# 8.通用流API

## 8.1.概述

此API提供了一种通用的方式来配置硬件以匹配特定的Ingress或Egress流量，根据用户的任何配置规则更改其操作或查询相关计数器。

所有API带有 rte\_flow前缀，在文件 rte\_flow.h 中定义。

* 可以对报文数据(如协议头部，载荷)及报文属性(如关联的物理端口，虚拟设备ID等)执行匹配。
* 可能的操作包括丢弃流量，将流量转移到特定队列、虚拟/物理设备或端口，执行隧道解封、添加标记等操作。

它比其涵盖和替代（包括所有功能和过滤类型）的传统过滤框架层次更高，以便发布所有轮询模式驱动程序（PMD）明确行为的通用操作接口。

迁移现有应用程序的几种方法在API迁移中有描述。

## 8.2.流规则

### 8.2.1.描述

流规则是具有匹配模式的属性和动作列表的组合。流规则构成了此API的基础。

一个流规则可以具有几个不同的动作(如在将数据重定向到特定队列之前执行计数，封装，解封装等操作)，而不是依靠几个规则来实现这些动作，并且通过应用程序操作具体的硬件实现细节来顺序执行这些规则。

API提供了基于规则的不同优先级支持，例如，当报文匹配两个规则时，强制先执行特定规则。然而，硬件是否支持多个优先级并不能保证。即使硬件支持，可用优先级的数量通常也较低，这也是为什么还需要通过PMDs在软件中实现(如通过重新排序规则可以模拟缺失的优先级)。

为了尽可能保持与硬件无关，默认情况下所有规则都被认为具有相同的优先级，这意味着重叠规则（当数据包被多个过滤器匹配时）之间的顺序是未定义的，不能保证谁先执行。

当给定一个优先级时，PMD如果能够检测到的话（例如，如果模式匹配现有过滤器），可能会拒绝在此优先级下创建重叠规则。

因此，对于给定的优先级，可预测的结果只能通过非重叠规则来实现，在所有协议层上使用完全匹配。

流规则也可以分组，流规则的优先级特定于它们所属的组。因此，给定组中的所有流规则在另一个流规则组所有规则之前或之后。

根据规则支持多个操作可以在非默认硬件优先级之前内部实现，因此两个功能可能不能同时应用于应用程序。？？？？

考虑到允许的模式/动作组合不能提前知道，并且将导致不切实际地大量的暴露能力，提供了从当前设备配置状态验证给定规则的方法。这样，在启动数据路径之前，应用程序可以检查在初始化时是否支持所需的规则类型。该方法可以随时使用，其唯一要求是应该存在规则所需的资源（例如，应首先配置目标RX队列）。

每个定义的规则与由PMD管理的不透明句柄相关联，应用程序负责维护它。这些句柄可用于查询和规则管理，例如检索计数器或其他数据并销毁它们。为了避免PMD方面的资源泄漏，在释放相关资源（如队列和端口）之前，应用程序必须显式地销毁句柄。

以下小节覆盖如下内容：

* 属性（由struct rte\_flow\_attr表示）：流规则的属性，例如其方向（Ingress或Egress）和优先级。
* 模式条目（由struct rte\_flow\_item表示）：匹配模式的一部分，匹配特定的数据包数据或流量属性。也可以描述模式本身属性，如反向匹配。
* 匹配模式：要查找的流量属性，组合任意的模式。
* 动作（由struct rte\_flow\_action表示）：每当数据包被模式匹配时执行的操作。

### 8.2.2.属性

#### 8.2.2.1.Group

流规则可以通过为其分配一个公共的组号来分组。较低的值具有较高的优先级。组0具有最高优先级。

虽然是可选的，但是建议应用程序尽可能将类似的规则分组，以充分利用硬件功能（例如，优化的匹配）并解决限制（例如，给定组中可能只允许单个模式类型）。

请注意，并不保证支持多个组。

#### 8.2.2.2.Priority

可以将优先级分配给流规则。像Group一样，较低的值表示较高的优先级，0为最大值。

具有优先级0的Group 8流规则，总是在Group 0优先级8的优先级之后才匹配（Group的优先级先得到保证）。

组和优先级是任意的，取决于应用程序，它们不需要是连续的，也不需要从0开始，但是最大数量因设备而异，并且可能受到现有流规则的影响。

如果某个报文在给定的优先级和Group中被几个规则匹配，那么结果是未定义的。 它可以采取任何路径，可能重复，甚至导致不可恢复的错误。

请注意，不保证能支持超过一个优先级。

#### 8.2.2.3.Traffic direction

流量规则可以应用于入站和/或出站流量（Ingress/Egress）。

多个模式条目和操作都是有效的，可以在两个方向中使用。但是必须至少指定一个方向。

不推荐对给定规则一次指定两个方向，但在少数情况下可能是有效的（例如共享计数器）。

### 8.2.3.模式条目

模式条目分成两类：

* 匹配协议头部及报文数据（ANY，RAW，ETH，VLAN，IPV4，IPV6，ICMP，UDP，TCP，SCTP，VXLAN，MPLS，GRE等等），通常关联一个规范结构。
* 匹配元数据或影响模式处理（END，VOID，INVERT，PF，VF，PORT等等），通常没有规范结构。

条目规范结构用于匹配协议字段（或项目属性）中的特定值。文档描述每个条目是否与一个条目及其类型名称相关联。

可以为给定的条目最多设置三个相同类型的结构：

* Spec： 要匹配的数值（如IPv4地址）。
* Last：规格中的相应字段的范围上限。
* Mask：应用于spec和last的位掩码（如匹配IPv4地址的前缀）。

使用限制和期望行为：

* 没有spec就设置mask和last是错误的。
* 错误值如0或者等于spce中相应值的last字段将被忽略，他们不产生范围。不支持低于spce的非0值。
* 设置spce和可选的last，而不设置mask会导致PMD使用该条目定义的默认mask（定义为rte\_flow\_item\_{name}\_mask常量）。
* 不设置任何值（如果支持）相当于提供空掩码的广泛匹配。
* 掩码是用于spec和last的简单位掩码，如果不小心使用，可能会产生意想不到的结果。例如，对于IPv4地址字段，spec提供10.1.2.3，last提供10.3.4.5，掩码为255.255.0.0，有效范围为10.1.0.0～10.3.255.255。

匹配以太网头部的条目示例：  
Table 8.1 Ethernet item

| **Field** | **Subfield** | **Value** |
| --- | --- | --- |
| Spec | Src | 00:01:02:03:04 |
|  | Dst | 02:2a:66:00:01 |
|  | Type | 0x22aa |
| last | Unspecified |  |
| mask | Src | 00:ff:ff:ff:00 |
|  | Dst | 00:00:00:00:ff |
|  | Type | 0x0000 |

无掩码的位表示任意的值（如下面显示的？），根据上面的条目，具有如下的属性以太头部的报文将被匹配：

* src：??:01:02:03:??
* dst：??:??:??:??:01
* type：0x????

