## **ACL总结文档**

## 底层原理

### **1.1 应用场景**

1. 协议报文上送cpu、丢弃或限速等。
2. 用户access-list配置。，access-list 分为基于 ip 标准、扩展、mac-acl、mpls-acl，udf-acl、ipv6 标准、扩展、高级（ipv6）advance acl。access-list 可以绑定端口、vlan、子接口、port-channel、环回接口等。
3. Policy-map相关配置，action 动作比 access-list 丰富，常见的动作有添加删除修改vlan、cos、dscp 等以及限速、重定向等等其他动作。使用上 policy-map 与class-map 关联使用，与 access-list 类似，可以基于端口、vlan、子接口、port-channel 等去绑定下发规则。
4. 其他需要对报文进行处理的场景，如统计、限速、流分类、修改报文、重定向、与QOS 联动等，该场景与设备上很多模块之间有关联，通常由 acl 模块提供相应的接口创建规则来实现。

### **1.2 基本概念**

芯片中关于 acl 模块上，涉及的几个主要基本概念有：Group、TCAM、Slice、entry、keyfield，action field、classid、portbitmap、League、Presel、Aset。

**Group**：sdk 将相同查找对象的 ace 组成 group 进行统一管理，group 中的多条acl entry 可以统一进行增加、删除等操作。（group id、group priority、group direction、group type），

**Groupid**:用于标识一个 acl group，范围为 0-0xFFFFFFFF,全局唯一，其中用户可分配范围为 0-0xFFFF0000，其他为 sdk 的预留内部分配使用。对于 tcam 的 key，sdk 没有默认创建，需要用户基于应用场景自行创建，而 hash key sdk 内部已经创建 group id。

**group priority** 与 **group direction** 决定了 group 下发到哪块 tcam 中去。

**grouptype**：标识基于哪个查找对象查找，type 类型主要分为 none，hash，portbitmap，vlanclass，portclass 等，sdk 推荐采用 none 类型，即 group 的信息由entry 来指定。

**TCAM**：存储 entry 的硬件资源；

**Slice**: Slice 是 TCAM 的物理分块，Slice 之间会进行并行的关键字匹配查找，每个slice之间得到独立的 action；如果 action 之间存在冲突，高优先级优先；如果action 之间没有冲突，action 会进行合并；

**entry**：单条 acl 规则，entry 的 priority 表示在 slice 内指定的物理位置，优先级高的会位置靠前，优先匹配；（entryid 、entry priority）

**entryid**:用于唯一标识一条 entry，entryid 全局唯一，不同 group 不能使用相同的entryid。

**key field**：entry 中的匹配字段，tcam 上的 entry 通常采用 key-mask 的形式；

**action filed**：entry 中的 action 动作字段；

**class id**：class id 是芯片内的逻辑概念，每个 port/vlan 可以指定一个class id，并且都可以指定不同的 class-id，也可是相同的 class id，比如 acl 可以匹配 class id 来用于多个端口或者 vlan 上，可以节约资源。

**portbitmap**：端口位图，acl 可以匹配端口位图用于多个端口的场景。

**League**：lookup 的 merge。

**Presel**：8180 支持用户自定 key 类型，挑选匹配的 key fileds，使用presel entry 管理该tcam（Presel id，presel priority，key type 范围 16-255）

Cli命令行配置acl举例：

1. acl create group 200 priority 3 direction ingress none

创建group 200，priority 为3，不是代表group的优先级，而是对应的是slice 3

2、acl add group 200 entry 200 ipv4-entry field-mode

创建entry 200，对应group为200，规则类型为ipv4-entry

3、acl entry 200 add key-field ip-sa 1.1.1.1 255.255.255.255

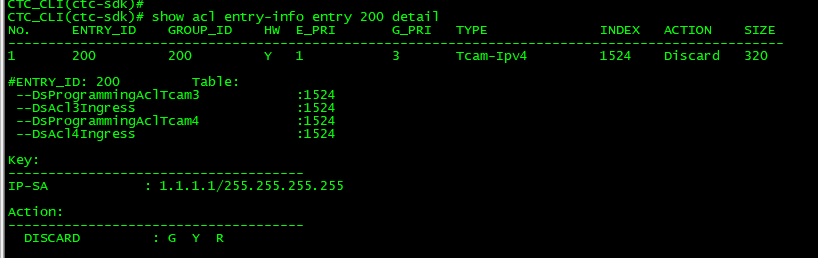
添加entry 200的匹配条件是匹配sip为1.1.1.1

1. acl entry 200 add action-field discard

添加entry 200的动作为丢弃

1. acl install group 200

使能group 下所有的entry，硬件生效。



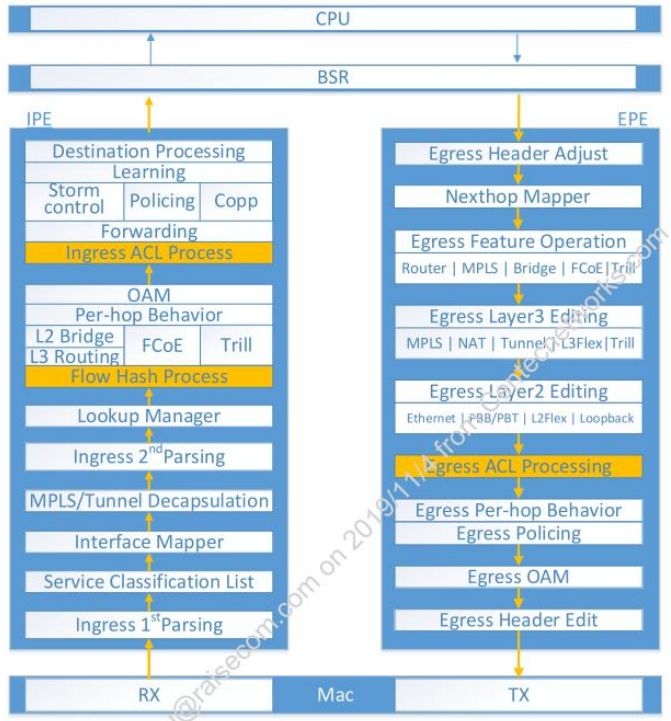
此时acl已经生效了，但是端口进来的报文sip为1.1.1.1规则还是不会命中，原因是，acl还需要使能。使能可以基于全局和端口或l3\_if，端口对应的查找模板应该选为l3模板。

1. port 112 acl-property priority 3 direction ingress acl-en enable tcam-lkup-type l3

