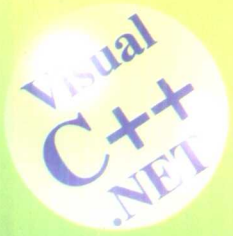
微软ATC统编教材

微软技术培训统编教材

专业程序员成长之路

Visual C++.NET 教程

北京希望电子出版社 总策划
彭明伟 朱诗兵 李启军 等 编 写



北京希望电子出版社
Beijing Hope Electronic Press
www.bhp.com.cn

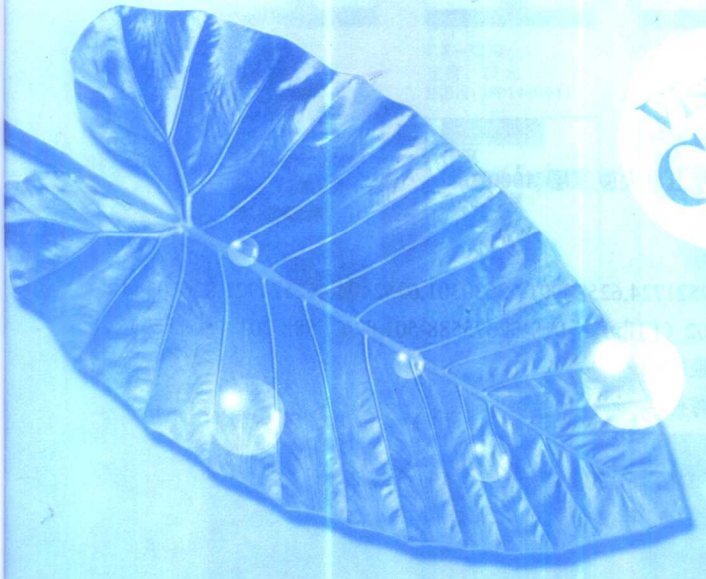
微软ATC统编教材

微软技术培训统编教材

专业程序员成长之路

Visual C++.NET 教程

北京希望电子出版社 总策划
彭明伟 朱诗兵 李启军 等 编 写



北京希望电子出版社
Beijing Hope Electronic Press
www.bhp.com.cn

内 容 简 介

本版书是微软授权培训中心（ATC）的统编教材之一。

本版书通过大量实例较系统地介绍了 Visual C++基础知识及其网络编程、数据库编程应用技术。全书共包含三篇，第一篇为基础知识篇，主要介绍使用 Visual C++编程的有关基础知识，如菜单、键盘及鼠标的使用，Windows 帮助的实现等；第二篇为 Internet 编程，主要介绍了 ActiveX 控件、WinInet 类、WinInet API 类及 WinSockets 类的编程方法及应用；第三篇为数据库管理编程技术的介绍，包括使用 ADO 和 ODBC 编写数据管理程序的方法。在每章的最后附有习题，并在附录中给出习题参考答案。全书的每一种编程技术都给出了具有代表性的应用实例，使读者通过实例的学习，能迅速掌握各种程序开发技术。

本版书不但是微软授权培训中心的统编教材，同时也可作为大中专院校相关专业师生自学、教学参考书和社会电脑培训班的教材。

需要本教程或需要得到技术支持的读者，请与北京海淀 083 信箱北京希望电子出版社（邮编 100080）联系。网址：www.bhp.com.cn，E-mail：lwm@bhp.com.cn。电话：010-62520290, 62630301, 传真 010-62520573。

系 列 书 名 ：微软技术培训（ATC）统编教材
书　　名 ：Visual C++.NET教程
总 策 划 ：北京希望电子出版社
文本著作者 ：彭明伟　朱诗兵　李启军等
责 任 编 辑 ：郭淑珍
出版、发行者 ：北京希望电子出版社
地　　址 ：北京市海淀区知春路63号卫星大厦三层 100080
网址: www.bhp.com.cn
E-mail: lwm@bhp.com.cn
电 话： 010-62520290,62521724,62528991,62630301,62524940,62521921,82610344
（发行） 010-82675588-202（门市） 010-82675588-501,82675588-201（编辑部）
经　　销 ：各地新华书店、软件连锁店
排　　版 ：希望图书输出中心　杜海燕
文本印刷者 ：北京媛明印刷厂
开本 / 规格 ：787 毫米×1092 毫米　1/16　20.25 印张　469 千字
版次 / 印次 ：2002 年 9 月第 1 版　2002 年 9 月第 1 次印刷
印　　数 ：0001-5000 册
本 版 号 ：ISBN 7-900118-44- 6
定　　价 ：25.00 元

前　言

微软公司最新推出的开发工具 Visual Studio.NET，包括 Visual C++.NET、Visual Basic.NET、Visual FoxPro 和 C#等，这些开发工具大多都是在 Visual Studio 6.0 的基础上开发出来的，并且它们将取代旧版本的开发工具，成为开发工具的主流。

