基于设计模式的服务器控制框架

杨烜会　吉国力　李海良
(厦门大学自动化系,厦门 361005)
E-mail:xhyang@xmu.edu.cn

摘　要　针对传统服务器框架设计中可扩展性和健壮性的不足,该文引入设计模式,提出面向对象的服务器控制框架模型。该模型可为应用层协议系列的支持提供可扩展的接口,为服务器优化和监控提供灵活的挂接。

关键词　客户服务器结构　控制框架　设计模式　面向对象

文章编号 1002-8331-(2003)29-0181-03　文献标识码 A　中图分类号 TP311;TP368.5

Daemon Control Framework Based on Design Pattern Methodology

Yang Xuanhui　Ji Guoli　Li Hailiang
(Department of Automation,Xiamen University,Xiamen 361005)

Abstract: In this paper,a flexible and robust daemon control framework based on design pattern methodology is described.In addition,scalability and flexibility of such a framework are illustrated.

Keywords: client/server architecture,daemon control framework,design pattern,object-oriented

1　引言

随着 Internet 的飞速发展,网络服务日新月异,对分布式客户/服务器结构中服务器端的可扩展性、健壮性提出了挑战。近年来面向对象软件工程领域中设计模式(Design Pattern)研究的兴起,为解决这一问题提供了新的契机。

2　服务器控制框架面对的问题

由于业务模型经常会有变化和扩展,甚至业务系列的彻底更新,导致应用层协议和服务器在结构上和细节上的剧烈变更,还有随之而来的监控逻辑、优化策略的变更。所有这些变更带来的编程与维护工作是复杂而艰巨的,如近年来 QICQ 版本的升级令人目不暇接,其编程的困难是可想而知的,更不必说动态支持多个版本。

传统的服务器控制框架,一般用接收模块接收用户请求,分派给不同的服务进程/线程,服务进程/线程使用 switch 结构,根据<版本号、请求号>调用与之对应的服务例程,进行计算并把结果返回给客户。这样的实现方式有以下缺点:

(1)对于多版本协议,当使用一个大的 switch 结构时,由于<版本号、请求号>的组合会把控制流程分散到众多的分支中,同样是用户请求,使用了不同的处理,导致流程的一致性难以维护、代码难以阅读。即使采用两级 switch 嵌套,仍然难以统一处理。

(2)由于支持的协议种类繁多,多种服务往往粒度不均匀、内容的差异也很大,一些局部的监控、优化(如计费、日志、并发等)会分散,无法统一起来,在整体层次上进行维护,导致代码冗余。

(3)全局资源的分配、异常处理机制也难于控制。而且由于同样的功能实体要分别编码,会出现命名空间问题。

如何既能对各种服务进行统一处理,又能保证具体服务的特性;既能支持灵活的控制流程,又保证程序扩展的方便和系统的健壮?这成为服务器控制框架设计中突出的问题。

3　基于设计模式的解决方案

3.1　设计模式

设计模式是当前面向对象软件工程领域的一大热点,它记录、提炼了一些反复出现的共性问题及其经过多次验证的成功解决方案。它具有如下特点:

(1)从特定的〈问题,解〉对中加以抽象,提取带普遍性的因素,便于交流和使用。重用已成功实现了的系统级概念模型部件,通用性、移植性好。

(2)强调系统中对象间的责任与协作。针对接口而不是具体实现来编程,这样系统结构的模块性、可扩展性就提高了。

(3)支持系统的变化,避免重新设计。设计模式把变化封装起来,允许系统结构的某个方面的变化独立于其他方面,所以系统对于某一类特殊变化就更加健壮。

客户/服务器模型中服务器的本质是执行循环:接收客户的服务请求,执行计算,把结果返回给客户。对于上节分析的服务器控制框架设计中存在的不足,通过面向对象的分析,围绕用户请求和请求处理器这两个关键类,使用抽象工厂、模板模式,并结合其他的设计模式(如代理等)可以设计出一个实用的服务器控制框架。

3.2　服务器控制框架的对象分析

设想要实现一个如下的并发服务器:支持某应用层协议 n 个版本 $V_1,V_2,...V_n$ 中的某一个,版本 V_i 支持 m_i 个服务。且对于服务 S_{ij}(为协议版本号,j 为服务号),可能的执行策略要立刻执行、在单独的进程/线程中执行,在预分配的进程/线程中

基金项目:福建省重点科技项目(编号:2001H020)

