



openEuler

1.0

管理员指南

发布日期 2020-02-08

目 录

法律声明.....	vi
前言.....	vii
1 基础配置.....	8
1.1 通过命令设置	8
1.1.1 设置语言环境	8
1.1.2 设置键盘	9
1.1.3 设置日期和时间	10
1.1.3.1 使用 <code>timedatectl</code> 命令设置.....	10
1.1.3.2 使用 <code>date</code> 命令设置.....	12
1.1.3.3 使用 <code>hwclock</code> 命令设置.....	13
2 查看系统信息.....	15
3 管理用户.....	16
3.1 增加用户	16
3.2 修改用户账号	17
3.3 删除用户	18
3.4 管理员账户授权	18
3.4.1 为普通用户分配特权	18
4 使用 DNF 管理软件包.....	21
4.1 配置 DNF	21
4.1.1 修改配置文件	21
4.1.2 创建本地软件源仓库	23
4.1.3 添加、启用和禁用软件源	23
4.2 管理软件包	24
4.3 管理软件包组	26
4.4 检查并更新	28
5 管理服务.....	30
5.1 简介	30
5.2 特性说明	31

5.3 管理系统服务	33
5.4 改变运行级别	38
5.5 关闭、暂停和休眠系统	40
6 管理进程.....	42
6.1 管理系统进程	42
6.1.1 调度启动进程	42
6.1.1.1 定时运行一批程序 (at)	42
6.1.1.2 周期性运行一批程序 (cron)	44
6.1.2 挂起/恢复进程	46
6.2 查看进程	46
7 配置网络.....	50
7.1 配置 IP.....	50
7.1.1 使用 nmcli	50
7.1.1.1 nmcli 介绍	50
7.1.1.2 设置网络连接	51
7.1.1.2.1 配置动态 IP 连接.....	51
7.1.1.2.2 配置静态 IP 连接.....	52
7.1.1.2.3 添加 Wi-Fi 连接.....	53
7.1.1.2.4 更改属性	54
7.1.1.3 配置静态路由	54
7.1.2 使用命令行	54
7.1.2.1 通过 ifcfg 文件配置网络	54
7.1.2.2 使用 ip 命令配置网络	55
7.1.2.3 静态路由及默认网关	56
7.2 配置主机名	57
7.2.1 简介	57
7.2.2 使用 hostnamectl 配置主机名	57
7.2.3 使用 nmcli 配置主机名	58
7.3 配置网络绑定	59
7.3.1 使用 nmcli	59
7.3.2 使用命令行	59
7.3.2.1 检查是否已安装 Bonding 内核模块.....	59
7.3.2.2 创建频道绑定接口	59
7.3.2.3 创建从属接口	60
7.3.2.4 激活频道绑定	60
7.3.2.5 创建多个绑定	61
7.4 IPv6 使用差异说明 (vs IPv4)	61
7.4.1 约束限制	61
7.4.2 配置说明	62

7.4.2.1 设置接口设备 MTU 值.....	62
7.4.2.2 有状态自动配置 IPv6 地址.....	63
7.4.2.3 内核支持 socket 相关系统调用.....	64
7.4.2.4 IPv4 的 dhclient 守护进程持久化配置.....	65
7.4.2.5 iproute 相关命令配置 IPv4 与 IPv6 时的差异说明.....	66
7.4.2.6 NetworkManager 服务配置差异说明.....	68
7.4.3 FAQ.....	70
7.4.3.1 iscsi-initiator-utils 不支持登录 fe80 IPv6 地址.....	70
7.4.3.2 网卡 down 掉之后, IPv6 地址丢失.....	70
7.4.3.3 bond 口已具有多个 IPv6 地址时, 添加或删除 IPv6 地址耗时过久.....	71
7.4.3.4 Rsyslog 在 IPv4 和 IPv6 混合使用场景中日志传输延迟.....	71
8 搭建服务.....	72
8.1 搭建 repo 服务器.....	72
8.1.1 概述.....	72
8.1.2 创建/更新本地 repo 源.....	72
8.1.2.1 获取 ISO 镜像.....	72
8.1.2.2 挂载 ISO 创建 repo 源.....	72
8.1.2.3 创建本地 repo 源.....	73
8.1.2.4 更新 repo 源.....	73
8.1.3 部署远端 repo 源.....	73
8.1.3.1 nginx 安装与配置.....	73
8.1.3.2 启动 nginx 服务.....	74
8.1.3.3 repo 源部署.....	75
8.1.4 使用 repo 源.....	76
8.1.4.1 repo 配置为 yum 源.....	76
8.1.4.2 repo 优先级.....	77
8.1.4.3 dnf 相关命令.....	77
8.2 搭建 FTP 服务器.....	78
8.2.1 总体介绍.....	78
8.2.2 使用 vsftpd.....	79
8.2.3 配置 vsftpd.....	80
8.2.3.1 vsftpd 配置文件介绍.....	80
8.2.3.2 默认配置说明.....	81
8.2.3.3 配置本地时间.....	82
8.2.3.4 配置欢迎信息.....	83
8.2.3.5 配置系统帐号登录权限.....	83
8.2.4 验证 FTP 服务是否搭建成功.....	84
8.2.5 配置防火墙.....	84
8.2.6 传输文件.....	84

8.3 搭建 web 服务器.....	86
8.3.1 概述	86
8.3.2 管理 httpd.....	87
8.3.3 配置文件说明	88
8.3.4 管理模块和 SSL	88
8.3.5 验证 web 服务是否搭建成功.....	90
9 FAQ.....	92
9.1 使用 systemctl 和 top 命令查询 libvirtd 服务占用内存不同.....	92
9.2 设置 RAID0 卷，参数 stripsize 设置为 4 时出错.....	93
9.3 使用 rpmbuild 编译 mariadb 失败.....	93

