*Safe C++*

# C++编程调试秘笈

[美] *Vladimir Kushnir* 著
徐波 译

O'REILLY®

人民邮电出版社
POSTS & TELECOM PRESS

O'REILLY®

# C++编程调试秘笈

[美] Vladimir Kushnir 著

徐波 译

人 民 邮 电 出 版 社

北 京

**图书在版编目（CIP）数据**

C++编程调试秘笈 / （美）弗拉基米尔（Kushnir, V.）著 ; 徐波译. -- 北京 : 人民邮电出版社, 2013.1
ISBN 978-7-115-29695-5

Ⅰ. ①C… Ⅱ. ①弗… ②徐… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第244984号

## 版权声明



**C++编程调试秘笈**

◆ 著 [美] Vladimir Kushnir
译 徐 波
责任编辑 陈冀康

◆ 人民邮电出版社出版发行 北京市东城区夕照寺街 14 号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：787×1000 1/16
印张：8.5
字数：159 千字 2013 年 1 月第 1 版
印数：1－3 000 册 2013 年 1 月河北第 1 次印刷

著作权合同登记号 图字：01-2012-5858 号
ISBN 978-7-115-29695-5

定价：29.00 元

**读者服务热线：(010)67132692 印装质量热线：(010)67129223**
**反盗版热线：(010)67171154**
**广告经营许可证：京崇工商广字第 0021 号**

# 前　言

敏锐的读者可能会根据本书的英文书名《Safe C++》推断出C++编程语言多少是有点儿不安全的。这确实是很灵敏的感觉！并且非常正确。C++语言可能导致程序员出现所有类型的错误，例如访问一个动态分配的数组边界之外的内存，或者从那些从未初始化过的内存进行读取，或者分配了内存但忘了销毁它。简而言之，程序员在使用C++进行编程的时候，会有很大的几率搬起石头砸自己的脚。很可能一切都非常顺利，程序却突然崩溃，或者产生不可理喻的结果，或者出现了计算机术语中称为“不可预料的行为”。因此，从这层意义上说，C++语言在本质上是不安全的。

本书讨论了程序员在C++编程中所犯的一些最为常见的错误，并提供了避免这些错误的方法。在过去的岁月里，C++社区积累了许多优秀的编程实践。在编写本书时，作者收集了其中的一些实践，并对它们进行了稍微的修改，另外增加了作者的一些实践。作者希望这些作为缺陷捕捉策略的规则集能够达到事半功倍的效果。

不可否认的真相是，任何比“Hello, World”复杂得多的程序都可能包含一些错误，或可以充满感情色彩地称之为缺陷（bug）[1]。编程的一个很大课题是怎样减少缺陷的数量，同时又不至于明显延缓开发进程使之陷入停顿。为此，我们需要回答下面这个问题：应该由谁来捕捉这些缺陷？

在软件程序的生命周期中，共涉及4类参与者（见图P-1）：

（1）程序员。

（2）编译器（例如Unix/Linux中的g++、Windows中的Microsoft Visual Studio和Mac OS X中的XCode）。

（3）应用程序的运行时代码。

（4）程序的用户。

当然，我们并不想让用户看到缺陷，甚至不想让他们知道缺陷的存在，因此现在只

[1] 译注：在程序员社区中，一般都是直接引用原文bug，并不进行翻译。出于出版物的严谨与规范，本书还是把bug翻译为“缺陷”。

剩下参与者 1 到 3。和用户一样，程序员也是人，人有可能疲劳、困倦、饥饿，以及由于同事的提问或者接听某位家庭成员或汽车修理工的电话而分心。因此，程序员是容易犯错的，很容易制造缺陷。反之，参与者 2 和 3（编译器和可执行代码）具有某种优势：它们并不会疲劳、困倦或热情消退，也不会参加会议或休假，更不需要用餐。它们只是执行指令，并且通常非常善于做这项工作！

