

<<Imperfect C++中文版>>

图书基本信息

书名：<<Imperfect C++中文版>>

13位ISBN编号：9787115277978

10位ISBN编号：7115277974

出版时间：2012-6-19

出版时间：人民邮电出版社

作者：[美]Matthew Wilson

页数：591

字数：875000

译者：荣耀,刘未鹏

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>



## <<Imperfect C++中文版>>

### 内容概要

即便是C++阵营里最忠实的信徒，也不得不承认：C++语言并不完美。实际上，世界上也没有完美的编程语言。

如何克服C++类型系统的不足？

在C++中，如何利用约束、契约和断言来实施软件设计？

如何处理被C++标准所忽略的动态库、静态对象以及线程等有关的问题？

隐式转换有何替代方案？

《Imperfect C++中文版》将为你一一解答这些问题。

针对C++的每一个不完美之处，《Imperfect C++中文版》都具体地分析原因，并探讨实用的解决方案。

书中也不乏许多作者创新的、你从未听说过或使用的技术，但这些确实能帮助你成为C++方面的专家。

《Imperfect C++中文版》适合有一定经验的C++程序员和项目经理阅读，也适合对C++编程的一些专门或高级话题感兴趣的读者参考。

## 作者简介

作者：（美国）Matthew Wilson 译者：荣耀 刘未鹏 Matthew Wilson 是一名软件开发顾问，也是 STLSoft 库的创建者。

他为双月刊 C/C++ Users Journal 撰写关于将 C/C++ 与其他语言和技术进行整合的专栏文章，同时也是 C++ Experts Forum 在线专栏作家。

Wilson 有十余年 C++ 开发经验。

荣耀，南京师范大学教师。

他是一名 C++ 讲师和研究者，译有《C++ 必知必会》、《C++ Templates 全览》以及《C++ Template Metaprogramming 中文版》（中文繁体版）等，并在期刊杂志上发表过多篇文章。

他原任电力自动化研究院工程师与项目经理，是数个企业级信息系统项目负责人。

刘未鹏，南京大学计算机系硕士毕业，现就职于微软亚洲研究院创新工程中心。

## &lt;&lt;Imperfect C++中文版&gt;&gt;

## 书籍目录

第一部分 基础知识	1
第1章 强制设计：约束、契约和断言	3
1.1 绿蛋和火腿	4
1.2 编译期契约：约束	4
1.2.1 must_have_base()	5
1.2.2 must_be_subscriptable()	6
1.2.3 must_be_subscriptable_as_decayable_pointer()	6
1.2.4 must_be_pod()	7
1.2.5 must_be_same_size()	9
1.2.6 使用约束	10
1.2.7 约束和TMP	11
1.2.8 约束：尾声	11
1.3 运行期契约：前置条件、后置条件和不变式	12
1.3.1 前置条件	13
1.3.2 后置条件	13
1.3.3 类不变式	15
1.3.4 检查？	
总是进行	16
1.3.5 DbC还是不DbC	17
1.3.6 运行期契约：尾声	17
1.4 断言	18
1.4.1 获取消息	19
1.4.2 不恰当的断言	20
1.4.3 语法以及64位指针	21
1.4.4 避免使用verify()	21
1.4.5 为你的断言命名	22
1.4.6 避免使用#ifdef_DEBUG	23
1.4.7 DebugBreak()和int 3	24
1.4.8 静态/编译期断言	24
1.4.9 断言：尾声	26
第2章 对象生命期	27
2.1 对象生命周期	27
2.2 控制你的客户端	28
2.2.1 成员类型	28
2.2.2 缺省构造函数	28
2.2.3 拷贝构造函数	29
2.2.4 拷贝赋值	29
2.2.5 new和delete	30
2.2.6 虚析构	30
2.2.7 explicit	31
2.2.8 析构函数	31
2.2.9 友元	32
2.3 MIL及其优点	33
2.3.1 取得一块更大的场地	35
2.3.2 成员顺序依赖	37

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 2.3.3 offsetof() 38
- 2.3.4 MIL : 尾声 39
- 第3章 资源封装 40
  - 3.1 资源封装分类 40
  - 3.2 POD类型 41
    - 3.2.1 直接操纵 41
    - 3.2.2 API函数和透明类型 42
    - 3.2.3 API函数和不透明类型 42
  - 3.3 外覆代理类 43
  - 3.4 RRID类型 45
    - 3.4.1 缺省初始化 : 缓式初始化 46
    - 3.4.2 未初始化 48
  - 3.5 RAII类型 51
    - 3.5.1 常性RAII和易变性RAII 51
    - 3.5.2 内部初始化和外部初始化 53
    - 3.5.3 RAII排列 53
  - 3.6 RAII : 尾声 54
    - 3.6.1 不变式 54
    - 3.6.2 错误处理 54
- 第4章 数据封装和值类型 55
  - 4.1 数据封装的分类学 55
  - 4.2 值类型和实体类型 56
  - 4.3 值类型的分类学 56
  - 4.4 开放式类型 58
    - 4.4.1 POD开放式类型 58
    - 4.4.2 C++数据结构 59
  - 4.5 封装式类型 60
  - 4.6 值类型 61
  - 4.7 算术值类型 62
  - 4.8 值类型 : 尾声 63
  - 4.9 封装 : 尾声 64
- 第5章 对象访问模型 68
  - 5.1 确定性生命期 68
  - 5.2 返回拷贝 70
  - 5.3 直接交给调用者 70
  - 5.4 共享对象 71
- 第6章 域守卫类 73
  - 6.1 值 73
  - 6.2 状态 78
  - 6.3 API和服务 83
    - 6.3.1 API 83
    - 6.3.2 服务 86
  - 6.4 语言特性 87
- 第二部分 生存在现实世界 89
- 第7章 ABI 91
  - 7.1 共享代码 91
  - 7.2 C ABI需求 93

