Java 编程思想 第4版

作者：Bruce Eckel

写在前面的话

我的兄弟Todd目前正在进行从硬件到编程领域的工作转变。我曾提醒他下一次大革命的重点将是遗传工程。

我们的微生物技术将能制造食品、燃油和塑料；它们都是清洁的，不会造成污染，而且能使人类进一步透视物理世界的奥秘。我认为相比之下电脑的进步会显得微不足道。

但随后，我又意识到自己正在犯一些科幻作家常犯的错误：在技术中迷失了（这种事情在科幻小说里常有发生）！如果是一名有经验的作家，就知道绝对不能就事论事，必须以人为中心。遗传对我们的生命有非常大的影响，但不能十分确定它能抹淡计算机革命——或至少信息革命——的影响。信息涉及人相互间的沟通：的确，汽车和轮子的发明都非常重要，但它们最终亦如此而已。真正重要的还是我们与世界的关系，而其中最关键的就是通信。

这本书或许能说明一些问题。许多人认为我有点儿大胆或者稍微有些狂妄，居然把所有家当都摆到了Web上。“这样做还有谁来买它呢？”他们问。假如我是一个十分守旧的人，那么绝对不这样干。但我确实不想再沿原来的老路再写一本计算机参考书了。我不知道最终会发生什么事情，但的确认为这是我对一本书作出的最明智的一个决定。

至少有一件事是可以肯定的，人们开始向我发送纠错反馈。这是一个令人震惊的体验，因为读者会看到书中的每一个角落，并揪出那些藏匿得很深的技术及语法错误。这样一来，和其他以传统方式发行的书不同，我就能及时改正已知的所有类别的错误，而不是让它们最终印成铅字，堂而皇之地出现在各位的面前。俗话说，“当局者迷，旁观者清”。人们对书中的错误是非常敏感的，往往毫不客气地指出：“我想这样说是错误的，我的看法是……”。在我仔细研究后，往往发现自己确实有不当之处，而这是当初写作时根本没有意识到的（检查多少遍也不行）。我意识到这是群体力量的一个可喜的反映，它使这本书显得的确与众不同。

但我随之又听到了另一个声音：“好吧，你在那儿放的电子版的确很有创意，但我想要的是从真正的出版社那里印刷的一个版本！”事实上，我作出了许多努力，让它用普通打印机机就能得到很好的阅读效果，但仍然不象真正印刷的书那样正规。许多人不想在屏幕上看完整本书，也不喜欢拿着一叠纸阅读。无论打印格式有多么好，这些人喜欢是仍然是真正的“书”（激光打印机的墨盒也太贵了一点）。现在看来，计算机的革命仍未使出版界完全走出传统的模式。但是，有一个学生向我推荐了未来出版的一种模式：书籍将首先在互联网上出版，然后只有在绝对必要的前提下，才会印刷到纸张上。目前，为数众多的书籍销售都不十分理想，许多出版社都在亏本。但如采用这种方式出版，就显得灵活得多，也更容易保证赢利。

这本书也从另一个角度也给了我深刻的启迪。我刚开始的时候以为Java“只是另一种程序设计语言”。这个想法在许多情况下都是成立的。但随着时间的推移，我对它的学习也愈加深入，开始意识到它的基本宗旨与我见过的其他所有语言都有所区别。

程序设计与对复杂性的操控有很大的关系：对一个准备解决的问题，它的复杂程度取决用于解决它的机器的复杂程度。正是由于这一复杂性的存在，我们的程序设计项目屡屡失败。对于我以前接触过的所有编程语言，它们都没能跳过这一框框，由此决定了它们的主要设计目标就是克服程序开发与维护中的复杂性。当然，许多语言在设计时就已考虑到了复杂性的问题。但从另一角度看，实际设计时肯定会有另一些问题浮现出来，需把它们考虑到这个复杂性的问题里。不可避免地，其他那些问题最后会变成最让程序员头痛的。例如，C++必须同C保持向后兼容（使C程序员能尽快地适应新环境），同时又要保证编程的效率。C++在这两个方面都设计得很好，为其赢得了不少的声誉。但它们同时也暴露出了额外的复杂性，阻碍了某些项目的成功实现（当然，你可以责备程序员和管理层，但假如一种语言能通过捕获你的错误而提供帮助，它为什么不那样做呢？）。作为另一个例子，Visual Basic（VB）同当初的BASIC有关的紧密的联系。而BASIC并没有打算设计成一种能全面解决问题的语言，所以堆加到VB身上的所有扩展都造成了令人头痛和难于管理和维护的语法。另一方面，C++、VB和其他如Smalltalk之类的语言均在复杂性的问题上下了一番功夫。由此得到的结果便是，它们在解决特定类型的问题时是非常成功的。

在理解到Java最终的目标是减轻程序员的负担时，我才真正感受到了震憾，尽管它的潜台词好象是说：“除了缩短时间和减小产生健壮代码的难度以外，我们不关心其他任何事情。”在目前这个初级阶段，达到那个目标的后果便是代码不能特别快地运行（尽管有许多保证都说Java终究有一天会运行得多么快），但它确实将开发时间缩短到令人惊讶的地步——几乎只有创建一个等效C++程序一半甚至更短的时间。这段节省下来的时间可以产生更大的效益，但Java并不仅止于此。它甚至更上一层楼，将重要性越来越明显的一切复杂任务都封装在内，比如网络程序和多线程处理等等。Java的各种语言特性和库在任何时候都能使那些任务轻而易举完成。而且最后，它解决了一些真正有些难度的复杂问题：跨平台程序、动态代码改换以及安全保护等等。换在从前，其中任何每一个都能使你头大如斗。所以不管我们见到了什么性能问题，Java的保证仍然是非常有效的：它使程序员显著提高了程序设计的效率！

在我看来，编程效率提升后影响最大的就是Web。网络程序设计以前非常困难，而Java使这个问题迎刃而解（而且Java也在不断地进步，使解决这类问题变得越来越容易）。网络程序的设计要求我们相互间更有效率地沟通，而且至少要比电话通信来得便宜（仅仅电子函件就为许多公司带来了好处）。随着我们网上通信越来越频繁，令人震惊的事情会慢慢发生，而且它们令人吃惊的程度绝不亚于当初工业革命给人带来的震憾。

在各个方面：创建程序；按计划编制程序；构造用户界面，使程序能与用户沟通；在不同类型的机器上运行程序；以及方便地编写程序，使其能通过因特网通信——Java提高了人与人之间的“通信带宽”。而且我认为通信革命的结果可能并不单单是数量庞大的比特到处传来传去那么简单。我们认为认清真正的革命发生在哪里，因为人和人之间的交流变得更方便了——个体与个体之间，个体与组之间，组与组之间，甚至在星球之间。有人预言下一次大革命的发生就是由于足够多的人和足够多的相互连接造成的，而这种革命是以整个世界为基础发生的。Java可能是、也可能不是促成那次革命的直接因素，但我在这里至少感觉自己在做一些有意义的工作——尝试教会大家一种重要的语言！

