

MIB 文件编写.....	2
C (h) 源文件自动生成.....	2
工具.....	2
iterate 类型 mib 节点生成: .....	2
scalar 类型 mib 节点生成: .....	2
修改自动生成的 C 文件.....	3
为什么要修改.....	3
修改方式.....	3
xxx_get_next_data_point 函数的修改 .....	4
initialize_table_xxx 的修改.....	6
新增 reset_xxx_head 函数.....	7
新增 xxx_get_data_point 函数 .....	7
原始文件和修改后文件, 自己可做详细比较.....	8
编译配置.....	8
添加节点的具体.....	9

Snmp 网管在运营商网络中是一种非常重要的管理方式, 实现对整个网络进行统一规划和统一管理, 也可对单一网元进行分散管理。简单而言, snmp 方式在设备侧实现的是对设备内部所有的数据结构进行读写操作, 每个数据结构对应有独立的 mib 节点, 需要一个 snmp agent 实体和相关的 mib 处理流程。

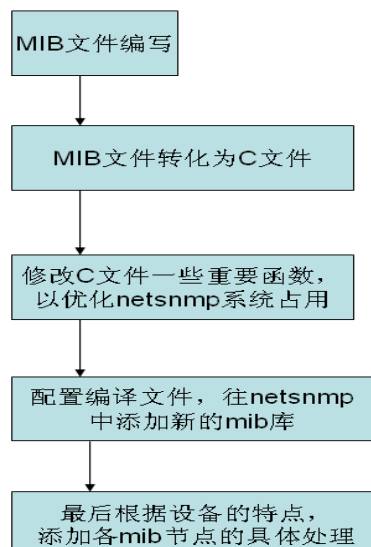
NETSNMP 是 snmp agent 开源项目, 在我们的设备中使用此项目来完成 snmp 功能的开



C:\Documents and Settings\chenfeng

发。关于 NETSNMP 和一般 SNMP 的知识介绍参考

新 MIB 的添加流程如下图所示



## MIB 文件编写

mib 文件是 snmp client 和 agent 之间通信的一个通用语言，有自己的语法规则，client 和 agent 都遵照这个语法规则进行理解，以达成管理中的一致性。

需要说明的是，由于 Mib 节点可分为 iterate 和 scalar 两类，所以在自动生成源码时两类节点的处理是分别产生的。

## C (h) 源文件自动生成

### 工具


mib2c (在 linux 环境下安装 net-snmp 后即可)。mib2c 工具有多种配置文件，使用不同的配置文件可以生成不同风格的源码。在这里，对于 iterate 类型节点使用的配置文件是 mib2c.iterate.conf；对于 scalar 类型节点采用 mib2c.scalar.conf。

### iterate 类型 mib 节点生成：

以 BRIDGE-MIB.mib 为例

- 1、把 mib 文件拷贝到 linux 服务器，路径/usr/local/share/snmp/mibs/，重命名为 txt。cp BRIDGE-MIB.mib /usr/local/share/snmp/mibs/ BRIDGE-MIB.txt
- 2、执行命令 env MIBS="BRIDGE-MIB" mib2c -c mib2c.iterate.conf dot1dBridge，即可在

当前目录生成两个文件 dot1dBridgeIterate.c (  E:\code\F803-snmp-for-ZTE ) 和

dot1dBridgeIterate.h (  E:\code\F803-snmp-for-ZTE )