端口112对应的slice 3查找模板为l3，此时端口112进来的报文sip为1.1.1.1的报文才会被丢弃。

### **1.3芯片流水线和资源**

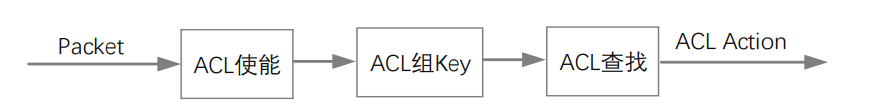
芯片的总体流程图与 acl 的环节如下：



Acl一共有三处

1. flow hash process，hash acl，目前没使用。
2. 只支持一次flow hash 查找
3. 在tcam acl前一级使用，规格最大为64k.
4. Ingress acl process，tcam。
5. 支持ingress 方向16次并行查找（16个slice，每个slice资源160bit \*2K）
6. 支持programmable key，包括udf
7. key长度支持160bits、320bits和480bits，（slice横向拼接）
8. egress acl proessing，tcam
9. 支持4次并行tcam查找（4个slice，每个slice资源160bit&1k）
10. 支持fixed key，不支持programmable key，不支持udf
11. key长度支持160bits、320bits、和640bits;（slice中entry拼接）

### **1.4芯片原理**



芯片收到报文后，首先对报文进行解析，如果使能了 acl 功能，根据配置的 acl 属性和报文类型组 key，

并进行 acl 规则的匹配查找，根据查找结果的 acl action 对报文进行处理。

#### 1.3.1 acl使能

使能分为三个部分：使能 ingress acl flow hash、使能 ingress acl tcam、使能 egress acl tcam。

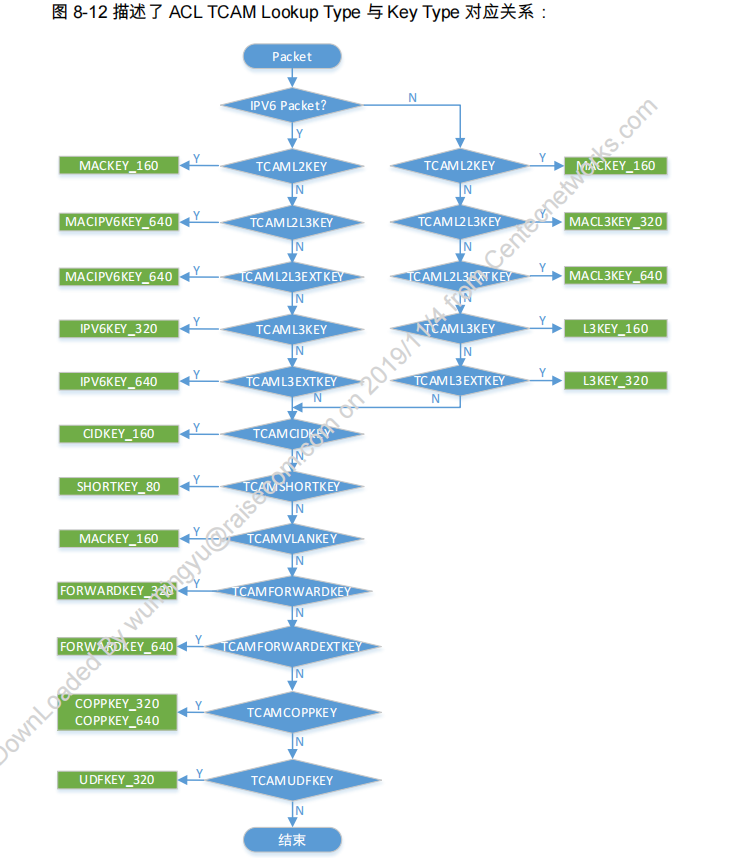
IPE ACL TCAM lookup 使能，芯片支持基于 port、vlan、

l3interface、基于业务、基于全局使能。

优先级如下：全局>业务>l3interface>vlan>port。优先级的顺序芯片可支持调整。业务使能包

含 DsUserId.serviceAclQosEn、DsTunnelld.serviceAclQosEn、DsMpls.serviceAclQosEn。

每个slice可以使能一种类型的模板，使能后只会去查找对应的模板的规则，其他模板的规则不会查找。所以一个slice里面的使能模板要和slice里面的规则的模板一致。(灵活模板除外，见附件)



#### 1.3.2 acl组key

以前7132芯片是每个模板对各自的key，匹配字段。

但是CTC8180芯片中，Ingress acL功能有较大升级，所支持的key由fixed key变为programmable key，兼容原本的fixed key模式

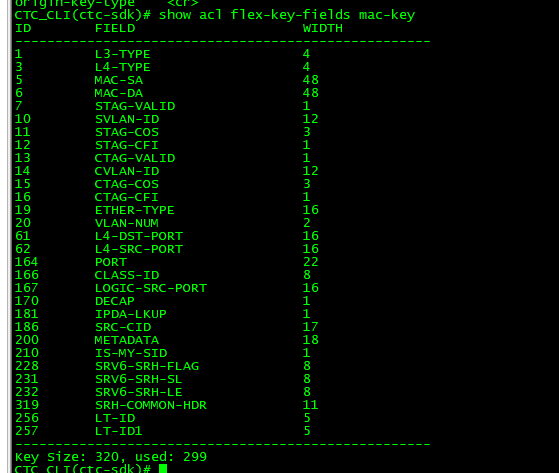
在fixed key模式下，基于端口属性以及报文，选择key type，决定进行tcam查找的key的field，比如选择了160bits的MACkey，ACL lookup的key field固定为MACDA、MACSA以及Ether Type等信息。在programmable key模式下，不再设置key type，根据logic table，灵活选择组key的field。

Programmable key模式下，与fixed key模式相比，采用logic table选key，选择更加灵活，另外programmable key选key时，支持从udf中选取。