引言

同人类任何语言一样，Java为我们提供了一种表达思想的方式。如操作得当，同其他方式相比，随着问题变得愈大和愈复杂，这种表达方式的方便性和灵活性会显露无遗。

不可将Java简单想象成一系列特性的集合；如孤立地看，有些特性是没有任何意义的。只有在考虑“设计”、而非考虑简单的编码时，才可真正体会到Java的强大。为了按这种方式理解Java，首先必须掌握它与编程的一些基本概念。本书讨论了编程问题、它们为何会成为问题以及Java用以解决它们的方法。所以，我对每一章的解释都建立在如何用语言解决一种特定类型的问题基础上。按这种方式，我希望引导您一步一步地进入Java的世界，使其最终成为您最自然的一种语言。

贯穿本书，我试图在您的大脑里建立一个模型——或者说一个“知识结构”。这样可加深对语言的理解。若遇到难解之处，应学会把它填入这个模型的对应地方，然后自行演绎出答案。事实上，学习任何语言时，脑海里有一个现成的知识结构往往会起到事半功倍的效果。

1. 前提

本书假定读者对编程多少有些熟悉。应已知道程序是一系列语句的集合，知道子程序／函数／宏是什么，知道象“If”这样的控制语句，也知道象“while”这样的循环结构。注意这些东西在大量语言里都是类似的。假如您学过一种宏语言，或者用过Perl之类的工具，那么它们的基本概念并无什么区别。总之，只要能习惯基本的编程概念，就可顺利阅读本书。当然，C/C++程序员在阅读时能占到更多的便宜。但即使不熟悉C，一样不要把自己排除在外（尽管以后的学习要付出更大的努力）。我会讲述面向对象编程的概念，以及Java的基本控制机制，所以不用担心自己会打不好基础。况且，您需要学习的第一类知识就会涉及到基本的流程控制语句。

尽管经常都会谈及C和C++语言的一些特性，但并没有打算使它们成为内部参考，而是想帮助所有程序员都能正确地看待那两种语言。毕竟，Java是从它们那里衍生出来的。我将试着尽可能地简化这些引用和参考，并合理地解释一名非C/C++程序员通常不太熟悉的内容。

2. Java的学习

在我第一本书《Using C++》面市的几乎同一时间（Osborne/McGraw-Hill于1989年出版），我开始教授那种语言。程序设计语言的教授已成为我的专业。自1989年以来，我便在世界各地见过许多昏昏欲睡、满脸茫然以及困惑不解的面容。开始在室内面向较少的一组人授课以后，我从作业中发现了一些特别的问题。即使那些上课面带会心的微笑或者频频点头的学生，对许多问题也存在认识上的混淆。在过去几年间的“软件开发会议”上，由我主持C++分组讨论会（现在变成了Java讨论会）。有的演讲人试图在很短的时间内向听众灌输过多的主题。所以到最后，尽管听众的水平都还可以，而且提供的材料也很充足，但仍然损失了一部分听众。这可能是由于问得太多了，但由于我是那些采取传统授课方式的人之一，所以很想使每个人都能跟上讲课进度。

有段时间，我编制了大量教学简报。经过不断的试验和修订（或称“反复”，这是在Java程序设计中非常有用的一项技术），最后成功地在一门课程中集成了从我的教学经验中总结出来的所有东西——我在很长一段时间里都在使用。其中由一系列离散的、易于消化的小步骤组成，而且每个小课程结束后都有一些适当的练习。我目前已在Java公开研讨会上公布了这一课程，大家可到http://www.BruceEckel.com了解详情（对研讨会的介绍也以CD-ROM的形式提供，具体信息可在同样的Web站点找到）。

从每一次研讨会收到的反馈都帮助我修改及重新制订学习材料的重心，直到我最后认为它成为一个完善的教学载体为止。但本书并非仅仅是一本教科书——我尝试在其中装入尽可能多的信息，并按照主题进行了有序的分类。无论如何，这本书的主要宗旨是为那些独立学习的人士服务，他们正准备深入一门新的程序设计语言，而没有太大的可能参加此类专业研讨会。

3. 目标

就象我的前一本书《Thinking in C++》一样，这本书面向语言的教授进行了良好的结构与组织。特别地，我的目标是建立一套有序的机制，可帮助我在自己的研讨会上更好地进行语言教学。在我思考书中的一章时，实际上是在想如何教好一堂课。我的目标是得到一系列规模适中的教学模块，可以在合理的时间内教完。随后是一些精心挑选的练习，可以在课堂上当即完成。

在这本书中，我想达到的目标总结如下：

(1) 每一次都将教学内容向前推进一小步，便于读者在继续后面的学习前消化前面的内容。

(2) 采用的示例尽可能简短。当然，这样做有时会妨碍我解决“现实世界”的问题。但我同时也发现对那些新手来说，如果他们能理解每一个细节，那么一般会产生更大的学习兴趣。而假如他们一开始就被要解决的问题的深度和广度所震惊，那么一般都不会收到很好的学习效果。另外在实际教学过程中，对能够摘录的代码数量是有严重限制的。另一方面，这样做无疑会有些人会批评我采用了“不真实的例子”，但只要能起到良好的效果，我宁愿接受这一指责。

(3) 要揭示的特性按照我精心挑选的顺序依次出场，而且尽可能符合读者的思想历程。当然，我不可能永远都做到这一点；在那些情况下，会给出一段简要的声明，指出这个问题。

(4) 只把我认为有助于理解语言的东西介绍给读者，而不是把我知道的一切东西都抖出来，这并非藏私。我认为信息的重要程度是存在一个合理的层次的。有些情况是95%的程序员都永远不必了解的。如强行学习，只会干扰他们的正常思维，从而加深语言在他们面前表现出来的难度。以C语言为例，假如你能记住运算符优先次序表（我从来记不住），那么就可以写出更“聪明”的代码。但再深入想一层，那也会使代码的读者／维护者感到困扰。所以忘了那些次序吧，在拿不准的时候加上括号即可。

(5) 每一节都有明确的学习重点，所以教学时间（以及练习的间隔时间）非常短。这样做不仅能保持读者思想的活跃，也能使问题更容易理解，对自己的学习产生更大的信心。

(6) 提供一个坚实的基础，使读者能充分理解问题，以便更容易转向一些更加困难的课程和书籍。

4. 联机文档

由Sun微系统公司提供的Java语言和库（可免费下载）配套提供了电子版的用户帮助手册，可用Web浏览器阅读。此外，由其他厂商开发的几乎所有类似产品都有一套等价的文档系统。而目前出版的与Java有关的几乎所有书籍都重复了这份文档。所以你要么已经拥有了它，要么需要下载。所以除非特别必要，否则本书不会重复那份文档的内容。因为一般地说，用Web浏览器查找与类有关的资料比在书中查找方便得多（电子版的东西更新也快）。只有在需要对文档进行补充，以便你能理解一个特定的例子时，本书才会提供有关类的一些附加说明。

