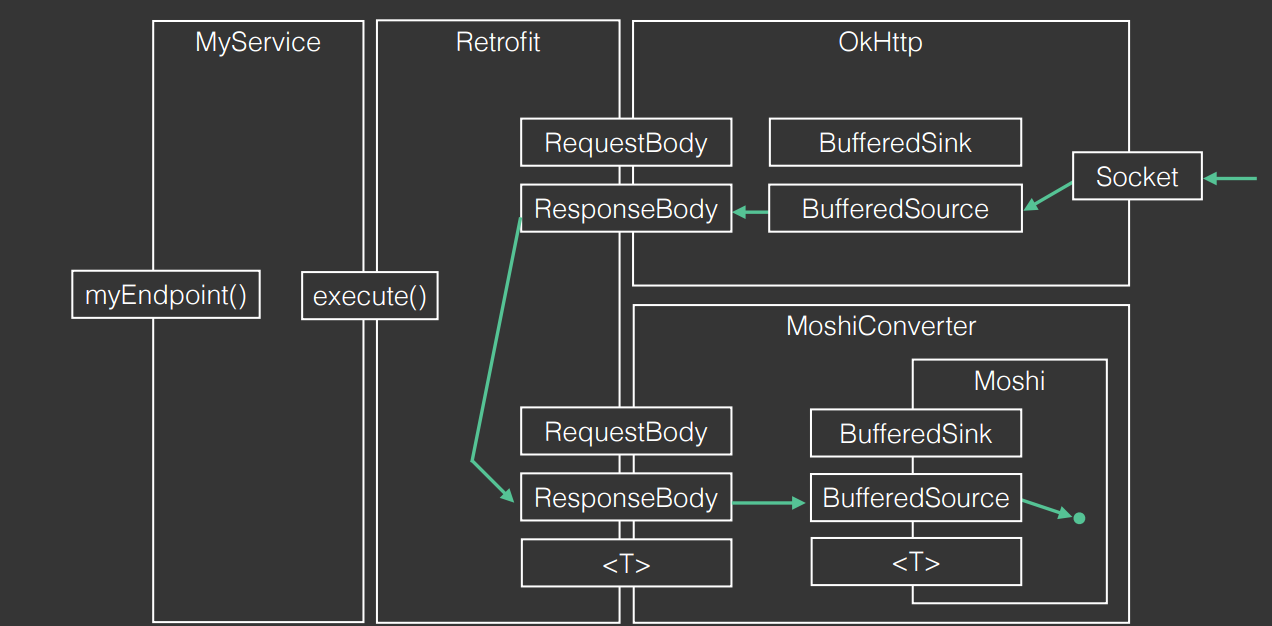
Caused by: java.lang.IllegalArgumentException: Could not locate call adapter for rx.Observable<com.bzh.sampleretrofit.ClubBean>. Tried:

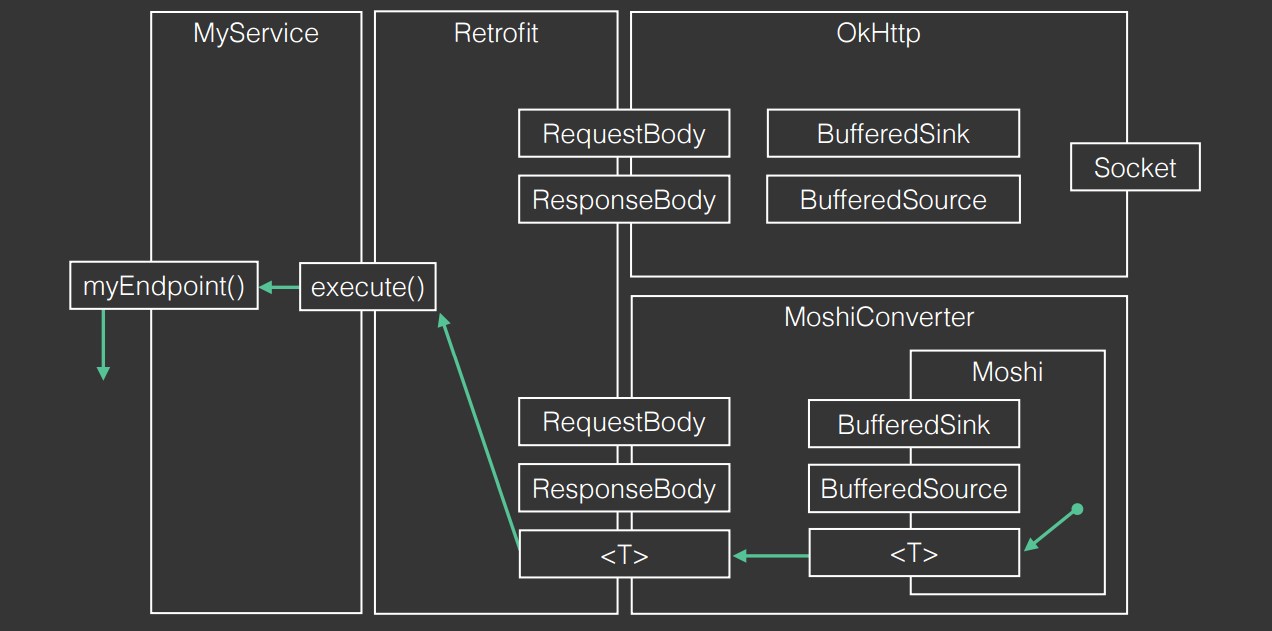
\* retrofit.ExecutorCallAdapterFactory

## Retrofit执行模式









## 最后

Retrofit作为一个上层框架，自然有很多底层lib库支持，okio和okhttp都包含其中。   
