

深入 C++ 系列

C++ FAQs Second Edition

C++ 经典问答 (第二版)

Marshall Cline Greg Lomow Mike Girou 著
周远成 译



深入C++系列

C++ FAQs Second Edition

C++ 经典问答 (第二版)

Marshall Cline Greg Lomow Mike Girou 著
周远成 译

中国电力出版社

C++FAQs SECOND EDITION (ISBN 0-201-30983-1)

Marshall Cline, Greg Lomow, Mike Girou

Authorized translation from the English language edition, entitled Generic Programming and the STL, published by Addison Wesley, Copyright©1999

All rights reserved. NO part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

**CHINESE SIMPLIFIED language edition published by China Electric Power Press
Copyright©2003**

本书由美国培生集团授权出版。

北京市版权局著作权合同登记号 图字：01-2000-3088 号

图书在版编目（CIP）数据

C++ 经典问答：第2版 / （美）克林等编著；周远成译．—北京：中国电力出版社，2002.2

（深入C++ 系列）

ISBN 7-5083-0868-9

I .C... II. ①克...②周... III. C语言—程序设计—问答 IV. TP312-44

中国版本图书馆CIP数据核字（2002）第002439号

责任编辑：朱恩从

丛 书 名：深入C++ 系列

书 名：C++ 经典问答：第2版

编 著：（美）Marshall Cline

翻 译：周远成

出版发行：中国电力出版社

地址：北京市三里河路6号 邮政编码：100044

电话：（010）88515918 传真：（010）88423191

印 刷：北京地矿印刷厂

开 本：787×1092 1/16 印 张：35.5 字 数：782千字

版 次：2003年5月北京第一版

印 次：2003年5月第一次印刷

定 价：55.00 元

致 谢

第二版体现了许多人的建议和帮助，在这些年来 Bjaren Stroustrup 和 Andrew Koenig 给我们提出了许多建议和妙想，我们为此要特别地感激他们。Paul Abrahams、Michael Ball、Ian Long、John Kwan、Jason Pritchard、Christopher Van Wyk 和 Steve Vinoski 对该书的出版做了许多有益的工作。我们感谢以下这些同事和朋友们的支持和见解，他们是 Mathew Denman、Mike Ferretti、Peter Jakab、Charles Martin、Robert Martin、Chris Rooney、Dave Stott、Ioannis Tollis、John Vlissides、Jim Watson、Claudia Woody、Demetrios Yannakopoulos 以及 Howard Young。Debbie Lafferty 对该书的两个版本的都给予了极大的支持，对于她的帮助和耐心，我们真的是感激不尽。

David W. Bray 向我 (Marshall) 介绍了自动定向思想的现状，Doug Lea 在每天上午 5:30 接发 Email (你获得优++)，Jamshid Afshur 和 Jr. 通过 Email 发送了许多建议，以及我在 Clarkson 大学的同事都给予了我很大的帮助，在此向以上人员表示特别的感激。对于我的学生和许许多多电子 FAQ 的支持者，我也在此向你们表示感谢。此外，我还要特别要感谢 MaryElaine、David、Elizabeth、Gabrielle、Peter、Katherine 和 Brandon，你们所做的工作是非常有价值的。

Greg 在此向 Brian Unger 和 Graham Birtwistle 表示特别的感谢，你们早在 1981 年就向我介绍了仿真 67 和面向对象编程，而那时面向对象编程还没流行。Brian Unger 和 Marshall Cline 为我们提供了在仿真工作环境方面从事有趣项目的机会，我 (Greg) 在此向他们表示感谢。另外还要对卡尔加里 (Calgary) 大学的同事、Jade Simulations、Paradigm Shift 公司以及 MT 系统公司在这些年来提供的帮助与支持表示感谢。Barb，谢谢！你对我的工作给予了大力支持并且忍受了我非同寻常的工作安排。

Dix Pettey 告诉我 (Mike) 到处都在研究什么，John Schmidt 教我做事要切合实际，Len Gollobin 向我介绍了如何以正确的方式考虑问题，在此我要向以上人员表示特别的感谢。由于个人和业务方面的原因，密苏里大学 (Missouri) 的数学系在我的心目中总是占有特殊的位置。感谢我的孩子 Beverly 和 James，因为他们容忍他们的爸爸所做工作不总是符合他们的打算，最后要感谢我的特殊朋友 Christian、Kelly 和 Tamie，因为他们是我生命的组成部分。

