**net-snmp开发教程**

康林（[kl222@126.com](mailto:kl222@126.com)）

目录

[**1** **基本概念** 3](#_Toc326827827)

[**1.1** **snmp协议** 3](#_Toc326827828)

[**1.2** **MIBS** 3](#_Toc326827829)

[1.2.1 scalar：变量，就是一个量（包括整数理、字符串型、IPADDRESS等） 3](#_Toc326827830)

[1.2.2 table：表，一组变量 3](#_Toc326827831)

[1.2.3 notification：通知消息。以前叫trap 3](#_Toc326827832)

[**1.3** **安全认证** 4](#_Toc326827833)

[**2** **下载net-snmp代码。** 4](#_Toc326827834)

[**3** **编译代码：** 4](#_Toc326827835)

[3.1 查看编译配置 4](#_Toc326827836)

[3.2 开始编译： 4](#_Toc326827837)

[**4** **程序** 4](#_Toc326827838)

[4.1 snmpd:响应snmp请求包的代理服务器。 4](#_Toc326827839)

[4.2 snmptrapd:接收并记录snmp trap消息服务器。 5](#_Toc326827840)

[4.3 snmpset:设置请求 5](#_Toc326827841)

[4.4 snmpget、snmpgetnext、snmpwalk：得到请求 5](#_Toc326827842)

[4.5 snmptrap：trap产生工具 5](#_Toc326827843)

[4.6 snmptranslate:mib oid数字到文本互相转换工具 5](#_Toc326827844)

[4.7 snmpconf:配置工具 5](#_Toc326827845)

[4.8 net-snmp-config:返回编译和安装信息，建立SNMPV3配置 5](#_Toc326827846)

[4.9 net-snmp-create-v3-user:建立SNMPV3认证配置 5](#_Toc326827847)

[4.10 mib2c：用于从mib文件生成扩展模块c代码的工具。 5](#_Toc326827848)

[4.11 net-snmp-cert：用于认证信息产生 5](#_Toc326827849)

[4.12 命令程序验证常用参数： 5](#_Toc326827850)

[**5** **配置** 5](#_Toc326827851)

[5.1 snmp.conf:用于代理和应用程序的公共配置。 5](#_Toc326827852)

[5.1.1 输出样式 5](#_Toc326827853)

[5.1.2 默认认证选项 5](#_Toc326827854)

[5.1.3 调试输出选项 5](#_Toc326827855)

[5.1.4 文本mib解析 5](#_Toc326827856)

[5.2 snmpd.conf:用于配置snmpd代理 6](#_Toc326827857)

[5.2.1 代理操作模式 6](#_Toc326827858)

[5.2.2 系统安装信息 6](#_Toc326827859)

[5.2.3 安装存取控制 6](#_Toc326827860)

[5.2.4 trap目的地定义 6](#_Toc326827861)

[5.2.5 扩展代理 7](#_Toc326827862)

[5.2.5.1 用exec运行一个简单的命令 7](#_Toc326827863)

[5.2.5.2 interprets在一个实体类上的请求运行命令 7](#_Toc326827864)

[5.2.5.3 interprets一个实体类上的请求运行一个执久的进程 7](#_Toc326827865)

[5.2.5.4 代理一个外部代理请求 7](#_Toc326827866)

[5.2.5.5 用system运行一个简单的命令 7](#_Toc326827867)

[5.2.5.6 用共享库扩展代理 7](#_Toc326827868)

[5.2.6 运行主机各种监测量 7](#_Toc326827869)

[5.3 snmptrap.conf：用于配置snmptrapd 7](#_Toc326827870)

[5.3.1 日志选项 7](#_Toc326827871)

[5.3.2 trap外理者 7](#_Toc326827872)

[5.3.3 接收的traps输出格式 7](#_Toc326827873)

[5.3.4 认证选项 7](#_Toc326827874)

[5.3.5 运行选项 7](#_Toc326827875)

[5.4 用net-snmp-config配置V3存储控制 7](#_Toc326827876)

[**6** **启动** 8](#_Toc326827877)

[**7** **验证是否成功** 8](#_Toc326827878)

[**8** **开发** 8](#_Toc326827879)

[**8.1** **程序逻辑** 8](#_Toc326827880)

[8.1.1 接收网管发过来的snmp包，并对接收到的snmp包进行解析，校验后，找到并调用相应的处理函数进行处理。 8](#_Toc326827881)

[8.1.2 调用注册了的告警函数，向网管发送告警信息。 8](#_Toc326827882)

[**8.2** **代码结构** 10](#_Toc326827883)

[**8.3** **扩展agent** 10](#_Toc326827884)

[8.3.1 定义MIBS文件 10](#_Toc326827885)

[8.3.2 用mib2c工具转化MIBS为代码 10](#_Toc326827886)

[8.3.2.1 为mib2c工具配置mibs搜索路径：有两种方法 10](#_Toc326827887)

[8.3.3 生成代码 11](#_Toc326827888)

[8.3.4 修改代码 18](#_Toc326827889)

[8.3.5 编译 28](#_Toc326827890)

[8.3.6 加载 29](#_Toc326827891)

[8.3.6.3 子代理方式 30](#_Toc326827892)

[8.4 测试 30](#_Toc326827893)

[9 工具 30](#_Toc326827894)

[9.1 应用工具： 30](#_Toc326827895)

[9.2 MG-SOFT工具 31](#_Toc326827896)

1. **基本概念**
   1. **snmp协议**

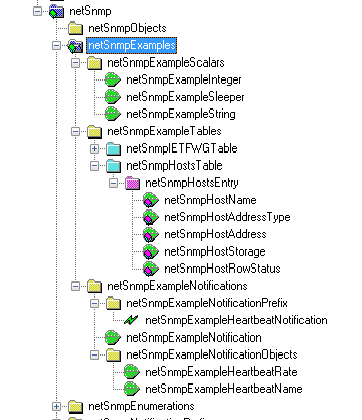
请参阅《TCP-IP详解卷1》——《第25章SNMP: 简单网络管理协议》

* 1. **MIBS**

MIBS rfc详见源码目录下的doc\rfc\smiV2

MIBS总共有三种实体：

* + 1. scalar：变量，就是一个量（包括整数理、字符串型、IPADDRESS等）
    2. table：表，一组变量
    3. notification：通知消息。以前叫trap



* 1. **安全认证**

snmp认证目前有三个版本：snmpv1、snmpv2、snmpv3

snmpv1（rfc1157）、snmpv2（rfc1901）都是基于密码字符串的认证，且密码为明文。

snmpv3（rfc2571）认证分为使用USM（基于用户的安全模式）和VACM（基于查看的访问控制模式）。认证的加协议有：MD5、SHA；加密协议有：DES、AES；加密协议用于加密PDU数据。

1. **下载net-snmp代码。**
2. **编译代码：**
   1. 查看编译配置

*./configure –help* #查看配置

* 1. 开始编译：

*./configure*

*make*

*make install*

1. **程序**
   1. snmpd:响应snmp请求包的代理服务器。

用法：-f 不调用fork,即不进入daemon，如果不用这个参数，则进入daemon模式

-L[efos]日志输出到指定的位置

-Le Log messages to the standard error stream.

-Lf FILE

Log messages to the specified file.

-Lo Log messages to the standard output stream.

