PROGRAMMER TO PROGRAMMER™

Visual Basic .NET Scalability Handbook

VB.NET
可伸缩性技术手册

(美) Damon Allison
Ben Hyrman 著
石相杰 吕莉莉 译

清华大学出版社

VB.NET 可伸缩性技术手册

(美) Damon Allison 著
Ben Hyrman
石相杰 吕莉莉 译

清华大学出版社
北 京

内 容 简 介

开发人员常常被寄予厚望——创建出健壮的符合业务要求的应用程序。但当前，创建高可伸缩性的应用程序的需求越来越普遍。

.NET 的引入大大提高了 VB 开发人员使用面向对象技术开发可伸缩的 n 层应用程序的能力。

本书逐层地剖析了 n 层架构中每一层所涉及的问题。书中用一个示范的应用程序讲解了如何在每一层着眼于可伸缩性的问题，创建一个伸缩性很高的应用程序。全书共分 6 章和 2 个附录，讲述了可伸缩性的规划、数据层、中间层、表示层，以及可伸缩性的测量等内容。

本书内容切合实际，适合希望了解如何开发可伸缩的企业级的应用程序的 VB.NET 程序员阅读。

EISBN：1-86100-788-4
Visual Basic .NET Scalability Handbook
Damon Allison，Ben Hyrman

北京市版权局著作权合同登记号　图字：01-2002-5384

图书在版编目(CIP)数据

VB.NET 可伸缩性技术手册/(美)艾历森，(美)赫曼著；石相杰，吕莉莉译. —北京：
清华大学出版社，2003
书名原文：Visual Basic .NET Scalability Handbook
ISBN 7-302-07038-5

Ⅰ.V…Ⅱ.①艾…②赫…③石…④吕…Ⅲ.BASIC 语言—程序设计—技术手册　Ⅳ.TP312-62

中国版本图书馆 CIP 数据核字(2003)第 070760 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969

组稿编辑：曹康
文稿编辑：于平
封面设计：康博
版式设计：康博
印 刷 者：北京市清华园胶印厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：21　**字数**：435 千字
版　　次：2003 年 8 月第 1 版　2003 年 8 月第 1 次印刷
书　　号：ISBN 7-302-07038-5/TP・5172
印　　数：1～4000
定　　价：43.00 元

前　言

人们总是期望开发人员能创建出符合业务需要的健壮的应用程序，但在今天的经济环境中，有一项需求与日俱增，即创建能快速扩展，以满足用户数量和用户计划内和计划外增长的需求。这意味着应用程序有了更多的用户，或者虽然用户数量不变，但对应用程序的要求越来越高。

随着 Microsoft 的.NET 的引入，Visual Basic 开发人员使用可靠的面向对象技术创建可伸缩的 n 层应用程序的能力大为提高。为了能够利用更多更好的硬件，您的应用程序的设计必须从一开始就考虑可伸缩性的问题。

本书逐层地剖析在开发可伸缩的应用程序时所涉及的种种问题。在数据层，您将了解到高效的数据库设计、存储过程和索引对可伸缩性有何影响；在数据访问层，您将学到如何利用 ADO.NET 的增强功能；在一般包含应用程序的大部分业务逻辑的中间层，您将看到如何设计和构建能够支持多种表示选择的可伸缩层；在表示层，您将领会怎样受益于 ASP.NET 的新功能，如有效的缓存和状态管理。

本书读者对象

Visual Basic .NET 手册丛书中的所有图书都是面向实践中的 Visual Basic .NET 编程人员，他们需要学习如何完成特定的任务。对于需要创建在当今的业务环境中能迅速满足计划内外的增长需求的、可伸缩的、企业级的应用程序的任何开发人员来说，本书都是一本很理想的参考书，并且掌握起来也不困难。