Acl key的选取若超过160b，会自动横向合并slice，最大为480bit，三个slice。

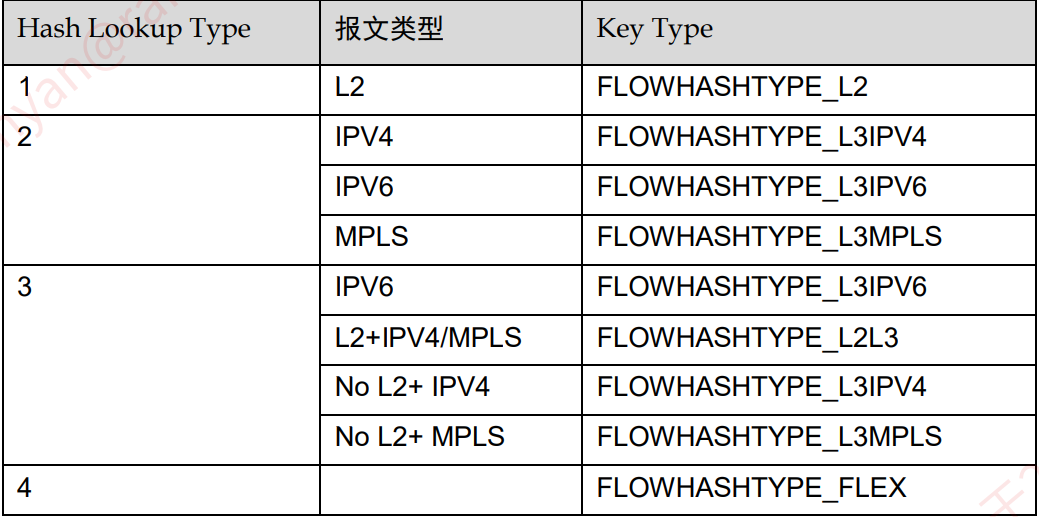
可以通过命令去查看当前模板的匹配字段。

查看mac-key中匹配字段，若匹配其他字段会报错。所以下发acl时首先判断用啥匹配字段，然后选择对应的模板。

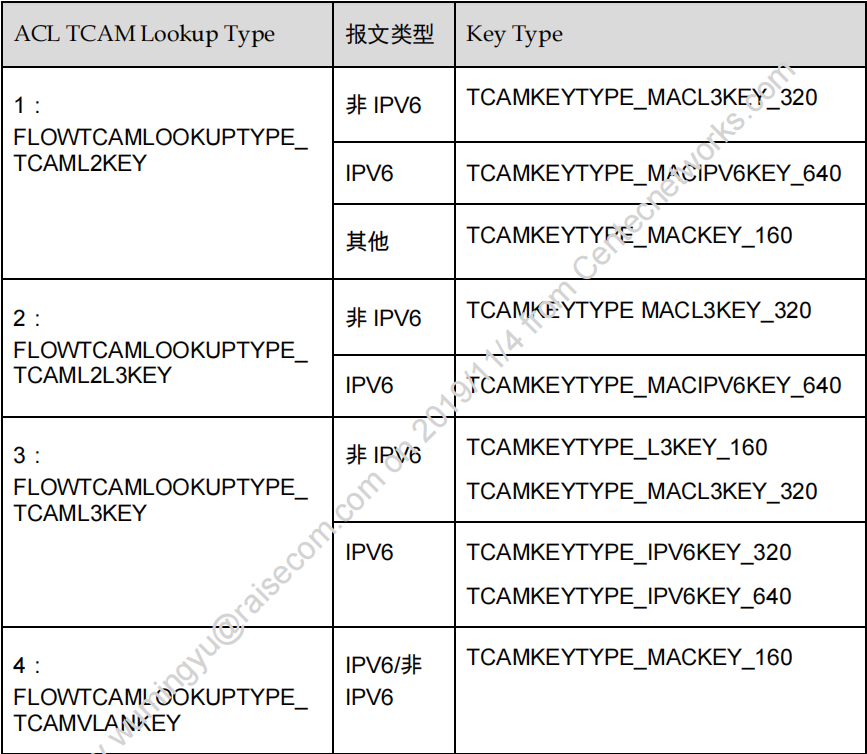


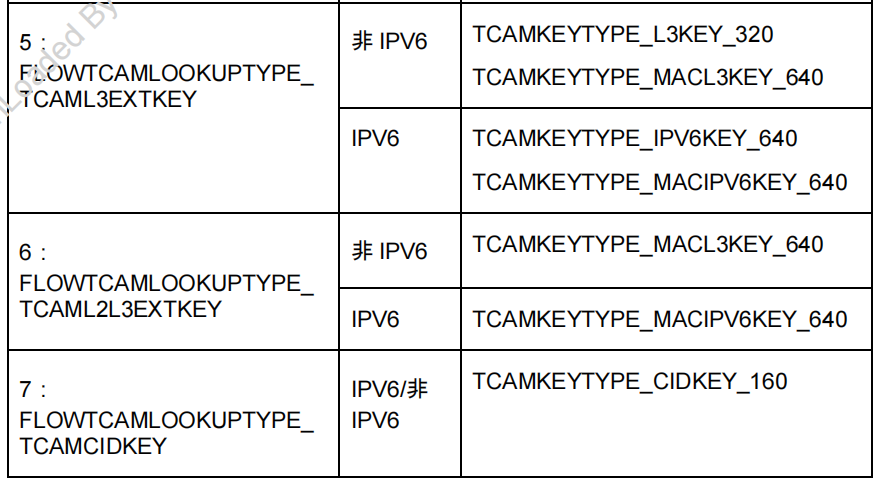
#### 1.3.3 acl查找和action

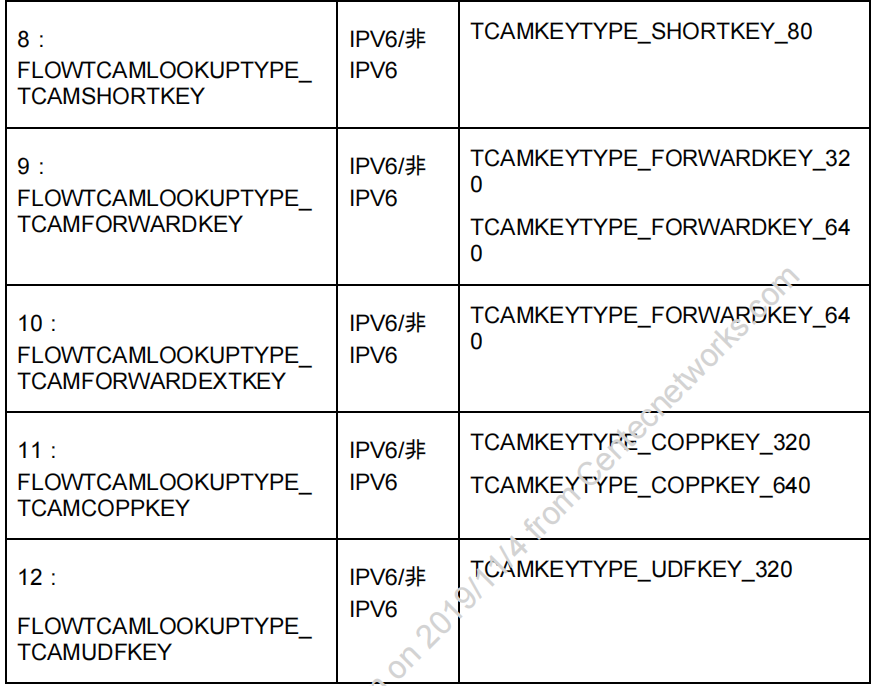
Hash lookup和keyType对应关系。



Ingress lookup







8180与之类似

若果查找结果存在冲突时，就存在合并和优先级选择。

ingress 方向并行查找结果合并的优先级为：acl0>acl1>acl2>acl3>acl4>acl5>acl6>acl7>...>acl15。

egress 方向并行查找结果合并的优先级为：acl0>acl1>acl2>acl3。

### **1.5 acl定位方式**

通过show acl相关信息以及流水可以发现该entry是否命中，acl的index与芯片中的key-index对应，g\_pri和slice对应，要想看比如下面规则104是否命中，

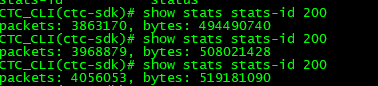
1. 可以加stats去查看。

stats create statsid 200 acl priority 3 ingress

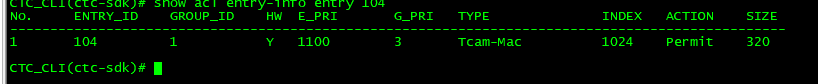
acl entry 104 add action-field stats 200

