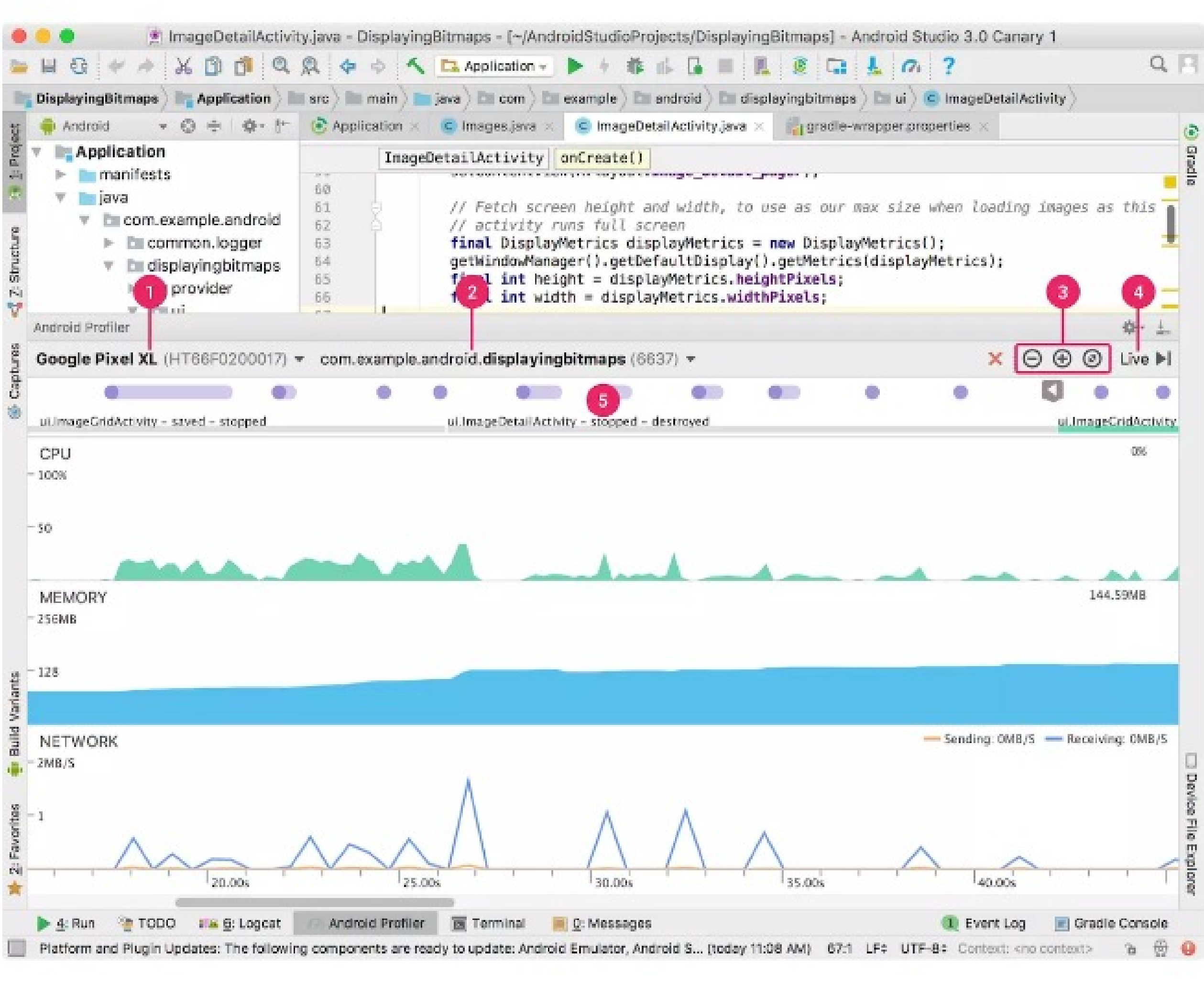
AS3.0 Profiler 性能分析利器

今年早些时候as升级到了3.0版本，自己前两天升级了，发现之前我们熟悉的 Android Monitor 不在了，取而代之的是Android Profiler，就参照官方文档过了一遍。

1:Android Profiler的是使用流程：

1.点击工具栏的图标即可打开

在Android Profiler窗口的顶部，如图所示，选择设备❶和你想要配置的app进程❷当我们连接一个设备后，打开Android Profiles运行我们的应用程序时，它会默认选中我们的程序，Android Profiler显示如图1（图片来自官网），



```
// Fetch screen height and width, to use as our max size when loading images as this
// activity runs full screen
final DisplayMetrics displayMetrics = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
final int height = displayMetrics.heightPixels;
final int width = displayMetrics.widthPixels;
```

Platform and Plugin Updates: The following components are ready to update: Android Emulator, Android S... (today 11:08 AM)

