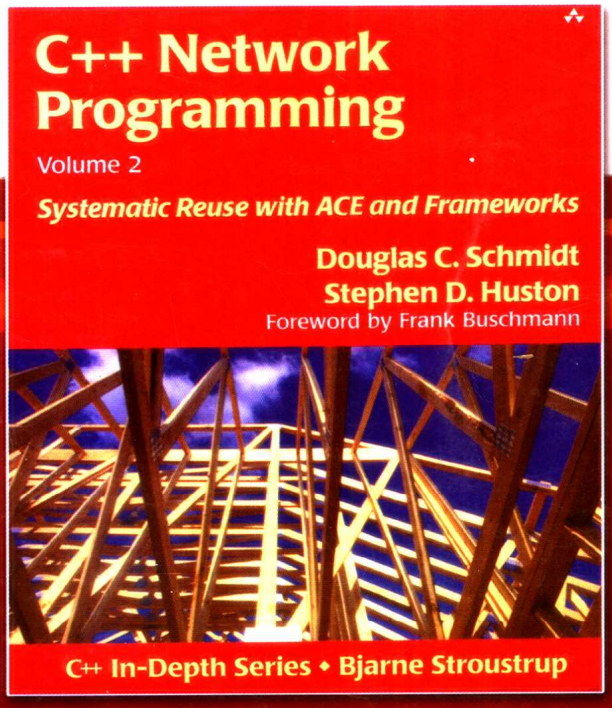
Broadview
WWW.BROADVIEW.COM.CN

C++网络编程，卷2

基于ACE和框架的系统化复用

C++ Network Programming, Volume 2

Systematic Reuse with ACE and Frameworks



Douglas C.Schmidt 著
Stephen D.Huston
马维达 译

PEARSON Addison Wesley
電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

C++网络编程，卷2
基于ACE和框架的系统化复用

C++ Network Programming ,Volume 2
Systematic Reuse with ACE and Frameworks

Douglas C. Schmidt　Stephen D. Huston　著

马维达　译

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

ACE（ADAPTIVE Communication Environment）是用于构建高性能网络化应用和下一代中间件的开放源码工具包，已在世界各地的许多软件项目中得到了广泛应用。本书是《C++网络编程》（卷1）的续篇，由ACE的创始人Douglas C. Schmidt及主要开发者之一Stephen D. Houston撰写而成，其内容涵盖了ACE中的各主要框架的基础概念、模式及使用规则。本书将向你描述这些框架的设计，以及它们可怎样帮助你克服较低级的本地操作系统API与较高级的分布式计算中间件的各种局限，高效地开发出高质量、可移植的C++网络化应用。本书是继受到了高度赞誉的POSA2（Pattern-OrientedSoftware Architecture: Patterns for Concurrent and Networked Objects）之后，Douglas C. Schmidt撰写的又一著作，通过学习本书，你将能更深入地了解和掌握适用于网络化应用开发的各种设计模式。

图书在版编目(CIP)数据

C++网络编程. 卷2, 基于ACE和框架的系统化复用 /（美）施米特（Schmidt,D C），（美）休斯顿（Huston,S.D.）著；马维达译. —北京：电子工业出版社，2004.1
书名原文：C++Network Programming V2:Systematic Reuse with ACE and Frameworks
ISBN 7-5053-9232-8

Ⅰ.C… Ⅱ.①施…②休…③马… Ⅲ.C语言—程序设计 Ⅳ.TP312

中国版本图书馆CIP数据核字（2003）第092005号

责任编辑：周 筠 张兴田
印 刷：北京市增富印刷有限责任公司
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路173信箱 邮编 100036
经 销：各地新华书店
开 本：787×980 1/16 印张：23.5 字数：490千字
版 次：2004年1月第1版 2004年1月第1次印刷
印 数：6 000册 定价：38.00元

译　序

开放源码运动的领导者之一 Eric Raymond 喜欢说这样一句话："Every good work of software starts by scratching a developer's personal itch"。这个说法显然适用于 Douglas C. Schmidt 博士的 ACE（ADAPTIVE Communication Environment）软件：当 1991 年 Douglas 开始 ACE 的开发时，其动机只是为了简化在实施其博士论文项目的过程中遇到的偶发复杂性。12 年之后，当年的"personal itch"已经演变成一个令人瞩目的面向对象框架，在其中实现了许多用于并发通信软件的核心模式；它提供的一组丰富的可复用 C++ Wrapper Facade（包装外观）和框架组件，可跨越多种平台完成通用的通信软件任务，其中包括：事件多路分离和事件处理器分派、信号处理、服务初始化、进程间通信、共享内存管理、消息路由、分布式服务动态（重）配置、并发执行和同步，等等。ACE 已在世界各地被广泛用于开发网络化应用，其中既有像 IRC 服务这样的一般应用，也有像航空控制（波音）和 PBX 监控应用（爱立信）这样的关键任务系统。而且，更让人感到振奋的是，ACE 是开放源码软件，可被自由地用于商业或非商业用途——Linux 操作系统的创始人 Linus Torvalds 曾说："Software is like sex: it's better when it's free"，你可以不同意这句话的前半部分，但是你很难不同意它的后半部分：当我们在使用 Linux 时，在使用 FreeBSD 时，在使用 Boost 时，在使用 ACE 时，我们就会更深切地感受到这一点！