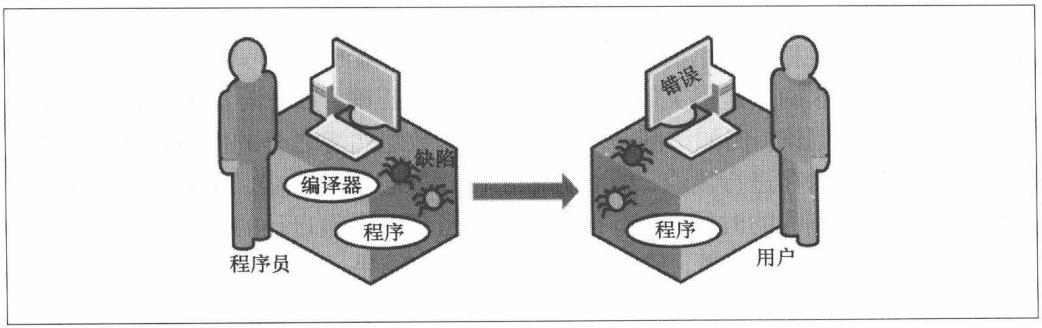



图 P-1　4 个参与者（缺陷多多的版本）

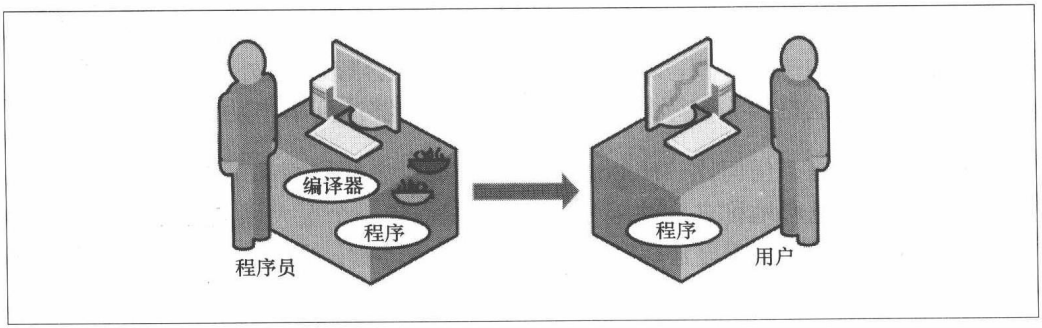



图 P-2　4 个参与者（愉快/没有缺陷的版本）

考虑我们必须面对的资源：一方面是程序员，另一方面是编译器和程序。我们可以采取两种策略之一来减少缺陷的数量。

1 号策略：说服程序员不许犯错。密切注视程序员，威胁每发现一处缺陷就要从他的奖金里扣除 10 美元，或者想方设法让他紧张起来以提高他的工作效率。例如，告诉他“每次当你分配内存时，不要忘了销毁它！”，诸如此类。

2 号策略：根据一种现实的推断（即使怀着最热切的意愿并保持激光一样的专注力，程序员仍然会在代码中制造缺陷），对编程和测试的整个过程进行组织。因此，不要对程序员说“每次当你做 A 时，不要忘了做 B”，而是制定一些规则，在用户运行应用程序之前，使大多数缺陷被编译器和运行时代码所捕捉，如图 P-2 所示。

当我们编写C++代码时，应该追求以下3个目标。

(1) 程序应该执行它的预定任务。例如，计算每月的银行票据、播放音乐或编辑视频等。

(2) 程序应该容易被人理解。也就是说，源代码不仅是为编译器写的，而且要能够被人理解。

(3) 程序应该具有自我诊断功能。也就是说，能够寻找它所包含的缺陷。

这3个目标是按照它们在现实的编程世界中的关注度从高到低排列的。第一个目标对于每个人都是不言而喻的，第二个目标是部分人所追求的，第三个目标就是本书的主题。我们不应该自己捕捉缺陷，而是由编译器和可执行代码为我们做这些事情。它们负责这些乏味的工作，我们的精力放在算法和设计上，也就是编程中有趣的那部分内容。

### 读者

如果读者从来没有用过C++进行编程，那么本书并不适合您。本书并不是为C++初学者准备的，它假设读者已经熟悉C++的语法，并且能够理解像构造函数、拷贝构造函数、赋值操作符、析构函数、操作符重载、虚拟函数和异常等概念。本书是为具有一定水准的C++程序员所准备的，包括初级和中级水平的程序员。

