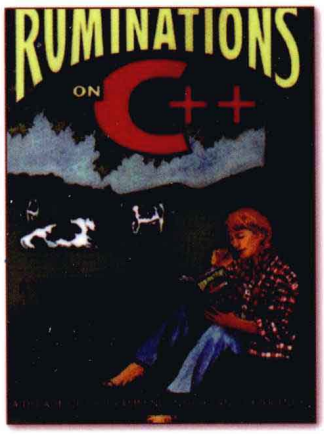
PEARSON

C和C++实务精选
品味岁月积淀，读享技术菁华

C++沉思录
（英文版）

[美] Andrew Koenig　Barbara E. Moo 著



Ruminations on C++

- C++经典著作
- 十年编程生涯的真知灼见
- 成长为C++高手的必经之路

人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

C++沉思录

（英文版）

[美] Andrew Koenig　Barbara E. Moo 著

人民邮电出版社
北京

图书在版编目（CIP）数据

C++沉思录 : 英文 / (美) 凯尼格 (Koenig, A.), (美) 莫 (Moo, B. E.) 著. -- 北京 : 人民邮电出版社, 2013.2
ISBN 978-7-115-30851-1

Ⅰ. ①C… Ⅱ. ①凯… ②莫… Ⅲ. ①C语言－程序设计－英文 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2013)第012372号

版权声明

C++沉思录（英文版）

◆ 著 [美] Andrew Koenig Barbara E. Moo
责任编辑 汪 振

◆ 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 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京铭成印刷有限公司印刷

◆ 开本：700×1000 1/16
印张：24.75
字数：525 千字 2013 年 2 月第 1 版
印数：1－2 500 册 2013 年 2 月北京第 1 次印刷

著作权合同登记号 图字：01-2012-9286 号
ISBN 978-7-115-30851-1

定价：59.00 元

读者服务热线：(010)67132692 印装质量热线：(010)67129223
反盗版热线：(010)67171154

内容提要

本书基于作者在知名技术杂志发表的技术文章、世界各地发表的演讲以及斯坦福大学的课程讲义整理、写作而成，融聚了作者 10 多年 C++程序生涯的真知灼见。

全书分为 6 篇 32 章，分别对 C++语言的历史和特点、类和继承、STL 与泛型编程、库的设计等几大技术话题进行了详细而深入的讨论，细微之处几乎涵盖了 C++所有的设计思想和技术细节。全书通过精心挑选的实例，向读者传达先进的程序设计的方法和理念。

本书适合有一定经验的 C++程序员阅读学习，可以帮助读者加强提高技术能力，成为 C++程序设计的高手。

作者简介

Andrew Koenig

AT&T 大规模程序研发部（前贝尔实验室）成员。他从 1986 年开始从事 C 语言的研究，1977 年加入贝尔实验室。他编写了一些早期的类库，并在 1988 年组织召开了第一个具有相当规模的 C++会议。在 ISO/ANSI C++委员会成立的 1989 年，他就加入了该委员会，并一直担任项目编辑，他已经发表了 C++方面的 100 多篇论文，在 Addsion-Wesley 出版了 *C Traps and Pitfalls* 一书（中文版名为《C 缺陷与陷阱》，由人民邮电出版社出版），还应邀到世界各地演讲。

Barbara Moo

现任 AT&T 网络体系结构部门负责人。在 1983 年加入贝尔实验室不久，她开始从事 Fortran77 编译器的研究工作，这是第一个用 C++编写的商业产品。她负责 AT&T 的 C++编译器项目直到 AT&T 卖掉它的软件业务。她还为 SIGS 会议、Lund 技术学院和 Stanford 大学提供辅导课程。她还是 C++Primer（5th Edition）的合著者。

Anderw Koenig 和 Barbara Moo 不仅有着多年的 C++开发、研究和教学经验，而且还亲身参与了 C++的演化和变革，对 C++的变化和发展起到重要的影响。

