Android移动开发课程

实验指导书-NetWork部分

【实验目的】

初步了解Android 的 Notification 组件

【实验设计】

本次实验包括四个验证实验和一个自选实验。

其中验证实验为个人实验，已经提供源代码、操作步骤、实验指导视频

由于NetWork涉及网络通讯，因此本实验提供了一个Jsp编写的服务器，源代码已经给出，可以在Tomcat下运行。

要求：

按照实验步骤完成，

以个人为单位提交，提交实验报告一份，实验报告需要回答指导书中问题。

自选实验为小组实验，建议2-4人组队，提供了参考选题和参考资料。

要求：

以小组为单位提交，提交实验报告一份、源代码一份、可以执行的APK文件一个。

【实验内容】

在本次实验中，我们只考虑使用HTTP协议进行通讯，如果想探讨利用Socket进行通讯可以参见附录。

在过去，Android发去http请求有HttpURLConnection和HttpClient两种方式，在Android4.4中，google已经开始将源码中的HttpURLConnection替换为OkHttp，在Android6.0中移除了HttpClient，HttpURLConnection也是java的标准网络通讯方式。

网络请求库- 对比

网络请求库 / 对比	android-async-http	Volley	OkHttp	Retrofit
功能	• 基于HttpClient • 在 UI 线程外、异步进行Http请求 • 在匿名回调中处理请求结果 callback使用了Android的Handler发送消息机制在创建它的线程中执行 • 自动智能请求重试 • 持久化cookie存储 保存cookie到你的应用程序的SharedPreferences	• 基于HttpUrlConnection • 封装了UIL图片加载框架，支持图片加载 • 网络请求的排序、优先级处理 • 缓存 • 多级别取消请求 • Activity和生命周期的联动（Activity结束时同时取消所有网络请求）	• 高性能Http请求库 可把它理解成是一个封装之后的类似HttpUrlConnection 的一个东西，属于同级并不是基于上述二者； • 支持 SPDY，共享同一个Socket来处理同一个服务器的所有请求； • 支持http 2.0、websocket • 支持同步、异步 • 封装了线程池、数据转换、参数使用、错误处理等 • 无缝的支持GZIP来减少数据流量 • 缓存响应数据来减少重复的网络请求 • 能从很多常用的连接问题中自动恢复 • 解决了代理服务器问题和SSL握手失败问题	• 基于OkHttp • RESTful Api设计风格 • 支持同步、异步； • 通过注解配置请求 包括请求方法，请求参数，请求头，返回值等 • 可以搭配多种Converter将获得的数据解析&序列化 支持Gson（默认）、 Jackson、Protobuf等 • 提供对 RxJava 的支持
性能	1. 作者已经停止对该项目维护； 2. Android5.0后不推荐用HttpClient； 所以不推荐在项目中使用了。	• 可拓展性好：可支持HttpClient、HttpUrlConnection和OkHttp	• 基于 NIO 和 Okio，所以性能更好：请求、处理速度快 （IO：阻塞式；NIO：非阻塞式；Okio 是 Square 公司基于 IO 和 NIO 基础上做的一个更简单、高效处理数据流的一个库）	• 性能最好，处理最快； • 扩展性差 高度封装所带来的必然后果：解析数据都是使用的统一的converter，如果服务器不能给出统一的API的形式，将很难进行处理。
开发者使用		• 封装性好：简单易用	• Api调用更加简单、方便； • 使用时需要进行多一层封装	• 简洁易用（RestfulAPI设计风格） • 代码简化（更加高度的封装性和注解用法） • 解耦的更彻底、职责更细分 • 易与其他框架联合使用（RxJava） • 使用方法较多，原理复杂，存在一定门槛
应用场景		• 适合轻量级网络交互：网络请求频繁、传输数据量小； • 不能进行大数据量的网络操作（比如下载视频、音频），所以不适合用来上传文件。	重量级网络交互场景：网络请求频繁、传输数据量大 （其实会更推荐Retrofit，反正Retrofit是基于Okhttp的）	任何场景下优先选择，特别是： 后台Api遵循RESTful的风格&项目中有使用RxJava；
备注		Volley的request和response都是把数据放到byte数组里，不支持输入输出流，把数据放到数组中，如果大文件多了，数组就会非常的大且多，消耗内存，所以不如直接返回Stream那样具备可操作性，比如下载一个大文件，不可能把整个文件都缓存到内存之后再写到文件里。	Android4.4的源码中可以看到HttpURLConnection已经替换成OkHttp实现了。所以我们更有理由相信OkHttp的强大。	

