LeakCanary: 让内存泄露无所遁形

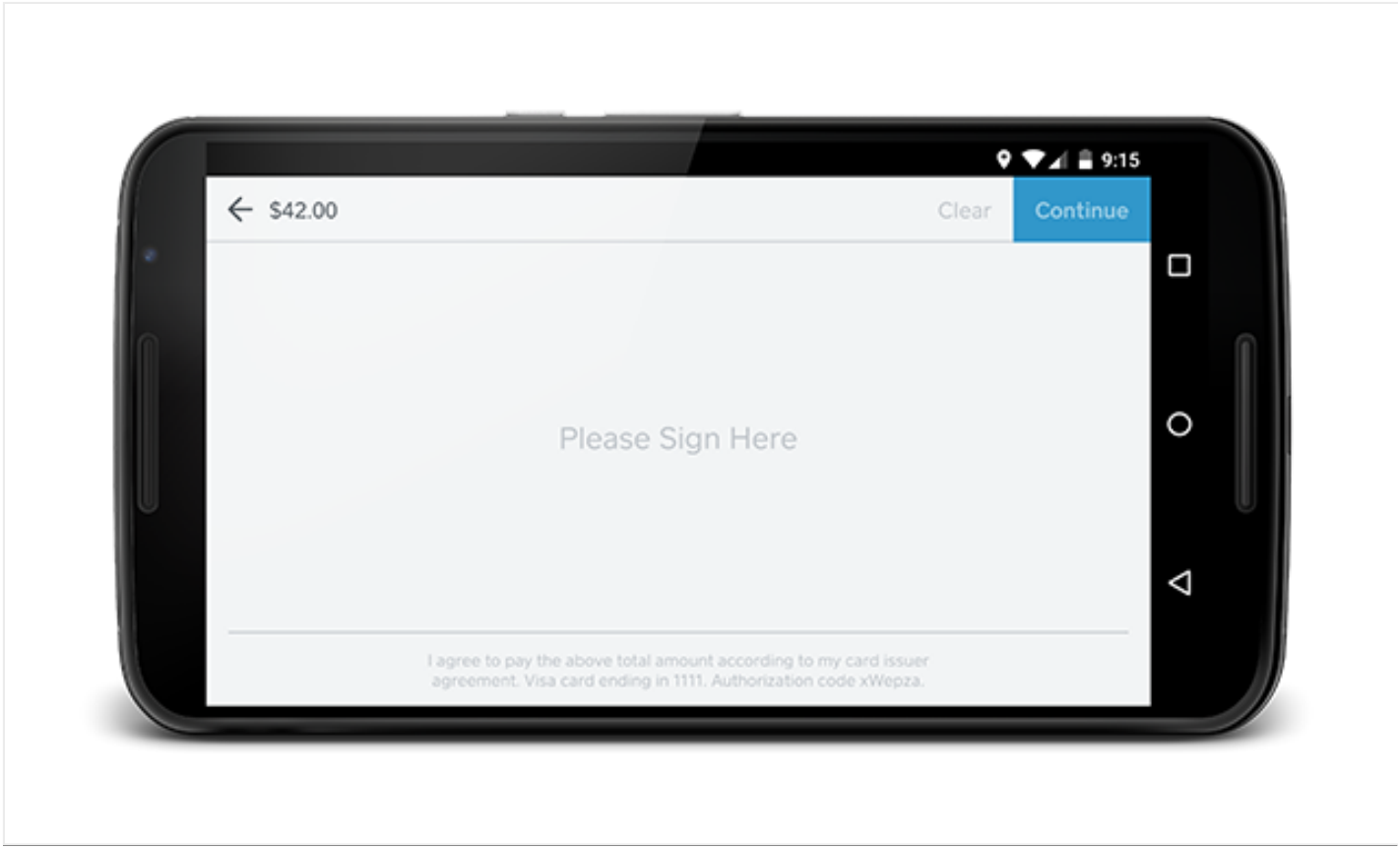
09 May 2015

> 本文为LeakCanary: Detect all memory leaks!的翻译。原文在:
> https://corner.squareup.com/2015/05/leak-canary.html

```
java.lang.OutOfMemoryError
        at android.graphics.Bitmap.nativeCreate(Bitmap.java:-2)
        at android.graphics.Bitmap.createBitmap(Bitmap.java:689)
        at com.squareup.ui.SignView.createSignatureBitmap(SignView.java:121)
```

