

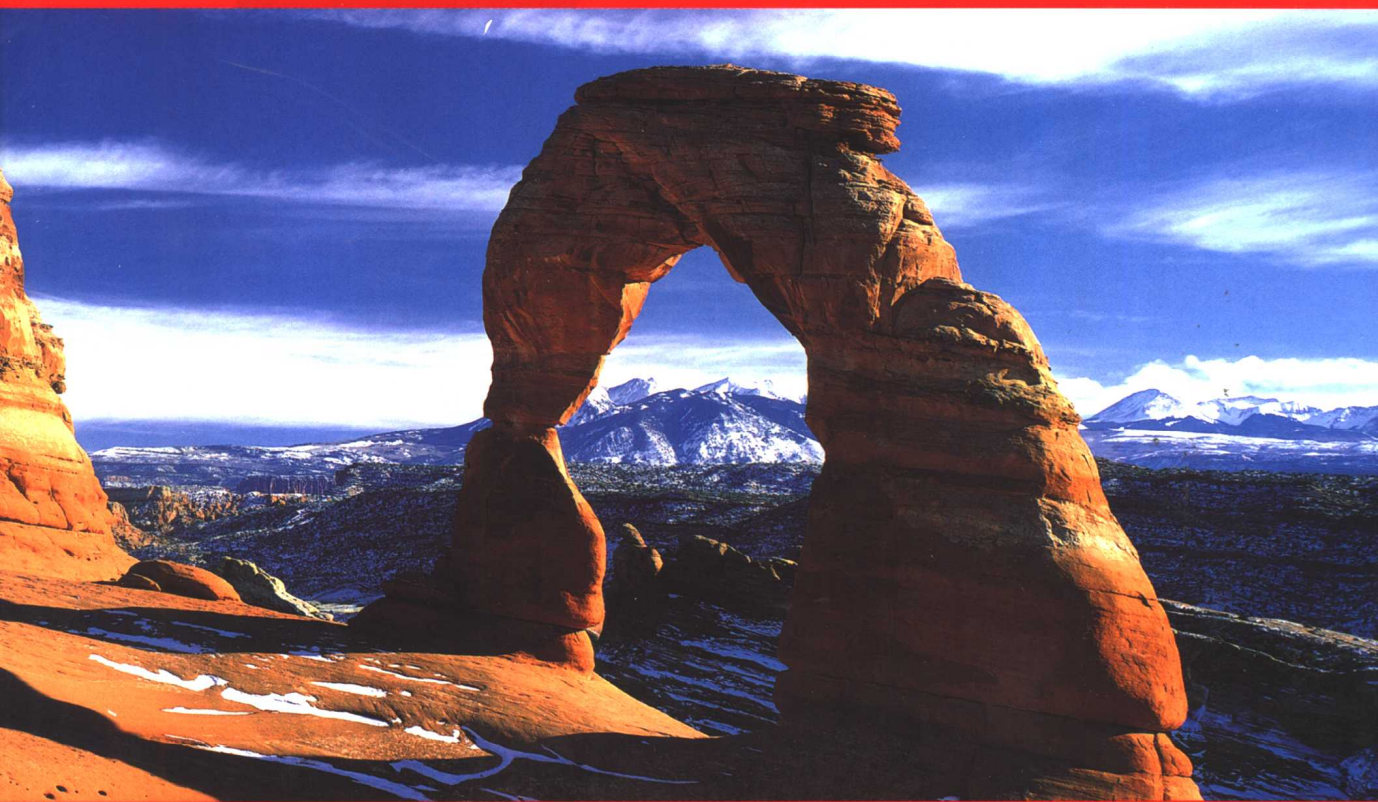
深入 C++ 系列

C++ Strategies and Tactics

C++ 编程惯用法

高级程序员常用方法和技巧

[美] Robert B. Murray 著
王 昕 译



深入 C++ 系列

C++ Strategies and Tactics

C++ 编程惯用法 —— **高级程序员常用方法和技巧**

[美] Robert B. Murray 著
王 昕 译



中国电力出版社

C++ Strategies and Tactics (ISBN 0-201-56382-7)

Robert B. Murray

Authorized translation from the English language edition, entitled C++ Strategies and Tactics,
published by Addison Wesley, Copyright©1993

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 the
Publisher.

