

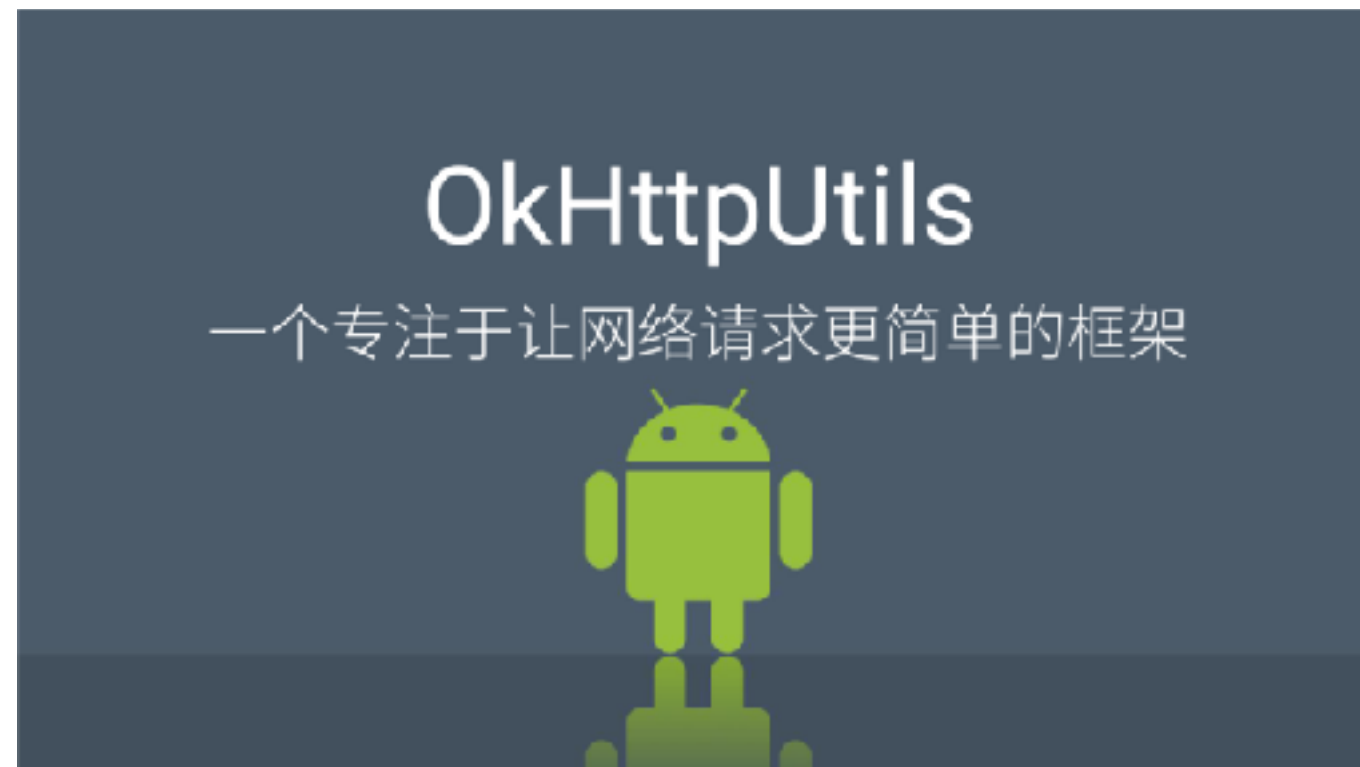
OkHttpUtils一个专注于让网络请求更简单的框架



廖子尧 (/u/d82085ac0ace) [+ 关注](#)

2016.06.21 11:50* 字数 4085 阅读 65054 评论 95 喜欢 250 赞赏 1

(/u/d82085ac0ace)



一句话概括，OkHttpUtils一个专注于让网络请求更简单的网络请求框架，对于任何形式的网络请求只需要一行代码。

- 项目地址：<https://github.com/jeasonlzy/okhttp-OkGo> (<https://link.jianshu.com?t=https://github.com/jeasonlzy/okhttp-OkGo>)
- 联系方式：QQ群 (489873144)
- 如果你想直接运行apk看效果，[点击这里下载](#)：okhttputils_v1.8.1.apk (https://link.jianshu.com?t=http://7xss53.com1.z0.glb.clouddn.com/okhttputils/okhttputils_v1.8.1.apk)