### scalar 类型 mib 节点生成：

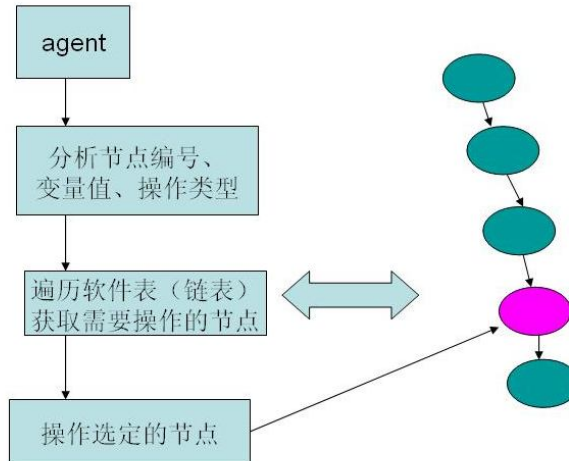
以 BRIDGE-MIB.mib 为例

- 1、把 mib 文件拷贝到 linux 服务器，路径/usr/local/share/snmp/mibs/，重命名为 txt。cp BRIDGE-MIB.mib /usr/local/share/snmp/mibs/ BRIDGE-MIB.txt
- 2、执行命令 env MIBS="BRIDGE-MIB" mib2c -c mib2c.scalar.conf dot1dBridge，即可在当前目录生成两个文件 dot1dBridgeScaler.c 和 dot1dBridgeScaler.h

## 修改自动生成的 C 文件

### 为什么要修改

Netsnmp 对表项的操作模型如下图：



这种方式存在的问题有：

A、内存的浪费。对于 iterate 类型的变量，netsnmp 系统运行需要保存所有表项，如端口配置表，netsnmp 会保存一份，在我们的 onu\_manage 中也保存了一份。其他所有的表项也都是双份，导致内存的浪费。

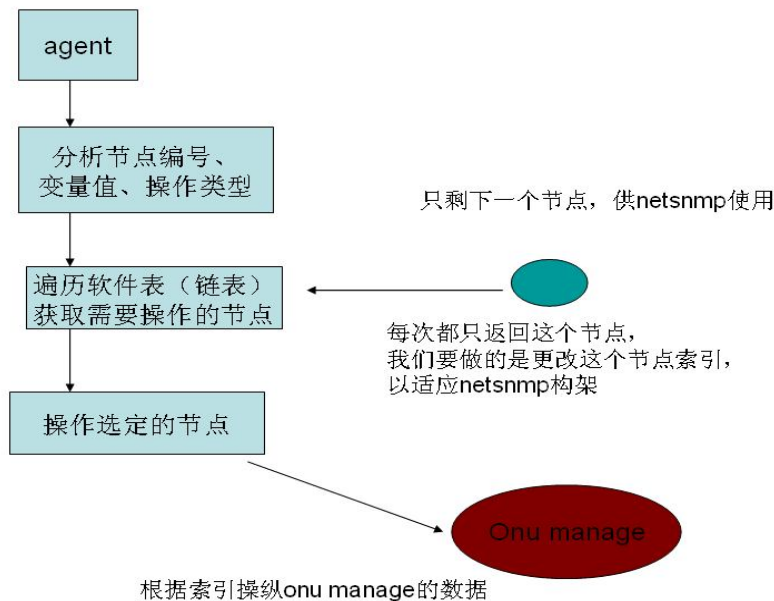
B、效率低。Netsnmp 获取表项的任一变量，都需要遍历所有表项。如获取 VLAN 表中 VLAN 50 的端口成员，netsnmp 会遍历 VLAN 1 到 VLAN 4096 之后才获取 VLAN 50 的端口成员。假如存在 4096 个 VLAN，那么遍历完所有的 VLAN 所需要的操作是  $4096 \times 4096$ 。这种遍历方式导致运行效率很低。实际上我们应该只需要 4096 即可。

C、实时性差。在 netsnmp 内部保存着软件表项的时候，变量的值是直接从 netsnmp 中的软件表中获取的。对于我们的网络设备，往往会通过其他方式（OAM、WEB、CLI）来更改变量。如果 netsnmp 不及时更改内部软件表的值，那就使用的是更改前的值，实时性很差。

D、语法错误。通过 mib2c 工具生成的 C 文件，存在少数的语法错误，也是需要修改才能编译通过的。

### 修改方式

修改后的模型：



对于 netsnmp 的核心算法和流程，我们不改动，那么我们就通过外部的修改来适应 netsnmp 核心算法和流程。

A、对于 scalar 类型变量而生成的 C，只需要修改语法错误就可以了。关于语法部分，这里不做说明。直接编译，根据报错进行修改即可。

B、iterate 类型变量生成的 C。修改函数 xxx\_get\_next\_data\_point、initialize\_table\_xxx。新增加函数 reset\_xxx\_head, xxx\_get\_data\_point。这里”xxx”代表你所要修改的表项名称。如，RFC1213-MIB 中，我们需要修改 iftable 的函数为：ifTable\_get\_next\_data\_point、initialize\_table\_ifTable，新增函数为 reset\_iftable\_head、ifTable\_get\_data\_point。

