第一部分-性能开发

原著：Apple Inc.
翻译：Gavin

性能是软件设计中经常被忽视的一个方面，直到它成为一个严重的问题。如果您要等到开发周期结束时才进行性能调优，那么要实现任何重大的改进可能为时已晚。性能在设计阶段的早期就包含在内，并在整个开发周期中继续改进。

当然，为了设计性能，它有助于理解性能。本章的章节提供了影响性能的因素的背景信息，这些因素如何在OS X和iOS中表现出来，以及如何对这些因素进行监控。

一、性能是什么?

“性能”一词对不同的人可能有不同的含义。因此，在着手改进应用程序性能之前，现在是考虑这个术语的意义的好时机。

许多人把性能和速度等同起来。实际上，如果一个程序在一秒钟内执行一个复杂的操作，您可能会认为这个程序有很好的性能。然而，就速度本身而言，它可能是一种误导性的度量。在复杂的软件系统中，操作的速度不是固定值。如果您在不同的条件下多次执行相同的操作，那么完成该操作所需的时间可能相差很大。这是因为程序只是在本地系统上共享资源的许多进程中的一个，而这些资源的使用(或滥用)会影响到所有其他进程。

以下部分将从两个不同的概念来解释性能:高效资源使用和感知性能。这两个概念对如何设计和实现应用程序都有重要的影响，理解如何使用这两个概念可以获得更好的整体性能。

1、资源的有效利用

计算机在所有的运行过程中共享有限的资源。在最低一级，这些资源可分为以下几类:

- CPU时间
- 内存空间
- 大规模存储空间

所有的数据都驻留在内存中或某种大型存储设备上，必须由CPU来操作。一个高效的应用程序会仔细地使用所有这些资源。下面部分将详细介绍每种资源类型及其对程序的影响

1.1、CPU时间

CPU时间由系统分配，所以您必须尽可能充分利用您所拥有的时间。因为OS X和iOS都

实现了对称的多处理，所以系统上的每个线程都被分配了一个运行的时间片(最多10毫秒)。在那个时候(或者在很多情况下)，系统会收回对CPU的控制，并将其交给不同的线程。

在一个具有许多活动线程的典型系统中，如果每个线程都使用它的全部时间分配，性能将会很糟糕。这导致了编写应用程序最重要的目标之一:

目标:如果您的程序没有任何事情要做，那么它不应该占用CPU时间。

实现此目标的最佳方法是使用基于事件的模型。使用现代事件处理系统(如Cocoa和iOS中的系统)意味着程序的线程只有在有工作要做时才会运行。

当应用程序确实有工作要做时，它应该尽可能有效地使用CPU时间。这意味着选择适合您期望处理的数据量的算法。它还意味着使用其他系统资源，例如可用的向量单元(AltiVec或OS X中的SSE)或图形处理器，进行专门的操作，从而实现以下目标:

目标:尽可能将工作从CPU中移出

有关如何有效使用CPU时间的基本信息，请参阅基本优化技巧。有关提高绘图操作速度的技巧，请参阅绘图代码。

1.2、内存空间

现代计算硬件上的内存通常由逐渐缓慢(但较大)的内存类型组成。CPU可用的最快内存是CPU自己的寄存器。第二大缓存是L1缓存，其次是L2和L3缓存。下一个最快的内存是主内存。所有内存中最慢的内存都是由位于磁盘上的OS X中的虚拟内存页组成的，在使用它们之前必须进行分页。

在理想的情况下，每个应用程序都足够小，以适应系统最快的缓存内存。不幸的是，应用程序的大部分代码和数据要么驻留在主内存中，要么驻留在磁盘上。因此，重要的是，应用程序的代码和数据的组织方式要尽量减少在这些较慢介质中花费的时间，这将导致以下目标:

目标:减少程序的内存占用。

减少程序的内存占用可以显著提高其性能。一个小的内存占用通常有两个优点。首先，程序越小，占用的内存页面就越少。内存页更少，通常意味着分页更少。其次，代码通常更小，这是由于优化得更充分、组织得更好。因此，执行给定任务所需的指令较少，并且该任务的所有代码都集中在同一组内存页上。