《C++网络编程》系列正是帮助我们进入一个更好的网络化应用开发世界的台阶。在卷 1 中，两位作者，Douglas C. Schmidt 与 Stephen D. Huston，从对编写并发网络化应用所涉及的问题和工具进行综述开始，向我们介绍了开发灵活而高效的并发网络化应用所需的各种设计维度、模式和原则。通过对卷 1 的学习，你将能够了解怎样在将 C++和模式有效地应用于开发面向对象网络化应用的同时，增强自己的设计技能。而在卷 2 中，两位作者将向我们描述 ACE 框架的设计原理，以及它们可以怎样帮助开发者在**较低级的本地操作系统 API** 和**较高级的分布式对象计算中间件**的局限之间"航行"：前者既不灵活也不可移植，而后者对有着苛刻的 QoS 和可移植性需求的网络化应用来说，常常缺乏效率和灵活性。传统上，生成和使用网络化应用框架所需的技能被锁在专家开发者的头脑中，或是深深地埋藏在散布于企业或是行业各处的众多项目的源代码中。这两种情况当然都不理想，因为要为每个新的应用或项目重新获取这些知识，既费时又易出错。为解决这一问题，卷 2 阐释了在

ACE 框架的结构和功能之下的各种关键模式；这同时也将会帮助我们理解 ACE 自身的设计、实现，以及有效使用。

任何有价值的工作总是以他人的工作为基础的，涉及了大量“模式”的本书就更不例外。在阅读本书时，有三本非常重要的参考书籍：

[POSA1] Frank Buschmann, Regine Meunier Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture—A System of Patterns*. Wiley & Sons, New York, 1996.

[POSA2] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, Volume 2. Wiley & Sons, New York, 2000.

[GoF] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

POSA1 和 GoF 都已出版了中译本，而 POSA2 的中译本也即将出版。在无法获取这三本书的情况下，你可以访问 Douglas 的网站 http://www.cs.wustl.edu/~schmidt/，或是通过 google 来查找相关的文献资料；你也可以访问我的网站 http://www.flyingdonkey.com/，获取已译为中文的部分 ACE 文档。此外，这并非是一本写给 C++或网络编程初学者的书；如果你不熟悉 C++或网络编程的基本原理和概念，你也就不可能有效地掌握本书所传授的内容。在本书末尾的“参考文献”中，列出了许多重要的参考书籍和文档，无论是初学者还是编程老手，都可以从中获得许多有益的信息。

需要特别说明的是，本书“术语表”中的术语解释未译成中文，因为在我看来，在技术领域中，翻译只是桥梁，只是“权宜之计”——读者应该借助这样的桥梁和“权宜之计”，最终走向原著的彼岸，而“术语表”无疑就是一个很好的出发点。我希望，读者能够理解和认同我的这个想法。此外，在翻译本书的过程中，我常常会感觉到翻译的困难，也感觉到（和许多优秀的译者相比）自己翻译水平的不足——无论如何，翻译终究是一项让人不安的事业。所以，我希望读者能够通过我的网站上的论坛及时地反馈各种意见和建议。我也希望，自己能够将下一本书译得更好。

在译本完稿之际，我要感谢侯捷先生和周筠老师，他们在我困窘之时给了我极大的支持和帮助。感谢山东烟台的王建林先生，他让我更真切地体会到了人与人之间互助的可贵。也感谢卷 1 的译者於春景先生，他对本书译稿进行了审校，谢谢他的信任和与我的交流。最后要感谢本书的编辑张兴田先生，他对译稿进行了细致的校对——与他的工作相比，我已不能说翻译是枯燥或繁琐的事情。

我从 1998 年开始接触 ACE，到现在已有 5 年。能够通过翻译这本书把它介绍给中国的程序员，是一件让人高兴的事情，因为，按照 C++ User Journal 上的一篇书评中的说法：

> ... if half the claims (by the authors) are true, I expect that ACE would be the library of choice, especially if multiple hardware platforms are to be targeted.

