Apress®

Pro .NET Performance

.NET 性能优化

萨沙·戈德斯汀（Sasha Goldshtein）
[美] 迪马·祖巴列夫（Dima Zurbalev） 著
伊多·弗莱托（Ido Flatow）
姚琪琳 刘夏 陈计节 禚娴静 译

- 来自.NET专家的建议
- 优化你的C#应用程序
- 通过将应用程序的性能最大化，使用户更开心、代码更快捷

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Pro .NET Perf

.NET 性能优化

萨沙·戈德斯汀（Sasha Goldshtein）
[美] 迪马·祖巴列夫（Dima Zurbalev） 著
伊多·弗莱托（Ido Flatow）
姚琪琳 刘夏 陈计节 禚娴静 译

人民邮电出版社
北京

图书在版编目（CIP）数据

.NET性能优化 / （美）萨沙·戈德斯汀
(Sasha Goldshtein)，（美）迪马·祖巴列夫
(Dima Zurbalev)，（美）伊多·弗莱托（Ido Flatow）
著；姚琪琳等译. -- 北京：人民邮电出版社，2018.8
ISBN 978-7-115-48586-1

Ⅰ. ①N… Ⅱ. ①萨… ②迪… ③伊… ④姚… Ⅲ. ①
计算机网络－程序设计 Ⅳ. ①TP393.09

中国版本图书馆CIP数据核字(2018)第118970号

版权声明

◆ 著 [美] 萨沙·戈德斯汀（Sasha Goldshtein）
[美] 迪马·祖巴列夫（Dima Zurbalev）
[美] 伊多·弗莱托（Ido Flatow）
译 姚琪琳 刘 夏 陈计节 禚娴静
责任编辑 杨大可
责任印制 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路 11 号
邮编 100164 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：18.75
字数：441 千字 2018 年 8 月第 1 版
印数：1－2 400 册 2018 年 8 月河北第 1 次印刷

著作权合同登记号 图字：01-2015-0750 号

定价：69.00 元

读者服务热线：(010)81055410 印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

内容提要

本书详细解释了影响应用程序性能的 Windows、CLR 和物理硬件的内部结构，并为读者提供了衡量代码如何独立于外部因素执行操作的知识和工具。书中提供了大量的 C#代码示例和技巧，将帮助读者最大限度地提高算法和应用程序的性能，提高个人竞争优势，使用更低的成本获取更多的用户。

本书共 11 章，第 1 章和第 2 章关注性能的度量指标及性能评测；第 3 章和第 4 章则深入 CLR 内部，专注于类型与 CLR 垃圾回收的内部实现；第 5~8 章及第 11 章讨论.NET 框架中的几个特定的方面，以及 CLR 提供的几种可用来进行性能优化的手段；第 9 章对复杂度理论和算法进行了简单的尝试；第 10 章则包含了一些独立话题，包括启动时间优化、异常及.NET 反射等。

本书适合已经拥有一定 C#语言和.NET 框架的编程基础，对相关概念较为熟悉的中高级程序员阅读学习。

序

截至目前（2012 年 2 月），最初的.NET 框架桌面版已经有 10 年历史了。我经历了这个团队从无到有，并在其中工作至今，且超过一半时间都是担任性能架构师。在.NET 十周年之际，我也在思索.NET 的现状及未来，以及该用何种“正确”的方式来考虑.NET 性能方面的问题。借着为这本专注于.NET 性能方面的书写序的机会，我想阐述一下我的一些思考。

为程序员提升生产力是.NET 框架一直以来的核心价值。垃圾回收（Garbage Collection，GC）是其中最为重要的方面，这不仅是因为它避免了一大类麻烦的问题（内存破坏），还因为它让类库编写者无法采用一些容易出错的内存分配模式（无须缓冲传递或制定一些“应该由谁来删除内存”这样的规则）。由强类型带来的类型安全（现在还包括泛型）是另一个重要方面，因为它迎合了程序员的一大类目标，还可以在程序尚未执行之前让工具发现许多错误。它要求软件组件之间定义严格的协议，这对于编写类库和大型项目来说至关重要。对于那些没有强类型的语言，如 JavaScript，总是在项目规模上升之后愈发显得乏力。基于这两大支柱，我们还添加了一系列着重于可用性的类库（极其统一，接口简单，一致的命名规范等）。我对结果十分自豪，我们创建了一个可以用来编写绝佳生产力代码的系统。

