

TURING

图灵程序设计丛书

PEARSON
Addison
Wesley

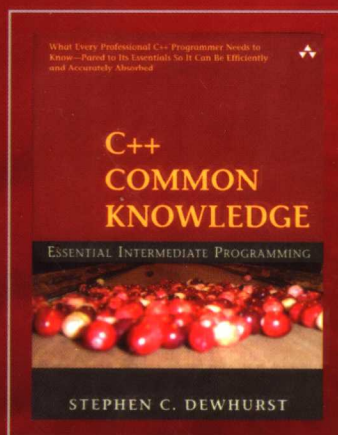
C++ Common Knowledge

Essential Intermediate Programming

C++ 必知必会

[美] Stephen C. Dewhurst 著
荣耀 译

- 职业 C++ 程序员必备常识
- 初学者登堂入室的阶梯
- C++ 界 20 年经验结晶



人民邮电出版社
POSTS & TELECOM PRESS

TURING

图灵程序设计丛书

C++ 必知必会

C++ Common Knowledge

Essential Intermediate Programming

[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, 2006.1

(图灵程序设计丛书)

ISBN 7-115-14101-0

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

中国版本图书馆 CIP 数据核字 (2005) 第 120478 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 陈贤舜
 - ◆ 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>
北京顺义振华印刷厂印刷
新华书店总店北京发行所经销
 - ◆ 开本: 800×1000 1/16
印张: 13.75
字数: 289 千字 2006 年 1 月第 1 版
印数: 1-5 000 册 2006 年 1 月北京第 1 次印刷

著作权合同登记号 图字: 01-2005-3575 号

ISBN 7-115-14101-0/TP·5034

定价: 29.00 元

读者服务热线: (010) 88593802 印装质量热线: (010) 67129223

版权声明

Authorized translation from the English language edition, entitled: *C++ COMMON KNOWLEDGE: ESSENTIAL INTERMEDIATE PROGRAMMING*, 1st Edition, 0321321928 by DEWHURST, STEPHEN C., published by Pearson Education, Inc., publishing as Addison-Wesley Professional, Copyright © 2005 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

CHINESE SIMPLIFIED language edition published by PEARSON EDUCATION ASIA LTD. and Posts & Telecommunications Press Copyright © 2005.

本书中文简体字版由 Pearson Education Asia Ltd. 授权人民邮电出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

本书封面贴有 Pearson Education (培生教育出版集团) 激光防伪标签，无标签者不得销售。版权所有，侵权必究。

对本书的赞誉

“我们正赶上 C++ 佳作不断涌现的时代，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——这些都是你自以为已知但只有真正掌握否则早晚会栽跟头的重要知识。很少有人能像 Steve 这样精通 C++ 和软件设计。在软件开发方面几乎无人具备 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 是未引起过多关注的早期英雄之一，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之间的知识鸿沟……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主题外，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。

本着对国内 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，便于直奔主题，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“参考书目”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！

荣耀

2005 年 6 月

南京师范大学

royal@royaloo.com

前言

一本成功的书不是由书中的内容所构成的，而是由该书省略的内容所构成的。

——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总是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++ 怀有敌意）。
- 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 C++ 产品开发经验。
- 专家级的 Java 程序员，他们仅有少量的 C++ 经验，并有以 Java 编程的方式来从事 C++ 编程的倾向。
- C++ 程序员，他们具有若干年维护现有 C++ 应用程序的经验，但尚未经受学习超出维护程序所需基础知识的任何东西之挑战。

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在有能力解决

业务之前，都需要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。更糟糕的是，我怀疑大多数 C++ 代码都至少忽略了其中一些基本要素，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。一般作者对这些复杂性的忽略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，又可能会使读者淹没于知识的海洋之中。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省略了对一些有趣的、甚至重要的问题（从理论的角度来说）的讨论，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，并占据了許多 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，但很少需要用到。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。

这并不是说所有展示于本书中的材料都很简单，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。我认为给职业人员看八级难度的东西算不上写作，不过是想迎合少数人的胃口而已。

本书中的许多条款描述的是一些简单的误解，这些误解都是我曾一再见到的，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正在成为职业 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，那就是可以节省更多有经验的 C++ 程序员的宝贵时间。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。希望“先读一读《C++ 必知必会》，再来和我讨论该问题”的回答方式，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，但这些条款自己却并不这么认为。它们倾向于彼此簇拥在一起，就这一点而言，有些很明显，有些则有点出乎意料。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。而不那么明显的是，“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，我不再试图将武断的章式结构强加于这些自然的分组上，而是决定授予个体条款自由结合权。自然而然，很多条款涉及的主题之

