APP研发录

Android 经典场景设计

ImageLoader实现异步网络加载图片，缓存以及显示，支持多线程异步加载

三级缓存：内存-》本地-》网络

我们可以每次推出一个页面，把ImageLoader内存中的缓存全部清除，这样就节省了大量内存，下次再用到时候再从本地取出来就是了

ImageLoaderConfiguration----对图片缓存进行总体配置，包括内存大小，本地缓存大小和位置，日志，下载策略（FIFO还是LIFO）

ImageLoader---我们一般使用displayImage来吧URL对应的图片显示在ImageVIew上。

DISplayImageOtions---在每个页面需要显示图片地方，控制如何显示的细节，比如制定下载时的默认图（包括下载中，下载失败，URL为空等）是否将缓存放到内存或者本地磁盘

ImageLoaderConfiguration就像厨房的规定，每一个厨师要怎么做着装，保持厨房的干净。ImageLoader就像是具体做菜的厨师，负责菜谱的制作，DIspalyIamgeOptions就像每个客户的偏好。 （可以使用buidler模式进行初始化）

ImageLoader优化：尽管ImageLoader很强大，但一直把图片缓存在内存中，会导致内存过高。虽然对图片的引用是软引用，软引用在内存不够的时候进行GC，但我们还是希望减少GC的次数，所以要经常手动清理ImageLoader中的缓存。

我们可以在AppBaseActivity中的onDestore方法中，以确保页面销毁的时候，把为了显示这个页面而增加的内存缓存清除。即使下次再复用，我们可以使用本地缓存策略

图片加载利器：Fresco

FacceBook开源了他的Android图片加载组件Fresco，开始优化App暂用内存

在Appication级别中，对Fresco进行初始化。

Fresco.initialize(getApplicationContext())

在IamgeLoader等传统第三方图片处理SDK不同，Fresco是基于控件级别，所以我们把程序中显示网络图片的IamgeVidw都替换为SimpleDraweeView。

<LineaerLayout

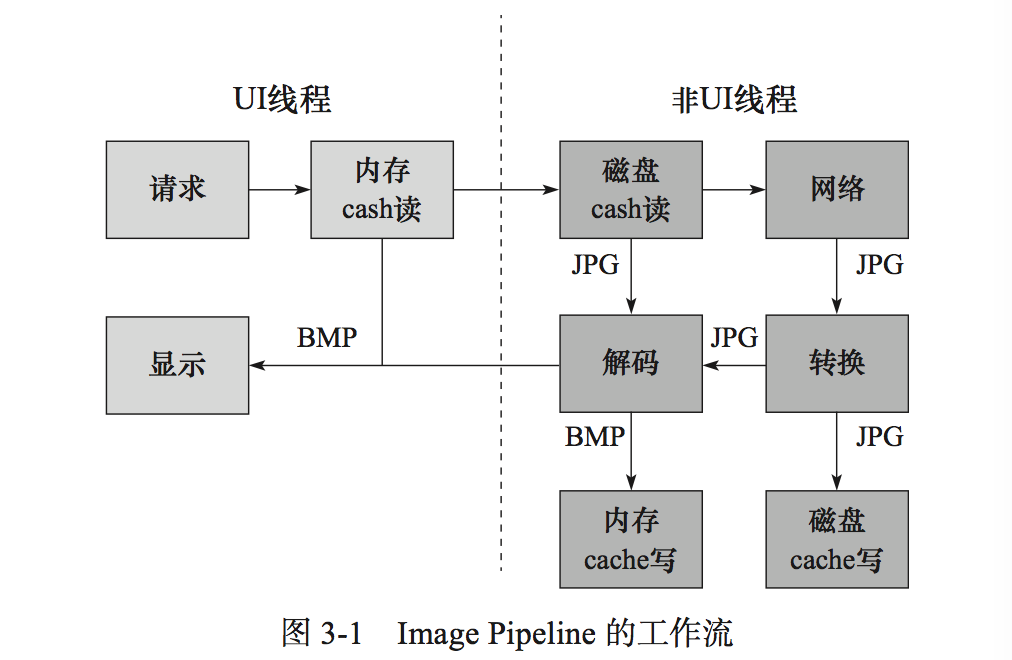
xmlns:android=<http://schemas.android.com/apk/res/android>

xmlns:fresco=http://schemas.android.com/apk

<com.facebook.drawee.view.simpleDraweeVIew

fresco:placeHolderImage=”drawable/placeholder”>

Fresco的原理是，设计了一个Image PipeLine的概念，他负责先检查内存、磁盘文件（Disk），乳沟都没有在老老实实从网络下载图片，如图所示：箭头上标记了jpg或者bmp格式的，表示cache中有图片，直接取出，没有标记，则表示cache中找不到



