**用法**

* Android Studio

compile 'com.zhy:okhttputils:2.6.2'

* Eclipse

下载最新jar:[okhttputils-2\_6\_2.jar](https://github.com/hongyangAndroid/okhttputils/blob/master/okhttputils-2_6_2.jar?raw=true)

注：需要同时导入okhttp和okio的jar，下载见：<https://github.com/square/okhttp>.

**目前对以下需求进行了封装**

* 一般的get请求
* 一般的post请求
* 基于Http Post的文件上传（类似表单）
* 文件下载/加载图片
* 上传下载的进度回调
* 支持取消某个请求
* 支持自定义Callback
* 支持HEAD、DELETE、PATCH、PUT
* 支持session的保持
* 支持自签名网站https的访问，提供方法设置下证书就行

**配置OkhttpClient**

默认情况下，将直接使用okhttp默认的配置生成OkhttpClient，如果你有任何配置，记得在Application中调用initClient方法进行设置。

public class MyApplication extends Application { @Override public void onCreate() { super.onCreate(); OkHttpClient okHttpClient = new OkHttpClient.Builder() // .addInterceptor(new LoggerInterceptor("TAG")) .connectTimeout(10000L, TimeUnit.MILLISECONDS) .readTimeout(10000L, TimeUnit.MILLISECONDS) //其他配置 .build(); OkHttpUtils.initClient(okHttpClient); } }

别忘了在AndroidManifest中设置。

**对于Cookie(包含Session)**

对于cookie一样，直接通过cookiejar方法配置，参考上面的配置过程。

CookieJarImpl cookieJar = new CookieJarImpl(new PersistentCookieStore(getApplicationContext())); OkHttpClient okHttpClient = new OkHttpClient.Builder() .cookieJar(cookieJar) //其他配置 .build(); OkHttpUtils.initClient(okHttpClient);

目前项目中包含：

* PersistentCookieStore //持久化cookie
* SerializableHttpCookie //持久化cookie
* MemoryCookieStore //cookie信息存在内存中

如果遇到问题，欢迎反馈，当然也可以自己实现CookieJar接口，编写cookie管理相关代码。

此外，对于持久化cookie还可以使用<https://github.com/franmontiel/PersistentCookieJar>.

相当于框架中只是提供了几个实现类，你可以自行定制或者选择使用。

**对于Log**

初始化OkhttpClient时，通过设置拦截器实现，框架中提供了一个LoggerInterceptor，当然你可以自行实现一个Interceptor 。

OkHttpClient okHttpClient = new OkHttpClient.Builder() .addInterceptor(new LoggerInterceptor("TAG")) //其他配置 .build(); OkHttpUtils.initClient(okHttpClient);

**对于Https**

依然是通过配置即可，框架中提供了一个类HttpsUtils

* 设置可访问所有的https网站

HttpsUtils.SSLParams sslParams = HttpsUtils.getSslSocketFactory(null, null, null); OkHttpClient okHttpClient = new OkHttpClient.Builder() .sslSocketFactory(sslParams.sSLSocketFactory, sslParams.trustManager) //其他配置 .build(); OkHttpUtils.initClient(okHttpClient);

* 设置具体的证书

HttpsUtils.SSLParams sslParams = HttpsUtils.getSslSocketFactory(证书的inputstream, null, null); OkHttpClient okHttpClient = new OkHttpClient.Builder() .sslSocketFactory(sslParams.sSLSocketFactory, sslParams.trustManager)) //其他配置 .build(); OkHttpUtils.initClient(okHttpClient);

* 双向认证

HttpsUtils.getSslSocketFactory( 证书的inputstream, 本地证书的inputstream, 本地证书的密码)

同样的，框架中只是提供了几个实现类，你可以自行实现SSLSocketFactory，传入sslSocketFactory即可。

**其他用法示例**

**GET请求**

String url = "http://www.csdn.net/"; OkHttpUtils .get() .url(url) .addParams("username", "hyman") .addParams("password", "123") .build() .execute(new StringCallback() { @Override public void onError(Request request, Exception e) { } @Override public void onResponse(String response) { } });

**POST请求**

OkHttpUtils .post() .url(url) .addParams("username", "hyman") .addParams("password", "123") .build() .execute(callback);

