维普资讯 http://www.cqvip.com

计算机科学 2004Vol. 31№. 2

通过设计模式构造的通信应用服务框架*)

高 鹏 糜宏斌 佘 彬 孙玉芳
(中国科学院软件研究所 北京 100080)

摘 要 由于通信软件内在的复杂性,通信软件的开发者通常需要面对诸多挑战,而利用针对通信软件的设计模式可以帮助软件开发者克服困难,开发出高性能的通信软件。本篇论文将阐释我们是如何利用针对通信软件的设计模式来开发出一个基于 Linux 的通信应用服务器框架的。

关键词 设计模式,框架,通信应用服务

A Framework for Communication Application Service Using Design Patterns

GAO Peng MI Hong-Bin SHE Bin SUN Yu-Fang
(Institute of Software, The Chinese Academy of Sciences, PO. Box 8718, Beijing 100080)

Abstract Because of the inherent complexities of communication software, developers of communication software must face many challenges. But it is easier to overcome these challenges and develop high-performance communication software by using design patterns for communication software. This paper illustrates how we have applied design patterns for communication software to develop a framework for communication application service of Linux.

Keywords Design patterns, Framework, Communication application service

1 引言

现在,社会对于健壮、高性能的通信应用软件的需求一直在稳定地增长。这种类型的系统包括全球个人通信系统、网络管理平台、企业级医学成像系统、在线金融分析系统以及实时航空控制系统。通信应用软件非常适合于以下场合:通过网络和双方的互连来加强合作;通过并行处理来改善性能;通过复制来改善可靠性和可用性;通过模块化来改善可伸缩性和可移植性;通过动态配置和重配置来改善可扩展性。

尽管通信应用软件提供了如此多的好处,但它的开发仍然是昂贵的而且易错的。现在在业界流行的面向对象编程语言、组件和框架被认为是可以降低软件成本并提高软件质量的技术。面向对象的主要好处源于它对模块化和可扩展性的强调,它将易变的实现细节封装在稳定的接口后面,并且增强了软件的复用。多年来,在某些已被广泛探索的领域中,开发者已经成功地应用了面向对象技术和工具。例如,Microsoft MFC GUI 构架和 OCX 组件已成为 PC 平台上用于创建商业图形应用软件的工业标准。尽管这些工具有着自身的局限,但它们仍然显示出了复用框架和组件在生产效率方面的优势。

但是,在像电信、医学成像、航空控制和在线事务处理这样的更复杂的领域中,软件开发者很少有标准的、成型的框架模型。结果,开发者在很大程度上是从头开始构建、验证和维护通信应用软件系统,从而使得通信应用软件的开发过程变得令人难以容忍的昂贵和费时。在业界,这样的情形导致了一场"通信应用软件危机":计算硬件和网络在变小、变快、变得更为便宜;而通信应用软件的开发和维护在变大、变慢、变得更为昂贵。

构建通信应用软件的挑战源于与分布式系统相关联的固有的和非固有的复杂性[1]。固有的复杂性源于开发通信应用软件的基本问题:检测网络和主机运行状态、从网络及主机失败中恢复、减小通信响应延迟对系统的影响。

非固有的复杂性源于用以开发通信应用软件的工具和技术的局限。例如,许多标准的网络机制(例如 socket[2]和 TLI[3])和可复用组件库(例如 X Windows 和 Sun RPC)缺乏类型安全的、可移植的、可重入的和可扩展的应用编程接口(API)。同样地,通用网络编程接口(例如 socket 和 TLI)使用弱类型的整型句柄,这可能会在运行时导致微妙的错误[4]。

复杂性的另一个原因是由于典型的通信应用软件使用基于面向算法的设计和实现技术进行开发,它致使通信应用软件系统不可扩展和不可复用。

另一个重要问题是:目前的通信应用软件缺乏可扩展性,且不能在最大限度上进行复用。可扩展性可以确保那些与特定应用相关的服务可以得到及时的修改和增强。复用可以确保开发者可以有效地利用专家的领域知识、以避免重新开发和重新验证那些"反复出现的软件需求的通用解决方案"。

