NET-SNMP 代理配置手册

中文版

译者：ioerr

2010 年 7 月 19 日

原文属于 windows 版 net-snmp 5.5.0 版自带 chm 文档中的 snmpd.conf 部分，如果大家在类 unix 环境中使用，可能会有区别。

从 5 月 21 日开始翻译到现在差不多 2 个月，校稿一次，实在没有心力校稿第二次了，呵呵。费了这半天劲希望对大家有所帮助，如果发现翻译有错误欢迎发邮件：ZHL98040011@hotmail.com，我会立即更正，以便大家参考。

大家可以随意转发，但请带上俺的名字 ioerr，每当我在别的网站看见我翻译的文档，心中之得意…嘿嘿 **:D**。

但请勿用于商业用途，否则后果自负。

还有下面几个文档在我的博客上（http://hi.baidu.com/ioerr），欢迎参考。

MRTG 配置参考手册
NTOP 中文手册
openBSD 4.2 packages 和 ports 系统

名称

snmpd.conf - Net-SNMP's snmp 代理的配置文件

描述

Net-SNMP 代理使用一个或者更多个配置文件来控制它的运行和提供管理信息。这些配置文件（snmpd.conf 和 snmpd.local.conf）位于 snmp_config（5）手册页指出的目录中。

snmpconf（是一个 perl 脚本程序）可以为 snmp 代理生成最常用的配置文件。查看 snmpconf（1）来获取进一步的详细信息，或者尝试运行命令：

snmpconf -g basic_setup

这里有大量的指令可以指定，绝多数可以被划入四个大类：

* 用户访问控制类
* 控制代理提供信息类
* 本地系统监控类
* 代理功能扩展类

有一些指令不能自然的划入这四类中，但是这四类这些指令包括了典型的 snmpd.conf 配置文件的主要部分。全部指令请运行以下命令来获得：

snmpd –H

代理行为控制

虽然绝大多数指令与代理提供的 MIB 包含的信息有关，但还有一些指令用于控制 snmpd 代理本身--通常它被认为是一个提供服务的守护进程。

agentaddress [<transport-specifier>:]<transport-address>[,...]

定义监听地址列表，在这些地址上接收 SNMP 请求。参见 snmpd（8）手册的 LISTENING ADDRESSES 一节，来获取关于定义监听地址格式的详细信息。

默认监听所有 IPv4 接口的 UDP161 端口。

agentgroup{GROUP|#GID}

在打开监听端口后切换到指定的 group。这可能是一个组名，也可能是“#”开头的组 ID。

agentuser{USER|#UID}

基本同上，不过是切换用户，而不是 group。

leave_pidfile yes

指示 snmp 代理在关闭的时候不要删除它的 pid 文件。等价于在命令行使用“-U”参数。

maxGetbulkRepeats NUM

在一次 getbulk 请求中，设置对某变量的最大重复响应次数。设置为“0”使用默认值，设置为“-1”则不限制。因为内存是提前分配的，如果你的用户群不可信，那么设置为不限制是不安全的。超过设置值的响应将会被截断。

默认设置为“-1”。

maxGetbulkResponse NUM

对一次 getbulk 请求，允许的最大的响应次数。默认设置为“100”。设置为“0”启用默认值，设置为“-1”则不进行限制。因为内存是提前分配的，如果你的用户群是不可信的，那么设置为不限制是不安全的。超过设置值的响应将会被截断。

通常情况下，响应的数量将不会被允许超过 maxGetbulkResponses，并且返回的响应次数是被查询变量的整数倍，重复计算的次数允许低于这个数字。

不像 maxGetbulkRepeats 是首先被处理的。

SNMPv3 配置

SNMPv3 需要 SNMP 代理定义一个唯一的“engineID”用于对 SNMPv3 请求进行响应。这个 ID 将会自动生成，其生成使用了两个合理的不可预测的值：一个伪随机数，一个当前时间（以秒为单位）。这是被推荐的方法。当然还有其它的方法来定义 engineID：

engineID STRING

指定生成 engineID 的字符串。

engineIDType 1|2|3

指定 engineID 从 IPv4 地址(1)，IPv6 地址(2)或者 MAC 地址(3)生成。注意：修改 IP 地址（或者更换网卡）会引起问题。

engeineIDNic INTERFACE

当前面的 engineType 定义为使用 MAC 的时候，指定 snmpd 使用哪个网络接口。如果 engineIDType 3 不被指定，那么这条指令无效。

默认使用 eth0 接口。

SNMPv3 认证

SNMPv3 开始被设计为使用基于用户的安全模型（User-Based Security Model，USM），它包含了一个用于 SNMPv3 协议的私有用户和密钥列表。然而，在实际运作的社区中，认为再另行维护一个数据库太痛苦而宁愿采用现存的架构。为此，IETF 创建了 ISMS 工作组来处理这个问题，ISMS 工作组于是决定在 SSH 和 DTLS 之上运行 SNMP，从而使用现有的用户和认证架构。

SNMPv3 USM Users

想要使用基于 USM 的 SNMPv3，你需要创建用户。推荐使用 net-snmp-config 命令来做，但是也可以自己使用 createUser 指令来做：

createUser [-e ENGINEID] username (MD5|SHA) authpassphrase [DES|AES] [privpassphrase]

MD5、SHA 是使用的身份认证类型，DES/AES 是使用的隐私加密协议。如果 privpassphrase 口令没有指定，默认为和验证字段的口令相同。注意这时创建的用户是没有用的，除非被加入到上面被提到的 VACM 访问控制表里面。

SHA 身份认证和 DES/AES 需要安装 OpenSSL，并且 snmp 代理需要在编译时添加 OpenSSL 相关支持。MD5 验证可以在没有 OpenSSL 的情况下使用。

警告：口令最短为 8 个字符。

SNMPv3 用户可以在运行的时候使用 snmpusm（1）命令来创建。

我们不介绍这条指令的使用方法和在哪里使用它（详情见后），请使用“net-snmp-config --create-snmpv3-user”，它知道把这些指令添加到正确的地方。

这条指令应该被放到/var/net-snmp/snmpd 的配置文件中，而不是通常的位置。理由是：相关信息在从文件中读取后会被删除（不为用户保存主密钥），并用从这个口令中衍生出来的密钥来取代之。这个密钥是一个本地密钥，如果被盗的话，不会被用于访问其它的代理，但如果原始的口令被盗，就会了。

如果你想要针对一个特定的 engineID 来本地化用户（这主要用于类似的 snmptrapd.conf 文件中），你可以使用“-e”参数来指定这个 EngineID，它是一个十六进制的值（比如：“0x01020304”）。

如果你想要直接生成主密钥或者本地密钥，请使用一个十六进制的字符串来取代给定的口令（字符串用‘0x’开头），并分别在字符串前使用 -m 或

者-l 参数。例如：

[这些密钥*不安全*，但是可以很容易的进行分析、学习。请在实际中生成随机密钥而不是使用这个例子。]

```
createUser myuser SHA -l 0x000102030405060708090001020304050607080 9 AES -l 0x000102030405060708090001020304050607080 9
createUser myuser SHA -m 0x000102030405060708090001020304050607080 9 AES -m 0x000102030405060708090001020304050607080 9
```

根据口令本地化算法的不同，本地化的隐私加密密钥要达到算法所需长度（所有被支持的算法都需要 128 位）。虽然，主密钥长度需要满足身份认证算法的长度要求，而不是隐私加密算法的长度要求（MD5:16 字节，SHA:20 字节）。