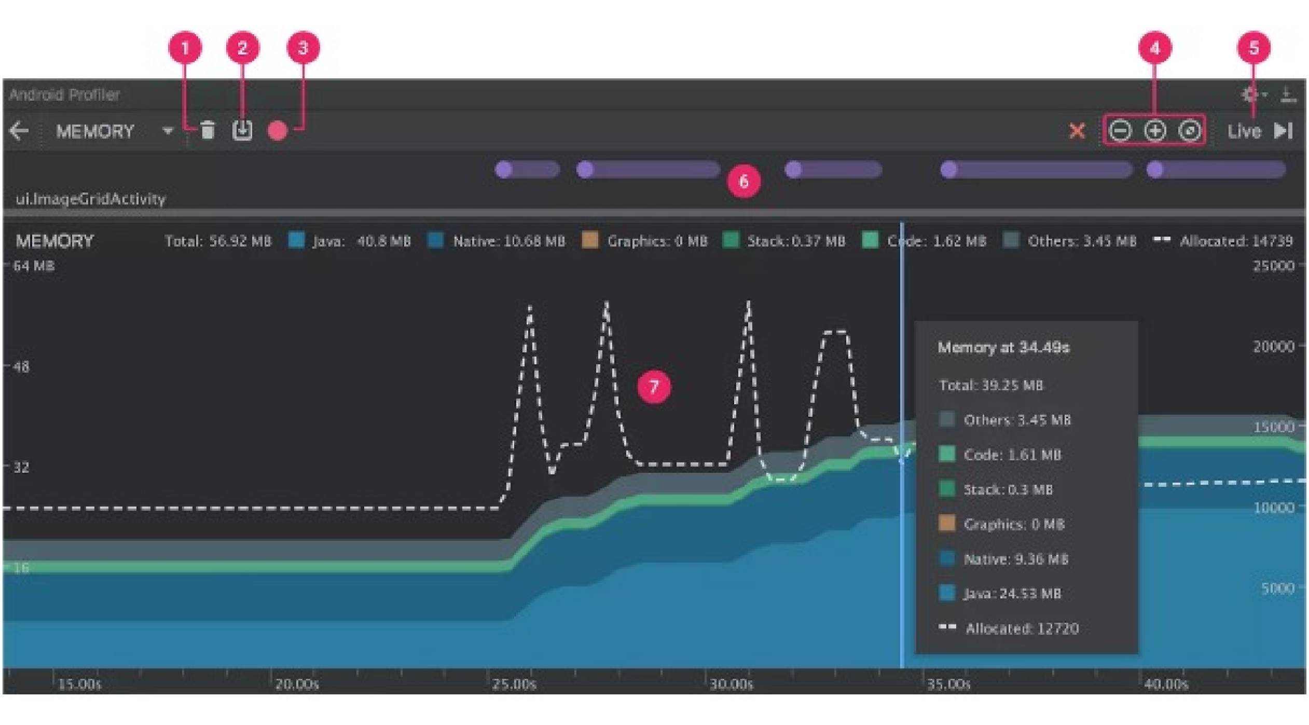
如果我们连接了多个设备可以在按钮❶的位置选择设备，通过按钮❷的位置选择想要的app进程，工具最底部显示了一个时间轴，其中包含了CPU、内存和网络使用的实时图。该窗口还包括时间轴缩放控制按钮❸，一个跳转到实时更新的按钮❹，以及显示活动状态、用户输入事件和屏幕旋转事件❺的事件时间轴。

我们想要查看对应工具详细的分析工具，只要单击性能数据相对应的图即可。下看看对应工具详细的使用。

2:内存工具

2.1内存工具简述（Memory Profiler）

我们打开Memory Profiler后界面如下图（图片来自官网）



如图所示，内存剖析器的默认视图包括以下内容:

❶:强制执行垃圾收集事件的按钮 ❷:捕获堆转储的按钮 ❸:一个记录内存分配的按钮,当连接到运行Android7.1或更低的设备时，该按钮才会出现 ❹:放大/退出时间线按钮 ❺:可以跳转到实时内存数据的按钮。 ❻:事件时间轴，显示活动状态、用户输入事件和屏幕旋转事件。 ❼:内存使用时间线，包括以下内容: 一个堆叠图，显示每个内存类别的内存大小，如左侧的y轴和顶部的颜色键。 虚线表示已分配对象的数量，如右侧的y轴所示。 每个垃圾收集事件的图标

2.2:内存计算指标

根据Android系统，你在内存分析器的顶部看到的数字(如下图)基于你的应用所提交的所有私有内存页面。此计数不包括与系统或其他应用程序共享的页。

Total: 126.21MB ▇ Java: 6.3MB ▇ Native: 71.48MB ▇ Graphics: 37.95MB ▇ Stack: 0.72MB ▇ Code: 7.54MB ▇ Others: 2.23MB ▪▪ Allocated: 106028

