NoHttp

使用文档

目　录

概述

该文档是Android开源网络框架NoHttp的使用文档，有任何疑问请联系：smallajax@foxmail.com。

1. NoHttp开源地址：https://github.com/yanzhenjie/NoHttp
2. NoHttp详细文档：http://doc.nohttp.net
3. NoHttp公益接口：http://api.nohttp.net
4. QQ技术交流群：46523908，点击直接加入。

有任何问题可以在文档下方评论，也可以加入上方的QQ群来交流。

欢迎关注我

关注我的Github，了解我的最新项目。关注我的博客，阅读我的最新文章。

扫码关注我的微信：

开始使用

目录

1. 目前Android的主流开发工具是AndroidStudio，但是也有部分同学是使用的Eclipse，所以这里给出两种开发工具的使用方法。
2. NoHttp的底层默认使用`HttpURLConnection`实现，但是NoHttp网络层接口允许在初始化的时候配置，所以它允许无缝替换底层框架，NoHttp作者也提供了一个基于`OKHttp`的底层接口实现。

AndtoidStudio如何依赖NoHttp（推荐）

- 如果仅仅使用HttpURLConnection作为网络层，在app的gralde中添加以下依赖即可：

```
compile 'com.yanzhenjie.nohttp:nohttp:1.1.1'
```

- 如果要使用OkHttp作为网络层，请再依赖（注意两个lib的版本需要一致）：

```
compile 'com.yanzhenjie.nohttp:okhttp:1.1.1'
```

注意：不论使用基于HttpURLConnection还是OkHttp的版本，NoHttp的使用方法都不会变，这是NoHttp的优点之一。

Eclipse如何依赖NoHttp

1. 如果想依赖源码，请到Github-NoHttp上自行下载源码，然后转为Eclipse的项目格式后导入Eclipse即可。
2. 使用jar包，请在Github-NoHttp上下载NoHttp提供的jar包，copy到你的项目下的libs下即可。

> 如果你仅仅想用HttpURLConnection只需要下载nohttp.jar即可，如果想使用okhttp的话还要下载nohttp-okhttp.jar，并且需要开发者自自行到okhttp主页下载okhttp的jar，到okio的主页下载okio的jar。

初始化与配置

目录

NoHttp初始化需要一个Context，最好在`Application#onCreate()`中初始化，记得在`manifest.xml`中注册`Application`。

Application：

```
package com.yanzhenjie.simple;

public class MyApplication extends android.app.Application {

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

}
```

manifest.xml：

```
...

<application
        android:name="com.yanzhenjie.simple.MyApplication"
        ...
 />
```

默认初始化

如果使用默认始化后，一切采用默认设置。如果你需要配置全局超时时间、缓存、Cookie、

底层为OkHttp的话，请看高级初始化。

```
...
public class MyApplication extends android.app.Application {

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

        NoHttp.initialize(this); // NoHttp默认初始化。
    }

}
```

高级初始化

超时配置

如果不设置，默认全局超时时间是10s。

```
NoHttp.initialize(this, new NoHttp.Config()
    .setConnectTimeout(30 * 1000) // 全局连接超时时间，单位毫秒。
    .setReadTimeout(30 * 1000) // 全局服务器响应超时时间，单位毫秒。
);
```

配置缓存，

默认是开启状态，且保存在数据库。

- 设置缓存到数据库、禁用缓存

  ```
  NoHttp.initialize(this, new NoHttp.Config()
  ...
  .setCacheStore(
        new DBCacheStore(this) // 配置缓存到数据库。
        .setEnable(true) // true启用缓存，fasle禁用缓存。
    )
  );
  ```

- 设置缓存到本地SD卡

  如果你想缓存数据到SD卡，那么你需要考虑6.0及以上系统的运行时权限，推荐你看这篇文章：Android6.0运行时权限最佳实践。

  ```
  NoHttp.initialize(this, new NoHttp.Config()
  ...
    .setCacheStore(
        new DiskCacheStore(this) // 配置缓存到SD卡。
    )
  ```

```
    );
```

配置Cookie

默认是开启状态，保存数据库，NoHttp暂时没有提供保存在其它位置的默认实现，开发者可以新建一个类，实现 `java` 自带的的 `CookieStore` 接口。

```
NoHttp.initialize(this, new NoHttp.Config()
        ...
    .setCookieStore(
        new DBCookieStore(this)
        .setEnable(false) // true启用自动维护Cookie, fasle禁用自动维护Cookie。
    )
);
```

配置网络层

NoHttp的网络层是通过 `NetworkExecutor` 接口来配置的，内部提供了一个基于 `HttpURLConnection` 的接口实现类 `URLConnectionNetworkExecutor` ，在 `NoHttp` 项目中用另一个 `module` 提供了一个基于 `OkHttp` 的接口实现类 `OkHttpNetworkExecutor` ，二者选其一即可，关于二者该如何使用选择请看项目如何引入NoHttp。

值得注意的是：切换了NoHttp的网络底层后，NoHttp的上层代码不需要任何改动，你的应用层代码也不需要任何改动。

默认采用 `HttpURLConnection` 的实现做底层，既 `URLConnectionNetworkExecutor` 。

```
NoHttp.initialize(this, new NoHttp.Config()
    ...
    .setNetworkExecutor(new URLConnectionNetworkExecutor()) // 使用HttpURL
Connection做网络层。
);
```

如果要使用OkHttp作为网络层，请在app的gradle中添加依赖：

```
compile 'com.yanzhenjie.nohttp:okhttp:1.1.0'
```

然后在初始化的时候这么做：

```
NoHttp.initialize(this, new NoHttp.Config()
```

```
    ...
    .setNetworkExecutor(new OkHttpNetworkExecutor())  // 使用OkHttp做网络层
。
);
```

切换底层为OkHttp

目录

网络层配置

强烈建议你看此文之前看看NoHttp的初始化与配置。

NoHttp的网络层是通过`NetworkExecutor`接口来配置的，内部提供了一个基于`HttpURLConnection`的接口实现类`URLConnectionNetworkExecutor`，在`NoHttp`项目中用另一个`module`提供了一个基于`OkHttp`的接口实现类`OkHttpNetworkExecutor`，二者选其一即可，关于二者该如何使用选择请看项目如何引入NoHttp。

