杭州云速科技

cfg 编程指南

配置项编程

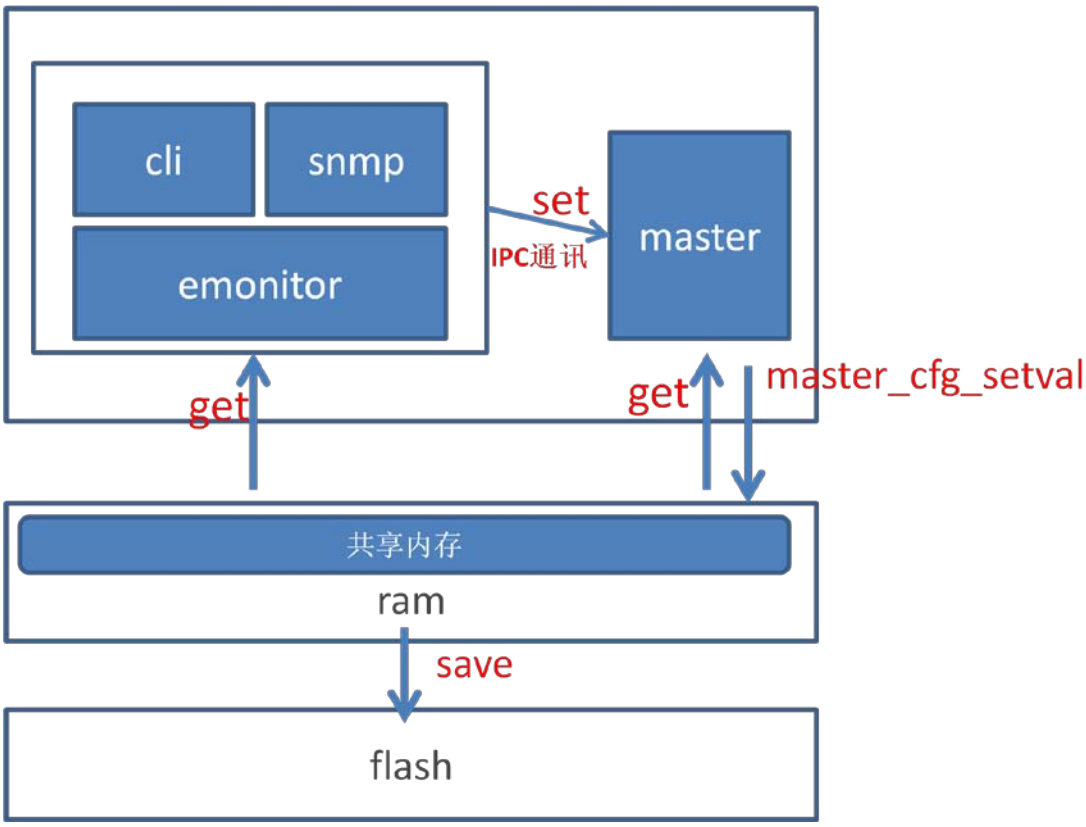
tengkaien

2014-7-22

cfg 配置编程过程中的个人总结，个人理解可能有不当之处，请自辨并指正。文档得到导师罗武通审阅并提出修改意见，配置编程学习实践过程中导师、胡孝全、苏揆提供了许多帮助，在此表示感谢。

目　录

1. 配置读写流程



上图中的箭头指示的是配置数据的流向，从图上可以看出，配置的读取可以在各个映射了 master 进程的共享内存的进程中进行，而配置的设置只能在 MASTER 进程中进行。平台给出了读取和设置配置的统一接口：

```
//读取接口，各进程均可调用
int cfg_getval(int ifindex,unsigned int oid,void *val,void* default_val,unsigned retlen);

//设置接口，只有 master 可调用
int master_cfg_setval(int ifindex,unsigned int oid,void *val)
```

基于以上前提，在编写代码之前，我们可以得出大概的思路：配置的读取只要在进程中映射 master 进程创建的共享内存，然后调用读配置接口就可以了，而写配置操作由于要经过 MASTER 进程，因而需要进程间通讯的协助，将我们要写入的配置数据传送到 MASTER 进程由 MASTER 进程调用 master_cfg_setval()函数代我们写入配置。