通过使用面向对象的设计模式和框架,开发者可以减少对通信应用软件的核心抽象概念的重新设计,因此面向对象的设计模式和框架越来越受到业界的重视。设计模式提供了一种封装设计知识的方法,这些设计知识为通信应用软件的开发问题提供了解决方案[5]。但是,这些被文档化为设计模式的抽象概念并不能直接产生可复用代码。因此,有必要通过框架的创建和使用来加强对设计模式的应用。

通过把一组相关的抽象类集成起来,并定义出这些类实例之间协作的标准途径,框架就为应用提供了可复用的软件

*)本文得到国家自然科学基金重点项目(No. 19831020)的资助。高 鹏 硕士生,研究方向为大型数据库与网络工程;糜宏斌 博士生,研究方向为大型数据库与网络工程;佘 彬 硕士生,研究方向为大型数据库与网络工程;孙玉芳 研究员,主要研究领域为系统软件和大型数据库与网络工程。

维普资讯 http://www.cqvip.com

组件[6]。在框架中，那些设计模式已经被设计实现，这样开发者就可以不必对通用通信应用组件再进行昂贵的重新发明。框架实际上是一个“半完成”的应用骨架，开发者可以通过继承和实例化框架中的可复用组件来进行定制。因为框架与那些关键任务(例如服务初始化、事件多路分离、并发控制)紧密地集成在一起，复用的范围明显地大于使用传统函数库，甚至是通常的面向对象类库。

本文笔者开发了一个通信应用服务框架，该框架通过实现通信软件的基本设计模式，包括 Acceptor-Connector、Reactor、Leader/Followers、Service Configurator，实现了通信应用软件的核心部分，形成了一个“半完成”的应用骨架，开发人员可以以此作为基础来开发应用。

2 设计模式和框架

软件复用是当前非常流行的一种用来减少开发费用的技术。复用使得开发人员可以把更大的精力投入到那些更为关键的专业领域中去，而不是重复前人所做过的工作，设计模式和框架就是这样两种可复用的技术。设计模式的概念最先来自于城市建筑专家 Christopher Alexander 对建筑模式的定义“每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动”[7]。该思想同样适用于面向对象的设计模式。框架是整个或部分系统的可重用设计，是一个可复用的设计构件，它规定了应用的体系结构，阐明了整个设计、协作构件之间的依赖关系、责任分配和控制流程，表现为一组抽象类以及其实例之间协作的方法，它为构件复用提供了上下文(Context)环境。因此，通过应用设计模式和框架可以帮助开发人员将新的设计建立在以往工作的基础上，复用以往成功的设计方案。

3 框架概述

通信应用服务框架提供的功能 为了满足通信应用软件的需要，通信应用服务框架必须提供以下这些通信应用的核心功能：

- 事件多路分离和分发
- 连接的建立
- 服务的初始化
- 进程间通信
- 内存及缓存管理
- 通信服务的动态配置
- 并发/并行和同步

在这些功能中，我们已经实现的有事件多路分离和分发、连接的建立、服务的初始化、通信服务的动态配置、并发/并行和同步；而对于内存及缓存的管理这一项还有待实现，但是这一项并不影响整体的应用，只是对整体的性能有一定的影响，因此如果今后能够实现这一部分将对整体性能的提高有很大的帮助。

通信应用服务框架的体系结构 如图 1，我们的通信应用服务框架采用了分层的体系结构。图 1 显示了各个组件间的纵向和横向关系。根据框架的需要，我们将该框架分为三个层次：

- 操作系统适配层：操作系统适配层是把底层操作系统提供的应用编程接口函数改写成一种统一的格式，便于上层的应用，同时便于出错处理和平台无关性的实现，它是函数一级的实现。

