深入 C++ 系列

C++ Footprint and Performance Optimization

C++ 高效编程：内存与性能优化

[美] R. Alexander G. Bensley 著
王峰 史金虎 译

SAMS
中国电力出版社
www.infopower.com.cn

深入 C++ 系列

C++ Footprint and Performance Optimization

C++ 高效编程：内存与性能优化

[美] R. Alexander G. Bensley 著
王峰 史金虎 译

中国电力出版社

33

C++ Footprint and Performance Optimization (ISBN 0-672-31904-7)
Rene Alexander Graham Bensley
Authorized translation from the English language edition, entitled C++ Footprint and Performance Optimization, published by Sams Publishing, Copyright©2000.

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：01-2003-4141 号

图书在版编目（CIP）数据
C++高效编程：内存与性能优化 /（美）亚历山大，（美）本斯林著；
王峰，史金虎译. —北京：中国电力出版社，2003
（深入C++系列）
ISBN 7-5083-1555-3
Ⅰ.C... Ⅱ.①亚...②本...③王...④史... Ⅲ.C语言—程序设计 Ⅳ.TP312
中国版本图书馆CIP数据核字（2003）第057349号

责任编辑：姚贵胜
丛 书 名：深入C++系列
书　　名：C++高效编程：内存与性能优化
编　　著：（美）Rene Alexander Graham Bensley
翻　　译：王峰 史金虎
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）88518169
本书如有印装质量问题，我社负责退换
印　　刷：汇鑫印务有限公司
开　　本：787×1092 1/16　　印　　张：21.5　　字　　数：409千字
书　　号：ISBN 7-5083-1555-3
版　　次：2003年9月北京第一版
印　　次：2003年9月第一次印刷
定　　价：38.00 元

献　词

本书谨献给对我们包容尽致的 Olivera、Tjitske 与 Leanne。

致　谢

我们要感谢 Sams 公司的工作人员，正是他们认识到本书的价值并帮助我们出版本书。在此特别感谢我们的家人，他们勤劳持家、任劳任怨，一直在背后默默地支持着我们。

关 于 作 者

作为本书作者，我们有幸在此进行自我介绍。我们都富有十多年的专业 IT 经验，并且为众多公司做过大量不同的项目。目前，其中一位正在专心设计时限嵌入式软件，而另一位则正致力于开发用于数字电视通信的 Internet 和卫星传输软件。

我们曾利用业余时间做过一个项目，其中涉及到关于 C/C++代码速度和大小优化方面的高级技术的开发。我们发现，开发专业软件时，无论是嵌入式软件还是桌面应用程序，总会出现同类问题和陷阱。深入理解这些问题似乎不仅可以有助于编写更好的软件，而且对于修改错误也会大有帮助。这就是我们为什么决定把工程结果做成一部教程的原因。在本教程中，我们和大家分享了实际问题中发现的新情况与我们所使用的解决方案，以及关于软件优化的理论。

自开始从事 IT 工作以来，我们与许多同事一直在追寻上述类型的书籍，本书终于使我们的梦想变成了现实。

R. Alexander

G. Bensley

前　言

——为什么要进行优化

目前，软件已经渗入了世界上的各个角落。虽然在谈论软件时，人们可能最初想到的仅仅是 PC 和工业计算机系统，但是应用程序却更为普遍。洗衣机、电动剃须刀、自动调温装置、微波炉、汽车、电视、监视器等等都是使用应用程序的实例。显然，这些实例中应用了许多不同类型的体系结构，使用了各种各样的微处理器。因此，需要不同的优化技术来优化性能和内存。

而且，现在程序设计人员的素质参差不齐。许多软件实现人员受过专门的程序设计教育，包括需求分析、设计和实现等方面。也有一些人可能是从业余爱好者起步，自学程序设计。我们看到，越来越多的人正从不同专业转到程序设计上来。这意味着所有程序员本质上应具有较深的技术背景的假设已经不再成立。

C/C++的程序设计课程和书籍详细介绍了 C/C++的程序设计内容，由于各个年龄段和各专业的人基本上都可以理解，因此，学习的人都有可能编写出可运行的 C/C++程序。但是，在开发软件时，无论是商业性的还是业余爱好，标准技术都存在很多可以避免的缺陷和繁琐之处。由于并非完全是在技术层次上理解程序设计方案，所以实现人员不免会在编写软件时牺牲软件的速度和大小。