值得注意的是：切换了NoHttp的网络底层后，NoHttp的上层代码不需要任何改动，你的应用层代码也不需要任何改动。

默认采用`HttpURLConnection`的实现做底层，既`URLConnectionNetworkExecutor`。

```
NoHttp.initialize(this, new NoHttp.Config()
    ...
    .setNetworkExecutor(new URLConnectionNetworkExecutor()) // 使用HttpURL
Connection做网络层。
);
```

如果要使用OkHttp作为网络层，请在app的gradle中添加依赖：

```
compile 'com.yanzhenjie.nohttp:okhttp:1.1.0'
```

然后在初始化的时候这么做：

```
NoHttp.initialize(this, new NoHttp.Config()
    ...
    .setNetworkExecutor(new OkHttpNetworkExecutor())  // 使用OkHttp做网络层
```

```
。
);
```

到底该用OKHttp还是URLConnection

好多人咨询到底是使用`HttpURLConnection`还是`OkHttp`，下面做个简单的解释。

`HttpURLConnection`是Android系统自带的api，无需依赖其它任何第三方库。

- HttpURLConnection
  - 不用依赖第三方底层框架，相应的apk的体积也不会增大。
  - 在5.0以下的系统中`DELETE`请求方法不允许发送`body`，因此会在http协议的实现上做一些妥协。
  - 在`Android4.4`以后`HttpURLConnection`的底层使用`OkHttp2.7.5`来实现。
- OkHttp
  - `square`开发的第三方框架（非系统集成），相对高效、稳定。
  - 写文档的时候OkHttp已经更新到`3.4.1`了。
  - 使用OkHttp的好处是第三方框架有bug可以改代码，不像系统集成的api没办法改动。

我个人比较推荐使用OkHttp作为NoHttp的底层，我们公司的所有项目也是用nohttp的，全都是配置okhttp为底层的。

调试模式

目录

一个优秀的lib，一定有它人性化的调试模式，NoHttp也不例外，为了方便开发者查看请求过程和请求日志，NoHttp用一个Logger类来负责Log的打印。

如果你请求失败了，请求发生异常了，在吐槽作者之前，请打开调试模式看看NoHttp打印出的优雅的Log。

打开调试模式、设置TAG

NoHttp的调试模式，主要是提供一个合理的日志来供开发者查看和排查错误，默认的Log的TAG是“NoHttp”字符串。

NoHttp的调试模式的控制NoHttp初始化与配置一样，最好在`Application#onCreate()`中设置。

```
package com.yanzhenjie.simple;

public class MyApplication extends android.app.Application {

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

        Logger.setDebug(true); // 开启NoHttp调试模式。
                Logger.setTag("NoHttpSample"); // 设置NoHttp打印Log的TAG。
                ...
    }

}
```

Log的预览说明

这里一个成功请求的例子：

```
-------------Request start-------------
Request address: http://api.nohttp.net/test
Request method: POST
Accept: application/json,application/xml,application/xhtml+xml,text/html;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh
Author: sample
Connection: keep-alive
Content-Length: 51
Content-Type: application/x-www-form-urlencoded; charset=utf-8
User-Agent: Mozilla/5.0 (Linux; U; Android 6.0; zh-cn; XT1079 Build/MPB24.65-34) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile Safari/534.30
-------Send request data start-------
Push RequestBody: userName=yolanda&userPass=1&userAge=1.25&nooxxx=1.2
-------Send request data end-------
-------Response start-------
Content-Type: text/html;charset=utf-8
Date: Thu, 20 Oct 2016 16:11:30 GMT
ResponseCode: 200
Set-Cookie: JSESSIONID=51E86614EC36170CF4D5FF4DEF608389;path=/;HttpOnly
Transfer-Encoding: chunked
X-Android-Received-Millis: 1476979904386
X-Android-Response-Source: NETWORK 200
X-Android-Selected-Protocol: http/1.1
X-Android-Sent-Millis: 1476979904314
-------Response end-------
-------------Request finish-------------
```

上方的Log打印了一个Request完整的声明周期，NoHttp的一个请求的Log有以下特点：

1. 开头和结尾打开了`---Request Start---`和`---Request Finish---`分割请求，完整的生命周期的内容都会打印在开头和结尾的里面。
2. 在`---Request Start---`之后会打印请求的`url`，如果是`GET`、`HEAD`请求方式，通过`request.add(key, value)`添加的参数将会在这里完整的以`url?key=value&key=value`的形式打印。
3. 接着会打印请求方法、请求头，如果你要查看Cookie是否发送，你添加的自定Head是否被发送，你应该查看这里。
4. 然后我们注意到`--Response start--`和`--Response end--`，这一段Log会在服务器响应后被打印，将会把服务器的响应头都打印出来，包括服务器发送过来的`Set-Cookie`、响应码等。

权限说明

如果要下载文件到SD卡、或者配置了缓存数据到SD卡，你必须要考虑到Android6.0及以上系统的运行时权限，推荐你看这篇文章：Android6.0运行时权限最佳实践。

因为要请求网络、监听网络状态、从SD卡读写缓存、下载文件到SD卡等等，所以需要在`manifest.xml`中配置以下几个权限，如果你已经配置过了这些权限，请不要重复配置：

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
 />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" /
>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

请求数据

目录

各种Request介绍

想要发起一个请求就要有请求对象，NoHttp的理念是你想请求什么数据，就构造什么样的请求对象。任何请求对象都支持发送任何数据，关于提交数据、提交参数文档待补充。

NoHttp支持请求String、JsonObject、JsonArray、Bitmap、byte[]，支持自定义请求，例如JavaBean、FastJson、Gson等。

StringRequest

- 下面的方式将创建一个请求方法为`GET`的StringRequest。

```
Request<String> request = NoHttp.createStringRequest(url);
```

- 如果要创建其它请求方法的StringRequest，第二个参数传入对应的Method即可。
  例如POST请求：

```
Request<String> request = NoHttp.createStringRequest(url, RequestMetho
d.POST);
```