而在我看来，两位作者在本书中所“声称”的，无疑全都是真的。

马维达　　于贵阳花溪

E-mail：weida@flyingdonkey.com

网　址：http://www.flyingdonkey.com

前　言

在网络化计算中间件领域中，ADAPTIVE Communication Environment（ACE，自适配通信环境）工具包已经取得了极大的成功。由于其所具有的灵活性、性能、平台覆盖率，以及其他一些关键属性，ACE 得到了来自网络化应用软件社群的广泛接受——它在数千种应用、数十个国家和成打的领域中得到使用就是证明。因为 ACE 是“高质量、设计良好的面向模式软件架构（Pattern-oriented Software Architecture）”的开放源码典范，所以在中间件社群之外，它也受到了相当的关注。

但 ACE 何以会如此成功？恰当地回答这一问题需要一番思量。让我们以对《*C++ Network Programming: Mastering Complexity with ACE and Patterns*》（C++NPv1）序言的反思，以及重新“启动”我的同事 Steve Vinoski 在其中介绍的公共交通系统类似物来作为开始。Steve 是对的，高质量的公共交通系统并不仅仅由飞机、机场、火车、火车站和铁轨组成，它还需要一些不那么明显的基础设施，比如调度、路线安排、售票、维护，以及监控。但即使是将所有的组成部分聚拢到一起，也仍然不足以发展出一套有效的公共交通系统；安排这些组成部分，使它们能够无缝地实现其首要目标，快速而可靠地运输旅客，也同样重要。售票点设在火车维修处或飞机机库，或是计划的与实际的调度及路线安排不为公众所知，你会使用这样的公共交通系统吗？我怀疑！

公共交通系统的成功不仅取决于对所提供的基础设施各部分的了解，它还取决于这些不同的部分怎样与它们的环境连接和集成在一起。有了这样的知识，公共交通系统的架构设计师才能够有效地将各个单独的部分集成进更高级的“积木”（Building Block）中，并将这些“积木”连接在一起。例如，售票处、问讯点、行包房和进站口被集成进位于城市中心和主要城郊中心的火车站。同样，机场常常临近大城市，并通过频繁的特别快车相连接。

即使是公共交通中心自身也要进行安排，以使各种事务能够有效地完成。例如，当你通过主入口进入火车站或机场时，你会发现票务代理、信息中心和时间表。你还会发现可以满足你旅行需求的商店。当你进入主候车室或机场中央大厅时，你会发现其他的信息中心、最新的班次信息，以及火车月台入口和登机门。因此，公共交通中心不仅要提供出发和到达所必需的服务，它们还要有效地组织内部的“控制流”。虽然大多数火车站和机场的核心结构和控制流是类似的，但它们的具体实现却可能相去甚远。但我们仍能够马上识别出这些公共交通中心的模式，因为它们符合我们通过多年经验所学到的一些不变的关键特征。

那么，在成功的公共交通系统设计和 ACE 的成功之间的关联是什么？答案很简单：除了基本的网络计算组件（Doug 和 Steve 在 C++NPv1 中介绍的各种 wrapper facade（包装外观）），ACE 还包括了一些有用的、在这些 wrapper facade 之上构建的面向对象框架（Object-Oriented Framework），并提供了一些有用的、更高级的通信服务，比如事件多路分离和分派（Event Demultiplexing and Dispatching）、连接管理（Connection Management）、服务配置（Service Configuration）、并发（Concurrency），以及分层的流处理（Stream-processing）。通过对你的应用结构和内部控制流进行有效的组织（借助于经由多年经验所掌握的关键模式），ACE 框架服务能满足许多网络化软件的需求。

ACE 框架能够带给你许多重要的好处：

- 你无需开发 ACE 已经提供的能力，这将节省相当可观的时间和精力。因此，你可以专注于你的关键责任：*实现你的客户和最终用户所需的应用功能。*
- ACE 框架使得大量的网络编程专家经验得以具体化，这些经验是 Doug、Steve，还有他们的同事在数十年中获得的。特别地，ACE 框架高效地实现了网络化应用共有的各种具有典范意义的类、类关系和控制流。世界各地的数千用户经常地测试 ACE 框架，从而产生了许多有益的修正和改进。作为一个 ACE 用户，你可以直接在你的应用中利用 ACE 框架的正确性、有效性和高效率。
- 如果一个框架不能适配（Adapted to）特定的用户需求，它就不是一个框架。这意味着你可以在网络化应用的各种关键的变化点上让 ACE 框架进行适配。例如，你可以让 ACE Reactor（反应器）框架进行适配，使用不同的事件多路分离器函数，比如 WaitForMultipleObjects()或 select()。同样，可以通过不同的 IPC 机制来配置 ACE Acceptor-Connector（接受器-连接器）框架。虽然这种可适配性自身就是有益的，ACE 还向前走了一步：你可以通过各种可用的和可互换的实现，来为许多适配工作配置合乎需要的策略。例如，除了上面所提到的不同的 Reactor 实现，ACE 还

