

Ultra-Fast ASP.NET

Build Ultra-Fast and Ultra-Scalable web sites using ASP.NET and SQL Server

构建高性能可扩展 ASP.NET网站

[美] Richard Kiessig 著
余昭辉 译

- 让ASP.NET和SQL Server网站飞
- 迅速提升网站性能，全面挖掘网站潜力
- 微软资深技术专家力作，亚马逊全五星评价



人民邮电出版社
POSTS & TELECOM PRESS

TURING

图灵程序设计丛书

-86

Ultra-Fast ASP.NET

Build Ultra-Fast and Ultra-Scalable web sites using ASP.NET and SQL Server

构建高性能可扩展 ASP.NET网站

[美] Richard Kiessig 著
余昭辉 译

人民邮电出版社
北京

图书在版编目 (C I P) 数据

构建高性能可扩展ASP.NET网站 / (美) 基斯格 (Kiessig, R.) 著 ; 余昭辉译. — 北京 : 人民邮电出版社, 2011.3

(图灵程序设计丛书)

书名原文: Ultra-Fast ASP.NET: Build Ultra-Fast and Ultra-Scalable web sites using ASP.NET and SQL Server

ISBN 978-7-115-24833-6

I. ①构… II. ①基… ②余… III. ①主页制作—程序设计 IV. ①TP393.092

中国版本图书馆CIP数据核字(2011)第012631号

内 容 提 要

本书针对 ASP.NET 网站开发中可能遇到的问题, 给出了经过实践检验的具体解决方法。涉及的内容包括: 加快显示 HTML 的方法、缓存的最佳方式、如何使用 IIS、如何处理会话状态、如何配置 SQL Server 以及如何优化基础设施等。

本书适用于所有 Web 开发和运维人员以及对优化网站感兴趣的读者。

图灵程序设计丛书

构建高性能可扩展ASP.NET网站

-
- ◆ 著 [美] Richard Kiessig
 - 译 余昭辉
 - 责任编辑 明永玲
 - 执行编辑 丁晓昀
 - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
 - 邮编 100061 电子函件 315@ptpress.com.cn
 - 网址 <http://www.ptpress.com.cn>
 - 北京艺辉印刷有限公司印刷
 - ◆ 开本: 800×1000 1/16
 - 印张: 24
 - 字数: 597千字 2011年3月第1版
 - 印数: 1-3 000册 2011年3月北京第1次印刷
 - 著作权合同登记号 图字: 01-2010-2367号
 - ISBN 978-7-115-24833-6
-

定价: 65.00元

读者服务热线: (010)51095186 印装质量热线: (010)67129223

反盗版热线: (010)67171154

版 权 声 明

Original English language edition, entitled *Ultra-Fast ASP.NET: Build Ultra-Fast and Ultra-Scalable web sites using ASP.NET and SQL Server* by Richard Kiessig, published by Apress, 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705 USA.

Copyright © 2009 by Ray Lischner. Simplified Chinese-language edition copyright © 2011 by Posts & Telecom Press. All rights reserved.

本书中文简体字版由Apress L.P.授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

译者序

长期以来，在.NET Web 开发领域一直缺少一本用来讲解性能优化的书籍，大多数.NET Web 开发书籍都将大量笔墨倾注在控件使用、控件开发以及数据源操作上。当性能出现问题时，有的.NET Web 开发人员要么束手无策，因为这些控件对于他们来说就像一个黑盒；要么去求助其他平台的 Web 性能优化书籍，但这些书籍往往没有针对.NET 的特殊处理，优化效果也大打折扣。

现在，.NET Web 开发人员终于迎来了这么一本属于我们自己的 Web 性能优化手册。这本亚马逊五星级图书，本着压榨每一个字节的态度从 Web 前端优化、缓存、服务器到数据库后端各个方面给.NET Web 性能优化提供端到端的指导。书中包含许多真实的代码，大部分都可以在自己的项目中直接使用，有些优化只需要花费少量的时间，就可以收到立竿见影的回报。

本书分为 11 章，第 1 章是原则和方法，给全书定下了一个基调：我们要压榨 Web 开发中每个字节。第 2 章讨论客户端性能，包括网络连接、CSS 以及 JavaScript 等资源的优化。第 3 章是本书的重头戏——缓存，缓存在 Web 性能中可谓是一哥，这章按照缓存离用户的远近顺序做了全面的介绍。做 ASP.NET 肯定离不开 IIS，第 4 章以 IIS 7 为例，对 IIS 调优进行了讲解，原来 IIS 不是配一下网站就万事大吉了。第 5 章介绍了线程与会话，对于构建可扩展的 Web 架构尤为重要。第 6 章从 ASP.NET 特有的特性来讲解其对优化的作用，比如母版页，比如针对特定浏览器生成输出等。第 7 章介绍了 `HttpHandler` 和 `HttpModule` 等方面的内容，对于习惯拖控件构建 Web 页面的开发人员，这一章会让你看到 ASP.NET 的另一个世界。第 8 章、第 9 章介绍了 SQL Server 的优化，从缓存、索引等方面做了详细的介绍，第 8 章也是本书篇幅最大的一章。第 10 章从运维的角度来讨论优化。第 11 章是集成，将前 10 章所有的东西结合在一起使用。