5. 章节

本书在设计时认真考虑了人们学习Java语言的方式。在我授课时，学生们的反映有效地帮助了我认识哪些部分是比较困难的，需特别加以留意。我也曾经一次讲述了太多的问题，但得到的教训是：假如包括了大量新特性，就需要对它们全部作出解释，而这特别容易加深学生们的混淆。因此，我进行了大量努力，使这本书一次尽可能地少涉及一些问题。

所以，我在书中的目标是让每一章都讲述一种语言特性，或者只讲述少数几个相互关联的特性。这样一来，读者在转向下一主题时，就能更容易地消化前面学到的知识。

下面列出对本书各章的一个简要说明，它们与我实际进行的课堂教学是对应的。

(1) 第1章：对象入门

这一章是对面向对象的程序设计（OOP）的一个综述，其中包括对“什么是对象”之类的基本问题的回答，并讲述了接口与实现、抽象与封装、消息与函数、继承与合成以及非常重要的多形性的概念。这一章会向大家提出一些对象创建的基本问题，比如构建器、对象存在于何处、创建好后把它们置于什么地方以及魔术般的垃圾收集器（能够清除不再需要的对象）。要介绍的另一些问题还包括通过违例实现的错误控制机制、反应灵敏的用户界面的多线程处理以及连网和因特网等等。大家也会从中了解到是什么使得Java如此特别，它为什么取得了这么大的成功，以及与面向对象的分析与设计有关的问题。

(2) 第2章：一切都是对象

本章将大家带到可以着手写自己的第一个Java程序的地方，所以必须对一些基本概念作出解释，其中包括对象“句柄”的概念；怎样创建一个对象；对基本数据类型和数组的一个介绍；作用域以及垃圾收集器清除对象的方式；如何将Java中的所有东西都归为一种新数据类型（类），以及如何创建自己的类；函数、自变量以及返回值；名字的可见度以及使用来自其他库的组件；static关键字；注释和嵌入文档等等。

(3) 第3章：控制程序流程

本章开始介绍起源于C和C++，由Java继承的所有运算符。除此以外，还要学习运算符一些不易使人注意的问题，以及涉及造型、升迁以及优先次序的问题。随后要讲述的是基本的流程控制以及选择运算，这些是几乎所有程序设计语言都具有的特性：用if-else实现选择；用for和while实现循环；用break和continue以及Java的标签式break和contiune（它们被认为是Java中“不见的gogo”）退出循环；以及用switch实现另一种形式的选择。尽管这些与C和C++中见到的有一定的共通性，但多少存在一些区别。除此以外，所有示例都是完整的Java示例，能使大家很快地熟悉Java的外观。

(4) 第4章：初始化和清除

本章开始介绍构建器，它的作用是担保初始化的正确实现。对构建器的定义要涉及函数过载的概念（因为可能同时有几个构建器）。随后要讨论的是清除过程，它并非肯定如想象的那么简单。用完一个对象后，通常可以不必管它，垃圾收集器会自动介入，释放由它占据的内存。这里详细探讨了垃圾收集器以及它的一些特点。在这一章的最后，我们将更贴近地观察初始化过程：自动成员初始化、指定成员初始化、初始化的顺序、static（静态）初始化以及数组初始化等等。

(5) 第5章：隐藏实现过程

本章要探讨将代码封装到一起的方式，以及在库的其他部分隐藏时，为什么仍有一部分处于暴露状态。首先要讨论的是package和import关键字，它们的作用是进行文件级的封装（打包）操作，并允许我们构建由类构成的库（类库）。此时也会谈到目录路径和文件名的问题。本章剩下的部分将讨论public，private以及protected三个关键字、“友好”访问的概念以及各种场合下不同访问控制级的意义。

(6) 第6章：类再生

继承的概念是几乎所有OOP语言中都占有重要的地位。它是对现有类加以利用，并为其添加新功能的一种有效途径（同时可以修改它，这是第7章的主题）。通过继承来重复使用原有的代码时（再生），一般需要保持“基础类”不变，只是将这儿或那儿的东西串联起来，以达到预期的效果。然而，继承并不是在现有类基础上制造新类的唯一手段。通过“合成”，亦可将一个对象嵌入新类。在这一章中，大家将学习在Java中重复使用代码的这两种方法，以及具体如何运用。

(7) 第7章：多形性

若由你自己来干，可能要花9个月的时间才能发现和理解多形性的问题，这一特性实际是OOP一个重要的基础。通过一些小的、简单的例子，读者可知道如何通过继承来创建一系列类型，并通过它们共有的基础类对那个系列中的对象进行操作。通过Java的多形性概念，同一系列中的所有对象都具有了共通性。这意味着我们编写的代码不必再依赖特定的类型信息。这使程序更易扩展，包容力也更强。由此，程序的构建和代码的维护可以变得更方便，付出的代价也会更低。此外，Java还通过“接口”提供了设置再生关系的第三种途径。这儿所谓的“接口”是对对象物理“接口”一种纯粹的抽象。一旦理解了多形性的概念，接口的含义就很容易解释了。本章也向大家介绍了Java 1.1的“内部类”。

(8) 第8章：对象的容纳

对一个非常简单的程序来说，它可能只拥有一个固定数量的对象，而且对象的“生存时间”或者“存在时间”是已知的。但是通常，我们的程序会在不定的时间创建新对象，只有在程序运行时才可了解到它们的详情。此外，除非进入运行期，否则无法知道所需对象的数量，甚至无法得知它们确切的类型。为解决这个常见的程序设计问题，我们需要拥有一种能力，可在任何时间、任何地点创建任何数量的对象。本章的宗旨便是探讨在使用对象的同时用来容纳它们的一些Java工具：从简单的数组到复杂的集合（数据结构），如Vector和Hashtable等。最后，我们还会深入讨论新型和改进过的Java 1.2集合库。

(9) 第9章：违例差错控制

Java最基本的设计宗旨之一便是组织错误的代码不会真的运行起来。编译器会尽可能捕获问题。但某些情况下，除非进入运行期，否则问题是不会被发现的。这些问题要么属于编程错误，要么则是一些自然的出错状况，它们只有在作为程序正常运行的一部分时才会成立。Java为此提供了“违例控制”机制，用于控制程序运行时产生的一切问题。这一章将解释try、catch、throw、throws以及finally等关键字在Java中的工作原理。并讲述什么时候应当“掷”出违例，以及在捕获到违例后该采取什么操作。此外，大家还会学习Java的一些标准违例，如何构建自己的违例，违例发生在构建器中怎么办，以及违例控制器如何定位等等。