Marshall

Greg

Mike

目 录

致 谢

第 1 部分 开 端

第 1 章 概述	3
FAQ1.01 这一章的目的是什么?	3
FAQ1.02 C++FAQ 是什么?	3
FAQ1.03 本书的读者对象是谁?	4
FAQ1.04 这是一本纯粹关于 C++的书吗?	4
FAQ1.05 开发人员为什么需要一本 C++和 OO 技术的指导书?	5
FAQ1.06 在对这些 FAQ 的解答方面给出了哪种类型的指导?	5
FAQ1.07 电子 FAQ 是什么? 当电子 FAQ 可免费得到时 为什么要买这本书?	6
FAQ1.08 假如已拥有该书的第一版, 为什么还要买这一版?	6
FAQ1.09 在这本书里都使用了哪些惯例?	7
第 2 章 基本的 C++语法和语义学	9
FAQ2.01 这一章的目的是什么?	9
FAQ2.02 main()的要点是什么?	9
FAQ2.03 函数的要点是什么?	10
FAQ2.04 默认参数的要点是什么?	11
FAQ2.05 局部(自动)对象的要点是什么?	11
FAQ2.06 使用显式参数构造对象的要点是什么?	12
FAQ2.07 动态分配(new)对象的要点是什么?	13
FAQ2.08 内部作用域中的局部对象的要点是什么?	15
FAQ2.09 通过引用传递对象的要点是什么?	16
FAQ2.10 通过数值传递对象的要点是什么?	17
FAQ2.11 通过指针传递对象的要点是什么?	17
FAQ2.12 输出流的要点是什么?	19
FAQ2.13 输入流的要点是什么?	20
FAQ2.14 使用包含重载运算符的类的要点是什么?	21
FAQ2.15 使用容器类的要点是什么?	22
FAQ2.16 创建类的头文件的要点是什么?	23

FAQ2.17	定义一个类的要点是什么？	24
FAQ2.18	定义成员函数的要点是什么？	26
FAQ2.19	往类中增加一个构造函数的要点是什么？	27
FAQ2.20	往类中增加一个析构函数的要点是什么？	29
FAQ2.21	定义一个包含指向堆中对象的指针的类的要点是什么？	30
FAQ2.22	全局对象的要点是什么？	32
FAQ2.23	抛出和捕捉异常的要点是什么？	34
FAQ2.24	继承和动态绑定的要点是什么？	36
第 3 章 领悟管理观点		39
FAQ3.01	这一章的目的是什么？	39
FAQ3.02	这一章（和这本书）的核心要旨是什么？	39
FAQ3.03	为什么是经理而不是懂技术的开发人员负责？	40
FAQ3.04	如何能管理不熟悉的事物？	41
FAQ3.05	关于 C++ 和 OO 设计的最普遍的错误是什么？	41
FAQ3.06	“Peter 软件原理”是什么？	42
FAQ3.07	机构应该在它的所有项目上都使用 OO 吗？	42
FAQ3.08	在 OO 离去之前能忽略它吗？	43
FAQ3.09	什么样的 OO 语言是最好的？	43
FAQ3.10	对过程和工具来说正确的方法是什么？	44
FAQ3.11	就现有的类库和框架来说，正确的拥有方法是什么？	45
第 4 章 体系结构观点		46
FAQ4.01	这一章的目的是什么？	46
FAQ4.02	软件体系结构为什么是重要的？	46
FAQ4.03	体系结构应该以什么为基础，是被解决的问题还是问题域？	47
FAQ4.04	软件体系结构应基于问题的策略吗？	48
FAQ4.05	客户可能改变他们的要求吗？	48
FAQ4.06	稳定的要求是令人满意的吗？	49
FAQ4.07	打算变化的关键是什么？	49
FAQ4.08	框架是什么？	50
FAQ4.09	由框架表现的“控制的倒置（inversion of control）”是什么？	50
FAQ4.10	可扩充的特定域框架是什么？	51
FAQ4.11	什么特点才能使框架既是可扩充的又是特定域的？	51
FAQ4.12	假如域分析不正确又会发生什么？	52
FAQ4.13	应该花费多少努力来支持变化——即，可扩充性值多少钱？	52
FAQ4.14	设计师是如何使软件体系结构具有弹性的？	53
FAQ4.15	实现重复使用的秘密是什么？	54