- 其中Demo中用到的图片选择是我的另一个开源项目，完全仿微信的图片选择库，自带 矩形图片裁剪 和 圆形图片裁剪 功能，有需要的可以去下载使用，附上地址：
<https://github.com/jeasonlzy/ImagePicker> (<https://link.jianshu.com?t=https://github.com/jeasonlzy/ImagePicker>)
- 其中的九宫格控件也是我的开源项目,类似QQ空间，微信朋友圈，微博主页等，展示图片的九宫格控件，自动根据图片的数量确定图片大小和控件大小，使用Adapter模式设置图片，对外提供接口回调，使用接口加载图片,支持任意的图片加载框架,如 Glide,ImageLoader,Fresco,xUtils3,Picasso 等，支持点击图片全屏预览大图。附上地址：<https://github.com/jeasonlzy/NineGridView> (<https://link.jianshu.com?t=https://github.com/jeasonlzy/NineGridView>)

1.OkHttpUtils 目前支持

- 一般的 get,post,put,delete,head,options请求
- 基于Post的大文本数据上传
- 多文件和多参数统一的表单上传
- 支持一个key上传一个文件，也可以一个Key上传多个文件
- 大文件下载和下载进度回调
- 大文件上传和上传进度回调
- 支持cookie的内存存储和持久化存储，支持传递自定义cookie
- 支持304缓存协议，扩展四种本地缓存模式,并且支持缓存时间控制
- 支持301、302重定向
- 支持链式调用
- 支持可信证书和自签名证书的https的访问,支持双向认证
- 支持根据Tag取消请求
- 支持自定义泛型Callback，自动根据泛型返回对象

2.OkHttpServer 扩展功能

2.1 统一的文件下载管理(DownloadManager)：



- 结合OkHttpUtils 的request进行网络请求,支持与OkHttpUtils 保持相同的全局公共参数,同时支持请求传递参数
- 支持断点下载 , 支持突然断网,强杀进程后,断点依然有效
- 支持 下载 暂停 等待 停止 出错 完成 六种下载状态
- 所有下载任务按照taskKey区分,切记不同的任务必须使用不一样的key,否者断点会发生覆盖
- 相同的下载url地址如果使用不一样的taskKey,也会认为是两个下载任务
- 默认同时下载数量为3个 , 默认下载路径 `/storage/emulated/0/download` , 下载路径和下载数量都可以在代码中配置
- 下载文件名可以自己定义,也可以不传,框架自动解析响应头或者url地址获得文件名,如果都没获取到,使用default作为文件名
- 下载管理使用了服务提高线程优先级 , 避免后台下载时被系统回收

2.2 统一的文件上传管理(UploadManager)

- 结合OkHttpUtils 的request进行网络请求,支持与OkHttpUtils 保持相同的全局公共参数,同时支持请求传递参数
- 上传只能使用 `Post` , `Put` , `Delete` , `Options` 这四种请求,不支持 `Get` , `Head`
- 该上传管理为简单管理 , 不支持断点续传或分片上传 , 只是简单的将所有上传任务使用线程池进行了统一管理
- 默认同时上传数量为1个,该数列可以在代码中配置修改
- 由于断点分片上传的技术需要大量的服务端代码配合 , 同时也会极大的增加客户端代码量 , 所以综合考虑 , 该框架不做实现。如果确实有特殊需要 , 可以自己做扩展。

项目的效果图如下 :





