.NET框架程序员参考手册·框架基础篇

武　装　等编著

国防工业出版社

·北京·

内 容 简 介

本书详细介绍了.NET 框架中的框架基础部分，和针对 Windows 的服务名称空间：System 名称空间、Microsoft.ComServices 名称空间和 Microsoft.Win32 名称空间。全书共5章，主要内容包括：.NET 框架的异常处理、数值类型、底层服务和功能支持，以及 Windows 通用服务等。

图书在版编目(CIP)数据

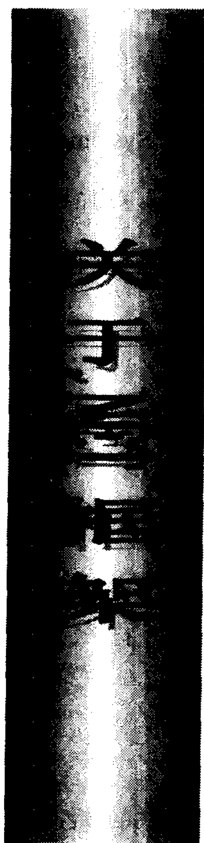
.NET 框架程序员参考手册.框架基础篇/武装等编著.
北京：国防工业出版社，2001.10
ISBN 7-118-02571-2

Ⅰ.N... Ⅱ.武... Ⅲ.窗口软件，Windows-系统结构-技术手册 Ⅳ.TP316.7-62

中国版本图书馆 CIP 数据核字(2001)第 037775 号

国防工业出版社出版发行
(北京市海淀区紫竹院南路 23 号)
(邮政编码 100044)
北京奥隆印刷厂印刷
新华书店经售
*
开本 787×1092 1/16 印张 32¼ 748 千字
2001 年 10 月第 1 版 2001 年 10 月北京第 1 次印刷
印数：1—3000 册 定价：49.00 元

(本书如有印装错误，我社负责调换)



Visual Studio.NET 7.0 是微软推出的新一代可视化集成开发环境，其中的.NET 就是指.NET 框架（.NET Framework）。.NET 框架是一种用于构建、配置、运行 Web 服务和应用程序的多语言环境，它主要由统一的编程类库、通用语言运行库（Common Language Runtime）和 ASP.NET（Active Server Pages.NET）三个部分组成。.NET 框架不但提供了诸如自动内存管理之类的很多强有力的功能，而且它的引入使得多语言间的无缝互用成为现实。

从某种程度上说，新版本的 Visual Studio 就是以.NET 框架为中心的：新引入的 C#语言本身并无类库，而是充分利用.NET 框架提供的功能； Visual Basic 7.0 语言上做了很大修改，而这些修改正是为了实现与.NET 框架的无缝兼容；Visual C++ 7.0 中提供了受控代码（Managed Extensions）编程，使得由 C++编写的代码也依然能够使用.NET 框架的服务。

.NET 框架中的类型有很多的功能，例如，封装数据结构、执行 I/O 操作、访问数据、控制服务器、获取类信息以及激活安全检查等等。.NET 框架中既包括比较抽象的基类，也包括由基类派生的、具有实际功能的类。这些派生类已经提供类足够强大的功能，但是如果需要，程序员依然可以通过继续派生提供更强大的功能。.NET 框架还包括接口及其默认实现。要使用接口的功能，程序员可以自己实现这些接口，也可以直接使用（或派生）运行库中的接口实现类。

由于.NET 框架类库的内容非常浩繁，为了向读者提供更具针对性的参考信息，《.NET 框架程序员参考手册》丛书分为以下 6 册：

- **框架基础篇**

本篇主要包括整个.NET 框架的根名称空间：System，以及微软的两个服务名称空间：Microsoft.ComServices 和 Microsoft.Win32。这些名称空间为用户提供了底层功能和服务支持，同时也是开发高级功能的基础。

