PEARSON

C和C++实务精选
品味岁月积淀，读享技术菁华

C++ 覆辙录

[美] | Stephen C. Dewhurst | 著
高博 | 译

C++ Gotchas:
Avoiding Common Problems in Coding and Design

- 市面上难得一见的C++编程病理学分析巨著
- C++元老、Cfront作者十五年一线工程经验精华
- 马上能用、常读常新的专家级程序员晋级手册

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

[美] | Stephen C. Dewhurst | 著
高博 | 译

C++覆辙录

C++ Gotchas:
Avoiding Common Problems in Coding and Design

人民邮电出版社
北京

图书在版编目（CIP）数据

C++覆辙录 /（美）杜赫斯特（Dewhurst, S. C.）著；高博译. -- 北京：人民邮电出版社，2016.4
ISBN 978-7-115-37259-8

Ⅰ. ①C… Ⅱ. ①杜… ②高… Ⅲ. ①C语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2015)第191898号

版权声明

◆ 著　[美] Stephen C. Dewhurst
译　高　博
责任编辑　傅道坤
责任印制　张佳莹　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　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：800×1000　1/16
印张：21.75
字数：430 千字　2016 年 4 月第 1 版
印数：1–3 000 册　2016 年 4 月河北第 1 次印刷

著作权合同登记号　图字：01-2014-5618 号

定价：69.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

内容提要

本书是C++大师 Stephen C. Dewhurst 根据多年教授C++课程中所遇到的常见错误的心得笔记编写而成。本书所有章节都从一个众所周知的、在日常编码或设计实践经常遭遇的问题入手，先指出其不足，再对其背后思想中存在的合理与不合理之处深入剖析，最后取其精华，去其糟粕，给出一个简洁、通用的方案，给出如何规避或纠正它们的建议，从而有助于 C++软件工程师避免重蹈前辈的覆辙。

本书适合具有一定C++编程经验的读者阅读。

再版序

从本书的上一版付梓，至今不觉一晃又七年矣。七年间，太多的语言和框架销声匿迹，或是面目全非。而 C++语言虽然也有了标准的版本更新，并且新的语言标准中在核心语言部分引入了不少重大的改变，但是总体来说，一方面，C++语言在代际兼容方面的表现当属最优秀之列（“仅仅使用新版本的编译器来重新编译一遍旧代码，就能够收获不少额外的好处”，Scott Meyers 语）；另一方面，C++语言的底层设计哲学，比如“不为未使用的特性付出性能代价”等，是始终未变的。作为一种多范型语言，C++语言坚持在各个范型之间追求平衡感，力求给予程序员以最灵活的方式将范型加以组合，以尽可能直接的方式将问题域映射到解域，而非像很多语言一样必须在解决问题时削足适履。任何一种语言，都必须经过时间的考验，方能说明其生命力是否强大。C++语言在目前新语言以每年近 30 种的速度问世的前提下，能够多年如一日地占据编程语言排行榜的前五位，不能不说它确实是一种长盛不衰的程序设计语言。

毋庸讳言，C++98 在时下仍然是使用最广泛的标准版本。虽然采用未来时态学习 C++11/14，甚至追学 C++17，永远是值得提倡的。但是，连 C++之父 Bjarne Stroustrup 也说，C++语言的学习，重要之处在于掌握正确的思维（见《程序员》2014 年 12 月刊，译者采访编译）。而欲掌握正确的思维，有两个基本的方法。一曰在实践中认识，即在日常工作中使用而非只在论坛上看到只言片语的例程就以为自己掌握了最新技术，其实每种新的语言特性应用起来都有要求的语境。高手能用的新招式，你未必能用。二曰考察语言特性的历史变迁，看看 C++98 的语法在不同的标准版本下有着怎样不同的语义，这些变化的原因是什么，如何在变化的过程中保证最大程度的代际兼容性，这些都是非常有意思的课题。还有，可以看看 C++98 中由于语言设计问题会带来程序员容易犯的哪些错误——这正是本书的主题——而在新的语言标准中，这些问题是如何通过语言设计的改进而解决的，甚至不妨问一问自己，如果手头只有 C 编译器又该怎么办。只有掌握了这样全面的信息，才能说学活了，才能在无论有没有先进语言设计支持的前提下都能