Fresco 有三个线程池，其中3个线程用于网络下载图片，2个线程用于磁盘文件读写，还有两个线程用于CPU相关操作，比如图片解码，转码，以及放在后台一些费时操作

Fresco三层缓存

第一层·Bitmap缓存

在android5.0系统中，考虑到内存管理有了很大进步，所以Bitmap缓存位于Java堆中

而在于android4.x和更低系统中，Bitmap缓存位于ashmem中，而不是位于java堆中，这意味着图片的创建和回收不回应引发过多的GC，从而让app运行更加快，当APP切换到后台时，Bitamp缓存会被清空

第二层 内存缓存

内存缓存中存储图片的原始压缩格式，从内存中取出的图片，在显示前必须先解码，当app切换到后台时，内存缓存也被清空

第三层 磁盘缓存

磁盘缓存（本地）磁盘缓存中存储的也是图片的原始压缩格式，在使用前也需要解码，党app切换到后台时，磁盘缓存不会丢失，即使关机也不会。

有时候我们使用APP原先占用180 现在只会占用80

对网络流量进行优化

对app的最低容忍限度，在2 3 4网络情况下，每个页面都能打开，都能正常跳转到指定的也买呢。要能够完成一次完整的支付流程

慢点没关系，尤其是2g网络，但是动不动就弹出 无法连接网络 或者网络连接失败 的对话框 就是我们开发人员必须解决的问题了

通信层优化：

1. MobileAPI接口放回的数据，要使用Gzip进行压缩 注意要大于1kb才进行压缩，否者得不偿失。
2. APP与MobileAPI之间的数据传输，通常是遵守JSON协议，JOSN因为是xml格式的，并且是以字符串，在数据量上还有可以压缩的控件，这里推荐一种新的数据数据传输协议，那就是ProtoBuffer。这种协议时二进制的，所以在表示大数据时，空间比json小很多
3. 接下来要解决频繁调用MobieAPI问题，我们知道，发起一次网络请求，服务器处理速度还是很快的，主要花费在数据传输上，也就是一来一回走路的时间，能一次取到的数据一次性获取
4. 我们知道，传统的MobileAPi使用的是Http无状态短链接。使用Http协议的速度远不如TCP协议，因为后者是长链接。所以我们使用TCP长链接，以提高访问速度，缺点是一台服务器能支持的长链接个数不多，也不再接收数据
5. 要建立取消网络请求机制，一个页面如果没有请求完网络数据，在跳转到另一个界面之前，要把之前的网络请求都取消，不在的等待，也不再接收数据
6. 增加重试机制。如果MoblieApi严格饿Resful风格，那么我们一般将获取数据的请求接口都定义为get，而把操作数据的请求接口都定义为post，那么我们就能为get请求配置重试机制，比如get请求失败三次后重试，

有人会问post请求失败后，是否需要重试机制呢。比如下单接口是一个post请求，如果请求失败了那么就会重试3次，直到下单成功，但是有时后post请求并没有失败，而是超时了，超时时间是30秒，但是却31秒返回了，如果因此重发新的下单请求，那么就会连续下单两次。所以post请求不建议有重试机制

此外，对所有的post请求，都要增加防止用户1分钟内频繁发起相同的请求机制，这样有效防止重复下单，重复发评论。

如果post请求具有防重机制，那倒是可以增加重试机制。但是要可以在服务器端灵活配置重试次数

图片策略优化：

1要确保下载的每张图片，都符合ImageView的控件大小

序列化的缺点：

再次强调：把全局变量序列化到本地的方案，只是一种过渡性解决方案，他有几个硬伤：

1. 每次设置全局变量的值都要强制执行一次序列化的操作，容易找成arn。

解决方案：每次set不做序列化，都设置完后，一次性序列化到本地。

1. 序列化生成文件，会因为内存不够而丢失

因为我们将序列化到本地文件放在了内存/data/data/com.youngheart/cache/这个目录下。内存空间十分有限，其实sharedPrefer e