# 内存优化

Memory Profiler 是 Android Profiler 中的一个组件，可帮助您识别导致应用卡顿、冻结甚至崩溃的内存泄漏和流失。 它显示一个应用内存使用量的实时图表，让您可以捕获堆转储、强制执行垃圾回收以及跟踪内存分配。

一、 为什么应分析您的应用内存

Android 提供一个托管内存环境—当它确定您的应用不再使用某些对象时，垃圾回收器会将未使用的内存释放回堆中。 虽然 Android 查找未使用内存的方式在不断改进，但对于所有 Android 版本，系统都必须在某个时间点短暂地暂停您的代码。 大多数情况下，这些暂停难以察觉。 不过，如果您的应用分配内存的速度比系统回收内存的速度快，则当收集器释放足够的内存以满足您的分配需要时，您的应用可能会延迟。 此延迟可能会导致您的应用跳帧，并使系统明显变慢。

尽管您的应用不会表现出变慢，但如果存在内存泄漏，则即使应用在后台运行也会保留该内存。 此行为会强制执行不必要的垃圾回收 Event，因而拖慢系统的内存性能。 最后，系统被迫终止您的应用进程以回收内存。 然后，当用户返回您的应用时，它必须完全重启。

为帮助防止这些问题，您应使用 Memory Profiler 执行以下操作：

在时间线中查找可能会导致性能问题的不理想的内存分配模式。

转储 Java 堆以查看在任何给定时间哪些对象耗尽了使用内存。 长时间进行多个堆转储可帮助识别内存泄漏。

记录正常用户交互和极端用户交互期间的内存分配以准确识别您的代码在何处短时间分配了过多对象，或分配了泄漏的对象。

如需了解可减少应用内存使用的编程做法，请阅读管理您的应用内存。

二、 Memory Profiler 概览

当您首次打开 Memory Profiler 时，您将看到一条表示应用内存使用量的详细时间线，并可访问用于强制执行垃圾回收、捕捉堆转储和记录内存分配的各种工具。

图 1. Memory Profiler

如图 1 所示，Memory Profiler 的默认视图包括以下各项：

用于强制执行垃圾回收 Event 的按钮。

用于捕获堆转储的按钮。

用于记录内存分配情况的按钮。 此按钮仅在连接至运行 Android 7.1 或更低版本的设备时才会显示。

用于放大/缩小时间线的按钮。

用于跳转至实时内存数据的按钮。

Event 时间线，其显示 Activity 状态、用户输入 Event 和屏幕旋转 Event。

内存使用量时间线，其包含以下内容：

一个显示每个内存类别使用多少内存的堆叠图表，如左侧的 y 轴以及顶部的彩色键所示。

虚线表示分配的对象数，如右侧的 y 轴所示。

用于表示每个垃圾回收 Event 的图标。

不过，如果您使用的是运行 Android 7.1 或更低版本的设备，则默认情况下，并不是所有分析数据均可见。 如果您看到一条消息，其显示“Advanced profiling is unavailable for the selected process”，则需要启用高级分析以查看下列内容：

Event 时间线

分配的对象数

垃圾回收 Event

在 Android 8.0 及更高版本上，始终为可调试应用启用高级分析。

三、 如何计算内存

您在 Memory Profiler（图 2）顶部看到的数字取决于您的应用根据 Android 系统机制所提交的所有私有内存页面数。 此计数不包含与系统或其他应用共享的页面。

图 2. Memory Profiler 顶部的内存计数图例

内存计数中的类别如下所示：

Java：从 Java 或 Kotlin 代码分配的对象内存。

Native：从 C 或 C++ 代码分配的对象内存。

即使您的应用中不使用 C++，您也可能会看到此处使用的一些原生内存，因为 Android 框架使用原生内存代表您处理各种任务，如处理图像资源和其他图形时，即使您编写的代码采用 Java 或 Kotlin 语言。

Graphics：图形缓冲区队列向屏幕显示像素（包括 GL 表面、GL 纹理等等）所使用的内存。 （请注意，这是与 CPU 共享的内存，不是 GPU 专用内存。）