为了更好地学习本书，您需要安装 Visual Basic .NET 或 Visual Studio .NET 和 SQL Server 2000。

本书主要内容

本书包括一个开发人员构建可伸缩的应用程序所必须了解和掌握的所有基础知识。下面是每一章所包含的内容。

第 1 章　可伸缩性简介

本章介绍扩展应用程序以便满足当今企业需求所涉及到的问题。讨论了 Windows

DNA 模型的限制和.NET 所带来的巨大收益。还介绍了贯穿本书使用的一个演示应用程序 MyInvestmentWatch.com。

第 2 章　可伸缩性规划

在一开始就必须为可伸缩性做规划，并将其设计到应用程序中去。这个在设计和早期开发阶段就必须做的决策将对所设计的应用程序的扩展能力有极大的影响。本章从最实际的角度讨论了上扩和外扩。

第 3 章　数据层

数据层对于可伸缩性是极为重要的。它即使不是应用程序资源最密集的部分，也应该是资源最密集的部分之一。本章讨论了数据库设计和数据访问层的设计。对于数据库，讨论了规范化、存储过程和索引。对于数据访问层，探讨了 ADO.NET 在可伸缩性方面的巨大改进，包括新的断开连接的 DataSet 对象。我们还为演示应用程序建立了数据库和数据访问层。

第 4 章　中间层

中间层一般说来是包含应用程序的绝大部分业务逻辑的地方。本章讨论了一个独立的中间层的好处，并分别从逻辑的观点和物理的观点出发讨论了应该怎样设计一个有效的中间层。涵盖了两个对于用.NET 开发中间层的任何人来讲都是重要的主题：.NET Remoting 和 Web 服务。然后在我们的演示应用程序中显示了如何实现中间层。

第 5 章　表示层

表示层用来管理和响应用户请求，它必须有能力扩展，以便支持预期的平均并发用户和峰值并发用户。本章的主要焦点是 ASP.NET 在可伸缩性方面所做的改进，例如缓存和状态管理。我们通过在演示应用程序中实现一个表示层结束本章内容。

第 6 章　可伸缩性的测量

测量应用程序的性能和扩展能力，在贯穿它的整个生命周期都是至关重要的。本章涵盖了诸如单元测试、压力测试和代码插装等技术的讨论；还讨论了能帮助我们监视应用程序的一些工具，如代码剖视器、Windows 性能监视器和 Application Center Test。

附录 A　MyInvestmentWatch.com 数据库布局

本附录是用于我们的 MyInvestmentWatch.com 演示 Web 站点的数据库结构的方便的参考。

附录 B　技术支持、勘误表和代码下载

本附录指导读者如何从 Wrox 出版社得到对本书的技术支持，如何查找或提交书中的错误，以及如何从 Wrox 的站点下载代码。

目　　录

第1章　可伸缩性简介

人们总是希望软件开发人员开发出满足业务需求的各种健壮的系统。然而在当今的经济界，对这样一种应用程序的需求在不断增强——为满足用户新的需要而能迅速伸缩的应用程序。这种需求在互联网的许多站点上体现最明显——这些站点由于系统在自重下失效而造成的停用问题已经频频见诸报道。

1.1　可伸缩性入门

用户对当今的应用程序有更多的期望，这尽人皆知。他们希望有一个功能丰富、自定义的用户界面，通过此界面，能给他们的家用电脑、办公室膝上型电脑、或便携式电话提供动态的个性化的内容。一方面是用户的这种需求以及开发人员自己想给应用程序增加的功能和“酷”特性的渴望；另一方面是，开发人员不但必须支持更多的客户程序，而且必须支持更多种类型的客户程序这样一个事实，因此，应用程序开发人员必须在这两个方面进行权衡。

客户程序有多种类型，从传统的 Windows 基于 Win32 的应用程序，到基于 HTML 的 Web 应用程序，到手持设备。当发展趋势是带宽越来越宽、处理器速度越来越快，信息提供者之间的竞争越来越激烈时，底层的应用程序的体系结构和编码设计，也必须考虑到可伸缩性，以满足由于不仅客户程序的增多，而且还有客户端设备的增多所带来的不断增长的需求。

企业需要和期望它们的应用程序具有良好的可伸缩性。也就是说，企业需要这样的应用程序—— 既能迅速上扩以满足不断增长的业务需求，又能在系统利用不足时下扩。另外，企业还期望这些应用程序能迅速响应新的需求。一个没有或有很少几个报表的应用程序，可能被突然要求向一大群用户提供临时的报表。