2. 配置项标识符 cfgOid

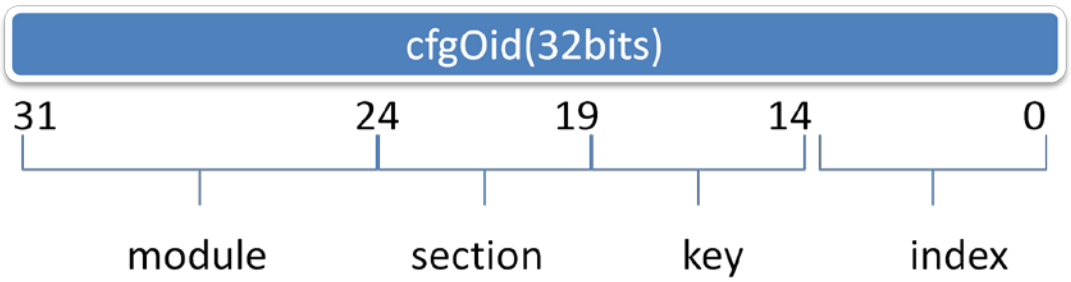
每一个配置项对应一个对象标识符 cfgOid,该 cfgOid 是一个 32 位的整型，用于唯一标示配置项。cfgOid 字段定义如下：

index 域: bits 0~13 非表项配置，此域为 0

key 域: bits 14~18，此域为具体的 key 值左移 14bits 后产生

section 域:bits 19~23，此域为具体 sectionc 值左移位 19bits 后产生

module 域 :bits 24~31,此域为具体 module 值(模块枚举)左移位 24bits 后产生



最终的 cfgOid 为以上各个域左移位后相或得到，由于不同的配置项传入的 key 值不一样，最终不同的配置项会得到不同的 cfgOid，这个 cfgOid 唯一标示该配置项。

3. 编程步骤

3.1 添加模块枚举

首先，在 lw_config_oid.h 的枚举类型 enum tagLW_MID_DEF 中添加一个成员，此成员其实对应的是 module 域的值，用于后面通过移位操作产生同一 module 域的不同配置项(cfgOid)，如下以添加光接收机模块的配置项为例：

```
00007: enum tagLW_MID_DEF
00008: {
00009:     CONFIG_MID_BASE=1,
00010:     CONFIG_MID_SYSTEM,
00011:     CONFIG_MID_NTP,
00012:     CONFIG_MID_WEB,
00013:     CONFIG_MID_TELNET,
00014:     CONFIG_MID_USER,
00015:     CONFIG_MID_ETHPORT,
00016:     CONFIG_MID_MIRROR,
00017:     CONFIG_MID_L2QOS,
00018:     CONFIG_MID_VLAN,
00019:     CONFIG_MID_MVLAN,
00020:     CONFIG_MID_LOOP,
00062:     CONFIG_MID_GLOBAL_RFDOWN,
00063:     CONFIG_MID_GLOBAL_RFDOWN_ND,//add by huxiaoquan 2014-05-28
00064:     CONFIG_MID_CATVCOM,//add by luowutong 2014/6/30
00065:     CONFIG_MID_MAX=255
```

可以看到该枚举最大项为 255，恰好是一个 module 域(8bits)表示的最大范围。

3.2 定义各个配置项的 cfgOid

接下来通过该枚举成员定义具体的 cfgOid，如下：

```
00634: /* begin: add by tengkaien 2014/7/1 */
00635: #define CONFIG_CATVCOM_MODULE                CONFIG_CMO_DEF(CONFIG_MID_CATVCOM
00636: #define CONFIG_CATVCOM_(key)                 CONFIG_CMO_DEF(CONFIG_MID_CATVCOM
00637: #define CONFIG_CATVCOM_SECTION               CONFIG_CATVCOM_(0)
00638: #define CONFIG_CATVCOM_TEMPERATURE_HI        CONFIG_CATVCOM_(1)
00639: #define CONFIG_CATVCOM_TEMPERATURE_LO        CONFIG_CATVCOM_(2)
00640: #define CONFIG_CATVCOM_VALTAGE_HI            CONFIG_CATVCOM_(3)
00641: #define CONFIG_CATVCOM_VALTAGE_LO            CONFIG_CATVCOM_(4)
00642: #define CONFIG_CATVCOM_OPTPOWER_HI           CONFIG_CATVCOM_(5)
00643: #define CONFIG_CATVCOM_OPTPOWER_LO           CONFIG_CATVCOM_(6)
00644: #define CONFIG_CATVCOM_OUTPUTLEVEL_HI        CONFIG_CATVCOM_(7)
00645: #define CONFIG_CATVCOM_OUTPUTLEVEL_LO        CONFIG_CATVCOM_(8)
00646:
00647: /* end: add by tengkaien 2014/7/1 */
```