SSH 支持

想要使用 SSH，你需要配置 sshd 来调用 sshtosnmp，同样需要配置访问控制列表，从而允许通过 tsm 安全模型使用指定用户（由 ssh 提供给 snmpd）进行访问。

DTLS 支持

对于 DTLS，snmpd 需要配置它的 X.509 证书，同样客户的证书也需要被明确的允许访问代理。访问控制也需要被设置为允许通过“tsm”安全模型来访问。X.509 证书中主题的 CommonName 将会被作为 SNMPv3 的用户名传递给 snmpd 来使用。参见 http://www.net-snmp.org/wiki/index.php/Using_DTLS 来获取关于配置 DTLS 的更详细信息。

defX509ServerPub FILE
defX509ServerPriv FILE

这两条指令用于指定证书的公钥和私钥文件，这个证书 snmpd 在处理呼入的连接时使用。

defX509ClientCerts FILE

这条指令指定客户在连接服务器时，使用的包括所有公钥（或者公钥的 CA）的文件。

访问控制

snmpd 支持基于“视图”的访问控制（VACM，在 RFC2575 中定义），控制谁能够检索并更新信息。因此，它能够识别不同的与访问控制相关的指令。

传统的访问控制

绝大多数简单的访问控制可以用 rouser/rwuser（对于 SNMPv3）或者 rocommunity/rwcommunity（对于 SNMPv1、SNMPv2）来满足。

rouser [-s SECMODEL] USER [noauth|auth|priv [OID | -V VIEW [CONTEXT]]]
rwuser [-s SECMODEL] USER [noauth|auth|priv [OID | -V VIEW [CONTEXT]]]

这两条指令用于分别指定 SNMPv3 用户的只读访问（GET 和 GETNEXT 操作）和读写操作（GET 和 GETNEXT 和 SET）。默认情况下，使用默认的上下文环境，对于经过验证的 SNMPv3 请求（包括加密的 SNMPv3 请求）允许对整个 OID 树进行访问。可以指定参数 noauth（允许未经授权的请求）来使用最低层次的安全保护，或者指定参数 priv（强制使用加密）。OID 参数定义用户可以访问的以指定 oid 为根的子树，或者在此位置指定“视图”而不是 OID 子树。也能够指定一个可选的上下文，或者使用“context*”来表示一个上下文前缀。如果没有 context 被指定（或者使用‘*’），这条指令将会匹配所有可能的上下文。

如果 SECMODEL 被指定，那么就会使得用户处于相应的安全模型中（注意：同一个用户名可能会处于不同的安全模型中，并可以通过‘访问控制’来进行区分）。默认的安全模型是“usm”；当使用 SSH 或者 DTSL 的时候，安全模型是“tsm”；如果有内置的 Kerberos 支持，那么安全模型是“ksm”。

rocommunity COMMUNITY [SOURCE [OID | -V VIEW [CONTEXT]]]
rwcommunity COMMUNITY [SOURCE [OID | -V VIEW [CONTEXT]]]

这两条指令分别为只读模式（GET and GETNEXT）和读写模式（GET, GETNEXT and SET）分别指定 SNMPv1、SNMPv2 的社团名称。默认情况下，这允许访问整个 OID 树。SOURCE 参数用于限定 snmp 请求的来源系统-详情参见 com2sec。OID 参数限定允许被访问的 OID 子树，或者“视图”。上下文对于使用 community 的 SNMPv1、SNMPv2 意义要少些，但是前面关于 context 的说明同样适用。

rocommunity6 COMMUNITY [SOURCE [OID | -V VIEW [CONTEXT]]]
rwcommunity6 COMMUNITY [SOURCE [OID | -V VIEW [CONTEXT]]]

这两条指令适用于 IPv6 环境中发出的请求（如果代理支持这样的传输）。所有参数含义和前面的 IPv4 相同。

在各种情况下，对于 SNMPv3 中的某个用户，要么是只读类型-rouser，要么是读写类型-rwuser，因为 rwuser 提供了 rouser 的所有权限（而且同时提供了 SET 权限）。对于 v1/v2 环境中的社区名称（community）情况也一样。

对于更复杂的访问需求（比如使用 GET 或者 SET 来访问两个或者更多个不同的 OID 树，或者不同的视图）应该使用其它的访问控制机制。注意，如果有几个不同的 community 或者 SNMPv3 环境下的用户需要被授予同样的访问权限，使用主流的 VACM

配置指令更有效。

VACM 配置

VACM 的灵活配置靠的是四条配置指令实现的 - com2sec，group，view 和 access。这实现了下面 VACM 表的配置。

com2sec [-Cn CONTEXT] SECNAME SOURCE COMMUNITY
com2sec6 [-Cn CONTEXT] SECNAME SOURCE COMMUNITY

把 SNMPv1、SNMPv2 社区名称（community）映射为一个安全名（secName）- 一个特定范围的 IP 地址段或者全部地址（"default"）。一个受限的访问源要么是某个主机名或者 IP 地址,要么是某个网段 - 使用 IP/MASK 形式指定（例如：10.10.10.0/255.255.255.0），或者 IP/BITS 例如：10.10.10.0./24），以上同样适用于 IPv6。

同一个 community 可以在几条不同的配置指令中使用（需要访问源不同），与 snmp 访问源相匹配的第一条指令会被选中。不同的 source/community 组合也能被映射到相同的安全名（secName）中。

如果 CONTEXT 被定义（使用-Cn 参数来指定），community 将会被映射到一个处于 SNMPv3 上下文的安全名中。否则默认的上下文("")会被使用。

com2secunix [-Cn CONTEXT] SECNAME SOCKPATH COMMUNITY
这是在 Unix 版本的 com2sec。

group GROUP {v1|v2c|usm|tsm|ksm} SECNAME
把一个安全名（处于特定的安全模型中）映射到一个组中。几条 group 指令能使用同一个组名，允许一条 access 指令对应多个用户或者 community。

注意你必须为 v1/v2 分别进行 group 定义。这样一条 com2sec 指令常常需要两条 group 指令对应。（译者:详见帮助文档中的实例。）

view VNAME TYPE OID [MASK]
把整个 OID 树的一个子树定义成一个视图（view）。多条指令可以用一个视图名字，这样可以构建更复杂的 OID 子树集合。TYPE 为“included”或者“excluded”，这也是用来帮助建立复杂视图的（比如可以将指定子树中的某些敏感部分排除在外。）

MASK 是一个十六进制的字符串（使用“.”或者“:”分隔）,其中“置 1”的位表示 OID 中的相应位置需精确匹配。如果没有指定，默认为需精确匹配（所有的位均为‘1’），这样就可以定义一个 OID 树。如下：

```
view iso1 included .iso  0xf0
view iso2 included .iso
view iso3 included .iso.org.dod.mgmt  0xf0
```

这些指令都定义了同一个视图，覆盖了整个“iso”OID 子树（第三个例子中忽略了那些没有被掩码覆盖的分隔符）。

更有用的地方是，掩码可以被用来定义一个视图，其可以包括表中某些特定的行，这需要对特定的表的索引值进行匹配，但是需要跳过列分隔符：

```
view ifRow4 included  .1.3.6.1.2.1.2.2.1.0.4  0xff:a0
```

请注意掩码长度如果长于 8 个 bit，就必须使用‘:’来分隔各个字节。（注：在 net-snmp 的网站看到一篇文章，说现在也可以使用‘.’号。）

(16:28 2010-5-25 关于前面这个 mask,验证了一下，费姥姥劲了!!!)

