Leak canary如何检测内存泄漏？

原理是我们常用的 WeakReference，其参数构造函数支持传入一个 ReferenceQueue，当其关联的对象回收时，会将 WeakReference 加入 ReferenceQueue 中。LeakCanary 的做法是继承 ReferenceQueue，增加一个值为 UUID 的属性 key，同时将每个需要监测的对象 WeakReference 以此 UUID 作为键加入一个 map 中。这样，在 GC 过后，removeWeaklyReachableObjects 方法通过遍历 ReferenceQueue，通过 key 值删除 map 中已回收的对象，剩下的对象就基本可以确定发生了内存泄露。

Leak canary初始化

AppWatcherInstaller通过继承ContentProvider在APP启动过程中进行初始化。AppWatcherInstaller有两个子类，MainProcess、LeakCanaryProcess。通过在Leak canary的Manifest文件中通过声明MainProcess来初始化AppWatcherInstaller。在AppWatcherInstaller初始化的时候将Application保存在AppWatcher中，AppWatcher作为监视器。

AppWatcher初始化

初始化中主要包括：1.通过反射获取InternalLeakCanary单例对象；2.注册四个InstallableWatcher观察对象，通过可达性分析封装类ReachabilityWatcher进行监测，包括Activity，Fragment, ViewModel, RootView, Service。

监测规则

ActivityWatcher：

注册的时候通过传入的Application调用registerActivityLifecycleCallbacks注册Activity 生命周期回调的监听，并在onActivityDestroyed回调中，通过调用reachabilityWatcher.expectWeaklyReachable将每个Activity对象加到观察列表。这个reachabilityWatcher就是通过可达性分析封装类ReachabilityWatcher。

ReachabilityWatcher：

ReachabilityWatcher的实现类是ObjectWatcher。在registerActivityLifecycleCallbacks接收到Activity销毁的时候会调用onActivityDestroyed。然后调用ObjectWatcher的expectWeaklyReachable方法，通过调用removeWeaklyReachableObjects将已经被 GC 的对象从 watchedObjects 集合中删除，然后将被监测的Activity对象、随机的UUID、描述语、当下时间封装成一个KeyedWeakReference对象并添加到集合中，它继承于 WeakReference，弱引用是不会阻止 GC 回收对象的，同时可以在构造函数中传递一个 ReferenceQueue，用于对象被 GC 后存放的队列。然后构建一个watchedObjects Map保存对应的UUID和对应的KeyedWeakReference待监测对象的映射。最终是通过mainHandler.postDelayed执行的moveToRetained方法。再次执行removeWeaklyReachableObjects将已经被 GC 的对象从 watchedObjects 集合中删除，如果watchedObjects中还保存被引用的Activity，那么就是泄漏的对象了，回调通知事件监听器onObjectRetainedListeners.forEach { it.onObjectRetained() }进行提示。

InternalLeakCanary初始化

InternalLeakCanary通过反射进行初始化的时候，最终会调用到InternalLeakCanary的invoke方法，在这里将自己作为监听器注册给了AppWatcher，AppWatcher.objectWatcher.addOnObjectRetainedListener(this)进行了onObjectRetainedListeners监听器的初始化。监听器的onObjectRetained方法通过调用scheduleRetainedObjectCheck方法调用了heapDumpTrigger.scheduleRetainedObjectCheck()，其中heapDumpTrigger在上面的invoke方法中有初始化。scheduleRetainedObjectCheck通过调用heapDumpTrigger封装的GcTrigger 对象，调用 Runtime.getRuntime().gc() 方法触发虚拟机进行 GC 操作。确认内存泄漏时，AndroidHeapDumper的dumpHeap方法通过调用 Debug.dumpHprofData() 方法从虚拟机中 dump hprof 文件。如果AndroidHeapDumper.dumpHeap的结果是HeapDump，会将结果转交给HeapAnalyzerService完成结果展示。

HeapAnalyzerService展示结果

HeapAnalyzerService继承了ForegroundService，ForegroundService又继承自IntentService，既然归根结底是IntentService，那么就要看看onHandleIntent方法是如何处理任务的。其中调用了analyzeHeap方法，其中HeapAnalyzer使用Shark来解析heapDumpFile。最后通过config.onHeapAnalyzedListener .onHeapAnalyzed(fullHeapAnalysis)触发相关通知显示以及点击后的 LeakActivity 数据展示。

FragmentAndViewModelWatcher

注册的时候通过传入的Application调用registerActivityLifecycleCallbacks注册Activity 生命周期回调的监听，并在onActivityCreated回调中，启动对Fragment和View的监测。

AndroidXFragmentDestroyWatcher

通过传递的Activity，在 activity 中注册了一个 FragmentLifecycleCallbacks，所以就能够监听到该 activity 中所有的 fragment 的生命周期。因为监听回调的参数传递的是递归注册，所以Fragment中包含Fragment的情况也可以监测到。Fragment与 Activity还不一样，它有两个 onDestroy 方法，一个是它自身的，一个是它的 view 的。当 fragment 的 onDestroyView 方法执行后，这个 View 应该被销毁，所以我们也要监测这个 view 是否泄露。因为很多情况下，Fragment的生命周期都只会走到 onDestroyView 这里，但是如果我们在内部类里面更新该 Fragment的界面的时候，就会出现内存泄露问题。

