第 1 章　概 述

C++和 Java 是当前最为流行的编程语言，而 C#则是微软新近推出的.NET 编程语言。对这三种语言进行比较和鉴别，不仅有助于编程人员对它们的特点和应用有所把握，而且可以根据它们各自的优劣在编程时有所选择，更好地实现自身的需求。

1.1 技 术 背 景

1.1.1 C++的技术背景

C 语言是一种被广泛应用的程序设计语言，它除了具有规模小、速度快、结构化、程序化、结构灵活等特点之外，也存在着一些不足：

（1）非强制类型，这既是它的长处，也是它的一个不足。从技术上来讲，类型机制反映了一种程序设计语言对强制使用变量类型的严格程度。在 C 中可以将一个整数赋给一个字符变量，反之亦可。这就意味着编程人员应当正确地管理程序中的变量。对有经验的编程人员来说，这没有什么问题，但新手应加以注意，因为这样做可能带来副作用。

（2）缺乏运行时刻检查，在 C 中，因为缺乏运行时刻检查机制，会引起很多神秘和短暂的问题，它们常常不易被编程人员发觉。

（3）传统程序设计技术，也就是面向过程的结构化程序设计技术。在这种设计方式下程序通常包含一个主函数，以及一个或者若干个由主函数调用的函数（子例程）。这种程序设计方法是自顶向下的。在这种模式下，代码和数据是分离的，过程定义要对数据进行处理，但两者不会融为一体，这就给程序的维护带来了不便。当必须增加和删减程序代码时，往往要牵动整个程序。这种方式下的开发时间和调试时间都比较长。

C++语言基于 C，是 C 的一个超级集合，保留了 C 的灵活性，并且具有强有力处理硬软件接口和低层系统程序设计的能力；C++保留了 C 语言的紧凑性和强有力的表达式。更为重要的是 C++提供了支持面向对象程序设计和高层问题抽象的平台，在支持面向对象程序设计方面，C++比 Ada 更前进了一步。在简洁性和模块化方面，C++类似于 Modula-2，而且保留了 C 的高效率和简洁性。

C++是一种混合性语言。它支持用面向对象的程序设计来求解问题，使之产生这个问题全新的面向对象解。实际上，C++语言表现出具有过程程序设计方法和新的面向对象设计方法。这种在 C++中的双重性，对于 C++初学者提出了一种特殊的挑战，即不仅要学习新语言，而且要学习新的问题求解的方法和掌握新的思维方式。

C++是由 BCPL 和 Simula 67 的某些灵感而产生的。它从 Simula 引进了子类（导出类）

和虚拟函数，借鉴了 AIgol 68 操作符重载能力和灵活地把说明紧挨着应用程序首次使用的地方的能力。像其他现代程序设计语言一样， C++语言进化和改良了我们所熟知的高级程序设计语言的某些最佳特征。当然，最接近于 C++的还是 C。

1980 年，C++由贝尔实验室的 Bjarne Stroustrup 创建。最初的动机是打算在效率上改进 Simula 67 语言并采用复杂事件驱动，当时的 C++称为带类的 C（C-with-classes），缺少操作符重载、引用和虚拟函数等许多细节。

1983 年，C++首次推广到外界，到 1987 年夏天，C++仍然在演变。AT&T 决定性地提交出与 C兼容的 C++ ,这样就保留了完整的 C 的几百万行代码及广泛的 C的库和 C 的工具。鼓励 C 程序员学习 C++而不放弃他已经工作多年的过程程序设计语言。这对 C 程序员来说是个大好消息：当从 C 前进到 C++时，他们仍没有放弃在过去工作的领域，从而使语言在发展中保持了稳定性。事实上，C 语言本身影响了 C++的发展。ANSI C 标准草案中包含了某些 C++的关键特征，诸如函数原型，1990 年通过的 ISO C 中也是如此。

C++语言的标准化工作是从 1989 年开始的，现在已建立了灵活的面向对象语言的经验库,并且牢固地与多值继承、封装技术和其他重要领域建立了联系，而且取得新的突破性进展。

C++ 语言支持大型软件系统的开发。在语言中增加了强类型检测以及与 C 语言比较，减少了在 C++软件系统的开发性错误，并提供了对多用户程序设计。

1.1.2 Java 的技术背景

Java 诞生于 1991 年，它是由 Sun Microsystems 公司的一个开发小组开发出来的，该小组隶属于一个最初名为 Green 的项目组。该项目组试图开发一种能够应用于消费性电子产品工业中的独立于平台的软件技术。这个小组创建的语言起初叫做 Oak。

与此同时，一股新的潮流席卷美国，它就是“ Web 漫游 ”。World Wide Web（环球网，WWW），这个应用于 Internet 的数百万个相互链接的 HTML 文档上的名字，突然在大众中变得流行起来。这是因为引入了一种由 NCSA 开发的名为 Mosaic 的图形 Web 浏览器。该浏览器简化了 Web 上的浏览操作，方法是通过把文本和图形组合到一个界面中，无需用户学会许多令人困惑的 UNIX 和 DOS 命令。使用 Mosaic 使得在 Web 上的浏览操作变得非常容易。

从 1994 年起，Oak 技术开始应用于 Web 上。1994 年，两名 Sun 公司的开发人员创建了 HotJava 的第一个版本，当时它被命名为 WebRunner，是一个用于 Web 的图形浏览器，至今仍在使用。该浏览器完全由 Oak 语言编写，后来改称 Java。不久之后，Java 编译器由最初的 C 语言代码改用 Java 语言进行了重写，这证明了 Java 可以作为一种应用程序语言而有效地使用。在 1995 年 5 月的 SunWorld 95 会议期间，Sun 正式推出了 Java。