access GROUP CONTEXT {any|v1|v2c|usm|tsm|ksm} LEVEL PREFX READ WRITE NOTIFY

依据接收到的请求，把用户/团体（community）映射到三个视图之一，需要为用户或者团体指定所属的安全模型和最低安全级别，并指定特定的上下文。

LEVEL 设置为 noauth，auth 或者 priv。PREFX 指定 CONTEXT 应该怎样匹配外来的请求,可设置为 exact 或者 prefix。READ,WRITE 和 NOTIFY 分别对应 GET*，SET 和 TRAP/INFORM 操作（虽然 NOTIFY 现在不再使用）。对于 v1、v2c 访问，LEVEL 需要设为“noauth”。

按类型进行视图配置

最后一组指令扩展了 VACM，使它具有更加灵活的机制，可以被用于满足更复杂的访问控制需求。比起只有三种视图的标准 VACM 机制，这能够被用于配置不同的视图类型。就通常的 SNMP 代理而言，主要的两个视图类型是 read 和 write，与前面提到的主要访问指令的 READ、WRITE 视图相应一致。参考“snmptrapd.conf”手册，获取其它视图类型的相关信息。

authcommunity TYPES COMMUNITY [SOURCE [OID | -V VIEW [CONTEXT]]]

这条指令可以代替 rocommunity/rwcommunity 指令。TYPES 通常是 read 或者“read，write”。视图的定义可以是一个 OID 子树（和前面说的一样），或者是一个命名的视图（使用 view 指令定义）--这样有更大的灵活性。如果没有定义，就允许访问整个 OID 树。如果 CONTEXT 被指定了，访问控制就在 SNMPv3 的上下文中被配置。否则使用默认的上下文（""）。

authuser TYPES [-s MODEL] USER [LEVEL [OID | -V VIEW [CONTEXT]]]

这条指令是对 rouser/rwuser 指令的替代。TYPES,OID,VIEW 和 CONTEXT 参数含义与 authcommunity 相同。

authgroup TYPES [-s MODEL] GROUP [LEVEL [OID | -V VIEW [CONTEXT]]]

是与 authuser 指令配合使用的，用于控制一个特定组的访问（和前面的 group 指令类似）。authuser 和 authgroup 默认用于已经认证的请求--LEVEL 可被指定为 noauth 或者 priv 来允许未认证的请求，也可用于需要加密的请求。authuser 和 authgroup 指令也默认用于配置 SNMPv3/USM 请求--使用“-s”来指定替代的安全模型（使用同前面‘access’一样的参数值）

authaccess TYPES [-s MODEL] GROUP VIEW [LEVEL [CONTEXT]]

这条指令用于配置特定组的访问控制，指定视图的名字和类型。"MODEL"和"LEVEL"参数与前面的 authgroup 类似。如果 CONTEXT 被指定了，访问就在 SNMPv3 的上下文中被配置(或者 CONTEXT 参数使用'*a'结尾)。否则就是用默认的上下文（""）。

setaccess GROUP CONTEXT MODEL LEVEL PREFIX VIEW TYPES

这条指令相当于 access 指令，典型的用途是列出视图的类型“read”或者“read, write”。（或者请参考"snmptrapd.conf"来）。所有其他的参数含义同 access 指令的参数相同。

系统信息配置

绝大多数 NET-SNMP 代理报告的信息是通过底层系统获取的或者通过 set 操作进行动态配置获取的（并保留直到下一次代理运行）。然而，某些 MIB 对象是可以通过 snmpd.conf（5）文件进行配置的。

System 组

绝大多数在‘system’组中的标量对象可以用下面的方法进行配置：

sysLocation STRING
sysContact STRING
sysName STRING

这几条指令分别设置系统的物理位置，联系方法，系统名称（sysLocation.0，syscontact.0，还有 sysName.0）。正常来说这些对象是可以通过 SET 来进行写操作的。然而，这些配置指令将使得相应的对象变成只读，如果进行 SET 操作，将会得到 notWritable 的错误提示。

sysServices NUMBER

这条指令设置 sysServices.0 对象。对于一个主机系统，设置为 72 是不错的选择（代表应用层+端到端层）。如果这条指令没有设置，那么对于 sysServices.0 对象就查询不到值。

sysDescr STRING
sysObjuectID OID

设置“系统描述”或者“代理 OID”。虽然设置的这些 MIB 对象不可写，但这些指令可以让网络管理员们为这些 OID 配置合适的值。

Interfaces 组

interface NAME TYPE SPEED

在这些接口上，当代理不能正确的获得这些信息时，这条指令被用于提供网络接口类型和工作速度相关信息。TYPE 是一个在 IANAifType-MIB 中提供的类型值,能够用数字或者名字指定(前提是这个 MIB 被加载了。)

Host Resources 组

这需要代理内建对“主机模块”的支持（这个模块在大多数平台中默认加载）。

ignoreDisk STRING

控制系统中哪些磁盘设备被扫描，其结果被放入 hrDiskStorageTable 中(还会放入 hrDeviceTable)。HostRes 包含了一个用于匹配当前操作系统中磁盘设备的的匹配模式。在代理尝试打开某些磁盘设备时可能会引起阻塞。这也可能会在遍历中导致超时，还可能会导致行为不一致。这条指令可以指定特定的设备不被检查（单独明确指定，或者使用通配符指定）。

注意：请参考 host/hr_disk.c 文件并检查某个特定的平台的 Add_HR_Diskentry 中与调用相关的内容，以便知道在哪些设备会被扫描。

这些模式包含在一个或多个通配符表达式中。请参见 snmpd.examples（5）中的实例。

skipNFSInHostResources true

控制在 hrStorageTable 中，NFS 和类 NFS 文件系统是否被忽略(设置为 true 或者 1 忽略，设置为 flase 或者 0 不忽略--这是默认值)。如果 Net-SNMP 代理在处理 NFS 文件系统时会进入挂起状态，你可以试试设置为 1。

storageUseNFS[1|2]

控制在 hrStorageTable 中怎样报告 NFS 和类 NFS 文件系统的类型。历史上有两种类型：“网络磁盘”、“固定磁盘”，Net-SNMP 代理把这些文件系统都报告为“固定磁盘”，并且这是默认的行为。设置这条指令为“1”，将会把这样的文件系统报告称“网络磁盘”，这对于 Host Resources MIB 是需要的。

进程监控（Process Monitoring）

在 Host Resources 的 MIB 中，hrSWRun 提供在本地系统中独立进程的有关信

息。在 UCD-SNMP-MIB 中的 prTable 中，通过对指定的系统服务进行报告来进一步完善这个功能(可能会涉及多个进程)。这需要代理内建对 ucd-snmp/proc 模块的支持（默认已经包含）。

proc NAME [MAX [MIN]]

监测本地系统中运行的名称为 NAME 的进程的数量（这个名称 NAME 是在“/bin/ps -e”中显示的)。

如果名称为“NAMEd”的进程少于 MIN 或者大于 MAX，那么就会把相应的 prErrorFlag 置为 1，并且会通过 prErrMessage 来对此报告一条合适的错误信息。

注意：这个状态不会自动的触发一个“trap”来报告发生的问题 - 参考后面 DisMan Event MIB 一节。

如果 MAX 和 MIN 都没有指定（或者都是 0)，他们将会分别默认为“无穷大”和“1”(“保证至少有一个进程”)。如果只有 MAX 被指定了，MIN 将会默认为“0”(“保证不超过 MAX”)。(注：感觉手册废话好多:D)

procfix NAME PROG ARGS

注册一条命令用于修复名称为 NAME 的进程出现的错误。这项设置在指定的 prErrFix 设置为‘1’的时候被调用。

注意：这条命令将不会被自动调用。

procfix 指令必须在相应的 proc 指令后被配置，并且不能自己处理自己。

如果没有 proc 指令被定义，那么对 prTable 的 walk 操作会失败（noSuchObject)。

磁盘使用监测（Disk Usage Monitoring）

这项功能需要代理内建对 ucd-snmp/disk 模块的支持（默认已被包含)。

disk PATH [MINSPACE | MINPERCENT%]