第 2 部分 面向对象的设计

第 5 章 面向对象的基本原理	57
FAQ5.01 这一章的目的是什么?	57
FAQ5.02 为什么要采用 OO 技术?	57
FAQ5.03 使用 C++ 进行 OO 程序设计的好处是什么?	58
FAQ5.04 面向对象技术的基本概念是什么?	59
FAQ5.05 为什么类是重要的?	59
FAQ5.06 对象是什么?	60
FAQ5.07 对象的理想特性是什么?	61
FAQ5.08 类如何好于过程软件的三个基本构成块?	62
FAQ5.09 合成的目的是什么?	62
FAQ5.10 继承的目的是什么?	63
FAQ5.11 多态性和动态绑定的优点是什么?	64
FAQ5.12 OO 如何有助于产生弹性的和可扩充的软件?	65
FAQ5.13 旧代码如何调用新代码?	65
FAQ5.14 抽象是什么? 它为什么重要?	68
FAQ5.15 抽象应以用户为中心还是以开发人员为中心?	69
FAQ5.16 封装和抽象之间的区别是什么?	69
FAQ5.17 封装一个糟糕的抽象会有什么后果?	70
FAQ5.18 把接口和实现分开的意义是什么?	70
FAQ5.19 把接口和实现分开如何能改善性能以及弹性?	71
FAQ5.20 为抽象创建一个合适的最佳方法是什么?	72
FAQ5.21 get/set 成员函数是如何与蹩脚的设计接口相关的?	73
FAQ5.22 每个成员数据都要有 get 和 set 成员函数吗?	73
FAQ5.23 类的真正目的是输出数据吗?	74
FAQ5.24 OO 是以数据为中心吗?	74
第 6 章 可觉察行为的说明	75
FAQ6.01 这一章的目的是什么?	75
FAQ6.02 成员函数的用户应该依靠代码实际做的还是说明?	75
FAQ6.03 依靠说明而不是实现的优点是什么?	77
FAQ6.04 什么是通告要求和通告承诺?	78
FAQ6.05 如何确定成员函数的通告要求和通告承诺?	78
FAQ6.06 没有使用说明的开发组织为什么害怕变化?	80
FAQ6.07 开发人员如何判断一个的变化是否会干扰原有的代码?	81
FAQ6.08 在说明里可替代(向后兼容的)变化的特性是什么?	81
FAQ6.09 如何能表明成员函数的实现履行了其说明?	82
FAQ6.10 保持说明与代码同步可能吗?	83

第 7 章 合适的继承.....	85
FAQ7.01 合适的继承是什么?	85
FAQ7.02 合适继承的好处是什么?	87
FAQ7.03 不合适的继承是什么?	87
FAQ7.04 合适的和不合适的继承之间的不同很不明显吗?	88
FAQ7.05 替代性是基于代码所做的还是代码将要做的说明承诺?	88
FAQ7.06 废除（隐藏）一个继承的 public: 成员函数合适吗?	90
FAQ7.07 专门化是什么?	91
FAQ7.08 子集和合适的继承有什么联系?	91
第 8 章 检测并修改不合适的继承.....	93
FAQ8.01 不合适的继承能破坏一个项目吗?	93
FAQ8.02 学会避免不合适的继承的最好方法是什么?	93
FAQ8.03 直觉是理解正确继承的可依靠的向导吗?	94
FAQ8.04 鸵鸟是一种鸟吗?	94
FAQ8.05 覆盖虚拟函数应该抛出异常吗?	96
FAQ8.06 覆盖虚拟函数可以是空操作（no-op）吗?	98
FAQ8.07 为什么 C++ 修改 Ostrich/Bird 窘境如此困难?	100
FAQ8.08 Circle 应该从 Ellipse 继承吗?	100
FAQ8.09 对于非对称圆的窘境我们能做些什么?	102
FAQ8.10 在这些 FAQ 里一直困惑着我们的问题是什么?	103
FAQ8.11 Stack 应该从 List 继承吗?	104
FAQ8.12 代码重用是继承的主要目的吗?	106
FAQ8.13 东西的容器是一种任何东西的容器吗?	107
FAQ8.14 当水果袋允许放进任何种类的水果时，就可以说苹果袋是 一种水果袋吗?	108
FAQ8.15 停放小汽车的停车区是一种停放任意的运载工具的 停车区吗（假定停放运载工具的停车区允许停放任 意种类的运载工具）?	111
FAQ8.16 Derived 的数组是一种 Base 的数组（array-of Base）吗?	113
FAQ8.17 当 Derived 数组能作为 Base 数组传递时，意味着该数组 是糟糕的吗?	115
第 9 章 错误处理策略.....	116
FAQ9.01 适当的错误处理是根本过失的主要来源吗?	116
FAQ9.02 如何用 C++ 对运行时错误进行处理?	116
FAQ9.03 在抛出/捕捉过程期间，在展开的栈帧（stack frame） 里面的对象会发生什么?	117