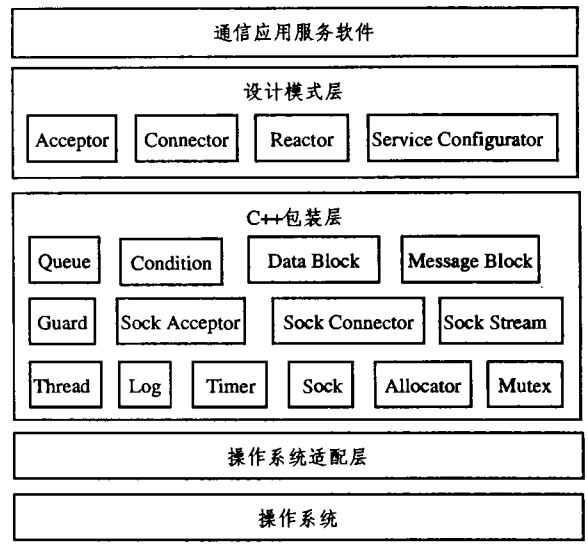

图 1 通信应用服务框架的体系结构

- C++包装层：C++包装层是把操作系统适配层所提供的那些函数分类并封装成 C++类，它就相当于操作系统提供了一组 C++类应用编程接口，而不是原来的 C 应用编程接口，它是数据结构和方法一级的实现。
- 设计模式层：设计模式层是实现了通信应用服务框架中的一些有效的、典型的设计模式，并形成了一个框架，通信应用服务软件的开发人员可以通过继承或实例化框架中的类，实现其中与具体应用相关的方法，就可以开发出自己的通信应用服务软件，它是对设计理念层次的实现，是最高层次的实现。

对于开发人员来说，他们可以利用所有这三层，但是通常来说，最下面的操作系统适配层，他们是不能进行修改的，因为这样将有可能改变该框架的平台无关性，同时会极大地影响上面两层的实现。对于 C++包装层来说，由于我们所提供 C++包装类中的方法几乎都是虚拟的，因此开发人员可以通过继承这些包装类来实现自己的 C++类及类中的方法。而对于设计模式层，那些模式的实现逻辑通常是无需修改的，用户需要改写的是框架中的那些虚拟方法，来实现与应用相关的处理。我们下文将对这三层进行详细的讲解。

4 操作系统适配层

在这一层中，需要解决的问题主要是对底层操作系统所提供的系统调用的封装。由于操作系统提供的系统调用有以下几个问题，所以我们不得不对它进行改造：

函数的输出格式不统一：有的输出的是操作的结果，有的输出的是错误代码，这为编程带来了极大地不便。为此，我们的做法是把所有函数的输出格式统一化：输出 0 代表操作成功，否则输出的为错误码，如果操作有输出的话，那么该输出是作为地址引用的输入参数传给该函数的。

函数的输入参数没有进行检错处理：例如有的指针为空，操作将会失败，但原函数并未指出失败的原因在于指针为空，因此，对函数的每个输入参数进行检查，并进行相应的出错处理，将会为开发人员带来极大的方便。

为了解决以上这些问题，我们对这些函数进行了如下的处理：

维普资讯 http://www.cqvip.com

改变参数:增加了一个参数作为输出结果;

对输入参数进行出错处理;

改变返回类型:当成功时,返回 0;否则返回错误码。

我们基本上对所有的系统调用都提供了类似的改造,通过建立操作系统适配层,我们为上层的开发人员提供了比原先操作系统友好的多的 API:

更为健壮的编程接口:把操作系统提供的函数接口统一化,这降低了发生编程错误的潜在可能性。

更为完善的错误处理:完善的错误处理使得上层的开发人员可以更容易找到发生错误的函数及其原因,从而可以极大地提高开发的效率。

5 C++包装层

通信应用服务软件通常使用本地操作系统所提供的低级函数和数据结构来编写。虽然这是一种被普遍采用的方法,但是由于它不能解决以下问题,通常会给应用开发者造成许多问题:

繁琐、不健壮的程序:如果直接使用本地操作系统所提供的低级函数和数据结构来进行编程,那么应用开发者不得不反复地重写大量冗长乏味的软件逻辑。一般而言,编写和维护很乏味的代码常常含有微妙而有害的错误,从而使程序在运行时发生不可预知的错误。

缺乏可移植性:由于直接使用底层函数和数据结构编写的软件无法隐藏那些与底层函数和数据结构相关的信息,因此通过它们编写出来的程序通常不能在不同的操作系统和编译器间进行移植的,有时甚至不能在同一操作系统或编译器的不同版本间进行移植。