CHINESE SIMPLIFIED language edition published by China Electric Power Press Copyright©2003

本书由美国培生集团授权出版。

北京市版权局著作权合同登记号 图字：01-2002-4846 号

图书在版编目（CIP）数据

C++编程惯用法——高级程序员常用方法和技巧 / （美）默里著；王昕译。

—北京：中国电力出版社，2003

ISBN 7-5083-1912-5

I.C... II.①默...②王... III.C 语言—程序设计 IV.TP312

中国版本图书馆 CIP 数据核字（2003）第 111163 号

丛 书 名：深入C++系列

书 名：C++编程惯用法——高级程序员常用方法和技巧

编 著：（美）Robert B. Murray

翻 译：王昕

责任编辑：陈维宁

出版发行：中国电力出版社

地址：北京市三里河路6号

邮政编码：100044

电话：（010）88515918

传 真：（010）88518169

印 刷：汇鑫印务有限公司

开 本：787×1092 1/16

印 张：13.625

字 数：297 千字

书 号：ISBN 7-5083-1912-5

版 次：2004 年 2 月北京第 1 版

2004 年 2 月第 1 次印刷

定 价：26.00 元

版权所有 翻印必究

代译者序

“It is one of those books that every professional C++ programmer should have read as it is full of good advice based on a wealth of experience.” ——ACCU 主席 Francis Glassborow

是的，上面的评价就是 ACCU 主席对于你现在手头上的这本书的评价。现在市面上关于 C++ 的书籍种类之多，可谓浩如繁星，但能够得到 Francis Glassborow 亲自介绍并且给出 “**Highly Recommended**” 这一最高推荐度的 C++ 书籍，不过几本而已，而这本就是其中之一。

作为一种支持多种设计范型的通用程序设计语言，C++ 程序设计语言自身所特有的内涵以及外延都决非一两出色的书籍所能包含亦或一两年艰苦的学习所能掌握的。究其原因就是因为在使用 C++ 语言的过程中，你会拥有太多的选择。无论是在用 C++ 进行设计或者实现，做出正确的选择都决非易事。非常多的 C++ 书籍都只是详细地介绍了语言的特性，即使是某些口碑良好的读本，也同样只是在传教语言本身却忽视了告诉读者如何在使用 C++ 的过程中去做出正确的选择。而这本书却在这个关键方面有着出色的表现。从这一方面来说，这本书是为那些严肃的以及真正书写 C++ 代码的程序员准备的。

从本书中，你不会学习到什么是左值，什么是循环，什么是类。因为这本书理想的读者是那些已经具备了一定 C++ 基础，期望并且正在迈向 C++ 专家之路的程序员。尽管本书的作者在一开始就已经表明了没有人可以仅仅通过阅读此书而成为专家，但是以我对这本书的观点来看，作者 Rob Murray 无疑是在以自己数十年的专家经验为大家铺设一条通往专家的大道，这也正印证了那句经典——“师傅领进门，修行靠个人”。

那么到底该如何给这本书定位呢？一位使用了 C++ 数十年的专家写成的一本关于如何在 C++ 中做出正确选择的专家级读本，这就是我心目中给此书的定位。在购买此书之前，你应该至少阅读过一本关于 C++ 语言的书籍，但无需对 C++ 有很深刻的认识，因为这正是此书想要达到的目的。如果你曾经阅读过《C++ 沉思录》，那么这本书的程度和那一本在某种意义上是一致的，因为它们都不是关于语言介绍的，而都是关于如何使用 C++ 的。并且同样的在书中遍布了许多关于如何使用 C++ 进行更好编程的真知灼见。