-Ls FACILITY

Log messages via syslog, using the specified facility

('d' for LOG\_DAEMON, 'u' for LOG\_USER, or '0'-'7' for

LOG\_LOCAL0 through LOG\_LOCAL7).

* 1. snmptrapd:接收并记录snmp trap消息服务器。
  2. snmpset:设置请求
  3. snmpget、snmpgetnext、snmpwalk：得到请求
  4. snmptrap：trap产生工具
  5. snmptranslate:mib oid数字到文本互相转换工具
  6. snmpconf:配置工具
  7. net-snmp-config:返回编译和安装信息，建立SNMPV3配置
  8. net-snmp-create-v3-user:建立SNMPV3认证配置
  9. mib2c：用于从mib文件生成扩展模块c代码的工具。
  10. net-snmp-cert：用于认证信息产生
  11. 命令程序验证常用参数：

-v 1|2c|3 指定SNMP使用的版本

-V, --version 显示命令的版本号

SNMP Version 1 or 2c specific

-c COMMUNITY 设置团体名

SNMP Version 3 specific

-a PROTOCOL 设置认证协议 (MD5|SHA)

-A PASSPHRASE 设置认证协议密码

-e ENGINE-ID set security engine ID (e.g. 800000020109840301)

-E ENGINE-ID set context engine ID (e.g. 800000020109840301)

-l LEVEL set security level (noAuthNoPriv|authNoPriv|authPriv)

-n CONTEXT set context name (e.g. bridge1)

-u USER-NAME set security name (e.g. bert)

-x PROTOCOL 设置加密协议 (DES|AES)

-X PASSPHRASE 设置加密协议密码

-Z BOOTS,TIME set destination engine boots/time

例子：

snmpgetnext -v 1 -c public 127.0.0.1 sysUpTime

snmpget –v 3 –a MD5 –A 1235678 –u name 127.0.0.1 oid

1. **配置**

用配置工具 snmpconf 进行配置

配置包括三个部分：

* 1. snmp.conf:用于代理和应用程序的公共配置。
     1. 输出样式
     2. 默认认证选项
     3. 调试输出选项
     4. 文本mib解析

###########################################################################

# SECTION: Textual mib parsing

#

# This section controls the textual mib parser. Textual

# mibs are parsed in order to convert OIDs, enumerated

# lists, and ... to and from textual representations

# and numerical representations.

# mibs: Specifies a list of mibs to be searched for and loaded.

# Adding a '+' sign to the front of the argument appends the new

# mib name to the list of mibs already being searched for.

# arguments: [+]mibname[:mibname...]

mibs +/root/test/NET-SNMP-EXAMPLES-MIB.txt

* 1. snmpd.conf:用于配置snmpd代理
     1. 代理操作模式
     2. 系统安装信息
     3. 安装存取控制

###########################################################################

# SECTION: Access Control Setup

#

# This section defines who is allowed to talk to your running

# snmp agent.

# rwcommunity: a SNMPv1/SNMPv2c read-write access community name

# arguments: community [default|hostname|network/bits] [oid]

rwcommunity public 127.0.0.1

rwcommunity public 192.168.0.1

# rwuser: a SNMPv3 read-write user

# arguments: user [noauth|auth|priv] [restriction\_oid]

rwuser public #这里定义用户名和验证方式，密码保存在/var/net-snmp/snmpd.conf中

# rouser: a SNMPv3 read-only user

# arguments: user [noauth|auth|priv] [restriction\_oid]

rouser public #这里定义用户名和验证方式，密码保存在/var/net-snmp/snmpd.conf中,详见——[用net-snmp-config配置V3](#V3存取控制)

* + 1. trap目的地定义

###########################################################################

# SECTION: Trap Destinations

#

# Here we define who the agent will send traps to.

# trap2sink: A SNMPv2c trap receiver

# arguments: host [community] [portnum]

trap2sink 127.0.0.1 public 162

trap2sink 192.168.0.1 public 162

* + 1. 扩展代理
       1. 用exec运行一个简单的命令
       2. interprets在一个实体类上的请求运行命令
       3. interprets一个实体类上的请求运行一个执久的进程
       4. 代理一个外部代理请求
       5. 用system运行一个简单的命令
       6. 用共享库扩展代理

###########################################################################

# SECTION: Extending the Agent

#

# You can extend the snmp agent to have it return information

# that you yourself define.

# dlmod: dynamically extend the agent using a shared-object

# arguments: module-name module-path

dlmod notification /data/snmp/test/.libs/ libnetsnmpmibs.so

* + 1. 运行主机各种监测量
  1. snmptrap.conf：用于配置snmptrapd
     1. 日志选项
     2. trap外理者
     3. 接收的traps输出格式
     4. 认证选项

#authCommunity TYPES COMMUNITY [SOURCE [OID | -v VIEW ]]

authCommunity log,execute,net public

* + 1. 运行选项
  1. 用net-snmp-config配置V3存储控制

root@ubuntu:~# net-snmp-config --create-snmpv3-user --help

Usage:

net-snmp-create-v3-user [-ro] [-A authpass] [-X privpass]

[-a MD5|SHA] [-x DES|AES] [username]

root@ubuntu:~# net-snmp-create-v3-user -ro -A SHA -a public1234 -x DES -X abcdefgmok public

adding the following line to /var/net-snmp/snmpd.conf:

createUser public SHA "public1234" DES abcdefgmok

adding the following line to /usr/local/share/snmp/snmpd.conf:

rouser public

net-snmp-create-v3-user实质上是个脚本文件

它实质上是把V3的用户配置（createUser public SHA "public1234" DES abcdefgmok）加到/var/net-snmp/snmpd.conf文件中。然后agent从这个文件中读取v3的用户配置信息。然后生成相应的密码并保存在这个文件中。

读写权限设置是在$SNMPCONF/snmpd.conf中配置的，详见——[5.2 snmpd.conf:用于配置snmpd代理](#存取控制)

SNMP Version 3 参数：

-a PROTOCOL 设置认证协议 (MD5|SHA)

-A PASSPHRASE 设置认证协议密码

-x PROTOCOL 设置加密协议 (DES|AES)

-X privpass 设置加密协议密码

1. **启动**

执行 *snmpd –f –Lo*

后台方式 *snmpd*

1. **验证是否成功**

root@ubuntu:~# snmpget -v 1 -c public 127.0.0.1 UDP-MIB::udpInDatagrams.0

UDP-MIB::udpInDatagrams.0 = Counter32: 7822

如果出现错误，请检查配置是否正确。

1. **开发**
   1. **程序逻辑**

snmpd代理完成两个功能：

* + 1. 接收网管发过来的snmp包，并对接收到的snmp包进行解析，校验后，找到并调用相应的处理函数进行处理。
    2. 调用注册了的告警函数，向网管发送告警信息。