提供了各种 IPC 机制（比如 Sockets、SSL、TLI，以及共享内存）的 wrapper facade，从而帮助你为特定的平台和应用配置 ACE Acceptor-Connector 框架。

- 最后但并非最不重要的一点是，各种 ACE 框架并非是孤立存在的。因此，你可以以各种新颖的方式来组合它们，创建各种网络化应用和全新的中间件类型。例如，你可以将 Reactor 框架与 Acceptor-Connector 框架集成在一起，从而把事件驱动应用中的连接建立和服务处理功能分离开来。使用 ACE Task（任务）框架，你同样可以将各种形式的并发引入你的应用。

多年来，我指导和领导了许多软件项目，结果发现 ACE 能极大地简化这样的任务：采用易于定制的可复用中间件来满足网络化应用的需求。不是所有的网络化应用都需要像应用服务器、Web 服务，以及复杂的组件模型这样的重量级中间件。而大多数网络化应用都可以从像 ACE 这样的可移植和高效的主机基础设施中间件（Host Infrastructure Middleware）中获益。这样的灵活性是 ACE 成功的关键，因为如果你不使用其全部功能的话，你无需采用整套的中间件。相反，你可以只组合你所需的最基本的 ACE 中间件类，创建小型的、却又足够强大的应用。因为这一原因，我预计，在今天的重量级中间件的影响衰退之后，ACE 仍将长久地为我们使用下去。

ACE 的巨大灵活性并不会让我们陷入大量不兼容的中间件实现之中。例如，如果你要构建一个嵌入式系统，它通过 CORBA Internet inter-ORB Protocol（IIOP）与外部世界进行“交谈”，你可以使用 The ACE ORB（TAO）——这是一个使用 ACE wrapper facade 和框架构建的对象请求代理（ORB），它是遵循 CORBA、开放源码和实时的 ORB。但是，如果 CORBA 对于你的应用需求来说太过度了，你可以使用适当的 ACE 类来构建定制的、且仍是可互操作的中间件。两种解决方案可以基于同样的核心结构和协议，比如 ACE Common Data Representation（CDR，通用数据表示）类和 TCP/IP Socket wrapper facade。这样它们可以无缝地互相通信，就像你乘坐从巴黎到伊斯坦布尔的列车——著名的“东方快车”——穿越许多欧洲国家，不必因为不兼容的铁路网络而更换列车。

如 Steve Vinoski 和我所指出的，在高质量的公共交通系统和高质量的网络中间件之间有许多相似之处。对于我和世界各地的其他 C++开发者来说，ACE **就是**用于构建高质量的网络中间件的工具包！不过，在说了如此之多的关于 ACE 的好话之后，让我们再回到这篇前言的主要意图上来：介绍《C++网络编程》系列的第 2 卷（C++NPv2）。如同所有的软件技术和中间件的使用一样，你对你的工具了解越多，你就越能够更好地应用它们。而在你的应用中使用 ACE 只是改善你的网络化软件的

一个方面。因此，要想显著地从 ACE 的许多优点中获益，你还需要彻底地理解在其强大的框架之下的核心概念、模式和使用规则。

许多年来，我们常常通过学习 ACE 的代码、注释，以及示例应用来学习 ACE。显然，这一过程既耗费时间，又容易出错。而且，即使你设法阅读了 ACE 中的数十万行代码，也很容易“只见树木，不见森林”。如两千年之前的希腊哲学家 Thucydides 所写下的：“一个人拥有知识，但却没有能力清晰地表达他自己，这简直就和他从来没有过任何思想一样。”

我们因而是幸运的：Doug 和 Steve 从他们繁忙的工作安排中挤出时间，撰写了这样一本高质量的关于 ACE 框架的书。使用流行的、来自 POSA[POSA1，POSA2]和“Gang of Four”[GoF]模式丛书的并发和网络模式，C++NPv2 以一种易于理解的形式解释了在 ACE 框架之下的思想和概念。这些模式又使得许多常见网络问题中有思想性的、经过了时间证明的解决方案得以具体化。例如，它们告诉你问题是什么、这些问题为何很困难、这些问题的解决方案是什么，以及这些应用于 ACE 的解决方案为何是高质量的。ACE 正在造就下一代的网络化应用软件，如果你想要全面了解 ACE 中的模式和框架，那就阅读本书吧。我已从中学到了许多东西，我确信你也将是如此。