在这里，我还给大家一点阅读方面的建议。本书本身是按照主题来组织的，也就是说一个章节一般都是独立地说明了 C++ 方面的某一个主题，并且给出了详细的解释以及说明。这样，阅读本书就并不需要循序而来，你可以一上来就选择你所感兴趣的章节进行阅读，不过我还是建议大家在阅读之前首先好好地熟读第一章。这一章是关于抽象（Abstraction）的，关于

抽象这一主题的说明并不十分常见于一般的编程书籍或者语言介绍书籍中。然而，抽象这个与编程有着本质联系的概念对于培养良好的程序观有着不可替代的作用。此书中关于抽象概念的说明虽然不是很深入，却非常直观和易于理解，我想在拥有了一个良好的抽象概念后，再去细细品味书中所提到的其他关于 C++ 的一切是有好处的。

最后，我还想单独地挑出书中的一章来谈谈我的看法。此书第 9 章标题非常醒目——重用 (Reusability)，软件工程中的一个核心而又敏感的词汇。在一本 C++ 书籍中，对如此主题进行论述是否合适？这也是我拿到书稿看过目录后的第一反应。因为重用这个话题决不是一两句话、一两个章节可以说清楚的，如此一来，如果作者没有很好的技术功底以及文字能力，这样的章节只可能沦落成“食之无味、弃之可惜”的鸡肋。在我个人阅读过的许多书籍中，这种现象常常见到，以至于现在我会把一本书是否包含了鸡肋章节作为评价一本书好坏的重要标准之一。然而，在详细地读过这一章节之后（这是我在阅读这本书的过程中看的第 2 章，第 1 章就是关于抽象的那章。是的，我个人是按照主题来阅读此书的），我想我的疑虑全部都被打消了。作者 Rob Murray 不愧是程序设计领域内的专家人物，可谓深谙重用之道，而其叙述手段更是精彩，首先从重用的目标以及发现说起，再一步步结合 C++ 语言的实现方式对多个主题进行了深入浅出的论述，其对软工学中重用的概念以及 C++ 的多个实现主题之间的结合可谓完美。在和此书译者王昕先生的私下交谈中，译者多次提到这一章也是让他本人最受益的，一个使用 C++ 已经有八年之久的程序员尚能从中获益，我相信一般读者在深刻理解了本章内容后一定会有同样的感觉。在我眼里这一章的内容就已经对得起读者对此书的所有投资了。

关于此书作者及译者，书底有很好的介绍，我也就不再生着墨了。书没有最好的，只有最适合自己的，在我说了这么多之后，这本书是否适合你，还是需要你自己做出判断，但再一次看看 Francis Glassborow 对此书的另外一条评价也许能让你更快并且正确地做出买与不买的选择，就像这本书将会告诉你更多关于 C++ 使用上的该与不该一样。

“Get a copy, read it and pass it on. Get your own copy if you want to keep ahead of your colleagues.” ——ACCU 主席 Francis Glassborow

倪 硕
于 长沙中南大学

序

在专家看来, C++可以帮助软件设计者和程序员构造出模块化、易维护以及高性能的系统。然而, 对新手来说, 这门语言的复杂程度是一个不小的威胁。在 C++中有着许多不同的语言特性, 只有具有一定经验之后, 我们才会知道各种特性的适用情况。

本书的目的就是加速读者的这种学习进程。大部分成功的 C++程序员并不会简单地从语言规则中复述出某些章节或是条款, 相反, 他们总是习惯于自己以往工作中所获得的行之有效的那些惯用法和技巧。我们的目的就是帮助那些 C++的新手学习到那些在实践中最有效的惯用法。同时, 我们也会在书中指出那些我们经常接触到的 C++缺陷。

在本书中, 我们的讨论并不试图覆盖整个语言, 那些有关语言语义的精确定义可以查看参考手册。我们主要帮助读者构造出可以被那些不是 C++语言专家的人所理解的程序。我们不但讨论了那些使程序变得优雅及高效的技巧, 也展示了如何使它们更容易被理解和维护的方法。

致谢

本书中的想法和编程惯用法几乎都不是我发明的。我的目的就是把它们(在我 8 年的 C++使用过程中从其他人那学到的并被我认为是最重要的策略和战术)用一种 C++新手也能学会的方式展现给读者。这些收获中的部分来自于实际的开发项目中(这些项目被我们从 C 移植到 C++上)的经验, 另外一些则来自于和其他高手的讨论。

