计算机科学 1999Vol. 26№. 6

网络环境中流式数据传输应用的开发问题

On Issues in Development of Streaming Transportation within Network Environment

严隽永
(中国纺织大学计算机系　上海 200051)

Abstract The problems of transportation using streaming data structures in network application software are discussed. In particular, the relationship between high-speed network performance and development techniques are studied. The efficiency of development, on one hand, and the efficiency of runing, on the other hand, are considered. A compromise upon them is possible, hereby based on object-oriented methodology a middle-level approach is conducted.

Keywords Computer network, Software development environment and tools, Object-oriented methodology

1 引言

网络计算与多媒体技术的结合开辟了信息科学的新的广泛的应用领域，比如实时图象传输、影视会议、点播式影视等等应用，诱发起人们广泛的兴趣。但是，这类应用对带宽和响应时间的要求比较敏感。归纳起来，这些应用大多属于流式(streaming)数据传输范畴。流式应用的数据类型较为简单，但吞吐率较高而对迟延性限制苛刻。因此，用于数据传输的开销不容许太高。在通常的数据类型较为复杂的交互式或单向传输中，开销的影响并不敏感。这些开销主要是由于表示层的语法转换、数据复制、存储管理、多路传输的分解(解复用)及其分派，以及传输中的同步控制与重发送等机制中各种复杂的、未优化的和非自适应的操作过程所引起的。

在网络环境中，软件开发方法至少分为两个层次：高层，如分布对象计算(DOC)框架方法，用远程过程调用(RPC)或 CORBA 实现；低层，如插座(socket)方法、传输层接口(TLI)方法。一般说来，高层方法的开放性好，开发效率高，但所生成软件的运行效率较低；低层方法所生成软件的运行效率较高，但开发复杂，类型安全性较差，易出运行错误。因此，对于流式应用，就目前技术水平而言，这两类方法都不甚适宜。

本文介绍和讨论一种折衷的解决方法，称为中层方法。这种方法用 C^{++} 编写的一整套包裹(wrapper)将低层网络编程接口(如 socket)封装起来，使程序正确性、易用性、可移植性和可重用性得到改善，如 ACE 中所用的方法。

2 高速网络性能敏感性

文[1]提供了一些基准测试结果。就数据吞吐率而言，在高速的 155Mbps ATM 网络上，低层和中层方法明显好于高层方法，低、中层方法的带宽利用率最高为 40%左右，约 62Mbps，而高层方法为 20%左右，约 30～40Mbps。在 10Mbps 以太网上，则各层次方法的差别不大，带宽利用率可达 85%左右(应不考虑冲突率)。在 ATM 网络上出现的带宽利用率较低，是由于操作系统与协议处理所需的开销引起的，例如其中数据复制、进程上下文切换及同步等都需要开销。而高层方法(如用 CORBA 实现)则加剧了这种情况，因为其中会发生多次数据复制、分片与组合、编组与解组，这些都会耗去很多开销。这个问题称为吞吐率保留问题，实质上就是只有一部分带宽被赋予应用任务本身而其余较多部分则被耗于各种开销所致。

试验表明[1]，低层和中层方法(C sockets 和 ACE C^{++} Wrappers)的性能差别不大。在 ATM 上，socket 队列长度对性能影响较大，较长队列增大了 TCP 窗口宽度，一次可传送多个 TCP 分段，因此提高了吞吐率。在 10Mbps 以太网上，队列长度的影响不大，因为限制性能的主要因素是网络速度本身较

低。

在高速网络上，CORBA 方法的开销才明显地暴露出来。同时，设计和实现中一些具体改进在高速网络上会显出较好的效果。所以建议软件开发者应当尽力改进他们对于流式应用的 ORB 实现，尤其是在高速网络上，如 ATM 和 FDDI。其中，优化的主要着手点是数据复制和监测、表示层转换、存储管理、接收端的解复用和分派，尤其要减少发送和接收端大缓冲量数据的复制。在上述优化尚未实现的情况下，对于流式应用应当着重考虑如下几点：

· 针对具体应用要求，必须进行仔细分析，根据不同的应用，选用不同的开发方法。可将高层方法和中、低层方法适当地结合起来。

· 对于所赖以支撑的通信网络应获得其性能测试结果，其中可采用一些标准软件包进行测试。

· CORBA 实现中应提供对下层支撑协议层的直接"吊钩"，即对它们的某些直接控制，如可方便地调整 socket 队列长度的方法，以便能调整数据缓冲，以适应网络性能。