### 本书的组织形式

在本书的第一部分，我们将讨论下面这3个问题。

在第1章中，我们将讨论书名的问题。注意，这个问题涵盖了所有问题。

在第2章中，我们将讨论为什么最好尽量在编译时捕捉缺陷。第2章的剩余部分将描述怎样实现这个目标。

在第3章中，我们将讨论在运行时发现缺陷时应该怎么做。为了捕捉错误，我们尽力使安全检查（一段为了诊断错误这个特定目标而编写的代码）的编写变得轻松。实际上，这项工作已经完成了，附录A包含了编写安全检查所需要的宏的代码，包括能够产生与发生了什么？在哪里发生？为什么发生等问题有关的详细信息，同时并不需要程序员做太多的事情。在本书的第二部分，我们将讨论不同类型的错误，一次讨论一种错误，并制定让这些错误不再发生（或至少很容易捕捉）的规则。在本书的第三部分，我们应用了第二部分介绍的安全C++库的所有规则和代码，并讨论了以最高效的方式捕捉缺陷的测试策略。

我们还讨论了怎样使程序“可调试”。在编写程序时，其中一个目标就是使它很容易被调试，我们将介绍怎样在两个得力的助手（编译器和运行时代码）中添加第三个助手，即调试器，特别是当我们处理那些以“调试器友好”的方式所编写的代码时。

现在，我们准备捕捉实际的缺陷了。在本书的第二部分，我们逐个讨论 C++代码中最为常见的错误类型，并为每种错误制定一种策略（或者简单地确定一个规则），使之不可能发生或者很容易在运行时被捕捉。接着，我们讨论了每个特定规则的长短优劣以及它所存在的限制。在每章的最后，用这个规则的简短表述形式进行总结，这样当读者想跳过具体的讨论内容直接观察结论时，就知道可以在哪里找到它们。第 17 章对所有的规则进行了总结。附录部分包含了本书所使用的所有必要的 C++文件的源代码。

此时读者可能会问：“可不可以这样说，现在不再像以前的说法‘当你做 A 时，不要忘了做 B’，而是变成了‘当你做 A 时，要遵循规则 C’？这又有什么区别呢？是不是存在更加确定的方式来摆脱缺陷呢？”好问题！首先，有些问题（例如内存的销毁）可以在语言层次上得到解决。实际上，这个目标已经实现，它们就是 Java 或 C#。但是对于本书而言，我们假设出于某些原因（例如存在大量的遗留代码，对性能有着极为严格的要求，或者对 C++语言有着非比寻常的感情），还是坚持使用 C++。

在这个前提下，遵循这些规则为什么要比原来的“不要忘记”是更好的答案呢？这是因为在许多情况下，规则的实际表述形式像下面所述。

- 原先的表述：“在这里分配了内存之后，不要忘了检查其他全部 20 处需要销毁它的地方，并保证在这个函数中添加其他的 return 语句后，不要忘了添加相应的清理代码。”
- 新的表述：“分配了内存之后，立即把它赋值给一个智能指针，接着就可以放松了，不再关注此事”。

我们显然更加同意第二种方式，它更简单并且更可靠。当然，我们并不能百分之百地保证程序员不会忘了把内存赋值给一个智能指针，但是它比原来的表述形式要容易实现得多，并且要可靠很多。

需要注意，本书并没有涵盖多线程。为了准确起见，本书在讨论内存泄漏时简单地提到了多线程，但也就到此为止了。多线程是非常复杂的，它向程序员提供了许多机会制造许多微妙的、难以复制并且很难寻找的错误。但是，这应该是另一本更为高阶的书籍的主题。

当然，作者并不是声称本书所建议的规则是唯一正确的规则。相反，程序员们可以充满热情地主张其他的实践，只要适合他们就是最好的。编写良好的C++代码的方法有很多种。但是，下面所述是作者的主张。

- 如果遵循本书所描述的规则的精神（甚至可以自己添加规则），可以加快代码的开发速度。
- 在一开始几分钟或几小时的测试过程中，可以捕捉到代码中所存在的大部分（如果不是全部）错误。因此，在编写代码时，就不会有太大的压力。
- 最后，在完成测试时，可以合理地保证自己的程序不再包含某种类型的缺陷。这是因为已经添加了所有必要的安全检查，并且它们都获得了通过。

那么可执行代码的效率会不会受到影响呢？读者可能会担心寻找缺陷需要付出代价。不必担心，在本书的第三部分“捕捉缺陷的乐趣：从测试到调试到产品”，我们将讨论怎样保证产品代码保持应有的效率。

### 本书所使用的约定

下面是本书所使用的字体约定。

#### 斜体

表示新术语、URL、电子邮件地址、文件名和文件扩展名。

#### 等宽

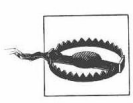
用于程序清单以及在段落中表示程序元素的片段，例如变量或函数名、数据库、数据类型、环境变量、语句和关键字等。