项目主页



请求详细信息

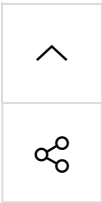


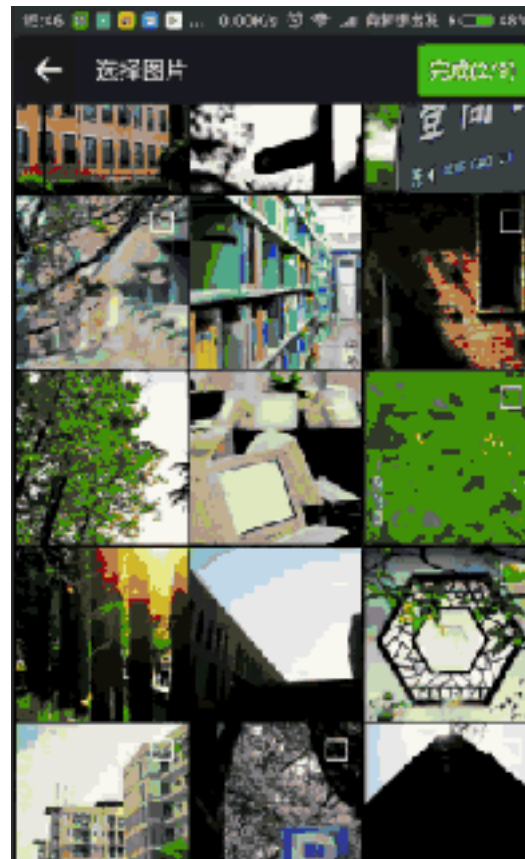


下载管理列表页



下载管理详情页





上传管理和图片选择

3.如何选择网络框架

说了这么多功能，我们来看看为什么要使用OkHttpUtils这个框架。
首先目前主流的几个网络框架

- android-async-http
- xUtils
- volley
- retrofit
- okhttp

在此引用知乎上Stay Zhang的回答：

我们来先说一个常识性的错误：volley, retrofit, android-async-http 帮你封装了具体的请求，线程切换以及数据转换。而OkHttp 是基于http协议封装的一套请求客户端，虽然它也可以开线程，但根本上它更偏向真正的请求，跟HttpClient, HttpURLConnection的职责是一样的。



所以不要混淆。

-----以下纯个人主观见解

首先，我想即使你单纯使用OkHttp，还是会再包一层的，这样就等价于Volley之流的框架，只是封装的好与坏而已。

android-async-http内部实现是基于HttpClient, 想必你肯定知道6.0之后HttpClient是不是系统自带的了，不过它在最近的更新中将HttpClient的所有代码copy了一份进来，所以还能使用。

Volley是官方出的，volley在设计的时候是将具体的请求客户端做了下封装：HurlStack，也就是说可以支持URLConnection, HttpClient, OkHttp，相当于模版模式吧，这样解耦还是非常方便的，可以随意切换，如果你之前使用过Volley，并习惯使用，那直接写个OkHttp扩展就行了。

Retrofit因为也是square出的，所以大家可能对它更崇拜些。Retrofit的跟Volley是一个套路，但解耦的更彻底:比方说通过注解来配置请求参数，通过工厂来生成CallAdapter，Converter，你可以使用不同的请求适配器(CallAdapter), 比方说RxJava，Java8, Guava。你可以使用不同的反序列化工具(Converter)，比方说json, protobuff, xml, moshi等等。关键是要用好这个框架，最好是和RxJava联用，否者和普通的网络框架也没什么区别，而对于RxJava，特别team人数多的情况下，总得有个完全精通的吧，万一掉坑里了呢。。。

4.OkHttpUtils的优势

- 优势一：性能高，专注于简单易用的网络请求，使用主流的okhttp进行封装，对于okhttp大家都知道，在Android4.4的源码中可以看到HttpURLConnection已经替换成OkHttp实现了，并且支持HTTP2/SPDY黑科技，支持socket自动选择最好路线，并支持自动重连，拥有自动维护的socket连接池，减少握手次数，拥有队列线程池，轻松写并发。
- 优势二：特有的网络缓存模式，是大多数网络框架所不具备的，说一个应用场景，老板说我们的app不仅需要在有网的情况下展示最新的网络数据，还要在没网的情况下使用缓存数据，这时候是不是项目中出现了大量的代码判断当前网络状况，根据不同的状态保存不同的数据，然后决定是否使用缓存。细想一下，这是个通用的写法，于是OkHttpUtils提供了四种缓存模式，让你不用关心缓存的实现，而专注于数据的处理。（具体缓存的使用方法请看最后第四章节）。
- 优势三：方便易用的扩展接口，可以添加全局的公共参数，全局拦截器，全局超时时间，更可以对单个请求定制拦截器，超时时间，请求参数修改等等，在使用上更是方



便，原生支持的链式调用让你的请求更加清晰。

- 优势四：强大的Cookie保持策略，我们知道在客户端对cookie的获取是个不太简单的事情，特别是还要处理cookie的过期时间，持久化策略等等，OkHttpUtils帮你彻底解决Cookie的难题，默认拥有内存存储和持久化存储两种实现，cookie全程自动管理，并且提供了额外的addCookie方式，允许介入到自动管理的过程中，添加你想创建的任何cookie。