本书的翻译和编辑工作差不多贯穿了 2010 年整整一年，其中因为工作变动对交付进度造成很大的影响，在这里要特别感谢图灵公司的编辑的体谅和支持。感谢好友王雄对部分章节的审阅。

最后，要感谢我的女友照云，非常感谢你这几个月的陪伴，对我因赶进度对你的冷落毫无怨言，甚至还自告奋勇帮忙审稿。

关于作者

1979 年我在加州大学圣巴巴拉分校 (UC Santa Barbara) 毕业，获得数学学士学位。毕业后我去了兰德公司 (Rand Corporation)，继续从事 Unix、C 以及 Internet 相关工作。20 世纪 80 年代，我去了硅谷 (Silicon Valley)，专攻底层操作系统，性能调优以及面向网络的应用。我曾从零编写过类 Unix 的操作系统，包括一个高性能文件系统。我还开发了一个基于 XNS 的网络栈，并且帮 Intel 的架构师开发了第一个从 Unix 到 x86 的移植版本。我还编写过一些 3D 的科学动画系统，以及一个门阵列模式布局 (gate array placement) 包。

20 世纪 90 年代，我编写了一个自定义实时操作系统，用在美国海军 F-18 飞机上。我开发的实时应用还用在航天器和相关的地面支持系统上，包括一个名为 Stellar Compass 的系统，使用行星的数字图像来测量飞行器状态。该软件曾登陆过一次月球，三次火星，而且与一个彗星打过照面并返回。我还曾经做过世界一流的图像卫星地面系统和各种飞行软件组件的首席架构师和设计者。

当我第一次听到 Java 时，我就疯狂地爱上它了。我使用 Java 开发的第一个大规模系统是一个音频会议系统。在这之后，我使用其开发了一个定制的高性能应用服务器。我帮助架构并构造了许多大规模基于 Java 的，数据密集型网站和 Web 应用，其中一个应用可以部署到两千万机顶盒中，为 TV 提供 Internet 支持。我最后一个基于 Java 的项目是构建一个面向文档管理的文件系统，我是许多相关模式的主要开发者。现在，许多金融机构还使用该系统解决风险管理问题。

在 1999 年末我去了 Microsoft。我的第一个项目是使用 C#、ASP.NET 和 SQL Server 开发一个通过面向 TV 的中间件平台分发 MSN 内容的综合架构，比如 WebTV。几年后，在开发完原始的系统后我进入了微软技术中心 (Microsoft Technology Center)，我的工作是建议 Microsoft 的大客户们关注他们系统架构中的 .NET 和 SQL Server。

将我的职业生涯联系在一起的主线是关注性能和可靠性。软件开发过程是我的另一个长期兴趣点，因为我曾直接看到它对项目的成功或失败有多大的影响。

在 2006 年 12 月，我们一家离开了压抑的硅谷，搬到了美丽的新西兰，就是我现在居住的地方。我的业余爱好包括业余无线电（呼号是 ZL2HAM）以及摄影。

前言

我在微软的工作经历给我的人生带来了意想不到的巨大转变。到微软之前，我所在的公司都是微软的竞争对手，时不时会听到反微软的言论和宣传。但是，当我听说.NET后，我就知道应该了解更多，而最好的方式就是到它的发源地去学习。

在进入微软公司深入了解这些技术之后，我被眼前的一切震撼了。.NET Framework、C#语言、ASP.NET以及SQL Server都非常先进，都有非常漂亮的技术架构。在使用Java（无疑也是非常优雅的语言）数年后，我可以使用一个集成良好的平台，将（几乎）所有东西无缝地集成在一起工作，确实令人精神振奋，干劲倍增。从技术层面上讲，我很认同平台开发人员的决策和权衡，而且最终的系统也能相当大地提高开发人员的效率。微软的工程团队也人才济济，极具创新意识，而给我这个曾经圈外人带来的最大惊奇是，他们打心眼儿里想要解决客户的问题。

我对技术的热衷让我成了直面客户的人——担任位于硅谷的微软技术中心的解决方案架构师。直面深层次的客户问题是另一项令人大开眼界的经历。首先，我可以在许多人或公司那里体会到微软技术产生的积极影响。其次，我还能感受到有些人面对很差劲的结果时的强烈挫败感。这本书，在某种程度上就是为减少挫败感而写的。