#### 等宽粗体

显示程序所产生的输出。

提示

这个图标用来强调一个提示、建议或一般说明。



警告

这个图标用来表示一个警告或注意事项。

### 命名约定

作者非常看重命名约定的作用。读者可以使用自己所喜欢的任何约定，不过下面是作者为本书所选择的约定。

- 类名由几个单词无缝拼接而成，每个单词的首字母均采用大写，例如：

```
class MyClass {
```

- 函数名（包括方法）的约定与类名相同，例如：

```
MyClass(const MyClass& that);
void DoSomething() const;
```

这是因为在 C++中，构造函数的名称必须与类名相同（析构函数也类似）。因为它们同时也是类的函数，所以我们把所有函数名的约定都设置为与类名相同。

- 变量由所有字母均小写的单词组成，单词之间用下划线连接，例如：

```
lowercase_and_glued_together_using_underscore
```

- 类的数据成员名采用与变量相同的约定，但在末尾添加一条下划线：

```
class MyClass {
 public:
  // some code

 private:
  int int_data_;
};
```

这些规则的唯一例外是在使用像 std::vector 这样的 STL（标准模板库）容器的时候。在这种情况下，我们使用 STL 的命名约定，这样当读者决定用 scpp::vector（本书定义的所有类都位于 scpp 名字空间）代替 std::vector 的时候，基本上不需要对代码进行什么修改。像 scpp::array 和 scpp::matrix 这样的类采用与 scpp::vector 相同的命名约定，因为它们是与 vector 相似的容器。

在我们开始学习之前最后需要注意的一点是，本书的所有代码例子都是在一台运行 Max OS X 10.6.8（Snow Leopard）的 Mac 计算机上使用 g++编译器或 XCode 进行编译并测试的。作者尽量避免依赖平台的特性，但是读者所面临的环境可能是各不相同的。作者尽最大的努力保证本书的附录部分所提供的安全 C++库的代码是正确的，在自己的知识能力的范围内保证它们是没有缺陷的。同样，并不能保证读者在使用它们的时候万无一失。书中所讨论的所有 C++代码和头文件都在书末的附录中列出，

读者也可以从 https://github.com/ vladimir-kushnir/SafeCPlusPlus 下载这些代码。

我们已经概括了本书的学习路线。最终的目标是产生包含更少缺陷的优质代码，能够提高程序员的工作效率并减少头疼的问题，缩短开发周期，并且更能保证代码正确地工作。

### 使用代码例子

本书是为了帮助读者更好地完成自己的工作。一般而言，读者可以在自己的程序和文档中使用本书的代码。读者并不需要联系我们以获得许可，除非读者复制了书中相当大部分的代码。例如，在编写程序时使用本书的几段代码，并不需要获得许可。但是，销售或发布 O'Reilly 书籍的实例 CD-ROM 则要求获得许可。引用本书的内容或实例代码回答问题并不需要获得许可。把本书中相当数量的实例代码复制到读者的产品文档中则需要获得许可。

我们赞赏但并不要求读者注明引用。如果要注明引用，它一般包含标题、作者、出版商和 ISBN。例如“《Safe C++》，作者 Vladimir Kushnir，版权所有 2012 Vladimir Kushnir， 978-1-449-32093-5”。

如果读者觉得对代码例子的使用超出了正常范围或上面所提到的许可，可以通过 permissions@oreilly.com 与我们联系。

### Safari® Books Online

Safari® Books Online（www.safaribooksonline.com）是一种按需求定制的数字图书馆，同时以书籍和视频的形式提供来自全球的技术和商业领域前沿作者的专家内容。

技术专业人员、软件开发人员、网页设计人员以及业务或创新专业人员可以使用 Safari. Books Online 作为探索、解决问题、学习和认证培训的主要资源。

Safari. Books Online 为各家公司、政府机构和个人提供了广泛的产品结构和定价程序。订阅者可以通过一个可完整搜索的数据库访问数以千计的书籍、培训视频和预出版手稿，包括 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、 McGraw-Hill、Jones & Bartlett 和 Course Technology 等出版权的作品。关于 Safari® Books Online 的详细信息，可以在线访问我们。

