

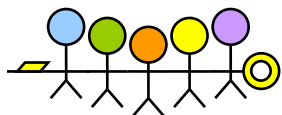
火龙果软件

学习中心 ([uml.org.cn](http://uml.org.cn))

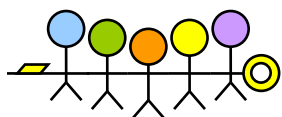
---

# QEMU 仿真测试实战教程

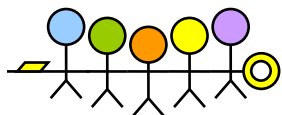
—— 从零入门 · 侧重实操



|                              |    |
|------------------------------|----|
| <b>目 录</b>                   |    |
| <b>一、QEMU 是什么——为什么用仿真做测试</b> | 4  |
| 1.1 什么是 QEMU                 | 4  |
| 1.2 两种仿真模式                   | 4  |
| 1.3 QEMU 在软件测试中的典型用途         | 4  |
| 1.4 版本信息                     | 5  |
| <b>二、安装 QEMU</b>             | 5  |
| 2.1 Ubuntu / Debian          | 5  |
| 2.2 macOS                    | 5  |
| 2.3 Windows                  | 5  |
| 2.4 验证安装                     | 5  |
| <b>三、基本概念与工作流程</b>           | 6  |
| 3.1 主机与客户机                   | 6  |
| 3.2 两种运行模式对比                 | 6  |
| 3.3 磁盘镜像工具 qemu-img          | 6  |
| 3.4 一条启动命令的工作流程              | 6  |
| <b>四、实战一：启动一个最小 Linux 系统</b> | 7  |
| 4.1 获取内核与根文件系统               | 7  |
| 4.2 首次启动                     | 7  |
| 4.3 常用启动参数详解                 | 7  |
| 4.4 退出与快照                    | 8  |
| <b>五、实战二：磁盘镜像与文件系统</b>       | 8  |
| 5.1 创建与扩容镜像                  | 8  |
| 5.2 挂载并准备文件系统                | 8  |
| 5.3 启动并安装完整系统                | 9  |
| <b>六、实战三：网络配置</b>            | 9  |
| 6.1 用户态网络 (user / slirp)     | 9  |
| 6.2 端口转发                     | 10 |
| 6.3 桥接网络 (tap)               | 10 |



|                             |    |
|-----------------------------|----|
| 七、实战四：多架构与嵌入式仿真 .....       | 10 |
| 7.1 ARM / AArch64 仿真 .....  | 10 |
| 7.2 RISC-V 仿真 .....         | 11 |
| 7.3 运行裸机/固件程序 .....         | 11 |
| 7.4 常用嵌入式开发板仿真 .....        | 11 |
| 八、实战五：GDB 远程调试 .....        | 11 |
| 8.1 开启调试端口 .....            | 11 |
| 8.2 连接 GDB .....            | 12 |
| 8.3 常用调试命令 .....            | 12 |
| 九、自动化测试与 CI 集成 .....        | 12 |
| 9.1 无头与批处理模式 .....          | 12 |
| 9.2 脚本化冒烟测试示例 .....         | 13 |
| 9.3 在 CI 中使用 QEMU 的要点 ..... | 13 |
| 十、常见问题排查 .....              | 13 |
| 10.1 命令找不到 .....            | 13 |
| 10.2 启动后黑屏或无输出 .....        | 13 |
| 10.3 性能较慢 .....             | 14 |
| 10.4 KVM 加速不可用 .....        | 14 |
| 十一、学习资源 .....               | 14 |



## 一、QEMU 是什么——为什么用仿真做测试

### 1.1 什么是 QEMU

QEMU (Quick Emulator) 是一个开源、通用的机器模拟器与虚拟化器，能够模拟完整的计算机系统 (CPU、内存、磁盘、网卡、串口等外设)，让一套操作系统或软件运行在不同于其原本设计目标的主机架构上。它也是当前嵌入式开发、内核调试与软件测试领域最常用的仿真工具之一。