JsonObjectRequest

- 下面的方式将创建一个请求方法为`GET`的JsonObjectRequest。

```
Request<JSONObject> request = NoHttp.createJsonObjectRequest(url);
```

- 如果要创建其它请求方法的JsonObjectRequest，第二个参数传入对应的Method即可。
  例如POST请求：

```
Request<JSONObject> request = NoHttp.createJsonObjectRequest(url,  RequestMethod.POST);
```

JsonArrayRequest

- 下面的方式将创建一个请求方法为 `GET` 的JsonArrayRequest。

```
Request<JSONArray> request = NoHttp.createJsonArrayRequest(url);
```

- 如果要创建其它请求方法的JsonArrayRequest，第二个参数传入对应的Method即可。
  例如POST请求：

```
Request<JSONArray> request = NoHttp.createJsonArrayRequest(url, RequestMethod.POST);
```

BitmapRequest

- 下面的方式将创建一个请求方法为 `GET` 的BitmapRequest

```
Request<Bitmap> request = NoHttp.createBitmapRequest(url);
```

- 如果要创建其它请求方法的BitmapRequest，第二个参数传入对应的Method即可
  例如POST请求：

```
Request<Bitmap> request = NoHttp.createBitmapRequest(url, RequestMethod.POST);
```

ByteArrayRequest

- 下面的方式将创建一个请求方法为 `GET` 的ByteArrayRequest。

```
Request<byte[]> request = NoHttp.createByteArrayRequest(url);
```

- 如果要创建其它请求方法的ByteArrayRequest，第二个参数传入对应的Method即可。
  例如POST请求：

```
Request<byte[]> request = NoHttp.createByteArrayRequest(url, RequestMethod.POST);
```

自定义请求FastJson、JavaBean

请看这里。

请求优先级

NoHttp的队列支持请求优先级，通过以下代码给某个请求设置请求优先级：

```
request.setPriority(Priority.DEFAULT);
```

NoHttp有以下四个优先级别，默认请求的优先级别是`DEFAULT`

```
public enum Priority {
    LOW, DEFAULT, HEIGHT, HIGHEST
}
```

值	说明
HIGHEST	优先级别最高，一般用于队列中有HEIGHT的请求时，需要立即执行的请求
HEIGHT	优先级别高，低于HIGHEST
DEFAULT	默认值，低于HEIGHT
LOW	优先级最低

推荐阅读

1. NoHttp的队列异步请求基本使用
2. NoHttp队列特性详解

自定义请求-JavaBean

目录

NoHttp的几个默认请求，例如`StringRequest`、`BitmapRequest`全都是继承`RestRequest<T>`这个基类的，所以我们自定义请求的时候也要继承`RestRequest<T>`这个基类，泛型写你想请求的数据。

FastJsonRequest

例如我们用`FastJson`的`JSONObject`自定义一个请求。

```
public class FastJsonRequest extends RestRequest<JSONObject> {

    public FastJsonRequest(String url) {
        this(url, RequestMethod.GET);
    }

    public FastJsonRequest(String url, RequestMethod requestMethod) {
        super(url, requestMethod);

        // 设置Accept请求头，告诉服务器，我们需要application/json数据。
        setAccept(Headers.HEAD_VALUE_ACCEPT_APPLICATION_JSON);
    }

    /**
     * 解析服务器响应数据为你的泛型T，这里也就是JSONObject。
     *
     * @param responseHeaders 服务器响应头。
     * @param responseBody    服务器响应包体。
     * @return 返回你的泛型对象。
     */
    @Override
    public JSONObject parseResponse(Headers responseHeaders, byte[] respo
nseBody) {
        String result = StringRequest.parseResponseString(responseHeaders
, responseBody);
        return JSON.parseObject(result); // StringRequest就是少了这句话而已
。
    }
}
```

- 关于`parseResponse()`方法的说明

1. 这里要把byte[] body解析成Sting，一般我们用`String s = new String(body);`解析。
2. 但是服务器发送的数据有编码，所以我们要分析`header`中的`contentType`的编码是`utf-8`还是`gbk`或者其它，为了避免每个`request`都要解析`String`，作者在`StringRequest`写了一个静态方法统一解析，有疑问的人看点击进去看源码。
3. 拿到String之后，利用FastJson把数据解析成`JSONObject`对象。当然这里可以直接解析成JavaBean，请往下看。

- 使用方法
  其它使用方法和上面的NoHttp的原生使用方法一样，要注意的是，因为这里是你自己写的类，所以不能通过`NoHttp.create...Request`来写了，只能通过`new`的方式来构造。

```
// 默认使用GET请求方法。
FastJsonRequst request = new FastJsonRequest(url);
```

// 或者指定请求方法。
FastJsonRequst request = new FastJsonRequest(url, RequestMethod.POST);

```
#  请求JavaBean
1. 如果你没有看上面的自定FastJsonRequest，请先看过后再看自定义请求JavaBean会更容易理解。这里还是继承`RestRequest<T>`基类。

2. 这里先假设我们服务器返回的还是`json`类型的数据，如果是XML的话，利用[Simple](http://simple.sourceforge.net/)之类的工具解析。
```java
public class JavaBeanRequest<T> extends RestRequest<T> {

        // 要解析的JavaBean的class。
    private Class<T> clazz;

    public JavaBeanRequest(String url, Class<T> clazz) {
        this(url, RequestMethod.GET, clazz);
    }

    public JavaBeanRequest(String url, RequestMethod requestMethod, Class<T> clazz) {
        super(url, requestMethod);
        this.clazz = clazz;
    }

    @Override
    public T parseResponse(Headers responseHeaders, byte[] responseBody)
```
```

```
throws Throwable {
        String response = StringRequest.parseResponseString(responseHeade
rs, responseBody);

        // 这里如果数据格式错误，或者解析失败，会在失败的回调方法中返回 ParseError
异常。
        return JSON.parseObject(response, clazz);
    }
}
```

- 关于 `parseResponse()` 方法的说明
  1. 和上面自定FastJson一样，先要把数据解析成String。
  2. 接着利用FastJson把String解析成我们想要的 `JavaBean` ，比如这里还需要一个 `JavaBean` 的 `class` 参数，那我们就在构造方法传入。

- 使用方法
  其它使用方法和上面的NoHttp的原生使用方法一样，要注意的是，因为这里是你自己写的类，所以不能通过 `NoHttp.create...Request` 来写了，只能通过 `new` 的方式来构造。

```
// 默认使用GET请求方法。
JavaBeanRequest<UserInfo> request = new JavaBeanRequest<>(url, UserInf
o.class);
```

// 或者指定请求方法。
JavaBeanRequest request = new JavaBeanRequest<>(url, RequestMethod.POST, GoodsInfo.class);

```

```

请求方法(GET, POST...)