Web 漫游在数百万的计算机用户间变得极为流行。然而，在 Java 出现之前，Internet 上信息的内容都是一系列乏味的 HTML 文档。Web 用户如饥似渴地期盼着一种交互式的、只需考虑所使用的软硬件平台即可执行的、能够访问多机种网络而不会把病毒传播给多机种网络计算机的应用程序。Java 能够创建这类应用程序。

1.1.3 C#的技术背景

C 和 C++作为开发商用和企业软件时使用最广泛的编程语言之一，为开发者提供了大

量的细致灵活的控制，但这种灵活性是以生产的成本作为代价的。为此微软提出了一种叫 C# 的语言。C#是一种现代的、面向对象的语言，它使开发人员能够在微软新的 .NET 平台上快速建立广泛的应用，其提供的工具和服务能充分发掘系统的计算和通能力。

C#语言将作为 Microsoft Visual Studio 7.0 的一部分而推出，同时这个开发环境还支持 Visual Basic、Visual C++以及脚本语言 VBScript、Jscript 等。微软的 Next Generation Windows Services（NWGS）平台提供了一个执行引擎和一个丰富的类库来支持上述语言的解释和执行。对于 C#开发人员来说，这就意味着即使 C#是一种新的语言,它也可以使用 Visual Basic、Visual C++所使用的类库，而自身并不用包含类库。

因为其优良的面向对象设计，在构建从高级业务对象到系统级应用的各种不同组件时，C#是一个首要的选择。使用简易的 C＃语言构造，组件可以被转换为 Web 服务，从而允许从运行在任何操作系统上的任何语言中跨越 Internet 调用它们。

不仅如此，C#的设计为 C++程序员带来了快速的开发能力，而不用牺牲 C++已有的功能和控制能力。通过这种继承，C#高度地保持了与 C 和 C++的一致。开发者只要熟悉 C 和 C++语言就可以快速地掌握 C#，并写出更多的 C#应用程序。

1.2 语 言 特 性

1.2.1 C++ 的语言特性

C++是从 C 语言发展起来的，在其发展过程中就包含了对 C 的一些特性的改进：

（1）注解。C++中引入了一种新的注解到行末的分界符“//”。

（2）枚举名。一个枚举的名即一个类型名，这就简化了原先枚举类型的表示符，不必在枚举类型名前加一个 enum。

（3）结构名或者类名。结构或者类的名即是类型名，这是 C++发展而来的，不需在一个结构或者类名前加限制符 struct 或者 class。

（4）分程序声明。C++允许将声明放在分程序内和代码语句之后，这就允许编程人员将标识符在离它的首次使用点最近的地方加以说明。它甚至允许在循环控制变量结构中的形式定义被声明。

（5）匿名联合。不具有名字的联合可以在定义一个变量或域的任何地方定义。利用这一特性，可以使一个结构的两个或多个域之间共享内存，从而节省内存。

（6）显式类型转换。可以用一个预定义类型或者用户定义的类型的名作为一个函数，将数据从一种类型转换至另一种类型。在特定的情况下，这种显式类型转换机制可以用来作为强制类型转换的替代。

（7）函数重载。在 C++中，利用说明符重载后，函数就可以使用相同的名字，并可根据参数的个数和类型来区别每个重载的函数。

（8）缺省的函数参数值。在 C++中，可以为函数的尾部参数赋予缺省值。有了这一特性后，可以用少于形式参数个数的实参来调用函数，所缺少的尾部参数即取它们的缺省值。

（9）inline 说明符。利用 inline 说明符可以指示编译程序在一个函数被调用处对该函数

进行扩展替换。

除此之外，C++在面向对象方面有了很大的增强：

（1）类构造和数据封装性。类构造是面向对象程序设计的基本工具。类定义可以封装所有的数据声明、初始值以及操作集，以实现数据抽象。对象可以被声明为属于某个给定的类，还可以向对象发送消息。此外，一个指定类的对象可以包含其自身的、关于该类的数据表示的私有部分和共有部分。

（2）struct 类。C++中的 struct 是类定义的子集，没有私有或保护部分。这种子类可以包含数据和函数。

（3）构造方法和析构方法。构造方法和析构方法用来对某一指定类的一个对象内定义的数据进行初始化。当一个对象被声明时，初始化方法就被激活。当从声明该对象所在作用域内退出后，析构方法就自动对该对象占用的内存进行释放。

（4）消息。对象是面向程序设计的基本成分。对对象的操作是通过向它们发送消息来完成的。对象也就是声明为某一个给定类的变量，只需利用与调用函数类似的机制，即可向对象发送消息。可以向一个对象发送的消息集在有关该对象的类描述中指定。在响应一条消息时，对象要根据消息的性质来确定采用哪一种动作比较合适。

（5）友元。数据隐藏和数据封装意味着不可以直接访问对象内部的结构。对于类之外的任何函数来说，类的私有部分通常是完全不可见的。但是， C++也确实允许方法或类之外的其他函数被声明为类的一个友元。友元关系透过类的保护墙，允许访问类的私有数据和方法。

（6）运算符重载。在 C++中，程序人员可以取编译程序员提供的预定义运算符和函数集，或取用户定义的运算符和函数，并赋予它们多重含义。

（7）派生类。派生类可以视为指定类的子类，从而形成了一个抽象层次。派生类对象通常继承其父类的全部或部分方法，还可以将这些继承来的方法与自身所包含的新方法综合起来。所有的子类对象都包含来自父类中的数据域，以及它们自己的私有数据。