监测 PATH 指定磁盘的使用空间情况。

最小的磁盘空间阀值可以使用 kB（MINSPACE）来指定或者通过磁盘使用百分比（注意要使用“%”）来指定，如果两项都没有指定，默认阀值为 100kB。如果可用的磁盘空间低于此阀值，那么相应的 dskErrorFlag 将会被置“1”，并通过在 dskErrorMsg 报告错误信息。

注意：这种情况下，不会自动触发报告问题的陷阱（trap） - 参见后面 DisMan Event MIB 一节。

includeAllDisks MINPERCENT%

使用指定的阀值（或百分比），监控在系统中所有磁盘的使用情况。对单个磁盘的阀值可以使用合适的 disk 指令来进行校正（可以放置在 includeAllDisks 指令前面或者后面）。

注意：无论 disk 指令出现在 includeAllDisks 之前或之后，都可能会影响 dskTable 的索引情况。

只能有一条 includeAllDisks 指令 - 出现后面的 includeAllDisks 都会被忽略。

当代理使用 setmntent(3) 和 getmntent(3), fopen(3) 和 getmntent(3), setfsent(3)和 getfsent(3) 这些系统调用启动时，被 mounted 的磁盘将会被检测。如果上面的系统调用都不能运行，那么就会监测"/"分区（在类 unix 系统中，'/' 总是被假定存在的）。在代理启动后 mount 的磁盘不会被监测。

如果 disk 指令或者 includeAllDisks 指令都没有定义，那么对于 dskTable 的 walk 操作将会失败（noSuchObject）。

系统负载监测（System Load Monitoring）

这需要代理内建对 ucd-snmp/loadave 模块或者 ucd-snmp/memory 模块的支持（这两项均已默认包含）。

load MAX1 [MAX5 [MAX15]]

监测本地系统的负载情况，在这里分别指定 1min，5min 和 15min 的平均值。如果其中有超过指定阀值的，那么就会产生相应的 laErrorFlag，其值会被置为 1，并生成一条信息保存在 laErrMessage 中。

注意：这种状态将不会自动的触发陷阱来报告消息 - 参见后面的 DisMan Event MIB 一节。

如果没有指定 MAX15 阀值，默认为使用 MAX5 的阀值。如果 MAX5 和 MAX15 都省略了，默认使用 MAX1 的阀值。如果这条指令没有指定，所有的阀值将是使用 DEFMAXLOADDAVE 的值。

如果阀值被指定为 0，代理将不再通过相关的 laErrorFlag 或者 laErrMessage 来报告错误，不再监测系统当前负载。

与 proc 和 disk 指令不同的是，即使 load 指令不存在，对 laTable 的 walk 操作将会成功（假定代理的 ucd-snmp/loadave 模块被启用。）。

swap MIN

监测本地系统的交换空间还有多少可用。如果低于设置的阀值（MIN kB），那么 memErrorSwap 将会被置为 1，并且生成一条错误信息保存在 memErrorMsg 中。

注意：这种情况将不会自动触发陷阱来报告发生的问题 - 参见后面 DisMan Event MIB 一节。

如果这条指令没有被指定，默认的阀值为 16MB。

日志文件监测（Log File Monitoring）

需要代理内建对 ucd-snmp/file 模块或 ucd-snmp/logmatch 模块（默认均已包含）。

file FILE [MAXSIZE]

监测指定文件的大小（单位 kB）。如果 MAXSIZE 被指定，并且文件大小超过了这个阀值，那么相应的会将 fileErrorFlag 置 1，并且会在 fileErrorMsg 中生成一条描述信息。

注意：这个情况将不会自动的触发陷阱报告这个问题 - 请参见后面的 DisMan Event MIB 一节。

注意：最多可以监测 20 个文件。

注意：如果没有 file 指令被配置，那么对于 fileTable 的 walk 操作会失败（noSuchObject）。

logmatch NAME FILE CYCLETIME REGEX

监测指定的文件中是否出现了指定的正则表达式。文件将要被读取的位置保存在内部，这样整个文件只会在第一次被全部读取，其后每次只读取新添加到数据。

NAME logmatch 实例的名字（将会在 ucd-snmp MIB 树的 logMatch/logMatchTable/logMatchEntry/logMatchName 中以 logMatchName 出现。）

FILE 日志文件的绝对路径。注意这个路径可以包含 date/时间指令（像 Unix 命令中一样）。参见“strftime”的手册页以获取可用的不同指令。

CYCLETIME 内部的时间间隔，以秒计算，为了对日志文件和内部变量更新而使用。注意：SNMPGET*操作将会立即触发日志文件的读取和变量更新。

REGEX 需要使用的正则表达式。注意：不要关闭引号中的正则表达式，即使表达式中有空格，因为引号会被作为表达式的一部分进行匹配。

例子：

logmatch apache-GETs /usr/local/apache/logs/access.log-%Y-%m-%d 60 GET.*HTTP.*

这条 logmatch 指令被命名为“apache-GETs”，“GET.*HTTP.*”作为正则表达式，它将会监测指定的日志文件（假定今天的日期为 May 6th 2009）：“/usr/local/apache/logs/access.log-2009-05-06”就是监测的对象。明天，日志的名字就会变成 access.log-2009-05-07。logfile 每 60 秒读取一次。

注意：最多可以配置 250 条 logmatch 指令。
注意：如果没有 logmatch 指令被定义，那么对 logMatchTable 的 walk 操作就会失败（noSuchObject）。

主动监控（ACTIVE MORNITORING）

通常情况下 SNMP 代理等待 SNMP 请求并进行响应 - 如果没有接收到请求，代理一般情况下不会有任何的动作。这一节介绍几种指令可以让 snmpd 扮演一个更具主动性的角色。

通告处理（NOTIFICATION HANDLING）

trapcommunity STRING

定义一个默认的 community 字符串，用于发送陷阱消息。注意，这条指令必须用在基于 community 的“陷阱接收端”定义指令之前。

trapsink HOST [COMMUNITY [PORT]]
trap2sink HOST [COMMUNITY [PORT]]
informsink HOST [COMMUNITY [PORT]]

定义通告接收端，通告可以有 SNMPv1 TRAPs/SNMPv2c TRAPs/SNMPv2 INFORM 几种类型。参考 snmpd（8）手册中 LISTENING ADDRESS 一节，获取关于定义 ip 地址格式的相关详细信息。如果 COMMUNITY 没有指定，那么将使用最近用过的 trapcommunity。
如果传送地址定义中不包括明确的端口定义，那么将会使用 PORT。如果 PORT 没有被定义，那么将会使用传统的 SNMP 端口 162。

注意：这种机制已不被推荐，并且监听地址应该用 HOST 指定。
如果定义了几个 sink 指令，那么每个通告都会生成多个拷贝（用相应的格式）。

注意：正常情况下，不要为一个接收端定义两条或三条 sink 指令。

trapsess [SNMPCMD_ARGS] HOST

提供了定义接收端的更通用的方法。SNMPCMD_ARGS 使用和 snmptrap（或者 snmpinform）相同的命令行参数来发送通告。参数-Ci 可以用来生成 INFORM