本书主要介绍了 Visual C++的编程方法，主要分为三篇：第一篇为基础知识介绍，包括 Windows 编程技术介绍、对话框的使用及 Windows 帮助的实现等；第二篇为 Internet 编程，介绍了 ActiveX 控件、WinInet 类、WinInet API 类及 WinSockets 类的编程方法及应用；第三篇为数据库管理编程，包括数据库管理基础、使用 ADO 建立数据库应用程序的方法及使用 ODBC 建立数据库应用程序的方法。

本书由彭明伟、朱诗兵、李启军、江亚萍等编著，书勤创作室策划、审定。另外参加编写工作和提供技术帮助的还有：庄剑南、许建明、牛静、张向龙、肖斌、胡敏、任玮、张超等。本书的编著由于他们的帮助得以顺利完成，在此表示深深的谢意。

由于时间仓促，书中难免有疏漏之处，敬请各位专家和读者批评指证。

作者

目　录

第一篇　编程基础

第1章　概　　述

1.1　Microsoft.NET——一场新的革命

2000年6月22日，不论对微软（Microsoft）公司还是对整个IT业界都将成为值得纪念的一天。这一天，美国微软公司正式推出了其下一代计算机计划——Microsoft. NET（以下简称. NET）。这项计划将使微软现有的软件在Web时代不仅适用于传统的PC，而且也能够满足目前呈强劲增长势头的新设备，诸如蜂窝电话以及个人数字助理（Personal Digital Assistant，PDA）等的需要。微软还计划通过创建新的工具来吸引软件开发人员和合作伙伴对Microsoft. NET的认同，并且开发出其他基于Internet的服务。

那么，究竟什么是. NET?

请听听微软官员的声音："……因特网的革命……从微软的角度来讲，我们就是要建设一个平台来创建并且支持新一代的应用。……我们必须有一套通用系统服务来支持这样的操作。这种观点就说明，我们还有下一个层次的发展，也就是说因特网下一步的发展，它将使因特网的作用远远超越建一个网站。"

. NET首先是一个开发平台，它定义了一种公用语言子集（Common Language Subset，CLS），这是一种为符合其规范的语言与类库之间提供无缝集成的混合语言。. NET统一了编程类库，提供了对下一代网络通信标准——可扩展标记语言（Extensible Markup Language，XML）的完全支持，使应用程序的开发变得更容易、更简单。Microsoft. NET计划还将实现人机交互方面的革命，微软将在其软件中添加手写和语音识别的功能，让人们能够与计算机进行更好的交流，并在此基础上继续扩展功能，增加对各种用户终端的支持能力。最为重要的是，. NET将改变因特网的行为方式：软件将变成为服务。与Microsoft的其他产品一样，. NET与Windows平台紧密集成，并且与其他微软产品相比更进一步，由于其运行库已经与操作系统融合在了一起，从广义上把它称为一个运行库也不为过。

简而言之，. NET是一种面向网络、支持各种用户终端的开发平台环境。微软的宏伟目标是让Microsoft. NET彻底改变软件的开发方式、发行方式、使用方式等等，并且不止是针对微软一家，而是面向所有开发商与运营商。. NET的核心内容之一就是要搭建第三代因特网平台，这个网络平台将解决网站之间的协同合作问题，从而最大限度地获取信息。在. NET平台上，不同网站之间通过相关的协定联系在一起，网站之间自动交流，协同工作，提供最全面的服务。

某一天，你出差到外地，在机场租借手持电话。在向终端插入自己的IC卡后，自己的

地址簿被自动下载，随即它就变成了你个人专用的 PDA。这不是梦境，这是.NET 为我们描绘的一个未来生活的场景。

人们的需要总是无法满足，我们不断地问自己："我们还应该有些什么？"需求推动着技术的进步。在 21 世纪，Internet 将成为商业活动的主要场所，B2B、B2C 等电子商务的动作方式，一对一营销的经营概念将网络的服务功能提高到了前所未有的程度。微软公司在此时提出.NET 有其深远的战略考虑：

改革商务模型。微软公司感觉到只靠销售软件包的商务模型没有什么前途，该公司打算今后将中心转移到可以在网络上使用"服务"型商务。这样，首要的问题就是解决网络上用来开发并执行"服务"的平台，这就是 Microsoft.NET。

提高软件开发生产效率，并且使应用软件的发布更为容易（再也不用因为 DLL 版本不同而烦恼，也不用重新启动电脑就能够安装应用软件）。

改进用户界面，并能支持多种用户终端。用户界面演进的结果包括两方面的内容，一是完成传统的 PC 界面向基于 XML 的浏览器界面的过渡，二是对自然语言和语音识别的支持，从而使用户与各种终端之间的沟通更加透明，真正达到网络互连的"3A"：Anywhere、Anytime、Any device。

