# 1、OkHttp同步请求和异步请求的区别

OKhttp中请求任务的管理是由dispatcher来负责的,负责请求的分发的发起。实际执行请求的是ConnectionPool。

## 1.1同步请求

同一时刻只能有一个任务发起,synchronized关键字锁住了整个代码，那么如果当前OkhttpClient已经执行了一个同步任务，如果这个任务没有释放锁，那么新发起的请求将被阻塞，直到当前任务释放锁。

@Override public Response execute() throws IOException {

//同一时刻只能有一个任务执行 因为是阻塞式的 由synchronized关键字锁住

synchronized (this) {

if (executed) throw new IllegalStateException("Already Executed");

executed = true;

}

captureCallStackTrace();

try {

client.dispatcher().executed(this);

Response result = getResponseWithInterceptorChain();

if (result == null) throw new IOException("Canceled");

return result;

} finally {

client.dispatcher().finished(this);

}

}

## 1.2异步请求

同一时刻可以发起多个请求，因为异步请求每一个都是在一个独立的线程，由两个队列管理，并且synchronized只锁住了代码校验是否执行的部分。

@Override public void enqueue(Callback responseCallback) {

synchronized (this) {

if (executed) throw new IllegalStateException("Already Executed");

executed = true;

}

//异步请求同一时刻可以有多个任务执行，由两个队列管理

captureCallStackTrace();

client.dispatcher().enqueue(new AsyncCall(responseCallback));

}

同步是指：发送方发出数据后，等接收方发回响应以后才发下一个数据包的通讯方式。

异步是指：发送方发出数据后，不等接收方发回响应，接着发送下个数据包的通讯方式。

同步：提交请求->等待服务器处理->处理完毕返回 这个期间客户端浏览器不能干任何事。

异步: 请求通过事件触发->服务器处理（这是浏览器仍然可以作其他事情）->处理完毕。

# 2、OkHttp2.x用法全解析

## 2.1使用前准备

Android Studio 配置gradle：

compile 'com.squareup.okhttp:okhttp:2.7.5'

compile 'com.squareup.okio:okio:1.7.0'

## 2.2异步GET请求 enqueue()

private void getAsynHttp() {

//创建okHttpClient对象

OkHttpClient mOkHttpClient = new OkHttpClient();

final Request request = new Request.Builder()

.url("http://www.baidu.com")

.build();

Call call = mOkHttpClient.newCall(request);

call.enqueue(new Callback() {

@Override

public void onFailure(Request request, IOException e) {

}

@Override

public void onResponse(final Response response) throws IOException {

String str = response.body().string();

Log.i("wangshu", str);

runOnUiThread(new Runnable() {

@Override

public void run() {

Toast.makeText(getApplication(), "请求成功",

Toast.LENGTH\_SHORT).show();

}

});

}

});

}

运行程序log打印出来的是百度首页的html文件，基本的步骤很简单，就是创建OkHttpClient、Request和Call，最后调用Call的enqueue()方法。但是每次这么写肯定是很麻烦，肯定是要进行封装的。需要注意的是onResponse回调并不是在UI线程。

## 2.3同步GET请求 execute()

private String getSyncHttp() throws IOException{

OkHttpClient mOkHttpClient = new OkHttpClient();

//创建请求Request

final Request request = new Request.Builder()

.url("http://www.baidu.com")

.build();

Call call = mOkHttpClient.newCall(request);

Response mResponse=call.execute();

if (mResponse.isSuccessful()) {

return mResponse.body().string();

} else {

throw new IOException("Unexpected code " + mResponse);

}

}

同步Get请求和异步调用区别就是调用了call的execute()方法。

## 2.4异步POST请求 RequestBody

private void postAsynHttp() {

OkHttpClient mOkHttpClient = new OkHttpClient();

**RequestBody formBody** = new FormEncodingBuilder()

.add("size", "10")

.build();

Request request = new Request.Builder()

.url("http://api.1-blog.com/biz/bizserver/article/list.do")

.post(formBody)

.build();

