TURING 图灵程序设计丛书

Addison Wesley

C++ Common Knowledge
Essential Intermediate Programming

C++必知必会

[美] Stephen C. Dewhurst 著
荣耀 译

人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目（CIP）数据

C++必知必会 / （美）杜赫斯特（Dewhurst, S. C.）著；荣耀译. -- 北京 : 人民邮电出版社, 2010.12
（图灵程序设计丛书）
ISBN 978-7-115-24045-3

Ⅰ. ①C… Ⅱ. ①杜… ②荣… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2010)第201410号

内 容 提 要

本书描述了C++编程和设计中必须掌握但通常被误解的主题，这些主题涉及的范围较广，包括指针操作、模板、泛型编程、异常处理、内存分配、设计模式等。作者根据本人以及其他有经验的管理人员和培训老师的经验总结，对与这些主题相关的知识进行了精心挑选，最终浓缩成63条。每一条款所包含的内容均为进行产品级C++编程所需的关键知识。作者称这些知识为C++程序员必备的“常识”，其实并非意味着简单或平庸，而是“必不可少”。

本书适合于中、高级C++程序员，也适合C或Java程序员转向C++程序设计时参考。

图灵程序设计丛书
C++必知必会

◆ 著 [美] Stephen C. Dewhurst
译 荣 耀
责任编辑 傅志红
执行编辑 郝富强

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：13.75
字数：288千字 2010年12月第2版
印数：7 001－10 000册 2010年12月北京第1次印刷

著作权合同登记号 图字：01-2005-3575号

ISBN 978-7-115-24045-3

定价：39.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

对本书的赞誉

“我们这个时代C++佳作不断涌现，本书不愧为其中的一员。尽管C++屹立于软件创新和生产力的最前沿已有20余年，但时至今日它才被充分理解和运用。这是一本难得的值得C++实践者和专家反复研习的佳作。它并非那种晦涩难懂的学术气的论文，而是着力帮助你完善对C++精髓的理解——这些重要知识你自以为已知，但除非真正掌握否则早晚会栽跟头。很少有人能像Steve这样精通C++和软件设计，而在软件开发方面更是无人具备Steve这般冷静的头脑。他知道你需要学习了解些什么。相信我！当他发话时，我总是洗耳恭听。我希望你也能这样。你（和你的客户）终将为此感到高兴。”

——Chuck Allison，*The C++ Source* 编辑

“Steve曾经教我学C++。这要追溯到1982或1983年，我想他那时刚在贝尔实验室以实习生的身份和Bjarne Stroustrup（C++的发明者）共事过一段时间。Steve是位C++元老，但未有太多人关注，他的任何作品我都优先列入阅读计划。本书是对Steve大量知识与经验的汇总，易于阅读，我强烈推荐！”

——Stan Lippman，*C++ Primer*（第4版）作者之一

“作者有意使本书成为一本短小精悍的非傻瓜书，我喜欢这种风格。”

——Matthew P. Johnson，哥伦比亚大学

“我赞同（作者）对不同类型的程序员的评价。作为一个开发者，我就遇到过这些类型的程序员，这样的书有助于填补他们的知识鸿沟……我认为这本书是对其他书籍（比如Scott Meyers的*Effective C++*）很好的补充。它以一种简练且易读的风格将一切知识展现于你的面前。”

——Moataz Kamel，资深软件设计师，摩托罗拉加拿大公司

“Dewhurst又写了一本佳作。本书应该是那些正在使用C++（并自以为对C++无所不知）的人们的必备读物。”

——Clovis Tondo，*C++ Primer Answer Book* 作者之一

译者序

尽管C++越来越像是“专家专用”的语言，精通它需要付出极大的努力，然而并非每一个人都需要成为C++语言专家。对于大多数人而言，学习C++的目的是为了“致用”，而非研究语言本身。我们的精力应该放在有效地掌握编程必备知识上，以便能够胜任目标领域的软件开发。职业程序员往往更应该是“领域专家”而非“语言专家”。

本书提供了C++程序员所必须具备的“常识”。这里所说的“常识”并非意味着简单或平庸，而是指“必不可少”，事实上，有些内容相当高级，比如设计模式和泛型编程。本书对散布于许多其他C++书籍中的知识进行了精心挑选，最终浓缩成63条。每一条款相对独立，可以随机查阅。许多条款内部还含有交叉索引，便于加深对该主题的理解。