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 7.2.1 结构布局 93
- 7.2.2 调用约定、符号名以及目标文件格式 94
- 7.2.3 静态连接 94
- 7.2.4 动态连接 95
- 7.3 C++ ABI需求 96
  - 7.3.1 对象布局 97
  - 7.3.2 虚函数 97
  - 7.3.3 调用约定和名字重整 97
  - 7.3.4 静态连接 99
  - 7.3.5 动态连接 99
- 7.4 现在知道怎么做了 100
  - 7.4.1 extern “ C ” 100
  - 7.4.2 名字空间 103
  - 7.4.3 extern “ C++ ” 103
  - 7.4.4 获得C++类的句柄 106
  - 7.4.5 “由实现定义”的隐患 108
- 第8章 跨边界的对象 110
  - 8.1 近乎可移植的虚函数表 110
    - 8.1.1 虚函数表布局 111
    - 8.1.2 动态操纵虚函数表 113
  - 8.2 可移植的虚函数表 114
    - 8.2.1 利用宏进行简化 116
    - 8.2.2 兼容的编译器 116
    - 8.2.3 可移植的服务端对象 117
    - 8.2.4 简化可移植接口的实现 119
    - 8.2.5 C客户代码 120
    - 8.2.6 OAB的约束 120
  - 8.3 ABI/OAB尾声 121
- 第9章 动态库 123
  - 9.1 显式调用函数 123
    - 9.1.1 显式调用C++函数 124
    - 9.1.2 打破C++访问控制 125
  - 9.2 同一性：连接单元和连接空间 125
    - 9.2.1 连接单元 125
    - 9.2.2 连接空间 126
    - 9.2.3 多重身份 126
  - 9.3 生命期 127
  - 9.4 版本协调 128
    - 9.4.1 丢失的函数 128
    - 9.4.2 变化的签名 128
    - 9.4.3 行为的改变 129
    - 9.4.4 常量 129
  - 9.5 资源所有权 130
    - 9.5.1 共享池 130
    - 9.5.2 返还给被调用方 130
  - 9.6 动态库：尾声 131
- 第10章 线程 132

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 10.1 对整型值的同步访问 133
  - 10.1.1 操作系统函数 134
  - 10.1.2 原子类型 135
- 10.2 对(代码)块的同步访问：临界区 136
  - 10.2.1 进程间互斥体和进程内互斥体 137
  - 10.2.2 自旋互斥体 138
- 10.3 原子整型的性能 139
  - 10.3.1 基于互斥体的原子整型 139
  - 10.3.2 运行期按架构派发 141
  - 10.3.3 性能比较 142
  - 10.3.4 原子整型操作：尾声 143
- 10.4 多线程扩展 144
  - 10.4.1 synchronized 144
  - 10.4.2 匿名synchronized 147
  - 10.4.3 atomic 147
- 10.5 线程相关的存储 148
  - 10.5.1 重入 148
  - 10.5.2 线程相关的数据/线程局部存储 148
  - 10.5.3 declspec(thread)和TLS 150
  - 10.5.4 Tss库 150
  - 10.5.5 TSS的性能 155
- 第11章 静态对象 156
  - 11.1 非局部静态对象：全局对象 157
    - 11.1.1 编译单元内的顺序性 158
    - 11.1.2 编译单元间的顺序性 159
    - 11.1.3 利用main()避免全局变量 161
    - 11.1.4 全局对象尾声：顺序性 162
  - 11.2 单件 163
    - 11.2.1 Meyers单件 163
    - 11.2.2 Alexandrescu单件 164
    - 11.2.3 即时Schwarz计数器：一个极妙的主意 165
    - 11.2.4 对API计数 166
    - 11.2.5 被计数的API、外覆类、代理类：最终得到一个顺序化的单件 168
  - 11.3 函数范围内的静态对象 169
    - 11.3.1 牺牲缓式求值能力 171
    - 11.3.2 自旋互斥体是救星 171
  - 11.4 静态成员 172
    - 11.4.1 解决连接问题 172
    - 11.4.2 自适应代码 174
  - 11.5 静态对象：尾声 175
- 第12章 优化 176
  - 12.1 内联函数 176
    - 12.1.1 警惕过早优化 176
    - 12.1.2 只含有头文件的库 177
  - 12.2 返回值优化 177
  - 12.3 空基类优化 180
  - 12.4 空派生类优化 183



## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 12.5 阻止优化 184
- 第三部分 语言相关的议题 188
- 第13章 基本类型 189
  - 13.1 可以给我来一个字节吗 189
    - 13.1.1 标明符号 190
    - 13.1.2 一切都在名字之中 190
    - 13.1.3 窥探void内部 191
    - 13.1.4 额外的安全性 191
  - 13.2 固定大小的整型 192
    - 13.2.1 平台无关性 193
    - 13.2.2 类型相关的行为 195
    - 13.2.3 固定大小的整型：尾声 197
  - 13.3 大整型 198
  - 13.4 危险的类型 200
    - 13.4.1 引用和临时对象 200
    - 13.4.2 bool 201
- 第14章 数组和指针 204
  - 14.1 不要重复你自己 204
  - 14.2 数组退化为指针 206
    - 14.2.1 下标索引操作符的交换性 206
    - 14.2.2 阻止退化 208
  - 14.3 dimensionof() 209
  - 14.4 无法将数组传递给函数 211
  - 14.5 数组总是按地址进行传递 214
  - 14.6 派生类的数组 215
    - 14.6.1 通过指针保存多态类型 216
    - 14.6.2 提供非缺省的构造函数 217
    - 14.6.3 隐藏向量式new和delete 218
    - 14.6.4 使用std::vector 218
    - 14.6.5 确保类型的大小相同 219
  - 14.7 不能拥有多维数组 222
- 第15章 值 226
  - 15.1 NULL的是非曲直 226
  - 15.2 回到0 232
  - 15.3 屈服于事实 235
  - 15.4 字面量 236
    - 15.4.1 整型 236
    - 15.4.2 后缀 238
    - 15.4.3 字符串 240
  - 15.5 常量 243
    - 15.5.1 简单常量 243
    - 15.5.2 类类型常量 244
    - 15.5.3 成员常量 245
    - 15.5.4 类类型的成员常量 248
- 第16章 关键字 251
  - 16.1 interface 251
  - 16.2 temporary 253

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 16.3 owner 256
- 16.4 explicit(\_cast) 261
  - 16.4.1 使用显式访问函数 263
  - 16.4.2 模拟显式转换 264
  - 16.4.3 使用特性垫片 265
- 16.5 unique 266
- 16.6 final 267
- 16.7 不被支持的关键字 267
- 第17章 语法 270
  - 17.1 类的代码布局 270
  - 17.2 条件表达式 273
    - 17.2.1 “使它布尔” 273
    - 17.2.2 一个危险的赋值 275
  - 17.3 for 277
    - 17.3.1 初始化作用域 277
    - 17.3.2 异质初始化类型 278
  - 17.4 变量命名 280
    - 17.4.1 匈牙利命名法 280
    - 17.4.2 成员变量 281
- 第18章 Typedef 284
  - 18.1 指针typedef 286
  - 18.2 定义里面有什么 288
    - 18.2.1 概念性的类型定义 288
    - 18.2.2 上下文相关的类型定义 289
  - 18.3 别名 292
    - 18.3.1 错误的概念性类型互换 293
    - 18.3.2 不能对概念性类型进行重载 294
  - 18.4 true\_typedef 294
  - 18.5 好的、坏的、丑陋的 300
    - 18.5.1 好的typedef 300
    - 18.5.2 坏的typedef 303
    - 18.5.3 可疑的typedef 304
- 第四部分 感知式转换 308
- 第19章 强制 310
  - 19.1 隐式转换 310
  - 19.2 C++中的强制 311
  - 19.3 适合使用C强制的场合 312
  - 19.4 模仿强制 314
  - 19.5 explicit\_cast 316
  - 19.6 literal\_cast 321
  - 19.7 union\_cast 323
  - 19.8 comstl::interface\_cast 327
    - 19.8.1 interface\_cast\_addrf 328
    - 19.8.2 interface\_cast\_nofaddrf 329
    - 19.8.3 interface\_cast\_test 329
    - 19.8.4 接口强制操作符的实现 330
    - 19.8.5 保护引用计数 333

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 19.8.6 interface\_cast\_base 334
- 19.8.7 IID\_traits 335
- 19.8.8 interface\_cast 尾声 336
- 19.9 boost::polymorphic\_cast 337
- 19.10 强制：尾声 339
- 第20章 垫片 341
  - 20.1 拥抱变化 拥抱自由 341
  - 20.2 特性垫片 344
  - 20.3 逻辑垫片 346
  - 20.4 控制垫片 347
  - 20.5 转换垫片 348
  - 20.6 复合式垫片概念 350
    - 20.6.1 访问垫片 351
    - 20.6.2 返回值生命期 352
    - 20.6.3 泛化的类型操纵 354
    - 20.6.4 效率方面的考虑 356
  - 20.7 名字空间和Koenig查找 357
  - 20.8 为何不使用traits 359
  - 20.9 结构一致性 360
  - 20.10 打破巨石 362
  - 20.11 垫片：尾声 363
- 第21章 饰面 365
  - 21.1 轻量级RAII 366
  - 21.2 将数据和操作绑定在一起 367
    - 21.2.1 pod\_veneer 368
    - 21.2.2 创建日志消息 370
    - 21.2.3 减少浪费 371
    - 21.2.4 类型安全的消息类 372
  - 21.3 “擦亮”饰面概念 374
  - 21.4 饰面：尾声 376
- 第22章 螺栓 377
  - 22.1 添加功能 377
  - 22.2 皮肤选择 378
  - 22.3 非虚重写 379
  - 22.4 巧用作用域 380
  - 22.5 拟编译期多态：逆反式螺栓 383
  - 22.6 参数化多态包装 384
  - 22.7 螺栓：尾声 386
- 第23章 模板构造函数 387
  - 23.1 不易察觉的开销 389
  - 23.2 悬挂引用 389
  - 23.3 模板构造函数特化 391
  - 23.4 实参代理 392
  - 23.5 明确实参的范畴 394
  - 23.6 模板构造函数：尾声 395
- 第五部分 操作符 396
- 第24章 operator bool() 398