但是仅有程序员的生产力也是不够的，尤其是对于像.NET 运行时，这种软件中的基础部分来说。我们还希望有很高的性能，这也是本书的目的所在。

这里有个坏消息：就像你不能期望程序第一次运行就正确一样，你也无法期望程序一开始就“刚好”有着很高的性能。我们有各种工具（垃圾回收、类型安全）和技术（断言、接口协议）来减少错误，我们同样也有各种工具（不同的 Profiler）和技术（性能计划、热路径原型和延迟初始化）来降低出现性能问题的可能性。

不过好消息是：性能方面的问题也遵循“90-10”规则。一般来说，你的应用程序中有超过 90% 的代码对性能是不敏感的，它可以使用程序员生产力最大化的方式来编写（尽可能使用最少、最简化、最容易的方式来编写代码）。然而，剩下的 10%，则值得投入大量关注。编写这部分代码需要做出精细计划，甚至需要在代码编写之前进行，这也正是第 1 章所关注的事情。为了做出正确的计划，必须收集数据（各类操作及类库调用究竟有多快），为此需要测量工具（Profiler），这是第 2 章的内容。这些是所有高性能软件项目的基石，务必留心。假如真正掌握了这些内容，写出高性能的软件也不会是困难的事情。

而本书剩下的部分，则都是关于在制定性能计划时所需的“各类选项”。在基于一个平台编写软件的时候，了解它的各类基础性能及特征是一件无可替代的事情，这也是第 3 章（类型）、第 4 章（垃圾回收）及第 5 章（基础类库）所关注的内容。假如一个简单的基于.NET 基础实现的原型不能满足性能需求，首先应该研究算法方面的优化或并发方面的内容，因为它们往往可以带来最大的效益。假如还需要更高性能，则可以尝试一些.NET 特有的技巧（第 10 章）。假如其他尝试都失败了，可以在小范围内牺牲程序员的生产力，用非安全或非托管代码（第 8 章）来编写最关键的部分。

这里我想强调的观点：首先需要一个计划（第 1 章），这是一切的起点。在这个计划里会发现系统中的高流量或性能关键路径，然后花费额外的开发时间去仔细测量，并针对这些重要路径开发原型方案。接着，根据从原型中所获得的测量结果并结合本书的内容，一般来说想要达到性能目标的过程会很直接。可能只需要避免犯一些常见的性能错误，或采用一些简单的性能优化方式。也可能只需要对关键路径进行并行化访问，或编写一些非安全或非托管代码。无论采用哪种方式，最终的结果是获得一个满足性能目标的设计（通过对 10%的代码进行额外工作），并同时对其余的 90%代码保留.NET 的高生产力。这也是.NET 框架自身的目标：高生产力与高性能，一个都不会少。

这就是我的看法。一方面是坏消息——性能提高不是没有代价的，必须进行详细规划；另一方面是好消息——它也不难，而通过阅读本书，你能在编写高性能.NET 应用程序方面迈出最为重要的一步。

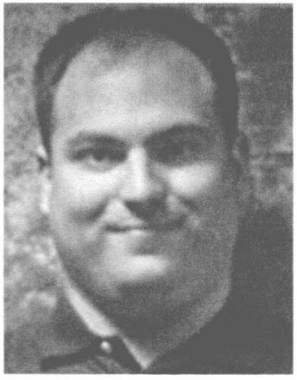
好好享受这本书吧，祝你好运！Sasha、Dima 和 Ido 给我们带来了十分出色的著作。

Vance Morrison
性能架构师，.NET 运行时团队

作者简介

Sasha Goldshtein 是微软公司 Visual C#方向的 MVP，也是 SELA Group 的首席技术官（CTO）。Sasha 领导了 SELA 技术中心的性能与排错团队，并且在多个领域提供咨询服务，包括生产环境调试、应用程序性能排错及分布式架构。Sasha 的经验主要集中在 C#与 C++应用程序开发，以及高可伸缩性和高性能系统架构等方面。他经常在微软公司的相关会议上发表演讲，并举办了如“.NET 性能” “.NET 调试”“深入 Windows”等多项培训课程。

