NET-SNMP 开发指南

Hangzhou CVN

lvcl

2012-12-13

版本历史及修订说明

版本	日期	拟制人/修改人	版本修订说明	审核人
V1.0	2012-12-13	Lvcl	creat	
V1.1	2012-12-19	Lvcl	增加 64MIB 说明	

目录

1 前言

本文参考 SNMP 代码的版本为 NET-SNMP-5.5，目前公司 8000 系列，超维系列，74 低频系列均使用该版本 SNMP，本文所有范例代码出自超维系列产品，对于新接触公司产品的应用模块和新接触 SNMP 的开发工作具有一定的参考价值。

因本人自身水平有限，参考资料较为缺乏，可能出现一些遗漏和错误，恳请指正。

2 概述

SNMP 目前有 V1、V2、v2c、v3 这几个版本，公司的所有的目前的产品均使用 v2c 版本进行开发。

NET-SNMP-5.5 的代码量较大，但绝大部分是用来处理信息、进行数据通信保护加密等工作的。我们开发时主要新增和修改的文件有如下 4 个：

mib_module_inits.h----------用来注册新增的 rainbow 模块；

rainbow.c ----------接口实现主函数；

rainbow.h----------定义头文件；

laketunerainbow.txt/ PLC-LKT-7410-MIB.MIB----------MIB 文件，用于节点定义；

3 详细代码分析

下面对每一个文件的相关代码进行分析。

3.1 mib_module_inits.h

```
if (should_init("rainbow")) init_rainbow();
```

Rainbow 模块为公司私有模块，在 SNMP 启动后加载，用于新增私有 MIB 节点的调用工作。在 mib_module_inits.h 头文件中加载该模块。

3.2 rainbow.c

3.2.1 init_rainbow

此处模块初始化，新增加列表以后，要在此处增加初始化表文件函数。

除此之外，该函数中一般还需要增加重启和系统启动的告警消息，重启告警放在所有初始化表函数之前，系统重启告警放在最后。

3.2.2 初始化表函数

```
/** Initialize the cwCfgAuthCfgTable table by defining its contents and how it's structured */
void
initialize_table_cwCfgAuthTable(void)    //初始化表函数
{
    static oid      cwCfgAuthTable_oid[] =
    { 1, 3, 6, 1, 4, 1, 5959, 2, 1, 7, 1 };      //定义静态列表oid号
    size_t          cwCfgAuthTable_oid_len =
        OID_LENGTH(cwCfgAuthTable_oid);          //取OID号的长度
    netsnmp_handler_registration *reg;           //注册信息结构体，包含了注册名、根节点号、优先级等信息
    netsnmp_iterator_info *iinfo;                //迭代信息结构体
    netsnmp_table_registration_info *table_info; //节点信息结构体，包含节点数量，min、max宏序号等信息

    reg =
        netsnmp_create_handler_registration("cwCfgAuthTable",
        cwCfgAuthTable_handler,
        cwCfgAuthTable_oid,
        cwCfgAuthTable_oid_len,
        HANDLER_CAN_RWRITE);                 //列表信息注册函数，传入表名、节点号、节点列表、读写权限

    table_info = SNMP_MALLOC_TYPEDEF(netsnmp_table_registration_info); //为表申请地址空间
    netsnmp_table_helper_add_indexes(table_info, ASN_INTEGER,    /* index: cwCfgAuthDsID */
        0);                                  //增加表索引的函数
    table_info->min_column = COLUMN_CWCFGAHTUENABLE;     //最小可操作节点
    table_info->max_column = COLUMN_CWCFGAHTUPORT;       //最大可操作节点
```

```
    iinfo = SNMP_MALLOC_TYPEDEF(netsnmp_iterator_info); //申请迭代信息地址空间
    iinfo->get_first_data_point =
        cwCfgAuthTable_get_first_data_point;
    iinfo->get_next_data_point = cwCfgAuthTable_get_next_data_point;
    iinfo->table_reginfo = table_info;                  //迭代信息结构体项赋值

    netsnmp_register_table_iterator(reg, iinfo);        //注册迭代列表，使之运行

    /*
    * Initialise the contents of the table here
    */
    cwCfgAuthTable_createEntry(SNMP_AGENT_ID);     //产生链表，如果有多个索引，将放在get-first-data中产生
} ? end initialize_table_cwCfgAuthTable ?
```