### 1.2 两种仿真模式

QEMU 提供两种主要运行模式，二者在测试中各有用途：

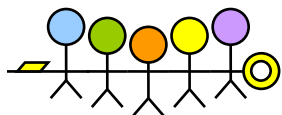
| 模式    | 命令行前缀                                 | 作用                           | 典型测试场景                   |
|-------|---------------------------------------|------------------------------|--------------------------|
| 全系统仿真 | <code>qemu-system-&lt;arch&gt;</code> | 模拟整台机器 (CPU+外设)，可启动完整操作系统    | 启动 Linux 内核、调试驱动、运行完整发行版 |
| 用户态仿真 | <code>qemu-&lt;arch&gt;</code>        | 仅模拟目标架构 CPU，直接在主机上运行单个目标架构程序 | 交叉编译程序的功能测试、运行测试二进制      |

对于“仿真测试”任务，绝大多数场景 (启动内核、跑完整系统、做 CI 回归) 使用的是全系统仿真模式 `qemu-system-*`；而 `qemu-*` (用户态模式) 适合快速运行单个交叉编译的测试用例。

### 1.3 QEMU 在软件测试中的典型用途

使用 QEMU 做仿真测试，能解决“没有真实硬件”和“测试不可重复”两大痛点，常见用途包括：

| 用途        | 说明                                    |
|-----------|---------------------------------------|
| 无硬件环境测试   | 在没有目标开发板时，即可运行并验证固件、内核或应用逻辑           |
| CI 自动化回归  | 在流水线中拉起虚拟机，自动执行冒烟测试与回归测试，结果可重复        |
| 内核/固件调试   | 配合 GDB 实现源码级断点、单步、查看寄存器与内存            |
| 多架构兼容验证   | 同一套代码在 x86、ARM、RISC-V 等不同架构上验证，无需多块硬件 |
| 故障注入与恢复测试 | 模拟断电、网卡异常、磁盘故障等场景，验证系统容错能力            |



## 1.4 版本信息

QEMU 采用持续快速的版本发布节奏。截至本文写作时，QEMU 官方最新稳定版本已进入 11.x（例如 2025 年发布的 10.1.0、10.2.0，以及后续的 11.x 系列）。命令和参数在各版本间保持高度向后兼容，本文示例基于 QEMU 8.x~11.x 均可运行。

## 二、安装 QEMU

QEMU 支持 Linux、macOS、Windows 三大平台。本教程示例以 Ubuntu/Debian 为主，同时给出其他平台的安装方式。

### 2.1 Ubuntu / Debian

使用 apt 直接安装所需的仿真器与磁盘镜像工具。按需选择架构对应的包，建议至少安装 qemu-system-x86 和 qemu-utils：

安装命令（Ubuntu/Debian）

```
sudo apt update
sudo apt install -y qemu-system-x86 qemu-system-arm qemu-system-aarch64
qemu-utils
```

说明：qemu-system-\* 提供各架构的全系统仿真器；qemu-utils 提供磁盘镜像管理工具 qemu-img。

### 2.2 macOS

macOS 通过 Homebrew 安装：

```
brew install qemu
```

### 2.3 Windows

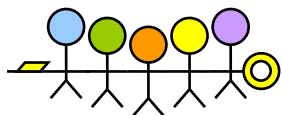
Windows 用户可以从 QEMU 官网下载页（<https://www.qemu.org/download/>）获取 Windows 安装包，按向导安装完成后，将 QEMU 安装目录加入 PATH 环境变量，即可在命令行中使用。

### 2.4 验证安装

安装完成后，运行以下命令确认版本与可用仿真器：

验证安装

```
qemu-system-x86_64 --version
qemu-img --version
```



```
ls /usr/bin/qemu-system-* # 列出已安装的各架构仿真器
```

## 三、基本概念与工作流程

### 3.1 主机与客户机

主机（host）是运行 QEMU 的当前机器；客户机（guest）是被仿真的目标系统。仿真测试的实质是：在主机上用软件模拟出目标机器环境，再把被测系统（内核、固件或应用）加载进去运行。

