Software Architecture

- principle and practice

第三讲　分布式体系结构分析

- 一、分布式处理(Distributed process)结构风格概述
  - 1.分布式处理结构风格的系统组织
    - 组件：系统由多个可以独立运行机制的部件组成，每个部件称为组件；同时系统由多台机器运行这些组件，这些机器共同协作完成系统的所有功能。
    - 透明性：系统中的每个组件不需要关心其他组件的运行位置、实现方式及数量。
    - 可靠性：当系统中的一个或少数几个组件出现故障(关闭或崩溃)时，系统仍然可以工作，所出现的故障造成的影响多只是系统运行速度。
  - 2.分布式处理结构风格的优缺点
    -
  - 3.构造分布式处理结构风格的主要技术
    - 层次结构：组件的一种组织方式。
    - 技术基础：进程通信
    - RPC：进程通信的发展
    - 组件技术：RPC的发展，如CORBA、J2EE等
    - 云计算技术：分布式处理技术的最新发展。
- 二、分布式系统（Distributed System)
  - 1.分布式系统是计算机发展的必然产物
    - 不断推出功能强大的微处理器，价格便宜
      - 自1945年计算机的出现至1985年,计算机系统经历了一场代替革命.开始,计算机系统庞大价格昂贵，各单位、组织的计算机拥有量很少，并且也仅仅是彼此独立的。到了80年代中期，计算机领域的两项新技术改变了这种状况。
    - 强有力的微处理器的发展
      - 开始是8位机，很快普及到16位．32位，甚至64位的微处理器，这些处理器的价格便宜。开始，一台机器1千万美元，每秒1条指令，现在1台机器1千美元，每秒1千万条指令。
    - 高带速网络的发展
      - 使得局域网可将同一单位几百台的计算机连接起来，它们中少量数据可在1μs内在机间传输，而广域网则能够将全球范围内成千上万的机器连成一体协调工作。
      - 网络的发展造成异构不可避免。一个原因是网络技术随着是不断地改进，不同时间最好的网络技术可能在同一网络中共存；另一个原因是网络大小不是固定不变的，任何一种计算机、操作系统、网络平台的组合都是为了能在一个网络内使得某一部分的性能达到最好；还有一个原因就是在一个网络内的多样化使得它具有可选余地，因为在某一机器类型，操作系统或应用程序所出现的问题可能在其他操作系统和应用程序上不成其为问题。
    - 在计算机网络中工作站的CPU浪费严重
      - 在计算机网络中，工作站之间的CPU无法共享、负载不平衡，在一个网络中，计算中心的工作站的CPU往往承担大量的计算，而许多办公室中的工作站其CPU又多数是空闲。
    - 单个微处理器的发展有极限
      - 处理器芯片集成度高，比特位的传输又产生巨大的热量，虽然在理论上传输速度可以达到光速，而实际上，在远远未达到这处速度之前，可能会使处理器烧毁。
  - 集中式系统和分布式系统
    - 只有一个CPU以及相应的存储器．外围设备和几个终端组成的计算机系统。
    - 在硬件方面，很容易将若干台机器通过物理线路连接起来，并且也可以实现对一些资源的共享，如数据、外围设备。这些都只是计算机网络的概念，但是要能实现CPU的共享、分散各地的CPU能协调地完成某项任务，还需要软件的支持，这就是分布式操作系统。
    - 一个分布式系统就是一个由若干独立的计算机组成的一个系统，该系统对其用户来说，操作起来就就像一台计算机一样。
  - 一个常见的分布式系统的例子
    - 　某学校的网络中，每台工作站都可以独立工作，我们将网络中的各个工作站的CPU组成一个CPU池，每个CPU并不指定分配某个工作站的用户而是可以根据需要动态地将某个CPU分配给某个用户。在这个网络中，也只有一个文件系统，所有机器都可以用同一方式和同样的路径来访问系统中的所有文件。比如一个用户在自己的工作终端上输入一个命令，系统将找到一个最佳的位置为其运行命令，可能是在自己的工作站上，也可能是在某一台机器上暂时没有工作的CPU上执行，或者是系统中某个尚未分配出去的CPU上执行。如果这个系统看上去像一个整体，而又像单处理器系统那样工作，我们就可称之为分布式系统。
  - 分布式系统的优点
    - 经济上 - - 分布式系统提供更高的性能 / 价格比
      - 通常价格与处理器能力的平方成正比。
      - 假定一个分布式系统由n个独立的计算机组成,其中计算机i有一个CPU其计算能力为P_i,那么这个分布式系统的计算能力P：　$P=\sum P_i, i=1, n.$
    - 运行速度

