第25章 SNMP：简单网络管理协议

25.1 引言

随着网络技术的飞速发展，网络的数量也越来越多。而网络中的设备来自各个不同的厂家，如何管理这些设备就变得十分重要。本章的内容就是介绍管理这些设备的标准。

基于TCP/IP的网络管理包含两个部分：网络管理站（也叫管理进程， manager）和被管的网络单元（也叫被管设备）。被管设备种类繁多，例如：路由器、 X 终端、终端服务器和打印机等。这些被管设备的共同点就是都运行 TCP/IP协议。被管设备端和管理相关的软件叫做代理程序(agent)或代理进程。管理站一般都是带有彩色监视器的工作站，可以显示所有被管设备的状态(例如连接是否掉线、各种连接上的流量状况等)。

管理进程和代理进程之间的通信可以有两种方式。一种是管理进程向代理进程发出请求，询问一个具体的参数值（例如：你产生了多少个不可达的 ICMP端口？）。另外一种方式是代理进程主动向管理进程报告有某些重要的事件发生（例如：一个连接口掉线了）。当然，管理进程除了可以向代理进程询问某些参数值以外，它还可以按要求改变代理进程的参数值（例如：把默认的IP TTL值改为64）。

基于TCP/IP的网络管理包含3个组成部分：

1) 一个管理信息库MIB（Management Information Base）。管理信息库包含所有代理进程的所有可被查询和修改的参数。 RFC 1213 [McCloghrie and Rose 1991]定义了第二版的MIB，叫做MIB-II。

2) 关于MIB的一套公用的结构和表示符号。叫做管理信息结构 SMI（Structure of Management Information）。这个在RFC 1155 [Rose and McCloghrie 1990] 中定义。例如：SMI定义计数器是一个非负整数，它的计数范围是 0~4 294 967 295，当达到最大值时，又从0开始计数。

3) 管理进程和代理进程之间的通信协议，叫做简单网络管理协议 SNMP（Simple Network Management Protocol）。在RFC 1157 [Case et al. 1990]中定义。SNMP包括数据报交换的格式等。尽管可以在运输层采用各种各样的协议，但是在 SNMP中，用得最多的协议还是UDP。

上面提到的RFC所定义的SNMP叫做SNMP v1，或者就叫做SNMP，这也是本章的主要内容。到1993年为止，又有一些新的关于SNMP的 RFC发表。在这些RFC中定义的SNMP叫做第二版SNMP（SNMP v2），这将在25.12章节中讨论。

本章首先介绍管理进程和代理进程之间的协议，然后讨论参数的数据类型。在本章中将用到前面已经出现过的名词，如： IP、UDP和TCP等。我们在叙述中将举一些例子来帮助读者理解，这些例子和前面的某些章节相关。

25.2 协议

关于管理进程和代理进程之间的交互信息， SNMP定义了5种报文：

1) get-request操作：从代理进程处提取一个或多个参数值。

2) get-next-request操作：从代理进程处提取一个或多个参数的下一个参数值（关于“下一个（next）”的含义将在后面的章节中介绍）。

3) set-request操作：设置代理进程的一个或多个参数值。

4) get-response操作：返回的一个或多个参数值。这个操作是由代理进程发出的。它是前面3中操作的响应操作。

5) trap 操作：代理进程主动发出的报文，通知管理进程有某些事情发生。

前面的3个操作是由管理进程向代理进程发出的。后面两个是代理进程发给管理进程的（为简化起见，前面3个操作今后叫做get、get-next和set操作）。图25-1描述了这5种操作。

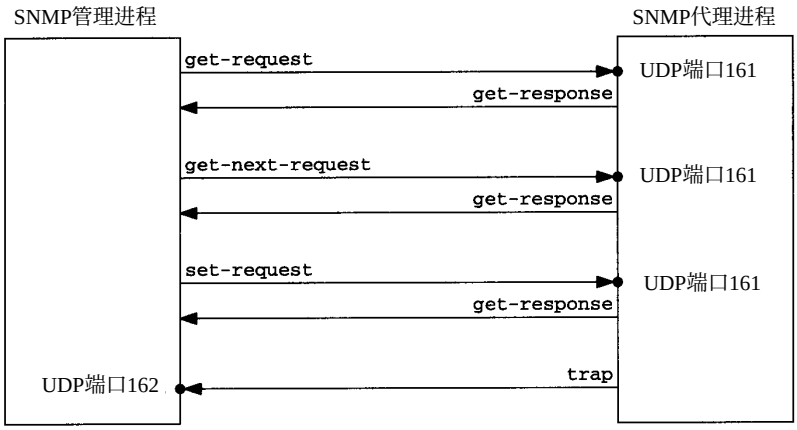

图25-1 SNMP的5种操作

既然这些操作中的前 4种操作是简单的请求 -应答方式（也就是管理进程发出请求，代理进程应答响应），而且在SNMP中往往使用UDP协议，所以可能发生管理进程和代理进程之间数据报丢失的情况。因此一定要有超时和重传机制。

管理进程发出的前面 3种操作采用UDP的161端口。代理进程发出的 Trap操作采用UDP的162端口。由于收发采用了不同的端口号，所以一个系统可以同时为管理进程和代理进程（参见习题25.1）。

图25-2是封装成UDP数据报的5种操作的SNMP报文格式。

在图中，我们仅仅对IP和UDP的首部长度进行了标注。这是由于：SNMP报文的编码采用了ASN.1和BER，这就使得报文的长度取决于变量的类型和值。关于 ASN.1和BER的内容将在后面介绍。在这里介绍各个字段的内容和作用。

版本字段是0。该字段的值是通过SNMP版本号减去1得到的。显然0代表SNMP v1。

图25-3显示各种PDU对应的值（PDU即协议数据单元，也就是分组）。

共同体字段是一个字符串。这是管理进程和代理进程之间的口令，是明文格式。默认的值是public。

对于get、get-next和set操作，请求标识由管理进程设置，然后由代理进程在 get-response中返回。这种类型的字段我们在其他UDP应用中曾经见过（回忆一下在图14-3中DNS的标识字段，或者是图16-2中的事务标识字段）。这个字段的作用是使客户进程（在目前情况下是管理进程）能够将服务器进程（即代理进程）发出的响应和客户进程发出的查询进行匹配。这个

字段允许管理进程对一个或多个代理进程发出多个请求，并且从返回的众多 应答中进行分类。

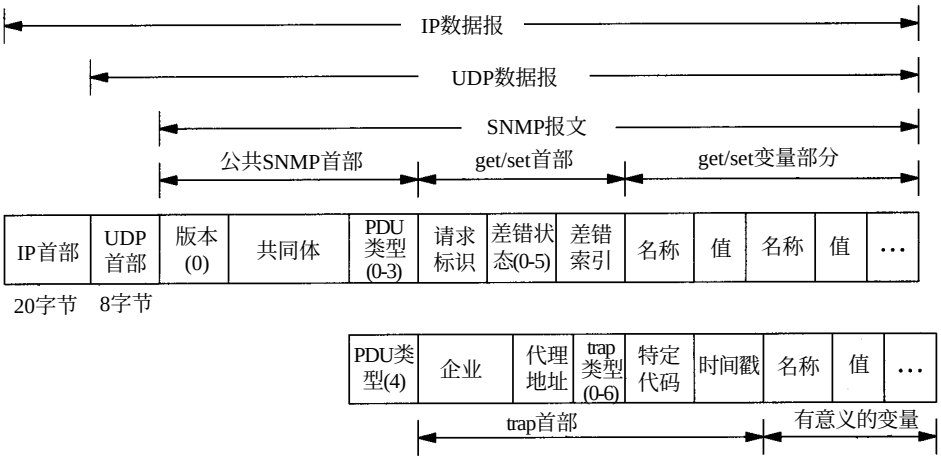

图25-2 SNMP报文的格式

差错状态字段是一个整数，它是由代理进程标注的，指明有差错发生。图25-4是参数值、名称和描述之间的对应关系。

差错索引字段是一个整数偏移量，指明当有差错发生时，差错发生在哪个参数。它是由代理进程标注的，并且只有在发生noSuchName、readOnly和badValue差错时才进行标注。

PDU类型	名　　称
0	get-request
1	get-next-request
2	get-response
3	set-request
4	trap

图25-3 SNMP报文中的PDU类型

差错状态	名　称	描　　述
0	noError	没有错误
1	tooBig	代理进程无法把响应放在一个SNMP消息中发送
2	noSuchName	操作一个不存在的变量
3	badValue	set操作的值或语义有错误
4	readOnly	管理进程试图修改一个只读变量
5	genErr	其他错误

图25-4 SNMP差错状态的值

在get、get-next和set的请求数据报中，包含变量名称和变量值的一张表。对于 get和get-next操作，变量值部分被忽略，也就是不需要填写。

对于trap操作符（PDU类型是4），SNMP报文格式有所变化。我们将在 25.10节中当讨论到trap时再详细讨论。

25.3 管理信息结构

SNMP中，数据类型并不多。在本节，我们就讨论这些数据类型，而不关心这些数据类型在实际中是如何编码的。

- INTEGER。一个变量虽然定义为整型，但也有多种形式。有些整型变量没有范围限制，有些整型变量定义为特定的数值（例如, IP的转发标志就只有允许转发时的1或者不允许转发时的2这两种），有些整型变量定义为一个特定的范围（例如, UDP和TCP的端口号就从0到65535）。
- OCTER STRING。0或多个8 bit字节，每个字节值在0~255之间。对于这种数据类型和