· IDL（接口定义语言）所定义的接口用 sequence 类型传送数据比用 string 类型更好，这样可避免不必要的数据存取。

3 低层网络编程接口的问题

低层网络编程接口（如 socket，TLI，STREAM 管道，命名管道）虽然所生成的软件运行效率较高，但除了编程效率低外还存在明显的不足。

以 socket 为例，从应用的角度观察，它是虚拟通信线路上本地的端点。只要应用的"插头"插入这个插座就能与远方的对象进行通信。应用程序通过所谓"柄"或"描述子"（handle，descriptor）对 socket 进行存取。柄表示为无符号的整数，指向由操作系统所管理的某表的表目，而操作系统内部的数据结构被屏蔽起来，应用对此是不可见的。在 UNIX 和 Windows NT 的实现中，socket 柄与其他对象（如文件、命名管道、终端设备）的柄共享同一个名字空间。标准的 socket 接口由 C 函数提供，它们实施各种通信服务。这类低层网络编程接口具有一些明显的不足之处。

首先，低层方法是弱类型的，出错可能性较高。柄表示为整数，而整数是一种初等类型。在实现中，任何一个整数都可能当作一个柄来处理，而编译器的类型检查无法区分是柄还是别的什么意义的整数，也无法区分是什么柄。在运行时，这种弱类型就可能引起出错。

C 语言低层网络编程必须精心设计，疏忽一些编程语义细节就可能引起运行时错误，而这类错误编译器也无法检查出来。C 语言并不支持数据抽象和面向对象编程，难以定义类型上安全的、可重用和可扩展的接口。为提高软件可靠性和健壮性，势必要增加运行时出错异常处理能力。

其次，低层方法的接口较为复杂。它们以单一接口支持多种协议，如 TCP/IP，IPX/SPX，ISO OSI，UNIX 通信域等。接口具有许多函数，包含各种连接方式、通信优化和协议等选项。把多种功能性包含在单一的通用接口中，这使得接口复杂且难以把握。

从通信机制考察，单一接口实际上分解为三个不同的侧面：一是通信服务的类型，如面向连接的消息，数据报，或流；二是连接角色，如主动连接或被动连接，客户往往是主动的，服务器往往是被动的；三是通信域，如本地 IPC 或本地及远地混合的 IPC。而 socket 这类接口所表示的这三个侧面是不清晰的。其函数也没有统一的命名规则，难以判别其作用域。

4 面向对象方法的应用

利用 C++ 类（Class）的机制可以克服 socket 这类低层方法之不足，以改进通信软件的正确性、易学易用性、可重用性和可移植性，而不失其性能。这是一种介于中间层次的面向对象的方法，既发挥面向对象技术的长处，又保留以柄为基础的、如 socket 这类低层方法的效率。这种技术对于流式应用比较适宜。软件开发者们正在研究开发这类工具软件，以满足网络环境中各种流式应用开发的需要。

这种方法的基本思想是首先把基本对象分析清楚，定义几个基类，然后从这些基类出发用继承手段定义若干子类（派生类），封装成一些包裹（Wrapper），形成应用软件的开发基础。这些类可统称为 SAP（Service Access Point，服务访问点）。这些类将以柄为基础的网络编程接口封装起来。如用 socket，便称为 SOCK SAP，用 TLI 为 TLI SAP，用 UNIX SVR4 STREAM 为 SPIPE SAP，用 UNIX 命名管道为 FIFO SAP 等。

