Linuc level1

李思怡 | 今天修改

1.🔷 SSH 连接命令的基本格式：

ssh [用户名@]服务器地址

这是最标准的写法。

举个例子：

- 我要以 user1 这个身份登录到名叫 server 的主机上

ssh user1@server

🔷 注意：如果你省略了"用户名@"这一部分，会发生什么？

比如你只写：

ssh server

那 SSH 会自动使用你当前电脑的用户名去尝试登录 server。

🧩 Step 2：题目解法说明

❓题目问：

> 想要以 user1 身份连接到 server 主机上，用哪个命令？
>
> サーバ「server」にユーザ「user1」としてSSH接続したい。適切なコマンドはどれか。

看选项：

选项	错误原因
ssh server@user1	把用户名和主机名反了
ssh server\user1	语法不对（Linux 里没有用 \ 分隔用户和主机）
ssh server user1	多了个空格，命令写法错误
ssh user1 server	多个参数，语法错误
✅ ssh user1@server	✅ 正确写法，用户名在前，主机名在后

📝 Step 3：总结考点 & 记忆法

✅ 本题考点：

- SSH 连接命令格式是：
- ssh ユーザ名@ホスト名
- 如果你写成其他形式，都会被系统识别为错误的命令。

🧠 记忆口诀：

> **"先说你是谁（user），再说你去哪儿（@server）"**
> 就像你寄信写收件人一样，要先写人，再写地址！

2.🧠 Step 1：从参考知识讲起（从最基础概念开始）

🔷 什么是 ssh-keygen？

ssh-keygen 是 Linux 系统中用来生成一对 SSH 密钥（公钥和私钥）的命令。

你可以把它想象成：

> “为你生成一把钥匙（私钥）和一把锁（公钥）”

- **私钥（private key）**：你自己保管，不能给别人
- **公钥（public key）**：可以交给远程服务器，证明你是“钥匙持有者”

◆ 生成的密钥存放在哪里？

从参考说明中可以看到：

> ssh-keygenコマンド実行時にユーザのホームディレクトリ（ログインディレクトリ）に「.ssh」というディレクトリが作成され、公開鍵と秘密鍵が格納されます。

翻译为中文就是：

> 当你执行 `ssh-keygen` 命令时，系统会在你的**主目录（home目录）下创建一个名为 `.ssh` 的隐藏文件夹（如果没有的话），并把生成的私钥（id_rsa）** 和 **公钥（id_rsa.pub）** 都放进去。

📷 图中显示的执行过程：

staff@client:~$ ssh-keygen
...
Created directory '/home/staff/.ssh'
...
Your identification has been saved in /home/staff/.ssh/id_rsa.
Your public key has been saved in /home/staff/.ssh/id_rsa.pub.

我们可以看到系统：

✅ 创建了 ~/.ssh 文件夹
✅ 把私钥存为 id_rsa
✅ 把公钥存为 id_rsa.pub

◆ 什么是 “~/.ssh”？

这个 .ssh 文件夹是专门用来存储：

- 私钥（id_rsa）
- 公钥（id_rsa.pub）
- 其他SSH相关配置，比如 known_hosts, authorized_keys, config

注意它前面有个“点”，表示这是一个**隐藏目录。**

🧩 Step 2：题目解法说明

❓题目：

> ssh-keygenコマンド実行時にユーザのホームディレクトリに作成され、公開鍵と秘密鍵が格納されるディレクトリはどれか。

意思是：

> 当你执行 `ssh-keygen` 时，生成的密钥对会被放到哪个目录？

选项有：

- .ssh ✅
- ssh ❌
- .key ❌
- .keys ❌
- ssh_keys ❌

只有 .ssh 是对的！

📝 Step 3：总结考点 & 记忆方法

✅ 本题考点：

- SSH 密钥对默认会被放在用户目录下的 .ssh 文件夹中
- 默认文件名：
  - 私钥：id_rsa
  - 公钥：id_rsa.pub

🧠 记忆口诀：

> **“钥匙放进 `.ssh`，自己藏好不能泄”**

再加一句图像化的联想：

> `.ssh` 就像你的“保险箱”，`id_rsa` 是你手中的钥匙，`id_rsa.pub` 是给门卫的开锁提示。

3.🧠 Step 1：参考资料知识讲解（从基础开始）

🔷 什么是「ホスト認証」？

Host Authentication（主机认证） 是 SSH 连接中的第一步。
目的是确认你连接的服务器**是不是“伪装者”**。

🔷 主机认证过程是怎么进行的？

当你第一次用 SSH 连接某个服务器时，会发生这些事情：

1. 服务器把自己的 **公开钥匙（public key）** 发给你
2. 你的 SSH 客户端检查这个公钥有没有**记录在案**
3. 如果是第一次连接，你的电脑不知道这个服务器是谁，就会问你：

Are you sure you want to continue connecting (yes/no)?

你输入 yes 之后，这个服务器的**主机名、IP地址、公钥信息**会被记录在：

~/.ssh/known_hosts

这个文件中。

📂 举个例子：

$ cat ~/.ssh/known_hosts
server,192.168.11.100 ecdsa-sha2-nistp256 AAAAE2V...（省略）

- server → 主机名
- 192.168.11.100 → IP地址
- ecdsa-sha2-nistp256 → 密钥算法
- AAAAE2V... → 公钥

下次连接就会自动比对这个记录：

- 如果一样，允许登录
- 如果不一样，会提示**有风险**（可能被篡改）

🔐 「known_hosts」是哪个角色？

它是**SSH客户端这边的“主机名单+身份证库”**，用来确认远程主机身份。

✅ 其他选项说明：

文件	作用
~/.ssh/known_hosts ✅	存放曾经连接过的主机的IP/主机名+公钥信息（**客户端用**）
/etc/ssh/ssh_config ❌	是客户端的"连接设置"，比如端口号、代理等，**不是公钥库**
~/.ssh/authorized_keys ❌	是服务器端用来**允许谁能登录**的（记录对方公钥）
~/.ssh/hostlist ❌	没有这个文件，这个是题目的干扰项
/etc/ssh/sshd_config ❌	是**服务器配置文件**，与主机认证无关

🧩 Step 2：题目解法

题目问的是：

> SSH连接时，主机名、IP、公钥等信息会记录在哪个客户端文件里？

✅ 答案就是：

~/.ssh/known_hosts

📝 Step 3：总结考点 & 记忆方法

✅ 本题重点：

- 主机认证是用来判断"你连接的是不是对的那台服务器"
- 所有已确认过的主机信息都会记录在 ~/.ssh/known_hosts
- 文件位置是：**客户端用户的家目录下**

🧠 记忆口诀：

> **"known_hosts 记老朋友，连接前先核身份"**
> ——只要服务器是你第一次连接的，就会往这个文件里记下来，下次验证用。

4.🧠 Step 1：从参考部分讲解知识点（基础讲起）

🔷 SSH 有两种认证机制：

类型	说明	文件位置
ホスト認証	确认"你连的是不是正确的服务器"	~/.ssh/known_hosts（客户端）
ユーザ認証	确认"你是不是一个被允许的用户"	~/.ssh/authorized_keys（服务器）

🔐 本题重点是【ホスト認証】

📌 主机认证的步骤如下：

1. 客户端（你）尝试 SSH 连接到某台服务器（例如 server）。
2. 服务器会把**自己的「公开鍵」**发给你。
3. 客户端检查：

> 1. 这个公钥在不在我 `.ssh/known_hosts` 文件里？
>   - **如果有记录** → 匹配通过 → 继续连接
>   - **如果没有** → 系统会提示你是否信任（输入 yes） → 然后把这台主机的公钥记录到 known_hosts

---

🌐 举个例子：

你第一次连接：

ssh user1@192.168.0.10

终端提示：

The authenticity of host '192.168.0.10' can't be established.
ECDSA key fingerprint is 8a:fc:...:1b.
Are you sure you want to continue connecting (yes/no)?

你输入 yes 后，会将服务器公钥存入：

~/.ssh/known_hosts

以后再次连接，会自动进行比对验证。

🧩 Step 2：题目解法说明

题目问：

> 「SSH连接时的主机认证的正确描述是哪一个？」
>
> SSH接続時のホスト認証の説明として正しいものはどれか。

我们来看每一个选项：

选项	正误	理由
❌ **SSH服务器发送"秘密鍵"，客户端用"known_hosts"里面的秘密鍵比对**	错	❌ 服务器永远**不会发送秘密鍵**，秘密鍵是保密的
❌ **客户端发送"秘密鍵"，服务器拿 authorized_keys 里的秘密鍵比对**	错	❌ authorized_keys 里面是**公钥**，验证时也用公钥
✅ **SSH服务器发送"公开鍵"，客户端拿 known_hosts 中记录的公钥比对**	对	✅ 正确！就是这个过程：对比是否是已知主机
❌ **客户端发送公钥，服务器用 authorized_keys 的公钥比对**	错	❌ 这是用户认证的过程，不是"主机认证"

📝 Step 3：总结考点 & 记忆方法

✅ 本题重点：

- 主机认证（ホスト認証）是由**客户端进行验证**
- 验证内容是：
  - "服务器这把公钥，是不是我认识的那一把？"
- 验证用的文件是：
- ~/.ssh/known_hosts
- 存放每次连接过的"服务器地址 + 公钥"组合

