

## 学习 &gt; Linux

概览

基本概念

Virtio 设备操作

设备的初始化



曹 丙部

2014 年 2 月 10 日发布

相关主题

Linux Kernel 支持很多 Hypervisor，比如 KVM、Xen 和 VMware 的 VMI 等。每个 Hypervisor 都有自己的一套设备模型，设备驱动多样化的特性和优化方案，这给开发带来了沉重的负担。各个平台共有性的东西越来越多，开发者的负担也越来越重。为了减少各 Hypervisor 虚拟化设备之间的差异，从而减少开发者的负担，2006 年，Red Hat 推出了 Virtio 项目。

虚拟化主要包括处理器的虚拟化、内存的虚拟化以及 I/O 的虚拟化。在 2006 年之前，KVM 的 I/O 设备都是通过软件模拟实现的，效率很低。从 2006 年开始，KVM 的 I/O 设备开始采用 Virtio 框架进行实现，此时由 Rusty Russell 开发的 virtio 引起了开发者的注意并逐渐被 KVM 等虚拟机监控器所采用。

Virtio 使用 virtqueue 来实现其 I/O 机制，每个 virtqueue 都是一个承载大量数据的 queue。针对 vring 会有相应的描述符表格进行描述。框架如下图所示：

图 1.virtio 框架

内容

概览

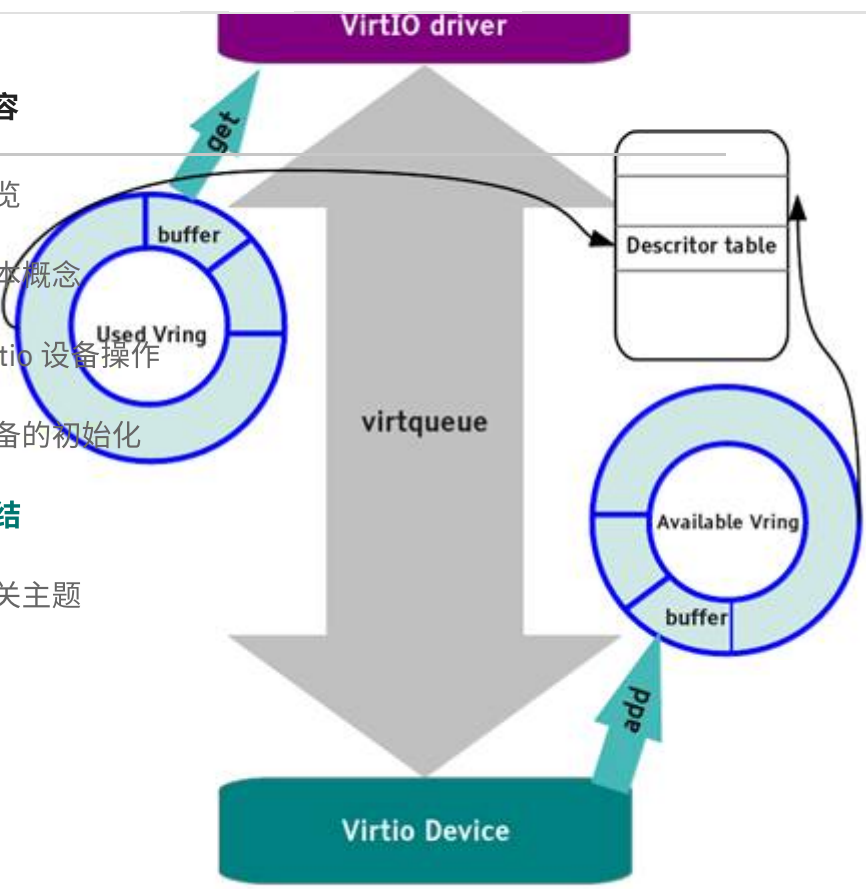
基本概念

Virtio 设备操作

设备的初始化

总结

相关主题



virtio 提供了一套有效，易维护、易开发、易扩展的中间层 API。virtio 使用 Feature Bits 来数据。使用 virtio 的设备在配置上于其他 PCI 设备没有太多不同，只不过它只应用于虚拟化