（8）使用虚函数的多态性。多态性涉及一种父类和子类的树结构。树中的每个子类可以接受一条或多条相同的消息。在这棵树中，当某个类的一个对象接受一条消息时，该对象就要确定对于指定子类的一个对象来说适合于该消息的特定应用。

1.2.2 Java 的语言特性

Java 是一种优秀的编程语言。它具有以下特点：

1. 简单

Java 最初是为对家用电器进行集成控制而设计的一种语言，因此它必须简单明了。 Java 语言的简单性主要体现在以下三个方面：

（1）Java 的风格类似于 C++，因此，C++程序员可以很快就掌握 Java 编程技术。

（2）Java 摒弃了 C++中容易引发程序错误的地方，如指针和内存管理。

（3）Java 提供了丰富的类库。

2. 面向对象

面向对象可以说是 Java 最重要的特性。 Java 语言的设计完全是面向对象的，它不支持类似 C 语言那样的面向过程的程序设计技术。Java 支持静态和动态风格的代码继承及重用。

3．分布式

Java 包括一个支持 HTTP 和 FTP 等基于 TCP/IP 协议的子库。因此，Java 应用程序可凭借 URL 打开并访问网络上的对象，其访问方式与访问本地文件系统几乎完全相同。

4．健壮

Java 致力于检查程序在编译和运行时的错误。类型检查帮助检查出许多开发早期出现的错误。Java 自己操纵内存减少内存出错的可能性。Java 还实现了真数组，避免了覆盖数据的可能。

5．结构中立

另外，为了建立 Java 作为网络的一个整体，Java 将它的程序编译成一种结构中立的中间文件格式。只要有 Java 运行系统的机器都能执行这种中间代码。Java 源程序被编译成一种高层次的与机器无关的 byte-code 格式语言，这种语言被设计在虚拟机上运行，由机器相关的运行调试器实现执行。

6．安全

Java 的安全性可从两个方面得到保证。一方面，在 Java 语言里，像指针和释放内存等 C++功能被删除，避免了非法内存操作。另一方面，当 Java 用来创建浏览器时，语言功能和浏览器本身提供的功能结合起来，使它更安全。

7．高性能

如果解释器速度不慢，Java 可以在运行时直接将目标代码翻译成机器指令。 Sun 用直接解释器一秒钟内可调用 300,000 个过程。翻译目标代码的速度与 C/C++ 的性能没什么区别。

8．多线程

Java 的多线程功能使得在一个程序里可同时执行多个小任务。因为 Java 实现了多线程技术，所以比 C 和 C++更健壮。多线程带来的更大的好处是更好的交互性能和实时控制性能。

1.2.3 C#的语言特性

1．生产力和安全

新的 Web 经济要求所有商业部门比从前更快地响应竞争的威胁。要求开发者在更短的时间内生产发行更多的程序版本，而不是单一纪念物似的版本。 C＃的设计考虑到了这些因素，它的设计帮助开发者用更少的代码行和更少的出错机会做更多的事情。

（1）遵循新的 Web 设计标准

新的应用程序开发模型意味着越来越多的解决方案需要使用新的 Web 标准，如：HTML、XML 和 SOAP（Simple Object Access Protocol）现存的开发工具都是在 Internet 之前或之初开发的。因此，它们总是不能很好地适应新的 Web 技术。

C#程序员可以用一个扩展的框架来构建微软 .NET 平台上的应用。C#包括内建的支持使任何组件转换为一个能在 Internet 上运行任何平台上的任何应用中被调用的服务。

对程序员来说，更出色的是这个 Web 服务框架能使现存的 Web 服务看起来像本地的 C#对象，因此允许开发者权衡利用它们已有的面向对象的编程技能与现存的 Web 服 务 。

当然，还有更多的特点使得 C#成为主要的 Internet 开发工具。例如，XML 是新出现的在 Internet 中传递结构化数据的方法。这种数据集通常很小。为了提高性能， C#允许 XML 数据被直接映射到一个结构数据类型而不是一个类。在数据量不大时，这是一种更有效的处

理方式。

（2）消除重要的编程错误

即便是专家级的 C++程序员都避免不了最简单的错误，如忘记初始化一个变量，这些简单的错误常常导致不可预见的问题，它们长时间隐蔽，不易发现。一旦程序投入生产运行后，要排除哪怕是最简单的编程错误，都要付出昂贵的代价。

C#现代的设计排除了大多数普通的 C++编程错误。例如：

- 垃圾清理减轻了程序员自己管理内存的负担。
- 在 C#中的变量是自动被环境初始化的。
- 变量类型是安全的。

（3）依赖内建的转换支持降低开发成本

更新软件组件是一个容易出错的任务。修订代码版本无意中可能会改动程序的语义。为了帮助程序员解决这个问题，C#语言中包含了转换支持。例如，与 C++和 Java 不同，方法重载必须显式进行；这有助于防止代码错误和保留转换的灵活性。有关的特点还有本地的接口和接口继承支持。这些特点进化并发展了复杂的框架。

总之，这些特点使一个工程的后继版本的开发方法更加健壮，同时为成功的应用降低了整体的开发成本。

2．功能、表现和灵活性

随着公司制作商业计划的水平越来越高，抽象的商业过程与实际软件实现之间的紧密连接已成一种必然趋势。但大多数语言工具没有一种简单易行的方式来链接业务逻辑和代码。例如，开发者可能使用代码注释来说明哪个类构成主要的抽象业务对象。