- **数据访问篇**

本篇主要包括为数据 I/O 提供支持的名称空间：System.IO；为数据库访问提供支持的名称空间：System.Data 及其下属的三级名称空间。通过这些名称空间，用户能够方便地进行数据存取、数据库事务和 XML 编档。

- **网络编程篇**

本篇主要包括为进行网络和 Web 服务提供支持的名称空间：System.Web、System.NET，及其下属的三级名称空间等。

- **用户界面篇**

本篇主要包括为用户界面提供支持的名称空间：System.Drawing，System.Winforms 等。

- **常规操作篇**

本篇主要包括为 Windows 下的常用操作提供支持的名称空间：System.Configration，System.Reflection，System.Resource 和 System.Text 等。

- **组件模型篇**

本篇主要包括为组件模型提供支持的名称空间：System.ComponentModel，System.Collections，System.Resources，System.Core，System.ServiceProcess 和 System.Threading 等。

由于.NET框架涉及面极宽，为了节省篇幅，使读者能以最少的费用得到最广泛的信息，本书有以下几点约定：

- 在介绍每个名称空间时，都会将其中定义的成员以表格形式给出。读者可以根据这些表格给出的信息，迅速确定将查阅的内容。而.NET框架命名规则也使确定成员功能变得更加容易，例如 DirectorySeparatorChar 的意思为目录分隔符字符（Directory-Separator-Char）。
- 凡是“待提供”的名称空间成员，都只是在列表中简述，而不作进一步详细说明。
- 对于那些功能、形式类似的名称空间成员，只是选择其中最具代表性的进行介绍，其他的只列表简述。读者若需得知这些简述成员的详细形式，只要参考对应的详述成员即可。
- 另外，类库的每个层次都会自动继承其基层次中的所有成员，因此在介绍时也不再介绍这些继承所得的成员，而只是指出其派生层次。这样读者即可根据其派生层次确定其中包含有哪些继承成员。当然，对于那些被重载的继承成员，我们还是会加以详细介绍。

试读结束：需要全本请在线购买： www.ertongbook.com

致谢

本书除封面署名作者外，武装、于佳音、朱石、周一兵、薛文涛、林茵茵、黄剑波、李义、王芳、沈鹏、刘树声、季洪飞、司马小凡、王江辉、廖晓筠、沈冰、汤春明、许颖、赵立峰、李国梁、胡建明、龚雪梅、黄君玲、吴强、郭士明、陈刚、刘洪宇、李飞、王东生、皮丽丽、郭清等同志也为本书的出版付出了不同程度的劳动，在此一并表示感谢。

由于时间所限，书中错误和疏漏之处在所难免，敬请指正。

目　录

第 1 章　理解.NET 框架 …… 1
1.1　Visual Studio 7.0 与.NET 框架 …… 1
1.2　统一的编程类库 …… 2
1.2.1　.NET 框架类库的组织方式 …… 2
1.2.2　System 名称空间 …… 2
1.3　通用语言运行库 …… 4
1.3.1　多语言互用 …… 4
1.3.2　自动内存管理 …… 6
1.3.3　部件 …… 8
1.3.4　应用程序域 …… 10
1.4　ASP.NET …… 11
第 2 章　异常处理与数值类型 …… 12
2.1　异常处理 …… 12
2.1.1　异常处理基类 …… 12
2.1.2　派生异常类 …… 19
2.2　数值类型 …… 33
2.2.1　数值类型的定义层次 …… 33
2.2.2　数值类型基类 …… 35
2.2.3　基础数据类型 …… 41
2.2.4　时间数据类型 …… 114
2.2.5　其他数值类型 …… 151
2.3　枚举 …… 164
2.3.1　枚举基类 …… 165
2.3.2　AttributeTargets 枚举 …… 175
2.3.3　LoaderOptimization 枚举 …… 176
2.3.4　PlatformID 枚举 …… 177
2.3.5　TypeCode 枚举 …… 177
第 3 章　底层服务和功能支持 …… 179
3.1　System 名称空间的类成员 …… 179
3.1.1　Activator 类 …… 181
3.1.2　AppDomain 类 …… 186
3.1.3　AppDomainFlags 类 …… 209

