

本书仅提供部分阅读，如需完整版，请联系QQ: 2011705918

提供各种IT类, 其它类书籍pdf下载，如有需要，请QQ:2011705918

注：链接至淘宝，整理那么多资料也不容易，请多多见谅！非诚勿扰！

[点击购买完整版](#)



果壳中的C#

C# 5.0 权威指南

C# 5.0 IN A NUTSHELL (*The Definitive Reference*)

O'REILLY®



中国水利水电出版社
www.waterpub.com.cn

[美] Joseph Albahari & Ben Albahari 著
陈昇 管学理 曾少宁 杨庆川 译

果壳中的C#——C# 5.0权威指南

本书将引领您达到C#的一个新高度

- 全面覆盖语法、数据类型、变量等基础知识
- 深入覆盖不安全代码、类型转换、预处理指令等高级主题；并发、异步、代码契约、动态编程、安全性、COM互操作性等技术；LINQ相关技术；.NET相关的XML、集合、I/O、网络、存储管理、反射、属性、安全及本地互操作性等技术
- 知识点与案例无缝配合，极大降低学习难度

“本书是我放在桌面上的少数参考书之一。
我推荐你阅读这本书。”

——Scott Guthrie
Microsoft公司副总裁

“无论你是初学者或一个专家，只要你想
提高你的最新异步编程技术，这本书里就
有你需要的内容。”

——Eric Lippert
Microsoft资深软件设计工程师

我们都被束缚在果壳中，却仍自以为是宇宙那无限空间之王

——霍金（引自莎士比亚）

O'REILLY®
oreilly.com.cn

O'Reilly Media, Inc. 授权中国水利水电出版社出版

此简体中文版仅限于在中华人民共和国境内（但不允许在中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

上架建议：编程语言与程序设计/C#

ISBN 978-7-5170-1084-5



定价：118.00元

果壳中的C#——C# 5.0权威指南

[美] Joseph Albahari Ben Albahari 著
陈昇 管学理 曾少宁 杨庆川 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'Reilly Media, Inc. 授权中国水利水电出版社出版



中国水利水电出版社
www.waterpub.com.cn

内 容 提 要

本书是一本C# 5.0的权威技术指南,也是第一本中文版C# 5.0的学习资料。本书通过26章的内容,系统、全面、细致地讲解了C# 5.0从基础知识到各种高级特性的命令、语法和用法。本书的讲解深入浅出,同时为每一个知识点都专门设计了贴切、简单、易懂的学习案例,从而可以帮助读者准确地理解知识点的含义并快速地学以致用。本书与之前的C# 4.0版本相比,还新增了丰富的并发、异步、动态编程、代码精练、安全、COM交互等高级特性相关的内容。

本书还融汇了作者多年在软件开发及C#方面的研究及其实践经验,非常适合作为C#技术的一本通自学教程,亦是一本中高级C#技术人员不可多得的必备工具书。

©2012 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and China Waterpower Press, 2013. Authorized translation of the English edition, 2012 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由O'Reilly Media, Inc. 出版2012。

简体中文版由中国水利水电出版社出版 2013。英文原版的翻译得到O'Reilly Media, Inc.的授权。此简体中文版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc.的许可。

版权所有,未得书面许可,本书的任何部分和全部不得以任何形式重制。

北京市版权局著作权合同登记号:图字:01-2013-4993号

图书在版编目(CIP)数据

果壳中的C# : C# 5.0权威指南 / (美)阿坝哈瑞,
(美)阿坝哈瑞著;陈昇,管学理,曾少宁,杨庆川译.

—北京:中国水利水电出版社,2013.8

书名原文:C# 5.0 in a nutshell, 5e

ISBN 978-7-5170-1084-5

I. ①果… II. ①阿… ②阿… ③陈… ④管… ⑤曾…
⑥杨… III. ①C语言—程序设计 IV. ①TP312

中国版本图书馆CIP数据核字(2013)第171436号

策划编辑:周春元 责任编辑:李 炎 加工编辑:李 燕 封面设计:李 佳

书 名	果壳中的C#——C# 5.0权威指南
作 者	[美] Joseph Albahari Ben Albahari 著
出版发行	陈昇 管学理 曾少宁 杨庆川 译 中国水利水电出版社 (北京市海淀区玉渊潭南路1号D座 100038) 网址: www.waterpub.com.cn E-mail: mchannel@263.net (万水) sales@waterpub.com.cn
经 售	电话: (010) 68367658 (发行部)、82562819 (万水) 北京科水图书销售中心(零售) 电话: (010) 88383994、63202643、68545874 全国各地新华书店和相关出版物销售网点
排 版	北京万水电子信息有限公司
印 刷	北京蓝空印刷厂
规 格	185mm×240mm 16开本 56印张 1950千字
版 次	2013年8月第1版 2013年8月第1次印刷
印 数	0001—2000册
定 价	118.00元

凡购买我社图书,如有缺页、倒页、脱页的,本社发行部负责调换
版权所有·侵权必究

O'Reilly Media, Inc.介绍

O'Reilly Media通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自1978年开始，O'Reilly一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O'Reilly的发展充满了对创新的倡导、创造和发扬。

O'Reilly为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了Make杂志，从而成为DIY革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O'Reilly的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O'Reilly现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项O'Reilly的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O'Reilly Radar博客有口皆碑。”

——Wired

“O'Reilly凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O'Reilly Conference是聚集关键思想领袖的绝对典范。”

——CRN

“一本O'Reilly的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照Yogi Berra的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去Tim似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal



前言

C# 5.0是微软旗舰编程语言的第4次重大升级，大大提升了C#语言的灵活性与功能。一方面，它实现了一些高级抽象，如查询表达式和异步延续；另一方面，它又通过自定义类型值和可选指针等设计实现了一些底层功能。

这部分增加的特性尤其值得学习。虽然诸如Microsoft的IntelliSense工具和各种在线参考文档在帮助你完成工作方面是非常好用的，但是它们需要由现有的一些概念知识来支撑。本书以简明统一的方式（而非繁杂冗长的介绍）准确到位地阐述了这些知识集。

本书是完全按照概念和用例组织的，因此无论是按顺序阅读还是随意浏览都可以。虽然只要求具备基本的背景知识，但它还是有一定的深度，因此比较适合中高级水平的读者阅读。

本书内容涵盖了C#、CLR和Framework程序集。我们之所以做出这样的选择，是为了重点讲解一些较难理解的主题，如并发性、安全性和应用程序域，同时不影响深度或可读性。C# 5.0及相关Framework的新特性已经被标注清楚，因此也可以将本书作为C# 4.0参考书使用。