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 24.1 operator int() const 398
- 24.2 operator void \*() const 399
- 24.3 operator bool() const 400
- 24.4 operator !() const 401
- 24.5 operator boolean const \*() const 401
- 24.6 operator int boolean::\*() const 402
- 24.7 在现实世界中操作 402
- 24.8 operator! 407
- 第25章 快速、非侵入性的字符串拼接 408
  - 25.1 fast\_string\_concatenator 409
    - 25.1.1 与用户自定义的字符串类协同工作 409
    - 25.1.2 将“拼接子”串起来 410
    - 25.1.3 fast\_string\_concatenator类 411
    - 25.1.4 内部实现 413
  - 25.2 性能 417
  - 25.3 与其他字符串类协作 420
    - 25.3.1 整合进标准库中 420
    - 25.3.2 整合进可改动的现存类中 420
    - 25.3.3 与不可更改的类互操作 420
  - 25.4 拼接提示 421
  - 25.5 病态括号 422
  - 25.6 标准化 423
- 第26章 你的地址是什么 424
  - 26.1 无法得到真实的地址 424
    - 26.1.1 STL式元素存放 424
    - 26.1.2 ATL外覆类和CAadapt 425
    - 26.1.3 获取真实的地址 426
  - 26.2 在转换过程中发生了什么 427
  - 26.3 我们返回什么 429
  - 26.4 你的地址是什么：尾声 431
- 第27章 下标索引操作符 434
  - 27.1 指针转换与下标索引操作符 434
    - 27.1.1 选择隐式转换操作符 436
    - 27.1.2 选择下标索引操作符 437
  - 27.2 错误处理 437
  - 27.3 返回值 439
- 第28章 增量操作符 441
  - 28.1 缺少后置式操作符 442
  - 28.2 效率 443
- 第29章 算术类型 446
  - 29.1 类定义 446
  - 29.2 缺省构造 447
  - 29.3 初始化(值构造) 447
  - 29.4 拷贝构造函数 450
  - 29.5 赋值 450
  - 29.6 算术操作符 451
  - 29.7 比较操作符 452

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 29.8 访问值 452
- 29.9 sinteger64 453
- 29.10 截断、提升以及布尔测试 453
  - 29.10.1 截断 453
  - 29.10.2 提升 455
  - 29.10.3 布尔测试 455
- 29.11 算术类型：尾声 456
- 第30章 短路 458
- 第六部分 扩展C++ 460
- 第31章 返回值生命期 461
  - 31.1 返回值生命期问题分类 461
    - 31.1.1 局部变量 462
    - 31.1.2 局部静态对象 462
    - 31.1.3 析构后指针(Postdestruction Pointers) 462
  - 31.2 为何按引用返回 462
  - 31.3 解决方案1——integer\_to\_string 462
  - 31.4 解决方案2——TSS 465
    - 31.4.1 --declspec(thread) 466
    - 31.4.2 Win32 TLS 466
    - 31.4.3 平台无关的API 469
    - 31.4.4 RVL 470
  - 31.5 解决方案3——扩展RVL 470
    - 31.5.1 解决线程内的RVL-LS问题 471
    - 31.5.2 RVL 472
  - 31.6 解决方案4——静态数组大小决议 472
  - 31.7 解决方案5——转换垫片 474
  - 31.8 性能 476
  - 31.9 RVL：垃圾收集的大胜利 477
  - 31.10 可能的应用 478
  - 31.11 返回值生命期：尾声 478
- 第32章 内存 479
  - 32.1 内存分类 479
    - 32.1.1 栈和静态内存 479
    - 32.1.2 栈扩张 480
    - 32.1.3 堆内存 481
  - 32.2 两者之间的折衷 481
    - 32.2.1 alloca() 482
    - 32.2.2 VLA 483
    - 32.2.3 auto\_buffer 483
    - 32.2.4 使用auto\_buffer 486
    - 32.2.5 EBO，在哪里 487
    - 32.2.6 性能 488
    - 32.2.7 堆、栈以及其他 490
    - 32.2.8 pod\_vector 491
  - 32.3 配置器 493
    - 32.3.1 函数指针 493
    - 32.3.2 配置器接口 494