维护的高开销:C 和 C++开发者通常通过使用 #ifdef 在他们的应用源码中显式地增加条件编译指令来获得可移植性。但是,使用条件编译来处理平台特有的变种在各方面都增加了应用源码的物理设计复杂性[8]。由于平台特有的实现细节分散在应用源文件的各个地方,因此开发者难以维护和扩展这样的软件。

由于有这些缺点,直接使用低级函数和数据结构进行编程来开发应用常常并非是有效的设计选择。为了解决以上的问题,也就是要确保应用不直接访问低级函数和数据结构,我们通过 Wrapper Facade Pattern[9]设计了 C++包装层,用它把上层应用同底层函数和数据结构隔离开来。包装外观模式是封装低级函数和数据结构的有效途径,它通过面向对象的类接口来封装低级函数和数据结构。

例如,我们与 Sock 相关的函数按照功能分为以下几个类:

· 基类:SOCK 类,它提供所有其他类共有的机制,比如打开和关闭本地通信端点,以及处理选项(像选择 socket 队列大小及启用组通信)。

· 连接建立:SOCK_Acceptor 类——被动地建立新通信端点、以响应主动连接请求;SOCK_Connector 类——主动地建立新通信端点。

· 流通信:SOCK_Stream 类——由上面描述的 SOCK_Acceptor 或 SOCK_Connector 创建,为在两个进程间传输数据提供机制,它在驻留在不同主机上的进程间交换数据。

· 数据报通信:SOCK_Dgram 类——为运行在本地和远端主机上的进程间提供数据报交换机制。

· 组通信:SOCK_Dgram_Mcast 类——将 UDP 数据报多点传送给运行在本地子网中的本地和/或远端主机上的进程;SOCK_Dgram_Bcast 类——将 UDP 数据报广播给(1)所有与主机相连的网络接口,或是(2)一个特定的网络接口如图 1,我们按以上方法设计的 C++包装类有:Sock 类属、Thread 类、Thread_Manager 类、Lock 类属、Guard 类等等。

6 设计模式层

对于通信应用服务,我们必须要实现以下几个模块:连接的建立、事件的分离分发、服务的初始化以及多线程的并发策略,这几个模块式是所有通信应用服务软件所必需的,而且通常是与具体的应用不相关的,因此,我们通过设计模式 Accept-or-Connector[10]、Reactor[11]、Leader/Followers、Service Configurator[12]实现了以上这些模块,这些模块构成了通信应用服务的基本框架,实现了通信应用服务的基本运行机制。通信应用服务的开发者可以直接利用这个框架,无需编写那些与应用无关的部分,只需通过继承框架中抽象的事件处理器类,实现与具体应用相关的事件处理,就可以构成一个通信应用服务软件。

图 2 是设计模式层的结构图,如图所示,开发人员需要进行的工作就是编写图中所示的那些具体事件处理器(Concrete Connector、Concrete Acceptor、Concrete Service Handler)以及其他一些与应用相关的附加模块。图中其他的部分是通信应用服务框架所提供的,开发人员无需对这些代码进行修改。当那些具体的事件处理器编写好之后,一个通信应用服务软件就形成了。