Frank Buschmann

高级首席工程师

西门子技术公司

慕尼黑，德国

关于本书

要在当今竞争激烈、快速发展的计算技术中取得成功，网络化应用软件必须具备以下品质：

- **可负担性**（Affordability），确保软件购置（Acquistion）和发展（Evolution）的总开销不会高得惊人。
- **可扩展性**（Extensibility），支持持续的快速更新和扩展，以满足新的需求和占领新兴的市场。
- **灵活性**（Flexibility），支持范围不断增长的多媒体类型、传输模式，以及端到端（End-to-End）QoS（服务质量）需求。
- **可移植性**（Portability），减少在异种 OS（操作系统）平台上支持各种应用所需的努力。
- **可预测性**（Predictability）和**效率**（Efficiency），给对延迟敏感的实时应用提供低延迟，给带宽密集的应用提供高性能，并在低带宽网络（比如无线链接）上提供可用性。
- **可靠性**（Reliability），确保应用是健壮、容错和高度可用的。
- **可伸缩性**（Scalability），使应用有能力同时处理大量客户。

编写能够展示出这些品质的高质量网络化应用是困难的：它昂贵、复杂，而且易于出错。通过将常见的结构和功能重构（Refactor）进可复用的 *Wrapper Facade* 类库中，在《*C++ Network Programming: Mastering Complexity with ACE and Patterns*》（C++NPv1）中介绍的模式、C++语言特性，以及面向对象设计原理都有助于使网络化应用中的复杂性和错误降至最低程度。但是，如果必须为每个新项目重写大部分使用这些类库的应用软件——或者更糟，类库自身——我们就会失去复用给我们带来的各种很关键的好处。

传统上，许多网络化应用软件项目都是这样开始的：

1. 设计和实现多路分离与分派基础机制，用于处理定时事件和多个 Socket 句柄上的 I/O。
2. 在多路分离和分派层之上增加服务实例化和处理机制，并且增加消息缓冲和排队机制。
3. 使用这种特别的主机基础设施中间件来实现大量的应用特有的代码。

这样的开发过程已在许多公司中被应用了许多次，并且被许多项目同时应用。更糟的是，它还被同一团队在一系列项目中反复应用。很遗憾，这一持续的“对核心概念和代码的重新发现和发明”无谓地使整个软件开发生命周期中的开销变得高昂。今天的硬件、操作系统、编译器，以及通信平台存在着固有的多样性，更加恶化了这一问题——网络化应用软件开发的基础因此而不停地变换着。

在用于解决上述问题的技术中，*面向对象框架*[FJS99b，FJS99a]是最为灵活和强大的技术之一。框架是可复用的、“半完成”的应用，我们可以对它进行特殊化处理，以产生定制的应用[JF88]。通过把已被证明的软件设计和模式实现进具体的源代码，框架有助于降低网络化应用的开销，并改善其质量。通过强调“应用特有的类”和“不依赖于应用的类”之间的集成与协作，框架赋予了我们进行大规模软件复用的能力，其规模超出了复用独立的类或函数所能获得的规模。

在上个世纪 90 年代早期，为把模式及框架的力量和效率带给网络化应用开发，Doug Schmidt 启动了开放源码的 ACE 项目。ACE 致力于解决专业软件开发者所面临的许多现实问题，在其中 Doug 做了大量工作。接下来的 10 年间，Doug 在加利福尼亚大学 Irvine 分校、圣路易斯华盛顿大学和范德比尔特大学的开发团体，与做出了许多贡献的 ACE 用户社群及 Riverace 的 Steve Huston 一道，共同开发了一个 C++工具包；这个工具包包含了世界上最为强大的、并发的面向对象网络编程框架，并得到了广泛的使用。通过应用可复用软件模式和轻量级的 OS 可移植层，ACE 中的框架提供了同步和异步的事件处理（Synchronous and Asynchronous Event Processing）、并发和同步（Concurrency and Synchronization）、连接管理（Connection Management），以及服务的配置（Configuration）、初始化（Initialization）和层次集成（Hierarchical Integration）。

ACE 的成功从根本上改变了网络化应用和中间件在许多操作系统上的设计和实现方式，在副栏 2 中概述了这些操作系统。从大型的“财富 500 强”到小型的新兴企业，再到大学和工业实验室的高级研究项目，数千开发团队正在使用 ACE。其开放源码的开发模式和自立的文化，在精神和热情上，同 Linus Torvalds 的著名的 Linux 操作系统是类似的。

本书描述 ACE 框架的设计，以及它们可怎样帮助开发者在两种限制间“航行”：

1. **低级的本地操作系统 API** 这些 API 既不灵活，也不可移植。
2. **高级中间件** 比如分布中间件（Distribution Middleware）和通用中间件服务（Common Middleware Services）。要支持有着苛刻的 QoS 和可移植性需求的网络化应用，这些中间件常常缺乏效率和灵活性。

