Retrofit：

基本用法：

1. 创建Retrofit：new Retrofit.Builder().baseUrl(baseUrl).build();//注意此处的Url必须以”/”结尾。
2. 创建请求接口 接口内定义请求函数，返回一个Call对象，用注解标注请求的目的和参数。如

public interface BlogService {

@GET("blog/{id}")//向baseUrl/blog/id地址发出了get请求，id的参数通过@Path(“id”)注解的参数获得（即此处的int id）.

Interface BlogService{

@GET(“blog/{id}”)

Call<ResponseBody> getFirstBlog(@Path("id") int id);}

而调用该接口时，需要通过retrofit.create(BlogService.class)来获得一个实例对象。

1. 调用接口 通过获得的接口实例对象获得一个Call对象，并调用call.enqueue(Callback callback)为其注入回调函数。

注解方法：



HTTP注解的用法：

@HTTP(method = "get", path = "blog/{id}", hasBody = false) （替代上面的get）



此处的Content-Type:application/x-www-form-urlencoded和Content-Type:multipart/form-data是请求头的Header部分，前者表示需要对ASCII以外的内容解码，后者表示需要将请求体部分分段上传。



Filed用于标注表单参数字段，将参数注入字段中，FiledMap则可直接根据Map对象参数将值填入键所表示的字段中。当然用之前需要@FormUrlEncoded注解，如

@POST("/form")

@FormUrlEncoded

Call<ResponseBody> testFormUrlEncoded1(@Field("username") String name, @Field("age") int age,@FiledMap(String,String) map);

而 Part可以标注的对象有三种：RequestBody、MultiPartBody.Part和其他任意类型值。其中除了已经内部标识的MultiPartBody.Part，其他对象都需要用Part标识目标字段。PartMap同样需要传入一个Map，Map的值部分必须为RequestBody对象，其他对象会被Convert转换。

/\*注：当Convert是Gson默认Convert时，其序列化时会给字符串增加一对双引号，需要另外处理。但直接传入RequestBody时不会\*/

例@POST("/form")

@Multipart

Call<ResponseBody> testFileUpload2(@PartMap Map<String, RequestBody> args, @Part(“name”) String name, @Part(“age”) RequestBody age, @Part MultipartBody.Part file);

推荐Path只用于与Url相关的注解，数据部分用Query解决，其实现方式与Filed相同。并且Query、Filed和Part都支持数组或者实现了Iterable接口的对象直接导入，其实现方式相当于传回用”&”连接的一系列相同字段的数据。

/\*接下来是Gson的转换\*/

上面的请求接口中Call<T>的泛型T除了ResponsBody之外，也可以是其他的对象，但是需要在Retrofit创建时设置相应的Convert，即addConverterFactory(Gson gson)方法。同时要对转化对象采用@Body注解，如Call<Result<Blog>> createBlog(@Body Blog blog)。如果需要自定义Convert的话，可以声明类实现Convert<F,T>接口，来重写各种向ResponseBody转换的过程或者ResponseBody转化为目标Object的方式，同时声明类继承Convert.Factory来创建Convert实现类，和获得各种标注信息以方便处理。注意：Convert的添加是有优先级的，当多个注册的Convert都能对某一类对象进行转换时，第一个注册的Convert才生效，并且GsonConvertFactory并不对某一种对象有效，当其处于高优先级时，会发生异常。

实际上Call<T>中的Call也是可以替换的，只要在类似的为Retrofit添加一个CallAdapter，方法为addCallAdapterFactory(CallAdapter adapter)。可以通过实现CallAdapter来自定义相应的对象适配。

其他Builder内的方法：

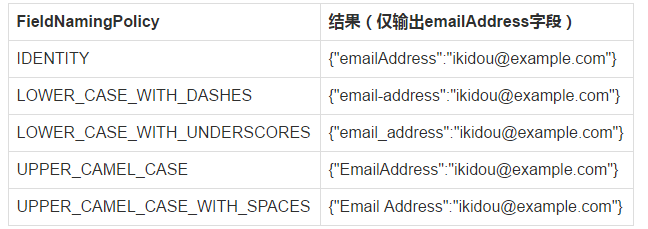




GSON：

Gson基础用法是利用fromJson(String jsonValue,Class Type)和toJson(Object object/BaseType value)方法进行反序列化和序列化。前者会将Json格式的字符串如{"name":"lazxy","age":21}转化为Type所表示的类对象形式，包括基本数据类型和自定义类（会将自定义类的成员变量和键值对一一对应），兼容一些字符串的格式强制转换；后者则会将输入的参数转化为相应的Json语句。

当Json语句中键的写法与Java的驼峰式写法可能不相符时，可以通过@SerializedName(“/\*name in other expressions\*/”) 或者@SerializedName(value=”/\*current name\*/”,alternate=”/\* name in other expressions \*/”)来给出其他的表达方式，这样也能保证转换正常。或者通过在GsonBuilder中setFieldNamingPolicy(FieldNamingPolicy. \*)方法转换成FieldNamingPolicy枚举类中指定的各种命名格式，具体格式如下：