(10) 第10章：Java IO系统

理论上，我们可将任何程序分割为三部分：输入、处理和输出。这意味着IO（输入／输出）是所有程序最为关键的部分。在这一章中，大家将学习Java为此提供的各种类，如何用它们读写文件、内存块以及控制台等。“老”IO和Java 1.1的“新”IO将得到着重强调。除此之外，本节还要探讨如何获取一个对象、对其进行“流式”加工（使其能置入磁盘或通过网络传送）以及重新构建它等等。这些操作在Java的1.1版中都可以自动完成。另外，我们也要讨论Java 1.1的压缩库，它将用在Java的归档文件格式中（JAR）。

(11) 第11章：运行期类型鉴定

若只有指向基础类的一个句柄，Java的运行期类型标鉴定（RTTI）使我们能获知一个对象的准确类型是什么。一般情况下，我们需要有意忽略一个对象的准确类型，让Java的动态绑定机制（多形性）为那一类型实现正确的行为。但在某些场合下，对于只有一个基础句柄的对象，我们仍然特别有必要了解它的准确类型是什么。拥有这个资料后，通常可以更有效地执行一次特殊情况下的操作。本章将解释RTTI的用途、如何使用以及在适当的时候如何放弃它。此外，Java 1.1的“反射”特性也会在这里得到介绍。

(12) 第12章：传递和返回对象

由于我们在Java中同对象沟通的唯一途径是“句柄”，所以将对象传递到一个函数里以及从那个函数返回一个对象的概念就显得非常有趣了。本章将解释在函数中进出时，什么才是为了管理对象需要了解的。同时也会讲述String（字串）类的概念，它用一种不同的方式解决了同样的问题。

(13) 第13章：创建窗口和程序片

Java配套提供了“抽象Windows工具包”（AWT）。这实际是一系列类的集合，能以一种可移植的形式解决视窗操纵问题。这些窗口化程序既可以程序片的形式出现，亦可作为独立的应用程序使用。本章将向大家介绍AWT以及网上程序片的创建过程。我们也会探讨AWT的优缺点以及Java 1.1在GUI方面的一些改进。同时，重要的“Java Beans”技术也会在这里得到强调。Java Beans是创建“快速应用开发”（RAD）程序构造工具的重要基础。我们最后介绍的是Java 1.2的“Swing”库——它使Java的UI组件得到了显著的改善。

(14) 第14章：多线程

Java提供了一套内建的机制，可提供对多个并发子任务的支持，我们称其为“线程”。这线程均在单一的程序内运行。除非机器安装了多个处理器，否则这就是多个子任务的唯一运行方式。尽管还有别的许多重要用途，但在打算创建一个反应灵敏的用户界面时，多线程的运用显得尤为重要。举个例子来说，在采用了多线程技术后，尽管当时还有别的任务在执行，但用户仍然可以毫无阻碍地按下一个按钮，或者键入一些文字。本章将对Java的多线程处理机制进行探讨，并介绍相关的语法。

(15) 第15章 网络编程

开始编写网络应用时，就会发现所有Java特性和库仿佛早已串联到了一起。本章将探讨如何通过因特网通信，以及Java用以辅助此类编程的一些类。此外，这里也展示了如何创建一个Java程序片，令其同一个“通用网关接口”（CGI）程序通信；揭示了如何用C++编写CGI程序；也讲述了与Java 1.1的“Java数据库连接”（JDBC）和“远程方法调

用”（RMI）有关的问题。

(16) 第16章 设计范式

本章将讨论非常重要、但同时也是非传统的“范式”程序设计概念。大家会学习设计进展过程的一个例子。首先是最初的方案，然后经历各种程序逻辑，将方案不断改革为更恰当的设计。通过整个过程的学习，大家可体会到使设计思想逐渐变得清晰起来的一种途径。

(17) 第17章 项目

本章包括了一系列项目，它们要么以本书前面讲述的内容为基础，要么对以前各章进行了一番扩展。这些项目显然是书中最复杂的，它们有效演示了新技术和类库的应用。

有些主题似乎不太适合放到本书的核心位置，但我发现有必要在教学时讨论它们，这些主题都放入了本书的附录。

(18) 附录A：使用非Java代码

对一个完全能够移植的Java程序，它肯定存在一些严重的缺陷：速度太慢，而且不能访问与具体平台有关的服务。若事先知道程序要在什么平台上使用，就可考虑将一些操作变成“固有方法”，从而显著加快执行速度。这些“固有方法”实际是一些特殊的函数，以另一种程序设计语言写成（目前仅支持C/C++）。Java还可通过另一些途径提供对非Java代码的支持，其中包括CORBA。本附录将详细介绍这些特性，以便大家能创建一些简单的例子，同非Java代码打交道。

(19) 附录B：对比C++和Java

对一个C++程序员，他应该已经掌握了面向对象程序设计的基本概念，而且Java语法对他来说无疑是非常眼熟的。这一点是明显的，因为Java本身就是从C++衍生而来。但是，C++和Java之间的确存在一些显著的差异。这些差异意味着Java在C++基础上作出的重大改进。一旦理解了这些差异，就能理解为什么说Java是一种杰出的语言。这一附录便是为这个目的设立的，它讲述了使Java与C++明显有别的一些重要特性。

(20) 附录C：Java编程规则

本附录提供了大量建议，帮助大家进行低级程序设计和代码编写。

(21) 附录D：性能

通过这个附录的学习，大家可发现自己Java程序中存在的瓶颈，并可有效地改善执行速度。

(22) 附录E：关于垃圾收集的一些话

这个附录讲述了用于实现垃圾收集的操作和方法。

(23) 附录F：推荐读物

列出我感觉特别有用的一系列Java参考书。

6. 练习

为巩固对新知识的掌握，我发现简单的练习特别有用。所以读者在每一章结束时都能找到一系列练习。

大多数练习都很简单，在合理的时间内可以完成。如将本书作为教材，可考虑在课堂内完成。老师要注意观察，确定所有学生都已消化了讲授的内容。有些练习要难些，他们是为那些有兴趣深入的读者准备的。大多数练习都可在较短时间内做完，有效地检测和加深您的知识。有些题目比较具有挑战性，但都不会太麻烦。事实上，练习中碰到的问题在实际应用中也会经常碰到。

