### 基础网络、新技术

1. 框架（图片 网络 实现）

－Volly、HttpClient、HttpUrlClient、Okhttp 区别

答：HttpClient 繁琐，且已被抛弃

HttpUrlConnection 2.2以下有重大bug，可读的Inputstream调用close()方法，会导致连接池失败，无法复用。Android 4.0中增加了缓存机制.

Okhttp 实现了连接池、gziping 、缓存，使用Okio实现数据访问存储。

Volley是一个简化网络任务的库。他负责处理请求，加载，缓存，线程，同步等问题。它可以处理JSON，图片，缓存，文本源，支持一定程度的自定义。

（https://www.jianshu.com/p/1b088dd37a2d）

－http https 架构

－图片框架库 使用 比较

Fresco

Universal-Image-Loader

Picasso

－使用：<https://www.jianshu.com/p/c68a3b9ca07a>

Picasso.with(context)

.load(url)

.placeholder(R.drawable.default)

.error(R.drawable.errorimage)

.noFade()

.onlyScaleDown()

.resize(300,400)

.resizeDimen(R.dimen.image\_width, R.dimen.image\_height)

.centerCrop()

.fit()

.rotate(180, 200, 100)

.transform(new BlueTransformation(this))

.priority(Picasso.Priority.HIGH)

.tag("photoTag")

.memoryPolicy(MemoryPolicy.NO\_CACHE, MemoryPolicy.NO\_STORE)

.networkPolicy(NetworkPolicy.NO\_CACHE)

.networkPolicy(NetworkPolicy.OFFLINE)

.setIndicatorsEnabled(true)

.setLoggingEnable(true)

.into(imageview)

.into(new Target(){

public void onBitmapLoaded(Bitmap bitmap, Picasso.LoadedFrom from) {

}

@Override

public void onBitmapFailed(Drawable errorDrawable) {

}

@Override

public void onPrepareLoad(Drawable placeHolderDrawable) {

}

})

Picasso.Builder builder = new Picasso.Buidler(this)

builder.download(new CustomerLoader())

builder.memoryCache(new Lurcache(5\*1024));

builder.executor(Executors.newFidxedThreadPool(8));

Picasso pi = builder.buil();

Picasso.setSingletonInstance(pi);

**内存缓存是使用的LRU 策略的缓存实现，它的大小是内存大小的15%,可以自定义它的大小。**

**磁盘缓存是磁盘容量的2%但是不超过50M,不少于5M。**

**处理一个请求的时候，按照这个顺讯检查：memory->disk->network。**

Glide

——

<https://www.jianshu.com/p/93198a80bda7>

－长连 短连 socket/http协议 参数 ／Tcp三次握手具体实现

答：

－异步 Future

答：

* OkHttp缓存机制

答：服务端缓存机制（了解）：cache-control、Etag、If-None-Match

Etag对应请求的资源在服务器中的唯一标识（具体规则由服务器决定），比如一张图片，它在服务器中的标识为ETag: W/”ACXbWXd1n0CGMtAd65PcoA==”。

If-None-Match 如果浏览器在Cache-Control:max-age=60设置的时间超时后，发现消息头中还设置了Etag值。然后，浏览器会再次向服务器请求数据并添加In-None-Match消息头，它的值就是之前Etag值。服务器通过Etag来定位资源文件，根据它是否更新的情况给浏览器返回200或者是304。

Etag机制比Last-Modified精确度更高，如果两者同时设置的话，Etag优先级更高。

OKHttp cache Response有两种类型：CacheResponse、NetworkResponse

手动添加拦截器，设置超时时间



<http://blog.csdn.net/briblue/article/details/52920531>

<https://www.jianshu.com/p/797d2d442d27>

大概流程：

<https://www.jianshu.com/p/00d281c226f6>

2、RxJava 线程切换原理 lift 操作符

Backpressure的几种策略 Buffer\Drop\Lastest

lift原理：创建新的Subscriber，Obserable

线程切换的原理

subscribeOn() 设置被观察者的所在的线程，只有第一次有效

observeOn()改变调用它之后代码的线程

简单说:最原始的subscribeOn()—调用了—-create(new OperatorSubscribeOn<T>(this, scheduler))—-创建一个代理被观察者—->OperatorSubscribeOn()中实现了call()方法—->call()方法中调用了NewThreadWorker.schedule(Action0)—-Action0被包装称一个RUnnable对象,然后schedule()方法内部使用了线程池,创建一个新的线程,并将包装的Runnable对象传递进去,这样就实现了线程的切换

步骤:

A、原始被观察者调用subscribeOn()方法准备切换线程,(这时候还没切换呢.)产生一个代理被观察者

B、原始订阅者订阅代理被观察者(明面代码上你能看得到的)

C、代理被观察者的onSubscribe.call()方法执行,提供了一个Runnable对象,也就是线程已经被切换了

D、新线程中产生一个新的代理观察者,代理观察者订阅原始被观察者(接下来的动作也都是在新线程中执行)

E、原始被观察者发射数据,这个动作已经是在新线程中执行了

F、代理观察者收到数据,再将数据转发给原始观察者

3、设计模式 复习

设计模式 装饰者模式、单例模式、适配器模式、工厂模式

代理模式和装饰模式区别

团长的PPT android中的应用

https://www.jianshu.com/p/fb8d21978e38

4、抽象类和接口 设计的区别

5、MVC、MVVM、MVP 三种模式 复习

ViewModel层的主要责任是表现逻辑和状态，即ViewModel层是连接View层和Model层的。Model层的一些业务逻辑的状态是需要通过ViewModel层暴露给View层来反映给软件使用者的。