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 32.3.3 每库初始化(Per-library Initialization) 495
- 32.3.4 每调用指定(Per-Call Specification) 496
- 32.4 内存：尾声 496
- 第33章 多维数组 497
- 33.1 激活下标索引操作符 498
- 33.2 运行时确定大小 499
  - 33.2.1 可变长数组 499
  - 33.2.2 vector< ... vector ... > 500
  - 33.2.3 boost::multi\_array 501
  - 33.2.4 fixed\_array\_1/2/3/4d 501
- 33.3 编译期确定大小 505
  - 33.3.1 boost::array 506
  - 33.3.2 static\_array\_1/2/3/4d 506
- 33.4 块访问 508
  - 33.4.1 使用std::fill\_n() 509
  - 33.4.2 array\_size垫片 510
- 33.5 性能 512
  - 33.5.1 运行期确定大小 513
  - 33.5.2 编译期确定大小 514
- 33.6 多维数组：尾声 515
- 第34章 仿函数和区间 516
- 34.1 语法混乱 516
- 34.2 for\_all() 517
  - 34.2.1 数组 518
  - 34.2.2 命名 518
- 34.3 局部仿函数 520
  - 34.3.1 手写循环 520
  - 34.3.2 自定义仿函数 521
  - 34.3.3 内嵌的仿函数 521
  - 34.3.4 温和一些 523
  - 34.3.5 泛化的仿函数：类型隧道(Type Tunneling) 524
  - 34.3.6 再进一步，走得太远了 526
  - 34.3.7 局部仿函数和回调API 527
- 34.4 区间 529
  - 34.4.1 区间概念 529
  - 34.4.2 概念性区间 531
  - 34.4.3 可迭代区间 533
  - 34.4.4 区间算法和标签 533
  - 34.4.5 过滤器 535
  - 34.4.6 虚伪 536
- 34.5 仿函数和区间：尾声 536
- 第35章 属性 537
- 35.1 编译器扩展 539
- 35.2 可供选择的实现方案 539
  - 35.2.1 将属性的实现分门别类 540
  - 35.2.2 EMO 540
- 35.3 字段属性 541

## &lt;&lt;Imperfect C++中文版&gt;&gt;

- 35.3.1 field\_property\_get 541
- 35.3.2 field\_property\_set 545
- 35.3.3 内置式字段属性：尾声 546
- 35.3.4 field\_property\_get\_external 546
- 35.3.5 field\_property\_set\_external 547
- 35.3.6 Hack掉 547
- 35.4 方法属性 548
  - 35.4.1 method\_property\_get 548
  - 35.4.2 method\_property\_set 555
  - 35.4.3 method\_property\_getset 555
  - 35.4.4 谨防无限循环 557
  - 35.4.5 method\_property\_get\_external 558
  - 35.4.6 method\_property\_set\_external 561
  - 35.4.7 method\_property\_getset\_external 562
- 35.5 静态属性 564
  - 35.5.1 静态字段属性 564
  - 35.5.2 内置式静态方法属性 564
  - 35.5.3 外置式静态方法属性 566
- 35.6 虚属性 567
- 35.7 属性的使用 568
  - 35.7.1 泛化性 568
  - 35.7.2 错误诊断中的类型替换 569
- 35.8 属性：尾声 570
- 附录A 编译器和库 572
  - A.1 编译器 572
  - A.2 库 573
    - A.2.1 Boost 574
    - A.2.2 STLSoft 574
    - A.2.3 其他库 574
  - A.3 其他资源 575
    - A.3.1 期刊 575
    - A.3.2 其他语言 575
    - A.3.3 新闻组 576
- 附录B “谦虚点，别骄傲” 577
  - B.1 操作符重载 577
  - B.2 后悔DRY 579
  - B.3 偏执式编程 579
  - B.4 精神错乱 580
- 附录C Arturius 582
- 附录D 随书光盘 583
- 尾声 584
- 参考书目 585



## 章节摘录

版权页：插图：如果你没有任何COM背景的话，上面的代码对你来说可能会相当陌生，但事实上它是相当直观的。

它之所以能够工作是因为从根本上说类也只不过是个结构，并且，如果该类定义了虚函数（或者派生自某个定义了虚函数的类）的话，那么其实例将会包含一个隐藏的vptr（指向vtable的指针）。

vptr指向一个表，该表中包含了指向该类的所有虚函数的指针，因而被称为虚函数表（vtable）。

一般情况下，同一个类的所有实例共享同一份vtable。

在上面的C代码中，vtable是以结构IObjectVTable来表现的，它包含了指向SetName（）和GetName（）函数的指针，这种函数指针跟其他的函数指针没什么两样，只不过其第一个参数总是指向该接口的指针，即C++中的this指针。

struct IObject中只有一个成员vtable，它是一个指针，指向该接口的虚函数表，即struct IObjectVTable的一个实例。

正如我前面说过的，我们的6个Win32编译器在这个布局方面完全一致。

或许Win32编译器支持这种布局并非巧合，因为这种布局正是COM所使用的，而Win32编译器要想得到流行的话必须支持COM。

尽管我们承认这当然是一个明显而高效的对象布局模型，然而确实存在着一些编译器不这么做。

一个不支持这种布局的Win32编译器就是GCC2.95（GCC3.2支持）。

经过一些“不足为外人道”的手法，探知GCC2.95使用的vtable布局是像这样的：v1、v3和v4的值是0，因此我猜测这与pack问题有关，v2看起来是指向某个地址跟SetName（）和GetName（）很靠近的函数，但我不知道这个函数的精确特征。