Virtio 设备具备以下特点：

1. 简单易开发

virtio PCI 设备使用通用的 PCI 的中断和 DMA 机制，对于设备驱动开发者来说不会带来困难

2. 高效

virtio PCI 设备使用针对输入和输出使用不同的 vring，规避了可能的由高速缓存带来的影响

3. 标准

virtio PCI 不假定其所处的环境一定需要对 PCI 的支持，实际上当前很多 virtio 设备已经在非需要 PCI。

因此，对与 guest OS 来说，只需要添加一个 PCI 设备驱动，然后 Hypervisor 添加设备的 virtio 内容

本文面向对 virtio 设备有一定了解的程序员，对 virtio 驱动和虚拟设备开发人员提供一定的帮助和概览

基本概念

virtio 设备操作

基本概念

设备的初始化

设备的类型

总结

相关主题

virtio 设备的 Vendor ID (厂商编号)为 0x1AF4，Device ID(设备编号)范围为 0x1000~0x103F，设备编号)如下：

表 1. virtio device ID

Subsystem Device ID	Virtio Device
1	Network card
2	Block device
3	Console
4	Entropy source
5	Memory ballooning
6	IoMemory
7	Rpmsg
8	SCSI host
9	9P transport

内容

# virtio 设备的配置空间

概览

需要使用 PCI 设备的第一块 I/O region 来对 PCI 设备进行配置。针对 virtio 设备来说，在 d 存放 virtio header(下表)。  
Virtio 设备操作

表 2.virtio 设备的配置空间  
设备的初始化

Bits	32	32	32	16	16	1
R/W	R	R+W	R+W	R	R+W	R
Purpose	Device Features	Guest Features	Queue Address	Queue Size	Queue Select	Queue Notify

最开始的 32bits 为设备的 feature bits,紧跟着的 32bits 为 guest(driver) feature bits,然后依次是 Queue Size(16bits),Queue Select(16bits),Queue Notify(16bits),Queue Status(8 bits),ISR status(8 bits)

如果设备开启了 MSI-X(Message Signalled Interrupt-Extended)，则在上述 bits 后添加了两块 32bits 的 Configuration Vector

表 3.virtio 设备的配置空间--MSI-X 开启下的附加配置

Bits	16
R/W	R+W
Purpose (MSI-X)	Configuration Vector

紧接着这些通常的 bits，可能会有指定设备专属的 headers：

表 4.virtio 设备的配置空间-设备专属配置

Bits	Device Specific
------	-----------------

## Purpose

Device Specific...

概览

各段的具体含义和作用会在后面给予解释。

基本概念

特征位操作

## 特征位(Feature Bits)

设备的初始化

使用 feature bits (32bits) 来指定设备支持的功能和特性。

总结

0~23: 根据设备类型的不同而不同

相关主题

24~32: 保留位, 用于 queue 和 feature 协商机制的扩展

有 2 组 feature bits, device 端列出支持的特性写入 device feature bits 域, guest 端把它支持的特性写入 device feature bits 域, 双方相互协商, 开始协商的唯一途径是 reset device

在设备配置 header 里面添加一个新的域通常会提供一个 device feature bit, 所以 guest 应当在读取 device feature bits 域时, 检查 device feature bit

考虑到设备的前后兼容性, 如果 device 使用新的 feature bits, guest 无法把新的 feature bits 写入 device feature bits 域, 那么 device 会进入向后兼容的模式。同样如果 guest driver 使用了 device 无法支持的 feature bits, guest 在 device feature bits 域写入的 feature bits, guest 也同样进入向后兼容的模式。

## 设备状态(Device Status)

Device Status 域主要由 guest 来更新, 表示当前 drive 的状态。状态包括:

0: 写入 0 表示重启该设备

1: Acknowledge, 表明 guest 已经发现了一个有效的 virtio 设备

2: Driver, 表明 guest 已经可以驱动该设备, guest 已经成功注册了设备驱动

3: Driver\_OK, 表示 guest 已经正确安装了驱动, 准备驱动设备

4: FAILED, 在安装驱动过程中出错

每次试图重新初始化设备前，需要设置 Device Status 为 0。

如何修改/添加/删除/更新

如果 MSI-X 没有启用的话，在 header 20 bytes 偏移的地方为设备指定的配置空间，而在 MSI-X table 的后面。

PCI 设备具有并且启用 MSI-X 中断时，在 virtio header 里会有 4 bytes 的区域用于把“配置空间”映射到 MSI-X 中断向量，通过写入 MSI-X table 入口号（有效值范围：0x0~0x7FFF）来映射中断向量。如果设备不支持 MSI-X 中断，则通过写入 0 来关闭中断取消映射。