早在编写实际代码之前，进行需求分析和设计与选择硬件目标阶段，就确定了是否能够编写出高效运行的程序。甚至在实际编写软件过程中，通过一个简单的语法处理，我们就可能足以看出程序设计人员是否能够设计出好的程序。如果了解程序设计内容，则可以容易地优化代码，实现程序的高效运行。甚至可以应用一些编程技巧进一步提高程序运行效率，程序员技术水平不同，优化层次也有所不同。

本书编写目的

正如书名所表现出的，编写本书的目的就是帮助读者优化软件的性能和内存。无论读者是软件构架师、实现人员，还是项目负责人，作为一本指南，本书意在帮助这些读者学习或者提高下列基本技能：

- 分析在开发过程中何时、何处容易出现问题。
- 认识标准设计和编程技术存在的缺陷。

- 提高 C/C++程序设计技巧。
- 深刻理解编程技术。
- 学习行之有效的解决方案及其应用场合。

这些技能是编写高效软件的基础。即使是初级程序员，应用本书所述的高级技术，也可以编写出比较好的软件。经验丰富的程序员可以直接学习本书的高级部分，同时也会发现本书是一座金光闪闪的知识宝库，其中含有各种各样的程序优化内容，使程序员们在激烈的人才竞争中始终立于不败之地。本书包含的许多关于开发过程的提示、见解与实例等对项目负责人和构架师也会大有帮助。

本书读者

本书开始几章论述优化方面的理论，然后详细介绍技术问题和实例，难度逐渐加深。因此，初级程序员可以把本书作为学习优化技术的指导教材，而富有经验的程序员可以快速浏览前面比较简单的部分，而后即可深入研究后面较为复杂的内容。诸如项目负责人或者项目构架师等并不直接参与实际程序实现的读者，可以阅读本书第一部分中所论述的基本理论与后面几章中的“陷阱”和“技巧”部分，将会受益匪浅。本书中，问题实例与优化解决方案比比皆是，任何喜爱编程的人皆会从中受益。

本书结构

本书结构清晰，分为三个部分：

- 第一部分“基本优化理论”（1~3 章）——讨论了实际进行程序设计之前的软件开发优化理论，同时在选择编程语言、分析目标硬件、考虑设备交互、确定正确的系统需求、设计新系统与优化存在性能问题的现有系统等方面，都提出了相关的建设性意见，并枚举了附带解决方案的实例。
- 第二部分“亲手尝试优化”（4~13 章）——通过分析一些与在软件编码阶段经常遇到的问题有关的实例，讨论了如何实现编码的问题，说明这些问题可能在何处发生以及为何会产生，同时也为高效的函数调用、内存管理、输入输出以及建立和处理数据结构提供了一些可以直接应用的解决方案。
- 第三部分“技巧和陷阱”（14 章和 15 章）——综述了使用 C/C++编程时可能遇到的潜在问题和陷阱。

关于本书中使用的代码实例，请登录网站http://www.samspublishing.com，使用本书英文版的 ISBN 号 0672319047 查找。

目　录

第三部分　技巧和陷阱

第一部分 基本优化理论

本部分讨论了实际进行程序设计之前的软件开发优化理论，同时在选择编程语言、分析目标硬件、考虑设备交互、确定正确的系统需求、设计新系统与优化存在性能问题的现有系统等方面，都提出了相关的建设性意见，并枚举了附带解决方案的实例。

本部分包括下列几章：

1 基本优化内容

2 创建新系统

3 修改原系统

基本优化内容

1

本章内容

- 性能
- 内存

本章详细介绍了优化内容。其中具体解释了优化中所使用的定义和术语，并且阐述了掌握软件优化时机和方式的非常重要的原因。通常情况下，术语“优化”只与性能增强有关，但是，尽量减小一个应用程序所需要的内存量和其他资源也是很重要的。本章所讨论的内容既包括性能优化也包括内存优化。

1.1 性能

本章第一部分是从性能角度来讨论优化问题，不仅讨论了软硬件的特性，也讨论了系统用户考察性能的方法。

1.1.1 什么是性能

软件性能实际上是什么意思呢？简单的回答是，性能指某一段时间中所完成的工作量的外在表现。程序在单位时间内做的工作越多，性能就越好。换一种表达方式，一个程序的性能是以在给定时间内程序把输入（数据）单元转换成输出（数据）单元的数量来衡量的。这种表达把性能含义直接变成完成上述转换所需要应用的算法步骤的数量。例如，使用执行 10 条语句把一个名字存入一个数据库中的算法，与存储同一个名字而只用 5 条语句的算法相比，前者性能就较差。同样地，在未知新数据插入位置时需要执行 20 步的数据库设置，与在同种情况下只需要 10 步的数据库设置相比，对程序性能产生的影响要大得多。但是，除了纯粹技术实现方面之外，还需要考虑更多情况，这也是本节的重点所在。

