**内存泄漏**

**一、为什么要解决内存泄漏**

如果一个无用对象（不需要再使用的对象）仍然被其他对象持有引用，造成该对象无法被系统回收，以致该对象在堆中所占用的内存单元无法被释放而造成内存空间浪费，这中情况就是内存泄露；

内存泄漏并不会导致程序功能异常，但是它会导致Android程序的内存占用过大，这将提高内存溢出的发生几率（内存溢出？），造成设备卡顿。

Android为每个应用只分配固定大小的内存，如要更多内存可分多个进程；

1. **Java垃圾回收机制**

**1、java垃圾回收机制（GC）检测策略**

**1）引用计数（缺点）**

首先介绍一种用于说明垃圾收集工作方式的策略，\*\* 引用计数 \*\*：

每个对象都含有一个引用计数器，当有引用连接至对象时，引用计数加1。当引用离开作用域或者被置为null时，引用计数减1。垃圾回收器在遍历所有对象时**发现引用计数为0便释放其内存**。这种策略很难处理循环引用的情况。不过我们无需过多的考虑此策略有何优缺点，这仅仅是用来让你了解一些垃圾回收的工作方式。而且现在JVM大多也不用这种策略来进行垃圾回收。

以上我们简单的了解了一下垃圾回收的大致流程，那么接下来我们来了解一下垃圾回收器如何判断一个对象是否可回收

**2）可达性分析算法（根搜索算法）**

既然引用计数有缺点，那么可以采用其他的策略，Java采用了一种新的算法：可达性分析算法。

对象引用遍历从一组对象开始（GC Roots），沿着整个对象图上的每条链接，递归确定可到达（reachable）对象并生成一棵引用树，树的节点视为可达对象，反之视为不可达。之后垃圾回收器在进行垃圾回收的时候便可以回收那些不可达的对象。

**2、Java的引用类型**

#### 强引用（Strong Reference）

#### 软引用（Soft Reference）

#### 弱引用（Weak Reference）

#### 虚引用（Phantom Reference）

**三、内存检测工具**

**1、MAT**

MAT全称Eclipse Memory Analyzer

内存泄漏实例演示MAT的使用

**2、Leakcanary**

1. **Leakcanary原理**

**Github地址：https://github.com/square/leakcanary**

**https://www.jianshu.com/p/51395d8e512f**

<https://www.jianshu.com/p/0448dab89625>

基本原理太简单了，就是新建一个WeakReference对象指向要关注的Activity，Activity被回收的时候WeakReference对象会加到WeakReferenceQueue 队列中，检测WeakReferenceQueue队列 是否有WeakReference对象就知道有没有泄露了，有的话就没有泄露，没有的话就泄露了

1. **集成soling Leakcanary jar包的问题**

Release版本是否集成，也该是不需要的？千伟确定是否打包到固件

集成文档请参考gitlab仓库的ReadMe文件；

1. **常见内存泄漏的原因及解决方案**

**1、静态变量导致的内存泄漏**

1. **单例模式模式导致的内存泄漏**

单例模式的生命周期对应应用程序的生命周期，所以

1. 我们在构造单例模式时，尽量避免使用Activity的上下文，而使用Application的上下文
2. 如果要使用Activity的上下文，则应该及时释放（如在onDestory时解除注册）
3. **属性动画导致的内存泄漏**

从Android 3.0开始，Google提供了属性动画，属性动画中有一类无线循环的动画，如果在Activity中播放此类动画且没有在onDestroy中停止动画，那么动画会一直播放下去，尽管已经无法在界面上看到动画效果了，并且这个时候Activity的View会被动画持有，而View又持有了Activity，最终Activity无法释放。

解决：

在Activity的onDestroy中调用animator.cancel()来停止动画；

属性动画的原理？属性动画与View动画的区别？属性动画在自定义View中的使用？

**4、非静态内部类导致内存泄露（常见）**

与handler一起讲，使用静态内部类+软引用代替非静态内部类

非静态内部类（包括匿名内部类）默认就会持有外部类的引用，当非静态内部类对象的生命周期比外部类对象的生命周期长时，就会导致内存泄露。

例如：handler内部类、Thread内部类

**5、Handler导致的内存泄漏**

**6、未取消注册或回调导致内存泄露**

观察者模式

**7、MVP框架中的内存泄漏**

MVP套路的同学应该会清除这么几点：

1.Model层获取数据

2.View层实现类执行回调的逻辑

3.Presenter层解除M和V的耦合，使M和V通过P层交互。

这么做肯定是有好处的，解除了M和V的耦合，他们俩互不感知，但是P层作为中间交互层不得不持有一个V层的引用和一个M层的实例。而当M层在进行一个耗时的操作时，由于P层是调用M层的逻辑实现一些功能，所以也可以将P层视为是一个耗时的操作。而且前面也说了，P层会持有一个V层的引用，如果在这个时候我们想要销毁这个Activity，那么这个Activity因为仍有P在持有Activity的引用从而导致其不会被回收，也就导致了内存泄露

可以这么解决：我们将presenter的生命周期和Activity的生命周期关联起来：

* 在presenter中声明一个onDestroy()方法，在这个方法中将testView置为null，然后在presenter中凡是使用到testView的使用的，都判断一下是否为空。
* 在activity的onDestroy()方法中调用presenter.onDestroy()，同时也将activity持有的presenter置空