跟进 CONFIG_CMO_DEF()可以发现其是对 CONFIG_MID_CATVCOM 这个枚举成员向左移位 24bits 后取值，这个值最终赋给 CONFIG_CATVCOM_MODULE，接下来定义一个操作宏，该宏传入一个 key 值，该值用于移位后产生 key 域的值，而该宏里面 section 域通过 1 移位产生，最终将各个域相或，便可得出一个唯一的配置项，即 cfgOid。

3.3 cfgOid 注册

接着在宏 CONFIG_PLAT_OIDTABLE 注册生成的 cfgOid：

```
00650: /***register info **************/
00651: #define CONFIG_PLAT_OIDTABLE \
00652: {CONFIG_SYSTEM, "system", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00653: {CONFIG_SYSTEM_SECTION, "system_info", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00654: {CONFIG_SYSTEM_NAME, "sys_systemname", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00655: {CONFIG_SUPER_PASSWORD, "super_password", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00656: {CONFIG_LAN_MAC_ADDR, "sys_mac", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00657: {CONFIG_NTP, "ntp", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00658: {CONFIG_NTP_SECTION, "ntp_info", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00659: {CONFIG_NTP_ZONE, "ntp_time_zone", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_INT},\
00660: {CONFIG_NTP_SERVER, "ntp_time_server", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00661: {CONFIG_NTP_DAYLIGHT, "ntp_daylight", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
```

```
00982: /* begin: add by tengkaien 2014/7/1 */\
00983: {CONFIG_CATVCOM_MODULE, "catvcom_cfg", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00984: {CONFIG_CATVCOM_SECTION,"catvcom_cfg_info",0,0,0,CFG_OID_TYPE_SINGLE,NULL,CFG_TYPE_STR},\
00985: {CONFIG_CATVCOM_TEMPERATURE_HI, "catv_temperature_upper_limit", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_
00986: {CONFIG_CATVCOM_TEMPERATURE_LO, "catv_temperature_lower_limit", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_
00987: {CONFIG_CATVCOM_VALTAGE_HI, "catv_valtage_upper_limit", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00988: {CONFIG_CATVCOM_VALTAGE_LO, "catv_valtage_lower_limit", 0,0,0, CFG_OID_TYPE_SINGLE, NULL, CFG_TYPE_STR},\
00989: {CONFIG_CATVCOM_OPTPOWER_HI,"catv_optpower_upper_limit",0,0,0,CFG_OID_TYPE_SINGLE,NULL,CFG_TYPE_STR},\
00990: {CONFIG_CATVCOM_OPTPOWER_LO,"catv_optpower_lower_limit",0,0,0,CFG_OID_TYPE_SINGLE,NULL,CFG_TYPE_STR},\
00991: {CONFIG_CATVCOM_OUTPUTLEVEL_HI,"catv_outputlevel_upper_limit",0,0,0,CFG_OID_TYPE_SINGLE,NULL,CFG_TYPE_STR},
00992: {CONFIG_CATVCOM_OUTPUTLEVEL_LO,"catv_outputlevel_lower_limit",0,0,0,CFG_OID_TYPE_SINGLE,NULL,CFG_TYPE_STR}
00993: /* end: add by tengkaien 2014/7/1 */
```

该过程其实是对 struct tagCFG_OID_REGINFO_S 进行初始化，上面产生的 cfgid 会被赋值到该结构体中，每产生一个 cfgid，便初始化一个 struct tagCFG_OID_REGINFO_S 结构体，最终宏 CONFIG_PLAT_OIDTABLE 会被包含进 struct tagCFG_OID_REGINFO_S 结构体类型的数组定义中，从而才真正实现结构体和结构体数组的初始化，如下：

```
00053: /*oid table */
00054: static CFG_OID_REGINFO_S  g_cfg_oid_tbl[]=
00055: {
00056:     CONFIG_PLAT_OIDTABLE
00057: };
```

struct tagCFG_OID_REGINFO_S 的定义如下：

```
00090: typedef struct  tagCFG_OID_REGINFO_S
00091: {
00092:     unsigned int  uiCmoid;
00093:     char  name[256];
00094:     int iMax;/*最大值*/
00095:     int iMin;/*最小值*/
00096:     int iRestore;/*能否被恢复出厂,0表示可以*/
00097:     CFG_OID_TYPE_E enNode_type ;/*NODE_TYPE_SINGLE表示非实例,NODE_TYPE_TABLE表示实例*/
00098:     int ( *CHECK_FUN_PF)(unsigned int uiCmoid,void * val);/*检查处理*/
00099:     int type;
00100: }CFG_OID_REGINFO_S;
```

