简答题

1、Android 系统将进程分为哪些类型？

2、当一个 Android 应用崩溃时，有可能导致整个系统的崩溃。你认为可能吗？

3、引发 GC 操作的原因是什么？

4、为什么要避免频繁的 GC 操作？

5、如何避免 GC 频繁发生？

6、常见的内存优问题都有哪些？

7、你在开发中都使用了哪些内存优化的工具？

8、内存泄露是什么，如何解决？

9、内存抖动是什么，如何解决？

10、内存溢出是什么，如何解决？

11、你在开发中都使用过哪些内存优化措施？

简答题答案

1、Android 系统将进程分为哪些类型？

答：

1) 前台进程：系统进程或正在前台运行的进程。
2) 可见进程：可见但不能与用户交互的进程。
3) 桌面进程-launcher：保证多任务切换。
4) 次要服务进程：正在运行的服务。
5) 后来进程，包括处于停滞状态的 Activity 的进程。
6) 内容供应节点进程：Content Provider 没有程序实体的进程。
7) 空进程：被 Back 键关闭的进程。

2、当一个 Android 应用崩溃时，有可能导致整个系统的崩溃。你认为可能吗？

答：

不可能，Android 系统为每个应用分配一个进程。应用崩溃只能导致当前进程崩溃。

3、引发 GC 操作的原因是什么？

答：

1) 调用 System.gc 方法- GC_EXPLICIT；
2) 短时间内有大量频繁的对象创建与释放操作；
3) 内存占用介于阈值边缘，当有新对象创建时会导致超越阈值触发 GC 操作- GC_CONCURRENT。
4) 内存分配失败时触发- GC_MALLOC；

5) 外部内存分配失败时触发- GC_EXTERNAL_ALLOC。

4、为什么要避免频繁的 GC 操作？

GC 操作会导致系统所有的线程暂停。频繁的 GC 操作会导致 UI 线程卡顿，影响用户体验。

5、如何避免 GC 频繁发生？

答：

1)尽量避免在频繁触发的逻辑方法中大量地分配对象，如：

a)尽量避免在 for 循环中频繁分配对象，如一定需要则用对象池复用创建过的对象；

b)避免在自定义 View 的 onDraw()方法中执行复杂的操作及创建对象（譬如 Paint 的实例化操作不要写在 onDraw()方法中等）；

2)对于并发下载等类似逻辑的实现尽量避免多次创建线程对象，而是交给线程池处理。

6、常见的内存优问题都有哪些？

1)内存溢出 OOM；

2)内存抖动: 短时间内有大量频繁的对象创建与释放操作，也就是俗称的内存抖动现象；

3)内存泄露：

7、你在开发中都使用了哪些内存优化的工具？

1)Allocation Tracker 工具-分配跟踪器，用于查看内存分配情况。

a)可以查看程序运行时某段时间的内存分配情况，如内存抖动。

b)可以查看程序运行结束后的内存分配情况，如内存泄露。

c)可以检查到指定类的内存分配情况。

2)DDMS-Heap 工具-检查指定应用的堆分配情况

a) 可以显示出当前引用占用的内存，剩余的内存等信息。

8、内存泄露是什么，如何解决？

答：

1)内存泄露就是一个对象被超过自己生命周期以外的对象强引用导致该对象无法被正常垃圾回收。例如：

a)外部类的静态变量持有了 Activity 对象。导致 Activity 退出后无法被系统回收。

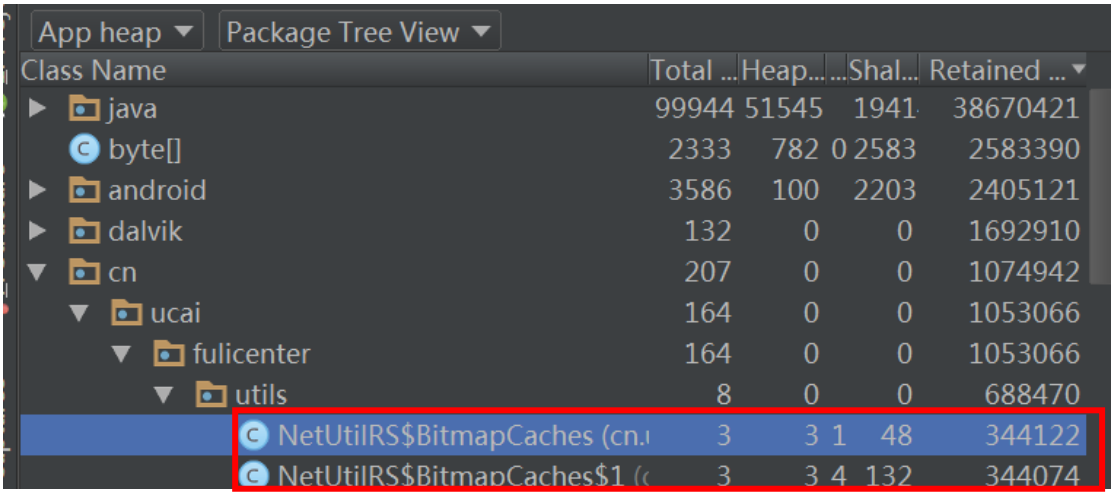
b)第三方框架持有了 Application 或 Activity，导致应用或 Activity 在退出后无法被回收。例如环信 SDK 中的监听聊天消息的监听器持有了 Application 对象。