间存在许多其他相互关系，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。对于该主题的讨论来说，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“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“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“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。

我很不情愿地包含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有关该主题更多的东西。

就像团聚在一起度假的家庭成员交换各自的趣事一样，提供例子早已成为我们编程文化的一个组成部分，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 rotate（旋转）一个 Shape，除了……之外”，或者“当连接两个 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，前些天……”

有别于我以前写的书，本书试图避免传递对一些糟糕的编程实践以及对 C++ 语言特性误用的评判——那是其他一些书的目标，其中最好的一些书已被我列于“参考书目”之中（不过，我并没有完全成功地避免说教的倾向，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++ 编程所需的本质技术。

Stephen C. Dewhurst

马萨诸塞州 卡沃尔

2005 年 1 月

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，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评论，他们是 Chuck Allison、Attila Feher、Kevlin Henney、Thorsten Ottosen、Dan Saks、Terje Slettebo、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。

本书同时还依赖于一些间接的贡献。非常感谢 Brandon Goldfeder 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，让我认识到那些期望从符合业界标准的、中等难度的 C++ 书籍中获益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。

我还要感谢邻居好友 Dick 和 Judy Ward，他们定期把我从计算机面前叫走，让我参加当地的蔓越桔收获活动。对于那些职业工作主要是处理对现实世界的简单抽象的人来说，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缺点。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	1
条款 2 多态	2
条款 3 设计模式	5
条款 4 STL	8
条款 5 引用是别名而非指针	10
条款 6 数组形参	13
条款 7 常量指针与指向常量的指针	16
条款 8 指向指针的指针	19
条款 9 新式转型操作符	21
条款 10 常量成员函数的含义	25
条款 11 编译器会在类中放东西	29
条款 12 赋值和初始化并不相同	31
条款 13 复制操作	34
条款 14 函数指针	37
条款 15 指向类成员的指针并非指针	40
条款 16 指向成员函数的指针并非指针	43
条款 17 处理函数和数组声明	46
条款 18 函数对象	48
条款 19 Command 模式与好莱坞法则	52

条款 20	STL函数对象	55
条款 21	重载与重写并不相同	58
条款 22	Template Method模式	60
条款 23	名字空间	62
条款 24	成员函数查找	66
条款 25	实参相依的查找	68
条款 26	操作符函数查找	70
条款 27	能力查询	72
条款 28	指针比较的含义	75
条款 29	虚构造函数与 Prototype 模式	77
条款 30	Factory Method模式	79
条款 31	协变返回类型	82
条款 32	禁止复制	85
条款 33	制造抽象基类	86
条款 34	禁止或强制使用堆分配	88
条款 35	placement new	90
条款 36	特定于类的内存管理	93
条款 37	数组分配	97
条款 38	异常安全公理	100
条款 39	异常安全的函数	103
条款 40	RAII	106
条款 41	new、构造函数和异常	110
条款 42	智能指针	112
条款 43	auto_ptr 非同寻常	114
条款 44	指针算术	116

条款 45	模板术语	119
条款 46	类模板显式特化	121
条款 47	模板局部特化	125
条款 48	类模板成员特化	129
条款 49	采用 <code>typename</code> 消除歧义	132
条款 50	成员模板	136
条款 51	采用 <code>template</code> 消除歧义	140
条款 52	针对类型信息的特化	142
条款 53	嵌入的类型信息	146
条款 54	<code>traits</code>	149
条款 55	模板的模板参数	154
条款 56	<code>policy</code>	159
条款 57	模板实参推导	163
条款 58	重载函数模板	167
条款 59	<code>SFINAE</code>	169
条款 60	泛型算法	172
条款 61	只实例化要用的东西	176
条款 62	包含哨位	179
条款 63	可选的关键字	181
参考文献	184
索引	185
代码示例索引	195

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“可以用这个对象来做什么”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到雇员、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。这样就允许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”，从而达到更简洁、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，不过许多成功的途径都遵循下面这组类似的步骤。

(1) 为类型选择一个描述性的名字。如果难以为这个类型命名，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纯的、有着良好定义的概念，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数据成员提供一串 `get/set`（获取/设置）操作——那不叫数据抽象，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。良好的接口设计除需考虑技术的威力外，心理学和情感方面的问题同样需要加以考虑。

(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。

1

2