计 算 机 科 学 丛 书

C++ 编程思想

Thinking in C++

(美) Bruce Eckel 著

刘宗田 邢大红 孙慧杰 等译

机械工业出版社
China Machine Press

Prentice Hall

计算机科学丛书

C++编程思想

（美）Bruce Eckel 著

刘宗田 邢大红 孙慧杰 等译

袁兆山 潘飚 冯鸿 等校

机械工业出版社
China Machine Press

本书作者根据自己学习C++的亲身体会及多年教学经验，用简单的例子和简练的叙述讲解C++编程，别具特色。

全书共分十八章，内容涉及对象的演化、数据抽象、隐藏实现、初始化与清除、函数重载与缺省参数、输入输出流介绍、常量、内联函数、命名控制、引用和拷贝构造函数、运算符重载、动态对象创建、继承和组合、多态和虚函数、模板和包容器类、多重继承、异常处理和运行时类型识别。

本书作为正式教材和自学用书均非常优秀，作为程序设计者的参考用书亦极为合适。

本书版权登记号：图字：01-98-2511

图书在版编目(CIP)数据

C++ 编程思想/（美）埃克尔（Eckel, B.）著；刘宗田等译. -北京：机械工业出版社，2000.1

(计算机科学丛书)

书名原文：Thinking in C++

ISBN 7-111-07116-6

Ⅰ.C… Ⅱ.① 埃… ② 刘… Ⅲ.C语言-程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字(1999)第64224号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码100037）

责任编辑：周　桦

北京牛山世兴印刷厂印刷・新华书店北京发行所发行

2000年1月第1版・2001年10月第10次印刷

787mm×1092mm 1/16・27.5印张

印数：47 001-52 000册

定价：39.00元

译 者 序

在计算机书店中，关于C++的书俯拾皆是。留心研究，不难发现，这些书大体可分为两类：一类是手册性书籍，基于某种特定C++编译系统版本的使用指南；另一类是教程类用书，从语法、语义到语用由浅入深地讲解。

现在献给读者的这本译著《C++编程思想》（原名为Thinking in C++）与众不同，虽然它也是第二类C++书籍，但其内容、讲授方法、选用例子和跟随的练习，别具特色。原书作者不是按传统的方法讲解C++的概念和编程方法，而是根据他自己过去学习C++的亲身体会，根据他多年教学中从他的学生们的学习中发现的问题，用一些非常简单的例子和简练的叙述，阐明了在学习C++中特别容易混淆的概念。特别是，他经常通过例子引导读者从C++编译实现汇编代码的角度反向审视C++的语法和语义，常常使读者有“心有灵犀一点通”的奇特效果，这在以往的C++书中并不多见。

译者认为，无论是作为正式教材还是作为自学用书，这本书都是非常优秀的。特别是对于已经熟悉了C或其他过程语言编程的程序员，这本书更加适宜。经验表明，对过程语言用得越熟练，接受面向对象概念就越困难。但是，有了这本书，情况会另当别论。

本书主要由刘宗田、邢大红、孙慧杰、潘飚、冯鸿、许东、姜川、刘莹、李连生、何允如、贾亮、童朝柱等翻译，由刘宗田、冯鸿通编整理，由袁兆山、潘飚、冯鸿、张卿、王锐等审校。参加本书翻译和整理工作的还有李心科、陈凯明、孙志勇、秦宗贵、谢志鹏、邵堃、姜桂华等。

由于时间和水平所限，翻译错误在所难免，恳请读者指正。

译　者

1999年8月

前　言

与任何人类语言一样，C++提供了一种表达思想的方法。如果这种表达方法是成功的，那么当程序变得更大和更复杂时，该方法应当明显地表现出比其他语言更容易和更灵活等优点。

不能只把C++看作是一组性能的汇集，因为有些性能不能单独使用。如果我们不只是用这种语言编写代码，而是用它思考“设计”问题，那么就能综合使用这些性能。而且，为了用这种方法理解C++，我们必须首先掌握C和一般的编程问题。这本书讨论的是编程问题，讨论为什么它是问题以及用C++解决编程问题所采用的方法。因此，在每一章中所解释的一组性能，都是建立在我用这种语言解决一类特殊问题所采用方法的基础上的。用这种方法，我希望一点一点地引导读者，从掌握C开始，直到使C++在读者的头脑中成为他自己的语言。

