OpenFlow交换机规格

版本1.5.1（协议版本0x06）

2015年3月26日

ONF TS-025

免责声明

本规范按“原样”提供，没有任何保证，
包括任何适销性，非侵权性，适用性的保证-

概需保任何担或特殊规格或求例生任何其他ONF
主张与使用有关的所有责任，包括侵犯任何专有权的责任
本规范中以及本规范实施中的信息
对替代品或服务的采购成本，利润损失，
使用，数据丢失或任何偶然的，后果性的，直接的，间接的或特殊的损害，无论是否
根据合同，侵权，担保或其他方式，以任何不使用或依赖的方式产生
规范或此处的任何信息。

禁止反言或以其他方式明示或暗示地授予任何Open Networking Foundation或
此处授予Open Networking Foundation成员知识产权。

ONF特此授予复制和复制本规范以用于
限内部使用。

请通过以下网址与开放网络基金会联系：http：//www.opennetworking.org。
通过成员资格协议进行规范许可。

在不限制上述免责声明的前提下，本公开说明
网络基金会（“ ONF”）享有免版税，
的合理和非歧视性（RANDZ）许可承诺
根据ONF的成员TO THE ONF知识产权 PROPERTY
权利政策。ONF并不保证所有必要的索赔
本规范的实施可能暗含的专利
ONF成员及其拥有或拥有的许可
服从兰兹成员的承诺。

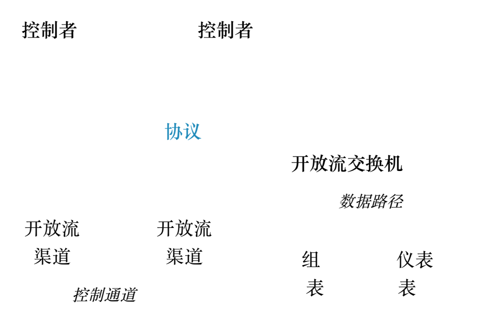
内容

表清单

图清单

1引言

本文档介绍了OpenFlow逻辑交换机的要求。附加信息
Open Networking Foundation上提供了描述OpenFlow和软件定义网络的信息
网站（https://www.opennetworking.org/）。本规范涵盖了组件和基本
交换机的功能以及OpenFlow交换机协议，以从
远程OpenFlow控制器。



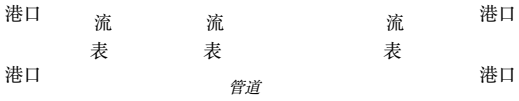


图1：OpenFlow交换机的主要组件。

2个开关组件

OpenFlow逻辑交换机由一个或多个*流表*和一个*组表组成*，它们执行数据包查找和转发，以及到外部控制器的一个或多个*OpenFlow通道*（图1）。的交换机与控制器通信，并且控制器通过OpenFlow交换机管理交换机协议。

使用OpenFlow交换协议，控制器可以在流中添加，更新和删除*流条目*表（主动响应（响应数据包）和主动）。交换机中的每个流表包含一组流条目；每个流条目均包含*匹配字段*，*计数器*和一组适用的*指令*匹配数据包（请参阅5.2）。

匹配从第一个流表开始，并可能继续到管道的其他流表（请参见5.1）。流条目按优先级顺序匹配数据包，每个表中的第一个匹配条目为使用（请参阅5.3）。如果找到匹配的条目，则与特定流条目关联的指令为执行（请参阅5.5）。如果在流表中未找到匹配项，则结果取决于配置

table-miss流条目：例如，可以通过OpenFlow将数据包转发到控制器通道，已删除或可能继续到下一个流表（请参阅5.4）。

与每个流条目关联的指令要么包含动作，要么修改管道处理（请参阅5.5）。指令中包含的操作描述了数据包转发，数据包修改和组表处理。管道处理指令允许将数据包发送到后续表以进一步处理处理并允许以元数据形式在表之间传递信息。表当与匹配流条目关联的指令集未指定时，流水线处理停止一张桌子 此时，通常会修改并转发数据包（请参阅5.6）。

流条目可以转发到*端口*。这通常是物理端口，但也可能是逻辑端口由交换机定义或由本规范定义的保留端口（请参阅4. 1）。保留的端口可能指定通用转发操作，例如发送到控制器，洪泛或使用非OpenFlow方法，例如“常规”交换机处理（请参阅4. 5），而交换机定义的逻辑端口可以指定链路聚合组，隧道或环回接口（请参阅4.4）。

与流条目关联的动作还可以将数据包定向到一个组，该组指定其他处理（请参阅5.10）。组代表洪水的动作集以及更复杂的转发语义（例如，多路径，快速重新路由和链路聚合）。作为间接的一般层，组还使多个流条目能够转发到单个标识符（例如，将IP转发到公共下一个标识符）跳）。这种抽象允许有效地更改跨流条目的常见输出操作。

组表包含组条目；每组条目包含的列表*作用桶*特定语义取决于组类型（请参阅5 .10.1）。应用了一个或多个动作存储桶中的动作发送到组的数据包。

开关设计师可以自由地以任何方便的方式实施内部结构，前提是正确匹配并且保留了指令语义。例如，虽然流条目可以使用全组来转发到多个端口，交换机设计人员可以选择将其实现为单个位掩码硬件转发表。另一个例子是匹配。OpenFlow公开的管道交换机可以通过不同数量的硬件表在物理上实现。

3词汇表

本节描述关键的OpenFlow规范术语。大多数术语特定于此规范。

- **操作**：对数据包执行的操作。一个动作可能会将数据包转发到端口，进行修改数据包（例如减少TTL字段）或更改其状态（例如将其与队列）。大多数动作都包含参数，例如set-field动作包含字段类型和字段值。可以将动作指定为与流条目相关联的指令集的一部分，或者与组条目相关联的操作存储区中。动作可以累积在动作集中包的包或立即应用于包（请参阅5.8）。

• **动作列表**：可包括在流条目动作的有序列表*适用—*

*动作*指令或包出消息，并按列表顺序立即执行
（请参阅5.7）。列表中的动作可以重复，其作用是累积的。

• **操作集**：*Write-Actions*指令的流条目中包含的*一组操作*，
被添加到操作集中，或按操作集顺序执行的组操作桶中（请参阅
5.6）。一组动作只能发生一次。

• **Action Bucket**：组中的一组动作。该小组将为每个小组选择一个或多个桶
包。

• **操作集**：与数据包相关联的一组操作，在数据包处于
由每个表处理，并在指令集终止时按指定顺序执行
管道处理（参见5.6）。

• **字节**：8位八位位组。

• **连接**：网络连接，在交换机和连接器之间传送OpenFlow消息。
可以使用各种网络传输协议来实现（见6 .3）。一个OpenFlow
通道有一个主连接，并有多个辅助连接（请参见6 .3.8）。

• **控制通道**：OpenFlow逻辑交换机的组件的聚合，这些组件负责-
与控制器之间的通讯老化。控制通道每个通道包含一个OpenFlow通道
OpenFlow控制器。

• **控制器**：请参阅OpenFlow控制器。

• **计数器**：计数器是OpenFlow统计信息的主要元素，并且在各种情况下进行累加
管道的特定点，例如在端口或流入口（请参阅5.9）。典型的柜台
计算通过OpenFlow元素传递的数据包和字节数，但是其他
还定义了计数器类型。

•**数据路径**：OpenFlow逻辑交换机中直接位于内部的组件的聚集。
参与流量处理和转发。数据路径包括流表的管道，
组表和端口。

• **流条目**：流表中用于匹配和处理数据包的元素。它包含一组
用于匹配数据包的匹配字段，用于匹配优先级的优先级，要跟踪的一组计数器
数据包，以及一组适用的说明（请参阅5.2）。

• **流程图**：管道的一个阶段。它包含流条目。

• **转发**：确定数据包的输出端口或一组输出端口，并进行传输
数据包到那些输出端口。

• **组**：操作桶的列表以及选择其中一个或多个要应用的桶的方法
基于每个数据包（请参阅5.10）。

• **标头**：嵌入在数据包中的控制信息，交换机使用它来识别数据包和
通知交换机如何处理和转发数据包。标头通常包括
各种标头字段，用于标识数据包的源和目的地，以及如何解释
其他标头和有效载荷。

• **标头字段**：数据包标头中的值。解析包头以提取其头
与相应的匹配字段匹配的字段。

• **混合**：集成OpenFlow操作和*常规*以太网交换操作（请参阅5.1）。

• **指令**：指令附在流条目上，并描述了

数据包与流条目匹配时发生。一条指令可以修改管道处理，
如引导分组到另一个流表，或包含一个*组*的行动，以添加到操作
设置或包含要立即应用于数据包的操作的*列表*（请参阅5.5）。

- **指令集**：流表中的流条目附带的一组指令。
- **匹配字段**：流条目的一部分，数据包与之匹配。匹配字段可以匹配
  各种数据包头字段（请参见7.2.3.3.8），数据包入口端口，元数据值和
  等管道领域（见7.2。 3.9）。匹配字段可以是通配符（匹配任何值），
  在某些情况下是位屏蔽的（匹配位的子集）。
- **匹配**：将数据包的标头字段和管道字段的集合与匹配字段进行比较
  流条目的编号（请参阅5.3）。
- **元数据**：可屏蔽的寄存器，用于将信息从一个表传送到下一个表。
- **消息**：通过OpenFlow连接发送的OpenFlow协议单元。可能是一个请求，一个回复，
  控制消息或状态事件。
- **仪表**：可以测量和控制数据包速率的交换元件。仪表触发
  如果通过仪表的数据包速率或字节速率超过预定义的仪表频段
  阈值（请参阅5.11）。如果仪表带丢弃数据包，则称为**速率限制器**。
- **OpenFlow Channel**：用于OpenFlow交换机和OpenFlow控制器之间的接口
  由控制器来管理开关。
- **OpenFlow Controller**：使用OpenFlow与OpenFlow交换机交互的实体
  交换协议。在大多数情况下，OpenFlow Controller是控制许多OpenFlow的软件。
  逻辑开关。
- **OpenFlow逻辑交换机**：一组OpenFlow资源，可以作为单个实体进行管理，
  包括数据路径和控制通道。
- **OpenFlow协议**：此规范定义的协议。也称为OpenFlow交换机
  协议。
- **OpenFlow交换机**：请参阅OpenFlow逻辑交换机。
- **数据包**：一系列字节，依次包含标题，有效负载和可选的尾部，
  并视为用于处理和转发的单位。默认的数据包类型是
  以太网，还支持其他数据包类型（请参见7 .2.3.11）。
- **管道**：提供匹配，转发和数据包修改的链接流表集
  在OpenFlow开关中（请参阅5.1）。
- **管道字段**：管道处理期间附加到数据包的一组值，这些值不是
  标头字段。包括入口端口，元数据值，Tunnel-ID值等（请参阅
  7.2.3.9）。
- **端口**：数据包进入和退出OpenFlow管道的位置（请参见4.1）。可能是物理端口，
  逻辑端口或OpenFlow交换协议定义的保留端口。
- **队列**：根据数据包在输出端口上的优先级调度数据包，以提供质量
  服务（QoS）。

- **开关**：请参阅OpenFlow逻辑开关。
- **标签**：可以通过推送和弹出操作将其插入或从数据包中删除的标头。
- **最外层标签**：最接近数据包开头的标签。

4个OpenFlow端口

本节介绍OpenFlow端口抽象和支持的各种类型的OpenFlow端口。
由OpenFlow移植。

4.1 OpenFlow端口

OpenFlow端口是用于在OpenFlow处理与
网络的其余部分。OpenFlow交换机通过其OpenFlow端口在逻辑上相互连接，
只能通过输出将数据包从一个OpenFlow交换机转发到另一个OpenFlow交换机
第一个交换机上的OpenFlow端口，第二个交换机上的入口OpenFlow端口。

OpenFlow交换机使许多OpenFlow端口可用于OpenFlow处理。一套

OpenFlow端口可能与交换机硬件提供的网络接口集不同，
某些网络接口可能已禁用OpenFlow，并且OpenFlow开关可能会定义其他
OpenFlow端口。

OpenFlow数据包在入口端口上接收，并由OpenFlow管道处理（请参阅5.1）
可以将它们转发到输出端口。数据包入口端口是数据包的属性
在整个OpenFlow管道中，代表接收数据包的OpenFlow端口
进入OpenFlow开关。匹配数据包时，可以使用入口端口（请参阅5. 3）。公开赛
流管道可以决定使用输出操作（参见5.8）在输出端口上发送数据包，该操作
定义数据包如何返回网络。

OpenFlow交换机必须支持三种类型的OpenFlow端口：*物理端口*，*逻辑端口*和
保留端口。

4.2标准端口

OpenFlow **标准端口**定义为物理端口，逻辑端口和保留的LOCAL
端口（如果支持）（不包括其他保留端口）。

标准端口可以用作入口和输出端口，它们可以成组使用（请参阅5.10），它们
具有端口计数器（请参阅5.9），并且具有状态和配置（请参见7.2.1）。

4.3物理端口

OpenFlow **物理端口**是与交换机的硬件接口相对应的交换机定义的端口。
开关。例如，在以太网交换机上，物理端口一对一映射到以太网接口。

在某些部署中，可以在交换机硬件上虚拟化OpenFlow交换机。在那种情况下
OpenFlow物理端口可以表示虚拟机对应硬件接口的虚拟片
开关。

4.4逻辑端口

OpenFlow **逻辑端口**是交换机定义的端口，与硬件不直接对应
交换机的接口。逻辑端口是可以在交换机中定义的更高级别的抽象
使用非OpenFlow方法（例如，链路聚合组，隧道，环回接口）。

逻辑端口可以包括数据包封装，并且可以映射到各种物理端口。加工
逻辑端口完成的操作取决于实现，并且必须对OpenFlow处理透明，
这些端口必须与OpenFlow处理（例如OpenFlow物理端口）进行交互。

*物理端口*和*逻辑端口*之间的唯一区别是与*逻辑端口*关联的数据包
端口可以具有称为一个额外的管线现场*隧道-ID*与它相关联（见7.2.3。 9），并且当分组
逻辑端口上接收到的逻辑端口及其基础物理端口均发送到控制器
被报告给控制器（见7.4.1）。

4.5预留端口

OpenFlow **保留端口**由该规范定义。他们指定通用转发
诸如发送到控制器，洪泛或使用非OpenFlow方法转发等操作，例如
作为“正常”开关处理。

不需要交换机支持所有保留的端口，只需支持下面标记为“ *Required”的*端口即可。

- *必需：* **ALL**：代表交换机可用于转发特定数据包的所有端口。能够
  仅用作输出端口。在这种情况下，数据包的副本将开始对所有数据包进行出口处理。
  标准端口，不包括数据包入口端口和配置为OFPPC_NO_FWD的端口。
- *必需：* **CONTROLLER**：代表带有OpenFlow控制器的控制通道。能够
  用作入口或输出端口。当用作输出端口时，封装
  数据包中的*数据包*，然后使用OpenFlow交换协议将其发送（请参见7.4。1）。什么时候
  用作入口端口，它标识源自控制器的数据包。

- *必需*：**TABLE**：代表OpenFlow管道的开始（请参阅5.1）。该端口仅在发出*打包*消息的操作列表中的输出操作中有效（请参见7.3.6），并提交数据包到第一个流表，以便可以通过常规OpenFlow处理数据包管道。
- *必需*：**IN PORT**：代表数据包入口端口。只能用作输出端口，通过其入口端口发送数据包。

- *必需*：**ANY**：未指定端口（即端口）时，在某些OpenFlow请求中使用的特殊值被通配）。一些OpenFlow请求包含对该请求的特定端口的引用仅适用于。在这些请求中使用*ANY*作为端口号允许该请求实例适用于所有端口。既不能用作入口端口，也不能用作输出端口。
- *必需*：**UNSET**：特殊值，用于指定尚未在输出端口中设置输出端口。*动作集*。仅在尝试通过以下方式匹配操作集中的输出端口时使用OXM_OF_ACTSET_OUTPUT匹配字段（请参阅7 .2.3.7）。既不能用作入口端口，也不能用作入口端口作为输出端口。
- *可选*：**LOCAL**：代表交换机的本地网络堆栈及其管理堆栈。可用作入口或输出端口。本地端口使远程实体能够通过OpenFlow网络而不是通过交换机与交换机及其网络服务进行交互单独的控制网络。有了一组适当的流条目，就可以用来实现一个带内控制器连接（这不在本规范范围内）。
- *可选*：**NORMAL**：代表使用传统的非OpenFlow管道转发开关（请参阅5. 1）。只能用作输出端口并使用普通端口处理数据包管道。通常将桥接或路由数据包，但是实际结果是实现依赖。如果交换机无法将数据包从OpenFlow管道转发到正常管道，它必须指示它不支持此操作。
- *可选*：**FLOOD**：表示使用交换机的传统非OpenFlow管道进行泛洪（请参阅5. 1）。只能用作输出端口，实际结果取决于实现。在general会将数据包发送出所有标准端口，但不会发送到入口端口，也不会发送到处于OFPPS_BLOCKED状态。交换机还可以使用数据包VLAN ID或其他条件来选择哪个端口用于泛洪。

*仅OpenFlow的*交换机不支持**NORMAL**端口和**FLOOD**端口，而*OpenFlow则是混合*端口开关可能支持它们（请参阅5 .1）。将数据包转发到**FLOOD**端口取决于交换机实施和配置，同时使用一组类型进行*所有*转发时，控制器可以更灵活地实施泛洪（请参阅5.10.1）。

4.6端口变更

交换机配置（例如使用OpenFlow配置协议）可以添加或删除可随时从OpenFlow交换机连接端口。交换机可能会根据以下情况更改端口状态：基础端口机制，例如，如果链路断开（请参阅7.2.1）。控制器或交换机配置可能会更改端口配置（请参阅7.2.1）。对端口状态的任何此类更改或配置必须传达给OpenFlow控制器（请参阅7 .4.3）。

端口的添加，修改或删除永远不会更改流表的内容，尤其是流引用那些端口的条目不会被修改或删除（流条目可以通过比赛或动作）。转发到不存在端口的数据包将被丢弃（请参阅5 .6）。同样，港口另外，修改和删除永远不会更改组表的内容，但是行为某些组中的某些组可能会通过活动检查进行更改（请参见6.7）。

如果端口被删除，并且以后将其端口号重新用于其他物理或逻辑端口，则任何仍然引用该端口号的其余流条目或组条目可以有效地重新定位

到新端口，可能会产生不良结果。因此，删除端口后，将其留给
如果需要，控制器清除所有引用该端口的流条目或组条目。

4.7端口再循环

可以选择使用逻辑端口在OpenFlow中插入网络服务或复杂处理
开关（请参阅4.4）。通常，发送到逻辑端口的数据包永远不会返回同一OpenFlow交换机，
它们要么被逻辑端口占用，要么最终通过物理端口发送。在其他情况下，
发送到逻辑端口的数据包在逻辑端口之后重新分配到OpenFlow交换机
处理。

通过逻辑端口的数据包再循环是可选的，并且OpenFlow支持多种类型的端口重新
循环。最简单的再循环是在逻辑端口上发送的数据包返回到
通过相同的逻辑端口进行切换。这可以用于回送或单向数据包处理。
端口对之间也可能发生再循环，其中在逻辑端口上发送的数据包会返回
通过一对中的另一个逻辑端口进入交换机。这可以用来表示隧道端点或
双向数据包处理。端口属性描述端口之间的再循环关系
（请参阅7.2.1.2）。

使用端口再循环时，交换机应保护自己免受数据包无限循环的侵害。这种机制
是特定于实现的，不在本规范范围之内。例如，开关可以
在每个数据包上附加一个内部再循环计数，该计数随每次再循环而增加，并且
交换机丢弃其计数器高于交换机定义的阈值的数据包。控制器很强大
鼓励避免生成可能产生再循环回路的流条目组合。

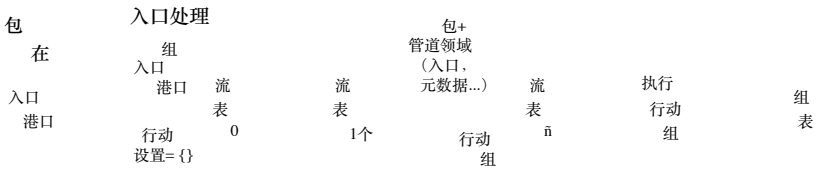
由于可能的处理范围很广，因此可以认为关于再循环的数据包几乎没有。
重新回到交换机。再循环的数据包返回到管道的第一个流表（请参阅5.1）
并可以通过其新的输入端口进行标识。数据包标题可能已更改，因此匹配
字段不能保证相同。逻辑端口可能会进行数据包分段和重组，
因此数据包可能不一对一匹配，并且可能具有不同的大小。

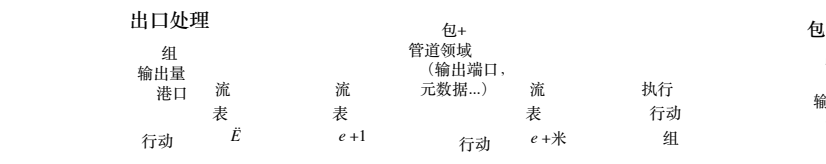
与数据包关联的*Tunnel-ID*字段和其他一些管道字段可以选择为
通过再循环保存，并在返回交换机时可用于匹配，
保留的管道字段通过端口匹配字段属性指示（请参见7.2.1.2）。如果一个
输出端口的OFPPDPT_PIPELINE_OUTPUT属性和

返回端口的OFPPDPT_PIPELINE_INPUT属性，然后使用
数据包（其值必须保持不变）。

5个OpenFlow表

本节介绍流表和组表的组成部分，以及
匹配和动作处理。





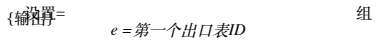


图2：通过处理管道的数据包流。

5.1管道处理

符合OpenFlow的交换机有两种类型：*仅OpenFlow*和*OpenFlow-hybrid*。开放流
仅交换机仅支持OpenFlow操作，在这些交换机中，所有数据包均由
OpenFlow管道，否则无法处理。

OpenFlow混合交换机支持OpenFlow操作和*常规*以太网交换操作，
，例如传统的L2以太网交换，VLAN隔离，L3路由（IPv4路由，IPv6路由...），
ACL和QoS处理。这些开关应在Open-
流量的路由流量*要么*开流管道*或*正常的管道。例如，一个开关
可以使用VLAN标记或数据包的输入端口来决定是否使用一个
管道或其他管道，也可以将所有数据包定向到OpenFlow管道。此分类机制-
无神论超出了本规范的范围。OpenFlow混合交换机也可能允许数据包
通过保留的*NORMAL*和*FLOOD*从OpenFlow管道转到普通管道
端口（请参阅4.5）。

每个OpenFlow逻辑交换机的**OpenFlow管道**包含一个或多个流表，每个流
包含多个流条目的表。OpenFlow管道处理定义了数据包如何交互
这些流表（请参见图2）。一个OpenFlow交换机需要至少一个入口流
表，并且可以选择具有更多流表。只有一个流表的OpenFlow交换机是
在这种情况下，流水线处理大大简化了。

OpenFlow交换机的流表按照可以被数据包遍历的顺序编号，

从0开始。流水线处理分为两个阶段：*入口处理*和*出口处理*。的
第一个出口表（请参见7.3。2）指示了两个阶段的分离，所有表均带有数字
小于第一个出口表的表必须用作入口表，并且没有任何表的数字大于
等于第一个出口表可以用作入口表。

管道处理始终从第一个流表的入口处理开始：数据包必须是
首先与流表0的流条目匹配（请参见图.3）。其他入口流表可能是
根据第一张表中比赛的结果使用。如果入口处理的结果是
为了将数据包转发到输出端口，OpenFlow交换机可以在
该输出端口的上下文。出口处理是可选的，交换机可能不支持任何出口表
或可能未配置为使用它们。如果没有将有效的出口表配置为第一个出口表（请参阅
7.3.2），数据包必须由输出端口处理，并且在大多数情况下，数据包都被转发出去
开关。如果将有效的出口表配置为第一个出口表（请参见7.3。2），则数据包必须
与该流表的流条目匹配，并且可以使用其他出口流表，具体取决于
该流表中的比赛结果。

当由流表处理时，将数据包与流表的流条目进行匹配，以
选择一个流条目（请参阅5.3）。如果找到流条目，则该流条目中包含的指令集为
被执行。这些指令可以将数据包明确定向到另一个流表（使用Goto-
表指令，请参阅5。5），其中再次重复相同的过程。流条目只能引导
数据包到大于其自身流表编号的流表编号，换句话说，管道
处理只能前进，而不能后退。显然，a的最后一个表的流条目
流水线阶段不能包含Goto-Table指令。如果匹配的流条目不定向
数据包到另一个流表，流水线处理的当前阶段在此表停止，该数据包是
用其相关联的动作集进行处理并通常转发（请参阅5.6）。

如果数据包与流表中的流条目不匹配，则表示表未命中。桌子上的行为小姐
取决于表配置（请参阅5.4）。表格缺失流条目中包含的说明
流表可以灵活指定如何处理不匹配的数据包，有用的选项包括丢弃
它们，将它们传递到另一个表或通过控制通道将它们发送到控制器
打包消息（请参阅6.1.2）。

在少数情况下，流条目未完全处理数据包并且流水线处理停止
无需处理数据包的操作集或将其定向到另一个表。如果没有表缺失流条目
如果存在，则丢弃数据包（请参阅5.4）。如果发现无效的TTL，则可以将数据包发送到
控制器（请参阅5.8）。

OpenFlow管道和各种OpenFlow操作会以一种特定的方式处理特定类型的数据包。
符合为该数据包类型定义的规范，除非本规范或

OpenFlow配置另有指定，例如，Open-
流必须符合IEEE规范，并且OpenFlow使用的TCP / IP标头定义必须
符合RFC规范。此外，OpenFlow交换机中的数据包重新排序必须符合
符合IEEE规范的要求，前提是数据包由同一流处理
条目，组桶和仪表带。

5.1.1管道一致性

OpenFlow管道是一种抽象，它映射到交换机的实际硬件。在一些
在这种情况下，OpenFlow交换机是在硬件上虚拟化的，例如，以支持多个OpenFlow
开关实例或混合开关。即使未对OpenFlow交换机进行虚拟化，
硬件通常将不对应于OpenFlow管道，例如，如果仅流表
支持以太网数据包，非以太网数据包必须映射到以太网。在另一个
例如，某些交换机可能在一些内部元数据中携带VLAN信息，而对于OpenFlow
流水线在逻辑上是数据包的一部分。一些OpenFlow交换机可能会定义逻辑端口来实现
复杂的封装，可广泛修改数据包头。结果是一个数据包
在链路上或在硬件上的映射在OpenFlow管道中可能会有所不同。

但是，OpenFlow管道期望到硬件的映射是一致的，并且
OpenFlow管道的行为始终如一。特别是，这是预期的：

- **表一致性**：所有OpenFlow流表中的数据包都必须以相同的方式进行匹配，
  并且匹配的唯一区别必须归因于流表内容和显式OpenFlow
  这些流表完成的处理。特别是，标头不能透明地删除，
  在表之间添加或更改，除非由OpenFlow处理明确指定。
- **流条目一致性**：流条目应用于数据包的方式必须保持一致
  与流条目匹配。特别是，如果流条目中的匹配字段与特定数据包匹配
  标头字段，流条目中的相应set-field操作必须修改相同的标头字段，
  除非显式OpenFlow处理已修改数据包。
- **组一致性**：组的应用必须与流表保持一致。尤其是，
  组存储桶的操作必须以与流表中相同的方式应用于数据包，
  唯一的差异必须归因于显式的OpenFlow处理。
- **包入一致性**：包入消息中嵌入的包必须与
  OpenFlow流表。特别是，如果输入包是直接由流条目生成的，
  控制器接收到的数据包必须与将其发送到控制器的流条目匹配。
- **数据包输出一致性**：由于数据包输出请求而生成的数据包必须是
  与OpenFlow流表和入包过程一致。特别是如果一个数据包
  通过数据包输入接收的数据无需进行任何修改就直接通过数据包输出发送到端口，
  该端口上的数据包必须相同，就好像该数据包已发送到该端口而不是
  封装在一个包中。同样，如果将数据包输出定向到流表，则流条目
  必须按照OpenFlow匹配过程的要求匹配封装的数据包。
- **端口一致性**：OpenFlow端口的入口和出口处理必须一致
  彼此。特别是，如果OpenFlow数据包在端口上输出并生成物理
  交换机物理链路上的数据包，那么如果交换机接收到相同的物理数据包
  在同一链路上的同一端口上生成一个OpenFlow数据包，该OpenFlow数据包必须是
  相同。

5.2流表和流条目

流表由流条目组成。

匹配字段优先级计数器指令超时Cookie标志

表1：流表中流条目的主要组件。

每个流表条目（请参阅表1）包含：

- **match字段**：与数据包匹配。这些由入口端口和数据包标头组成，以及可选的其他管道字段，例如上一个表指定的元数据。
- **优先级**：流条目的匹配优先级。
- **计数器**：匹配数据包时更新。
- **说明**：修改操作集或管道处理。
- **超时**：交换机终止流量之前的最大时间或空闲时间。
- **cookie**：控制器选择的不透明数据值。可能被控制器用来过滤流量流统计，流修改和流删除请求影响的表项 何时不使用处理数据包。
- **标志**：标志更改流条目的管理方式，例如标志OFPFF_SEND_FLOW_REM触发该流条目的流删除消息。

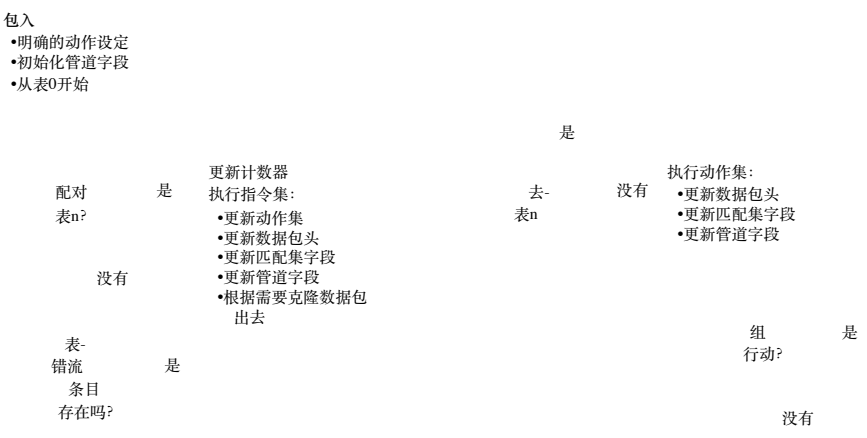
流表条目由其匹配字段和优先级标识：匹配字段和优先级一起标识特定流表中的唯一流条目。通配所有字段的流条目（省略所有字段）并且优先级等于0的情况称为表缺失流条目（请参见5.4）。

流进入指令可能包含要在流水线某个点对数据包执行的动作（请参阅5.8）。该*集合场*行动可以指定一些头字段重写。每个流表可能不支持本规范定义的每个匹配字段，每个指令，每个动作或每个设置字段，以及交换机的不同流表可能不支持相同的子集。表格功能请求启用控制器以发现每个表支持的内容（请参见7.3.5.18）。

5.3匹配

收到数据包后，OpenFlow交换机将执行图3所示的功能。交换机将启动通过在第一流表中执行表查找，并基于流水线处理，可以执行其他流表中的表查找（请参阅5.1）。

从数据包中提取数据包头字段，并检索数据包管道字段。包用于表查找的标头字段取决于数据包类型，通常包括各种协议标头字段，例如以太网源地址或IPv4目标地址（请参阅7.2.3）。此外数据包标头，也可以对入口端口，元数据字段和其他数据进行匹配管道字段。元数据可用于在交换机中的表之间传递信息。包报头字段和管道字段表示当前状态的数据包（如果在上一个字段中应用了操作）



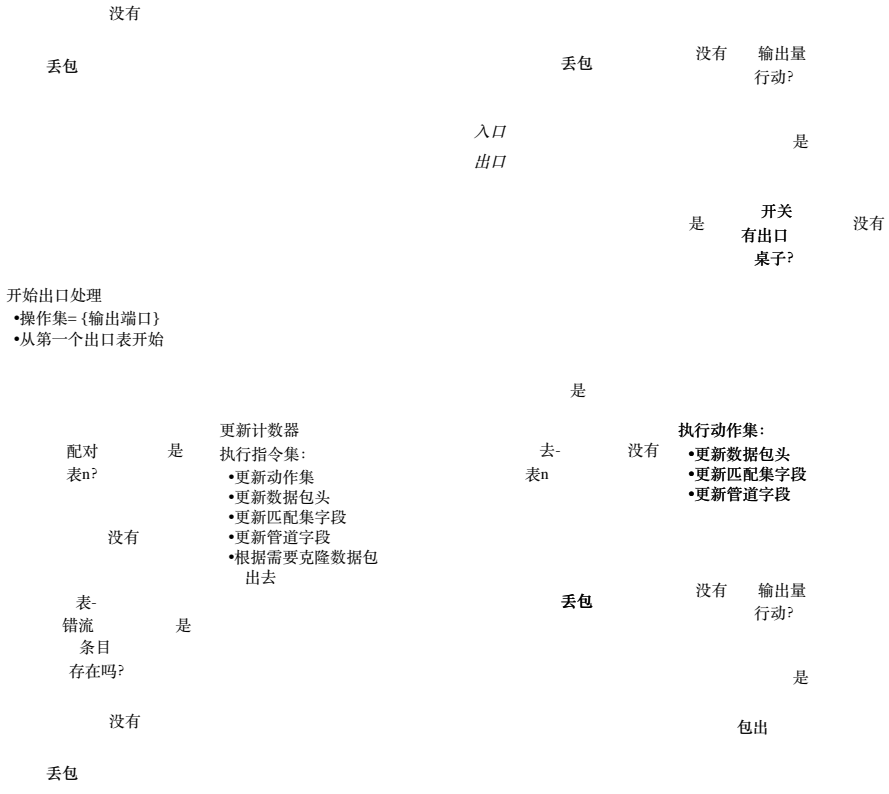


图3：简化了流程图，详细说明了通过OpenFlow交换机的数据包流。

使用*Apply-Actions*指令的表更改了数据包头或管道字段，这些更改
反映在数据包头字段和管道字段中。

如果流条目的所有匹配字段都与对应的流条目匹配，则数据包与流条目匹配
数据包的标头字段和管道字段。如果流程条目中省略了匹配字段（即值
ANY），它匹配数据包的标头字段或管道字段中的所有可能值（请参见7 .2.3.4）。如果
match字段存在且不包含掩码，match字段与相应字段匹配
包中的标头字段或管道字段（如果具有相同的值）。如果交换机支持任意
特定匹配字段上的位掩码，这些掩码可以更精确地指定匹配项，匹配字段为
如果它与掩码中设置的位具有相同的值，则匹配（参见7 .2.3.5）。

数据包与流表中的流条目进行匹配，并且*仅*匹配最高优先级的流条目
必须选择与数据包匹配的数据包。与所选流条目关联的计数器必须
进行更新（请参阅5 .9），并且必须执行所选流条目中包含的指令集
（请参见5.5）。如果存在多个具有相同最高优先级的匹配流条目，则选择
流条目是显式未定义的。仅当控制器编写器从不设置

流模块消息上的OFPFF_CHECK_OVERLAP位，并添加重叠条目。

如果交换机配置中包含IP片段，则必须在流水线处理之前重新组合IP片段。

OFPC_FRAG_REASM标志（请参阅7 .3.2 ）。

交换机收到恶意代码后，此规范版本*并未*定义预期的行为
在OpenFlow端口（请参阅4 .1）或在*Packet-Out*消息（请参阅4 ）中形成或损坏的数据包
6.1.1）。

5.4表格遗漏

每个流表都必须支持一个表缺失流条目，以处理未命中的表。表格缺失流条目
指定如何处理流表中其他流条目无法匹配的数据包（请参阅5.1），并且可以，
例如，将数据包发送到控制器，丢弃数据包或将数据包定向到后续表。

table-miss流条目由其匹配项和优先级标识（请参阅5.2），通配符全部匹配
字段（忽略所有字段），并具有最低优先级（0）。表格缺失流条目的匹配
可能不在流表支持的正常匹配范围内，例如完全匹配
表将不支持其他流条目的通配符，但必须支持table-miss流条目
通配所有字段。表缺失流条目可能不具有与常规流条目相同的功能
（请参见7.3.5.18）。table-miss流条目必须至少支持使用以下命令向控制器发送数据包

CONTROLLER保留端口（请参阅4.5），并使用Clear-Actions指令丢弃数据包
（请参见5.5）。鼓励实现支持在以下情况下将数据包定向到后续表：
与本规范的早期版本兼容。

table-miss流条目的行为在大多数方面都与其他任何流条目相同：默认情况下不存在
在流表中，控制器可以随时添加或删除它（请参阅6.4），它可以过期（请参阅
可以被逐出（见6.4），并且可以被控制器读取（见7.3.5.2）。表格缺失流条目
根据其匹配字段和优先级集（见5.3），按预期匹配表中的数据包：
流表中其他流条目无法匹配的数据包。表格未命中流程输入说明为
适用于与表未命中流条目匹配的数据包（请参阅5.5）。如果直接进入表缺失流
使用CONTROLLER保留端口（请参阅4.5）将数据包发送到控制器，原因是数据包进入
必须标识一个表格缺失（参见7.4.1）。

如果不存在表缺失流条目，则默认丢弃与流条目不匹配的数据包
（丢弃）。交换机配置（例如使用OpenFlow配置协议）可以
覆盖此默认值并指定其他行为。使用最低优先级（0）和
具有一个不通配符的匹配项（如果流表支持的话，则可以使用所有匹配项）
这不是表缺失流条目。仅当表缺失流时，使用此类流条目才有意义
该条目未使用，因为如果存在表缺失流条目，则它们将重叠，然后匹配
未定义。因此，建议控制器不要创建非表丢失流
使用最低优先级（0）的条目。

5.5说明

图4：流表中的匹配和指令执行。

每个流条目包含一组指令，当数据包与该条目匹配时将执行这些指令。这些
指令导致数据包，操作集和/或流水线处理的更改（请参见图4）。

不需要开关来支持所有指令类型，只需支持那些标记为“ *Required Instruction”的指令即可*
下面。控制器还可以查询交换机有关哪种“*可选指令”的*类型
支持（请参阅7.3.5.18）。

- *可选说明：* **Apply-Actions** *操作*：立即应用特定操作，
  无需对操作集进行任何更改。该指令可用于修改之间的数据包
  两个表或执行相同类型的多个动作。这些动作被指定为列表
  动作（请参阅5.7）。

- *必需的指令：* **清除动作**：立即清除动作集中的所有动作。支持
  仅对于*表缺失流条目*（请参阅5.4）才需要此指令的端口，而对于
  其他流条目。

- *必需的指令：* **Write-Actions** ***操作***：将指定的一组操作合并到
  当前操作集（请参阅5.6）。如果当前集中存在给定类型的操作，则将其覆盖，
  否则添加它。如果当前集中存在具有给定字段类型的*set-field*操作，则将其覆盖，
  否则添加它。所有流表都必须支持此指令。

- *可选指令：* **Write-Metadata** ***元数据/掩码***：写入掩码的元数据值
  进入元数据字段。掩码指定应修改元数据寄存器的哪些位
  （即，新的元数据=旧的元数据＆～掩码值＆掩码）。

- *可选说明：* **Stat-Trigger** ***stat thresholds***：如果有一些则向控制器生成事件
  流量统计信息的其中一个跨过统计阈值之一。

- *必需的指令：* **转到表*next-table-id***：指示处理中的下一个表
  管道。table-id必须大于当前的table-id。此说明必须是
  移植到除最后一个表之外的所有表中，只有一个表的OpenFlow交换机
  不需要执行此指令。管道最后一个表的流条目可以
  不包括此指令（请参阅5.1）。

与流条目关联的指令集最多包含每种类型的一条指令。
该*实验者*指令由他们的实验者-id和实验者型鉴定，因此
该指令集最多可包含一个*实验者*指令，用于以下各项的每种组合：
实验者编号和实验者类型。该集合的指令按以下指定的顺序执行
上面的清单。实际上，唯一的限制是执行“ *清除动作*”指令
在*Write-Actions*指令之前，*Apply-Actions*在*Write-Metadata*和
是*后藤表*最后执行。

如果交换机无法执行指令或指令的一部分，则必须拒绝该流程条目
与流条目关联。在这种情况下，交换机必须返回与以下内容相关的错误消息：
问题（请参见7.5.4.4）。

5.6动作集

动作集与每个分组相关联。该集合默认为空。流条目可以修改
使用与特定指令关联的*Write-Action*指令或*Clear-Action*指令设置*动作*
比赛。操作集在流表之间进行。当流条目的指令集
不包含*Goto-Table*指令，流水线处理将停止，并且该操作的操作集中的操作
包被执行（参见图3）。

一个动作集最多包含每种类型的一个动作。该*集合场*行动是由他们鉴定
目标字段类型，因此操作集每个字段最多包含一个*设置字段*操作
类型（即可以设置多个字段，但每个字段只能设置一次）。的效果*拷贝场*行动
由于竞争条件，未定义动作集中的，因此不鼓励其实施。的
*实验者的*动作是通过其实验者ID和实验者类型来识别的，因此该动作
对于实验者ID和的每种组合，集合最多可以包含一个*实验者*动作。
实验者类型。在操作集中添加特定类型的操作时，如果相同的操作

类型存在，将被以后的操作覆盖。如果需要多个相同类型的动作，例如
推送多个MPLS标签或弹出多个MPLS标签，*Apply-Actions*指令应
被使用（见5.7）。

用于出口处理的操作集具有一个限制，*输出*操作和*组*操作都可以
不添加到出口动作集中。在开始时初始化用于出口处理的操作集
当前输出端口具有*输出*操作的出口处理，而为入口设置的操作
处理开始为空。

动作集中的动作按照以下指定的顺序应用，无论它们的顺序如何
被添加到集合中。如果操作集包含组操作，则相应操作中的操作
该组的存储桶也按以下指定的顺序应用。该开关可支持任意
通过*Apply-Actions*指令的动作列表执行动作执行顺序。

1. **向内复制TTL**：将TTL向内复制动作应用于数据包

2. **pop**：将所有标签弹出操作应用于数据包
3. **push-MPLS**：将MPLS标签推送操作应用于数据包
4. **push-PBB**：对数据包应用PBB标签推送操作
5. **push-VLAN**：对数据包应用VLAN标记推送操作
6. **向外复制TTL**：对数据包应用向外复制TTL动作
7. **递减TTL**：对数据包应用递减TTL操作
8. **set**：将所有set-field操作应用于数据包
9. **qos**：将所有QoS操作（例如仪表和设置队列）应用于数据包
10. 组：如果指定了组操作，则在
    此列表指定的顺序
11. 输出：如果未指定组操作，则在输出指定的端口上转发数据包
    行动

操作集中的*输出*操作最后执行。如果*输出*动作和*组*动作均是
如果在操作集中指定了该操作，则将忽略输出操作，而组操作优先。如果不
如果在操作集中未指定输出操作和组操作，则丢弃数据包。如果没有团体
如果指定了action，并且输出action引用了一个不存在的端口，则该数据包将被丢弃。的
如果交换机支持，则组的执行是递归的；组存储桶可以指定另一个组
在这种情况下，动作的执行会遍历组配置指定的所有组。

的*输出*动作是在入口和出口不同的处理（参见5. 1）。设定进入动作时
包含将数据包转发到端口的*输出*操作或*组*操作，数据包必须开始
该端口上的出口处理（请参阅5 .1）。如果*输出*操作引用了*ALL*保留端口，则进行克隆
每个相关端口的数据包开始出口处理的副本（副本）（请参阅4. 5）。设置出口动作时
包含*输出*操作，数据包必须退出出口处理，并且必须由端口处理，
并在大多数情况下将其转发出交换机。

5.7行动清单

该*应用，操作*指令和*数据包出*消息，包括操作的列表。的语义
动作列表与OpenFlow 1.0规范相同。动作列表中的动作是
按列表指定的顺序执行，并立即应用于数据包。

动作列表的执行从列表中的第一个动作开始，并且每个动作都在
依次分组。如果动作列表包含两个Push，则这些动作的效果是累积的
VLAN操作，两个VLAN标头已添加到数据包。

如果操作列表包含*输出*操作，则将在其当前包中转发数据包的克隆（副本）
状态到开始进行出口处理的所需端口。如果*输出*动作引用了*ALL*
保留端口，是每个相关端口的数据包开始出口处理的克隆（请参见4.5）。如果输出
如果操作引用了不存在的端口，则将删除数据包的克隆。如果操作列表包含
一*组*动作，在其当前状态下的分组的克隆由相关组桶处理（见
5.10）。管道字段集随数据包一起克隆。对副本的任何修改
输出或组操作生成的数据包或其管道字段，例如更改标头
组存储区或出口表中的“字段”或元数据字段，仅适用于该克隆，不适用于
适用于原始数据包或其他克隆。

在执行*Apply-Actions*指令中的动作列表之后，管道执行继续
在修改后的数据包上（请参阅5.1）。数据包的动作集通过列表的执行保持不变
行动。

5.8动作

不需要开关来支持所有操作类型，只需支持下面标记为“*必需的操作*”*的*那些类型即可。回覆-
所有流表中都必须支持所需的操作。控制器还可以查询开关有关
它支持“*可选操作*”*中的*哪一个（请参见7.3.5.18）。

必需的操作： 输出*端口号*。Output操作将数据包转发到指定的OpenFlow端口
（请参阅4.1）在哪里开始出口处理。OpenFlow交换机必须支持转发到物理
端口，交换机定义的逻辑端口和所需的保留端口（请参阅4.5）。

必需的操作： 组*组**ID**。通过指定的组处理数据包（请参阅5.10）。的

确切的解释取决于组类型。

必需的操作： **Drop**。没有明确的动作可以表示掉落。相反，其作用的数据包
集合没有输出动作，也不能删除任何组动作。此结果可能来自空
指令集或处理流水线中的空操作桶（请参阅5 .6），或在执行*Clear-*
*动作*说明（请参阅5.5）。

可选操作： **Set-Queue *queue id***。set-queue操作设置数据包的队列ID。当。。。的时候
使用输出操作将数据包转发到端口，队列ID确定连接的队列
此端口用于调度和转发数据包。转发行为由
队列的配置，并用于提供基本的服务质量（QoS）支持（请参阅“
7.3.5.8）。

可选操作： **Meter *meter id***。将数据包定向到指定的仪表（请参阅5.11）。作为结果
在进行计量时，可能会丢弃数据包（取决于计量器配置和状态）。如果

开关支持仪表，它必须支持此操作作为操作*列表中*的第一个*操作*，以便向后
与规范的早期版本兼容。可选地，支持仪表的开关也可以
在动作*列表*中的其他位置支持电表动作，在动作*列表中支持*多个电表*动作*
以及*动作集中*的仪表动作（请参阅第7.3.5.14节）。

可选操作： **Push-Tag / Pop-Tag *ethertype***。开关可能支持推送/弹出标签的功能
如表2所示。为了帮助与现有网络集成，我们建议采用推送/弹出功能
支持VLAN标签。

新插入的标签应*始终*作为最外面的标签插入最有效的位置
该标签（请参见7.2.6。6）。将多个推送操作添加到数据包的操作集中后，它们将适用
按照操作集规则定义的顺序对数据包进行处理，首先是MPLS，然后是PBB，而不是VLAN（请参见5.6）。
当动作列表中包含多个推送动作时，它们将按列表顺序应用于数据包
（请参阅5.7）。

注意：有关默认字段值的信息，请参阅第5.8.1节。

行动	关联数据	描述
推送VLAN标头	以太网类型	将新的VLAN标头压入数据包。 以太网类型用作标签的以太网类型。只要 应使用以太类型0x8100和0x88a8。
弹出VLAN标头	--	从数据包中弹出最外面的VLAN标头。
推送MPLS标头	以太网类型	将新的MPLS填充标头推入数据包。 以太网类型用作标签的以太网类型。只要 应该使用以太类型0x8847和0x8848。
弹出MPLS标头	以太网类型	从数据包中弹出最外面的MPLS标签或填充头。 以太网类型用作结果数据包的以太网类型 （MPLS有效负载的以太网类型）。
推PBB接头	以太网类型	将新的PBB服务实例标头（I-TAG TCI）推送到 数据包（请参阅7.2.6.6）。 以太网类型用作标签的以太网类型。只要 应该使用以太类型0x88E7。
弹出式PBB标头	--	弹出最外面的PBB服务实例标头（I-TAG TCI） 从数据包（请参见7 .2.6.6）。

表2：推送/弹出标签操作。

可选操作： **Set-Field *字段类型value***。各种“设置字段”操作由它们的标识
字段类型，并修改数据包中各个标头字段的值（请参见7.2.3.7）。虽然不严格
需要使用Set-Field操作重写各种头字段的支持大大增加了
OpenFlow实现的有用性。为了帮助与现有网络集成，我们建议
支持VLAN修改操作。Set-Field操作应*始终*应用于最外层-
可能的标头（例如，“设置VLAN ID”操作始终设置最外面的VLAN标签的ID），除非
字段类型另行指定。

可选操作： ***复制字段src字段类型dst字段类型***。复制字段操作可能会复制数据
在任何标题或管道字段之间。它通常用于将数据从标头字段复制到数据包
寄存器流水线字段或从数据包寄存器流水线字段到标头字段，在某些情况下从
一个标头字段到另一个标头字段。交换机可能不支持以下之间的所有副本组合：
标头或管道字段。

可选操作： **Change-TTL *ttl***。各种Change-TTL操作会修改IPv4的值
数据包中的TTL，IPv6跳数限制或MPLS TTL。虽然不是严格要求，但显示的动作
表3中的内容大大提高了OpenFlow实现对实现路由的实用性
功能。Change-TTL操作应*始终*应用于最外面的报头。

行动	关联数据	描述
设置MPLS TTL	8位：新的MPLS TTL	更换现有的MPLS TTL。仅适用于数据包 与现有的MPLS填充标头。
递减MPLS TTL	--	减少MPLS TTL。仅适用于具有 现有的MPLS填充标头。
设定IP TTL	8位：新IP TTL	替换现有的IPv4 TTL或IPv6跃点限制，并 更新IP校验和。仅适用于IPv4和IPv6 包。
减少IP TTL	--	减少IPv4 TTL或IPv6跳数限制字段，然后 更新IP校验和。仅适用于IPv4和IPv6 包。
向外复制TTL	--	将TTL从最下到最外复制到最外 带TTL的标头。 复制可以是IP到IP，MPLS到MPLS或IP到MPLS。
向内复制TTL	--	将TTL从最外层复制到最下层 带TTL的标头。 复制可以是IP到IP，MPLS到MPLS或MPLS到IP。

表3：Change-TTL操作。

OpenFlow交换机检查IP TTL或MPLS TTL无效的数据包并拒绝它们。检查
不需要为每个数据包都执行无效的TTL，但必须至少每个
时间*递减TTL*动作被应用到的分组。交换机的异步配置可能
进行更改（请参阅6.1.1），以通过控制通道通过以下方式将具有无效TTL的数据包发送到控制器
打包消息（请参阅6.1.2）。

5.8.1推送字段的默认值

通过推送操作添加新标签时（请参阅5.8），推送操作本身并未指定大多数
标签的标头字段部分的值。其中大多数是通过“设置字段”操作单独设置的。的
在push操作期间初始化这些标头字段的方式取决于这些字段是否定义为
OpenFlow匹配字段（请参见7.2.3.7）和数据包的状态。标头推送操作的标头字段部分
表4列出了定义为OpenFlow匹配字段的字段。

执行推送操作时，对于添加的标头的表4部分中指定的所有字段，该值
数据包的现有外部标头中对应字段中的字段应复制到新的
领域。如果数据包中不存在相应的外部报头字段，则应将新字段设置为
零。

未定义为OpenFlow匹配字段的标头字段应初始化为适当的
协议值：这些标头字段的设置应遵守规范
字段，并考虑包头和交换机配置。例如，DEI位在
大多数情况下，VLAN标头应设置为零。

新领域	现有领域
VLAN编号	←VLAN ID
VLAN优先级	←VLAN优先级
MPLS标签	←MPLS标签
MPLS流量类别	←MPLS流量类别
MPLS TTL	← { MPLS TTL IP TTL
PBB I-SID	←PBB I-SID
多溴联苯I-PCP	←VLAN PCP
多溴联苯C-DA	←ETH DST
PBB C-SA	←ETH SRC

表4：可以通过推送操作复制到新字段中的现有字段。

通过为适当的字段指定“设置字段”操作，可以覆盖新标题中的字段
推送操作后。操作集中操作的执行顺序旨在简化操作
处理（请参阅5.6）。

5.9柜台

为每个流表，流条目，端口，队列，组，组存储桶，仪表和计数器维护计数器
米带。兼容OpenFlow的计数器可以在软件中实现并通过轮询来维护
范围更有限的硬件计数器。表5 包含由计数器定义的一组计数器
OpenFlow规范。不需要开关来支持所有计数器，只需标记为“ *Required”的*计数器即可。
在表5中。

持续时间是指流条目，端口，组，队列或仪表已经过的时间
安装在交换机中，并且必须以第二精度进行跟踪。接收错误字段是总计
表5中定义的所有接收和冲突错误，以及表中未提及的任何其他错误。

与OpenFlow对象有关的数据包相关计数器必须对使用该对象的每个数据包进行计数，即使
对象对数据包没有影响，或者最终还是丢弃了数据包或将其发送给控制器。
例如，交换机应维护以下与分组相关的计数器：

- •仅具有goto-table指令且没有操作的流条目
- •组输出到不存在的端口
- •触发TTL异常的流条目
- •发生故障的端口

计数器是无符号的，并且没有溢出指示器。计数器必须使用完整位
翻转前为计数器定义的范围，例如，如果将计数器定义为64位，则可以
不要只使用低32位。如果交换机中没有特定的数字计数器，则其值必须
设置为最大字段值（无符号等效值-1）。

计数器	位	
每流表		
参考计数（活动条目）	32	*需要*
数据包查找	64	*可选的*
数据包匹配	64	*可选的*
每个流条目		
收到报文	64	*可选的*
接收字节	64	*可选的*
持续时间（秒）	32	*需要*
持续时间（纳秒）	32	*可选的*
每端口		
收到报文	64	*需要*
传送的封包	64	*需要*
接收字节	64	*可选的*
传输字节	64	*可选的*
接收滴	64	*可选的*
传输丢包	64	*可选的*
接收错误	64	*可选的*
传输错误	64	*可选的*
接收帧对齐错误	64	*可选的*
接收超限错误	64	*可选的*
接收CRC错误	64	*可选的*
碰撞	64	*可选的*
持续时间（秒）	32	*需要*
持续时间（纳秒）	32	*可选的*
每个队列		
传送封包	64	*需要*
传输字节	64	*可选的*
传输溢出错误	64	*可选的*
持续时间（秒）	32	*需要*
持续时间（纳秒）	32	*可选的*
每组		
参考计数（流条目）	32	*可选的*
包数	64	*可选的*
	64	

字节数		
持续时间（秒）	32	*可选的*
持续时间（纳秒）	32	*需要*
		可选的
每组桶		
包数	64	*可选的*
字节数	64	*可选的*
每米		
流量计数	32	*可选的*
输入数据包计数	64	*可选的*
输入字节数	64	*可选的*
持续时间（秒）	32	*需要*
持续时间（纳秒）	32	*可选的*
每米带		
带内数据包计数	64	*可选的*
带内字节数	64	*可选的*

表5：计数器列表。

5.10组表

组表由组条目组成。流条目指向*组的能力*启用了OpenFlow
代表其他转发方法（例如全选和全选）。

组标识符组类型计数器操作桶

表6：组表中组条目的主要组成部分。

每个组条目（请参阅表6）由其组标识符标识，并且包含：

- **组标识符**：一个32位无符号整数，用于唯一标识OpenFlow上的组
  开关。
- **组类型**：确定组语义（请参阅第5.10.1节）。
- **计数器**：组处理数据包时更新。
- **操作桶**：**操作桶**的有序列表，其中每个操作桶包含一组
  要执行的动作和相关参数。值区中的动作始终作为
  动作集（请参阅5.6）。

组条目可能包含零个或多个存储桶，但*间接*类型始终会
有一个水桶。没有存储桶的组可以有效地丢弃数据包（请参阅5.6）。

存储桶通常包含修改数据包的操作和将其转发到的输出操作
一个端口。存储桶还可以包括一个组操作，如果交换机支持，该组操作将调用另一个组
组链接（请参阅6.7），在这种情况下，数据包处理将在调用的组中继续（请参见5.6）。一种
没有动作的存储桶有效，没有输出或组动作的存储桶有效地删除了
与该存储桶关联的数据包。

5.10.1组类型

不需要交换机来支持所有组类型，只需支持下面标记为“*Required*”*的组即可*。控制器
也可以查询交换机支持哪种“*可选*”组类型。

- *必需：* **间接**：执行该组中定义的一个存储桶。该组仅支持
  一个水桶。允许多个流条目或组指向一个公共组标识符，
  支持更快，更有效的融合（例如IP转发的下一跳）。这个团体类型
  实际上与一个水桶的所有群组相同。该组是最简单的组，
  因此，交换机通常会比其他组类型支持更多数量的交换机。
- *必需：* **全部**：执行组中的所有存储桶。该组用于多播或广播
  转发。有效地为每个存储桶克隆该数据包；每个处理一个数据包
  该组的存储桶。如果存储桶将数据包明确定向到入口端口，则此数据包克隆
  被丢弃。如果控制器编写者想转发入口端口，则该组必须包括
  一个额外的存储桶，其中包括对OFPP_IN_PORT保留端口的输出操作。

- *可选：* **选择**：在组中执行一个存储桶。数据包由单个存储桶处理
  该组，基于开关计算的选择算法（例如，一些用户配置的哈希
  元组或简单循环）。选择算法的所有配置和状态都是外部的
  到OpenFlow。选择算法应实现均等的负载分担，并且可以选择
  基于铲斗重量。当选择组的存储桶中指定的端口出现故障时，
  开关可能会将存储桶选择限制为其余的设置（那些具有转发操作的选项会生效）
  端口），而不是丢弃发往该端口的数据包。这种行为可以减少干扰
  断开的链路或交换机。
- *可选：* **快速故障转移**：执行第一个活动存储桶。每个动作存储区都与
  控制其活动性的特定端口和/或组。铲斗按顺序评估
  由组定义，并选择与活动端口/组关联的第一个存储桶。
  此组类型使交换机可以更改转发，而无需往返于
  控制器。如果没有存储桶，则丢弃数据包。该组类型必须实现*生动性*
  机制（请参见6.7）。

5.10.2团体活动监控

*快速故障转移*组支持需要活动性监视，以确定要执行的特定存储桶。
不需要其他组类型来实现活动监控，但是可以选择实现
它。如果交换机无法对组中的任何存储桶执行活动性检查，则必须拒绝该组
mod并返回错误。

组存储桶可以显式监视端口或另一个组的活动。确定规则
活动包括：

- 如果端口存在于数据路径中并且在其端口中设置了OFPPS_LIVE标志，则该端口被认为是活动的
  州。可以通过交换机的OpenFlow部分之外的代码来管理端口活动性
  OpenFlow规范之外的其他内容，例如生成树或KeepAlive机制。的
  如果端口之一处于活动状态，则该端口不得视为活动端口（并且必须未设置OFPPS_LIVE标志）
  在该OpenFlow端口上启用的交换机的活动性机制认为该端口不活动，
  或者端口配置位OFPPC_PORT_DOWN指示端口已关闭，或者端口状态位
  OFPPS_LINK_DOWN表示链接已断开。
- 如果watch_port不是OFPP_ANY并且监视的端口是活动的，则将桶视为活动的，
  或者，如果watch_group不是OFPG_ANY，并且观看的组是直播的。换句话说，桶是
  如果watch_port为OFPP_ANY或观看的端口未启用，并且watch_group被认为未启用
  是OFPG_ANY或观看的群组未直播。
- 如果某个组中至少有一个桶是活动的，则该组被视为活动的。

控制器可以通过监视各个端口的状态来推断组的活动状态。

5.11表

仪表表由仪表条目组成，用于定义每个流量计。每流量计启用OpenFlow
为了实现速率限制，一个简单的QoS操作将一组流约束到选定的带宽。
每流量计量还可以使OpenFlow实施更复杂的QoS策略操作，例如

基于DSCP的计量，可以根据其速率将一组数据包分类为多个类别。仪表
完全独立于每个端口队列（请参阅5. 8），但是在很多情况下，这两个功能可以
结合以实现复杂的工作节省QoS框架，例如DiffServ。

仪表可以测量分配给它的数据包的速率，并可以控制这些数据包的速率。
仪表直接连接到流条目（与连接到端口的队列相反）。如果
开关支持它，任何流量条目都可以在动作列表中指定仪表动作（请参阅5. 8）：仪表
测量并控制其所连接的所有流条目的聚合速率。

仪表标识符仪表带计数器

表7：电表中电表条目的主要组成部分。

每个仪表条目（请参阅表7）均由其仪表标识符标识，并且包含：

- **仪表标识符**：一个唯一标识仪表的32位无符号整数
- **米带**：米带的无序列表，其中每个米带指定
  频段和处理数据包的方式
- **计数器**：由仪表处理数据包时更新

同一流表中的不同流条目可以使用相同的仪表，不同的仪表或不使用仪表
所有。通过在流表中使用不同的计量表，可以独立地计量不相交的一组流量条目。
在每个流表的连续流表中使用仪表时，数据包可能会经过多个仪表
匹配的流量条目可能会将其引导至一米。使用多米的另一种方式，如果切换
支持它，就是每个流条目使用多个仪表操作。这可以用来执行分层
计量，其中首先独立计量各种流量，然后再对其进行计量（请参见图1）。
5）。

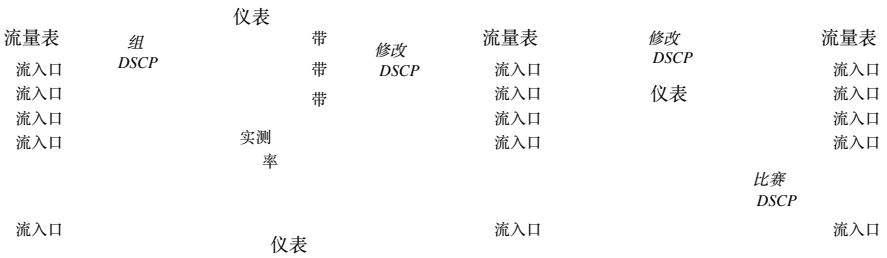


图5：仪表和分层DSCP仪表。

5.11.1米带

每个仪表可以具有一个或多个仪表带。这些带用于定义传感器的行为。
仪表包上的仪表适用于仪表测量速率的各种范围。电表*实测速率*计算

使用所有流条目中的所有数据包将数据包定向到该仪表。仪表为每个包选择
根据测量的速率，带速率值和电表配置，选择一个电表频段。
一旦选择了频段，仪表将对该仪表频段指定的处理应用于数据包。
数据包仅由仪表中的单个仪表带处理。

每个仪表频带指定该频带的目标速率，以及如果该速率应处理数据包的方式
被超过。默认的仪表频段始终包含在仪表中，并且无法设置，等效
到目标速率为0且不执行任何操作的频段，它只是让数据包通过而无所事事。

频段类型速率突发计数器类型特定参数

表8：表条目中表带的主要组成部分。

每米条带（见表8）是由它的目标速率标识，并包含：

- **频段类型**：定义如何处理数据包
- **rate**：该频段的目标速率-仪表用来选择仪表频段，通常是最低的
  频段适用的速率
- **突发**：定义仪表频带的粒度
- **计数器**：在仪表频带处理数据包时更新
- **类型特定的参数**：某些波段类型具有可选参数

本规范中没有“*必需*”的频段类型。控制器可以查询开关有关哪个
支持的“*可选*”电表频段类型。

- *可选*：**drop**：丢弃（丢弃）数据包。可用于定义速率限制器频段。
- *可选*：**dscp备注**：增加DSCP字段在IP头中的丢弃优先级。
  包。可用于定义一个简单的DiffServ策略程序。

频段选择指南每包单测量速率的计算映射到每个频段的米并且
通过增加目标速率对等级进行排序，默认频段的等级为0，而第一个可配置的频段
具有等级1。频段选择过程的结果必须遵守以下所有限制条件：

- 数据包仅由单个仪表频带处理。
- 仅当仪表的*测量速率*超过频段时，才必须按仪表频段处理数据包
  目标利率。
- 对于正在处理数据包的任何仪表带，所有仪表处理的流量
  等级较低的频段必须等于频段目标速率。例如，如果处理了数据包
  根据第一个可配置的仪表频段，默认频段处理的流量必须
  等于其目标利率。

一个简化的模型是，对于每个数据包，电表都会应用具有最高目标的电表带
低于当前测量速率的速率。

实际上，对于短时间窗口，由于数据包的原因，电表可能无法完全遵守这些约束
粒度，速率测量近似值和各种实施约束，因此，
仅描述长期行为。所述*测得的速率*的计算通常与令牌桶完成
或滑动窗口。特别是使用令牌桶时，短暂的数据包突发可能会暂时超过
频段目标速率，而不会触发该频段。所述*突发*参数定义的粒度
频段：对于超过该值的数据包突发，将实际流量和频段进行比较
目标速率必须准确，短脉冲可能不准确。

5.12进出处理差异

流表可用于入口或出口处理（请参阅5.1）。入口处理是主要的
数据包进入OpenFlow交换机时发生的处理，可能涉及一个或多个
流量表。出口处理是确定输出端口后发生的处理，
它发生在输出端口的上下文中，并且可能涉及零个或多个流表。

用于入口和出口处理的流表之间几乎没有区别，流条目具有
相同的组件（请参阅5.2），流表匹配相同（请参见5.3）和执行指令
是相同的（请参见5.5）。表未命中的处理相同（请参见5.4），因此强烈
建议在每个出口表中设置一个表缺失流条目，以避免丢失数据包。在下，
可以通过更改第一个出口表将流表重新配置为进行入口或出口处理（请参阅
7.3.2）。

在入口处理开始时，操作集为空。用于入口处理的流表
只能通过*Goto-Table*指令将数据包定向到入口流表，而不能将数据包定向
使用*Goto-Table*指令导出流表。入口流表通常不支持
*动作集输出*匹配字段，但在某些情况下可能会支持它。

在出口处理开始时，操作集仅包含当前操作的*输出*操作
输出端口。该输出操作的有效值为任何物理或逻辑端口，或者为*CONTROLLER*
或*LOCAL*保留端口（请参见4.5），其他保留端口必须替换为其实际值。
除了“ *动作集输出*”匹配字段之外，将保留管道字段的值；例如
元数据匹配字段的初始值是将数据包发送到出口时的值
处理。用于出口处理的流表只能通过*Goto-Table*指令定向数据包
出口流表。它们必须支持“ *动作集输出*”匹配字段，以允许基于流
在输出端口上下文中。他们必须禁止在*写*操作中使用*输出*操作或*组操作*
指令，以便不能更改数据包的输出端口。这些限制必须公告
在流表功能中（请参见7.3.5.18）。

出口表可以可选地支持*apply-action*指令中的*输出*动作或*组*动作。
这些操作的行为就像在入口表中一样，将数据包的克隆转发到指定的
端口，这些克隆必须从第一个出口表开始进行出口处理（请参阅5.7）。如果支持，
例如，这可以用于选择性出口镜像。交换机应该保护自己免受数据包的攻击
在出口表中使用*输出*操作或*组*操作时发生无限循环，此机制是实现
特定的，不在本规范范围之内。

6 OpenFlow通道和控制通道

OpenFlow通道是将每个OpenFlow逻辑交换机连接到OpenFlow的接口
控制器。通过此接口，控制器可配置和管理交换机，接收事件
从交换机，并将数据包发送出交换机。交换机的控制通道可能支持
具有单个控制器的单个OpenFlow通道，或支持多个的多个OpenFlow通道
控制器共享交换机的管理。

在数据路径和OpenFlow通道之间，该接口是特定于实现的，但是
所有OpenFlow通道消息都必须根据OpenFlow交换协议进行格式化。的
OpenFlow通道通常使用TLS加密，但可以直接在TCP上运行。

6.1 OpenFlow Switch协议简介

OpenFlow交换协议支持三种消息类型，即*控制器到交换机*，*异步*和
对称，每个都有多个子类型。控制器到交换机的消息由控制器启动
用于直接管理或检查交换机的状态。异步消息已启动
由交换机用于更新控制器有关网络事件和交换机状态的更改。
对称消息由交换机或控制器启动，并且无需征求即可发送。
OpenFlow使用的消息类型如下所述。

6.1.1控制器到开关

控制器/开关消息由控制器启动，可能需要也可能不需要以下响应：
开关。

特点：控制器可以通过发送请求交换机的身份和基本功能
功能请求；交换机必须以功能回复作为响应，该回复指定身份和基本
交换机的功能。通常在建立OpenFlow通道时执行此操作。

配置：控制器能够在交换机中设置和查询配置参数。的
开关仅响应来自控制器的查询。

Modify-State：Modify-State消息由控制器发送，以管理交换机上的状态。其
主要目的是添加，删除和修改流/组条目以及插入/删除操作的桶
组在OpenFlow表中并设置交换机端口属性。

读取状态：控制器使用读取状态消息来收集来自读取状态消息的各种信息。
开关，例如当前配置，统计信息和功能。多数读取状态的请求和答复
使用多部分消息序列实现（请参见7 .3.5）。

数据包输出：控制器使用它们将数据包从交换机上的指定端口发送出去，并且
转发通过Packet-in消息接收的数据包。封包输出消息必须包含完整的封包或
引用存储在交换机中的数据包的缓冲区ID。该消息还必须包含操作列表
按照指定的顺序应用；空的操作列表将丢弃数据包。

屏障：控制器使用屏障请求/答复消息来确保消息依赖性
已达到要求或已收到有关完成操作的通知。

Role-Request：角色请求消息由控制器用来设置其OpenFlow的角色
频道，设置其控制器ID或进行查询。当交换机连接到
多个控制器（请参阅6.3.7）。

异步配置：控制器使用异步配置消息
在它想在其OpenFlow通道上接收的异步消息上设置一个附加过滤器，
或查询该过滤器。当交换机连接到多个控制器时，这尤其有用（请参见6.3.7）
通常在建立OpenFlow频道时执行。

6.1.2异步

发送异步消息时，无需控制器从交换机请求它们。开关发送
发送给控制器的异步消息，以表示数据包到达或交换机状态更改。主要的
异步消息类型如下所述。

数据包输入：将**数据包**的控制权转移到控制器。对于所有转发到**CON-的**数据包
使用流条目或表未命中流条目的**TROLLER**保留端口，总是发生入包事件
发送给控制器（请参阅5 .8）。其他处理（例如TTL检查）也可能生成入包事件
发送数据包到控制器。

可以将Packet-in事件配置为缓冲数据包。对于由输出动作生成的数据包
流条目或组存储桶，可以在输出操作本身中单独指定它（请参见7.2.6.1）、
对于其他输入包，可以在交换机配置中配置（请参见7 .3.2）。如果packet-in事件是
配置为缓冲数据包，并且交换机具有足够的内存来缓冲它们，数据包进入事件
仅包含一部分数据包头和一个缓冲区ID，控制器在使用该缓冲区ID时
准备好交换机转发数据包。不支持内部缓冲的交换机是
配置为不为入包事件缓冲数据包，或者内部缓冲已用尽，必须
将完整的数据包作为事件的一部分发送给控制器。缓冲的数据包通常将通过
来自控制器的**Packet-out**或**Flow-mod**消息，或在一段时间后自动过期。

如果数据包被缓冲，则原始数据包要包含在数据包中的字节数可以是
配置。默认情况下，它是128个字节。对于流中的输出操作生成的入包事件
条目或组存储桶，可以在输出操作本身中单独指定它（请参见7 .2.6.1），
其他数据包进入事件可以在交换机配置中配置（请参见7.3.2）。

流量已删除：通知控制器流量表中流量条目的删除。流-
删除的消息仅发送给设置了OFPFF_SEND_FLOW_REM标志的流条目。他们是
由于控制器流删除请求而产生的结果，当以下情况之一时，开关流到期过程
超过了流超时或其他原因（请参见6.5）。

端口状态：通知控制器端口更改。交换机将发送端口状态
端口配置或端口状态更改时，向控制器发送消息。这些事件包括
端口配置事件，例如是否由用户直接将其关闭，以及端口状态更改
事件，例如，链接断开。

角色状态：通知控制器其角色的更改。当新的控制器当选自己的主人时，
预计交换机将向先前的主控制器发送角色状态消息（请参见6.3.7）。

Controller-Status：当OpenFlow通道的状态更改时通知控制器。开关
当任何交换机的OpenFlow通道的状态更改时，将这些消息发送给所有控制器。
如果控制器失去相互之间通信的能力，则可以协助故障转移处理。

流量监控器：将流量表中的更改通知控制器。控制器可以定义一组
监视以跟踪流表中的更改（请参见7.3.5.19）。

6.1.3对称

对称消息不经请求即向任一方向发送。

Hello：连接启动后，交换机和控制器之间将交换Hello消息。

回显：回显请求/回复消息可以从交换机或控制器发送，并且必须
返回回显答复。它们主要用于验证控制器-交换机连接的活动性，以及
也可以用来衡量其延迟或带宽。

错误：交换机或控制器使用错误消息将问题通知到另一侧
连接。交换机通常使用它们来指示由服务器发起的请求失败。
控制器。

实验者：实验者消息为OpenFlow交换机提供了额外的标准方法
OpenFlow消息类型空间中的常规功能。这是功能的暂存区
用于将来的OpenFlow版本。

6.2消息处理

开放流交换机协议提供可靠的消息传递和处理，但并*不能*自动-
主动提供确认或确保有序的消息处理。OpenFlow消息

本节中描述的处理行为在主连接和辅助连接上提供。
使用可靠的传输，但是使用不可靠的辅助连接不支持
运输（见6.3.8）。

消息传递：除非OpenFlow通道完全失败，否则保证**消息传递**。
在这种情况下，控制器不应假定任何有关开关状态的信息（例如，开关可能具有
进入“失败独立模式”）。

消息处理：交换机必须完整处理从控制器收到的每条消息，可能
产生回复。如果交换机无法完全处理从控制器收到的消息，则必须
发回错误消息。对于包出消息，无法完全处理消息
包含的数据包实际上退出了交换机。包含的数据包可能在以下时间被静默丢弃
由于交换机拥塞，QoS策略或发送给受阻或无效而导致的OpenFlow处理
港口。

此外，交换机必须将OpenFlow生成的所有异步消息发送到控制器
状态更改，例如流量删除，端口状态或入站消息，以便控制器视图
开关的状态与其实际状态一致。这些邮件可能会根据
异步配置（请参阅6.1。1）。而且，会触发OpenFlow状态的条件
更改可能会在导致更改之前被过滤掉。例如，在数据端口上收到的数据包

应该转发给控制器，可能会由于内部的拥塞或QoS策略而掉线
切换并且不生成任何打包消息。对于具有明确输出的数据包，可能会发生这些丢弃
控制器的动作。当数据包无法匹配网络中的任何条目时，也会发生这些丢弃
表，该表的默认操作是发送到控制器。数据包的监管到
建议控制器，以防止拒绝控制器连接的服务，这可以通过以下方式完成
控制器端口上的队列（请参阅7.3.5.8）或使用流量计（请参阅5.11）。

控制器可以随意忽略收到的消息，但是必须响应回显消息以防止
从终止连接切换。

消息分组：控制器可以使用可选的*捆绑软件*将相关消息分组在一起
消息（请参阅6.9）。捆绑包中的消息集将作为一个单元应用，它们将被一起处理，
如果出现任何错误，则不会应用任何消息。如果捆绑软件按顺序配置，则设置
捆绑中的邮件数量也按顺序应用。捆绑包的处理与
处理其他消息和其他包。

消息排序：可以通过使用*障碍*消息来确保**消息的排序**（请参阅
7.3.7）。在没有障碍消息的情况下，交换机可以随意重新排序消息以最大程度地
性能: 因此，控制器不应依赖特定的处理顺序。特别是流量
条目可以以与接收到的流mod消息不同的顺序插入表中。
开关。不得跨屏障消息对消息进行重新排序，并且屏障消息必须是
仅当所有先前的消息都已处理后才进行处理。更确切地说：

1.屏障之前的消息必须在屏障之前得到充分处理，包括发送任何结果
答复或错误

2.然后必须处理屏障，将状态提交到数据路径并发送屏障回复

3.障碍之后的消息然后可以开始处理

如果来自控制器的两条消息相互依赖，则必须用一个屏障将它们分开
消息或放入相同的有序捆绑销售商品中。此类消息依赖项的示例包括一个组
使用流mod add引用该组的mod add，使用将包输出转发到该端口的port mod
端口或带有以下数据包输出的流量模块添加到OFPP_TABLE。

6.3 OpenFlow通道连接

OpenFlow通道用于在OpenFlow交换机和交换机之间交换OpenFlow消息。
OpenFlow控制器。典型的OpenFlow控制器管理多个OpenFlow通道，每个通道
到另一个OpenFlow交换机。一个OpenFlow交换机可能有一个OpenFlow通道到一个
控制器，或多个可靠性通道，每个通道到一个不同的控制器（请参见6.3.7）。

OpenFlow控制器通常通过一个或多个网络远程管理OpenFlow交换机。
用于OpenFlow通道的网络规范不在本协议范围内
规格。它可以是单独的专用网络（带外控制器连接），也可以是开放式
流通道可以使用由OpenFlow交换机管理的网络（带内控制器连接）。
唯一的要求是它应该提供TCP / IP连接。

OpenFlow通道通常被实例化为交换机和交换机之间的单个网络连接。
控制器，使用TLS或纯TCP（请参见6.3。6）。可替代地，可以组成OpenFlow信道

多个网络连接以利用并行性（请参阅6.3 .8）。OpenFlow开关必须能够
通过启动与OpenFlow控制器的连接来创建OpenFlow通道（请参阅6.3。3）。一些
开关实现可以选择允许OpenFlow控制器连接到OpenFlow
交换机，在这种情况下，交换机通常应将自己限制在安全连接上（请参见6.3.6），以防止
未经授权的连接。

6.3.1连接URI

开关通过唯一的**Connection URI**标识控制器连接。此URI必须符合
RFC 3986中定义的URI的语法，尤其是在字符编码方面。的
连接URI可能具有以下格式之一：

- *协议*：*名称或地址*：*端口*
- *协议*：*名称或地址*

该*协议*字段定义了用于开放流消息的传输。为可接受的值*协议*是

tls或tcp用于主要连接（请参阅6 .3.6）。辅助连接的可接受值（参见6.3.8）为
tls，dtls，tcp或udp。

该*名称或地址*字段是控制器的主机名或IP地址。如果主机名是本地的
配置后，建议URI使用IP地址。如果主机名可通过DNS使用
URI可以使用任何一个，只要它是一致的即可。当地址为IPv6时，该地址应为
根据RFC 2732的建议，用方括号括起来。

所述*端口*字段是在控制器上使用的传输端口。如果未指定*端口*字段，则必须为
等效于将此字段设置为默认的OpenFlow传输端口6653。

如果交换机实现支持未定义的OpenFlow连接的传输，
在此规范中，它必须使用正确标识该传输类型的*协议*。在这种情况下，
连接URI的其余部分是协议定义的。

下面给出了一些连接URI的示例。

- tcp：198.168.10.98：6653
- tcp：198.168.10.98
- tls：test.opennetworking.org：6653
- tls：[3ffe：2a00：100：7031 :: 1]：6653

连接URI始终代表交换机的连接信息视图。如果地址
或端口由网络转换，这可能与控制器的地址或端口视图不匹配。
控制器可以请求不经过网络转换的唯一ID（请参阅6 .3.7）。

6.3.2备用连接传输

当前规范仅在tls和tcp上定义OpenFlow主连接（请参见6.3.1）。辅助
iliary连接是可选的，当前规范仅在tls上定义了辅助连接，

tcp，dtls和udp（请参阅6 .3.8）。可靠传输上的辅助连接（tls或tcp）可以使用完整的
OpenFlow协议，但是在不可靠传输（dtls或udp）上的辅助连接只能使用
OpenFlow协议的一小部分。

完整的OpenFlow协议可用于替代传输协议，而不是tls或tcp：
可以在备用传输协议上定义主连接或可靠的辅助连接
仅当此传输协议提供以下功能时：

- **多路复用**：OpenFlow交换机必须能够连接到多个控制器或打开
  到同一控制器的多个连接，因此传输协议必须能够
  识别和解复用各种OpenFlow连接。
- **可靠性**：OpenFlow协议没有任何错误检测和恢复机制，
  并假设消息永不丢失，因此传输协议必须可靠。
- **有序交付**：OpenFlow屏障消息期望OpenFlow消息在以下位置的顺序
  要保留的连接，因此传输协议必须按顺序传递消息。
- **分段和重组**：OpenFlow消息的大小最大为64kB，
  超过了大多数链接MTU的大小，并且OpenFlow不提供将大多数邮件分散的机制。
  智者，因此传输协议必须能够执行碎片和重组
  所需的OpenFlow消息。
- **流量控制**：OpenFlow协议假定交换机可以减慢或停止流量控制。
  控制器通过停止从底层连接读取消息来发送消息，因此
  传输协议必须实现一些流控制。
- **安全性**：OpenFlow协议本身并不实现安全性，因此传输
  协议或其下的协议必须提供安全性（如果需要）。

如果传输协议不能满足这些要求，则必须使用适配层，例如
在该传输上分层TCP / IP。除了这些要求之外，OpenFlow消息的映射
替代传输协议的定义是由适配层或传输协议定义的，并且
超出了本规范的范围。

只有在以下情况下，才能在备用传输协议上定义不可靠的辅助连接：
端口协议提供多路复用（请参见上文），并且仅限于为以下内容定义的OpenFlow子集：
不可靠的运输（见6.3.8）。

6.3.3连接设置

交换机必须能够以用户可配置的方式与控制器建立通信（但
（否则已修复）使用用户指定的传输端口或默认OpenFlow的连接URI
传输端口6653。如果交换机配置了控制器的连接URI以进行连接
到，交换机将启动到控制器的标准TLS或TCP连接，具体由的字段指定
连接URI。交换机必须允许建立和维护此类连接。对于

带外连接，交换机必须确保往返于OpenFlow通道的流量为
无法通过OpenFlow管道运行。对于带内连接，交换机必须设置适当的
OpenFlow管道中的连接流条目集。

可选地，开关可以允许控制器启动连接。在这种情况下，开关应
使用用户指定的方式从控制器接受传入的标准TLS或TCP连接
传输端口或默认的OpenFlow传输端口6653。连接由交换机和
建立传输连接后，控制器的行为相同。

首次建立OpenFlow连接时，连接的每一端必须立即发送
OFPT_HELLO消息，其中version字段设置为最高OpenFlow交换机协议版本
由发送者支持（请参阅7.1。1）。此Hello消息可以选择包含一些OpenFlow元素
帮助建立连接（请参阅7.5.1）。收到此消息后，收件人必须计算
要使用的OpenFlow交换机协议版本。如果同时发送了Hello消息和Hello消息
收到的内容包含OFPHET_VERSIONBITMAP hello元素，并且如果这些位图具有一些公共位
设置，协商的版本必须是两个位图中设置的最高版本。否则，
版本必须小于已发送的版本号和在

版本字段。

如果收件人支持协商的版本，则连接继续。否则，
收件人必须回复*Hello Hello Failed*错误消息（请参见7.5.4.1），然后终止连接。

交换机和控制器交换OFPT_HELLO消息并成功协商后
一个通用的版本号，完成连接设置并可以创建标准OpenFlow消息
通过连接交换。控制器应该做的第一件事是发送一个

OFPT_FEATURES_REQUEST消息，用于获取交换机的数据*路径ID*（请参见7.3.1）。

6.3.4连接维护

OpenFlow连接维护主要由基础TLS或TCP连接机制完成。
从本质上讲，这确保了广泛的网络和条件都支持OpenFlow连接。
检测连接中断和终止连接的主要机制必须是TCP
超时和TLS会话超时（如果可用）。

如果与特定控制器或交换机的连接是
终止或损坏，这不应导致与其他控制器或交换机的连接终止
（请参阅6.3。7）。当相应的主连接为时，辅助连接必须终止。
另一方面，如果辅助连接已终止或断开，则终止或断开（参见6.3.8）
这不应影响主连接或其他辅助连接。

由于网络状况或超时而终止连接时，如果交换机或控制器
连接的发起者，它应尝试重新连接到另一方，直到新的连接-
建立连接或直到另一方的连接URI从其配置中删除。
完成重新连接尝试后，应以增加的间隔进行操作，以免双方都感到不知所措
网络和另一方，初始间隔应大于基础完整TCP
连接建立超时。

OpenFlow消息乱序处理（请参阅6 .2），并且某些请求由
切换可能会花费很长时间，因此控制器绝不能因为请求而终止连接

花费太多时间。该规则的例外是*回声回复*，控制器或开关可能
如果发送的回显请求的回复花费太多时间，则终止连接，但是必须终止连接
是禁用该功能的一种方法，并且超时应足够大以容纳各种各样的功能
条件。

控制器或交换机可能会在收到OpenFlow错误消息时终止连接，例如
例如，如果找不到通用的OpenFlow版本（请参阅6.3。3），或者找不到必需的功能
在另一方的支持下。在这种情况下，终止连接的一方不应尝试
自动重新连接。

流控制也使用基础TCP机制（如果可用）完成。对于连接
基于TCP，如果交换机或控制器无法足够快地处理传入的OpenFlow消息，
它应该停止服务该连接（停止从套接字读取消息）以引发TCP流
控制以停止发送者。控制器应监视其发送队列，并避免发送队列太长
大。对于基于UDP的辅助连接，如果交换机或控制器无法处理传入
OpenFlow消息足够快，它应该丢弃多余的消息。

6.3.5连接中断

如果交换机与控制器失去联系，则必须向所有控制器发送控制器状态消息
其余连接的控制器（如果有）指示受影响的角色和控制通道状态
控制器。如果交换机与多个控制器失去联系，则必须发送多个控制器状态
消息，每个受影响的控制器一个。这使其余的控制器即使采取行动也可以采取行动
也失去了彼此的沟通。重新建立OpenFlow频道后，
交换机必须向所有控制器发送更新的控制器状态消息。

如果由于回显请求超时导致交换机与所有控制器失去联系，则TLS
会话超时或其他断开连接，交换机必须立*即*进入“故障安全模式”
或“失败独立模式”，具体取决于交换机的实现和配置。在“失败保护”中
模式”，交换行为的唯一变化是发往控制器的数据包和消息是
掉了。流条目应根据其在“故障安全模式”下的超时值继续到期。在
“独立失败模式”，交换机使用OFPP_NORMAL保留端口处理所有数据包；其他
换而言之，该交换机充当旧式以太网交换机或路由器。在“失败独立模式”下，
交换机可以随意使用流表，交换机可以删除，添加或修改任何流
条目。通常，“故障独立模式”仅在混合交换机上可用（请参阅5.1）。

重建OpenFlow通道后，此时流表中会出现流条目
保留，并恢复正常的OpenFlow操作。如果需要，控制器可以选择
读取所有带有*flow-stats*请求的流条目（请参见7.3.5.2）以将其状态与
切换状态。或者，控制器然后可以选择使用*flow-mod*删除所有流条目
请求（请参阅6.4）从交换机的干净状态开始。

交换机首次启动时，将以“故障安全模式”或“故障独立模式”运行
模式，直到成功连接到控制器为止。配置默认流条目集
启动时使用的功能不在OpenFlow交换协议的范围内。

6.3.6加密

OpenFlow协议的默认安全性机制是TLS（传输层安全性）。的
交换机和控制器可以通过TLS连接进行通信，以提供身份验证和
连接的加密。交换机和控制器应该使用TLS的安全版本; 在
发布时，我们建议使用TLS 1.2版或更高版本。

如果有效的连接URI将tls指定为协议，则TLS连接将由打开设备启动
启动到控制器，该控制器正在用户指定的TCP端口或默认TCP上侦听
端口6653。可选地，TLS连接由控制器启动到正在监听的交换机
在用户指定的TCP端口或默认的TCP端口6653上。

交换机和控制器通过交换特定于站点的签名的证书来相互认证
私钥（推荐）。每个交换机必须是用户可配置的，带有自己的证书和一个
特定于站点的公共证书，用于验证控制器。另外，开关
可以选择支持多个CA证书的配置以及维护证书
与多个控制器建立连接时的吊销列表（CRL）。推荐
使用以下命令配置和管理所有相关的安全凭证（密码设置，密钥和证书）
交换机管理协议，例如OpenFlow配置协议。

或者，对于某些部署，交换机还可以支持自签名控制器证书或
使用预共享密钥交换来认证实体（不推荐）。

交换机和控制器可以选择使用纯TCP进行通信。普通TCP可以用于
非加密通讯（不建议使用），或实施其他安全机制，
例如使用IPsec，VPN或单独的物理网络，则此类配置的详细信息不在
规格。如果有效的连接URI将tcp指定为协议，则将启动TCP连接
通过启动到控制器的开关，该控制器正在用户指定的TCP端口上侦听
默认的TCP端口6653。可选地，TCP连接由控制器启动到交换机，
正在用户指定的TCP端口或默认的TCP端口6653上监听。
普通TCP，建议使用其他安全措施以防止控制器被窃听
模拟或其他对OpenFlow频道的攻击。

6.3.7多个控制器

交换机可以与单个控制器建立通信，也可以建立通信
与多个控制器。由于可以继续进行切换，因此拥有多个控制器可以提高可靠性
如果一个控制器或控制器连接失败，则以OpenFlow模式运行。之间的交接
控制器由控制器本身启动，从而可以从故障中快速恢复，并且还可以
控制器负载平衡。控制器之间相互协调交换机的管理
通过本规范范围之外的机制以及多重控制器的目标
功能仅用于帮助同步控制器执行的控制器切换。多
ple控制器功能仅解决控制器故障转移和负载平衡，而不能解决
虚拟化可以在OpenFlow交换协议之外完成。

启动OpenFlow操作时，交换机必须连接到配置了它的所有控制器，
并尝试同时保持它们之间的连通性。许多控制器可能会发送控制器-
将命令切换到交换机，与这些命令相关的答复或错误消息只能是

在与该命令关联的控制器连接上发送。异步消息可能需要
被发送到多个控制器，对于每个合格的OpenFlow通道和每个
当相应的控制器连接允许时发送的消息。

控制器的默认角色是OFPCR_ROLE_EQUAL。在这个角色中，控制器拥有对
开关，并且等于具有相同角色的其他控制器。默认情况下，控制器接收所有开关

异步消息（例如入站，已删除流）。控制器可以发送控制器到开关
命令来修改交换机的状态。交换机不进行任何仲裁或资源共享
控制器之间。

控制器可以请求将其角色更改为OFPCR_ROLE_SLAVE。在这个角色中，控制器
对交换机具有只读访问权限。默认情况下，控制器不接收异步开关
消息，除了端口状态消息。控制器被拒绝执行所有功能
从控制器到交换机的命令，用于发送数据包或修改交换机的状态。例如，

OFPT_PACKET_OUT，OFPT_FLOW_MOD，OFPT_GROUP_MOD，OFPT_PORT_MOD，OFPT_TABLE_MOD请求以及
OFPMP_TABLE_FEATURES具有非空主体的多部分请求必须被拒绝。如果有
拖曳器发送这些命令之一，交换机必须以“*从属*”错误消息答复（请参阅
7.5.4。2）。其他控制器到交换机的消息，例如OFPT_ROLE_REQUEST，OFPT_SET_ASYNC和

OFPT_MULTIPART_REQUEST仅查询数据，应正常处理。

控制器可以请求将其角色更改为OFPCR_ROLE_MASTER。这个角色类似于

OFPCR_ROLE_EQUAL并具有对交换机的完全访问权限，不同之处在于，该交换机确保
担任此角色的唯一控制器。当控制器将其角色更改为OFPCR_ROLE_MASTER时，开关
将角色为OFPCR_ROLE_MASTER的当前控制器更改为角色OFPCR_ROLE_SLAVE，但是
不会影响具有OFPCR_ROLE_EQUAL角色的控制器。当交换机执行此类角色时，如果
控制器角色从OFPCR_ROLE_MASTER更改为OFPCR_ROLE_SLAVE，此交换机必须生成一个
该控制器的控制器角色状态事件，以通知其新状态（在许多情况下，该控制器
不再可访问，并且交换机可能无法传输该事件）。

每个控制器可以发送OFPT_ROLE_REQUEST消息以将其角色传达给交换机（请参阅
7.3.8），并且交换机必须记住每个控制器连接的角色。控制器可能会改变
只要消息中的generation_id是最新的（见下文），它就可以随时扮演其角色。

角色请求消息提供了一种轻量级的机制来帮助控制器主选举过程，
控制器配置其角色，通常仍然需要在它们之间进行协调。开关
不能单独更改控制器的状态，由于
来自一个控制器的请求。任何从控制器或相等控制器都可以选举自己为主。一种
开关可以*同时*连接到处于相等状态的多个控制器，而处于
从属状态，最多有一个控制器处于主控状态。处于主控状态（如果有）的控制器，以及
所有处于平等状态的控制器都可以完全更改开关状态，没有强制执行的机制
这些控制器之间的开关分区。如果担任主角色的控制器需要
只有能够在交换机上进行更改的控制器，则没有控制器应该处于平等状态，并且所有
其他控制器应处于从属状态。

控制器也可以使用OFPT_ROLE_REQUEST消息来设置交换机使用的ID号。
控制器状态消息。当OpenFlow连接遍历NAT时，这特别有用
更改IP标识符的边界，因为Controller-Status消息中的URI始终表示
交换机的连接视图（请参阅6.3.1）。

控制器还可以控制通过其OpenFlow发送哪些类型的交换机异步消息。
频道，并更改上述默认设置。这是通过*异步配置完成的*
消息（请参阅6 .1.1），列出需要启用或过滤掉的每种消息类型的所有原因（请参阅
7.3.10）的特定OpenFlow通道。使用此功能，不同的控制器可以接收不同的
通知，处于主模式的控制器可以有选择地禁用它不关心的通知，
从属模式下的控制器可以启用它想要监视的通知。

要在主/从转换期间检测乱序消息，请使用OFPT_ROLE_REQUEST消息
包含一个64位序列号字段generation_id，用于标识给定的主视图。如
主选举机制的一部分，控制者（或代表他们的第三方）协调
generation_id的分配。generation_id是一个单调递增的计数器：一个新的（较大的）

每次更改主控权视图时（例如，指定新主控器时）都会分配generation_id。
generation_id可以环绕。

在收到角色等于OFPCR_ROLE_MASTER或OFPCR_ROLE_SLAVE的OFPT_ROLE_REQUEST时，
交换机必须将消息中的generation_id与迄今为止看到的最大生成ID进行比较。
世代ID小于先前看到的世代ID的消息必须被认为是过时的
并丢弃，并且交换机必须回复*Stale*错误消息（请参见7.5.4.12）。