谁也不会喜欢 `OutOfMemoryError`

在 Square Register 中, 在签名页面，我们把客户的签名画在 bitmap cache 上。 这个 bitmap 的尺寸几乎和屏幕的尺寸一样大，在创建这个 bitmap 对象时，经常会引发 `OutOfMemoryError`，简称`OOM`。



当时，我们尝试过一些解决方案，但都没解决问题

- 使用 Bitmap.Config.ALPHA_8 因为，签名仅有黑色。
- 捕捉 `OutOfMemoryError`，尝试 GC 并重试（受 GCUtils 启发）。
- 我们没想过在 Java heap 内存之外创建 bitmap 。苦逼的我们，那会 Fresco 还不存在。

路子走错了

其实 bitmap 的尺寸不是真正的问题，当内存吃紧的时候，到处都有可能引发 OO。在创建大对象，比如 bitmap 的时候，更有可能发生。OOM 只是一个表象，更深层次的问题可能是: **内存泄露**。

什么是内存泄露

一些对象有着有限的生命周期。当这些对象所要做的事情完成了，我们希望他们会被回收掉。但是如果有一系列对这个对象的引用，那么在我们期待这个对象生命周期结束的时候被收回的时候，它是不会被回收的。它还会占用内存，这就造成了内存泄露。持续累加，内存很快被耗尽。

比如，当 `Activity.onDestroy` 被调用之后，activity 以及它涉及到的 view 和相关的 bitmap 都应该被回收。但是，如果有一个后台线程持有这个 activity 的引用，那么 activity 对应的内存就不能被回收。这最终将会导致内存耗尽，然后因为 OOM 而 crash。

对战内存泄露

排查内存泄露是一个全手工的过程，这在 Raizlabs 的 Wrangling Dalvik 系列文章中有详细描述。

以下几个关键步骤：

1. 通过 Bugsnag, Crashlytics 或者 Developer Console 等统计平台，了解 `OutOfMemoryError` 情况。
2. 重现问题。为了重现问题，机型非常重要，因为一些问题只在特定的设备上会出现。为了找到特定的机型，你需要想尽一切办法，你可能需要去买，去借，甚至去偷。 当然，为了确定复现步骤，你需要一遍一遍地去尝试。一切都是非常原始和粗暴的。
3. 在发生内存泄露的时候，把内存 Dump 出来。具体看这里。
4. 然后，你需要在 MAT 或者 YourKit 之类的内存分析工具中反复查看，找到那些原本该被回收掉的对象。
5. 计算这个对象到 GC roots 的最短强引用路径。
6. 确定引用路径中的哪个引用是不该有的，然后修复问题。

很复杂对吧?

如果有一个类库能在发生 OOM 之前把这些事情全部都搞定，然后你只要修复这些问题就好了，岂不妙哉！

LeakCanary

LeakCanary 是一个检测内存泄露的开源类库。你可以在 debug 包种轻松检测内存泄露。

先看一个例子:

```
class Cat {
}

class Box {
  Cat hiddenCat;
}
class Docker {
    // 静态变量，将不会被回收，除非加载 Docker 类的 ClassLoader 被回收。
    static Box container;
}

// ...
```

```
Box box = new Box();

// 薛定谔之猫
Cat schrodingerCat = new Cat();
box.hiddenCat = schrodingerCat;
Docker.container = box;
```

创建一个`RefWatcher`，监控对象引用情况。

```
// 我们期待薛定谔之猫很快就会消失（或者不消失），我们监控一下
refWatcher.watch(schrodingerCat);
```