### 8.2.4.匹配模式

通过堆叠方式从最底层协议开始匹配条目的方式形成模式。这种堆叠限制不适用于那些可以放在任意位置，但是不影响匹配结构的元条目。

模式由END条目终结。

例子：

Table 8.2 TCPv4 as L4

| **Index** | **Item** |
| --- | --- |
| 0 | Ethernet |
| 1 | IPv4 |
| 2 | TCP |
| 3 | END |

Table 8.3 TCPv6 in VXLAN

| **Index** | **Item** |
| --- | --- |
| 0 | Ethernet |
| 1 | IPv4 |
| 2 | UDP |
| 3 | VXLAN |
| 4 | Ethernet |
| 5 | IPv6 |
| 6 | TCP |
| 7 | END |

Table 8.4 TCPv4 as L4 with meta items

| **Index** | **Item** |
| --- | --- |
| 0 | VOID |
| 1 | Ethernet |
| 2 | VOID |
| 3 | IPv4 |
| 4 | TCP |
| 5 | VOID |
| 6 | VOID |
| 7 | END |

这个例子显示了元条目如何不影响匹配结果，只要他们保持堆叠正确。这个例子得到的匹配结果与模式“TCPv4 as L4”相同。

Table 8.5 UDPv6 anywhere

| **Index** | **Item** |
| --- | --- |
| 0 | IPv6 |
| 1 | UDP |
| 2 | END |

假如PMD支持，如上述例子，缺少Ethernet规范，忽略堆栈底部的一个或多个协议层，也可以匹配数据包总的任意指定位置。

Table 8.6 Invalid, missing L3

| **Index** | **Item** |
| --- | --- |
| 0 | Ethernet |
| 1 | UDP |
| 2 | END |

由于L2（以太网）和L4（UDP）之间的L3规范缺失，上述模式无效。也就是说，只允许在堆叠的底部或顶部忽略协议层。

### 8.2.5.元条目类型

元条目只匹配元数据或影响模式处理过程，而不是直接匹配数据包数据，一般不需要规范结构。这种特殊性允许他们在堆栈中的任何位置，而不会对匹配结果造成影响。

##### 8.2.5.1.END条目

条目列表的结束标记。阻止进一步处理条目，从而结束模式匹配。

* 为了方便起见，其数值为0。
* PMD必须强制支持这个条目。
* 忽略spec、last、mask域。

Table 8.7 END

| **Field** | **Value** |
| --- | --- |
| spec | Ignored |
| last | Ignored |
| mask | Ignored |

#### 8.2.5.2.VOID条目

方便起见，用作占位符，被PMD忽略并简单丢弃，跳过不处理。

* PMD必须强制支持这个条目。
* 忽略spec、last、mask域。

Table 8.8 VOID

| **Field** | **Value** |
| --- | --- |
| spec | Ignored |
| last | Ignored |
| mask | Ignored |

此类型条目的一个使用情景是快速生成共享共用前缀的规则，而无需重新分配内存，仅需要更新条目类型。

Table 8.9 TCP, UDP or ICMP as L4

| **Field** | **Item** |
| --- | --- |
| 0 | Ethernet |
| 1 | IPv4 |
| 2 | UDP | VOID | VOID |
| 3 | VOID | TCP | VOID |
| 4 | VOID | VOID | ICMP |
| 5 | END |  |  |

#### 8.2.5.3.INVERT条目

反向匹配，即与模式不匹配的数据包的处理。

* 忽略spec、last、mask域。

Table 8.10 INVERT

| **Field** | **Value** |
| --- | --- |
| spec | ignored |
| last | Ignored |
| mask | Ignored |

下面的使用场景，匹配非TCPv4的报文：  
Table 8.11 Anything but TCPv4

| **Index** | **Item** |
| --- | --- |
| 0 | INVERT |
| 1 | Ethernet |
| 2 | IPv4 |
| 3 | TCP |
| 4 | END |

#### 8.2.5.4.PF条目

匹配寻址到设备物理功能的数据包。

如果底层设备功能与正常接收到匹配流量的功能不同，则指定此项可防止报文到达该设备，除非流规则包含Action: PF。默认情况下，设备实例之间的数据包不会重复。

* 如果条目应用于VF设备，可能返回错误或不匹配任何流量。
* 可以和任意数据的条目组合：VF组合以匹配PF或者VF流量。
* spec、last、mask不能设置。

Table 8.12 PF

| **Field** | **Value** |
| --- | --- |
| spec | unset |
| last | unset |
| mask | unset |

#### 8.2.5.5.VF条目

匹配寻址到设备虚拟功能ID的数据包。

如果底层设备功能与正常接收匹配流量的功能不同，则指定此项可防止报文到达该设备，除非流规则包含Action：VF。默认情况下，设备实例之间的数据包不会重复。

* 如果这导致VF设备匹配到不同VF的流量，则可能返回错误或不匹配任何流量。
* 可以指定多次以匹配寻址到多个VF ID的流量。
* 可以与PF项目组合以匹配PF和VF流量。
* 默认掩码匹配任何VF ID。

Table 8.13 VF

| **Field** | **Subfield** | **Value** |
| --- | --- | --- |
| spec | Id | destination VF ID |
| last | Id | upper range value |
| mask | id | zeroed to match any VF ID |