这里的请求方法指的是Http的请求方法，例如GET、POST、DELETE等。

NoHttp支持以下8种请求方法，以`RequestMethod`枚举的形式给出：

```
public enum RequestMethod {
     GET, POST, PUT, DELETE, HEAD, PATCH, OPTIONS, TRACE
}
```

所以我们调用的时候是通过：`RequestMethod.GET`这样来引用其中一个请求方法的。

指定Request的Method

请求方法需要在构造的时候指定，下面以`StringRequest`举例说明：

- GET

  如果不传入第二个参数，默认为GET方法，当然你也可以选择传入：

  ```
  Request<String> stringReq = NoHttp.createStringRequest(url);
  // 或者
  Request<String> stringReq = NoHttp.createStringRequest(url , RequestMe
  thod.GET);
  ```

- POST

  ```
  Request<String> stringReq = NoHttp.createStringRequest(url , RequestMe
  thod.POST);
  ```

推荐阅读

1. 如何请求JSON、Bitmap、JavaBean等。

同步请求

Android同步网络请求就是在当前线程发起请求。当然同步请求不能直接在主线程使用，所以一般是在子线程使用这种方法。

NoHttp的核心还是同步请求，下一章要讲的异步请求也是基于同步请求的。

这里以`StringRequest`为例：

```
// 创建请求。
Request<String> request = NoHttp.createStringRequest(url, RequestMethod.G
ET);

// 调用同步请求，直接拿到请求结果。
Response<String> response = NoHttp.startRequestSync(request);
```

推荐阅读

1. 如何请求JSON、Bitmap、JavaBean。

异步请求

目录

NoHttp的核心还是同步请求，本章要讲的异步请求也是基于同步请求的。

NoHttp的异步请求是在同步请求的基础上用`任务队列` + `Handler`实现的，如果你想更好的理解NoHttp，更好的使用队列的特性，强烈建议看这里。

异步请求的步骤

1、创建队列（队列特性讲解点我）

```
RequestQueue queue = NoHttp.newRequestQueue();
```

2、创建请求

```
Request<String> request = new StringRequest(url);

// 添加url?key=value形式的参数
request.add("enName", "yanzhenjie");
request.add("zhName", "严振杰");
request.add("website", "http://www.yanzhenjie.com");
```

3、添加请求到队列

```
...

queue.add(0, request, new OnResponseListener<String>(){

    @Override
    public void onSucceed(int what, Response<String> response) {
        if(response.responseCode() == 200) {// 请求成功。
            String result = response.get();
        }
    }

    @Override
    public void onFailed(int what, Response<String> response) {
```

```
                Excepition exception = response.getException();
        if(exception instanceof NetworkError) {// 网络不好。
        }

        // 这里还有很多错误类型，可以看demo：
        https://github.com/yanzhenjie/NoHttp
        ...
    }

        @Override
    public void onStart(int what) {
        // 这里可以show()一个wait dialog。
    }

    @Override
    public void onFinish(int what) {
                // 这里可以dismiss()上面show()的wait dialog。
    }
});
```

这里对其中`queue.add(what, request, listener)`中的`what`做个说明，任意添加多个请求到队列时，使用同一个Listener接受结果，listener的任何一个方法被回调时都会返回在添加请求到队列时写的相应what值，可以用这个what来区分是哪个请求的回调结果，你可以理解为它的作用和`handler`的`what`一样的作用，就是用来区分信号来源的。

这样做的好处是不像其它框架一样，每个请求都`new listener()`来接受结果，这样及省了代码量，又让代码更加整洁。

当然如果你不想这么用，那么你可以每个请求都`new listener()`。

对于队列使用的建议

创建一个队列应该多次使用，因为每创建一个队列，就会创建指定`并发值`个线程，如果创建太多队列就会耗资源，所以我们要把队列封装成单例模式，强烈建议你请看这篇文档：队列特性讲解。

推荐阅读
队列的详解和封装
同步请求

队列详解与封装

目录

队列的创建

```
// 默认并发值为3
RequestQueue reqQueue = NoHttp.newRequestQueue();

// 或者传入并发值
RequestQueue reqQueue = NoHttp.newRequestQueue(1);
```

> 并发的意思是，最多可以同时并行多少个请求。

1. 当一个页面初始化时要请求多个接口，那么耗时相对会更长，对于用户的体验是很差的，所以如果能同时执行多个请求，那么将会缩短网络请求的时间。
2. 假如我们传入的并发值是3，但是我们同时添加了5个请求到队列，第四个请求将会在前三个请求的任何一个完成后被发起，以此类推...
3. 假如我们想让请求一个个执行，那么我们讲队列设置为1个并发，我们连续添加了10个请求到队列，这个10个队列将会按照添加顺序依次执行。如果要设置请求优先级，请往下看。
4. 如果给`Request`设置了优先级，将优先执行优先级别较高的请求，关于优先级别请看这里。