### 联系我们

读者可以把与本书有关的评论和问题发给出版商：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938（美国或加拿大）
707-829-0515（国际或本地）
707-829-0104（传真）

我们为本书提供了一个网页，列出了勘误表、实例以及所有的额外信息。读者可以通过下面的地址访问这一网页：

```
http://oreil.ly/SafeCPP
```

如果想对本书进行评价或提出技术问题，可以发邮件到：

```
bookquestions@oreilly.com
```

关于我们的书籍、课程、会议和新闻的更多信息，可以访问我们的网站：`http://www.oreilly.com`。

我们的 Facebook：`http://facebook.com/oreilly`。

我们的 Twitter：`http://twitter.com/oreillymedia`。

在 YouTube 关注我们：`http://www.youtube.com/oreillymedia`。

**致谢**

首先，我想感谢 O'Reilly 出版社的 Mike Hendrickson，他认识到了本书的价值，并鼓励我编写本书。

我非常感谢我的编辑 Andy Oram，他所接受的任务非同寻常，因为这是我第一次写作，并且英语并非我的母语。Andy 的编辑工作大大增强了本书的可读性。我还非常欣赏他与作者的良好合作方式，并对我们之间的合作感到非常愉快。我尤其要感谢 Emily Quill 对本书的风格和文本的清晰作出的重要贡献。如果发现书中的错误，都是由我引起的。

我还想利用这个机会感谢 Valery Fradkov 博士，他在很早以前曾经教我编程，并为我们的第一个程序提供了许多思路。

我还想感谢我的儿子 Misha，他帮助我了解最新版本的 Microsoft Visual Studio 的动态。最后，我必须对我的妻子 Daria 表达永恒的感谢，感谢她在进行此项目期间对我的支持。

# 目　录


## 第一部分　C++的缺陷捕捉策略 …… 1

第 1 章　C++的缺陷来自哪里 …… 3
第 2 章　什么时候捕捉缺陷 …… 5
2.1　为什么编译器是捕捉缺陷的最好场合 …… 5
2.2　怎样用编译器捕捉缺陷 …… 6
2.3　处理类型的正确方式 …… 7
第 3 章　在运行时遇见错误该怎么办 …… 11

## 第二部分　捕捉缺陷：一次处理一个缺陷 …… 17

第 4 章　索引越界 …… 19
4.1　动态数组 …… 19
4.2　静态数组 …… 24
4.3　多维数组 …… 26
第 5 章　指针运算 …… 31
第 6 章　无效的指针、引用和迭代器 …… 33
第 7 章　未初始化的变量 …… 37
7.1　初始化的数值（int、double 等） …… 37
7.2　未初始化的布尔值 …… 40
第 8 章　内存泄漏 …… 43
8.1　引用计数指针 …… 47
8.2　作用域指针 …… 49
8.3　用智能指针实行所有权 …… 51
第 9 章　解引用 NULL 指针 …… 53
第 10 章　拷贝构造函数和赋值操作符 …… 55
第 11 章　避免在析构函数中编写代码 …… 57
第 12 章　怎样编写一致的比较操作符 …… 63
第 13 章　使用标准 C 函数库的错误 …… 67

# 第三部分　捕捉缺陷的乐趣：从测试到调试到产品 ····· 69

第 14 章　基本的测试原则 ····· 71
第 15 章　调试错误的策略 ····· 75
第 16 章　使代码更容易调试 ····· 79
第 17 章　总结 ····· 85
附录 A　本书所使用的 scpp 库的源代码 ····· 89
附录 B　scpp_assert.hpp 和 scpp_assert.cpp 文件的源代码 ····· 91
附录 C　scpp_vector.hpp 文件的源代码 ····· 93
附录 D　scpp_array.hpp 文件的源代码 ····· 95
附录 E　scpp_matrix.hpp 文件的源代码 ····· 97
附录 F　scpp_types.hpp 文件的源代码 ····· 99
附录 G　scpp_refcountptr.hpp 文件的源代码 ····· 103
附录 H　scpp_scopedptr.hpp 文件的源代码 ····· 105
附录 I　scpp_ptr.hpp 文件的源代码 ····· 107
附录 J　scpp_date.hpp 和 scpp_date.cpp 文件的源代码 ····· 109