下面是处理snmp包的堆栈：

#0 reachalarmSubscriberTable\_handler (handler=0x8159980, reginfo=0x8122b78,

reqinfo=0x817d050, requests=0x817d138) at reachalarmSubscriberTable.c:116

#1 0x008e08e6 in netsnmp\_call\_handler (requests=0x817d138, reqinfo=0x817d050,

reginfo=0x8122b78, next\_handler=0x8159980) at agent\_handler.c:526

#2 netsnmp\_call\_next\_handler (current=0x8146088, reginfo=0x8122b78,

reqinfo=0x817d050, requests=0x817d138) at agent\_handler.c:640

#3 0x008dab83 in netsnmp\_table\_iterator\_helper\_handler (handler=0x8146088,

reginfo=0x8122b78, reqinfo=0x817d050, requests=0x817d138)

at helpers/table\_iterator.c:936

#4 0x008e08e6 in netsnmp\_call\_handler (requests=0x817d138, reqinfo=0x817d050,

reginfo=0x8122b78, next\_handler=0x8146088) at agent\_handler.c:526

#5 netsnmp\_call\_next\_handler (current=0x813dcf0, reginfo=0x8122b78,

reqinfo=0x817d050, requests=0x817d138) at agent\_handler.c:640

#6 0x008d268c in table\_helper\_handler (handler=0x813dcf0, reginfo=0x8122b78,

reqinfo=0x817d050, requests=0x817d138) at helpers/table.c:712

#7 0x008e0291 in netsnmp\_call\_handler (requests=0x817d138, reqinfo=0x817d050,

reginfo=0x8122b78, next\_handler=0x813dcf0) at agent\_handler.c:526

#8 netsnmp\_call\_handlers (reginfo=0x8122b78, reqinfo=0x817d050,

requests=0x817d138) at agent\_handler.c:611

#9 0x008efc4a in handle\_var\_requests (asp=0x817c738) at snmp\_agent.c:2676

#10 0x008f0150 in handle\_getnext\_loop (asp=0x817c738) at snmp\_agent.c:3122

#11 0x008f0b0b in handle\_pdu (asp=0x817c738) at snmp\_agent.c:3499

#12 0x008f0c98 in netsnmp\_handle\_request (asp=0x817c738, status=0)

at snmp\_agent.c:3278

#13 0x008f16bc in handle\_snmp\_packet (magic=0x0, pdu=0x8191178,

session=0x817c898, op=<optimized out>, reqid=<optimized out>)

at snmp\_agent.c:1987

#14 handle\_snmp\_packet (op=1, session=0x817c898, reqid=8, pdu=0x8191178,

magic=0x0) at snmp\_agent.c:1889

#15 0x003808a6 in \_sess\_process\_packet (sessp=0x817c6c0, sp=0x817c898,

isp=0x817c860, transport=0x817c6d8, opaque=0x817d618, olength=36,

packetptr=0x8180ba0 "0&\002\001", length=40) at snmp\_api.c:5400

#16 0x00381d0b in \_sess\_read (sessp=0x817c6c0, fdset=0xbfd00128)

at snmp\_api.c:5829

#17 0x0038250a in snmp\_sess\_read2 (sessp=0x817c6c0, fdset=0xbfd00128)

at snmp\_api.c:5861

#18 0x003825f4 in snmp\_read2 (fdset=0xbfd00128) at snmp\_api.c:5463

#19 0x0804bcee in receive () at snmpd.c:1315

#20 main (argc=6, argv=0xbfd00414) at snmpd.c:1105

* 1. **代码结构**

net-snmp可以进行对snmp包处理程序和告警进行扩展。

示例程序存放在源码目录：agent\mibgroup\examples 下。

示例MIBS文件：

mibs\NET-SNMP-MIB.txt

mibs\NET-SNMP-EXAMPLES-MIB.txt

* 1. **扩展agent**
     1. 定义MIBS文件

用mg-soft builder生成。

mibs\NET-SNMP-EXAMPLES-MIB.txt

* + 1. 用mib2c工具转化MIBS为代码
       1. 为mib2c工具配置mibs搜索路径：有两种方法
          1. 配置环境变量：

export MIBS=ALL

并把MIBS文件复制到net-snmp安装目录的中的mibs目录下

* + - * 1. 在配置文件snmp.conf中加入

MIBS ALL #搜索/usr/share/snmp/mibs下的所有文件

MIBS +/usr/share/snmp/ NET-SNMP-EXAMPLES-MIB.txt #加入新建立的文件

* + - * 1. 验证是否能加载刚加入的文件

root@ubuntu:~# snmptranslate -TB netSnmpExamples

NET-SNMP-EXAMPLES-MIB::netSnmpExamples

NET-SNMP-EXAMPLES-MIB::netSnmpExampleScalars

NET-SNMP-EXAMPLES-MIB::netSnmpExampleString

NET-SNMP-EXAMPLES-MIB::netSnmpExampleSleeper

解析正确，就说明MIBS文件配置正确。

* + 1. 生成代码
       1. 直接在你想要生成代码的目录下执行下面命令：

mib2c实体名

mib2c会提示你进行操作

root@ubuntu:~#mib2c netSnmpExamples

writing to -

mib2c has multiple configuration files depending on the type of

code you need to write. You must pick one depending on your need.

You requested mib2c to be run on the following part of the MIB tree:

OID: netSnmpExamples

numeric translation: .1.3.6.1.4.1.8072.2

number of scalars within: 8

number of tables within: 3

number of notifications within: 1

First, do you want to generate code that is compatible with the

ucd-snmp 4.X line of code, or code for the newer Net-SNMP 5.X code

base (which provides a much greater choice of APIs to pick from):

1) ucd-snmp style code

2) Net-SNMP style code

Select your choice :

建议做法是：为每一个实体单独生成一组文件。这样条理比较清楚。

* + - 1. 如果你清楚模块，可以用下面命令直接指定模板。

*mib2c –c 模板文件 实体名*

模板文件位于: /usr/share/snmp/\*.conf

实体名：本列中是：netSnmpExamples

注意：模块名不要与已存的的模块名重复，否则会出错（没有提示，这就看mib2c先搜索到哪个MIBS，就应用哪个）。且模块名是大小写敏感觉的。

执行完后，会生成一组列文件。

* + - 1. scalar实体：

root@ubuntu:~/test/mib# mib2c netSnmpExampleInteger

writing to -

mib2c has multiple configuration files depending on the type of

code you need to write. You must pick one depending on your need.

You requested mib2c to be run on the following part of the MIB tree:

OID: netSnmpExampleInteger

numeric translation: .1.3.6.1.4.1.8072.2.1.1

number of scalars within: 1

number of tables within: 0

number of notifications within: 0

First, do you want to generate code that is compatible with the

ucd-snmp 4.X line of code, or code for the newer Net-SNMP 5.X code

base (which provides a much greater choice of APIs to pick from):

1) ucd-snmp style code

2) Net-SNMP style code

Select your choice : 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GENERATING CODE FOR SCALAR OBJECTS:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

It looks like you have some scalars in the mib you requested, so I

will now generate code for them if you wish. You have two choices

for scalar API styles currently. Pick between them, or choose not

to generate any code for the scalars:

1) If you're writing code for some generic scalars

(by hand use: "mib2c -c mib2c.scalar.conf netSnmpExampleInteger")

2) If you want to magically "tie" integer variables to integer

scalars

(by hand use: "mib2c -c mib2c.int\_watch.conf netSnmpExampleInteger")

3) Don't generate any code for the scalars

Select your choice: 1

using the mib2c.scalar.conf configuration file to generate your code.

writing to netSnmpExampleInteger.h

writing to netSnmpExampleInteger.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* NOTE WELL: The code generated by mib2c is only a template. \*YOU\* \*

\* must fill in the code before it'll work most of the time. In many \*

\* cases, spots that MUST be edited within the files are marked with \*

\* /\* XXX \*/ or /\* TODO \*/ comments. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

running indent on netSnmpExampleInteger.h

running indent on netSnmpExampleInteger.c

* + - 1. table实体：

root@ubuntu:~/test/mib# mib2c netSnmpHostsTable

writing to -

mib2c has multiple configuration files depending on the type of

code you need to write. You must pick one depending on your need.