我的观点是ASP.NET和SQL Server很有潜力。但是，这些技术的关键点还未被很多开发人员认识到。我曾经与许多想要挖掘这种潜力，但却苦于不得其门的开发人员和经理交谈过。可是，认识到该技术全部的潜力必须投入更多努力；其中的特性丰富多彩，但要完全领略其风采必须有正确的视角。本书的一个目的就是赶走遮住技术美景的层层迷雾，让你亲眼目睹ASP.NET和SQL Server的美丽。

写这本书的另一个原因是我常被一些很慢的网站所折磨，希望本书介绍的内容能帮你改变这个现状。在Web的世界里一切皆有可能，未来的Web一定会超过现在的水平——但良好的性能是实现它们的基础。到那时谁也不会再遇到缓慢的网站。

我现在使用的是高于3 Mb/s的DSL连接到Internet，我的台式机4个CPU内核都工作在3GHz，与过去几年相比是出奇得快。即使这样的连网速度和处理能力，加载许多网站仍然要花很长时间，有时甚至加载一个页面都要好几分钟，而此时我的本地网络和CPU几乎是空闲的。作为软件专业人员，我们应该关注这种问题，因为出现这种问题实在太丢人了。我不仅要自己做得更好，更想让我的职业为人称道。为此，我们开发的网站就不仅要快，还要超快！

读者对象

本书的前两章和最后两章提供的信息几乎对所有 Web 开发人员都有用，且与使用的底层技术无关。中间 7 章是为使用 ASP.NET 和 SQL Server 设计、构建或维护网站的中高级的架构师和开发人员写的。刚刚从 Java 或 PHP 转到 .NET 的有经验的 Web 开发人员也会从本书找到很多有价值的信息。

本书也非常适合那些不是开发人员，但对优化网站技术感兴趣的读者。尤其是对参与网站运维、测试或管理的相关人员，这本书给出了许多开发团队应该重视的原则和问题，并提供了示例帮助理解。

联系作者

可以通过 rick@12titans.net 联系到我，也欢迎访问我的网站 <http://www.12titans.net/>。

我非常想听到你关于高性能方法的经验。

提高性能和可扩展性的技术会随着底层技术的变迁不断地发展。我非常想听到你发现很有效，但本书却没有涉及的技术。

欢迎指出正文或代码示例中的错误，也欢迎各种改进本书的建议。

致谢

我要感谢 Apress 优秀的团队：Ewan Buckingham 早期给了我大力支持和鼓励；Matthew Moodie 对全书的结构和流程提供了帮助；Simon Taylor 负责了技术审校，还复查了代码示例；Anita Castro 负责项目管理，Kim Wimpsett 和 Tiffany Taylor 帮我润色了本书的文字。

我还要感谢 Phil de Joux 的反馈意见。

目 录

第 1 章 原则和方法	1	2.5 少下载	22
1.1 性能和可扩展性的差异	2	2.5.1 减少每个页面中资源的数量	22
1.2 为什么需要高性能和高可扩展性	2	2.5.2 降低 HTML、CSS 和 JavaScript 的大小	24
1.2.1 优化	3	2.5.3 最大化压缩	25
1.2.2 过程	3	2.5.4 图片优化	25
1.2.3 体验	4	2.5.5 网站图标文件	29
1.3 完整的页面处理过程	4	2.5.6 HTML、CSS 和 JavaScript 的一般优化	29
1.4 原则概述	6	2.6 使用 JavaScript 限定页面请求	32
1.4.1 性能原则	6	2.6.1 提交按钮	33
1.4.2 次要技术	7	2.6.2 链接	33
1.5 本书使用的环境和工具	8	2.7 使用 JavaScript 降低 HTML 大小	34
1.5.1 软件工具和版本	8	2.7.1 生成重复的 HTML	34
1.5.2 术语	9	2.7.2 为标签添加重复文本	35
1.5.3 排版约定	9	2.8 减少上传	36
1.5.4 作者网站	9	2.9 CSS 优化	38
1.6 小结	10	2.10 图片精灵和集群	41
第 2 章 客户端性能	11	2.11 利用 DHTML	43
2.1 浏览器页面处理	12	2.12 使用 Ajax	43
2.1.1 网络连接和初始 HTTP 请求	12	2.13 使用 Silverlight	45
2.1.2 页面解析和新的资源请求	13	2.13.1 创建 HTML 控件	46
2.1.3 页面资源的顺序以及重新排序	14	2.13.2 在 JavaScript 中调用 Silverlight 内代码	48
2.2 浏览器缓存	15	2.13.3 使用 Silverlight 改进性能的其他方式	49
2.3 网络优化	16	2.14 提高呈现速度	50
2.4 脚本包含文件的处理	18	2.15 预缓存	51
2.4.1 通过在脚本之前排队资源请求提高并行化	18	2.15.1 预缓存图片	51
2.4.2 减少脚本文件的数量	19	2.15.2 预缓存 CSS 和 JavaScript	52
2.4.3 在页面剩余时间里请求对象	21	2.16 使用 CSS 实现无表格的布局	53
2.4.4 脚本延迟	22		
2.4.5 针对脚本的服务器端方法	22		