上图 enNode_type 项注释似乎不合理，这项如果是 NODE_TYPE_SINGLE 表示非表项节点的配置 ID，如果是 NODE_TYPE_TABLE 表示的是表项节点的配置 ID（个人理解）。

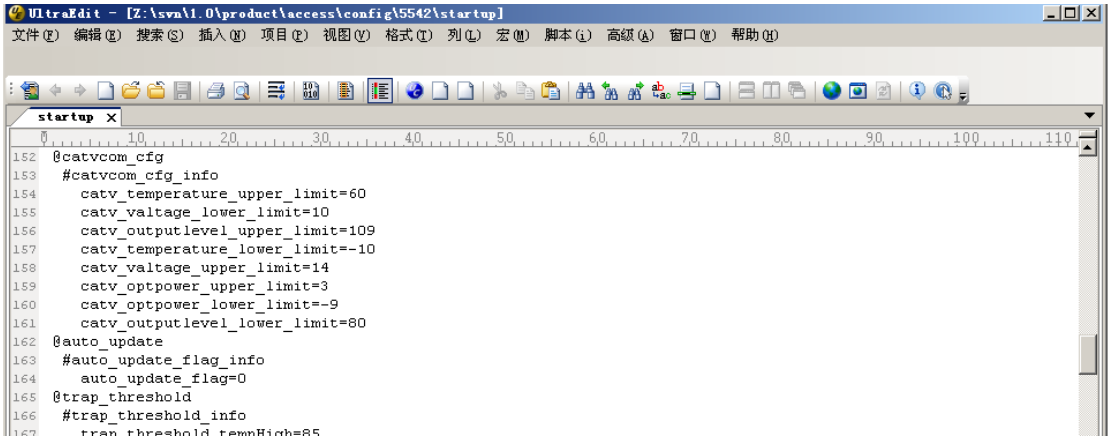
3.4 Startup 文件修改

Lw_config_oid.h 中添加了新的配置项后，对应的出场配置文件也需要添加，出场配置文件在 Product/xxx/config/yyy/startup 中

Xxx 为：产品线名

Yyy 为：产品名

比如在/1.0/product/access/config/5542 下的 strtup 中添加的光接收机模块配置项如下：



注意打开 startup 文件时使用 UE 打开编辑，否则容易出错。如果需要添加在特定接口下面，则找到对应的接口添加。无指定接口则添加在全局模式下（即 root 下面）。

模块名以@开头，段或者表项名以#开头

这些名称要与 cfgid 注册时的名称一致

如果是表象节点，则段名=注册的 Section 名称+编号，编号从 1 开始，且注册时 enNode_type 的值为 NODE_TYPE_TABLE，而非 NODE_TYPE_SINGLE。表项节点在 startup 文件中的形式如下：

```
    mac_collection_period=30
@user
 #vty_1
   user_vty_password=cvn
   user_vty_password_type=cleartext
   user_vty_timeout=300
   user_vty_name=enable
   user_vty_authmode=none
 #vty_2
   user_vty_password=cvn
   user_vty_password_type=cleartext
   user_vty_timeout=300
   user_vty_name=root
   user_vty_authmode=password
 #vty_3
   user_vty_password=cvn
   user_vty_password_type=cleartext
   user_vty_timeout=300
   user_vty_name=root
   user_vty_authmode=password
```

3.5 读配置

读取配置项代码的编写比较简单，大概分两步进行。

3.5.1 映射 master 进程共享内存

如果读配置发生在 cli 等已经映射了 master 进程共享内存的进程中，此步骤可以省略，直接调用 cfg_getval()进行读配置即可，如果是新写的程序，则此步骤不可省。可以在程序中添加如下代码：

```
00555:    Cfginit(SLAVER);
```

在 master 进程中，该函数的调用传入的参数是 MASTER。在该函数内完成共享内存的映射过程（如果是 master 进程，即传入 MASTER，会有更多操作）。

3.5.2 调用 cfg_getval()读配置

int cfg_getval(int ifindex,unsigned int oid,void *val,void* default_val,unsigned retlen)函数各个参数的含义如下：