You requested mib2c to be run on the following part of the MIB tree:

OID: netSnmpHostsTable

numeric translation: .1.3.6.1.4.1.8072.2.2.2

number of scalars within: 0

number of tables within: 1

number of notifications within: 0

First, do you want to generate code that is compatible with the

ucd-snmp 4.X line of code, or code for the newer Net-SNMP 5.X code

base (which provides a much greater choice of APIs to pick from):

1) ucd-snmp style code

2) Net-SNMP style code

Select your choice : 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GENERATING CODE FOR TABLES:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The Net-SNMP agent API is extremely extensive and, in fact, lets

each programmer write agent code according to the style that works

best for them based on their experience and their preference. We're

going to ask you a serious of questions that will help mib2c

generate code that best suits \*your\* needs, as the programmer that

will be responsible for taking the code and further refining it. If

you don't like how the results look, you are always welcome to

re-run mib2c and select a different set of options.

There are essentially two tasks involved in processing requests

for OIDs within a MIB table - firstly identifying the relevant row

of the table for a given request, and then returning (or updating)

the appropriate column value within that row. Many MIB tables model

the state of some external system (the kernel, a device, processes,

etc), and the MIB implementation module (the code we're about to

produce a template for) acts as an interface between this underlying

system and the SNMP side. Other tables hold data internally that is

only available within the agent itself, or at least the master copy

of the data is held within the agent.

There are a number of different code templates that can be used to

implement MIB tables, using various approaches to these two tasks.

There are three basic approaches to identifying the relevant row:

1) Pass the request through to the table-specific code, and

identify the requested row there (for both GET and GETNEXT

requests). This is typically the most efficient way to get

up-to-date information, but relies on suitable

(programmer-provided) code within the MIB handler.

Most importantly, you should be an expert to use this choice.

This will produce code based on the table\_dataset handler.

2) Have table-specific code to provide information about which

rows exist in the table (by iterating through them in turn),

but utilise standard helper code to select the appropriate

row for a given request. This is particularly suitable for

tables where the data is naturally stored in a "random" order

(or differently to the MIB table index), or where rows are

frequently added to or removed from the table.

However searching for the requested row is not very efficient,

and performance can be slow - particularly for large tables with

many rows.

3) Hold a locally cached copy of the contents of the table (or at

least a cache of which rows are valid), and utilise standard

helper code to select the appropriate row. This is

significantly faster than the iterator-based approach, but

cached data is inevitably slightly "stale" with respect to the

data from the underlying system being managed. This approach,

since it relies on caching of data, is also results in a larger

memory footprint. It is less appropriate for tables where rows

are frequently added or removed externally to the agent (i.e.,

not via SNMP requests).

This approach can also be used where \_all\_ use of the table is

via SNMP, and there is no external "underlying system". In

this case, the local cache is the canonical version of the

table.

4) Do not generate code for the tables.

Select the option that best fits your requirements: 2

Having identified the appropriate row for a given request, there are

three basic styles of code for returning (or updating) the requested

column value from within this row:

1) A single handler routine, which contains all the code needed to

handle GET and SET requests for each of the column objects.

The code typically looks like a single function with a large 'case'

statement covering each of the columns.

This will produce code based on the 'iterator' hepler.

2) A set of individual routines, each of which is concerned

with a particular aspect of processing the request.

Each column object within the table has one routine for

retrieving the current value, and another for setting a new one.

This will produce code based on the 'iterate\_access' hepler.

3) A (different) set of individual routines, each of which is

smaller and more tightly focused than the code generated by

style 2. The aim here is to reduce the amount of SNMP specific

knowledge required to implement a module, and hide much of the

SNMP terminology and processing within standard generated code

(which can simply be used sight unseen).

This will produce code based on the 'mfd' hepler ('MIB for Dummies').

4) Do not generate code for the tables.

(In all cases, GETNEXT requests are automatically converted

into the equivalent GET request, so the MIB specific code

need only be concerned with GET and SET requests.).

Select the code style you wish to use: 1

The same template code can be generated using

mib2c -c mib2c.iterate.conf netSnmpHostsTable

This framework can work in one of two ways:

1) Hold a local cached copy of some external data

which is then used to service incoming requests.

2) Query the external data directly for each request.

The first is typically more efficient, but results in

slightly "stale" data (depending on the expiration timeout

for the cache) and greater memory usage. The second can

provide more up-to-date information, but at the cost of

higher processing overheads.

Which is more appropriate for your needs?

Select your choice : 1

writing to netSnmpHostsTable.h

writing to netSnmpHostsTable.c

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* NOTE WELL: The code generated by mib2c is only a template. \*YOU\* \*

\* must fill in the code before it'll work most of the time. In many \*

\* cases, spots that MUST be edited within the files are marked with \*

\* /\* XXX \*/ or /\* TODO \*/ comments. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

running indent on netSnmpHostsTable.h

running indent on netSnmpHostsTable.c

* + - 1. notification实体：

root@ubuntu:~/test/mib# mib2c netSnmpExampleHeartbeatNotification

writing to -

mib2c has multiple configuration files depending on the type of

code you need to write. You must pick one depending on your need.

You requested mib2c to be run on the following part of the MIB tree:

OID: netSnmpExampleHeartbeatNotification

numeric translation: .1.3.6.1.4.1.8072.2.3.0.1

number of scalars within: 0

number of tables within: 0

number of notifications within: 1

First, do you want to generate code that is compatible with the

ucd-snmp 4.X line of code, or code for the newer Net-SNMP 5.X code

base (which provides a much greater choice of APIs to pick from):

1) ucd-snmp style code

2) Net-SNMP style code

Select your choice : 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GENERATING CODE FOR NOTIFICATIONS:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Would you like to generate code for sending notifications from within

the agent?

"y" or "n": y

using mib2c.notify.conf to generate code for sending notifications

writing to netSnmpExampleHeartbeatNotification.h

writing to netSnmpExampleHeartbeatNotification.c

# GENERATING HEADER FILE DEFINITIONS

#

# To generate just a header with a define for each column number in

# your table:

#

# mib2c -c mib2c.column\_defines.conf netSnmpExampleHeartbeatNotification

#

# To generate just a header with a define for each enum for any

# column containing enums:

#

# mib2c -c mib2c.column\_enums.conf netSnmpExampleHeartbeatNotification

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* NOTE WELL: The code generated by mib2c is only a template. \*YOU\* \*

\* must fill in the code before it'll work most of the time. In many \*

\* cases, spots that MUST be edited within the files are marked with \*

\* /\* XXX \*/ or /\* TODO \*/ comments. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

running indent on netSnmpExampleHeartbeatNotification.c

running indent on netSnmpExampleHeartbeatNotification.h

* + 1. 修改代码
       1. scalar实体和table实体代码：

scalar实体和table实体其实是一类代码，只是处理的复杂度不一样。

代码的核心，就是初始化函数时（程序初始化加载或动态库加载），向代理(snmpd)注册了回调处理函数，当(snmpd)接收到一个snmp请求包时，它会先对包进行校验，如果校验不通过，会返回相应的错误。如果通过后，它会解析请求包，并把请求包的内容转换成请求结构（netsnmp\_agent\_request\_info【包含请求包的pdu信息】，netsnmp\_request\_info【包含请求包的vb信息】）。匹配到相关的oid时，就调用相应的注册处理函数并传入请求结构给处理函数，处理函数只需要根据结构中的内容进行相应的业务处理就可以了。