还有一种转换方式为setFieldNamingStrategy(FieldNamingStrategy strategy)，通过自定义strategy中的translateName(Filed f )函数来进行转换。但后两种转换方式优先级都低于@SerializedName，同时存在时按后者策略执行。

Gson解析数组时，可以用数组和集合两种方式容纳解析结果，但集合的泛型在fromJson传入字节码List.class参数时会被擦除，故需要传入 new TypeToken<List<String>>(){}.getType()(注意此处TypeToken类的构造方法是protected，所以构造方式其实是通过一个匿名类隐式继承了TypeToken类实现调用其类中的方法)为对应的参数。当自定义对象的成员变量变化大部分较小时，可以通过设置对象的未定义泛型来单一地改变某个成员变量，在TypeToken中将需要的对象类型作为参数注入(如需要一个Topic对象，则设置TypeToken<BaseEntity<Topic>>.getType()为Json类型)，即可实现实体类的复用和拓展。

Gson的toJson重载函数toJson(String name,Appendable writer)可以通过传入StringBuffer、StreamWriter等对象实现自动流式转换；或者通过JsonWriter对象beginObject()/beginArray()开始转换事务，成对调用.name() .value()，然后endObject()/endArray()，最后flush()清除缓冲区，将JsonWriter对象传入toJson(String name,JsonWriter writer)实现手动流式转换。

Gson对象除了直接构造之外还可以通过GsonBuilder.create()创建，并且在这中间可以加入各种对它的配置从而实现不同的要求（例如加入serializeNulls可以使其对于值为空的键也同样输出”null”字符串）。

当Entity类中有成员变量不需要被序列化输出时，可以用 ①@Exposed(deserialize=true/false,serialize=true/false)来注解标注成员变量的序列化需求，并在GsonBuilder创建Gson的链中加入excludeFieldsWithoutExposeAnnotation()。②或者以版本为界定，利用@Since(double value) 和 @Until(double value)来标识相应的成员变量，只有满足大于Since或者小于Until条件版本的成员变量能被输出(条件可叠加)。设置版本的语句也在GsonBuilder创建链中，为setVersion(double value)函数。③还有一种方式为以其修饰符为限定，限定参数来自java.lang.reflect.Modifier类，设置条件语句同样处于GsonBuilder链中：excludeFieldsWithModifiers(Modifier.FINAL, Modifier.STATIC, Modifier.PRIVATE)//限定final static private 修饰符修饰的成员变量不输出。④最后一种方式是通过addSerializationExclusionStrategy 和addDeserializationExclusionStrategy函数来自定义排除策略，这两个函数的参数都是一个ExclusionStrategy对象，可以通过匿名类的方式重写相关的策略函数，包括按字段判断和按类判断等策略。

除了用上面提到的注解和各种其他手段修改序列化和反序列化的过程时，还可以用TypeAdapter全程接管整个过程。该类有两个主要的重载方法write(JsonWriter writer,T value)和read(JsonReader reader).前者的转换过程类似手动流式转换的过程，键值对注入过程在beginObject和endObject之间成对出现；后者类似Android的pull解析过程，采用switch判断当前的键名然后解析出相应的值。TypeAdapter使用需要在Gson创建时注册，即在链中加入registerTypeAdapter(Class Object.class, TypeAdapter<Object> adpater)。也可以将前面的参数TypeAdapter换成JsonSerializer 或JsonDeserializer，这样就只会接管其中一个过程。但其接管的数据类型必须为泛型规定的类型，不能为其子类型。如果需要指定一类对象，则可以用**r**egisterTypeHierarchyAdapter()通过传入父类来指定一类相似对象。

为了简化TypeAdapter的注册过程，可以在需要被解析的实体类前添加@JsonAdapter(TypeAdapter ObjectAdapter)来声明其适配器，且其优先级高于注册优先级与单独的字段注解。