作者简介:杨烜会,工程师,在职硕士研究生,主要研究方向:面向对象技术。吉国力,教授,主要研究方向:系统工程。李海良,硕士研究生,主要研究方向:网络通讯。

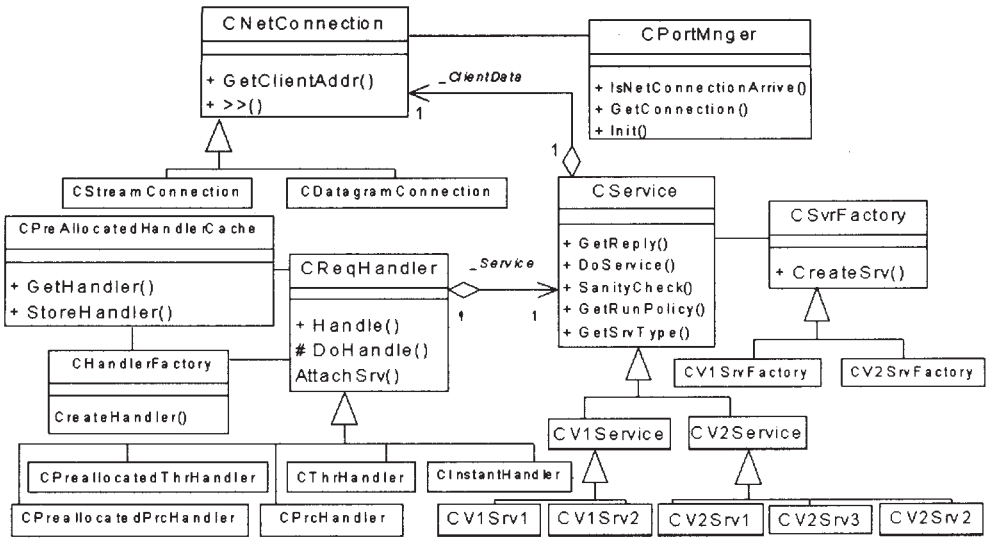

图 1　服务器控制框架的类图

执行等几种。可能的网络接入可以是流式连接、也可以是数据报。

通过分析与抽象，获得如图 1 所示的类图　使用 UML 和 C++伪码表示)。

图 1 中,所有的父类均为抽象类,负责提供通用接口;为简单起见只画了两个版本系列的部分服务。

端口管理器类（CPortMnger）:用来管理服务器监听的网络端口。成员函数 Init 【用来初始化网络端口。用户请求到达时，成员函数 IsNetConnectionArrive 【的值为真,此处 CPortMnger 设计成阻塞式,也可改为事件驱动式。GetConnection 【返回到达的用户连接。

用户连接类（CNetConnection）:网络中用户请求的抽象,子类 CStreamConnection,CDatagramConnection 分别对应流式和数据报连接。成员函数 GetClientAddr 【返回连接的地址信息。而流式操作符>>将以流式提供连接中的用户数据。

请求受理器类（CReqHandler）:并发策略和服务控制流程的抽象，全局性监控模块的挂接点。通过保护型的变量 _Service 可以调用用户请求类 CService 的操作。公共成员函数 Handle 【调用 DoHandle 【对外提供服务。CReqHandler 的子类将重载 Handle 【，以便在适当的进程/线程中执行 DoHandle 【,实现并行策略。AttchSrv 【则通过 _Service 把用户请求和受理器关联起来。

用户请求类（CService）:用户请求服务的抽象,实质性的用户服务都在这里处理。GetRunPolicy 【返回执行策略,有启动进程/线程式,预分配进程/线程式,也有立即执行式,以对应不同子类 服务)的粒度。SanityCheck 【做验证用,如检校和、解密、合法性检查等。DoService 【是服务计算的场所,GetSrvType 【通过 _ClientData 中<版本号、请求号>数据所唯一确定的服务内容主要在这里执行。GetReply 【返回计算后生成的应答。子类 CV1Service 抽象了版本 V_1 提供的系列服务,如 CV1Srv1 等。其它子类的命名方式类似。

预分配受理器缓冲池类 （CPreallocatedHandlerCache）:用来初始化、存储预先分配的进程/线程。GetHandler 【和 StoreHandler 【分别负责预分配进程/线程的进出。预分配受理器缓冲池类和抽象工厂类 CHandlerFactory 配合使用。

两个抽象工厂类 CHandlerFactory 和 CSrvFactory 根据参数定做 CService 和 CReqHandler 的子类。

通过以上的类分析,用抽象的接口统一了系统中对象间的协作,为框架的可扩展性奠定了基础。而类的模块性和封装性则为系统的健壮性提供了保证。

3.3　抽象工厂模式（Abstract Factory）

抽象工厂模式是一个创建型的模式,它提供创建系列对象的接口,而无需指定它们具体的类。当一个系统要独立于它的产品来创建、组合和表示时,或要强调一系列相关产品对象的设计以便进行联合使用时,特别是一个系统要由多个产品系列中的一个来配置时,抽象工厂模式提供可扩展的灵活接口。它的结构如图 2 所示。

