目录

[摘要 2](#_Toc1504047)

[1 介绍 2](#_Toc1504048)

[1.1 规范性引用文件 3](#_Toc1504049)

[1.2 非规范性引用文件 4](#_Toc1504050)

[1.3 术语 4](#_Toc1504051)

[1.4 结构规范 5](#_Toc1504052)

[2 Virtio 设备的基本设施 5](#_Toc1504053)

[2.1 设备状态字段 5](#_Toc1504054)

[2.2 功能位 7](#_Toc1504055)

[2.3 设备配置空间 8](#_Toc1504056)

[2.4 虚拟队列 10](#_Toc1504057)

[3 一般初始化和设备操作 20](#_Toc1504058)

[3.1 设备初始化 20](#_Toc1504059)

[3.2 设备操作 21](#_Toc1504060)

[3.3设备清理 24](#_Toc1504061)

[4 Virtio传输选项 24](#_Toc1504062)

[4.1 PCI总线下的Virtio 24](#_Toc1504063)

[4.2 MMIO下的Virtio 42](#_Toc1504064)

[4.3 Channel I/O下的Virtio 53](#_Toc1504065)

[5 设备类型 62](#_Toc1504066)

[5.1 网络设备 63](#_Toc1504067)

[5.2 块设备 80](#_Toc1504068)

[5.3控制台设备 89](#_Toc1504069)

[5.4熵设备 94](#_Toc1504070)

[5.5传统内存气球（Memory Balloon）设备 95](#_Toc1504071)

[5.6 SCSI主机设备 101](#_Toc1504072)

[6 保留的功能位 114](#_Toc1504073)

[6.1驱动程序要求：保留功能位 115](#_Toc1504074)

[6.2设备要求：保留功能位 115](#_Toc1504075)

[6.3传统接口：保留功能位 115](#_Toc1504076)

[7 一致性 115](#_Toc1504077)

[7.1一致性目标 115](#_Toc1504078)

[7.2 驱动程序的一致性 116](#_Toc1504079)

[7.3设备一致性 118](#_Toc1504080)

[7.4传统接口：过渡设备和过渡驱动器一致性 121](#_Toc1504081)

[附录B.创建新设备类型 122](#_Toc1504082)

[B.1有多少个Virtqueues？ 122](#_Toc1504083)

[B.2什么是设备配置空间布局？ 123](#_Toc1504084)

[B.3什么是设备号？ 123](#_Toc1504085)

[B.4有多少MSI-X向量？ （用于PCI） 123](#_Toc1504086)

[B.5设备的改进 123](#_Toc1504087)

## 摘要

本文档描述了“虚拟IO”家族的设备规范。这些设备可以在虚拟环境中找到，而通过设计，可以使它们看起来像虚拟机中的访客的物理设备——本文档也会将它们如此进行看待。这种相似性允许访客使用标准驱动程序和发现机制。

Virtio及此规范的目的是——虚拟环境及访客应具有直观，高效，标准以及可扩展的虚拟设备机制，而非精确到每个环境或者每个操作系统。

## 1 介绍

本文档描述了“virtio”系列设备的规格。这些设备位于虚拟环境，但从设计上来看，它们在虚拟机中对客户机来说就像是物理设备，这份文件就是这样看待它们的。这种相似性允许客户机使用标准驱动程序和发现机制。

Virtio和这个规范的目的就是虚拟环境和客户机应该有一个直接、高效、标准而且可扩展的可供虚拟设备使用的机制，而不是特定每个环境的或者每个操作系统的机制。

**直接：**virtio设备使用正常的中断总线机制和DMA，这应该是任何设备驱动程序作者所熟悉的。没有异乎寻常的page-flipping和COW机制：这只是一种正常设备[[1]](#footnote-0)。

**高效**：virtio设备由于用于输入输出的描述符环组成，这些描述符环排列整齐。以避免驱动程序和设备写入统一缓存线时产生缓存效果。

**标准**：virtio除了支持连接设备的总线。在本规范中，virtio设备通过MIMO、通道I/O和PCI总线传输[[2]](#footnote-1)实现，早期的草案已在其他总线上实现，不包括在这里。

**可扩展：**virtio设备包含在设备启动期间由客户机操作系统所承认的功能位。它允许向前和先后兼容：设备提供所有它知道的功能，驱动程序也承认并且希望使用那些功能。

### 1.1 规范性引用文件

[RFC2119] Bradner S., “Key words for use in RFCs to Indicate Requirement Levels”, BCP 14,  
RFC 2119, March 1997.  
http://www.ietf.org/rfc/rfc2119.txt  
[S390 PoP] z/Architecture Principles of Operation, IBM Publication SA22-7832,  
http://publibfi.boulder.ibm.com/epubs/pdf/dz9zr009.pdf, and any future revisions  
[S390 Common I/O] ESA/390 Common I/O-Device and Self-Description, IBM Publication SA22-7204,  
http://publibfp.dhe.ibm.com/cgi-bin/bookmgr/BOOKS/dz9ar501/CCONTENTS,  
and any future revisions  
[PCI] Conventional PCI Specifications,  
http://www.pcisig.com/specifications/conventional/, PCI-SIG  
[PCIe] PCI Express Specifications  
http://www.pcisig.com/specifications/pciexpress/, PCI-SIG  
[IEEE 802] IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture,  
http://standards.ieee.org/about/get/802/802.html, IEEE

[SAM] SCSI Architectural Model,  
http://www.t10.org/cgi-bin/ac.pl?t=f&f=sam4r05.pdf  
[SCSI MMC] SCSI Multimedia Commands,  
<http://www.t10.org/cgi-bin/ac.pl?t=f&f=mmc6r00.pdf>

### 1.2 非规范性引用文件

[Virtio PCI Draft] Virtio PCI Draft Specification  
<http://ozlabs.org/~rusty/virtio-spec/virtio-0.9.5.pdf>

### 1.3 术语

关键词“MUST”、“MUST NOT”、“REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”,本文中的“RECOMMENDED”, “MAY”, and “OPTIONAL”应按照[RCF2119]中进行说明解释。

#### 1.3.1传统接口：术语

对于本协议的早期草案（i.e. revisions before 1.0, see e.g. [Virtio PCI Draft]）中定义了一个相似的，但是差别在于驱动和设备的接口不同。由于这些都是广泛部署的，所以本规范提供“OPTIONAL”功能以简化从这些早期草案中接口的转换。

具体来说，设备和驱动程序可能支持：

**Legacy Interface**是由本规范的早期草案（1.0之前）指定的接口。

**Legacy Device**是在发布此规范之前实现的设备，并在主机端实现传统接口。

旧设备和旧驱动程序并不符合本规范。

为了简化这些早期草案接口的转换，设备可以实现：

**过渡设备**（Transitional Device）支持符合本规范的两个驱动程序并允许旧驱动。

同样，驱动程序也可以实现：

**过渡性驱动程序**（Transitional Driver）支持符合本规范的两个设备和旧设备的驱动程序。

**注意**：不需要旧接口；也就是说，除非您有向后的需求，否则不要实现它们。

兼容性！

不具备传统兼容性的设备或驱动程序分别称为非过渡性设备和驱动程序。

#### 1.3.2 从早期规范草案过渡

对于已经实现旧接口的设备和驱动程序，必须进行一些更改以支持此规范。

在这种情况下，读者可以将注意力集中在章节标题中标记为“Legacy Interface”的章节上。这些突出显示了自早期草稿以来所做的更改。

### 1.4 结构规范

内存结构布局中的许多设备和驱动程序都是使用C结构语法编写的。假设所有结构都没有额外的填充。为了强调这一点，已知公共C编译器在结构中插入额外填充的情况下，使用GNU C\_attribute\_（packed））语法标记。对于结构定义中使用的整数数据类型，使用以下约定：

u8、u16、u32、u64以位为单位指定长度的无符号整数。

le16，le32，le64指定长度的无符号整数，以位为单位，以小尾数字节顺序排列。

be16、be32、be64是指定长度的无符号整数，以位为单位，按高位字节顺序排列。

## 2 Virtio 设备的基本设施

Virtio 设备通过特定于总线的方式发现并识别（参见具体总线部分：4.1 PCI总线上的Virtio，4.2 MMIO下的Virtio，4.3 Channel I/O下的Virtio）。每个设备由以下部分组成：

·设备状态字段

·功能位

·设备配置空间

·一个或多个虚拟队列

### 2.1 设备状态字段

当驱动程序初始化设备期间，驱动程序将遵循3.1中给定的步骤顺序。

设备状态字段给出一个序列已完成步骤的简单低级指示。将它想象成它与控制台上的交通灯相连，指示每个设备的状态是最有效的。下列位已定义（在下面以通常的设置顺序列出）：

ACKNOWLEDGE（1）表示访客操作系统已经找到，并识别为有效的virtio设备。

DRIVER（2）表示访客操作系统已经知道如何驱动该设备

**注：**在设置该位之前可能存在着显著地（或无限的）延迟。例如，在Linux下，驱动程序可为可加载模块。

FAILED（128）表示访客出现了某些，并已放弃设备。这可能是内部错误，或是驱动程序由于某些原因无法识别此设备，甚至是在设备操作期间发生了致命错误。

FEATURES\_OK（8）表示驱动程序已经确认其理解的所有功能，且功能协商已完成。

DRIVER\_OK（4）表示驱动程序已经设置完毕且准备好驱动此设备。

DEVICE\_NEEDS\_RESET（64）表示此设备发生了无法恢复的错误。

表示此设备在访客端发生了某种错误，并已放弃设备。这可能是内部错误，或是驱动程序由于某些原因无法识别此设备，甚至是在设备操作期间发生了致命错误。

#### 2.1.1 驱动要求：驱动状态字段

驱动程序**必须**更新设备状态，通过设定位来指示3.1中所指定的驱动程序初始化序列中已完成的步骤。驱动程序**不得**清除设备状态位。如果驱动程序置位了FAILED位，则驱动程序**必须**稍后复位设备，然后再尝试重新初始化。

如果DEVICE\_NEEDS\_RESET已被置位，那么驱动程序**不该**依靠设备操作完成。

**注：**例如，如果置位了DEVICE\_NEEDS\_RESET，则驱动程序不能假设跑飞的请求能被完成，也不能假设它们未完成。好的应用会尝试用发出复位来进行恢复。

#### 2.1.2 设备要求：设备状态字段

设备初始化时**必须**将设备状态初始化至0。

设备不得在DRIVER\_OK前使用缓冲区或通知驱动程序。

当设备进入错误状态，需要复位时，**应该**置位DEVICE\_NEEDS\_RESET。如果DRIVER\_OK已被置位，则在置位DEVICE\_NEEDS\_RESET后，设备**必须**向驱动程序发送设备配置更改通知。

### 2.2 功能位

每个virtio设备都提供其理解的所有功能。在设备初始化期间，驱动程序读取该信息并向设备告知驱动程序接收到的子集。复位设备是重新协商的唯一方法。

此方式允许向前和向后兼容：如果设备增加了新的功能位，则旧版驱动程序将不会将该功能位写回设备。类似的，如果驱动程序增加了设备不支持的功能，设备将会认为驱动程序没有提供新的功能。

功能位分配如下：

0到23 特定设备类型功能位

24到32 为队列扩展和功能协商机制保留的功能位

33及以上 为未来扩展保留的功能位

**注：**例如，网络设备的功能位0用于（即设备ID 1）指示设备支持数据包校验和。

特别地，通过提供新的功能位来指示设备配置空间的新字段。

#### 2.2.1 驱动要求：功能位

驱动程序**不得**接受设备未提供的功能，且**不得**接受需要另一个未被接受的功能的功能。

如果设备没有提供驱动程序所理解的功能，驱动程序**应该**进入向后兼容模式，否则**必须**置位FAILED设备状态位并停止初始化。

#### 2.2.2 设备要求：功能位

设备**不得**提供需要另一个未提供的功能的功能。设备**应该**接受任何驱动程序接收的有效功能子集，否则在驱动程序写入时置位FEATURE\_OK设备状态位**必须**失败。

#### 2.2.3 旧版接口：关于功能位的注意事项

过渡驱动程序必须通过检测未提供的功能位VIRTIO\_F\_VESION\_1来检测旧版设备。过渡设备必须通过检测驱动程序未确认VIRTIO\_F\_VERSION\_1来检测旧版驱动程序。

在这种情况下，设备通过旧版接口进行使用。

旧版接口支持是**可选的**。因此，过渡和非过渡设备及驱动程序都符合此规范。

与过渡设备和驱动程序有关的要求均包含于像这个部分这样的名为“旧版接口”的部分中。

当通过旧版接口使用设备时，过渡设备和过渡驱动**必须**根据旧版接口部分中所记录的要求来进行操作。这些部分中的规范文本通常不适用于非过渡设备。

### 2.3 设备配置空间

设备配置空间通常用于很少改变或初始化时间参数。配置字段是可选的，它们的存在由功能位进行指示：该规范的未来版本可能通过在末尾添加额外的字段来扩展配置空间。

**注：**设备配置空间在多字节字段中使用小端模式。

每次传输还为设备配置空间提供世代数，只要对设备配置空间的两次访问可能看到该空间产生两种不同的版本，该世代数就会发生变化。

#### 2.3.1 驱动要求：设备配置空间

驱动程序**不得**假设从大于32位宽的字段读取为原子读取，也**不得**从多个字段读取：驱动程序**应该**应该像这样读取设备配置空间：

u32 before, after;

do {

before = get\_config\_generation(device);

// read config entry/entries.

after = get\_config\_generation(device);

} while (after != before);

对于可选的配置空间字段，驱动程序**必须**在访问对应配置空间之前检查是否提供了相应功能。

**注：**有关功能协商的详细信息参阅3.1节。

驱动程序**不得**限制结构大小以及设备配置空间大小。相反，驱动程序只**应该**检查设备配置空间大小是否足够容纳设备操作所需的字段。

**注：**例如，如果规范生命设备配置空间“包含单个8位字段”，则驱动程序应该理解这意味着设备配置空间还可能包含任意数量的尾部填充，并接受任何设备配置空间的大小大于或等于所规定的8位大小。

#### 2.3.2 设备要求：设备配置空间

在驱动程序置位FEATURES\_OK之前，设备必须允许读任何特定于设备的配置字段。只要特征位由设备提供，就包括允许读以功能位为条件的字段。

#### 2.3.3 旧版接口：关于设备配置空间字节序的注意事项

注意对旧版接口，设备配置空间通常为访客的本地字节序，而不是PCI的小端字节序。每个设备都记录了正确的字节序。

#### 2.3.4 旧版接口：设备配置空间

旧版设备并没有配置生成字段，因此如果进行更新配置，则容易受到竞争条件的影响。这将会影响块容量（见5.2.4）以及网络mac（见5.1.4）字段；当使用旧版接口时，直到两次读取得到相同结果之前，驱动程序应该多次读取这些字段。

### 2.4 虚拟队列

在virtio设备上进行批量数据传输的机制被称为虚拟队列。每个设备可拥有零个或多个虚拟队列。每个队列含有一个16位的队列大小参数，该参数设置项数并反映队列的总大小。

每个虚拟队列由三部分组成：

·描述符表

·可用环

·已用环

其中每个部分在访客内存中物理连续，且有着不同的对其要求。

下表总结了虚拟队列每个部分的内存对其与以字节为单位的大小要求：

|  |  |  |
| --- | --- | --- |
| 虚拟队列组成 | 对齐值 | 大小 |
| 描述符表 | 16 | 16\*（队列大小） |
| 可用环 | 2 | 6+2\*（队列大小） |
| 已用环 | 4 | 6+8\*（队列大小） |

对齐值列给出虚拟队列中每个部分的最小对齐值。

大小列给出虚拟队列中每个部分总的字节数。

队列大小对应与虚拟队列中缓冲区的最大大小。队列大小始终为2的幂。最大队列大小为326768。这个值由特定于总线的方式指定。

当驱动程序想要向设备发送缓冲时，它会将其填充至描述符表中的一个槽中（或将一些连接在一起），并将描述符索引写入可用环中，然后通知设备。当设备完成缓冲，它会将描述符索引写入已用环中，并发送一个中断。

#### 2.4.1 驱动要求：虚拟队列

驱动程序必须保证每个虚拟队列部分的第一个字节的物理地址为上表中指定的对其值的倍数。

#### 2.4.2 旧版接口：关于虚拟队列布局的注意事项

对旧版接口来说，虚拟队列布局还存在着一些额外的限制：

每个虚拟队列占用两个或多个物理连接的页（通常定义为4096字节，但取决于传输；此后称之为队列对齐值）且由三部分组成：

|  |  |  |
| --- | --- | --- |
| 描述符表 | 可用环（…填充…） | 已用环 |

特定于总线的队列大小字段控制着虚拟队列的总字节数。当使用旧版接口时，传输驱动程序**必须**从设备处得到队列大小字段，且必须根据公式对虚拟队列总字节数进行分配（队列对齐值由qalign给出，队列大小由qsz给出）：

#define ALIGN(x) (((x) + qalign) & ~qalign)

static inline unsigned virtq\_size(unsigned int qsz)

{

return ALIGN(sizeof(struct virtq\_desc)\*qsz + sizeof(u16)\*(3 + qsz))

+ ALIGN(sizeof(u16)\*3 + sizeof(struct virtq\_used\_elem)\*qsz);

}

这会浪费一些填充空间。当使用旧版接口时，过渡设备与驱动程序**必须**使用以下虚拟队列布局结构来定位虚拟队列元素：

struct virtq {

// The actual descriptors (16 bytes each)

struct virtq\_desc desc[ Queue Size ];

// A ring of available descriptor heads with free-running index.

struct virtq\_avail avail;

// Padding to the next Queue Align boundary.

u8 pad[ Padding ];

// A ring of used descriptor heads with free-running index.

struct virtq\_used used;

};

#### 2.4.3 旧版接口：关于虚拟队列字节序的注意事项

注意，当我们使用旧版接口时，过度设备与驱动程序**必须**使用访客本地字节序作为虚拟队列中字段的字节序。这与本标准规定的小端字节序相反。本标准假设主机已经知道了访客的字节序。

#### 2.4.4 消息结构

使用描述符的消息结构与缓冲区中的内容无关。例如，网络传输缓冲区由12字节的头部后跟网络数据包组成。这可以最简单地放在描述符表中作为12字节的输出描述符，后跟1514字节的输出描述符，而在头部与数据包相邻的情况下，它也可以由单个1526字节的输出描述符组成，甚至可以包含三个或更多个描述符（在这种情况下效率可能会降低）。

注意，某些设备工具对总的描述符大小有着大而合理的限制（例如主机操作系统中的IOV\_MAX）。这在实践中并不构成问题：对创建如通过将网络数据包分成1500个单字节描述符这样不合理的描述符大小的驱动程序，我们将毫不留情。

##### 2.4.4.1 设备要求：信息结构

设备**不得**对描述符的特定排列做出假设。设备**可能**在链中允许存在合理的描述符限制。

##### 2.4.4.2 驱动要求：信息结构

驱动程序**必须**在任何设备可读描述符元素之后放置任何设备可写的描述符元素。

驱动程序**不得**使用过多的描述符来描述缓冲区。

##### 2.4.4.3 旧版接口：信息结构

遗憾的是，尽管有这样规范的结构存在，但是初始的驱动程序使用简单的布局，且设备已经开始依赖于此。此外，virtio\_blk SCSI命令规范要求在帧边界有直观的字段长度（参见5.2.6.3旧版接口：设备操作）。

因此，当使用旧版接口时，VIRTIO\_F\_ANY\_LAYOUT功能同时表示设备与驱动程序对结构均没有进行假设。当其未经协商时，每个设备部分都包含对过度设备的要求。

#### 2.4.5 虚拟队列描述符表

描述符表指的是驱动程序用于设备的缓冲区。addr为物理地址，且缓冲区可与下一个缓冲区相连。每个描述符描述一个缓冲区，缓冲区对设备来说是只读的（“设备可读”）或对设备来说是只写的（“设备可写”），但描述符链可以包含设备可读和设备可写的缓冲区。

对设备提供的内存实际内容取决于设备类型。最常见的是以头部（包含小端字节序字段）开始数据供设备读取，并加上状态标签后缀供设备写入。

struct virtq\_desc {

/\* Address (guest-physical). \*/

le64 addr;

/\* Length. \*/

le32 len;

/\* This marks a buffer as continuing via the next field. \*/

#define VIRTQ\_DESC\_F\_NEXT 1

/\* This marks a buffer as device write-only (otherwise device read-only). \*/

#define VIRTQ\_DESC\_F\_WRITE 2

/\* This means the buffer contains a list of buffer descriptors. \*/

#define VIRTQ\_DESC\_F\_INDIRECT 4

/\* The flags as indicated above. \*/

le16 flags;

/\* Next field if flags & NEXT \*/

le16 next;

};

表中描述符的数量由此虚拟队列的队列大小定义：此为可能的最大描述符链长度。

**注：**旧版[Virtio PCI Draft]将此结构成为wring\_desc，将常量称为VRING\_DESC\_F\_NEXT等，而布局与值是相同的。

##### 2.4.5.1 设备要求：虚拟队列描述符表

设备**不得**向设备可读缓冲器进行写入，且设备**不应**从设备可写缓冲区进行读取（这种操作**可能**在调试或诊断中进行）。

##### 2.4.5.2 驱动要求：虚拟队列描述符表

驱动程序**不得**添加总长度超过字节描述符链；这意味着描述符链中的环是禁止的。

##### 2.4.5.3 简介描述符

某些设备受益于同时分配大量大请求。VIRTIO\_F\_INDIRECT\_DESC功能允许这种操作（见A virtio\_queue.h）。为了增加环的容量，驱动程序可以在内存的任意位置存储间接描述符表，并在主虚拟队列中插入描述符（带有flags&VIRTQ\_DESC\_F\_INDIRECT启动），该描述符引用包含此间接描述符表的内存缓冲区；addr与len分别表示间接表的地址与长度（以字节为单位）。间接表布局结构如下所示（len为引用此表的描述符长度，其为变量，因此代码不会编译）：

struct indirect\_descriptor\_table {

/\* The actual descriptors (16 bytes each) \*/

struct virtq\_desc desc[len / 16];

};

第一个间接描述符位于间接描述符表的开头（索引0），其他间接描述符连接在此之后。如果间接描述符没有下一个有效内容（带有flags&VIRTQ\_DESC\_F\_NEXT关闭）意味着描述符的结束。单个间接描述符表可以同时包括设备可读与设备可写描述符。

###### 2.4.5.3.1

除非已经写上了VIRTIO\_F\_INDIRECT\_DESC功能，否则驱动程序**不得**置位VIRTQ\_DESC\_F\_INDIRECT flag。驱动程序不得在间接描述符中置位VIRTQ\_DESC\_F\_INDIRECT flag（即每个描述符只能有一个表）。

驱动程序**不得**创建超过设备队列大小的描述符链。

驱动程序**不得**在flags中置位VIRTQ\_DESC\_F\_INDIRECT与VIRTQ\_DESC\_F\_NEXT。

###### 2.4.5.3.2 设备要求：间接描述符

设备**必须**忽略引用间接描述符中的只写flag（flags&VIRTQ\_DESC\_F\_WRITE）。

设备**必须**处理零个或多个正常链接的描述符，其后跟包含flags&VIRTQ\_DESC\_F\_INDIRECT在内的单个描述符的情况。

**注：**虽然不常见（大多数工具要么单独使用非间接描述符创建链，要么使用单个费简介元素），但这种布局是有效的。

#### 2.4.6 虚拟队列可用环

struct virtq\_avail {

#define VIRTQ\_AVAIL\_F\_NO\_INTERRUPT 1

le16 flags;

le16 idx;

le16 ring[ /\* Queue Size \*/ ];

le16 used\_event; /\* Only if VIRTIO\_F\_EVENT\_IDX \*/

};

驱动程序使用可用环为设备提供缓冲区：每个环项指描述符链的头部。它仅由驱动程序写，由设备读取。

idx字段指示驱动程序将下一个描述符项放置在环中的位置（以队列大小为模）。它从0开始，并累加。

**注：**旧版[Virtio PCI草案]中将此结构称为vring\_avail，将常量称为VRING\_AVAIL\_F\_NO\_INTERRUPT，但布局和值相同。

#### 2.4.7 虚拟队列中断抑制

如果VIRTIO\_F\_EVENT\_IDX位未经协商，可用环中的flags字段为驱动程序提供了一种粗略的机制来通知设备当缓冲区正在被使用的时候，驱动程序不希望被中断。若已经协商，那么used\_event是一种性能更高效的替代方案，其中驱动程序指定设备在中断之前能够执行多少。

因为这两种方法与设备不同步，因此这些中断抑制方法均不可靠，但他们可以做为有用的优化方法。

##### 2.4.7.1 驱动要求：虚拟队列中断抑制

如果VIRTIO\_F\_EVENT\_IDX功能位未协商：

·驱动程序**必须**将flags置位为0或1。

·驱动程序**可以**通过将flags置位为1来通知设备不需要中断。

否则，如果VIRTIO\_F\_EVENT\_IDX功能位已协商：

·驱动程序**必须**将flags置位为0。

·驱动程序**可以**使用used\_event通知设备是非必要的，直到设备将带有used\_event指定的索引项写入已使用的环（等效地，直到已用环中的idx值达到used\_event+1）。

驱动程序**必须**处理来自设备的伪中断。

##### 2.4.7.2 设备要求：虚拟队列中断抑制

如果VIRTIO\_F\_EVENT\_IDX功能位未协商：

·设备**必须**无视used\_value值

·设备将描述符索引写入已用环后：

-如果flags为1，则设备**不应**发送中断。

-如果flags为0，则设备**必须**发送中断。

否则，如果VIRTIO\_F\_EVENT\_IDX功能已经协商：

·设备必须无视flags的低位。

·设备将描述符索引写入已用环后：

-如果已用环中的idx字段（用于确定该描述符索引的放置位置）与used\_event相等，则设备**必须**发送中断。

-否则设备**不应**发送中断。

**注：**例如，如果used\_event为0，则设备使用VIRTIO\_F\_EVENT\_IDX将会在第一次缓冲区被使用后（并在第65536次缓冲后再一次，等）中断。

#### 2.4.8 虚拟队列已用环

struct virtq\_used {

#define VIRTQ\_USED\_F\_NO\_NOTIFY 1

le16 flags;

le16 idx;

struct virtq\_used\_elem ring[ /\* Queue Size \*/];

le16 avail\_event; /\* Only if VIRTIO\_F\_EVENT\_IDX \*/

};

/\* le32 is used here for ids for padding reasons. \*/

struct virtq\_used\_elem {

/\* Index of start of used descriptor chain. \*/

le32 id;

/\* Total length of the descriptor chain which was used (written to) \*/

le32 len;

};

已用环为设备完成缓冲后返回缓冲区的地方：它只由设备写入，并由驱动程序读取。

环中每个项均是成对的：id表示描述缓冲区的描述符链的头项（这与之前置于访客端的可用环中的项匹配），且len表示写入缓冲区的总字节数。

**注：**len在驱动程序使用不信任的缓冲区时非常有用：如果驱动程序不知道设备具体写入了多少数据，那么驱动程序就需要将缓冲区置零，保证没有数据泄露的发生。

例如，网络驱动程序可以将接收到的缓冲区直接交给非特权用户空间应用程序。如果网络设备没有覆盖此缓冲区中的字节，则可能将其他进程释放过的内存泄漏到应用程序中。

idx字段表示驱动程序将下一个描述符项放入环中的位置（以队列大小为模）。它从0开始，并递增。

**注：**旧版[Virtio PCI草案]将这些结构称为vring\_used和vring\_used\_elem，并将常量称为VRING\_USED\_F\_NO\_NOTIFY，但布局和值相同。

##### 2.4.8.1 旧版接口：虚拟队列已用环

从历史角度看，许多驱动程序忽略了len值，因此导致许多设备对len值的设置不正确。因此在使用旧版接口时，通常最好忽略环项中的len值。每种设备类型都对应列出了其已知的特定问题。