3.1.4 Array 类 …… 215
3.1.5 Attribute 类 …… 235
3.1.6 AttributeUsageAttribute 类 …… 239
3.1.7 BitConverter 类 …… 241
3.1.8 Buffer 类 …… 250
3.1.9 CallContext 类 …… 253
3.1.10 CLSCompliantAttribute 类 …… 254
3.1.11 Console 类 …… 256
3.1.12 ContextBoundObject 类 …… 267
3.1.13 ContextStaticAttribute 类 …… 267
3.1.14 Convert 类 …… 268
3.1.15 DBNull 类 …… 294
3.1.16 Delegate 类 …… 297
3.1.17 Empty 类 …… 308
3.1.18 Environment 类 …… 309
3.1.19 EventArgs 类 …… 319
3.1.20 FlagsAttribute 类 …… 320
3.1.21 LoaderOptimizationAttribute 类 …… 321
3.1.22 LogicalCallContext 类 …… 322
3.1.23 MarshalByRefObject 类 …… 322
3.1.24 Math 类 …… 324
3.1.25 MTAThreadAttribute 类 …… 337
3.1.26 MulticastDelegate 类 …… 338
3.1.27 NonSerializedAttribute 类 …… 342
3.1.28 ObsoleteAttribute 类 …… 343
3.1.29 OperatingSystem 类 …… 344
3.1.30 ParamArrayAttribute 类 …… 347
3.1.31 Radix 类 …… 348
3.1.32 Random 类 …… 351
3.1.33 SerializableAttribute 类 …… 354
3.1.34 ServicedComponent 类 …… 354
3.1.35 STAThreadAttribute 类 …… 355
3.1.36 ThreadStaticAttribute 类 …… 356
3.1.37 Type 类 …… 357
3.1.38 UnhandledExceptionEvent 类 …… 411
3.1.39 URI 类 …… 413
3.1.40 Version 类 …… 427
3.1.41 WeakReference 类 …… 432
3.2 System 名称空间的接口成员 …… 435

3.2.1 IAsyncResult 接口 ··· 436
3.2.2 ICloneable 接口 ··· 437
3.2.3 IComparable 接口 ··· 438
3.2.4 IConvertible 接口 ··· 439
3.2.5 ICustomFormatter 接口 ··· 440
3.2.6 IFormattable 接口 ··· 440
3.2.7 IServiceObjectProvider 接口 ··· 441
3.3 System 名称空间的 Delegate 成员 ··· 442
3.3.1 AsyncCallback Delegate ··· 442
3.3.2 CrossAppDomainDelegate Delegate ··· 443
3.3.3 EventHandler Delegate ··· 443
3.3.4 UnhandledExceptionEventHandler Delegate ··· 444
第 4 章 Microsoft.ComServices 名称空间 ··· 445
4.1 Microsoft.ComServices 名称空间的类成员 ··· 447
4.1.1 ApplicationAccessControlAttribute 类 ··· 447
4.1.2 ApplicationActivationAttribute 类 ··· 449
4.1.3 ApplicationIDAttribute 类 ··· 450
4.1.4 ApplicationNameAttribute 类 ··· 451
4.1.5 ApplicationQueuingAttribute 类 ··· 452
4.1.6 AutoCompleteAttribute 类 ··· 453
4.1.7 ComponentAccessControlAttribute 类 ··· 454
4.1.8 ConstructionEnabledAttribute 类 ··· 454
4.1.9 ContextUtil 类 ··· 456
4.1.10 DescriptionAttribute 类 ··· 463
4.1.11 EventClassAttribute 类 ··· 463
4.1.12 EventTrackingEnabledAttribute 类 ··· 465
4.1.13 InterfaceQueuingAttribute 类 ··· 465
4.1.14 JustInTimeActivationAttribute 类 ··· 466
4.1.15 MustRunInClientContextAttribute 类 ··· 467
4.1.16 ObjectPoolingAttribute 类 ··· 468
4.1.17 RegistrationHelper 类 ··· 470
4.1.18 SecurityCallContext 类 ··· 473
4.1.19 SecurityCallers 类 ··· 475
4.1.20 Securityldentifier 类 ··· 475
4.1.21 Securityldentity 类 ··· 479
4.1.22 SecurityRoleAttribute 类 ··· 480
4.1.23 SharedPropertyGroupManager 类 ··· 481
4.1.24 SynchronizationAttribute 类 ··· 483
4.1.25 TransactionAttribute 类 ··· 484