传统上，创建和使用网络化应用框架所需的技能被锁在专家开发者的头脑中，或是深深地埋藏在散布于企业或是行业各处的众多项目的源代码中。这两种情况当然都不理想，因为要为每个新的应用或项目重建（Reengineer）这些知识，既费时又易错。为解决这一问题，本书阐释了在 ACE 框架的结构和功能之下的关键模式[POSA2，POSA1，GoF]。我们对这些模式的涵盖也能够让开放源码的 ACE 工具包自身的设计、实现以及有效使用变得更易于为读者所理解。

目标读者

本书的目标读者是“一线”C++开发者，或是对理解怎样设计面向对象框架并应用它们来开发网络化应用有兴趣的、掌握了较高技能的学生。它是以 C++NPv1 中的材料为基础撰写的，这些材料显示了开发者可以怎样应用模式，克服随使用本地 OS API 编写网络化应用而来的复杂性。因此，在阅读本书之前，切实地领会在 C++NPv1 中所涵盖的下列主题十分重要：

- **网络化应用设计维度** 包括 C++NPv1 的第 1 章中所讨论的各种通信协议和数据传输机制。
- **Internet 编程机制** 比如 C++NPv1 的第 2 章中所讨论的 TCP/IP 连接管理和数据传输 API[Ste98]。
- **并发设计维度** 包括在 C++NPv1 的第 5 章到第 9 章中所讨论的进程和线程的使用、循环式（Iterative） vs. 并发式（Concurrent） vs. 反应式（Reactive）服务器，以及线程模型[Ste99]。
- **同步技术** 在 C++NPv1 的第 10 章中讨论，对于协调各种 OS 平台上的进程和线程的交互来说是必需的[KSS96，Lew95，Ric97]。
- **面向对象的设计和编程技术[Boo94，Mey97]** 这些技术可以简化 OS API，并在对模式的使用中避免出现编程错误——比如 C++NPv1 的第 3 章和附录 A 中所讨论的 Wrapper Facade [POSA2]和 Proxy（代理）[POSA1，GoF]模式。

ACE 各框架高度灵活而又强大，这在很大程度上应归因于它们对 C++语言特性的使用[Bja00]。因此，你应该熟悉 C++类继承和虚函数（动态绑定），以及模板（参数化类型）和你的编译器所提供的对它们进行实例化的机制。ACE 对克服各种 C++编译器之间的差异提供了大量帮助。但是，无论在何种情况下，你都须要知道你的开发工具的能力及怎样使用它们。了解你的工具能够使你更容易跟上本书中的源代码示例，并在你的系统上编译和运行它们。最后，当你阅读本书中的示例时，别忘了在副栏 7 中提到的关于 UML 图表和 C++代码的要点。

结构和内容

我们的 C++NPv1 致力于怎样克服网络化应用开发的某些复杂性，其焦点是使用 ACE 的各种 wrapper facade 来消除用 C 编写的操作系统 API 所具有的问题。本书（我们称之为 C++NPv2）将我们的焦点转移到：阐释与开发和使用 ACE 框架相关联的模式、设计技术和 C++特性的动机，并且使它们“非神秘化”。通过把已被证明的软件设计和模式实现进框架中——可以跨越项目和企业而系统地使用它们——有助于降低网络化应用的开销，并改善其质量。ACE 框架扩展了复用技术，使其远远地超出了通过复用单独的类甚或类库所能达到的程度。

本书介绍了众多的 C++应用，以通过显示怎样使用 ACE 框架的具体例子来加强各种设计讨论。这些示例提供了一步一步的指导，能够帮助你将关键的面向对象技术和模式应用到你自己的网络化应用中。本书还显示了怎样增强你的设计技能，其焦点是一些关键的概念和原理，正是这些概念和原理造就了成功的、用于网络化应用和中间件的面向对象框架的设计。

本书各章组织如下：

- 第 1 章介绍面向对象框架的概念，并显示框架与其他的复用技术有何不同——比如类库、组件、模式，以及模型集成式计算（Model-integrated Computing）。我们随即概述了在后续各章中所涵盖的、ACE 工具包中的各种框架。
- 第 2 章完成在 C++NPv1 中开始的领域分析，其中涵盖了通信协议和机制，以及网络化应用所使用的并发架构。本书的焦点在服务和配置设计维度上，这些维度处理的是一些关键的网络化应用属性，比如持续时间和结构，怎样标识网络化应用，以及它们被绑定在一起来形成完整应用的时间。

