# 目 录

# 1 半虚拟化驱动

## 1.1 QEMU 模拟 I/O 设备的基本原理

模拟 I/O 设备的过程如下：

1. 客户机中的设备驱动程序发起 I/O 操作请求，KVM 模块中的 I/O 操作捕获代码会拦截这次 I/O 请求
2. I/O 操作捕获代码对 I/O 请求的信息处理后，将其放到 I/O 共享页，并通知用户控件的 QEMU 程序
3. QEMU 模拟程序获得 I/O 操作的具体信息后，交由硬件模拟代码来模拟出本次的 I/O 操作
4. 硬件模拟代码的模拟操作完成后，把结果放回到 I/O 共享页，并通知 KVM 模块的 I/O 操作捕获代码
5. 由 KVM 模块中的 I/O 操作捕获代码读取 I/O 共享页中的操作结果，并把结果返回到客户机中

## 1.2 virtio 的介绍

KVM 实现半虚拟化驱动的方式是采用 virtio 这个 Linux 上的设备驱动的那个标准框架。

virtio 由四层组成，为前端驱动层、virtio 层、transport 层和后端处理层。前端驱动层是客户机中的驱动程序模块，后端处理层是 QEMU 中的后端处理程序。而 virtio 层和 transport 层用于支持客户机和 QEMU 之间的通信。

### 1.2.1 virtio_balloon 的介绍

首先介绍一下 ballooning 技术。ballooning 技术可以在客户机运行时动态地调整它所占用的宿主机的内存资源，而不需要关闭客户机。这个技术实现了，当宿主机内存紧张时，可以请求客户机的部分内存，从而客户机就会释放其空闲内存。如果此时客户机空闲内存不足，可能还会回收部分使用中的内存。

KVM 中 ballooning 的工作过程如下：

1. KVM 发送请求到客户机操作系统，让其归还部分内存给宿主机。
2. 客户机操作系统中的 virtio_balloon 驱动接收到 KVM 的请求，然后使客户机中的内存气球膨胀，气球中的内存不能被客户机访问。

3. 客户机操作系统将气球中的内存还给 KVM，KVM 可以把气球中的内存分配到任何需要的地方。

使用如下命令即可使用 ballooning 技术：

```
-balloon virtio
// 如，qemu-system-x86_64 ubuntu1604.img -m 2048 -balloon virtio
```

可以在 qemu monitor 中查看和设置客户机内存的大小，命令如下：

```
info balloon // 查看客户机内存占用量
balloon num // 设置客户机内存占用量为numMB
```

通过如下命令，可以在客户机中看到 balloon 技术的使用，如下图所示：

```
psd@scholes:~$ lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Cirrus Logic GD 5446
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
00:04.0 Unclassified device [00ff]: Red Hat, Inc Virtio memory balloon
```

### 1.2.2 virtio_net 的介绍

选择 KVM 网络设备时，使用 virtio_net 半虚拟化驱动可以提高网络吞吐量和降低网络延迟。

通过以下命令即可将客户机的网卡设备指定为 virtio 类型：

```
-net nic,model=virtio
// 如，qemu-system-x86_64 ubuntu1604.img -m 2048 -net nic,model=virtio
```

以下命令可以将 virtio_net 的后端处理任务放到内核空间中执行，从而提高效率。如下所示：

```
-net tap,vhost=on
// 如，qemu-system-x86_64 ubuntu1604.img -m 2048 -net nic,model=virtio -net tap,
    vhost=on
```

### 1.2.3 virtio_blk 的介绍

使用 virtio_blk 半虚拟化驱动可以提高访问块设备 I/O 的方法。

使用如下命令可以启用 virtio_blk 驱动：

```
file=filename,if=virtio
// 如，qemu-system-x86_64 -m 2048 -net nic file=ubuntu1604.img,if=virtio
```

### 1.2.4 kvm_clock 的介绍

使用 kvm_clock 半虚拟化时钟，可以为客户机提供精确的 system time 和 wall time，从而避免客户机时间不准确的问题。

使用 qemu 命令启动客户机时，已经将 kvm_clock 默认作为客户机的时钟来源。可以通过如下命令查看客户机中与时钟相关的信息，如下图所示：

```
psd@scholes:~$ dmesg | grep -i clock
[    0.000000] kvm-clock: Using msrs 4b564d01 and 4b564d00
[    0.000000] kvm-clock: cpu 0, msr 0:7fff5001, primary cpu clock
[    0.000000] kvm-clock: using sched offset of 2033147586 cycles
[    0.000000] clocksource: kvm-clock: mask: 0xffffffffffffffff max_cycles: 0x1c
d42e4dffb, max_idle_ns: 881590591483 ns
[    0.000000] clocksource: refined-jiffies: mask: 0xffffffff max_cycles: 0xffff
ffff, max_idle_ns: 7645519600211568 ns
[    0.000000] clocksource: hpet: mask: 0xffffffff max_cycles: 0xffffffff, max_i
dle_ns: 19112604467 ns
[    0.000000] hpet clockevent registered
[    0.164683] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, ma
x_idle_ns: 7645041785100000 ns
[    0.168627] acpi PNP0A03:00: _OSC: OS supports [ASPM ClockPM Segments MSI]
[    0.187051] clocksource: Switched to clocksource kvm-clock
[    0.196852] clocksource: acpi_pm: mask: 0xffffff max_cycles: 0xffffff, max_id
le_ns: 2085701024 ns
[    0.611594] rtc_cmos 00:00: setting system clock to 2016-12-21 14:45:17 UTC (
1482331517)
[    1.575328] tsc: Refined TSC clocksource calibration: 2394.517 MHz
[    1.575331] clocksource: tsc: mask: 0xffffffffffffffff max_cycles: 0x2283fbcd
3b3, max_idle_ns: 440795270903 ns
```