**Post String**

OkHttpUtils .postString() .url(url) .content(new Gson().toJson(new User("zhy", "123"))) .build() .execute(new MyStringCallback());

提交一个Gson字符串到服务器端。

**Post File**

OkHttpUtils .postFile() .url(url) .file(file) .build() .execute(new MyStringCallback());

将文件作为请求体，发送到服务器。

**Post表单形式上传文件**

OkHttpUtils.post()// .addFile("mFile", "messenger\_01.png", file)// .addFile("mFile", "test1.txt", file2)// .url(url) .params(params)// .headers(headers)// .build()// .execute(new MyStringCallback());

支持单个多个文件，addFile的第一个参数为文件的key，即类别表单中<input type="file" name="mFile"/>的name属性。

**自定义CallBack**

目前内部包含StringCallBack,FileCallBack,BitmapCallback，可以根据自己的需求去自定义Callback，例如希望回调User对象：

public abstract class UserCallback extends Callback<User> { @Override public User parseNetworkResponse(Response response) throws IOException { String string = response.body().string(); User user = new Gson().fromJson(string, User.class); return user; } } OkHttpUtils .get()// .url(url)// .addParams("username", "hyman")// .addParams("password", "123")// .build()// .execute(new UserCallback() { @Override public void onError(Request request, Exception e) { mTv.setText("onError:" + e.getMessage()); } @Override public void onResponse(User response) { mTv.setText("onResponse:" + response.username); } });

通过parseNetworkResponse回调的response进行解析，该方法运行在子线程，所以可以进行任何耗时操作，详细参见sample。

**下载文件**

OkHttpUtils// .get()// .url(url)// .build()// .execute(new FileCallBack(Environment.getExternalStorageDirectory().getAbsolutePath(), "gson-2.2.1.jar")// { @Override public void inProgress(float progress) { mProgressBar.setProgress((int) (100 \* progress)); } @Override public void onError(Request request, Exception e) { Log.e(TAG, "onError :" + e.getMessage()); } @Override public void onResponse(File file) { Log.e(TAG, "onResponse :" + file.getAbsolutePath()); } });

注意下载文件可以使用FileCallback，需要传入文件需要保存的文件夹以及文件名。

**显示图片**

OkHttpUtils .get()// .url(url)// .build()// .execute(new BitmapCallback() { @Override public void onError(Request request, Exception e) { mTv.setText("onError:" + e.getMessage()); } @Override public void onResponse(Bitmap bitmap) { mImageView.setImageBitmap(bitmap); } });

显示图片，回调传入BitmapCallback即可。

**上传下载的进度显示**

new Callback<T>() { //... @Override public void inProgress(float progress) { //use progress: 0 ~ 1 } }

callback回调中有inProgress方法，直接复写即可。

**HEAD、DELETE、PUT、PATCH**

OkHttpUtils .put()//also can use delete() ,head() , patch() .requestBody(RequestBody.create(null, "may be something"))// .build()// .execute(new MyStringCallback());

如果需要requestBody，例如：PUT、PATCH，自行构造进行传入。

**同步的请求**

Response response = OkHttpUtils .get()// .url(url)// .tag(this)// .build()// .execute();

execute方法不传入callback即为同步的请求，返回Response。

**取消单个请求**

RequestCall call = OkHttpUtils.get().url(url).build(); call.cancel();

**根据tag取消请求**

目前对于支持的方法都添加了最后一个参数Object tag，取消则通过OkHttpUtils.cancelTag(tag)执行。

例如：在Activity中，当Activity销毁取消请求：

OkHttpUtils .get()// .url(url)// .tag(this)// .build()// @Override protected void onDestroy() { super.onDestroy(); //可以取消同一个tag的 OkHttpUtils.cancelTag(this);//取消以Activity.this作为tag的请求 }

比如，当前Activity页面所有的请求以Activity对象作为tag，可以在onDestory里面统一取消。

**混淆**

#okhttputils -dontwarn com.zhy.http.\*\* -keep class com.zhy.http.\*\*{\*;} #okhttp -dontwarn okhttp3.\*\* -keep class okhttp3.\*\*{\*;} #okio -dontwarn okio.\*\* -keep class okio.\*\*{\*;}