##### 2.4.8.2 设备要求：虚拟队列已用环

设备**必须**在更新idx之前设置len值。

在更新已用idx之前，设备**必须**从第一个可写设备缓冲区开始，向描述符写入最少len字节。

设备**可能**向描述符写超过len字节。

**注：**存在设备可能不知道缓冲区的哪部分被写入这样的潜在错误情况。这就是为什么len允许被低估：这比未初始化的内存没有被覆盖时，驱动程序相信它已经被覆盖来的要好。

##### 2.4.8.3 驱动要求：虚拟队列已用环

驱动程序**不得**对超过第一个len字节的设备可写缓冲区中的数据进行假设，而**应该**无视这些数据。

#### 2.4.9 虚拟队列通知抑制

设备可以通过类似于驱动程序抑制中断的方式来抑制通知，详见2.4.7节。设备通过操作已用环中的flags或avail\_event进行通知抑制，方法与驱动程序在可用环中操作flags或used\_event的方式相同。

##### 2.4.9.1 驱动要求：虚拟队列通知抑制

驱动程序**必须**在分配已用环时将已用环初始化为0。

如果VIRTIO\_F\_EVENT\_IDX功能位没有被协商：

·驱动程序**必须**无视avail\_event值。

·当驱动程序向可用环中写入描述符索引后：

-如果flags为1，驱动程序**不应**发送通知。

-如果flags为0，驱动程序**必须**发送通知。

否则，如果VIRTIO\_F\_EVENT\_IDX功能已经协商：

·驱动程序**必须**无视flags的低位。

·当驱动程序向可用环中写入描述符索引后：

-如果可用环中的idx字段（它确定了描述符索引的放置位置）与avail\_event相等，那么驱动程序**必须**发送通知。

-否则驱动程序**不应**发送通知。

##### 2.4.9.2 驱动要求：虚拟队列通知抑制

如果VIRTIO\_F\_EVENT\_IDX功能没有被协商：

·设备**必须**将flags置位为0或1。

·设备可以将flags置位为1来告知驱动程序不需要通知。

否则，如果VIRTIO\_F\_EVENT\_IDX功能被协商：

·设备**必须**将flags置位为0。

·设备**可以**使用avail\_event告知驱动程序不需要通知直到驱动程序将具有avail\_event指定的索引项写入可用环（等效地，直到可用环中的idx达到值avail\_event+1）。

设备**必须**处理来自驱动程序的伪通知。

##### 2.4.10 操作虚拟队列的帮助

Linux内核源代码以更加好用的形式包含了以上定义，以及辅助例程，位于include/uapi/linux/virtio\_ring.h中。这是由IBM和Red Hat根据（3条款）BSD许可证明确许可的，因此它可以在其他项目中自由使用，且在（有些许改变）virtio\_queue.h中重新处理。

## 3 一般初始化和设备操作

我们从设备初始化的概述开始，然后展开设备的详细信息以及如何执行每个步骤。该部分最好与总线特定部分一起阅读，该部分描述如何与特定设备通信。

### 3.1 设备初始化

#### 3.1.1 驱动程序要求:设备初始化

驱动程序必须依照这个规则去初始化一个设备：

1. 重启设备。
2. 设置ACKNOWLEDGE 状态位：客户机操作系统已注意该设备。
3. 设置DRIVER 状态位：客户机操作系统怎样去驱动该设备。
4. 读取设备特征位，并将OS和驱动程序理解的特征位的子集写入设备。在此步骤中，驱动程序可以（MAY）读取（但不能写入（MUST NOT））设备特定的配置在接受设备之前检查它是否支持该设备的字段
5. 设置FEATURES\_OK状态位。在此步骤之后，驱动程序（MUST NOT）不得接受新的特征功能位。
6. 重新读取设备状态以确保 “FEATURES\_OK”位仍然为设置状态。否则，设备不支持我们的功能子集，并且设备不可用。
7. 执行设备特定的设置，包括发现设备的virtqueues、可选的per\_bus设置，读取并且可能写入设备的virtio配置空间，以及virtqueues的数量。
8. 设置 DRIVER\_OK状态位。此时设备处于“live”状态。

如果这些步骤任何一步出现了无法恢复的错误，驱动程序应（SHOULD）设置故障(FAILES)状态位，以指示它已放弃设备（如果需要，它可以稍后重置设备以重新启动）。在这种情况下，驱动程序不能（MUST NOT）继续初始化。

在设置 “DRIVER\_OK”位之前，驱动程序不得（MUST NOT）通知设备。

#### 3.1.2 旧接口：设备初始化

传统设备不支持功能“FEATURES\_OK”位，因此设备没法优雅地指示不支持的功能组合。他们也没有提供一个明确的机制来结束功能协商，这意味着设备在首次使用时最终确定了功能，并且不能引入任何功能，这从根本上改变了设备的初始操作。传统的驱动程序实现通常在设置DRIVER\_OK”位之前使用设备，有时甚至在将特征位写入设备之前。结果省略了步骤5和6，将步骤4、7和8合并。

因此，在使用传统接口时：

•过渡驱动程序必须（MUST）执行3.1中所述的初始化顺序，但省略了第5步和第6步。

•在步骤4之前，过渡设备必须（MUST）支持驱动程序写入设备配置字段。

•在步骤8之前，过渡设备必须支持驱动程序使用该设备。

### 3.2 设备操作

设备操作有两个部分：为设备提供新的缓冲区（buffers），以及处理设备中使用的缓冲区(buffers)。

**注意**：比如，最简单的virtio网络设备有两个virtqueue：传输virtqueue和接收virtqueue。驱动程序将传出（设备可读device-readable）数据包添加到传输virtqueue，然后在使用后释放它们。类似地，传入（设备可写device-writable）缓冲区被添加到接收virtqueue，并在使用后进行处理。

#### 3.2.1 为设备提供缓冲区

驱动程序向设备的virtqueues中一个virtqueue提供缓冲区，如下所示：

1、驱动程序将缓冲区放入描述符表中的自由描述符中，并根据需要链接（见  
2.4.5 The Virtqueue Descriptor Table）。

2、驱动程序将描述符链头的索引放入可用环中的下一环形入口（ring）。

3、如果可以进行批处理，则可以重复执行步骤1和2。

4、驱动程序执行适当的内存屏障（memory barrier）程序，以确保设备能看到更新的描述符表，以及下一步之前的可用环（available ring）。

5、可用的***idx***随着添加到可用环中的描述符链头的数量增加而增加。

6、驱动程序执行适当的内存屏障（memory barrier）程序，以确保在通知取消之前更新***idx***字段。

7、如果不取消通知，驱动程序会通知新的可用缓冲区设备。

请注意，上面的代码没有针对可用的环缓冲区缠绕采取预防措施：并且这也是不可能的，因为环缓冲区的大小与描述符表的大小相同，所以步骤1将防止出现这种情况。此外，最大队列的大小为32768（2的最大指数，适合16位），因此16位***idx***值始终可以区分一个完整缓冲区和空缓冲区。

下面是每个阶段的详细要求。

##### 3.2.1.1 将缓冲区放入描述符表中

缓冲区由零个或多个可读设备的物理相邻元素组成，后跟零个或多个物理相邻的可写设备元素（每个至少含有一个元素）。该算法将其映射到描述符表中以形成描述符链。

对于每一个缓冲区元素b:

1. 获取下一个自由的描述符表入口，d；
2. 设置d.addr 到b的起始物理地址；
3. 设置d.len为b的长度；
4. 如果b是设备可写，设置d.flags为VIRTQ\_DESC\_F\_WRITE,否则设为0；
5. 如果有一个a缓冲区元素在这之后：
6. 设置d.next 为下一自由描述符元素的索引。
7. 在d.flags中设置VIRTQ\_DESC\_F\_NEXT位;

在实践中，d.next通常用于链接自由描述符，并且在开始映射之前保持一个单独的计数以检查是否有足够的自由描述符。

##### 3.2.1.2 更新可用环（Aailable Ring）

描述符链头是上述算法中的第一个d，即引用缓冲区第一部分的描述符表项的索引。一个单纯的驱动程序实现可以（MAY）执行以下操作（假定与小字节进行了适当的转换）：



但是，一般来说，驱动程序可能会在更新***idx***之前添加许多描述符链（此时，它们对设备可见），因此通常会保留一个计数器来记录驱动程序添加的数量：

##### 3.2.1.3 更新*idx*

***idx***始终递增，并以65536为一自然循环。



一旦驱动程序更新了可用的***idx***，就暴露了描述符及其内容。设备可以（MAY）立即访问所创建的描述符链及其引用的内存。

##### 3.2.1.3.1驱动程序要求：更新idx

驱动程序必须在***idx***更新之前执行适当的内存屏障，以确保设备看到最新的副本。

###### 3.2.1.4 通知设备

设备通知的实际方法是特定于总线的，但一般来说可能会很昂贵。因此，如第2.4.9节所述，如果设备不需要这些通知，它可能(MAY)会禁止这些通知。

在检查是否禁止通知之前，驱动程序必须小心地公开新的***idx***值。

##### 3.2.1.4.1驱动程序要求：通知设备

在读取标志(flags)或可用事件(avail\_event)之前，驱动程序必须(MUST)执行适当的内存屏障，以避免丢失通知。

#### 3.2.2 从设备接收已用缓冲区

一旦设备使用了描述符所指的缓冲区（读写缓冲区或者两者的一部分，取决于virtqueue和设备的性质），它就会中断驱动程序，如第2.4.7节所述。

**注意**：为了获得最佳性能，驱动程序可能（MAY）会在处理使用的环时禁用中断，但要注意清空环和重新启用中断之间缺少中断的问题。这通常通过重新启用中断后重新检查更多使用的缓冲区来处理：

#### 3.2.3 设备配置更改通知

对于可以更改设备特定配置信息的设备，当发生设备特定配置更改时，将发送中断。此外，此中断由设备设置位“DEVICE\_NEEDS\_RESET”触发（见2.1.2）。

### 3.3设备清理

一旦驱动程序设置了DRIVER\_OK状态位，设备的所有配置virtqueue都将被视为“live”的。重置设备后，设备的虚拟队列均不“live”。

#### 3.3.1驱动程序要求：设备清理

驱动程序不得（MUST NOT）更改已在live virtqueue的可用环（在已用环中未被设备标记使用的）中公开的描述符表入口数。

驱动程序不能（MUST NOT）减少live virtqueue上的可用***idx***（即，无法“释放”缓冲区）。

因此，在删除暴露的缓冲区之前，驱动程序必须确保virtqueue不活动（通过设备重置的方式）。

## 4 Virtio传输选项

Virtio可以使用许多不同种类的总线，因此标准分为一般的virtio以及特定于总线的部分。

### 4.1 PCI总线下的Virtio

Virtio设备通常实现为PCI设备。

Virtio设备可以时限为任意类型的PCI设备：常规PCI或PCI Express设备。为了保证设计符合最新级别的要求，请参阅PCI-SIG的主页http://www. pcisig.com for any approved changes。

#### 4.1.1 设备要求：PCI总线下的Virtio

使用PIC总线下的Virtio的Virtio设备必须向访客开放一个符合相应PCI规范的规范接口：分别是[PCI]与[PCIe]。

#### 4.1.2 发现PCI设备

任何带有PCI供应商ID 0x1AF4以及PCI设备ID 0x1000到0x107F的PCI设备都是virtio设备。这个范围内的实际值表示设备支持哪种virtio设备。PCI设备ID通过在Virtio设备ID加上0x1040来计算，如第5节所示。此外，设备可以使用范围为0x1000到0x103F的过渡PCI设备ID，具体取决于设备类型。

##### 4.1.2.1 设备要求：发现PCI设备

设备**必须**拥有PCI供应商ID 01AF4。设备**必须**具有如第5部分所示的通过在Virtio设备ID加上0x1040计算的PCI设备ID，或具有取决于设备类型过渡PCI设备ID，如下所示：

|  |  |
| --- | --- |
| 过渡PCI设备ID | Virtio设备 |
| 0x1000 | 网卡 |
| 0x1001 | 块设备 |
| 0x1002 | 内存气球（传统） |
| 0x1003 | 控制台 |
| 0x1004 | SCSI主机 |
| 0x1005 | 熵源 |
| 0x1009 | 9P传输 |

例如，具有Virtio设备ID 1的网卡设备的PCI设备ID 0x1041或过渡PCI设备ID 0X1000。

PCI子系统供应商ID与PCI子系统设备ID**可能**反映环境的PCI供应商和设备ID（由驱动程序提供信息）。

非过渡设备**应该**具有范围在0x1040到0x107f之间的PCI设备ID。非过渡设备**应该**具有1或更高的PCI版本ID。非过渡设备**应该**具有0x40或者更高的PCI子系统设备ID。

这是为了减少旧版驱动程序试图驱动设备情况的发生。

##### 4.1.2.2 驱动要求：发现PCI设备

驱动程序**必须**匹配具有PCI供应商ID 0x1AF4，以及如第5接所示的通过在Virtio设备ID加上0x1040计算的范围在0x1040到0x107f之间的PCI设备ID的设备。在4.1.2节中所列出的设备类型的驱动程序**必须**匹配PCI供应商ID 0x1AF4和4.1.2节中指出的过渡PCI设备ID。

驱动程序**必须**匹配任何PCI修订版ID值。设备**可以**匹配任何PCI子系统供应商ID以及PCI子系统设备ID值。

##### 4.1.2.3 旧版接口：关于发现PCI设备的注意事项

过渡设备的PCI修订版ID**必须**为0。过度设备**必须**具有如第5节所示的与Virtio设备ID匹配的设备ID。过度设备**必须**具有范围在0x1000到0x103f内的过度PCI设备ID。

这是为了匹配旧版的驱动程序。

#### 4.1.3 PCI设备布局

根据Virtio 结构的PIC功能规定，通过I/O与/或内存区域配置器件（但请参见4.1.4.7通过PCI配置空间访问）。

设备配置区域中存在着不同大小的字段。所有64位，32位和16位字段均为小端格式。64位字段将被视为两个32位字段，低32位部分后跟高32位部分。

##### 4.1.3.1 驱动要求：PCI设备布局

对于器件配置访问，驱动程序**必须**对8位宽字段使用8位宽访问，对16位宽字段使用16位宽和对齐访问，对32位和64位使用32位宽和对齐访问。对于64位字段，驱动程序**可以**独立访问字段的高32位部分和低32位部分。

##### 4.1.3.2 设备要求：PCI设备布局

对于64位器件配置字段，设备**必须**允许驱动程序独立访问该字段的高32位部分和低32位部分。

#### 4.1.4 Virtio结构的PCI功能

virtio设备配置布局包括几个结构：

·常规结构

·通知

·ISR状态

·特定于设备的配置（可选）

·PCI配置访问

每个结构可以由属于该功能的基地址寄存器（BAR）映射，或者通过PCI配置空间中的特殊VIRTIO\_PCI\_CAP\_PCI\_CFG字段访问。使用位于设备PCI配置空间中的功能列表中的特定于供应商的PCI功能来指定每个结构的位置。这种virtio结构功能使用小端字节序格式；除非另有说明，否则对驱动程序来说，所有字段对都是只读的：

struct virtio\_pci\_cap {

u8 cap\_vndr; /\* Generic PCI field: PCI\_CAP\_ID\_VNDR \*/

u8 cap\_next; /\* Generic PCI field: next ptr. \*/

u8 cap\_len; /\* Generic PCI field: capability length \*/

u8 cfg\_type; /\* Identifies the structure. \*/

u8 bar; /\* Where to find it. \*/

u8 padding[3]; /\* Pad to full dword. \*/

le32 offset; /\* Offset within bar. \*/

le32 length; /\* Length of the structure, in bytes. \*/

};

根据cfg\_type，此结构可以后跟额外数据，如下所述。

字段解释如下：

**cap\_vndr** 0x09；标识特定于供应商的功能

**cap\_next** 链接到PCI配置空间中功能列表中的下一个功能

**cap\_len** 此功能结构的长度，包括整个struct virtio\_pci\_cap和额外数据（如果有）。该长度**可以**包括填充或驱动程序未使用的字段。

**cfg\_type** 根据下表确定结构：

/\* Common configuration \*/

#define VIRTIO\_PCI\_CAP\_COMMON\_CFG 1

/\* Notifications \*/

#define VIRTIO\_PCI\_CAP\_NOTIFY\_CFG 2

/\* ISR Status \*/

#define VIRTIO\_PCI\_CAP\_ISR\_CFG 3

/\* Device specific configuration \*/

#define VIRTIO\_PCI\_CAP\_DEVICE\_CFG 4

/\* PCI configuration access \*/

#define VIRTIO\_PCI\_CAP\_PCI\_CFG 5

任何其他值均保留，供将来使用。

每个结构在下面单独详述。

设备可以提供多种类型的结构-这使得设备可以将多个接公开向给驱动程序。功能列表中功能的顺序指定设备建议的首选顺序。

**注：**例如，在某些虚拟机管理程序上，使用IO访问的通知比内存访问更快。在这种情况下，设备将公开两个功能， cfg\_type随之设置为VIRTIO\_PCI\_CAP\_NOTIFY\_CFG：第一个用于寻址I / O BAR，第二个用于寻址存储器BAR。在此示例中，如果I / O资源可用，驱动程序将使用I / O BAR，并在I / O资源不可用时返回内存BAR。

**bar** 0x0到0x5的值指定属于PCI配置空间中10h开始的函数的基址寄存器（BAR），并用于将结构映射到存储器或I / O空间。BAR允许为32位或64位，它可以映射存储空间或I / O空间。

任何其他值均保留，供将来使用。

**offset**表示结构相对于与BAR关联的基址的开始位置。offset的对齐要求在下面的每个 特定与结构 部分中指出。

**length** 表示结构的长度。

结构可以包括填充，或驱动程序未使用的字段，或将来的扩展。

**注：**例如，未来的设备可能呈现几MB的大型结构大小。由于当前设备从不使用大小超过4KB的结构，因此驱动程序**可以**将映射的结构大小限制为例如 4KBytes（因此在前4KBytes之后忽略结构的一部分）以允许与这些设备的向前兼容性而不损失功能并且不浪费资源。

##### 4.1.4.1 驱动要求：Virtio结构的PCI功能

驱动程序**必须**忽略任何特定于供应商的具有保留的cfg\_type值。

驱动程序**应该**使用他们可以支持的每种virtio结构类型的第一个实例。

驱动程序**必须**接受大小大于此处指定的cap\_len值。

驱动程序**必须**忽略具有保留bar值的任何特定于供应商的功能结构。

驱动程序**应该**只映射配置结构大小足够设备操作的部分。驱动程序**必须**处理长度大到意料之外的情况，但可以检查长度是否足够设备操作。

除了具有如4.1.4.7.2中所述的cap\_type VIRTIO\_PCI\_CAP\_PCI\_CFG的那些字段之外，驱动程序**不得**写入功能结构的任何字段

##### 4.1.4.2 设备要求：Virtio结构的PCI功能

设备**必须**包含cap\_len中的任何额外数据（从cap\_vndr字段的开头到额外数据字段的末尾，如果有的话）。设备**可以**向其后的任何结构附加额外的数据或填充。

如果设备呈现多个相同类型的结构，它**应该**从最优（第一）到最劣（最后）排序。

##### 4.1.4.3 常规配置结构布局

常规配置结构可在VIRTIO\_PCI\_CAP\_COMMON\_CFG功能中的bar与offset找到，它的布局如下

struct virtio\_pci\_common\_cfg {

/\* About the whole device. \*/

le32 device\_feature\_select; /\* read-write \*/

le32 device\_feature; /\* read-only for driver \*/

le32 driver\_feature\_select; /\* read-write \*/

le32 driver\_feature; /\* read-write \*/

le16 msix\_config; /\* read-write \*/

le16 num\_queues; /\* read-only for driver \*/

u8 device\_status; /\* read-write \*/

u8 config\_generation; /\* read-only for driver \*/

/\* About a specific virtqueue. \*/

le16 queue\_select; /\* read-write \*/

le16 queue\_size; /\* read-write, power of 2, or 0. \*/

le16 queue\_msix\_vector; /\* read-write \*/

le16 queue\_enable; /\* read-write \*/

le16 queue\_notify\_off; /\* read-only for driver \*/

le64 queue\_desc; /\* read-write \*/

le64 queue\_avail; /\* read-write \*/

le64 queue\_used; /\* read-write \*/

};

**device\_feature\_selsct** 驱动程序使用它来选择device\_feature显示的特征位。值0x0选择功能位0到31,0x1选择功能位32到63，等等。

**device\_feature** 设备使用它来报告它向驱动程序提供的功能位：驱动程序写入device\_feature\_select来选择显示哪些功能位。

**driver\_feature\_select** 驱动程序使用它来选择driver\_feature显示的功能位。值0x0选择功能位0到31,0x1选择功能位32到63，等等。

**driver\_feature** 驱动程序将其写入，用于接受设备提供的功能位。驱动程序功能位由driver\_feature\_select选择。

**config\_msix\_vector** 驱动程序为MSI-X设置配置向量。

**num\_queues** 设备指定此处支持的最大虚拟队列数。

**device\_status** 驱动程序在此处写入设备状态（参见2.1）。将0写入此字段会复位设备。

**config\_generation** 配置原子性值。每次配置发生明显的改变时，设备都会更改此设置。

**queue\_select** 队列选择。驱动程序选择接下来的字段引用的虚拟队列。

**queue\_size** 队列大小。在复位时指定管理程序支持的最大队列大小。这可以由驱动程序修改以减少内存需求。0表示队列不可用。

**queue\_msix\_vector** 驱动程序使用它来指定MSI-X的队列向量。

**queue\_enable** 驱动程序使用它来有选择地阻止设备执行来自此virtqueue的请求。1-启用; 0-禁用。

**queue\_notify\_off** 驱动程序读取此内容来计算此虚拟队列所在的通知结构的起始偏移量。

注：这不是以字节为单位的偏移量。见下文4.1.4.4。

**queue\_desc** 驱动程序在此处写入描述符表的物理地址。见2.4节。

**queue\_avail** 驱动程序在此处写入可用环的物理地址。见2.4节。

**queue\_used** 驱动程序在此处写入已用环的物理地址。见2.4节。

**4.1.4.3.1 设备要求：常规配置结构布局**

offset**必须**以4字节对齐。

设备**必须**至少具有一个常规配置功能。

设备**必须**提供它在device\_feature中提供的功能位，从位device\_feature\_select\*32开始，用于驱动程序写入的任何device\_feature\_select。

**注：**由于此处定义的功能不超过63**，**因此这意味着它将为除0或1以外的任何device\_feature\_select呈现0。

设备**必须**提供驱动程序在driver\_feature中写入的任何有效特征位，从位driver\_ feature\_select\*32开始，用于驱动程序写入的任何driver\_feature\_select。设备**必须**提供驱动程序写入的任何有效的功能位，其为device\_feature位对应的子集。设备**可以**提供驱动程序写入的无效位。

**注：**这意味着设备可以忽略它从未提供的功能位的写入，并且在读取时只显示0。或者它只能反映驱动程序写的内容（但是当驱动程序设置FEATURES\_OK时仍然需要检查它们）。

**注：**根据每个3.1.1，驱动程序无论如何都不应该写无效位，但会尝试处理它。

在驱动程序读取特定于设备的配置值之后，设备必须提供更改过的config\_generation，该配置值自上次读取设备特定配置的任何部分后已发生更改。

**注：**由于config\_generation是一个8位值，在每次配置中单纯地对其进行递增都可能因wrap而违反这个要求。当其发生变化时设置一个内部标志，且如果驱动程序从特定于设备的配置中读取了这个标志，则时config\_generation递增并清空这个标志，这样做会更加好。

设备**必须**在0被写入device\_status时复位，并在完成后在device\_status中显示0。

设备**必须**在复位时在queue\_enable中显示0。

如果对应于当前queue\_select的virtqueue不可用，则设备**必须**在queue\_size中呈现0。

###### 4.1.4.3.2 设备要求：常规配置结构布局

驱动程序**不得**写入device\_feature，num\_queues，config\_generation或queue\_notify\_off。

驱动程序**不得**向queue\_size写入非2的幂的值。

在使用queue\_enable使能virtqueue之前，驱动程序**必须**配置其他virtqueue字段。

将0写入device\_status后，驱动程序**必须**在重新初始化设备之前等待device\_status读取后使之返回0。

驱动程序**不得**向queue\_enable写入0。

##### 4.1.4.4 通知结构布局

通知地址通过VIRTIO\_PCI\_CAP\_NITIFY\_CFG功能进行查找。此功能后紧跟一个附加字段，如下所示：

struct virtio\_pci\_notify\_cap {

struct virtio\_pci\_cap cap;

le32 notify\_off\_multiplier; /\* Multiplier for queue\_notify\_off. \*/

};

notify\_off\_multiplier与queue\_notify\_off结合使用，可以在BAR中为虚拟队列派生Queue Notify地址：

cap.offset + queue\_notify\_off \* notify\_off\_multiplier

cap.offset和notify\_off\_multiplier取自上面的通知功能结构，queue\_notify\_off取自常规配置结构。

**注：**例如，如果notifier\_off\_multiplier为0，则设备对所有队列使用相同的队列通知地址。

###### 4.1.4.4.1 设备要求：通知功能

设备**必须**提供至少一个通知功能。

cap.offset**必须**是2字节对齐的。

设备**必须**要么使notify\_off\_multiplier为偶数幂2，要么使notify\_off\_multiplier为0。

设备提供的值cap.length**必须**至少为2，并且**必须**拥有支持所有可能配置中所有支持的队列的队列通知偏移的大小。

对于所有队列，设备提供的值cap.length**必须**满足：

cap.length >= queue\_notify\_off \* notify\_off\_multiplier + 2

##### 4.1.4.5 ISR状态功能

VIRTIO\_PCI\_CAP\_ISR\_CFG功能至少指一个字节，其中包含用于INT＃x中断处理的8位ISR状态字段。

ISR状态的offset没有对齐要求。

ISR位允许器件区分特定于器件的配置更改中断和正常的virtqueue中断：

|  |  |  |  |
| --- | --- | --- | --- |
| 位 | 0 | 1 | 2到31 |
| 目的 | 队列中断 | 设备配置中断 | 保留 |

为避免额外访问，只需读取该寄存器即可将其复位为0并使器件取消置位中断。

通过这种方式，驱动程序读取ISR状态会导致设备取消置位中断。

具体如何使用，请参见第4.1.5.3和4.1.5.4节。

###### 4.1.4.5.1 设备要求：ISR状态功能

设备**必须**至少提供一个VIRTIO\_PCI\_CAP\_ISR\_CFG功能。

在向驱动程序发送设备配置更改通知之前，设备**必须**将设备配置中断位设置为ISR状态。

如果禁用了MSI-X功能，则在向驱动程序发送虚拟队列通知之前，设备**必须**将ISR状态中的队列中断位置1。

如果禁用了MSI-X功能，则在向驱动程序发送虚拟队列通知之前，设备**必须**将ISR状态中的队列中断位置1。

如果禁用了MSI-X功能，则器件必须将器件的PCI配置头中的PCI状态寄存器中的中断状态位设置为器件ISR状态中所有位的逻辑或。

然后，除非根据标准PCI规则[PCI]进行屏蔽，器件将置位/取消置位INT＃x中断。

在驱动程序读取时，设备必须将ISR状态重置为0。

###### 4.1.4.5.2 驱动要求：ISR状态功能

如果启用了MSI-X功能，则驱动程序在检测到队列中断时**不应**访问ISR状态。

##### 4.1.4.6 特定于设备的配置

设备必须为具有特定于设备配置的任何设备类型提供至少一个VIRTIO\_PCI\_CAP\_DEVICE\_CFG功能。

###### 4.1.4.6.1 设备要求：特定于设备的配置

特定于器件的配置的offset必须是4字节对齐的。

##### 4.1.4.7 PCI配置访问功能

VIRTIO\_PCI\_CAP\_PCI\_CFG功能为常规配置，通知，ISR和特定于设备的配置区域创建替代（并且可能是次优的）访问方法。

该功能紧接着是一个额外的字段，如下所示：

struct virtio\_pci\_cfg\_cap {

struct virtio\_pci\_cap cap;

u8 pci\_cfg\_data[4]; /\* Data for BAR access. \*/

};

cap.bar，cap.length，cap.offset和pci\_cfg\_data字段是驱动程序的读写（RW）字段。

为了访问设备区域，驱动程序将对功能结构进行写入（即在PCI配置空间内），如下所示：

·驱动程序通过写入cap.bar来设置要访问的BAR。

·驱动程序通过将1或2或4写入cap.length来设置访问的大小。

·驱动程序通过写入cap.offset在BAR中设置偏移量。

此时，pci\_cfg\_data将在偏移量cap.offset的给定cap.bar中提供一个大小为cap.length的窗口。

###### 4.1.4.7.1 设备要求：PCI配置访问功能

设备必须至少提供一个VIRTIO\_PCI\_CAP\_PCI\_CFG功能。

在检测到对pci\_cfg\_data的驱动程序写访问时，设备必须使用来自pci\_cfg\_data的第一个cap.length字节，在cap.bar选择的BAR处的偏移cap.offset处执行写访问。