由于Retrofix是基于OkHttp的，经过考量，本次实验以Okhttp3为例。

在ExpConfig下是服务器信息的配置，请保证服务器能够正常运行以后再进行本次实验，注意修改服务器信息。

验证实验一：认识okhttp

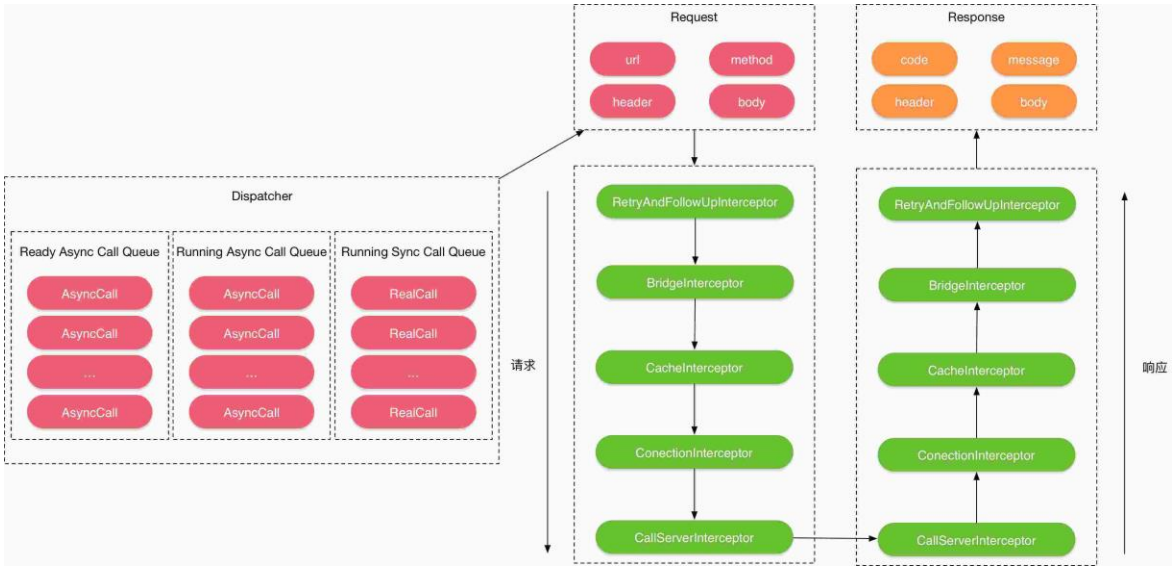
依赖文件：implementation 'com.squareup.okhttp3:okhttp:3.12.0'

该依赖包括okhttp库和okio库。

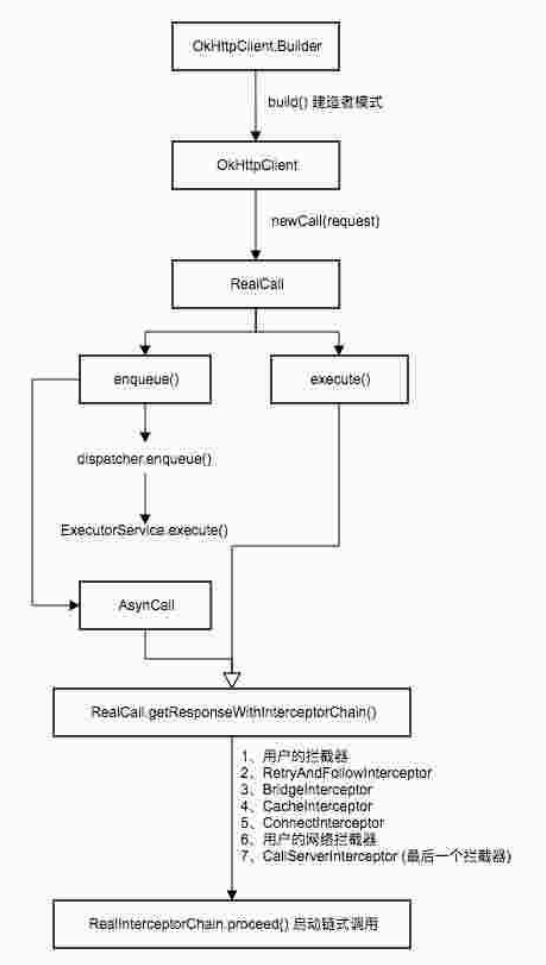
现在的Android要求使用HTTPS进行通讯，如果不使用HTTPS进行通讯，会引发异常：“CLEARTEXT communication to exmaple.com not permitted by network security policy”

参考：https://blog.csdn.net/auccy/article/details/106214640

关于OkHttp的请求流程



关于OkHttp的工作流程:



了解OkHttpClient的创建：

```
OkHttpClient okHttpClient = new OkHttpClient.Builder().build();
```

了解Request的构造：

```
Request request = new Request.Builder().url(Your URL).build();
```

了解Call的构造：

```
Call call = okHttpClient.newCall(request);
```

了解execute的使用和enqueue的使用

Execute会阻塞线程，等请求完成以后会返回一个Response

Enquene会使用OkHttp的线程池，需要接收一个回调接口CallBack

1. 了解在MainThread中进行网络请求会引发异常，点击“在MainThread中进行网络操作会引发异常”按钮，查看Logcat情况（过滤条件：Level为Info，关键词为Exp1）