目标读者

本书主要针对中高级开发人员。不要求读者具备C#知识，但是需要有一些普通编程经验。对于初学者，本书能够补充教程类编程介绍书籍，但不能替代教程类书籍。

熟悉C# 4.0的读者会发现，我们重写了关于并发性的小节，其中包括深入介绍C# 5.0的异步函数及其相关类型，并且还介绍了异步编程的原则，以及它如何能够提供效率和线程安全性。

本书是各种介绍实用技术图书的理想伴侣，如WPF、ASP.NET或WCF。这些书籍所省略的语言与.NET Framework方面的内容，本书都进行了详细介绍，反之亦然。

这本书并不会详细介绍每一种.NET Framework技术。此外，这本书也不会介绍平板电脑或Windows Phone开发的专用API。

本书的结构

本书前三章集中介绍C#语言，先介绍语法、类型和变量，然后介绍一些高级特性，如不安全代码和预处理指令。如果你是初学者，应该循序渐进地阅读这些章节。

其余各章的内容涵盖核心.NET Framework，包括LINQ、XML、集合、I/O与网络、内存管理、反射、动态编程、属性、安全性、并发、应用域和原生互操作性等主题。除了第6章和第7章之外，你

可以按任意顺序阅读，因为这两章是后续主题的基础。关于LINQ的三章内容最好也按顺序阅读。一些章节要求读者理解并发的基础知识，这部分知识将在第14章介绍。

使用本书所需的其他材料

本书的例子需要使用C# 5.0编译器和微软.NET Framework 4.5。此外，微软的.NET文档可以帮助查找各个类型及其成员（在线版本）。

虽然在记事本中可以编写源代码和从命令执行编译器，但是为了提高效率，最好使用一个代码编辑器即时测试各个代码版本，并且使用集成开发环境（IDE）生成可执行程序 and 库。

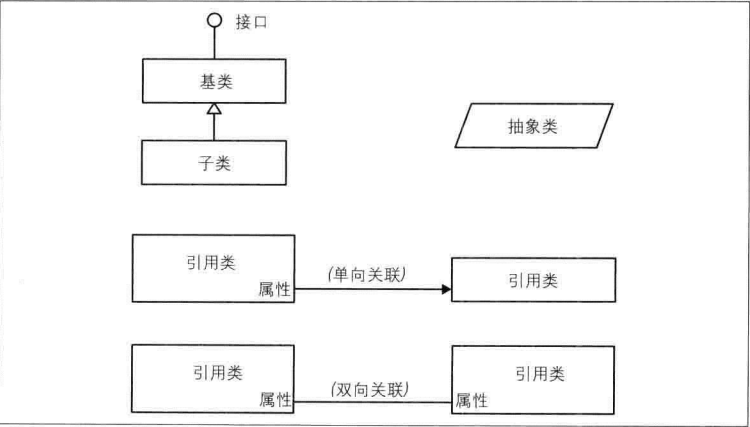
推荐从www.linqpad.net下载LINQPad 4.40或更高版本（免费）作为代码编辑器。LINQPad完全支持C# 5.0，并且由C# 5.0的作者之一维护。

对于IDE，建议下载Microsoft Visual Studio 2012：除了免费的简洁版，其他版本都适合本书介绍的内容。

提示：第2～10章及并发性、并行编程和动态编程等章节所列代码清单都是可交互（可编辑）的LINQPad示例。

本书中使用的约定

本书使用简单的UML符号来说明类之间的关系，如图P-1所示。斜矩形表示抽象类；圆圈表示一个接口。带空心三角形的线条表示继承，其中三角形指向基类。带箭头的线条表示单向关联；不带箭头的线条表示双向关联。



图P-1：示例图

本书还使用以下的排版约定：

斜体 (*Italic*)

表示URI、文件名、目录和应该由用户提供的值所替代的文本

等宽字体 (Constant Width)

表示C#代码、关键字与标识符以及程序输出

等宽粗体 (Constant Width Bold)

突出显示部分代码

使用示例代码

本书的作用是帮助你完成工作。一般而言，你可能会在程序和文档中使用本书所提供的代码。除非必须复制大部分代码，否则不需要联系我们获得授权。例如，你不需要授权就可以使用本书的多个代码段来编写程序；销售或分发O'Reilly书籍中的示例代码CD-ROM也不需要授权；引用本书及其示例代码来回答某个问题也不需要授权；将本书的大量示例应用到你的产品文档中也不需要授权。

我们欢迎你标注内容出处，但不强制要求。一般的标注通常包括书名、作者、出版社和ISBN。例如：“C# 5.0技术手册，作者：Joseph Albahari和Ben Albahari。版权所有 2010 Joseph Albahari和Ben Albahari，ISBN：978-1-449-32010-2”。

如果你认为你的代码示例使用方式超出一般用途或超出了此处的授权范围，请随时与我们联系：permissions@oreilly.com。

Safari® Books Online

Safari Books Online是一个按需供应的数字图书馆，你可以轻松搜索到7500多种技术和创意参考图书与视频，可以快速帮助你找到问题答案。

订阅后，你可以在图书馆中在线阅读任何页面和观看任何视频。你还可以在手机和移动设备上阅读这些图书。你可以查看未出版的新书，唯一地访问仍在编写中的书稿以及给作者发送反馈信息。你还可以复制和粘贴代码示例、整理收藏夹、下载章节、收藏关键章节、编写注解、打印内容页，以及使用无数其他可以节约时间的特性。

O'Reilly Media已经将本书上传到Safari Books Online服务。想要获得本书或O'Reilly及其他出版商的类似书籍的完整电子版，请先免费注册一个账号：<http://my.safaribooksonline.com>。

联系我们

对于本书，如果有任何意见或疑问，请按照以下地址联系本书出版商：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）
奥莱利技术咨询（北京）有限公司

本书也有相关的网页，我们在上面列出了源代码、范例以及其他一些信息。你可以访问：

<http://www.albahari.com/nutshell/> (英文版)

对本书做出评论或者询问技术问题, 请发送E-mail至:

bookquestions@oreilly.com

希望获得关于本书、会议、资源中心和O'Reilly网络的更多信息, 请访问:

<http://www.oreilly.com>

<http://www.oreilly.com.cn>

致谢

Joseph Albahari

首先, 我要感谢我的兄弟和合著者Ben Albahari, 感谢他在最初说服我参与这项后来非常成功的项目。我非常享受与Ben一起探究难题的过程: 他不仅与我一样勇于向传统观点提出质疑, 而且都具有刨根问底的精神。

我最希望感谢的还有一些优秀的技术审阅者。首先是来自Microsoft的校阅者, Stephen Toub (并行编程团队) 和Chris Burrows (C#编译器团队) 提供的大量信息显著地增强了关于并发性、动态编程和C#语言等章节的内容。从CLR团队, 我收获了来自Shawn Farkas、Brian Grunkemeyer、Maoni Stephens和David DeWinter关于安全性和内存管理方面的非常宝贵的信息。

我极力向读者推荐Jon Skeet (《C# in Depth》的作者以及堆栈溢出的专家), 他的许多宝贵建议丰富了许多章节的内容 (虽然任职于谷歌公司, 但是我们尊重他的选择!)。我也同样感激C# MVP Nicholas Paldino敏锐的眼光, 他发现了其他工作人员未发现的一些错误和疏忽。我同时还要感谢另外两位C# MVP: Mitch Wheat和Brian Peek, 以及本书所基于的3.0版本的校阅者。这里面包括了前面提到的Nicholas Paldino, 他将其博大渊深的知识应用到了本书的大多数章节, 以及Krzysztof Cwalina、Matt Warren、Joel Pobar、Glyn Griffiths、Ion Vasilian、Brad Abrams、Sam Gentile和Adam Nathan。

最后, 我还要感谢O'Reilly团队, 包括行动迅速及非常高效的编辑Laurel Ruma、宣传人员Kathryn Barrett、文字编辑Audrey Doyle以及我的家人Miri和Sonia。

Ben Albahari

由于我的兄弟在我之前写下了他的感言, 他所表达的大多数内容也正是我的肺腑之言: 事实上, 当我们还是孩子的时候, 就已经开始编写程序了 (我们共用一台Apple IIe; 他编写他自己的操作系统, 而我则是在编写我的Hangman), 因此, 现在我们能一起撰写这些书籍是一件非常惬意的事。我希望我们在此书中所浓缩的经验可以丰富读者们的编程经验。

同时, 我还要感谢我之前在Microsoft工作时的同事。很多人在那里工作, 他们不仅智商高而且情商更高, 我怀念与他们共事的时光。我还要特别感谢Brian Beckman, 从他的身上我学到了很多。



目录

前言

目标读者	1
本书的结构	1
使用本书所需的其他材料	2
本书中使用的约定	2
使用示例代码	3
联系我们	3
Safari® Books Online	4
致谢	4

第1章 C#和.NET Framework简介1

1.1 面向对象	1
1.2 类型安全性	1
1.3 内存管理	2
1.4 平台支持	2
1.5 C#与CLR的关系	2
1.6 CLR和.NET Framework	3
1.7 C#与Windows Runtime	4
1.8 C# 5.0新特性	5
1.9 C# 4.0新特性	5
1.10 C# 3.0新特性	5

第2章 C#语言基础7

2.1 第一个C#程序	7
2.2 语法	9
2.3 类型基础	11

2.4 数值类型	19
2.5 布尔类型和运算符	25
2.6 字符串和字符	27
2.7 数组	29
2.8 变量和参数	32
2.9 表达式和运算符	40
2.10 语句	43
2.11 命名空间	51
第3章 在C#中创建类	57
3.1 类	57
3.2 继承	69
3.3 object类型	76
3.4 结构体	80
3.5 访问权限修饰符	81
3.6 接口	83
3.7 枚举类型	87
3.8 嵌套类型	91
3.9 泛化	92
第4章 C#高级特性	103
4.1 委托	103
4.2 事件	111
4.3 Lambda表达式	117
4.4 匿名方法	120
4.5 try语句和异常	121
4.6 枚举类型和迭代	128
4.7 可空类型	132
4.8 运算符重载	137
4.9 扩展方法	140
4.10 匿名类型	143
4.11 动态绑定	144
4.12 属性	151
4.13 调用者信息属性 (C# 5)	152
4.14 不安全代码和指针	154

4.15 预处理指令	157
4.16 XML文档	159

第5章 框架概述163

5.1 CLR和核心框架	165
5.2 应用技术	168

第6章 框架基础174

6.1 字符串与文本处理	174
6.2 日期和时间	185
6.3 日期与时区	191
6.4 标准格式字符串与解析标记	202
6.5 其他转换机制	208
6.6 全球化	211
6.7 操作数字	212
6.8 枚举类型	216
6.9 元组	219
6.10 Guid结构体	220
6.11 等值比较	220
6.12 顺序比较	229
6.13 实用类	232

第7章 集合235

7.1 枚举	235
7.2 ICollection和IList接口	242
7.3 Array类	245
7.4 复制	251
7.5 List、Queue、Stack和Set	252
7.6 字典	259
7.7 可定制的集合和委托	264
7.8 等值和顺序插入	270

第8章 LINQ查询277

8.1 入门	277
8.2 运算符流语法	279
8.3 查询表达式	285

8.4 延迟执行	289
8.5 子查询	295
8.6 LINQ构造方式	298
8.7 映射策略	301
8.8 解释型的查询	303
8.9 LINQ to SQL 和 Entity Framework	309
8.10 查询表达式的创建	323
第9章 LINQ运算符	328
9.1 概述	329
9.2 筛选	332
9.3 映射	336
9.4 连接	347
9.5 Zip 运算符	355
9.6 排序	355
9.7 Grouping	358
9.8 集合运算符	361
9.9 转换方法	363
9.10 元素运算符	365
9.11 聚合方法	367
9.12 数量词	372
9.13 生成集合的方法	373
第10章 LINQ to XML	375
10.1 架构概述	375
10.2 X-DOM概述	376
10.3 实例化X-DOM	379
10.4 指定内容	380
10.5 导航和查询	381
10.6 更新X-DOM	386
10.7 使用Value	389
10.8 文档和声明	391
10.9 名称和命名空间	394
10.10 注解	400
10.11 将数据映射到X-DOM	400