他的 Twitter 账户：@goldshtn

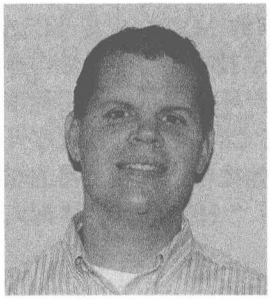


Dima Zurbalev 是 SELA Group 性能与调试团队紧急响应组的高级咨询师。Dima 在性能优化和排错上帮助客户完成了许多几乎不可能完成的任务，引导他们深入理解 CLR 及 Windows 的内部细节。他的大部分开发经验围绕.NET 与 C++基础项目进行，同时，他也在为 CodePlex 上的多个项目贡献代码。

Ido Flatow 是微软公司 Connected 系统方向的 MVP，也是 SELA 团队的高级架构师。他拥有超过 15 年的行业经验，目前是 SELA 的 Windows Azure 及 Web 领域的专家之一，专长为 WCF、ASP.NET、Silverlight 及 IIS 等技术。他是一名微软认证培训师（Microsoft Certified Trainer，MCT），也是微软官方 WCF 4.0 课程（10263A）的合作者。他同样也经常在微软公司的相关会议上发表演讲。

他的 Twitter 账户：@IdoFlatow

技术审阅者简介



Todd Meister 在信息技术（IT）行业工作超过 15 年了。他在 SQL Server 和.NET Framework 方面发表了超过 75 篇文章。除了技术编辑工作之外，他同时还是美国印第安纳州曼西市 Ball State University 的高级 IT 架构师。

Fabio Claudio Ferracchiati 是一位在前沿技术方面上多产的作家及技术审阅者。他给许多.NET、C#、Visual Basic、SQL Server、Silverlight 和 ASP.NET 等方面的书籍做出过贡献。他是一位.NET 方向的微软认证开发者（Microsoft Certified Solution Developer，MCSD）。他住在意大利罗马，受雇于 Brain Force。

致　谢

写书是一件费尽心神的事情。如果没有家庭、朋友及同事们的无私帮助，这本书可能还需要一年时间才能与大家见面。

SELA Group 的经理 David Bassa 在我们写这本书的过程中提供了大量的帮助，甚至在本书即将截稿之时，对不甚理想的项目进度还表示理解。

与 Apress 出版社的编辑团队的合作也十分愉快，他们容忍了我们非母语式的英语，并将书稿打磨成现在的样子。Gwenan、Kevin 还有 Corbin，谢谢你们的专业与耐心。

最后还要感谢我们的家人，你们为这本书的诞生牺牲了大量的时间。如果没有你们的支持，就不会有这本书。

前　言

之所以编写这本书，是因为我们觉得还没有哪本权威著作完整地覆盖了与.NET 应用程序性能相关的以下 3 个方面：

- 确定一个性能度量指标，并以此测量应用程序的性能是否达到或超过这个指标；
- 针对各方面（内存管理、网络、I/O、并发以及其他一些方面）进行应用程序性能优化；
- 在充分理解 CLR 及.NET 内部细节，以便设计高性能的应用程序，以及在性能问题出现之时及时修复。

我们相信，.NET 程序员只有在完全理解这 3 方面以后，才能开发出系统的高性能软件解决方案。例如，.NET 内存管理（由 CLR 垃圾回收完成）是一个极其复杂的方面，且有可能导致严重的性能问题，包括内存泄漏以及很长的 GC 暂停时间。如果不理解 CLR 垃圾回收的运作方式，在.NET 中实现高性能的内存管理就变成“撞大运”了。同样，是选择.NET 框架已经提供的集合类型，还是使用自己的实现，需要全面了解 CPU 缓存、运行时复杂度及多线程同步等问题。