### 3.2 两种运行模式对比

| 对比项   | 全系统仿真（system）    | 用户态仿真（user）             |
|-------|------------------|-------------------------|
| 模拟对象  | 完整机器（CPU + 外设）   | 仅 CPU 指令集，复用主机操作系统      |
| 可运行内容 | 完整操作系统           | 单个用户态程序                 |
| 典型命令  | qemu-system-arm  | qemu-arm / qemu-riscv64 |
| 测试用途  | 内核启动、驱动、CI 系统级测试 | 交叉编译程序快速功能测试            |

### 3.3 磁盘镜像工具 qemu-img

qemu-img 用于创建、查看和转换虚拟机磁盘镜像。最常用的格式是 qcow2（Copy-On-Write，按需增长，占用空间小）。

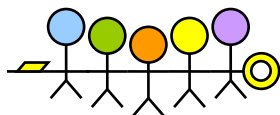
qemu-img 基本用法

```
# 创建一个 10GB 的 qcow2 镜像
qemu-img create -f qcow2 test-disk.qcow2 10G

# 查看镜像信息
qemu-img info test-disk.qcow2
```

### 3.4 一条启动命令的工作流程

一次完整的 QEMU 启动流程为：加载磁盘镜像或内核文件 → 分配内存与 CPU → 初始化模拟外设（网卡、串口、显卡） → 按启动参数引导客户机系统 → 客户机系统运行并输出到主机。整个过程由一行命令行参数驱动，理解这些参数是掌握 QEMU 仿真的关键。



## 四、实战一：启动一个最小 Linux 系统

本章以启动一个精简 Linux 内核为例，掌握 QEMU 最核心的启动参数。这是后续所有仿真测试的基础。

### 4.1 获取内核与根文件系统

可以下载官方构建的最小内核镜像和根文件系统，也可以使用自己交叉编译的内核。下面以较通用的方式获取测试素材：

获取素材

```
# 下载 Linux 内核二进制镜像 (x86_64)
wget https://mirrors.edge.kernel.org/pub/linux/kernel/v6.x/linux-6.x.tar.xz

# 或使用发行版提供的内核镜像 / 现成的测试镜像 (如 buildroot 生成的 Image)
# 本教程为演示，也可直接使用发行版 vmlinuz 加 initramfs
```

### 4.2 首次启动

最精简的启动命令如下。它会分配 1GB 内存、1 个 CPU，不启动图形界面，将客户机串口重定向到当前终端：

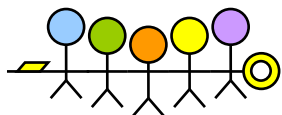
最小启动命令

```
qemu-system-x86_64 \
-m 1024 \
-smp 1 \
-kernel bzImage \
-initrd initramfs.cpio.gz \
-append "console=ttyS0 rdinit=/bin/sh" \
-nographic
```

参数含义：-kernel 指定内核镜像；-initrd 指定初始内存盘；-append 向内核传递命令行参数（console=ttyS0 让内核把控制台输出到串口，配合 -nographic 在当前终端显示）；-nographic 表示不使用图形窗口，将客户机的串口与标准输入输出相连。

### 4.3 常用启动参数详解

| 参数             | 作用              | 示例                      |
|----------------|-----------------|-------------------------|
| -m <MB>        | 分配客户机内存大小       | -m 2048                 |
| -smp <n>       | 设置客户机 CPU 核数    | -smp 4                  |
| -kernel <file> | 直接加载内核镜像（免磁盘启动） | -kernel bzImage         |
| -initrd <file> | 指定初始 RAM 盘      | -initrd initrd.gz       |
| -append <str>  | 向内核追加命令行参数      | -append "console=ttyS0" |



| 参数                | 作用                | 示例                                  |
|-------------------|-------------------|-------------------------------------|
| -drive <opt>      | 挂载磁盘/光驱设备         | -drive file=disk.qcow2,format=qcow2 |
| -netdev / -device | 配置网络后端与客户机网卡      | -netdev user,id=n1                  |
| -nographic        | 无图形窗口，串口重定向到终端    | -nographic                          |
| -display none     | 禁用显示（适合 CI 无头运行）  | -display none                       |
| -accel kvm        | 启用 KVM 硬件加速（大幅提速） | -accel kvm                          |