在检测到对pci\_cfg\_data的驱动程序读取访问时，设备必须在cap.bar选择的BAR处的偏移cap.offset处执行长度为cap.length的读访问，并将第一个cap.length字节存储在pci\_cfg\_data中。

###### 4.1.4.7.2 驱动要求：PCI配置访问功能

驱动程序**不得**向cap.offset写入非cap.length的倍数（即所有访问必须对齐）。

驱动程序**不得**读取或写入pci\_cfg\_data，除非cap.bar，cap.length和cap.offset将cap.length字节设置在由VIRTIO\_PCI\_CAP\_PCI\_CFG以外的其他Virtio 结构的 PCI 功能指定的BAR范围内。

##### 4.1.4.8 旧版接口：关于PCI设备布局的注意事项

过渡设备**必须**在PCI设备的第一个I / O区域中的BAR0中的旧版配置结构中提供配置寄存器的部分，如下所述。使用旧版接口时，过渡驱动程序**必须**使用PCI设备第一个I / O区域中BAR0中的旧版配置结构，如下所述。

当使用旧版接口时，驱动程序**可以**使用任何宽度访问访问特定于设备的配置区域，并且过渡设备**必须**提供与使用“自然”访问方法访问时相同结果的驱动程序（即32位访问32位字段，等等）。

注意，由于virtio常规配置结构是PCI（即小端）字节序，因此这是可能的，而当使用旧版接口时，特定于设备的配置区域被编码在客户的本地端（这种区别适用）。

当通过旧版接口使用时，virtio常规配置结构如下所示：

如果为设备启用了MSI-X，则会立即在此标头后面添加两个附加字段：

|  |  |  |
| --- | --- | --- |
| 位 | 16 | 16 |
| 读/ 写 | R+W | R+W |
| 目的(MSI-X) | Config\_msix\_vector | Queue\_msix\_vector |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 位 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| 读/写 | R | R+W | R+W | R | R+W | R+W | R+W | R |
| 目的 | 设备功能位0:31 | 设备功能位0:31 | 队列  地址 | queue\_-  size | queue\_-  slelct | 队列  提示 | 设备  状态 | ISR  状态 |

注意：启用MSI-X功能时，特定于设备的配置从virtio常规配置结构结构中的字节偏移量24开始。如果未启用MSI-X功能，则特定于设备的配置将从virtio标头中的字节偏移量20开始，即， 一旦你启用了设备上的MSI-X，其他字段就会移动。如果再次关闭它们，它们会向后移动！

任何特定于设备的配置空间都紧跟在以下常规标头之后：

|  |  |  |
| --- | --- | --- |
| 位 | 特定于设备 | … |
| 读/写 | 特定于设备 |
| 目的 | 特定于设备 |

使用旧版接口访问特定于设备的配置空间时，过渡驱动程序**必须**在通用标头后面的偏移处访问特定于设备的配置空间。

使用旧版接口时，过渡设备**必须**在紧跟通用标头之后的偏移处提供特定于设备的配置空间。

请注意，只有功能位 0到31可通过旧版接口访问。当通过旧版接口使用时，过渡设备**必须**假设驱动程序没有理会功能位32至63。

由于旧设备没有config\_generation字段，请参阅2.3.4旧版接口：设备配置空间以获取解决方法。

##### 4.1.4.9 具有旧版驱动程序的非过渡设备：关于PCI设备布局的注意事项

所有已知的旧版驱动程序都会检查PCI版本或设备和供应商ID，因此不会尝试驱动非过渡设备。

有缺陷的旧版驱动程序可能会错误地尝试驱动非过渡设备。如果需要支持这样的驱动程序（而不是修复bug），以下是检测和处理它们的推荐方法。

**注：**目前尚不知道这种有缺陷的驱动程序在生产中进行过使用。

4.1.4.9.0.1 设备要求：具有旧版驱动程序的非过渡设备

非过渡设备，在已知存在具有相同ID（包括PCI版本，设备和供应商ID）的旧设备的旧驱动程序的平台上，**应该**采取以下步骤，以使旧驱动程序在尝试驱动它们时正常地失败：

1.在BAR0中提供I/O BAR，并

2. 通过对每个BAR提供零并忽略写入来响应对BAR0的偏移18（对应于旧版布局中的设备状态寄存器）的单字节零写入。

#### 4.1.5 特定于PCI的初始化与设备操作

##### 4.1.5.1 设备初始化

记录了设备初始化期间执行的特定于PCI的步骤。

###### 4.1.5.1.1 Virtio设备配置布局检测

作为设备初始化的先决条件，驱动程序使用Virtio Structure PCI功能检测virtio配置布局，扫描PCI功能列表，详见4.1.4

4.1.5.1.1.1 旧版接口：关于设备布局检测的注意事项

旧版驱动程序跳过设备布局检测步骤，并无条件对I/O空间中的BAR0中的旧版设备配置空间进行假设。

旧版设备的功能列表中没有Virtio PCI功能。

因此：

过渡设备**必须**在BAR0中的I/O空间中公开旧版接口。

过渡驱动程序**必须**在功能列表中查找Virtio PCI功能。如果其不存在，则驱动程序必须假设为旧版设备，并通过旧版接口使用它。

非过渡驱动程序**必须**在功能列表中查找Virtio PCI功能。如果其不存在，驱动程序必须假设为旧版设备，并正常地失败。

###### 4.1.5.1.2 MSI-X向量配置

当MSI-X功能存在并在设备中启用时（通过标准PCI配置空间），config\_msix\_vector和queue\_msix\_vector用于将配置更改和队列中断映射到MSI-X向量。在这种情况下，ISR状态未启用。

通过将配置改变/选择的队列事件触发的中断分别映射到相应的MSI-X向量，来将有效的MSI-X表项号0到0x7FF写入config\_msix\_vector / queue\_msix\_vector。

要禁用某事件类型的中断，驱动程序通过写入一个特殊的NO\_VECTOR值来取消映射该事件：

/\* Vector value used to disable MSI for queue \*/

#define VIRTIO\_MSI\_NO\_VECTOR 0xffff

请注意，将事件映射到向量可能需要设备分配内部设备资源，因此可能会失败。

4.1.5.1.2.1 设备要求：MSI-X向量配置

具有MSI-X功能的设备**应该**支持至少2个和最多0x800个MSI-X向量。设备**必须**按照[PCI]中的规定报告MSI-X能力中表格大小中支持的向量数量。设备应该将报告的MSI-X表大小字段限制为可能有益于系统性能的值。

**注：**例如：例如，不希望以高频率发送中断的设备可能仅指定2个MSI-X向量。

设备**必须**支持将任何事件类型映射到任何有效向量0到MSI-X表大小。 设备**必须**支持取消任何事件类型的映射。

在读取config\_msix\_vector / queue\_msix\_vector时，设备**必须**返回映射到给定事件的向量（如果未映射，则返回NO\_VECTOR）。设备**必须**在复位时取消映射所有队列和配置更改事件。

设备**不应**使将事件映射到向量失败，除非设备不可能满足映射请求。当相关的config\_msix\_vector / queue\_msix\_vector字段被读取时，设备必须通过返回NO\_VECTOR值来报告映射失败。

4.1.5.1.2.2 驱动要求：MSI-X向量配置

驱动程序**必须**支持具有MSI-X表大小0到0x7FF的任何设备。对于仅支持一个MSI-X向量（MSI-X表大小= 0）的设备，驱动程序**可以**使用INT＃x中断。

驱动程序**可以**将表大小解释为来自设备的提示，以使用建议的MSI-X向量数。

如MSI-X Capability中的表大小所报告，驱动程序不得尝试将事件映射到设备支持的MSI-X表外的向量。

在将事件映射到向量之后，驱动程序必须通过读取Vector字段值来验证成功：在成功时返回先前写入的值，失败时返回NO\_VECTOR。如果检测到映射失败，则驱动程序可以使用较少的向量重试映射，禁用MSI-X或报告设备故障。

###### 4.1.5.1.3 虚拟队列配置

由于设备可以为批量数据传输提供零或更多的优点，因此驱动程序需要将它们配置为特定于设备的配置的一部分。

对于设备具有的每个virtqueue，驱动程序通常按如下方式执行此操作：

1.将virtqueue索引（第一个队列为0）写入queue\_select。

2.从queue\_size中读取virtqueue大小。这控制着虚拟队列的大小（见2.4虚拟队列）。如果此字段为0，则virtqueue不存在。

3.（可选）选择较小的virtqueue大小并将其写入queue\_size。

4.为连续物理内存中的virtqueue分配和归零描述符表，可用环和已用环。

5. 可选地，如果设备上存在并启用了MSI-X功能，请选择用于请求由virtqueue事件触发的中断的向量。将与此对应的MSI-X表条目号矢量写入queue\_msix\_vector。 读取queue\_msix\_vector：成功时，返回先前写入的值；失败时，返回NO\_VECTOR值。

4.1.5.1.3.1 旧版接口：关于虚拟队列配置的注意事项

使用旧接口时，队列布局遵循2.4.2传旧版口：关于Virtqueue布局的注释，对齐为4096。驱动程序将物理地址除以4096写入队列地址字段2。不存在协商队列大小的机制。

##### 4.1.5.2 通知设备

驱动程序通过将此虚拟队列的16位虚拟队列索引写入Queue Notify地址来通知设备。有关如何计算此地址，请参见4.1.4.4。

##### 4.1.5.3 来自设备的虚拟队列中断

如果虚拟队列需要中断，则设备通常会进行如下操作：

·如果MSI-X功能关闭：

1. 设置设备的ISR状态字段的低位。

2. 为设备发送适当的PCI中断。

·如果MSI-X功能启用：

1. 如果queue\_msix\_vector不是NO\_VECTOR，则设备需要相应的MSI-X中断消息，那么queue\_msix\_vector将设置MSI-X表条目号

###### 4.1.5.3.1 设备要求：来自设备的虚拟队列中断

如果启用了MSI-X功能且queue\_msix\_vector为虚拟队列的NO\_VECTOR，则设备不得为该虚拟队列提供中断。

##### 4.1.5.4 设备配置更改提示

某些virtio PCI设备可以更改设备配置状态，如设备的特定于设备的配置区域所反映的那样。在这种情况下：

·如果MSI-X功能关闭：

1. 设置设备的ISR状态字段的第二个低位。

2. 为设备发送适当的PCI中断。

·如果MSI-X功能开启：

1. 如果config\_msix\_vector不是NO\_VECTOR，则设备需要相应的MSI-X中断消息，config\_msix\_vector设置MSI-X表条目号。

2. 单个中断可以指示已使用一个或多个virtqueue并且配置空间已更改。

单个中断**可以**指示一个或多个虚拟队列已被使用并且配置空间已被更改。

###### 4.1.5.4.1 设备要求：关于设备配置更改的注意事项

如果启用了MSI-X功能且config\_msix\_vector为NO\_VECTOR，则设备**不得**为设备配置空间更改提供中断。

###### 4.1.5.4.2 驱动要求：关于设备配置更改的注意事项

驱动程序**必须**处理使用相同中断来指示设备配置空间更改以及一个或多个虚拟队列正在被使用的情况。

##### 4.1.5.5 驱动处理中断

驱动程序中断处理程序通常是：

·如果MSI-X功能被关闭：

-读取ISR状态字段，将其复位为零。

-如果设置了较低位：通过查看设备所有虚拟队列的已用环来查看需要服务的设备是否取得进展。

-如果设置了第二个低位：通过重新检查配置空间查看被更改的内容。

·如果MSI-X功能被启用：

-通过查看映射到设备的MSI-X向量的所有虚拟队列的已用环来查看需要服务的设备是否取得进展。

-如果MSI-X向量等于config\_msix\_vector，通过重新检查配置空间来查看被更改的内容。

### 4.2 MMIO下的Virtio

没有PCI支持的虚拟环境（嵌入式设备模型中的常见情况）可能使用简单的内存映射设备（“virtio-mmio”）而不是PCI设备。

内存映射的virtio设备行为基于PCI设备规范。因此，大多数操作（包括设备初始化，队列配置和缓冲区传输）几乎相同。以下各节介绍了现有差异。

#### 4.2.1 发现MMIO设备

与PCI不同，MMIO不提供通用的设备发现机制。对于每个设备来说，客户操作系统将需要知道寄存器的位置和使用的中断。本例中展示了建议使用的利用展平设备树绑定系统：

// EXAMPLE: virtio\_block device taking 512 bytes at 0x1e000, interrupt 42.

virtio\_block@1e000 {

compatible = "virtio,mmio";

reg = <0x1e000 0x200>;

interrupts = <42>;

}

#### 4.2.2 MMIS设备寄存器布局

MMIO virtio设备提供一组内存映射控制寄存器，后跟特定于器件的配置空间，如表4.1所示。

所有寄存器值在此以小端格式总结。

|  |  |
| --- | --- |
| 名字 | **功能** |
| 相对于基的偏置 | 描述 |
| 方向 |  |

|  |  |
| --- | --- |
| MagicValue | **Magic value** |
| 0x000 | 0x74726979（与“virt”字符串在小端格式下相等） |
| R |  |
| Version | **设备版本号** |
| 0x004 | 0x2 |
| R | **注**：旧版设备（见4.2.4旧版接口）使用0x1 |
| DeviceID | **Virtio子系统设备ID** |
| 0x008 | 有关可能的值，请参见5种设备类型值零（0x0）用于定义系统内存映射，其中占位符设备位于静态的，已知的的地址，根据用户的需要为其分配功能。 |
|  |
| R |
| VendorID | **Virtio子系统供应商ID** |
| 0x00c |  |
| R |  |
| DeviceFeatures | **代表设备支持的功能的标志** |
| 0x010 | 从该寄存器读取返回32个连续标志位，最低有效位取决于写入DeviceFeaturesSel的最后一个值。访问该寄存器会将DeviceFeaturesSel 32位返回到（DeviceFeaturesSel \* 32）+31，例如，如果DeviceFeaturesSel设置为0，则功能位为0到31；如果DeviceFeaturesSel设置为1，则功能位为32到63.另请参见2.2功能位。 |
| R |
| DeviceFearuresSel | **设备（主机）功能字选择** |
| 0x014 | 写入该寄存器，用于通过读取DeviceFeatures来选择一组32个器件功能位。 |
| W |
| DeviceFeatures | **此标志代表设备功能由驱动程序理解和激活** |
| 0x020 | 写入该寄存器会设置32个连续标志位，最低有效位取决于最后一个写入DriverFeaturesSel的值。访问该寄存器将DriverFeaturesSel \* 32位设置为（DriverFeaturesSel \* 32）+ 31，例如，如果DriverFeaturesSel设置为0，则功能位为0到31;如果DriverFeaturesSel设置为1，则功能位为32到63.另请参见2.2功能位。 |
| W |
| DriverFeatureSel | **激活（访客）功能字选择** |
| 0x024 | 写入该寄存器会选择一组32个激活的功能位，可通过写入DriverFeatures访问。 |
| W |
| QueueSel | **虚拟队列索引** |
| 0x030 | 写入此寄存器将选择以下操作在QueueNumMax，QueueNum，QueueReady，QueueDescLow，QueueDescHigh，QueueAvailLow，QueueAvailHigh，QueueUsedLow和QueueUsedHigh上应用的虚拟队列。第一个队列的索引号为零（0x0）。 |
| W |

|  |  |
| --- | --- |
| 名字 | **功能** |
| 相对于基的偏置 | 描述 |
| 方向 |  |

|  |  |
| --- | --- |
| QueueNumMax | **最大虚拟队列大小** |
| 0x034 | 从寄存器读取将返回设备准备处理的队列的最大大小（元素数），如果队列不可用则返回零（0x0）。这种方法适用于通过写入QueueSel来选择队列。 |
| R |
| QueueNum | **虚拟队列大小** |
| 0x038 | 队列大小是队列中元素的数量，对每个描述符表来说，即为可用环和已用环中的元素数量。写入此寄存器会通知设备驱动程序将使用的队列大小。这种方法适用于通过写入QueueSel来选择队列。 |
| W |
| QueueReady | **虚拟队列准备位** |
| 0x044 | 向该寄存器写入一个（0x1）会通知设备它可以执行来自该虚拟队列的请求。读取该寄存器将返回最后一个写入它的值。读写访问都适用于通过写入QueueSel来选择队列。 |
| RW |
|  |
| QueueNotify | **队列通知器** |
| 0x050 | 将队列索引写入此寄存器会通知设备队列中有新的缓冲区要处理。 |
| W |
| InterruptStatus | **中断状态** |
| 0x60 | 从该寄存器读取将返回导致器件中断被置位的事件的位掩码。以下事件是可能的：  **已用环更新** -位0-中断被设置，因为设备已在至少一个活动虚拟队列中更新了已使用的环。  **配置更改** -位1-中断被设置，因为设备的配置已经发生改变。 |
| R |
| InterruptACK | **中断确认** |
| 0x064 | 将具有InterruptStatus中定义的位的值写入该寄存器会通知器件已处理产生中断的事件。 |
| W |
| Status | **设备状态** |
| 0x070 | 从该寄存器读取将返回当前的设备状态标志，指示驱动程序的进度。将非零值写入该寄存器会设置状态标志，指示驱动程序进度。向该寄存器写入零（0x0）会触发器件复位。另见p.4.2.3.1设备初始化。 |
| W |
| QueueDescLow | **虚拟队列的描述附表的64位宽物理地址** |
| 0x080  QueueDescHigh  0x084 | 写入这两个寄存器（地址的低32位到QueueDescLow，高32位到QueueDescHigh）通过写入QueueSel寄存器通知设备队列选择的描述符表的地址。 |
| W |
| QueueAvailLow | **虚拟队列索引** |
| 0x090  QueueAvailHigh  0x094  W | 写入这两个寄存器（地址的低32位到QueueAvailLow，高32位到QueueAvailHigh）通过向QueueSel写入通知设备选择的队列的可用环的地址。 |

|  |  |
| --- | --- |
| 名字 | **功能** |
| 相对于基的偏置方向 | 描述 |
|  |  |

|  |  |
| --- | --- |
| QueueUsedLow | **虚拟队列的已用环64位宽物理地址** |
| 0x0a0  QueueUsedHigh  0x0a4 | 写入这两个寄存器（地址的低32位到QueueUsed-低，高32位到QueueUsedHigh）通过向QueueSel写入通知设备选择的队列的已用环的地址。 |
| W |
| ConfigGeneration | **配置原子性值** |
| 0x0fc | 从该寄存器读取将返回描述特定于设备的配置空间版本的值（请参阅Config）。然后，驱动程序可以访问配置空间，完成后再次读取ConfigGeneration。如果在这两个ConfigGeneration读取之间配置空间的任何部分都没有发生更改，则返回的值是相同的。如果值不同，则配置空间访问不是原子的，驱动程序必须再次执行操作。另见2.3。 |
| R |
| Config | **配置空间** |
| 0x100+ | 特定于器件的配置空间从偏移量0x100开始，并通过字节对齐访问。其含义和大小取决于设备和驱动程序。 |
| RW |

##### 4.2.2.1 设备要求：MMIO设备寄存器布局

设备**必须**向MagicValue返回0x74726976。

设备**必须**向Version中返回值0x2。

从每个事件开始，设备**必须**通过设置InterruptStatus中相应的位来显示每个事件，直到驱动器通过将相应的位掩码写入中断ACK寄存器来确认中断。不代表发生事件的位必须为零。

复位时，为了清除设备中的所有队列，设备**必须**清除InterruptStatus中的所有位和QueueReady寄存器中的就绪位。

如果存在驱动程序看到配置状态不一致的风险，则设备**必须**更改ConfigGeneration中返回的值。

当QueueReady为零（0x0）时，设备**不得**访问虚拟队列内容。

##### 4.2.2.2 驱动要求：MMIO设备寄存器布局

驱动程序**不得**访问表4.1中未描述的存储器位置（或者，如果是配置空间，请参见设备规范），**不得**写入只读寄存器（方向为R）且**不得**从只写寄存器中读取（方向为W）。

驱动程序**必须**仅使用32位宽且对齐的读写来访问表4.1中所述的控制寄存器。对于特定于器件的配置空间，驱动器**必须**对8位宽的字段使用8位宽访问，对16位宽的字段进行16位宽和对齐访问，对32和64位宽字段使用32位宽和对齐访问。

驱动程序**必须**忽略不是0x74726976的MagicValue的设备，尽管这**可能**会报告错误。

驱动程序**必须**忽略版本不是0x2的设备，尽管它**可能**会报告错误。

驱动程序**必须**忽略具有DeviceID 0x0的设备，但**不得**报告任何错误。

在从DeviceFeatures读取之前，驱动程序**必须**向DeviceFeaturesSel写入值。

在写入DriverFeatures寄存器之前，驱动程序必**须**向DriverFeaturesSel寄存器写入值。

驱动程序**必须**向QueueNum写入值，该值小于或等于QueueNumMax中设备提供的值。

当QueueReady不为零时，驱动程序**不得**访问QueueNum，QueueDescLow，QueueDescHigh，QueueAvailLow，QueueAvailHigh，QueueUsedLow，QueueUsedHigh。

要停止使用队列，驱动程序**必须**向QueueReady写入零（0x0）并且必须重新读取该值以确保同步。

驱动程序**必须**忽略InterruptStatus中未定义的位。

驱动程序**必须**写入一个带有位掩码的值，该位掩码描述在完成处理中断时处理到InterruptACK的事件，并且**不得**设置值中的任何未定义位。

#### 4.2.3 特定于MMIO的初始化与设备操作

##### 4.2.3.1 设备初始化

###### 4.2.3.1.1驱动要求：设备初始化

驱动程序**必须**通过读取和检查MagicValue和Version中的值来启动设备初始化。如果两个值都有效，它**必须**读取DeviceID，如果它的值为零（0x0），**必须**中止初始化并且**不得**访问任何其他寄存器。

进一步初始化**必须**遵循3.1设备初始化中描述的过程。

##### 4.2.3.2 虚拟队列配置

驱动程序通常会以下列方式初始化虚拟队列：

1. 选择将其索引（第一个队列为0）写入QueueSel的队列。

2. 检查队列是否尚未使用：读取QueueReady，并期望返回值为零（0x0）。

3. 从QueueNumMax读取最大队列大小（元素数）。 如果返回值为零（0x0），则队列不可用。

4. 分配队列页并将其归零，确保内存在物理上是连续的。建议将Used Ring对齐到最佳边界（通常是页面大小）。

5. 通过将大小写入QueueNum来通知设备有关队列大小的信息。

6. 将队列的Descriptor Table，可用环和已用环的物理地址写入（分别）QueueDescLow / QueueDescHigh，QueueAvailLow / QueueAvailHigh和QueueUsedLow / QueueUsedHigh寄存器对。

7. 向0x1写入QueueReady。

##### 4.2.3.3 通知设备

驱动程序通过将更新的队列的索引写入QueueNotify来通知设备有关队列中可用的新缓冲区。

##### 4.2.3.4 来自设备的通知

内存映射的virtio设备使用单个专用中断信号，当设置了InterruptStatus描述中描述的至少一个位时，该信号被置位。这是设备通知驱动程序有关队列中可用的新使用的缓冲区或设备配置更改的信息的方式。

###### 4.2.3.4.1 驱动要求：来自设备的通知

收到中断后，驱动程序**必须**读取InterruptStatus以检查产生中断的原因（参见寄存器说明）。处理完中断后，驱动程序必须通过将与处理事件对应的位掩码写入InterruptACK寄存器来确认。

#### 4.2.4 旧版接口

旧版的MMIO传输使用基于页的寻址，导致控制寄存器布局，设备初始化和虚拟队列配置过程略有不同。

表4.2给出了控制寄存器布局，省略了不改变其功能和行为的寄存器的描述：

**表4.2：MMIO设备旧版寄存器布局**

|  |  |
| --- | --- |
| 名字 | **功能** |
| 相对于基的偏置方向 | 描述 |
|  |  |

|  |  |
| --- | --- |
| MagicValue | **Magic value** |
| 0x000 |  |
| R |  |

|  |  |
| --- | --- |
| 名字 | **功能** |
| 相对于基的偏置方向 | 描述 |
|  |  |

|  |  |
| --- | --- |
| Version | **设备版本号** |
| 0x004 | 旧版设备返回值0x1 |
| R |
| DeviceID | **Virtio子系统设备ID** |
| 0x008 |  |
| R |
| VendorID | **Virtio子系统供应商ID** |
| 0x00c |  |
| R |
| HostFeatures | **代表设备支持的功能的标志** |
| 0x010 |  |
| R |
| HostFeaturesSel | **设备（主机）功能字选择** |
| 0x014 |  |
| W |
| GuestFeatures | **代表驱动程序理解并激活的设备功能的标志** |
| 0x020 |  |
| W |
| GuestFeaturesSel | **激活（访客）功能字选择** |
| 0x024 |  |
| W |
| GuestPageSize | **访客页大小** |
| 0x028 | 在使用任何队列之前，驱动程序在初始化期间将客户页面大小以字节为单位写入寄存器。此值应为2的幂，并由设备用于计算第一个队列页面的访客地址（请参阅QueuePFN）。 |
| W |
| QueueSel | **虚拟队列索引** |
| 0x030  W | 写入此寄存器选择以下操作QueueNumMax，QueueNum，QueueAlign和QueuePFN寄存器应用的虚拟队列值。第一个队列的索引号为零（0x0）。 |
| QueueNumMax | **最大虚拟队列大小** |
| 0x034 | 从寄存器读取将返回设备准备处理的队列的最大大小，如果队列不可用，则返回零（0x0）。这种方法适用于通过写入QueueSel来选择队列，且只在QueuePFN置为0（0x0）时，才允许使用，因此当队列没有活动时，才能使用。 |
| R |
| QueueNum | **虚拟队列大小** |
| 0x038 | 队列大小是队列中元素的数量，因此描述符表的大小以及可用和已用环。写入此寄存器会通知设备驱动程序将使用的队列大小。这种方法适用于通过写入QueueSel来选择队列 |
| W |

|  |  |
| --- | --- |
| 名字 | **功能** |
| 相对于基的偏置方向 | 描述 |
|  |  |

|  |  |
| --- | --- |
| QueueAlign | **虚拟队列中的已用环对齐** |
| 0x03c | 写入该寄存器会通知设备有关已使用环的以字节为单位的对齐边界。此值应为2的幂，并应用于通过写入QueueSel进行选择的队列。 |
| W |
| QueuePFN | **Virtio子系统设备ID** |
| 0x040 | 写入此寄存器会通知设备虚拟队列在访客的物理地址空间中的位置。此值是以队列描述符表开头的页的索引号。值零（0x0）表示物理地址零（0x00000000）并且是非法的。当驱动程序停止使用队列时，它会将零（0x0）写入该寄存器。从该寄存器读取将返回当前使用的队列页码，因此除零（0x0）以外的值意味着该队列正在使用中。读写访问都适用于通过写入QueueSel进行选择的队列。 |
| RW |
| QueueNotify | **队列通知器** |
| 0x050 |  |
| W |
| InterruptStatus | **中断状态** |
| 0x60 |  |
| R |
| InterruptACK | **中断确认** |
| 0x064 |  |
| W |
| Status | **设备状态** |
| 0x070 | 从该寄存器读取将返回当前的设备状态标志。将非零值写入该寄存器会设置状态标志，指示操作系统/驱动程序进度。向该寄存器写入零（0x0）会触发设备复位。设备将QueuePFN设置为零（0x0），用于设备中的所有队列初始化。另请参见3.1设备初始化。 |
| RW |
| Config | **配置空间** |
| 0x100+ |  |
| RW |