该函数作用为指定列表的初始化工作，包括指定列表 OID 号，声明结构体，形成链表等工作。

增加新表时主要需要修改的项有对应的列表名，OID 号，min/max cloumn 的宏定义值，以及产生列表的函数。CreateEntry 函数的入参代表有几级索引，当列表为终端列表时，需要添加多级索引，则不在该函数中进行，而移至 get-first 函数中进行。

netsnmp_table_helper_add_indexes 函数中，列表有几级索引，就需要添加几个 ASN_INTEGER 入参。

3.2.3 列表节点结构体

```
/*
* Typical data structure for a row entry
*/
struct cwCfgAuthTable_entry {                   //列表结构体声明
    /*
    * Index values
    */
    long            cwCfgAuthDsID;              //索引

    /*
    * Column values
    */
    long            cwCfgAuthEnable;            //
    long            old_cwCfgAuthEnable;        //
    u_long          cwCfgAuthAddr;              //     具体节点信息，old_用来恢复原值
    u_long          old_cwCfgAuthAddr;          //
    long            cwCfgAuthPort;              //
    long            old_cwCfgAuthPort;          //

    /*
    * Illustrate using a simple linked list
    */
    int             valid;
    struct cwCfgAuthTable_entry *next;          //将其定义为循环链表
} ? end cwCfgAuthTable_entry ? ;

struct cwCfgAuthTable_entry *cwCfgAuthTable_head;    //申明链表头
```

该结构体定义了列表的详细节点信息，做开发时，重点要关注索引节点的个数，以及每个变量的类型。能够进行设置的节点需要有对应的 old_变量做备份用。

3.2.4 产生新链表函数

```
/*
* create a new row in the (unsorted) table
*/
struct cwCfgAuthTable_entry *
    cwCfgAuthTable_createEntry(long cwCfgAuthDsID)     //产生新链表的函数
{
    struct cwCfgAuthTable_entry *entry;

    entry = SNMP_MALLOC_TYPEDEF(struct cwCfgAuthTable_entry);    //申请列表的地址空间
    if (!entry)
        return NULL;    //未申请成功，退出

    entry->cwCfgAuthDsID = cwCfgAuthDsID;    //
    entry->next = cwCfgAuthTable_head;       //     产生链表
    cwCfgAuthTable_head = entry;             //
    return entry;
}
```

该函数用来产生一个新的链表，其参数须将全部索引节点加入，并且每一个索引节点都需要赋至 entry 中相应的位置。

3.2.5 移除链表函数

```
/*
* remove a row from the table
*/
void
cwCfgAuthTable_removeEntry(struct cwCfgAuthTable_entry *entry)   //移除链表函数
{
    struct cwCfgAuthTable_entry *ptr, *prev;

    if (!entry)
        return;                 /* Nothing to remove */  //空链表，直接退出

    for (ptr = cwCfgAuthTable_head, prev = NULL;         //链表搜索，清除全部
        ptr != NULL; prev = ptr, ptr = ptr->next) {
            if (ptr == entry)                            //已为最后一项，跳出
                break;
        }
        if (!ptr)                                        //无最后一项，返回
            return;                 /* Can't find it */

        if (prev == NULL)
            cwCfgAuthTable_head = ptr->next;             //目前位置为第一项，链接至下一项
        else
            prev->next = ptr->next;                      //链接至下一项

        SNMP_FREE(entry);                                //移除链表中的该项
} ? end cwCfgAuthTable_removeEntry ?
```

该函数用于取值出错时移除建立的链表，用递归的方法清除全部链表项，一般情况下除了修改列表名称相关的参数以外无需修改其他代码。

3.2.6 获取链表头项和下一项

```
/*
* Example iterator hook routines - using 'get_next' to do most of the work
*/
netsnmp_variable_list *
cwCfgAuthTable_get_first_data_point(void **my_loop_context,   //获取链表第一项的值并循环链表
                                    void **my_data_context,
                                    netsnmp_variable_list *
                                    put_index_data,
                                    netsnmp_iterator_info *mydata) //如果有多级索引，则在此处添加索引
{
    *my_loop_context = cwCfgAuthTable_head;
    return cwCfgAuthTable_get_next_data_point(my_loop_context,          //链表的下一项
        my_data_context,
        put_index_data,
        mydata);
}

netsnmp_variable_list *
cwCfgAuthTable_get_next_data_point(void **my_loop_context,
                                   void **my_data_context,
                                   netsnmp_variable_list *
                                   put_index_data,
                                   netsnmp_iterator_info *mydata)
{
    struct cwCfgAuthTable_entry *entry =
        (struct cwCfgAuthTable_entry *) *my_loop_context;     //声明链表项
    netsnmp_variable_list *idx = put_index_data;              //声明列表的索引

    if (entry) {
        snmp_set_var_typed_integer(idx, ASN_INTEGER,          //设置索引项，有几项加几项
            entry->cwCfgAuthDsID);
        idx = idx->next_variable;
        *my_data_context = (void *) entry;
        *my_loop_context = (void *) entry->next;              //链表下一项地址
        return put_index_data;
    } else {
        return NULL;
    }
} ? end cwCfgAuthTable_get_next_data_point ?
```

这两个函数用来获取链表中的参数的具体值，get_next_point 函数中，需要根据索引的级数来添加对应数量的 my_data_context 和 my_loop_context。