Stack： 您的应用中的原生堆栈和 Java 堆栈使用的内存。 这通常与您的应用运行多少线程有关。

Code：您的应用用于处理代码和资源（如 dex 字节码、已优化或已编译的 dex 码、.so 库和字体）的内存。

Other：您的应用使用的系统不确定如何分类的内存。

Allocated：您的应用分配的 Java/Kotlin 对象数。 它没有计入 C 或 C++ 中分配的对象。

当连接至运行 Android 7.1 及更低版本的设备时，此分配仅在 Memory Profiler 连接至您运行的应用时才开始计数。 因此，您开始分析之前分配的任何对象都不会被计入。 不过，Android 8.0 附带一个设备内置分析工具，该工具可记录所有分配，因此，在 Android 8.0 及更高版本上，此数字始终表示您的应用中待处理的 Java 对象总数。

与以前的 Android Monitor 工具中的内存计数相比，新的 Memory Profiler 以不同的方式记录您的内存，因此，您的内存使用量现在看上去可能会更高些。 Memory Profiler 监控的类别更多，这会增加总的内存使用量，但如果您仅关心 Java 堆内存，则“Java”项的数字应与以前工具中的数值相似。

然而，Java 数字可能与您在 Android Monitor 中看到的数字并非完全相同，这是因为应用的 Java 堆是从 Zygote 启动的，而新数字则计入了为它分配的所有物理内存页面。 因此，它可以准确反映您的应用实际使用了多少物理内存。

注：目前，Memory Profiler 还会显示应用中的一些误报的原生内存使用量，而这些内存实际上是分析工具使用的。 对于大约 100000 个对象，最多会使报告的内存使用量增加 10MB。 在这些工具的未来版本中，这些数字将从您的数据中过滤掉。

四、 查看内存分配

内存分配显示内存中每个对象是如何分配的。 具体而言，Memory Profiler 可为您显示有关对象分配的以下信息：

分配哪些类型的对象以及它们使用多少空间。

每个分配的堆叠追踪，包括在哪个线程中。

对象在何时被取消分配（仅当使用运行 Android 8.0 或更高版本的设备时）。

如果您的设备运行 Android 8.0 或更高版本，您可以随时按照下述方法查看您的对象分配： 只需点击并按住时间线，并拖动选择您想要查看分配的区域（如视频 1 中所示）。 不需要开始记录会话，因为 Android 8.0 及更高版本附带设备内置分析工具，可持续跟踪您的应用分配。

视频 1. 对于Android 8.0 及更高版本，选择一个现有时间线区域以查看对象分配

如果您的设备运行 Android 7.1 或更低版本，则在 Memory Profiler 工具栏中点击 Record memory allocations

。 记录时，Android Monitor 将跟踪您的应用中进行的所有分配。 操作完成后，点击 Stop recording

（同一个按钮；请参阅视频 2）以查看分配。

视频 2. 对于 Android 7.1 及更低版本，您必须显式记录内存分配

在选择一个时间线区域后（或当您使用运行 Android 7.1 或更低版本的设备完成记录会话时），已分配对象的列表将显示在时间线下方，按类名称进行分组，并按其堆计数排序。

注：在 Android 7.1 及更低版本上，您最多可以记录 65535 个分配。 如果您的记录会话超出此限值，则记录中仅保存最新的 65535 个分配。 （在 Android 8.0 及更高版本中，则没有实际的限制。）

要检查分配记录，请按以下步骤操作：

浏览列表以查找堆计数异常大且可能存在泄漏的对象。 为帮助查找已知类，点击 Class Name 列标题以按字母顺序排序。 然后点击一个类名称。 此时在右侧将出现 Instance View 窗格，显示该类的每个实例，如图 3 中所示。