# 2 PCI 设备直接分配

PCI 设备直接分配允许将宿主机中的物理 PCI 设备直接分配给客户机完全使用。Inte 定义的 PCI 设备直接分配技术规范称为 VT-d。

当 KVM 将宿主机的 PCI 设备附加到客户机时，客户机对该设备的 I/O 交互操作和实际的物理设备操作完全一样，不需要 KVM 的参与。

## 2.1 VT-d 环境配置

VT-d 环境配置包括以下几个方面：

1. 硬件支持和 BIOS 设置。需要在 BIOS 中将 VT-d 功能设置为“Enabled”状态。
2. 宿主机内核的配置。在配置内核时，需要配置如下几个 VT-d 相关的配置选项：

```
CONFIG_IOMMU_SUPPORT=y
CONFIG_DMAR_TABLE=y
CONFIG_INTEL_IOMMU=y
CONFIG_INTEL_IOMMU_DEFAULT_ON=y
CONFIG_IRQ_REMAP=y
CONFIG_PCI_STUB=m
```

可以通过以下两个命令查看宿主机是否支持 VT-d：

```
dmesg | grep DMAR -i
dmesg | grep IOMMU -i
```

3. 绑定设备到 pci_stub 驱动，从而对需要分配给客户机的设备进行隐藏，使得宿主机和其他客户机无法使用该设备。命令如下所示：

```
modprobe pci_stub // 加载pci_stub驱动
// 通过下一行命令得到设备的domain:bus:slot.function vendor_ID:device_ID
lspci -Dn -s BDF
// 绑定设备到pci_stub驱动
echo -n "vendor_ID device_ID" > /sys/bus/pci/drivers/pci-stub/new_id
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/domain:bus:slot.
    function/driver/unblind
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/pci_stub/blind
```

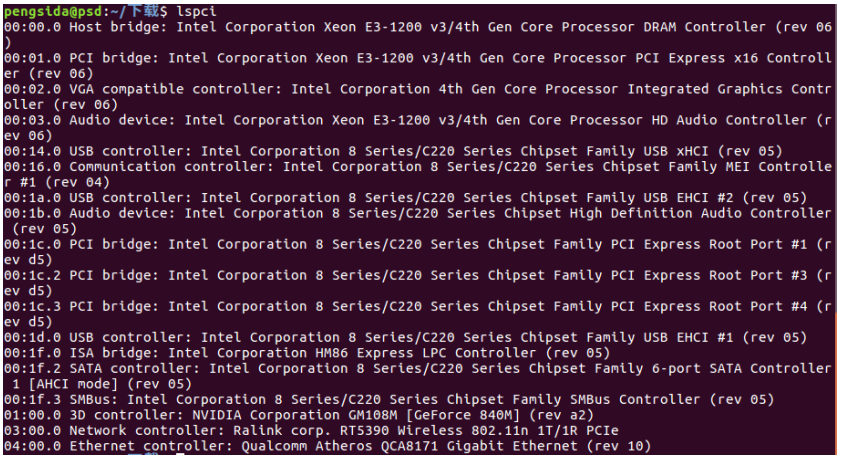
4. 使用 qemu 命令分配设备给客户机，命令如下所示：

```
-device pci-assign,host=BDF
// 如，qemu-system-x86_64 ubuntu1604.img -device pci-assign,host=08:00.0
```

5. 当客户机不需要使用该设备后，让宿主机重新使用该设备命令如下：

```
echo -n "vendor_ID device_ID" > /sys/bus/pci/drivers/domain:bus:slot.function
    /driver/new_id
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/pci_stub/unblind
echo "domain:bus:slot.function" > /sys/bus/pci/drivers/domain:bus:slot.
    function/driver/blind
```

在绑定设备到 pci_stub 驱动和使用 qemu 命令分配设备给客户机两个步骤，主要需要知道设备的 BDF。可以通过 lspci 查看电脑所有设备的 BDF，每行设备信息前面的 bus:slot.function 就是设备的 BDF。如下图所示：



## 2.2 SR-IOV 技术

SR-IOV 技术实现了多个虚拟机能够共享同一个物理设备的资源，并且达到设备直接分配的性能。SR-IOV 有两个功能，如下所示：

1. 物理功能 (PF)，放在宿主机中配置和管理虚拟功能，它本身也可以作为一个普通的 PCI-e 设备使用。
2. 虚拟功能 (VF)，轻量级 PCI-e 功能。虚拟功能通过物理功能配置后，可以分配到客户机中作为独立功能使用。