队列的封装

创建一个队列应该多次使用，因为每创建一个队列，就会创建指定`并发值`个线程，如果创建太多队列就会耗资源，所以我们要把队列封装成单例模式。

队列的单例模式封装

```
public class CallServer {

    private static CallServer instance;

    /**
     * 请求队列。
     */
    private RequestQueue requestQueue;

    private CallServer() {
        requestQueue = NoHttp.newRequestQueue(3);
    }

    /**
     * 请求队列。
     */
    public static CallServer getInstance() {
        if (instance == null)
                synchronized (CallServer.class) {
                if (instance == null)
                        instance = new CallServer();
            }
        return instance;
    }

    /**
     * 添加一个请求到请求队列。
     *
     * @param what      用来标志请求, 当多个请求使用同一个Listener时, 在回调方法
中会返回这个what。
     * @param request   请求对象。
     * @param listener  结果回调对象。
     */
    public <T> void add(int what, Request<T> request, OnResponseListener
listener) {
        requestQueue.add(what, request, listener);
    }

    /**
     * 取消这个sign标记的所有请求。
     * @param sign 请求的取消标志。
     */
    public void cancelBySign(Object sign) {
        requestQueue.cancelBySign(sign);
    }

    /**
     * 取消队列中所有请求。
     */
    public void cancelAll() {
        requestQueue.cancelAll();
    }
}
```

- 封装如何使用
  我们只需要在需要使用请求的地方这样调用即可：

```
Request request = ...
...
```

CallServer.getInstance().add(0, request, Listener);

```
## 在BaseActivity和BaseFragment中封装
如果你看[这篇文章](http://doc.nohttp.net/222886)，你会发现取消请求虽然可以与`Act
ivity`、`Fragment`的生命周期绑定，但是每个Activity和Fragment都这么写就显得有点麻
烦了，所以我们这里把这些操作写在`BaseActivity`、`BaseFragment`中。

在Base中提供一个请求的方法，具体参数请结合自己的业务和习惯封装。
```java
...

private Object cancelObject = new Object();

public <T> void request(int what, Request<T> request, OnResponseListener<
T> listener) {
    // 这里设置一个sign给这个请求。
    request.setCancelSign(cancelObject);

    CallServer.getInstance().add(what, request, listener);
}

@Override
protected void onDestroy() {
    // 在组件销毁的时候调用队列的按照sign取消的方法即可取消。
    CallServer.getInstance().cancelBySign(cancelObject);
    super.onDestroy();
}
```

推荐阅读

1. NoHttp的队列异步请求
2. 取消请求、取消队列中的请求
3. Request优先级别
```

发送数据/文件/json/表单

目录

这里要说明的是NoHttp除了提交自定义`Body`外，其它提交任何数据都是以`Request#add(key, value)`的形式添加到`Request`的，这是NoHttp为了方便开发者故意这么设计的。而且NoHttp提供了`Request#add(Map)`这样的方法添加参数，还有其它小惊喜等待你去挖掘。

add()方法的特点说明

注意：NoHttp添加多个相同key的参数，不会被覆盖，会全部发送到服务器，但是允许在添加后通过`Request#set(String...)`覆盖这个key下所有的参数，也可以通过`Request#remove(String)`移除这个key下的所有参数。

GET提交普通参数

Http GET发送请求的时候，最终参数都会以`url?key=value&key1=value1`这样的形式拼接在url末尾。

例如：

- url: `http://api.nohttp.net/upload?id=123&name=yanzhenjie&desc=abc`
- method: `GET`

用NoHttp的时候你不用拼接参数，如下写即可：

```
String url = "http://api.nohttp.net/upload";

Request<String> request = new StringRequest(url);
request.add("id", 123)
        .add("name",  "yanzhenjie")
    .add("desc", "abc");
```

就是这么简单，剩下的事情`NoHttp`会自动完成。

POST提交普通参数

Http POST发送请求的时候，不像GET一样，POST的url最终不会变的，参数也不会拼接到url后面，它的参数会拼接成`key=value&key1=value`的形式用流写出去，也就是说它的参数是以body的形式发送的。

例如：

- ur: `http://api.nohttp.net/upload`
- method `POST`
- params: `name=yanzhenjie&pwd=12345`

这些参数用nohttp这样写：

```
String url = "http://api.nohttp.net/upload";

Request<String> request = new StringRequest(url, RequestMethod.POST)
    .add("id", 123)
        .add("name",  "yanzhenjie")
    .add("desc", "abc");
```

和GET没什么需别，唯一的区别的就是指定了请求方法为POST。其它类似POST的请求，比如PUT、PATCH、DELETE等方法都是如此。

推荐阅读

请求各种类型的数据

自定义请求JavaBean

表单文件上传

目录

NoHttp提供了两种上传文件的方式，这一章介绍了模拟表单上传文件，可以传大文件、多文件，不会发生内存溢出等问题。

```
Request<String> request = new StringRequest(url, RequestMethod.POST)
   .add("name",  "严振杰") // 普通参数。
   .add("head", Binary...); // 文件参数。
```

这里要说明的是，在表单上传中NoHttp把文件都看成`Binary`，`Binary`是NoHttp的一个接口，它有如下几种默认实现：

- FileBinary
- InputStreamBinary
- ByteArrayBinary
- BitmapBinary

多个key，上传多个文件

参数	值	描述
head	非空	用户头像
image1	非空	发布图片1
image2	非空	发布图片2
image3	非空	发布图片3

服务器要求我们每个key对应一个文件上传，这种时候我们可以像添加普通参数那样添加文件参数：

```
File file = ...
Bitmap bitmap = ...

Binary binary1 = new FileBinary(file);
```

```
Binary binary2 = new BitmapBinary(bitmap);

Request<String> request = new StringRequest(url, RequestMethod.POST);
request.add("file", binary1).add("userHead", binary2);
```

1个key，上传多个文件

参数	值	描述
images	非空	用户发布的多张图片

服务器要求我们1个key对应多文件上传，这里NoHttp支持两种做法。

第一种做法，添加`List<Binary>`：

```
List<Binary> binaries = new Arraylist<>(); // 文件list。
binaries.add(new FileBinary(file));
binaries.add(new BitmapBinary(bitmap, "head.png"));

Request<String> request = new StringRequest(url, RequestMethod.POST)
   .add("images", binaries); // 添加文件list。
```

第二种做法，添加多个相同Key的Binary：

```
File file = ...
Bitmap bitmap = ...

Request<String> request = new StringRequest(url, RequestMethod.POST);
        // 添加多个相同key的Binary。
   .add("images", new FileBinary(file))
   .add("images", new BitmapBinary(bitmap, "nohttp.png"));
```