c)Activity 的匿名内部类运行没有结束，由于匿名内部类持有 Activity，导致 Activity 在退出后无法被回收。

案例：Day12_08 模块，MainActivity 内部的匿名内部类中运行了一个无限循环。导致 MainActivity 在退出后无法被系统回收。

d)加载大量的图片缓存在内存中，没有在当前页面退出后释放。

福利社项目中 Volley 框架加载图片后，应用退出后，仍驻留在内存中的图片缓存。



2)常见的内存泄露导致问题如下：

a)应用卡顿，响应速度慢（内存占用高时 JVM 虚拟机会频繁触发 GC）；

b)应用被从后台进程干为空进程；

c)应用莫名的崩溃（内存超过了阈值 OOM）；

3)如何解决内存泄露？

a)查找所有持有 Application 或 Activity 的静态变量的类，在这些类中编写释放持有 Activity、Application 的静态变量的方法，在 Activity.onDestroy()中调用该工具类的方法释放静态变量持有的对象。

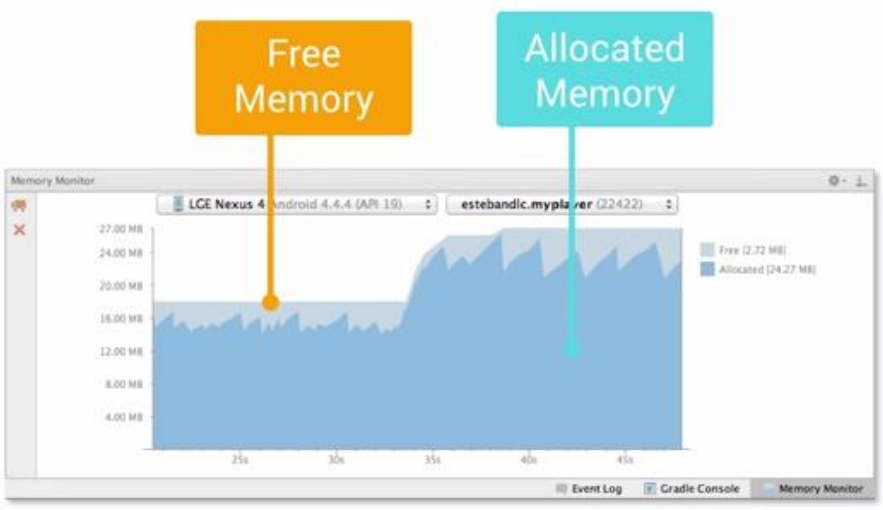
9、内存抖动是什么，如何解决？

答：

1、**内存抖动(Memory Churn)**是因为在短时间内大量的对象被创建又马上被释放。瞬间产生大量的对象会严重占用 Young Generation 的内存区域，当达到阀值，剩余空间不够的时候，会触发 GC 从而导致刚产生的对象又很快被回收。即使每次分配的对象占用了很少的内存， 但是他们叠加在一起会增加 Heap 的压力，从而触发更多其它类型的 GC。这个操作有可能会影响到帧率，并使得用户感知到性能问题。

2、监测内存抖动的工具

1)Android Studio 中的 Memory Monitor 用于帮助查看程序的内存使用情况。如下图所示：



2)DDMS-Heap 工具。

3、如何解决内存抖动？

1)避免在循环中大量创建对象，若使用大量对象，应使用对象池。如在循环中使用 Message 时，用 Message.obtain 方法从消息池中复用 Message。

2)尽量避免在 onDraw 方法中创建对象。

3)在循环中连接字符串时，要使用 StringBuilder 或 StingBuffer，避免使用 String。

10、内存溢出是什么，如何解决？

1、Android 的应用程序所能申请的最大内存都是有限的，内存溢出(OutOfMemoery)简称 OOM，是指 APP 向系统申请内存的请求超过了系统为应用分配的上限，系统无法再分配多余的空间，就会造成 OOM。

1)内存泄漏(Memory Leak)，累积到一定程度导致 OOM。

2)一次性申请很多内存，如加载大图片。

2、OOM 解决：

1)大图片加载时：

a)通过 BitmapFactory.options 压缩图片尺寸。

b)通过 BitmapFactory. inPreferredConfig 控制图片的质量，例如将图片每个像素默认的字节数由 Bitmap.Config ARGB_8888 改为 Bitmap.Config.RGB_565。

c)bitmap.compress 方法中第二个参数保存图片质量 quailty 由 100 改为 80 等。

4)为防止 OOM 造成的异常，用 try-catch(OutOfMeory e)捕获 OOM 异常。

11、你在开发中都使用过哪些内存优化措施？

答：

1、大图片加载的三种优化措施：

a)通过 BitmapFactory.options 压缩图片尺寸。

b)通过 BitmapFactory. inPreferredConfig 控制图片的质量，例如将图片每个像素默认的字节数由 Bitmap.Config ARGB_8888 改为 Bitmap.Config.RGB_565。

c)compress 方法中第二个参数保存图片质量 quailty 由 100 改为 80 等。

2、在多个 Activity 或 Fragment 都可能加载大量图片时，从一个 Activity 跳转到另一个 Activity 时，在当前 Activity.onStop 方法中调用 Bitmap.recycle 方法释放当前 Activity 中的所有 Bitmap。

如果不采取以上措施，处于 stop 状态的 Activity 中仍缓存着大量图片。

在所有可能退出当前应用的 Activity 的 onDestroy 方法中调用 ImageLoader.release 方法，将缓存在 LruCache 集合中的图片释放，否则会造成内存泄露。

3、在加载大图片时，为防止 OOM 造成的异常，用 try-catch(OutOfMeoryError e)捕获 OOM 异常。