前言

原由

1988 年初，大概是我刚刚写完 *C Traps and Pitfalls*（本书中文版《C 陷阱与缺陷》由人民邮电出版社出版）的时候，Bjarne Stroustrup 跑来告诉我，他刚刚被邀请参加了一个新杂志的编委会，那个杂志叫做《面向对象编程月刊》(*Journal of Object-Oriented Programming，JOOP*)。该杂志试图在那些面孔冰冷的学术期刊与满是产品介绍和广告的庸俗杂志之间寻求一个折中。他们在找一个 C++专栏作家，问我是否感兴趣。

那时，C++对于编程界的重要影响才刚刚开始。Usenix 其时才刚刚在新墨西哥圣达菲举办了第一届 C++交流会。他们预期有 50 人参加，结果到场的有 200 人。更多的人希望搭上 C++快车，这意味着 C++社群急需一个准确而理智的声音，去对抗必然汹涌而至的谣言大潮。需要有个人能够在谣言和实质之间明辨是非，在任何混乱之中保持冷静的头脑。无论如何，我顶了上去。

在写下这些话的时候，我正在构思我为 *JOOP* 撰写的第 63 期专栏。这个专栏每期或者每两期就会刊登。其间，我也有过非常想中断的时候，非常幸运的是，Jonathan Shopiro 接替了我。偶尔，我只是写一些当期专栏的介绍，然后到卓越的丹麦计算机科学家 Bjørn Stavtrup[1]那里去求助。此外，Livleen Singh 曾跟我谈起为季刊 *C++ Journal* 撰写稿件的事，那个杂志在发行 6 期之后停刊了。Stan Lippman 也甜言蜜语地哄着我在 *C++ Report* 上开了个专栏，当时这本杂志刚刚从一分简陋的通信时刊正式成为成熟的杂志。加上我在 *C++ Report* 上发表的 29 篇专栏文章，我一共发表了 98 篇文章。

在这么多的杂志刊物里，分布着大量的材料。如果这些文章单独看来是有用的，

---

[1] 就是 C++创造者 Bjarne Stroustrup，这里可能是丹麦文。——译者注

那么集结起来应该会更有用。所以，Barbara[2]和我（主要是 Barbara）重新回顾了所有的专栏，选择出其中最好的，并根据一致性和连续性的原则增补和重写了这些文章。

本书正是世界所需的又一本 C++书籍

既然你已经知道了本书的由来，我就再讲讲为什么要读这本书，而不是其他的 C++书籍。天知道！C++方面的书籍太多了，为什么要选这一本呢？

第一个原因是，我想你们会喜欢它。大部分 C++书籍都没有顾及到这点：它们应该是基于科目教学式的。吸引人最多不过是次要目标。

杂志专栏则不同。我猜想肯定会有一些人站在书店里，手里拿着一本 *JOOP*，扫一眼我 Koenig 的专栏之后，便立刻决定购买整本杂志。但是要是我自认为这种情况很多的话，就未免太狂妄自大了。绝大多数读者都是在买了书之后读我的专栏的，也就是说他们有绝对的自由来决定是否读我的专栏。所以，我得让我的每期专栏都货真价实。

本书不对那些晦涩生僻的细节进行琐碎烦人的长篇大论。初学者不应该指望只读这本书就能学会 C++。具备了一定基础的人，比如已经知道几种编程语言的人，以及已经体会到如何通过阅读代码推断出一门新语言的规则的人，将能够通过本书对 C++有所了解。大部分从头开始学的读者先读 Bjarne Stroustrup 的 *The C++ Programming Language*（Addison-Wesley 1991）或者 Stan Lippman 的 *C++ Primer*（Addison-Wesley 1991），然后再读这本书，效果可能会更好。[3]

这是一本关于思想和技术的书，不是关于细节的。如果你试图了解怎样用虚基类实现向后翻腾两周半，就请到别处去找吧。这里所能找到的是许多等待你去阅读分析的代码。请试一试这些范例。根据我们的课堂经验，想办法使这些程序运行起来，然后加以改进，能够很好地巩固你的理解。至于那些更愿意从分析代码开始学习的人，我们也从本书中挑选了一些范例，放在 ftp.aw.com 的目录 cseng/authors/koenig/ruminations 下，可以匿名登录获取。