当有多级索引时，get_first_data 中需要增加获取具体级数的代码，如下图所示：

```
netsnmp_variable_list *
staStatTable_get_first_data_point(void **my_loop_context,       //多级索引列表范例
                                  void **my_data_context,
                                  netsnmp_variable_list * put_index_data,
                                  netsnmp_iterator_info *mydata)
{
    struct staStatTable_entry *entry = staStatTable_head;
    while(entry != NULL)
    {
        struct staStatTable_entry *entry_temp = entry->next;
        staStatTable_removeEntry(entry);
        entry = entry_temp;
    }

    int ret = 0;
    int i,j;
    l2mng_topo_info_t ti;                //声明拓扑信息结构体
    l2mng_send_param_t send_param;        //声明二层取值参数结构体

    memset(&send_param, 0, sizeof(l2mng_send_param_t));  //清零

    ret = l2mng_get_topo(&send_param, &ti);    //获取二层拓扑信息及其具体参数
    if (ret == 0)
    {
        for (i = 0; ti.node_list[i].id != 0 && i < L2MNG_MAX_NODES; i++) //循环获取每一个终端的信息
        {
            tdprintf("-0");
            if (ti.node_list[i].device_type == STA
                && ti.node_list[i].id < L2MNG_MAX_NODES)
            {
                for (j = 1;j <= 2;j++)                                  //如果是有效终端，则列表索引数+1
                {
                    staStatTable_createEntry(SNMP_AGENT_ID,ti.node_list[i].id,j);
                }
            }
        }
    }
    else
    {
        tdprintf("l2mng_get_topo() failed(%d)",ret); //二层拓扑获取失败报错
    }

    tdprintf("<");
```

该函数中增加了通过二层拓扑获取终端数量的代码，通过获取该数量，就可以知道二级索引中需要有几层节点，同理，当有多级索引时，需要取得每级索引下的节点层数，便可以通过 createEntry 函数产生完整的列表。

3.2.7 具体操作函数

```
/** handles requests for the cwCfgAuthTable table */
int
cwCfgAuthTable_handler(netsnmp_mib_handler *handler,
                       netsnmp_handler_registration *reginfo,   //分解、填写数据的函数
                       netsnmp_agent_request_info *reqinfo,
                       netsnmp_request_info *requests)
{
    int ret;
    netsnmp_request_info *request;              //定义需求报文信息参数指针
    netsnmp_table_request_info *table_info;     //定义列表信息参数指针
    struct cwCfgAuthTable_entry *table_entry;   //定义列表节点参数指针

    switch (reqinfo->mode)                      //根据mode区分实现功能
    {
        /*
        * Read-support (also covers GetNext requests)
        */
```

该函数中放置的对每个节点的具体操作信息，分为 4 个部分：

1. GET

```
    switch (reqinfo->mode)                      //根据mode区分实现功能
    {
        /*
        * Read-support (also covers GetNext requests)
        */
    case MODE_GET:                              //获取节点值
        for (request = requests; request; request = request->next) //循环获取每一个请求
        {
            table_entry = (struct cwCfgAuthTable_entry *)
                netsnmp_extract_iterator_context(request); //将结构体中的列表节点信息取出
            table_info = netsnmp_extract_table_info(request); //将结构体中的列表OID等信息取出

            if (!table_entry) {                             //如空，提示无此节点
                netsnmp_set_request_error(reqinfo, request,
                    SNMP_NOSUCHINSTANCE);
                continue;
            }

            /*此处填写获取值的具体实现*/
            /*可以将全部的取值函数放置在此处*/

            switch (table_info->colnum)
            {
            case COLUMN_CWCFGAHTUENABLE:                //根据宏判断取哪个节点的值
               snmp_set_var_typed_integer(request->requestvb, ASN_INTEGER,
                    table_entry->cwCfgAuthEnable);      //将值根据指定类型填入
                break;
            /*. . . . . */
            default:
                netsnmp_set_request_error(reqinfo, request,
                    SNMP_NOSUCHOBJECT);                 //无此节点
                break;
            }
        } ? end for request=requests;requ... ?
        break;
```

当 mode 为 MODE_GET 时进入此项，接口函数先将传入的结构体拆分，取出所需的 OID、colnum 等信息，然后根据具体的函数实现所获得的值，填入对应的节点中，完成取值过程。

需要注意的是，有时候因为一张列表中的所有节点的实现都是放在 switch (table_info->colnum)函数外面，会导致只需要一个节点而把所有节点的值去上来，当某些取值函数返回参数较慢时，会导致 SNMP 整体取值速率下降，此时可以将该节点的具体取值参数移动到对应的 case 项之内，这样就不会影响到取其他值的速度。

