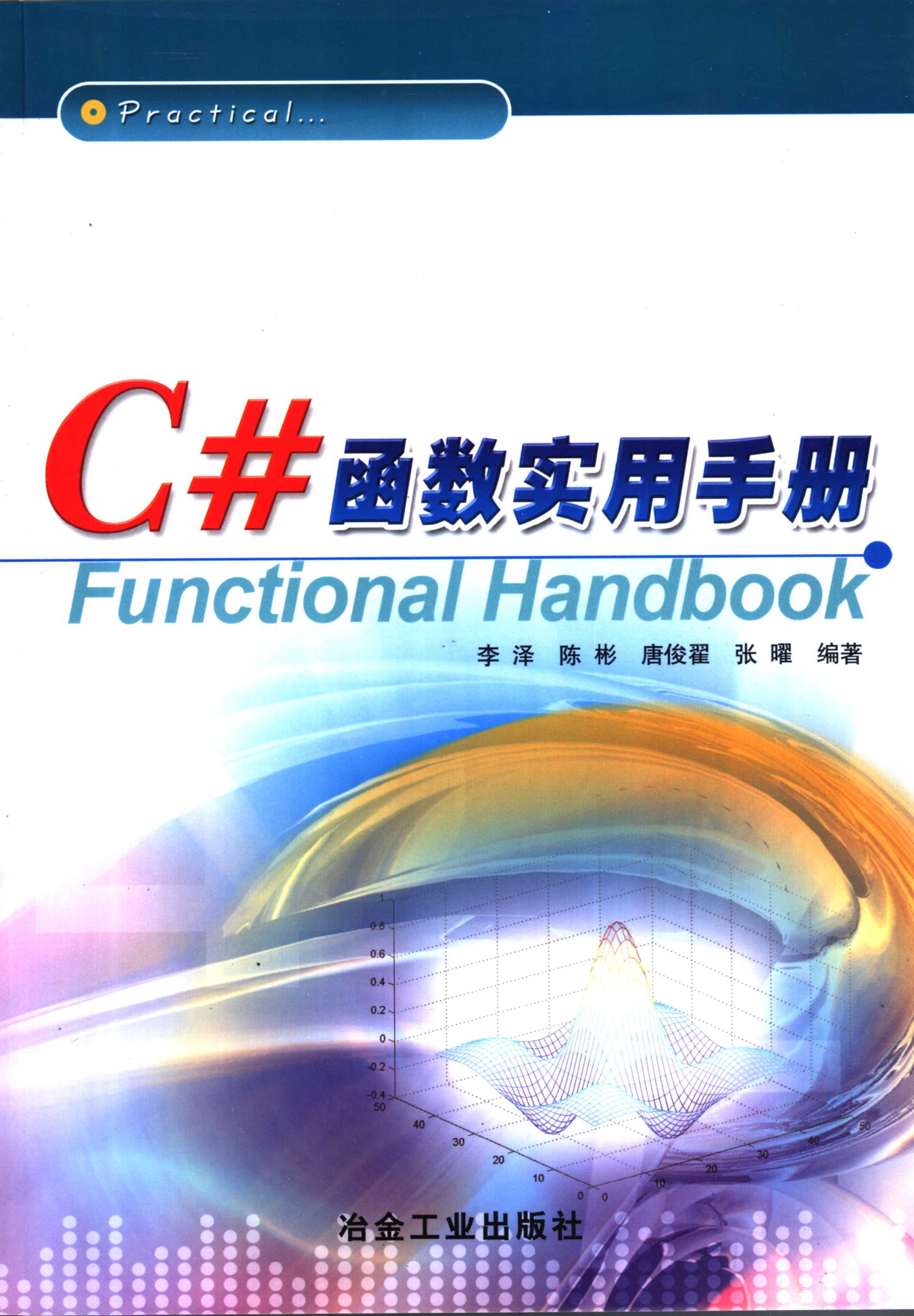
Practical...

C#函数实用手册

Functional Handbook

李泽 陈彬 唐俊翟 张曜 编著



冶金工业出版社

内 容 简 介

C#是 Microsoft 公司推出专门用于.NET 的面向对象编程语言。本书涵盖了 C#的核心类库，可以帮助程序员进行字符串处理、绘图、数学计算、数据库访问、网络应用等方面的工作，全面而系统地描述了使用 C#语言编程常用的各种类库，主要包括类描述和接口描述。从类的层次结构示意图、成员变量、成员函数等做了详细的说明。此外还辅以示例对类的典型成员函数的使用做了介绍，让读者迅速地掌握函数的具体用法，并且提供快速的函数索引，让读者查找相关函数的用法。

本书内容丰富，结构清晰，使用方便，主要面向中级以上水平的程序员，同时兼顾了初级水平用户的学习和参考。本书是参考手册而不是指导手册，相信无论是对 C#程序员还是学习和使用 C#技术的广大读者和用户，本书都将是他获取完整、专业和权威的 C#信息的重要源泉。

图书在版编目（CIP）数据

C#函数实用手册 / 李泽等编著. —北京：冶金工业出版社，2005.12
ISBN 7-5024-3882-3

I. C… II. 李… III. C 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（2005）第 139379 号

出版人 曹胜利（北京沙滩嵩祝院北巷 39 号，邮编 100009）
责任编辑 程志宏
佛山市新粤中印刷有限公司印刷；冶金工业出版社发行；各地新华书店经销
2006 年 2 月第 1 版第 1 次印刷
787mm × 1092mm 1/16； 23.5 印张； 1127 千字； 366 页
45.00 元
冶金工业出版社发行部 **电话**：（010）64044283 **传真**：（010）64027893
冶金书店 **地址**：**北京东四西大街 46 号**（100711） **电话**：（010）65289081
（本社图书如有印装质量问题，本社发行部负责退换）

前　言

一、关于本书

C#是 Microsoft 公司推出的专门用于.NET 的面向对象编程语言。它广泛地应用于字符串处理、绘图、数学计算、数据库和网络等方面。由于 C#的核心数据库函数数量庞大，给广大 C#程序员的工作和学习带来了一些不便，广大 C#用户迫切需要一本内容全面、方便查询的 C#函数实用手册作为自己工作和学习的参考书。同时，C#语言是高校的一门重要学科，广大的师生又有类似的需要，可是目前的 C#工具型书籍并不多见。为此，本书作者总结自己的 C#语言的教学经验和研究经验，按照 C#函数的应用领域和功能进行分类说明，并对各个函数的功能、语法、参数、返回值和应用实例都做了详细的介绍说明。