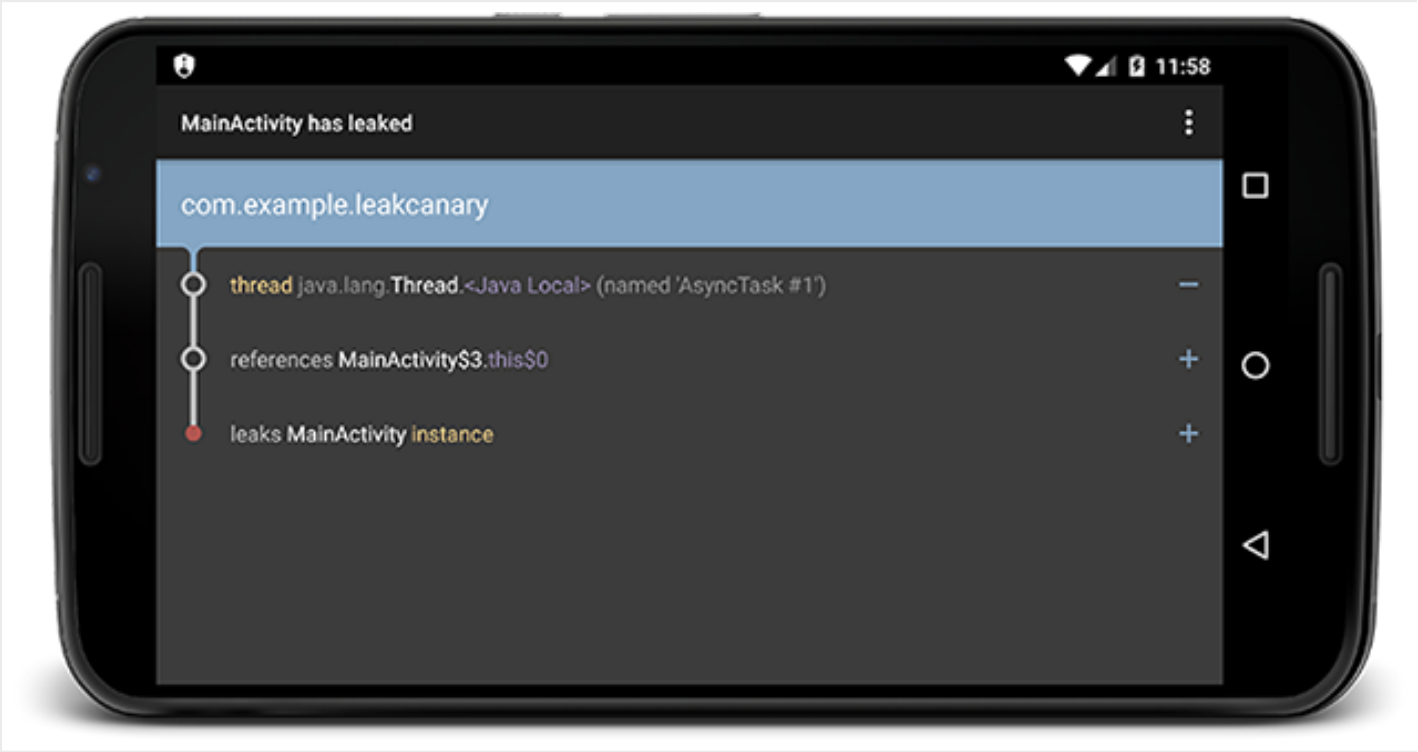
当发现有内存泄露的时候，你会看到一个很漂亮的 leak trace 报告:

- GC ROOT static Docker.container
- references Box.hiddenCat
- leaks Cat instance

我们知道，你很忙，每天都有一大堆需求。所以我们把这个事情弄得很简单，你只需要添加一行代码就行了。然后 LeakCanary 就会自动侦测 activity 的内存泄露了。

```
public class ExampleApplication extends Application {
  @Override public void onCreate() {
    super.onCreate();
    LeakCanary.install(this);
  }
}
```