🧠 记忆口诀：

> **"服务自报家门（公钥），我查黑名单（known_hosts）"**
> → 服务器说：我是 server，
> → 客户端：让我翻一下通讯录，看你是不是我认识的人。

5.🧠 Step 1：参考知识讲解（从最基础开始）

🔐 SSH 用户认证方式有两种：

认证方式	原理	安全性
パスワード認証	用用户名 + 密码 登录	普通
公開鍵認証	用公钥和私钥配对方式登录	更高（推荐）

🔑 公開鍵認証の工作原理（通俗解释）：

- 你自己生成一对**钥匙**（SSH密钥对）：
  - 私钥（private key）：**自己保管，不能给别人**
  - 公钥（public key）：**告诉服务器你这把钥匙能开它的门**
- 服务器保存公钥（在 ~/.ssh/authorized_keys）
- 你连接服务器时，客户端用私钥证明“这把钥匙是我的”
- 如果公钥和私钥匹配，登录成功

📦 那么，怎么生成这对钥匙？

就用本题的主角命令：

ssh-keygen

这个命令会在你的主目录下自动创建一个 .ssh 文件夹（如果没有的话），并生成：

- id_rsa：私钥
- id_rsa.pub：公钥

🔍 图中的操作完全就是这个流程！

📂 图像说明：

$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/staff/.ssh/id_rsa): ←Enter
Created directory '/home/staff/.ssh'
...
Your identification has been saved in /home/staff/.ssh/id_rsa.
Your public key has been saved in /home/staff/.ssh/id_rsa.pub.

所以你可以看到生成了密钥对，放在 .ssh 文件夹下。

🧩 Step 2：题目解法说明

题目问：

> SSH的用户认证中，生成公钥/私钥配对的命令是哪一个？
>
> SSHのユーザ認証に使用する公開鍵と秘密鍵のペアを作成するコマンドは次のうちどれか。

选项分析：

选项	正误	理由
genssh	❌	不存在这个命令
✅ ssh-keygen	✅	✅ 正确命令！用来生成SSH密钥对
ssh-key	❌	错误命令，不存在
ssh --genkey	❌	不支持这个参数格式
ssh --key	❌	同样是不存在的选项

📝 Step 3：总结考点 & 记忆方法

✅ 本题考点：

- SSH 公钥认证用的是「一对密钥」
- 密钥对生成命令是：
- bash
- 复制编辑

- ssh-keygen
- 默认保存路径为：
- bash
- 复制编辑
- ~/.ssh/id_rsa　　（私钥）
  ~/.ssh/id_rsa.pub　（公钥）

🧠 记忆口诀：

> **“钥匙钥匙我来造，ssh-keygen 来报道”**

或者更图像化一点：

> **“ssh-keygen 就像钥匙工厂，一敲命令就造出一对钥匙”**

6. 🧠 Step 1：参考知识讲解（基础知识）

🔐 什么是「ユーザ認証」？

在 SSH 的连接过程中：

1. **第一步**是**主机认证**：确认“这台服务器是不是你要连的”，靠的是 ~/.ssh/known_hosts
2. **第二步**是**用户认证**：确认“你这个用户是不是有权限登录这台服务器”，这就用到了**公开鍵認証（public key authentication）**

🧩 公開鍵認証是怎么工作的？

假设你是客户端，现在要登录到 SSH 服务器，流程如下：

1. 你在本机（SSH客户端）用 ssh-keygen 生成密钥对
   - 私钥（private key）→ 自己保存好，不能泄露
   - 公钥（public key）→ 给服务器
2. 把你的公钥写进服务器用户目录的这个文件：

~/.ssh/authorized_keys

3. 你执行 ssh 登录时，客户端会自动用私钥进行认证
4. 服务器会拿 authorized_keys 中的公钥与客户端提供的身份做匹配
   - 一致 → 认证成功
   - 不一致 → 拒绝连接

📄 所以 authorized_keys 是什么？

它是服务器端用于“保存可以登录用户的公钥”的文件。
一行一个公钥。你可以手动编辑它，或者用 ssh-copy-id 命令自动追加。

🧩 Step 2：题目解法说明

题目问：

> 「~/.ssh/authorized_keys」文件中应该注册的是哪一个？
>
> SSHサーバの「~/.ssh/authorized_keys」ファイルに登録されるものは次のうちどれか。

我们来一个个分析：

选项	正误	原因
ユーザの秘密鍵	❌	秘密钥 **只能存在于客户端**，绝不会放在服务器
サーバの秘密鍵	❌	与用户认证无关，属于主机认证使用的，在 /etc/ssh 下
✅ ユーザの公開鍵	✅	✅ 正解！这是 authorized_keys 的内容本质
ユーザのパスワード	❌	公钥认证方式不需要也不使用密码
サーバの公開鍵	❌	用于主机认证，被保存在客户端的 ~/.ssh/known_hosts 文件中

📝 Step 3：总结考点 & 记忆方法

✅ 本题考点：

- SSH 公钥认证中：
  - **客户端保留私钥**
  - **服务器保存客户端的公钥**
- 存放公钥的地方：
- ~/.ssh/authorized_keys （服务器上）

🧠 记忆口诀：

> **“钥匙放自己兜里，锁交给服务器保管”**
>
> - 私钥 = 自己兜里的钥匙（客户端）
> - 公钥 = 提前交给服务器的“门锁图纸”（authorized_keys）

7.🧠 Step 1：基础知识讲解

🧩 什么是仮想マシン（虚拟机）？

仮想マシン（Virtual Machine，简称 VM）是用 **Hypervisor（虚拟机管理器）** 在物理机器上模拟出一整套“虚拟的电脑”，包括：

- 虚拟的 CPU / 内存 / 网卡
- 自己的 **操作系统（OS）**

也就是说，每个虚拟机像是一个**完整的电脑系统。**

🧩 什么是コンテナ（容器）？

コンテナ是比虚拟机更轻量的一种技术。

- 它不需要完整的操作系统
- 是在主机操作系统上创建的**“独立运行环境”**
- 通常使用 **Docker** 来创建和管理

容器是通过共享**主机的 OS 内核**来实现的，所以**更轻、更快。**

✅ 二者对比：

项目	仮想マシン（VM）	コンテナ（Container）
是否包含完整OS	✅ 有，启动慢，重资源	❌ 无，只共享主机的内核
启动速度	慢	快
移植性	一般	高，可打包镜像带走
是否可以运行不同操作系统	✅ 可以运行不同的 OS	❌ 只能使用主机 OS 的内核
资源隔离性	高（安全性强）	较低（更依赖主机 OS）
容易移植从物理机迁移	✅ 易于封装完整系统迁移	⚠️ 通常用于轻量级部署

🧩 Step 2：题目解法说明

题目问：

> **与容器相比，虚拟机更优秀的点有哪些？（选两个）**
>
> コンテナと比較して、仮想マシンが優れている点はどれか。（2つ選択）

我们挨个分析每个选项：

选项内容	正误	理由
オーバーヘッドが小さい（开销小）	❌	容器的启动更快，占用资源更小。这个是**容器的优点**。
1つのホスト環境で複数のOSを共存できる（可运行多个不同OS）	✅	这是虚拟机的优势，比如同时运行 Linux 和 Windows。
物理マシンから構成を移行しやすい（便于从物理机迁移）	✅	VM 可以打包整个系统（操作系统+应用+配置），迁移方便。
別の環境においても実行しやすい（便于跨环境部署）	❌	这个是容器的优势，容器镜像可以在任何支持Docker的平台上运行。

📝 Step 3：总结考点与记忆方法

✅ 本题考点：

仮想マシン优点	容器优点
可以运行多个不同操作系统	启动快、占资源少
更强的资源隔离与安全性	容易在不同环境中部署（高可搬性）
更适合从物理机"整体打包迁移"	镜像打包方便、分发容易

🧠 记忆口诀：

> **虚拟机像一套房子，容器像租一个房间**

- 虚拟机：带厨房厕所，有门有锁，自己装 OS，安全但占资源。
- 容器：在公寓里租一间，轻便省事，和其他人共用一个 OS。

✅ 本题正解：

- ✅ **1つのホスト環境で複数のOSを共存できる**
- ✅ **物理マシンから構成を移行しやすい**

8.🧠 Step 1：基础知识讲解（ホストOS & ゲストOS）

✅ 什么是ホストOS（Host OS）？

> Host OS 是安装在**物理电脑上**的操作系统。

你可以把物理机器想象成一个“宿主机”（host），上面运行的是主操作系统，叫做 **ホストOS**，比如 Windows、Linux、macOS 等。

✅ 什么是ゲストOS（Guest OS）？

> Guest OS 是安装在**虚拟机里的操作系统**。

它是通过**虚拟化软件**（如 VirtualBox、VMware）运行的，就像你电脑里开了一台“虚拟电脑”，在这个虚拟电脑里你可以再装一个完整的操作系统，这个就是 **ゲストOS**。

💡 二者的关系可以这样理解：

项目	ホストOS	ゲストOS
安装位置	安装在物理机器上	安装在虚拟机上
启动顺序	先启动主机系统	之后才能打开虚拟机运行 guest OS
是否能单独运行	✅ 独立操作系统	❌ 依赖于虚拟机软件和主机系统