2. 使用execute需要新开线程，点击“同步请求,采用execute”按钮，查看Logcat情况（过滤条件：Level为Info，关键词为Exp1）

3. 使用enqueue不需要新开线程，点击“异步请求,采用enqueue”按钮，查看Logcat情况（过滤条件：Level为Info，关键词为Exp1）

本验证实验提交内容：

1. 提交观察到的结果

验证实验二：了解Get请求和Post请求，了解OkHttp里Get请求的构造

Get请求和Post请求最直观的区别就是GET把参数包含在URL中，POST通过request body传递参数。

Get请求和Post请求的区别：

GET在浏览器回退时是无害的，而POST会再次提交请求。

GET产生的URL地址可以被Bookmark，而POST不可以。

GET请求会被浏览器主动cache，而POST不会，除非手动设置。

GET请求只能进行url编码，而POST支持多种编码方式。

GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。

GET请求在URL中传送的参数是有长度限制的，而POST没有。

对参数的数据类型，GET只接受ASCII字符，而POST没有限制。

GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。

GET参数通过URL传递，POST放在Request body中。

1. 了解Get请求的构造

Get请求把参数放在URL里，最简单的构造就是构造URL

点击“利用拼接进行get请求”按钮，观察源码和Logcat（过滤条件：Level为Info，关键词为Exp2）

2. 利用HttpUrl进行构造

点击“利用httpurl进行get请求”按钮，观察源码和Logcat（过滤条件：Level为

Info，关键词为Exp2）

3. 由于网络请求不在MainThread里，所以不能更新UI

   点击“不能在非UI线程中修改UI组件”按钮，观察Logcat（过滤条件：Level为Info，关键词为Exp2）

4. 利用RunOnUiThread切换线程

   点击“利用RunOnThread修改UI组件”按钮，观察界面和Logcat（过滤条件：Level为Info，关键词为Exp2

本验证实验提交内容：

1. 提交观察到的结果

验证实验三：了解OkHttp里Post请求的构造

1. 了解Post请求的构造

   Post请求把参数放在Body里，构造就是构造Request Body

   点击“利用post提交键值对”按钮，观察源码和Logcat（过滤条件：Level为Info，关键词为Exp3）

2. 了解利用Post请求提交小文件

   提交文件也是构造Request Body，利用

   RequestBody.create(MediaType.parse("application/octet-stream"), text)构造Body

   其中text是一个InputStream，这里使用SAF框架选择文件获取文件的Uri，并且通过getContentResolver().openInputStream(uri)获取InputStream

   点击“利用post提交小型文件”按钮，观察源码和Logcat（过滤条件：Level为Info，关键词为Exp3）

本验证实验提交内容：

1. 提交观察到的结果

验证实验四：了解数据解析

服务器返回的最常见的是XML和Json，对于XML和Json的解析有很多方法，以下提供了XML的Pull解析和Json的JsonObject解析。

有兴趣的同学可以了解XML的SAX解析和Json的Gson

1. 点击“XML解析按钮”，观察Logcat（过滤条件：Level为Info，关键词为Exp4）

2. 点击“Json解析按钮”，观察Logcat（过滤条件：Level为Info，关键词为Exp4）

本验证实验提交内容：

1. 提交观察到的结果

自选实验：

下载是极其常见的功能，涉及到网络通讯、多线程、服务等多个方面，由于比较复杂，提供了一个简单的Demo，位于Download下。

DownloadActity主要是用于用户控制下载行为，然后使用绑定模式启动服务，利用Binder控制Service。

DownloadService中主要是对AsyncTask的控制，并且使用了Notification反馈下载情况。获得notificationManager类管理通知，考虑到Android8.0以后需要通知渠道，在createNotificationChannel中创建通知渠道,由于这些通知除了下载进度不一样以外都相同，使用getNotification创建通知。异步通讯使用接口DownloadListener，在binder中实例化这个接口以便于AsyncTask回调，并且提供startDownload、pauseDownload、cancelDownload方法供Acitivty调用。

DownloadTask继承于AsyncTask，主要用于异步执行网络请求。通过Url获得下载文件名，在下载目录新建文件或打开文件，如果文件存在，获得文件大小。接着我们发起一次网络请求，获的需要下载的文件大小，比较差异，判断是否下载完成过。然后我们再次发起网络请求，通过在header加入参数以实现断点续传，使用client.newCall(request).execute()获得response，获得字节流（Response.body().byteStream），然后将字节流写入文件，并且通过PublishProgress更改下载进度，onProgressUpdate中使用回调接口listener的onProgress方法进而执行在Binder里listener的onProgress，从而更新通知。最后关闭文件流、网络请求流、文件，返回状态（成功、失败、取消）并在onPostExecute中调用listener对应的方法。

本Demo仅供参考。