如此“特立独行”的并非仅GCC2.95一家，Sun的C++编译器1使用类似下面的这种布局：所以现在我们有了一个不太理想的选择。

选择之一是采用这种部分的解决方案，至少Win32上的现代编译器看起来都支持这种方案。

如果在其他平台上试一下的话，可能也会发现类似的一致布局的情况，对此我们可以采取同样的立场。

。

且慢！

我们是不完美主义的实践者，并且这种做法远没达到预期的程度。

我们需要寻找一种彻底的解决方案。

8.1.2动态操纵虚函数表 在我们试图设计出一个完全可移植的方案之前，我暂且充当一次不负责任的角色，向你展示一些“奇技淫巧”，借此你可以对C++对象布局动点手术。

可能你会觉得这里讲的这些东西太高深，并担心是否必须自己定义这些C虚函数表。

通常你不必这么做，C++编译器会为你打理好一切。

干预任何C++编译器生成的类的虚函数表都不是明智之举，因为你对它的任何干预都会在整进程的生命期中反映在该类的所有实例身上。

1但是在c里面你完全可以操纵你自己的虚函数表，这种能力在某些非常罕见的场合下可以被用来改变对象的运行期特征（行为）。

我并不建议你采取这种做法，它的主要作用在于可以学习c++实现相关的机理，但是作为用在产品软件中的技术却不大合适。

不过，知道它是怎么做的倒也不算坏事。

基本上，这可以分为3个步骤。

首先，你必须分配自己的虚函数表。

这在C里面实现，只不过是简单地分配一份可用于保存虚函数表内容的内存块而已。

其次，你得从一个有效的对象中将虚函数表拷贝出来，这也是用c来实现的，只不过是在c++编译单元中创建的对象身上进行操作。

最后，你可以改变你的新虚函数表中的成员，并将这个新的虚函数表挂到你要改变其行为的对象身上，这仍然是在C里面实现的，但目标对象可以是c++编译单元中创建的对象。



所有这3个步骤可以被包装在同一个函数中。

## &lt;&lt;Imperfect C++中文版&gt;&gt;

## 媒体关注与评论

“千万不要被书名所误导！

这是一本拥抱（而非诋毁）C++的著作。

它有着独特的定位：为现实世界中的程序员提供切合实际的解决方案，以解决C++语言自身的各种“不完美”。

世界上没有完美的编程语言。

在本书中，Matthew Wilson不但为我们指出C++中诸多不完美之处，还提供了经过实践检验的应对技术和技巧，便于我们利用“不完美的C++”编写出近乎完美的代码——强健、高效、灵活、可移植、优雅的代码，而这些代码在声称为“完美的语言”中往往更难实现。

本书对给出的每一个“不完美”都进行了细致的探讨：为什么说它是一个“不完美”？

对其修复的指导思想是什么？

有时候只是告诫你避免做些什么，给出一些约束和建议，更多的时候则为你提供现实的解决方案，这些方案往往离不开对现代模板编程技术的使用。

书中包含有许多你未曾听过或用过的技术，有些属于作者的创新，有些则是对现有技术的精化，二者均被提升到“范式”的高度。

例如：应用程序二进制接口（ABI）、垫片（Shim）、饰面（Veneer）、螺栓（Bolt-in）、区间（range）、属性（property）等。

不少主题难度大，此前为其他C++专家所忽略。

要探讨它们除了需要勇气外，第一手经验更是不可或缺。

作为STLSoft库的主创者，Matthew在举例时，对Windows API、MFC、ATL、COM以及UNIX等都是信手拈来。

除了丰富的实践、扎实的理论以及缜密的逻辑外，Matthew的文笔流畅，语言幽默，说理直接，字里行间流露出过人的自信，使得本书极具阅读趣味。

本书具有一定的阅读门槛，目标读者为中、高级职业C++程序员。

书中展示的代码示例、编程技术往往在几款甚至十几款编译器上进行验证，辅以表格对其各色特性加以比较，并针对不同编译器所表现出的差异性而给出高效、可移植的解决方案——就像很多现实世界中的C++程序员应该做（而没做到）的那样。

如果你正在寻找一本真材实料的“C++实战”参考书，本书不会让你失望。

本书中文版由我和刘未鹏先生合译。

未鹏思维敏捷，技术、文笔俱佳，我很高兴与他合作。

感谢陈冀康编辑给予的理解和支持。

感谢朱艳的照料和热爱。

荣坤则常常用他的小拳头乱砸书房的门，并大声地叫“爸爸”，这种干扰让我获得了必不可少的休息时间。

已有的经典名著使得C++新书问世难度加大，后来者若无过人之处就很难引起C++社群的注意，Imperfect C++、C++ Common Knowledge以及C++ Template Metaprogramming等佳作一经问世便得到广泛的关注。

作为译者（之一），我祝愿它们能够带给各位久违的快乐！

——荣耀 刀有很多种，有单刀，双刀，朴刀，戒刀，锯齿刀，砍山刀，鬼头刀，雁翎刀，五凤朝阳刀，鱼鳞紫金刀。

——古龙《飞刀，又见飞刀》 这里我们要说的刀，是瑞士军刀，瑞士军刀其实严格来说并不能算是一种刀，其功能的繁杂和精细已然超过了刀的范畴。

它包含的工具一般有主刀、小刀、剪刀、开瓶器、木锯、小改锥、拔木塞钻、牙签、小镊子等，而在一些工具上还设计了多种功用，如开瓶器上，就具有开瓶、平口改锥、电线剥皮槽3种功用。