下一种数据类型的BER编码，字符串的字节个数要超过字符串本身的长度。这些字符串不是以NULL结尾的字符串。

- DisplayString。0或多个8 bit字节，但是每个字节必须是ASCII码（26.4中有ASCII字符集）。在MIB-II中，所有该类型的变量不能超过255个字符（0个字符是可以的）。
- OBJECT IDENTIFIER将在下一节中介绍。
- NULL。代表相关的变量没有值。例如，在get或get-next操作中，变量的值就是NULL，因为这些值还有待到代理进程处去取。
- IpAddress。4字节长度的OCTER STRING，以网络序表示的IP地址。每个字节代表IP地址的一个字段。
- PhysAddress。OCTER STRING类型，代表物理地址（例如以太网物理地址为6个字节长度）。
- Counter。非负的整数，可从0递增到 $2^{32}-1$（4 294 976 295）。达到最大值后归0。
- Gauge。非负的整数，取值范围为从0到4 294 976 295(或增或减)。达到最大值后锁定，直到复位。例如，MIB中的tcpCurrEstab就是这种类型的变量的一个例子，它代表目前在ESTABLISHED或CLOSE_WAIT状态的TCP连接数。
- TimeTicks。时间计数器，以0.01秒为单位递增，但是不同的变量可以有不同的递增幅度。所以在定义这种类型的变量的时候，必须指定递增幅度。例如，MIB中的sysUpTime变量就是这种类型的变量，代表代理进程从启动开始的时间长度，以多少个百分之一秒的数目来表示。
- SEQUENCE。这一数据类型与C程序设计语言中的“structure”类似。一个SEQUENCE包括0个或多个元素，每一个元素又是另一个ASN.1数据类型。例如，MIB中的UdpEntry就是这种类型的变量。它代表在代理进程侧目前“激活”的UDP数量（“激活”表示目前被应用程序所用）。在这个变量中包含两个元素：
  1) IpAddress类型中的udpLocalAddress，表示IP地址。
  2) INTEGER类型中的udpLocalPort，从0到65535，表示端口号。
- SEQUENDE OF。这是一个向量的定义，其所有元素具有相同的类型。如果每一个元素都具有简单的数据类型，例如是整数类型，那么我们就得到一个简单的向量（一个一维向量）。但是我们将看到，SNMP在使用这个数据类型时，其向量中的每一个元素是一个SEQUENCE（结构）。因而可以将它看成为一个二维数组或表。

例如，名为udpTable的UDP监听表(listener)就是这种类型的变量。它是一个二元的SEQUENCE变量。每个二元组就是一个UdpEntry。如图25-5所示。

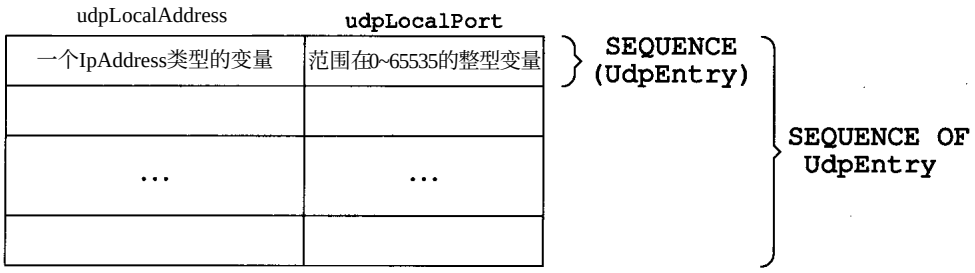

图25-5 表格形式的udpTable 变量

在SNMP中，对于这种类型的表格并没有标注它的列数。但在 25.7节中，我们将看到 get-next操作是如何判断已经操作到最后一列的情况。同时，在 25.6节中，我们还将介绍管理进程如何表示它对某一行数据进行 get或set操作。

25.4 对象标识符

对象标识是一种数据类型，它指明一种“ 授权 ”命名的对象。“ 授权 ”的意思就是这些标识不是随便分配的，它是由一些权威机构进行管理和分配的。

对象标识是一个整数序列，以点（“ . ”）分隔。这些整数构成一个树型结构，类似于 DNS（图14-1）或Unix的文件系统。对象标识从树的顶部开始，顶部没有标识，以 root表示（这和Unix中文件系统的树遍历方向非常类似）。

图25-6显示了在SNMP中用到的这种树型结构。所有的 MIB变量都从1.3.6.1.2.1这个标识开始。

树上的每个结点同时还有一个文字名。例如标识 1.3.6.1.2.1就和`iso.org.dod.internet.memt.mib`对应。这主要是为了人们阅读方便。在实际应用中，也就是说在管理进程和代理进程进行数据报交互时，MIB变量名是以对象标识来标识的，当然都是以1.3.6.1.2.1开头的。

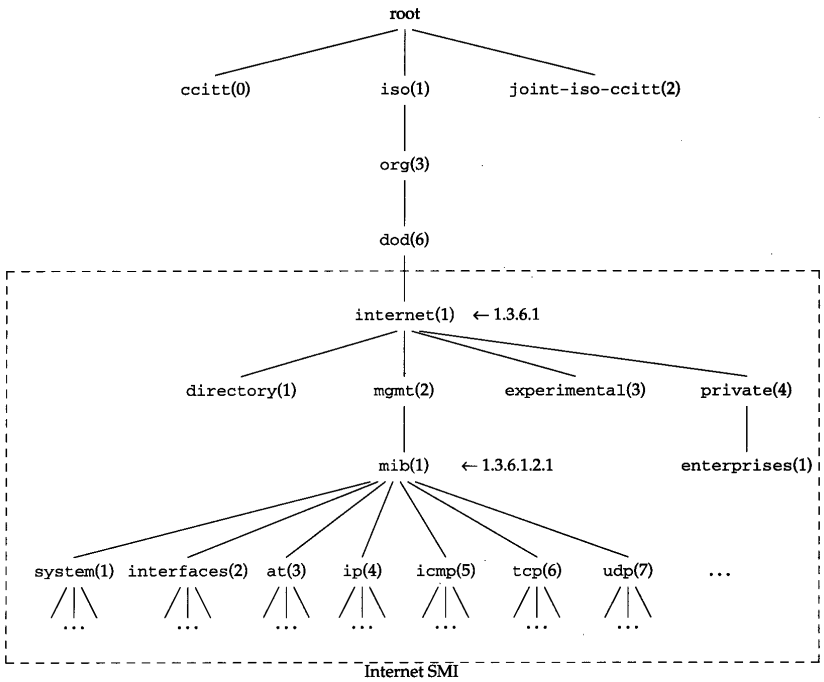

图25-6 管理信息库中的对象标识

在图25-6中，我们除了给出了`mib`对象标识外，还给出了`iso.org.dod.internet.private.enterprises`（1.3.6.1.4.1）这个标识。这是给厂家自定义而预留的。在 Assigned Number RFC中列出了在该结点下大约400个标识。

25.5 管理信息库介绍

所谓管理信息库，或者MIB，就是所有代理进程包含的、并且能够被管理进程进行查询和设

置的信息的集合。我们在前面已经提到了在RFC 1213 [McColghrie 和Rose 1991]中定义的MIB-II。

如图25-6所示，MIB被划分为若干个组，如system、interfaces、at（地址转换）和ip组等。

在本节，我们仅仅讨论UDP组中的变量。这个组比较简单，它包含几个变量和一个表格。在下一节，我们将以UDP组为例，详细讲解什么是实例标识（instance identification），什么是字典式排序(lexicographic ordering)以及和这些概念有关的一些简单例子。在这些例子之后，在25.8节我们继续回到MIB，描述MIB中的其他一些组。

在图25-6中我们画出了udp组在mib的下面。图25-7就显示了UDP组的结构。

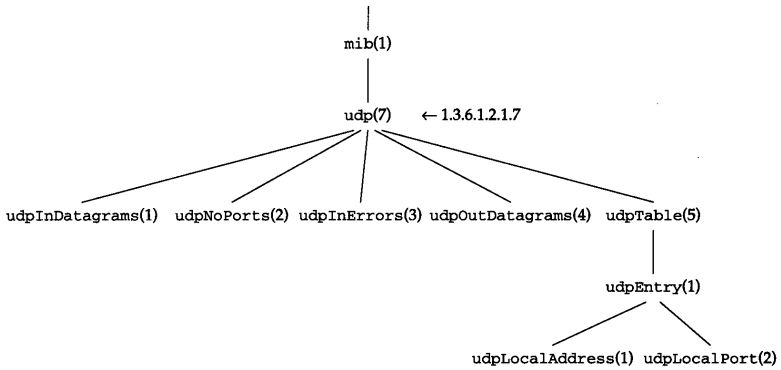

图25-7 UDP组的结构

在该组中，包含4个简单变量和1个由两个简单变量组成的表格。图 25-8描述了这4个简单变量。

名　　称	数据类型	R/W	描　　述
udpInDatagrams	Counter		UDP数据报输入数
udpNoPorts	Counter		没有发送到有效端口的UDP数据报个数
udpInErrors	Counter		接收到的有错误的UDP数据报个数(例如检验错误)
udpOutDatagrams	Counter		UDP数据报输出数