可以通过如下命令查看设备是否具备 SR-IOV 的能力：

```
lspci -v -s BDF
```

在宿主机中，当加载支持 SR-IOV 技术的 PCI 设备的驱动时，可以加上相应的参数来指定启用多少个 VF。相关命令如下所示：

```
modprobe driver max_vfs=num
```

在已知设备 domain:bus:slot.function 的情况下，可以通过以下命令查看该设备的 VF：

```
ls -l /sys/bus/pci/devices/domain:bus:slot.function/virtfn*
```

# 3 热拔插

热拔插指的是可以在电脑运行时插上或拔除硬件。在 KVM 虚拟化环境中，在不关闭客户机的情况下，也可以对客户机的设备进行热拔插。

## 3.1 PCI 设备的热拔插

PCI 设备的热拔插需要以下几个方面的支持：

1. 硬件支持。现在的 BIOS 和 PCI 总线都支持热拔插。
2. 客户机操作系统支持，内核配置文件中需要有以下配置：

```
CONFIG_HOTPLUG=y
CONFIG_HOTPLUG_PCI_PCIE=y
CONFIG_HOTPLUG_PCI=y
CONFIG_HOTPLUG_PCI_FAKE=m
CONFIG_HOTPLUG_PCI_ACPI=y
CONFIG_HOTPLUG_PCI_ACPI_IBM=m
```

可以在 qemu monitor 中完成热拔插功能，比如要将 BDF 为 02:00.0 的 PCI 设备动态添加到客户机中，在 monitor 中的命令如下：

```
device_add pci-assign,host=02:00.0,id=mydevice
```

也可以将设备从客户机中动态移除，在 monitor 中的命令如下：

```
device_del mydevice
```

需要注意的是，如果要把宿主机中的 PCI 设备给客户机作为热拔插使用，需要绑定设备到 pci_stub 驱动，从而对需要分配给客户机的设备进行隐藏，使得宿主机和其他客户机无法使用该设备。

# 4 动态迁移

## 4.1 虚拟化环境中的迁移

在虚拟化环境中的迁移分为静态迁移和动态迁移。

静态迁移有两种的实现方式:

- 一种实现方式是，关闭客户机后，将其硬盘镜像复制到另一台宿主机上然后恢复启动起来。
- 另一种实现方式是，两台宿主机共享存储系统，只需要在暂停客户机后，复制其内存镜像到另一台宿主机中恢复启动。

可以通过以下两个步骤实现静态迁移:

1. 在源宿主机上某客户机的 qemu monitor 中使用“savevm my_tag”命令来保存一个完整的客户机镜像快照。
2. 在源宿主机中关闭或暂停该客户机。
3. 将该客户机的镜像文件复制到另外一台宿主机中，在其 qemu monitor 中用“loadvm my_tag”命令来加载保存快照时的客户机快照。

动态迁移指的是在保证客户机上应用服务正常运行的同时，让客户机在不同的宿主机之间进行迁移。一个成功的动态迁移，需要保证客户机的内存、硬盘存储、网络连接在迁移到目的主机后依然保存不变，而且迁移过程的服务暂停时间较短。

## 4.2 动态迁移的应用场景

1. 负载均衡。当一台物理服务器的负载较高时，可以将其上运行的客户机动态迁移到负载较低的宿主机服务器中。
2. 解除硬件依赖。当系统管理员需要在宿主机上升级、添加或移除某些硬件设备时，可以将该宿主机上运行的客户机动态迁移到其他宿主机上。
3. 节约能源。可以将宿主机上的客户机动态迁移到几台服务器上，而某些宿主机上的客户机完全迁移走后，就可以将其关闭电源，从而省电。
4. 实现客户机地理位置上的远程迁移。

## 4.3 KVM 动态迁移原理

### 4.3.1 基于共享存储系统的动态迁移的原理

当源宿主机和目的宿主机共享存储系统时，只需要通过网络发送客户机的 vCPU 执行状态、内存中的内容和虚拟设备的状态到目的主机上。具体迁移过程如下所示：

1. 在客户机在源客户机运行的同时，将客户机的内存页传输到目的主机上。
2. KVM 会监控并记录下迁移过程中所有已经被传输的内存页的任何修改。
3. 当内存数据量传输完成时，KVM 会关闭源宿主机上的客户机，然后将剩余的数据量传输到目的主机上去。
4. 当所有内存内容传输到目的宿主机后，就可以在目的宿主机上恢复客户机的运行状态。

需要注意的是，如果目的主机上缺少一些配置，那么客户机就无法正常运行。比如，在原宿主机上有给客户机配置好网桥类型的网络，但是目的主机没有网桥配置，那么迁移后的客户机就会网络不通。