现代人时常有一种困惑，感觉到如今生活在技术与机器架构的丛林中，我们在努力地去适应机器，适应技术，而不是机器和技术适应人类。科技以人为本还只是一个美好的愿望。这是因为我们还不能将控制信息的权利交给那些需要信息的人们。.NET 的出现，意味着只要用一种简单的界面就可以编写、浏览、编辑和分享信息，而且还可以得到功能强大的信息管理工具。由于所有使用的文件都以符合网络协议的格式存在，所以所有的商业用户和个人用户都可以方便地查找和使用其中的信息，任何规模的公司都可以使用相同的工具与它们的供应商、商业伙伴和客户高效地沟通和分享信息，这样就创造出一种全新的协同工作模式。

总之，.NET 战略是一场软件革命。

- .NET 对最终用户来说非常重要，因为计算机的功能将会得到大幅度提升，同时计算机操作也会变得非常简单。特别地，用户将完全摆脱人为的硬件束缚，可以自由冲浪于因特网的多维时空，自由访问、自由查看、自由使用自己的数据，而不是束缚在便携式电脑的方寸空间；可通过任何桌面系统、任何便携式电脑、任何移动电话或 PDA 进行访问，并可对其进行跨应用程序的集成。.NET 能确保用户从任何地点、任何设备都可访问其个人数据和应用程序。
- .NET 对开发人员来说也十分重要，因为它不但会改变开发人员开发应用程序的方式，而且使得开发人员能创建出全新的各种应用程序，大幅度提高软件生产率。
- .NET 把雇员、客户和商务应用程序整合成一个协调的、能进行智能交互的整体，而各公司无疑将是这场效率和生产力革命的最大受益者。

.NET 的核心组件包括：

- 一组用于创建互联网操系统的构建块，其中包括 Passport.NET（用于用户认证）以及用于文件存储的服务、用户首选管理、日历管理以及众多的其他服务。
- 构建和管理新一代服务的基本结构和工具，包括 Visual Studio.NET，.NET 企业服

务器，. NET Framework 和 Windows. NET。

- 能够启用新型智能互联网设备的. NET 设备软件。
- . NET 用户体验。

1.2　.NET 与 C#

多年以前，当微软的组件对象模型（Component Object Model，COM）尚未推出时，软件的复用性对于开发人员仅仅是一种美好的憧憬。成千上万的程序员为了处理、通信接口和不同语言间的冲突而通宵达旦地艰辛劳动，但却收效甚微。COM 的出现改变了这一切。通过将组件改变为通用、集成型的构件，开发人员正逐渐地从过去的繁复编程事务中解脱出来，可以选择自己最得心应手的编程语言进行编程。然而，软件组件与应用程序之间的结合仍然是松散的，不同的编程语言与开发平台限制了模块间的互用性，其结果是产生了日益庞大的应用程序与不断升级的软硬件系统。举个很简单的例子，只用 5 行 C 语言代码就能编写出的一个简单程序，若使用 COM 来编写，需要几百行代码。COM 在带来巨大价值的同时，也大大增加了开发开销。而. NET Framework 的出现使得一切问题都迎刃而解。实际上，在. NET Framework 中，所有的编程语言，从相对简单的 Jscript 到复杂的 C++语言，一律是等同的。

在最近的一段时间里，C 和 C++一直是最有生命力的程序设计语言。这两种语言为程序员提供了丰富的功能、高度的灵活性和强大的底层控制能力。而这一切都不得不在效率上作出不同程度的牺牲。C 和 C++为我们带来了高度的灵活性，又使我们必须忍受学习的艰苦和开发的长期性，许多 C 和 C++程序员一直在寻求一种新的语言，以图在开发能力和效率之间取得更好的平衡。

今天，人们改进、开发出了许多语言以提高软件生产率，但这些或多或少都以牺牲 C 和 C++程序员所需要的灵活性为代价。这样的解决方案在程序员身上套上了太多的枷锁，限制了他们能力的发挥。它们不能很好地与原有的系统兼容，更为令人头痛的是，它们并不能与当前的 Web 应用结合得很好。

理想的解决方案是，将快速的应用开发与对底层平台所有功能的访问紧密结合在一起。程序员们需要一种环境：它与 Web 标准完全同步，并且具备与现存应用间方便地进行集成的能力。除此之外，程序员们喜欢允许在需要时使用底层代码。

针对该问题，微软的解决方案是一种称之为 C#的程序语言。C#是一种现代的面向对象的程序开发语言，它方便程序员能够在新的微软. NET 平台上快速开发种类丰富的应用程序。. NET 平台提供了大量的工具和服务，能够最大限度地发掘和使用计算及通信能力。