2. SET_RESERVE

```
    /*
    * Write-support
    */
case MODE_SET_RESERVE1:                 //设置节点值时对值的判断，判断类型和范围
    for (request = requests; request; request = request->next)
    {
        table_entry = (struct cwCfgAuthTable_entry *)
            netsnmp_extract_iterator_context(request);
        table_info = netsnmp_extract_table_info(request);
        if (!table_entry)
        {
            netsnmp_set_request_error(reqinfo, request,SNMP_ERR_GENERR);
            return SNMP_ERR_NOERROR;
        }
        switch (table_info->colnum)
        {
        case COLUMN_CWCFGAHTUENABLE:
            /*
            * or possibly 'netsnmp_check_vb_int_range'
            */
            ret = netsnmp_check_vb_truthvalue(request->requestvb);  //此处是对TURE/FALSE值的判断
            if (ret != SNMP_ERR_NOERROR) {
                netsnmp_set_request_error(reqinfo, request, ret);
                return SNMP_ERR_NOERROR;
            }
            break;
        /*. . . . .*/
        default:
            netsnmp_set_request_error(reqinfo, request,
                SNMP_ERR_NOTWRITABLE);
            return SNMP_ERR_NOERROR;
        }
    } ? end for request=requests;requ... ?
    break;
```

此 case 项用来效验 SET 时传入的值是否正确，判断函数主要有如下几种：

1. netsnmp_check_vb_truthvalue

   用来判断输入为 1 和 2 的值，常用在开关等节点功能上

2. netsnmp_check_vb_type_and_max_size

   用来判断输入为字符串的值，不能超过其定义长度

3 netsnmp_check_vb_uint_range

   用来判断输入为无符号型的数值，可设定参数的范围

4 netsnmp_check_vb_type

配置 IP 地址等特殊参数时用此函数

3. SET_ACTION & SET_UNDO

```
case MODE_SET_ACTION:          //设置节点参数
    for (request = requests; request; request = request->next)
    {
        table_entry = (struct cwCfgAuthTable_entry *)
            netsnmp_extract_iterator_context(request);
        table_info = netsnmp_extract_table_info(request);

        switch (table_info->colnum)
        {
        case COLUMN_CWCFGAHTUENABLE:
            table_entry->old_cwCfgAuthEnable =
                table_entry->cwCfgAuthEnable;      //将原有参数保存在old_中，以便取消设置时恢复
            table_entry->cwCfgAuthEnable =
                *request->requestvb->val.integer; //将请求报文中的值取出

            //此处填写具体的节点实现函数

            break;
        /*. . . . . */
        }
    }
    break;

case MODE_SET_UNDO:      //取消设置参数时，恢复原值
    for (request = requests; request; request = request->next)
    {
        table_entry = (struct cwCfgAuthTable_entry *)
            netsnmp_extract_iterator_context(request);
        table_info = netsnmp_extract_table_info(request);

        switch (table_info->colnum)
        {
        case COLUMN_CWCFGAHTUENABLE:
            table_entry->cwCfgAuthEnable =
                table_entry->old_cwCfgAuthEnable;
            table_entry->old_cwCfgAuthEnable = 0;
            break;
        /*. . . . . */
        }
    }
    break;
```

此处两项为节点设置项，设置的具体实现函数放置在对应的宏定义的项中，同时 undo 项也需要添加，以便在设置取消的时候能够取回原值。

3.3 rainbow.h

```
void            init_rainbow(void);
void            initialize_table_cwCfgAuthTable(void);
Netsnmp_Node_Handler cwCfgAuthTable_handler;
Netsnmp_First_Data_Point cwCfgAuthTable_get_first_data_point;
Netsnmp_Next_Data_Point cwCfgAuthTable_get_next_data_point;
struct cwCfgAuthTable_entry ;
struct cwCfgAuthTable_entry *cwCfgAuthTable_createEntry(long cwCfgAuthDsID);
void cwCfgAuthTable_removeEntry(struct cwCfgAuthTable_entry *entry);
/*
 * column number definitions for table cwCfgAuthTable
 */
#define COLUMN_CWCFGAHTUDSID        1
#define COLUMN_CWCFGAHTUENABLE      2
#define COLUMN_CWCFGAHTUADDR        3
#define COLUMN_CWCFGAHTUPORT        4
```

该头文件中声明了 rainbow.c 中用到的几个函数，每当添加一张新表时，需要在该头文件中增加对应声明。

增减节点时，宏定义的值也需要做相应的修改。

3.4 Laketunerainbow.txt/ PLC-LKT-7410-MIB.MIB

该文件主要用于编译后形成列表节点索引文件，配合 SNMP 自带的 MIB 即可形成一张完整的列表。下面将整个文件分为几个部分进行描述。

```
LaketuneRainbow DEFINITIONS ::= BEGIN
    IMPORTS
        MacAddress, TruthValue, DateAndTime, DisplayString
            FROM SNMPv2-TC
        MODULE-IDENTITY, OBJECT-TYPE, IpAddress, Integer32, Unsigned32, NOTIFICATION-TYPE
            FROM SNMPv2-SMI;
```