还有一种情况就是，如果内存中数据被修改的速度大于 KVM 能够传输的内存速度时，动态迁移就无法完成。

### 4.3.2 动态迁移的注意事项

- 共享存储在源宿主机和目的宿主机上的挂载位置必须完全一致。
- 为了提高动态迁移的成功率，尽量在同类型 CPU 的主机上面进行动态迁移。
- 64 位的客户机只能在 64 位宿主机之间迁移，而 32 位客户机可以在 32 位宿主机和 64 位宿主机之间迁移。
- 动态迁移的源宿主机和目的宿主机对 NX 位的设置必须相同。
- 在目的宿主机上不能有与被迁移客户机同名的客户机存在。
- 目的宿主机和源宿主机的软件配置应该尽可能相同。

### 4.3.3 实现基于共享存储系统的动态迁移

动态迁移的实现如下所示：

1. 在源宿主机挂载 NFS 上的客户机镜像，并启动客户机。命令如下所示：

```
// 挂载客户机镜像
mount my-nfs:/rw-images/ /mnt/
// 启动客户机
qemu-system-x86_64 /mnt/ubuntu1604.img -smp 2 -m 2048 -net nic -net tap
```

2. 在目的宿主机上挂载 NFS 上的客户机镜像，并启动一个客户机用于接受动态迁移过来的内存内容。需要注意的是共享存储在源宿主机和目的宿主机上的挂载位置必须完全一致。命令如下所示：

```
// 挂载客户机镜像
mount vt-nfs:/rw-images/ /mnt/
// 启动客户机
qemu-system-x86_64 /mnt/ubuntu1604.img -smp 2 -m 2048 -net nic -net tap -
    incoming tcp:0:6666
```

参数“-incoming tcp:0:6666”表示在 6666 端口建立一个 TCP Socket 连接，用于接受来自源主机的动态迁移的内容，其中“0”表示允许来自任何主机的连接。

3. 在源宿主机的客户机的 qemu monitor 中使用如下命令进入动态迁移的流程：

```
migrate tcp:vt-snb9:6666
```

“vt-snb9”是目的宿主机的主机名，tcp 协议和 6666 端口号需要与目的宿主机上 qemu-kvm 命令行的“-incoming”参数中的值保持一致。

### 4.3.4 实现使用相同后端镜像文件的动态迁移

过程如下所示：

1. 在源宿主机上，根据一个后端镜像文件，创建一个 qcow2 格式的镜像文件，并启动客户机。相应的命令如下所示：

```
// 创建一个qcow2格式的镜像文件
qemu-img create -f qcow2 -o backing_file=ubuntu1604.img,size=20G ubuntu1604.
    qcow2
// 启动客户机
qemu-system-x86_64 ubuntu1604.qcow2 -smp 2 -m 2048 -net nic -net tap
```

2. 在目的宿主机上，也建立相同的 qcow2 格式的客户机镜像，并用“-incoming”参数使得客户机处于迁移监听状态。相应的命令如下所示：

```
// 创建一个qcow2格式的镜像文件
qemu-img create -f qcow2 -o backing_file=ubuntu1604.img,size=20G ubuntu1604.
    qcow2
// 启动客户机
qemu-system-x86_64 ubuntu1604.qcow2 -smp 2 -m 2048 -net nic -net tap -
    incoming tcp:0:6666
```

3. 在源宿主机上的客户机的 qemu monitor 中，运行如下命令进行动态迁移：

```
migrate -i tcp:vt-snb9:6666
```

### 4.3.5 实现直接复制客户机磁盘镜像的动态迁移

过程如下所示：

1. 在源宿主机上启动客户机。相应的命令如下所示：

```
// 启动客户机
qemu-system-x86_64 ubuntu1604.img -smp 2 -m 2048 -net nic -net tap
```

2. 在目的宿主机上用“-incoming”参数使得客户机处于迁移监听状态。相应的命令如下所示：

```
// 启动客户机
qemu-system-x86_64 ubuntu1604.img -smp 2 -m 2048 -net nic -net tap -incoming
    tcp:0:6666
```

3. 在源宿主机上的客户机的 qemu monitor 中，运行如下命令进行动态迁移：

```
// -b代表传输块设备
migrate -b tcp:vt-snb9:6666
```

## 4.4 动态迁移的相关命令

下面完整介绍一下前几个小节使用的 migrate 命令：

| 标准格式：migrate [-d][-b][-i] uri | |
|---|---|
| 参数选项： | |
| uri | 统一资源标识符，“协议: 主机名: 端口号” |
| -d | 表示不用等待迁移完成就让 qemu monitor 处于可输入命令的状态 |
| -b | 表示传输整个磁盘镜像 |
| -i | 表示在有相同的后端镜像的情况下增量传输 qcow2 类型的磁盘镜像 |

其他命令:

| |
|---|
| migrate_cancel<br>在动态迁移进行过程中取消迁移 |
| migrate_set_speed value<br>设置动态迁移中的最大传输速度 |
| migrate_set_downtime value<br>设置允许的最大停机时间 |

# 5 嵌套虚拟化

## 5.1 嵌套虚拟化的介绍

嵌套虚拟化是指在虚拟化的客户机中运行一个 KVM，从而在虚拟化运行一个客户机。

对于 KVM 这样的必须依靠硬件虚拟化扩展的方案，必须在客户机中模拟硬件虚拟化特性的支持，并在 KVM 的操作指令进行模拟。

## 5.2 KVM 嵌套 KVM

KVM 嵌套 KVM，就是在 KVM 上面运行的第一级客户机中，再加载 kvm 和 kvm_intel 模块，然后在第一级客户机中用 qemu-kvm 启动带有 kvm 加速的第二级客户机。