4.2 Microsoft.ComServices 名称空间的接口成员 ······485
4.2.1 IObjectConstruct 接口 ······485
4.2.2 IObjectConstructString 接口 ······486
4.2.3 IObjectContext 接口 ······486
4.2.4 IObjectContextInfo 接口 ······487
4.2.5 IObjectControl 接口 ······488
4.2.6 ITransaction 接口 ······489
4.2.7 SharedProperty 接口 ······490
4.2.8 SharedPropertyGroup 接口 ······490
4.3 Microsoft.ComServices 名称空间的枚举成员 ······492
4.3.1 AccessChecksLevelOption 枚举 ······493
4.3.2 ActivationOption 枚举 ······493
4.3.3 AuthenticationOption 枚举 ······494
4.3.4 ImpersonationLevelOption 枚举 ······494
4.3.5 SynchronizationOption 枚举 ······495
4.3.6 TransactionOption 枚举 ······496
第 5 章 Microsoft.Win32 名称空间 ······497
5.1 Microsoft.Win32 名称空间层次 ······497
5.2 操作注册表 ······497

第 1 章　理解.NET 框架

Visual Studio.NET 7.0 是微软推出的新一代可视化集成开发环境，其中的.NET 就是指.NET 框架（.NET Framework）。.NET 框架是一种用于构建、配置、运行 Web 服务和应用程序的多语言环境，它主要由统一的编程类库、通用语言运行库（Common Language Runtime）和 ASP.NET（Active Server Pages.NET）3 个部分组成。

.NET 框架不但提供了很多诸如自动内存管理之类的强有力的功能，而且它的引入使得多语言间的无缝互用成为现实。实际上，正如微软在桌面操作系统中所取得的成就那样，它现在正致力于创建一种统一的开发工具。从商业上说，也许这可以被认为是具有垄断的倾向，然而这种努力的确为开发者提供了极大的方便。

1.1　Visual Studio 7.0 与.NET 框架

因版本延续与称呼方便之故，Visual Studio.NET 7.0 又被称为 Visual Studio 7.0。Visual Studio 7.0 中除了 C#、Visual Basic、Visual C++和 Visual FoxPro 等基于组件的开发工具外，还提供了一系列能够简化团队设计开发，提出解决方案的新技术。当然一直被视为开发宝典的 MSDN，也是 Visual Studio 的重要组成部分之一，有经验的程序员能够从中得到最好的帮助。

从某种程度上说，新版本的 Visual Studio 就是以.NET 框架为中心的：新引入的 C#语言本身并无类库，而是充分利用.NET 框架提供的功能；Visual Basic 7.0 语言上做了很大修改，而这些修改正是为了实现与.NET 框架的无缝兼容；Visual C++ 7.0 中提供了受控代码（Managed Extensions）编程，使得由 C++编写的代码也依然能够使用.NET 框架的服务。

由新版本 Visual Studio 的名称就可以断定：它较之其以前版本，至少在网络开发方面有了显著的改进。这一点从微软对 Visual Studio 7.0 的定位即可得到验证——它主要用于开发企业级 Web 应用程序和高性能桌面应用程序。而这些功能则主要是通过.NET 框架提供和实现的。