Ifindex：接口索引，根据配置存在的位置去指定

Oid ：要读取的配置的 cfgOid

Val ：用于接收配置项值的缓冲指针

Default：读取不到配置时返回的默认值指针

Retlen ：接收配置值 buffer 的大小

注意 startup 文件中配置项的存储为字符串类型，因为返回的也是字符串。

下面给出一个读取光机模块参数阀值的实例：

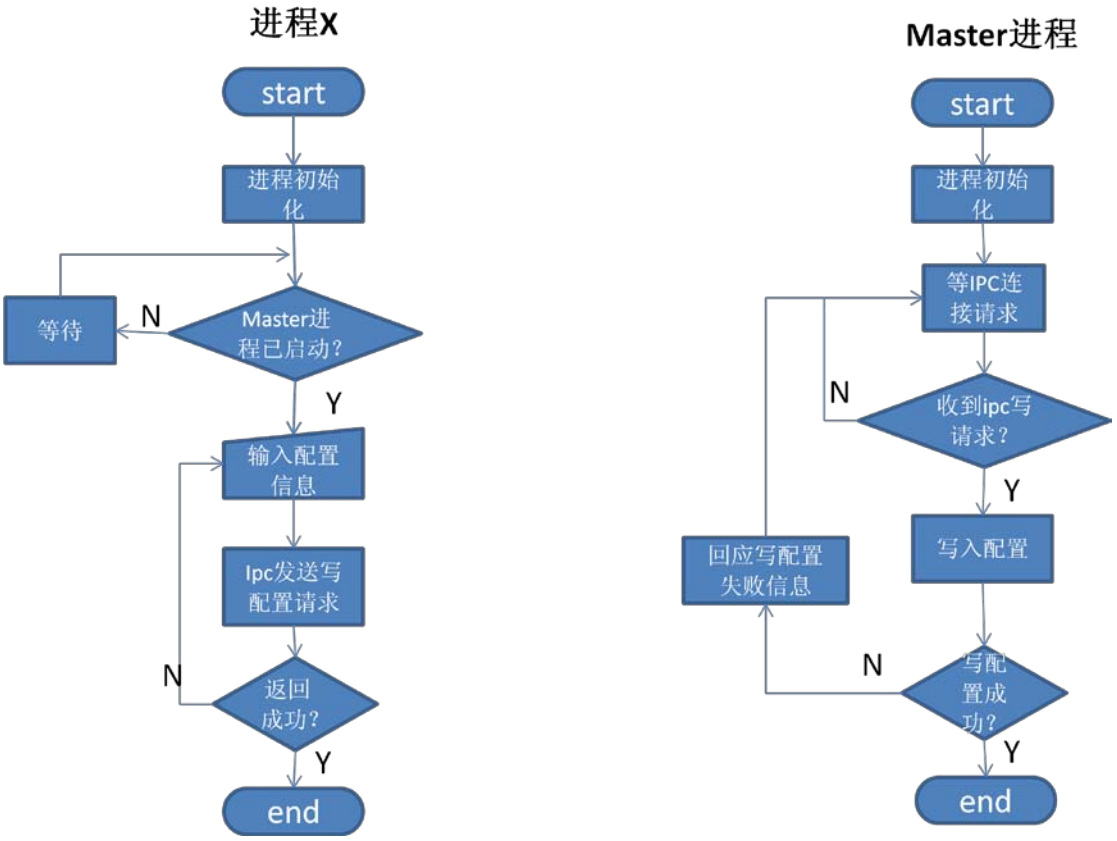
```
00500: INT CATV_ComThresholdInfoGet(CATV_THR_INFO *pstCatvThrInfo)
00501: {
00502:     INT iRet = OPT_FAILED;
00503:     CHAR szCatvThrCfg[BUF_SIZE_32] = {0};
00504:     if (NULL == pstCatvThrInfo)
00505:     {
00506:         return OPT_FAILED;
00507:     }
00508:
00509:     memset(&szCatvThrCfg, 0, sizeof(szCatvThrCfg));
00510:     iRet = cfg_getval(IF_ROOT_IFINDEX, CONFIG_CATVCOM_TEMPERATURE_HI,
00511:         szCatvThrCfg, "", sizeof(szCatvThrCfg));
00512:
00513:     if (iRet < 0)
00514:     {
00515:         pstCatvThrInfo->iTempHi = CATVCOM_TEMPTHR_HI_DEF;
00516:     }
00517:     else
00518:     {
00519:         sscanf(szCatvThrCfg, "%d", &pstCatvThrInfo->iTempHi);
00520:     }
00521:
00522:     memset(&szCatvThrCfg, 0, sizeof(szCatvThrCfg));
00523:     iRet = cfg_getval(IF_ROOT_IFINDEX, CONFIG_CATVCOM_TEMPERATURE_LO,
00524:         szCatvThrCfg, "", sizeof(szCatvThrCfg));
00525:
00526:     if (iRet < 0)
00527:     {
00528:         pstCatvThrInfo->iTempLo = CATVCOM_TEMPTHR_LO_DEF;
00529:     }
00530:     else
00531:     {
00532:         sscanf(szCatvThrCfg, "%d", &pstCatvThrInfo->iTempLo);
00533:     }
00534:
```