🧪 例子说明：

你买了一台笔记本，装了 Windows 11，这是你的 **ホストOS**。

然后你在 Windows 里装了 VirtualBox，然后用它装了 Ubuntu 20.04，这就是你的 **ゲストOS**。

🧩 Step 2：题目分析

题目问：

> 关于 ホストOS 和 ゲストOS 的正确描述有哪些？（选两个）
>
> ホストOSとゲストOSの説明として正しいものはどれか。(2つ選択)

我们来一个个分析：

选项内容	正误
❌ **ホストOSは仮想マシンにインストールしたOSである**	错误。**ホストOS 是装在物理机上的系统**，装在虚拟机里的是 Guest。
❌ **ホストOSを停止してもゲストOSを動かせる**	错误。虚拟机依赖主机系统，主机关了虚拟机当然也就停了。
❌ **1つのゲストOS上で複数のホストOSを起動できる**	错误。**是1台 Host 可以起多个 Guest**，反过来不行。
✅ **ゲストOSは通常のOSのように停止したり再起動できる**	正确！可以像真机一样操作关机、重启等。
✅ **ゲストOSは仮想マシンにインストールしたOSである**	正确！Guest 就是装在虚拟机里的操作系统。

📝 Step 3：总结考点 & 记忆技巧

✅ 本题考点：

- **Host OS**：安装在物理机上的操作系统
- **Guest OS**：安装在虚拟机（Virtual Machine）里的操作系统

🧠 记忆法：

> **“主人是 Host，客人是 Guest”**

就像你是屋主（Host），你请朋友来家里住（Guest），朋友住在你搭的帐篷里（虚拟机），他能自己洗澡吃饭（能重启、关机），但你家没电他也没办法用电（Host 关机他也关机）

✅ 本题正解：

- ✅ ゲストOSは仮想マシンにインストールしたOSである
- ✅ ゲストOSは通常のOSのように停止したり再起動できる

9.🧠 Step 1：基础知识讲解（Kubernetes 是什么？为什么不是 Docker？）

🔷 什么是容器（コンテナ）？

容器是一种轻量级的虚拟化技术，它将应用程序及其所有依赖打包在一起，可以在任何地方运行。

容器 ≠ 虚拟机（VM）。容器**共享主机的内核**，所以比虚拟机更"轻"、更"快"。

🔶 Docker 是干嘛的？

Docker 是一个容器运行工具，你可以：

- 用它"做出"一个容器（像是做菜）
- 用它"运行"一个容器（像是把菜端上桌）
- 一台机器上运行少量容器用 Docker 就够了

> 🚫 但当你有**几十、上百个容器**、多个服务器时，Docker 本身就不够用了。

✅ Kubernetes 是什么？（重点！）

Kubernetes（K8s）是一个 **容器编排平台（オーケストレーション）**。

就像你要指挥一个乐团，而不是一个独奏者。

它能做的事有：

功能	举例说明
自動デプロイメント	自动在多台服务器上部署容器
オートスケーリング	用户多了，自动增加容器
自動復旧	某个容器挂了，它自己重新拉起来
ロードバランス	把用户请求平均分发给不同的容器

所以，**Kubernetes 是管理一大群容器的"管家 + 指挥家"**，专为大规模容器环境而生。

🧩 Step 2：题目分析

题干：

> 多数のコンテナを効率的に管理するために、最適なソフトウェアはどれか。

选项逐一分析：

选项	正误	原因
Nested Virtualization	❌	是嵌套虚拟化，不是用于管理容器的
Docker	❌	适合少量容器，不适合管理大规模容器集群
Kernel-based Virtual Machine (KVM)	❌	这是虚拟机（VM）技术，和容器不同
Kubernetes	✅	专为管理**大规模容器**设计的开源平台

📝 Step 3：总结考点 & 记忆技巧

✅ 本题考点

- Docker 是"一个容器"的运行工具
- Kubernetes 是"大量容器"的管理平台
- KVM 是虚拟机的技术，不适用于容器
- Nested Virtualization 是嵌套虚拟，不用于容器管理

🧠 记忆法：

> **Docker 做菜，Kubernetes 管厨房！**

- Docker：炒一盘菜（一个容器）
- Kubernetes：管理整个厨房的厨师团队（成百上千容器）

✅ 正解：

- **Kubernetes**

10. 🧠 Step 1：基础知识讲解（什么是嵌套虚拟化？）

🔷 什么是"ネストされた仮想化"?

嵌套虚拟化（Nested Virtualization）就是：

> **在一台虚拟机里再跑一台虚拟机。**

举例说明：

你有一台物理电脑（A），你在上面装了 VirtualBox，然后你创建了一个虚拟机（B），在 B 里面你又运行了另一个虚拟机（C）。

层级	内容
A	Host OS（物理机 + Hypervisor）
B	Guest OS（第一层虚拟机）
C	Nested Guest（第二层虚拟机）

🔶 为什么需要嵌套虚拟化?

嵌套虚拟化非常适合这些场景：

- **教学演示或技术培训**：比如讲 KVM 如何创建虚拟机时，本机没有物理服务器，只能在虚拟机中模拟。
- **测试环境**：测试虚拟机管理、云平台架构（如 OpenStack）
- **开发环境隔离**：做更复杂的集群模拟测试

🔧 启用嵌套虚拟化，需要什么?

1. **CPU支持虚拟化指令集**
   - Intel：VT-x
   - AMD：AMD-V
     ✅ BIOS 里要开启
2. **使用支持嵌套虚拟化的 Hypervisor**
   - ✅ 有支持：KVM, VMware, VirtualBox
   - ❌ 并不是所有 Hypervisor 都支持
3. **注意性能开销**
   - 因为一层套一层，会明显拖慢运行速度

🧩 Step 2：题目解法说明

题目问：

> 关于虚拟机和嵌套虚拟化，正确的描述有哪些？（选3个）
>
> 仮想マシンやハイパーバイザーのネストについて、正しい説明を選べ。（3つ選択）

我们来一项一项分析：

选项	正误	理由
✅ **仮想マシンのOS上でさらに別の仮想マシンとOSを動作させる**	正确	这正是嵌套虚拟化的定义
✅ **パフォーマンス低下を招くことがある**	正确	嵌套层数越多，资源开销越大，性能下降明显
✅ **CPUの仮想化支援機能が必要、または推奨とされる**	正确	VT-x / AMD-V 是必须开启的底层支持
❌ **どのハイパーバイザーでも実現できる**	错误	只有部分支持，如 KVM / VMware / VirtualBox，不能一概而论
❌ **ハイパーバイザーで複数の仮想マシンを並列に動作させる**	错误	这只是一般的虚拟化功能，与"嵌套"无关

📝 Step 3：总结考点与记忆方法

✅ 本题考点：

项目	正误	说明
嵌套虚拟化的定义	✅	虚拟机内再运行虚拟机
性能开销	✅	嵌套越多越慢
对 CPU 的要求	✅	必须支持 VT-x 或 AMD-V
并非所有 Hypervisor 都支持	✅	一定要使用兼容的，比如 KVM、VMware
并行虚拟机 ≠ 嵌套	❌	并行不是嵌套概念

🧠 记忆口诀：

> **"套中套，慢又耗，支持嵌套要硬靠"**
> （嵌套虚拟化慢，资源消耗大，必须靠CPU硬件支持）

或者图像化理解：

> 嵌套虚拟化就像：你在一间屋子里，搭了个帐篷，帐篷里又放了个娃娃屋，层层嵌套，当然又热又挤🫠

✅ 正确答案：

- 仮想マシンのOS上でさらに別の仮想マシンとOSを動作させる
- パフォーマンス低下を招くことがある
- CPUの仮想化支援機能が必要、または推奨とされる

11.🧠 Step 1：基础知识讲解（从什么是虚拟机讲起）

🔷 什么是仮想マシン（Virtual Machine, VM）？

虚拟机就是通过一种叫做 **ハイパーバイザー（Hypervisor）** 的软件，在一台物理电脑里"模拟出另一台电脑"，然后在这个"虚拟的电脑"里运行一个操作系统，这个被安装的系统就叫做 **ゲストOS（Guest OS）**。

🔶 Hypervisor 是什么？

Hypervisor 是虚拟机的"大本营"，负责：

- 把物理主机的资源（CPU、内存、磁盘）**虚拟化**
- 分配给各个虚拟机使用
- 控制虚拟机的启动、暂停、快照、关机等操作

🔷 Hypervisor 的分类（VM 的两种运行模式）

类型	名称	说明	示例
Type 1	ネイティブ型	安装在物理机上	KVM, Hyper-V, ESXi
Type 2	ホスト型	安装在 Host OS 上（作为软件）	VirtualBox, VMware Workstation

---

✅ 典型结构图：

[物理机]
└── Hypervisor（如 KVM）
├── VM1（运行 Ubuntu）
├── VM2（运行 Windows）

每个 VM 中都可以像独立电脑一样安装 OS、运行服务、连接网络。

🧩 Step 2：题目分析与解法

题目问：

> 关于虚拟机，正确的说明是哪一项?
>
> 仮想マシンの説明として正しいものはどれか。

我们逐一来看选项：

选项	正误	原因
❌ ゲストOS上で仮想マシンを起動し、ホストOSを動作させる	错	顺序错了：应是 Host OS → Hypervisor → Guest OS
❌ 代表的なソフトウェアにはLXC、Dockerがある	错	LXC 和 Docker 是 **容器技术**，不是虚拟机
❌ ホスト型とコンテナ型がある	错	正确应该是：**ネイティブ型 和 ホスト型**（容器≠虚拟机）
✅ **ハイパーバイザー上で仮想マシンを起動し、ゲストOSを動作させる**	正确	这是虚拟机系统结构的标准描述

📝 Step 3：总结考点 & 记忆方法

✅ 本题考点

- 虚拟机运行在 **Hypervisor 上**
- Hypervisor 可以是安装在物理机上的 Type 1，也可以是装在 Host OS 上的 Type 2
- 每台虚拟机内部运行的就是 **Guest OS**
- 容器（如 Docker）和虚拟机是不同的技术栈

🧠 记忆口诀

> **“虚拟机搭在 hypervisor 上，hypervisor 搭在主机或主系统上”**

再形象点：

- Hypervisor 就像“搭帐篷的地基”
- 每个 VM 就像是帐篷
- 帐篷里可以装上不同的“家具”（即不同的操作系统）

✅ 正确答案：

- **ハイパーバイザー上で仮想マシンを起動し、ゲストOSを動作させる**

12.🧠 参考知识讲解

什么是 virsh?

virsh 是一个用于管理 KVM 虚拟化环境中虚拟机（VM）的命令行工具。它是 libvirt 提供的管理接口之一。

什么是「一時停止中」?