图25-8 UDP组下的简单变量

在本章中，我们就以图 25-8的格式来描述所有的MIB变量。“ R/W ”列如果为空，则代表该变量是只读的；如果变量是可读可写的，则以“ · ”符号来表示。哪怕整个组中的变量都是只读的，我们也将列出“ R/W ”列，以提示读者管理进程只能对这些变量进行查询操作(上图UDP组我们就是这样做的)。同样，如果变量类型是INTEGET类型并且有范围约束，我们也将标明它的下限和上限，就如我们在下图中描述 UDP端口号所做的一样。

图25-9描述了在udpTable中的两个简单变量。

UDP监听表，索引=<udpLocalAddress>.<udpLocalPort>			
名　　称	数据类型	R/W	描　　述
udpLocalAddress	IpAddress		监听进程的本地IP地址。0.0.0.0代表接收任何接口的数据报
udpLocalPort	[0..65535]		监听进程的本地端口号

图25-9 udpTable 中的变量

每次当我们以SNMP表格形式来描述MIB变量时，表格的第1行都表示索引的值，它是表

格中的每一列的参考。在下一节中读者将看到的一些例子也是这样做的。

Case图

在图25-8中，前3个计数器是有相互关系的。Case图真实地描述了一个给出的MIB组中变量之间的相互关系。图25-10就是UDP组的Case图。

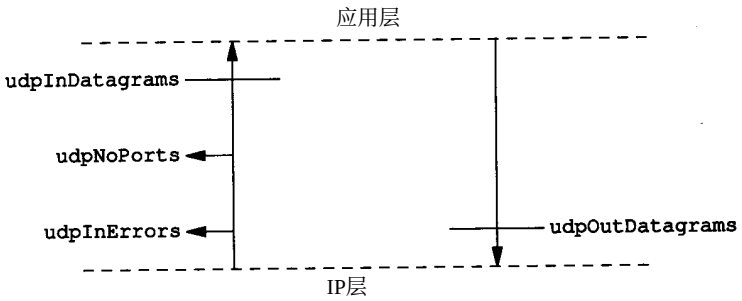

图25-10 UDP组的Case图

这张图表明，发送到应用层的UDP数据报的数量（udpInDatagrams）就是从IP层送到UDP层的UDP数据报的数量，当然udpInError和udpNoPorts也类似。同样，发送到IP层的UDP数据报的数量（udpOutDatagrams）就是从应用层发出的UDP数据报的数量。这表明udpInDatagram不包括udpInError和udpNoPorts。

在深入讲解MIB的时候，这些Case图被用来验证：分组的所有数据路径都是被计数的。[Rose 1994] 中显示了所有MIB组的Case图。

25.6 实例标识

当对MIB变量进行操作，如查询和设置变量的值时，必须对MIB的每个变量进行标识。首先，只有叶子结点是可操作的。SNMP没法处理表格的一整行或一整列。回到图25-7，在图25-8和图25-9中描述过的变量就是叶子结点，而mib、udp、udpTable和udpEntry就不是叶子结点。

25.6.1 简单变量

对于简单变量的处理方法是通过在其对象标识后面添加“ .0 ”来处理的。例如图25-8中的计数器udpInDatagrams，它的对象标识是1.3.6.1.2.1.7.1，它的实例标识是1.3.6.1.2.1.7.1.0，相对应的文字名称是iso.org.dod.internet.mgmt.mib.udp.udpInDatagrams.0。

虽然这个变量处理后通常可以缩写为 udpInDatagrams.0，但我们还是要提醒读者在SNMP报文中（图25-2）该变量的名称是其对象的标识1.3.6.1.2.1.7.1.0。

25.6.2 表格

表格的实例标识就要复杂得多。回顾一下图25-8中的UDP 监听表。

每个MIB中的表格都指明一个以上的索引。对于 UDP监听表来说， MIB定义了包含两个变量的联合索引，这两个变量是：udpLocalAddress，它是一个IP地址；udpLocalPort，它是一个整数（在图25-9中的第1行就显示了这个索引）。

假设在UDP监听表中有3行具体成员：第1行的IP地址是0.0.0.0，端口号是67；第2行的IP

地址是0.0.0.0，端口号是161；第3行的IP地址是0.0.0.0，端口号是520。如图25-11所示。

这意味着系统将从端口 67（BOOTP服务器）、端口161（SNMP）和端口520（RIP）接受来自任何接口的UDP数据报。表格中的这 3行经过处理后的结果在图 25-12中显示。

udpLocalAddress	udpLocalPort
0.0.0.0	67
0.0.0.0	161
0.0.0.0	520

图25-11 UDP监听表

25.6.3 字典式排序

MIB中按照对象标识进行排序时有一个隐含的排序规则。 MIB表格是根据其对象标识按照字典的顺序进行排序的。这就意味着图 25-12中的6个变量排序后的情况如图 25-13所示。从这种字典式排序中可以得出两个重要的结论。

行	对象标识	简　　称	值
1	1.3.6.1.2.1.7.5.1.1.0.0.0.0.67 1.3.6.1.2.1.7.5.1.2.0.0.0.0.67	udpLocalAddress.0.0.0.0.67 udpLocalPort.0.0.0.0.67	0.0.0.0 67
2	1.3.6.1.2.1.7.5.1.1.0.0.0.0.161 1.3.6.1.2.1.7.5.1.2.0.0.0.0.161	udpLocalAddress.0.0.0.0.161 udpLocalPort.0.0.0.0.161	0.0.0.0 161
3	1.3.6.1.2.1.7.5.1.1.0.0.0.0.520 1.3.6.1.2.1.7.5.1.2.0.0.0.0.520	udpLocalAddress.0.0.0.0.520 udpLocalPort.0.0.0.0.520	0.0.0.0 520

图25-12 UDP监听表中行的实例标识

列	对象标识(字典序)	简　　称	值
1	1.3.6.1.2.1.7.5.1.1.0.0.0.0.67 1.3.6.1.2.1.7.5.1.1.0.0.0.0.161 1.3.6.1.2.1.7.5.1.1.0.0.0.0.520	udpLocalAddress.0.0.0.0.67 udpLocalAddress.0.0.0.0.161 udpLocalAddress.0.0.0.0.520	0.0.0.0 0.0.0.0 0.0.0.0
2	1.3.6.1.2.1.7.5.1.2.0.0.0.0.67 1.3.6.1.2.1.7.5.1.2.0.0.0.0.161 1.3.6.1.2.1.7.5.1.2.0.0.0.0.520	udpLocalPort.0.0.0.0.67 udpLocalPort.0.0.0.0.161 udpLocalPort.0.0.0.0.520	67 161 520

图25-13 UDP监听表的字典式排序

1) 在表格中，一个给定变量（在这里指udpLocalAddress）的所有实例都在下个变量（这里指udpLocalPort）的所有实例之前显示。这暗示表格的操作顺序是“先列后行”的次序。这是由于对对象标识进行字典式排序所得到的，而不是按照人们的阅读习惯而排列的。

2) 表格中对行的排序和表格中索引的值有关。在图25-13中，67的字典序小于161，同样161的字典序小于520。

图25-14描述了例子中UDP监听表的这种“先列后行”的次序。

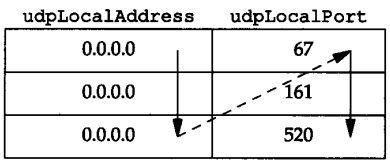

图25-14 按“先列后行”次序显示的UDP监听表

在下节中，讲述到get-next操作时，同样还会遇到这种“先列后行”的次序。

25.7 一些简单的例子

在本节中，我们将介绍如何从SNMP代理进程处获取变量的值。对代理进程进行查询的软件属于ISODE系统，叫做snmpi。两者在[Rose 1994]中有详细的介绍。

25.7.1 简单变量

对一个路由器取两个UDP组的简单变量值：

```
sun % snmpi -a gateway -c secret

snmpi> get udpInDatagrams.0 udpNoPorts.0
udpInDatagrams.0=616168
udpNoPorts.0=33

snmpi> quit
```

其中，-a选项代表要和之通信的代理进程名称，-c选项表示SNMP的共同体名。所谓共同体名，就是客户进程（在这里指snmpi）提供、同时能被服务器进程（这里指代理进程gateway）所识别的一个口令，共同体名称是管理进程请求的权限标志。代理进程允许客户进程用只读共同体名对变量进行读操作，用读写共同体名对变量进行读和写操作。

Snmpi程序的输出提示符是snmpi>，在后面可以键入如get这样的命令，该软件将把它转化为SNMP中的get-request报文。当结束时，键入quit就退出（在后面的例子中，我们将省略掉quit的操作）。