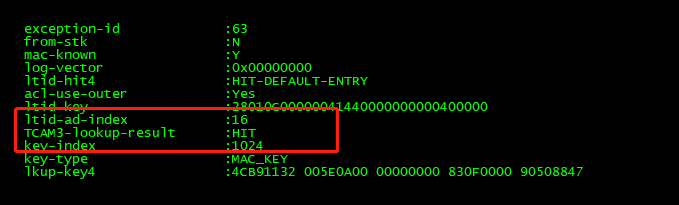
show stats stats-id 200

创建的statsid 中priority 要和group中的priority 保持一致，这样才会生效。



1. 也可以去打印芯片流水，首先看到g\_pri为3，所以要看tcam3的lookup-result，其次entry 104对应的index的1024，需查看tcam3\_lookup-result是否为hit，且key-index是否为1024，此处代表命中entry 104。

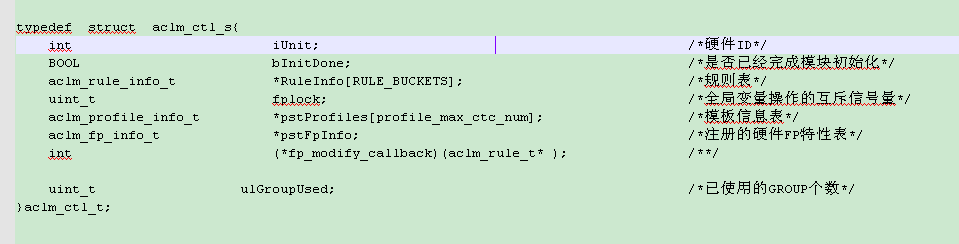




## 2、实现方式

底层设计思路为1个slice（拼接后的slice不算）对应一个group，一个group对应一种类型的规则。然后每个端口的slice对应使能相同的规则的模板。因为slice是横向拼接的，所以整体来说对应的并行查找资源很少，初始化所有的slice都会使用完毕。每个group有各自的规则类型。这样上层下发一个规则，只需要找到对应的模板类型，查询到对应group，然后添加entry即可。

### **2.1全局结构体**



Static aclm\_ctl\_t m\_AclmCtlInfo[CTC\_MAX\_NUM\_UNITS]；全局acl表

1. iUnit 芯片号，默认为0
2. bInitDone 是否初始化成功，若成功则置为TRUE，避免反复初始化
3. Aclm\_rule\_info\_t \*RuleInfo[RULE\_BUCKETS]

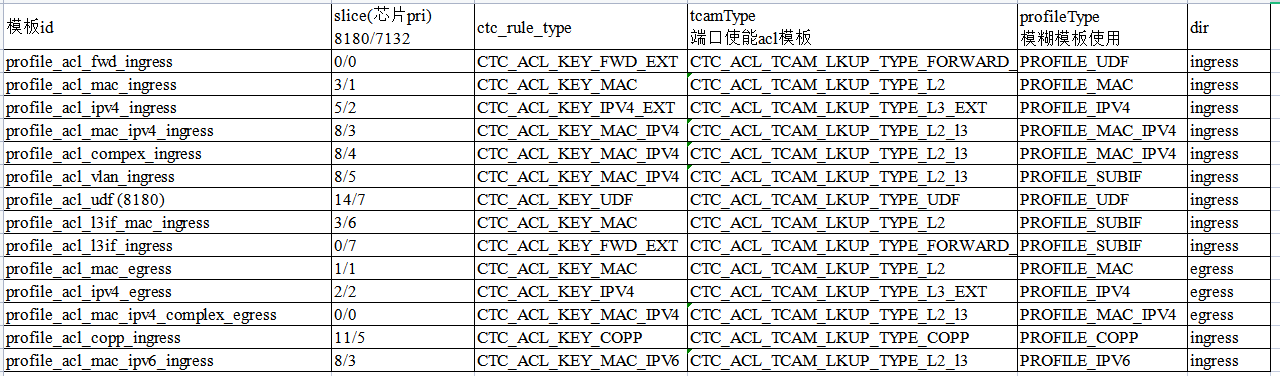
一个hash表，rule\_id/RULE\_BUCKETS算出到哪个表中，rule 0 到127 hash 到同一个下标中，表中用链表把所有的规则连接起 来，最开始指向最后一个节点，添加规则的时候需要填充数 据

1. Fplock 全局变量操作的互斥信号量
2. pstProfiles 模板信息表
3. pstFpInfo 芯片特性表，暂时为用到
4. Rule 回调函数
5. ulGroupUsed 已经使用group个数(acl最大为20)

### **2.2初始化操作**

1. 初始化每个模板

每个模板赋值如下



Ingress 每个block 8180 160\*2048 7132 160\*512

Egress 8180 160\*1024 7132 160\*512

依次使能每个模板，其中7132不需要使能udf

7132 一般的slice是160b\*512条规则，slice 2和slice 3特殊

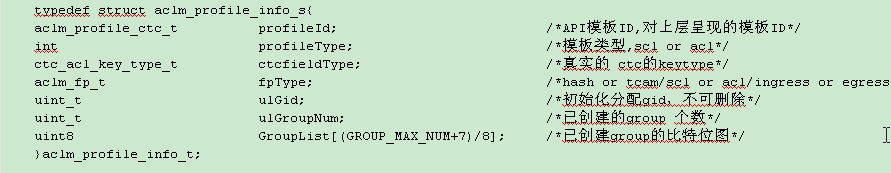
Mem\_profile 单独又分配了160b\*1024 所以一共160b\*1536 = 80b\*3k

可见7132和8180中有模板共用一个pri，但是下发的规则类型不一样，这时需查看端口或全局的使能方式去选哪些规则可以命中，同时占用一个pri，查找方式和规则类型一致，则可以按照优先级去命中。

调用芯片接口申请group 得到group\_id，除了profile\_acl\_vlan\_ingress(默认不用到，不使能)和profile\_acl\_copp\_ingress(全局使能，初始化使能)，遍历每个端口使能每个pri对应的acl

申请成功后每个模板的信息都存放在m\_AclmCtlInfo[iUnit].pstProfiles[profileId]中

结构体参数数据含义



ProfileId 对应提供给上次的模板id

profileType 相关模板的type信息，模糊模板的时候使用

CtcfieldType 底层规则的keytype

fpType hash or tcam /scl or acl(目前只有acl)/ingress or egress

ulGid 生成的group\_id (初始化生成的，后面不能删除)

ulGroupNum 该模板使用的group数量 创建成功后为1

GroupList 创建的bitmap图

创建模板成功后，对应的信息都存放在软表中，更新GroupList 比特位图，和使用的ulGroupNum数目，同时更新aclmCtlInfo[iUnit].ulGroupUsed++，这样软表的模板信息创建完毕。