关于模板以及函数库设计的许多好的想法(包括那些隐藏在本书所给出的容器类后的想法)都来自于最初由 Martin Carroll、Andrew Koenig 以及 Jonathan Shopiro 所设计的 USL 标准组件库¹中的类。对于在本书中出现的任何错误, 则都由我负责。Andrew Koenig 是一个 C++语言专家, 他对我的帮助非常大。另外, 那些参与过我所举行过的几期“C++策略与战术”研讨会的人也帮助启发了我书写本书, 并对书中的想法进行提炼。其他的重要的想法来自于 Tom Cargill、John Carolan、Jim Coplien、Mark Linton、Gerald Schwarz, 当然, 还有 Bjarne Stroustrup, 要不是他发明了 C++语言, 本书也就不可能出现了。

Brian Kernighan 阅读过本书的多次草稿, 他极好的反馈意见起了很大的帮助作用。同时我

¹ 译注: 关于 USL 标准组件的文章参见于 1993 年 6 月的《C++ Report》中 Martin Carroll 所写的《Design of the USL Standard Components》。

还要感谢 David Annatone、Steve Buroff、Tom Cargill、Bill Hopkins、Cay Horstman、Lorraine Juhl、Peter Juhl、Stan Lippman、Dennis Mancl、Scott Meyers、Barbara Moo、Lorraine Weisbrot Murray、Bjarne Stroustrup、Clovis Tondo、Steve Vinoski 以及 Christopher Van Wyk，他们对本书早期的草稿提出了意见。此外，Lorraine Weisbrot Murray²对我的鼓励、理解、支持以及爱使得这一切努力都成为了可行的。

2 译注：Lorraine Weisbrot Murray 是 Robert B. Murray 的妻子（经 Robert B. Murray 证实）。

目 录

代译者序

序

第 0 章 简介	1
0.1 关于示例代码	1
0.2 本书的组织	2
0.3 问题	2
0.4 语言规则	3
第 1 章 抽象	4
1.1 有关电话号码的抽象模型	6
1.2 抽象模型间的关系	7
1.3 请考虑边界条件	12
1.4 使用 CRC 卡片来辅助设计	13
1.5 小结	13
1.6 问题	14
第 2 章 类	15
2.1 构造函数	15
2.2 赋值	23
2.3 公用数据	24
2.4 隐式类型转换	28
2.5 操作符重载：成员或非成员？	33
2.6 重载、缺省值以及省略符	36
2.7 Const	37
2.8 返回值为引用	43
2.9 静态对象的构造	44
2.10 小结	45
2.11 问题	46
第 3 章 句柄	48
3.1 一个 String 类	48
3.2 使用计数器来避免多份拷贝	50

3.3	避免进行重编译: Cheshire Cat	55
3.4	使用句柄来隐藏设计	56
3.5	多种实现	57
3.6	作为对象的句柄	61
3.7	综述	61
3.8	小结	61
3.9	问题	61
第 4 章	继承	63
4.1	is-a 关系	63
4.2	公有继承	68
4.3	私有继承	68
4.4	保护型继承	70
4.5	与基类抽象的一致性	71
4.6	纯虚函数	73
4.7	有关继承的细节和陷阱	75
4.8	小结	78
4.9	问题	79
第 5 章	多重继承	80
5.1	作为交集的多重继承	80
5.2	虚基类	85
5.3	一些有关多重继承的细节问题	89
5.4	小结	91
5.5	问题	91
第 6 章	考虑继承的设计	93
6.1	被保护的接口	93
6.2	我们的设计是否应该考虑到继承?	96
6.3	一些为继承所做的设计的例子	100
6.4	结论	105
6.5	小结	105
6.6	问题	106
第 7 章	模板	107
7.1	模板类 Pair	109
7.2	一些有关模板的细节	110
7.3	模板的实例化	111

