虚拟化技术（KVM）

虚拟化

一、虚拟化技术概述

学习目标

- [ ] 能够了解虚拟化技术分类方法
- [ ] 能够了解虚拟化技术实现目的
- [ ] 能够了解虚拟化产品有哪些
- [ ] 能够了解QEMU功能
- [ ] 能够掌握KVM功能
- [ ] 能够掌握libvirt功能

1.虚拟化技术即是对资源的抽象

2.从资源提供角度分类

2.1 平台虚拟化

如果把X86平台的CPU，内存和外设做为资源，那对应的虚拟化技术就是平台虚拟化，在同一个X86平台上面，可以虚拟多个X86平台，每个平台可以运行自己独立完整的操作系统。

例如：QEMU,KVM,XEN Server,EXSi，Hyper-V等所管理的虚拟机

2.2 操作系统虚拟化

如果把操作系统及其提供的系统调用作为资源，那虚拟化就表现为操作系统虚拟化，例如Linux容器虚拟化技术就是在同一个Linux操作系统之上，虚拟出多个同样的操作系统，每个应用程序认为自己运行在一个独立的OS。

例如：LXC或Docker等所管理的容器

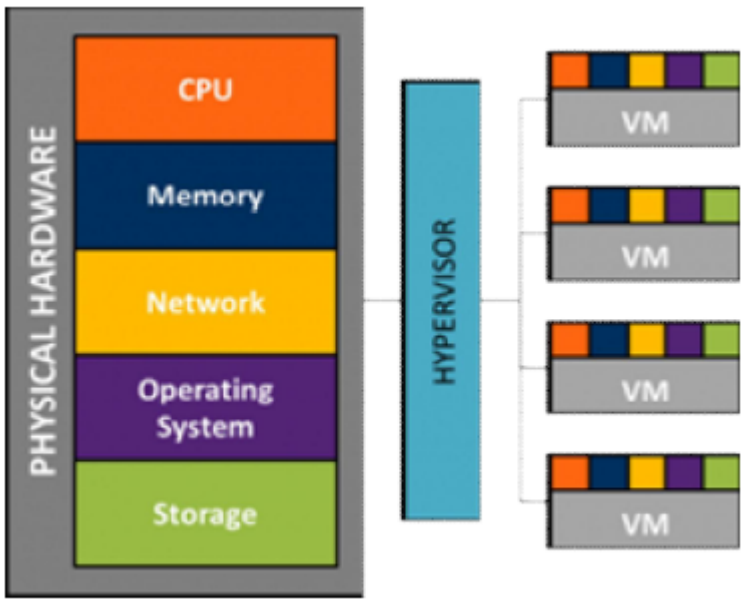
3.从虚拟化实现方式分类

3.1 什么是虚拟化管理程序 Hypervisor（VMM）

一种运行在物理机和虚拟机操作系统之间的中间软件层，可以允许多个操作系统和应用共享硬件，即虚拟机监视器，也可称之为VMM。

Hypervisors是一种在虚拟环境中的“元”操作系统。他们可以访问服务器上包括磁盘和内存在内的所有物理设备。Hypervisors不但协调着这些硬件资源的访问，而且在各个虚拟机之间施加防护。当服务器启动并执行Hypervisor时，它会加载所有虚拟机客户端的操作系统同时会分配给每一台虚拟机适量的内存，CPU，网络和磁盘。

如下图所示：



3.1.1 Hypervisors作用

Hypervisor是所有虚拟化技术的核心。 非中断地支持多工作负载迁移的能力是Hypervisor的基本功能。

3.1.2 Hypervisors分类

目前市场上各种x86 管理程序(hypervisor)的架构存在差异，三个最主要的架构类别包括：

I型：虚拟机直接运行在系统硬件上，创建硬件全仿真实例，被称为“裸机”型。 裸机型在虚拟化中Hypervisor直接管理调用硬件资源，不需要底层操作系统，也可以将Hypervisor看作一个很薄的操作系统。这种方案的性能处于主机虚拟化与操作系统虚拟化之间。

> 3.2小节示图

II型：虚拟机运行在传统操作系统(HOST OS)上，同样创建的是硬件全仿真实例，被称为“托管（宿主）”型。托管型/主机型Hypervisor运行在基础操作系统上，构建出一整套虚拟硬件平台（CPU/Memory/Storage/Adapter），使用者根据需要安装新的操作系统和应用软件，底层和上层的操作系统可以完全无关化，如Windows运行Linux操作系统。主机虚拟化中VM的应用程序调用硬件资源时需要经过:VM内核->Hypervisor->主机内核，因此相对来说，性能是三种虚拟化技术中最差的。

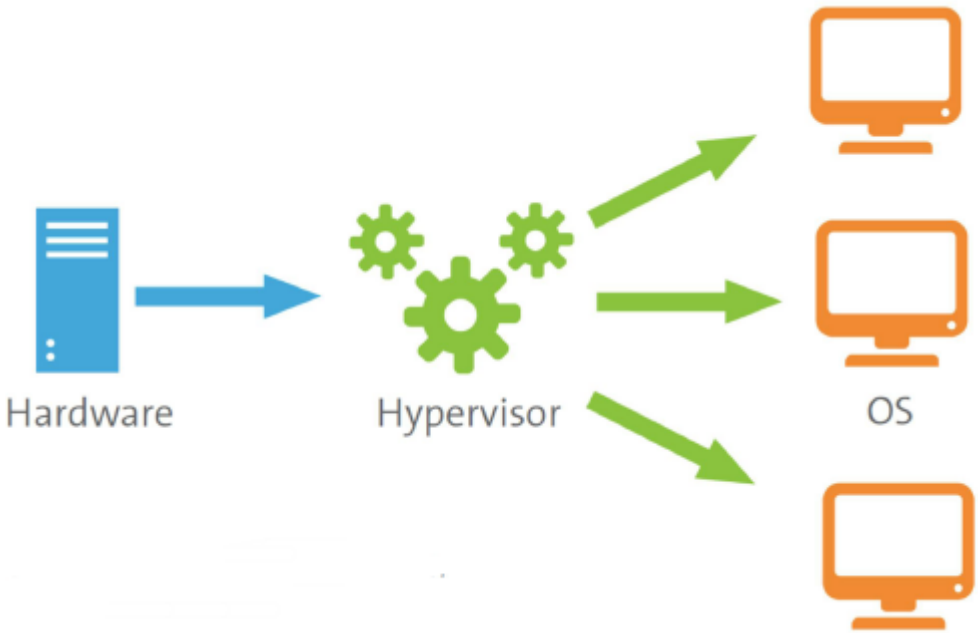
> 3.3小节示图

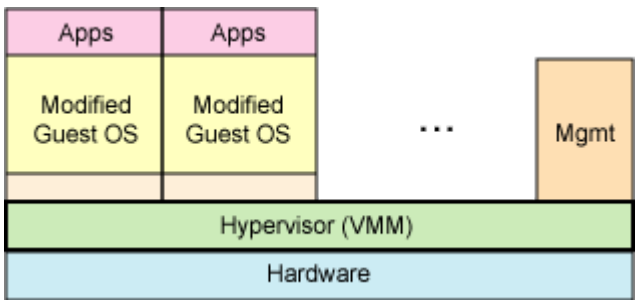
Ⅲ型：虚拟机运行在传统操作系统上，创建一个独立的虚拟化实例（容器），指向底层托管操作系统，被称为“操作系统虚拟化”。操作系统虚拟化是在操作系统中模拟出运行应用程序的容器，所有虚拟机共享内核空间，性能最好，耗费资源最 少。但是缺点是底层和上层必须使用同一种操作系统，如底层操作系统运行的是Windows系统，则VPS/VE就必须运行Windows？

> 参考3.4及3.5小节示图

3.2 Hypervisor type 1:半虚拟化（Para-virtualization）

对客户操作系统（VM）的内核进行修改，将运行在Ring 0上的指令转为调用Hypervisor，例如：XEN

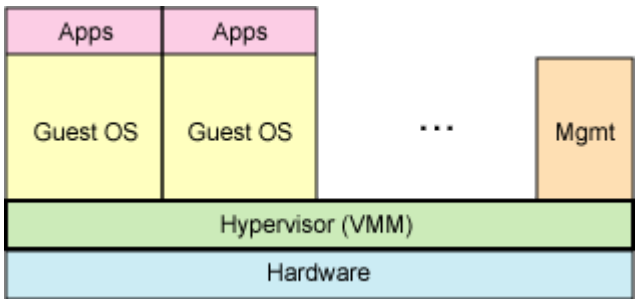


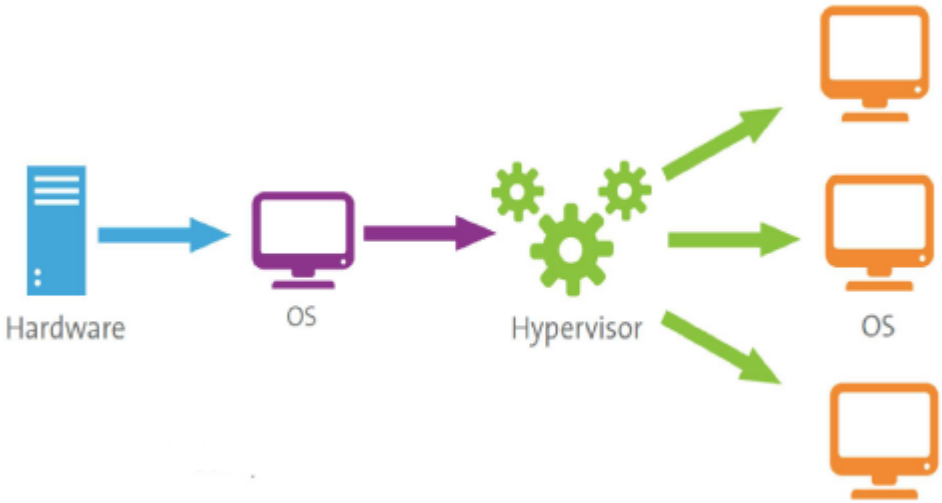


3.3 Hypervisor type 2:硬件辅助全虚拟化（Hardware-Assisted Full Virtualzation）

3.3.1)对CUP指令集进行改造，例如：Inter VT-x / AMD-V

3.3.2)客户操作系统可以直接使用Ring 0而无需要修改，例如：KVM





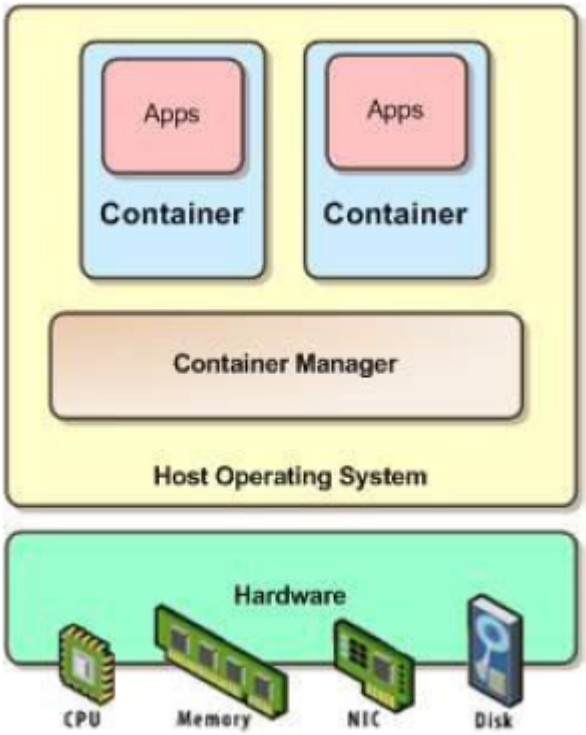
3.4 Hypervisor type 3:软件全虚拟化

所谓的软件全虚拟化，即非硬件辅助全虚拟化，模拟CPU让VM使用，效率低，例如：QEMU

3.5 操作系统虚拟化（Hypervisor type 3）

称为轻量级虚拟化，允许操作系统内核拥有彼此隔离和分割的多用户空间实例（instance），这些实例也被称之为容器。其是基于Linux内核中的namespace,cgroup实现

