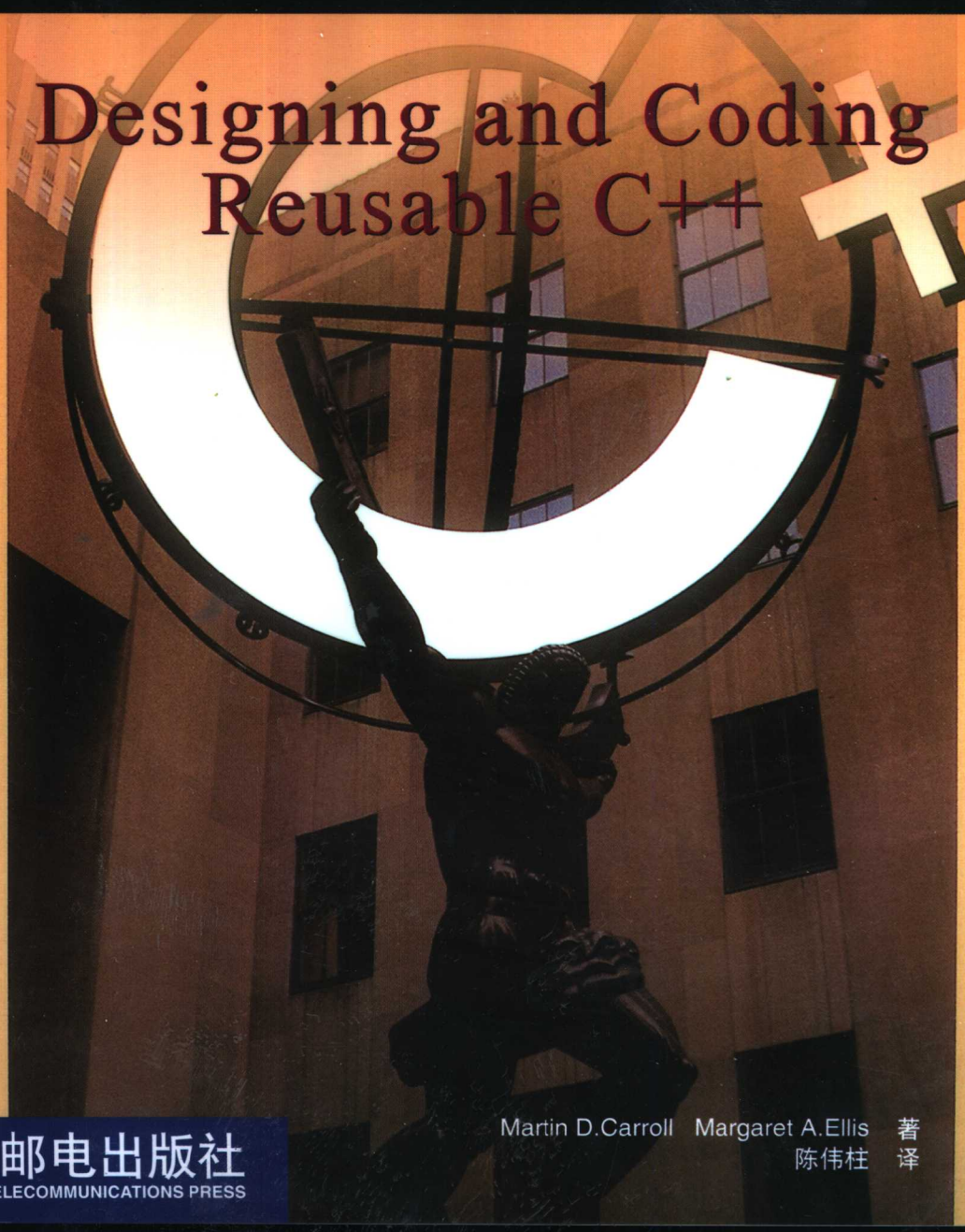
C 和 C++ 实务精选

Addison Wesley

# C++代码设计与重用

## Designing and Coding Reusable C++



Martin D.Carroll Margaret A.Ellis 著
陈伟柱 译

人民邮电出版社
POSTS & TELECOMMUNICATIONS PRESS

C和C++实务精选

# C++代码设计与重用

Martin D.Carroll
Margaret A.Ellis 著
陈伟柱 译

人民邮电出版社

C 和 C++实务精选

**C++ 代码设计与重用**

◆ 著　　Martin D.Carroll　Margaret A.Ellis
译　　陈伟柱
责任编辑　陈冀康

◆ 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
读者热线　010-67132705
北京汉魂图文设计有限公司制作
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：18.25
字数：400 千字　　2002 年 11 月第 1 版
印数：1-4 000 册　　2002 年 11 月北京第 1 次印刷

著作权合同登记　图字：01 - 2002 - 1550 号
ISBN 7-115-10624-X/TP • 3081

定价：38.00 元

# 内容提要

本书全面展示如何使用C++编写可重用的代码，从而提高程序员的开发效率。

全书分为12章。包括重用性基本概念、类设计、扩展性、效率、错误、冲突、兼容性、继承、移植性、程序库等和重用相关的诸多话题。每一章的最后，通过总结和练习帮助你巩固概念、加深理解，参考文献和相关资料为你指明了深入学习的方向。

本书适合有一定C++经验的程序员阅读，也可供以提高代码重用性为专门学习方向的读者参考。

# 作者简介

Martin Carroll 是 AT&T 贝尔实验室的技术人员，他曾经用好几年的时间致力于设计和实现可重用的 C++程序库，包括 AT&T 标准组件库（Standard Components Library）。他在 Rutgers 大学获得计算机科学博士学位。