我将始终坚持一种观点：读者应当在头脑中建立一个模型，以便于理解这个语言，直到炉火纯青。如果遇到难题，他可以将问题纳入这个模型，推导出答案。我将努力把已经印在我脑海中的见解传授给读者，正是这些见解，使我能开始“用C++思考”。

1. 预备知识要求

在我的第一本C++书[1]中，我把C和C++放在一起讲解，很多人说他们喜欢这样，但我发现，现在已经有了许多关于C语言的好的指南和好的教师。我觉得，如果读者已经学过C，并且至少能用它顺利地读程序，那么这本书就会更有用。在本书中，我将致力于解决C++中的难点。另一方面，就像能在小说中通过上下文直观地学习许多新名词一样，从本书的文字中读者也可以学习到大量关于C的知识。

我还发现，我没有耐心讲授同时覆盖C和C++的课程。我能保持三天的高度积极性和旺盛精力，然后就开始低落。似乎也就在这时，听众的头脑也装满了，所以这也就是我的小型培训研讨班的期限。对于大型班，我们根本不可能完成练习，在当前材料的情况下，我可以在2~3天的教学中涵盖大部分内容。

我尽力不用任何特定厂商的C++版本，因为对于学习语言，我不认为特定实现的细节像语言本身一样重要。而大部分厂商的文档适合于他们自己的特定实现。

2. 学习C++

我希望这本书的读者有和我进入C++时相同的情况，因为一个C程序员对于编程持有实在而执着的态度。但糟糕的是，我的背景和经验是在硬件层的嵌入式编程方面。在那里，C常常被看作高层语言，它对于位操作是低效率的。后来我发现，自己甚至不是一个好的C程序员，平时掩盖着对malloc() & free()、setjmp() & longjmp()结构和其他“复杂”概念的无知，开始触及这些主题时就回避了，而不是努力去获取新的知识。

在我开始致力于学习C++时，当时唯一像样的书是Stroustrup的“自学专家指南”[2]，因此我只好自己消化基本概念。这引出了我的第一本C++书[3]，基本上就是直接把我头脑中的经验

---

[1] Bruce Eckel, Using C++. Osborne/McGraw-Hill, 1989. 第二版更名为C++ Inside & Out, Osborne/McGraw-Hill, 1993. 这本书供已经用过其他语言，但还没有用过C语言设计程序的读者使用。

[2] Bjarne Stroustrup, The C++ Programming Language, Addision-Wesley, 1986(第一版)。该书第二版由机械工业出版社翻译出版。

[3] Using C++, 出处同[1]。

倒出来而写成的。它被用来作为读者指南，引导程序员同时进入C和C++。这本书的两个版本[1]都得到了积极的反响，我至今仍然认为它是有价值的。

几乎就在《Using C++》出版的同时，我开始讲授这门语言。讲授C++已经变成了我的职业。自1989年以来，我看到了世界各地听众的昏昏欲睡的样子、茫然不知的面容和困惑莫解的表情。当我对一个较小的组进行室内训练时，在练习过程中又发现了问题。即便那些面带微笑和会心点头的学生，实际上对许多问题也还是糊涂的。通过近三年主持“软件开发会议”的C++分组讨论会，我发现，我和其他讲演者都有一种倾向，即过快地向听众灌输了过多的主题。后来，我做了一些努力，通过区别对待不同层次的听众和调整表述内容的方法，尽量吸引听众。也许这是过分的要求，但是因为我是一个坚持传统教学的人，所以希望通过努力，使每一个人都能跟得上教学进度。

有一段时间，我编写了大量的教学简报。这样，我结束了通过实验和重复方式进行学习（在设计C++程序的过程中这也是很有用的一项技术）的阶段。最后，从我多年的教学经验中总结出来的所有内容，形成了一门课程。在课程中，我用一系列离散的、易分解的步骤和举办手把手的小型研讨班的形式解决学习中的问题（理想的学习情况），并在每个短课后面跟随着练习。