在 Instance View 窗格中，点击一个实例。 此时下方将出现 Call Stack 标签，显示该实例被分配到何处以及哪个线程中。

在 Call Stack 标签中，点击任意行以在编辑器中跳转到该代码。

图 3. 有关每个已分配对象的详情显示在右侧的 \*\*Instance View\*\* 中

默认情况下，左侧的分配列表按类名称排列。 在列表顶部，您可以使用右侧的下拉列表在以下排列方式之间进行切换：

Arrange by class：基于类名称对所有分配进行分组。

Arrange by package：基于软件包名称对所有分配进行分组。

Arrange by callstack：将所有分配分组到其对应的调用堆栈。

五、 捕获堆转储

堆转储显示在您捕获堆转储时您的应用中哪些对象正在使用内存。 特别是在长时间的用户会话后，堆转储会显示您认为不应再位于内存中却仍在内存中的对象，从而帮助识别内存泄漏。 在捕获堆转储后，您可以查看以下信息：

您的应用已分配哪些类型的对象，以及每个类型分配多少。

每个对象正在使用多少内存。

在代码中的何处仍在引用每个对象。

对象所分配到的调用堆栈。 （目前，如果您在记录分配时捕获堆转储，则只有在 Android 7.1 及更低版本中，堆转储才能使用调用堆栈。）

图 4. 查看堆转储

要捕获堆转储，在 Memory Profiler 工具栏中点击 Dump Java heap

。 在转储堆期间，Java 内存量可能会暂时增加。 这很正常，因为堆转储与您的应用发生在同一进程中，并需要一些内存来收集数据。

堆转储显示在内存时间线下，显示堆中的所有类类型，如图 4 所示。

注：如果您需要更精确地了解转储的创建时间，可以通过调用dumpHprofData()在应用代码的关键点创建堆转储。

要检查您的堆，请按以下步骤操作：

浏览列表以查找堆计数异常大且可能存在泄漏的对象。 为帮助查找已知类，点击 Class Name 列标题以按字母顺序排序。 然后点击一个类名称。 此时在右侧将出现 Instance View 窗格，显示该类的每个实例，如图 5 中所示。

在 Instance View 窗格中，点击一个实例。此时下方将出现 References，显示该对象的每个引用。

或者，点击实例名称旁的箭头以查看其所有字段，然后点击一个字段名称查看其所有引用。 如果您要查看某个字段的实例详情，右键点击该字段并选择 Go to Instance。

在 References 标签中，如果您发现某个引用可能在泄漏内存，则右键点击它并选择 Go to Instance。 这将从堆转储中选择对应的实例，显示您自己的实例数据。

默认情况下，堆转储不会向您显示每个已分配对象的堆叠追踪。 要获取堆叠追踪，在点击 Dump Java heap 之前，您必须先开始记录内存分配。 然后，您可以在 Instance View 中选择一个实例，并查看 Call Stack 标签以及 References 标签，如图 5 所示。不过，在您开始记录分配之前，可能已分配一些对象，因此，调用堆栈不能用于这些对象。 包含调用堆栈的实例在图标