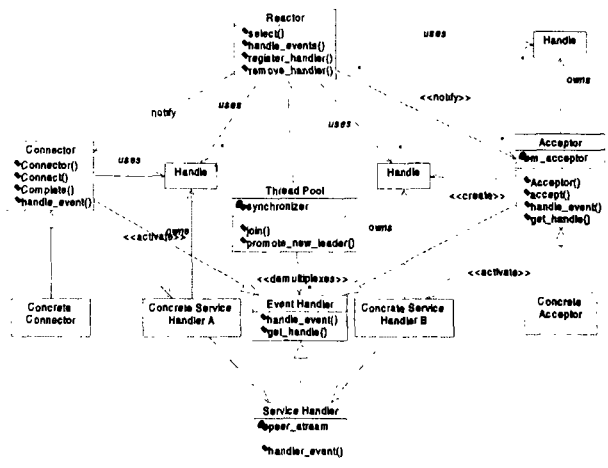

图 2 设计模式层整体结构

· Acceptor-Connector 模式 该模式把连接建立与连接一旦建立后所执行的服务分离开来。Acceptor 用于创建并初始化被动连接端点,来被动地在特定地址上侦听来自远端 Connector 的连接请求;Connector 用于主动地发起到远端 Acceptor 的连接。当 Acceptor 和 Connector 连接成功后,它们就初始化相应的服务处理器来处理在连接上交换的数据。服务处理器一旦被初始化,就开始执行与应用相关的处理,而不再与 Acceptor 和 Connector 进行交互。

· Reactor 模式 该模式把同步事件多路分离及事件分发与为响应事件而执行的服务分离开来。它使用了系统提供的 select 系统调用来监测事件源,通过建立一个事件、事件源与事件处理器之间的一个对应关系来分发事件。本框架通过使用 Reactor 模式把与应用无关的分发、分离机制同与应用

维普资讯 http://www.cqvip.com

相关的事件处理器分割开来，提高了软件重用性、移植性和灵活性。

· Leader/Followers 模式　该模式是一种性能极佳的同步模型，在这个模型中，核心是一个线程池，线程池中的线程通过一个同步装置(信号量)来交替地占有一组事件源来进行监测、分离、分发以及事件处理。

· Service Configurator 模式　该模式把系统中个体组件的实现与它们被配置进系统的时间分离开来。通信应用服务框架使用服务配置器模式来在安装时或运行时动态地优化、控制及重配置服务策略。

设计模式层是通信应用服务框架中最重要的一层：我们通过 Acceptor-Connector、Reactor、Leader/Followers 和 Service Configurator几种设计模式实现了通信应用服务的基本组成部分。这样使得开发人员只需关心那些与具体应用相关的处理，使得开发效率大为提高。

7　应用

笔者根据通信应用服务框架成功地构造了一个应用级网关。该网关将作为底层基础应用于金融业务中。

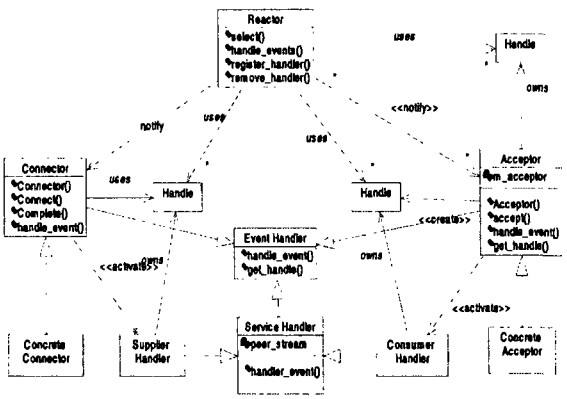

图 3　网关结构图

如图 3 所示的网关结构图，与图 2 进行比较可以看到，我们只是在原框架的基础上实现了两个主要的事件处理器 Supplier Handler 和 Consumer Handler，它们实现的是与消息格式和网关的路由协议有关的应用特有的细节。从网关这个应用可以看到：开发通信应用服务软件无需再开发那些与应用无关的组件，直接使用通信应用服务框架提供的组件即可，开发人员需要开发的是那些与应用相关的部分，这些功能可以通过继承被加入到通信应用服务框架中，从而可以很方便地实现一个通信应用服务软件。

总结　本文描述了如何通过设计模式构造一个通信应用服务框架，并通过此框架设计通信应用服务软件。可以看到，设计模式和框架可以帮助通信应用服务的开发人员克服通信软件中那些常见的困难，从而提高开发的效率，并构造出易维护、通常也是高性能的通信软件。