如果你已经对 C++有所了解，那么本书不仅能让你过一把瘾，而且能对你有所启示。这也是你应该阅读本书的第二个原因。我的意图并不是教 C++本身，而是想告诉你用 C++编程时怎样进行思考，以及如何思考问题并用 C++表述解决方案。知识可以通过系统学习获取，智慧则不能。

---

[2] 本书合作者 Barbara Moo 是 Andrew Koenig 的夫人，退休前是 Bell 实验室高级项目管理人员，曾负责 Fortran 和 CFront 编译器的项目管理。——译者注

[3] 这两本 C++百科大全类的名著分别于 1997 年和 1998 年推出了各自的第三版，Bjarne Stroustrup 还于 2000 年推出了 *The C++ Programming Language* 特别版。——译者注

组织

就专栏来说，我尽力使每期文章都独立成章，但我相信，对于结集来说，如果能根据概念进行编排，将更易于阅读，也更有趣味。因此，本书划分为 6 篇。

第一篇是对主题的扩展介绍，这些主题将遍布本书的其余部分中。本部分中没有太多的代码，但是所展现的有关抽象和务实的基本思想贯穿本书，更重要的是，这些思想渗透了 C++设计原则和应用策略。

第二篇着眼于继承和面向对象编程，大多数人都认为这些是 C++中最重要的思想。你将知道继承的重要性何在，它能做什么。你还会知道为什么将继承对用户隐藏起来是有益的，以及什么时候要避免继承。

第三篇探索模板技术，我认为这才是 C++里最重要的思想。我之所以这样认为，是因为这些模板提供了一种特别的强大的抽象机制。它们不仅可以构造对所包含的对象类型一无所知的容器，还可以建立远远超出类型范畴的泛型抽象。

继承和模板之所以重要的另一个原因是，它们能够扩展 C++，而不必等待（或者雇佣）人去开发新的语言和编译器。进行扩展的方法之一就是通过类库。第四篇谈到了库——包括库的设计和使用。

对基础有了很好的理解以后，我们可以学习第五篇中的一些特殊编程技术了。在这部分，你可以知道如何把类紧密地组合在一起，或者把它们尽可能地分离开。

最后，在第六篇，我们将返回头来对本书所涉及到的内容做一个回顾。

编译和编辑

这些经年累月写出来的文章有一个缺陷，就是它们通常都没有用到语言的现有特性。这就导致了一个问题：我们是应该在 C++标准尚未最终定稿的时候，假装 ISO C++已经成熟了，然后重写这些专栏，还是维持古迹，保留老掉牙的过时风格呢？[4]

还有许多这样的问题，我们选择了折中。对那些原来的栏目有错的地方——无论是由于后来语言规则的变化而导致的错误，还是由于我们看待事物的方式改变而导致的错误——我们都做了修正。一个很普遍的例子就是对 const 的使用，自从 const 加入到语言中以来，它的重要性就在我们的意识中日益加强。

另一方面，例如，尽管标准委员会已经接受 bool 作为内建数据类型，这里大量的范例还是使用 int 来表示真或者假的值。这是因为这些专栏文章早在这之前就完成了，使用 int 作为真、假值还将继续有效，而且要使绝大多数编译器支持 bool 还需

[4] 本书编写于 1996 年底，当时 C++标准已经发布了草案第二版，非常接近最终标准。次年(1997)，C++标准正式定稿。本书内容是完全符合 C++标准的。——译者注