Call call = mOkHttpClient.newCall(request);

call.enqueue(new Callback() {

@Override

public void onFailure(Request request, IOException e) {

}

@Override

public void onResponse(Response response) throws IOException {

String str = response.body().string();

Log.i("wangshu", str);

runOnUiThread(new Runnable() {

@Override

public void run() {

Toast.makeText(getApplicationContext(), "请求成功", Toast.LENGTH\_SHORT).show();

}

});

}

});

}

post与get不同的就是要要创建RequestBody并传进Request中，同样onResponse回调不是在UI线程。

## 2.5请求缓存设置

首先我们设置缓存路径和大小并设置给OkHttpClient：

mOkHttpClient = new OkHttpClient();

File sdcache = getExternalCacheDir();

int cacheSize = 10 \* 1024 \* 1024;

mOkHttpClient.setCache(new Cache(sdcache.getAbsoluteFile(), cacheSize));

接下来异步GET请求baidu：

private void getAsynHttp() {

//创建请求Request

final Request request = new Request.Builder()

.url("http://www.baidu.com")

.build();

Call call = mOkHttpClient.newCall(request);

call.enqueue(new Callback() {

@Override

public void onFailure(Request request, IOException e) {

}

@Override

public void onResponse(final Response response) throws IOException {

if (null != response.cacheResponse()) {

String str = response.cacheResponse().toString();

Log.i("wangshu", "cache---" + str);

} else {

response.body().string();

String str=response.networkResponse().toString();

Log.i("wangshu", "network---" + str);

}

runOnUiThread(new Runnable() {

@Override

public void run() {

Toast.makeText(getApplicationContext(), "请求成功", Toast.LENGTH\_SHORT).show();

}

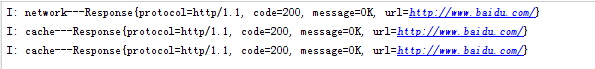
});

}

});

}

第一次请求会请求网络得到数据，第二次以及后面的请求则会从缓存中取出数据：



当然也有种情况是有的请求每次都需要最新的数据，则在创建Request，来设置cacheControl为“CacheControl.FORCE\_NETWORK”，用来表示请求会一直请求网络得到数据：

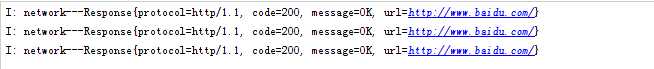
final Request request = new Request.Builder()

.url("http://www.baidu.com")

.cacheControl(CacheControl.FORCE\_NETWORK)

.build();

运行程序结果为：



## 2.6设置超时时间

另外我们也需要设置超时的时间用来处理各种网络超时的情况，超时的原因可能是网络问题也可能是服务器响应慢等问题，OkHttp当然不会忽略这一点，它支持连接、读取和写入超时的时间设置：

mOkHttpClient = new OkHttpClient();

mOkHttpClient.setConnectTimeout(15, TimeUnit.SECONDS);

mOkHttpClient.setWriteTimeout(20, TimeUnit.SECONDS);

mOkHttpClient.setReadTimeout(20, TimeUnit.SECONDS);

## 2.7取消请求

（1）使用call.cancel()可以立即停止掉一个正在执行的call。如果一个线程正在写请求或者读响应）将会引发IOException。当用户离开一个应用时或者跳到其他界面时，使用Call.cancel()可以节约网络资源，另外不管同步还是异步的call都可以取消。

（2）也可以通过tags来同时取消多个请求。当你构建一请求时，使用RequestBuilder.tag(tag)来分配一个标签。之后你就可以用OkHttpClient.cancel(tag)来取消所有带有这个tag的call。

为了模拟这个场景我们首先创建一个定时的线程池：

private ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);

接下来的代码为：

private void cancel(){

final Request request = new Request.Builder()

.url("http://www.baidu.com")

.cacheControl(CacheControl.FORCE\_NETWORK)

.build();

Call call=null;

call = mOkHttpClient.newCall(request);

final Call finalCall = call;

//100毫秒后取消call

executor.schedule(new Runnable() {

@Override public void run() {

finalCall.cancel();