随着时代的发展，一些新兴的电子技术也被引入瑞士军刀中，如内藏激光笔、电筒等。

## &lt;&lt;Imperfect C++中文版&gt;&gt;

瑞士军刀是军人在野外生存的必备工具，其小体积浓缩众多实用功能的精心设计能够将一把刀的容限发挥到最大，丝毫不逊于《第一滴血》中蓝博带在身上的那把锐利的寒光闪闪的钢刀。

那么现在你拿在手里的这本书就是一把瑞士军刀！

这是一本非常特别的C++图书，在市面上已经存在的大量经典C++书籍当中，这本书的着眼点和写作风格使它显得那么特立独行和标新立异，甚至有点另类。

书中几乎巨细靡遗地涵盖了C++中大大小小的不完美之处，并以一系列成功案例证明C++的确不完美同时也提供了迂回之道、解决之道，再加上其用本主义的立场，正如同一把实用的瑞士军刀，功能繁杂而面面俱到，实用之至。

同其他C++著作不一样，本书虽然尊重标准，但同时又超越标准，当标准不能满足需求或成为拦路石的时候，需求才是第一位的，于是有了作者所谓的“不完美主义的实践者”以及“不完美工具箱”之说。

此外，作者的所谓“苦行僧式编程”哲学在我看来也是极其实用的一种编码方式！

我们以前看到的绝大部分C++书籍可说是统统走的“阳关大道”，然而Matthew这本书却偏要走他的“独木小桥”，蹊径虽小，然则别有一番风味和景观。

我们意识到原来C++中也存在着如此多大大小小的不完美之处，就像宫崎骏电影中的那些打满补丁的海盗飞机一样。

Bjarne本就说过，C++是为“用本”而设计的，诚然！

而本书最大的趣味就在于它并不去一味抱怨这些缺点，而是积极地采取其他替代方案来达到同样的目的，并借此展现出C++自由强大的一面！

作者Matthew常用“survive”一词来描述在编码的现实世界中的境况，作为STLSoft库的主要编写者，他十几年来积累的经验在书中充盈四溢，很多我们平常看不到的方面都会被他挑出来，甚至连我这个译者都觉得有点“啰嗦”。

不过，对于喜欢他这种“唠叨”讲法的人，他那种辩证的严密论证法倒是能令你获益颇多。

另外，书中随处可见具有作者个人特色的幽默，在大量平淡无奇的技术书籍当中可算是一个亮点。

本书的一个小缺憾就是它不适合初学者，某些地方甚至对于中级读者来说都有一定的难度，作者自己经验非常丰富，因此有些地方就不加解释地一带而过，为此译者适当添加了一些译注，以便读者理解和阅读。

最后，感谢荣耀先生在本书初译的过程中一直给予的支持和信任，并容忍我总是延期交付各章译稿。

荣耀先生对技术的精益求精和一丝不苟也令我在翻译的过程中获益良多。

最大的感激要归于我的父母和我的爷爷，感谢他们一直以来对我的追求的支持和鼓励，没有他们我无法想象能够完成这项工作。

希望这本令我在翻译过程中获益匪浅的书也能够给你带来美妙而独一无二的阅读享受，Let's dig in！

——刘未鹏

## <<Imperfect C++中文版>>

### 编辑推荐

《Imperfect C++中文版》是市面上唯一一本讨论C++的不足之处，并给出解决方案的C++经典图书，荣获Amazon五星级评价。

由知名译者荣耀与刘未鹏合译，在技术开发者中拥有极大影响力。

## &lt;&lt;Imperfect C++中文版&gt;&gt;

## 名人推荐

千万不要被书名所误导！

这是一本拥抱（而非诋毁）C++的著作。

它有着独特的定位：为现实世界中的程序员提供切合实际的解决方案，以解决C++语言自身的各种“不完美”。

世界上没有完美的编程语言。

在本书中，Matthew Wilson不但为我们指出C++中诸多不完美之处，还提供了经过实践检验的应对技术和技巧，便于我们利用“不完美的C++”编写出近乎完美的代码——强健、高效、灵活、可移植、优雅的代码，而这些代码在声称为“完美的语言”中往往更难实现。

本书对给出的每一个“不完美”都进行了细致的探讨：为什么说它是一个“不完美”？

对其修复的指导思想是什么？

有时候只是告诫你避免做些什么，给出一些约束和建议，更多的时候则为你提供现实的解决方案，这些方案往往离不开对现代模板编程技术的使用。

书中包含有许多你未曾听过或用过的技术，有些属于作者的创新，有些则是对现有技术的精化，二者均被提升到“范式”的高度。

例如：应用程序二进制接口（ABI）、垫片（Shim）、饰面（Veneer）、螺栓（Bolt-in）、区间（range）、属性（property）等。

不少主题难度大，此前为其他C++专家所忽略。

要探讨它们除了需要勇气外，第一手经验更是不可或缺。

作为STLSoft库的主创者，Matthew在举例时，对Windows API、MFC、ATL、COM以及UNIX等都是信手拈来。

除了丰富的实践、扎实的理论以及缜密的逻辑外，Matthew的文笔流畅，语言幽默，说理直接，字里行间流露出过人的自信，使得本书极具阅读趣味。

本书具有一定的阅读门槛，目标读者为中、高级职业C++程序员。

书中展示的代码示例、编程技术往往在几款甚至十几款编译器上进行验证，辅以表格对其各色特性加以比较，并针对不同编译器所表现出的差异性而给出高效、可移植的解决方案——就像很多现实世界中的C++程序员应该做（而没做到）的那样。