所以说这么多啦，选最适合项目的，选大多数人选择的，选简单易用的，就这么个标准，而OkHttpUtils正是在这种情况下诞生啦！！

5.使用方法

- 对于Android Studio的用户，可以选择添加:

```
compile 'com.lzy.net:okhttputils:1.8.1' //可以单独使用，不需要依赖下方的扩展包
compile 'com.lzy.net:okhttpserver:1.0.3' //扩展了下载管理和上传管理，根据需要添加

compile 'com.lzy.net:okhttputils:+' //版本号使用 + 可以自动引用最新版
compile 'com.lzy.net:okhttpserver:+' //版本号使用 + 可以自动引用最新版
```

- 为了方便大家使用，更加通俗的理解http的网络协议，建议做网络请求的时候，对每个请求抓包后查看请求信息和响应信息。
- 如果是 Windows 操作系统，可以使用 Fiddler 对手机的请求进行抓包查看。
- 如果是 Mac OS 操作系统，可以使用 Charles 对手机的请求进行抓包查看。
- 具体的下载地址和抓包配置方法，我这就不提供了，请自行百度或谷歌。

6.使用注意事项

- okhttputils 使用的 okhttp 的版本是最新的 3.4.1 版本，和以前的 2.x 的版本可能会存在冲突。
- okhttpserver 是对 okhttputils 的扩展，统一了下载管理和上传管理，对项目有需要做统一下载的可以考虑使用该扩展，不需要的可以直接使用 okhttputils 即可。
- 对于缓存模式使用，需要与返回对象相关的所有 javaBean 必须实现 Serializable 接口，否则会报 NotSerializableException 。



- 使用缓存时，如果不指定 `cacheKey` ，默认是用url带参数的全路径名为 `cacheKey` 。
- 使用该网络框架时，必须要在 `Application` 中做初始化 `OkHttpUtils.init(this);` 。

一、全局配置

一般在 `Aplication` ，或者基类中，只需要调用一次即可，可以配置调试开关，全局的超时时间，公共的请求头和请求参数等信息，所有的请求参数都支持中文，

```
@Override
public void onCreate() {
    super.onCreate();

    HttpHeaders headers = new HttpHeaders();
    headers.put("commonHeaderKey1", "commonHeaderValue1");    //所有的 header 都不支持中文
    headers.put("commonHeaderKey2", "commonHeaderValue2");
    HttpParams params = new HttpParams();
    params.put("commonParamsKey1", "commonParamsValue1");    //所有的 params 都支持中文
    params.put("commonParamsKey2", "这里支持中文参数");

    //必须调用初始化
    OkHttpUtils.init(this);
    //以下都不是必须的，根据需要自行选择
    OkHttpUtils.getInstance().//
        .debug("OkHttpUtils")                                //是否打开debug
        .setConnectTimeout(OkHttpUtils.DEFAULT_MILLISECONDS) //全局的连接超时时间
        .setReadTimeOut(OkHttpUtils.DEFAULT_MILLISECONDS)   //全局的读取超时时间
        .setWriteTimeOut(OkHttpUtils.DEFAULT_MILLISECONDS)  //全局的写入超时时间
        //.setCookieStore(new MemoryCookieStore())              //cookie使用内存存储
        //.setCookieStore(new PersistentCookieStore())           //cookie持久化存储
        .addCommonHeaders(headers)                             //设置全局公共请求头
        .addCommonParams(params);                               //设置全局公共请求参数
}
```

二、普通请求

1.基本的网络请求

```
OkHttpUtils.get(Urls.URL_METHOD)    // 请求方式和请求url
    .tag(this)                       // 请求的 tag，主要用于取消对应的请求
    .cacheKey("cacheKey")             // 设置当前请求的缓存key,建议每个不同功能的请求设置一个
    .cacheMode(CacheMode.DEFAULT)     // 缓存模式，详细请看缓存介绍
    .execute(new JsonCallback<RequestInfo>(RequestInfo.class) {
        @Override
        public void onResponse(boolean isFromCache, RequestInfo requestInfo, Request request) {
            // requestInfo 对象即为所需要的结果对象
        }
    });
```

2.请求 Bitmap 对象

```
OkHttpUtils.get(Urls.URL_IMAGE)//
    .tag(this)//
    .execute(new BitmapCallback() {
        @Override
        public void onResponse(boolean isFromCache, Bitmap bitmap, Request request, @Nullable Response response) {
            // bitmap 即为返回的图片数据
        }
    });
```