}

}, 100, TimeUnit.MILLISECONDS);

call.enqueue(new Callback() {

@Override

public void onFailure(Request request, IOException e) {

}

@Override

public void onResponse(final Response response) {

if (null != response.cacheResponse()) {

String str = response.cacheResponse().toString();

Log.i("wangshu", "cache---" + str);

} else {

try {

response.body().string();

} catch (IOException e) {

Log.i("wangshu", "IOException");

e.printStackTrace();

}

String str = response.networkResponse().toString();

Log.i("wangshu", "network---" + str);

}

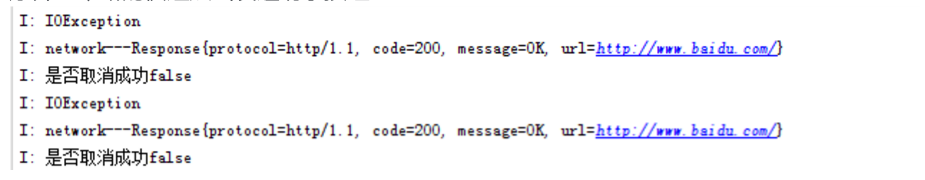
}

});

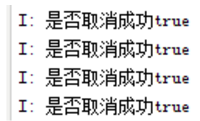
Log.i("wangshu", "是否取消成功"+call.isCanceled());

}

100毫秒后调用call.cancel()，为了能让请求耗时，我们设置每次请求都要请求网络，运行程序并且不断的快速点击发送请求按钮：



很明显每次cancel()都失败了，仍旧成功的访问了网络，在cancel()时已经有读写操作了所以会报IOException。每隔100毫秒来调用call.cancel()显然时间间隔太长，我们设置为1毫秒并不断的快速的点击发送请求按钮：



没有请求网络的log，几乎每次都取消成功了。

## 2.8关于封装

如果每次请求网络都需要写重复的代码绝对是令人头疼的，网上也有很多对OkHttp封装的优秀开源项目，功能也非常强大，封装的意义就在于更加方便的使用，具有拓展性，但是对OkHttp封装最需要解决的是以下的两点：

**避免重复代码调用**

**将请求结果回调改为UI线程**

根据以上两点，我们也简单封装一下，在此只是举个例子，如果想要使用OkHttp封装的开源库，推荐使用OkHttpFinal。

首先呢我们写一个抽象类用于请求回调：

public abstract class ResultCallback<T>

{

public abstract void onError(Request request, Exception e);

public abstract void onResponse(Response response);

}

接下来封装OkHttp，并实现了异步GET请求：

public class OkHttpEngine {

private volatile static OkHttpEngine mInstance;

private OkHttpClient mOkHttpClient;

private Handler mHandler;

public static OkHttpEngine getInstance() {

if (mInstance == null) {

synchronized (OkHttpEngine.class) {

if (mInstance == null) {

mInstance = new OkHttpEngine();

}

}

}

return mInstance;

}

private OkHttpEngine() {

mOkHttpClient = new OkHttpClient();

mOkHttpClient.setConnectTimeout(15, TimeUnit.SECONDS);

mOkHttpClient.setWriteTimeout(20, TimeUnit.SECONDS);

mOkHttpClient.setReadTimeout(20, TimeUnit.SECONDS);

mHandler = new Handler();

}

public OkHttpEngine setCache(Context mContext) {

File sdcache = mContext.getExternalCacheDir();

int cacheSize = 10 \* 1024 \* 1024;

mOkHttpClient.setCache(new Cache(sdcache.getAbsoluteFile(), cacheSize));

return mInstance;

}

/\*\*

\* 异步get请求

\* @param url

\* @param callback

\*/

public void getAsynHttp(String url, ResultCallback callback) {

final Request request = new Request.Builder()

.url(url)

.build();

Call call = mOkHttpClient.newCall(request);

dealResult(call, callback);

}

private void dealResult(Call call, final ResultCallback callback) {

call.enqueue(new Callback() {

@Override

public void onFailure(Request request, IOException e) {

sendFailedCallback(request, e, callback);