通告（用于-v2c 或者-v3）而不是生成不被响应的 TRAP。

这条指令适用于定义 SNMPv3 陷阱接收端。请参考 http://www.net-snmp.org/tutoria/tutorial-5/commands/snmptrap-v3.html 以获取关于 SNMPv3 通告的更多信息。

authtrapenable {1|2}

用于定义是否生成验证失败的陷阱消息（是：enabled(1)）或者（否：disable(2) - 默认）。通常情况下，相关的 MIB OID（snmpEnableAuthenTraps.0）是可读可写的，但是指定了这条指令就会变成只读，如果尝试对其进行 SET 操作，将会收到 notWritable 的错误响应。

v1trapaddress HOST

定义用于 SNMPv1 TRAP 消息的代理地址。如果这个参数没有定义，那么会随机的使用本地 IPv4 地址中的一个。当外界只能通过特定的 IP 地址来访问代理的时候，这个参数就很有用了（例如：有 NAT 或者防火墙的情况下）。

DisMan Event MIB

前一条指令用于配置陷阱消息向哪里发送，但是与什么时候发送，发送什么消息无关。下面要涉及的是 Event MIB - 由 IETF 的 DisMan（Distrabuted Management）工作组开发。

这要求代理内建对 disman/event 模块的支持(最近的发布版大多已默认具备此模块)。

注意：最新版本的实现和老版本在一些小的地方有所不同，现有的脚本可能需要做一点修改。

iquerySecName NAME

agentSecName NAME

指定当进行内部查询时使用的默认 SNMPv3 用户名,用于检索任何所需的信息(用于评估监测表达式或者构建一个通告的负载)。这些内部请求总是使用 SNMPv3，即使代理的普通查询请求是使用 SNMPv1、SNMPv2 的。

注意用户必须明确的被创建(createUser)和授予访问权限(比如:使用 rouser)。这条指令纯粹是为了指定使用哪个用户的 - 不是用来设置用户的。

monitor [OPTIONS] NAME EXPRESSION

定义一个监测的 MIB 对象。如果 EXPRESSION 条件成立（参加下文），那么这条指令将会触发相应的事件，并且会发送一个通告或者进行一次 SET 操作（或许两者都执行）。注意当表达式第一次匹配的时候，相应的事件只会触发一次。且这个监测项将不会被再次触发,直到条件再次变为 false,然后表达式再次匹配。NAME 是一个管理用的名字，被用于对 mteTriggerTable 进行索引（还有相关的其他 tables)。

注意这些监测都是内部使用 SNMPv3 查询的，来检索被监测的值（即使正常的查询使用 SNMPv1、SNMPv2）。

参见前面的 iquerySecName 指令。

EXPRESSION

事件 MIB 支持三种不同类型的监测表达式 - existence，boolean 和 thresholdtest（阀值测试）

OID | ! OID | != OID

定义 existence（0）类型监测。第一种情况，单纯是一个“OID”，那么就是指定使用 present（0）进行监测，当指定的 OID 生成时将会触发相应动作。第二种情况，“! OID”指定一个 absent（0）进行监测，当指定的 OID 被删除时会触发相应动作。“!= OID”指定一个 changed（2）监测，当被监测 OID 的值变化时触发相应动作。注意在 OID 之前必须要有空格!

OID OP VALUE

定义一个 boolean(1)监测。OP 是以下符号之一(!=, ==, <, <=, >, >=)，且 VALUE 应该是一个整型值。注意 OP 左右都必须有空格! 如果表达式为“某 OID !=0”（右边没有空格），这将不会被正确的处理。

OID MIN MAX [DMIN DMAX]

定义一个阀值监测。MIN、MAX 是整型值，指定阀值的范围。如果监测的 OID 的值不在此范围内，将会触发相应的事件。

注意上限阀值如果被触发，那么只有当监测值低于 MIN 后才会被还原。相似的，下限阀值如果被触发，只有当监测值超过 MAX 后才会被还原。
(原文：Note that the rising threshold event will only be re-armed when the monitored value falls below the lower threshold (MIN). Similarly, the falling threshold event will be re-armed by the upper threshold (MAX).)

DMIN、DMAX 参数是与上面相似的阀值检测，但是处理的是连续采样值之间的差值。

OPTIONS

这里有不同的选项可以控制监控正则表达式的实际执行。各选项如下：

-D 指定表达式处理采样值之间的差值（而不是采样值本身）。

-d OID

-di OID 为验证差值指定一个间断标志。-di 精确的使用指定 OID。-d 只需要处理的 OID 一部分符合指定的 OID。如果使用了-I，那么在这两个选项就相同了。

这个选项也包含了-D 的功能。

-e EVENT

设定当指定的监测项被触发后生成的事件。如果这个选项没有被指定，将会生成一个在 DISMAN-EVENT-MIB 中定义标准的通告。

-I 指定监测表达式需要精确匹配指定的 OID。默认情况下，指定的 OID 被认为是一个带通配符的对象，只要部分匹配即可。

-i OID

-o OID 定义一个额外的变量，当指定的监测项触发时，将其添加到 SNMP 通告中去。对于一个通配符类型的表达式来说，如果使用了-o 参数，那么匹配指定的部分即可。

但是对于-i 参数就需要精确的匹配指定的 OID。如果使用了-I 标志，那么这两个参数就没有什么区别了。

参见 strictDisman 以获取更过关于通告的相关细节。

-r FREQUENCY 指定对给定表达式的监测间隔。默认为 600s（即 10 分钟）。

-S 指定当代理初次启动的时候不对监测表达式进行评估。当到了第一次监测的时间后再进行第一次评估。

-s 指定当代理第一次启动时对表达式进行评估。这是默认的情况。
注意：初始评估触发的通告将在 coldStart 陷阱作用前发送。

-u SECNAME 指定一个安全名称用于扫描本地机，替代默认的

iquerySecName。再次说明，这个用户必须被明确的创建并被给予合适的权限。

notificationEvent ENAME NOTIFICATION [-m] [-i OID | -o OID]*

定义一个名称为 ENAME 的通告事件。这是靠-e ENAME 参数指定的监测项触发的。生成的通告应该是标准中定义的类型。(原文：NOTIFICATION should be the OID of the NOTIFICATION-TYPE definition for the notification to be generated.这句翻不好，大家自己对比看吧。-_-!!!)

如果-m 参数被指定，通告的附加信息应该包括标准的变量，其在通告的 MIB 定义里面的 OBJECTS 相关条款中已经说明。这个选项必须在

NOTIFICATION OID 之后（对于代理来说，其它相关的 MIB 文件必须是可用的并已经加载的）。否则这些变量必需被明确的列出（不是在这里，就是在相应的监测指令中）。

在标准的列表之后，-i OID 和-o OID 指定了在通告附加信息中另行添加的变量。如果触发这个事件的监测项包括了一个有通配符的表达式，那么这个通配符表达式将附加到-o 指定的 OID 前面（译者注：就是部分匹配即可），然而-i 指定的 OID 将会被精确匹配。如果在监测指令中使用了-I 参数，那么这在两个选项将不会有任何区别。

setEvent ENAME [-I] OID = VALUE

定义一个名为 ENAME 的 set 操作事件，把 VALUE(整型值)赋值给指定的 OID。这是将要被-e 参数（参见上面相关内容）的监测项触发的。

如果监测项包含有通配符表达式，那么会匹配前面部分相匹配的 OID。如果使用了-I 参数被指定给 monitor 或者 setEvent 指令，指定的 OID 将被精确匹配。

strictDisman yes

在 SNMP 通告定义中指定了在 OBJECT 规定里的变量应被首先添加（按指定顺序），然后是添加通告生成器觉得有用的“extra”类变量。最自然的方法是把这些必须的变量和 notificationEvent 联系起来，然后添加把与监测有关的能触发通告的变量添加到列表的末尾这是 Net-SNMP