#### 8.2.5.6.PORT条目

匹配来自指定底层设备物理端口的数据包。

第一个PORT条目覆盖的物理端口通常与指定的DPDK输入端口（port\_id）相关联。该条目可以提供多次以匹配其他物理端口。

请注意，当这些端口不在DPDK控制下时，物理端口不一定与DPDK输入端口（port\_id）绑定。 可能的值是特定于每个设备，它们不一定从零开始，并且可能不是连续的。

作为设备属性，可以通过其他方式检索允许的值列表以及与port\_id关联的值。  
Table 8.14 PORT

| **Field** | **Subfield** | **Value** |
| --- | --- | --- |
| spec | index | destination port index |
| last | index | upper range value |
| mask | index | zeroed to match any port index |

### 8.2.6.数据匹配条目类型

大多数的数据匹配条目是具有位掩码的基本协议头部定义。必须从最底层到最高层协议指定（堆叠方式）以形成匹配模式。

下面的描述并不详尽，将来会添加新的协议。

#### 8.2.6.1.ANY

匹配当前层的任何协议，单个ANY也可以代表多个协议层。  
当在数据包中的任意位置寻找协议时，通常将其指定为第一个模式条目。

* 默认掩码匹配任何数目的协议层。

Table 8.15 ANY

| **Field** | **Subfield** | **Value** |
| --- | --- | --- |
| spec | num | 覆盖的层数 |
| last | num | 最大范围 |
| mask | num | 匹配任意层数 |

例如VXLAN TCP负载外部L3（IPv4或IPv6）及L4（UDP）使用第一个ANY来匹配，内部L3（IPv4或IPv6）用第二个ANY来匹配：

Table 8.16 TCP in VXLAN with wildcards

| **Index** | **Item** | **Field** | **Subfield** | **Value** |
| --- | --- | --- | --- | --- |
| 0 | Ethernet |  |  |  |
| 1 | ANY | Spec | Num | 2 |
| 2 | VXLAN |  |  |  |
| 3 | Ethernet |  |  |  |
| 4 | ANY | Spec | Num | 1 |
| 5 | TCP |  |  |  |
| 6 | END |  |  |  |

#### 8.2.6.2.RAW

匹配指定偏移量下指定长度的字节串。

偏移量可以是是绝对偏移（使用数据包开始）或者相对于堆栈中先前匹配项的结尾，相对偏移量允许为负值。

如果启用搜索，则使用偏移量作为起点。搜索区域可以通过将限制设置为非零值来定界，该值可以是可以开始模式的偏移量后的最大字节数。

允许匹配0长度，这样做会重置后续项目的相对偏移量。

* 这个条目不支持区间（last）。
* 默认的掩码精确匹配所有字段。

Table 8.17 RAW

| **Field** | **Subfield** | **Value** |
| --- | --- | --- |
| spec | relative | 上一个条目之后的搜索模式 |
|  | search | 搜索模式 |
|  | reserved | 预留，必须为0 |
|  | offset | 绝对/相对偏移量 |
|  | limit | 搜索区域限制 |
|  | length | 模式长度 |
|  | pattern | 要匹配的字节串 |
| last | 如果指定，必须全0或者与spec相等 |  |
| mask | 应用于spec |  |

使用组合的RAW条目在UDP有效载荷的各种偏移量下查找几个字符串的示例模式：

| **Index** | **Item** | **Field** | **Subfield** | **Value** |
| --- | --- | --- | --- | --- |
| 0 | Ethernet |  |  |  |
| 1 | IPv4 |  |  |  |
| 2 | UDP |  |  |  |
| 3 | RAW | spec | relative | 1 |
|  |  |  | search | 1 |
|  |  |  | offset | 10 |
|  |  |  | limit | 0 |
|  |  |  | length | 3 |
|  |  |  | pattern | “foo” |
| 4 | RAW | spec | relative | 1 |
|  |  |  | search | 0 |
|  |  |  | offset | 20 |
|  |  |  | limit | 0 |
|  |  |  | length | 3 |
|  |  |  | pattern | “bar” |
| 5 | RAW | spec | relative | 1 |
|  |  |  | search | 0 |
|  |  |  | offset | -29 |
|  |  |  | limit | 0 |
|  |  |  | length | 3 |
|  |  |  | pattern | “baz” |
| 6 | END |  |  |  |

含义如下：

* 在UDP有效载荷内偏移10个自己的地方匹配“foo”。
* 在“foo”之后20个字节的地方匹配“bar”。
* 在“bar”向后退29字节的地方匹配“baz”。

满足这样条件的报文可以表示为如下：

0 >= 10 B == 20 B| |<--------->| |<--------->|

| | | | ||-----|------|-----|-----|-----|-----|-----------|-----|------|| ETH | IPv4 | UDP | ... | baz | foo | ......... | bar | .... ||-----|------|-----|-----|-----|-----|-----------|-----|------|

| |

|<--------------------------->|

== 29 B

注意，匹配模式后续条目将在”baz”之后恢复，而不是“bar”，因为总是在堆栈的先前项之后执行匹配。

#### 8.2.6.3.ETH

匹配以太头部。

* dst：目的MAC。
* src：源MAC。
* type：EtherType。
* 默认掩码仅匹配源MAC和目的MAC。

#### 8.2.6.4.VLAN

匹配802.1Q/ad VLAN tag。

* tpid：标签协议标识符。
* tci：标签控制信息。
* 默认掩码仅匹配tci。

#### 8.2.6.5.IPv4

匹配IPv4头部。注意，IPv4选项由专用模式条目处理。

* hdr：IPv4头部定义（ret\_ip.h）。
* 默认掩码仅匹配源和目的IP地址。

#### 8.2.6.6.IPv6

匹配IPv6头部。注意，IPv6选项由专用模式条目处理。

* hdr：IPv6头部定义（ret\_ip.h）。
* 默认掩码仅匹配源和目的IPv6地址。

#### 8.2.6.7.ICMP

匹配ICMP头部。

* hdr：ICMP头部定义（ret\_icmp.h）。
* 默认掩码仅匹配ICMP类型及代码。

#### 8.2.6.8.UDP

匹配UDP头部。

* hdr：UDP头部定义（ret\_udp.h）。
* 默认掩码仅匹配源端口和目的端口。

#### 8.2.6.9.TCP

匹配TCP头部。

* hdr：UDP头部定义（ret\_tcp.h）。
* 默认掩码仅匹配源端口和目的端口。

#### 8.2.6.10.SCTP

匹配SCTP头部。

* hdr：UDP头部定义（ret\_sctp.h）。
* 默认掩码仅匹配源端口和目的端口。

#### 8.2.6.11.VXLAN

匹配VXLAN头部（RFC 7348）。

* flags：通常是0x80。
* rsvd0：预留，通常为0x00000。
* vni：VXLAN网络标识符。
* rsvd1：预留，通常为0x00。
* 默认掩码仅匹配vni。