图25-15显示的是对于这个例子tcpdump的两行输出结果。

```
1  0.0                    sun.1024 > gateway.161: GetRequest(42)
                          1.3.6.1.2.1.7.1.0 1.3.6.1.2.1.7.2.0

2  0.348875 (0.3489)      gateway.161 > sun.1024: GetResponse(46)
                          1.3.6.1.2.1.7.1.0=616168
                          1.3.6.1.2.1.7.2.0=33
```

图25-15 简单SNMP查询操作tcpdump 的输出结果

对这两个变量的查询请求是封装在一个UDP数据报中的，而响应也在一个UDP数据报中。

显示的变量是以其对象标识的形式显示的，这是在SNMP报文中实际传输的内容。我们必须指定这两个变量的实例是0。注意，变量的名称（它的对象标识）同样也在响应中返回。在下面我们将看到对于get-next操作这是必需的。

25.7.2 get-next操作

get-next操作是基于MIB的字典式排序的。在下面的例子中，首先向代理进程询问UDP后的下一个对象标识（由于不是一个叶子对象，没有指定任何实例）。代理进程将返回UDP组中的第1个对象，然后我们继续向代理进程取该对象的下一个对象标识，这时候第2个对象将被返回。重复上面的步骤直到取出所有的对象为止。

```
sun % snmpi -a gateway -c secret

snmpi> next udp
udpInDatagrams.0=616318

snmpi> next udpInDatagrams.0
udpNoPorts.0=33

snmpi> next udpNoPorts.0
udpInErrors.0=0
```

这个例子解释了为什么get-next操作总是返回变量的名称，这是因为我们向代理进程询问下一个变量，代理进程就把变量值和名称一起返回了。

采用这种方式进行get-next操作，我们可以想象管理进程只要做一个简单的循环程序，

就可以从MIB树的顶点开始，对代理进程一步步地进行查询，就可以得出代理进程处所有的变量值和标识。该方式的另外一个用处就是可以对表格进行遍历。

25.7.3 表格的访问

对于“先列后行”次序的UDP监听表，只要采用前面的简单查询程序一步一步地进行操作，就可以遍历整个表格。只要从询问代理进程 udpTable的下一个变量开始就可以了。由于udpTable不是叶子对象，我们不能指定一个实例，但是get-next操作依然能够返回表格中的下一个对象。然后就可以以返回的结果为基础进行下一步的操作，代理进程也会以“先列后行”的次序返回下一个变量，这样就可以遍历整个表格。我们可以看到返回的次序和图25-14相同。

```
sun % snmpi -a gateway -c secret

snmpi> next udpTable
udpLocalAddress.0.0.0.0.67=0.0.0.0

snmpi> next udpLocalAddress.0.0.0.0.67
udpLocalAddress.0.0.0.0.161=0.0.0.0

snmpi> next udpLocalAddress.0.0.0.0.161
udpLocalAddress.0.0.0.0.520=0.0.0.0

snmpi> next udpLocalAddress.0.0.0.0.520
udpLocalPort.0.0.0.0.67=67

snmpi> next udpLocalPort.0.0.0.0.67
udpLocalPort.0.0.0.0.161=161

snmpi> next udpLocalPort.0.0.0.0.161
udpLocalPort.0.0.0.0.520=520

snmpi> next udpLocalPort.0.0.0.0.520
snmpInPkts.0=59
```

我们已完成了对UDP监听表的操作

但是管理进程如何知道已经到达表格的最后一行呢？既然get-next操作返回结果中包含表格中的下一个变量的值和名称，当返回的结果是超出表格之外的下一个变量时，管理进程就可以发现变量的名称发生了较大的变化。这样就可以判断出已经到达表格的最后一行。例如在我们的例子中，当返回的是snmpInPkts变量的时候就代表已经到了UDP监听表的最后一个变量了。

25.8 管理信息库（续）

现在继续讨论MIB。我们仅仅介绍下列MIB组：system（系统标识）、if（接口）、at（地址转换）、ip、icmp和tcp。

25.8.1 system组

system组非常简单，它包含7个简单变量（例如，没有表格）。图25-16列出了system组的名称、数据类型和描述。

可以对netb路由器查询一些简单变量：

```
sun % snmpi -a netb -c secret

snmpi> get sysDescr.0 sysObjectID.0 sysUpTime.0 sysServices.0
sysDescr.0="Epilogue Technology SNMP agent for Telebit NetBlazer"
sysObjectID.0=1.3.6.1.4.1.12.42.3.1
sysUpTime.0=22 days, 11 hours, 23 minutes, 2 seconds (194178200 timeticks)
sysServices.0=0xc<internet,transport>
```

名　称	数据类型	R/W	描　　述
sysDescr	DisplayString		系统的文字描述
sysObjectID	ObjectID		在子树1.3.6.1.4.1中的厂商标识
sysUpTime	TimeTicks		从系统的网管部分启动以来运行的时间（以百分之一秒为计算单位）
sysContact	DisplayString	•	联系人的名字及联系方式
sysName	DisplayString	•	结点的完全合格的域名(FQDN)
sysLocation	DisplayString	•	结点的物理位置
sysServices	[0…127]	•	指示结点提供的服务的值。该值是此结点所支持的OSI模型中层次的和。根据所提供的服务，将下面的一些值相加：0x01(物理层)、0x02（数据链路层）、0x04（互联网层）、0x08（端到端的运输层）和 0x40（应用层）

图25-16　system组中的简单变量

回到图25-6中，system的对象标识符在internet.private.enterprises组（1.3.6.1.4.1）中，在Assigned Numbers RFC文档中可以确定下一个对象标识符（12）肯定是指派给了厂家（Epilogue）。

同时还可以看出，sysServices变量的值是4与8的和，它支持网络层（例如选路）和运输层的应用（例如端到端）。

25.8.2　interface组

名　称	数据类型	R/W	描述
ifNumber	INTEGER		系统上的网络接口数

图25-17　if组中的简单变量

在本组中只定义了一个简单变量，那就是系统的接口数量，如图25-17所示。

在该组中，还有一个表格变量，有 22列。表格中的每行定义了接口的一些特征参数。如图25-18所示。

可以向主机sun查询所有这些接口的变量。如3.8节中所示，我们还是希望访问三个接口，如果SLIP接口已经启动：

```
sun % snmpi -a sun                          首先看第一个接口的接口索引

snmpi> next ifTable
ifIndex.1=1

snmpi> get ifDescr.1 ifType.1 ifMtu.1 ifSpeed.1 ifPhysAddress.1
ifDescr.1="le0"
ifType.1=ethernet-csmacd(6)
ifMtu.1=1500
ifSpeed.1=10000000
ifPhysAddress.1=0x08:00:20:03:f6:42

snmpi> next ifDescr.1 ifType.1 ifMtu.1 ifSpeed.1 ifPhysAddress.1
ifDescr.2="sl0"
ifType.2=propPointToPointSerial(22)
ifMtu.2=552
ifSpeed.2=0
ifPhysAddress.2=0x00:00:00:00:00:00

snmpi> next ifDescr.2 ifType.2 ifMtu.2 ifSpeed.2 ifPhysAddress.2
ifDescr.3="lo0"
ifType.3=softwareLoopback(24)
ifMtu.3=1536
ifSpeed.3=0
ifPhysAddress.3=0x00:00:00:00:00:00
```

接口表，索引＝<*IfIndex*>			
名称	数据类型	R/W	描　述
ifIndex	INTEGER		接口索引，介于1和ifNumber之间
ifDescr	DisplayString		接口的文字描述
ifType	INTEGER		类型，例如：6 = 以太网，7 = 802.3以太网，9 = 802.5令牌环，23 = PPP，28 = SLIP,还有其他一些值
ifMtu	INTEGER		接口的MTU
ifSpeed	Gauge		以b/s为单位的速率
ifPhysAddress	PhysAddress		物理地址，对无物理地址的接口，以一串0表示（例如，串行链路）
ifAdminStatus	[1…3]	•	所希望的接口状态：1= 工作，2 = 不工作，3 = 测试
ifOperStatus	[1…3]	•	当前接口的状态：1= 工作，2 = 不工作，3 = 测试
ifLastChange	TimeTicks		当接口进入目前运行状态时sysUpTime的值
ifInOctets	Counter		收到的字节总数，包括组帧字符
ifInUcastPkts	Counter		交付给高层的单播分组数
ifInNUcastPkts	Counter		交付给高层的非单播（例如，广播或多播）分组数
ifInDiscards	Counter		收到的被丢弃的分组数，即使在分组中无差错（例如，无缓存空间）
ifInErrors	Counter		收到的由于差错被丢弃的分组数
ifInUnknownProtos	Counter		收到的由于未知的协议被丢弃的分组数
ifOutOctets	Counter		发送的字节总数，包括组帧字符
ifOutUcastPkts	Counter		从高层接收到的单播分组数
ifOutNUcastPkts	Counter		从高层接收到的非单播（如广播或多播）分组数
ifOutDiscards	Counter		发出的被丢弃的分组数，即使在分组中无差错（如无缓存空间）
ifOutErrors	Counter		发出的由于差错被丢弃的分组数
ifOutQLen	Gauge		在输出队列中的分组数
ifSpecific	ObjectID		对这种特定媒体类型的MIB定义的引用