/\*\* Initializes the netSnmpExampleInteger module \*/

void

init\_netSnmpExampleInteger(void)

{

const oid netSnmpExampleInteger\_oid[] = { 1,3,6,1,4,1,8072,2,1,1 };

DEBUGMSGTL(("netSnmpExampleInteger", "Initializing\n"));

netsnmp\_register\_scalar(

netsnmp\_create\_handler\_registration("netSnmpExampleInteger", handle\_netSn

mpExampleInteger,

netSnmpExampleInteger\_oid, OID\_LENGTH(netSnmpExamp

leInteger\_oid),

HANDLER\_CAN\_RWRITE

));

}

初始函数调用netsnmp\_create\_handler\_registration向snmpd注册netSnmpExampleInteger\_oid的处理函数 handle\_netSnmpExampleInteger。

函数定义：

int

handle\_netSnmpExampleInteger(netsnmp\_mib\_handler \*handler,

netsnmp\_handler\_registration \*reginfo,

netsnmp\_agent\_request\_info \*reqinfo,

netsnmp\_request\_info \*requests)

当snmpd调用它时，便会把相应的snmp包的信息从这几个参数中传进来。其中reqinfo 就是snmp请求包被snmpd解析后得到的结构，包含了请求包的会话和pdu信息；requests: 主要包含了 VB 信息。

/\*\* @struct netsnmp\_agent\_request\_info\_s

\* The agent transaction request structure

\*/

typedef struct netsnmp\_agent\_request\_info\_s {

int mode;

/\*\* pdu contains authinfo, eg \*/

/\* netsnmp\_pdu \*pdu; \*/

struct netsnmp\_agent\_session\_s \*asp; /\* may not be needed \*/

/\*

\* can be used to pass information on a per-pdu basis from a

\* helper to the later handlers

\*/

netsnmp\_data\_list \*agent\_data;

} netsnmp\_agent\_request\_info;

typedef struct netsnmp\_agent\_session\_s {

int mode;

netsnmp\_session \*session;

netsnmp\_pdu \*pdu;

netsnmp\_pdu \*orig\_pdu;

int rw;

int exact;

int status;

int index;

int oldmode;

struct netsnmp\_agent\_session\_s \*next;

/\*

\* new API pointers

\*/

netsnmp\_agent\_request\_info \*reqinfo;

netsnmp\_request\_info \*requests;

netsnmp\_tree\_cache \*treecache;

netsnmp\_variable\_list \*\*bulkcache;

int treecache\_len; /\* length of cache array \*/

int treecache\_num; /\* number of current cache entries \*/

netsnmp\_cachemap \*cache\_store;

int vbcount;

} netsnmp\_agent\_session;

typedef struct netsnmp\_request\_info\_s {

/\*\*

\* variable bindings

\*/

netsnmp\_variable\_list \*requestvb;

/\*\*

\* can be used to pass information on a per-request basis from a

\* helper to the later handlers

\*/

netsnmp\_data\_list \*parent\_data;

/\*

\* pointer to the agent\_request\_info for this request

\*/

struct netsnmp\_agent\_request\_info\_s \*agent\_req\_info;

/\*\* don't free, reference to (struct tree)->end \*/

oid \*range\_end;

size\_t range\_end\_len;

/\*

\* flags

\*/

int delegated;

int processed;

int inclusive;

int status;

/\*\* index in original pdu \*/

int index;

/\*\* get-bulk \*/

int repeat;

int orig\_repeat;

netsnmp\_variable\_list \*requestvb\_start;

/\* internal use \*/

struct netsnmp\_request\_info\_s \*next;

struct netsnmp\_request\_info\_s \*prev;

struct netsnmp\_subtree\_s \*subtree;

} netsnmp\_request\_info;

每次处理一个SNMP请求，就调用一次注册函数。从模板生成的文件，可以很容易看出：通过requests遍历vb。致于如何处理，你可以用模板生成的方法做，也可以按自己的想法做，你可以把这个看成应用程序的main函数就好了：）

* + - 1. notification实体代码

例子可以看net-snmp自带的示例程序。源码目录\agent\mibgroup\examples\notification.c

send\_v2trap:已封装好了，按snmp v2 trap发送。

send\_easy\_trap：按snmpv1 trap发送。

以上两种发送，都是通过 netsnmp\_send\_traps进行封装发送的。

/\*

\* send\_trap\_to\_sess: sends a trap to a session but assumes that the

\* pdu is constructed correctly for the session type.

\*/

void

send\_trap\_to\_sess(netsnmp\_session \* sess, netsnmp\_pdu \*template\_pdu)

最终调用：

/\*

\* These functions send PDUs using an active session:

\* snmp\_send - traditional API, no callback

\* snmp\_async\_send - traditional API, with callback

\* snmp\_sess\_send - single session API, no callback

\* snmp\_sess\_async\_send - single session API, with callback

\*

\* Call snmp\_build to create a serialized packet (the pdu).

\* If necessary, set some of the pdu data from the

\* session defaults.

\* If there is an expected response for this PDU,

\* queue a corresponding request on the list

\* of outstanding requests for this session,

\* and store the callback vectors in the request.

\*

\* Send the pdu to the target identified by this session.

\* Return on success:

\* The request id of the pdu is returned, and the pdu is freed.

\* Return on failure:

\* Zero (0) is returned.

\* The caller must call snmp\_free\_pdu if 0 is returned.

\*/

int

snmp\_send(netsnmp\_session \* session, netsnmp\_pdu \*pdu)

自己定义发送，可以参考netsnmp\_send\_traps的实现。

设置trap目标地址：

int create\_trap\_session(char \*sink, u\_short sinkport,

char \*com, int version, int pdutype)

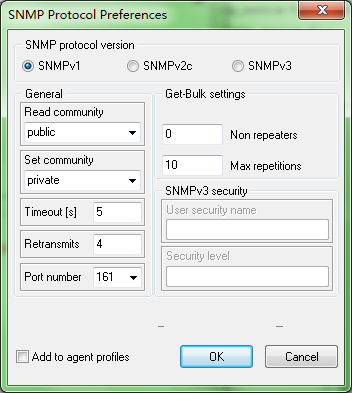
int add\_trap\_session(netsnmp\_session \*, int, int, int);//如果是通告的话，就会调用notifyTable\_register\_notifications 把新的会话目的地址加入，但在remove\_trap\_session中没有对其进行删除。

