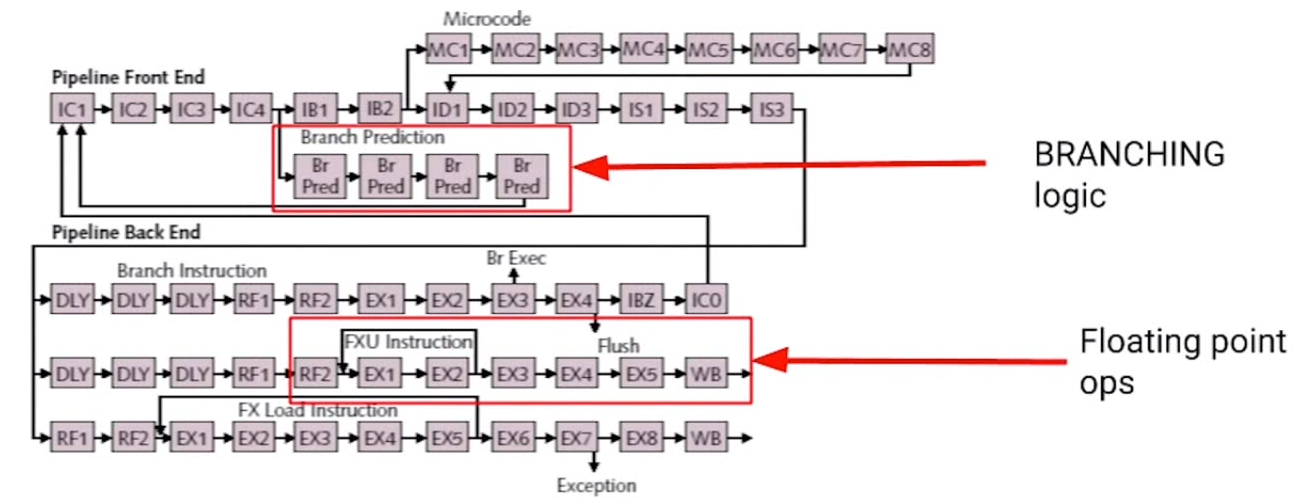
**低性能函数**

让我们首先从一个非常熟悉的问题开始，也就是函数执行过慢的问题，这是性能概念方面的初级基础计算机科学知识。有时候，当你写完代码之后，你发现它的运行速度，比你预想的要慢，这种情况经常会无缘无故地出现，你专注于采用某种方法编写代码，来解决特定的问题，但是很快你发现代码的运行时间过长，远远超过你的预期。

代码运行时间过长的问题，在很大程度上可以归咎于编程语言，当然还有相关硬件执行代码的方式。例如，在一些老旧硬件上，执行浮点比较算法分支语句所花的时间，几乎是整数或布尔数值的四倍。其原因在于芯片架构，负责浮点计算的CPU部分在分支逻辑阶段之后开始工作。这意味着任何浮点比较都需要等待，直到循环管道结束，从而拖延其他运算，直到分支逻辑最终执行完成，但是请不要感到害怕。现代硬件通常不需要处理这种细微问题，但是这也说明了一个很重要的观点，即，你编写代码的方式会影响性能，具体视硬件上执行的编程语言而有所不同。这个问题甚至可以追溯到芯片架构。我想要说的是，为了优化你的代码，你需要理解系统如何运行代码。