图25-18　在接口表中的变量：ifTable

对于第1个接口，采用get操作提取5个变量值，然后用get-next操作提取第2个接口的相同的5个参数。对于第3个接口，同样采用get-next操作。

对于SLIP链路的接口类型，所报告的是一个点到点的专用串行链路，而不是　SLIP链路。此外，SLIP链路的速率没有报告。

这个例子对我们理解get-next操作和“先列后行”次序之间的关系十分重要。如果我们键入命令“next ifDescr.1”，则系统返回的是表格中的下一行所对应的变量，而不是同一行中的下个变量。如果表格是按照“先行后列”次序存放，我们就不能通过一个给定变量来读取下一个变量。

25.8.3　at组

地址转换组对于所有的系统都是必需的，但是在 MIB-II中已经没有这个组。从 MIB-II开

始，每个网络协议组（如IP组）都包含它们各自的网络地址转换表。例如对于IP组，网络地址转换表就是ipNetToMediaTable。

在该组中，仅有一个由3列组成的表格变量。如图25-19所示。

我们将用snmpi程序中的一个新命令来转储(dump)整个表格。向一个叫做kinetics的路由器（该路由器连接了一个TCP/IP网络和一个AppleTalk网络）查询其整个ARP高速缓存。命令的输出是字典式排序的整个表格内容。

地址转换表，索引＝<*atIfIndex*>.1.<*atNetAddress*>			
名　称	数据类型	R/W	描　述
atIfIndex	INTEGER	•	接口数：ifIndex
atPhysAddress	PhysAddress	•	物理地址。若设置为长度为0的字符串，则表示无效表项
atNetAddress	NetworkAddress	•	IP地址

图25-19　网络地址转换表：atTable

```
sun % snmpi -a kinetics -c secret dump at

atIfIndex.1.1.140.252.1.4=1
atIfIndex.1.1.140.252.1.22=1
atIfIndex.1.1.140.252.1.183=1
atIfIndex.2.1.140.252.6.4=2
atIfIndex.2.1.140.252.6.6=2

atPhysAddress.1.1.140.252.1.4=0xaa:00:04:00:f4:14
atPhysAddress.1.1.140.252.1.22=0x08:00:20:0f:2d:38
atPhysAddress.1.1.140.252.1.183=0x00:80:ad:03:6a:80
atPhysAddress.2.1.140.252.6.4=0x00:02:16:48
atPhysAddress.2.1.140.252.6.6=0x00:02:3c:48

atNetAddress.1.1.140.252.1.4=140.252.1.4
atNetAddress.1.1.140.252.1.22=140.252.1.22
atNetAddress.1.1.140.252.1.183=140.252.1.183
atNetAddress.2.1.140.252.6.4=140.252.6.4
atNetAddress.2.1.140.252.6.6=140.252.6.6
```

让我们来分析一下用tcpdump命令时的分组交互情况。当snmpi要转储整个表格时，首先发出一条get-next命令以取得表格的名称（在本例中是at），该名称就是要获取的第一个表项。然后在屏幕上显示的同时生成另一条get-next命令。直到遍历完整个表格的内容后才终止。

atIfIndex	atPhysAddress	atNetAddress
1	0xaa:00:04:00:f4:14	140.252.1.4
1	0x08:00:20:0f:2d:38	140.252.1.22
1	0x00:80:ad:03:6a:80	140.252.1.183
2	0x00:02:16:48	140.252.6.4
2	0x00:02:3c:48	140.252.6.6

图25-20　at表举例（ARP高速缓存）

图25-20显示了在路由器中实际表格的内容。

注意图中，接口2的AppleTalk协议的物理地址是32 bit的数值，而不是我们所熟悉的以太网的48 bit物理地址。同时请注意，正如我们所希望的那样，在图中有一条记录和netb路由器（其IP地址是140.252.1.183）有关。这是因为netb路由器和kinetics路由器在同一个以太网中（140.252.1），而且kinetics路由器必需采用ARP来回送SNMP响应。

25.8.4　ip组

ip组定义了很多简单变量和3个表格变量。图25-21显示了所有的简单变量。

名称	数据类型	R/W	描 述
ipForwarding	[1…2]	•	1代表系统正在转发IP数据报，2则代表不在转发
ipDefaultTTL	INTEGER	•	当运输层不提供TTL值时的默认TTL值
ipInReceives	Counter		从所有接口收到的IP数据报的总数
ipInHdrErrors	Counter		由于首部差错被丢弃的数据报数（例如，检验和差错，版本不匹配，TTL超过等）
ipInAddrErrors	Counter		由于不正确的目的地址被丢弃的IP数据报数
ipForwDatagrams	Counter		曾进行过一次转发尝试的IP数据报数
ipInUnknownProtos	Counter		具有无效协议字段的发往本地的IP数据报数
ipInDiscards	Counter		由于缓存空间不足被丢弃的收到的数据报数
ipInDelivers	Counter		交付到适当的协议模块的IP数据报数
ipOutRequests	Counter		传递给IP层来传输的IP数据报总数。不包括已经在ipForwDatagrams中计入的那些
ipOutDiscards	Counter		由于缓存空间不足被丢弃的输出数据报数
ipOutNoRoutes	Counter		由于找不到路由被丢弃的数据报数
ipReasmTimeout	INTEGER		在等待重装时已收到的数据报片被保留的最大秒数
ipReasmReqds	Counter		收到的需要进行重装的IP数据报片的数目
ipReasmOKs	Counter		已成功重装的IP数据报数
ipReasmFails	Counter		IP重装算法失败次数
ipFragOKs	Counter		被成功分片的IP数据报数
ipFragFails	Counter		需要进行分片但由于设置了“不分片”标志而不能分片的IP数据报数
ipFragCreates	Counter		由分片而产生的IP数据报片的数目
ipRoutingDiscards	Counter		所选择的选路表项即使是有效的但也要丢弃的数目

图25-21 ip组中的简单变量

ip组中的第一个表格变量是IP地址表。系统的每个IP地址都对应该表格中的一行。每行中包含了5个变量，如图25-22所示。

IP地址表，索引＝<*ipAdEntAddr*>			
名 称	数据类型	R/W	描 述
ipAdEntAddr	IpAddress		这一行的IP地址
ipAdEntIfIndex	INTEGER		对应的接口数：ifIndex
ipAdEntNetMask	IpAddress		对这个IP地址的子网掩码
ipAdEntBcastAddr	[0…1]		IP广播地址中的最低位的值。通常为1
ipAdEntReasmMaxSize	[0…65535]		在这个接口上收到的、能够进行重装的、最长的IP数据报

图25-22 IP地址表：ipAddrTable

同样可以向主机sun查询整个IP地址表：

```
sun % snmpi -a sun dump ipAddrTable

ipAdEntAddr.127.0.0.1=127.0.0.1
ipAdEntAddr.140.252.1.29=140.252.1.29
ipAdEntAddr.140.252.13.33=140.252.13.33
```

```
ipAdEntIfIndex.127.0.0.1=3                环回接口
ipAdEntIfIndex.140.252.1.29=2             SLIP接口
ipAdEntIfIndex.140.252.13.33=1            以太网接口

ipAdEntNetMask.127.0.0.1=255.0.0.0
ipAdEntNetMask.140.252.1.29=255.255.255.0
ipAdEntNetMask.140.252.13.33=255.255.255.224

ipAdEntBcastAddr.127.0.0.1=1              所有这三个都使用一个比特进行广播
ipAdEntBcastAddr.140.252.1.29=1
ipAdEntBcastAddr.140.252.13.33=1

ipAdEntReasmMaxSize.127.0.0.1=65535
ipAdEntReasmMaxSize.140.252.1.29=65535
ipAdEntReasmMaxSize.140.252.13.33=65535
```

输出的接口号码可以和图25-18中的输出进行比较，同样IP地址和子网掩码可以和3.8节中采用ifconfig命令时的输出进行比较。

ip组中的第二个表是IP路由表（请回忆一下我们在9.2节中讲到的路由表），如图25-23所示。访问该表中每行记录的索引是目的IP地址。

名　称	数据类型	R/W	描　述
ipRouteDest	IpAddress	•	目的IP地址。值0.0.0.0表示一个默认的表项
ipRouteIfIndex	INTEGER	•	接口数：ifIndex
ipRouteMetric1	INTEGER	•	主要的选路度量。这个度量的意义取决于选路协议（ipRouteProto）。-1表示未使用
ipRouteMetric2	INTEGER	•	可选的选路度量
ipRouteMetric3	INTEGER	•	可选的选路度量
ipRouteMetric4	INTEGER	•	可选的选路度量
ipRouteNextHop	IpAddress	•	下一跳路由器的IP地址
ipRouteType	INTEGER	•	路由类型：1 = 其他，2 = 无效路由，3 = 直接，4= 间接
ipRouteProto	INTEGER		选路协议：1 = 其他，4 = ICMP重定向，8 = RIP, 13 = OSPF, 14 = BGP, 以及其他
ipRouteAge	INTEGER	•	自从路由上次被更新或确定是正确的以后所经历的秒数
ipRouteMask	IpAddress	•	在和ipRouteDest相比较之前，掩码要与目的IP地址进行逻辑“与”
ipRouteMetric5	INTEGER	•	其他的选路度量
ipRouteInfo	ObjectID	•	对这种特定选路协议的MIB定义的引用