4.1 基类

SOCK SAP 可以从三个方面定义基类（参见图 1）。

第一个基类：对于进程间通信 IPC 机制，从总的方面说，可以定义一个 C++ 类包裹，作为 IPC SAP 的根，由此继承开来，形成 IPC SAP 的层次结

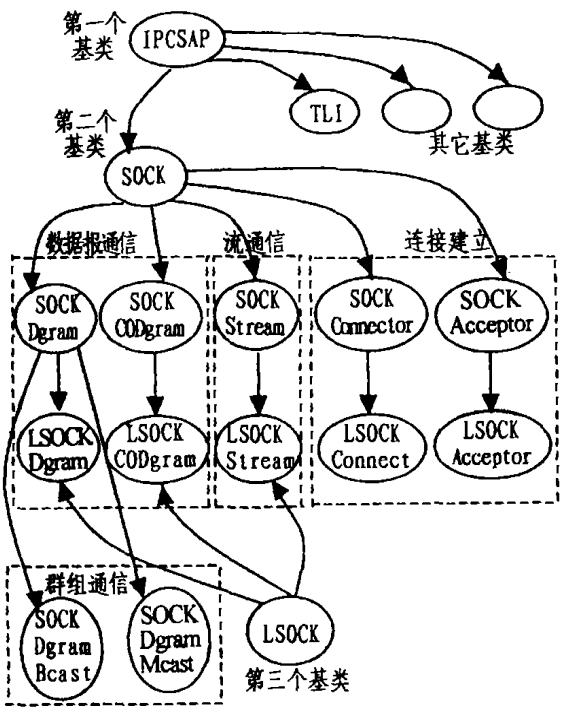

图 1　类范畴图一例

构。这个根类，称为 IPCSAP。它提供对于所有类而言共同的机制，诸如处理一些选项，设置柄值，开设异步 I/O 等。

第二个基类：对于 socket 通信机制，从 IPCSAP 继承下来，定义 SOCK SAP 层次结构的根，称为 SOCK。它提供所有 SOCK SAP 类所共有的机制，如打开和关闭本地通信端点，处理一些选项，如设置 socket 队列长度，开设群组通信等。同样，从 IPCSAP 也可继承定义 TLI SAP，SPIPE SAP，和 FIFO SAP 的根基类。

第三个基类：在本地宿主机上，对于进程的打开文件，为应用提供发送和接收文件柄的机制。这是为本地文件服务的类，称为 LSOCK。目前 UNIX SVR 和 BSD UNIX 均支持这一特点，但 Windows NT 不支持。其他类若从此类继承下来，就能获得这一特点。

IPCSAP，SOCK，LSOCK 三个基类派生出整个类体系，并保证后继类的共享性。这些类的对象本身不能被实例化，因为它们的构造子(constructor)是在类定义的保护(protected)节中说明的。LSOCK 和 SOCK 类依据网络地址格式和通信语义来加以区分。前者用 UNIX 路径名作为地址，只允许本地 IPC；后者用 IP 地址和端口号，允许本地及远地 IPC。

4.2　类范畴

可用画形表示类范畴关系，范畴表示类的分类划分，如图 1 所示，为 SAP 类范畴图。图中椭圆圈表示类，箭头表示继承关系，虚线框划出各类范畴。

4.3　连接建立

一般说，客户与服务器连接角色是不对称的。前者是主动的，后者是被动的。服务器被动地侦听由客户主动发来的连接请求。这种主被动连接建立模式以及数据传送模式由下列面向连接的 SOCK SAP 类所控制。

(1)SOCK Connector 和 LSOCK Connector。这两个连接子是一种工厂(factory)，它们能主动地建立新的通信端点，并与对应(远地或本地)端点相连接，产生相应的 SOCK Stream 和 LSOCK Stream 连接端点对象。

(2)SOCK Acceptor 和 LSOCK Acceptor。这两个接受子也是一种工厂，它们响应从远地或本地端点发来的主动连接请求，被动地建立新的通信端点，并建立连接，产生相应的 SOCK Stream 和 LSOCK Stream 连接端点对象。

上述连接子和接受子本身都是对象类；它们只是工厂，产生 Stream 对象，以备传送数据，但它们本身并不提供发送和接收数据的方法。这些工厂是强类型接口，它们能够在编译时检查并防止本地和非本地 Stream 对象之间的误用，而 socket 只能在运行时检查出这类误用。

4.4　流通信

如上所述，连接子和接受子产生 SOCK Stream 和 LSOCK Stream 对象类，后两者则提供传送数据的机制，即方法(method)。SOCK Stream 对象可在不同宿主机(host)的进程之间交换数据，LSOCK Stream 仅能在本地宿主机上进程间交换数据。

流通信(streaming)是一种面向连接的通信方式。由上述 Stream 类可提供读写方法。

方法 send 和 recv 是重载(overloading)的，是一种多形函数(polymorphism)。它们实现 UNIX write 和 read 的语义。这些类还可以提供其他语义的读写方法。

4.5　数据报通信

socket 接口也提供无连接服务，利用 UDP 和 IP 协议实现。这是一种低可靠的数据服务，并不能保证消息一定到达目的地。在某些容许一定程度信息丢失的应用中为了提高效率可采用这种方式，同时它也作为高层可靠协议的基础。

SOCK SAP 将 socket 数据报封装在如下类中。

(1)SOCK Dgram 和 LSOCK Dgram。这两类分别提供本地远地均可的和仅本地的进程间数据报通信。在 send 和 recv 操作中必须随同每个数据报一起给出相应的发送或接收地址。LSOCK Dgram 类继承 SOCK Dgram 和 LSOCK 的所有操作。

