**Android 图片加载框架**

参考：<https://blog.csdn.net/guolin_blog/article/details/53759439>

<https://blog.csdn.net/guolin_blog/article/details/9316683>

<https://www.jianshu.com/p/17644406396b>

1. **Glide**
2. Glide的使用

Glide.with(this)

.load(url)

.asBitmap()

.placeholder(R.drawable.loading)

.error(R.drawable.error)

.diskCacheStrategy(DiskCacheStrategy.NONE)

.override(100, 100)

.into(imageView);

1. Glide#with()
2. with()中传入的为Application时，通过应用程序的Lifecycle自动和应用程序的生命周期是同步的。若应用程序关闭，则Glide的加载也会同时终止；
3. 在非主线程当中使用的Glide，那么不管你是传入的Activity还是Fragment，都会被强制当成Application来处理。
4. 在主线程当中使用的Glide，向当前的Activity当中添加一个隐藏的Fragment，从而通过该Fragment获取到加载的生命周期（Glide无法知道Activity的生命周期，而Fragment的生命周期和Activity是同步的，若Activity被销毁了，Fragment是可以监听到，Glide可以通过隐藏的Fragment捕获生命周期并停止图片加载了）
5. load()返回DrawableTypeRequest对象（其内部提供了两个方法asBitmap()和asGif()分别用于强制指定加载静态图片和动态图片，这两方法分别创建了一个BitmapTypeRequest和GifTypeRequest。若没有进行强制指定，则默认使用DrawableTypeRequest）
6. into()
7. 将into()中的参数包装成一个Target；
8. 调用request请求加载网络图片（加载之前有异常则加载出错图片，否则加载兜底图）
9. 如果内存缓存中有要加载的图片，则从内存缓存中取，否则开启子线程进入异步流程，子线程中先从磁盘缓存中取，磁盘缓存中没有时再从网络上下载。
10. 请求到图片之后对其解码，并对图片做压缩、旋转、圆角等处理
11. 通过handler机制将图片发送到主线程，并设置到view中显示
12. 缓存：key由id、signature、width、height等10个参数组成的
13. 内存缓存：防止应用重复将图片数据读取到内存当中
14. Glide默认开启内存缓存，通过skipMemoryCache(true)可以关闭内存缓存；
15. 使用LruCache算法+弱引用缓存，先从内存缓存（LruCache）中取，再从弱引用缓存取；
16. 从内存缓存（LruCache）获取到图片后，将其从LruCache中删除，并放入到弱引用缓存；（弱引用缓存正在使用的图片，可以保护这些图片不会被LruCache算法回收掉。）
17. 子线程下载图片后，通过Handler机制发送到主线程，Engine先将其放入弱引用缓存。每次使用该图片都会调用EngineResource#acquire()使引用计数acquired加1，图片使用完之后调用EngineResource#release()使引用计数acquired减1。当acquired为0时，说明图片不在使用，将其从弱引用缓存中移除并放入到内存缓存（LruCache）中。
18. 磁盘缓存：防止应用重复从网络或其他地方重复下载和读取数据。
19. Glide通过diskCacheStrategy ()方法设置磁盘缓存策略；

DiskCacheStrategy.NONE： 不缓存任何内容。

DiskCacheStrategy.SOURCE： 只缓存原始图片。

DiskCacheStrategy.RESULT： 只缓存转换过后的图片（默认选项）。

DiskCacheStrategy.ALL：既缓存原始图片，也缓存转换过后的图片。

1. 默认通过decodeFromCache()从硬盘缓存当中读取图片（decodeFromCache()中通过decodeResultFromCache从缓存中获取转换后的图片，没有获取到再通过decodeSourceFromCache()从缓存中获取原始图片），缓存中不存在时才通过decodeFromSource()从网络上获取原始图片。
2. Glide提供了LruBitmapPool，统一管理Bitmap的创建和释放，避免Bitmap的频繁创建，提高复用，减少了内存抖动。
3. 通过BitmapPool防止OOM。新加载图片时，会先从BitmapPool里面找有没有相应大小的Bitmap，有则直接使用，没有才会申请新的Bitmap；回收时，则会提交给BitmapPool, 供下次使用。

<https://blog.csdn.net/changwilling/article/details/53836702?utm_source=blogxgwz0>