1. udf初始化

初始化三个udf模板，pri分别为1、2、3

Pri 1 匹配ethertype为0x8847(mpls报文) ud offset从4 8 12 48开始偏移，每个4字节，8180暂未用到。

Pri 2 从ip头开始偏移 匹配0到15字节（对应网管 udf中layer2报文）

Pri 3 匹配ethertype 为0x0800 ipv4报文 从ip头后20开始偏移16字节，所以一般是传输层tcp或udp的前16字节或者ipv4扩展头的后开始的16字节

1. 还会初始化pir 21

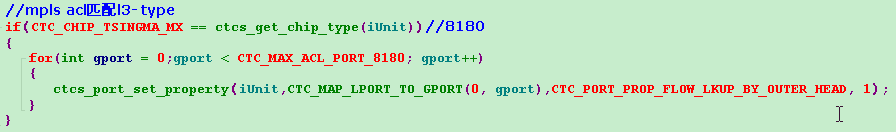
Pir 21 匹配mpls报文，且传输层偏移0~5 12~13 20~21 24~25 28~29，因为设置了pir 21,匹配条件和pri 1一致，优先级比pir 1高，所以pir 1所用的偏移用不到，目前pir 1没有人使用。

1. 全局使能acl查找，把来l2l3\_ext\_key\_use\_l2l3\_key置为0

全局查找pir 8180为11 7132为5

使能对应lookup的全局查找 类型为CTC\_ACL\_TCAM\_LKUP\_TYPE\_COPP

4、8180 mpls acl匹配l3-type需要把每个端口的相关字段置位，匹配外层字段。

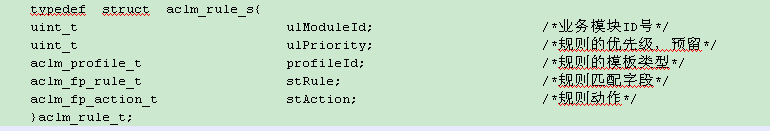


初始化结束，出来pstFpInfo（暂时未使用）和RuleInfo（规则信息，添加规则时填充），其他字段都初始化完毕，且赋值。

### **2.3规则添加**

函数为：FDI\_acllm\_RuleCreate(int IUnit, aclm\_rule\_t \*pstRule, uint\_t \*pulRuleId)

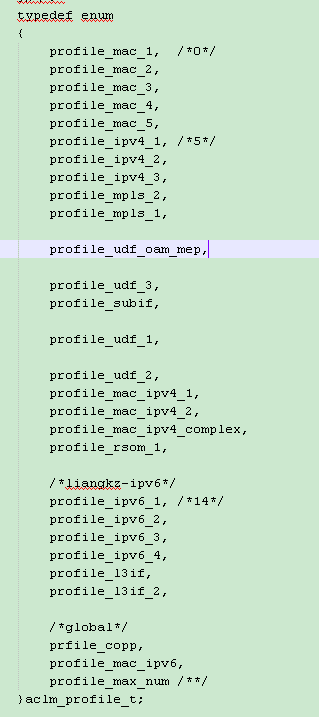
结构体如下：



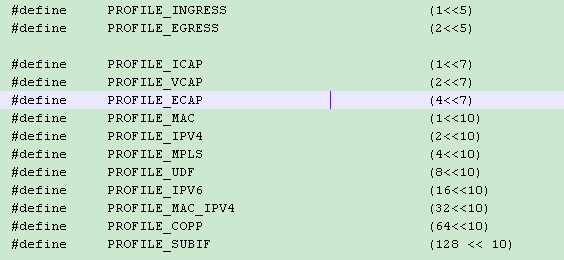
PstRule 由上层传入

1. ulModuleId 不同的应用类型
2. UlPriority 规则的优先级，和芯片pri不一致，是区分同一个slice不同entry的优先级
3. ProfileId 模板类型 和初始化模板类型的模板id不一致，存在对应的转化关系。

如下：



或动态模板



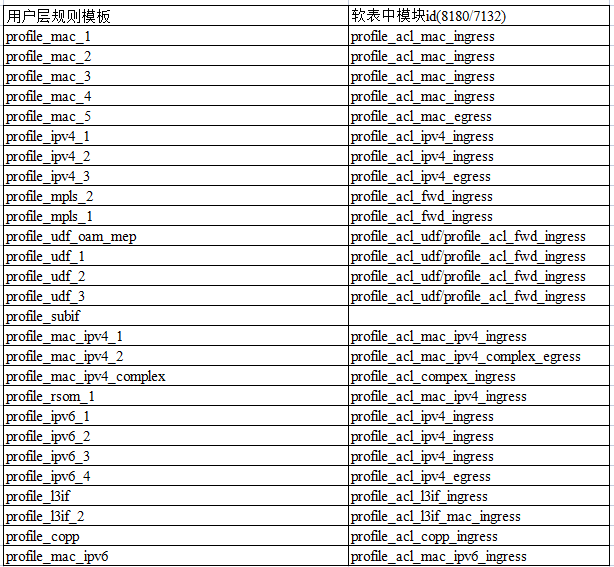
1. stRule 规则匹配字段
2. StAction 规则动作

输出参数

\*pulRuleId 芯片添加成功后，返回的芯片entry\_id。

1. 进入函数后，首先参数检查是否合法，
2. 然后根据上层传入的模板信息转化为芯片的模板信息。

静态模板对应关系为



逐一转化，得到对应的模板

目前profile\_mpls\_1和profile\_mpls\_2使用会报错。

动态模板对应关系为（目前只有ssp层Acl才会使用这种）

遍历每个模板，得到每个模板对应的profileType和fpType(类型scl ingress egresss等等)，然后对比profileType和上层传入的profileId相关信息，初始化把每个模板都置位，逐个对比，把不合理的情况全部剔除，最终得到合适的模板。

最终输出的模板的bitmap（可能有多个置位）

3、找到对应的模板后，调用芯片接口申请规则id

4、遍历模板的bitmap。找到查询对应的模板软表

获取对应模板的ctc\_key\_type和groupList

遍历grouplist的bitmap表，找出置位的group

5、芯片规则添加

遍历模板使用的group 的bitmap，可以一个模板有多个group，（默认一个模板只有一个group）第一个group添加失败后添加第二个，直到所有遍历结束。

CTC\_ACL\_KEY\_IPV4\_EXT 匹配ipv6报文自动修改为 CTC\_ACL\_KEY\_MAC\_IPV6

CTC\_ACL\_KEY\_MAC\_IPV4或CTC\_ACL\_KEY\_MAC\_IPV4\_EXT匹配ipv6报文自动修改规则为CTC\_ACL\_KEY\_MAC\_IPV6

添加entry，填充模板和优先级

Ctcs\_acl\_add\_entry(iUnit, ulGid, &acl\_entry)