Event MIB 的默认行为。

不幸的是，DisMan Event MIB 规则实际先声明了“触发相关”变量，然后是“事件相关”变量。这条指令用于恢复这项严格的行为（但实际上不太好用）。

注意：如果 notificationEvent -n 参数和 monitor -o（或者-i）一起使用，那么 Strict DisMan 顺序可能会导致无效的通告负载生成。

如果没有 monitor 项指定负载变量，那么设置这条指令无关紧要。

linkUpDownNotifications yes

这个参数将配置 Event MIB 表对 ifTable 进行监测，监测网络接口的启用、关闭，并在合适的时机触发 linkUp 和 linkDown 通告。

下面的配置是完全等价的：

```
notificationEvent   linkUpTrap      linkUp    ifIndex  ifAdminStatus
ifOperStatus
notificationEvent    linkDownTrap    linkDown  ifIndex  ifAdminStatus
```

```
ifOperStatus
monitor  -r 60 -e linkUpTrap   "Generate linkUp" ifOperStatus != 2
monitor  -r 60 -e linkDownTrap "Generate linkDown" ifOperStatus == 2
```

defaultMonitors yes
这将配置 Event MIB 表对不同的 UCD-SNMP-MIB 出现的问题进行监测（像 xxErrFlag 一样）。

下面的配置是完全等价的。

```
 monitor   -o prNames -o prErrMessage "process table" prErrorFlag != 0
 monitor   -o memErrorName -o memSwapErrorMsg "memory" memSwapError !=
0
 monitor   -o extNames -o extOutput "extTable" extResult != 0
 monitor   -o dskPath -o dskErrorMsg "dskTable" dskErrorFlag != 0
 monitor   -o laNames -o laErrMessage  "laTable" laErrorFlag != 0
 monitor   -o fileName -o fileErrorMsg  "fileTable" fileErrorFlag != 0
```

在这两组例子中的后面几个，snmpd.conf 应该包括 iquerySecName 指令，并相应配置 createUser 和其它权限控制指令。

DISMAN 的计划任务 --- DisMan Schedule MIB

DisMan 工作组还制定了一个“计划任务”机制（SET 操作）。这需要代理内建 disman/schedule 模块（默认已经包括）。

有三种方法来执行“计划任务”：

repeat FREQUENCY OID = VALUE
xxx 定时一个 SET 赋值操作，将一个整型值赋值给 MIB 中的某个 OID，每 FREQUENCY 秒执行一次。

cron MINUTE HOUR DAY MONTH WEEKDAY OID = VALUE
定时执行一个 SET 操作，将一个整型值赋值给 MIB 的某个 OID，通过 MINUTE 等参数指定具体时间。含义和 crontab(5)相同。

注意：这些参数应该在一个列表中用数字指定（之间用逗号分隔）。对于 MONTH 和 WEEKDAY 来说，不支持月份和星期几的名字，例如 June 和 Sunday 等，也不支持指定一个数值范围。可以使用通配符“*”。

DAY 参数也能接受负值，来指定从月末倒数的日子。

at MINUTE HOUR DAY MONTH WEEKDAY OID = VALUE
配置一次性的 SET 赋值，在第一个符合条件的时间运行，通过 MINUTE HOUR DAY MONTH WEEKDAY 参数指定。参数的含义和 cron 指令相同。

代理功能扩展

EXTENDING AGENT FUNCTIONALITY

原来的 UCD 套件的特色功能之一就是代理的功能扩展，而不仅仅是把新的 MIB 模块编译进去，还能配置运行中的代理来报告其它的信息。我们有很多种方法来支持代理的功能扩展，包括：

* 运行外部命令（exec、extend、pass）
* 动态加载代码（嵌入 perl，dlmod）
* 与其它 agents 进行通讯（proxy，SMUX，AgentX）

11:28 2010-6-1

增强型扩展命令

最早的扩展机制是支持运行系统命令和脚本。这样命令就不需要了解 SNMP 操作，或者支持特定的行为 - MIB 结构被设计为兼容任何形式的命令输出。使用这个机制需要代理内建对“snmp/extensible”、“agent/extend”模块的支持，需要支持一个或全部（默认情况下软件中两个模块都包括了）。

exec [MIBOID] NAME PROG ARGS
sh [MIBOID] NAME PROG ARGS

使用参数 ARGS 调用 PROG。默认情况下，退出状态和第一行输出放在 extTable 中，其它输出都会被丢弃。

注意：各个条目在 extTable 中出现的顺序由他们在配置文件中的顺序决定。这意味着增加新的 exec 或者 sh 指令或者重启代理，可能会影响到查询 extTable 所用的索引号（译者注：此部分中，代理对于每一条指令会生成一个索引号）。

在 exec 指令中 PROG 必须是可执行程序的完整路径，因为这是通过“exec()”系统调用实现的。想要调用脚本，请使用 sh 指令。

如果 MIBOID 被指定了，那么就以这个 OID 为根，返回 MIBOID.100.0，将其作为退出状态，并且整个命令的输出是以 MIBNUM.101 为根的“伪表”---对于每一行输出带有一行“row”。

注意：exec 和 sh 指令的这种形式的结果，不是 MIB 结构中的严格的形式。

这种机制不被推荐使用，会慢慢被淘汰 - 请参见“extend directive”（在后面）部分。

代理并不缓存退出状态和程序的输出。

execfix NAME PROG ARGS

注册一个在需要的时候可以调用的命令 - 典型用途为调用对应的 exec 或者 sh 命令，来响应某些操作或者处理错误。当一个名为 NAME 的条目的 extErrFix 被设置为整数值“1”时，这条命令会被调用执行。

注意：这条指令只能和 exec 或者 sh 指令配合使用，exec 和 sh 指令必须被提前配置好，否则 execfix 会失败。

exec 和 sh 扩展仅仅能够通过 snmpd.conf 文件被配置。他们不能通过 SNMP SET 来进行配置。

Extend [MIBOID] NAME PROG ARGS

与 exec 指令类似，但是有更多方面的改进。MIB 表（nsExtendConfigTable 等等）是按照参数 NAME 生成相关索引的，这样就不受指令在配置文件中出现顺序的影响了。这里有两个相关的结果表格 - 一个表格是 nsExtendOutput1Table，包含了退出状态，还有每条扩展指令的第一行输出和指令的完整输出（一个字符串），另一个表格(nsExtendOutput2Table)包含了全部的分隔成行的输出。

如果 MIBOID 被指定了，则配置和结果表格将会被指定以此 OID 为子树的根，其它的指令也会同样处理。这意味着几个不同的 extend 指令能够指定同样的 MIBOID 根，而不会发生冲突。

代理会为每条指令生成的“退出状态”和“输出”分别进行缓存，并且能够使用 nsCacheTable 对指令生成结果（还有配置的缓存操作）进行清除。

extendfix NAME PROG ARGS

注册一个在需要的时候可以调用的命令，这靠设置相应的 nsExtendRunType 来实现，设置的值为 run-command(3)。不像 execfix 指令，本条指令不必要与相应的 extend 指令配合使用，可以单独出现。

extend 和 extendfix 指令能够被动态的进行配置，使用 SNMP SET 对 NET-SNMP-EXTEND-MIB 进行操作来实现。

10:09 2010-6-3

MIB-Specific Extension Commands

第一组“功能扩展”调用系统命令，需要依赖 MIB（或者管理程序）才能对输出进行适当的处理。这可以使得信息的获取变得快捷简单，但是当处理某些特定的 MIB 对象时，由于需要服从 MIB 的结构，就无能为力了（反之亦然）。其它的扩展机制都与特定的 MIB 有关 - 它们是以"pass-through"开头的脚本。要想使用这种处理机制，代理需要在编译时加入对 ucd-snmp/pass 和 ucd-snmp/pass_persist 模块的支持（默认已包括）。