Margaret Ellis 是 *The Annotated C++ Reference Manual* 的合著者（另一个作者是大名鼎鼎的 C++之父 Bjarne Stroustrup），她主要致力于 AT&T 贝尔实验室、UNIX 系统实验室和美国 Novell 公司的编译器开发。她曾获得加州大学计算机专业的硕士学位。

# 中文版序

最早知道这本书，是在 Scott Meyers 著名的 *More Effective C++* 书籍推荐列表里。在谈到这本书的时候，Meyers 说："有意撰写程序库的人，若没有读过此书，那只能是匹夫之勇。"而 Francis Glassborow 则说，大多数程序库设计者应该搬过小板凳来，像小学生那样学习这本书。

用 C++的人，历来以能撰写出重用性好的程序库而引以为傲。然而，撰写可用的程序库已属不易，撰写可重用的程序库更是专家级任务，以至于有人感叹道，重用是 C++程序员们的"圣杯"——总在嚷嚷着，却始终得不到。其实何止 C++如此，在其他语言中，重用又何尝不是令人胆怯的挑战！只不过 C++编译语言的本质与人们对 C++程序性能的严苛要求，使得用 C++编写可重用组件更为困难。这样一个艰难的主题，敢于涉足已经是非常有勇气了，而能得到 Meyers 等人如此高的评价，则非 C++中的顶尖人物倾力而为不可。事实上，作者 Martin Carroll 和 Margaret Ellis 正是这样的顶尖人物。他们是 AT&T 实验室的研究人员，从 20 世纪 80 年代中期开始就处于 C++演化和发展的中心，长期从事编译器和基础库的开发工作，积累了普通程序员无法比拟的经验。特别值得一提的是，Margaret Ellis 曾与 Bjarne Stroustrup 合作撰写了 C++早期的经典著作 ARM（*The Annotated C++ Reference Manual*），是 C++社群里最受尊敬的巾帼英雄之一。在这本书中，两位作者把自己多年来撰写可重用库的经验和教训高浓度地总结起来，给出了很多具有深刻洞察力的建议。虽然从此书英文版出版至今已有几年的时间，但是书中的真知灼见仍然值得我们一再学习体会。我甚至认为，即使你不打算撰写程序库，甚至不是 C++程序员，认真阅读这本书都是很有好处的。毕竟这本书来自具有丰富实战经验的顶尖专家之手，而这样的书是在是太少了。

