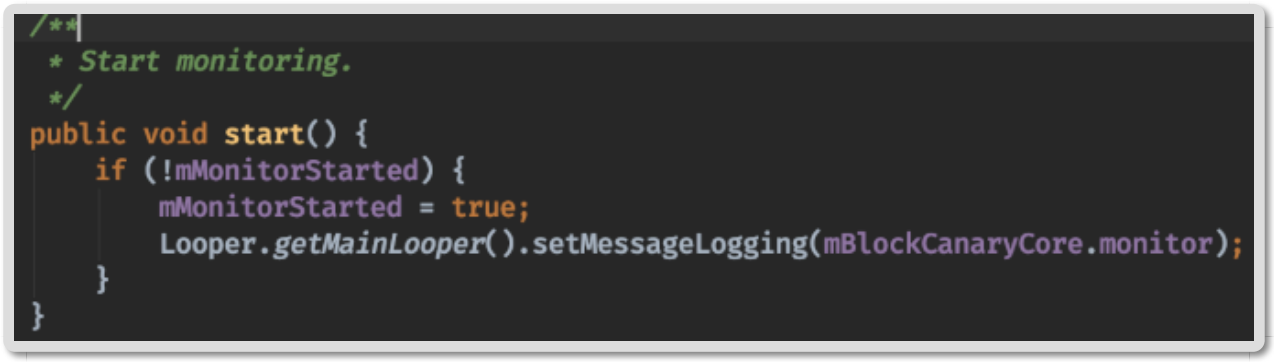
HenCoderPlus 讲义

BlockCanary 源码解析

BlockCanary 的原理

在调用 `start()` 时，通过调用主线程的 `Looper.setMessageLogging()` 方法，为 `Looper` 的 `mLogging` 成员变量赋值。



```
/**
 * Start monitoring.
 */
public void start() {
    if (!mMonitorStarted) {
        mMonitorStarted = true;
        Looper.getMainLooper().setMessageLogging(mBlockCanaryCore.monitor);
    }
}
```

在 `Looper` 死循环中， `println` 方法分别会在 `dispatchMessage(msg)` 之前和之后被调用。

```
public static void loop() {
    final Looper me = myLooper();
    ...

    for (;;) {
        Message msg = queue.next(); // might block
        ...

        // This must be in a local variable, in case a UI event sets the logger
        final Printer logging = me.mLogging;
        if (logging != null) {
            logging.println(">>>>> Dispatching to " + msg.target + " " +
                    msg.callback + ": " + msg.what);
        }
        ...
        try {
            msg.target.dispatchMessage(msg);
        } catch (Exception exception) {
        ...
        } finally {
        ...
        }
        ...

        if (logging != null) {
            logging.println("<<<<< Finished to " + msg.target + " " + msg.callback);
        }
        ...
    }
}
```

所以通过自定义 `Printer` 对象，我们就可以获得 `dispatchMessage` 的耗时，从而判断出是否有应用卡顿。

```
@Override
public void println(String x) {
    if (mStopWhenDebugging && Debug.isDebuggerConnected()) {
        return;
    }
    if (!mPrintingStarted) {
        mStartTimestamp = System.currentTimeMillis();
        mStartThreadTimestamp = SystemClock.currentThreadTimeMillis();
        mPrintingStarted = true;
        startDump();
    } else {
        final long endTime = System.currentTimeMillis();
        mPrintingStarted = false;
        if (isBlock(endTime)) {
            notifyBlockEvent(endTime);
        }
        stopDump();
    }
}
```

同时 BlockCanary 还会在子线程中执行一个获取主线程堆栈信息的定时任务，这个任务会在 dispatchMessage 结束的时候被移除。

```
protected void doSample() {
    StringBuilder stringBuilder = new StringBuilder();

    for (StackTraceElement stackTraceElement : mCurrentThread.getStackTrace()) {
        stringBuilder
                .append(stackTraceElement.toString())
                .append(BlockInfo.SEPARATOR);
    }

    synchronized (sStackMap) {
        if (sStackMap.size() == mMaxEntryCount && mMaxEntryCount > 0) {
            sStackMap.remove(sStackMap.keySet().iterator().next());
        }
        sStackMap.put(System.currentTimeMillis(), stringBuilder.toString());
    }
}
```