2.17 优化 JavaScript 的性能	56	4.5.2 设置压缩选项	120
2.18 小结	57	4.5.3 使用 web.config 配置压缩	121
第 3 章 缓存	59	4.5.4 缓存压缩的内容	121
3.1 在所有层中缓存	59	4.5.5 使用编程方式启用压缩	122
3.2 浏览器缓存	60	4.6 HTTP Keep-Alive	122
3.2.1 缓存静态内容	61	4.7 优化 URL	122
3.2.2 缓存动态内容	63	4.7.1 虚拟目录	123
3.3 ViewState	65	4.7.2 URL 重写	123
3.4 Cookie	71	4.8 管理流量	126
3.4.1 设置会话 Cookie	71	4.8.1 使用 robots.txt	126
3.4.2 单个 Cookie 中多个名称/值对	72	4.8.2 网站地图	127
3.4.3 Cookie 属性	72	4.8.3 带宽节流	128
3.5 Silverlight 独立存储	80	4.9 跟踪失败请求	130
3.5.1 示例程序:“欢迎回来”	80	4.10 IIS 性能调校提示	133
3.5.2 部署并更新 Silverlight 应用	84	4.11 小结	134
3.6 代理缓存	85	第 5 章 ASP.NET 线程与会话	135
3.6.1 使用 Cache-Control HTTP 头	85	5.1 线程影响可扩展性	135
3.6.2 管理相同内容的不同版本	87	5.2 ASP.NET 页面生命周期	136
3.7 Web 服务器缓存	87	5.3 应用程序线程池	137
3.7.1 Windows 内核缓存	87	5.3.1 同步页面	138
3.7.2 IIS 7 输出缓存	92	5.3.2 异步页面	138
3.7.3 ASP.NET 输出缓存	93	5.3.3 负载测试	140
3.7.4 ASP.NET 对象缓存	98	5.3.4 改进已有的同步页面的可 扩展性	142
3.8 SQL Server 缓存	103	5.3.5 从单个页面上执行多个异步 任务	143
3.9 分布式缓存	104	5.3.6 超时处理	144
3.10 缓存过期时间	105	5.3.7 异步 Web 服务	145
3.10.1 动态内容	105	5.3.8 异步文件 I/O	148
3.10.2 静态内容	105	5.3.9 异步 Web 请求	149
3.11 小结	106	5.4 后台工作线程	151
第 4 章 IIS 7	108	5.4.1 使用后台线程记录日志	152
4.1 应用程序池和 Web 园	108	5.4.2 任务串行化	157
4.2 请求处理管道	110	5.5 锁定指南和使用 ReaderWriterLock	157
4.3 Windows 系统资源管理器	111	5.6 会话状态	158
4.4 常见的 HTTP 问题	113	5.6.1 会话 ID	159
4.4.1 HTTP 重定向	114	5.6.2 InProc 模式	159
4.4.2 HTTP 头	115	5.6.3 使用 StateServer	160
4.5 压缩	118	5.6.4 使用 SQL Server	160
4.5.1 启用压缩	118		

5.6.5 有选择的启用会话状态, 使用 ReadOnly 模式	161	7.1.1 示例 HttpModule 的需求	195
5.6.6 可扩展性会话状态支持	162	7.1.2 Init() 方法	195
5.6.7 调校	167	7.1.3 PreRequestHandlerExecute 事件处理器	197
5.6.8 完全自定义会话状态	167	7.1.4 BeginAuthenticateRequest 事件处理器	197
5.6.9 会话序列化	168	7.1.5 EndAuthenticateRequest 事件处理器	199
5.7 会话状态的其他方案	168	7.1.6 EndRequest 事件处理器	200
5.8 小结	169	7.1.7 数据库表和存储过程	201
第 6 章 使用 ASP.NET 实现和管理优化技术	170	7.1.8 在 web.config 中注册该 HttpModule	202
6.1 母版页	170	7.2 自定义 HttpHandler	202
6.2 用户控件	173	7.2.1 开始请求	203
6.2.1 示例	173	7.2.2 结束请求	204
6.2.2 注册和使用控件	175	7.3 页面基类	205
6.2.3 将控件放在 DLL 中	175	7.4 页面适配器	206
6.3 主题	176	7.4.1 示例: PageStatePersister	206
6.3.1 静态文件	176	7.4.2 PageAdapter 类	207
6.3.2 皮肤	176	7.4.3 注册 PageAdapter	208
6.3.3 动态设定主题	176	7.5 URL 重写	208
6.3.4 可以放在主题里的属性	177	7.5.1 在 HttpModule 中重写 URL	208
6.3.5 示例	178	7.5.2 修改表单使用重写的 URL	209
6.3.6 预缓存主题图片	179	7.6 标签转换	210
6.4 特定于浏览器的代码	179	7.7 深入理解控件适配器	211
6.4.1 使用 Request.Browser	181	7.8 重定向	212
6.4.2 特定于浏览器属性前缀	182	7.8.1 传统的重定向	212
6.4.3 缓存特定于浏览器的页面	183	7.8.2 永久重定向	213
6.4.4 控件适配器	184	7.8.3 使用 Server.Transfer()	213
6.4.5 浏览器提供程序	185	7.9 尽早刷新响应	214
6.4.6 伪装	187	7.9.1 标记	214
6.5 动态生成 JavaScript 和 CSS	187	7.9.2 隐藏文件	214
6.5.1 示例	188	7.9.3 包跟踪	215
6.5.2 从 JavaScript 里访问 ASP.NET 控件	189	7.9.4 块编码	216
6.6 给静态文件设置多个域名	190	7.9.5 小结	217
6.7 修改图片大小	191	7.10 过滤空格	218
6.8 小结	193	7.11 避免不必要工作的其他方法	220
第 7 章 管理 ASP.NET 应用程序策略	194	7.11.1 检查 Page.IsPostBack	220
7.1 自定义 HttpModule	194	7.11.2 确定页面是否刷新	220
		7.11.3 避免在回发之后重定向	220