KVM 嵌套 KVM 的配置有如下几个步骤：

1. 在宿主机中，加载 kvm-intel 模块时，需要添加“nested=1”的选项，从而打开嵌套虚拟化的特性。如下所示：

```
modprobe kvm_intel nested=1
```

2. 在启动第一级客户机时，应该在 qemu-kvm 命令中加上“-cpu host”或者“-cpu cpu_model,+vmx”选项，从而将 CPU 的硬件虚拟化扩展特性暴露给第一级客户机。相关的命令如下所示：

```
qemu-system-x86_64 ubuntu1604.img -m 4096 -smp 2 -net nic -net tap -cpu host
// 或者用下面的命令
qemu-system-x86_64 ubuntu1604.img -m 4096 -smp 2 -net nic -net tap -cpu cpu\
    _model,+vmx
```

“-cpu host”参数可以将宿主机的 CPU 暴露给第一级客户机，而“-cpu cpu_model,+vmx”参数以某个 CPU 模型为基础，然后加上 Intel VMX 特性。

3. 在第一级客户机中加载 kvm 和 kvm_intel 模块，然后启动第二级客户机，命令如下：

```
modprobe kvm
modprobe kvm_intel
qemu-system-x86_64 ubuntu1604.img -m 1024 -smp 2
```

因为 KVM 为第一级客户机提供了有硬件辅助虚拟化特性的透明的硬件环境，所以在第一级客户机中启动第二级客户机的操作，和宿主机启动第一级客户机的操作完全一样。

# 6 KSM 技术

## 6.1 KSM 技术的介绍

KSM 时“Kernel SamePage Merging”的缩写，中文叫做“内核同页合并”。KSM 让内核扫描正在运行中的程序并比较它们的内存，如果它们的内存页是完全相同的，就将多个相同的内存合并为一个单一的内存页，并将其标识为“写时复制”。如果有进程试图去修改被标识为“写时复制”的合并的内存页时，就为该进程复制出一个新的内存页供其使用。

如果同一宿主机上的多个客户机运行的是相同的操作系统，那么客户机之间的相同内存页数量会比较多，于是 KSM 的作用就比较大。在 KVM 虚拟化环境中，KSM 可以从以下两个方面提高内存的速度和使用效率：

1. 相同的内存页合并后，减少了客户机的内存使用量。
2. KSM 通过减少每个客户机实际占用的内存数量，就可以让多个客户机分配的内存数量之和大于物理上的内存数量。这样的话，在物理内存量不变的情况下，可以在一个宿主机中创建更多的客户机，提高了物理资源的利用效率。

## 6.2 查看系统中的 KSM 技术

内核的 KSM 守护进程是 ksmd，配置 ksmd 的文件在“/sys/kernel/mm/ksm/”目录下，可以通过以下命令查看该目录下的几个文件：

```
ls -l /sys/kernel/mm/ksm/*
```

结果如下图所示：

```
pengsida@psd:~/下载$ ls -l /sys/kernel/mm/ksm/*
-r--r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/full_scans
-rw-r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/merge_across_nodes
-r--r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/pages_shared
-r--r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/pages_sharing
-rw-r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/pages_to_scan
-r--r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/pages_unshared
-r--r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/pages_volatile
-rw-r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/run
-rw-r--r-- 1 root root 4096 12月 23 08:40 /sys/kernel/mm/ksm/sleep_millisecs
```

下面介绍一下这些文件的作用：

- full_scans: 记录着已经对所有可合并的内存区域扫描过的次数。
- pages_shared: 记录着正在使用中的共享内存页的数量。

- pages_sharing: 记录着有多少数量的内存页正在使用被合并的共享页，不包括合并的内存页本身。
- page_unshared: 记录着守护进程去检查并试图合并，却发现了并没有重复内容而不能被合并的内存页数量。
- pages_volatitle: 记录了因为其内容很容易变化而不被合并的内存页。
- pages_to_scan: 记录着在 ksmd 进程休眠之前会去扫描的内存页数量。
- sleep_millisecs: 记录着 ksmd 进程休眠的时间。
- run: 记录着控制 ksmd 进程是否运行的参数。

可以通过对 pages_to_scan、sleep_millisecs 和 run 这三个文件写入自定义的值来控制 ksmd 的运行。如下所示：

```
// 调整每次扫描的内存页数量
echo num >/sys/kernel/mm/ksm/pages_to_scan
// 设置ksmd休眠的时间
echo num >/sys/kernel/mm/ksm/sleep_millisecs
// 激活ksmd的运行
echo 1 >/sys/kernel/mm/ksm/run
```

# 7 KVM 其他特性简介

## 7.1 1GB 大页

1GB 大页通过 hugetlbfs 文件系统来利用硬件提供的大页支持。下面是在 KVM 环境中使用 1GB 大页的具体操作步骤：

1. 检查硬件和内核配置对 1GB 大页的支持，相关命令如下所示：

```
cat /proc/cpuinfo | grep pdpelgb
```

如果没有这个支持，需要在内核编译时配置，config 文件需要有以下两项配置：

```
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y
```