Pass [-p priority] MIBOID PROG

这条指令将把以指定的 MIBOID 为根的子树传递给指定的 PROG 命令。在此子树内对 OID 的 GET、GETNEXT 请求将会触发执行这条命令，类似于这样的调用：

PROG -g OID
PROG -n OID

PROG 命令应该返回三行数据到标准输出 stdout - 第一行是 OID，第二行是代表返回值的类型（其实是一个字符串，可能是：integer, gauge, counter, timeticks, ipaddress, objectid,string），还有第三行是返回值。

如果命令没有合适的返回值 - 比如指定的 OID 没有响应 GET 操作，或者对于 GETNEXT 操作并没有下一个 OID - 然后没有任何输出就退出了。这将会产生 SNMP 的 noSuchName 错误或者 noSuchInstance 异常。

注意：SNMPv2 的 counter64 类型和 noSuchObject 异常不被支持。

SET 操作将会让命令如下被调用：

PROG -s OID TYPE VALUE

这里，TYPE 表示 VALUE 的类型，具体和上面提到的类型相同（译:integer,gauge 等）。如果赋值成功，PROG 不生成任何输出而结束。如果有错误，那么会在标准输出上提示 not-writable 或者 wrong-type，代理也会生成相应的错误信息。

注意：其它的 SNMPv2 错误不被支持。

在其它的情况下，命令会随着执行的完成而结束。每一个请求（随每个请求有一个 varbind）将会触发一个单独的命令调用。

默认的注册优先级为 127。这可以使用-p 参数来进行指定，低优先值先于高优先值使用。

pass_persist [-p priority] MIBOID PROG

本条指令会把以 MIBOID 为根的子树的控制权传递给 PROG 命令。而且这条命令将会在第一条 snmp 请求完成之后继续运行，这样后继的 snmp 请求不再需要初始化的代价。

在初始化的时候，字符串"ping\n"将会被传递给 PROG 命令，并且它会在 stdout 标准输出上输出"PONG\n"。

对于 GET、GETNEXT 请求，PROG 将会从 stdin 接收到两行输入，包括"get"、"getnext"关键字，还有请求的 OID。PROG 会向 stdout 输出三行 - 包括 OID、其类型 TYPE 还有 OID 的值 - 这和上面的 pass 指令是一样的。如果 PROG 命令不能产生需要的结果，它会向 stdout 打印"NONE\n"（但是会继续运行）。

对于 SET 请求，PROG 将会从 stdin 接收到三行输入，包括命令本身（set）、请求的 OID、还有类型和值（这两项在一行）。如果赋值成功，PROG 命令会向 stdout 输出“DONE\n”。如果有错误发生，那么会输出下面之一到 stdout：**not-writable, wrong-type, wrong-length, wrong-value or inconsistent-value** ，代理将会发出相应的错误响应。在这种情况下，命令也会继续运行。

关于注册优先级也使用-p 参数来设定，和 pass 指令相同。

pass 和 pass_persist 扩展只能够通过 snmpd.conf 文件进行配置。他们不能通过 SNMP SET 请求来设置。

对嵌入 Perl 的支持

前面提及的扩展机制所使用的程序可以使用任何计算机语言编制 - 包括 perl，特别适合于制作扩展。然而 Net-SNMP 也包括对直接嵌入的 Perl 代码的支持（与 Apache 的 mod_perl 类似）。这允许代理直接解释 perl 脚本，因而避免了收到 SNMP 请求时“启动进程”和“初始化 perl 系统”的开销。

使用这种机制要求代理内建对于“嵌入式 perl”的支持，这不是默认的功能。必须在编译时在百编译配置脚本中指定“--enable-embedded-perl”参数。

如果启用了 perl 支持，下面的指令都可以被识别。

disablePerl true
　　这将关闭嵌入式 perl 支持（比如：安装 perl 出现了问题）。

perlInitFile FILE
　　如果存在指定的初始化文件，那么在第一个 perl 指令被分析前加载它。如果没有明确的被指定，

　　代理将会查找默认的初始化文件 c:/usr/share/snmp/snmp_perl.pl。

　　默认的初始化文件创建一个 Net-SNMP::agent 对象的实例 - 一个变量 $agent，它被用于注册基于 perl 的 MIB 处理例程。

perl EXPRESSION

评估给定的表达式。这通常会注册一个处理例程，当某个 OID 树收到 snmp 请求时：

```
perl use Data::Dumper;
perl sub myroutine  { print "got called: ",Dumper(@_),"\n"; }
perl $agent->register("mylink", ".1.3.6.1.8765", \&myroutine);
```

这个表达式可能是一个外部文件的内容：

```
perl "do /path/to/file.pl";
```

或者执行一个其它的基于 perl 的脚本。

动态加载模块

绝大多数 Net-SNMP 代理支持的 MIB 都是 C 语言编制的模块，当 net-snmp 生成时候就被编译并连接到代理的库中了。这些可执行模块也能被单独的编译并加载到正在运行的代理上。使用这种机制需要代理内建 ucd-snmp/dlmod 模块支持（默认已经包括）。

dlmod NAME PATH

这将从位于 PATH 的文件加载共享的模块（一个含绝对路径的文件名），并调用初始化例程"init_NAME"。

注意：如果指定的 PATH 不是一个完整的文件名，它将会被认为是相对于“c:/usr/lib/snmp/dlmod”而言的，并且会在文件的末尾加上“.so”后缀。

这项功能可以通过使用 SNMP SET 对 UCD-DLMOD-MIB 进行操作来配置。

代理支持

另一种扩展代理功能的机制是把 snmp 请求（或者所用变量）传递给另一个 SNMP 代理，这个代理可能运行在同一个主机上（另一个端口），也可能在一台远程主机上。这可以看作主代理把请求分发给一个远程代理，或者主代理充当一个请求代理。使用这种机制需要代理内建对 ucd-snmp/proxy 模块的支持（默认已经包括）。

proxy [-Cn CONTEXTNAME] [SNMPCMD_ARGS] HOST OID [REMOTEOID]

这条指令将会把任何接收到的对指定 OID 的 SNMP 请求转发到指定主机上。对于地址、端口的指定格式参见 snmpd（8）“LISTENING ADDRESSES”一节。

注意：代理整个 MIB 树，需要使用 OID ‘.1.3’（而不是 ‘.1’）。

SNMPCMD_ARGS 应该提供足够的版本和管理信息来生成一个有效的 SNMP 请求（请参见 snmpcmd(1)）。

注意：被代理的请求将不会使用原始请求的管理相关设置。

如果指定了 CONTEXTNAME，那么将会在本地代理上以指定的上下文注册一个代理委托。为同一个 OID 定义多个不同上下文的代理指令，这可以用于通过一个代理来对多个远程代理进行查询，这需要在收到的请求中指定相应的 SNMPv3 上下文（或者使用配置好的‘community’字符串 - 参见 com2sec 指令一节）。