根据rule信息添加key\_field

Aclm\_RuleQualifySet(iUnit, PsrRule,ulEid);

Acl变动回调函数，目前为NULL

根据action信息添加action

Aclm\_RuleActionSet(iUnit, psrRule, ulEid, ulGid);

芯片使能规则

Ctcs\_acl\_install\_entry(iUnit,ulEid);

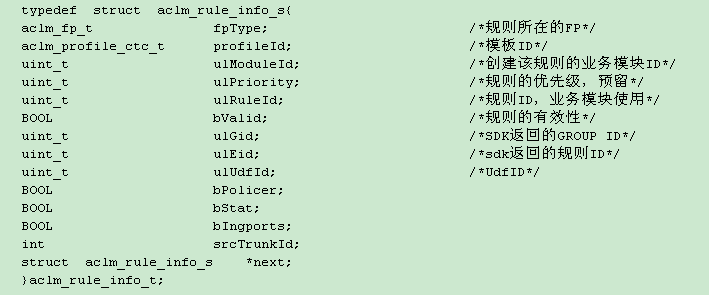
添加成功则转8.

6、若添加失败，可能group里面规则满了（也就是一个slice资源已经用完了），查询是否还有空闲资源，当使用模板group>1且是入方向时，看是否还有空闲块可以使用。

一般是7132 silce 6 7是空闲，打的补丁，申请group\_id，然后调用sdk接口初始化对应的slice，和初始化其他模板一样的处理

7、再次重复执行步骤5，若添加成功则更新全局变量软表对应模板使用group和bitlist， 使用group数量加1，添加失败回滚。

8、添加成功后填充软件数据



1. fpType 规则使用的是什么资源，比如scl或hash acl或者ingress acl或者egress acl等等，该字段是从entry对应的模板结构体中取得，才赋值。
2. ProfileId 模板id，该字段是从entry对应的对应的模板结构体中取得，才赋值
3. UlModuleId 应用类型，上次rule结构体赋值
4. ulPriority 规则优先级，上层rule结构体赋值
5. ulRuleId 返回给上层的rule\_id，一般为sdk生成的entry\_id+ 1
6. bValid 规则的有效性，成功后为TRUE
7. UlGid sdk返回的group\_id
8. UlEid sdk返回的entry\_id,删除需使用的key
9. ulUdfid 暂时未使用
10. bPolicer action是否开启限速，从action中提取
11. bStat action 是否开启计数，从action中提取
12. bIngports 是否开启端口匹配，从rule中提取
13. SrcTrunkId 记录上层传入的聚合口id从rule中提取
14. Next 下一个节点信息

填充上结构体后，插入全局变量软表中，插入的方式类似拉链法

Rule/RULE\_BUCKETS 算出到哪个表中，ruke 0到127 hash到同一个下标中，表中用链表把该entry挂在相同index的第一个节点前，然后指向第一个节点，也就是新增的节点，后添加的在前面

### **2.4规则删除**

函数为int FDI\_aclm\_RuleDelete(int iUnit, uint\_t ulRuleId)

根据ulRuleId查询软表 遍历链表的带rule信息，未查到则报错

查到则获取对应的 芯片eid和gid和模板id

1. 如果rule里面bStat置位，如果stat对应节点只剩最后一条entry，那么删除stat计数
2. 某些情况下删除对应的policer限速
3. 调用sdk接口删除entry信息 ctcs\_acl\_uninstall\_entry、ctcs\_acl\_remove\_entry
4. 删除entry软表信息
5. 比对rule\_id 若是首节点，则直接删除，更新首节点指向第二个节点
6. 若是中间节点，用一前一后指针遍历，找到后删除
7. 调用sdk接口 free掉eid。
8. 根据模板id获取对应模板数据，若是初始化分配的group，则不可删除，否则走7
9. 调用sdk接口查询 ctcs\_acl\_get\_group\_info得到group的pri和dir

Ctcs\_acl\_get\_multi\_entry的带group的entry\_num

若果entry\_num为0，则代表group下没有entry，则可以删除该group

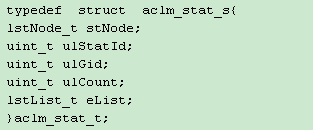
8）调用ctcs\_acl\_destroy\_group 销毁group 且调用sdk接口free掉gid，把每个端口使能对应的slice的信息全部清空，ctcs\_port\_set\_acl\_property

9）更新模板对应的group软表信息，去使能对应模板的groupbitlist group使用数量减一，最外层使用group数量减1

### **2.5添加和删除stats(计数 counter)**

LOCAL lstList\_t m\_AclmStatInfo[CTC\_MAX\_NUM\_UNITS]

链表存放的数据为



1. stNode 通过node连接下一个节点
2. ulStatId 芯片申请的stats\_id
3. ulGid 芯片的group\_id
4. ulCount 该stats规则计数
5. eList 规则id链表

Typedef struct aclm\_entry\_s{

lstNode\_t stNode;

Uint\_t ulEid;

}aclm\_entry\_t；

1. stNode 通过node连接下一个节点
2. ulEid 芯片规则id
3. 添加action时判断statflag是否置位，若置位说明使能counter功能
4. 根据gid调用sdk接口得到对应芯片的pri和dir，调用sdk接口申请stats\_id
5. 若是statflag 为FIELD\_STAT\_ACL，则统计所有报文，若是FIELD\_STAT\_FTP和FIELD\_START\_CAR则统计黄色报文及以上
6. 若没有stareStatId为0，调用ctcs\_stats\_create\_statsid接口创建stats，然后acl的action关联此stats\_id
7. 若是sharestatflag置位，代表共享counter，修改shareStatId为申请的stats\_id。（这个变量存放在action里面，上层如果想要共享counter，则需关注这个值）
8. 保存软表aclm\_stat\_t软表，ulCount++，加入m\_AclmStatInfo中。
9. 接3，与4同级，若是shareStatId不为0，而且sharestatflag置位，代表共享counter
10. acl的action直接关联该shareStatId
11. 查询stats软表，查到对应的stat\_id节点，ulCount++，eList链表中插入新的rule\_id

删除stats计数，在删除rule中实现

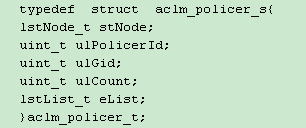
1. 首先看规则里面的bStat是否置位，若置位则表示使能stats
2. 双层循环遍历，第一层是stats\_id所在节点，第二层是stats\_id里面的e\_id，查看eid和删除规则的eid是否一致，若一致则查询到。
3. 找到rule\_id后，eList链表中剔除该节点，对应的stats节点中的ulCount--。
4. 若stats中ulCount为0，代表没有规则使用该stats，调用ctcs\_stats\_destroy\_statsid销毁计数，然后free掉stats\_id，销毁m\_AclmStatInfo数据中的该stats节点数据。

清空stats计数

1. 根据rule\_id 查询对应的rule信息软表
2. 得到芯片的eid，遍历stats软表根据eid得到stats\_id
3. 调用ctcs\_srars\_clear\_stats清空计数