3.请求 文件下载

```
OkHttpUtils.get(Urls.URL_DOWNLOAD)//
    .tag(this)//
    .execute(new FileCallback("/sdcard/temp/", "file.jpg") { //文件下载时，需要指定下载的文件名
        @Override
        public void onResponse(boolean isFromCache, File file, Request request, @Nullable Response response) {
            // file 即为文件数据，文件保存在指定目录
        }
    });
```

4.普通Post，直接上传String类型的文本

一般此种用法用于与服务器约定的数据格式，当使用该方法时，params中的参数设置是无效的，所有参数均需要通过需要上传的文本中指定，此外，额外指定的header参数仍然保持有效。

```
OkHttpUtils.post(Urls.URL_TEXT_UPLOAD)//
    .tag(this)//
    .postString("这是要上传的长文本数据! ")//
    .execute(new StringCallback() {
        @Override
        public void onResponse(boolean isFromCache, String s, Request request, @Nullable Response response) {
            //上传成功
        }
    });
```

5.普通Post，直接上传Json类型的文本

该方法与postString没有本质区别，只是数据格式是json,一般来说，需要自己创建一个实体bean或者一个map，把需要的参数设置进去，然后通过三方的Gson或者fastjson转换成json字符串，最后直接使用该方法提交到服务器。



```
OkHttpUtils.post(Urls.URL_TEXT_UPLOAD)//
    .tag(this)//
    .postJson("{\"des\": \"这里面要写标准的json格式数据\"}");//
    .execute(new StringCallback() {
        @Override
        public void onResponse(boolean isFromCache, String s, Request request, @Nullable Res
            //上传成功
        }
    });
```

6.请求功能的所有配置讲解

以下代码包含了以下内容：

- 一次普通请求所有能配置的参数，真实使用时不需要配置这么多，按自己的需要选择性的使用即可
- 多文件和多参数的表单上传，同时支持进度监听
- 自签名网站https的访问，调用 `setCertificates` 方法即可
- 为单个请求设置超时，比如涉及到文件的需要设置读写等待时间多一点。
- Cookie一般情况下只需要在初始化的时候调用 `setCookieStore` 即可实现cookie的自动管理，如果特殊业务需要，需要手动额外向服务器传递自定义的cookie，可以在每次请求的时候调用 `addCookie` 方法，该方法提供了3个重载形式，可以根据自己的需要选择使用。



```

OkHttpUtils.get(Urls.URL_METHOD) // 请求方式和请求url, get请求不需要拼接参数, 支持get, post, put
    .tag(this) // 请求的 tag, 主要用于取消对应的请求
    .connTimeOut(10000) // 设置当前请求的连接超时时间
    .readTimeOut(10000) // 设置当前请求的读取超时时间
    .writeTimeOut(10000) // 设置当前请求的写入超时时间
    .cacheKey("cacheKey") // 设置当前请求的缓存key, 建议每个不同功能的请求设置一个
    .cacheMode(CacheMode.FIRST_CACHE_THEN_REQUEST) // 缓存模式, 详细请看第四部分, 缓存介绍
    .setCertificates(getAssets().open("srca.cer")) // 自签名https的证书, 可变参数, 可以设置多个
    .addInterceptor(interceptor) // 添加自定义拦截器
    .headers("header1", "headerValue1") // 添加请求头参数
    .headers("header2", "headerValue2") // 支持多请求头参数同时添加
    .params("param1", "paramValue1") // 添加请求参数
    .params("param2", "paramValue2") // 支持多请求参数同时添加
    .params("file1", new File("filepath1")) // 可以添加文件上传
    .params("file2", new File("filepath2")) // 支持多文件同时添加上传
    .addUrlParams("key", List<String> values) //这里支持一个key传多个参数
    .addFileParams("key", List<File> files) //这里支持一个key传多个文件
    .addFileWrapperParams("key", List<HttpParams.FileWrapper> fileWrappers) //这里支持一个key传多个文件
    .addCookie("aaa", "bbb") // 这里可以传递自己想传的Cookie
    .addCookie(cookie) // 可以自己构建cookie
    .addCookies(cookies) // 可以一次传递批量的cookie
    //这里给出的泛型为 RequestInfo, 同时传递一个泛型的 class对象, 即可自动将数据结果转成对象返回
    .execute(new DialogCallback<RequestInfo>(this, RequestInfo.class) {
        @Override
        public void onBefore(BaseRequest request) {
            // UI线程 请求网络之前调用
            // 可以显示对话框, 添加/修改/移除 请求参数
        }

        @Override
        public RequestInfo parseNetworkResponse(Response response) throws Exception{
            // 子线程, 可以做耗时操作
            // 根据传递进来的 response 对象, 把数据解析成需要的 RequestInfo 类型并返回
            // 可以根据自己的需要, 抛出异常, 在onError中处理
            return null;
        }

        @Override
        public void onResponse(boolean isFromCache, RequestInfo requestInfo, Request request) {
            // UI 线程, 请求成功后回调
            // isFromCache 表示当前回调是否来自于缓存
            // requestInfo 返回泛型约定的实体类型参数
            // request 本次网络的请求信息, 如果需要查看请求头或请求参数可以从此对象获取
            // response 本次网络访问的结果对象, 包含了响应头, 响应码等, 如果数据来自于缓存, 该对象为null
        }

        @Override
        public void onError(boolean isFromCache, Call call, @Nullable Response response, @Nullable RequestInfo requestInfo) {
            // UI 线程, 请求失败后回调
            // isFromCache 表示当前回调是否来自于缓存
            // call 本次网络的请求对象, 可以根据该对象拿到 request
            // response 本次网络访问的结果对象, 包含了响应头, 响应码等, 如果网络异常 或者数据来源于网络
            // e 本次网络访问的异常信息, 如果服务器内部发生了错误, 响应码为 400~599之间
        }

        @Override
        public void onAfter(boolean isFromCache, @Nullable RequestInfo requestInfo, Call call) {
            // UI 线程, 请求结束后回调, 无论网络请求成功还是失败, 都会调用, 可以用于关闭显示对话框
        }
    })