        - 分布式系统的计算能力是系统中各个独立计算机计算能力的总和。这使得分布式系统不仅有更高的性能/价格比，并且其计算能力是集中式系统不可达到的。比如：
        - 由10，000个CPU组成的分布式系统，每个CPU的执行速度为50MIPS，该系统总的执行能力为50MIPS*10,000。而对单个处理器而言，要达到这个程度，一条指令必须在0.002ηs内完成，这在理论上都是不可能的（因为将要求电子在芯片中的运行速度要超过光速).
    - 应用领域的需要
        - 有些应用领域需要是分布式系统：比如商业分布式系统。超市连锁店系统中，对每个分店，可独立营销，时刻掌握本店的库存情况，而高层管理人员也需要时刻了解各个分店的情况。最好的作法就是建立一个分布式系统将各个分店连结起来形成一个整体。
        - 另外，计算机支持协调工作：有一群决策成员彼此分散各地，而共同完成一个主题。
    - 可靠性
        - 分布式系统比集中式系统具有更高的可靠性，由于分布式系统有多个机器，单个芯片的故障，不会影响整个系统的工作，而至多只是降低系统的计算能力。
    - 扩充性
        - 计算能力可以逐步增加。
    - 适应性
        - 在分布式系统中，工作量以有效的方式分布于可使用的机器上
- 分布式系统的不足
    - 软件 - 适用于分布式系统的软件很少
        - 到目前为止，还没有一个成熟的实用的分布式操作系统、程序设计语言、以及适合的应用软件。
        - 可喜的是，这方面的研究工作不断进展，软件问题将不再存在。OMG组织的CORBA规范就是一个很好的例子，可用于开发分布式应用系统。
    - 通讯网络 - 网络饱和以及其他问题
        - 分布式系统的硬件基础是计算机网络，网络使分布式系统的优点受到影响。
        - 当现有的网络饱和时，必须重新规划新的网络或增加第二个网络，从而带来布线等方面的不便和开销
        - 网络通信信息可能会丢失，为此需要特殊的软件来恢复，但它将使网络负荷增加。
    - 安全性 - 保密的数据也变得容易存取
        - 数据容易有利有弊：如果人们很容易地访问系统上的所有数据，他们也可很方便地查看与他们无关的数据。为此人们不喜欢自己的机器上网。
    - 尽管有这些潜在的问题，人们还是觉得利大于弊，并预计分布式系统将越来重要，事实上，不久的将来，有更多的组织、单位要求连接自己的机器构成一个大的分布式系统，提供更好、更经济、更方便的服务。一个独立的计算机系统在中、大型企业、组织将不再存在。

三、客户/服务器（Client/Server)结构

- 1.C/S结构
    - 在一个信息处理系统中，通常由若干台计算机组成。其中用于提供数据和服务的机器，称为服务器(Server)，向服务器提出请求数据和服务的计算机称为客户(Client),这样的系统工作模型称为客户/服务器(Client/Server)模型，简称C/S模型。
    - 在广义上说，客户、服务器也可以是进程。因此，C/S模型可以在单处理器的计算机系统中实现。
    - 在多计算机或多任务的单处理器系统中，C/S模型有多种组织方式：单客户单服务器、单客户多服务器、多客户单服务器、多客户多服务器等。
- 2.C/S模型的软件结构的特点
    - 简化软件产品的设计
        - 这种结构把软件分成两个部分，客户部分可专门解决应用问题、界面设计、人机交互等方面，服务器则侧重于服务操作的实现、数据的组织，以及系统性能等。
    - 提高软件的可靠性
        - 在这种结构风格的系统中，不仅客户与服务器是独立的，服务器与服务器之间也是独立的，一个服务由一个服务器完成，它不影响其他服务器的工作。
    - 适合分布式计算环境
        - C/S模型的软件中，Client与服务器之间的通信通常是以消息传递方式实现，对客户来说，它只关心服务请求的结果能正确地获得，而至于服务的处理是在本地还是在远程并不重要。
- 3.C/S模型的工作过程
    - 客户方处于主动，向服务器提出服务请求，而服务器处于被动，服务器进程启动后，就处于等待接收客户的请求，当请求到达时被唤醒；
    - 客户方以通常的方式发出服务请求，由客户机上的通信软件把它装配成请求包，再经过传输协议软件把请求包发送给服务器方；
    - 服务器中的传输协议软件接收到请求包后，对该包进行检查，若无错误便将它提交给服务器方的通信软件进行处理；
    - 服务器通信软件根据请求包中的请求，完成相应的处理或服务，并将服务结果装配成一个响应包，由传输协议软件将其发送给客户；
    - 由客户的传输软件协议把收到的响应包转交给客户的通信软件，由通信软件做适当的处理后提交给客户。