这本书是在两年课程教学的基础上写成的，并且书中的内容在许多不同的研讨班上通过了多种形式的实际测试。我从每个研讨班上收集反馈意见，不断地修改和调整内容，直到我感觉到它已经成为一本很好的教材为止。但这本书不仅仅是研讨班的分发教材，而且我在其中放入了尽可能多的信息，在结构上使得它能引导读者顺利地通过当前主题并进入下一个主题。另外，这本书也适合于自学读者，能帮助他们尽快地掌握这门新的编程语言。

3. 目标

在这本书中，我的目标是：

1) 以适当的进度介绍内容。每次将学习向前推进一小步，因此读者能很容易地在继续下一步学习之前消化每个已学过的概念。

2) 尽可能使用简短的例子。当然，这有时会妨碍我解决“现实世界”的问题。但是，我发现，当初学者能够掌握例子的每个细节，而不受问题的领域所影响时，他们通常会更有兴趣。另外，选材对代码的长短也有严格的限制，以便代码能在课堂上使用。为此，我无疑会受到使用“玩具例子”的批评，但是我宁愿承受这一批评，因为这样更有利于教学。如果想要得到更复杂的例子，可以查阅《C++ Inside & Out》[2]的后面几章。

3) 仔细安排描述内容的顺序，不让读者看到还没有揭示的内容。当然，这不是总能做得到的，在这种情况下，将会给出简明的介绍性的描述。

4) 只把对于理解这门语言是重要的东西介绍给读者，而不是介绍我知道的所有的内容。我相信，存在着“信息重要层次结构”。有些内容是95%的程序员不需要知道的，这些东西只会迷惑人们，增加他们对该语言复杂性的感觉，以致于C++现在被认为比ADA还复杂。举一个C语言的例子，如果我们记住运算符优先表（我是记不住的），就可以写更漂亮的代码。但是，如果一定要这样做，反而会使代码的读者或维护者糊涂。所以可以忘掉优先级，当不清楚时使用括号。对于在C++中的某些内容可以取同样的态度，因为我认为这些内容对于写编译器的人比对于程序员更重要。

5) 保持每一节的内容充分集中，使得授课时间以及两个练习之间的间隔时间不长。这不仅

[1] Using C++ 和 C++ Inside & Out，出处同第iv页[1]。

[2] 出处同前言注[1]。

能使听众思想活跃，在研讨班上精力集中，而且会使他们有更大的成就感。

6) 帮助读者打下坚实的基础，使得他们能充分地理解这些问题，从而可以顺利地转向学习更困难的课程和书籍。

4. 章节安排

C++是一个在已经存在的文法上面增加了新的不同性能的语言（因此，它被认为是混合的面向对象的语言）。由于很多人学习走了弯路，因此，我们已经开始探索C程序员转移到C++语言性能层上的方法。因为这是C语言训练思想的自然延伸，所以我决定去理解和重复相同的道路，并通过引出和回答一些问题来加速这一进程，这些问题是当我学习该语言时遇到的和听众在听我的课时提出来的。

设计这门课时，我始终注意一件事：人们学习C++语言的方法。听众的反馈意见帮助我认识到哪些部分是很难学习的，需要额外解释。在这个领域中，我曾经雄心勃勃，一次讲解包括了太多的内容。通过讲解过程，我知道如果包括大量新性能，就必须对它们全部作出解释，而且学生也特别容易混淆。因此，我努力一次只介绍尽可能少的性能，理想的情况是每章一次只介绍一个。

本书的目标是在每一章中只讲授一个性能，或只讲授一小组相关的性能，用这种方法，不会有多余的性能被关联。这样，在进入下一章的学习之前，学生可以对自己的当前知识融汇贯通。为了实现这些目标，我离开C性能比我希望的还要远。例如，我宁肯马上用C++ 的输入输出流库，而不用熟悉的C语言printf( )函数族，这样就需要过早地介绍这一主题，因此，在较早的章节中，允许C库函数与它们并用。这样做的好处是C程序员不会因为用到了许多未解释的C++性能而困惑，因此，对该语言的介绍将是平稳的，并且能反映出消化和理解这些性能的方法。