C#语言允许分类，扩展能应用于任何对象的元数据。一个工程的构建者能够定义特定领域的属性并将它们应用到任何语言的类元素、接口等等。然后开发者编程测试每个元素的属性。这就简化了自动化工具的编写，而自动化工具将保证每个类或者接口被正确表示为特定抽象商业对象，或简化了创建基于对象特定属性的过程。紧密连接定制元数据和程序代码有助于加强意料的程序行为与实际的实现之间的联系。

可管理的类型安全环境对大多数企业应用是适合的。但是现实世界中的经验告诉我们，有些应用延续需要“本地”代码，其原因要么是因为性能问题，要么是因为与现存的应用程序接口问题。这样的情况强迫开发者使用 C++来开发，即使他们喜欢使用生产性更好的开发环境。

C#通过下面的方式解决了这个问题：包含组件对象模型（COM）的本地支持和基于 Windows APIs 的支持。

在 C# 中,每一个对象自动地成为一个 COM 对象。开发者不再需要显式地实现 IUnknown 和其他 COM 接口，而是将它们内置了。同样，C#编程能在本地使用现存的 COM 对象，与它们用什么语言创建的无关。

对于需要这种特性的开发者来说，C# 包含了一个特别的特性：那就是一个程序可以调用任何本地 API 在一个特别标记的代码块中，允许开发者使用指针和传统的 C/C++特点，如自己管理内存和指针算法。

相对于其他开发环境，这是一个突出的优点。它意味着 C# 程序员能建立他们自己现存的大量 C和 C++代码，而不用丢弃它们。

COM 支持和本地 API 访问支持，目的是向开发者提供了重要的能力和控制，而不用离

开 C#环境。

具体来说则包括如下一些特点：

（1）现代性

你在学习 C#上所付出的努力将是一项巨大的投资，因为 C#被设计成为开发.NET 应用的首选语言。你会发现许多在 C++中必须由你自己来实现或者干脆没有的特征，都成为基础 C#的一部分了。

小数类型对于企业级编程语言来说是很受欢迎的一个附加类型。你可以使用一个新的 DECIMAL 数据类型来进行货币计算。如果你不喜欢这个简单类型，可以很容易地为你的应用特别设计一个新的类型。

在 C#中，指针不再是你的编程武器了，所以你也就不必负责整个内存的管理了。.NET 平台环境提供了自动内存管理机制负责你的 C#程序的内存管理工作。

对于 C++程序员来说，异常处理是 C#的一个主要特性，但这决不是什么新鲜事。然而，和 C++不同的是，异常处理是跨语言的（运行环境的另一特点）。在 C#之前，你必须对付离奇的 HRESULTs，现在不必要了，稳健的基于异常处理的错误处理机制能做好这些。

（2）面向对象

C#作为一种新的语言理所当然地支持面相对象的所有关键概念：封装、继承和多态性。整个 C#的类模型是建立在.NET 虚拟对象系统（VOS，Virtual Object System）之上的。对象模型是基础构架的一部分，而不是编程语言的一部分。

在面向对象的程序设计中，没有全局函数、变量或者常数。这些必须被封装在一个类中，或者作为一个实例成员（通过类的一个实例对象来访问），或者作为一个静态成员（通过类型来访问),这会使你的 C#代码具有更好的可读性，并且减少了发生命名冲突的可能性。

在类中定义的方法缺省情况下不是虚拟的（不能被派生类所覆盖），这一做法的要点是可以去掉另一个产生错误的来源——对方法的错误覆盖。如果一个方法可以被覆盖，那么它必须是有一个显式 virtual 修饰符。这样不但可以减少虚函数表的长度，还能保证正确的版本处理行为。

对于 C++程序员而言，已经熟悉了使用访问权限修饰符为类的成员指定不同的访问级别。C#也支持私有（private）、保护（protected）和公共（public）访问权限修饰符，而且增加了第四个：internal。在实际的程序设计中，很多情况下只需要从一个类派生，从多个基类派生所带来的问题比这种做法所能解决的问题更多，这就是 C#只允许一个基类的原因，但是完全可以用接口来实现同样的多重继承的功能。

（3）类型安全性

在 C++中，定义了指针之后，你可以自由地把它指向任何一个类型，包括做一些相当愚昧的事情，比如将一个整型指针指向一个双精度型指针，只要内存支持这一操作，它就会凑合着工作，而往往是这种情况导致了程序的错误。

为了避免上述问题，C#实施了最严格的类型安全来保护它自身及垃圾收集器。在 C#中，必须遵守关于变量的一些规定：

不能使用未初始化的变量。对于对象的成员变量，编译器负责把它们置零。局部变量需要你自己处理。如果使用了未初始化的变量，编译器就会提醒你。这样做的好处是：你可以避免使用了一个未被初始化的变量带来的严重后果。

C#不支持不安全的指向。不能将整数指向引用类型，并且 C#会时时验证指向的有效性。

边界检查是 C#的一部分。当数组实际上只有 n-1个元素时，不可能使用超过它的元素。

算术运算可能溢出结果数据类型的范围。 C#允许在应用级或者语句级检查这种操作中的溢出，当溢出发生时会抛出一个异常。

C#中传送的引用参数是类型安全的。

（4）版本技术

在过去的几年中，几乎所有的程序员都和 DDL 打过交道，也备受折磨。产生这个问题的原因是许多计算机上安装了同一 DDL 的不同版本。有时老的应用在新的 DDL 版本上还侥幸可以运行，但大多数情况下，它们都会崩溃。版本处理技术是当前面临的一个重要问题。