这是虚拟机被「暂停（suspend）」的状态，相当于按了暂停键，虚拟机会保存当前状态到内存中，但 CPU 不再执行虚拟机的进程。

「resume」是什么意思?

英文的 resume 有「继续」、「恢复」的意思。在虚拟机中，resume 指的是恢复一个之前暂停（suspend）状态的虚拟机，让它继续运行。

🔍 题目解说

题目问的是：

> virshコマンドのサブコマンドで、一時停止中の仮想マシンを再開するものはどれか。
> （用 virsh 恢复暂停状态的虚拟机，使用哪个子命令？）

我们先看一下五个选项：

420. **suspend**：暂停虚拟机 → ❌（题目问的是“恢复”，不是“暂停”）
421. **unpause**：这是 Docker 或 container 的用法，在 virsh 中并不存在 → ❌
422. **resume**：恢复虚拟机 → ✅ 正解！
423. **start**：启动虚拟机，但只对“关闭状态”的虚拟机有效（不是从“暂停”状态恢复）→ ❌
424. **reboot**：重启虚拟机 → ❌

所以，正确答案是：

> ✅ **resume**

📝 总结：考点 & 记忆法

📌 本题考点

- virsh 的基本命令；
- 理解虚拟机状态（运行中 / 暂停中 / 关闭中）；
- 熟悉 resume 是恢复暂停虚拟机的命令。

🧠 记忆方法小技巧：

- **suspend（暂停）↔ resume（恢复）**：像遥控器上的暂停/继续按钮。
- **start** 是“冷启动”，resume 是“继续执行”。
- 可以联想：“resume your work after lunch”（吃完饭继续工作）！

13.🧠 **题目：**

> virshコマンドのサブコマンドで、仮想マシンの定義ファイルをXML形式で出力するものはどれか。

选项如下：

- xml

- dumpxml
- console
- config
- list

正解： dumpxml

🔍 参考知识讲解

✅ 1. 什么是 virsh 命令?

virsh 是一个命令行工具，用于管理基于 libvirt 的虚拟化平台（比如使用 KVM 的虚拟机）。

你可以通过 virsh 来做几乎所有关于虚拟机的操作，比如：

- 启动/关闭虚拟机
- 暂停/恢复虚拟机
- 显示虚拟机配置
- 创建/销毁虚拟机

✅ 2. 什么是 "仮想マシンの定義ファイル"?

定义文件（也叫"配置文件"）是描述一台虚拟机的**所有配置细节**的 XML 文件，比如：

- 内存大小
- CPU 数量
- 使用的磁盘镜像
- 网络接口的设置
- 引导方式等

在你提供的截图中，virsh dumpxml centos7 命令的输出就展示了一个叫 centos7 的虚拟机的定义细节。这个文件对**虚拟机的创建、导入、迁移**都非常重要。

✅ 3. dumpxml 是什么?

dumpxml 是 virsh 提供的子命令之一，它的作用就是：

📝 **将某个虚拟机的定义（配置信息）以 XML 形式输出。**

语法如下：

virsh dumpxml <虚拟机名>

你也可以输出到文件中，例如：

virsh dumpxml centos7 > centos7_dump.xml

这样可以保存成文件，之后用 virsh create 或 define 创建或重新导入虚拟机。

❌ 其他选项错误的原因：

选项	原因
xml	virsh 并没有 xml 这个子命令
console	用于连接虚拟机控制台，和定义文件无关
config	virsh 没有 config 子命令
list	显示虚拟机列表，而不是输出定义文件

✅ 题目解法说明：

题目问的是：**哪个 virsh 子命令可以将虚拟机定义文件以 XML 形式输出?**

根据我们上面讲解的内容，这正是 **dumpxml** 的作用，因此选择：

✔ **正解：dumpxml**

🧠 总结考点 & 记忆方法：

🧠 考点总结：

- virsh 是管理虚拟机的 CLI 工具
- 每台虚拟机都有一个定义文件（XML）
- dumpxml 可以导出这个配置文件
- virsh dumpxml <虚拟机名> 是日常运维常用命令

✨ 记忆技巧：

你可以这样记：

> **dump（导出） + xml（XML格式） = 导出 XML 定义文件**

或者编个口诀：

> **想看虚机配置信，dumpxml 是法宝。**

サブコマンド	説明
create	XML形式のファイルから仮想マシンを作成し、 新規の仮想マシンを起動
console	仮想マシンのコンソールに接続
start	仮想マシンを起動
shutdown	仮想マシンを正常に終了
destroy	仮想マシンを強制終了
reboot	仮想マシンを再起動
suspend	仮想マシンを一時停止
resume	仮想マシンを再開
list	仮想マシンの一覧を表示(デフォルト:稼働中のみ) --all：全ての仮想マシンを表示
dumpxml	仮想マシンの定義ファイルをXML形式で出力
※リソースを含む仮想マシンの構成を「ドメイン」という	

🔍 为什么这是重点?

✅ 1. virsh 是 KVM 虚拟化的标准操作工具

你在学习 Linux 的虚拟化时，几乎所有对虚拟机的日常操作都要通过 virsh 命令来完成，比如：

- 启动虚拟机
- 关闭虚拟机
- 查看虚拟机列表
- 导出配置文件

考试题目会非常喜欢考你这些基础命令的**用法和作用对照。**

🧠 如何记忆这个表?

我建议你分组记忆 + 编口诀：

【启动/停止】

- start：启动
- shutdown：正常关机
- destroy：强制关机（像拔电源）

- reboot：重启

📝 **记法**：start → shutdown → reboot（流程顺序），destroy 是紧急强制手段。

【暂停/恢复】

- suspend：暂停
- resume：恢复

📝 **记法**："挂起后恢复"，就是 sleep/resume。

【创建与控制台】

- create：从 XML 定义文件创建虚拟机
- console：连接虚拟机的控制台

📝 **记法**：create 是初始化用的，console 是远程连进去操作用的。

【查询类】

- list：查看运行中的虚拟机（加 --all 可查看所有）
- dumpxml：输出 XML 格式的定义文件

📝 **记法**：list 是列表，dumpxml 是导出定义。

🧩 小总结口诀：

> 启动 start，关机 shut，紧急 destroy，重启 reboot；
> 暂停 suspend，再开 resume；创建 create，控制台 console；
> 查询 list，看配置 dumpxml！

📌 另外表格下面还有一句：

> リソースを含む仮想マシンの構成を「ドメイン」という
> 这也要记！考试中可能会用「ドメイン」来指代"虚拟机"。

14. ✅ 参考知识讲解：virsh reboot 相关知识

1. 什么是 virsh?

virsh 是 libvirt 提供的命令行工具，用于管理虚拟机（也叫"域" domain）。通过它可以启动、停止、挂起、恢复、列出虚拟机等操作。

2. 什么是虚拟机的"再起动"?

虚拟机重启（reboot）是指不关闭虚拟机，而是让其像按下实体电源按钮那样"重新开机"。

✅ 各选项解释：

命令	功能	是否为有效的 virsh 子命令
start	启动尚未运行的虚拟机	✅ 有效命令（但不能用于再启动）
restart	**不是 virsh 的有效命令**	❌ 无此命令
reboot	**重启正在运行的虚拟机**	✅ 正确答案
halt	**不是 virsh 的有效命令**（但类 Unix 系统中用于关机）	❌ 无此命令
shutdown	正常关闭虚拟机	✅ 有效命令（但不是重启）

✅ 解题说明：

本题问的是："哪一个 virsh 的子命令可以让虚拟机再启动？"

从定义来看：

- virsh reboot 会对正在运行的虚拟机发送“重启指令”，即关机后自动再启动。
- 所以选择：
  - ✅ reboot

✅ 总结与记忆法：

📌 关键考点

- virsh reboot = 再启动虚拟机
- virsh start = 启动（从关闭状态）
- virsh shutdown = 正常关闭
- virsh destroy = 强制关闭（断电）

🧠 记忆口诀

> 「start 启，shutdown 关，reboot 是重启，destroy 是硬断电」

15. ✅ 参考知识讲解：virsh destroy 是做什么的?