第11章 其他XML技术	407
11.1 XmlReader.....	407
11.2 XmlWriter.....	415
11.3 使用XmlReader/XmlWriter的模式	417
11.4 XmlDocument.....	421
11.5 XPath.....	424
11.6 XSD和模式验证	428
11.7 XSLT.....	431
第12章 销毁和垃圾回收	432
12.1 IDisposable接口、Dispose方法和Close方法	432
12.2 自动垃圾回收	436
12.3 终止器	438
12.4 垃圾回收器如何工作.....	442
12.5 托管内存泄露	445
12.6 弱引用	448
第13章 诊断和代码契约	452
13.1 条件编译.....	452
13.2 Debug和Trace类	455
13.3 代码契约概述	458
13.4 先决条件.....	461
13.5 后置条件.....	465
13.6 断言和对象不变式.....	467
13.7 接口和抽象方法中的契约	468
13.8 处理契约错误	469
13.9 选择性执行契约	471
13.10 静态契约检查	472
13.11 调整器集成.....	473
13.12 进程和处理线程	474
13.13 StackTrace和StackFrame类.....	475
13.14 Windows事件日志	476
13.15 性能计数器	478
13.16 Stopwatch类.....	483
第14章 并发与异步	484
14.1 简介	484

14.2 线程处理.....	485
14.3 任务.....	498
14.4 异步原则.....	506
14.5 C# 5.0的异步函数.....	510
14.6 异步模式.....	523
14.7 旧模式	530

第15章 流与I/O533

15.1 流体系结构.....	533
15.2 使用流	534
15.3 流适配器.....	546
15.4 压缩流	553
15.5 操作Zip文件	555
15.6 文件与目录操作	555
15.7 Windows Runtime中的文件输入/输出.....	565
15.8 内存映射文件	567
15.9 隔离存储区.....	569

第16章 网络575

16.1 网络体系结构.....	575
16.2 地址与端口	577
16.3 URI.....	578
16.4 客户端类.....	579
16.5 HTTP访问.....	591
16.6 编写HTTP服务器	596
16.7 使用FTP	599
16.8 使用DNS	600
16.9 通过SmtpClient发送邮件.....	601
16.10 使用TCP	602
16.11 使用TCP接收POP3邮件	605
16.12 在Windows Runtime中建立TCP连接.....	606

第17章 序列化.....608

17.1 序列化概念.....	608
17.2 数据契约的序列化.....	611
17.3 数据契约与集合	620

17.4 扩展数据契约	622
17.5 二进制序列化器	625
17.6 二进制序列化属性	627
17.7 使用ISerializable进行二进制序列化	629
17.8 XML序列化	632

第18章 程序集.....641

18.1 程序集包含的内容	641
18.2 强名称和程序集签名	645
18.3 程序集名称	647
18.4 Authenticode签名	649
18.5 全局程序集高速缓存	652
18.6 资源和卫星程序集	654
18.7 解析和加载程序集	661
18.8 在基础文件夹外部署程序集	665
18.9 打包单个可执行文件	667
18.10 处理未引用的程序集	668

第19章 反射和元数据.....670

19.1 反射和激活类型	670
19.2 反射和调用成员	677
19.3 反射程序集	688
19.4 使用属性	689
19.5 动态生成代码	694
19.6 发出程序集和类型	700
19.7 发出类型成员	703
19.8 发出泛型方法和类型	708
19.9 复杂的发出目标	710
19.10 解析IL	713
19.11 编写反编译器	713

第20章 动态编程.....718

20.1 动态语言运行时	718
20.2 数字类型统一	719
20.3 动态成员重载解决方案	720

20.4 实现动态对象	726
20.5 通过动态语言交互操作	729

第21章 安全731

21.1 权限	731
21.2 代码访问安全 (CAS)	734
21.3 允许部分可信的调用程序	737
21.4 CLR 4.0中的透明模型	739
21.5 沙箱化程序集	746
21.6 操作系统安全	749
21.7 身份和角色安全	751
21.8 加密综述	752
21.9 Windows数据保护	753
21.10 散列法	754
21.11 对称加密	755
21.12 公共密钥加密和签名	759

第22章 高级线程763

22.1 同步概述	763
22.2 排他锁	764
22.3 锁与线程安全性	771
22.4 非排他锁	775
22.5 使用事件等待处理器发送信号	780
22.6 Barrier类	787
22.7 延后初始化	788
22.8 线程本地存储	790
22.9 Interrupt和Abort	792
22.10 Suspend和Resume	793
22.11 定时器	793

第23章 并行编程797

23.1 PFX	797
23.2 PLINQ	799
23.3 Parallel类	810
23.4 任务并行	816

23.5 处理AggregateException异常	825
23.6 并发集合.....	827
23.7 BlockingCollection<T>.....	829

第24章 应用域.....833

24.1 应用域架构.....	833
24.2 创建和销毁应用域.....	833
24.3 多应用域的使用	836
24.4 DoCallBack的应用	837
24.5 应用域的监视	838
24.6 应用域和线程	838
24.7 应用域间通信	839

第25章 本地化和COM组件交互.....844

25.1 调用本地库.....	844
25.2 类型封送.....	845
25.3 非托管代码的回调函数	847
25.4 模拟C共用体	848
25.5 内存共享.....	849
25.6 映射结构体到非托管内存区.....	851
25.7 COM交互	854
25.8 在C#中调用COM对象	856
25.9 内嵌互操作类型	859
25.10 主互操作程序集	859
25.11 COM中调用C#对象	860

第26章 正则表达式.....861

26.1 正则表达式基础	861
26.2 量词.....	865
26.3 零宽度断言	866
26.4 分组	869
26.5 文本替换和拆分	870
26.6 正则表达式实例	871
26.7 正则表达式语言参考	874



C#和.NET Framework简介

C#是一种通用的类型安全且面向对象的编程语言。这种语言的目标是提高程序员的生产力，为此，需要在简单性、可表达性和性能之间实现平衡。C#语言的首席架构师从第一个版本开始就是Anders Hejlsberg（Turbo Pascal的发明者和Delphi架构师）。C#语言与平台无关，但是它能够很好地与Microsoft .NET Framework协同工作。

1.1 面向对象

C#实现了面向对象编程的广泛特性，包括封装、继承和多态。封装表示在对象周围创建一个边界，将它的外部（公开）行为与内部（私有）实现细节隔离。C#在面向对象方面的特性包括：