(2)SOCK CODgram 和 LSOCK CODgram。这两类提供有连接数据报机制。在 send 和 recv 中可以忽略地址。这种有连接的方式只是为了语法上的需要,数据报的基本语义并没有改变。传送仍然是低可靠的。

4.6 群组通信

群组通信提供两类通信模式:一种是广播模式,另一种是多播模式。

(1)SOCK Dgram Bcast。这个类提供广播 UDP 数据机制,可向本地及远地进程广播数据报,只要相应宿主机与本地子网接通。

(2)SOCK Dgram Mcast。这个类提供向多个本地及远地进程发送 UDP 数据报机制。这是一种向特定群组广播的通信接口。

以上两类接口都避免终端用户去考虑下层广播或多播的细节。

4.7 网络寻址

在当前异构网络和操作系统平台广泛互连的时代,要设计一种统一高效网络寻址方式是十分困难的。例如,在 Internet 通信域范围内,现利用两个字段表示:4 个字节 IP 地址和 2 个字节端口号。前者在 Internet 上唯一标识宿主机;后者将接收来的数据多路分解成各个单路单元,并传送给相应的客户或服务器进程。而在 UNIX 通信域范围内,仅在本地宿主机内通信,使用 UNIX 路径名寻址,最长可达 108 字节。

现有 socket 寻址方式是烦琐而易出错的。开发者首先必须显式地将地址所有字节初始化为 0,并且显式地指定投向。而在 SOCK SAP 寻址方式中,它把细节封装起来。图 2 表示 SOCK SAP 寻址类范畴。其中 Addr 是基类,它的构造子保证正确无误地初始化。三个派生类保证三种不同的通信子域,封装各自的寻址细节。UNIX Addr 子类对应于 LSOCK 类,INET Addr 子类对应于 SOCK 和 TLI 子类,SPIPE Addr 子类对应于 STREAM 子类。这种寻址结构也保证了可扩展性,以后可方便地加入新的通信域。

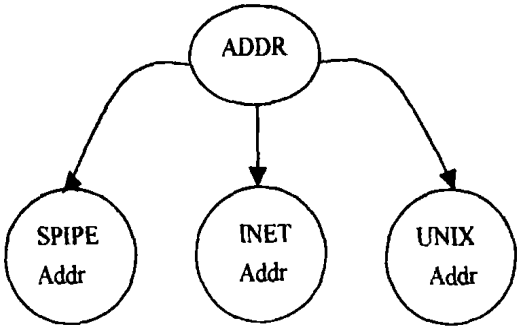

图 2 SOCK SAP 寻址类范畴

5 中层方法设计原则

中层方法利用面向对象技术将低层机制封装起来,形成所谓包裹,如上节所述 SOCK SAP 类范畴那样。在设计这些包裹时需遵循下列原则。

5.1 编译时保证类型安全性

如前所述,socket 的不足在于接口中类型安全性差。而 SOCK SAP 用构造子进行对象初始化,且对其对象仅能实施合法的操作。SOCK SAP 是强类型的,因此在编译时就能排除非法操作。返回值也只能用于传递操作成功或失败的信息。这就减少了赋值表达式误用的可能性。

5.2 有控制地放松类型安全性

为了提高运行效率,应当提供能直接访问下层机制的一些"吊钩"。例如在 IPC SAP 根类中有处理柄的方法:get_handle 和 set_handle。这使应用软件可以超越中层方法的类型检查机制而直接与 UNIX 涉及柄的系统调用相接口。

5.3 应尽量简化公用成分

在设计一些公用成分时,能尽量加以简化。具体有:

对公用方法的参数,凡能提供缺省结构应尽量提供,以减轻程序员每次运用这些公用方法时都要提供参数初始值的负担。如 SOCK Connector 构造子是一个公用成分,其六个参数中有四个参数可提供缺省值。

在定义接口时应尽量减少不必要的细节,将某一特定抽象的细节加以局部化,不必扩散到这个抽象的每次运用中。例如 IPC SAP 根类派生一些子类,这些子类实施各种类型的通信(面向连接或无连接的)及各种连接角色(主动的或被动的)。SOCK Acceptor 类只允许被动角色的相关操作。SOCK SAP 中发送和接收被打开文件柄的调用接口较简单,而直接用 UNIX 调用就较复杂。LSOCK 类中传送 socket 描述子也很简洁。