### **2.6添加和删除policy（meter限速）**

LOCAL lstList\_t m\_AclmPolicerInfo[CTC\_MAX\_NUM\_UNITS];



1. stNode 通过node连接下一个节点
2. ulPolicerId 芯片申请的policer\_id
3. ulGid 芯片的group\_id
4. ulCount 该policer\_id规则计数
5. eList 规则id链表

Typedef struct aclm\_entry\_s{

lstNode\_t stNode;

Uint\_t ulEid;

}aclm\_entry\_t；

1）stNode 通过node连接下一个节点

2）ulEid 芯片规则id

添加policy限速

1. 添加action时判断meterflag和policerflag和hiepolicerflag是否置位，若置位说明使能限速功能
2. 根据对应的信息填充ctc的限速结构体

Meterflag和hiepolicerflag都使用 CTC\_QOS\_POLICER\_MODE\_STBM

Policerflag

1. 若pstFpAction->opa\_mode == policerModeCommitted 则只需要关注pir和pbs
2. 若pstFpAction->opa\_mode == policerModeCoupledTrTcmDs 使用CTC\_QOS\_POLICER\_MODE\_RFC4115
3. 其他情况ctc使用的算法policer\_mode为上层传入的stFpAction->opa\_mode

PolicerModeSrTcm /\* RFC 2697\*/

policerModeTrTcm /\* RFC 2698 \*/

policerModeTrTcmDs /\* RFC 4115\*/

4、普通限速：

调用sdk接口申请policy\_id，调用ctcs\_qos\_set\_policer下发限速相关数据，采用的都是red-drop，acl的action关联该policer\_id，类似stats一样保存软表。

1. 共享限速：

SrcFpid对应rule\_id，根据rule\_id查询软表找到对应的policy\_id

添加软表，把新规则挂在eList中，且ulCount++，类似stats一样保存软表。

若是pstFpAction->meterrate和pstFpAction->meterflag不为0，则重新调用ctc接口下发policy信息，policy\_id不变

删除policy限速，在删除rule中体现

1. 双层循环遍历，第一层是policy\_id所在节点，第二层是policy\_id里面的e\_id，查看eid和删除规则的eid是否一致，若一致则查询到
2. 找到rule\_id后，eList链表中剔除该节点，对应的policy\_id节点中的ulCount--。
3. 若policy\_id中ulCount为0，代表没有规则使用该policy，调用ctcs\_qos\_set\_policer清0限速，然后free掉policy\_id，销毁m\_AclmStatInfo数据中的该policy节点数据

更新policy限速

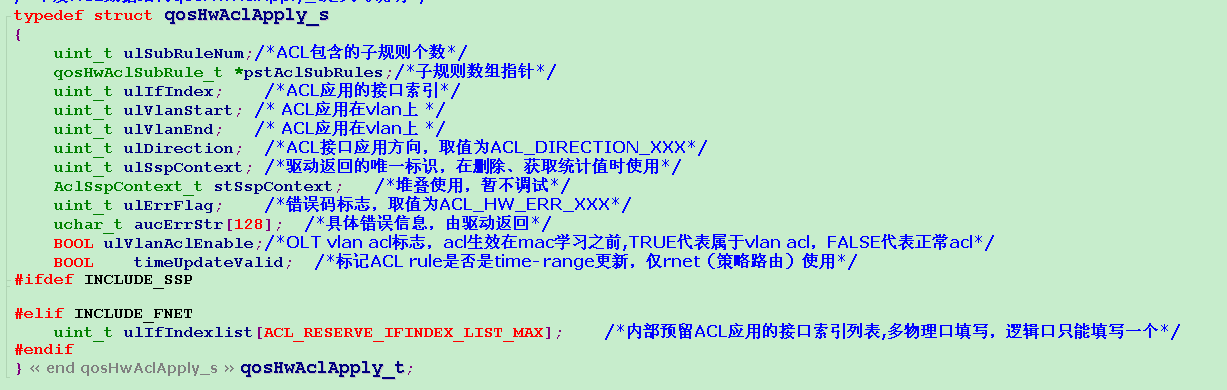
Int FDI\_aclm\_RulePolicerCfgSet(int iUnit, uint\_t ulRuleId, aclm\_policer\_config\_t \*policerCfg)

1. 上层调用根据rule\_id找到对应的policy\_id
2. 调用ctcs\_qos\_set\_policer更新限速数据
3. 重新使能ctcs\_acl\_install\_entry entry

### **2.7 Ssp层实现**

**2.7.1 添加**

1. 入口：
2. Ssp\_acl\_group\_set(qosHwAclApply\_t \*pstHwAclApply)