法律声明

版权所有 © 2020 华为技术有限公司。

您对“本文档”的复制、使用、修改及分发受知识共享(Creative Commons)署名—相同方式共享 4.0 国际公共许可协议(以下简称“CC BY-SA 4.0”)的约束。为了方便用户理解，您可以通过访问 <https://creativecommons.org/licenses/by-sa/4.0/> 了解 CC BY-SA 4.0 的概要 (但不是替代)。CC BY-SA 4.0 的完整协议内容您可以访问如下网址获取：
<https://creativecommons.org/licenses/by-sa/4.0/legalcode>。

商标声明

openEuler 为华为技术有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

免责声明

本文档仅作为使用指导，除非适用法强制规定或者双方有明确书面约定，华为技术有限公司对本文档中的所有陈述、信息和建议不做任何明示或默示的声明或保证，包括但不限于不侵权，时效性或满足特定目的的担保。

前言

概述



本文档提供了 openEuler 系统常用的管理员操作，方便管理员更好地使用 openEuler。

读者对象

本文档适用于所有使用 openEuler 系统的管理员。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

1 基础配置

1.1 通过命令设置

1.1 通过命令设置

1.1.1 设置语言环境

您可以通过 `localectl` 修改系统的语言环境，对应的参数设置保存在 `/etc/locale.conf` 文件中。这些参数会在系统启动过程中被 `systemd` 的守护进程读取。

显示当前语言环境状态

显示当前语言环境，命令如下：

```
localectl status
```

例如显示系统当前的设置，命令和输出如下：

```
$ localectl status
System Locale: LANG=zh_CN.UTF-8
VC Keymap: cn
X11 Layout: cn
```

列出可用的语言环境

显示当前可用的语言环境，命令如下：

```
localectl list-locales
```

例如显示当前系统中所有可用的中文环境，命令和输出如下：

```
$ localectl list-locales | grep zh
zh_CN.UTF-8
```

设置语言环境

要设置语言环境，在 `root` 权限下执行如下命令，其中 `locale` 是您要设置的语言类型，请根据实际情况修改。


```
localectl set-locale LANG=locale
```

例如设置为简体中文语言环境，在 root 权限下执行如下命令：

```
# localectl set-locale LANG=zh_CN.UTF-8
```

📖 说明

修改后需要重新登录或者执行如下命令刷新配置文件，使修改生效。

```
source /etc/locale.conf
```

1.1.2 设置键盘

您可以通过 `localectl` 修改系统的键盘设置，对应的参数设置保存在 `/etc/locale.conf` 文件中。这些参数，会在系统启动的早期被 `systemd` 的守护进程读取。

显示当前设置

显示当前键盘设置，命令如下：

```
localectl status
```

例如显示系统当前的设置，命令和输出如下：

```
$ localectl status
System Locale: LANG=zh_CN.UTF-8
VC Keymap: cn
X11 Layout: cn
```

列出可用的键盘布局

显示当前可用的键盘布局，命令如下：

```
localectl list-keymaps
```

例如显示系统当前的中文键盘布局，命令和输出如下：

```
$ localectl list-keymaps | grep cn
cn
```

设置键盘布局

设置键盘布局，在 root 权限下执行如下命令，其中 `map` 是您想要设置的键盘类型，请根据实际情况修改：

```
localectl set-keymap map
```

此时设置的键盘布局同样也会应用到图形界面中。

设置完成后，查看当前状态：

```
$ localectl status
System Locale: LANG=zh_CN.UTF-8
VC Keymap: cn
X11 Layout: us
```

1.1.3 设置日期和时间

本节介绍如何通过 `timedatectl`、`date`、`hwclock` 命令来设置系统的日期、时间和时区等。

1.1.3.1 使用 `timedatectl` 命令设置

显示日期和时间

显示当前的日期和时间，命令如下：

```
timedatectl
```

例如显示系统当前的日期和时间，命令和输出如下：

```
$ timedatectl
      Local time: Mon 2019-09-30 04:05:00 EDT
      Universal time: Mon 2019-09-30 08:05:00 UTC
      RTC time: Mon 2019-09-30 08:05:00
      Time zone: America/New York (EDT, -0400)
System clock synchronized: no
      NTP service: inactive
      RTC in local TZ: no
```

通过远程服务器进行时间同步

您可以启用 NTP 远程服务器进行系统时钟的自动同步。是否启用 NTP，可在 `root` 权限下执行如下命令进行设置。其中 *boolean* 可取值 `yes` 和 `no`，分别表示启用和不启用 NTP 进行系统时钟自动同步，请根据实际情况修改。

说明

若启用了 NTP 远程服务器进行系统时钟自动同步，则不能手动修改日期和时间。若需要手动修改日期或时间，则需确保已经关闭 NTP 系统时钟自动同步。可执行 `timedatectl set-ntp no` 命令进行关闭。

```
timedatectl set-ntp boolean
```

例如开启自动远程时间同步，命令如下：

```
# timedatectl set-ntp yes
```

修改日期

说明

修改日期前，请确保已经关闭 NTP 系统时钟自动同步。

修改当前的日期，在 `root` 权限下执行如下命令，其中 *YYYY* 代表年份，*MM* 代表月份，*DD* 代表某天，请根据实际情况修改：

```
timedatectl set-time YYYY-MM-DD
```

例如修改当前的日期为 2019 年 8 月 14 号，命令如下：

```
# timedatectl set-time '2019-08-14'
```

修改时间

📖 说明

修改时间前，请确保已经关闭 NTP 系统时钟自动同步。

修改当前的时间，在 `root` 权限下执行如下命令，其中 *HH* 代表小时，*MM* 代表分钟，*SS* 代表秒，请根据实际情况修改：

```
timedatectl set-time HH:MM:SS
```

例如修改当前的时间为 15 点 57 分 24 秒，命令如下：

```
# timedatectl set-time 15:57:24
```

修改时区

显示当前可用时区，命令如下：

```
timedatectl list-timezones
```

要修改当前的时区，在 `root` 权限下执行如下命令，其中 *time_zone* 是您想要设置的时区，请根据实际情况修改：

```
timedatectl set-timezone time_zone
```