由于其一流的面向对象的设计，从构建组件形式的高层商业对象到构造系统级应用程序，C#都将是最合适的选择。使用 C#语言设计的组件能够用于 Web 服务，这样通过 Internet，可以被运行于任何操作系统上任何编程语言所调用。

不但如此，C#还能为 C++程序员提供快捷的开发方式，又没有丢掉 C 和 C++的基本特征——强大的控制能力。C#与 C 和 C++有着很大程度上的相似性，熟悉 C 和 C++的开发人员很快就能精通 C#。

C#在带来对应用程序的快速开发能力的同时，并没有牺牲 C 与 C++程序员所关心的各种“我”。它忠实地继承了 C 和 C++的优点。如果你对 C 或 C++有所了解，就会发现它是那样的熟悉。即使是一位新手，C#也不会给你带来任何其他的麻烦，快速应用程序开发（Rapid Application Development，RAD）的思想与乘法的语法将会使你迅速成为一名熟练的开发人员。

正如前文所述，C#是专门为.NET 应用而开发出的语言。这从根本上保证了 C#与.NET 框架的完美结合。在.NET 运行库的支持下，.NET 框架的各种优点在 C#中表现得淋漓尽致。C#有以下一些突出的特点：

- 简洁的语法；
- 精心地面向对象设计；
- 与 Web 的紧密结合；
- 完整的安全性与错误处理；
- 版本处理技术；
- 灵活性与兼容性。

下面分别对它们进行简要说明。

1. 简洁的语法

在默认的情况下，C#的代码在.NET 框架提供的“可操控”环境下运行，不允许直接地对内存操作。它所带来的最大特色是没有了指针。与此相关的，那些在 C++中被疯狂使用的操作符（如：“::”、“->”、“.”）已经不再出现。C#只支持一个“.”，对于编程人员来说，现在需要理解的一切仅仅是名字嵌套而已。

C#用真正的关键字换掉了那些把活动模板库(Active Template Library，ATL)和 COM 搞得乱糟糟的伪关键字，如 OLE_COLOR、BOOL、VARIANT_ BOOL、DISPID_XXXXX 等等。每种 C#类型在.NET 类库中都有了新名字。

语法中的冗余是 C++中的常见的问题，比如“const”和“#define”、各种各样的字符类型等等。C#对此进行了简化，只保留了常见的形式，而别的冗余形式从它的语法结构中被清除了出去。

2. 精心地面向对象设计

从 Smalltalk 开始，面向对象的话题就始终缠绕着任何一种现代程序设计语言。的确，C#具有面向对象的语言所应有的一切特性：封装、继承与多态。然而，通过精心地面向对象设计，从高级商业对象到系统级应用，C#是建造广泛组件的绝对选择。

在 C#的类型系统中，每种类型都可以看作一个对象。C#提供了一个叫做装箱(boxing)与拆箱(unboxing)的机制来完成这种操作，而不给使用者带来麻烦。

C#只允许单继承，即一个类不会有多个基类，从而避免了类型定义的混乱。C#中没有了全局函数，没有了全局变量，也没有了全局常数。一切的一切，都必须封装在一个类之中。代码将具有更好的可读性，并且减少了发生命名冲突的可能。

整个 C#的类模型是建立在.NET 虚拟对象系统(Virtual Object System，VOS)的基础之

上，其对象模型是.NET 基础架构的一部分，而不再是其本身的组成成分。这样做的另一个好处是兼容性。

借助于从 VB 中得来的丰富的 RAD 经验，C#具备了良好的开发环境。结合自身强大的面向对象功能，C#使得开发人员的生产效率得到极大的提高。对于公司而言，软件开发周期的缩短将能使它们更好地应付网络经济的竞争。在功能与效率的杠杆上人们终于找到了支点。

3. 与 Web 的紧密结合

. NET 中新的应用程序开发模型意味着越来越多的解决方案需要与 Web 标准相统一，例如超文本标记语言(HyperText Markup Language，HTML)和 XML。由于历史的原因，现存的一些开发工具不能与 Web 紧密地结合。简单对象访问协议（SOAP）的使用使得 C#克服了这一缺陷，大规模深层次的分布式开发从此成为可能。

由于有了 Web 服务框架的帮助，对程序员来说，网络服务看起来就像是 C#的本地对象。程序员们能够利用他们已有的面向对象的知识与技巧开发 Web 服务。仅需要使用简单的 C#语言结构，C#组件将能够方便地为 Web 服务，并允许它们通过 Intemet 被运行在任何操作系统上的任何语言所调用。举个例子，XML 已经成为网络中数据结构传送的标准，为了提高效率，C#允许直接将 XML 数据映射成为结构。这样就可以有效地处理各种数据。