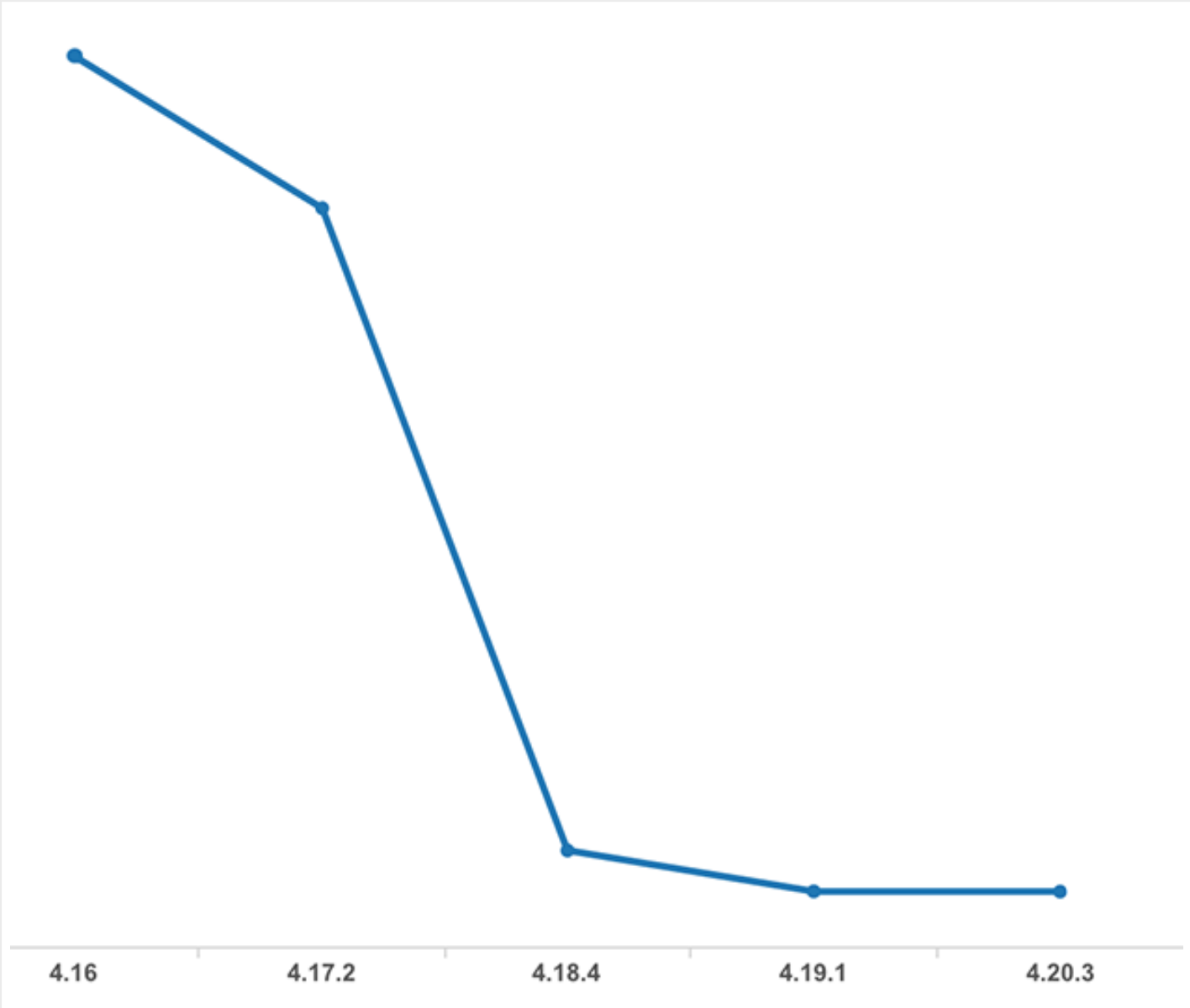
然后你会在通知栏看到这样很漂亮的一个界面:



结论

使用 LeakCanary 之后，我们修复了我们 APP 中相当多的内存泄露。我们甚至发现了 Android SDK 中的一些内存泄露问题。

结果是惊艳的，我们减少了 94% 的由 OOM 导致的 crash。



如果你也想消灭 OOM crash，那还犹豫什么，赶快使用 LeakCanary

相关链接:

- LeakCanary 中文使用说明
- 一个非常简单的 LeakCanary demo: https://github.com/liaohuqiu/leakcanary-demo