1. virsh 是什么?

virsh 是 libvirt 的命令行工具，用来操作虚拟机（在 libvirt 中称为“域” domain）。通过它我们可以启动、关闭、挂起虚拟机等操作。

---

2. 虚拟机的关闭方式有两种：

关闭方式	命令	特点
正常关闭	virsh shutdown	就像你在操作系统中点击“关机”，会依赖操作系统自己关机
强制关闭	virsh destroy	就像直接按电脑的电源键断电，立即关闭，不管系统状态

---

3. 为什么叫 destroy（不是 kill 或 stop）？

这是 libvirt 的命名规则，它的意思并不是“删除虚拟机”，而是**强制终止正在运行的虚拟机。**

实际效果相当于断电，对数据有风险，不推荐平时使用，除非系统卡死或无法正常关机。

✅ 解题说明：

本题问的是：

> virsh 的哪个子命令可以“强制关闭”虚拟机?
>
> virshコマンドのサブコマンドで、仮想マシンを強制終了するものは次のうちどれか。

选项分析如下：

选项	是否为 virsh 命令	含义	正确吗
shutdown	✅ 有效命令	正常关机（温和）	❌
force	❌ 无此命令	没有 virsh 的 force 命令	❌
destroy	✅ 有效命令	**强制关闭**虚拟机	✅ 正确
kill	❌ 无此命令	类 Unix 命令，但不是 virsh 的	❌
stop	❌ 无此命令	Docker 有 stop，virsh 没有	❌

✅ 正确答案：

- **destroy**

✅ 总结与记忆法：

🧠 重点记忆命令

命令	含义
start	启动虚拟机
shutdown	正常关闭虚拟机
destroy	**强制关闭虚拟机**
reboot	重启虚拟机
suspend	暂停虚拟机
resume	恢复运行虚拟机

📌 记忆小技巧：

> **“destroy 是断电，shutdown 是关机”**
> 就像电脑死机了，你会按电源键强制关机一样，destroy 是最暴力的方法。

16. ✅ 参考知识讲解：virsh shutdown 是做什么的？

1. virsh 是什么？

virsh 是管理虚拟机的命令行工具，是 libvirt 虚拟化平台提供的接口，可以控制虚拟机的启动、停止、暂停、查看信息等。

---

2. shutdown 的含义是什么？

- shutdown 的含义是**正常关机**，就像我们在 Windows 或 Linux 桌面系统上点击“关机”一样，系统会：
  - 关闭正在运行的程序
  - 释放资源
  - 安全写入磁盘
  - 然后才断电

🟢 所以，**它是一种优雅、安全的虚拟机关机方式。**

---

3. destroy 和 shutdown 的区别（再次对比）

命令	含义	是否推荐	类比现实操作
shutdown	正常关机	✅ 推荐	正常点击“关机”按钮
destroy	**强制断电关闭**	⚠️ 不推荐	拔电源/长按电源按钮

✅ 解题说明：

题目问的是：

> virsh 的哪个命令能“正常地结束虚拟机运行”？

选项分析如下：

选项	是否为 virsh 子命令	说明	是否正确
halt	❌	不存在于 virsh	❌
shutdown	✅	✅ 正常关机	✅ 正确
finish	❌	virsh 中无此命令	❌
stop	❌	Docker 有，virsh 没有	❌
destroy	✅	⚠️ 强制关闭（非正常）	❌

✅ 正确答案：

- **shutdown**

✅ 总结考点 + 记忆法

🧠 重点命令对比

命令	含义	是否安全
start	启动虚拟机	✅
shutdown	**正常关机**	✅ 推荐
destroy	**强制断电关机**	⚠️ 不推荐
reboot	重启虚拟机	✅
suspend	暂停虚拟机	✅
resume	恢复运行虚拟机	✅

📌 记忆口诀：

> ✅「shutdown 是关机」
> ⚠️「destroy 是拔电源」

17.✅ 参考知识讲解：virsh create 是做什么的?

1. virsh 是什么?

virsh 是管理虚拟机（虚拟化环境中"**域（domain）**"）的命令行工具，属于 libvirt 的一部分。

2．什么是"XML形式の定義ファイル"?

在 libvirt 中，**虚拟机的配置**（例如 CPU 核心数、内存、磁盘路径、网络设置等）会写成一个 **XML 文件**，这就像是虚拟机的"说明书"。

例如这样的内容：

```
<domain type='kvm'><name>centos7</name><memory>1048576</memory><vcpu>1</vcpu>
  ...
</domain>
```

---

3. virsh create 命令的作用

virsh create 是：

> **从 XML 文件创建并启动一个虚拟机实例（一次性启动）**

🟢 使用场景：

virsh create centos7.xml

这会立刻：

- 读取 XML 配置
- 分配资源
- 启动虚拟机

🟠 注意：
这个方法 **不会把虚拟机注册进永久清单（不会保存）**，只是在当前 session 下运行一次。

如要注册进系统永久清单，需要使用 virsh define 命令。

✅ 解题说明：

题目问的是：

> virsh 中哪个命令是 "从 XML 文件创建并启动虚拟机"的?
>
> virshコマンドのサブコマンドで、XML形式のファイルから仮想マシンを作成し、新規の仮想マシンを起動するものは次のうちどれか。

我们来分析选项：

选项	含义/存在性	是否正确
start	启动一个"已定义"的虚拟机（不能从 XML 创建）	❌
new	virsh 中不存在	❌
xml	virsh 中不存在	❌
create	✅ **从 XML 创建并启动虚拟机**	✅ 正确
console	用于连接虚拟机终端，不是启动	❌

✅ 正确答案：

- **create**

✅ 总结考点与记忆法：

🧠 对比：create vs start

命令	功能	注册保存?
create	从 XML 创建并启动虚拟机（一次性）	❌
start	启动已存在的虚拟机（已定义）	✅
define	从 XML 文件注册虚拟机（不启动）	✅

🧠 口诀记忆法：

> **create：创建并启动！**
> **start：已存在才能启！**

18. 🧠 Step 1：基础知识讲解——什么是 systemctl rescue?

🔷 单用户模式（rescue.target）是什么?

- 在 systemd 系统中，执行：
- systemctl rescue
- 就会让系统进入 **单用户模式（rescue mode）**
- 特点是：
  - 仅启动最基本的服务
  - **只允许 root 登录**
  - 用于系统维护，比如修复文件系统、重设密码等
  - 图形界面、网络等服务都不会启动

🔷 进入 rescue 模式后，如何"返回正常状态"?

1. **Ctrl-D**
   这个是图中提示的最直观做法，就像"告诉 systemd：我已经修好了，请继续启动"
2. **systemctl default**
   这个命令会让 systemd 切回"默认的目标（target）"，也就是你平常开机时进入的 multi-user.target（文本界面）或 graphical.target（图形界面）
3. **systemctl reboot**
   重启系统，重启后自然就进入默认的 target，也等于"退出了 rescue 模式"

🧩 Step 2：题目解法说明

题目问：

> 你执行了 `systemctl rescue` 进入了单用户模式，现在要回到正常模式，**哪三个方法是正确的?**
>
> 作業が終わったので通常の起動状態に戻したい。どうすれば良いか（3つ選択）

我们逐项来看选项：

选项	含义	正误	说明
✅ **Ctrl-Dを入力する**	相当于继续启动，图中有明确提示	正确	✔
✅ **systemctl default**	切换回默认启动 target	正确	✔
✅ **systemctl reboot**	重启回默认模式	正确	✔
❌ Ctrl-C	是发送中断信号（SIGINT），不是退出模式的方法	错误	✗
❌ 电源断电重开	危险行为，有可能造成数据丢失	错误	✗
❌ systemctl endrescue	不存在这样的命令	错误	✗

📝 Step 3：总结考点 & 记忆方法

✅ 你要记住的命令逻辑：

命令	用途
systemctl rescue	进入单用户模式
Ctrl-D	从 rescue 模式继续启动
systemctl default	切换回默认模式（multi-user / graphical）
systemctl reboot	重启后回默认模式

🧠 记忆口诀：

> **进入模式用 rescue，回去方式三条路：Ctrl-D、default、reboot**

或者像按钮操作一样想象：

> 按了暂停（rescue）之后 →
> ✔ 点继续（Ctrl-D）
> ✔ 按默认模式按钮（default）
> ✔ 重启系统也可以（reboot）

19.🧠 Step 1：基础知识讲解——systemd 的 target 与传统 runlevel 的关系

在旧的 SysVinit 时代，我们使用 runlevel（运行级别）来控制系统的启动状态。例如：

Runlevel	含义
0	关机
1	单用户模式
2~4	多用户模式
5	图形界面
6	重启

而在现在广泛使用的 **systemd** 中，用的是叫做 **target（目标单元）** 的概念，下面是它们的对应关系：

ランレベル（旧）	systemdのターゲット
0	poweroff.target
1	rescue.target
2,3,4	multi-user.target
5	graphical.target
6	reboot.target

🔍 Step 2：题目分析——systemctl start poweroff.target 会发生什么？

语法说明：

systemctl start <target名>

表示「**启动这个目标对应的服务集合**」。

在这个题目中执行的是：

systemctl start poweroff.target

意思就是「执行关机操作」，相当于以前的 telinit 0 或 shutdown -h now。

🔥 常见错误选项的解释：

- ❌ **强制断电处理**
  → 强制关机（如 poweroff -f）会跳过关机脚本。而 poweroff.target 是"正常地"执行关机流程，所以不对。
- ❌ **启动名为 poweroff 的服务**
  → .target 是目标（Target Unit），不是 .service 服务单元，不是启动服务。
- ❌ **重新启动系统**
  → 如果是想要 reboot，应该是 reboot.target 或 systemctl reboot。