FAQ9.04	什么是异常说明？	118
FAQ9.05	为了错误处理而使用返回码的缺点是什么？	118
FAQ9.06	throw...catch 的优点是什么？	119
FAQ9.07	为什么把正常逻辑从异常处理逻辑中分开是有帮助的？	120
FAQ9.08	使用异常处理时最困难的部分是什么？	123
FAQ9.09	函数应该在什么时候抛出异常？	124
FAQ9.10	对于异常对象的层次来讲最好的方法是什么？	126
FAQ9.11	异常类应该如何命名？	127
FAQ9.12	setjmp 和 longjmp 应放在 C++ 的什么地方？	128
第 10 章	测试策略	129
FAQ10.01	这一章的目的是什么？	129
FAQ10.02	自测对象的优点是什么？	129
FAQ10.03	人们通常因为什么而不把自测加进他们的对象中？	130
FAQ10.04	假如不使用在这里描述的技术将会发生什么情况？	131
FAQ10.05	什么时候类才是正确的？	131
FAQ10.06	什么是行为自测？	132
FAQ10.07	类的不变式是指什么？	134
FAQ10.08	为什么应明确地捕获不变式？	135
FAQ10.09	应该在什么时候调用 testInvariant() 成员函数？	136
FAQ10.10	我们能为确保对象不被野蛮指针转移做些什么？	137

第 3 部分 语 言 组 件

第 11 章	引用	141
FAQ11.01	引用是什么？	141
FAQ11.02	“引用物”意味着什么？	142
FAQ11.03	什么时候才能把引用与它的引用物连接起来？	142
FAQ11.04	把一个值赋给一个引用时会发生什么？	143
FAQ11.05	局部引用是什么？	143
FAQ11.06	返回一个引用意味着什么？	144
FAQ11.07	使用引用地址的结果是什么？	145
FAQ11.08	引用能指向其他引用物吗？	146
FAQ11.09	当指针能做引用能做的所有事情时，为什么还要使用引用？	146
FAQ11.10	引用不是伪装的指针吗？	147
FAQ11.11	什么时候需要指针？	147
FAQ11.12	为什么一些人会憎恨引用？	148
FAQ11.13	int& const x 有意义吗？	148

第 12 章 New 和 Delete	150
FAQ12.01 new 比分配内存做得更多吗?	150
FAQ12.02 为什么说 new 好于可靠的、老的、值得信赖的 malloc()?	150
FAQ12.03 在 C++中有与 new 和 delete 类似的 realloc()的等价物吗?	151
FAQ12.04 从 new 返回的指针能用 free()解除分配吗? 从 malloc() 返回的指针能用 delete 解除分配吗?	151
FAQ12.05 delete p 删除的是指针 p 还是引用物*p?	152
FAQ12.06 应该检查从 new Fred()返回的指针以便发现它是否是 NULL 吗?	152
FAQ12.07 如何让 new 返回 NULL 而不是抛出一个异常?	153
FAQ12.08 如何使 new 在内存用到下限时自动地刷新再循环对象池?	154
FAQ12.09 当 p 是 NULL 时, 调用 delete p 会发生什么?	157
FAQ12.10 如果对同一个指针执行两遍删除操作会发生什么?	158
FAQ12.11 如何能分配和解除分配一个对象数组?	158
FAQ12.12 假如使用 delete p (不是 delete[] p) 来删除一个通过 new Fred[n]分配的数组将会发生什么?	159
FAQ12.13 当 p 指向某种内置类型 (比如 char) 的数组时能省略 delete[] p 中的[]吗?	160
FAQ12.14 如何在一个预定的内存位置建立一个对象?	161
FAQ12.15 Fred 类如何保证只是用 new 而不是在栈上创建 Fred 对象?	161
FAQ12.16 如何释放由布局 new 创建的对象?	162
FAQ12.17 在 p = new Fred()里, 假如 Fred 构造函数抛出一个异常, 那么 Fred 内存会“泄漏”吗?	163
FAQ12.18 成员函数使用 delete this 是合法 (和规范) 的吗?	165
FAQ12.19 在 p = new Fred[n]之后, 在 delete[] p 期间编译器如何知道 要释放 n 个对象?	166
第 13 章 inline 函数	167
FAQ13.01 inline 函数的作用是什么?	167
FAQ13.02 关键字“inline”和“inlined”函数之间的联系是什么?	168
FAQ13.03 关于直接插入有什么特殊的规则吗?	168
FAQ13.04 一次定义规则 (ODR) 是什么?	169
FAQ13.05 就 inline 函数来说要考虑哪些性能?	169
FAQ13.06 内联函数能改善性能吗?	170
FAQ13.07 内联函数能增加可执行代码的大小吗?	171
FAQ13.08 为什么不应在第一次编写代码时做直接插入的决定?	172
FAQ13.09 当程序员使用从第三方获得的直接内联函数时会发生什么?	172
FAQ13.10 inline 和非 inline 代码之间的转换有没有容易的方法?	173