很简单，企业需要和期望的是能迅速响应新的挑战的软件。换言之，需要的是用最小的努力就能满足新的需求的灵敏的软件，而可伸缩性就是这种灵敏的软件必备的基础之一。

一个可伸缩的系统就是这样一个系统，它能被扩展以承受更大的用户负载和改变了的用户行为而不用修改它现有的体系结构和设计，或至少保证这些修改带来尽可能小的影响。如果预期的系统使用率突然翻了一番，或系统承受着相当重的峰值负载，那么一个可伸缩系统将有能力应对这些挑战而不会带来昂贵的花费，或更糟糕的重新编

码。如果底层代码设计得当，那么伸缩一个系统将出人意料地并不会要求您做多少工作。相反，如果应用程序的底层代码的设计没考虑使用率的增长或变化的性质，要应对增长了的要求或峰值使用率的情况是不可能的。

更确切地说，应用程序设计的一个目标—— 可伸缩性—— 指的是提供这样一个系统，它能在所论问题的领域内满足当前和将来的负载量的需求。显然，并不需要每一个应用程序都能处理当今最大站点的通信量问题，或指望它们明天应对比今天多达400%的用户，但是它必须能应对预期的增长。不论应用程序增长的确切值是多少，设计师、基础结构支持人员和开发人员(最重要的是开发人员)必须回答这个问题："我的系统在应对用户可能大量增长的方面该如何定位？"

可伸缩性的另一个方面，也是在当今经济领域特别明显的问题，是建立的系统要能够下扩以应对使用率减少的要求。一个需要用大量硬件来维持功能的系统，当费用必须削减的时候，是不能很好地为公司服务的。

一个开发人员不论在软件开发过程中所担当的角色是什么，他最不希望做的是去解释：需要怎样进行重编码或重设计，一个系统才能应对预期的增长。因为一旦一个系统被设计完毕，再要增强它，那将很费时费力，且代价昂贵。如果在设计阶段对可伸缩性的重要性能有正确的理解，那就会使我们知晓常见的导致系统不可伸缩的习惯做法，并进行预测，且加以避免。当然，不要指望能设计出一个可以满足那些尚未预见的需求，或尚未明确说明需求的系统。但是，从一开始就必须保证：不能把您的系统归入一个脆弱的和不可伸缩的设计中。

1.1.1 Microsoft 和可伸缩性

长期以来，Microsoft 在创建不可伸缩的服务器软件，或至少是创建需要付出很大的代价才能得到可伸缩性的结构方面名闻天下。随着 .NET 的出台，Microsoft 在平息对它的批评方面迈出了一大步。现在来看一下 Microsoft 提供的产品，当然我们最关注的是它们的可伸缩性。

1. Microsoft: .NET 之前

Microsoft 始终在不断改进它的产品的可伸缩性。随着 Windows NT4 Symmetric Multi-Processor(对称多处理器，SMP)技术和具有群集能力的应用程序(如 Microsoft SQL Server)的出台，Microsoft 在增强系统可伸缩性方面迈进了一大步。Windows 2000 和 COM+跟随着 NT 的引导，继续为开发人员创建可伸缩的应用程序奠定基础。

对于许多应用程序来说，Visual Basic 6 在 Microsoft 环境中是一种理想的编程语言的选择。举例来说，在 Visual Basic 中创建一个具有完整功能的用户界面，是极为容易的。只要把可视控件拖放到用户界面窗体中，然后处理事件，或许再调用 COM 组件或

数据库，就可以了。

但是，当后来需要创建可伸缩的 Web 应用程序时，Visual Basic 的局限就变得引人注意了。事实上，从一个高伸缩性系统的某些方面来考虑，这些限制开始使得 Visual Basic 不可用了。或许人们可以发现，影响使用 Visual Basic 语言的常见的那句怨言是："Visual Basic 不能伸缩！"

在 Visual Basic 中，线程是受限制的。如果要在 Visual Basic 6 中创建 COM 组件，那么有关线程的选项就受到限制，而无法利用 C++语言中相同的线程选项。正是语言的这种局限性对开发人员的影响，致使开发人员在开发许多企业级应用系统时，避开了 Visual Basic 6 语言。那些想利用他们的硬件最大限度地提高性能的开发人员和设计师，舍弃了简捷的 Visual Basic，而选择了功能强大的 C++。

