SAMS

Microsoft
核心技术丛书

LINQ
编程技术内幕

LINQ Unleashed: For C#

（美）Paul Kimmel 著
唐学韬 等译

机械工业出版社
China Machine Press

LINQ

编程技术内幕

LINQ Unleashed: For C#

(美) Paul Kimmel 著
唐学韬 等译

机械工业出版社
China Machine Press

本书结合 C# 3.0 和 Visual Studio 2008 对 LINQ 进行了实操型讲解。本书介绍了 LINQ 编程的各个方面，展示了 LINQ 是如何帮你显著提高生产效率的，还告诉了你应当如何用 LINQ 创建具有更高可靠性和可维护性的应用程序。

本书首先介绍了 LINQ 所用到的最新的 C#编程技术，比如匿名类型、分部方法以及 Lambda 表达式等。接着，通过具体的示例和易于改写的范例代码，分别针对对象、数据库和 XML 详细介绍了 LINQ 这个强大的新技术。你将会对 LINQ 的工作方式有一个深入而实用的理解，还会学到如何通过 XML 与其它数据模型的结合来实现各种各样的数据处理功能。

本书不仅适合 LINQ 的初学者，而且对 .NET 数据处理人员、ASP.NET 开发人员和 Windows Form 开发人员都有非常大的帮助，同时也可以作为高等学校计算机或相关专业的教材使用。

图书在版编目（CIP）数据

LINQ 编程技术内幕/（美）基默（Kimmel, P.）著；唐学韬等译. —北京：机械工业出版社，2009.6

书名原文：LINQ Unleashed: For C#

ISBN 978-7-111-26759-1

Ⅰ.L…　Ⅱ.①基…　②唐…　Ⅲ.①计算机网络－程序设计　②C 语言－程序设计　Ⅳ.TP393　TP312

中国版本图书馆 CIP 数据核字（2009）第 050789 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）

责任编辑：李东震

北京京师印务有限公司印刷

2009 年 6 月第 1 版第 1 次印刷

186mm×240mm · 26.75 印张

标准书号：ISBN 978-7-111-26759-1

定价：59.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

译 者 序

非常荣幸我能够翻译这本书，不仅因为它是 SAMS 公司大名鼎鼎的 Unleashed 系列的一员，更因为我平时也经常关注 Paul Kimmel 的系列文章。对我而言，Paul Kimmel 是一个爱好非常广泛的程序员前辈，这一点从他的书或文章中能够很明显地感觉到，而我也一直想成为他这样的程序员。因此，这次当机械工业出版社华章分社的陈冀康老师找我翻译这本书的时候，我着实兴奋了一把。

本书的用辞很有意思，我在拿到原书之后，花了 5 天的时间一口气读完了，感觉就像是在看小说。Paul Kimmel 不仅给我们带来了 LINQ 的完整学习方案，而且还在其中加入了不少的小故事，让读者能够更加愉快地学习 LINQ 这一门新技术，真正实现了“寓教于乐”的教学模式。

本书分四部分，共 23 章，分别介绍了 LINQ 的各个方面。在正式开始介绍 LINQ 之前，Paul 在第一部分向我们讲解了使用 LINQ 所必需的一些基础知识，如匿名类型（Anonymous Type）和 Lambda 表达式（Lambda Expression）等，这使得我们可以在后续的学习过程中不会感到一丁点儿的吃力。在第二部分中，Paul 向我们介绍了针对对象的 LINQ。在这部分中，他着重介绍了 LINQ 的一些基本概念，为了使我们能够更好地理解这些知识，他还安排了一个章节的内容来介绍如何用 LINQ 操作 Outlook 和 AD 中的数据。在第三部分中，Paul 着重向我们介绍了针对数据的 LINQ，还前瞻性地介绍了 ADO. NET 3. 0 和 Entity Framework（当然，这本书出版的时候，这些东西差不多都已经正式发布了）。在本书最后一部分中，Paul 介绍了针对 XML 的 LINQ，在最后一章中还介绍了我们在 XML 变成时期待已久的智能感知解决方案——LINQ to XSD。由此可见，本书所讲解的内容真真切切地覆盖了 LINQ 编程的方方面面。

在我看来，Paul 真的是做了一件很漂亮的工作，因为他不仅系统全面地介绍了 LINQ 这门新技术，而且还能让许多由于对 LINQ 一知半解而对其嗤之以鼻的人重新认识它。