以下伪代码描述了交换机在处理generation_id时的行为。

在交换机启动时：

```
generation_is_defined = false;
```

在收到角色等于OFPCR_ROLE_MASTER或OFPCR_ROLE_SLAVE的OFPT_ROLE_REQUEST时，
使用给定的generation_id，例如GEN_ID_X：

```
如果（generation_is_defined AND
      distance（GEN_ID_X，cached_generation_id）<0）{
   <丢弃OFPT_ROLE_REQUEST消息>;
   <发送一条错误消息，代码为OFPRRFC_STALE>;
}其他{
   cached_generation_id = GEN_ID_X;
   generation_is_defined = true;
   <正常处理消息>;
}
```

其中distance（）是*包装序列号距离*运算符，定义如下：

距离（a，b）：=（int64_t）（a-b）

即distance（）是序列号之间的无符号差，被解释为两个数字
签名的值。如果a大于b（在圆形意义上），这将导致正距离，但
小于“一半的序列号空间”。否则会导致负距离
（a <b）。

如果OFPT_ROLE_REQUEST中的角色是OFPCR_ROLE_EQUAL，则交换机必须忽略generation_id，
因为generation_id专门用于消除主/从种族条件的歧义
过渡。

6.3.8辅助连接

默认情况下，OpenFlow交换机和OpenFlow控制器之间的*OpenFlow通道*是单个
网络连接。OpenFlow通道也可以由一个主连接和多个
辅助连接。辅助连接由OpenFlow交换机创建，有助于
提高交换处理性能并利用大多数交换实现的并行性。
辅助连接始终由交换机启动，但可以由控制器进行配置。

从交换机到控制器的每个连接都由交换机的数据*路径ID*和一个*Auxiliary ID*（见7.3。1）。主连接的辅助ID必须设置为零，而辅助连接
连接必须具有非零的辅助ID和相同的数据路径ID。辅助连接必须
使用与主连接相同的源IP地址，但是可以使用其他传输方式，例如
TLS，TCP，DTLS或UDP，具体取决于交换机配置。辅助连接应
与主连接具有相同的目标IP地址和传输目标端口，
除非交换机配置另有规定。控制器必须识别传入连接
使用非零辅助ID作为辅助连接，并将它们绑定到具有
相同的数据路径ID。

在完成连接设置之前，交换机不得启动辅助连接
主连接（请参阅6.3.3），它必须建立并维护与控制器的辅助连接
仅在相应的主连接处于活动状态时。辅助连接的连接设置
与主连接相同（请参见6.3.3）。如果交换机检测到主连接
控制器损坏，必须立即关闭与该控制器的所有辅助连接，才能启用
控制器以正确解决数据路径ID冲突。

OpenFlow交换机和OpenFlow控制器都必须接受任何OpenFlow消息类型和
所有连接上的子类型：主连接或辅助连接不限于
特定的消息类型或子类型。但是，不同消息类型或
不同连接上的子类型可能不同。交换机可以为以下设备提供辅助连接
不同的优先级，例如一个辅助连接可能专用于高优先级请求
并且始终在其他辅助连接之前由交换机处理。交换机配置，用于
使用OpenFlow配置协议的示例，可以选择配置辅助设备的优先级
连接。

对OpenFlow请求的回复必须在其进入的同一连接上进行。
连接之间的同步，以及在不同连接上发送的消息可能会在
任何命令。障碍消息仅适用于使用该消息的连接（请参阅6.2）。辅助的
使用DTLS或UDP的连接可能会丢失或重新排序消息，OpenFlow不提供排序
或这些连接的交付保证（请参见6.2）。如果必须按顺序处理消息，则它们
必须通过相同的连接发送，使用不重新排序数据包的连接，并使用屏障
消息。

控制器可以自由使用各种交换机连接来发送整个OpenFlow消息
但是，为了最大程度地发挥大多数交换机的性能，建议遵循以下准则：

- 所有非Packet-out的OpenFlow控制器请求（flow-mod，统计请求...）都应

通过主连接发送。
•连接维护消息（您好，回显请求，功能请求）应在
主连接和每个辅助连接（如果需要）。

•所有包含来自Packet-In消息的数据包的Packet-Out消息都应在
Packet-In来自的连接。

•所有其他Packet-Out消息应使用
保持相同流的数据包映射到相同连接的机制。

•如果所需的辅助连接不可用，则控制器应使用主连接。

交换机可以随意使用各种控制器连接来发送OpenFlow消息，
但是，建议遵循以下准则：

•所有非Packet-in的OpenFlow消息都应通过主连接发送。

•使用保持机制，所有Packet-In消息分布在各个辅助连接上
相同流的数据包映射到相同连接。

不可靠传输（UDP，DTLS）上的辅助连接还有其他限制
以及不适用于其他传输方式（TCP，TLS）上的辅助连接的规则。唯一的
不可靠的辅助连接支持的消息类型为OFPT_HELLO，OFPT_ERROR，
OFPT_ECHO_REQUEST， OFPT_ECHO_REPLY， OFPT_FEATURES_REQUEST， OFPT_FEATURES_REPLY，
OFPT_PACKET_IN，OFPT_PACKET_OUT和OFPT_EXPERIMENTER，其他消息类型不支持
由规范移植。每个UDP数据包必须包含整数个OpenFlow
信息; 许多OpenFlow消息可能会在同一UDP数据包和一个OpenFlow消息中发送
不能在UDP数据包之间拆分。建议使用UDP和DTLS连接进行部署
配置网络和交换机，以避免UDP数据包的IP分片。

在不可靠的辅助连接上，在连接启动时会发送*Hello消息*以设置
连接（请参阅6.3。3）。如果OpenFlow设备在不可靠的辅助设备上收到另一条消息
连接在收到*Hello消息*之前，设备必须假定已建立连接
正确并使用该消息中的版本号，否则它必须返回错误消息，并带有
OFPET_BAD_REQUEST类型和OFPBRC_BAD_VERSION代码。如果OpenFlow设备收到错误
非可靠辅助连接上的消息，其类型为OFPET_BAD_REQUEST，且代码为OFPBRC_BAD_VERSION
否，它必须发送新的*Hello消息*或终止不可靠的辅助连接（
稍后可以重试连接）。如果在辅助连接上未收到任何消息
在某些实现方式下，如果选择的时间少于5秒，则设备必须发送
新的*Hello*消息或终止不可靠的辅助连接。如果发送*功能请求后*
消息，选择某些实现后，控制器未收到*功能回复*消息
如果时间少于5秒，则设备必须发送新的*功能请求*消息或
终止不可靠的辅助连接。如果在收到消息后，设备未收到
在某种实施方式下选择的时间少于30秒的任何其他消息，设备
必须终止不可靠的辅助连接。如果设备收到不可靠的消息
辅助连接已终止，必须假定它是新连接。

使用不可靠辅助连接的OpenFlow设备应遵循RFC 5405中的建议
若有可能。大多数情况下，不可靠的辅助连接都可以视为封装隧道。
除OFPT_PACKET_IN和OFPT_PACKET_OUT消息外，禁止OpenFlow消息。如果
OFPT_PACKET_IN消息中封装的某些数据包不是TCP数据包还是没有
属于具有适当拥塞控制的协议，控制器应将交换机配置为
由仪表或队列处理这些消息以管理拥塞。

6.4流表修改消息

流表修改消息可以具有以下类型：

```
of p_flow_mod_command枚举{
    OFPFC_ADD              = 0，/ *新流程。* /
    OFPFC_MODIFY           = 1，/ *修改所有匹配的流。* /
    OFPFC_MODIFY_STRICT = 2，/ *修改严格匹配通配符的条目，并且
                                优先。* /
    OFPFC_DELETE           = 3，/ *删除所有匹配的流。* /
    OFPFC_DELETE_STRICT = 4，/ *删除与通配符严格匹配的条目，并且
                                优先。* /
};
```

对于设置了OFPFF_CHECK_OVERLAP标志的添加请求（OFPFC_ADD），交换机必须首先检查是否存在
请求表中的所有重叠流条目。如果单个数据包可能会重叠两个流条目
两者都匹配，并且两个流条目具有相同的优先级，但是两个流条目的优先级不同
相同的比赛。如果现有流条目和添加请求之间存在重叠冲突，则切换
必须拒绝添加并以*Overlap*错误消息响应（请参见7.5.4.6）。

对于非重叠添加请求，或没有重叠检查的请求，交换机必须插入流
请求表中的条目。如果具有相同匹配字段和优先级的流条目已驻留在
请求的表，然后必须从表中清除该条目及其持续时间，并且
添加了新的流条目。如果设置了OFPFF_RESET_COUNTS标志，则必须清除流条目计数器，
否则，应从替换的流条目中复制它们。不会产生流删除消息
作为添加请求的一部分消除的流条目；控制器是否要删除流
它应该在添加新流条目之前显式发送对旧流条目的删除请求。

对于修改请求（OFPFC_MODIFY或OFPFC_MODIFY_STRICT），如果表中存在匹配项，
此条目的指令字段将替换为请求中的值，而其Cookie是

idle_timeout，hard_timeout，重要性，标志，计数器和持续时间字段保持不变。如果
如果设置了OFPFF_RESET_COUNTS标志，则必须清除流条目计数器。对于修改请求，如果
当前驻留在请求表中的流条目与该请求不匹配，未记录任何错误，并且
不会修改流表。

对于删除请求（OFPFC_DELETE或OFPFC_DELETE_STRICT），如果表中存在匹配的条目，
必须删除它，并且如果该条目设置了OFPFF_SEND_FLOW_REM标志，则它应生成一个流
删除的消息。对于删除请求，如果请求表中当前没有流条目匹配
该请求，不会记录任何错误，也不会发生流表修改。

修改和删除流mod命令具有*非严格*版本（OFPFC_MODIFY和OFPFC_DELETE）
和*严格的*版本（OFPFC_MODIFY_STRICT或OFPFC_DELETE_STRICT）。在*严格*版本中，
匹配字段，所有匹配字段（包括其掩码和优先级）均严格匹配
条目，只有相同的流条目被修改或删除。例如，如果要删除的消息
发送的条目中不包含匹配字段，OFPFC_DELETE命令将删除所有流
表中的条目，而OFPFC_DELETE_STRICT命令只会删除
适用于指定优先级的所有数据包。

对于*非严格的***Modify**和**Delete**命令，所有与流mod描述匹配的流条目
被修改或删除。这些命令不能部分应用：如果一个流条目不能

修改或删除，则所有匹配流条目都不会被修改或删除，并且会出现错误
回到。在*非严格*版本中，当流条目完全匹配或更多时，将发生匹配
比flow mod命令中的描述具体；在流程模块中，缺少的匹配字段是
通配符，字段掩码处于活动状态，其他流程模块字段（例如优先级）将被忽略。例如，
如果OFPFC_DELETE命令说要删除目标端口为80的所有流条目，则流
通配符所有匹配字段的条目将不会被删除。但是，一个OFPFC_DELETE命令可以
通配符所有匹配字段将删除与所有端口80流量匹配的条目。同样的解释
混合通配符和完全匹配字段的组合也适用于单个和汇总流统计信息请求。

可以按目的地组或输出端口筛选删除命令。如果out_port字段
包含非OFPP_ANY的值，则在匹配时会引入约束。这个约束是
每个匹配的流条目必须包含指向操作中指定端口的*输出*操作
与该流条目相关联。此约束仅限于直接与
流条目。换句话说，切换不得通过指向
组，它们可能具有匹配的*输出*操作。如果out_group与OFPG_ANY不同，则引入
对*小组*行动的类似限制。这些字段将被OFPFC_ADD，OFPFC_MODIFY和

OFPFC_MODIFY_STRICT消息。

如果cookie_mask字段包含以下内容，则还可以按Cookie值过滤修改和删除命令：
保留一个非0的值。此约束是cookie_mask在两个
流模块的cookie字段和流条目的cookie值必须相等。换一种说法，
（*flow entry.cookie*和*flow mod.cookie掩码*）==（*flow mod.cookie* &*flow mod.cookie掩码*）。

删除命令可以将OFPTT_ALL值用于table-id来指示匹配的流条目是

从所有流表中删除。
如果交换机支持驱逐，则流表可以配置为在流条目时执行驱逐
被添加到该表中（请参见7 .3.4.1）。如果在流表上启用了逐出，并且该流表是
已满，添加请求将使用驱逐机制尝试为新的流条目腾出空间。的
驱逐过程由开关定义。如果驱逐成功，则从中删除驱逐的流条目
流表（请参阅6.5），并将来自请求的新流条目插入流表中。如果驱逐是
如果失败，则不会删除任何流条目，并且添加请求将返回“ *表已满*”错误消息。

如果交换机无法在请求的表中找到任何要在其中添加传入流条目的空间，则
开关必须返回*Table Full*错误消息（请参见7.5.4.6）。

交换机必须验证内容*Flow-mod*消息，并且必须返回相应的错误消息-
指令错误（见7.5.4.4），匹配错误（见7.5.4.5），动作错误（
错误（请参见7.5.4.13）和其他flow-mod错误（请参见7.5.4.6）

6.5流量去除

按照控制器的要求，通过三种方式从流表中删除流条目
开关流量到期机制，或通过可选的开关逐出机制。

交换机流量到期机制由交换机独立于控制器运行，并且基于
流条目的状态和配置。每个流条目都有一个idle_timeout和hard_timeout
与之关联（请参阅7.3.4。2）。如果hard_timeout字段不为零，则交换机必须记下流
条目的到达时间，因为稍后可能需要逐出该条目。hard_timeout字段非零会导致

在给定的秒数后要删除的流条目，无论它有多少个数据包
已匹配。如果idle_timeout字段不为零，则交换机必须记下最后一个的到达时间
与流关联的数据包，因为稍后可能需要逐出该条目。非零的idle_timeout字段
导致流条目在给定的秒数内未匹配任何数据包时被删除。
交换机必须实现流到期并在其中之一进入流表时将其从流表中删除
已超过超时时间。

控制器可以通过发送删除流表修改来主动从流表中删除流条目。
消息（OFPFC_DELETE或OFPFC_DELETE_STRICT-参见6 .4）。流条| ¦被删除
作为由控制器移除组（参见6.7）或仪表（参见6.8）的结果。另一方面
当添加，修改或删除端口时，流条目永远不会删除或修改，控制器
如果需要，必须显式删除这些流条目（请参见4.6）。

当交换机需要回收资源时，可以从流表中逐出流条目（请参见6.4）。
流条目驱逐仅在显式启用流表的流表上发生（请参阅7.3。3）。流输入
驱逐是一项可选功能，用于选择驱逐哪些流条目的机制是switch
定义，并且可能取决于流条目参数，交换机中的资源映射以及其他内部
切换约束。

当流条目被删除时，可以通过控制器，流到期机制或可选开关
驱逐机制，交换机必须检查流条目的OFPFF_SEND_FLOW_REM标志。如果设置了此标志，
交换机必须向控制器发送流删除消息（请参见7.4。2）。每个流删除的消息
包含流程条目的完整说明，删除原因（到期，删除或逐出），
删除时的流条目持续时间，以及删除时的流统计信息。

6.6流表同步

一个流表可以可选地与另一个流表*同步*。同步流表后，
交换机自动更新其内容，以反映它同步的流表中的更改
用。此机制使交换机可以对同一数据的不同视图执行多个匹配
在OpenFlow管道的不同位置。

流表可以单向同步（一个流表与另一个流表同步）
或双向（两个流表彼此同步），一个开关也可以定义一个流表
从多个表同步或自身同步的表。流表的同步是
如表功能消息中所述，每个同步流表都包括一个表属性，用于标识
同步源流表（请参见7.3.5.18）。对于同步流表，当流
在源流表中添加，修改或删除该条目，则必须是对应的流条目
由交换机自动在同步流表中添加，修改或删除。如果有流条目
在源流表中添加了OFPFF_CHECK_OVERLAP标志，并且如果
条目将与已存在于同步流表中的流条目重叠，交换机必须拒绝
完整的flow-mod请求，并显示“*无法同步*”错误消息（请参见7.5.4.6）。如果有流条目
在源流表中添加，并且相应的流条目具有相同的匹配项和优先级

对于已存在于同步流表中的较旧的流条目，现有流条目必须为
如果替换为同步流条目，则不会为旧流条目生成流删除消息。
如果在源流表中修改或删除了流条目，并且对应的流条目为否
如果同步流表中的流条目不再存在，则不应在其中修改，删除或创建任何流条目
同步流表。

在同步表中创建的流条目可能与其中的相应流条目不同
源流表，因为它们通常是同一数据的不同视图。那些之间的翻译
同步流条目不是由OpenFlow协议指定的，而是取决于交换机的实现-
设计和配置。例如，同步流条目可能具有不同的指令
设置和操作，并且匹配字段可以进行转置（源和目标取反）。如果流
表是双向同步的，转换必须是双射的，每个转换必须是
与其他翻译完全相反，因此翻译后的流条目将始终翻译回
原始源流条目。流条目的计数器必须独立于流中的计数器保存。
源流表中，这些计数器不同步。通常，创建同步流条目
作为永久流条目（两个过期计时器均设置为零），以便其生命周期得到正确同步
源流表中的流条目。

该开关可以选择允许控制器修改或删除那些自动添加的流条目
在同步表中，并且还可能允许控制器在同步表中创建新条目
表。这使控制器能够自定义交换机之外的那些同步流条目
自动翻译。如果修改了源流条目，这些更改可能会丢失。如果流表
同步是双向的（两个流表彼此同步），更改由
由同步机制创建的流条目上的控制器需要重新反映在
其他同步流表（就像该流表中的非同步流条目一样）。

交换机可能对流表同步有限制。如果交换机无法同步流
控制器在源流表中生成的mod请求，交换机必须拒绝流mod
并立即返回“*无法同步*”错误消息（请参见7.5.4.6）。如果控制器尝试添加，
修改或删除同步流表中的流条目，并且交换机不支持此类更改，
交换机必须拒绝flow mod并立即返回*Is Sync*错误消息（请参见7.5.4.6）。

流表同步功能在理论上可以用来描述复杂的同步
交互，例如从多个表同步表或从已同步表同步
表。本规范建议仅描述简单和通用的同步配置-
位置，例如仅涉及两个表的位置。特别是，应避免同步循环。

使用同步表的一个示例是映射以太网学习/转发硬件表
作为两个同步流表的集合进入OpenFlow。同步流表有多种方式
可以用于以太网学习/转发，该规范没有建议
使用方式。在这个说明性和简化的示例中，第一流表表示学习查找
并匹配以太网源地址和输入端口（以及其他可选字段，例如VLAN）
ID）。第二个流表表示转发查找并与以太网目标地址匹配
并设置输出端口。这两个表与以太网地址相互同步
在比赛中换位。根据学习老化情况，将第一个表中的流条目设置为过期
计时器，第二个表中的流条目被设置为没有到期计时器，并在出现以下情况时自动删除
第一个表中的相应流条目到期。控制器可以在以下任何一个中添加流条目
这两个流表，然后修改自动添加到
其他流表。使用同步表的另一个示例是为多播映射RPF检查
数据包作为流表。

6.7组表修改消息

组表修改消息包括命令和组的各个组成部分
该命令所需（请参见7.3.4.3）。Group-Mod消息中的命令可以是以下命令之一：

```
/ *组命令* /
ofp_group_mod_command枚举{
    OFPGC_ADD      = 0，            / *新组。* /
    OFPGC_MODIFY = 1，              / *修改所有匹配的组。* /
    OFPGC_DELETE = 2                / *删除所有匹配的组。* /
    OFPGC_INSERT_BUCKET = 3，/ *将操作存储桶插入到已经可用的存储桶中
                                匹配组中的操作段列表* /
    / * OFPGC _ ??? = 4，* /        / *保留供将来使用。* /
    OFPGC_REMOVE_BUCKET = 5，/ *删除所有操作存储桶或任何特定操作
                                匹配组中的存储桶* /
};
```

该消息在许多情况下都包含操作桶（请参阅5.10）。每个存储桶的操作集必须
使用与流程模块相同的规则进行验证（请参阅7 .5.4.3），并附加特定于组的规则
检查（请参阅7.5.4.7）。

带有**添加**命令（OFPGC_ADD）的group-mod请求会在组表中添加新的组条目。
如果指定的组已经存在，则交换机必须返回*Group Exist*错误消息（请参见7.5.4.7）。

对于**修改**请求（OFPGC_MODIFY），如果已经存在具有指定组标识符的组条目
在组表中，然后是该组条目的配置，包括其类型和操作存储区，
必须删除（清除），并且组条目必须使用请求中包含的新配置。
对于该条目，将保留（继续）组顶级统计信息，并重置组存储桶统计信息
（已清除）。如果指定的组尚不存在，则交换机必须拒绝该组mod并发送
一个*未知的组*错误消息（见7.5.4.7）。

对于**删除**请求（OFPGC_DELETE），如果当前没有具有指定组标识符的组条目
组表中已存在，不会记录任何错误，并且不会发生组表修改。否则，
组将被删除，并且在“组”操作中包含该组的所有流条目也将被删除。的
不需要为**删除**请求指定组类型。**删除**也不同于**添加**或**修改**
未指定存储分区的情况下，以后尝试**添加**组标识符将不会导致出现组
存在错误。如果一个人希望有效删除一个组，但仍保留使用它的流程条目，则该组可以
通过发送未指定存储区的**修改**来清除。

要用一条消息删除所有组，请指定OFPG_ALL作为组值。

对于**插入存储桶**请求（OFPGC_INSERT_BUCKET），如果具有指定组标识符的组条目
已经驻留在组表中，新的操作存储段会插入到当前表中的指定位置
动作存储区列表。对于该组，对现有组进行顶级统计信息和组存储桶统计信息
存储桶被保留（续）。

在类型为OFPGC_INSERT_BUCKET的请求中，command_bucket_id字段用于指定po-
在当前操作存储区列表中定位，以插入新的操作存储区。以下值

支持OFPGC_INSERT_BUCKET请求中的command_bucket_id字段：

- 如果command_bucket_id为OFPG_BUCKET_FIRST，则新操作存储段将插入到
  当前操作存储桶之前列表的开头，即整个当前操作
  存储桶列表将被推送并附加到新的操作存储桶中。如果当前动作为bucket
  list为空，新的动作存储桶仅替换当前的动作存储桶列表。

- 如果command_bucket_id为OFPG_BUCKET_LAST，则将在
  当前存储区列表，即将新的操作存储区添加到当前操作存储区列表中。如果
  当前操作存储桶列表为空，新的操作存储桶仅替换当前操作
  遗愿清单。

- 如果command_bucket_id是当前操作存储区列表中的有效标识符，则新操作存储区
  将在command_bucket_id值之后插入，即将插入新的操作存储桶
  在command_bucket_id中提到的存储区标识符值和该存储区的下一个存储区之间
  清单。如果command_bucket_id标识了最后一个存储桶，则command_bucket_id将为
  被视为OFPG_BUCKET_LAST。

如果command_bucket_id字段值不是以上保留值之一，或者当前不存在
在操作存储区列表中，交换机必须返回“ *未知存储区*”错误消息（请参见7 .5.4.7）。

对于**删除存储桶**请求（OFPGC_REMOVE_BUCKET），如果具有指定组ID的组条目为-
修饰符已驻留在组表中，并从当前表中删除所有存储桶或指定的存储桶
动作存储区列表。对于该组，剩余的组顶级统计信息和组存储桶统计信息
存储桶被保留（续）。

在类型为OFPGC_REMOVE_BUCKET的请求中，command_bucket_id字段用于指定标识符
从当前操作存储区列表中删除的存储区的数量。command_bucket_id的以下值
支持anOFPGC_REMOVE_BUCKET请求中的字段：

- 如果command_bucket_id为OFPG_BUCKET_ALL，则从组中删除所有操作存储桶，从而删除该组的当前操作存储区列表为空。
- 如果command_bucket_id为OFPG_BUCKET_FIRST，则从当前操作中删除第一个操作存储桶遗愿清单。
- 如果command_bucket_id为OFPG_BUCKET_LAST，则从当前操作中删除上一个操作存储桶遗愿清单。
- 如果command_bucket_id是当前操作存储区列表中的有效标识符，则删除指定的列表中的操作存储桶。

如果交换机支持，则可以将组链接起来，当至少一个组转发到另一组时，
或更复杂的配置。例如，快速重路由组可能有两个存储桶，其中每个
指向一个选择组。

交换机可能支持检查链接组时是否未创建循环：如果发送了组mod
这样就可以创建转发循环，交换机必须拒绝组mod，并且必须发送
*循环*错误消息（请参阅7.5.4.7）。如果交换机不支持这种检查，则转发行为
未定义。

交换机可能支持检查是否删除了其他组转发的组：
无法删除一个组，因为它被另一个组引用，它必须拒绝删除该组
条目，并且必须发送“ *链接组*”错误消息（请参见7.5.4.7）。如果交换机不支持
检查，转发行为未定义。

6.8仪表修改信息

仪表修改消息可以具有以下命令：

```
/ *仪表命令* /
ofp_meter_mod_command枚举{
    OFPMC_ADD    = 0,          / *新仪表。* /
    OFPMC_MODIFY = 1           / *修改指定的仪表。* /
    OFPMC_DELETE = 2           / *删除指定的仪表。* /
};
```

带添加命令（OFPMC_ADD）的电表修改请求在交换机中添加新的电表条目。如果
如果指定的电表已经存在，则交换机必须返回“ *电表存在*”错误消息（请参见7.5.4.13）。

对于修改请求（OFPMC_MODIFY），如果已经存在具有指定仪表标识符的仪表条目，
然后必须删除（清除）该仪表入口的配置，包括其标志和频段，
仪表条目必须使用请求中包含的新配置。对于该条目，仪表
保留顶层统计（续），重置仪表频带统计（清除）。如果有电表入口
具有指定仪表标识符的设备尚不存在，则交换机必须拒绝仪表mod
并发送*Unknown Meter*错误消息（请参阅7.5.4.13）。

对于删除请求（OFPMC_DELETE），如果当前没有带有指定仪表标识符的仪表条目
存在，没有记录错误，也没有进行仪表修改。否则，将仪表卸下，并且将所有
在其指令集中包含仪表的流量条目也将被删除。只有电表标识符
需要为删除请求指定，其他字段（如乐队）可以省略。

要用一条消息删除所有仪表，请指定OFPM_ALL作为仪表值。虚拟电表可以
删除所有仪表时永远不会被删除，也不会被删除。

6.9捆绑消息

6.9.1捆绑包概述

捆绑包是来自控制器的一系列OpenFlow修改请求，可作为一个应用
OpenFlow操作。如果捆绑软件中的所有修改均成功，则所有修改都将保留，
但是如果出现任何错误，则不会保留任何修改。

捆绑软件的第一个目标是将交换机上相关的状态更改分组，以便应用所有更改
在一起，或者没有一个被应用。第二个目标是更好地同步整个
OpenFlow交换机集：可以在每个交换机上准备并预先验证捆绑包，然后将其应用
大约在同一时间由控制器。

将捆绑包指定为捆绑添加的邮件中传送的所有邮件，添加邮件中具有相同bundle_id的邮件
特定的控制器连接（请参阅 7.3。9.7）。捆绑添加消息中传达的消息被格式化
像常规的OpenFlow消息一样，并且具有相同的语义。捆绑中包含的消息是
由于它们存储在分发包中，因此经过了预先验证，从而最大程度地减少了分发包时出现错误的可能性
应用。捆绑添加消息中包含的消息的应用程序推迟到
捆绑包已提交（请参阅7 .3.9.9）。

捆绑包是一项可选功能，并非所有的OpenFlow交换机都支持。开关不是
需要接受捆绑中的任意消息，交换机可能不接受
包，而交换机可能不允许将消息类型的所有组合都捆绑在一起（请参见
7.3.9。7）。例如，交换机不应允许在捆绑添加中嵌入捆绑消息
信息。至少，如果交换机支持捆绑销售商品，则它必须能够支持多个捆绑销售商品
以任何顺序配置flow-mods和port-mods。

6.9.2捆绑示例用法

控制器可以发出以下消息序列，以对以下消息应用一系列修改：
盖特。

1. OFPBCT_OPEN_REQUEST bundle_id
2. OFPT_BUNDLE_ADD_MESSAGE bundle_id *修改1*
3. OFPT_BUNDLE_ADD_MESSAGE bundle_id ...
4. OFPT_BUNDLE_ADD_MESSAGE bundle_id *修改n*
5. OFPBCT_CLOSE_REQUEST bundle_id
6. OFPBCT_COMMIT_REQUEST bundle_id

预期开关的行为如下。打开捆绑包时，修改将保存到
临时暂存区，但不生效。提交捆绑包后，分段中的更改
区域应用于交换机使用的状态（例如表）。如果一次修改发生错误，则否
更改应用于状态。

6.9.3捆绑包错误处理

捆绑软件中的OpenFlow消息部分必须先经过验证，然后才能存储在捆绑软件中（请参见
7.3.9.7）。对于控制器发送的每个消息，交换机必须验证消息的语法
有效并且消息中的所有功能都是受支持的功能，并立即返回错误
如果无法验证此消息，则显示此消息。交换机可以选择验证资源可用性
并且可能会在此时提交资源并产生错误。邮件添加到时会产生错误
捆绑软件未存储在捆绑软件中，并且捆绑软件未修改。

提交捆绑包后，大多数错误将已经被检测到并报告。其中一个
捆绑中包含的错误消息在提交期间可能仍然会失败，例如由于资源可用性。
在这种情况下，没有应用捆绑软件的消息部分，并且交换机必须生成错误消息
对应于故障（参见7.3.9 .9）。捆绑包中的邮件应具有唯一的xid以帮助匹配
消息错误。如果捆绑包中的所有消息部分均未生成错误消息，则开关
通知控制器捆绑软件的成功应用。

6.9.4捆绑原子修改

提交捆绑包必须是控制器原子的，即，查询交换机的控制器绝不能
看到中间状态，它必须看到无开关状态或所有开关状态
已应用捆绑软件中包含的修改。特别是，如果捆绑包失败，则控制器
不应收到由于部分应用捆绑软件而导致的任何通知。

如果指定了标志OFPBF_ORDERED（见7.3.9。3），该束的消息部分必须应用
严格按顺序排列，好像由OFPT_BARRIER_REQUEST消息分隔（但是没有OFPT_BARRIER_REPLY
生成）。如果未指定此标志，则无需按顺序应用消息。

如果指定了标志OFPBF_ATOMIC（见7.3.9。3），提交束还必须包原子，
即，来自输入端口或出包请求的给定包应不进行任何处理或
已应用所有修改。是否支持此标志取决于
切换硬件和软件架构。数据包和消息可以暂时排队
应用更改。由于转发/处理延迟的增加可能是不可接受的，
通常使用双缓冲技术。

如果未指定标志OFPBF_ATOMIC，则提交捆绑包不需要是数据包原子的。
数据包可能会由于部分应用捆绑而产生的中间状态进行处理，甚至
如果捆绑包提交最终失败。各种OpenFlow计数器也将反映部分
在这种情况下应用捆绑软件。

6.9.5捆绑并行

交换机必须支持在创建和交换过程中交换回显请求和回显回复消息。
捆绑的数量，交换机必须回复回显请求，而无需等待结束
束。捆绑包中不能包含回音请求和回音回复消息。同样，异步
交换机生成的消息不受捆绑影响，交换机必须发送状态事件
无需等待捆绑包的结束。

如果交换机支持多个控制器通道或辅助连接，则交换机必须保持
每个控制器-交换机连接都有一个单独的捆绑存储区。这允许多个捆绑包
在给定的交换机上并行增量填充。bundle_id名称空间特定于每个
控制器连接，因此控制器无需协调。

交换机还可以选择支持在同一控制器上并行填充多个捆绑软件
连接，方法是将包消息与不同的bundle_id复用。控制器可以创建和
发送多个包，每个包由唯一的bundle_id标识，然后可以在其中应用任何一个
通过在提交消息中指定其bundle_id来确定任何订单。相反，开关可能不允许
创建另一个捆绑包或接受在打开捆绑包与其中一个之间的任何常规OpenFlow消息
提交或丢弃它。

在某些实现中，当开关开始存储包时，它可能会锁定某些对象
捆绑包引用。如果同一或另一个控制器连接上的消息尝试修改
如果对象被捆绑包锁定，则交换机必须拒绝该消息并返回错误。开关是
不需要锁定捆绑包引用的对象。如果开关未锁定由a引用的对象
捆绑软件，如果通过其他控制器修改了这些对象，捆绑软件的应用可能会失败
连接。

6.9.6预定的捆绑包

一束提交请求可以包括一个*执行时间*，指定*当*束应commit-
泰德 收到预定捆绑包的交换机将捆绑包提交到与执行尽可能接近的位置
提交消息中指定的时间。

6.9.6.1计划捆绑程序

计划的捆绑程序过程如图6所示：

1.控制器通过发送OFPBCT_OPEN_REQUEST开始捆绑过程，并接收
从交换机回复。

2.控制器然后发送一组$\tilde{N}$ OFPT_BUNDLE_ADD_MESSAGE消息，对于一些$\tilde{N} \geq 1$。

3.然后，控制器可以发送OFPBCT_CLOSE_REQUEST。关闭请求是可选的，因此
控制器可以跳过此步骤。

4.控制器发送OFPBCT_COMMIT_REQUEST。OFPBCT_COMMIT_REQUEST包括
OFPBF_TIME标志，指示这是否是*计划的commit*。预定的提交重新

任务包含时间属性ofp_bundle_prop_time，其中包含计划的时间
预期交换机将应用捆绑软件。

5.交换机收到提交消息后，将在计划的时间*T s*应用捆绑，并且
发送OFPBCT_COMMIT_REPLY到控制器。

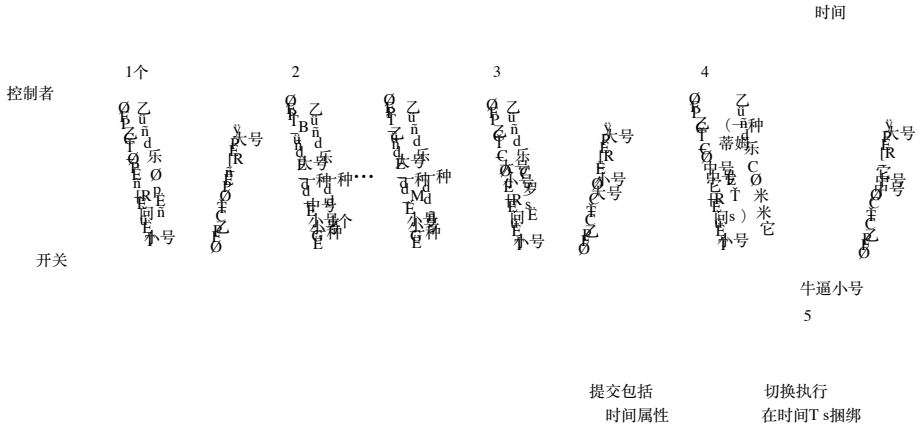


图6：预定捆绑程序

6.9.6.2丢弃预定的捆绑包

控制器可以通过发送类型为OFPT_BUNDLE_CONTROL的消息来取消已调度的捆绑包
OFPBCT_DISCARD_REQUEST。图7中显示了一个示例。如果交换机接收调度提交
消息，并且无法安排捆绑包，它将以错误消息响应控制器。这个
指示可用于实施协调更新，其中所有开关均成功
安排操作，或将包丢弃；当控制器收到调度错误消息时
它可以从其中一台交换机向其他交换机发送丢弃消息（图7中的步骤5'）
需要同时提交捆绑包，然后中止捆绑包。

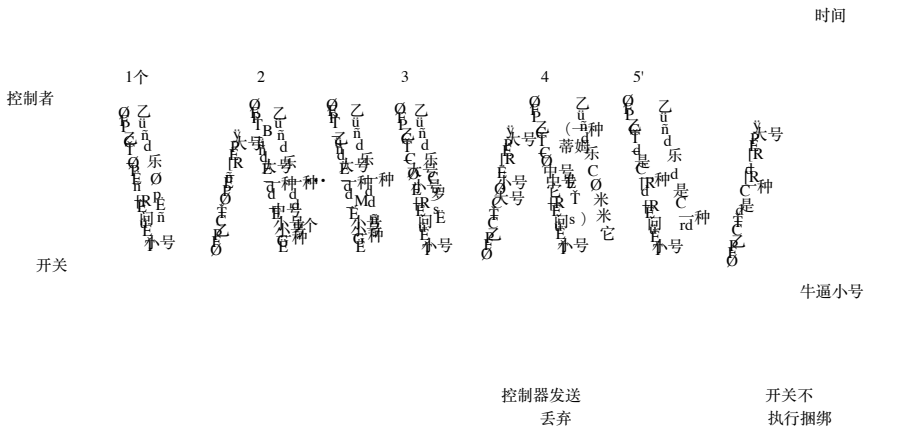


图7：丢弃预定的提交

6.9.6.3计时和同步

每个支持计划捆绑的交换机都必须维护一个时钟。假设时钟是

通过本规范或精确时间协议PTP（例如，网络时间协议）进行同步

有两个因素会影响交换机提交计划的捆绑包的准确性。一个因素是准确性
用来同步交换机时钟的时钟同步方法的数量，第二个因素是
交换机执行实时操作的能力，在很大程度上取决于其实现方式。

本规范未定义与执行精度有关的任何要求
预定的操作。但是，每个支持预定捆绑软件的交换机都必须能够报告

它对控制器的估计调度精度。控制器可以从
使用第7 .3.5.20节中定义的OFPMP_BUNDLE_FEATURES消息来进行切换。

由于交换机不会立即提交捆绑包，因此所需操作的处理时间
不容忽视；在预定的时间和执行时间总是指*开始时间*的
相关操作。

6.9.6.4安排公差

交换机收到调度的提交请求时，必须验证调度时间T_s是否未达到
过去或将来都太远了。如图8所示，交换机验证T_s是否在T_s范围内。
*计划公差*范围。

T_s的下限验证了捆束的新鲜度，从而避免作用于旧的并且可能的情况下
不相关的消息。同样，T_s的上限可确保开关不占用
长期承诺执行一个捆绑软件，该捆绑软件在预定的时间可能已过时
调用。

图8：计划公差

调度容差由两个参数sched_max_future和sched_max_past确定。
这两个参数的默认值为1秒。控制器可以将这些字段设置为其他字段
如7.3.5.20节所述，使用捆绑包功能请求获取值。

如果调度时间T_s在调度容限范围内，则提交调度捆绑。
如果T_s发生在过去且在调度容限之内，则交换机会尽快提交该捆绑包
可能。如果T_s是将来的时间，则交换机将束尽可能地靠近T_s提交。如果T_s不是
在调度容差范围内，交换机将通过错误消息响应控制器。

7 OpenFlow交换协议

OpenFlow交换器规范的核心是用于OpenFlow交换器的结构集
协议消息。

7.1协议基本格式

使用通过OpenFlow传输的OpenFlow消息实现OpenFlow协议
通道（请参阅6）。每种消息类型都由特定的结构描述，该结构从通用
OpenFlow标头（请参见7 .1.1），并且消息结构可以包括其他结构，这些结构可能是
多种消息类型通用（请参见7 .2）。每个结构都定义了信息的顺序
包含在消息中，并且可能包含其他结构，值，枚举或位掩码（请参见
7.1.3）。

下面描述的结构，定义和枚举均来自文件openflow.h，后者
包含在本文档中（请参阅附录A）。大多数结构都包含填充和8字节
对齐，由断言语句检查（请参阅7.1。2）。所有OpenFlow消息均以big-endian发送
格式。

OpenFlow是一个简单的二进制协议，因此由OpenFlow生成的无效OpenFlow消息
实现通常会导致消息长度错误或某些字段包含
无效值（请参阅7.1 .3）。建议监视这些条件以检测不符合要求
实现。

7.1.1 OpenFlow标头

每个OpenFlow消息均以OpenFlow标头开头：

```
/ *所有OpenFlow数据包的标头。* /
struct ofp_header {
        uint8_t版本;                / * OFP_VERSION。* /
        uint8_t类型;                / * OFPT_常量之一。* /
        uint16_t长度;               / *长度，包括此ofp_header。* /
        uint32_t xid;               / *与该数据包关联的交易ID。
                                        回复使用与请求中相同的ID
                                        方便配对。* /
};
OFP_ASSERT（sizeof（struct ofp_header）== 8）;
```

版本指定使用的OpenFlow交换机协议版本。最高位
版本字段是保留字段，必须将其设置为0。低7位表示版本的修订号。
协议。当前规范描述的协议版本为1.5.1，其*ofp版本*
是0x06。

长度字段指示消息的总长度，因此不使用其他框架来区分
下一帧。该类型可以具有以下值：

```
of_p的枚举{
        / *不可变的消息。* /
        OFPT_HELLO                  = 0，/ *对称消息* /
        OFPT_ERROR                  = 1，/ *对称消息* /
        OFPT_ECHO_REQUEST           = 2，/ *对称消息* /
        OFPT_ECHO_REPLY             = 3，/ *对称消息* /
        OFPT_EXPERIMENTER           = 4，/ *对称消息* /
```



```
        / *切换配置消息。* /
        OFPT_FEATURES_REQUEST   = 5，/ *控制器/开关消息* /
        OFPT_FEATURES_REPLY     = 6，/ *控制器/开关消息* /
        OFPT_GET_CONFIG_REQUEST = 7，/ *控制器/开关消息* /
        OFPT_GET_CONFIG_REPLY   = 8，/ *控制器/开关消息* /
        OFPT_SET_CONFIG         = 9，/ *控制器/开关消息* /

        / *异步消息。* /
        OFPT_PACKET_IN          = 10，/ *异步消息* /
        OFPT_FLOW_REMOVED       = 11，/ *异步消息* /
        OFPT_PORT_STATUS        = 12，/ *异步消息* /

        / *控制器命令消息。* /
        OFPT_PACKET_OUT         = 13，/ *控制器/开关消息* /
        OFPT_FLOW_MOD           = 14，/ *控制器/开关消息* /
        OFPT_GROUP_MOD          = 15，/ *控制器/开关消息* /
        OFPT_PORT_MOD           = 16，/ *控制器/开关消息* /
        OFPT_TABLE_MOD          = 17，/ *控制器/开关消息* /

        / *多部分消息。* /
        OFPT_MULTIPART_REQUEST      = 18，/ *控制器/开关消息* /
```

```
    OFPT_MULTIPART_REPLY          = 19，/ *控制器/开关消息* /

    / *障碍消息。* /
    OFPT_BARRIER_REQUEST      = 20，/ *控制器/开关消息* /
    OFPT_BARRIER_REPLY        = 21，/ *控制器/开关消息* /

    / *控制器角色更改请求消息。* /
    OFPT_ROLE_REQUEST         = 24，/ *控制器/开关消息* /
    OFPT_ROLE_REPLY           = 25，/ *控制器/开关消息* /

    / *异步消息配置。* /
    OFPT_GET_ASYNC_REQUEST = 26，/ *控制器/开关消息* /
    OFPT_GET_ASYNC_REPLY      = 27，/ *控制器/开关消息* /
    OFPT_SET_ASYNC            = 28，/ *控制器/开关消息* /

    / *仪表和速率限制器配置消息。* /
    OFPT_METER_MOD            = 29，/ *控制器/开关消息* /

    / *控制器角色更改事件消息。* /
    OFPT_ROLE_STATUS          = 30，/ *异步消息* /

    / *异步消息。* /
    OFPT_TABLE_STATUS         = 31，/ *异步消息* /

    / *通过交换机请求转发。* /
    OFPT_REQUESTFORWARD       = 32，/ *异步消息* /

    / *捆绑操作（多个消息作为一个操作）。* /
    OFPT_BUNDLE_CONTROL       = 33，/ *控制器/开关消息* /
    OFPT_BUNDLE_ADD_MESSAGE = 34，/ *控制器/开关消息* /

    / *控制器状态异步消息。* /
    OFPT_CONTROLLER_STATUS = 35，/ *异步消息* /

};
```


7.1.2填充

大多数OpenFlow消息都包含填充字段。这些包含在各种消息类型中，并且
在各种常见的结构中。这些填充字段中的大多数可以通过以下事实来识别：
名称以pad开头。填充字段的目标是在自然处理器上对齐多字节实体
边界。

消息中包含的所有通用结构在64位边界上对齐。各种其他类型
根据需要对齐，例如32位整数在32位边界上对齐。的例外
填充规则是从不填充的OXM匹配字段（请参阅7. 2.3.2）。一般而言，
除非明确指定，否则不填充OpenFlow消息。另一方面，共同的结构
几乎总是在最后填充。

填充字段应设置为零。OpenFlow实现必须接受在
填充字段，并且必须忽略填充字段的内容。

7.1.3保留和不受支持的值和位位置

大多数OpenFlow消息均包含枚举，例如用于描述类型，com-
要求或理由。规范定义了该协议版本使用的所有值，以及所有其他值
除非明确指定，否则保留值。不推荐使用的值也将保留。保留值
不应该在OpenFlow消息中使用。该规范还可以定义为
有些值是可选的，因此实现可能不支持这些可选值。如果是OpenFlow
实现接收到包含保留值或它不支持的可选值的请求，
它必须拒绝该请求并返回适当的错误消息。如果是OpenFlow实施
收到包含保留值或可选值的回复或异步消息，但不包含
支持，它应该忽略包含未知值的对象并记录错误。

一些消息包含位图（位数组），例如，它们用于编码配置
标志，状态位或功能。规范定义了此版本的版本使用的所有位位置
在协议中，所有其他位位置均保留，除非明确指定。不推荐使用的位位置是
也保留。OpenFlow消息中的保留位位置应设置为零。规格可能
还定义对某些位位置的支持是可选的，因此实现可能不支持
那些可选的位位置。如果OpenFlow实现收到包含保留的请求
不支持将其设置为1的位位置或可选位位置，它必须拒绝该请求，并且
返回适当的错误消息。如果OpenFlow实现收到回复或异步
包含保留位位置或不支持将其设置为1的可选位的消息

应该忽略该位的位置并记录一个错误。

一些消息包含TLV（类型，长度，值）。这些例如用于编码属性，
操作，匹配字段或结构中的可选属性。规范定义了所有TLV类型
除非明确指定，否则在此协议版本中使用的所有其他TLV类型都是保留的。弃用
保留的TLV类型也保留。保留的TLV类型不应在OpenFlow消息中使用。的
规范可能还会定义对某些TLV类型的支持是可选的，因此实现可能
不支持那些可选的TLV类型。如果OpenFlow实现收到包含以下内容的请求：
保留的TLV类型或它不支持的可选TLV类型，它必须拒绝请求并返回
相应的错误消息。如果OpenFlow实现收到回复或异步消息

包含保留的TLV类型或不支持的可选TLV类型，则应忽略TLV
并记录错误。

7.2通用结构

本节描述了多种消息类型使用的结构。

7.2.1端口结构

OpenFlow管道在端口上接收和发送数据包。开关可以定义物理和逻辑
端口，并且OpenFlow规范定义了一些保留端口（请参阅4.1）。

交换机上的每个端口由一个端口号（32位数字）唯一标识。预留端口有
规范定义的端口号。物理和逻辑端口的端口号由
开关，可以是从1开始到OFPP_MAX结尾的任何数字。端口号使用
以下约定：

```
/ *端口编号。端口从1开始编号。* /
枚举ofp_port_no {
	/ *物理和逻辑交换机端口的最大数量。* /
	OFPP_MAX = 0xffffff00，

	/ *保留的OpenFlow端口（伪输出“端口”）。* /
	OFPP_UNSET = 0xfffffff7，/ *未在操作集中设置输出端口。
                                    仅在OXM_OF_ACTSET_OUTPUT中使用。* /
	OFPP_IN_PORT = 0xfffffff8，/ *将数据包发送出输入端口。这个
                                    必须明确使用保留端口
                                    为了发回输入
                                    港口。* /
	OFPP_TABLE = 0xfffffff9，/ *将数据包提交到第一个流表
                                    注意：此目标端口只能是
                                    在打包消息中使用。* /
	OFPP_NORMAL = 0xfffffffa，/ *使用非OpenFlow管道转发。* /
	OFPP_FLOOD = 0xfffffffb，/ *使用非OpenFlow管道进行泛洪。* /
	OFPP_ALL = 0xfffffffc，/ *除输入端口外的所有标准端口。* /
	OFPP_CONTROLLER = 0xfffffffd，/ *发送给控制器。* /
	OFPP_LOCAL = 0xfffffffe，/ *本地开放流“端口”。* /
	OFPP_ANY = 0xffffffff / *在以下情况下在某些请求中使用的特殊值
                                    没有指定端口（即通配符）。* /
};
```

7.2.1.1端口描述结构

物理端口，交换机定义的逻辑端口和OFPP_LOCAL保留端口描述如下：
以下结构：

```
/ *端口说明* /
struct ofp_port {
	uint32_t port_no;
	uint16_t长度；
```

```
    uint8_t pad [2];
    uint8_t hw_addr [OFP_ETH_ALEN];
    uint8_t pad2 [2];                                / *对齐64位。* /
    字符名称[OFP_MAX_PORT_NAME_LEN]; / *空终止* /

    uint32_t配置;                    / * OFPPC_ *标志的位图。* /
    uint32_t状态;                    / * OFPPS_ *标志的位图。* /

    / *端口描述属性列表-0个或多个属性* /
    p_port_desc_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_port）== 40）;
```

port_no字段是端口号，它唯一地标识交换机内的端口。hw_addr
字段通常是端口的MAC地址；OFP_ETH_ALEN为6。名称字段以空值结尾
包含接口可读名称的字符串。OFP_MAX_PORT_NAME_LEN的值为
16。

config字段是一个位图，描述了端口管理设置，并且可能包含组合
以下标志之一：

```
/ *指示物理端口行为的标志。这些标志是
 *在ofp_port中用于描述当前配置。他们是
 *在ofp_port_mod消息中使用，用于配置端口的行为。
 * /
of_p_port_config的枚举{
    OFPPC_PORT_DOWN    = 1 << 0，/ *端口在管理上已关闭。* /

    OFPPC_NO_RECV      = 1 << 2，/ *丢弃端口接收的所有数据包。* /
    OFPPC_NO_FWD       = 1 << 5，/ *丢弃转发到端口的数据包。* /
    OFPPC_NO_PACKET_IN = 1 << 6        / *不要为端口发送输入包的消息。* /
};
```

OFPPC_PORT_DOWN位指示该端口已被管理性关闭，应
OpenFlow不使用它来发送流量。OFPPC_NO_RECV位指示在
该端口应被忽略。OFPPC_NO_FWD位指示OpenFlow不应发送数据包
到那个港口。OFPPC_NO_PACKET_IN位指示该端口上的生成表的数据包
miss绝不应触发向控制器发送入包消息。

通常，端口配置位由控制器之一设置，而不由交换机更改。那些
这些位可能对控制器实施诸如STP或BFD的协议很有用。端口配置
比特也可以通过交换机配置（例如使用OpenFlow配置）进行更改
协议。如果在交换机上更改了端口配置位，则交换机必须发送OFPT_PORT_STATUS
消息，通知控制器更改。

状态字段是描述端口内部状态的位图，可以包括以下各项的组合：
以下标志：

```
/ *物理端口的当前状态。这些不能从以下位置配置
 *控制器。
 * /
of_p_port_state的枚举{
```



```
    OFPPS_LINK_DOWN    = 1 << 0，/ *不存在物理链接。* /
    OFPPS_BLOCKED      = 1 << 1，/ *端口被阻塞* /
    OFPPS_LIVE         = 1 << 2，/ *为快速故障转移组而活。* /
};
```

端口状态位表示OpenFlow外部的物理链路或交换协议的状态。的

OFPPS_LINK_DOWN位指示物理链路不存在。OFPPS_BLOCKED位指示
OpenFlow外部的交换协议（例如802.1D生成树）阻止使用
带有OFPP_FLOOD的端口。

所有端口状态位都是只读的，不能由控制器更改。当端口标志是
更改后，交换机必须发送OFPT_PORT_STATUS消息以将更改通知控制器。

属性字段是端口描述属性的列表，描述了各种配置和状态
港口

7.2.1.2端口描述属性

当前定义的端口描述属性类型的列表是：

```
/ *端口描述属性类型。
 * /
of_p_port_desc_prop_type的枚举{
    OFPPDPT_ETHERNET          = 0，      / *以太网属性。* /
    OFPPDPT_OPTICAL           = 1        / *光学性质。* /
    OFPPDPT_PIPELINE_INPUT    = 2        / *入口管道字段。* /
    OFPPDPT_PIPELINE_OUTPUT   = 3，      / *出口管道字段。* /
    OFPPDPT_RECIRCULATE       = 4        / *再循环属性。* /
    OFPPDPT_EXPERIMENTER      = 0xFFFF，/ *实验者属性。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有端口描述属性的公共头。* /
struct ofp_port_desc_prop_header {
    uint16_t        类型;       / * OFPPDPT_ *之一。* /
    uint16_t        长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_header的结构== 4）；
```

OFPPDPT_ETHERNET属性使用以下结构和字段：

```
/ *以太网端口描述属性。* /
p_port_desc_prop_ethernet结构{
    uint16_t        类型;       / * OFPPDPT_ETHERNET。* /
    uint16_t        长度; / *此属性的长度（以字节为单位）。* /
    uint8_t         垫[4]; / *对齐64位。* /
    / * OFPPF_ *位图，用于描述功能。如果所有位清零
      *不支持或不可用。* /
    uint32_t curr;              / *当前功能。* /
```



```
    uint32_t广告;               / *端口正在通告的功能。* /
    uint32_t支持;               / *端口支持的功能。* /
    uint32_t对等体;             / *由对等方通告的功能。* /

    uint32_t curr_speed;        / *当前端口比特率，以kbps为单位。* /
    uint32_t max_speed;         / *最大端口比特率（kbps）* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_ethernet的结构== 32）；
```

curr，advertised，supported和peer对等字段是位图，它们指示链接模式（速度和双工），链接类型（铜缆/光纤）和链接功能（自动协商和暂停）。端口功能可能包括以下标志的组合：

```
/ *数据路径中可用端口的功能。* /
ofp_port_features枚举{
    OFPPF_10MB_HD     = 1 << 0，/ * 10 Mb半双工速率支持。* /
    OFPPF_10MB_FD     = 1 << 1，/ * 10 Mb全双工速率支持。* /
    OFPPF_100MB_HD    = 1 << 2，/ * 100 Mb半双工速率支持。* /
    OFPPF_100MB_FD    = 1 << 3，/ * 100 Mb全双工速率支持。* /
    OFPPF_1GB_HD      = 1 << 4，/ * 1 Gb半双工速率支持。* /
    OFPPF_1GB_FD      = 1 << 5，/ * 1 Gb全双工速率支持。* /
    OFPPF_10GB_FD     = 1 << 6，/ * 10 Gb全双工速率支持。* /
    OFPPF_40GB_FD     = 1 << 7，/ * 40 Gb全双工速率支持。* /
    OFPPF_100GB_FD    = 1 << 8，/ * 100 Gb全双工速率支持。* /
    OFPPF_1TB_FD      = 1 << 9，/ * 1 Tb全双工速率支持。* /
    OFPPF_OTHER       = 1 << 10，/ *其他速率，不在列表中。* /

    OFPPF_COPPER      = 1 << 11，/ *铜介质。* /
    OFPPF_FIBER       = 1 << 12，/ *纤维介质。* /
    OFPPF_AUTONEG     = 1 << 13，/ *自动协商。* /
    OFPPF_PAUSE       = 1 << 14，/ *暂停。* /
    OFPPF_PAUSE_ASYM = 1 << 15 / *非对称暂停。* /
};
```

可以同时设置多个这些标志。如果未设置任何端口速度标志，则max_speed或curr_speed被使用。

curr_speed和max_speed字段指示当前和最大比特率（原始传输速度）（以kbps为单位）。该数字应四舍五入以匹配常用用法。例如，

光纤10 Gb / s以太网端口应将此字段设置为10000000（而不是10312500），并且
OC-192端口应将此字段设置为10000000（而不是9953280）。

max_speed字段指示链接的最大已配置容量，而curr_speed
指示当前容量。如果端口是具有3个1 Gb / s容量链路的LAG，则其中一个端口
LAG关闭时，一个端口以1 Gb / s的速率自动协商，而一个端口以100 Mb / s的速率自动协商，
max_speed为3 Gb / s，curr_speed为1.1 Gb / s。

OFPPDPT_OPTICAL属性使用以下结构和字段：

```
/ *光端口描述属性。* /
struct ofp_port_desc_prop_optical {
    uint16_t            类型;       / * OFPPDPT_3OPTICAL。* /
    uint16_t            长度; / *此属性的长度（以字节为单位）。* /
```



```
    uint8_t             垫[4]; / *对齐64位。* /

    uint32_t支持;                / *端口支持的功能。* /
    uint32_t tx_min_freq_lmda;          / *最小发射频率/波长* /
    uint32_t tx_max_freq_lmda;          / *最大发射频率/波长* /
    uint32_t tx_grid_freq_lmda; / * TX网格间距频率/波长* /
    uint32_t rx_min_freq_lmda;          / *最小接收频率/波长* /
    uint32_t rx_max_freq_lmda;          / *最大接收频率/波长* /
    uint32_t rx_grid_freq_lmda; / * RX网格间距频率/波长* /
    uint16_t tx_pwr_min;                / *最小发射功率* /
    uint16_t tx_pwr_max;                / *最大发射功率* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_optical的结构）== 40）;
```

为发送和接收光端口指定了最小，最大和网格间距
作为以MHz为单位的频率或以nm * 100为单位的波长（λ）。对于不可调整的端口，
最小值和最大值将相同，并指定固定值。tx_pwr_min和
tx_pwr_max为dBm *10。光端口功能可能包括以下标志的组合：

```
/ *交换机中可用的光学端口功能。* /
of_p_optical_port_features的枚举{
    OFPOPF_RX_TUNE    = 1 << 0，/ *接收器可调* /
    OFPOPF_TX_TUNE    = 1 << 1，/ *传输可调* /
    OFPOPF_TX_PWR     = 1 << 2，/ *电源是可配置的* /
    OFPOPF_USE_FREQ = 1 << 3，/ *使用频率，而不是波长* /
};
```

OFPOPF_RX_TUNE表示端口接收功能能够调谐，OFPOPF_TX_TUNE表示
端口发送功能可以调整。OFPOPF_TX_PWR表示可以设置发射功率。
OFPOPF_USE_FREQ表示必须使用频率而不是波长来描述
端口调整。使用相同的调谐域（频率或波长）作为
表示，因为波长和频率之间的转换通常会由于范围而容易出错
值。

OFPPDPT_PIPELINE_INPUT和OFPPDPT_PIPELINE_OUTPUT属性使用以下结构
和字段：

```
/ *入口或出口管道字段。* /
struct ofp_port_desc_prop_oxm {
    uint16_t            类型;       / * OFPPDPT_PIPELINE_INPUT之一或
                                      OFPPDPT_PIPELINE_OUTPUT。* /
    uint16_t            长度; / *此属性的长度（以字节为单位）。* /
    /* 其次是：
     *     -恰好（长度-4）个字节包含oxm_id，然后
     *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t            oxm_ids [0];        / * OXM标头数组* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_oxm的结构）== 4）;
```


oxm_ids是端口支持的OXM类型的列表（请参阅7. 2.3.2）。该列表的元素是
非实验者OXM字段的32位OXM头或实验者OXM的64位OXM头
字段，那些OXM字段不包含任何有效负载。OXM标头中的oxm_length字段必须为
为OXM字段定义的长度值，即OXM字段具有有效负载的有效负载长度。
对于有效负载大小可变的实验者OXM字段，oxm_length字段必须为最大
有效载荷的长度。

OFPPDPT_PIPELINE_INPUT属性指示为数据包提供的管道字段的列表
在端口上收到（请参阅7.2.3.7）。端口在数据包上使用有用的值设置那些管道字段
它根据内部处理接收。如果OXM_OF_IN_PORT是唯一受支持的管道字段，则此
属性不需要包括在内。

OFPPDPT_PIPELINE_OUTPUT属性指示由管道使用的管道字段的列表。
端口，用于在该端口上输出的数据包（请参见7.2.3.7）。端口使用那些管道中设置的值
字段，供转发数据包时进行内部处理。如果不支持管道字段，则此属性
不需要包括在内。

OFPPDPT_RECIRCULATE属性使用以下结构和字段：

```
/ *再循环端口描述属性。* /
p_port_desc_prop_recirculate结构{
        uint16_t            类型;                   / * OFPPDPT_RECIRCULATE。* /
        uint16_t            长度;                   / *属性的字节长度，
                                                        包括此标头，不包括填充。* /
        /* 其次是：
          *     -确切地（长度-4）个字节包含端口号，然后
          *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
          *       全零字节的字节* /
        uint32_t            port_nos [0];           / *再循环的输入端口号列表。
                                                      0或更大。端口号数
                                                      从中的长度字段推断
                                                      标头。* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_recirculate的结构== 4）；
```

port_nos字段是输入端口的列表，当前端口上的数据包输出可在此循环使用
返回到OpenFlow管道（请参阅4.7）。仅当端口再循环时才必须使用此属性
如果数据包返回到OpenFlow管道，则其他端口不得使用此属性。对于环回或
在同一端口上返回的单向服务，此字段将仅包括当前端口号。
对于封装/解封装服务或其他双向服务，此字段将包括
代表服务另一端的其他端口号。如果再循环可以通过
任意端口号，列表将为空。

OFPPDPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者端口描述属性。* /
struct ofp_port_desc_prop_experimenter {
        uint16_t            类型;       / * OFPPDPT_EXPERIMENTER。* /
        uint16_t            长度; / *此属性的长度（以字节为单位）。* /
        uint32_t            实验者 / *采用相同的实验者ID
                                        形式如结构
                                        ofp_experimenter_header。* /
```



```
        uint32_t            exp_type;           / *实验者定义。* /
        /* 其次是：
          *     -确切地（长度-12个）字节包含实验者数据，然后
          *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
          *       全零字节的字节* /
        uint32_t            实验者数据[0];
};
OFP_ASSERT（sizeof（p_port_desc_prop_experimenter的结构）== 12）；
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.2.2标头类型结构

标头类型用于标识数据包标头的各个标头部分。标头类型为
使用ofp_header_type结构定义：

```
/ *标头类型结构。* /
struct ofp_header_type {
    uint16_t        命名空间          / * OFPHTN_ *之一。* /
    uint16_t        ns_type;          / *在名称空间中输入。* /
};
OFP_ASSERT（sizeof（struct ofp_header_type）== 4）；
```

名称空间字段标识用于标头类型的名称空间。大多数名称空间都很好
在OpenFlow协议之外管理的已知数据包类型列表。

ns_type字段是所选名称空间内的数据包类型。该字段的定义取决于
在名称空间上。

定义了以下名称空间：

```
ofp_header_type_namespaces的枚举{
    OFPHTN_ONF                  = 0，/ * ONF名称空间。* /
    OFPHTN_ETHERTYPE            = 1，/ * ns_type是一个以太类型。* /
    OFPHTN_IP_PROTO             = 2，/ * ns_type是IP协议号。* /
    OFPHTN_UDP_TCP_PORT         = 3，/ * ns_type是TCP或UDP端口。* /
    OFPHTN_IPV4_OPTION          = 4，/ * ns_type是IPv4选项号。* /
};
```

- OFPHTN_ONF是ONF命名空间。对于此命名空间，ns_type是定义的值之一
  下面。
- OFPHTN_ETHERTYPE是Ethertype命名空间。对于此命名空间，ns_type是一个Ethertype，
  由IEEE标准协会定义。
- OFPHTN_IP_PROTO是IP协议名称空间。对于此命名空间，ns_type是IP协议
  编号，由IETF定义。仅ns_type的低8位有效，高位
  位必须设置为零。

- OFPHTN_UDP_TCP_PORT是TCP UDP端口名称空间。对于此命名空间，ns_type是一个TCP
  或UDP端口号（由IETF定义）。
- OFPHTN_IPV4_OPTION是IPv4选项名称空间。对于此命名空间，ns_type是IPv4选项
  编号，由IETF定义。仅ns_type的低8位有效，高位
  位必须设置为零。

定义了ONF名称空间中ns_type的以下值：

```
枚举ofp_header_type_onf {
    OFPHTO_ETHERNET             = 0，/ *以太网（DIX或IEEE 802.3）-默认。* /
    OFPHTO_NO_HEADER            = 1，/ *无标题，例如 电路开关。* /
    OFPHTO_OXM_EXPERIMENTER = 0xFFFF，/ *使用实验者OXM。* /
};
```

- OFPHTO_ETHERNET是基本的以太网头，由以太网DIX的组合定义
  标准和IEEE 802.3标准。它包含一个源地址，一个目的地址和
  以太网类型。
- OFPHTO_NO_HEADER是一个空标题。例如，当无法使用数据包头时使用
  用于电路开关。
- OFPHTO_OXM_EXPERIMENTER引用OXM实验者值。如果标题类型等于

  OFPHTO_OXM_EXPERIMENTER用于OXM值列表中，标头类型由
  跟随它的第一个OXM实验者值（请参阅7.2.3.12）。这使实验者可以定义任何
  任意标题类型。

数据包标头的大多数部分都可以通过其标头类型（即元组（名称空间，

ns_type）。数据包标头的特定部分可能具有多种标头类型，例如一种
每个名称空间。表9中定义了常见标头的规范标头类型。对于标题
该表未列出，规范的标头类型是名称空间最小的标头类型
数字，并且在单个命名空间中，具有最小ns_type数字的标头类型。

命名空间	ns类型	标头定义	非规范
0	0	以太网头：IEEE Std 802.3。	（1，0x6558）
0			

1个	0x800	IPv4标头：RFC 791。	（2，0x4）
1个	0x8100	VLAN标记：IEEE Std 802.1Q。	（1，0x88a8）
1个	0x86dd	IPv6标头：RFC 2460。	（2，0x29）
1个	0x8847	MPLS填充头：RFC 3032。	（1，0x8848），（2，0x89）
1个	0x88E7	PBB I-TAG：IEEE标准802.1Q。	
2	6	基本TCP标头：RFC 793。	
2	17	UDP标头：RFC 768。	
3	4789	VxLAN标头：RFC 7348。	

表9：常见规范标题类型。

7.2.3流匹配结构

OpenFlow匹配结构用于匹配数据包（请参阅5 .3）或匹配流中的流条目。表（请参阅5. 2）。它由流匹配头和一系列零个或多个流匹配字段组成使用OXM TLV编码。

7.2.3.1流匹配标头

流匹配头由ofp_match结构描述：

```
/ *与流量匹配的字段* /
struct ofp_match {
        uint16_t类型;                   / * OFPMT_ *之一* /
        uint16_t长度;                   / * ofp_match的长度（不包括填充）* /
        /* 其次是：
         *      -恰好（长度-4）（可能为0）个字节包含OXM TLV，然后
         *      -正好（（length + 7）/ 8 * 8-length）（0至7之间）个字节
         *          全零字节
         *总之，根据需要填充ofp_match，以使其整体大小
         * 8的倍数，以保留使用它的结构中的对齐方式。
         * /
        uint8_t oxm_fields [0];         / * 0个或更多OXM匹配字段* /
        uint8_t pad [4];                / *零字节-有关大小调整，请参见上文* /
};
OFP_ASSERT（sizeof（struct ofp_match）== 8）；
```

类型字段设置为OFPMT_OXM，长度字段设置为ofp_match结构的实际长度包括所有匹配字段。OpenFlow匹配的有效负载是一组OXM Flow匹配字段。

```
/ *匹配类型表示匹配结构（构成
 *匹配）。匹配类型放在开头的类型字段中
 *所有匹配结构。“ OpenFlow可扩展匹配”类型对应
 *为以下所述的OXM TLV格式，并且必须由所有OpenFlow支持
 *开关。定义其他匹配类型的扩展程序可能会发布在
 * ONF Wiki。对扩展的支持是可选的。
 * /
of_p_match_type的枚举{
        OFPMT_STANDARD = 0，            / *不推荐使用。* /
        OFPMT_OXM       = 1             / * OpenFlow可扩展匹配* /
};
```

此规范中唯一有效的匹配类型是OFPMT_OXM。OpenFlow 1.1匹配类型

OFPMT_STANDARD已弃用。如果使用替代匹配类型，则匹配字段和有效载荷可能会设置不同，但这超出了本规范的范围。

7.2.3.2流匹配字段结构

使用OpenFlow可扩展匹配（OXM）格式描述流匹配字段，该格式是
紧凑的类型长度值（TLV）格式。每个OXM TLV的长度为5到259（含）字节。OXM
TLV不在任何多字节边界上对齐或填充。OXM TLV的前4个字节为
它的标题，然后是条目的正文。

OXM TLV的标头按网络字节顺序解释为32位字（请参见图9）。

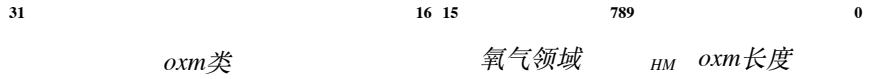


图9：OXM TLV标头布局。

表10中定义了OXM TLV的标头字段

名称		宽度	用法
oxm类型	oxm_class	16	匹配类：成员类或保留类
	oxm_field	7	课程内的比对栏位
	oxm_hasmask	1个	设置OXM是否在有效载荷中包含位掩码
	oxm_length	8	OXM有效载荷的长度

表10：OXM TLV标头字段。

oxm_class是OXM匹配类，其中包含相关的匹配类型，将在一节中进行描述
7.2.3.3。oxm_field是特定于类的值，标识匹配类中的匹配类型之一。
oxm_class和oxm_field（标头的最高有效23位）的组合是
tively oxm_type。oxm_type通常指定协议标头字段，例如以太网类型，
但它也可以引用数据包管道字段，例如数据包到达的交换机端口。

oxm_hasmask定义OXM TLV是否包含位掩码，更多详细信息在第7 .2.3.5节中进行了说明。

oxm_length是一个正整数，以字节为单位描述OXM TLV有效负载的长度，即所有
跟随4字节OXM TLV标头。OXM TLV的长度（包括标题）正好是
4 + oxm_length个字节。

对于给定的oxm_class，oxm_field和oxm_hasmask值，oxm_length是一个常数。它在
只是为了允许软件以最低限度地分析未知类型的OXM TLV而被欺骗。（类似地，
给定oxm_class，oxm_field和oxm_length值，oxm_hasmask是一个常数。）

7.2.3.3 OXM类

匹配类型是使用OXM匹配类构造的。OpenFlow规范区分
OXM匹配类的两种类型，ONF成员类和ONF保留类，由它们的区别
高阶位。高阶位设置为1的类是ONF保留的类，它们用于
OpenFlow规范本身。高阶位设置为零的类是ONF成员类，它们是
由ONF根据需要分配，它们唯一地标识ONF成员并可以使用
由该成员任意决定。对ONF成员类的支持是可选的。

定义了以下OXM类：

```
/ * OXM类ID。
  *高阶位区分保留类和成员类。
  *类0x0000至0x7FFF是由ONF分配的成员类。
  * 0x8000至0xFFFE类是保留类，为标准化保留。
  * /
of_p_oxm_class的枚举{
      OFPXMC_NXM_0              = 0x0000，    / *与NXM的向后兼容性* /
      OFPXMC_NXM_1              = 0x0001，    / *与NXM的向后兼容性* /
      OFPXMC_OPENFLOW_BASIC = 0x8000，    / * OpenFlow的基本类* /
      OFPXMC_PACKET_REGS        = 0x8001，    / *数据包寄存器（管道字段）。* /
      OFPXMC_EXPERIMENTER       = 0xFFFF，    / *实验者课程* /
};
```

类OFPXMC_OPENFLOW_BASIC包含基本的OpenFlow匹配字段集（请参见7 .2.3.7）。
可选类OFPXMC_PACKET_REGS包含数据包寄存器（请参见7 .2.3.10）。可选类
OFPXMC_EXPERIMENTER用于实验者匹配（请参阅7.2 .3.12）。它与其他类不同-
因为它在OXM TLV标头和有效负载中的值之间包括一个实验者标头。
其他ONF保留类保留给以后使用，例如规范的模块化。的
保留前两个ONF成员类OFPXMC_NXM_0和OFPXMC_NXM_1以便向后兼容
符合Nicira可扩展匹配（NXM）规范。

7.2.3.4流量匹配

零长度的OpenFlow匹配项（没有OXM TLV的匹配项）匹配每个数据包。匹配的字段
应该将通配符从OpenFlow匹配中省略。

OXM TLV对由OpenFlow匹配所匹配的数据包施加约束：

- 如果oxm_hasmask为0，则OXM TLV的主体包含该字段的值，称为oxm_value。的
  OXM TLV匹配仅匹配相应字段等于oxm_value的数据包。
- 如果oxm_hasmask为1，则oxm_entry的主体包含该字段的值（oxm_value），
  后跟一个与该值长度相同的位掩码，称为oxm_mask。oxm_mask中的每个1位
  限制OXM TLV仅匹配其中字段的相应位的数据包
  等于oxm_value中的相应位。oxm_mask中的每个0位对
  字段中的对应位。

使用屏蔽时，oxm_mask中的0位在oxm_value中具有对应的1位是错误的
（请参阅7.5.4.5）。

下表总结了一对对应的oxm_mask和oxm_value的约束
使用屏蔽时，这些位将放置在相应的字段位上。省略oxm_mask等效于
提供全为1位的oxm_mask。

当有多个OXM TLV时，必须满足所有约束：数据包字段必须匹配
OpenFlow比赛的所有OXM TLV的一部分。没有OXM TLV的字段是
通配为ANY，省略的OXM TLV有效地完全屏蔽为零。

oxm_mask	oxm值	
	0	1个
0无限制		错误
1个	必须为0	必须为1

表11：OXM掩码和值。

7.2.3.5流匹配字段屏蔽

当oxm_hasmask为1时，OXM TLV包含一个跟随字段值的位掩码，它具有
与字段值大小相同，并且以相同的方式编码。对于不在实验者课程中的字段，
这意味着oxm_length有效地加倍，因此即使oxm_hasmask为
1.对于实验者类中的字段，实验者标题不重复，长度增加
仅对应于蒙版的大小，并且产生的oxm_length取决于实验者
标头使用。

定义掩码，以使给定位位置的0表示相同的“无关”匹配
位在相应字段中，而1表示完全匹配该位。全零位oxm_mask是
相当于完全省略了OXM TLV。全一位oxm_mask等效于指定0
用于oxm_hasmask并省略oxm_mask。

某些oxm_types可能不支持屏蔽的通配符，也就是说，当以下情况时oxm_hasmask必须始终为0
这些字段已指定。例如，用于标识数据包所在的入口端口的字段
收到的内容可能不会被屏蔽。

一些确实支持屏蔽通配符的oxm_types可能仅支持某些oxm_mask模式。对于
例如，某些具有IPv4地址值的字段可能仅限于CIDR掩码（子网掩码）。

这些限制在各个字段的规范中有详细说明。交换机可以接受

规范不允许的oxm_hasmask或oxm_mask值，但前提是必须正确执行该开关-
对oxm_hasmask或oxm_mask值的支持。交换机必须拒绝尝试设置
包含不支持的oxm_hasmask或oxm_mask值的流条目（请参见6.4）。

7.2.3.6流匹配字段先决条件

可以基于存在或值来限制具有给定oxm_type的OXM TLV的存在
其他OXM TLV的前提条件。通常，只能匹配协议的标头字段
如果OpenFlow匹配明确匹配相应的协议。

例如：

- 仅当在oxm_type = OXM OF IPV4 SRC的情况下才允许使用OXM TLV。
  尝试使用oxm_type = OXM_OF_ETH_TYPE，oxm_hasmask = 0和oxm_value = 0x0800。也就是说，匹配-
  仅在将以太网类型显式设置为IPv4时，才允许在IPv4源地址上进行设置。

- 仅当oxm_type = OXM OF TCP SRC时才允许使用OXM TLV

  oxm_type = ETHM类型的ETHM，oxm_hasmask = 0，oxm_value = 0x0800或0x86dd，另一个
  依次使用oxm_type = OXM OF IP PROTO，oxm_hasmask = 0，oxm_value = 6。那是，

  仅当以太网类型为IP并且IP协议时才允许在TCP源端口上进行匹配
  是TCP。

- 仅当oxm_type = OXM OF MPLS LABEL的OXM TLV允许时，才允许
  oxm_type = ETHM ETH类型的条目，oxm_hasmask = 0，oxm_value = 0x8847或0x8848。

- 仅当oxm_type = OXM OF VLAN PCP时才允许使用OXM TLV
  其中oxm_type = OXM OF VLAN VID，oxm_value！= OFPVID NONE。

匹配字段的先决条件是另一个匹配字段类型和与此匹配的匹配字段值
字段取决于。大多数匹配字段都有先决条件，这些限制在以下规范中有所说明：
各个字段（请参阅7.2.3。8）。前提条件是累积性的，匹配字段会继承所有限制
其先决条件（请参见上面的示例），并且所有先决条件链都必须存在于
比赛。

交换机可以实现这些限制的宽松版本。例如，一个开关可能不接受
前提条件。交换机必须拒绝尝试建立违反其限制的流条目的尝试
（请参见7.5.4。5），并且必须处理由于缺少先决条件而导致的不一致匹配（例如
同时匹配TCP源端口和UDP目标端口）。

成员定义的新匹配字段（在成员类中或作为实验者字段）可能会提供备用
已指定匹配字段的先决条件。例如，这可用于重用现有IP
通过替换ETH_TYPE前提条件来匹配备用链路技术（例如PPP）上的字段
根据需要（对于PPP，可以是假设的PPP_PROTOCOL字段）。

具有先决条件限制的OXM TLV必须在OXM TLV之后显示以其先决条件。
在OpenFlow匹配中对OXM TLV的排序没有其他限制（除了数据包之外）
键入匹配字段）。

任何给定的oxm_type最多只能出现在OpenFlow匹配项中，否则开关必须
产生错误（请参阅7.5.4。5）。交换机可以实施此规则的宽松版本，并且可以允许
在某些情况下，oxm_type在OpenFlow匹配中出现多次，但是
然后根据实现定义匹配。

如果流表实现了特定的OXM TLV，则此流表必须接受包含以下内容的有效匹配项：
此OXM TLV的前提条件，即使流表不支持匹配所有可能的值
这些先决条件指定的匹配字段。例如，如果流表与IPv4匹配
源地址，此流表必须接受将Ethertype与IPv4完全匹配，但是此流
表不需要支持将Ethertype匹配到任何其他值。

7.2.3.7流匹配字段

规范使用oxm_class = OFPXMC_OPENFLOW_BASIC定义了一组默认的匹配字段，
可以具有以下值：

```
/ * OXM流匹配字段类型，用于OpenFlow基本类。* /
列举oxm_ofb_match_fields {
    OFPXMT_OFB_IN_PORT          = 0，/ *开关输入端口。* /
    OFPXMT_OFB_IN_PHY_PORT      = 1，/ *切换物理输入端口。* /
    OFPXMT_OFB_METADATA         = 2，/ *在表之间传递元数据。* /
    OFPXMT_OFB_ETH_DST          = 3，/ *以太网目标地址。* /
```

第79页 OpenFlow交换机规格 版本1.5.1

```
    OFPXMT_OFB_ETH_SRC          = 4，/ *以太网源地址。* /
    OFPXMT_OFB_ETH_TYPE         = 5，/ *以太网帧类型。* /
    OFPXMT_OFB_VLAN_VID         = 6，/ * VLAN ID。* /
    OFPXMT_OFB_VLAN_PCP         = 7，/ * VLAN优先级。* /
    OFPXMT_OFB_IP_DSCP          = 8，/ * IP DSCP（ToS字段中为6位）。* /
    OFPXMT_OFB_IP_ECN           = 9，/ * IP ECN（ToS字段中的2位）。* /
    OFPXMT_OFB_IP_PROTO         = 10，/ * IP协议。* /
    OFPXMT_OFB_IPV4_SRC         = 11，/ * IPv4源地址。* /
    OFPXMT_OFB_IPV4_DST         = 12，/ * IPv4目标地址。* /
    OFPXMT_OFB_TCP_SRC          = 13，/ * TCP源端口。* /
    OFPXMT_OFB_TCP_DST          = 14，/ * TCP目标端口。* /
    OFPXMT_OFB_UDP_SRC          = 15，/ * UDP源端口。* /
    OFPXMT_OFB_UDP_DST          = 16，/ * UDP目标端口。* /
    OFPXMT_OFB_SCTP_SRC         = 17，/ * SCTP源端口。* /
    OFPXMT_OFB_SCTP_DST         = 18，/ * SCTP目标端口。* /
    OFPXMT_OFB_ICMPV4_TYPE      = 19，/ * ICMP类型。* /
    OFPXMT_OFB_ICMPV4_CODE      = 20，/ * ICMP代码。* /
    OFPXMT_OFB_ARP_OP           = 21，/ * ARP操作码。* /
    OFPXMT_OFB_ARP_SPA          = 22，/ * ARP源IPv4地址。* /
    OFPXMT_OFB_ARP_TPA          = 23，/ * ARP目标IPv4地址。* /
    OFPXMT_OFB_ARP_SHA          = 24，/ * ARP源硬件地址。* /
    OFPXMT_OFB_ARP_THA          = 25，/ * ARP目标硬件地址。* /
    OFPXMT_OFB_IPV6_SRC         = 26，/ * IPv6源地址。* /
    OFPXMT_OFB_IPV6_DST         = 27，/ * IPv6目标地址。* /
    OFPXMT_OFB_IPV6_FLABEL      = 28，/ * IPv6流标签* /
    OFPXMT_OFB_ICMPV6_TYPE      = 29，/ * ICMPv6类型。* /
    OFPXMT_OFB_ICMPV6_CODE      = 30，/ * ICMPv6代码。* /
    OFPXMT_OFB_IPV6_ND_TARGET = 31，/ * ND的目标地址。* /
    OFPXMT_OFB_IPV6_ND_SLL      = 32，/ * ND的源链路层。* /
    OFPXMT_OFB_IPV6_ND_TLL      = 33，/ * ND的目标链路层。* /
    OFPXMT_OFB_MPLS_LABEL       = 34，/ * MPLS标签。* /
    OFPXMT_OFB_MPLS_TC          = 35，/ * MPLS TC。* /
    OFPXMT_OFP_MPLS_BOS         = 36，/ * MPLS BoS位。* /
    OFPXMT_OFB_PBB_ISID         = 37，/ * PBB I-SID。* /
    OFPXMT_OFB_TUNNEL_ID        = 38，/ *逻辑端口元数据。* /
    OFPXMT_OFB_IPV6_EXTHDR      = 39，/ * IPv6扩展头伪字段* /
    OFPXMT_OFB_PBB_UCA          = 41，/ * PBB UCA标头字段。* /
    OFPXMT_OFB_TCP_FLAGS        = 42，/ * TCP标志。* /
    OFPXMT_OFB_ACTSET_OUTPUT = 43，/ *操作集元数据的输出端口。* /
    OFPXMT_OFB_PACKET_TYPE      = 44，/ *数据包类型值。* /
};
```

支持以太网数据包类型的交换机必须支持表12中列出的必填匹配字段
在其管道中。交换机的至少一个流表中必须支持每个必需的匹配字段：
该流表必须启用与该字段的匹配，并且必须满足该匹配字段的先决条件
表（请参阅7.2.3。6）。不需要在所有流表中都实现必填字段，也不需要
在同一流表中实施。流程图可以支持非必需和实验者
匹配字段。控制器可以查询交换机每个流中支持哪些匹配字段
表（请参见7.3.5.18）。

匹配字段分为两种类型：标头匹配字段（请参见7 .2.3.8）和管道匹配字段（请参见7.2.3.9）。

第80页 OpenFlow交换机规格 版本1.5.1

领域		描述
OXM_OF_IN_PORT	*需要进入*	入口端口。这可能是物理或逻辑端口。
OXM_OF_ACTSET_OUTPUT	*需要在出口*	操作集的出口端口。
OXM_OF_ETH_DST	*需要*	以太网目标地址。可以使用任意位掩码
OXM_OF_ETH_SRC	*需要*	以太网源地址。可以使用任意位掩码
OXM_OF_ETH_TYPE	*需要*	VLAN之后，OpenFlow数据包有效负载的以太网类型 标签。
OXM_OF_IP_PROTO	*需要*	IPv4或IPv6协议号

OXM_OF_IPV4_SRC	*需要*	IPv4源地址。可以使用子网掩码或任意位面具
OXM_OF_IPV4_DST	*需要*	IPv4目标地址。可以使用子网掩码或任意位掩码
OXM_OF_IPV6_SRC	*需要*	IPv6源地址。可以使用子网掩码或任意位面具
OXM_OF_IPV6_DST	*需要*	IPv6目标地址。可以使用子网掩码或任意位掩码
OXM_OF_TCP_SRC	*需要*	TCP源端口
OXM_OF_TCP_DST	*需要*	TCP目标端口
OXM_OF_UDP_SRC	*需要*	UDP源端口
OXM_OF_UDP_DST	*需要*	UDP目标端口

表12：必填的匹配字段。

7.2.3.8标头匹配字段

报头匹配字段是与从分组报头提取的值匹配的匹配字段。大多数标题匹配字段直接映射到数据路径协议定义的数据包头中的特定字段。

所有表头匹配字段具有不同的大小，先决条件和屏蔽功能，如表中所指定13.如果未在字段描述中明确指定，则每种字段类型均指最外层包头中字段的字段。

领域	位	字节数	面具	先决条件		描述
OXM_OF_ETH_DST	48	6	是	包类型=（0,0） 没有	要么	以太网目标MAC地址。
OXM_OF_ETH_SRC	48	6	是	包类型=（0,0） 没有	要么	以太网源MAC地址。
OXM_OF_ETH_TYPE	16	2	没有	包类型=（0,0） 没有	要么	OpenFlow数据包的以太网类型付费在VLAN标签之后加载。
OXM_OF_VLAN_VID	12 + 1	2	是	包类型=（0,0） 没有	要么	来自802.1Q标头的VLAN-ID。CFI位表示存在有效的VLAN-ID，见下文。
OXM_OF_VLAN_PCP	3	1个	没有	VLAN VID！=无		来自802.1Q标头的VLAN-PCP。
OXM_OF_IP_DSCP	6	1个	没有	ETH类型= 0x0800 ETH类型= 0x86dd 资料包类型=（1,0x800） 要么 资料包类型=（1,0x86dd）	要么 要么	区分服务代码点（DSCP）。的一部分 IPv4 ToS字段或IPv6流量类别字段。

表13 –接下页

表13 –前一页总结

领域	位	字节数	面具	先决条件		描述
OXM_OF_IP_ECN	2	1个	没有	ETH类型= 0x0800 ETH类型= 0x86dd 资料包类型=（1,0x800） 要么 资料包类型=（1,0x86dd）	要么 要么	IP标头的ECN位。IPv4的一部分ToS字段或IPv6流量类别字段。
OXM_OF_IP_PROTO	8	1个	没有	ETH类型= 0x0800 ETH类型= 0x86dd 资料包类型=（1,0x800） 要么 资料包类型=（1,0x86dd）	要么 要么	IPv4或IPv6协议号。
OXM_OF_IPV4_SRC	32	4	是	ETH类型= 0x0800 资料包类型=（1,0x800）	要么	IPv4源地址。可以使用子网掩码或任意位掩码
OXM_OF_IPV4_DST	32	4	是	ETH类型= 0x0800 资料包类型=（1,0x800）	要么	IPv4目标地址。可以使用子网掩码或任意位掩码
OXM_OF_TCP_SRC	16	2	没有	IP协议= 6		TCP源端口
OXM_OF_TCP_DST	16	2	没有	IP协议= 6		TCP目标端口
OXM_OF_TCP_FLAGS	12	2	是	IP协议= 6		TCP标志
OXM_OF_UDP_SRC	16	2	没有	IP协议= 17		UDP源端口
OXM_OF_UDP_DST	16	2	没有	IP协议= 17		UDP目标端口
OXM_OF_SCTP_SRC	16	2	没有	IP协议= 132		SCTP源端口
OXM_OF_SCTP_DST	16	2	没有	IP协议= 132		SCTP目标端口
OXM_OF_ICMPV4_TYPE	8	1个	没有	IP PROTO = 1		ICMP类型
OXM_OF_ICMPV4_CODE	8	1个	没有	IP PROTO = 1		ICMP代码
OXM_OF_ARP_OP	16	2	没有	ETH类型= 0x0806		ARP操作码
OXM_OF_ARP_SPA	32	4	是	ETH类型= 0x0806		ARP有效负载中的源IPv4地址。可以使用子网掩码或任意位掩码
OXM_OF_ARP_TPA	32	4	是	ETH类型= 0x0806		ARP有效负载中的目标IPv4地址。可以使用子网掩码或任意位掩码
OXM_OF_ARP_SHA	48	6	是	ETH类型= 0x0806		ARP有效负载中的源以太网地址。
OXM_OF_ARP_THA	48	6	是	ETH类型= 0x0806		ARP有效负载中的目标以太网地址。
OXM_OF_IPV6_SRC	128	16	是	ETH类型= 0x86dd 资料包类型=（1,0x86dd）	要么	IPv6源地址。可以使用子网掩码或任意位掩码
OXM_OF_IPV6_DST	128	16	是	ETH类型= 0x86dd 资料包类型=（1,0x86dd）	要么	IPv6目标地址。可以使用子网掩码或任意位掩码
OXM_OF_IPV6_FLABEL	20	4	是	ETH类型= 0x86dd 资料包类型=（1,0x86dd）	要么	IPv6流标签。
OXM_OF_ICMPV6_TYPE	8	1个	没有	IP协议= 58		ICMPv6类型
OXM_OF_ICMPV6_CODE	8	1个	没有	IP协议= 58		ICMPv6代码

OXM_OF_IPV6_ND_TARGET	128	16	没有	**ICMPV6类型= 136**	要么	邻居消息Dis-中的目标地址
OXM_OF_IPV6_ND_SLL	48	6	没有	ICMPV6类型= 135		链接中的源链接层地址选项IPv6邻居发现消息。
OXM_OF_IPV6_ND_TLL	48	6	没有	ICMPV6类型= 136		目标中的目标链接层地址选项IPv6邻居发现消息。
OXM_OF_MPLS_LABEL	20	4	没有	ETH类型= 0x8847 ETH类型= 0x8848	要么	第一个MPLS填充标头中的LABEL。
OXM_OF_MPLS_TC	3	1个	没有	ETH类型= 0x8847 ETH类型= 0x8848	要么	第一个MPLS填充头中的TC。
OXM_OF_MPLS_BOS	1个	1个	没有	ETH类型= 0x8847 ETH类型= 0x8848	要么	第一个中的BoS位（堆栈底部位）MPLS填充头。
OXM_OF_PBB_ISID	24	3	是	ETH类型= 0x88E7		第一个PBB服务实例中的I-SID标签。
OXM_OF_IPV6_EXTHDR	9	2	是	ETH类型= 0x86dd 资料包类型=（1,0x86dd）	要么	IPv6扩展头伪字段。
OXM_OF_PBB_UCA	1个	1个	没有	ETH类型= 0x88E7		第一个PBB服务中的UCA字段是-立场标签。

表13：标题匹配字段的详细信息。

省略OFPXMT_OFB_VLAN_VID字段指定流条目应与数据包匹配，无论
它们是否包含相应的标签。下面为VLAN标签定义了特殊值
允许匹配具有任何标签的数据包，而与标签的值无关，并支持匹配
没有VLAN标签的数据包。为OFPXMT_OFB_VLAN_VID定义的特殊值为：

```
/ * VLAN ID是12位，因此我们可以使用全部16位来表示
  * 特殊情况。
  * /
of_plan_id的枚举{
    OFPVID_PRESENT = 0x1000，/ *表示已设置VLAN ID的位* /
    OFPVID_NONE      = 0x0000，/ *未设置VLAN ID。* /
};
```

当通配符为OFPXMT_OFB_VLAN_VID字段时，必须拒绝OFPXMT_OFB_VLAN_PCP字段
（不存在）或OFPXMT_OFB_VLAN_VID的值设置为OFPVID_NONE时。表14总结
特定VLAN标签匹配的通配符位和字段值的组合。

OXM领域	**oxm值**	**氧气面罩**	**匹配包**
缺席	--	--	*带有*和*不带有*VLAN标签的数据包
当下	OFPVID_NONE	缺席	只有*没有* VLAN标签的数据包
当下	OFPVID_PRESENT	OFPVID_PRESENT	仅*带有* VLAN标记的数据包，无论其是否值
当下	*价值*| OFPVID_PRESENT	缺席	仅具有VLAN标签和VID的数据包相等*值*

表14：VLAN标签的匹配组合。

字段OXM_OF_IPV6_EXTHDR是一个伪报头匹配字段，指示存在各种
数据包标头中的IPv6扩展标头。IPv6扩展标头位组合在一起
字段OXM_OF_IPV6_EXTHDR，这些位可以具有以下值：

```
/ * IPv6扩展头伪字段的位定义。* /
枚举ofp_ipv6exthdr_flags {
    OFPIEH_NONEXT = 1 << 0，    / *遇到“没有下一个标头”。* /
    OFPIEH_ESP      = 1 << 1，    / *存在加密的有效载荷头。* /
    OFPIEH_AUTH     = 1 << 2，    / *存在身份验证标头。* /
    OFPIEH_DEST     = 1 << 3，    / *存在1或2个dest标头。* /
    OFPIEH_FRAG     = 1 << 4，    / *存在片段头。* /
    OFPIEH_ROUTER = 1 << 5，    / *存在路由器标头。* /
    OFPIEH_HOP      = 1 << 6，    / *存在逐跳标题。* /
    OFPIEH_UNREP = 1 << 7，     / *遇到意外的重复。* /
    OFPIEH_UNSEQ = 1 << 8，     / *遇到意外的排序。* /
};
```

- 如果逐跳IPv6扩展标头作为第一个扩展，则OFPIEH_HOP设置为1
  数据包中的标头。
- 如果存在路由器IPv6扩展头，则将OFPIEH_ROUTER设置为1。
- 如果存在碎片IPv6扩展头，则OFPIEH_FRAG设置为1。
- 如果存在一个或多个“目标”选项IPv6扩展头，则将OFPIEH_DEST设置为1。它
  通常在一个IPv6数据包中包含其中一个或两个（请参阅RFC 2460）。

- 如果存在身份验证IPv6扩展头，则OFPIEH_AUTH设置为1。
- 如果存在加密的安全有效载荷IPv6扩展标头，则OFPIEH_ESP设置为1。
- 如果不存在下一报头IPv6扩展标头，则OFPIEH_NONEXT设置为1。
- 如果IPv6扩展头的优先顺序不是首选（但不是优先顺序），则OFPIEH_UNSEQ设置为1由RFC 2460）。
- 如果一个给定的IPv6扩展标头中有多个意外出现，则OFPIEH_UNREP设置为1遇到。（两个目标选项标头可能是预期的，并且不会导致此位设置。）