虚拟队列页面大小是通过写入GuestPageSize来定义，由访客写入。驱动程序在配置虚拟队列之前执行此操作。

虚拟队列布局遵循p.2.4.2旧版接口：关于虚拟队列布局的注释，其中对齐在QueueAlign中定义。

虚拟队列配置如下：

1. 选择将其索引（第一个队列为0）写入QueueSel的队列。

2. 检查队列是否尚未使用：读取QueuePFN，期望返回值为零（0x0）。

3. 从QueueNumMax读取最大队列大小（元素数）。如果返回值为零（0x0），则队列不可用。

4. 在连续虚拟内存中分配和归零队列页，将Used Ring对齐到最佳边界（通常是页面大小）。驱动程序应选择小于或等于QueueNumMax的队列大小。

5. 通过将大小写入QueueNum来通知设备有关队列大小的信息。

6. 通过将其值以字节为单位写入QueueAlign，通知设备使用的对齐方式。

7. 将队列的第一页的物理编号写入QueuePFN寄存器。通知机制没有发生改变。

### 4.3 Channel I/O下的Virtio

基于S/390的虚拟机既不支持PCI也不支持MMIO，因此需要不同的传输方式。

virtio-ccw使用基于标准Channel I / O的机制用于S / 390上的大多数设备。具有特殊控制单元类型的虚拟通道设备作为virtio设备的代理（类似于virtio-pci使用PCI设备的方式），且virtio设备的配置和操作（通常）通过通道命令完成。这意味着通过标准操作系统算法可以发现virtio设备，并且添加virtio支持主要是支持新控制单元类型的问题。由于S/390是一个大端机器，通过通道命令传输的数据结构是大端的：通过使用be16，be32和be64类型可以明显地看出这一点。

#### 4.3.1 基本概念

作为代理设备，virtio-ccw使用带有特殊控制单元类型（0x3832）的通道连接I / O控制单元，并通过虚拟I / O子通道以及0x32类型的虚拟通道路径访问所连接的virtio设备的子系统设备ID相对应的控制单元模型。这个代理设备通过普通通道子系统发现（通常为存储子通道循环），并对基本通道命令进行回答。

·NO-OPERATION (0x03)

·BASIC SENSE (0x04)

·TRANSFER IN CHANNEL (0x08)

·SENSE ID (0xe4)

对于virtio-ccw代理设备，SENSE ID将返回以下信息

|  |  |  |
| --- | --- | --- |
| 字节数 | 描述 | 内容 |
| 0 | 保留 | 0xff |
| 1-2 | 控制单元类型 | 0x3832 |
| 3 | 控制单元模型 | <virtio设备id> |
| 4-5 | 设备类型 | 零（未设置） |
| 6 | 设备模型 | 零（未设置） |
| 7-255 | 扩展的senseld数据 | 零（未设置） |

除了基本的通道命令外，virtio-ccw还定义了一组与virtio的配置和操作相关的通道命令：

#define CCW\_CMD\_SET\_VQ 0x13

#define CCW\_CMD\_VDEV\_RESET 0x33

#define CCW\_CMD\_SET\_IND 0x43

#define CCW\_CMD\_SET\_CONF\_IND 0x53

#define CCW\_CMD\_SET\_IND\_ADAPTER 0x73

#define CCW\_CMD\_READ\_FEAT 0x12

#define CCW\_CMD\_WRITE\_FEAT 0x11

#define CCW\_CMD\_READ\_CONF 0x22

#define CCW\_CMD\_WRITE\_CONF 0x21

#define CCW\_CMD\_WRITE\_STATUS 0x31

#define CCW\_CMD\_READ\_VQ\_CONF 0x32

#define CCW\_CMD\_SET\_VIRTIO\_REV 0x83

##### 4.3.1.1 设备要求：基本概念

virtio-ccw设备的作用类似于普通通道设备，如[S390 PoP]和[S390通用I / O]中所述。特别是：

·对任何不支持的命令，设备**必须**发布一个带有拒绝指令的单元检查。

·如果驱动程序没有抑制通道命令的长度检查，则当实际长度与预期长度不匹配时，设备**必须**呈现结构中详述的子通道状态。

·如果驱动程序确实抑制了通道命令的长度检查，则如果传输的数据不包含足够的数据来处理命令，则设备**必须**提供检查条件。 如果驱动程序提交的缓冲区太长，设备**应该**接受该命令。

##### 4.3.1.2 设备要求：基本概念

驱动程序应该尝试在通道命令中提供正确的长度，即使它抑制了该命令的长度检查。

#### 4.3.2 设备初始化

virtio-ccw使用多个通道命令来对设备进行设置。

##### 4.3.2.1 设置Virtio修订版

CCW\_CMD\_SET\_VIRTIO\_REV由驱动程序发出，用于设置virtio-ccw的修正版并将其内容进行传输，用于驱动设备。它使用以下通信结构：

struct virtio\_rev\_info {

be16 revision;

be16 length;

u8 data[];

};

revision包含所需的修订版ID，长度为数据部分的长度以及依赖于数据修订版的其他所需选项。

以下为支持值：

|  |  |  |  |
| --- | --- | --- | --- |
| 修正版 | 长度 | 数据 | 备注 |
| 0 | 0 | <空> | 旧版接口；只支持发送设备 |
| 1 | 0 | <空> | Virtio 1.0 |
| 2-n |  |  | 为后续版本保留 |

请注意，virtio标准的更改不一定与virtio-ccw修订版中的更改相对应。

###### 4.3.2.1.1 设备要求：设置Virtio修订版

对于任何不支持的修订版本，设备**必须**发布带有拒绝命令的单元检查。对于版本，长度和数据的任何无效组合，它也都**必须**使用拒绝命令发布单元检查。非过渡设备**必须**拒绝修订版ID 0。

设备**必须**用带有拒绝命令的回答来回答任何virtio-ccw特定的通道命令，该命令未包含在驱动程序选择的修订版中。

在驱动程序成功选择修订版之后，设备**必须**以拒绝命令回答任何选择不同修订版本的尝试。

设备**必须**从启用相关子通道的时候开始将修订视为未设置，直到驱动程序成功设置修订。这意味着在禁用和启用相关子通道时，修订不会持续存在。

###### 4.3.2.1.2 驱动要求：设置Virtio修订版

驱动程序**应该**首先尝试设置它支持的最高版本，如果收到拒绝命令，则继续尝试较低版本。

在设置修订版之前，驱动程序**不得**发出任何其他virtio-ccw特定通道命令。

在驱动程序成功选择修订版之后，它**不得**尝试选择其他版本。

###### 4.3.2.1.3 旧版接口：关于设置Virtio修订版的注意事项

旧设备不支持CCW\_CMD\_SET\_VIRTIO\_REV并使用拒绝命令进行应答。旧设备不支持CCW\_CMD\_SET\_VIRTIO\_REV并使用拒绝命令进行应答。在这种情况下，非过渡驱动程序**必须**停止尝试操作此设备。过渡驱动程序**必须**像操作设置版本0一样操作设备。

旧版驱动程序在发出其他virtio-ccw特定通道命令之前不会发出CCW\_CMD\_SET\_VIRTIO\_REV。因此，非过渡设备必**须**使用拒绝命令来回答任何此类尝试。在这种情况下，就像驱动程序选择了修订版0一样，过渡设备必须假设驱动程序是旧版驱动程序并继续。这意味着设备必须拒绝任何对版本0无效的命令，包括后续的CCW\_CMD\_SET\_VIRTIO\_REV。

##### 4.3.2.2 配置虚拟队列

CCW\_CMD\_READ\_VQ\_CONF由驱动程序发出以获取有关队列的信息。 它使用以下结构进行通信：

struct vq\_config\_block {

be16 index;

be16 max\_num;

};

请求的队列索引缓冲区数在max\_num中返回。

然后，驱动程序发出CCW\_CMD\_SET\_VQ，用于通知设备其队列的位置。其传输的结构为：

struct vq\_info\_block {

be64 desc;

be32 res0;

be16 index;

be16 num;

be64 avail;

be64 used;

};

desc，avail和used分别包含描述符表的访客地址，可用环与已用环分别包含其队列索引。实际的虚拟队列大小（分配的缓冲区数）以num表示。

###### 4.3.2.2.1 设备要求：配置虚拟队列

res0被保留，设备**必须**忽略它。

###### 4.3.2.2.2 旧版接口：关于配置虚拟队列的注意事项

对于旧版驱动程序或选择版本0的驱动程序，CCW\_CMD\_SET\_VQ使用以下通信块：

struct vq\_info\_block\_legacy {

be64 queue;

be32 align;

be16 index;

be16 num;

};

队列包含队列索引的访客地址，num包含缓冲区的数量并对齐。队列布局遵循2.4.2旧版接口：关于虚拟队列布局的注意事项。

##### 4.3.2.3 传输状态信息

驱动程序通过CCW\_CMD\_WRITE\_STATUS命令发送8位状态值更改设备的状态。

如2.2.2中所述，设备有时无法设置状态字段：例如，在设备初始化期间，它可能无法接受FEATURES\_OK状态位。

###### 4.3.2.3.1 驱动要求：传输状态信息

如果设备使用拒绝命令发布单元检查，用以响应CCW\_CMD\_WRITE\_STATUS命令，则驱动程序**必须**假定设备未能设置状态并且状态字段保留其先前值。

###### 4.3.2.3.2 设备要求：传输状态信息

如果设备未能将状态字段设置为驱动程序写入的值，则设备**必须**确保状态字段保持不变，并且**必须**使用命令拒绝发布单元检查。

##### 4.3.2.4 处理设备功能

特征位以32位值的数组排列，总共有8192个特征位。 Feature位以小端字节顺序排列。

处理功能的CCW命令使用以下通信块：

struct virtio\_feature\_desc {

le32 features;

u8 index;

};

功能是当前访问的32位功能，而索引描述了要访问哪些功能位值。在结构的末尾没有添加填充，它的长度恰好是5个字节。

访客通过CCW\_CMD\_READ\_FEAT命令获取设备的设备功能集。设备将功能存储为功能索引处。

为了将其支持的功能传递给设备，驱动程序使用CCW\_CMD\_WRITE\_FEAT命令，表示功能/索引组合。

##### 4.3.2.5 设备配置

设备的配置空间位于主机内存中。

要从配置空间获取信息，驱动程序使用CCW\_CMD\_READ\_CONF，指定设备要写入的访客内存。

为了更改配置信息，驱动程序使用CCW\_CMD\_WRITE\_CONF，指定设备要读取的访客内存。

在这两种情况下，完整的配置空间都会被传输。这允许驱动程序将新配置空间与旧版本进行比较，并在内部更改时保持世代数。

##### 4.3.2.6 设置指示符

为了设置主机->访客通知的指示位，驱动程序使用不同的通道命令，具体取决于它是否希望使用与子通道相关的传统I/O中断或适配器I/O中断来进行虚拟队列通知。对于任何给定的设备，这两种机制是互斥的。

对于配置更改指示符，无论传统I/O中断是否用于虚拟队列通知，均只提供使用旧版I/O中断的机制。

###### 4.3.2.6.1 设置经典队列指示符

通过经典I/O中断进行通知的指示符包含在每个virtio-ccw代理设备的64位值中。

为了传输主机->访客通知的指示符位的位置，驱动程序使用CCW\_CMD\_SET\_IND命令，指向包含64位值中指示符的访客地址的位置。

如果驱动程序已经通过CCW\_CMD\_SET\_IND\_ADAPTER命令设置了两阶段队列指示符，则设备**必须**使用拒绝命令对任何后续CCW\_CMD\_SET\_IND命令发布单元检查。

###### 4.3.2.6.2 设置配置变化指示符

配置更改主机->访客通知的指示符包含在每个virtio-ccw代理设备的64位值中。

为了传输配置更改主机->访客通知中使用的指示符位的位置，驱动程序发出CCW\_CMD\_SET\_CONF\_IND命令，指向包含64位值中指示符的访客地址的位置。

###### 4.3.2.6.3 设置两阶段队列指示符

通过适配器I/O中断通知的指示器包括两个阶段：

·一个摘要指示符字节包含虚拟队列的一个或多个virtio-ccw代理。

·用于virtio-ccw代理设备的虚拟队列的一组连续指示符位

要传输摘要和队列指示符位的位置，驱动程序将使用带有以下有效内容的CCW\_CMD\_SET\_IND\_ADAPTER命令：

struct virtio\_thinint\_area {

be64 summary\_indicator;

be64 indicator;

be64 bit\_nr;

u8 isc;

} \_\_attribute\_\_ ((packed));

summary\_indicator包含8位摘要指示符的访客地址。indicator包含一个区域的访客地址，其中包含设备的指示符，从bit\_nr开始，每个虚拟队列设备占一位。位号从左侧开始，即第一个字节中的最高位被分配位号0. isc包含用于适配器I/O中断的I/O中断子类。它**可能**与代理virtio-ccw设备的子通道使用的isc不同。结构的末尾没有添加填充，它的长度恰好是25个字节。

4.3.2.6.3.1 设备要求：设置两阶段队列指示符

如果驱动程序已经通过CCW\_CMD\_SET\_IND命令设置了经典队列指示符，则设备必须使用拒绝命令对任何后续CCW\_CMD\_SET\_IND\_ADAPTER命令发布单元检查。

###### 4.3.2.6.4 旧版接口：关于设置两阶段指示符的注意事项

在某些情况下，旧版设备仅支持经典队列指示器；在这种情况下，由于它们不知道该命令，它们将拒绝CCW\_CMD\_SET\_IND\_ADAPTER。然而，一些旧版设备不仅支持两阶段队列指示器，并且驱动程序能够成功使用CCW\_CMD\_SET\_IND\_ADAPTER来设置它们。

#### 4.3.3 设备操作

##### 4.3.3.1 主机->访客通知

主机->访客通知存在两种操作模式，经典I/O中断和适配器I/O中断。使用模式由驱动程序分别使用CCW\_CMD\_SET\_IND，CCW\_CMD\_SET\_IND\_ADAPTER来设置队列指示符来确定。

对于配置更改，驱动程序始终使用经典I/O中断。

###### 4.3.3.1.1 来自Classic I/O中断的通知

如果驱动程序使用CCW\_CMD\_SET\_IND命令设置队列指示符，则设备将使用经典I/O中断来进行关于虚拟队列活动的主机->访客通知。

为了向驱动程序通知虚拟队列的缓冲区，设备在访客提供的指示符中设置相应的位。如果子通道尚未处于中断状态，则设备会产生未经请求的I/O中断。

如果设备想要通知驱动程序有关配置更改的信息，则会在配置指示符中设置位0，并在需要时生成未经请求的I/O中断。如果将适配器I/O中断用于队列通知，这也适用。

###### 4.3.3.1.2 通过Adapter I/O中断通知

如果驱动程序使用CCW\_CMD\_SET\_IND\_ADAPTER命令设置队列指示符，则设备将使用适配器I/O中断来进行有关虚拟队列活动的主机->访客通知。

为了向驱动程序通知虚拟队列的缓冲区，设备将访客提供的指示符区域中的位设置为相应的偏移量。访客提供的摘要指示符设置为0x01。并生成了相应中断子类的适配器I/O中断。

驱动程序处理适配器I/O中断的推荐方法如下：

·处理与摘要指示符关联的所有队列指示符位。

·清除摘要指示符，然后执行同步（内存屏障）。

·再次处理与摘要指示符关联的所有队列指示符位。

4.3.3.1.2.1 设备要求：关于通过Adapter I/O中断提示的注意事项

如果在通知之前未设置摘要指示符，则设备**应该**只生成适配器I/O中断。

4.3.3.1.2.2 驱动要求：关于通过Adapter I/O中断提示的注意事项

在处理队列指示符之前，驱动程序**必须**在收到适配器I/O中断后清除摘要指示符。

###### 4.3.3.1.3 旧版接口：关于主机->访客通知的注意事项

由于旧版设备和驱动程序仅支持经典队列指示器，因此主机->访客通知将始终通过经典I/O中断完成。

##### 4.3.3.2 访客->主机通知

不幸的是，为了向设备通知虚拟队列缓冲区，驱动程序不能使用通道命令（通道I/O的异步特性与主机块I/O后端交互不良）。取而代之的，它使用诊断0x500调用，子代码3指定队列，如下所示：

|  |  |  |
| --- | --- | --- |
| GPR | 输入值 | 输出值 |
| 1 | 0x3 |  |
| 2 | 子通道ID | 主机Cookie |
| 3 | 虚拟队列号 |  |
| 4 | 主机Cookie |  |

###### 4.3.3.2.1 设备要求：访客->主机通知

设**备必须**忽略GPR2的0-31位（从左边开始计数）。这使得子通道ID与传递给现有I/O指令的方式一致。

设备**可以**在GPR2中返回64位主机cookie，以加快通知执行速度。

###### 4.3.3.2.2 驱动要求：访客->主机通知

对于每个通知，驱动程序应该使用GPR4从前一个通知中传递GPR2中收到的主机cookie。

**注：**例如

info->cookie = do\_notify(schid,

virtqueue\_get\_queue\_index(vq),

info->cookie);

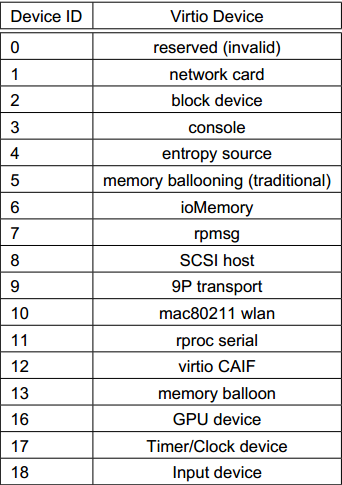
##### 4.3.3.3 复位设备

为了重置设备，驱动程序发送CCW\_CMD\_VDEV\_RESET命令。

## 5 设备类型

在队列、配置空间和virtio中内置的功能协议工具之上，定义了几个设备。

以下设备ID用于标识不同类型的virtio设备。某些设备ID是为当前未在此标准中定义的设备保留的。

发现可用的设备及其取决于总线的类型。

本文件未指明上述部分设备，因为它们被视为不成熟或是没有合适的位置。请注意，有些仅由唯一的现有实现指定；它们可能成为未来规范的一部分，或完全废弃，或存在本标准之外。我们不再谈论它们了。

### 5.1 网络设备

virtio网络设备是虚拟以太网卡，是virtio目前支持的最复杂的设备。它快速地增强并清楚地展示了如何支持将新功能的添加到现有设备中。空缓冲区被放置在一个virtqueue中以接收数据包，而传出的数据包则按该顺序排队进入另一个virtqueue进行传输。第三个命令队列用于控制高级筛选功能。

#### 5.1.1设备ID

1

#### 5.1.2 Virtqueues(虚拟队列)

**0** 接收 q1

**1** 传输 q1

**…**

**2N** 接收 qN

**2N+1** 传输 qN

**2N+2** 控制q

N = 1 如果VIRTIO\_NET\_MQ未被议定，否则 N被设为max\_virtqueue\_pairs.

#### 5.1.3 特征位

**VIRTIO\_NET\_F\_CSUM (0)** 设备用“部分校验和”处理数据包。这种“校验和卸载”是现代网卡上的一个常见功能。

**VIRTIO\_NET\_F\_GUEST\_CSUM (1)** 驱动程序用部分校验和处理数据包。

**VIRTIO\_NET\_F\_CTRL\_GUEST\_OFFLOADS(2)** 控制通道卸载重新配置支持。

**VIRTIO\_NET\_F\_MAC (5)** 设备已给出MAC地址。

**VIRTIO\_NET\_F\_GUEST\_TSO4 (7)** 驱动程序可以接收TSOV4。

**VIRTIO\_NET\_F\_GUEST\_TSO6 (8)** 驱动程序可以接收 TSOv6。

**VIRTIO\_NET\_F\_GUEST\_ECN (9)** 驱动程序可以接收TSO 和 ECN。

**VIRTIO\_NET\_F\_GUEST\_UFO (10)** 驱动程序可以接收 UFO。

**VIRTIO\_NET\_F\_HOST\_TSO4 (11)** 设备可以接收TSOV4。

**VIRTIO\_NET\_F\_HOST\_TSO6 (12)** 设备可以接收TSOv6。

**VIRTIO\_NET\_F\_HOST\_ECN (13)** 设备可以接收 TSO 和ECN。

**VIRTIO\_NET\_F\_HOST\_UFO (14)** 设备可以接收UFO。

**VIRTIO\_NET\_F\_MRG\_RXBUF (15)** 驱动程序可以合并接收缓冲区。

**VIRTIO\_NET\_F\_STATUS (16)** 配置状态字段可用。

**VIRTIO\_NET\_F\_CTRL\_VQ (17)** 控制通道可用。

**VIRTIO\_NET\_F\_CTRL\_RX (18)** 控制通道RX模式支持**。**

**VIRTIO\_NET\_F\_CTRL\_VLAN (19)** 控制通道VLAN过滤。

**VIRTIO\_NET\_F\_GUEST\_ANNOUNCE(21)** 驱动程序可以无偿发送数据包。

**VIRTIO\_NET\_F\_MQ(22)** 设备支持自动接收转向的多队列。

**VIRTIO\_NET\_F\_CTRL\_MAC\_ADDR(23)** 通过控制通道设置MAC地址。

##### 5.1.3.1 特征位要求

某些网络特征位需要其他网络特征位（见2.2.1）：

**VIRTIO\_NET\_F\_GUEST\_TSO4** 需要VIRTIO\_NET\_F\_GUEST\_CSUM.

**VIRTIO\_NET\_F\_GUEST\_TSO6** 需要 VIRTIO\_NET\_F\_GUEST\_CSUM.

**VIRTIO\_NET\_F\_GUEST\_ECN** 需要 VIRTIO\_NET\_F\_GUEST\_TSO4 or VIRTIO\_NET\_F\_GUEST\_TSO6.

**VIRTIO\_NET\_F\_GUEST\_UFO** 需要 VIRTIO\_NET\_F\_GUEST\_CSUM.

**VIRTIO\_NET\_F\_HOST\_TSO4** 需要 VIRTIO\_NET\_F\_CSUM.

**VIRTIO\_NET\_F\_HOST\_TSO6** 需要VIRTIO\_NET\_F\_CSUM.

**VIRTIO\_NET\_F\_HOST\_ECN** 需要 VIRTIO\_NET\_F\_HOST\_TSO4 or VIRTIO\_NET\_F\_HOST\_TSO6.

**VIRTIO\_NET\_F\_HOST\_UFO** 需要 VIRTIO\_NET\_F\_CSUM.

**VIRTIO\_NET\_F\_CTRL\_RX** 需要VIRTIO\_NET\_F\_CTRL\_VQ.

**VIRTIO\_NET\_F\_CTRL\_VLAN** 需要 VIRTIO\_NET\_F\_CTRL\_VQ.

**VIRTIO\_NET\_F\_GUEST\_ANNOUNCE** 需要VIRTIO\_NET\_F\_CTRL\_VQ.

**VIRTIO\_NET\_F\_MQ** 需要 VIRTIO\_NET\_F\_CTRL\_VQ.

**VIRTIO\_NET\_F\_CTRL\_MAC\_ADDR** 需要 VIRTIO\_NET\_F\_CTRL\_VQ

##### 5.1.3.2 传统接口：特征位

**VIRTIO\_NET\_F\_GSO (6)** 设备处理任何GSO类型的数据包。

这本来是为了表示分段卸载支持，但经过进一步调查，很明显需要多个位

#### 5.1.4 设备配置布局

 当前定义了三个驱动程序只读配置字段。MAC地址字段始终存在（仅在设置VIRTIO\_NET\_F\_MAC时有效），并且*status*仅在设置VIRTIO\_NET\_F\_STATUS时存在。当前为状态字段定义了两个只读位（用于驱动程序）：VIRTIO\_NET\_S\_LINK\_UP 和 VIRTIO\_NET\_S\_ANNOUNCE。

 以下驱动程序只读字段max\_virtqueue\_pairs仅在设置VIRTIO\_NET\_F\_MQ时存在。在设置VIRTIO\_NET\_F\_MQ之后，此字段指定可以配置的每个发送和接收virtqueue（分别为receiveq1…receiveqn和transmitq1…transmitqn）的最大数量。

##### 5.1.4.1设备要求：设备配置布局

如果提供VIRTIO\_NET\_F\_MQ，设备必须（MUST）将max-virtqueue-pairs设置为1到0x8000之间（含1和0x8000）。

##### 5.1.4.2驱动程序要求：设备配置布局

如果设备提供VIRTIO\_NET\_F\_MAC，驱动程序应该(SHOULD)与它协商。如果驱动程序协商VIRTIO\_NET\_F\_MAC功能位，则驱动程序必须（MUST）将NIC的物理地址设置为mac。否则，它应该（SHOULD）使用本地管理的MAC地址（参见IEEE 802, “9.2 48-bit universal LAN MAC addresses”）。如果驱动程序不协商VIRTIO\_NET\_F\_STATUS功能位，它应该假设链接是激活的，否则它应该（SHOULD）从***status***的底部读取链接状态。

##### 5.1.4.3传统接口：设备配置布局

当使用传统接口时，过渡设备和驱动程序必须根据客户机的本地字节存储次序而不是（不使用传统接口时是必须的）小字节来格式化在结构virtio\_net\_config中的***status***和***max virtqueue\_pairs***。当使用传统接口时，Mac是驱动程序可写的，这为驱动程序提供了一种在不协商VIRTIO\_NET\_F\_CTRL\_MAC\_ADDR位的情况下更新mac。

#### 5.1.5设备初始化

驱动程序将执行典型的初始化例程，如：

1、识别并初始化接收和传输虚拟队列，每种虚拟队列最多N个。如果 VIRTIO\_NET\_F\_MQ特征位已协商，N=***max\_virtqueue\_pairs***，否则标识N=1。

2、如果协商VIRTIO\_NET\_F\_MQ特征位，则标识控制virtqueue。

3、用缓冲区填充接收队列：见5.1.6.3。

4、即使使用VIRTIO\_NET\_F\_MQ，默认情况下也只使用receiveq1、transmitq1 和 controlq。这个驱动程序将发送 VIRTIO\_NET\_CTRL\_MQ\_VQ\_PAIRS\_SET命 令，指定要使用的传输和接 收队列的数量。

5、如果设置VIRTIO\_NET\_F\_MAC功能位，则配置空间***mac***入口地址指示 的是 网卡的“物理”地址，否则驱动程序通常会生成随机本地MAC地址。

6、如果协商VIRTIO\_NET\_F\_STATUS特征位，则链接状态来自***status***的底 部。 否则，驱动程序假定它处于活动状态。

7、性能驱动程序指示通过协商VIRTIO\_NET\_F\_CSUM特征位来生成“无校 验 和”的数据包。

8、如果协商了该特征位（features），则驱动程序可以通过协商 VIRTIO\_NET\_F\_HOST\_TSO4（IPv4 TCP），VIRTIO\_NET\_F\_HOST\_TSO6 （IPv6 TCP）和VIRTIO\_NET\_F\_HOST\_UFO（UDP碎片）特征位，使用TCP 或UDP分段卸载。

9、相反的特征位也是可用的：驱动程序可以通过协商这些特征位来保存虚 拟设 备的一些工作。

**注意：**例如，如果两个guest虚拟机都是易受控制的（amenable），则在同一系统上的两个guest虚拟机之间传输的网络数据包可能根本不需要校验和，也不需要分段。 VIRTIO\_NET\_F\_GUEST\_CSUM特征位表明可以接收到部分校验和的数据包，如果可以接收，那么VIRTIO\_NET\_F\_GUEST\_TSO4，VIRTIO\_NET\_F\_GUEST\_TSO6，VIRTIO\_NET\_F\_GUEST\_UFO和VIRTIO\_NET\_F\_GUEST\_ECN与上述特征位的有相同的作用。 请参见下面的5.1.6.3设置接收缓冲区和5.1.6.4传入数据包的处理。