大家准备好了吗？那么，我们就跟着 Paul 踏上美妙的 LINQ 之旅吧！

唐学韬

2009 年 5 月于广州

前　言

从我第一次写代码算起，到你把这本书拿到手的那个时候，我就已经写了30年的代码了。最早的那段代码是用一台TRS－80上的ROM－BASIC编写的，那会儿我正在密歇根州奥沃索的华盛顿中学念五年级。让“坦克”倒着走并在屏幕上来回地放子弹，这的确是一件很爽的事情。还有更爽的，那就是，通过调整代码的方式来改变子弹的速度和目标的数量。年复一年，三十年过去了，我越来越为此而感到兴奋。有不少很棒的技术即将出现在我们面前，比如微软的Surface、Popfly以及LINQ等。本书就是关于LINQ（Language INtegrated Query，语言集成查询）的。

LINQ是为C#开发的一种类似于SQL的语言。我第一次看到它时，我真的不喜欢它。我的第一印象是，有人把C#搞得更烂了，用这玩意儿写出来的东西跟SQL一样丑。我不喜欢它完全是因为我还不了解它。不过，我给了LINQ第二次机会（我希望你也能这样），然后我发现，LINQ是高度集成的而且非常强大，简直就像是一辆Tesla Roadster或是一架Extra 300L。

LINQ的查询能力扩展到了对象、SQL、数据集、XML、XSD、实体上面，此外，还能扩展到其他提供者上面去，比如活动目录和SharePoint。也就是说，你可以针对对象、数据、XML、XSD、实体，或是活动目录而编写不同的查询语句（语法上都是相似的），就像是在数据库中编写SQL查询一样。另外，LINQ非常巧妙且出色的构建于泛型技术以及．NET 3．5中的一些新特性（比如扩展方法、匿名类型和Lambda表达式）之上。LINQ的另一个重要特点就是，它非常明显的表露出了微软的一种意愿，即既要革新又要利用那些最好的现有技术（如20世纪30年代出现的Lambda Calculus（λ演算））；只要觉得某种技术还不错，那么就把这些元素吸收到我们所喜爱的工具或语言中去。

LINQ及其构成基础都很强大，在本书中，你将会学到许多知识，这将有助于你全面掌握LINQ的使用方法并快速上手。你将学到有关匿名方法、扩展方法、Lambda表达式、状态机等知识，并学习如何编写LINQ查询，还会了解到泛型和CodeDOM在诸如LINQ这样的强大工具中所扮演的角色。你将弄清楚一个问题，即为什么要在更大更复杂的应用场景中使用LINQ。你将意识到，如果不用再编写那些你没有必要亲自动手的代码的话，究竟可以省下多少时间和精力。此外，你还将认识到，LINQ其实是很适合于n层架构的，它能使您在沿其总体方针的情况下，按时完成任务。

本书是由一名获得了四次Microsoft MVP称号且做了十多年专栏作者的人编写的，本书将向你讲解有关LINQ以及．NET 3．5新特性的一切知识，并告诉你如何变得比以前更有效率且获得更多的乐趣。

本书使用约定

本书所使用的排版约定如下：

代码行、命令、语句、变量以及你将在屏幕上看到的文字将使用等宽字体。

代码清单中偶尔出现的粗体字是用来提醒你当前所讨论的就是这段代码。

语法描述中的占位符将使用斜体等宽字体。你需要将这些占位符替换为实际的文件名、参

数，或是它所代表的其他任何元素。

斜体字用于突出显示当前正在定义的技术术语。

代码行前面的连接符图标说明该行是上一行的延续。有时，一行代码会很长，因此无法在页面的一个单行上显示出来。如果你在某个代码行的前面看到了➥，那么请记得该行是前一行的一部分。

本书也含有注意、提示、警告等内容，它们将帮助你更快地找到重要或有用的信息。

致　谢

有时，我也会看看别人所写的致谢，不过，除非上面有你的名字，否则那一溜长长的名单实在是显得有些无聊。如果你现在很有空，而且又刚好读到这里的话，那么就请你尽量帮我感谢一下所有那些使这本书顺利出版的人士吧。如果你是这里所提到的人之一，那么请接受我的深切谢意，感谢你对我所作出的一切帮助。

