Volley：优点：Google出口，高频小数据访问，缺点：大数据访问

Retrofit：优点：Github排行第一，且是Square公司出口。底层实现是OkHttp，

# Retrofit

compile 'com.squareup.retrofit2:retrofit:2.2.0'

最小需求：Java7，Android 2.3

代码混淆：

# Platform calls Class.forName on types which do not exist on Android to determine platform.

-dontnote retrofit2.Platform

# Platform used when running on Java 8 VMs. Will not be used at runtime.

-dontwarn retrofit2.Platform$Java8

# Retain generic type information for use by reflection by converters and adapters.

-keepattributes Signature

# Retain declared checked exceptions for use by a Proxy instance.

-keepattributes Exceptions

Retrofit依赖了okio，所以还得加下面的：

-dontwarn okio.\*\*

## 官方文档

### 介绍

Retrofit是Android和Java平台下类型安全的Http客户端

Retrofit将HTTP API转换为Java接口：

public interface GitHubService {

@GET("users/{user}/repos")

Call<List<Repo>> listRepos(@Path("user") String user);

}

Retrofit类生成GitHubService接口的一个实例：

Retrofit retrofit = new Retrofit.Builder()

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

.build();

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

创建的GitHubService的每个Call都可以制造一个同步或异步的HTTP请求到远程服务器

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

Response< List<Repo>> response = call.execute(); // 同步请求

call.enqueue(mCallback); // 异步请求

使用注解来描述HTTP请求:

* URL参数替换与查询参数支持
* 对象转换为请求体（例如：JSON、协议缓存）
* 多请求体与文件上传

### API声明

在接口方法上的注解和它的参数表示一个请求将会被怎样处理。

请求类型

每个方法必须有一个HTTP注解来说明请求方法和相对URL。有5种固定的注解：GET，POST，PUT，DELETE和HEAD。资源的相对URL在注释中指定。

@GET("users/list")

你也可以在URL中指定查询参数

@GET("users/list?sort=desc")

URL操作

可以在方法中使用替换块和参数动态地更新一个请求URL。一个替换块是通过{ 和 }包围的一个字母数字的字符串。相应的参数必须用同样的字符串通过@Path注解。

@GET("group/{id}/users")

Call<List<User>> groupList(@Path("id") int groupId);

也可以添加查询参数

@GET("group/{id}/users")

Call<List<User>> groupList(@Path("id") int groupId, @Query("sort") String sort);

对于复杂的查询参数可以用Map组合起来

@GET("group/{id}/users")

Call<List<User>> groupList(@Path("id") int groupId, @QueryMap Map<String, String> options);

请求体

一个对象可以通过@Body注解在HTTP请求体中指定。

可以使用@Body注解来把一个对象指定为一个HTTP请求体

@POST("user/new")

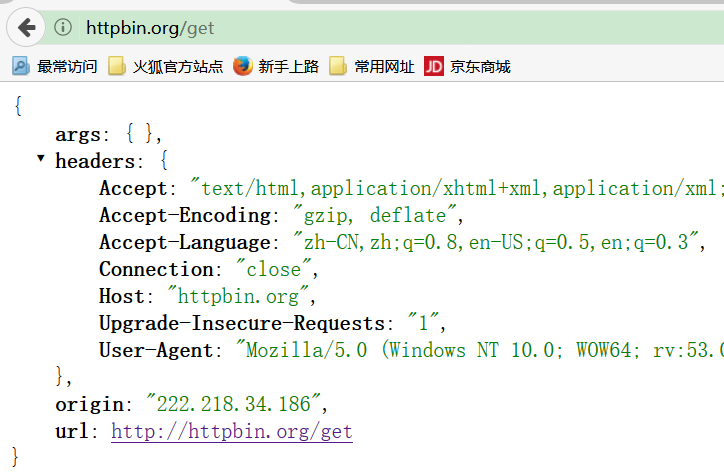
Call<User> createUser(@Body User user);

这个对象同样会被在Retrofit实例中指定的转换器转换。如果没有添加转换器，则只会使用RequestBody。

## 简单易理解的文档

### get请求

在浏览器访问：<http://httpbin.org/get>如下：



如上图，是一个get请求，没有请求参数，请求返回的结果是一个Json字符串，在Android中，如何使用Retrofit完成这个get请求呢，步骤如下：

1、设置Retrofit依赖：'com.squareup.retrofit2:retrofit:2.2.0'