6、kotlin 编程

学习视频，了解

7、ReactNative 学习

搭建环境，再深入一点 —— Note.js

8、通信加密 对称 非对称 sha1

9、图片在内存的大小控制，影响因素

图片下载控件 逻辑

图片的内存影响

RGB\_8888 每个图片像素占据4个字节

RGB\_565每个图片像素占据2个字节

Android中有四种，分别是：

ALPHA\_8：每个像素占用1byte内存

ARGB\_4444:每个像素占用2byte内存

ARGB\_8888:每个像素占用4byte内存

RGB\_565:每个像素占用2byte内存

Picasso图片加载内存优化方案：

<https://www.jianshu.com/p/6b746c904a49>

总结：1、图片裁剪；2、放弃大图memory Cache；3、DetachView时，去除imageView中图片的引用；4、在新进程查看图片；5、页面滑动停止加载图片；6、对于不透明的图片可以使用RGB\_565来优化内存；7、reference

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Java部分

1、各种排序算法

算法的复杂度 计算

一些案例分析

2、多线程 面试回答整理

同步、异步 (同步方法 同步代码块 同步锁)

阻塞、非阻塞

实现一个阻塞模型

跨进程通信—— contentProvider

3、GC垃圾回收机制 内存回收几种算法 内存分代 算法复习

Devid ART 区别

java虚拟机原理

Dalvik虚拟机

4、Java容器

java中的容器主要有List，Set，Map三种。了解这些容器的基本属性以及底层的实现，对正确高效的使用他们很有效。

List下有ArrayList与LinkedeList。

ArrayList 底层类似数组。

LinkedList 底层类似与链表。

Set下有HashSet，TreeSet，LinkedeHashSet。

HashSet重复的值存储一次，查询快。

TreeSet重复的值存储一次，存储为升序排列。

LinkedHashSet重复的值存储一次，存储为放入顺序，查询快。

Map下有HashMap,TreeMap,LinkedHashMap.

HashMap重复键，存储后一个值，查询快。

TreeMap重复键，存储后一个值，存储为升序排列。

LinkedHashMap重复键，存储后一个值，存储为放入顺序，查询快

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### Android部分

1、APK架构、AAPT 打包过程、包内文件目录结构

2、热修复 hotfix 原理 插桩法 几种热修复方案

andfix

美团实现Robust

3、Gradle／Android工程的编译/实现拆包／分库／

dex拆分 细节 MultiDex 优化

multidex 异步加载

multiDex突破65535

4、泰坦 webview 了解逻辑，面试可问的问题

https://www.jianshu.com/p/3c94ae673e2a

5、android View ViewGroup 消息分发

viewGroup的考察(touch事件分发机制)

View的描绘原理

公司wiki

onMeasure()和onLayout()方法

6、android 启动 app 新进程 过程

7、android动态化配置框架

8、android组件化实现方案

9、android项目解耦合具体方案

－组件化

－1.app模块插件化，每一个功能模块都类似于一个项目可以独立开发、运行、测试。这些模块UI跳转之间的耦合可以使用URL来进行跳转。

－2.项目本身使用更加合理的框架，例如MVP，不能生搬硬套，更具项目情况合理使用。

- 3. 使用RxAndroid，这样观察者模式

https://www.jianshu.com/p/45cb536be2f4

10、视频播放器 研究，实现

11、券商相关：分时K线图实现

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

———————其他 工具

1、python 复习

2、正则表达式

3、Lambda表达式

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

————————DP业务相关：

crash上报、符号化，解析 Python shell脚本 工具化

zxing二维码

push 保活，定位，手机Donzon状态下如何保活

混淆

多渠道打包管理

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

高性能的注意项

0、注意内存泄漏（比如数据库的Corsor对象关闭，content内存泄漏，使用工具Leaks检测）

Thread会造成无法被GC吗？必须手动？

Handler的惯用写法，内存泄漏

Activity中使用非静态内部类导致的，如果其实例的持有对象的生命周期大于其外部类对象，那么就有可能导致内存泄露。

1、android 内存优化 onTrimMemory，主要是提示开发者在内存不足的时候，主动处理资源，释放内存，从而避免Android系统杀死，毕竟热启比冷启耀快很多

2、避免使用Enum，使用SparseArray替代HashMap

3、避免创建不必要的对象，对生命周期较短的临时变量，想办法规避掉每次都要创建，减少GC的次数，也是减少内存抖动，减少了Android系统的卡顿

这样GC回收被强制调用机会就会更少，留给Android系统进行UI渲染或者音频加载的时间就会更多，从而避免了卡顿现象

4、慎用Service，下载使用IntentService

5、将应用分为多个进程，减少被系统杀死的概率（注意后台进程不要与前台UI耦合）

6、尽量提供与设备分辨率匹配的bitmap进入内存，否则，分辨率高的，会被压缩，当然，本身分辨率高的本身占用空间就大

7、lint检查，去除不必要的资源，减小APK大小，使用Proguard混淆（好处？？补充下 ）

8、软引用：内存足＝强，不足，回收

弱引用：GC时即回收，生命周期更短

9、尽量避免使用Enum。枚举相对于静态常量来说，需要两倍甚至更多的内存。

10、布局层级 过度绘制

11、启动 使用异步加载

12、谨慎使用面向对象，避免不必要的额外代码，耗费了资源

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 分割线 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lint 接入、FindBugs、CheckStyle，

打点架构实现

整体架构

动态代理 Proxy InvocationHandler

静态模型，模块化实现， 静态化模型 babel实现方式

material design

管道流

BaseDetectHelper.java

splash页面实现 NovaMainApplication 阅读 皮肤实现

app包大小控制方案 包体大小监控，缩小

查看应用内存 adb shell dmpsms package name