第二部分-基本性能优化建议

原著：Apple Inc.
翻译：Gavin

这一章为如何调整你的程序提供了实用的建议。它提供了您应该使用性能工具监视的领域的建议，并提供了一份改进性能的实用技巧清单。

一、公共区域监控

许多性能问题可以追溯到程序的特定部分。在设计和实现代码时，您应该监视这些区域，以确保它们满足您设置的性能目标。

1.1、为程序的关键任务编写代码

在设计程序时，请考虑用户最常遇到的任务或工作流。在实现阶段，请确保监视这些任务的代码，并确保它们的性能不会下降到可接受的级别以下。如果是这样，你应该立即采取行动来纠正问题。

程序执行的关键任务因程序而异。例如，在文本输入和显示过程中，文字处理器可能需要很快，而文件实用程序需要快速扫描硬盘上的文件和目录。由您决定用户最可能执行哪些任务。

有关如何识别和修复程序中缓慢操作的信息，请参见代码速度性能指南。

1.2、绘图代码

大多数程序都做一些绘图工作。如果您的程序只使用标准的窗口和控件，那么您可能不需要过多地担心绘图性能。但是，如果您执行任何自定义绘图，您需要监视您的绘图代码，并确保它在可接受的级别上执行。特别是，如果您支持以下任何一种，您应该研究优化绘图代码的方法。

- 现场调整
- 自定义视图绘图代码，特别是如果视图的某些部分可以在不更新整个视图的情况下进行更新
- 纹理图像
- 完全不透明的观点

有关如何优化绘图性能的信息，请参阅绘图性能指南。

1.3、启动时间初始化代码

启动时间是初始化程序数据结构并准备应用程序接收用户输入的时间。然而，许多程序在启动时所做的工作远比必要的多。在许多情况下，在启动时执行的任务可以推迟到应用程序发布其用户界面并开始处理事件之后。这种延迟给用户的感觉是您的应用程序很快，这是一个很好的第一印象。

对于需要在OS X version 10.3.3和更早版本中运行的应用程序，另一种改进启动时间的方法是预绑定应用程序。预绑定包括预计算库地址范围，并将这些值存储在应用程序二进制文件中。这个步骤消除了动态加载程序(dyld)在启动时计算这些地址范围的需要。对OS X版本10.3.4的dyld的改进使预绑定在以后的版本中变得不必要。同样，在iOS中不需要预绑定。

有关如何改进启动时性能的信息，请参阅启动时性能指南。

1.4、文件访问代码

文件系统是将信息输入内存和CPU的瓶颈。在访问文件所需的时间内，可能执行数千万条指令。因此，您必须检查您的程序使用文件的方式，并确保这些文件是实际需要和正确使用的。

最小化您使用的文件数量是改进与文件相关的性能的一种方法。当您访问文件时，请记住以下内容:

- 了解系统如何缓存文件数据，并知道如何优化这些缓存的使用。避免自己缓存数据，除非您计划多次引用它。
- 尽可能按顺序读写数据。在文件中跳来跳去需要额外的时间来寻找新的位置。
- 尽可能从文件中读取较大的数据块，记住一次读取过多的数据可能会导致不同的问题。例如，读取32mb文件的全部内容可能会在操作完成之前触发这些内容的分页。
- 避免不必要地关闭和重新打开文件。如果启用了缓存，那么即使数据没有更改，这样做也可能导致缓存刷新。

有关如何识别和修复与文件相关的性能问题的信息，请参阅文件系统性能指南。

1.5、应用程序占用

代码的大小会对系统性能产生巨大的影响。程序使用的内存页越多，系统和其他程序可用的内存页就越少。这种内存压力最终可能会导致分页和系统总体放缓。

管理您的代码占用是关于组织您的代码和数据结构。您需要确保内存中有正确的片段，并且不会导致不必要地读取或写入任何内存页。造成较大内存占用的一些问题如下:

- 代码页包含未使用的代码。编译器通常通过编译模块组织代码，这并不总是最好的技术。根据代码一起执行的内容组织模块可能是更好的选择。
- 静态或常量数据存储在可写页面上。在分页期间，此数据不必要地写入到磁盘。尽你所能将这些数据移动到可以快速清除的非可写页面。
- 一个程序输出的符号比实际需要的要多。符号占用了空间，并且只需要外部代码模块来调用程序。删除任何外部不应使用的符号。
- 编译器和链接器没有对代码进行适当的优化。一定要尝试编译器的优化设置，看看哪一个最适合您的程序。
- 程序包含了太多的框架。只加载程序实际使用的框架。

有关如何查找和修复代码占用问题的信息，请参阅代码大小性能指南。

1.6、内存分配的代码

程序为存储永久和临时数据结构分配内存。每个内存分配都有相关的成本，包括CPU时间和内存消耗。了解程序何时分配内存以及如何使用内存可以帮助您降低这两方面的成本。

理解程序的内存使用可以帮助确定减少这种使用的方法。您可以使用可用的性能工具来确定是否在自动上传的Objective-C对象引起过多的分页之前被释放。您还可以使用这些工具查找代码中的错误导致的内存泄漏。查看您调用malloc的次数也可以指出可以重用现有内存块而不是创建新内存块的地方。