字段OXM_OF_TCP_FLAGS与TCP标头中的标志位匹配。最少考虑位0有效位，具体包含：

- 位0-5：RFC 793中定义的原始TCP标志。
- 位6-8：RFC 3168和3540定义的其他TCP标志。
- 位9-11：保留位尚未由RFC标准化。
- 位12–15：不是OXM字段的一部分，强制为零。（TCP头使用相应的位用于其他目的，因此它们永远不会被标准化为标志。）

7.2.3.9管道匹配字段

管道匹配字段是匹配字段，该匹配字段匹配附加到数据包以进行管道处理的值而不与数据包头相关联。

如表中所指定，所有管道匹配字段具有不同的大小，先决条件和屏蔽功能[15]

领域	位	字节数	面具	先决条件	描述
OXM_OF_IN_PORT	32	4	没有	没有	入口端口。数值表示从1开始的端口。这可能是物理端口或交换机定义的逻辑端口。
OXM_OF_IN_PHY_PORT	32	4	没有	在港礼物	物理端口。在ofp_packet_in条消息中，-位于日志上的数据包时位于物理端口ical端口。
OXM_OF_METADATA	64	8	是	没有	表元数据。用于在之间传递信息表。
OXM_OF_TUNNEL_ID	64	8	是	没有	与逻辑端口关联的元数据。
OXM_OF_ACTSET_OUTPUT	32	4	没有	没有	操作集元数据的输出端口或OFPP未设定。
OXM_OF_PACKET_TYPE	16 + 16	4	没有	没有	数据包类型-最外层的规范报头类型标头。

表15：管道匹配字段的详细信息。

入口端口字段OXM_OF_IN_PORT用于匹配数据包所在的OpenFlow端口在OpenFlow数据路径中收到。它可以是物理端口，逻辑端口，OFPP_LOCAL保留端口或OFPP_CONTROLLER保留端口（请参阅4.1）。入口端口必须是有效的标准

OpenFlow端口或OFPP_CONTROLLER保留端口，对于标准端口，端口配置必须允许它接收数据包（已清除OFPPC_PORT_DOWN和OFPPC_NO_RECV配置位）。

*分组入*消息中使用物理端口字段OXM_OF_IN_PHY_PORT 标识物理端口在逻辑端口下面（请参见7.4.1）。交换机必须支持*Packet-in*消息中的物理端口字段仅当物理端口是该交换机的OpenFlow端口且其值与入口不同时港口。此外，某些交换机可以选择允许匹配流条目中的物理端口字段，在这种情况下，当物理端口与入口端口相同或不是OpenFlow端口时，此字段的值必须是入口端口。物理端口必须是有效的OpenFlow端口，并且

必须包含在端口描述请求中（请参阅7.3.5。6）。物理端口不需要
任何特定的配置，例如它可能已关闭或接收数据包可能被禁用。

当直接在物理端口上接收到数据包并且未由逻辑端口处理数据包时，

OXM_OF_IN_PORT和OXM_OF_IN_PHY_PORT具有相同的值，该phys的OpenFlow port_no
ical端口（请参阅4。1）。当通过物理端口在逻辑端口上接收到数据包时，该物理端口不是
一个OpenFlow端口OXM_OF_IN_PORT和OXM_OF_IN_PHY_PORT具有相同的值，即OpenFlow

此逻辑端口的port_no。当通过物理端口在逻辑端口上接收到数据包时
这是一个OpenFlow端口，OXM_OF_IN_PORT是逻辑端口的port_no，而OXM_OF_IN_PHY_PORT是
物理端口的port_no。例如，考虑在以下接口定义的隧道接口上收到的数据包：
具有两个物理端口成员的链路聚合组（LAG）。如果隧道接口是逻辑的
端口绑定到OpenFlow，则OXM_OF_IN_PORT是隧道的port_no，而OXM_OF_IN_PHY_PORT是
LAG的物理port_no成员，在其上配置了隧道。

元数据字段OXM_OF_METADATA用于在多个查询之间传递信息
表。该值可以任意屏蔽。

隧道ID字段OXM_OF_TUNNEL_ID携带与
逻辑端口。在支持隧道ID的逻辑端口上接收到数据包时，“隧道ID”字段
与数据包相关联的内容由封装元数据设置，并且可以与流条目进行匹配。
如果逻辑端口不提供此类数据，或者如果在物理端口上接收到数据包，则其值
是零。在支持tunnel-id字段的逻辑端口上发送数据包时，它将使用该值
该字段用于内部封装处理。例如，该字段可以由流条目设置
使用*设定场*动作。

如果逻辑端口支持“隧道ID”字段，则在输入上提供它，在输出上使用它，或
两者都应在相应的端口描述属性中报告支持（请参见7 .2.1.2）。

隧道ID字段中可选封装元数据的映射由逻辑定义
端口实现，它取决于逻辑端口的类型，并且是实现特定的。我们
建议对于通过GRE隧道接收的包含（32位）密钥的数据包，将密钥存储在
低32位和高位清零。我们建议对于MPLS逻辑端口，将较低的
20位代表MPLS标签。我们建议对于VxLAN逻辑端口，低24位
代表VNI。

动作集输出字段OXM_OF_ACTSET_OUTPUT可用于匹配嵌入的输出动作
在数据包的动作集中，在大多数情况下，可根据它们的输出来匹配数据包
港口。需要所有用作出口表的流表来支持此字段，而用作入口的流表
表可以选择支持该字段，具体取决于交换机的功能。如果操作集包含
输出操作，没有组操作，此字段等于该操作的输出端口。否则，如果

操作集不包含输出操作或包含组操作，该字段等于其初始值
值OFPP_UNSET。

7.2.3.10数据包寄存器

数据包寄存器字段OXM_OF_PKT_REG（N）用于存储临时值和信息
通过流水线处理与数据包一起。每个数据包寄存器为64位宽且可屏蔽。在
在大多数情况下，数据包寄存器无法在表中匹配，即它们通常不能在表中使用。
流条目匹配结构。它们可以与*set-field*和*copy-field*操作一起使用（请参见7.2.6.7和
7.2.6.8）。

交换机可以选择实现任意数量的数据包寄存器，但不超过OXM格式的限制
（即128）。每个数据包寄存器由其OXM类型字段标识。如果交换机支持数据包寄存器
并且使用从零开始的连续编号，仅支持的分组寄存器具有最大的编号
字段号需要在表功能属性中列出，否则所有受支持的数据包寄存器
必须单独列出（请参见7.3.5.18。2）。不支持数据包寄存器字段的交换机必须
表功能属性中不包括任何数据包寄存器条目。

7.2.3.11数据包类型匹配字段

该*数据包类型匹配字段*是OXM型OXM_OF_PACKET_TYPE管道领域，其值标识
数据路径中的数据包类型。常见数据包类型的值在表16中定义。对于其他
包类型，该值必须是包最外头的*规范头类型*（请参见7 .2.2）。
数据包类型匹配字段必须显示为第一个OXM TLV，并且仅匹配该数据包的数据包
具体类型。大多数数据包类型值也是其他匹配字段的前提条件。如果是数据包类型
匹配字段不存在，数据包类型必须是以太网。

表16列出了规范定义的数据包类型。可以定义新的数据包类型
使用其他规范的标头类型或实验器机制（请参见7.2.2）。大多数开关将支持
仅支持单个数据包类型，目前不支持多种数据包类型
（但可能会在将来的版本中添加）。

命名空间ns	类型匹配	描述	报文输入和输出垫
0	0	以太网数据包（默认类型）。	以太网头和以太网付费加载。
1个	0x800	IPv4数据包（不包含标头）面前）。	IPv4标头和IPv4有效负载。
1个	0x86dd	IPv6数据包（无标题）面前）。	IPv6标头和IPv6有效负载。
0	1个	没有数据包（例如电路）开关）。	空的
0	0xFFFF实验者已定义。		实验者定义。

表16：数据包类型。

支持特定数据包类型的OpenFlow交换机会通告相应的数据包类型
*表格要素属性*中的“匹配字段”（请参见7.3.5.18.2）。*Flow-mod*消息必须包含适当的
数据包类型匹配匹配中的字段，除非数据包类型为以太网（请参阅7.3.4.2）。*包入*
消息中，匹配中必须包含“数据包类型匹配”字段，除非数据包类型为以太网
（请参见7.4.1）。*封包输出*消息必须在匹配中包括“封包类型匹配”字段，除非
数据包类型为以太网（请参见7.3.6）。

数据包类型（0，0）是以太网数据包的数据包类型。对于此数据包类型，匹配可能
包括，例如，以太网目标地址，以太网源地址，以太类型和其他
根据以太类型匹配字段。对于此数据包类型，数据包输入和输出数据字段
包括以太网报头和以太网帧的以太网有效载荷，但不包括
前导，帧的开始，以太网FCS / CRC和帧间间隙。此数据包类型将
通常用于以太网交换机。

数据包类型（1，0x800）是IPv4数据包的数据包类型。对于此数据包类型，匹配可能
包括例如IPv4目标和源地址，IP协议以及其他基于
在协议上。对于此数据包类型，数据包输入和输出数据字段包括IPv4标头
以及IPv4数据报的IPv4有效负载。例如，此数据包类型可以在IP路由器中使用。

数据包类型（1，0x86dd）是IPv6数据包的数据包类型。对于此数据包类型，匹配可能
包括例如IPv6目标和源地址，IP协议和其他基于
在协议上。对于此数据包类型，数据包输入和输出数据字段包括IPv6标头
以及IPv6数据报的IPv6有效负载。例如，此数据包类型可以在IP路由器中使用。

数据包类型（0，1）是与数据包不匹配的表的数据包类型。对于此数据包类型，
匹配仅包括管道匹配字段，伪标题字段，并且不包含任何实际标题
领域。例如，该分组类型可以在电路交换机中使用。

数据包类型（0，0xFFFF）是实验数据包类型的数据包类型。实际数据包类型
由其后的第一个OXM实验者值定义（请参见7.2.3.12）。这使实验者能够
定义任意的数据包类型。

7.2.3.12实验者流程匹配字段

对实验者特定的流量匹配字段的支持是可选的。实验者特定的流量匹配字段
可以使用oxm_class = OFPXMC_EXPERIMENTER进行定义。

```
/ * OXM实验者匹配字段的标题。
  *实验者类不应使用OXM_HEADER（）宏进行定义
  *由于此额外的标题而产生的字段。* /
struct ofp_oxm_experimenter_header {
        uint32_t oxm_header;                    / * oxm_class = OFPXMC_EXPERIMENTER * /
        uint32_t实验者；                        / *实验者ID。* /
};
OFP_ASSERT（sizeof（p_oxm_experimenter_header的结构）== 8）；
```

OXM标头中的oxm_class字段必须设置为OFPXMC_EXPERIMENTER。

OXM标头中的oxm_field字段是实验者类型，由实验者管理
（请参阅7.2.8）。实验者标识符和实验者类型共同标识实验者
比赛场。

OXM标头中的oxm_length字段计算oxm_header之后的所有字节。对于实验
指导者OXM，oxm_length尤其包括实验者字段本身。

实验者字段被编码在OXM TLV主体的前四个字节中。它包含
实验者标识符，其格式与典型实验者结构中的形式相同（请参见7.2.8）。

OXM TLV主体的其余部分是由实验人员定义的，不需要填充或对齐。

7.2.4流统计结构

OpenFlow统计信息由流统计信息头和一系列零个或多个流统计信息字段组成。

7.2.4.1流统计头

流统计头由ofp_stats结构描述：

```
/ *流统计结构-统计字段列表。* /
struct ofp_stats {
        uint16_t保留；                          / *保留以供将来使用，当前为零。* /
        uint16_t长度；                          / * ofp_stats的长度（不包括填充）* /
        /* 其次是：
         *      -恰好（长度-4）（可能为0）个字节包含OXS TLV，然后
         *      -正好（（length + 7）/ 8 * 8-length）（0至7之间）个字节
         *          全零字节
         *总而言之，ofp_stats根据需要进行填充，以使其整体大小
         * 8的倍数，以保留使用它的结构中的对齐方式。
         * /
        uint8_t oxs_fields [0];                 / * 0个或更多OXS统计字段* /
        uint8_t pad [4];                        / *零字节-有关大小调整，请参见上文* /
};
OFP_ASSERT（sizeof（struct ofp_stats）== 8）；
```

长度字段设置为ofp_stats结构的实际长度，包括所有stat字段。有效载荷
OpenFlow统计信息中的一组是OXS Flow统计信息字段。

7.2.4.2 Flow Stat字段结构

流统计字段使用OpenFlow可扩展统计（OXS）格式描述，该格式非常紧凑
类型长度值（TLV）格式。每个OXS TLV的长度为5到259（含）字节。OXS TLV不是
在任何多字节边界上对齐或填充。OXS TLV的前4个字节是其标头，
其次是条目的正文。

OXS TLV的标头按网络字节顺序解释为32位字（请参见图10）。

表17中定义了OXS TLV的标头字段

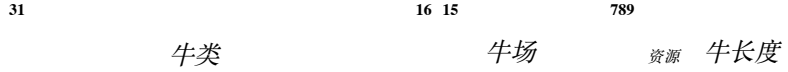


图10：OXS TLV标头布局。

名称	宽度	用法
oxs_class	16	统计类：成员类或保留类

牛型	oxs_field	7	类中的统计字段
	oxs_reserved	1个	保留以备将来使用
	oxs_length	8	OXS有效载荷的长度

表17：OXS TLV标头字段。

oxs_class是OXS状态类，其中包含相关的状态类型，在第7.0.2.3节中进行了描述。

oxs_field是特定于类的值，标识stat类中的一种统计类型。组合-
oxs_class和oxs_field（标头的最重要的23位）的位置合在一起为oxs_type。
oxs_type通常指定一个统计字段，例如流字节数。

oxs_reserved保留供将来使用，并且必须设置为零。

oxs_length是一个正整数，以字节为单位描述OXS TLV有效负载的长度。长度
包括标头在内的OXS TLV正好是4 + oxs_length个字节。

对于给定的oxs_class，oxs_field和oxs_reserved值，oxs_length是一个常数。它在
只是为了允许软件以最低限度地分析未知类型的OXS TLV而被欺骗。（类似地，对于给定
oxs_class，oxs_field和oxs_length值，oxs_reserved是一个常数。）

7.2.4.3 OXS类别

统计类型使用OXS统计类来构造。这些遵循与OXM相同的约定
匹配类（请参见7.2.3.3）。定义了以下OXS类：

```
/ * OXS类别ID。
  *高阶位区分保留类和成员类。
  *类0x0000至0x7FFF是由ONF分配的成员类。
  * 0x8000至0xFFFE类是保留类，为标准化保留。
  * /
of_p_oxs_class的枚举{
      OFPXSC_OPENFLOW_BASIC = 0x8002，      / * OpenFlow的基本统计信息类* /
      OFPXSC_EXPERIMENTER   = 0xFFFF，      / *实验者课程* /
};
```

类OFPXSC_OPENFLOW_BASIC包含OpenFlow状态字段的基本集合（请参见7.2.4.4）。的
可选类OFPXSC_EXPERIMENTER用于实验者状态字段（请参见7 .2.4.5）。

7.2.4.4流统计字段

规范使用oxs_class = OFPXSC_OPENFLOW_BASIC定义了一组默认的stat字段
可以具有以下值：

```
/ * OpenFlow基本类的OXS流统计字段类型。* /
列举oxs_ofb_stat_fields {
      OFPXST_OFB_DURATION          = 0，/ *时间流条目已激活。* /
      OFPXST_OFB_IDLE_TIME         = 1，/ *时间流条目已空闲。* /
      OFPXST_OFB_FLOW_COUNT        = 3，/ *聚合流条目的数量。* /
      OFPXST_OFB_PACKET_COUNT      = 4，/ *流条目中的包数。* /
      OFPXST_OFB_BYTE_COUNT        = 5，/ *流条目中的字节数。* /
};
```

交换机的每个流表必须支持表18中列出的必需stat字段。可选统计
仅当流表支持字段时，它们才必须包含在ofp_stats结构中；如果
流表不支持它们，它们不能包含在ofp_stats结构中。

所有stat字段的大小都不同，如表18所示。

领域	位	流输入	骨料	描述
OXS_OF_DURATION	2 * 32	需要	未定义	时间流条目仍然有效。
OXS_OF_IDLE_TIME	2 * 32	可选的	未定义	时间流条目已空闲。
OXS_OF_FLOW_COUNT	32	未定义	需要	聚合流条目数。
OXS_OF_PACKET_COUNT	64	可选的	可选的	流表项匹配的报文数。
OXS_OF_BYTE_COUNT	64	可选的	可选的	流条目匹配的字节数。

表18：“统计信息”字段详细信息。

OXS_OF_DURATION字段指示流条目已安装在交换机中的经过时间。
有效负载中的第一个32位值是duration_sec，第二个32位值是duration_nsec。的
总持续时间（以纳秒为单位）可以计算为*持续时间sec* ×10 9 + *持续时间nsec*。实作
需要提供第二精度；如果可能，鼓励更高的精度。该字段是
没有为OFPMP_AGGREGATE答复定义的，在这种情况下不应该使用。

OXS_OF_IDLE_TIME字段指示流条目未匹配的经过时间
任何数据包，并且是交换机将用于与流条目进行比较的值
idle_timeout（请参阅7 .3.4.2）。此时间值的编码与OXS_OF_DURATION字段相同。
未为OFPMP_AGGREGATE答复定义此字段，在这种情况下不应使用该字段。

OXS_OF_FLOW_COUNT字段仅在OFPMP_AGGREGATE答复中使用，并指示有多少流量
条目与请求匹配（请参阅7 .3.5.4）。未为流条目定义此字段（即，不在
OFPMP_AGGREGATE答复），在这种情况下不应该使用。

OXS_OF_PACKET_COUNT字段是流条目匹配的数据包数量。在OFPMP_AGGREGATE
答复，它是与请求相匹配的所有流条目所匹配的包数。

OXS_OF_BYTE_COUNT字段是流条目匹配的数据包中的字节数。在
OFPMP_AGGREGATE答复，它是与重新匹配的所有流条目匹配的字节数
寻求。

第90页

7.2.4.5实验者流量统计字段

对实验者特定的流量统计信息字段的支持是可选的。实验者特定的流量统计信息字段可能
使用oxs_class = OFPXSC_EXPERIMENTER定义。

```
/ * OXS实验者状态字段的标题。* /
struct ofp_oxs_experimenter_header {
    uint32_t oxs_header;                / * oxs_class = OFPXSC_EXPERIMENTER * /
    uint32_t实验者;                     / *实验者ID。* /
};
OFP_ASSERT（sizeof（p_oxs_experimenter_header的结构）== 8）;
```

OXS标头中的oxs_class字段必须设置为OFPXMC_EXPERIMENTER。

OXS标头中的oxs_field字段是实验者类型，由实验者管理（请参见
7.2.8）。实验者标识符和实验者类型共同标识实验者状态
领域。

实验者字段被编码在OXS TLV主体的前四个字节中。它包含
实验者标识符，其格式与典型实验者结构中的形式相同（请参见7.2.8）。

OXS TLV主体的其余部分是由实验人员定义的，不需要填充或对齐。

7.2.5流程指令结构

当流与条目匹配时，执行与流表条目关联的流指令。的
当前定义的指令列表为：

```
of_p_instruction_type的枚举{
    OFPIT_GOTO_TABLE = 1            / *在查询中设置下一个表
                                        管道* /
    OFPIT_WRITE_METADATA = 2        / *设置元数据字段，以供以后在
                                        管道* /
    OFPIT_WRITE_ACTIONS = 3，       / *将操作写入数据路径操作
                                        设置* /
    OFPIT_APPLY_ACTIONS = 4         / *立即应用操作* /
    OFPIT_CLEAR_ACTIONS = 5         / *清除数据路径中的所有操作
                                        动作集* /
    OFPIT_DEPRECATED = 6            / *不推荐使用（适用计量器）* /
    OFPIT_STAT_TRIGGER = 7          / *统计触发器* /

    OFPIT_EXPERIMENTER = 0xFFFF / *实验者指令* /
};
```

指令集在第5节中进行了描述。 5 .流表可以支持指令类型的子集。
指令定义包含指令类型，长度和任何相关数据：


```
/ *所有指令共有的指令头。长度包括
  *标头和用于使指令64位对齐的任何填充。
  *注意：指令的长度*必须*始终是8的倍数。* /
struct ofp_instruction_header {
        uint16_t类型;                   / * OFPIT_ *之一。* /
        uint16_t len;                   / *此结构的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_instruction_header的结构）== 4）；
```

OFPIT_GOTO_TABLE指令使用以下结构和字段：

```
/ * OFPIT_GOTO_TABLE的指令结构* /
struct ofp_instruction_goto_table {
        uint16_t类型;                   / * OFPIT_GOTO_TABLE * /
        uint16_t len;                   / *长度为8。* /
        uint8_t table_id;               / *在查找管道中设置下一个表* /
        uint8_t pad [3];                / *填充到64位。* /
};
OFP_ASSERT（sizeof（p_instruction_goto_table的结构）== 8）；
```

table_id指示数据包处理管道中的下一个表。

OFPIT_WRITE_METADATA指令使用以下结构和字段：

```
/ * OFPIT_WRITE_METADATA的指令结构* /
struct ofp_instruction_write_metadata {
        uint16_t类型;                   / * OFPIT_WRITE_METADATA * /
        uint16_t len;                   / *长度为24。* /
        uint8_t pad [4];                / *对齐64位* /
        uint64_t元数据；                / *要写入的元数据值* /
        uint64_t metadata_mask;         / *元数据写入位掩码* /
};
OFP_ASSERT（sizeof（p_instruction_write_metadata的结构）== 24）；
```

可以使用以下命令中的元数据和metadata_mask写入下一个表查询的元数据
以便在match字段上设置特定位。如果未指定此指令，则传递元数据，
不变。

OFPIT_WRITE_ACTIONS，OFPIT_APPLY_ACTIONS和OFPIT_CLEAR_ACTIONS指令使用
以下结构和字段：

```
/ * OFPIT_WRITE / APPLY / CLEAR_ACTIONS的指令结构* /
struct ofp_instruction_actions {
        uint16_t类型;                   / * OFPIT _ * _ ACTIONS之一* /
        uint16_t len;                   / *长度填充为64位。* /
        uint8_t pad [4];                / *对齐64位* /
        struct ofp_action_header actions [0]; / * 0个或更多与之相关的动作
                                                OFPIT_WRITE_ACTIONS和
                                                OFPIT_APPLY_ACTIONS * /
};
OFP_ASSERT（sizeof（struct ofp_instruction_actions）== 8）；
```


对于Apply-Actions指令，将action字段视为一个列表，并将这些操作应用于
数据包*按顺序排列*（请参阅5.7）。

对于Write-Actions指令，将action字段视为一个集合，并将这些动作合并
放入当前操作集中（请参阅5.6）。如果一组动作包含两个或两个相同类型的动作

相同类型的set_field操作。开关可以返回错误（请参见7.5.4.3），或者开关可以
按顺序合并操作集中的操作集，然后覆盖操作集中的后续操作
相同类型的早期动作（请参见5.6）。

对于“清除动作”指令，该结构不包含任何动作。

OFPIT_STAT_TRIGGER指令使用以下结构和字段：

```
/ * OFPIT_STAT_TRIGGER的指令结构* /
struct ofp_instruction_stat_trigger {
    uint16_t类型;                    / * OFPIT_STAT_TRIGGER * /
    uint16_t len;                    / *长度填充为64位。* /
    uint32_t标志;                    / * OFPSTF_ *标志的位图。* /
    p_stats阈值的结构；/ *阈值列表。大小可变。* /
};
OFP_ASSERT（sizeof（p_instruction_stat_trigger的结构）== 16）；
```

标志字段是一个位图，用于定义统计触发器的行为。它可能包括
以下标志的组合：

```
of_p_stat_trigger_flags的枚举{
    OFPSTF_PERIODIC       = 1 << 0，/ *为所有阈值的倍数触发。* /
    OFPSTF_ONLY_FIRST = 1 << 1，/ *仅在第一个达到阈值时触发。* /
};
```

设置OFPSTF_PERIODIC标志后，触发器不仅会应用于阈值中的值，
而且还可以是这些值的所有倍数。例如，它允许每100个数据包有一个触发器
流的生命周期。

设置OFPSTF_ONLY_FIRST时，仅考虑超过的第一个阈值，其他
阈值被忽略。它允许控制器仅接收单个触发事件来触发多个阈值。
老年人。

阈值是统计字段阈值的列表。当流的统计字段值之一
如果条目超过其阈值，则会将*统计触发*事件发送到控制器。流条目未被修改
此操作。例如，如果阈值列表包括字节数10000，则在流条目时
字节数从9000到10500，必须生成一条消息。

当必须将*统计触发*事件发送到控制器时，交换机必须添加带有
流统计多部分消息中出现OFPFSR_STAT_TRIGGER原因（请参阅7.3.5.3），并且该消息必须
被发送到控制器。流统计多部分消息可能包含多个不相关的*统计触发器*
事件以减少控制通道上的开销。开关可能会丢弃或延迟任何*状态触发*事件
由于控制信道上的拥塞和备份，或者为了优化控制信道的使用。一种
强烈建议控制器使用此指令，以使事件发生率可通过以下方式管理
开关。

OFPIT_EXPERIMENTER指令使用以下结构和字段：


```
/ *实验指令的指令结构* /
struct ofp_instruction_experimenter_header {
    uint16_t类型;                    / * OFPIT_EXPERIMENTER。* /
    uint16_t len;                    / *长度填充为64位。* /
    uint32_t实验者；                 / *实验者ID。* /
    / *实验者定义的任意附加数据。* /
};
OFP_ASSERT（sizeof（p_instruction_experimenter_header的结构）== 8）；
```

实验者字段是实验者ID，其格式与典型的实验-
imenter结构（参见7.2。 8）。指令的其余部分由实验者定义，整个
指令需要填充为64位。

7.2.6动作结构

许多动作可以与流条目，组或分组相关联。当前定义
动作类型为：

```
of_p_action_type的枚举{
    OFPAT_OUTPUT          = 0，/ *输出到交换机端口。* /
    OFPAT_COPY_TTL_OUT = 11，/ *复制TTL“向外”-从最下到最外
                                到最外面* /
    OFPAT_COPY_TTL_IN = 12，/ *将TTL“向内”复制-从最外面到
                               倒数第二个* /
    OFPAT_SET_MPLS_TTL = 15，/ * MPLS TTL * /
```

```
    OFPAT_DEC_MPLS_TTL = 16，/ *递减MPLS TTL * /

    OFPAT_PUSH_VLAN     = 17，/ *推送新的VLAN标记* /
    OFPAT_POP_VLAN      = 18，/ *弹出外部VLAN标记* /
    OFPAT_PUSH_MPLS     = 19，/ *推送新的MPLS标签* /
    OFPAT_POP_MPLS      = 20，/ *弹出外部MPLS标签* /
    OFPAT_SET_QUEUE     = 21，/ *在输出到端口时设置队列ID * /
    OFPAT_GROUP         = 22，/ *应用组。* /
    OFPAT_SET_NW_TTL    = 23，/ * IP TTL。* /
    OFPAT_DEC_NW_TTL    = 24，/ *减少IP TTL。* /
    OFPAT_SET_FIELD     = 25，/ *使用OXM TLV格式设置标题字段。* /
    OFPAT_PUSH_PBB      = 26，/ *推送新的PBB服务标签（I-TAG）* /
    OFPAT_POP_PBB       = 27，/ *弹出外部PBB服务标签（I-TAG）* /
    OFPAT_COPY_FIELD    = 28，/ *在标头和寄存器之间复制值。* /
    OFPAT_METER         = 29，/ *应用仪表（速率限制器）* /
    OFPAT_EXPERIMENTER = 0xffff
};
```

输出，组和设置队列操作在第5.8节中进行了描述，标签推送/弹出操作在其中进行了描述
在表2中描述了Set-Field操作，在表13中从它们的OXM类型描述了Set-Field操作。

在流进入指令（请参阅7.2.5），组存储桶（请参见7.3.4.3）和包输出中使用操作
消息（请参阅7.3.6）。使用修改数据包的操作假定相应的一组
标头存在于数据包中。动作对没有相应动作的数据包的影响
标头集未定义（取决于开关和字段）。例如，使用一个*流行VLAN*上的作用
没有VLAN标签的数据包，或在没有IP或MPLS标头的数据包上使用*设置的TTL*，

未定义。交换机可以选择拒绝行为与之不一致的流条目
流条目的匹配结构和先前动作（请参见6.4）。强烈鼓励管制员
避免生成可能产生不一致动作的表条目组合。

动作定义包含动作类型，长度和任何相关数据：

```
/ *所有动作共有的动作标头。长度包括
  *标头和用于使操作64位对齐的任何填充。
  *注意：动作的长度*必须*始终是8的倍数。* /
struct ofp_action_header {
    uint16_t类型;                    / * OFPAT_ *之一。* /
    uint16_t len;                    / *此结构的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_action_header的结构）== 4）；
```

7.2.6.1输出动作结构

的*輸出*动作使用以下结构和字段：

```
/ * OFPAT_OUTPUT的操作结构，该结构将数据包从“端口”发送出去。
  *当“端口”为OFPP_CONTROLLER时，“ max_len”表示最大
  *要发送的字节数。“ max_len”为零表示没有字节
  *数据包应发送。OFPCML_NO_BUFFER的“ max_len”表示
  *数据包未缓冲，完整的数据包将发送到
  *控制器。* /
struct ofp_action_output {
    uint16_t类型;                    / * OFPAT_OUTPUT。* /
    uint16_t len;                    / *长度为16。* /
    uint32_t端口;                    / *输出端口。* /
    uint16_t max_len;                / *要发送到控制器的最大长度。* /
    uint8_t pad [6];                 / *填充到64位。* /
};
OFP_ASSERT（sizeof（p_action_output的结构）== 16）；
```

端口指定数据包应通过其发送的端口。max_len表示
当端口为OFPP_CONTROLLER时，应从数据包发送的最大数据量。如果

max_len为零，交换机必须发送该数据包的零字节。OFPCML_NO_BUFFER的max_len
表示应该发送完整的数据包，并且不应该对其进行缓冲。

```
of_p_controller_max_len的枚举{
        OFPCML_MAX          = 0xffe5，/ *可以使用的最大max_len值
                                        请求特定的字节长度。* /
        OFPCML_NO_BUFFER = 0xffff / *指示不应进行任何缓冲
                                        应用，整个包将被
                                        发送到控制器。* /
};
```


7.2.6.2小组行动结构

一*组*动作采用以下结构和字段：

```
/ * OFPAT_GROUP的操作结构。* /
struct ofp_action_group {
        uint16_t类型;                           / * OFPAT_GROUP。* /
        uint16_t len;                           / *长度为8。* /
        uint32_t group_id;                      / *组标识符。* /
};
OFP_ASSERT（sizeof（p_action_group的结构）== 8）；
```

group_id指示用于处理此数据包的组。适用的水桶组取决于
组类型。

7.2.6.3设置队列操作结构

所述*的Set-队列*动作设置将要用于映射的流条目，以配置已经一个队列ID
不论IP DSCP和VLAN PCP位如何，端口上的队列。数据包不应更改为
Set-Queue操作的结果。如果交换机需要设置DSCP / PCP位用于内部处理，则
发送数据包之前，应恢复原始值。

交换机可能仅支持绑定到特定PCP / DSCP位的队列。在这种情况下，它不能
将任意流条目映射到特定队列，因此不支持Set-Queue操作。的
用户仍然可以使用这些队列，并通过设置相关字段（IP DSCP，
VLAN PCP），但是从DSCP或PCP到队列的映射是特定于实现的。

“ *设置队列*”操作使用以下结构和字段：

```
/ * OFPAT_SET_QUEUE的操作结构。* /
struct ofp_action_set_queue {
        uint16_t类型;                   / * OFPAT_SET_QUEUE。* /
        uint16_t len;                   / * Len是8。* /
        uint32_t queue_id;              / *数据包的队列ID。* /
};
OFP_ASSERT（sizeof（p_action_set_queue的结构）== 8）；
```

7.2.6.4仪表动作结构

该*仪表*操作使用下面的结构和字段：

```
/ * OFPAT_METER的操作结构* /
p_action_meter的结构{
        uint16_t类型;                   / * OFPAT_METER * /
        uint16_t len;                   / *长度为8。* /
        uint32_t meter_id;              / *仪表实例。* /
};
OFP_ASSERT（sizeof（p_action_meter的结构）== 8）；
```

meter_id指示要在数据包上应用的仪表。

7.2.6.5 TTL动作结构

一*组MPLS TTL*操作使用下面的结构和字段：

```
/ * OFPAT_SET_MPLS_TTL的操作结构。* /
struct ofp_action_mpls_ttl {
        uint16_t类型;                           / * OFPAT_SET_MPLS_TTL。* /
        uint16_t len;                           / *长度为8。* /
        uint8_t mpls_ttl;                       / * MPLS TTL * /
        uint8_t pad [3];
};
OFP_ASSERT（sizeof（struct ofp_action_mpls_ttl）== 8）;
```

mpls_ttl字段是要设置的MPLS TTL。

一个*递减MPLS TTL*动作没有参数，仅由一个通用ofp_action_generic。
该动作使MPLS TTL减小。

```
/ * OFPAT_COPY_TTL_OUT，OFPAT_COPY_TTL_IN，
  * OFPAT_DEC_MPLS_TTL，OFPAT_DEC_NW_TTL，OFPAT_POP_VLAN和OFPAT_POP_PBB。* /
struct ofp_action_generic {
        uint16_t类型;                           / * OFPAT_ *之一。* /
        uint16_t len;                           / *长度为8。* /
        uint8_t pad [4];                        / *填充到64位。* /
};
OFP_ASSERT（sizeof（struct ofp_action_generic）== 8）;
```

一个*集网络TTL*操作使用下面的结构和字段：

```
/ * OFPAT_SET_NW_TTL的操作结构。* /
struct ofp_action_nw_ttl {
        uint16_t类型;                           / * OFPAT_SET_NW_TTL。* /
        uint16_t len;                           / *长度为8。* /
        uint8_t nw_ttl;                         / * IP TTL * /
        uint8_t pad [3];
};
OFP_ASSERT（sizeof（p_action_nw_ttl的结构）== 8）;
```

nw_ttl字段是在IP标头中设置的TTL地址。

一个*递减网络TTL*动作没有参数，只由一个通用

ofp_action_generic。如果存在IP报头中的TTL，则此操作将使其递减。

一个*副本TTL向外*动作没有参数，仅由一个通用ofp_action_generic。
该操作将TTL从具有TTL的倒数第二个报头复制到具有TTL的最外部报头
TTL。

一个*副本TTL向内*行动没有参数，仅由一个通用ofp_action_generic。
该操作将TTL从带有TTL的最外面的标头复制到带有TTL的倒数第二个标头
TTL。

7.2.6.6推和弹出动作结构

该*推VLAN头*，*推MPLS头*和*推PBB头*操作使用以下结构
和字段：

```
/ * OFPAT_PUSH_VLAN / MPLS / PBB的操作结构。* /
struct ofp_action_push {
        uint16_t类型;                           / * OFPAT_PUSH_VLAN / MPLS / PBB。* /
        uint16_t len;                           / *长度为8。* /
        uint16_t ethertype;                     / *以太网类型* /
        uint8_t pad [2];
};
OFP_ASSERT（sizeof（struct ofp_action_push）== 8）;
```

ethertype字段指示新标签的以太类型。在推送新的VLAN标签时使用它，
新的MPLS标头或PBB服务标头以标识新标头的类型，并且必须有效
对于该标签类型。

Push标记操作*始终*在该标记的最外面有效位置插入新的标记头，例如
由管理该标签的规范定义。

所述*推送VLAN报头*动作将一个新的VLAN标记到分组（C-TAG或S-TAG）。什么时候
推送了一个新的VLAN标签，它应该是以太网之后紧接插入的最外面的标签
标头和其他标签之前。ethertype字段必须为0x8100或0x88a8。

该*推MPLS头*动作将一个新的MPLS垫片包头到包。当新的MPLS
标签被推送到IP数据包上，它是最外面的MPLS标签，立即作为填充头插入
在任何MPLS标签之前或在IP标头之前，以先到者为准。以太网类型字段
必须为0x8847或0x8848。

该*推PBB头*动作逻辑推新PBB业务实例头添加到包（I-TAG
TCI），并将数据包的原始以太网地址复制到客户地址（C-DA和
C-SA）。PBB服务实例标头应立即插入最外面的标签
在以太网头之后和其他标签之前。ethertype字段必须为0x88E7。顾客
I-TAG的地址位于封装数据包的原始以太网地址的位置，
因此，此操作可以看作是添加主干MAC-in-MAC标头和I-SID
数据包前面的字段。该*推PBB头*动作不增加骨干VLAN头
（B-TAG）到数据包，可以通过在*推送* PBB报头之后的*推送VLAN报头*操作添加
操作。完成此操作后，可以使用常规的设置字段操作来修改外部以太网
地址（B-DA和B-SA）。

一个*流行VLAN头*动作没有参数，仅由一个通用ofp_action_generic。
该操作将从数据包中弹出最外面的VLAN标记。

一个*流行PBB头*动作没有参数，仅由一个通用ofp_action_generic。的
从逻辑上讲，该操作会从数据包（I-TAG TCI）中弹出最外面的PBB服务实例标头，
将客户地址（C-DA和C-SA）复制到数据包的以太网地址中。这个动作
可以视为从主干网的前端删除了主干MAC-in-MAC标头和I-SID字段
包。的*流行PBB头*动作不会从去除骨干VLAN报头（B-TAG）
数据包，则应在执行此操作之前通过*Pop VLAN标头*操作将其删除。

甲*流行MPLS报头*动作使用以下结构和字段：

```
/ * OFPAT_POP_MPLS的操作结构。* /
struct ofp_action_pop_mpls {
        uint16_t类型;                          / * OFPAT_POP_MPLS。* /
        uint16_t len;                          / *长度为8。* /
        uint16_t ethertype;                    / *以太网类型* /
        uint8_t pad [2];
};
OFP_ASSERT（sizeof（struct ofp_action_pop_mpls）== 8）；
```

ethertype表示MPLS有效负载的以太类型。以太类型用作以太类型
生成的数据包，无论是否已将已删除的“堆栈底部（BoS）”位设置为
MPLS垫片。建议使用此操作的流条目同时匹配MPLS标签和
MPLS BoS字段，以避免将错误的以太类型应用于MPLS有效负载。

当BoS为MPLS规范时，不允许将任意以太类型设置为MPLS有效负载
不等于1，并且控制器有责任遵守此要求，并且只能设置0x8847
或0x8848作为这些MPLS有效负载的以太类型。交换机可以选择强制执行此MPLS
要求：在这种情况下，交换机应拒绝匹配通配符BoS的任何流条目以及任何流
在流行的MPLS标头操作中，BoS匹配0且醚类型错误的条目，在两种情况下
应该返回*匹配不一致*错误消息（请参见7.5.4.3）。

7.2.6.7设置字段动作结构

“ *设置字段*”操作使用以下结构和字段：

```
/ * OFPAT_SET_FIELD的动作结构。* /
struct ofp_action_set_field {
        uint16_t类型;                          / * OFPAT_SET_FIELD。* /
        uint16_t len;                          / *长度填充为64位。* /
        /* 其次是：
         *      -恰好（4 + oxm_length）个字节包含一个OXM TLV，然后
         *      -完全（（8 + oxm_length）+ 7）/ 8 * 8-（8 + oxm_length）
         *          （0到7之间）全零字节的字节
         * /
        uint8_t字段[4];                        / * OXM TLV-使编译器满意* /
};
OFP_ASSERT（sizeof（p_action_set_field的结构）== 8）；
```

该字段包含使用单个OXM TLV结构描述的标头字段或管道字段（请参见
7.2.3）。*Set-Field*操作由oxm_type（OXM TLV的类型）定义，并修改相应的-
数据包的标头字段或管道字段的值为oxm_value，即

OXM TLV，如果设置了oxm_hasmask，则可以选择使用位掩码oxm_mask。

*设置字段*操作的类型是有效的OXM标头类型之一。[illegible]OXM类型列表
在7.2.3.7节和表13中进行了说明。所有标头匹配字段 field ac-
除了OXM_OF_IPV6_EXTHDR。管道字段OXM_OF_[illegible]DATA，OXM_OF_TUNNEL_ID和
所有OXM_OF_PKT_REG（N）在设置字段操作，其他管道字段OXM_OF_IN_PORT和

OXM_OF_IN_PHY_PORT在设置字段操作中无效。该*集合场*行动可以包括有经验的
指导者OXM字段，由“实验者OXM”定义“实验者设置字段”动作的有效性
自行输入。

oxm_value的值指示要在数据包头或管道字段中写入的值。什么时候
没有使用屏蔽，OXM TLV的有效载荷中的值必须有效，尤其是
未由OXM类型定义的有效负载必须设置为零。使用屏蔽时，

与oxm_mask中的0位对应的oxm_value必须设置为0，其余位必须形成
该字段的有效修改。在OXM_OF_VLAN_VID集字段操作中，OFPVID_PRESENT位必须
在oxm_value和oxm_mask中为1位。

oxm_hasmask的值用于指示该操作包含该值的位掩码。这个
该功能是可选的，并且仅在交换机指示其可用性时才可用（请参阅7.3.5.18.2），否则
必须生成一个错误（请参阅7 .5.4.3）。如果oxm_hasmask为0，则完整的标头字段设置为oxm_value。
如果oxm_hasmask为1，则oxm_entry的主体包含该字段的值（oxm_value），后跟一个
与值长度相同的位掩码，称为oxm_mask。对于oxm_mask中的每个1位，对应的
标头字段的位设置为oxm_value中的相应位。对于oxm_mask中的每个0位，
标头字段的相应位被保留。

使用屏蔽时，oxm_mask中的0位在oxm_value中具有对应的1位是错误的
（请参见7.5.4.3）。

当“ *设置字段*”操作更改标头字段时，它将覆盖
OXM类型指定的数据包。OXM字段是指在
标头，除非字段类型明确指定，否则通常使用set-field操作
适用于最外面的报头（例如，“设置VLAN ID”设置字段操作始终设置
最外面的VLAN标签）。当标头字段被*set-field*操作覆盖时，开关必须
在将数据包发送到输出端口或输入数据包之前，重新计算所有受影响的校验和字段
信息。

流程中必须包含与要设置的字段相对应的OXM前提条件（参见7 .2.3.6）
输入，否则必须生成一个错误（请参阅7.5.4.3）。每个前提条件都必须包含在其中
流条目的匹配项，或者必须通过在设置字段操作之前发生的操作来满足（对于
例如推送标签）。使用*set-field*操作假定相应的标头字段存在
在数据包中，*设置字段*操作对没有相应报头的数据包的影响
字段未定义（取决于开关和字段）。特别是在不设置数据包的情况下设置VID时
VLAN标头，交换机可能会或可能不会自动添加新的VLAN标签以及控制器
必须明确使用*Push VLAN标头*操作以与所有交换机实现兼容。
强烈建议控制器避免生成可能产生的表项组合
设定场动作不一致。

7.2.6.8复制字段操作结构

“ *复制字段*”操作使用以下结构和字段：

```
/ * OFPAT_COPY_FIELD的动作结构。* /
struct ofp_action_copy_field {
        uint16_t类型;                           / * OFPAT_COPY_FIELD。* /
        uint16_t len;                           / *长度填充为64位。* /
```



```
        uint16_t n_bits;                        / *要复制的位数。* /
```

```
    uint16_t src_offset;                    /*源中的起始位偏移量。*/
    uint16_t dst_offset;                    /*目标中的起始位偏移量。*/
    uint8_t pad [2];                        /*对齐32位。*/
    /* 其次是：
     *     -包含oxm_id的8、12或16个字节，然后
     *     -足够全零的字节（0或4个字节）来构成一个整体
     *       8字节长度的倍数*/
    uint32_t                oxm_ids [0];        /*源和目标OXM标头*/
};
OFP_ASSERT（sizeof（p_action_copy_field的结构）== 12）；
```

将src_oxm_id [src_offset：src_offset + n_bits]复制到dst_oxm_id [dst_offset：dst_offset + n_bits]，
其中a [b：c]表示“ a”中编号为“ b”至“ c”的位（不包括“ c”位）。位编号
最低有效位从0开始，下一个最高有效位从1开始，依此类推。

n_bits字段包含要从src_oxm_id复制到dst_oxm_id的位数。

src_offset字段指示src_oxm_id中应读取位的位偏移量。

dst_offset字段指示dst_oxm_id中应写入位的位偏移量。

oxm_ids字段是包含两种OXM类型的列表。此列表的第一个元素src_oxm_id，
标识从中复制值的标头或管道字段。此列表的第二个元素，

dst_oxm_id，标识将值复制到的标头或管道字段。那的要素
列表是用于非实验者OXM字段的32位OXM标头或用于实验者的64位OXM标头
OXM字段，那些OXM字段不包含任何有效负载。oxm_hasmask的值必须为零
并且不包含任何值或掩码。如果支持“ *复制字段*”操作，则它必须支持src_oxm_id
或dst_oxm_id是交换机支持的数据包寄存器之一。支持其他管道
字段和标头字段是可选的，对于这些字段，此操作必须遵守与“ *设置字段* ”相同的规则
采取行动，尤其是在先决条件方面。

开关必须拒绝src_offset + n_bits大于宽度的“ *复制字段*”操作
src_oxm_id或dst_offset + n_bits的值大于带有*错误集参数*的dst_oxm_id的宽度
*ment*错误消息（请参阅7.5.4。3）。当src_oxm_id与以下内容重叠时，此操作必须正确执行
dst_oxm_id，也就是说，它的行为就像将src_oxm_id复制到临时缓冲区中一样，然后
子缓冲区复制到dst_oxm_id，如果这不可能，则交换机必须拒绝“ *复制字段*”操作
用*坏集类型*的错误消息。

由于竞争条件，未定义操作集中的*复制字段*操作的影响，因此
不建议在操作集中执行（请参见5.6）。

7.2.6.9实验者动作结构

一个*实验者*操作使用下面的结构和字段：

```
/ * OFPAT_EXPERIMENTER的操作标头。
  *身体的其余部分由实验者定义。* /
struct ofp_action_experimenter_header {
    uint16_t类型;                           / * OFPAT_EXPERIMENTER。* /
    uint16_t len;                           /*长度是8的倍数。* /
```



```
    uint32_t实验者；                        /*实验者ID。*/
};
OFP_ASSERT（sizeof（p_action_experimenter_header的结构）== 8）；
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。指令的其余部分由实验者定义，整个动作
需要填充到64位。

7.2.7控制器状态结构

OpenFlow交换机使用控制器状态结构来通知控制器有关状态
每个控制器维护的控制通道的数量。此信息主要用于
多控制器部署。

控制器状态结构始终代表交换机的状态视图，如下所示：

```
/ * OFPMP_CONTROLLER_STATUS回复正文和异步正文
  * OFPT_CONTROLLER_STATUS消息* /
struct ofp_controller_status {
    uint16_t长度；                          /*此项的长度。* /
```

```
    uint16_t short_id;              / *标识控制器的ID号。* /
    uint32_t角色;                   / *控制器的角色。OFPCR_ROLE_ *之一。* /
    uint8_t原因;                    / * OFPCSR_ *原因码之一。* /
    uint8_t channel_status;         / *控制通道的状态。
                                      * OFPCT_STATUS_ *之一。* /
    uint8_t pad [6];                / *对齐64位。* /

     / *控制器状态属性列表。连接URI属性是
          需要; 其他属性是可选的。* /
     p_controller_status_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_controller_status的结构）== 16）；
```

short_id是该控制器在OFPT_ROLE_REQUEST中设置的控制器标识符
信息。如果交换机未请求标识符，则应该将short_id设置为OFPCID_UNDEFINED。

OFPCID_UNDEFINED为0。

角色是控制器当前具有的角色（请参见6.3.7）。

原因表明交换机为何发送控制器状态。它是以下之一：

```
/ *为什么要报告控制器状态？* /
of_p_controller_status_reason的枚举{
    OFPCSR_REQUEST              = 0，/ *控制器请求的状态。* /
    OFPCSR_CHANNEL_STATUS       = 1，/ *通道的运行状态已更改。* /
    OFPCSR_ROLE                 = 2，/ *控制器角色已更改。* /
    OFPCSR_CONTROLLER_ADDED     = 3，/ *添加了新控制器。* /
    OFPCSR_CONTROLLER_REMOVED = 4，/ *控制器已从配置中删除。* /
    OFPCSR_SHORT_ID             = 5，/ *控制器ID已更改。* /
    OFPCSR_EXPERIMENTER         = 6，/ *实验者数据已更改。* /
};
```


channel_status是交换机针对控制信道的操作状态的视图。
指示的控制器。它的价值来自于：

```
/ *控制通道状态。* /
of_p_control_channel_status的枚举{
    OFPCT_STATUS_UP     = 0，     / *控制通道可操作。* /
    OFPCT_STATUS_DOWN   = 1       / *控制通道无法运行。* /
};
```

属性字段是控制器状态属性的列表，描述了其他状态信息。
当前定义的控制器状态属性类型的列表为：

```
/ *控制器状态属性类型。
  * /
of_p_controller_status_prop_type的枚举{
    OFPCSPT_URI                 = 0，          / *连接URI属性。* /
    OFPCSPT_EXPERIMENTER        = 0xFFFF，/ *实验者属性。* /
};
```

OFPCSPT_URI属性是必需的，并使用以下结构和字段：

```
/ *连接URI控制器状态属性* /
struct ofp_controller_status_prop_uri {
    uint16_t            类型;       / * OFPCSPT_URI。* /
    uint16_t            长度; / *此属性的长度（以字节为单位）。* /

    /* 其次是：
      *     -恰好（长度-4）个字节包含连接URI，然后
      *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
      *        全零字节的字节* /
    烧焦                uri [0];
};
OFP_ASSERT（sizeof（p_controller_status_prop_uri的结构== 4）；
```

uri是控制器的连接URI，格式为*protocol：name-or-address：port*。对于
例如：tls：192.168.34.23：6653。有关更多详细信息，请参见6 .3.1。

可选的OFPCSPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者控制器状态属性* /
p_controller_status_prop_experimenter的结构{
    uint16_t            类型;       / * OFPCSPT_EXPERIMENTER。* /
    uint16_t            长度; / *此属性的长度（以字节为单位）。* /
    uint32_t            实验者 / *采用相同的实验者ID
```

```
                                                形式如结构
                                                ofp_experimenter_header。* /
    uint32_t                exp_type;           / *实验者定义。* /
    /* 其次是：
     *     -确切地（长度-12个）字节包含实验者数据，然后
     *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_controller_status_prop_experimenter的结构）== 12）；
```


实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.2.8实验人员结构

实验者扩展提供了OpenFlow交换机提供其他功能的标准方法
在OpenFlow消息类型空间中。这是用于未来OpenFlow的功能的过渡区域
修订。许多OpenFlow对象类型提 er扩展，例如请 参阅
7.5.5），OXM匹配（请参阅7.2. ，： 见7.2.6.9）， 3.5.8），
仪表（见7.3.4.5）和误差（见7.5

典型的实验者结构如下所示：

```
/ *典型的实验者结构。* /
struct ofp_experimenter_structure {
    uint32_t实验者；                / *实验者ID：
                                      *-MSB 0：低位字节是IEEE OUI。
                                      *-MSB！= 0：由ONF定义。* /
    uint32_t exp_type;              / *实验者定义。* /
    uint8_t Experimenter_data [0];
};
OFP_ASSERT（sizeof（p_experimenter_structure的结构）== 8）；
```

由于某些对象中包含的实际Experimenter扩展可能与该结构不完全匹配
受各种约束。

实验者字段是一个32位值，用于唯一标识实验者，该实体定义了
实验者扩展。如果最高有效字节为零，则接下来的三个字节是实验者的
IEEE OUI。如果最高有效字节不为零，则为开放网络分配的值
基础。如果实验者没有（或希望使用）他们的OUI，则应联系公开课
Networking Foundation获得唯一的实验者ID。

exp_type字段是某些实验者扩展中提供的32位值，它定义了实验者
类型。此实验者类型由实验者管理，可以具有任何值和功能
实验者的愿望。

Experimenter_data字段表示Experimenter扩展主体的其余部分，它是
未由标准OpenFlow处理解释，并由相应的ex-
蠕动者。该主体的长度必须适合OpenFlow对象和OpenFlow对象
长度必须正确设置以容纳此数据。一些OpenFlow对象需要整个
实验者扩展将填充为64位。

实际的Experimenter扩展超出了本规范的范围，它们始终是可选的，
并且不需要任何开关即可支持任何Experimenter扩展。如果开关不明白
实验者扩展程序，它必须拒绝整个消息并发送适当的错误消息。
使用的错误消息取决于OpenFlow对象（请参见7.5 .4）。如果开关拒绝特定
实验者扩展，控制器不应再使用该特定实验者扩展
后续消息。

7.3控制器到交换机的消息

7.3.1握手

控制器使用OFPT_FEATURES_REQUEST消息来标识开关并读取
它的基本功能。建立会话后（请参见6.3.3），控制器应发送一个

OFPT_FEATURES_REQUEST条消息。该消息不包含OpenFlow标头之外的正文。
交换机必须以OFPT_FEATURES_REPLY消息作为响应：

```
/ *切换功能。* /
struct ofp_switch_features {
        struct ofp_header标头；
        uint64_t datapath_id;               / *数据路径唯一ID。低48位用于
                                              MAC地址，而高16位是
                                              实施者定义的。* /

        uint32_t n_buffers;                 / *一次缓冲的最大数据包数。* /

        uint8_t n_tables;                   / *数据路径支持的表数。* /
        uint8_t assistant_id;               / *标识辅助连接* /
        uint8_t pad [2];                    / *对齐64位。* /

        /* 特征。* /
        uint32_t功能；/ *支持“ ofp_capabilities”的位图。* /
        uint32_t保留；
};
OFP_ASSERT（sizeof（p_switch_features的结构）== 32）；
```

datapath_id字段唯一标识一个数据路径。低48位用于交换机
MAC地址，而高16位取决于实现者。使用高16位的示例
为VLAN ID，以区分单个物理交换机上的多个虚拟交换机实例。这个领域
控制器应将其视为不透明的位字符串。

n_buffers字段指定交换机发送时可以缓冲的最大数据包数
使用*包入*消息将*数据包*发送到控制器（请参见6.1.2）。

n_tables字段描述了交换机支持的表数，每个表可以具有一个
支持的匹配字段，操作和条目数不同。当控制器和开关
首先进行通信，控制器将从功能部件中找出交换机支持的表数量。
回复。如果希望了解查询表的大小，类型和顺序，则控制器
发送OFPMP_TABLE_FEATURES多部分请求（请参见7. 3.5.18）。交换机必须将这些表返回
数据包遍历表的顺序。

assistant_id字段标识从交换机到控制器（主
connection将该字段设置为零，辅助连接将此字段设置为非零值（请参见
6.3.8）。

功能字段是使用以下标志的组合的位图：


```
/ *数据路径支持的功能。* /
ofp_capabilities枚举{
        OFPC_FLOW_STATS         = 1 << 0，/ *流量统计。* /
        OFPC_TABLE_STATS        = 1 << 1，/ *表统计信息。* /
        OFPC_PORT_STATS         = 1 << 2，/ *端口统计信息。* /
        OFPC_GROUP_STATS        = 1 << 3，/ *组统计。* /
        OFPC_IP_REASM           = 1 << 5，/ *可以重组IP片段。* /
        OFPC_QUEUE_STATS        = 1 << 6，/ *队列统计信息。* /
        OFPC_PORT_BLOCKED       = 1 << 8，/ *交换机将阻塞循环端口。* /
        OFPC_BUNDLES            = 1 << 9，/ *开关支持包。* /
        OFPC_FLOW_MONITORING = 1 << 10，/ *开关支持流量监控。* /
};
```

OFPC_PORT_BLOCKED位指示OpenFlow外部的交换协议，例如802.1D
生成树将检测拓扑循环并阻止端口以防止数据包循环。如果未设置此位，
在大多数情况下，控制器应实施一种机制以防止数据包循环。

7.3.2交换机配置

控制器能够使用OFPT_SET_CONFIG在交换机中设置和查询配置参数
和OFPT_GET_CONFIG_REQUEST消息。交换机响应配置请求
带有OFPT_GET_CONFIG_REPLY消息；它不会回复设置配置的请求。

除了OpenFlow标头之外，OFPT_GET_CONFIG_REQUEST没有其他正文。OFPT_SET_CONFIG
和OFPT_GET_CONFIG_REPLY使用以下内容：

```
/ *开关配置。* /
struct ofp_switch_config {
        struct ofp_header标头；
        uint16_t标志；                          / * OFPC_ *标志的位图。* /
        uint16_t miss_send_len;                 / *该数据路径的最大数据包字节数
                                                    应该发送给控制器。看到
                                                    ofp_controller_max_len获取有效值。
                                                    * /
};
OFP_ASSERT（sizeof（p_switch_config的结构）== 12）；
```

标志字段是使用以下配置标志的组合的位图：

```
of_p_config_flags枚举{
        / *处理IP片段。* /
        OFPC_FRAG_NORMAL  = 0，             / *对片段没有特殊处理。* /
        OFPC_FRAG_DROP    = 1 << 0，/ *丢弃片段。* /
        OFPC_FRAG_REASM   = 1 << 1，/ *重新组装（仅在设置了OFPC_IP_REASM的情况下）。* /
        OFPC_FRAG_MASK    = 3，             / *处理碎片的标志的位掩码。* /
};
```

OFPC_FRAG_ *标志指示IP片段是应该正常处理，丢弃还是重新分配-
流血。片段的“正常”处理意味着应该尝试传递片段
通过OpenFlow表。如果不存在任何字段（例如，TCP / UDP端口不适合），则

数据包不应与设置了该字段的任何条目匹配。对“正常”的支持是强制性的，支持
“掉落”和“理性”是可选的。

miss_send_len字段定义由主机发送到控制器的每个数据包的字节数。
例如，不使用对OFPP_CONTROLLER逻辑端口的输出操作时的OpenFlow管道
如果启用了此消息原因，则发送具有无效TTL的数据包。如果此字段等于0，则交换机必须
在ofp_packet_in消息中发送数据包的零字节。如果该值设置为OFPCML_NO_BUFFER
完整的数据包必须包含在消息中，并且不应进行缓冲。

7.3.3流程图配置

流表的编号从0开始，可以使用任何数字，直到OFPTT_MAX。OFPTT_ALL是保留的
值。

```
/ *表编号。表格最多可以使用OFPT_MAX个数字。* /
ofp_table枚举{
        / *最后可用的表号。* /
        OFPTT_MAX           = 0xfe，

        / *假表。* /
        OFPTT_ALL           = 0xff      / *用于表配置的通配符表，
                                            流量统计信息和流量删除。* /
};
```

使用OFPT_FLOW_MOD请求在流表中修改流条目（请参见7.3.4.2）。表
可以使用OFPMP_TABLE_STATS多部分请求查询统计信息（请参阅7 .3.5.16）。控制器
可以使用OFPT_TABLE_MOD请求在流表中配置动态状态（请参见7.3.4.2）。的
可以使用OFPMP_TABLE_FEATURES多部分请求来修改流表的静态功能（请参阅
7.3.5.18）。

7.3.4修改状态消息

7.3.4.1修改流表消息

控制器可以使用OFPT_TABLE_MOD请求在流表中配置动态状态。的
OFPT_TABLE_MOD使用以下结构和字段：

```
/ *配置/修改流表的行为* /
struct ofp_table_mod {
        struct ofp_header标头;
        uint8_t table_id;                / *表的ID，OFPTT_ALL表示所有表* /
        uint8_t pad [3];                 / *填充到32位* /
        uint32_t配置;                    / * OFPTC_ *标志的位图* /

        / *表Mod属性列表* /
        p_table_mod_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_table_mod）== 16）;
```


table_id选择应将配置更改应用到的表。如果table_id
如果是OFPTT_ALL，则该配置将应用于交换机中的所有表。

config字段是用于配置表行为的位图。定义了以下值
对于那个领域：

```
/ *标志来配置表。* /
of_p_table_config的枚举{
        OFPTC_DEPRECATED_MASK    = 3，              / *不建议使用的位* /
        OFPTC_EVICTION           = 1 << 2，/ *授权表逐出流程。* /
        OFPTC_VACANCY_EVENTS     = 1 << 3，/ *启用空缺事件。* /
};
```

标志OFPTC_EVICTION控制该流表中的流条目逐出（请参见6. 5）。如果设置了此标志，
交换机可以从该流表中逐出流条目。如果未设置此标志，则交换机无法退出流量
该表中的条目。流条目逐出是可选的，因此开关可能不支持
设置此标志。

标志OFPTC_VACANCY_EVENTS控制该表中的空缺事件（请参见7.4。5）。如果设置了此标志，
交换机必须为该表生成空缺事件。如果未设置此标志，则开关不得生成
这些事件。可以使用OFPTMPT_VACANCY属性指定空缺事件的参数（请参见
如下），如果此属性未包含在table-mod消息中，请先前设置参数的值
必须使用。

properties字段是表mod属性的列表，描述了表config-的动态参数。
排尿。

当前定义的表mod属性类型为：

```
/ *表Mod属性类型。
  * /
of_p_table_mod_prop_type的枚举{
        OFPTMPT_EVICTION                 = 0x2，        / *逐出属性。* /
        OFPTMPT_VACANCY                  = 0x3，        / *空置属性。* /
        OFPTMPT_EXPERIMENTER             = 0xFFFF，/ *实验者属性。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有表Mod属性的通用标头* /
struct ofp_table_mod_prop_header {
        uint16_t                 类型;        / * OFPTMPT_ *之一。* /
        uint16_t                 长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_table_mod_prop_header的结构）== 4）;
```

通常无法使用OFP_TABLE_MOD请求来修改OFPTMPT_EVICTION属性，因为
驱逐机制是由开关定义的，并且此属性通常仅在OFPMP_TABLE_DESC中使用
多部分答复，以告知控制器所使用的驱逐机制。它使用以下内容
结构和字段：

```
/ *逐出表mod属性。通常在OFPMP_TABLE_DESC答复中使用。* /
struct ofp_table_mod_prop_eviction {
        uint16_t            类型;        / * OFPTMPT_EVICTION。* /
        uint16_t            长度; / *此属性的长度（以字节为单位）。* /
        uint32_t            标志;        / * OFPTMPEF_ *标志的位图* /
};
OFP_ASSERT（sizeof（p_table_mod_prop_eviction的结构）== 8）;
```

标志字段是一个位图，它定义了交换机正在实施的逐出类型。它可能包括
以下标志的组合：

```
/ *逐出标志。* /
of_p_table_mod_prop_eviction_flag的枚举{
        OFPTMPEF_OTHER          = 1 << 0,       / *使用其他因素。* /
        OFPTMPEF_IMPORTANCE     = 1 << 1,       / *使用流条目重要性。* /
        OFPTMPEF_LIFETIME       = 1 << 2,       / *使用流条目生存期。* /
};
```

如果设置了标志OFPTMPEF_IMPORTANCE，则收回过程将使用流条目重要性字段
（请参阅7.3.4。2）执行驱逐。如果标志OFPTMPEF_IMPORTANCE是唯一设置的标志，则逐出将
严格按照重要性顺序执行，即重要性较低的流进入始终
在进入流程之前具有更高的重要性。如果要插入的新流条目不如
所有现有的流条目，都不会发生逐出。驱逐机制是开关定义的，因此它
无法预测哪个重要性相同的流条目将被逐出。

如果设置了标志OFPTMPEF_LIFETIME，则收回过程将使用流条目的剩余生存期，即
最短的到期时间（请参阅6. 5）执行驱逐。如果标志OFPTMPEF_LIFETIME是唯一的
如果设置了标志，则将按照剩余寿命和永久流条目的顺序严格执行驱逐
永远不会被删除。

如果设置了标志OFPTMPEF_OTHER，则驱逐过程将使用其他因素，例如内部约束，
执行搬迁。设置此标志时，控制器无法预测流条目将按哪个顺序
被驱逐。可以将此标志与其他标志组合以表示例如驱逐大部分
由交换机根据重要性或寿命来完成，但不严格。

OFPTMPT_VACANCY属性指定空缺事件的参数，并使用以下内容
结构和字段：

```
/ *空位表mod属性* /
struct ofp_table_mod_prop_vacancy {
        uint16_t            类型;        / * OFPTMPT_VACANCY。* /
        uint16_t            长度; / *此属性的长度（以字节为单位）。* /
        uint8_t vacancy_down;           / *空间减少时的空缺阈值（％）。* /
        uint8_t vacancy_up;             / *空间增加时的空缺阈值（％）。* /
        uint8_t_空缺;                   / *当前空缺率（％）-仅在ofp_table_desc中。* /
        uint8_t pad [1];                / *对齐64位。* /
};
OFP_ASSERT（sizeof（struct ofp_table_mod_prop_vacancy）== 8）;
```


字段vacancy_down和vacancy_up是生成空缺事件的阈值，该事件应
在此流表上配置，以百分比表示。

当流表中的剩余空间减少到小于vacancy_down时，以及是否有空位
启用down事件，必须使用

OFPT_TABLE_STATUS消息类型，其原因为OFPTR_VACANCY_DOWN（请参见7 .4.5 ）。进一步空缺
禁用事件，直到生成空缺事件为止。

流表中的剩余空间增加到超过vacancy_up时，以及是否出现空缺事件
启用后，必须使用OFPT_TABLE_STATUS向控制器生成一个空缺事件
消息类型，原因为OFPTR_VACANCY_UP。禁用更多的空缺活动，直到空缺为止
down事件生成。

如果使用OFPTC_VACANCY_EVENTS标志在表上启用了空缺事件，则va-
启用cancy up或vacancy down事件。启用事件时，如果当前空缺小于

vacancy_up，必须启用空缺事件，而必须禁用空缺事件。什么时候
启用事件，如果当前空缺大于或等于vacancy_up，则空缺事件必须
启用，并且必须禁用空缺事件。当桌子上的空缺事件被禁用时

使用OFPTC_VACANCY_EVENTS标志，必须同时禁用空置上升和空置下降事件。
如果在OFP_TABLE_MOD消息中，vacancy_down的值大于的值。

vacancy_up，交换机应拒绝OFP_TABLE_MOD消息并发送带有以下内容的ofp_error_msg
OFPET_BAD_PROPERTY类型和OFPBPC_BAD_VALUE代码。

只有当此属性包含在OFPMP_TABLE_DESC多部分中时，才使用空缺字段
回复或OFPT_TABLE_STATUS消息，并表示表的当前空缺，表示为
一个百分点。在OFP_TABLE_MOD请求中，此字段必须设置为0。

OFPTMPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者表mod属性* /
struct ofp_table_mod_prop_experimenter {
        uint16_t                类型;          / * OFPTMPT_EXPERIMENTER。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        uint32_t                实验者 / *采用相同的实验者ID
                                                形式如结构
                                                ofp_experimenter_header。* /
        uint32_t                exp_type;               / *实验者定义。* /
        /* 其次是：
         *      -确切地（长度-12个）字节包含实验者数据，然后
         *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
         *          全零字节的字节* /
        uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_table_mod_prop_experimenter的结构）== 12）；
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.3.4.2修改流条目消息

控制器对流表的修改是通过OFPT_FLOW_MOD消息完成的：

```
/ *流设置和拆除（控制器->数据路径）。* /
struct ofp_flow_mod {
        struct ofp_header标头；
        uint64_t cookie;                        / *不透明的控制器发出的标识符。* /
        uint64_t cookie_mask;                   / *用于限制cookie位的掩码
                                                    该命令必须匹配
                                                    OFPFC_MODIFY *或OFPFC_DELETE *。一个值
                                                    0表示没有限制。* /
        uint8_t table_id;                       / *要放入流的表的ID。
                                                    对于OFPFC_DELETE_ *命令，为OFPTT_ALL
                                                    也可以用来删除匹配项
                                                    所有表的流量。* /
        uint8_t命令；                           / * OFPFC_ *之一。* /
        uint16_t idle_timeout;                  / *丢弃前的空闲时间（秒）。* /
        uint16_t hard_timeout;                  / *丢弃前的最长时间（秒）。* /
        uint16_t优先级；                        / *流条目的优先级。* /
        uint32_t buffer_id;                     / *要应用于的缓冲数据包，或
                                                    OFP_NO_BUFFER。
                                                    对于OFPFC_DELETE *没有意义。* /
        uint32_t out_port;                      / *对于OFPFC_DELETE *命令，要求
                                                    匹配条目以将此作为
                                                    输出端口。值为OFPP_ANY
                                                    表示没有限制。* /
        uint32_t out_group;                     / *对于OFPFC_DELETE *命令，要求
                                                    匹配条目以将此作为
                                                    输出组。值为OFPG_ANY
                                                    表示没有限制。* /
        uint16_t标志；                          / * OFPFF_ *标志的位图。* /
        uint16_t重要性；                        / *驱逐优先级（可选）。* /
        struct ofp_match match;                 / *要匹配的字段。大小可变。* /
        / *变量大小和填充匹配始终带有指令。* /
        // p_instruction_header指令的结构[0];
                                        / *指令集-0或更多。长度
                                                的指令集是从
                                                标头中的长度字段。* /
};
OFP_ASSERT（sizeof（p_flow_mod的结构）== 56）；
```

cookie字段是控制器选择的不透明数据值。此值显示在已删除流中
消息（请参见7.4.2）和流描述多部分消息（请参见7.3.5.2），也可以用于过滤
流统计，流修改和流删除（请参见6. 4）。包处理不使用
流水线，因此不需要驻留在硬件中。值-1（0xffffffffffffffff）是保留值，必须为
不被使用。当流条目通过OFPFC_ADD消息插入表中时，其cookie字段
设置为提供的值。修改流条目时（OFPFC_MODIFY或OFPFC_MODIFY_STRICT
消息），其Cookie字段保持不变。

如果cookie_mask字段非零，则与cookie字段一起使用以限制流匹配，而
修改或删除流条目。OFPFC_ADD消息将忽略此字段。cookie_mask
字段的行为将在6.4节中说明。

table_id字段指定应将流条目插入，修改或删除的表。
表0表示管道中的第一个表。OFPTT_ALL的使用仅对删除请求有效。

命令字段必须是以下之一：

```
of p_flow_mod_command枚举{
    OFPFC_ADD              = 0，/ *新流程。* /
    OFPFC_MODIFY           = 1，/ *修改所有匹配的流。* /
    OFPFC_MODIFY_STRICT = 2，/ *修改严格匹配通配符的条目，并且
                                优先。* /
    OFPFC_DELETE           = 3，/ *删除所有匹配的流。* /
    OFPFC_DELETE_STRICT = 4，/ *删除与通配符严格匹配的条目，并且
                                优先。* /
};
```

OFPFC_MODIFY和OFPFC_MODIFY_STRICT之间以及OFPFC_DELETE和之间的差异
OFPFC_DELETE_STRICT在6.4节中进行了说明。需要一个OpenFlow交换机来实现所有
上面定义的命令。

idle_timeout和hard_timeout字段控制流条目过期的速度（请参阅6. 5）。什么时候
将流条目插入表中，并使用以下值设置其idle_timeout和hard_timeout字段
从消息中。修改流条目（OFPFC_MODIFY或OFPFC_MODIFY_STRICT消息）后，
idle_timeout和hard_timeout字段将被忽略。

流进入超时值由交换机流失效机制使用。如果idle_timeout
设置且hard_timeout为零，则该条目必须在idle_timeout秒后过期，且不能重新
感知流量。如果idle_timeout为零且设置了hard_timeout，则该条目必须在

hard_timeout秒数，无论数据包是否命中该条目。如果两者都为idle_timeout
和hard_timeout设置后，流条目将在idle_timeout秒后无流量超时，或者

hard_timeout秒，以先到者为准。如果idle_timeout和hard_timeout均为零，则
进入被认为是永久性的，永远不会超时。仍然可以使用flow_mod消息将其删除
类型为OFPFC_DELETE。规格未定义流到期过程的准确性，并且
取决于交换机的实现。

优先级字段指示指定流表中的流条目的优先级。更高
数字表示匹配数据包时的优先级更高（请参阅5.3）。该字段仅用于

匹配和添加流条目以及OFPFC_MODIFY_STRICT时的OFPFC_ADD消息或
匹配流条目时的OFPFC_DELETE_STRICT消息。该字段不用于OFPFC_MODIFY
或OFPFC_DELETE（非严格）消息。

buffer_id是指在交换机上缓冲并通过输入*包*发送到控制器的*数据包*
信息。如果没有缓冲的数据包与流mod关联，则必须将其设置为OFP_NO_BUFFER。
包含有效buffer_id的流mod将从缓冲区中删除相应的数据包，并
在插入流之后，从第一个流开始，通过整个OpenFlow管道对其进行处理
表。这实际上等效于发送流消息和数据包输出的两个消息序列
转发到OFPP_TABLE逻辑端口（请参见7.3.6），要求交换机必须完全
在发送数据包之前处理流程mod。这些语义适用，无论
请参阅flow mod或flow mod中包含的说明。该字段被OFPFC_DELETE忽略
和OFPFC_DELETE_STRICT流mod消息。

out_port和out_group字段可以选择过滤OFPFC_DELETE和

按输出端口和组的OFPFC_DELETE_STRICT消息。如果out_port或out_group包含
分别具有非OFPP_ANY或OFPG_ANY的值，当匹配-

ing。此约束是流条目必须包含指向该端口或

组。仍然使用其他约束，例如ofp_match结构和优先级。这纯粹是一个

*附加*约束。请注意，要禁用输出过滤，out_port和out_group都必须为
分别设置为OFPP_ANY和OFPG_ANY。这些字段将被OFPFC_ADD，OFPFC_MODIFY或

OFPFC_MODIFY_STRICT消息。

标志字段是位图，可能包含以下标志的组合：

```
of_p_flow_mod_flags的枚举{
    OFPFF_SEND_FLOW_REM = 1 << 0，/ *发送流时发送流删除消息
                                    *过期或被删除。* /
    OFPFF_CHECK_OVERLAP = 1 << 1，/ *首先检查重叠条目。* /
    OFPFF_RESET_COUNTS = 1 << 2，/ *重置流数据包和字节计数。* /
    OFPFF_NO_PKT_COUNTS = 1 << 3，/ *不跟踪数据包计数。* /
    OFPFF_NO_BYT_COUNTS = 1 << 4，/ *不跟踪字节数。* /
};
```

当设置了OFPFF_SEND_FLOW_REM标志时，交换机必须在发送流时发送流删除消息
条目到期或被删除（请参阅6 .5）。需要使用OpenFlow开关来兑现此标志并实施
相关的行为。

设置OFPFF_CHECK_OVERLAP标志时，交换机必须检查是否没有重叠流
优先级相同的条目，然后再将其插入流表（请参见6 .4）。流条目将
如果不具有相同的匹配项，则重叠；具有相同的优先级，并且一个数据包可以匹配两者
条目。如果存在，则流程模块失败并返回错误消息（请参阅7.5.4.6）。如果切换
不支持此标志，则必须返回OFPET_FLOW_MOD_FAILED类型的ofp_error_msg，

OFPFMFC_BAD_FLAGS代码。

设置OFPFF_RESET_COUNTS标志时，交换机必须清除计数器（字节和数据包计数）
如果没有设置此标志，则保留计数器。一个OpenFlow开关是
需要履行此标志并实现相关的行为。

设置OFPFF_NO_PKT_COUNTS标志时，交换机无需跟踪流数据包
计数。设置OFPFF_NO_BYT_COUNTS标志时，交换机无需跟踪流
字节数。设置这些标志可能会减少某些OpenFlow交换机上的处理负载，但是
这些计数器可能在此流条目的流统计信息和流删除消息中不可用。一个
不需要OpenFlow开关来遵守这两个标志，它可能会默默地忽略它们并保持
跟踪流计数并返回，尽管设置了相应的标志。如果开关不能保持
跟踪流量计数，相应的计数器不可用，必须将其设置为最大字段
值（请参阅5.9）。

将流条目插入表中时，其标志字段将使用消息中的值进行设置。什么时候
流条目被匹配和修改（OFPFC_MODIFY或OFPFC_MODIFY_STRICT消息），标志为
流条目不会更改，仅使用OFPFF_RESET_COUNTS，其他标志将被忽略。

匹配字段包含匹配结构和一组匹配字段，这些字段定义流条目如何匹配
数据包（请参阅7.2.3）。匹配字段和优先级字段的组合唯一标识流条目
在表中（请参阅5.2）。

重要性字段是流条目的重要性。该字段可以由
流进入驱逐机制（请参阅6.5）。将流条目插入表中时，其重要性
字段设置有消息中的值。修改流条目时（OFPFC_MODIFY或

OFPFC_MODIFY_STRICT消息），重要性字段将被忽略。

指令字段包含添加或修改时流条目的指令集
条目（请参阅7.2。5）。如果指令集无效或不受支持，则开关必须生成错误
（请参阅7.5.4.4）。

7.3.4.3修改组条目消息

控制器对分组表的修改是通过OFPT_GROUP_MOD消息完成的：

```
/ *组设置和拆除（控制器->数据路径）。* /
struct ofp_group_mod {
```

```
    struct ofp_header标头;
    uint16_t命令;                    / * OFPGC_ *之一。* /
    uint8_t类型;                     / * OFPGT_ *之一。* /
    uint8_t pad;                     / *填充到64位。* /
    uint32_t group_id;               / *组标识符。* /
    uint16_t bucket_array_len;       / *操作存储区数据的长度。* /
    uint8_t pad2 [2];                / *填充到64位。* /
    uint32_t command_bucket_id;      / *桶ID用作
                                       OFPGC_INSERT_BUCKET和OFPGC_REMOVE_BUCKET
                                       命令执行。* /
    /* 其次是：
     *      -确切的“ bucket_array_len”字节包含一个数组
     *         p_bucket结构。
     *      -零个或多个字节的组属性可填充整体
     *         标题中的长度。* /
    p_bucket桶的结构[0]; / *存储桶数组的长度为
                                       bucket_array_len字节。* /
    // p_group_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_group_mod的结构）== 24）；
```

类型和组字段的语义在6.7节中说明。

命令字段必须是以下之一：

```
/ *组命令* /
ofp_group_mod_command枚举{
    OFPGC_ADD      = 0，          / *新组。* /
    OFPGC_MODIFY = 1，            / *修改所有匹配的组。* /
    OFPGC_DELETE = 2              / *删除所有匹配的组。* /
    OFPGC_INSERT_BUCKET = 3，/ *将操作存储桶插入到已经可用的存储桶中
                                   匹配组中的操作段列表* /
    / * OFPGC _ ??? = 4，* /      / *保留供将来使用。* /
    OFPGC_REMOVE_BUCKET = 5，/ *删除所有操作存储桶或任何特定操作
                                   匹配组中的存储桶* /
};
```


OFPGC_ADD命令根据请求的内容创建一个新组（并返回一个错误
如果已经存在具有相同group_id的组）。命令OFPGC_MODIFY替换定义
具有请求内容的现有组。命令OFPGC_DELETE破坏组
由group_id指定；如果group_id为OFPG_ALL，则由所有组指定。命令OFPGC_INSERT_BUCKET
将存储桶添加到现有组。命令OFPGC_REMOVE_BUCKET从一个
现有组。

类型字段指定组的行为方式以及将使用哪些组桶来处理数据包
（请参阅5.10.1），它必须是以下值之一：

```
/ *组类型。[128，255]范围内的值保留用于实验
  * 采用。* /
of_p_group_type的枚举{
    OFPGT_ALL        = 0，/ *全部（多播/广播）组。* /
    OFPGT_SELECT     = 1，/ *选择组。* /
    OFPGT_INDIRECT = 2，/ *间接组。* /
    OFPGT_FF         = 3，/ *快速故障转移组。* /
};
```

group_id字段唯一地标识交换机中的一个组，一个组的组ID可以低一些
只要它在交换机上是唯一的，就不会超过OFPG_MAX。定义了以下特殊组标识符：

```
/ *组编号。群组最多可以使用OFPG_MAX个。* /
ofp_group的枚举{
    / *最后可用的组号。* /
    OFPG_MAX          = 0xffffff00，

    / *伪造的群组。* /
    OFPG_ALL          = 0xfffffffc，/ *表示要删除组的所有组
                                        命令。* /
    OFPG_ANY          = 0xffffffff          / *特殊通配符：未指定组。* /
};
```

bucket_array_len字段是buckets字段中的bucket数组的大小（以字节为单位）。

command_bucket_id字段是已存在于其中的存储桶之一的存储桶ID。
组或以下特殊存储桶标识符之一：

```
/ *存储桶ID可以是0到OFPG_BUCKET_MAX之间的任何值* /
of_p_group_bucket的枚举{
        OFPG_BUCKET_MAX  = 0xffffff00，/ *最后可用的存储区ID。* /
        OFPG_BUCKET_FIRST = 0xfffffffd，/ *操作列表中的第一个存储区ID
                                                一桶水。这适用
                                                对于OFPGC_INSERT_BUCKET和
                                                OFPGC_REMOVE_BUCKET命令。* /
        OFPG_BUCKET_LAST = 0xfffffffe，/ *操作列表中的最后一个存储桶ID
                                                一桶水。这适用
                                                对于OFPGC_INSERT_BUCKET和
                                                OFPGC_REMOVE_BUCKET命令。* /
        OFPG_BUCKET_ALL    = 0xffffffff / *一组中的所有操作段，
                                                这适用于
                                                仅OFPGC_REMOVE_BUCKET命令。* /
};
```


对command_bucket_id字段的解释以及对该字段有效的Bucket ID取决于
在命令字段上。对于OFPGC_INSERT_BUCKET，它定义了存储区列表中的位置以插入新的
桶。对于OFPGC_REMOVE_BUCKET，它定义要删除的存储桶。命令OFPGC_ADD，

OFPGC_MODIFY和OFPGC_DELETE不要使用command_bucket_id字段，对于这些命令，它必须
设置为OFPG_BUCKET_ALL。

buckets字段是动作bucket的列表（请参阅5.10）。组存储桶将在下一部分中介绍
（请参见7.3.4.3.1）。

属性字段是组属性的列表。组属性在以下部分中描述
（请参见7.3.4.3.2）。

7.3.4.3.1组存储桶

OFPT_GROUP_MOD消息中的存储桶字段是存储桶的列表。一个*间接*组必须包含
正好是一个存储桶（请参阅5.10.1），因此buckets字段必须包含一个存储桶以及任何
使这样的组具有多个存储桶的命令必须返回错误（请参阅7 .5.4.7）。
其他组类型在存储桶字段中可能有多个存储桶。对于*快速故障转移*组，存储桶
order确实定义了存储桶优先级（请参阅5.10 .1），并且可以通过修改来更改存储桶顺序
组（例如，使用带有命令OFPGC_MODIFY的OFPT_GROUP_MOD消息）。

组存储桶使用以下结构：

```
/ *供组使用的存储桶。* /
struct ofp_bucket {
        uint16_t len;                           / *存储桶的长度（以字节为单位），包括
                                                    这个标题和任何填充它
                                                    64位对齐。* /
        uint16_t action_array_len;              / *所有动作的长度（以字节为单位）。* /
        uint32_t bucket_id;                     / *用于标识存储桶的存储桶ID * /
        /* 其次是：
          *      -包含数组的“ action_array_len”字节
          *          p_action_ *结构。
          *      -零或更多字节的组存储桶属性以填写
          *          header.length中的总长度。* /
        struct ofp_action_header actions [0]; / *操作数组的长度为
                                                                action_array_len字节。* /
        // p_group_bucket_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_bucket的结构）== 8）；
```

action_array_len字段是action字段中一组操作的大小（以字节为单位）。

bucket_id字段唯一地标识交换机内组的操作桶。每桶
该组必须具有唯一的桶ID; 如果一个组mod会使该组有重复
存储桶ID，交换机必须拒绝组mod并发送*存储桶存在*错误消息（请参见7.5.4.7）。
bucket_id栏位必须小于OFPG_BUCKET_MAX（不能为特殊值区之一
值）。

操作字段是与存储桶关联的一组操作。当选择铲斗作为
数据包，其动作将作为操作集应用于数据包（请参见5.6）。

属性字段是组存储桶属性的列表。

当前定义的组存储桶属性类型的列表是：

```
/ *组存储桶属性类型。* /
of_p_group_bucket_prop_type的枚举{
    OFPGBPT_WEIGHT                  = 0，/ *仅选择组。* /
    OFPGBPT_WATCH_PORT              = 1，/ *仅快速故障转移组。* /
    OFPGBPT_WATCH_GROUP             = 2，/ *仅快速故障转移组。* /
    OFPGBPT_EXPERIMENTER        = 0xFFFF，/ *实验者定义。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有组存储桶属性的通用标头。* /
struct ofp_group_bucket_prop_header {
    uint16_t            类型;       / * OFPGBPT_ *之一。* /
    uint16_t            长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_group_bucket_prop_header的结构== 4）；
```

OFPGBPT_WEIGHT属性使用以下结构和字段：

```
/ *组铲斗重量属性，仅适用于选定组。* /
struct ofp_group_bucket_prop_weight {
    uint16_t            类型;       / * OFPGBPT_WEIGHT。* /
    uint16_t            长度; / * 8. * /
    uint16_t            重量; / *铲斗的相对重量。* /
    uint8_t             垫[2]; / *填充到64位。* /
};
OFP_ASSERT（sizeof（p_group_bucket_prop_weight的结构）== 8）；
```

权重字段仅为选定的组定义，其支持是可选的。对于其他组类型，
此属性必须省略。在某些组中，权重字段用于支持不相等的负载
分享。如果交换机不支持不均等的负载分担，则必须将此字段设置为1。
群组处理流量的份额由单个存储桶的权重除以
组中铲斗重量的总和。如果其权重设置为零，则选择器不使用该存储桶
组。当端口出现故障时，流量分布的变化是不确定的。精度
未定义应匹配桶权重的交换机的数据包分布。

OFPGBPT_WATCH_PORT和OFPGBPT_WATCH_GROUP属性使用以下结构，并且
领域：

```
/ *组存储区监视端口或监视组属性，用于快速故障转移组
  * 只要。* /
struct ofp_group_bucket_prop_watch {
    uint16_t            类型;       / * OFPGBPT_WATCH_PORT或OFPGBPT_WATCH_GROUP。* /
    uint16_t            长度; / * 8. * /
    uint32_t            看;         / *端口或组。* /
};
OFP_ASSERT（sizeof（p_group_bucket_prop_watch的结构）== 8）；
```


监视字段分别是存储桶正在监视的端口号或组标识符。

只有*快速故障转移*才需要OFPGBPT_WATCH_PORT和OFPGBPT_WATCH_GROUP属性
组，并且可以针对其他组类型选择性地实现。这些属性指示端口
和/或群组的活跃度控制此存储桶是否适合转发（请参见6.7）。如果
watch_port是OFPP_ANY，没有端口正在监视。如果watch_group为OFPG_ANY，则没有任何组
看着。对于快速故障转移组，定义的第一个存储桶是优先级最高的存储桶，只有
使用最高优先级的实时存储桶（请参阅5.10.1）。

OFPGBPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者组存储桶属性* /
p_group_bucket_prop_experimenter的结构{
    uint16_t            类型;               / * OFPGBPT_EXPERIMENTER。* /
```

```
    uint16_t                长度;                   / *此属性的长度（以字节为单位）。* /
    uint32_t                实验者 / *采用相同的实验者ID
                                                    形式如结构
                                                    ofp_experimenter_header。* /
    uint32_t                exp_type;               / *实验者定义。* /
    /* 其次是：
     *      -确切地（长度-12个）字节包含实验者数据，然后
     *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *          全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_group_bucket_prop_experimenter的结构== 12）；
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.3.4.3.2组属性

OFPT_GROUP_MOD消息中的属性字段是组属性的列表。

当前定义的组属性类型的列表为：

```
/ *组属性类型。* /
of_p_group_prop_type的枚举{
    OFPGPT_EXPERIMENTER         = 0xFFFF，/ *实验者定义。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有组属性的通用头。* /
struct ofp_group_prop_header {
    uint16_t                类型;           / * OFPGPT_ *之一。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_group_prop_header的结构）== 4）；
```

OFPGPT_EXPERIMENTER属性使用以下结构和字段：


```
/ *实验者组属性* /
struct ofp_group_prop_experimenter {
    uint16_t                类型;                   / * OFPGPT_EXPERIMENTER。* /
    uint16_t                长度;                   / *此属性的长度（以字节为单位）。* /
    uint32_t                实验者 / *采用相同的实验者ID
                                                    形式如结构
                                                    ofp_experimenter_header。* /
    uint32_t                exp_type;               / *实验者定义。* /
    /* 其次是：
     *      -确切地（长度-12个）字节包含实验者数据，然后
     *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *          全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_group_prop_experimenter的结构）== 12）；
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.3.4.4端口修改消息

控制器使用OFPT_PORT_MOD消息来修改端口的行为：

```
/ *修改物理端口的行为* /
struct ofp_port_mod {
    struct ofp_header标头；
    uint32_t port_no;
    uint8_t pad [4];
    uint8_t hw_addr [OFP_ETH_ALEN]; / *硬件地址不是
                                                    可配置的。这用来
                                                    检查请求，因此必须
                                                    与返回的相同
                                                    ofp_port结构。* /
    uint8_t pad2 [2];               / *填充到64位。* /
    uint32_t配置;                   / * OFPPC_ *标志的位图。* /
```

```
    uint32_t掩码;                    / *要更改的OFPPC_ *标志的位图。* /
    / *端口mod属性列表-0个或更多属性* /
    p_port_mod_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_port_mod）== 32）;
```

config字段描述端口管理设置（请参阅7.2.1）。

掩码字段用于选择配置字段中的位以进行更改。

属性字段是端口描述属性的列表，可用于配置其他属性
端口的属性。所有端口属性都不必包含在OFPT_PORT_MOD消息中，
省略的属性保持不变。

当前定义的端口描述属性类型的列表是：


```
/ *端口mod属性类型。
  * /
of_p_port_mod_prop_type的枚举{
    OFPPMPT_ETHERNET          = 0，      / *以太网属性。* /
    OFPPMPT_OPTICAL           = 1        / *光学性质。* /
    OFPPMPT_EXPERIMENTER      = 0xFFFF，/ *实验者属性。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有端口mod属性的通用头。* /
struct ofp_port_mod_prop_header {
    uint16_t          类型;      / * OFPPMPT_ *之一。* /
    uint16_t          长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_port_mod_prop_header的结构）== 4）;
```

OFPPMPT_ETHERNET属性使用以下结构和字段：

```
/ *以太网端口mod属性。* /
struct ofp_port_mod_prop_ethernet {
    uint16_t          类型;          / * OFPPMPT_ETHERNET。* /
    uint16_t          长度;          / *此属性的长度（以字节为单位）。* /
    uint32_t          广告; / * OFPPF_ *的位图。将所有位清零以防止
                                      采取任何行动。* /
};
OFP_ASSERT（sizeof（p_port_mod_prop_ethernet的结构）== 8）;
```

播发字段描述了在链接（7.2.1）上播发的以太网功能。它没有面具。所有港口
功能一起改变。

OFPPMPT_OPTICAL属性使用以下结构和字段：

```
struct ofp_port_mod_prop_optical {
    uint16_t          类型;          / * OFPPMPT_OPTICAL。* /
    uint16_t          长度;          / *此属性的长度（以字节为单位）。* /
    uint32_t          配置; / * OFPOPF_ *的位图。* /
    uint32_t          freq_lmda; / *“中心”频率* /
    int32_t           fl_offset; / *带符号的频偏* /
    uint32_t          grid_span; / *此端口的网格大小* /
    uint32_t          tx_pwr;        / *发射功率设置* /
};
OFP_ASSERT（sizeof（p_port_mod_prop_optical的结构）== 24）;
```

配置字段描述了要更改此端口的光学功能。可以设置OFPOPF_ *中的任何一个。
频率以MHz为单位指定，波长（λ）为nm *100。OFPOPF_USE_FREQ必须匹配
公告的端口功能。tx_pwr为dBm * 10。

调谐频率是freq_lmda和fl_offset之和。这是为了方便软件，
因为某些调整选项（包括Flex Grid）可能基于中心频率和偏移量。
“中心”频率通常用于无源滤波器，这再次使软件方便。

grid_span是此端口消耗的带宽量，对于Flex Grid和其他
调整信息。

OFPPMPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者端口mod属性。* /
struct ofp_port_mod_prop_experimenter {
        uint16_t                类型;         / * OFPPMPT_EXPERIMENTER。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        uint32_t                实验者 / *采用相同的实验者ID
                                                形式如结构
                                                ofp_experimenter_header。* /
        uint32_t                exp_type;               / *实验者定义。* /
        /* 其次是：
         *     -确切地（长度-12个）字节包含实验者数据，然后
         *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
         *        全零字节的字节* /
        uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_port_mod_prop_experimenter的结构）== 12）；
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.3.4.5仪表修改消息

使用OFPT_METER_MOD消息可以对控制器的仪表进行修改：

```
/ *仪表配置。OFPT_METER_MOD。* /
struct ofp_meter_mod {
        struct ofp_header标头；
        uint16_t                命令;                   / * OFPMC_ *之一。* /
        uint16_t                标志;                   / * OFPMF_ *标志的位图。* /
        uint32_t                meter_id;               / *仪表实例。* /
        struct ofp_meter_band_header bands [0]; / *波段列表长度为
                                                        从长度字段推断
                                                        在标题中。* /
};
OFP_ASSERT（sizeof（p_meter_mod的结构）== 16）；
```

meter_id字段唯一地标识交换机内的仪表。仪表的定义始于
meter_id = 1，最高可达交换机可以支持的最大仪表数。开放流
交换协议还定义了一些其他无法与流关联的虚拟仪表：

```
/ *仪表编号。流量计最多可以使用OFPM_MAX个数字。* /
ofp_meter {
        / *最后可用的仪表。* /
        OFPM_MAX            = 0xffff0000，

        / *虚拟仪表。* /
        OFPM_SLOWPATH   = 0xfffffffd，/ *计量慢速数据路径。* /
```



```
        OFPM_CONTROLLER = 0xfffffffe，/ *用于控制器连接的仪表。* /
        OFPM_ALL            = 0xffffffff，/ *代表状态请求的所有仪表
                                                命令。* /
};
```

提供虚拟仪表以支持OpenFlow的现有实现。新的实施
建议使用常规的每流量计（请参阅5 .11）或优先级队列（请参阅7.3.5.8）。

- OFPM_CONTROLLER：虚拟仪表控制通过发送到控制器的所有数据包*分组进入* mes-
  使用**CONTROLLER**保留端口或进行其他处理（见6.1。2）。可
  用于限制发送到控制器的流量。

•OFPM_SLOWPATH：虚拟仪表控制交换机的慢速数据路径处理的所有数据包。

许多交换机的实现都有快速和慢速的数据路径，例如，硬件交换机可能会拥有缓慢的软件数据路径，或者软件开关可能拥有缓慢的用户空间数据路径。

命令字段必须是以下之一：