4. 完全的安全性与错误处理

语言的安全性与错误处理能力，是衡量一种语言是否优秀的重要依据。任何人都会犯错误，即使是最熟练的程序员也不例外：忘记变量的初始化，对不属于自己管理范围的内存空间进行修改……。这些错误常常产生难以预见的后果。一旦这样的软件被投入使用，寻找与改正这些简单错误的代价将会让人无法承受。C#的先进设计思想可以消除软件开发中的许多常见错误，并提供了包括类型安全在内的完整的安全性能。为了减少开发中的错误，C#会帮助开发者通过更少的代码完成相同的功能，这不但减轻了编程人员的工作量，同时更有效地避免了错误发生。

.NET 运行库提供了代码访问安全特性，它允许管理员和用户根据代码的 ID 来配置安全等级。在默认情况下，从 Internet 和 Intranet 下载的代码都不允许访问任何本地文件和资源。比方说，一个在网络上的共享目录中运行的程序，如果它要访问本地的一些资源，那么异常将被触发，它将会无情地被异常拒绝，若将其拷贝到本地硬盘上运行则一切正常。内存管理中的垃圾收集机制减轻了开发人员对内存管理的负担。.NET 平台提供的垃圾收集器(Garbage Collection，GC)将负责资源的释放与对象撤销时的内存清理工作。

变量是类型安全的。C#中不能使用未初始化的变量，对象的成员变量由编译器负责将其置为零，当局部变量未经初始化而被使用时，编译器将做出警告；C#不支持不安全的指向，不能将整数指向引用类型。例如，当某个对象进行下行指向时，C#将自动验证指向的有效性——C#中提供了边界检查与溢出检查功能。

5. 版本处理技术

C#提供内置的版本控制来减少开发费用，使用 C#将会使开发人员更加轻易地开发和维

护各种商业应用。

升级软件系统中的组件(模块)是一件容易产生错误的工作。在代码修改过程中可能对现存的软件产生影响，很有可能导致程序的崩溃。为了帮助开发人员处理这些问题，C#在语言中内置了版本控制功能。例如：函数重载必须被显式地声明，而不会像在 C++或 Java 中经常发生的那样不经意地被进行，这可以防止代码级错误和保留版本化的特性。另一个相关的特性是接口和接口继承的支持。这些特性保证复杂的软件可以被方便地开发和升级。

6. 灵活性与兼容性

在简化语法的同时，C#并没有失去灵活性。尽管它不是一种无限制的语言，比如它不能用来开发硬件驱动程序，在默认的状态下没有指针等等。但是，在学习过程中将发现，它仍然是那样的灵活。

如果需要，C#允许将某些类或者类的某些方法声明为非安全的。这样一来，将能够使用指针、结构和静态数组，并且调用这些非安全的代码不会带来任何其他的问题。此外，它还提供了一个另外的东西来模拟指针的功能代表（delegates）。再举一个例子：C#不支持类的多继承，但是通过对接口的继承，将获得这一功能。

正是由于其灵活性，C#允许与 C 风格的需要传递指针型参数的 API 进行交互操作，DLL 的任何入口点都可以在程序中进行访问。C#遵守. NET 公用语言规范(Common Language Specification，CLS)，从而保证了 C#组件与其他语言组件间的互操作性。元数据(metadata)概念的引入既保证了兼容性，又实现了变量的类型安全。

1.3 全面了解. NET

C#运行在. NET 平台之上，其各种特性与. NET 有着密切联系。它没有自己的运行库，许多强大的功能均来自. NET 平台的支持。因此，要想真正掌握 C#首先必须了解. NET。下面将介绍 C#的运行环境，重点放在. NET 公用语言运行时环境与公用语言规范上，最后介绍. NET 的开发工具。

1.3.1 NET 结构

. NET 包括 4 个组成部分：

- 虚拟对象系统
- 元数据
- 公用语言规范
- 虚拟执行系统

下面分别对它们进行简要介绍。

1. 虚拟对象系统

. NET 跨语言集成的特性来自于虚拟对象系统(VOS)的支持。在不同语言间进行代码复用和应用集成中所遇到的最大问题，是不同语言类型系统间的相容性问题。可以想像，不

同的语言虽然语法结构大体相同，但数据类型与语言环境本身的各种特点联系紧密，很难想像一种解释性的语言所拥有的数据类型会与一种编译语言相同，而即使相同的数据类型在不同的语言环境中表示的意义也存在差别。例如，同样是整数类型，在 MS SQL 中的长度是 32 位，而在 VB 中却是 16 位，至于日期时间与字符串类型在这方面的区别就更加明显了。