如果你正在寻找一本真材实料的“C++实战”参考书，本书不会让你失望。

本书中文版由我和刘未鹏先生合译。

未鹏思维敏捷，技术、文笔俱佳，我很高兴与他合作。

感谢陈冀康编辑给予的理解和支持。

感谢朱艳的照料和热爱。

荣琬则常常用他的小拳头乱砸书房的门，并大声地叫“爸爸”，这种干扰让我获得了必不可少的休息时间。

已有的经典名著使得C++新书问世难度加大，后来者若无过人之处就很难引起C++社群的注意，Imperfect C++、C++ Common Knowledge以及C++ Template Metaprogramming等佳作一经问世便得到广泛的关注。

作为译者（之一），我祝愿它们能够带给各位久违的快乐！

——荣耀刀有很多种，有单刀，双刀，朴刀，戒刀，锯齿刀，砍山刀，鬼头刀，雁翎刀，五凤朝阳刀，鱼鳞紫金刀。

——古龙《飞刀，又见飞刀》这里我们要说的刀，是瑞士军刀，瑞士军刀其实严格来说并不能算是一种刀，其功能的繁杂和精细已然超过了刀的范畴。

它包含的工具一般有主刀、小刀、剪刀、开瓶器、木锯、小改锥、拔木塞钻、牙签、小镊子等，而在一些工具上还设计了多种功用，如开瓶器上，就具有开瓶、平口改锥、电线剥皮槽3种功用。

随着时代的发展，一些新兴的电子技术也被引入瑞士军刀中，如内藏激光笔、电筒等。

瑞士军刀是军人在野外生存的必备工具，其小体积浓缩众多实用功能的精心设计能够将一把刀的容限

## &lt;&lt;Imperfect C++中文版&gt;&gt;

发挥到最大，丝毫不逊于《第一滴血》中蓝博带在身上的那把锐利的寒光闪闪的钢刀。

那么现在你拿在手里的这本书就是一把瑞士军刀！

这是一本非常特别的C++图书，在市面上已经存在的大量经典C++书籍当中，这本书的着眼点和写作风格使它显得那么特立独行和标新立异，甚至有点另类。

书中几乎巨细靡遗地涵盖了C++中大大小小的不完美之处，并以一系列成功案例证明C++的确不完美同时也提供了迂回之道、解决之道，再加上其用本主义的立场，正如同一把实用的瑞士军刀，功能繁杂而面面俱到，实用之至。

同其他C++著作不一样，本书虽然尊重标准，但同时又超越标准，当标准不能满足需求或成为拦路石的时候，需求才是第一位的，于是有了作者所谓的“不完美主义的实践者”以及“不完美工具箱”之说。

此外，作者的所谓“苦行僧式编程”哲学在我看来也是极其实用的一种编码方式！

我们以前看到的绝大部分C++书籍可说是统统走的“阳关大道”，然而Matthew这本书却偏要走他的“独木小桥”，蹊径虽小，然则别有一番风味和景观。

我们意识到原来C++中也存在着如此多大大小小的不完美之处，就像宫崎俊电影中的那些打满补丁的海盗飞机一样。

Bjarne本就说过，C++是为“用本”而设计的，诚然！

而本书最大的趣味就在于它并不去一味抱怨这些缺点，而是积极地采取其他替代方案来达到同样的目的，并借此展现出C++自由强大的一面！

作者Matthew常用“survive”一词来描述在编码的现实世界中的境况，作为STLSoft库的主要编写者，他十几年来积累的经验在书中充盈四溢，很多我们平常看不到的方面都会被他挑出来，甚至连我这个译者都觉得有点“啰嗦”。

不过，对于喜欢他这种“唠叨”讲法的人，他那种辩证的严密论证法倒是能令你获益颇多。

另外，书中随处可见具有作者个人特色的幽默，在大量平淡无奇的技术书籍当中可算是一个亮点。

本书的一个小缺憾就是它不适合初学者，某些地方甚至对于中级读者来说都有一定的难度，作者自己经验非常丰富，因此有些地方就不加解释地一带而过，为此译者适当添加了一些译注，以便读者理解和阅读。

最后，感谢荣耀先生在本书初译的过程中一直给予的支持和信任，并容忍我总是延期交付各章译稿。

荣耀先生对技术的精益求精和一丝不苟也令我在翻译的过程中获益良多。

最大的感激要归于我的父母和我的爷爷，感谢他们一直以来对我的追求的支持和鼓励，没有他们我无法想象能够完成这项工作。

希望这本令我在翻译过程中获益匪浅的书也能够给你带来美妙而独一无二的阅读享受，Let's dig in！

——刘未鹏

<<Imperfect C++中文版>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介, 请支持正版图书。

更多资源请访问:<http://www.tushu007.com>