此部分为 MIB 文件的头，定义了 MIB 库的名称，能够使用的类型，以及使用哪个版本的 SNMP 等信息。

```
org OBJECT IDENTIFIER
    ::=  {  iso  3  }

dod OBJECT IDENTIFIER
    ::=  {  org  6  }

internet    OBJECT IDENTIFIER
    ::=  {  dod  1  }

private OBJECT IDENTIFIER
    ::=  {  internet  4  }

enterprises OBJECT IDENTIFIER
    ::=  {  private  1  }

laketune    OBJECT IDENTIFIER
    ::=  {  enterprises  5959  }

central-office  OBJECT IDENTIFIER
    ::=  {  enterprises  17409  }

trap    OBJECT IDENTIFIER
    ::=  {  central-office  1  }
```

此列表为索引列表，我们的私有 MIB 表挂载在 enterprises 的节点下，为子节点 5959，低频上为 17409。

```
cwCfgNwTable    OBJECT-TYPE
    SYNTAX      SEQUENCE  OF  CwCfgNwEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION "Description"
    ::=  { cwCfgNw  1 }

cwCfgNwEntry    OBJECT-TYPE
    SYNTAX      CwCfgNwEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION "Row Description"
    INDEX       {  cwCfgNwDsID  }
    ::=  { cwCfgNwTable 1 }
```

此为列表的索引头，在 laketune 的根目录下挂载两级目录，然后下接一张完整的 Entry 表。

```
CwCfgNwEntry  ::=  SEQUENCE {
    cwCfgNwDsID  Integer32,
    cwCfgNwIpAddr  IpAddress,
    cwCfgNwNetMask  IpAddress,
    cwCfgNwTelnetTmt  Unsigned32,
    cwCfgNwFtpSvr  IpAddress,
    cwCfgNwFtpUser  DisplayString,
    cwCfgNwFtpPwd  DisplayString,
    cwCfgNwEmac  DisplayString
    }
```

此为整张列表的定义，要根据每个节点不同的数据类型定义正确的类型，否则可能导致取/赋值失败。

```
cwCfgNwDsID OBJECT-TYPE
    SYNTAX          Integer32 (1)
    MAX-ACCESS      not-accessible
    STATUS          current
    DESCRIPTION     "ID of Distributer"
    ::=  {  cwCfgNwEntry  1  }
```

列表中的索引节点，不可读写，但也不可省略定义。

```
cwCfgNwIpAddr   OBJECT-TYPE
    SYNTAX          IpAddress
    MAX-ACCESS      read-write
    STATUS          current
    DESCRIPTION     "IP address of distribute"
    DEFVAL          { 'C0A8DFA'h }
    ::=  {  cwCfgNwEntry  2  }
```

列表中的可读/取节点，DEFVAL 中可以设置默认值，既输入为空时默认下发/读取上来的值。

3.5 64 MIB 开发相关说明

64 低频的 SNMP 使用版本为 5.4.2.1，基本架构和 5.5 版本一致，但和 74、高频、超威不同的是，64 的每一张 table 都是单独初始化的，在 mib_module_inits.h 中并未注册统一的初始化函数，并且相关代码分为多个文件分开放置，下面将对每一部分进行阐述。

1. 表/节点的实现（类同 rainbow.c 拆分）

64 中分为两种节点定义类型，单一节点定义以及表定义。

单一节点定义：

```
init_modEoCCBATAuthSettingGroup(void)
{
    static oid      modEoCCABTDhcpUeOption82_oid[] =
        { 1, 3, 6, 1, 4, 1, 5960, 2, 4, 1, 8, 1 };
    static oid      modEoCCBATDhcpSnooping_oid[] =
        { 1, 3, 6, 1, 4, 1, 5960, 2, 4, 1, 8, 2 };
    static oid      modEoCCBATAuthOnOffState_oid[] =
        { 1, 3, 6, 1, 4, 1, 5960, 2, 4, 1, 8, 3 };
    static oid      modEoCCBATAuthHostName_oid[] =
        { 1, 3, 6, 1, 4, 1, 5960, 2, 4, 1, 8, 4 };
    static oid      modEoCCBATAuthIpAddr1_oid[] =
        { 1, 3, 6, 1, 4, 1, 5960, 2, 4, 1, 8, 5 };
    static oid      modEoCCBATAuthIpAddr2_oid[] =
        { 1, 3, 6, 1, 4, 1, 5960, 2, 4, 1, 8, 6 };
    static oid      modEoCCBATAuthIpAddr3_oid[] =
        { 1, 3, 6, 1, 4, 1, 5960, 2, 4, 1, 8, 7 };
```