这些条款涉及的主题范围较广，除了指针操作、面向对象、异常处理以及内存分配等外，对于现代C++编程技术亦有很好的描述，其中仅设计模式就占了好几个条款。除了一个总论性的条款外，另外还具体介绍了Prototype、Factory Method、Command、Template Method等经典模式，而trait、policy以及智能指针亦可归入这个范畴。作者“短、平、快”式的介绍，可以使你迅速掌握这些常用模式的概念和用法。

现代C++程序员应该像熟悉面向对象编程那样熟悉模板和泛型编程。本书中，模板和泛型编程内容占了条款总数的近1/3，其中包括：类模板显式特化、模板局部特化、类模板成员特化、成员模板、嵌入的类型信息、模板的模板参数、模板实参推导、重载函数模板等等。这些都是进行泛型编程不可或缺的知识。

作者Steve Dewhurst是贝尔实验室C++元老之一，有着20多年C++应用经验，所解决的问题涉及多个领域，并是两款C++编译器的作者。他的文风一向简练明快，并不失尖锐。阅读本书，可以给你带来“拨开迷雾见青天”的感觉。

根据我对国内C++应用现状的了解，我认为本书首先适合业界程序员用作快速参考。一些程序员项目经验不少，但对C++的使用仅限于一小套子集，而且往往是一套原始的子集。这本小册子可以快速弥补这方面的知识结构缺陷。作者奉行“有所为、有所不为”的指导思想，忽略了复杂而很少使用的细节，而是带领读者直奔主题，抓住重点。此外，有了实战经

验作为后盾，本书很容易上手。

对于已经系统地学习过一门C++课程的在校大学生来说，这本小书可以开阔你的眼界。如果你的C++基础尚不足以顺畅地阅读书中部分条款，读来如雾里看花，或因作者点到为止而感觉意犹未尽，可以考虑选读书末“参考文献”中列出的书籍，它们大都有高品质的中文版。

在沉寂许久之后，一批优秀的C++新作终于陆续面世，我有幸参与翻译包括*C++ Common Knowledge Imperfect C++* 以及 *C++ Template Metaprogramming* 等在内的几本佳作。希望在第一时间完成翻译的这本新书，能够给期待已久的你带来新鲜的快乐！

前言

一本书之所以成功，不在于书本身的内容，而在于它的未尽之言。

——马克·吐温

……尽可能简单，但不过分简单。

——阿尔伯特·爱因斯坦

……一个对读者的能力持怀疑态度的作家根本不能称其为作家，只不过是个阴谋家而已。

——E. B. 怀特

当 Herb Sutter 接手 *C++ Report* 的编辑工作时，他很快就邀我为之写一个专栏，主题由我来定。我将该专栏命名为“Common Knowledge”（常识）。用 Herb 的话来说，该专栏预期为“对每一位职业 C++ 程序员应该知道但未必知道的基础知识之定期概述”。然而，在以那样的风格写了一些专栏文章后，我对模板元编程（template metaprogramming）技术的兴趣日渐浓厚，故而此后“Common Knowledge”中讨论的一些主题距离“Common”越来越远。

然而，在C++程序设计界，当初促使我选定写这个专栏的问题仍然存在。在我的培训和咨询工作中，常常会遇到下面几类人员：

- 领域专家，他们是专家级的C程序员，但对C++只有一些基本的认知（并可能对 C++ 没有好感）；
- 直接从大学雇来的新手，他们有才干，但对 C++ 语言只有理论上的认识，缺乏实际产品开发经验；
- 专家级的 Java 程序员，他们仅有少量的 C++ 经验，常会以 Java 的方式来从事 C++ 编程；
- C++ 程序员，他们具有若干年维护现有 C++ 应用程序的经验，但尚未经受学习高级编程知识的挑战。

我希望能即刻进行建设性的工作，但是，许多我共事过的或培训过的人都需要先接受形形色色的关于C++语言特性、模式以及编程技术的预备性的教育，才有能力处理业务问题。

更糟糕的是，我怀疑大多数C++代码在编写时都至少忽略了一些基本要素，因而不具备大多数C++专家所认可的产品级的质量。

这本书致力于解决这个具有普遍性的问题，它提供了每一位职业C++程序员需要知道的常识，并且这些常识都被精简至本质，因此可被高效而精确地吸收。其中有不少信息也可以从其他途径获得，而有些知识则是所有专家级C++程序员知道但未成文信息的完整摘要。本书优势在于，这些材料现在被集中于一处，并依据我多年的培训和咨询经验进行了遴选，经验表明，这些都是最常被误解同时也是最有用的语言特性、概念和技术。