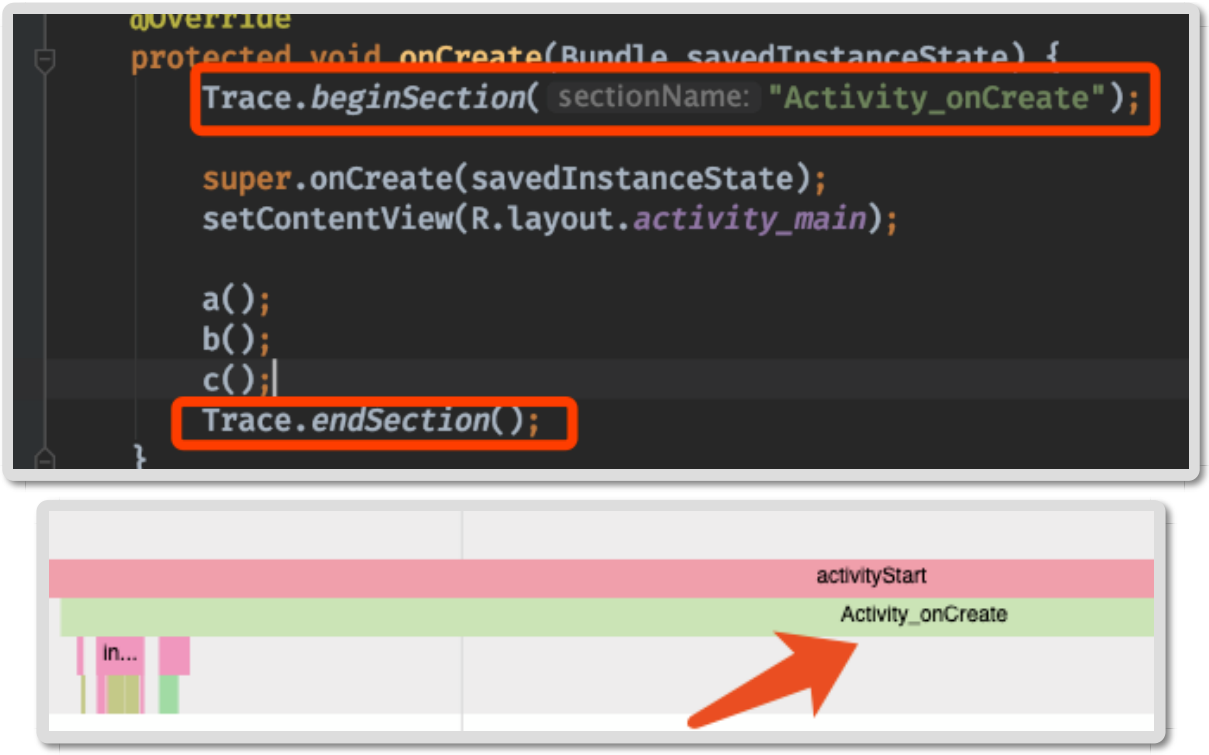
BlockCanary 的缺陷

- 依靠定时获取堆栈的方法，定位不够精准。
- `println` 方法中会拼接字符串对象

获取方法运行时间

- Hugo
- TraceView
- Systrace
    - 使用 python 终端命令生成 Trace 文件 官方文档
    - 在高版本中，可以通过 System tracing 生成 trace 文件，生成的文件可以在 这里 在线分析

