ACE 中设计模式在分布式网络服务中的应用研究

刘益志
(四川大学 吴玉章学院, 成都 610041)

摘　要：以网关应用服务的构建为例,详细探讨如何利用 Acceptor-Connector 模式来构建分布式网络服务的过程。通过分析其构造过程可知,利用 Acceptor-Connector 模式可提高程序的灵活性、可扩展性,减少重复开发代码工作量。

关键词：自适应通信环境；设计模式；Acceptor-Connector；分布式网络服务
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2014)05-1548-03
doi:10.3969/j.issn.1001-3695.2014.05.064

Research on application of design patterns of ACE in distributed network services

LIU Yi-zhi
(*College of Wuyuzhang Science*, *Sichuan University*, *Chengdu* 610041, *China*)

Abstract: In view of the example of construction of gateway application service, this paper studied the procedure of how to develop the distributed network services by using Acceptor-Connector pattern deeply. Through the procedure of the construction, it proved that the Acceptor-Connector pattern could help to improve software's flexibility and extensibility, and it also could decrease the coding workload.

Key words: ACE(adaptive communication environment); design patterns; Acceptor-Connector; distributed network services

ACE 是基于 C++ 的一个开源通信软件开发框架。它适用于并发通信软件的开发环境,可跨平台满足通用的通信软件应用需求。目前,ACE 的应用领域遍及应用网络通信的各个行业[1~4],能实现高性能实时的通信服务和应用,除此之外,ACE 还能增强通信软件的可移植性,提高通信软件的灵活性、可复用性和可扩展性。使用 ACE 来开发通信软件有如下优势[5]:

a)提高软件质量。ACE 的设计模式能够提高软件的灵活性、可复用性、可扩展性和模块化。

b)良好的可移植性和跨平台特性。ACE 使用 C++ 开发,且代码开发相当规范,所以具有良好的可移植性,且能跨平台。

通信系统越来越多地采用分布式架构。尽管分布式计算为通信系统开发带来很多优势,但开发通信系统还是极容易出错,这主要是由于分布式架构固有的复杂性所致。这些复杂性主要表现在其缺乏可扩展性和重用性。由于面向对象的设计模式能够提供一种对设计知识的封装方法,所以它能有效解决分布式架构的复杂性。例如,设计模式能够用于描述微体系结构中的 recurring。微体系结构是通用对象结构的抽象,其在分布式通信软件的构建过程中是先决条件[6]。然而,抽象的模式文档不能直接生产可重用的代码,因此需要对设计模式进行深入研究,使其能适用于分布式网络服务的构建工作。

1　ACE 中的设计模式

1.1　ACE 介绍

ACE 是一个面向对象、开源的开发框架,它实现了通信软件的基本应用和模式。它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[7]。ACE 中组件的层次结构如图 1 所示。

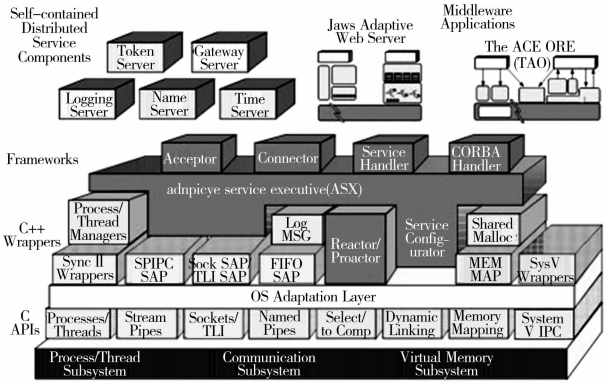

图 1　ACE 框架中组件的层次结构

ACE 工具包采用层次体系结构设计。ACE 的底层是封装了现存操作系统网络编程机制的面向对象的 wrappers,高层则扩展了底层的 wrappers,以提供面向对象的框架和组件,这些框架和组件覆盖了面向应用程序的绝大多数网络任务和服务。Wrappers 上面是 ACE 操作系统适配层,它包括以下操作系统机制,即多线程和同步、进程间通信、事件分离、直接动态链接、内存映射文件和共享内存。在操作系统适配层之上是面向对象的 wrappers,这些 wrappers 封装了程序并发机制、进程间通信和虚拟内存管理机制。应用程序可以通过有选择地继承、聚合和实例化 ACE 包的分类来组织这些组件。面向对象的 wrappers 使用通过类型安全的面向对象接口封装操作系统的通信、并发和虚拟内存机制,提高了应用程序的健壮性。这样避免了应用程序直接访问操作系统库[8]。