ulSubRuleNum acl下发的rule数目

\*PstAclSubRules acl中rule信息

ulIfIndex acl应用的接口索引  
 ulVlanStart acl应用vlan上的起始vlan，暂未使用

ulVlanEnd acl应用vlan上的终止vlan，暂未使用

ulDirection acl应用方向

ulSspContext 驱动返回的唯一标识，记录acl\_id的底层索引，最大为2048， 软表数据索引。

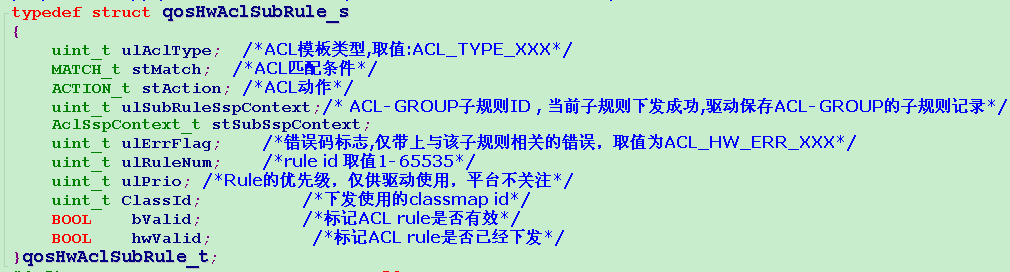
StSspContext 堆叠使用，暂未使用

ulErrFlag 驱动返回的错误码

ulVlanAclEnable 是否是vlanacl，暂未使用

TimeUpdateVaild 暂未使用

其中rule信息结构体为：



ulAclType acl模板类型

StMatch rule的匹配条件

StAction rule的动作

ulSubRuleSspContext 驱动返回的rule\_id

stSubSspContext中ulAclNum acl号，copy\_to\_cpu时需带入metedate为acl号

ulErrFlag 驱动返回错误码

ulRuleNum rule\_id，暂未使用

ulPrio rule的优先级，暂未使用

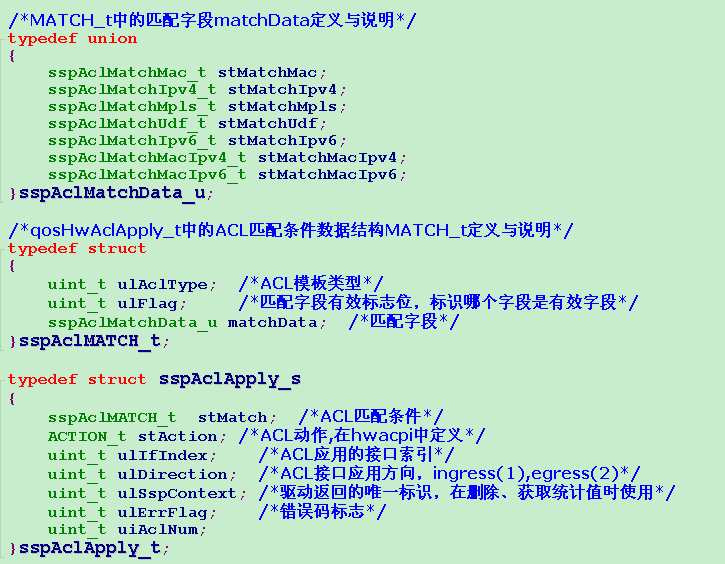
ClassId class\_id，暂未使用  
 bValid 标记acl rule是否有效

hwVaild 标记acl rule是否已经下发

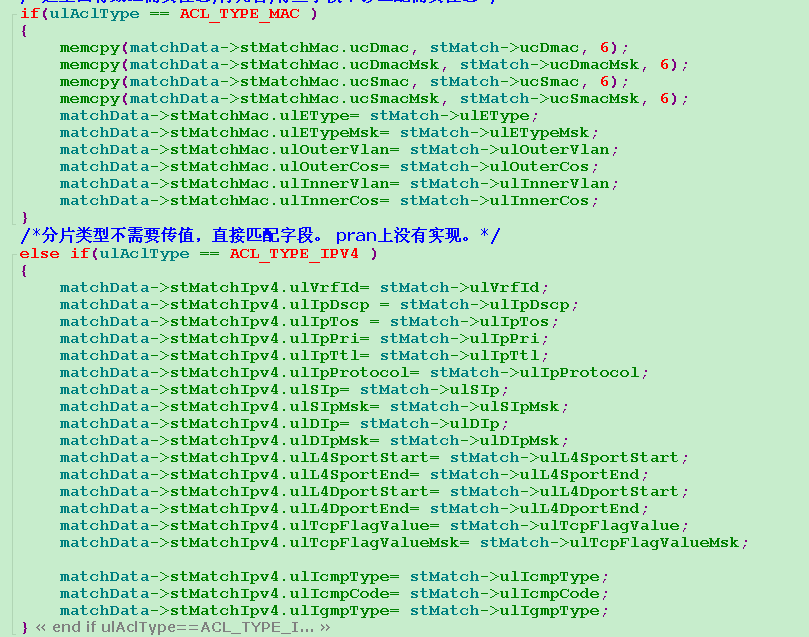
其中match中有个参数

Uchar\_t aucFlag[ACL\_MATCH\_FLAG\_MAX] /\*匹配字段有效标志位\*/只有相关字段置1了，匹配数据才有效。

单挑rule和软表结构体



1. 首先获取空闲的ulIdx索引，acl最大条目数为2048。若已经下发超过2048条acl，报错。
2. 创建sspAclApplt\_t pstAclCfg结构体，赋值ulIfIndex和ulDirection。
3. 遍历acl的所有子规则，赋值所有子规则的动作。
4. 调用ssp\_acl\_matchflag\_adapt对平台传入的aucFlag进行转化，转化为ulFlag，(由数组转化为一个int标志，注意此处把各自匹配类型的放在一起，规划好bitmap)
5. 调用ssp\_acl\_group\_analyse对平台传入的匹配字段进行转化。不同类型的acl匹配字段如下：









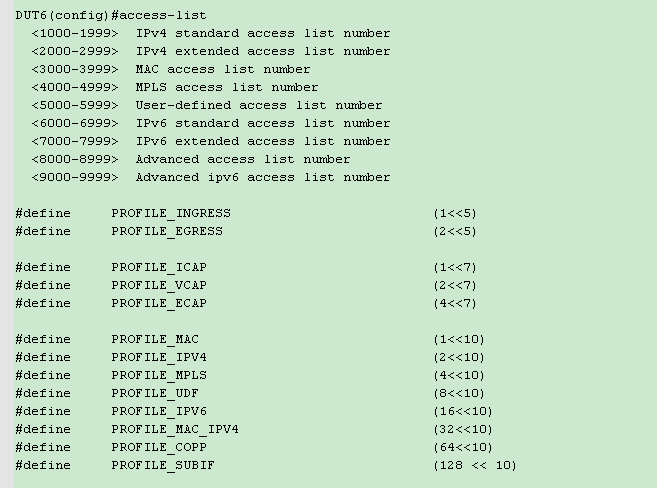
1. 调用ssp\_acl\_set下发规则，对ulIfIndex进行解析，转化为芯片口
2. \_ssp\_acl\_action\_para对ssp的action转化为fdi的action
3. \_\_ssp\_acl\_key\_para把ssp的匹配条件转化为fdi的匹配条件。
4. \_ssp\_acl\_ifIndex\_analyze把ulIfIndex解析，转化为fdi转化匹配条件（端口 vlan 聚合口）。
5. Ipv6取消vlan匹配，mac模板若是匹配innervlan 或cos,改为mac\_ipv4模板.
6. 模板转化为fdi的动态模板，rule优先级为0。
7. 调用FDI\_aclm\_RuleCreate添加规则。
8. 用ulSubRuleSspContext记录返回的rule\_id。ulErrFlag记录返回的错误码，记录软表，把连接插入的m\_plstAcl[ulddx]中。把ulSspContext赋值为ulIdx。
9. 若失败，回滚删除前面下发的规则。

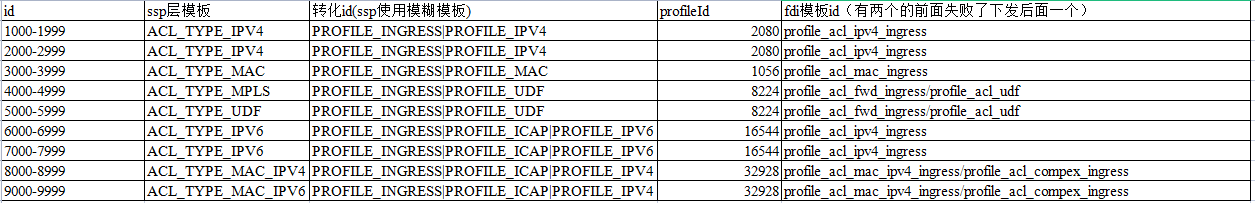
**2.7.2 删除**

Ssp\_acl\_group\_delete

取出上层传入的ulSspContext，遍历软表，得到该acl中的所有rule\_id，依次删除销毁软表。

### **2.8 Ssp层对应关系**

****



## 3、附件