第 14 章	const 正确性	176
FAQ14.01	如何看懂指针的声明?	176
FAQ14.02	C++程序员应如何避免对对象做出不期望的变化?	176
FAQ14.03	const 意味着运行时的开销吗?	178
FAQ14.04	const 允许编译器产生更有效的代码吗?	178
FAQ14.05	const 正确性是冗长乏味的吗?	178
FAQ14.06	为什么实施 const 正确性应该尽早而不尽晚?	179
FAQ14.07	检查程序和变态程序 (mutator) 之间的区别是什么?	180
FAQ14.08	应该在什么时候将一个成员函数声明为 const?	181
FAQ14.09	const 是应用于对象的位状态还是对象的抽象状态?	182
FAQ14.10	在什么情况下不应在声明形参的过程中使用 const?	183
FAQ14.11	在什么情况下不应在声明函数返回类型的过程中使用 const?	183
FAQ14.12	在 const 成员函数内如何修改“不能观察到的”数据成员?	184
FAQ14.13	如果对对象有一个 const 引用 (指针), 那么我们还能 合法地改变该对象吗?	186
FAQ14.14	const_cast 意味着失去优化机会吗?	187
第 15 章	名称空间	188
FAQ15.01	这一章的目的是什么?	188
FAQ15.02	名称空间是什么?	188
FAQ15.03	名称空间外的代码如何使用在名称空间内声明的名字?	190
FAQ15.04	若两个名称空间包含相同的名字会发生什么?	191
FAQ15.05	名称空间的使用规则是什么?	192
FAQ15.06	名字查找是什么?	192
FAQ15.07	使用来自名称空间特别是标准名称空间的名字的各种技术 之间的折衷办法是什么?	193
FAQ15.08	名称空间能中断代码吗?	194
FAQ15.09	名称空间还有其他的应用吗?	195
FAQ15.10	名称空间如何解决长标识符的问题?	195
第 16 章	使用 static	196
FAQ16.01	这一章的目的是什么?	196
FAQ16.02	什么是静态类成员?	196
FAQ16.03	可以将静态数据成员类比成什么?	197
FAQ16.04	inline 函数能安全地访问静态数据成员吗?	199
FAQ16.05	可将静态成员函数类比成什么?	201
FAQ16.06	静态数据成员与全局变量有多类似?	203
FAQ16.07	静态成员函数是如何类似于友元函数的?	203