7.11.4	检查 Response.IsClient- Connected.....	221	8.13.1	分区函数.....	265
7.11.5	关闭调试模式.....	221	8.13.2	分区架构.....	265
7.11.6	批量编译.....	222	8.13.3	生成测试数据.....	266
7.12	小结.....	222	8.13.4	添加索引, 配置锁升级.....	268
第 8 章	SQL Server 关系数据库.....	224	8.13.5	存档旧数据.....	269
8.1	SQL Server 如何管理内存.....	224	8.13.6	小结.....	269
8.1.1	内存组织.....	225	8.14	全文搜索.....	269
8.1.2	读和写.....	225	8.14.1	创建全文目录和索引.....	270
8.1.3	性能影响.....	226	8.14.2	全文查询.....	271
8.2	存储过程.....	226	8.14.3	获得搜索分级的细节.....	271
8.3	批量命令.....	228	8.14.4	全文搜索语法小结.....	271
8.3.1	使用 SqlDataAdapter.....	228	8.15	Service Broker.....	272
8.3.2	构建参数化命令串.....	231	8.15.1	启用和配置 Service Broker.....	273
8.4	事务.....	233	8.15.2	发送消息的存储过程.....	274
8.5	多结果集.....	238	8.15.3	使用存储过程接收消息.....	274
8.5.1	使用 SqlDataReader. NextResult().....	238	8.15.4	测试示例.....	275
8.5.2	使用 SqlDataAdapter 和 DataSet.....	239	8.15.5	避免中毒消息.....	276
8.6	数据预缓存.....	240	8.16	通过 Service Broker 发送 E-mail.....	276
8.6.1	方法.....	240	8.16.1	创建后台工作线程.....	277
8.6.2	预缓存基于表单的数据.....	241	8.16.2	读取并处理信息.....	278
8.6.3	预缓存每次一个页面的数据.....	241	8.16.3	排队消息和发送邮件的 Web 窗体.....	280
8.7	数据访问层.....	242	8.16.4	结果.....	282
8.8	查询和结构优化.....	243	8.17	数据变化通知.....	283
8.9	其他查询优化指导原则.....	251	8.17.1	查询约束.....	283
8.10	数据分页.....	251	8.17.2	示例: 简单的配置系统.....	284
8.10.1	公用表表达式.....	251	8.18	Resource Governor.....	287
8.10.2	数据分页的详细信息.....	252	8.18.1	配置.....	287
8.11	对象关系模型.....	258	8.18.2	测试.....	290
8.12	XML 列.....	259	8.19	横向扩展与纵向扩展.....	290
8.12.1	XML 架构.....	260	8.19.1	纵向扩展.....	290
8.12.2	创建示例表.....	261	8.19.2	横向扩展.....	291
8.12.3	基本 XML 查询.....	262	8.19.3	确定系统瓶颈.....	292
8.12.4	修改 XML 数据.....	263	8.20	高可用性.....	293
8.12.5	XML 索引.....	263	8.21	其他方面的性能提示.....	294
8.12.6	其他 XML 查询技巧.....	264	8.22	小结.....	295
8.13	数据分区.....	264	第 9 章	SQL Server Analysis Services.....	297
			9.1	分析服务概览.....	297
			9.2	MDDb 示例.....	299