# 第一部分

# C++的缺陷捕捉策略

本书第一部分对可能潜入到C++程序中的各种错误进行了分类。我们描述了在编译时而不是在测试时捕捉错误的价值所在，并提出了一些基本原则。当我们寻找特定的技巧以防止或捕捉后面章节所讨论的缺陷时，就需要记住这些原则。

# 第 1 章

# C++的缺陷来自哪里

C++语言是非常独特的。虽然实际上所有的编程语言都从其他语言中吸收了一些思路、语法元素和关键字，C++却是吸收了另一种完整的语言，即 C 语言。事实上，C++语言的创建者 Bjarne Stroustrup 原先把他的新语言命名为“带类的 C”。这意味着如果我们已经使用了一些 C 代码，并且由于某种原因（例如科研或贸易）切换到一种面向对象的语言，就不需要在移植代码方面采取任何措施，只要安装新的 C++编译器，就可以对旧的 C 代码进行编译了，并且效果和原先的一模一样。我们甚至会觉得已经完成了从 C 到 C++的转换。最后这种想法虽然距离真相还很远，用真正的 C++所编写的代码与 C 代码看上去存在很明显的区别，但它还是提供了一个逐渐过渡的选项。也就是说，我们可以从现在编译运行的 C 代码出发，逐渐引入用 C++所编写的新代码段，慢慢与它们混合在一起，最终实现到纯 C++的切换。因此，C++的层次式设计具有它独特的市场推动力。

但是，其中还是存在一些复杂的地方：随着 C 的完整语法被新语言完整地吸收，它的设计哲学和存在的问题也同样被吸收。C 语言是在 1969 年～1973 年期间由 Dennis Rithie 在贝尔实验室创建的，其出发点是为了编写 Unix 操作系统。这项工作的一个伴随成果是诞生了一种高效的高级编程语言（与需要编写每条计算机指令的汇编语言相比）。也就是说，它所产生的编译后的代码应该具有尽可能快的速度。这种新的 C 语言的其中一项公开原则是，用户不应该为他没有使用到的特性而受到拖累。因此，为了追求高效的编译代码，对于程序员没有提出明确的要求，C 就绝对不会加以考虑。C 语言是为了速度而不是为了舒适而创建的，这就产生了一些问题。

首先，程序员可以创建一个某种长度的数组，并用一个超出该数组边界的索引值访问一个元素。更容易被滥用的是 C 的指针运算，程序员可以把指针运算所产生的任何值作为内存地址并对它进行访问，不管这块内存是否应该被访问。（实际上，这两个问题其实是同一个，只不过使用了不同的语法。）

程序员还可以在运行时使用 calloc()和 malloc()函数动态分配内存，并使用 free()函数负责动态内存的销毁。但是，如果忘了销毁或者不小心销毁了多次，其结果可能是灾难性的。

我们将在本书的第二部分深入讨论这些问题中的每一个。需要重视的是，C++在继承整个 C 时，除了传承它的高效原则，还继承了它的所有问题。因此，C++代码中的部分缺陷就来源于 C。

但是，故事并没有结束。除了来自于 C 的问题，C++自身也存在一些问题。例如，大多数人认为友函数和多重继承并不是良好的编程思路。C++具有自己分配内存的方法，它并不是调用像 calloc()或 malloc()这样的函数，而是使用操作符 new。new 操作符并不仅仅分配内存，它还创建对象，即调用它们的构造函数。与 C 的精神相同，使用 delete 操作符删除动态分配的内存是程序员的责任。现在的情况看起来与 C 相同：我们分配了一些内存，然后删除它。但是，复杂之处在于 C++具有两种不同的 new 操作符：

```
MyClass* p_object = new MyClass();  // 创建一个对象
MyClass* p_array = new MyClass[number_of_elements]; // 创建一个数组
```

在第一种情况下，new 操作符创建了一个 MyClass 类型的对象。在第二种情况下，它创建了一个相同类型的对象数组。与之对应的是，C++具有两种不同的 delete 操作符：

```
delete p_object;
delete [] p_array;
```

当然，一旦使用了“带方括号的 new”创建对象，就需要使用“带方括号的 delete”删除它们。这样就可能导致一种新的错误：混用 new 和 delete，其中一个带了方括号而另一个没有带方括号。如果出现了这种错误，就会对内存堆产生巨大的破坏。因此，我们可以总结如下：C++的缺陷大部分来源于 C，但 C++也引入了一些自讨苦吃的新方法。我们将在本书的第二部分讨论这些话题。