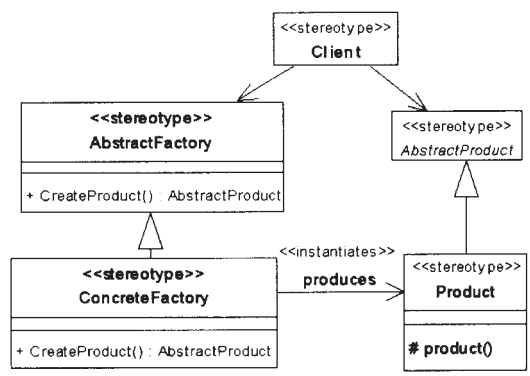

图 2　抽象工厂模式结构图

服务器支持的多版本用户请求,是多个系列,而支持多系列正是抽象工厂的特点。在创建对象时指定类名将受到特定实现的约束，会使框架失去变化的机会。通过用户请求类 CService,统一了产品,又通过 CV1Service 和 CV2Service 统一了版本,同时增加了层次,为同一版本中服务系列的公共模块提供了安放点,并提供了切换版本的机制。请求处理器工厂 CHandlerFactory 则是一个退化的抽象工厂,同样解耦了并发策略的创建和使用。抽象工厂同时也是统一分配全局资源的好地方。示例如下：

CReqHandler*CHandlerFactory：:CreateHandler（CService & aSrv）

```
{
  switch (Srv.GetRunPolicy ())
  {
  case INSTANT_EXCUTING:                    //立即执行
    return new CInstantHandler (Srv); break;      //创建,帮定
  case FORK_PROCESS:                        //在单独进程中执行
    return new CPrcHandler (Srv); break;
  case PRE_ALLOCATED_PROCEES: //在预分配地进程中执行
    CReqHandler*tmp=PrcCache.GetHandler ();
                //PrcCache 是 CPreallocatedHandlerCache 的实例
    tmp->AttachSrv (Srv);                   //帮定服务
    return tmp; break;
  ...其他的执行策略对应的分支...
  }
}
```

3.4 模板模式 (Template Method)

模板模式是面向对象技术中动态绑定机制的应用。它使用一组接口来定义算法框架,在算法框架执行时刻利用动态绑定机制自动调用适当的子类代码完成计算。模板方法实现了算法中不变的部分,将可变的部分留给子类扩充,但这些扩充又被限制在特定点,从而避免了 switch 结构的缺点。模板方法还可以把子类公共模块集中到父类以避免重复。

服务器控制框架的工作流程为:

(1)初始化 (参数,网络端口等)

(2)进入循环

(3)接受网络连接

(4)派发给请求受理器 (进程/线程)

(5)返回 (2)

请求受理器的工作流程为:

(1)语法解析,验证

(2)语义解析,服务

(3)生成响应

(4)发出响应

都是典型的流水结构, 但具体的服务项目又差别很大,通过使用模板方法,就可以统一处理。而且,控制模块可以插在适当的地方,利用类层次达到公共实现,避免了外部函数的滥用。模板方法很优雅地提供了一个清晰而又易扩展的框架。

模板方法的伪码如下:

3.4.1 服务器控制框架算法

```
CHandlerFactory HandlerFacotry;
CSrvFactory*CurrFactory=new CV1Service;
  //此处配置为支持 V1 版本的服务
CNetConnection*CurrConnection;
CService*CurrSrv;
CReqHandler*CurrHandler;
CPortMnger Port;
Port.Init ();
while (true)
{
  //模板方法,可在这个 if 语句体中插入各种控制模块
  //利用抽象工厂进行创建和绑定,使用了指针,编码中要注意对象的销毁责任
  if (Port.IsNetConnectionArrive ())
  {
    CurrConnection=Port.getNetConnection ();   //取得用户连接
    CurrSrv=CurrFacotry->CreateSrv (CurrConnection);
      //生成特定地服务请求,并绑定连接
    if (CurrSrv==NULL)throw Exception;//不支持的非法服务请求
    CurrHandler=HandlerFactory.CreateHandlder (CurrSrv);
      //生成受理器,并绑定服务
    CurrHandeler->Handle ();                   /服务过程
  }
}
```