下面是对该书章节内容的简要说明。

第1章　对象的演化。当项目对于维护变得太大和太复杂时，就产生了软件危机。人们常说，“我不能得到希望做的项目，即便能，它们也太昂贵”。这引出了一些问题，在本章中我将讨论这些问题，并且讨论面向对象程序设计思想和如何运用这一思想解决软件危机问题。另外，在这一章中我还将阐述采用这种语言的好处，提出关于如何转入C++世界的建议。

第2章　数据抽象。C++的许多性能都围绕着一个根本的思想：创建新数据类型的能力。这不仅可以提供超级代码组织，而且为许多强大的OOP能力奠定了基础。在本章中，读者将可以看到如何用将函数放入结构内部的简单过程而实现这一思想，可以看到如何具体地完成这样的过程和创建什么样的代码。

第3章　隐藏实现。通过说明结构中的一些数据和函数是private（私有的），可以把它们设置为对于这个新结构类型的用户是不可见的。这意味着能够把下层实现和客户程序员看到的接口隔离开来，因此更容易改变具体实现，而不影响客户代码。另外，C++还使用关键词class作为描述新数据类型的更奇特的方法。单词“object”的意思并不神秘，在某种意义上它就是变量。

第4章　初始化与清除。C语言的最通常的一类错误是由于变量未初始化而引起的。C++的构造函数使得程序员能保证他的新数据类型的变量（即类的对象）总是能被恰当地初始化。如果他的数据类型还要求某种方式的清除，他可以保证这个清除动作总是由C++的析构函数来完成。

第5章　函数重载和缺省参数。C++打算帮助程序员建立大而复杂的项目。这时，可能会引进使用相同函数名的多个库，还可能会在同一个库中选择具有不同含义的相同的名字。C++采用函数重载使这一问题容易解决。重载允许当参数表不同时重用相同的函数名。缺省参数使我

们能以不同的方法调用同一个函数，并能自动地对某些缺省的参数提供缺省值。

第6章　输入输出流介绍。输入输出流是提供基本I/O工具的C++库。输入输出流希望以I/O库代替C库的STDIO.H。I/O库更容易使用，更灵活、且更可扩充——也就是可以让它和新类一起工作。对于标准I/O、文件I/O、内存格式化，本章将教会读者如何最好地使用已存在的输入输出流库。

第7章　常量。本章包含了const和volatile关键字，它们在C++中有另外的含义，特别是在类的内部。本章还说明const的含义在类的内部和外部有何不同，如何在类的内部创建编译时常量。

第8章　内联函数。预处理宏省去了函数调用开支，但是也排除了有价值的C++类型检查。内联函数具有预处理宏和实际函数调用的所有好处。

第9章　命名控制。在程序设计中，创建名字是基本的活动，而当项目变大时，名字的数目是没有限制的。C++允许在名字创建、可视性、存储代换和连接方面有大量的控制。这一章将说明如何用两个技术控制名字。第一，static关键字用以控制可视性和连接，我们还将研究它对于类的特殊含义。另一个在全局范围内更有用的控制名字的技术是C++的namespace（名字空间）性能，它允许把全局名字空间划分为不同的区域。

第10章　引用和拷贝构造函数。C++指针的作用和C指针一样，而且具有严格的C++类型检查的好处。继Algol和Pascal之后，C++使用了“引用”，这是处理地址的新方法，当我们使用平常的符号时，引用让编译器处理地址操作。读者还会遇到拷贝构造函数，它控制对象通过传值方式传送给函数或从函数中返回。最后，本章还将解释C++ 指向成员的指针（pointer-to-member）。

第11章　运算符重载。这个性能有时被称为“文法糖”。由于允许运算符也是函数调用，这使得程序员使用他的类型在文法上更有趣。在这一章中，读者将学到，运算符重载只是不同类型的函数调用，学会如何写自己的运算符重载，特别是当参数、返回类型混合使用和让运算符成为成员或友元时。

第12章　动态对象创建。一个空中交通系统将处理多少架飞机？一个CAD系统将需要多少种形状？在一般的程序设计问题中，我们不可能在程序运行之前知道所使用的对象的数量、生命期和类型。在这一章中，我们将学习C++的new和delete，研究如何漂亮地通过在堆上安全地创建对象而解决上述问题。