- 第 3 章描述 ACE Reactor 框架的设计和使用。该框架实现了 Reactor 模式[POSA2]，允许事件驱动应用对“从一个或多个客户递送到应用的服务请求”进行多路分离和分派。
- 第 4 章随即描述 ACE_Reactor 接口的最为常用的实现的设计和使用。该接口支持广泛的 OS 事件多路分离机制，包括 select()、WaitForMultipleObjects()、XtAppMainLoop()，以及/dev/poll。
- 第 5 章描述 ACE Service Configurator(服务配置器)框架的设计和使用。该框架实现了 Component Configurator（组件配置器）模式[POSA2]，允许应用在运行时链接它的组件服务实现，或是解除其链接，而无需静态地修改、重编译，或重链接应用。
- 第 6 章描述 ACE Task 框架的设计和有效使用。该框架可被用于实现一些关键的并发模式，比如 Active Object（主动对象）和 Half-Sync/Half-Async（半同步/半异步）模式[POSA2]。
- 第 7 章描述 ACE Acceptor-Connector 框架的设计和有效使用。该框架实现 Acceptor-Connector 模式[POSA2]，解除了网络化系统中“相互协作的对等端服务（Peer Services。参见书末“术语表”对 Peer-to-Peer 的解释——译注）的连接及初始化”与“连接一旦初始化后它们所执行的处理”的耦合。
- 第 8 章描述 ACE Proactor（前摄器）框架的设计和使用。该框架实现了 Proactor 和 Acceptor-Connector 模式[POSA2]，允许事件驱动应用高效地多路分离和分派由“异步发起的操作的完成”所触发的服务请求。
- 第 9 章描述 ACE Streams（流）框架的设计和使用。该框架实现 Pipes and Filters（管道与过滤器）模式[POSA1]，为处理数据流的系统提供了一种结构。
- 本书的最后提供了技术术语词汇表、进一步学习的参考文献列表，以及综合主题索引。

本书以前面的章节为基础来撰写后面的章节，并尽量减少提前引用。因此，我们建议你顺次阅读各章。

尽管本书阐释了 ACE 的各个最为重要的框架的关键能力，我们并没有涵盖这些框架的所有用途和方法。要更多地了解 ACE，我们推荐你阅读“ACE 程序员指南”（The ACE Programmer's Guide）[HJS]，以及通过 Doxygen[Dim01]生成的在线 ACE 参考文档。ACE 的参考文档可在 http://ace.ece.uci.edu/Doxygen/和 http://www.riverace.com/docs/找到。

相关材料

本书基于 2002 年秋发布的 ACE 5.3 版。ACE 5.3 和我们的书中描述的所有示例应用都是开放源码软件。副栏 3 说明了怎样获取 ACE 的拷贝，这样你就可以与其相伴而行，看到实际而详尽的类和框架，并且在阅读本书的同时交互式地运行各个代码示例。

要更多地了解 ACE，或是报告你在书中发现的错误，我们建议你订阅 ACE 邮件列表：ace-users@cs.wustl.edu。你可以通过向 ace-users-request@cs.wustl.edu 发送请求来进行订阅。在邮件的正文中（标题会被忽略）包括下面的命令：

subscribe ace-users [emailaddress@domain]

只有当你的消息的 From 地址不是你想要使用的地址时，你才需要提供 emailaddress@domain。如果你使用这种方法，在允许你加入列表之前，列表服务器会要求进行额外的认证步骤。

发到 ace-users 列表的消息，连同发到其他几个与 ACE 有关的邮件列表的消息，还会被转发到 comp.soft-sys.ace USENET 新闻组。如果你不需要即时地看到每天发到 ACE 邮件列表的 30 到 60 条消息，通过新闻组来阅读消息是一个追踪 ACE 新闻和活动的好办法。

发往 comp.soft-sys.ace 新闻组的消息的存档可在这里找到：http://groups.google.com/。在搜索框中输入 comp.soft-sys.ace，就可以到达已存档消息的列表。Google 有着完整的、可搜索的 ACE 存档，其中的消息超过了 40,000 条。

致　谢

评阅优胜者的荣誉归于 Alain Decamps、Don Hinton、Alexander、Maack、Chris Uzdavinis，以及 Johnny Willemsen，他们多次评阅本书，并给出了广泛而详细的意见，极大地帮助我们改进本书的形式和内容。我们还很感谢各位正式的评阅者，Timothy Culp、Dennis Mancl、Phil Mesnier，以及 Jason Pasion，他们阅读了全书并给了我们许多有益的意见。其他的许多 ACE 用户为本书提供了反馈，其中包括 Marc M. Adkins、Tomer Amiaz、Vi Thuan Banh、Kevin Bailey、Stephane Bastien、John Dilley、Eric Eide、Andrew Finnell、Dave Findlay、Jody Hagins、Jon Harnish、Jim Havlicek、Martin Johnson、Christopher Kohlhoff、Alex Libman、Harald Mitterhofer、Llori Patterson、Nick Pratt、Dieter Quehl、Tim Rozmajzl、Irma Rastegayeva、Eamonn Saunders、Harvinder Sawhney、Christian Schuhegger、Michael Searles、Kalvinder Singh、Henny Sipma、Stephen Sturtevant、Leo Stutzmann、Tommy Svensson、Bruce Trask、Dominic Williams，以及 Vadim Zaliva。