3.4.2 请求受理器处理算法

```
//模板方法,可在这个函数体中插入其它控制模块,日志模块 log ()就是一个例子
void CReqHandler::DoHandle ()
{
  if (!_Service->SanityCheck ())throw Exception;
    //检校和、解密、合法性等检查
  _Service ->DoService ();//语法、语义分析,进行计算,生成响应
  Out (_Service->GetReply ());                 //发出响应
  Log ();                                      //日志
}
```

在按各种 CReqHandler 子类的 Handle ()中执行 DoHandle ()也是模板方法,另外 CService::DoSercive ()中也可按服务的结构使用模板方法。

4 服务器控制框架的可扩展性与可维护性

如果要扩展协议,只需从 CService 派生子类,并重载相应的方法,然后通过工厂发行即可。如果要更新协议系列,把服务器控制框架中的 CurrFacorty 变量初始化成新的版本工厂即可。通过把协议的扩展、更新工作集中在 CServicce 和 CSrvFactory 的子类中,开发人员得以聚焦在服务相关的高层设计,而无须担心流程的维护。

用户如果要改监控流程和优化策略,只需在模板方法中改变即可。同时,类层次结构中上层的变化会统一地自动扩展到下层,从而达到了集中维护的目的。

设计模式还可用在客户服务器结构的诸多方面, 如订阅-发行 (Subscribe-Publish)模式用于动态配置,认证模式 (Authority)用于多模式身份验证,外观 (Facade)用于本地化等。该文讨论的框架对客户端编程还有指导意义。

文章讨论的技术已在某公司的在线服务器上应用,取得了良好的效果。特别是在业务设计的初期,由于能方便地支持服务定义的变更,大大节省了编程时间;在后期的优化工作中,也为比较不同的优化效果提供了方便;该服务器得益于面向对象技术的健壮性,也在后续的维护工作中得到了体现。

5 小结

通过以上讨论,文章示范了一个支持多种执行策略、多个系列协议的服务器控制框架。主要采用的分析方法是通过抽象,提供系统实体间的统一接口;利用类层次结构,提供安全的、多样的挂接点;利用面向对象语言多态、动态绑定技术实现特色服务。

使用设计模式,把策略的分配与使用解耦,服务的实现与执行解耦,从而构成一个可扩展的、健壮的服务器框架,使服务器的可重用性、可维护性增强,生命周期延长。

(下转 187 页)

ECDSA 可能的攻击有如下几种：

(1)对椭圆曲线离散对数问题的攻击；

(2)对所用的 Hash 函数的攻击；

(3)其它攻击。

5.1 对椭圆曲线离散对数问题的攻击

椭圆曲线离散对数问题(ECDLP)是指：通过给定的域参数(q,FR,a,b,G,n,h)和公钥 Q,对手能计算出私钥 d,从而伪造签名,数学上定义指：给定有限域 Fq 上的椭圆曲线 E,点 $P\in E(Fq)$,其阶为 n,另有一点 $Q=lP$,$0\leqslant l\leqslant n-1$,推出 l。

5.1.1 已知主要的攻击

(1)天真但彻底的搜索：简单计算 P 的倍乘：P,$2P$,……,直至找到 Q,最坏需要 n 步。

(2)Pohlig-Hellman 算法：该算法将找到 l 的问题化简为找到 l 模 n 的每个素因子的问题；而 l 能用中国剩余定理推出。

(3)小步,大步算法：该算法是彻底搜索的时间——存储空间转换的方法,它需要大约 $\sqrt{n}$ 个点的存储空间,运行时间最坏约需 $\sqrt{n}$ 步。

(4)Pollard's rho 算法：该算法是小步,大步算法的随机版本,它的运行时间与小步、大步算法差不多($\sqrt{\pi n/2}$ 步),但只需很小的存储空间。

(5)并行 Pollard's rho 算法,当 Pollard's rho 算法在 r 个处理器上并行处理时,运行时间约为($\sqrt{\pi n}$)/(2r)步。

(6)乘积算法：如果 ECDLP 的单独一个例子被并行 Pollard's rho 算法解决了,那么在解决这个例子时所做的工作能被用来加速其它 ECDLP 例子的解决。例如：解决第一个例子花了时间 t,第 2 个例子就只需花($\sqrt{2}-1$)t=0.41t,第三个例子在前 2 个问题的解决基础上,只需($\sqrt{3}-\sqrt{2}$)t=0.32t 等等。