例如修改当前的时区，首先查询所在地域的可用时区，此处以 *Asia* 为例：

```
# timedatectl list-timezones | grep Asia
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
Asia/Aqtobe
Asia/Ashgabat
Asia/Baghdad
Asia/Bahrain
.....
Asia/Seoul
Asia/Shanghai
Asia/Singapore
Asia/Srednekolymsk
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Thimphu
Asia/Tokyo
```

然后修改当前的时区为“*Asia/Shanghai*”，命令如下：

```
# timedatectl set-timezone Asia/Shanghai
```

1.1.3.2 使用 date 命令设置

显示当前的日期和时间

显示当前的日期和时间，命令如下：

```
date
```

默认情况下，date 命令显示本地时间。要显示 UTC 时间，添加 --utc 或 -u 参数：

```
date --utc
```

要自定义对应的输出信息格式，添加 +"format" 参数：

```
date +"format"
```

表1-1 参数说明

格式参数	说明
%H	小时以 HH 格式（例如 17）。
%M	分钟以 MM 格式（例如 37）。
%S	秒以 SS 格式（例如 25）。
%d	日期以 DD 格式（例如 15）。
%m	月份以 MM 格式（例如 07）。
%Y	年份以 YYYY 格式（例如 2019）。
%Z	时区缩写（例如 CEST）。
%F	日期整体格式为 YYYY-MM-DD（例如 2019-7-15），等同 %Y-%m-%d。
%T	时间整体格式为 HH:MM:SS（例如 18:30:25），等同 %H:%M:%S。

实际使用示例如下：

- 显示当前的日期和本地时间。

```
$ date
2019年 08月 17日 星期六 17:26:34 CST
```

- 显示当前的日期和 UTC 时间。

```
$ date --utc
2019年 08月 17日 星期六 09:26:18 UTC
```

- 自定义 date 命令的输出。

```
$ date +"%Y-%m-%d %H:%M"
2019-08-17 17:24
```

修改时间

要修改当前的时间，添加--set 或者-s 参数。在 root 权限下执行如下命令，其中 *HH* 代表小时，*MM* 代表分钟，*SS* 代表秒，请根据实际情况修改：

```
date --set HH:MM:SS
```

默认情况下，date 命令设置本地时间。要设置 UTC 时间，添加--utc 或-u 参数：

```
date --set HH:MM:SS --utc
```

例如修改当前的时间为 23 点 26 分 00 秒，在 root 权限下执行如下命令：

```
# date --set 23:26:00
```

修改日期

修改当前的日期，添加--set 或者-s 参数。在 root 权限下执行如下命令，其中 *YYYY* 代表年份，*MM* 代表月份，*DD* 代表某天，请根据实际情况修改：

```
date --set YYYY-MM-DD
```

例如修改当前的日期为 2019 年 11 月 2 日，命令如下：

```
# date --set 2019-11-02
```

1.1.3.3 使用 hwclock 命令设置

可以使用 hwclock 命令设置硬件时钟 RTC (Real Time Clock)。

硬件时钟和系统时钟

Linux 将时钟分为：

- 系统时钟 (System Clock)：当前 Linux Kernel 中的时钟。
- 硬件时钟 RTC：主板上由电池供电的主板硬件时钟，该时钟可以在 BIOS 的 "Standard BIOS Feature" 项中进行设置。

当 Linux 启动时，会读取硬件时钟，并根据硬件时间来设置系统时间。

显示日期和时间

显示当前硬件的日期和时间，在 root 权限下执行如下命令：

```
hwclock
```

例如显示当前硬件的日期和时间，命令和输出如下：

```
# hwclock
2019-08-26 10:18:42.528948+08:00
```

设置日期和时间

修改当前硬件的日期和时间，在 root 权限下执行如下命令，其中 *dd* 表示日，*mm* 表示月份，*yyyy* 表示年份，*HH* 表示小时，*MM* 表示分钟，请根据实际情况修改：

```
hwclock --set --date "dd mm yyyy HH:MM"
```

例如修改当前的时间为 2019 年 10 月 21 日 21 点 17 分，命令如下：

```
# hwclock --set --date "21 Oct 2019 21:17" --utc
```

2 查看系统信息

- 查看系统信息，命令如下：

```
cat /etc/os-release
```

例如，命令和输出如下：

```
# cat /etc/os-release
NAME="openEuler"
VERSION="1.0 ()"
ID="openEuler"
VERSION ID="1.0"
PRETTY_NAME="openEuler 1.0 ()"
ANSI_COLOR="0;31"
```

- 查看系统相关的资源信息。
查看 CPU 信息，命令如下：

```
lscpu
```

查看内存信息，命令如下：

```
free
```

查看磁盘信息，命令如下：

```
fdisk -l
```

3 管理用户

在 Linux 中，每个普通用户都有一个账户，包括用户名、密码和主目录等信息。除此之外，还有一些系统本身创建的特殊用户，它们具有特殊的意义，其中最重要的是管理员账户，默认用户名是 `root`。同时 Linux 也提供了用户组，使每一个用户至少属于一个组，从而便于权限管理。

用户和用户组管理是系统安全管理的重要组成部分，本章主要介绍 openEuler 提供的用户管理和组管理命令，以及为普通用户分配特权的方法。

3.1 增加用户

3.2 修改用户账号

3.3 删除用户

3.4 管理员账户授权

3.1 增加用户

useradd 命令

在 `root` 权限下，通过 `useradd` 命令可以为系统添加新用户信息，其中 *options* 为相关参数，*user_name* 为用户名称。

```
useradd [options] user_name
```

用户信息文件

与用户账号信息有关的文件如下：

- `/etc/passwd`：用户账号信息文件。
- `/etc/shadow`：用户账号信息加密文件。
- `/etc/group`：组信息文件。
- `/etc/default/useradd`：定义默认设置文件。
- `/etc/login.defs`：系统广义设置文件。
- `/etc/skel`：默认的初始配置文件目录。