2、创建请求接口

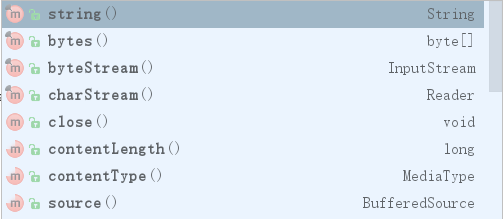
**public interface** ILogin { // 必须声明为接口，类名随意  
 @GET(**"get"**) // @GET为请求类型，参数“get”对应<http://bttpbin.org/get>中的get,<http://httpbin.org/> 为BaseUrl  
 Call<ResponseBody> login();// 方法名随意，返回类型是固定写法，执行这个方法就是发送登录请求，返回结果保存在ResponseBody中  
}

3、固定写法：

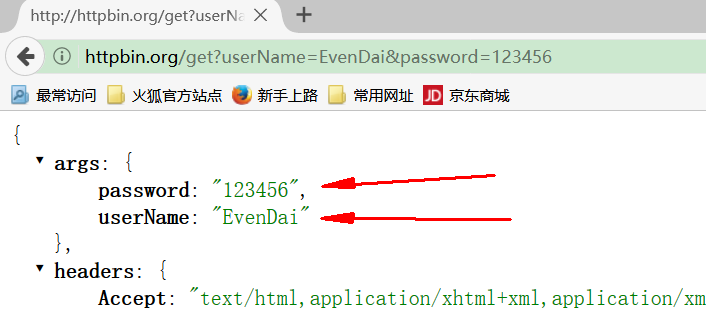
Retrofit retrofit = **new** Retrofit.Builder().baseUrl(**"http://httpbin.org"**).build();  
ILogin login = retrofit.create(ILogin.**class**);  
Call<ResponseBody> call = login.login();  
**try** {  
 Response<ResponseBody> response = call.execute(); *// 注意：这是同步请求，要开个子线程来执行，这里为了代码简洁没写子线程* ResponseBody responseBody = response.body();  
 String json = responseBody.string();  
} **catch** (IOException e) {  
 e.printStackTrace();  
}

注：call.execute()是同步请求，call.enqueue(callback);是异步请求

注：因为返回结果是String，所以我们调用了responseBody的string方法快速的把流中的内容变成了String，ResponseBody代表了响应体，它还有如下方法获取响应体相关的数据：



4、同样的url，咱们带点参数：<http://httpbin.org/get?userName=EvenDai&password=123456> ，浏览器访问效果如下：



在代码中，可以在GET注解参数中直接写参数，如下：|

**public interface** ILogin {  
 @GET(**"get?userName=EvenDai&password=123456"**)  
 Call<ResponseBody> login();  
}

这种写法基本不会用，因为参数是在调用方法的时候才传的，使用@Query注解在方法中增加请求参数，如下：

**public interface** ILogin {  
 @GET(**"get"**)  
 Call<ResponseBody> login(@Query(**"userName"**)String name, @Query(**"password"**) String pwd);  
}

注：“userName”为参数的名字，会拼接到url上，name为变量名，会被实际的值替换掉，如：<http://httpbin.org/get?userName=name>，当我们调用login方法时会给name变量赋值，如下：

Retrofit retrofit = **new** Retrofit.Builder().baseUrl(**"http://httpbin.org/"**).build();  
ILogin login = retrofit.create(ILogin.**class**);  
Call<ResponseBody> call = login.login(**"EvenDai"**, **"123456"**);

习惯使用Map传参数的也可以这样：

**public interface** ILogin {  
 @GET(**"get"**)  
 Call<ResponseBody> login(@QueryMap Map<String, String> params);  
}

调用login方法的时候则需要map，如下：

Map<String, String> params = **new** HashMap<>();  
params.put(**"userName"**, **"EvenDai"**);  
params.put(**"password"**, **"123456"**);  
Call<ResponseBody> call = login.login(params);

5、使用注解@Path来生成动态URL路径，比如有这样的应用场景，有如下两个请求：  
 <http://www.abc.com/category/34/book?bookId=12345>