C#考虑到了这种要求，它将尽其所能支持这种版本处理功能。虽然 C#自己并不能保证提供正确的版本处理结果，但它为程序员提供了这种版本处理的可能性。有了这种支持，开发者可以确保当它开发的类库升级时，会与已有的客户应用保持二进制兼容。

（5）兼容性

C#不是存在于一个封闭的世界里。它允许你通过遵守重要的.NET 公用语言规范（CLS Common Language Specification)访问不同的 API。CLS 定义了遵守这些规则的语言间相互操作标准。为了增强 CLS 的兼容性，C#编译器会检查所有公开输出项所遵守的条件，发现不符合规则的就会报错。

当然你也可能希望能够访问老的 COM 对象，.NET 平台提供了对 COM 的透明访问。另外，C#还支持 OLE 自动化，并且不需要考虑细节问题。

C#同样允许与 C 风格的 API 进行相互操作。动态链接库（DDL）的任何入口点（以风格给出）都可以在应用程序中进行访问。这种访问本地 API 的特性叫做平台调用服务。

（6）简单性

C#同 C++相比，其简单性就是一个优点，它通过增加了许多新特性，同时舍弃了一些老的东西，对 C#的简单性做出了贡献。

没有指针是 C#的一个显著特性。在缺省情况下，你使用一种可控制的代码进行工作，此时，一些不安全的操作，如直接内存操作，将是不允许的。在 C++中，有“ ::, . ”和“ → ”操作符，分别用于名空间、成员和引用操作，在 C#中，去掉了别的操作符，只支持一个“ . ”. 程序员现在需要理解的一切就是名字嵌套的概念了。

在 C#中不再需要记住那些源于不同处理器结构的神秘类型了，包括可变长的整数类型。C#通过提供了一个统一的类型系统解决了这个问题。这个类型系统使每种类型都可以被看作是一个对象。与别的语言不同的是，把单个类型看作对象并不会增加执行上的困难。

（7）灵活性

虽然 C#代码缺省是安全模式，也可以声明某些类或者仅仅是类的某些方法为非安全的，这一声明使你能够使用指针、结构和静态分配的数组。安全和不安全代码两者都在可操作空间中运行。这就意味着从安全代码调用不安全代码不会有什么问题。

1.3 面向对象方法

面向对象开发方法是以对象为中心的开发方法，是从客观世界中抽象出来的软件开发

的新的思维方法。面向对象开发方法可以促进软件工作者在应用领域中最大限度地发挥他的才能和思维能力。

面向对象方法有如下的基本特征：

（1）对象是数据和有关操作的封装体，突破了传统的将数据与操作分离的模式，较好地实现了数据的抽象。

（2）面向对象方法的继承性体现了概念分离抽象。在对象继承结构上，下层对象继承上层对象的特征（属性和操作），因而面向对象方法便于软件的演化和增量式扩充。

（3）面向对象方法用消息将对象动态链接在一起。与传统的模块调用不同，面向对象方法采用了灵活的消息传递方式，从而便于在概念上体现并行和分布式结构。

（4）面向对象方法具有信息隐藏性。对象将其实现细节隐藏在它的内部，因此无论是对象功能的完善扩充，还是对象实现的修改，其影响仅限于该对象内部，而不会对外界产生影响。这就保证了面向对象软件的可构造性和易维护性。

面向对象开发是不依赖于程序设计语言的概念化开发过程，面向对象开发本质上是一种新的符合人的思维的方式，而不是程序设计技术；最大好处是帮助研究和开发人员清晰地表达用户所需的抽象概念，并能够将用户与研究、开发人员之间的信息相互表达、沟通。面向对象开发方法也是在软件开发各个阶段中（如规格说明、分析、文件接口）的中间媒介。面向对象开发可以有不同的目标，其中包括通常的程序设计语言和数据库，以及面向对象语言。

面向对象方法学有以下几个阶段：

（1）分析阶段

分析人员从问题的陈述开始，逐步建立具体特性的客观世界模型，分析过程就是提取一个系统的需求的过程。分析人员必须与用户一道工作，了解问题和分析现状，因为用户此时对问题的陈述很少是完整和正确的。

分析的模型应该是简要的，在这个分析模型中的对象是应用领域的概念，而不是计算机的实现概念。一个好的分析模型，应该是能够被应用方面的专家（不是程序员）了解或者能够指出不对之处。

（2）系统设计阶段

系统设计者对全局结构的高层进行决策。在系统设计阶段，目标系统是由分析的结构和建议的结构的子系统所组成的。系统设计者必须决策优化的性能特性是什么，选择涉及到的问题的策略，并对资源分配进行试验。

（3）对象设计阶段

对象设计者建立基于分析模型的设计模型，但不包含实现细节。设计者根据系统设计期间建立的策略原则，增加设计模型的细节；对象设计的中心是对每个实现的类，建立它们的数据结构和算法。分析对象的类是有重要意义的，它们可以增强要实现的关键数据结构，以及对选取的算法测试其性能。应用领域对象和计算机领域的对象，都用面向对象概念和表示进行描述，当然这两个领域中存在着不同的概念方案。

（4）实现阶段

在对象设计期间开发的对象类和它们之间的联系，最后要进入到具体程序设计语言、数据库或硬件的实现阶段。所有关键性决策，已在设计阶段作出，这时程序设计只是开发周期的一部分。语言对设计决策只有一定程度的影响，但设计的最后决策并不依赖于程序设计语