统一的类型系统

C#中的基础构建块是一种被称为类型的数据与函数的封装单元。C#有一个统一的类型系统，其中所有类型最终都共享一个公共的基类。这意味着所有的类型，不管它们是表示业务对象，或者像数字等基本类型，都共享相同的基本功能集。例如，任何类型都可以通过调用它的ToString方法转换为一个字符串。

类与接口

在纯粹的面向对象泛型中，唯一的类型就是类。但是C#中还有其他几种类型，其中一种是接口（类似于Java中的接口）。接口与类相似，但它只是某种类型的定义，而不是实现。在需要使用多继承时，它是非常有用的（与C++和Eiffel等语言不同，C#不支持类的多继承）。

属性、方法与事件

在纯粹的面向对象泛型中，所有函数都是方法（Smalltalk中就是这样）。在C#中，方法只是一种函数成员，也包含一些属性和事件以及其他组成部分。属性是封装了一部分对象状态的函数成员，如按钮的颜色或标签的文本。事件是简化对象状态变化处理的函数成员。

1.2 类型安全性

C#首先是一种类型安全的语言，这意味着类型只能够通过它们定义的协议进行交互，从而保证每一种类型的内部一致性。例如，C#不允许将字符串类型作为整型进行处理。

更具体地说，C#支持静态类型化，这意味着这种语言会在编译时执行静态类型安全性检查。另外一

种是动态类型安全性，.NET CLR在运行时执行动态类型安全性检查。

静态类型化能够在程序运行之前去除大量的错误。它将大量的运行时单元测试转移到编译器中，验证程序中所有类型之间都是相互适合的。这样大型程序就更容易管理、更具可预测性和健壮性。而且，静态类型化使一些诸如Visual Studio 的IntelliSense等工具有助于编写程序，因为它知道某个特定变量的类型是什么，因此也知道能够调用哪些方法来处理这个变量。

提示：C#允许部分代码通过新的dynamic关键字来动态指定类型。然而，C#在大多数情况下仍然是一种静态类型化语言。

C#之所以被称为一种强类型语言，是因为它的类型规则（以静态或动态的方法执行）是非常严格的。例如，不能够使用一个浮点型参数来调用一个定义时接收整数的函数，除非显式地将这个浮点数转换为整数。这有助于防止编码错误。

强类型也是C#代码能够在沙箱中运行的原因之一。沙箱指的是安全性的所有方面都由主机控制的一种环境。在沙箱中，一定要注意不能随意忽略一个对象的类型规则从而破坏其状态。

1.3 内存管理

C#依靠运行时环境来执行自动的内存管理。CLR有一个垃圾回收器，它是作为程序一部分运行的，负责回收不再被引用的对象所占用的内存。这让程序员不需要显式地释放为对象分配的内存，从而避免了诸如C++等语言中错误使用指针造成的内存问题。

C#并没有去除指针：它只是使大多数编程任务不需要使用指针。对于性能至关重要的热点和互操作性方面，还是可以使用指针，但是只允许在显式标记为不安全的代码块中使用。

1.4 平台支持

C#一般用来编写运行在Windows平台的代码。虽然Microsoft通过ECMA实现了C#语言和CLR的标准化，但是专门用来支持非Windows平台C#的资源（包括Microsoft内部和外部的）总量相对较少。这意味着，如果很注重多平台支持，那么诸如Java等语言可能是更明智的选择。因此，C#可在以下情况用于编写跨平台代码：

- C#代码运行在服务器上，生成可运行在任意平台的DHTML。这正是ASP.NET采用的方法。
- C#代码运行在一个非Microsoft Common Language Runtime的运行时环境中。最典型的例子是Mono项目，它具有自己的C#编译器和运行时环境，可运行在Linux、Solaris、Mac OS X和Windows上。
- C#代码运行在一台支持Microsoft Silverlight（支持Windows和Mac OS X）的主机上。这是一种与Adobe Flash Player类似的新技术。

1.5 C#与CLR的关系

C#依赖于一个运行时环境，它包括许多特性，如自动内存管理和异常处理。C#的设计与CLR的设计非常接近，CLR提供了这些运行时特性（但C#技术上不依赖于CLR）。而且C#的类型系统与CLR的

类型系统也非常接近（例如，都共享相同的基础类型定义）。

1.6 CLR和.NET Framework

.NET Framework由名为Common Language Runtime (CLR) 的运行时环境和大量的程序库组成。这些程序库由核心库（本书主要介绍）和应用库组成，应用库依赖于核心库。图1-1是这些程序库的可视化概况（也可以作为本文导航辅助图）。

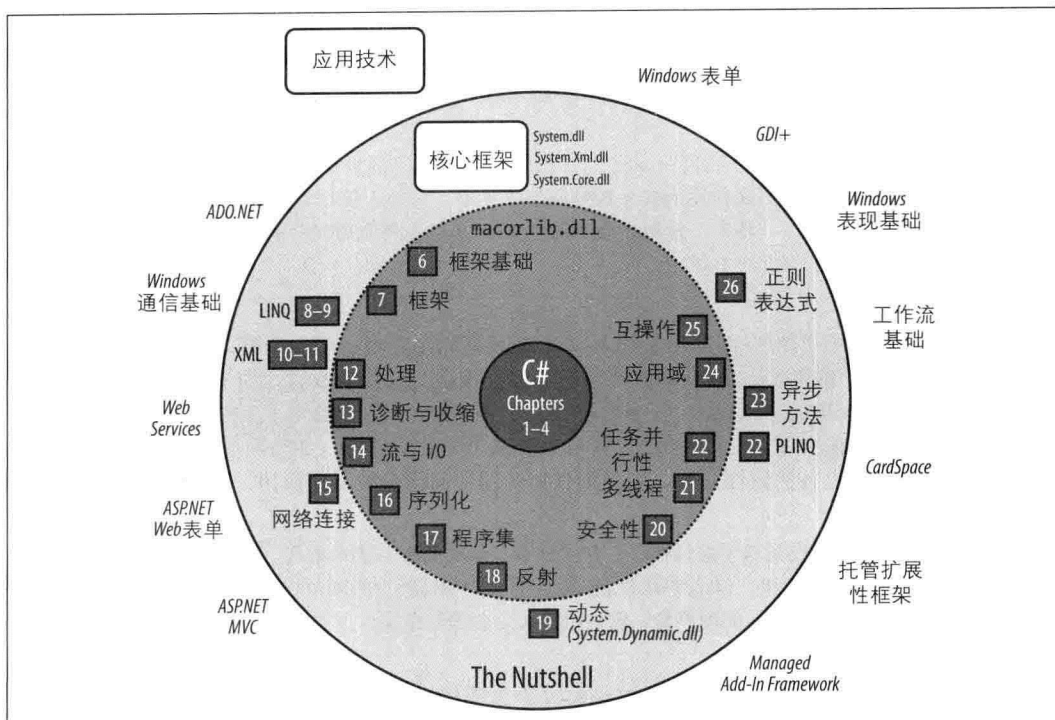


图1-1：本图说明了本文涉及的主题及其所在章节。超出本书范围的特殊框架和类库名称位于The Nutshell的边界之外

CLR是执行托管代码的运行时环境。C#是几种将源代码编译为托管语言之一。托管代码会被打包成程序集，它可以是可执行文件（.exe文件）或程序库（.dll）的形式，包括类型信息或元数据。