2. 在宿主机的内核启动参数重配置 1GB，配置文件中添加如下参数：

```
hugepagesz=size hugepages=num default_hugepagesz=default_size
// hugepagesz表示HugeTLB内存页的大小
// hugepages表示启动时大页分配的数量
// default_hugepagesz表示在挂在hugetlb文件系统时，没有设置大页的大小时默认使用的大页的大小
// hugepagesz和hugepages选项可以成对地多次使用，可以让系统在启动时同时保留多个大小不同的大页
```

如下图所示：

```
menuentry 'Ubuntu' --class ubuntu --class gnu-linux --class gnu --class os $menuentry_id_option 'gnulinux-simple-84ca4c1b-ff58-4c6d-8808-45e3dc5f4e63' {
        recordfail
        load_video
        gfxmode $linux_gfx_mode
        insmod gzio
        if [ x$grub_platform = xxen ]; then insmod xzio; insmod lzopio; fi
        insmod part_gpt
        insmod ext2
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint-bios=hd0,gpt2 --hint-efi=hd0,gpt2 --hint-baremetal=ahci0,gpt2  84ca4c1b-ff58-4c6d-8808-45e3dc5f4e63
        else
          search --no-floppy --fs-uuid --set=root 84ca4c1b-ff58-4c6d-8808-45e3dc5f4e63
        fi
        linux   /boot/vmlinuz-4.9.0-rc8+ root=UUID=84ca4c1b-ff58-4c6d-8808-45e3dc5f4e63 ro  quiet splash $vt_handoff
        hugepagesz=1GB hugepages=6 default_hugepagesz=1GB
        initrd  /boot/initrd.img-4.9.0-rc8+
}
```

3. 挂在 hugetlbfs 文件系统，命令行如下：

```
mount -t hugetlbfs hugetlbfs /dev/hugepages
```

如果使用了两种大小的大页，可以在挂在 hugetlbfs 文件系统的时候，通过“pagesize”选项来制定挂载 hugetlbfs 的大页的大小，相关命令如下所示：

```
mount -t hugetlbfs hugetlbfs /dev/hugepages -o pagesize=size
```

4. 通过“-mem-path”参数为客户机提供大页的支持，命令如下所示：

```
qemu-system-x86_64 ubuntu1604.img -mem-path /dev/hugepages/
```

需要注意的是，如果要使用大页，需要一开始就预留大页的内存。大页的内存不能被交换到交换分区，也不能通过 ballooning 方式使用大页内存。

## 7.2 透明大页

透明大页的好处有以下三点：

1. 应用程序不需要任何修改即可享受透明大页带来的好处。
2. 透明大页可以被交换到交换空间，此时透明大页会被打碎为常规的 4KB 大小的内存页。
3. 使用透明大页时，如果因为内存碎片导致大页内存分配失败，系统可以优雅地使用常规的 4KB 页替换。

使用透明大页的方式如下：

- 在编译 Linux 内核时，配置好透明大页的支持，如下所示：

```
CONFIG_TRANSPARENT_HUGEPAGE=y
// 表示默认对所有应用程序的内存分配都尽可能地使用透明大页
CONFIG_TRANSPARENT_HUGEPAGE_ALWAYS=y
```

除此之外，还可以通过修改内核启动参数来调整透明大页的使用频率。相关参数如下所示：

```
transparent_hugepage=[always|madvise|never]
```

还可以通过以下命令来配置透明大页的使用频率：

```
echo "never" > /sys/kernel/mm/transparent_hugepage/defrag
```

## 7.3 暴露宿主机 CPU 特性

在启动客户机时，可以用过“+”号来添加一个或多个 CPU 特性到一个基础的 CPU 模型上，如下所示：

```
-cpu qemu64,+vmx
```

可以通过如下命令查看宿主机 CPU 的特性：

```
cat /proc/cpuinfo
```

terminal 输出信息中的 flags 就是宿主机 CPU 的特性。

qemu-kvm 还提供了“-cpu host”参数来尽可能多地暴露宿主机 CPU 特性给客户机，从而在客户机中可以看到和使用宿主机 CPU 的各种特性。不过“-cpu host”参数可能会阻止客户机的动态迁移，所以需要根据场景使用。