除了性能的限制之外，Visual Basic 在支持 OO(面向对象)编程方面也很有限。Visual Basic 版本 3 不支持 OO 概念，版本 4 开始有少许支持，直到最近的版本(VB6)仍然也只有很少的支持。例如，在 VB6 中，不提供实现继承、参数化的构造函数和方法重载这些功能。对于一个从另一种语言转向 VB 的开发人员来说，VB 的这些缺点足以使他们调头而返了。尽管 VB 具有易用和应用程序快速设计的特点，但是它的局限性已被证明，真正的 OO 程序员不能忍受这些局限性，于是只有选择使用更支持 OO 的语言，如 C++ 或 Java。

2. .NET 的效果

在 Windows 2000 和 COM+的基础上进一步创建的.NET 平台能使开发人员通过把更多的管道抽象到 Framework 中，从而设计出更健壮的系统。.NET 引入了一个统一的类型系统(type system)、版本更新(versioning)、并列部署(side-by-side deployment)、Remoting 和集成的 Web 服务支持，这里只举几个例子。从全局着眼，.NET 在设计和生产力的节省方面给开发人员更多的自由，允许开发人员把精力集中在建立可伸缩性更强的系统上。从技术上说，这些进步对于创建可伸缩系统是至关重要的。

随着.NET 的引入，Visual Basic 已经很成熟了，变成了编程语言舞台上的第一流选手。实际上，曾经限制 Visual Basic 使用的那些 OO 概念，已经全部集成到语言中。实现继承(Implementation inheritance)和那些普遍的概念，如多态性、多界面继承、重载，和构造函数，开发人员都可选用。

Visual Basic .NET 和它的"表兄弟"C#语言相比，仅缺少很少几个特性。VB.NET 不能调用非安全代码(指直接访问内存或硬件的代码)，也不能像 C#那样执行运算符重载。然而，这些性质对于普通的开发项目来说可能并不重要。

随着工业的聚焦点从基于客户的 Windows Form 应用程序转向基于 Web 的 ASP.NET 应用程序，更大的压力加在了服务器和用在服务器级别的语言上。如前所述，由于 Visual Basic 本身的局限性，Visual Basic 在企业应用程序中的空间受到了很大限制。有了.NET，这些局限性已被消除。Visual Basic 的功能已经大大超出了一般企业级

的功能需求。

1.1.2 本章主要内容

本书的主要目的是介绍如何用 VB.NET 开发可伸缩的应用程序。Microsoft .NET 和.NET Framework 给开发人员提供了许多关键代码和系统级的改进，这些都是难以控制的传统的 COM 和 Windows DNA 所不能提供的。

本章包括以下内容：

- **可伸缩性的奥妙**—— 有一些常用的概念需要探讨和解释，以使读者对可伸缩性有一个更具体的理解。尤其是可伸缩性的概念一般易与另外两个表示“能力”的概念相混淆：性能(performance)和可靠性(reliability)。虽然这些不同的“能力”之间是相关的，但按可伸缩性设计的系统与按另外两个性能设计的系统互不相同。
- **可伸缩性的重要性**—— 探讨在当前的计算环境下，确定可伸缩性的重要程度所常用的技术的发展趋势。通过示例分析，说明创建一个可伸缩应用程序的重要性，并说明对增长对于系统上的影响没有全面理解的情况下而设计出来的系统，将出现什么问题。
- **可伸缩性问题**—— 简单而浅显地讨论 VB6 给开发人员造成的局限。本节更详细地探讨这些局限，并同时分析.NET 是怎样解决这些问题的。
- **欢迎使用.NET**—— 随着.NET 的出现，先前向 VB 开发人员提出的许多问题，现在已不再是问题。本节探讨由这个新的、革命性的 Microsoft 平台所迈出的这一大步。
- **示例应用程序(MyInvestmentWatch.com)**—— 纵观全书，可见到诸多的程序代码及关于局限性和提高性能的讨论，这些局限性中有许多困扰着.NET 之前的系统。本书论述了这些局限性是如何用 .NET Framework 进行克服的。尤其是剖析了一个现成的 ASP / VB 应用程序，并逐章地展示在将其迁移到 VB.NET 的过程中，为找出存在的各个可伸缩性问题所做的工作。本书不是一本专门论述如何把 VB6 转换成 VB.NET，或把 ASP 转换成 ASP.NET 的书，但是，既然.NET 提供的许多性能改进找出了先前的 ASP 或 Windows DNA 的缺陷，那么，在准.NET 和.NET 代码之间进行一下比较，将会使我们更充分地理解和利用.NET 平台必须提供的内容。我们会觉得，在可伸缩性方面，.NET 已经有了一些非常令人兴奋的进展。