ACE 包含一个高级的网络编程框架。它集成并加强了底层 OS wrappers,这个框架支持由应用程序服务组成的动态配置和并发网络后台程序。ACE 的框架部分包含以下分类[9]:a)Reactor,ACE reactor 提供了可扩展的、面向对象的多路输出,根据不同事件的类型进行处理器调度;b)Service configura-

收稿日期：2013-07-09；**修回日期**：2013-08-23
作者简介：刘益志(1994-),男,四川成都人,本科,主要研究方向为移动互联网开发应用(644073664@qq.com).

tor,ACE service configurator 支持在安装或运行时配置服务的应用程序结构;c)Streams,ACE streams 组件简化了包含一个或多个层次相关的服务并发通信软件的开发。

1.2　设计模式

1)Reactor 模式　ACE 提供了 ACE_Reactor 对各种类型的事件进行多路分离并进通行相应的处理。同时,为了保证应用的可移植性,无论是在何种操作系统上,也无论底层使用了何种事件多路分离机制,ACE_Reactor 所提供的接口都是一样的。

2)Proactor 模式　当操作系统平台支持异步操作时,一种高效而方便地实现高性能 Web 服务器的方法是使用 ACE_Proactor 模式。使用 ACE_Proactor 模式设计的 Web 服务器通过一个或多个线程控制来处理异步操作的完成。通过集成完成事件多路分离和事件处理器分派,ACE_Proactor 模式简化了异步的 Web 服务器。

3)Strategy 模式　在 ACE 中,Strategy 模式定义了不同算法的接口,分别封装起来,让它们彼此之间可以互相替换,其优点在于系统中容易发生变化的部分与稳定的部分相互独立,该模式对于后续的维护是非常有用的。使用 Strategy 可以应对不稳定的需求,对于经常需要需求变更的部分,可以将改变的部分限制在 Strategy 内,避免在系统中修改。

4)Acceptor-Connector 模式　在 ACE 中,Acceptor-Connector 模式借助名为 ACE_Acceptor 的 Factory 实现。Factory 是用于对助手对象的实例化过程进行抽象的类。Factory 允许一个对象通过改变它所委托的对象来改变它的底层策略,而 Factory 提供给应用的接口却是一样的,因此无须对客户代码进行改动。

2　基于设计模式构建可扩展的分布式网络服务

2.1　问题的提出

网关是网络服务中最为普遍的应用之一,它是一种充当转换重任的计算机系统或设备。网关可以用于广域网互连,也可以用于局域网互连。网关一般采用基于连接的进程间通信机制来传递数据。如图 2 所示,网关所联系的 peers 区域为对等,也就意味着通过网关有着许多不一样的连接和通信服务,其传递方向可以是相互的。而传统的网关应用中,在建立连接时,应在网关和 peers 区域按照主/被动模式建立,即网关必须采用被动模式,peers 区域需采用主动模式。这就带来一个在程序设计时缺乏灵活性和扩展性的问题。当面对各种不同的应用时,需要设计各种不同的应用程序,也缺乏重用性。此外,使用 socket 或是 TLI 这种较为底层的方式进行网络编程时会出现类型检查等问题,这会导致程序在运行时出错。所以,应借助设计模式所拥有良好的灵活性、可扩展、可封装等特点来有效解决这些问题。

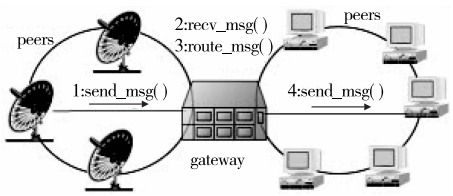

图 2　多服务应用级网关

2.2　构建可扩展的应用网关服务

Acceptor-Connector 模式的运行结构大致划分为三个阶段,即端点或连接初始化阶段(endpoint initialization phase)、服务初始化阶段(service activation phase)、服务处理阶段(service processing phase)[5]。图 3 为 ACE 中 Acceptor-Connector 模式的运行体系。