# 8 QEMU 监控器

QEMU 监控器是 qemu 与用户交互的控制台。以下是 monitor 中一些常用的命令:

| | |
|---|---|
| help [cmd] | 显示命令的帮助信息。 |
| info [subcommand] | 显示 subcommand 描述的系统状态。如果 subcommand 为空，就显示当前所有的 info 命令组合及其介绍。 |
| commit | 将变化部分提交到后端镜像文件中。 |
| stop | 用于停止 qemu 模拟器 |
| cont | 恢复 qemu 模拟器继续工作 |
| change | 改变一个设备的配置 |
| balloon | 改变分配给客户机的内存大小 |
| device_add | 动态添加设备 |
| device_del | 动态移除设备 |
| usb_add | 添加一个 USB 设备 |
| usb_del | 移除一个 USB 设备 |
| savevm、loadvm devlvm | 创建、加载和删除客户机的快照 |
| migrate migrate_cancel | 动态迁移和取消动态迁移 |
| cpu index | 指定系统默认的 CPU |
| log | 将制定的项目保存到/tmp/qemu.log 中 |
| logfile filename | 将 log 文件输出到 filename 文件中 |
| sendkey keys | 像客户机发送 keys 按键，产生如同在非虚拟环境中那样的按键效果 |
| system_powerdown | 关闭客户机 |
| system_reset | 让客户机重置，相当于直接拔掉电源，然后插上电源，按开机键开机 |
| system_wakeup | 将客户机从暂停中唤醒 |
| x /fmt addr | 转存出从 addr 开始的虚拟内存地址，fmt 指定了转存出来的内存信息的格式 |
| xp /fmt addr | 转存出从 addr 开始的物理内存地址，fmt 指定了转存出来的内存信息的格式 |
| print fmt expr | 按 fmt 格式打印 expr 表达式的值 |

更多的命令可以通过 help 来查看。

# 9 qemu-kvm 命令行参数

## 9.1 与配置客户机相关的参数

以下是 qemu-kvm 命令行中一些常用的参数：

| | |
|---|---|
| -help | 查看 qemu-kvm 命令行中参数以及命令的帮助信息 |
| -cpu 参数 | 指定 CPU 模型，“-cpu ?”可以查询当前 qemu-kvm 支持哪些 CPU 模型 |
| -smp 参数 | 设置客户机的逻辑 CPU |
| -m 参数 | 设置客户机内存大小 |
| -mem-path | 为客户机分配内存，主要是分配大页内存 |
| -balloon | 开启内存气球 |
| -hda -hdb -cdrom | 设置客户机的 IDE 磁盘和光盘设备 |
| -drive | 详细地配置一个驱动器 |
| -boot | 设置客户机启动时的各种选项 |
| -net nic | 为客户机创建一个网卡，并可以详细配置该网卡 |
| -net user | 让客户机使用不需要管理员权限的用户模式网络 |
| -net tap | 使用宿主机的 TAP 网络接口来帮助客户机建立网络 |
| -net dump | 转存出网络中的数据流量 |
| -net none | 不配置任何的网络设备 |
| -sdl | 使用 SDL 方式显示客户机 |
| -vnc | 使用 VNC 方式显示客户机 |
| -vga | 设置客户机中的 VGA 显卡类型 |
| -nographic | 关闭 qemu 的图形界面输出 |
| -device | 添加一个设备驱动器 |
| -incoming | 让 qemu-kvm 进程进入到迁移监听模式，而不是真正以命令行中的镜像文件运行客户机 |
| -daemonize | 让 qemu-kvm 作为守护进程在后台运行，这样 qemu-kvm 就不会占用着当前的 terminal |
| -usb | 开启客户机中的 USB 总线 |
| -version | 显示 qemu-kvm 的版本信息 |
| -k | 设置键盘布局的语言 |
| -soundhw | 开启声卡硬件的支持 |
| -display | 设置显示方式 |
| -name | 设置客户机名称 |
| -uuid | 设置系统的 UUID |
| -rtc | 设置 RTC 开始时间和时钟类型 |

| | |
|---|---|
| -chardev | 配置字符型设备 |
| -bios | 指定客户机的 BIOS 文件 |
| -loadvm | 加载快照状态 |
| -pidfile | 保存进程 ID 到文件中 |
| -nodefaults | 不创建默认的设备 |
| -readconfig | 从文件中读取客户机设备的配置 |
| -writeconfig | 将客户机中设备的配置写到文件中 |
| -nodefconfig | 使 qemu-kvm 不加载默认的配置文件 |
| -no-user-config | 使 qemu-kvm 不加载用户定义的配置文件 |

## 9.2 与调试相关的参数

qemu-kvm 中也有一些与调试相关的参数，如下所示：

| | |
|---|---|
| -singlestep | 以单步执行的模式运行 qemu 模拟器 |
| -S | 在启动时并不启动 CPU，需要在 monitor 中运行“c”命令才能继续运行 |
| -d | 将 qemu 的日志保存在/tmp/qemu.log 中，以便调试时查看日志 |
| -D logfile | 将 qemu 的日志保存在 logfile 中，以便调试时查看日志 |

# 10 迁移到 KVM 虚拟化环境

## 10.1 从一种虚拟化迁移到另一种虚拟化

virt-v2v 工具可用于将虚拟客户机从一些 hypervisor 迁移到 KVM 环境中。使用如下命令可以安装 virt-v2v：

```
sudo apt-get install virt-manager
sudo apt install libguestfs-tools
```

下面通过一个例子介绍 virt-v2v 的使用方法。现在要将 Xen 上的 HVM 客户机迁移到 KVM 上，源宿主机的 IP 地址是 192.168.127.163。步骤如下所示：