除了减少应用程序的内存占用之外，还应该尝试减少应用程序中可写内存页的占用。可写内存页存储应用程序的全局或分配的数据。在OS X中，如果需要，可以将这些页面写到磁盘上，但这样做很慢。在iOS中，这些页面的内容必须由应用程序本身手动清除，这可能需要应用程序重新创建这些页面上的数据。在这两种情况下，系统释放内

存的工作都需要一些时间，这些时间可以更好地用于执行应用程序代码。

有关如何减少程序占用空间的基本信息，请参见应用程序占用空间。有关更有效地使用内存的技巧，请参阅内存分配代码。

1.3、大规模存储空间

任何计算机上的文件系统性能都很重要，因为几乎所有文件都驻留在某个文件中。与系统的其他部分相比，应用程序、数据甚至操作系统本身都位于必须从设备加载到内存的文件中，这些文件的速度非常慢。文件系统，无论是本地的还是基于网络的，都是性能的最大瓶颈之一。这又引出了另一个目标:

目标:消除不必要的文件操作，并将其他操作延迟到实际需要的信息。

通过消除或延迟文件操作来消除这个瓶颈对于提高应用程序的整体性能非常重要。在从文件请求数据到程序实际看到数据之间可以传递数千万个CPU周期。如果您的程序访问大量的文件，它可能会等待许多秒后才收到所有请求的数据。

另一件需要记住的重要事情是，您的应用程序及其创建的任何文件可能在网络上，而不是在本地硬盘上。特别地，OS X使网络尽可能地不可见，所以您永远不应该对文件的位置做出假设。

有关如何改进程序基于文件的性能的基本信息，请参见文件访问代码

2、感知速度

即使您调整应用程序代码以获得最佳性能，您的应用程序在用户看来也完全有可能显得很慢。问题是不可避免的:如果您有很多工作要做，那么您需要CPU时间和资源来完成这些工作。你可以做一些事情给你的应用程序一个速度的外观，这将导致以下目标:

目标:使你的程序响应用户。

对用户来说，响应性通常比原始速度更重要。只要程序能够及时响应命令，用户通常就愿意接受某些任务需要更长时间才能执行的事实。速度感知是通过让用户在程序处理后台数据时继续工作来实现的。改进应用程序执行的并发任务的数量是使其响应用户的好方法。并发通常是使用中央调度或线程来实现的。当应用程序的主线程响应用户时，分派队列或后台线程执行计算或处理其他耗时的任务。

另一种使应用程序快速出现的常见方法是改进它的启动时间。一个需要超过一到两秒钟就可以启动的应用程序可能做得太多了。它不仅在此期间对用户没有响应，而且还可能正在加载不需要的资源或根本不使用的资源，这是一种浪费。

有关如何改进启动时间的信息，请参见启动时间初始化代码。有关提高程序的感知性能的信息，请参见利用感知性能。

二、跟踪性能

确保高性能的唯一方法是在产品设计中包含性能目标，并在整个开发过程中根据这些目标对产品进行度量。在开发周期结束时，您不能将高性能移植到代码中;它与这个周期密切相关。编写代码时，重要的是了解它对您的程序的总体性能的影响。如果您及早发现性能问题，您就有很好的机会在为时已晚之前修复它们。

确定您是否达到或超过特定目标的方法是收集度量。苹果提供了一些工具来监视和分析程序的性能。您还可以直接在代码中构建度量工具，以帮助自动收集数据。无论你选择哪种方法，你都需要经常练习这些工具并分析结果。

2.1、建立基线指标

您需要做的第一件事是确定您想度量的一组基线指标。选择您认为对用户最重要的任务，并为执行这些任务确定一组约束。例如，您可能希望应用程序在1秒内加载并显示它的初始窗口，或者希望将总内存使用量保持在给定的目标范围内。

您选择度量的任务应该反映用户的需求。你的营销部门应该能够帮助你选择一组用户认为相关的任务。如果您有一个已建立的产品，请与您的用户交谈，找出他们认为慢的特性，并考虑在您计划的更新中改进这些特性的性能。