译者陈伟柱是我的好朋友，在 C++、Java、模式和重用等方面都有比较深刻的理解。他对待此书翻译工作的态度极为认真，正是这一点令我对这本书的翻译质量很有信心。原书作者是以技术扎实而谦恭简和著称的，他们的文字也是平实简练而深富内涵的，我认为伟柱的为人和他的文字也具有这样的特点。因此，我衷心希望读者也能以平实谦和的心态认真、扎实地阅读和学习此书，并祝愿大家能有满意的收获。

孟岩

2002 年 10 月于北京

# 译者的话

这是一本令我受益匪浅的书，我相信很多读者也会有相同的感受。

重用——可以说是一个永恒的话题，只要有软件工程的继续存在，重用就必定会有一席之地，也会成为软件工程师或者程序员们关注的话题。重用的历史和重要性毋庸多言，几乎每本软件工程的书都会叙述重用的概念和大体内容；但是，现今关于重用的中文书，大多对重用的理论和优点介绍得非常详细，而很少涉及到如何设计与编写可重用代码的内容，给人一种隔靴搔痒、不着痛处的感觉。所以说，对于这本书，看完之后，我想说的一句话就是："它终于来了。

重用有3个基本的问题，一是必须有可以重用的对象，二是所重用的对象必须是有用的，三是重用者需要知道如何去使用被重用的对象。解决好这3个方面的问题才能实现真正成功的软件重用。本书的大体内容就是依据前面这两个问题，从类的层次体系设计、扩展性、效率、兼容性、移植性等方面详细叙述了可重用代码的设计原则，深入浅出地讨论了当重用性和上述的诸多特性发生冲突的时候，应该如何取舍，从而达到程序员或者程序库设计者预期的目的；而且，本书的第11章还针对上面的第3个问题给出了编写可重用代码文档的要求，为成功的重用代码设计加上了不可或缺的一环。书中的具体内容第1章已经详细给出，这里我就不再赘述。

本书的写作风格很有特色，它并不会教你要如何做，也不会给你叙述哪种设计与编码才是最佳的途径——金无足赤，永远为最佳的选择本来就是子虚乌有。作者凭借多年的编译器开发和可重用代码设计的经验，详细地阐述了在编写可重用代码的过程中一定会碰到的许多抉择，在阐述中还重点指出了许多会被普通开发者忽略的考虑因素；然后在不同的上下文中详尽地比较各种设计选择的优劣，让读者知其然更知其所以然，给致力于设计和编码的程序员点亮了一盏通向成功重用的明灯。

对于此书，我不敢说它弥补了国内重用书籍的空缺，但就程序库设计方面而言，它确实有着不可替代的作用，衷心希望每个致力于程序库设计的读者，都可以从此书获益。

翻译过程中，译者尽量以平实无华的语言来对待这本技术书籍；在力求技术准确的同时，更加注重上下文的连贯性，希望我的努力能给你带来一个顺畅的阅读过程。

译者

2002年10月

# 前言

*一切事物都将得到检验并因此被称为问题。*
*——Edith Hamilton*

这本书的主要目的在于：展示如何以C++编程语言编写可重用代码——就是说，根据不同的需要，在不经过修改，或者经过很少修改的前提下，可重用代码可以很容易地应用到5个、50个甚至500个程序当中，而且这些程序往往是不同程序员编写的，可能运行在不同的系统上。在整个阐述的过程中，我们的目的并不在于争论是否所有的代码都是可重用的，也不在于说明可重用代码能够解决所有的程序问题。显然，不论是对程序员而言，还是对可重用代码本身而言，提高代码的重用性都是需要代价的；通常只有当我们有理由相信所给代码在将来有可能会被重用时，我们才会付出这些重用的代价。因此，本书的目的在于详细分析重用性的这些代价，于是当你面对是否编写可重用代码的选择时，可以从容地做出明智的决定。

## 关于本书

本书主要面向的读者是：那些希望从书中包含的许多深层C++编程见解中受益的读者，或者是那些需要或希望学习如何编写可重用代码的读者。在论述过程中，我们假设读者已经知道如何编写正确的C++代码。