初始化函数后分别静态定义每一个节点的 OID 号。

```
netsnmp_register_scalar(netsnmp_create_handler_registration    //注册节点函数
                        ("modEoCCBATDhcpSnooping",             //节点名
                         handle_modEoCCBATDhcpSnooping,        //节点具体实现函数
                         modEoCCBATDhcpSnooping_oid,           //节点OID号
                         OID_LENGTH(modEoCCBATDhcpSnooping_oid), //OID长度
                         HANDLER_CAN_RWRITE));                 //节点读写权限
```

以上为每一个节点的注册函数，以及调用节点的具体实现。

在 handle_modEoCCBATDhcpSnooping 中的节点具体实现和前述 handle 函数基本一致，开发时参考 74/高频/超维即可。

表定义：

```
void
init_modEoCCBATCardTable(void)
{
    /*
     * here we initialize all the tables we're planning on supporting
     */
    initialize_table_modEoCCBATCardTable();
}
```

将初始化函数外面又包了一层。

在初始化表函数中，架构和 rainbow.c 中的表实现函数基本一致，下面列出不同的那部分代码，以作参考。

```
void* modEoCCBATCardTable_make_data_context(void *loop_context, netsnmp_iterator_info *iinfo)
{append_tag_snmp_log
    struct modEoCCBATCardTable_entry *loopctx = loop_context;
    struct modEoCCBATCardTable_entry *datactx = SNMP_MALLOC_TYPEDEF(struct modEoCCBATCardTable_entry);

    if (!datactx)
        return NULL;
    memcpy(datactx, loopctx, sizeof(struct modEoCCBATCardTable_entry));
    return datactx;
}

void modEoCCBATCardTable_free_data_context(void *data, netsnmp_iterator_info *iinfo)
{append_tag_snmp_log
    free(data);
}

void modEoCCBATCardTable_free_loop_context_at_end(void *loopctx, netsnmp_iterator_info *iinfo)
{append_tag_snmp_log
    free(loopctx);
}
```

这三个函数是用来打印存储调试信息所用的，如果无此需求，可以省略。

此外，头文件和 MIB 表的定义文件和前述 74/高频/超威的基本一致，参考上述文档即可。

4 附录

4.1 SNMP 的 5 种协议数据单元

SNMP 规定了 5 种协议数据单元 PDU（也就是 SNMP 报文），用来在管理进程和代理之间的交换。

- get-request 操作：从代理进程处提取一个或多个参数值
- get-next-request 操作：从代理进程处提取紧跟当前参数值的下一个参数值
- set-request 操作：设置代理进程的一个或多个参数值
- 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，它是前面三种操作的响应操作。
- 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 种操作是由管理进程向代理进程发出的，后面的 2 个操作是代理进程发给管理进程的，为了简化起见，前面 3 个操作今后叫做 get、get-next 和 set 操作。图 4 描述了 SNMP 的这 5 种报文操作。请注意，在代理进程端是用熟知端口 161 俩接收 get 或 set 报文，而在管理进程端是用熟知端口 162 来接收 trap 报文。

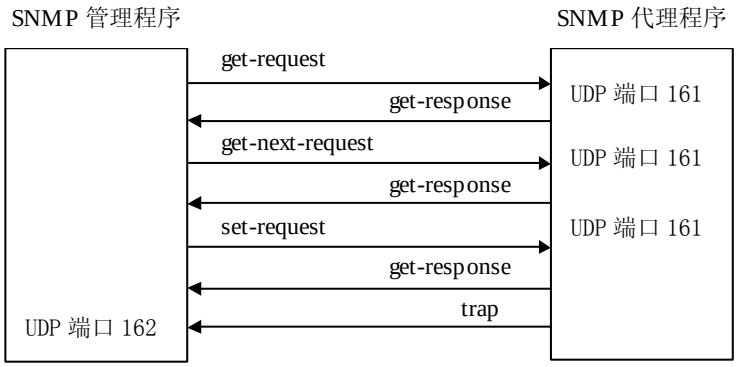

图 4 SNMP 的 5 种报文操作

图 5 是封装成 UDP 数据报的 5 种操作的 SNMP 报文格式。可见一个 SNMP 报文共有三个部分组成，即公共 SNMP 首部、get/set 首部 trap 首部、变量绑定。

