<<Exceptional C++ Styl>>

图书基本信息

书名：<<Exceptional C++ Style中文版>>

13位ISBN编号：9787115142252

10位ISBN编号：7115142254

出版时间：2006-1

出版时间：人民邮电出版社

作者：Herb Sutter

页数：276

译者：刘未鹏

<<Exceptional C++ Styl>>

内容概要

软件“风格”所要讨论的主题是如何在开销与功能之间、优雅与可维护性之间、灵活。
性与过分灵活之间寻找完美的平街点。
在本书中，著名的C++大师Herb Sutter给出了40个编程问题。
其目的是为了使读者不仅“知其然”，更要“知其所以然”，并帮助读者在软件开发中进行正确的决策。
本书是围绕实际问题及其解决方案展开论述的，对一些至关重要的C++细节和相互关系提出了新的见解，为当今的关键C++编程技术(如泛型编程、STL、异常安全等)提供了新的策略。
本书中，C++大师Herb sutter通过40个编程问题，使读者不仅“知其然”，更要“知其所以然”，帮助程序设计人员在软件中寻找恰到好处的折中，即讨论如何在开销与功能之间、优雅与可维护性之间、灵活性与过分灵活之间寻找完美的平衡点。
本书是围绕实际问题及其解决方案展开论述的，对一些至关重要的C++细节和相互关系提出了新的见解，为当今关键的C++编程技术(如泛型编程、STL、异常安全等)提供了新的策略。
本书的目标是让读者在设计、架构和编码过程中保持良好的风格，从而使编写的C++软件更健壮、更高效。
本书适合中高级C++程序员阅读。

作者简介

Herb Sutter ISO C++标准委员会主席，C++ Users Journal杂志特邀编辑和专栏作家。
他目前在微软公司领导NET环境下C++语言扩展的设计工作。
除本书外，他还撰写了三本广受赞誉的图书：Exceptional C++ Style（中文版即将由人民邮电出版社出版）、Exceptional C++ Style和More Exceptional C++ Style。

<<Exceptional C++ Styl>>

书籍目录

章节摘录

前言　　布达佩斯，匈牙利的首都。
一个炎热的夏日傍晚。
穿过美丽的多瑙河望去，余晖中的东岸景色优美恬静。
本书封面上色彩柔和的欧洲风光中，哪栋建筑首先映入你的眼帘？
几乎可以肯定，是照片左边的国会大厦。
这栋巨大的新哥特式建筑以它优美的圆穹、直插天际的尖塔、不计其数的外墙雕塑以及其他华丽装饰一下攫住了你的目光，而它更引人注目之处，在于它与四周建筑在多瑙河畔那些刻板的实用建筑形成了极其鲜明的对照。
为什么会有这么大的差异呢？
一方面，国会大厦是在1902年竣工的，而其他味同嚼蜡的建筑则大部分都是在二战以后建成的。
“啊哈，”你可能会想，“这的确解释了为什么差异如此之大。
然而这与本书到底有什么关系呢？
”　　毫无疑问，风格的表达与你在表达风格时灌注的哲学和思维方式是有很大关系的，这一点不管对于建筑学还是对于软件架构来说都同样适用。
我相信你们都见过像封面上国会大厦那样宏伟而华丽的软件，我同样相信你们也见过仅能工作而且一团乱麻似的软件。
从一个极端的角度来说，我相信你也见过许多过分追求风格反而弄巧成拙的华而不实之作和许多只顾尽快完成任务而毫无风格的“丑小鸭”（而且永远也不会变成天鹅）。
风格还是实用？
哪个更好？
不要太相信自己知道答案。
一方面，除非你给出一个明确的标准，否则“更好”只是一个无意义的评价。
对什么更好呢？
在哪些方面更好呢？
另一方面，答案几乎总是这两者的平衡，最开始总是“这取决于……”。
本书讨论的是如何在使用C++进行软件设计和实现的诸多细节方面找到最佳平衡点，如果更好地理解所拥有的工具和设施，弄清它们应该在什么时候应用。
快速回答：与四周索然无味的建筑相比，封面上的国会大厦是更好的建筑吗？
其建筑风格更好吗？
如果不加思索就给出答案，很可能你会说“当然”，但是别忘了，你还没有考虑其建造和修缮的代价呢：　　？建造。
在1902年竣工之时，它是当时世界上最大的国会大厦。
人们花费了难以想像的时间、不计其数的人力物力来兴建它，以至于许多人称它为“白象（white elephant）”，意思是耗资过大的美丽事物。
考虑这样一个问题：比较起来，花费同样的投资能够建造多少幢周围那不美观、单调、或许干脆是令人厌烦的实用建筑？
如果你是在一个工程进度压力远比这座国会大夏建造时代要大得多的行业工作，你又会怎么做？
？修缮。
你们中那些熟悉这座建筑的人会注意到照片中的建筑正在进行修缮翻新，而且这个工作已经持续了好多年，其间又极有争议地花费了巨量的资金。
然而除了最近的这轮昂贵的修缮之外，之前还有多次修缮，因为遗憾的是，这座建筑外墙上的精美雕刻是由错误的材料制成的，其材料太过柔软了。
在大楼建成后不久，这些雕刻成了人们持续进行修缮的主要内容，它们逐渐被替换为更为坚固而耐久的材料，这些华丽之物的大规模修缮自从20世纪初开始就一直没停过，持续了近一个世纪。
软件开发中的情形也与此类似，重要的是在建造的代价和获得的功能之间、在优雅与可维护性之