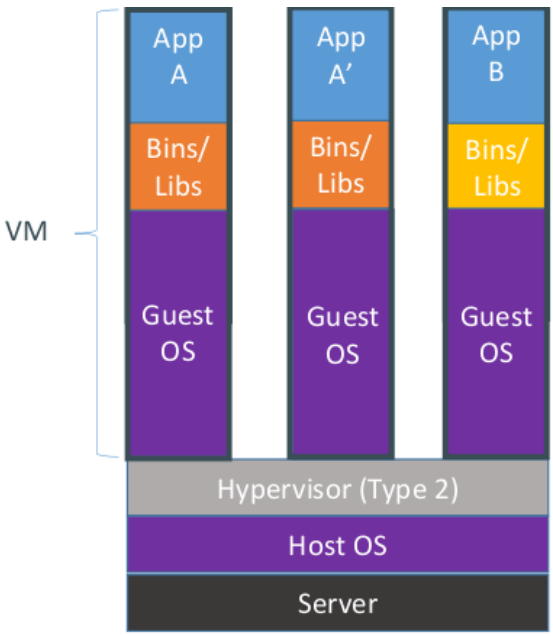
例如：LXC,Docker



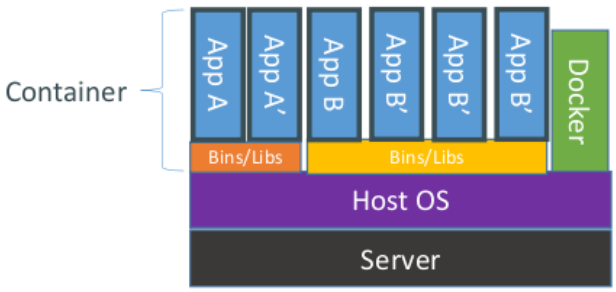
总结：

- typeI 半虚拟化
- typeII 硬件辅助全虚拟化
- typeIII 软件全虚拟化或者叫操作系统虚拟化

3.6 虚拟机与容器对比



Container和普通的虚拟机Image相比，最大的区别是它并不包含操作系统内核



3.7 Hypervisor管理工具的对比

名称	厂商	主CPU	目标CPU	主系统	目标系统
qemu	Fabrice Bellard（其他开发者帮助）	Intel，AMD，ARM	x86，x86-64，ARM	Windows，Linux，Mac OS X	Linux
kvm	Red Hat	Intel，AMD（硬件辅助全虚拟化）	x86，x86-64	Linux	Linux，Windows
Xen	英国剑桥大学，Intel，AMD	Intel，AMD（半虚拟化、硬件辅助全虚拟化）	x86/x86-64	Linux，Solaris，Windows	Linux，Windows
Hyper-V	微软	Intel，AMD（硬件辅助全虚拟化）	x86/x86-64	Windows	Linux，Windows
VMware ESXi Server	VMware	Intel，AMD（硬件辅助全虚拟化）	x86/x86-64	裸机安装	Linux，Windows
Oracle VM	Oracle	Intel，AMD（硬件辅助全虚拟化）	x86/x86-64	裸机安装	Linux，Windows

总结：

- KVM linux
- vmware esxi

3.8 QEMU

软件模拟虚拟化、可以模拟多种硬件，包括X86架构处理器、AMD64架构处理器、ARM、SPARC 与 PowerPC ，AIX架构等。效率低，一般用于研究测试场景。

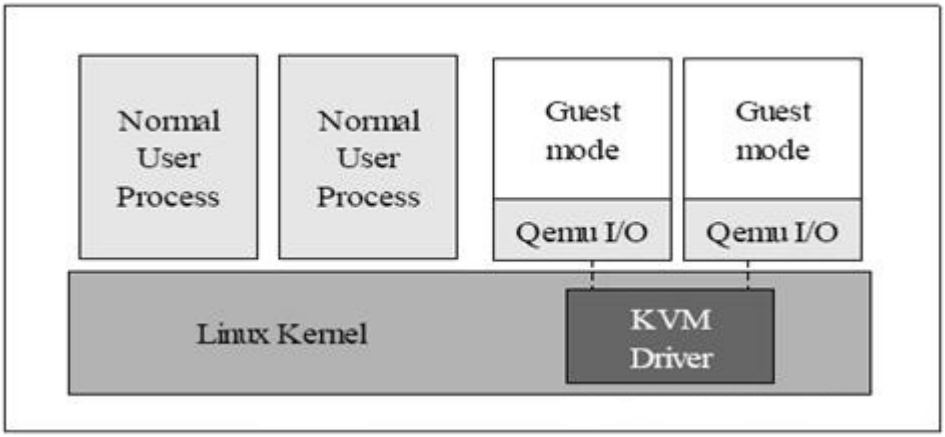
此处使用QEMU的主要目的是为了方便管理KVM虚拟机镜像文件 ？

注：

QEMU由Fabrice Bellard编写的模拟处理器的自由软件，它是一个完整的可以单独运行的软件，可以独立模拟出整台计算机，包括CPU，内存，IO设备，通过一个特殊的“重编译器”对特定的处理器的二进制代码进行翻译，从而具有了跨平台的通用性。QEMU有两种工作模式：系统模式，可以模拟出整个电脑系统，另一种是用户模式，可以运行不同与当前硬件平台的其他平台上的程序（比如在x86平台上运行跑在ARM平台上的程序）；其代码地址 http://git.qemu.org/qemu.git，1.0以后版本就只能使用qemu-kvm（只支持Linux）进行加速了，1.3版本后QEMU和QEMU-KVM合二为一了。

3.9 KVM

- KVM kernel-based virtual machine 基于内核的虚拟机 是x86架构下硬件辅助的全虚拟化的首选解决方案 KVM需要经过修改的QEMU软件（qemu-kvm）来实现虚拟机的管理 KVM就是内核的一个模块，用户空间通过QEMU模拟硬件提供给虚拟机使用，一台虚拟机就是一个普通的Linux进程，虚拟机中的VCPU就是该进程中的线程。



注：

不同的基于KVM的虚拟化平台，可能会采用不同的虚拟化组件，目前主流的采用QEMU-KVM组件，但在不同的产品里版本有所不同，功能也有差异。

KVM通过调用Linux本身内核功能，实现对CPU的底层虚拟化和内存的虚拟化，使Linux内核成为虚拟化层，需要x86架构支持虚拟化功能的硬件支持（比如Intel-VT，AMD-V），是一种全虚拟化架构。

KVM在2007年2月被导入Linux 2.6.20内核中。

从存在形式来看，它包括两个内核模块：kvm.ko 和 kvm_intel.ko（或kvm_amd.ko），本质上，KVM是管理虚拟硬件设备的驱动，该驱动使用字符设备/dev/kvm（由KVM本身创建）作为管理接口，主要负责vCPU的创建，虚拟内存的分配，vCPU寄存器的读写以及vCPU的运行。

KVM是linux内核的模块，它需要CPU的支持，采用硬件辅助虚拟化技术Intel-VT，AMD-V，内存的相关如Intel的EPT和AMD的RVI技术，Guest OS的CPU指令不用再经过Qemu转译，直接运行，大大提高了速度，KVM通过/dev/kvm暴露接口，用户态程序可以通过ioctl函数来访问这个接口。

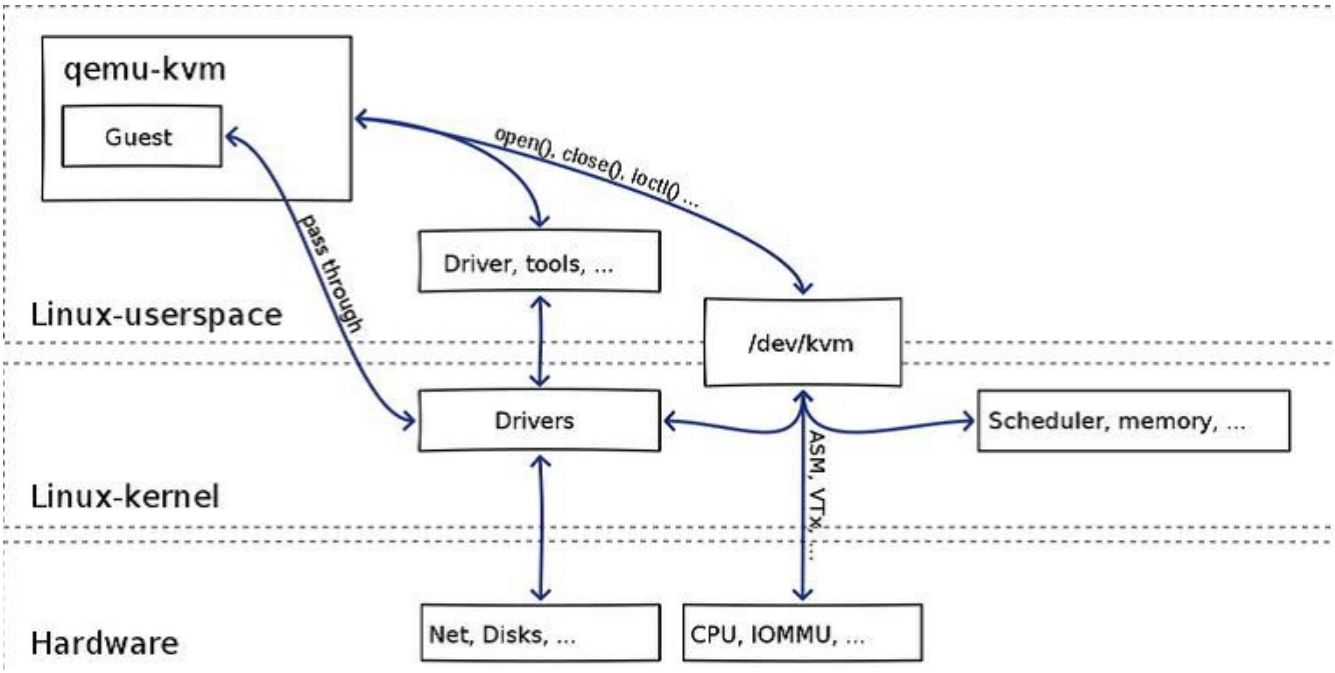
总结：

- 是基于内核的虚拟机
- 是内核模块
- 做为驱动出现
- 在硬件辅助全虚拟化下虚拟出CPU,MEM
- 暴露接口/dev/kvm

3.10 QEMU-KVM

从前面对KVM内核模块的介绍知道，它只负责CPU和内存的虚拟化，加载了它以后，用户就可以进一步通过工具创建虚拟机（KVM提供接口），但仅有KVM还是不够的，用户无法直接控制内核去做事情（KVM只提供接口，怎么创建虚拟机，分配vCPU等并不在它上面进行），还必须有个运行在用户空间的工具才行，KVM的开发者选择了比较成熟的开源虚拟化软件QEMU来作为这个工具，并对其进行了修改，最后形成了QEMU-KVM。

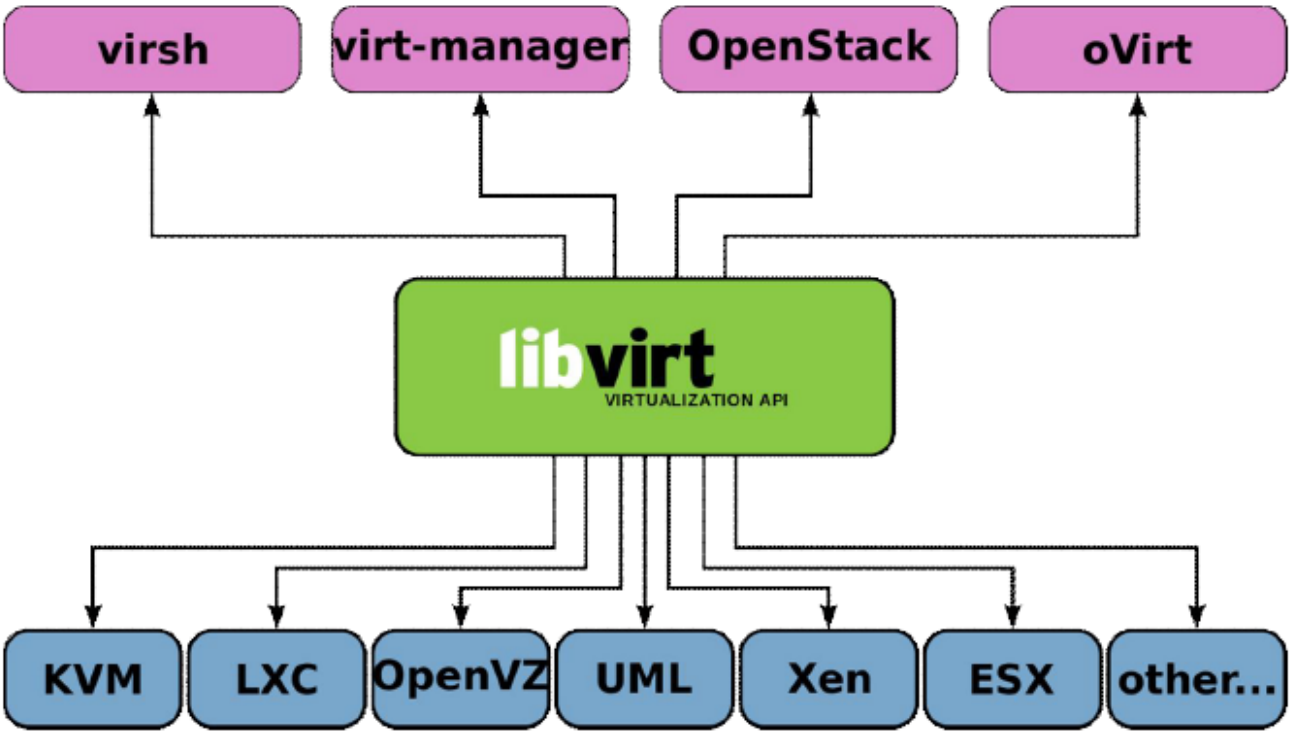
在QEMU-KVM中，KVM运行在内核空间，QEMU运行在用户空间，实际模拟创建，管理各种虚拟硬件，QEMU将KVM整合了进来，通过/ioctl 调用 /dev/kvm，从而将CPU指令的部分交给内核模块来做，KVM实现了CPU和内存的虚拟化，但kvm不能虚拟其他硬件设备，因此qemu还有模拟IO设备（磁盘，网卡，显卡等）的作用，KVM加上QEMU后就是完整意义上的服务器虚拟化。当然，由于qemu模拟io设备效率不高的原因，现在常常采用半虚拟化的virtio方式来虚拟IO设备。