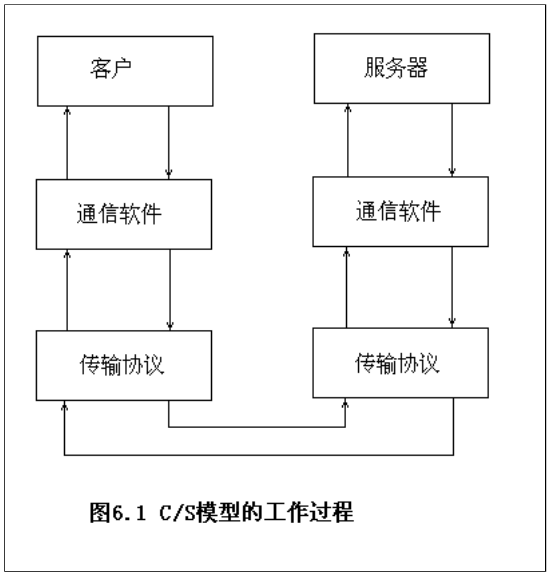

图6.1 C/S模型的工作过程

- 4.数据共享的方式
  - 数据移动共享
  - 计算移动共享
- 5.C/S的连接
  - 使用消息传递的方式
    - 这就是上讲中的基于消息传递的通信方式，连接可以是同步的也可以是异步的。
  - 使用过程的方式
    - 每个部件之间通过良好的接口定义与其他部件通信，这种方式的技术被称为过程过程调用(Remote procedure call,即RPC)，通常RPC是采用同步的连接。
  - 使用对象引用方式
    - 这是目前普遍采用一个跨平台分布式软件的设计技术，我们将在后续的讲授中计划安排两个的时间介绍。
- 6.C/S的服务器设计、实现关键
  - 服务器的调度任务和调度方式
    - 服务器的调度任务是解决多客户和多请求同时执行的问题。作为独立工作的服务器，无法确定什么时间会发生多少的客户请求，所以它必须随时准备，处理来自不同的客户的请求，如果服务器已经无能为力处理新的请求，它必须以一定的策略建立等待队列或通知客户不能立即处理请求。
    - 现在，基于应用层的服务器都是运行于多任务的操作系统上，甚至是支持线程的操作系统，使用服务器对客户的请求可以采用进程调度方式(process scheduling) 和线程调度(thread scheduling)方式。
    - 当有多个请求到达时，服务器必须按一定的方式来为每个请求服务，这种方式称为服务器的调度方式。
    - 基于进程的调度方式：主服务器进程每收到一个消息，就创建一个子进程并回到循
    - 环的顶部，由子进程(从服务器(salve))负责消息的处理。
  - 线程技术
  - 服务器缓冲技术

四、C/S案例

- 1.一个C/S模型的文件服务器的设计
  - 消息格式(message structure)设计
发送进程标识(source)
接收进程标识(dest)
请求服务类型(opcode)
传送的字节数(count)
文件读/写指针(offset)
备用1(reserved1)
备用2(reserved2)
服务结果(result)
访问的文件(name)
数据缓冲区(data)
  - 服务器的工作流程