要一些年头。

致谢

除了在 *JOOP*、*C++ Report*、*C++ Journal* 中发表我们的观点外，我们还在许多地方通过发表讲演（和听取学生的意见）来对这些观点进行提炼。尤其值得感谢的是 Usenix Association 和 SIGS Publications 举办的会议，以及 *JOOP* 和 *C++ Report* 的发行人。另外，在 Western Institute in Computer Science 的赞助下，我们俩在斯坦福大学讲授过多次单周课程，在贝尔实验室我们为声学研究实验室和网络服务研究实验室的成员讲过课。还有 Dag Brück 曾为我们在瑞典组织了一系列的课程和讲座。Dag Brück 当时在朗德理工学院自动控制系任教，现在在 Dynasim AB。

我们也非常感谢那些阅读过本书草稿以及那些专栏并对它们发表意见的人：Dag Brück、Jim Coplien、Tony Hansen、Bill Hopkins、Brian Kernighan(他曾笔不离手地认真阅读了两遍)、Stan Lippman、Rob Murray、George Otto 和 Bjarne Stroustrup。

如果没有以下人员的帮助，这些专栏永远也成不了书。他们是 Deborah Lafferty、Loren Stevens、Addison-Welsey 的 Tom Stone 以及本书编辑 Lyn Dupré。

我们特别感谢 AT&T 开通的经理们，是他们使得编写这些专栏并编辑成书成为可能。他们是 Dave Belanger、Ron Brachman、Jim Finucane、Sandy Fraser、Wayne Hunt、Brian Kernighan、Rob Murray、Ravi Sethi、Bjarne Stroustrup 和 Eric Sumner。

Andrew Koenig
Barbara Moo
新泽西州吉列
1996 年 4 月

Contents

0

Prelude

Once upon a time, I met someone who had worked in many programming languages, but had never written a C or C++ program. He asked a question that made me think: "Can you convince me that I should learn C++ instead of C?" I've given talks about C++ to quite a number of people, but it suddenly dawned on me that these people had all been C programmers. How could I explain C++ to someone who had never used C?

I started by asking him what languages he had used that were similar to C. I knew he had programmed a lot in Ada—but that didn't help, because I didn't know Ada. He did know Pascal, though, as did I. I wanted to find an example that would rest on the limited basis of our common understanding.

Here is how I tried to explain to him what C++ can do well that C can't.

0.1 First try

The central concept of C++ is the class, so I started by defining a class. I wanted to write a complete class definition that was small enough to explain and might still be useful. Moreover, I wanted it to have public and private data, because I wanted to work data hiding into the example. After a few minutes of head scratching, I came up with something like this:

```
#include <stdio.h>

class Trace {
public:
        void print(char* s) { printf("%s", s); }
};
```

I explained how this defined a new type called `Trace`, and how to use `Trace` objects to print output messages:

```
int main()
{
        Trace t;
        t.print("begin main()\n");
        // the body of main goes here
        t.print("end main()\n");
}
```

So far, what I have done looks much like it would in other languages. In fact, even in C++, it might have made sense to use `printf` directly, rather than going through the extra step of defining a class and then creating an object of that class, in order to print these messages. However, as I went on explaining how the definition of class `Trace` worked, I realized that even this simple example had touched on some of the important things that make C++ so flexible.

0.1.1 A refinement

For instance, as soon as I had used the `Trace` class, I realized that it might be useful to be able to turn off the trace output on occasion. No problem; just change the class definition:

```
#include <stdio.h>

class Trace {
public:
        Trace() { noisy = 0; }
        void print(char* s) { if (noisy) printf("%s", s); }
        void on() { noisy = 1; }
        void off() { noisy = 0; }

private:
        int noisy;
};
```

Now the class definition includes two new public member functions called `on` and `off`, which affect the state of a private member named `noisy`. Printing occurs only if `noisy` is on (nonzero). Thus, saying

```
t.off();
```

will suspend printing on `t` until we later resume printing by saying

```
t.on();
```

I also pointed out that, because these member functions were part of the `Trace` class definition itself, the C++ implementation would expand them inline, thus making it inexpensive to keep the `Trace` objects in the program even when no tracing was being done. It occurred to me that I could effectively turn off all `Trace` objects at once simply by defining `print` to do nothing and recompiling the program.