QEMU-KVM作用： 1.提供对cpu，内存（KVM负责），IO设备（QEMU负责）的虚拟 2.对各种虚拟设备的创建，调用进行管理（QEMU负责）

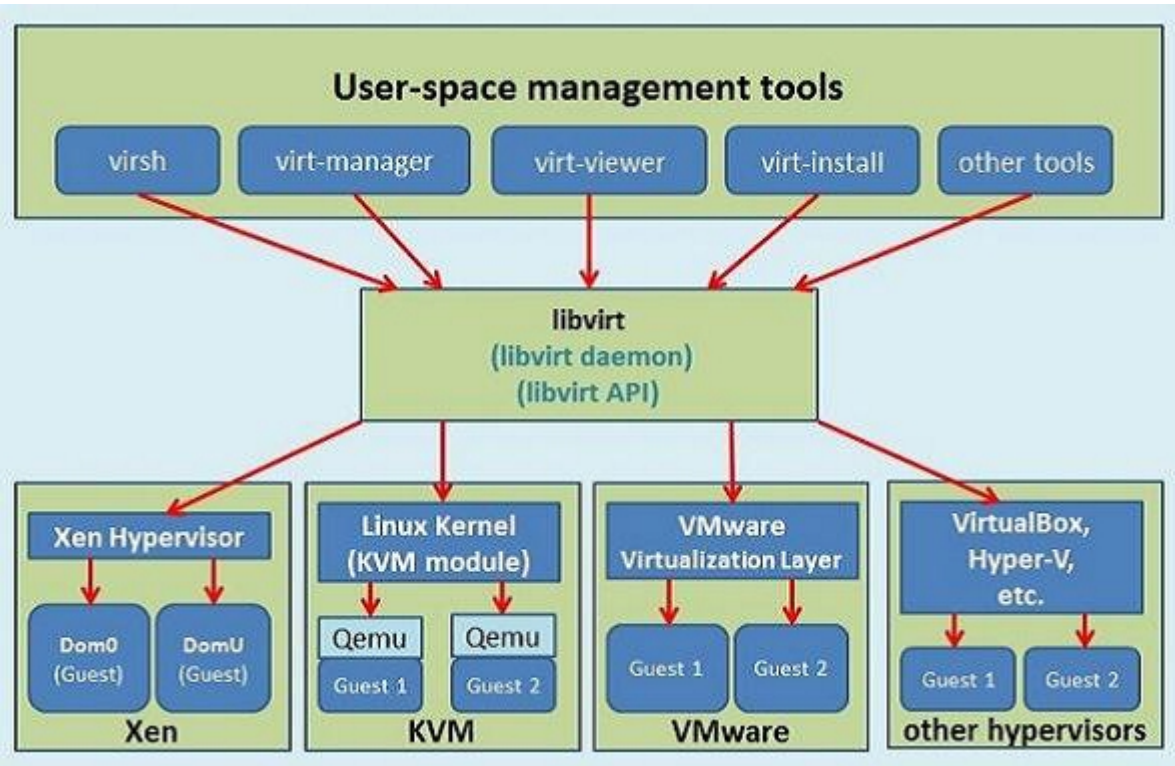
3.11 libvirt

libvirt是一套免费、开源的支持Linux下主流虚拟化管理程序的C函数库，其旨在为包括KVM在内的各种虚拟化管理程序提供一套方便、可靠的编程接口。 当前主流Linux平台上默认的虚拟化管理工具virt-manager,virsh等都是基于libvirt开发。



注：

一个针对各种虚拟化平台的虚拟机管理的API库，一些常用的虚拟机管理工具如virsh（类似vim编辑器），virt-install，virt-manager等和云计算框架平台（如OpenStack，OpenNebula，Eucalyptus等）都在底层使用libvirt提供的应用程序接口。libvirt主要由三个部分组成：API库，一个守护进程 libvirtd 和一个默认命令行管理工具 virsh。



QEMU,KVM,Libvirt三者关系总结：

QEMU是一个独立的虚拟化解决方案，并不依赖KVM（它本身自己可以做CPU和内存的模拟，只不过效率较低），而KVM是另一套虚拟化解决方案，对CPU进行虚拟效率较高（采用了硬件辅助虚拟化），但本身不提供其他设备的虚拟化，借用了QEMU的代码进行了定制，所以KVM方案一定要依赖QEMU 即使后来RedHat后来开发了libvirt，也只能简单的认为是个虚拟机管理工具，仍然需要通过用户空间QEMU来与KVM进行交互。

二、KVM虚拟机管理工具部署

学习目标

- [ ] 能够为KVM虚拟机管理工具部署准备环境
- [ ] 能够部署KVM虚拟机管理工具

1 KVM系统需求

```
Host system requirements
1.1核心
2.2G内存
3.6G硬盘
```

```
KVM hypervisor requirements
[root@localhost ~]# lscpu
虚拟化：           VT-x       #intel虚拟技术
```

```
[root@localhost ~]#  egrep 'svm|vmx'  /proc/cpuinfo
vmx
```

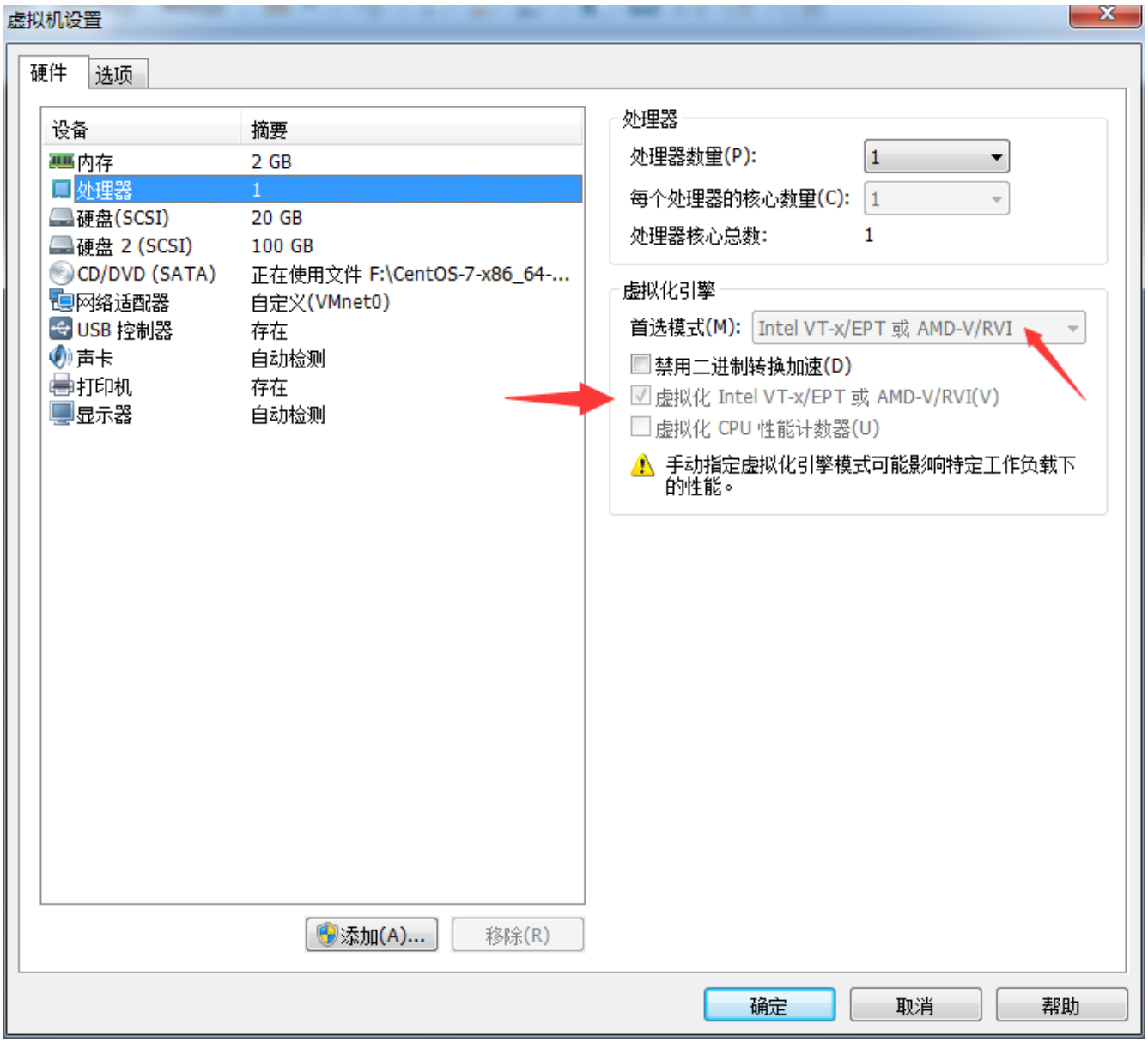
```
BIOS Enable Virtualization
```

Intel CPU: VT-x AMD CPU: AMD -V

总结：

- 物理机BIOS开启虚拟化支持
- 检查CPU flags是否包含vmx/smx

如果需要在vmware workstation 测试环境上安装kvm:



2 虚拟化主机部署

```
[root@localhost ~]# yum grouplist
[root@localhost ~]# yum -y groupinstall "虚拟化*"

```

> 如果具备CentOS7，可以按上述进行部署。

3 虚拟化主机验证

```
[root@localhost ~]#systemctl status libvirtd
[root@localhost ~]#lsmod | grep kvm
```

4 虚拟机网络连接状态

```
[root@localhost ~]#firewall-cmd --permanent --zone=external --list-all
```

```
==external==
  target: default
  icmp-block-inversion: no
  interfaces:
  sources:
  services: ssh
  ports:
  protocols:
  ==masquerade: yes==
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:

```

5 查看虚拟机列表

```
[root@localhost ~]# virsh list --all
```

三、KVM虚拟机安装

学习目标

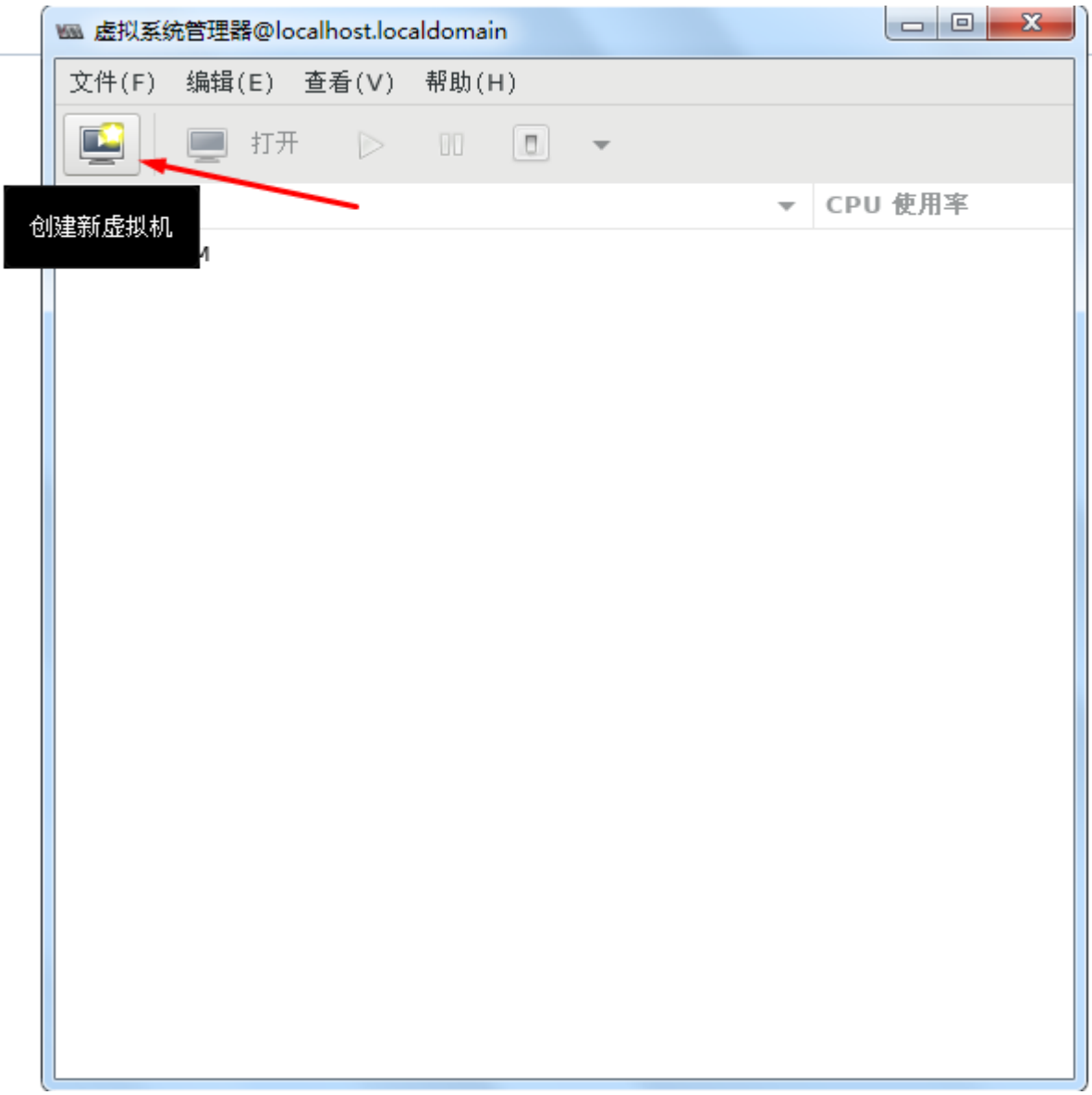
- [ ] 能够使用Virt-manager安装虚拟机
- [ ] 能够使用virt-install安装虚拟机

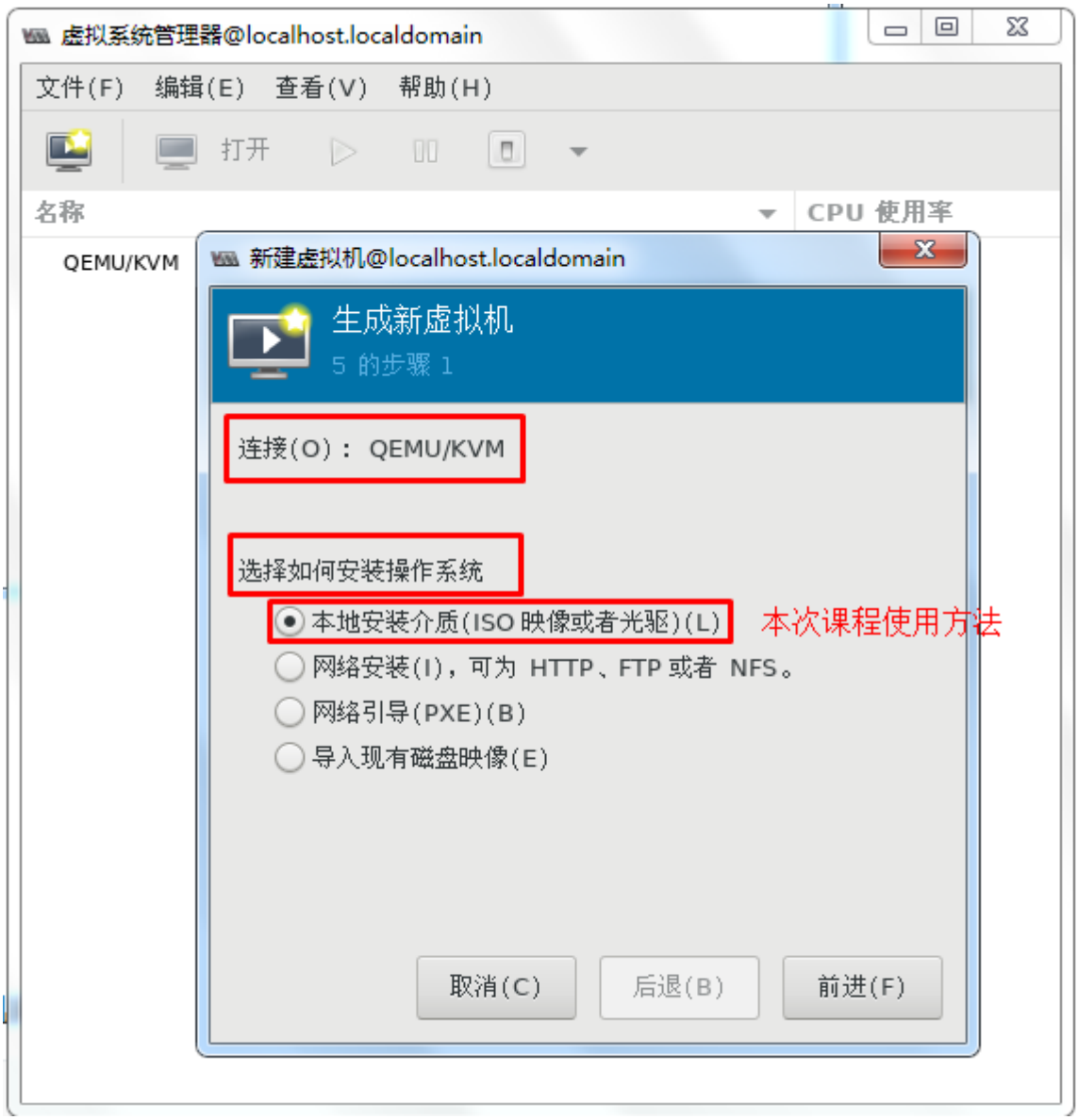
1 virt-manager

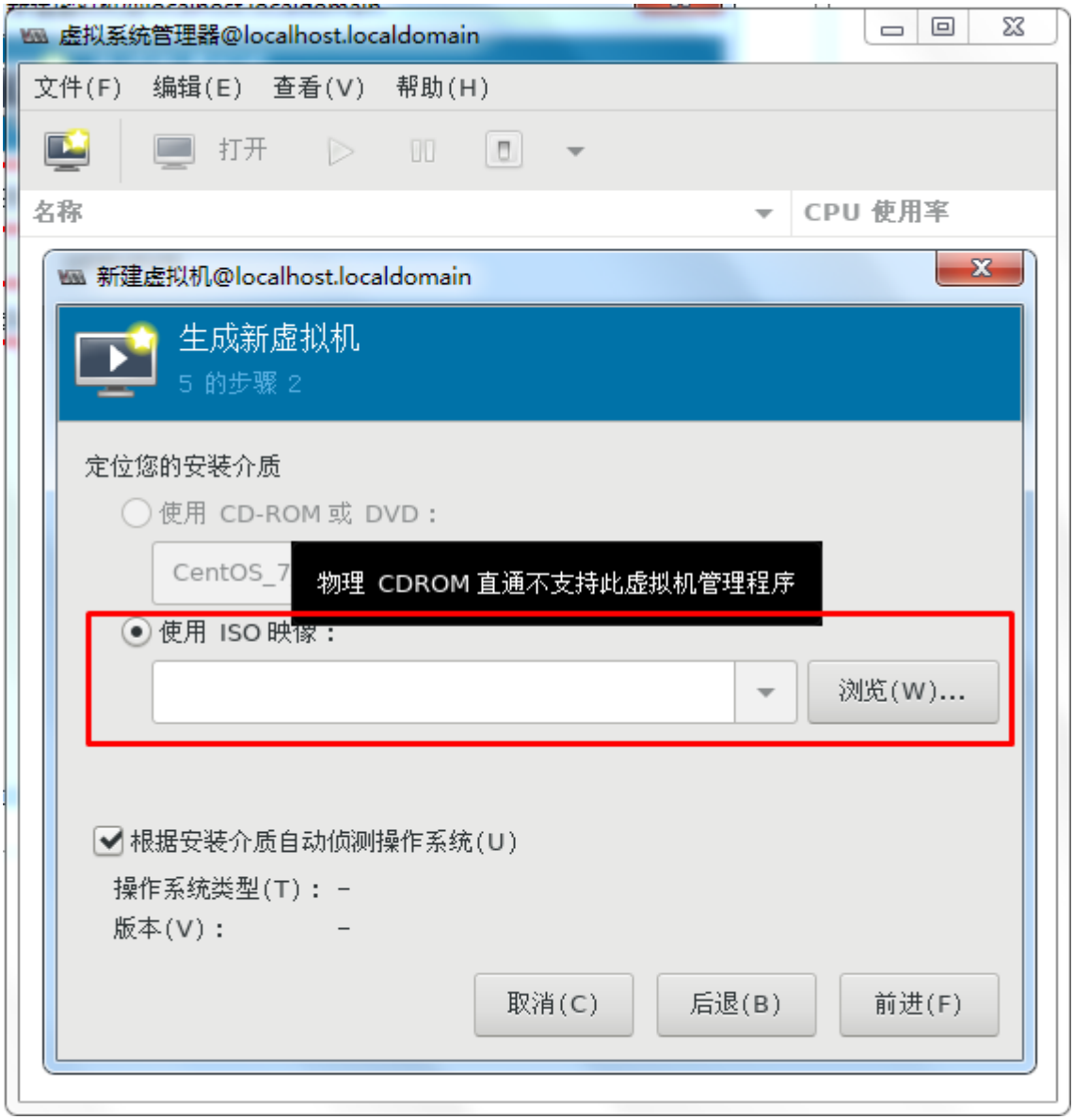
- 准备iso镜像文件
- 准备一套系统预备工具 PXE&kickstart cobbler