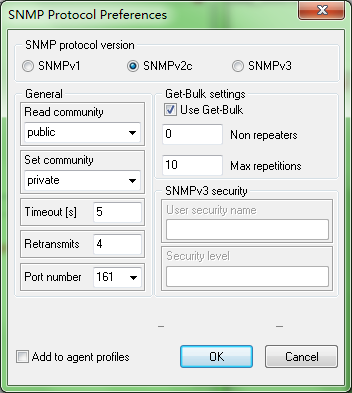
int remove\_trap\_session(netsnmp\_session \*);//这个没有实现通告目的地址删除功能

sink:就是目标地址IP

sinkport：就是目标地址端口。默认是162

com：团体名（community），用于安全认证的字符。





对于v1、v2来说，这个字段就是community

struct snmp\_session {

/\*

\* SNMPv1 & SNMPv2c fields

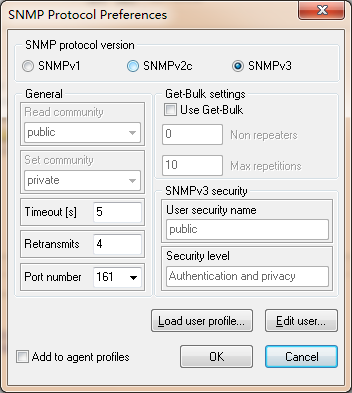
\*/

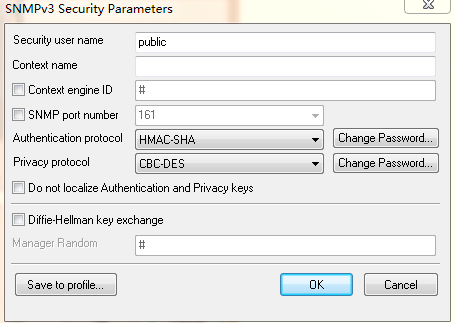
/\*\* community for outgoing requests. \*/

u\_char \*community;

……

};





struct snmp\_session {

/\*

\* SNMPv3 fields

\*/

/\*\* are we the authoritative engine? \*/

u\_char isAuthoritative;

/\*\* authoritative snmpEngineID \*/

u\_char \*contextEngineID;

/\*\* Length of contextEngineID \*/

size\_t contextEngineIDLen;

/\*\* initial engineBoots for remote engine \*/

u\_int engineBoots;

/\*\* initial engineTime for remote engine \*/

u\_int engineTime;

/\*\* authoritative contextName \*/

char \*contextName;

/\*\* Length of contextName \*/

size\_t contextNameLen;

/\*\* authoritative snmpEngineID \*/

u\_char \*securityEngineID;

/\*\* Length of contextEngineID \*/

size\_t securityEngineIDLen;

/\*\* on behalf of this principal \*/

char \*securityName;

/\*\* Length of securityName. \*/

size\_t securityNameLen;

/\*\* auth protocol oid \*/

oid \*securityAuthProto;

/\*\* Length of auth protocol oid \*/

size\_t securityAuthProtoLen;

/\*\* Ku for auth protocol XXX \*/

u\_char securityAuthKey[USM\_AUTH\_KU\_LEN];

/\*\* Length of Ku for auth protocol \*/

size\_t securityAuthKeyLen;

/\*\* Kul for auth protocol \*/

u\_char \*securityAuthLocalKey;

/\*\* Length of Kul for auth protocol XXX \*/

size\_t securityAuthLocalKeyLen;

/\*\* priv protocol oid \*/

oid \*securityPrivProto;

/\*\* Length of priv protocol oid \*/

size\_t securityPrivProtoLen;

/\*\* Ku for privacy protocol XXX \*/

u\_char securityPrivKey[USM\_PRIV\_KU\_LEN];

/\*\* Length of Ku for priv protocol \*/

size\_t securityPrivKeyLen;

/\*\* Kul for priv protocol \*/

u\_char \*securityPrivLocalKey;

/\*\* Length of Kul for priv protocol XXX \*/

size\_t securityPrivLocalKeyLen;

/\*\* snmp security model, v1, v2c, usm \*/

int securityModel;

/\*\* noAuthNoPriv, authNoPriv, authPriv \*/

int securityLevel;

/\*\* target param name \*/

char \*paramName;

/\*\*

\* security module specific

\*/

void \*securityInfo;

}

version:

/\*

\* SNMP versions

\*/

/\*

\* There currently exists the following SNMP versions.

\* \* (Note that only SNMPv1 is in widespread usage, and this code supports

\* \* only SNMPv1, SNMPv2c, and SNMPv3.

\* \*

\* \* SNMPv1 - (full) the original version, defined by RFC 1157

\* \* SNMPsec - (historic) the first attempt to add strong security

\* \* to SNMPv1, defined by RFCs 1351, 1352, and 1353.

\* \* SNMPv2p - (historic) party-based SNMP, which was another

\* \* attempt to add strong security to SNMP, defined

\* \* by RFCs 1441, 1445, 1446, 1448, and 1449.

\* \* SNMPv2c - (experimental) community string-based SNMPv2,

\* \* which was an attempt to combine the protocol

\* \* operations of SNMPv2 with the security of

\* \* SNMPv1, defined by RFCs 1901, 1905, and 1906.

\* \* SNMPv2u - (experimental) user-based SNMPv2, which provided

\* \* security based on user names and protocol

\* \* operations of SNMPv2, defined by RFCs 1905,

\* \* 1909, and 1910.

\* \* SNMPv2\* (or SNMPv2star) - (experimental) an attempt to add the

\* \* best features of SNMPv2p and SNMPv2u, defined

\* \* by unpublished documents found at WEB site

\* \* owned by SNMP Research (a leading SNMP vendor)

\* \* SNMPv3 - the current attempt by the IETF working group to merge

\* \* the SNMPv2u and SNMPv2\* proposals into a more widly

\* \* accepted SNMPv3. It is defined by not yet published

\* \* documents of the IETF SNMPv3 WG.

\* \*

\* \* SNMPv1, SNMPv2c, SNMPv2u, and SNMPv3 messages have a common

\* \* form, which is an ASN.1 sequence containing a message version

\* \* field, followed by version dependent fields.

\* \* SNMPsec, SNMPv2p, and SNMPv2\* messages have a common form,

\* \* which is a tagged ASN.1 context specific sequence containing

\* \* message dependent fields.

\* \*

\* \* In the #defines for the message versions below, the value

\* \* for SNMPv1, SNMPv2c, SNMPv2u, and SNMPv3 messages is the

\* \* value of the message version field. Since SNMPsec, SNMPv2p,

\* \* and SNMPv2\* messages do not have a message version field,

\* \* the value in the defines for them is choosen to be a large

\* \* arbitrary number.

\* \*

\* \* Note that many of the version ID's are defined below purely for

\* \* documentational purposes. At this point the only protocol planned

\* \* for future implementations is SNMP3, as the other v2 protocols will

\* \* not be supported by the IETF (ie, v2u, v2sec, v2star) or used by

\* \* the snmp community at large (at the time of this writing).