9.2.1 RDBMS 结构	299	10.8 临时环境	343
9.2.2 数据源视图	300	10.9 部署	343
9.2.3 Cube	303	10.9.1 数据层升级	344
9.2.4 时间维度	303	10.9.2 提高部署速度	344
9.2.5 Items 和 Users 维度	305	10.9.3 页面编译	345
9.2.6 计算的成员	306	10.9.4 预热缓存	345
9.2.7 部署和测试	307	10.10 服务器监控	346
9.3 示例 MDX 查询	307	10.11 小结	347
9.4 ADOMD.NET	313	第 11 章 综合起来	348
9.4.1 单个单元格结果示例	313	11.1 从何处入手	348
9.4.2 使用 GridView 显示多行结果	315	11.2 开发过程	349
9.5 使用 SSIS 更新 Cube	316	11.2.1 组织	350
9.6 预先缓存	319	11.2.2 项目阶段和里程碑	350
9.6.1 数据存储选项	319	11.2.3 编码	351
9.6.2 缓存模式	320	11.2.4 测试	351
9.7 使用中间数据库	323	11.2.5 Bug 跟踪	351
9.8 小结	324	11.2.6 用户反馈	352
第 10 章 基础和运维	325	11.2.7 高性能的技巧	352
10.1 插桩	325	11.3 League	354
10.2 容量规划	329	11.4 工具	355
10.3 磁盘子系统	330	11.5 架构	356
10.3.1 随机每秒钟 I/O 次数与顺序		11.6 备忘录	357
每秒钟 I/O 次数	330	11.6.1 原则和方法 (第 1 章)	357
10.3.2 NTFS 碎片	331	11.6.2 客户端性能 (第 2 章)	357
10.3.3 磁盘分区设计	333	11.6.3 缓存 (第 3 章)	359
10.3.4 RAID 选项	334	11.6.4 IIS 7 (第 4 章)	360
10.3.5 存储阵列网络	336	11.6.5 ASP.NET 线程和会话	
10.3.6 控制器缓存	337	(第 5 章)	361
10.3.7 固态硬盘	337	11.6.6 使用 ASP.NET 实现并管理	
10.4 网络设计	338	优化技术 (第 6 章)	361
10.4.1 巨型帧	338	11.6.7 管理 ASP.NET 应用策略	
10.4.2 链接集成	339	(第 7 章)	362
10.5 防火墙和路由器	340	11.6.8 SQL Server 关系数据库	
10.5.1 Windows 防火墙和防病毒		(第 8 章)	363
软件	341	11.6.9 SQL Server 分析服务	
10.5.2 使用路由器作为硬件防火		(第 9 章)	364
墙的替代方案	341	11.6.10 基础设施和运维 (第 10 章)	364
10.6 负载均衡器	341	11.7 小结	365
10.7 DNS	342	词汇表	367

第 1 章

原则和方法



建设现代的大型网站是一项令人惊讶的复杂工程。许多网站随着规模的扩大，都会遇到严重的性能和可扩展性问题。我经常见到一些大型网站在某一时刻，为了应付不断增长的访问量而不得不将设计推倒重来。所幸的是，如果能在网站规模还不大的时候就坚持遵循一些基本原则，不仅可以让当前网站的速度更快，而且还能避免将来网站成长过程中的一些问题。

本书会探讨这些原则，并帮助你理解为何及如何应用这些原则。

这些原则是我在近 30 年互联网软件开发实践的基础上总结出来的。从 1974 年开始我就从事因特网方面的工作，1979 年开始使用 Unix 和 C，后来转到 C++，接着又加入到 Java 和 C#阵营。在微软工作的日子里，我深入地学习了 ASP.NET 和 SQL Server。在那里，我为 MSN TV 设计并开发了一个大型网站。后来，在位于硅谷的 MTC（Microsoft Technology Center，微软技术中心）担任架构师的几年里，我对建设大型网站有了更深刻的理解。当时，我每周都要给微软的大客户们上一两堂架构设计课，每次两到三天。我和其他 MTC 的架构师会首先了解客户的烦恼和问题，然后从架构的角度提出解决方案。

没过多久我就发现，许多人都碰到了相同的问题，而其中大部分都是围绕性能和可扩展性的。以下是我们常听到的问题。

- 如何才能让 HTML 显示得更快？（第 2 章）
- 缓存的最佳方式是什么？（第 3 章）
- 如何使用 IIS 让网站更快？（第 4 章）
- 如何处理会话状态？（第 5 章）
- 如何改进 ASP.NET 代码？（第 5~7 章）
- 我的数据库为什么这么慢？（第 8 章和第 9 章）
- 如何优化基础设施和运维？（第 10 章）
- 从哪儿开始改进性能？（第 11 章）

本书会针对这些问题给出有效的解决方案。

我的做法是，不要将网站仅仅看成运行在远程服务器的应用程序，而要将其看成分布式的组件集合，这个集合应该像一个完整系统那样很好地协作。

本章首先从性能和可扩展性的差异谈起，接着谈一谈我对高性能和高可扩展性的理解。然后我从一个较高的层次来讲解网页生成的完整过程，并且描述一些核心原则，这些原则是提升网站