也许构成本书的63个简短条款最重要的方面在于它们所省略掉的东西，而不是它们所包含的东西。许多主题都可以进行更复杂的讨论。如果忽略掉这些复杂性，会导致传达的信息不够充分，从而可能会误导读者，但对一个主题的全部复杂性进行专家级的讨论，又可能会使读者应接不暇。本书采取的方式是在讨论每一个主题时过滤掉那些“不必要”的复杂性。我希望有幸留下的这些东西是对产品级C++编程所必需的知识的清晰萃取。较真的C++语言专家可能意识到我没有讨论某些有趣的甚至重要的问题（从理论的角度来说），但我所省略的那些东西通常并不会影响阅读和编写产品级C++代码的能力。

写作这本书的另一个动机来自于我在一次会议上同一群知名C++专家的谈话。这些专家对于一件事情颇感沮丧，那就是他们认为现代C++是如此复杂，以至于“普通”程序员已经不再能够理解它了（比如，在模板和名字空间上下文中的名字绑定问题。是的，解决这样的问题确实需要普通C++程序员下更多的功夫）。在我看来，应该说其实我们的态度有些过于自负了，我们的沮丧也是不合情理的。我们这些“专家”们自己就不存在这样的问题，实际上，使用C++编程就像说一门（远比C++复杂的）自然语言那样容易，尽管我们不能完全分析我们所说的每一句话的语法结构。本书不断出现的一个主题是，虽然对特定语言特性细节的完整描述可能让人望而生畏，但是，日常使用的语言特性都是直观而自然的。

不妨考虑一下函数重载。有关它的完整描述占据了很大一块标准文档，并占据了许多C++教程的一整章（甚至多章）。然而，当我们面对如下代码时

```
void f( int );
void f( const char * );
//...
f( "Hello" );
```

一个职业C++程序员是不可能不知道哪一个f被调用的。有关重载函数调用解析规则的完整知识当然是有意义的，但很少会用到。同样的道理适用于其他许多看上去很复杂的C++语言特性和惯用法。

这并不是说本书中出现的所有内容都很简单，它们“尽可能简单，但不过分简单”。在

C++编程中，以及任何其他值得从事的智力活动中，许多重要的细节都无法写在“索引卡片”上。此外，这并不是一本“傻瓜”书。我感觉自己对那些挤出宝贵时间来阅读我的书的读者负有极大的责任。我尊重这些读者，并且努力与他们交流，就像我亲自与同事交流一样。我认为给职业人员写初中水平的东西算不上写作，不过是想低就迎合而已。

本书中的许多条款针对的是一些简单的误解，这些误解都是我曾一再看见的，只要予以指正即可，例如成员函数查找的作用域顺序、重写（override）和重载（overload）之间的区别等。另外一些条款则论述那些逐渐为职业C++程序员所必需、但常常又被错误地认为过于困难并因而被避免使用的知识，例如类模板局部特化（template partial specialization）和模板的模板参数（template template parameter）等。为此我受到了一些专家级审稿人的批评，说我在模板问题上花费的篇幅过多（约占全书的1/3），而这些知识并非真的是“常识”。然而，这其中每一位专家又都指出有一两个甚至好几个模板主题应该包含于本书之中。一个有趣的现象是，这些建议中几乎没有重叠，每一个和模板有关的条款都至少有一个支持者。

这就是构成本书所含条款的问题症结。我并不认为有哪一位读者对本书每一个条款所谈的主题都一无所知，而且我还认为甚至有人熟悉本书中的所有条款。显然，如果某一位读者对某个特定的主题不熟悉，我认为阅读本书应该会从中受益。然而，即使某一位读者已经熟悉某个主题，我还是希望他能从一个全新的角度来阅读它，这样也许能够澄清一些轻微的误解或进一步加深对该主题的理解。这本书可能还有一个作用，那就是可以为富有经验的C++程序员节省宝贵的时间。那些能干的C++程序员常常发现他们一再被问及同样的问题，从而影响了他们自身的工作。下次他们可以说：“先读一读《C++必知必会》，再来和我讨论该问题。”这样能够为这些C++专家节省大量的时间，从而允许他们将自己的专家经验用在更复杂的问题上，而这些问题才是需要专家来解决的。