```



```
        // isFromCache 表示当前回调是否来自于缓存
        // requestInfo 返回泛型约定的实体类型参数，如果网络请求失败，该对象为 null
        // call        本次网络的请求对象，可以根据该对象拿到 request
        // response     本次网络访问的结果对象，包含了响应头，响应码等，如果网络异常 或者数据来
        // e            本次网络访问的异常信息，如果服务器内部发生了错误，响应码为 400~599之间
    }

    @Override
    public void upProgress(long currentSize, long totalSize, float progress, long networkSpeed) {
        // UI 线程，文件上传过程中回调，只有请求方式包含请求体才回调（GET,HEAD不会回调）
        // currentSize 当前上传的大小（单位字节）
        // totalSize   需要上传的总大小（单位字节）
        // progress    当前上传的进度，范围 0.0f ~ 1.0f
        // networkSpeed 当前上传的网速（单位秒）
    }

    @Override
    public void downloadProgress(long currentSize, long totalSize, float progress, long networkSpeed) {
        // UI 线程，文件下载过程中回调
        //参数含义同 上传相同
    }
});
```

7.取消请求

每个请求前都设置了一个参数 tag ，取消则通过 `OkHttpUtils.cancel(tag)` 执行。
例如：在Activity中，当Activity销毁取消请求，可以在onDestory里面统一取消。

```
@Override
protected void onDestroy() {
    super.onDestroy();

    //根据 Tag 取消请求
    OkHttpUtils.getInstance().cancelTag(this);
}
```

8.同步的请求

`execute`方法不传入callback即为同步的请求，返回 `Response` 对象，需要自己解析

```
Response response = OkHttpUtils.get("http://www.baidu.com")//
    .tag(this)//
    .headers("aaa", "111")//
    .params("bbb", "222")
    .execute();
```

三、自定义CallBack使用

目前内部提供的包含 `AbsCallback` , `StringCallBack` , `BitmapCallback` , `FileCallBack` ,可以根据自己的需求去自定义`Callback`

- `AbsCallback` : 所有回调的父类，抽象类
- `StringCallBack` : 如果返回值类型是纯文本数据，即可使用该回调
- `BitmapCallback` : 如果请求的是图片数据，则可以使用该回调
- `FileCallBack` : 如果要做文件下载，则必须使用该回调，内部封装了关于文件下载进度回调的方法

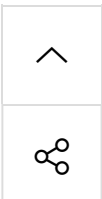
该网络框架的核心使用方法即为 `Callback` 的继承使用，详细请看 Demo 源码中 `callback` 包下的代码。