## xxx\_get\_next\_data\_point 函数的修改

最重要的函数是 xxx\_get\_next\_data\_point 的修改。以 ifTable\_get\_next\_data\_point 为例，原始函数为：

```

netsnmp_variable_list *
ifTable_get_next_data_point(void **my_loop_context,
                           void **my_data_context,
                           netsnmp_variable_list * put_index_data,
                           netsnmp_iterator_info *mydata)
{
    struct ifTable_entry *entry =
        (struct ifTable_entry *) *my_loop_context;
    netsnmp_variable_list *idx = put_index_data;

    if(entry) {
        snmp_set_var_typed_integer(idx, ASN_INTEGER, entry->ifIndex);
        idx = idx->next_variable;
        *my_data_context = (void *) entry;
    }
}
  
```

`*my_loop_context = (void *) entry->next; /*因为 netsnmp 保存了所有软件表，这里就返回下一个，我们要做的修改就是怎么避免使用链表。*/`

```
        return put_index_data;
    } else {
        return NULL;
    }
}
```

修改后的函数：

```
netsnmp_variable_list *
ifTable_get_next_data_point(void **my_loop_context,
                             void **my_data_context,
                             netsnmp_variable_list * put_index_data,
                             netsnmp_iterator_info *mydata)
```

```
{
    struct ifTable_entry *entry =
        (struct ifTable_entry *) *my_loop_context;
    netsnmp_variable_list *idx = put_index_data;
    int i;

    if (entry) {
```

`/*`

这里 for 循环是为了找下一个有效表项，对于 iftable，有效表项的索引一定是连续的，但其他有些表项就不一定连续了，如 VLAN，这些表项就需要在 for 循环中进行判断，查找下一个有效表项。

找到有效表项后，这里仅仅是把索引返回给 netsnmp 内核，然后再调用 dot1dPortGmrpTable\_handler 函数的 get 操作

因此在 dot1dPortGmrpTable\_handler 函数中需要根据索引再编程

`*/`

```
    for(i=entry->ifIndex+1;i<=ZTE_F803_MAX_ETH_PORT_NUM;i++)/*
```

`entry->ifIndex` 为传入的索引，我们要找下一个，对于 iftable 而言，下一个就是 `entry->ifIndex+1`，也就是当前的 `i`\*/

```
{
```

```
    entry->ifIndex=i; //更改单个节点 entry 中的索引值，以供 netsnmp 使用
    snmp_set_var_typed_integer(idx, ASN_INTEGER, entry->ifIndex);
```

`*my_data_context = (void *) entry; /* 告诉 netsnmp 这里的 entry 就是下一个数据*/`

`*my_loop_context = (void *) NULL; /* 告诉 netsnmp 没有变量了，避免 netsnmp 循环查找*/`

```
    ifTable_head->ifIndex=i;
    return put_index_data;
}
```

```

//如果已经找完了，则恢复最初值
if(i>ZTE_F803_MAX_ETH_PORT_NUM)
{
    *my_data_context = (void *) NULL;
    *my_loop_context = (void *) NULL;

    ifTable_head->ifIndex=ZTE_F803_FIRST_ETH_PORT_ID-1;
    snmp_set_var_typed_integer(idx, ASN_INTEGER, 0);
    return NULL;
}
}
else {

    return NULL;
}
}

```

## initialize\_table\_xxx 的修改

以 initialize\_table\_ifTable 为例，函数中蓝色字体为我们新增的内容。

/\*\* Initialize the ifTable table by defining its contents and how it's structured \*/

void

initialize\_table\_ifTable(void)

```

{
    static oid      ifTable_oid[] = { 1, 3, 6, 1, 2, 1, 2, 2 };
    size_t          ifTable_oid_len = OID_LENGTH(ifTable_oid);
    netsnmp_handler_registration *reg;
    netsnmp_iterator_info *iinfo;
    netsnmp_table_registration_info *table_info;

