目录

1. 图片库对比

https://www.jianshu.com/p/fc72001dc18d

Glide 和 Picasso 主要体现在下面几个方面：

(1)**库的大小和方法的数量：**Glide 要比 Picasso 大的 3.5 倍左右：（但是，Glide 这里有一个非常招人喜欢的地方，就是 Glide 在设计的时候，就有 Activity 和 Fragment 的生命周期。什么意思呢？ 就是说你可以传递 Activity 或者 Fragment 的 context 给 Glide.with()， 然后 Glide 就会非常智能的同 Activity 的生命周期集成， 比如 OnResume 或者 onPause()）

(2)**缓存大小：**Picasso 是缓存的 full size 图片的大小，而 Glide 是加载已经改变大小后的图片，显而易见使用的 memory 会小很多。 这样可以减少 OutOfMemoryError 的可能性。

Picasso 是下载图片然后缓存完整的大小到本地，比如说图片的大小是 1080p 的，之后如果我需要同一张图片，就会返回这张 full size 的，如果我需要 resize（调整大小），也是对这种 full size 的做 resize（调整大小）。

Glide 则是完全不一样的做法。Glide 是会先下载图片，然后改变图片的大小，以适应 imageView 的要求，然后缓存到本地。 所以如果你是下载同一张图片，但是设定两个不一样大小的 imageView，那么 Glide 实际上是会缓存两份。

换个角度来看，这里不仅仅是缓存的问题，比如一个 ImageView 要改变它的大小，Picasso 就只需要下载一次 full size 的图片，但是 Glide 实际上就不仅仅是下载一次了，它需要去单独下载然后改变大小适配 imageView，因为对于 Glide 来讲，需要缓存不同大小的同一张图片。

(3)加载图片的时间

Picasso 会比 Glide 快一点。缓存机制导致，因为Picasso 是直接把图加载到内存中，而 Glide 则需要改变图片大小再加载到内存中去。这个应该是会耗费一定的时间。

但是，当加载图片从内存中的时候，Glide 则比 Picasso 要快。其原理还是因为缓存机制的区别。因为Picasso 从缓存中拿到的图片，还要先去 resize 后，然后设定给 imageView，但是 Glide 则不需要这样。

(4)其他功能的对比

GIF 支持：Glide 支持 GIF。 对于加载 GIF 来说，Glide 只需要简单使用 Glide.with(...).load(...)。但是 Picasso 是不支持的

灵活性：Glide 提供了非常多的配置，你可以非常灵活的根据你的需求来客制化，从而缩减 Glide 库的大小等。

总结：如果希望 app 大小小一点，没那么多额外功能，那就使用 Picasso.

如果你的应用粗腰加载 GIF，或者对于内存的大小比较在意，选择 Glide。

2. 图片库的源码分析（Glide）

https://blog.csdn.net/guolin_blog/article/details/53759439

到时候看框架

3. 图片框架缓存实现

4. Glide 使用什么缓存？

https://blog.csdn.net/guolin_blog/article/details/53759439

生成进行缓存的 Key：决定缓存 Key 的参数有 10 个之多。

一张图片的 url 地址连同着 signature、width、height 等等 10 个参数一起传入到 EngineKeyFactory 的 buildKey()方法当中，从而构建出了一个 EngineKey 对象，这个 EngineKey 也就是 Glide 中的

缓存 Key 了。（决定缓存 Key 的条件非常多，即使你用 override()方法改变了一下图片的 width 或者 height，也会生成一个完全不同的缓存 Key）

EngineKey 类中主要就是重写了 equals()和 hashCode()方法，保证只有传入 EngineKey 的所有参数都相同的情况下才认为是同一个 EngineKey 对象，我就不在这里将源码贴出来了。

内存缓存

Glide 内存缓存的实现自然也是使用的 LruCache 算法。不过除了 LruCache 算法之外，Glide 还结合了一种弱引用的机制，共同完成了内存缓存功能。

Glide 的图片加载过程中会调用两个方法来获取内存缓存，loadFromCache()和 loadFromActiveResources()。这两个方法中一个使用的就是 LruCache 算法，另一个使用的就是弱引用。