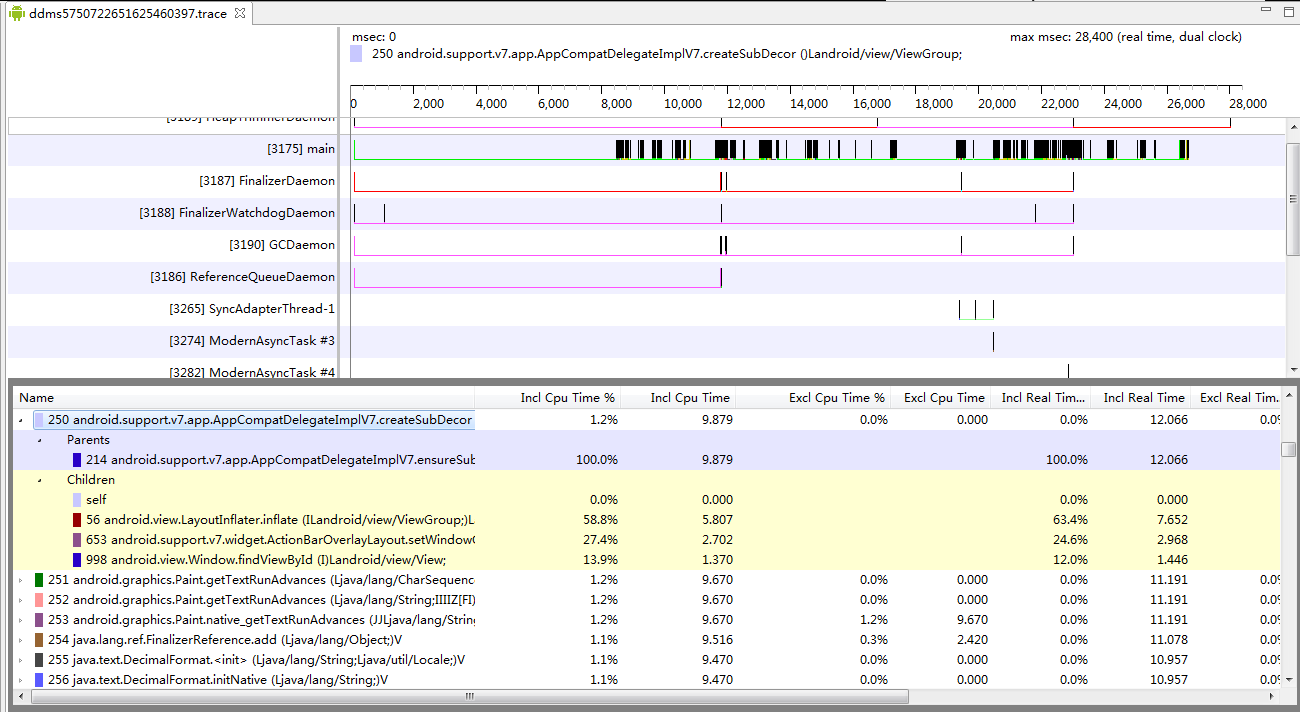


**缓慢的函数执行通常是由于两方面的问题造成的，第一种是执行速度很慢的函数，这种函数很容易被发现。**你的某些函数所花费的时间，超过你的预期2倍、10倍，甚至50倍，这种问题容易解决。只要找到那些运行很慢的函数，查看代码找到问题所在，然后想办法解决就可以了。**更难发现的是第二种类型，想方设法也难以发现，尤其是当你有数以千计的函数时，每个函数所用的时间都额外增加一毫秒，从而导致整个程序执行速度变慢数百毫秒，这种类型的问题很难跟踪，而且更难以解决，因为通常你需要分析，每段执行代码才能发现这些小问题，这最终会影响你的产品发布，进而影响公司业绩。**

要解决这些细小的问题，主要的方法是剖析（profiling），通过时间线分析方法，找到执行速度缓慢的代码部分，或者时间明显长于其它代码的部分，然后进行一些细小变更。然后再次进行时间线分析，找到执行很慢的函数之后，你需要对这些函数的代码行进行时间线分析，以及调用这些函数的所有其他函数。这项工作确实相当繁琐，除非你是这个领域的专家，但是不要害怕，Android SDK有一些很不错的工具，帮助你找到这些有问题的代码部分，让你能够立即解决它们，让我们来了解它们。

**Traceview工具**

我将演示如何跟踪你的应用程序中的一些计算相关的性能问题。在这个演示中，我们将使用工具来跟踪Sunshine应用程序，这个工具是TraceView。我们载入它，打开DDMS视图选择我们要分析的应用，请你注意一下，工具栏上的一些图标，尤其是这个图标。看上去像是三面箭头，上面有红色的圆点，如果按这些按钮，会出现一些提示，说将开始进行方法分析。这是TraceView的启动方法，我们点击它。将出现一个弹出窗口，提示有两种方法来分析你的应用程序。你可以记录每个方法的输入和输出，他们对资源的要求很高，或者，你也利用示例进行一些分析。其含义是，默认情况下分析程序，将会每1000毫秒侦测一次你的应用程序，以发现和记录实际上在运行的功能，现在，让我们来使用这些默认设置。我点击一下OK，既然分析程序已经在继续，我们就与你的应用程序进行交互，看能否记录一些动作。因此跳转到这里与Sunshine应用程序进行交互，好极了，山景地区的天气。不幸的是，周末的天气却不太好，将会下雨。我们看看海岸区域，我们在南加州的朋友还好吗？奇怪，他们已经进入冬季，这在圣地亚哥是很少见的。我们回到Android Device Monitor，我们想要停止分析。我们还是应该点击这个图标，启动时也是点击它。在最上方有一个黑色图标或者黑色方块，点击它可以停止分析。现在，可能需要一点时间来载入跟踪记录，将会显示在窗口上方，选项卡的这个位置。请记住，实际所用的时间可能更长一点，具体取决于实际记录的内容。



我们来看跟踪视图，跟踪视图有两个主要组成部分。上方窗格的名称是timeline面板，下方窗格内有很多的信息，称为profile面板。这个时间线能够很好的显示代码的执行情况，这里显示的每一行，实际上对应于一个线程。显示的每一个颜色，对应于一个正在运行的特定方法。例如，我们可以看到，主线程的所有活动，我们可以看到方法启动和停止时间点，更有用的是放大这里，找到特定的方法，了解他们是如何执行的。它们会以这种U型模式显示出来。这里的条形表示，方法的启动时间。右侧的条形表示，方法的停止时间。条形的宽度表示方法执行所用的时间。现在，我们选择一个特定的方法，我们跳转到跟踪视图窗口的底部，这里，我们看到一些分析数据显示出来。我们可以看到哪些方法调用了我们选定的方法。在这里，用他们的父级方法显示为蓝色，我们还可以看到一些信息，显示在这个方法内调用了哪些方法。也就是说，我们调用了一个发送输入事件方法。在选择之后，会显示一个本地条柱。对于我们选择的所有这些方法，都有大量的附加统计信息。例如，可以看到独占CPU时间，我们可以使用这些信息，找到具体方法的特定问题。非独占CPU时间是特定方法在其内部调用的所有方法所用的时间。这可以帮你在信息树内找到，选定方法的特定问题。另一个十分有用的信息是，方法被调用了多少次，或者递归调用本身多少次。如果我们滚动到右侧，我们可以找到这些信息，这里有一个列名为“calls and recursion”，此列显示方法被调用多少次，或者它被递归调用多少次。在这个分析面板中，有大量的附加信息。另外，不要忘记这个搜索框，它可以帮助查找你所关心的功能。