```

项 //这里需要创建一个表项，供 netsnmp 内核算法使用，否则 netsnmp 不能处理此表

ifTable\_createEntry(ZTE\_F803\_FIRST\_ETH\_PORT\_ID-1); //创建起始 ID 为 0，作为  
无效表项

```

reg = netsnmp_create_handler_registration("ifTable", ifTable_handler,
                                          ifTable_oid, ifTable_oid_len,
                                          HANDLER_CAN_RWRITE);

```

```

table_info = SNMP_MALLOC_TYPEDEF(netsnmp_table_registration_info);
netsnmp_table_helper_add_indexes(table_info, ASN_INTEGER, /* index: ifIndex */
                                  0);
table_info->min_column = COLUMN_IFINDEX;
table_info->max_column = COLUMN_IFSPECIFIC;

```

```

        iinfo = SNMP_MALLOC_TYPEDEF(netsnmp_iterator_info);
        iinfo->get_first_data_point = ifTable_get_first_data_point;
        iinfo->get_next_data_point = ifTable_get_next_data_point;
#ifdef TW_NETSNMP_EXTEND
        iinfo->get_data_point = ifTable_get_data_point;
        iinfo->header_reset = reset_iftable_head;
#endif
        iinfo->table_reginfo = table_info;

        netsnmp_register_table_iterator(reg, iinfo);

        /*
         * Initialise the contents of the table here
         */
    }

```

## 新增 reset\_xxx\_head 函数

```

void reset_iftable_head(int para1 ,char *p)
{
    ifTable_head->ifIndex=ZTE_F803_FIRST_ETH_PORT_ID-1;
}

```

## 新增 xxx\_get\_data\_point 函数

```

netsnmp_variable_list *
ifTable_get_data_point(void **my_loop_context,
                        void **my_data_context,
                        netsnmp_variable_list *
                        put_index_data,
                        netsnmp_iterator_info *mydata)
{
    struct ifTable_entry *entry=NULL;
    netsnmp_variable_list *idx = put_index_data;
    int index;

    if(put_index_data->val.integer)
    {
        index=*(put_index_data->val.integer);
    }
    /* 如果 index 对应的表现有效 */
    if(index<=ZTE_F803_MAX_ETH_PORT_NUM)

```

```

{

    *my_loop_context = ifTable_head;
    entry =
        (struct ifTable_entry *) *my_loop_context;

    entry->ifIndex=index;
    snmp_set_var_typed_integer(idx, ASN_INTEGER, entry->ifIndex);

    *my_data_context = (void *) entry;
    *my_loop_context = (void *) NULL;

    return put_index_data;
}
//无效则返回空指针
else
{

    *my_data_context = (void *) NULL;
    *my_loop_context = (void *) NULL;
    return NULL;
}

}

```

原始文件和修改后文件，自己可做详细比较



## 编译配置

我们增加 iftable 的源码后，需要做如下修改：

- 1、新建目录，\Default\_Apps\snmp\net-snmp-5.4.2\agent\mibgroup\tw-iftable-mib
- 2、拷贝源码 iftable.c 和 iftable.h 到目录 tw-iftable-mib 下
- 3、修改\Default\_Apps\snmp\Makefile（新增红色内容即可）  
 --with-mib-modules="**tw-iftable-mib** mibII bridge-mib pBridge qBridge zxAnInterfaceMib" \
- 4、在\snmp\net-snmp-5.4.2\agent\mibgroup 目录下增加文件 tw-iftable-mib.h。内容为：



```
/*  
 * module to include the modules  
 */  
config_require(tw-iftable-mib/ifNumber); //说明需要编译 tw-iftable-mib/ifNumber.c 文件  
config_require(tw-iftable-mib/ifTable); //说明需要编译 tw-iftable-mib/ifTable.c 文件
```

tw-iftable-mib.h

- 5、在工程根目录下运行命令，`make config`。工程就会根据新的配置生成 `netsnmp` 新的 `makefile`，自动包含 `iftale.c` 和 `ifnumber.c`。
- 6、再运行 `make snmp=1` 就会编译 `netsnmp`
- 7、最后运行 `make snmp=1 install`；`./image_tools`

## 添加节点的具体

在修改后的 C 文件中添加具体处理即可，可以参考 `iftable.c` 的[原始文件和修改后文件差异](#)。