创建用户实例

例如新建一个用户名为 `user_example` 的用户，在 `root` 权限下执行如下命令：

```
# useradd user_example
```

说明

没有任何提示，表明用户建立成功。这时并没有设置用户的口令，请使用 `passwd` 命令修改用户的密码，没有设置密码的新账号不能登录系统。

使用 `id` 命令查看新建的用户信息，命令如下：

```
# id user example
uid=502(user_example) gid=502(user_example) groups=502(user_example)
```

修改用户 `user_example` 的密码：

```
# passwd user_example
```

根据提示两次输入新用户的密码，完成密码更改。过程如下：

```
# passwd user example
Changing password for user user example.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

说明

若打印信息中出现“`BAD PASSWORD: The password fails the dictionary check - it is too simplistic/sytematic`”，表示设置的密码过于简单，建议设置复杂度较高的密码。

3.2 修改用户账号

修改密码

普通用户可以用 `passwd` 修改自己的密码，只有管理员才能用 `passwd username` 为其他用户修改密码。

修改用户 shell 设置

使用 `chsh` 命令可以修改自己的 shell，只有管理员才能用 `chsh username` 为其他用户修改 shell 设置。

用户也可以使用 `usermod` 命令修改 shell 信息，在 `root` 权限下执行如下命令，其中 `new_shell_path` 为目标 shell 路径，`username` 为要修改用户的用户名，请根据实际情况修改：

```
usermod -s new_shell_path username
```

例如，将用户 `user_example` 的 shell 改为 `csh`，命令如下：

```
# usermod -s /bin/csh user_example
```

修改主目录

- 修改主目录，可以在 `root` 权限下执行如下命令，其中 `new_home_directory` 为已创建的目标主目录的路径，`username` 为要修改用户的用户名，请根据实际情况修改：

```
usermod -d new_home_directory username
```

- 如果想将现有主目录的内容转移到新的目录，应该使用 `-m` 选项，命令如下：

```
usermod -d new_home_directory -m username
```

修改 UID

修改用户 ID，在 `root` 权限下执行如下命令，其中 `UID` 代表目标用户 ID，`username` 代表用户名，请根据实际情况修改：

```
usermod -u UID username
```

该用户主目录中所拥有的文件和目录都将自动修改 UID 设置。但是，对于主目录外所拥有的文件，只能使用 `chown` 命令手动修改所有权。

修改账号的有效期

如果使用了影子口令，则可以在 `root` 权限下，执行如下命令来修改一个账号的有效期，其中 `MM` 代表月份，`DD` 代表某天，`YY` 代表年份，`username` 代表用户名，请根据实际情况修改：

```
usermod -e MM/DD/YY username
```

3.3 删除用户

在 `root` 权限下，使用 `userdel` 命令可删除现有用户。

例如，删除用户 `Test`，命令如下：

```
# userdel Test
```

如果想同时删除该用户的主目录以及其中所有内容，要使用 `-r` 参数递归删除。

📖 说明

不建议直接删除已经进入系统的用户，如果需要强制删除，请使用 `userdel -f Test` 命令。

3.4 管理员账户授权

3.4.1 为普通用户分配特权

使用 `sudo` 命令可以允许普通用户执行管理员账户才能执行的命令。

`sudo` 命令允许已经在 `/etc/sudoers` 文件中指定的用户运行管理员账户命令。例如，一个已经获得许可的普通用户可以运行如下命令：

```
sudo /usr/sbin/useradd newuser1
```

实际上，`sudo` 的配置完全可以指定某个已经列入 `/etc/sudoers` 文件的普通用户可以做什么，不可以做什么。

`/etc/sudoers` 的配置行如下所示。

- 空行或注释行（以#字符打头）：无具体功能的行。
- 可选的主机别名行：用来创建主机列表的简称。必须以 `Host_Alias` 关键词开头，列表中的主机必须用逗号隔开，如：

```
Host_Alias linux=ted1,ted2
```

其中 `ted1` 和 `ted2` 是两个主机名，可使用 `linux`（别名）称呼它们。

- 可选的用户别名行：用来创建用户列表的简称。用户别名行必须以 `User_Alias` 关键词开头，列表中的用户名必须以逗号隔开。其格式同主机别名行。
- 可选的命令别名行：用来创建命令列表的简称。必须以 `Cmnd_Alias` 开头，列表中的命令必须用逗号隔开。
- 可选的运行方式别名行：用来创建用户列表的简称。不同的是，使用这样的别名可以告诉 `sudo` 程序以列表中某一用户的身份来运行程序。
- 必要的用户访问说明行。

用户访问的说明语法如下：

```
user host = [ run as user ] command list
```

在 `user` 处指定一个真正的用户名或定义过的别名，`host` 也可以是一个真正的主机名或者定义过的主机别名。默认情况下，`sudo` 执行的所有命令都是以 `root` 身份执行。如果您想使用其他身份可以指定。`command list` 可以是以逗号分隔的命令列表，也可以是一个已经定义过的别名，如：

```
ted1 ted2=/sbin/shutdown
```

这一句说明 `ted1` 可以在 `ted2` 主机上运行关机命令。

```
newuser1 ted1=(root) /usr/sbin/useradd,/usr/sbin/userdel
```

这一句说明 `ted1` 主机上的 `newuser1` 具有以 `root` 用户权限执行 `useradd`，`userdel` 命令的功能。

📖 说明

- 可以在一行定义多个别名，中间用冒号 (:) 隔开。
- 可在命令或命令别名之前加上感叹号 (!)，使该命令或命令别名无效。
- 有两个关键词：`ALL` 和 `NOPASSWD`。`ALL` 意味着“所有”（所有文件、所有主机或所有命令），`NOPASSWD` 意味着不用密码。
- 通过修改用户访问，将普通用户的访问权限修改为同 `root` 一样，则可以给普通用户分配特权。

下面是一个 `sudoers` 文件的例子：