FAQ16.08	什么是命名构造函数的习惯语法？	204
FAQ16.09	应该如何调用静态成员函数？	205
FAQ16.10	为什么带有静态数据成员类会出现连接程序错误？	206
FAQ16.11	如何初始化 <code>const</code> 静态数据成员？	206
FAQ16.12	实现维护调用之间状态的函数的正确策略是什么？	208
FAQ16.13	函数调用运算符如何帮助函数类？	209
FAQ16.14	忽略静态初始化次序问题是安全的吗？	210
FAQ16.15	解决静态初始化次序问题的简单而又健壮的方法是什么？	212
FAQ16.16	假如静态对象的析构函数有最后一定要出现的重要的 副作用怎么办？	213
FAQ16.17	假如静态对象的析构函数有最终一定发生的重要的副作用，并且 另一个静态对象的析构函数必须访问该静态对象将怎么办？	214
FAQ16.18	在前面介绍的各种技术之间进行选择的准则是什么？	216
第 17 章 派生类		218
FAQ17.01	这一章的目的是什么？	218
FAQ17.02	C++如何表达继承？	218
FAQ17.03	具体的派生类是什么？	220
FAQ17.04	为什么派生类不能访问其基类的 <code>private:</code> 成员？	221
FAQ17.05	基类如何保护派生类以致于基类的变化不会影响到它们？	222
FAQ17.06	能把一个派生类指针转变成它的公共基类的指针吗？	223
FAQ17.07	Y 类如何既是 X 类的子类又能得到 X 的存储单元？	224
FAQ17.08	如何既不使 Y 成为 X 的子类又能使 Y 类获得当前 X 类的 存储单元？	224
FAQ17.09	Y 类如何能既不继承 X 的存储单元又是 X 类的子类？	225
第 18 章 访问控制		227
FAQ18.01	这一章的目的是什么？	227
FAQ18.02	<code>private:</code> 、 <code>protected:</code> 和 <code>public:</code> 之间有何不同的？	227
FAQ18.03	为什么子类不能访问其基类的 <code>private:</code> 部分？	228
FAQ18.04	关键字 <code>struct</code> 和 <code>class</code> 之间有什么不同？	228
FAQ18.05	什么时候数据成员应该是 <code>protected:</code> 而不是 <code>private:</code> ？	229
FAQ18.06	为什么 <code>private:</code> 是类的默认访问权限？	229
第 19 章 友元类和友元函数		231
FAQ19.01	友元是什么？	231
FAQ19.02	好的友元类的思维模型应是什么样？	232
FAQ19.03	使用友元类的优点是什么？	233

FAQ19.04	友元类会破坏封装屏障吗？	233
FAQ19.05	友元函数是什么？	234
FAQ19.06	什么时候应该把函数实现为友元函数而不是成员函数？	235
FAQ19.07	保证正确使用友元函数的方针是什么？	235
FAQ19.08	友谊不能传递意味着什么？	236
FAQ19.09	友谊不能继承意味着什么？	237
FAQ19.10	友元不能是虚拟的意味着什么？	239
FAQ19.11	在什么情况下应该使用友元函数而不是成员函数？	240
FAQ19.12	友元函数应被声明在一个类的 <code>private</code> 、 <code>protected</code> ： 还是 <code>public</code> 部分？	243
FAQ19.13	私有类是什么？	243
FAQ19.14	如何输出类的对象？	244
FAQ19.15	类的对象如何接收输入流？	245
第 20 章 构造函数和析构函数		246
FAQ20.01	构造函数的作用是什么？	246
FAQ20.02	在 C++ 中，构造函数的规则是什么？	247
FAQ20.03	析构函数的作用是什么？	247
FAQ20.04	在 C++ 中，析构函数的规则是什么？	248
FAQ20.05	当执行一个析构函数时会发生什么？	248
FAQ20.06	拷贝构造函数的目的是什么？	250
FAQ20.07	应在什么时候调用拷贝构造函数？	251
FAQ20.08	什么是“默认构造函数”？	252
FAQ20.09	构造函数应该调用另一个构造函数作为原语吗？	254
FAQ20.10	派生类的析构函数必须明确地调用其基类的析构函数吗？	254
FAQ20.11	如何在一个局部对象的函数结束前释放该对象？	255
FAQ20.12	为类提供直观的、多重构造函数的好方法是什么？	258
FAQ20.13	当基类的构造函数调用一个虚拟函数时， 为什么不调用覆盖函数？	259
FAQ20.14	当基类的析构函数调用一个虚拟函数时， 为什么不调用覆盖函数？	261
FAQ20.15	布局 <code>new</code> 的作用是什么？	262
第 21 章 虚拟函数		264
FAQ21.01	这一章的目的是什么？	264
FAQ21.02	虚拟成员函数是什么？	264
FAQ21.03	和调用普通的函数相比，调用一个虚拟函数的 花费有多大？	265