练习项目：用*Dribbble*的API做一个第三方应用。

API 整理与记录

baseUrl：api.dribbble.com/v1/

应用申请信息

Client ID

d6b8c242643ae825c0056550d4f78a65065dacac0e79235a71a7a51ee069b7ba

Client Secret

ae11777201efe6ac96c95a93c1d853fb0d407763695781acbd195c91de41b08e

Client Access Token

5acfc8ea6f4a554c438a8ff67def6ec948122148bb79e7d461840f0e80c9a2b6

当用POST、DELETE、PUT方式传参时，header必须设为application/x-www-form-urlencoded

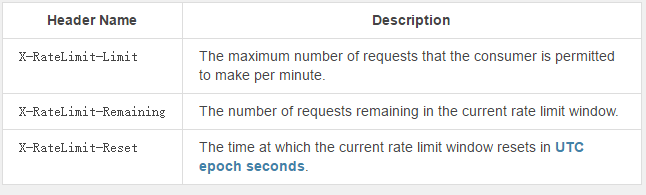
密钥登陆方式：

https://api.dribbble.com/v1/user?access\_token=OAUTH\_TOKEN

单页条目限制为100条，省略页数将返回第一页内容：

https://api.dribbble.com/v1/user/followers?page=2&per\_page=100

每分钟发出请求次数最多为60次，登录状态下一个账号每天最多发出1440次请求，未登陆状态下最多10000次请求。可以通过请求头得到次数限制情况，如下：



（最后一个时间为1970至今的秒数）

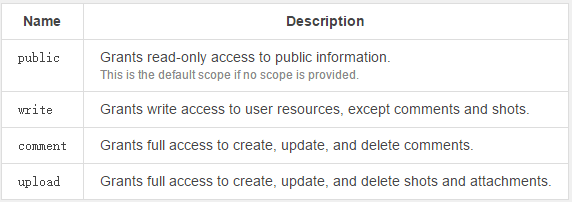
重复加载校验：

If-None-Match : 请求url后加上s-H 'If-None-Match: "e612e16d3c4d113573edb015d8eac1d5"'

（其中Match后为MD5码），则会根据MD5码对当前页面信息与服务器信息进行比对，如果有新的信息则返回状态码200并返回新的ETag（被请求变量实体），否则返回304。

If-Modifide-Since：请求url后加上 上一次请求Last-Modified的时间，如 Sat, 22 Feb 2014 17:10:33 GMT，则服务器会将该时间与当前信息的修改时间进行对比，同样若改动则返回状态码200与新的ETag，否则返回304。该机制有利于减小服务器压力和便于本地加载。

可以根据传递的参数确定权限申请范围，主要有以下几种：



用法：https://dribbble.com/oauth/authorize?

client\_id=...&

scope=public+write

项目实现过程：

1. 确定身份验证机制，确定数据接收模式与方式，建立Api类并封装，定义Json解析模式（如果需要的话）。
2. 建立MVP基础框架，确定基类的公共功能。
3. 完成各个界面的显示和操作。顺序为MainActivity、LoginActivity、ShotFragment、BucketFragment、TeamFragment（如果时间允许的话）。

功能拆分实现：

1. Login

Dribbble不支持客户端直接登陆访问方式，故只能通过WebView 登陆Oath验证的方式来获得Token。实现方式主要为通过WebView的shouldOverrideUrlLoading(WebView view,Url url)函数来对传入的url进行解析。用户在登陆过程中会看到两个页面，第一个页面为账号密码输入界面，进入该界面前以及输入完成并且点击相应登陆按钮后都会返回一个url值，这些值可以忽略，而值得注意的是点击授权按钮后会返回一段值，格式为ApiConstants.Url.*REDIRECT\_URL* ?code=/\*code value\*/&state= ApiConstants.ParamValue.STATE.而中间的codeValue就是临时授权码，可以通过该授权码调用getToken即可获得Token。

（Dribbble网页端似乎默认保存所有登陆信息，第一次登陆后其登陆界面会留下相应的Cookie，如果第一次授权被中断，再第二次登陆时会直接跳到授权界面。如果曾经授权成功，则再次登陆时会直接返回Code而省略授权过程。）

另外，参考例程里当获取到Token后又没有成功获取到User信息时，会造成初始化失败无法成功进入MainActivity，这个问题觉得暂时可以用异常捕捉然后重新请求用户信息的方式二次处理。

/\*关于WebView\*/

/\*WebView 在这里有几个地方可以注意：一、shouldOverrideUrlLoading函数会返回一个boolean值，其值为true时表示将重定向至自定义的url值（包括直接取消跳转），false时为交由WebView自动执行，等同于直接调用loadUrl(url)。二、setWebChromeClient函数，设置一个WebChromeClient对象，其包括onCloseWindow(关闭WebView) 、onCreateWindow() 、onJsAlert (WebView上alert是弹不出来东西的，需要定制你的WebChromeClient处理弹出) 、onJsPrompt 、onJsConfirm 、onProgressChanged 、nReceivedIcon 、onReceivedTitle等可复写方法，用于渲染WebView界面和处理Js事件，最常用的大概是监听onProgressChanged显示加载进度条。三、setWebViewClient函数，设置一个WebViewClient对下岗，其包括onLoadResource 、onPageStart 、onPageFinish 、onReceiveError 、onReceivedHttpAuthRequest 等可复写函数，用于处理消息和各种事件，根据函数名看起来更像对WebView生命周期的监听处理。\*/

https://dribbble.com/oauth/authorize?state=hunter&scope=public%20write%20comment%20upload&redirect\_uri=http://lhunter.org/&client\_id=e9976e732df8ed56f928882d23be3b47543a4e0b8c3a6a05fa0af7d7a7d3b34a