}

@Override

public void onResponse(final Response response) throws IOException {

sendSuccessCallback(response, callback);

}

private void sendSuccessCallback(final Response object, final ResultCallback callback) {

mHandler.post(new Runnable() {

@Override

public void run() {

if (callback != null) {

callback.onResponse(object);

}

}

});

}

private void sendFailedCallback(final Request request, final Exception e, final ResultCallback callback) {

mHandler.post(new Runnable() {

@Override

public void run() {

if (callback != null)

callback.onError(request, e);

}

});

}

});

}

}

原理很简单就是，写一个双重检查模式的单例。在开始创建的时候配置好OkHttpClient，在请求网络的时候用Handler将请求的结果回调给UI线程。

最后调用这个OkHttpEngine的getAsynHttp()方法：

OkHttpEngine.getInstance().getAsynHttp("http://www.baidu.com", new ResultCallback() {

@Override

public void onError(Request request, Exception e) {

}

@Override

public void onResponse(Response response) {

String str = response.networkResponse().toString();

Log.i("wangshu", str);

Toast.makeText(getApplicationContext(), "请求成功",

Toast.LENGTH\_SHORT).show();

}

});

使用起来简单多了，而且请求结果回调是在UI线程的。下一篇我们会讲到OkHttp3，来看看它与OkHttp2.x之间的使用方式上有什么区别。

# 3、OkHttp3用法全解析

## 3.1使用前准备

Android Studio 配置gradle：

compile 'com.squareup.okhttp3:okhttp:3.2.0'

compile 'com.squareup.okio:okio:1.7.0'

添加网络权限：

<uses-permission android:name="android.permission.INTERNET"/>

## 3.2异步GET请求

private void getAsynHttp() {

mOkHttpClient=new OkHttpClient();

Request.Builder requestBuilder = new Request.Builder().url("http://www.baidu.com");

//可以省略，默认是GET请求

requestBuilder.method("GET",null);

Request request = requestBuilder.build();

Call mcall= mOkHttpClient.newCall(request);

mcall.enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) {

}

@Override

public void onResponse(Call call, Response response) throws IOException {

if (null != response.cacheResponse()) {

String str = response.cacheResponse().toString();

Log.i("wangshu", "cache---" + str);

} else {

response.body().string();

String str = response.networkResponse().toString();

Log.i("wangshu", "network---" + str);

}

runOnUiThread(new Runnable() {

@Override

public void run() {

Toast.makeText(getApplicationContext(), "请求成功", Toast.LENGTH\_SHORT).show();

}

});

}

});

}

与2.x版本并没有什么不同，比较郁闷的是回调仍然不在UI线程。

## 3.3异步POST请求

OkHttp3异步POST请求和OkHttp2.x有一些差别就是没有FormEncodingBuilder这个类，替代它的是功能更加强大的FormBody：

private void postAsynHttp() {

mOkHttpClient=new OkHttpClient();

RequestBody formBody = new FormBody.Builder()

.add("size", "10")

.build();

Request request = new Request.Builder()

.url("http://api.1-blog.com/biz/bizserver/article/list.do")

.post(formBody)

.build();

Call call = mOkHttpClient.newCall(request);

call.enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) {

}

@Override

public void onResponse(Call call, Response response) throws IOException {

String str = response.body().string();

Log.i("wangshu", str);

runOnUiThread(new Runnable() {

@Override

public void run() {

Toast.makeText(getApplicationContext(), "请求成功", Toast.LENGTH\_SHORT).show();

}

});

}

});

}

## 3.4异步上传文件

上传文件本身也是一个POST请求，上一篇没有讲，这里我们补上。首先定义上传文件类型：

public static final MediaType MEDIA\_TYPE\_MARKDOWN

= MediaType.parse("text/x-markdown; charset=utf-8");

将sdcard根目录的wangshu.txt文件上传到服务器上：

private void postAsynFile() {

mOkHttpClient=new OkHttpClient();

File file = new File("/sdcard/wangshu.txt");