所谓 Web 服务就是指能够使用 XML，通过 HTTP 接受请求和数据的应用程序。Web 服务并非绑定于某个特定的组件技术或对象调用协定，因此使用任意的语言、组件模型或操作系统都能够对其进行访问。在 Visual Studio 7.0 中，用户能够使用 Visual Basic、C#或 ATL 服务器，创建和包含 Web 服务。

XML（可扩展置标语言，eXtensible Markup Language）为描述结构化数据提供了一个框架，它是 SGML 的一个子集，并根据 Web 传送进行了相应的优化。WWW 协会

试读结束：需要全本请在线购买： www.ertongbook.com

（W3C）将 XML 作为标准，这样结构化数据将被统一并独立于应用程序。Visual Studio 7.0 为 XML 提供了完全的支持，例如引入了 XML Designer 以简化 XML 编辑和创建等。

1.2 统一的编程类库

.NET 框架提供了统一的、面向对象的可扩展层次类库（API）。它将 C++开发者使用的 MFC 类库，Java 开发者使用的 WFC（Windows Foundation Classes）类库；Visual Basic 开发者使用的 Visual Basic API 等完全不同的库统一起来。通过其中定义的通用 API，.NET 框架支持跨语言的继承、错误处理和调试，这实际上意味着那些原本不同的编程语言间都被划上了等号。在开发应用程序组件时，开发者可以根据具体条件选择最合适的语言。而由不同语言编写的组件间的通信和互用，则由.NET 框架负责。

1.2.1 .NET 框架类库的组织方式

.NET 框架的类库是通过名称空间组织起来的。对于有些语言来说（例如 C#），名称空间既可以作为程序的内部组织系统，也可以作为其外部组织系统（使其他程序能调用本程序元素）。

全限定名是名称空间和类型的唯一标识。以下规则用以确定名称空间或类型 N 的全限定名：

• 如果 N 为全局名称空间的成员，则其全限定名为 N。

• 如果 N 是在另一名称空间 S（S 为该名称空间的全限定名）中声明的成员，则其全限定名为 S.N。

换句话说，N 的全限定名为标识符的完整层次（始自全局名称空间）路径。

名称空间声明的全限定名可能为单个标识符，也可能为一个以“.”分隔的标识符序列。这种分层方法用于将逻辑相关类组织在一起，这样使用和引用会更方便。例如，System.Data.DataRow 类与具有 System.Data.x 名称形式的类相关，也就是说，所有具有 System.Data 前缀的类都可以用来在运行库中确定类型信息。

名称空间中的类或结构中声明的类型被称为嵌套类型。类名中最后一个点号前的部分（例如 System.Data），通常为名称空间名；而最后一个点号后的部分（例如 DataRow）为类名。通过命名规则将相关类归入名称空间的方法，对于类库的建立和文档化是非常有好处的。一个名称空间可能由多个部件组成；而一个部件也可能包含多个名称空间中的类。使用 using 指令能够方便地引用名称空间。

1.2.2 System 名称空间

.NET 框架中既包括比较抽象的基类，也包括由基类派生的、具有实际功能的类。这些派生类已经提供类足够强大的功能，但是如果需要，程序员依然可以通过继续派生提供更强大的功能。.NET 框架还包括接口及其默认实现。要使用接口的功能，程序员可以自己实现这些接口，也可以直接使用（或派生）运行库中的接口实现类。

.NET 框架中的类型有很多的功能，例如，封装数据结构、执行 I/O 操作、访问数据、控制服务器、获取类信息以及激活安全检查等。类型是.NET 框架应用程序、组件和控件的构建基础。

System 名称空间是整个.NET 框架的基础，其成员负责为整个框架提供底层服务支持和功能实现。例如，在 System 名称空间中定义了数据类型、事件和事件处理函数、接口、属性、网络和异常处理等。

System 名称空间由 94 个类、8 个接口、25 个结构、6 个枚举和 4 个 delegate 构成，与其对应的组件为 mscorlib.dll。System 名称空间中包括了代表基础数据类型的结构，以及能够完成异常处理、创建 delegate、处理应用程序域以及自动内存管理的其他类。