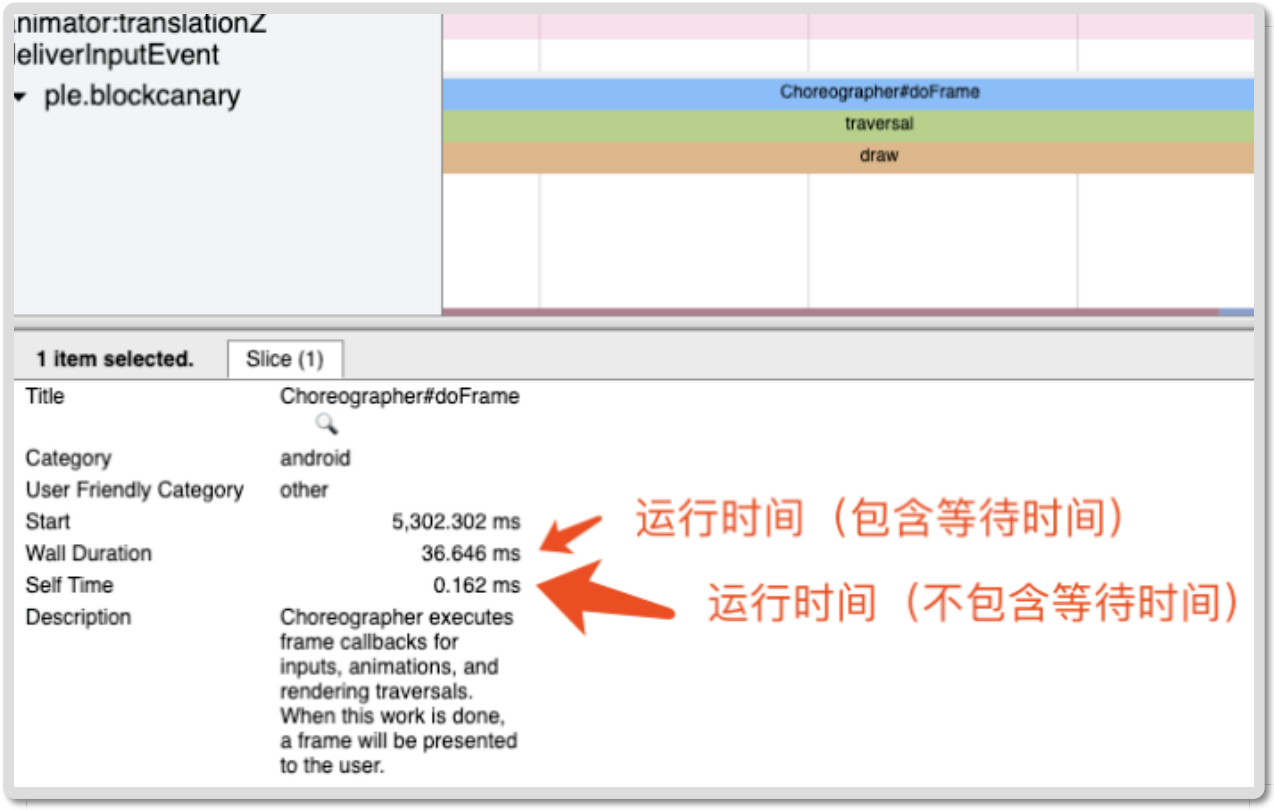
  在代码中主动做标记：



Trace 分析界面常用操作：

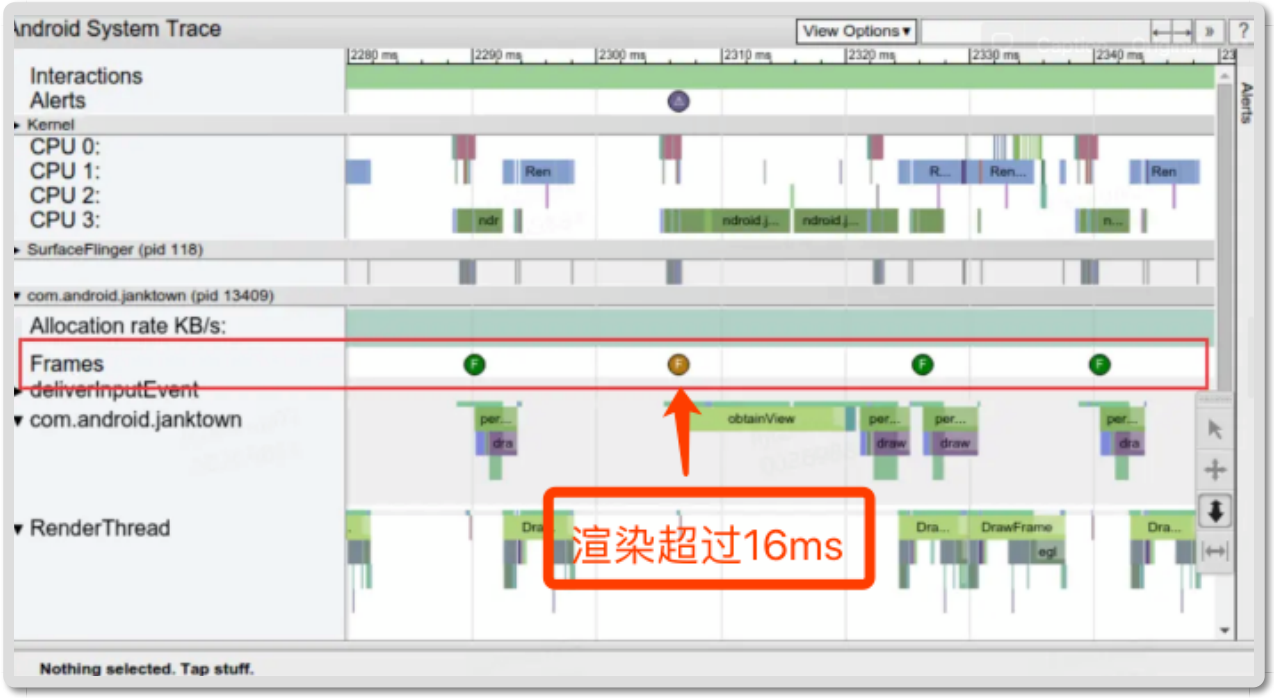
- W： 放大 （加 Shift 效果加倍）
- S： 缩小 （加 Shift 效果加倍）
- A： 左移
- D： 右移
- M：标记当前选中的时间线
- 1： 选中区域
- 2： 拖拽
- 3： 放大缩小
- 4： 裁剪时间线