C++语言至今还没有经过标准化（译注：本书写作于1995年，C++于1997年标准化），任意两个不同的编译器实现支持的语言几乎都是不同（稍微不同或者相差很大）的。当我编写这本书的时候，并没有一个编译器实现可以完全支持ANSI/ISO C++标准中最终定义的整

个语言特性；而且，就算对同类型的编译器而言，前后版本实现的语言特性也不尽相同。于是，编写一本对所有编译器都适用的书是很困难的，或者是不可能的。因此，当我们讨论或使用一些不能被主流 C++编译器所实现的特性时，我们将会另加说明。

当声明本书中的代码例子被认为合法时，我们所指的合法性是以 1994 年 9 月份 ANSI/ISO C++的工作文件（有时候也称为“标准草案”）[ANS94]为依据的。而且，在我们的代码例子里，我们将尽量避免使用那些我们认为在最终 ANSI/ISO C++标准公布之前，很有可能会被删除或者进行重大修改的语言特性。

要跟踪 C++语言的演化，有几个资源是可以利用的。Internet 讨论组 comp.lang.c 和 comp.std.c++主要致力于 C++的讨论。互联网中还有一个关于 C++的主页：

```
http://info.desy.de/user/projects/C++.html
```

而由纽约 SIGS 出版社定期发行的 *C++ Report*，也经常会刊登许多 C++程序员感兴趣的文章。至于学习 C++的书籍，Lippman 的[Lip91]和 Stroustrup 的[Str91]是最佳的学习教材。

任何关于重用性的完整讨论都会涉及到软件开发的一些其他主题，如接口的设计、实现的效率、可移植性、冲突等等。在书中的论述中，我们假设读者对软件开发已经有所了解；因此，我们只注重于那些应用于可重用代码的主题，而不是那些应用于一般软件开发的主题。

“每本教材都会撒谎”，这是我们老师非常喜欢的一句谚语。在这里，她实际要表达的意思是：每本教材为了教学的简化性和明确性，都必须简化书中的内容，从而都会歪曲所要表达的信息。显然，即使在需要简化的前提下，我们还是应该用抓住众多细节间的本质信息的方法来表述每个主题，这本书也不例外。于是，对于某些主题，即使有些读者已经对这个主题的方方面面都有所了解，我们还是希望他们能够和我们一起共同探讨该主题的内容。而且，对于一些主题内容的省略，我们也希望能够得到这些读者的谅解。

## 选择与建议

设计和编写可重用代码将会涉及到许多选择（即在多种实现方案中选定某种方案），而某些选择的决定又是左右为难的。通常，对于这些选择，并没有完全确定或者正确的答案——无论你做出哪个选择，每个选择都是有代价的。我们将讨论各种不同选择方案的优点和缺点，从而让将来可重用代码的编写者可以从容地做出明智的决定。

有时，对于某个给定的决定，我们会认为某种风格或者方法要优于其他所有的风格或者方法。于是，当遇到这种情况时，我们会明确地建议采用这种风格或者方法，并给出支持这种风格或者方法的详细理由。然而，在大多数情况下，只有 C++程序库的设计者，才能决定哪种方法对程序库用户而言是最好的。

这本书的一个目的在于：详细并且清楚地给出编写可重用代码的大多数重要选择。如果我们忽略了某些重要的选择，那么我们愿闻其详。

书中大多数关于选择的讨论都会提到效率。虽然我们并不认为效率是所有可重用 C++代

码最重要的设计目标，因为对于许多程序库设计而言，譬如扩展性、灵活性等其他特性将会比效率更加重要；但我们在文章中又处处提到效率，这是因为效率将会和可重用 C++代码的其他每个可取的特性都相互制约。既然对效率已经花费了这么多笔墨，我们也希望可以给出所有和效率相互制约的因素。