```
#sudoers files
#User alias specification
User_Alias ADMIN=ted1:POWERUSER=globus,ted2
#user privilege specification
ADMIN ALL=ALL
POWERUSER ALL=ALL,!/bin/su
```

其中：

- `User_Alias ADMIN=tel1:POWERUSER=glabus,tel2`
定义了两个别名 ADMIN 和 POWERUSER
- `ADMIN ALL=ALL`
说明在所有主机上，ADMIN 用户都可以以 root 身份执行所有命令
- `POWERUSER ALL=ALL,!/bin/su`
给 POWERUSER 用户除了运行 su 命令外等同 ADMIN 的权限

4 使用 DNF 管理软件包

DNF 是一款 Linux 软件包管理工具，用于管理 RPM 软件包。DNF 可以查询软件包信息，从指定软件库获取软件包，自动处理依赖关系以安装或卸载软件包，以及更新系统到最新可用版本。

📖 说明

- DNF 与 YUM 完全兼容，提供了 YUM 兼容的命令行以及为扩展和插件提供的 API。
- 使用 DNF 需要管理员权限，本章所有命令需要在管理员权限下执行。

4.1 配置 DNF

4.2 管理软件包

4.3 管理软件包组

4.4 检查并更新

4.1 配置 DNF

4.1.1 修改配置文件

DNF 的主要配置文件是 `/etc/dnf/dnf.conf`，该文件中“main”部分保存着 DNF 的全局设置；用户可以在该文件中通过添加一个或者多个“repository”部分的方式来设置需要安装的软件源位置。

另外，在 `/etc/yum.repos.d` 目录中保存着一个或多个 repo 源相关文件，它们定义了不同的“repository”。

软件源的配置一般有两种方式，一种是直接配置 `/etc/dnf/dnf.conf` 文件，另外一种是在 `/etc/yum.repos.d` 目录下增加 `.repo` 文件。

修改 main 部分

`/etc/dnf/dnf.conf` 文件包含的“main”部分，配置文件示例如下：

```
[main]
gpgcheck=0
```

```
installonly_limit=3  
clean_requirements_on_remove=True  
best=True
```

常用选项说明：

表4-1 main 参数说明

参数	说明
cachedir	缓存目录，该目录用于存储 RPM 包和数据库文件。
keepcache	可选值是 1 和 0，表示是否要缓存已安装成功的那些 RPM 包及头文件，默认值为 0，即不缓存。
debuglevel	设置 dnf 生成的 debug 信息。取值范围：[0-10]，数值越大输出越详细的 debug 信息。默认值为 2，设置为 0 表示不输出 debug 信息。
clean_requirements_on_remove	删除在 dnf remove 期间不再使用的依赖项，如果软件包是通过 DNF 安装的，而不是通过显式用户请求安装的，则只能通过 clean_requirements_on_remove 删除软件包，即它是作为依赖项引入的。默认值为 True。
best	升级包时，总是尝试安装其最高版本，如果最高版本无法安装，则提示无法安装的原因并停止安装。默认值为 True。
obsoletes	可选值 1 和 0，设置是否允许更新陈旧的 RPM 包。默认值为 1，表示允许更新。
gpgcheck	可选值 1 和 0，设置是否进行 gpg 校验。默认值为 1，表示需要进行校验。
plugins	可选值 1 和 0，表示启用或禁用 dnf 插件。默认值为 1，表示启用 dnf 插件。
installonly_limit	设置可以同时安装“installonlypkgs”指令列出包的数量。默认值为 3，不建议降低此值。

修改 repository 部分

repository 部分允许您定义定制化的软件源仓库，各个仓库的名称不能相同，否则会引起冲突。下面是[repository]部分的一个最小配置示例：

```
[repository]  
name=repository name  
baseurl=repository_url
```

选项说明：

表4-2 repository 参数说明

参数	说明
name=repository_name	软件仓库（repository）描述的字符串。
baseurl=repository_url	软件仓库（repository）的地址。 <ul style="list-style-type: none">• 使用 http 协议的网络位置：例如 http://path/to/repo• 使用 ftp 协议的网络位置：例如 ftp://path/to/repo• 本地位置：例如 file:///path/to/local/repo

显示当前配置

- 要显示当前的配置信息：

```
dnf config-manager --dump
```

- 要显示相应软件源的配置，首先查询 repo id：

```
dnf repolist
```

然后执行如下命令，显示对应 id 的软件源配置，其中 *repository* 为查询得到的 repo id：

```
dnf config-manager --dump repository
```

- 您也可以使用一个全局正则表达式，来显示所有匹配部分的配置：

```
dnf config-manager --dump glob_expression
```

4.1.2 创建本地软件源仓库

要建立一个本地软件源仓库，请按照下列步骤操作。

1. 安装 createrepo 软件包。在 root 权限下执行如下命令：

```
dnf install createrepo
```

2. 将需要的软件包复制到一个目录下，如/mnt/local_repo/。

3. 创建软件源，执行以下命令：

```
createrepo --database /mnt/local_repo
```

4.1.3 添加、启用和禁用软件源

本节将介绍如何通过“dnf config-manager”命令添加、启用和禁用软件源仓库。

添加软件源

要定义一个新的软件源仓库，您可以在 `/etc/dnf/dnf.conf` 文件中添加“repository”部分，或者在 `/etc/yum.repos.d/` 目录下添加“.repo”文件进行说明。建议您通过添加“.repo”的方式，每个软件源都有自己对应的“.repo”文件，以下介绍该方式的操作方法。

要在您的系统中添加一个这样的源，请在 root 权限下执行如下命令，执行完成之后会在 `/etc/yum.repos.d/` 目录下生成对应的 repo 文件。其中 *repository_url* 为 repo 源地址，

详情请参见“使用 DNF 管理软件包 > 配置 DNF > 修改配置文件”章节中的“repository 参数说明”。

```
dnf config-manager --add-repo repository_url
```

启用软件源

要启用软件源，请在 root 权限下执行如下命令，其中 *repository* 为新增.repo 文件中的 repo id（可通过 `dnf repolist` 查询）：