一个真正最小的驱动程序只接受VIRTIO\_NET\_F\_MAC并忽略其他所有内容。

#### 5.1.6 设备操作

通过将数据包放在transmitq1…transmitqn中来传输数据包，传入数据包的缓冲区放在receiveq1…receiveqn中。在每种情况下，包本身前面都有一个头：

controlq用于控制过滤等设备features。

##### 5.1.6.1传统接口：设备操作

使用传统接口时，过渡设备和驱动程序必须（MUST）根据guest的本地（native）endian来格式化struct virtio\_net\_hdr中的字段，而不是（不使用传统接口时）小endian。当VIRTIO\_NET\_F\_MRG\_RXBUF被协商时，传统驱动程序只在struct virtio-net-hdr中提供num-buffers；如果没有这个特性，结构将缩短2个字节。

使用传统接口时，驱动程序应（SHOULD）忽略传输队列和controlq队列的已用环条目中的len值。

注意：在之前，有些设备即使没有写入任何数据都将描述符的总长度放在那里。

##### 5.1.6.2数据包传输

传输单个数据包很简单，但根据驱动程序协商的不同功能（features）而有所不同。

1、驱动程序可以发送一个完整的校验和数据包。在这种情况下，***flags***为0， 并且gso\_type为VIRTIO\_NET\_HDR\_GSO\_NONE。

2、如果驱动程序协商VIRTIO\_NET\_F\_CSUM位，它可以跳过对数据包的校 验和阶段：

1）flags设置了VIRTIO\_NET\_F\_NEEDS\_CSUM；

2）***csum\_start***设置为数据包内到校验和的偏移量，以及；

3）***csum\_offset***表示***csum\_start*** 之后由设备放置的校验和新的字节数（16 位补码）。

4）数据包中的TCP校验和字段设置为TCP伪头的总和，以便通过替换 TCP头（header）和主体（body）的某一个补码校验和得到正确的结果。

**注意**：例如，如果有一个部分校验和的TCP（IPv4）包。它将有一个14字节的 以太网头和20字节的IP头，紧跟着的是TCP头（在该头中有16个字 节的TCP校验和字段）。***csum\_start***将为14+20=34（TCP校验和包括报 头），***csum\_offset***将为16。

3、如果驱动程序协商VIRTIO\_NET\_F\_HOST\_TSO4,TS06或者UFO， 并 且数据包需要TCP分段或UDP分段，然后将***gso\_type***设置为 VIRTIO\_NET\_HDR\_GSO\_TCPV4，TCPV6或UDP。（否则，设置为 VIRTIO\_NET\_GSO\_NONE）。在这种情况下，数据包大于1514个字节 时可以传输：元数据（metadata）指示如何复制数据包头以便将其剪切 为较小的数据包。其他GSO字段设置为：

1）***hdr-len***是一个提示，提示设备需要保留多少头才能复制到每个数据 包中，通常设置为头的长度，包括传输头[[3]](#footnote-2)。

2）***gso\_size***是超出该头段的每个数据包大小的最大值（即MSS）。

3）如果驱动程序协商了VIRTIO\_NET\_F\_HOST\_ECN位，那么***gso-type*** 中的VIRTIO\_NET\_GSO\_ECN位表示TCP数据包包设置了ECN位[[4]](#footnote-3)。

4、***num\_buffers***设置为***0***，此字段在传输的数据包上未使用。

5、头和数据包作为一个输出描述符添加到transmitq中，设备将收到新条目 的通知。（见[5.1.5设备初始化](#_5.1.5设备初始化)）。

###### 5.1.6.2.1驱动程序要求：数据包传输

驱动程序必须（MUST）将***num\_buffers***设置为***0***。

**如果**未协商VIRTIO\_NET\_F\_CSUM，则驱动程序必须将***flags***设置为***0***，并且应该(SHOULD)向设备提供完全校验和的数据包。

**如果**协商了VIRTIO\_NET\_F\_HOST\_TSO4，则驱动程序可以(MAY)将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_TCPV4以请求TCPv4分段，否则驱动程序不得(MUST NOT)将***gso\_type***设置为 VIRTIO\_NET\_HDR\_GSO\_TCPV4。

**如果**协商了VIRTIO\_NET\_F\_HOST\_TSO6，则驱动程序可以将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_TCPV6以请求TCPv6分段，否则驱动程序不得将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_TCPV6。

**如果**协商了VIRTIO\_NET\_F\_HOST\_UFO，则驱动程序可以将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_UDP以请求UDP分段，否则驱动程序不得 (MUST NOT)将 ***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_UDP。

驱动程序不应该(SHOULD NOT)向设备发送需要分段卸载的TCP数据包，其中设置了显式拥塞通知位，除非协商了VIRTIO\_NET\_F\_HOST\_ECN 功 能，否则驱动程序必须在gso\_type中设置VIRTIO\_NET\_HDR\_GSO\_ECN 位。

如果已协商VIRTIO\_NET\_F\_CSUM功能，则驱动程序可以在flags中设置VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位，如果是这样：

1. 驱动程序必须（MUST）验证来自***csum\_start***的偏移量***csum\_offset***的数据包校验和以及所有先前的偏移量;
2. 驱动程序必须（MUST）将存储在缓冲区中的数据包校验和设置为TCP / UDP伪标头;
3. 驱动程序必须（MUST）设置***csum\_start***和***csum\_offset***，以便从***csum\_start***开始到数据包结束计算一个补码校验和并将结果存储在***csum\_start***的偏移量***csum\_offset***中，将得到完全校验和的数据包;

**如果**没有协商VIRTIO\_NET\_F\_HOST\_TSO4，TSO6或UFO选项，则驱动程序必须（MUST）将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_NONE。

**如果*gso\_type***与VIRTIO\_NET\_HDR\_GSO\_NONE不同，那么驱动程序还必须在***flags***中设置VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位并且必须设置***gso\_size***以指示所需的**MSS**。

**如果**已协商其中一个VIRTIO\_NET\_F\_HOST\_TSO4，TSO6或UFO选项，则驱动程序应该（SHOULD）将***hdr\_len***设置为一个不小于标头长度的值，包括传输标头在内。

驱动程序不得（MUST NOT）在***flags***中设置VIRTIO\_NET\_HDR\_F\_DATA\_VALID位。

###### 5.1.6.2.2设备要求：数据包传输

设备必须（MUST）忽略它无法识别的标志位。

如果未设置***flags***中的VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位，则设备不得（MUST NOT）使用***csum\_start***和***csum\_offset***。

如果已经协商了VIRTIO\_NET\_F\_HOST\_TSO4，TSO6或UFO选项之一，则设备可以（MAY）使用***hdr\_len***仅作为有关传输头大小的提示。设备绝不能依赖***hdr\_len***来更正。

**注意：**这是由于实现中的各种**bugs**。

如果未设置VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位，则设备不得依赖于数据包校验和来判断是否正确。

###### 5.1.6.2.3数据包传输中断

通常，驱动程序将使用VIRTQ\_AVAIL\_F\_NO\_INTERRUPT标志来抑制传输中断（请参阅[3.2.2从设备接收已用缓冲区](#_3.2.2_从设备接收已用缓冲区)）并检查后续数据包传输路径中使用过的数据包。

此中断处理程序中的正常行为是从已使用的环中检索新的描述符并释放对应的标头和数据包。

##### 5.1.6.3设置接收缓冲区

通常情况下，尽可能的保持接收virtqueue完全填充是一个好方法：如果它耗尽，网络性能将受到影响。 如果是VIRTIO\_NET\_F\_GUEST\_TSO4，VIRTIO\_NET\_F\_GUEST\_TSO6或VIRTIO\_NET\_F\_GUEST\_UFO被使用，最大传入数据包将达到65550字节长（TCP或UDP数据包的最大值，再加上14字节以太网头），否则为1514字节。这个12字节的struct virtio\_net\_hdr就是这样的，它产生65562字节或1526字节。

###### 5.1.6.3.1驱动程序要求：设置接收缓冲区

·如果未协商VIRTIO\_NET\_F\_MRG\_RXBUF位：

-如果协商了VIRTIO\_NET\_F\_GUEST\_TSO4，VIRTIO\_NET\_F\_GUEST\_TSO6或VIRTIO\_NET\_F\_GUEST\_UFO，驱动程序应该（SHOULD）使用至少65562字节的缓冲区填充接收队列。

- 否则，驱动程序应该（SHOULD）使用至少1526字节的缓冲区填充接收队列。

•如果协商了VIRTIO\_NET\_F\_MRG\_RXBUF位，则每个缓冲区必须（MUST）大于struct virtio\_net\_hdr的大小。

**注意：**很显然，每个缓冲区可以分割为多个描述符元素。

如果协商了VIRTIO\_NET\_F\_MQ，那么每个将要被使用的receiveq1 ... receiveqN填充接收的缓冲区。

###### 5.1.6.3.2设备要求：设置接收缓冲区

设备必须（MUST）将***num\_buffers***设置为用于保存传入数据包的描述符数。如果未协商VIRTIO\_NET\_F\_MRG\_RXBUF位，则设备必须（MUST）仅使用单个描述符。

**注意：**这意味着如果未协商VIRTIO\_NET\_F\_MRG\_RXBUF，***num\_buffers***将始终为1。

##### 5.1.6.4传入数据包的处理

当数据包被复制到receiveq中的缓冲区时，最佳路径是禁用receiveq的进一步中断（参见[3.2.2从设备接收使用过的缓冲区](#_3.2.2_从设备接收已用缓冲区)）并处理数据包，直到找不到更多数据包，然后重新启用它们。

处理传入的数据包涉及：

1. ***num\_buffers***表示此数据包分布的描述符数量（包括此数据包）：如果未协商VIRTIO\_NET\_F\_MRG\_RXBUF位，则该数据包始终为1。 这允许接收大量数据包而无需分配大缓冲区。 在这种情况下，所使用的环中将至少存在***num\_buffers***，并且设备将它们链接在一起以形成单个数据包。 其他缓冲区不会以一个struct virtio\_net\_hdr开头。
2. 如果***num\_buffers***为1，则整个数据包将包含在此缓冲区中，紧跟struct virtio\_net\_hdr之后。
3. 如果协商了VIRTIO\_NET\_F\_GUEST\_CSUM功能位，则可以设置***flags***中的VIRTIO\_NET\_HDR\_F\_DATA\_VALID位：如果是，则设备已验证数据包校验和。在多个封装协议的情况下，已经验证了一级校验和。

此外，VIRTIO\_NET\_F\_GUEST\_CSUM，TSO4，TSO6，UDP和ECN位可以使能接收校验和，大量接收卸载和支持ECN，这些功能等同于传输校验和、传输分段卸载和ECN功能，如[5.1.6.2](#_5.1.6.2数据包传输)所述：

1. 如果协商了VIRTIO\_NET\_F\_GUEST\_CSUM功能位，则可以设置***flags***中的VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位：如果是，则从***csum\_start***偏移***csum\_offset***处的数据包校验和以及任何前面的校验和已经经过验证。 数据包的校验和不完整，***csum\_start***和***csum\_offset***指示如何计算它（请参阅数据包传输点1）。
2. 如果协商了VIRTIO\_NET\_F\_GUEST\_TSO4，TSO6或UFO选项，则***gso\_type***可以（MAY）是除VIRTIO\_NET\_HDR\_GSO\_NONE之外的其他内容，并且***gso\_size***字段指示所需的MSS（参见数据包传输点2）。

###### 5.1.6.4.1设备要求：传入数据包的处理

如果尚未协商VIRTIO\_NET\_F\_MRG\_RXBUF位，则设备必须（MUST）将***num\_buffers***设置为**1**。

如果已经协商了VIRTIO\_NET\_F\_MRG\_RXBUF位，则设备必须（MUST）设置***num\_buffers***以表示数据包（包括标题）的描述符数量。

设备必须（MUST）同时使用单个接收数据包所使用的所有描述符，通过***num\_buffers***值自动增加所用环中的***idx***。

如果未协商VIRTIO\_NET\_F\_GUEST\_CSUM，则设备必须（MUST）将***flags***设置为***0***，并且应该（SHOULD）提供一个完全校验和的数据包给驱动程序。

如果未协商VIRTIO\_NET\_F\_GUEST\_TSO4，则设备不得（MUST NOT）将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_TCPV4。

如果未协商VIRTIO\_NET\_F\_GUEST\_UDP，则设备不得（MUST NOT）将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_UDP。

如果未协商VIRTIO\_NET\_F\_GUEST\_TSO6，则设备不得（MUST NOT）将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_TCPV6。

设备不应该（SHOULD NOT）向驱动程序发送需要分段卸载的TCP数据包（其中设置了显式拥塞通知位），除非协商了VIRTIO\_NET\_F\_GUEST\_ECN功能位，在这种情况下，设备必须（MUST）在***gso\_type***中设置VIRTIO\_NET\_HDR\_GSO\_ECN位。

如果已协商VIRTIO\_NET\_F\_GUEST\_CSUM功能位，则设备可以（MAY）在***flags***中设置VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位，如果是这样：

1.设备必须（MUST）验证来自***csum\_start***的偏移量***csum\_offset***的数据包校验和以及所有先前的偏移量;

2.设备必须（MUST）将存储在接收缓冲区中的数据包校验和设置为TCP / UDP伪标头;

3、设备必须（MUST）设置***csum\_start***和***csum\_offset***，以便从***csum\_start***开始到数据包结束计算一个补码校验和并且将结果存储在***csum\_start***的偏移量***csum\_offset***中，由此将得到完全校验和的数据包;

如果没有协商VIRTIO\_NET\_F\_GUEST\_TSO4，TSO6或UFO选项，则设备必须（MUST）将***gso\_type***设置为VIRTIO\_NET\_HDR\_GSO\_NONE。

如果***gso\_type***与VIRTIO\_NET\_HDR\_GSO\_NONE不同，那么设备也必须（MUST）在***flags***中设置VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位，也必须（MUST）设置***gso\_size***以指示所需的MSS。

如果已协商VIRTIO\_NET\_F\_GUEST\_TSO4，TSO6或UFO选项的其中一个，则设备应该（SHOULD）将***hdr\_len***设置为不小于标头长度的值，包括传输标头。

如果已协商VIRTIO\_NET\_F\_GUEST\_CSUM功能位，则设备可以（MAY）在***flags***中设置VIRTIO\_NET\_HDR\_F\_DATA\_VALID位，如果是这样，则设备必须（MUST）验证数据包校验和（在多个封装协议的情况下，验证一级校验和）。

###### 5.1.6.4.2驱动程序要求：传入数据包的处理

驱动程序必须（MUST）忽略它无法识别的***flags***。

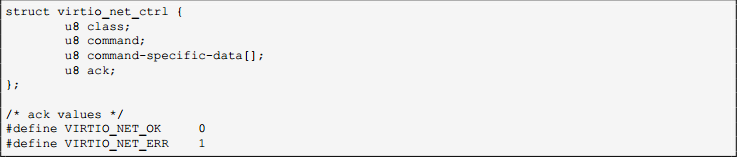
如果未设置flags中的VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM位，则驱动程序不得（MUST NOT）使用***csum\_start***和***csum\_offset***。

如果已协商VIRTIO\_NET\_F\_GUEST\_TSO4，TSO6或UFO选项其中之一，则驱动程序可以（MAY）仅仅将***hdr\_len***用作传输标头大小的提示。驱动程序绝不能依赖***hdr\_len***来更正。

注意：这是由于实现中的各种**bugs**。

如果既未设置VIRTIO\_NET\_HDR\_F\_NEEDS\_CSUM也未设置VIRTIO\_NET\_HDR\_F\_DATA\_VALID，则驱动程序绝不能依赖数据包校验和来更正。

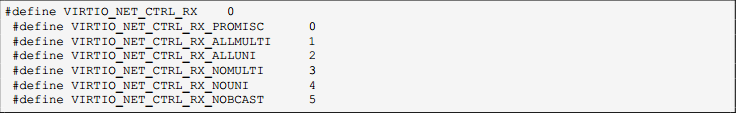
##### 5.1.6.5控制Virtqueue

 驱动程序使用控制virtqueue（如果VIRTIO\_NET\_CTRL\_VQ位被协商）发送命令操纵设备的各种功能（features），这些功能不容易映射到配置空间。 所有命令都具有以下形式：

类（***class***），命令（***command***）和命令特定数据（***command-specific-data***）由驱动程序设置，设备设置***ack***字节。如果***ack***不是VIRTIO\_NET\_OK的话，设备除了发出诊断外，几乎无能为力。

###### 5.1.6.5.1数据包接收过滤

如果协商了VIRTIO\_NET\_F\_CTRL\_RX和VIRTIO\_NET\_F\_CTRL\_RX\_EXTRA功能位，则驱动程序可以发送混杂模式，组播，单播和广播接收的控制命令。

 **注意**：通常，这些命令是最有效的：不需要的数据包仍然可以到达。

5.1.6.5.1.1设备要求：数据包接收过滤

如果已协商VIRTIO\_NET\_F\_CTRL\_RX功能位，则设备必须（MUST）支持以下VIRTIO\_NET\_CTRL\_RX类（class）命令：

•VIRTIO\_NET\_CTRL\_RX\_PROMISC打开和关闭混杂模式。命令特定数据（command-specific-data）是一个包含0（关闭）或1（打开）的字节。如果启用了混杂模式，则设备应该接收所有传入的数据包。即使由VIRTIO\_NET\_CTRL\_RX类命令设置的其他模式打开，这也应该生效。

•VIRTIO\_NET\_CTRL\_RX\_ALLMULTI打开和关闭全组播接收。命令特定数据（command-specific-data）是一个包含0（关闭）或1（打开）的字节。当设备上的全组播接收（all-multicast receive）打开时，应该允许所有传入的组播数据包。

如果已协商VIRTIO\_NET\_F\_CTRL\_RX\_EXTRA功能位，则设备必须支持以下VIRTIO\_NET\_CTRL\_RX类命令：

·VIRTIO\_NET\_CTRL\_RX\_ALLUNI打开和关闭全单播接收。 命令特定数据（command-specific-data）是一个包含0（关闭）或1（打开）的字节。 当设备上的全单播接收打开时，应该允许所有传入的单播数据包

•VIRTIO\_NET\_CTRL\_RX\_NOMULTI禁止组播接收。命令特定数据（command-specific-data）是包含0（允许组播接收）或1（组播接收抑制）的一个字节。当抑制组播接收时，设备不应该向驱动程序发送组播数据包。即使VIRTIO\_NET\_CTRL\_RX\_ALLMULTI打开时，这也是生效的。此过滤器不应该（SHOULD NOT）应用于广播数据包。

•VIRTIO\_NET\_CTRL\_RX\_NOUNI禁止单播接收。命令特定数据（command-specific-data）是一个字节，包含0（允许单播接收）或1（单播接收抑制）。当单播接收被抑制时，设备不应该向驱动程序发送单播数据包。即使VIRTIO\_NET\_CTRL\_RX\_ALLUNI打开，这也是生效的。

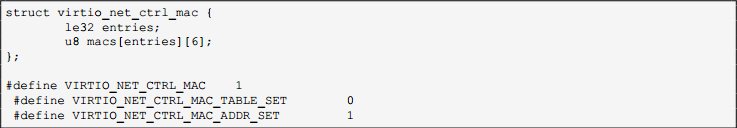
•VIRTIO\_NET\_CTRL\_RX\_NOBCAST禁止广播接收。命令特定数据（command-specific-data）是包含0（允许广播接收）或1（广播接收抑制）的一个字节。当广播接收被抑制时，设备不应该向驱动程序发送广播数据包。即使VIRTIO\_NET\_CTRL\_RX\_ALLMULTI打开，这也是生效。

5.1.6.5.1.2驱动程序要求：数据包接收过滤

如果尚未协商VIRTIO\_NET\_F\_CTRL\_RX功能位，则驱动程序不得（MUST NOT）发出命令VIRTIO\_NET\_CTRL\_RX\_PROMISC或VIRTIO\_NET\_CTRL\_RX\_ALLMULTI。

如果尚未协商VIRTIO\_NET\_F\_CTRL\_RX\_EXTRA功能位，则驱动程序不得（MUST NOT）发出命令VIRTIO\_NET\_CTRL\_RX\_ALLUNI，VIRTIO\_NET\_CTRL\_RX\_NOMULTI，VIRTIO\_NET\_CTRL\_-RX\_NOUNI或VIRTIO\_NET\_CTRL\_RX\_NOBCAST。

###### 5.1.6.5.2配置MAC地址过滤

 如果协商了VIRTIO\_NET\_F\_CTRL\_RX功能位，则驱动程序可以发送用于MAC地址过滤的控制命令。

设备可以通过任意数量的目标MAC地址过滤传入的数据包[[5]](#footnote-4)。 使用类VIRTIO\_NET\_CTRL\_MAC和命令VIRTIO\_NET\_CTRL\_MAC\_TABLE\_SET设置此表。 命令特定数据（command-specific-data）是两个6字节MAC地址的可变长度表（如struct virtio\_net\_ctrl\_mac中所述）。 第一个表包含单播地址，第二个表包含多播地址。VIRTIO\_NET\_CTRL\_MAC\_ADDR\_SET命令用于设置***rx***过滤接收的默认的MAC地址（如果已经协商了VIRTIO\_NET\_F\_MAC\_ADDR，则这将在配置空间中的mac中反映出来）。

VIRTIO\_NET\_CTRL\_MAC\_ADDR\_SET的命令特定数据（command-specific-data）是6字节MAC地址。

5.1.6.5.2.1设备要求：设置MAC地址过滤

设备在复位时必须（MUST）有一个空的MAC过滤表。

设备（MUST）必须在使用VIRTIO\_NET\_CTRL\_MAC\_TABLE\_SET命令之前更新MAC过滤表。

如果已协商VIRTIO\_NET\_F\_MAC\_ADDR位,设备必须（MUST）在配置空间中更新***mac***才能使用VIRTIO\_NET\_CTRL\_MAC\_ADDR\_SET命令。

设备应该丢弃传入的数据包，这些数据包的目标MAC既不匹配***mac***（或设置为VIRTIO\_NET\_CTRL\_MAC\_ADDR\_SET），也不匹配MAC过滤表。

5.1.6.5.2.2驱动程序要求：设置MAC地址过滤

如果尚未协商VIRTIO\_NET\_F\_CTRL\_RX位，则驱动程序不得（MUST NOT）发出VIRTIO\_NET\_CTRL\_MAC类命令。

如果已协商VIRTIO\_NET\_F\_CTRL\_RX，如果VIRTIO\_NET\_CTRL\_MAC\_ADDR\_SET与***mac***不同，则驱动程序应（SHOULD）将VIRTIO\_NET\_CTRL\_MAC\_ADDR\_SET设置默认mac。

驱动程序必须（MUST）以le32型数跟随VIRTIO\_NET\_CTRL\_MAC\_TABLE\_SET命令，紧跟着是非组播MAC地址的数量，接着是另一个le32型数，然后跟该组播地址数。两个数字都可以（MAY）是0。

5.1.6.5.2.3传统接口：设置MAC地址过滤

使用旧版接口时，过渡设备和驱动程序根据guest的本地端（native endian）而不是（必须在不使用传统的接口时）小尾数（little-endian）必须（MUST）在struct virtio\_net\_ ctrl\_mac中格式化***entries***。

当NIC接受传入数据包时，未协商VIRTIO\_NET\_F\_CTRL\_MAC\_ADDR位的旧版驱动程序在配置空间中更改了***mac***。 这些驱动程序总是从第一个字节到最后一个字节写入***mac***值，因此，在检测到这样的驱动程序之后，过渡设备可以推迟MAC更新，或者可以推迟处理传入的数据包，直到驱动程序在配置空间中写入***mac***的最后一个字节。

###### 5.1.6.5.3 VLAN过滤

 如果驱动程序协商VIRTION\_NET\_F\_CTRL\_VLAN功能位，它可以控制设备中的VLAN过滤器表。

VIRTIO\_NET\_CTRL\_VLAN\_ADD和VIRTIO\_NET\_CTRL\_VLAN\_DEL命令都将little-endian 16位VLAN id作为命令特定数据（command-spesific-data）。

5.1.6.5.3.1传统接口：VLAN过滤

使用传统接口时，过渡设备和驱动程序必须根据guest的本地端（native-endian）而不是（必须在不使用传统接口时）小尾数（little-endian）来格式化VLAN id。

###### 5.1.6.5.4无偿数据包发送

 如果驱动程序协商VIRTIO\_NET\_F\_GUEST\_ANNOUNCE（取决于VIRTIO\_NET\_F\_CTRL\_VQ），则设备可以要求驱动程序发送免费数据包; 这通常在guest进行物理迁移后完成，并且需要在新的网络链接上公布其存在。 （因为管理程序没有具有guest网络配置的能力（例如标记的vlan），以这种方式刺激guest是最简单的）。

当驱动程序注意到设备配置的更改时，驱动程序会检查设备配置状态字段中的VIRTIO\_NET\_S\_ANNOUNCE位。命令VIRTIO\_NET\_CTRL\_ANNOUNCE\_ACK用于指示驱动程序已收到通知并且用于设备清除***status***中的VIRTIO\_NET\_S\_ANNOUNCE位。

处理此通知涉及以下两个方面：

1.发送免费数据包（例如ARP）或标记有待发送的待处理无偿数据包并让延迟的例程发送给它们。

2.通过控制vq发送VIRTIO\_NET\_CTRL\_ANNOUNCE\_ACK命令。

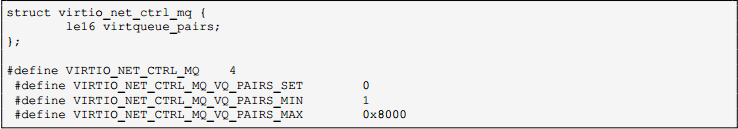
5.1.6.5.4.1驱动程序要求：无偿数据包发送

如果驱动程序协商VIRTIO\_NET\_F\_GUEST\_ANNOUNCE，它应该（SHOULD）在看到***status***中的VIRTIO\_NET\_S\_ANNOUNCE位后，通知网络同行的新位置。 驱动程序必须（MUST）使用类VIRTIO\_NET\_CTRL\_ANNOUNCE和命令VIRTIO\_NET\_CTRL\_ANNOUNCE\_ACK在命令队列（command queue）上发送命令。

5.1.6.5.4.2设备要求：无偿数据包发送

如果协商了VIRTIO\_NET\_F\_GUEST\_ANNOUNCE，在将缓冲区标记为已使用之前，设备必须在收到类VIRTIO\_NET\_CTRL\_ANNOUNCE的命令缓冲区后清除***status***中的VIRTIO\_NET\_S\_ANNOUNCE位。在将缓冲区标记为已使用之前命令VIRTIO\_NET\_CTRL\_ANNOUNCE\_ACK。

###### 5.1.6.5.5多队列模式下的自动接收转向

 如果驱动程序协商VIRTIO\_NET\_F\_MQ功能位（取决于VIRTIO\_NET\_F\_CTRL\_VQ），它可以在多个transmitq1 ... transmitqN中的一个上发送输出数据包，并要求设备排队输入数据包进入到多个receiveq1 ... receiveqN之一中，具体有多少取决于数据包流。

默认情况下禁用多队列。 驱动程序通过执行VIRTIO\_NET\_CTRL\_MQ\_VQ\_PAIRS\_SET命令来启用多队列，指定最多使用的发送和接收队列的数量***max\_virtqueue\_pairs***; 随后， transmitq1 ... transmitqn和receiveq1 ... receiveqn可以被使用，其中n = ***virtqueue\_pairs***。

启用多队列后，设备必须（MUST）使用基于数据包流的自动接收转向（automatic receive steering）。 接收转向分类器的编程是隐含的。 驱动程序在transmitqX上传输数据包流之后，设备应该（SHOULD）引导该传入数据包流到receiveqX。 对于单向协议，或尚未发送数据包，设备可以将数据包引导到指定的receiveq1 ... receiveqn以外的随机队列。

通过将***virtqueue\_pairs***设置为1（这是默认值）并等待设备使用命令缓冲区来禁用多队列。

5.1.6.5.5.1驱动程序要求：多队列模式下自动接收转向

在使用VIRTIO\_NET\_CTRL\_MQ\_VQ\_PAIRS\_SET命令启用它们之前，驱动程序必须（MUST）配置virtqueues。

驱动程序不得（MUST NOT）在设备配置空间中使得***virtqueue\_pairs =*0**或大于***max\_virtqueue\_pairs***。