间、在发展的潜在可能与过分追求华丽之间寻求合理的平衡。
使用C++来进行软件设计和架构时，我们每天都得面对这些类似的权衡。
在本书中讨论的问题当中有这样几个问题：使代码成为异常安全的就意味着将它变得更好了吗？
如果是这样的，那么这里所谓的“更好”是指什么意义上的呢？
什么时候它可能不是“更好”的呢？
在本书中你会得到明确的答案。
封装呢？
封装是否令软件变得更好？
为什么？
什么时候封装反倒不能令你的软件变得更好？
如果你想知道答案，继续往下读。
内联是一项有益的优化吗？
内联是什么时候进行的呢？
（你在回答这个问题的时候可得十分小心了。
）C++中的模板导出（export）特性与封面上的国会大厦有什么共通之处呢？
std::string跟多瑙河畔的巨型建筑又有何共通之处呢？
最后，在考虑了许多C++技术和特性之后，本书的最后，我们会用一整个部分来考察源自公开发布的代码中的几个实际例子，看看代码的作者在哪些方面做得好，在哪些方面做得不好，以及什么样的替代方案可能在实用性与良好的C++风格之间取得更好的平衡。
我希望本书以及Exceptional C++系列的其他图书能够开阔你的视野，增加你有关许多细节及其相互关系的知识，让你进一步了解到如何在编写自己的软件时找到合理的平衡点。
请再看一眼封面上的照片，在照片的右上方，你会看到一个热气球。
如果我们乘坐那样的热气球飞越城市的上空，整个城市的景色将尽收眼底。
我们会看到风格跟实用是如何相互影响相互依存的，我们也会知道如何去进行权衡并找到合理的平衡点，所有的决策将各得其所，构成一个富于生机的整体。
是的，我想布达佩斯是一个伟大的城市——充满着如此丰富的历史底蕴，充满着不尽的神秘喻义。
伟大的苏格拉底　　古希腊哲学家苏格拉底通过问问题来教他的学生们，他精心准备的问题是为了引导并帮助学生从已知的知识引出结论，并说明他们所学的东西是如何彼此相关、如何与他们现有的知识有着千丝万缕的联系的。
这种教学方式后来变得如此有名，以至于我们称它为“苏格拉底方法”。
而从学生的角度来看，苏格拉底的这种著名的教学方法能够吸引我们，让我们思考，并帮助我们从已知的东西出发去引出新的东西。
本书跟它的前面几本书（Exceptional C++ [Sutter00]和More Exceptional C++ [Sutter02]）一样，正是借鉴了苏格拉底的做法。
本书假定你在编写C++产品代码已有一些经验，书中使用了一种问答的形式来告诉你如何有效地利用标准C++及其标准库，特别地，我们将关注的中心放在如何用现代C++中实施可靠的软件构造上。
书中的许多问题都是从我以及其他人在编写C++产品代码时遇到的问题当中提炼出来的。
问题的目标是帮助你从已知的以及刚学到的东西出发得出结论，并展示它们之间如何关联。
书中给出的问题会展示如何对C++设计和编程问题作出理性的分析和判断，其中有些只是常见问题，有些不是那么常见；有些是非常单纯的问题，而有些则深奥一些；另外还有几个问题之所以放在书中只是因为……因为它们比较有趣。
本书涉及了C++的方方面面。
我的意思并不是说它触及了C++的每个细枝末节（那可需要多得多的篇幅了），我只不过是说它是从C++语言和库特性这块大调色板上取色，并描绘出一幅图景，展示那些看似毫无瓜葛的特性如何编织到一起，从而构成常见问题的一个个漂亮解决方案。
另外，本书还展示了那些看似无关的部分是如何互相之间盘根错节、存在着千丝万缕的联系的（即便