够采用适当方式解决问题，才能说掌握了正确的思维。

译者也幸。由于种种原因，上一版的图书中，本人意欲加入的译者注十去其七。本以为已无再版可能，全系培生教育集团版权经理李乐强先生鼓励，又承人民邮电出版社信息分社刘涛社长支持，我将尘封有年的旧译稿又翻出来仔细从头到尾审改一次，并将旧译中的加注选了相当部分还原出来。但求能达到初心里让读者能够通过读这一本书，能参考到十几二十本书的相关内容对照阅读、全面深刻地理解书中通过病理学方式讲述的 C++语言的重要知识点之目标。如果能实现这一点，也是满足了我个人的一点小小心愿。成书过程中，承淘宝高级经理林应、百度高级工程师徐章宁、刘海平和林向东、华为社区运营总监林旅强、Ucloud CEO 季昕华、亮风台合伙人唐荣兴、Ping++市场总监冯飞等费心审阅稿件并受益于他们的反馈良多，在此一并致谢。当然限于本人才疏，缺点错误在所难免，此概应由我本人一体负责。家人在我译书过程中体谅照顾甚多，希望本书的出版，能给你们带来快乐。

2016 年 3 月

草于上海交通大学软件学院

技术翻译：一种笔记体的创作尝试（译者序）

经过近一年的工作，这本近四百页的小册子终于和大家见面了。

这本书从一个读者的角度来看，当然主要地可以视为是对于当之无愧的C++大师 Stephen C. Dewhurst 在近十五年前原创的一本技术书籍的译作。但如果从译者的本意出发，它未尝不可以说是我本人十年来学习 C++、领悟C++和运用 C++的一个小结。2005 年起，我开始陆续在论坛中发表一些零碎的技术文章和翻译作品，并在企业和大学里作了一些演讲。和真正的一线工程师，以及即将踏上工程师岗位的同道们作了一些比较深入的交流之后，我才真真切切地感受到他们对于书本知识转化为真正实力的那种热切的渴求。现在每年出版的有关 C++的书籍车载斗量，但是如何能把这些“知识”尽可能多地转化成工程师手中对付真正的项目需求的“武器”？我感到自己负有责任来做一些工作，来对这个问题做出自己尝试性的解答。那末，最好的方式是创作一本新书吗？经过再三的权衡，我认为并非如此。作为一个未在 *C/C++ Users Journal* 或是 *Dr. Dobb* 上发表过任何文字的人，原创很难企及自己欲达成的号召力。并且，原创的话就意味着要自己照顾一切技术细节，我还决没有自大到认为已经有了那种实力的程度。可是，是否仅仅再去翻译一本新的 C++著作呢？那也不是。C++近几年来已不比往昔，新著作的翻译效率简直高得惊人，但单纯的翻译工作其实并不能消释读书人的费解。那末，我就想到：为什么不能挑选一本书，一方面将它翻译过来，另一方面以它作为“蓝本”，将自己的见解以笔记的形式融入其文字，并引导读者参读其它的技术书籍呢？对于某一个特定的技术细节，我希望达到的效果是：读者能够从我的翻译这“小小的一隅”延拓开去，从深度而言他们能够参阅其它专门就此发力的技术资料，获得某种技术或习惯用法的历史背景、推导逻辑、常见形式等翔实、全面、准确的信息；从广度而言，他们可以了解到编码与设计、细节与全局的关系，从而做到取舍中见思路、简化中见智慧，真正地把 C++这种优秀的、有着长久生命力的程序设计语言背后的有关软件工程的科学和艺术的成分“提炼”出来，化为自己实实在在的内功提升。这样的工作，我认为才是有它的价值在的，也是我这些年来下苦功夫研读了一二十种C++的高质量书籍，以及使用C++

交付了一些成功的工程之后有实力完成的——这就是我创作本书的初衷和原动力——以技术翻译为主体，并进行“笔记体”的再创作予读者以诠释和阅读参考的附加值，这就是我的答案。