* 每个方法前面都有一个数字，按照Incl CPU Time时间的排序序号
* 展开一个方法后可以看到有两部分

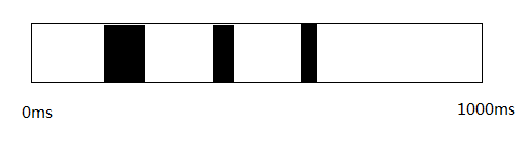
Parent表示调用这个方法的方法，可以叫做父方法

Children表示这个方法中调用的其他方法，可以叫做子方法

* Profile面板中各列作用说明

|  |  |
| --- | --- |
| 列名 | 作用 |
| Name | 该进程运行过程中所调用的函数名 |
| Incl Cpu Time | 函数占用的CPU时间，包含内部调用其它函数的CPU时间 |
| Excl Cpu Time | 函数占用的CPU时间，但不包含内部调用其它函数所占用的CPU时间 |
| Incl Real Time | 函数运行的真实时间（以毫秒为单位），内含调用其它函数所占用的真实时间 |
| Excl Real Time | 函数运行的真实时间（以毫秒为单位），不包含调用其它函数所占用的真实时间 |
| Calls+Recur Calls/Total | 函数被调用次数以及递归调用占总调用次数的百分比 |
| Cpu Time/Call | 函数调用CPU时间与调用次数的比（该函数平均执行时间） |
| Real Time/Call | 同CPU Time/Call类似，只不过统计单位换成了真实时间 |

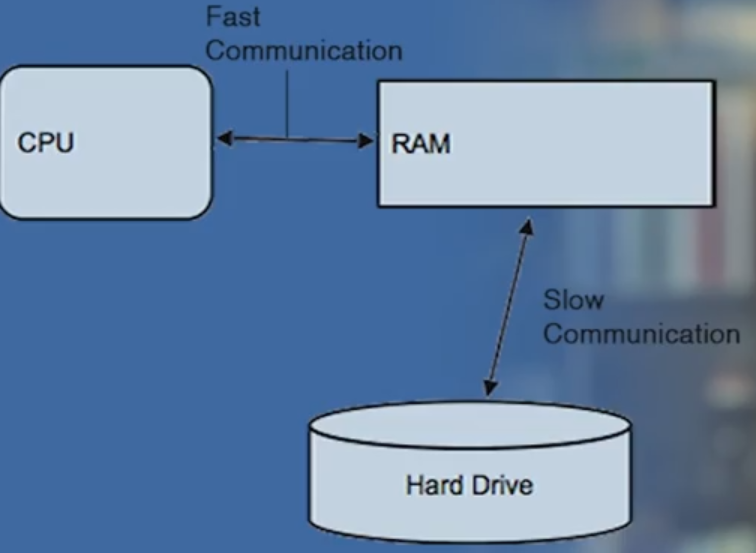
备注：CPU time就是CPU实际花了多少时间在运行函数，CPU在多进程环境下不会把所有时间用在一个进程上的