图25-23　IP路由表：ipRouteTable

图25-24显示的是用snmpi程序采用dump ipRouteTable命令从主机sun得到的路由表。在这张表中，已经删除了所有5个路由度量，那是由于这5条记录的度量都是-1。在列的标题中，对每个变量名称已经删除了ipRoute这样的前缀。

Dest	IfIndex	NextHop	Type	Proto	Mask
0.0.0.0	2	140.252.1.183	间接(4)	其他(1)	0.0.0.0
127.0.0.1	3	127.0.0.1	直接(3)	其他(1)	255.255.255.255
140.252.1.183	2	140.252.1.29	直接(3)	其他(1)	255.255.255.255
140.252.13.32	1	140.252.13.33	直接(3)	其他(1)	255.255.0.0
140.252.13.65	1	140.252.13.35	间接(4)	其他(1)	255.255.255.255

图25-24　路由器sun上的IP路由表

为比较起见，下面的内容是我们用netstat命令（在9.2节中曾经讨论过）格式显示的路由表信息。图25-24是按字典序显示的，这和netstat命令显示格式不同：

```
sun % netstat -rn
Routing tables
Destination          Gateway              Flags    Refcnt Use        Interface
140.252.13.65        140.252.13.35        UGH      0      115        le0
127.0.0.1            127.0.0.1            UH       1      1107       lo0
140.252.1.183        140.252.1.29         UH       0      86         sl0
default              140.252.1.183        UG       2      1628       sl0
140.252.13.32        140.252.13.33        U        8      68359      le0
```

ip组的最后一个表是地址转换表，如图 25-25所示。正如我们前面所说的， at组已经被删除了，在这里已经用IP表来代替了。

IP地址转换表，索引＝<*ipNetToMediaIfIndex*>.<*ipNetToMediaNetAddress*>			
名　　称	数 据 类 型	R/W	描　　述
ipNetToMediaIfIndex	INTEGER	•	对应的接口：ifIndex
ipNetToMediaPhysAddress	PhysAddress	•	物理地址
ipNetToMediaNetAddress	IpAddress	•	IP地址
ipNetToMediaType	[1…4]	•	映射的类型：1 = 其他，2 = 无效的，3 = 动态的，4 = 静态的。

图25-25 IP地址转换表：ipNetToMediaTable

这里显示的是系统sun上的ARP高速缓存信息：

```
sun % arp -a
svr4 (140.252.13.34) at 0:0:c0:c2:9b:26
bsdi (140.252.13.35) at 0:0:c0:6f:2d:40
```

相应的SNMP输出：

```
sun % snmpi -a sun dump ipNetToMediaTable

ipNetToMediaIfIndex.1.140.252.13.34=1
ipNetToMediaIfIndex.1.140.252.13.35=1
ipNetToMediaPhysAddress.1.140.252.13.34=0x00:00:c0:c2:9b:26
ipNetToMediaPhysAddress.1.140.252.13.35=0x00:00:c0:6f:2d:40
ipNetToMediaNetAddress.1.140.252.13.34=140.252.13.34
ipNetToMediaNetAddress.1.140.252.13.35=140.252.13.35
ipNetToMediaType.1.140.252.13.34=dynamic(3)
ipNetToMediaType.1.140.252.13.35=dynamic(3)
```

25.8.5 icmp组

icmp组包含4个普通计数器变量（ICMP报文的输出和输入数量以及 ICMP差错报文的输入和输出数量）和22个其他ICMP报文数量的计数器：11个是输出计数器，另外11个是输入计数器。如图25-26所示。

对于有附加代码的 ICMP报文（请回忆一下图 6-3中，有15种报文代表目的不可达），SNMP没有为它们定义专门的计数器。

25.8.6 tcp组

图25-27显示的是tcp组中的简单变量。其中的很多变量和图 18-12描述的TCP状态有关。

名　称	数据类型	R/W	描　述
icmpInMsgs	Counter		收到的ICMP报文总数
icmpInErrors	Counter		收到的有差错的ICMP报文数（例如，无效的ICMP检验和）
icmpInDestUnreachs	Counter		收到的ICMP目的站不可达报文数
icmpInTimeExcds	Counter		收到的ICMP超时报文数
icmpInParmProbs	Counter		收到的ICMP参数问题报文数
icmpInSrcQuenchs	Counter		收到的ICMP源站抑制报文数
icmpInRedirects	Counter		收到的ICMP重定向报文数
icmpInEchos	Counter		收到的ICMP回显请求报文数
icmpInEchosReps	Counter		收到的ICMP回显应答报文数
icmpInTimeStamps	Counter		收到的ICMP时间戳请求报文数
icmpInTimeStampReps	Counter		收到的ICMP时间戳应答报文数
icmpInAddrMasks	Counter		收到的ICMP地址掩码请求报文数
icmpInAddrMaskReps	Counter		收到的ICMP地址掩码应答报文数
icmpOutMsgs	Counter		输出的ICMP报文总数
icmpOutErrors	Counter		由于在ICMP报文中有一个问题（例如，缓存空间不足）而未发送的ICMP报文数
icmpOutDestUnreachs	Counter		发送的ICMP目的站不可达报文数
icmpOutTimeExcds	Counter		发送的ICMP超时报文数
icmpOutParmProbs	Counter		发送的ICMP参数问题报文数
icmpOutSrcQuenchs	Counter		发送的ICMP源站抑制报文数
icmpOutRedirects	Counter		发送的ICMP重定向报文数
icmpOutEchos	Counter		发送的ICMP回显请求报文数
icmpOutEchosReps	Counter		发送的ICMP回显应答报文数
icmpOutTimeStamps	Counter		发送的ICMP时间戳请求报文数
icmpOutTimeStampReps	Counter		发送的ICMP时间戳应答报文数
icmpOutAddrMasks	Counter		发送的ICMP地址掩码请求报文数
icmpOutAddrMaskReps	Counter		发送的ICMP地址掩码应答报文数

图25-26　icmp组中的简单变量

名　称	数据类型	R/W	描　述
tcpRtoAlgorithm	INTEGER		用来计算重传超时值的算法：1 =除下列值以外，2 = 固定的RTO, 3 = MIL-STD-1778附件B, 4 = Van Jacobson算法
tcpRtoMin	INTEGER		以毫秒计的最小重传超时值
tcpRtoMax	INTEGER		以毫秒计的最大重传超时值
tcpMaxConn	INTEGER		最大的TCP连接数。若为动态的，则值为- 1
tcpActiveOpens	Counter		从CLOSED到SYN_SENT的状态变迁数
tcpPassiveOpens	Counter		从LISTEN到SYN_RCVD的状态变迁数
tcpAttempFails	Counter		从SYN_SENT或SYN_RCVD到CLOSED的状态变迁数,加上从SYN_RCVD到LISTEN的状态变迁数
tcpEstabResets	Counter		从ESTABLISHED或CLOSE_WAIT状态到CLOSED的状态变迁数
tcpCurrEstab	Gauge		当前在ESTABLISHED或CLOSE_WAIT状态的连接数
tcpInSegs	Counter		收到的报文段的总数
tcpOutSegs	Counter		发送的报文段的总数，但将仅包含重传字节的除外
tcpRetransSegs			重传的报文段的总数
tcpInErrs	Counter		收到的具有一个差错(如无效的检验和)的报文段总数
tcpOutRsts	Counter		具有RST标志置位的报文段的总数

图25-27　tcp组中的简单变量

现在向系统sun查询一些tcp组变量：

```
sun % snmpi -a sun

snmpi> get tcpRtoAlgorithm.0 tcpRtoMin.0 tcpRtoMax.0 tcpMaxConn.0
tcpRtoAlgorithm.0=vanj(4)
tcpRtoMin.0=200
tcpRtoMax.0=12800
tcpMaxConn.0=-1
```

本系统（指SunOS4.1.3）使用的是Van Jacobson超时重传算法，超时定时器的范围在200 ms~12.8 s之间，并且对TCP连接数量没有特定的限制（这里的超时上限12.8 s恐怕有错，因为我们在21章中曾经介绍大多数应用的超时上限是64 s）。

tcp组还包括一个表格变量，即TCP连接表，如图25-28所示。对于每个TCP连接，都对应表格中的一条记录。每条记录包含5个变量：连接状态、本地IP地址、本地端口号、远端IP地址以及远端端口号。