在使用VIRTIO\_NET\_CTRL\_MQ\_VQ\_PAIRS\_SET命令之前，驱动程序必须仅在任意的transmitq1…上排列数据包。

一旦将VIRTIO\_NET\_CTRL\_MQ\_VQ\_PAIRS\_SET命令放入可用环（available ring）中，驱动程序就不得（MUST NOT）在大于***virtqueue\_pairs***的传输queues上排列数据包。

5.1.6.5.5.2设备要求：多队列模式下的自动接收转向

在使用VIRTIO\_NET\_CTRL\_MQ\_VQ\_PAIRS\_SET命令之前，设备必须仅在任意的receiveq1…上排列数据包。

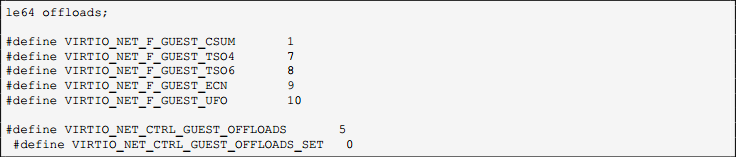
一旦设备将VIRTIO\_NET\_CTRL\_MQ\_VQ\_PAIRS\_SET命令放入已使用环（used ring）中，设备就不得（MUST NOT）在大于***virtqueue\_pairs***的接收queues上排列数据包。

5.1.6.5.5.3传统接口：多队列模式下自动接收转向

在使用传统接口时，过渡设备和驱动程序必须（MUST）根据guest的本地端（native endian），而不是（不使用传统接口时有必要）小尾数（little-endian）来格式化***virtqueue\_pairs***。

###### 5.1.6.5.6卸载状态配置

如果协商了VIRTIO\_NET\_F\_CTRL\_GUEST\_OFFLOADS功能位，则驱动程序可以发送用于动态卸载状态配置的控制命令。

5.1.6.5.6.1设置卸载状态

VIRTIO\_NET\_CTRL\_GUEST\_OFFLOADS类（class）有一个命令：VIRTIO\_NET\_CTRL\_GUEST\_OFFLOADS\_SET应用于新的卸载配置。

作为命令数据传递的le64类型值是一个位掩码，“设置定义卸载”位要使能，“清除-卸载”位要禁用。

每个卸载都有相应的设备feature。在feature协商之后，相应的卸载被启用来保持向后的兼容性。

5.1.6.5.6.2驱动程序要求：设置卸载状态

驱动程序不得启用尚未协商相应功能的卸载。

5.1.6.5.6.3传统接口：设置卸载状态

使用传统接口时，过渡设备和驱动程序必须（MUST）根据guest的本地端（native endian）而不是（不使用传统接口时有必要）little-endian来格式化***offloads***。

5.1.6.6传统接口：框架要求

使用传统接口时，尚未协商VIRTIO\_F\_ANY\_LAYOUT位的过渡驱动程序必须（MUST）在发送和接收上都使用对struct virtio\_net\_hdr的单个描述符，并使用以下描述符中的网络数据。

此外，在使用control virtqueues（[参见5.1.6.5](#_5.1.6.5控制Virtqueue)）时，尚未协商VIRTIO\_F\_ANY\_LAYOUT的过渡驱动程序必须（MUST）：

•对于所有命令，使用单个2字节的描述符，包括前两个字段：***class***和***command***

•对于除VIRTIO\_NET\_CTRL\_MAC\_TABLE\_SET之外的所有命令，包括没有填充（padding）的（command-specific-data），使用单个描述符。

•对于VIRTIO\_NET\_CTRL\_MAC\_TABLE\_SET命令，恰好使用两个描述符，包括没有填充（padding）的（command-specific-data）：这些描述符中的第一个必须（MUST）包含没有填充（padding）的单播地址的virtio\_net\_ctrl\_mac表结构，这些描述符中的第二个必须（MUST）包括没有填充（padding）的多播地址的virtio\_net\_ctrl\_mac表结构。

•对于所有命令，对***ack***字段使用单个的1-字节描述符。

见2.4.4。

### 5.2 块设备

virtio块设备是一个简单的虚拟块设备（即磁盘）。读取和写入请求（以及其他外部请求）被放置在队列中，并由设备进行服务（可能是无序的），除非另有说明。

#### 5.2.1设备ID

2

#### 5.2.2 虚拟队列

0 requestq

#### 5.2.3 功能位

**VIRTIO\_BLK\_F\_SIZE\_MAX(1)** 任何单个段的最大大小为size\_max。

**VIRTIO\_BLK\_F\_SEG\_MAX(2)** 请求中的最大段数位于seg\_max中。

**VIRTIO\_BLK\_F\_GEOMETRY(4)** 几何中指定的磁盘样式几何。

**VIRTIO\_BLK\_F\_RO(5)** 设备为只读的。

**VIRTIO\_BLK\_F\_BLK\_SIZE(6)** 磁盘块大小位于blk\_size中。

**VIRTIO\_F\_FLUSH(9)** 缓存刷新命令支持。

**VIRTIO\_BLK\_F\_TOPOLOGY(10)** 设备导出有关最佳I / O对齐的信息。

**VIRTIO\_BLK\_F\_CONFIG\_WCE(11)** 设备可以在回写和写入模式之间切换其缓存。

##### 5.2.3.1 旧版接口：功能位

**VIRTIO\_BLK\_F\_BARRIER(0)** 设备支持请求屏障。

**VIRTIO\_BLK\_F\_SCSI(7)** 设备支持scsi包命令。

缓存刷新命令支持。

设备可以在回写和写入模式之间切换其缓存。

**注：**在旧版接口中，VIRTIO\_BLK\_F\_FLUSH也被称为VIRTIO\_BLK\_F\_WCE：

旧版驱动程序必须只在能够发送VIRTIO\_BLK\_T\_FLUSH命令时才协商此功能。

#### 5.2.4 设备配置布局

设备的容量（以512字节扇区表示）始终存在。其他的可用性都取决于如上所述的各种特征位。上表

struct virtio\_blk\_config {

le64 capacity;

le32 size\_max;

le32 seg\_max;

struct virtio\_blk\_geometry {

le16 cylinders;

u8 heads;

u8 sectors;

} geometry;

le32 blk\_size;

struct virtio\_blk\_topology {

// # of logical blocks per physical block (log2)

u8 physical\_block\_exp;

// offset of first aligned logical block

u8 alignment\_offset;

// suggested minimum I/O size in blocks

le16 min\_io\_size;

// optimal (suggested maximum) I/O size in blocks

le32 opt\_io\_size;

} topology;

u8 reserved;

};

##### 5.2.4.1 旧版接口：设备配置布局

struct virtio\_blk\_config {

le64 capacity;

le32 size\_max;

le32 seg\_max;

struct virtio\_blk\_geometry {

le16 cylinders;

u8 heads;

u8 sectors;

} geometry;

le32 blk\_size;

struct virtio\_blk\_topology {

// # of logical blocks per physical block (log2)

u8 physical\_block\_exp;

// offset of first aligned logical block

u8 alignment\_offset;

// suggested minimum I/O size in blocks

le16 min\_io\_size;

// optimal (suggested maximum) I/O size in blocks

le32 opt\_io\_size;

} topology;

u8 writeback;

};

使用旧版接口时，过渡设备与驱动程序**必须**根据访客本机端字节序格式格式化sturct\_virtio\_blk\_config中的字段，而不是使用小段格式进行格式化。

#### 5.2.5 设备初始化

1. 可以从容量中读取设备大小。

2. 如果协商了VIRTIO\_BLK\_F\_BLK\_SIZE功能，则可以通过读取blk\_size来确定要使用的驱动程序的最佳扇区大小。这不会影响协议中使用的单位（总是512字节），但是对正确值的了解会影响性能。

3. 如果设备设置了VIRTIO\_BLK\_F\_RO功能，则任何写入请求都将失败。

4. 如果协商了VIRTIO\_BLK\_F\_TOPOLOGY功能，则可以通过读取拓扑结构中的字段来确定驱动程序要使用的物理块大小和最佳I/O长度。这也不会影响协议中的单位，只会影响性能。

5. 如果协商提供VIRTIO\_BLK\_F\_CONFIG\_WCE功能，则可以通过写回字段读取或设置高速缓存模式。0对应于写通缓存，1对应于写回缓存。复位后的缓存模式可以是回写或写入。实际模式可以通过在特征协商后读取回写来确定。

##### 5.2.5.1 旧版接口：设备初始化

保留字段曾被称为回写。

##### 5.2.5.1 驱动要求：设备初始化

如果驱动程序无法发送VIRTIO\_BLK\_T\_FLUSH命令，则**不应**协商VIRTIO\_BLK\_F\_FLUSH。

如果VIRTIO\_BLK\_F\_CONFIG\_WCE和VIRTIO\_BLK\_F\_FLUSH均没有被协商，驱动程序**可以**推断出存在写入缓存。如果VIRTIO\_BLK\_F\_\_CONFIG\_WCE没有被协商，但是VIRTIO\_BLK\_F\_FLUSH已协商，则驱动程序**应该**假设存在回写缓存。

在设置FEATURES\_OK状态位之前，驱动程序不得读取回写。

##### 5.2.5.2 设备要求：设备初始化

设备**应该**总是提供VIRTIO\_BLK\_F\_FLUSH，如果它们提供了VIRTIO\_BLK\_F\_CONFIG\_WCE，则**必须**提供它。

如果VIRTIO\_BLK\_F\_CONFIG\_WCE已被协商而VIRTIO\_BLK\_F\_FLUSH没有协商，则设备必须将回写初始化为0。

##### 5.2.5.3 旧版接口：设备初始化

由于旧版设备没有FEATURES\_OK，因此在通过旧版接口使用设备时，设备**必须**在功能协商时才去略微不同的行为。特别是，当使用旧版接口时：

·在设置DRIVER或DRIVER\_OK状态位之前，驱动程序**可以**读或写入回写。

·由于驱动程序设置了状态位，除非DRIVER\_OK位置1且驱动程序未设置VIRTIO\_BLK\_F\_CONFIG\_WCE驱动程序功能位，否则设备**不得**修改缓存模式（和写回）。

·由于驱动程序对驱动功能位进行了修改，设备**不得**修改缓存模式（和写回），例如，如果驱动程序设置了VIRTIO\_BLK\_F\_CONFIG\_WCE驱动功能位，但是没有设置VIRTIO\_BLK\_F\_FLUSH。

#### 5.2.6 设备操作

驱动程序将请求排队到virtqueue，它们由设备使用（不一定按顺序）。 每个请求的形式如下：

struct virtio\_blk\_req {

le32 type;

le32 reserved;

le64 sector;

u8 data[][512];

u8 status;

};

请求的类型为读取（VIRTIO\_BLK\_T\_IN），写入（VIRTIO\_BLK\_T\_OUT）或刷新（VIRTIO\_BLK\_T\_FLUSH）。

#define VIRTIO\_BLK\_T\_IN 0

#define VIRTIO\_BLK\_T\_OUT 1

#define VIRTIO\_BLK\_T\_FLUSH 4

扇区号表示要读或写发生的位置的偏移量（乘以512）。此字段未使用，并且对于scsi数据包命令和flush命令设置为0。

最终状态字节由设备写入：VIRTIO\_BLK\_S\_OK表示成功，VIRTIO\_BLK\_S\_IOERR表示设备或驱动程序错误，VIRTIO\_BLK\_S\_UNSUPP表示设备不支持的请求：

#define VIRTIO\_BLK\_S\_OK 0

#define VIRTIO\_BLK\_S\_IOERR 1

#define VIRTIO\_BLK\_S\_UNSUPP 2

##### 5.2.6.1 驱动要求：设备操作

驱动程序**不得**提交会导致读取或写入超出容量的请求。

如果提供了VIRTIO\_BLK\_F\_RO功能，则驱动程序**应该**接受它。

对于VIRTIO\_BLK\_T\_FLUSH请求，驱动程序**必须**将扇区设置为0。驱动程序不**应**在VIRTIO\_BLK\_T\_FLUSH请求中包含任何数据。

如果协商了VIRTIO\_BLK\_F\_CONFIG\_WCE功能，则驱动程序可以通过分别将0和1写入写回字段来切换到写入或回写模式。在回写中写入0后，驱动程序**不得**进行任何易失性写入已向持久设备后端存储提交的假设。

##### 5.2.6.2 设备要求：设备操作

如果提供了VIRTIO\_BLK\_F\_RO功能，设备必须通过将状态字节设置为VIRTIO\_BLK\_S\_IOERR来进行写请求，并且不得写入任何数据。

写入在提交时被视为易失性的；在写入稳定之前，在持久性设备后端存储中未定义易失性写入所覆盖的扇区的内容。一旦写入完成并且满足以下一个或多个条件，写入就会变得稳定：

1. VIRTIO\_BLK\_F\_CONFIG\_WCE和VIRTIO\_BLK\_F\_FLUSH功能均未协商，但设备提供了VIRTIO\_BLK\_F\_FLUSH；

2. 已协商VIRTIO\_BLK\_F\_CONFIG\_WCE功能，且在提交写入和完成的期间内，配置空间中的写回字段都为0；

3. 写入完成后发送VIRTIO\_BLK\_T\_FLUSH请求并自行完成。

4. 如果设备由持久存储支持，则设备必须确保在报告写入完成（情况1和2）或刷新（情况3）之前向其提交稳定写入。如果不这样做，可能会在发生崩溃时导致数据丢失。

如果驱动程序在写入提交和完成之间更改回写，则在报告完成时写入可能是易失性的或稳定的；换句话说，实际的行为是不确定的。

如果设备未提供VIRTIO\_BLK\_F\_FLUSH，则设备也可以在报告完成之前提交对持久设备\后端存储的写入。然而，与情况1不同，这不是规范所规定的绝对要求。

**注：**如果发生崩溃，不提供向工具VIRTIO\_BLK\_F\_FLUSH且未提交完成写入将不具有数据丢失的弹性。不向工具提供VIRTIO\_BLK\_F\_FLUSH是对不希望发生这种数据丢失安全绝对要求。

##### 5.2.6.3 旧版接口：设备操作

使用旧版接口时，过渡设备和驱动程序必须根据访客的本机端字节序格式化struct virtio\_blk\_req中的字段，而不是（必须在没有使用旧版接口时）小端字节序。

使用旧版接口时，过渡驱动程序应该忽略已用环项中的len值。

**注：**从历史上看，有些设备将总描述符长度或设备可写缓冲区的总长度放在那里，即使实际上只写入状态字节也是如此。

保留字段以前称为ioprio。 ioprio是关于设备请求的相对优先级的提示：数字越大表示请求越重要。

#define VIRTIO\_BLK\_T\_FLUSH\_OUT 5

命令VIRTIO\_BLK\_T\_FLUSH\_OUT是VIRTIO\_BLK\_T\_FLUSH的同义词；驱动程序必须将其视为VIRTIO\_BLK\_T\_FLUSH命令。

#define VIRTIO\_BLK\_T\_BARRIER 0x80000000

如果设备具有VIRTIO\_BLK\_F\_BARRIER功能，则高位（VIRTIO\_BLK\_T\_BARRIER）表示此请求充当屏障，并且所有先前的请求**应该**在此之前完成，并且所有后续请求在此完成之前**不应**启动。

**注：**屏障不会刷新主机中底层后端设备中的缓存，因此不能作为数据一致性保证。只有VIRTIO\_BLK\_T\_FLUSH请求才能这样做。

当提供VIRTIO\_BLK\_F\_FLUSH但未协商时，一些较旧的旧版设备不会提交对持久性设备后端存储的完整写入。为了解决这个问题，驱动程序**可以**将回写设置为0（如果可用），或者**可以**在每次完成写入后发送显式刷新请求。

如果设备具有VIRTIO\_BLK\_F\_SCSI功能，它还可以支持scsi数据包命令请求，这些请求中的每一个都具有以下形式：

/\* All fields are in guest's native endian. \*/

struct virtio\_scsi\_pc\_req {

u32 type;

u32 ioprio;

u64 sector;

u8 cmd[];

u8 data[][512];

#define SCSI\_SENSE\_BUFFERSIZE 96

u8 sense[SCSI\_SENSE\_BUFFERSIZE];

u32 errors;

u32 data\_len;

u32 sense\_len;

u32 residual;

u8 status;

};

请求类型也可以是scsi包命令（VIRTIO\_BLK\_T\_SCSI\_CMD或VIRTIO\_BLK\_T\_SCSI\_CMD\_OUT）。这两种类型是等价的，设备不对它们进行区分：

#define VIRTIO\_BLK\_T\_SCSI\_CMD 2

#define VIRTIO\_BLK\_T\_SCSI\_CMD\_OUT 3

cmd字段仅用于scsi数据包命令请求，并指示要执行的命令。该字段必须位于单独的设备可读缓冲区中；命令长度可以从该缓冲区的长度导出。

请注意，前三个（scsi数据包命令有四个）字段始终是设备可读的：数据是设备可读的或设备可写的，具体取决于请求。读取或写入的大小可以从请求缓冲区的总大小中导出。

sense仅用于scsi包命令请求中，指示scsi检测数据的缓冲区。

data\_len仅用于scsi数据包命令请求，不推荐使用此字段，驱动程序应该忽略该字段。 从历史上看，设备从那里复制数据的长度。

sense\_len仅用于scsi包命令请求，指示实际写入检测缓冲区的字节数。

residual字段仅用于scsi数据包命令请求中，指示剩余大小，计算为数据长度-实际传输的字节数。

##### 5.2.6.4 旧版接口：框架要求

使用旧版接口时，尚未协商VIRTIO\_F\_ANY\_LAYOUT的过渡驱动程序：

·必须使用包括类型，保留和扇区的单个8字节描述符，后跟数据描述符，最后是单独1字节的状态描述符

·对于SCSI命令，还存在其他约束。errors，data\_len，sense\_len和residual必须博保存在一个单独的设备可写描述符中，sense必须保存在一个大小为96字节的单独的可设置可编写描述符中，而errors，data\_len，sense\_len和residual必须保存在一个单独的设备可写描述符中。

见2.4.4。

### 5.3控制台设备

virtio控制台设备是一个用于数据输入和输出的简单设备。 设备可以（MAY）有一个或多个端口。 每个端口都有一对输入和输出功能。 此外，设备具有一对控制IO virtqueues。 控制virtqueues用于设备和驱动程序之间传递有关连接两侧打开和关闭的端口信息，从设备指示特定端口是否是控制台端口、添加新端口、或者热插拔/ 拔出端口等等，以及驱动程序指示端口或设备是否已成功添加、打开/关闭等的指示。对于数据IO，一个或多个空缓冲区放置在接收队列（放置传入数据）中，传出字符则放在传输队列中。

#### 5.3.1设备ID

3

#### 5.3.2 Virtqueues

**0**  receiveq(port0)

**1** transmitq(port0)

**2** control receiveq

**3** control transmitq

**4** receiveq(port1)

**5**  transmitq(port1)

...

端口0的接收和发送队列始终存在：仅在设置了VIRTIO\_CONSOLE\_F\_MULTIPORT位时才存在其他队列。

#### 5.3.3功能位

**VIRTIO\_CONSOLE\_F\_SIZE（0**）配置列和行有效。 **VIRTIO\_CONSOLE\_F\_MULTIPORT（1）**设备支持多个端口; ***max\_nr\_ports***有效，control virtqueues将被使用。

**VIRTIO\_CONSOLE\_F\_EMERG\_WRITE（2）**设备支持紧急写入。 配置字段emerg\_wr有效。

#### 5.3.4设备配置布局

如果设置了VIRTIO\_CONSOLE\_F\_SIZE功能位，则会在配置空间中提供控制台的大小。 此外，如果设置了VIRTIO\_CONSOLE\_F\_MULTIPORT功能位，则可以获取设备支持的最大端口数。

 如果设置了VIRTIO\_CONSOLE\_F\_EMERG\_WRITE位，则驱动程序可以使用紧急写入来输出单个字符，而无需初始化virtio queues，甚至承认该功能。

##### 5.3.4.1传统接口：设备配置布局

使用传统接口时，过渡设备和驱动程序必须（MUST）根据guest虚拟机的本机端（native endian）而不是（必要时不使用旧版接口）little-endian来格式化struct virtio\_console\_config中的字段。

#### 5.3.5设备初始化

1、如果提供了VIRTIO\_CONSOLE\_F\_EMERG\_WRITE功能，则配置的***emerg\_wr***字段可以随时写。因此，它适用于在启动调试输出时以及灾难性OS故障（例如，virtio环损坏）。

2、如果协商了VIRTIO\_CONSOLE\_F\_SIZE功能，则驱动程序可以从***cols***和***rows***读取控制台尺寸。

3、如果协商了VIRTIO\_CONSOLE\_F\_MULTIPORT功能，则驱动程序可以生成多个端口，并非所有端口都必须连接到控制台。有些可能是通用端口。在这种情况下，启用control virtqueues，并根据***max\_nr\_ports***，创建适当数量的virtqueues。指示驱动就绪的控制消息被发送到设备。然后，设备可以发送控制消息，以便向设备添加新端口。创建并初始化每个端口后，会向该端口的设备发送VIRTIO\_CONSOLE\_PORT\_READY控制消息，以便可以让驱动程序知道为该端口设置的任何其他配置选项。

4、每个端口的receiveq填充一个或多个接收缓冲区。

##### 5.3.5.1设备要求：设备初始化

即使在未配置的设备上，设备也必须（MUST）允许写入***emerg\_wr***。

设备应该（SHOULD）将写入***emerg\_wr***的低位字节发送到适当的log或输出method。

#### 5.3.6设备操作

1、对于输出，包含字符的缓冲区放在端口的transmitq[[6]](#footnote-5)中。

2、当在receiveq中使用缓冲区（由中断发出信号）时，内容是与接收通知的virtqueue相关联端口的输入。

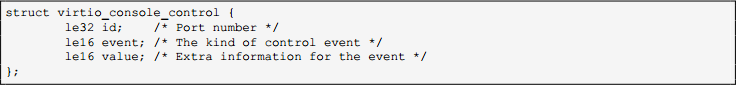
3、如果驱动程序协商了VIRTIO\_CONSOLE\_F\_SIZE功能，则配置更改中断表示可以从配置字段中读取更新的大小。 此大小仅适用于端口**0**。

4、如果驱动程序协商了VIRTIO\_CONSOLE\_F\_MULTIPORT功能，则设备使用VIRTIO\_CONSOLE\_PORT\_ADD控制消息通知活动（active）端口。 相同的消息也用于端口热插拔（hot-plug）。

##### 5.3.6.1驱动程序要求：设备操作

驱动程序不得（MUST NOT）将设备可读的内容放入receiveq中。 驱动程序绝不能（MUST NOT）将设备可写缓冲区放在transmitq中。

##### 5.3.6.2多端口设备操作

 如果驱动程序协商了VIRTIO\_CONSOLE\_F\_MULTIPORT功能，则两个control queues用于操作不同的控制台端口：control receiveq用于从设备到驱动程序的信息，以及control sendq用于驱动程序到设备的信息。 控制消息的设计是：

***event***的值是：

**VIRTIO\_CONSOLE\_DEVICE\_READY（0）**由驱动程序在初始化时发送，表示它已准备好接收控制消息。 值**1**表示成功，**0**表示失败。 端口号***id***未使用。

**VIRTIO\_CONSOLE\_DEVICE\_ADD（1）**由设备发送，以创建新端口。 ***Value***值未使用。

**VIRTIO\_CONSOLE\_DEVICE\_REMOVE（2）**由设备发送，以删除现有端口。 ***value***值未使用。

**VIRTIO\_CONSOLE\_PORT\_READY（3）**由驱动程序在响应设备的VIRTIO\_CONSOLE\_PORT\_ADD消息时发送，以指示端口已准备好被使用。 值**1**表示成功，**0**表示失败。

**VIRTIO\_CONSOLE\_CONSOLE\_PORT（4）**由设备发送以指定端口作为控制台端口。 可能（MAY）有多个控制台端口。

 **VIRTIO\_CONSOLE\_RESIZE（5）**由设备发送以指示控制台大小已更改。 ***value***值未使用。缓冲区后跟列数（cols）和行数（rows）：

**VIRTIO\_CONSOLE\_PORT\_OPEN（6）**此消息由设备和驱动程序发送。 ***value***值表示的状态：**0**（端口关闭）或**1**（端口打开）。 这允许客户机和主机进程直接使用端口（以定义应用程序的方式）进行通信。 **VIRTIO\_CONSOLE\_PORT\_NAME（7）**由设备发送给端口一个标记。这个控制命令后面紧跟着端口的UTF-8名称，以便在guest中进行标识（不带NULL终止符）。

###### 5.3.6.2.1设备要求：多端口设备操作

设备不得（MUST NOT）指定存在于一个VIRTIO\_CONSOLE\_DEVICE\_ADD消息中的端口，也不得（MUST NOT）指定端口值等于或大于***max\_nr\_ports***的端口。

设备不得（MUST NOT）在VIRTIO\_CONSOLE\_DEVICE\_REMOVE中指定一个尚未使用先前由VIRTIO\_CONSOLE\_DEVICE\_ADD创建的端口。

###### 5.3.6.2.2驱动程序要求：多端口设备操作

如果协商了VIRTIO\_CONSOLE\_F\_MULTIPORT功能位，驱动程序必须（MUST）发送一个VIRTIO\_CONSOLE\_DEVICE\_READY消息。在收到VIRTIO\_CONSOLE\_CONSOLE\_PORT消息后，驱动程序应该（SHOULD）（以适合文本控制台访问（text console access）的方式）处理端口，并且必须（MUST）使用VIRTIO\_CONSOLE\_PORT\_OPEN消息进行响应，该消息必须将值设置为1。

##### 5.3.6.3传统接口：设备操作

使用传统接口时，过渡设备和驱动程序必须（MUST）根据guest虚拟机的native endian格式化struct virtio\_console\_control中的字段，而不是（在不使用传统接口时有必要）little-endian。使用旧接口时，驱动程序应该（SHOULD）忽略传输队列和控制传输的已使用环条目中的***len***值。

注意：从之前来看，即使实际上没有写入数据，有些设备会将总描述符长度放在那里。

##### 5.3.6.4传统接口：框架要求

使用传统接口时，在尚未协商VIRTIO\_F\_ANY\_LAYOUT时，过渡驱动程序必须（MUST）仅对在control receiveq和control transmitq中的所有缓冲区使用单个描述符。

### 5.4熵设备

virtio熵设备为guest提供高质量的随机性。

#### 5.4.1设备ID

4

#### 5.4.2 Virtqueues

**0** requestq

#### 5.4.3特征位

目前没有定义。

#### 5.4.4设备配置布局

目前没有定义。

#### 5.4.5设备初始化

1.virtqueues初始化。

#### 5.4.6设备操作

当驱动程序需要随机字节时，它会将一个或多个缓冲区的描述符放在队列中。 它将由设备填满随机数据。

##### 5.4.6.1驱动程序要求：设备操作

驱动程序不得（MUST NOT）将驱动程序可读缓冲区放入队列中。驱动程序必须（MUST）检查设备写入的长度，来确定接收到的随机字节数目。

##### 5.4.6.2设备要求：设备操作

设备必须（MUST）将一个或多个随机字节放入缓冲区，但它可以使用少于整个缓冲区长度的随机字节长度。

### 5.5传统内存气球（Memory Balloon）设备

这是传统的balloon设备。 设备号13的保留用于新的内存balloon接口，具有不同的含义，这在标准的未来版本中是可预见的。

传统的virtio内存balloon设备是用于管理guest内存的原始设备：设备要求一定量的内存，驱动程序提供给它（如果设备超过它要求的话，撤回它）。 这允许guest适应底层物理内存容量的变化。 如果协商了该功能，则该设备还可用于将guest内存统计信息传递给主机（host）。