不过，选取这样的一本作为“蓝本”的书籍殊非易事。首先，它本身需要有相当的深度和广度，否则难以面面俱到，从而也就难以体现 C++语言在各个层次上的大能。其次，它必须有相当的发散性，否则它就难以和已有的大量资料相结合，难以引导读者去重读他们之间已经看过，但未能充分理解的资料。再次，它还要有明确的主题组织，否则很可能会陷入空谈，使读者感觉难以理解和掌握，从而不能发挥应有的“知识”向“实力”的转化之效。最后，*C++ Gotchas* 落入我的视线，研读数次之后，我觉得它不仅完全符合“蓝本”的一切要求，并且 Stephen C. Dewhurst 大师还在数个方面给予了我太多的启迪：这本书所有的章节都从一个众所周知的、在日常编码或设计实践经常遭遇的问题入手，先是就事论事地指出其不足，再是对其背后思想中存在何种合理与不合理之处深入剖析，最后取之精华弃之糟粕，给出一个简洁、通用、美轮美奂的方案。有的条款中，大师会给出数种不同的解决之道，并一一评点其优劣之处，指出其适用场合；有的条款中，大师步步推进，先是给出一个去除错误的解，再进一步地优化它，直至与某种习惯用法和设计模式接壤作为点题之笔。从翻译的过程中，我自己真的是受益良多，希望我的读者能够收获更大。

在本书的翻译中，清华大学出版社的龙启铭编辑给予了我很大的帮助和鼓励，并促成这本书最终完稿。微软亚洲研究院的徐宁研究员和 EMC 中国的柴可夫工程师通读了全书，并给予了全面的审阅意见，包括不少技术和文字的问题，在此向他们深深致谢。另外，Hewlett-Packard 总部的 Craig Hilderbrandt 经理、上海交通大学计算机系的张尧弼教授、Phoenix 中国的唐文蔚高级工程师、谷歌中国的龚理工程师、微软亚洲工程院的魏波工程师、微软全球技术中心的陈曦工程师和 SAP 中国的劳佳工程师也都在本书写作的过程中给了我不小的帮助，在此一并致谢。当然，书中的错误和纰漏在所难免，这些理应由我本人负全部责任。另外要感谢的还有我的家人和同事们，没有你们的支持，我不可能坚持到底。希望本书的出版能够给你们带来快乐。

高博

2008年11月

于微软亚洲工程院上海分院

前言

本书之渊薮乃是近 20 年的小小挫折、大错特错、不眠之夜和在键盘的敲击中不觉而过的无数周末。里面收集了普遍的、严重的或有意思的 C++常见错误，共计九十有九。其中的大多数，（实在惭愧地说）都是我个人曾经犯过的。

术语“gotcha”[①]有其云谲波诡的形成历史和汗牛充栋的不同定义。但在本书中，我们将它定义为 C++范畴里既普遍存在又能加以防范的编码和设计问题。这些常见错误涵盖了从无关大局的语法困扰，到基础层面上的设计瑕疵，再到源自内心的离经叛道等诸方面。

大约 10 年前，我开始在我教授的 C++课程的相关材料中添加个别常见错误的心得笔记。我的感觉是，指出这些普遍存在的误解和误用，配合以正确的用法指导就像给学生打了预防针，让他们自觉地与这些错误作斗争，更可以帮助新入门的 C++软件工程师避免重蹈他们前辈的覆辙。大体而言，这种方法行之有效。我也深受鼓舞，于是又收集了一些互相关联的常见错误的集合，在会议上作演讲用。未想这些演讲大受欢迎（或是同病相怜之故也未可知？），于是就有人鼓励我写一本“常见错误之书”。