本书内容全面、细致、分类清晰，因而对不同层次的用户都有很大的参考价值，本书一定会成为广大 C#用户的最有力的助手。

二、本书结构

本书共分 8 章，具体内容如下：

第 1 章：概述。主要介绍了 C#的相关背景知识以及 C#的特点、C#和 C++的比较、C#和 Java 的比较、C#的使用以及 C#常用术语的解析。

第 2 章：基础类。主要介绍了 C#常用工具、C#容器工具、C#规则化工具、C#线程工具、C#国际化工具以及 C#资源管理工具。

第 3 章：I/O 操作。主要介绍了和 I/O 操作相关的函数类以及 I/O 异常处理。

第 4 章：图形编程。主要介绍了图形编程类库以及窗口制作工具。

第 5 章：网络编程。主要介绍了关于网络编程的 C#类。

第 6 章：数据操作。主要介绍了数据访问管理工具以及 XML 支持工具。

第 7 章：Web 应用。主要介绍了关于 Web 的 HttpApplicationState 类、HttpUtility 类以及 Web 编程的异常处理。

第 8 章：安全和权限。主要介绍了 DES 类、DSA 类、MD5 类、RSA 类以及 System.Security 命名空间的异常处理。

三、本书特点

本书根据 C#语言的应用范围进行分类，形成三层体系的结构。第一层目录按 C#的各个应用功能来进行划分；第二层目录则按更细致的功能进一步划分；第三层则是具体的按字母升序排列的划分。所以您只要知道类或方法的大致功能就可以按功能定位到具体对象类位置，轻松定位到所要找的方法。

四、本书适用对象

本书内容全面，深入浅出。不仅适合于 C#的开发者，也可供广大编程爱好者参考。

由于编写时间仓促，水平有限，书中难免会存在疏漏，欢迎广大读者批评指正。联系方法如下：

电子邮箱：service@cnbook.net

网址：www.cnbook.net

此外，**本书的源代码可在该网站的下载中心免费下载，**如果读者在查阅本书过程中遇到疑难问题或觉得有不妥之处，也可到该网站的相关论坛进行探讨。该网站还有一些其他相关书籍的介绍，可以方便读者选购参考。

编　者

2005 年 9 月

目　录

第1章 概 述

本章提要：

本章主要介绍了C#及其特点，并与C++和Java进行比较，简单介绍了C#的使用及其常用专业术语的解析等内容，并对全书的主要内容进行了概述。

1.1 C#简介

C#是Microsoft公司在推出用于生成集成应用程序的新平台——Microsoft .NET框架的同时推出的编程语言，因此它被视为一种新的专门用于.NET运行的面向对象编程语言。

C#语言是微软公司专门为与.NET运行时高度兼容而设计的一种现代面向对象编程语言。在设计过程中，微软公司学习了近二十年里其他类似语言的经验，使得C#语言和C语言一样非常简洁和强大；与C++语言一样是面向对象的，同时C#还在语句、表达式和运算符等方面继承了许多C++功能；和Visual Basic一样都采用图形化设计方法、简化用户界面的创建工作；与Java语言一样编译依赖于运行时的内部服务的字节代码。同时C#在类型安全性、版本转换、事件和垃圾回收等方面进行了一系列的创新和优化。C#语言可以在使用.NET所有基类的同时，还可以实现对其他常用API样式（如COM、自动化和C样式API等）的访问。它还支持unsafe模式，在此模式下可以使用指针操作不受垃圾回收器控制的内存。

C#就其本身来说只是一门语言，尽管它是用在生成面向.NET环境的代码，但它本身并不是.NET的一部分。一些特性.NET支持，而C#不支持；另外一些特性C#支持，而.NET不支持。但C#语言也不能孤立地去看待，很多地方需要和.NET框架一起考虑。现在的C#编译器是专门用于.NET的，因此在C#中编写的所有代码总是在.NET框架里运行的。在许多情况下，C#的特定语言特性取决于.NET的特性，也就是说依赖于.NET的基类，如C#的基本数据类型等，这也是C#的优势所在，因为它有丰富的API资源。

C#代表了编程语言演化的新的发展方向。它可以创建不同类型的工程——从类库到控制台应用程序，到Web接口，为企业开发人员提供了一种建立客户机—服务器应用程序的强大工具。这是一种功能全面的语言、支持属性等高级工具。

1.2 C#的特点

1.2.1 面向对象功能

C++语言只是在C语言的基础上嫁接了类的概念，因此，在C++中进行面向对象编程总是不太得心应手。例如，可以采用更过程化的方法，在使用对象的地方依赖全局函数；或者究竟有多少种方式能使类变得抽象，让它只提供一个接口，而不能被实例化。更准确地说就是在要表达某些东西的时候，会被非常繁多的选项所淹没，从来就没有真正觉得自己是以最好的方式实现某个概念。

C#语言在限制至关重要的核心的语法上，它的面向对象技术相对来说已经成熟，而且有很高的效率。同时，要鼓励用户建立好的类结构，对继承性就肯定会有一定的限制。例如，一个类可以从无数个类中继承接口，但它只能从一个基类中继承其实现的方式。又如多态函数的模糊性问题在C#里用一种更清晰的新语法来解决，即声明虚拟和“纯虚拟”的函数。特别是一个类可以先行提供方法执行方式，在该方法的前面通过使用abstract关键字来进行声明，使得其子类也这么做。而且，通过使用密封关键字sealed进行声明的话，可以创建不能被继承的类。和C++一样，控制谁在查看类中的数据成员和方法由各种关键字来决定。

C#的属性方法是C#类机制最好的特性之一。这种方法把属性的读写集中到一个地方，以便更容易对它进行控制和使用。其整体效果是使得从对象的角度去看它也是对象，因为它们的属性在语法上更像特性，而不是类似伪函数的调用。使用索引符来对属性的特殊形式进行说明，它通过一种非常直观的语法来显示类中的数组。