一个好的软件工程的实施，实现阶段是非常重要的，它不仅能够非常简洁地实现系统设计时所确定的目标，而且能够保留系统的灵活性和可扩充住。

面向对象概念可以应用整个系统开发生命周期，同样的类能够支持所有开发阶段，当然，在最后阶段需要增加具体的实现细节。譬如，窗口（Window）的分析视图和实现视图，在它们的不同抽象层次有着不同的表示，代表了不同的目的，但都是正确的。

1.4 面向对象程序设计

面向对象的程序设计不同于传统的过程化方法。它需要一种新的设计思想，常常不易为传统的面向过程的程序员所掌握。如果读者已为 Microsoft Windows 编写或检验过程序，将会尝到其中使用的一种原理的趣味——程序包含一系列相关的对象。

面向对象语言的消息与 Windows 和 Presentation Manager 程序是共同的。在面向对象设计中，对象不仅拥有数据（成员数据），而且作用了数据的方法（成员函数），这两项被结合成一个工作原理，简单地说，对象包括数据和操作这些数据的方法。

使用面向对象设计，对程序员有三个直接好处。第一个是程序维护。程序易读、易理解，并且面向对象设计能控制程序员只能看到必要的细节。第二个益处是程序的修改。读者向类如数据库程序中添加或删除，可以仅通过增加或删除对象。新对象可以从父对象那里继承，仅需增删不同的条目。第三个好处是，可以多次使用对象。可以将一个设计好的对象存在工具箱中。这样只需少量的代码改动，就可将它括入新代码。

1.5 小　　结

面向对象程序设计是软件系统的设计与实现的新方法。这种方法是通过增加软件可扩充性和可重用性来改善并提高程序员的生产能力的，并能控制维护软件的复杂性和软件维护的开销。而 C++、Java 和 C#是三种在不同条件背景下推出来的面向对象程序设计语言，它们分别经历了不同的发展过程，并且具有自身的特点。

第 2 章　数据类型和变量

本章主要介绍了 C++、Java 和 C#三种语言中的数据类型和变量的基本知识。主要侧重于 C#中的数据类型和变量的讲述。最后，对这三种语言中的数据类型和变量中的差异进行了对比和分析。

2.1　C++的数据类型和变量

2.1.1　基本类型

C++有一个基本类型集，它与计算机中最常用的基本存储单元及使用它们的最普通方式相对应：

Char
Short int
Int
Long int

用以表示不同大小的整数；

float
double

用以表示浮点数；

unsigned char
unsigned short int
unsigned int
unsigned long int

用以表示无符号整数、逻辑值和数组等。

整类型 char 最适合于保存和操作给定计算机上的字符，一般是一个 8 位字节。C++对象的大小以 char 大小的倍数来表示，故定义 sizeof（char）=1。一个 char 是有符号的或无符号的整数要取决于硬件。当然，unsigned char 总是无符号的，用它可产生更便于移植的程序，但用它代替简单的 char 会使性能明显下降。

提供多种整型、多种无符号类型和多种浮点类型，其原因在于允许程序员去利用硬件特征的长处。在多数机器上，存储要求、存储访问时间和运算速度对各种不同的类型存在着显著的差别。例如，熟悉了机器，通常为特定变量选择合适的整数类型就很容易了。编写真正可移植的低级代码是困难的。基本类型的大小应满足下式：

1＝sizeof(char)＜＝sizeof(short)＜＝sizeof(int)＜＝sizeof(1ong)

因此，假定一个 char 可保存范围为 0~127 中的整数（它总是能保存机器字符集中的一个字符),一个 short(短整型)或一个 int 至少有 16 位，int 的大小适合于整型算术，以及 long（长整型）至少有 24 位，这通常是合理的。假定太多会有风险，甚至单凭经验的办法也不能适合于任何事物。

2.1.2 隐式类型转换

在赋值语句和表达式中，基本类型可随意混用，即任何可能的情况下都被转换而不致漏掉信息。

也有一些情况信息会漏掉甚至曲解。将某种类型的值赋给另一种用较少的位即可表示的类型的变量，这是出毛病的潜在根源。例如：假定下面程序片段在具有二进制补码表示的整型和 8 位字符的机器上执行：

```
Int  i1=256+255;
Char ch=i1;
Int  i2=ch;
```

在赋值语句 ch＝il 中最高有效位漏掉了，ch 将存入“全 1”(即八个 1）的位模式,当它向 i2 赋值时没有办法使其值变为 511！i2 的值又会是多少呢？在 DEC VAX 机上，char 是有符号的，答案就是一 1;在 AT&T 3B20 机上，char 是无符号的，答案为 255。C++ 没有检查这类问题的运行机制，一般情况下，要在编译时进行检查是很困难的，所以程序员必须仔细。

2.1.3 派生类型

从基本类型（和用户定义类型）可派生其他类型，要运用的声明运算符如下：

```
*    指 针
&  引 用
[ ]    数组
()     函数
```

也可使用结构定义机制，如：

```
Int *a;
float v[10]l
char *p[20];
void f(int);
struct  str  { short length;    char * p; } ;
```

利用这些运算符构成类型的规则基本思想是派生类型的声明要反映它们的用途，例如：

```
Int  v[10];
I= v[3];
Int * p;
I= * p;
```

要想了解派生类型表示法的所有问题，其症结在于弄清以下事实： *和＆是前缀运算符，[]和（）是后缀运算符，所以必须使用括号来表达难以判定运算符优先级的类型。例如，由