任何有关规避或修复 C++常见错误的讨论都涉及了其他的议题，最多见的是设计模式、习惯用法以及 C++语言特征的技术细节。

这并非一本讲设计模式的书，但我们经常在规避或修复 C++常见错误时发现设计模式是如此管用的方法。习惯上，设计模式的名字我们把每个单词的首字母大写，比如模板方法（Template Method）设计模式或桥接（Bridge）设计模式。当我们提及一种设计模式的时候，若它不是很复杂，则简介其工作机制，而详细的讨论则放在它们和实际代码相结合的时候才进行。除非特别说明，本书不提供设计模式的完全描述或极为详尽的讨论，这些内容可以参考 Erich Gamma 等人编写的 *Design Patterns* 一书。无环访问者（Acyclic Visitor）、单态（Monostate）和空件（Null Object）等设计模式的描述请参见 Robert Martin 编写的 *Agile Software Development* 一书。

---

① 译者注：gotcha 在本书中通译为“常见错误”，固然较之原文失之神韵，倒也算得通俗易懂。

从常见错误的视角来看，设计模式有两个可贵的特质。首先，它们描述了已经被验证成功的设计技术，这些技术在特定的软件环境中可以采用自定义的手法搞出很多新的设计花样。其次，或许更重要的是，提及设计模式的应用，对于文档的贡献不仅在于使运用的技术一目了然，同时也使应用设计模式的原因和效果一清二楚。

举例来说，当我们看到在一个设计里应用了桥接设计模式时，我们就知道在一个机制层里，一个抽象数据型别的实现并分解成了一个接口类和一个实现类。犹有进者，我们也知道了这样做是为了强有力地把接口部分同底层实现剥离，是故底层实现的改变将不会影响到接口的用户。我们不仅知道这种剥离会带来运行时的开销，还知道此抽象数据型别的源代码应该怎么安排，并知道很多其他细节。

一个设计模式的名字是关于某种技术极为丰富的信息和经验之高效、无疑义的代号。在设计和撰写文档时仔细而精确地运用设计模式及其术语会使代码洗练，也会阻止常见错误的发生。

C++是一门复杂的软件开发语言，而一种语言愈是复杂，习惯用法在软件开发中之运用就愈是重要。对一种软件开发语言来说，习惯用法就是常用的、由低阶语言特征构成的高阶语言结构的特定用法组合。总的来说，这和设计模式与高阶设计的关系差不多。是故，在 C++ 语言里，我们可以直接讨论复制操作、函数对象、智能指针以及抛出异常等概念，而不需要一一指出它们在语言层面上的最低阶实现细节。

有一点要特别强调一下，那就是习惯用法并不仅仅是一堆语言特征的常见组合，它更是一组对此种特征组合之行为的期望。复制操作是什么意思呢？当异常被抛出的时候，我们能指望发生什么呢？大多数本书中的建议都是在提请注意以及建议应用 C++编码和设计中的习惯用法。很多这里列举的常见错误常常可以直接视作对某种 C++习惯用法的背离，而这些常见错误对应的解决方案则常常可以直接视作对某种 C++习惯用法的皈依（参见常见错误 10）。

本书在 C++语言的犄角旮旯里普遍被误解的部分着了重墨，因为这些语言材料也是常见错误的始作俑者。这些材料中的某些部分可能让人有武林秘笈的感觉，但如果不熟悉它们，就是自找麻烦，在通往 C++语言专家的阳关大道上也会平添障碍。这些语言死角本身研究起来就是其乐无穷，而且

产出颇丰。它们被引入 C++语言总有其来头，专业的 C++软件工程师经常有机会在进行高阶的软件开发和设计时用到它们。

另一个把常见错误和设计模式联系起来的东西是，描述相对平凡的实例对于两者来说是差不多同等重要的。平凡的设计模式是重要的。在某些方面，它们也许比在技术方面更艰深的设计模式更为重要，因为平凡的设计模式更有可能被普遍应用。所以从对平凡设计模式的描述中获得的收益就会以杠杆方式造福更大范围的代码和设计。

差不多以完全相同的方式，本书中描述的常见错误涵盖了很宽范围内的技术困难，从如何成为一个负责的专业软件工程师的循循善诱（常见错误 12）到避免误解虚拟继承下的支配原则的苦口良言（常见错误 79）。不过，就与设计模式类似的情况看，表现得负责专业当然比懂得什么支配原则要对日复一日的软件开发工作来得受用。