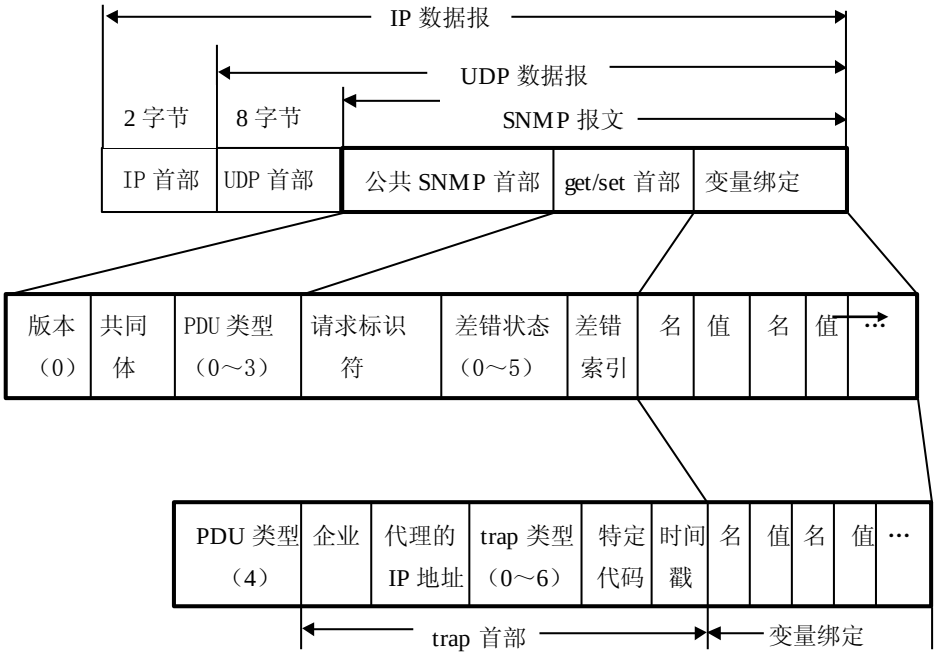

图 5 SNMP 报文格式

（1）公共 SNMP 首部

共三个字段：

- 版本

写入版本字段的是版本号减 1，对于 SNMP（即 SNMPV1）则应写入 0。

- 共同体（community）

共同体就是一个字符串，作为管理进程和代理进程之间的明文口令，常用的是 6 个字符“public”。

- PDU 类型

根据 PDU 的类型，填入 0～4 中的一个数字，其对应关系如表 2 所示意图。

表 2 PDU 类型

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

（2）get/set 首部

- 请求标识符(request ID)

这是由管理进程设置的一个整数值。代理进程在发送 get-response 报文时也要返回此请求标识符。管理进程可同时向许多代理发出 get 报文，这些报文都使用 UDP 传送，先发送

的有可能后到达。设置了请求标识符可使管理进程能够识别返回的响应报文对于哪一个请求报文。

- 差错状态（error status）

由代理进程回答时填入 0～5 中的一个数字，见表 3 的描述。

表 3 差错状态描述

差错状态	名字	说明
0	noError	一切正常
1	tooBig	代理无法将回答装入到一个 SNMP 报文之中
2	noSuchName	操作指明了一个不存在的变量
3	badValue	一个 set 操作指明了一个无效值或无效语法
4	readOnly	管理进程试图修改一个只读变量
5	genErr	某些其他的差错

- 差错索引(error index)

当出现 noSuchName、badValue 或 readOnly 的差错时，由代理进程在回答时设置的一个整数，它指明有差错的变量在变量列表中的偏移。

（3）trap 首部

- 企业（enterprise）

填入 trap 报文的网络设备的对象标识符。此对象标识符肯定是在图 3 的对象命名树上的 enterprise 结点{1.3.6.1.4.1}下面的一棵子树上。

- trap 类型

此字段正式的名称是 generic-trap，共分为表 4 中的 7 种。

表 4　trap 类型描述

trap 类型	名字	说明
0	coldStart	代理进行了初始化
1	warmStart	代理进行了重新初始化
2	linkDown	一个接口从工作状态变为故障状态
3	linkUp	一个接口从故障状态变为工作状态
4	authenticationFailure	从 SNMP 管理进程接收到具有一个无效共同体的报文
5	egpNeighborLoss	一个 EGP 相邻路由器变为故障状态
6	enterpriseSpecific	代理自定义的事件，需要用后面的“特定代码”来指明

当使用上述类型 2、3、5 时，在报文后面变量部分的第一个变量应标识响应的接口。

- 特定代码(specific-code)

指明代理自定义的时间（若 trap 类型为 6），否则为 0。

- 时间戳(timestamp)

指明自代理进程初始化到 trap 报告的事件发生所经历的时间，单位为 10ms。例如时间戳为 1908 表明在代理初始化后 1908ms 发生了该时间。

（4）变量绑定(variable-bindings)

指明一个或多个变量的名和对应的值。在 get 或 get-next 报文中，变量的值应忽略。

4.2 管理信息库 MIB

管理信息库 MIB 指明了网络元素所维持的变量(即能够被管理进程查询和设置的信息)。MIB 给出了一个网络中所有可能的被管理对象的集合的数据结构。SNMP 的管理信息库采用和域名系统 DNS 相似的树型结构，它的根在最上面，根没有名字。图 3 画的是管理信息库的一部分，它又称为对象命名（object naming tree）。