此外，System 名称空间还下属 24 个二级名称空间：

• 与数据相关的名称空间：System.Data 和 System.XML。

• 与组件模型相关的名称空间：System.CodeDOM、System.ComponentModel 和 System.Core。

• 与配置相关的名称空间：System.Configuration。

• 与框架服务相关的名称空间：System.Diagnostics、System.DirectoryServices、System.ServiceProcess、System.Messaging 和 System.Timers。

• 与全球化相关的名称空间：System.Globalization、System.Resources。

• 与网络相关的名称空间：System.Net。

• 与编程要素相关的名称空间：System.Collection、System.IO、System.Text 和 System.Threading。

• 与映像相关的名称空间：System.Reflection。

• 与客户端 GUI 相关的名称空间：System.Drawing 和 System.Winforms。

• 与运行库底层服务相关的名称空间：System.Runtime。

• 与.NET 框架安全相关的名称空间：System.Security。

• 与 Web 服务相关的名称空间：System.Web。

在本书其余部分的内容中，将向读者详细介绍.NET 框架类库中各个名称空间及其中成员的功能。由于.NET 框架涉及面极宽，为了节省篇幅，使读者能以最少的费用得到最广泛的信息，本书有以下几点约定：

• 在介绍每个名称空间时，都会将其中定义的成员以表格形式给出。读者可以根据这些表格给出的信息，迅速确定将查阅的内容。而.NET 框架命名规则也使确定成员功能变得更加容易，例如 DirectorySeparatorChar 的意思为目录分隔符字符（Directory-Separator-Char）。

• 凡是“待提供”的名称空间成员，都只是在列表中简述，而不作进一步详细说明。

• 对于那些功能、形式类似的名称空间成员，只是选择其中最有代表性的进行介绍，对其他只列表简述。当需要得知这些简述成员的详细形式时，参考对应的详述成员即可得到。

• 另外，类库的每个层次都会自动继承其基层次中的所有成员，因此在介绍时也不再介绍这些继承所得的成员，而只是指出其派生层次。这样读者即可根据其派生层次确定其中包含有哪些继承成员。当然，对于那些被重载的继承成员，我们还是会加以详细

介绍。

1.3 通用语言运行库

通用语言运行库负责管理代码的运行，并提供使开发称得到简化的服务，其功能由编译器和工具负责解释。这种运行于管理运行环境下的代码被称为受控代码。受控代码能够得益于通用语言运行库提供的各种服务，例如跨语言集成、异常处理、安全强化、版本和安装支持、简化的组件交互模型以及调试和优化等。

通用语言运行库对于组件的运行和开发起着重要的作用。当组件在运行时，运行库负责管理内存分配、启动、结束进程、执行安全策略以及满足组件间可能存在的依赖性。除此之外，诸如映像这样的功能，极大地降低了将商用逻辑转化为可重用的组件时的代码编写量。

1.3.1 多语言互用

通用语言运行库简化了需要进行跨语言交互的组件和应用程序的设计。这不但使得由不同语言编写的对象之间能够进行通信，而且还能牢固地整合其行为。在通用语言运行库出现以前，不同语言编写的对象只能通过标准二进制进行通信。而由于不同语言有时会使用相互冲突的协议，因此使得语言间的通信变得更加困难。.NET 框架的出现，解决了这个问题。无论.NET 框架对象是否使用同样的语言编写，它们都将自动具有相互通信和交互的能力。

例如，可以使用一种语言定义一个类，然后使用另一种语言生成该类的派生类，或调用该类中的方法；也可以将某类的实例传递给用不同语言编写的另一个类的方法。正是由于语言编译器和工具使用了由运行库定义的通用类型系统，才使得这种跨语言的整合得以实现。语言编译器和工具必须遵循运行库规则进行类型的定义、创建、使用、维护和绑定。