本书有两个指导思想。第一个是有关习惯用法的极端重要性。这对于像 C++这样的复杂语言来说尤为重要。对业已形成的习惯用法的严格遵守使我们能够既高效又准确地和同行交流。第二个是对“其他人迟早会来维护我们写的代码”这件事保持清醒头脑。这种维护可能是直截了当的，所以这就要求我们把代码写得很洗练，以使那些称职的维护工程师一望即知；这种维护也可能是拐了好几道弯的，在那种情况下我们就得保证即使远在天边的某个变化影响了代码的行为，它仍然能够给出正确的结果。

本书中的常见错误以一组小的论说文章的形式呈现，其中每一组都讨论了一个常见错误或一些相互关联的常见错误，以及有关如何规避或纠正它们的建议。由于常见错误这个主题内廉的无政府倾向，我不敢说哪本书可以特别集中有序地讨论它。然而，在本书中，所有的常见错误都按照其错误本质或应用（误用）所涉的领域归类到相应的章节。

还有，对一个常见错误的讨论无可避免地会牵涉到其他的常见错误。当这种关联有它的意义时——通常确实是有的——我会显式地作出链接标记。其实，这种每个常见错误的为了增强关联性的描述本身也是有其讨厌之处的。比方说经常遇到一种情况就是还没来得及描述一个常见错误自身，倒先把为什么会犯这个错误的前因后果交代了一大篇。要说清这些个前因后果呢，好家伙，又非得扯上某种技术啦、习惯用法啦、设计模式啦或是语言细节什么的，结果在言归正传之前要兜更大的圈子。我已经尽力把这种

发散式的跑题减到最少了，但要是说完全消除了这种现象，那我就没说实话。要把 C++程序设计做到很高效的境界，那就得在非常多水火不容的方面作出如履薄冰的协调，想在研究大量相似的主题前就对语言作出像样的病理学分析，那只能说是不现实的。

把这本书从第 1 个常见错误到第 99 个常见错误这么挨个地读下去，不仅毫无必要，而且也谈不上明智。一气儿服下这么一帖虎狼之剂恐怕会让你一辈子再也学不成 C++了。比较好的阅读方法应该是拣一条你不巧犯过的，或是你看上去有点儿意思的常见错误开始看，再沿着里面的链接看一些相关的。另一种办法就是你干脆由着性子，想看哪儿看哪儿，也行。

本书也使用了一些固定格式来阐明内容。首先，错误的和不提倡的代码以灰色背景来提示，而正确和适当的代码却没有任何背景。其次，这里作示意用的代码为了简洁和突出重点，都经过了编辑。这么做的一个结果是，这里示例用的代码若是没有额外的支撑代码往往不能单独通过编译。那些并非平凡无用的示例源代码则可以在作者的网站里找到：www.semantics.org。所有这样的代码都由一个相对路径引出，像这样：

➤➤ gotcha00/somecode.cpp

最后，提个忠告：你不要把常见错误的重要性提升到和习惯用法、设计模式一样[①]。一个你已经学会正确地使用习惯用法和设计模式的标志是，当某个习惯用法或是设计模式正好是你手头的设计或编码对症良方时，它就“神不知鬼不觉地”在你最需要时从你的脑海里浮现出来了。

对常见错误的清醒意识就好比是对危险的条件反射：一回错，二回过。就像对待火柴和枪械一样，你不必非得烧伤或是走火打中了脑袋才学乖。总之，只要加强戒备就行了。把我这本手册当作是你面对 C++常见错误时自我保护的武器吧！

Stephen C. Dewhurst
于美国马萨诸塞州卡佛市
2002 年 7 月

① 译者注：作者用心良苦，怕读者“近墨者黑”，好的没记住反而坏的学会了。所以特意提醒所有读者，常见错误有些奇技淫巧，但毕竟难登大雅之堂。

致谢

编辑们经常在图书的“致谢”里落得个坐冷板凳的下场，有时甚至用一句“……其实我也挺感谢我那编辑的，我估计在我拼了命爬格子的时候此人大概肯定也是出过一点什么力的吧”就打发了。Debbie Lafferty，也就是本人的编辑，负责本书的问世。有一次，我拿着一本不足为道的介绍性的程序设计教材去找她搞个不足为道的合作提案，结果她反而建议我把其中一个有关常见错误的章节扩展成一本书。我不肯。她坚持。她赢了。值得庆幸的是，Debbie 在胜利面前表现得特别有风度，只是淡淡了说了一句站在编辑立场上的“你瞧，我叫你写的吧。”当然不止于此，在我拼了命爬格子的时候，她是颇出了一些力的。