```
dnf config-manager --set-enable repository
```

您也可以使用一个全局正则表达式，来启用所有匹配的软件源。其中 *glob_expression* 为对应的正则表达式，用于同时匹配多个 repo id：

```
dnf config-manager --set-enable glob_expression
```

禁用软件源

要禁用软件源，请在 root 权限下执行如下命令：

```
dnf config-manager --set-disable repository
```

同样的，您也可以使用一个全局正则表达式来禁用所有匹配的软件源：

```
dnf config-manager --set-disable glob_expression
```

4.2 管理软件包

使用 `dnf` 能够让您方便的进行查询、安装、删除软件包等操作。

搜索软件包

您可以使用 `rpm` 包名称、缩写或者描述搜索需要的 RPM 包，使用命令如下：

```
dnf search term
```

示例如下：

```
$ dnf search httpd
===== N/S matched: httpd
=====
httpd.aarch64 : Apache HTTP Server
httpd-devel.aarch64 : Development interfaces for the Apache HTTP server
httpd-manual.noarch : Documentation for the Apache HTTP server
httpd-tools.aarch64 : Tools for use with the Apache HTTP Server
libmicrohttpd.aarch64 : Lightweight library for embedding a webserver in
applications
mod_auth_mellon.aarch64 : A SAML 2.0 authentication module for the Apache Httpd
Server
mod_dav_svn.aarch64 : Apache httpd module for Subversion server
```


列出软件包清单

要列出系统中所有已安装的以及可用的 RPM 包信息，使用命令如下：

```
dnf list all
```

要列出系统中特定的 RPM 包信息，使用命令如下：

```
dnf list glob_expression...
```

示例如下：

```
$ dnf list httpd
Available Packages
httpd.aarch64          2.4.34-8.h5.oe1      Local
```

显示 RPM 包信息

要显示一个或者多个 RPM 包信息，使用命令如下：

```
dnf info package_name...
```

例如搜索，命令如下：

```
$ dnf info httpd
Available Packages
Name       : httpd
Version    : 2.4.34
Release    : 8.h5.oe1
Arch       : aarch64
Size       : 1.2 M
Repo       : Local
Summary    : Apache HTTP Server
URL        : http://httpd.apache.org/
License    : ASL 2.0
Description: The Apache HTTP Server is a powerful, efficient, and extensible
            : web server.
```

安装 RPM 包

要安装一个软件包及其所有未安装的依赖，请在 root 权限下执行如下命令：

```
dnf install package_name
```

您也可以通过添加软件包名字同时安装多个软件包。配置文件/etc/dnf/dnf.conf 添加参数 `strict=False`，运行 `dnf` 命令参数添加 `--setopt=strict=0`。请在 root 权限下执行如下命令：

```
dnf install package_name package_name... --setopt=strict=0
```

示例如下：

```
# dnf install httpd
```

下载软件包

使用 `dnf` 下载软件包，请在 root 权限下输入如下命令：

```
dnf download package_name
```

如果需要同时下载未安装的依赖，则加上--resolve，使用命令如下：

```
dnf download --resolve package_name
```

示例如下：

```
# dnf download --resolve httpd
```

删除软件包

要卸载软件包以及相关的依赖软件包，请在 root 权限下执行如下命令：

```
dnf remove package_name...
```

示例如下：

```
# dnf remove totem
```

4.3 管理软件包组

软件包集合是服务于一个共同的目的的一组软件包，例如系统工具集等。使用 dnf 可以对软件包组进行安装/删除等操作，使相关操作更高效。

列出软件包组清单

使用 summary 参数，可以列出系统中所有已安装软件包组、可用的组，可用的环境组的数量，命令如下：

```
dnf groups summary
```

使用示例如下：

```
# dnf groups summary
Last metadata expiration check: 0:11:56 ago on Sat 17 Aug 2019 07:45:14 PM CST.
Available Groups: 8
```

要列出所有软件包组和它们的组 ID，命令如下：

```
dnf group list
```

使用示例如下：

```
# dnf group list
Last metadata expiration check: 0:10:32 ago on Sat 17 Aug 2019 07:45:14 PM CST.
Available Environment Groups:
  Minimal Install
  Custom Operating System
  Server
Available Groups:
  Development Tools
  Graphical Administration Tools
  Headless Management
  Legacy UNIX Compatibility
  Network Servers
```

```
Scientific Support
Security Tools
System Tools
```

显示软件包组信息

要列出包含在一个软件包组中必须安装的包和可选包，使用命令如下：

```
dnf group info glob_expression...
```

例如显示 **Development Tools** 信息，示例如下：

```
# dnf group info "Development Tools"
Last metadata expiration check: 0:14:54 ago on Wed 05 Jun 2019 08:38:02 PM CST.

Group: Development Tools
Description: A basic development environment.
Mandatory Packages:
  binutils
  glibc-devel
  make
  pkgconf
  pkgconf-m4
  pkgconf-pkg-config
  rpm-sign
Optional Packages:
  expect
```

安装软件包组

每一个软件包组都有自己的名称以及相应的 ID (**groupid**)，您可以使用软件包组名称或它的 ID 进行安装。

要安装一个软件包组，请在 **root** 权限下执行如下命令：

```
dnf group install group_name
dnf group install groupid
```

例如安装 **Development Tools** 相应的软件包组，命令如下：

```
# dnf group install "Development Tools"
# dnf group install development
```

删除软件包组

要卸载软件包组，您可以使用软件包组名称或它的 ID，在 **root** 权限下执行如下命令：

```
dnf group remove group_name
dnf group remove groupid
```

例如删除 **Development Tools** 相应的软件包组，命令如下：

```
# dnf group remove "Development Tools"
# dnf group remove development
```

4.4 检查并更新

`dnf` 可以检查您的系统中是否有软件包需要更新。您可以通过 `dnf` 列出需要更新的软件包，并可以选择一次性全部更新或者只对指定包进行更新。

检查更新

如果您需要显示当前系统可用的更新，使用命令如下：