✅ Step 3：总结考点 & 记忆方法

✅ 一张表背 systemd 的常用目标（target）

目标	用法或对应命令
poweroff.target	关机，相当于 telinit 0
rescue.target	单用户模式
multi-user.target	多用户文字界面，类似 runlevel 3
graphical.target	多用户图形界面，类似 runlevel 5
reboot.target	重启，相当于 telinit 6

🧠 记忆口诀：

> 0关1修234多5图6转（零关机、一修复、二三四多用户、五图形、六重启）
>
> systemctl + target 是最安全的管理方式！

20.🔍 参考知识讲解：如何「现在马上重启系统」

🧠 先讲最基本的概念：

🚀 什么是"重启系统"？

就是让当前操作系统关机后**重新启动**，这一步会关闭所有服务，断开所有连接，然后重新开机。
重启常用于系统更新、配置变更生效等场景。

🧱 常见的几种重启方法（命令）

命令	含义	是否立即重启	使用范围
shutdown -r now	立刻重启	✅	传统Linux系统
systemctl reboot	立刻重启	✅	使用 systemd 的系统
systemctl start reboot.target	启动 reboot 目标，相当于 telinit 6	✅	与 systemd 兼容

🔥 容易混淆的命令解释（不正确选项）：

命令	实际作用	为什么错?
shutdown -h now	**立刻关机**（halt）	❌ 是关机不是重启
poweroff	立刻关机	❌ 是关机
systemctl restart	重启「服务」如 nginx/mysql	❌ 不是重启系统本身

✅ 解题说明

题目问的是「**现在马上重启系统**」的命令有哪些？正确答案应能立刻让系统重新开机。

✅ 正确选项解释：

1. shutdown -r now
   → -r 表示重启，now 表示立刻
   → ✅ 正确
2. systemctl reboot
   → systemd 的标准重启指令，相当于 reboot
   → ✅ 正确
3. systemctl start reboot.target
   → 启动 reboot.target，等价于 telinit 6
   → ✅ 正确

✅ 正确答案是：

- shutdown -r now
- systemctl reboot
- systemctl start reboot.target

📌 记忆方法总结

行为	命令	记忆法
正常重启	shutdown -r now	r = reboot
正常重启	systemctl reboot	systemd 系统推荐写法
正常重启	systemctl start reboot.target	目标（target）就是重启系统
正常关机	shutdown -h now, poweroff	h = halt = 停止 = 关机

📒 口诀记忆：

> 「shutdown -r 是老派，systemctl reboot 是新派，reboot.target 是终极派」

systemctl 指令常用サブコマンド（子命令）对照表

サブコマンド	用途
disable	サービスの自動起動を無効にする
enable	サービスの自動起動を有効にする
get-default	次回起動時のターゲットを表示する
halt	システムを停止しhalt状態にする
is-active	サービスが稼働しているかを表示する
isolate	他のUnitを停止して対象のUnitを起動する
list-unit-files	すべてのUnit定義ファイルを一覧表示する
reboot	システムを再起動する
reload	サービスの設定ファイルを再読み込みする
rescue	レスキューモードに移行する
restart	サービスを再起動する
set-default	次回起動時のターゲットを設定する
start	サービスを起動する
status	サービスの状態を表示する
stop	サービスを停止する
poweroff	システムを停止し電源を切断する

🧩 按用途分组记忆：

🟩 【服务的起停控制类】

子命令	用途
start	启动服务
stop	停止服务
restart	重启服务
status	查看服务状态
reload	重新加载配置文件
is-active	是否正在运行（布尔值）

📌 记忆口诀：

> 「スタートで起動、ストップで停止、リスタートで再起動！」

🟦 【服务自启设置类】

子命令	用途
enable	启用自启（开机自动启动）
disable	禁用自启
get-default	查看当前默认目标（target）
set-default	设置下次开机默认目标

🟨 【系统层级操作类】

子命令	用途
reboot	重新启动整个系统
halt	停止系统（不一定断电）
poweroff	关机 + 断电
rescue	进入救援模式（单用户维护模式）
isolate	切换 target（只保留一个 Unit）

📌 **记忆重点：**

- reboot：相当于 restart OS
- poweroff：彻底断电关机

- rescue：修系统用（单用户模式）

🟧 【信息/文件查看类】

子命令	用途
list-unit-files	显示所有 Unit 定义文件

21.🧠 一、【参考知识点讲解】——什么是"カーネル（Kernel）"?

我们从 **系统启动流程** 的大背景开始说起。

🔁 系统启动的顺序是：

1. **BIOS/UEFI**：最初运行，负责检测并初始化硬件（比如内存、硬盘） → 然后找出可引导设备。
2. **Boot Loader（引导加载程序）**：比如 GRUB，负责从硬盘中加载 Linux **内核（kernel）**。
3. **Kernel（内核）**：主角来了！负责：
   - 加载硬件驱动
   - 挂载根文件系统（/ root）
   - 启动用户空间的第一个程序 /sbin/init 或 systemd
4. **init 或 systemd**：根据配置文件（如 /etc/inittab 或 systemd 的 unit）来启动后台服务、网络、图形界面等

🧩 那么 Kernel（内核）到底干了什么？

功能	说明
硬件识别与初始化	比如 CPU、内存、磁盘、USB、网卡等等
驱动模块加载	有些驱动在 initramfs 里，内核负责加载
文件系统挂载	尤其是挂载 / 根文件系统，才能读取其他程序
启动第一个进程	/sbin/init 或 /lib/systemd/systemd

✅ 二、题目解法说明

> 题干：システム起動時においてのカーネルの説明として正しいものは?

我们来判断选项：

选项内容	正误	理由
✅ 高度にハードウェアを認識・制御し、ルートファイルシステムのマウントなど様々な初期化処理を行う	✔ 正确	这是 kernel 的核心职责
✅ /sbin/initを起動する	✔ 正确	内核启动完后会执行第一个进程就是 init
記憶装置（HDD）内のカーネルをロードし、制御を移す	❌ 错误	这是 **Boot Loader** 做的事情
記憶装置等に関して最低限の認識を行う	❌ 错误	这是 **BIOS/UEFI** 的任务
/etc/inittab に基づいてプロセスを立ち上げる	❌ 错误	这是 **init/systemd** 的任务

📝 三、总结考点与记忆法

✅ 必背考点

- **Kernel 的职责**：
  - 初始化硬件驱动

- 加载根文件系统
- 启动 /sbin/init 或 systemd

- 启动顺序：
  BIOS → Bootloader → Kernel → Init

🧠 记忆口诀：

> 「BI-K-I-I」
> BIOS → Kernel → Init → 守护进程
> 或者说：
> **“BIKINI启动法”：BIOS → Kernel → init!**

22.🧠 一、参考知识点讲解：什么是 nohup 和 &?

🔷 背景问题：为什么我们需要 nohup?

平时在终端中执行一个命令（例如 ping localhost），这个命令属于当前 shell 的「子进程」。
一旦你关闭终端、登出，shell 就会结束，它的子进程也会跟着被「杀死」。

这就带来了一个问题：

> 有些任务你希望它「即使你登出也继续执行」。

为了解决这个问题，我们就要用两个东西搭配起来：

✅ nohup（No Hang Up）

nohup 是 “no hang up” 的缩写，表示「**即使你登出，命令也不会被挂起或中断**」。

基本用法：

nohup コマンド

它默认会把输出写到 nohup.out 文件中。

✅ &（バックグラウンドで実行）

这是将命令放入后台运行的符号。

コマンド &

你可以继续输入其他命令，而这个任务就在后台悄悄运行。

🧠 综合用法：

nohup コマンド > 出力先ファイル &

这就是「**登录后自动运行，并把输出写入文件，还能在登出后继续执行**」的标准写法。

🧪 二、题目解法说明

> 题干：`ping localhost > ping.txt` をログアウト後もバックグラウンドで実行し続けたい。

逐个分析选项：

选项	是否正确	理由
nohup ping localhost > ping.txt &	✅ 正确	正确格式：nohup + 输出重定向 + 后台执行
nohup > ping localhost > ping.txt &	❌ 错误	nohup > 语法错误，> 是用于输出重定向，而不是放在 nohup 后面
nohup && ping localhost > ping.txt &	❌ 错误	&& 表示前一个命令成功后再执行 ping，根本没使用 nohup ping
`nohup		ping localhost > ping.txt &`
nohup ; ping localhost > ping.txt &	❌ 错误	; 表示顺序执行，这样也不能让 ping 继承 nohup 的保护

所以正确答案是：

nohup ping localhost > ping.txt &

📝 三、总结考点 & 记忆方法

✅ 必记知识点：

- nohup：让命令在登出后继续运行
- &：后台运行
- 输出要定向：> filename

🧠 记忆口诀：

> **「nohup 保命，& 偷跑」**
> “nohup + 命令 + 输出 + &” 就能登录不怕断，悄悄干到底！

想要查看后台运行中的命令，可以用：

```
jobs      # 查看后台任务
ps -ef | grep ping   # 查看是否在运行
```

23.🧠 一、从基础概念开始讲解：Xサーバ 和 认证机制

◆ 什么是 X サーバ?