7.4	智能指针.....	113
7.5	作为模板参数的表达式.....	119
7.6	模板函数.....	121
7.7	小结.....	124
7.8	问题.....	124
第 8 章	模板的高级用法	126
8.1	使用了模板的容器类.....	126
8.2	示例: Block	127
8.3	有关 Block 的设计细节	130
8.4	带有迭代器的容器: List	134
8.5	迭代器的设计细节.....	141
8.6	性能上的考虑.....	144
8.7	对模板参数的限制.....	146
8.8	模板特化.....	148
8.9	小结.....	154
8.10	问题.....	154
第 9 章	重用	156
9.1	发现和获得.....	157
9.2	健壮性.....	158
9.3	内存管理.....	163
9.4	可选的内存分配方案.....	165
9.5	传递参数给 operator new	168
9.6	管理外部资源.....	170
9.7	寻找有关内存的 bug.....	171
9.8	名字冲突.....	175
9.9	性能.....	178
9.10	不要去猜想, 而应该度量!	179
9.11	算法.....	179
9.12	动态内存分配中的瓶颈.....	180
9.13	内嵌.....	185
9.14	Tiemann 法则.....	187
9.15	小结.....	187
9.16	问题.....	188
第 10 章	异常	190
10.1	一个负面的声明.....	190

10.2	为什么需要异常?	192
10.3	一个异常的例子	192
10.4	异常只应该用来表述异常情况	194
10.5	理解异常	195
10.6	责任评估	196
10.7	设计异常对象	197
10.8	小结	200
10.9	问题	200
第 11 章	向 C++移植	202
11.1	选择使用 C++	202
11.2	采用 C++	203
11.3	设计和实现	204
11.4	开发一个坚实的基础	206
11.5	相关的思考	207
11.6	小结	207
11.7	问题	208

第 0 章

简介

C++是一门规模庞大的编程语言。只了解 C++规则的程序员就和只知道棋子如何移动的棋手一样（失败）。为了获得成功，还要学习一些相关的法则和策略。

本书所面对的读者是使用 C++语言的初级及中级程序员们，他们一方面期待学到更多有关使用这门语言的知识，一方面又没有时间去参加一个大学学期那么长的一门课程。我们假设读者已经了解了一些 C++的基础知识——如：什么是成员函数？如何使用 `public`、`private` 以及 `protected` 等等。本书的读者没有必要是一个 C++专家，在每次提及一些该语言中比较高级的话题（如多继承）时，我们都会给出一个“回顾”栏目对此进行一个简短的总结。我们关注的是那些被证明在实际中十分有效的策略。在本书中，我们将避免多谈理论，而尽量为读者给出例子及实用的建议。

我们尽量保证不偏离惯例。在本书中，没有一个想法或技术会超出语言的基础，它们大多来自于有着多年 C++经验的程序员日常的实践中。很显然，最新的想法总是具有最大的风险。我们希望能够避免将 C++的初级用户引导到那些想法上去。除了第 10 章之外¹，本书中所有的技术都经过了实践的证明。

我们也无意将读者变成 C++专家。C++中也包含了一些阴暗的角落，我们会在书中将它们一一指出并教导读者如何避免它们。写出依赖于语言中定义含糊、意义微妙的规则的程序是不好的，即便作者本人清楚它的意义并保证它可以正确运行，下一个来维护这段代码的人也可能未必能够做到这点。比较好的做法是：坚持使用语言中那些被广泛使用及理解的部分来书写程序。

0.1 关于示例代码

本书中的示例代码都通过了编译（或是取自一个通过编译的文件）。为了使代码尽可能短小，我们在此略掉了那些和主题无关的周边代码（如 `#include` 指令）。

本书中的示例大多只处理接口，而不是实现。因此，为了清晰起见，大部分的示例代码都略去了类的私有部分：

¹ 译注：在本书写作过程中，异常还只是 C++中的一个新增特性，但到了现在，在 C++中异常已经发展得比较成熟了。

```
class Something {  
    // 私有部分被省略……  
public:  
    Something();  
};
```