3.6 写配置

写配置由于只能在 MASTER 进程中调用 int master_cfg_setval()函数写入，所以在其他进程中如果要写配置项，则需通过进程间通讯 IPC 实现。int master_cfg_setval()各参数含义与 int cfg_getval()类似，区别是 val 是待写入内容的缓存指针。

3.6.1 整体流程

首先写一个 IPC 接口函数，用于向 MASTER 进程发送 IPC 通讯请求，该请求携带此次 ipc_request 的类型和待写入的配置信息,MASTER 进程收到该请求后，会依据请求帧中的 ipc_request 类型，调用相应的处理函数。大体的过程如下图：



接下来开始以上过程的具体编码实现。

3.6.2 添加 ipc_request 类型枚举，定义帧结构

首先在 ipc_ protocol.h 中添加一个 IPC_REQUEST 类型，如下：

```
00037: typedef enum {
00038:     IPC_PING = 0,
00039:     IPC_CFG_ACCESS,
00040:     IPC_CFG_BACKUP,
00041:     IPC_GET_USER_INFO,
00042:     IPC_SYSMON_CMD,
00043:     IPC_NETWORK_INFO,
00044:     IPC_GET_CNU_INFO,
00045:     IPC_GET_CNU_MIB,
00046:     IPC_GET_CNU_LINK_STATS,
00047:     IPC_GET_MME_STAT,
00048:     IPC_GET_VLAN_CONFIG,
00049:     IPC_GET_IF_CONFIG,
00050:     IPC_SERVICE_TEMPLATE,
00051:     IPC_SUPPORTED_DEVICE,
00052:     IPC_CLT_CABLE_PARAM,
```

```
00117:     IPC_DVN_UPTFTP,
00118:     /* begin: add by tengkaien 2014/7/1 */
00119:     IPC_CATVCOM_MODULE
00120:     /* end: add by tengkaien 2014/7/1 */
00121: }
00122: ipc_request_type_t;
```

接着定义携带配置信息的请求帧结构，以及用于 MASTER 进程回应的 ack 结构：

```
01357: /* begin: add by tengkaien 2014/7/1 */
01358: typedef struct __packed{
01359:     ipc_request_hdr_t hdr;
01360:     uint8_t apply_option;
01361:     int op;
01362:     int set_or_get;
01363:     char catv_temperature_upper_limit[BUF_SIZE_32];
01364:     char catv_temperature_lower_limit[BUF_SIZE_32];
01365:     char catv_valtage_upper_limit[BUF_SIZE_32];
01366:     char catv_valtage_lower_limit[BUF_SIZE_32];
01367:     char catv_optPower_upper_limit[BUF_SIZE_32];
01368:     char catv_optPower_lower_limit[BUF_SIZE_32];
01369:     char catv_outputLevel_upper_limit[BUF_SIZE_32];
01370:     char catv_outputLevel_lower_limit[BUF_SIZE_32];
01371: }
01372: ipc_catvcom_set_req_t;
01373:
01374: typedef struct __packed{
01375:     ipc_acknowledge_hdr_t hdr;
01376:     char catv_temperature_upper_limit[BUF_SIZE_32];
01377:     char catv_temperature_lower_limit[BUF_SIZE_32];
01378:     char catv_valtage_upper_limit[BUF_SIZE_32];
01379:     char catv_valtage_lower_limit[BUF_SIZE_32];
01380:     char catv_optPower_upper_limit[BUF_SIZE_32];
01381:     char catv_optPower_lower_limit[BUF_SIZE_32];
01382:     char catv_outputLevel_upper_limit[BUF_SIZE_32];
01383:     char catv_outputLevel_lower_limit[BUF_SIZE_32];
01384: }
01385: ipc_catvcom_set_ack_t;
01386:
```

上图_packed 结构中 hdr 结构成员用于携带 ipc_request 类型等信息，apply_option 用于定义相应请求的优先级，其他成员自己定义，依具体情况增删成员。