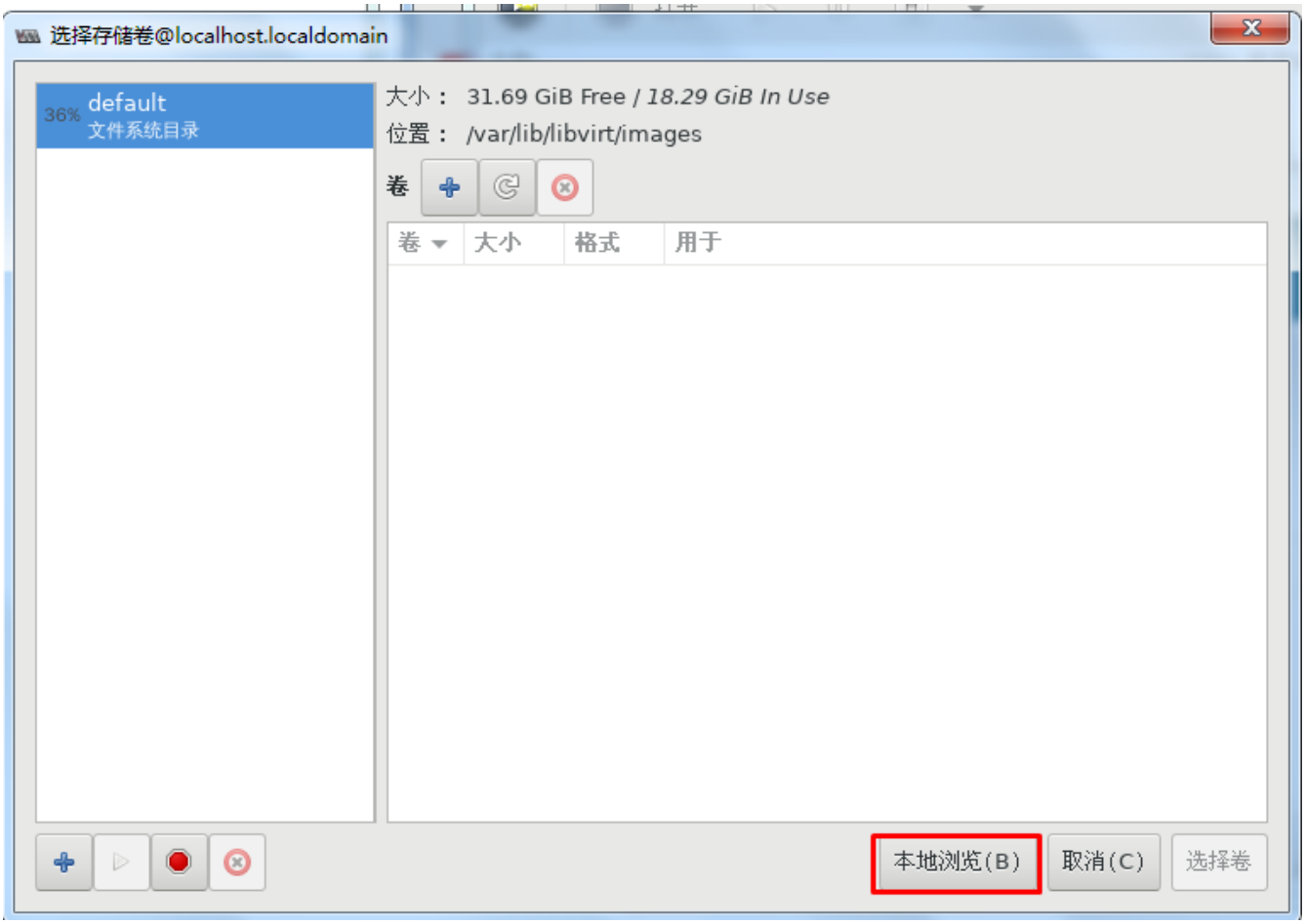
- 磁盘镜像文件 /var/lib/libvirt/images
- 配置文件 /etc/libvirt/qemu/

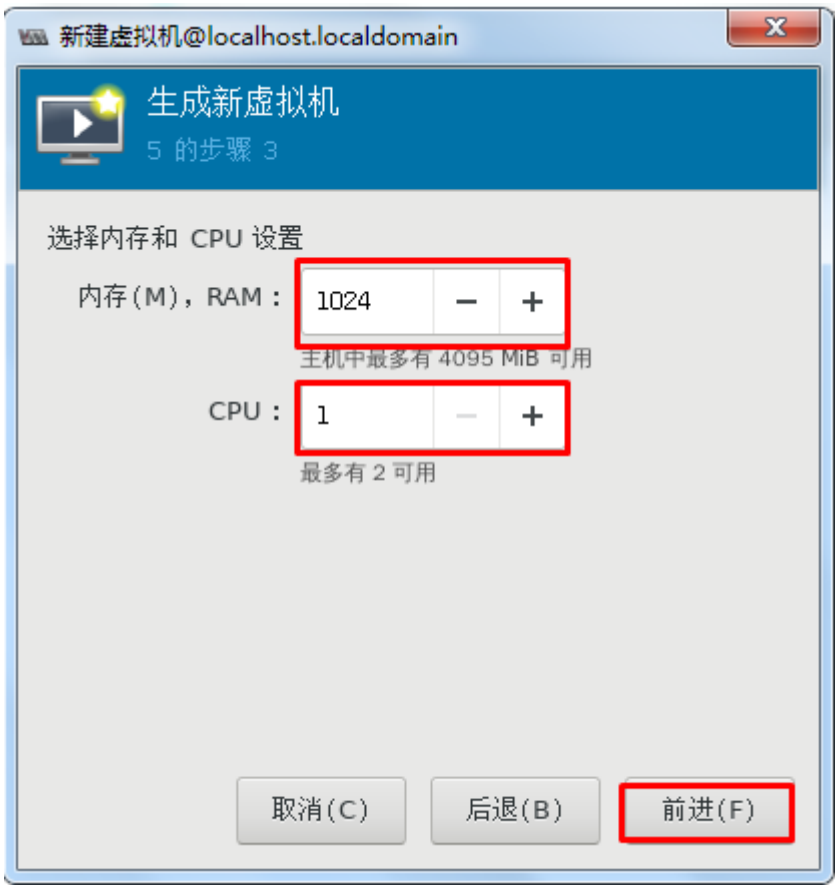
1.1 Linux主机安装


虚拟系统管理器@localhost.localdomain
文件(F)
编辑(E)
查看(V)
帮助(H)
打开
CPU 使用率
创建新虚拟机

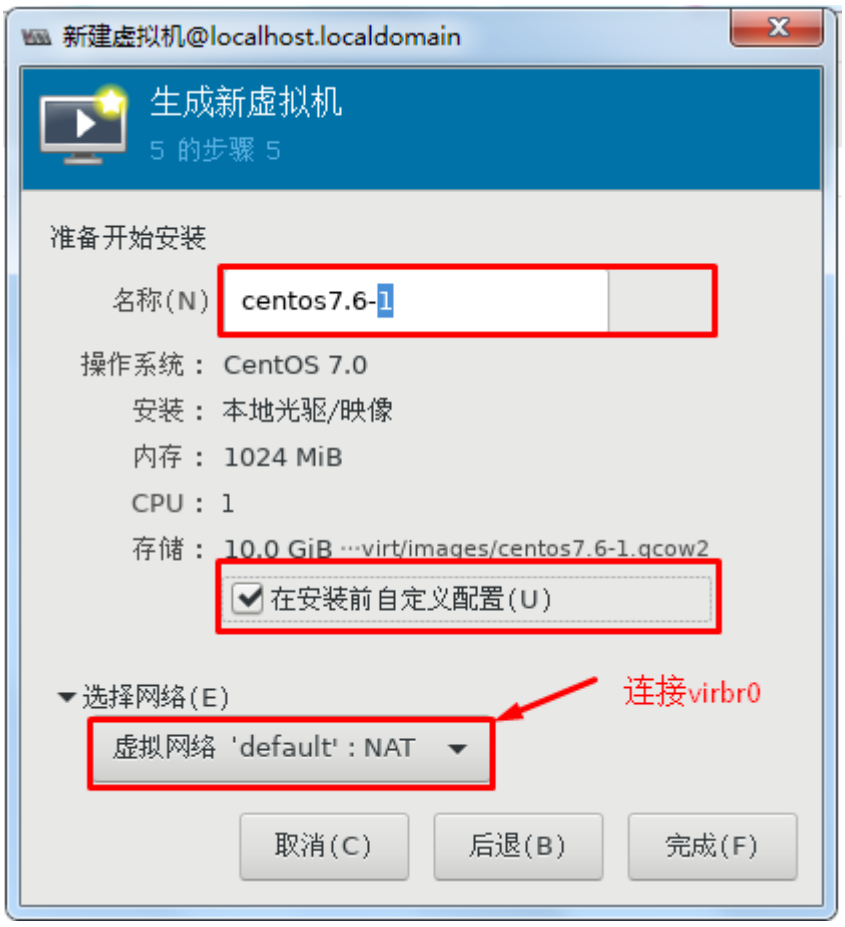

虚拟系统管理器@localhost.localdomain
文件(F) 编辑(E) 查看(V) 帮助(H)
打开
名称
CPU 使用率
QEMU/KVM
新建虚拟机@localhost.localdomain
生成新虚拟机
5 的步骤 1
连接(O)： QEMU/KVM
选择如何安装操作系统
本地安装介质(ISO 映像或者光驱)(L)
本次课程使用方法
网络安装(I)，可为 HTTP、FTP 或者 NFS。
网络引导(PXE)(B)
导入现有磁盘映像(E)
取消(C)
后退(B)
前进(F)

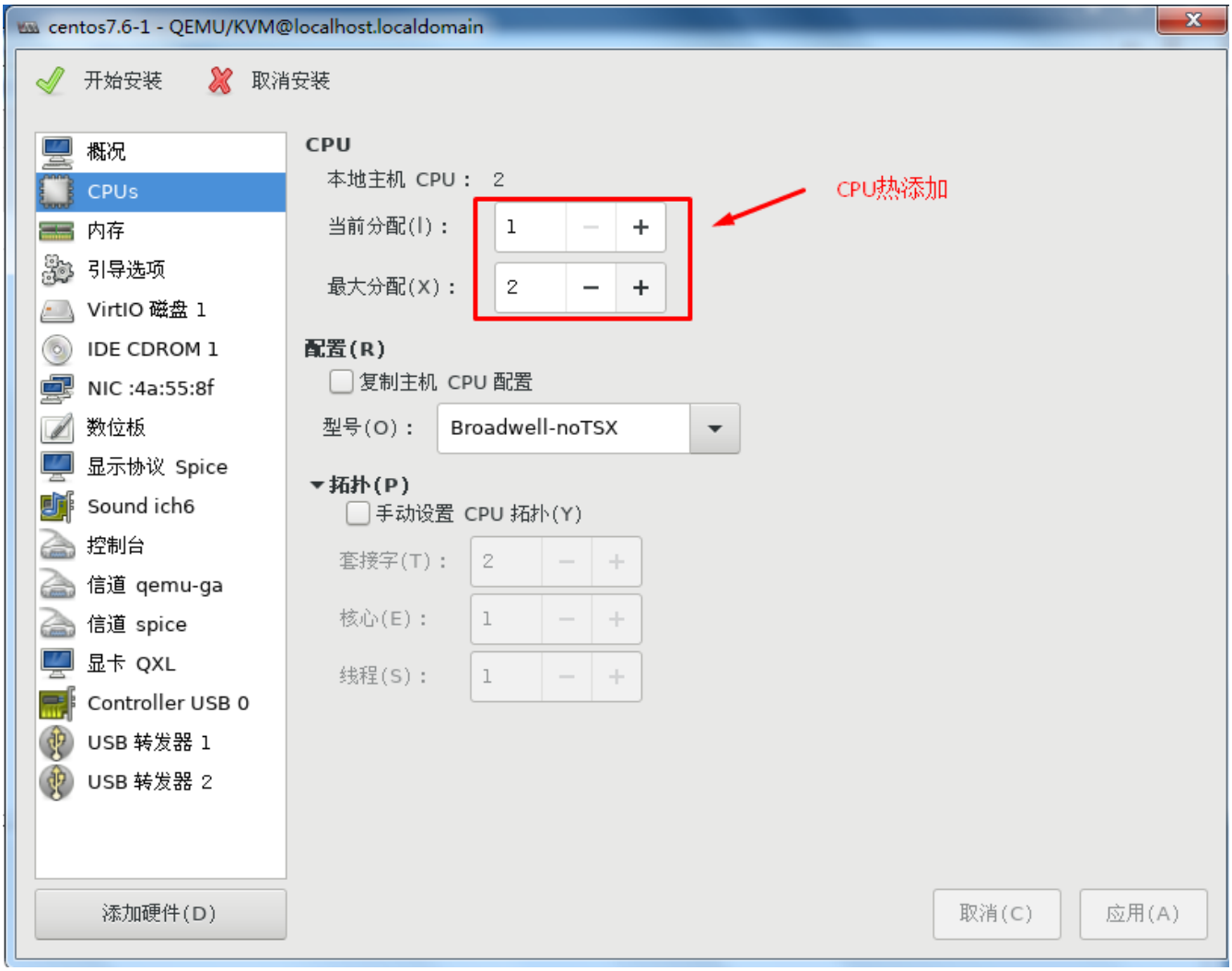

虚拟系统管理器@localhost.localdomain
文件(F) 编辑(E) 查看(V) 帮助(H)
打开
名称
CPU 使用率
新建虚拟机@localhost.localdomain
生成新虚拟机
5 的步骤 2
定位您的安装介质
使用 CD-ROM 或 DVD：
CentOS_7
物理 CDROM 直通不支持此虚拟机管理程序
使用 ISO 映像：
浏览(W)...
根据安装介质自动侦测操作系统(U)
操作系统类型(T)：
版本(V)：
取消(C)
后退(B)
前进(F)