7. 多媒体CD-ROM

本书配套提供了一片多媒体CD-ROM，可单独购买及使用。它与其他计算机书籍的普通配套CD不同，那些CD通常仅包含了书中用到的源码（本书的源码可从www.BruceEckel.com免费下载）。本CD-ROM是一个独立的产品，包含了一周“Hads-OnJava”培训课程的全部内容。这是一个由Bruce Eckel讲授的、长度在15小时以上的课程，含500张以上的演示幻灯片。该课程建立在这本书的基础上，所以是非常理想的一个配套产品。

CD-ROM包含了本书的两个版本：

(1) 本书一个可打印的版本，与下载版完全一致。

(2) 为方便读者在屏幕上阅读和索引，CD-ROM提供了一个独特的超链接版本。这些超链接包括：

■230个章、节和小标题链接

■3600个索引链接

CD-ROM刻录了600MB以上的数据。我相信它已对所谓“物超所值”进行了崭新的定义。

CD-ROM包含了本书打印版的所有东西，另外还有来自五天快速入门课程的全部材料。我相信它建立了一个新的书刊品质评定标准。

若想单独购买此CD-ROM，只能从Web站点www.BruceEckel.com处直接订购。

8. 源代码

本书所有源码都作为保留版权的免费软件提供，可以独立软件包的形式获得，亦可从http://www.BruceEckel.com下载。为保证大家获得的是最新版本，我用这个正式站点发行代码以及本书电子版。亦可在其他站点找到电子书和源码的镜像版（有些站点已在http://www.BruceEckel.com处列出）。但无论如何，都应检查正式站点，确定镜像版确实是最新的版本。可在课堂和其他教育场所发布这些代码。

版权的主要目标是保证源码得到正确的引用，并防止在未经许可的情况下，在印刷材料中发布代码。通常，只要源码获得了正确的引用，则在大多数媒体中使用本书的示例都没有什么问题。

在每个源码文件中，都能发现下述版本声明文字：

可在自己的开发项目中使用代码，并可在课堂上引用（包括学习材料）。但要确定版权声明在每个源文件中得到了保留。

9. 编码样式

在本书正文中，标识符（函数、变量和类名）以粗体印刷。大多数关键字也采用粗体——除了一些频繁用到的关键字（若全部采用粗体，会使页面拥挤难看，比如那些“类”）。

对于本书的示例，我采用了一种特定的编码样式。该样式得到了大多数Java开发环境的支持。该样式问世已有几年的时间，最早起源于Bjarne Stroustrup先生在《The C++ Programming Language》里采用的样式（Addison-Wesley 1991年出版，第2版）。由于代码样式目前是个敏感问题，极易招致数小时的激烈辩论，所以我在这儿只想指出自己并不打算通过这些示例建立一种样式标准。之所以采用这些样式，完全出于我自己的考虑。由于Java是一种形式非常自由的编程语言，所以读者完全可以根据自己的感觉选用了适合的编码样式。

本书的程序是由字处理程序包括在正文中的，它们直接取自编译好的文件。所以，本书印刷的代码文件应能正常工作，不会造成编译器错误。会造成编译错误的代码已经用注释//!标出。所以很容易发现，也很容易用自动方式进行测试。读者发现并向作者报告的错误首先会在发行的源码中改正，然后在本书的更新版中校订（所有更新都会在Web站点http://www.BruceEckel.com处出现）。

10. Java版本

尽管我用几家厂商的Java开发平台对本书的代码进行了测试，但在判断代码行为是否正确时，却通常以Sun公司的Java开发平台为准。

当您读到本书时，Sun应已发行了Java的三个重要版本：1.0，1.1及1.2（Sun声称每9个月就会发布一个主要更新版本）。就我看，1.1版对Java语言进行了显著改进，完全应标记成2.0版（由于1.1已作出了如此大的修改，真不敢想象2.0版会出现什么变化）。然而，它的1.2版看起来最终将Java推入了一个全盛时期，特别是其中考虑到了用户界面工具。

本书主要讨论了1.0和1.1版，1.2版有部分内容涉及。但在有些时候，新方法明显优于老方法。此时，我会明显偏向于新方法，通常教给大家更好的方法，而完全忽略老方法。然而，有的新方法要以老方法为基础，所以不可避免地要从老方法入手。这一特点尤以AWT为甚，因为那儿不仅存在数量众多的老式Java 1.0代码，有的平台仍然只支持Java 1.0。我会尽量指出哪些特性是哪个版本特有的。

大家会注意到我并未使用子版本号，比如1.1.1。至本书完稿为止，Sun公司发布的最后一个1.0版是1.02；而1.1的最后版本是1.1.5（Java 1.2仍在做β测试）。在这本书中，我只会提到Java 1.0，Java 1.1及Java 1.2，避免由于子版本编号过多造成的键入和印刷错误。

11. 课程和培训

我的公司提供了一个五日制的公共培训课程，以本书的内容为基础。每章的内容都代表着一堂课，并附有相应的课后练习，以便巩固学到的知识。一些辅助用的幻灯片可在本书的配套光盘上找到，最大限度地方便各位读者。欲了解更多的情况，请访问：

http://www.BruceEckel.com

或发函至：

Bruce@EckelObjects.com

我的公司也提供了咨询服务，指导客户完成整个开发过程——特别是您的单位首次接触Java开发的时候。

12. 错误

无论作者花多大精力来避免，错误总是从意想不到的地方冒出来。如果您认为自己发现了一个错误，请在源文件（可在http://www.BruceEckel.com处找到）里指出有可能是错误的地方，填好我们提供的表单。将您推荐的纠错方法通过电子函件发给Bruce@EckelObjects.com。经适当的核对与处理，Web站点的电子版以及本书的下一个印刷版本会作出相应的改正。具体格式如下：

(1) 在主题行（Subject）写上“TIJ Correction”（去掉引号），以便您的函件进入对应的目录。

(2) 在函件正文，采用下述形式：

find: 在这里写一个单行字串，以便我们搜索错误所在的地方

Comment: 在这里可写多行批注正文，最好以“here's how I think it shoud read”开头

###

其中，“###”指出批注正文的结束。这样一来，我自己设计的一个纠错工具就能对原始正文来一次“搜索”，而您建议的纠错方法会在随后的一个窗口中弹出。

若希望在本书的下一版添加什么内容，或对书中的练习题有什么意见，也欢迎您指出。我们感谢您的所有意见。

13. 封面设计

《Thinking in Java》一书封面的创作灵感来源于American Arts & CraftsMovement（美洲艺术＆手工艺品运动）。这一运动起始于世纪之交，1900到1920年达到了顶峰。它起源于英格兰，具有一定的历史背景。当时正是机器革命产生的风暴席卷整个大陆的时候，而且受到维多利亚地区强烈装饰风格的巨大影响。Arts&Crafts强调的是原始风格，回归自然的初衷是整个运动的核心。那时对手工制作推崇备至，手工艺人特别得到尊重。正因为如此，人们远远避开现代工具的使用。这场运动对整个艺术界造成了深远的影响，直至今天仍受到人们的怀念。特别是我们面临又一次世纪之交，强烈的怀旧情绪难免涌上心来。计算机发展至今，已走过了很长的一段路。我们更迫切地感到：软件设计中最重要的是设计者本身，而不是流水化的代码编制。如设计者本身的素质和修养不高，那么最多只是“生产”代码的工具而已。