3.6.3 定义 ipc 通讯接口函数

定义了请求帧和应答帧的结构，接下来就可以定义填充请求帧信息并发送请求帧的 ipc 接口函数了，该接口函数一般定义在 ipc_func.c 中，该文件中定义的接口函数可以供其他进程调用。以下给出接口函数实例：

```
01616: /* begin: add by tengkaien 2014/7/1 */
01617: int ipc_catvcom_cfg(int ipc_fd, int get_or_set, int op, cur_catvcom_t *pcatvcomValue)
01618: {
01619:     int ret=0;
01620:     ipc_catvcom_set_req_t *req;
01621:     ipc_catvcom_set_ack_t *pack;
01622:     req=NULL;
01623:     pack=NULL;
01624:     req = (ipc_catvcom_set_req_t *)malloc(sizeof(ipc_catvcom_set_req_t));
01625:     DBG_ASSERT(req, "fail to allocate memory!");
01626:     if(NULL==req) return NULL;
01627:     req->apply_option = IPC_APPLY_NOW;
01628:     req->op=op;
01629:     req->set_or_get = get_or_set;
01630:     req->hdr.ipc_type = IPC_CATVCOM_MODULE;
01631:     req->hdr.msg_len =sizeof(ipc_catvcom_set_req_t) - sizeof(ipc_request_hdr_t);
01632:
01633:     if (get_or_set == CATVCOM_CFG_SET)
01634:     {
01635:         memcpy(req->catv_temperature_upper_limit,pcatvcomValue->catv_temperature_upper_limit,sizeo
01636:         memcpy(req->catv_temperature_lower_limit,pcatvcomValue->catv_temperature_lower_limit,sizeo
01637:         memcpy(req->catv_valtage_upper_limit,pcatvcomValue->catv_valtage_upper_limit,sizeof(req->c
01638:         memcpy(req->catv_valtage_lower_limit,pcatvcomValue->catv_valtage_lower_limit,sizeof(req->c
01639:         memcpy(req->catv_optPower_upper_limit,pcatvcomValue->catv_optPower_upper_limit,sizeof(req-
01640:         memcpy(req->catv_optPower_lower_limit,pcatvcomValue->catv_optPower_lower_limit,sizeof(req-
01641:         memcpy(req->catv_outputLevel_upper_limit,pcatvcomValue->catv_outputLevel_upper_limit,sizeo
01642:         memcpy(req->catv_outputLevel_lower_limit,pcatvcomValue->catv_outputLevel_lower_limit,sizeo
01643:
01644:
01645:
01646:     }
01647:     ipc_send_request(ipc_fd, req);
01648:     free(req);
01649:     req=NULL;
01650:
01651:     IPC_RECEIVE(pack, ipc_dvm_upftp_ack_t, ipc_fd);
```

该接口函数构造 ipc 请求帧，请求帧中携带此次 ipc 请求类型和我们希望写

入的配置项信息， MASTER 进程依据 ipc 请求类型调用处理函数。

3.6.4 定义 ipc 请求处理函数

在 ipc_packet.c 中写 MASTER 进程对 IPC 请求的处理函数，在接收到其他进程的 IPC 请求后，MASTER 进程首先从请求帧中获取此次请求的 ipc_request 类型，即发起此次 ipc 请求的进程填充到 hdr 的 ipc_type 成员的值，依据该值，MASTER 进程选择相应的处理函数，如下：

```
04545:      switch(nreq->ipc_type) {
04546:      case IPC_SYSMON_CMD:
04547:          ipc_sysmon_cmd(ipc_entry->fd, (ipc_sysmon_req_t *)nreq);
04548:          break;
04549:      #if 0
04550:      case IPC_NETWORK_INFO:
04551:          ipc_network_info(ipc_entry->fd, (ipc_network_info_req_t *)nreq);
04552:          break;
04553:
04554:      case IPC_GET_MME_STAT:
04555:          ipc_mme_statistics(ipc_entry->fd, (ipc_mme_stat_req_t *)nreq);
04556:          break;
04557:
04558:      case IPC_GET_CNU_INFO:
04559:          ipc_cnu_info(ipc_entry->fd, (ipc_cnu_status_req_t *)nreq);
04560:          break;
04561:      case IPC_GET_CNU_LINK_STATS:
04562:          ipc_cnu_link_stats(ipc_entry->fd, (ipc_cnu_status_req_t *)nreq);
04563:          break;
04564:      case IPC GET CNU MIB:
```

```
04844:      /*begin: add by tengkaien 2014/7/1 */
04845:      case IPC_CATVCOM_MODULE:
04846:      {
04847:          ipc_catvcom_cfg_handle(ipc_entry->fd,(ipc_catvcom_set_req_t *)nreq);
04848:          break;
04849:      }
04850:      /*end: add by tengkaien 2014/7/1 */
04851:      default:
04852:          DBG_PRINTF("Unknown IPC request: %d", nreq->ipc_type);
04853:      }
```