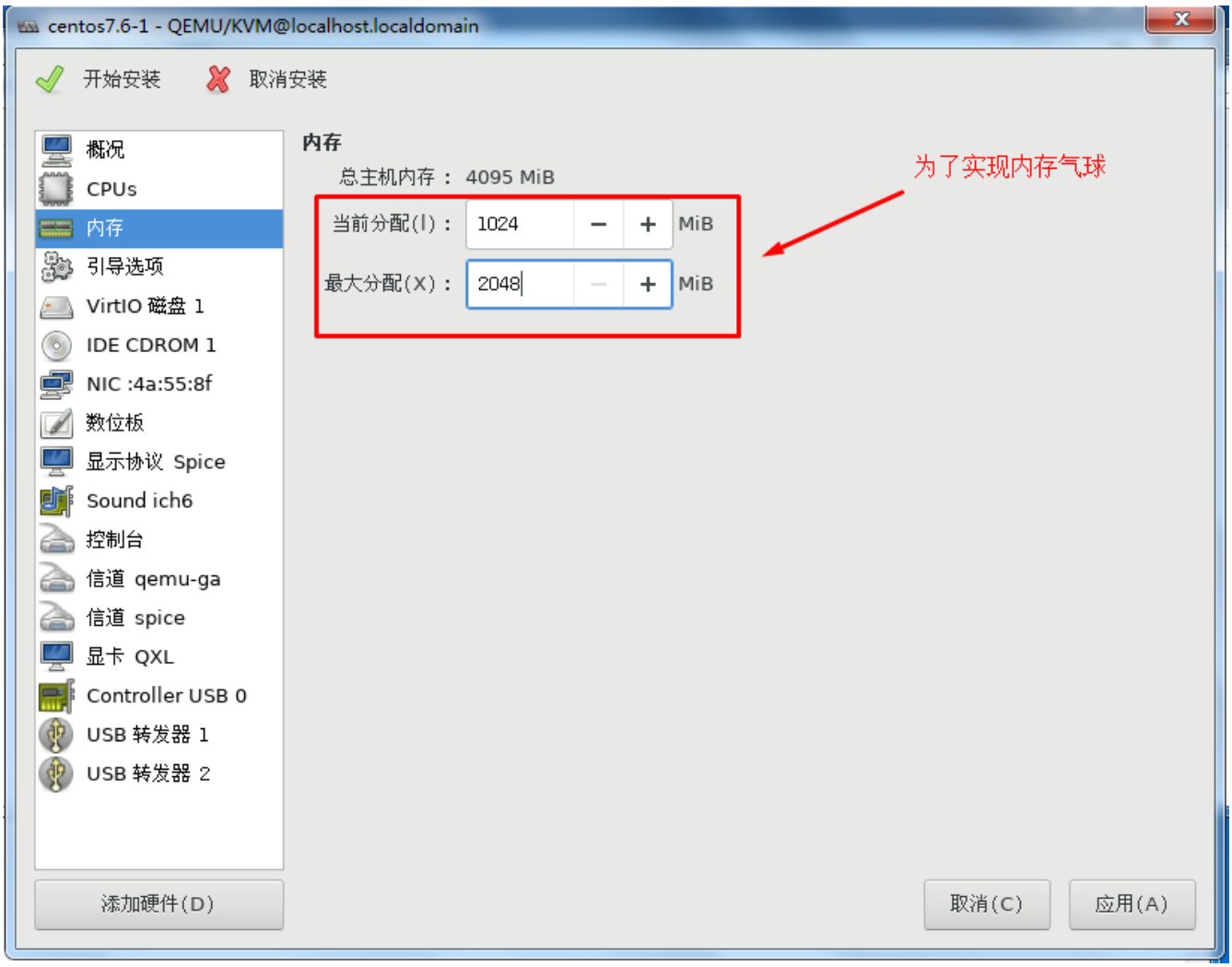

选择存储卷@localhost.localdomain
36%
default
文件系统目录
大小： 31.69 GiB Free / 18.29 GiB In Use
位置： /var/lib/libvirt/images
卷
卷
大小
格式
用于
本地浏览(B)
取消(C)
选择卷

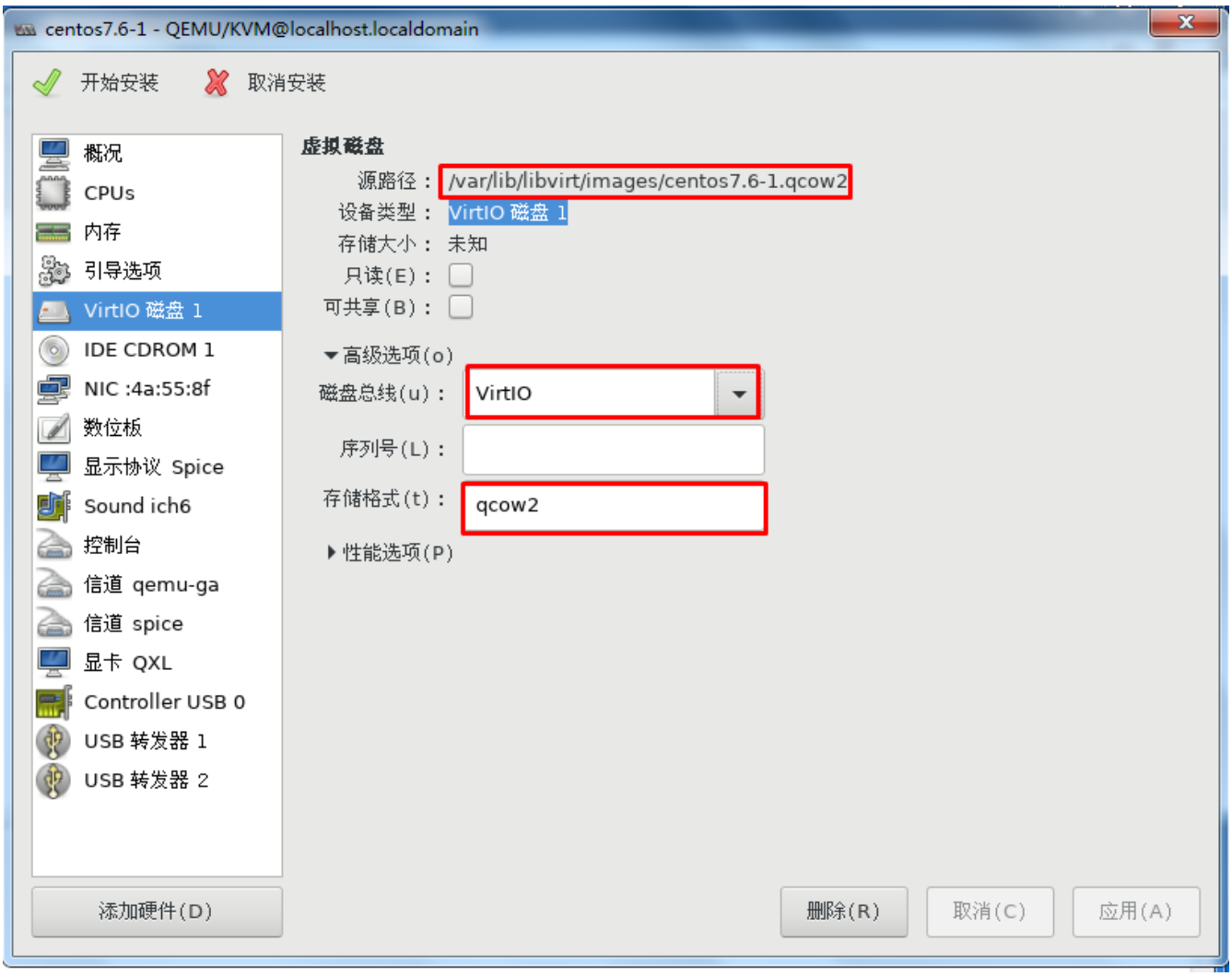

新建虚拟机@localhost.localdomain
生成新虚拟机
5 的步骤 3
选择内存和 CPU 设置
内存(M)，RAM：
1024
主机中最多有 4095 MiB 可用
CPU：
1
最多有 2 可用
取消(C)
后退(B)
前进(F)

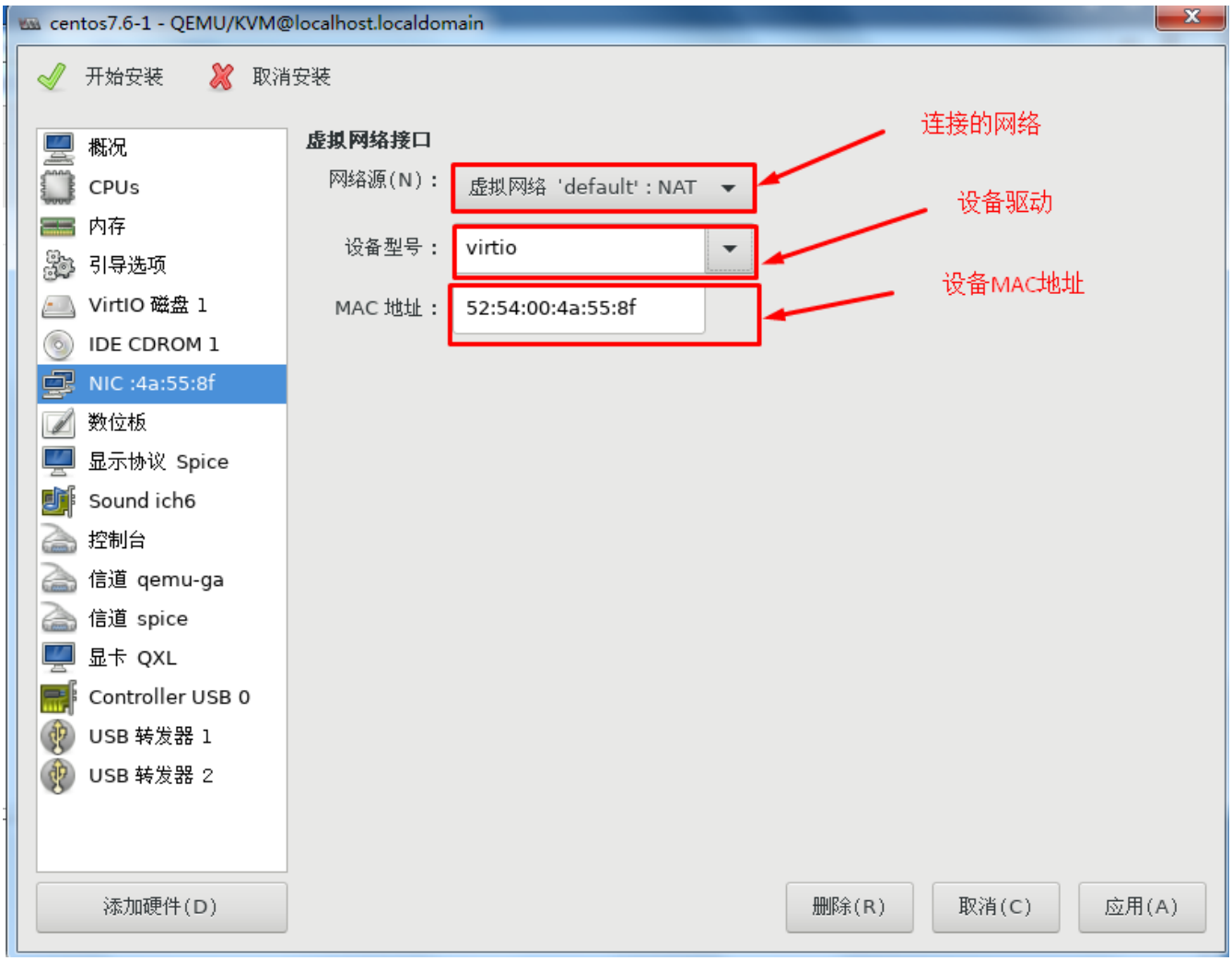

新建虚拟机@localhost.localdomain
生成新虚拟机
5 的步骤 4
为虚拟机启用存储(E)
为虚拟机创建磁盘镜像(R)
10.0
GiB
31.7 GiB available in the default location
选择或创建自定义存储(S)
管理(M)...
取消(C)
后退(B)
前进(F)