设备的初始化

```
1 #define VIRTIO_MSI_NO_VECTOR 0xFFFF
```

读取这些寄存器返回映射到指定 event 上的 vector,如果取消映射了返回 NO\_VECTOR。默认

映射一个 event 到 vector 上需要分配资源，可能会失败。读取寄存器的值，返回 NO\_VECTOR。如果映射失败的话，会尝试映射较少的 vector 或者

在 Linux3.5.2 内核中，驱动会首先尝试为“配置修改”中断分配一个 vector,为每个 queue event 分配一个 vector。如果失败，则为所有的 queue events 映射一个共享的 vector。如果失败，则关闭 MSI-X 中断，为常规中断。

## 设备的专属配置

此配置空间包含了虚拟设备特殊的一些配置信息，可由 guest 读写。

比如网络设备含有一个 VIRTIO\_NET\_F\_MAC 特征位，表明 host 想要设备包含一个特定的 MAC 地址。设备中存有此 MAC 地址。

这种专属的配置空间和特征位的使用可以实现设备特性功能的扩展。

## virtio 设备配置的操作

针对 virtio 设备配置的操作主要包括四个部分——读写特征位，读写配置空间，读写状态位，

```
1 struct virtio_config_ops {
2     void (*get)(struct virtio_device *vdev, unsigned offset,void *buf, unsigned int *len);
3     void (*set)(struct virtio_device *vdev, unsigned offset, const void *buf, unsigned int *len);
4 }
```

```

4 |     u8 (*get_status)(struct virtio_device *vdev);
5 |     void (*set_status)(struct virtio_device *vdev, u8 status);
6 |     void (*reset)(struct virtio_device *vdev);

```

```

10 |         const char *names[];
11 |     void (*del_vqs)(struct virtio_device *);
12 |     u32 (*get_features)(struct virtio_device *vdev);
13 |     void (*finalize_features)(struct virtio_device *vdev);
14 |     const char *(*bus_name)(struct virtio_device *vdev);
15 | };

```

## 基本概念

**get():** 读取某配置域的值

Virtio 设备操作

**set():** 设置写入某配置域

设备的初始化

**status():** 读取状态位

## 相关主题

**set\_status():** 写入状态位

**reset():** 重启设备主要是清除状态位并重新配置，还有 所有的中断及其回调。当然也

**get\_features():** 读取 feature bits

**finalize\_features():** 确认最终使用的特征并写入 Guest

## Virtqueue

每个设备拥有多个 virtqueue 用于大块数据的传输。virtqueue 是一个简单的队列，guest 把一个分散-聚集数组。驱动调用 find\_vqs() 来创建一个与 queue 关联的结构体。virtqueue 的 block 设备有一个 virtqueue, network 设备有 2 个 virtqueue, 一个用于发送数据包，一个用于 virtqueue。

针对 virtqueue 的操作包括：

1)

```

1 | .int virtqueue_add_buf(
2 |     struct virtqueue *_vq,
3 |     struct scatterlist sg[],
4 |     unsigned int out,
5 |     unsigned int in,
6 |     void *data,
7 |     gfp_t gfp)

```

add\_buf()用于向 queue 中添加一个新的 buffer, 参数 data 是一个非空的令牌, 用于识别 b 会返回。

IBM Developer 学习 开发 社区

合作

Guest 通知 host 单个或者多个 buffer 已经添加到 queue 中,调用 virtqueue\_notify(), notify\_notify(VIRTIO\_PCI\_QUEUE\_NOTIFY)寄存器写入 queue index 来通知 host。

概览

3).void \*virtqueue\_get\_buf(struct virtqueue \*\_vq, unsigned int \*len)

基本概念

返回使用过的 buffer, len 为写入到 buffer 中数据的长度。获取数据, 释放 buffer,更新 vring Virtio 设备操作

4).virtqueue\_disable\_cb()

设备的初始化

**总结** guest 不再需要再知道一个 buffer 已经使用了, 也就是关闭 device 的中断。驱动会在 virtqueue\_disable\_cb()通常在这个 virtqueue 回调函数中使用, 用于关闭再次的回调发生。

相关主题

5).virtqueue\_enable\_cb()

与 disable\_cb()刚好相反, 用于重新开启设备中断的上:

## Vring

virtio\_ring 是 virtio 传输机制的实现, vring 引入 ring buffer 作为我们数据传输的载体。

virtio\_ring 包含 3 部分:

描述符数组 (descriptor table) 用于存储一些关联的描述符, 每个描述符都是一个对 buffer 的配对。

可用的 ring(available ring)用于 guest 端表示那些描述符链当前是可用的。

使用过的 ring(used ring)用于表示 Host 端表示那些描述符已经使用。

Ring 的数目必须是 2 的次幂。

## 描述符和描述符表格

vring descriptor 用于指向 guest 使用的 buffer。



addr: guest 物理地址

flags: flags 的值含义包括:

- 内容
- VRING\_DESC\_F\_NEXT: 用于表明当前 buffer 的下一个域是否有效, 也间接表明当前 b
- 概览
- VRING\_DESC\_F\_WRITE: 当前 buffer 是 read-only 还是 write-only。
- 基本概念
- VRING\_DESC\_F\_INDIRECT: 表明这个 buffer 中包含一个 buffer 描述符的 list

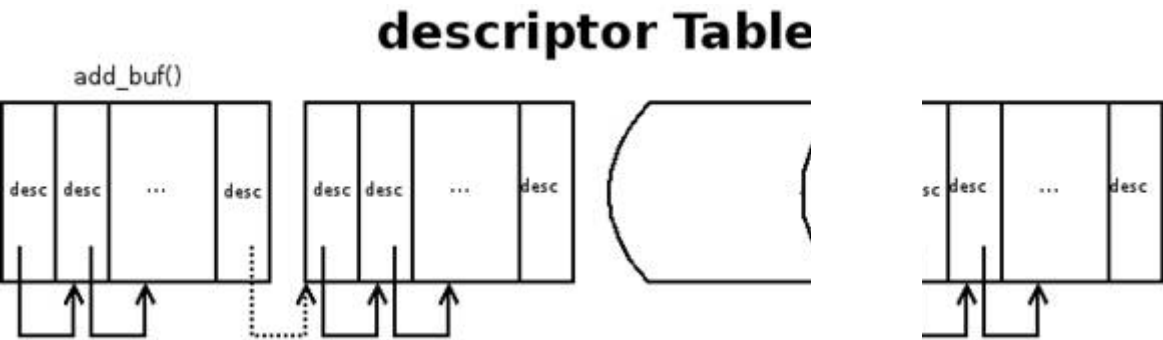
next: 所有的 buffers 通过 next 串联起来组成 descriptor table

设备的初始化  
多个 buffer 组成一个 list 由 descriptor table 指向这些 list。

总结  
约定俗成, 每个 list 中, read-only buffers 放置在 write-only buffers 前面。

相关主题

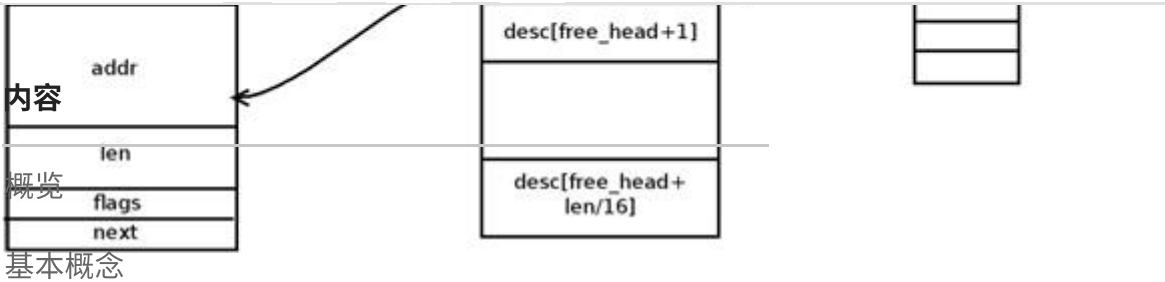
图 2.descriptor table



## Indirect Descriptors

有些设备可能需要同时完成大量数据传输的大量请求, 设备 VIRTIO\_RING\_F\_INDIRECT\_D 加 ring 的容量, vring 可以指向一个可以处于内存中任何位置 indirect descriptors table, 而 descriptors, 而这些 vring descriptor 分别指向一组 buffer list (如图所示)。当然 indirect 能再次指向 indirect descriptors table。单个 indirect descriptor table 可以包含 read-only 和 write-only flag 的 descriptor 会被忽略。