1.2 可伸缩性的奥妙

可伸缩性常常与在系统设计期间被讨论的其他的那些表示“能力”和系统特性的概

念相混淆。因为它们之间的差别很细微，所以把可伸缩性与其他的这些“能力”区别开来就变得很重要。为使问题的论述更条理整齐，我们要做的是将可伸缩性和它的两个近亲—— 性能(performance)和可靠性(reliability)—— 相比较。为了有效地将可伸缩性和其他的系统度量指标相比较，现在让我们仔细地研究一下可伸缩性的概念，请在继续阅读时努力记住这些定义和概念。

可伸缩性有以下特征：

- 可伸缩性是计算机应用程序或产品(硬件或软件)，在它(或它的上下文)的规模或容量被改变以适应用户需求的时候，能够继续正常执行它的功能的能力。一般是扩展规模或容量。这种改变可以是产品本身的(例如，在存储、RAM 等方面大小不同的一系列计算机系统)，或到一个新环境的改变(如新的操作系统)。
- 可伸缩性可以被粗略地定义为一个应用程序支持比当初设计所应对的用户数量大得多的用户的能力。例如，一个为 100 个用户同时使用而编写的 Web 应用程序，可能不是可伸缩的(也就是说，当用户数量增加到 300、500、1 万或 1 百万时，它可能不能按预期的功能和性能来正常运行)。可伸缩性不仅取决于软件而且还取决于服务器的能力和硬件的性能。

1.2.1　可伸缩性不等同于性能

似是而非的说法：

使一个系统具有高度的可伸缩性也会使它运行得更快。

为可伸缩性做规划和为性能作规划，常常会有同样的任务，但目标不同。可伸缩性的目标是在同一时间支持尽可能多的用户；而性能的目标是尽可能快地完成一项任务。显然，这两个概念是紧密关联的。更快地处理用户的能力将最终对提高可伸缩性提供帮助，但是，我们也将看到，为提高可伸缩性的目的所做的某些决策可能阻碍性能指标的提高。

现以一个信用银行模拟系统为例来说明。一个出纳员用 5 分钟办理一个顾客，增加一个出纳员并不能减少办理单个顾客的时间，但是这允许同时办理更多的顾客，也就是说，我们只是线性地扩展了我们的银行系统，就使同时办理的顾客数量增加了一倍，但办理任何单个请求所需要的时间并没有减少。

可伸缩性概念的关键是设计这样一个应用程序，它能被分隔和扩展到多个服务器。假定一个应用程序设计没有问题，这时添加一个物理中间层能增加可伸缩性，但是因为必须跨多个进程和网络边界时，它的性能会降低。反过来，可以把函数和对象组合起来以提高速度，但是因为这样做会降低分隔应用程序的能力，所以使可伸缩性降低。

但是事情并不像说的那么可怕。可伸缩性和性能两者并不是互相排斥的概念。您会看到，我们创建一个可伸缩系统所遵循的概念，也有助于创建一个运行速度很快的应用程序。例如，为使访问数据库的往返次数最少，在前端采取的数据缓存，既有助于提高可伸缩性，也有助于提高性能。还有另外一些考虑可以帮助提高性能，如尽量减少数据库调用、状态管理，和层之间的通信。