```
/ *仪表命令* /
ofp_meter_mod_command枚举{
    OFPMC_ADD     = 0，          / *新仪表。* /
    OFPMC_MODIFY = 1            / *修改指定的仪表。* /
    OFPMC_DELETE = 2            / *删除指定的仪表。* /
};
```

标志字段是位图，可能包含以下标志的组合：

```
/ *仪表配置标志* /
of_p_meter_flags枚举{
    OFPMF_KBPS        = 1 << 0，/ *速率值（kb / s）（每秒千比特）。* /
    OFPMF_PKTPS       = 1 << 1，/ *速率值（以包/秒为单位）。* /
    OFPMF_BURST       = 1 << 2，/ *执行突发大小。* /
    OFPMF_STATS       = 1 << 3，/ *收集统计信息。* /
};
```

标志OFPMF_PKTPS指定应解释仪表频带速率和burst_size作为每秒的数据包值。如果未设置标志OFPMF_PKTPS，则标志OFPMF_KBPS指定仪表带速率和burst_size应该解释为千比特每秒的值。旗

OFPMF_STATS指定如果未设置标志，则电表必须收集其支持的电表统计信息不需要收集统计信息。

标志OFPMF_BURST指定必须使用仪表带的burst_size值。如果旗

未设置OFPMF_BURST，并且会忽略仪表频带中的burst_size值，如果仪表实现使用突发值，此突发值必须设置为实现定义的最佳值值。

频段字段是速率频段的列表。它可以包含任意数量的频段，每种频段类型可以是在有意义的时候重复一遍。仪表默认频段（速率0）不能包括在内，也不能在其中配置该列表，对数据包不执行任何操作。每个数据包仅使用一个频段，频段选择该过程基于频段速率和电表测量速率（请参阅5.11.1）。

所有仪表带均使用相同的公共标头定义：


```
/ *所有仪表波段的通用标头* /
struct ofp_meter_band_header {
    uint16_t        类型;      / * OFPMBT_ *之一。* /
    uint16_t        len;       / *此频段的长度（以字节为单位）。* /
    uint32_t        率;        / *此频段的价格。* /
    uint32_t        突发大小; / *突发大小。* /
};
OFP_ASSERT（sizeof（p_meter_band_header的结构）== 12）;
```

速率字段指示速率值，在该速率值之上，相应的频段可应用于数据包（请参见5.11.1）。速率值以每秒千比特为单位，除非flags字段包含OFPMF_PKTPS，其中情况下，速率是每秒的数据包数。

仅当flags字段包含OFPMF_BURST时才使用burst_size字段。如果OFPMF_BURST标志为如果未设置，则必须将burst_size字段设置为0。它定义了仪表带的粒度，对于所有长度大于突发值的数据包或字节突发，则仪表速率始终严格强制执行。除非标志字段包含OFPMF_PKTPS，否则突发值以千比特为单位，在这种情况下，突发值以包为单位。

类型字段必须是以下之一：

```
/ *表带类型* /
of_p_meter_band_type的枚举{
    OFPMBT_DROP             = 1           / *丢弃数据包。* /
    OFPMBT_DSCP_REMARK      = 2           / *在IP标头中标记DSCP。* /
    OFPMBT_EXPERIMENTER     = 0xFFFF / *实验仪仪表带。* /
};
```

OpenFlow交换机可能不支持所有频段类型，并且可能不允许使用其所有受支持的交换机所有仪表上的频段类型，即某些仪表可能是专门的。

OFPMBT_DROP频段定义了一个简单的速率限制器，用于丢弃超过频段速率值的数据包，并使用以下结构：

```
/ * OFPMBT_DROP频段-丢包* /
struct ofp_meter_band_drop {
        uint16_t            类型;       / * OFPMBT_DROP。* /
        uint16_t            len;        / *长度为16。* /
        uint32_t            率;         / *丢包率。* /
        uint32_t            突发大小; / *突发大小。* /
        uint8_t             垫[4];
};
OFP_ASSERT（sizeof（p_meter_band_drop的结构）== 16）;
```

OFPMBT_DSCP_REMARK乐队定义了一个简单的DiffServ策略器，用于标记的丢弃优先级
超过带宽速率值的数据包的IP报头中的DSCP字段，并使用以下内容
结构体：


```
/ * OFPMBT_DSCP_REMARK频段-在IP标头中标记DSCP * /
struct ofp_meter_band_dscp_remark {
        uint16_t            类型;       / * OFPMBT_DSCP_REMARK。* /
        uint16_t            len;        / *长度为16。* /
        uint32_t            率;         / *标记数据包的速率。* /
        uint32_t            突发大小; / *突发大小。* /
        uint8_t             prec_level; / *要添加的丢弃优先级的数量。* /
        uint8_t             垫[3];
};
OFP_ASSERT（sizeof（struct ofp_meter_band_dscp_remark）== 16）;
```

prec_level字段指示应将数据包的丢弃优先级提高多少
如果超出频段。该频段将编码的丢弃优先级增加此数量，而不是原始数量
DSCP值; 它始终会产生有效的DSCP值，并且不会对丢弃进行编码的DSCP值
优先级未修改。

OFPMBT_EXPERIMENTER频段是由实验人员定义的，并使用以下结构：

```
/ * OFPMBT_EXPERIMENTER波段-实验者类型。
  *乐队的其余部分由实验者定义。* /
struct ofp_meter_band_experimenter {
        uint16_t            类型;       / * OFPMBT_EXPERIMENTER。* /
        uint16_t            len;        / *此频段的长度（以字节为单位）。* /
        uint32_t            率;         / *此频段的价格。* /
        uint32_t            突发大小;           / *突发大小。* /
        uint32_t            实验者 / *实验者ID。* /
};
OFP_ASSERT（sizeof（p_meter_band_experimenter的结构）== 16）;
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。乐队的其余部分是由实验者定义的。

7.3.5多部分消息

多部分消息用于编码可能携带大量数据的请求或答复
并且并不总是适合单个OpenFlow消息，该消息限制为64KB。要求或
答复被编码为与特定分段相同的连接上的分段消息序列
类型，然后由接收器重新组装。每个多部分消息序列都包含一个多部分
请求或答复。多部分消息主要用于请求统计信息或状态信息
开关。

该请求在一个或多个OFPT_MULTIPART_REQUEST消息中承载：

```
struct ofp_multipart_request {
        struct ofp_header标头;
        uint16_t类型;                       / * OFPMP_ *常量之一。* /
        uint16_t标志;                       / * OFPMPF_REQ_ *标志。* /
        uint8_t pad [4];
        uint8_t body [0];                   / *请求的正文。0个或更多字节。* /
};
OFP_ASSERT（sizeof（struct ofp_multipart_request）== 16）;
```

第124页 OpenFlow交换机规格 版本1.5.1

交换机以一个或多个OFPT_MULTIPART_REPLY消息作为响应：

```
struct ofp_multipart_reply {
    struct ofp_header标头；
    uint16_t类型;                    / * OFPMP_ *常量之一。* /
    uint16_t标志；                   / * OFPMPF_REPLY_ *标志。* /
    uint8_t pad [4];
    uint8_t body [0];                / *回复的正文。0个或更多字节。* /
};
OFP_ASSERT（sizeof（p_multipart_reply的结构）== 16）；
```

正文字段包含请求或答复的一部分。每个多部分的请求或答复都已定义
可以是单个结构，也可以是0个或多个相同类型结构的数组。如果是多部分
请求或答复定义为单个结构，它必须使用单个多部分消息，而整个消息
请求或答复必须包含在正文中。如果将多部分请求或答复定义为数组
的结构，主体字段必须包含整数个对象，并且不能拆分任何对象
跨两个消息。为了简化实施，可以将定义为数组的多部分请求或答复使用
在多部分序列的任何点上都没有其他条目（即空主体）的消息。

标志字段是用于控制分段/重组过程的位图。在多部分请求中
消息，它可能包含以下标志的组合：

```
of_p_multipart_request_flags的枚举{
    OFPMPF_REQ_MORE = 1 << 0 / *还有更多请求。* /
};
```

在多部分答复消息中，它可能包含以下标志的组合：

```
of_p_multipart_reply_flags的枚举{
    OFPMPF_REPLY_MORE = 1 << 0 / *更多答复。* /
};
```

OFPMPF_REQ_MORE位和OFPMPF_REPLY_MORE位指示将有更多请求/回复
按照当前的。如果在多部分消息中设置了OFPMPF_REQ_MORE或OFPMPF_REPLY_MORE，则
同一多部分序列的另一多部分消息必须始终跟随该消息。一个要求
或回复跨越多封邮件（已设置一个或多个邮件并设置了*更多*标志），则必须使用
消息序列中所有消息的相同的多部分类型和事务处理ID（xid）。

来自多部分请求或回复的消息可以与其他OpenFlow消息类型进行交织，
包括其他多部分请求或答复，但在连接上必须具有不同的交易ID
如果同时有多个未答复的多部分请求或答复。的交易编号
答复必须始终与提示他们的请求匹配。

如果多部分请求跨越多条消息，并且增长到交换机无法通过的大小
要缓冲，交换机必须以OFPET_BAD_REQUEST类型的错误消息和代码作为响应

OFPBRC_MULTIPART_BUFFER_OVERFLOW。如果多部分请求包含不支持的类型，
移植后，交换机必须以OFPET_BAD_REQUEST类型和代码的错误消息响应

OFPBRC_BAD_MULTIPART。如果交换机收到的多部分消息具有与多部分相同的xid
先前在未终止的同一连接上接收到的序列（未收到消息

第125页 OpenFlow交换机规格 版本1.5.1

没有设置*更多*标志），并且具有不同的多部分类型，则该开关必须响应一个错误
类型为OFPET_BAD_REQUEST的消息和代码为OFPBRC_BAD_MULTIPART。如果是多部分请求序列
包含一个以上的多部分请求或单个请求之外的其他数据，交换机必须响应
错误消息类型为OFPET_BAD_REQUEST，代码为OFPBRC_BAD_LEN。

如果交换机收到的多部分请求消息序列不包含带有

在开关定义的时间量大于100毫秒后，OFPMPF_REQ_MORE标志设置为零
从最后一条消息开始，交换机必须丢弃不完整的多部分请求，并且可能会生成一个
类型错误的错误消息OFPET_BAD_REQUEST和代码OFPBRC_MULTIPART_REQUEST_TIMEOUT
拖钓者。如果控制器收到的多部分回复消息序列不包含带有

在控制器定义的时间量大于1秒后，OFPMPF_REPLY_MORE标志设置为零
从最后一条消息开始，控制器必须丢弃不完整的多部分回复，并且可能会生成一个
OFPET_BAD_REQUEST类型的错误消息和代码OFPBRC_MULTIPART_REPLY_TIMEOUT到交换机。
控制器可以根据需要重试失败的开关操作，作为新的多部分请求。

在所有包含统计信息的多部分答复中，如果特定的数字计数器在
开关，如果使用OXS描述计数器（请参见7.2.4.2），则必须将其省略，否则其值必须
设置为最大字段值（无符号等效值-1）。计数器未签名并包装
周围没有溢出指示器。计数器必须使用为计数器定义的完整位范围
例如，如果将计数器定义为64位，则它不能仅使用低32位。

在请求和响应中，类型字段都指定要传递的信息的类型，
确定如何解释正文字段：

```
of_p_multipart_type的枚举{
    / *此OpenFlow开关的说明。
      *请求正文为空。
      *回复正文为p_desc的结构。* /
    OFPMP_DESC = 0，

    / *各个流的描述和统计信息。
      *请求主体是p_flow_stats_request的结构。
      *回复主体是p_flow_desc结构的数组。* /
    OFPMP_FLOW_DESC = 1

    / *汇总流量统计信息。
      *请求主体是p_aggregate_stats_request的结构。
      *回复主体为p_aggregate_stats_reply的结构。* /
    OFPMP_AGGREGATE_STATS = 2

    / *流表统计信息。
      *请求正文为空。
      *回复正文是p_table_stats结构的数组。* /
    OFPMP_TABLE_STATS = 3，

    / *端口统计信息。
      *请求主体是p_port_multipart_request的结构。
      *回复主体是p_port_stats结构的数组。* /
    OFPMP_PORT_STATS = 4

    / *端口的队列统计信息
      *请求主体是p_queue_multipart_request的结构。
      *回复主体是p_queue_stats结构的数组* /
```



```
    OFPMP_QUEUE_STATS = 5

    / *组计数器统计信息。
      *请求主体是p_group_multipart_request的结构。
      *答复是p_group_stats结构的数组。* /
    OFPMP_GROUP_STATS = 6

    /* 团体简介。
      *请求主体是p_group_multipart_request的结构。
      *回复正文是p_group_desc结构的数组。* /
    OFPMP_GROUP_DESC = 7

    / *组功能。
      *请求正文为空。
      *回复正文是p_group_features的结构。* /
    OFPMP_GROUP_FEATURES = 8

    / *仪表统计信息。
      *请求主体是p_meter_multipart_request的结构。
      *回复主体是p_meter_stats结构的数组。* /
    OFPMP_METER_STATS = 9

    / *仪表配置。
      *请求主体是p_meter_multipart_request的结构。
      *回复正文是p_meter_desc结构的数组。* /
    OFPMP_METER_DESC = 10，

    / *仪表功能。
      *请求正文为空。
      *回复正文为p_meter_features的结构。* /
    OFPMP_METER_FEATURES = 11

    / *表功能。
      *请求正文为空或包含一个数组
      *包含控制器的p_table_features结构
```

```
        *所需的开关视图。如果开关无法
        *设置指定的视图，返回错误。
        *回复主体是p_table_features结构的数组。* /
    OFPMP_TABLE_FEATURES = 12

    / *端口说明。
        *请求主体是p_port_multipart_request的结构。
        *回复正文是struct ofp_port的数组。* /
    OFPMP_PORT_DESC = 13

    / *表说明。
        *请求正文为空。
        *回复主体是p_table_desc结构的数组。* /
    OFPMP_TABLE_DESC = 14

    / *队列描述。
        *请求主体是p_queue_multipart_request的结构。
        *回复正文是struct ofp_queue_desc的数组。* /
    OFPMP_QUEUE_DESC = 15

    / *流量监视器。回复可能是异步消息。
        *请求主体是p_flow_monitor_request结构的数组。
```



```
        *回复正文是p_flow_update_header结构的数组。* /
    OFPMP_FLOW_MONITOR = 16

    / *单个流统计信息（无描述）。
        *请求主体是p_flow_stats_request的结构。
        *回复主体是p_flow_stats结构的数组。* /
    OFPMP_FLOW_STATS = 17

    / *控制器状态。
        *请求正文为空。
        *回复主体是p_controller_status结构的数组。* /
    OFPMP_CONTROLLER_STATUS = 18，

    / *捆绑功能。
    *请求主体是ofp_bundle_features_request。
    *回复正文是p_bundle_features的结构。* /
    OFPMP_BUNDLE_FEATURES = 19，

    / *实验者扩展。
        *请求和回复正文以
        *结构为p_experimenter_multipart_header。
        *请求和回复主体是实验者定义的。* /
    OFPMP_EXPERIMENTER = 0xffff
};
```

7.3.5.1说明

有关交换机制造商，硬件版本，软件版本，序列号和相关信息的信息
描述字段可从OFPMP_DESC多部分请求类型获得：

```
/ *对OFPMP_DESC请求的回复正文。每个条目都是以NULL终止的
  * ASCII字符串。* /
struct ofp_desc {
    字符mfr_desc [DESC_STR_LEN];          / *制造商说明。* /
    字符hw_desc [DESC_STR_LEN];           / *硬件描述。* /
    char sw_desc [DESC_STR_LEN];          / *软件说明。* /
    字符serial_num [SERIAL_NUM_LEN];      /* 序列号。* /
    字符dp_desc [DESC_STR_LEN];           / *人类可读的描述
                                            数据路径。* /
};
OFP_ASSERT（sizeof（struct ofp_desc）== 1056）；
```

每个条目均采用ASCII格式，并在右侧用空字节（\0）填充。DESC_STR_LEN为256，

SERIAL_NUM_LEN是32。dp_desc字段是一个自由格式的字符串，用于描述调试的数据路径
目的，例如“房间3120中的switch3”。因此，不能保证它是唯一的，并且不应该是唯一的。
用作数据路径的主要标识符-使用开关功能中的datapath_id字段
相反（请参见7.3.1）。

7.3.5.2单个流说明

通过OFPMP_FLOW_DESC多部分请求来请求有关各个流条目的信息
类型：


```
/ *类型为OFPMP_FLOW_DESC和OFPMP_FLOW_STATS的ofp_multipart_request的正文。* /
struct ofp_flow_stats_request {
    uint8_t table_id;               / *要读取的表的ID（从ofp_table_desc），
                                        所有表的OFPTT_ALL。* /
    uint8_t pad [3];                / *对齐32位。* /
    uint32_t out_port;              / *需要匹配条目才能包含此条目
                                        作为输出端口。值为OFPP_ANY
                                        表示没有限制。* /
    uint32_t out_group;             / *需要匹配条目才能包含此条目
                                        作为输出组。值为OFPG_ANY
                                        表示没有限制。* /
    uint8_t pad2 [4];               / *对齐64位。* /
    uint64_t cookie;                / *需要匹配的条目才能包含此
                                        Cookie值* /
    uint64_t cookie_mask;           / *用于限制cookie位的掩码
                                        必须匹配。值为0表示
                                        没有限制。* /
    struct ofp_match match;         / *要匹配的字段。大小可变。* /
};
OFP_ASSERT（sizeof（p_flow_stats_request的结构）== 40）；
```

匹配字段包含应匹配的流条目的描述，并且可能包含
通配符和掩码字段。该字段的匹配行为在6.4节中描述。

table_id字段指示要读取的单个表的索引，或所有表的OFPTT_ALL。

out_port和out_group字段可选地按输出端口和组过滤。如果是out_port或

out_group分别包含OFPP_ANY和OFPG_ANY以外的值，它引入了约束
匹配时。该约束是流条目必须包含针对以下内容的输出操作：
该端口或组。其他约束如比赛场仍在使用; 这纯粹是*额外的*
约束。请注意，要禁用输出过滤，必须将out_port和out_group都设置为OFPP_ANY
和OFPG_ANY。

cookie和cookie_mask字段的用法在6.4节中定义。

对OFPMP_FLOW_DESC多部分请求的答复的正文由以下数组组成：

```
/ *对OFPMP_FLOW_DESC请求的回复正文。* /
struct ofp_flow_desc {
    uint16_t长度；                  / *此项的长度。* /
    uint8_t pad2 [2];               / *对齐64位。* /
    uint8_t table_id;               / *表流的ID来自。* /
    uint8_t pad;
    uint16_t优先级；                / *条目的优先级。* /
    uint16_t idle_timeout;          / *到期前空闲的秒数。* /
    uint16_t hard_timeout;          / *到期前的秒数。* /
    uint16_t标志；                  / * OFPFF_ *标志的位图。* /
    uint16_t重要性；                / *驱逐优先级。* /
    uint64_t cookie;                / *不透明的控制器发出的标识符。* /
    struct ofp_match match;         / *字段说明。大小可变。* /
    // struct ofp_stats统计信息；/ *统计信息列表。大小可变。* /
    // p_instruction_header指令的结构[0];
                                    / *指令集-0或更多。* /
};
OFP_ASSERT（sizeof（struct ofp_flow_desc）== 32）；
```


这些字段包括在创建流条目的flow_mod中提供的字段（请参见7 .3.4.2）、以及

插入条目的table_id。

stats字段包含OXS字段的列表（请参见7.2.4 .2）。字段OXS_OF_DURATION是必填项
并跟踪流条目已在交换机中安装多长时间。栏位OXS_OF_PACKET_COUNT
和OXS_OF_BYTE_COUNT是可选的，并且对流条目处理的所有数据包进行计数。

7.3.5.3个人流量统计
还可以使用OFPMP_FLOW_STATS多个部分来请求有关各个流条目的统计信息
请求类型：

```
/ *类型为OFPMP_FLOW_DESC和OFPMP_FLOW_STATS的ofp_multipart_request的正文。* /
struct ofp_flow_stats_request {
        uint8_t table_id;               / *要读取的表的ID（从ofp_table_desc），
                                            所有表的OFPTT_ALL。* /
        uint8_t pad [3];                / *对齐32位。* /
        uint32_t out_port;              / *需要匹配条目才能包含此条目
                                            作为输出端口。值为OFPP_ANY
                                            表示没有限制。* /
        uint32_t out_group;             / *需要匹配条目才能包含此条目
                                            作为输出组。值为OFPG_ANY
                                            表示没有限制。* /
        uint8_t pad2 [4];               / *对齐64位。* /
        uint64_t cookie;                / *需要匹配的条目才能包含此
                                            Cookie值* /
        uint64_t cookie_mask;           / *用于限制cookie位的掩码
                                            必须匹配。值为0表示
                                            没有限制。* /
        struct ofp_match match;         / *要匹配的字段。大小可变。* /
};
OFP_ASSERT（sizeof（p_flow_stats_request的结构）== 40）；
```

该消息中的字段具有与各个流desc请求类型相同的含义
（OFPMP_FLOW_DESC-参见7 ,3.5.2）。

对OFPMP_FLOW_STATS多部分请求的答复的正文由以下数组组成：

```
/ *对OFPMP_FLOW_STATS请求的回复正文
  *和正文为OFPIT_STAT_TRIGGER生成的状态。* /
struct ofp_flow_stats {
        uint16_t长度；                  / *此项的长度。* /
        uint8_t pad2 [2];               / *对齐64位。* /
        uint8_t table_id;               / *表流的ID来自。* /
        uint8_t原因；                   / * OFPFSR_ *之一。* /
        uint16_t优先级；                / *条目的优先级。* /
        struct ofp_match match;         / *字段说明。大小可变。* /
        // struct ofp_stats统计信息；  / *统计信息列表。大小可变。* /
};
OFP_ASSERT（sizeof（struct ofp_flow_stats）== 16）；
```


该消息中的字段具有与各个流程desc回复类型相同的含义
（OFPMP_FLOW_DESC-参见7 ,3.5.2）。

原因字段是以下内容之一：

```
/ *生成流量统计信息的原因。* /
of_p_flow_stats_reason枚举{
        OFPFSR_STATS_REQUEST = 0，      / *回复OFPMP_FLOW_STATS请求。* /
        OFPFSR_STAT_TRIGGER = 1         / *状态由OFPIT_STAT_TRIGGER生成。* /
};
```

原因值OFPRR_STATS_REQUEST表示该消息是对OFPMP_FLOW_STATS的答复
多部分请求。原因值OFPRR_STAT_TRIGGER表示消息是事件（状态）
当在*统计触发*指令中超过流量统计阈值之一时生成（见7.2.5）。

7.3.5.4汇总流量统计

需使用OFPMP_AGGREGATE_STATS mult请求有关多个流条目的汇总信息
tipart请求类型：

```
/ *类型为OFPMP_AGGREGATE_STATS的ofp_multipart_request的主体。* /
struct ofp_aggregate_stats_request {
        uint8_t table_id;               / *要读取的表的ID（来自ofp_table_stats）
                                            所有表的OFPTT_ALL。* /
        uint8_t pad [3];                / *对齐32位。* /
        uint32_t out_port;              / *需要匹配条目才能包含此条目
                                            作为输出端口。值为OFPP_ANY
                                            表示没有限制。* /
```

```
    uint32_t out_group;          / *需要匹配条目才能包含此条目
                                   作为输出组。值为OFPG_ANY
                                   表示没有限制。* /
    uint8_t pad2 [4];            / *对齐64位。* /
    uint64_t cookie;             / *需要匹配的条目才能包含此
                                   Cookie值* /
    uint64_t cookie_mask;        / *用于限制cookie位的掩码
                                   必须匹配。值为0表示
                                   没有限制。* /
    struct ofp_match match;      / *要匹配的字段。大小可变。* /
};
OFP_ASSERT（sizeof（p_aggregate_stats_request的结构）== 40）；
```

该消息中的字段具有与各个流desc请求类型相同的含义
（OFPMP_FLOW_DESC-参见7 .3.5.2）。

答复的正文包括以下内容：

```
/ *对OFPMP_AGGREGATE_STATS请求的回复正文。* /
struct ofp_aggregate_stats_reply {
    struct ofp_stats统计信息；   / *汇总统计信息列表。大小可变。* /
};
OFP_ASSERT（sizeof（struct ofp_aggregate_stats_reply）== 8）；
```


该统计数据字段包含OXS字段列表（见7.2。 4.2）。字段OXS_OF_FLOW_COUNT是必填项
并且等于与请求匹配的流条目的数量。栏位OXS_OF_PACKET_COUNT
和OXS_OF_BYTE_COUNT是可选的，并计算所有与之匹配的流条目处理的所有数据包
请求。

7.3.5.5端口统计

使用OFPMP_PORT_STATS多部分请求类型请求有关端口统计信息的信息：

```
/ *类型为OFPMP_PORT_STATS的ofp_multipart_request的主体，
  * OFPMP_PORT_DESC。* /
struct ofp_port_multipart_request {
    uint32_t port_no;            / * OFPMP_PORT消息必须请求统计信息
                                   *单个端口（在
                                   * port_no）或所有端口（如果port_no ==
                                   * OFPP_ANY）。* /
    uint8_t pad [4];
};
OFP_ASSERT（sizeof（p_port_multipart_request的结构）== 8）；
```

port_no字段可以选择将统计信息请求过滤到给定的端口。要请求所有端口统计信息，

port_no必须设置为OFPP_ANY。

答复的正文由以下数组组成：

```
/ *对OFPMP_PORT_STATS请求的回复正文。如果不支持计数器，
  *将字段设置为全部。* /
struct ofp_port_stats {
    uint16_t长度；               / *此项的长度。* /
    uint8_t pad [2];             / *对齐64位。* /
    uint32_t port_no;
    uint32_t duration_sec;       / *时间端口以秒为单位处于活动状态。* /
    uint32_t duration_nsec; / *时间端口已经存活了十亿分之一秒
                                   duration_sec。* /
    uint64_t rx_packets;         / *接收到的数据包数。* /
    uint64_t tx_packets;         / *传输的数据包数。* /
    uint64_t rx_bytes;           / *接收的字节数。* /
    uint64_t tx_bytes;           / *传输的字节数。* /

    uint64_t rx_dropped;         / * RX丢弃的数据包数。* /
    uint64_t tx_dropped;         / * TX丢弃的数据包数。* /
    uint64_t rx_errors;          / *接收错误数。这是一个超集
                                   更具体的接收错误，应该是
                                   大于或等于所有的总和
                                   属性中的rx _ * _ err值。* /
    uint64_t tx_errors;          / *传输错误数。这是一个超集
                                   更具体的传输错误，应该是
                                   大于或等于所有的总和
                                   tx _ * _ err值（当前未定义。）* /

    / *端口描述属性列表-0个或多个属性* /
    p_port_stats_prop_header属性的结构[0];
```

```
};
OFP_ASSERT（sizeof（struct ofp_port_stats）== 80）；
```


duration_sec和duration_nsec字段指示端口已配置的经过时间
进入OpenFlow管道。总持续时间（以纳秒为单位）可以计算为*持续时间sec* ×
10 9 + *持续时间nsec*。需要实现以提供第二精度。更高的精度是
鼓励（如果有）。

rx_packets，tx_packets，rx_bytes和tx_bytes是基本端口计数器，用于测量要发送的数据包
通过港口。

属性字段是端口统计信息属性的列表，其中包括特定于端口类型的统计信息。

当前定义的端口描述属性类型的列表是：

```
/ *端口统计信息属性类型。
  * /
of_p_port_stats_prop_type的枚举{
    OFPPSPT_ETHERNET            = 0，          / *以太网属性。* /
    OFPPSPT_OPTICAL             = 1            / *光学性质。* /
    OFPPSPT_EXPERIMENTER        = 0xFFFF，/ *实验者属性。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有端口统计信息属性的公共头。* /
struct ofp_port_stats_prop_header {
    uint16_t                类型;       / * OFPPSPT_ *之一。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_port_stats_prop_header的结构）== 4）；
```

OFPPSPT_ETHERNET属性使用以下结构和字段：

```
/ *以太网端口统计信息属性。* /
struct ofp_port_stats_prop_ethernet {
    uint16_t                类型;       / * OFPPSPT_ETHERNET。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
    uint8_t                 垫[4]; / *对齐64位。* /

    uint64_t rx_frame_err;              / *帧对齐错误数。* /
    uint64_t rx_over_err;               / * RX溢出的数据包数。* /
    uint64_t rx_crc_err;                / * CRC错误数。* /
    uint64_t碰撞;                       / *碰撞次数。* /
};
OFP_ASSERT（sizeof（p_port_stats_prop_ethernet的结构）== 40）；
```

OFPPSPT_OPTICAL属性使用以下结构和字段：

```
/ *光端口统计信息属性。* /
struct ofp_port_stats_prop_optical {
    uint16_t                类型;       / * OFPPSPT_OPTICAL。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
    uint8_t                 垫[4]; / *对齐64位。* /
```



```
    uint32_t标志;                      / *端口启用的功能。* /
    uint32_t tx_freq_lmda;              / *当前TX频率/波长* /
    uint32_t tx_offset;                 / * TX偏移* /
    uint32_t tx_grid_span;              / * TX网格间距* /
    uint32_t rx_freq_lmda;              / *当前RX频率/波长* /
    uint32_t rx_offset;                 / *接收偏移* /
    uint32_t rx_grid_span;              / * RX网格间距* /
```

```
    uint16_t tx_pwr;                / *当前发射功率* /
    uint16_t rx_pwr;                / *当前接收功率* /
    uint16_t bias_current;          / * TX偏置电流* /
    uint16_t温度;                   / * TX激光温度* /
};
OFP_ASSERT（sizeof（p_port_stats_prop_optical的结构）== 44）;
```

标志是一个位图，指示哪些统计值有效，并且可以包括组合
以下标志之一：

```
/ *标志是以下OFPOSF_之一* /
of_p_port_stats_optical_flags的枚举{
    OFPOSF_RX_TUNE    = 1 << 0，/ *接收机调谐信息有效* /
    OFPOSF_TX_TUNE    = 1 << 1，/ *发送调谐信息有效* /
    OFPOSF_TX_PWR     = 1 << 2，/ * TX功率有效* /
    OFPOSF_RX_PWR     = 1 << 4，/ * RX功率有效* /
    OFPOSF_TX_BIAS    = 1 << 5，/ *发送偏差有效* /
    OFPOSF_TX_TEMP    = 1 << 6，/ * TX Temp有效* /
};
```

端口调整值的表示方式如端口说明所述（请参见7.2。1）。频率
值以MHz为单位，波长（λ）以nm * 100为单位。tx_pwr和rx_pwr为dBm * 10。

bias_current为mA * 10，可用作激光效率参考。激光温度为度
C * 10。

OFPPSPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者端口统计信息属性。* /
struct ofp_port_stats_prop_experimenter {
    uint16_t                类型;         / * OFPPSPT_EXPERIMENTER。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
    uint32_t                实验者 / *采用相同的实验者ID
                                          形式如结构
                                          ofp_experimenter_header。* /
    uint32_t                exp_type;           / *实验者定义。* /
    /* 其次是：
     *     -确切地（长度-12个）字节包含实验者数据，然后
     *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *        全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（结构的p_port_stats_prop_experimenter）== 12）;
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.3.5.6端口说明

端口描述请求OFPMP_PORT_DESCRIPTION使控制器能够获得对
OpenFlow交换机的所有标准端口（请参阅4.2）。请求正文包含以下内容
结构体：

```
/ *类型为OFPMP_PORT_STATS的ofp_multipart_request的主体，
  * OFPMP_PORT_DESC。* /
struct ofp_port_multipart_request {
    uint32_t port_no;               / * OFPMP_PORT消息必须请求统计信息
                                      *单个端口（在
                                      * port_no）或所有端口（如果port_no ==
                                      * OFPP_ANY）。* /
    uint8_t pad [4];
};
OFP_ASSERT（sizeof（p_port_multipart_request的结构）== 8）;
```

port_no字段指定交换机上的有效端口，或者可以将其设置为OFPP_ANY来引用交换机上的所有端口。
开关。

答复正文由以下数组组成：

```
/ *端口说明* /
struct ofp_port {
    uint32_t port_no;
    uint16_t长度;
    uint8_t pad [2];
```

```
        uint8_t hw_addr [OFP_ETH_ALEN];
        uint8_t pad2 [2];                                   / *对齐64位。* /
        字符名称[OFP_MAX_PORT_NAME_LEN]; / *空终止* /

        uint32_t配置;                       / * OFPPC_ *标志的位图。* /
        uint32_t状态;                       / * OFPPS_ *标志的位图。* /

        / *端口描述属性列表-0个或多个属性* /
        p_port_desc_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_port）== 40）;
```

该结构是描述端口的通用端口结构（请参见第7.2.1节），包括端口
编号，端口配置和端口状态。与端口类型相关的所有端口属性必须包含在
端口说明。例如，以太网类型的物理端口必须包含以太网属性
在其说明中，逻辑端口不需要包含它。

端口描述答复必须包括OpenFlow交换机中定义的所有标准端口，或者
不管其状态或配置如何。建议标准清单
端口不应动态更改，而应仅是由于网络拓扑配置或
交换机配置，例如使用OpenFlow配置协议（请参见4.6）。

7.3.5.7队列统计

OFPMP_QUEUE_STATS多部分请求消息提供一个或多个端口的队列统计信息，以及
一个或多个队列。请求主体包含一个port_no字段，该字段标识用于
请求哪个统计信息，或者使用OFPP_ANY引用所有端口。queue_id字段标识以下内容之一
优先级队列，或者称为OFPQ_ALL，以引用在指定端口配置的所有队列。OFPQ_ALL是
0xffffffff。

```
/ *类型为OFPMP_QUEUE_DESC的ofp_multipart_request的主体
  * OFPMP_QUEUE_STATS。* /
struct ofp_queue_multipart_request {
        uint32_t port_no;                   / *所有端口（如果为OFPP_ANY）。* /
        uint32_t queue_id;                  / *如果是OFPQ_ALL，则所有队列。* /
};
OFP_ASSERT（sizeof（p_queue_multipart_request的结构）== 8）;
```

答复的正文由以下结构的数组组成：

```
/ *对OFPMP_QUEUE_STATS请求的回复正文。* /
struct ofp_queue_stats {
        uint16_t长度;                       / *此项的长度。* /
        uint8_t pad [6];                    / *对齐64位。* /
        uint32_t port_no;                   / *队列连接到的端口。* /
        uint32_t queue_id;                  / *队列ID * /
        uint64_t tx_bytes;                  / *传输的字节数。* /
        uint64_t tx_packets;                / *传输的数据包数。* /
        uint64_t tx_errors;                 / *由于溢出而丢弃的数据包数量。* /
        uint32_t duration_sec;              / *时间队列以秒为单位还活着。* /
        uint32_t duration_nsec; / *时间队列已经超过了十亿分之一秒
                                              duration_sec。* /

        p_queue_stats_prop_header属性的结构[0]; / *属性列表。* /
};
OFP_ASSERT（sizeof（struct ofp_queue_stats）== 48）;
```

duration_sec和duration_nsec字段指示队列已安装在其中的经过时间
开关。总持续时间（以纳秒为单位）可以计算为*持续时间sec* ×10 9 + *持续时间nsec*。
需要实现以提供第二精度。鼓励在可能的情况下提高精度-
能够。

属性字段是队列统计信息属性的列表。

当前定义的队列统计信息属性类型的列表是：

```
ofp_queue_stats_prop_type的枚举{
        OFPQSPT_EXPERIMENTER = 0xffff / *实验者定义的属性。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：


```
/ *所有队列属性的通用标头* /
struct ofp_queue_stats_prop_header {
        uint16_t                类型;          / * OFPQSPT_ *之一。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（struct ofp_queue_stats_prop_header）== 4）;
```

OFPQSPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者队列属性描述。* /
struct ofp_queue_stats_prop_experimenter {
        uint16_t                类型;          / * OFPQSPT_EXPERIMENTER。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        uint32_t                实验者 / *采用相同的实验者ID
                                                形式如结构
                                                ofp_experimenter_header。* /
        uint32_t                exp_type;               / *实验者定义。* /
        /* 其次是：
         *      -确切地（长度-12个）字节包含实验者数据，然后
         *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
         *          全零字节的字节* /
        uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_queue_stats_prop_experimenter的结构）== 12）;
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.3.5.8队列说明

OpenFlow交换机通过简单的排队机制，可提供有限的服务质量支持（QoS），
无神论。

交换机可以选择将一个或多个队列附加到特定的输出端口，并且这些队列
可用于计划在该输出端口上退出数据路径的数据包。交换机上的每个队列是
由*端口号*和*队列ID*唯一标识。不同端口上的两个队列可以具有相同的队列
队列ID。根据数据包输出端口和数据包将数据包定向到队列之一
队列ID，分别使用“ *输出*”操作和“ *设置队列*”操作进行*设置*。

映射到特定队列的数据包将根据该队列的配置（例如最小
率）。在所有管道处理之后，逻辑上进行队列处理。使用队列进行数据包调度
不是由本规范定义的，并且取决于开关，特别是*队列ID*之间没有优先级
假设。

队列配置是通过命令行在OpenFlow交换协议之外进行的
工具或通过外部专用配置协议。控制器可以查询开关以获取
使用OFPMP_QUEUE_DESC多部分请求在端口上配置的队列。

OFPMP_QUEUE_DESC多部分请求消息提供一个或多个端口的队列描述
和一个或多个队列。请求主体包含一个port_no字段，该字段标识用于
请求哪些描述，或者使用OFPP_ANY来引用所有端口。queue_id字段标识一个

优先级队列，或OFPQ_ALL表示在指定端口配置的所有队列。OFPQ_ALL是
0xffffffff。

```
/ *类型为OFPMP_QUEUE_DESC的ofp_multipart_request的主体
```

```
 * OFPMP_QUEUE_STATS。* /
struct ofp_queue_multipart_request {
    uint32_t port_no;                   / *所有端口（如果为OFPP_ANY）。* /
    uint32_t queue_id;                  / *如果是OFPQ_ALL，则所有队列。* /
};
OFP_ASSERT（sizeof（p_queue_multipart_request的结构）== 8）；
```

答复的正文由以下结构的数组组成：

```
/ *对OFPMP_QUEUE_DESC请求的回复正文。* /
struct ofp_queue_desc {
    uint32_t port_no;                   / *此队列连接到的端口。* /
    uint32_t queue_id;                  / *特定队列的ID。* /
    uint16_t len;                       / *此队列desc的字节长度。* /
    uint8_t pad [6];                    / * 64位对齐。* /

    p_queue_desc_prop_header属性的结构[0]; / *属性列表。* /
};
OFP_ASSERT（sizeof（p_queue_desc的结构）== 16）；
```

属性字段是队列描述属性的列表。

当前定义的队列描述属性类型的列表是：

```
of_p_queue_desc_prop_type的枚举{
    OFPQDPT_MIN_RATE        = 1         / *保证最低数据速率。* /
    OFPQDPT_MAX_RATE        = 2         / *最大数据速率。* /
    OFPQDPT_EXPERIMENTER = 0xffff / *实验者定义的属性。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有队列属性的通用标头* /
struct ofp_queue_desc_prop_header {
    uint16_t            类型;       / * OFPQDPT_ *之一。* /
    uint16_t            长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_queue_desc_prop_header的结构== 4）；
```

OFPQDPT_MIN_RATE属性使用以下结构和字段：

```
/ *最小速率队列属性描述。* /
struct ofp_queue_desc_prop_min_rate {
    uint16_t            类型;       / * OFPQDPT_MIN_RATE。* /
    uint16_t            长度; / *长度为8。* /
    uint16_t率;                 / * 1％的百分比；> 1000->禁用。* /
    uint8_t pad [2];            / * 64位对齐* /
};
OFP_ASSERT（sizeof（struct ofp_queue_desc_prop_min_rate）== 8）；
```


速率字段是保证队列的最小速率，表示为当前速度的一部分
队列所连接到的输出端口的大小（请参见7.2.1。2），以1/1000的增量进行编码。如果率是
未配置，则设置为OFPQ_MIN_RATE_UNCFG，即0xffff。

OFPQDPT_MAX_RATE属性使用以下结构和字段：

```
/ *最大速率队列属性描述。* /
struct ofp_queue_desc_prop_max_rate {
    uint16_t            类型;       / * OFPQDPT_MAX_RATE。* /
    uint16_t            长度; / *长度为8。* /
    uint16_t率;                 / * 1％的百分比；> 1000->禁用。* /
    uint8_t pad [2];            / * 64位对齐* /
};
OFP_ASSERT（sizeof（struct ofp_queue_desc_prop_max_rate）== 8）；
```

速率字段是队列可以使用的最大速率，表示为当前速度的一部分
队列所连接到的输出端口的大小（请参见7.2.1。2），以1/1000的增量进行编码。如果率是
未配置，则设置为OFPQ_MAX_RATE_UNCFG，即0xffff。

OFPQDPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者队列属性描述。* /
struct ofp_queue_desc_prop_experimenter {
    uint16_t            类型;       / * OFPQDPT_EXPERIMENTER。* /
    uint16_t            长度; / *此属性的长度（以字节为单位）。* /
```

```
    uint32_t                实验者 / *采用相同的实验者ID
                                                形式如结构
                                                ofp_experimenter_header。* /
    uint32_t                exp_type;           / *实验者定义。* /
    /* 其次是:
     *     -确切地（长度-12个）字节包含实验者数据，然后
     *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_queue_desc_prop_experimenter的结构）== 12）;
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.3.5.9组统计

OFPMP_GROUP_STATS多部分请求消息提供一个或多个组的统计信息。的
请求主体由以下结构组成:

```
/ * OFPMP_GROUP_STATS和OFPMP_GROUP_DESC请求的主体。* /
p_group_multipart_request的结构{
    uint32_t group_id;          / *所有组（如果为OFPG_ALL）。* /
    uint8_t pad [4];            / *对齐64位。* /
};
OFP_ASSERT（sizeof（p_group_multipart_request的结构）== 8）;
```


group_id字段指定交换机上的有效组，也可以设置为OFPG_ALL来引用所有组
在开关上。

答复的正文由以下结构的数组组成:

```
/ *对OFPMP_GROUP_STATS请求的回复正文。* /
struct ofp_group_stats {
    uint16_t长度;               / *此项的长度。* /
    uint8_t pad [2];            / *对齐64位。* /
    uint32_t group_id;          / *组标识符。* /
    uint32_t ref_count;         / *直接转发的流或组的数量
                                  到这个小组。* /
    uint8_t pad2 [4];           / *对齐64位。* /
    uint64_t packet_count;      / *按组处理的数据包数。* /
    uint64_t byte_count;        / *按组处理的字节数。* /
    uint32_t duration_sec;      / *时间组仅以秒为单位。* /
    uint32_t duration_nsec; / *时间组的生存时间超过了十亿分之一秒
                                  duration_sec。* /
    p_bucket_counter bucket_stats [0]; / *每个存储桶设置一个计数器。* /
};
OFP_ASSERT（sizeof（struct ofp_group_stats）== 40）;
```

字段包括引用该组的group_id，ref_count字段计算流数
直接引用组，packet_count和byte_count字段计数的条目或组
该组处理的所有数据包。

duration_sec和duration_nsec字段指示该组已安装的经过时间
开关。总持续时间（以纳秒为单位）可以计算为*持续时间sec* ×10 9 + *持续时间nsec*。
需要实现以提供第二精度。鼓励在可能的情况下提高精度-
能够。

bucket_stats字段由ofp_bucket_counter结构数组组成:

```
/ *用于群组统计信息回复。* /
struct ofp_bucket_counter {
    uint64_t packet_count;      / *桶处理的数据包数。* /
    uint64_t byte_count;        / *存储桶处理的字节数。* /
};
OFP_ASSERT（sizeof（p_bucket_counter的结构）== 16）;
```

7.3.5.10组说明

OFPMP_GROUP_DESC多部分请求消息提供了一种列出交换机上的组集合的方法，
以及相应的存储桶操作。请求主体由以下结构组成:

```
/*OFPMP_GROUP_STATS和OFPMP_GROUP_DESC请求的主体。* /
p_group_multipart_request的结构{
        uint32_t group_id;                  / *所有组（如果为OFPG_ALL）。* /
        uint8_t pad [4];                    / *对齐64位。* /
};
OFP_ASSERT（sizeof（p_group_multipart_request的结构）== 8）；
```


group_id字段指定交换机上的有效组，也可以设置为OFPG_ALL来引用所有组
在开关上。

答复的正文由以下结构的数组组成：

```
/ *对OFPMP_GROUP_DESC请求的回复正文。* /
struct ofp_group_desc {
        uint16_t长度；                      / *此项的长度。* /
        uint8_t类型；                       / * OFPGT_ *之一。* /
        uint8_t pad;                        / *填充到64位。* /
        uint32_t group_id;                  / *组标识符。* /
        uint16_t bucket_array_len;          / *操作存储区数据的长度。* /
        uint8_t pad2 [6];                   / *填充到64位。* /
        /* 其次是：
         *      -确切的“ bucket_array_len”字节包含一个数组
         *         p_bucket结构。
         *      -零个或多个字节的组属性可填充整体
         *         长度字段中的长度。* /
        p_bucket桶的结构[0];                    / *桶清单-0或更多。* /
        // p_group_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_group_desc）== 16）；
```

用于组描述的字段与用于ofp_group_mod结构的字段相同（请参见7 .3.4.3）。

7.3.5.11组功能

OFPMP_GROUP_FEATURES多部分请求消息提供了一种列出组功能的方法
在开关上。请求正文为空，而回复正文具有以下结构：

```
/ *回复OFPMP_GROUP_FEATURES请求的正文。组功能。* /
struct ofp_group_features {
        uint32_t类型；                      / *支持（1 << OFPGT_ *）个值的位图。* /
        uint32_t功能；                      / *支持OFPGFC_ *功能的位图。* /
        uint32_t max_groups [4];            / *每种类型的最大组数。* /
        uint32_t动作[4];                    / *支持（1 << OFPAT_ *）个值的位图。* /
};
OFP_ASSERT（sizeof（struct ofp_group_features）== 40）；
```

类型字段是交换机支持的组类型的位图。位掩码使用来自的值

ofp_group_type作为相关组类型向左移的位数。实验者类型
应该不会通过该位掩码进行报告。例如，OFPGT_ALL将使用掩码0x00000001。

功能字段是使用以下标志的组合的位图：

```
/ *组配置标志* /
ofp_group_capabilities枚举{
        OFPGFC_SELECT_WEIGHT    = 1 << 0，/ *选定组的支撑重量* /
        OFPGFC_SELECT_LIVENESS = 1 << 1，/ *支持选定组的活动* /
        OFPGFC_CHAINING         = 1 << 2，/ *支持链接组* /
        OFPGFC_CHAINING_CHECKS = 1 << 3，/ *检查链接的循环并删除* /
};
```


max_groups字段是每种组类型的最大组数。

动作字段是每种组类型支持的一组动作位图。第一个位图适用

到OFPGT_ALL组类型。位掩码使用ofp_action_type中的值作为
位左移以执行相关操作。实验者的行为应该不通过这个位掩码进行报告。
例如，OFPAT_OUTPUT将使用掩码0x00000001。

7.3.5.12仪表统计

OFPMP_METER_STATS统计信息请求消息提供一个或多个仪表的统计信息。的
请求主体包含meter_id字段，可以将其设置为OFPM_ALL来引用
开关。

```
/ * OFPMP_METER_STATS和OFPMP_METER_DESC请求的主体。* /
struct ofp_meter_multipart_request {
        uint32_t meter_id;                  / *仪表实例，或OFPM_ALL。* /
        uint8_t pad [4];                    / *对齐64位。* /
};
OFP_ASSERT（sizeof（p_meter_multipart_request的结构）== 8）；
```

答复的正文由以下结构的数组组成：

```
/ *对OFPMP_METER_STATS请求的回复正文。仪表统计。* /
struct ofp_meter_stats {
        uint32_t                meter_id;                   / *仪表实例。* /
        uint16_t                len;                        / *此统计信息的长度（以字节为单位）。* /
        uint8_t                 垫[6];
        uint32_t                ref_count;                  / *的流或组的数量
                                                              直接参考此仪表。* /
        uint64_t                packet_in_count; / *输入中的包数。* /
        uint64_t                byte_in_count;              / *输入的字节数。* /
        uint32_t        duration_sec; / *计时器以秒为单位处于活动状态。* /
        uint32_t        duration_nsec; / *计时器的活动时间超过十亿分之一秒
                                                    duration_sec。* /
        p_meter_band_stats的结构band_stats [0]; / * band_stats的长度为
                                                                从长度字段推断。* /
};
OFP_ASSERT（sizeof（p_meter_stats的结构）== 40）；
```

packet_in_count和byte_in_count字段对仪表所处理的所有数据包进行计数。的

ref_count字段计算直接引用仪表的流量条目或组的数量。

duration_sec和duration_nsec字段指示仪表已安装的经过时间
开关。总持续时间（以纳秒为单位）可以计算为*持续时间sec* ×10 9 + *持续时间nsec*。
需要实现以提供第二精度。鼓励在可能的情况下提高精度-
能够。

band_stats字段由ofp_meter_band_stats结构数组组成：


```
/ *每个米波段的统计数据* /
struct ofp_meter_band_stats {
        uint64_t                packet_band_count;          / *频段中的数据包数。* /
        uint64_t                byte_band_count;            / * band中的字节数。* /
};
OFP_ASSERT（sizeof（p_meter_band_stats的结构）== 16）；
```

packet_band_count和byte_band_count字段计算该频段处理的所有数据包。

频段统计信息的顺序必须与OFPMP_METER_DESC多部分答复中的顺序相同。的
仪表默认频段（速率0）不能包含在该列表中。

7.3.5.13仪表说明

OFPMP_METER_DESC多部分请求消息提供了一个或多个仪表的配置。的
请求主体包含meter_id字段，可以将其设置为OFPM_ALL来引用
开关。

```
/ * OFPMP_METER_STATS和OFPMP_METER_DESC请求的主体。* /
struct ofp_meter_multipart_request {
        uint32_t meter_id;                  / *仪表实例，或OFPM_ALL。* /
        uint8_t pad [4];                    / *对齐64位。* /
```

```
};
OFP_ASSERT（sizeof（p_meter_multipart_request的结构）== 8）；
```

答复的正文由以下结构的数组组成：

```
/ *对OFPMP_METER_DESC请求的回复正文。仪表配置。* /
struct ofp_meter_desc {
        uint16_t                长度;                   / *此项的长度。* /
        uint16_t                标志;                   / *所有适用的OFPMF_ *。* /
        uint32_t                meter_id;               / *仪表实例。* /
        struct ofp_meter_band_header bands [0]; / *波段长度为
                                                        从长度字段推断。* /
};
OFP_ASSERT（sizeof（p_meter_desc的结构）== 8）；
```

这些字段与用于配置电表的字段相同（请参见7.3.4.5。5）。仪表默认频段（速率0）不能包含在每个仪表的频段列表中。

如果未设置仪表的OFPMF_BURST标志，并且仪表使用内部突发值，则仪表应该使用相应的内部突发值设置每个频段的burst_size字段。

7.3.5.14仪表功能

OFPMP_METER_FEATURES多部分请求消息提供了计量的功能集子系统。请求正文为空，回复的正文由以下结构组成：

```
/ *回复OFPMP_METER_FEATURES请求的正文。仪表功能。* /
struct ofp_meter_features {
        uint32_t        max_meter;          / *最大米数。* /
        uint32_t        band_types;         / *支持（1 << OFPMBT_ *）个值的位图。* /
        uint32_t        能力; / *“ ofp_meter_flags”的位图。* /
        uint8_t         max_bands;          / *每米的最大频段* /
        uint8_t         max_color;          / *最大色值* /
        uint8_t         垫[2];
        uint32_t        特征;               / *“ ofp_meter_feature_flags”的位图。* /
        uint8_t         pad2 [4];
};
OFP_ASSERT（sizeof（p_meter_features的结构）== 24）；
```

band_types字段是交换机支持的频段类型的位图，该交换机可能具有其他有关如何在特定仪表中组合频段类型的限制。位掩码使用值从ofp_meter_band_type开始，作为相关频段类型向左移的位数。实验M输入类型应该不通过该位掩码进行报告。例如，OFPMBT_DROP将使用掩码0x00000002。

max_bands字段是每个仪表支持的最大频带数。如果仅切换支持简单的速率限制器，该值为1。如果交换机还支持经典的两个速率警务人员，此值为2。

功能字段是使用以下标志的组合的位图：

```
/ *仪表配置标志* /
of_p_meter_flags枚举{
        OFPMF_KBPS              = 1 << 0，/ *速率值（kb / s）（每秒千比特）。* /
        OFPMF_PKTPS             = 1 << 1，/ *速率值（以包/秒为单位）。* /
        OFPMF_BURST             = 1 << 2，/ *执行突发大小。* /
        OFPMF_STATS             = 1 << 3，/ *收集统计信息。* /
};
```

features字段是使用以下标志的组合的位图：

```
/ *仪表功能标志* /
of_p_meter_feature_flags的枚举{
        OFPMFF_ACTION_SET       = 1 << 0，/ *在操作集中支持仪表操作。* /
```

```
    OFPMFF_ANY_POSITION = 1 << 1, /*支持动作列表中的多个场景。*/
};
```


7.3.5.15控制器状态多部分

OFPMP_CONTROLLER_STATUS多部分消息允许控制器请求状态，角色
以及交换机上配置的其他控制器的控制通道。请求正文为空。的
回复主体由控制器状态结构（7.2.7）的列表组成，每个配置的控制器一个。

在多部分中，每个控制器状态结构中的原因字段必须设置为OFPCSR_REQUEST
响应。

7.3.5.16表统计

使用OFPMP_TABLE_STATS多部分请求类型请求有关表的信息。要求
主体中不包含任何数据。

答复的正文由以下数组组成：

```
/*对OFPMP_TABLE_STATS请求的回复正文。*/
struct ofp_table_stats {
    uint8_t table_id;                   /*表的标识符。编号较低的表格
                                          首先咨询。*/
    uint8_t pad [3];                    /*对齐32位。*/
    uint32_t active_count;              /*活动条目数。*/
    uint64_t lookup_count;              /*在表中查找的数据包数。*/
    uint64_tmatched_count; /*命中表的数据包数量。*/
};
OFP_ASSERT（sizeof（struct ofp_table_stats）== 24）;
```

对于交换机支持的每个表，该阵列都有一个结构。条目返回到
命令数据包遍历表。

7.3.5.17表说明

OFPMP_TABLE_DESC多部分请求消息提供了一种列出当前配置的方法
使用OFPT_TABLE_MOD消息设置的交换机上的表。请求正文为空，
而回复主体是具有以下结构的数组：

```
/*对OFPMP_TABLE_DESC请求的回复正文。*/
struct ofp_table_desc {
    uint16_t长度;                       /*长度填充为64位。*/
    uint8_t table_id;                   /*表的标识符。编号较低的表格
                                          首先咨询。*/
    uint8_t pad [1];                    /*对齐32位。*/
    uint32_t配置;                       /* OFPTC_ *值的位图。*/

    /* Table Mod属性列表-0或更多。*/
    p_table_mod_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_table_desc的结构）== 8）;
```

用于表描述的字段与用于ofp_table_mod结构的字段相同（请参见7.3.4.1）。

7.3.5.18表功能

OFPMP_TABLE_FEATURES多部分类型允许控制器同时查询以下功能：
现有表，并有选择地要求交换机重新配置其表以匹配提供的配置
配给。通常，表格功能代表表格的所有可能功能，但是其中一些功能
功能可能互斥，并且当前的功能结构不允许表示
这样的排除。

7.3.5.18.1表功能请求和答复

如果OFPMP_TABLE_FEATURES请求主体为空，则开关将返回一个数组

ofp_table_features结构，包含当前配置的流表的功能。的
此操作不会更改流表和管道。

如果请求主体包含一个或多个p_table_features结构的数组，则开关将
尝试更改其流表以匹配请求的流表配置更改。支持
这样的请求是可选的，并且在使用其他协议配置表时不鼓励此类请求。
作为OpenFlow配置协议。如果不支持这些请求，则交换机必须返回
OFPET_BAD_REQUEST类型和OFPBRC_BAD_LEN代码的ofp_error_msg。如果这些要求是
禁用后，该开关必须返回OFPET_TABLE_FEATURES_FAILED类型的ofp_error_msg，并且

OFPTFFC_EPERM代码。

包含一组ofp_table_features结构的请求可以启用，禁用，修改流表，
也可以替换整个管道，具体取决于请求中的*命令*。以下*命令*
支持：

```
/ *表功能请求命令* /
of_p_table_features_command的枚举{
    OFPTFC_REPLACE = 0，          / *替换完整的管道。* /
    OFPTFC_MODIFY = 1，           / *修改流表功能。* /
    OFPTFC_ENABLE = 2             / *在管道中启用流表。* /
    OFPTFC_DISABLE = 3，          / *禁用管道中的流表。* /
};
```

命令OFPTFC_REPLACE配置整个管道以及管道中的流表集
必须与请求中的设置匹配，否则必须返回错误（请参见7 .5.4.14）。所有流量的能力
要求中包含的表格必须更新。特别是，如果请求的配置没有
对于交换机支持的一个或多个流表，包含一个ofp_table_features结构，这些
如果成功设置了配置，将从管道中删除流表。

命令OFPTFC_MODIFY仅配置请求中指定的流表集，其他流
未包含在请求中的表保持不变，并且它们在管道中的存在保持不变。的
请求中包含的流表的功能必须更新。此命令可用于
修改单个流表的功能。如果请求中的流表未包含在
管道，它仍然排除在外。

命令OFPTFC_ENABLE在管道中插入请求中指定的流表。流
表根据其*表ID*号插入到管道中。流程图的功能
请求中包含的内容不变，并且不得更新。

命令OFPTFC_DISABLE从管道中删除请求中指定的流表。的
请求中包含的流表的功能不变，因此不能更新。

成功的配置更改将配置请求中的所有流表，即所有
请求中指定的流表被修改或不修改。如果该命令修改了流功能
表，每个流表的新功能必须是所请求的超集或等于所请求的
能力。如果请求中包含的表功能的属性列表为空，则
该流表的属性不变，并且仅更新该流表的其他功能。
如果流表配置成功，则流表中已删除或流的流条目
在先前配置和新配置之间功能发生变化的表将从以下位置删除
流表，但是没有发送ofp_flow_removed消息。交换机然后回复新的
组态。

交换机必须能够完整设置请求的配置，或者返回错误消息
（请参阅7.5.4.14）。交换机可能不支持上面列出的所有命令（请参阅7.5.4.14）。对
与OFPTFC_REPLACE相比，该交换机在每个请求中可能支持有限数量的流表（请参
7.5.4.14）。

包含ofp_table_features的请求和答复应满足以下最低要求
要求：

•在单个table_features结构中指定的table_id字段值应唯一
在消息中的所有p_table_features结构中。

•每个p_table_features结构中包含的属性字段必须为空或
必须完全包含ofp_table_feature_prop_type每个属性之一，其中两个
例外。首先，如果带有_MISS后缀的属性与相应的-
常规流条目的sponding属性。其次，类型为OFPTFPT_EXPERIMENTER的属性和
OFPTFPT_EXPERIMENTER_MISS可能被省略或包含很多次。订货不明，
但是鼓励实施人员使用规范中列出的顺序（请参见7 .3.5.18.2）。

收到不满足这些要求的请求的交换机应返回错误类型
OFPET_TABLE_FEATURES_FAILED及其相应的错误代码（请参阅7 .5.4.14）。

以下结构描述了表功能主体的请求和答复：

```
/ *类型为OFPMP_TABLE_FEATURES的ofp_multipart_request的正文。/
  *回复OFPMP_TABLE_FEATURES请求的正文。* /
struct ofp_table_features {
        uint16_t长度；                  / *长度填充为64位。* /
        uint8_t table_id;               / *表的标识符。编号较低的表格
                                            首先咨询。* /
        uint8_t命令；                   / * OFPTFC_ *之一。* /
        uint32_t功能；                  / * OFPTFF_ *值的位图。* /
        字符名称[OFP_MAX_TABLE_NAME_LEN];
        uint64_t metadata_match; / *元数据表的位可以匹配。* /
        uint64_t metadata_write; / *元数据表的位可以写入。* /
        uint32_t功能；                  / * OFPTC_ *值的位图。* /
        uint32_t max_entries;           / *支持的最大条目数。* /

        / *表功能属性列表* /
        p_table_feature_prop_header属性的结构[0]; / *属性列表* /
};
```



```
OFP_ASSERT（sizeof（struct ofp_table_features）== 64）；
```

OFPMP_TABLE_FEATURES回复包含一个或多个ofp_table_features结构的数组，
交换机支持的每个流表的一种结构。条目始终按顺序返回
数据包遍历流表。

命令字段是请求的配置操作（请参见上文）。仅命令字段来自
首先使用了p_table_features条目，它适用于整个请求。来自的命令字段
随后的ofp_table_features条目将被忽略。在表格功能回复中，此字段为保留字段，
应该设置为0（请参阅7.1.3）。

名称字段是一个空终止的字符串，其中包含流表的可读名称。的
OFP_MAX_TABLE_NAME_LEN的值为32。

metadata_match字段指示流表可以匹配的元数据字段的位，
当使用struct ofp_match的元数据字段时。值0xFFFFFFFFFFFFFFFF表示
流表可以匹配完整的元数据字段。

metadata_write字段指示流表可以使用写入的元数据字段的位
OFPIT_WRITE_METADATA指令。值0xFFFFFFFFFFFFFFFF表示流表
可以写完整的元数据字段。

功能字段是流表配置中支持的配置标志集
消息（请参见7 .3.4.1）。

max_entries字段描述可插入该流的最大流条目数
表。由于现代硬件的限制，应考虑使用max_entries值
咨询和尽力而为的流量表估算。尽管高层
流表的抽象，实际上单个流条目消耗的资源不是恒定的。对于
例如，一个流条目可能会消耗多个条目，具体取决于其匹配参数（例如，
IPv4与IPv6）。另外，在OpenFlow级别上看起来不同的流表实际上可能共享相同的流表。
基础物理资源。此外，在OpenFlow混合交换机上，这些流表可以共享
具有非OpenFlow功能。结果是交换实施者应该报告一个近似值
支持的总流条目中，控制器编写者不应将此值视为固定的物理值
不变。

可能会要求控制器更改ofp_table_features中的所有字段，但以下情况除外
max_entries字段的值是只读的，由交换机返回。

features字段是位图，用于定义流表的使用方式及其基本原理
特征。它可能包含以下标志的组合：

```
/ *表支持的功能标志。* /
of_p_table_feature_flag的枚举{
    OFPTFF_INGRESS_TABLE        = 1 << 0，/ *可以配置为入口表。* /
    OFPTFF_EGRESS_TABLE         = 1 << 1，/ *可以配置为出口表。* /
    OFPTFF_FIRST_EGRESS         = 1 << 4，/ *是第一个出口表。* /
};
```


如果设置了标志OFPTFF_INGRESS_TABLE，则表明可以将流表配置为入口流量表（请参阅5.12）。流程表0必须设置此标志，即流程表0始终可以用作入口流表。通常无法通过表功能请求更改此标志。

如果设置了标志OFPTFF_EGRESS_TABLE，则表明可以将流表配置为出口流表（请参阅5.12）。流表0必须未设置该标志，即表0永远不能用作出口流表。通常无法通过表功能请求更改此标志。

如果设置了标志OFPTFF_FIRST_EGRESS，则表示此流表是第一个出口表：数据包输出到端口，出口处理将从该流表开始（请参阅5.12）。在表格中回复，只有单个流表可以设置此标志。仅在流表中可以在流表上设置此标志该表可以作为出口表，即设置了OFPTFF_EGRESS_TABLE标志。当旗

在流表上设置OFPTFF_FIRST_EGRESS以使其成为第一个出口表，该流表先前如果用作第一出口表，则必须清除其对应的标志。如果有多个流表此标志在请求中设置，仅设置最后一个。如果没有流表具有OFPTFF_FIRST_EGRESS标志置位，不使用出口表：当数据包输出到端口时，它直接由输出处理港口。

属性字段是表要素属性的列表，描述了流程的各种功能表。

7.3.5.18.2表功能属性

当前定义的表要素属性类型的列表为：

```
/ *表要素属性类型。
  *低位清零表示常规流条目的属性。
  *低位设置指示表丢失流条目的属性。
  * /
of_p_table_feature_prop_type的枚举{
    OFPTFPT_INSTRUCTIONS                = 0，/ *指令属性。* /
    OFPTFPT_INSTRUCTIONS_MISS           = 1，/ *有关表缺失的说明。* /
    OFPTFPT_NEXT_TABLES                 = 2，/ *下一表属性。* /
    OFPTFPT_NEXT_TABLES_MISS            = 3，/ *下一个表缺失表。* /
    OFPTFPT_WRITE_ACTIONS               = 4，/ *写动作属性。* /
    OFPTFPT_WRITE_ACTIONS_MISS          = 5，/ *表缺失的写操作。* /
    OFPTFPT_APPLY_ACTIONS               = 6，/ * Apply Actions属性。* /
    OFPTFPT_APPLY_ACTIONS_MISS          = 7，/ *为表缺失应用动作。* /
    OFPTFPT_MATCH                       = 8，/ *匹配属性。* /
    OFPTFPT_WILDCARDS                   = 10，/ *通配符属性。* /
    OFPTFPT_WRITE_SETFIELD              = 12，/ *写入Set-Field属性。* /
    OFPTFPT_WRITE_SETFIELD_MISS         = 13，/ *写表缺失的Set-Field。* /
    OFPTFPT_APPLY_SETFIELD              = 14，/ *应用设置字段属性。* /
    OFPTFPT_APPLY_SETFIELD_MISS         = 15，/ *为表缺失应用Set-Field。* /
    OFPTFPT_TABLE_SYNC_FROM             = 16，/ *表同步属性。* /
    OFPTFPT_WRITE_COPYFIELD             = 18，/ *写入Copy-Field属性。* /
    OFPTFPT_WRITE_COPYFIELD_MISS        = 19，/ *写表缺失的复制字段。* /
    OFPTFPT_APPLY_COPYFIELD             = 20，/ *应用复制字段属性。* /
    OFPTFPT_APPLY_COPYFIELD_MISS        = 21，/ *为表缺失应用复制字段。* /
    OFPTFPT_PACKET_TYPES                = 22，/ *数据包类型属性。* /
    OFPTFPT_EXPERIMENTER                = 0xFFFE，/ *实验者属性。* /
    OFPTFPT_EXPERIMENTER_MISS           = 0xFFFF，/ *表丢失的实验器。* /
};
```

带有_MISS后缀的属性描述了表缺失流条目的功能（请参见5.4），
而其他属性描述了常规流输入的功能。如果特定属性不
具有任何功能（例如，不支持Set-Field），必须包括一个具有空列表的属性
在属性列表中。当表缺失流条目的属性与对应的表项相同时
常规流条目的属性（即，两个属性具有相同的功能列表），此表缺失
属性可以从属性列表中省略。

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有表功能属性的公用头* /
struct ofp_table_feature_prop_header {
        uint16_t                类型;         / * OFPTFPT_ *之一。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_table_feature_prop_header的结构）== 4）;
```

OFPTFPT_INSTRUCTIONS和OFPTFPT_INSTRUCTIONS_MISS属性使用以下结构
和字段：

```
/ *说明属性* /
struct ofp_table_feature_prop_instructions {
        uint16_t                类型;         / * OFPTFPT_INSTRUCTIONS之一，
                                                OFPTFPT_INSTRUCTIONS_MISS。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        /* 其次是：
          *      -恰好（长度-4）个字节，包含指令ID，然后
          *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
          *          全零字节的字节* /
        struct ofp_instruction_idstruction_ids [0];                             / *指令列表* /
};
OFP_ASSERT（sizeof（p_table_feature_prop_instructions的结构）== 4）;
```

structions_ids字段是此表支持的指令列表（请参见5.5）。的要素
该列表大小可变，可以表达实验者的说明。非实验者教学
位置是4个字节，它们使用以下结构：

```
/ *指令ID * /
struct ofp_instruction_id {
        uint16_t类型;                           / * OFPIT_ *之一。* /
        uint16_t len;                           / *长度为4或由实验人员定义。* /
        uint8_t exp_data [0];                   / *可选的实验者ID +数据。* /
};
OFP_ASSERT（sizeof（p_instruction_id的结构）== 4）;
```

OFPTFPT_NEXT_TABLES，OFPTFPT_NEXT_TABLES_MISS和OFPTFPT_TABLE_SYNC_FROM属性
使用以下结构和字段：


```
/ *下一个表和表从属性同步* /
struct ofp_table_feature_prop_tables {
        uint16_t                类型;         / * OFPTFPT_NEXT_TABLES之一，
                                                OFPTFPT_NEXT_TABLES_MISS，
                                                OFPTFPT_TABLE_SYNC_FROM。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        /* 其次是：
          *      -精确地（长度-4）个字节包含table_id，然后
          *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
          *          全零字节的字节* /
        uint8_t                 table_ids [0];                  / *表ID的列表。* /
};
OFP_ASSERT（sizeof（p_table_feature_prop_tables的结构== 4）;
```

对于OFPTFPT_NEXT_TABLES和OFPTFPT_NEXT_TABLES_MISS，table_ids字段是表的数组
可以使用OFPIT_GOTO_TABLE指令从当前表直接访问（请参见5.1）。

对于OFPTFPT_TABLE_SYNC_FROM，table_ids字段是当前表正在同步的表的数组-
修改内容（请参见6. 6）。在流之一中添加，修改或删除流条目时
阵列中列出的表格中，将在以下位置自动添加，修改或删除相应的流条目

当前流表。
的 OFPTFPT_WRITE_ACTIONS、 OFPTFPT_WRITE_ACTIONS_MISS、 OFPTFPT_APPLY_ACTIONS 和
OFPTFPT_APPLY_ACTIONS_MISS属性使用以下结构和字段：

```
/ *动作属性* /
struct ofp_table_feature_prop_actions {
        uint16_t                类型;         / * OFPTFPT_WRITE_ACTIONS之一，
                                                OFPTFPT_WRITE_ACTIONS_MISS、
                                                OFPTFPT_APPLY_ACTIONS、
                                                OFPTFPT_APPLY_ACTIONS_MISS。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        /* 其次是：
          *      -恰好（length-4）个字节包含action_id，然后
          *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
          *          全零字节的字节* /
        struct ofp_action_id action_ids [0];                    / *动作清单* /
};
OFP_ASSERT（sizeof（p_table_feature_prop_actions的结构）== 4）；
```

OFPTFPT_WRITE_ACTIONS和OFPTFPT_WRITE_ACTIONS_MISS属性描述了支持以下操作：
由表使用OFPIT_WRITE_ACTIONS指令移植，而OFPTFPT_APPLY_ACTIONS
和OFPTFPT_APPLY_ACTIONS_MISS属性使用

OFPIT_APPLY_ACTIONS指令。

action_ids字段表示功能支持的所有操作（请参见5.8）。的要素
该列表的大小可变，可以表达实验者的动作，非实验者的动作是4
个字节，它们使用以下结构：

```
/ *操作ID * /
struct ofp_action_id {
        uint16_t类型;                   / * OFPAT_ *之一。* /
        uint16_t len;                   / *长度为4或由实验人员定义。* /
```



```
        uint8_t exp_data [0];           / *可选的实验者ID +数据。* /
};
OFP_ASSERT（sizeof（struct ofp_action_id）== 4）；
```

的 OFPTFPT_MATCH、 OFPTFPT_WILDCARDS、 OFPTFPT_WRITE_SETFIELD、
OFPTFPT_WRITE_SETFIELD_MISS、 OFPTFPT_APPLY_SETFIELD、 OFPTFPT_APPLY_SETFIELD_MISS、
OFPTFPT_WRITE_COPYFIELD、 OFPTFPT_WRITE_COPYFIELD_MISS、 OFPTFPT_APPLY_COPYFIELD 和
OFPTFPT_APPLY_COPYFIELD_MISS属性使用以下结构和字段：

```
/ *匹配，通配符或设置字段属性* /
struct ofp_table_feature_prop_oxm {
        uint16_t                类型;         / * OFPTFPT_MATCH之一，
                                                OFPTFPT_WILDCARDS、
                                                OFPTFPT_WRITE_SETFIELD、
                                                OFPTFPT_WRITE_SETFIELD_MISS、
                                                OFPTFPT_APPLY_SETFIELD、
                                                OFPTFPT_APPLY_SETFIELD_MISS、
                                                OFPTFPT_WRITE_COPYFIELD、
                                                OFPTFPT_WRITE_COPYFIELD_MISS、
                                                OFPTFPT_APPLY_COPYFIELD、
                                                OFPTFPT_APPLY_COPYFIELD_MISS。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        /* 其次是：
          *      -恰好（长度-4）个字节包含oxm_id，然后
          *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
          *          全零字节的字节* /
        uint32_t                oxm_ids [0];          / * OXM标头数组* /
};
OFP_ASSERT（sizeof（p_table_feature_prop_oxm的结构== 4）；
```

oxm_ids字段是功能部件的OXM类型的列表（请参见7 .2.3.2）。该列表的元素是
非实验者OXM字段的32位OXM头或实验者OXM的64位OXM头
字段，那些OXM字段不包含任何有效负载。OXM标头中的oxm_length字段必须为
为OXM字段定义的长度值，即OXM字段具有有效负载的有效负载长度。
对于有效负载大小可变的实验者OXM字段，oxm_length字段必须为最大
有效载荷的长度。在某些情况下，只有具有最高类型字段的数据包寄存器才需要
包含在列表中（请参阅7.2.3.10）。

OFPTFPT_MATCH属性指示特定表支持匹配的字段
开启（请参阅7.2.3。7）。例如，如果表可以匹配入口端口，则使用类型为的OXM标头

OXM OF_IN_PORT应该包含在列表中，如果OXM标头上的HASMASK位置1，则
开关必须支持给定类型的掩码。此属性中未列出的字段不能为
在表中用作匹配字段，除非它们用作另一个字段的先决条件。那个领域
只能作为先决条件在表中使用，并且不能与其他值匹配，因此不能包含在此表中
属性（请参见7.2.3。6）。例如，如果表与IP数据包的IPv4地址匹配，但不匹配
任意以太类型，该表必须在匹配中接受等于IPv4的以太类型的先决条件，
但是，该属性中不得列出Ethertype字段。另一方面，如果表格匹配
IP数据包的IPv4地址，并且可以匹配任意以太类型，必须列出以太类型字段
在这个属性中。

OFPTFPT_WILDCARDS属性指示特定表支持通配符的字段，
满足他们的先决条件时在比赛中（取消）。此属性必须是严格的子集
OFPTFPT_MATCH属性的值。例如，直接查找哈希表会将列表清空，
而TCAM或顺序搜索的表会将其设置为与OFPTFPT_MATCH相同的值
属性。如果仅当某个先决条件无效时才可以省略该字段（前提条件字段具有
值不同于其要求的值），并且在满足其先决条件时必须始终存在，
它一定不能包含在OFPTFPT_WILDCARDS属性中。例如，如果可以省略TCP端口
当IP协议为TCP时，它们将包含在此属性中。另一方面，如果TCP
每次IP协议为TCP时都需要存在端口，并且仅当IP协议时才省略端口
与TCP（例如UDP）不同，它们不会包含在此属性中。

OFPTFPT_WRITE_SETFIELD和OFPTFPT_WRITE_SETFIELD_MISS属性描述*设置字段*
表使用OFPIT_WRITE_ACTIONS指令支持的操作类型，而

OFPTFPT_APPLY_SETFIELD和OFPTFPT_APPLY_SETFIELD_MISS属性描述“ *设置字段*”操作
表使用OFPIT_APPLY_ACTIONS指令支持的类型（请参见7 .2.6.7）。如果HASMASK
在OXM标头中设置了位，则交换机必须支持给定类型的部分重写（请参见
7.2.6.7）。

OFPTFPT_WRITE_COPYFIELD和OFPTFPT_WRITE_COPYFIELD_MISS属性描述了*复制字段*
表使用OFPIT_WRITE_ACTIONS指令支持的操作类型，而

OFPTFPT_APPLY_COPYFIELD和OFPTFPT_APPLY_COPYFIELD_MISS属性描述了*复制字段* ac-
表使用OFPIT_APPLY_ACTIONS指令支持的格式类型（请参见7 .2.6.8）。

OFPTFPT_APPLY_ACTIONS，OFPTFPT_APPLY_ACTIONS_MISS，OFPTFPT_APPLY_SETFIELD和

OFPTFPT_APPLY_SETFIELD_MISS属性包含表可以应用的操作和字段。
对于这些列表中的每一个，如果存在一个元素，则表示该表至少能够应用
隔离一次。当前无法指示可以应用哪些元素
一起，可以在单个流条目中以什么顺序和元素应用多少次。

OFPTFPT_PACKET_TYPES属性使用以下结构和字段：

```
/ *数据包类型属性* /
p_table_feature_prop_oxm_values的结构{
    uint16_t                类型;         / * OFPTFPT_PACKET_TYPES。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
    /* 其次是：
     *     -确切地（长度-4）个字节包含oxm值，然后
     *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t                oxm_values [0];          / * OXM值数组* /
};
OFP_ASSERT（sizeof（p_table_feature_prop_oxm_values的结构== 4）；
```

oxm_values是功能部件的OXM值的列表（请参见7 .2.3.2）。该列表的元素是
完整的OXM字段，包括OXM标头和有效负载。

OFPTFPT_PACKET_TYPES属性包含以下项支持的一组*数据包类型匹配字段*值：
表（请参阅7.2.3.11），定义表支持的数据包类型。对于每种支持的数据包类型
根据表格，它要么包含一个OXM_OF_PACKET_TYPE字段（64位长），要么包含一个实验者OXM
领域。如果表与以太网数据包以外的数据包类型匹配，则必须包含此属性。

该规范的该版本每个交换机仅支持单个数据包类型，具有多个数据包
此属性中的类型不在规范范围内。

OFPTFPT_EXPERIMENTER和OFPTFPT_EXPERIMENTER_MISS属性使用以下结构
和字段：

```
/ *实验者表功能属性* /
p_table_feature_prop_experimenter的结构{
        uint16_t                类型;          / * OFPTFPT_EXPERIMENTER之一，
                                                  OFPTFPT_EXPERIMENTER_MISS。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        uint32_t                实验者 / *采用相同的实验者ID
                                                  形式如结构
                                                  ofp_experimenter_header。* /
        uint32_t                exp_type;               / *实验者定义。* /
        /* 其次是：
         *      -确切地（长度-12个）字节包含实验者数据，然后
         *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
         *          全零字节的字节* /
        uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_table_feature_prop_experimenter的结构）== 12）；
```

实验者字段是实验者ID，exp_type字段是实验者类型，它们
采取与典型实验者结构相同的形式（见7.2 .8）。实验者数据字段
由实验者定义，整个属性需要填充为64位。

7.3.5.19流量监控

OFPMP_FLOW_MONITOR多重类型允许控制器管理跟踪的流量监视器
对流表的更改。在多控制器部署中，这使控制器可以了解
其他控制器对流表所做的更改。控制器可以创建许多流量监视器，
每个流监视器都与某些流表中的流条目的子集匹配。流量监控器将生成
与监视器之一匹配的流表的任何更改的事件。

流监视是一项可选功能，并非所有的OpenFlow交换机都支持。的
可以在交换机上创建的流量监视器的数量也可能受到限制。

7.3.5.19.1流量监控请求

流监视器配置是通过OFPMP_FLOW_MONITOR多部分请求完成的。此请求包含
包含ofp_flow_monitor_request结构的数组，并且可能跨越多个多部分消息。
交换机的缓冲区可能有限，每个请求只能接受有限数量的监视器（请参阅
7.3.5）。

OFPMP_FLOW_MONITOR请求的主体由以下结构的数组组成：

```
/ *类型为OFPMP_FLOW_MONITOR的ofp_multipart_request的正文。
 *
 * OFPMP_FLOW_MONITOR请求的主体由零或更大的数组组成
 *此结构的实例。该请求安排监视流程
```



```
 *符合指定条件，其解释方式与
 *用于OFPMP_FLOW。
 *
 *'id'标识特定的监视器，以使其成为
 *稍后用OFPFMC_DELETE取消。“ id”必须是唯一的
 *尚未取消的现有监视器。
 * /
struct ofp_flow_monitor_request {
        uint32_t monitor_id;            / *此监视器的控制器分配的ID。* /
        uint32_t out_port;              / *必需的输出端口，如果不是OFPP_ANY。* /
        uint32_t out_group;             / *必需的组号（如果不是OFPG_ANY）。* /
        uint16_t标志;                   / * OFPFMF_ *。* /
        uint8_t table_id;               / *一个表的ID或OFPTT_ALL（所有表）。* /
        uint8_t命令;                    / * OFPFMC_ *之一。* /
        struct ofp_match match;         / *要匹配的字段。大小可变。* /
};
OFP_ASSERT（sizeof（struct ofp_flow_monitor_request）== 24）；
```

monitor_id字段唯一地标识用于交换机内特定控制器连接的监视器。

相同的monitor_id可通过两个不同的控制器连接重用于不同的监视器
相同的开关

命令字段定义必须在该监视器上执行的操作，并且必须是其中之一
以下：

```
/ *流量监控器命令* /
of p_flow_monitor_command的枚举{
        OFPFMC_ADD     = 0,             / *新的流量监视器。* /
        OFPFMC_MODIFY = 1,              / *修改现有的流量监视器。* /
        OFPFMC_DELETE = 2               / *删除/取消现有的流量监视器。* /
};
```

命令OFPFMC_ADD创建一个新的流量监控器，并开始根据以下内容监控流量条目
请求参数。如果指定了monitor_id，则OFPFMC_ADD命令必须返回错误
已在使用中（请参阅7.5.4。17）。命令OFPFMC_DELETE销毁指定的流量监视器
通过monitor_id并停止关联的流监视和流监视事件。命令

OFPFMC_MODIFY等效于删除和创建流监视器。

table_id字段指定要监视的流表。如果此字段设置为OFPTT_ALL，则所有表均为
受监控。

匹配字段包含应监视的流条目的描述，并且可能包含
通配符和掩码字段。该字段的匹配行为在6.4节中描述。

out_port和out_group字段可以选择通过输出端口减小流监视器的范围。
如果out_port或out_group分别包含OFPP_ANY或OFPG_ANY之外的其他值，则它
在将流条目与监视器匹配时引入了附加约束。这个约束是
流条目必须包含在更改之前或之后针对该端口或该组的输出操作
触发流量监控器事件。

标志字段是仅适用于流监视器创建的位图，并且可以包括组合
以下标志之一：


```
/ *结构of_flow_monitor_request中的“标志”位。* /
of_p_flow_monitor_flags的枚举{
        / *何时发送更新。* /
        OFPFMF_INITIAL = 1 << 0,            / *最初匹配的流。* /
        OFPFMF_ADD = 1 << 1,                / *添加新匹配流。* /
        OFPFMF_REMOVED = 1 << 2,            / *旧的匹配流将被删除。* /
        OFPFMF_MODIFY = 1 << 3,             / *匹配流，因为它们已更改。* /

        / *包含在更新中的内容。* /
        OFPFMF_INSTRUCTIONS = 1 << 4,  / *如果置位，则包含说明。* /
        OFPFMF_NO_ABBREV = 1 << 5,      / *如果已设置，请完全包含自己的更改。* /
        OFPFMF_ONLY_OWN = 1 << 6,       / *如果设置，则不包括其他控制器。* /
};
```

设置OFPFMF_INITIAL标志时，交换机必须在对流监视器请求的答复中包括：
在请求时与新监视器匹配的所有流条目。

设置OFPFMF_ADD标志时，流监视器必须匹配流条目的所有添加项，例如
使用OFPFC_ADD命令的流程模块。设置OFPFMF_REMOVED标志时，流量监视器必须
匹配所有删除的流条目，例如，使用OFPFC_DELETE或OFPFC_DELETE_STICT的流mod
命令，由于超时，逐出或组或仪表删除导致的流量到期。当OFPFMF_MODIFY
设置标志后，流监视器必须匹配流条目的所有修改，例如流mod请求
使用OFPFC_MODIFY或OFPFC_MODIFY_STICT命令。

类型为OFPFMF_REMOVED的流更新和流除去的消息（请参见7. 4.2）是独立的通知。
机制。流更新未考虑OFPFF_SEND_FLOW_REM标志的值
在流条目上。两个通知机制都可以报告单个流条目。

设置OFPFMF_INSTRUCTIONS标志后，交换机必须在
流监控器事件。如果未设置此标志，则交换机必须省略流中的流进入指令
监视事件。

设置OFPFMF_NO_ABBREV标志时，交换机不得在事件中使用缩写的流更新。如果
如果未设置此标志，则交换机必须在与流表更改相关的事件中使用缩写的流更新
由创建流量监控器的控制器完成。

设置OFPFMF_ONLY_OWN标志时，交换机必须仅针对流表更改生成更新
由创建流量监控器的控制器完成。如果未设置此标志，则开关必须生成
所有控制器完成的流表更改的更新。

7.3.5.19.2流监控回复

当交换机收到OFPMP_FLOW_MONITOR多部分请求时，它将使用

OFPMP_FLOW_MONITOR多部分回复，此回复的交易ID（xid）必须与
请求。与设置了OFPFMF_INITIAL标志的请求的一个流监视器匹配的所有流条目
必须作为流更新包含在回复中（事件类型为OFPFME_INITIAL）。统一的多部分
将为请求中包含的所有监视器生成答复。

OFPMP_FLOW_MONITOR多部分答复还用作异步消息。随时，
只要流表的更改与某个出色的流监视器匹配，则交换机必须发送一个
使用OFPMP_FLOW_MONITOR多部分答复向控制器发送通知，交易ID（xid）为
此答复必须为0。与一个未完成的流量监控器匹配的所有流量条目都必须包含在

回复。例如，如果flow_mod请求修改了与某些流监视器匹配的多个流条目，
通知必须为每个与流监视器匹配的流条目包括一个流更新。

如果修改后的流条目与多个流监视器匹配，则该流条目只能在通知中包含一次。
流更新必须包括由匹配的流监视器指定的所有信息的汇总。
例如，如果任何匹配的流监视器指定指令（OFPFMF_INSTRUCTIONS），则
流更新必须包括说明，并且如果任何匹配的流监视器指定完整的更改
对于控制器自己的更改（OFPFMF_NO_ABBREV），控制器自己的更改将包含在
充分。

不会更改流表的flow_mod消息不会触发任何通知，即使
缩写之一。例如，与任何内容都不匹配的“修改”或“删除” flow_mod消息
流条目将不会触发通知。“添加”或“修改” flow_mod消息可以指定所有
流条目已具有的相同参数，此条件是否触发通知是
未指定，并可能在本规范的将来版本中进行更改。

OFPMP_FLOW_MONITOR多重答复不能越过障碍握手（请参阅7.3.7）。开关必须
始终在给定流表更改之前发送OFPMP_FLOW_MONITOR多部分答复
在负责流表更改的请求之后的OFPT_BARRIER_REQUEST请求。

任何OFPMP_FLOW_MONITOR多部分答复的主体均包含一系列流更新。全流
更新包含更新类型和长度：

```
/ * OFPMP_FLOW_MONITOR答复标头。
 *
 * OFPMP_FLOW_MONITOR答复的正文是可变长度的数组
 *结构，每个结构都以该标头开头。“长度”成员可以
 *用于遍历数组，并且'event'成员可以用于
 *确定特定的结构。
 *
 *每个实例都是8字节长的倍数。* /
struct ofp_flow_update_header {
        uint16_t长度；                          / *此项的长度。* /
        uint16_t事件；                          / * OFPFME_ *之一。* /
        / * ...取决于“事件”的其他数据... * /
};
OFP_ASSERT（sizeof（p_flow_update_header的结构）== 4）；
```

事件字段是流更新类型。当前定义的流更新类型的列表为：

```
/ * p_flow_update_header结构中的'event'值。* /
of_p_flow_update_event的枚举{
        / * struct ofp_flow_update_full。* /
        OFPFME_INITIAL = 0，                    / *创建流量监控器时存在流量。* /
        OFPFME_ADDED = 1                        / *添加了流。* /
        OFPFME_REMOVED = 2                      / *流程已删除。* /
        OFPFME_MODIFIED = 3，                   / *流程说明已更改。* /

        / * struct ofp_flow_update_abbrev。* /
        OFPFME_ABBREV = 4，                     / *缩写答复。* /

        / * struct ofp_flow_update_header。* /
        OFPFME_PAUSED = 5                       / *监视已暂停（缓冲区空间不足）。* /
```



```
    OFPFME_RESUMED = 6,             /*恢复监视。*/
};
```

OFPFME_ADDED、OFPFME_REMOVED和OFPFME_MODIFIED类型表示与流匹配的流条目在流表中已分别添加，删除或修改的监视器，并与
主动流量监控器。OFPFME_INITIAL表示流监视器的初始回复中的流更新部分
并设置了OFPFMF_INITIAL标志。他们使用以下结构和字段：

```
/* OFPMP_FLOW_MONITOR回复OFPFME_INITIAL、OFPFME_ADDED、OFPFME_REMOVED、
 *和OFPFME_MODIFIED。*/
struct ofp_flow_update_full {
    uint16_t长度;                   /*长度为32 +匹配+指令。*/
    uint16_t事件;                   /* OFPFME_*之一。*/
    uint8_t table_id;               /*流表的ID。*/
    uint8_t原因;                    /* OFPRR_*为OFPFME_REMOVED，否则为零。*/
    uint16_t idle_timeout;          /*到期前空闲的秒数。*/
    uint16_t hard_timeout;          /*到期前的秒数。*/
    uint16_t优先级;                 /*条目的优先级。*/
    uint8_t零[4];                   /*保留，当前为零。*/
    uint64_t cookie;                /*不透明的控制器发出的标识符。*/
    struct ofp_match match;         /*要匹配的字段。大小可变。*/
    /*      指令系统。
     *      如果未指定OFPFMF_INSTRUCTIONS，或者“事件”为
     *      OFPFME_REMOVED，不包含任何指令。
     */
    // p_instruction指令的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_flow_update_full）== 32）；
```

这些字段由创建流条目的flow_mod中提供的字段组成（请参见7.3.4.2）。对于
对于类型OFPFME_REMOVED的更新，原因字段是删除原因（请参见7. 4.2），对于其他类型
字段必须为零。

OFPFME_ABBREV类型表示简化的流更新。控制器未指定时

OFPFMF_NO_ABBREV在监控器请求中，任何流表都会由于控制器自身的请求而发生更改
（在同一OpenFlow通道上）（如果可能）将被缩写。它使用以下结构
和字段：

```
/* OFPMP_FLOW_MONITOR回复OFPFME_ABBREV。
 *
 *当控制器未在监视请求中指定OFPFMF_NO_ABBREV时，
 *由于控制器自身的请求（在同一时间），任何流表都会发生变化
 *（OpenFlow通道）将在可能的情况下缩写为这种形式，
 *仅指定OpenFlow请求的“ xid”（例如OFPT_FLOW_MOD）
 *导致更改。
 *有些更改不能缩写，将被完整发送。
 */
struct ofp_flow_update_abbrev {
    uint16_t长度;                   /*长度为8。*/
    uint16_t事件;                   /* OFPFME_ABBREV。*/
    uint32_t xid;                   /*来自flow_mod的控制器指定的xid。*/
};
OFP_ASSERT（sizeof（struct ofp_flow_update_abbrev）== 8）；
```


xid字段是与此更新相关的控制器请求的事务ID。

OFPFME_PAUSED和OFPFME_RESUMED类型表示由于缓冲区而导致的监视暂停和恢复
管理约束。他们使用以下结构和字段：

```
/* OFPMP_FLOW_MONITOR回复OFPFME_PAUSED和OFPFME_RESUMED。
 */
struct ofp_flow_update_paused {
    uint16_t长度;                   /*长度为8。*/
    uint16_t事件;                   /* OFPFME_*之一。*/
    uint8_t零[4];                   /*保留，当前为零。*/
};
OFP_ASSERT（sizeof（p_flow_update_paused的结构）== 8）；
```

7.3.5.19.3流量监控暂停

用于流监视器通知的OpenFlow消息可能会使交换机可用的缓冲区空间溢出，
临时地（例如由于网络条件而降低OpenFlow流量）或永久性地（例如
流量表的持续变化率超过了交换机和控制器之间的网络带宽。
当排队的流更新数量达到一定数量时，流监视暂停机制将禁用流更新
达到限制，并使交换机和控制器能够正常恢复。暂停机制
确保流更新的最大缓冲区空间要求由限制加
支持的最大流条目数

当交换机的通知缓冲区空间达到交换机定义的限制阈值时，交换机必须
反应如下：

1.交换机必须按照所有已经存在的命令向控制器发送OFPFME_PAUSED流更新
排队的流更新。收到此消息后，控制器知道其对流的视图
流监视器通知所代表的表不完整。

2.只要通知缓冲区不为空：

•不得发送OFPFMF_ADD和OFPFMF_MODIFY流更新。

•OFPFMF_REMOVED流更新仍必须发送，但仅适用于之前存在的流条目
交换机发送了OFPFME_PAUSED流更新。

•OFPFMF_ABBREV流更新必须遵循相同的规则，尤其是与流更新相关的
不得发送添加或修改流条目。

3.当通知缓冲区为空时，交换机必须发送OFPFME_ADDED流更新以进行流
自缓冲区达到极限以来添加的条目，并且仍存在于流表中，并且必须发送
更改限制前已存在的流条目的OFPFME_MODIFIED流更新，已更改
达到限制后仍然存在于流表中

4.交换机必须向控制器发送OFPFME_RESUMED流更新。收到这个之后
消息，控制器知道它对流表的视图，由流监视器表示
通知，再次完成。

7.3.5.20捆绑包功能分为多个部分

捆绑包功能请求使用OFPMP_BUNDLE_FEATURES消息类型，允许控制器执行以下操作：
查询交换机的捆绑功能，包括是否支持原子捆绑
包和预定的包。

7.3.5.20.1捆绑包功能请求消息格式

捆绑包功能请求消息的主体由struct ofp_bundle_features_request定义，
如下：

```
/ * OFPMP_BUNDLE_FEATURES请求的主体。* /
struct ofp_bundle_features_request {
    uint32_t feature_request_flags; / *“ ofp_bundle_feature_flags”的位图。* /
    uint8_t pad [4];

    / *捆绑功能特性列表-0或更多。* /
    p_bundle_features_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_bundle_features_request的结构）== 8）；
```

主体由标志字段组成，后跟零个或多个属性TLV字段。捆绑包功能
属性在第7.3.5.20节.3。中指定。标志字段feature_request_flags定义为
如下：

```
of_pundle_feature_flags的枚举{
    OFPBF_TIMESTAMP = 1 << 0，              / *请求中包含时间戳。* /
    OFPBF_TIME_SET_SCHED = 1 << 1，/ *请求包括sched_max_future和
                                    * sched_max_past参数。* /
};
```