## 代码例子

我们希望这本书的价值在于：与刚开始读这本书之前相比，当你读完这本书的时候，你将知道更多关于如何编写可重用代码的知识。因此，我们使用了大量的例子，而且许多例子的代码是来自于现实中已经存在的代码，因为，那些不是取材于真实代码的例子，不足以阐明实际编程中会出现的问题，也不足以说明实际编程中使用的各种技术。在这里，我们只是为了阐明重用性的目的而利用这些代码。

为了节省篇幅，成员函数的主体通常都是在类的声明中给出；即使在重用性的代码设计中，这些函数也不会被实现为内联函数。例如，读者不应该从下面的类定义语句中得出：函数 f 是内联函数（就是说实际上 f 不是内联函数）的结论。

```
class Z {
    void f() {
        //...
    }
};
```

（关于如何决定某个函数是否应该实现为内联函数，我们将在 4.4.1 小节讨论这个问题）

当提到模板成员函数的时候，在不致引起混淆的前提下，我们通常都会省略模板参数。例如，我们将把下面代码中的成员函数 f：

```
template<class T>
class X {
    void f();
};
```

表示为 X::f，而不是 X<T>::f。

通常，现实程序库例子中给出的类大多是模板，而在书中，为了能够使阅读更加容易，并且有利于更加清楚地表述每个主题，我们就进行了简化。相似地，现实程序库中的嵌套类也并不都如嵌套语法所定义的那样，但是，当我们在书中给出一个嵌套类的时候，我们将不使用前置声明语法。例如，我们将如下编写嵌套类：

```
class X {
    class Y {
        //...
    };
//...
};
```

而不是如下编写嵌套类：

```
class X {
    class Y;
    //...
};
class X::Y {
    //...
};
```

虽然前置声明语法能够提高可重用代码的可读性，但是许多C++编译器都不能实现这种语法。

现今，任何我们可用的编译器，都还没有实现新的强制转型（cast）语法。这个语法是一个很新的C++特性（针对于编写这本书的时候），我们也还不具备应用这个语法的充分经验，因此，我们在书中没有使用这个语法。

## 术语

在本文中，我们将使用程序库概念来表示可重用代码的集合。我们也经常互换地使用程序库和可重用代码这两个概念。另外，当我们说“编写”代码段的时候，我们的意思是“设计和实现”这段代码。

我们将使用编译程序来指代任何程序或者任何把源代码翻译成可执行代码的程序集合。翻译将包括：预处理C++使之成为另一种语言，然后把这种语言编译或者汇编成可执行代码。

为了行文简洁，我们在文章中使用的基类，不仅用来表述其他类的基类，而且还表述这个类可让其他类派生自该类。

C++操作符

```
ostream& operator<<(ostream& o, const T& t);
istream& operator>>(istream& i, T& t);
```

（对于某个给定的类型T）提供了T类型的流插入和流输出操作符。为了和通常的使用习惯保持一致，我们把它们分别叫做T的输入操作符和输出操作符。

关于模板和模板实例化的术语，C++社区仍然有许多不一致的意见。不同的人可能会使用术语模板类、类模板、实例化、实例、特化等中的一个，来描述不同（或者相似）的事物。这些术语准确而且标准的定义最后将由ANSI/ISO C++委员会来决定。下面说明了我们在此书中如何使用这些概念：

类模板：形如template<...>X的模板，其中X为类的定义。

函数模板：形如template<...>f的模板，其中f为函数的定义。

类模板特化：当类模板的参数被实际参数替代时，而得到的类。

函数模板特化：当函数模板的参数被实际参数替代时，而得到的函数。

实例化：产生特化的过程。

模板类：我们不使用这个术语

模板函数：我们不使用这个术语。

例如，考虑下面的类模板：

```
template<class T>
class X {
    //...
};
```

类 X<int>和类 X<char>就是这个模板的两个特化。另外，特化通常是隐式定义的，如：

```
template<class T> class X { /*...*/ };
X<int> x;        //X<int>就是隐式定义的。
```

偶尔，某些 C++程序员也会显式定义特化，例如：

```
template<class T> class X { /*...*/ };
class X<int> {/*...*/};      //X<int>在这里显式定义。
X<int> x;
```