#### 8.2.6.12.E\_TAG

匹配IEEE 802.1BR E-Tag。

* tpid：标签协议标识符，通常是0x893F。
* epcp\_edei\_in\_ecid\_b：E-TAG控制信息（E-TCI），E-PCB（3b），E-DEI（1b），ingress E-CDI base（12b）。
* rsvd\_grp\_ecid\_b：reserver（2b），GRP（2b），C-CID base（12b）。
* in\_ecid\_e：ingress E-CID ext。
* ecid\_e：E-CID ext。
* 默认掩码同时匹配GRP和E-CID base。

#### 8.2.6.13.NVGRE

匹配NVGRE头部（RFC7637）。

* c\_k\_s\_rsvd0\_ver：checksum（1b），key bit（1b），seq number（1b），reserved 0（9b），version（3b）。根据RFC7637，这个字段必须是0x2000。
* protocol：协议类型0x6558.
* tni：虚拟子网ID。
* flow\_id：流ID。
* 默认掩码只匹配TNI。

#### 8.2.6.14.MPLS

匹配MPLS头部。

* label\_tc\_s\_ttl：label，TC，Bottom of stack及ttl。
* 默认掩码只匹配label。

#### 8.2.6.15.GRE

匹配GRE头部。

* c\_rsvd0\_ver：checksum，rsvd0，version。
* protocol：协议类型。
* 默认掩码只匹配协议类型。

### 8.2.7.动作

每个可能的动作都由一个类型来表示。一些动作具有相关的配置结构。列表中组合的几个操作可能会影响流规则。列表并不有序。

动作分成以下三种：

* 终结动作（如QUEUE、DROP、RSS、PF、VF），防止后续流程处理匹配的数据包，除非动作被PASSTHRU覆盖。
* 非终结动作（如PASSTHRU、DUP），将匹配的数据包保留以便后续的流程规则进行额外的处理。
* 非终结元动作（END、VOID、MARK、FLAG、COUNT）等不会影响数据包最终结果的动作。

当流规则中组合了多个动作时，他们应该具有不同的类型（如丢弃数据包两次是不可能的）。相同类型的动作，只有最后一个会被考虑到，但是PMD仍然在动作链表中执行错误检查。

类似于匹配模式，动作链表被END条目终结。注意，只有PASSTHUR才能覆盖终结规则。  
下面实例为将数据包重定向到队列索引为10的队列：  
Table 8.18 Queue action

| **Field** | **Value** |
| --- | --- |
| index | 10 |

动作列表示例，其顺序不重要，应用程序必须考虑同时执行所有的动作：  
Table 8.19 Count and drop

| **Field** | **Action** |
| --- | --- |
| 0 | COUNT |
| 1 | DROP |
| 2 | END |

Table 8.20 Mark, count and redirect

| **Index** | **Action** | **Field** | **Valie** |
| --- | --- | --- | --- |
| 0 | MARK | make | 0x2a |
| 1 | COUNT |  |  |
| 2 | QUEUE | queue | 10 |
| 3 | END |  |  |

| **Index** | **Action** | **Field** | **Valie** |
| --- | --- | --- | --- |
| 0 | Drop |  |  |
| 1 | QUEUE | queue | 5 |
| 2 | END |  |  |

Table 8.21 Redirect to queue 5

| **Index** | **Action** | **Field** | **Valie** |
| --- | --- | --- | --- |
| 0 | Drop |  |  |
| 1 | QUEUE | queue | 5 |
| 2 | END |  |  |

在上例中，考虑到同时执行这两个动作，最终的结果只有QUEUE有效果。

Table 8.22 Redirect to queue 3

| **Index** | **Action** | **Field** | **Valie** |
| --- | --- | --- | --- |
| 0 | QUEUE | queue | 5 |
| 1 | VOID |  |  |
| 2 | QUEUE | queue | 3 |
| 3 | END |  |  |

如前所述，仅考虑列表中相同类型的最后一个动作。这个例子也显示了VIOD被忽略。

### 8.2.8.动作类型

通用动作类型将在本节描述。与模式条目类型一样，这个列表并不完全，后续将添加新的动作类型。

#### 8.2.8.1.END

END标记动作列表结束，防止进一步处理动作。

* 方便起见，其数值为0。
* PMD需要强制支持。
* 不配置任何属性。

Table 8.23 END

| **Field** |
| --- |
| no properties |

#### 8.2.8.2.VOID

方便起见，用作占位符。它被PMD简单的忽略处理。

* PMD需要强制支持。
* 不配置任何属性。

Table 8.24 VOID

| **Field** |
| --- |
| no properties |

#### 8.2.8.3.PASSTHUR

将报文留给后续流规则进行额外的处理。当流规则不包含终结动作时，这个操作时默认的，当时可以指定强制规则变为非终止。

* 不配置任何属性。

Table 8.25 PASSTHRU

| **Field** |
| --- |
| no properties | s |

| **Index** | **Action** | **Field** | **Valie** |
| --- | --- | --- | --- |
| 0 | PASSTHUR |  |  |
| 1 | QUEUE | queue | 8 |
| 2 | END |  |  |

将数据包复制到队列并按照后续流规则继续处理的示例：  
Table 8.26 Copy to queue 8

| **Index** | **Action** | **Field** | **Valie** |
| --- | --- | --- | --- |
| 0 | PASSTHUR |  |  |
| 1 | QUEUE | queue | 8 |
| 2 | END |  |  |