第13章　继承和组合。数据抽象允许由草稿创建新的类型。通过组合和继承，程序员可以用已存在的类型创建新的类型；用组合方法，可以以老的类型作为零件组装成新的类型；而用继承方法，可以创建已存在类型的一个更特殊的版本。在这一章中，读者将学习这一文法，学习如何重定义函数，理解构造和析构对于继承和组合的重要性。

第14章　多态和虚函数。单靠读者自己，可能要花九个月的时间才能发现和理解作为OOP基础的这一性能。而通过一些小而简单的例子，读者可以看到如何通过继承创建一个类型族，看到在这个类型族中如何通过公共基类对这个族中的对象进行操作。关键字virtual允许笼统地谈论这个族中的所有对象，这意味着大批代码将不依赖于特殊类型的信息，因此，程序扩充性好，构造程序和维护代码也变得更容易和代价更低。

第15章　模板和包容器类。继承和组合允许重用对象代码，但不能解决有关重用的所有问题。模板为编译器提供了一种在类或函数体中代换类型名的方法，由此而重用源代码。这就支持了包容器类库的使用，包容器类库是使我们能快速而精力充沛地开发面向对象程序的重要工具。这一章给出了这个基本主题的详尽阐述。

第16章　多重继承。乍听起来这很简单：一个新的类是从一个以上已存在的类继承得来的。然而，由此引起的基类对象的二义性和多拷贝问题会给我们造成困难。这些问题靠虚基类解决，但是还有更大的问题：什么时候要用到它？当需要通过用一个以上公共基类操作对象时，多重继承才是唯一必需的。这一章解释多重继承文法，并且介绍另外的方法，特别是，介绍如何用模板解决一个共同的问题。用多重继承修理一个“受损”的类界面是这一性能的主要的和非常有价值的应用。

第17章　异常处理。在程序设计中，出错处理总是很成问题。即便如实地返回了出错信息或设置了标记，函数调用者仍然容易忽视它。在C++中，异常处理是主要性能，当出现严重错误时，它靠将一个对象从函数中“抛出”以解决这个问题。对于不同的错误，抛出不同类型的对象。函数调用者在不同的出错处理例程中“捕捉”这些对象，一旦抛出一个异常，就不能忽略。所以能保证对错误做出必要的反应。

第18章　运行时类型识别。当我们仅给出一个指向基本类型的指针或引用时，运行时类型识别（RTTI）可以找出对象的准确类型。通常，我们会有意忽略对象的准确数据类型，而让虚函数机制来实现这个类的正确行为。而有时候，知道只由基类指针指向的对象的准确类型是有用的，这个信息常常允许更有效地完成特殊情况的运算。这一章解释RTTI做什么和如何使用它。

附录A　其他性能。到写这本书时，C++标准还没有最终形成。虽然最终出现在这个语言中的所有性能都会被加到这个标准中，但仍有一些还没有出现在所有的编译器中。这个附录简要地说明在实际编译器中（或在编译器的未来版本中）会出现的其他性能。

附录B　编程准则。这个附录是对C++程序设计的一系列建议。这些建议主要是从我的讲课内容和程序设计练习中收集来的，还有的来自其他老师的见解，许多内容是这本书的总结。

附录C　模拟虚构造函数。构造函数不能有任何虚性质，这是因为有时会因此产生笨拙的代码。这个附录论述了关于“虚构造函数”的两种方法。

5. 练习

我已经发现，在研讨班学习期间，简单的练习对学生加强理解是特别有用的，因此，从第2章起，每章后面都有一组练习。

这些练习难度适中，可以在一堂课内完成，老师可以通过观察以确信所有的学生正在领会这些内容。有些练习能激发优秀学生的学习兴趣。它们被设计成能在短期内求解的规模，因为这主要为了测试和复习学生的知识，而不是提出大的挑战。