感谢 Neil Rowe（我的组稿编辑）、Curt Johnson（Pearson 的市场部经理），还有 Joan Murray（Pearson 的编辑）。这本书我写得比较慢，不过完成一本书真的是一件很激动的事情哦。

感谢 Joe Kunk，感谢他没有抱怨技术编辑方面的工作量，也感谢他认真地测试了书中的那些示例。Joe 还为 Glugnet East Lansing and Flint（我们的 .NET 用户组）做了很多粗重活。还要感谢 glugnet 委员会的其他成员，我不在的这段时间，他们帮我做了不少的事情：Vivek Joshi、Alireza Namvar、Eric Vogel、Jeff McWherter、Vijay Jagdale、Jason Harris、Aaron Lilywhite 以及 Rich Hamilton。

感谢微软的 Bart De Smet，他在 Active Directory IQueryable Provider 方面给了我很大的帮助（LINQ to AD 真的很酷）。多谢 Daryl Hogan 帮我写了前言。

我还要感谢 Brian Dawson、Brad Jones、Tyler Durden、Jackson Wayfare，感谢开发出魔兽的暴雪以及开发出 GTA IV 的 Rockstar Games，感谢我的孩子 Alex 和 Noah，感谢 Dena 和 Joe Swanson（为我提供了酒水和漫画），感谢 Ed Swanson（为我提供了雪茄）。还要感谢 Pearson 的伙伴们，正是因为他们，我这乱七八糟的手稿才能变成一本闪亮光鲜的新书。

序

数据几乎影响着我们生活中的方方面面。我们所做的每件事情其实就是分析查阅数据，然后再以优惠券或其他市场营销手段对分析结果做出反应。在编写一个应用程序的时候，你可以清楚地知道这种或那种数据将会成为该解决方案的一部分。对于软件开发人员来说，在开发大型应用程序的时候，降低存取数据以及分析数据时的难度是很有必要的。事实上，数据会以各种各样的形式出现，人们很快便找到了一个具有很高价值的办法，即使用一个统一的框架来访问各种类型的数据。

多年以来，可供 Windows 开发人员使用的各种数据访问方法层出不穷。ADO 和 OLEDB，以及再后来的 ADO. NET 都为我们提供了一种统一的关系型数据库访问方法。MSXML 和 ADO. NET 使我们能够遍历和操作 XML 文档。虽然这些技术都有着不同的优点和缺点，但是它们都有着同一个问题：它们都无法让开发人员在编写数据访问操作代码的时候觉得舒服、自然。

现在，LINQ 使数据访问成为了 .NET 中的一个高级编程概念，它使得开发人员能够用一种更有意义的方式来表述其所期望的查询。LINQ 能够如此的强大，那是因为它让开发人员能够完全依靠智能感知技术来创建类型安全的数据访问代码和编译期的语法检查。

Paul Kimmel 做了一件非常漂亮的事情，因为他用一种简洁而又完整的方式介绍了 LINQ。他不仅使你了解了 LINQ，而且还非常专业地阐述了诸如匿名类型（Anonymous Type）以及 Lambda 表达式（Lambda Expression）等概念。本书中的示例代码给出了一些利用该技术的应用程序，它们都是以一种简明扼要的方式呈现出来的。这是一本非常不错的图书，可以在周六早晨沏一壶咖啡边喝边看。我希望你也能研读一下这本书，然后你会跟我一样从中学到很多知识的。

Darryl Hogan

Architect Evangelist，Microsoft

目　录

第一部分　为 LINQ 作准备

第 1 章　使用匿名类型

“从开始的地方开始吧，一直读到末尾，然后停止。”

——出自 Lewis Carroll 所著的《爱丽丝漫游奇境记》

本章学习内容：

- 理解匿名类型
- 使用匿名类型
- 匿名类型的数据绑定
- 测试匿名类型的相等性
- 通过 LINQ 查询使用匿名类型
- 泛型匿名方法简介

对于计算机图书而言，寻找起点的工作往往都是比较主观的。这是因为许多东西会依赖于更多其他东西。通常，最好的办法就是在地上插一根木桩，然后直接从那里开始。匿名类型（Anonymous type）就是我们的木桩。