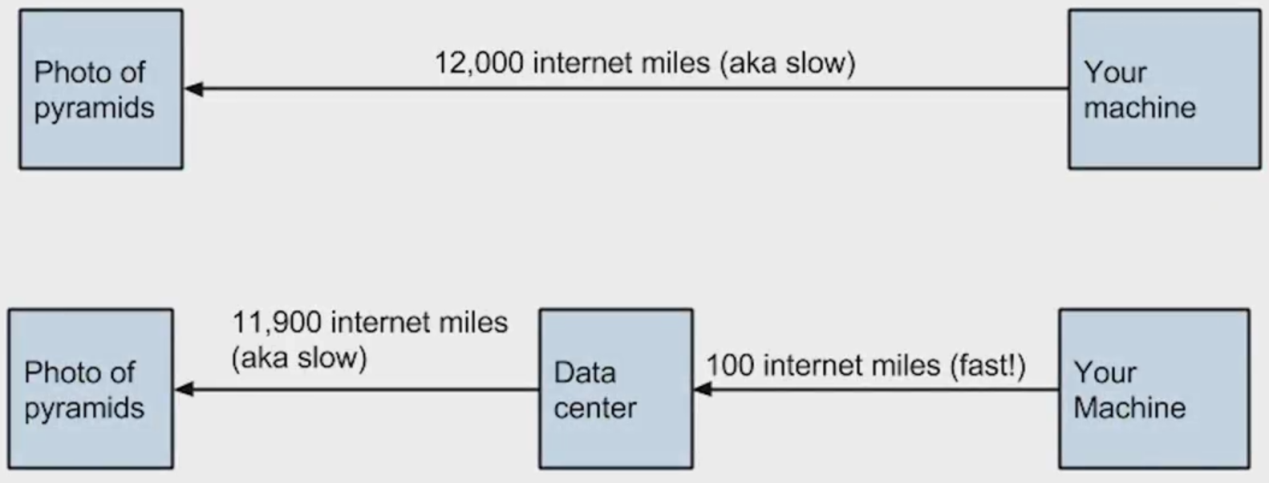
**批处理与缓存技术**

我想向你们介绍我最喜欢的两个性能技术，批处理（batching）和缓存（caching）。前面我们已经说过一些函数和运算，需要非常大的资源开销，这也会影响计算性能。例如，在执行之前把数据载入特定的内存区域，或者，在搜索之前对数值集进行排序，在执行多次之后，而且次数确实是个很大的数字，资源开销将会严重影响应用程序的性能。批处理是可以帮助解决这种性能问题，它消除每个运算的独立执行开销，好像是所有人都开一辆车，而不是每个都开一辆，从而节省汽油。这种情况最常见于在执行运算之前，你需要准备数据。例如，在查找集合中的值时，最有效的方法是进行排序，然后进行二分法搜索等等。有一点必须弄清楚，这并不是最有效的方法。这只是举一个例子而已，最简单的方法是写一个函数，提供一个集合和一个值，对集合进行排序，然后查看值是否存在于集合之中。对于某些性能要求来说，这样做是可以的。但是，如果有10000个值，而且总共需要数百万租数据，排序所花费的时间，将会增加很多倍，答案很明确。对这组数据一次性完成排序，然后查找所有10000个值，并不是明智的方法。这时就需要用到批处理，我们找到重复的运算，找到之后，进行批处理。

缓存是与批处理相似的概念，这也是目前为止，你能理解的最重要的性能技术。这项技术全面地推动现代计算机科学的发展，以计算机为例，内存的作用是用来存储信息，让CPU能够更快的访问数据，其速度远快于访问硬盘数据。



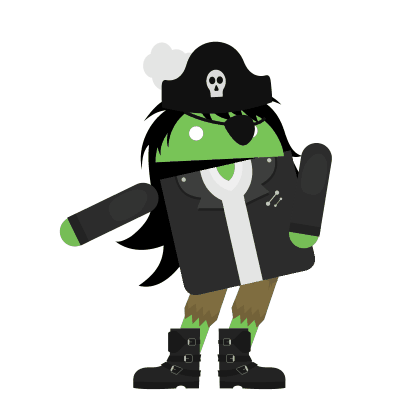
或者以网络为例，世界各地存在大型服务器仓库，它们被称为数据中心，它们的作用是存储火缓冲被频繁访问的内容。这样，你的计算机就不必每次都访问远在12000英里之外的服务器。你在埃及的朋友可能在这个服务器上发布了一张图片，当然，如果你在埃及，这样的缓冲服务器可能就没有什么意义，但是你已经明白其中的道理。以代码为例，最常见的缓存优化通常涉及多次计算，但是如果始终相同的数据，例如，在循环计算中，你计算一个4x4数列的导数，结算始终是相同的，每次重新计算循环迭代，实际是在浪费计算机资源。相反，在循环流程的外部存储导数的结果，并让你的内部循环语句引用缓存结果，可以极大地提升效率。我之所以喜爱缓存和批处理，是因为他们能够改善所有你能够想到的性能问题，包括我们在本课程中提到的问题，这是两个非常有效的技术。如果你想成为一名性能专家，你最好能够熟练掌握这两项强大的技术。



**使用批处理与缓存解决问题**

目标：在这个例子中，您将使用Traceview工具来查找并确定哪些是阻碍应用程序性能问题的代码。然后，您将应用批处理或缓存来优化性能不佳的代码。

1）观察：首先，让我们看看我们能够用怎样一个给定的应用功能来观察一个问题。在设备上启动compute应用，然后启动Batching/CachingActivity。你会看到一个正在舞蹈的海盗动画。