\*/

/\*

\* versions based on version field

\*/

#ifndef NETSNMP\_DISABLE\_SNMPV1

#define SNMP\_VERSION\_1 0

#endif

#ifndef NETSNMP\_DISABLE\_SNMPV2C

#define SNMP\_VERSION\_2c 1

#endif

#define SNMP\_VERSION\_2u 2 /\* not (will never be) supported by this code \*/

#define SNMP\_VERSION\_3 3

/\*

\* versions not based on a version field

\*/

#define SNMP\_VERSION\_sec 128 /\* not (will never be) supported by this code \*/

#define SNMP\_VERSION\_2p 129 /\* no longer supported by this code (> 4.0) \*/

#define SNMP\_VERSION\_2star 130 /\* not (will never be) supported by this code \*/

pdutype:

/\*

\* PDU types in SNMPv1, SNMPsec, SNMPv2p, SNMPv2c, SNMPv2u, SNMPv2\*, and SNMPv3

\*/

#define SNMP\_MSG\_GET (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x0) /\* a0=160 \*/

#define SNMP\_MSG\_GETNEXT (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x1) /\* a1=161 \*/

#define SNMP\_MSG\_RESPONSE (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x2) /\* a2=162 \*/

#ifndef NETSNMP\_NO\_WRITE\_SUPPORT

#define SNMP\_MSG\_SET (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x3) /\* a3=163 \*/

#endif /\* !NETSNMP\_NO\_WRITE\_SUPPORT \*/

/\*

\* PDU types in SNMPv1 and SNMPsec

\*/

#define SNMP\_MSG\_TRAP (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x4) /\* a4=164 \*/

/\*

\* PDU types in SNMPv2p, SNMPv2c, SNMPv2u, SNMPv2\*, and SNMPv3

\*/

#define SNMP\_MSG\_GETBULK (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x5) /\* a5=165 \*/

#define SNMP\_MSG\_INFORM (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x6) /\* a6=166 \*/

#define SNMP\_MSG\_TRAP2 (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x7) /\* a7=167 \*/

/\*

\* PDU types in SNMPv2u, SNMPv2\*, and SNMPv3

\*/

#define SNMP\_MSG\_REPORT (ASN\_CONTEXT | ASN\_CONSTRUCTOR | 0x8) /\* a8=168 \*/

* + 1. 编译

扩展 net-snmp agent有三种方式：

* + - 1. 静态联编（此方式需要net-snmp源码树）
         1. 把生成的代码放入源码目录agent\mibgroup下。
         2. 执行配置

./configure --with-mib-modules=examples/notification.c

--with-mib-modules=后面参数是 agent/mibgroup目录下模块文件名

* + - * 1. 生成

make

这样，模块就被编译进libnetsnmpmibs.so 库中

* + - * 1. 检查

nm libnetsnmpmibs.so ｜ grep init\_notification

* + - 1. 动态加载库(windows不支持此方法)
         1. 简单编译：

前提示是，在系统中已安装好net-snmp的开发环境。（包括头文件也安装到系统头文件目录，libsnmp.a等库安装到系统目录下）

*gcc -I. -c -o* notification.cnotification.c#生成目标文件

*gcc -g -fPIC -shared -o* notification*.so* notification*.o* #生成动态库文件.so

* + - * 1. 用automake编译

建立 configure.in和Makefile.am

*./configure*

*make*

参考：[autoconf和automake生成makefile](http://blog.csdn.net/kl222/article/details/6037145)  http://blog.csdn.net/kl222/article/details/6037145

* + - 1. 子代理方式
    1. 加载
       1. 静态联编（此方式需要net-snmp源码树）
          1. 按照6.2.5.1方法进行编译
          2. 加载

重启snmpd -Dexample\_notification #（打开输出调试信息）

* + - * 1. 检查是否加载成功

检查日志。如果成功，会在日志文件（/var/log/net-snmp.log）中输出：

example\_notification: initializing (setting callback alarm)

* + - 1. 动态加载库(windows不支持此方法)
         1. 不停snmpd进程的情况下加载

需要net-snmp支持ucd-snmp/dlmod模块。默认是没有编译此模块的，所以你需要自己配置，并重新编译。

*./configure --with-mib-modules=ucd-snmp/dlmod*

--with-mib-modules=后面参数是 agent\mibgroup\目录下模块文件名

注：此方法本人未成功，可能是没有配置ucd模块。

* + - * 1. 修改snmpd.conf加载

修改配置文件

格式：

dlmod 模块名 模块动态库文件（.so）位置

dlmod notification /data/snmp/test/.libs/ libnetsnmpmibs.so

加载

打开调试开关

如果输出调试信息是这个：

DEBUGMSGTL(("example\_notification",

"initializing (setting callback alarm)\n"));

那么打开调试开关 example\_notification

启动snmpd服务

*snmpd -Ddlmod –D*example\_notification

检查是否加载成功(二种方法)

检查日志

如果成功，会在日志文件（/var/log/net-snmp.log）中输出：

dlmod: dlmod\_path: /usr/lib/snmp/dlmod

dlmod: dlmod\_create\_module

dlmod: dlmod\_load\_module notification： /data/snmp/test/.libs/ libnetsnmpmibs.so

example\_notification: initializing (setting callback alarm)

用lsof检查

linux-65wt:/etc/snmp # ps -ef| grep snmpd

root 25124 27661 0 11:43 pts/5 00:00:00 tail -f net-snmpd.log

root 25153 1 0 11:43 ? 00:00:00 /usr/sbin/snmpd -c /etc/snmp/snmpd.conf -Ddlmod, example\_notification -r -A -LF d /var/log/net-snmpd.log -p /var/run/snmpd.pid

linux-65wt:/etc/snmp # lsof -p 25153

snmpd 25153 root mem REG 8,2 28511 111979 /data/snmp/test/.libs/ libnetsnmpmibs.so

* + - 1. 子代理方式
  1. 测试

1. 工具
   1. 应用工具：
      1. snmptranslate [OPTIONS] OID [OID]...

-T TRANSOPTS Set various options controlling report produced:

B: print all matching objects for a regex search

d: print full details of the given OID

p: print tree format symbol table

a: print ASCII format symbol table

l: enable labeled OID report

o: enable OID report

s: enable dotted symbolic report

z: enable MIB child OID report

t: enable alternate format symbolic suffix report

用于解析OID成为可读信息

* + 1. snmpget [OPTIONS] AGENT OID [OID]...

*root@ubuntu:~/reach# snmpget -v 1 -c public localhost subscriberPort.1*

*REACH-MIB::subscriberPort.1 = INTEGER: 10*

多值查询：

root@ubuntu:~/reach# snmpget -v 1 -c public localhost subscriberPort.1 subscriberStatus.1

REACH-MIB::subscriberPort.1 = INTEGER: 10

REACH-MIB::subscriberStatus.1 = INTEGER: addSubscribe(4)

此命令仅能用于实体查询

* + 1. snmpgetnext [OPTIONS] AGENT OID [OID]...

*root@ubuntu:~/reach# snmpgetnext -v 1 -c public localhost subscriberPort*

*REACH-MIB::subscriberPort.1 = INTEGER: 10*

* + 1. snmpwalk [OPTIONS] AGENT [OID]

用于查询表

snmpwalk –v 1 –c public localhost subscriberTable

* + 1. snmpset [OPTIONS] AGENT OID TYPE VALUE [OID TYPE VALUE]...