6. 源代码

这本书的源代码是免费拷贝件，作为单个包分发的。读者可以在许多公告板系统（例如CompuServe和AOL）、Internet结点（例如SimTel 档案室；oak.oakland.edu是一个结点，参看SimTel/msdos/cplusplus）中找到它，并且在有些产品厂商的包中也包含它。这意味着读者可以免费共享这个包，可以发送它到公告板和Internet上，也可以把它放在共享件包中。但是，不能分块分发这些代码（即必须分发整个包）。代码的版权防止未经允许在印刷媒体上重新发表这些代码。读者可以找到以ECKELT为文件名开头的文件，在文件名中的另外两个字符用来表示版本号，例如，ECKELT01.ZIP表示PKZIP格式版本1（其他压缩工具对其他系统适用）。在每个源代码文件中会发现下面的版权通告：

```
//////////////////////////////////////////////////
// Copyright (c) Bruce Eckel, 1995
// Source code file from the book "Thinking in C++",
// Prentice Hall, 1995, ISBN: 0-13-917709-4
```

```
// All rights reserved EXCEPT as allowed by the following
// statements: You may freely use this file for your own
// work, including modifications and distribution in
// executable form only. You may copy and distribute this
// file, as long as it is only distributed in the complete
// (compressed) package with the other files from this
// book and you do not remove this copyright and notice.
// You may not distribute modified versions of the source
// code in this package. This package may be freely placed
// on bulletin boards, internet nodes, shareware disks and
// product vendor disks. You may not use this file in
// printed media without the express permission of the
// author. Bruce Eckel makes no
// representation about the suitability of this software
// for any purpose. It is provided "as is" without express
// or implied warranty of any kind. The entire risk as to
// the quality and performance of the software is with
// you. Should the software prove defective, you assume
// the cost of all necessary servicing, repair, or
// correction.
// If you think you've found an error, please
// email all modified files with loudly commented changes
// to: eckel@aol.com (please use the same
// address for non-code errors found in the book).
//////////////////////////////////////////////////
```

读者可以在他的项目和课堂上使用这些代码，只要保留出现在每个源文件中的版权公告就行。

• 编码标准

在这本书的例子中，我使用特殊的编码风格，这种风格已沿用了多年，并受到了Bjarne Stroustrup的《C++ Programming Language》[1]的风格启发。关于风格形式的话题可以争论几小时，所以我并不试图通过我的例子规定正确的风格。我有使用我规定风格的动机。因为C++是自由形式的程序设计语言，所以我们可以使用任何已经习惯了的风格。

这本书中的程序是由字处理器自动地包含在正文中的文件，是直接可编译的。（在每个文件的第一行中我用特殊的格式以便于这个包含，这一行以//:和跟随文件名开始。）这样，在书中印出的代码文件应当没有编译错误。那些会在编译时引起错误信息的代码用注释//! 注出，所以很容易被发现和用自动的方法被测试。作者已发现和得到报告的出现在被分发的代码中的错误已在这本书中订正了。

这本书的指标之一是所有程序无编译和连接错误（虽然有时引起警告）。一些程序只是示范编码例子，不表示独立的程序，它们有空main()函数，如同

```
main(){}
```

这就使得连接器能无错误地完成连接。

[1] 出处同上。

在标准情况下main()返回int，但是标准C++规定，如果在main()内没有return语句，编译器将自动产生对应于return 0的代码。在本书中遵从这一规定（尽管一些编译器对此仍可能产生警告）。

7. 语言标准

当认为与ANSI/ISO C标准一致时，全书将用术语：标准C。

虽然到写这本书时ANSI/ISO C++委员会仍在为这个语言而工作，但这里假设：被这个委员会接受的性能都要设法实现，所有“主要的”性能都被增加进来。这样，当提到写这本书时的在ANSI/ISO草案中的性能时，我使用术语：标准C++。记住，无论如何这个信息是基于我写这本书时的草案，而且，在这本书中描述的性能在标准完成之前有一些可能已经改变了。

8. 语言支持

读者的编译器可能不支持在这本书中讨论的所有性能，特别是当还没有编译器的最新版本时。实现像C++这样的语言是艰巨的任务，而且读者可能希望将这些性能划分成一些部分后分别出现，而不是一下子全出现。如果读者试用在这本书中的某个例子，从编译器中得到了大量的错误，这可能不是代码或编译错误，而可能是他的特定编译器还没有实现例子中的某个性能。