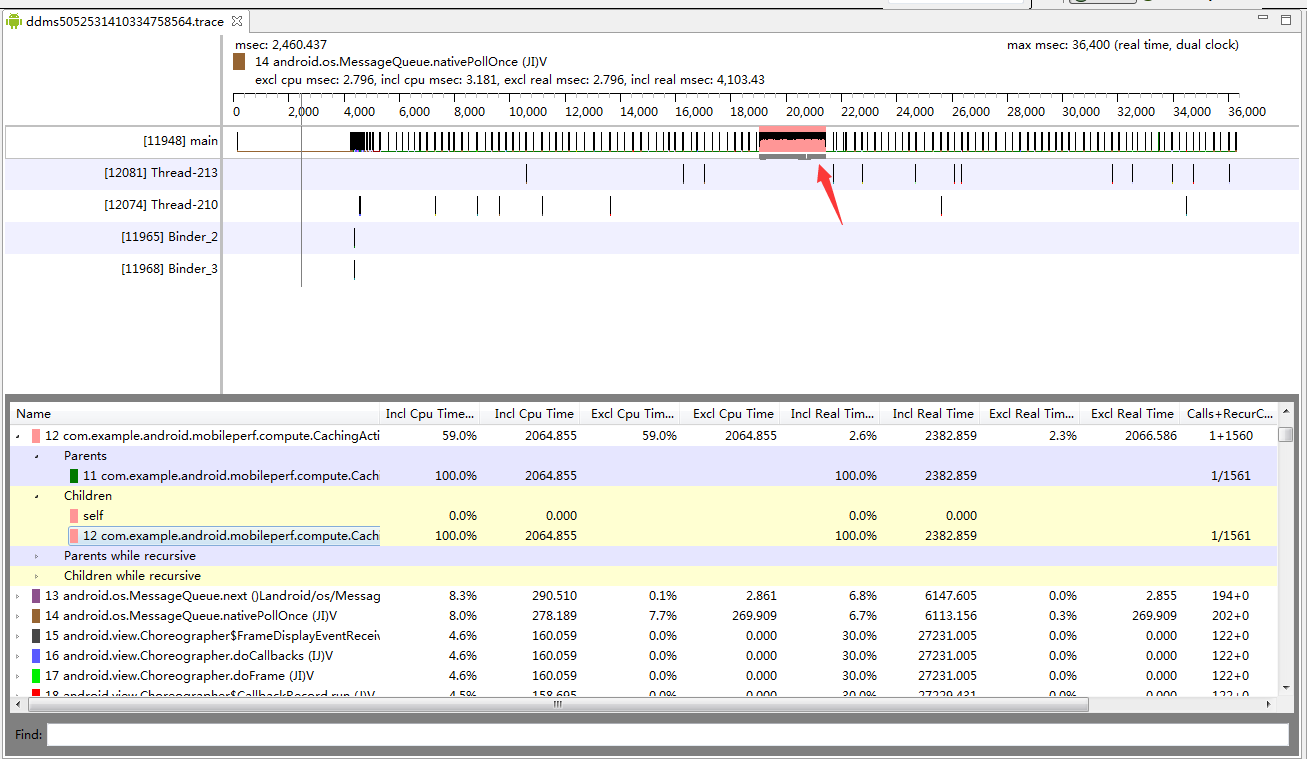


上面有一个写着“计算一些Fibonacci数”的按钮，点击它。你观察到了什么，为什么会这样？不难看出，我们的海盗朋友停止了摇摆。肯定是按下按钮之后才导致了这个现象，现在，你已经观察到了一个问题，我们来分析下。

2）Profile：您刚刚了解了Traceview，让我们把它运用起来。点击按钮，观察程序的运行轨迹。

3）分析：根据运行轨迹分析，找出导致问题的方法是谁？

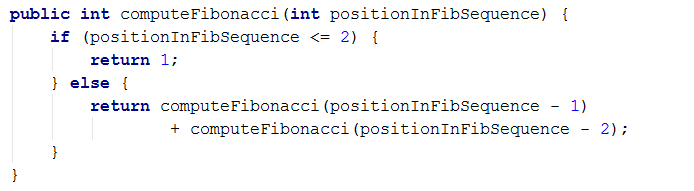
在这里，我们要找的答案，实际上是computeFibonacci方法。让我们看看，我是怎么样找到它的。在下面的活动上运行TraceView，当我按下这个computeFibonacci函数，将会剖析这个函数。这是Traceview的输出，这是运行Traceview时看到的输出，你应该看到一些类似的内容，请注意这个大的粉色区域，这很糟，基本上，这表示有些函数在我们的主线程上占用大量的CPU时间。如果你按照独占CPU时间排序或者将鼠标悬停在这个粉色区域。你会发现computeFibonacci方法，它来自于我们的缓存活动，是占用CPU资源最多的函数，我们需要解决这个问题。



**解决斐波那契**

有哪些好方法可以解决这个问题呢？请选择所有正确答案。我们不应该为主线程增加任何不必要的额外的工作，我们应该让线程仅处理用户输入和屏幕绘制.我们看看是否能够优化函数，使用共享技术提高运行速度和减少资源开销，让我们缓存这些中间值。

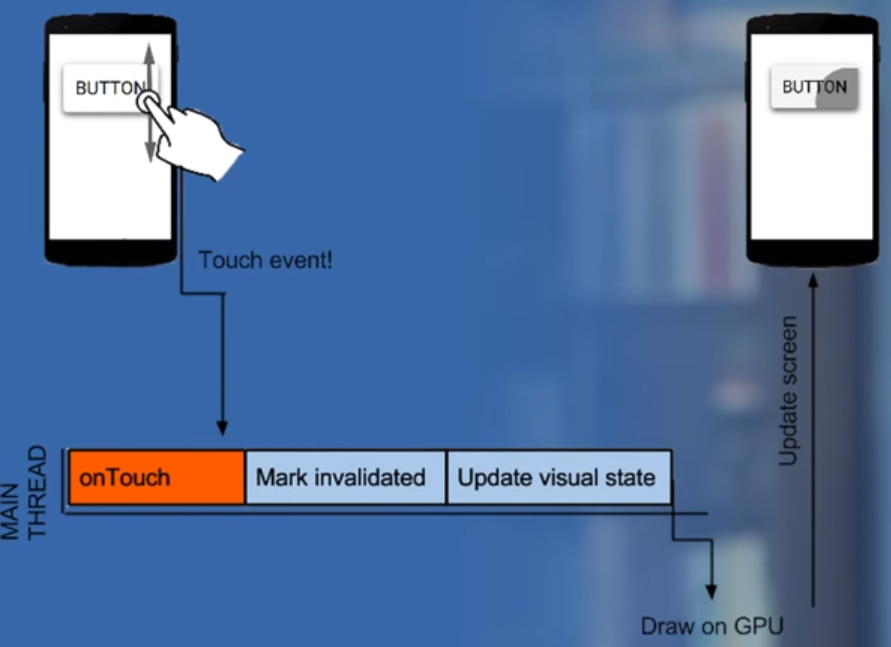
让我们来看看代码，看看发生了什么。我们在主线程收到一个onClick（）事件计算斐波那契。看看方法：