我以同样的眼光来看待Java：作为一种将程序员从操作系统繁琐机制中解放出来的尝试，它的目的是使人们成为真正的“软件艺术家”。

无论作者还是本书的封面设计者（自孩提时代就是我的朋友）都从这一场运动中获得了灵感。所以接下来的事情就非常简单了，要么自己设计，要么直接采用来自那个时期的作品。

此外，封面向大家展示了一个收集箱，自然学者可能用它展示自己的昆虫标本。我们认为这些昆虫都是“对象”，全部置于更大的“收集箱”对象里，再统一置入“封面”这个对象里。它向我们揭示了面向对象编程技术最基本的“集合”概念。当然，作为一名程序员，大家对于“昆虫”或“虫”是非常敏感的（“虫”在英语里是Bug，后指程序错误）。这里的“虫”已被抓获，在一只广口瓶中杀死，最后禁闭于一个小的展览盒里——暗示Java有能力寻找、显示和消除程序里的“虫”（这是Java最具特色的特性之一）。

14. 致谢

首先，感谢Doyle Street Cohousing Community（道尔街住房社区）容忍我花两年的时间来写这本书（其实他们一直都在容忍我的“胡做非为”）。非常感谢Kevin和Sonda Donovan，是他们把科罗拉多Crested Butte市这个风景优美的地方租给我，使我整个夏天都能安心写作。感谢Crested Butte友好的居民；以及Rocky Mountain Biological Laboratory（岩石山生物实验室），他们的工作人员总是面带微笑。

这是我第一次找代理人出书，但却绝没有后悔。谢谢“摩尔文学代理公司”的Claudette Moore小姐。是她强大的信心与毅力使我最终梦想成真。

我的头两本书是与Osborne/McGraw-Hill出版社的编辑Jeff Pepper合作出版的。Jeff又在正确的地方和正确的时间出现在了Prentice-Hall出版社，是他为了清除了所有可能遇到的障碍，也使我感受了一次愉快的出书经历。谢谢你，Jeff——你对我非常重要。

要特别感谢Gen Kiyooka和他的Digigami公司，我用的Web服务器就是他们提供的；也要感谢Scott Callaway，服务器是由他负责维护的。在我学习Web的过程中，一个服务器无疑是相当有价值的帮助。

谢谢Cay Horstmann（《Core Java》一书的副编辑，Prentice Hall于1997年出版）、D'Arcy Smith（Symantec公司）和Paul Tyma（《Java Primer Plus》一书的副编辑，The Waite

Group于1996年出版），感谢他们帮助我澄清语言方面的一些概念。

感谢那些在“Java软件开发会议”上我的Java小组发言的同志们，以及我教授过的那些学生，他们提出的问题使我的教案愈发成熟起来。

特别感谢Larry和Tina O'Brien，是他们将这本书和我的教学内容制成一张教学CD-ROM（关于这方面的问题，http://www.BruceEckel.com有更多的答案）。

有许多人送来了纠错报告，我真的很感激所有这些朋友，但特别要对下面这些人说声谢谢：Kevin Raulerson（发现了多处重大错误），Bob Resendes（发现的错误令人难以置信），John Pinto，Joe Dante，Joe Sharp，David Combs（许多语法和表达不清的地方），Dr. Robert Stephenson，Franklin Chen，Zev Griner，David Karr，Leander A. Stroschein，Steve Clark，Charles A. Lee，AustinMaher，Dennis P. Roth，Roque Oliveira，Douglas Dunn，Dejan Ristic，NeilGalarneau，David B. Malkovsky，Steve Wilkinson，以及其他许多热心读者。

为了使这本书在欧洲发行，Prof. Ir. Marc Meurrens进行了大量工作。

有一些技术人员曾走进我的生活，他们后来都和我成了朋友。最不寻常的是他们全是素食主义者，平时喜欢练习瑜珈功，以及另一些形式的精神训练。我在练习了以后，觉得对我保持精力的旺盛非常有好处。他们是Kraig Brockschmidt，GenKiyooka和Andrea provaglio，是这些朋友帮我了解了Java和程序设计在意大利的情况。

显然，在Delphi上的一些经验使我更容易理解Java，因为它们有许多概念和语言设计决定是相通的。我的Delphi朋友提供了许多帮助，使我能够洞察一些不易为人注意的编程环境。他们是Marco Cantu（另一个意大利人——难道会说拉丁语的人在学习Java时有得天独厚的优势？）、Neil Rubenking（他最喜欢瑜珈／素食／禅道，但也非常喜欢计算机）以及Zack Urlocker（是我游历世界时碰面次数最多的一位同志）。

我的朋友Richard Hale Shaw（以及Kim）的一些意见和支持发挥了非常关键的作用。Richard和我花了数月的时间将教学内容合并到一起，并探讨如何使学生感受到最完美的学习体验。也要感谢KoAnn Vikoren，Eric Eaurot，DeborahSommers，Julie Shaw，Nicole Freeman，Cindy Blair，Barbara Hanscome，Regina Ridley，Alex Dunne以及MFI其他可敬的成员。

书籍设计、封面设计以及封面照片是由我的朋友Daniel Will-Harris制作的。他是一位著名的作家和设计家（http://www.WillHarris.com），在初中的时候就已显露出了过人的数学天赋。但是，小样是由我制作的，所以录入错误都是我的。我是用Microsoft Word 97 for Windows来写这本书，并用它生成小样。正文字体采用的是Bitstream Carmina；标题采用Bitstream Calligraph 421（www.bitstream.com）；每章开头的符号采用的是来自P22的Leonardo Extras（http://www.p22.com）；封面字体采用ITC Rennie Marckintosh。

感谢为我提供编译器程序的一些著名公司：Borland，Microsoft，Symantec，Sybase/Powersoft/Watcom以及Sun。

特别感谢我的老师和我所有的学生（他们也是我的老师），其中最有趣的一位写作老师是Gabrielle Rico（《Writing the Natural Way》一书的作者，Putnam于1983年出版）。