而许多人把特化这个术语仅仅看成是显式特化，这是不正确的。虽然我们定义的这些术语都是非常相似的，但我们相信，ANSI/ISO 很快将会给出这些术语很好的定义。

通常，在不会引起混淆的前提下，我们会使用“类”来代替“类模板”，“函数”来代替“函数模板”。

# 致谢

在这本书的创作过程中，许多人通过多种方式支持过我们。我们在这里对他们表示衷心的感谢。

这本书的评审者所作出的贡献和该书的价值几乎是等同的。我们特别感谢那些对我们要求严格的评论者，他们总是相信我们可以做得更好。这其中包括 Tom Allocco、Manuel Bermudez、James Coggins、Keith Gorlen、Tony Hansen、Chris Hornick、Peter Juhl、Brian Kernighan、Andrew Koenig、Eason Kung、Rao Kuimala、Doug Lea、Stan lippman、Tom Lyon、Glen McCluskey、Barbara Moo、Rob Murray、Jishnu Mukerji、Scott Myers、Steve Pendergrast、Ed Schiebel、Jonathan Schilling、Jonathan Shopiro、Bjarne Stroustrup、Steve Vinoski、Jedy Ward 和 Clay Wilson。

我们也收到了许多读者很有用的反馈信息，他们是 Dag Bruck, Rich Kempinski, Josee Lajoie, Deborah McGuinness, David C.Oliver, Jeffrey Persch, Ellia Weixelbaum 和两个分别叫做 Lars 和 Steve 的网友。

这本书的许多观点首次来自于我们设计 C++标准组件库的实践，这个组件库最初是由贝尔实验室开发的。因此，我们要感谢我们的同事们关于这个项目的许多有深度的讨论。他们是 John Isner, Andrew Koenig, Dennis Mancl, Rob Murray, Jonathan Shopiro, Alex Stepanov, Terry Weitzen 和 Nancy Wilkinson。

如果没有 C++，那么这本书（也包括其他的许多书）肯定是不会存在的。因此我们非常

感谢 Bjarne Stroustrup 给我们带来了这样一个我们最喜欢的编程语言。另外也感谢 ANSI/ISO C++标准委员会的所有成员，正是因为他们孜孜不倦的工作（往往都是自愿而且无功利的），才能使整个 C++社区生机焕发。

感谢在我们这本书的编写过程中，我们这本书的审订者——Marha Currie, Barbara Moo 和 John Spicer——感谢他们的支持和鼓励。也感谢我们的雇主——贝尔实验室和 Unix 系统实验室（现在是 Novell Unix 系统小组）——感谢他们给了我们编写这本书的机会，而且还感谢他们为我们提供的时间、硬盘空间、打印纸等等。

我们非常感谢来自 Addison-Wesley 的大力支持。其中，充满活力的 Tom Stone 和生性乐观的 Debbie Lafferty 都是很好的合作伙伴。另外，Lyn Dupre 给了我们最好的编辑指导，书的设计者 Juliet Silveri 的工作总是一丝不苟，Pat Daly 是一个很专业的编辑，Roberta Clark 对我们的草稿进行了校对。我们还要感谢 John Wait，正是因为他一贯坚持让 Martin 编写一本关于 C++和重用的书，才会有这本书的诞生。然后，也很感谢 Jim DeWolf，他为我们提供了暖气和空调，这笔支付也是不菲的，从而给我们带来了一个很好的环境和开端。

衷心感谢 Tom Reinhardt，他可以说是世界上最好的外科临床医学家。也感谢我们的朋友 Paul Lustgarten，他是我们以前的房东，很慷慨地允许我们时时占用他的电话线。还感谢 David Wooley，正是他让 Martin 在大学中进入到 BASIC 编程领域中来的。真诚地感谢 Marybeth 为我们拍了书背面的那张照片。

最后，但也是很重要的，我们真诚感谢我们的家人和朋友，他们和我们度过了这本书编写过程中的许多酸甜苦辣的日子，并且时时刻刻给予我们支持与关怀。

**图书在版编目（CIP）数据**

C++代码设计与重用/（美）卡罗尔（Carroll,M.A.），（美）埃利斯（Ellis,M.A.）著；陈伟柱译. —北京：人民邮电出版社，2002.11
ISBN 7-115-10624-X