这也就是说，无论编译时使用何种语言，.NET 框架对象总能调用其他对象中的方法、继承其他对象的实现，或将某个类实例传递给其他对象。此外，运行库的多语言互用功能使调试器（无论其种类）能够理解，并调试由多种语言编写的应用程序。运行库对不同语言使用同样的异常处理机制，这意味着使用一种语言抛出的异常，可以被由其他语言编写的对象截获和理解。

为了让使用不同语言编写的对象间能够进行交互，就必须将这些语言的共同数据类型和特性提供给调用者。因此，运行库定义了一套 CLS（通用语言规范）的语言特性。为了支持语言互用，.NET 框架类型都与 CLS（通用语言规范，Common Language Specification）兼容，并且能用于任何支持动态语言运行库的编译器。

1. CLS 规范

CLS 中定义的语言特性包括：

• 使用的基础数据类型包括：System.Object （类的根层次，可代表任何类型的对象）、System.Boolean （true 或 false）、System.Char （2 字节的无符号整数）、

System.Byte（1 字节的无符号整数）、System.Int16（2 字节有符号整数）、System.Int32（4 字节有符号整数）、System.Int64（8 字节有符号整数）、System.Single（4 字节浮点数）、System.Double（8 字节浮点数）和 System.String（字符串）。

• 数组具有以下特性：可以为现有任何类型，起始索引必须为 0，而数组维数大于等于 1；并且其元素类型必须为 CLS 类型。Visual Basic 7.0 以前的版本中，数组的起始索引为 1，而在 7.0 版中数组的起始索引被修改为 0，以与.NET 框架相适应。

• 类型具有以下特性：可以是封装或抽象型；可以定义多个或 0 个构造函数；可以实现大于等于 0 个接口，并且不同的接口可以包括同名或数字签名的方法；可以派生自某个类型，并且可以重载和隐藏该类型提供的成员；可以拥有大于等于 0 个成员：字段、方法、事件或类型；可以拥有公有或可见部件，但是只有公有部件能作为类型的公共接口；值类型必须继承自 System.ValueType，除非是继承自 System.Enum 的枚举。

• 类型成员具有以下特性：可以在其他类型中被重载或隐藏；参数类型和返回类型必须为与 CLS 兼容的类型；构造函数、方法和性质可以被多次载入；可以为抽象，但是如果类型被封装则不行；可以拥有公有、私有、族、部件、可见的族和/或部件，但是只有公有、族或部件能作为类型的公共接口。

• 方法具有以下特性：可以为虚函数、实例或静态类型；虚函数和实例方法可以为抽象或具体（具有实现），静态方法必须为具体；虚函数可以为 final 型。

• 字段具有以下特性：可以为静态或非静态；静态字段可以有初始值或精确值。

• 属性具有以下特性：可以向外部提供 Get 和 Set 方法；必须遵循命名格式 get_<属性名>和 set_<属性名>；Get 函数的返回类型，和 Set 函数的第一个参数必须一致，即属性类型；不能只根据属性类型区分；如果定义了 X 属性，就不能在同一类中再定义 Get_X 和 Set_X 方法；可以被指定索引。

• 枚举具有以下特性：基础类型必须为 Byte、Int16、Int32 或 Int64；每个成员都是枚举类型的静态字段；不能实现任何接口；可以为多个枚举成员赋予同一个值；必须继承自 System.Enum。

• 异常具有以下特性：可以被抛出和捕获；必须继承自 System.Exception。

• 接口具有以下特性：可以需要其他接口的实现；可以定义属性、事件和虚函数。

• 事件具有以下特性：方法的添加和删除必须同时提供或消失，每个方法必须有一个由 System.Delegate 派生的参数；必须遵循命名格式 add_<事件名>、remove_<事件名>和 raise_<事件名>。

• 定制标志只能为 Type、String、Char, Boolean、Byte、Int16、Int32、Int64、Single、Double、Enum (CLS 类型)和 Object 类型。