X 是 Linux 系统中使用的**图形显示系统**，例如 GNOME、KDE 背后都依赖 X。
它的运行结构有些特殊：**X Server 是运行在显示设备上的进程，而应用程序（X Client）则通过网络连接它，发送显示请求。**

举个比喻：

> **X Server 就像一块显示屏 + 键鼠，X Client 就是各个程序。**

◆ 为什么需要认证机制?

因为 X Server 可以允许来自「远程」的 X Client 发起连接（比如你在服务器上运行 gedit，窗口会显示在你本地），
如果**没有认证机制，任何人都可以连接到 X Server 并控制你的桌面**，非常危险！

因此，引入了认证机制。常见有两种：

方式	说明	安全性
xhost	允许某台主机连接	比较粗放，按主机判断，**不安全**
xauth	使用认证文件 ~/.Xauthority，按用户授权	**较安全，推荐使用**

🔧 二、xauth 命令详解

✅ xauth 是什么?

xauth 是专门用于**查看、修改 X Server 认证信息**的命令。配合认证文件 ~/.Xauthority 使用。

常用子命令：

命令	作用
xauth list	查看认证列表（本机有哪些可连接的 X Server）
xauth add	手动添加认证信息到认证文件
xauth remove	删除认证信息
xauth extract	导出认证文件
xauth merge	合并认证文件

🧪 三、题目解法说明

> 题干：**Xサーバへの接続に使用される資格情報を表示したり、クライアント認証ファイルを編集するコマンドは?**

这就非常明显是在考：

- 是否理解 X 的认证方式
- 是否知道 xauth 用于处理 ~/.Xauthority

所以正确答案是：

xauth

❌ 错误选项说明

选项	理由
xhost	按主机授权，粗放且安全性差（并且不涉及「编辑认证文件」）
xedit	X 图形编辑器，与认证无关
xeyes	显示两只会跟随鼠标动的眼睛的图形工具
xshow	无此命令，错误选项

📝 四、总结考点 & 记忆法

✅ 必考知识点：

- X Server 是**图形界面控制核心**
- 安全连接 X Server 需要**认证机制**
- xauth 是用于**认证文件**的命令，安全性优于 xhost
- ~/.Xauthority 是认证用的关键文件

🧠 记忆口诀：

> **「xauth 守门，xhost 开门」**
> `xauth` 用钥匙（认证文件）控制进入，
> `xhost` 直接开门让别人进（不安全）

24.🧠 一、基础知识讲解：什么是 ps 命令?

ps 是 Linux 系统中用于**查看当前正在运行的进程（Process）**的命令，类似于 Windows 中的「任务管理器」。

它的全称是 **process status（进程状态）**。

🗂️ 二、ps 的用法与选项分类

🔶 ps 有两种参数风格：

类型	示例	说明
BSD风格（不带 -）	ps aux	经典 UNIX 风格，常见于 FreeBSD 或 macOS 系统
UNIX风格（带 -）	ps -ef	更标准、更清晰的格式，推荐用于脚本中

🧩 查找指定 PID（进程ID）

当你知道某个进程的 PID（比如：1756），你可以用以下方式仅查看它：

命令	含义
ps -p 1756	使用 UNIX 风格的 -p（--pid），显示该 PID 的信息
ps p 1756	使用 BSD 风格的 p（pid），结果类似，但少一列 UID

两者都能正确显示 PID 为 1756 的信息，所以**两个都是正确答案。**

🧪 三、题目解法说明

> 题干：「プロセスIDが1756のプロセスのみの情報を表示したい。」

匹配这个要求的命令必须**只显示一个 PID 的信息**，所以我们筛选一下：

命令	是否正确	原因
ps -p 1756 ✅	正确	标准写法，查指定 PID
ps p 1756 ✅	正确	BSD 风格，也查指定 PID
ps x 1756 ❌	错误	x 是 BSD 的参数，显示所有进程，但 1756 不被识别
ps -e 1756 ❌	错误	-e 显示所有进程，1756 被忽略
ps a 1756 ❌	错误	a 是显示所有与终端有关的进程，也不会筛选出 1756

✅ 正确答案总结：

ps -p 1756
ps p 1756

📌 四、记忆要点与口诀

关键词记忆：

用法	功能	关键词记忆
ps -p PID	查询某个进程	「p 就是 PID」
ps -e	所有进程	e = every
ps aux	所有详细信息（BSD风格）	常用于实际排查

🏷️ 记忆口诀：

> **「p 查单个，e 查全部，x 是无终端」**

オプション	説明
a	他のユーザのプロセスも表示 (xオプションとの併用で全てのプロセスを表示)
f	プロセスの親子関係をツリー状で表示
u	プロセスの実行ユーザ名も表示
x	制御端末の無いデーモン等のプロセスも表示
-e	全てのプロセスを表示
-f	完全なフォーマットでプロセスを表示
-p | p PID	指定したPID(プロセスID)のプロセス情報を表示
-l | l	親プロセスのPID（PPID）や実行優先度を決定するnice値（NI）なども併せて表示

📜 一、常见选项说明（图解补充）

选项	含义	举例用法	说明记忆法
a	显示所有用户的进程	ps a	all users（注意不是 all 进程）
x	显示没有控制终端的进程（如守护进程）	ps x	x = daemon 也能看
e	显示系统中所有进程	ps -e	e = every process
f	以树状（forest）显示父子关系	ps f	f = forest（树）
u	显示执行用户	ps u	u = user
-f	全格式显示（与 BSD 的 f 不一样）	ps -ef	e + f = 所有详细信息
-p PID / p PID	指定显示某个进程的 PID 信息	ps -p 1234	p = process
-l / l	显示更详细（包含 PPID 和 nice 值 NI）	ps -l	l = long format

🧪 三、组合举例说明

命令	功能
ps -ef	标准格式列出所有进程（UNIX 风格）
ps aux	BSD 风格列出所有进程（不带 -）
ps -p 1234	查看 PID 为 1234 的进程
ps fax	以树状方式查看所有无终端的进程
ps -l	查看详细进程状态，包括 PPID / NI

🎯 四、记忆重点与口诀

🌟 最常考的组合命令：

命令	用途	记忆口诀
ps aux	查看全部进程（BSD 风格）	a = all users, u = user, x = daemon
ps -ef	查看全部进程（UNIX 风格）	e = every, f = full format
ps -p 1756	查看指定 PID	p = process
ps fax	查看进程树结构	f = forest, a = all, x = daemon

🏷️ 口诀记忆：

> **「aux 是全员大合照，ef 是官方合照」**

25.🧠 参考知识点讲解：ps -l 与 ps l

✅ 什么是 ps 命令？

ps（Process Status）是用来查看当前系统中运行的进程信息的命令。

✅ 什么是 -l 或 l 选项？

这是 **"长格式（long format）"显示进程信息** 的选项。

使用 ps -l（或 BSD 风格的 ps l）可以多显示一些重要信息，例如：

字段名	含义
F	进程标志（Flags）
UID	用户ID
PID	进程ID
PPID	**父进程ID**（Parent Process ID）✅本题考点
PRI	优先级（Priority）
NI	**nice值**（决定优先级高低）✅本题考点
ADDR	内存地址
SZ	进程占用的内存页数
WCHAN	当前正在等待的内核函数
TTY	终端
TIME	占用的CPU时间
CMD	执行的命令

✅ 题目解法说明

题目：

> 親プロセスのPIDや実行優先度を決定するnice値なども併せて表示するpsコマンドのオプションはどれか？

🔍 题目关键词：

- 「親プロセスのPID（PPID）」
- 「nice値（NI）」

这两个信息只有使用长格式 ps -l 或 ps l 时才会显示。

❌ 错误选项分析：

选项	是否正确	理由
ax	❌	是组合选项，显示所有进程（没有显示 PPID 和 NI）
x	❌	仅显示没有终端的后台进程，不显示 PPID / NI
a	❌	显示所有用户的进程，不包含 PPID / NI

✅ 正确答案：

- -l
- l

✍️ 总结 & 记忆法

📌 本题考点

- 想看到 **PPID**（父进程）与 **NI**（nice 值） → 就要用长格式 l

🧠 记忆方法

> **「l 看 long 信息」**（包含 parent、nice）

26.🧠 参考知识点讲解：systemd 的启动目标（target）

🔗 什么是 systemd 的 target?