托管代码用Intermediate Language或IL表示。当CLR加载一个程序集时，它会将IL转换为该主机的原生代码，如x86。这个转换是通过CLR的JIT（Just-In-Time）编译器执行的。程序集几乎保留了所有源语言的结构，这使它很容易检测，也很容易动态生成代码。

提示：Red Gate的.NET Reflector是一个重要的分析程序集内容的工具（你可以将它作为反编译器使用）。

CLR是无数运行时服务的主机。这些服务包括内存管理、程序库加载和安全性服务。CLR是与语言无关的，它允许开发人员用多种语言（例如C#、Visual Basic .NET、Managed C++、Delphi.NET、Chrome .NET和J#）开发应用程序。

.NET Framework由只支持基于所有Windows平台或Web的应用程序的程序库组成。第5章概括介绍了.NET Framework程序库。

1.7 C#与Windows Runtime

C# 5.0还实现了Windows Runtime (WinRT) 库的互操作。WinRT是一个扩展接口和运行时环境，它可以用面向对象和与语言无关的方式访问库。Windows 8带有这个运行时库，属于Microsoft组件对象模型或COM的扩展版本（参见第25章）。

Windows 8带有一组非托管WinRT库，它是通过Microsoft应用商店交付的支持触摸屏的Metro风格应用程序框架（术语WinRT也指这些库）。作为WinRT，这些程序库不仅可以通过C#和VB访问，也可以通过C++和JavaScript访问。

警告： 有一些WinRT库也可以用在普通的非平板机应用程序中。然而，使用WinRT要求应用程序的最低操作系统是Windows 8。（将来，使用新版本的WinRT甚至会将应用程序的最低操作系统要求提升到Windows 9。）

WinRT库支持新的Metro用户界面（用于编写沉浸式触摸优先应用程序）、移动设备专属特性（传感器、短信等）以及一系列核心功能（与.NET框架部分重叠）。由于功能上存在重叠，所以Visual Studio包含了一个Metro项目参考模板（一组.NET参考程序集），它们隐藏了.NET框架中与WinRT重叠的部分。这个模板还隐藏了.NET框架中大量与平板应用无关的部分（如访问数据库）。Microsoft的应用商店控制着消费设备的软件分发，它拒绝任何试图访问隐藏类型的应用。

提示： 参考程序集的存在仅仅是为了编译程序，并且只有有限的类型与成员集合。因此，开发者可以在计算机上安装完整的.NET框架，但是采用未完全安装的方式编写特定的项目。实际的运行时功能来自全局程序集缓存（参见第18章）的程序集，它们可能大于参考程序集。

隐藏大部分.NET框架可以简化Microsoft平台新开发者的学习过程。除此之外，还有两个更重要的目标：

- 它将应用程序限制在沙箱中（限制功能，减少恶意软件危害）。例如，禁止任意文件访问操作，以及严格限制启动或访问计算机中其他程序的行为。
- 它允许在只支持Metro的低功耗平板电脑上运行.NET框架（Metro模板），降低操作系统资源需求。

WinRT与普通COM的区别是，WinRT的程序库支持多种语言，包括C#、VB、C++和JavaScript，所以每一种语言（几乎）都将WinRT类型视为自己的专属类型。例如，WinRT将调整大小写规则，使之符合目标语言的要求，甚至还会重新映射一些函数与接口。WinRT程序集还在.winmd文件中包含丰富的元数据，其格式与.NET程序集文件相同，不需要特殊处理就可以无缝对接。事实上，除了命名空间存在区别之外，开发者甚至不知道使用的是WinRT，而非.NET类型。（此外，WinRT类型遵循COM风格限制，例如，它们只支持有限的继承和泛型。）

提示： WinRT/Metro并不会取代整个.NET框架。仍然推荐（和必须）将后者用于标准桌面和服务器端开发，而且具有以下优势：

- 程序不仅限于运行在沙箱中。
- 程序可以使用整个.NET框架和第三方库。
- 应用程序分发不依赖于Windows应用商店。
- 应用程序可以使用最新版框架，但是不要求用户安装最新版的操作系统。

1.8 C# 5.0新特性

C# 5.0两个较大的新特性是通过两个关键字（`async`和`await`）支持异步功能（`asynchronous function`）。异步功能支持使用异步延续（`asynchronous continuation`），从而可以简化快速响应和线程安全富客户端应用程序的编写。它们还有利于编写高度并发和高效I/O密集型应用程序，不需要为每个操作绑定一个线程资源。

第14章将详细介绍异步功能。

1.9 C# 4.0新特性

C# 4.0增加的新特性有：

- 动态绑定
- 可选参数和命名参数
- 用泛型接口和代理实现类型变化
- 改进COM互操作性

动态绑定（第4章和第20章）将绑定过程（解析类型与成员的过程）从编译时延迟到运行时，这种方法适用于一些需要避免使用复杂反射代码的场合。动态绑定还适合用于实现动态语言和COM组件的互操作。

可选参数（第2章）允许函数指定参数的默认值，这样调用者就可以省略一些参数，而命名参数则允许函数的调用者按名字指定参数，而非按位置指定。

类型变化规则在C# 4.0中很宽松（第3~4章），因此泛型接口和泛型代理类型参数可以标记为`covariant`或`contravariant`，从而支持更多自然类型转换。

COM互操作性（第25章）在C# 4.0中进行了3个方面的改进。第一，参数可以通过引用传递，而不需要使用`ref`关键字（特别适用于与可选参数一同使用）。第二，包含COM interop类型的程序集可以链接，而不需要引用。链接的interop类型支持类型等值转换，从而不需要使用主互操作程序集，并且解决了版本控制和部署的难题。第三，返回来自链接interop类型的COM变体类型的函数可以映射到`dynamic`，而非映射为`object`，从而不需要进行强制转换。