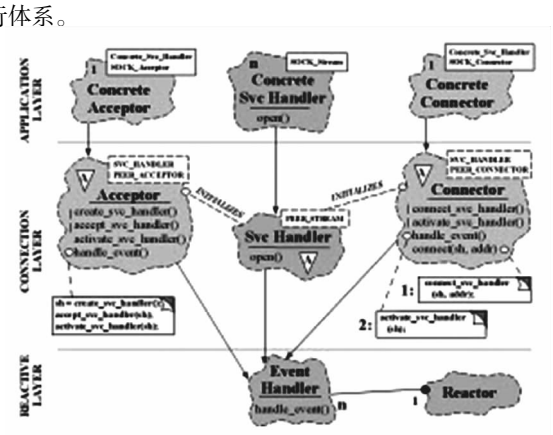

图 3　Acceptor-Connector 模式的运行体系

下面详细说明构建可扩展的应用级网关服务的过程。

1)端点或连接初始化阶段　在使用 Acceptor 的情况下,网关应用程序可以调用 ACE_Acceptor 中 Factory 的 open()方法,或是用它的缺省构造器来开始被动侦听连接。当 Acceptor 工厂的 open()方法被调用时,如果 Reactor 单体还没有被实例化,open()方法就首先对其进行实例化,然后调用底层具体 Acceptor 的 open()方法完成必要的初始化来侦听连接。例如,在使用 ACE_SOCK_Acceptor 的情况中打开 socket,将其绑定到用户想要在其上侦听新连接的端口和地址上,在绑定端口后,它将会发出侦听调用。Open 方法随后将 Acceptor 工厂登记到 Reactor,因而在接收到任何到来的连接请求时,Reactor 会自动回调 Acceptor 工厂的 handle_input()方法。正是因为这一原因,Acceptor 工厂才从 ACE_Event_Handler 类层次派生,这样它才可以响应 Accept 事件,并被 Reactor 自动回调,提高程序开发的灵活性。在使用 Connector 的情况中,网关应用程序调用 Connector 工厂的 connect()方法来发起到对端的连接,除了其他一些选项,这个方法的参数包括程序想要连接到的远地地址,以及同步及异步方式。程序可以同步或异步进行连接,如果连接请求是异步的,ACE_Connector 会在 Reactor 上登记自己,等待连接被建立。一旦连接被建立,Reactor 将随即自动回调连接器。但如果连接请求是同步的,connect()调用将会阻塞,直到连接被建立或超时到期为止。Acceptor-Connector 模式初始化流程如图 4 所示。

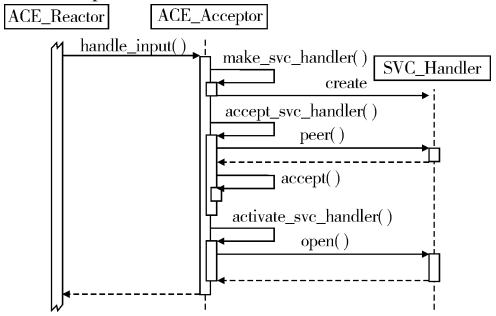

图 4　Acceptor-Connector 模式初始化流程

2)Acceptor 服务初始化阶段　在监听到端口上的连接需求后,Reactor 自动回调 ACE_Acceptor 工厂的 handle_input()方法。该方法是一个模板方法(template method)。模板方法用于定义一个算法的若干步骤的顺序,并允许改变特定步骤的执行。这种变动是通过允许子类定义这些方法的实现来完成的。

在网关应用程序中,模板方法将算法定义如下:

```
make_svc_handler();//创建服务处理器
accept_svc_handler();//将连接前一步创建的服务处理器
activate_svc_handler();//启动这个新服务处理器
```