通过C#编写的类和C++编写的一样，可以使用多个参数化的构造函数，但它们执行析构函数的方式是完全不同的。除了析构函数外，C#类还提供finalize方法，当出现对象没有释放，但运行中的无用存储单元收集器要删除该对象这种情况时，就由该方法来解决。同时，C#类不支持一般类型代码的模板。

通过对运算符进行重载，可以指定在运算符应用到类的实例上时，类的实例要完成些什么动作。但是如果滥用运算符重载这一特性的话，会使得编写的代码极其混乱，降低代码的可读性。为了避免这种代码混乱出现，C#对部分可以进行重载的运算符进行了一定的限制。

提供了最好的方式来体现面向对象的概念，使编程过程中更好地体现面向对象的思想，是C#的一大特点。

1.2.2 流控制语句

C#提供了正常的流控制语句，例如if...else条件语句、do...while循环、while循环和for循环等语句。同时还支持for each结构在集合、数组、列表和其他容器中迭代。C#提供支持if...else的缩写形式（A?B:C），而且C#还通过要求switch...case块中的每个case从句都要有一个break语句来避免错误。

1.2.3 .NET基类

早期C++最大的一个缺点就是缺乏有效的数据类型。虽然可以创建指向任何数据类型的指针、创建任何所希望的类型的类，但必须要扩展它们的功能，为字符串、布尔值等创建数据类型。

C#和.NET框架及.NET基类会进行交互，所有表示数据类型的C#关键字都由编译器直接映射到对应的基类上，而抛出的异常对象和捕获的异常对象也是继承System.Exception的异常对象。

C#的许多基本特性，主要是通过某些基类来实现的，因此，C#在得到.NET类库支持这方面是许多其他编程语言所没有的。

1.2.4 反射和属性

多形态和后期绑定是面向对象编程实现其优势的两种方式。由于后期绑定对象的客户机常常需要确定它们处理的对象的子类型，以使用特殊的方式对它们进行处理，所以，通常由客户机通过反射确定服务器对象的子类和该子类支持的方法——不需要标识代码来区分服务器类。

属性是把元数据和类关联起来的一种方式，所以元数据可以通过反射来显示。在C#语言里，用户可以创建自己的属性，通过

反射传送信息，或者使用其他人创建的属性。例如，可以由一些属性来确定一个类在COM+服务范围里的动作，就可以使用这些服务，因为C#编译器可以解释带有“挂钩”的组件编译版本属性。

1.2.5 适用性

C#是Web开发的一个利器是无可非议的，通过C#语言能迅速开发出健全的n层客户机——服务器的应用程序。通过使用ADO.NET，C#能快速对多种类型的数据库进行访问，返回的数据由ADO.NET对象处理后，能以XML形式在网络间传送。同时，C#也是实现数据访问对象层的一个优秀模式，通过它来对不同数据库进行插入、更新和删除等操作。

C#是第一个基于组件的C语言，因此它非常适合用于实现事务对象层。它为内部组件的通信封装了杂乱的信息通道系统，开发人员不必通过严格遵循组织事务规则的方法来将数据访问对象组合在一起。通过对属性的使用，C#事务对象可以进行COM+服务提供的方法级的安全检查、对象暂存和JIT的启用，而且可以使用.NET附带的工具将新的事务对象和原来的COM组件互相连接起来。

使用C#来编写ASP.NET页面是定制用户界面的最好方法之一。ASP.NET页面的编译和执行都非常快，它们还可以在VS.NET的开发环境里调试，也支持早期绑定、模块化等各种语言特性，而且使用C#编写的ASP.NET页面整齐，可读性好，易于维护。

在支持Web开发的同时，C#仍然能极好地支持胖客户应用程序的开发，它以Window Form的形式运行；C#也可以用于创建控制台应用程序，也就是运行在DOS窗口下的纯文本应用程序，也就是说，C#也适合开发后台调度程序和驱动程序。

1.2.6 代码安全性

C#非常擅长于隐藏基本内存管理，因为它使用了无用存储单元收集器和引用。但在很多时候，开发人员需要对内存进行直接访问。众所周知，指针的使用会带来很大的风险，但指针在编写高效代码的强大性和灵活性也是极其突出的，为了平衡风险和效率，在C#语言里，只允许在特别标识的代码块里使用指针。也就是说，C#为了提高代码的安全性，虽然允许使用人们普遍认为不安全的C++习惯用语代码，但要求用户只能在前面加了不安全关键字的特殊代码块里才能够使用。在代码块前面加上表示不安全的关键字，让开发人员知道该段代码是危险的，并作为服务于C#编译器的一个标识符，降低对不安全代码的限制。

1.3 C#和C++的比较

C++是在计算机作为运行基本命令行的用户接口的独立机器时人们设计的一门面向对象的编程语言；C#是微软公司专门为结合.NET工作而设计的，适用于视窗和鼠标控制用户界面、网络和因特网的现代开发环境。C#继承了早期面向对象编程语言的优点，其中也包括C++，因此两者的语法和工作方式都有相似的地方，因此，对二者进行比较能更好地使有C++基础的程序员掌握并应用好C#语言，下面分几方面进行比较：

1.3.1 程序结构

C#面向对象的要求比C++更为严格。C++程序包括一个入口点，以及各种类、结构和定义在类外部的全局变量和函数。虽然一个好的面向对象程序设计要求尽可能将程序中最顶层的项定义为对象，但是C++没有做这样的要求。C#则强调了这一点，它要求所有项都为类的成员。

在C++中，构建程序的语法很大程度上依赖于文件，因为该文件是源代码的一个单元。在程序中可以有多个源文件，每个.cpp文件里都包括了一个#include预处理器，它包括了相关的头文件。当目标文件链接到源文件后，编译进程将分别编译每个源文件，以生成最终的可执行文件。最终的可执行文件不包含原始源文件和目标文件的信息，开发人员需要显式地编写源代码文件的结构。C#编译器注重每个源文件的匹配。源文件可存放在一个或多个文件里，这些对编译器并不重要，因为任何文件都不需要显式地指定为另一个文件。在单个文件里，没有要求项目在引用前必须定义，无论项目在哪出现，编译器都会自动找出它定义的位置。在C#里，链接只限于用装配件中存在的库代码来链接代码，而且在C#中已不存在头文件。