上图 IPC_CATVCOM_MODULE 正是我们之前在 ipc_protocal.h 中定义的 ipc 请求帧类型，当 MASTER 进程收到该类型的 IPC 请求帧后，调用 ipc_catvcom_cfg_handle()函数对请求帧中携带的配置信息进行处理，完成配置项的写入。ipc_catvcom_cfg_handle()函数是我们要定义的处理函数，在 ipc_packet.c 中，如下：

```
04415: /*begin: add by tengkaien 2014/7/1 */
04416: int ipc_catvcom_cfg_handle(int ipc_fd, ipc_catvcom_set_req_t*preq)
04417: {
04418:     ipc_catvcom_set_ack_t ack;
04419:     int update;
04420:     int ret = 0;
04421:     ipc_acknowledge_hdr_t hdr;
04422:     IPC_HEADER_CHECK(ipc_catvcom_set_req_t, preq);
04423:
04424:     memset(&ack, 0, sizeof(ack));
04425:     if (preq->set_or_get == CATVCOM_CFG_SET)
04426:     {
04427:     switch(preq->op)
04428:     {
04429:     case CATVCOM_CFG_TEMP_OP:
04430:     {
04431:         master_cfg_setval(IF_ROOT_IFINDEX,CONFIG_CATVCOM_TEMPERATURE_HI,preq->catv_temperature_upp
04432:         master_cfg_setval(IF_ROOT_IFINDEX,CONFIG_CATVCOM_TEMPERATURE_LO,preq->catv_temperature_low
04433:         ack.hdr.status = IPC_STATUS_OK;
04434:         break;
04435:     }
04436:     case CATVCOM_CFG_VOLTAGE_OP:
04437:     {
04438:         master_cfg_setval(IF_ROOT_IFINDEX,CONFIG_CATVCOM_VALTAGE_HI,preq->catv_valtage_upper_limit
04439:         master_cfg_setval(IF_ROOT_IFINDEX,CONFIG_CATVCOM_VALTAGE_LO,preq->catv_valtage_lower_limit
04440:         ack.hdr.status = IPC_STATUS_OK;
04441:         break;
04442:     }
04443:     case CATVCOM_CFG_OPTPOWER_OP:
04444:     {
04445:         master_cfg_setval(IF_ROOT_IFINDEX,CONFIG_CATVCOM_OPTPOWER_HI,preq->catv_optPower_upper_lim
04446:         master_cfg_setval(IF_ROOT_IFINDEX,CONFIG_CATVCOM_OPTPOWER_LO,preq->catv_optPower_lower_lim
04447:         ack.hdr.status = IPC_STATUS_OK;
04448:         break;
04449:     }
```

可以看到，正是在这个函数中，我们调用 master_cfg_setval 函数对配置项进行写入。至此，一次完整的配置项的读写过程结束。

4. 问题及注意事项

4.1 编译

Startup 文件修改后，编译代码的时候先 make software-cfg-clean 一下，再 make software-cfg 来一下，然后才编译平台代码。（以上过程是否必须有待验证）

4.2 机器启动过程提示读写错误

机器启动的时候，emonitor 进程会打印一两次读写配置错误信息，完全开机以后无读配置错误信息提示。原因是刚开机的时候 emonitor 先于 master 进程启动，故读配置错误。解决方法，等待 master 进程正常启动后再读写配置，如加入下面代码：

```
do
{
    if((access(IPC_SERVER_PATH, F_OK)) != -1)
    {
        break;
    }
    sleep(1);
}while (1);
```

4.2 第一次读配置失败

机器正常启动以后，若果是第一次读取配置项，会出现读取配置失败的情况，而如果此时进行写配置操作能正常进行，写配置完成以后再进行读配置操作，这回能读到写入的配置项。造成这种情况的原因是新增的配置项虽然存在于 etc/config/strtup 中，但由于机器没有执行过恢复出厂配置操作，mnt/strtup 文件中尚未存在新的配置项，故读取不到。解决的方法是程序里判断第一次读取不到的时候将配置项的默认值写入，这样下次读便可以读到。