现在所编写的软件中，很大一部分都是为一个或者多个用户的交互使用而完成的。例如，字处理软件、工程管理工具与绘画程序等。这类程序的用户通常是坐在计算机前面，始终使用一个程序工作，直到完成某项任务为止，例如，编制部门工作计划、绘制一幅图表或者写一封敲诈信。因此，让我们考察一下该类用户是如何定义性能的，毕竟在绝大多数情况下他们将是优化软件的使用者。基本上，只有在下列三种情况下，用户才会考虑性能问题：

- 当完成一项任务的时间比用户预期的少时。
- 当完成一项任务的时间比用户预期的多时。
- 当用户了解任务的规模或者复杂性时。

研究上述情况可以为定义性能提供进一步的指导。下面使用三个实例来说明上面加黑点的内容：

完成一项任务的时间少于用户预期的情形。例如，某用户一直在同一台计算机上使用

同一个程序工作了数年，后来，老板决定使用新型计算机。他仍然使用同一个程序，但是由于新硬件可以更快地执行，所以性能似乎好多了。同时，他已经对上述工作习以为常了。现在，情况完全出乎他的预料，他再也不会为保存大型文件或者执行复杂计算而苦苦等待了。

完成一项任务的时间多于用户预期的情形。例如，某用户工作时使用一个处理大型名称数据库的程序。启动后，在未进行系统更新前，该程序就会花费 15 秒钟加载和排列数据。即使该程序算法优化得很好，该用户也会把意料之外的“延迟”看作软件性能缺陷。

用户了解任务的规模或者复杂性的情形。例如，某用户使用一个程序在一个数兆字节的文本中查找某个字符串出现的次数。这项操作只用了几秒钟。由于他具有一定的技术背景，知道执行操作所涉及到的工作内容。因此，他会满意该查找程序的性能。

这些实例说明，性能不仅仅是时间和数据处理的简单测量。从用户的观点来说，性能更多的是他对程序的感觉而不是程序每秒钟内完成的实际工作量。下列多种因素都会影响这种感觉：

- 意料之外的和令人莫名其妙的等待会让用户对性能的感觉产生负面影响。
- 性能是软、硬件共同作用的结果。
- 性能依赖于用户所习惯的行为。
- 性能依赖于用户对程序执行内容的了解。
- 无论用户知识多么丰富，重复执行技术效率高的操作仍会影响到对性能的感觉。

1.1.2 为什么要优化

虽然优化对编写时限或者实时程序的程序员来说是合乎逻辑的选择，但是它有着更为广泛的应用。实际上所有类型的软件都可以从优化中受益。本节说明需要进行优化的四点原因：

- 当程序离开开发环境投入现场使用时，程序需要处理的数据量必定会逐步增长。最终会导致程序运行速度减慢，甚至可能无法使用。
- 仔细设计和实现的程序在将来易于扩展。设想可以在现有程序中增加功能，而原代码中的问题不会降低扩展程序的性能，真是使人受益匪浅。
- 对用户来说，使用运行快的程序工作会更加舒适。实际上，只有在速度减慢时，用户才会考虑速度问题。
- 时间就是金钱。

大家早晚一定会试探性地问：单购置速度更快的硬件不就行了吗？如果软件似乎已经不能再做任何改进，为什么不简单地升级使用运行速度更快的处理器或者更大更快的存储

器呢？数据处理速度趋向于每18个月翻一番，因此，许多在六个月或一年前就已经面临速度滞后问题的算法可能直到现在才能加速。然而，还有很多原因解释为什么始终需要优化软件。

在一个算法中，存在缺点的设计或者实现方案的运行速度很容易降低到原来的1%～10%，例如数据分类或者存储时。等待使用新硬件也“只能”使速度提高两倍，因此不是适当的解决方案。

在18个月内，程序会得到迅猛发展，程序处理的数据量也会日益增加，而且用户也希望同时运行越来越多的应用程序。这意味着程序的速度需求增长可能与硬件运行速度的增长同样快，有时甚至更快。