图 3.indirect decriptors

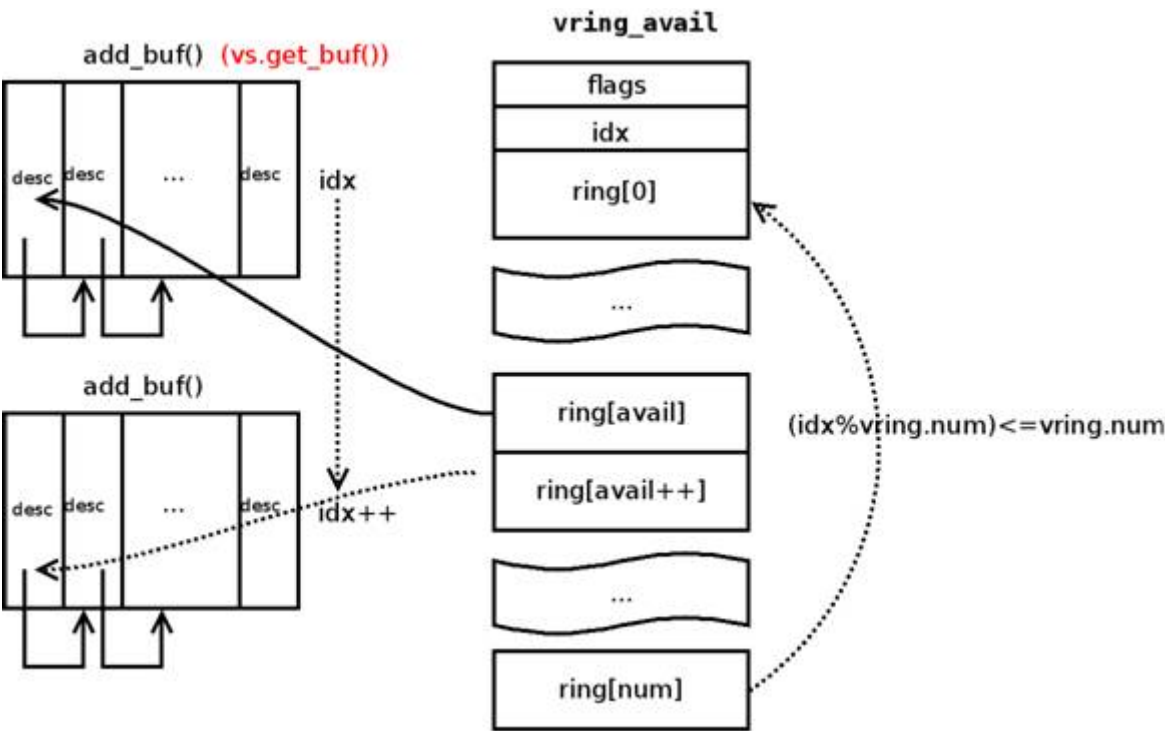


## Available Ring

设备的初始化  
Available ring 指向 guest 提供给设备的描述符,它指向一个 descriptor 链表的头。Available 值 0 或者 1，1 表明 Guest 不需要 device 使用完这些 descriptor 时上报中断。idx 指向我开始，一直增加，使用时需要取模：  
相关主题

`idx=idx&(vring.num-1)`

图 4.available ring



## Used Ring

Used ring 指向 device(host)使用过的 buffers。Used ring 和 Available ring 之间在内存中的: 和 guest 两端由于 cache 的影响而会写入到 virtqueue 结构体的同一部分的情况。

flags 用于 device 告诉 guest 再次添加 buffer 到 available ring 时不再提醒，也就是说 guest 进行 kick 操作。

len 为写入到 buffer 中的字节数。

内容

```
1 struct vring_used_elem {
2     /* Index of start of used descriptor chain. */
3     __u32 id;
4     /* Total length of the descriptor chain which was used (written to) */
5     __u32 len;
6 };
```

设备的初始化

## 总结 Virtio 设备操作

相关主题

## 设备的初始化

1. 重启设备状态，状态位写入 0
2. 设置状态为 ACKNOWLEDGE，guest(driver)端当前E |到了设备
3. 设置状态为 Driver，guest 知道如何驱动当前设备
4. 设备特定的安装和配置：特征位的协商，virtqueue 的安装，可选的 MSI-X 的安装，读写
5. 设置状态为 Driver\_OK 或者 Failed（如果中途出现错误）
6. 当前设备初始化完毕，可以进行配置和使用