FAQ21.04	C++如何在支持动态绑定 (dynamic binding) 的时候 执行静态定型?	265
FAQ21.05	析构函数能是虚拟的吗?	266
FAQ21.06	虚拟析构函数的作用是什么?	266
FAQ21.07	什么是虚拟构造函数?	267
FAQ21.08	构造函数或者析构函数在其对象里调用虚拟函数时 应该使用什么语法?	270
FAQ21.09	在调用虚拟成员函数时应该使用作用域运算符::吗?	272
FAQ21.10	什么是纯虚拟成员函数?	273
FAQ21.11	能在声明纯虚拟函数的类中定义该函数吗?	273
FAQ21.12	当一个虚拟析构函数没有代码时应如何定义它?	274
FAQ21.13	ABC 能有一个纯虚拟析构函数吗?	275
FAQ21.14	如何阻止编译器产生直接插入虚拟函数的 重复轮廓的拷贝?	276
FAQ21.15	带有虚拟函数的类至少有一个非直接插入的虚拟函数吗?	276
第 22 章 初始化列表		278
FAQ22.01	什么是构造函数的初始化列表?	278
FAQ22.02	如果没有使用构造函数初始化列表将会发生什么?	279
FAQ22.03	在构造函数的定义里使用初始化列表的原则是什么?	280
FAQ22.04	构造函数的体内什么都没有是正常的吗?	280
FAQ22.05	如何初始化一个 const 数据成员?	282
FAQ22.06	如何初始化一个引用数据成员?	283
FAQ22.07	初始化表达式 (initializers) 是以它们出现在初始化 列表中的顺序被执行的吗?	283
FAQ22.08	如何在一个构造函数的初始化列表里安排初始化表达式?	285
FAQ22.09	在构造函数的初始化列表里用一个成员对象来初始化 另一个成员对象是可能的吗?	286
FAQ22.10	假如某一个成员对象必须用另一个成员对象来初始化 它自己又该怎么办?	288
FAQ22.11	“在一个初始化列表里初始化所有成员对象” 的规则有例外吗?	289
FAQ22.12	如何用明确的初始表达式来初始化一个对象数组?	290
第 23 章 运算符重载		292
FAQ23.01	重载运算符像正常的函数吗?	292
FAQ23.02	什么时候应该使用运算符重载?	293
FAQ23.03	什么样的运算符不能重载?	294

FAQ23.04	使类更易于理解是运算符重载的目的吗?	295
FAQ23.05	为什么下标运算符通常成对出现?	295
FAQ23.06	对于+=、+=和-=这样的运算符,最重要的是考虑什么?	297
FAQ23.07	如何区分 operator++的前缀和后缀两种形式?	298
FAQ23.08	operator++的前缀和后缀形式应该返回什么?	299
FAQ23.09	类似矩阵的类如何能产生带有两个或者更多下标的 下标运算符?	300
FAQ23.10	**运算符能当作指数运算符吗?	301
第 24 章 赋值运算符		303
FAQ24.01	赋值运算符应该返回什么?.....	303
FAQ24.02	对象自己给自己赋值有错吗?	304
FAQ24.03	关于自我赋值应该做些什么?	305
FAQ24.04	在部分地赋值一个对象之后,赋值运算符是否 该抛出一个异常?	306
FAQ24.05	如何在 ABC 中声明赋值运算符?	308
FAQ24.06	用户定义的赋值运算符什么时候应该模仿编译 器自动产生的赋值运算符?.....	310
FAQ24.07	private: 和 protected: 赋值运算符返回的是什么?	310
FAQ24.08	增加编译器合成的赋值运算符的技术存在吗?	311
FAQ24.09	如何在派生类内使用赋值运算符?	311
FAQ24.10	ABC 的赋值运算符能够是虚拟的吗?	313
FAQ24.11	假如基类的赋值运算符是虚拟的,那么派生类 应该做什么?	315
FAQ24.12	应该通过使用布局 new 和拷贝构造函数来实现 赋值运算符吗?	317
第 25 章 模板		318
FAQ25.01	模板的作用是什么?	318
FAQ25.02	类模板的语法和语义是什么?	318
FAQ25.03	如何为处理特殊的情况而专门化模板类?	320
FAQ25.04	函数模板的语法和语义是什么?	323
FAQ25.05	模板应该使用 memcpy()来拷贝其模板变元的对象吗?.....	324
FAQ25.06	当一个模板在它的内部使用另一个模板时为什么 编译器会对>>提出警告?	326
第 26 章 异常策略		328
FAQ26.01	在 try 块中都有什么?	328
FAQ26.02	函数应该在什么时候捕获异常?	328