•OFPBF_TIMESTAMP指示当前请求包括时间戳，即当前
租用时间取决于控制器的时钟。 时间戳记包含在
ofp_bundle_features_prop_time属性。

•OFPBF_TIME_SET_SCHED表示请求中包含调度公差参数，

sched_max_future和sched_max_past，并且交换机应更新其调度容限-
根据接收到的值

如果设置了标志OFPBF_TIMESTAMP或OFPBF_TIME_SET_SCHED中的至少一个，则捆绑包功能
该请求包括ofp_bundle_features_prop_time属性。

7.3.5.20.2捆绑包功能回复消息格式

接收到捆绑功能请求并成功处理的交换机将发送捆绑
功能回复控制器。捆绑销售商品功能的回复是OFPT_MULTIPART_REPLY邮件-
类型为OFPMP_BUNDLE_FEATURES的贤哲。捆绑包功能回复消息的主体是

p_bundle_features的结构，如下所示：


```
/ *回复OFPMP_BUNDLE_FEATURES请求的正文。* /
struct ofp_bundle_features {
    uint16_t功能； / *“ ofp_bundle_flags”的位图。* /
    uint8_t pad [6];

    / *捆绑功能特性列表-0或更多。* /
    p_bundle_features_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_bundle_features的结构）== 8）；
```

功能字段指示交换机支持的与捆绑软件相关的功能。的
功能字段的格式由struct ofp_bundle_flags定义，如下所示：

```
/ *捆绑包配置标志。* /
of_pundle_flags枚举{
    OFPBF_ATOMIC = 1 << 0，/ *自动执行。* /
    OFPBF_ORDERED = 1 << 1，/ *按指定顺序执行。* /
    OFPBF_TIME      = 1 << 2，/ *在指定的时间执行。* /
};
```

- OFPBF_ATOMIC和OFPBF_ORDERED表示交换机支持原子束，并按顺序排列
  束（分别参见6.9.4节）。
- OFPBF_TIME表示该交换机支持调度的捆绑包，如第二节中所述。
  6 .9.6。 设置此标志后，捆绑包功能回复消息将包含
  ofp_bundle_features_prop_time属性。

捆绑包功能回复消息可能包含零个或多个属性。捆绑包的格式
功能属性定义如下。

如果交换机收到捆绑功能请求，但无法处理它，则会回复一条错误消息。
类型为OFPET_BAD_REQUEST（请参见7.5.4.2 。）。代码OFPBRC_MULTIPART_BAD_SCHED表示
请求启用了OFPBF_TIME_SET_SCHED标志，并且交换机无法更新调度
公差值。

7.3.5.20.3捆绑包功能属性

捆绑包特征属性是具有通用标头格式的可选TLV字段，如下所示：

```
struct ofp_bundle_features_prop_header {
    uint16_t类型;                / * OFPTMPBF_ *之一。* /
    uint16_t长度;                / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_bundle_features_prop_header的结构）== 4）；
```

当前定义的类型如下：

```
/ *捆绑包具有属性类型。* /
of_pundle_features_prop_type的枚举{
    OFPTMPBF_TIME_CAPABILITY = 0x1，/ *时间要素属性。* /
    OFPTMPBF_EXPERIMENTER = 0xFFFF，/ *实验者属性。* /
};
```


7.3.5.20.4捆绑包具有时间属性

捆绑特征请求，其中标志OFPBF_TIMESTAMP或OFPBF_TIME_SET_SCHED中的至少一个设置，并包含ofp_bundle_features_prop_time属性。具有以下功能的捆绑包功能回复OFPBF_TIME标志集包含了ofp_bundle_features_prop_time属性。

ofp_bundle_features_prop_time属性的定义如下：

```
/ *捆绑时间功能。* /
struct ofp_bundle_features_prop_time {
        uint16_t类型;                          / * OFPTMPBF_TIME_CAPABILITY。* /
        uint16_t长度;                          / *此属性的长度（以字节为单位）。* /
        uint8_t pad [4];
        struct ofp_time sched_accuracy;        / *调度精度，即
                                                 *开关可以准确执行
                                                 *预定的提交。使用该字段
                                                 *仅在捆绑包功能中提供回复，并且
                                                 *在捆绑包功能中被忽略
                                                 * 要求。* /
        p_time sched_max_future结构；/ *之间的最大差
                                                 *安排时间和当前时间。* /
        struct ofp_time sched_max_past;        / *如果计划时间在
                                                 *过去，定义最大差异
                                                 *当前时间和
                                                 *安排时间。* /
        struct ofp_time timestamp;             / *表示
                                                 *发送此消息。* /
};
OFP_ASSERT（sizeof（p_bundle_features_prop_time的结构== 72）；
```

如第7 .3.9.5节中所定义，使用struct ofp_time表示时间。

捆绑功能请求中的ofp_bundle_features_prop_time属性包括：

- sched_accuracy：此字段仅与捆绑包功能答复有关，并且开关必须忽略
  捆绑包功能请求中的此字段。
- sched_max_future和sched_max_past：使用以下命令接收捆绑功能请求的开关

  OFPBF_TIME_SET_SCHED设置应尝试根据以下方法更改其计划公差值
  来自time属性的sched_max_future和sched_max_past值。如果开关确实
  未能成功更新其计划公差值，它会返回一条错误消息，类型为

  OFPET_BAD_REQUEST，以及OFPBRC_MULTIPART_BAD_SCHED代码。
- 时间戳，指示在此消息传输期间控制器的时间。一个开关
  接收到具有OFPBF_TIMESTAMP设置的捆绑包功能请求，可以使用接收到的时间戳来
  粗略估计其时钟与控制器时钟之间的偏移量。

捆绑包功能答复中的ofp_bundle_features_prop_time属性包括：

- sched_accuracy，指示估计的交换机调度准确性。例如，如果
  sched_accuracy的值是1000000纳秒（1 ms），这意味着当交换机接收到
  如果捆绑提交计划在时间*T s进行*，则实际上在*T s* ±1 *ms*调用提交。的
  在第6.9.6.3节中讨论了影响调度准确性的因素。

- sched_max_future和sched_max_past，包含交换机的调度容差值。
  如果相应的捆绑包功能请求已启用OFPBF_TIME_SET_SCHED标志，则这些
  两个字段与请求中的控制器发送的字段相同。
- 时间戳，指示此功能回复的传输过程中交换机的时间。每捆
  包含时间属性的功能回复还包含时间戳。时间戳可能是
  由控制器用来粗略估计开关的时钟是否与
  控制器的。

7.3.5.21实验者多部分

请求特定于实验者的多部分消息，其类型为OFPMP_EXPERIMENTER。

请求和回复正文的前几个字节具有以下结构：

```
/ *类型为OFPMP_EXPERIMENTER的ofp_multipart_request / reply的正文。* /
struct ofp_experimenter_multipart_header {
        uint32_t实验者；                    / *实验者ID。* /
        uint32_t exp_type;                  / *实验者定义。* /
        / *实验者定义的任意附加数据。* /
};
OFP_ASSERT（sizeof（p_experimenter_multipart_header的结构）== 8）；
```

实验者字段是实验者ID，exp_type字段是实验者类型，它们
采取与典型实验者结构相同的形式（请参见7.2.8）。

请求和回复主体的其余部分是实验者定义的，不需要填充。

7.3.6包出消息

当控制器希望通过数据路径发送数据包时，它使用OFPT_PACKET_OUT
信息：

```
/ *发送数据包（控制器->数据路径）。* /
struct ofp_packet_out {
        struct ofp_header标头；
        uint32_t buffer_id;                 / *由数据路径分配的ID（OFP_NO_BUFFER
                                                如果没有）。* /
        uint16_t actions_len;               / *操作数组的大小，以字节为单位。* /
        uint8_t pad [2];                    / *对齐64位。* /
        struct ofp_match match;             / *数据包管道字段。大小可变。* /
        / *变量大小和填充匹配后面是操作列表。* /
        / * struct ofp_action_header actions [0]; * // *动作列表-0或更多。* /
        / *可变大小操作列表后面有数据包数据（可选）。
          *仅当buffer_id == -1时，此数据才有意义。* /
        / * uint8_t数据[0]; * /              / *数据包数据。长度推断
                                                从标题的长度字段开始。* /
};
OFP_ASSERT（sizeof（p_packet_out的结构）== 24）；
```


buffer_id与ofp_packet_in消息中给出的相同。如果buffer_id为OFP_NO_BUFFER，
然后将数据包数据包含在数据数组中，并将封装在消息中的数据包
由消息的动作处理。OFP_NO_BUFFER为0xffffffff。如果buffer_id有效，则相应的-
响应数据包从缓冲区中删除，并通过消息的操作进行处理。

匹配字段是一组OXM TLV，其中包含与数据包关联的管道字段。的
匹配字段中的TLV定义了必须与数据包关联的管道字段的值
用于OpenFlow处理。OXM TLV必须包括“ *数据包类型匹配”字段*以标识
数据包中包含的数据包类型，除非数据包是以太网（请参见7.2.3.11）。比赛场
只能包含管道字段，并且不能包含任何头字段（请参见7 .2.3.7）。如果匹配字段
包含任何头字段，交换机必须拒绝ofp_packet_out消息并发送*管道*

*仅字段*错误消息（请参见7 .5.4.2）。

如果控制器通过ofp_packet_in消息接收到数据包，则match字段应为
使用ofp_packet_in消息中提供的管道字段值进行设置（请参见7.4.1）。

开关中必须将未包含在匹配字段中的管道字段设置为零，但

OXM_OF_IN_PORT是强制性的，而OXM_OF_IN_PHY_PORT必须设置为的值
OXM_OF_IN_PORT。

匹配字段中包含的OXM_OF_IN_PORT TLV指定必须关联的入口端口
与用于OpenFlow处理的数据包一起使用。此TLV在比赛字段中是必填项，并且必须是
设置为有效的标准交换机端口（请参阅4. 2）或OFPP_CONTROLLER。例如，此值
使用组OFPP_TABLE，OFPP_IN_PORT和OFPP_ALL处理数据包时，将使用TLV。如果

OXM_OF_IN_PORT TLV不包含在match字段中，交换机必须拒绝ofp_packet_out
消息并发送*错误端口*错误消息（请参见7.5.4.2）。

操作字段是操作列表，这些操作定义了交换机应如何处理数据包。它
可能包括数据包修改，组处理和输出端口。一个动作的清单

OFPT_PACKET_OUT消息还可以指定OFPP_TABLE保留端口作为要处理的输出操作
从第一个流表开始（参见4.5），通过OpenFlow管道传输数据包。如果OFPP_TABLE为
如果指定，则将匹配字段用作流表查找中的管道字段集。

如果操作列表中的任何操作无效或受支持，则交换机必须生成错误消息

类型为OFPET_BAD_ACTION（请参见7 .5.4.3）。
在某些情况下，发送到OFPP_TABLE的数据包可能会作为流的结果转发回控制器
输入动作或表格未命中。对于此类控制器到开关回路的检测和采取措施不在
本规范的范围。通常，不能保证OpenFlow消息按顺序处理，
因此，如果使用OFPP_TABLE的OFPT_PACKET_OUT消息取决于最近发送给
交换机（带有OFPT_FLOW_MOD消息）之前，可能需要OFPT_BARRIER_REQUEST消息
到OFPT_PACKET_OUT消息以确保流条目在提交之前已提交到流表
执行OFPP_TABLE。

数据字段为空，或者当buffer_id为OFP_NO_BUFFER时，数据字段包含数据包
待处理。数据包类型由*匹配中*包含的“ *数据包类型匹配*”*字段*确定
字段，如果不存在“数据包类型匹配字段”，则该数据包必须是以太网（以太网帧，包括
以太网报头和以太网有效负载，但没有以太网FCS-参见7 .2.3.11）。唯一的处理
数据包上的开关完成的操作是由操作中的OpenFlow操作明确指定的处理
字段，特别是交换机一定不能透明地设置以太网源地址或IP校验和。

7.3.7屏障消息

当控制器想要通过交换机控制消息的处理和排序时，它可以使用

OFPT_BARRIER_REQUEST消息。此消息没有正文，仅由一个OpenFlow组成
标头。

屏障消息的第一个功能是确保消息排序，从而使消息依赖性
不违反（请参阅6.2）。屏障消息的第二个功能是让控制器接收通知
用于完成的操作。屏障的第三项功能是确保控制器发送的状态
提交到数据路径。

当交换机在连接上收到OFPT_BARRIER_REQUEST消息时，交换机必须完成
处理该连接上所有以前收到的消息，包括发送相应的答复
或错误消息，在执行屏障请求之前或之后在连接上收到的任何消息
屏障请求。如果在切换时已经处理了所有以前收到的消息
接收到屏障请求后，交换机可以立即执行屏障请求。何时加工
在屏障完成之前所有消息中的第一个，交换机将执行屏障请求。所有状态变更
与屏障请求之前的消息对应的消息必须提交到数据路径，以便任何
交换机在发送屏障响应后收到的数据包必须由数据路径处理
在屏障命令之前使用所有消息的状态更改进行更新（此要求不
适用于正在执行屏障请求的飞行中或通过管道排队的数据包）。什么时候
屏障请求的执行完成后，交换机必须发送OFPT_BARRIER_REPLY消息
与该障碍请求的名称相同。屏障答复必须在答复或错误消息之后发送
对应于屏障请求之前的消息。发送屏障回复后，交换机可以
正常处理屏障请求后的所有消息（请参见6.2）。

7.3.8角色请求消息

当控制器想要更改其角色时，它将使用OFPT_ROLE_REQUEST消息，并包含以下内容：
结构体：

```
/ *角色请求和回复消息。* /
struct ofp_role_request {
        struct ofp_header标头；                / *类型OFPT_ROLE_REQUEST / OFPT_ROLE_REPLY。* /
        uint32_t角色；                         / * OFPCR_ROLE_ *之一。* /
        uint16_t short_id;                     / *控制器的ID号。* /
        uint8_t pad [2];                       / *对齐64位。* /
        uint64_t generation_id;                / *主选举产生ID * /
};
OFP_ASSERT（sizeof（struct ofp_role_request）== 24）；
```

字段角色是控制器要承担的新角色，可以具有以下值：

```
/ *控制器角色。* /
of_p_controller_role的枚举{
        OFPCR_ROLE_NOCHANGE = 0，      / *不要更改当前角色。* /
        OFPCR_ROLE_EQUAL    = 1        / *默认角色，完全访问权限。* /
        OFPCR_ROLE_MASTER   = 2        / *完全访问权限，最多一个主服务器。* /
        OFPCR_ROLE_SLAVE    = 3，      / *只读访问。* /
};
```


如果消息中的角色值为OFPCR_ROLE_MASTER或OFPCR_ROLE_SLAVE，则交换机必须验证
generation_id以检查过时的消息（请参阅6. 3.7）。如果验证失败，则交换机必须丢弃
角色请求并返回*Stale*错误消息（请参见7.5.4.12）。

如果角色值为OFPCR_ROLE_MASTER，则所有其他角色为OFPCR_ROLE_MASTER的控制器为
更改为OFPCR_ROLE_SLAVE（请参阅6. 3.7）。如果角色值为OFPCR_ROLE_NOCHANGE，则当前角色
控制器的位置不变; 这使控制器可以查询其当前角色而无需更改
它。

收到OFPT_ROLE_REQUEST消息后，如果没有错误，则交换机必须重新执行以下操作：
打开OFPT_ROLE_REPLY消息。该消息的结构与

OFPT_ROLE_REQUEST消息，并且字段角色是控制器的当前角色。场
generation_id设置为当前的generation_id（与最后一个成功关联的generation_id
如果角色为当前的generation_id，则具有角色OFPCR_ROLE_MASTER或OFPCR_ROLE_SLAVE的角色请求
从未被控制器设置过，回复中的字段generation_id必须设置为最大字段
值（等于-1的无符号值）。

如果交换机必须将另一个控制器的角色从OFPCR_ROLE_MASTER更改为OFPCR_ROLE_SLAVE，
它必须向该控制器发送OFPT_ROLE_STATUS消息（请参见7 .4.4）。

OFPT_ROLE_REQUEST消息还为控制器设置short_id。此ID（如果已定义）是唯一的
标识特定的控制器。该ID最初为OFPCID_UNDEFINED，并且控制器可以设置

将short_id设置为他们选择的值，如果不希望使用它，则将其设置为OFPCID_UNDEFINED。只要
可以将OFPCID_UNDEFINED ID分配给多个控制器，并将short_id设置为
一个已经使用的值将导致整个OFPT_ROLE_REQUEST消息被拒绝（请参阅7 .5.4.12）。

OFPCID_UNDEFINED为0。

7.3.9捆绑消息

7.3.9.1捆绑控制消息

控制器可以使用OFPT_BUNDLE_CONTROL请求创建，销毁和提交捆绑包。的

OFPT_BUNDLE_CONTROL消息使用以下结构：

```
/ * OFPT_BUNDLE_CONTROL的消息结构。* /
struct ofp_bundle_ctrl_msg {
    p_header结构                标题
    uint32_t                    bundle_id;          / *标识捆绑包。* /
    uint16_t                    类型;               / * OFPBCT_ *。* /
    uint16_t                    标志;               / * OFPBF_ *标志的位图。* /

    / *捆绑包属性列表。* /
    p_bundle_prop_header属性的结构[0]; / *零个或多个属性。* /
};
OFP_ASSERT（sizeof（p_bundle_ctrl_msg的结构）== 16）;
```

bundle_id字段是包标识符，这是控制器选择的32位数字。捆绑包
标识符在生命周期内在连接上应该是唯一的。

类型字段是控制消息类型。当前定义的控件类型为：


```
/ *捆绑控制消息类型* /
of_pundle_ctrl_type的枚举{
    OFPBCT_OPEN_REQUEST     = 0，
    OFPBCT_OPEN_REPLY       = 1
    OFPBCT_CLOSE_REQUEST    = 2
    OFPBCT_CLOSE_REPLY      = 3，
    OFPBCT_COMMIT_REQUEST = 4
    OFPBCT_COMMIT_REPLY     = 5
    OFPBCT_DISCARD_REQUEST = 6
    OFPBCT_DISCARD_REPLY    = 7
};
```

标志字段是捆绑标志的位掩码（请参见7 .3.9.3）。
properties字段是捆绑软件属性的列表（请参见7.3.9.4）。

7.3.9.2捆绑添加消息

控制器可以使用OFPT_BUNDLE_ADD_MESSAGE消息将请求添加到包中。的
OFPT_BUNDLE_ADD_MESSAGE消息使用以下结构：

```
/ * OFPT_BUNDLE_ADD_MESSAGE的消息结构。
  *完成捆绑中添加消息的操作。* /
struct ofp_bundle_add_msg {
        p_header结构                标题
        uint32_t                    bundle_id;          / *标识捆绑包。* /
        uint16_t                    垫;                 / *对齐64位。* /
        uint16_t                    标志;               / * OFPBF_ *标志的位图。* /

        p_header结构                信息;               / *消息已添加到分发包中。* /

        / *如果存在一个或多个属性，则“消息”后跟：
          *       -准确（message.length + 7）/ 8 * 8-（message.length）（0到7之间）
          *          全零字节的字节* /

        / *捆绑包属性列表。* /
        / * p_bundle_prop_header属性的结构[0]; * / / *零或多个
                                                                    属性。* /
};
OFP_ASSERT（sizeof（struct ofp_bundle_add_msg）== 24）;
```

bundle_id字段是包标识符，这是控制器选择的32位数字。捆绑包
标识符应为先前已打开但尚未关闭的包。

标志字段是捆绑标志的位掩码（请参见7 .3.9.3）。

该消息是要添加到捆绑软件中的OpenFlow消息，它可以是
交换机可以捆绑销售。消息中的字段xid必须与的字段xid相同
OFPT_BUNDLE_ADD_MESSAGE消息。

properties字段是捆绑软件属性的列表（请参见7.3.9.4）。

7.3.9.3捆绑标志

捆绑标志可修改捆绑的行为。

当前定义的捆绑标志是：

```
/ *捆绑包配置标志。* /
of_pundle_flags枚举{
        OFPBF_ATOMIC = 1 << 0，/ *自动执行。* /
        OFPBF_ORDERED = 1 << 1，/ *按指定顺序执行。* /
        OFPBF_TIME       = 1 << 2，/ *在指定的时间执行。* /
};
```

- •OFPBF_ATOMIC设置为请求束的完全原子应用。
- •OFPBF_ORDERED设置为请求严格按顺序应用分发包的消息。
- •OFPBF_TIME用于捆绑提交消息中，以指示该提交消息包括
  使用ofp_bundle_prop_time，该包的执行时间。有关预定的更多详细信息
  捆绑包，请参阅第6.9.6节。

必须在每个捆绑消息部分上指定捆绑标志OFPBF_ATOMIC和OFPBF_ORDERED
捆绑软件，并且它们必须保持一致。只能在捆绑提交中设置标志OFPBF_TIME
消息，并且接收者在其他捆绑消息中应将其忽略。

7.3.9.4捆绑包属性

当前定义的捆绑软件属性类型的列表为：

```
/ *捆绑包属性类型。* /
of_pundle_prop_type的枚举{
        OFPBPT_TIME                     = 1
```

```
    OFPBPT_EXPERIMENTER             = 0xFFFF，/ *实验者属性。* //
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有捆绑包属性的通用标头* /
struct ofp_bundle_prop_header {
    uint16_t                类型;       / * OFPBPT_ *之一。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_bundle_prop_header的结构）== 4）；
```

ofp_bundle_prop_time属性包含在具有OFPBF_TIME标志的捆绑消息中组。如6.9节所述，此属性用于计划的包中。6.此属性只能包含在捆绑包提交消息中。OFPBPT_TIME属性使用以下结构，并且领域：


```
/ *捆绑时间属性。* /
    struct ofp_bundle_prop_time {
    uint16_t类型;               / * OFPBPT_TIME * /
    uint16_t长度;               / *此属性的长度，以字节为单位（24）。* /
    uint8_t pad [4];
    struct ofp_time schedule_time;
};
OFP_ASSERT（sizeof（p_bundle_prop_time的结构）== 24）；
```

cheduled_time字段指定交换机应执行调度的时间束。该字段的格式为ofp_time，在第7 .3.9.5节中定义。

OFPBPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者捆绑包属性* /
struct ofp_bundle_prop_experimenter {
    uint16_t                类型;       / * OFPBPT_EXPERIMENTER。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
    uint32_t                实验者 / *采用相同的实验者ID
                                        形式如结构
                                        ofp_experimenter_header。* /
    uint32_t                exp_type;           / *实验者定义。* /
    /* 其次是：
     *      -确切地（长度-12个）字节包含实验者数据，然后
     *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *         全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_bundle_prop_experimenter的结构）== 12）；
```

实验者字段是实验者ID，其格式与struct中的相同ofp_experimenter。

7.3.9.5时间格式

捆绑包提交请求消息可能包含schedule_time，该时间指定捆绑包*何时*应该承诺。schedule_time字段使用struct ofp_time。时间格式ofp_time也用于捆绑包功能消息（请参见7.3.5.20.4）。

时间格式ofp_time基于IEEE 1588-2008标准中定义的格式。它包含两个子字段中的一个；秒字段，以秒为单位表示时间的整数部分1个和一纳秒字段，以秒为单位表示时间的小数部分，即0≤ *纳秒* ≤（10 9 -1）。

```
/* 时间格式。* /
struct ofp_time {
    uint64_t秒;
    uint32_t纳秒;
    uint8_t pad [4];
};
OFP_ASSERT（sizeof（struct ofp_time）== 16）；
```

1 IEEE 1588-2008中的秒字段为48位长。此处定义的秒字段是64位字段，但具有相同的语义作为IEEE 1588-2008时间格式中的秒字段。

根据IEEE 1588-2008标准中的规定，时间是根据国际原子力计进行测量的
时间（TAI）时标。时标的起点（也称为纪元）定义为1月1日
1970 00:00:00 TAI; 在每个瞬间，时间都用已过去的秒数表示
从那个时代开始

7.3.9.6创建和打开包

要创建捆绑包，控制器会发送OFPT_BUNDLE_CONTROL消息，其类型为

OFPBCT_OPEN_REQUEST。交换机必须创建一个ID为bundle_id的新捆绑包
租用连接，以及标志和属性中指定的选项。如果操作成功，

交换机必须返回类型为OFPBCT_OPEN_REPLY的OFPT_BUNDLE_CONTROL消息。如果
错误出现，返回错误消息。

捆绑包打开请求的参数必须经过验证，如果不正确，则返回错误
支持（请参阅7.5.4.18）。

如果bundle_id已引用附加到同一连接的现有捆绑软件，则该开关
必须拒绝打开新的捆绑包，并发送具有OFPET_BUNDLE_FAILED类型的ofp_error_msg和

OFPBFC_BAD_ID代码。必须丢弃bundle_id标识的现有捆绑包。

如果交换机由于在交换机上打开的捆绑包过多而无法打开此捆绑包
或连接到当前连接，则交换机必须拒绝打开新捆绑包并发送

OFp_error_msg，具有OFPET_BUNDLE_FAILED类型和OFPBFC_OUT_OF_BUNDLES代码。如果开关
由于连接使用的传输方式不可靠而无法打开捆绑包，因此交换机必须
拒绝打开新的捆绑包，并发送具有OFPET_BUNDLE_FAILED类型的ofp_error_msg和

OFPBFC_OUT_OF_BUNDLES代码。

如果标志字段请求了某些交换机无法实现的功能，则交换机必须
拒绝打开新的捆绑包，并发送具有OFPET_BUNDLE_FAILED类型的ofp_error_msg和

OFPBFC_BAD_FLAGS代码。

7.3.9.7将消息添加到分发包

交换机使用OFPT_BUNDLE_ADD_MESSAGE将消息添加到包中。捆绑包打开后，
控制器可以发送一系列OFPT_BUNDLE_ADD_MESSAGE消息来填充捆绑包。每

OFPT_BUNDLE_ADD_MESSAGE包含一个OpenFlow消息，该OpenFlow消息经过验证，并且
如果成功，它将存储在当前连接上的bundle_id指定的包中。如果有消息
验证错误或出现捆绑错误情况时，将返回错误消息（请参见7.5.4.18）。

消息验证至少包括语法检查以及消息中请求的所有功能
支持，并且可以选择包括检查资源可用性（请参阅6 9.3）。如果消息失败
验证，必须返回错误消息。错误消息必须使用违规的xid
消息，与该消息相对应的错误数据字段和与该消息相对应的错误代码
验证失败。

如果bundle_id引用了当前连接上不存在的捆绑软件，则对应的
捆绑包是使用OFPT_BUNDLE_ADD_MESSAGE消息中的参数创建的。如果从发生错误

创建束，则返回相应的错误消息（参见7.3。 9.6）。没有OFPT_BUNDLE_CONTROL
在这种情况下，交换机将返回类型为OFPBCT_OPEN_REPLY的消息。

如果bundle_id引用已关闭的捆绑包，则交换机必须拒绝添加消息
到捆绑包，丢弃捆绑包并发送OFPET_BUNDLE_FAILED类型的ofp_error_msg，

OFPBFC_BUNDLE_CLOSED代码。

如果标志字段与打开包时指定的标志不同，则切换
必须拒绝将消息添加到分发包，丢弃分发包并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_FLAGS代码。

如果请求中的消息通常在交换机上受支持，但在Bun-
闲置时，交换机必须拒绝将消息添加到捆绑包，并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_UNSUP代码。hello，echo和bundle就是这种情况
消息，非请求消息（例如答复，状态和错误消息），或者
实现不支持在捆绑包中包含特定的修改消息。

如果请求中的消息与包中已存储的另一条消息不兼容，
闲置时，交换机必须拒绝将消息添加到捆绑包，并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_CONFLICT代码。

如果捆绑包已满且无法容纳请求中的消息，则交换机必须拒绝添加消息
到捆绑包中，并发送具有OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_TOO_MANY类型的ofp_error_msg
码。

如果请求中的消息没有有效的长度字段，则交换机必须拒绝添加消息
到捆绑包中，并发送OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_BAD_LEN的ofp_error_msg
码。

捆绑中添加的邮件应具有唯一的xid，以帮助将错误与邮件进行匹配，但
相同消息序列的多部分消息部分，并且捆绑包添加消息的xid必须是
与嵌入式消息相同。交换机可以选择验证消息的两个xid字段
一致或捆绑包的两个非多部分消息没有相同的xid，如果
并非如此，拒绝将新消息添加到分发包并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_BAD_XID代码。

7.3.9.8关闭捆绑

为了完成录制捆绑包，控制器可以发送类型为OFPT_BUNDLE_CONTROL的消息

OFPBCT_CLOSE_REQUEST。交换机必须关闭当前con-上的bundle_id指定的捆绑软件。
连接。关闭捆绑包后，将无法再对其进行修改，也无法向其添加任何消息，
它只能被提交或丢弃。关闭包是可选的。如果操作成功，

交换机必须返回类型为OFPBCT_CLOSE_REPLY的OFPT_BUNDLE_CONTROL消息。如果
错误出现，返回错误消息（请参见7.5.4.18）。

如果bundle_id引用了不存在的捆绑包，则交换机必须拒绝该请求并发送

ofp_error_msg，具有OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_ID代码。

如果bundle_id引用已关闭的捆绑包，则交换机必须拒绝关闭
捆绑软件，丢弃捆绑软件并发送OFPET_BUNDLE_FAILED类型的ofp_error_msg和

OFPBFC_BUNDLE_CLOSED代码。

如果标志字段与打开捆绑商品时指定的标志不同，则
交换机必须拒绝靠近包，丢弃包并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_FLAGS代码。

交换机可以选择对捆绑包的消息部分进行额外的验证，作为
关闭请求，例如验证资源可用性。对于每条未通过此附加消息的消息
验证，必须生成一条错误消息，该错误消息引用了有问题的消息。发送那些之后
个别错误消息，交换机必须丢弃该捆绑包并发送附加的p_error_msg
具有OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_FAILED代码。

7.3.9.9提交捆绑包

为了完成并应用捆绑包，控制器发送类型为OFPT_BUNDLE_CONTROL的消息

OFPBCT_COMMIT_REQUEST。交换机必须提交当前目录上的bundle_id指定的包。
租用连接，它必须将存储在捆绑软件中的所有消息作为单个操作应用，否则返回错误。
提交操作和捆绑软件消息部分的应用方式取决于捆绑软件
标志（请参见7.3.9。3）。如果在此请求之前未关闭捆绑销售商品，它将自动关闭（请参见
7.3.9.8）。

仅当捆绑中的所有消息都可以正确应用时，提交才成功。
如果捆绑包不包含任何消息，则提交总是成功的。如果提交成功，
交换机必须将捆绑包中的所有消息作为单个操作应用，并且必须使用OFPT_BUNDLE_CONTROL
交换机必须返回类型为OFPBCT_COMMIT_REPLY的消息。OFPT_BUNDLE_CONTROL
处理完所有消息后，必须将类型为OFPBCT_COMMIT_REPLY的消息发送到控制器
保证捆绑包的消息部分不再失败或产生错误。

如果捆绑包的一个或多个消息部分不能无错误地应用，例如由于资源问题
可用性，提交失败，并且必须丢弃捆绑包中的所有消息，而不能
正在应用。当提交失败时，交换机必须生成一条错误消息，对应于
无法应用的消息。错误消息必须使用有问题的消息的xid，
与该消息相对应的错误数据字段和与失败相对应的错误代码。如果
多个消息正在生成错误，交换机可能仅返回找到的第一个错误或生成
同一包的多个错误消息。发送这些单独的错误消息后，交换机
必须发送额外的OFp_BUNDLE_FAILED类型和OFPBFC_MSG_FAILED的ofp_error_msg
码。

提交操作之后，无论提交是否成功，都将放弃捆绑包。后
收到此操作的成功答复或错误消息后，控制器可以重新使用bundle_id。

修改请求可能要求将回复返回给控制器或生成事件。
由于该捆绑包中的任何消息都可能失败，因此只有在所有修改后，才能生成回复和事件。
束中的阳离子已被应用。对于答复，每个答复都包含事务ID
的相应请求。

如果bundle_id引用了不存在的捆绑包，则交换机必须拒绝该请求并发送

ofp_error_msg，具有OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_ID代码。

如果标志字段与打开捆绑商品时指定的标志不同，则
交换机必须拒绝提交捆绑软件，丢弃捆绑软件并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_FLAGS代码。

7.3.9.10丢弃捆绑包

为了完成并丢弃捆绑包，控制器发送类型为OFPT_BUNDLE_CONTROL的消息

OFPBCT_DISCARD_REQUEST。交换机必须丢弃当前设备上bundle_id指定的捆绑软件
连接，并且捆绑包中的所有消息都将被丢弃。如果操作成功，
交换机必须返回类型为OFPBCT_DISCARD_REPLY的OFPT_BUNDLE_CONTROL消息。
如果出现错误，则返回错误消息。收到成功答复或错误后
消息，控制器可以重用bundle_id。

所有实现都必须能够在当前配置下处理现有捆绑上的丢弃请求。
连接，不会触发错误。如果bundle_id引用了不存在的捆绑软件，则交换机必须
拒绝请求并发送OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_ID的ofp_error_msg
码。

7.3.9.11其他包错误情况

如果OFPT_BUNDLE_CONTROL消息包含无效类型，则交换机必须拒绝该请求，并且
发送带有OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_TYPE代码的ofp_error_msg。

如果OFPT_BUNDLE_CONTROL或OFPT_BUNDLE_ADD_MESSAGE消息为这些消息指定了不同的标志
交换机必须已经在现有捆绑包上指定了它，但它必须拒绝该请求并发送ofp_error_msg
具有OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_FLAGS代码。后者的例外是

OFPBF_TIME标志，在捆绑包提交请求消息中可能具有与先前不同的值
捆绑消息，如7.3.9.3所述。

如果交换机未收到任何OFPT_BUNDLE_CONTROL或OFPT_BUNDLE_ADD_MESSAGE消息
交换机定义的时间大于1s的打开的bundle_id，可能会发送带有以下内容的ofp_error_msg：

OFPET_BUNDLE_FAILED类型和OFPBFC_TIMEOUT代码。如果交换机没有收到任何新消息，
交换机中的鼠尾草定义的时间大于1s时，它可能会发送带有以下内容的ofp_error_msg：

OFPET_BUNDLE_FAILED类型和OFPBFC_TIMEOUT代码。

如果由于捆绑包锁定资源而无法处理OpenFlow消息，则此消息为
使用时，交换机必须拒绝该消息并发送带有OFPET_BUNDLE_FAILED的ofp_error_msg
类型和OFPBFC_BUNDLE_IN_PROGRESS代码。如果交换机之间无法处理其他消息
打开一个包并提交或丢弃它，交换机必须拒绝该消息，并且
发送具有OFPET_BUNDLE_FAILED类型和OFPBFC_BUNDLE_IN_PROGRESS代码的ofp_error_msg。

如果交换机收到调度的提交但无法执行，则会发送一条错误消息，其中包含
类型OFPET_BUNDLE_FAILED。以下错误代码特定于计划的捆绑包：

- OFPBFC_SCHED_NOT_SUPPORTED-当交换机不支持计划的时使用此代码
  捆绑执行并接收带有OFPBF_TIME标志设置的提交消息。

- OFPBFC_SCHED_FUTURE-在交换机收到预定的提交消息并且调度时间超过sched_max_future（请参阅第6.9.6.4节）。
- OFPBFC_SCHED_PAST-在交换机收到预定的提交消息并且预定的时间超过sched_max_past（请参阅第6.9.6.4节）。

7.3.10设置异步配置消息

交换机管理每个控制器的异步配置，该配置定义了异步
想要在给定的OpenFlow通道上接收的消息（错误消息除外）。的
控制器能够使用OFPT_SET_ASYNC设置和查询其异步配置，并且

分别为OFPT_GET_ASYNC_REQUEST条消息。交换机响应OFPT_GET_ASYNC_REQUEST
带有OFPT_GET_ASYNC_REPLY消息的消息；它不会回复设置配置的请求
tion。

OFPT_SET_ASYNC消息设置控制器是否应接收给定的异步消息
由交换机生成。其他OpenFlow功能控制给定消息是否生成
兴高采烈 例如，每个流条目中的OFPFF_SEND_FLOW_REM标志控制是否切换
删除流表项时，生成OFPT_FLOW_REMOVED消息。异步配置
tion作为每个控制器的附加过滤器；例如，它定义特定的控制器是否接收到那些

OFPT_FLOW_REMOVED条消息。

使用OFPT_SET_ASYNC消息设置的配置特定于特定的OpenFlow通道。它
不会影响任何其他OpenFlow渠道，无论是当前建立的还是即将建立的
未来。使用OFPT_SET_ASYNC设置的配置不会过滤或以其他方式影响错误消息。

交换机配置（例如使用OpenFlow配置协议）可以设置初始配置
建立OpenFlow连接时配置异步消息。离席期间
对于这种交换机配置，初始配置应为：

- 以“主”或“相等”角色，启用所有OFPT_PACKET_IN消息，但那些消息除外由于OFPR_INVALID_TTL的原因，启用所有OFPT_PORT_STATUS，OFPT_FLOW_REMOVED，OFPT_CONTROLLER_STATUS 讯息， 和 禁用 所有 OFPT_ROLE_STATUS 和OFPT_REQUESTFORWARD消息。
- 在“从站”角色中，启用所有OFPT_PORT_STATUS和OPFT_CONTROLLER_STATUS消息，以及禁用所有OFPT_PACKET_IN，OFPT_FLOW_REMOVED，OFPT_ROLE_STATUS和OFPT_REQUESTFORWARD消息。

除了OpenFlow标头之外，OFPT_GET_ASYNC_REQUEST没有其他正文。OFPT_SET_ASYNC
和OFPT_GET_ASYNC_REPLY消息具有以下格式：

```
/ *异步消息配置。* /
struct ofp_async_config {
        struct ofp_header标头；                    / * OFPT_GET_ASYNC_REPLY或OFPT_SET_ASYNC。* /

        / *异步配置属性列表-0或更多* /
        p_async_config_prop_header属性的结构[0]；
};
OFP_ASSERT（sizeof（p_async_config的结构）== 8）；
```


属性字段是异步配置属性的列表，描述了是否有各种不同的
是否启用异步消息。在OFPT_SET_ASYNC中，仅应为
需要包含已更改的内容，消息中省略的属性不变。在

OFPT_GET_ASYNC_REPLY消息，必须包括所有属性。

当前定义的异步配置类型为：

```
/ *异步配置属性类型。
 *低位清零表示从属角色的属性。
 *低位设置指示主/等角色的属性。
 * /
of_p_async_config_prop_type的枚举{
    OFPACPT_PACKET_IN_SLAVE          = 0，/ *从站的入站掩码。* /
    OFPACPT_PACKET_IN_MASTER         = 1，/ *主服务器的入站掩码。* /
    OFPACPT_PORT_STATUS_SLAVE        = 2，/ *从站的端口状态掩码。* /
    OFPACPT_PORT_STATUS_MASTER       = 3，/ *主设备的端口状态掩码。* /
    OFPACPT_FLOW_REMOVED_SLAVE       = 4，/ *删除了从站的流屏蔽。* /
    OFPACPT_FLOW_REMOVED_MASTER      = 5，/ *主机的流移除掩码。* /
    OFPACPT_ROLE_STATUS_SLAVE        = 6，/ *从站的角色状态掩码。* /
    OFPACPT_ROLE_STATUS_MASTER       = 7，/ *主服务器的角色状态掩码。* /
    OFPACPT_TABLE_STATUS_SLAVE       = 8，/ *从站的表状态掩码。* /
    OFPACPT_TABLE_STATUS_MASTER      = 9，/ *主服务器的表状态掩码。* /
    OFPACPT_REQUESTFORWARD_SLAVE     = 10，/ *从站的RequestForward掩码。* /
    OFPACPT_REQUESTFORWARD_MASTER    = 11，/ *主服务器的RequestForward掩码。* /
    OFPACPT_FLOW_STATS_SLAVE         = 12，/ *从站的流量统计信息掩码。* /
    OFPACPT_FLOW_STATS_MASTER        = 13，/ *主设备的流量统计信息掩码。* /
    OFPACPT_CONT_STATUS_SLAVE        = 14，/ *从站的控制器状态掩码。* /
    OFPACPT_CONT_STATUS_MASTER       = 15，/ *主控制器的控制器状态掩码。* /
    OFPACPT_EXPERIMENTER_SLAVE       = 0xFFFE，/ *从器件的实验器。* /
    OFPACPT_EXPERIMENTER_MASTER      = 0xFFFF，/ *实验者为主。* /
};
```

每种属性类型都控制控制器是否接收带有特定内容的异步消息

ofp_type的枚举。带有_MASTER后缀的属性在控制器时指定感兴趣的消息
具有OFPCR_ROLE_EQUAL或OFPCR_ROLE_MASTER角色（请参阅6. 3.7）。带有_SLAVE后缀的属性
指定当控制器具有OFPCR_ROLE_SLAVE角色时感兴趣的消息。

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有异步配置属性的通用头* /
p_async_config_prop_header的结构{
    uint16_t                类型;       / * OFPACPT_ *之一。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_async_config_prop_header的结构）== 4）；
```

的 OFPACPT_PACKET_IN_SLAVE， OFPACPT_PACKET_IN_MASTER， OFPACPT_PORT_STATUS_SLAVE，
OFPACPT_PORT_STATUS_MASTER， OFPACPT_FLOW_REMOVED_SLAVE，OFPACPT_FLOW_REMOVED_MASTER，
OFPACPT_ROLE_STATUS_SLAVE和OFPACPT_ROLE_STATUS_MASTER，OFPACPT_TABLE_STATUS_SLAVE，
OFPACPT_TABLE_STATUS_MASTER，OFPACPT_REQUESTFORWARD_SLAVE，OFPACPT_REQUESTFORWARD_MASTER，
OFPACPT_FLOW_STATS_SLAVE， OFPACPT_FLOW_STATS_MASTER， OPFT_CONT_STATUS_SLAVE 和
OPFT_CONT_STATUS_MASTER属性使用以下结构和字段：


```
/ *基于各种原因的属性* /
p_async_config_prop_reasons的结构{
    uint16_t                类型;       / * OFPACPT_PACKET_IN_ *中的一个，
                                          OFPACPT_PORT_STATUS_ *，
                                          OFPACPT_FLOW_REMOVED_ *，
                                          OFPACPT_ROLE_STATUS_ *，
                                          OFPACPT_TABLE_STATUS_ *，
                                          OFPACPT_REQUESTFORWARD_ *，
                                          OFPACPT_FLOW_STATS_ *，
                                          OFPACPT_CONT_STATUS_ *。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
    uint32_t                面具;       / *原因值的位掩码。* /
};
OFP_ASSERT（sizeof（p_async_config_prop_reasons的结构）== 8）；
```

掩码字段是位掩码，其中0位禁止接收使用原因码发送的消息
对应于位索引和1位使能接收它。例如，如果属性类型为

OFPACPT_PORT_STATUS_SLAVE，值为2的位 掩码中的 2 = 4决定控制器是否将
当控制器具有角色时，接收原因为OFPPR_MODIFY（值2）的OFPT_PORT_STATUS消息

OFPCR_ROLE_SLAVE。

OFPACPT_EXPERIMENTER_SLAVE和OFPACPT_EXPERIMENTER_MASTER属性使用以下内容
结构和字段：

```
/ *实验者异步配置属性* /
p_async_config_prop_experimenter的结构{
    uint16_t                类型;       / * OFPTFPT_EXPERIMENTER_SLAVE之一，
                                          OFPTFPT_EXPERIMENTER_MASTER。* /
    uint16_t                长度; / *此属性的长度（以字节为单位）。* /
```

```
    uint32_t                实验者 / *采用相同的实验者ID
                                                形式如结构
                                                ofp_experimenter_header。* /
    uint32_t                exp_type;           / *实验者定义。* /
    /* 其次是:
     *      -确切地（长度-12个）字节包含实验者数据，然后
     *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *        全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_async_config_prop_experimenter的结构）== 12）;
```

实验者字段是实验者ID，其格式与典型实验中的形式相同。
导师结构（请参见7.2.8）。

7.4异步消息

7.4.1包入消息

当数据路径接收到数据包并将其发送到控制器时，它们将使用OFPT_PACKET_IN
信息:


```
/ *在端口（数据路径->控制器）上接收到的数据包。* /
struct ofp_packet_in {
    struct ofp_header标头;
    uint32_t buffer_id;             / *由数据路径分配的ID。* /
    uint16_t total_len;             / *整个帧的长度。* /
    uint8_t原因;                    / *原因包被发送（OFPR_ *之一）* /
    uint8_t table_id;               / *查找的表的ID * /
    uint64_t cookie;                / *查找的流条目的Cookie。* /
    struct ofp_match match; / *数据包元数据。大小可变。* /
    / *变量大小和填充匹配始终后跟:
     *      -恰好2个全零填充字节，然后
     *      -以太网帧，其长度是根据header.length推断的。
     *以太网帧之前的填充字节可确保IP
     *以太网报头后面的报头（如果有）是32位对齐的。
     * /
    // uint8_t pad [2];             / *对齐64位+ 16位* /
    // uint8_t data [0];            / *以太网帧* /
};
OFP_ASSERT（sizeof（p_packet_in的结构）== 32）;
```

buffer_id是数据路径用来标识缓冲数据包的不透明值。如果包
与入站消息相关联的缓冲区被缓冲后，buffer_id必须是在
当前连接是指交换机上的该数据包。如果数据包没有缓冲-要么是因为
没有可用缓冲区，或者由于通过OFPCML_NO_BUFFER显式请求-buffer_id
必须为OFP_NO_BUFFER。

预计实现缓冲的开关将通过文档公开这两个数量
可用缓冲的时间长度以及可以重新使用缓冲区的时间长度。开关必须优雅地
处理缓冲的packet_in消息不产生来自控制器的响应的情况。开关
应该防止缓冲区重用，直到控制器处理了缓冲区或缓冲区有一定数量为止
时间（在文档中指示）已经过去。

total_len是触发入包消息的包的全长。实际长度
如果数据包已被截断，则消息中数据字段的值可能小于total_len
缓冲（请参阅6.1。2）。字段total_len在缓冲之前必须始终与数据包相对应
和截断。字段total_len计算为数据包定义的*数据包中*的字节数
键入（请参阅7.2.3.11）。

原因字段指示哪个上下文触发了打包消息，并且可以是以下任何一种
值:

```
/ *为什么将此数据包发送到控制器？* /
of_p_packet_in_reason枚举{
    OFPR_TABLE_MISS     = 0,    / *没有匹配的流（表缺失流条目）。* /
    OFPR_APPLY_ACTION = 1       / *在apply-actions中输出到控制器。* /
    OFPR_INVALID_TTL = 2        / *数据包具有无效的TTL * /
    OFPR_ACTION_SET     = 3,    / *输出到动作集中的控制器。* /
    OFPR_GROUP          = 4     / *输出到组存储区中的控制器。* /
    OFPR_PACKET_OUT     = 5     / *输出到包中的控制器。* /
};
```

OFPR_INVALID_TTL表示具有无效IP TTL或MPLS TTL的数据包已被拒绝
OpenFlow管道并传递给控制器。检查无效的TTL不需要

每个数据包都必须完成，但每次OFPAT_DEC_MPLS_TTL或

OFPAT_DEC_NW_TTL操作应用于数据包。

cookie字段包含导致数据包发送到控制器的流条目的cookie。
如果cookie无法与特定流关联，则必须将此字段设置为-1（0xffffffffffffffff）。对于
例如，如果打包是在组存储桶中或从操作集中生成的。

匹配字段是一组OXM TLV，其中包含与数据包关联的管道字段。的
无法从数据包数据确定流水线字段值，例如包括输入
端口和元数据值（请参阅7.2.3。9）。OXM TLV必须包括*数据包类型匹配字段*
标识数据字段中包括的数据包类型，除非数据包是以太网（请参阅7.2.3.11）。的
OXM TLV集合必须包括与该数据包关联的所有管道字段，并由交换机支持
并且哪个值不是全零。字段OXM_OF_ACTSET_OUTPUT应该从此省略
组。如果OXM_OF_IN_PHY_PORT与OXM_OF_IN_PORT具有相同的值，则应从中省略
组。OXM TLV集合可以可选地包括其值为全比特零的管线字段。套装
OXM TLV的数量还可以可选地包括数据包头字段。大多数开关不应该包括那些
可选字段，以最小化打包的大小，因此控制器不应依赖
它们的存在，应从数据字段中提取标头字段。OXM TLV集必须
反映触发包入消息的事件时，反映包的标头和上下文，
它们应包括先前处理过程中所做的所有修改。

OXM_OF_IN_PORT TLV引用的端口是用于匹配流的数据包入口端口
条目，并且必须是有效的标准OpenFlow端口（请参阅7.2.3.9）。引用的端口

OXM_OF_IN_PHY_PORT TLV是基础物理端口（请参见7 .2.3.9）。

pad字段是除了match字段的潜在填充以外的其他填充，以使
确保以太网帧的IP标头正确对齐。即使数据
字段为空。

数据字段包含触发了包入消息的包的一部分，该包是
完整包含在数据字段中，如果已缓冲，则将其截断。数据包类型由
*匹配字段中*包含的*数据包类型匹配*字段，如果不存在数据包类型匹配字段，则
数据包必须是以太网（以太网帧，包括以太网报头和以太网有效负载，但不能
以太网FCS-参见7.2.3.11）。数据包标头反映了先前应用于数据包的任何更改
处理中，但不包括操作集中的待处理更改。

当一个数据包被缓冲时，从数据包开始的一些字节将被包含在数据中
消息的字段。如果数据包是由于“发送到控制器”操作而发送的，则此数量的
字节必须等于触发消息的ofp_action_output中的max_len值。如果
出于其他原因（例如无效的TTL）发送数据包，则此字节数必须等于
使用OFPT_SET_CONFIG消息配置的miss_send_len值。默认的miss_send_len
是128个字节。如果数据包没有缓冲-是由于没有可用的缓冲区，或者是因为
通过OFPCML_NO_BUFFER明确请求-整个数据包必须包含在数据字段中。

7.4.2流删除消息

如果流条目超时或从表中删除时请求控制器通知
6.5），数据路径使用OFPT_FLOW_REMOVED消息执行此操作：


```
/ *流被删除（数据路径->控制器）。* /
struct ofp_flow_removed {
    struct ofp_header标头；
    uint8_t table_id;                / *表的ID * /
    uint8_t原因；                    / * OFPRR_ *之一。* /
    uint16_t优先级；                 / *流条目的优先级。* /
```

```
    uint16_t idle_timeout;          /*原始流模块的空闲超时。*/
    uint16_t hard_timeout;          /*从原始流模块硬超时。*/
    uint64_t cookie;                /*不透明的控制器发出的标识符。*/
    struct ofp_match match;         /*字段说明。大小可变。*/
    // struct ofp_stats统计信息；/*统计信息列表。大小可变。*/
};
OFP_ASSERT（sizeof（p_flow_removed的结构）== 32）；
```

匹配，cookie和优先级字段与流mod请求中使用的字段相同。

原因字段是以下内容之一：

```
/*为什么要删除此流程？*/
of_p_flow_removed_reason枚举{
    OFPRR_IDLE_TIMEOUT = 0，    /*流空闲时间超过了idle_timeout。*/
    OFPRR_HARD_TIMEOUT = 1      /*时间超出了hard_timeout。*/
    OFPRR_DELETE        = 2     /*由DELETE flow mod驱逐。*/
    OFPRR_GROUP_DELETE = 3，    /*组已删除。*/
    OFPRR_METER_DELETE = 4      /*仪表已卸下。*/
    OFPRR_EVICTION      = 5     /*将驱逐切换为自由资源。*/
};
```

到期过程使用原因值OFPRR_IDLE_TIMEOUT和OFPRR_HARD_TIMEOUT（请参见6.5）。当流条目被flow-mod请求删除时，使用原因值OFPRR_DELETE（请参见6.4）。当通过group-mod删除流条目时，将使用原因值OFPRR_GROUP_DELETE请求（请参阅6. 7）。原因值OFPRR_METER_DELETE在流条目被删除时使用。meter-mod请求（请参阅6.8）。删除流条目时使用原因值OFPRR_EVICTION驱逐过程（参见6.5）。

stats字段包含OXS字段的列表（请参阅 7.2。4.2 ）。字段OXS_OF_DURATION是必填字段，跟踪流条目已在交换机中安装多长时间。字段OXS_OF_PACKET_COUNT和

OXS_OF_BYTE_COUNT是可选的，指示关联的数据包和字节数分别使用此流条目。

idle_timeout和hard_timeout字段是直接从创建此文件的流mod中复制的条目。使用上面的两个字段和OXS_OF_DURATION，可以找到流的时间量条目处于活动状态，以及流条目接收流量的时间。

7.4.3端口状态消息

在添加，修改和从数据路径中删除端口时，需要通知控制器OFPT_PORT_STATUS消息：


```
/*数据路径中的物理端口已更改*/
struct ofp_port_status {
    struct ofp_header标头；
    uint8_t原因；                   /* OFPPR_*之一。*/
    uint8_t pad [7];                /*对齐64位。*/
    struct ofp_port desc;
};
OFP_ASSERT（sizeof（p_port_status的结构）== 56）；
```

原因字段可以是以下值之一：

```
/*关于物理端口的更改*/
of_p_port_reason枚举{
    OFPPR_ADD       = 0，       /*端口已添加。*/
    OFPPR_DELETE = 1            /*端口已删除。*/
    OFPPR_MODIFY = 2            /*端口的某些属性已更改。*/
};
```

原因值OFPPR_ADD表示数据路径中不存在并已添加的端口。原因值OFPPR_DELETE表示已从数据路径中删除且不再存在的端口存在。原因值OFPPR_MODIFY表示状态或配置已更改的端口（请参见7 .2.1）。

desc字段是端口说明（请参见7.2.1）。在端口描述中，不需要所有属性包括在内，仅必须包含已更改的那些属性，而不变的属性可以选择包括在内。

7.4.4控制器角色状态消息

如果某个控制器的角色由交换机更改，而不是由该控制器使用

OFPT_ROLE_REQUEST消息，必须通过OFPT_ROLE_STATUS通知相应的控制器
信息。交换机可能在其他时间生成OFPT_ROLE_STATUS消息，例如
实验者数据更改，这不在本规范的范围之内。

OFPT_ROLE_STATUS消息使用以下结构和字段：

```
/ *角色状态事件消息。* /
struct ofp_role_status {
        struct ofp_header标头；                / *输入OFPT_ROLE_STATUS。* /
        uint32_t角色；                         / * OFPCR_ROLE_ *之一。* /
        uint8_t原因；                          / * OFPCRR_ *之一。* /
        uint8_t pad [3];                       / *对齐64位。* /
        uint64_t generation_id;                / *主选举产生ID * /

        / *角色属性列表* /
        p_role_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_role_status）== 24）；
```

原因可能是以下值之一：


```
/ *关于控制器角色的更改* /
of_p_controller_role_reason的枚举{
        OFPCRR_MASTER_REQUEST = 0，/ *另一个控制器要求是主控制器。* /
        OFPCRR_CONFIG              = 1，/ *交换机上的配置已更改。* /
        OFPCRR_EXPERIMENTER    = 2，/ *实验者数据已更改。* /
};
```

该角色是控制器的新角色（参见7.3 .8）。generation_id是生成ID
包含在触发角色更改的角色请求消息中。

属性字段是角色属性的列表，描述了表配置的动态参数。

当前定义的角色属性类型的列表是：

```
/ *角色属性类型。
  * /
of_p_role_prop_type的枚举{
        OFPRPT_EXPERIMENTER                = 0xFFFF，/ *实验者属性。* /
};
```

属性定义包含属性类型，长度和任何关联的数据：

```
/ *所有角色属性的通用标头* /
struct ofp_role_prop_header {
        uint16_t                类型；          / * OFPRPT_ *之一。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（struct ofp_role_prop_header）== 4）；
```

OFPTMPT_EXPERIMENTER属性使用以下结构和字段：

```
/ *实验者角色属性* /
struct ofp_role_prop_experimenter {
        uint16_t                类型；          / * OFPRPT_EXPERIMENTER之一。* /
        uint16_t                长度; / *此属性的长度（以字节为单位）。* /
        uint32_t                实验者 / *采用相同的实验者ID
                                                形式如结构
                                                ofp_experimenter_header。* /
        uint32_t                exp_type;               / *实验者定义。* /
        /* 其次是：
          *     -确切地（长度-12个）字节包含实验者数据，然后
          *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
          *         全零字节的字节* /
        uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_role_prop_experimenter的结构）== 12）；
```

实验者字段是实验者ID）其格式与典型实验中的形式相同。
等师结构（请参见7.2.3）。

7.4.5表状态消息

当表状态更改时，需要通过OFPT_TABLE_STATUS消息通知控制器-
智者：

```
/ *表配置已在数据路径中更改* /
struct ofp_table_status {
        struct ofp_header标头；
        uint8_t原因；                    / * OFPTR_ *之一。* /
        uint8_t pad [7];                 / *填充至64位* /
        struct ofp_table_desc表；                / *新表配置。* /
};
OFP_ASSERT（sizeof（struct ofp_table_status）== 24）；
```

原因可能是以下值之一：

```
/ *表格发生了什么变化* /
of_p_table_reason枚举{
        OFPTR_VACANCY_DOWN = 3，          / *空缺人数减少阈值事件。* /
        OFPTR_VACANCY_UP     = 4          / *空缺阈值事件。* /
};
```

原因值OFPTR_VACANCY_DOWN和OFPTR_VACANCY_UP标识空缺消息。的
当流表中的剩余空间更改并超过一个时，将生成空缺事件
空缺阈值的最大值（请参见7.3.4.1），无论引起剩余空间变化的原因是什么。这个变化
剩余空间可能是表上OpenFlow操作的结果（流插入，到期或
移除），或切换内部处理的结果更改了
表。

7.4.6请求转发消息

当控制器修改组和仪表的状态时，成功修改
这种状态可能会转发给其他控制器。其他控制器被告知
OFPT_REQUESTFORWARD讯息：

```
/ *组/仪表请求转发。* /
struct ofp_requestforward_header {
        struct ofp_header标头；          / *输入OFPT_REQUESTFORWARD。* /
        struct ofp_header请求；/ *正在转发请求。* /
};
OFP_ASSERT（sizeof（p_requestforward_header的结构）== 16）；
```

使用Set Asynchronous在每个控制器通道上启用请求转发
配置消息（请参阅7.3.10）。以下转发原因可能会在
requestforward_mask字段：

```
/ *请求转发原因* /
ofp_requestforward_reason枚举{
        OFPRFR_GROUP_MOD = 0，           / *转发组mod请求。* /
        OFPRFR_METER_MOD = 1             / *转发仪表Mod请求。* /
};
```


7.4.7控制器状态消息

当控制器的状态由于以下任何原因而改变时，交换机必须发送
控制器状态消息发送给所有连接的控制器。

```
/ *为什么要报告控制器状态？ * /
of_p_controller_status_reason的枚举{
    OFPCSR_REQUEST              = 0，/ *控制器请求的状态。 * /
    OFPCSR_CHANNEL_STATUS       = 1，/ *通道的运行状态已更改。 * /
    OFPCSR_ROLE                 = 2，/ *控制器角色已更改。 * /
    OFPCSR_CONTROLLER_ADDED     = 3，/ *添加了新控制器。 * /
    OFPCSR_CONTROLLER_REMOVED = 4，/ *控制器已从配置中删除。 * /
    OFPCSR_SHORT_ID             = 5，/ *控制器ID已更改。 * /
    OFPCSR_EXPERIMENTER         = 6，/ *实验者数据已更改。 * /
};
```

使用Set Asynchronous Con-（设置异步同步）可在每个控制器通道上启用控制通道更新。形象化消息（见7.3.10）。原因代码OFPCSR_REQUEST在多部分消息中使用（请参阅7.3.5.15），并且不应使用此原因码生成异步消息。开关应为此，请忽略异步配置。

控制器状态消息包含OpenFlow标头和一个控制器状态结构，代表发送带有原因码的已更改控制器的当前状态。如果单个事件导致状态要更改的多个控制器的数量（例如，网络链接的故障，可以通过该链接进行多个控制通道运行），则交换机必须为每个受影响的控制器发送不同的消息。

```
struct ofp_controller_status_header {
    struct ofp_header标头；                    / *输入OFPT_CONTROLLER_STATUS。 * /
    p_controller_status状态的结构；            / *控制器状态。 * /
};
```

如果从交换机的配置中删除了控制器，则交换机必须将控制器状态发送到其余所有已连接的控制器，其原因设置为OFPCSR_CONTROLLER_REMOVED。控制器对于已删除的控制器，应忽略role和channel_status的值。

控制器状态结构在7.2.7中描述。

7.5对称消息

7.5.1你好

OFPT_HELLO消息由一个OpenFlow标头和一组可变大小的hello元素组成。

```
/ * OFPT_HELLO。此消息包含零个或多个具有的hello元素
  *可变大小。未知元素类型必须被忽略/跳过，以允许
  *用于将来的扩展。 * /
struct ofp_hello {
    struct ofp_header标头；
```



```
    / *你好元素列表* /
    p_hello_elem_header元素的结构[0]; / *元素列表-0或更多* /
};
OFP_ASSERT（sizeof（struct ofp_hello）== 8）；
```

标头字段的version字段部分（请参见7.1。1）必须设置为最高的OpenFlow开关发送方支持的协议版本（请参见6 .3.3）。

elements字段是一组hello元素，其中包含可选数据以通知初始握手连接的。实现必须忽略（跳过）它们未包含的Hello消息的所有元素支持。当前定义的hello元素类型的列表为：

```
/ * Hello元素类型。
  * /
of_p_hello_elem_type的枚举{
    OFPHET_VERSIONBITMAP            = 1，/ *支持版本的位图。 * /
};
```

元素定义包含元素类型，长度和任何关联的数据：

```
/ *所有Hello元素的通用标头* /
struct ofp_hello_elem_header {
    uint16_t        类型；      / * OFPHET_ *之一。 * /
    uint16_t        长度; / *元素的长度（以字节为单位），
```

```
                                        包括此标头，不包括填充。* /
};
OFP_ASSERT（sizeof（struct ofp_hello_elem_header）== 4）；
```

OFPHET_VERSIONBITMAP元素使用以下结构和字段：

```
/ *版本位图Hello元素* /
struct ofp_hello_elem_versionbitmap {
        uint16_t                类型；          / * OFPHET_VERSIONBITMAP。* /
        uint16_t                长度; / *此元素的长度（以字节为单位），
                                        包括此标头，不包括填充。* /
        /* 其次是：
         *      -恰好（长度-4）个字节包含位图，然后
         *      -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
         *          全零字节的字节* /
        uint32_t                位图[0];                / *位图列表-支持的版本* /
};
OFP_ASSERT（sizeof（p_hello_elem_versionbitmap的结构== 4）；
```

位图字段指示设备支持的OpenFlow交换协议的版本集，
并且可以在版本协商期间使用（请参见6.3.3）。位图集的位被索引
通过协议的*版本*号; 如果由左移位数确定的位等于
设置为*off版本*号，则支持此OpenFlow版本。包含的位图数量
字段中的取决于所支持的最高版本号：*ofp版本*0到31编码为
第一个位图，*版本* 32到63，被编码在第二个位图中，依此类推。例如，
如果交换机仅支持1.0版（*ofp版本* = 0x01）和1.3版（*ofp版本* = 0x04），则第一个
位图将设置为0x00000012。

7.5.2回声请求

回声请求消息包含一个OpenFlow头以及一个任意长度的数据字段。的
数据字段可能是用于检查延迟的消息时间戳，用于测量带宽的各种长度或
零尺寸，以验证开关和控制器之间的活动性。

7.5.3回声回复

回声回复消息由OpenFlow标头以及回声的未修改数据字段组成
请求消息。

在分为多个层的OpenFlow交换协议实现中，回显请求/答复
逻辑应该在“最深”的实践层中实现。例如，在OpenFlow参考中
包含中继到内核模块的用户空间进程的实现，回显请求/回复是
在内核模块中实现。接收到格式正确的回声回复后，显示出更大的
正确的端到端功能的可能性（如果在
用户空间流程，以及提供更准确的端到端延迟时间。

7.5.4错误信息

交换机或控制器使用错误消息将问题通知给设备的另一侧。
连接。交换机通常使用它们来指示由服务器发起的请求失败。
控制器。

OpenFlow消息中的错误条件，例如无效的字段值或不受支持的功能
大多数情况下必须返回错误消息，而在其他情况下则应将其忽略（请参见7 .1.3）。如果有
是多个错误代码，可以应用于特定的错误情况，如果该错误的错误代码
在此文档中未指定条件，则必须是最具体和适当的错误代码
用过的。如果一条消息包含多个错误，则这些错误之一的一条错误消息必须
被退回。不同的开关将以不同的顺序评估消息字段，因此控制器
存在多个错误时，不应期望使用特定的错误代码。

OFPT_ERROR_MSG消息使用以下结构：

```
/ * OFPT_ERROR：错误消息。* /
struct ofp_error_msg {
        struct ofp_header标头；

        uint16_t类型;
        uint16_t代码;
        uint8_t data [0];                       / *可变长度数据。口译依据
                                                在类型和代码上。没有填充。* /
};
```

```
OFP_ASSERT（sizeof（struct ofp_error_msg）== 12）；
```

类型字段指示错误的高级类型。代码字段根据类型进行解释，
并指示高级错误类型内的确切错误。有效类型值和代码的列表
每种类型的值如下。

字段数据的长度可变，并根据类型和代码进行解释。除非特别说明
否则，数据字段必须至少包含导致错误的失败消息的64个字节
要生成的消息，如果失败的消息少于64个字节，则必须包含完整的消息
没有任何填充。如果请求是一个多部分请求，可以跨越多个多部分消息
（即定义为结构数组-参见7.3.5），如果错误指定数据字段包含
失败请求的一部分，则数据字段必须包含多部分消息的一部分，其中
发现错误，或者为空。

如果错误消息是对控制器发出的特定消息的响应，则该错误消息的xid字段
错误消息中的标头必须与有问题的消息中的标头匹配。如果错误是由
处理请求时，错误消息将代替回复，而普通回复将发送给
请求未发送。

以_EPERM结尾的错误代码对应于例如OpenFlow生成的权限错误
管理程序插入控制器和交换机之间。

当前定义的错误类型是：

```
/ * ofp_error_message中“类型”的值。这些值是不变的：它们
  *在协议的未来版本中不会更改（尽管可能会增加新值
  *添加）。* /
of_p_error_type的枚举{
        OFPET_HELLO_FAILED              = 0，/ * Hello协议失败。* /
        OFPET_BAD_REQUEST               = 1，/ *无法理解请求。* /
        OFPET_BAD_ACTION                = 2，/ *动作描述错误。* /
        OFPET_BAD_INSTRUCTION           = 3，/ *指令列表中的错误。* /
        OFPET_BAD_MATCH                 = 4，/ *匹配错误。* /
        OFPET_FLOW_MOD_FAILED           = 5，/ *修改流条目时出现问题。* /
        OFPET_GROUP_MOD_FAILED          = 6，/ *修改组条目时出现问题。* /
        OFPET_PORT_MOD_FAILED           = 7，/ *端口mod请求失败。* /
        OFPET_TABLE_MOD_FAILED          = 8，/ *表mod请求失败。* /
        OFPET_QUEUE_OP_FAILED           = 9，/ *队列操作失败。* /
        OFPET_SWITCH_CONFIG_FAILED = 10，/ *切换配置请求失败。* /
        OFPET_ROLE_REQUEST_FAILED = 11，/ *控制器角色请求失败。* /
        OFPET_METER_MOD_FAILED          = 12，/ *仪表错误。* /
        OFPET_TABLE_FEATURES_FAILED = 13，/ *设置表格功能失败。* /
        OFPET_BAD_PROPERTY              = 14，/ *某些属性无效。* /
        OFPET_ASYNC_CONFIG_FAILED = 15，/ *异步配置请求失败。* /
        OFPET_FLOW_MONITOR_FAILED = 16，/ *设置流量监控器失败。* /
        OFPET_BUNDLE_FAILED             = 17，/ *捆绑操作失败。* /
        OFPET_EXPERIMENTER = 0xffff                 / *实验者错误消息。* /
};
```

7.5.4.1 Hello失败错误代码

对于OFPET_HELLO_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_HELLO_FAILED的p_error_msg'代码'值。“数据”包含一个
  *可能提供故障详细信息的ASCII文本字符串。* /
of_p_hello_failed_code {
        OFPHFC_INCOMPATIBLE = 0，       / *没有兼容的版本。* /
        OFPHFC_EPERM            = 1     / *权限错误。* /
};
```


数据字段包含一个ASCII文本字符串，可添加有关错误发生原因的详细信息。

如果通过OFPT_HELLO消息协商的OpenFlow协议版本不受支持，
收件人，则收件人必须回复OFPT_ERROR消息，类型字段为
OFPET_HELLO_FAILED、OFPHFC_INCOMPATIBLE的代码字段以及可选的ASCII字符串，用于说明
数据中的情况。

7.5.4.2错误的请求错误代码

对于OFPET_BAD_REQUEST错误类型，当前定义了以下代码：

```
/ * OFPET_BAD_REQUEST的p_error_msg'代码'值。“数据”至少包含
  *失败请求的前64个字节。* /
of_p_bad_request_code的枚举{
    OFPBRC_BAD_VERSION          = 0，不支持/ * ofp_header.version。* /
    OFPBRC_BAD_TYPE             = 1，/ * ofp_header.type不支持。* /
    OFPBRC_BAD_MULTIPART        = 2，/ * ofp_multipart_request.type不是
                                    支持的。* /
    OFPBRC_BAD_EXPERIMENTER = 3，/ *不支持实验者ID
                                    *（在ofp_experimenter_header中或
                                    * ofp_multipart_request或
                                    * ofp_multipart_reply）。* /
    OFPBRC_BAD_EXP_TYPE         = 4，/ *不支持实验者类型。* /
    OFPBRC_EPERM                = 5，/ *权限错误。* /
    OFPBRC_BAD_LEN              = 6，/ *类型的请求长度错误。* /
    OFPBRC_BUFFER_EMPTY         = 7，/ *已使用指定的缓冲区。* /
    OFPBRC_BUFFER_UNKNOWN = 8，/ *指定的缓冲区不存在。* /
    OFPBRC_BAD_TABLE_ID         = 9，/ *指定的表ID无效或无效
                                    *存在。* /
    OFPBRC_IS_SLAVE             = 10，/ *由于控制器是从属设备而被拒绝。* /
    OFPBRC_BAD_PORT             = 11，/ *无效的端口或缺少的端口。* /
    OFPBRC_BAD_PACKET           = 12，/ *无效的数据包输出。* /
    OFPBRC_MULTIPART_BUFFER_OVERFLOW   = 13，/ * ofp_multipart_request
                                    溢出分配的缓冲区。* /
    OFPBRC_MULTIPART_REQUEST_TIMEOUT = 14，/ *多部分时超时
                                    请求。* /
    OFPBRC_MULTIPART_REPLY_TIMEOUT = 15，/ *分段答复期间超时。* /
    OFPBRC_MULTIPART_BAD_SCHED = 16，/ *交换机收到OFPMP_BUNDLE_FEATURES
                                    请求，但无法更新
                                    计划公差。* /
    OFPBRC_PIPELINE_FIELDS_ONLY = 17，/ *匹配字段必须仅包含
                                    管道字段。* /
    OFPBRC_UNKNOWN              = 18，/ *未指定错误。* /
};
```

错误 信息 使用 的 码 OFPBRC_MULTIPART_BUFFER_OVERFLOW，
OFPBRC_MULTIPART_REQUEST_TIMEOUT和OFPBRC_MULTIPART_REPLY_TIMEOUT必须包含一个
数据字段为空，并且xid字段必须与有问题的多部分消息序列的字段匹配。

如果角色设置为*从动*的控制器向交换机发送控制器到交换机的命令，该命令发送
数据包或修改交换机的状态，则该交换机必须以OFPT_ERROR消息答复，并带有
OFPET_BAD_REQUEST的类型字段，OFPBRC_IS_SLAVE的代码字段。

如果*Packet-Out*消息中的match字段包含任何报头字段，则交换机必须拒绝重新
搜寻并发送OFPET_BAD_REQUEST类型和OFPBRC_PIPELINE_FIELDS_ONLY的ofp_error_msg
码。

如果*Packet-Out*消息的匹配字段中未包含输入端口，则交换机必须拒绝
请求并发送具有OFPET_BAD_REQUEST类型和OFPBRC_BAD_PORT代码的ofp_error_msg。

7.5.4.3错误操作错误代码

对于OFPET_BAD_ACTION错误类型，当前定义了以下代码：

```
/ * OFPET_BAD_ACTION的p_error_msg'代码'值。“数据”至少包含
  *失败请求的前64个字节。* /
of_p_bad_action_code的枚举{
    OFPBAC_BAD_TYPE             = 0，/ *未知或不受支持的操作类型。* /
    OFPBAC_BAD_LEN              = 1，/ *动作中的长度问题。* /
    OFPBAC_BAD_EXPERIMENTER     = 2，/ *指定了未知的实验者ID。* /
    OFPBAC_BAD_EXP_TYPE         = 3，/ *实验者ID的未知操作。* /
    OFPBAC_BAD_OUT_PORT         = 4，/ *验证输出端口时出现问题。* /
    OFPBAC_BAD_ARGUMENT         = 5，/ *错误动作参数。* /
    OFPBAC_EPERM                = 6，/ *权限错误。* /
    OFPBAC_TOO_MANY             = 7，/ *无法处理这么多动作。* /
```

```
    OFPBAC_BAD_QUEUE                = 8，/ *验证输出队列时出现问题。* /
    OFPBAC_BAD_OUT_GROUP            = 9，/ *群组操作中无效的群组ID。* /
    OFPBAC_MATCH_INCONSISTENT = 10，/ *无法为该比赛申请动作，
                                      或缺少设置字段的先决条件。* /
    OFPBAC_UNSUPPORTED_ORDER = 11，/ *不支持操作顺序
                                   Apply-Actions指令中的动作列表* /
    OFPBAC_BAD_TAG                  = 12，/ *操作使用不受支持的
                                      标签/封套。* /
    OFPBAC_BAD_SET_TYPE             = 13，/ * SET_FIELD操作中不受支持的类型。* /
    OFPBAC_BAD_SET_LEN              = 14，/ * SET_FIELD动作中的长度问题。* /
    OFPBAC_BAD_SET_ARGUMENT = 15，/ * SET_FIELD操作中的错误参数。* /
    OFPBAC_BAD_SET_MASK             = 16，/ * SET_FIELD操作中的错误掩码。* /
    OFPBAC_BAD_METER                = 17，/ *仪表操作中无效的仪表ID。* /
};
```

如果消息指定的操作无效或在指定的上下文中不受支持（表，
组，数据包输出），交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，

OFPBAC_BAD_TYPE代码。

如果*Write-Actions*指令中包含的一组动作包含多个以下实例
操作类型或设置字段操作类型，该开关可以选择返回一个ofp_error_msg，其中包含

OFPET_BAD_ACTION类型和OFPBAC_TOO_MANY代码。

如果*Apply-Actions*指令中包含的动作列表包含的动作多于
交换机支持，则交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，

OFPBAC_TOO_MANY代码。

如果操作列表包含交换机无法按指定顺序支持的一系列操作，则
开关必须返回OFOF_BAD_ACTION类型和OFPBAC_UNSUPPORTED_ORDER的ofp_error_msg
码。

如果任何*操作*引用的端口在交换机上永远无效，则该交换机必须返回一个

OFPET_BAD_ACTION类型和OFPBAC_BAD_OUT_PORT代码的ofp_error_msg。如果引用的端口
可能在将来有效，例如，当将线卡添加到机箱交换机或端口动态
添加到软件交换机后，交换机必须要么静默丢弃发送到参考端口的数据包，要么
立即返回OFPBAC_BAD_OUT_PORT错误并拒绝流程mod。

如果消息中的*操作*引用了交换机上当前未定义或保留的组
组，例如OFPG_ALL，交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，

OFPBAC_BAD_OUT_GROUP代码。

如果消息中的*操作*引用了交换机上当前未定义或保留的仪表
仪表，例如OFPM_ALL，交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，并且

OFPBAC_BAD_METER代码。

如果消息中的某个*操作*的值无效，例如带有无效以太币的Push操作，
类型，并且这种情况不会被其他错误代码所覆盖（例如*错误的输出端口*、*错误的组ID*）
或*设置参数错误*），则交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，

OFPBAC_BAD_ARGUMENT代码。

如果消息中的某个*动作*执行的操作与匹配和先前的动作不一致
相同消息的消息，例如，*弹出VLAN*操作，其中匹配项没有指定VLAN，或者是一*组*

使用以通配符以太类型进行匹配的*TTL*动作，交换机可以选择拒绝flow mod和
立即返回OFPET_BAD_ACTION类型和OFPBAC_MATCH_INCONSISTENT的ofp_error_msg
码。

如果消息中的*设置字段*操作的值无效，例如OXM的设置字段操作

如果VLAN_VID的值大于4095，或者DSCP值使用的位数超过6位，则交换机必须
返回具有OFPET_BAD_ACTION类型和OFPBAC_BAD_SET_ARGUMENT代码的ofp_error_msg。

如果流mod消息中的*设置字段*操作未包含在匹配项中，或者
同一条消息中的先前操作，例如，具有匹配通配符的设置的IPv4地址操作
以太网类型，交换机必须拒绝流mod并立即返回一个ofp_error_msg

OFPET_BAD_ACTION类型和OFPBAC_MATCH_INCONSISTENT代码。任何不一致动作的影响
匹配的数据包上未定义。强烈建议控制器避免产生组合
可能产生不一致动作的表条目数。

如果*set-field*操作或*Copy-Field*操作指定了指定ta-中不支持的字段或类，
ble，交换机必须返回OFPET_BAD_ACTION类型和OFPBAC_BAD_SET_TYPE的ofp_error_msg
码。

如果“ *设置字段*”操作包括OXM掩码（设置了oxm_hasmask字段），并且该开关不支持
对于这种OXM类型的端口屏蔽，交换机必须拒绝流量模块并立即返回

OFPET_BAD_ACTION类型和OFPBMC_BAD_SET_MASK代码的ofp_error_msg。

如果*Set-Field*操作中的OXM字段包含掩码，则oxm_mask中的0位对应的位
如果oxm_value为1位，则交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，并且
OFPBAC_BAD_SET_MASK代码。

如果“ *复制字段*”操作指定了无效的偏移量，则该开关必须返回带有以下内容的ofp_error_msg：

OFPET_BAD_ACTION类型和OFPBAC_BAD_SET_ARGUMENT代码。

如果“ *复制字段*”操作指定了源和目标重叠，并且该开关不支持
端口进行此类操作，交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，并且

OFPBAC_BAD_SET_TYPE代码。

7.5.4.4错误的指令错误代码

对于OFPET_BAD_INSTRUCTION错误类型，当前定义了以下代码：

```
/ * OFPET_BAD_INSTRUCTION的p_error_msg'代码'值。“数据”包含在
  *至少失败请求的前64个字节。* /
of_p_bad_instruction_code的枚举{
        OFPBIC_UNKNOWN_INST         = 0，/ *未知指令。* /
        OFPBIC_UNSUP_INST           = 1，/ *开关或表不支持
                                          指令。* /
        OFPBIC_BAD_TABLE_ID         = 2，/ *指定了无效的表ID。* /
        OFPBIC_UNSUP_METADATA       = 3，/ *数据路径不支持元数据值。* /
        OFPBIC_UNSUP_METADATA_MASK = 4，/ *不支持的元数据掩码值
                                          数据路径。* /
        OFPBIC_BAD_EXPERIMENTER = 5，/ *指定了未知的实验者ID。* /
        OFPBIC_BAD_EXP_TYPE         = 6，/ *实验者ID的未知说明。* /
        OFPBIC_BAD_LEN              = 7，/ *指令中的长度问题。* /
        OFPBIC_EPERM                = 8，/ *权限错误。* /
        OFPBIC_DUP_INST             = 9，/ *重复指令。* /
};
```

如果消息中包含的指令未知，则交换机必须返回一个带有以下内容的ofp_error_msg：

OFPET_BAD_INSTRUCTION类型和OFPBIC_UNKNOWN_INST代码。

如果消息中包含的说明不受支持，则交换机必须返回ofp_error_msg
具有OFPET_BAD_INSTRUCTION类型和OFPBIC_UNSUP_INST代码。

如果消息中的一组指令中包含的指令包含两个或多个带有
相同类型的交换机必须返回OFPET_BAD_INSTRUCTION类型的ofp_error_msg

OFPBIC_DUP_INST代码。

如果消息中包含的指令包含*Goto-Table*，并且next-table-id引用了
无效的表，交换机必须返回OFPET_BAD_INSTRUCTION类型的ofp_error_msg，并且

OFPBIC_BAD_TABLE_ID代码。

如果请求的指令包含*Write-Metadata*和元数据值或元数据掩码值
不支持此命令，则交换机必须返回OFPET_BAD_INSTRUCTION类型的ofp_error_msg，并且

OFPBIC_UNSUP_METADATA或OFPBIC_UNSUP_METADATA_MASK代码。

如果“ *清除动作*”指令包含一些动作，则开关必须返回带有以下内容的ofp_error_msg：

OFPET_BAD_INSTRUCTION类型和OFPBIC_BAD_LEN代码。

7.5.4.5错误的匹配错误代码

对于OFPET_BAD_MATCH错误类型，当前定义了以下代码：

```
/ * OFPET_BAD_MATCH的p_error_msg'代码'值。“数据”至少包含
  *失败请求的前64个字节。* /
of_p_bad_match_code {
        OFPBMC_BAD_TYPE             = 0，/ *由
                                          比赛 */
        OFPBMC_BAD_LEN              = 1，/ *匹配中的长度问题。* /
        OFPBMC_BAD_TAG              = 2，/ *匹配使用不受支持的标签/封包。* /
        OFPBMC_BAD_DL_ADDR_MASK = 3，/ *不支持的数据链路地址掩码-开关
                                          不支持任意数据链接
                                          地址掩码。* /
        OFPBMC_BAD_NW_ADDR_MASK = 4，/ *不支持的网络地址掩码-开关
                                          不支持任意网络
                                          地址掩码。* /
        OFPBMC_BAD_WILDCARDS    = 5，/ *屏蔽的字段不受支持的组合
                                          或在比赛中省略。* /
        OFPBMC_BAD_FIELD            = 6，/ *匹配中不支持的字段类型。* /
        OFPBMC_BAD_VALUE            = 7，/ *匹配字段中不支持的值。* /
        OFPBMC_BAD_MASK             = 8，/ *匹配中指定了不受支持的掩码。* /
        OFPBMC_BAD_PREREQ           = 9，/ *不满足先决条件。* /
        OFPBMC_DUP_FIELD            = 10，/ *字段类型重复。* /
        OFPBMC_EPERM                = 11，/ *权限错误。* /
};
```

如果消息中的匹配项指定了指定表中不支持的字段或类，则
开关必须返回具有OFPET_BAD_MATCH类型和OFPBMC_BAD_FIELD代码的ofp_error_msg。

如果消息中的匹配项指定了一个以上字段，则交换机必须返回ofp_error_msg
具有OFPET_BAD_MATCH类型和OFPBMC_DUP_FIELD代码。

如果消息中的匹配项指定一个字段，但未指定其关联的先决条件（请参见7.2.3.6），则
示例指定一个IPv4地址而不将EtherType匹配为0x800，则交换机必须返回
OFPET_BAD_MATCH类型和OFPBMC_BAD_PREREQ代码的ofp_error_msg。

如果消息中的匹配为数据链接或网络地址指定了任意位掩码
交换机无法支持，交换机必须返回带有OFPET_BAD_MATCH的ofp_error_msg
类型和OFPBMC_BAD_DL_ADDR_MASK或OFPBMC_BAD_NW_ADDR_MASK。如果在
*两个*数据链路和网络地址，不支持那么OFPBMC_BAD_DL_ADDR_MASK应
用过的。如果消息中的匹配为另一个字段指定了任意位掩码，则交换机无法
支持，交换机必须返回类型为OFPET_BAD_MATCH和OFPBMC_BAD_MASK的ofp_error_msg
码。

如果消息中的匹配指定了无法匹配的值，例如，VLAN ID更大
大于4095且不是保留值之一，也不是使用超过6位的DSCP值，则交换机必须
返回具有OFPET_BAD_MATCH类型和OFPBMC_BAD_VALUE代码的ofp_error_msg。

如果匹配中的OXM字段包含掩码，则如果oxm_mask中的0位为，则对应的位为

oxm_value是一个1位，交换机必须返回OFPET_BAD_MATCH类型的ofp_error_msg，并且
OFPBMC_BAD_WILDCARDS代码。

如果匹配中的两个OXM字段具有相同的类型，则交换机应返回带有以下内容的ofp_error_msg：

OFPET_BAD_MATCH类型和OFPBMC_DUP_FIELD代码。

7.5.4.6 Flow-mod失败错误代码

对于OFPET_FLOW_MOD_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_FLOW_MOD_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
of_p_flow_mod_failed_code {
        OFPFMFC_UNKNOWN      = 0，    / *未指定的错误。* /
        OFPFMFC_TABLE_FULL   = 1      / *由于表已满，因此未添加流。* /
        OFPFMFC_BAD_TABLE_ID = 2      / *表不存在* /
        OFPFMFC_OVERLAP      = 3，    / *尝试添加重叠流
                                          CHECK_OVERLAP标志设置。* /
        OFPFMFC_EPERM        = 4      / *权限错误。* /
        OFPFMFC_BAD_TIMEOUT = 5       / *由于不支持而未添加流
                                          空闲/硬超时。* /
        OFPFMFC_BAD_COMMAND = 6       / *不支持或未知的命令。* /
        OFPFMFC_BAD_FLAGS    = 7      / *不支持或未知的标志。* /
```

```
    OFPFMFC_CANT_SYNC = 8          /*未同步由于问题*/ /*
    OFPFMFC_BAD_PRIORITY = 9       /*未支持的优先级值。*/ /*
    OFPFMFC_IS_SYNC = 10，/*同步流条目为只读。*/
};
```

如果交换机在请求的表中找不到任何要在*Flow-中*添加流条目的空间，

*Mod*消息，交换机必须发送OFPET_FLOW_MOD_FAILED类型的OFP_ERROR_MSG和
OFPFMFC_TABLE_FULL代码。

如果流修改消息指定了无效的表ID，则交换机必须发送ofp_error_msg
具有OFPET_FLOW_MOD_FAILED类型和OFPFMFC_BAD_TABLE_ID代码。如果流程修改消息
在添加或修改请求中为table-id指定OFPTT_ALL ，交换机必须发送相同的错误
信息。

如果带有*添加*请求的*Flow-Mod*消息设置了OFPFF_CHECK_OVERLAP标志，并且该请求具有
重叠冲突（请参见 6.4），则交换机必须使用OFPET_FLOW_MOD_FAILED响应ofp_error_msg
类型和OFPFMFC_OVERLAP代码。

如果在源流表中添加或修改了流条目，并且是否同步了另一个流表
在该源流表上并且交换机无法同步流mod请求，交换机必须重新执行
注入流程mod并立即返回OFPET_FLOW_MOD_FAILED类型的ofp_error_msg

OFPFMFC_CANT_SYNC代码。

如果将流条目添加到设置了OFPFF_CHECK_OVERLAP标志的源流表中，并且是否存在另一个流
该表在该源流表上同步，并且相应的流条目将与流重叠
条目已经存在于同步流表中，交换机必须拒绝完整的flow-mod请求，并且
使用OFPET_FLOW_MOD_FAILED类型和OFPFMFC_CANT_SYNC代码响应ofp_error_msg。

如果控制器尝试在同步流表中添加，修改或删除流条目，则控制器会
交换机无法支持此类更改，因此该交换机必须拒绝flow mod并立即返回

OFPET_FLOW_MOD_FAILED类型和OFPFMFC_IS_SYNC代码的ofp_error_msg。

如果在处理过程中发生任何无法映射到现有错误代码的错误，
flow mod消息，交换机可能会返回OFPET_FLOW_MOD_FAILED类型的ofp_error_msg，并且

OFPFMC_UNKNOWN代码。

7.5.4.7 Group-mod失败错误代码

对于OFPET_GROUP_MOD_FAILED错误类型，当前定义了以下代码：

```
/* OFPET_GROUP_MOD_FAILED的p_error_msg'代码'值。“数据”包含
 *至少失败请求的前64个字节。*/
of_p_group_mod_failed_code的枚举{
    OFPGMFC_GROUP_EXISTS            = 0，/*未添加组，因为添加了组
                                          试图更换
                                          已经存在的组。*/
    OFPGMFC_INVALID_GROUP           = 1，/*未添加组，因为组
                                          指定的无效。*/
    OFPGMFC_WEIGHT_UNSUPPORTED      = 2，/*开关不支持不相等的负载
                                          与选定的组共享。*/
    OFPGMFC_OUT_OF_GROUPS           = 3，/*组表已满。*/
    OFPGMFC_OUT_OF_BUCKETS          = 4，/*最大操作桶数
                                          超过了一个组。*/
    OFPGMFC_CHAINING_UNSUPPORTED    = 5，/*开关不支持
                                          转发给团体。*/
    OFPGMFC_WATCH_UNSUPPORTED       = 6，/*此群组无法观看watch_port
                                          或指定的watch_group。*/
    OFPGMFC_LOOP                    = 7，/*组条目将导致循环。*/
    OFPGMFC_UNKNOWN_GROUP           = 8，/*组未修改，因为组
                                          修改尝试修改
                                          不存在的组。*/
    OFPGMFC_CHAINED_GROUP           = 9，/*组未删除，因为另一个
                                          组正在转发给它。*/
    OFPGMFC_BAD_TYPE                = 10，/*不支持或未知的组类型。*/
    OFPGMFC_BAD_COMMAND             = 11，/*不支持或未知的命令。*/
    OFPGMFC_BAD_BUCKET              = 12，/*存储桶错误。*/
    OFPGMFC_BAD_WATCH               = 13，/*监视端口/组中的错误。*/
    OFPGMFC_EPERM                   = 14，/*权限错误。*/
    OFPGMFC_UNKNOWN_BUCKET          = 15，/*在
                                          插入桶或取出桶
                                          命令。*/
    OFPGMFC_BUCKET_EXISTS           = 16，/*无法插入存储桶，因为存储桶
                                          该存储桶ID已经存在。*/
};
```

如果带有命令**add**的Group-Mod请求指定了已经存在于
组表，则交换机必须拒绝添加组条目，并且必须发送带有以下内容的ofp_error_msg

OFPET_GROUP_MOD_FAILED类型和OFPGMFC_GROUP_EXISTS代码。

如果使用命令**Modify**或**Insert Bucket**的Group-Mod请求指定的组标识符为
尚不存在，则交换机必须拒绝组mod并使用以下命令发送ofp_error_msg

OFPET_GROUP_MOD_FAILED类型和OFPGMFC_UNKNOWN_GROUP代码。

如果指定的组类型无效或交换机不支持，则交换机必须拒绝
添加组条目，并且必须发送类型为OFPET_GROUP_MOD_FAILED的ofp_error_msg，并且

OFPGMFC_BAD_TYPE代码。

如果group-mod请求为一组*Indirect*类型的组指定多个存储桶，则开关必须
拒绝添加组条目，并且必须发送类型为OFPET_GROUP_MOD_FAILED的ofp_error_msg
和OFPGMFC_INVALID_GROUP代码。

如果指定的组是无效的（即：为类型不是se-的组指定非零权重
选择），则交换机必须拒绝添加组条目，并且必须发送带有以下内容的ofp_error_msg

OFPET_GROUP_MOD_FAILED类型和OFPGMFC_INVALID_GROUP代码。

如果交换机不支持与选定组不平等的负载分担（重量不同的存储桶，
大于1），它必须拒绝添加组条目，并且必须发送带有以下内容的ofp_error_msg

OFPET_GROUP_MOD_FAILED类型和OFPGMFC_WEIGHT_UNSUPPORTED代码。

如果由于空间不足而无法添加传入的组条目，则交换机必须发送一个

ofp_error_msg，具有OFPET_GROUP_MOD_FAILED类型和OFPGMFC_OUT_OF_GROUPS代码。

如果交换机由于限制（硬件或其他方式）的限制而无法添加传入的组条目
组存储桶的数量，它必须拒绝添加组条目，并且必须发送ofp_error_msg
具有OFPET_GROUP_MOD_FAILED类型和OFPGMFC_OUT_OF_BUCKETS代码。

如果交换机由于不支持建议的活动性而无法添加传入组
配置时，交换机必须发送OFPET_GROUP_MOD_FAILED类型的ofp_error_msg，

OFPGMFC_WATCH_UNSUPPORTED代码。这包括为组指定watch_port或watch_group
在watch_port中不支持活动性的端口，或指定了不支持活动性的端口，或者
在watch_group中指定一个不支持活动的组。

如果使用命令**删除**的Group-Mod请求在*Group-Mod*消息中包含一些存储区，
鼠尾草，则交换机必须返回OFOF_GROUP_MOD_FAILED类型的ofp_error_msg，

OFPGMFC_INVALID_GROUP代码。

对于带有命令的Group-Mod请求，如果是command_bucket_id，则**插入bucket**或**remove bucket**
字段值在指定组中尚不存在，或者不是该字段的保留值之一
该命令，则交换机必须拒绝组mod并发送带有以下内容的ofp_error_msg

OFPET_GROUP_MOD_FAILED类型和OFPGMFC_UNKNOWN_BUCKET代码。

对于使用命令**remove bucket的** Group-Mod请求，如果当前操作存储区列表为空，则
交换机必须拒绝组mod并发送OFPET_GROUP_MOD_FAILED类型的ofp_error_msg
和OFPGMFC_UNKNOWN_BUCKET代码。

对于具有命令**insert bucket的** Group-Mod请求，如果有一个组请求会使该组成为组
桶ID重复，则交换机必须拒绝组mod并发送带有以下内容的ofp_error_msg

OFPET_GROUP_MOD_FAILED类型和OFPGMFC_BUCKET_EXISTS代码。

如果交换机不支持组链接或组引用组，则它必须发送ofp_error_msg
具有OFPET_GROUP_MOD_FAILED类型和OFPGMFC_CHAINING_UNSUPPORTED代码。

如果存储桶中的某个操作引用了当前未在交换机上定义或保留的组
组，例如OFPG_ALL，交换机必须返回OFPET_BAD_ACTION类型的ofp_error_msg，

OFPBAC_BAD_OUT_GROUP代码。

如果交换机支持组链接并检查循环，并且group-mod是否会创建
转发循环，交换机必须拒绝组mod，并且必须发送带有以下内容的ofp_error_msg：

OFPET_GROUP_MOD_FAILED类型和OFPGMFC_LOOP代码。

如果交换机支持组链接并检查链接组是否未删除，并且该组-
mod将删除另一个组引用的组，交换机必须拒绝删除该组条目
并且必须发送类型为OFPET_GROUP_MOD_FAILED和OFPGMFC_CHAINED_GROUP的ofp_error_msg
码。