如果不掌握优化程序的技巧，程序员们会发现自己需要一再地进行硬件升级。

由于软件属于大销售量产品（例如，在电视机、录像机、置顶盒等中的嵌入式软件），所以成本控制非常重要。软件投资只需一次，而硬件产品每生产一次都需要投资成本。

在处理器速度越来越快而价格却越来越便宜的同时，设计也需要不断更新。这意味着升级硬件系统的其他部分也需要更多的资金投入。

某种程序系统需求越低，销量也就越好。

购置新硬件来解决软件问题只是一种临时性的处理方法，是隐藏而不是解决问题。

在谈论性能问题时需要记住一件事情：一般情况下，性能与其说是整个软件产品的问题，还不如说是软件特定部分的问题。下述部分集中讨论那些特别容易产生性能问题的程序设计部分。

1.1.3 物理设备的性能

当程序使用一台或者多台物理设备时，其中与物理设备存在交互作用的部分就可能出现性能问题。物理设备的访问速度比普通存储器慢，因为它们通常配置运动部件而且需要特殊协议才能访问。同时，不同类型的物理设备运行速度也不同。编制重要的性能解决方案时，需要确定使用物理设备的目的与程序何时和何地访问设备等问题。第12章“优化IO”更加详细地阐释了上述问题。

费时（相对而言）物理设备包括硬盘、智能卡读卡机、打印机、扫描仪、磁盘终端、CD-ROM播放器、DVD播放器与调制解调器等。

下列是一些在使用物理设备时需要考虑的事项：

（1）显而易见，一组数据使用越频繁，我们就希望把它们存放在距程序越近的位置上。因此，如果有可能，经常访问的数据应该存储在内存中。当数据集不很大而且不会变时，它甚至可以置于可执行文件中。然而，当数据集需要经常修改，或者应与其他程序共享时，

最好把它存储在网络硬盘上，在运行时载入内存。但是把它存储在网络驱动器上是不明智的，除非专门想让不同的工作站访问或者远程备份，否则只会增加无谓系统开销。选择设备时，应该重点考虑设备存储数据的将来使用情况。

（2）在程序的设计阶段，应仔细研究数据访问的时间和方式。临时复制（一块）数据，可能会提高访问数据的速度。例如，考虑下列情况：程序必须访问一台物理设备正在使用的数据。如果上述程序和设备都只读取数据而不进行修改，则不会产生任何问题。但是，当正在修改数据时，则必须使用某种锁定机制。当然，这需要消耗时间，因为上述程序和设备之间不得不相互等待。此类问题在程序设计时是可以认识到的，而且也是有可能避免的。例如，如果设备单独使用上述数据的临时副本，则程序可以继续修改数据，而物理设备此时仅仅用于输出（如果愿意，可以制作数据快照）。结果，程序和设备就不会彼此牵制了。当设备工作完成后，可以重新利用存储数据副本的内存。当涉及数据量太庞大无法完成高效复制或者二次分配内存时，仍然可以使用上述技术，只不过是针对这些数据的子集而已。

（3）当特定设备通过速度较慢的连接（例如两台计算机通过串行电缆或者调制解调器连接）与速度相对较快的设备通信时，通常理想的处理方法是在发送数据前压缩数据。当选择压缩算法时，需要确认与使用该连接速度最慢的终端压缩或者解压缩数据的耗时相比，使用该连接发送压缩数据所节省的时间是否要多。

1.1.4 系统资源性能

除了物理装置外，系统资源自身也能导致运行速度显著减慢（EPROM、ROM、RAM等）。当然，这并不一定表明硬件选择错误，但是要求我们在项目初步设计阶段和后来的实现阶段中，应注意这个问题。例如，当使用 ROM 减缓了程序运行时，可以考虑把部分 ROM 中的内容传送到 RAM 中。虽然此类复制操作会占用掉所需的 CPU 时钟周期，但是仅仅需要执行一次，而且由于响应速度加快，我们正在讨论的内存单址存取也会随之加快。显然，只有对 ROM 中使用频繁的部分，才应该考虑传送，同时，也只有当剩余内存充足时，才可以这样处理。已经说过，由于极有可能在整个程序生存期中一直使用上述内存空间，因此无论选择何种内存管理方案都不会产生碎片。详情请参阅第 9 章“高效内存管理”。

尽管限制条件更多，对于 RAM 对 CPU 寄存器存取，也可以实现类似的性能增强。大多数编译器都可以允许把变量直接存放到 CPU 寄存器中，优点是寄存器存取速度甚至比 RAM 存取速度还快。执行 RAM 存取时，CPU 必须在内部总线上为数据向内存地址映射表发送一个请求，而地址映射表必须翻译此请求，以查找相应的内存地址，然后把所包含的数值作为响应发送出去（在一些操作系统中，也可以使用内存存取间接模式）。寄存器是