索引＝<tcpConnLocalAddress>.<tcpConnLocalPort>.<tcpConnRemAddress>.<tcpConnRemPort>			
名　称	数据类型	R/W	描　述
tcpConnState	[1…12]	•	连接状态：1 = CLOSED, 2 = LISTEN, 3 = SYN_SENT, 4 = SYN_RCVD, 5 = ESTABLISHED, 6 = FIN_WAIT, 7 = FIN_WAIT, 8 = CLOSE_WAIT, 9 = LAST_ACK, 10 = CLOSING, 11 = TIME_WAIT, 12 = 删除TCB。管理进程对此变量可以设置的唯一值就是12（例如，立即终止此连接）
tcpConnLocalAddress	IpAddress		本地IP地址。0.0.0.0代表监听进程愿意在任何接口接受连接
tcpConnLocalPort	[1…65535]		本地端口号
tcpConnRemAddress	IpAddress		远程IP地址
tcpConnRemPort	[1…65535]		远程端口号

图25-28 TCP连接表：tcpConnTable

让我们看一看在系统sun上的这个表。由于有许多服务器进程在监听这些连接，所以我们只显示该表的一部分内容。在转储全部表格的变量之前，我们必需先建立两条TCP连接：

```
sun %rlogin gemini              gemini的IP地址是140.252.1.11
```

和

```
sun %rlogin localhost           IP地址应该是127.0.0.1
```

在所有的监听服务器进程中，我们仅仅列出了FTP服务器进程的情况，它使用21号端口。

```
sun % snmpi -a sun dump tcpConnTable

tcpConnState.0.0.0.0.21.0.0.0.0.0=listen(2)
tcpConnState.127.0.0.1.23.127.0.0.1.1415=established(5)
tcpConnState.127.0.0.1.1415.127.0.0.1.23=established(5)
tcpConnState.140.252.1.29.1023.140.252.1.11.513=established(5)

tcpConnLocalAddress.0.0.0.0.21.0.0.0.0.0=0.0.0.0
tcpConnLocalAddress.127.0.0.1.23.127.0.0.1.1415=127.0.0.1
tcpConnLocalAddress.127.0.0.1.1415.127.0.0.1.23=127.0.0.1
tcpConnLocalAddress.140.252.1.29.1023.140.252.1.11.513=140.252.1.29
```

```
tcpConnLocalPort.0.0.0.0.21.0.0.0.0.0=21
tcpConnLocalPort.127.0.0.1.23.127.0.0.1.1415=23
tcpConnLocalPort.127.0.0.1.1415.127.0.0.1.23=1415
tcpConnLocalPort.140.252.1.29.1023.140.252.1.11.513=1023

tcpConnRemAddress.0.0.0.0.21.0.0.0.0.0=0.0.0.0
tcpConnRemAddress.127.0.0.1.23.127.0.0.1.1415=127.0.0.1
tcpConnRemAddress.127.0.0.1.1415.127.0.0.1.23=127.0.0.1
tcpConnRemAddress.140.252.1.29.1023.140.252.1.11.513=140.252.1.11

tcpConnRemPort.0.0.0.0.21.0.0.0.0.0=0
tcpConnRemPort.127.0.0.1.23.127.0.0.1.1415=1415
tcpConnRemPort.127.0.0.1.1415.127.0.0.1.23=23
tcpConnRemPort.140.252.1.29.1023.140.252.1.11.513=513
```

对于rlogin到gemini，只显示一条记录，这是因为gemini是另外一个主机。而且我们仅仅能够看到连接的客户端信息（端口号是1023）。但是Telnet连接，客户端和服务器端都显示（客户端口号是1415，服务器端口号是23），这是因为这种连接通过环回接口。同时我们还可以看到，FTP监听服务器程序的本地IP地址是0.0.0.0。这表明它可以接受通过任何接口的连接。

25.9 其他一些例子

现在开始回答前面一些没有回答的问题，我们将用SNMP的知识进行解释。

25.9.1 接口MTU

回忆一下在11.6节的实验中，我们试图得出一条从netb到sun的SLIP连接的MTU。现在可以采用SNMP得到这个MTU。首先从IP路由表中取到SLIP连接（140.252.1.29）的接口号（ipRouteIfIndex），然后就可以用这个数值进入接口表并且取得想要的SLIP连接的MTU（通过SLIP的描述和数据类型）。

```
sun % snmpi -a netb -c secret

snmpi> get ipRouteIfIndex.140.252.1.29
ipRouteIfIndex.140.252.1.29=12
snmpi> get ifDescr.12 ifType.12 ifMtu.12
ifDescr.12="Telebit NetBlazer dynamic dial virtual interface"
ifType.12=other(1)
ifMtu.12=1500
```

可以看到，即使连接的类型是SLIP连接，但是MTU仍设置为以太网，其值为1500，目的可能是为了避免分片。

25.9.2 路由表

回忆一下在14.4节中，我们讨论了DNS如何进行地址排序的问题。当时我们介绍了从域名服务器返回的第1个IP地址是和客户有相同子网掩码的情况。还介绍了用其他的IP地址也会正常工作，但是效率比较低。现在我们从SNMP的角度来查阅路由表的入口，在这里将用到前面章节中和IP路由有关的很多相关知识。

路由器gemini是一个多接口主机，有两个以太网接口。首先确认一下两个接口都可以Telnet登录：

```
sun % telnet 140.252.1.11 daytime
Trying 140.252.1.11 ...
Connected to 140.252.1.11.
```

```
Escape character is '^]'.
Sat Mar 27 09:37:24 1993
Connection closed by foreign host.

sun % telnet 140.252.3.54 daytime
Trying 140.252.3.54 ...
Connected to 140.252.3.54.
Escape character is '^]'.
Sat Mar 27 09:37:35 1993
Connection closed by foreign host.
```

可以看出这两个地址的连接没有什么区别。现在我们采用 traceroute命令来看一下对于每个地址，是否有选路方面的不同：

```
sun % traceroute 140.252.1.11
traceroute to 140.252.1.11 (140.252.1.11), 30 hops max, 40 byte packets
 1  netb (140.252.1.183)  299 ms  234 ms  233 ms
 2  gemini (140.252.1.11)  233 ms  228 ms  234 ms

sun % traceroute 140.252.3.54
traceroute to 140.252.3.54 (140.252.3.54), 30 hops max, 40 byte packets
 1  netb (140.252.1.183)  245 ms  212 ms  234 ms
 2  swnrt (140.252.1.6)  233 ms  229 ms  234 ms
 3  gemini (140.252.3.54)  234 ms  233 ms  234 ms
```

可以看到：如果采用属于140.252.3子网的地址，就多了额外的一跳。下面解释造成这个额外一跳的原因。

图25-29是系统的连接关系图。从traceroute命令的输出结果可以看出主机gemini和路由器swnrt都连接了两个网段：140.252.3子网和140.252.1子网。

回忆一下在图4-6中，我们解释了路由器netb采用ARP代理进程，使得sun工作站好象是直接连接到140.252.1子网上的情况。我们忽略了sun和netb之间SLIP连接的调制解调器，因为这和我们这里的讨论不相关。

在图25-29中，我们用虚线箭头画出了当Telnet到140.252.3.54时的路径。返回的数据报怎么知道直接从gemini到netb，而不是从原路返回呢？我们采用在8.5节中介绍过的，带有宽松选路特性的traceroute版本来解释：

```
sun % traceroute -g 140.252.3.54 sun
traceroute to sun (140.252.13.33), 30 hops max, 40 byte packets
 1  netb (140.252.1.183)  244 ms  256 ms  234 ms
 2  * * *
 3  gemini (140.252.3.54)  285 ms  227 ms  234 ms
 4  netb (140.252.1.183)  263 ms  259 ms  294 ms
 5  sun (140.252.13.33)  534 ms  498 ms  504 ms
```

当在命令中指明是宽松源站选路时，swnrt路由器就不再有响应。看一下前面没有指明源站选路的traceroute命令输出，可以看出swnrt路由器是事实上的第2跳。超时数据必须这样设置的原因是：当数据报指定了宽松源站选路选项时，该路由器没有发生ICMP超时差错。所以在traceroute命令的输出中可以得出，返回路径是从gemini(TTL 3, 4和5)路由器直接到达netb路由器，而不通过swnrt路由器。

还剩下一个需要用SNMP来解释的问题就是：在netb路由器的路由表中，哪条信息代表寻径到140.252.3？该信息表示netb路由器把分组发送给swnrt而不是直接发送给gemini？用get命令来取下一跳路由器的值。

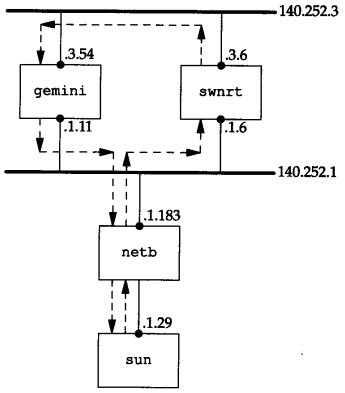

图25-29 例子中的网络拓扑结构

```
sun % snmpi -a netb -c secret get ipRouteNextHop.140.252.3.0
ipRouteNextHop.140.252.3.0=140.252.1.6
```

正如我们所看到发生的那样，路由表设置使得 netb路由器把分组发送到swnrt路由器。

为什么gemini路由器直接把分组回送给netb路由器？那是因为在gemini路由器端，它要回送的分组目的地址是140.252.1.29，而子网140.252.1是直接连接到gemini路由器上的。