7.5.4.8 Port-mod失败错误代码

对于OFPET_PORT_MOD_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_PORT_MOD_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
of_p_port_mod_failed_code的枚举{
        OFPPMFC_BAD_PORT          = 0,       / *指定的端口号不存在。* /
        OFPPMFC_BAD_HW_ADDR = 1          / *指定的硬件地址不正确
                                             *匹配端口号。* /
        OFPPMFC_BAD_CONFIG      = 2        / *指定的配置无效。* /
        OFPPMFC_BAD_ADVERTISE = 3,       / *指定的广告无效。* /
        OFPPMFC_EPERM                = 4        / *权限错误。* /
};
```

7.5.4.9 Table-mod失败错误代码

对于OFPET_TABLE_MOD_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_TABLE_MOD_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
of_p_table_mod_failed_code的枚举{
        OFPTMFC_BAD_TABLE = 0,             / *指定的表不存在。* /
        OFPTMFC_BAD_CONFIG = 1,            / *指定的配置无效。* /
        OFPTMFC_EPERM          = 2             / *权限错误。* /
};
```

7.5.4.10队列操作失败错误代码

对于OFPET_QUEUE_OP_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_QUEUE_OP_FAILED的p_error msg'代码'值。“数据”包含
  *至少失败请求的前64个字节* /
of_p_queue_op_failed_code {
        OFPQOFC_BAD_PORT    = 0,           / *无效的端口（或端口不存在）。* /
        OFPQOFC_BAD_QUEUE = 1              / *队列不存在。* /
        OFPQOFC_EPERM          = 2             / *权限错误。* /
};
```


7.5.4.11开关配置失败错误代码

对于OFPET_SWITCH_CONFIG_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_SWITCH_CONFIG_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
p_switch_config_failed_code的枚举{
        OFPSCFC_BAD_FLAGS      = 0,       / *指定的标志无效。* /
        OFPSCFC_BAD_LEN          = 1        / *指定的错过发送len无效。* /
        OFPSCFC_EPERM              = 2        / *权限错误。* /
};
```

7.5.4.12角色请求失败错误代码

对于OFPET_ROLE_REQUEST_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_ROLE_REQUEST_FAILED的p_error_msg'代码'值。“数据”包含
```

```
  *至少失败请求的前64个字节。* /
of_p_role_request_failed_code的枚举{
    OFPRRFC_STALE       = 0，       / *过时消息：旧generation_id。* /
    OFPRRFC_UNSUP       = 1         / *不支持控制器角色更改。* /
    OFPRRFC_BAD_ROLE    = 2         / *无效的角色。* /
    OFPRRFC_ID_UNSUP    = 3，       / *开关不支持更改ID。* /
    OFPRRFC_ID_IN_USE = 4           / *请求的ID正在使用。* /
};
```

如果交换机收到角色等于OFPCR_ROLE_MASTER的OFPT_ROLE_REQUEST或

OFPCR_ROLE_SLAVE，并且generation_id小于以前看到的generation ID，该开关
必须使用类型OFPET_ROLE_REQUEST_FAILED和代码OFPRRFC_STALE的错误消息进行回复。

7.5.4.13 Meter-mod失败错误代码

对于OFPET_METER_MOD_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_METER_MOD_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
of_p_meter_mod_failed_code {
    OFPMMFC_UNKNOWN       = 0，/ *未指定错误。* /
    OFPMMFC_METER_EXISTS = 1，/ *未添加仪表，因为仪表已添加
                                *试图更换现有的仪表。* /
    OFPMMFC_INVALID_METER = 2，/ *未添加仪表，因为指定了仪表
                                * 是无效的，
                                *或电表动作中的电表无效。* /
    OFPMMFC_UNKNOWN_METER = 3，/ *仪表未修改，因为仪表已修改
                                *尝试修改不存在的仪表，
                                *或电表动作不良。* /
    OFPMMFC_BAD_COMMAND = 4，/ *不支持或未知的命令。* /
    OFPMMFC_BAD_FLAGS     = 5，/ *不支持标志配置。* /
    OFPMMFC_BAD_RATE      = 6，/ *不支持费率。* /
    OFPMMFC_BAD_BURST     = 7，/ *不支持连拍大小。* /
```



```
    OFPMMFC_BAD_BAND      = 8，/ *频段不受支持。* /
    OFPMMFC_BAD_BAND_VALUE = 9，/ *不支持频带值。* /
    OFPMMFC_OUT_OF_METERS = 10，/ *没有更多可用的仪表。* /
    OFPMMFC_OUT_OF_BANDS = 11，/ *最大属性数
                                *已超过米。* /
};
```

如果带有命令**add**的Meter-Mod请求指定了已经存在的仪表标识符，则
交换机必须拒绝添加仪表，并且必须发送带有OFPET_METER_MOD_FAILED的ofp_error_msg
类型和OFPMMFC_METER_EXISTS代码。

如果用命令仪表-MOD要求修改指定了一个不存在的仪表标识，
则交换机必须拒绝电表mod并使用OFPET_METER_MOD_FAILED发送ofp_error_msg
类型和OFPMMFC_UNKNOWN_METER代码。

如果由于空间不足，交换机无法添加或修改传入的电表条目，则交换机必须发送
类型为OFPET_METER_MOD_FAILED和OFPMMFC_OUT_OF_METERS代码的ofp_error_msg。

如果由于限制（硬件或其他原因），交换机无法添加或修改传入的电表条目
限制频段数量，它必须拒绝添加仪表条目，并且必须发送ofp_error_msg
具有OFPET_METER_MOD_FAILED类型和OFPMMFC_OUT_OF_BANDS代码。

7.5.4.14表功能失败的错误代码

对于OFPET_TABLE_FEATURES_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_TABLE_FEATURES_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
of_p_table_features_failed_code的枚举{
    OFPTFFC_BAD_TABLE     = 0，       / *指定的表不存在。* /
    OFPTFFC_BAD_METADATA = 1          / *无效的元数据掩码。* /
    OFPTFFC_EPERM         = 5         / *权限错误。* /
    OFPTFFC_BAD_CAPA      = 6         / *无效的功能字段。* /
    OFPTFFC_BAD_MAX_ENT = 7           / *无效的max_entries字段。* /
    OFPTFFC_BAD_FEATURES = 8          / *无效的功能字段。* /
    OFPTFFC_BAD_COMMAND = 9           / *无效的命令。* /
    OFPTFFC_TOO_MANY      = 10，      / *无法处理这么多的流表。* /
};
```

如果交换机无法完全根据表格功能请求设置请求的配置，
返回类型为OFPET_TABLE_FEATURES_FAILED的错误以及相应的错误代码。

如果交换机不支持请求的表功能请求命令，则错误类型为

OFPET_TABLE_FEATURES_FAILED和代码OFPTFFC_BAD_COMMAND被返回。

对于OFPTFC_REPLACE以外的命令，如果流表的数量大于开关的数量
支持，则返回类型为OFPET_TABLE_FEATURES_FAILED的错误，并返回代码OFPTFFC_TOO_MANY。

7.5.4.15错误的属性错误代码

对于OFPET_BAD_PROPERTY错误类型，当前定义了以下代码：

```
/ * OFPET_BAD_PROPERTY的p_error_msg'代码'值。“数据”至少包含
  *失败请求的前64个字节。* /
of_p_bad_property_code的枚举{
        OFPBPC_BAD_TYPE                 = 0，/ *未知或不受支持的属性类型。* /
        OFPBPC_BAD_LEN                  = 1，/ *属性中的长度问题。* /
        OFPBPC_BAD_VALUE                = 2，/ *不支持的属性值。* /
        OFPBPC_TOO_MANY                 = 3，/ *无法处理这么多属性。* /
        OFPBPC_DUP_TYPE                 = 4，/ *属性类型重复。* /
        OFPBPC_BAD_EXPERIMENTER = 5，/ *指定了未知的实验者ID。* /
        OFPBPC_BAD_EXP_TYPE             = 6，/ *实验者ID的未知exp_type。* /
        OFPBPC_BAD_EXP_VALUE            = 7，/ *实验者ID的未知值。* /
        OFPBPC_EPERM                    = 8，/ *权限错误。* /
};
```

7.5.4.16异步配置失败错误代码

对于OFPET_ASYNC_CONFIG_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_ASYNC_CONFIG_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
p_async_config_failed_code的枚举{
        OFPACFC_INVALID         = 0，       / *一个掩码无效。* /
        OFPACFC_UNSUPPORTED = 1             / *不支持请求的配置。* /
        OFPACFC_EPERM           = 2         / *权限错误。* /
};
```

7.5.4.17流监控器失败错误代码

对于OFPET_FLOW_MONITOR_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_FLOW_MONITOR_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
of_p_flow_monitor_failed_code的枚举{
        OFPMOFC_UNKNOWN         = 0，/ *未指定错误。* /
        OFPMOFC_MONITOR_EXISTS = 1，/ *未添加监视器，因为添加了监视器
                                        *尝试更换现有的监视器。* /
        OFPMOFC_INVALID_MONITOR = 2，/ *未添加监视器，因为指定了监视器
                                        * 是无效的。* /
        OFPMOFC_UNKNOWN_MONITOR = 3，/ *监视器未修改，因为监视器
                                          修改试图修改不存在的内容
                                          监控。* /
        OFPMOFC_BAD_COMMAND= 4，/ *不支持或未知的命令。* /
        OFPMOFC_BAD_FLAGS       = 5，/ *不支持标志配置。* /
        OFPMOFC_BAD_TABLE_ID = 6，/ *指定的表不存在。* /
        OFPMOFC_BAD_OUT         = 7，/ *输出端口/组中的错误。* /
};
```


7.5.4.18捆绑失败的错误代码

对于OFPET_BUNDLE_FAILED错误类型，当前定义了以下代码：

```
/ * OFPET_BUNDLE_FAILED的p_error_msg'代码'值。“数据”包含
  *至少失败请求的前64个字节。* /
of_pundle_failed_code {
    OFPBFC_UNKNOWN          = 0，/ *未指定错误。* /
    OFPBFC_EPERM            = 1，/ *权限错误。* /
    OFPBFC_BAD_ID           = 2，/ *捆绑包ID不存在。* /
    OFPBFC_BUNDLE_EXIST     = 3，/ *捆绑包ID已存在。* /
    OFPBFC_BUNDLE_CLOSED = 4，/ *捆绑包ID已关闭。* /
    OFPBFC_OUT_OF_BUNDLES = 5，/ *束ID过多。* /
    OFPBFC_BAD_TYPE         = 6，/ *不支持或未知的消息控件类型。* /
    OFPBFC_BAD_FLAGS        = 7，/ *不支持，未知或不一致的标志。* /
    OFPBFC_MSG_BAD_LEN      = 8，/ *包含的消息中的长度问题。* /
    OFPBFC_MSG_BAD_XID      = 9，/ * XID不一致或重复。* /
    OFPBFC_MSG_UNSUP        = 10，/ *此捆绑包中不支持的消息。* /
    OFPBFC_MSG_CONFLICT     = 11，/ *在此不支持的消息组合
                                束。* /
    OFPBFC_MSG_TOO_MANY = 12，/ *无法捆绑处理这么多邮件。* /
    OFPBFC_MSG_FAILED       = 13，/ *捆绑中的一条消息失败。* /
    OFPBFC_TIMEOUT          = 14，/ *捆绑包花费的时间太长。* /
    OFPBFC_BUNDLE_IN_PROGRESS = 15，/ *捆绑包正在锁定资源。* /
    OFPBFC_SCHED_NOT_SUPPORTED = 16，/ *已收到计划的提交，并且
                                        不支持调度。* /
    OFPBFC_SCHED_FUTURE = 17        / *计划的提交时间超出上限。* /
    OFPBFC_SCHED_PAST = 18，        / *计划的提交时间超出下限。* /
};
```

如果“ *捆绑包打开请求”中*的bundle_id 引用了连接到同一连接的现有捆绑包，
交换机必须拒绝打开新的捆绑包，并发送带有OFPET_BUNDLE_FAILED的ofp_error_msg
类型和OFPBFC_BAD_ID代码。必须丢弃bundle_id标识的现有捆绑包。

如果交换机无法打开“捆绑*包打开请求”中*指定的*捆绑包*，因为它包含太多
交换机上已打开的捆绑包或已连接到当前连接的交换机，交换机必须拒绝打开
新的捆绑包并发送具有OFPET_BUNDLE_FAILED类型和OFPBFC_OUT_OF_BUNDLES类型的ofp_error_msg
码。

如果由于连接正在使用而无法打开*捆绑打开请求中*指定的*捆绑*，
如果传输不可靠，则交换机必须拒绝打开新捆绑包并发送ofp_error_msg
具有OFPET_BUNDLE_FAILED类型和OFPBFC_OUT_OF_BUNDLES代码。

如果在*捆绑请求中*指定的标志字段指定了一些无法实现的功能
通过交换机，交换机必须拒绝打开新捆绑包，并发送带有以下内容的ofp_error_msg：

OFPET_BUNDLE_FAILED类型和OFPBFC_BAD_FLAGS代码。

如果*捆绑添加请求中*的bundle_id 引用已关闭的捆绑，则该开关
必须拒绝将消息添加到分发包、丢弃分发包并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_BUNDLE_CLOSED代码。

如果*捆绑包添加请求*，*捆绑包关闭请求*或*捆绑包提交请求中*的标志字段不同
从打开捆绑包时指定的标志中，交换机必须拒绝添加消息

到捆绑包，丢弃捆绑包并发送OFPET_BUNDLE_FAILED类型的ofp_error_msg，

OFPBFC_BAD_FLAGS代码。

如果交换机通常支持*捆绑添加请求中*的消息，但不支持
捆绑软件，交换机必须拒绝将消息添加到捆绑软件，并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_UNSUP代码。hello，echo和bundle就是这种情况
消息，非请求消息（例如答复，状态和错误消息），或者
实现不支持在捆绑包中包含特定的修改消息。

如果*捆绑包添加请求*中的消息与已经存储在
捆绑软件，交换机必须拒绝将消息添加到捆绑软件，并发送带有以下内容的ofp_error_msg

OFPET_BUNDLE_FAILED类型和OFPBFC_MSG_CONFLICT代码。

如果捆绑包已满并且无法容纳*捆绑包添加请求中*的消息，则交换机必须拒绝
将消息添加到分发包中并发送OFPET_BUNDLE_FAILED类型的ofp_error_msg和

OFPBFC_MSG_TOO_MANY代码。

如果*捆绑添加请求*中的消息没有有效的长度字段，则交换机必须拒绝
将消息添加到分发包中并发送OFPET_BUNDLE_FAILED类型的ofp_error_msg和
OFPBFC_MSG_BAD_LEN代码。

如果*捆绑包关闭请求*或*捆绑包提交请求中*的bundle_id 引用了
不存在，交换机必须拒绝该请求，并发送带有OFPET_BUNDLE_FAILED的ofp_error_msg
类型和OFPBFC_BAD_ID代码。

如果*捆绑包关闭请求中*的bundle_id 引用已关闭的捆绑包，则
交换机必须拒绝靠近包，丢弃包并发送带有以下内容的ofp_error_msg
OFPET_BUNDLE_FAILED类型和OFPBFC_BUNDLE_CLOSED代码。

7.5.4.19实验者错误消息

对于OFPET_EXPERIMENTER错误类型，错误消息由以下结构定义，并且
领域：

```
/ * OFPET_EXPERIMENTER：错误消息（数据路径->控制器）。* /
struct ofp_error_experimenter_msg {
        struct ofp_header标头；

        uint16_t类型；                  / * OFPET_EXPERIMENTER。* /
        uint16_t exp_code；             / *实验者定义。* /
        uint32_t实验者；                / *实验者ID。* /
        uint8_t data [0]；              / *可变长度数据。口译依据
                                            关于类型和实验者。没有填充。* /
};
OFP_ASSERT（sizeof（p_error_experimenter_msg的结构）== 16）；
```

实验者字段为实验者ID，其格式与典型实验中的形式相同-
imenter结构（参见7.2。 8）。exp_code字段是实验者代码，由16位字段管理
实验者，它可以具有实验者希望的任何价值和功能。数据字段是
实验者已定义，无需填充。

7.5.5实验者留言

实验者消息定义如下：

```
/ *实验者扩展消息。* /
struct ofp_experimenter_msg {
        struct ofp_header标头；         / *输入OFPT_EXPERIMENTER。* /
        uint32_t实验者；                / *实验者ID：
                                            *-MSB 0：低位字节是IEEE OUI。
                                            *-MSB！= 0：由ONF定义。* /
        uint32_t exp_type；             / *实验者定义。* /
        / *实验者定义的任意附加数据。* /
        uint8_t Experimenter_data [0]；
};
OFP_ASSERT（sizeof（p_experimenter_msg的结构）== 16）；
```

实验者字段标识实验者，exp_type字段是实验者类型，并且
实验者数据是实验者任意数据。它们采用与典型形式相同的形式
实验者结构（请参阅7.2。 8）。实验者消息无需填充。

与其他Experimenter扩展相反，Experimenter消息与特定的
OpenFlow对象，因此可用于创建全新的API和管理全新的
对象。

如果交换机不理解实验者消息，则必须发送OFPT_ERROR消息，其中包含
OFPBRC_BAD_EXPERIMENTER错误代码和OFPET_BAD_REQUEST错误类型。

附录A头文件openflow.h

文件openflow.h包含OpenFlow使用的所有结构，定义和枚举
协议。为规范1.5.1的当前版本定义的openflow.h版本是

包含在下面，并在此附加到此文档中（并非在所有PDF阅读器中都可用）。

```
/ *版权所有（c）2008 Leland Stanford董事会
 *初级大学
```

```
*版权所有（c）2011、2013、2014开放网络基金会
*我们正在制作OpenFlow规范和相关文档
*（软件），可供公众使用并受益于期望
*其他人将使用，修改和增强该软件并做出贡献
*将这些增强功能返回给社区。但是，因为我们会
*希望使该软件可以最广泛地使用
*特此授予可能的限制，不得
*向获得本软件副本进行交易的任何人收取费用
*本软件不受版权限制，包括
*但不限于使用，复制，修改，合并，发布，
*分发，再许可和/或出售本软件的副本，并
*允许提供软件的人员这样做，但须遵守
*以下条件：
*
*以上版权声明和本许可声明应为
*包含在软件的所有副本或重要部分中。
*
*该软件按“原样”提供，没有任何形式的保证，
*明确或暗示，包括但不限于以下保证
*适销性，特定目的的适用性和
*非侵权。在任何情况下，作者或版权持有人都不得
*对于任何索赔，损害或其他责任，无论是在
*来自，外或内的合同，侵权或其他行为
*与本软件的连接或使用或其他处理
```



```
*软件。
*
*版权持有人的名称和商标不得用于
*与软件或任何相关的广告或宣传
*未经明确事先书面许可的衍生产品。
* /

/ * OpenFlow：控制器和数据路径之间的协议。* /

#ifndef OPENFLOW_OPENFLOW_H
＃定义OPENFLOW_OPENFLOW_H 1

#ifdef __KERNEL__
#include <linux / types.h>
＃其他
#include <stdint.h>
＃万一

#ifdef SWIG
＃定义OFP_ASSERT（EXPR）/ * SWIG无法处理OFP_ASSERT。* /
#elif ！defined（__ cplusplus）
/ *在声明上下文中使用的构建时声明。* /
＃定义OFP_ASSERT（EXPR）                                    \
        extern int（* build_assert（void））[sizeof（struct {          \
                    unsigned int build_assert_failed：（EXPR）吗？1：-1；}]]
#else / * __cplusplus * /
#define OFP_ASSERT（_EXPR）typedef int build_assert_failed [（_ EXPR）吗？1：-1]
#endif / * __cplusplus * /

#ifndef SWIG
＃定义OFP_PACKED__属性__（（包装））
＃其他
#define OFP_PACKED          / * SWIG不理解__attribute。* /
＃万一

/ *版本号：
*发布的OpenFlow版本：0x01 = 1.0; 0x02 = 1.1; 0x03 = 1.2
*0x04 = 1.3.X; 0x05 = 1.4.X; 0x06 = 1.5.X
* /
/ *版本字段中的最高有效位已保留，必须
*设置为零。
* /
#define OFP_VERSION 0x06

#define OFP_MAX_TABLE_NAME_LEN 32
#define OFP_MAX_PORT_NAME_LEN 16

/ * OpenFlow的IANA官方注册端口。* /
＃定义OFP_TCP_PORT 6653
#define OFP_SSL_PORT 6653

#define OFP_ETH_ALEN 6    / *以太网地址中的字节。* /

/ *端口编号。端口从1开始编号。* /
枚举ofp_port_no {
    / *物理和逻辑交换机端口的最大数量。* /
    OFPP_MAX    = 0xffffff00，

    / *保留的OpenFlow端口（伪输出“端口”）。* /
    OFPP_UNSET = 0xfffffff7，  / *未在操作集中设置输出端口。
                                  仅在OXM OF_ACTSET_OUTPUT中使用。* /
    OFPP_IN_PORT = 0xfffffff8，  / *将数据包发送出输入端口。这个
                                  必须明确使用保留端口
                                  为了发回输入
                                  港口。* /
    OFPP_TABLE = 0xfffffff9，  / *将数据包提交到第一个流表
                                  注意：此目标端口只能是
                                  在打包消息中使用。* /
    OFPP_NORMAL = 0xfffffffa，  / *使用非OpenFlow管道转发。* /
    OFPP_FLOOD = 0xfffffffb，  / *使用非OpenFlow管道进行泛洪。* /
    OFPP_ALL    = 0xfffffffc，  / *除输入端口外的所有标准端口。* /
    OFPP_CONTROLLER = 0xfffffffd，  / *发送到控制器。* /
    OFPP_LOCAL = 0xfffffffe，  / *本地开放流“端口”。* /
    OFPP_ANY    = 0xffffffff    / *在以下情况下在某些请求中使用的特殊值
                                  没有指定端口（即通配符）。* /
};

of_p的枚举{
    / *不可变的消息。* /
    OFPT_HELLO        = 0，  / *对称消息* /
    OFPT_ERROR        = 1，  / *对称消息* /
    OFPT_ECHO_REQUEST = 2，  / *对称消息* /
    OFPT_ECHO_REPLY = 3，  / *对称消息* /
    OFPT_EXPERIMENTER = 4，  / *对称消息* /

    / *切换配置消息。* /
    OFPT_FEATURES_REQUEST = 5，  / *控制器/开关消息* /
    OFPT_FEATURES_REPLY = 6，  / *控制器/开关消息* /
    OFPT_GET_CONFIG_REQUEST = 7，  / *控制器/开关消息* /
    OFPT_GET_CONFIG_REPLY = 8，  / *控制器/开关消息* /
```

```
    OFPT_SET_CONFIG  = 9    / *控制器/开关消息* /

    / *异步消息。* /
    OFPT_PACKET_IN     = 10，/ *异步消息* /
    OFPT_FLOW_REMOVED  = 11，/ *异步消息* /
    OFPT_PORT_STATUS= 12，/ *异步消息* /

    / *控制器命令消息。* /
    OFPT_PACKET_OUT = 13，/ *控制器/开关消息* /
    OFPT_FLOW_MOD   = 14，/ *控制器/开关消息* /
    OFPT_GROUP_MOD = 15，/ *控制器/开关消息* /
    OFPT_PORT_MOD    = 16，/ *控制器/开关消息* /
    OFPT_TABLE_MOD  = 17，/ *控制器/开关消息* /

    / *多部分消息。* /
    OFPT_MULTIPART_REQUEST = 18，/ *控制器/开关消息* /
    OFPT_MULTIPART_REPLY = 19，/ *控制器/开关消息* /

    / *障碍消息。* /
    OFPT_BARRIER_REQUEST = 20，/ *控制器/开关消息* /
    OFPT_BARRIER_REPLY = 21，/ *控制器/开关消息* /

    / *控制器角色更改请求消息。* /
    OFPT_ROLE_REQUEST = 24，/ *控制器/开关消息* /
    OFPT_ROLE_REPLY = 25，/ *控制器/开关消息* /

    / *异步消息配置。* /
    OFPT_GET_ASYNC_REQUEST = 26，/ *控制器/开关消息* /
    OFPT_GET_ASYNC_REPLY = 27，/ *控制器/开关消息* /
    OFPT_SET_ASYNC   = 28，/ *控制器/开关消息* /

    / *仪表和速率限制器配置消息。* /
    OFPT_METER_MOD = 29，/ *控制器/开关消息* /

    / *控制器角色更改事件消息。* /
    OFPT_ROLE_STATUS= 30，/ *异步消息* /

    / *异步消息。* /
    OFPT_TABLE_STATUS = 31，/ *异步消息* /

    / *通过交换机请求转发。* /
    OFPT_REQUESTFORWARD = 32，/ *异步消息* /

    / *捆绑操作（多个消息作为一个操作）。* /
    OFPT_BUNDLE_CONTROL = 33，/ *控制器/开关消息* /
    OFPT_BUNDLE_ADD_MESSAGE = 34，/ *控制器/开关消息* /

    / *控制器状态异步消息。* /
    OFPT_CONTROLLER_STATUS = 35，/ *异步消息* /

};

/ *所有OpenFlow数据包的标头。* /
struct ofp_header {
    uint8_t版本；          / * OFP_VERSION。* /
    uint8_t类型；          / * OFPT_常量之一。* /
    uint16_t长度；         / *长度，包括此ofp_header。* /
    uint32_t xid；         / *与该数据包关联的交易ID。
                             回复使用与请求中相同的ID
                             方便配对。* /
};
OFP_ASSERT（sizeof（struct ofp_header）== 8）；

/ * Hello元素类型。
 * /
of_p_hello_elem_type的枚举{
    OFPHET_VERSIONBITMAP = 1   / *支持版本的位图。* /
};

/ *所有Hello元素的通用标头* /
struct ofp_hello_elem_header {
    uint16_t          类型；    / * OFPHET_ *之一。* /
    uint16_t          长度；    / *元素的长度（以字节为单位），
                                  包括此标头，不包括填充。* /
};
OFP_ASSERT（sizeof（struct ofp_hello_elem_header）== 4）；

/ *版本位图Hello元素* /
struct ofp_hello_elem_versionbitmap {
    uint16_t          类型；    / * OFPHET_VERSIONBITMAP。* /
    uint16_t          长度；    / *此元素的长度（以字节为单位），
                                  包括此标头，不包括填充。* /
    /* 其次是：
     *    -恰好（长度-4）个字节包含位图，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *    全零字节的字节* /
    uint32_t          位图[0];      / *位图列表-支持的版本* /
};
OFP_ASSERT（sizeof（p_hello_elem_versionbitmap的结构== 4）；
```



```
/ * OFPT_HELLO消息包含零个或多个具有的hello元素
 *可变大小，未知元素类型必须被忽略/跳过，以允许
 *用于将来的扩展。* /
struct ofp_hello {
    struct ofp_header标头；

    / *你好元素列表* /
    p_hello_elem_header元素的结构[0]; / *元素列表-0或更多* /
};
OFP_ASSERT（sizeof（struct ofp_hello）== 8）；

#define OFP_DEFAULT_MISS_SEND_LEN 128

of_p_config_flags枚举{
    / *处理IP片段。* /
    OFPC_FRAG_NORMAL    = 0，/ *对片段没有特殊处理。* /
    OFPC_FRAG_DROP      = 1 << 0，/ *删除片段。* /
    OFPC_FRAG_REASM     = 1 << 1，/ *重新组装（仅在设置了OFPC_IP_REASM的情况下）。* /
    OFPC_FRAG_MASK      = 3，      / *处理碎片的标志的位掩码。*7
};

/ *开关配置。* /
struct ofp_switch_config {
    struct ofp_header标头；
    uint16_t标志；                / * OFPC_ *标志的位图。* /
```

```
    uint16_t miss_send_len;    / *该交换机在发送数据包时应发送的字节数
                                 ofp_controller_max_len获取有效值。
                                 * /
};
OFP_ASSERT（sizeof（p_switch_config的结构）== 12）；

/ *表编号。表格最多可以使用OFPT_MAX个数字。* /
ofp_table枚举{
    / *最后可用的表号。* /
    OFPTT_MAX   = 0xfe，

    / *假表。* /
    OFPTT_ALL    = 0xff    / *用于表配置的通配符表
                             流量统计信息和流量删除。* /
};

/ *标志来配置表。* /
of_p_table_config的枚举{
    OFPTC_DEPRECATED_MASK   = 3，  / *不建议使用的位* /
    OFPTC_EVICTION      = 1 << 2，/ *授权表逐出流程。* /
    OFPTC_VACANCY_EVENTS = 1 << 3，/ *启用空缺事件。* /
};

/ *表Mod属性类型。
 * /
of_p_table_mod_prop_type的枚举{
    OFPTMPT_EVICTION        = 0x2，   / *逐出属性。* /
    OFPTMPT_VACANCY         = 0x3，   / *空置属性。* /
    OFPTMPT_EXPERIMENTER = 0xFFFF，/ *实验者属性。* /
};

/ *所有表Mod属性的通用标头* /
struct ofp_table_mod_prop_header {
    uint16_t            类型；    / * OFPTMPT_ *之一。* /
    uint16_t            长度；    / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_table_mod_prop_header的结构）== 4）；

/ *逐出标志。* /
of_p_table_mod_prop_eviction_flag的枚举{
    OFPTMPEF_OTHER    = 1 << 0，    / *使用其他因素。* /
    OFPTMPEF_IMPORTANCE = 1 << 1，    / *使用流条目重要性。* /
    OFPTMPEF_LIFETIME = 1 << 2，    / *使用流条目生存期。* /
};

/ *逐出表mod属性。通常在OFPMP_TABLE_DESC答复中使用。* /
struct ofp_table_mod_prop_eviction {
    uint16_t            类型；    / * OFPTMPT_EVICTION。* /
    uint16_t            长度；    / *此属性的长度（以字节为单位）。* /
    uint32_t            标志；    / * OFPTMPEF_ *标志的位图* /
};
OFP_ASSERT（sizeof（p_table_mod_prop_eviction的结构）== 8）；

/ *空位表mod属性* /
struct ofp_table_mod_prop_vacancy {
    uint16_t            类型；  / * OFPTMPT_VACANCY。* /
    uint16_t            长度；/ *此属性的长度（以字节为单位）。* /
    uint8_t vacancy_down；/ *空间减少时的空缺阈值（％）。* /
    uint8_t vacancy_up；    / *空间增加时的空缺阈值（％）。* /
    uint8_t空缺；          / *当前空缺率（％）-仅在ofp_table_desc中。* /
    uint8_t pad [1]；       / *对齐64位。* /
};
OFP_ASSERT（sizeof（struct ofp_table_mod_prop_vacancy）== 8）；
```



```
/ *实验者表mod属性* /
struct ofp_table_mod_prop_experimenter {
    uint16_t            类型；    / * OFPTMPT_EXPERIMENTER。* /
    uint16_t            长度；    / *此属性的长度（以字节为单位）。* /
    uint32_t            实验者          / *采用相同的实验者ID
                                          形式如结构
                                          ofp_experimenter_header。* /
    uint32_t            exp_type；      / *实验者定义。* /
    /* 其次是：
     *     -确切地（长度-12个）字节包含实验者数据，然后
     *     -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t            实验者数据[0]；
};
OFP_ASSERT（sizeof（p_table_mod_prop_experimenter的结构）== 12）；

/ *配置/修改流表的行为* /
struct ofp_table_mod {
    struct ofp_header标头；
    uint8_t table_id；        / *表的ID，OFPTT_ALL表示所有表* /
    uint8_t pad [3]；         / *填充到32位* /
    uint32_t配置；            / * OFPTC_ *标志的位图* /

    / *表Mod属性列表* /
    p_table_mod_prop_header属性的结构[0]；
};
OFP_ASSERT（sizeof（struct ofp_table_mod）== 16）；

/ *数据路径支持的功能。* /
ofp_capabilities枚举{
    OFPC_FLOW_STATS    = 1 << 0，/ *流量统计。* /
    OFPC_TABLE_STATS   = 1 << 1，/ *表统计信息。* /
    OFPC_PORT_STATS    = 1 << 2，/ *端口统计信息。* /
    OFPC_GROUP_STATS   = 1 << 3，/ *组统计信息。* /
    OFPC_IP_REASM      = 1 << 5，/ *可以重组IP片段。* /
    OFPC_QUEUE_STATS   = 1 << 6，/ *队列统计信息。* /
    OFPC_PORT_BLOCKED  = 1 << 8，/ *开关将阻止循环端口。* /
    OFPC_BUNDLES       = 1 << 9，/ *开关支持捆绑软件。* /
    OFPC_FLOW_MONITORING = 1 << 10 / *开关支持流量监视。* /
};

/ *指示物理端口行为的标志。                    这些标志是
 *在ofp_port中用于描述当前配置。                他们是
 *在ofp_port_mod消息中使用，用于配置端口的行为。
 * /
of_p_port_config的枚举{
    OFPPC_PORT_DOWN    = 1 << 0，/ *端口在管理上已关闭。* /

    OFPPC_NO_RECV      = 1 << 2，/ *丢弃端口接收的所有数据包。* /
    OFPPC_NO_FWD       = 1 << 5，/ *丢弃转发到端口的数据包。* /
    OFPPC_NO_PACKET_IN = 1 << 6 / *不要为端口发送输入包的消息。* /
};

/ *物理端口的当前状态。                这些不能从以下位置配置
 *控制器。
 * /
of_p_port_state的枚举{
    OFPPS_LINK_DOWN = 1 << 0，/ *不存在物理链接。* /
    OFPPS_BLOCKED   = 1 << 1，/ *端口被阻止* /
    OFPPS_LIVE      = 1 << 2，/ *为快速故障转移组而活。* /
};
```

```
of数据路径ofp_port_features枚举{
    OFPPF_10MB_HD   = 1 << 0，/ * 10 Mb半双工速率支持。* /
    OFPPF_10MB_FD   = 1 << 1，/ * 10 Mb全双工速率支持。* /
    OFPPF_100MB_HD  = 1 << 2，/ * 100 Mb半双工速率支持。* /
    OFPPF_100MB_FD  = 1 << 3，/ * 100 Mb全双工速率支持。* /
    OFPPF_1GB_HD    = 1 << 4，/ * 1 Gb半双工速率支持。* /
    OFPPF_1GB_FD    = 1 << 5，/ * 1 Gb全双工速率支持。* /
    OFPPF_10GB_FD   = 1 << 6，/ * 10 Gb全双工速率支持。* /
    OFPPF_40GB_FD   = 1 << 7，/ * 40 Gb全双工速率支持。* /
    OFPPF_100GB_FD  = 1 << 8，/ * 100 Gb全双工速率支持。* /
    OFPPF_1TB_FD    = 1 << 9，/ * 1 Tb全双工速率支持。* /
    OFPPF_OTHER     = 1 << 10，/ *其他速率，不在列表中。* /

    OFPPF_COPPER    = 1 << 11，/ *铜介质。* /
    OFPPF_FIBER     = 1 << 12，/ *纤维介质。* /
    OFPPF_AUTONEG   = 1 << 13，/ *自动协商。* /
    OFPPF_PAUSE     = 1 << 14，/ *暂停。* /
    OFPPF_PAUSE_ASYM = 1 << 15 / *非对称暂停。* /
};

/ *端口描述属性类型。
 * /
of_p_port_desc_prop_type的枚举{
    OFPPDPT_ETHERNET    = 0，      / *以太网属性。* /
    OFPPDPT_OPTICAL     = 1，      / *光学性质。* /
    OFPPDPT_PIPELINE_INPUT  = 2，  / *入口管道字段。* /
    OFPPDPT_PIPELINE_OUTPUT = 3，  / *出口管道字段。* /
```



```
    OFPPDPT_RECIRCULATE = 4，      / *再循环属性。* /
    OFPPDPT_EXPERIMENTER = 0xFFFF，/ *实验者属性。* /
};

/ *所有端口描述属性的公共头。* /
struct ofp_port_desc_prop_header {
    uint16_t 类型;    / * OFPPDPT_ *之一。* /
    uint16_t 长度;    / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_header的结构== 4）；

/ *以太网端口描述属性。* /
p_port_desc_prop_ethernet结构{
    uint16_t 类型;    / * OFPPDPT_ETHERNET。* /
    uint16_t 长度;    / *此属性的长度（以字节为单位）。* /
    uint8_t 垫[4];    / *对齐64位。* /
    / * OFPPF_ *位图，用于描述功能。      如果所有位清零
     *不支持或不可用。* /
    uint32_t curr;          / *当前功能。* /
    uint32_t广告;           / *端口正在通告的功能。* /
    uint32_t支持;           / *端口支持的功能。* /
    uint32_t对等体;         / *由对等方通告的功能。* /

    uint32_t curr_speed;    / *当前端口比特率，以kbps为单位。* /
    uint32_t max_speed;     / *最大端口比特率（kbps）* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_ethernet的结构== 32）；

/ *交换机中可用的光学端口功能。* /
of_p_optical_port_features的枚举{
    OFPOPF_RX_TUNE  = 1 << 0，/ *接收器可调* /
    OFPOPF_TX_TUNE  = 1 << 1，/ *传输可调* /
    OFPOPF_TX_PWR   = 1 << 2，/ *电源是可配置的* /
    OFPOPF_USE_FREQ = 1 << 3，/ *使用频率，而不是波长* /
};

/ *光端口描述属性。* /
struct ofp_port_desc_prop_optical {
    uint16_t 类型;    / * OFPPDPT_3OPTICAL。* /
    uint16_t 长度;    / *此属性的长度（以字节为单位）。* /
    uint8_t 垫[4];    / *对齐64位。* /

    uint32_t支持;           / *端口支持的功能。* /
    uint32_t tx_min_freq_lmda; / *最小发射频率/波长* /
    uint32_t tx_max_freq_lmda; / *最大发射频率/波长* /
    uint32_t tx_grid_freq_lmda; / * TX网格间距频率/波长* /
    uint32_t rx_min_freq_lmda; / *最小接收频率/波长* /
    uint32_t rx_max_freq_lmda; / *最大接收频率/波长* /
    uint32_t rx_grid_freq_lmda; / * RX网格间距频率/波长* /
    uint16_t tx_pwr_min;        / *最小发射功率* /
    uint16_t tx_pwr_max;        / *最大发射功率* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_optical的结构）== 40）；

/ *入口或出口管道字段。* /
struct ofp_port_desc_prop_oxm {
    uint16_t 类型;    / * OFPPDPT_PIPELINE_INPUT之一或
                        OFPPDPT_PIPELINE_OUTPUT。* /
    uint16_t 长度;    / *此属性的长度（以字节为单位）。* /
    /* 其次是：
     *   -恰好（长度-4）个字节包含oxm_id，然后
     *   -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *     全零字节的字节* /
    uint32_t oxm_ids [0]; / * OXM标头数组* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_oxm的结构）== 4）；

/ *再循环端口描述属性。* /
p_port_desc_prop_recirculate结构{
    uint16_t 类型;          / * OFPPDPT_RECIRCULATE。* /
    uint16_t 长度;          / *属性的字节长度，
                              包括此标头，不包括填充。* /
    /* 其次是：
     *   -确切地（长度-4）个字节包含端口号，然后
     *   -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *     全零字节的字节* /
    uint32_t port_nos [0];  / *再循环的输入端口号列表。
                              0或更大。端口号数
                              从中的长度字段推断
                              标头。* /
};
OFP_ASSERT（sizeof（p_port_desc_prop_recirculate的结构== 4）；

/ *实验者端口描述属性。* /
struct ofp_port_desc_prop_experimenter {
    uint16_t 类型;    / * OFPPDPT_EXPERIMENTER。* /
    uint16_t 长度;    / *此属性的长度（以字节为单位）。* /
    uint32_t 实验者         / *采用相同的实验者ID
                              形式如结构
                              ofp_experimenter_header。* /
```

第206页


```
    uint32_t            exp_type;       /*实验者定义。*/
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节*/
    uint32_t            实验者数据[0];
};
OFP_ASSERT（sizeof（p_port_desc_prop_experimenter的结构）== 12）；

/*端口说明*/
struct ofp_port {
    uint32_t port_no;
    uint16_t长度；
    uint8_t pad [2];
    uint8_t hw_addr [OFP_ETH_ALEN];
    uint8_t pad2 [2];                   /*对齐64位。*/
    字符名称[OFP_MAX_PORT_NAME_LEN]；/*空终止*/

    uint32_t配置；              /* OFPPC_*标志的位图。*/
    uint32_t状态；              /* OFPPS_*标志的位图。*/

    /*端口描述属性列表-0个或多个属性*/
    p_port_desc_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_port）== 40）；

/*切换功能。*/
struct ofp_switch_features {
    struct ofp_header标头；
    uint64_t datapath_id;   /*数据路径唯一ID。 低48位用于
                               MAC地址，而高16位是
                               实施者定义的。*/

    uint32_t n_buffers;     /*一次缓冲的最大数据包数。*/

    uint8_t n_tables;       /*数据路径支持的表数。*/
    uint8_t assistant_id;   /*标识辅助连接*/
    uint8_t pad [2];        /*对齐64位。*/

    /* 特征。*/
    uint32_t功能；          /*支持“ ofp_capabilities”的位图。*/
    uint32_t保留；
};
OFP_ASSERT（sizeof（p_switch_features的结构）== 32）；

/*关于物理端口的更改*/
of_p_port_reason枚举{
    OFPPR_ADD = 0,          /*端口已添加。*/
    OFPPR_DELETE,           /*端口已删除。*/
    OFPPR_MODIFY            /*端口的某些属性已更改。*/
};

/*数据路径中的物理端口已更改*/
struct ofp_port_status {
    struct ofp_header标头；
    uint8_t原因；               /* OFPPR_*之一。*/
    uint8_t pad [7];            /*对齐64位。*/
    struct ofp_port desc;
};
OFP_ASSERT（sizeof（p_port_status的结构）== 56）；

/*端口mod属性类型。
 */
of_p_port_mod_prop_type的枚举{
    OFPPMPT_ETHERNET = 0,       /*以太网属性。*/
    OFPPMPT_OPTICAL     = 1,    /*光学性质。*/
    OFPPMPT_EXPERIMENTER = 0xFFFF，/*实验者属性。*/
};

/*所有端口mod属性的通用头。*/
struct ofp_port_mod_prop_header {
    uint16_t            类型；  /* OFPPMPT_*之一。*/
    uint16_t            长度；  /*此属性的长度（以字节为单位）。*/
};
OFP_ASSERT（sizeof（p_port_mod_prop_header的结构）== 4）；

/*以太网端口mod属性。*/
struct ofp_port_mod_prop_ethernet {
    uint16_t        类型；      /* OFPPMPT_ETHERNET。*/
    uint16_t        长度；      /*此属性的长度（以字节为单位）。*/
    uint32_t        广告；      /* OFPPF_*的位图。将所有位请零以防止
                                   采取任何行动。*/
};
OFP_ASSERT（sizeof（p_port_mod_prop_ethernet的结构）== 8）；

struct ofp_port_mod_prop_optical {
    uint16_t        类型；      /* OFPPMPT_OPTICAL。*/
    uint16_t        长度；      /*此属性的长度（以字节为单位）。*/
    uint32_t        配置；      /* OFPOPF_*的位图。*/
    uint32_t        freq_lmda; /*“中心”频率*/
```


第207页


```
    int32_t         fl_offset;  /*带符号的频偏*/
    uint32_t        grid_span;  /*此端口的网格大小*/
    uint32_t        tx_pwr;     /*发射功率设置*/
};
OFP_ASSERT（sizeof（p_port_mod_prop_optical的结构）== 24）；

/*实验者端口mod属性。*/
struct ofp_port_mod_prop_experimenter {
    uint16_t            类型；  /* OFPPMPT_EXPERIMENTER。*/
    uint16_t            长度；  /*此属性的长度（以字节为单位）。*/
    uint32_t            实验者          /*采用相同的实验者ID
                                          形式如结构
                                          ofp_experimenter_header。*/
    uint32_t            exp_type;       /*实验者定义。*/
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节*/
    uint32_t            实验者数据[0];
};
OFP_ASSERT（sizeof（p_port_mod_prop_experimenter的结构）== 12）；
```

```
struct ofp_port_mod {
    /* 修改物理端口的行为* /
    struct ofp_header标头;
    uint32_t port_no;
    uint8_t pad [4];
    uint8_t hw_addr [OFP_ETH_ALEN]; / *硬件地址不是
                                       可配置的。 这用来
                                       检查请求，因此必须
                                       与返回的相同
                                       ofp_port结构。* /
    uint8_t pad2 [2];          / *填充到64位。* /
    uint32_t配置;              / * OFPPC_ *标志的位图。* /
    uint32_t掩码;              / *要更改的OFPPC_ *标志的位图。* /

    / *端口mod属性列表-0个或更多属性* /
    p_port_mod_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_port_mod）== 32）;

/ * ## ------------------------ ## * /
/ * ## OpenFlow标头类型。## * /
/ * ## ------------------------ ## * /

/ *标头类型结构。* /
struct ofp_header_type {
    uint16_t     命名空间      / * OFPHTN_ *之一。* /
    uint16_t     ns_type;      / *在名称空间中输入。* /
};
OFP_ASSERT（sizeof（struct ofp_header_type）== 4）;

ofp_header_type_namespaces的枚举{
    OFPHTN_ONF          = 0，/ * ONF名称空间。* /
    OFPHTN_ETHERTYPE = 1    / * ns_type是一个以太类型。* /
    OFPHTN_IP_PROTO  = 2    / * ns_type是IP协议号。* /
    OFPHTN_UDP_TCP_PORT / * ns_type是TCP或UDP端口。* /
    OFPHTN_IPV4_OPTION      / * ns_type是IPv4选项号。* /
};

枚举ofp_header_type_onf {
    OFPHTO_ETHERNET= 0，/ *以太网（DIX或IEEE 802.3）-默认。* /
    OFPHTO_NO_HEADER= 1    / *没有标题，例如 电路开关。* /
    OFPHTO_OXM_EXPERIMENTER = 0xFFFF，/ *使用实验者OXM。* /
};

/ * ## ------------------------ ## * /
/ * ## OpenFlow可扩展匹配。## * /
/ * ## ------------------------ ## * /

/ *匹配类型表示匹配结构（构成
 *匹配）。匹配类型放在开头的类型字段中
 *所有匹配结构。“ OpenFlow可扩展匹配”类型对应
 *为以下所述的OXM TLV格式，并且必须由所有OpenFlow支持
 *开关。定义其他匹配类型的扩展程序可能会发布在
 * ONF Wiki。对扩展的支持是可选的。
 * /
of_p_match_type的枚举{
    OFPMT_STANDARD = 0，/ *不推荐使用。* /
    OFPMT_OXM= 1            / * OpenFlow可扩展匹配* /
};

/ *与流量匹配的字段* /
struct ofp_match {
    uint16_t类型;               / * OFPMT_ *之一* /
    uint16_t长度;               / * ofp_match的长度（不包括填充）* /
    /* 其次是:
     *     -恰好（长度-4）（可能为0）个字节包含OXM TLV，然后
     *     -正好（（length + 7）/ 8 * 8-length）（0至7之间）个字节
     *       全零字节
```



```
     *总之，根据需要填充ofp_match，以使其整体大小
     * 8的倍数，以保留使用它的结构中的对齐方式。
     * /
    uint8_t oxm_fields [0];     / * 0个或更多OXM匹配字段* /
    uint8_t pad [4];            / *零字节-有关大小调整，请参见上文* /
};
OFP_ASSERT（sizeof（struct ofp_match）== 8）;

/ * OXM TLV标头的组件。
 *这些宏对于实验者类无效，而对于
 *实验者类将取决于所使用的实验者标头。* /
# 定义OXM_HEADER （CLASS，FIELD，HASMASK，LENGTH）\
    （（（CLASS）<< 16）|（（FIELD）<< 9）|（（HASMASK）<< 8）|（长度））
# 定义OXM_HEADER（CLASS，FIELD，LENGTH）\
    OXM_HEADER （CLASS，FIELD，0，LENGTH）
# 定义OXM_HEADER_W（CLASS，FIELD，LENGTH）\
    OXM_HEADER （CLASS，FIELD，1，（LENGTH）* 2）
# 定义OXM_CLASS（HEADER）（（HEADER）>> 16）
# 定义OXM_FIELD（标题）（（（标题）>> 9）和0x7f）
# 定义OXM_TYPE（HEADER）（（（HEADER）>> 9）&0x7fffff）
# 定义OXM_HASMASK（HEADER）（（（HEADER）>> 8）&1）
# 定义OXM_LENGTH（标题）（（标题）和0xff）

# 定义OXM_MAKE_WILD_HEADER（HEADER）\
    OXM_HEADER_W（OXM_CLASS（HEADER），OXM_FIELD（HEADER），OXM_LENGTH（HEADER））

/ * OXM类ID。
 *高阶位区分保留类和成员类。
 *类0x0000至0x7FFF是由ONF分配的成员类。
 * 0x8000至0xFFFE类是保留类，为标准化保留。
 * /
of_p_oxm_class的枚举{
    OFPXMC_NXM_0   = 0x0000，/ *与NXM的向后兼容性* /
    OFPXMC_NXM_1   = 0x0001，/ *与NXM的向后兼容性* /
    OFPXMC_OPENFLOW_BASIC = 0x8000，/ * OpenFlow的基本类* /
    OFPXMC_PACKET_REGS = 0x0001，/ *数据包寄存器（管道字段）。* /
    OFPXMC_EXPERIMENTER = 0xFFFF，/ *实验者课程* /
};

/ * OXM流匹配字段类型，用于OpenFlow基本类。* /
列举oxm_ofb_match_fields {
    OFPXMT_OFB_IN_PORT 0，/ *开关输入端口。* /
    OFPXMT_OFB_IN_PHY_PORT / *切换物理输入端口。* /
    OFPXMT_OFB_METADATA / *在表之间传递的元数据。* /
    OFPXMT_OFB_ETH_DST，/ *以太网目标地址。* /
    OFPXMT_OFB_ETH_SRC，/ *以太网源地址。* /
    OFPXMT_OFB_ETH_TYPE，/ *以太网帧类型。* /
    OFPXMT_OFB_VLAN_VID，/ * VLAN ID。* /
    OFPXMT_OFB_VLAN_PCP / * VLAN优先级。* /
    OFPXMT_OFB_IP_DSCP 8，/ * IP DSCP（ToS字段中的6位）。* /
    OFPXMT_OFB_IP_ECN= 9，/ * IP ECN（ToS字段中的2位）。* /
    OFPXMT_OFB_IP_PROTO，/ * IP协议。* /
    OFPXMT_OFB_IPV4_SRC，/ * IPv4源地址。* /
    OFPXMT_OFB_IPV4_DST，/ * IPv4目标地址。* /
    OFPXMT_OFB_TCP_SRC，/ * TCP源端口。* /
    OFPXMT_OFB_TCP_DST，/ * TCP目标端口。* /
    OFPXMT_OFB_UDP_SRC，/ * UDP源端口。* /
    OFPXMT_OFB_UDP_DST，/ * UDP目标端口。* /
    OFPXMT_OFB_SCTP_SRC
```

```
    OFPXMT_OFB_SCTP_DST = 19，/* SCTP目标端口。* /
    OFPXMT_OFB_ICMPV4_TYPE = 20，/* ICMP类型。* /
    OFPXMT_OFB_ICMPV4_CODE = 21，/* ICMP代码。* /
    OFPXMT_OFB_ARP_OP = 22，/* ARP操作码。* /
    OFPXMT_OFB_ARP_SPA = 23，/* ARP源IPv4地址。* /
    OFPXMT_OFB_ARP_TPA = 24，/* ARP目标IPv4地址。* /
    OFPXMT_OFB_ARP_SHA = 25，/* ARP源硬件地址。* /
    OFPXMT_OFB_ARP_THA = 26，/* ARP目标硬件地址。* /
    OFPXMT_OFB_IPV6_SRC = 27，/* IPv6源地址。* /
    OFPXMT_OFB_IPV6_DST = 28，/* IPv6目标地址。* /
    OFPXMT_OFB_IPV6_FLABEL = 29，/* IPv6流标签* /
    OFPXMT_OFB_ICMPV6_TYPE = 30，/* ICMPv6类型。* /
    OFPXMT_OFB_ICMPV6_CODE = 31，/* ICMPv6代码。* /
    OFPXMT_OFB_IPV6_ND_TARGET = 32，/* ND的目标地址。* /
    OFPXMT_OFB_IPV6_ND_SLL = 33，/* ND的源链路层。* /
    OFPXMT_OFB_IPV6_ND_TLL = 34，/* ND的目标链路层。* /
    OFPXMT_OFB_MPLS_LABEL = 35，/* MPLS标签。* /
    OFPXMT_OFB_MPLS_TC = 36，/* MPLS TC。* /
    OFPXMT_OFP_MPLS_BOS = 37，/* MPLS BoS位。* /
    OFPXMT_OFB_PBB_ISID = 38，/* PBB I-SID。* /
    OFPXMT_OFB_TUNNEL_ID = 39，/*逻辑端口元数据。* /
    OFPXMT_OFB_IPV6_EXTHDR = 40，/* IPv6扩展头伪字段* /
    OFPXMT_OFB_PBB_UCA = 41，/* PBB UCA标头字段。* /
    OFPXMT_OFB_TCP_FLAGS = 42，/* TCP标志。* /
    OFPXMT_OFB_ACTSET_OUTPUT = 43，/*操作集元数据的输出端口。* /
    OFPXMT_OFB_PACKET_TYPE = 44，/*数据包类型值。* /
};

#define OFPXMT_OFB_ALL（UINT64_C（1）<< 45）-1）

/ *接收数据包的OpenFlow端口。
```

```
 *可以是物理端口，逻辑端口或保留端口OFPP_LOCAL
 *
 *先决条件：无。
 *
 *格式：网络字节顺序的32位整数。
 *
 *遮罩：不可遮罩。* /
#define OXM_OF_IN_PORT OXM_HEADER（0x8000，OFPXMT_OFB_IN_PORT，4）

/ *接收数据包的物理端口。
 *
 *考虑在通过链路定义的隧道接口上接收到的数据包
 *具有两个物理端口成员的聚合组（LAG）。 如果隧道
 * interface是绑定到OpenFlow的逻辑端口。 在这种情况下，
 * OFPXMT_OF_IN_PORT是隧道的端口号，而OFPXMT_OF_IN_PHY_PORT是
 *在其上配置隧道的LAG的物理端口号。
 *
 *当直接在物理端口上接收到数据包且未由
 *逻辑端口OFPXMT_OF_IN_PORT和OFPXMT_OF_IN_PHY_PORT具有相同的端口
 *值。
 *
 *此字段通常在常规比赛中不可用，仅可用
 *在ofp_packet_in消息中，不同于OXM_OF_IN_PORT。
 *
 *先决条件：必须存在OXM_OF_IN_PORT。
 *
 *格式：网络字节顺序的32位整数。
 *
 *遮罩：不可遮罩。* /
#define OXM_OF_IN_PHY_PORT OXM_HEADER（0x8000，OFPXMT_OFB_IN_PHY_PORT，4）

/ *表元数据。
 *
 *先决条件：无。
 *
 *格式：网络字节顺序的64位整数。
 *
 *遮罩：任意遮罩。
 * /
# 定义OXM_OF_METADATA OXM_HEADER（0x8000，OFPXMT_OFB_METADATA，8）
# 定义OXM_OF_METADATA_W OXM_HEADER_W（0x8000，OFPXMT_OFB_METADATA，8）

/ *以太网头中的源或目标地址。
 *
 *先决条件：无。
 *
 *格式：48位以太网MAC地址。
 *
 *遮罩：任意遮罩。* /
# 定义OXM_OF_ETH_DST OXM_HEADER（0x8000，OFPXMT_OFB_ETH_DST，6）
# 定义OXM_OF_ETH_DST_W OXM_HEADER_W（0x8000，OFPXMT_OFB_ETH_DST，6）
# 定义OXM_OF_ETH_SRC OXM_HEADER（0x8000，OFPXMT_OFB_ETH_SRC，6）
# 定义OXM_OF_ETH_SRC_W OXM_HEADER_W（0x8000，OFPXMT_OFB_ETH_SRC，6）

/ *数据包的以太网类型。
 *
 *先决条件：无。
 *
 *格式：网络字节顺序的16位整数。
 *
 *遮罩：不可遮罩。* /
# 定义OXM_OF_ETH_TYPE OXM_HEADER（0x8000，OFPXMT_OFB_ETH_TYPE，2）

/ * VLAN ID是12位，因此我们可以使用全部16位来表示
 *特殊情况。
 * /
of_plan_id的枚举{
    OFPVID_PRESENT = 0x1000，/ *表示已设置VLAN ID的位* /
    OFPVID_NONE = 0x0000，/ *未设置VLAN ID。* /
};
/ *定义兼容性* /
# 定义OFP_VLAN_NONE OFPVID_NONE

/ * 802.1Q VID。
 *
 *对于具有802.1Q标头的数据包，这是来自IP地址的VLAN-ID（VID）
 *最外面的标签，CFI位强制为1。对于没有802.1Q的数据包
 *标头，其值为OFPVID_NONE。
 *
 *先决条件：无。
 *
 *格式：网络字节顺序的16位整数，其中第13位指示
 *存在VLAN标头和3个最高有效位（强制为0）。
 *只有低13位才有意义。
 *
 *遮罩：任意遮罩。
 *
 *此字段可以以多种方式使用：
 *
 *    -如果完全不受限制，则nx_match匹配没有
```

第210页 OpenFlow交换机规格 版本1.5.1

```
 *      802.1Q标头或具有任何VID值的802.1Q标头。
 *
 *    -测试与0x0的完全匹配仅匹配没有
 *      802.1Q标头
 *
 *    -测试与CFI = 1的VID值完全匹配的数据包
 *      具有具有指定VID的802.1Q标头的文件。
 *
 *    -测试CFI = 0时非零VID值是否完全匹配
 *      没有道理。      交换机可能会拒绝此组合。
 *
 *    -使用nxm_value = 0，nxm_mask = 0x0fff进行测试，以匹配没有802.1Q的数据包
 *      标头或带有VID为0的802.1Q标头。
 *
 *    -使用nxm_value = 0x1000进行测试，nxm_mask = 0x1000匹配具有以下内容的数据包：
 *      具有任何VID值的802.1Q标头。
 */
#定义OXM_OF_VLAN_VID OXM_HEADER（0x8000，OFPXMT_OFB_VLAN_VID，2）
#定义OXM_OF_VLAN_VID_W OXM_HEADER_W（0x8000，OFPXMT_OFB_VLAN_VID，2）

/ * 802.1Q PCP。
 *
 *对于具有802.1Q标头的数据包，这是来自
 *最外面的标签。对于没有802.1Q标头的数据包，它具有价值
 * 0。
 *
 *先决条件：OXM_OF_VLAN_VID必须不同于OFPVID_NONE。
 *
 *格式：8位整数，其中5个最高有效位强制为0。
 *只有低3位才有意义。
 *
 *遮罩：不可遮罩。
 */
#定义OXM_OF_VLAN_PCP OXM_HEADER（0x8000，OFPXMT_OFB_VLAN_PCP，1）

/ * IP标头的区分服务代码点（DSCP）位。
 * IPv4 ToS字段或IPv6 Traffic Class字段的一部分。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须为0x0800或0x86dd，
 *    或PACKET_TYPE必须为（1,0x800）或（1,0x86dd）。
 *
 *格式：8位整数，其中2个最高有效位强制为0。
 *只有低6位才有意义。
 *
 *遮罩：不可遮罩。* /
#定义OXM_OF_IP_DSCP OXM_HEADER（0x8000，OFPXMT_OFB_IP_DSCP，1）

/ * IP标头的ECN位。
 * IPv4 ToS字段或IPv6 Traffic Class字段的一部分。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须为0x0800或0x86dd，
 *    或PACKET_TYPE必须为（1,0x800）或（1,0x86dd）。
 *
 *格式：8位整数，其中6个最高有效位强制为0。
 *只有低2位才有意义。
 *
 *遮罩：不可遮罩。* /
#定义OXM_OF_IP_ECN OXM_HEADER（0x8000，OFPXMT_OFB_IP_ECN，1）

/ * IP标头中的“协议”字节。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须为0x0800或0x86dd，
 *    或PACKET_TYPE必须为（1,0x800）或（1,0x86dd）。
 *
 *格式：8位整数。
 *
 *遮罩：不可遮罩。* /
#define OXM_OF_IP_PROTO OXM_HEADER（0x8000，OFPXMT_OFB_IP_PROTO，1）

/ * IP标头中的源或目标地址。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须与0x0800完全匹配，
 *    或PACKET_TYPE必须完全匹配（1,0x800）。
 *
 *格式：网络字节顺序的32位整数。
 *
 *遮罩：任意遮罩。
 */
#定义OXM_OF_IPV4_SRC OXM_HEADER（0x8000，OFPXMT_OFB_IPV4_SRC，4）
#定义OXM_OF_IPV4_SRC_W OXM_HEADER_W（0x8000，OFPXMT_OFB_IPV4_SRC，4）
#定义OXM_OF_IPV4_DST OXM_HEADER（0x8000，OFPXMT_OFB_IPV4_DST，4）
#定义OXM_OF_IPV4_DST_W OXM_HEADER_W（0x8000，OFPXMT_OFB_IPV4_DST，4）

/ * TCP头中的源端口或目标端口。
 *
 *先决条件：
```


第211页 OpenFlow交换机规格 版本1.5.1

```
 *    OXM_OF_ETH_TYPE必须为0x0800或0x86dd，
 *    或PACKET_TYPE必须为（1,0x800）或（1,0x86dd），
 *    OXM_OF_IP_PROTO必须完全匹配6。
 *
 *格式：网络字节顺序的16位整数。
 *
 *遮罩：不可遮罩。* /
#定义OXM_OF_TCP_SRC OXM_HEADER（0x8000，OFPXMT_OFB_TCP_SRC，2）
#定义OXM_OF_TCP_DST OXM_HEADER（0x8000，OFPXMT_OFB_TCP_DST，2）

/ * TCP标头中的标志。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须为0x0800或0x86dd。
 *    OXM_OF_IP_PROTO必须完全匹配6。
 *
 *格式：16位整数，其中4个最高有效位强制为0。
 *
```

#定义OXM_OF_TCP_FLAGS OXM_HEADER（0x8000，OFPXMT_OFB_TCP_FLAGS，2）
#定义OXM_OF_TCP_FLAGS_W OXM_HEADER_W（0x8000，OFPXMT_OFB_TCP_FLAGS，2）

/* UDP标头中的源端口或目标端口。
 *
 *先决条件：
 *OXM_OF_ETH_TYPE必须匹配0x0800或0x86dd，
 *或PACKET_TYPE必须为（1,0x800）或（1,0x86dd），
 *OXM_OF_IP_PROTO必须完全匹配17。
 *
 *格式：网络字节顺序的16位整数。
 *
 遮罩：不可遮罩。 /
#定义OXM_OF_UDP_SRC OXM_HEADER（0x8000，OFPXMT_OFB_UDP_SRC，2）
#定义OXM_OF_UDP_DST OXM_HEADER（0x8000，OFPXMT_OFB_UDP_DST，2）

/* SCTP标头中的源端口或目标端口。
 *
 *先决条件：
 *OXM_OF_ETH_TYPE必须匹配0x0800或0x86dd，
 *或PACKET_TYPE必须为（1,0x800）或（1,0x86dd），
 *OXM_OF_IP_PROTO必须完全匹配132。
 *
 *格式：网络字节顺序的16位整数。
 *
 遮罩：不可遮罩。 /
#定义OXM_OF_SCTP_SRC OXM_HEADER（0x8000，OFPXMT_OFB_SCTP_SRC，2）
#定义OXM_OF_SCTP_DST OXM_HEADER（0x8000，OFPXMT_OFB_SCTP_DST，2）

/* ICMP标头中的类型或代码。
 *
 *先决条件：
 *OXM_OF_ETH_TYPE必须与0x0800完全匹配，
 *或PACKET_TYPE必须完全匹配（1,0x800）。
 *OXM_OF_IP_PROTO必须完全匹配1。
 *
 *格式：8位整数。
 *
 遮罩：不可遮罩。 /
#定义OXM_OF_ICMPV4_TYPE OXM_HEADER（0x8000，OFPXMT_OFB_ICMPV4_TYPE，1）
#定义OXM_OF_ICMPV4_CODE OXM_HEADER（0x8000，OFPXMT_OFB_ICMPV4_CODE，1）

/* ARP操作码。
 *
 *对于以太网+ IP ARP数据包，ARP标头中的操作码。 总是0
 *除此以外。
 *
 *先决条件：OXM_OF_ETH_TYPE必须与0x0806完全匹配。
 *
 *格式：网络字节顺序的16位整数。
 *
 遮罩：不可遮罩。 /
#定义OXM_OF_ARP_OP OXM_HEADER（0x8000，OFPXMT_OFB_ARP_OP，2）

/ *对于以太网+ IP ARP数据包，源或目标协议地址
 *在ARP标头中。 否则始终为0。
 *
 *先决条件：OXM_OF_ETH_TYPE必须与0x0806完全匹配。
 *
 *格式：网络字节顺序的32位整数。
 *
 *遮罩：任意遮罩。
 * /
#define OXM_OF_ARP_SPA OXM_HEADER（0x8000，OFPXMT_OFB_ARP_SPA，4）
#define OXM_OF_ARP_SPA_W OXM_HEADER_W（0x8000，OFPXMT_OFB_ARP_SPA，4）
#定义OXM_OF_ARP_TPA OXM_HEADER（0x8000，OFPXMT_OFB_ARP_TPA，4）
#定义OXM_OF_ARP_TPA_W OXM_HEADER_W（0x8000，OFPXMT_OFB_ARP_TPA，4）

/ *对于以太网+ IP ARP数据包，源或目标硬件地址
 *在ARP标头中。 否则始终为0。
 *

 *先决条件：OXM_OF_ETH_TYPE必须与0x0806完全匹配。
 *
 *格式：48位以太网MAC地址。
 *
 遮罩：不可遮罩。 /
#define OXM_OF_ARP_SHA OXM_HEADER（0x8000，OFPXMT_OFB_ARP_SHA，6）
#定义OXM_OF_ARP_THA OXM_HEADER（0x8000，OFPXMT_OFB_ARP_THA，6）

/* IPv6标头中的源或目标地址。
 *
 *先决条件：
 *OXM_OF_ETH_TYPE必须完全匹配0x86dd，
 *或PACKET_TYPE必须完全匹配（1,0x86dd）。
 *
 *格式：128位IPv6地址。
 *
 *遮罩：任意遮罩。
 * /
#定义OXM_OF_IPV6_SRC OXM_HEADER（0x8000，OFPXMT_OFB_IPV6_SRC，16）
#定义OXM_OF_IPV6_SRC_W OXM_HEADER_W（0x8000，OFPXMT_OFB_IPV6_SRC，16）
#定义OXM_OF_IPV6_DST OXM_HEADER（0x8000，OFPXMT_OFB_IPV6_DST，16）
#定义OXM_OF_IPV6_DST_W OXM_HEADER_W（0x8000，OFPXMT_OFB_IPV6_DST，16）

/* IPv6流标签
 *
 *先决条件：
 *OXM_OF_ETH_TYPE必须完全匹配0x86dd，
 *或PACKET_TYPE必须完全匹配（1,0x86dd）。
 *
 *格式：32位整数，其中12个最高有效位强制为0。
 *只有低20位才有意义。
 *
 *遮罩：任意遮罩。
 * /
#定义OXM_OF_IPV6_FLABEL OXM_HEADER（0x8000，OFPXMT_OFB_IPV6_FLABEL，4）
#定义OXM_OF_IPV6_FLABEL_W OXM_HEADER_W（0x8000，OFPXMT_OFB_IPV6_FLABEL，4）

/* ICMPv6标头中的类型或代码。
 *
 *先决条件：
 *OXM_OF_ETH_TYPE必须完全匹配0x86dd，
 *或PACKET_TYPE必须完全匹配（1,0x86dd），
 *OXM_OF_IP_PROTO必须完全匹配58。
 *
 *格式：8位整数。
 *
 遮罩：不可遮罩。 /
#define OXM_OF_ICMPV6_TYPE OXM_HEADER（0x8000，OFPXMT_OFB_ICMPV6_TYPE，1）
#define OXM_OF_ICMPV6_CODE OXM_HEADER（0x8000，OFPXMT_OFB_ICMPV6_CODE，1）

/* IPv6邻居发现消息中的目标地址。
 *
 *先决条件：
 *OXM_OF_ETH_TYPE必须完全匹配0x86dd，
 *或PACKET_TYPE必须完全匹配（1,0x86dd），

```
 *
 *    OXM_OF_IP_PROTO必须完全匹配58或136。
 *
 *格式：128位IPv6地址。
 *
 *遮罩：不可遮罩。* /
# 定义OXM_OF_IPV6_ND_TARGET OXM_HEADER（0x8000，OFPXMT_OFB_IPV6_ND_TARGET，16）

/ * IPv6邻居发现中的源链接层地址选项
 * 信息。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x86dd，
 *    或PACKET_TYPE必须完全匹配（1,0x86dd），
 *    OXM_OF_IP_PROTO必须完全匹配58。
 *    OXM_OF_ICMPV6_TYPE必须完全为135。
 *
 *格式：48位以太网MAC地址。
 *
 *遮罩：不可遮罩。* /
# 定义OXM_OF_IPV6_ND_SLL OXM_HEADER（0x8000，OFPXMT_OFB_IPV6_ND_SLL，6）

/ * IPv6邻居发现中的目标链路层地址选项
 * 信息。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x86dd，
 *    或PACKET_TYPE必须完全匹配（1,0x86dd），
 *    OXM_OF_IP_PROTO必须完全匹配58。
 *    OXM_OF_ICMPV6_TYPE必须恰好是136。
 *
 *格式：48位以太网MAC地址。
 *
 *遮罩：不可遮罩。* /
# 定义OXM_OF_IPV6_ND_TLL OXM_HEADER（0x8000，OFPXMT_OFB_IPV6_ND_TLL，6）
```



```
/ *第一个MPLS填充程序标头中的LABEL。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x8847或0x8848。
 *
 *格式：网络字节顺序的32位整数，最高12位
 *位强制为0。只有低20位才有意义。
 *
 *遮罩：不可遮罩。* /
#define OXM_OF_MPLS_LABEL OXM_HEADER（0x8000，OFPXMT_OFB_MPLS_LABEL，4）

/ *第一个MPLS填充标头中的TC。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x8847或0x8848。
 *
 *格式：8位整数，其中5个最高有效位强制为0。
 *只有低3位才有意义。
 *
 *遮罩：不可遮罩。* /
# 定义OXM_OF_MPLS_TC OXM_HEADER（0x8000，OFPXMT_OFB_MPLS_TC，1）

/ *第一个MPLS填充标头中的BoS位。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x8847或0x8848。
 *
 *格式：8位整数，其中7个最高有效位被强制为0。
 *只有最低位才有意义。
 *
 *遮罩：不可遮罩。* /
#define OXM_OF_MPLS_BOS OXM_HEADER（0x8000，OFPXMT_OFB_MPLS_BOS，1）

/ * IEEE 802.1ah I-SID。
 *
 *对于带有PBB标头的数据包，这是来自
 *最外面的服务标签。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x88E7。
 *
 *格式：网络字节顺序的24位整数。
 *
 *遮罩：任意遮罩。* /
# 定义OXM_OF_PBB_ISID OXM_HEADER（0x8000，OFPXMT_OFB_PBB_ISID，3）
# 定义OXM_OF_PBB_ISID_W OXM_HEADER_W（0x8000，OFPXMT_OFB_PBB_ISID，3）

/ *逻辑端口元数据。
 *
 *与逻辑端口关联的元数据。
 *如果逻辑端口执行封装和解封装，则此
 *是封装头中的多路分解字段。
 *例如，对于通过GRE隧道接收的包含（32位）密钥的数据包，
 *密钥存储在低32位中，高位清零。
 *对于MPLS逻辑端口，低20位表示MPLS标签。
 *对于VxLAN逻辑端口，低24位表示VNI。
 *如果未通过逻辑端口接收到数据包，则该值为0。
 *
 *先决条件：无。
 *
 *格式：网络字节顺序的64位整数。
 *
 *遮罩：任意遮罩。* /
# 定义OXM_OF_TUNNEL_ID OXM_HEADER（0x8000，OFPXMT_OFB_TUNNEL_ID，8）
# 定义OXM_OF_TUNNEL_ID_W OXM_HEADER_W（0x8000，OFPXMT_OFB_TUNNEL_ID，8）

/ * IPv6扩展头伪字段。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x86dd，
 *    或PACKET_TYPE必须完全匹配（1,0x86dd）。
 *
 *格式：16位整数，其中7个最高有效位被强制为0。
 *只有低9位才有意义。
 *
 *遮罩：可遮罩。* /
# 定义OXM_OF_IPV6_EXTHDR OXM_HEADER（0x8000，OFPXMT_OFB_IPV6_EXTHDR，2）
# 定义OXM_OF_IPV6_EXTHDR_W OXM_HEADER_W（0x8000，OFPXMT_OFB_IPV6_EXTHDR，2）

/ * IPv6扩展头伪字段的位定义。* /
枚举ofp_ipv6exthdr_flags {
    OFPIEH_NONEXT = 1 << 0，/ *遇到“没有下一个标头”。* /
    OFPIEH_ESP = 1 << 1，    / *存在加密的有效载荷头。* /
    OFPIEH_AUTH = 1 << 2，   / *存在身份验证标头。* /
    OFPIEH_DEST = 1 << 3，   / *存在1或2个dest标头。* /
    OFPIEH_FRAG = 1 << 4，   / *存在片段头。* /
    OFPIEH_ROUTER = 1 << 5，/ *存在路由器标头。* /
    OFPIEH_HOP = 1 << 6，    / *存在逐跳标题。* /
    OFPIEH_UNREP = 1 << 7，  / *遇到意外的重复。* /
```


第214页 OpenFlow交换机规格 版本1.5.1

```
    OFPIEH_UNSEQ = 1 << 8,     /*遇到意外的排序。*/
};

/* IEEE 802.1ah UCA。
 *
 *对于带有PBB标头的数据包，这是UCA（使用客户地址）
 *来自最外面的服务标签。
 *
 *先决条件：
 *    OXM_OF_ETH_TYPE必须完全匹配0x88E7。
 *
 *格式：8位整数，其中7个最高有效位被强制为0。
 *仅低1位有意义。
 *
 *遮罩：不可遮罩。*/
#定义OXM_OF_PBB_UCA OXM_HEADER（0x8000，OFPXMT_OFB_PBB_UCA，1）

/*操作集元数据的输出端口。
 *
 *表示操作集中的转发决策的元数据。
 *如果动作集包含输出动作，则此字段等于
 *输出端口。
 *否则，该字段等于其初始值OFPP_UNSET。
 *
 *先决条件：无。
 *
 *格式：网络字节顺序的32位整数。
 *
 *遮罩：不可遮罩。*/
#define OXM_OF_ACTSET_OUTPUT OXM_HEADER（0x8000，OFPXMT_OFB_ACTSET_OUTPUT，4）

/*数据包类型。
 *
 *数据包类型以识别数据包。这是规范的标头类型
 *最外面的标题。
 *如果未指定，则默认为以太网标头类型。
 *
 *先决条件：无。
 *
 *格式：名称空间的16位整数，后跟ns_type的16位整数。
 *
 *遮罩：不可遮罩。*/
#定义OXM_OF_PACKET_TYPE OXM_HEADER（0x8000，OFPXMT_OFB_PACKET_TYPE，4）

/* OXM实验者匹配字段的标题。
 *实验者类不应使用OXM_HEADER（）宏进行定义
 *由于此额外的标题而产生的字段。*/
struct ofp_oxm_experimenter_header {
    uint32_t oxm_header;       /* oxm_class = OFPXMC_EXPERIMENTER */
    uint32_t实验者；              /*实验者ID。*/
};
OFP_ASSERT（sizeof（p_oxm_experimenter_header的结构）== 8）；

/*数据包寄存器（暂存空间）。
 *
 *与数据包关联的存储寄存器。
 *当数据包进入管道时设置为零。
 *可用于通过管道存储值。
 *通常不可用作匹配字段。
 *开关实现可变数量的寄存器，可以为0。
 *
 *先决条件：无。
 *
 *格式：网络字节顺序的64位整数。
 *
 *遮罩：任意遮罩。
 */
#定义OXM_OF_PKT_REG（N） OXM_HEADER（0x8001，N，8）
#定义OXM_OF_PKT_REG_W（N） OXM_HEADER_W（0x8001，N，8）

/* ## ------------------------- ## */
/* ## OpenFlow可扩展统计信息。## */
/* ## ------------------------- ## */

/*流统计结构-统计字段列表。*/
struct ofp_stats {
    uint16_t保留；                /*保留以供将来使用，当前为零。*/
    uint16_t长度；                /* ofp_stats的长度（不包括填充）*/
    /*其次是：
     *    -恰好（长度-4）（可能为0）个字节包含OXS TLV，然后
     *    -正好（（length + 7）/ 8 * 8-length）（0至7之间）个字节
     *全零字节
     *总而言之，ofp_stats根据需要进行填充，以使其整体大小
     * 8的倍数，以保留使用它的结构中的对齐方式。
     */
    uint8_t oxs_fields [0];       /* 0个或更多OXS统计字段*/
    uint8_t pad [4];              /*零字节-有关大小调整，请参见上文*/
};
OFP_ASSERT（sizeof（struct ofp_stats）== 8）；
```


第215页 OpenFlow交换机规格 版本1.5.1

```
/* OXS TLV标头的组件。*/
#定义OXS_HEADER__（类，字段，保留，长度）\
    （（（CLASS）<< 16）|（（FIELD）<< 9）|（（保留）<< 8）|（长度））
#定义OXS_HEADER（CLASS，FIELD，LENGTH）\
    OXS_HEADER__（CLASS，FIELD，0，LENGTH）
#定义OXS_CLASS（HEADER）（（HEADER）>> 16）
#定义OXS_FIELD（标题）（（（标题）>> 9）和0x7f）
#定义OXS_TYPE（HEADER）（（（HEADER）>> 9）&0x7fffff）
#定义OXS_RESERVED（HEADER）（（（（HEADER）>> 8）&1）
#定义OXS_LENGTH（标题）（（标题）和0xff）

/* OXS类别ID。
 *高阶位区分保留类和成员类。
 *类0x0000至0x7FFF是由ONF分配的成员类。
```

```
 * 0x8000至0xFFFE类是保留类，为标准化保留。
of_p_oxs_class的枚举{
    OFPXSC_OPENFLOW_BASIC = 0x8002，/ *OpenFlow的基本统计信息类* /
    OFPXSC_EXPERIMENTER = 0xFFFF，/ *实验者课程* /
};

/ * OpenFlow基本类的OXS流统计字段类型。* /
列举oxs_ofb_stat_fields {
    OFPXST_OFB_DURATION = 0，/ *时间流条目已激活。* /
    OFPXST_OFB_IDLE_TIME = 1，/ *时间流条目已空闲。* /
    OFPXST_OFB_FLOW_COUNT = 2，/ *聚合流条目数。* /
    OFPXST_OFB_PACKET_COUNT = 3，/ *流条目中的包数。* /
    OFPXST_OFB_BYTE_COUNT = 4，/ *流条目中的字节数。* /
};

#define OFPXST_OFB_ALL（UINT64_C（1）<< 6）-1）

/ * OpenFlow流持续时间。
 *时间流条目以秒和纳秒为单位有效。
 *
 *格式：一对以网络字节顺序排列的32位整数。
 *     第一个32位数字是持续时间（以秒为单位）。
 *     第二个32位数字比以秒为单位的持续时间长十亿分之一秒。
 *如果不支持，第二个数字必须设置为零。
 * /
#define OXS_OF_DURATION OXS_HEADER（0x8002，OXPXST_OFB_DURATION，8）

/ * OpenFlow流空闲时间。
 *时间流条目已闲置（无数据包匹配），以秒为单位
 *和纳秒。
 *
 *格式：一对以网络字节顺序排列的32位整数。
 *     第一个32位数字是空闲时间，以秒为单位。
 *     第二个32位数字比空闲时间（以秒为单位）高出纳秒。
 *如果不支持，第二个数字必须设置为零。
 * /
#define OXS_OF_IDLE_TIME OXS_HEADER（0x8002，OXPXST_OFB_IDLE_TIME，8）

/ * OpenFlow流条目计数。
 *聚合流条目数。
 *在OFPMP_AGGREGATE答复中必需，在其他上下文中未定义。
 *
 *格式：网络字节顺序的32位整数。
 * /
# 定义OXS_OF_FLOW_COUNT1 OXS_HEADER（0x8002，OFPXST_OFB_FLOW_COUNT，4）

/ * OpenFlow流条目数据包计数。
 *流表项匹配的报文数。
 *
 *格式：网络字节顺序的64位整数。
 * /
# 定义OXS_OF_PACKET_COUNT1 OXS_HEADER（0x8002，OFPXST_OFB_PACKET_COUNT，8）

/ * OpenFlow流条目数据包计数。
 *流条目匹配的字节数。
 *
 *格式：网络字节顺序的64位整数。
 * /
# 定义OXS_OF_BYTE_COUNT1 OXS_HEADER（0x8002，OXPXST_OFB_BYTE_COUNT，8）

/ * OXS实验者状态字段的标题。* /
struct ofp_oxs_experimenter_header {
    uint32_t oxs_header;        / * oxs_class = OFPXSC_EXPERIMENTER * /
    uint32_t实验者；            / *实验者ID。* /
};
OFP_ASSERT（sizeof（p_oxs_experimenter_header的结构）== 8）；

/ * ## ---------------- ## * /
/ * ## OpenFlow操作 ## * /
/ * ## ---------------- ## * /

of_p_action_type的枚举{
    OFPAT_OUTPUT = 0，/ *输出到交换机端口。* /
    OFPAT_COPY_TTL_OUT = 11，/ *复制TTL“向外”-从最下到最外
                                到最外面* /
    OFPAT_COPY_TTL_IN = 12，/ *将TTL“向内”复制-从最外面到
                               倒数第二个* /
    OFPAT_SET_MPLS_TTL = 15，/ * MPLS TTL * /
    OFPAT_DEC_MPLS_TTL = 16，/ *递减MPLS TTL * /

    OFPAT_PUSH_VLAN = 17，/ *推送新的VLAN标记* /
    OFPAT_POP_VLAN = 18，/ *弹出外部VLAN标记* /
    OFPAT_PUSH_MPLS = 19，/ *推送新的MPLS标签* /
    OFPAT_POP_MPLS = 20，/ *弹出外部MPLS标签* /
    OFPAT_SET_QUEUE = 21，/ *在输出到端口时设置队列ID * /
    OFPAT_GROUP = 22，/ *应用组。* /
    OFPAT_SET_NW_TTL = 23，/ * IP TTL。* /
    OFPAT_DEC_NW_TTL = 24，/ *减少IP TTL。* /
    OFPAT_SET_FIELD = 25，/ *使用OXM TLV格式设置标题字段。* /
    OFPAT_PUSH_PBB = 26，/ *推送新的PBB服务标签（I-TAG）* /
    OFPAT_POP_PBB = 27，/ *弹出外部PBB服务标签（I-TAG）* /
    OFPAT_COPY_FIELD = 28，/ *在标头和寄存器之间复制值。* /
    OFPAT_METER = 29，/ *应用仪表（速率限制器）* /
    OFPAT_EXPERIMENTER = 0xffff
};

/ *所有动作共有的动作标头。                长度包括
 *标头和用于使操作64位对齐的任何填充。
 *注意：动作的长度*必须*始终是8的倍数。* /
struct ofp_action_header {
    uint16_t类型；                  / * OFPAT_ *之一。* /
    uint16_t len;                   / *此结构的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_action_header的结构）== 4）；

of_p_controller_max_len的枚举{
        OFPCML_MAX = 0xffe5，/ *可以使用的最大max_len值
                                请求特定的字节长度。* /
        OFPCML_NO_BUFFER = 0xffff，/ *表示不应该缓冲
                                    应用，整个包将被
                                    发送到控制器。* /
};

/ * OFPAT_OUTPUT的操作结构，该结构将数据包从“端口”发送出去。
 *当“端口”为OFPP_CONTROLLER时，“ max_len”表示最大
 *要发送的字节数。          “ max_len”为零表示没有字节
 *数据包应发送。OFPCML_NO_BUFFER的“ max_len”表示
 *数据包未缓冲，完整的数据包将发送到
 *控制器。* /
struct ofp_action_output {
    uint16_t类型；                  / * OFPAT_OUTPUT。* /
    uint16_t len;                   / *长度为16。* /
    uint32_t端口；                  / *输出端口。* /
    uint16_t max_len;               / *要发送到控制器的最大长度。* /
    uint8_t pad [6];                / *填充到64位。* /
};
```

```
OFP_ASSERT（sizeof（p_action_output的结构）== 16）；
/ * OFPAT_COPY_TTL_OUT、OFPAT_COPY_TTL_IN、
 * OFPAT_DEC_MPLS_TTL、OFPAT_DEC_NW_TTL、OFPAT_POP_VLAN和OFPAT_POP_PBB。* /
struct ofp_action_generic {
    uint16_t类型;                / * OFPAT_ *之一。* /
    uint16_t len;                / *长度为8。* /
    uint8_t pad [4];             / *填充到64位。* /
};
OFP_ASSERT（sizeof（struct ofp_action_generic）== 8）；

/ * OFPAT_SET_MPLS_TTL的操作结构。* /
struct ofp_action_mpls_ttl {
    uint16_t类型;                / * OFPAT_SET_MPLS_TTL。* /
    uint16_t len;                / *长度为8。* /
    uint8_t mpls_ttl;            / * MPLS TTL * /
    uint8_t pad [3];
};
OFP_ASSERT（sizeof（struct ofp_action_mpls_ttl）== 8）；

/ * OFPAT_PUSH_VLAN / MPLS / PBB的操作结构。* /
struct ofp_action_push {
    uint16_t类型;                / * OFPAT_PUSH_VLAN / MPLS / PBB。* /
    uint16_t len;                / *长度为8。* /
    uint16_t ethertype;          / *以太网类型* /
    uint8_t pad [2];
};
OFP_ASSERT（sizeof（struct ofp_action_push）== 8）；

/ * OFPAT_POP_MPLS的操作结构。* /
struct ofp_action_pop_mpls {
    uint16_t类型;                / * OFPAT_POP_MPLS。* /
    uint16_t len;                / *长度为8。* /
    uint16_t ethertype;          / *以太网类型* /
    uint8_t pad [2];
};
OFP_ASSERT（sizeof（struct ofp_action_pop_mpls）== 8）；
```



```
/ * OFPAT_SET_QUEUE的操作结构。* /
struct ofp_action_set_queue {
    uint16_t类型;                / * OFPAT_SET_QUEUE。* /
    uint16_t len;                / * Len是8。* /
    uint32_t queue_id;           / *数据包的队列ID。* /
};
OFP_ASSERT（sizeof（p_action_set_queue的结构）== 8）；

/ * OFPAT_GROUP的操作结构。* /
struct ofp_action_group {
    uint16_t类型;                / * OFPAT_GROUP。* /
    uint16_t len;                / *长度为8。* /
    uint32_t group_id;           / *组标识符。* /
};
OFP_ASSERT（sizeof（p_action_group的结构）== 8）；

/ * OFPAT_SET_NW_TTL的操作结构。* /
struct ofp_action_nw_ttl {
    uint16_t类型;                / * OFPAT_SET_NW_TTL。* /
    uint16_t len;                / *长度为8。* /
    uint8_t nw_ttl;              / * IP TTL * /
    uint8_t pad [3];
};
OFP_ASSERT（sizeof（p_action_nw_ttl的结构）== 8）；

/ * OFPAT_SET_FIELD的动作结构。* /
struct ofp_action_set_field {
    uint16_t类型;                / * OFPAT_SET_FIELD。* /
    uint16_t len;                / *长度填充为64位。* /
    /* 其次是：
     *   恰好（4 + oxm_length）个字节包含一个OXM TLV，然后
     *   -完全（（8 + oxm_length）+ 7）/ 8 * 8-（8 + oxm_length）
     *     （0到7之间）全零字节的字节
     * /
    uint8_t字段[4];              / * OXM TLV-使编译器满意* /
};
OFP_ASSERT（sizeof（p_action_set_field的结构）== 8）；

/ * OFPAT_COPY_FIELD的动作结构。* /
struct ofp_action_copy_field {
    uint16_t类型;                / * OFPAT_COPY_FIELD。* /
    uint16_t len;                / *长度填充为64位。* /
    uint16_t n_bits;             / *要复制的位数。* /
    uint16_t src_offset;         / *源中的起始位偏移量。* /
    uint16_t dst_offset;         / *目标中的起始位偏移量。* /
    uint8_t pad [2];             / *对齐32位。* /
    /* 其次是：
     *   -包含oxm_id的8、12或16个字节，然后
     *   -足够全零的字节（0或4个字节）来构成一个整体
     *     8字节长度的倍数* /
    uint32_t                oxm_ids [0]; / *源和目标OXM标头* /
};
OFP_ASSERT（sizeof（p_action_copy_field的结构）== 12）；

/ * OFPAT_METER的操作结构* /
p_action_meter的结构{
    uint16_t类型;                / * OFPAT_METER * /
    uint16_t len;                / *长度为8。* /
    uint32_t meter_id;           / *仪表实例。* /
};
OFP_ASSERT（sizeof（p_action_meter的结构）== 8）；

/ * OFPAT_EXPERIMENTER的操作标头。
 *身体的其余部分由实验者定义。* /
struct ofp_action_experimenter_header {
    uint16_t类型;                / * OFPAT_EXPERIMENTER。* /
    uint16_t len;                / *长度是8的倍数。* /
    uint32_t实验者;              / *实验者ID。* /
};
OFP_ASSERT（sizeof（p_action_experimenter_header的结构）== 8）；

/ * ## ---------------------- ## * /
/ * ## OpenFlow指令。## * /
/ * ## ---------------------- ## * /

of_p_instruction_type的枚举{
    OFPIT_GOTO_TABLE = 1/ *在查询中设置下一个表
                          管道* /
    OFPIT_WRITE_METADATA = 2 / *设置元数据字段，以供以后在
                          管道* /
    OFPIT_WRITE_ACTIONS = 3 / *将操作写入数据路径操作
                          设置* /
    OFPIT_APPLY_ACTIONS = 4 / *立即应用操作* /
    OFPIT_CLEAR_ACTIONS = 5 / *清除数据路径中的所有操作
                          动作集* /
    OFPIT_DEPRECATED = 6 / *不推荐使用（适用计量器）* /
    OFPIT_STAT_TRIGGER = 7 / *统计触发器* /

    OFPIT_EXPERIMENTER = 0xFFFF
```

```
};                                  / *实验者说明* /
```



```
/ *所有指令共有的指令头。                                    长度包括
 *标头和用于使指令64位对齐的任何填充。
 *注意：指令的长度*必须*始终是8的倍数。* /
struct ofp_instruction_header {
    uint16_t类型;                   / * OFPIT_ *之一。* /
    uint16_t len;                   / *此结构的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_instruction_header的结构）== 4）；

/ * OFPIT_GOTO_TABLE的指令结构* /
struct ofp_instruction_goto_table {
    uint16_t类型;                   / * OFPIT_GOTO_TABLE * /
    uint16_t len;                   / *长度为8。* /
    uint8_t table_id;               / *在查找管道中设置下一个表* /
    uint8_t pad [3];                / *填充到64位。* /
};
OFP_ASSERT（sizeof（p_instruction_goto_table的结构）== 8）；

/ * OFPIT_WRITE_METADATA的指令结构* /
struct ofp_instruction_write_metadata {
    uint16_t类型;                   / * OFPIT_WRITE_METADATA * /
    uint16_t len;                   / *长度为24。* /
    uint8_t pad [4];                / *对齐64位* /
    uint64_t元数据;                 / *要写入的元数据值* /
    uint64_t metadata_mask;         / *元数据写入位掩码* /
};
OFP_ASSERT（sizeof（p_instruction_write_metadata的结构）== 24）；

/ * OFPIT_WRITE / APPLY / CLEAR_ACTIONS的指令结构* /
struct ofp_instruction_actions {
    uint16_t类型;                   / * OFPIT _ * _ ACTIONS之一* /
    uint16_t len;                   / *长度填充为64位。* /
    uint8_t pad [4];                / *对齐64位* /
    struct ofp_action_header actions [0]; / * 0个或更多与之相关的动作
                                            OFPIT_WRITE_ACTIONS和
                                            OFPIT_APPLY_ACTIONS * /
};
OFP_ASSERT（sizeof（struct ofp_instruction_actions）== 8）；

of_p_stat_trigger_flags的枚举{
    OFPSTF_PERIODIC = 0，/ *为所有阈值的倍数触发。* /
    OFPSTF_ONLY_FIRST = 1，/ *仅在第一个达到阈值时触发。* /
};

/ * OFPIT_STAT_TRIGGER的指令结构* /
struct ofp_instruction_stat_trigger {
    uint16_t类型;                   / * OFPIT_STAT_TRIGGER * /
    uint16_t len;                   / *长度填充为64位。* /
    uint32_t标志;                   / * OFPSTF_ *标志的位图。* /
    p_stats阈值的结构;              / *阈值列表。大小可变。* /
};
OFP_ASSERT（sizeof（p_instruction_stat_trigger的结构）== 16）；

/ *实验指令的指令结构* /
struct ofp_instruction_experimenter_header {
    uint16_t类型;                   / * OFPIT_EXPERIMENTER。* /
    uint16_t len;                   / *长度填充为64位。* /
    uint32_t实验者;                 / *实验者ID。* /
    / *实验者定义的任意附加数据。* /
};
OFP_ASSERT（sizeof（p_instruction_experimenter_header的结构）== 8）；

/ * ## ------------------------------ ## * /
/ * ## OpenFlow流程修改。 ## * /
/ * ## ------------------------------ ## * /

of p_flow_mod_command枚举{
    OFPFC_ADD             = 0，/ *新流程。* /
    OFPFC_MODIFY  = 1，/ *修改所有匹配的流。* /
    OFPFC_MODIFY_STRICT = 2，/ *修改严格匹配通配符的条目，并且
                                优先。* /
    OFPFC_DELETE  = 3，/ *删除所有匹配的流。* /
    OFPFC_DELETE_STRICT = 4，/ *删除与通配符严格匹配的条目，并且
                                优先。* /
};

/ *在“ idle_timeout”和“ hard_timeout”中使用的值表示该条目
 *是永久性的。* /
＃定义OFP_FLOW_PERMANENT 0