#### 5.5.1设备ID

5

#### 5.5.2 Virtqueues

**0** inflateq

**1** deflateq

**2** statsq.

Virtqueue 2仅在VIRTIO\_BALLON\_F\_STATS\_VQ设置时存在。

#### 5.5.3特征位

**VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST（0）**在使用balloon之前必须告知主机（host）。

**VIRTIO\_BALLOON\_F\_STATS\_VQ（1）**呈现用于报告客户机内存统计信息的一个virtqueue。

**VIRTIO\_BALLOON\_F\_DEFLATE\_ON\_OOM（2）**在guest内存不足的条件下收缩balloon。

##### 5.5.3.1驱动程序要求：特征位

如果设备提供VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能位，驱动程序应该接受VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能。

##### 5.5.3.2设备要求：特征位

如果设备提供VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能位，并且驱动程序不接受此功能位，当驱动程序写入时，设备可能（MAY）会因未设置FEATURES\_OK设备状态（device bits）位而发出故障信号。

5.5.3.2.0.1传统接口：特征位

由于传统接口无法正常报告功能协商失败信息，因此在使用旧接口时，过渡设备必须支持不协商VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能位的guest，而且如果未协商VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST位，则应该允许guest在使用内存之前通知主机（host）。

#### 5.5.4设备配置布局

此配置空间的两个字段始终可用。

5.5.4.0.0.2传统接口：设备配置布局

使用旧接口时，过渡设备和驱动程序必须根据little-endian格式来格式化struct virtio\_balloon\_config中的字段。注意：这与传统设备字段为guest endian的常规约定不同。

#### 5.5.5设备初始化

设备初始化过程概述如下：

1、确定了膨胀和缩小的virtqueues。

2、如果协商了VIRTIO\_BALLOON\_F\_STATS\_VQ功能位：

（a）确定stats virtqueues；

（b）向stats virtqueue添加一个空缓冲区；

（c）设置DRIVER\_OK：设备操作开始；

（d）通知设备有关stats virtqueue缓冲区的信息。

#### 5.5.6设备操作

通过接收配置更改中断或通过更改客户机内存需求（例如执行内存压缩操作或响应内存不足情况）来驱动设备。

1、检查***num\_pages***配置字段。 如果这大于***actual*** pages数，则balloon需要来自guest的更多内存。 如果它小于***actual***值，则balloon不需要全部内存。

2、为balloon提供内存（又名inflate）：

（a）驱动程序构造未使用的内存pages的地址数组。 这些地址除以4096[[7]](#footnote-6)，描述生成的32位数组的描述符被添加到inflateq中。

3、从balloon中移除内存（又称deflate）：

（a）如上所述，驱动程序构造一个它以前提供给balloon的内存pages地址数组。 此描述符将添加到deflateq。

（b）如果协商了VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能，则guest在使用它们之前会通知设备。

（c）否则，允许guest在设备确认退出之前重新使用之前提供给balloon的pages[[8]](#footnote-7)。

4.在任何一种情况下，设备都会通过使用描述符来确认膨胀和收缩请求。

5.一旦设备确认膨胀或收缩，驱动程序就会更新***actual***来反映balloon中的新pages。

##### 5.5.6.1驱动程序要求：设备操作

当***num\_pages***大于balloon中的实际pages时，驱动程序应该（SHOULD）向balloon提供pages。

当***num\_pages***小于balloon中的实际pages时，驱动程序可以（MAY）使用balloon中的pages。

当***num\_pages***大于或等于balloon中的实际pages时，驱动程序可以（MAY）向balloon提供pages。

如果尚未协商VIRTIO\_BALLOON\_F\_DEFLATE\_ON\_OOM位，则当***num\_pages***小于或等于balloon中的实际pages时，驱动程序不得（MUST NOT）使用balloon中的pages。

如果已经协商了VIRTIO\_BALLOON\_F\_DEFLATE\_ON\_OOM，则当***num\_pages***小于或等于balloon中的实际pages时，驱动程序可以（MAY）使用balloon中的pages，如果这是系统稳定性所需的（例如，如果客户机内运行的应用程序需要内存） ，驱动程序必须（MUST）使用deflateq来通知设备它要从balloon中使用的pages。

如果协商了VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能，则在设备确认收缩请求之前，驱动程序不得（MUST NOT）使用balloon中的pages。

否则，如果未协商VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能，则在设备确认收缩请求之前，驱动程序可以（MAY）开始重新使用之前提供给balloon的pages。

在任何情况下，在将pages添加到balloon后，但在设备确认膨胀请求之前，驱动程序不得（MUST NOT）使用balloon中的pages。

在设备确认膨胀请求之前，驱动程序不得（MUST NOT）要求balloon中的pages收缩。

在更改balloon中的pages后，驱动程序必须（MUST）更新***actual***。

在多次膨胀和收缩操作后，驱动程序可以（MAY）更新***actual***一次。

##### 5.5.6.2设备要求：设备操作

在检测到膨胀请求中的物理编号之后以及在使用inflateq描述符确认膨胀请求之前，设备可以（MAY）修改balloon中pages的内容。

如果协商了VIRTIO\_BALLOON\_F\_MUST\_TELL\_HOST功能，则设备可以（MAY）在检测到膨胀请求中的物理编号之后，或者在检测到其收缩请求中的物理编号并确认收缩请求之前修改balloon中pages的内容。

###### 5.5.6.2.1传统接口：设备操作

使用传统接口时，驱动程序应该（SHOULD）忽略使用的环条目中的***len***值。

**注意**：从历史上看，有些设备会将总描述符长度放在那里，即使实际上没有写入数据。

使用传统接口时，驱动程序必须（MUST）在每次使用单个原子操作（atomic operation）更新配置空间中的***actual***值时写出所有4个字节。

使用传统接口时，设备不应（SHOULD NOT）使用在配置空间中由驱动程序写入的***actual***值，直到最后一个最重要的字节被写入之后才行。

**注意**：历史上，设备使用***actual***值，即使使用Virtio Over PCI总线时，设备特定的配置空间也不能保证是原子（atomic）的。除了调试之外，最好避免在驱动程序更新期间使用中间值。

从历史上看，使用Virtio Over PCI总线的驱动程序通过使用多个单字节按顺序写入***actual***值，从最小有效值到最大有效值。

##### 5.5.6.3内存统计

stats virtqueue不典型，因为通信由设备（而不是驱动程序）驱动。当驱动程序添加空缓冲区并且通知设备时，通道在驱动程序初始化时变为活动状态。

内存统计信息请求的过程如下：

1、该设备将缓冲器推到使用过的环上，并发送一个中断。

2、驱动程序弹出使用过的缓冲区并将其丢弃。

3、驱动程序收集内存统计信息并将其写入新的缓冲区。

4、驱动程序将缓冲区添加到virtqueue并通知设备。

5、设备弹出缓冲区（保留缓冲区以启动后续请求），并使用统计信息。

 在缓冲区中，统计信息是一个由6个字节的条目组成的数组。每个统计由一个16位标记(tag)和一个64位值(value)组成。所有统计数据都是可选的，驱动程序选择要提供的统计数据。为了保证向后兼容性，设备省略了不支持的统计信息。

###### 5.5.6.3.1驱动程序要求：内存统计

本节中的规范性声明适用于virtio\_balloon\_f\_stats\_vq功能已经协商的情况。

驱动程序必须（MUST）始终为Statsq中的设备提供最多一个缓冲区。

初始化设备后，驱动程序必须（MUST）在statsq中提供一个输出缓冲区。

在检测到设备在statsq中使用了缓冲区后，驱动程序必须（MUST）在statsq中提供一个输出缓冲区。

在使输出缓冲区在statsq中可用之前，驱动程序必须（MUST）初始化它，包括一个struct virtio\_balloon\_stat条目用于它支持的每个统计数据。

驱动程序必须（MUST）对提交给statsq的所有缓冲区使用6字节倍数输出缓冲区大小。

驱动程序可以（MAY）在提交给statsq的输出缓冲区中以任何顺序提供struct virtio\_balloon\_stat条目，不考虑标签（tag）值。

驱动程序可以（MAY）在提交给statsq的输出缓冲区中提供所有统计信息的子集。

驱动程序必须（MUST）在提交给statsq的所有缓冲区中提供相同的统计子集。

###### 5.5.6.3.2设备要求：内存统计

本节中的规范性声明适用于virtio\_balloon\_f\_stats\_vq功能已经协商的情况。

在提交给statsq的输出缓冲区中，设备必须（MUST）忽略带有其无法识别的标记值的条目。

在提交给statsq的输出缓冲区中，设备必须（MUST）以任何顺序接受struct virtio\_balloon\_stat条目，而不考虑标记值。

###### 5.5.6.3.3传统接口：内存统计

使用传统接口时，过渡设备和驱动程序必须根据guest的native endian来格式化struct virtio\_balloon\_stat中的字段，而不是（不使用传统接口时little-endian。

使用传统接口时，设备应忽略驱动程序在设备初始化后提供的statsq中第一个缓冲区中的所有值。

**注意：**历史上，驱动程序在第一个缓冲区中提供了一个未初始化的缓冲区。

##### 5.5.6.4内存统计标签

**VIRTIO\_BALLOON\_S\_SWAP\_IN（0）**已交换的内存量（以字节为单位）。 **VIRTIO\_BALLOON\_S\_SWAP\_OUT（1）**已换出磁盘的内存量（以字节为单位）。

**VIRTIO\_BALLOON\_S\_MAJFLT（2）**已发生的主要页面错误的数量。 **VIRTIO\_BALLOON\_S\_MINFLT（3）**已发生的次要页面错误的数量。 **VIRTIO\_BALLOON\_S\_MEMFREE（4）**未用于任何目的的内存量（以字节为单位）。

**VIRTIO\_BALLOON\_S\_MEMTOT（5）**可用内存总量（以字节为单位）。

### 5.6 SCSI主机设备

virtio SCSI主机设备将一个或多个虚拟逻辑单元（例如磁盘）组合在一起，并允许使用SCSI协议与它们进行通信。

virtio SCSI设备提供两种请求：

·对逻辑单元的命令请求

·与逻辑单元，目标或命令相关的任务管理功能

该设备还能够发送有关添加和删除的逻辑单元的通知。这些功能共同提供了使用虚拟队列作为传输介质的SCSI传输协议。在传输协议中，virtio驱动程序充当启动程序，而virtio SCSI主机提供一个或多个接收和处理请求的目标。

本节依赖于SAM的定义。

#### 5.6.1 设备ID

8

#### 5.6.2 虚拟队列

0 controlq

1 eventq

2…n 请求队列

#### 5.6.3 功能位

**VIRTIO\_SCSI\_F\_INOUT(0)** 单个请求可以包括设备可读和设备可写数据缓冲。

**VIRTIO\_SCSI\_F\_HOTPLUG(1)** 主机应该向SCSI总线上的LUN和目标报告热插拔和热拔插事件。访客应该处理热插拔和热拔插事件。

**VIRTIO\_SCSI\_F\_CHANGE(2)** 主机将通过VIRTIO\_SCSI\_T\_PARAM\_CHANGE事件报告对LUN参数的更改；访客**应该**对其进行处理。

**VIRTIO\_SCSI\_F\_T10\_PI(3)** T10保护信息（DIF / DIX）的扩展字段包含于SCSI请求标头中。

#### 5.6.4 设备配置布局

此配置的所有字段始终可用。

struct virtio\_scsi\_config {

le32 num\_queues;

le32 seg\_max;

le32 max\_sectors;

le32 cmd\_per\_lun;

le32 event\_info\_size;

le32 sense\_size;

le32 cdb\_size;

le16 max\_channel;

le16 max\_target;

le32 max\_lun;

};

**num\_queue** 是设备公开的请求virtqueue的总数。驱动程序**可以**只使用一个请求队列，也可以使用更多请求队列获得更好的性能。

**seg\_max** 是命令中可以包含的最大段数。双向命令可以包括seg\_max输入段和seg\_max输出段。

**max\_sectors** 是驱动程序使用的最大传输大小的提示。

**cmd\_per\_lun** 告知驱动程序它可以向一个LUN发送的最大链接命令数。

**event\_info\_size** 是设备驱动程序向eventq中的缓冲区填充的最大大小。它由设备根据协商的功能集写入。

**sense\_size** 是设备能够写入的感测数据的最大大小。默认值由设备写入，该默认值必须为96，但驱动程序可以对其进行修改。

**cdb\_size** 是驱动程序能够写入的CDB的最大大小。默认值由设备写入，该默认值必须为32，但驱动程序同样可以对其进行修改。复位设备时，它将恢复为默认值。

**max\_channel, max\_target与max\_lun** 可由驱动程序作为提示使用，用于从主机上限制扫描逻辑单元到小于或等于字段值的通道/目标/逻辑单元号。max\_channel**应该**为零。max\_target**应该**小于或等于255。max\_lun**应该**小于或等于16383。

##### 5.6.4.1 驱动要求：设备配置布局

驱动程序**不得**写入除sense\_size和cdb\_size之外的设备配置字段。

驱动程序**不得**向一个LUN发送超过cmd\_per\_lun链接的命令，并且不得向一个LUN发送超过虚拟队列大小的数量链接命令。

##### 5.6.4.2 设备要求：设备配置布局

复位时，器件**必须**将sense\_size设置为96，将cdb\_size设置为32。

##### 5.6.4.3 旧版接口：设备配置布局

使用旧版接口时，过渡设备和驱动程序必须根据访客的本机端字节序而不是（必要时不使用旧版接口）小端字节序格式化struct virtio\_scsi\_config中的字段。

#### 5.6.5 设备要求：设备初始化

在初始化时，驱动程序**应该**首先发现设备的虚拟队列。

如果驱动程序使用eventq，驱动程序**应该**在eventq中至少放置一个缓冲区。

驱动程序**可以**立即发出请求或任务管理功能。

#### 5.6.6 设备操作

设备操作包括操作请求队列，控制队列和事件队列。

###### 5.6.6.0.1 旧版接口：设备操作

使用旧版接口时，驱动程序应该忽略已用环项中的len值。

**注：**从历史上看，即使实际只写入了部分缓冲区，设备仍将总描述符长度或设备可写缓冲区的总长度放在那里。

##### 5.6.6.1 设备操作：请求队列

驱动程序将请求排队到任意请求队列，并且设备在同一队列上使用它们。驱动程序有责任确保对放置在不同队列上的命令进行严格的请求排序，因为它们将在没有顺序约束的情况下被使用。

请求具有以下格式：

struct virtio\_scsi\_req\_cmd {

// Device-readable part

u8 lun[8];

le64 id;

u8 task\_attr;

u8 prio;

u8 crn;

u8 cdb[cdb\_size];

// The next two fields are only present if VIRTIO\_SCSI\_F\_T10\_PI

// is negotiated.

le32 pi\_bytesout;

le32 pi\_bytesin;

u8 pi\_out[pi\_bytesout];

u8 dataout[];

// Device-writable part

le32 sense\_len;

le32 residual;

le16 status\_qualifier;

u8 status;

u8 response;

u8 sense[sense\_size];

// The next two fields are only present if VIRTIO\_SCSI\_F\_T10\_PI

// is negotiated

u8 pi\_in[pi\_bytesin];

u8 datain[];

};

/\* command-specific response values \*/

#define VIRTIO\_SCSI\_S\_OK 0

#define VIRTIO\_SCSI\_S\_OVERRUN 1

#define VIRTIO\_SCSI\_S\_ABORTED 2

#define VIRTIO\_SCSI\_S\_BAD\_TARGET 3

#define VIRTIO\_SCSI\_S\_RESET 4

#define VIRTIO\_SCSI\_S\_BUSY 5

#define VIRTIO\_SCSI\_S\_TRANSPORT\_FAILURE 6

#define VIRTIO\_SCSI\_S\_TARGET\_FAILURE 7

#define VIRTIO\_SCSI\_S\_NEXUS\_FAILURE 8

#define VIRTIO\_SCSI\_S\_FAILURE 9

/\* task\_attr \*/

#define VIRTIO\_SCSI\_S\_SIMPLE 0

#define VIRTIO\_SCSI\_S\_ORDERED 1

#define VIRTIO\_SCSI\_S\_HEAD 2

#define VIRTIO\_SCSI\_S\_ACA 3

lun对REPORT LUNS已知的的逻辑单元，或virtio-scsi设备的SCSI域中的目标和逻辑单元进行寻址。当用于寻址REPORT LUNS逻辑单元时，lun为0xC1,0x01与6个零字节。

virtio-scsi设备应该实现REPORT LUNS已知的逻辑单元。

当用于寻址目标和逻辑单元时，lun唯一支持的格式是：第一个字节设置为1，第二个字节设置为目标，第三个和第四个字节表示单级LUN结构，后跟四个零字节。通过这种表示，virtio-scsi设备可以为每个目标提供多达256个目标和16384个LUN。该设备还可以支持在第三和第四字节中已知的逻辑单元。

id为命令标识符（“tag”）。

task\_attr定义了上表中的任务属性，但所有任务属性都**可能**由设备映射到SIMPLE。某些命令由SCSI标准定义为“隐式队列头”；对于此类命令，所有任务属性也**可以**映射到**队列头**。如果命令具有隐式**队列头**属性，则驱动程序和应用程序不应发送带有**已排序的**任务属性的命令，因为**已排序的**任务属性是否支持是特定于供应商的。

crn也可以由客户端提供，但通常预期为0。 由于CRN存储在8位整型中，协议定义的最大CRN值为255。

CDB包含在cdb中，其大小cdb\_size取自配置空间。

所有这些字段都在SAM中定义，并且始终是设备可读的。

pi\_bytesout以字节为单位确定pi\_out字段的大小。如果它非零，则pi\_out字段包含用于写操作的传出保护信息。pi\_bytesin用于确定设备可写部分中pi\_in字段以字节为单位的大小。三个字段只在VIRTIO\_SCSI\_F\_T10\_PI被协商时才存在。

除了设备可读部分的其余部分是数据输出缓冲器dataout。

sense和后续字段始终是设备可写的。sense\_len表示实际写入sense缓冲区的字节数。

residual表示读取或写入操作的残差大小，计算为“data\_length- number\_of\_transferred\_bytes”。 对于双向命令，number\_of\_transferred\_bytes包括读取和写入字节。残差小于datain意味着dataout被完整处理。残差超过datain的大小的意味着dataout被部分处理，且datain完全没有被处理。

如果pi\_bytesin非零，则pi\_in字段包含用于读取操作的传入保护信息。pi\_in只有在协商了VIRTIO\_SCSI\_F\_T10\_PI时才会出现。

除了设备可写部分的其余部分是数据输入缓冲区datain。

###### 5.6.6.1.1 设备要求：设备操作：请求队列

设备**必须**将状态字节写为SAM中定义的状态代码。

设备必须将响应字节写为以下之一：

**VIRTIO\_SCSI\_S\_OK** 当请求完成并且状态字节填充了SCSI状态代码（不一定是“GOOD”）时。

**VIRTIO\_SCSI\_S\_OVERRUN** 如果CDB的内容（例如分配长度，参数长度或传输大小）需要的数据多于数据和数据输出缓冲区中可用的数据。

**VIRTIO\_SCSI\_S\_ABORTED** 如果由于ABORT TASK或ABORT TASK SET任务管理功能而取消了请求。

**VIRTIO\_SCSI\_S\_BAD\_TARGET** 如果因为lun指示的目标不存在，请求从未被处理。

**VIRTIO\_SCSI\_S\_RESET** 如果请求因总线或设备复位（包括任务管理功能）而被取消。

**VIRTIO\_SCSI\_S\_TRANSPORT\_FAILURE** 如果请求由于主机和目标之间的连接问题而失败（断开链接）。

**VIRTIO\_SCSI\_S\_TARGET\_FAILURE** 如果目标出现故障并告知驱动程序不要在其他路径上重试。

**VIRTIO\_SCSI\_S\_NEXUS\_FAILURE** 如果nexus出现故障但在其他路径上重试可能会产生不同的结果。

**VIRTIO\_SCSI\_S\_BUSY** 如果请求失败但在同一路径上重试可能会起作用。

**VIRTIO\_SCSI\_S\_FAILURE** 其他主机或驱动程序错误。特别是，如果dataout和datain都不为空，且尚未协商VIRTIO\_SCSI\_F\_INOUT功能，则将立即返回请求，其响应等于VIRTIO\_SCSI\_S\_FAILURE。

所有命令**必须**在复位或拔出virtio-scsi设备之前完成。设备可以选择中止它们，如果不这样做，**必须**选择VIRTIO\_SCSI\_S\_FAILURE响应。

###### 5.6.6.1.2 驱动要求：设备操作：请求队列

task\_attr，prio和crn**应该**为零。

收到VIRTIO\_SCSI\_S\_TARGET\_FAILURE响应后，驱动程序**不应**在其他路径上重试该请求。

###### 5.6.6.1.3 旧版接口：设备操作：请求队列

使用旧版接口时，过渡设备和驱动程序必须根据访客的本机端字节序而不是（必要时不使用旧版接口）小端字节序格式化struct virtio\_scsi\_req\_cmd中的字段。

##### 5.6.6.2 设备操作：controlq

controlq用于其他SCSI传输操作。请求具有以下格式：

struct virtio\_scsi\_ctrl {

le32 type;

. . .

u8 response;

};

/\* response values valid for all commands \*/

#define VIRTIO\_SCSI\_S\_OK 0

#define VIRTIO\_SCSI\_S\_BAD\_TARGET 3

#define VIRTIO\_SCSI\_S\_BUSY 5

#define VIRTIO\_SCSI\_S\_TRANSPORT\_FAILURE 6

#define VIRTIO\_SCSI\_S\_TARGET\_FAILURE 7

#define VIRTIO\_SCSI\_S\_NEXUS\_FAILURE 8

#define VIRTIO\_SCSI\_S\_FAILURE 9

#define VIRTIO\_SCSI\_S\_INCORRECT\_LUN 12

该类型标识剩余的字段。

定义了以下命令：

·任务管理功能。

#define VIRTIO\_SCSI\_T\_TMF 0

#define VIRTIO\_SCSI\_T\_TMF\_ABORT\_TASK 0

#define VIRTIO\_SCSI\_T\_TMF\_ABORT\_TASK\_SET 1

#define VIRTIO\_SCSI\_T\_TMF\_CLEAR\_ACA 2

#define VIRTIO\_SCSI\_T\_TMF\_CLEAR\_TASK\_SET 3

#define VIRTIO\_SCSI\_T\_TMF\_I\_T\_NEXUS\_RESET 4

#define VIRTIO\_SCSI\_T\_TMF\_LOGICAL\_UNIT\_RESET 5

#define VIRTIO\_SCSI\_T\_TMF\_QUERY\_TASK 6

#define VIRTIO\_SCSI\_T\_TMF\_QUERY\_TASK\_SET 7

struct virtio\_scsi\_ctrl\_tmf

{

// Device-readable part

le32 type;

le32 subtype;

u8 lun[8];

le64 id;

// Device-writable part

u8 response;

}

/\* command-specific response values \*/

#define VIRTIO\_SCSI\_S\_FUNCTION\_COMPLETE 0

#define VIRTIO\_SCSI\_S\_FUNCTION\_SUCCEEDED 10

#define VIRTIO\_SCSI\_S\_FUNCTION\_REJECTED 11

类型为VIRTIO\_SCSI\_T\_TMF；子类型定义了任务管理功能。除响应之外的所有字段都由驱动程序填充。

与请求的TMF无关的其他字段将被忽略，但它们仍然存在。lun的格式与请求队列的格式相同; 当任务管理功能解决整个I\_T连接时，将忽略单级LUN相关时，id的值与requestq上传递的id值匹配。

任务管理功能的结果由设备作为响应写入。特定于命令的响应值与SAM中定义的响应值1对1映射。

任务管理功能可以影响请求队列中尚未完成的命令的响应值。例如，设备**必须**在接收到“逻辑单元复位”或“I\_T nexus reset”TMF时完成逻辑单元或目标上的所有活动命令（可能含有VIRTIO\_SCSI\_S\_RESET响应代码）。类似地，设备**必须**在接收到“中止任务”或“中止任务集”TMF时完成所选命令（可能具有VIRTIO\_SCSI\_S\_ABORTED响应代码）。这种效果必须在TMF本身成功完成之前发生，并且设备必须适当地使用内存屏障，以确保驱动程序以正确的顺序看到这些写入。

·异步通知查询。

#define VIRTIO\_SCSI\_T\_AN\_QUERY 1

struct virtio\_scsi\_ctrl\_an {

// Device-readable part

le32 type;

u8 lun[8];

le32 event\_requested;

// Device-writable part

le32 event\_actual;

u8 response;

}

#define VIRTIO\_SCSI\_EVT\_ASYNC\_OPERATIONAL\_CHANGE 2

#define VIRTIO\_SCSI\_EVT\_ASYNC\_POWER\_MGMT 4

#define VIRTIO\_SCSI\_EVT\_ASYNC\_EXTERNAL\_REQUEST 8

#define VIRTIO\_SCSI\_EVT\_ASYNC\_MEDIA\_CHANGE 16

#define VIRTIO\_SCSI\_EVT\_ASYNC\_MULTI\_HOST 32

#define VIRTIO\_SCSI\_EVT\_ASYNC\_DEVICE\_BUSY 64

通过发送此命令，驱动程序会询问设备给定LUN可以报告哪些事件，如SCSI MMC的第6.6和A.6段所述。驱动程序将其感兴趣的事件写入event\_requested；设备通过将其支持的事件写入event\_actual来进行响应。

类型为VIRTIO\_SCSI\_T\_AN\_QUERY。lun和event\_requested由驱动程序写入。event\_actual和响应字段由设备写入。

没有为响应字节定义特定于命令的值。

·异步通知订阅

#define VIRTIO\_SCSI\_T\_AN\_SUBSCRIBE 2

struct virtio\_scsi\_ctrl\_an {

// Device-readable part

le32 type;

u8 lun[8];

le32 event\_requested;

// Device-writable part

le32 event\_actual;

u8 response;

}

通过发送此命令，驱动程序会要求指定的LUN报告其物理接口的事件，再次如SCSI MMC中所述。驱动程序将其感兴趣的事件写入event\_requested；设备通过将其支持的事件写入event\_actual来进行响应。

事件类型与异步通知查询消息相同。

类型为VIRTIO\_SCSI\_T\_AN\_SUBSCRIBE。lun和event\_requested由驱动程序写入。event\_actual和response由设备写入。

没有为响应字节定义特定于命令的值。

###### 5.6.6.2.1 旧版接口：设备操作：controlq

当使用旧版接口时，过渡设备以及驱动程序**必须**根据访客的本机端字节序而不是（必要时不使用旧版接口）小端字节序格式化struct virtio\_scsi\_ctrl，struct virtio\_scsi\_ctrl\_tmf，struct virtio\_scsi\_ctrl\_an和struct virtio\_scsi\_ctrl\_an中的字段。

##### 5.6.6.3 设备操作：eventq

eventq由驱动程序填充，用于设备报告有关它附加的逻辑单元的信息。通常，设备不会通过对事件进行排列来处理空eventq，且如果没有找到准备就绪的缓冲区，就会以丢弃事件结束。然而，在报告许多LUN事件时（例如，当整个目标消失时），设备可以通过扼住事件以免丢弃它们。因此，在事件队列上放置10-15个缓冲区就足矣。

设备从eventq上返回的缓冲区在本节的其余部分将被称为“事件”。 事件具有以下格式：

#define VIRTIO\_SCSI\_T\_EVENTS\_MISSED 0x80000000

struct virtio\_scsi\_event {

// Device-writable part

le32 event;

u8 lun[8];

le32 reason;

}

设备在事件中通过置位第31位来报告由于丢失缓冲区而丢失的事件。

REASON的含义取决于事件的内容。以下事件已被定义：

·无事件

#define VIRTIO\_SCSI\_T\_NO\_EVENT 0

此事件会因以下情况被触发：

- 当设备在事件中检测的缓冲区到比配置字段中指示的缓冲区短时，它**可以**立即使用它并将该虚拟值置于事件中。设计优秀的驱动程序永远不会检测到这种情况的发生。

- 当事件被丢弃时，一旦驱动程序使一个缓冲区可用，设备就**可以**发送该事件的信号，用于向驱动程序请求操作。当然，在这种情况下，将会使用VIRTIO\_SCSI\_T\_EVENTS\_MISSED标志对此事件进行报告。

