Bitmap

1、Bitmap 使用需要注意哪些问题 ？

- 参考回答：
    - **要选择合适的图片规格（bitmap 类型）**：通常我们优化 Bitmap 时，当需要做性能优化或者防止 OOM，我们通常会使用 RGB_565，因为 ALPHA_8 只有透明度，显示一般图片没有意义，Bitmap.Config.ARGB_4444 显示图片不清楚，Bitmap.Config.ARGB_8888 占用内存最多。：
        - 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（默认）
        - RGB_565 每个像素占用 2byte 内存
    - **降低采样率**：BitmapFactory.Options 参数 inSampleSize 的使用，先把 options.inJustDecodeBounds 设为 true，只是去读取图片的大小，在拿到图片的大小之后和要显示的大小做比较通过 calculateInSampleSize()函数计算 inSampleSize 的具体值，得到值之后。options.inJustDecodeBounds 设为 false 读图片资源。

- **复用内存**：即通过软引用(内存不够的时候才会回收掉)，复用内存块，不需要再重新给这个 bitmap 申请一块新的内存，避免了一次内存的分配和回收，从而改善了运行效率。
- **使用 recycle()方法及时回收内存。**
- **压缩图片。**

2、Bitmap.recycle()会立即回收么？什么时候会回收？如果没有地方使用这个 Bitmap，为什么垃圾回收不会直接回收？

- 参考回答：
  - 通过源码可以了解到，加载 Bitmap 到内存里以后，是包含**两部分内存区域**的。简单的说，一部分是 **Java 部分**的，一部分是 **C 部分**的。这个 Bitmap 对象是由 Java 部分分配的，不用的时候系统就会自动回收了
  - 但是那个对应的 **C 可用**的内存区域，虚拟机是不能直接回收的，这个只能调用底层的功能释放。所以需要调用 recycle()方法来释放 C 部分的内存
  - bitmap.recycle()方法用于回收该 Bitmap 所占用的内存，接着将 bitmap 置空，最后使用 System.gc()调用一下系统的垃圾回收器进行回收，调用 System.gc()并不能保证立即开始进行回收过程，而只是为了加快回收的到来。

3、一张 Bitmap 所占内存以及内存占用的计算

- 参考回答：
    - Bitamp 所占内存大小 = 宽度像素 x （inTargetDensity / inDensity） x 高度像素 x （inTargetDensity / inDensity） x 一个像素所占的内存字节大小
        - 注：这里 inDensity 表示目标图片的 dpi（放在哪个资源文件夹下），inTargetDensity 表示目标屏幕的 dpi，所以你可以发现 inDensity 和 inTargetDensity 会对 Bitmap 的宽高进行拉伸，进而改变 Bitmap 占用内存的大小。
    - 在 Bitmap 里有两个获取内存占用大小的方法。
        - **getByteCount()**：API12 加入，代表存储 Bitmap 的像素需要的最少内存。
        - **getAllocationByteCount()**：API19 加入，代表在内存中为 Bitmap 分配的内存大小，代替了 getByteCount() 方法。
        - 在**不复用 Bitmap** 时，getByteCount() 和 getAllocationByteCount 返回的结果是一样的。在通过**复用 Bitmap** 来解码图片时，那么 getByteCount() 表示新解码图片占用内存的大小，getAllocationByteCount() 表示被复用 Bitmap 真实占用的内存大小

4、Android 中缓存更新策略 ？

- 参考回答：
    - Android 的缓存更新策略**没有统一的标准**，一般来说，缓存策略主要包含**缓存的添加、获取和删除**这三类操作，但不管是内存缓存还是存储设备缓存，它们的缓存容量是有限制的，因此删除一些旧缓存并添加新缓存，如何定义缓存的新旧这就是一种策略，**不同的策略就对应着不同的缓存算法**
    - 比如可以简单地根据文件的最后修改时间来定义缓存的新旧，当缓存满时就将最后修改时间较早的缓存移除，这就是一种缓存算法，但不算很完美

5、LRU 的原理 ？

- 参考回答：
    - 为减少流量消耗，可采用缓存策略。常用的缓存算法是 LRU(Least Recently Used)：当缓存满时，会优先淘汰那些近期最少使用的缓存对象。主要是两种方式：
    - **LruCache(内存缓存)**：LruCache 类是一个线程安全的泛型类：内部采用一个 LinkedHashMap 以强引用的方式存储外界的缓存对象，并提供 get 和 put 方法来完成缓存的获取和添加操作，当缓存满时会移除较早使用的缓存对象，再添加新的缓存对象。

- **DiskLruCache(磁盘缓存)**：　通过将缓存对象写入文件系统从而实现缓存效果