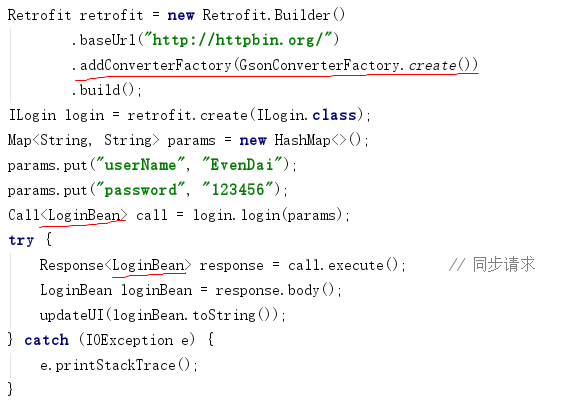
<http://www.abc.com/category/35/book?bookId=23458>

上面Url中34和35为分类的Id，书是有分类的，我们看到这两个Url很相似，如果分类有100个，难道我们要创建100个对应的接口类吗？使用@Path来生成动态URL，如下：

**public interface** IGetBook {  
 @GET(**"category/{categoryId}/book"**)  
 Call<ResponseBody> getBook(@Path(**"categoryId"**) **int** cId, @Query(**"bookId"**) **int** bookId);  
}

@Path(**"categoryId"**) **int** cId的意思是：把cId的值替换掉请求路径中的{categoryId}

6、转换器，上面的ILogin接口中，login方法的返回值是Call<RsponseBody>，泛型代表返回来的结果如何封装，如果我们希望返回来的结果封装为JavaBean，则可以把泛型写成JavaBean，如Call<LoginBean>,把一个String转换为一个JavaBean是有很多开源工具的，Retrofit给了我们选择的余地，通过方法可以设置使用什么工具，如使用Gson，则需要在依赖中设置：compile 'com.squareup.retrofit2:converter-gson:2.2.0'，在生成Retrofit实现的时候可以指定使用gson作为转换器，如下：



我们可以看到，请求返回的结果直接就是我们需要的JavaBean了。

### post请求

post请求和get请求差不多一样，只要把注解@GET换成@POST，@Query/QueryMap换成@Field/FieldMap，并增加表单类型注解@FormUrlEncoded，如下：

**public interface** ILogin {  
 @POST(**"post"**)  
 @FormUrlEncoded  
 Call<ResponseBody> login(@FieldMap Map<String, String> params);  
}

### 添加请求头

也是在接口的方法上加注解，如下：

**public interface** ILogin {  
 @Headers(**"User-Agent:ICBC"**)  
 @POST(**"post"**)  
 @FormUrlEncoded  
 Call<ResponseBody> login(@FieldMap Map<String, String> params);  
}

如需添加多个请求头需要使用大括号，如下：

**public interface** ILogin {  
 @Headers({**"User-Agent:ICBC"**,**"Haha:Hehe"**})  
 @POST(**"post"**)  
 @FormUrlEncoded  
 Call<ResponseBody> login(@FieldMap Map<String, String> params);  
}

### 关于请求URL

在2.2.0版本，我试过在BaseUrl后面，可以加“/”，也可以不加，在相对Url的前面也一样，BaseUrl和相对Url的“/”都加或者都不加，或者只加其中之一都是没有问题的，为了简洁，推荐都不加。

### 提交Json数据

**public interface** ILogin {  
 @POST(**"post"**)  
 Call<ResponseBody> login(@Body RequestBody requestBody);  
}

Retrofit retrofit = **new** Retrofit.Builder()  
 .baseUrl(**"http://httpbin.org"**)  
 .build();  
ILogin login = retrofit.create(ILogin.**class**);  
String json = **"{\"userName\":\"EvenDai\", \"password\":\"123456\"}"**;  
RequestBody requestBody =RequestBody.*create*(okhttp3.MediaType.*parse*(**"application/json;charset=utf-8"**),json);  
Call<ResponseBody> call = login.login(requestBody);

另一种用法：

@POST("users/new")

Call<User> createUser(@Body User user);

这个方式是使用@Body表示user对象会转换为json进行请求

### 关于Response类的一些用法

if (!response.isSuccessful()) {

System.out.println(call.request().url() + " failed: " + response.code());

return;

}

# 与RxJava配合使用

需要添加如下依赖：