```
dnf check-update
```

使用实例如下：

```
# dnf check-update
Last metadata expiration check: 0:02:10 ago on Sun 01 Sep 2019 11:28:07 PM CST.

anaconda-core.aarch64      19.31.123-1.14      updates
anaconda-gui.aarch64       19.31.123-1.14      updates
anaconda-tui.aarch64       19.31.123-1.14      updates
anaconda-user-help.aarch64 19.31.123-1.14      updates
anaconda-widgets.aarch64   19.31.123-1.14      updates
bind-libs.aarch64          32:9.9.4-29.3       updates
bind-libs-lite.aarch64     32:9.9.4-29.3       updates
bind-license.noarch        32:9.9.4-29.3       updates
bind-utils.aarch64         32:9.9.4-29.3       updates
...
```

升级

如果您需要升级单个软件包，在 `root` 权限下执行如下命令：

```
dnf update package_name
```

例如升级 `rpm` 包，示例如下：

```
# dnf update anaconda-gui.aarch64
Last metadata expiration check: 0:02:10 ago on Sun 01 Sep 2019 11:30:27 PM CST.
Dependencies Resolved

=====
Package                Arch      Version      Repository    Size
=====
Updating:
anaconda-gui           aarch64   19.31.123-1.14  updates      461 k
anaconda-core          aarch64   19.31.123-1.14  updates      1.4 M
anaconda-tui           aarch64   19.31.123-1.14  updates      274 k
anaconda-user-help     aarch64   19.31.123-1.14  updates      315 k
anaconda-widgets       aarch64   19.31.123-1.14  updates      748 k

Transaction Summary
=====
Upgrade 5 Package

Total download size: 3.1 M
Is this ok [y/N]:
```

类似的，如果您需要升级软件包组，在 `root` 权限下执行如下命令：

```
dnf group update group_name
```

更新所有的包和它们的依赖

要更新所有的包和它们的依赖，在 `root` 权限下执行如下命令：