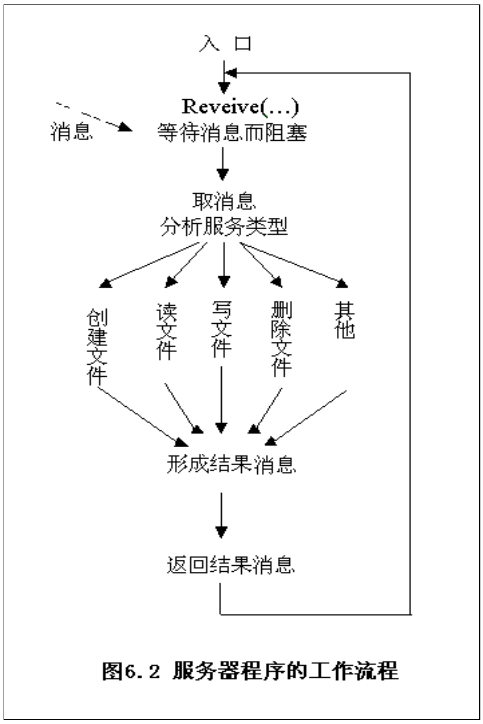

图6.2 服务器程序的工作流程

- 客户程序工作流程
  - 客户程序将服务器中的一个文件的信息写到另一个空文件(目标文件)中。

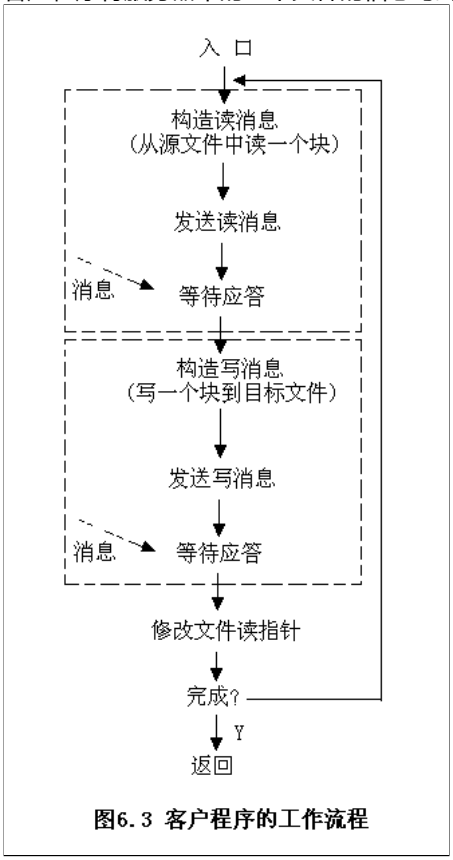

图6.3 客户程序的工作流程

- 2.一个C/S模型的文件服务器的C语言实现
  - **文件服务器**
    - 头文件
    - header.h

```
#define MAX_PATH 255

#define BUF_ZISE 1024

#define FILE_SERVER 243

#define CREATE 1

#define READ 2
```

```
#define WRITE 3
#define DELETE 4
#define OK  0
#define E_BAD_OPCODE -1
#define E_BAD_PARAM -2
#define E_IO -3
struct message{
long source;
long dest;
long opcode;
long count;
long offset;
long result;
char name[MAX_PATH];
char data[BUF_SIZE];
};
```

- 服务器程序
-

```
#include <header.h>
main ( )
{
  struct message m1,m2;
  int         r;
   while (1) {
      receive (FILE_SERVER, &m1);
      switch (m1.opcode)
      {
       case CREATE:
         r=do_create(&m1,&m2);
         break;
       case READ:
         r=do_read(&m1,&m2);
           break;
       case WRITE:
         r=do_write(&m1,&m2);
           break;
       case DELETE:
         r=do_delete(&m1,&m2);
           break;
         default:
```

```
    }
   m2.result=r;
   send (m1.source, &m2);
 }
}
```

- 客户程序

```
int copy(char *src,char *dst)
   {
    struct message m1;
    long position;
    long client = 110;
    initialize();
    position=0;
    do  {
     m1.opcode=READ;
    m1.offset=position;
    m1.source=client;
    m1.count=BUF_SIZE;
    strcpy(m1.name,src);
    send(FILE_SRVER,&m1);
   receice(client,&m1);
   if(m1.result<=0)break;
   m1.opcode=WRITE;
   m1.offset=position;
   m1.count=m1.result;
   m1.source=client;
  strcpy(m1.name,dst);
  send(FILE_SERVER,&m1);
  receive(client,&m1);
  position+=m1.result;
 } while(m1.result>0);
 return(m1.result>=0OK:m1.result);
}
```

==========