于[ ]的优先级高于*：

```
int *v[10];
int (*p) [10];
```

多数人只要记住最普通的类型是什么样子就足够了。

要把每一个想引入程序的名字都加上一个声明是很乏味的，尤其是它们的类型一样的时候，因而，可以在一个声明中声明几个名字。声明中仅包含一个用逗号分隔的名字表，以此代替单个名字。

2.1.4 void 类型

void 类型是 C++ 的一项创新，在 C 语言中原来是没有的，后来却又被 C 所接受。

void 的第一层意思是指没有数值的数据类型。没有任何返回值的函数应该被说明成 void 类型：

```
void fn(int, *);
```

这个语句说明 fn( )是一个函数,以一个指向 int 类型的指针作为输入参数，并且不返回任何值。在 fn( )函数体内,可以使用 return 语句，但 return 后面不能有任何表达式(C++ 对此检查很严格)。将 C 程序转化为与 C++兼容的最常见的工作就是将所有没有返回值的函数说明成 void 类型。

void 也常常出现在函数说明中，用来表示函数不需要任何入口参数。

```
int g(void);
```

此语句说明 g( )是一个不需要任何入口参数的函数,并且返回 int 类型的值。在 C++ 中也可以用如下语句来描述上面等同的情况：

```
int g();
```

因为在 C++ 中，参数列表为空，意味着没有入口参数（传统的 C 中这两者是不相同的)。

不能把数据说明成 void 类型，因为将变量说明成没有任何值是没有意义的。但可以用 void 来说明 void 指针。void 指针在 C++ 中是一种普通指针类型，如下语句创建了一个普通指针：

```
void *ptr;
```

指针 ptr 可以被赋给指向任何类型的指针。如 fptr 是一个指向 float 的指针，则下面的语句是可行的：

```
ptr=fptr;
```

C++ 同样可以保证赋给 void 指针的指针值可以按原类型被转换回去，但仅此而已，不可能再对它进行什么算术运算，也不保证将它转换成其他类型，它更不可能再属于 void 类型，只能将它转换回原来的类型：

```
Fptr=(float *)ptr;
```

2.1.5 指针

对于多数类型 T，T *是指向 T 的指针类型，即一个 T*类型的变量保存有一个 T 类型对象的地址，对于指向向量和函数的指针，需要更为复杂的表示：

```
int *pi;
```

```
char **cpp;
int(* vp)  [10];
int(* fp)  (char, char*);
```

在指针上的基本运算是引用，即引用由指针指向的对象。这种运算也叫做间接运算。引用运算符是（前缀）单目 *。例如：

```
char c1='a';
char *p=&c1 ;
char  c2=*p;
```

p 所指向的变量是 c1，存在 c1 中的值是'a'，所以赋给了 c2 的*p 值为 'a'。

另外，在指针上也可实施某些算术运算。

2.1.6 数组

数组的说明形式为：type 说明符<常量表达式>

此处说明符指的是数组名，用来说明 type 类型的元素组成的数组。在 C++ 中，一个数组可看作是可足够容纳所有元素的一片连续的空间。

如果数组说明中用常量表达式来说明数据项个数，那么此表达式的值应该是一个正整数，它表示数组的元素个数。数组元素编号从 0 到元素个数－1 为止。

可以通过说明数组类型的数组来构造多维数组，如二维数组是一维数组的数组，例如 5 行 7 列的二维数组说明为：type alpha[5][7];

在有些上下文中，多维数组说明中的第一个数组说明中的括号内没有表达式，这样的数组是不确定大小的。这在合法的上下文中可以这样做。所谓合法上下文是指那种不需要为数组保留空间的情况。例如一个数组说明为 extern，则既不需要说明此数组的实际维数，也不需要说明数组各维的大小。作为对 ANSI C 的一个特殊扩展，C++ 允许用一个不确定大小的数组作为一个结构的最后一个成分。除了为保证数组被正确定位而增加的空白外，这样的数组不增加结构的大小。这样的结构通常用于动态分配中，但数组的实际大小必须明确地加在结构大小上以保证适当地保留空间。

除非数组类型表达式中包括有 sizeof 或＆操作符，否则数组类型表达式都被转换成指向数组第一个元素的常量指针。

也可以通过指针访问数组元素，例如：

```
char str [40]，  *p1；
p1＝Str;
```

此时 p1 被置成 str 中第一个数组元素的开头地址。为了访问 str 中的第 5 个元素，可以写成 str[4]或*（p1+4）。

使用指针来访问数组通常是很有效的。如果数组要以严格递增或递减的顺序去访问，则使用指针既快又方便；但如果数组要随机地被访问，那么使用数组下标会更好些。

2.1.7 结构

结构是一种派生类型，一般用来表示用户定义的一组命名成员的集合。结构成员可以

为任何类型，既可以是基本类型，也可以是派生类型，它们可以按任一顺序排列。结构的成员还可以是不允许在其他地方出现的位域。

在 C++ 中，结构类型被当作类类型（有某些区别：缺省访问类型为公共的，缺省的基类类型也是公共的）。这就允许通过 C++访问说明符 public，private 和 protected 对结构成员的访问进行更高级的控制。除了这些可选的访问机制和列出的例外情况外，下面关于结构的文法表示和使用的讨论也适于 C 和 C++。

1．结构的说明

结构说明的一般形式为：

```
struct struct  name
 {
   type_1 field_1；
type_2 field_2;
… …
type_n field_n ；
}；
```