VOS 的建立就是为了改变这种状况。它既支持过程性语言也支持面向对象的语言，同时提供了一个类型丰富的系统来容纳它所支持的各种语言的特性。它在最大程度上屏蔽了不同语言类型系统间的转换，使程序员能够随心所欲地选择自己喜欢的语言(当然，这种语言必须支持. NET 应用)从事开发，保证了不同语言间的集成。

对于过程性语言，它描述了值的类型并指定了类型的所有值必须遵守的规则；在面向对象的语言方面，它统一了不同编程语言的对象模型。每一个对象在 VOS 中都被惟一标识以与其他对象相区别。

2. 元数据

元数据是对 VOS 中类型描述代码的一种称呼。在编译程序将源代码转换成为中间代码时，它将自动生成，并与编译后的源代码共同包含在二进制代码文件中。元数据携带了源代码中类型信息的描述，这在一定程度上解决了版本问题：程序使用的类型描述与其自身绑定在一起。

在公用语言运行时（Common Language Runtime，CLR）定位与装载类型时，系统通过读取并解析元数据来获得应用程序中的类型信息，即时（Just In Time，JIT）编译器获得加载的类型信息后，将中间语言代码翻译成为本地代码，在此基础上根据程序或用户要求建立类型的实例。由于整个过程中，CLR 始终根据元数据建立并管理对应特定应用程序的类型，从而保证了类型安全性。

此外，元数据在解决方法的调用，建立运行期上下文界限等方面都有着自己的作用。而关于元数据的一切都由. NET 在后台完成。

3. 公用语言规范

公用语言规范是 CLR 定义的语言特性集合，主要用来解决互操作问题。如果一个类库遵守 CLS，那么同样遵守 CLS 规范的其他编程语言将能够使用它的外部可见项。

4. 虚拟执行系统

虚拟执行系统(Virtual Execution System，VES)是 VOS 的实现，它用来驱动运行环境。元数据的生成与使用、公用语言规范的满足性检查以及应用程序执行过程中的内存管理均由它来完成。具体说来，VES 主要完成以下功能：

- 装入中间代码。
- 使用 JIT 编译器将中间代码转换为本地码。
- 装入元数据。
- 代码管理服务——包括垃圾收集器和异常处理。
- 定制与调试服务。

- 线程和环境管理。

1.3.2 公用语言运行时环境与公用语言规范

了解了.NET的结构之后，该看看.NET利用其结构创造的运行环境——公用语言运行时环境。它是C#及其他支持.NET平台开发环境的运行基础。具体来说，它为应用提供了以下益处：

- 跨语言集成的能力。
- 跨语言异常处理。
- 内存管理自动化。
- 强化的安全措施。
- 版本处理技术。
- 组件交互的简化模型。

1. 理解CLR

.NET提供了一个运行时环境，叫做公用语言运行时，它管理着代码的执行，并使得开发过程变得更加简单。这是一种可操控的执行环境，其功能通过编译器与其他工具共同展现，代码将受益于这一环境。依靠一种以运行时为目标的（指完全支持运行时环境的）编译器所开发的代码叫做可操控代码。它得益于可操控环境的各种特性：跨语言集成、跨语言异常处理、增强的安全性、版本处理与开发支持、简单的组件交互模型以及调试服务。为了使运行时环境能够向可操控代码提供服务，语言编译器需要产生一种元数据，它将提供使用语言中的类型、成员、引用的信息。元数据与代码一起存储，每个可加载的CLR映像均包含了元数据。运行时环境使用元数据定位并载入类，在内存中展开对象实例，解决方法调用，产生本地代码，强制执行安全性，并建立运行时环境的边界。

运行时环境自动处理对象的展开与引用，当不再使用时负责它们的释放。被运行时环境进行这样的生命期管理的对象称为可操控代码。自动内存管理消除了内存溢出，同时也解决了其他一些常见的语法错误。如果代码是可操控的，仍然可以在需要的时候使用非可控代码，或者在应用.NET的同时使用可控与非可控代码。由于语言编译器只支持自己的类型，比如一些原始类型，用户可能并不总是知道数据是否可控。

CLR使设计跨语言的组件与应用变得更加容易。不同语言设计的对象能够彼此间进行通信，并且它们的行为能够紧密地综合与协调。例如，定义一个类，然后可以在另一种不同的语言中从该类中派生一个类或者调用它其中的一个方法，也可以向另一种语言中类的方法传递该类的一个实例。这种跨语言的集成之所以可能，是因为以运行时为目标的语言编译器与工具使用一种运行时所定义的公用类型系统，它们遵守运行时的规则（公用语言规范）来定义新的类型，生成、使用、保持并绑定类型。

作为元数据的一部分，所有可控组件携带了关于所依赖的组件与资源的信息。运行时环境使用这些信息来保证组件或应用具有需要的所有东西的特定版本，其结果是代码将不会因为版本冲突而崩溃。注册信息与状态数据不再保存在难以建立与维护的注册表中，所定义的类型及附属信息作为元数据被保存，这使得复制与移动组件的复杂程度得到降低。