/ *默认情况下，在中间选择一个优先级。* /
＃定义OFP_DEFAULT_PRIORITY 0x8000

of_p_flow_mod_flags的枚举{
    OFPFF_SEND_FLOW_REM = 1 <<0，/ *发送流时发送流删除消息
                                    *过期或被删除。* /
    OFPFF_CHECK_OVERLAP = 1 <<1，/ *首先检查重叠条目。* /
    OFPFF_RESET_COUNTS = 1 <<2，/ *重置流包和字节数。* /
    OFPFF_NO_PKT_COUNTS = 1 <<3，/ *不要跟踪数据包计数。* /
```



```
    OFPFF_NO_BYT_COUNTS = 1 <<4，/ *不要跟踪字节数。* /
};

/ *流设置和拆除（控制器->数据路径）。* /
struct ofp_flow_mod {
    struct ofp_header标头;
    uint64_t cookie;                / *不透明的控制器发出的标识符。* /
    uint64_t cookie_mask;           / *用于限制cookie位的掩码
                                       该命令必须匹配
                                       OFPFC_MODIFY *或OFPFC_DELETE *。一个值
                                       0表示没有限制。* /
    uint8_t table_id;
```

```
                                    / *对流OFPFC_DELETE *命令，为OFPTT_ALL
                                       也可以用来删除匹配项
                                       所有表的流量。* /
    uint8_t命令;                    / * OFPFC_ *之一。* /
    uint16_t idle_timeout;          / *丢弃前的空闲时间（秒）。* /
    uint16_t hard_timeout;          / *丢弃前的最长时间（秒）。* /
    uint16_t优先级;                 / *流条目的优先级。* /
    uint32_t buffer_id;             / *要应用于的缓冲数据包，或
                                       OFP_NO_BUFFER。
                                       对于OFPFC_DELETE *没有意义。* /
    uint32_t out_port;              / *对于OFPFC_DELETE *命令，要求
                                       匹配条目以将此作为
                                       输出端口。 值为OFPP_ANY
                                       表示没有限制。* /
    uint32_t out_group;             / *对于OFPFC_DELETE *命令，要求
                                       匹配条目以将此作为
                                       输出组。 值为OFPG_ANY
                                       表示没有限制。* /
    uint16_t标志;                   / * OFPFF_ *标志的位图。* /
    uint16_t重要性;                 / *驱逐优先级（可选）。* /
    struct ofp_match match;         / *要匹配的字段。大小可变。* /
    / *变量大小和填充匹配始终带有指令。* /
    // p_instruction_header指令的结构[0];
                                / *指令集-0或更多。长度
                                   的指令集是从
                                   标头中的长度字段。* /
};
OFP_ASSERT（sizeof（p_flow_mod的结构）== 56）;

/ *组编号。群组最多可以使用OFPG_MAX个。* /
ofp_group的枚举{
    / *最后可用的组号。* /
    OFPG_MAX    = 0xffffff00，

    / *伪造的群组。* /
    OFPG_ALL    = 0xfffffffc，  / *代表所有要删除的组
                                   命令。* /
    OFPG_ANY    = 0xffffffff    / *特殊通配符：未指定组。* /
};

/ *组命令* /
ofp_group_mod_command枚举{
    OFPGC_ADD = 0，         / *新组。* /
    OFPGC_MODIFY = 1 / *修改所有匹配的组。* /
    OFPGC_DELETE = 2 / *删除所有匹配的组。* /
    OFPGC_INSERT_BUCKET = 3，/ *将操作存储桶插入到已经可用的存储桶中
                                匹配组中的操作段列表* /
    / * OFPGC _ ??? = 4，??保留供将来使用。* /
    OFPGC_REMOVE_BUCKET = 5，/ *删除所有操作存储桶或任何特定操作
                               匹配组中的存储桶* /
};

/ *组存储桶属性类型。          * /
of_p_group_bucket_prop_type的枚举{
    OFPGBPT_WEIGHT          = 0，/ *仅选择组。* /
    OFPGBPT_WATCH_PORT    = 1   / *仅快速故障转移组。* /
    OFPGBPT_WATCH_GROUP= 2    / *仅快速故障转移组。* /
    OFPGBPT_EXPERIMENTER = 0xFFFF / *实验者定义。* /
};

/ *所有组存储桶属性的通用标头。* /
struct ofp_group_bucket_prop_header {
    uint16_t            类型;    / * OFPGBPT_ *之一。* /
    uint16_t            长度;    / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_group_bucket_prop_header的结构== 4）;

/ *组铲斗重量属性，仅适用于选定组。* /
struct ofp_group_bucket_prop_weight {
    uint16_t            类型;    / * OFPGBPT_WEIGHT。* /
    uint16_t            长度;    / * 8. * /
    uint16_t            重量;    / *铲斗的相对重量。* /
    uint8_t             垫[2];   / *填充到64位。* /
};
OFP_ASSERT（sizeof（p_group_bucket_prop_weight的结构）== 8）;

/ *组存储区监视端口或监视组属性，用于快速故障转移组
 *只要。* /
```


第220章 OpenFlow交换机规格 版本1.5.1

```
struct ofp_group_bucket_prop_watch {
    uint16_t            类型;    / * OFPGBPT_WATCH_PORT或OFPGBPT_WATCH_GROUP。* /
    uint16_t            长度;    / * 8. * /
    uint32_t            端口;    / *端口或组。                * /
};
OFP_ASSERT（sizeof（p_group_bucket_prop_watch的结构）== 8）;

/ *实验者组存储桶属性* /
p_group_bucket_prop_experimenter的结构{
    uint16_t            类型;          / * OFPGBPT_EXPERIMENTER。* /
    uint16_t            长度;          / *此属性的长度（以字节为单位）。* /
    uint32_t            实验者         / *采用相同的实验者ID
                                          形式如结构
                                          ofp_experimenter_header。* /
    uint32_t            exp_type;      / *实验者定义。* /
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t            实验者数据[0];
};
OFP_ASSERT（sizeof（p_group_bucket_prop_experimenter的结构== 12）;

/ *供组使用的存储桶。* /
struct ofp_bucket {
    uint16_t len;                   / *存储桶的长度（以字节为单位），包括
                                      这个标题和任何填充它
                                      64位对齐。* /
    uint16_t action_array_len;      / *所有动作的长度（以字节为单位）。* /
    uint32_t bucket_id;             / *用于标识存储桶的存储桶ID * /
    /* 其次是：
     *    -包含数组的“ action_array_len”字节
     *       p_action_ *结构。
     *    -零或更多字节的组存储桶属性以填写
     *       header.length中的总长度。* /
    struct ofp_action_header actions [0]; / *操作数组的长度为
                                             action_array_len字节。* /
    // p_group_bucket_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_bucket的结构）== 8）;

/ *存储桶ID可以是0到OFPG_BUCKET_MAX之间的任何值* /
of_p_group_bucket的枚举{
    OFPG_BUCKET_MAX = 0xffffff00，/ *最后可用的存储区ID。* /
    OFPG_BUCKET_FIRST = 0xfffffffd，/ *操作列表中的第一个存储区ID
                                      一桶水。这适用
                                      对于OFPGC_INSERT_BUCKET和
                                      OFPGC_REMOVE_BUCKET命令。* /
```

```
    OFPG_BUCKET_LAST = 0xfffffffe，/ *操作列表中的最后一个存储桶。
                                     对于OFPGC_INSERT_BUCKET和
                                     OFPGC_REMOVE_BUCKET命令。* /
    OFPG_BUCKET_ALL = 0xffffffff    / *一组中的所有操作分区，
                                     这适用于
                                     仅OFPGC_REMOVE_BUCKET命令。* /
};

/ *组属性类型。              * /
of_p_group_prop_type的枚举{
    OFPGPT_EXPERIMENTER = 0xFFFF / *实验者定义。* /
};

/ *所有组属性的通用头。* /
struct ofp_group_prop_header {
    uint16_t            类型;    / * OFPGPT_ *之一。* /
    uint16_t            长度;    / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_group_prop_header的结构）== 4）；

/ *实验者组属性* /
struct ofp_group_prop_experimenter {
    uint16_t            类型;          / * OFPGPT_EXPERIMENTER。* /
    uint16_t            长度;          / *此属性的长度（以字节为单位）。* /
    uint32_t            实验者         / *采用相同的实验者ID
                                         形式如结构
                                         ofp_experimenter_header。* /
    uint32_t            exp_type;      / *实验者定义。* /
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t            实验者数据[0];
};
OFP_ASSERT（sizeof（p_group_prop_experimenter的结构）== 12）；

/ *组设置和拆除（控制器->数据路径）。* /
struct ofp_group_mod {
    struct ofp_header标头；
    uint16_t命令;                  / * OFPGC_ *之一。* /
    uint8_t类型;                   / * OFPGT_ *之一。* /
```



```
    uint8_t pad;                    / *填充到64位。* /
    uint32_t group_id;              / *组标识符。* /
    uint16_t bucket_array_len;      / *操作存储区数据的长度。* /
    uint8_t pad2 [2];               / *填充到64位。* /
    uint32_t command_bucket_id;     / *桶ID用作
                                      OFPGC_INSERT_BUCKET和OFPGC_REMOVE_BUCKET
                                      命令执行。* /
    /* 其次是：
     *    -确切的“ bucket_array_len”字节包含一个数组
     *       p_bucket结构。
     *    -零个或多个字节的组属性可填充整体
     *       标题中的长度。* /
    p_bucket桶的结构[0]; / *存储桶数组的长度为
                                      bucket_array_len字节。* /
    // p_group_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_group_mod的结构）== 24）；

/ *组类型。      [128，255]范围内的值保留用于实验
 * 采用。* /
of_p_group_type的枚举{
    OFPGT_ALL = 0，/ *全部（多播/广播）组。          * /
    OFPGT_SELECT，/ *选择组。* /
    OFPGT_INDIRECT = 2，/ *间接组。* /
    OFPGT_FF    = 3，/ *快速故障转移组。* /
};

/ *特殊的缓冲区ID，表示“无缓冲区” * /
#define OFP_NO_BUFFER 0xffffffff

/ *发送数据包（控制器->数据路径）。* /
struct ofp_packet_out {
    struct ofp_header标头；
    uint32_t buffer_id;             / *由数据路径分配的ID（OFP_NO_BUFFER
                                      如果没有）。* /
    uint16_t actions_len;           / *操作数组的大小，以字节为单位。* /
    uint8_t pad [2];                / *对齐64位。* /
    struct ofp_match match;         / *数据包管道字段。大小可变。* /
    / *变量大小和填充匹配后面是操作列表。* /
    / * struct ofp_action_header actions [0]; * // *动作列表-0或更多。* /
    / *可变大小操作列表后面有数据包数据（可选）。
     *仅当buffer_id == -1时，此数据才有意义。* /
    / * uint8_t数据[0]; * /          / *数据包数据。长度推断
                                      从标题的长度字段开始。* /
};
OFP_ASSERT（sizeof（p_packet_out的结构）== 24）；

/ *为什么将此数据包发送到控制器？* /
of_p_packet_in_reason枚举{
    OFPR_TABLE_MISS = 0，/ *没有匹配的流（表缺失流条目）。* /
    OFPR_APPLY_ACTION = 1，/ *在apply-actions中输出到控制器。* /
    OFPR_INVALID_TTL = 2，/ *数据包具有无效的TTL * /
    OFPR_ACTION_SET = 3，/ *输出到动作集中的控制器。* /
    OFPR_GROUP = 4，/ *输出到组存储区中的控制器。* /
    OFPR_PACKET_OUT = 5 / *输出到包中的控制器。* /
};

/ *在端口（数据路径->控制器）上接收到的数据包。* /
struct ofp_packet_in {
    struct ofp_header标头；
    uint32_t buffer_id;         / *由数据路径分配的ID。* /
    uint16_t total_len;         / *整个帧的长度。* /
    uint8_t原因;                / *原因包被发送（OFPR_ *之一）* /
    uint8_t table_id;           / *查找的表的ID * /
    uint64_t cookie;            / *查找的流条目的Cookie。* /
    struct ofp_match match; / *数据包元数据。大小可变。* /
    / *变量大小和填充匹配始终后跟：
     *    -恰好2个全零填充字节，然后
     *    -以太网帧，其长度是根据header.length推断的。
     *以太网帧之前的填充字节可确保IP
     *以太网报头后面的报头（如果有）是32位对齐的。
     * /
    // uint8_t pad [2];         / *对齐64位+ 16位* /
    // uint8_t data [0];        / *以太网帧* /
};
OFP_ASSERT（sizeof（p_packet_in的结构）== 32）；

/ *为什么要删除此流程？* /
of_p_flow_removed_reason枚举{
    OFPRR_IDLE_TIMEOUT = 0，/ *流空闲时间超过了idle_timeout。* /
    OFPRR_HARD_TIMEOUT = 1，/ *时间超出了hard_timeout。* /
    OFPRR_DELETE = 2            / *由DELETE flow mod驱逐。* /
    OFPRR_GROUP_DELETE = 3，/ *组已删除。* /
    OFPRR_METER_DELETE = 4，/ *仪表已剔下。* /
    OFPRR_EVICTION = 5          / *将驱逐切换为自由资源。* /
```

```
};
/ *流被删除（数据路径->控制器）。* /
struct ofp_flow_removed {
    struct ofp_header标头；
```


第222页


```
    uint8_t table_id;              / *表的ID * /
    uint8_t原因；                  / * OFPRR_ *之一。* /
    uint16_t优先级；               / *流条目的优先级。* /
    uint16_t idle_timeout;     / *原始流模块的空闲超时。* /
    uint16_t hard_timeout;     / *从原始流模块硬超时。* /
    uint64_t cookie;               / *不透明的控制器发出的标识符。* /
    struct ofp_match match;  / *字段说明。大小可变。* /
    // struct ofp_stats统计信息；  / *统计信息列表。大小可变。* /
};
OFP_ASSERT（sizeof（p_flow_removed的结构）== 32）；

/ *仪表编号。流量计最多可以使用OFPM_MAX个数字。* /
ofp_meter {
    / *最后可用的仪表。* /
    OFPM_MAX    = 0xffff0000，

    / *虚拟仪表。* /
    OFPM_SLOWPATH = 0xfffffffd，  / *计量器用于慢速数据路径。* /
    OFPM_CONTROLLER = 0xfffffffe，  / *用于连接控制器的仪表。* /
    OFPM_ALL    = 0xffffffff，    / *代表所有用于统计请求的仪表
                                      命令。* /
};

/ *表带类型* /
of_p_meter_band_type的枚举{
    OFPMBT_DROP     = 1      / *丢弃数据包。* /
    OFPMBT_DSCP_REMARK = 2     / *在IP标头中标记DSCP。* /
    OFPMBT_EXPERIMENTER = 0xFF / *实验仪表带。* /
};

/ *所有仪表波段的通用标头* /
struct ofp_meter_band_header {
    uint16_t           类型；    / * OFPMBT_ *之一。* /
    uint16_t           len;       / *此频段的长度（以字节为单位）。* /
    uint32_t           率；       / *此频段的价格。* /
    uint32_t           突发大小; / *突发大小。* /
};
OFP_ASSERT（sizeof（p_meter_band_header的结构）== 12）；

/ * OFPMBT_DROP频段-丢包* /
struct ofp_meter_band_drop {
    uint16_t           类型；    / * OFPMBT_DROP。* /
    uint16_t           len;       / *长度为16。* /
    uint32_t           率；       / *丢包率。* /
    uint32_t           突发大小; / *突发大小。* /
    uint8_t            垫[4];
};
OFP_ASSERT（sizeof（p_meter_band_drop的结构）== 16）；

/ * OFPMBT_DSCP_REMARK频段-在IP标头中标记DSCP * /
struct ofp_meter_band_dscp_remark {
    uint16_t           类型；    / * OFPMBT_DSCP_REMARK。* /
    uint16_t           len;       / *长度为16。* /
    uint32_t           率；       / *标记数据包的速率。* /
    uint32_t           突发大小; / *突发大小。* /
    uint8_t            prec_level; / *要添加的丢弃优先级的数量。* /
    uint8_t            垫[3];
};
OFP_ASSERT（sizeof（struct ofp_meter_band_dscp_remark）== 16）；

/ * OFPMBT_EXPERIMENTER波段-实验者类型。
  *乐队的其余部分由实验者定义。* /
struct ofp_meter_band_experimenter {
    uint16_t           类型；    / * OFPMBT_EXPERIMENTER。* /
    uint16_t           len;       / *此频段的长度（以字节为单位）。* /
    uint32_t           率；       / *此频段的价格。* /
    uint32_t           突发大小;    / *突发大小。* /
    uint32_t           实验者; / *实验者ID。* /
};
OFP_ASSERT（sizeof（p_meter_band_experimenter的结构）== 16）；

/ *仪表命令* /
ofp_meter_mod_command枚举{
    OFPMC_ADD    = 0       / *新仪表。* /
    OFPMC_MODIFY = 1 / *修改指定的仪表。* /
    OFPMC_DELETE = 2 / *删除指定的仪表。* /
};

/ *仪表配置标志* /
of_p_meter_flags枚举{
    OFPMF_KBPS   = 1 << 0，/ *速率值，以kb / s（每秒千比特）为单位。* /
    OFPMF_PKTPS  = 1 << 1，/ *速率值（以包/秒为单位）。* /
    OFPMF_BURST  = 1 << 2，/ *执行连拍大小。* /
    OFPMF_STATS  = 1 << 3，/ *收集统计信息。* /
};

/ *仪表配置。OFPT_METER_MOD。* /
struct ofp_meter_mod {
    struct ofp_header标头；
    uint16_t           命令;              / * OFPMC_ *之一。* /
```


第223页


```
    uint16_t           标志;              / * OFPMF_ *标志的位图。* /
    uint32_t           meter_id;          / *仪表实例。* /
    struct ofp_meter_band_header bands [0]; / *波段列表长度为
                                               从长度字段推断
                                               在标题中。* /
};
OFP_ASSERT（sizeof（p_meter_mod的结构）== 16）；
```

```
/* ## ---------- ## */
/* ## OpenFlow错误消息 ## */
/* ## ---------- ## */

/* ofp_error_message中“类型”的值。 这些值是不变的：它们
 *在协议的未来版本中不会更改（尽管可能会增加新值
 *添加）。*/
of_p_error_type的枚举{
    OFPET_HELLO_FAILED = 0，/* Hello协议失败。*/
    OFPET_BAD_REQUEST = 1，/*无法理解请求。*/
    OFPET_BAD_ACTION = 2，/*动作描述错误。*/
    OFPET_BAD_INSTRUCTION = 3，/*指令列表中的错误。*/
    OFPET_BAD_MATCH = 4，/*匹配错误。*/
    OFPET_FLOW_MOD_FAILED = 5，/*修改流条目时出现问题。*/
    OFPET_GROUP_MOD_FAILED = 6，/*修改组条目时出现问题。*/
    OFPET_PORT_MOD_FAILED = 7，/*端口mod请求失败。*/
    OFPET_TABLE_MOD_FAILED = 8，/*表mod请求失败。*/
    OFPET_QUEUE_OP_FAILED = 9，/*队列操作失败。*/
    OFPET_SWITCH_CONFIG_FAILED = 10，/*切换配置请求失败。*/
    OFPET_ROLE_REQUEST_FAILED = 11，/*控制器角色请求失败。*/
    OFPET_METER_MOD_FAILED = 12，/*仪表错误。*/
    OFPET_TABLE_FEATURES_FAILED = 13，/*设置表格功能失败。*/
    OFPET_BAD_PROPERTY = 14，/*某些属性无效。*/
    OFPET_ASYNC_CONFIG_FAILED = 15，/*异步配置请求失败。*/
    OFPET_FLOW_MONITOR_FAILED = 16，/*泛洪量监控器失败。*/
    OFPET_BUNDLE_FAILED = 17，/*捆绑操作失败。*/
    OFPET_EXPERIMENTER = 0xffff/*实验者错误消息。*/
};

/* OFPET_HELLO_FAILED的p_error_msg'代码'值。“数据”包含一个
 *可能提供故障详细信息的ASCII文本字符串。*/
of_p_hello_failed_code {
    OFPHFC_INCOMPATIBLE = 0，/*没有兼容的版本。*/
    OFPHFC_EPERM = 1，/*权限错误。*/
};

/* OFPET_BAD_REQUEST的p_error_msg'代码'值。“数据”至少包含
 *失败请求的前64个字节。*/
of_p_bad_request_code的枚举{
    OFPBRC_BAD_VERSION = 0，/* ofp_header.version不支持。*/
    OFPBRC_BAD_TYPE = 1，/*不支持/* ofp_header.type。*/
    OFPBRC_BAD_MULTIPART = 2，/* ofp_multipart_request.type不是
                                支持的。*/
    OFPBRC_BAD_EXPERIMENTER = 3，/*不支持实验者ID
                                *（在ofp_experimenter_header中或
                                * ofp_multipart_request或
                                * ofp_multipart_reply）。*/
    OFPBRC_BAD_EXP_TYPE = 4，/*不支持实验者类型。*/
    OFPBRC_EPERM = 5，/*权限错误。*/
    OFPBRC_BAD_LEN = 6，/*类型的请求长度错误。*/
    OFPBRC_BUFFER_EMPTY = 7，/*指定的缓冲区已被使用。*/
    OFPBRC_BUFFER_UNKNOWN = 8，/*指定的缓冲区不存在。*/
    OFPBRC_BAD_TABLE_ID = 9，/*指定的表ID无效或无效
                                *存在。*/
    OFPBRC_IS_SLAVE = 10，/*由于控制器是从属设备而被拒绝。*/
    OFPBRC_BAD_PORT = 11，/*无效的端口或缺少的端口。*/
    OFPBRC_BAD_PACKET = 12，/*无效的数据包输出。*/
    OFPBRC_MULTIPART_BUFFER_OVERFLOW = 13，/* multipart_request
                                溢出分配的缓冲区。*/
    OFPBRC_MULTIPART_REQUEST_TIMEOUT = 14，/*多部分超时
                                请求。*/
    OFPBRC_MULTIPART_REPLY_TIMEOUT = 15，/*分段答复期间超时。*/
    OFPBRC_MULTIPART_BAD_SCHED = 16，/*交换机收到OFPMP_BUNDLE_FEATURES
                                请求，但无法更新
                                计划公差。*/
    OFPBRC_PIPELINE_FIELDS_ONLY = 17，/*匹配字段必须仅包含
                                管道字段。*/
    OFPBRC_UNKNOWN = 18，/*未指定错误。*/
};

/* OFPET_BAD_ACTION的p_error_msg'代码'值。“数据”至少包含
 *失败请求的前64个字节。*/
of_p_bad_action_code的枚举{
    OFPBAC_BAD_TYPE = 0，/*未知或不受支持的操作类型。*/
    OFPBAC_BAD_LEN = 1，/*动作中的长度问题。*/
    OFPBAC_BAD_EXPERIMENTER = 2，/*指定了未知的实验者ID。*/
    OFPBAC_BAD_EXP_TYPE = 3，/*实验者ID的未知操作。*/
    OFPBAC_BAD_OUT_PORT = 4，/*验证输出端口时出现问题。*/
    OFPBAC_BAD_ARGUMENT = 5，/*错误动作参数。*/
    OFPBAC_EPERM = 6，/*权限错误。*/
    OFPBAC_TOO_MANY = 7，/*无法处理这么多动作。*/
    OFPBAC_BAD_QUEUE = 8，/*验证输出队列时出现问题。*/
```



```
    OFPBAC_BAD_OUT_GROUP = 9，/*组操作中的组ID无效。*/
    OFPBAC_MATCH_INCONSISTENT = 10，/*无法为该比赛申请动作，
                                或缺少设置字段的先决条件。*/
    OFPBAC_UNSUPPORTED_ORDER = 11，/*不支持操作顺序
                                Apply-Actions指令中的动作列表*/
    OFPBAC_BAD_TAG = 12，/*操作使用不受支持的
                                标签/封套。*/
    OFPBAC_BAD_SET_TYPE = 13，/* SET_FIELD操作中不受支持的类型。*/
    OFPBAC_BAD_SET_LEN = 14，/* SET_FIELD动作中的长度问题。*/
    OFPBAC_BAD_SET_ARGUMENT = 15，/* SET_FIELD操作中的错误参数。*/
    OFPBAC_BAD_SET_MASK = 16，/* SET_FIELD操作中的错误掩码。*/
    OFPBAC_BAD_METER = 17，/*仪表操作中无效的仪表ID。*/
};

/* OFPET_BAD_INSTRUCTION的p_error_msg'代码'值'数据”包含在
 *至少失败请求的前64个字节。*/
of_p_bad_instruction_code的枚举{
    OFPBIC_UNKNOWN_INST = 0，/*未知指令。*/
    OFPBIC_UNSUP_INST = 1，/*开关或表不支持
                                指令。*/
    OFPBIC_BAD_TABLE_ID = 2，/*指定了无效的表ID。*/
    OFPBIC_UNSUP_METADATA = 3，/*数据路径不支持元数据值。*/
    OFPBIC_UNSUP_METADATA_MASK = 4，/*不支持的元数据掩码值
                                数据路径。*/
    OFPBIC_BAD_EXPERIMENTER = 5，/*指定了未知的实验者ID。*/
    OFPBIC_BAD_EXP_TYPE = 6，/*实验者ID的未知说明。*/
    OFPBIC_BAD_LEN = 7，/*指令中的长度问题。*/
    OFPBIC_EPERM = 8，/*权限错误。*/
    OFPBIC_DUP_INST = 9，/*重复指令。*/
};

/* OFPET_BAD_MATCH的p_error_msg'代码'值。“数据”至少包含
 *失败请求的前64个字节。*/
of_p_bad_match_code {
    OFPBMC_BAD_TYPE = 0，/*不支持的匹配类型，由
                                比赛*/
    OFPBMC_BAD_LEN = 1，/*匹配中的长度问题。*/
    OFPBMC_BAD_TAG = 2，/*匹配使用不受支持的标签/封套。*/
    OFPBMC_BAD_DL_ADDR_MASK = 3，/*不支持的数据链路地址掩码-开关
                                不支持任意数据链接
                                地址掩码。*/
    OFPBMC_BAD_NW_ADDR_MASK = 4，/*不支持的网络地址掩码-开关
                                不支持任意网络
                                地址掩码。*/
    OFPBMC_BAD_WILDCARDS = 5，/*不支持的字段组合被屏蔽
```

```
    OFPBMC_BAD_FIELD = 6    /* 或在此表中不支持的字段类型。 */
    OFPBMC_BAD_VALUE = 7    /* 匹配字段中不支持的值。 */
    OFPBMC_BAD_MASK = 8     /* 匹配中指定了不受支持的掩码。 */
    OFPBMC_BAD_PREREQ = 9   /* 不满足先决条件。 */
    OFPBMC_DUP_FIELD = 10,  /* 字段类型重复。 */
    OFPBMC_EPERM = 11,      /* 权限错误。 */
};

/* OFPET_FLOW_MOD_FAILED的p_error_msg'代码'值数据”包含
 *至少失败请求的前64个字节。 */
of_p_flow_mod_failed_code {
    OFPFMFC_UNKNOWN = 0,    /* 未指定的错误。 */
    OFPFMFC_TABLE_FULL,     /* 由于表已满，因此未添加流。 */
    OFPFMFC_BAD_TABLE_ID,   /* 表不存在 */
    OFPFMFC_OVERLAP,        /* 尝试添加重叠流
                               CHECK_OVERLAP标志设置。 */
    OFPFMFC_EPERM,          /* 权限错误。 */
    OFPFMFC_BAD_TIMEOUT,    /* 由于不支持而未添加流
                               空闲/硬超时。 */
    OFPFMFC_BAD_COMMAND,    /* 不支持或未知的命令。 */
    OFPFMFC_BAD_FLAGS,      /* 不支持或未知的标志。 */
    OFPFMFC_CANT_SYNC,      /* 表同步中的问题。 */
    OFPFMFC_BAD_PRIORITY,   /* 不支持的优先级值。 */
    OFPFMFC_IS_SYNC,        /* 同步流条目是只读的。 */
};

/* OFPET_GROUP_MOD_FAILED的p_error_msg'代码'值数据”包含
 *至少失败请求的前64个字节。 */
of_p_group_mod_failed_code的枚举{
    OFPGMFC_GROUP_EXISTS = 0,   /* 未添加组，因为添加了组
                                   试图更换
                                   已经存在的组。 */
    OFPGMFC_INVALID_GROUP,      /* 未添加组，因为组
                                   指定的无效。 */
    OFPGMFC_WEIGHT_UNSUPPORTED, /* 不支持不相等的负载
                                   与选定的组共享。 */
    OFPGMFC_OUT_OF_GROUPS,      /* 组表已满。 */
    OFPGMFC_OUT_OF_BUCKETS,     /* 最大动作桶数
                                   超过了一个组。 */
    OFPGMFC_CHAINING_UNSUPPORTED,/* 不支持
                                   转发给团体。 */
    OFPGMFC_WATCH_UNSUPPORTED,  /* 该组无法观看watch_port
                                   或指定的watch_group。 */
    OFPGMFC_LOOP = 7    /* 组条目将导致循环。 */
```



```
    OFPGMFC_UNKNOWN_GROUP,  /* 组未修改，因为组
                               修改尝试修改
                               不存在的组。 */
    OFPGMFC_CHAINED_GROUP,  /* 组未删除，因为另一个
                               组正在转发给它。 */
    OFPGMFC_BAD_TYPE = 10,  /* 不支持或未知的组类型。 */
    OFPGMFC_BAD_COMMAND,    /* 不支持或未知的命令。 */
    OFPGMFC_BAD_BUCKET,     /* 存储桶错误。 */
    OFPGMFC_BAD_WATCH = 13, /* 监视端口/组中的错误。 */
    OFPGMFC_EPERM,          /* 权限错误。 */
    OFPGMFC_UNKNOWN_BUCKET, /* 在
                               插入桶或取出桶
                               命令。 */
    OFPGMFC_BUCKET_EXISTS,  /* 无法插入存储桶，因为存储桶
                               该存储桶ID已经存在。 */
};

/* OFPET_PORT_MOD_FAILED的p_error_msg'代码'值'数据”包含
 *至少失败请求的前64个字节。 */
of_p_port_mod_failed_code的枚举{
    OFPPMFC_BAD_PORT,       /* 指定的端口号不存在。 */
    OFPPMFC_BAD_HW_ADDR,    /* 指定的硬件地址不正确
                               *匹配端口号。 */
    OFPPMFC_BAD_CONFIG,     /* 指定的配置无效。 */
    OFPPMFC_BAD_ADVERTISE,  /* 指定的广告无效。 */
    OFPPMFC_EPERM = 4       /* 权限错误。 */
};

/* OFPET_TABLE_MOD_FAILED的p_error_msg'代码'值数据”包含
 *至少失败请求的前64个字节。 */
of_p_table_mod_failed_code的枚举{
    OFPTMFC_BAD_TABLE = 10, /* 指定的表不存在。 */
    OFPTMFC_BAD_CONFIG,     /* 指定的配置无效。 */
    OFPTMFC_EPERM,          /* 权限错误。 */
};

/* OFPET_QUEUE_OP_FAILED的p_error msg'代码'值。“数据”包含
 *至少失败请求的前64个字节* /
of_p_queue_op_failed_code {
    OFPQOFC_BAD_PORT = 1,   /* 无效的端口（或端口不存在）。 */
    OFPQOFC_BAD_QUEUE,      /* 队列不存在。 */
    OFPQOFC_EPERM,          /* 权限错误。 */
};

/* OFPET_SWITCH_CONFIG_FAILED的p_error_msg'代码'值。“数据”包含
 *至少失败请求的前64个字节。 */
p_switch_config_failed_code的枚举{
    OFPSCFC_BAD_FLAGS = 0,  /* 指定的标志无效。 */
    OFPSCFC_BAD_LEN,        /* 指定的错过发送len无效。 */
    OFPSCFC_EPERM = 2       /* 权限错误。 */
};

/* OFPET_ROLE_REQUEST_FAILED的p_error_msg'代码'值。“数据”包含
 *至少失败请求的前64个字节。 */
of_p_role_request_failed_code的枚举{
    OFPRRFC_STALE = 0,      /* 过时消息：旧generation_id。 */
    OFPRRFC_UNSUP,          /* 不支持控制器角色更改。 */
    OFPRRFC_BAD_ROLE,       /* 无效的角色。 */
    OFPRRFC_ID_UNSUP,       /* 开关不支持更改ID。 */
    OFPRRFC_ID_IN_USE,      /* 请求的ID正在使用。 */
};

/* OFPET_METER_MOD_FAILED的p_error_msg'代码'值数据”包含
 *至少失败请求的前64个字节。 */
of_p_meter_mod_failed_code {
    OFPMMFC_UNKNOWN = 0,    /* 未指定的错误。 */
    OFPMMFC_METER_EXISTS,   /* 未添加仪表，因为仪表已添加
                               *试图更换现有的仪表。 */
    OFPMMFC_INVALID_METER,  /* 未添加仪表，因为指定了仪表
                               * 是无效的，
                               *或电表动作中的电表无效。 */
    OFPMMFC_UNKNOWN_METER,  /* 仪表未修改，因为仪表已修改
                               *尝试修改不存在的仪表，
                               *或电表动作不良。 */
    OFPMMFC_BAD_COMMAND,    /* 不支持或未知的命令。 */
    OFPMMFC_BAD_FLAGS,      /* 不支持标志配置。 */
    OFPMMFC_BAD_RATE,       /* 不支持费率。 */
    OFPMMFC_BAD_BURST,      /* 不支持连拍大小。 */
    OFPMMFC_BAD_BAND,       /* 频段不受支持。 */
    OFPMMFC_BAD_BAND_VALUE, /* 不支持频带值。 */
    OFPMMFC_OUT_OF_METERS,  /* 没有更多可用的仪表。 */
    OFPMMFC_OUT_OF_BANDS
```

```
};                                  = 11，/*已经达到属性数。*/

/* OFPET_TABLE_FEATURES_FAILED的p_error_msg'代码'值。“数据”包含
 *至少失败请求的前64个字节。*/
of_p_table_features_failed_code的枚举{
    OFPTFFC_BAD_TABLE       = 0，/*指定的表不存在。*/
    OFPTFFC_BAD_METADATA    = 1，/*无效的元数据掩码。*/
```



```
    OFPTFFC_EPERM = 5           /*权限错误。*/
    OFPTFFC_BAD_CAPA = 6        /*无效的功能字段。*/
    OFPTFFC_BAD_MAX_ENT = 7     /*无效的max_entries字段。*/
    OFPTFFC_BAD_FEATURES = 8    /*无效的功能字段。*/
    OFPTFFC_BAD_COMMAND = 9     /*无效的命令。*/
    OFPTFFC_TOO_MANY = 10，     /*无法处理这么多的流表。*/
};

/* OFPET_BAD_PROPERTY的p_error_msg'代码'值。“数据”至少包含
 *失败请求的前64个字节。*/
of_p_bad_property_code的枚举{
    OFPBPC_BAD_TYPE     = 0，/*未知或不受支持的属性类型。*/
    OFPBPC_BAD_LEN      = 1  /*属性的长度问题。*/
    OFPBPC_BAD_VALUE = 2     /*不支持的属性值。*/
    OFPBPC_TOO_MANY   = 3，/*无法处理这么多属性。*/
    OFPBPC_DUP_TYPE   = 4    /*属性类型已重复。*/
    OFPBPC_BAD_EXPERIMENTER = 5 /*指定了未知的实验者ID。*/
    OFPBPC_BAD_EXP_TYPE = 6  /*实验者ID的未知exp_type。*/
    OFPBPC_BAD_EXP_VALUE = 7 /*实验者ID的未知值。*/
    OFPBPC_EPERM        = 8  /*权限错误。*/
};

/* OFPET_ASYNC_CONFIG_FAILED的p_error_msg'代码'值。“数据”包含
 *至少失败请求的前64个字节。*/
p_async_config_failed_code的枚举{
    OFPACFC_INVALID = 0      /*一个掩码无效。*/
    OFPACFC_UNSUPPORTED = 1  /*不支持请求的配置。*/
    OFPACFC_EPERM = 2        /*权限错误。*/
};

/* OFPET_FLOW_MONITOR_FAILED的p_error_msg'代码'值。“数据”包含
 *至少失败请求的前64个字节。*/
of_p_flow_monitor_failed_code的枚举{
    OFPMOFC_UNKNOWN = 0，/*未指定的错误。*/
    OFPMOFC_MONITOR_EXISTS = 1，/*未添加监视器，因为添加了监视器
                                 *尝试更换现有的监视器。*/
    OFPMOFC_INVALID_MONITOR = 2，/*未添加监视器，因为指定了监视器
                                  * 是无效的。*/
    OFPMOFC_UNKNOWN_MONITOR = 3，/*监视器未修改，因为监视器
                                  修改试图修改不存在的内容
                                  监控。*/
    OFPMOFC_BAD_COMMAND = 4，/*不支持或未知的命令。*/
    OFPMOFC_BAD_FLAGS = 5，/*不支持标志配置。*/
    OFPMOFC_BAD_TABLE_ID = 6，/*指定的表不存在。*/
    OFPMOFC_BAD_OUT = 7      /*输出端口/组中的错误。*/
};

/* OFPET_BUNDLE_FAILED的p_error_msg'代码'值。“数据”包含
 *至少失败请求的前64个字节。*/
of_pundle_failed_code {
  OFPBFC_UNKNOWN = 0，/*未指定的错误。*/
  OFPBFC_EPERM    = 1  /*权限错误。*/
  OFPBFC_BAD_ID   = 2  /*捆绑包ID不存在。*/
  OFPBFC_BUNDLE_EXIST = 3 /*捆绑包ID已存在。*/
  OFPBFC_BUNDLE_CLOSED = 4 /*捆绑包ID已关闭。*/
  OFPBFC_OUT_OF_BUNDLES = 5 /*太多。*/
  OFPBFC_BAD_TYPE = 6  /*不支持或未知的消息控件类型。*/
  OFPBFC_BAD_FLAGS = 7 /*不支持，未知或不一致的标志。*/
  OFPBFC_MSG_BAD_LEN *包含的消息中的长度问题。*/
  OFPBFC_MSG_BAD_XID /*XID不一致或重复。*/
  OFPBFC_MSG_UNSUP 0，/*此捆绑包中不支持的消息。*/
  OFPBFC_MSG_CONFLICT /*在此不支持的消息组合
                       来。*/
  OFPBFC_MSG_TOO_MANY *无法捆绑处理这么多邮件。*/
  OFPBFC_MSG_FAILED 3，/*捆绑中的一条消息失败。*/
  OFPBFC_TIMEOUT = 14  /*捆绑包花费的时间太长。*/
  OFPBFC_BUNDLE_IN_PROGRESS 捆绑包正在锁定资源。*/
  OFPBFC_SCHED_NOT_SUPPORTED = 16，/*已收到计划的提交，并且
                                    不支持调度。*/
  OFPBFC_SCHED_FUTURE 计划的提交时间超出上限。*/
  OFPBFC_SCHED_PAST = 18 计划的提交时间超出下限。*/
};

/* OFPT_ERROR：错误消息。*/
struct ofp_error_msg {
    struct ofp_header标头；

    uint16_t类型;
    uint16_t代码;
    uint8_t data [0];           /*可变长度数据。       口译依据
                                  在类型和代码上。     没有填充。*/
};
OFP_ASSERT（sizeof（struct ofp_error_msg）== 12）；

/* OFPET_EXPERIMENTER：错误消息（数据路径->控制器）。*/
struct ofp_error_experimenter_msg {
    struct ofp_header标头；

    uint16_t类型;           /* OFPET_EXPERIMENTER。*/
    uint16_t exp_code;      /*实验者定义。*/
```



```
    uint32_t实验者;         /*实验者ID。*/
    uint8_t data [0];       /*可变长度数据。       口译依据
                              关于类型和实验者。       没有填充。*/
};
OFP_ASSERT（sizeof（p_error_experimenter_msg的结构）== 16）；
```

```
/* ## ------------------- ## */
/* ## OpenFlow多部分。## */
/* ## ------------------- ## */

of_p_multipart_type的枚举{
    /*此OpenFlow开关的说明。
     *请求正文为空。
     *回复正文为p_desc的结构。*/
    OFPMP_DESC = 0,

    /*各个流的描述和统计信息。
     *请求主体是p_flow_stats_request的结构。
     *回复主体是p_flow_desc结构的数组。*/
    OFPMP_FLOW_DESC = 1

    /*汇总流量统计信息。
     *请求主体是p_aggregate_stats_request的结构。
     *回复主体为p_aggregate_stats_reply的结构。*/
    OFPMP_AGGREGATE_STATS = 2

    /*流表统计信息。
     *请求正文为空。
     *回复正文是p_table_stats结构的数组。*/
    OFPMP_TABLE_STATS = 3,

    /*端口统计信息。
     *请求主体是p_port_multipart_request的结构。
     *回复主体是p_port_stats结构的数组。*/
    OFPMP_PORT_STATS = 4

    /*端口的队列统计信息
     *请求主体是p_queue_multipart_request的结构。
     *回复主体是p_queue_stats结构的数组*/
    OFPMP_QUEUE_STATS = 5

    /*组计数器统计信息。
     *请求主体是p_group_multipart_request的结构。
     *答复是p_group_stats结构的数组。*/
    OFPMP_GROUP_STATS = 6

    /* 团体简介。
     *请求主体是p_group_multipart_request的结构。
     *回复正文是p_group_desc结构的数组。*/
    OFPMP_GROUP_DESC = 7

    /*组功能。
     *请求正文为空。
     *回复正文是p_group_features的结构。*/
    OFPMP_GROUP_FEATURES = 8

    /*仪表统计信息。
     *请求主体是p_meter_multipart_request的结构。
     *回复主体是p_meter_stats结构的数组。*/
    OFPMP_METER_STATS = 9

    /*仪表配置。
     *请求主体是p_meter_multipart_request的结构。
     *回复正文是p_meter_desc结构的数组。*/
    OFPMP_METER_DESC = 10,

    /*仪表功能。
     *请求正文为空。
     *回复正文为p_meter_features的结构。*/
    OFPMP_METER_FEATURES = 11

    /*表功能。
     *请求正文为空或包含一个数组
     *包含控制器的p_table_features结构
     *所需的开关视图。如果开关无法
     *设置指定的视图，返回错误。
     *回复主体是p_table_features结构的数组。*/
    OFPMP_TABLE_FEATURES = 12

    /*端口说明。
     *请求主体是p_port_multipart_request的结构。
     *回复正文是struct ofp_port的数组。*/
    OFPMP_PORT_DESC = 13

    /*表说明。
     *请求正文为空。
     *回复主体是p_table_desc结构的数组。*/
    OFPMP_TABLE_DESC = 14

    /*队列描述。
```



```
     *请求主体是p_queue_multipart_request的结构。
     *回复正文是struct ofp_queue_desc的数组。*/
    OFPMP_QUEUE_DESC = 15

    /*流量监视器。回复可能是异步消息。
     *请求主体是p_flow_monitor_request结构的数组。
     *回复正文是p_flow_update_header结构的数组。*/
    OFPMP_FLOW_MONITOR = 16

    /*单个流统计信息（无描述）。
     *请求主体是p_flow_stats_request的结构。
     *回复主体是p_flow_stats结构的数组。*/
    OFPMP_FLOW_STATS = 17

    /*控制器状态。
     *请求正文为空。
     *回复主体是p_controller_status结构的数组。*/
    OFPMP_CONTROLLER_STATUS = 18,

    /*捆绑功能。
    *请求主体是ofp_bundle_features_request。
    *回复正文是p_bundle_features的结构。*/
    OFPMP_BUNDLE_FEATURES = 19,

    /*实验者扩展。
     *请求和回复正文以
     *结构为p_experimenter_multipart_header。
     *请求和回复主体是实验者定义的。*/
    OFPMP_EXPERIMENTER = 0xffff
};

/*与1.3.1的向后兼容性-避免破坏API。*/
#define ofp_multipart_types ofp_multipart_type

of_p_multipart_request_flags的枚举{
    OFPMPF_REQ_MORE = 0 /*还有更多要求。*/
};

struct ofp_multipart_request {
    struct ofp_header标头;
    uint16_t类型;                   /* OFPMP_ *常量之一。*/
```

```
    uint8_t pad[4];             /* OFPMPF_REQ_*标志。*/
    uint8_t body[0];            /*请求的正文。0个或更多字节。*/
};
OFP_ASSERT（sizeof（struct ofp_multipart_request）== 16）；

of_p_multipart_reply_flags的枚举{
    OFPMPF_REPLY_MORE = 1 << 0  /*有更多回覆。*/
};

struct ofp_multipart_reply {
    struct ofp_header标头；
    uint16_t类型；              /* OFPMP_*常量之一。*/
    uint16_t标志；              /* OFPMPF_REPLY_*标志。*/
    uint8_t pad[4];
    uint8_t body[0];            /*回复的正文。0个或更多字节。*/
};
OFP_ASSERT（sizeof（p_multipart_reply的结构）== 16）；

#定义DESC_STR_LEN   256
#define SERIAL_NUM_LEN 32
/*对OFPMP_DESC请求的回复正文。每个条目都是以NULL终止的
 * ASCII字符串。*/
struct ofp_desc {
    字符mfr_desc [DESC_STR_LEN];/*制造商说明。*/
    字符hw_desc [DESC_STR_LEN];/*硬件描述。*/
    char sw_desc [DESC_STR_LEN];/*软件说明。*/
    字符serial_num [SERIAL_NUM_LEN];/*序列号。*/
    字符dp_desc [DESC_STR_LEN];/*人类可读的描述
                                数据路径。*/
};
OFP_ASSERT（sizeof（struct ofp_desc）== 1056）；

/*类型为OFPMP_FLOW_DESC和OFPMP_FLOW_STATS的ofp_multipart_request的正文。*/
struct ofp_flow_stats_request {
    uint8_t table_id;           /*要读取的表的ID（从ofp_table_desc），
                                所有表的OFPTT_ALL。*/
    uint8_t pad [3];            /*对齐32位。*/
    uint32_t out_port;          /*需要匹配条目才能包含此条目
                                作为输出端口。 值为OFPP_ANY
                                表示没有限制。*/
    uint32_t out_group;         /*需要匹配条目才能包含此条目
                                作为输出组。 值为OFPG_ANY
                                表示没有限制。*/
    uint8_t pad2 [4];           /*对齐64位。*/
    uint64_t cookie;            /*需要匹配的条目才能包含此
                                Cookie值*/
    uint64_t cookie_mask;       /*用于限制cookie位的掩码
                                必须匹配。值为0表示
```



```
                                没有限制。*/
    struct ofp_match match;     /*要匹配的字段。大小可变。*/
};
OFP_ASSERT（sizeof（p_flow_stats_request的结构）== 40）；

/*对OFPMP_FLOW_DESC请求的回复正文。*/
struct ofp_flow_desc {
    uint16_t长度；              /*此项的长度。*/
    uint8_t pad2 [2];           /*对齐64位。*/
    uint8_t table_id;           /*表流的ID来自。*/
    uint8_t pad;
    uint16_t优先级；            /*条目的优先级。*/
    uint16_t idle_timeout;      /*到期前空闲的秒数。*/
    uint16_t hard_timeout;      /*到期前的秒数。*/
    uint16_t标志；              /* OFPFF_*标志的位图。*/
    uint16_t重要性；            /*驱逐优先级。*/
    uint64_t cookie;            /*不透明的控制器发出的标识符。*/
    struct ofp_match match;     /*字段说明。大小可变。*/
    // struct ofp_stats统计信息；/*统计信息列表。大小可变。*/
    // p_instruction_header指令的结构[0]；
                                /*指令集-0或更多。*/
};
OFP_ASSERT（sizeof（struct ofp_flow_desc）== 32）；

/*生成流量统计信息的原因。*/
of_p_flow_stats_reason枚举{
    OFPFSR_STATS_REQUEST = 0,   /*回复OFPMP_FLOW_STATS请求。*/
    OFPFSR_STAT_TRIGGER = 1     /*状态由OFPIT_STAT_TRIGGER生成。*/
};

/*对OFPMP_FLOW_STATS请求的回复正文
 *和正文为OFPIT_STAT_TRIGGER生成的状态。*/
struct ofp_flow_stats {
    uint16_t长度；              /*此项的长度。*/
    uint8_t pad2 [2];           /*对齐64位。*/
    uint8_t table_id;           /*表流的ID来自。*/
    uint8_t原因；               /* OFPFSR_*之一。*/
    uint16_t优先级；            /*条目的优先级。*/
    struct ofp_match match;     /*字段说明。大小可变。*/
    // struct ofp_stats统计信息；/*统计信息列表。大小可变。*/
};
OFP_ASSERT（sizeof（struct ofp_flow_stats）== 16）；

/*类型为OFPMP_AGGREGATE_STATS的ofp_multipart_request的主体。*/
struct ofp_aggregate_stats_request {
    uint8_t table_id;           /*要读取的表的ID（来自ofp_table_stats）
                                所有表的OFPTT_ALL。*/
    uint8_t pad [3];            /*对齐32位。*/
    uint32_t out_port;          /*需要匹配条目才能包含此条目
                                作为输出端口。 值为OFPP_ANY
                                表示没有限制。*/
    uint32_t out_group;         /*需要匹配条目才能包含此条目
                                作为输出组。 值为OFPG_ANY
                                表示没有限制。*/
    uint8_t pad2 [4];           /*对齐64位。*/
    uint64_t cookie;            /*需要匹配的条目才能包含此
                                Cookie值*/
    uint64_t cookie_mask;       /*用于限制cookie位的掩码
                                必须匹配。值为0表示
                                没有限制。*/
    struct ofp_match match;     /*要匹配的字段。大小可变。*/
};
OFP_ASSERT（sizeof（p_aggregate_stats_request的结构）== 40）；

/*对OFPMP_AGGREGATE_STATS请求的回复正文。*/
struct ofp_aggregate_stats_reply {
    struct ofp_stats统计信息；/*汇总统计信息列表。大小可变。*/
};
OFP_ASSERT（sizeof（struct ofp_aggregate_stats_reply）== 8）；

/*表要素属性类型。
 *低位清零表示常规流条目的属性。
 *低位设置指示表丢失流条目的属性。
 */
of_p_table_feature_prop_type的枚举{
    OFPTFPT_INSTRUCTIONS = 0    /*指令属性。*/
    OFPTFPT_INSTRUCTIONS_MISS/*有关表缺失的说明。*/
    OFPTFPT_NEXT_TABLES  = 2    /*下一表属性。*/
```

```
    OFPTFPT_NEXT_TABLES_MISS = 5,  /* 写表缺失其他表*/
    OFPTFPT_WRITE_ACTIONS = 6,     /* 写表缺失写操作。*/
    OFPTFPT_WRITE_ACTIONS_MISS = 7,/* 应用动作属性。*/
    OFPTFPT_APPLY_ACTIONS = 8,     /* 对表缺失应用操作。*/
    OFPTFPT_APPLY_ACTIONS_MISS = 9,/* 匹配属性。*/
    OFPTFPT_MATCH = 10,            /* 通配符属性。*/
    OFPTFPT_WILDCARDS = 12,        /* 写入Set-Field属性。*/
    OFPTFPT_WRITE_SETFIELD = 13,   /* 写表缺失的Set-Field。*/
    OFPTFPT_WRITE_SETFIELD_MISS = 14,/* 应用设置字段属性。*/
    OFPTFPT_APPLY_SETFIELD = 15,   /* 为表缺失应用Set-Field。*/
    OFPTFPT_APPLY_SETFIELD_MISS = 16,/* 表同步属性。*/
    OFPTFPT_TABLE_SYNC_FROM = 17,
```



```
    OFPTFPT_WRITE_COPYFIELD = 18,  /* 写入Copy-Field属性。*/
    OFPTFPT_WRITE_COPYFIELD_MISS = 19,/* 写表缺失的复制字段。*/
    OFPTFPT_APPLY_COPYFIELD = 20,  /* 应用复制字段属性。*/
    OFPTFPT_APPLY_COPYFIELD_MISS = 21,/* 为表缺失应用复制字段。*/
    OFPTFPT_PACKET_TYPES = 22,     /* 数据包类型属性。*/
    OFPTFPT_EXPERIMENTER = 0xFFFE, /* 实验者属性。*/
    OFPTFPT_EXPERIMENTER_MISS = 0xFFFF, /* 表丢失的实验器。*/
};

/* 所有表功能属性的公用头*/
struct ofp_table_feature_prop_header {
    uint16_t           类型;    /* OFPTFPT_ *之一。*/
    uint16_t           长度;    /* 此属性的长度（以字节为单位）。*/
};
OFP_ASSERT（sizeof（p_table_feature_prop_header的结构）== 4）;

/* 指令ID */
struct ofp_instruction_id {
    uint16_t类型;                 /* OFPIT_ *之一。*/
    uint16_t len;                 /* 长度为4或由实验人员定义。*/
    uint8_t exp_data [0];         /* 可选的实验者ID +数据。*/
};
OFP_ASSERT（sizeof（p_instruction_id的结构）== 4）;

/* 说明属性*/
struct ofp_table_feature_prop_instructions {
    uint16_t           类型;    /* OFPTFPT_INSTRUCTIONS之一，
                                   OFPTFPT_INSTRUCTIONS_MISS。*/
    uint16_t           长度;    /* 此属性的长度（以字节为单位）。*/
    /* 其次是：
     *    -恰好（长度-4）个字节，包含指令ID，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节*/
    p_instruction_id结构       struction_ids [0];      /* 指令列表*/
};
OFP_ASSERT（sizeof（p_table_feature_prop_instructions的结构）== 4）;

/* 下一个表和表从属性同步*/
struct ofp_table_feature_prop_tables {
    uint16_t           类型;    /* OFPTFPT_NEXT_TABLES之一，
                                   OFPTFPT_NEXT_TABLES_MISS，
                                   OFPTFPT_TABLE_SYNC_FROM。*/
    uint16_t           长度;    /* 此属性的长度（以字节为单位）。*/
    /* 其次是：
     *    -精确地（长度-4）个字节包含table_id，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节*/
    uint8_t            table_ids [0];          /* 表ID的列表。*/
};
OFP_ASSERT（sizeof（p_table_feature_prop_tables的结构== 4）;

/* 操作ID */
struct ofp_action_id {
    uint16_t类型;                 /* OFPAT_ *之一。*/
    uint16_t len;                 /* 长度为4或由实验人员定义。*/
    uint8_t exp_data [0];         /* 可选的实验者ID +数据。*/
};
OFP_ASSERT（sizeof（struct ofp_action_id）== 4）;

/* 动作属性*/
struct ofp_table_feature_prop_actions {
    uint16_t           类型;    /* OFPTFPT_WRITE_ACTIONS之一，
                                   OFPTFPT_WRITE_ACTIONS_MISS，
                                   OFPTFPT_APPLY_ACTIONS，
                                   OFPTFPT_APPLY_ACTIONS_MISS。*/
    uint16_t           长度;    /* 此属性的长度（以字节为单位）。*/
    /* 其次是：
     *    -恰好（length-4）个字节包含action_id，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节*/
    p_action_id结构        action_ids [0];       /* 动作清单*/
};
OFP_ASSERT（sizeof（p_table_feature_prop_actions的结构）== 4）;

/* 匹配，通配符或设置字段属性*/
struct ofp_table_feature_prop_oxm {
    uint16_t           类型;    /* OFPTFPT_MATCH之一，
                                   OFPTFPT_WILDCARDS，
                                   OFPTFPT_WRITE_SETFIELD，
                                   OFPTFPT_WRITE_SETFIELD_MISS，
                                   OFPTFPT_APPLY_SETFIELD，
                                   OFPTFPT_APPLY_SETFIELD_MISS，
                                   OFPTFPT_WRITE_COPYFIELD，
                                   OFPTFPT_WRITE_COPYFIELD_MISS，
                                   OFPTFPT_APPLY_COPYFIELD，
                                   OFPTFPT_APPLY_COPYFIELD_MISS。*/
    uint16_t           长度;    /* 此属性的长度（以字节为单位）。*/
    /* 其次是：
     *    -恰好（长度-4）个字节包含oxm_id，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
```



```
     *      全零字节的字节*/
```

```
};  uint32_t           oxm_ids [0];  / * OXM标头数组* /
OFP_ASSERT（sizeof（p_table_feature_prop_oxm的结构== 4）；

/ *数据包类型属性* /
p_table_feature_prop_oxm_values的结构{
    uint16_t          类型；    / * OFPTFPT_PACKET_TYPES。* /
    uint16_t          长度；    / *此属性的长度（以字节为单位）。* /
    /* 其次是：
     *    -确切地（长度-4）个字节包含oxm值，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节* /
    uint32_t          oxm_values [0]; / * OXM值数组* /
};
OFP_ASSERT（sizeof（p_table_feature_prop_oxm_values的结构== 4）；

/ *实验者表功能属性* /
p_table_feature_prop_experimenter的结构{
    uint16_t          类型；    / * OFPTFPT_EXPERIMENTER之一，
                                  OFPTFPT_EXPERIMENTER_MISS。* /
    uint16_t          长度；    / *此属性的长度（以字节为单位）。* /
    uint32_t          实验者          / *采用相同的实验者ID
                                  形式如结构
                                  ofp_experimenter_header。* /
    uint32_t          exp_type;      / *实验者定义。* /
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节* /
    uint32_t          实验者数据[0];
};
OFP_ASSERT（sizeof（p_table_feature_prop_experimenter的结构）== 12）；

/ *表支持的功能标志。* /
of_p_table_feature_flag的枚举{
    OFPTFF_INGRESS_TABLE = 1 << 0，/ *可以配置为入口表。* /
    OFPTFF_EGRESS_TABLE = 1 << 1，/ *可以配置为出口表。* /
    OFPTFF_FIRST_EGRESS = 1 << 4，/ *是第一个出口表。* /
};

/ *表功能请求命令* /
of_p_table_features_command的枚举{
    OFPTFC_REPLACE = 0，/ *替换完整的管道。* /
    OFPTFC_MODIFY          / *修改流表功能。* /
    OFPTFC_ENABLE          / *在管道中启用流表。* /
    OFPTFC_DISABLE = 3，/ *禁用管道中的流表。* /
};

/ *类型为OFPMP_TABLE_FEATURES的ofp_multipart_request的正文。/
  *回复OFPMP_TABLE_FEATURES请求的正文。* /
struct ofp_table_features {
    uint16_t长度；          / *长度填充为64位。* /
    uint8_t table_id;       / *表的标识符。          编号较低的表格
                              首先咨询。* /
    uint8_t命令；           / * OFPTFC_ *之一。* /
    uint32_t功能；          / * OFPTFF_ *值的位图。* /
    字符名称[OFP_MAX_TABLE_NAME_LEN]；
    uint64_t metadata_match; / *元数据表的位可以匹配。* /
    uint64_t metadata_write; / *元数据表的位可以写入。* /
    uint32_t功能；          / * OFPTC_ *值的位图。* /
    uint32_t max_entries;   / *支持的最大条目数。* /

    / *表功能属性列表* /
    p_table_feature_prop_header属性的结构[0]; / *属性列表* /
};
OFP_ASSERT（sizeof（struct ofp_table_features）== 64）；

/ *对OFPMP_TABLE_STATS请求的回复正文。* /
struct ofp_table_stats {
    uint8_t table_id;       / *表的标识符。          编号较低的表格
                              首先咨询。* /
    uint8_t pad [3];        / *对齐32位。* /
    uint32_t active_count;  / *活动条目数。* /
    uint64_t lookup_count;  / *在表中查找的数据包数。* /
    uint64_tmatched_count; / *命中表的数据包数量。* /
};
OFP_ASSERT（sizeof（struct ofp_table_stats）== 24）；

/ *对OFPMP_TABLE_DESC请求的回复正文。* /
struct ofp_table_desc {
    uint16_t长度；          / *长度填充为64位。* /
    uint8_t table_id;       / *表的标识符。          编号较低的表格
                              首先咨询。* /
    uint8_t pad [1];        / *对齐32位。* /
    uint32_t配置;           / * OFPTC_ *值的位图。* /

    / * Table Mod属性列表-0或更多。* /
    p_table_mod_prop_header属性的结构[0];
};
```



```
OFP_ASSERT（sizeof（p_table_desc的结构）== 8）；

/ *类型为OFPMP_PORT_STATS的ofp_multipart_request的主体，
  * OFPMP_PORT_DESC。* /
struct ofp_port_multipart_request {
    uint32_t port_no;       / * OFPMP_PORT消息必须请求统计信息
                              *单个端口（在
                              * port_no）或所有端口（如果port_no ==
                              * OFPP_ANY）。* /
    uint8_t pad [4];
};
OFP_ASSERT（sizeof（p_port_multipart_request的结构）== 8）；

/ *端口统计信息属性类型。
  * /
of_p_port_stats_prop_type的枚举{
    OFPPSPT_ETHERNET  = 0，      / *以太网属性。* /
    OFPPSPT_OPTICAL     = 1，    / *光学性质。* /
    OFPPSPT_EXPERIMENTER = 0xFFFF，/ *实验者属性。* /
};

/ *所有端口统计信息属性的公共头。* /
struct ofp_port_stats_prop_header {
    uint16_t          类型；    / * OFPPSPT_ *之一。* /
    uint16_t          长度；    / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_port_stats_prop_header的结构）== 4）；

/ *以太网端口统计信息属性。* /
struct ofp_port_stats_prop_ethernet {
    uint16_t          类型；    / * OFPPSPT_ETHERNET。* /
    uint16_t          长度；    / *此属性的长度（以字节为单位）。* /
    uint8_t           垫[4]；   / *对齐64位。* /

    uint64_t rx_frame_err;  / *帧对齐错误数。* /
    uint64_t rx_over_err;   / * RX溢出的数据包数。* /
    uint64_t rx_crc_err;    / * CRC错误数。* /
    uint64_t碰撞；          / *碰撞次数。* /
```

```
};
OFP_ASSERT（sizeof（p_port_stats_prop_ethernet的结构）== 40）；

/ *光端口统计信息属性。* /
struct ofp_port_stats_prop_optical {
    uint16_t            类型；    / * OFPPSPT_OPTICAL。* /
    uint16_t            长度；    / *此属性的长度（以字节为单位）。* /
    uint8_t             垫[4]；   / *对齐64位。* /

    uint32_t标志；                / *端口启用的功能。* /
    uint32_t tx_freq_lmda;  / *当前TX频率/波长* /
    uint32_t tx_offset;     / * TX偏移* /
    uint32_t tx_grid_span;  / * TX网格间距* /
    uint32_t rx_freq_lmda;  / *当前RX频率/波长* /
    uint32_t rx_offset;     / *接收偏移* /
    uint32_t rx_grid_span;  / * RX网格间距* /
    uint16_t tx_pwr;        / *当前发射功率* /
    uint16_t rx_pwr;        / *当前接收功率* /
    uint16_t bias_current;  / * TX偏置电流* /
    uint16_t温度；                / * TX激光温度* /
};
OFP_ASSERT（sizeof（p_port_stats_prop_optical的结构）== 44）；

/ *标志是以下OFPOSF_之一* /
ofp_port_stats_optical_flags的枚举{
    OFPOSF_RX_TUNE << 0，  / *接收器调谐信息有效* /
    OFPOSF_TX_TUNE << 1，  / *传送调谐信息有效* /
    OFPOSF_TX_PWR << 2，   / * TX电源有效* /
    OFPOSF_RX_PWR << 4，   / * RX电源有效* /
    OFPOSF_TX_BIAS << 5，  / *发送偏差有效* /
    OFPOSF_TX_TEMP << 6，  / * TX Temp有效* /
};

/ *实验者端口统计信息属性。* /
struct ofp_port_stats_prop_experimenter {
    uint16_t            类型；    / * OFPPSPT_EXPERIMENTER。* /
    uint16_t            长度；    / *此属性的长度（以字节为单位）。* /
    uint32_t            实验者          / *采用相同的实验者ID
                                          形式如结构
                                          ofp_experimenter_header。* /
    uint32_t            exp_type;       / *实验者定义。* /
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t            实验者数据[0];
};
OFP_ASSERT（sizeof（结构的p_port_stats_prop_experimenter）== 12）；

/ *对OFPMP_PORT_STATS请求的回复正文。如果不支持计数器，
 *将字段设置为全部。* /
struct ofp_port_stats {
```



```
    uint16_t长度；                / *此项的长度。* /
    uint8_t pad [2]；             / *对齐64位。* /
    uint32_t port_no;
    uint32_t duration_sec;  / *时间端口以秒为单位处于活动状态。* /
    uint32_t duration_nsec; / *时间端口已经存活了十亿分之一秒
                               duration_sec。* /
    uint64_t rx_packets;    / *接收到的数据包数。* /
    uint64_t tx_packets;    / *传输的数据包数。* /
    uint64_t rx_bytes;      / *接收的字节数。* /
    uint64_t tx_bytes;      / *传输的字节数。* /

    uint64_t rx_dropped;    / * RX丢弃的数据包数。* /
    uint64_t tx_dropped;    / * TX丢弃的数据包数。* /
    uint64_t rx_errors;     / *接收错误数。                这是一个超集
                               更具体的接收错误，应该是
                               大于或等于所有的总和
                               属性中的rx _ * _ err值。* /
    uint64_t tx_errors;     / *传输错误数。                这是一个超集
                               更具体的传输错误，应该是
                               大于或等于所有的总和
                               tx _ * _ err值（当前未定义。）* /

    / *端口描述属性列表-0个或多个属性* /
    p_port_stats_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_port_stats）== 80）；

/ * OFPMP_GROUP_STATS和OFPMP_GROUP_DESC请求的主体。* /
p_group_multipart_request的结构{
    uint32_t group_id;      / *所有组（如果为OFPG_ALL）。* /
    uint8_t pad [4];        / *对齐64位。* /
};
OFP_ASSERT（sizeof（p_group_multipart_request的结构）== 8）；

/ *用于群组统计信息回复。* /
struct ofp_bucket_counter {
    uint64_t packet_count;  / *桶处理的数据包数。* /
    uint64_t byte_count;    / *存储桶处理的字节数。* /
};
OFP_ASSERT（sizeof（p_bucket_counter的结构）== 16）；

/ *对OFPMP_GROUP_STATS请求的回复正文。* /
struct ofp_group_stats {
    uint16_t长度；                / *此项的长度。* /
    uint8_t pad [2]；             / *对齐64位。* /
    uint32_t group_id;      / *组标识符。* /
    uint32_t ref_count;     / *直接转发的流或组的数量
                               到这个小组。* /
    uint8_t pad2 [4];       / *对齐64位。* /
    uint64_t packet_count;  / *按组处理的数据包数。* /
    uint64_t byte_count;    / *按组处理的字节数。* /
    uint32_t duration_sec;  / *时间组仅以秒为单位。* /
    uint32_t duration_nsec; / *时间组的生存时间超过了十亿分之一秒
                               duration_sec。* /
    p_bucket_counter bucket_stats [0]; / *每个存储桶设置一个计数器。* /
};
OFP_ASSERT（sizeof（struct ofp_group_stats）== 40）；

/ *对OFPMP_GROUP_DESC请求的回复正文。* /
struct ofp_group_desc {
    uint16_t长度；                / *此项的长度。* /
    uint8_t类型；                 / * OFPGT_ *之一。* /
    uint8_t pad;                  / *填充到64位。* /
    uint32_t group_id;            / *组标识符。* /
    uint16_t bucket_array_len;    / *操作存储区数据的长度。* /
    uint8_t pad2 [6];             / *填充到64位。* /
    /* 其次是：
     *    -确切的“ bucket_array_len”字节包含一个数组
     *       p_bucket结构，
     *    -零个或多个字节的组属性可填充整体
     *       长度字段中的长度。* /
    p_bucket桶的结构[0];          / *桶清单-0或更多。* /
    // p_group_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_group_desc）== 16）；
```

```
/ *与1.3.1的向后兼容性-避免破坏API。* /
#define ofp_group_desc_stats ofp_group_desc

/ *组配置标志* /
ofp_group_capabilities枚举{
    OFPGFC_SELECT_WEIGHT = 1 << 0，/ *选择组的支撑重量* /
    OFPGFC_SELECT_LIVENESS = 1 << 1，/ *支持确定群体的活力* /
    OFPGFC_CHAINING = 1 << 2，/ *支持链接组* /
    OFPGFC_CHAINING_CHECKS = 1 << 3 / *检查循环链接并删除* /
};

/ *回复OFPMP_GROUP_FEATURES请求的正文。组功能。* /
struct ofp_group_features {
    uint32_t 类型; / *支持（1 << OFPGT_ *）个值的位图。* /
```



```
    uint32_t 能力; / *支持OFPGFC_ *功能的位图。* /
    uint32_t max_groups [4]; / *每种类型的最大组数。* /
    uint32_t 行动[4]; / *支持（1 << OFPAT_ *）个值的位图。* /
};
OFP_ASSERT（sizeof（struct ofp_group_features）== 40）；

/ * OFPMP_METER_STATS和OFPMP_METER_DESC请求的主体。* /
struct ofp_meter_multipart_request {
    uint32_t meter_id; / *仪表实例，或OFPM_ALL。* /
    uint8_t pad [4]; / *对齐64位。* /
};
OFP_ASSERT（sizeof（p_meter_multipart_request的结构）== 8）；

/ *每个米波段的统计数据* /
struct ofp_meter_band_stats {
    uint64_t packet_band_count; / *频段中的数据包数。* /
    uint64_t byte_band_count; / * band中的字节数。* /
};
OFP_ASSERT（sizeof（p_meter_band_stats的结构）== 16）；

/ *对OFPMP_METER_STATS请求的回复正文。仪表统计。* /
struct ofp_meter_stats {
    uint32_t meter_id; / *仪表实例。* /
    uint16_t len; / *此统计信息的长度（以字节为单位）。* /
    uint8_t 垫[6];
    uint32_t ref_count; / *的流或组的数量
                           直接参考此仪表。* /
    uint64_t packet_in_count; / *输入中的包数。* /
    uint64_t byte_in_count; / *输入的字节数。* /
    uint32_t duration_sec; / *计时器以秒为单位处于活动状态。* /
    uint32_t duration_nsec; / *计时器的活动时间超过十亿分之一秒
                               duration_sec。* /
    p_meter_band_stats的结构band_stats [0]; / * band_stats的长度为
                                              从长度字段推断。* /
};
OFP_ASSERT（sizeof（p_meter_stats的结构）== 40）；

/ *对OFPMP_METER_DESC请求的回复正文。仪表配置。* /
struct ofp_meter_desc {
    uint16_t 长度; / *此项的长度。* /
    uint16_t 标志; / *所有适用的OFPMF_ *。* /
    uint32_t meter_id; / *仪表实例。* /
    struct ofp_meter_band_header bands [0]; / *波段长度为
                                              从长度字段推断。* /
};
OFP_ASSERT（sizeof（p_meter_desc的结构）== 8）；

/ *仪表功能标志* /
of_p_meter_feature_flags的枚举{
    OFPMFF_ACTION_SET = 1 << 0，/ *在操作集中支持仪表操作。* /
    OFPMFF_ANY_POSITION = 1 << 1，/ *支持动作列表中的任何位置。* /
    OFPMFF_MULTI_LIST = 1 << 2，/ *支持动作列表中的多个动作。* /
};

/ *回复OFPMP_METER_FEATURES请求的正文。仪表功能。* /
struct ofp_meter_features {
    uint32_t max_meter; / *最大米数。* /
    uint32_t band_types; / *支持（1 << OFPMBT_ *）个值的位图。* /
    uint32_t 能力; / *“ ofp_meter_flags”的位图。* /
    uint8_t max_bands; / *每米的最大频段* /
    uint8_t max_color; / *最大色值* /
    uint8_t 垫[2];
    uint32_t 特征; / *“ ofp_meter_feature_flags”的位图。* /
    uint8_t pad2 [4];
};
OFP_ASSERT（sizeof（p_meter_features的结构）== 24）；

/ *“全1”用于指示端口中的所有队列（用于统计信息检索）。* /
#define OFPQ_ALL 0xffffffff

/ *最小速率> 1000表示未配置。* /
#define OFPQ_MIN_RATE_UNCFG 0xffff

/ *最大速率> 1000表示未配置。* /
#define OFPQ_MAX_RATE_UNCFG 0xffff

of_p_queue_desc_prop_type的枚举{
    OFPQDPT_MIN_RATE = 1 / *保证最低数据速率。* /
    OFPQDPT_MAX_RATE = 2 / *最大数据速率。* /
    OFPQDPT_EXPERIMENTER = 0xFFFF / *实验者定义的属性。* /
};

/ *所有队列属性的通用标头* /
struct ofp_queue_desc_prop_header {
    uint16_t 类型; / * OFPQDPT_ *之一。* /
    uint16_t 长度; / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_queue_desc_prop_header的结构== 4）；

/ *最小速率队列属性描述。* /
```

```
struct ofp_queue_desc_prop_min_rate {
    uint16_t           类型;    / * OFPQDPT_MIN_RATE。* /
    uint16_t           长度;    / *长度为8。* /
    uint16_t率;                 / * 1 /10的百分比；> 1000->禁用。* /
    uint8_t pad [2];           / * 64位对齐* /
};
OFP_ASSERT（sizeof（struct ofp_queue_desc_prop_min_rate）== 8）；

/ *最大速率队列属性描述。* /
struct ofp_queue_desc_prop_max_rate {
    uint16_t           类型;    / * OFPQDPT_MAX_RATE。* /
    uint16_t           长度;    / *长度为8。* /
    uint16_t率;                 / * 1 /10的百分比；> 1000->禁用。* /
    uint8_t pad [2];           / * 64位对齐* /
};
OFP_ASSERT（sizeof（struct ofp_queue_desc_prop_max_rate）== 8）；

/ *实验者队列属性描述。* /
struct ofp_queue_desc_prop_experimenter {
    uint16_t           类型;    / * OFPQDPT_EXPERIMENTER。* /
    uint16_t           长度;    / *此属性的长度（以字节为单位）。* /
    uint32_t           实验者          / *采用相同的实验者ID
                                         形式如结构
                                         ofp_experimenter_header。* /
    uint32_t           exp_type;       / *实验者定义。* /
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *    全零字节的字节* /
    uint32_t           实验者数据[0];
};
OFP_ASSERT（sizeof（p_queue_desc_prop_experimenter的结构）== 12）；

/ *类型为OFPMP_QUEUE_DESC的ofp_multipart_request的主体
 * OFPMP_QUEUE_STATS。* /
struct ofp_queue_multipart_request {
    uint32_t port_no;          / *所有端口（如果为OFPP_ANY）。* /
    uint32_t queue_id;         / *如果是OFPQ_ALL，则所有队列。* /
};
OFP_ASSERT（sizeof（p_queue_multipart_request的结构）== 8）；

/ *对OFPMP_QUEUE_DESC请求的回复正文。* /
struct ofp_queue_desc {
    uint32_t port_no;          / *此队列连接到的端口。* /
    uint32_t queue_id;         / *特定队列的ID。* /
    uint16_t len;              / *此队列desc的字节长度。* /
    uint8_t pad [6];           / * 64位对齐。* /

    p_queue_desc_prop_header属性的结构[0]; / *属性列表。* /
};
OFP_ASSERT（sizeof（p_queue_desc的结构）== 16）；

ofp_queue_stats_prop_type的枚举{
    OFPQSPT_EXPERIMENTER = 0xFFFF / *实验者定义的属性。* /
};

/ *所有队列属性的通用标头* /
struct ofp_queue_stats_prop_header {
    uint16_t           类型;    / * OFPQSPT_ *之一。* /
    uint16_t           长度;    / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（struct ofp_queue_stats_prop_header）== 4）；

/ *实验者队列属性描述。* /
struct ofp_queue_stats_prop_experimenter {
    uint16_t           类型;    / * OFPQSPT_EXPERIMENTER。* /
    uint16_t           长度;    / *此属性的长度（以字节为单位）。* /
    uint32_t           实验者          / *采用相同的实验者ID
                                         形式如结构
                                         ofp_experimenter_header。* /
    uint32_t           exp_type;       / *实验者定义。* /
    /* 其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *    全零字节的字节* /
    uint32_t           实验者数据[0];
};
OFP_ASSERT（sizeof（p_queue_stats_prop_experimenter的结构）== 12）；

/ *对OFPMP_QUEUE_STATS请求的回复正文。* /
struct ofp_queue_stats {
    uint16_t长度;              / *此项的长度。* /
    uint8_t pad [6];           / *对齐64位。* /
    uint32_t port_no;          / *队列连接到的端口。* /
    uint32_t queue_id;         / *队列ID * /
    uint64_t tx_bytes;         / *传输的字节数。* /
    uint64_t tx_packets;       / *传输的数据包数。* /
    uint64_t tx_errors;        / *由于溢出而丢弃的数据包数量。* /
    uint32_t duration_sec;     / *时间队列以秒为单位还活着。* /
    uint32_t duration_nsec;    / *时间队列已经超过了十亿分之一秒
```



```
                                    duration_sec。* /

    p_queue_stats_prop_header属性的结构[0]; / *属性列表。* /
};
OFP_ASSERT（sizeof（struct ofp_queue_stats）== 48）；

/ *类型为OFPMP_FLOW_MONITOR的ofp_multipart_request的正文。
 *
 * OFPMP_FLOW_MONITOR请求的主体由零或更大的数组组成
 *此结构的实例。                该请求安排监视流程
 *符合指定条件，其解释方式与
 *用于OFPMP_FLOW。
 *
 *'id'标识特定的监视器，以使其成为
 *稍后用OFPFMC_DELETE取消。“ id”必须是唯一的
 *尚未取消的现有监视器。
 * /
struct ofp_flow_monitor_request {
    uint32_t monitor_id;       / *此监视器的控制器分配的ID。* /
    uint32_t out_port;         / *必需的输出端口，如果不是OFPP_ANY。* /
    uint32_t out_group;        / *必需的组号（如果不是OFPG_ANY）。* /
    uint16_t标志;              / * OFPFMF_ *。* /
    uint8_t table_id;          / *一个表的ID或OFPTT_ALL（所有表）。* /
    uint8_t命令;               / * OFPFMC_ *之一。* /
    struct ofp_match match;    / *要匹配的字段。大小可变。* /
};
OFP_ASSERT（sizeof（struct ofp_flow_monitor_request）== 24）；

/ *流量监控器命令* /
of p_flow_monitor_command的枚举{
    OFPFMC_ADD,                / *新的流量监视器。* /
    OFPFMC_MODIFY = 1，/ *修改现有的流量监视器。* /
    OFPFMC_DELETE = 2 / *删除/取消现有的流量监视器。* /
};
```

```
/*结构of_flow_monitor_request中的“标志”位。*/
of_p_flow_monitor_flags的枚举{
    /*何时发送更新。*/
    OFPFMF_INITIAL = 1 << 0,*最初匹配的流。*/
    OFPFMF_ADD = 1 << 1,  /*添加新匹配流。*/
    OFPFMF_REMOVED = 1 << 2,*已删除匹配流将被删除。*/
    OFPFMF_MODIFY = 1 << 3,*匹配流，因为它们已更改。*/

    /*包含在更新中的内容。*/
    OFPFMF_INSTRUCTIONS = 1 << 4, /*如果置位，则包含说明。*/
    OFPFMF_NO_ABBREV = 1 << 5,*如果已设置，请完全包含自己的更改。*/
    OFPFMF_ONLY_OWN = 1 << 6,*如果设置，则不包括其他控制器。*/
};

/* OFPMP_FLOW_MONITOR答复标头。
 *
 * OFPMP_FLOW_MONITOR答复的正文是可变长度的数组
 *结构，每个结构都以该标头开头。“长度”成员可以
 *用于遍历数组，并且'event'成员可以用于
 *确定特定的结构。
 *
 *每个实例都是8字节长的倍数。*/
struct ofp_flow_update_header {
    uint16_t长度;              /*此项的长度。*/
    uint16_t事件;              /* OFPFME_*之一。*/
    /* ...取决于“事件”的其他数据... */
};
OFP_ASSERT（sizeof（p_flow_update_header的结构）== 4）；

/* p_flow_update_header结构中的'event'值。*/
of_p_flow_update_event的枚举{
    /* struct ofp_flow_update_full。*/
    OFPFME_INITIAL = 0，    /*创建流量监控器时存在流量。*/
    OFPFME_ADDED = 1       /*添加了流。*/
    OFPFME_REMOVED = 2  /*流程已删除。*/
    OFPFME_MODIFIED = 3, /*流程说明已更改。*/

    / * struct ofp_flow_update_abbrev。*/
    OFPFME_ABBREV = 4，   /*缩写答复。*/

    / * struct ofp_flow_update_header。*/
    OFPFME_PAUSED = 5     /*监视已暂停（缓冲区空间不足）。*/
    OFPFME_RESUMED = 6,/*恢复监视。*/
};

/* OFPMP_FLOW_MONITOR回复OFPFME_INITIAL，OFPFME_ADDED，OFPFME_REMOVED，
 *和OFPFME_MODIFIED。*/
struct ofp_flow_update_full {
    uint16_t长度;              /*长度为32 +匹配+指令。*/
    uint16_t事件;              /* OFPFME_*之一。*/
    uint8_t table_id;          /*流表的ID。*7
    uint8_t原因;               /* OFPRR_*为OFPFME_REMOVED，否则为零。*/
    uint16_t idle_timeout;     /*到期前空闲的秒数。*7
    uint16_t hard_timeout;     /*到期前的秒数。*/
    uint16_t优先级;            /*条目的优先级。*/
```



```
    uint8_t零[4];              /*保留，当前为零。*/
    uint64_t cookie;           /*不透明的控制器发出的标识符。*/
    struct ofp_match match;    /*要匹配的字段。大小可变。*/
    /*      指令系统。
     *      如果未指定OFPFME_INSTRUCTIONS，或者“事件”为
     *      OFPFME_REMOVED，不包含任何指令。
     */
    // p_instruction指令的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_flow_update_full）== 32）；

/* OFPMP_FLOW_MONITOR回复OFPFME_ABBREV。
 *
 *当控制器未在监视请求中指定OFPFMF_NO_ABBREV时，
 *由于控制器自身的请求（在同一时间），任何流表都会发生变化
 *（OpenFlow通道），将在可能的情况下缩写为这种形式，
 *仅指定OpenFlow请求的“ xid”（例如OFPT_FLOW_MOD）
 *导致更改。
 *有些更改不能缩写，将被完整发送。
 */
struct ofp_flow_update_abbrev {
    uint16_t长度;              /*长度为8。*/
    uint16_t事件;              /* OFPFME_ABBREV。*/
    uint32_t xid;              /*来自flow_mod的控制器指定的xid。*/
};
OFP_ASSERT（sizeof（struct ofp_flow_update_abbrev）== 8）；

/* OFPMP_FLOW_MONITOR回复OFPFME_PAUSED和OFPFME_RESUMED。
 */
struct ofp_flow_update_paused {
    uint16_t长度;              /*长度为8。*/
    uint16_t事件;              /* OFPFME_*之一。*/
    uint8_t零[4];              /*保留，当前为零。*/
};
OFP_ASSERT（sizeof（p_flow_update_paused的结构）== 8）；

/*控制器状态属性类型。
 */
of_p_controller_status_prop_type的枚举{
    OFPCSPT_URI                = 0，    /*连接URI属性。*/
    OFPCSPT_EXPERIMENTER= 0xFFFF，/*实验者属性。*/
};

/*所有控制器状态属性的通用头*/
struct ofp_controller_status_prop_header {
    uint16_t          类型;    /* OFPCSPT_*之一。*/
    uint16_t          长度;    /*此属性的长度（以字节为单位）。*/
};
OFP_ASSERT（sizeof（p_controller_status_prop_header的结构）== 4）；

/*连接URI控制器状态属性*/
struct ofp_controller_status_prop_uri {
    uint16_t          类型;    /* OFPCSPT_URI。*/
    uint16_t          长度;    /*此属性的长度（以字节为单位）。*/

    /* 其次是：
     *   -恰好（长度-4）个字节包含连接URI，然后
     *   -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节*/
    烧焦              uri [0];
};
OFP_ASSERT（sizeof（p_controller_status_prop_uri的结构== 4）；

/*实验者控制器状态属性*/
p_controller_status_prop_experimenter的结构{
    uint16_t          类型;    /* OFPCSPT_EXPERIMENTER。*/
    uint16_t          长度;    /*此属性的长度（以字节为单位）。*/
    uint32_t          实验者          /*采用相同的实验者ID
                                        形式如结构
                                        ofp_experimenter_header。*/
    uint32_t          exp_type;       /*实验者定义。*/
```

```
    /* 其次填充地（长度-12个）字节包含实验者数据。然后
     *    -准确（长度+7）/ 8 * 8-（长度）（0至7之间）
     *    全零字节的字节* /
    uint32_t                实验者数据[0];
};
OFP_ASSERT（sizeof（p_controller_status_prop_experimenter的结构）== 12）；

/ *为什么要报告控制器状态？ * /
of_p_controller_status_reason的枚举{
    OFPCSR_REQUEST      = 0，/ *控制器请求的状态。* /
    OFPCSR_CHANNEL_STATUS = 1，/ *通道的运行状态已更改。* /
    OFPCSR_ROLE         = 2，/ *控制器角色已更改。* /
    OFPCSR_CONTROLLER_ADDED = 3，/ *添加了新控制器。* /
    OFPCSR_CONTROLLER_REMOVED = 4，/ *从配置中删除。* /
    OFPCSR_SHORT_ID     = 5，/ *控制器ID已更改。* /
    OFPCSR_EXPERIMENTER = 6 / *实验者数据已更改。* /
};
```



```
/ * OFPMP_CONTROLLER_STATUS回复正文和异步正文
 * OFPT_CONTROLLER_STATUS消息* /
struct ofp_controller_status {
    uint16_t长度；                / *此项的长度。* /
    uint16_t short_id;              / *标识控制器的ID号。* /
    uint32_t角色；                / *控制器的角色。OFPCR_ROLE_ *之一。* /
    uint8_t原因；                 / * OFPCSR_ *原因码之一。* /
    uint8_t channel_status;         / *控制通道的状态。
                                      * OFPCT_STATUS_ *之一。* /
    uint8_t pad [6];                / *对齐64位。* /

    / *控制器状态属性列表。          连接URI属性是
       需要; 其他属性是可选的。* /
    p_controller_status_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_controller_status的结构）== 16）；

struct ofp_controller_status_header {
    struct ofp_header标头；                / *输入OFPT_CONTROLLER_STATUS。* /
    p_controller_status状态的结构；        / *控制器状态。* /
};

/ *控制通道状态。* /
of_p_control_channel_status的枚举{
    OFPCT_STATUS_UP     = 0，/ *控制通道可操作。* /
    OFPCT_STATUS_DOWN = 1 / *控制通道无法运行。* /
};

struct ofp_bundle_features_prop_header {
    uint16_t类型；              / * OFPTMPBF_ *之一。* /
    uint16_t长度；              / *此属性的长度（以字节为单位）。* /
};
OFP_ASSERT（sizeof（p_bundle_features_prop_header的结构）== 4）；

/ * OFPMP_BUNDLE_FEATURES请求的主体。* /
struct ofp_bundle_features_request {
    uint32_t feature_request_flags;  / *“ ofp_bundle_feature_flags”的位图。* /
    uint8_t pad [4];

    / *捆绑功能特性列表-0或更多。* /
    p_bundle_features_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_bundle_features_request的结构）== 8）；

/ *捆绑包具有属性类型。* /
of_pundle_features_prop_type的枚举{
    OFPTMPBF_TIME_CAPABILITY = 0x1，/ *时间要素属性。* /
    OFPTMPBF_EXPERIMENTER = 0xFFFF，/ *实验者属性。* /
};

/* 时间格式。* /
struct ofp_time {
    uint64_t秒；
    uint32_t纳秒；
    uint8_t   垫[4];
};
OFP_ASSERT（sizeof（struct ofp_time）== 16）；

/ *捆绑时间功能。* /
struct ofp_bundle_features_prop_time {
    uint16_t类型；                        / * OFPTMPBF_TIME_CAPABILITY。* /
    uint16_t长度；                        / *此属性的长度（以字节为单位）。* /
    uint8_t pad [4];
    struct ofp_time sched_accuracy;  / *调度精度，即
                                       *开关可以准确执行
                                       *预定的提交。使用该字段
                                       *仅在捆绑包功能中提供回复，并且
                                       *在捆绑包功能中被忽略
                                       * 要求。* /
    p_time sched_max_future结构；  / *之间的最大差
                                       *安排时间和当前时间。* /
    struct ofp_time sched_max_past; / *如果计划时间在
                                       *过去，定义最大差异
                                       *当前时间和
                                       *安排时间。* /
    struct ofp_time timestamp;       / *表示
                                       *发送此消息。* /
};
OFP_ASSERT（sizeof（p_bundle_features_prop_time的结构== 72）；

of_pundle_feature_flags的枚举{
    OFPBF_TIMESTAMP = 1 << 0，/ *请求中包含时间戳。* /
    OFPBF_TIME_SET_SCHED = 1 << 1 / *请求包括sched_max_future和
                                   * sched_max_past参数。* /
};

/ *回复OFPMP_BUNDLE_FEATURES请求的正文。* /
struct ofp_bundle_features {
    uint16_t功能；  / *“ ofp_bundle_flags”的位图。* /
    uint8_t pad [6];
```

```
    /*捆绑功能特性列表-0或更多。*/
    p_bundle_features_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（p_bundle_features的结构）== 8）；

/*类型为OFPMP_EXPERIMENTER的ofp_multipart_request / reply的正文。*/
struct ofp_experimenter_multipart_header {
    uint32_t实验者；          /*实验者ID。*/
    uint32_t exp_type;        /*实验者定义。*/
    /*实验者定义的任意附加数据。*/
};
OFP_ASSERT（sizeof（p_experimenter_multipart_header的结构）== 8）；

/*典型的实验者结构。*/
struct ofp_experimenter_structure {
    uint32_t实验者；              /*实验者ID：
                                   *-MSB 0：低位字节是IEEE OUI。
                                   *-MSB！= 0：由ONF定义。*/
    uint32_t exp_type;            /*实验者定义。*/
    uint8_t   实验者数据[0];
};
OFP_ASSERT（sizeof（p_experimenter_structure的结构）== 8）；

/*实验者扩展消息。*/
struct ofp_experimenter_msg {
    struct ofp_header标头；       /*输入OFPT_EXPERIMENTER。*/
    uint32_t实验者；              /*实验者ID：
                                   *-MSB 0：低位字节是IEEE OUI。
                                   *-MSB！= 0：由ONF定义。*/
    uint32_t exp_type;            /*实验者定义。*/
    /*实验者定义的任意附加数据。*/
    uint8_t   实验者数据[0];
};
OFP_ASSERT（sizeof（p_experimenter_msg的结构）== 16）；

/*配置发送控制器的“角色”。                                默认角色是：
 *
 *    -等于（OFPCR_ROLE_EQUAL），允许控制器访问所有
 *      OpenFlow功能。所有控制器都有同等的责任。
 *
 *其他可能的角色是相关的对：
 *
 *    -Master（OFPCR_ROLE_MASTER）等于Equal，除了
 *      一次最多可以是一个主控制器：
 *      将自己配置为Master，任何现有的Master都会降级为
 *      从角色。
 *
 *    -从站（OFPCR_ROLE_SLAVE）允许控制器以只读方式访问
 *      OpenFlow功能。  特别是尝试修改流表
 *      将被拒绝，并显示OFPBRC_EPERM错误。
 *
 *      从控制器不接收OFPT_PACKET_IN或OFPT_FLOW_REMOVED
 *      消息，但它们确实收到OFPT_PORT_STATUS消息。
 *
 *还配置控制器的ID。                                      默认ID是
 * OFPCID_UNDEFINED，并且控制器可以设置short_id来更改其ID。
 *只能将OFPCID_UNDEFINED ID分配给多个控制器，
 *并将short_id设置为使用中的值将被拒绝。
 */

/*控制器角色。*/
of_p_controller_role的枚举{
    OFPCR_ROLE_NOCHANGE = 0,  /*不要更改当前角色。*/
    OFPCR_ROLE_EQUAL    = 1,  /*默认角色，完全访问权限。*/
    OFPCR_ROLE_MASTER   = 2,  /*完全访问权限，最多一个主服务器。*/
    OFPCR_ROLE_SLAVE    = 3,  /*只读访问。*/
};

/*角色请求和回复消息。*/
struct ofp_role_request {
    struct ofp_header标头；       /*类型OFPT_ROLE_REQUEST / OFPT_ROLE_REPLY。*/
    uint32_t角色；                /* OFPCR_ROLE_ *之一。*/
    uint16_t short_id;            /*控制器的ID号。*/
    uint8_t pad [2];              /*对齐64位。*/
    uint64_t generation_id;       /*主选举产生ID */
};
OFP_ASSERT（sizeof（struct ofp_role_request）== 24）；

＃定义OFPCID_UNDEFINED 0

/*角色属性类型。
 */
of_p_role_prop_type的枚举{
    OFPRPT_EXPERIMENTER = 0xFFFF，/*实验者属性。*/
};

/*所有角色属性的通用标头*/
struct ofp_role_prop_header {
    uint16_t              类型；    /* OFPRPT_ *之一。*/
```



```
    uint16_t              长度；    /*此属性的长度（以字节为单位）。*/
};
OFP_ASSERT（sizeof（struct ofp_role_prop_header）== 4）；

/*实验者角色属性*/
struct ofp_role_prop_experimenter {
    uint16_t              类型；    /* OFPRPT_EXPERIMENTER之一。*/
    uint16_t              长度；    /*此属性的长度（以字节为单位）。*/
    uint32_t              实验者          /*采用相同的实验者ID
                                           形式如结构
                                           ofp_experimenter_header。*/
    uint32_t              exp_type;       /*实验者定义。*/
    /*其次是：
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *      全零字节的字节*/
    uint32_t              实验者数据[0];
};
OFP_ASSERT（sizeof（p_role_prop_experimenter的结构）== 12）；

/*关于控制器角色的更改*/
of_p_controller_role_reason的枚举{
    OFPCRR_MASTER_REQUEST = 0,  /*另一个控制器要求是主控制器。*/
    OFPCRR_CONFIG  = 1,  /*开关上的配置已更改。*/
    OFPCRR_EXPERIMENTER = 2,  /*实验者数据已更改。*/
};

/*角色状态事件消息。*/
struct ofp_role_status {
    struct ofp_header标头；       /*输入OFPT_ROLE_STATUS。*/
    uint32_t角色；                /* OFPCR_ROLE_ *之一。*/
```

```
    uint8_t pad[3];             /* OFPCR_*之一。*/
    uint64_t generation_id;     /*主选举产生ID*/

    /*角色属性列表*/
    p_role_prop_header属性的结构[0];
};
OFP_ASSERT（sizeof（struct ofp_role_status）== 24）；