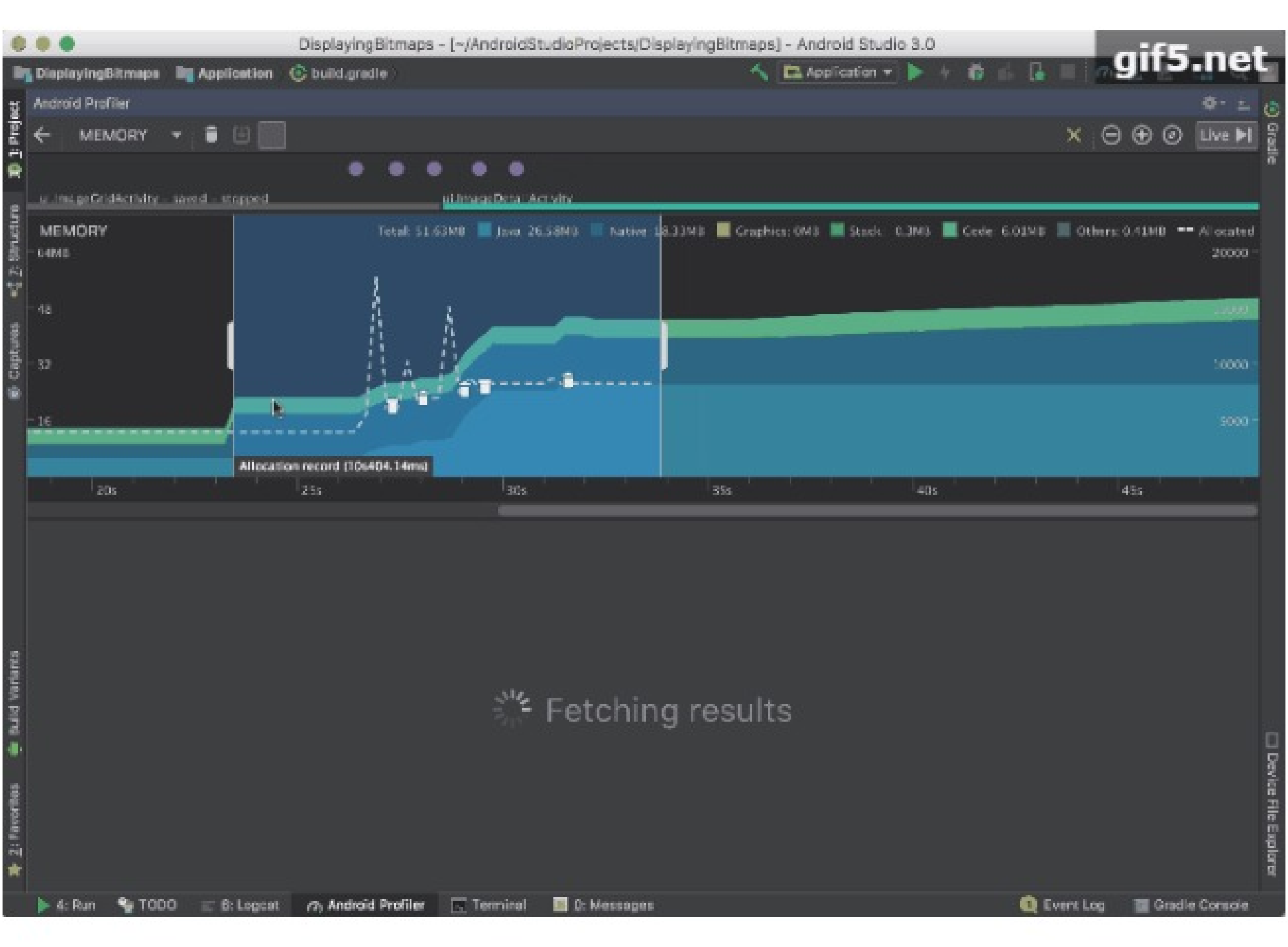
内存类别如下:

- Java:从Java或Kotlin代码中分配的对象的内存
- Native:从C或c++代码中分配的对象的内存，即使你没有在app中使用c++，你可能会看到一些本地内存，因为Android框架使用Native内存来处理各种任务，比如处理图像资产和其他图形——即使你写的代码是Java或Kotlin
- Graphics:用于图形缓冲区队列的内存用于显示屏幕上的像素，包括GL表面、GL纹理等。(注意，这是与CPU共享的内存，而不是专用的GPU内存)
- Stack:在你的应用程序中，Native和Java栈使用的内存。这通常与你的应用程序运行的线程数有关
- Code:您的应用程序用于代码和资源的内存，如dex字节码，优化或编译的dex代码。所以库和字体
- Other:应用程序使用的内存，系统不确定如何分类
- Allocated:应用程序分配的Java/Kotlin对象的数量。这并不计算用C或c++分配的对象