#### 8.2.8.4.MARK

将一个整形数值附加到数据包并设置PKT\_RX\_FDIR和PKT\_RX\_FDIR\_ID mbuf标志。

该值的意义可以由应用程序任意指定。最大允许值取决于底层实现。它在hash.fdir.hi mbuf字段中返回。

Table 8.27 MARK

| **Field** | **Value** |
| --- | --- |
| id | 数据包返回的整形数值 |

#### 8.2.8.5.FLAG

标记数据包。与MARK动作类似，只是没有特定值，只能设置PKT\_RX\_FDIR mbuf标志。

Table 8.28 FLAG

| **Field** |
| --- |
| no properties |

#### 8.2.8.6.QUEUE

将报文重定向到指定队列。

* 默认为终结动作。

Table 8.29 QUEUE

| **Field** | **Value** |
| --- | --- |
| index | 队列索引 |

#### 8.2.8.7.DROP

丢弃报文。

* 不配置任何属性。
* 默认为终结动作。
* 如果指定， PASSTHUR可以覆盖这个动作。

Table 8.30 DROP

| **Field** |
| --- |
| no properties |

#### 8.2.8.8.COUNT

在规则中使能计数器。  
这些计数器可以通过rte\_flow\_query()检索和重置，具体请参阅rte\_flow\_query\_count结构体描述。

* 计数器可以通过rte\_flow\_query()检索。
* 不配置任何属性。

Table 8.31 COUNT

| **Field** |
| --- |
| no properties |

检索和重置流量规则计数器的查询结构：  
Table 8.32 COUNT query

| **Field** | **IO** | **Value** |
| --- | --- | --- |
| reset | in | 查询之后重置计数器 |
| hit\_set | out | hits字段设置 |
| bytes\_set | out | bytes字段设置 |
| hits | out | 此规则hit的次数 |
| bytes | out | 此规则经过的bytes |

#### 8.2.8.9.DUP

复制报文给指定的队列。  
这个动作通常与QUEUE结合，但当单独使用时，实际上类似于QUEUE+PASSTHUR。

* 默认为非终结动作。

Table 8.33 DUP

| **Field** | **Value** |
| --- | --- |
| index | 队列索引 |

#### 8.2.8.10.RSS

与QUEUE类似，但是根据参数在数据包上额外执行RSS以将他们扩展到多个队列中。

注意：RSS哈希结果存储在hash.rss mbuf字段中，该字段与hash.fdir.lo重叠。由于MARK动作仅设置hash.fdir.hi字段，因此可以同时查询这两个字段。

* 默认为终结动作。

Table 8.34 RSS

| **Field** | **Value** |
| --- | --- |
| rss\_conf | RSS参数 |
| num | Queue队列中的数目 |
| queue[] | 使用的Queue |

#### 8.2.8.11.PF

将数据包重定向到当前设备的物理功能（PF）。

* 不配置任何属性。
* 默认为终结动作。

Table 8.35 PF

| **Field** |
| --- |
| no properties |

#### 8.2.8.12.VF

将数据包重定向到当前设备的虚拟功能（VF）。  
由VF模式项匹配的数据包可以重定向到其原始VF ID而不是指定的VF ID。如果VF部分与先前的流规则相匹配，或者首先将数据包未发送到VF，则此参数可能不可用，并且不能保证其正常工作。

* 默认为终结动作。

Table 8.36 VF

| **Field** | **Value** |
| --- | --- |
| original | 使用原始的VF ID |
| vf | VF ID |

### 8.2.9.负数类型

所有指定的模式条目（枚举rte\_flow\_item\_type）和动作条目（枚举rte\_flow\_action\_type）都使用正的标识符。

负数空间用于在运行期间由PMD生成的动态类型。PMD可能会遇到他们，但不能接受他们不知道的负数标识符。

生成负数类型的方法有待定义。

### 8.2.10.计划扩展类型

随着新协议的实现，将添加模式条目类型。

通过专用模式项支持变量头，例如为了匹配特定的IPv4选项，IPv6扩展头将在IPv4 / IPv6项目之后堆叠。

其他动作类型已经在计划，但尚未定义。包括以多种方式改变分组数据的能力，例如执行隧道报头的封装/解封装等。

DPDK提供了一个极简的API用于所有的流规则管理。

每个创建的流规则都与PMD中不透明的句柄相关联，应用程序负责维护它，直到流规则被删除为止。

流规则由结构体struct rte\_flow对象表示。

### 8.2.11.验证

鉴于完全表达一组确定的设备功能是不实际的，因此，DPDK提供了专用的函数来检查流规则是否被支持，并且可以被创建。

int rte\_flow\_validate(uint8\_t port\_id,

const struct rte\_flow\_attr \*attr,

const struct rte\_flow\_item pattern[],

const struct rte\_flow\_action actions[],

struct rte\_flow\_error \*error);

这个函数验证流规则是否正确，以及是否可以由设备给定的资源所接受。根据当前设备模式和队列配置检查规则。还可以根据现有流规则和设备资源选择性地验证规则。此函数对目标设备无影响。

由于可能的冲突或资源限制，只要在同一时间没有成功调用rte\_flow\_create()或者rte\_flow\_destroy()，并且没有任何影响流规则的设备参数被修改，那么返回值可以保证有效。

参数：

* port\_id：以太设备端口标识。
* attr：流规则属性。
* pattern：指定模式（以END为结尾的链表）。
* actions：关联动作（以END为结尾的链表）。
* error：如果不为NULL，执行详细错误报告。PMD初始化这个结构以防止出错。

返回值：

* 如果流规则有效，并且可以被创建，那么返回0。否则返回一个负的error值，定义为如下。
* -ENOSYS：底层设备不支持此功能。
* -EINVAL：未知或无效的流规则。
* -ENOTSUP：有效但不支持的规则（例如不支持部分位掩码）。
* EEXIST：与现有规则冲突。仅当设备支持流规则冲突检查并且存在流规则冲突时才返回这个值，如果不处理此返回值，则不能保证创建的流规则不会因为冲突而失败。
* ENOMEM：内存不足，或者设备支持资源验证，但是设备上的资源有限制。
* -EBUSY：由于设备繁忙，无法执行操作，如果受影响的队列甚至整个端口处于停止状态（参考接口rte\_eth\_dev\_rx\_queue\_stop()及rte\_eth\_dev\_stop()），则可能会执行。