有时你也许并不希望它们之间有什么联系），以及如何去处理这些复杂关系。
你会看到一些关于模板和名字空间的讨论，也会看到一些关于异常与继承的讨论，同样，另外还有关于坚实的类设计和设计模式的讨论，关于泛型编程与宏技巧的讨论等等。
此外，还有一些实实在在的（而不是一些花边新闻式的边栏小字）条款是用来向你展示现代C++中的所有这些部分之间的相互关系的。

Exceptional C++ Style遵循了Exceptional C++和More Exceptional C++前两本书的传统：它通过短小精悍的条款的组织形式，并将这些条款再进一步分组为一个个的主题来介绍新东西。
读过我的第一本书的读者会发现一些熟悉的主题，不过现在包含了新的东西，诸如异常安全、泛型编程以及优化和内存管理技术。
我的几本书在主题上有部分重叠，但内容上并没有重复。
本书沿袭了对泛型编程和高效使用标准库的一贯强调态度，包括一些重要的模板和泛型编程技术的讨论。

书中的大多数条款最初出现在杂志专栏上以及网上，尤其是我为C/C++ Users Journal和Dr. Dobbs Journal，已停刊的C++ Report以及其他期刊所写的专栏文章，另外还有我的Guru of the Week[GotW]问题63到86。
不同的是本书中的材料与最初的版本相比经过了重大的修订、扩展、校正和更新，因此这本书（以及www.gotw.ca网站上的勘误表）应该被当成原先那些文章的最新而且权威的版本。

预备知识　我假定读者已经知道一些C++的基础知识。
如果不是这样，那就先去阅读一些好的关于C++的介绍和概述的文章或书籍。
像Bjarne Stroustrup的《The C++ Programming Language》[Stroustrup00]或者Stan Lippman和Josee Lajoie合著的《C++ Primer（第三版）》[Lippman98] 这样的经典是良好的选择。
接下来，一定要选择一本Scott Meyers的经典书籍《(More)Effective C++》[Meyers96,Meyers97]这样的风格指南，我发现这两本书基于Web浏览方式的CD版本[Meyers99]比较方便实用。

如何阅读本书　本书中的每一条都是以一个谜题或问题来展开的，都有一个介绍性的标题，如下：　##. 条款的主题 难度：#　一段简短的介绍性文字，说明该条将要讨论的内容。