曾向我提供过支持的朋友包括（当然还不止）：Andrew Binstock，SteveSinofsky，JD Hildebrandt，Tom Keffer，Brian McElhinney，Brinkley Barr，《Midnight Engineering》杂志社的Bill Gates，Larry Constantine和LucyLockwood，Greg Perry，Dan Putterman，Christi Westphal，Gene Wang，DaveMayer，David Intersimone，Andrea Rosenfield，Claire Sawyers，另一些意大利朋友（Laura Fallai，Corrado，Ilsa和Cristina Giustozzi），Chris和Laura Strand，Almquists，Brad Jerbic，Marilyng Cvitanic，Mabrys，Haflingers，Pollocks，Peter Vinci，Robbins Families，Moelter Families（和McMillans），Michael Wilk，Dave Stoner，Laurie Adams，Cranstons，Larry Fogg，Mike和Karen Sequeira，Gary Entsminger和Allison Brody，KevinDonovan和Sonda Eastlack，Chester和Shannon Andersen，Joe Lordi，Dave和Brenda Bartlett，David Lee，Rentschlers，Sudeks，Dick，Patty和Lee Eckel，Lynn和Todd以及他们的家人。最后，当然还有我的爸爸和妈妈。

第1章 对象入门

“为什么面向对象的编程会在软件开发领域造成如此震憾的影响？”

面向对象编程（OOP）具有多方面的吸引力。对管理人员，它实现了更快和更廉价的开发与维护过程。对分析与设计人员，建模处理变得更加简单，能生成清晰、易于维护的设计方案。对程序员，对象模型显得如此高雅和浅显。此外，面向对象工具以及库的巨大威力使编程成为一项更使人愉悦的任务。每个人都可从中获益，至少表面如此。

如果说它有缺点，那就是掌握它需付出的代价。思考对象的时候，需要采用形象思维，而不是程序化的思维。与程序化设计相比，对象的设计过程更具挑战性——特别是在尝试创建可重复使用（可再生）的对象时。过去，那些初涉面向对象编程领域的人都必须进行一项令人痛苦的选择：

(1) 选择一种诸如Smalltalk的语言，“出师”前必须掌握一个巨型的库。

(2) 选择几乎根本没有库的C++（注释①），然后深入学习这种语言，直至能自行编写对象库。

①：幸运的是，这一情况已有明显改观。现在有第三方库以及标准的C++库供选用。

事实上，很难很好地设计出对象——从而很难设计好任何东西。因此，只有数量相当少的“专家”能设计出最好的对象，然后让其他人享用。对于成功的OOP语言，它们不仅集成了这种语言的语法以及一个编译程序（编译器），而且还有一个成功的开发环境，其中包含设计优良、易于使用的库。所以，大多数程序员的首要任务就是用现有的对象解决自己的应用问题。本章的目标就是向大家揭示出面向对象编程的概念，并证明它有多么简单。

本章将向大家解释Java的多项设计思想，并从概念上解释面向对象的程序设计。但要注意在阅读完本章后，并不能立即编写出全功能的Java程序。所有详细的说明和示例会在本书的其他章节慢慢道来。

1.1 抽象的进步

所有编程语言的最终目的都是提供一种“抽象”方法。一种较有争议的说法是：解决问题的复杂程度直接取决于抽象的种类及质量。这儿的“种类”是指准备对什么进行“抽象”？汇编语言是对基础机器的少量抽象。后来的许多“命令式”语言（如FORTRAN，BASIC和C）是对汇编语言的一种抽象。与汇编语言相比，这些语言已有了长足的进步，但它们的抽象原理依然要求我们着重考虑计算机的结构，而非考虑问题本身的结构。在机器模型（位于“方案空间”）与实际解决的问题模型（位于“问题空间”）之间，程序员必须建立起一种联系。这个过程要求人们付出较大的精力，而且由于它脱离了编程语言本身的范围，造成程序代码很难编写，而且要花较大的代价进行维护。由此造成的副作用便是一门完善的“编程方法”学科。

为机器建模的另一个方法是为要解决的问题制作模型。对一些早期语言来说，如LISP和APL，它们的做法是“从不同的角度观察世界”——“所有问题都归纳为列表”或“所有问题都归纳为算法”。PROLOG则将所有问题都归纳为决策链。对于这些语言，我们认为它们一部分是面向基于“强制”的编程，另一部分则是专为处理图形符号设计的。每种方法都有自己特殊的用途，适合解决某一类的问题。但只要超出了它们力所能及的范围，就会显得非常笨拙。

面向对象的程序设计在此基础上则跨出了一大步，程序员可利用一些工具表达问题空间内的元素。由于这种表达非常普遍，所以不必受限于特定类型的问题。我们将问题空间中的元素以及它们在方案空间的表示物称作“对象”（Object）。当然，还有一些在问题空间没有对应体的其他对象。通过添加新的对象类型，程序可进行灵活的调整，以便与特定的问题配合。所以在阅读方案的描述代码时，会读到对问题进行表达的话语。与我们以前见过的相比，这无疑是一种更加灵活、更加强大的语言抽象方法。总之，OOP允许我们根据问题来描述问题，而不是根据方案。然而，仍有一个联系途径回到计算机。每个对象都类似一台小计算机；它们有自己的状态，而且可要求它们进行特定的操作。与现实世界的“对象”或者“物体”相比，编程“对象”与它们也存在共通的地方：它们都有自己的特征和行为。

Alan Kay总结了Smalltalk的五大基本特征。这是第一种成功的面向对象程序设计语言，也是Java的基础语言。通过这些特征，我们可理解“纯粹”的面向对象程序设计方法是什么样的：

(1) 所有东西都是对象。可将对象想象成一种新型变量；它保存着数据，但可要求它对自身进行操作。理论上讲，可从要解决的问题身上提出所有概念性的组件，然后在程序中将其表达为一个对象。

(2) 程序是一大堆对象的组合；通过消息传递，各对象知道自己该做些什么。为了向对象发出请求，需向那个对象“发送一条消息”。更具体地讲，可将消息想象为一个调用请求，它调用的是从属于目标对象的一个子例程或函数。

(3) 每个对象都有自己的存储空间，可容纳其他对象。或者说，通过封装现有对象，可制作出新型对象。所以，尽管对象的概念非常简单，但在程序中却可达到任意高的复杂程度。

(4) 每个对象都有一种类型。根据语法，每个对象都是某个“类”的一个“实例”。其中，“类”（Class）是“类型”（Type）的同义词。一个类最重要的特征就是“能将什么消息发给它？”。

(5) 同一类所有对象都能接收相同的消息。这实际是别有含义的一种说法，大家不久便能理解。由于类型为“圆”（Circle）的一个对象也属于类型为“形状”（Shape）的一个对象，所以一个圆完全能接收形状消息。这意味着可让程序代码统一指挥“形状”，令其自动控制所有符合“形状”描述的对象，其中自然包括“圆”。这一特性称为对象的“可替换性”，是OOP最重要的概念之一。