在标准的C++程序中，程序入口点默认为全局函数Main()，在C#中，入口点的原则基本和C++相同，但入口点的Main()函数不再是全局函数，而是一个类的成员，它以一个包含有静态成员方法Main()的类的形式出现。

虽然C++和C#都依赖于扩展库，但是两者有很大的不同。C++依赖的是以继承和模板为基础、库和语言本身相分离的标准库；而C#是依赖以单一继承为基础、相互依靠、关系紧密的.NET基类，而且C#关键字的执行是直接依赖于特定的基类的。例如，为了显示一个消息框，C++语言需要通过非面向对象的方式，调用非标准类库的Windows API函数来实现；而C#则只需通过调用.NET基类的静态方法就能实现，因此C#程序的易理解性也就大大提高了。

同时，C#和C++的编译目标是完全不同的，这也是两者的一大区别。C++代码通常编译成汇编语言，而C#则是编译成为中间语言（IL），IL随后再通过Just-In-Time编译进程转换成本机可执行的代码。IL代码将作为一个装配件存储在一个或多个文件里，一个装配件就是一个单元，IL代码就是在该单元中被打包的，相当于C++编译器创建一个动态链接库DLL或可执行文件。

1.3.2 语言特征

在语法上，C#和C++基本上是完全相同的，它们都忽略语句间的空白，都用分号来分隔语句，用大括号作为块语句的标识。当然，C#作为一门从C++发展来的新语言，它也进行了一定的改进。在C++程序里，定义类的语句结尾要用分号来表示，在C#里，在类定义的结尾就不再需要使用分号来表示；在C++里允许没有任何意义的表达式作为一个独立的语句，在C#里，这样的语句被视为是语法出错，使得C#更符合逻辑思维。

在C++程序里，任何一个被引用的成员都必须在引用前被声明过，而在C#里，由于编译器不再对源文件中成员定义的次序进行约束，所以在程序里就可以引用任何成员，无论该成员是在引用前或引用后声明的都行，这样使得不需再对成员进行向前声明。同时，C#语言和Java语言一样，将项目的定义和声明结合在一起，不像C++语言那样，是在头文件里声明，在cpp文件里定义，而且，C#还自动为代码生成XML格式的文档说明工具。

C++的程序流控制语句在C#里基本都保留着，而且语法和工作方式基本上都是一样的，下面介绍一下C#在这方面的一些改进。为了提高安全性，在C#中数值类型变量和布尔类型变量不能互相转换，因此，在if...else语句和while语句里就不能再使用一个数值变量来作为判断是否执行的条件，弥补了C++语言在这方面的漏洞；C#语言里规定switch语句里的每个case子句都需要显式标明退出语句，而且除非第一条case字句为空，否则不允许由

一条 case 子句自动跳到另外一条 case 子句，同时规定使用 goto 语句来实现 case 子句间的跳转，以此防止程序员要使程序跳出 switch 语句时，却进入另一个 case 子句而造成出错；C#增加了一个可以对数组和集合进行循环操作的流控制语句——Foreach。Foreach 语句可以不通过下标进行显式说明来对一个数组里所有项目或一个包含所有项目的集合进行读取信息的操作。

在定义变量方面，C#基本上保留了 C++语言的模式，但也进行了许多优化和修改。在 C#里没有和 C++等同的全局和静态变量，在 C#里变量是类的成员，使用“字段”来表示，而且 C#对存储在堆栈（数值类型）和栈（引用类型）的数据类型进行了严格区别。C#的数据类型如表 1-1 所示。

表 1-1 C#的数据类型

类别	数据类型	说明
整数	byte	8 位的无符号整数
	sbyte	8 位的有符号整数
	short	16 位的有符号整数
	int	32 位的有符号整数
	long	64 位的有符号整数
	ushort	16 位的无符号整数
	uint	32 位的无符号整数
	ulong	64 位的无符号整数
浮点	float	单精度（32 位）浮点数字
	double	双精度（64 位）浮点数字
逻辑	bool	布尔值（真或假）
其他	char	Unicode（16 位）字符
	decimal	96 位十进制值
	IntPtr 无内置类型	大小取决于基础平台（32 位平台上为 32 位值，64 位平台上为 64 位值）的有符号整数
	UIntPtr 无内置类型	大小取决于基础平台的无符号整数（32 位平台上为 32 位值，64 位平台上为 64 位值）
类对象	object	对象层次结构的根
	string	Unicode 字符的不变的定长串

从上表可以发现，在 C#里定义数据类型要比在 C++里严格得多。C#的每个预定义的数据类型（string 和 object 除外）都有其显式规定的总内存量，因此它们的大小都是固定的，这也是 C#将 sizeof 运算符规定在不安全代码里才使用的一个原因。虽然类型的名称都很相似，但在 C#语言里还是有了改进的。例如，signed 和 unsigned 在 C#里已不再是关键字。

与 C++的一个更大区别是：C#将所有的基本类型都看作是对象，可以在这些对象上调用一些方法，这也反映了 C#和.NET 基类库之间的紧密程度。C#将每个基本类型映射到一个基类上，通过此来对基本数据类型进行编译，而在代码运行时，这些类型是作为底层中间语言类型来执行的，因此将这些基类数据类型当作对象并不会有相关的性能上的损失。C#的类型安全性要比 C++高，因此在 C#里对数据进行转换就没有 C++语言里那样随意。在 C#语言里，显式转换和隐式转换都是正式的转换。定义为隐式转换的数据可以用显式和隐式两种方式执行，而定义为显式转换的数据转换不能使用隐式转换。C#的这种工作方式，是因为 C#对数据类型间的转换规则极其讲究逻辑关系。在 C#里，隐式转换不会导致数据丢失，而显式转换则可能会导致数据的丢失，引发溢出错误、符号错误或部分数据丢失。