1.10 C# 3.0新特性

C# 3.0增加的这些特性主要集中在语言集成查询功能上（*Language Integrated Query*，简称LINQ）。LINQ支持在C#程序中直接编写查询和以静态方式检查其正确性，并且可以同时查询本地集合（如列表或XML文档）或远程数据源（如数据库）。C# 3.0中用于支持LINQ的新特性还包括隐式类型化局

部变量、匿名类型、对象构造器、Lambda表达式、扩展方法、查询表达式和表达式树。

隐式类型化局部变量（var关键字，第2章）允许在声明语句中省略变量类型，然后由编译器推断其类型。这样可以简化代码和支持匿名类型（第4章），这是一些即时创建的类，它们常用于LINQ查询的最终输出结果中。数组也可以隐式类型化（第2章）。

对象构造器（第3章）允许在调用构造函数之后以内联方式设置属性，从而简化了对象的构造过程。对象构造器同时支持命名和匿名类型。

Lambda表达式（第4章）介绍了由编译器即时创建的微型函数，并且特别适合用于创建“流畅的”LINQ查询（第8章）。

扩展方法（第4章）使用新方法扩展现有的类型（而不需要修改类型定义），使静态方法变得像实例方法一样。

查询表达式（第8章）提供了用于编写LINQ查询的更高级语法，从而可以大大简化操作多个序列或范围变量的LINQ查询。

表达式树（第8章）介绍了描述了分配给特殊类型Expression<TDelegate>的lambda表达式的代码DOM（文档对象模型）。表达式树使LINQ查询能够远程执行（例如在数据库服务器上），因为它们可以在运行时转换和翻译（例如变成SQL语句）。

C# 3.0也增加了自动化和局部方法

自动化属性（第3章）详细讲解一种编写属性的方法，它将私有域的get/set操作进行简化，将它们交给编译器自动执行。局部方法（第3章）让一个自动生成的局部类提供可定制的钩子函数，从而不需要手工编写，而且它会在不使用时“消失”。



在本章中，我们将介绍一些C#语言的基础知识。

提示：在本章和接下来的两章中，所有的程序和代码片段都可以作为可交互的例子在LINQPad中使用。阅读本书时学习这些例子可以加快你的学习进度，编辑这些例子，可以立即看到结果，而不需要像在Visual Studio中那样建立项目和解决方案。

要下载这些例子，请点击LINQPad中的Samples选项卡，然后点击“Download more samples”。LINQPad是免费程序，详见<http://www.linqpad.net>。

2.1 第一个C#程序

这是一个计算12乘以30，并把结果360打印到屏幕上的程序。双斜线“//”表示其后的内容是注释。

```
using System;                // 导入命名空间

class Test                    // 类声明
{
    static void Main()        // 方法声明
    {
        int x = 12 * 30;      // 语句1
        Console.WriteLine(x); // 语句2
    }                          // 方法结束
}                              // 类结束
```

在C#中语句按顺序执行。每个语句都以分号(;)结尾。该程序的核心是以下两个语句。

```
int x = 12 * 30;
Console.WriteLine(x);
```

C#语句按顺序执行，以分号(或后面将介绍的代码块)结尾。

第1个语句计算表达式12*30的值，并把结果存储到整型局部变量x中。第2个语句调用Console类的WriteLine方法，把变量x的值打印到屏幕上的文本窗口中。

方法是执行一系列语句的行为。这些语句叫做语句块。语句块由一对大括号中的0个或多个语句组成。下面我们定义一个名为Main的方法：

```
static void Main()
{
    ...
}
```

编写可调用低级函数的高级函数可以简化程序。我们可以通过一个可重用的方法重构程序，计算某个整数乘以12的值：

```
using System;

class Test
{
    static void Main()
    {
        Console.WriteLine (FeetToInches (30)); // 结果: 360
        Console.WriteLine (FeetToInches (100)); // 结果: 1200
    }

    static int FeetToInches (int feet)
    {
        int inches = feet * 12;
        return inches;
    }
}
```

方法可以通过参数来接收调用者输入的数据，并通过返回类型给调用者返回输出数据。现在定义一个FeetToInches方法，该方法有一个用于输入英尺的参数和一个用于输出英寸的返回类型：

```
static int FeetToInches (int feet) {...}
```

上例中的字面值30和100是传递给FeetToInches方法的参数。例子中的Main方法后面只有括号，因为它没有参数，用void是因为它不给调用者返回任何值。

```
static void Main()
```

C#把Main方法作为程序的默认执行入口。Main方法也可以返回一个整型（而不是void），从而为程序执行的环境返回一个值。Main方法也可以接受一个字符串数组作为参数（数组中包含可传递给可执行内容的任何参数）。例如：

```
static int Main (string[] args) {...}
```

提示： 数组（例如string[]）代表某种特定类型，固定数量的元素的集合。数组由元素类型和它后面的方括号指定，将在“数组”小节中介绍。

方法是C#中的一种函数，另一种函数是我们用来执行乘法运算的*运算符。还有构造方法、属性、事件、索引器和终结器。

在我们的例子中，将两个方法组合到一个类中。类由函数成员和数据成员组成，形成面向对象的构建块。Console类的组成员处理命令行输入/输出功能，如WriteLine方法。我们的Test类由Main方法

和FeetToInches方法组成。类也是一种类型，我们将在“类型基础”小节中介绍它。

在程序的最外层，类型被组织到命名空间中。Using指令用来使System命名空间在我们的应用程序中有效，以使用Console类。我们能够在TestPrograms命名空间中定义我们所有的类，例如：

```
using System;

namespace TestPrograms
{
    class Test {...}
    class Test2 {...}
}
```

.NET Framework的组织方式为嵌套的命名空间。例如，以下命名空间中包含着处理文本的类型：

```
using System.Text;
```

这里的using指令仅仅是为了方便，也可以用命名空间+类型名（例如System.Text.StringBuilder）这种完全限定名称来引用某种类型。

编译

C#编译器把一系列.cs扩展名的源代码文件编译成程序集。程序集是.NET中的最小打包和部署单元。一个程序集可以是一个应用程序，或者是一个库。一个普通的控制台程序或Windows应用程序是一个.exe文件，包含一个Main方法。一个库是一个.dll文件，它相当于一个没有入口的.exe文件。库是用来被应用程序或其他库调用（引用）的。.NET Framework就是一组库。