0.2 本书的组织

在第1章中，我们讨论了如何选择一种合适的高阶抽象方式来实现设计，并着重阐述了抽象和其实现之间的区别。

后续的两章主要关注如何在 C++ 的类中实现较高层次的抽象。第2章的讨论涵盖了自高阶的设计策略到底层的接口和实现细节。在第3章中，我们阐述了几种句柄在类实现中的用法。

接下来的三章介绍了继承。第4章中讨论了公用、保护、私用继承的区别，并给出了如何选择使用各种继承方式的参考意见。第5章中的讨论主要针对超过基类不止一个的继承，我们讨论了应该如何使用它以及有关多继承的一些难以掌握的细节之处。在第6章中，我们讲述了一些有关如何构建类的方法，以使得其他人可以把这些类作为他们的基类来发展他们自己的继承体系。

模板，作为 C++ 语言的一个新特性，其功能非常强大，它必将对人们以后编写 C++ 程序造成极大的影响。我们在第7章和第8章中详细地讲解了模板。由于大部分 C++ 用户对于模板还不甚了解，我们开始只是以基础为始，先描述一些用来构建有效模板的技巧，接着才构建各种各样的模板。包括：使用模板来实现智能指针，一些简单的容器，最后是一个接近实际应用的容器（一个 List 模板）。然后我们才讨论一些关于模板的高级技巧，如：如何对那个 List 模板进行重新组织，使其更快、更小、并且具有更多功能。

第9章中讨论了如何把一段可工作代码加入到重用库中：重用的困难程度要远远超过其字面意思。我们在本章中也展示了如何使用前几章中介绍的技巧来提高代码的健壮性、使其更容易使用以及运行得更快。

C++ 中新的语言特性——异常——可以使我们在编写程序时不再需要经常去进行出错情况的检测，但它也很容易被人们给误用。在第10章中，我们讲述了如何使用异常以及不适合使用它的情况。工业界目前也正在学习如何使用异常，这也使这一章显得稍微有点投机。

将一个项目从 C 平台移植到 C++ 平台所牵涉到的不止是更改一个编译器那么简单：我们必须项目开发中习惯一种新的开发软件的方式。我们将会在第11章中简单讲述一些有关采用 C++ 开发的项目中经常碰到的技术及人们自身上的问题。

0.3 问题

除本章外，本书中的其他章节都以一个“小结”加一个“问题”作为结束。在“小结”中，

我们将会以列表的方式给出每一章中的主要概念。那些问题是用来促使读者去思考以及讨论有关技术和人们自身方面的问题，以使得他们可以更容易地理解和记忆每一章中给出的内容。问题中的一部分具有“标准”答案，但更多的则没有，我们衷心地希望读者就此进行的讨论会对他们产生帮助。

0.4 语言规则

在本书书写过程中，C++语言的当前“官方”标准文档是 Ellis 和 Stroustrup 所编写的《Annotated C++ Reference Manual》（也经常被简称为 ARM）²。ARM 是 ISO/ANSI C++标准委员会（一个由来自工业界以及学术界的自愿者所组成的团体，他们正在制订一个就会被 ANSI 和 ISO 同时采纳的 C++语言标准）用以制订标准文档的基础。除非 ISO/ANSI 委员会明确提出了某些不同之处，否则在本书中所提到的“语言规则”都是基于 ARM 之上的。实际上，（ISO/ANSI 委员会与 ARM 之间的）大部分的区别都微不足道。我们也鼓励所有的 C++用户都能够遵循标准化结果。

² 译注：目前 C++的标准文档是 1998 年 7 月所发布的 ISO 14882 文档（也称 C++ 98）。

第 1 章

抽 象

数据抽象（data abstraction）是面向对象设计的一个重要概念。数据抽象要优先于面向对象的设计；然而，随着 C++ 这样直接支持数据抽象的语言变得流行起来，它的应用范围也变得越来越广泛。

抽象数据类型（abstract data type，也称为 ADT）是一种由用户定义、拥有明显不同的两部分的类型：

一个公用的接口（public interface），用于指定用户使用该类型的方式，而一个私用的实现（private implementation），在类型内部被使用，以提供公用接口所给定的功能。