关键字 struct 标记这是一个结构的说明。

也可以在定义一个结构时不用结构名作为此结构的标记，这称为无标记名结构。无标记结构说明一般是将结构说明和说明结构的对象放在一起。

用花括号括起来的结构成员说明表说明所有结构成员的名字和类型，结构成员可以为任何类型，但有个例外：首先结构成员类型不能和正在定义的结构具有同样的结构类型，但结构成员可以是指向正在定义的结构的指针。

2．结构的名字空间

结构标记名和联合、枚举标记名共享同样的名字空间，这意味着在同样的名字空间中这些标记名应该是惟一的。但结构、联合、枚举标记名，在标号名字空间、成员名字空间和单名空间可不需要用不同的标识符。

在给定的结构或者联合中，成员名应当是惟一的。但它们可以和其他结构或者联合中的成员同名。例如：

```
goto s
…
struct s
{
   int s;
float s;
} s;
union s
{
   int s;
   float f;
} f;
struct t
{
   int s
} s
```

3．结构成员在内存中的安排

结构在内存中按成员顺序，从左到右，从低字节到高字节意依次存放。例如：

```
struct mystruct
{
  int i;
  char str[21];
  double d ;
}
```

则 s 在内存中存放形式为：2 个字节存放整型 i，21 个字节存放字符串 str，8 个字节存放双精度 d

结构对象在内存中存放形式依赖于 C++中字定位选择项，若为 off 则 s 将占 31 个字节连续空间。若选择 on 和_a 编译命令，C++将按下列方式对结构成员定位：结构的存放从子边界开始；所有的非字符成员都从相对结构开始处的偶地址处存放；如果必要的话，在结构的最后加一个字节以保证整个结构占据偶数个字节空间。

4．位域

C++提供了访问字节内每一位的能力，这就是位域。位域是一种特殊的结构成员，它定义了所占用的维数。可以定义从 1 到 16 位的 signed 或 unsigned 型的成员作为位域，位域可以按如下形式说明：

类型说明符<位域标识符>：宽度；

此处类型说明符可以为 char、unsigned char 或 int、unsigned int 类型。位域标识符为位域名称，这是可选的。宽度表示它所占用的位数。位域在内存中从低位到高位分配在一个字节中。宽度表达式 width 不能缺少，且只能取值为 0 到 16 的整常数。

如果没有位域标识符，但它仍然占用 width 说明的位数，只是此位域不能被访问，这往往用于位模式匹配而跳过这些不用的位。

2.1.8 联合

联合也是派生类型，它的语法形式及基本特征和结构相同，不同之处在于联合在某一时刻只允许有一个成员是“活动”的。联合的存储空间为其成员最大域所要求的存储空间。任何时刻只能存储其中一个成员。

```
Union myunion
 {
  int i;
  double d;
  char ch;
} mu, *muptr=&mu ;
```

mu 为联合 myunion 类型，可以用来存放 2 个字节的整数 i，也可以用来存放 8 个字节 double 或单字节 char 的 ch，但某一时刻只能为其中的一个，也就是说不同的数据类型存入同一内存地址。

sizeof（union myunion）和 sizeof（mu）都返回 8 个字节。

但是，若 mu 含一个整型对象，那么 6 个字节是无用的；若 mu 只用于 char，则 7 个字节不用。

可以用成员选择符“ . ”和“ —> ”访问联合的成员，但应小心使用。

示例：

```
mu.d = 4.016;
printf ("mu.d = %f\n", mu.d); .
printf ("mu.i = %d \n", mu.i); .
mu.ch = 'A';
printf ("mu.ch= %c \n". mu.ch); .
printf ("mu.d =%.f\n". mu.d); ,
muptr—>i = 3;
printf ("mu.i x=%d \n", mu.i); ,
```

第二个 printf 是合法的，因为 mu.i 是整型。但是 mu.i 位模式对应的是在这之前对 double 类型所赋的值，因此不能按原来整型量的存储模式来理解。

经过适当转换后，联合的指针也就指向它的每个成员，反之亦然。

总的说来，联合的说明与结构的说明大致相同，不同点在于：

（1）联合可以包括位域，但只有一个处于活动状态，它们都从联合的起始点开始存放。

（2）和 C++ 的结构不同，C++ 中的联合不能使用类访问修饰符 public、private 和 protected，联合的所有域都是 public。

（3）联合只能通过第一个成员进行初始化。

（4）在 C++ 中，联合不能从任何类中派生，且它本身也不能作为基类，但联合可以有构造函数。

（5）C++ 中，无名的联合不能有成员函数。

2.1.9 引用

引用是给对象一个替换的名字，引用的主要用途是为用户定义的类型指定操作。引用还可以作为函数参数使用。引用表示为 x&，它的意思是应用 x。引用必须初始化，要注意引用初始化做的事和给定赋值不同。

尽管运算符可以出现在引用上，但它并没有对应用进行操作，例如：

```
int ii=0;
int& rr=ii;
rr++;
```

是合法的，但 rr++ 并不增加 rr 的值，++ 只作用在 int，发生在 ii 上。因此，在初始化之后引用的值是不会改变的，它总是初始化所标明的对象。为了得到用引用 rr 表示的指向某对象的指针可以写作 &rr。

引用还可以用来定义一个函数，此函数可以用在赋值语句的任何一边。

2.1.10 存储类

C++ 支持四种存储类说明符，它们是：Auto、Register、Static、Extern。

存储说明符位于变量声明的前面，通知编译程序该变量如何存储。用 auto 和 register 说明符声明的变量具有局部生存期；用 static 或 extern 说明符声明的变量具有全局生存期。