C#编译器的名称是csc.exe。可以使用像Visual Studio这样的IDE编译C#程序，也可以在命令行中手动调用csc命令编译C#程序。要手动编译C#程序，首先将程序保存成文件（例如MyFirstProgram.cs），然后进入命令行并按照下面的方式调用csc命令（csc位于%SystemRoot%\Microsoft.NET\Framework\<framework-version>下，其中%SystemRoot%指Windows目录）：

```
csc MyFirstProgram.cs
```

这个命令将生成名为MyFirstProgram.exe的应用程序。

要生成库（.dll），使用如下方式：

```
csc /target:library MyFirstProgram.cs
```

提示：我们将在第18章详细介绍程序集。

2.2 语法

C#的语法是基于C和C++语法的。在本节中，我们将用下面的程序介绍C#的语法元素。

```
using System;

class Test
{
    static void Main()
```



```

{
    int x = 12 * 30;
    Console.WriteLine (x);
}
}

```

2.2.1 标识符和关键字

标识符是程序员为类、方法、变量等选择的名字。下面按顺序排列上例中的标识符：

System Test Main x Console WriteLine

标识符必须是一个完整的词，它是由字母和下划线开头的Unicode字符组成的。C#标识符是区分大小写的。通常约定参数、局部变量和私有字段应该由小写字母开头（例如myVariable），而其他类型的标识符则应该由大写字母开头（例如MyMethod）。

关键字是编译器保留的名称，不能把它们用作标识符。以下是上例中的关键字：

using class static void int

下面是所有的C#关键字：

abstract	do	in	protected	true
as	double	int	public	try
base	else	interface	readonly	typeof
bool	enum	internal	ref	uint
break	event	is	return	ulong
byte	explicit	lock	sbyte	unchecked
case	extern	long	sealed	unsafe
catch	false	namespace	short	ushort
char	finally	new	sizeof	using
checked	fixed	null	stackalloc	virtual
class	float	object	static	void
const	for	operator	string	volatile
continue	foreach	out	struct	while
decimal	goto	override	switch	
default	if	params	this	
delegate	implicit	private	throw	

1. 避免冲突

如果用关键字作为标识符，可以在关键字前面加上@前缀。例如：

```

class class {...}    // 不合法的
class @class {...}   // 合法的

```

@并不是标识符的一部分，所以@myVariable和myVariable是一样的。

提示：@前缀在调用其他有不同关键字的.NET语言编写的库时非常有用。

2. 上下文关键字

一些关键字是上下文相关的，它们也可以不用@前缀就能作为标识符。它们是：

add	dynamic	in	partial	where
ascending	equals	into	remove	yield
async	from	join	select	
<hr/>				
await	get	let	set	
by	global	on	value	
descending	group	orderby	var	

使用上下文关键字作为标识符时，应避免与上下文中的关键字混淆。

2.2.2 字面值、分隔符和运算符

字面值是静态嵌入程序中的原始数据片段。我们的例子中用到的字面值有12和30。

标点有助于划分程序的结构。下面是我们的例子中用到的标点：

```
; { }
```

分号 (;) 用于结束一条语句。这意味着语句也可以放在多行中，例如：

```
Console.WriteLine  
    (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10);
```

大括号用于把多条语句作为一个语句块。

运算符用于改变和结合表达式。大多数C#运算符都用符号表示，例如乘法运算符*。我们将在后续章节中详细讨论运算符。我们的例子中用到的运算符有：

```
. () * =
```

点号 (.) 表示某个对象的成员（或数字的小数点）。括号在声明或调用方法时使用，空括号在方法没有参数时使用。而等号则用于赋值操作（双等号==才用于相等比较，我们将在之后见到）。

2.2.3 注释

C#提供了两种方式的注释：单行注释和多行注释。单行注释由双斜线开始，到本行结束为止。例如：

```
int x = 3; // 将数字3的值赋给变量x
```

多行注释由/*开始，由*/结束。例如：

```
int x = 3; /* 这是  
            多行注释*/
```

注释也可以嵌入到XML文档标签中，我们将在第4章的“XML文档”小节中介绍。

2.3 类型基础

类型定义了值的蓝图。值是由变量或常量表示的存储位置。变量代表它的值可以改变，而常量则表示它的值不可以更改（我们将在后面的章节中介绍常量）。我们在第一个程序中创建了一个名为x的局

部变量：

```
static void Main()
{
    int x = 12 * 30;
    Console.WriteLine (x);
}
```

变量是表示存储位置的符号，它包含的值可能会不断变化。相反，常量总是表示同一个值（后面会详细介绍）：

```
const int y = 360;
```

C#中所有值都是一种类型的实例。一个值或一个变量所包含的一组可能值均由其类型决定。

2.3.1 预定义类型示例

预定义类型是指那些由编译器特别支持的类型。int就是一种预定义类型，它代表从 -2^{31} 到 $2^{31}-1$ 的32位整数集。我们能够按照下面的方式对int类型的实例执行数学运算等函数：

```
int x = 12 * 30;
```

C#中另一个预定义类型就是string。string类型表示由零个或多个字符组成的有限序列，例如“.NET”或<http://oreilly.com>。我们可以通过下面的方式调用函数来使用字符串：

```
string message = "Hello world";
string upperMessage = message.ToUpper();
Console.WriteLine (upperMessage); // HELLO WORLD

int x = 2010;
message = message + x.ToString();
Console.WriteLine (message); // Hello world2010
```

预定义类型bool只有两种值：true和false。bool类型通常与if语句一起用于条件分支。例如：

```
bool simpleVar = false;
if (simpleVar)
    Console.WriteLine ("This will not print");

int x = 5000;
bool lessThanAMile = x < 5280;
if (lessThanAMile)
    Console.WriteLine ("This will print");
```

提示：在C#中，预定义类型（也称为内建类型）被当作C#关键字。在.NET Framework中的System命名空间下包含了很多并不是预定义类型的重要类型（例如DateTime）。

2.3.2 自定义类型示例

正如我们能使用简单函数来构建复杂函数一样，也可以使用基本类型来构建复杂类型。在下面的例子中，将定义一个名为UnitConverter的自定义类型，这个类将作为单位转换的蓝图。

```
using System;
```