C#将字符串作为一种基本数据类型，而且该数据类型能被编译器识别，因此在 C#里对字符串的处理自然要比在 C++语言里简单得多，而且，C#语言不像 C++那样把字符串作为字符串数组来对待。C#字符串包含 Unicode 字符，拥有更多的属性和方法。

C#将所有数据类型又分为两类：值类型和引用类型。在 C#里这两种数据类型的差别和 C++的不同。在 C++中，变量总是隐式地包含数值，除非它被声明为对另外一个变量的引用。在 C#中，值类型实际上包含它本身的值。C#所有的预定义数据类型中，除了 string 和 object 之外，都是值类型，同时是值类型的还有用户定义的结构和枚举类型。而 C#的引用类型更接近于 C++中的指针，C#的引用也可以通过重新赋值来指向不同的数据项。

C#对变量的初始化要求比 C++更加严格。C#规定，若变量没有被显式初始化，作为成员字段的变量将被默认地初始化零，而如果是方法里的局部变量则不会被默认初始化。

C#的方法定义和 C++的函数定义基本相同，但 C#增加了一个限制，就是成员方法不能声明为 const。因为在 C++中，将方法显式地声明为 const 就是说明不能修改方法包含的类实例，但是一个不改变类公共状态的方法通常要改变私有成员变量的值，在 C++程序中，程序员经常要使用 const_cast 操作来回避被声明为 const 的方法就反映了这些实际问题，为了避免这些问题困扰程序员，const 方法在 C#被禁止使用。

类和结构的概念在 C++里经常被混淆，因为两者的区别仅仅是结构中的成员默认为公共类型，类中的成员默认为私有成员。因为两者难以区别，导致许多程序员以不同方式来使用类和结构，他们只将结构使用在数据对象中，而且只包含成员变量，不包含成员函数或显式的构造函数。C#根据人们使用的习惯，对类和结构进行了较大的区分。C#认为类和结构是完全不相同的对象类型。区别如下：

（1）结构是不支持继承的，也就是说一个结构不能被继承，也不能由其他类或结构那里继承而来；类可以继承某个类，但只能派生于一个基类，而接口的派生个数则没有限制。

（2）结构是数值类型，而类是引用类型。

（3）结构里可以组织字段在内存中的布局方式，也可以定义对应于 C++的 union，而且结构的默认构造函数只能由编译器提供，不能替代。

使用 C#语言对类进行定义和 C++语言十分相似，但也有下面这些区别：

（1）在 C#中，不能将方法声明为 inline，因为 C#程序被编译成为中间语言（IL），任何内联是在编译的第二阶段即 Just-In-Time 编译器将 IL 转换成本地机器代码的时候发生，JIT 编译器将会访问 IL 的所有信息，来决定哪些适合被内联，因此这一过程不需要程序员在源代码中作出声明。

（2）在 C#中，方法的执行代码总是与它定义的代码写在一起的，不能将类的执行方式编写在类的外部，而 C++语言允许这样的操作，而且很多 C++程序员也十分喜欢这种代码编写方式。

（3）变量、函数、构造函数、析构函数和运算符重载这些 C++的类的成员类型 C#都包括了，同时，C#还增加了几个新的成员类型：委托、事件和属性。

（4）访问修饰符 C#在 C++的基础上增加了另外两个访问修饰符：internal（只能访问同一装配件中的其他代码）和 protected internal（只能访问同一装配件中的派生类）。

（5）C#可以在类的定义里初始化变量，同时，C#在类定义结束的大括号后不需要加分号，这一点 C++程序员要特别注意。

在类的编程模式上，C#继承了 C++大部分的编程模式，但在析构函数的编程模式上做了重新设计。C#不能保证析构函数被调用后会马上执行；而且除非要除空的是具体的外部源代码，如文件和数据库连接，否则 C#不可以在析构函数里放置代码，而 C++可以。这些并不是 C#语言缺点，恰恰相反，这正是 C#比 C++更优秀的地方。因为 C#中包含有无用存储单元收集器，它是在后台

运行，对无用的内存自动进行回收，因此，对无用内存的清空就不再需要程序员去考虑，这也使得析构函数在C#里没有像在C++里那样重要，同时这也大大提高了程序的性能。

C#和C++一样也包含一些预处理指令，但比C++少得多，C#的其他语言新特性减少了对预处理指令的依赖。

C#扩充了枚举的功能，但它仍然是作为基本的数字类型来执行的，因此在性能上并不会比C++的差。

如果说在语法上C#和C++有区别，那么在工作方式上，两者可以说是完全不同。在C++中，变量（包括类实例和结构）能存放在堆栈或栈中。

通常如果一个变量或一个包含类已经由new分配过，它就存储在栈中，否则就存储在堆栈里。这样，通过选择是否用new来为变量动态分配内存，可以自由选择变量是应该存储在堆栈里还是存储在栈中。由于堆栈的工作方式，只有当对应的变量在作用域内时，存放在堆栈里的数据才会有效。

在C++语言的设计概念里，new运算符就是在堆中请求一段内存空间。而C#是不允许选择为特定实例分配内存的方式，new运算符只是表示调用了一个变量的构造函数。虽然该变量是个引用类型的变量，调用该构造函数也就是实现在堆栈里分配一段空间，但是这样的改进使得编程人员不用对亲自去对空间进行管理和分配，只是在需要一个这样的实例时就生成一个该类型的对象，不用再去考虑如何分配空间的问题。在C#里一个更大的突破就是使用C#语言编程时，程序员无须再亲自动手去清除内存。C#的.NET无用存储单元收集器对代码中的引用进行周期性扫描，以标识当前程序正在被使用的堆的区域，并自动清除不再使用的对象，它相当于在C++语言里delete操作符的作用，因此，C#里也就没有delete这个操作符。

C#在C++的基础上引进了一些新的概念，包括：

（1）委托：C#不再支持函数指针，但其对应的功能并不是被删除，而是以更好的方式——委托来实现相应的功能。委托的主要作用是传递并激活方法的引用，它将引用以一种特殊的类的形式封装到方法里，可以在方法间传递，用来调用包含引用的方法。委托确保了类型的安全，可以防止激活带有错误签名的方法。虽然委托的工作方式和C++指针十分类似，但它的意义在于可以将一个对象引用和一个方法引用合并起来，而且包含了调用类中方法实例的信息。