这些方法都可以被重新编写,从而灵活地决定这些操作怎样来实际执行。这样,handle_input()将首先调用 make_svc_han-

dler( ) 方法,创建适当类型的服务处理器。在服务处理器被创建后,handle_input( ) 方法调用 accept_svc_handler( ) 。该方法将连接服务处理器。在 ACE_SOCK_Acceptor 被用做具体 Acceptor 的情况下,它调用 BSD accept( ) 例程来建立连接。在连接建立后,连接句柄在服务处理器中被自动设置,这个服务处理器是通过调用 make_svc_handler( ) 创建的。该方法也可被重载,以提供更复杂的功能。Handle_input 函数的伪代码如下:

```
ClientService: : handle_input ( ACE_HANDLE)
{   const size_t INPUT_SIZE = 4096;
    cher buffer[ INPUT_SIZE] ;
    ssize_t recv_cnt,  send_cnt;
    recv_cnt = this→peer ( ) . recv ( buffer,  sizeof ( buffer) ) ;
    if ( recv_cnt < = 0)
     {   ACE_DEBUG (( LM_DEBUG, ACE_TEXT ( " (% P|% t) Connection closed\n") ) ) ;
         return - 1;
     }
  send_cnt = this→peer ( ) . send ( buffer, ACE_static_cast ( size_t, recv_cnt) ) ;
  if ( send_cnt = = recv_cnt)
    return 0;
  if ( send_cnt == - 1 && ACE_OS: : last_error ( ) ! = EWOULDBLOCK)
    ACE_ERROR_RETURN (( LM_ERROR,
                        ACE_TEXT ( "(% P|% t)  % p\n") ,
                        ACE_TEXT ( "send") ) , 0) ;
  if ( send_cnt = = - 1)
    send_cnt = 0;
  ACE_Message_Block  * mb;
  size_t remaining = ACE_static_cast ( size_t,  ( recv_cnt - send_cnt) ) ;
  ACE_NEW_RETURN
      ( mb,  ACE_Message_Block ( &buffer [ send_cnt] ,  remaining) ,  - 1) ;
  int output_off = this→msg_queue ( ) →is_empty ( ) ;
  ACE_Time_Value nowait ( ACE_OS: : gettimeofday ( ) ) ;
  if ( this→putq ( mb,  &nowait) = = - 1)
    { ACE_ERROR (( LM_ERROR,
                   ACE_TEXT ( "% P|% t)  % p;  discarding data\n") ,
                   ACE_TEXT ( "enqueue failed") ) ) ;
      mb→release ( ) ;
      return 0;
    }
  if ( output_off)
    return this→reactor ( ) →register_handler( this,  ACE_Event_Handler: : WRITE_MASK) ;
  return 0;
  }
```

3) Connector 服务初始化阶段　应用发出的 connect( ) 方法与 Acceptor Factory 中的 handle_input( ) 相类似,也即它是一个模板方法。在构建网关服务的应用中,模板方法 connect( ) 定义下面一些可被重定义的步骤:

```
make_svc_handler( ) ;//创建服务处理器
connect_svc_handler( ) ;//将连接接收进前一步创建的服务处理器
activate_svc_handler( ) ;//启动这个新服务处理器
```

每一方法都可以被重新编写,从而灵活地决定这些操作怎样来实际执行。这样,在应用发出 connect( ) 调用后,连接器的 Factory 通过调用 make_svc_handler( ) 来实例化恰当的服务处理器。在服务处理器被创建后,connect( ) 调用确定连接是要成为异步的还是同步的。如果是异步的,在继续下一步前,它将自己登记到 Reactor,随后调用 svc_handler( ) 方法。在使用 ACE_SOCK_Connector 的情况下,这意味着将适当的选项设置为阻塞或非阻塞式 I/O,然后发出 BSD connect( ) 调用。如果连接被指定为同步的,connect( ) 调用将会阻塞,直到连接完全建立。在这种情况下连接建立后,它将在服务处理器中设置句柄来与它现在连接到的对端通信。在服务处理器中设置句柄后,连接器模式将进行到最后阶段。如果连接被指定为异步的,在向底层的具体连接器发出非阻塞 connect( ) 调用后,对 connect_svc_handler( ) 的调用将立即返回。在使用 ACE_SOCK_Connector 的情况中,这意味着发出非阻塞式 BSD connect( ) 调用。在连接稍后被实际建立时,Reactor 将回调 ACE_Connector Factory 的 handle_output( ) 方法,该方法在通过 make_svc_handler( ) 方法创建的服务处理器中设置新句柄,然后 Factory 将进行下一阶段。与 accept_svc_handler( ) 类似,connect_svc_handler( ) 是一个桥接方法,可进行重载和扩展,减少重复开发代码的工作量。