条款主题大致告诉你本条讨论的是什么，通常后面会跟有介绍性的/回顾性的问题[JG问题，JG是指新来的、级别较低的军官（少尉）]，然后就是主要问题（即Guru问题）。
注意，难度系数只是我对特定主题对大多数读者而言的难度所作的一个大致推测，这就是说你可能会发现一个难度为7的问题对你来说却比一个难度为5的问题要来得简单。
实际上我的前两本书：Exceptional C++ [Sutter00]和More Exceptional C++ [Sutter02]就曾不断地收到一些读者来信说：“ 嗨！
第N条比它实际上要更难（简单）！
”不同的人对于“ 简单 ”的评价标准各有不同。
所谓难度系数只是因人而异的，任何条款的难度实际上都是取决于你所掌握的知识和经验，而其他人则可能觉得它更容易或更难。
不过大多数情况下应当将我给出的难度系数作为一个合理的指示，让你能够知道下面会出现什么问题。

你可能会选择从头至尾按顺序阅读本书，这很好，但是不是非要这么做。
你可能决定读某个特定部分的所有条款，因为对该部分的主题特别感兴趣；这也没关系。
一般来说书中的所有条款都是基本独立的，除非标注有“ 之一 ”、“ 之二 ”等的条款之间才会有紧密联系。
因此你在阅读本书的时候完全可以以跳跃式的方式，顺着条款中的交叉引用（包括对我前两本书的引用）来阅读。
惟一需要注意的地方就是，标注了“ 之几 ”的连续几个章节之间互有关联，构成了一个主题，除此之外其他条款你完全可以自由选择阅读。

除非我注明某段代码是一个完整的程序，否则它就不是。
记住，代码示例通常只是从一个完整程序中摘取出来的一小段代码，不要期望它们都能够独立编译。

一般来说你得为其添上一副骨架才能够使其成为一个完整的可编译的程序。

最后，关于书中的URL：网上的东西总是在变，尤其是那些我无权干涉的东西。因此随意将某些网站地址放在一本刊印的书籍中可不大妥当，因为恐怕在书付印之前有些地址就已经作废了，更不用说在这本书在你书架上躺了几年之后了。

所以说，当我在书中引用其他人的文章或网址的时候，我给出的地址是链接到我自己网站（即www.gotw.ca）上相关内容的地址，后者我是可以控制的，因此我可以在我网站上的相关网页上随时作相应更新，让其中的相关地址指向实际存在的网页。

几乎所有在书中引用到的其他人的作品我都放在参考书目里了，而且还在我的网站上也放置了一份副本，其中的链接都是有效的。

如果你发现本书中的链接无效了，请发电子邮件向我告知，我会在网站上更新相关的链接（如果我可以找到新地址的话），或者注明链接已经失效（如果我无法找到新地址的话）。

无论如何，虽说书一旦印刷便白纸黑字，不可再改，但我网站上的相关内容会保持更新。

媒体关注与评论

本书是围绕实际问题及其解决方案展开论述的，对一些至关重要的C++细节和相互关系提出了新的见解，为当今的关键C++编程技术（如泛型编程、STL、异常安全等）提供了新的策略。
读者会在书中找到下列问题的答案。

- 可以从STL本身学习哪些库设计的知识？
- 如何避免削弱甚至损害模板代码的通用性？
- 为什么不应该对函数模板进行特化？

正确的做法是什么？

- 异常安全如何超越TRY和CATCH语句？
- 什么情况下应当“泄漏”一个类的私有成分，怎么做？
- 如何让一个类不受版本变动的影响？
- 使用标准库容器的实际内存开销是多少？
- 如何使用CONST才能真正优化代码？
- 内联对程序性能有何影响？
- 为什么有些看似错误的代码实际上却能够通过编译且运行得很好？

为什么我们要关心这种情况？

- STD：：STRING的设计有什么问题？

本书使读者能够在设计、架构和编码的过程中保持良好的风格，从而编写出更健壮和更高效的C++软件。

版权说明