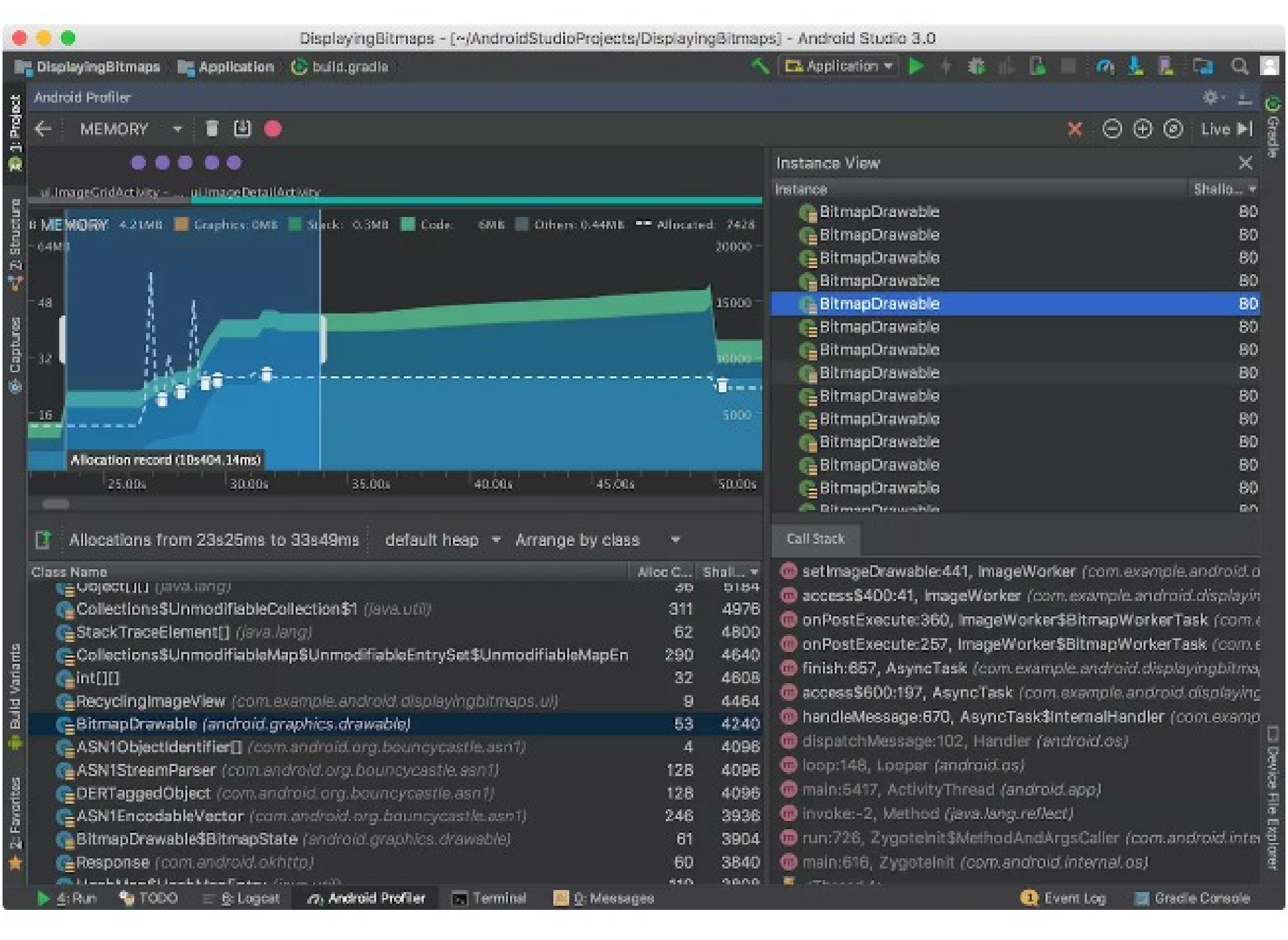
注意:当前应用程序中，native内存统计值可能会偏大，因为分析工具的一部内存也被算进去了，多达10MB的内存被添加到~100k对象，在未来版本的工具中，这些数字将从您的数据中过滤出来。

2.3:查看内存分配情况

单击工具栏记录内存分配在内存分析器 我们想要查看某一时刻程序的内存分配情况也很简单，如下图：



最后的界面如下图



Allocations from 23s25ms to 33s49ms　default heap　Arrange by class

Class Name	Alloc C...	Shall...
Collections$UnmodifiableCollection$1 (java.util)	311	4976
StackTraceElement[] (java.lang)	62	4800
Collections$UnmodifiableMap$UnmodifiableEntrySet$UnmodifiableMapEn	290	4640
int[][]	32	4608
RecyclingImageView (com.example.android.displayingbitmaps.ui)	9	4464
BitmapDrawable (android.graphics.drawable)	53	4240
ASN1ObjectIdentifier[] (com.android.org.bouncycastle.asn1)	4	4096
ASN1StreamParser (com.android.org.bouncycastle.asn1)	128	4096
DERTaggedObject (com.android.org.bouncycastle.asn1)	128	4096
ASN1EncodableVector (com.android.org.bouncycastle.asn1)	246	3936
BitmapDrawable$BitmapState (android.graphics.drawable)	61	3904
Response (com.android.okhttp)	60	3840