起先我试图将这63个条款分组到若干章中，那样显得更加整洁，但这些条款自己却似乎并不乐意。它们往往彼此簇拥在一起，有时道理很明显，有时则有点出乎意料。举个例子，与异常和资源管理有关的条款形成了相当自然的一个组；而“能力查询”、“指针比较的含义”、“虚构造函数与Prototype模式”、“Factory Method模式”以及“协变返回类型”这几个条款之间的关系紧密得有点出乎意料，因而最好被安排在一起；“指针算术”和“智能指针”放在一块儿，而不是和出现于本书较早部分的指针和数组方面的素材放在一起。因此，我不再去武断地将章式结构强加于这些自然的分组上，而是决定让各个条款自由结合起来。自然而然，很多条款涉及的主题之间存在相互关系，简单的线形顺序很难表达出这一点，因此条款中还频繁出现内部交叉引用。所以说，它们是一个簇拥但紧密连接的共同体。

尽管本书写作的主要指导思想是短小精悍，但对一个主题的讨论有时会包括一些辅助性的细节，尽管它们和眼前讨论的主题并不直接相关。对于该主题来说，这些细节并非必需，

但读者由此可注意到特定程序或技术的存在。例如，在好几个条款中都出现的Heap模板例子可以让读者顺便了解到有用但很少被讨论到的STL堆算法，而对placement new的讨论则勾画出许多标准库组件所用到的复杂老练的内存管理技术基础。只要这么做看起来很自然，我就会利用机会，将辅助性话题的讨论混入某个特定的具名条款中。因此，条款40“RAII”包含了对构造函数和析构函数激活顺序的简短讨论，条款57“模板实参推导”讨论了用于特化类模板的辅助函数的使用，条款12“赋值和初始化并不相同”则混入了对计算性的构造函数的讨论。这本书的条款数目本来很容易翻倍，但是，就像这些簇拥的条款自身一样，辅助性话题和具体条款的相关性，使得该主题被放置于合适的上下文之中，并且有助于读者高效、精确地吸收所表达的知识。

我很不情愿地加进了几个不适合在本书中讨论的主题（要知道，本书的写作风格是条款简短）。特别是有关设计模式和标准模板库的设计方面的主题，看上去短得可笑，很不完整。它们的现身只是为了消除一些常见的误解，并强调这些主题的重要性，从而鼓励读者去学习该主题更多的东西。

就像团聚在一起度假的家庭成员交流各自的趣事一样，常用到的例子早已成为我们编程文化的一部分，因此Shape、String、Stack以及任何其他常见的“嫌犯”都一一露面。对这些基准例子达成共识，可以使我们的交流像使用设计模式那样高效。例如“设想我希望旋转（rotate）一个Shape，除了……之外”，或者“当拼接两个String时……”这些常见的例子更适合于交流，可以避免费时的背景介绍，比如“你知道当你的兄弟被逮捕时的表现吗？呃，说来话长，前些天……”

有别于我以前写的书，本书试图避免对一些糟糕的编程实践以及对C++语言特性误用作出评判——那是其他一些书的目标，其中最好的一些书已被我列于“参考文献”之中（但我免不了会有一些说教的倾向，在本书中还是提到了一些糟糕的编程实践，虽然只是顺带一提）。一句话，本书的目的在于以尽可能高效的方式告诉读者产品级C++编程所必需的技术。

Stephen C. Dewhurst
2005年1月于马萨诸塞州卡沃尔

致谢

在忍受了我对C++社群教育现状满腹牢骚很长一段时间后，编辑Peter Gordon（一位非凡的人士）建议我为之做一些实事，本书正是该建议的结果。Kim Boedigheimer设法保持本书的一切事务顺利进行，使我未曾受到哪怕一次与写作有关的“威胁”。

感谢专家级技术审稿人 Matthew Johnson、Moataz Kamel、Dan Saks、Clovis Tondo 以及 Matthew Wilson，他们指出了手稿中的一些错误和语言表达不当之处，从而使本书变得更出色。作为一个老顽固，我并没有完全采纳他们的建议，因此，书中残存的任何错误和不妥，全是我的错。