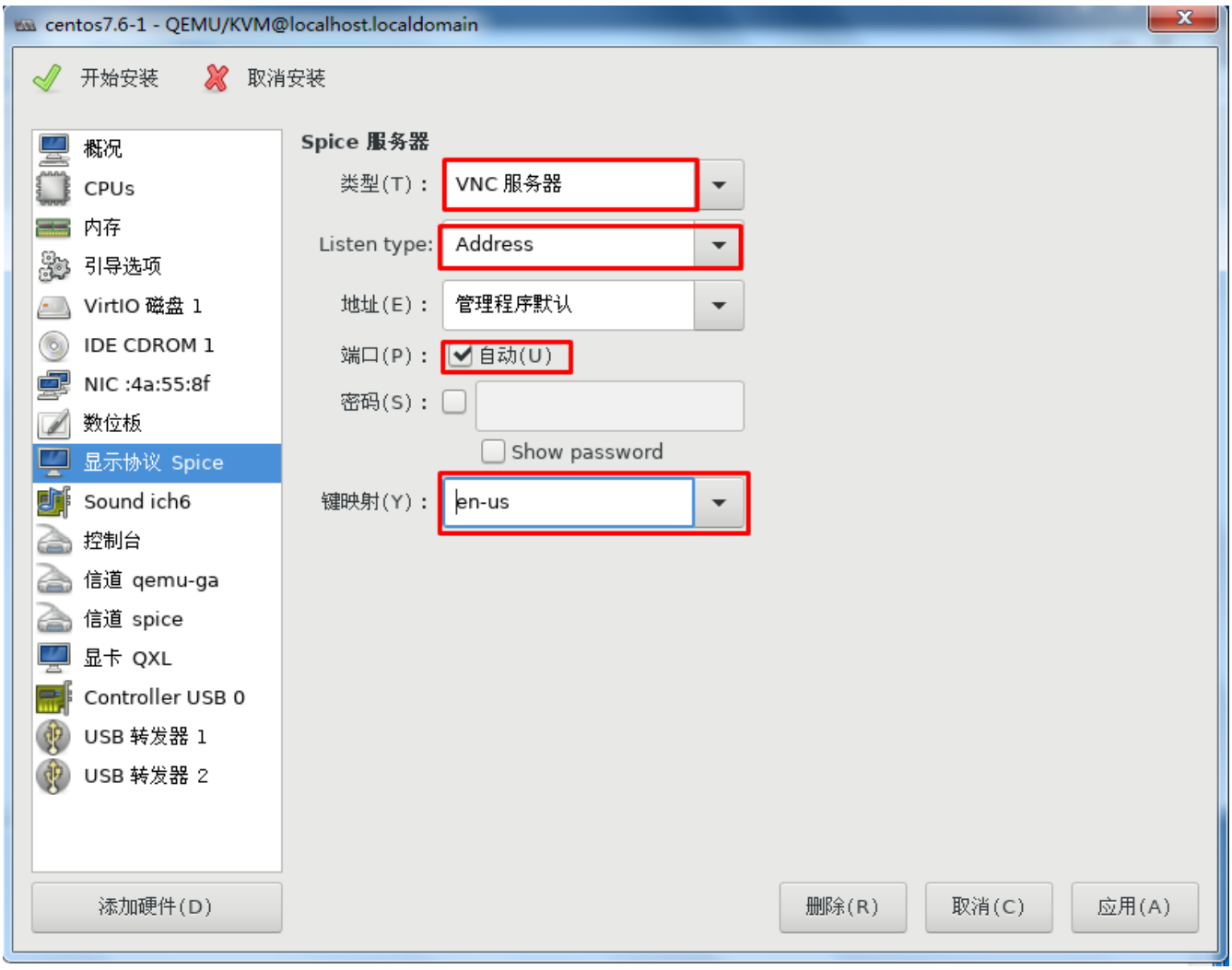

新建虚拟机@localhost.localdomain
生成新虚拟机
5 的步骤 5
准备开始安装
名称(N)
centos7.6-1
操作系统： CentOS 7.0
安装： 本地光驱/映像
内存： 1024 MiB
CPU： 1
存储： 10.0 GiB ···virt/images/centos7.6-1.qcow2
在安装前自定义配置(U)
选择网络(E)
连接virbr0
虚拟网络 'default' : NAT
取消(C)
后退(B)
完成(F)

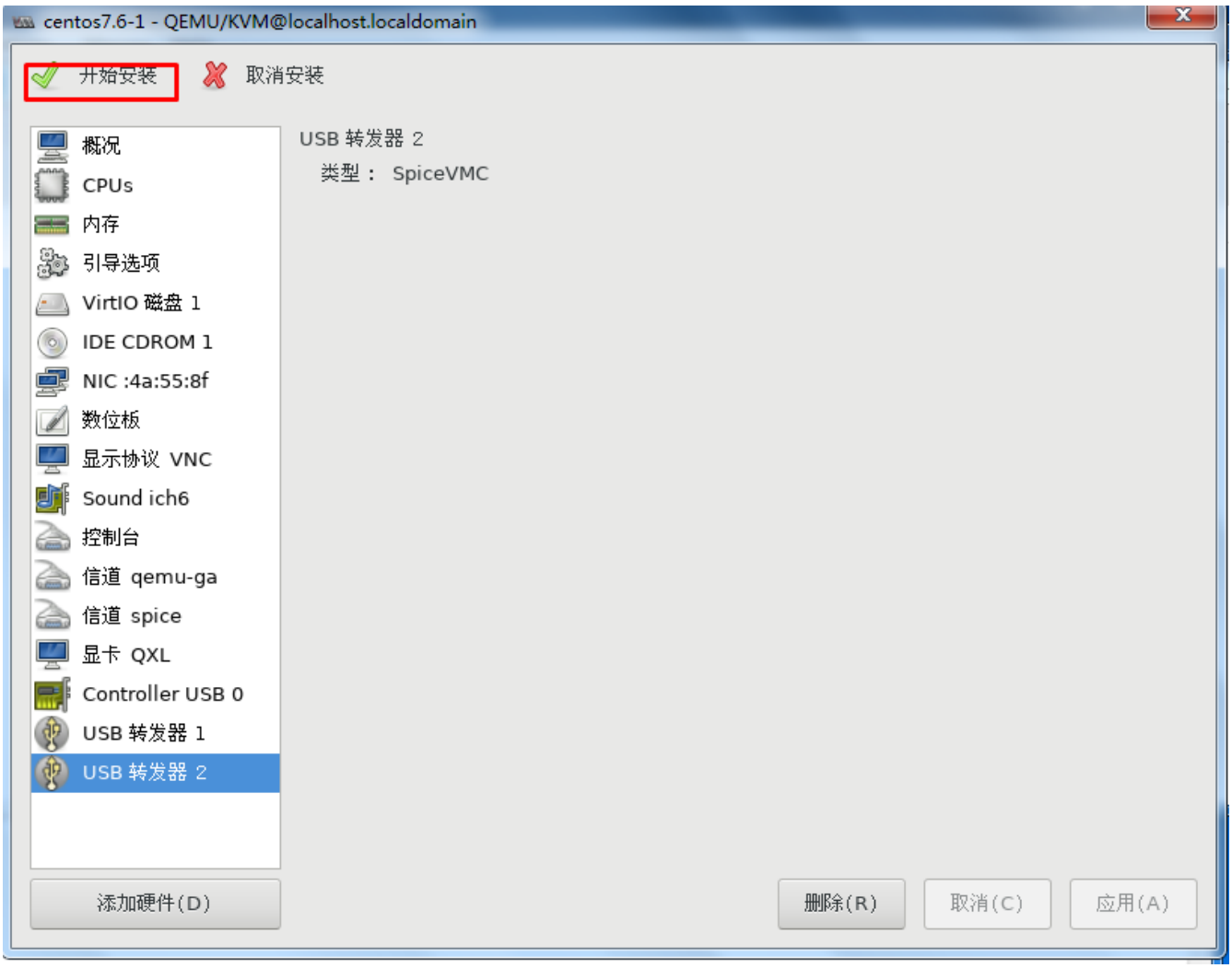

centos7.6-1 - QEMU/KVM@localhost.localdomain
开始安装
取消安装
概况
CPUs
内存
引导选项
VirtIO 磁盘 1
IDE CDROM 1
NIC :4a:55:8f
数位板
显示协议 Spice
Sound ich6
控制台
信道 qemu-ga
信道 spice
显卡 QXL
Controller USB 0
USB 转发器 1
USB 转发器 2
添加硬件(D)
CPU
本地主机 CPU： 2
当前分配(I)： 1
最大分配(X)： 2
CPU热添加
配置(R)
复制主机 CPU 配置
型号(O)： Broadwell-noTSX
拓扑(P)
手动设置 CPU 拓扑(Y)
套接字(T)： 2
核心(E)： 1
线程(S)： 1
取消(C)
应用(A)

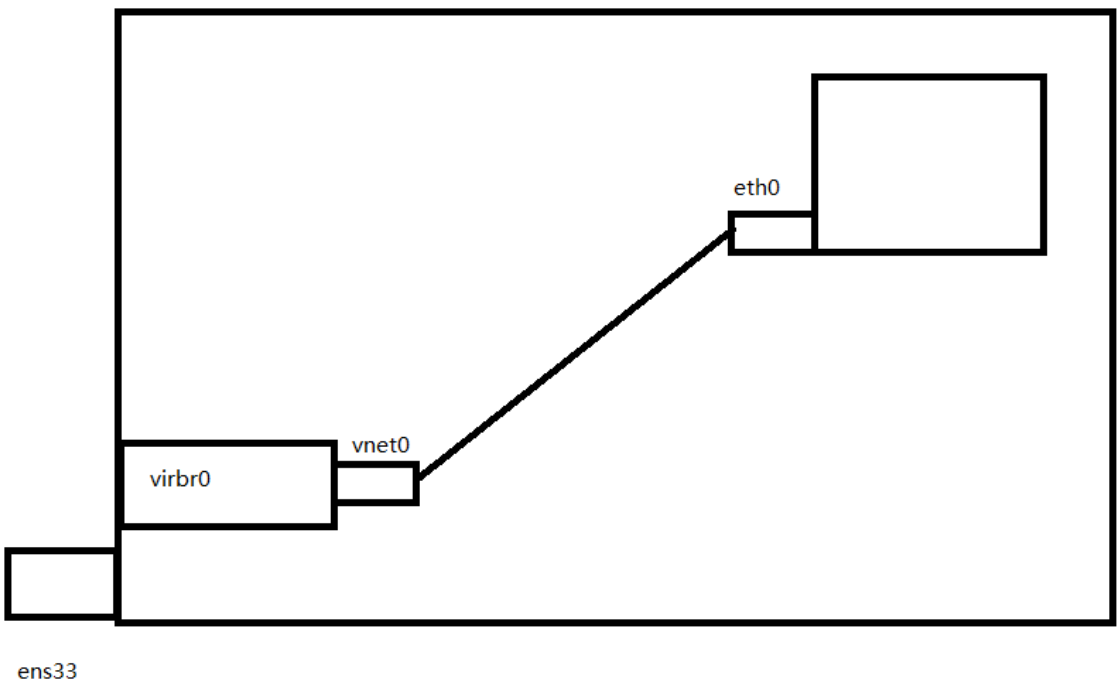

centos7.6-1 - QEMU/KVM@localhost.localdomain
开始安装
取消安装
概况
CPUs
内存
引导选项
VirtIO 磁盘 1
IDE CDROM 1
NIC :4a:55:8f
数位板
显示协议 Spice
Sound ich6
控制台
信道 qemu-ga
信道 spice
显卡 QXL
Controller USB 0
USB 转发器 1
USB 转发器 2
内存
总主机内存： 4095 MiB
当前分配(I)： 1024 MiB
最大分配(X)： 2048 MiB
为了实现内存气球
添加硬件(D)
取消(C)
应用(A)


centos7.6-1 - QEMU/KVM@localhost.localdomain
开始安装
取消安装
概况
CPUs
内存
引导选项
VirtIO 磁盘 1
IDE CDROM 1
NIC :4a:55:8f
数位板
显示协议 Spice
Sound ich6
控制台
信道 qemu-ga
信道 spice
显卡 QXL
Controller USB 0
USB 转发器 1
USB 转发器 2
添加硬件(D)
虚拟磁盘
源路径：
/var/lib/libvirt/images/centos7.6-1.qcow2
设备类型：
VirtIO 磁盘 1
存储大小：
未知
只读(E)：
可共享(B)：
高级选项(o)
磁盘总线(u)：
VirtIO
序列号(L)：
存储格式(t)：
qcow2
性能选项(P)
删除(R)
取消(C)
应用(A)


centos7.6-1 - QEMU/KVM@localhost.localdomain
开始安装
取消安装
概况
CPUs
内存
引导选项
VirtIO 磁盘 1
IDE CDROM 1
NIC :4a:55:8f
数位板
显示协议 Spice
Sound ich6
控制台
信道 qemu-ga
信道 spice
显卡 QXL
Controller USB 0
USB 转发器 1
USB 转发器 2
添加硬件(D)
虚拟网络接口
网络源(N)：
虚拟网络 'default' : NAT
设备型号：
virtio
MAC 地址：
52:54:00:4a:55:8f
连接的网络
设备驱动
设备MAC地址
删除(R)
取消(C)
应用(A)