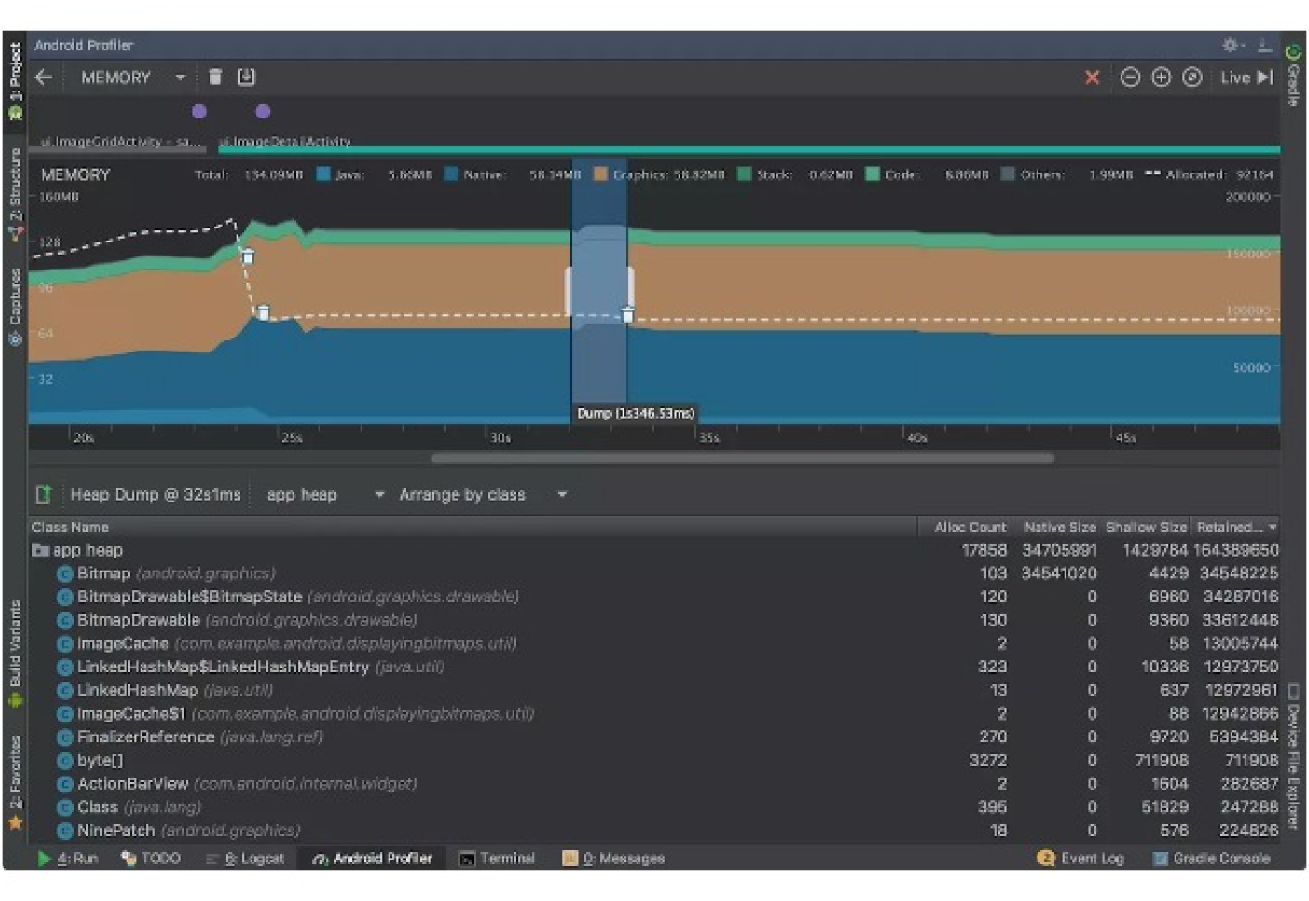
```
setImageDrawable:441, ImageWorker (com.example.android.d
access$400:41, ImageWorker (com.example.android.displayin
onPostExecute:360, ImageWorker$BitmapWorkerTask (com.e
onPostExecute:257, ImageWorker$BitmapWorkerTask (com.e
finish:657, AsyncTask (com.example.android.displayingbitma
access$600:197, AsyncTask (com.example.android.displaying
handleMessage:670, AsyncTask$InternalHandler (com.examp
dispatchMessage:102, Handler (android.os)
loop:148, Looper (android.os)
main:5417, ActivityThread (android.app)
invoke:-2, Method (java.lang.reflect)
run:726, ZygoteInit$MethodAndArgsCaller (com.android.inte
main:616, ZygoteInit (com.android.internal.os)
```

2.4 捕获heap dump

heap dump显示在你捕获heap dump时应用程序中的哪些对象正在使用内存,特别是在扩展的用户会话之后，heap dump可以通过显示仍在内存中的对象来帮助识别内存泄漏。一旦捕获heap dump，可以查看以下内容：