## 4.4 退出与快照

在 -nographic 模式下，按组合键 Ctrl+A 后按 X 可退出 QEMU。qcow2 镜像支持快照，便于测试中保存/恢复现场：

### 快照操作

```
# 在客户机内部创建快照（可选，也可用 monitor）
# 主机侧 monitor：按 Ctrl+A 再按 C 进入 QEMU monitor

# monitor 中操作
savevm snap1      # 保存快照
loadvm snap1      # 恢复快照
quit              # 退出
```

## 五、实战二：磁盘镜像与文件系统

实际测试往往需要在客户机内安装完整的发行版或自建文件系统。本章介绍磁盘镜像的创建、挂载与系统安装。

### 5.1 创建与扩容镜像

#### 创建/扩容镜像

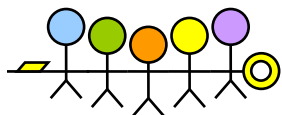
```
# 创建 qcow2 镜像（10G，按需占用）
qemu-img create -f qcow2 disk.qcow2 10G

# 扩容到 20G
qemu-img resize disk.qcow2 20G
```

### 5.2 挂载并准备文件系统

qcow2 不能直接由主机挂载，需先转换为 raw 格式或通过客户机内部挂载。若需在主机侧检查内容，可转换为 raw：

#### 挂载镜像



```
# 转换为 raw (体积与分配大小一致)
qemu-img convert -f qcow2 -O raw disk.qcow2 disk.raw

# 用 loop 设备挂载 (需 root)
sudo losetup -fP disk.raw
sudo mount /dev/loop0p1 /mnt/guestfs
```

## 5.3 启动并安装完整系统

使用发行版提供的 cloud 镜像或安装 ISO 启动完整系统，是搭建测试环境的常见方式。以 Ubuntu cloud 镜像为例：

启动完整云镜像

```
# 下载 Ubuntu 云镜像 (qcow2 格式)
wget https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-
amd64.img

# 启动 (云镜像需指定 UEFI/BIOS 与串口控制台)
qemu-system-x86_64 \
  -m 2048 -smp 2 \
  -drive file=noble-server-cloudimg-amd64.img,format=qcow2 \
  -netdev user,id=n1,hostfwd=tcp::2222-:22 \
  -device virtio-net-pci,netdev=n1 \
  -nographic
```

通过参数 `hostfwd=tcp::2222-:22` 将客户机的 22 端口映射到主机的 2222 端口，这样主机可用 ssh 连接客户机进行测试，非常便于 CI 与自动化脚本。

## 六、实战三：网络配置

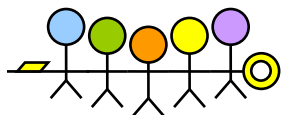
网络是仿真测试的关键一环。QEMU 默认提供用户态网络 (user/slirp)，无需特权即可让客户机联网；需要更复杂网络拓扑时可使用 tap 桥接。

### 6.1 用户态网络 (user / slirp)

用户态网络是默认且最简单的方式，客户机通过主机转发访问外网，无需 root 权限。

用户态网络

```
qemu-system-x86_64 \
  -netdev user,id=net0 \
  -device virtio-net-pci,netdev=net0 \
  ...
```



## 6.2 端口转发

通过 hostfwd 将客户机端口映射到主机，便于 ssh、调试端口等访问：

端口转发

```
-netdev user,id=net0,hostfwd=tcp::2222-:22,hostfwd=tcp::1234-:1234
```

## 6.3 桥接网络（tap）

当需要客户机作为网络中独立节点（接收外部主动连接）时，使用 tap 桥接。此方式需要 root 权限并预先配置网桥：

tap 桥接

```
# 创建 tap 设备并桥接
sudo ip tuntap add dev tap0 mode tap
sudo ip link set tap0 up
# 将 tap0 加入网桥 br0（需已存在）
sudo ip link set tap0 master br0

qemu-system-x86_64 \
  -netdev tap,id=n1,ifname=tap0,script=no,downscript=no \
  -device virtio-net-pci,netdev=n1 \
  ...
```