(7)超奇异椭圆曲线：在温和的假设下,ECDLP 能被简化为普通的乘法群上的 DLP,该群是某个扩域 Fq^k 的乘法群($k\geqslant 1$),这里数域筛法将被采用,简化算法只有在 k 小的时候才有实际意义。为防止该简化算法,必须检查 n 对所有 k 不整除 q^k-1,实际上 $n>2^{160}$,$1\leqslant k\leqslant 20$ 就足够了。

(8)素域不规则曲线：曲线性质为：$\#E(F_p)=p$,对此类曲线,能有效地解决 ECDLP,只需使 $\#E(F_p)\neq p$,就能避免此攻击。

对 ECDLP 已知的具有一般意义的最好算法是并行 Pollard's rho 算法,其运行时间为($\sqrt{\pi n}$)/(2r)步(k 是基点的素数阶,r 是处理器数)。当 n 为一个大素数时(如 160bits 的数)其算法是不可行的,而在 ECDSA 要求 n 至少是 160bits,所以对 ECDLP 问题的攻击在 ECC 中是不可行的。

5.1.2 硬件攻击

若 $n=2^{120}$,花费 1000 万美元建造的有 330000 个处理器的计算机为解决某个 ECDLP 需 32 天,但实际应用中 $n>2^{160}$,这样的硬件攻击用今天的技术是不可行的。

5.2 对 Hash 函数的攻击

(1)对手先选一个任意整数 l,r 为 $Q+l$Gmod n 的 x 座标,置 $s=r$,算 $e=rl$ mod n,若找到一个消息 m 使 e=SHA-1(m),那么(r,s)就是 m 的有效签名,对手就能伪造合法签名。

(2)若用户 A 生成两个消息 m,m',SHA-1(m)=SHA-1(m'),他对 m 签名,随后就可称对 m' 已签了名。

但是 ECDSA 中 Hash 函数采用 SHA-1,SHA-1 被认为是理想的非常安全的 Hash 函数,类似 1 和 2 的攻击几乎是不可能发生的。

5.3 其它攻击

在 ECDSA 签名生成过程中的"每个消息秘密"k 同私钥 d 一样也对安全性有很高要求,这是因为对手若知道用来生成消息 m 签名(r,s)的 k,他就能通过 $d=r^{-1}(ks-e)$mod n(e=SHA-1(m))来恢复私钥 d。因此,k 必须安全地产生、存储、破坏。用来签两个或更多消息的 k 必须独立生成,最好,不同的签名消息就生成不同的 k;否则,私钥 d 就能被恢复。这种情况也可避免。

6 结束语

ECDSA 现在成为 ANSI,IEEE,NIST,和 ISO 的标准.其安全基础是 ECDLP,从上面的论述可以看出椭圆曲线密码体制几乎是完美的密码体制,它无懈可击。这也是 ECC 成为目前密码学界热点的原因之一。在实现 ECDSA 时需要考虑很多因素正如前面的论述需要注意的问题是域的选择,域中元素的表示,域中元素的运算,椭圆曲线的选择,椭圆曲线上点的运算以及数据类型的转换等。其中椭圆曲线上点运算效率关系到整个 ECDSA 实现的效率,也是 ECC 实现的关键所在。

(收稿日期：2002 年 11 月)

参考文献

1.陈彦学.信息安全理论与实务[M].北京：中国铁道出版社,2001

2.王育民,刘伟建.通信网的安全—理论与技术[M].西安：西安电子科技大学,1999

3.Bruce Schneier.Applied Cryptography Second Edition：protocols,algorithms,and source code in C[M].John Wiley & Son,Inc,1996

4.Ceiticom Corporation Whitepaper.Canada：Ceiticom Corporation,1997

5.Menezes A J.Ellipse curve public key cryptosystem[M].USA：Kluwer Academic Publishers,1993

6.William Stallings.Cryptography and Network Security：Principles and Practice[M].Second Edition,Prentice-Hall,Inc,1999

7.Working Draft American National Standard X9.62-1998 Public key Cryptography For The Financial Services Industry：The Elliptic Curve Digital Signature Algorithm(ECDSA)[S]

(上接 183 页)

(收稿日期：2002 年 11 月)

参考文献

1.Erich Gamma et al.设计模式：可复用面向对象软件的基础[M].北京：机械工业出版社,2000

2.Douglas Comer,David Stevens.用 TCP/IP 进行网际互联[M].北京：电子工业出版社,2001

3.Douglas Schmidt,Tatsuya Suda.The Service Configurator Framework[C].In：Proceeding of 2nd International Workshop on Configurable Distributed Systems,Pittsburgh：IEEE Computer Socitey,1994：190~201

4.Bruce Eckel.C++编程思想[M].北京：机械工业出版社,2000