4) 服务处理　一旦服务处理器被创建、连接被建立以及句柄在服务处理器中被设置,ACE_Acceptor 的 handle_input( ) 方法将调用 activate_svc_handler( ) 方法。该方法将随即启用服务处理器。其缺省行为是调用作为服务处理器入口的 open( ) 方法。在服务处理器开始执行时,open( ) 方法是第一个被调用的方法,程序调用 activate( ) 方法来创建多个线程控制,并在 Reactor 上登记服务处理器,这样当新的数据在连接上到达时,它会被自动回调。该方法也是一个桥接方法,可被重载以提供更为复杂的功能。这个重载、可复用的方法可以提供更为复杂的并发策略,能为分布式应用服务提供丰富的开发功能。服务处理流程如图 5 所示。

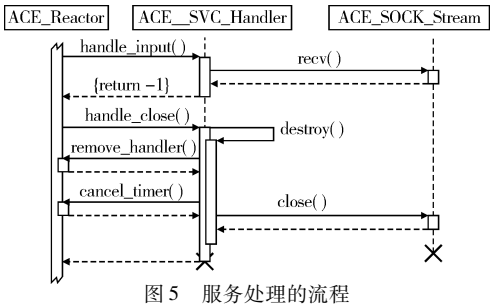

图 5　服务处理的流程

3　结束语

ACE 能广泛地应用于分布式网络应用中的原因就在于其采用多种设计模式。本文详细探讨了如何利用 Acceptor-Connector 模式来构建分布式网络服务的过程。当然,ACE 中的其他设计模式,如 Proactor 和 Strategy 模式等也均可以用来构建分布式网络服务。后续的研究工作可结合云计算中分布式网络应用的新特点[10,11]来探讨 ACE 的应用新模式。

参考文献:

[1] SCHMIDT D C, STAL M, ROHNERT H, *et al*. Pattern-oriented software architecture: patterns for concurrency and distributed objects, volume2 [M]. New York: Wiley, 2000.

[2] 齐恒. 利用自适应通信环境构建消息通信平台[J]. 新疆师范大学学报:自然科学版,2009,25(3):75-80.

[3] 李冠宇,谢康林. 基于 ACE 开发环境的 NAT/FW 穿透算法的研究[J]. 计算机应用与软件,2007,24(11):133-135.

[4] 李宜达. 基于 ACE 的分布式服务器集群系统框架的设计与实现[D]. 成都:电子科技大学,2009.

[5] BUSCHMANN F, MEUNIER R. Pattern-oriented software architecture:a system of patterns [M]. New York:Wiley, 1996.

[6] 肖万斌,任雄伟. 基于 ACE 设计模式的海军 EWS 系统设计方案[J]. 微计算机信息,2009, 21(19):35-38.

[7] 朱建文,廖建新,朱晓明,等. 基于 ACE 并发编程模式的告警关联系统设计与实现[J]. 计算机系统应用,2007(11):26-29.

[8] 胡宁馨. 黄小平. ACE 编程框架快速通道分析与优化[J]. 小型微型计算机系统,2006,27(3):51-57.

[9] 蓝炳雄,张丽. 基于 ACE 的共享内存[J]. 计算机系统应用,2005(4):67-69.

[10] BEACH T H, RANA O F, REZGUI Y,*et al*. Cloud computing for the architecture, engineering & construction sector: requirements, prototype & experience[J]. Journal of Cloud Computing,2013,2(1):8.

[11] WU Jian-ming, TONG Wen, ZHU Pei-ying. Adaptive scheduling of voice traffic in a multi-carrier communication environment:US,2012/0195280 A1[P]. 2012-08-02.