编译工具用自己的方式向开发人员展现 CLR 的功能。这意味着运行时的一些特性在不同的语言中的表现形式将会有所不同。怎样体验运行时的特性将取决于所使用的语言。比如说，一位 VB 开发人员可能会注意到在运行时环境的帮助下，VB 语言比以前具有更多的面向对象的特性。

2. 可操控执行的含义

前面的叙述多次提到了“可操控”这一概念。这意味着指向的对象在执行过程中完全被运行时环境所控制。在执行过程中，运行时环境提供以下服务：自动内存管理、调试支持、增强的安全性及与非可操控代码的互操作性，例如 COM 组件。

在可控执行进程中的第一步是选择源代码的生成工具。如果希望你的应用拥有 CLR 提供的优势，必须使用一种（或多种）以运行时为目标的语言编译器，例如：VB、C#、VC 的编译器，或者一种第三方编译器如 PERL 或 COBOL 编译器。

由于运行时是一种多语言执行环境，它支持众多的数据类型和语言特性。所使用的语言编译器将决定使用运行时的哪一部分功能子集。在代码中使用的语法由编译器决定，而不是运行时环境。如果组件需要被其他语言的组件完全使用，那么必须在组件的输出类型中使用 CLR 所要求的语言特征。

当完成并编译代码时，编译器将它转换为微软中间语言(Microsoft Intermediate Language，MSIL)，同时产生元数据。当要执行代码时，这种中间语言被 JIT 编译器编译成为本地代码。如果安全策略需要的代码是类型安全的——通常情况下都是如此，JIT 编译器将在编译进程中对中间语言进行类型检查。一旦失败，在代码执行中将会触发异常。

3. CLR 的突出特色

CLR 包含了一个丰富的语言特性集，保证了它与各种程序设计语言的兼容性。在执行过程中管理应用程序的资源是一项单调而困难的工作。它会将注意力从本应解决的问题中引开。而垃圾收集机制完全解决了程序员在编程过程中头痛的问题，跟踪内存的使用，并知道何时将它们释放。

在面向对象的环境中，每种类型都标识了对应用有用的某种资源。为了使用这些资源，需要为类型分配内存。在应用中，访问一种资源要通过以下步骤：

1）为类型分配内存。

2）初始化内存，设置资源的初始状态并使其可用。

3）通过访问该类型的实例来访问资源。

4）卸下将被清除的资源状态。

5）释放内存。

这一看似简单的过程在实际的编程中是产生程序错误的主要来源之一。更可怕的是，内存中的错误往往导致不可预见的结果。如果你有过编程的经验，想想看，有多少次你的程序因为内存访问错误而崩溃？

CLR 要求所有的资源从可操控的堆中分配。当一个进程被初始化后，CLR 保留了一个未被分配的地址空间。这一区域叫做可操控堆。在堆中保留了指向下一个将被分配给对象的堆地址的指针(NEXT)。初始状态下，该指针是保留地址空间的基地址。一个应用使用新

的操作产生对象时，此操作首先检查新对象需要字节的大小是否会超出保留空间。如果对象大小合适，指向下一个地址的指针将指向堆中的这个对象，该对象的构造器被调用，新的操作返回对象的地址。

当一个应用请求建立一个对象时，地址空间可能不够大。堆将发现这一点(通过将新对象的大小与 NEXT 指针相加，并与堆的大小进行比较)，这时垃圾收集器就将被调用。在这里，CLR 引入了“代”的概念。代，指堆中对象产生的先后。这样，垃圾收集器在将发生溢出时回收属于特定的“代”的对象，而不是回收堆中的所有对象。

在各种语言的编译器对源代码进行编译之后，在 CLR 环境中产生的是中间代码(出于兼容性与跨语言集成的考虑)，其内容虽然有效，但在转化为本地代码之前它本身是不可执行的。这就是 JIT 编译器需要完成的工作。

这里需要说明一个问题：为什么要即时编译，而不是一次性地将中间代码文件进行编译？答案很简单：原因在于效率。在大型的应用中，很少会用到程序的全部功能，这种边执行边编译的措施比一次性地完全编译效率更高。

在 Windows 平台中，CLR 带有 3 个不同的 JIT 编译器：

- 默认的编译器——主编译器，由它进行数据流分析并输出经过优化的本地代码，所有的中间代码指令均可被它处理。
- PREJIT，它建立在主 JIT 编译器之上。其运行方式更像一个传统的编译器——每当一个.NET 组件被安装时它就运行。
- ECONOJIT，在并不充分优化的前提下，它能够快速完成中间语言（IL）代码到本地码的转换，编译速度与运行速度都很快。