8.2.12. 创建  
创建流规则与验证流规则类型，除了实际创建规则并返回句柄。

struct rte\_flow \*

rte\_flow\_create(uint8\_t port\_id,

const struct rte\_flow\_attr \*attr,

const struct rte\_flow\_item pattern[],

const struct rte\_flow\_action \*actions[],

struct rte\_flow\_error \*error);

参数：

* port\_id：以太设备端口标识。
* attr：流规则属性。
* pattern：指定模式（以END为结尾的链表）。
* actions：关联动作（以END为结尾的链表）。
* error：如果不为NULL，执行详细错误报告。PMD初始化这个结构以防止出错。

返回值：  
在创建成功时返回一个有效的句柄，否则为NULL，并且rte\_errno设置为rte\_flow\_vaildate()定义的错误代码的正值。

8.2.13. 销毁  
流规则的删除不是自动完成的，如果任何队列或端口仍然引用他们，则不应该释放。应用程序必须在释放资源之前执行此步骤。

int rte\_flow\_destroy(uint8\_t port\_id,

struct rte\_flow \*flow,

struct rte\_flow\_error \*error);

当其他流规则依赖于它时，删除流规则句柄可能会失败，并且删除操作可能会造成不一致的状态。  
如果句柄与创建顺序相反的顺序执行销毁，则可以保证成功。

参数：

* port\_id：以太设备端口标识。
* flow：要销毁的流规则。
* error：如果不为NULL，执行详细错误报告。PMD初始化这个结构以防止出错。

返回值：

* 成功返回0，失败返回负值，且rte\_errno被设置。

8.2.14. 清空  
一个更加方便的功能，可以用于销毁与port关联的所有流规则。通过连续调用rte\_flow\_destory()来实现。

intrte\_flow\_flush(uint8\_t port\_id,

struct rte\_flow\_error \*error);

当执行失败时（不太可能发生），流规则句柄仍然视为已经销毁，不再有效，但是端口必须处于不一致的状态。

参数：

* port\_id：以太设备端口标识。
* error：如果不为NULL，执行详细错误报告。PMD初始化这个结构以防止出错。

返回值：

* 成功返回0，失败返回负值，且rte\_errno被设置。

### 8.2.15.查询

查询一个存在的流规则。  
此功能允许检索特定于流的数据，如计数器等。数据通过必须存在于流规则定义中的特殊操作来收集。

intrte\_flow\_query(uint8\_t port\_id,

struct rte\_flow \*flow,

enum rte\_flow\_action\_type action,

void \*data,

struct rte\_flow\_error \*error);

参数：

* port\_id：以太设备端口标识。
* flow：要查询的流规则句柄。
* actions：查询的动作类型。
* data：用于存储相关查询数据类型的指针
* error：如果不为NULL，执行详细错误报告。PMD初始化这个结构以防止出错。

返回值：

* 成功返回0，失败返回负值，且rte\_errno被设置。

8.3. 详细错误报告  
对于想要调查与流规则管理有关的问题的用户或应用程序开发人员，定义的错误值可能不够详细。为此目的定义了专门的错误对象：

enum rte\_flow\_error\_type {

RTE\_FLOW\_ERROR\_TYPE\_NONE, /\*\*< No error. \*/

RTE\_FLOW\_ERROR\_TYPE\_UNSPECIFIED, /\*\*< Cause unspecified. \*/

RTE\_FLOW\_ERROR\_TYPE\_HANDLE, /\*\*< Flow rule (handle). \*/

RTE\_FLOW\_ERROR\_TYPE\_ATTR\_GROUP, /\*\*< Group field. \*/

RTE\_FLOW\_ERROR\_TYPE\_ATTR\_PRIORITY, /\*\*< Priority field. \*/

RTE\_FLOW\_ERROR\_TYPE\_ATTR\_INGRESS, /\*\*< Ingress field. \*/

RTE\_FLOW\_ERROR\_TYPE\_ATTR\_EGRESS, /\*\*< Egress field. \*/

RTE\_FLOW\_ERROR\_TYPE\_ATTR, /\*\*< Attributes structure. \*/

RTE\_FLOW\_ERROR\_TYPE\_ITEM\_NUM, /\*\*< Pattern length. \*/

RTE\_FLOW\_ERROR\_TYPE\_ITEM, /\*\*< Specific pattern item. \*/

RTE\_FLOW\_ERROR\_TYPE\_ACTION\_NUM, /\*\*< Number of actions. \*/

RTE\_FLOW\_ERROR\_TYPE\_ACTION, /\*\*< Specific action. \*/

};

struct rte\_flow\_error {

enum rte\_flow\_error\_type type; /\*\*< Cause field and error types. \*/

const void \*cause; /\*\*< Object responsible for the error. \*/

const char \*message; /\*\*< Human-readable error message. \*/

};

错误类型RTE\_FLOW\_ERROR\_TYPE\_NONE代表无错误，在这种情况下可以忽略其他字段。其他错误类型描述了由cause指向的对象类型。

如果cause不为NULL，cause指向的对象用于描述错误。对于一个流规则，这个对象可能是个模式条目或单个动作。

该对象通常由应用程序分配，并在PMD发生错误的情况下由PMD设置，消息指向不需要被应用程序释放的常量字符串，但是只要它的关联DPDK端口保持配置。关闭底层设备或卸载PMD使其无效。

## 8.4.注意事项

* DPDK不会自动跟踪流规则定义或流规则对象。应用程序必须跟踪后者，也可以跟踪前者。PMD每部也可以实现，但是应用程序不能依赖于此。
* 在连续的端口初始化之间不保留流规则。应用程序退出而没有释放的话，重启时必须重新创建。
* API操作是同步和阻塞的（EAGAIN不能作为返回值）。
* 尽管没什么措施来防止不同的设备被同时配置，仍然没有方法保证重入/多线程安全，需要特别注意。
* 管理流规则时，不需要停止数据路径（RX/TX）。如果不能自然地实现或解决，则必须返回适当的错误代码（EBUSY）。
* PMD负责在停止和重新启动端口或执行可能影响端口的其他操作时维护流规则配置，而不是应用程序。但是流规则只能被应用程序明确地销毁。