• 可以创建和调用 delegate。

• 标识符具有以下特性：第一个字符必须来自约束集；在单个命名空间内，不能用大小写的方式区分标识符（例如部件中的类型和类型的成员）。

2. CLS 兼容组件

如果组件在为外部提供的 API 中，只使用了 CLS 特性，那么任何支持 CLS 的对象都能访问该组件。遵循 CLS 规则，并只使用 CLS 特性的组件也被称为 CLS 兼容组件。如果希望组件或应用程序与 CLS 兼容，那么在下列地方必须使用 CLS 特性：公共类的

定义；公共类中的公共成员定义，及其子类中可访问成员的定义；公共类中公共方法的参数定义，以及其子类中可访问方法的参数定义。

对于私有类或私有成员（例如，私有类的定义、公共类中私有方法的定义以及局部变量等），只要您的编译器支持就可以使用任意的语言特性。同样，在类的实现代码中也可以使用任意语言特性，这不会影响到组件的 CLS 兼容性。

支持运行库的许多语言编译器，也支持 CLS 中的数据类型和特性。这些语言编译器通过提供 CLS 数据类型和特性，使程序员能够直接使用它们进行开发，从而简化了 CLS 兼容程序的设计。对 CLS 的兼容程度可以分为以下两种：

- CLS 兼容客户工具：允许开发者访问 CLS 兼容库中的所有特性。这些编译器的用户可能不能创建新类型，但是可以使用 CLS 兼容库中定义的所有类型。
- CLS 兼容扩展工具：不仅允许开发者使用 CLS 兼容库中定义的所有类型，而且允许定义新类型。

在设计 CLS 兼容组件时，选择那些 CLS 兼容编译器是很有好处的。这些语言编译器提供了一种“开关”，它能在代码中使用了 CLS 不支持的功能时发出通知。如果使用非 CLS 兼容编译器，那么编写 CLS 兼容代码的工作就会相对困难一些。这是因为：首先，必须自己注意代码的 CLS 兼容性；其次，编译器可能无法提供您想使用的 CLS 特性。

多语言互用是开发者梦寐以求的，而且这一点对于 Web 应用程序尤其重要。它不但保证了程序能够与其他开发者提供的组件进行交互，而且允许一个服务器应用程序使用由不同语言编写的组件。如果您开发出了 CLS 兼容框架（类库），那么它将能被许多编程语言使用，其应用范围之广是无法不令人激动的。

1.3.2 自动内存管理

在开发时，由于其高度的自动化（例如内存管理），运行库使程序员感觉开发变得异常简单，在与目前使用的 COM 相比时尤其如此。手动内存管理（Manual Memory Management）需要开发者分配和释放内存块，它既浪费时间，又比较困难。.NET 框架中提供的自动内存管理，将开发者从这个繁重的工作中解脱出来。在大多数情况下，自动内存管理能够提高代码质量和开发效率，而不会为程序性能带来负面影响。请看如下 C#示例程序：

```
using System;

public class Stack
{
    private Node first = null;
    public bool Empty
{
        get
        {
            return (first == null);
```

```
            }
        }
        public object Pop()
    {
            if (first == null)
                throw new Exception("不能弹出空堆栈!");
            else
            {
                object temp = first.Value;
                first = first.Next;
                return temp;
            }
        }
        public void Push(object o)
    {
            first = new Node(o, first);
        }
        class Node
        {
            public Node Next;
            public object Value;
            public Node(object value): this(value, null) { }
            public Node(object value, Node next)
            {
                Next = next;
                Value = value;
            }
        }
    }
```

此段代码中定义的 Stack 类，由 Node 实例链表实现，而 Node 实例则由 Push 方法创建。当某个 Node 实例不会再被其他代码访问时，它就可以被冗码收集器（Garbage Collector）收集了。例如，当从 Stack 中删除某个条目时，相关的 Node 实例就符合被冗码收集器收集的条件了。下面给出 Stack 类的使用示例：

```
class Example
{
    static void Main() {
        Stack s = new Stack();
        for (int i = 0; i < 10; i++)
            s.Push(i);
```


试读结束：需要全本请在线购买： www.ertongbook.com