- 应用分配了哪些类型的对象，以及每个对象的数量
- 每个对象使用多少内存。
- 每个对象的引用都被保存在你的代码中。
- 调用堆栈分配对象的位置,（当您在录制分配时捕获heap dump 时，调用堆栈当前仅在Android 7.1中使用堆转储时才可用。

要捕获heap dump，在Memory Profiler工具栏中单击Dump Java heap按钮即可 在转储堆时，Java内存量可能会暂时增加，因为堆转储发生在和你的应用程序相同的进程，并需要一些内存来收集数据，heap dump出现在内存时间线下方，显示了堆中的所有类型，如图下图所示。



Heap Dump @ 32s1ms　app heap　Arrange by class

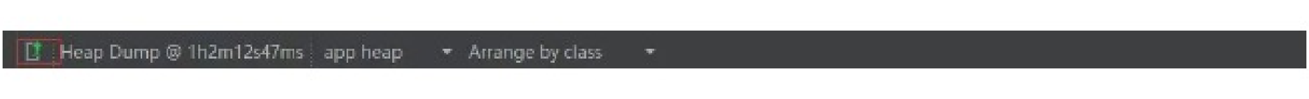
Class Name	Alloc Count	Native Size	Shallow Size	Retained...
app heap	17858	34705991	1429784	164389650
Bitmap (android.graphics)	103	34541020	4429	34548225
BitmapDrawable$BitmapState (android.graphics.drawable)	120	0	6960	34287016
BitmapDrawable (android.graphics.drawable)	130	0	9360	33612448
ImageCache (com.example.android.displayingbitmaps.util)	2	0	58	13005744
LinkedHashMap$LinkedHashMapEntry (java.util)	323	0	10336	12973750
LinkedHashMap (java.util)	13	0	637	12972961
ImageCache$1 (com.example.android.displayingbitmaps.util)	2	0	88	12942866
FinalizerReference (java.lang.ref)	270	0	9720	5394384
byte[]	3272	0	711908	711908
ActionBarView (com.android.internal.widget)	2	0	1604	282687
Class (java.lang)	395	0	51829	247288
NinePatch (android.graphics)	18	0	576	224826

在图中可以看到Class Name列表，，在列表中可以看到以下信息：

- Alloc Count：堆中的分配数量。
- Native Size：此对象类型使用的Native内存总量（以字节为单位），此列仅适用于Android 7.0及更高版本。 你会在这里看到一些在Java中分配的对象的内存，因为Android为一些框架类（比如Bitmap）使用本地内存。
- Shallow Size：此对象类型使用的Java内存总量（以字节为单位）
- Retained Size：由于此类的所有实例而保留的内存总大小（以字节为单位）在class 列表顶部，可以使用左侧的下拉列表在下列堆转储之间切换
- Default heap：当系统没有指定堆时。
- App heap：你的应用程序分配内存的主要堆。
- Image heap：系统引导映像，包含在引导期间预加载的类，这里的分配保证不会移动或消失，
- Zygote heap：Android系统中的应用程序进程分支的写入时复制堆。

2.5 将 heap dump 保存为 HPROF

如果你想保存 heap dump为日后查看，导出heap dump到一个HPROF文件的话，如需要点击 Export capture to file按钮，如下图



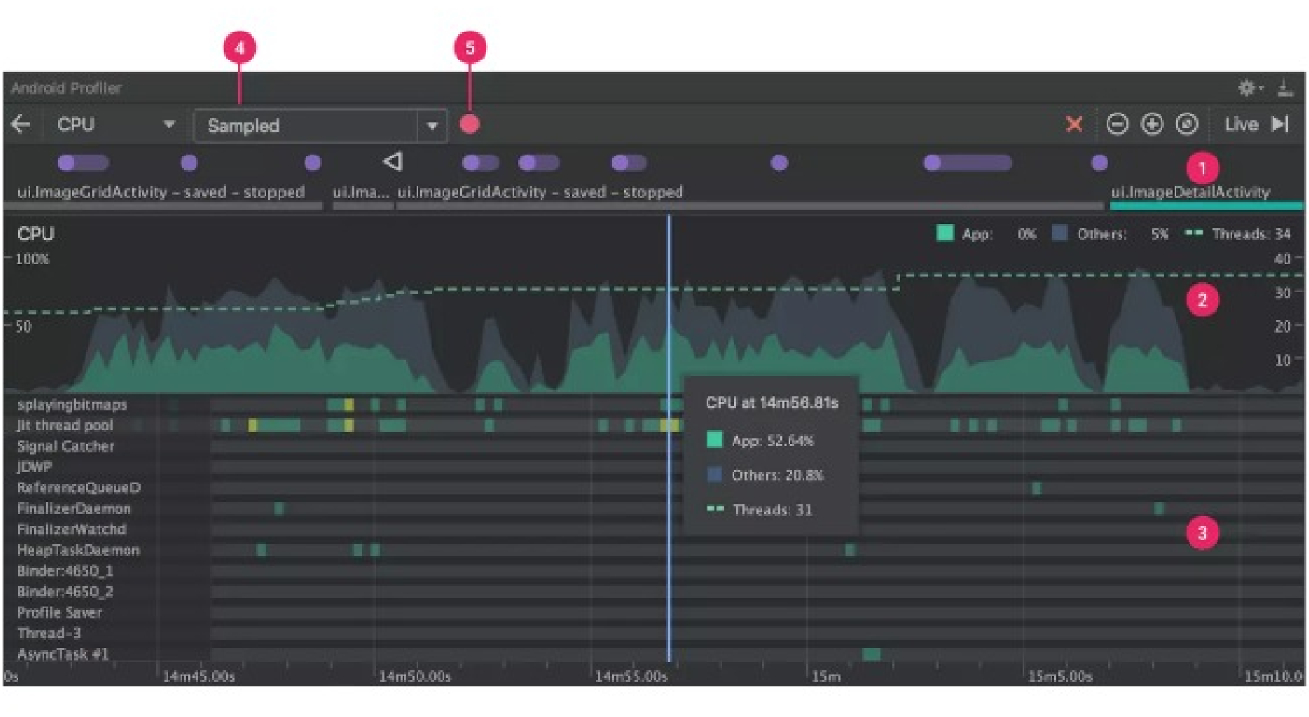
Heap Dump @ 1h2m12s47ms　app heap　Arrange by class