****

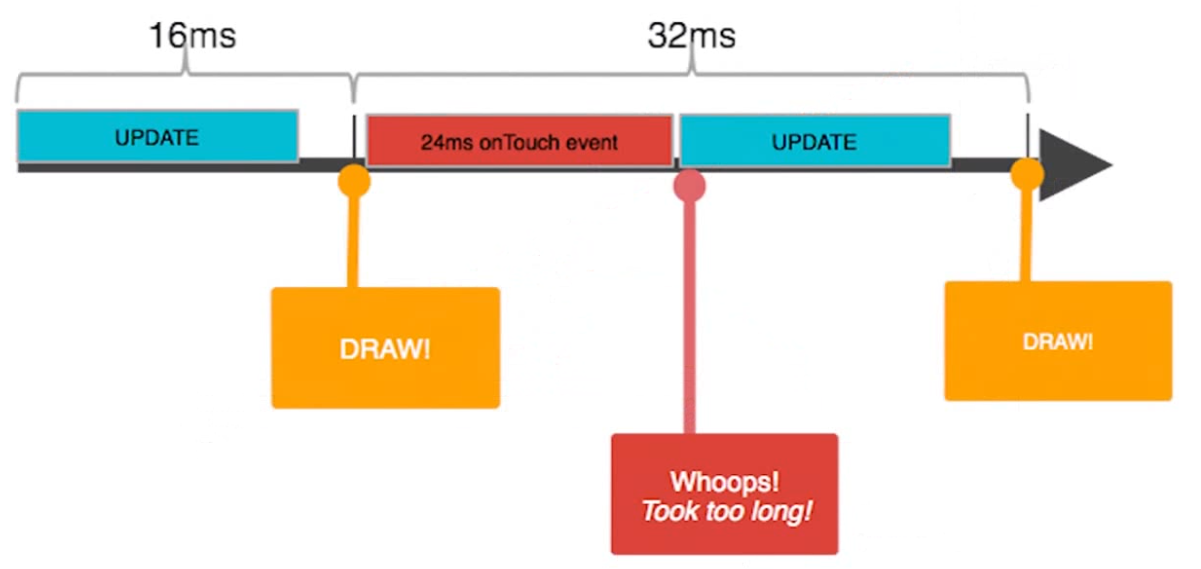
如果你熟悉算法理论，你可以看到，斐波那契数递归执行，从代码运行的角度来看，它是非常耗时的。解决这个问题的方法之一是通过计算和缓存中间结果。

**主线程阻塞**

为了确保应用程序的高性能，每项功能都应该尽可能高效地运行。但是这些功能的执行时间以及它们在代码中所处的位置也很重要，当你首次启动一个Android应用程序时，朱执行线程就已经创建了，主线程非常重要，因为它负责运行你的代码，并在合适的视图位置发送事件和执行绘图功能。这些前面我们已经讲过，基本上来说，主线程是应用程序所在的线程，有时候，主线程也称为UI线程。例如，如果你触摸屏幕上的按钮，UI线程将会发送一个触摸事件给视图，视图将按钮状态设定为已按下设定，然后向事件队列发送一个有效请求，然后UI线程处理此请求，并通知按钮将其本身绘制为已按下状态。如果你有任何触摸事件的处理代码块，将会在线程中执行，这些触摸处理所用的时间越长，线程的执行时间就会越长，在绘图功能执行完之前，视图将会更新显示状态，让用户能够看到其状态，这里需要记住的是，输入处理代码与渲染和更新代码，共享这个线程的处理周期时间。



这意味着，在触摸事件处理，网络访问或数据库查询等计算周期时间，UI不会更新绘图，在简单的情况下，渲染周期可能会延误16毫秒左右，而让用户感到延迟。但是，如果你暂停UI线程渲染超过5秒，用户将会看到“应用程序未响应”对话框，并询问用户是否会想要关闭你的应用程序，这样可能导致用户停止使用。那你如何解决这个问题，你要找出不需要在主线程上执行的功能，也就是说，不需要等它们完成之后，才能执行绘图。你应该将这个功能转移到一个单独的独立线程，这个线程不会阻止UI线程。例如，如果你按一下提交按钮，以完成一个订单，然后编写和发送确认邮件，这可以在单独的线程上完成。Android有系列很好用的API，能够简化这些工作。



**使用Traceview工具明确问题所在**

我将要演示如何操作traceview，这个程序，可以识别所有安装程序的帧速率，操作方法如下，安装后，点击show on click handler按键，各位会看到很眼熟的跳舞海盗，然后点选display an image按钮，现在大家会看到海盗不动了，但如果再这样做，海盗又继续跳舞了，同时Android图示也会出现在海盗下方。跟之前一样，只要点击这个按钮，海盗就不动了，下面要怎样操作，想必大家都知道了，就是要利用这些工具，现在演示怎样操作traceview。