本书中的一些材料曾出现于我在 *C/C++ Users Journal* 上开设的“Common Knowledge”专栏中，二者之间只有些细微的差别。另有不少材料曾经出现于 semantics.org 上的“Once, Weakly”Web专栏中。我曾收到一些人士就印刷版和网络版文章提出的富有洞察力的意见，他们是Chuck Allison、Attila Fehér、Kevlin Henney、Thorsten Ottosen、Dan Saks、Terje Slettebø、Herb Sutter 以及 Leor Zolman。几次三番与 Dan Saks 的深入讨论使我对模板的特化和实例化之间的区别有了更深刻的理解，并帮助我澄清了普通重载与 ADL（Argument Dependent Lookup，实参相依的查找）以及“中缀操作符查找”（infix operator lookup）中出现的重载的区别。

本书同时还得到一些间接的贡献。非常感谢 Brandon Goldfedder 对出现于本书“设计模式”条款中的算法类比建议。感谢Clovis Tondo的激励以及在寻找优秀的审稿人方面给予的协助。我很幸运能够连续几年教授基于 Scott Meyers 的 *Effective C++*、*More Effective C++* 和 *Effective STL* 3 本书的课程。这使我获得了第一手的资料，让我认识到学习这些实践型、中等难度的C++书籍的学生们通常欠缺哪些背景知识，这些观察帮助我确定了本书所要讨论的主题。Andrei Alexandrescu的作品鼓舞我对模板元编程进行实验，而不是仅凭主观臆断行事。Herb Sutter 和 Jack Reeves 在异常方面的工作帮助我更好地理解应该如何使用异常。

我还要感谢邻居兼好友Dick和Judy Ward夫妇，他们定期把我从计算机面前叫走，让我参加当地的蔓越桔采摘活动。对于我这个以处理对现实世界的简单抽象作为职业的人来说，这真是极有益于身心的调剂。从某种意义上讲，说服蔓越桔结果实和尝试模板局部特化这二者之间的复杂性还真是有一拼。

一如既往，Sarah G. Hewins 和 David R. Dewhurst 为本书的写作提供了颇有价值的协助（当然也设置了一些必要的“障碍”）。

我自认为是一个性格沉稳的人，惯于自省而讨厌乌鸦般的喋喋不休。然而，就像有些人一旦手握方向盘就跟换了个人似的，我在写作本书手稿时差不多就是这个状态。Addison-Wesley 厉害的“行为矫正专家”们识破了我性格上的弱点。Chanda Leary Coutu 与 Peter Gordon 、Kim Boedigheimer 齐心协力将我的手稿从感性的宣泄转换为理性的商业提议并监管实施。Molly Sharp 和 Julie Nahil 不仅将笨拙的 Word 文档变成现在看到的优美的页面，还改正了手稿中存在的诸多错误，并且保留了我那古老的句式结构、非同寻常的措辞以及特殊的连字符[①]等写作风格。尽管我的要求变个不停，但Richard Evans还是设法使得本书如期出版，并且制作了两份单独的索引。Chuti Prasertsith则为本书设计了精美的蔓越桔封面。多谢诸位！

① 作者对连字符的用法比较怪异，不过你从中文版里看不出这一点。此外，不同于其他C++书籍的是，本书还提供了一份“代码示例索引”。——译者注

目录

条款1
数据抽象

“类型”是一组操作，“抽象数据类型”则是一组具有某种实现的操作。当我们在某个问题领域中识别对象时，首先考虑的问题是“可以用这个对象做什么”而不是“这个对象是如何实现的”。因此，如果某个问题的自然描述涉及雇员、合同和薪水记录，那么用来解决该问题的编程语言就应该包含Employee、Contract和PayrollRecord类型。这样就可以在问题领域和解决方案领域之间进行双向、高效地转换，用这种方式编写的软件才能尽量避免产生“转换噪音”，从而做到更简洁、更准确。

在C++这样的通用编程语言中，不会有像Employee这样特定于应用程序的类型，我们有更好的东西：C++为创建复杂的抽象数据类型提供了便利。从本质上说，抽象数据类型的用途在于将编程语言扩展到一个特定的问题领域。