本书的 11 章是按适合依次阅读的形式设计的，但读者也可以根据需要来阅读相应的部分，进行查漏补缺。本书各章按逻辑可以分为以下几个部分。

第 1 章和第 2 章关注性能的度量指标及性能的评测。我们会引入一些相关工具来测量应用程序性能。

第 3 章和第 4 章则深入 CLR 内部，专注类型及 CLR 垃圾回收的内部实现——这是应用程序性能优化中涉及内存管理的至关重要的两个话题。

第 5～8 章和第 11 章讨论.NET 框架中的几个特定的方面，以及 CLR 提供的几种可用来进行性能优化的手段——正确使用集合，将串行代码并行化，优化 I/O 及网络操作，高效实用互操作解决方法，以及对 Web 应用程序进行性能优化。

第 9 章对复杂度理论和算法进行简单的尝试。它让读者对算法优化有基本的了解。

第 10 章包含多个不太适合放入其他部分的话题，包括启动时间优化、异常和.NET 反射等。

为了更好地理解某些内容，部分话题会要求读者具备一些预备知识。这本书假设读者已经拥有足够的 C#语言及.NET 框架的编程经验，对相关的基础概念也较为熟悉，包括如下内容。

- Windows：线程、同步、虚拟内存。
- 通用语言运行时（Common Language Runtime，CLR）：即时（Just-In-Time，JIT）编译器、微软中间语言（Microsoft Intermediate Language，MSIL）、垃圾回收器。
- 计算机组成：主存、缓存、磁盘、显卡和网络接口。

书中包含大量示例程序、代码片段以及性能评测。我们不想让这本书篇幅变得很大，因此，往往只引用了其中的简要部分——不过读者可以在本书网站上找到完整的程序及其源代码。

部分章节会使用 x86 汇编语言来展示 CLR 的机制，或更直接地解释某个性能优化原理。尽管这些并不是本书想要表达的最重要的内容，但我们还是推荐读者花些许时间来了解 x86 编程语言的基

础知识。Randall Hyde 的免费书 *The Art of Assembly Language Programmer* 是绝佳的材料。

总而言之，本书包含大量的性能评测工具、用来提高应用程序细微之处性能的小贴士及小技巧、许多 CLR 机制的理论基础、实践代码示例以及根据作者经验总结出的案例。过去 10 年中我们一直在为客户优化应用程序，也从零开始设计高性能系统。这些年来，我们还培训了数百名程序员，让他们了解如何从软件开发的各个阶段思考应用程序性能。读完本书之后，读者会成为一名高阶的.NET 高性能应用程序开发者，以及针对现有应用的性能优化研究人员。

Sasha Goldshtein
Dima Zurbalev
Ido Flatow

目　录

第 1 章

性能指标

在开启.NET 性能之旅前，我们需要首先理解性能测试和优化的指标及目标。在第 2 章会探究一系列分析和监控工具。不过在此之前，需要确定哪些性能指标是值得关注的。

应用程序有不同的类型，其性能目标也各不相同，而它们与业务及运营需求密切相关。有些时候，应用程序架构决定了关键的性能指标。例如，你的 Web 服务器假如需要支撑数百万并行用户，则必然要引入多台服务器来实现分布式系统及负载均衡。而对于另外一些情况，性能度量结果决定了应用程序的架构。我们遇到过的大量案例都是在压力测试后系统不得不翻新架构，甚至更糟的是，它们直接在生产环境中崩溃了。

就我们过去的经验来看，相比中途贸然进行性能优化，事先确定系统的性能目标与其环境约束的做法更有针对性。下面是过去几年来我们诊断和修复的部分案例。

- 某台强劲的 Web 服务器上出现了严重性能问题，我们发现该问题是由测试工程师使用的低延迟 4 Mbit/s 共享链接引起的。由于不了解关键性的性能指标，工程师们花费了数天时间来调整 Web 服务器的性能，而事实上它一切正常。
- 为了改进一个富 UI 应用程序的滚动性能，我们调整了 CLR 垃圾回收器的行为，从表面上来看，两者并不相关。然而，通过精确控制内存分配的时机与调整垃圾回收（GC）的工作模式，我们去除了影响用户多时的 UI 延迟问题。
- 我们把一个硬盘移到了 SATA 端口，以此规避微软 SCSI 磁盘驱动里的缺陷（bug），让编译性能提高了 10 倍。
- 考虑到系统的伸缩性和 CPU 负载，我们调整了 WCF 序列化机制，从而把一个 WCF 服务中的消息尺寸减少了 90%。
- 我们进行了代码压缩，并仔细剥离启动时无须加载的依赖组件，把一个拥有 300 个程序集的大型应用程序，在一组过时硬件上的启动时间从 35 s 减少到了 12 s。