查看函数时间：



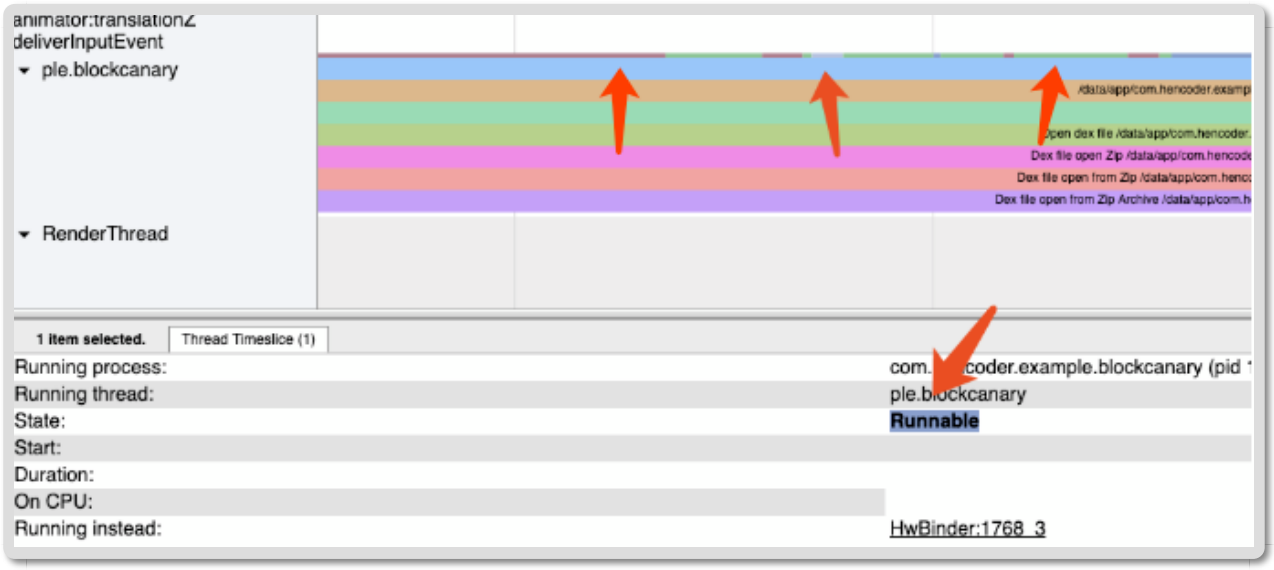
查看绘制帧：

绿色表示在 16ms 内完成，黄色和红色表示超过 16 ms。



查看绘制状态：

- 绿色（Running）表示运行中
- 蓝色（Runnbale）表示可以被运行但是没有分配到 CPU
- 灰色（白色）（Sleeping）
- 桔红色（Uninterruptible Sleep）表示在执行 I/O 操作