C++中不存在针对抽象数据类型设计的公认方案，这方面的编程依然需要灵感和艺术才能，不过许多成功的方法都遵循下面这组类似的步骤。

（1）为类型取一个描述性的名字。如果很难给这个类型命名，那就说明你还不知道你想要实现什么，你需要多开动脑筋。一个抽象数据类型应该表示一个单一的、明确的概念，而且为该概念所取的名字应该是显而易见的。

（2）列出类型所能执行的操作。定义一个抽象数据类型的依据是能用它做什么。不要忘了初始化（构造函数）、清理（析构函数）、复制（复制操作）以及转换（不带explicit关键字修饰的单参数构造函数和转换操作符）。要避免在实现时简单地为数据成员提供一串get/set（获取/设置）操作——那不是数据抽象，而是懒惰且缺乏想象力的表现。

（3）为类型设计接口。正如Scott Meyers告诉我们的那样，一个类型应该做到“易于正确使用、难以错误使用”。既然抽象数据类型是对语言的扩展，那么务必要正确地进行语言设计。你要为类型的用户设身处地地想一想，并且编写一些使用类型接口的代码。良好的接口设计除了需要高超的技术能力之外，还需要理解用户的心理，做到与用户心灵相通。

（4）实现类型。不要让实现影响类型的接口。要实现类型的接口所承诺的约定。记住，在大多数情况下，对抽象数据类型的实现的改动，远比对其接口的改动来得频繁。

条款 2
多态

多态（polymorphism）在一些编程教程中享有不可思议的地位，而在另外一些教程中则被忽略，其实它不过是C++语言所支持的一个简单而有用的概念。按照C++标准所言，“多态类型”（polymorphic type）就是带有虚函数的类类型（class type）。从设计的角度来看，“多态对象”（polymorphic object）就是一个具有不止一种类型的对象，而“多态基类”（polymorphic base class）则是一个为满足多态对象的使用需求而设计的基类。

让我们来看一个金融期权的类型 AmOption，如图 1 所示。

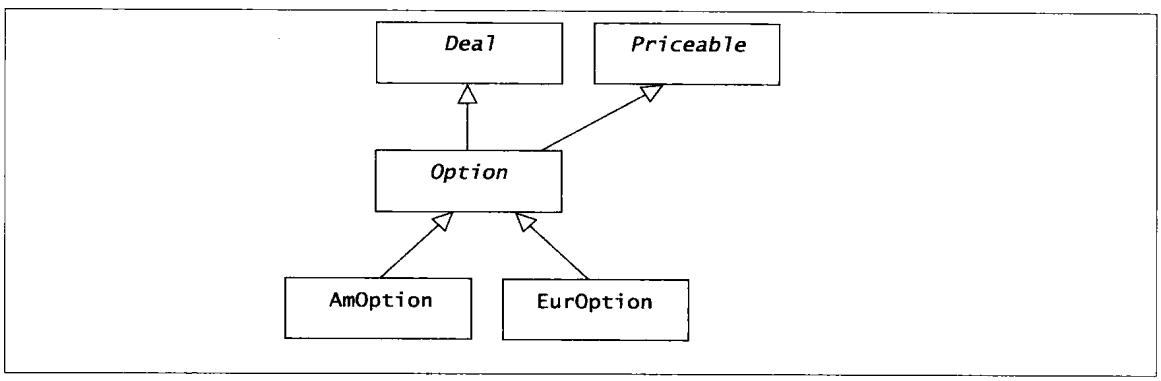

图 1　在一个金融期权层次结构中多态的作用。AmOption 有四种类型

AmOption 对象同时具有四种类型：AmOption、Option、Deal 以及 Priceable。由于一个类型是一组操作（参见“数据抽象”[条款 1]和“能力查询”[条款 27]），因此，AmOption对象可以通过其4个接口中的任何一个进行操纵。这意味着一个AmOption对象可以被针对Deal、Priceable和Option接口编写的代码所操纵，从而允许AmOption的实现利用或复用所有那些代码。对于AmOption这样的多态类型，从基类继承的最重要的东西就是它们的接口，而不是它们的实现。事实上，仅仅由接口组成的基类不但很常见，而且通常正是我们所希望见到的（参见“能力查询”[条款 27]）。

当然，这里有一个需要注意的地方。如果让这种优势能够发挥出来，一个良好设计的多