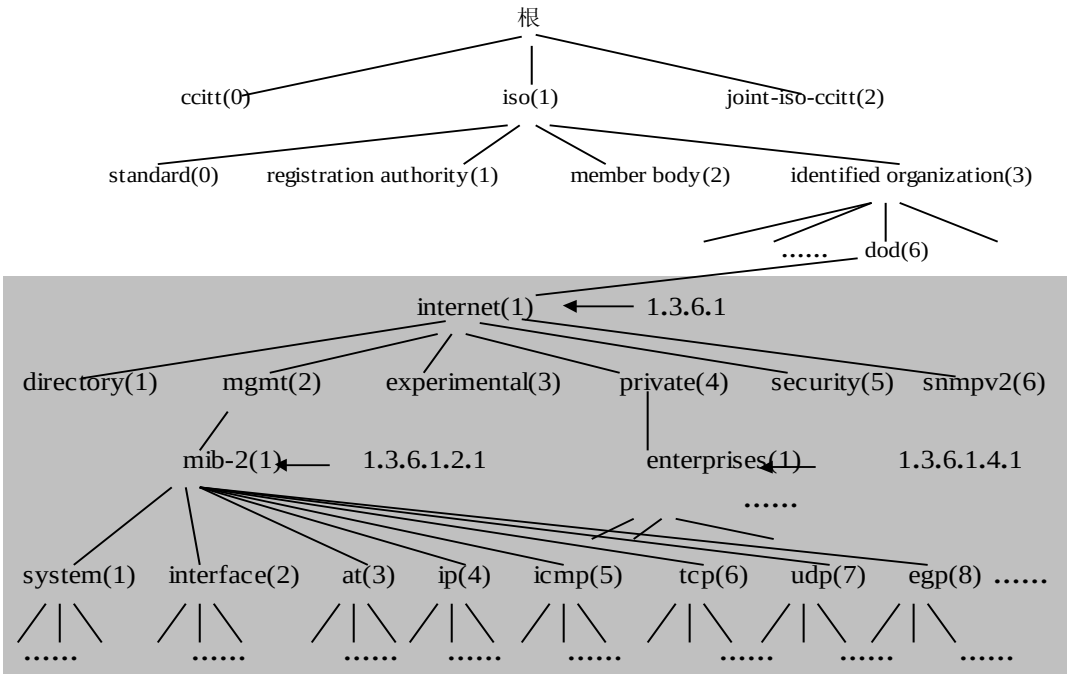

图 3 管理信息库的对象命名举例

对象命名树的顶级对象有三个，即 ISO、ITU-T 和这两个组织的联合体。在 ISO 的下面有 4 个结点，其中的饿一个（标号 3）是被标识的组织。在其下面有一个美国国防部（Department of Defense）的子树（标号是 6），再下面就是 Internet（标号是 1）。在只讨论 Internet 中的对象时，可只画出 Internet 以下的子树（图中带阴影的虚线方框），并在 Internet 结点旁边标注上{1.3.6.1}即可。

在 Internet 结点下面的第二个结点是 mgmt（管理），标号是 2。再下面是管理信息库，原先的结点名是 mib。1991 年定义了新的版本 MIB-II，故结点名现改为 mib-2，其标识为{1.3.6.1.2.1}，或{Internet(1) .2.1}。这种标识为对象标识符。

最初的结点 mib 将其所管理的信息分为 8 个类别，见表 1。现在 de　mib-2 所包含的信息类别已超过 40 个。

表 1 最初的结点 mib 管理的信息类别

类别	标号	所包含的信息
system	(1)	主机或路由器的操作系统
interfaces	(2)	各种网络接口及它们的测定通信量
address translation	(3)	地址转换（例如 ARP 映射）
ip	(4)	Internet 软件（IP 分组统计）
icmp	(5)	ICMP 软件（已收到 ICMP 消息的统计）
tcp	(6)	TCP 软件（算法、参数和统计）
udp	(7)	UDP 软件（UDP 通信量统计）
egp	(8)	EGP 软件（外部网关协议通信量统计）

应当指出，MIB 的定义与具体的网络管理协议无关，这对于厂商和用户都有利。厂商可以在产品（如路由器）中包含 SNMP 代理软件，并保证在定义新的 MIB 项目后该软件仍遵守标准。用户可以使用同一网络管理客户软件来管理具有不同版本的 MIB 的多个路由器。当然，一个没有新的 MIB 项目的路由器不能提供这些项目的信息。

这里要提一下 MIB 中的对象{1.3.6.1.4.1}，即 enterprises（企业），其所属结点数已超过 3000。例如 IBM 为{1.3.6.1.4.1.2}，Cisco 为{1.3.6.1.4.1.9}，Novell 为{1.3.6.1.4.1.23}等。世界上任何一个公司、学校只要用电子邮件发往 iana-mib@isi.edu 进行申请即可获得一个结点名。这样各厂家就可以定义自己的产品的被管理对象名，使它能用 SNMP 进行管理。

5 参考文献

1. 中兴通讯 SNMP 培训教材
2. SNMP 简单网络管理协议　　电子工业出版社-李明江著 2007-05