一些语言设计者认为面向对象的程序设计本身并不足以方便解决所有形式的程序问题，提倡将不同的方法组合成“多形程序设计语言”（注释②）。

②：参见Timothy Budd编著的《Multiparadigm Programming in Leda》，Addison-Wesley 1995年出版。

1.2 对象的接口

亚里士多德或许是认真研究“类型”概念的第一人，他曾谈及“鱼类和鸟类”的问题。在世界首例面向对象语言Simula-67中，第一次用到了这样的一个概念：

所有对象——尽管各有特色——都属于某一系列对象的一部分，这些对象具有通用的特征和行为。在Simula-67中，首次用到了class这个关键字，它为程序引入了一个全新的类型（clas和type通常可互换使用；注释③）。

③：有些人进行了进一步的区分，他们强调“类型”决定了接口，而“类”是那个接口的一种特殊实现方式。

Simula是一个很好的例子。正如这个名字所暗示的，它的作用是“模拟”（Simulate）象“银行出纳员”这样的经典问题。在这个例子里，我们有一系列出纳员、客户、帐号以及交易等。每类成员（元素）都具有一些通用的特征：每个帐号都有一定的余额；每名出纳都能接收客户的存款；等等。与此同时，每个成员都有自己的状态；每个帐号都有不同的余额；每名出纳都有一个名字。所以在计算机程序中，能用独一无二的实体分别表示出纳员、客户、帐号以及交易。这个实体便是“对象”，而且每个对象都隶属一个特定的“类”，那个类具有自己的通用特征与行为。

因此，在面向对象的程序设计中，尽管我们真正要做的是新建各种各样的数据“类型”（Type），但几乎所有面向对象的程序设计语言都采用了“class”关键字。当您看到“type”这个字的时候，请同时想到“class”；反之亦然。

建好一个类后，可根据情况生成许多对象。随后，可将那些对象作为要解决问题中存在的元素进行处理。事实上，当我们进行面向对象的程序设计时，面临的最大一项挑战性就是：如何在“问题空间”（问题实际存在的地方）的元素与“方案空间”（对实际问题进行建模的地方，如计算机）的元素之间建立理想的“一对一”对应或映射关系。

如何利用对象完成真正有用的工作呢？必须有一种办法能向对象发出请求，令其做一些实际的事情，比如完成一次交易、在屏幕上画一些东西或者打开一个开关等等。每个对象仅能接受特定的请求。我们向对象发出的请求是通过它的“接口”（Interface）定义的，对象的“类型”或“类”则规定了它的接口形式。“类型”与“接口”的等价或对应关系是面向对象程序设计的基础。

下面让我们以电灯泡为例：

```
Light lt = new Light();

lt.on();
```

在这个例子中，类型／类的名称是Light，可向Light对象发出的请求包括包括打开（on）、关闭（off）、变得更明亮（brighten）或者变得更暗淡（dim）。通过简单地声明一个名字（lt），我们为Light对象创建了一个“句柄”。然后用new关键字新建类型为Light的一个对象。再用等号将其赋给句柄。为了向对象发送一条消息，我们列出句柄名（lt），再用一个句点符号（.）把它同消息名称（on）连接起来。从中可以看出，使用一些预先定义好的类时，我们在程序里采用的代码是非常简单和直观的。

1.3 实现方案的隐藏

为方便后面的讨论，让我们先对这一领域的从业人员作一下分类。从根本上说，大致有两方面的人员涉足面向对象的编程：“类创建者”（创建新数据类型的人）以及“客户程序员”（在自己的应用程序中采用现成数据类型的人；注释④）。对客户程序员来讲，最主要的目标就是收集一个充斥着各种类的编程“工具箱”，以便快速开发符合自己要求的应用。而对类创建者来说，他们的目标则是从头构建一个类，只向客户程序员开放有必要开放的东西（接口），其他所有细节都隐藏起来。为什么要这样做？隐藏之后，客户程序员就不能接触和改变那些细节，所以原创者不用担心自己的作品会受到非法修改，可确保它们不会对其他人造成影响。

④：感谢我的朋友Scott Meyers，是他帮我起了这个名字。

“接口”（Interface）规定了可对一个特定的对象发出哪些请求。然而，必须在某个地方存在着一些代码，以便满足这些请求。这些代码与那些隐藏起来的数据便叫作“隐藏的实现”。站在程式化程序编写（Procedural Programming）的角度，整个问题并不显得复杂。一种类型含有与每种可能的请求关联起来的函数。一旦向对象发出一个特定的请求，就会调用那个函数。我们通常将这个过程总结为向对象“发送一条消息”（提出一个请求）。对象的职责就是决定如何对这条消息作出反应（执行相应的代码）。

对于任何关系，重要一点是让牵连到的所有成员都遵守相同的规则。创建一个库时，相当于同客户程序员建立了一种关系。对方也是程序员，但他们的目标是组合出一个特定的应用（程序），或者用您的库构建一个更大的库。

若任何人都能使用一个类的所有成员，那么客户程序员可对那个类做任何事情，没有办法强制他们遵守任何约束。即便非常不愿客户程序员直接操作类内包含的一些成员，但倘若未进行访问控制，就没有办法阻止这一情况的发生——所有东西都会暴露无遗。

有两方面的原因促使我们控制对成员的访问。第一个原因是防止程序员接触他们不该接触的东西——通常是内部数据类型的设计思想。若只是为了解决特定的问题，用户只需操作接口即可，毋需明白这些信息。我们向用户提供的实际是一种服务，因为他们很容易就可看出哪些对自己非常重要，以及哪些可忽略不计。

进行访问控制的第二个原因是允许库设计人员修改内部结构，不用担心它会对客户程序员造成什么影响。例如，我们最开始可能设计了一个形式简单的类，以便简化开发。以后又决定进行改写，使其更快地运行。若接口与实现方法早已隔离开，并分别受到保护，就可放心做到这一点，只要求用户重新链接一下即可。

Java采用三个显式（明确）关键字以及一个隐式（暗示）关键字来设置类边界：public，private，protected以及暗示性的friendly。若未明确指定其他关键字，则默认为后者。这些关键字的使用和含义都是相当直观的，它们决定了谁能使用后续的定义内容。“public”（公共）意味着后续的定义任何人均可使用。而在另一方面，“private”（私有）意味着除您自己、类型的创建者以及那个类型的内部函数成员，其他任何人都不能访问后续的定义信息。private在您与客户程序员之间竖起了一堵墙。若有人试图访问私有成员，就会得到一个编译期错误。“friendly”（友好的）涉及“包装”或“封装”（Package）的概念——即Java用来构建库的方法。若某样东西是“友好的”，意味着它只能在这个包装的范围内使用（所以这一访问级别有时也叫作“包装访问”）。“protected”（受保护的）与“private”相似，只是一个继承的类可访问受保护的成员，但不能访问私有成员。继承的问题不久就要谈到。