# 七、实战四：多架构与嵌入式仿真

QEMU 最大的价值之一是跨架构仿真：无需 ARM 或 RISC-V 硬件，即可运行和测试对应架构的程序与内核。本章演示交叉编译并在 QEMU 中运行。

## 7.1 ARM / AArch64 仿真

ARM 仿真

```
# 安装 ARM 工具链
sudo apt install -y gcc-arm-linux-gnueabi g++-arm-linux-gnueabi qemu-system-arm

# 用用户态仿真运行一个 ARM 程序
arm-linux-gnueabi-gcc -static -o hello hello.c
qemu-arm ./hello

# 或用全系统仿真启动 ARM 内核
qemu-system-arm -M virt -m 512 \
  -kernel zImage -dtb virt.dtb \
  -append "console=ttyAMA0" -nographic
```

## 7.2 RISC-V 仿真

### RISC-V 仿真

```
# 安装 RISC-V 工具链
sudo apt install -y gcc-riscv64-linux-gnu qemu-system-misc

# 用户态运行 RISC-V 程序
riscv64-linux-gnu-gcc -static -o hello hello.c
qemu-riscv64 ./hello
```

## 7.3 运行裸机/固件程序

对于无操作系统的裸机固件，可用 `-kernel` 直接加载 ELF 固件，或用 `-device loader` 加载到指定地址。嵌入式仿真测试通常关注串口输出是否正常、外设寄存器读写是否正确、中断是否触发。

## 7.4 常用嵌入式开发板仿真

| 机器名 (-M)   | 架构            | 典型用途             |
|------------|---------------|------------------|
| virt       | ARM/RISC-V 通用 | 最常用的通用开发板仿真，外设齐全 |
| mps2-an385 | Cortex-M3     | Cortex-M 系列固件测试  |
| raspi3b    | AArch64       | 树莓派 3 仿真         |
| sifive_u   | RISC-V        | RISC-V 64 位开发板仿真 |

## 八、实战五：GDB 远程调试

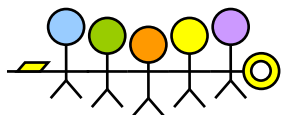
QEMU 内置 GDB 服务器支持，可实现源码级调试，这是内核与固件测试最核心的能力。

### 8.1 开启调试端口

启动 QEMU 时加 `-s`（等价于 `-gdb tcp::1234`）和 `-S`（启动后暂停 CPU，等待调试器连接）：

#### QEMU 开启调试端口

```
qemu-system-x86_64 -m 1024 \
  -kernel bzImage -initrd initrd.gz \
  -append "console=ttyS0 nokaslr" \
  -S -s -nographic
# 启动后 QEMU 暂停，等待 gdb 连接
```



## 8.2 连接 GDB

连接 GDB

```
# 另开终端，使用配套架构的 gdb（此处 x86_64）
gdb vmlinux

(gdb) target remote localhost:1234
(gdb) break start_kernel
(gdb) continue
```

## 8.3 常用调试命令

| 命令                           | 作用                |
|------------------------------|-------------------|
| target remote localhost:1234 | 连接 QEMU 的 GDB 服务器 |
| break <函数/行>                 | 下断点               |
| continue / c                 | 继续执行              |
| stepi / si                   | 单步执行指令            |
| info registers               | 查看寄存器             |
| x/16wx <地址>                  | 查看内存              |
| bt                           | 查看调用栈             |

调试内核时注意加 `nokaslr` 内核参数，避免内核地址随机化导致断点地址偏移。对于 ARM/RISC-V，只需使用对应架构的 `gdb`（如 `aarch64-linux-gnu-gdb`、`riscv64-linux-gnu-gdb`）并连接相同端口即可。

## 九、自动化测试与 CI 集成

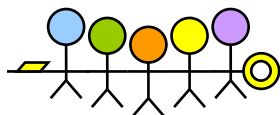
QEMU 的无头（无显示）运行能力和脚本退出机制，使其非常适合接入自动化测试与持续集成（CI）。