监听文件上传进度

其实`Binary`这个接口只有四个简单的方法，我们看下源码：

```
public interface Binary extends Cancelable {

    long getLength(); // 返回文件大小。

    String getFileName(); // 返回文件名称。

    String getMimeType(); // 返回文件MimeType。

    void onWriteBinary(OutputStream outputStream); // 用传入的流，写出文件。

}
```

这四个方法是不是很简单，但是我们看到没有提供统计文件上传进度的方法啊。其实进度是在`onWriteBinary()`中计算出来的，为了避免每一个`Binary`都计算，所以NoHttp提供一个`BasicBinary`，`BasicBinary`实现了`Binary`接口，并且已经实现好了进度计算，我们上面说的四种文件上传的类都是继承自`BasicBinary`：

- FileBinary
- InputStreamBinary
- ByteArrayBinary
- BitmapBinary

所以如果你要自定义了上传文件的类，你可以继承`BasicBinary`基类，继承后值需要简单的几行代码就可以搞定一个自定义上传类，具体代码可以参考上述四个上传的类。

因为上述四种`Binary`都继承自`BasicBinary`，所以他们全都支持监听上传进度，用法如下：

注意：所有的方法都会在主线程中被回调，所以你可以直接更新UI。

```
...
FileBinary fileBinary = new FileBinary(useHeadFile)
fileBinary.setUploadListener(0, mOnUploadListener); // 设置一个上传监听器。
request.add("key", fileBinary);

private OnUploadListener mOnUploadListener = new OnUploadListener() {
        @Override
        public void onStart(int what) {// 文件开始上传。
        }

        @Override
        public void onCancel(int what) {// 文件的上传被取消时。
        }

        @Override
        public void onProgress(int what, int progress) {// 文件的上传进度发
生变化。
        }

        @Override
        public void onFinish(int what) {// 文件上传完成
        }

        @Override
        public void onError(int what, Exception exception) {// 文件上传发生
错误。
        }
    };
```

提交普通表单

我们知道，表单一般用POST类型（POST、DELETE、PATCH等）的请求方式来提交，在客户端开发中，通常表单一般用来上传文件，但是如果服务器同学需要我们以表单的形式提交普通参数那也无可厚非，所以NoHttp也提供了这种方式。

例如：

- ur: `http://api.nohttp.net/upload`
- method: `POST`，模拟表单

参数	值	描述
name	非空	用户名
desc	非空	描述文字

如果服务器要求我们提交这样一个表单，用NoHttp依然不变：

```
String url = "http://api.nohttp.net/upload";

Request<String> request = new StringRequest(url, RequestMethod.POST);
request.add("id", 123)
        .add("name",  "yanzhenjie")
    .add("desc", "abc")
    .setMultipartFormEnable(true); // 就多了这一句。
```

和POST提交普通参数几乎没有区别，唯一多了一句`.setMultipartFormEnable(true)`把这个请求标志为表单请求即可。

提交Body/Json/InputStream

目录

Http POST类型的请求，允许发送body到服务器，这也意味着我们可以push任意数据到服务器，NoHttp提供了一下几种方式供开发者选择使用。

提交/上传json

NoHttp提供了两种方式，开发者只需要传入json格式的数据即可，NoHttp会自动修改`ContentType`为`application/json`。

```
request.setDefineRequestBodyForJson(JsonString); // 传入json格式的字符串即可。

request.setDefineRequestBodyForJson(JSONObject); // 传入JSONObject即可。
```

提交/上传xml

开发者只需要传入相应格式的数据即可，NoHttp会自动修改`ContentType`为`application/xml`。

```
request.setDefineRequestBodyForXML(XmlString); // 提交xml字符串
```

提交/上传String

这里因为是自定义格式的数据，需要开发者自行指定body的`ContentType`。

```
request.setDefineRequestBody(String requestBody, String contentType);
```

提交文件、InputStream等自定义数据

同样的，这里可以传一个`InputStream`进去，因为`Stream`也属于自定义数据，所以必须传一个`ContentType`。

```
request.setDefineRequestBody(InputStream, ContentType)
```

如果你们是直接push一个文件上去：

```
request.setDefineRequestBody(new FileInputStream(file), "application/octe
t-stream")
```

PHP后台传文件list接受不到

目录

PHP后台接受不到文件数组

有开发者反馈后台是PHP，用html写的表单传文件数组，PHP可以接受到这个数组，但是用NoHttp的时候PHP后台只能接受到数组的最后一个文件。

我是做Java开发的，于是写了个Java后台测试，结果显示NoHttp完全没有问题，于是我去翻了PHP的文档，发现这个PHP的特性，在说明之前先看看NoHttp是怎么一个`key`传文件数组的。

传送门：php传文件数组的官方文档说明连接。

第一种做法，添加`List<Binary>`：

```
List<Binary> binaries = new Arraylist<>(); // 文件list。
binaries.add(new FileBinary(file));
binaries.add(new BitmapBinary(bitmap, "head.png"));

Request<String> request = new StringRequest(url, RequestMethod.POST)
   .add("images", binaries); // 添加文件list。
```

第二种做法，添加多个相同Key的Binary：

```
File file = ...
Bitmap bitmap = ...

Request<String> request = new StringRequest(url, RequestMethod.POST);
        // 添加多个相同key的Binary。
   .add("images", new FileBinary(file))
   .add("images", new BitmapBinary(bitmap, "nohttp.png"));
```

PHP用html传文件list的方式

解释这个问题之前要先看看PHP在`html`中是如何传文件数组的，`html`的`form`是这样写的：

```
<form action="file-upload.php" method="post" enctype="multipart/form-data
">
  <input name="userfile[]" type="file" /><br />
  <input name="userfile[]" type="file" /><br />
  <input type="submit" value="上传" />
</form>
```

我们看到这里有两个`file input`，`name`为`userfile[]`，也就是NoHttp中的`Request#add(String key, Binary value)`中的key。