一旦有了要跟踪的任务列表，就需要确定每个任务的性能目标。对于现有的产品，您可能只是试图改进前一个版本的性能。您还可以尝试度量竞争产品的性能，并设置满足或超过其性能的目标。如果你有一个新产品，你可能需要用数字做实验才能找到合理的值。或者，您可能希望建立积极的基线值，并尽可能接近它们。

与任何性能度量一样，一致性是重要的。您建立基线度量的过程应该包含有关收集这些度量的系统的信息。详细记录系统的硬件和软件配置，并始终在相同的配置下运行测试。尝试使用最慢的硬件配置来建立基线。对一台快速计算机的测量可能会使您相信您的软件运行得很好，但是许多用户将使用较慢的处理器和较少的内存来运行计算机。

2.2、早测量，常测量

性能数据不是你收集一次的，就可以找到您希望程序中的所有性能瓶颈。如果维护程序性能的历史记录，就更容易发现问题。维护历史记录可以很容易地看到应用程序的性能是在提高还是在下降。如果它在下降，你可以采取行动在你的产品发货之前纠正问题。

定期度量性能的另一个原因是，您可以将这些结果与代码检查相关联。如果某个特定里程碑的性能下降，您可以检查在此期间签入的代码并尝试找出原因。类似地，如果性能提高，您可以使用最近的代码签入作为良好编程实践的模型，并鼓励您的团队使用类似的技术。

一旦有了部分功能的程序，就应该开始进行性能测试。随着新特性的添加，您可以为这些特性添加度量。将一组自动诊断例程直接集成到您的程序中可以使您的团队成员更容易立即看到结果。有了这些信息，在签入代码之前就可以更容易地修复性能问题。

2.3、分析你的结果

收集数据是识别性能瓶颈的最重要步骤。一旦你有了数据，你就可以用它来发现问题。分析性能数据并不像查看输出并立即发现问题那么简单。你可能很幸运，很快就能看到问题，但有些问题比较微妙，需要更仔细的分析。

一种方法是用图形化的方式来帮助我们分析结果。可视化性能数据可以帮助您更快地看到趋势，而不是在电子表格或其他基于文本的媒体中。例如，您可以规划时间来完成针对特定构建的单个操作，以确定性能是否正在改进，或者从构建到构建是否下降。在同一组里程碑上绘制多个数据集也可能揭示趋势，并提供有关性能增加或减少的洞察力。

2.3.1、高级算法分析

在分析性能数据时，请对问题所在的抽象级别保持开放的心态。假设您所拥有的数据表明在一个特定的函数中花费了大量的时间。可能是函数本身的代码可以优化以使其执行更快，但这是问题的真正原因吗?再次运行程序，但是这次要检查调用该函数的次数。如果这个函数被调用一百万次，那么问题可能出现在更高级别的算法中，即首先调用它。如果函数被调用一次，那么函数的主体很可能就是问题所在。

注意:Instruments是分析程序运行时行为的强大工具。Instruments允许您记录多个指标并并排显示它们，从而更容易发现趋势。Instruments的数据挖掘能力是另一种快速识别高级算法问题的好方法。有关工具和其他苹果提供的工具的更多信息，请参见性能工具。

性能工具本身有一些限制，您需要在分析数据时理解和考虑这些限制。例如，抽样程序可能指出应用程序花费大量时间的地方，但是在得出太多结论之前，您应该了解这些工具是如何收集数据的。采样工具不会跟踪每个函数调用。相反，他们提供一个统计分析你的程序基于样本在固定的间隔。使用这些工具的输出作为指导，但是一定要将其与您记录的其他数据关联起来。

2.3.2、其他分析技术

如果您对性能问题的真正原因有疑问，请避免对问题的原因做出假设。相反，通过将数据收集工作集中在相关代码上来细化分析。尝试使用不同的工具来收集新类型的信息。另一个工具可能提供一个独特的视角来揭示实际问题的更多信息。

您可以分析您的程序的其他一些方法包括:

- 注意调试器中的代码。在调试器中遍历代码可能会发现逻辑错误，从而减慢代码的速度。
- 将检查点添加到代码中，以记录有关该代码何时执行的信息。有关使用检查点跟踪初始化代码的示例，请参阅启动时性能指南。
- 尝试为问题编写替代解决方案，看看它们是否遇到类似的问题。