在这里，查看下刚才追踪的数据，结果出现冗长的数据，这是什么意思呢？检查数据，找出哪两个方法调用得最频繁。辨别哪些程序，占用了大量的CPU资源？

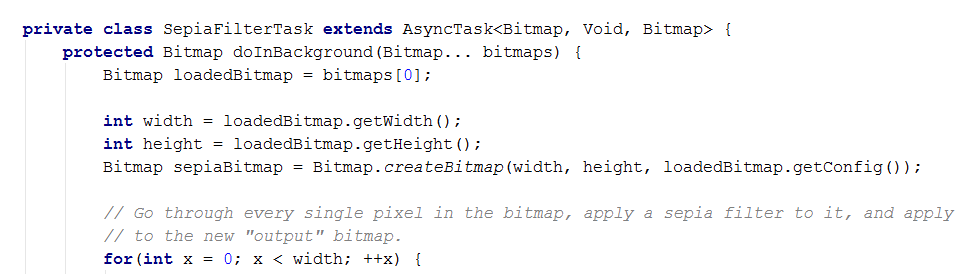
我们来看一下Traceview输出，请注意这个大的活动区域，我们来探讨一些观察到的信息，你会发现，最上方的这个函数，在排序时占用的资源最多。还有其他一些函数，例如这个nativePullOnce,但是它们是系统调用，我们无法控制它们。如果我们更进一步，就会注意到这个nativeSetPixel和这个nativeGetPixel。让我们看看它们来自于哪里，我们展开它，看到了！！在繁忙的UI线程活动的某个位置，setPixel被调用，它来自于我们的应用代码，getPixel也是如此，似乎来自于繁忙的UI线程活动。现在，我们发现setPixel和getPixel，来自于繁忙的UI线程活动，让我们来更深入地探讨。

因此getPixel和setPixel并不是我们编写的代码，在这个代码中哪个父级方法调用getPixel和setPixel？父级方法实际上是sepiaAndDisplayImage，让我们来看Traceview，来了解其含义。我们回到跟踪视图，如果我们展开可收缩菜单，我们可以找到父级调用堆栈，看到setPixel实际上被sepiaAndDislayImage方法调用。它在我们繁忙的UI线程活动之内，让我们看看如何优化。

**异步任务**

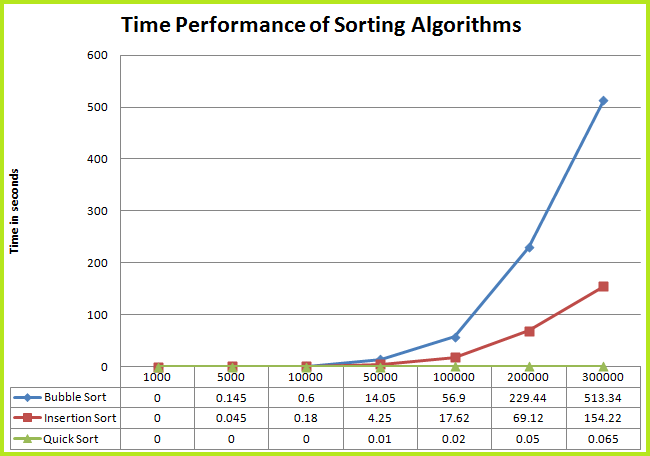
对图像进行编码处理是一个相当大的任务，这也是我们不能轻易地优化掉的任务。特别是那些涉及网络接入，冗长的数据库调用和图像处理，那么一般规则是将它们从主线程移除。

让我们来使用异步任务：

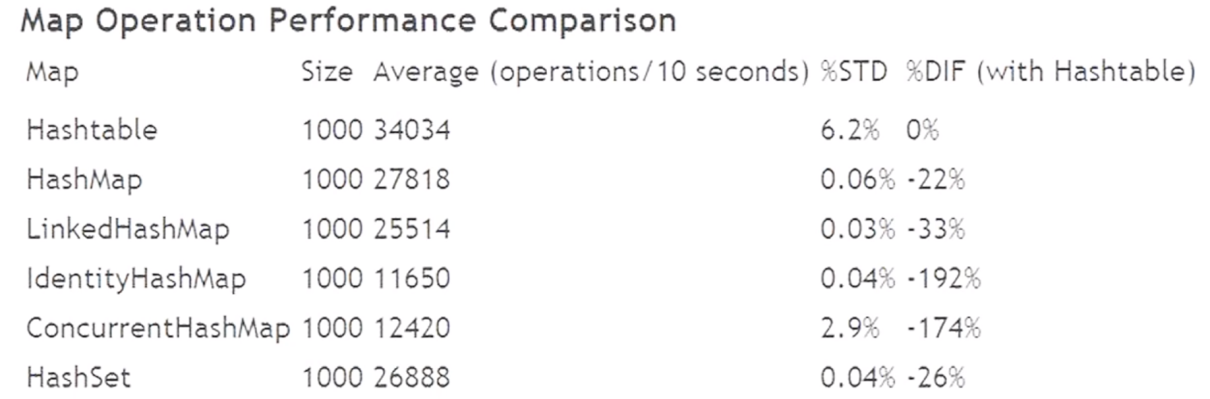


**容器性能**

前面我们讲过，一些类型的硬件可能会造成程序执行速度较慢，还记得那个浮点分支问题吗？对于今天的硬件来说，这已经不是问题。但是有一些问题还是需要引起注意，比如说，你所使用的编程语言的基本元素的效率，以排序等基本算法为例，现在，有很多的排序算法，对于不同的情况，它们各有优劣，例如，当元素数量少于一千或在大型已排序列表中寻找一个对象时，快速排序法通常比起冒泡排序法更快。一般情况下，最好的方法是二分查找算法，但是，当在未排数组中寻找对象时情况变得完全不同，不同于比较每一个对象以查找你想要的值。你可以使用一个哈希函数来立即找到它，这是现代计算机科学和数据结构方面的基本知识。