我也感谢那些无私奉献了他们的时间和专业技能来使本书变得更好的审阅者们。审阅一本未经推敲的稿件是相当费时的，常常也是枯燥乏味的，有时甚至会气不打一处来，而且几乎肯定是讨不着什么好的（参见常见错误12），这里要特别赞美一下我的审阅者们入木三分而又深中肯綮的修改意见。Steve Clamage、Thomas Gschwind、Brian Kernighan、Patrick McKillen、Jeffrey Oldham、Dan Saks、Matthew Wilson 和 Leor Zolman 对书中的技术问题、行业规矩、誊对校样、代码片段和偶然出现的冷嘲热讽都提出了自己的宝贵意见。

Leor 在稿件出来之前很久就开始了对本书的“审阅”，书中一些常见错误的原始版本只是我在互联网论坛里发的一些帖子，他针对这些帖子回复了不少逆耳忠言。Sarah Hewins 是我最好的朋友，同时也是最不留情的批评家，不过这两个头衔都是在审阅我一改再改的稿件时获称的。David R. Dewhurst 在写作项目进行的时候经常把我拉回正轨。Greg Comeau 慷慨地让我有幸使用他堪称一流的标准 C++编译器来校验书里的代码[①]。

就像所有关于 C++的任何有意义的工作那样，本书也是集体智慧的结晶。这些年来，我的很多学生、客户和同事为我在 C++常见错误面前表现的呆

① 译者注：这应该就是著名的 Comeau C/C++ Front/End 编译器。

头呆脑和失足跌跤可没少数落过我，并且他们中的好多人都帮我找到了问题的解决之道。当然，这些特别可贵的贡献者中的大部分都没法在这里一一谢过，不过有些提供了直接贡献的人还是可以列举如下的。

常见错误 11 中的 Select 模板和常见错误 70 中的 OpNewCreator 策略都取自 Andrei Alexandrescu 编写的 *Modern C++ Design* 一书。

我在常见错误 44 中描述了有关返回一个常量形参的引用带来的问题[①]，此问题我初见于 Cline 等人编写的 *C++ FAQs* 一书（我客户的代码中在此之后马上就用上了这个解决方案）。此书还描述了我在常见错误 73 中提到的用于规避重载虚函数的技术。

常见错误 83 中的那个 Cptr 模板，其实是 Nicolai Josuttis 编写的 *The C++ Standard Library* 一书中 CountedPtr 模板的一个变形。

Scott Meyers 在他的 *More Effective C++*一书中，对运算符&&、||和,,的重载之不恰当性提出了比我在常见错误 14 的描述更深入的见解。他也在他的 *Effective C++*一书中，对我在常见错误 58 中讨论的二元运算符以值形式返回的必要性作了更细节的描述，还在 *Effective STL* 一书中描述了我在常见错误 68 里说的对 auto_ptr 的误用。在后置自增、自减运算符中返回常量值的技术，也在他的 *More Effective C++*一书中提到了。

Dan Saks 对我在常见错误 8 中描述的前置声明文件技术提出了最有说服力的论据，他也是区别出常见错误 17 中提及的“中士运算符”的第一人，他也说服了我在 enum 型别的自增和自减中不去做区间校验，这一点被我写在了常见错误 87 中。

Herb Sutter 的 *More Exceptional C++*一书中的条款 36 促使我去重读了 C++ 标准 8.5 节，然后修正了我对形参初始化的理解（见常见错误 57）。

常见错误 10、27、32、33、38～41、70、72～74、89、90、98 和 99 中的一些材料出自我先是在 *C++ Report*，后来在 *The C/C++ Users Journal* 撰写的 *Common Knowledge* 专栏。

① 译者注：那是个有关临时对象生存时域的问题。

目录