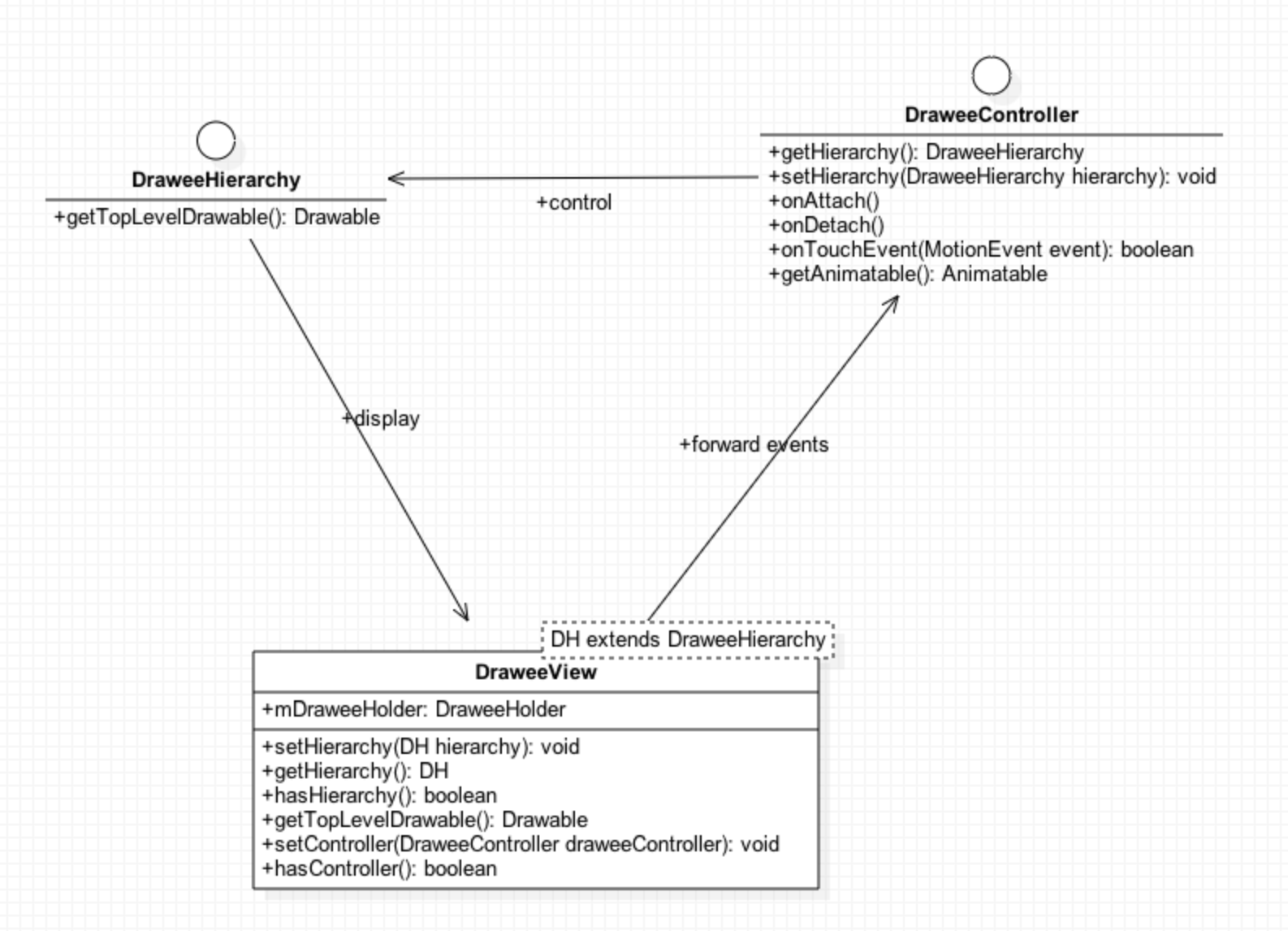
1. preload()替换into()实现预加载，具体实现：获取PreloadTarget传入到into()方法中，下载并缓存好图片后，调用PreloadTarget#onResourceReady()时不做处理
2. downloadOnly()只下载图片，不会对图片进行预加载。downloadOnly()返回FutureTarget对象，FutureTarget#get()可以获取到下载的图片，但FutureTarget#get()会阻塞线程，所以需要在子线程中执行。
3. listener()监听，GenericRequest#onResourceReady()会先判断listener()传入的listener的onResourceReady()的返回值，返回false时才会继续执行target的onResourceReady()；GenericRequest#onException()会先判断listener()传入的listener的onException ()的返回值，返回false时才会继续执行setErrorPlaceholder()；
4. 图片变换
5. centerCrop()
6. fitCenter()
7. transform()，centerCrop()和fitCenter()都是对transform()的封装；
8. dontTransform()表示不对图片进行变换
9. **Fresco**

<https://www.jianshu.com/p/cbfbc7d8b634>

<https://blog.csdn.net/a910626/article/details/52387261>

<https://www.jianshu.com/p/47322042c052>

1. 结构



1. DraweeView
2. SimpleDraweeView：为DraweeHolder设置DraweeController
3. GenericDraweeView：解析xml属性，并为DraweeHolder设置DraweeHierarchy
4. DraweeHolder

通过DraweeHolder使DraweeView与DraweeHierarchy、DraweeController解耦；

1. DraweeHierarchy
2. DraweeController
3. onAttach()通过DataSource的观察者模式获取图片；
4. 通过ImagePipeline获取图片，由Producer的责任链模式，对未编码的图片数据进行处理；
5. 缓存

<https://www.cnblogs.com/wgwyanfs/p/7239166.html>

1. BitmapMemoryCache：已解码的内存缓存
2. EncodedMemoryCache：未解码的内存缓存
3. DiskStorage：文件缓存
4. 使用了lru算法，超过了回收队列的最大值或者是回收队列的对象的最多个数，就从回收队列移除第一个对象；
5. 早期版本会通过native的内存空间（ashmem匿名共享内存）缓存图片；
6. 图片获取过程：先从BitmapMemoryCache中读取，BitmapMemoryCache中不存在时再从EncodedMemoryCache中读取，EncodedMemoryCache中也不存在时才从DiskStorage中获取；
7. fresco使用大量对象池，避免频繁的申请内存、回收内存造成内存抖动。
8. **Glide和Fresco的区别**
9. Glide可以通过diskCacheStrategy()设置缓存策略（缓存原图或结果图），Fresco只能缓存原图
10. Glide为了节省内存采用的图片格式为RGB565，比Fresco的ARGB8888节省近一半的内存；
11. Fresco可以显示图片加载进度，但Glide不能显示加载进度；
12. Fresco可以先加载小尺寸图片后加载大尺寸图片，Glide只有占位图；