在 systemd 系统中，**target（ターゲット）** 是一组服务（Unit）的集合，用于控制系统在启动时或运行中达到的状态。你可以把它当作 SysVinit 的"运行级别（runlevel）"的升级版。

🗂️ target 与旧式 runlevel 的对照表

旧式 Runlevel	systemd Target 名称	含义
0	poweroff.target	关机
1	rescue.target 或 runlevel1.target	单用户模式（最低限服务） ✅
3	multi-user.target	多用户命令行登录（无GUI）
5	graphical.target	多用户 + GUI（图形界面）
6	reboot.target	重启

👉 所以，**rescue.target = runlevel1.target` = 单用户模式（维护用）**

✅ rescue.target 是什么?

rescue.target 是 systemd 提供的最低限度服务启动目标，仅加载：

- 基本文件系统
- 最少的服务
- root shell 登录环境

用途：进行维护、修复系统、重设密码等操作。

✅ 题目解法

题目：

> systemdの動作するシステムにおいて、次回起動時にメンテナンスを行うために最低限のシステムサービス状態で起動させたい。

要点关键词：

- 「次回起動時」
- 「メンテナンス」
- 「最低限のシステムサービス状態」→ 单用户模式！

✅ 正确答案是：

- rescue.target（systemd 的标准单用户目标）
- runlevel1.target（是 rescue.target 的别名）

❌ 错误选项解释：

	A	B
1	**选项**	**理由**
2	single.target	✗ 不存在的 target
3	maintenance.target	✗ 不存在的 target
4	multi-user.target	✗ 是多用户模式（含网络服务），不是最低限模式

✍️ 总结与记忆法

📌 本题考点：

- systemd 的目标（target）系统结构
- 单用户模式用于系统维护

🧠 记忆口诀：

> **“维修救援找rescue，一人操作runlevel1”**
> （rescue 和 runlevel1 就是最低限启动）

27.❓题目：

systemdの動作するシステムにおいて、次回起動時にメンテナンスを行うために最低限のシステムサービス状態で起動させたい。次回起動時のターゲットとしてどれを指定すればよいか（2つ選択）

选项：

- rescue.target ✅
- maintenance.target ❌
- runlevel1.target ✅
- single.target ❌
- multi-user.target ❌

【一】相关知识点

在 systemd 系统中，Linux 的“启动等级”不再使用旧的 runlevel 数字（如 runlevel 1, 3, 5），而是使用所谓的 **“target”** 取而代之。

每一个 target 代表系统的一种运行状态，类似过去的 runlevel：

旧的 runlevel	systemd 的 target	用途说明
0	poweroff.target	关机
1	rescue.target	**单用户模式**（维护模式）
3	multi-user.target	命令行多用户模式（无图形）
5	graphical.target	启动图形界面
6	reboot.target	重启

👉 **重点**：单用户模式（也叫维护模式）是用于系统维护、忘记密码恢复等情况，只有 root 用户登录，且运行的服务最少。

在 systemd 中有两个与 runlevel 1 等价的 target：

- rescue.target：标准的维护模式
- runlevel1.target：为了兼容旧系统，是 rescue.target 的一个软链接

【二】题目解法说明：

题干问的是：**“下次启动时设置为最低限度的系统服务状态（即进入维护模式）”。**

这正是：

- rescue.target ✅（systemd 正式 target）
- runlevel1.target ✅（兼容旧习惯的别名）

其余选项解析如下：

- maintenance.target ❌：系统中 **没有这个 target。**
- single.target ❌：这不是 systemd 中有效的 target 名称。
- multi-user.target ❌：这是正常服务全部启动的状态，不是维护模式。

【三】总结考点与记忆方法：

🎯 考点：

- systemd 的 target 是 runlevel 的新版本
- 进入维护模式用的是 rescue.target 或其别名 runlevel1.target

🧠 记忆法：

- **救援模式 = rescue（就是要去“救”系统）**
- **runlevel1 = 1人操作，最少服务**
- 把这两个看作是“最低启动等级”即可。

28.【一】相关知识点讲解（从基础开始）

在使用 systemd 的 Linux 系统中，系统启动时会根据一个“目标（target）”来决定要启动哪些服务和环境。这个目标文件实际上是一个指向具体 target 的**符号链接**（default.target）。

什么是 Target?

- Target 是 systemd 中用来**管理服务组和系统状态**的单位。
- 最常见的几个 target：
  - graphical.target：图形界面（等同于旧系统的 runlevel 5）
  - multi-user.target：命令行多用户模式（等同 runlevel 3）
  - rescue.target：单用户维护模式（等同 runlevel 1）

如何查看下次启动时的目标（default.target 指向谁）？

使用命令：

systemctl get-default

这个命令会显示当前系统设置的“默认启动 target”。例如：

```
$ systemctl get-default
graphical.target
```

【二】题目解法说明

❓题目：

systemdが稼働するシステムにおいて、次回起動時のターゲットが確認できるsystemctlのサブコマンドはどれか

✅正解：

- get-default

❌错误选项解释：

- set-default：是用来**设置**下次启动目标的，不是查询
- get-next：没有这个命令
- nextboot：不是 systemctl 的命令
- default-target：不是 systemctl 的子命令

【三】总结考点与记忆方法

🎯考点总结：

- get-default：查看当前设定的开机启动 target
- set-default：设置开机启动 target

🧠记忆技巧：

- “get” 是“获取”的意思 → get-default = 获取默认目标
- “set” 是“设置”的意思 → set-default = 设置默认目标

你可以记成：

> **get 看，set 改**

29.【一】基础知识讲解：systemd 的启动流程和 default.target 的作用

🌟 什么是 systemd?

- systemd 是 Linux 系统中的初始化系统（init system），用于在系统启动时管理服务、挂载文件系统、设置网络等。
- systemd 使用 **unit 文件** 来管理服务或系统状态。每个 unit 有对应的 .service、.target 等扩展名。

📌 什么是 default.target?

在 systemd 系统中：

- **系统启动时会首先读取 default.target 所指向的 Unit。**
- 这个文件位于目录：/etc/systemd/system/default.target
- 它是一个**符号链接**，通常会指向实际的 target，比如：
  - graphical.target：图形界面启动
  - multi-user.target：命令行登录界面
  - rescue.target：最小化单用户模式（维护模式）

【二】题目解析：空格填空判断

题目中是两行命令：

cd _____
ls -l _____

目的是要“确认 next boot 时 systemd 会使用哪个启动目标（相当于 SysVinit 的 runlevel）”。

✅ 正确的答案：

- 第一空应是：/etc/systemd/system
- 第二空应是：default.target

🔍 理由：

- /etc/systemd/system/default.target 是一个符号链接，用于指定系统下次启动时使用哪个 target（runlevel 相当值）
- 查看这个链接，就可以判断当前设置的启动级别

举个实际例子：

cd /etc/systemd/system
ls -l default.target
default.target -> /usr/lib/systemd/system/multi-user.target

这里你就能知道系统下次会进入的是 multi-user.target。

【三】考点总结与记忆法

概念	说明
/etc/systemd/system	存放默认启动的 target 的目录
default.target	指向实际启动模式的符号链接文件
systemctl get-default	查看 default.target 当前指向的目标

🎓 记忆法：

> 想知道下次开机会进哪个"模式"？记得去 `/etc/systemd/system/default.target` 看！

你也可以通过这个命令直接查看：

systemctl get-default

📘 SysVinit 与 systemd 的运行级别（Runlevel）对照表

Runlevel	systemd Target	用途说明
0	poweroff.target	系统停止
1	rescue.target / runlevel1.target	单用户模式（维护用）
2	runlevel2.target	多用户模式（无网络）
3	multi-user.target / runlevel3.target	多用户模式（有网络）
4	runlevel4.target	用户自定义使用
5	graphical.target / runlevel5.target	图形界面模式
6	reboot.target	系统重启

💡记忆方法建议：

- **0 与 6** 是终点和起点：0 是关闭系统，6 是重启系统。
- **1 是单用户模式（Rescue）**，用于修复系统。
- **3 是服务器常用模式（无图形）**。
- **5 是桌面常用模式（有图形）**。
- systemd 下的 multi-user.target、graphical.target、rescue.target 等是关键考点，务必记住它们的作用和对应关系。

30.🛠题目：

> systemdを管理するためのコマンドはどれか
>
> - systemd-control
> - systemctl
> - unitctl
> - initctl
> - journalctl

✅ 正解：**systemctl**

🔍 一、基础知识讲解：systemd 和 systemctl 是什么？

🔧 什么是 systemd？

- **systemd** 是 Linux 系统中用于“管理系统启动过程和服务”的一种初始化系统（init system）。
- 它取代了传统的 **SysVinit**，几乎是所有现代 Linux 发行版（如 CentOS 7、Ubuntu 18.04 之后）默认使用的系统初始化管理器。

🧰 什么是 systemctl？

- systemctl 是 **管理 systemd 的命令行工具。**
- 你可以用它来：
  - 启动 / 停止服务（Unit）
  - 设置服务的自启
  - 更改系统启动级别（Target）
  - 查看服务状态
  - 重启系统，等等

📌 语法示例：

bash

复制编辑

```
systemctl start nginx       # 启动 nginx 服务
systemctl stop sshd         # 停止 ssh 服务
systemctl status firewalld  # 查看防火墙状态
systemctl enable httpd      # 设置 httpd 开机自启
```

📝 二、题目解法说明

选项分析：

选项	判断	说明
systemctl	✅ 正确	systemd 的官方管理工具
journalctl	❌ 错误	是查看 systemd 的日志，不是“管理工具”
initctl	❌ 错误	是旧的 Upstart 系统用的命令
systemd-control	❌ 错误	根本不存在这个命令，是干扰项
unitctl	❌ 错误	也不存在，是干扰项

所以选项只有 **systemctl** 是正确的。

🎯 三、总结：考点与记忆法

✅ 本题考点

- systemd 与 systemctl 的基础概念
- 常用命令识别能力
- 干扰项识别能力（虚构命令）

📌 记忆法

- systemd → 对应命令是 systemctl
  - d 是管理器（daemon），ctl 是控制命令（control）
- journalctl = 日志查看器（像“日记”journal）
- 所有带 -control、unitctl 的都是伪命令