### 9.1 无头与批处理模式

在 CI 环境中，通常使用 `-display none` 或 `-nographic`，并通过内核参数让系统自动执行测试脚本后关机，避免人工交互：

无头批处理启动

```
qemu-system-x86_64 \
  -display none -serial stdio \
  -kernel bzImage -initrd test-initrd.cpio.gz \
  -append "console=ttyS0 rdinit=/test.sh" \
  -no-reboot
```



## 9.2 脚本化冒烟测试示例

用脚本拉起 QEMU、等待串口输出预期标志、然后关闭并依据输出判定通过/失败：

冒烟测试脚本

```
#!/bin/bash
# smoke_test.sh
qemu-system-x86_64 -display none -serial file:serial.log \
  -kernel bzImage -initrd initrd.cpio.gz \
  -append "console=ttyS0 rdinit=/test.sh" &
QEMU_PID=$!
# 等待串口日志出现成功标志（超时 60s）
for i in $(seq 1 60); do
  if grep -q "TEST PASSED" serial.log; then
    echo "测试通过"; kill $QEMU_PID; exit 0
  fi
  sleep 1
done
echo "测试超时失败"; kill $QEMU_PID; exit 1
```

## 9.3 在 CI 中使用 QEMU 的要点

| 要点               | 说明                                  |
|------------------|-------------------------------------|
| 使用 -display none | 避免依赖图形环境，保证在无头 CI runner 上可运行       |
| -no-reboot       | 客户机异常重启时直接退出，避免无限循环占用资源             |
| 固定串口输出           | 用 -serial file: 捕获日志作为测试断言依据        |
| 考虑 KVM 加速        | 若 CI 支持，用 -accel kvm 显著提速；不支持则用 TCG |
| 限制超时             | 脚本内设置超时，防止测试卡死拖垮流水线                 |

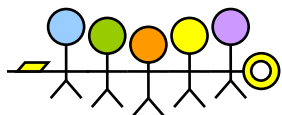
# 十、常见问题排查

## 10.1 命令找不到

确认已安装对应架构的 qemu-system-\* 包，并已将安装目录加入 PATH。用 qemu-system-x86\_64 --version 验证。

## 10.2 启动后黑屏或无输出

通常是串口控制台参数不对。确保内核参数包含 console=ttyS0（x86）或 console=ttyAMA0（ARM），并使用 -nographic 或 -serial stdio 将串口重定向到终端。



### 10.3 性能较慢

默认使用 TCG 软件模拟，性能有限。若主机 CPU 支持虚拟化且架构相同，加 `-accel kvm` (x86) 或 `-accel hvf` (macOS) 启用硬件加速；用户态模式也可用。减少 `-smp` 核数超过实际需要也会拖慢整体。

### 10.4 KVM 加速不可用

错误提示 “`kvm: no hardware support`” 表示当前环境（如无嵌套虚拟化的云主机或容器）不支持 KVM。此时去掉 `-accel kvm`，回退到 TCG 即可运行，只是性能下降。

## 十一、学习资源

以下为官方权威资源，可进一步深入学习：

| 资源       | 地址                                                                                                              | 说明                |
|----------|-----------------------------------------------------------------------------------------------------------------|-------------------|
| QEMU 官网  | <a href="https://www.qemu.org/">https://www.qemu.org/</a>                                                       | 项目主页、最新版本与发布说明    |
| QEMU 下载页 | <a href="https://www.qemu.org/download/">https://www.qemu.org/download/</a>                                     | 各平台安装包与源码构建说明     |
| 系统仿真手册   | <a href="https://www.qemu.org/docs/master/system/">https://www.qemu.org/docs/master/system/</a>                 | 全系统仿真完整文档         |
| GDB 调试文档 | <a href="https://www.qemu.org/docs/master/system/gdb.html">https://www.qemu.org/docs/master/system/gdb.html</a> | QEMU 与 GDB 远程调试指南 |
| 用户模式仿真   | <a href="https://www.qemu.org/docs/master/user/">https://www.qemu.org/docs/master/user/</a>                     | 用户态仿真说明           |

建议学习路径：先掌握本章启动参数与磁盘镜像，再上手 GDB 调试，最后接入 CI 自动化。结合自己的被测系统反复练习，即可快速上手 QEMU 仿真测试。