1. 在源宿主机系统中启动 libvirtd 服务，并且关闭所要迁移的客户机，该客户机名为 xen-hvm1。启动 libvirtd 服务的命令如下：

   ```
   service libvirtd start
   ```

2. 在 KVM 宿主机中启动 libvirtd 服务，然后使用 virt-v2v 工具进行从 Xen 到 KVM 的迁移。相关命令如下：

   ```
   service libvirtd start
   virt-v2v -ic xen+ssh://root@192.168.127.163 -o default -b br0 xen-hvm1
   ```

   从 Xen 上迁移过来的客户机，其镜像文件默认在/var/lib/libvirt/images/目录下，其 XML 配置文件默认在/etc/libvirt/qemu/目录下。

3. 根据客户机的配置文件直接开启客户机，命令如下：

   ```
   virsh create /etc/libvirt/qemu/xen-hvm1.xml
   ```

这里介绍一下 virt-v2v 工具：

| 从 Xen 客户机迁移到 KVM 的命令如下：<br>virt-v2v -ic xen+ssh://root@xen0.demo.com -os pool -b brname vm-name | |
|---|---|
| 其中的参数介绍如下： | |
| -ic URI | 表示连接远程 Xen 宿主机中 libvirt 的 URI |
| -os pool | 表示迁移过来后，用于存放镜像文件的本地目录 |
| -b brname | 表示本地网桥的名称，用于建立与客户机的网络连接。如果本地使用虚拟网络，则使用-n nerwork 参数来指定虚拟网络 |
| vm-name | 表示在 Xen 的源宿主机中将要被迁移的客户机的名称 |

## 10.2 从 VMware 迁移到 KVM

### 10.2.1 借助 virt-v2v 工具实现迁移

从 VMware 迁移到 KVM 的方法，与前一节中从 Xen 迁移到 KVM 的完全类似。下面通过一个例子介绍迁移过程。现在要将 VMware ESX 上的一个客户机迁移到 KVM 上，命令行如下：

```
virt-v2v -ic esx+ssh://esx.demo.com/ -os pool -b brname vm-name
```

当连接到 VMware 的 ESX 服务器时，一般需要认证和授权，此时 virt-v2v 读取 $HOME/.netrc 文件中的机器名、用户名和密码用于认证。$HOME/.netrc 文件中信息的格式如下所示：

```
machine esx.demo.com login root password 123456
```

### 10.2.2 通过复制客户机镜像实现迁移

这个方法很直接：先关闭 VMware 客户机，然后直接将 VMware 的客户机镜像复制到 KVM 的客户机镜像存储系统上，接着通过 qemu-kvm 命令行工具启动该镜像文件即可。命令如下所示：

```
qemu-system-x86_64 -m 1024 win7.vmdk
```

也可以将 vmdk 的镜像文件转化为 qcow2 格式，然后用 qemu-kvm 命令行启动。相关命令如下所示：

```
qemu-img convert win7.vmdk -O qcow2 win7.qcow2
qemu-system-x86_64 -m 1024 win7.qcow2
```

## 10.3 从 VirtualBox 迁移到 KVM

从 VirtualBox 迁移到 KVM 完全类似于从 VMware 迁移到 KVM。

借助 virt2-v2v 的方法如下所示：

```
virt-v2v -ic vbox+ssh://root@vbox.demo.com -os pool -b brname vm-name
```

也可以直接将 VirtualBox 中的客户机镜像文件复制到 KVM 宿主机中使用，qemu-kvm 可以直接将.vdi 格式的镜像文件作为客户机镜像直接启动：

```
qemu-system-x86_64 -m 1024 ubuntu.vdi
```

也可以将.vdi 的镜像文件转化为 qcow2 格式，然后用 qemu-kvm 命令行启动。相关命令如下所示：

```
qemu-img convert ubuntu.vdi -O qcow2 ubuntu.qcow2
qemu-system-x86_64 -m 1024 ubuntu.qcow2
```

## 10.4 从物理机迁移到 KVM 虚拟化环境

使用 virt-p2v 工具的方法将物理机迁移到 KVM 虚拟化环境中需要经过以下几个步骤：

1. 在 KVM 宿主机上安装 virt-v2v、libvirt 等工具。
2. 修改宿主机/etc/virt-v2v.conf 配置文件，如下所示：

   首先打开/etc/virt-v2v.conf 文件，命令如下：

   ```
   sudo vi /etc/virt-v2v.conf
   ```

   然后将其配置为如下内容：

3. 制作 virt-p2v 可以启动的设备。将 virt-p2vISO 镜像文件烧录到 U 盘中，作为物理机的启动设备。
4. 在待迁移的物理机上，选择第 3 步中制作的设备来启动系统。
5. 在 virt-p2v 客户端启动后，根据提示，填写第二步中所写的服务端的 IP、登陆用户等信息。在连接到服务器端后，可以进一步填写转移后的虚拟客户机的配置，最后点击“convert”按钮，virt-p2v 客户端会讲物理机中的磁盘信息通过网络传输到服务器端，待传输完成后，选择关闭该物理机即可。
6. 在 KVM 宿主机上通过 qemu-kvm 命令行启动第 5 步转移过来的客户机即可。