因为不同的项目需求，可能对数据格式进行了不同的封装，于是在 Demo 中的进行了详细的代码示例，以下是详细介绍：

- `CommonCallback` :继承自 `AbsCallback` ,主要作用是做全局共同请求参数的添加，同样也可以在第一步全局配置的时候设置，效果一样。
- `EncryptCallback` : 继承自 `CommonCallback` ,主要功能是做 `Url` 参数加密，对每个请求的参数进行编码，防止拦截数据包，篡改数据。
- `JsonCallback` : 继承自 `EncryptCallback` ,一般来说，服务器返回的响应码都包含 `code` , `msg` , `data` 三部分，在此根据自己的业务需要完成相应的逻辑判断，并对数据进行解析，可以使用 `Gson` 或者 `fastjson` ,将解析的对象返回。
- `DialogCallback` : 继承自 `JsonCallback` ,对需要在网络请求的时候显示对话框，使用该回调。
- `StringDialogCallback` : 继承自 `EncryptCallback` ,如果网络返回的数据只是纯文本，使用该回调
- `BitmapDialogCallback` : 继承自 `BitmapCallback` ,如果网络返回的是`Bitmap`对象，使用该回调
- `DownloadFileCallBack` : 继承自 `FileCallback` ,如果需要做文件下载，使用该回调

以上基本是包含了大部分的业务逻辑，具体情况请参照demo示例，根据业务需求修改！

四、缓存的使用



使用缓存前，必须让缓存的数据 javaBean 对象实现 Serializable 接口，否则会报 NotSerializableException。

因为缓存的原理是将对象序列化后直接写入 数据库中，如果不实现 Serializable 接口，会导致对象无法序列化，进而无法写入到数据库中，也就达不到缓存的效果。

目前提供了四种 CacheMode 缓存模式

- DEFAULT：按照HTTP协议的默认缓存规则，例如有304响应头时缓存
- REQUEST_FAILED_READ_CACHE：先请求网络，如果请求网络失败，则读取缓存，如果读取缓存失败，本次请求失败。该缓存模式的使用，会根据实际情况，导致 onResponse，onError，onAfter 三个方法调用不只一次，具体请在三个方法返回的参数中进行判断。
- IF_NONE_CACHE_REQUEST：如果缓存不存在才请求网络，否则使用缓存。
- FIRST_CACHE_THEN_REQUEST：先使用缓存，不管是否存在，仍然请求网络，如果网络顺利，会导致 onResponse 方法执行两次，第一次 isFromCache 为true，第二次 isFromCache 为false。使用时根据实际情况，对 onResponse，onError，onAfter 三个方法进行具体判断。

无论对于哪种缓存模式，都可以指定一个 cacheKey，建议针对不同需要缓存的页面设置不同的 cacheKey，如果相同，会导致数据覆盖。

📖 日记本 (/nb/4170641)

举报文章 © 著作权归作者所有



廖子尧 (/u/d82085ac0ace)

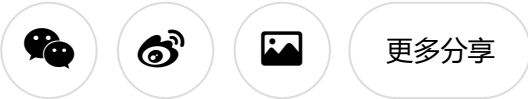
写了 11793 字，被 754 人关注，获得了 564 个喜欢

(/u/d82085ac0ace)

+ 关注

欢迎访问我的github地址：<https://github.com/jeasonlzy?tab=repositories>








(http://cwb.assets.jianshu.io/notes/images/4485317/weibo/image_d46aab5459aa.jp



(/apps/download?utm_source=nbc)

被以下专题收入，发现更多相似内容

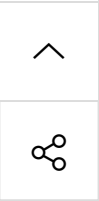
-  Android (/c/8a3f811d7ce8?utm_source=desktop&utm_medium=notes-included-collection)
-  Android开发 (/c/0dc880a2c73c?utm_source=desktop&utm_medium=notes-included-collection)
-  Android... (/c/c8ec4ae97839?utm_source=desktop&utm_medium=notes-included-collection)
-  安卓资源收集 (/c/c2955a70fdd6?utm_source=desktop&utm_medium=notes-included-collection)
-  Android开发 (/c/d1591c322c89?utm_source=desktop&utm_medium=notes-included-collection)
-  MobDevG... (/c/1f37d896b5de?utm_source=desktop&utm_medium=notes-included-collection)
-  Android知识 (/c/3fde3b545a35?utm_source=desktop&utm_medium=notes-included-collection)

展开更多 ∨


(/p/6aa5cb272514?
utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

OkGo，一个专注于让网络请求更简单的框架，与RxJava完美结合，比Retr...