分配内存时要遵循的一个重要规则是懒惰。将内存分配延迟到实际需要使用的内存时。有关内存分配懒惰的其他方法，请参见“懒惰”。

有关优化内存分配模式的信息，请参阅内存使用性能指南。

二、基本的优化建议

在开始实现新程序之前，应该考虑添加几个性能增强。尽管您可能无法在每种情况下利用所有这些增强，但至少应该在设计阶段考虑它们。

2.1、偷懒

提高性能的一个非常简单的方法是确保应用程序不执行任何不必要的工作。应用程序的每一分钟都应该用于响应用户当前的请求，而不是预测未来的请求。如果您不需要资源，比如包含首选项窗口的nib文件，不要加载它。这样的操作需要时间来执行，因为它访问文件系统，如果用户从未打开该首选项窗口，加载其nib文件的过程就是浪费时间。

要遵循的基本规则是，等待用户从应用程序请求一些东西，然后使用必要的资源来完成请求。您应该只在有可度量的性能好处的情况下缓存数据。假定应用程序的其余部分将运行得更快，则预加载缓存实际上会降低低内存情况下的性能。在这种情况下，缓存的数据可能在被使用之前被分页到磁盘。因此，缓存数据所获得的任何节省都变成了损失，因为在使用之前，数据必须从磁盘读取两次。如果您确实希望缓存数据，请在执行一次操作之后进行缓存。

其他一些懒惰的事情包括:

- 将内存分配延迟到实际需要内存时。
- 不要对内存块进行零初始化。调用calloc函数来延迟地完成它。
- 给系统一个延迟加载代码的机会。配置和组织您的代码，以便系统只加载当前操作所需的代码。
- 延迟读取文件的内容，直到您真正需要信息为止。

2.2、利用可感知的性能

在许多情况下，对性能的感知和实际性能一样有效。许多程序任务可以在后台、使用分派队列或空闲时间执行。这样做可以使应用程序的主线程自由地处理用户交互，并使程序界面对用户更有响应性。在设计程序时，考虑哪些任务可以有效地转移到后台。例如，如果您的程序需要扫描许多文件或执行冗长的计算，请使用分派队列。

另一种提高感知性能的方法是确保应用程序快速启动。在启动时，延迟任何对您的应用程序接口立即表示没有贡献的任务。例如，在应用程序完成启动并显示主窗口之后，延迟创建大型数据结构。如果您使用主窗口显示一些必须在启动时计算或检索的数据，请首先显示窗口，并显示进度指示符或指示数据正在加载的其他状态消息。对于使用插件的应用程序，在实际需要插件代码之前，避免加载插件。

2.3、使用事件处理程序

所有现代的Mac应用都应该使用Cocoa事件系统或Carbon event Manager。(类似地，iPhone应用程序必须使用UIKit框架提供的基于触摸的事件系统。)应用程序不应该通过轮询系统来检索事件。这样做效率非常低。事实上，当没有事件要处理时，轮询代码会浪费百分之百的CPU时间。现代基于事件的api旨在提供以下好处:

- 它们使你的程序更响应用户。
- 它们减少了应用程序的CPU使用量。
- 它们将应用程序的工作集最小化——在任何给定时间载入内存的代码页的数量。
- 它们允许系统积极地管理权力。

除了用户事件，在其他情况下也应该避免轮询。OS X和iOS中的线程使用run循环为计时器、网络事件和其他传入数据提供按需响应。许多框架还对某些任务使用异步编程模型，在任务完成时通知指定的处理程序函数或方法。在OS X v10.6和以后的版本中，分派源还为您提供了异步接收重要事件并在分派队列上执行它们的方法。

2.4、改进程序任务的并发性

在多核计算机上，并发是另一种提高程序的感知和实际性能的方法。能够同时执行多个任务的程序可以在多核计算机上并行执行这些任务。即使一台计算机只有一个核心，将代码分解成多个异步任务，如果操作正确，也可以提高速度。具体地说，您应该使用分派队列执行自定义任务，并让主线程自由处理用户事件并主要更新用户界面。

但是，在开始添加对并发的支持之前，一定要考虑如何有效地实现相应的任务。将代码分解为不同的任务需要考虑程序数据结构和代码路径。共享数据结构的任务可能需要使用串行分派队列来同步访问这些结构。

有关如何在程序中实现并发性的信息，请参阅并发编程指南

2.5、使用加速框架

如果您的应用程序对标量数据执行大量的数学计算，您应该考虑使用加速框架(accelerator.framework)来加速这些计算。加速框架利用任何可用的向量处理单元(如Intel x86 SSE扩展)并行执行多个计算。通过对框架进行编码，而不是向vector单元编码，您可以避免为每个平台体系结构创建单独的代码路径。加速框架对所有OS X支持的体系结构都进行了高度调优。

诸如工具之类的工具可以帮助指出程序中使用加速框架可能会受益的部分。有关使用这些工具和其他工具的更多信息，请参见性能工具。

2.6、应用程序现代化

如果你的程序被设计成运行在旧版本的Mac OS上，你应该更新你的代码以支持OS x的惯例。特别是，你应该避免使用旧的技术，比如碳或者遗留技术，比如QuickDraw。相反，您应该使用Cocoa或Cocoa Touch来构建应用程序。您还应该将二进制格式更新为Mach-O。Mach-O格式是基于intel的Macintosh计算机和基于ios的设备上唯一支持的格式。

有关可在Mac应用程序中使用的技术列表，请参阅Mac技术概述。有关iOS中可用的技术列表，请参见iOS技术概述。