指定 REMOID 参数将会把本地的 MIB 树映射成在远程主机的 REMOID 树下的一棵子树。

SMUX 子代理

Net-SNMP 代理支持通过 SMUX 协议（RFC 1227）和基于 SMUX 的子代理通讯（比如：gated，zebra 或者 quagga）。使用这种机制需要代理内建对 smux 模块的支持,此模块默认不包含，需要在编译 net-snmp 的时候在编译脚本中使用‘--with-mib-modules=smux’参数明确指定。

注意：这个扩展协议官方不推荐使用，而是使用 AgentX 来替代（见下文）。

smuxpeer OID PASS

这条指令将会为基于 SMUX 的处理过程注册一棵子树，使用 PASS 作为口令来授权使用。如果子代理（或者一个‘代理点’）连接到代理并注册了这棵子树，然后发出对子树中某些 OID 的请求，这将会被传递到相应的 SMUX 子代理来进行处理。

对 OSPF 路由守护进程来说，一个合适的定义可能类似如下：

```
smuxpeer .1.3.6.1.2.1.14 ospf_pass
```

smuxsocket <IPv4-address>

对 SMUX 节点义了一个 IPv4 地址来和 Net-SNMP 代理进行通讯。默认情况下监听所有的 IPv4 接口（“0.0.0.0”），除非在编译的时候指定了“--enable-local-smux”参数，这将使得 SMUX 节点只监听 127.0.0.1，SMUX 使用保留的 TCP 端口 199。

注意：Net-SNMP 代理只能作为一个 SMUX 主代理存在。它不能成为 SMUX 子代理。

AgentX 子代理

Net-SNMP 代理支持 AgentX 协议（RFC 2741）的两种模式 master 和 subagent。使用这种机制，需要代理内建对 agentx 模块的支持（默认已包含），并且启用（比如通过 snmpd.conf 文件）。

有两个指令与作为 AgentX 主代理运行有关：

master agentx

这条指令将启用 AgentX 功能，并让 snmp 代理开始监听传入的 AgentX 注册。这也能通过命令行激活，需要在命令行中加入‘-x’参数（用来指定监听的套接字）。

agentXPerms SOCKPERMS [DIRPERMS [USER|UID [GROUP|GID]]]

定义 AgentX Unix 域套接字的许可权和拥有权，还有这个套接字的父目录。SOCKPERMS 和 DIRPERMS 必须是八进制数字（参见 chmod(1)）。默认这个套接字将只允许子代理访问，这个子代理与 snmp 代理使用同样的 userid。

有一条指令与作为 AgentX 子代理运行有关：

agentXPingInterval NUM

这条指令指定当连接断开以后，子代理尝试连接到主 AgentX 的时间间隔。

剩下的指令与 AgentX 主代理和 AgentX 子代理都有关：

agentXSocket [<transport-specifier>:]<transport-address>[,...]

这条指令定义主代理监听的地址，或者子代理应该连接到的地址。默认是 Unix Domain socket "/var/agentx/master"。另一个通常的替代用法是 tcp:localhost:705。参见 snmpd(8)LISTENING ADDRESSES 一节获取关于 ip 地址格式的更多信息。

注意：指定 AgentX 套接字不会自动的启用 AgentX 功能（这和在命令行中使用‘-x’参数不同）。

agentXTimeout NUM

这个指令定义 AgentX 请求的超时时间（NUM 秒）。默认 1 秒。

agentXRetries NUM

这个指令定义 AgentX 请求的尝试次数。默认 5 次。

net-snmp 包含了 c 和 perl 的 API，你可以开发自己的 AgentX 子代理。

其它配置参数

override [-rw] OID TYPE VALUE

这条指令允许你使用一个不同的值（可能连类型也不同）覆盖一个指定的 OID。“-rw”允许使用 snmp SET 操作修改这个值。（注意如果你覆盖了某个原始的功能，这个功能就会丢失掉。因而 SET 操作除了在内部修改值以外不会操作 MIB 提供的原始功能。它只是模拟而已。）例子：

override sysDescr.0 octet_str "my own sysDescr"

这会把 sysDescr.0 设置为“my own sysDescr”，同样会让这个 oid 变成可以使用 SNMP SET 来修改（实际上对 MIB 规范来说这是非法的）。

注意当这样使用的时候必须十分的小心。例如，如果你视图覆盖在 ifTable 中的第三个网络接口的值，那么 ifTable 中的索引值就会发生变动，这最终会导致发生问题，并且你修改的值也不会在正确的地方出现。

有效的数据类型有：integer, uinteger, octet_str, object_id, counter, null (对于 gauge 类型,使用 "uinteger"; 对于 bit strings, 使用 "octet_str")。注意把一个对象设置为“null”会在它可访问的条件下删除掉它。如果某个对象的类型为 null，那么就不需要为它赋值。

将来会有更多的数据类型。

如果你正想找出各种 MIB 模块（说不定其中还有你自己写的），下面的内容会帮助你找到一些有用的调试信息。首先，请使用-D 参数来阅读 snmpd 手册。然后下面将要提到的 snmpd.conf 中的配置参数，再配合命令行的 -D，就能够生成很多有用的输出：

injectHandler HANDLER modulename

这将在 MIB 树中“modulename”影响到的那部分插入一个新的句柄。句柄的可用类型有：

stash_cache

缓存更低一层返回的信息。这极大的提升了代理的性能，能更好的降低由于缓存保留时间少于 30 秒的不便（缓存时间将来就可以进行配置了）。同时需要注意的是，这将加大 snmpd 对于内存的消耗。现在这只限于使用 table_iterator 支持的少数 mib 表格进行注册的句柄。想成功的如此使用，你需要确信在 table_iterator 点之前注册句柄，可以这样做：

injectHandler stash_cache NAME table_iterator

如果你想要一个表格如此，试试 walk 一下 nsModuleTable，使用 injected 或者不用。

debug 当-Dhelper:debug 标志被传递给 snmpd，就打印出大量的调试信息。

read_only 对于给定模块，强制关闭“写”支持。

serialize

如果一个模块没有正确的处理收到的多个请求（使用新的 5.0 版模块 API），这将强制模块每次仅仅接收一个请求。

bulk_to_next

如果注册一个模块用来处理 getbulk 请求，但是出于某些原因没能正确的实现它，这个模块将会在最终的模块接收到这些请求前，把所有的 getbulk 转换成 getnext。

dontLogTCPWrappersConnects

如果 snmpd 编译时包含了 TCP Wriapper 支持，它会记录下所有的到 snmp 代理的连接。这个参数禁用对这些 tcp 连接的日志记录。拒绝的连接仍然会被记录。

Figuring out module names

为了找出你可以注入的模块，请运行 snmpwalk 查看 nsModuleTable，这将会生成一个模块名字的列表。

内部数据表

table NAME
add_row NAME INDEX(ES) VALUE(S)

注意：

* Net-SNMP 代理可以重新加载配置文件，方法是使用 **snmpset** 设置

UCD-SNMP-MIB::versionUpdateConfig.0 (.1.3.6.1.4.1.2021.100.11.0)为 integer(1)，另一种方法是使用命令 **kill -HUP** 来发送相应的号给代理进程。

* 所有的可以使用“yes”作为值的指令都可以使用布尔值。这些指令会接收‘1’、‘yes’、‘true’来使能相应的功能，或者接收‘0’、‘no’、‘false’来禁用。默认情况下，这些功能都是关闭的，这些指令通常只用来启用这些功能。

配置文件实例

参见顶层目录的 EXAMPLE.CONF 文件，以便获取更详细的关于这些配置指令怎样使用的信息。

文件

c:/usr/etc/snmp/snmpd.conf

参考

snmpconf(1), snmpusm(1), snmp.conf(5), snmp_config(5), snmpd(8), EXAMPLE.conf, read_config(3).