对于暴露多个端口共享流规则影响的全局设置的设备：

* DPDK控制下的多有端口必须一致，PMD负责确保端口上的现有流规则不受其他端口的影响。
* 不受DPDK控制的端口，由用户负责处理。他们可能会影响现有的流规则并导致未定义的行为。感知这种情况的PMD可能会阻止流规则的创建。

## 8.5.PMD接口

PMD接口在rte\_flow\_driver.h中定义。它不受API/ABI版本控制的约束，因为它不会暴露于应用程序，并可能独立发展。

目前，它通过过滤器类型RTE\_ETH\_FILTER\_GENERIC实现了传统过滤框架，该类型接受单个操作RTE\_ETH\_FILTER\_GET以返回包含在struct rte\_flow\_ops内的PMD特定的rte\_flow回调。

为了保持与遗留过滤框架的兼容性，这个开销是暂时必需的，但是这些框架应该最终消失。

* PMD回调完全实现了规则管理中描述的接口，除了已经转换为指向底层结构rte\_eth\_dev的指针的端口ID参数外。
* 在调用PMD函数之前，公共API函数根本不处理流规则定义（无基本错误检查，无任何验证）。 他们只确保这些回调非NULL或返回ENOSYS（不支持功能）错误。

此接口另外定义了以下帮助函数：

* rte\_flow\_ops\_get()：从端口获取泛型流操作结构。
* rte\_flow\_error\_set()：初始化通用流错误结构。

## 8.6.设备兼容性

没有已知的实现可以支持所有描述的功能。

出于性能原因，PMD不支持的功能或组合不会在软件中完全模拟实现。硬件执行大部分工作（例如队列重定向和数据包识别），部分支持的功能在软件中完成。

只要这样做不会影响现有流规则的行为，PMD将尽力通过解决硬件限制来满足应用程序请求。

以下部分提供了这种情况的一些示例，并描述了PMD如何处理它们，它们基于之前的API中内置的限制。

### 8.6.1.全局的bit-mask

每个流规则都带有自己的每层位掩码，而硬件可能仅支持给定层类型的单个设备范围的位掩码，以便两个IPv4规则不能使用不同的位掩码。  
在这种情况下，预期的行为是PMD根据创建的第一个流规则的需要自动配置全局位掩码。  
仅当其位掩码与其匹配时才允许后续规则，否则应返回EEXIST错误代码。

### 8.6.2.不支持的layer类型

可以通过使用RAW类型条目来模拟许多协议。  
PMD可以依靠此功能来模拟对不被硬件直接识别的头部的协议的支持。

### 8.6.3.ANY 模式条目

这种模式条目代表任何东西，这可能难以转换为硬件理解的信息，特别是如果跟随更具体的类型。

考虑如下的模式：  
Table 8.37 Pattern with ANY as L3

| **Index** | **Item** |
| --- | --- |
| 0 | Ethernet |
| 1 | ANY | num | 1 |
| 2 | TCP |  |  |
| 3 | END |  |  |

我们知道TCP对IPv4和IPv6之外的其他东西并不合适，这样的模式可能被转换为两个流规则：  
Table 8.38 ANY replaced with IPV4

| **Index** | **Item** |
| --- | --- |
| 0 | Ethernet |
| 1 | IPv4(zeroed mask) |
| 2 | TCP |
| 3 | END |

Table 8.39 ANY replaced with IPV6

| **Index** | **Item** |
| --- | --- |
| 0 | Ethernet |
| 1 | IPv6(zeroed mask) |
| 2 | TCP |
| 3 | END |

请注意，一旦任何规则覆盖了多个层次，这种方法可能会产生大量隐藏的流规则。 因此建议仅支持最常见的情况（ANY处于L2和/或L3）。

### 8.6.4.不支持的动作

* 当与QUEUE动作组合时，只要目标队列由单个规则使用，可以在软件中实现数据包计数（COUNT动作）和标记（MARK动作或FLAG动作）。
* 指定动作DUP + QUEUE的规则可能会转换为两个隐藏的规则QUEUE和PASSTHRU动作。
* 当提供单个目标队列时，动作RSS也可以通过QUEUE来实现。

### 8.6.5.流规则优先级

虽然自然是有意义的，但是由于以下几个原因，流量规则不能被假定为与其创建相同的顺序由硬件处理：

* 它们可以作为树或哈希表而不是列表在内部进行管理。
* 在添加另一个流规则之前删除流规则可以将新规则放在列表的末尾或重新使用释放的条目。
* 当数据包被几个规则匹配时，可能会发生重复。

对于重叠的规则（特别是为了使用动作PASSTHRU），可预测的行为只能通过使用不同的优先级来保证。

优先级不一定在硬件中实现，或者可能受到严重限制（例如单个优先级位）。

由于这些原因，优先级可以纯粹在PMD的软件中实现。

* 对于期望以正确顺序添加流规则的设备，PMD可能会在添加具有较高优先级的新规则之后破坏并重新创建现有规则。
* 可以在初始化时间创建可配置数量的空或空规则，以节省高优先级槽位供以后使用。
* 为了节省优先级，PMD可以评估规则是否可能相应地发生冲突并调整其优先级。

## 8.7.未来演变

* 设备配置文件选择功能，可用于强制永久性配置文件，而不是依靠现有流程规则自动配置。
* 通过PMD即时配置生成的特定模式条目和动作类型优化rte\_flow规则的方法。DPDK应该为这些类型分配负数值，以便不与现有类型相冲突。具体参考负数类型。
* 添加指定Egress模式条目和动作，如Traffic direction中所描述。
* PMD无法处理所请求的流规则时，可以选择软件实现作为后备，使得应用程序无需自己实现。

## 8.8.API迁移

这一部分描述在rte\_eth\_ctrl.h中找到的已弃用的过滤器类型（通常以RTE\_ETH\_FILTER\_为前缀）的完整列表以及将其转换为rte\_flow规则的方法。