如果你需要从Android HPROF文件格式转换为Java SE HPROF格式，可以使用hprof-conv工具进行转化，它的位置在 android_sdk/platform-tools/目录下，运行hprof-conv命令

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

3:CPU分析工具(CPU Profiler)

当你打开CPU分析器,它会立即开始显示应用程序的CPU使用率和线程的活动，如下图：



CPU Profiler的默认视图包含以下内容：

❶：Event timeline：显示应用在他们的生命周期中不同状态间转换时的活动，并指示用户与设备的交互，包括屏幕旋转事件 ❷：CPU timeline：显示应用程序的实时CPU使用情况（占可用CPU总时间的百分比）以及应用程序正在使用的线程总数。时间表还显示了其他进程（如系统进程或其他应用程序）的CPU使用情况，因此您可以将其与应用程序的使用情况进行比较。可以通过沿着时间轴的横轴移动鼠标来检查历史CPU使用率数据。 ❸：Thread activity timeline：列出属于你的应用程序进程的每个线程，并使用下面列出的颜色在时间线上指示其活动。记录方法跟踪之后，可以从此时间线中选择一个线程，以在跟踪窗格中检查其数据。

- 绿色：线程处于活动状态或准备好使用CPU。也就是说，它处于"运行"或"可运行"状态。
- 黄色：线程处于活动状态，但它正在等待I / O操作（例如磁盘或网络I / O），然后才能完成工作。
- 灰色：线程正在休眠，不占用任何CPU时间。当线程需要访问尚不可用的资源时，有时会发生这种情况。线程进入自愿睡眠，或者内核使线程进入休眠状态，直到需要的资源变为可用。

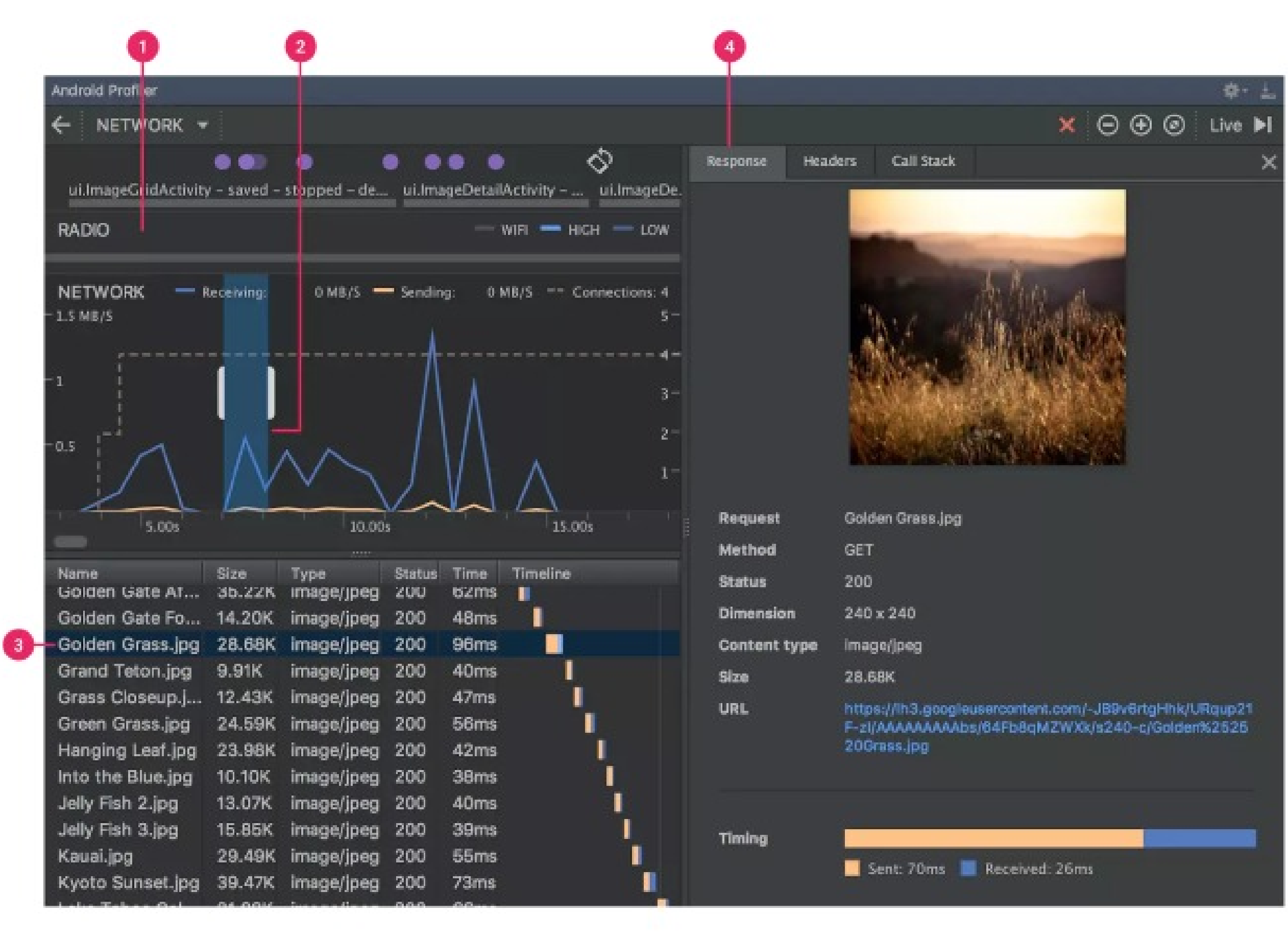
❹:Recording configurations：允许您选择以下选项之一来确定探查器如何记录方法跟踪。