在 C++ 中，`private` 和 `public` 关键字被用于指定类声明中哪部分是实现，哪部分又是接口。通过这种方法，编译器就可以确保类的使用者不会绕过类的接口而直接访问其私用成员。然而，另一个事实就是，类有私用成员也不表示它的设计就很好。

对于 `private` 和 `public` 来说，它们本身还存在着一个远比低层次语言规则更重要的思想。这个思想还从未被 C++ 编译器显式地检测过，但它对于设计模块化、易维护的程序来说却是至关重要。它表达的是一种概念，那就是——每个编程问题的解决方案都可以被划分为两部分：一个抽象模型（abstract model），它是在程序员和程序用户之间都能取得一致意见的、用于描述该问题以及其解决方案的智力模型；对于该模型的实现（implementation），就是程序员使用特定的方法使计算机可以表述出这种抽象。在本章中，我们将会通过对一个类的设计、细化以及记录过程，来逐步体现出抽象的过程。如我们将看到的那样，“简单地书写一页手册，然后让代码来适合它”这种方式在程序设计中是行不通的。

下面是一些理由，用于说明为什么一个经过详细考虑并记录下来的抽象模型对于程序设计来说很重要：

它有助于帮助其他人来理解如何使用你所设计的类。如果你正试图使用一个链表类，你的主要的（也是最初的）考虑应该不是包含这个类的头文件的名称，也不是该类中成员函数的名称以及其参数类型等。相对来说，你会更注重那些有关该类抽象模型的基础问题，如：它提供了什么操作？能不能在链表上进行回退？能不能在常数时间内访问到链表的头和尾？同一对象能不能在一个链表中出现多次？链表中包含的是对象本身呢，还是对对象的引用？如果链表包含的是对象的引用，那么由谁来负责创建及销毁这些对象？当链表中的某个对象被销毁

时，链表应该有什么样的调整？等等。

对这些问题的回答将会直接影响到那些使用该链表类的应用程序的设计。（例如，如果你的应用程序需要能够在链表上双向移动，那么一个单向链表类在此处就不再适合了。）

如果你不能理解并记录下你的类所支持的抽象模型，那么用户就可能会选择不使用你所提供的类（更糟的是，用户决定使用你的类，然后却发现他们做了一个错误的选择）。

抽象模型是你与你的用户之间的一个协议。由于抽象模型对用户的程序设计有着很重要的影响，你将发现对它进行不向上兼容的改动会有多困难，有时那样的改动甚至就是不可能做到的。例如，一个去除掉链表中后退功能的决议，对公开使用的 List 类来说将会是一场灾难。为什么这样说呢？因为某些 List 的用户可能会将其应用程序的设计完全围绕着它能够支持后退这种抽象来进行。这种在抽象模型中的不兼容改动肯定会使某些已有的用户陷入困境。如果一个类已经被广泛使用，那么它的每一个特性都将至少被一个用户所使用，如果你轻率地决定去除掉某个特性，必将在特定的用户群中引起骚乱。

这就意味着，在你决定将你的库发布给客户时，对库的抽象模型进行细化和完善尤为重要。在实现中产生的错误通常都可以很容易地在下一个版本中得到更正；但在抽象模型中产生的问题（除去冗余错误之外）则将持续存在于类的整个生命周期中。

在记录抽象模型的过程中，我们常常可以发现设计中的重要缺陷。在软件项目设计的早期，我们通常会过高地估计我们对于问题以及我们所提出的解决方案的理解程度。在将早期的模糊想法用精确的言语表示清楚的过程（即抽象模型的记录过程）中，我们就能更关注于那些我们以前从未考虑到的方面。毕竟，在认识不清晰的时候，我们是无法提出合理的解决方案的。

清晰的抽象模型文档有助于其他人重新构造出你的类的新版本（这包括继承或是重新实现两种方式）。要想实现出一个能够与现有代码共同工作的新版本，你所需做的不止是在成员函数的名字以及类型签名上做到和最初的代码完全匹配；更重要的是，新版本的类必须还能够符合旧版本的抽象模型的要求。