OkGo - OkHttpUtils-2.0.0升级后改名 OkGo,全新完美支持RxJava 项目地址：



https://github.com/jeasonlzy , 欢迎star , 欢迎issue 该库是封装了okhttp的网...

 廖子尧 (/u/d82085ac0ace?)



utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/61d3eaecc7ca?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

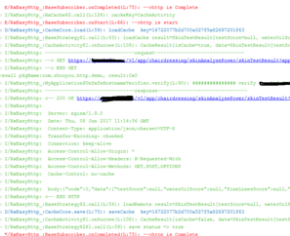
NoHttpRxUtils网络通讯框架让使用者更专注于项目业务，而非客户端与服...

由于NoHttpRxUtils是通过RxJava对NoHttp网络框架操作进行一系列封装。首先对RxJava和NoHttp网络框架做一个简介 RxJava框架是什么? RxJava是响应式程序设计的一种实现。 在响应式程序设计中，当数据到...

 木子v大可 (/u/3d97ee09869c?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/6c341916f477?)



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)

RxEasyHttp一款基于RxJava2+Retrofit2实现简单易用的网络请求框架（完...


github源码地址：https://github.com/zhou-you/RxEasyHttp RxEasyHttp 本库是一款基于RxJava2+Retrofit2实现简单易用的网络请求框架，结合android平台特性的网络封装库,采用api链式调用一点到底,集成co...

 zhou_you (/u/4060186e538c?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

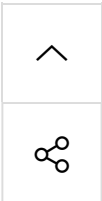
Android - 收藏集 - 掘金 (/p/5ad013eb5364?utm_campaign=maleskine&...

用两张图告诉你，为什么你的 App 会卡顿? - Android - 掘金Cover 有什么料？从这篇文章中你能获得这些料：知道setContentView()之后发生了什么？ ... Android 获取 View 宽高的常用正确方式，避免为零 - 掘金...


 掘金官方 (/u/5fc9b6410f4f?)

utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

Android - 收藏集 (/p/dad51f6c9c4d?utm_campaign=maleskine&utm_c...




用两张图告诉你，为什么你的 App 会卡顿? - Android - 掘金 Cover 有什么料？从这篇文章中你能获得这些料：知道setContentView()之后发生了什么？ ... Android 获取 View 宽高的常用正确方式，避免为零 - 掘金...

 passiontim (/u/e946d18f163c?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


高一学生好大胆 拍摄电视不简单 (/p/7779b39f14c6?utm_campaign=male...

请关注个人微信公众号：泸中陈健 高一学生好大胆 拍摄电视不简单 2017年5月，按照成都七中网校第三届“野草之旅”大型散文阅读系列活动的要求，泸定中学高2019届1班的同学们勇于创新、敢于尝试、大胆学...

 泸中陈健 (/u/939f1d0af6a4?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

Vue-3D-Model：用简单的方式来展示三维模型 (/p/c093ff00ea1b?utm_ca...

一个用来展示三维模型的Vue组件，用最简单的方式在网页中展示三维模型，解决模型视角控制、鼠标事件等一系列问题。项目地址：https://github.com/hujiulong/vue-3d-model 为什么做这个组件 我经常听到前端...

 hujiulong (/u/922bdc1251d1?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)


(/p/d714bcfd722c?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation)


企业直播的爆款，你还没把乐直播添加到购物车？ (/p/d714bcfd722c?utm_...

最近“穿”乐直播的企业太多，一时间成了企业直播的爆款，这不，昨天就有3家合伙人“撞衫”了！乐直播的技术部忙到“起飞”，上海分公司同样应接不暇，这样的直播盛宴可是千万不能错过，还不快来“饱餐一顿”呐！ ...

 lezhibo (/u/e0ebba8ad8a8?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)

沉默 (/p/e4208f08b922?utm_campaign=maleskine&utm_content=note...

善意的谎言 不被我说破 你藏起脸上的伤悲 想不让我难过 我走到你面前 什么都不说 转过身 彼此沉默

 夏有千阳 (/u/c45d64e94f8e?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendation)