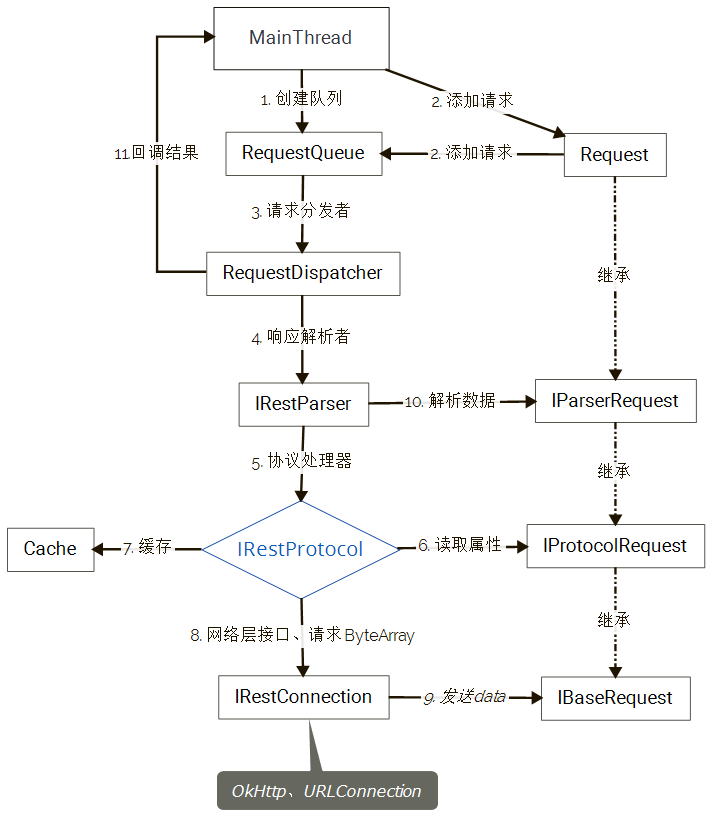
一个Android网络请求框架

NoHttp

万峥嵘 2017.3.10



1. 现阶段开发中所用到的网络请求

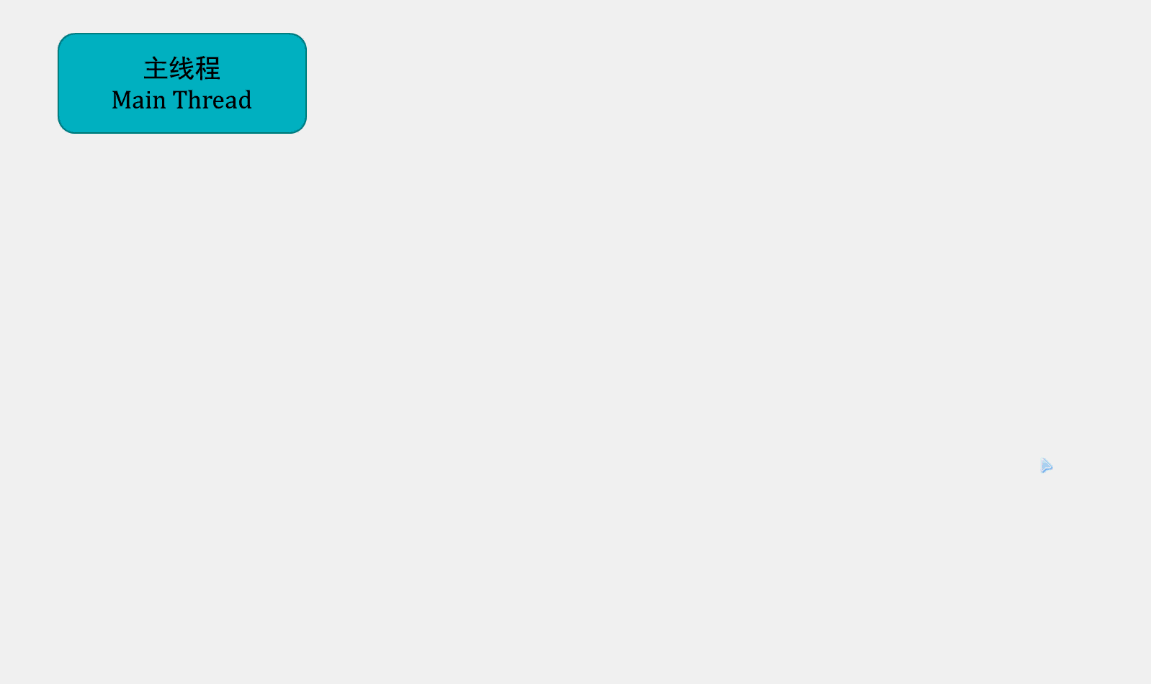
在移动一张图的开发中，在线的数据请求很多，比如在线的用户登录，地图加载，离线包项目包的数据下载。运用一个比较实用的网络请求框架，对开发者而言，显然是事半功倍的。那NoHttp有哪些特点呢？

1. 支持并发，支持异步/同步请求，自动维持[Cookie](http://baike.baidu.com/item/Cookie/1119)，大文件/多文件上传，文件下载。
2. 支持304缓存，302/303重定向，支持通过[代理](http://baike.baidu.com/item/%E4%BB%A3%E7%90%86/3242667)请求。
3. 简单、方便、高效、功能全、体积小(154k)。
4. okhttp与nohttp的异同

okhttp: 是Java和Android都可以用的(随着Android6.0删除了HttpClient后，Android也不能使用HttpClient封装的框架了)。okhttp从socket层入手。Android请求网络需要的子线程发起，然后用handler把结果发送到主线程。必须在子线程请求，还要自己封装参数、请求头、下载、传文件等等。

nohttp：是Android专用的，是不能用在Java的，不是在socket层做事情，不用去理睬socket，而是在上面的基础上衍生的框架。不论传参数，传文件，等都做了不同程度的封装。都可以在主线程直接调用，拿到结果后也可以直接更新UI。

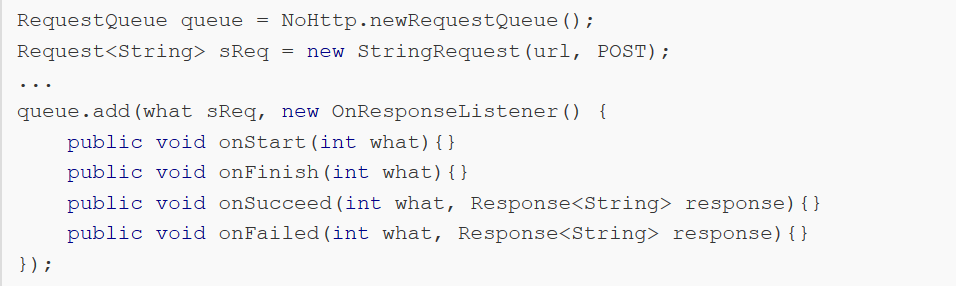
1. NoHttp动态流程图





从动态演示图中我们可以看到，任何一个请求都是从主线程开始发起，先把Request添加到RequestQueue（队列）中，队列此时会启动子线程，再由ReuqestDispatcher（请求分发者）把队列中请求按照一定的顺序分发给子线程去执行网络操作，ReuqestDispatcher拿到结果后，用Handler发送到主线程，这个过程就是NoHttp的整个异步请求的过程。

1. nohttp相关API以及代码解读



1、nohttp创建Resquest Queue：



这里有5个方法可以创建ReuqestQueue，但是每个方法最终都会来到最后方法，最后一个方法需要两个参数，第一个参数是IRestParser，也就是响应解析者，第二个参数是threadPoolSize，也就是队列中启动的线程的数量，创建好ReuqestQueue后，调用start()方法启动队列，这时就可以往队列中添加请求了。

那么在ReuqestQueue中究竟是怎么处理Request的？

public class RequestQueue {

/\*\*

\* 保存没有完成的请求，包括正在执行的请求。

\*/

private final BlockingQueue<Request<?>> mUnFinishQueue;

/\*\*

\* 保存没有执行的请求，不包括正在执行的请求。

\*/

private final BlockingQueue<Request<?>> mRequestQueue;

/\*\*

\* Http 请求结果解析器。

\*/

private final IRestParser mImplRestParser;

/\*\*

\* 线程池。

\*/

private RequestDispatcher[] mDispatchers;

public RequestQueue(IRestParser implRestParser, int threadPoolSize) {

mImplRestParser = implRestParser;

mDispatchers = new RequestDispatcher[threadPoolSize];

}

/\*\*

\* 启动线程池，轮训请求队列。

\*/

public void start() {

stop();

for (int i = 0; i < mDispatchers.length; i++) {

RequestDispatcher networkDispatcher = new RequestDispatcher(..., mImplRestParser);

mDispatchers[i] = networkDispatcher;

networkDispatcher.start();

}

}

/\*\*

\* 1. 添加一个请求到队列，如果队列中请求数没有满，则会立即执行，否则等待前面的请求执行完成后再执行。

\* 2. 在真正添加到队列前检查当前要添加的请求是否在队列中，如果重复添加则无任何操作。

\*/

public <T> void add(int what, Request<T> request, OnResponseListener<T> responseListener) {

...

}

/\*\*

\* 没有开始执行的请求数量。

\*/

public int unStartSize() {}

/\*\*

\* 没有完成的请求数量，包括正在执行的请求。

\*/

public int unFinishSize() {}

/\*\*

\* 停止队列，使三个线程停止，不进行轮训队列。

\*/

public void stop() {}

/\*\*

\* 根据sign取消所有用sign打标的请求。

\*/

public void cancelBySign(Object sign) {}

/\*\*

\* 取消队列中所有请求。

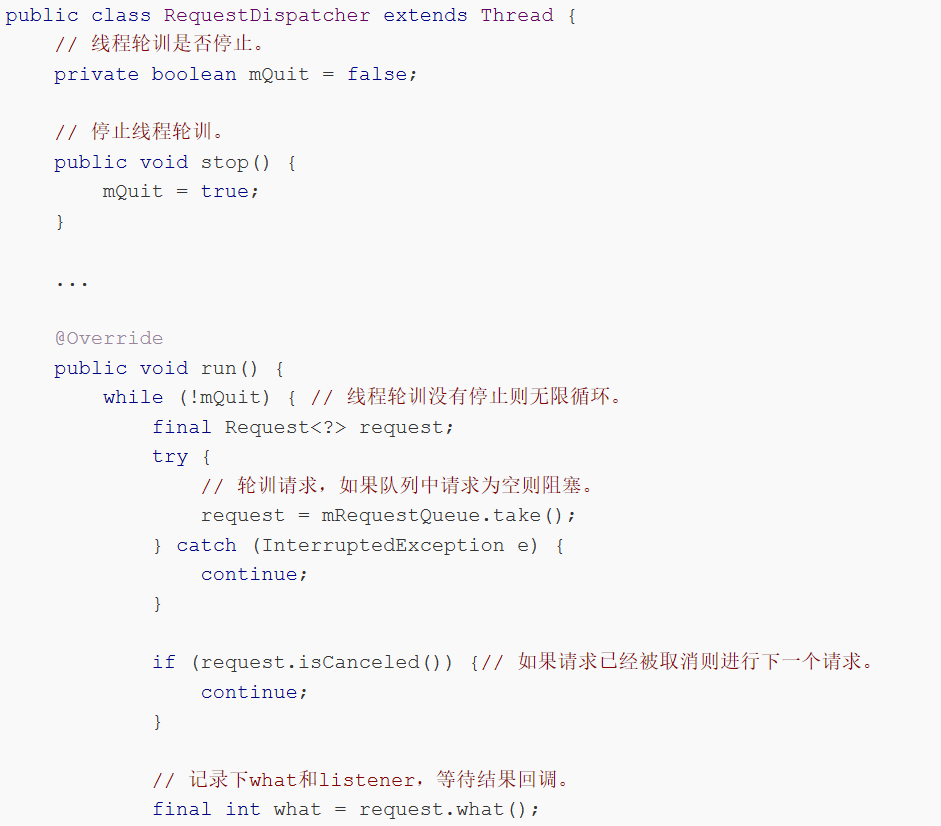
\*/

public void cancelAll() {}

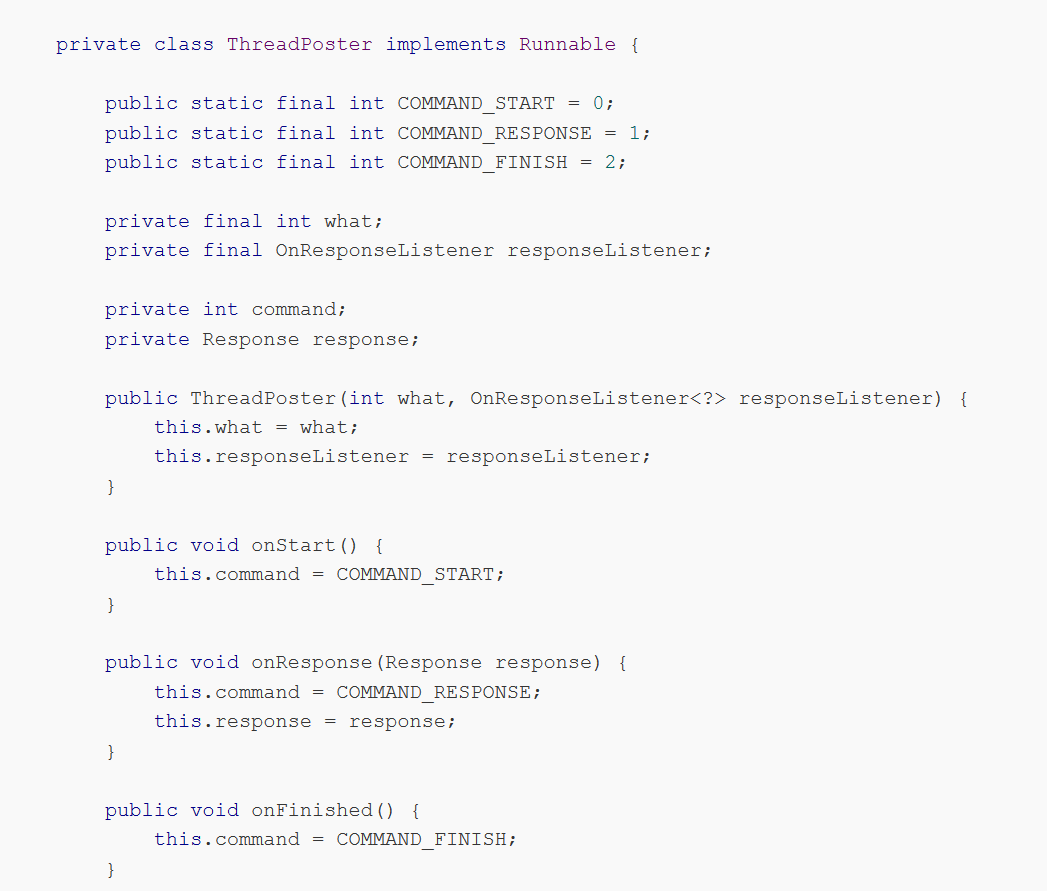
}

在构造方法中创建了一个threadPoolSize大小的RequestDispatcher数组，调用start()的时候为数组的每一个index赋值一个真正的RequestDispatcher线程，并启动这个线程去轮训Queue，同时在创建RequestDispatcher的时候把我们创建队列的时候的IRestParser穿进去了，说明真正的调用网络还是在RequestDispatcher这个子线程中，具体怎么调用并处理接着看下文

2、RequestDispatcher：NoHttp异步请求中最重要的类之一









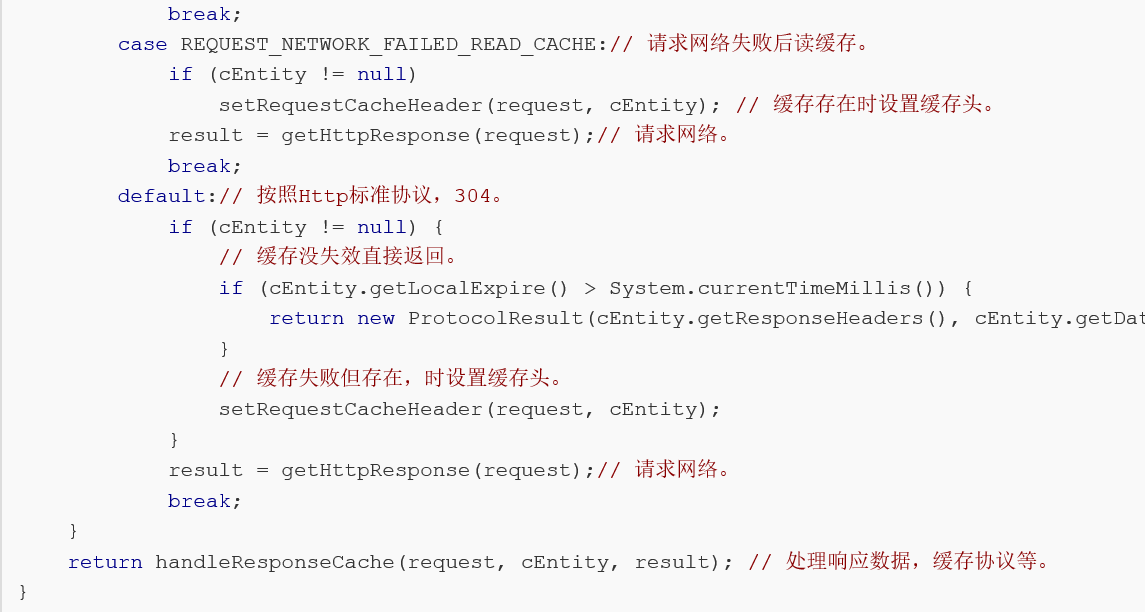
RequestDispatcher在线程没有被标志停止的情况下会一直循环调用Queue.take()轮训队列中的请求，如果线程中没有请求，由于Queue.take()的特性，这个子线程会处于阻塞状态，当然这不会使APP卡顿，因为它在子线程。当它每拿到一个Request先会判断请求是否被取消，如果是取消了的则去轮训下一个请求，如果没有取消会利用Handler发送一个Runnable回调Listener.onStart()方法通知主线程请求。

4、针对以后移动端开发需求需要用到http缓存技术，简单了解下Nohttp的几种缓存模式

NoHttp的几种缓存模式：

* 只读缓存
* 只请求缓存
* 先读缓存，没有缓存再请求网络
* 先请求网络，请求网络失败再读取缓存
* 按照Http标准协议，重定向缓存机制





1. 总结与讨论

在我们现有的项目中对于nohttp的应用还不是太多，Nohttp作为一个android比较实用，且应用方便快捷的网络框架，个人认为对于以后的开发是比较实用的。