## 设备的安装和配置

设备操作包括两个部分：driver(guest)提供 buffers 给设备，处理 device(host)使用过的 buf

## 初始化 virtqueue

该部分代码的实现在 virtio-pci.c 里 setup\_vps()里面，具体为：

1.选择 virtqueue 的索引，写入 Queue Select 寄存器

IBM Developer 学习 开发 社区

合作

3.分配并清零 4096 字节对齐的连续物理内存用于存放 virtqueue(调用 alloc\_pages\_exact())  
Address 寄存器(VIRTIO\_PCI\_QUEUE\_ADDR\_SHIFT)  
内容

4.可选情况下，如果 MSI-X 中断机制启用，选择一个向量用于 virtqueue 请求的中断，把对 Queue Vector 寄存器域，然后再次读取该域以确认返回正确值。

基本概念

Virtqueue 所需要的字节数由下面的公式获得：

Virtio 设备操作

```
1 | ((sizeof(struct vring_desc) * num  
2 | + sizeof(__u16) * (3 + num) + align - 1) & ~(align - 1))  
3 | + sizeof(__u16) * 3 + sizeof(struct vring_used_elem) * num
```

相关主题

其中，num 为 virtqueue 的数目。

## Guest 向设备提供 buffer

- 1.把 buffer 添加到 description table 中,填充 addr,len,f
- 2.更新 available ring head
- 3.更新 available ring 中的 index
- 4.通知 device，通过写入 virtqueue index 到 Queue Notify 寄存器

## Device 使用 buffer 并填充 used ring

device 端使用 buffer 后填充 used ring 的过程如下：

- 1.virtqueue\_pop()——从描述符表格（descriptor table）中找到 available ring 中添加的 buf
- 2.从分散-聚集的 buffer 读取数据
- 3.virtqueue\_fill()——取消内存映射,更新 ring[idx]中的 id 和 len 字段
- 4.virtqueue\_flush()——更新 vring\_used 中的 idx

5.virtio\_notify()——如果需要的话，在 ISR 状态位写入 1，通知 guest 描述符已经使用

在 MSI-X 关闭的情况下,设备端会设置 ISR bit lower bit 并发送 PCI 中断给客户机,

客户机端会读取 ISR lower bit,同时会清零,如果 lower bit 为 0,则无中断.

如果有中断,遍历一遍该设备每个 virtqueue 上的 used rings,来判断是否有中断服务需要处

理。

在 MSI-X 开启的情况下,设备端会为设备请求一个 MSI-X interrupt message 并设置 Queue entry,如果 Queue Vector 为 NO\_VECTOR,不再请求 interrupt message。

客户机端会遍历映射到该 MSI-X vector 每个 virtqueue 上的 used rings,来判断是否有中断服

务。

## Config Changed

设备端如果改变其 configure space,也存在两种情况。

客户机端会在 MSI-X 关闭的情况下,读取 ISR 高位,并设置 changed 的处理函数。在 MSI-X 开启的情况下,与中断为 MSI-X table entry。

## 总结

virtio 是 KVM 虚拟化技术中 IO 虚拟化的一个重要框架,现在有很多虚拟设备都使用了 virtio 和设备操作,可以方便读者更深一层地理解 virtio,同时也对 virtio 感兴趣的朋友能够从本文

### 相关主题

- Rusty Russell 的“[Virtio: towards a de factor standard for virtual I/O devices](#)”是深入了
- [developerWorks 云计算站点](#) 提供了有关云计算的更新资源,包括
  - 云计算 简介。
  - 更新的 技术文章和教程, 以及网络广播, 让您的开发变得轻松, 专家研讨会和录制播
  - 连接转为云计算设计的 IBM 产品下载和信息。

- 关于 [社区最新话题](#) 的活动聚合。

**IBM Developer**    学习    开发    社区

合作

**IBM Developer**

站点反馈

我要投稿

报告滥用

第三方提示

关注微博

大学合作

选择语言

English

中文

日本語

Русский

Português (Brasil)

Español

한글

Code patterns

技术文档库

软件下载

开发者中心

订阅源

时事通讯

视频

总结

相关主题