在定义操作时,尽量将多个相近操作合并在一个操作中,如 SOCK_Acceptor 是一个工厂,它将

socket 的 socket(),bind(),listen()等多个操作合并在一个 acceptor()中。

5.4 用层次化的类范畴替代线性接口

socket 接口是一维线性的,而中层方法是按面向对象原则构成类,类范畴是层次化的。这一原则保证最大限度的类成分的重用性和共享性。这充分利用了面向对象的方法的继承性和封装性。

继承性支持将各种功能性分解为不同的子类。例如,有些操作系统不支持对已打开文件的柄的传送,如 Windows NT,因此可以方便地将 LSOCK 类从继承性层次结构中略去,而不影响 SOCK SAP 的其它类接口。而 SVR 和 BSD 版 UNIX 均支持传送已打开文件的柄,故可以传送 LSOCK 子类。

继承性也支持代码重用性,改善模块化。基类反映了类范畴的共性,而派生类则反映了其个性。

5.5 用参数化类型增强可移植性。

使用 C++类将 socket 包裹起来,而不是单独使用 C 函数,使得可以通过参数化类型将网络编程接口整套地加以改换,而不需要个别接口单独加以改换,这样有助于改善可移植性。参数化类型方法保证应用程序不依赖于具体的网络编程接口。针对不同的操作系统平台,可以实施不同的实例化,例如使用 SOCK SAP 还是 TLI SAP。

5.6 采用行间扩展函数对性能关键方法加以定义

为了提高包裹的运行效率,在其中对性能敏感的执行路径上,相应的方法应当用 C++行间扩展函数来加以定义,以减少这些方法的运行时开销。因为这些方法通常只有几行,非常短,用行间扩展既节省时间,又节省空间。这也意味着应当尽量少用虚拟函数,因为大多数 C++编译器对虚拟函数的优化做得并不好。

5.7 定义附加类以隐藏易出错误细节

在 C 编程中一些易出错的细节,比如 socket 寻址,可定义附加类将它们隐蔽起来,例如 IPC SAP 定义了 Addr 类范畴,它支持各种各样的网络寻址格式,只要用一个安全的 C++接口即可,避免直接使用 C 的 struct 定义的一系列数据结构,也就避免了相应的出错可能性。

结语 网络环境中流式数据传输应用对通信性能的要求很高,如实时图象传输、影视会议等应用对网络带宽和延迟性要求比较敏感。但当前 CORBA 的实现尚不支持这些性能的要求,因为其中存在较多的数据复制、解复用和内存管理等开销。这些开销在低速网络上对性能影响相对来说并不明显,但在诸如 ATM 和 FDDI 这种高速网络上对性能影响就很明显。

面向对象的中层方法(如 ACE C++包裹),既解决了 CORBA 的效率问题,又隐蔽了低层编程细节,改善了可移植性。但这种方法的主要不足是,对可靠性和可用性、对象定位和选择的灵活性、对事务处理的支持、安全性、进程激活的延迟、异种计算机体系结构之间二进制数据的交换等问题,没有得到充分的考虑。比如用 ACE 包裹方法,对表示层中的交换,必须显式地加以编程。因此对于数据类型较为简单的应用,这种包裹方法才是有用的。

C++包裹可与 CORBA 结合在一起,以提高流式应用的性能。比如,利用 CORBA 作为信令机制,用以与地点无关的方式标识通信的端点;而包裹方法被用来建立端到端 TCP 连接,并在该连接上高效地传输大群数据。这种策略吸取了两者的强处,而避开了两者的弱处。

ACE C++包裹已被用于 UNIX 各个版本和 Windows NT,形成当前一些商业产品,如 Bellcore 和西门子 Q-port ATM 信令软件产品,Ericsson EOS 通信监控应用系列,Motorola 铱项目的系统控制部分,以及 Kodak 保健图像系统中企业范围医药图形发送系统。

这种面向对象的中层方法是一种有效的、有发展前途的网络应用软件开发途径。

参考文献

1 Schmidt D C, et al. Object-Oriented Components for High-Speed Network Programming. In: Proc. of the 1st Conf. on Object-Oriented Techniques and Systems. USENIX, June 1995

2 Pyarali I, et al. Design and Performance of an Object-Oriented Framework for High-Speed Electronic Medical Imaging. Computing Systems Journal, USENIX, 1997, 9(3)

3 Vinoski S. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. IEEE Communication Magazine, 1997, 4(2)

4 Schmidt D C, Vinoski S. Object Interconnections. (Columnl~Column12), C++ Report Magazine, 1995~1998 issues.