匿名类型使用关键字 var。var 是个很有意思的东西，因为直到今天，Pascal 和 Delphi 仍然在使用它，不过 Delphi 中的 var 只是相当于 Visual Basic（VB）中的 ByRef 或是 C#中的 ref。. NET3. 5 所引入的 var 用于表示匿名类型。现在，我们那些使用 VB 的朋友们可能要说了，“嗯，variant 我们已经用了很多年了，真是赚大了。”不过 var 既没有使 C#简单化也没有使其复杂化。匿名类型是一种新的且很重要的东西。

在学习匿名类型之前，先来看看我们的最终目标。我们的最终目标是掌握 C#中的针对对象、可扩展标记语言（Extensible Markup Language，XML）以及数据的 LINQ（集成查询）。我们想要这么干的原因是因为它不仅很酷而且很有意思，更重要的是，它非常强大。为了实现这个最终目标，我们首先必须找到一个地方开始才行，这里，匿名类型就是我们的起点。

简而言之，匿名类型就是说你不用指定具体的类型。你只需要写上 var 即可，C#将计算出右侧表达式所定义的数据类型，然后 C#再发出（emit，即由 C#来编写代码）以指定该类型。从那以后，该类型就是强定义的且由编译器（而不是在运行时）检查的了，而且它将作为一个完整的类型存在于代码中。请记住，你无需编写类型定义，因为 C#会帮你计算出来。这一点很重要，因为

在查询语言中，你所请求的以及所获取到的任何特定类型都是根据上下文（即查询结果）来定义的。简单地说，查询结果可能会返回一个之前并没有定义的类型。

这里有一个重要的思想，即你无需编写代码去定义这些特定的类型（C#会帮你定义），因此，你将节约不少的时间。你将省去设计时间、编码时间以及调试时间。微软帮你承担了这些工作。匿名类型就是这样一种能够让你使用任意特定类型的容器。在读完本章之后，你将会掌握运算符的左侧表达式，这也是 LINQ 最重要的内容之一。

此外，每章都将会加入一些其他的但很有帮助的相关知识。本章加入的是有关泛型匿名方法（Generic Anonymous Method[⊖]）的讨论。

1.1　理解匿名类型

通过 var 定义的匿名类型不能等同于 VB 中的 variant。关键字 var 示意编译器应该根据运算符右侧表达式的值来发出一个强类型。匿名类型可以用于诸如整型或字符串之类的简单类型的初始化，不过那将会降低一些代码可读性，而且也得不到多少好处。只有对复合类型（比如由 LINQ 查询所返回的那些）进行初始化的时候，var 才能真正地发挥出它强大的力量。当定义了这样一个匿名类型时，编译器就会发出一个永远不会改变的（只读属性）类。

匿名类型支持智能感知技术（IntelliSense），不过类是不能在代码中引用的，只能引用其成员。

下面的列表给出了使用匿名类型时需要遵守的一些基本规则：

- 匿名类型必须有一个初始化值，而且这个值不能是空值（null），因为类型是根据初始化器推断出来的。
- 匿名类型可以用于简单类型，也可以用于复杂类型。不过，用于定义简单类型时，其价值不大。
- 复合匿名类型需要有成员声明。比如说，var joe = new {Name = " Joe" [, *declaratory = value*, ...]}。（在这个例子中，Name 就是一个成员声明。）
- 匿名类型支持智能感知技术。
- 匿名类型不能用于类的字段。
- 匿名类型可以在 for 循环中用作初始化器。
- 可以使用 new 关键字；数组的初始化器必须使用 new 关键字。
- 匿名类型可以用于数组。
- 所有匿名类型都派生自 Object 类型。
- 方法可以返回匿名类型，不过必须转换成 object，这破坏了强类型原则。
- 初始化匿名类型时可以加上方法，不过可能只有所谓的“语言学家”才会对这种做法产生兴趣。

匿名类型最大的价值在于，可以用它来创建单次使用的元素以及由 LINQ 查询所返回的复合类型，而程序员则无需在静态代码中完整地定义这些类型。也就是说，设计人员可以将注意力放在基本问题域中的类型上，而程序员则可以随意创建一些单次使用的匿名类型（相应的类定义就