上用一个“堆栈”标志表示。 （遗憾的是，由于堆叠追踪需要您执行分配记录，因此，您目前无法在 Android 8.0 上查看堆转储的堆叠追踪。）

在您的堆转储中，请注意由下列任意情况引起的内存泄漏：

长时间引用 Activity、Context、View、Drawable 和其他对象，可能会保持对 Activity 或 Context 容器的引用。

可以保持 Activity 实例的非静态内部类，如 Runnable。

对象保持时间超出所需时间的缓存。

图 5. 捕获堆转储需要的持续时间标示在时间线中

在类列表中，您可以查看以下信息：

Heap Count：堆中的实例数。

Shallow Size：此堆中所有实例的总大小（以字节为单位）。

Retained Size：为此类的所有实例而保留的内存总大小（以字节为单位）。

Allocations：在选定时间段内通过 malloc() 或 new 运算符分配的对象数。

Deallocations：在选定时间段内通过 free() 或 delete 运算符解除分配的对象数。

Allocations Size：在选定时间段内所有分配的总大小（以字节为单位）。

Deallocations Size：在选定时间段内所有已释放内存的总大小（以字节为单位）。

Total Count：Allocations 列中的值减去 Deallocations 列中的值所得的结果。

Remaining Size：Allocations Size 列中的值减去 Deallocations Size 列中的值所得的结果。

Shallow Size Change：选定时间内堆中实例（字节单位）大小变化

在类列表顶部，您可以使用左侧下拉列表在以下堆转储之间进行切换：

Default heap：系统未指定堆时。

App heap：您的应用在其中分配内存的主堆。

Image heap：系统启动映像，包含启动期间预加载的类。 此处的分配保证绝不会移动或消失。

Zygote heap：写时复制堆，其中的应用进程是从 Android 系统中派生的。

默认情况下，此堆中的对象列表按类名称排列。 您可以使用其他下拉列表在以下排列方式之间进行切换：

Arrange by class：基于类名称对所有分配进行分组。

Arrange by package：基于软件包名称对所有分配进行分组。

Arrange by callstack：将所有分配分组到其对应的调用堆栈。 此选项仅在记录分配期间捕获堆转储时才有效。 即使如此，堆中的对象也很可能是在您开始记录之前分配的，因此这些分配会首先显示，且只按类名称列出。

默认情况下，此列表按 Retained Size 列排序。 您可以点击任意列标题以更改列表的排序方式。

在 Instance View 中，每个实例都包含以下信息：

Depth：从任意 GC 根到所选实例的最短 hop 数。

Shallow Size：此实例的大小。

Retained Size：此实例支配的内存大小（根据 dominator 树）。

六、 将堆转储另存为 HPROF

在捕获堆转储后，仅当分析器运行时才能在 Memory Profiler 中查看数据。 当您退出分析会话时，您将丢失堆转储。 因此，如果您要保存堆转储以供日后查看，可通过点击时间线下方工具栏中的 Export heap dump as HPROF file

，将堆转储导出到一个 HPROF 文件中。 在显示的对话框中，确保使用 .hprof 后缀保存文件。

然后，通过将此文件拖到一个空的编辑器窗口（或将其拖到文件标签栏中），您可以在 Android Studio 中重新打开该文件。

要使用其他 HPROF 分析器如 jhat，您需要将 HPROF 文件从 Android 格式转换为 Java SE HPROF 格式。 您可以使用 android\_sdk/platform-tools/目录中提供的 hprof-conv 工具执行此操作。 运行包括以下两个参数的 hprof-conv命令：原始 HPROF 文件和转换后 HPROF 文件的写入位置。 例如：

hprof-conv heap-original.hprof heap-converted.hprof

七、 分析内存的技巧

使用 Memory Profiler 时，您应对应用代码施加压力并尝试强制内存泄漏。 在应用中引发内存泄漏的一种方式是，先让其运行一段时间，然后再检查堆。 泄漏在堆中可能逐渐汇聚到分配顶部。 不过，泄漏越小，您越需要运行更长时间的应用才能看到泄漏。

您还可以通过以下方式之一触发内存泄漏：

将设备从纵向旋转为横向，然后在不同的 Activity 状态下反复操作多次。 旋转设备经常会导致应用泄漏 Activity、Context 或 View 对象，因为系统会重新创建 Activity，而如果您的应用在其他地方保持对这些对象之一的引用，系统将无法对其进行垃圾回收。

处于不同的 Activity 状态时，在您的应用与另一个应用之间切换（导航到主屏幕，然后返回到您的应用）。

提示： 您还可以使用 monkeyrunner 测试框架执行上述步骤。

# 文档

使用内存分析器查看应用的内存使用情况

<https://developer.android.google.cn/studio/profile/memory-profiler?hl=zh-cn>

Android内存泄漏监控

<https://www.jianshu.com/p/b4b055301a95>

Android LeakCanary使用详细教程

<https://blog.csdn.net/chennai1101/article/details/103798870>

Android LruCache

<https://blog.csdn.net/mChenys/article/details/127017788>

Android 缓存浅谈（二） DiskLruCache

<https://ww.dandelioncloud.cn/article/details/1573892377058439170>

Android DiskLruCache完全解析，硬盘缓存的最佳方案

<https://ww.dandelioncloud.cn/article/details/1534880041037230081>

Android 缓存浅谈（二） DiskLruCache

<https://ww.dandelioncloud.cn/article/details/1573892377058439170>

式样

微软雅黑 Light 5号

段落 固定值 23磅

# 目录1

## 目录2

### 目录3

#### 目录4

##### 目录5