*root@ubuntu:~/reach# snmpset -v 1 -c public localhost subscriberIP.1 a 192.168.0.1 subscriberPort.1 i 1000*

*REACH-MIB::subscriberIP.1=IPADDRESS:192.168.0.1*

*REACH-MIB::subscriberPort.1 = INTEGER: 1000*

在net-snmp安装目录下 agent/mibgroup/examples/ 下，有扩展 net-snmp 的例子。

* 1. MG-SOFT工具

分为以下几部分：

Browser:浏览MIB库，并提供snmp工具

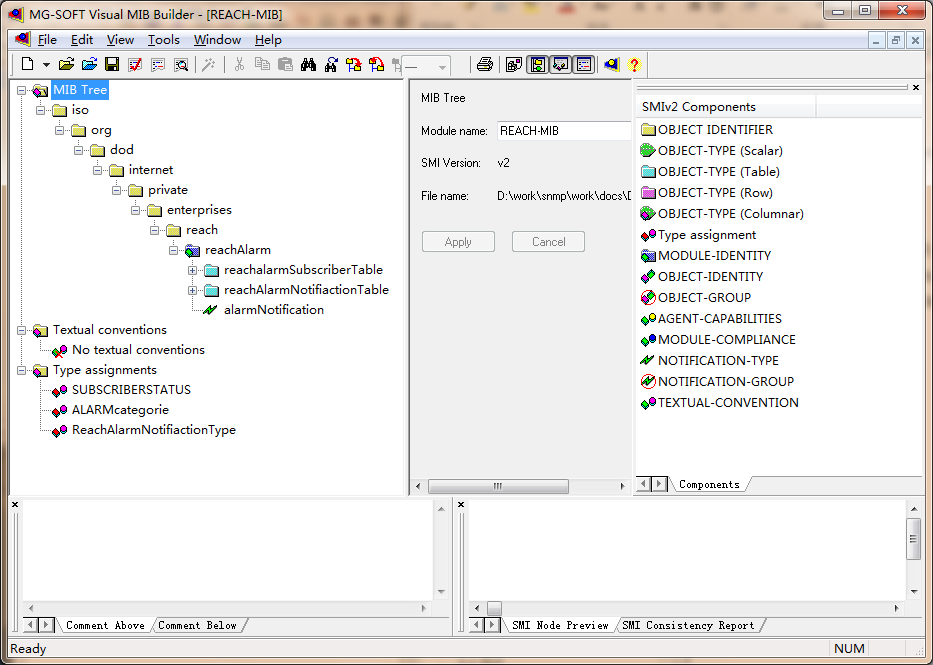
Builder：用于建立MIB库SMI

Compiler：用于把建立好的SMI编译成Browser格式

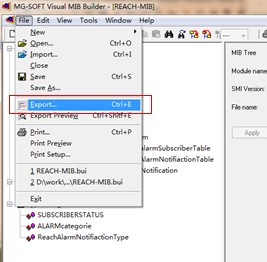
Explorer：

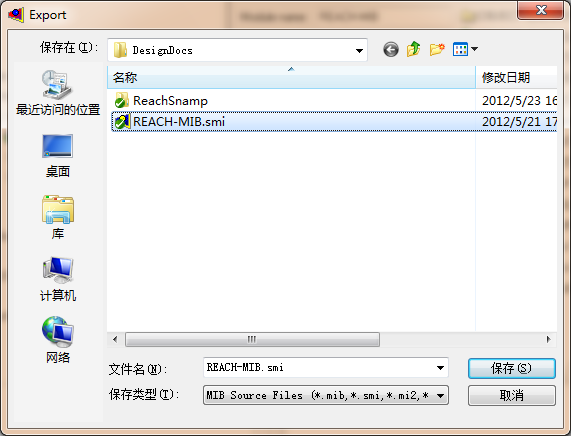
用builder生成一个SMI，SMI有三种实体：标量（scalar）、表（table）、消息（notification）

建立SMI



导出SMI

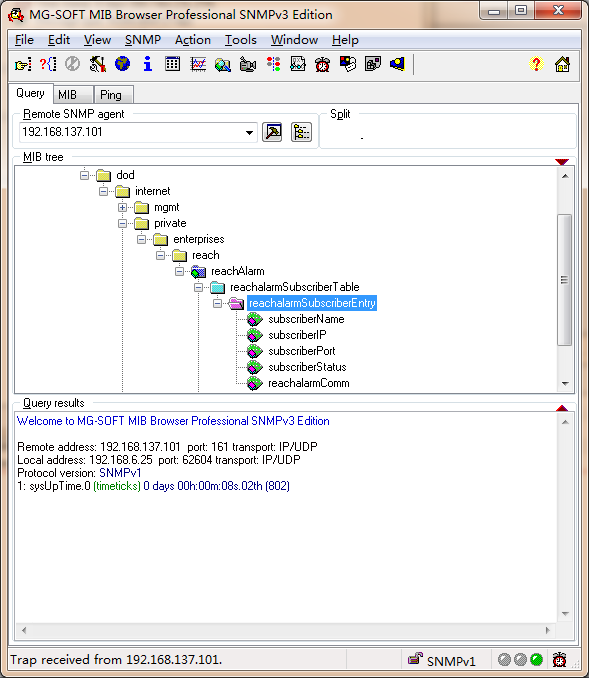


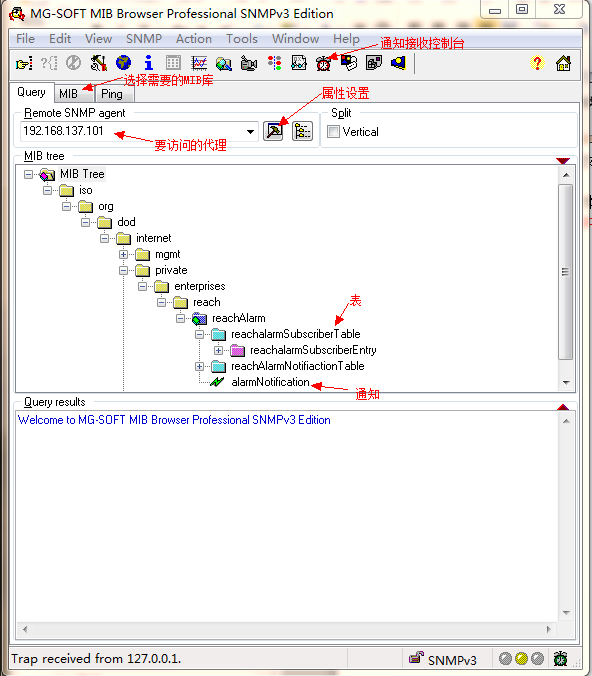


点保存，就可以生成文本格式的SMI

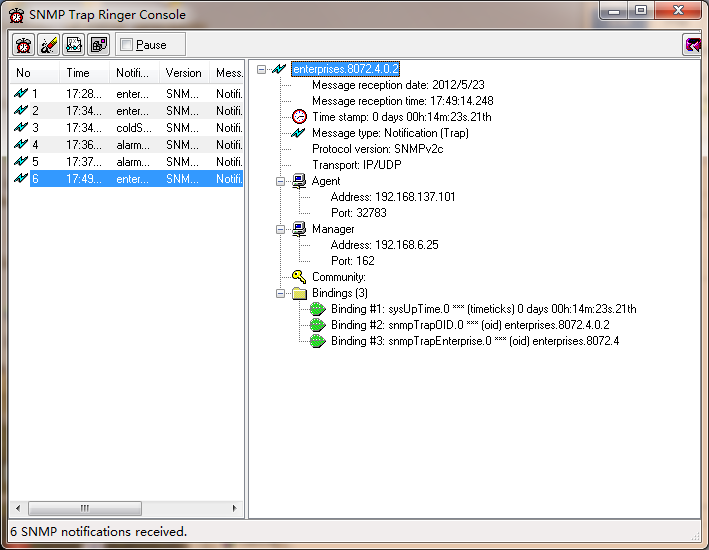
在LINUX下用的是文本格式的SMI。

注意，路径中不能含中文，否则，导出编译后，不能把编译后的文件保存到MIB-SOFT自己的MIB中。





通知接收控制台：



注意，操作系统可能已安装了默认的SNMP 消息跟踪器，这里，需要先停止系统的SNMP消息跟踪器服务。