⊖ 也有叫做匿名泛型方法（Anonymous Generic Method）的。本书中，这两种叫法都有。——译者注

交给编译器来完成了）。

最后，由于匿名类型是不可修改的（没有属性设置器），因此，两个单独定义的匿名类型，如果它们各个字段的值都相同的话，就可以认为它们是相等的了。

1.2　使用匿名类型

本章将向你介绍匿名类型的所有使用方法，然后再介绍由 LINQ 查询所返回的匿名类型，这里并不会对 LINQ 查询进行完整的讲解。你只需要关注这些匿名类型本身以及要如何使用它们，对于那些查询，随便看看就可以了。

1.2.1　定义简单匿名类型

任何一个简单的匿名类型都需要关键字 var、赋值运算符（=）以及一个非空初始值。赋值运算符左边的名称会被赋值给匿名类型，而 Microsoft 中间语言（Microsoft Intermediate Language，MSIL）所发出的类型则根据赋值运算符的右侧得出。比如：

```
var title = "LINQ Unleashed for C#";
```

上面这句代码使用了匿名类型语法，所赋予的初始值为字符串“LINQ Unleashed for C#”。翻译成 MSIL 的话，它跟下面这条代码是一样的：

```
string title = "LINQ Unleashed for C#";
```

通过中间语言反编译器（Intermediate Language Disassembler，ILDASM）这个实用工具，我们可以发现上面那两条代码所得到的中间语言（Intermediate Language，IL）是一样的（参见图 1-1）。

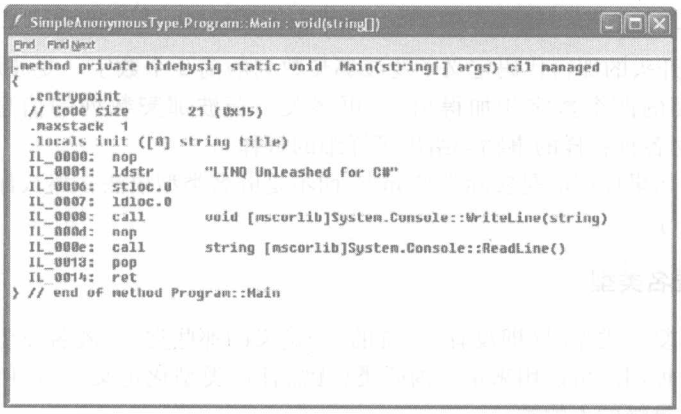

图 1-1　从这段 MSIL 中的 .locals init 语句以及 Console::Write（string）语句上可以看出，title 已经被发出为一个字符串了

其实，声明简单匿名类型没有多少实际意义，Microsoft 允许我们这样做的原因仅仅是为了实现匿名类型的完整性和对称性。在一些关于语言歧义性的争论中，纯粹主义者可能会反对这样做，因为这将会给代码带来一些歧义性。实际上，在类似于上面示例的那些简单应用中，数据类型都是很明显的，根本就不会有任何问题。

1.2.2　使用数组初始化器语法

我们还可以通过匿名类型语法来初始化数组，前提条件是必须使用 new 关键字。例如，示例 1-1 给出了一个简单的控制台应用程序，其中通过斐波那契数列初始化了一个匿名数组。（匿名类型和数组初始化语句用粗体字高亮标出。）

示例 1-1　一个由整型数组初始化的匿名类型

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ArrayInitializer
{
  class Program
  {
    static void Main(string[] args)
    {
      // array initializer
      var fibonacci = new int[]{ 1, 1, 2, 3, 5, 8, 13, 21 };
      Console.WriteLine(fibonacci[0]);
      Console.ReadLine();
    }
  }
}
```

由 var fibonacci 开头的那行代码定义了斐波那契数列的前 8 个数字。斐波那契数列从 1 开始，接下来的数字由前面的两个数字相加得出。（更多关于斐波那契数列的信息请查看 Wikipedia。Wikipedia 很酷，它对各种各样的事物都给出了详细的解释。）

在示例 1-1 中，如果你用的是实际类型 int[ ]而不是匿名类型语法，那么你就不会被牵扯到语言歧义性的争论中去了。

1.2.3　创建复合匿名类型

只有在用于定义复合类型时（即没有“传统的”类定义的那些类），匿名类型才能真正发挥它的作用。匿名类型的这种用法可以用来定义内联类（无需任何类型化定义）。示例 1-2 中所给出的匿名类型定义了一个轻量级的人物类。

示例 1-2　含有两个字段和两个属性的匿名类型，程序员无需编写完整的类型定义代码

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ImmutableAnonymousTypes
```