为了配合编译器的工作，在.NET SDK 安装路径下的 / bin 目录中有一个负责管理 JIT 的应用程序：jitman.exe。具体的使用参见联机帮助。

在当前以组件为基础的系统中，开发人员和用户对于软件版本和发布中存在的问题已经十分熟悉了。当安装了一个新的应用之后，很可能发现原本正常的某个应用程序奇怪地停止了工作。绝大多数开发人员将时间花在了确保所有注册表入口的一致性，以便激活 COM 类上。这就是所谓的“DLL 地狱”。

.NET 平台通过使用集合来解决这一问题。在这里，“集合”是一个专有名词，指类型与资源的发布单元，在很大程度上它等同于今天的 DLL。正像.NET 用元数据描述类型一样，它也用元数据描述包含类型的集合。通常说来，集合由 4 个部分组成：集合的元数据(集合的内部清单)、元数据描述的类型、实现类型的中间语言代码和一组资源。在一个集合中，以上 4 个部分并不是都必须存在，但是集合中必须包含类型或资源，这样集合才有意义。

在.NET 中一个基本的设计方针是使用孤立的组件。一个孤立集合的含义是指一个集合只能被一个应用所访问。在一台机器上，它不被多个应用共享，也不会受其他应用程序对系统更改的影响。“孤立”赋予了开发人员在自己的程序中对代码的完全控制权。任何共享代码都需要被明确地标识。同时，.NET 框架也支持共享集合的概念。一个共享集合指在一台机器上被多个应用共享的集合。共享集合需要有严格的命名规定。有了.NET，应用程序间的共享代码是明确定义的。共享集合需要一些额外的规则来避免我们今天遇到的共享冲突问题。共享代码必须有一个全局惟一的名称，系统必须提供名称保护，并在每次引用共享集合时，CLR 将对版本信息进行检查，此外，.NET 框架允许应用或管理员在明确说

明的版本政策下重写集合的版本信息。

4. 公用语言规范

使被不同语言的编译器所编译的对象能够相互理解的惟一方法，是让所有在互操作过程中涉及的数据类型和语言特性对所有的语言来说是公共的。为了这个目的，公用运行时环境标识了一组语言特征的集合，称为公用语言规范(CLS)。如果组件在应用程序接口(Application Program Interface)中仅使用 CLS 的特征语言(包括子类)，那么该组件能够被任何支持 CLS 的语言所编译的组件访问。所有支持 CLS 并仅使用 CLS 中语言特征的组件被称为符合 CLS 的组件。

设计公用语言规范时遇到的一个最主要的挑战是选择适当的语言特性子集的大小。它应具有完全的表达能力，又应足够小，使得所有的语言能够容纳它。由于 CLS 是关于语言互用性的规范，它的规则仅应用于外部可见的条目中。CLS 假设语言间的互操作性仅在语言集合的边界发生交叉时才是重要的。也就是说，在单一的语言集中对于编程技术的使用没有任何限制。CLS 的规则仅作用于在定义它们的语言集合之外仍然可见的项上。这样就大大缩小了 CLS 的范围，减轻了系统的负担。

在 CLS 中是用 System.CLSCompliantAttribute 类来标识一个集合或者类是否是符合 CLS 规范的。在 System.CLSCompliantAttribute 的构造器中有一个 Boolean（布尔）型的返回值，代表了与之相关联的项是否符合 CLS 规范。

1.3.3 开发工具

.NET 为使用与开发人员提供了功能强大、种类丰富的管理与开发工具，同时这些工具也是. NET 框架提供的服务，正是由于有了它们的支持. NET 才变得如此强大。

（1）Visual Studio. NET：是. NET 的核心开发工具，包括微软提供的各种开发语言，其中有 Visual C#。

（2）Assembly Generation Utility(a1.exe)：用来建立集合的工具。它能够将资源文件或 MSIL 格式的文件转换为带有内容清单的集合。

（3）Windows Forms ActiveX Control Importer(aximp.exe)：完成 COM 类库中类型定义的转换，使 ActiveX 控件能够在 Windows 窗口控件上使用。

（4）Code Access Security Policy Utility(casp01.exe)：在用户与机器水平上修改安全策略。

（5）Software Publisher Certificate Test Utility(Cert2spc.exe)：用于从 X.509 证书中生成软件出版证明书(SPC)。

（6）Certificate Manager Utility(certmgr.exe)：管理证书、证书信任列表和证书回收列表。

（7）Certificate Verification Utility(chktrust.exe)：检查证书签名的合法性。

（8）Runtime Debugger(cordbg.exe)：运行时调试器，是一个命令行程序，帮助开发人员发现和调试基于 CLR 的应用程序中的错误。

（9）Global Assembly Cache Utility(gacutil.exe)：允许浏览与操纵全局集合缓存中内容的命令行程序。