I. C… II. ①卡…②埃…③陈… III. C 语言—程序设计 IV. TP312

中国版本图书馆 CIP 数据核字（2002）第 076775 号

**版 权 声 明**

# 目录

**第 1 章　重用性介绍** ····· 1

1.1　什么是重用性 ····· 1
1.1.1　提取代码来作为重用 ····· 2
1.1.2　可重用代码的基本特性 ····· 2
1.2　重用的神话 ····· 3
1.3　重用的障碍 ····· 4
1.3.1　非技术障碍 ····· 4
1.3.2　技术障碍 ····· 5
1.4　希望是否尚存 ····· 6
1.5　这本书能给我们带来什么 ····· 7
1.6　练习 ····· 8
1.7　参考文献和相关资料 ····· 9

**第 2 章　类的设计** ····· 11

2.1　抽象性 ····· 11
2.2　正规函数 ····· 12
2.3　Nice 类 ····· 14
2.4　存在最小标准接口吗 ····· 15
2.4.1　缺省构造函数 ····· 16
2.4.2　赋值运算符 ····· 17
2.4.3　拷贝构造函数 ····· 18

2.4.4 相等运算符 …… 18
2.4.5 析构函数 …… 18
2.5 浅拷贝和深拷贝 …… 19
2.6 接口一致性 …… 22
2.7 转型 …… 25
2.7.1 多重所有权（Multiple Ownership） …… 26
2.7.2 敏感转型 …… 26
2.7.3 不敏感转型 …… 28
2.7.4 转型数目（Fanout） …… 28
2.8 const 关键字的使用 …… 29
2.8.1 抽象 const 对比位元 const …… 29
2.8.2 最大限度地使用 const …… 31
2.8.3 对 const 不安全的解释 …… 32
2.9 总结 …… 33
2.10 练习 …… 34
2.11 参考文献和相关资料 …… 37

**第 3 章 扩展性** …… 39

3.1 扩展性的权衡 …… 39
3.2 扩展性和继承 …… 40
3.2.1 只继承基类的接口 …… 41
3.2.2 只继承基类的实现 …… 42
3.2.3 同时继承基类的接口和实现 …… 43
3.3 继承语义（Semantic） …… 43
3.4 继承的障碍 …… 45
3.4.1 非虚成员函数 …… 45
3.4.2 过度保护 …… 47
3.4.3 模块化不足 …… 48
3.4.4 friend 关键字的使用 …… 51
3.4.5 成员变量过多 …… 51
3.4.6 非虚（Nonvirtual）派生 …… 52
3.4.7 妨碍继承的成员函数 …… 53
3.5 派生赋值问题 …… 55
3.6 允许入侵（用户修改源代码）继承 …… 57

3.7 总结……57
3.8 练习……58
3.9 参考文献和相关资料……60

**第4章 效率**……61

4.1 效率和重用性……61
4.2 程序创建时间……62
4.2.1 编译时间……62
4.2.2 实例化时间……64
4.3 代码大小……69
4.3.1 源文件分割……69
4.3.2 外联的（outlined）inline……71
4.3.3 模板特化大小……71
4.4 运行时间……72
4.4.1 内联（inlining）……72
4.4.2 虚函数……74
4.4.3 返回引用……76
4.5 空闲存储空间（free-store）和堆栈空间（stack space）……78
4.5.1 使用高效的算法……79
4.5.2 尽可能快地释放空闲资源……80
4.5.3 静态对象……81
4.5.4 庞大的对象……82
4.6 效率的权衡……83
4.6.1 实现更加困难……84
4.6.2 使用更加困难……86
4.7 总结……86
4.8 练习……87
4.9 参考文献和相关资料……89

**第5章 错误**……91

5.1 可重用代码中的错误……91
5.2 错误检测……92
5.2.1 函数前提条件……93
5.2.2 表示不变性……93