自定义 Plugin

代码已上传到 github

重点：在 transform 方法中，遍历文件夹。

```
override fun transform(transformInvocation: TransformInvocation) {
    transformInvocation.inputs.forEach(Consumer { transformInput ->
        transformInput.directoryInputs.forEach { directoryInput ->
            traceDirectoryFiles(directoryInput, transformInvocation.outputProvider)
        }
        transformInput.jarInputs.forEach { jarInput ->
            traceJarFiles(jarInput, transformInvocation.outputProvider)
        }
    })
}
```

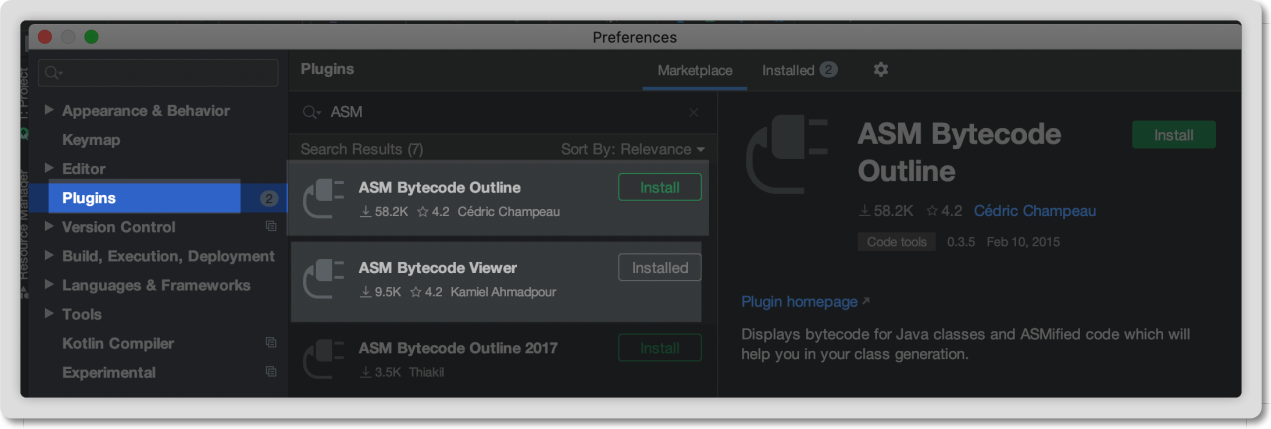
然后通过 ASM 对 class 文件进行处理

```
private fun traceDirectoryFiles(directoryInput: DirectoryInput, outputProvider: TransformO
    directoryInput.file.walkTopDown()
            .filter { it.isFile }
            .forEach { file ->
                FileInputStream(file).use { fis ->
                    val classReader = ClassReader(fis)
                    val classWriter = ClassWriter(classReader, ClassWriter.COMPUTE_MAXS)
                    val classTraceVisitor = ClassTraceVisitor(classWriter)
                    classReader.accept(classTraceVisitor, EXPAND_FRAMES)

                    file.writeBytes(classWriter.toByteArray())
                }
            }
    val dest : File! = outputProvider.getContentLocation(
            directoryInput.name,
            directoryInput.contentTypes,
            directoryInput.scopes,
            Format.DIRECTORY)

    FileUtils.copyDirectory(directoryInput.file, dest)
}
```

查看 ASM 字节码的工具 (两个都可以)



问题和建议?

课上技术相关的问题，都可以去群里和大家讨论，对于比较通用的、有价值的问题，可以去我们的知识星球提问。

具体技术之外的问题和建议，都可以找丢物线（微信：diuwuxian），丢丢会为你解答技术以外的一切。

觉得好?

如果你觉得课程很棒，欢迎给我们好评呀！https://ke.qq.com/comment/index.html?cid=381952

> 一定要是你真的觉得好，再给我们好评。不要仅仅因为对扔物线的支持而好评（报名课程已经是你最大的支持了，再不够的话 B 站多来点三连我也很开心），另外我们也坚决不做好评返现等任何的交易。我们只希望，在课程对你有帮助的前提下，可以看到你温暖的评价。

更多内容：

- 网站：https://hencoder.com；https://kaixue.io
- 各大搜索引擎、微信公众号、微博、知乎、掘金、哔哩哔哩、YouTube、西瓜视频、抖音、快手、微视：统一账号「**扔物线**」，我会持续输出优质的技术内容，欢迎大家关注。
- 哔哩哔哩快捷传送门：https://space.bilibili.com/27559447

大家如果喜欢我们的课程，还请去扔物线的哔哩哔哩，帮我素质三连，感谢大家！