- Sampled（采样）：一个默认配置，可以在应用程序执行期间频繁地捕获应用程序的调用堆栈。分析器比较捕获的数据集合以获取关于应用代码执行的时间和资源使用信息。基于抽样的跟踪的一个固有问题是，如果您的应用程序在捕获调用堆栈之后输入方法，并在下次捕获之前退出该方法，则该方法调用不会被分析器记录。如果您对如此短的生命周期跟踪方法感兴趣，则应使用检测跟踪。
- Instrumented：默认配置，在运行时测试您的应用程序，以在每个方法调用的开始和结束时记录一个时间戳。收集时间戳并进行比较，以生成方法跟踪数据，包括定时信息和CPU使用情况。请注意，与每种方法相关的开销都会影响运行时性能，并可能影响分析数据 - 对于生命周期相对较短的方法来说，这一点更为明显。此外，如果您的应用程序在很短的时间内执行大量方法，分析器可能会快速超出其文件大小限制，并且无法记录任何进一步的跟踪数据。
- Edit configurations：允许您更改上述采样和检测记录配置的某些默认设置，并将其保存为自定义配置。

❺:Record button：开始和停止录制方法跟踪按钮

4:网络分析工具(Network Profiler)

网络分析工具比较简单，界面如下图：



Name	Size	Type	Status	Time
Golden Gate Fo...	14.20K	image/jpeg	200	48ms
Golden Grass.jpg	28.68K	image/jpeg	200	96ms
Grand Teton.jpg	9.91K	image/jpeg	200	40ms
Grass Closeup.j...	12.43K	image/jpeg	200	47ms
Green Grass.jpg	24.59K	image/jpeg	200	56ms
Hanging Leaf.jpg	23.98K	image/jpeg	200	42ms
Into the Blue.jpg	10.10K	image/jpeg	200	38ms
Jelly Fish 2.jpg	13.07K	image/jpeg	200	40ms
Jelly Fish 3.jpg	15.85K	image/jpeg	200	39ms
Kauai.jpg	29.49K	image/jpeg	200	55ms
Kyoto Sunset.jpg	39.47K	image/jpeg	200	73ms

Request	Golden Grass.jpg
Method	GET
Status	200
Dimension	240 x 240
Content type	image/jpeg
Size	28.68K
URL	https://lh3.googleusercontent.com/-JB9v6rtgHhk/URqup21F-zI/AAAAAAAAAbs/64Fb8qMZWXk/s240-c/Golden%252520Grass.jpg

窗口的顶部的❶处,可以看见wifi无线信号的强弱，在时间线上可以在❷处点击和拖动一部分的时间线来检测流量，然后在窗口❸中会显示所选时间段内收发的文件，包括文件名，大小，类型，状态和花费时间，你可以对窗口❸的列表根据列来进行排序。还可以查看所选时间段的详细拆分，拆分的timeline可以显示文件是什么时候收发的，点击窗口3的其中一个文件，可以在窗口❹中查看文件的详细信息。通过切换窗口❹上方标签可以查看response data（响应数据）, header information, or the call stack（调用栈）。