在PHP中这样接受：`$_FILES['userfile']`，所以这里我们发现后台获取这个FileList的时候的`key`是`userfile`，所以后台告诉Android开发者你传文件数组的时候的key是`userfile`，这是因为PHP要求一个`key`传文件数组的时候，`key`必须是`key[]`，它才会视为文件数组，到PHP后台的时候会自动把这个`[]`去掉，否则它将会被相同的key覆盖，这就是为什么使用NoHttp传文件list的时候PHP只能接受最后一个文件的原因。

解决方案

根据上面的分析我想你肯定明白了，就是在我们之前的`key`后面加`[]`：

```
List<Binary> binaries = new Arraylist<>(); // 文件list。
binaries.add(new FileBinary(file));
binaries.add(new BitmapBinary(bitmap, "head.png"));

Request<String> request = new StringRequest(url, RequestMethod.POST)
   .add("images[]", binaries); // 添加文件list。
```

取消请求

NoHttp提供了以下几种方法来取消请求：

取消单个请求

直接调用`Request`的`cancel()`方法即可：

```
/**
  * 请求对象。
  */
private Request<String> mRequest;

@Override
protected void onActivityCreate(Bundle savedInstanceState) {
    setContentView(R.layout.activity_cacel_demo);

        // 发起请求。
    mRequest = NoHttp.createStringRequest(Constants.URL_NOHTTP_TEST, Requ
estMethod.GET);
    ...
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // 退出时取消请求。
    if (mRequest != null)
        mRequest.cancel();
}
```

取消队列中的指定请求

一般用于一个页面有多个请求，在退出之前如果请求还没执行完时取消时。需要给这个`Request`设置一个`sign`，在取消的时候调用队列的`queue.cancenBySign(Object)`即可。

```
private RequestQueue queue;

private Object cancelSign = new Object();

@Override
```

```
protected void onActivityCreate(Bundle savedInstanceState) {
    setContentView(R.layout.activity_cacel_demo);

        // 初始化队列等。
    ...

    // 请求1。
    Request<String> request1 = NoHttp.createStringRequest(Constants.URL_N
OHTTP_TEST, RequestMethod.GET);
    request1.setCancelSign(cancelSign);

    // 请求2。
    Request<String> request2 = NoHttp.createStringRequest(Constants.URL_N
OHTTP_TEST, RequestMethod.GET);
    request2.setCancelSign(cancelSign);

    // 把请求加入队列
        queue.add(1, request1, listener);
    queue.add(2, request2, listener);
}

private OnResponseListener listener = new OnResponseListener() {
    ...
}

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

    queue.cancelBySign(cancelSign);
}
```

取消队列中的所有请求

一般用于APP退出时取消未完成的所有请求，或者某个页面退出时取消所有请求。

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

    queue.cancelAll();
}
```

推荐阅读

1. NoHttp队列特性详解

1. 请求的优先级

取消请求

取消请求的封装

强烈建议在阅读本文之前阅读取消请求章节。

如果你看了上面的文章，你会发现，取消请求虽然可以与`Activity`、`Fragment`的生命周期绑定，但是每个Activity和Fragment都这么写就显得有点麻烦了，所以我们这里把这些操作写在`BaseActivity`、`BaseFragment`中。

Base中的封装

在Base中提供一个请求的方法，具体参数请结合自己的业务和习惯封装。

```
...

private Object cancelSign = new Object();

public <T> void request(int what, Request<T> request, OnResponseListener<
T> listener) {
    // 这里设置一个sign给这个请求。
    request.setCancelSign(cancelSign);

    queue.add(this, what, request, listener);
}

@Override
protected void onDestroy() {
    // 在组件销毁的时候调用队列的按照sign取消的方法即可取消。
    queue.cancelBySign(cancelSign);
    super.onDestroy();
}
```

推荐阅读

取消请求的几种方式

队列详解与封装

Cookie管理 WebView同步

目录

NoHttp的Cookie管理原理

在文档的初始化配置一章讲了NoHttp如何配置或者禁用cookie自动管理。

NoHttp的Cookie自动维护，严格遵守Http协议，即区分临时Cookie和有效期Cookie。

- 临时Cookie在本次App运行期内一直有效，直到App被杀死即被清除。
- 有效期Cookie会带有一个过期时间，不论App是否被杀死过，这个Cookie在到期时会被自动清除。

关于Session的维持登录

Session是对于服务端来说的，客户端是没有Session一说的。Session是服务器在和客户端建立连接时添加客户端连接标志，最终会在服务器软件（Apache、Tomcat、JBoss）转化为一个临时Cookie发送给给客户端，当客户端第一请求时服务器会检查是否携带了这个Session（临时Cookie），如果没有则会添加Session，如果有就拿出这个Session来做相关操作。

综上所述Session也就是客户端在一次运行期内一直有效，客户端被重启或者杀死时这个Session转化来的临时Cookie即被清除，下次客户端启动后请求服务器时会重新有一个新的Session。

有写开发者是用Session维持App端用户登录状态的，根据上述描述，App重启后上次登录时的Session就失效了，此时要想维护Session的持续有效有两个办法：

一、每次启动App就登录一次

第一个办法很土，不安全，但很有效。当用户登录成功后，保存用户的帐号、密码、是否登录状态在本地（记得加密），然后在APP每次重启时检查用户是否登录，如果是登录，那么后台自动调用登录接口登录一次，就可以拿到登录的有效Cookie。

二、用NoHttp的Cookie管理监听

第二个办法相对安全，建议采用第二种办法。NoHttp在初始化的时候可以配置一个CookieStore，我们可以给这个CookieStore设置一个Cookie管理的监听，当Cookie被保存时设置Cookie的有效期为永久：

```
public class App extends Application {

    private static App mainCourseInstance;

    @Override
    public void onCreate() {
        super.onCreate();
                NoHttp.initialize(this, new NoHttp.Config()
                        .setCookieStore(new DBCookieStore(this).setCookie
StoreListener(mListener))
                );
    }