当 acquired 变量大于 0 的时候，说明图片正在使用中，也就应该放到 activeResources 弱引用缓存当中。而经过 release()之后，

如果 acquired 变量等于 0 了，说明图片已经不再被使用了，那么此时会调用 listener 的 onResourceReleased()方法来释放资源

当图片不再使用时，首先会将缓存图片从 activeResources 中移除，然后再将它 put 到 LruResourceCache 当中。这样也就实现了正在使用中的图片使用弱引用来进行缓存，不在使用中的图片使用 LruCache 来进行缓存的功能。

硬盘缓存

Glide 是使用的自己编写的 DiskLruCache 工具类，但是基本的实现原理都是差不多的。

一种是调用 decodeFromCache()方法从硬盘缓存当中读取图片，一种是调用 decodeFromSource()来读取原始图片。默认情况下 Glide 会优先从缓存当中读取，只有缓存中不存在要读取的图片时，才会去读取原始图片。

调用 transform()方法来对图片进行转换，然后在 writeTransformedToCache()方法中将转换过后的图片写入到硬盘缓存中，调用的同样是 DiskLruCache 实例的 put()方法，不过这里用的缓存 Key 是 resultKey（也就是调整图片大小后进行缓存）。

5. LRUCache 原理

https://www.cnblogs.com/tianzhijiexian/p/4248677.html

LruCache 底层是 LinkedHashMap

任意时刻，当一个值被访问时，它就会被移动到队列的开始位置，所以这也是为什么要用 LinkedHashMap（数据结构+算法） 的原因，因为要频繁的做移动操作，为了提高性能，所以要用 LinkedHashMap。这个大小是可以设定的（自定义 sizeOf()）。当 cache 满了时，此时再向 cache 里面添加一个值，那么，在队列最后的值就会从队列里面移除，这个值就有可能被 GC 回收掉。（放入这个 map 的 item 应该会被强引用，要回收这个对象的时候是让这个 key 为空，这样就让有向图找不到对应的 value，最终被 GC。）

我们应该能通过某个方法来清空缓存，这个缓存在 app 被退出后就自动清理，不会常驻内存。

6. 图片加载原理

https://blog.csdn.net/guolin_blog/article/details/53759439

参考 Glide 原理

7. 自己去实现图片库，怎么做？

（随便套个开源框架的原理）

套 Glide 的就 OK 拉，从设计思想，然后到实现方式

应该具备以下功能：

1) 图片的同步加载

2) 图片的异步加载

3) 图片压缩

4) 内存缓存

5) 磁盘缓存

6) 网络拉取

8. Glide 源码解析

https://blog.csdn.net/guolin_blog/article/details/53759439

等待框架图源码分析成文字

https://blog.csdn.net/guolin_blog/article/details/53759439

同上 3

9. Glide 内存缓存如何控制大小？

https://blog.csdn.net/guolin_blog/article/details/53759439

问的应该是如何更适应的更优秀的控制大小？

Glide 使用了两种缓存类型来控制大小：

一种是 Resource 缓存，一类是 Bitmap 缓存。

Resource 缓存：图片从网络加载，将图片缓存到本地，当需要再次使用时，直接从缓存中取出而无需再次请求网络。

Glide 在缓存 Resource 使用三层缓存，包括：

1. 一级缓存:缓存被回收的资源,使用 LRU 算法(Least Frequently Used，最近最少使用算法）。当需要再次使用到被回收的资源，直接从内存返回。
2. 二级缓存：使用弱引用缓存正在使用的资源。当系统执行 gc 操作时，会回收没有强引用的资源。使用弱引用缓存资源，既可以缓存正在使用的强引用资源,也不阻碍系统需要回收无引用资源。
3. 三级缓存：磁盘缓存。网络图片下载成功后将以文件的形式缓存到磁盘中。

Bitmap 缓存

通过 Bitmap 压缩质量参数：Glide 默认使用 RGB_565，比系统默认使用的 ARGB_8888 节省一半的资源，但 RGB_565 无法显示透明度。

Bitmap 缓存算法：

在 Glide 中，使用 BitmapPool 来缓存 Bitmap,使用的也是 LRU 算法。当需要使用 Bitmap 时，从 Bitmap 的池子中取出合适的 Bitmap,若取不到合适的，则再新创建。当 Bitmap 使用完后，不直接调用 Bitmap.recycler()回收，而是放入 Bitmap 的池子。