一旦理解了抽象模型，我们就可以避免“以实现来驱动设计”的情况出现。不管其承认与否，许多的软件设计者在设计一个新类的接口时，他们的脑海中都已经有了一个“明显”的实现方案。这就会使得他们在设计中不自觉地抽象模型向这种实现方式靠拢。这种做法不但不能提供一个用户易于理解的、并且认为就应该如此的接口，它还会使实现的细节遍布于接口之中，导致以后对实现进行改动变得异常困难。

当然，其他极端的做法也同样会导致麻烦的出现（如：设计出一个完全不顾及实现方案的可能性的接口）。一个接口，不管它有多么优雅，如果它不可能被实现出来，或是实现它需要一些让人无法接受的性能上的损失，那么对用户来说它就起不到任何帮助作用。在这些考量之间取得一个合理的平衡是类设计中最具有挑战性的部分。最后说明一点，设计抽象模型和设计实现方案应该是两个独立的行为。但尽管如此，这并不意味着我们需要用不同的人手来分开处理这两部分，重要的是，开发人员必须知道自己在某个特定的时间时，自己到底是在负责抽象呢，还是负责实现？

仅仅考虑抽象（而不是实现），我们有几种常用的方法。在抽象的过程中决定“什么应该有，什么不应该有”是面向对象设计者的一个关键技巧。在下一小节中，我们将开始构造出一个简单的抽象模型。

1.1 有关电话号码的抽象模型

在本节中，我们将对用于表示电话号码的类的抽象模型进行最初的探讨。我们期望用这个类来代指各种电话应用程序（如交换机系统、账单系统等等）中的电话号码。

我们应该怎么开始我们的第一步呢？一个不错的方法就是：用一句话来描述该对象是用来干什么的（注意，不是“是什么”）。这种描述方式应该尽可能的抽象，尽量不要涉及到对象的内部结构。这种“官面总结（executive summary）”应该尽可能的简单，简单到即便是行政人员也可以轻而易举地理解它。

例如，下面就是对于电话号码的一个糟糕的官面总结：

“电话号码包括3位地区号码，后面紧接着3位交换机号码，最后是4位数字。”

这样的描述并没有提及电话号码是用来做什么，或者它和谁交互的问题。相反的是，它简单地把电话号码的结构描述了一遍。这不但没有解释清楚电话号码是做什么的，它还限制了电话号码的范围。例如，通过 modem 所拨打的电话号码可以含有非数字的字符（如&就用于告诉 modem 来等待第二次的拨号音，另外还有#和*这两个特殊的按键）。更严重的就是，它还限制了在美国和加拿大之间能够合法通话的电话号码集。

下面的修改就要稍微好一些：

“一个电话号码指定了世界上的某一部特定电话。”

此处我们并没有丝毫提及电话号码的结构，它只是阐述了另一个用于电话号码的抽象。

当你写下这样的句子后，请花一些时间来仔细琢磨它。请像律师对待一份合法的合同那样仔细检查其中的每个单词。这句话中有什么明显的漏洞吗？它所隐含表达的意思又是什么？

让我们先来注意“指定”这个词。它意味着什么呢？一个电话号码是不是惟一指定一部电话呢？换句话说，能不能有两部以上的电话共用同一个号码呢？事实上，在许多的商业机构中，大部分电话都是共用一个电话号码的：在有电话进来时，会有一个本地的交换机将它转到一部空闲的电话上面去。同样，对于一个特定的电话来说，电话号码也不是惟一的。当你拨打555-1234时，对方的电话就取决于你拨打时所处的区域。

这样，我们就得知了：一个电话号码并不代表惟一一部电话，它的含义取决于使用它的区域。为了反映出这种更新，我们进行了如下的官面总结：

“当与呼叫电话关连起来时，电话号码就是决定被呼电话的关键。”

这样，我们就把电话号码与呼出电话的关系给涵盖进去了，并且也隐式地表达了被呼电话