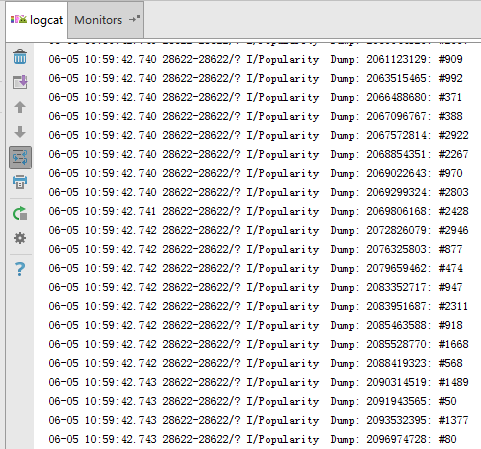


幸运的是，现代编程语言像Java等，为你提供了这些容器和算法，因此你不再需要自己反复地编写Murmur3哈希函数和快速排序算法。但是你需要知道另外一些事情，在我多年的编程生涯中，一个经常会影响项目性能的问题，是由于这些语言提供的容器对象的性能所引起的。这听起来不可思议！Java提供一个矢量类的实现，你可以任意push、pop，添加和取消对象，为了获得这种灵活性，它在内部使用链式列表结构，这种结构具有一系列独特的性能特性，在你操作这种列表时，它的速度超级快，但是，当你在其他位置进行插入或删除时，它会消耗大量的时间。我要说的是，底层系统提供的这些容器并不会考虑，你的程序将会如何实际使用它们，James Sutherland发表了一系列的基准测试报告，他认为，我们需要注意性能与功能之间的一些差异。例如，他发现Hashtable比HashMap大约快22%，具体视你如何使用这些容器而有所不同，我们需要思考的是，你是否曾经分析过你在代码中使用的容器类。你是否坚信，你在代码中使用的容器的实际运行速度绝对是最快的。一个好消息是，你可以使用Android中的MPI来剖析这些容器的性能。

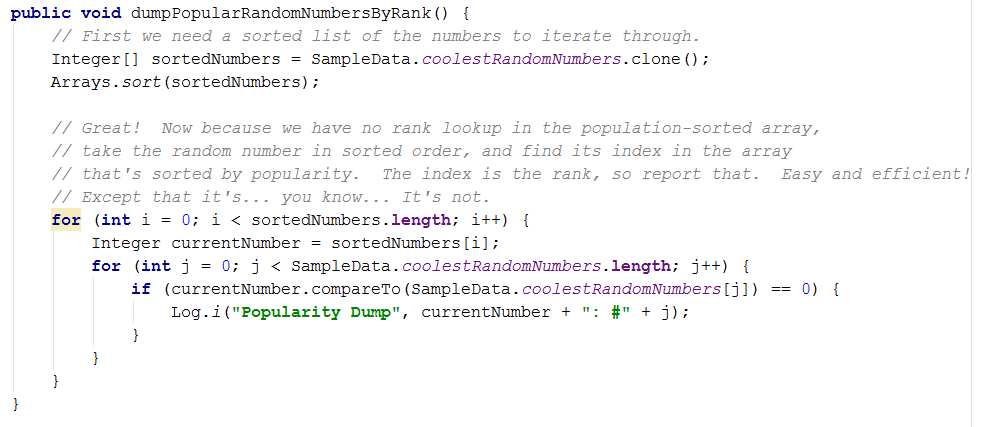


**数据结构**

在这个例子中，你将会看到，创建应用时，容器中不恰当的数据结构所造成的性能问题，为此我们可以使用Android SDK中的工具，来识别不恰当数据结构带来的性能问题。存储与修改应用程序数据代码的性能问题，很容易被开发人员忽悠，所以让我们来探讨这个问题，并从性能角度来解决。对于这个示例，我们将重点关注方法的执行时间，这个方法生成一个数字列表，并按它们的受喜爱程度排名，为了便于演示，当我们按下这个按钮，也就是dump popular numbers to log按钮，将会调用这个方法，与前面的示例相似。这段代码会造成图像短暂停顿从而影响跳舞海盗图像的帧率。如果你看一下logcat，你会发现，它通过tag popularity dump运行，像这样。让我们看看底层发生了什么，并学习如何测量这段代码的运行速度，我们想要仔细弄清楚，当我们点击这个按钮时发生了什么，因此我们使用trace类beginsection和endsection方法来指定开始测量位置和结束位置。首先，让我们找到计算受喜爱程度的代码，我们将使用它计量代码效率和运行时间。



问题：beginSection与endSection应该放在哪个地方？



在这个例子中，你会看到我们是如何选择一个更有效的数据结构，提高了数据访问时间。

问题：通过使用dumpPopularRandomNumbersByRank（），我们粗略招致排序操作的成本，再加上o通过双循环迭代来生成排名列表（N^ 2）成本。

改进注意事项：

总有一个项排序一次性成本（你会根据自己的数据集的大小挑选理想的排序项）。

通过使用一个HashMap中，我们通过获得直查询时间（为O（n））与二次时间为O（n^2）未优化的情况下，阵列的访问节省时间。我们节省的访问时间这一个订单，因为数据已经存储在键值对！

这显著事项当n（或样品尺寸）特别大，这可能是如果你用，比如工作，在世界上所有的职业足球运动员名单，想碰到一些属性显示它们的排名情况。

让我们花点时间确认的改善。来吧，在加回（如果你删除它）调用endSection和beginSection。

然后，运行在优化代码systrace。