        /**
     * Cookie管理监听。
     */
        private DBCookieStore.CookieStoreListener mListener = new DBCooki
eStore.CookieStoreListener() {
        @Override
        public void onSaveCookie(URI uri, HttpCookie cookie) { // Cookie
被保存时被调用。
            // 1. 判断这个被保存的Cookie是我们服务器下发的Session。
            // 2. 这里的JSessionId是Session的name,
            //    比如java的是JSessionId, PHP的是PSessionId,
            //    当然这里只是举例, 实际java中和php不一定是这个, 具体要咨询你们服务
器开发人员。
            if("JSessionId".equals(cookie.getName())) {
                // 设置有效期为最大。
                cookie.setMaxAge(HeaderUtil.getMaxExpiryMillis());
            }
        }

        @Override
        public void onRemoveCookie(URI uri, HttpCookie cookie) {// Cookie
被移除时被调用。
        }
    }
}
```

NoHttp同步Cookie到原生的WebView

这里推荐一个方法，我们可以继承系统的WebView，然后设置一些必要属性后，重写 `WebView#loadUrl(String, Map<String, String>)` 方法。

第一步，继承WebView，重写 `loadUrl(String, Map<String, String>)` 方法：

```
public class MyWebView extends android.webkit.WebView {

    public MyWebView(Context context) {
        super(context);
    }

    public MyWebView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public MyWebView(Context context, AttributeSet attrs, int defStyleAtt
r) {
        super(context, attrs, defStyleAttr);
    }

    @Override
        public void loadUrl(String url, Map<String, String> httpHeader) {
        super.loadUrl(url, httpHeader);
        }
}
```

第二步，给 `loadUrl(String, Map<String, String>)` 方法添加具体添加自定义头和同步Cookie的代码：

```
@SuppressWarnings("deprecation")
@SuppressLint("NewApi")
@Override
public void loadUrl(String url, Map<String, String> httpHeader) {
    if (httpHeader == null) {
        httpHeader = new HashMap<>();
    }

    // 这里你还可以添加一些自定头。
    httpHeader.put("AppVersion", "1.0.0"); // 比如添加app版本信息，当然实际开
发中要自动获取哦。

    URI uri = null;
    try {
        uri = new URI(url);
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
    if (uri != null) {
        java.net.CookieStore cookieStore = NoHttp.getCookieManager().getC
ookieStore();
```

```
        List<HttpCookie> cookies = cookieStore.get(uri);

        // 同步到WebView。
        android.webkit.CookieManager webCookieManager = android.webkit.Co
okieManager.getInstance();
        webCookieManager.setAcceptCookie(true);
        for (HttpCookie cookie : cookies) {
                String cookieUrl = cookie.getDomain();
            String cookieValue = cookie.getName() + "=" + cookie.getValue
()
                        + "; path=" + cookie.getPath()
                        + "; domain=" + cookie.getDomain();

            webCookieManager.setCookie(cookieUrl, cookieValue);
        }

        if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_
CODES.LOLLIPOP) {
            webCookieManager.flush();
        } else {
            android.webkit.CookieSyncManager.createInstance(NoHttp.getCon
text()).sync();
        }
    }
    super.loadUrl(url, httpHeader);
}
```

NoHttp同步Cookie到腾讯X5 WebView

很多人在使用它腾讯提供的X5服务器，来替代Android原生的WebView，如果你正是使用腾讯X5内核的话，同样NoHttp也支持Cookie同步。

步骤和上面原生WebView没区别，但是要注意几点：

1. 继承不是系统的`android.webkit.WebView`，而是`com.tencent.smtt.sdk.WebView`。
2. 同步到X5内核时不再是`android.webkit.CookieManagerr`，而是`com.tencent.smtt.sdk.CookieManager`。
3. 同步到X5内核时不再是`android.webkit.CookieSyncManager`，而是`com.tencent.smtt.sdk.CookieSyncManager`。

具体代码如下：

```
public class MyWebView extends com.tencent.smtt.sdk.WebView {

    public MyWebView(Context context) {
        super(context);
    }
```

```
    public MyWebView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public MyWebView(Context context, AttributeSet attrs, int defStyleAtt
r) {
        super(context, attrs, defStyleAttr);
    }

        @SuppressWarnings("deprecation")
    @SuppressLint("NewApi")
    @Override
    public void loadUrl(String url, Map<String, String> httpHeader) {
        if (httpHeader == null) {
            httpHeader = new HashMap<>();
        }

        // 这里你还可以添加一些自定头。
        httpHeader.put("AppVersion", "1.0.0"); // 比如添加app版本信息，当然实
际开发中要自动获取哦。

        URI uri = null;
        try {
            uri = new URI(url);
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
        if (uri != null) {
            java.net.CookieStore cookieStore = NoHttp.getCookieManager().
getCookieStore();
            List<HttpCookie> cookies = cookieStore.get(uri);

            // 同步到腾讯X5 WebView。
            com.tencent.smtt.sdk.CookieManager webCookieManager = com.ten
cent.smtt.sdk.CookieManager.getInstance();
            webCookieManager.setAcceptCookie(true);
            for (HttpCookie cookie : cookies) {
                String cookieUrl = cookie.getDomain();
                String cookieValue = cookie.getName() + "=" + cookie.getV
alue()
                        + "; path=" + cookie.getPath()
                        + "; domain=" + cookie.getDomain();

                webCookieManager.setCookie(cookieUrl, cookieValue);
            }
            com.tencent.smtt.sdk.CookieSyncManager.createInstance(NoHttp.
getContext()).sync();
        }
        super.loadUrl(url, httpHeader);
    }
```

代码混淆

NoHttp设计到兼容高版本系统的api采用反射调用，所以所有类都可以被混淆，如果遇到问题，如下配置即可。

原生NoHttp混淆

```
-dontwarn com.yolanda.nohttp.**
-keep class com.yolanda.nohttp.**{*;}
```

如果使用okhttp的版本

```
// nohttp
-dontwarn com.yolanda.nohttp.**
-keep class com.yolanda.nohttp.**{*;}

// nohttp-okhttp
-dontwarn com.yanzhenjie.nohttp.**
-keep class com.yanzhenjie.nohttp.**{*;}

// okhttp
-dontwarn okhttp3.**
-keep class okhttp3.** { *;}
-dontwarn okio.**
-keep class okio.** { *;}
```