（2）事件：事件和委托相似，但它支持回调模式。也就是说，当执行一些操作时，客户端在通知服务器的同时将这些操作也告知服务器，它的工作方式和Visual Basic一样。

（3）特性：C#包含了在Visual Basic和COM中广泛应用的这一概念。一个特性其实就是类中的一个方法，但从类的使用者看来，更像是在使用一个字段。

（4）接口：接口的使用和抽象类十分类似，可以包含静态成员、嵌套类型、抽象、虚拟成员、属性和事件。接口的作用是定义类之间相同功能的方法和属性，因此实现接口的任何类都必须提供接口中所声明的抽象成员的定义。接口可以要求任何实现类必须实现一个或多个其他接口；接口可以用任何可访问性来声明，但接口成员必须全都具有公共可访问性；接口可以定义类构造函数，但不能定义实例构造函数。C#接口和COM接口虽然原理是相同的，但C#接口只是简单的方法列表，而在COM接口里还可有其他相关特性。

（5）属性：C#提供一种被称为属性的声明标记进行定义的机制，可以将属性置于源代码的某些实体上以指定附加信息，也可以在运行时通过反射检索属性包含的信息。程序员可以使用属性来创造新的说明性信息种类，然后在运行时通过对属性的访问来决定要执行的代码。

（6）定位线程语句：C#增加lock语句来实现线程同步功能。

（7）反射：C#的代码可以自动获得已编译装配件里的类定义信息，也可以编写类和方法信息的程序。

1.4 C#和Java的比较

初看C#，会发现它和Java非常相似。只要了解它们的历史，就会明白其原因。下面从它们的发展史开始，通过几方面来比较两者的特点：

1.4.1 发展历史

Java是一群C++高手开发出来的平台无关编程语言，所以当时Java语言在语法上基本是保持和C++一致；而C#是微软公司借鉴20多年编程语言开发出来的一门新语言，它在语法上大量借鉴了C++，所以C#语言的语法和C++也基本一致，所以C#和Java两者在语法上的相似也就在情理之中。

C#的运行环境，如内存管理、构架一致性、可伸缩性和基础平台独立性等方面参考了Java语言的JRE（Java虚拟机）运行环境的设计，C#在运行时需要一个称为CLR（公共语言运行环境）；同时C#扩充了CLR的功能，通过运行VOS（虚拟对象系统），提供了对多种语言的支持。VOS提供了一个丰富的类型系统，该系统旨在支持执行多种编程语言的不同类型。

在编译方式上，Java源代码可以被编译成字节代码的一种中间状态，然后由JRE来执行这些字节代码。CLR没有提供虚拟机，但是C#代码也被编译成一种中间状态，称为IL（中间语言）。IL代码被传输到由CLR管理的执行进程上，或是CLR的JIT编译器，将各部分IL代码按要求编译成本机代码。

1.4.2 程序结构

Java和C#都是将源代码编译成为一种中间状态，但两者是不同的。Java中，程序被编译成为字节代码，而且它不再被编译成本机指令，所以它就需要有一个虚拟机作为执行环境来运行程序，因此编译的文件名被限定为写入代码的文件名，而该文件名又限定为文件中公共类的类名。

也就是说，所有的Java文件都被编译成一个带有.class扩展名的字节代码文件才可以在虚拟机上运行。如果同一文件中定义了多个类，那么每个类定义都会生成一个相应的类文件，该类文件的文件名和已定义的类名相匹配。

C#程序编译成IL后，有VES（虚拟执行系统）运行，该系统支持IL、加载受管制的代码和JITters（将中间语言形式的管理代码转换为本机代码）。C#的文件名没有限制在最终可执行程序的命名，可以在编译时用/out选项来改变文件名。如果没有指定输出的文件名，exe文件将采用包含Main方法的源代码文件的文件名，同时DLL文件将采用第一个指定的源代码文件的文件名。实际上，该文件名和文件中提供的类定义无关。C#代码在编译后总会自动捆绑在组件的一个类型中，而且编译单元可包含任何需要的文件和类定义。

C#编译成IL中间状态后，可以与其他.NET语言编写的代码进行无缝交互操作，同时.NET基类为C#提供了一个统一、标准化的资源库，来满足常用功能的需要，例如XML、图形化等，这一优势是Java远远比不上的。

1.4.3 语言特征

C#的语法要比Java强大，因为C#支持运算符重载和类型安

全的枚举；而且，在需要的情况下，还可以将C#程序中选择嵌入式指针和其他不合法的语法，只要把它们放在“非安全”代码块里就可以。同时，在语法上C#还填补了Java一些空门，例如，在Java语言和C#语言中关键字都不是标识符，但在C#里，可以在关键字前加上前缀@，就可以将它作为标识符使用。

C#使用了.NET框架，所以可以提供内置的、类型安全的、面向对象的程序代码，这些程序可以和支持CTS(公共类型系统)的任何语言实现互操作。之所以说是安全的，是因为.NET框架的安全策略的执行使得由.NET框架管制的代码成为安全代码。每个加载的装配件都会遇到安全策略，安全策略将在基于信任的基础上给代码赋予权限，这些信任是以程序的证据为基础的。只有当程序有访问权限时，.NET安全策略才允许程序使用已保护的资源。为了执行这一访问操作，必须使用已保护资源的访问权限。程序请求获得它所需要的权限，.NET应用的安全策略将确定程序被给予了哪种权限。.NET提供了C#程序的访问权限类，每个权限类都可以访问特定的资源。与每个权限类相关的是权限标记枚举，它用来为权限对象指定访问标记。可以使用这些权限来向.NET说明程序需要什么权限，说明程序的用户必须被授予什么级别的权限，而安全策略就是使用这些对象来确定给程序授予什么权限。Java虽然也可以实现C#和C++间的互操作，但它不是类型安全的代码，而且实现相当复杂。