FAQ26.03	catch 块应该完全地从一个错误中恢复过来吗？	329
FAQ26.04	构造函数如何处理错误？	331
FAQ26.05	僵死 (zombie) 对象是什么 (为什么应该避免它们)？	332
FAQ26.06	假如对象的某一成员对象在其构造函数期间能够抛出异常， 该对象应该做些什么？	332
FAQ26.07	当析构函数失败时它们应该抛出异常吗？	334
FAQ26.08	析构函数应该调用可能抛出异常的子程序吗？	335
FAQ26.09	资源释放分配原语应该通过抛出一个异常发出错误的 信号吗？	336
FAQ26.10	terminate()函数都做些什么？	336
FAQ26.11	unexpected()函数都做些什么？	337
FAQ26.12	在什么情况下覆盖虚拟成员函数抛出的异常， 会与基类里成员函数说明中列出的异常不同？	339
FAQ26.13	异常处理机制是如何引起一个程序默默地崩溃的？	341

第 27 章 类型和 RTTI 342

FAQ27.01	这一章的目的是什么？	342
FAQ27.02	什么是静态类型检查？	342
FAQ27.03	什么是动态类型检查？	343
FAQ27.04	动态类型检查的基本问题是什么？	345
FAQ27.05	如何避免使用动态类型检查？	346
FAQ27.06	对于动态类型检查有没有更好的方案？	346
FAQ27.07	什么是能力询问 (capability query)？	347
FAQ27.08	带有容器的动态类型检查的可选方案是什么？	348
FAQ27.09	有没有动态类型检查是必要的情况？	349
FAQ27.10	假定知道指向 ABC 的指针，那么如何才能查找 引用物的类？	349
FAQ27.11	什么是 downcast？	351
FAQ27.12	使用向下的类型转换的可选方案是什么？	353
FAQ27.13	为什么向下的类型转换是危险的？	355
FAQ27.14	C++体系的继承图应该是高的还是矮的？	355
FAQ27.15	C++体系的继承图应该是单块的还是一个森林？	355
FAQ27.16	什么是运行时类型标识 (RTTI)？	356
FAQ27.17	dynamic_cast<T>()的作用是什么？	356
FAQ27.18	dynamic_cast<T>()是一个包治百病的灵药吗？	357
FAQ27.19	static_cast<T>()的作用是什么？	358
FAQ27.20	typeid()的作用是什么？	358

FAQ27.21	类型安全的向下类型转换 (type-safe downcast)	
	有什么隐藏的代价吗?	359
第 28 章	容器	360
FAQ28.01	容器类是什么? 与容器类有关的最普通的错误是什么?	360
FAQ28.02	数组是福还是祸?	361
FAQ28.03	应用程序开发组织应该创建它们自己的容器类吗?	361
FAQ28.04	指针容器的普通错误都是什么?	362
FAQ28.05	这意味着应该避免使用指针容器吗?	362
FAQ28.06	可靠的、老式的 char* 肯定是一个例外, 对吗?	363
FAQ28.07	auto_ptr<T> 能简化指针容器的所有权问题吗?	363
FAQ28.08	像 Java 的 Object 类能简化 C++ 的容器吗?	363
FAQ28.09	同类和异类容器之间有什么不同?	364
FAQ28.10	用“最好种类”的观点来选择容器类是一个好主意吗?	364
FAQ28.11	所有项目都应使用 C++ 的标准化容器吗?	365
FAQ28.12	什么是 C++ 标准化的容器类?	366
FAQ28.13	什么是标准化的 C++ 数列容器类的最好应用?	366
FAQ28.14	标准 C++ 关联容器类 (Associative Container Class)	
	的最好应用场合是什么?	369

第 4 部分 专 题

第 29 章	把继承和重载混合在一起	373
FAQ29.01	重载函数和覆盖函数之间有什么不同?	373
FAQ29.02	什么是隐藏规则 (hiding rule)?	374
FAQ29.03	应该如何处理隐藏规则?	377
FAQ29.04	当派生类重新定义从基类继承而来的一组重载成员函数的一部分时, 应该怎么做?	379
FAQ29.05	能重载虚拟函数吗?	381
第 30 章	大三法则	384
FAQ30.01	这一章的目的是什么?	384
FAQ30.02	大三法则是什么?	384
FAQ30.03	当释放一个没有明确的析构函数的对象时会发生什么?	385
FAQ30.04	假如拷贝一个没有明确的拷贝构造函数的对象将会发生什么?	386
FAQ30.05	当给没有明确的赋值运算符的对象赋值时会发生什么?	387
FAQ30.06	大三法则是什么?	389