Request request = new Request.Builder()

.url("https://api.github.com/markdown/raw")

.post(RequestBody.create(MEDIA\_TYPE\_MARKDOWN, file))

.build();

mOkHttpClient.newCall(request).enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) {

}

@Override

public void onResponse(Call call, Response response) throws IOException {

Log.i("wangshu",response.body().string());

}

});

}

当然如果想要改为同步的上传文件只要调用 mOkHttpClient.newCall(request).execute()就可以了。

当然不要忘了添加如下权限：

<uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"/>

<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>

## 3.5异步下载文件

下载文件同样在上一篇没有讲到，实现起来比较简单，在这里下载一张图片，我们得到Response后将流写进我们指定的图片文件中就可以了。

private void downAsynFile() {

mOkHttpClient = new OkHttpClient();

String url = "http://img.my.csdn.net/uploads/201603/26/1458988468\_5804.jpg";

Request request = new Request.Builder().url(url).build();

mOkHttpClient.newCall(request).enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) {

}

@Override

public void onResponse(Call call, Response response) {

InputStream inputStream = response.body().byteStream();

FileOutputStream fileOutputStream = null;

try {

fileOutputStream = new FileOutputStream(new

File("/sdcard/wangshu.jpg"));

byte[] buffer = new byte[2048];

int len = 0;

while ((len = inputStream.read(buffer)) != -1) {

fileOutputStream.write(buffer, 0, len);

}

fileOutputStream.flush();

} catch (IOException e) {

Log.i("wangshu", "IOException");

e.printStackTrace();

}

Log.d("wangshu", "文件下载成功");

}

});

}

## 3.6异步上传Multipart文件

这种场景很常用，我们有时会上传文件同时还需要传其他类型的字段，OkHttp3实现起来很简单，需要注意的是没有服务器接收我这个Multipart文件，所以这里只是举个例子，具体的应用还要结合实际工作中对应的服务器。

首先定义上传文件类型：

private static final MediaType MEDIA\_TYPE\_PNG = MediaType.parse("image/png");

private void sendMultipart(){

mOkHttpClient = new OkHttpClient();

RequestBody requestBody = new MultipartBody.Builder()

.setType(MultipartBody.FORM)

.addFormDataPart("title", "wangshu")

.addFormDataPart("image", "wangshu.jpg",

RequestBody.create(MEDIA\_TYPE\_PNG, new File("/sdcard/wangshu.jpg")))

.build();

Request request = new Request.Builder()

.header("Authorization", "Client-ID " + "...")

.url("https://api.imgur.com/3/image")

.post(requestBody)

.build();

mOkHttpClient.newCall(request).enqueue(new Callback() {

@Override

public void onFailure(Call call, IOException e) {

}

@Override

public void onResponse(Call call, Response response) throws IOException {

Log.i("wangshu", response.body().string());

}

});

}

## 3.7设置超时时间和缓存

和OkHttp2.x有区别的是不能通过OkHttpClient直接设置超时时间和缓存了，而是通过OkHttpClient.Builder来设置，通过builder配置好OkHttpClient后用builder.build()来返回OkHttpClient，所以我们通常不会调用new OkHttpClient()来得到OkHttpClient，而是通过builder.build()：

File sdcache = getExternalCacheDir();

int cacheSize = 10 \* 1024 \* 1024;

OkHttpClient.Builder builder = new OkHttpClient.Builder()

.connectTimeout(15, TimeUnit.SECONDS)

.writeTimeout(20, TimeUnit.SECONDS)

.readTimeout(20, TimeUnit.SECONDS)

.cache(new Cache(sdcache.getAbsoluteFile(), cacheSize));

OkHttpClient mOkHttpClient=builder.build();

## 3.8关于取消请求和封装

取消请求仍旧可以调用call.cancel()，这个没有变化，不明白的可以查看上一篇文章Android网络编程（五）OkHttp2.x用法全解析，这里就不赘述了，封装上一篇也讲过仍旧推荐OkHttpFinal，它目前是基于OkHttp3来进行封装的。