·传输复位

#define VIRTIO\_SCSI\_T\_TRANSPORT\_RESET 1

#define VIRTIO\_SCSI\_EVT\_RESET\_HARD 0

#define VIRTIO\_SCSI\_EVT\_RESET\_RESCAN 1

#define VIRTIO\_SCSI\_EVT\_RESET\_REMOVED 2

设备通过发送该事件，以信号通知目标上的逻辑单元已被复位，包括新的设备出现或消失在总线上的情况。此设备填充所有字段。event被设置为VIRTIO\_SCSI\_T\_TRANSPORT\_RESET。lun定义SCSI主机中的逻辑单元。

reason的值为上面出现的三个#define的值之一：

**VIRTIO\_SCSI\_EVT\_RESET\_REMOVED** 如果目标或逻辑单元不在能够接受命令，则(“LUN/target removed”)将会被使用。

**VIRTIO\_SCSI\_EVT\_RESET\_HARD** 如果逻辑单元被复位但仍然存在，则(“LUN hard reset”)将会被使用。

**VIRTIO\_SCSI\_EVT\_RESET\_REASCAN** 如果目标或者逻辑单元刚刚在设备上出现，则(rescan LUN/target)将会被使用。

当协商VIRTIO\_SCSI\_F\_HOTPLUG功能时，可能会发生“已删除”和“重新扫描”事件；当它们被发送到LUN 0时，它们可以应用于整个目标，因此驱动程序可以要求启动器重新扫描目标以检测它。

事件也将通过感知代码进行报告（这显然不适用于新出现的总线或目标，因为应用程序从未发现它们）：

- “LUN/target removed”映射到sense key ILLEGAL REQUEST，asc 0x25，ascq 0x00（不支持逻辑单元）。

- “LUN hard reset”映射到sense key ATTENTION, asc 0x29（发生上电，复位或总线设备复位）。

- “rescan LUN/target” 映射到sense key UNIT ATTENTION，asc 0x3f，ascq 0x0e（报告的LUN数据已被更改）。

检测传输重置的首选方法始终是使用事件进行检测，因为感知代码仅在驱动程序向逻辑单元或目标发送SCSI命令时才能被看到。但是，当事件被丢弃，如果驱动程序要求启动器重新扫描SCSI总线，启动器仍然能够与控制器的实际状态同步。在重新扫描期间，启动器将能够观察上述感测代码，并且它将像处理器已经接收到等效事件一样处理它们。

·异步通知

#define VIRTIO\_SCSI\_T\_ASYNC\_NOTIFY 2

通过发送此事件，设备会发送信号提示异步时间从物理接口被发出。

所有字段均由设备写入。event被设置为VIRTIO\_SCSI\_T\_ASYNC\_NOTIFY。lun定义SCSI主机中的逻辑单元。reason是驱动程序通过“异步通知订阅”命令订阅的事件的子集。

·LUN参数改变

#define VIRTIO\_SCSI\_T\_PARAM\_CHANGE 3

通过发送此事件，设备会发送信号提示逻辑单元的配置参数发生变化，例如容量或高速缓存模式。event设置为VIRTIO\_SCSI\_T\_PARAM\_CHANGE。lun定义SCSI主机中的逻辑单元。

同样的事件也**应该**被报告为单元注意条件。reason包含附加的感知代码和附加的感知代码限定符，分别位于0..7和8..15。

**注：**例如容量变化将以asc 0x2a, ascq 0x09(容量数据已发生变化)进行报告。

对于MMC设备（查询类型5），此事件与异步通知事件之间会有一部分重叠，因此为简单起见，主机永远不会为MMC设备报告此事件。

###### 5.6.6.3.1 驱动要求：设备操作：eventq

驱动程序**应该**用缓冲区填满eventq。这些缓冲区必须是设备可写的，并且应该至少为event\_info\_size字节长，并且其大小必须至少为struct virtio\_scsi\_event。

如果时间使第31位被设置，则驱动程序应该通过轮询逻辑单元来获取单元注意条件，与/或使用任何形式的适用于访客操作系统的总线扫描，并**应该**使用SCSI命令手动轮询异步时间。

当由于对LUN0设置VIRTIO\_SCSI\_EVT\_RESET\_REMOVED或VIRTIO\_SCSI\_EVT\_RESET\_RESCAN而收到VIRTIO\_SCSI\_T\_TRANSPORT\_RESET消息时，驱动程序**应该**要求启动器重新扫描目标，用于检测整个目标出现或者消失的情况。

###### 5.6.6.3.2 设备要求：设备操作：eventq

由于丢失缓冲区而丢失事件，则设备**必须**在事件中设置第31位，且它**可以**使用VIRTIO\_SCSI\_T\_NO\_EVENT事件来报告此情况。

除非协商了VIRTIO\_SCSI\_F\_HOTPLUG，否则设备**不得**发送原因设置为VIRTIO\_SCSI\_EVT\_RESET\_REMOVED或VIRTIO\_SCSI\_EVT\_RESET\_RESCAN的VIRTIO\_SCSI\_T\_TRANSPORT\_RESET消息。

设备**不得**向MMC设备报告VIRTIO\_SCSI\_T\_PARAM\_CHANGE。

###### 5.6.6.3.3 旧版接口：设备操作：eventq

使用旧版接口时，过渡设备和驱动程序必须根据访客的本机端字节序格式化struct virtio\_scsi\_event中的字段，而不是（必要时不使用旧版接口）小端字节序。

##### 5.6.6.4 旧版接口：框架要求

当使用旧版接口时，使用传统接口时，尚未协商VIRTIO\_F\_ANY\_LAYOUT的过渡驱动程序必须对lun，id，task\_attr，prio，crn和cdb字段使用单个描述符，并且必须对sense\_len，residual，status\_qualifier，status，response以及sense fields只使用单个描述符。

## 6 保留的功能位

目前，定义了三个与设备无关的功能位：

**VIRTIO\_F\_RING\_INDIRECT\_DESC（28）**协商此功能表明驱动程序可以使用设置了VIRTQ\_DESC\_F\_INDIRECT标志的描述符，如2.4.5.3间接描述符中所述。

**VIRTIO\_F\_RING\_EVENT\_IDX（29）**此功能启用used\_event和avail\_event字段，如2.4.7和2.4.8中所述。

**VIRTIO\_F\_VERSION\_1（32）**这表示符合此规范，提供了一种检测旧设备或驱动程序的简单方法。

### 6.1驱动程序要求：保留功能位

如果提供VIRTIO\_F\_VERSION\_1功能位，驱动程序必须（MUST）接受VIRTIO\_F\_VERSION\_1。

如果未提供VIRTIO\_F\_VERSION\_1，则驱动程序可能（MAY）无法进一步操作。

### 6.2设备要求：保留功能位

设备必须（MUST）提供VIRTIO\_F\_VERSION\_1功能位。 如果不接受VIRTIO\_F\_VERSION\_1，设备可能无法进一步运行

### 6.3传统接口：保留功能位

过渡设备可提供以下内容：

**VIRTIO\_F\_NOTIFY\_ON\_EMPTY（24）**如果驱动程序已协商此功能，则设备必须（MUST）发出中断，如果设备用尽了virtqueue上的可用描述符，即使使用VIRTQ\_AVAIL\_F\_NO\_INTERRUPT flag或中***used\_event***字段抑制了中断。

**注意：**使用此功能的驱动程序的一个示例是传统网络驱动程序：它不需要知道每次传输数据包，但它的确需要在传输后的有限时间内释放传输的数据包。 如果设备在发送完所有数据包时中断它，则可以避免使用定时器。

过渡设备必须被提供，如果设备被提供，过渡驱动程序必须接受以下内容：

**VIRTIO\_F\_ANY\_LAYOUT（27）**此功能表示设备接受任意描述符布局，如第2.4.4.3节“传统接口：消息帧”中所述。

**UNUSED（30）**qemu的实现是使用位30来检查未正确执行特征协商的virtio的实验早期版本，并且不应该被协商。

## 7 一致性

本章列出了每个章节的一致性目标和条款; 这也构成了一个有用的清单，要求作者参考他们的实施！

### 7.1一致性目标

一致性目标：

**驱动程序** 驱动程序必须符合三个一致性条款：

• 条款 [7.2](#_7.2_驱动程序的一致性)。

• 条款 [7.2.1](#_7.2.1_PCI驱动程序一致性), [7.2.2](#_7.2.2_MMIO驱动程序一致性) or [7.2.3](#_7.2.3通道I_/_O驱动程序一致性)之一。

• 条款[7.2.4](#_7.2.4网络驱动程序一致性), [7.2.5](#_7.2.5块驱动程序一致性), [7.2.6](#_7.2.6控制台驱动程序一致性), [7.2.7](#_7.2.7熵驱动程序一致性), [7.2.8](#_7.2.8传统内存气球驱动程序一致性) or [7.2.9](#_7.2.9_SCSI主机驱动程序一致性)之一。

**设备** 设备必须符合三个一致性条款：

• 条款[7.3](#_7.3设备一致性),

• 条款 [7.3.1](#_7.3.1_PCI设备一致性), [7.3.2](#_7.3.2_MMIO设备一致性) or [7.3.3](#_7.3.3通道I_/_O设备一致性)之一。

• 条款 [7.3.4](#_7.3.4网络设备一致性), [7.3.5](#_7.3.5块设备一致性), [7.3.6](#_7.3.6控制台设备一致性), [7.3.7](#_7.3.7熵设备一致性), [7.3.8](#_7.3.8传统的内存气球设备一致性) or [7.3.9](#_7.3.9_SCSI主机设备一致性)之一。

### 7.2 驱动程序的一致性

• [2.1.1](#_2.1.1_驱动要求：驱动状态字段)

• [2.2.1](#_2.2.1_驱动要求：功能位)

• [2.3.1](#_2.3.1_驱动要求：设备配置空间)

• [2.4.1](#_2.4.1_驱动要求：虚拟队列)

• [2.4.4.2](#_2.4.4.2_驱动要求：信息结构)

• [2.4.5.2](#_2.4.5.2_驱动要求：虚拟队列描述符表)

• [2.4.5.3.1](#_2.4.5.3.1)

• [2.4.7.1](#_2.4.7.1_驱动要求：虚拟队列中断抑制)

• [2.4.8.3](#_2.4.8.3_驱动要求：虚拟队列已用环)

• [2.4.9.1](#_2.4.9.1_驱动要求：虚拟队列通知抑制)

• [3.1.1](#_3.1.1_驱动程序要求:设备初始化)

• [3.2.1.3.1](#_3.2.1.3.1驱动程序要求：更新idx)

• [3.2.1.4.1](#_3.2.1.4.1驱动程序要求：通知设备)

• [3.3.1](#_3.3.1驱动程序要求：设备清理)

• [6.1](#_6.1驱动程序要求：保留功能位)

#### 7.2.1 PCI驱动程序一致性

PCI驱动程序必须符合以下规范性声明：

• [4.1.2.2](#_4.1.2.2_驱动要求：发现PCI设备)

• [4.1.3.1](#_4.1.3.1_驱动要求：PCI设备布局)

• [4.1.4.1](#_4.1.4.1_驱动要求：Virtio结构的PCI功能)

• [4.1.4.3.2](#_4.1.4.3.2_设备要求：常规配置结构布局)

• [4.1.4.5.2](#_4.1.4.5.2_驱动要求：ISR状态功能)

• [4.1.4.7.2](#_4.1.4.7.2_驱动要求：PCI配置访问功能)

• [4.1.5.1.2.2](#_4.1.5.1.2.2_驱动要求：MSI-X向量配置)

• [4.1.5.4.2](#_4.1.5.4.2_驱动要求：关于设备配置更改的注意事项)

#### 7.2.2 MMIO驱动程序一致性

MMIO驱动程序必须符合以下规范性声明：

• [4.2.2.2](#_4.2.2.2_驱动要求：MMIO设备寄存器布局)

• [4.2.3.1.1](#_4.2.3.1.1驱动要求：设备初始化)

• [4.2.3.4.1](#_4.2.3.4.1_驱动要求：来自设备的通知)

#### 7.2.3通道I / O驱动程序一致性

通道I / O驱动程序必须符合以下规范性声明：

• [4.3.1.2](#_4.3.1.2_设备要求：基本概念)

• [4.3.2.1.2](#_4.3.2.1_驱动要求：设置Virtio修订版)

• [4.3.2.3.1](#_4.3.2.3.1_驱动要求：传输状态信息)

• [4.3.3.1.2.2](#_4.3.3.1.2.2_驱动要求：关于通过Adapter_I/O中断提)

• [4.3.3.2.2](#_4.3.3.2.2_驱动要求：访客->主机通知)

#### 7.2.4网络驱动程序一致性

网络驱动程序必须符合以下规范性声明：

• [5.1.4.2](#_5.1.4.2驱动程序要求：设备配置布局)

• [5.1.6.2.1](#_5.1.6.2.1驱动程序要求：数据包传输)

• [5.1.6.3.1](#_5.1.6.3.1驱动程序要求：设置接收缓冲区)

• [5.1.6.4.2](#_5.1.6.4.2驱动程序要求：传入数据包的处理)

• [5.1.6.5.1.2](#_5.1.6.5.1.2驱动程序要求：数据包接收过滤)

• [5.1.6.5.2.2](#_5.1.6.5.2.2驱动程序要求：设置MAC地址过滤)

• [5.1.6.5.4.1](#_5.1.6.5.4.1驱动程序要求：无偿数据包发送)

• [5.1.6.5.5.1](#_5.1.6.5.5.1驱动程序要求：多队列模式下自动接收转向)

• [5.1.6.5.6.2](#_5.1.6.5.6.2驱动程序要求：设置卸载状态)

#### 7.2.5块驱动程序一致性

块驱动程序必须符合以下规范性声明：

• [5.2.5.1](#_5.2.5.1_驱动要求：设备初始化)

• [5.2.6.1](#_5.2.6.1_驱动要求：设备操作)

#### 7.2.6控制台驱动程序一致性

控制台驱动程序必须符合以下规范性声明：

• [5.3.6.1](#_5.3.6.1驱动程序要求：设备操作)

• [5.3.6.2.2](#_5.3.6.2.2驱动程序要求：多端口设备操作)

#### 7.2.7熵驱动程序一致性

熵驱动程序必须符合以下规范性声明：

• [5.4.6.1](#_5.4.6.1驱动程序要求：设备操作)

#### 7.2.8传统内存气球驱动程序一致性

传统的内存气球驱动程序必须符合以下规范性声明：

• [5.5.3.1](#_5.5.3.1驱动程序要求：特征位)

• [5.5.6.1](#_5.5.6.1驱动程序要求：设备操作)

• [5.5.6.3.1](#_5.5.6.3.1驱动程序要求：内存统计)

#### 7.2.9 SCSI主机驱动程序一致性

SCSI主机驱动程序必须符合以下规范性声明：

• [5.6.4.1](#_5.6.4.1_驱动要求：设备配置布局)

• [5.6.6.1.2](#_5.6.6.1.2_驱动要求：设备操作：请求队列)

• [5.6.6.3.1](#_5.5.6.3.1驱动程序要求：内存统计)

### 7.3设备一致性

• [2.1.2](#_2.1.2_设备要求：设备状态字段)

• [2.2.2](#_2.2.2_设备要求：功能位)

• [2.3.2](#_2.3.2_设备要求：设备配置空间)

• [2.4.4.1](#_2.4.4.1_设备要求：信息结构)

• [2.4.5.1](#_2.4.5.1_设备要求：虚拟队列描述符表)

• [2.4.5.3.2](#_2.4.5.3.2_设备要求：间接描述符)

• [2.4.7.2](#_2.4.7.2_设备要求：虚拟队列中断抑制)

• [2.4.8.2](#_2.4.8.2_设备要求：虚拟队列已用环)

• [2.4.9.2](#_2.4.9.2_驱动要求：虚拟队列通知抑制)

• [6.2](#_6.2设备要求：保留功能位)

#### 7.3.1 PCI设备一致性

PCI设备必须符合以下规范性声明：

• [4.1.1](#_4.1.1_设备要求：PCI总线下的Virtio)

• [4.1.2.1](#_4.1.2.1_设备要求：发现PCI设备)

• [4.1.3.2](#_4.1.3.2_设备要求：PCI设备布局)

• [4.1.4.2](#_4.1.4.2_设备要求：Virtio结构的PCI功能)

• [4.1.4.3.1](#_4.1.4.3.2_设备要求：常规配置结构布局)

• [4.1.4.4.1](#_4.1.4.4.1_设备要求：通知功能)

• [4.1.4.5.1](#_4.1.4.5.1_设备要求：ISR状态功能)

• [4.1.4.6.1](#_4.1.4.6.1_设备要求：特定于设备的配置)

• [4.1.4.7.1](#_4.1.4.7.1_设备要求：PCI配置访问功能)

• [4.1.4.9.0.1](#_4.1.4.9.0.1_设备要求：具有旧版驱动程序的非过渡设备)

[• 4.1.5.1.2.1](#_4.1.5.1.2.1_设备要求：MSI-X向量配置)

• [4.1.5.3.1](#_4.1.5.3.1_设备要求：来自设备的虚拟队列中断)

• [4.1.5.4.1](#_4.1.5.4.1_设备要求：关于设备配置更改的注意事项)

#### 7.3.2 MMIO设备一致性

MMIO设备必须符合以下规范性声明：

• [4.2.2.1](#_4.2.2.1_设备要求：MMIO设备寄存器布局)

#### 7.3.3通道I / O设备一致性

通道I / O设备必须符合以下规范性声明：

• [4.3.1.1](#_4.3.1.1_设备要求：基本概念)

• [4.3.2.1.1](#_4.3.2.1.1_设备要求：设置Virtio修订版)

• [4.3.2.2.1](#_4.3.2.2.1_设备要求：配置虚拟队列)

• [4.3.2.3.2](#_4.3.2.3.2_设备要求：传输状态信息)

• [4.3.2.6.3.1](#_4.3.2.6.3.1_设备要求：设置两阶段队列指示符)

• [4.3.3.1.2.1](#_4.3.3.1.2.1_设备要求：关于通过Adapter_I/O中断提)

• [4.3.3.2.1](#_4.3.3.2.1_设备要求：访客->主机通知)

#### 7.3.4网络设备一致性

网络设备必须符合以下规范性声明：

• [5.1.4.1](#_5.1.4.1设备要求：设备配置布局)

• [5.1.6.2.2](#_5.1.6.2.2设备要求：数据包传输)

• [5.1.6.3.2](#_5.1.6.3.2设备要求：设置接收缓冲区)

• [5.1.6.4.1](#_5.1.6.4.1设备要求：传入数据包的处理)

• [5.1.6.5.1.1](#_5.1.6.5.1.1设备要求：数据包接收过滤)

• [5.1.6.5.2.1](#_5.1.6.5.2.1设备要求：设置MAC地址过滤)

• [5.1.6.5.4.2](#_5.1.6.5.4.2设备要求：无偿数据包发送)

• [5.1.6.5.5.2](#_5.1.6.5.5.2设备要求：多队列模式下的自动接收转向)

#### 7.3.5块设备一致性

块设备必须符合以下规范性声明：

• [5.2.5.2](#_5.2.5.2_设备要求：设备初始化)

• [5.2.6.2](#_5.2.6.2_设备要求：设备操作)

#### 7.3.6控制台设备一致性

控制台设备必须符合以下规范性声明：

• [5.3.5.1](#_5.3.5.1设备要求：设备初始化)

• [5.3.6.2.1](#_5.3.6.2.1设备要求：多端口设备操作)

#### 7.3.7熵设备一致性

熵设备必须符合以下规范性声明：

• [5.4.6.2](#_5.4.6.2设备要求：设备操作)

#### 7.3.8传统的内存气球设备一致性

传统的内存气球设备必须符合以下规范性陈述：

• [5.5.3.2](#_5.5.3.2设备要求：特征位)

• [5.5.6.2](#_5.5.6.2设备要求：设备操作)

• [5.5.6.3.2](#_5.5.6.3.2设备要求：内存统计)

#### 7.3.9 SCSI主机设备一致性

SCSI主机设备必须符合以下规范性声明：

• [5.6.4.2](#_5.6.4.2_设备要求：设备配置布局)

• [5.6.5](#_5.6.5_设备要求：设备初始化)

• [5.6.6.1.1](#_5.6.6.1.1_设备要求：设备操作：请求队列)

• [5.6.6.3.2](#_5.6.6.3.2_设备要求：设备操作：eventq)

### 7.4传统接口：过渡设备和过渡驱动器一致性

一致的实现必须是过渡的或非过渡，见[1.3.1](#_1.3.1传统接口：术语)。

如果非过渡实现满足所有MUST或REQUIRED，则符合本规范上面定义的级别要求。

实现可以选择实现对旧接口的OPTIONAL支持，包括支持对于传统驱动程序或设备，通过额外的符合所有必须（MUST）或必需（REQUIRED）的级别要求来用于过渡设备和驱动程序的传统接口。

过渡实现的遗留接口的要求位于命名的部分中下面列出了“传统接口（Legacy Interface）”：

•第[2.2.3](#_2.2.3_旧版接口：关于功能位的注意事项)节

•第[2.3.3](#_2.3.3_旧版接口：关于设备配置空间字节序的注意事项)节

•第[2.3.4](#_2.3.4_旧版接口：设备配置空间)节

•第[2.4.2](#_2.4.2_旧版接口：关于虚拟队列布局的注意事项)节

•第[2.4.3](#_2.4.3_旧版接口：关于虚拟队列字节序的注意事项)节

•第[2.4.4.3](#_2.4.4.3_旧版接口：信息结构)节

•第[3.1.2](#_3.1.2_旧接口：设备初始化)节

•第[4.1.2.3](#_4.1.2.3_旧版接口：关于发现PCI设备的注意事项)节

•第[4.1.4.8](#_4.1.4.8_旧版接口：关于PCI设备布局的注意事项)节

•第[4.1.5.1.1.1](#_4.1.5.1.1.1_旧版接口：关于设备布局检测的注意事项)节

•第[4.1.5.1.3.1](#_4.1.5.1.3.1_旧版接口：关于虚拟队列配置的注意事项)节

•第[4.2.4](#_4.2.4_旧版接口)节

•第[4.3.2.1.3](#_4.3.2.1.3_旧版接口：关于设置Virtio修订版的注意事项)节

•第[4.3.2.2.2](#_4.3.2.2.2_旧版接口：关于配置虚拟队列的注意事项)节

•第[4.3.3.1.3](#_4.3.3.1.3_旧版接口：关于主机->访客通知的注意事项)节

•第[4.3.2.6.4](#_4.3.2.6.4_旧版接口：关于设置两阶段指示符的注意事项)节

•第[5.1.3.2](#_5.1.3.2_旧接口：特征位)节

•第[5.1.4.3](#_5.1.4.3传统接口：设备配置布局)节

•第[5.1.6.1](#_5.1.6.1传统接口：设备操作)节

•第[5.1.6.5.2.3](#_5.1.6.5.2.3传统接口：设置MAC地址过滤)节

•第[5.1.6.5.3.1](#_5.1.6.5.3.1传统接口：VLAN过滤)节

•第[5.1.6.5.5.3](#_5.1.6.5.5.3传统接口：多队列模式下自动接收转向)节

•第[5.1.6.5.6.3](#_5.1.6.5.6.3传统接口：设置卸载状态)节

•第[5.2.3.1](#_5.2.3.1_旧版接口：功能位)节

•第[5.2.4.1](#_5.2.4.1_旧版接口：设备配置布局)节

•第[5.2.5.3](#_5.2.5.3_旧版接口：设备初始化)节

•第[5.2.6.3](#_5.2.6.3_旧版接口：设备操作)节

•第[5.3.4.1](#_5.3.4.1传统接口：设备配置布局)节

•第[5.3.6.3](#_5.3.6.3传统接口：设备操作)节

•第[5.5.3.2.0.1](#_5.5.3.2.0.1传统接口：特征位)节

•第[5.5.6.2.1](#_5.5.6.2.1传统接口：设备操作)节

•第[5.5.6.3.3](#_5.5.6.3.3传统接口：内存统计)节

•第[5.6.4.3](#_5.6.4.3_旧版接口：设备配置布局)节

•第[5.6.6.0.1](#_5.6.6.0.1_旧版接口：设备操作)节

•第[5.6.6.1.3](#_5.6.6.1.3_旧版接口：设备操作：请求队列)节

•第[5.6.6.2.1](#_5.6.6.2.1_旧版接口：设备操作：controlq)节

•第[5.6.6.3.3](#_5.6.6.3.3_旧版接口：设备操作：eventq)节

•第[6.3](#_6.3传统接口：保留功能位)节

## 附录B.创建新设备类型

创建新设备类型时需要考虑各种因素。

### B.1有多少个Virtqueues？

一个非常简单的设备可能完全通过其设备配置空间运行，但大多数设备至少需要一个可以发出请求的virtqueue。 具有输入和输出的设备（例如，这里描述的控制台和网络设备）需要两个queues：一个驱动程序填充缓冲区以接收输入，另一个驱动程序将缓冲区放置到传输输出。

### B.2什么是设备配置空间布局？

设备配置空间仅应用于初始化时间参数。 这是一种有限的资源，在驱动程序写入的字段之间没有同步，因此对于大多数用途，最好使用virtqueue来更新配置信息（网络设备执行此操作以进行过滤，否则配置空间中的表（table）可能非常大）。 请记住，驱动程序可能无法原子写入（atomically writable）超过32位宽的配置字段。 因此，触发动作的可写字段不应超过32位。

### B.3什么是设备号？

设备编号可由OASIS委员会保留：发送电子邮件至virtio-dev@lists.oasis-open.org以确保其独一无二。

同时对于实验性驱动程序，请使用65535并向后操作。

### B.4有多少MSI-X向量？ （用于PCI）

使用可选的MSI-X功能设备可以通过消除客户驱动程序读取ISR状态寄存器的需求量（这可能是一项付出严重代价的操作）来加速中断处理，减少在设备内，设备和队列之间的共享中断，以及处理来自多个设备的CPU中断。 但是，某些系统对可分配给所有设备的MSI-X向量总数施加限制（可能低至256个）。 设备和驱动程序应考虑到这一点，限制使用的向量数量，除非期望的设备会导致大量中断。 设备可以控制的数量通过限制MSI-X表的大小或在PCI配置空间中不提供MSI-X功能使用的向量。

驱动程序可以通过将事件映射到尽可能少的向量来控制这种情况，或者完全禁用MSI-X功能。

### B.5设备的改进

通过协商一个新的功能位来指出对设备配置空间、新增功能、行为更改的任何改变， 这确立了阐释[[9]](#footnote-8)并避免了未来的扩展问题。

总是一起实现的功能集群可以使用单个位，但是如果一个特征在没有其他特征的情况下有意义，则它们不应无偿地组合在一起来节省特征位。

1. 缺少页面共享意味着设备的实现（例如，管理程序或主机）需要对访客存储器的完全访问。 与不信任方（即，客户间通信）的通信需要复制。 [↑](#footnote-ref-0)
2. Linux进一步的实现将virtio传输代码与特定的virtio驱动程序分开：这些驱动程序在不同的传输过程之间共享。 [↑](#footnote-ref-1)
3. 由于实现中存在各种错误，因此此字段不能用作传输头大小的保证。 [↑](#footnote-ref-2)
4. 这种情况不是由一些较旧的硬件处理的，因此在协议中特别指出。 [↑](#footnote-ref-3)
5. 由于没有保证，如果给定的地址太多，它可以使用哈希过滤器或默默地切换到allmulti或混杂模式。 [↑](#footnote-ref-4)
6. 4由于这有很高的重要性和低带宽，因此当前的Linux实现轮询使用的缓冲区，而不是等待中断，从而大大简化了实现。 但是，对于设置了O\_NONBLOCK标志的通用串行端口，将放宽轮询限制，并在下次写入或轮询调用或端口关闭或热拔出时释放消耗的缓冲区。 [↑](#footnote-ref-5)
7. 这是历史记录，与guest page大小无关。 [↑](#footnote-ref-6)
8. 在这种情况下，收缩建议（delation advice）只是一种出于礼貌的称呼。 [↑](#footnote-ref-7)
9. 即使它确实意味着文件设计或实现错误！ [↑](#footnote-ref-8)