从上面这个例子可以看出选路的策略。由于gemini是打算作一个多接口主机而不是路由器，所以默认的到140.253.3子网的路由器是swnrt。这是多接口主机和路由器之间差异的一个典型例子。

25.10 Trap

本章我们看到的例子都是从管理进程到代理进程的。当然代理进程也可以主动发送 trap到管理进程，以告诉管理进程在代理进程侧有某些管理进程所关心的事件发生，如图 25-1所示。trap发送到管理进程的162号端口。

在图25-2中，我们已经描述了 trap PDU的格式。在下面关于 tcpdump输出内容中我们将再一次用到这些字段。

现在已经定义了6种特定的trap类型，第7种trap类型是由供应商自己定义的特定类型。图25-30给出了trap报文中trap类型字段的内容。

trap类型	名　称	描　述
0	coldStart	代理进程对自己初始化
1	warmStart	代理进程对自己重新初始化
2	linkDown	一个接口已经从工作状态改变为故障状态（图 25-18）。报文中的第一个变量标识此接口
3	linkUp	一个接口已经从故障状态改变为工作状态（图 25-18）。报文中的第一个变量标识此接口
4	authenticationFailure	从SNMP管理进程收到无效共同体的报文
5	egpNeighborLoss	一个EGP邻站已变为故障状态。报文中的第一个变量包含此邻站的IP地址
6	enterpriseSpecific	在这个特定的代码字段中查找trap信息

图25-30　trap的类型

用tcpdump命令来看看trap的情况。我们在系统sun上启动SNMP代理进程，然后让它产生coldStart类型的trap（我们告诉代理进程把trap信息发送到bsdi主机。虽然在该主机上并没有运行处理trap的管理进程，但是可以用tcpdump来查看产生了什么样的分组。回忆一下在图25-1中，trap是从代理进程发送到管理进程的，而管理进程不需要给代理进程发送确认。所以我们不需要trap的处理程序）。然后我们用snmpi程序发送一个请求，但该请求的共同体名称是无效的。这将产生一个authenticationFailure类型的trap。图25-31显示了命令的输出结果。

```
1  0.0              sun.snmp > bsdi.snmp-trap: C=traps Trap(28)
                    E:unix.1.2.5 [140.252.13.33] coldStart 20

2  18.86 (18.86)    sun.snmp > bsdi.snmp-trap: C=traps Trap(29)
                    E:unix.1.2.5 [140.252.13.33] authenticationFailure 1907
```

图25-31　tcpdump 输出的由SNMP代理进程产生的trap

首先注意一下两个 UDP数据报都是从 SNMP代理进程（端口是 161，图中显示的名称是 snmp）发送到目的端口号是162的服务器进程上的（图中显示的名称是 snmp-trap）。

再注意一下C=traps是trap报文的共同体名称。这是ISODE SNMP代理进程的配置选项。

下一个要注意的是：第1行中的Trap（28）和第2行中的Trap（29）是PDU类型和长度。

两个输出行的第一项内容都是相同的 E:unix.1.2.5。这代表企业名字段和代理进程的sysObjiectID。它是图25-6中1.3.6.1.4.1（iso.org.dod.internet.private.enterparses）结点下面的某个结点，所以代理进程的对象标识是1.3.6.1.4.1.4.1.2.5。它的简称是：unix.agents.fourBSD-isode.5。最后一个数字“5”代表ISODE代理进程软件的版本号。这些企业名的值代表了产生trap的代理进程软件信息。

输出的下一项是代理进程的IP地址（140.252.13.33）。

在第1行中，trap的类型显示的是coldStart，第2行中，显示的是authenticationFailure。与之相对应的trap类型值是0和4（如图25-30所示）。由于这些都不是厂家自定义的trap，所以特定代码必须是0，在图中没有显示。

输出的最后分别是20和1907，这是时间戳字段。它是TimeTicks类型的值，表示从代理进程初始化开始到trap发生所经历了多少个百分之一秒。在冷启动 trap的情况下，在代理进程初始化后到trap的产生共经历了200 ms。同样，tcpdump的输出表明第2个trap在第1个trap产生的18.86s后出现，这对应于打印出的1907个百分之一秒减去200 ms所得到的值。

图25-2表明trap报文还包含很多代理进程发送给管理进程的变量，但在这些在例子中没有再讨论。

25.11 ASN.1和BER

在正式的SNMP规范中都是采用ASN.1（Abstract Syntax Notation 1）语法，并且在SNMP报文中比特的编码采用BER（Basic Encoding Rule）。和其他介绍SNMP的书不同，我们有目的地把ASN.1和BER的讨论放到最后。因为如果放在前面讨论，有可能使读者产生混淆而忽略了SNMP的真正目的是进行网络管理。在这里我们也只是对这两个概念简单地进行解释，[Rose 1990]的第8章详细讨论了ASN.1和BER。

ASN.1是一种描述数据和数据特征的正式语言。它和数据的存储及编码无关。 MIB和SNMP报文中的所有的字段都是用 ASN.1描述的。例如：对于 SMI中的ipAddress数据类型，ASN.1是这样描述的：

```
IpAddress ::=
    [APPLICATION 0]              -- in network-byte order
        IMPLICIT OCTET STRING (SIZE (4))
```

同样，在MIB中，简单变量的定义是这样描述的：

```
udpNoPorts OBJECT-TYPE
    SYNTAX  Counter
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "The total number of received UDP datagrams for which
            there was no application at the destination port."
    ::= { udp 2 }
```

用SEQUENCE和SEQUENCE OF来定义表格的描述更加复杂。

当有了这样的ASN.1定义，可以有多种编码方法把数据编码为传输的比特流。 SNMP使用的编码方法是BER。例如，对于一个简单的整数如64，在BER中需要用3个字节来表示。第一个字节说明类型是一个整数，下个字节说明用了多少个字节来存储该整数（在这里是 1），最后一个字节才是该整数的值。

幸运的是，ASN.1和BER这两个繁琐的概念仅仅在实现SNMP的时候才重要，对我们理解

网络管理的概念和流程并没有太大的关系。

25.12 SNMPv2

在1993年，发表了定义新版本SNMP的11个RFC。RFC 1441 [Case et al. 1993]是其中的第一个，它系统地介绍了SNMPv2。同样，有两本书 [Stallings 1993; Rose 1994]也对SNMPv2进行了介绍。现在已经有两个SNMPv2的基本模型（参见附录 B.3中的[Rose 1994]），但是厂家的实现到1994年才能广泛使用。

在本节中，我们主要介绍SNMPv1和SNMPv2之间的重要区别。

1) 在SNMPv2中定义了一个新的分组类型`get-bulk-request`，它高效率地从代理进程读取大块数据。

2) 另的一个新的分组类型是`inform-request`，它使一个管理进程可以向另一个管理进程发送信息。

3) 定义了两个新的MIB，它们是：SNMPv2 MIB和SNMPv2-M2M MIB（管理进程到管理进程的MIB）。

4) SNMPv2的安全性比SNMPv1大有提高。在SNMPv1中，从管理进程到代理进程的共同体名称是以明文方式传送的。而SNMP v2可以提供鉴别和加密。

厂家提供的设备支持SNMPv2的会越来越多，管理站将对两个版本的 SNMP代理进程进行管理。[Routhier 1993]中描述了如何将SNMPv1的实现扩展到支持SNMPv2。

25.13 小结

SNMP是一种简单的、SNMP管理进程和SNMP代理进程之间的请求-应答协议。MIB定义了所有代理进程所包含的、能够被管理进程查询和设置的变量，这些变量的数据类型并不多。

所有这些变量都以对象标识符进行标识，这些对象标识符构成了一个层次命名结构，由一长串的数字组成，但通常缩写成人们阅读方便的简单名字。一个变量的特定实例可以用附加在这个对象标识符后面的一个实例来标识。

很多SNMP变量是以表格形式体现的。它们有固定的栏目，但有多少条记录并不固定。对于SNMP来讲，重要的是对表格中的每一行如何进行标识（尤其当我们不知道表格中有多少条记录时）以及如何按字典方式进行排序（“先列后行”的次序）。最后要说明的一点是：SNMP的`get-next`操作符对任何SNMP管理进程来讲都是最基本的操作。

然后我们介绍了下列的SNMP变量组：system、interface、address translation、IP、ICMP、TCP和UDP。接着是两个例子，一个介绍如何确定一个接口的MTU，另外一个介绍如何获取路由器的路由信息。

在本章的后面介绍了SNMP的trap操作，它是当代理进程发生了某些重大事件后主动向管理进程报告的。最后我们简单介绍了 ASN.1和BER，这两个概念比较繁琐，但所幸的是，它对我们了解SNMP并不十分重要，仅仅在实现SNMP的时候才要用到。

习题

25.1 我们说过采用两个不同的UDP端口（161和162）可以使得一个系统既可以是管理进程，也可以是代理进程。如果对管理进程和代理进程采用一个同样的端口，会出现什么情况？

25.2 用`get-next`操作，如何列出一张路由表的完整信息？