然后在onFragmentViewDestroyed方法监测Fragment中的View是否泄漏通过调用reachabilityWatcher.expectWeaklyReachable将每个View对象加到观察列表。这个reachabilityWatcher就是通过可达性分析封装类ReachabilityWatcher。在onFragmentDestroyed监测Fragment是否泄漏，通过调用reachabilityWatcher.expectWeaklyReachable将每个Fragment对象加到观察列表。这个reachabilityWatcher就是通过可达性分析封装类ReachabilityWatcher。

ViewModelClearedWatcher

在onFragmentCreated方法回调的时候注册ViewModelClearedWatcher.install(fragment, reachabilityWatcher)进行ViewModel的监测。

ViewModelClearedWatcher注册的时候会调用它的构造方法，使用反射拿到 activity 的 viewModeStore 的 mMap字段。可以理解为在 activity 的 viewModelStore 里面添加了一个 ViewModelClearedWatcher 对象。

当 activity 执行 onDestroy 方法的时候，viewModel 的 onClear 方法会执行。所以，我们可以在 ViewModelClearedWatcher 这个对象的 onClear 方法里面去判断其他的 viewModel 是否内存泄露了。通过调用reachabilityWatcher.expectWeaklyReachable将每个ViewModel对象加到观察列表。这个reachabilityWatcher就是通过可达性分析封装类ReachabilityWatcher。

RootViewWatcher

RootView 有多种，可以通过View的windowType进行分类：PHONE\_WINDOW(Activity、Dialog)、POPUP\_WINDOW(由于 popup\_window 通常会被缓存，所以没必要监测，当它 dismiss 后，还可能再显示出来

)、TOOLTIP(Tooltips可以实现类似pc端网页鼠标悬停时出现描述信息的功能，而到安卓中，如果给一个控件使用了Tooltips，那么当用户长按这个控件时，我们预设的描述信息就会悬浮出现在控件附件某个位置。)、TOAST。

所以，Dialog，TOOLTIP，TOAST 是需要监测的对象。

View想要显示出来，需要将自己添加到 window上，而 window 是由 WMS 管理的，app 与 WMS 是通过 Binder 通信，app 端的代理就是 WindowManagerGlobal 这个对象。所以，每个 activity，dialog，toast 显示的时候，都需要经过 WindowManagerGlobal 的 addView 方法，而所有的添加的 view 都存放在了 WindowManagerGlobal 的 mViews 字段里面。WindowManagerGlobal 会在View依附到自己的时候调用对应的回调。那我们可以在为每个 rootView 都添加一个 addOnAttachStateChangeListener 监听，在 onViewAttachedToWindow的时候移除这个监听，在onViewDetachedFromWindow 的时候重新去监听这个对象是否泄露。

Curtains

这是个桥接RootViewsSpy功能的类，本身没什么功能，它会将RootViewsSpy初始化。

RootViewsSpy

RootViewsSpy是一个工具类，里面包含了WindowManager更新的所有RootView集合。它是怎么获取到WindowManager的view呢，是通过WindowManagerSpy这个实现类。

WindowManagerSpy

通过反射获取WindowManager在app进程的实例，进而获取WindowManager的view集合，然后将他们设置给RootViewsSpy中的delegatingViewList集合。它重写了 list 的 add 与 removeAt 方法，在这两个方法执行的时候，调用了 listener 的对应方法。这个 listener 是RootViewWatcher添加的。通过调用reachabilityWatcher.expectWeaklyReachable将每个View对象加到观察列表。这个reachabilityWatcher就是通过可达性分析封装类ReachabilityWatcher。

ServiceWatcher

service 的泄露监测点要稍微复杂一点，涉及到两个方面：mH、ActivityManager。

初始化时通过反射获取ActivityThread类的currentActivityThread对象，进而获取到mServices数组。注册监听的时候替换ActivityThread类中的 mH 的 callback 字段为uninstallActivityThreadHandlerCallback，uninstall 的时候，需要将替换的字段替换回来。创建新的 callback 对象，替换原来的，在新的callback对象中，针对 STOP\_SERVICE 消息进行处理，从 activityThread 的 mServices 字段里面，拿到 Service 对象，将 service 用弱引用包装后放入 map。然后找到真正 service 执行 onDestroy 的地方。因为 handler 的 handleMessage 逻辑里面，会调用 ActivityManager.getService().serviceDoneExecuting() 方法，所以需要 hook ActivityManager，在这个方法里去监测 service 的泄露情况。通过反射获取ActivityManager实例保存在uninstallActivityManager，uninstall 的时候，需要将替换的字段替换回来。service 的 onDestroy 方法执行后，接下来就执行 ActivityManager 的 serviceDoneExecuting 方法。动态代理，hook ActivityManager 的 serviceDoneExecuting 方法，然后判断调用serviceDoneExecuting 的service是否保存在servicesToBeDestroyed数组中，serviceDoneExecuting 方法的参数里面只有 token，所以在 onServicePreDestroy 里面保存了 token - service 的键值对（弱引用map）。通过调用reachabilityWatcher.expectWeaklyReachable将每个Service对象加到观察列表。这个reachabilityWatcher就是通过可达性分析封装类ReachabilityWatcher。