参 考 文 献

1　Brooks F P. The Mythical Man-Month. Reading, MA: Addison-Wesley, 1975
2　Leffler S J, McKusick M, Karels M, Quarterman J. The Design and Implementation of the 4. 3BSD UNIX Operating System. Addison-Wesley, 1989
3　Rago S. UNIX System V Network Programming. Reading, MA: Addison-Wesley, 1993
4　Schmidt D C, Harrison T H, Al-Shaer E. Object-Oriented Components for High-speed Network Programming. In: Proc. of the 1st Conf. on Object-Oriented Technologies and Systems, (Monterey, CA), USENIX, June 1995
5　Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995
6　Johnson R, Foote B. Designing Reusable Classes. Journal of Object-Oriented Programming, 1988, 1: 22～35
7　Alexander C, et al. A Pattern Language. Oxford University Press, New York, 1977
8　Lakos J. Large-scale Software Development with C++. Reading, MA: Addison-Wesley, 1995
9　Schmidt D C. Wrapper Facade: A Structural Pattern for Encapsulating Functions within Classes: [C++ Report]. Feb. 1999
10　Schmidt D C. Acceptor and Connector: Design Patterns for Actively and Passively Initializing Network Services. Workshop on Pattern Languages of Object-Oriented Programs, Aarhus, Denmark: ECOOP '95, Aug. 1995
11　Schmidt D C. Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching. in Pattern Languages of Program Design (J. O. Coplien and D. C. Schmidt, eds.), MA: Addison-Wesley, 1995
12　Jain P, Schmidt D C. Service Configurator: A Pattern for Dynamic Configuration of Services. In: Proc. of the 3rd Conf. on Object-Oriented Technologies and Systems, USENIX, June 1997

(上接第 29 页)

3　Leland W, et al. On the Self-Similar Nature of Ethernet Traffic. IEEE/ACM Transactions on Networking, Feb. 1994
4　Paxson P, Floyd S. Wide-Area Traffic: The Failure of Poisson Modeling. IEEE/ACM Transactions on Networking, 1995, 3: 226～244
5　Crovella M, Bestavros A. Self-similarity in world wide web traffic: Evidence and possible causes. IEEE/ACM Trans- actions on Networking, 1997, 5(6): 835～846
6　Garrett M W, Willinger W. Analysis, modeling, and generation of self-similar VBR video traffic. In: Proc. of ACM SIGCOMM, 1994. 269～280
7　Duffy D E, McIntosh A A, Rosenstein M, Willinger W. Statistical analysis of CCSN/SS7 traffic data from working CCS subnetworks. IEEE Journal on Selected Areas in Communications, 1994, 12(3)
8　Taqqu M S, Teverovsky V. On Estimating the Intensity of Long-Range Dependence in Finite and In nite Variance Time Series. In: R. J. Alder, R. E. Feldman, M. S. Taqqu, eds. A Practical Guide to Heavy Tails: Statistical Techniques and Applications, Birkhauser, Boston, 1998. 177～217
9　Sikdar B, Chandrayana K, Vastola K S, Kalyanaraman S. Queue Management Algorithms and Network Traffic Self-Similarity. In: Proc. of IEEE HPSR, Kobe, Japan, May 2002. 319～323
10　Deng S. Empirical model of WWW document arrivals at access link. in ICC, 1996
11　Mah B. An empirical model of http network traffic. In: Proc. INFOCOM '97, Apr. 1997
12　Feldmann A, Gilbert A C, Huang P, Willinger W. Dynamics of IP traffic: A study of the role of variability and the impact of control. SIGCOMM'99, 1999
13　Popescu A. Traffic Self-Similarity, (invited) tutorial. In: the IEEE Intl. Conf. on Telecommunications, ICT2001, Bucharest, Romania, June 2001
14　Barakat C, Thiran P, Iannaccone G, Diot C. On Internet backbone traffic modeling. ACM Sigmetrics, Marian del Rey, June 2002 (extended abstract)
15　Peixin S, Youyue S, Zhenming L. A Study on TCP Throughput Stabilization in Slow Start Phase. Journal of Computer Engineering and Application. Adopted, to be released on 12th issue, 2003
16　Olsén J, Kaj I. Slowstart Window Modeling for Single Connection TCP-Tahoe: [U. U. D. M Report]. Department of Mathematics, Uppsala University, 2001. 21
17　Postel J. Transmission control protocol. RFC793, Sep. 1981
18　Hoe J. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In: Proc. of ACM SIGCOMM'96, Stanford, CA, Aug. 1996. 270～280