C#的名称空间的命名不受文件夹名的限制，而Java中软件包名是受到限制的。C#的名称空间可以更加灵活地将相关类组合起来。

C#引入了委托概念，通过它来实现函数指针的功能，也就是用具体的签名来封装一个方法。

C#提供了一套丰富的内置值类型，包括类型安全的枚举、结构和内置简单数据类型，这些都远比Java的简单数据类型丰富得多。

C#在引用和值类型之间提供了双向转换，分别成为封箱和拆箱。Java语言没有这个功能。C#支持使用类、字段、构造函数和方法，支持作为描述类型的模板，并能够定义析构函数，能在无用存储单元收集器收集类之前定义方法。

C#为方法参数提供了in、out和ref三种方法，其中in是默认的方法。

C#引入了方法隐藏的概念，通过关键字virtual和override也支持显式的重写。C#提供了属性来代替get和set方法，它们可以安全地访问内部字段。另外，C#允许创建索引符，它可以提供对对象内部字段的索引访问，但C#不能声明抛出异常的方法。

C#提供了预处理器仿真，而且在C#里，已定义的符号可以有条件地包含或不包含程序。C#提供了一个基于权限的安全模式，可以通过编程来控制。

在运行平台方面，虽然C#是依赖于CLR运行环境的，但是因为目前只有Windows版本，因此在平台独立性方面，C#没有Java的优势。

毕竟在理论上所有相同的Java代码可以在任何可以实现Java运行的平台上运行。但.NET完善的资源完全可弥补这一不足，而且它的扩展性很强，因为C#毕竟是在基于GUI的环境下设计的。

1.5 C#的使用

C#的使用可以通过命令行或VS.NET这个IDE环境使用，但两者都是通过对C#编译器的调用来完成客户的需求的。因此，针对C#的使用，只要了解C#编译器的使用就行了。

1. C#编译器简介

C#编译器csc.exe是随MS.NET SDK免费发布的，它可以在DOS命令下被调用，也可以在几个图形化的IDE上被调用。这些IDE包括了一些共享软件，大多数程序员都是使用VS.NET IDE。

编译器的运行由命令行参数控制。如果通过IDE来编译一个C#程序，IDE就会查看其设置，然后动态建立带有命令行参数的命令行字符串，再由该字符串执行csc.exe过程。使用像VS.NET这样的IDE肯定可以节省很多时间，同时也使得操作更加简易，但是了解控制C#编译器的命令行参数还是极其重要的，因为只有了解了才能准确设置参数，而且组织通过脚本、批处理文件等自动建立过程的任务只有了解了这些参数才能完成。

C#编译器不会输出机器代码，而且C#程序只能编译为MSIL，再由MS.NET运行时把IL转换为机器代码。

2. 重要参数的功能

可以用C#创建控制台应用程序、Windows Form应用程序和组件等不同类型的工程。这些具体工程的设置由/target命令行参数来告知C#编译器需要建立什么类型的工程，也就是指定输出文件的格式：

（1）/target:exe 参数告诉C#编译器建立一个控制台应用序。

（2）/target:winexe 参数告诉C#编译器建立一个Windows Form应用程序。

（3）/target:library 参数告诉C#编译器建立一个包含清单配件。

（4）/target:module 参数告诉C#编译器建立一个没有包含清单的装配件。由它建立的没有装配件清单的组合可以被包含到其他包含清单的装配件组件里。

如果编译包括多个具有Main方法的类型，可以通过参数/main:XXX来指定哪个类型包含要用作程序入口点的Main方法，指定是XXX里的Main方法。

参数/debug是告诉编译器生成调试信息并将其放置在输出文件中，可以通过使用该选项来创建调试版本。

C#编译器的其他参数功能如表1-2所示，它们的具体使用在后面章节会有实例。

表1-2 C#编译器的其他参数功能

选项	作用
@	指定响应文件
/?	列出编译器选项到stdout
/addmodule	指定一个或多个模块作为此程序集的一部分
/baseaddress	指定加载DLL的首选基址
/bugreport	创建一个包含信息（该信息便于报告错误）的文件
/checked	指定溢出数据类型边界的整数算法是否将在运行时导致异常
/codepage	指定编译中的所有源代码文件所使用的代码页
/define	定义预处理器符号
/doc	处理XML文件的文档注释
/filealign	指定输出文件中节的大小
/fullpaths	指定编译器输出的文件的绝对路径
/HELP	列出编译器选项到stdout
/incremental	启用源代码文件的增量编译
/lib	指定通过/reference引用的程序集的位置
/linkresource	创建指向托管资源的链接
/NOLOGO	取消编译器版权标志信息
/nostdlib	不导入标准库(mscorlib.dll)
/noconfig	不使用csc.rsp的全局版本或本地版本进行编译
/nowarn	禁用编译器生成指定警告的能力
/optimize	启用/禁用优化
/out	指定输出文件
/recurse	搜索子目录中要编译的源文件
/reference	从包含程序集的文件中导入元数据

续表 1-2

选项	作用
/resource	将.NET 框架资源嵌入到输出文件中
/unsafe	编译使用 unsafe 关键字的代码
/utf8output	使用 UTF-8 编码显示编译器输出
/warn	设置警告等级
/warnaserror	将警告提升为错误
/win32icon	将.ico 文件插入到输出文件中
/win32res	将 Win32 资源插入到输出文件中

3. 命令参数使用实例

实例 1：在 C#编译器里指定源文件和目标文件的命令是：

```
csc SourceFile.cs  /out:TargetFile.exe
```

SourceFile.cs 是源文件；/out:TargetFile.exe 是目标文件的参数设置。

实例 2：响应文件指定的命令是：

```
csc    SourceFile.cs    /out:TargetFile.exe
@<responseFile>
```

@<responseFile>是指定响应文件

1.6 C#常用术语的解析

1. 公共类型系统（CTS）

公共类型系统（CTS）是指中间语言有一组设计合理、内容丰富的预定义数据类型集，通常这些数据类型都在一个类型层次结构中。如果类派生其他类，或包含其他类的实例，它必定需要知道其他类所使用的所有数据类型，因此，一个公共类型规范特别在这种多机开发、运行的环境是十分重要的。而且公共类型系统不但要指定基本数据类型，还要定义一个内容丰富的类型层次结构，该公共类型系统的层次结构反映了中间语言的单一继承的面向对象的方法。