/*异步配置属性类型。
 *低位值零表示从属角色的属性。
 *低位设置指示主/等角色的属性。
 */
of_p_async_config_prop_type的枚举{
    OFPACPT_PACKET_IN_SLAVE/*从站的入站掩码。*/
    OFPACPT_PACKET_IN_MASTER=主机的入库掩码。*/
    OFPACPT_PORT_STATUS=SLAVE从站的端口状态掩码。*/
    OFPACPT_PORT_STATUS_MASTER服务器的端口状态掩码。*/
    OFPACPT_FLOW_REMOVED_SLAVE从属流的掩码。*/
    OFPACPT_FLOW_REMOVED_MASTER母版的流量。*/
    OFPACPT_ROLE_STATUS=SLAVE从站的角色状态掩码。*/
    OFPACPT_ROLE_STATUS=MASTER服务器的角色状态掩码。*/
    OFPACPT_TABLE_STATUS_SLAVE站的表状态掩码。*/
    OFPACPT_TABLE_STATUS_MASTER服务器的表状态掩码。*/
    OFPACPT_REQUESTFORWARD_SLAVE从站的RequestForward掩码。*/
    OFPACPT_REQUESTFORWARD_MASTER = 11，/*主服务器的RequestForward掩码。*/
    OFPACPT_FLOW_STATS_SLAVE从站的流量统计信息掩码。*/
    OFPACPT_FLOW_STATS_MASTER主设备的流量统计信息掩码。*/
    OFPACPT_CONT_STATUS_SLAVE从站的控制器状态掩码。*/
    OFPACPT_CONT_STATUS_MASTER控制器的控制器状态掩码。*/
    OFPACPT_EXPERIMENTER_SLAVE/*从器件的实验器。*/
    OFPACPT_EXPERIMENTER_MASTER/*实验者为主。*/
};

/*所有异步配置属性的通用头*/
p_async_config_prop_header的结构{
    uint16_t        类型;    /* OFPACPT_*之一。*/
    uint16_t        长度;    /*此属性的长度（以字节为单位）。*/
};
OFP_ASSERT（sizeof（p_async_config_prop_header的结构）== 4）；

/*基于各种原因的属性*/
p_async_config_prop_reasons的结构{
    uint16_t        类型;    /* OFPACPT_PACKET_IN_*中的一个，
                                OFPACPT_PORT_STATUS_*，
                                OFPACPT_FLOW_REMOVED_*，
                                OFPACPT_ROLE_STATUS_*，
                                OFPACPT_TABLE_STATUS_*，
                                OFPACPT_REQUESTFORWARD_*，
                                OFPACPT_FLOW_STATS_*，
                                OFPACPT_CONT_STATUS_*。*/
    uint16_t        长度;    /*此属性的长度（以字节为单位）。*/
    uint32_t        面具;    /*原因值的位掩码。*/
};
OFP_ASSERT（sizeof（p_async_config_prop_reasons的结构）== 8）；

/*实验者异步配置 物业*/
p_async_config_prop_experimenter的结构{
    uint16_t        类型;    /* OFPTFPT_EXPERIMENTER_SLAVE之一，
                                OFPTFPT_EXPERIMENTER_MASTER。*/
    uint16_t        长度;    /*此属性的长度（以字节为单位）。*/
    uint32_t        实验者       /*采用相同的实验者ID
                                    形式如结构
                                    ofp_experimenter_header。*/
    uint32_t        exp_type;    /*实验者定义。*/
    /*其次是：
     *   -确切地（长度-12个）字节包含实验者数据，然后
     *   -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *     全零字节的字节*/
    uint32_t        实验者数据[0];
};
OFP_ASSERT（sizeof（p_async_config_prop_experimenter的结构）== 12）；

/*异步消息配置。*/
struct ofp_async_config {
    struct ofp_header标头；      /* OFPT_GET_ASYNC_REPLY或OFPT_SET_ASYNC。*/

    /*异步配置属性列表-0或更多*/
    p_async_config_prop_header属性的结构[0]；
};
OFP_ASSERT（sizeof（p_async_config的结构）== 8）；

/*表格发生了什么变化*/
of_p_table_reason枚举{
    OFPTR_VACANCY_DOWN  /*空缺人数减少阈值事件。*/
    OFPTR_VACANCY_UP    /*空缺阈值事件。*/
};

/*表配置已在数据路径中更改*/
struct ofp_table_status {
    struct ofp_header标头；
    uint8_t原因；               /* OFPTR_*之一。*/
    uint8_t pad[7]；            /*填充至64位*/
    struct ofp_table_desc表；   /*新表配置。*/
};
OFP_ASSERT（sizeof（struct ofp_table_status）== 24）；

/*请求转发原因*/
ofp_requestforward_reason枚举{
    OFPRFR_GROUP_MOD = 0转发组mod请求。*/
    OFPRFR_METER_MOD = 1转发仪表Mod请求。*/
};

/*组/仪表请求转发。*/
struct ofp_requestforward_header {
    struct ofp_header标头；     /*输入OFPT_REQUESTFORWARD。*/
    struct ofp_header请求；     /*正在转发请求。*/
};
OFP_ASSERT（sizeof（p_requestforward_header的结构）== 16）；

/*捆绑包属性类型。*/
of_pundle_prop_type的枚举{
    OFPBPT_TIME                 = 1        /*时间属性。*/
    OFPBPT_EXPERIMENTER = 0xFFFF，/*实验者属性。*/
};

/*所有捆绑包属性的通用标头*/
struct ofp_bundle_prop_header {
    uint16_t        类型;    /* OFPBPT_*之一。*/
    uint16_t        长度;    /*此属性的长度（以字节为单位）。*/
};
OFP_ASSERT（sizeof（p_bundle_prop_header的结构）== 4）；

/*实验者捆绑包属性*/
struct ofp_bundle_prop_experimenter {
    uint16_t        类型;    /* OFPBPT_EXPERIMENTER。*/
    uint16_t        长度;    /*此属性的长度（以字节为单位）。*/
```

```
    uint32_t        实验者          / *采用相同的实验者ID
                                      形式如结构
                                      ofp_experimenter_header。* /
    uint32_t        exp_type;       / *实验者定义。* /
    /* 其次是:
     *    -确切地（长度-12个）字节包含实验者数据，然后
     *    -准确（长度+ 7）/ 8 * 8-（长度）（0至7之间）
     *       全零字节的字节* /
    uint32_t        实验者数据[0];
};
OFP_ASSERT（sizeof（p_bundle_prop_experimenter的结构）== 12）;

/ *捆绑时间属性。* /
    struct ofp_bundle_prop_time {
    uint16_t类型;          / * OFPBPT_TIME * /
    uint16_t长度;          / *此属性的长度，以字节为单位（24）。* /
    uint8_t   垫[4];
    struct ofp_time schedule_time;
};
OFP_ASSERT（sizeof（p_bundle_prop_time的结构）== 24）;

/ *捆绑控制消息类型* /
```



```
of_pundle_ctrl_type的枚举{
    OFPBCT_OPEN_REQUEST = 0,
    OFPBCT_OPEN_REPLY,
    OFPBCT_CLOSE_REQUEST,
    OFPBCT_CLOSE_REPLY,
    OFPBCT_COMMIT_REQUEST,
    OFPBCT_COMMIT_REPLY,
    OFPBCT_DISCARD_REQUEST = 6
    OFPBCT_DISCARD_REPLY
};

/ *捆绑包配置标志。* /
of_pundle_flags枚举{
    OFPBF_ATOMIC = 1 << 0，/ *自动执行。* /
    OFPBF_ORDERED = 1 << 1，/ *按指定顺序执行。* /
    OFPBF_TIME = 1 << 2，/ *在指定的时间执行。* /
};

/ * OFPT_BUNDLE_CONTROL的消息结构。* /
struct ofp_bundle_ctrl_msg {
    p_header结构        标题
    uint32_t            bundle_id;      / *标识捆绑包。* /
    uint16_t            类型;           / * OFPBCT_ *。* /
    uint16_t            标志;           / * OFPBF_ *标志的位图。* /

    / *捆绑包属性列表。* /
    p_bundle_prop_header属性的结构[0]; / *零个或多个属性。* /
};
OFP_ASSERT（sizeof（p_bundle_ctrl_msg的结构）== 16）;

/ * OFPT_BUNDLE_ADD_MESSAGE的消息结构。
 *完成捆绑中添加消息的操作。* /
struct ofp_bundle_add_msg {
    p_header结构        标题
    uint32_t            bundle_id;      / *标识捆绑包。* /
    uint16_t            垫;             / *对齐64位。* /
    uint16_t            标志;           / * OFPBF_ *标志的位图。* /

    p_header结构        信息;           / *消息已添加到分发包中。* /

    / *如果存在一个或多个属性，则“消息”后跟:
     *    -准确（message.length + 7）/ 8 * 8-（message.length）（0到7之间）
     *       全零字节的字节* /

    / *捆绑包属性列表。* /
    / * p_bundle_prop_header属性的结构[0]; * / / *零或多个
                                                  属性。* /
};
OFP_ASSERT（sizeof（struct ofp_bundle_add_msg）== 24）;
#endif / * openflow / openflow.h * /
```

附录B发行说明

本部分包含发行说明，重点介绍了主要版本之间的主要更改。
OpenFlow协议。

发行说明的文本内容丰富，具有历史意义，不应视为规范性文件。
发行说明中的许多内容涉及已被删除，替换或升级的功能和文本，
在本规范的后续版本中注明日期，因此不一定与实际
规格。

B.1 OpenFlow版本0.2.0

发布日期：2008年3月28日
协议版本：1

B.2 OpenFlow版本0.2.1

发布日期：2008年3月28日
协议版本：1
协议不变。

B.3 OpenFlow版本0.8.0

发布日期：2008年5月5日
协议版本：0x83

- 重新组织OpenFlow消息类型
- 添加OFPP_TABLE虚拟端口以将数据包输出数据包发送到表
- 添加全局标志OFPC_SEND_FLOW_EXP以配置流过期消息的生成
- 添加流优先级
- 删除流组ID（实验性QoS支持）
- 添加错误消息
- 使统计请求和统计回复更加通用，并带有通用标头和特定于统计的正文
- 更改统计信息回复的分段策略，使用显式标志OFPSF_REPLY_MORE代替空包
- 添加表统计信息和端口统计信息

B.4 OpenFlow版本0.8.1

发布日期：2008年5月20日
协议版本：0x83
协议不变。

B.5 OpenFlow版本0.8.2

发布日期：2008年10月17日
协议版本：0x85

- 添加回显请求和回显回复消息
- 使所有消息64位对齐

B.6 OpenFlow版本0.8.9

发布日期：2008年12月2日
协议版本：0x97

B.6.1 IP网络掩码

现在，流条目可以包含IP子网掩码。这是通过更改通配符来完成的
字段，已扩展为32位：

```
/ *流通配符。* /
of_p_flow_wildcards枚举{
OFPFW_IN_PORT = 1 << 0，/ *开关输入端口。* /
OFPFW_DL_VLAN = 1 << 1，/ * VLAN。* /
OFPFW_DL_SRC    = 1 << 2，/ *以太网源地址。* /
OFPFW_DL_DST    = 1 << 3，/ *以太网目标地址。* /
OFPFW_DL_TYPE = 1 << 4，/ *以太网帧类型。* /
OFPFW_NW_PROTO = 1 << 5，/ * IP协议。* /
OFPFW_TP_SRC    = 1 << 6，/ * TCP / UDP源端口。* /
OFPFW_TP_DST    = 1 << 7，/ * TCP / UDP目标端口。* /
```

```
/ * IP源地址通配符位数。0是完全匹配，32和更高的通配符
  *整个领域。这与通常的约定相反
  *例如/ 24表示通配符为8位（不是24位）。* /
OFPFW_NW_SRC_SHIFT = 8
OFPFW_NW_SRC_BITS = 6
OFPFW_NW_SRC_MASK =（（1 << OFPFW_NW_SRC_BITS）-1）<< OFPFW_NW_SRC_SHIFT，
OFPFW_NW_SRC_ALL = 32 << OFPFW_NW_SRC_SHIFT，

/ * IP目标地址通配符位数。与源格式相同。* /
OFPFW_NW_DST_SHIFT = 14
OFPFW_NW_DST_BITS = 6
OFPFW_NW_DST_MASK =（（（1 << OFPFW_NW_DST_BITS）-1）<< OFPFW_NW_DST_SHIFT，
OFPFW_NW_DST_ALL = 32 << OFPFW_NW_DST_SHIFT，

/ *通配所有字段。* /
OFPFW_ALL =（（1 << 20）-1）
};
```

源和目标网络掩码在通配符描述中均用6位数字指定。
它的解释类似于CIDR后缀，但含义相反，因为它被用于
指出应将IP地址中的哪些位视为“通配符”。例如，CIDR后缀为“ 24”
表示使用“ 255.255.255.0”的网络掩码。但是，通配符掩码值“ 24”表示
最低有效的24位是百搭的，因此形成的网络掩码为“ 255.0.0.0”。

B.6.2新的物理端口统计

ofp_port_stats消息已扩展为返回更多信息。如果没有开关
支持特定字段，则应将值设置为启用所有位（例如，如果该值是“ -1”
视为已签名）。这是新格式：

```
/ *对OFPST_PORT请求的回复正文。如果不支持计数器，请设置
  *该字段适用于所有人。* /
struct ofp_port_stats {
        uint16_t port_no;
```



```
        uint8_t pad [6];                / *对齐64位。* /
        uint64_t rx_packets;            / *接收到的数据包数。* /
        uint64_t tx_packets;            / *传输的数据包数。* /
        uint64_t rx_bytes;              / *接收的字节数。* /
        uint64_t tx_bytes;              / *传输的字节数。* /
        uint64_t rx_dropped;            / * RX丢弃的数据包数。* /
        uint64_t tx_dropped;            / * TX丢弃的数据包数。* /
        uint64_t rx_errors;             / *接收错误数。这是一个超集
                                            接收错误，应大于或
                                            等于al rx _ * _ err值的总和。* /
        uint64_t tx_errors;             / *传输错误数。这是一个超集
                                            传输错误。* /
        uint64_t rx_frame_err;          / *帧对齐错误数。* /
        uint64_t rx_over_err;           / * RX溢出的数据包数。* /
        uint64_t rx_crc_err;            / * CRC错误数。* /
        uint64_t碰撞；                  / *碰撞次数。* /
};
```

B.6.3 IN PORT虚拟端口

在规范的早期版本中，没有明确定义发送入端口的行为。
化。现在禁止使用，除非将输出端口显式设置为OFPP_IN_PORT虚拟端口
（0xfff8）已设置。主要用于无线链接，通过该链接接收数据包
无线接口，并且需要通过同一接口发送到另一台主机。例如，
如果需要将数据包发送到交换机上的所有接口，则需要指定两个操作：
“ actions = output：ALL，output：IN PORT”。

B.6.4端口和链接状态和配置

交换机应将端口和链接状态的更改通知控制器。这是用新的
在ofp_port_config中标记：

- •OFPPC_PORT_DOWN-端口已配置为“关闭”。

...以及在ofp_port_state中的新标志：

- •OFPPS_LINK_DOWN-不存在物理链接。

交换机应通过修改OFPPFL_PORT_DOWN支持启用和禁用物理端口
ofp_port_mod消息中的标志（和掩码位）。请注意，这是不一样的添加或删除
OpenFlow监视端口列表中的接口；它等效于“ ifconfig eth0 down”
Unix系统。

B.6.5回声请求/回复消息

交换机和控制器可以使用新的OpenFlow协议验证正确的连接
回显请求（OFPT_ECHO_REQUEST）并回复（OFPT_ECHO_REPLY）消息。邮件正文
是未定义的，仅包含未解释的数据，这些数据将被回显给请求者。的
请求者将回复与OpenFlow标头中的交易ID相匹配。

B.6.6供应商扩展

供应商现在可以添加自己的扩展，同时仍符合OpenFlow。首要的
做到这一点的方法是使用新的OFPT_VENDOR消息类型。消息正文的形式为：

```
/ *供应商扩展。* /
struct ofp_vendor {
        struct ofp_header标头；                 / *输入OFPT_VENDOR。* /
        uint32_t供应商；                        / *供应商ID：
                                                  *-MSB 0：低位字节是IEEE OUI。
                                                  *-MSB！= 0：由OpenFlow定义
                                                  *       财团。* /
        / *供应商定义的任意附加数据。* /
};
```

的*供应商*字段是一个32位值，该值唯一地标识供应商。如果最高有效字节为零，
接下来的三个字节是供应商的IEEE OUI。如果供应商没有（或希望使用）他们的OUI，
他们应联系OpenFlow联盟以获取一个。身体的其余部分未被解释。

也可以使用OFPST_VENDOR统计类型为统计消息添加供应商扩展。的
如前所述，消息的前四个字节是供应商标识符。身体的其余部分是
供应商定义的。

为了表明交换机不理解供应商扩展，OFPBRC_BAD_VENDOR错误代码
已在OFPET_BAD_REQUEST错误类型下定义。

供应商定义的操作在下面的“可变长度和供应商操作”部分中进行了描述。

B.6.7 IP片段的显式处理

在规范的先前版本中，IP片段的处理没有明确定义。开关
现在可以告诉控制器是否能够重组片段。这是通过
在ofp_switch功能消息中传递的以下功能标志：

```
OFPC_IP_REASM       = 1 << 5 / *可以重组IP片段。* /
```

控制器可以通过以下新设置来在交换机中配置片段处理
ofp_switch_config消息中的ofp_config_flags：

```
/ *处理IP片段。* /
OFPC_FRAG_NORMAL  = 0 << 1，/ *对片段没有特殊处理。* /
OFPC_FRAG_DROP    = 1 << 1，/ *丢弃片段。* /
OFPC_FRAG_REASM   = 2 << 1，/ *重新组装（仅在设置了OFPC_IP_REASM的情况下）。* /
OFPC_FRAG_MASK    = 3 << 1
```

片段的“正常”处理意味着应该尝试将片段传递通过
OpenFlow表。如果不存在任何字段（例如，TCP / UDP端口不适合），则该数据包
不应与设置该字段的任何条目匹配。

B.6.8 802.1D生成树

OpenFlow现在具有一种配置和查看802.1D Spanning的交换机上实施结果的方法
树协议。

实现STP的交换机必须在其“功能”字段中设置新的OFPC_STP位

OFPT_FEATURES_REPLY消息。完全实现STP的交换机必须使其在所有
其物理端口，但不必在虚拟端口（例如OFPP_LOCAL）上实现。

几个端口配置标志与STP相关联。完整的端口配置标志集
是：

```
of_p_port_config的枚举{
    OFPPC_PORT_DOWN     = 1 << 0，/ *端口在管理上已关闭。* /
    OFPPC_NO_STP        = 1 << 1，/ *禁用端口上的802.1D生成树。* /
    OFPPC_NO_RECV       = 1 << 2，/ *丢弃端口上收到的大多数数据包。* /
    OFPPC_NO_RECV_STP = 1 << 3，/ *丢弃收到的802.1D STP数据包。* /
    OFPPC_NO_FLOOD      = 1 << 4，/ *泛洪时不要包括此端口。* /
    OFPPC_NO_FWD        = 1 << 5，/ *丢弃转发到端口的数据包。* /
    OFPPC_NO_PACKET_IN = 1 << 6         / *不要为端口发送输入包的消息。* /
};
```

控制器可以将OFPPFL_NO_STP设置为0以启用端口上的STP或设置为1以禁用端口上的STP。
（后者对应于“禁用的STP”端口状态。）默认值是交换机实现定义的；默认值是“交换机实现定义的”。
默认情况下，OpenFlow参考实现将此位设置为0（启用STP）。

当OFPPFL_NO_STP为0时，STP直接控制OFPPFL_NO_FLOOD和OFPPFL_STP_ *位。

当STP端口状态为“转发”时，OFPPFL_NO_FLOOD设置为0，否则设置为1。
根据当前STP端口将OFPPFL_STP_MASK设置为其他OFPPFL_STP_ *值之一
州。

当STP更改了端口标志时，交换机会将OFPT_PORT_STATUS消息发送给no-
确定更改的控制器。OFPPFL_NO_RECV，OFPPFL_NO_RECV_STP，OFPPFL_NO_FWD和

OpenFlow端口标志中的OFPPFL_NO_PACKET_IN位可能对控制器实现有用
STP，尽管它们与带内控制的交互性较差。

B.6.9修改现有流条目中的操作

添加了新的ofp_flow_mod命令以支持修改现有条目的操作：

OFPFC_MODIFY和OFPFC_MODIFY_STRICT。他们使用匹配字段来描述应输入的条目
通过提供的操作进行修改。OFPFC_MODIFY与OFPFC_DELETE相似，因为通配符是
“活性”。OFPFC_MODIFY_STRICT与OFPFC_DELETE_STRICT相似，因为通配符不是“活动的”，
因此，通配符和优先级都必须与条目匹配。找到匹配的流程时，只有其动作
已修改–计数器和计时器等信息未重置。

如果控制器使用OFPFC_ADD命令添加已经存在的条目，则新条目
替换旧的并且重置所有计数器和计时器。

B.6.10更灵活的表说明

以前版本的OpenFlow具有非常有限的功能来描述交换机支持的表。
ofp_switch_features中的n_exact，n_compression和n_general字段已替换为

n_tables，列出交换机中的表数。

OFPST_TABLE状态回复的行为已稍作修改。现在的ofp_table_stats主体
包含通配符字段，该字段指示特定表支持通配符的字段。
例如，直接查找哈希表会将该字段设置为零，而依次搜索
表会将其设置为OFPFW_ALL。ofp_table_stats条目按以下顺序返回：
数据包遍历表。

当控制器和交换机首次通信时，控制器将找出
从功能回复中切换支持。如果希望了解其尺寸，类型和顺序

查阅表后，控制器将发送OFPST_TABLE统计信息请求。

B.6.11表中的查找计数

返回了p_table_stats结构的表统计信息现在将返回具有
在桌子上抬起头来（无论他们是否命中）。这存储在lookup_count字段中。

B.6.12修改Port-Mod中的标志更加明确

ofp_port_mod用于修改交换机端口的特征。提供的p_phy_port
结构通过其标志字段描述了交换机的行为。但是，
控制器希望更改特定标志，并且可能不知道所有标志的当前状态。一张面具
字段已添加，该字段为应在交换机上更改的每个标志设置了位。

新的ofp_port_mod消息如下所示：

```
/ *修改物理端口的行为* /
struct ofp_port_mod {
        struct ofp_header标头；
        uint32_t掩码；                  / *应该为“ ofp_port_flags”的位图
                                          改变了。* /
        struct ofp_phy_port desc;
};
```

B.6.13新的Packet-Out消息格式

先前版本的数据包输出消息根据对可变长度数组的不同对待
是否设置了buffer_id。如果设置，则该数组由要执行的操作和

out_port被忽略。如果不是，则该数组由应放置在
通过out_port接口连接。这有点丑陋，这意味着为了进行非缓冲
数据包以对其执行多个操作，即创建一个新的流条目以使其与之匹配
条目。

现在使用了一种新格式，可以将消息清除一些。数据包始终包含以下内容的列表：
动作。在动作列表之后还有一个附加的可变长度数组，其中包含数据包的内容
如果未设置buffer_id。这是新格式：

```
struct ofp_packet_out {
        struct ofp_header标头；
        uint32_t buffer_id;                     / *由数据路径分配的ID（如果没有则为-1）。* /
        uint16_t in_port;                       / *数据包的输入端口（如果没有，则为OFPP_NONE）。* /
        uint16_t n_actions;                     / *操作数。* /
        p_action操作的结构[0]; / *动作。* /
        / * uint8_t数据[0]; * /                 / *数据包数据。长度推断
                                                  从标题的长度字段开始。
                                                  （仅在buffer_id == -1时有意义。）* /
};
```

B.6.14流条目的硬超时

硬超时值已添加到流条目。如果设置，则该条目必须在
指定的秒数，无论数据包是否命中该条目。一次硬超时
字段已添加到flow_mod消息中以支持此操作。max_idle字段已重命名

空闲超时。零值表示尚未设置超时。如果idle_timeout和
hard_timeout为零，则该流是永久流，如果没有显式，则不应将其删除
删除。

新的ofp_flow_mod格式如下所示：

```
struct ofp_flow_mod {
        struct ofp_header标头；
        struct ofp_match match;                 / *要匹配的字段* /

        / *流动作。* /
        uint16_t命令；                          / * OFPFC_ *之一。* /
        uint16_t idle_timeout;                  / *丢弃前的空闲时间（秒）。* /
        uint16_t hard_timeout;                  / *丢弃前的最长时间（秒）。* /
        uint16_t优先级；                        / *流条目的优先级。* /
```

```
    uint32_t buffer_id;                 / *要应用于的缓冲数据包（或-1）。
                                          对于OFPFC_DELETE *没有意义。* /
    uint32_t保留;                       / *保留供将来使用。* /
    p_action操作的结构[0]; / *从中推断出操作数
                                          标头中的长度字段。* /
};
```

由于流条目现在可以由于空闲或硬超时而过期，因此将原因字段添加到了
ofp_flow_expired消息。值为0表示空闲超时，值为1表示硬超时：

```
of_p_flow_expired_reason枚举{
    OFPER_IDLE_TIMEOUT，          / *流空闲时间超过了idle_timeout。* /
    OFPER_HARD_TIMEOUT            / *时间超出了hard_timeout。* /
};
```

新的ofp_flow_expired消息如下所示：


```
struct ofp_flow_expired {
    struct ofp_header标头;
    struct ofp_match match;         / *字段说明* /

    uint16_t优先级;                 / *流条目的优先级。* /
    uint8_t原因;                    / * OFPER_ *之一。* /
    uint8_t pad [1];                / *对齐32位。* /

    uint32_t持续时间;               / *时间流以秒为单位。* /
    uint8_t pad2 [4];               / *对齐64位。* /
    uint64_t packet_count;
    uint64_t byte_count;
};
```

B.6.15重新设计了初始握手以支持向后兼容

OpenFlow现在包括基本的“版本协商”功能。当一个OpenFlow连接是
建立后，连接的每一端都应立即发送OFPT_HELLO消息作为其第一个消息
OpenFlow消息。您好消息中的“版本”字段应为最高的OpenFlow协议
发送者支持的版本。收到此消息后，收件人可以计算OpenFlow
协议版本，用作发送的版本号和发送的版本号中的较小者
收到。

如果收件人支持协商的版本，则连接继续。否则，
收件人必须回复OFPT_ERROR消息，其“类型”值为OFPET_HELLO_FAILED，
OFPHFC_COMPATIBLE的“代码”，以及可选的ASCII字符串，用于解释“数据”中的情况，以及
然后终止连接。

OFPT_HELLO消息没有正文；也就是说，它仅包含一个OpenFlow标头。实作
必须准备好接收包含正文的hello消息，忽略其内容，以允许
以后的扩展。

B.6.16开关状态描述

添加了OFPST_DESC状态，以描述在交换机上运行的硬件和软件：

```
＃定义DESC_STR_LEN          256
#define SERIAL_NUM_LEN 32
/ *对OFPST_DESC请求的回复正文。每个条目都是以NULL终止的
  * ASCII字符串。* /
struct ofp_desc_stats {
    字符mfr_desc [DESC_STR_LEN];        / *制造商说明。* /
    字符hw_desc [DESC_STR_LEN];         / *硬件描述。* /
    char sw_desc [DESC_STR_LEN];        / *软件说明。* /
    字符serial_num [SERIAL_NUM_LEN];    /* 序列号。* /
};
```

它包含制造商，硬件类型和软件版本的256个字符的ASCII描述。
它还包含一个32个字符的ASCII序列号。每个条目在右侧填充0个字节。

B.6.17可变长度和供应商操作

供应商定义的操作已添加到OpenFlow。为了实现更多功能，已更改了动作
从固定长度到可变长度。所有操作均具有以下标头：

```
struct ofp_action_header {
    uint16_t类型;                    / * OFPAT_ *之一。* /
    uint16_t len;                    / *动作时长，包括此
                                       标头。这是行动的时长，
                                       包括任何填充物
                                       64位对齐。* /
    uint8_t pad [4];
};
```

动作的长度必须始终是八的倍数，以帮助进行64位对齐。动作类型
如下面所述：

```
of_p_action_type的枚举{
    OFPAT_OUTPUT，          / *输出到交换机端口。* /
    OFPAT_SET_VLAN_VID，    / *设置802.1q VLAN ID。* /
    OFPAT_SET_VLAN_PCP，    / *设置802.1q优先级。* /
    OFPAT_STRIP_VLAN，      / *剥离802.1q标头。* /
    OFPAT_SET_DL_SRC，      / *以太网源地址。* /
    OFPAT_SET_DL_DST，      / *以太网目标地址。* /
    OFPAT_SET_NW_SRC，      / * IP源地址。* /
    OFPAT_SET_NW_DST，      / * IP目标地址。* /
    OFPAT_SET_TP_SRC，      / * TCP / UDP源端口。* /
    OFPAT_SET_TP_DST，      / * TCP / UDP目标端口。* /
    OFPAT_VENDOR = 0xffff
};
```

供应商定义的操作标头如下所示：

```
struct ofp_action_vendor_header {
    uint16_t类型;                    / * OFPAT_VENDOR。* /
    uint16_t len;                    / *长度为8。* /
    uint32_t供应商；                 / *供应商ID，格式相同
                                       就像“ struct ofp_vendor”中一样。* /
};
```

供应商字段使用前面“供应商扩展”部分中所述的相同供应商标识符。
除了使用ofp_action_vendor标头和64位对齐要求之外，供应商都是免费的
使用他们喜欢的信息的任何内容。

B.6.18 VLAN操作更改

现在可以在VLAN标签中设置优先级字段，剥离VLAN标签现在是单独的
行动。OFPAT_SET_VLAN_VID操作的行为类似于以前的OFPAT_SET_DL_VLAN操作，但没有
不再接受一个特殊的值，该值将导致其剥离VLAN标签。OFPAT_SET_VLAN_PCP操作
修改VLAN标记中的3位优先级字段。对于现有标签，这两个操作仅修改位
与要更新的字段相关联。如果需要添加新的VLAN标记，则所有其他值
字段为零。

OFPAT_SET_VLAN_VID操作如下所示：

```
struct ofp_action_vlan_vid {
    uint16_t类型;                    / * OFPAT_SET_VLAN_VID。* /
    uint16_t len;                    / *长度为8。* /
    uint16_t vlan_vid;               / * VLAN ID。* /
    uint8_t pad [2];
};
```

OFPAT_SET_VLAN_PCP操作如下所示：

```
struct ofp_action_vlan_pcp {
    uint16_t类型;                    / * OFPAT_SET_VLAN_PCP。* /
    uint16_t len;                    / *长度为8。* /
    uint8_t vlan_pcp;                / * VLAN优先级。* /
    uint8_t pad [3];
};
```

OFPAT_STRIP_VLAN操作不带任何参数，如果存在VLAN标签，则剥离VLAN标签。

B.6.19最大支持的端口设置为65280

内容：增加最大数量的端口以支持大型供应商交换机；以前是256，已选择
任意地。

原因： HP 5412机箱支持288个以太网端口，并且某些Cisco交换机的性能更高。
当前限制（OFPP_MAX）为255，设置为等于以下网桥段中的最大端口数
1998 STP规范。2004年的RSTP规范最多支持4096（12位）端口。

如何：将OFPP_MAX更改为65280。（但是，开箱即用，引用开关实现
最多支持256个端口。）

B.6.20由于全表未添加流时发送错误消息

现在，当添加流时，该开关会发送一条错误消息，但由于所有表都已满，因此无法发送。
该消息的错误类型为OFPET_FLOW_MOD_FAILED，代码为OFPFMFC_ALL_TABLES_FULL。如果
Flow-Mod命令引用一个缓冲的数据包，则不对该数据包执行操作。如果
控制器希望发送数据包，而不管是否添加了流条目，然后
应该直接使用Packet-Out。

B.6.21控制器连接断开时的行为

内容：确保控制器连接为时，所有交换机至少具有一种共同的行为
丢失。

原因：当与控制器的连接丢失时，交换机应以明确定义的方式运行。
合理的行为包括“不执行任何操作-让流自然超时”，“冻结超时”，“变得
学习开关”和“尝试连接到其他控制器”。交换机可以实现一个或多个
其中，网络管理员可能希望确保如果控制器出故障，他们知道
网络可以做到。

第一个是最简单的：确保每个交换机都实现默认的“不执行任何操作-让流程超时
自然'。更改必须通过供应商特定的命令行界面或供应商扩展进行
OpenFlow消息。

第二个可能有助于确保单个控制器可以与来自多个供应商的交换机一起使用。的
为交换机设置的功能位可能包括不同的故障行为以及“其他”。开关仍然会
只需要支持默认值即可。

这里担心的是，我们可能无法提前枚举全部故障行为，
主张第一种方法。

如何：在规格中添加了文字：“如果交换机与控制器失去联系，默认
行为必须是什么都不做-让流量自然超时。可以实现其他行为
通过特定于供应商的命令行界面或供应商扩展OpenFlow消息。”

B.6.22 ICMP类型和代码字段现在可以匹配

内容：允许根据类型或代码匹配ICMP流量。

原因：我们无法区分不同类型的ICMP流量（例如，回声回复与回声请求
与重定向）。

作法：更改规格以允许在这些字段上进行匹配。
至于实现：类型和代码都是单个字节，因此它们很容易适合我们现有的流程
结构体。将tp_src字段重载为ICMP类型，并将tp_dst重载为ICMP代码。由于它们只是一个
单字节，它们将驻留在这两个字节字段的低字节中（以网络字节顺序存储）。这个
将允许控制器使用现有的通配符位对这些ICMP字段进行通配。

B.6.23用于删除*，流统计信息和聚合统计信息的输出端口过滤

添加对基于输出端口列出和删除条目的支持。

为此，outpport字段已添加到ofp_flow_mod，ofp_flow_stats_request，
和ofp_aggregate_stats_request消息。如果out_port包含非OFPP_NONE的值，
匹配时引入了约束。此约束是规则必须包含输出操作
直接指向那个港口。仍然使用其他约束，例如ofp_match结构和优先级。这是
纯粹是一个*附加*约束。注意，要获得以前的行为，必须设置out_port

到OFPP_NONE，因为“ 0”是有效的端口ID。这仅适用于delete和delete_strict流
mod命令; 该字段会被添加，修改和Modify_strict忽略。

B.7 OpenFlow版本0.9

发布日期：2009年7月20日
协议版本：0x98

B.7.1故障转移

现在，参考实现包括一个简单的故障转移机制。可以配置开关
与控制器列表。如果第一个控制器发生故障，它将自动切换到第二个
列表上的控制器。

B.7.2紧急流缓存

协议和参考实现已扩展为允许插入和管理
紧急流量条目。

在交换机断开与控制器的连接之前，紧急特定流条目将处于非活动状态。如果这
发生这种情况时，交换机会使所有正常流表条目无效，并将所有紧急流复制到
正常流量表。

再次连接到控制器后，流缓存中的所有条目均保持活动状态。然后，控制器具有
如果需要，可以选择重置流缓存。

B.7.3屏障命令

屏障命令是一种在OpenFlow消息完成执行时得到通知的机制
在开关上。交换机收到屏障消息后，必须先完成所有发送的命令，然后再发送
在执行任何命令之前的屏障消息。当所有命令都在屏障之前
消息已完成，它必须将屏障回复消息发送回控制器。

B.7.4 VLAN优先位匹配

有一个可选的新功能，可以匹配优先级VLAN字段。0.9以下的VLAN ID
是用于标识流的字段，但VLAN标记中的优先级不是。在此版本中
我们将优先级位作为一个单独的字段来标识流。可以进行精确匹配
在3个优先级位上匹配，或者在整个3位上作为通配符。

B.7.5选择性流到期

现在可以按流而不是按交换机的粒度请求流到期消息。

B.7.6 Flow Mod行为

现在有一个CHECK_OVERLAP标志用于流mod，这需要开关执行（可能更多）
费用高昂），检查是否存在优先级相同的冲突流程。如果有一个，
mod失败，并返回错误代码。OpenFlow交换机中需要支持此标志。

B.7.7流到期持续时间

Flow Expiration消息中“ duration”字段的含义已稍有更改。上一个
规范中对此的定义存在矛盾。在0.9中，返回的值就是时间
该流处于活动状态，并且不包括超时时间。

B.7.8流删除通知

如果控制器删除流，则如果通知位已设置，则控制器现在会收到通知。在以前
仅释放流到期，但不删除操作将触发通知。

B.7.9在IP ToS标头中重写DSCP

现在，添加了一个Flow操作来重写IP ToS字段中的DiffServ CodePoint位部分。
IP标头。这可以在某些交换机中使用OpenFlow对基本QoS提供基本支持。更多
计划为将来的OpenFlow版本提供完整的QoS框架。

B.7.10端口枚举现在从1开始

先前版本的OpenFlow的端口号从0开始，而版本0.9则将其端口号从1开始。

B.7.11规范的其他变更

- 6633 / TCP现在是推荐的默认OpenFlow端口。长期目标是获得IANA
  批准的OpenFlow端口。
- 不建议使用“类型1”和“类型0”，并删除了对其的引用。
- 明确了流量修改和统计信息的匹配行为
- 明确指出在通过生成树禁用的端口上收到的数据包必须遵循
  正常流表处理路径。
- 说明标头中的交易ID应该与OFPET_BAD_REQUEST的违规消息匹配，
  OFPET_BAD_ACTION，OFPET_FLOW_MOD_FAILED。
- 阐明了Strip VLAN操作的格式
- 阐明交换机等待回复时在交换机上缓冲的数据包的行为
  来自控制器
- 添加了新的EPERM错误类型
- 固定流程图匹配图
- 澄清的数据路径ID从48位增加到64位
- 阐明了输出动作的未发送镜头和最大镜头

B.8 OpenFlow版本1.0

发布日期：2009年12月31日
协议版本：0x01

B.8.1切片

OpenFlow现在在每个输出端口上支持多个队列。队列支持提供以下功能：
最低带宽保证；分配给每个队列的带宽是可配置的。切片名称

从向每个队列提供可用网络带宽的一部分的能力派生而来。

B.8.2流cookie

流已扩展为包括不透明标识符，称为cookie。cookie被指定
安装流程时由控制器提供；Cookie将作为每个流量统计信息的一部分返回，并且
流过期消息。

B.8.3用户指定的数据路径描述

OFPST DESC（交换机描述）回复现在包括一个数据路径描述字段。这是一个用户-
可指定的字段，该字段允许交换机返回由交换机所有者指定的字符串，以描述
开关。

B.8.4 ARP报文中IP字段的匹配

现在，参考实现可以在ARP数据包内的IP字段上进行匹配。来源和目的
着色协议地址分别映射到nw src和nw dst字段，并且操作码为
映射到nw proto字段。

B.8.5 IP ToS / DSCP位匹配

OpenFlow现在支持IP ToS / DSCP位的匹配。

B.8.6查询单个端口的端口统计信息

端口统计信息请求消息包括port_no字段，以允许查询各个端口的统计信息。
通过将OFPP_NONE指定为端口号，仍可以请求所有端口的端口统计信息。

B.8.7改进了统计/到期消息中的流持续时间分辨率

统计信息和到期消息中的流持续时间现在以纳秒分辨率表示。注意
在参考实现中，流持续时间的精度大约为毫秒。（
实际精度部分取决于内核参数。）

B.8.8规范的其他变更

- •删除multi_phy_tx规范文本和功能位
- •明确动作的执行顺序
- •用TLS替换SSL参考
- •解决重叠歧义
- •明确流模式到不存在的端口
- •明确端口定义
- •更新数据包流程图
- •更新ICMP的标头解析图
- •修复了英语中英语流歧义的问题
- •修复异步消息的英语歧义
- •请注意，未定义多控制器支持
- •阐明字节等于八位字节
- •便条纸环绕
- •删除了警告，请不要按照此规范构建开关

B.9 OpenFlow版本1.1

发布日期：2011年2月28日
协议版本：0x02

B.9.1多个表

OpenFlow规范的先前版本确实向控制器公开了单个的抽象
表。OpenFlow管道可以在内部映射到多个表，例如具有单独的表
通配符和完全匹配表，但这些表在逻辑上始终充当单个表。

OpenFlow 1.1引入了具有多个表的更灵活的管道。暴露多个表有
许多优点。第一个优点是很多硬件内部都有多个表（对于
示例L2表，L3表，多个TCAM查找）以及OpenFlow的多个表支持
可能允许以更高的效率和灵活性公开此硬件。第二个优点是
许多网络部署结合了数据包的正交处理（例如ACL，QoS和
路由），由于叉积，迫使所有这些处理都放在一个表中会创建巨大的规则集
个别规则。多个表有助于解耦单独的处理功能并进行映射
有效地利用各种硬件资源。

具有多个表的新OpenFlow管道与之前的简单管道有很大不同
OpenFlow版本。新的OpenFlow管道公开了一组完全通用的表，支持
完整的比赛和完整的动作。很难准确地构建管道抽象

代表所有可能的硬件，因此OpenFlow 1.1基于通用且灵活的管道，
可能被映射到硬件。一些有限的表功能可用来表示每个
桌子可以支撑。

数据包通过管道进行处理，它们在第一个表中进行匹配和处理，并且可能
在其他表格中进行匹配和处理。在通过管道时，一个数据包与
一个*动作集*，一个累积动作和一个通用*元数据*寄存器。操作集在
管道的末端并应用于数据包。元数据可以匹配并写入每个表
并允许在表之间携带状态。

OpenFlow引入了一个新的协议对象，称为*指令*来控制管道处理。动作
以前版本中直接附加到流程的内容现在封装在指令中，
位置可以在表之间应用这些操作，也可以将它们累积在数据包操作集中。使用说明
也可以更改*元数据*，或将数据包定向到另一个表。

- 交换机现在公开带有多个表的管道
- 流程条目具有控制管道处理的指令
- 控制器可以通过goto指令选择表的数据包遍历
- 可以在表中设置和匹配元数据字段（64位）
- 数据包操作可以合并到数据包操作集中
- 数据包操作集在管道末尾执行
- 可以在表阶段之间应用数据包操作
- 表未命中可以发送给控制器，继续到下一个表或删除
- 基本表功能和配置

B.9.2组

新的组抽象使OpenFlow可以将一组端口表示为单个实体进行转发
包。提供了不同类型的组来表示不同的抽象，例如多播
或多路径。每个组由一组组存储桶组成，每个组存储桶包含一组
转发到端口之前要应用的操作。群组存储桶也可以转发给其他群组，
使组链接在一起。

- 组间接表示一组端口
- 具有4种类型的组的组表：
  - –全部-用于多播和泛洪
  - –选择-用于多路径
  - –间接-简单间接
  - –快速故障转移-使用第一个实时端口
- 小组动作以将流程引导至小组
- 组存储桶包含与单个端口相关的操作

B.9.3标签：MPLS＆VLAN

以前版本的OpenFlow规范仅对VLAN提供有限支持，仅支持单个
语义含糊的VLAN标记级别。新的标记支持具有添加的显式操作，

修改和删除VLAN标记，并且可以支持多个级别的VLAN标记。它还添加了类似的内容
支持MPLS填充头。

- 支持VLAN和QinQ，添加，修改和删除VLAN标头
- 支持MPLS，添加，修改和删除MPLS填充头

B.9.4虚拟端口

OpenFlow规范的先前版本假定OpenFlow交换机的所有端口均为
物理端口。此规范版本增加了对虚拟端口的支持，可以表示
复杂的转发抽象，例如LAG或隧道。

- 使端口号为32位，启用更多端口
- 启用交换机以将虚拟端口提供为OpenFlow端口
- 增强报文输入以报告虚拟和物理端口

B.9.5控制器连接失败

OpenFlow规范的先前版本引入了紧急流缓存作为一种处理方法
失去与控制器的连接。紧急流缓存功能已在此删除
该规范的版本，由于缺乏采用，实现它的复杂性和其他问题
具有特征语义。

该规范版本增加了两个更简单的模式来处理与
控制器。在故障安全模式下，交换机继续以OpenFlow模式运行，直到重新连接
到控制器。在独立故障模式下，交换机将恢复为使用常规处理（以太网
交换）。

- 从规范中删除紧急流缓存
- 连接中断触发器失败安全模式或失败独立模式

B.9.6其他变更

- 从规范中删除特定于802.1d的文本
- Cookie增强建议-用于过滤的cookie掩码
- 设置队列操作（与输出端口操作解除捆绑）
- 可屏蔽的DL和NW地址匹配字段
- 为IPv4和MPLS添加TTL减量，设置和复制操作
- SCTP标头匹配和重写支持
- 设置ECN动作
- 定义消息处理：没有损失，如果没有障碍，可能会重新排序
- 将VENDOR API重命名为EXPERIMENTER API
- 许多其他错误修复，重新措辞和澄清

B.10 OpenFlow 1.2版

发布日期：2011年12月5日
协议版本：0x03

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.10.1可扩展比赛支持

OpenFlow规范的先前版本使用静态固定长度结构来指定p_match，
这会阻止灵活表达匹配项并阻止包含新的匹配项。ofp_match
已更改为TLV结构，称为OpenFlow可扩展匹配（OXM），该结构可显着
增加灵活性。

比赛字段本身已重新组织。在以前的静态结构中，许多字段是
超载; 例如tcp.src_port，udp.src_port和icmp.code使用相同的字段条目。
现在，每个逻辑字段都有自己的唯一类型。

OpenFlow可扩展匹配的功能列表：

- •灵活紧凑的TLV结构，称为OXM（EXT-1）
- •启用灵活的匹配表达和灵活的位掩码（EXT-1）
- •确保匹配一致性的前提系统（EXT-1）
- •为每个比赛字段指定一个唯一的类型，消除重载（EXT-1）
- •修改VLAN匹配以更灵活（EXT-26）
- •添加供应商类别和实验者匹配项（EXT-42）
- •允许开关覆盖匹配要求（EXT-56，EXT-33）

B.10.2可扩展的“设置字段”包重写支持

OpenFlow规范的先前版本使用手工操作来重写标头字段。的
可扩展的set_field操作重用为匹配定义的OXM编码，并允许重写
单个动作（EXT-13）中的任何标题字段。这允许任何新的匹配字段，包括实验者
字段，可以重写。这使规格更整洁，并降低了引入成本
新领域。

- •弃用大多数标头重写操作
- •介绍通用设定场操作（EXT-13）
- •在设定场动作中重复使用匹配TLV结构（OXM）

B.10.3“ packet-in”中的可扩展上下文表达

包入消息确实包括一些包上下文（入口），但不是全部（元数据），
阻止控制器确定表中匹配如何发生以及哪些流条目
会匹配还是不匹配。与其在打包消息中引入硬编码字段，不如
灵活的OXM编码用于承载数据包上下文。

- •重用匹配TLV结构（OXM）来描述包入（EXT-6）中的元数据

- •在封包中包含“元数据”字段
- •将入口端口和物理端口从静态字段移动到OXM编码
- •允许在TLV结构中选择性地包括数据包头字段

B.10.4通过实验者错误类型的可扩展错误消息

添加了实验者错误代码，使实验者功能可以生成自定义
错误消息（EXT-2）。格式与其他实验者API相同。

B.10.5支持IPv6

通过灵活匹配支持，已添加了对IPv6匹配和标头重写的基本支持。

- •新增了对IPv6源地址，目标地址，协议号，流量进行匹配的支持
  类，ICMPv6类型，ICMPv6代码和IPv6邻居发现标头字段（EXT-1）
- •增加了对IPv6流标签匹配的支持（EXT-36）

B.10.6 flow-mod请求的简化行为

flow-mod请求的行为已得到简化（EXT-30）。

- •MODIFY和MODIFY STRICT命令从不在表中插入新流
- •新标志OFPFF RESET COUNTS用于控制计数器复位
- •删除Cookie字段的古怪行为。

B.10.7删除了数据包解析规范

OpenFlow规范不再尝试定义如何解析数据包（EXT-3）。比赛
字段仅在逻辑上定义。

- •OpenFlow不要求如何解析数据包
- •通过OXM前提条件实现的解析一致性

B.10.8控制器角色更改机制

控制器角色更改机制是一种简单的机制，可支持多个控制器进行故障转移
（EXT-39）。该方案完全由控制器驱动；开关只需要记住
每个控制器的角色有助于控制器的选举机制。

- •支持多个控制器进行故障转移的简单机制
- •交换机现在可以并行连接到多个控制器
- •使每个控制器可以将其角色更改为相等，为主或从属

B.10.9其他变更

- •每表元数据位掩码功能（EXT-34）
- •基本小组功能（EXT-61）
- •在已删除流的消息中添加硬超时信息（OFP-283）
- •增加了控制器检测STP支持的功能（OFP-285）
- •使用OFPCML NO BUFFER（EXT-45）关闭数据包缓冲
- •新增了查询所有队列的功能（EXT-15）
- •添加了实验者队列属性（EXT-16）
- •添加了最大速率队列属性（EXT-21）
- •启用所有表中的删除流程（EXT-10）
- •删除组时启用开关以检查链接（EXT-12）
- •启用控制器以禁用缓冲（EXT-45）
- •将虚拟端口重命名为逻辑端口（EXT-78）
- •新的错误消息（EXT-1，EXT-2，EXT-12，EXT-13，EXT-39，EXT-74和EXT-82）
- •在规范文档中包含发行说明
- •许多其他错误修复，重新措辞和澄清

B.11 OpenFlow版本1.3

发布日期：2012年4月13日
协议版本：0x04

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.11.1重构能力协商

OpenFlow规范的先前版本仅包含对Open-
流量开关。OpenFlow 1.3包括一个更灵活的框架来表达功能（EXT-123）。

主要更改是对表功能的改进描述。这些功能已被移走
表统计结构在其自己的请求/回复消息中，并使用灵活的TLV进行编码
格式。这样就可以添加下一个表功能，表缺失流输入功能以及
实验者能力。

其他更改包括将“统计”框架重命名为“多部分”框架以反映事实
现在将其用于统计和功能，以及将端口描述移至其自身
多部分消息，以支持更多端口。

重构功能协商的功能列表：

- •将“统计”框架重命名为“多部分”框架。
- •启用“多部分”请求（跨多个消息的请求）。
- •将端口列表描述移至其自己的多部分请求/答复。
- •将表功能移至其自己的多部分请求/答复。
- •创建灵活的属性结构来表达表功能。
- •启用以表达实验者的能力。

•添加表缺失流条目的功能。

•添加下一个表（即goto）功能

B.11.2更灵活的表丢失支持

OpenFlow规范的先前版本包括用于选择三个3之一的表配置标志。
处理表缺失的行为（数据包与表中的任何流都不匹配）。OpenFlow 1.3替代
带有表缺失流条目的那些有限标志，一个特殊的流条目描述了表的行为
错过（EXT-108）。

table-miss流条目使用标准的OpenFlow指令和操作来处理table-miss数据包，
这样就可以在处理这些数据包时充分利用OpenFlow的灵活性。所有先前的行为
可以使用table-miss流条目来表示由table-miss config标志表示的内容。许多新
现在可以简单地描述处理表缺失的方式，例如正常处理表缺失
通过OpenFlow协议。

•删除表缺失配置标志（EXT-108）。
•将表缺失流条目定义为所有通配符，最低优先级流条目（EXT-108）。
•强制支持每个表中的表缺失流条目以处理表缺失数据包（EXT-108）。
•添加了描述表缺失流条目（EXT-123）的功能。
•将表丢失默认值更改为丢弃数据包（EXT-119）。

B.11.3 IPv6扩展头处理支持

添加功能以匹配常见的IPv6扩展标头和某些异常
IPv6扩展标头（EXT-38）中的说明。新的OXM伪标头字段OXM_OF_IPV6_EXTHDR
使符合以下条件成为可能：

•存在逐跳IPv6扩展标头。
•存在路由器IPv6扩展标头。
•存在碎片IPv6扩展头。
•存在目标选项IPv6扩展头。
•存在身份验证IPv6扩展头。
•存在加密的安全有效负载IPv6扩展标头。
•没有下一个标头IPv6扩展标头。
•IPv6扩展标头的顺序不正确。
•遇到意外的IPv6扩展头。

B.11.4每流量计

添加对每流量计的支持（EXT-14）。每流量计可以附加到流量条目，并且可以
测量和控制数据包的速率。每流量计的主要应用之一是速率限制
数据包发送到控制器。

每流量计功能基于新的灵活仪表框架，该框架具有以下功能：
通过使用多个计量频段，计量统计数据和功能来描述复杂的计量表。
当前，在此框架上仅定义了简单的限速器。支持色彩意识

支持Diff-Serv风格操作并紧密集成在管道中的仪表被推迟
到更高版本。

•基于每流量计和仪表带的灵活仪表框架。
•仪表统计信息，包括每频段统计信息。
•允许将仪表灵活地连接到流量条目。

•简单的速率限制器支持（丢弃数据包）。

B.11.5每个连接事件过滤

规范的1.2版引入了将交换机连接到多个控制器以实现以下功能的功能：
容错和负载平衡。每个连接事件过滤改善了多控制器支持
通过使每个控制器能够过滤来自其不需要的开关的事件（EXT-120）。

一组新的OpenFlow消息使控制器能够在其自己的连接上配置事件过滤器
到开关。可以按类型和原因过滤异步消息。这个事件过滤器进来了
除了其他启用或禁用异步消息的机制之外，例如
可以为每个流配置流删除事件的生成。每个控制器可以有一个单独的过滤器
从属角色和主/同等角色。

•为每个控制器连接添加异步消息过滤器。
•控制器消息，用于设置/获取异步消息过滤器。
•设置默认过滤器值以匹配OpenFlow 1.2行为。
•删除OFPC_INVALID_TTL_TO_CONTROLLER配置标志。

B.11.6辅助连接

在规范的先前版本中，开关和控制器之间的通道专门
由单个TCP连接组成，不允许利用大多数情况下可用的并行性
切换实现。OpenFlow 1.3使交换机能够创建辅助连接以进行补充
开关和控制器之间的主要连接（EXT-114）。辅助连接主要是
用于携带入站和出站消息。

•启用开关以创建与控制器的辅助连接。
•强制在主连接不活动时不能存在辅助连接。
•在协议中添加辅助ID，以消除连接类型的歧义。
•通过UDP和DTLS启用辅助连接。

B.11.7 MPLS BoS匹配

已添加新的OXM字段OXM_OF_MPLS_BOS，以匹配来自BOM的堆栈底部比特（BoS）。
MPLS标头（EXT-85）。BoS位指示其他MPLS填充标头是否在
当前的MPLS数据包，匹配该位有助于消除MPLS标签为
可在MPLS封装的各个级别重复使用。

B.11.8提供商骨干网桥标记

使用提供者骨干网桥接（PBB）封装（EXT-105）添加对标记数据包的支持。
这使OpenFlow能够支持基于PBB的各种网络部署，例如常规PBB
和PBB-TE。

•推和弹出操作可将PBB标头添加为标签。
•新的OXM字段与PBB标头的I-SID匹配。

B.11.9返工标签顺序

在规范的先前版本中，静态地指定了数据包中标签的最终顺序。对于
例如，始终在数据包中的所有VLAN标记之后插入MPLS填充头。开放流
1.3消除了此限制，数据包中标签的最终顺序由标签的顺序决定
操作，每个标记操作会将其标记添加到最外面的位置（EXT-121）。

•从规范中删除数据包中标签的已定义顺序。
•现在始终将标签添加在最外面的位置。
•操作列表可以按任意顺序添加标签。
•标签顺序是预定义的，用于在操作集中进行标签。

B.11.10隧道ID元数据

逻辑端口抽象使OpenFlow能够支持多种封装。隧道

端口数据(OXM_OF_TUNNEL_ID)是的新的OXM字段，它表示了与逻辑端口关联到OpenFlow管道。最常见的是标识封装的GRE密钥。
标头（EXT-107）。

例如，如果逻辑端口执行GRE封装，则tunnel-id字段将映射到GRE
GRE标头中的关键字段。解封装后，OpenFlow将能够匹配GRE密钥
在tunnel-id匹配字段中。同样，通过设置tunnel-id，OpenFlow可以设置
封装报文中的GRE密钥。

B.11.11打包中的Cookie

Cookie字段已添加到打包消息（EXT-7）。该字段的价值来自于
将数据包发送到控制器。如果数据包不是由流发送的，则此字段设置为0xffffffffffffffff。

将Cookie包含在输入包中，可使控制器更有效地对输入包进行分类
不必将数据包与完整流表进行匹配。

B.11.12统计持续时间

持续时间字段已添加到大多数统计信息中，包括端口统计信息，组统计信息，队列统计信息
和电表统计信息（EXT-102）。持续时间字段允许更准确地计算数据包和字节
这些统计信息中包含的计数器的费率。

B.11.13按需流量计数器

添加了新的flow-mod标志，以逐个流禁用包和字节计数器。禁用
这样的计数器可以改善交换机中的流处理性能。

B.11.14其他变更

- 修复了描述VLAN匹配的错误（EXT-145）。
- 现在，流条目描述提到优先级（EXT-115）。
- 现在，流条目说明提到了超时和cookie（EXT-147）。
- 现在必须将不可用的计数器设置为全1（EXT-130）。
- 正确引用流条目而不是规则（EXT-132）。
- 许多其他错误修复，重新措辞和澄清。

B.12 OpenFlow版本1.3.1

发售日期：2012年9月6日
协议版本：0x04

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.12.1改进的版本协商

OpenFlow规范的先前版本包括用于版本协商，选择的简单方案
双方支持的最高版本中的最低版本。不幸的是，该方案不起作用
在任何情况下都应正确处理; 如果两种实现都没有实现最高版本的所有版本，
该方案可能无法协商它们共有的版本（EXT-157）。

主要更改是在协商期间使用的Hello消息中添加了版本号的位图。
通过拥有完整的版本号列表，协商可以始终协商适当的版本，如果
一个可用。此版本的位图以灵活的TLV格式编码，以保留将来的可扩展性
您好消息。

改进版本协商的功能列表：

- Hello Elements，Hello消息的新的灵活TLV格式
- Hello消息中的可选版本位图。
- 使用可选的版本位图改进版本协商。

B.12.2其他变更

- 要求表缺失流条目支持下降和控制器（EXT-158）。
- 阐明OXM_OF_TUNNEL_ID（EXT-161）中封装数据的映射。
- UDP连接的规则和限制（EXT-162）。

•澄清虚拟仪表（EXT-165）。
•删除有关交换机碎片的参考-令人困惑（EXT-172）。
•将仪表常数名称固定为始终为多部分（OFPST_ => OFPMT_）（EXT-184）。

•在规范（EXT-198）中添加OFPG_ *定义。
•在规范文本中添加了ofp_instruction和ofp_table_feature_prop_header（EXT-200）。
•连接设置中的错误错误代码必须为OFPHFC_INCOMPATIBLE（EXT-201）。
•指令的长度必须是8个字节的倍数（EXT-203）。
•端口状态包括原因而不是状态（EXT-204）。
•澄清表配置字段（EXT-205）的用法。
•说明不需要在每个流表（EXT-206）中都支持必需的匹配字段。
•说明先决条件不需要完全匹配字段支持（EXT-206）。
•在规范中包括openflow.h（EXT-207）缺少的定义。
•修复无效的错误代码OFPQCFC_EPERM-> OFPSCFC_EPERM（EXT-208）。
•阐明有关B-VLAN的PBB语言（EXT-215）
•修复了源和目标以太网地址之间的反转（EXT-215）
•阐明如何重新排序组时段，以及相关的组时段说明（EXT-217）。
•添加免责声明发布说明可能不符合规范（EXT-218）
•图1仍然显示“安全通道”（EXT-222）。
•必须计算OpenFlow版本（EXT-223）。
•仪表带下降优先级应该提高，而不是降低（EXT-225）
•修正可能/可以/应该/必须的模棱两可的用法（EXT-227）
•修正错别字（EXT-228）
•许多错别字（EXT-231）

B.13 OpenFlow版本1.3.2

发布日期：2013年4月25日
协议版本：0x04

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.13.1变更

•在OXM中，要求掩码中的0位必须是值的0位（EXT-238）。
•允许从其中一个控制器（EXT-252）启动连接。
•在规范（EXT-259）上增加关于框架失序的子句。
•设置表功能不会生成流删除的消息（EXT-266）。
•修复了设置表功能错误响应（EXT-267）的描述。
•在角色回复消息（EXT-272）中定义generation_id的使用。
•仅具有一个流表的交换机无权实施goto（EXT-280）。

B.13.2澄清

•明确说明MPLS Pop动作使用Ethertype而不管BOS位（EXT-194）如何。
•使用辅助连接（EXT-240）的控制器消息优先级。
•阐明填充规则和可变大小数组（EXT-251）。
•更好地描述了流程模块（EXT-257）中的buffer-id。
•OFPPS_LIVE（EXT-258）的语义。
•改进多部分介绍（EXT-263）。

•澄清设置表功能说明（EXT-266）。
•澄清仪表标志和脉冲串字段（EXT-270）。
•阐明从属访问权限（EXT-271）。
•说明交换机不能更改控制器角色（EXT-276）。

•阐明基存的主控制器和均等控制器（EXT-277）的角色。
•各种错别字和改写（EXT-282，EXT-288，EXT-290）

B.14 OpenFlow版本1.3.3

发售日期：2013年9月27日
协议版本：0x04

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.14.1变更

•使用IANA注册的TCP端口进行更新：6653（EXT-133）。
•明确说明默认情况下IPv6流标签不可屏蔽（EXT-101）。
•阐明多部分分段规则，明确使用空的多部分消息（EXT-321）。
•指定常规片段处理是强制性的，降落/ reasm可选（EXT-99）。
•说明先决条件是累积性的（EXT-285）。
•指定每个连接的缓冲区ID是唯一的（EXT-286）。
•明确设定场动作（EXT-289）中可以使用哪种OXM类型。
•为表功能中的OXM ID定义oxm_len以具有有效载荷长度（EXT-330）。
•设置域先决条件可以通过其他操作来满足（EXT-331）。
•弄清无效的组类型和无效的重量的错误代码（EXT-344）。
•指定组和仪表功能位图（EXT-345）。

B.14.2澄清

•说明1.3.X（EXT-269）中已弃用OFP_TABLE_MOD。
•稍作澄清，将“转到”替换为“转到表”，将“阅读消息”替换为“多部分”
讯息”（EXT-297）。
•在流条目的描述中提到标记（EXT-298）。
•阐明对控制器（EXT-300）的入站策略。
•澄清无效的DSCP值，所有六个位均有效（EXT-305）。
•在词汇表（EXT-309）中添加了许多新定义。
•改进许多现有的词汇表定义（EXT-309）。
•辅助通道的详细UDP拥塞控制（EXT-311）。
•更好的文档控制器启动的连接（EXT-311）。
•明确每个多部分序列（EXT-321）仅有一个请求/答复。
•澄清辅助连接（EXT-323）上的连接维护消息。
•澄清设置字段和hello元素中的填充（EXT-326）。
•明确包入（EXT-286）中的填充，数据和total_len字段。
•明确表功能中的操作没有填充（EXT-287）。
•在独立故障中，交换机拥有流表和流条目（EXT-291）。

•阐明队列与端口和数据包的关系，并且该队列是可选的（EXT-293）。
•动作设置可能在生成包入（EXT-296）之前执行。
•在表中添加描述OXM类型的字节列（EXT-313）。
•确认OFPP_MAX是可用的端口号（EXT-315）。
•指定如何在UDP（EXT-332）中打包OpenFlow消息。
•Flow-mod修改：替换说明，而不更新（EXT-294）。
•阐明OFPBAC_BAD_TYPE适用于不受支持的操作（EXT-343）。
•在规范（EXT-261）中说明消除流量的原因。
•删除端口不会删除流条目（EXT-281）。
•明确说明设置字段操作（EXT-331）必须包含标题字段。
•澄清按钮式操作（EXT-342）上字段的默认值。
•明确在flow-mods（EXT-354）中使用优先级字段。
•用ONF通用URL（EXT-83 / EXT-356）替换特定于白皮书的URL。
•明确说明并非总是执行操作集（EXT-359）。
•连接设置可能用于带内连接（EXT-359）。
•澄清将组转发到无效组的错误（EXT-359）。
•将“ OpenFlow协议”替换为“ OpenFlow交换协议”（EXT-357）。
•用“协议版本”替换“有线协议”

B.15 OpenFlow版本1.3.4

发售日期：2014年3月27日

协议版本：0x04
有关每个更改的更多详细信息，请参考错误跟踪ID。

B.15.1变更

- •使IPv6流标签可屏蔽（EXT-101）。
- •修改组/仪表时澄清统计信息（EXT-341）。
- •澄清表功能匹配列表不应仅包含前提条件字段（EXT-387）。
- •澄清表功能通配符列表在某些情况下不应包含必填字段
  仅（EXT-387）。
- •添加有关控制通道维护（EXT-435）的部分。
- •Push MPLS应该在IP报头之前和MPLS标签之前而不是在MPLS标签之前添加MPLS报头。
  无效的VLAN（EXT-457）。

B.15.2澄清

- •为电表动作中的电表不良指定错误（EXT-327）。
- •将无效的前缀固定在仪表多部分常数（EXT-302）上。
- •添加有关保留值和保留位位置的部分（EXT-360）。
- •更好地描述协议基本格式（EXT-360）。
- •修复有关实验者乐队类型（EXT-363）的注释。
- •明确说明端口描述仅列出标准端口（EXT-364）。
- •使用OFPFF_RESET_COUNTS（EXT-365）更新流模型描述。

- •澄清流量计的流量计数（EXT-374）。
- •实验者的动作/类型不能在位图（EXT-376）中报告。
- •澄清错误的参数错误操作（EXT-393）。
- •许多小的澄清，实现定义的功能（EXT-395）。
- •明确说明存储桶中的操作始终作为操作集应用（EXT-408）。
- •合并操作集需要在设置字段中识别（EXT-409）。
- •将操作列表更改为一致性操作列表（EXT-409）。
- •在词汇表（EXT-409）中引入正确的操作集。
- •澄清DSCP备注表频段（EXT-416）。
- •添加有关管道一致性（EXT-415）的部分。
- •澄清与匹配或数据包（EXT-417）不一致的动作的处理。
- •阐明港口内和港口内的OXM字段定义（EXT-418）。
- •确认OFPP_CONTROLLER是有效的入口端口（EXT-418）。
- •澄清带遮罩的实验场的流匹配场长度（EXT-420）。
- •澄清对写操作指令或组存储桶中重复操作的处理，允许
  返回错误或过滤重复的操作（EXT-421）。
- •使用非空操作集（EXT-422）澄清“清除操作”指令的错误代码。
- •澄清不支持的OXM_CLASS和OXM_FIELD（EXT-423）上的错误。
- •添加有关保留的属性/ TLV类型（EXT-429）的部分。
- •阐明OFPG_ANY的含义，用于观看组存储区中的组（EXT-431）。
- •将流水线字段定义与标头字段定义（EXT-432）分开。
- •澄清Packet-In OXM列表（EXT-432）中标头字段的存在。
- •当没有待处理的请求（EXT-433）时，必须生成屏障答复。
- •为不支持的操作澄清错误代码（EXT-434）。
- •当不支持或未启用设置表格功能时，请澄清错误代码（EXT-436）。
- •端口说明必须包括所有标准端口，无论配置或状态（EXT-437）。
- •改进通道重新连接建议（EXT-439）。
- •前缀错误，将OFPPFL_NO_PACKET_IN修复为OFPPC_NO_PACKET_IN（EXT-443）。
- •在数据包输入和输出中正确指定数据包数据字段，尤其是CRC（EXT-452）。
- •指定tunnel-id如何与逻辑端口交互，尤其是在输出（EXT-453）中。
- •更详细地描述“隧道ID管道”字段（EXT-453）。
- •各种错别字，语法和拼写修复（EXT-455）。

B.16 OpenFlow版本1.3.5

发售日期：2015年3月26日
协议版本：0x04

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.16.1变更

•带有命令宽松修改和宽松删除的Flow-mod请求不能是局部的（EXT-362）。
•澄清流条目重叠的定义，等于不重叠（EXT-406）。
•添加了备用OpenFlow连接传输的规范（EXT-463）。
•从OF 1.5（EXT-275）添加控制器通道连接URI的规范。

•表号不再是连续的，表仅在潜在的数据包遍历中编号
订购（EXT-467）。
•澄清哪些flow-mod命令和标志是必需的（EXT-478）。
•澄清物理端口是可选的，如果使用，则为OpenFlow端口（EXT-491）。
•指定包含在多部分请求错误中的数据是当前的多部分消息
或为空（EXT-506）。
•指定计数器必须使用整个位范围（EXT-529）。

B.16.2澄清

•控制器必须能够读取Table-miss流条目（EXT-464）。
•阐明包入结构（EXT-468）中total_len的含义。
•允许对设置字段操作（EXT-469）进行延迟的CRC重新计算。
•允许在“设置字段”操作（EXT-471）中使用Experimenter OXM字段。
•保留版本的最高位，而不是实验性（EXT-480）。
•在注释（EXT-481）中用OFPMF修复错误的前缀OFPMC。
•对实验者OXM（EXT-482）的oxm_length更明确。
•带内控制通道不在规格范围内，不再提及默认流（EXT-497）。
•所有流表（EXT-498）中都必须支持必需的说明和操作。
•为错误的queue_id，group_id或meter_id（EXT-500）的多声部指定错误。
•明确表缺失流条目（EXT-509）需要Clear-Actions指令。
•澄清组删除请求（包括存储桶）的错误代码（EXT-510）。
•澄清空的桶或没有桶的组只是丢弃数据包（EXT-511）。
•阐明交换机可以支持优先级为0的流，而不是表丢失流表项（EXT-512）。
•明确障碍答复对应于障碍请求（EXT-513）。
•指定可能出现多个错误代码时，应使用最具体的错误（EXT-
514）。
•指定如果一条消息中有多个错误，则交换机仅返回一条错误消息
（EXT-514）。
•说明所有错误的错误必须具有匹配的xid（EXT-514）。
•阐明OFPC_FRAG_MASK（EXT-517）的定义。
•改进术语表（EXT-518）中的连接定义。
•扩大术语表中的计数器定义（EXT-518）。
•修复各种规格技术错误（EXT-518）。
•纠正打字错误和不正确的英语（EXT-518）。
•修复表和枚举的格式（EXT-518）。
•改进了匹配的通用描述（EXT-519）。
•改进各种描述以使其更加精确（EXT-519）。
•添加交叉引用并修复重复的交叉引用（EXT-519）。
•更好地指定位图和标志（EXT-520）。
•澄清oxm_length（EXT-520）。
•澄清队列属性中的速率字段是指输出端口当前速度的一小部分
（EXT-522）。
•指定建议使用安全版本的TLS（EXT-525）。
•阐明交换机（EXT-304）的证书配置。
•指定格式错误的数据包引用数据路径中的数据包（EXT-528）。

•指定如何处理格式错误的OpenFlow消息（EXT-528）。
•拼写，语法和其他拼写错误（EXT-542）。

B.17 OpenFlow版本1.4.0

发布日期：2013年8月5日
协议版本：0x05

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.17.1更可扩展的有线协议

OpenFlow协议最初设计时具有许多静态固定结构和有限的扩展性，
能力。在1.2版中引入了OpenFlow可扩展匹配（OXM），这增加了很多需求
OpenFlow分类器中的可扩展性。在版本1.4中，该协议的许多其他部分
使用TLV结构进行翻新，以提高可扩展性（EXT-262）。

TLV的这项工作影响了协议的许多领域。先前已修复的新TLV已添加
结构以结构形式结尾处的属性。在某些地区，现有的TLV具有
已更改为使用公共属性TLV格式。TLV的规则已经阐明。这个额外的
协议的可扩展性将提供一种更简便的方法，可在
未来，也大大扩展了Experimenter Extension API。

- 端口结构：添加端口描述属性，添加端口mod属性和添加端口统计信息
  属性。
- 表结构：添加表mod属性，添加表描述多部分，添加表状态
  异步消息。
- 队列结构：将队列描述迁移到多部分，转换队列描述属性
  要标准化TLV，请添加队列统计信息属性。
- 集异步结构：将集异步配置转换为TLV，添加集异步实验者属性。
- 指令结构：阐明指令TLV。
- 行动结构：阐明行动TLV。
- 实验结构：阐明实验者TLV。
- 属性错误：为所有属性添加一组统一的错误代码。

B.17.2包入的更多描述性原因

自1.0版以来，OpenFlow管道发生了广泛的变化，但是，

ofp_packet_in消息未更改。结果，管道的许多不同部分都在使用
相同的原因值。1.4版引入了更多描述性原因，因此控制器可以
正确区分流水线的哪一部分将数据包重定向到控制器（EXT-136）。

主要的变化是，“输出动作”原因OFPR_ACTION有效地分为四个原因，
“应用动作”，“动作集”，“组存储区”和“分组输出”，代表了四个不同的上下文
使用此操作的位置。重命名OFPR_NO_MATCH的“不匹配”原因是为了正确反映
它是由表未命中流条目生成的事实。

ofp_packet_in消息的新原因值集是：

- OFPR_TABLE_MISS：没有匹配的流（表丢失流条目）。
- OFPR_APPLY_ACTION：输出到应用动作中的控制器。
- OFPR_INVALID_TTL：数据包具有无效的TTL。
- OFPR_ACTION_SET：输出到操作集中的控制器。
- OFPR_GROUP：输出到组存储区中的控制器。
- OFPR_PACKET_OUT：以包输出的形式输出到控制器。

B.17.3光端口属性

一组新的端口属性增加了对光端口的支持，它们包括用于配置和监视的字段
激光的发射和接收频率及其功率（EXT-154）。这些新属性可以
用于配置和监视以太网光端口或电路交换机上的光端口。

- p_port_mod_prop_optical的光端口mod属性可配置光端口。
- p_port_stats_prop_optical的光端口统计属性用于监视光端口。
- p_port_desc_prop_optical的光端口描述属性，用于描述光端口容量
  能力。

B.17.4流量计删除的流量去除原因

为ofp_flow_removed消息添加新的原因值OFPRR_METER_DELETE，以表示
流表条目由于仪表的删除而被删除（EXT-261）。

在交换机上删除仪表后，所有使用该仪表的流量条目都将被删除。这是
类似于小组的运作方式。被删除的流条目可能会生成被删除流的消息
（取决于配置）。在1.3版中，我们没有出现这种情况的原因，而1.4版修复了以下问题：
错误。

B.17.5流量监控

OpenFlow协议定义了一种多控制器方案，其中多个控制器可以管理一个
开关。流量监控允许控制器实时监控对任何子集的更改。
流量表由其他控制器（EXT-187）完成。

流监控框架允许控制器定义多个监控器，每个监控器选择一个
流表的子集。每个监视器都包含一个表ID和一个定义子集的匹配模式
受监控。在流定义的子集之一中添加，修改或删除任何流条目时
监视器，将事件发送到控制器以通知其更改。

- p_flow_monitor_request的多部分请求，用于在交换机上设置流量监控器。
- 使用of_p_flow_update_full或发送到控制器的流量监控器更新事件的详细信息
  使用ofp_flow_update_abbrev缩写。
- 监视标志以定义更新的格式。
- 流控制机制可避免积压监视器更新。

B.17.6角色状态事件

规范的1.2版增加了控制器在多控制器中设置其角色的能力。
环境。当某个控制器当选为“主”角色时，先前的主控制器将被降级
扮演“奴隶”角色，但是未通知该控制器。在版本1.4中，角色状态消息
使交换机能够通知控制器其角色的更改（EXT-191）。

- 角色状态事件OFPT_ROLE_STATUS，通知控制器更改角色。
- 实验者数据的角色状态属性，为p_role_prop_experimenter。

B.17.7驱逐

大多数流表具有有限的容量。在规范的先前版本中，当流表已满时，
新的流条目未插入流表中，并且错误返回给控制器。然而，
达到这一点是很成问题的，因为控制器需要时间来操作流表，并且
这可能会导致服务中断。驱逐增加了一种机制，该机制使交换机能够自动
消除重要性较低的条目，以便为较新的条目留出空间（EXT-192）。这样可以更平滑
表已满时行为的降级。

- Table-mod标志OFPTC_EVICTION用于启用或禁用表驱逐。
- Flow-mod重要性，以可选地表示驱逐流程条目的重要性。
- p_table_mod_prop_eviction的表描述逐出属性，用于描述逐出类型
  由开关执行。

B.17.8空缺事件

大多数流表具有有限的容量。在规范的先前版本中，当流表已满时，
新的流条目未插入流表中，并且错误返回给控制器。然而，
达到这一点是很成问题的，因为控制器需要时间来操作流表，并且
这可能会导致服务中断。空缺事件添加了一种机制，使控制器能够获取
基于控制器（EXT-192）选择的容量阈值的预警。这允许
控制器要事先做出反应，并避免桌子满了。

- 表状态事件 OFPT_TABLE_STATUS 与 原因 OFPTR_VACANCY_DOWN 和
  OFPTR_VACANCY_UP通知控制器空缺更改。
- 滞后机制可通过使用两个阈值vacancy_down和vacancy_down来避免虚假事件
  职位空缺。
- Table-mod空缺属性，用于设置空缺阈值ofp_table_mod_prop_vacancy。

B.17.9捆绑包

添加捆绑机制，使单个操作可以应用一组OpenFlow消息
（EXT-230）。这样可以准原子应用相关的更改，并更好地同步
一系列开关之间的变化。

- •捆绑包控制消息OFPT_BUNDLE_CONTROL，用于创建，销毁和提交捆绑包。
- •捆绑添加消息OFPT_BUNDLE_ADD_MESSAGE，以将OpenFlow消息添加到捆绑中。

- •捆绑包错误类型OFPET_BUNDLE_FAILED，用于报告捆绑包操作错误。

B.17.10同步表

许多交换机可以对相同的查找数据执行多个查找。例如，标准以太网
学习表对同一组MAC地址执行学习查询和转发查询。
另一个示例是RPF检查，它可以重用IP转发数据。同步表功能
允许将这些结构表示为两个表的集合，其数据是同步的（EXT-232）。

同步表使用表功能OFPTFPT_TABLE_SYNC_FROM中的新属性表示。
它定义了两个流表之间的同步抽象，但是没有定义和
表示流表之间的流条目转换。

B.17.11组和计量表更改通知

OpenFlow协议定义了一种多控制器方案，其中多个控制器可以管理一个
开关。组和仪表更改通知允许控制器实时监视对设备的更改
由其他控制器完成的组表或仪表表（EXT-235）。

“ group-mod”和“ meter-mod”请求被简单地封装在OFPT_REQUESTFORWARD中
发送到其他控制器的异步消息。这些通知可通过
“设置异步配置”消息。

B.17.12错误代码的优先级低

某些开关可能会限制表中可以使用的优先级。例如一个开关
可能会强制执行表中的“最长前缀匹配”规则，要求优先级与
面具。新的错误代码OFPFMFC_BAD_PRIORITY使交换机能够正确通知控制器
发生这种情况时（EXT-236）。

B.17.13 Set-async-config的错误代码

OFPT_GET_ASYNC_REQUEST功能是在1.3.0版中引入的。没有错误消息-
为该功能定义的先贤，但是此请求有可能失败。一种新的错误类型，

OFPET_ASYNC_CONFIG_FAILED，带有一组适当的错误代码，可使交换机正确运行
发生这种情况时通知控制器（EXT-237）。

为OFPET_ASYNC_CONFIG_FAILED定义的错误代码集为：

- •OFPACFC_INVALID：一个掩码无效。
- •OFPACFC_UNSUPPORTED：不支持请求的配置。
- •OFPACFC_EPERM：权限错误。

B.17.14 PBB UCA标头字段

添加了新的OXM字段OFPXMT_OFB_PBB_UCA以匹配“使用客户地址”标头
PBB标头（EXT-256）中的字段。

B.17.15重复指令的错误代码

OpenFlow规范将流条目中包含的指令定义为一组，并且
该指令不能在该集中重复。一个新的错误代码OFPBIC_DUP_INST启用了该开关
当流条目包含重复指令时正确通知控制器（EXT-260）。

B.17.16多部分超时的错误代码

多部分请求和答复被编码为一系列消息。此版本的规范
定义如何处理未终止的序列，定义了一些最小超时以及错误
代码（EXT-264）。

- 为以下项定义最小超时（100毫秒）和错误代码（OFPBRC_MULTIPART_REQUEST_TIMEOUT）
  未终止的多部分请求序列。
- 为不确定的时间定义最小超时（1 s）和错误代码（OFPBRC_MULTIPART_REPLY_TIMEOUT）
  有序的多部分答复序列。

B.17.17将默认TCP端口更改为6653

IANA分配给ONF的OpenFlow交换协议要使用的TCP端口号6653。
应当停止使用以前的端口号6633和976。OpenFlow交换机和
默认情况下，OpenFlow控制器必须使用6653（未使用用户指定的端口号时）。

B.18 OpenFlow版本1.4.1

发售日期：2015年3月26日
协议版本：0x05

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.18.1变更

- 当由于同步而添加流条目时，请覆盖现有条目而不是合并
  他们（EXT-355）。
- 将重叠检查标志的范围扩展到同步流表（EXT-355）。
- 授权双向同步表（EXT-361）的可逆转换。
- 添加同步错误OFPFMFC_IS_SYNC（EXT-378）。
- 在交换机功能中添加用于捆绑和流量监控（EXT-404）的功能。

B.18.2澄清

- 说明表可以自身同步（EXT-346）。
- 明确说明在table-mods（EXT-346）中，vacancy属性是可选的。
- 修正拼写错误，在流量监控部分（EXT-346）中将OFPFMF_OWN替换为OFPFMF_NO_ABBREV。
- 解决错误，如果捆绑的消息具有相同的xid，则发送错误，而不是相反（EXT-346）。
- 明确说明不能将捆绑包完全同时应用于不同的交换机，
  （EXT-355）。
- 说明如果同步流条目不存在，则不会删除任何流条目（EXT-355）。
- 在同步部分中澄清学习示例，并阐明它只是说明性的（EXT-
  355）。
- 建议不要使用复杂的表同步（EXT-361）。
- 明确说明流量监视器回复消息可能是异步消息（EXT-371）。
- 阐明同步错误OFPFMFC_CANT_SYNC（EXT-378）的用法。
- 说明捆绑是可选的（EXT-404）。
- 说明流量监控是可选的（EXT-404）。
- 添加重复指令错误OFPBIC_DUP_INST（EXT-424）的描述。
- 各种错别字，语法和拼写修复程序（EXT-346，EXT-355）。
- 指定在flow-mod修改（EXT-496）上重要性不变。
- 说明收回成功和失败的后果，指定错误（EXT-502）。

•明确指出不太重要的流条目不会触发逐出（EXT-502）。
•明确监视器请求可以包含多个监视器和部件（EXT-505）。
•阐明可以清除表缺失流条目（EXT-538）。
•拼写，语法和其他拼写错误（EXT-543）。

B.19 OpenFlow版本1.5.0

发售日期：2014年12月
协议版本：0x06

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.19.1出口表

在规范的先前版本中，所有处理都是在输入端口的上下文中完成的。
1.5版引入了出口表，使处理可以在输出端口的上下文中完成
（EXT-306）。当数据包输出到端口时，它将在第一个出口表处开始处理，其中
流条目可以定义处理并将数据包重定向到其他出口表。OXM的新领域，
OXM_OF_ACTSET_OUTPUT，启用出口流条目以匹配输出端口（EXT-233）。

•发送到输出端口的数据包由第一个出口表处理。
•组处理和保留端口替换发生在出口表之前。
•定义出口表和出口流条目的行为，大部分与入口相似。
•新的匹配字段OXM_OF_ACTSET_OUTPUT用于匹配操作集中的输出值，对于
出口表，对于入口表是可选的。
•禁止在出口操作集中添加输出或组操作，以防止更改输出端口。
•允许出口流条目将操作列表中的输出操作用于出口镜像。

•管道字段从入口表携带到出口表。
•表功能可设置第一个出口表。
•表功能可以识别表，而不是用于入口和/或出口。
•引入表功能请求命令以更简单地更新表功能。

B.19.2知道包类型的管道

在规范的早期版本中，所有数据包都必须是以太网数据包。1.5版引入了
数据包感知管道，支持处理其他类型的数据包，例如IP数据包或PPP数据包
（EXT-112）。新的OXM管道字段标识数据包类型。数据包类型在
可扩展的方式，它使用其他标准定义的多个名称空间，并允许实验者
数据包类型。数据包类型字段可以在匹配中使用，并用作标头字段的前提。
包类型字段也可以在包入和包出消息中使用，以标识有效负载
这些消息携带。

•基于各种现有名称空间的标头类型定义，定义规范标头类型。
•New Packet Type管道字段。该OXM包含数据包的规范头类型
最外面的标题。
•数据包类型OXM可用于匹配以匹配特定的数据包类型。
•数据包类型OXM用作各种报头匹配字段的先决条件。
•数据包输入中使用的数据包类型OXM来标识有效负载。
•数据包输出中用于识别有效负载的数据包类型OXM。
•表功能属性公开了每个流表支持的数据包类型。

目前，该规范仅限于每个交换机使用单个数据包类型，因为数据包类型会转换为
sions没有定义。

B.19.3可扩展流条目统计

规范的早期版本使用固定的结构进行流条目统计。1.5版引入
灵活的编码，OpenFlow可扩展统计信息（OXS），用于编码任意流条目统计信息
（EXT-334）。将现有流条目统计信息重新定义为标准OXS字段。

•基于OXM并与OXM兼容的OXS TLV格式来表示流条目统计信息
更简单的实现。
•将现有流条目统计信息表示为标准OXS字段：流持续时间，流计数，数据包
计数，字节计数。
•引入新的标准流条目统计：流空闲时间。
•启用基于任意类别或基于实验者的流条目统计信息。

•在所有带有流条目统计信息的消息中使用OXS：流删除的消息，流统计信息
多部分，流动聚集多部分。
•引入轻量级流量统计信息，将现有流量统计信息重命名为流量描述。

B.19.4流条目统计触发器

轮询流条目统计信息可能会导致交换机的高开销和利用率。新统计
触发机制可根据各种统计信息将统计信息自动发送到控制器
阈值（EXT-335）。

•新指令OFPIT_STAT_TRIGGER，用于使用OXS定义一组统计阈值。
•统计信息触发标志以允许周期性阈值。
•使用轻量级流量统计信息将统计信息阈值有效地分批给控制器
多部分。

B.19.5在两个OXM字段之间复制的复制字段操作

现有的*Set-Field*操作允许使用静态值设置标头字段或管道字段。新的
*Copy-Field*操作允许将值从一个标头或管道字段复制到另一个标头或
管道字段（EXT-320）。

•新指令OFPAT_COPY_FIELD，用于在字段之间复制值。
•定义为OXM ID的源和目标字段，支持标准OXM，基于类的OXM
和实验者OXM。
•复制字段指令可以将任意长度的位子字段从源复制到目标的任何位置
偏移量。
•“新表功能”属性表示对“复制字段”的支持。

B.19.6数据包寄存器流水线字段

引入数据包寄存器，它是一组流水线字段，可用作*复制-的*通用暂存空间
*现场*行动（EXT-244）。数据包寄存器是可选匹配的。

•数据包寄存器管道字段的新OXM类。
•交换机可以支持任意数量的寄存器，每个寄存器为64位。

B.19.7 TCP标志匹配

添加了新的OXM字段OFPXMT_OFB_TCP_FLAGS以匹配TCP头中的标志位
（EXT-109）。此字段允许匹配所有标志，例如SYN，ACK和FIN，可用于检测
TCP连接的开始和结束。

B.19.8用于选择性存储区操作的组命令

在规范的早期版本中，组操作只能更改整个组存储桶
在一组。两个新的组命令使您可以插入或删除特定的组桶，而无需
影响群组中的其他群组（EXT-350）。

•组命令OFPGC_INSERT_BUCKET将单个组存储桶插入组中。
•组命令OFPGC_REMOVE_BUCKET从组中删除特定的组桶。

•将bucket_id字段添加到存储桶以标识存储桶。
•添加组属性以扩展组。
•添加组存储桶属性以扩展组存储桶。
•重新定义存储桶字段的权重，watch_port和watch_group作为组存储桶属性。

B.19.9允许设置字段操作设置元数据字段

规范的先前版本对set-field操作（管道字段）有限制
OXM_OF_METADATA不能是设置字段操作的有效参数。在本规范中，
取消了限制，并且OXM_OF_METADATA可以是设置字段操作的有效参数（EXT-
46）。

B.19.10允许在设置字段操作中使用通配符

在规范的早期版本中，“ *设置字段*”操作只能设置引用的整个字段。
此版本的规范使交换机能够在“设置字段”中支持掩码，以便仅
修改字段的指定位（EXT-314）。

B.19.11预定的捆绑包

规范的先前版本引入了*捆绑消息*。捆绑包是Open-
来自控制器的流修改请求被应用为单个OpenFlow操作。

当前规范扩展了捆绑功能（EXT-340），允许：

•预设束：束提交消息可以包括一个*执行时间*，指定*当*所述
交换机有望提交捆绑。
•捆绑功能请求：允许控制器向交换机查询其捆绑功能，包括：
包括它是否支持原子束，有序束和预定束。

B.19.12控制器连接状态

Controller Connection Status（控制器连接状态）使控制器可以知道来自以下位置的所有连接的状态
切换到控制器（EXT-454）。这允许控制器检测控制网络分区或
监视其他控制器的状态。使用标准URI（EXT-275）识别控制器。

•控制器状态多部分消息，用于查询控制器连接的状态。
•定义URI方案以标识控制器连接。
•使控制器可以设置一个简短的ID来标识自己。
•控制器状态消息，用于通知控制器连接状态的变化。

B.19.13仪表动作

在规范的早期版本中，计量是通过Meter指令完成的。这个版本
用仪表动作（EXT-379）替换该指令。结果，可以连接多个仪表
到流量条目，仪表可用于组存储桶中。

•介绍仪表操作OFPAT_METER。
•弃用仪表指令OFPIT_METER。

B.19.14启用设置包输出中的所有管道字段

规范的早期版本支持在以下位置设置数据包的*In-Port*管道字段：
*包出*消息。该规范版本支持在以下位置设置数据包的所有管道字段：
的*分组输出*消息（EXT-427）。例如，这可以为以下设置隧道ID管道字段：
逻辑端口。

•在packet_out消息中添加匹配结构，以将管道字段编码为OXM。
•在比赛中将in_port字段移动为OXM。

•指定数据包输出根据匹配中包含的字段设置数据包的管道字段。

B.19.15管道字段的端口属性

逻辑端口可能会占用并生成管道字段。例如，管道字段“ *隧道ID”*可以
由逻辑端口用来表示与封装关联的元数据。新端口属性
允许逻辑端口表示使用哪些管道字段以及由逻辑产生哪些管道字段
端口（EXT-388）。

•端口描述属性OFPPDPT_PIPELINE_INPUT是为数据包提供的OXM字段的列表
在该端口上收到。
•端口描述属性OFPPDPT_PIPELINE_OUTPUT是包含以下内容的管道字段的列表：
数据包发送给端口时的总和。

B.19.16港口财产的再循环

逻辑端口在处理完数据包后可能会将其循环回OpenFlow管道。一种
新属性允许一个端口与另一个将数据包返回到OpenFlow的端口相关联
管道（EXT-399）。这样的再循环可以保留管道场。

B.19.17澄清和改善障碍

在整个规范中，对障碍消息的描述及其对交换机的影响是统一的。
阳离子并进行澄清（EXT-189）。屏障也更改为将状态提交到数据路径。

•澄清障碍控制消息的顺序。
•澄清障碍通知完成的处理。
•指定屏障将状态提交到数据路径。

•阐明有序束可以用作障碍的替代品。

B.19.18始终在端口配置更改时生成端口状态

在规范的先前版本中，仅在以下情况下将端口状态消息发送到控制器：
端口状态已更改，或者在OpenFlow外部更改了配置。另一方面，如果
一个控制器更改了配置，没有端口状态消息发送到控制器。这是
多控制器设置中存在问题，因为无法将端口配置更改通知其他控制器
由一个控制器完成。

在本规范中，端口状态消息会发送给控制器以更改端口
配置，包括由一个OpenFlow控制器（EXT-338）完成的配置。

B.19.19将所有实验者OXM-ID设置为64位

该规范的先前版本尚不清楚如何对Experimenter OXM进行编码。这个规格
验证要求实验者OXM在的oxm_field字段中对实验者类型进行编码
OXM标头（EXT-380）。这使用于识别OXM的所有OXM-ID统一为64
位。

B.19.20组，端口和队列多部分的统一请求

统计和描述的多部分请求已更改为使用以下通用结构：
组，端口和队列（EXT-69）的情况。

•端口统计信息和端口描述使用通用请求格式。
•可以请求各个端口的描述。
•组统计信息和组描述使用通用请求格式。
•可以请求各个组的描述。
•队列统计信息和队列描述使用通用请求格式。

B.19.21重命名某些类型以保持一致性

为了保持一致性，已将某些OpenFlow类型重命名。

- 统计部分现在带有_STATS后缀（EXT-302）。
- 将多部分METER_CONFIG重命名为METER_DESC（EXT-302）。
- 仪表统计信息字段flow_count重命名为ref_count（EXT-374）。

第283页

B.19.22规范重组

规范中移动了一些文本，以提高可读性和更准确的交叉引用（EXT-507）。

- 为错误代码添加段落标题。
- 将错误条件的说明移至错误消息部分。
- 在“ OpenFlow”表部分的末尾移动组和仪表。
- 将关于流量去除的小节移至关于控制通道的小节。
- 在“多部分”部分中将“多部分表”子部分进一步下移。
- 为操作结构添加段落标题。

B.20 OpenFlow版本1.5.1

发售日期：2015年3月26日
协议版本：0x06

有关每个更改的更多详细信息，请参考错误跟踪ID。

B.20.1变更

- 为flow-mod（EXT-530）中的仪表损坏添加新的错误OFPBAC_BAD_METER。
- 不要指定每个数据包如何映射到每个仪表频带（EXT-474）。

B.20.2澄清

- 添加数字表（EXT-474）。
- 正确指定仪表标志并说明max_bands功能（EXT-474）。
- 在用于描述仪表命令的枚举中使用显式值（EXT-539）。
- 拼写，语法和其他拼写错误（EXT-544）。

附录C学分

规格贡献，按字母顺序排列：

Anders Nygren，Ben Pfaff，Bob Lantz，Brandon Heller，Casey Barker，Curt Beckmann，Dan Cohn，Dan Malek，Dan Talayco，David Erickson，David McDysan，David Ward，Edward Crabbe，Fabian Schneider，Glen Gibb，Guido Appenzeller，Jean Tourrilhes，Johann Tonsing，Justin Pettit，KK Yap，Leon Poutievski，Linda Dunbar，Lorenzo Vicisano，Martin Casado，Masahiko Takahashi，小林正义（Masayoshi Kobayashi），迈克尔·奥尔（Michael Orr），纳文德拉·亚达夫（Navindra Yadav），尼克·麦克基恩（Nick McKeown），尼科·德·赫里乌斯（N 拉吉什（Rajesh Madabushi），拉吉夫·拉马纳丹（Rajiv Ramanathan），里德·普莱斯（Reid Price），罗伯·舍伍德（Rob Sherwood），索拉夫·达斯（Saurav Das），斯派克·柯蒂斯（Spike Curtis），斯里拉姆·纳塔拉然（Sirram Natarajan），塔尔·米兹拉希（Tal Mizrahi），达也矢（Tatsuya Yabe），丁万福，扬尼斯·雅库米斯（Y 摩西，佐尔坦·拉霍斯·基斯。