性能的关键原则。最后介绍一下本书中示例所采用的环境和工具。

1.1 性能和可扩展性的差异

当有人对我说他希望系统运行得更快的时候，我就会问：“你所说的快是什么意思？”典型的回答大概是：“应该能支持成千上万的用户。”即使是一个很慢的网站，也可以支持成千上万的用户。实际上，一些大型网站都很慢。

可扩展性和性能是完全不同的概念。在本书中，当我谈到提升网站性能时，我的意思是减少特定页面加载的时间或完成对特定用户可见的操作的时间。用户坐在电脑前所看到的就是“性能”。

另一方面，可扩展性与网站可以支持的用户数有关。可扩展性高的网站很容易通过添加更多的硬件和网络带宽（不对软件做重大变更）支持更多的用户，而且对性能的影响很小或没有影响。如果用户的增加导致网站严重变慢，而且添加硬件或增大带宽还不能解决问题，那么就说明网站已到了扩展的临界值了。可扩展性设计的一个目标就是提高这个极值，这是一个永恒的话题。

1.2 为什么需要高性能和高可扩展性

速度和可扩展性不仅仅是 Web 服务器端的话题，Web 开发的许多方面都能也都应该提升速度和可扩展性。所有代码都应该快速运行，不管是运行在客户端、Web 层还是数据层。所有页面（而不是其中一小部分）都应该快速显示，而且变更、修正以及部署也应该够快。

随着性能和可扩展性目标在项目中日益深入人心，一个良性互动的局面就会形成。不仅客户和用户会更满意，工程师也会更高兴并觉得更具挑战性。也许有人会不相信，其实做到这一点通常不需多少硬件，质保及运维团队也不用很多人。这就是我所说的高性能和易扩展的含义（简称高性能，易扩展则隐含其中）。

高性能不是靠“想到哪做到哪”的编程方式实现的。如果不以一种系统化的方式开发，架构问题就会出现，通过捷径获得的短期效益也会被抵消掉。大规模软件开发项目都是马拉松式的，不是短跑式的，预先的计划和准备可以获得巨大的长期效益。

表 1-1 是我归纳的高性能和易扩展方法的目标。

表 1-1 高性能和易扩展方法的目标

组 件	高性能和易扩展的目标
页面	在负载下，每个页面都是可扩展的，并且响应速度快
层	在负载下，每个层都是可扩展的，并且运行速度快
敏捷性	可以快速地响应业务需求的变化，而且在变化时能够保持性能和可扩展性
可维护性	快速地发现和修正性能相关的bug
运维	快速地部署和扩大网站，容量规划直观且可靠
硬件	服务器在负载下被充分利用，并且尽可能少使用硬件设备

从总体上看，构建运行速度快、可扩展的网站与制造赛车有诸多的相似性。为了达到预期目标，一开始就要考虑和设计 with 核心性能相关的各个方面因素。对赛车而言，需要确定参与哪种

比赛或哪个俱乐部。是参加 F1 (Formula One, 一级方程式)、普通型汽车赛、重型拉力赛、直线加速赛, 或者仅仅参加小型车赛? 如果只是为小型车赛设计汽车, 那么你不仅不能参加 F1, 而且, 如果想参加新等级的比赛, 还要完全抛开原来的设计, 然后重新开始。对于网站来说, 为你自己和朋友构建网站当然与构建 eBay 或 Yahoo 那样的大型站点完全不同。适合一个网站的设计可能完全不适合另外一个网站。

高级的赛车不仅要跑得快、换胎要快、加油要快, 而且更换引擎也要快。换句话说, 赛车的快是多种维度的。网站也应该在多个维度里表现得都够快。

与设计赛车只考虑让它跑得快但没有考虑安全性一样, 如果设计网站时只考虑高性能而忽略安全性的话, 那后果也是不堪设想的。因此, 在接下来的几章中我会时不时地讨论有关安全性的问题, 这些问题与性能都是密切相关的, 比如第 3 章中对 Cookie 的讨论。

1.2.1 优化

许多业界的专家已经指出, 优化有时候就是死亡陷阱, 只会浪费时间。构建高性能网站的关键是通过缜密的设计, 让网站完全不需要优化。不过, 就像赛车一样, 如果想获得最好的成绩, 就需要在研制过程中反复测量、调整、调校、修正, 并有所创新。只要你有时间、资金和想法, 就一定能在某些地方做得更好。

在实际工作中, 关键是要知道哪里会有性能和可扩展性问题, 哪种变更能带来最大的性能提升。比较几个车轮螺栓的重量简直就是在浪费时间, 但是使用合适的引擎混合比则可能赢得比赛。同理, 改进调用不怎么频繁的函数不会提高网站的可扩展性, 而使用异步页面则可以做到。