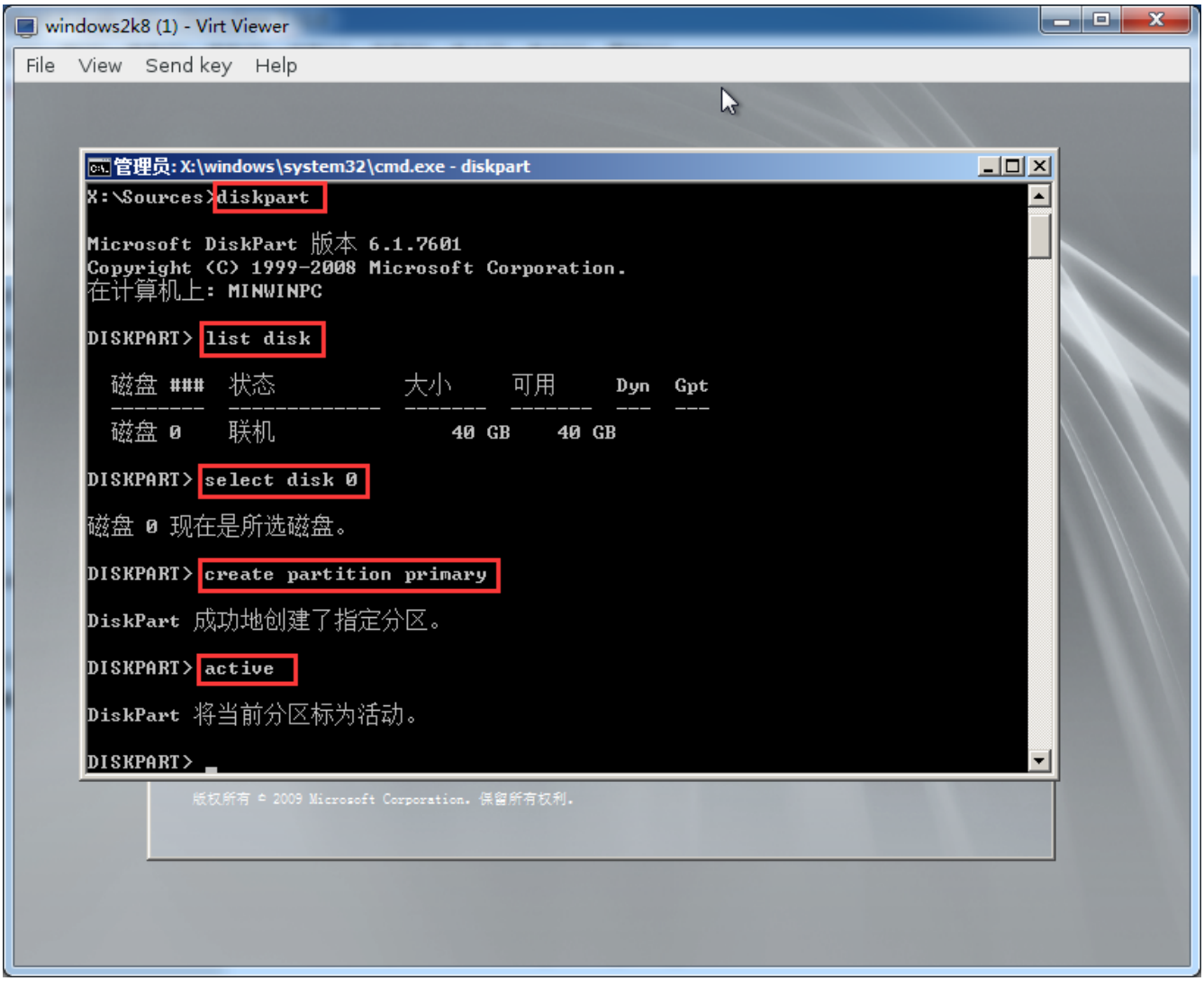

centos7.6-1 - QEMU/KVM@localhost.localdomain
开始安装
取消安装
概况
CPUs
内存
引导选项
VirtIO 磁盘 1
IDE CDROM 1
NIC :4a:55:8f
数位板
显示协议 Spice
Sound ich6
控制台
信道 qemu-ga
信道 spice
显卡 QXL
Controller USB 0
USB 转发器 1
USB 转发器 2
Spice 服务器
类型(T)：
VNC 服务器
Listen type:
Address
地址(E)：
管理程序默认
端口(P)：
自动(U)
密码(S)：
Show password
键映射(Y)：
en-us
添加硬件(D)
删除(R)
取消(C)
应用(A)


centos7.6-1 - QEMU/KVM@localhost.localdomain
开始安装
取消安装
概况
CPUs
内存
引导选项
VirtIO 磁盘 1
IDE CDROM 1
NIC :4a:55:8f
数位板
显示协议 VNC
Sound ich6
控制台
信道 qemu-ga
信道 spice
显卡 QXL
Controller USB 0
USB 转发器 1
USB 转发器 2
USB 转发器 2
类型： SpiceVMC
添加硬件(D)
删除(R)
取消(C)
应用(A)


eth0
vnet0
virbr0
ens33

1.2 windows server 2008 r2主机安装

```
shift+f10
```



2 virt-install

案例1

```
[root@localhost ~]# virt-install \
        --name=smartgo1 \
        --graphics vnc,listen=0.0.0.0,port=5920,keymap=en_us \
        --ram=512 \
        --vcpus=1 \
        --arch=x86_64 \
        --os-type=linux \
        --os-variant=rhel7 \
        --disk path=/var/lib/libvirt/images/smartgo1.img,size=8,format=qcow2 \
        --bridge=virbr0 \
        --cdrom=/root/CentOS-7-x86_64-DVD-1804.iso
```

> --location=ftp://{IP}/rhel7.5
>
> --extra-args="ks=ftp://{IP}/rhel7.5.ks"

案例2

```
[root@localhost ~]# virt-install \
        --name=smartgo2 \
        --graphics vnc,listen=0.0.0.0,port=5930,keymap=en_us \
        --noautoconsole \
        --ram=512 \
        --vcpus=1 \
        --arch=x86_64 \
        --os-type=linux \
        --os-variant=rhel7 \
        --disk path=/var/lib/libvirt/images/smartgo2.img,size=8,format=qcow2 \
        --bridge=virbr0 \
        --cdrom=/root/CentOS-7-x86_64-DVD-1804.iso
```

案例3

```
[root@localhost ~]# virt-install \
        --nographics \
        --name=smartgo3 \
        --ram=512 \
        --vcpus=1 \
        --arch=x86_64 \
        --os-type=linux \
        --os-variant=rhel7 \
        --disk path=/var/lib/libvirt/images/smartgo3.img,size=8,format=qcow2 \
        --bridge=virbr0 \
        --cdrom=/root/CentOS-7-x86_64-DVD-1804.iso
```

案例4

```
[root@localhost ~]# virt-install \
        --name=windows2k8 \
        --graphics vnc,listen=0.0.0.0,port=5940,keymap=en_us \
        --ram=1024 \
        --vcpus=1 \
        --disk path=/guests_images/windows2k8.img,size=50,format=qcow2 \
        --bridge=virbr0 \
        --cdrom=/root/Windows_Server_2008_R2_VL_x64_CN_2018.04.iso
```

案例5

```
[root@localhost ~]# virt-install \
        --name=centos7  \
        --memory 512,maxmemory=1024 \
        --vcpus 1,maxvcpus=4 \
        --os-type=linux \
        --cdrom=/root/CentOS-7-x86_64-DVD-1804.iso \
        --disk path=/var/lib/libvirt/images/centos7u2.img,size=50,format=qcow2 \
        --network bridge=virbr0 \
        --vnc \
        --vncport=5950 \
        --vnclisten=0.0.0.0 \
        --autostart
```

案例6

```
[root@localhost ~]# virt-install \
        --name=windows2008  \
        --memory 1024,maxmemory=4096 \
        --vcpus 1,maxvcpus=4 \
        --os-type=windows \
        --cdrom=/root/Windows_Server_2008_R2_VL_x64_CN_2018.04.iso \
        --disk path=/usr/share/virtio-win/virtio-win_amd64.vfd,device=floppy \
        --disk path=/home/images2/windows2008.img,format=qcow2,size=40,bus=virtio \
        --network bridge=virbr0,model=virtio \
        --vnc \
        --vncport=5950 \
        --vnclisten=0.0.0.0 \
        --autostart
```

> 驱动下载：https://fedorapeople.org/groups/virt/virtio-win/direct-downloads/archive-virtio/

四、KVM虚拟机组成文件

学习目标

- [ ] 能够使用virsh命令查看虚拟机
- [ ] 能够找到虚拟机配置文件所在位置
- [ ] 能够找到虚拟机磁盘文件所在位置

```
虚拟机(guest,vm,domain,instance) = 磁盘文件image(Linux or Windows) +  配置文件（虚拟机名，2 VCPU，512M...)
```

1.KVM虚拟机查看方法

```
[root@localhost ~]# virsh list --all
```

2.虚拟机配置文件XML

```
[root@localhost ~]#ls /etc/libvirt/qemu
[root@localhost ~]#virsh edit startgo_01 #编辑
[root@localhost ~]#virsh dumpxml startgo_01 > new_smartgo_01 #备份

```

3.设定vm自动运行

```
[root@localhost ~]#virsh autostart smtargo_01
[root@localhost ~]#ls /etc/libvirt/qemu/autostart/
[root@localhost ~]#ll /etc/libvirt/qemu/autostart/smartgo_01.xml

```

4.网络配置文件XML

```
[root@localhost ~]#ls /etc/libvirt/qemu/networks/
[root@localhost ~]#ls /etc/libvirt/qemu/networks/autostart/
```

5.存储池配置文件XML

```
[root@localhost ~]#ls /etc/libvirt/storage/
[root@localhost ~]#ls /etc/libvirt/storage/autostart/
```

6.镜像文件（磁盘文件）

```
[root@localhost ~]#ls /var/lib/libvirt/images/
```

五、KVM虚拟机CPU热添加

学习目标

- [ ] 能够了解虚拟机添加CPU的作用及预准备
- [ ] 能够通过virt-manager为虚拟机添加CPU

- [ ] 能够通过virsh命令为虚拟机添加CPU

```
[root@localhost ~]#virsh list
[root@localhost ~]#virsh dominfo smartgo_01

[root@smartgo_01 ~]#lscpu

[root@localhost ~]#virsh setvcpus smartgo_01 4 --live
[root@localhost ~]#virsh dominfo smartgo_01

[root@smartgo_01 ~]#lscpu
```

> windows server 2008 r2添加CPU方法与上述方法一致

六、KVM虚拟机内存气球应用

学习目标

- [ ] 能够了解虚拟机添加内存的作用及预准备
- [ ] 能够通过virt-manager为虚拟机添加内存
- [ ] 能够通过virsh命令为虚拟机添加内存

KVM内存气球技术可以对虚拟机使用的内存按需调节，从而提高内存的利用率。

linux默认就支持内存气球

1.宿主机内存气球配置

```
[root@localhost ~]#virsh dumpxml smartgo_01 | grep memballon -C2
```

2.虚拟机内存气球配置

虚拟机需要安装virtio balloon驱动，内核开启CONFIG_VIRTIO_BALLOON。CentOS6/7默认已安装驱动并开启。

```
[root@smartgo_01 ~]#lspci

```

```
[root@localhost ~]#virsh qemu-monitor-command smartgo_01 --hmp --cmd info balloon #查看内存信息
[root@localhost ~]#virsh qemu-monitor-command smartgo_01 --hmp --cmd  balloon 1024
[root@localhost ~]#virsh qemu-monitor-command smartgo_01 --hmp --cmd info balloon
```

```
[root@smartgo_01 ~]#free -m
```