### 8.8.1.MACVLAN to ETH?VF，RF

MACVLAN可以转换为基本条目：ETH流规则，终止操作为VF或PF。

Table 8.40 MACVLAN conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | ETH | Spec | Any | VF、PF |
|  |  | Last | N/A |  |
|  |  | Mask | Any |  |
| 1 | END |  |  | END |

### 8.8.2.ETHERTYPE to ETH?QUEUE，DROP

ETHERTYPE可以看成是条目：具有终止操作的ETH流规则，动作为QUEUE或DROP。

Table 8.41 ETHERTYPE conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | ETH | Spec | Any | QUEUE、DROP |
|  |  | Last | N/A |  |
|  |  | Mask | Any |  |
| 1 | END |  |  | END |

### 8.8.3.FLEXIBLE to RAW?QUEUE

FLEXIBLE可以转换为一个条目：RAW模式，终止动作为QUEUE和定义的优先级。

Table 8.42 FLEXIBLE conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | RAW | Spec | Any | QUEUE |
|  |  | Last | N/A |  |
|  |  | Mask | Any |  |
| 1 | END |  |  | END |

### 8.8.4.SYN to TCP?QUEUE

SYN条目：只有syn位启用和屏蔽的TCP规则，以及终止动作QUEUE。

优先级可以设置为模拟高优先级位。

Table 8.43 SYN conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | ETH | Spec | unset | QUEUE |
|  |  | Last | unset |  |
|  |  | Mask | unset |  |
| 1 | IPv4 | Spec | unset | END |
|  |  | Last | unset |  |
|  |  | Mask | unset |  |
| 2 | TCP | Spec | Syn | 1 |
|  |  | Mask | Syn | 1 |
| 3 | END |  |  |  |

### 8.8.5.NTUPLE to IPV4，TCP，UDP?QUEUE

NTUPLE类似于指定一个空的L2， IPV4作为L3， TCP或UDP做为L4，终止动作为QUEUE。  
也可以指定优先级。

Table 8.44 NTUPLE conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | ETH | Spec | unset | QUEUE |
|  |  | Last | unset |  |
|  |  | Mask | unset |  |
| 1 | IPv4 | Spec | any | END |
|  |  | Last | unset |  |
|  |  | Mask | any |  |
| 2 | TCP/UDP | Spec | any |  |
|  |  | Last | unset |  |
|  |  | Mask | any |  |
| 3 | END |  |  |  |

### 8.8.6.TUNNEL to ETH，IPV4，IPV6，VXLAN?QUEUE

TUNNEL匹配常见的基于IPv4和IPv6 L3 / L4的隧道类型。  
在下表中，ANY条目用于覆盖可选的L4。

Table 8.45 TUNNEL conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | ETH | Spec | any | QUEUE |
|  |  | Last | unset |  |
|  |  | Mask | unset |  |
| 1 | IPv4/IPv6 | Spec | any | END |
|  |  | Last | unset |  |
|  |  | Mask | any |  |
| 2 | ANY | Spec | any |  |
|  |  | Last | unset |  |
|  |  | Mask | num 0 |  |
| 3 | VXLAN, GENEVE, TEREDO, NVGRE, GRE, ... | Spec | any |  |
|  |  | Last | unset |  |
|  |  | Mask | any |  |
| 3 | END |  |  |  |

### 8.8.7.FDIR to most item types?QUEUE，DROP，PASSTHRU

FDIR比任何其他类型更复杂，有几种方法来模拟其功能。 大部分在下表中进行了总结。  
部分功能有意不实现支持：

* 配置整个设备的匹配输入集和掩码的能力，PMD应根据请求的流规则自动处理。  
  例如，如果设备每个协议类型仅支持一个位掩码，则源/地址IPv4位掩码可以通过第一个创建的规则变成不可变的。 随后的IPv4或TCPv4规则只能在兼容的情况下创建。  
  请注意，只有现有流规则影响的协议位掩码是不可变的，其他可以稍后更改。相关的流量规则被破坏后，它们再次变得可变。
* 使用flex字节过滤时返回四个或八个字节的匹配数据。虽然具体的操作可以实现它，但它与支持它的设备上更有用的32位标记冲突。
* 对整个设备的RSS处理的副作用。不允许与当前设备配置冲突的流规则。 同样，当它影响现有流规则时，不应允许设备配置。
* 设备操作模式。“none”不受支持，因为只要存在流规则，就不能禁用过滤。
* 应根据创建的流规则自动设置“MAC VLAN”或“隧道”完美匹配模式。
* 签名模式的操作未定义，但如果需要，可以通过特定的项目类型处理。

Table 8.46 FDIR conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | ETH/RAW | Spec | any | QUEUE、DROP、PASSTHRU |
|  |  | Last | N/A |  |
|  |  | Mask | any |  |
| 1 | IPv4/IPv6 | Spec | any | MARK |
|  |  | Last | N/A |  |
|  |  | Mask | any |  |
| 2 | TCP、UDP、SCTP | Spec | any | END |
|  |  | Last | N/A |  |
|  |  | Mask | any |  |
| 3 | VF、RF | Spec | any |  |
|  |  | Last | N/A |  |
|  |  | Mask | any |  |
| 3 | END |  |  |  |

### 8.8.8.HASH

没有这个过滤器类型的对应物，因为它转换为全局设备设置而不是模式项。设备设置根据创建的流规则自动设置。

### 8.8.9.L2\_TUNNEL to VOID?VXLAN

所有数据包都匹配。这种类型改变了传入的数据包，将它们封装在选定的隧道类型中，也可以将它们重定向到VF。

可以使用动作DUP使用其他流规则来模拟基于标签的转发的目标池。

Table 8.47 L2\_TUNNEL conversion

| **Pattern** |  |  |  | **Action** |
| --- | --- | --- | --- | --- |
| 0 | VOID | Spec | N/A | VXLAN, GENEVE, ... |
|  |  | Last | N/A |  |
|  |  | Mask | N/A |  |
| 1 | END | VF |  |  |
| 2 | END | END |  |  |

原文链接：[http://www.jianshu.com/p/405c99678e23](https://www.jianshu.com/p/405c99678e23" \t "_blank) 。