我的意思并不是说这些小事情不重要。实际上, 许多小问题也会迅速累积成大问题。但是, 当给任务设定优先级, 为其分配时间时, 就要先关注能产生显著效果的任务。对赛车做抛光处理, 也许可以让车跑得更快点, 但如果变速器不怎么样, 你就得先处理这个问题。给内部的 API 换个好记的名字也许会让你自己觉得很满足, 但更重要的还是减少客户端与服务器间的往返次数。

1.2.2 过程

高性能是一个追求, 一个过程。从架构和设计开始, 涉及系统的所有方面, 从开发到测试、部署、维护、升级以及优化。但是, 像制造赛车或其他任何复杂的项目一样, 在这个过程中, 你常常会有紧迫感, 期望尽快完成, 只要“够好”就行了。在确保商业目标的同时, 还要有效地提升网站性能, 关键就在于抓住那些对性能影响最大的核心问题。本书关注的是应该做的事情, 而不是探索所有可以做的事情, 目标是帮助你关注高影响的领域, 避免迷失在杂乱的过程中。

我曾经工作过的许多团队的管理层都难以接受在性能上花费精力。通常这些团队都会碰到性能危机, 而且这些危机有时还导致站点的推倒重来。管理层必然更关注功能, 而只要性能“足够好”就可以了。问题是在真正出现问题之前性能都会足够好。根据我的经验, 要想让管理层重视性能问题, 就不能把它列为一项功能。它不是功能, 安全和质量问题都不是功能。性能和其他与速度相关的方面是应用程序不可分割的一部分, 它们会影响每个功能。如果要制造一辆赛车, 让其跑得快不是一个可以在最后添加的额外功能, 它是总体设计的一部分, 应该在每个组件和每个过程之中都考虑到这个问题。

这里没有捷径。下面是完成这项工作的关键点。

- 全面深入理解系统的方方面面。
- 构建稳固的架构。
- 关注最主要的问题，知道哪些可以不用考虑或延期考虑。
- 懂得“凡事预则立，不预则废”的道理。
- 使用合适的软件开发过程和工具。

你也许听到过一些类似针对 Web 性能的“8 秒钟原则”。这是一种人为因素指标，意思是如果一个页面的加载时间超过 8 秒，用户就会感到不耐烦，从而去点击其他同类网站。本书会采取完全不同的方式而不是关注类似这样的原则。我们不关注主观的性能指标，而是首先立足于架构。就像把赛车放到合适的“赛道上”。然后，遵照一系列有效的指导原则来构建网站。有了这些基础，就无需花费很多精力做优化了。在一开始就应用一些高端的设计技术可以开阔你的眼界。不要等到竞争对手都晋级到 F1 了，你才想起抛弃自己的普通赛车。

1.2.3 体验

性能应该完全围绕用户体验展开。比如，加载整个页面的时间只是全部用户体验的一方面，感知性能甚至更重要。如果有用的内容“立即”显示，而一些广告在 10 秒钟之后显示，大部分用户不会抱怨，甚至许多用户都不会发觉。但是，如果按照相反的顺序显示页面，即本来应该后出现的广告先显示了出来，而有用的内容出现在广告之后，那你就面临丢失大部分用户的风险，即使整个页面的加载时间是一样的。

一个人构建和维护的小型 Web 站点也可以像构建更大的 Web 站点那样从这种方式中获益(想象普通赛车安装了 F1 赛车配件后的情景)。速度快的站点会比速度慢的站点吸引更多的流量和更多回头客。你也许只需要一个更小的服务器或更便宜的主机。你的用户也许会访问更多的页面。

当你关注架构和性能时，这里有一个使用 ASP.NET 和 SQL Server 的示例：一个软件开发人员独立构建的站点 plentyoffish.com，该站点现在是加拿大最高流量的站点之一。该站点每个月有超过 4 500 万次访问，12 亿页面浏览，每秒 500~600 次页面访问。然而这个网站只使用了 3 台负载均衡 Web 服务器，这些服务器使用的是双四核 CPU，8G 内存，还有一些数据库服务器，以及一个内容分发网络(CDN)。Web 服务器的 CPU 占用率一般在 30%左右。我不知道该站点的其他细节，但是查看它生成的 HTML 后，我有把握你可以使用本书提供的技术构建一个可与之媲美甚至更快的站点。

不幸的是，天下没有免费的午餐：构建高性能的站点比粗制滥造地制作网站需要更多思考和计划，而且需要更多的开发工作，当然大部分的工作量一般都集中在项目开始。经过长时间的运行，维护和开发成本实际会很小，而且还可以避免任何重写的成本。最后，我希望你认可为这些收益所付出的努力都是值得的。

1.3 完整的页面处理过程

人们一般都认为 Web 就是一个浏览器在网络连接的一端，Web 服务器和数据库在连接的另一端，如图 1-1 所示。