2. 公共语言规范（CLS）

公共语言规范（Common Language Specification）和公共类型系统一起确保语言的互操作性。CLS 是已经建立的最低标准集，所有面向.NET 的编译器都要支持该标准集。因为 IL 是一种内涵非常丰富的语言，为避免编译器编写人员把给定的编译器的功能限制为只支持 IL 和 CLS 提供的部分特性，所以建立 CLS。只要编译器支持 CLS 里定义的内容就可以。

3. .NET 基类库

.NET 基类库是 Microsoft 公司已经编好的一个内容丰富的受管制的代码类集合，它可以完成以前要通过 Windows API 来完成的绝大多数任务。这些类派生于与中间语言相同的对象模型，也基于单一继承性。.NET 基类都可以实例化对象，也可以从中派生类，而且这些基类十分易用。

4. 装配件

装配件是包含编译好的、面向.NET 的代码的逻辑单元。类似于旧式的 DLL 或可执行文件，或者驻留 COM 组件的文件，装配件是完全自我描述的，也是一个逻辑单元而不是物理单元，可存储在多个文件中。

5. 命名空间

命名空间是将相关的类型进行分组的逻辑命名方案。它只是一组数据类型，但命名空间中所有数据类型的名称都会自动加上一个该命名空间的名字作为前缀，而且命名空间可以互相嵌套。通过这种方式，.NET 避免类名冲突。

6. 上下文

上下文是一种属性的有序序列，为实现驻留在环境内的对象定义环境。在对象的激活过程中创建上下文，对象被配置为要求某些自动服务。上下文内可以存留多个对象。

7. 委托

委托是一种引用类型，它是 C++ 函数指针的托管版本。除了引用实例，委托还可以引用静态方法。所有委托都是从 System.Delegate 类继承而来的，并且有一个调用列表，这是在调用委托时所执行的方法的一个链接列表。

8. 装箱

装箱是值类型实例到对象的转换，它暗示在运行时实例将携带完整的类型信息，并在堆中进行分配。Microsoft 中间语言（MSIL）指令集的 box 指令通过复制值类型，将它嵌入到新分配的对象中，将值类型转换为对象。

9. 封装

封装是对象隐藏其内部数据和方法的能力，使得只有对象的预期部分以编程方式可访问。

10. 枚举

枚举是值类型的一种特殊形式，从 System.Enum 继承，并为基础基元类型的值提供备用名称。一个枚举类型包含一个名称、一个基础类型和一组字段。

11. 垃圾回收

垃圾回收是一个跟踪过程，它传递性地跟踪指向当前使用的对象的所有指针，以便找到可以引用的所有对象，然后重新使用在此跟踪过程中未找到的任何堆内存。公共语言运行库垃圾回收器还压缩使用中的内存，以缩小堆所需要的工作空间。

12. 全球化

全球化是设计和开发软件产品以便在多个区域设置中运行的过程。涉及识别必须得到支持的区域设置，设计支持这些区域设置的功能，以及编写在任何受支持的区域设置中都同样很好地运行的代码。

13. 中间语言

中间语言是用作许多编译器的输出和实时（JIT）编译器的输入的语言。

14. 接口

接口是对协定进行定义的引用类型。其他类型实现接口，以保证它们支持某些操作。接口指定必须由类提供的成员或实现它的其他接口。包含方法、属性、索引器和事件作为成员。

15. 本地化

本地化是自定义或翻译特定地区或语言所需要的单独的数据和资源的过程。

16. 托管代码

托管代码是在与公共语言运行库的“合作协定”下运行的代码。托管代码必须提供运行库所需要的源数据，以提供诸如内存管理、跨语言集成、代码访问安全性以及对象生存期的自动控制等各种服务。

17. .NET 框架

.NET 框架是一种用于构建、部署和运行 XML Web Services 及应用程序的平台。它提供效率极高的、基于标准的多语言环境，它能将现有的投资与下一代应用程序和服务集成，并能迅速应对部署和操作 Internet 规模应用程序的挑战。.NET 框架包括三个主要部分：公共语言运行库、一组分层的统一类库和名为 ASP.NET 的 ASP 组件化版本。

18. 反射

反射是在运行时获取有关程序集和它们里面所定义的类型的信息，以及创建、调用和访问类型实例的过程。

19. 正则表达式

正则表达式是查找和替换文本模式的简洁而灵活的表示法。

该表示法包括两个基本字符类型：原义（普通）文本字符，表示在目标字符串中必须存在的文本；元字符，表示在目标字符串中可以变化的文本

20. 安全模式

安全模式是一种特定的版本策略，它要求给定程序集与它在编译时所针对的其依赖项的准确版本一起运行

21. 安全策略

安全策略是管理员制定的活动策略，它基于代码所请求的权限为所有托管代码以编程方式生成授予的权限。对于要求的权限比策略允许的权限还要多的代码，将不允许它运行。

22. 序列化

序列化是指将对象实例的状态存储到存储媒体过程。在此过程中，先将对象的公共字段和私有字段以及类的名称（包括类所在的程序集）转换成字节流，然后再把字节流写入数据流。在随后对对象进行反序列化时，将创建出与原对象完全相同的副本

23. 结构

结构是用户定义的值类型，可以包含构造函数、常数、字段、方法、属性、索引器、运算符和嵌套类型，但不支持继承

24. 非托管代码

非拖托管代码是不考虑公共语言运行库的约定和要求而创建的代码。非托管代码在只有最少的服务（例如无垃圾回收、有限的调试等）的公共语言运行库环境中执行

25. 版本策略

版本策略是指定要绑定到依赖程序集的哪一版本的规则，用配置文件来表达

26. XML

XML 是标准通用置标语言（SGML）的子集，它为在 Web 上传输而优化。XML 提供统一的方法来描述和交换结构化数据，这些数据不依赖于应用程序或供应商

1.7 本书内容概述

本书详细介绍了 C#的 API 核心类库。全书共分 8 部分，分别介绍了.NET 基类库的多个命名空间里方法的使用，并对 C#语言中的一些细节问题加以阐述。本书包含大量实例，介绍了如何使用核心类库。本书浅显易懂，既可作为工具书，也可以作为 C#语言的入门辅导材料，还适用于各种程度的 C#语言开发工作者。