这些案例覆盖了各式各样的系统，从低功耗的触摸设备到拥有强大图形功能的高端用户工作站，直到多服务器的数据中心。它们展现了各种包含大量细节元素的独特性能特征。在本章中，我们将大致浏览一下常见现代软件类型中的各种性能指标。下一章里，我们将会说明如何准确测量这些指标，另外还展示了如何系统地优化它们。

1.1 性能目标

性能目标基本上取决于应用程序的外延和架构。当需求确定之后，你便需要决定大体的性能目

标。软件开发过程中会根据需求的变化、新的业务及运营需求来调整这些目标。我们先来了解几种典型的应用程序性能目标和运营规范的示例，但与任何性能相关的内容一样，这些指引需要适配你的软件领域。

首先，以下这些性能目标的说法并不恰当。

- 该应用程序需要在大量用户同时访问购物车界面时保持响应度。
- 该应用程序需要在用户数量正常的情况下不会占用大量内存。
- 单台数据库服务器需要在多台应用程序服务器满载的情况下快速响应查询请求。

这些说法的主要问题是过于一般化且过于主观。如果使用这样的性能目标，那么就会涉及不同参照系下的解读。业务分析师可能认为 10 万个并发用户属于“正常数量”，但一个技术团队的成员则深知单台机器是不可能支持得了这么多用户的。相反，开发人员可能会认为 500 ms 的响应时间实属正常，但一个用户界面专家则认为其过于卡顿且需要调整。

性能目标可以用可量化的性能指标来表示，这些指标还需要能够通过某种性能测试手段进行度量。性能目标同时也应该包含环境信息——一般或特定于其性能目标的。下面是一些定义恰当的性能目标示例。

- 在购物车界面的并发用户量不超过 5000 的情况下，该应用程序需要在 300 ms（不包括网络来回时间）以内处理完“重要”类别下的任意页面请求。
- 该应用程序的单个空闲用户会话的内存占用不能超过 4 KB。
- 在不超过 10 台应用服务器访问的情况下，数据库服务器的 CPU 和磁盘占用率不能超过 70%，且能够在 75 ms 内返回“常用”分类下的查询请求。

注意　上述示例中的“重要”“常用”分类为业务分析师或应用架构师所定义的通用术语。一般来说，确保所有方面的性能目标是不现实的做法，其花费的开发、硬件及运营成本很难获得合理回报。

现在，我们来关注几种典型应用程序的性能目标示例（见表 1-1）。这个列表不求详尽，不能作为清单，也不能作为你自己的性能目标模板来使用。它只是一份通用框架，用来展示不同类型应用程序的性能目标差异。

表 1-1　典型应用程序的性能目标示例

系统类型	性能目标	环境约束
外部 Web 服务器	从请求开始到回复生成完毕不得超过 300 ms	不超过 300 个并发活跃请求
外部 Web 服务器	虚拟内存占用（包括缓存）不得超过 1.3 GB	不超过 300 个并发活跃请求，且不超过 5000 个在线用户会话
应用服务器	CPU 使用率不得超过 75%	不超过 1000 个并发活跃 API 请求
应用服务器	硬页面失效不得超过每秒两次	不超过 1000 个并发活跃 API 请求
智能客户端应用程序	从双击桌面快捷方法开始算起，到员工列表主界面显示为止，不得超过 1500 ms	--
智能客户端应用程序	空闲时 CPU 占用不得超过 1%	--
Web 页面	包括顺序调整动画在内，邮件列表的过滤与排序操作不得超过 750 ms	单屏幕不超过 200 封邮件