我们深深感激位于圣路易斯华盛顿大学和加利福尼亚大学 Irvine 分校的 DOC 组织的——过去的和现在的——所有成员，以及在 Rivrace Corporation 和 Object Computing Inc.的团队成员，他们开发、提炼和优化了在本书中介绍的 ACE 的许多功能。其中包括 Everett Anderson、Alex Arulanthu、Shawn Atkins、John Aughey、Luther Baker、Jaiganesh Balasubramanian、Darrell Brunsch、Don Busch、Chris Cleeland、Angelo Corsaro、Chad Elliot、Sergio Flores-Gaitan、Chris Gill、Pradeep Gore、Andy Gokhale、Priyanka Gontla、Myrna Harbibson、Tim Harrison、Shawn Hannan、John Heitmann、Joe Hoffert、James Hu、Frank Hunleth、Prashant Jain、Vishal Kachroo、Ray Klefstad、Kitty Krishnakumar、Yamuna Krishnamurthy、Michael Kircher、Fred Kuhns、David Levine、Chanaka Liyanaarachchi、Michael Moran、Ebrahim Moshiri、Sumedh Mungee、Bala Natarajan、Ossama Othman、Jeff Parsons、Kirthika Parameswaran、Krish Pathayapura、Irfan Pyarali、Sumita Rao、Carlos O'Ryan、Rich Siebel、Malcolm Spence、Marina Spivak、Naga Surendran、Steve Totten、Bruce Trask、Nanbor Wang，以及 Seth Widoff。

我们还要感谢来自 50 多个国家的数千开发者，他们 10 多年来一直在为 ACE 做出贡献。ACE 的卓越和成功是许多有才能的开发者和有前瞻性的公司的技能和慷慨支持的证明，这些开发者和公司有远见地把他们的工作贡献给了 ACE 的开放源码库。没有他们的支持、持续的反馈和鼓励，我们就不可能写出本书。为了对 ACE 开放源码社群的努力表示认可，我们在 http://ace.ece.uci.edu/ACE-members.html 列出了所有作出贡献者的名单。

我们还感谢许多同事和资助人，他们为我们在 ACE 工具包的模式和开发的研究上提供了支持；尤其是以下人士做出了许多贡献：Ron Akers(Motorola)、Steve Bachinsky(SAIC)、John Bay(DARPA)、Detlef Becker(Siemens)、Frank Buschmann(Siemens)、Dave Busigo(DARPA)、John Buttitto(Sun)、Becky Callison(Boeing)、Wei Chiang(Nokia Inc.)、Joe Cross(Lockheed Martin)、Lou DiPalma(Raytheon)、Bryan Doerr(Savvis)、Karlheinz Dorn(Siemens)、Scott Ellard(Madison)、Matt Emerson(Escient Convergence Group, Inc.)、Sylvester Fernandez(Lockheed Martin)、Nikki Ford(DARPA)、Andreas Geisler(Siemens)、Helen Gill(NSF)、Jody Hagins(ATD)、Andy Harvery(Cisco)、Sue Kelly(Sandia National Labs)、Gary Koob(DARAP)、Petri Koskelainen(Nokia Inc.)、Sean Landis(Motorola)、Patrick Lardieri(Lockheed Martin)、Doug Lea(SUNY Oswego)、Joe Loyall(BBN)、Kent Madsen(EO Thorpe)、Ed Margand(DARPA)、Mide Masters(NSWC)、Major Ed Mays(U.S. Marine Corps)、John Mellby(Raytheon)、Jeanette Milos(DARPA)、Stan Moyer(Telcordia)、Ivan Murphy(Siemens)、Russ Moseworthy(Object Sciences)、Adam Porter(U. of Maryland)、Dieter Quehl(Siemens)、Vijay Raghavan(Vanderbilt U.)、Lucie Robillard(U.S. Air Force)、Craig Rodrigues(BBN)、Rick Schantz(BBN)、Andreas Schulke(Siemens)、Steve Shaffer(Kodak)、Tom Shields(Raytheon)、Dave Sharp(Boeing)、Naval Sodha(Ericsson)、Paul Stephenson(Ericsson)、Tatsuya Suda(UCI)、Umar Syyid(Storetrax, Inc.)、Janos Sztipanovits(Vanderbilt U.)、Gautam Thaker(Lockheed