因为可伸缩性和性能两者互相缠绕在一起，所以从考虑性能的角度出发来编写和测试代码就变得很重要了。但也要注意避免过分地对性能进行不必要的优化。也就是说，如果花费过多的时间来优化一个报表，而这个报表一个月才运行一次，而在其他许多地方进行优化能立即产生效果，那就不必要优化这个报表了。有趣的是，还要注意到当应用程序的大小趋于增长时，应用程序的内核(请注意，应用程序的内核消耗大多数资源)一般相对来说比较小。编写坚固的测试脚本、使用代码分析工具(如剖视器，Profiler)，和用 Windows 系统工具监视系统性能，这些都是在编写代码的时候必须明了的。

1.2.2 可伸缩性不同于可靠性

似是而非的说法：

系统可伸缩性的提高也能使系统可靠性提高。

从考虑可靠性出发而建立的系统基础结构，尤其是一个基于 Windows 的系统，将与从考虑可伸缩性出发而建立的基础结构具有一些相同的特征。例如，二者常常都有群集的 Web 服务器和数据库服务器。然而，二者的目标不同。

创建一个高可靠性应用程序的目标是创建这样一个系统，它既能抵御失效，又能在万一失效的情况下自动恢复，且使数据丢失最少。显然，有些东西，如负载平衡 Web 服务器(load-balanced Web server)，既有助于提高可伸缩性，又有助于提高可靠性。通过添加第二个 Web 服务器，我们提高了可伸缩性，而且这台服务器在第一个服务器失效时，可以接管处理用户请求。

然而，有些对可靠性的考虑全然无助于可伸缩性。可以部署一个主动的 SQL Server 群集和一个被动的 SQL Server 群集，如果主动服务器失效，则被动节点立即接管下来。这大大改进了系统可靠性，但丝毫无助于可伸缩性。那个被动群集的接管与提高可伸缩性的目标毫不相干，这是因为它并没有增强系统处理更多的并发用户的能力。

这里也应该说一下，本书中所提到的应用程序在创建时都没有重点考虑可靠性的问题。冗余度、容错和可靠度等很多内容都是很广泛的话题，需要写出另外一本(或三本)这么厚的书来讨论。在适当之处，我们将指出在应用程序的每一层所能做的在可靠性方面的改进，系统的功能特性可以应用于那些地方是很明显的。例如，会话管理这个

概念，就很难保证可靠性，很难避免一旦服务器失效而不丢失数据。但是有了 ASP.NET 和 SQL Server Session Manager(SQL Server 会话管理程序)，会话管理就变得非常容易了。而进行会话管理所使用的方法恰好和我们获得可伸缩性所使用的方法相同。

1.3　可伸缩性的重要性

读到这里，我们已经知道这个问题的一些答案了。当然，可伸缩性的重要性所涵盖的内容很多，本节只能讨论其中的一部分。

1.3.1　规划内增长

在应用程序的环境下，通过观察当前的基线指示器(baseline indicator)，我们就可以预测系统的变化率。确定当前基线性能，并找正走势，这样才可能由可伸缩性出发来设计应用系统，而这一点很容易被忽视。

1. 当前基线指示器

系统需求的变化是不可避免的，正像企业不断演变一样。通过一般规划的实施，我们就可预测将来系统的需求。我们经过审查当前使用率模式，就会相当有把握地对可预见的未来一段时间进行预测。

系统的增长趋势必会反映业务的走势。如果业务的规模增长了，我们就可以预测系统会成正比地增长。一般应用程序有一个线性增长轨迹和已知的时间跨度。当然，这些增长特征仅仅是对一些典型系统而言，并不能很好地反映一个具有庞大系统的 Web 站点的增长。

在审查系统时，重要的是确定系统的当前使用率和那些基线。当设计和规划体系结构要求时，在关于可伸缩性方面，我们需要问自己以下几个问题：

- 当前的使用率模式是什么？在既定条件下，当前系统完成预定功能的情况如何？
- 历史增长率是多少？我们的系统能否满足今后几年的增长？
- 规划内增长率是多少(业务的和系统的)？
- 应用程序的哪些部分用少量的投资即可进行足够的扩展？
- 应用程序的哪些部分需用大量投资才能满足需求？
- 哪些系统对一个企业的核心是至关重要的？它们怎样应对预期的增长率？

通过问我们自己这些问题，就可以开始确定可伸缩性基线。这些假设和参数为系统建立了约束条件。通过确定一个基线的性能，我们即可在可预料的业务变化中恰当地为自己定位。