9. 错误

无论作者有多少发现错误的技巧，总有一些错误漏网，它们常常能被新读者发现。如果读者发现了任何认为是错误的地方，请在最初的源文件（读者能在Internet上和其他来源中找到它们，如本书前言“源代码”部分所列）中明显地注释出错误，提出纠正建议，然后通过电子邮件发往eckel@aol.com，以便在本书下一次印刷时更正。欢迎为下一个修订本的补充练习或要求提出建议。感谢读者的帮助。

10. 感谢

这本书中的思想和观点有很多来源于：我的朋友，例如Dan Saks、Scott Meyers、Charles Petzold和Michael Wilk；该语言的倡导者，例如Bjarne Stroustrup、Andrew Koening和Rob Murray；C++标准委员会的成员，例如Tom Plum、Reg Charney、Tom Penello、Chuck Allison、Sam Druker、Nathan Myers和Uwe Stienmueller；在软件开发会议上对我的C++教学发过言的人们；经常参加我训练班的学生，我认真地听取他们问的问题，以使得这本教材更清晰。

我在为Miller Freeman公司出差旅途中向我的朋友Richard Hale Shaw介绍了这本教材。Richard（还有Kim）的见解和支持对我很有帮助。我还要感谢KoAnn Vikoren、Lisa Monson、Julie Shaw、Nicole Freeman、Cindy Blair、Barbara Hanscome、Yvonne Labat、Regina Ridley、Alex Dunne、Judy DeMocker和该公司的其他人员。

这本教材首先发表在由Tyler Sperry编辑的Embedded Systems Programming杂志上。

目　录

第1章　对象的演化

计算机革命起源于一台机器，程序设计语言也源于一台机器。

然而计算机并不仅仅是一台机器，它是心智放大器和另一种有表述能力的媒体。这一点使它不很像机器，而更像我们大脑的一部分，更像其他有表述能力的手段，例如写作、绘画、雕刻、动画制作或电影制作。面向对象的程序设计是计算机向有表述能力的媒体发展中的一部分。

本章将介绍面向对象程序设计（OOP）的基本概念，然后讨论OOP开发方法，最后介绍使程序员、项目和公司使用面向对象程序设计方法而采用的策略。

本章是一些背景材料，如果读者急于学习这门语言的具体内容，可以跳到第2章，然后再回过头来学习本章。

1.1　基本概念

C++包含了比面向对象程序设计基本概念更多的内容，读者应当在学习设计和开发程序之前先理解该语言所包含的基本概念。

1.1.1　对象：特性+行为[1]

第一个面向对象的程序设计语言是60年代开发的Simula-67。其目的是为了解决模拟问题。典型的模拟问题是银行出纳业务，包括出纳部门、顾客、业务、货币的单位等大量的“对象”。把那些在程序执行期间除了状态之外其他方面都一样的对象归在一起，构成对象的“类”，这就是“类”一词的来源。

类描述了一组有相同特性（数据元素）和相同行为（函数）的对象。类实际上就是数据类型，例如，浮点数也有一组特性和行为。区别在于程序员定义类是为了与具体问题相适应，而不是被迫使用已存在的数据类型。这些已存在的数据类型的设计动机仅仅是为了描述机器的存储单元。程序员可以通过增添他所需要的新数据类型来扩展这个程序设计语言。该程序设计系统欢迎创建、关注新的类，对它们进行与内部类型一样的类型检查。

这种方法并不限于去模拟具体问题。尽管不是所有的人都同意，但大部分人相信，任何程序都模拟所设计系统。OOP技术能很容易地将大量问题归纳成为一个简单的解，这一发现产生了大量的OOP语言，其中最著名的是Smalltalk——C++ 之前最成功的OOP语言。

抽象数据类型的创建是面向对象程序设计中的一个基本概念。抽象数据类型几乎能像内部类型一样准确工作。程序员可以创建类型的变量（在面向对象程序设计中称为“对象”或“实例”）并操纵这些变量（称为发送“消息”或“请求”，对象根据发来的消息知道需要做什么事情）。

1.1.2　继承：类型关系

类型不仅仅说明一组对象上的约束，还说明与其他类型之间的关系。两个类型可以有共同的特性和行为，但是，一个类型可能包括比另一个类型更多的特性，也可以处理更多的消息

[1] 这一描述部分引自我对《The Tao of Objects》（Gary Entsminger著）一书的介绍。