```
dnf update
```

5 管理服务

本章介绍如何使用 `systemd` 进行系统和 service 管理。

- 5.1 简介
- 5.2 特性说明
- 5.3 管理系统服务
- 5.4 改变运行级别
- 5.5 关闭、暂停和休眠系统

5.1 简介

`systemd` 是在 Linux 下，与 SysV 和 LSB 初始化脚本兼容的系统和 service 管理器。`systemd` 使用 socket 和 D-Bus 来开启 service，提供基于守护进程的按需启动策略，支持快照和系统状态恢复，维护挂载和自挂载点，实现了各 service 间基于从属关系的一个更为精细的逻辑控制，拥有更高的并行性能。

概念介绍

`systemd` 开启和监督整个系统是基于 `unit` 的概念。`unit` 是由一个与配置文件对应的名字和类型组成的（例如：`avahi.service unit` 有一个具有相同名字的配置文件，是守护进程 Avahi 的一个封装单元）。`unit` 有多重类型，如表 5-1 所示。

表5-1 unit 说明

unit 名称	后缀名	描述
Service unit	.service	系统服务。
Target unit	.target	一组 <code>systemd units</code> 。
Automount unit	.automount	文件系统挂载点。
Device unit	.device	内核识别的设备文件。

unit 名称	后缀名	描述
Mount unit	.mount	文件系统挂载点。
Path unit	.path	在一个文件系统中的文件或目录。
Scope unit	.scope	外部创建的进程。
Slice unit	.slice	一组用于管理系统进程分层组织的 units。
Snapshot unit	.snapshot	systemd manager 的保存状态。
Socket unit	.socket	一个进程间通信的 Socket。
Swap unit	.swap	swap 设备或者 swap 文件。
Timer unit	.timer	systemd 计时器。

所有的可用 systemd unit 类型，可在如表 5-2 所示的路径下查看。

表5-2 可用 systemd unit 类型

路径	描述
/usr/lib/systemd/system/	随安装的 RPM 产生的 systemd units。
/run/systemd/system/	在运行时创建 systemd units。
/etc/systemd/system/	由系统管理员创建和管理的 systemd units。

5.2 特性说明

更快的启动速度

systemd 提供了比 UpStart 更激进的并行启动能力，采用了 socket/D-Bus activation 等技术启动服务，带来了更快的启动速度。

为了减少系统启动时间，systemd 的目标是：

- 尽可能启动更少的进程。
- 尽可能将更多进程并行启动。

提供按需启动能力

当 sysvinit 系统初始化的时候，它会将所有可能用到的后台服务进程全部启动运行。并且系统必须等待所有的服务都启动就绪之后，才允许用户登录。这种做法有两个缺点：首先是启动时间过长；其次是系统资源浪费。

某些服务很可能在很长一段时间内，甚至整个服务器运行期间都没有被使用过。比如 CUPS，打印服务在多数服务器上很少被真正使用到。您可能没有想到，在很多服务器上 SSHD 也是很少被真正访问到的。花费在启动这些服务上的时间是不必要的；同样，花费在这些服务上的系统资源也是一种浪费。

systemd 可以提供按需启动的能力，只有在某个服务被真正请求的时候才启动它。当该服务结束，systemd 可以关闭它，等待下次需要时再次启动它。

采用 cgroup 特性跟踪和管理进程的生命周期

init 系统的一个重要职责就是负责跟踪和管理服务进程的生命周期。它不仅可以启动一个服务，也能够停止服务。这看上去没有什么特别的，然而在真正用代码实现的时候，您或许会发现停止服务比一开始想的要困难。

服务进程一般都会作为守护进程（daemon）在后台运行，为此服务程序有时候会派生（fork）两次。在 UpStart 中，需要在配置文件中正确地配置 expect 小节。这样 UpStart 通过对 fork 系统调用进行计数，从而获知真正的精灵进程的 PID 号。

cgroup 已经出现了很久，它主要用来实现系统资源配额管理。cgroup 提供了类似文件系统的接口，使用方便。当进程创建子进程时，子进程会继承父进程的 cgroup。因此无论服务如何启动新的子进程，所有的这些相关进程都会属于同一个 cgroup，systemd 只需要简单地遍历指定的 cgroup 即可正确地找到所有的相关进程，将它们逐一停止即可。

启动挂载点和自动挂载的管理

传统的 Linux 系统中，用户可以用/etc/fstab 文件来维护固定的文件系统挂载点。这些挂载点在系统启动过程中被自动挂载，一旦启动过程结束，这些挂载点就会确保存在。这些挂载点都是对系统运行至关重要的文件系统，比如 HOME 目录。和 sysvinit 一样，systemd 管理这些挂载点，以便能够在系统启动时自动挂载它们。systemd 还兼容/etc/fstab 文件，您可以继续使用该文件管理挂载点。

有时候用户还需要动态挂载点，比如打算访问 DVD 内容时，才临时执行挂载以便访问其中的内容，而不访问光盘时该挂载点被取消（umount），以便节约资源。传统地，人们依赖 autofs 服务来实现这种功能。

systemd 内建了自动挂载服务，无需另外安装 autofs 服务，可以直接使用 systemd 提供的自动挂载管理能力来实现 autofs 的功能。

实现事务性依赖关系管理

系统启动过程是由很多的独立工作共同组成的，这些工作之间可能存在依赖关系，比如挂载一个 NFS 文件系统必须依赖网络能够正常工作。systemd 虽然能够最大限度地并发执行很多有依赖关系的工作，但是类似“挂载 NFS”和“启动网络”这样的工作还是存在天生的先后依赖关系，无法并发执行。对于这些任务，systemd 维护一个“事务一致性”的概念，保证所有相关的服务都可以正常启动而不会出现互相依赖，以至于死锁的情况。

与 SysV 初始化脚本兼容

和 UpStart 一样，systemd 引入了新的配置方式，对应用程序的开发也有一些新的要求。如果 systemd 想替代目前正在运行的初始化系统，就必须和现有程序兼容。任何一个 Linux 发行版都很难为了采用 systemd 而在短时间内将所有的服务代码都修改一遍。

systemd 提供了和 sysvinit 以及 LSB initscripts 兼容的特性。系统中已经存在的服务和进程无需修改。这降低了系统向 systemd 迁移的成本，使得 systemd 替换现有初始化系统成为可能。

能够对系统进行快照和恢复

systemd 支持按需启动，因此系统的运行状态是动态变化的，人们无法准确地知道系统当前运行了哪些服务。systemd 快照提供了一种将当前系统运行状态保存并恢复的能力。

比如系统当前正运行服务 A 和 B，可以用 systemd 命令行对当前系统运行状况创建快照。然后将进程 A 停止，或者做其他的任意的对系统的改变，比如启动新的进程 C。在这些改变之后，运行 systemd 的快照恢复命令，就可立即将系统恢复到快照时刻的状态，即只有服务 A，B 在运行。一个可能的应用场景是调试：比如服务器出现一些异常，为了调试用户将当前状态保存为快照，然后可以进行任意的操作，比如停止服务等。等调试结束，恢复快照即可。

5.3 管理系统服务

systemd 提供 systemctl 命令来运行、关闭、重启、显示、启用/禁用系统服务。

sysvinit 命令和 systemd 命令

systemd 提供 systemctl 命令与 sysvinit 命令的功能类似。当前版本中依然兼容 service 和 chkconfig 命令，相关说明如表 5-3，但建议用 systemctl 进行系统服务管理。

表5-3 sysvinit 命令和 systemd 命令的对照表

sysvinit 命令	systemd 命令	备注
service <i>network</i> start	systemctl start <i>network.service</i>	用来启动一个服务 (并不会重启现有的)。
service <i>network</i> stop	systemctl stop <i>network.service</i>	用来停止一个服务 (并不会重启现有的)。
service <i>network</i> restart	systemctl restart <i>network.service</i>	用来停止并启动一个服务。
service <i>network</i> reload	systemctl reload <i>network.service</i>	当支持时，重新装载配置文件而不中断等待操作。
service <i>network</i> condrestart	systemctl condrestart <i>network.service</i>	如果服务正在运行那么重启它。

sysvinit 命令	systemd 命令	备注
<code>service network status</code>	<code>systemctl status network.service</code>	检查服务的运行状态。
<code>chkconfig network on</code>	<code>systemctl enable network.service</code>	在下次启动时或满足其他触发条件时设置服务为启用。
<code>chkconfig network off</code>	<code>systemctl disable network.service</code>	在下次启动时或满足其他触发条件时设置服务为禁用。
<code>chkconfig network</code>	<code>systemctl is-enabled network.service</code>	用来检查一个服务在当前环境下被配置为启用还是禁用。
<code>chkconfig --list</code>	<code>systemctl list-unit-files --type=service</code>	输出在各个运行级别下服务的启用和禁用情况。
<code>chkconfig network --list</code>	<code>ls /etc/systemd/system/*.wants/network.service</code>	用来列出该服务在哪些运行级别下启用和禁用。
<code>chkconfig network --add</code>	<code>systemctl daemon-reload</code>	当您创建新服务文件或者变更设置时使用。

显示所有当前服务

如果您需要显示当前正在运行的服务，使用命令如下：

```
systemctl list-units --type service
```

如果您需要显示所有的服务（包括未运行的服务），需要添加 `--all` 参数，使用命令如下：

```
systemctl list-units --type service --all
```

例如显示当前正在运行的服务，命令如下：

```
$ systemctl list-units --type service
UNIT                                LOAD    ACTIVE SUB    JOB    DESCRIPTION
atd.service                         loaded active running Deferred execution scheduler
auditd.service                      loaded active running Security Auditing Service
avahi-daemon.service               loaded active running Avahi mDNS/DNS-SD Stack
chronyd.service                    loaded active running NTP client/server
crond.service                      loaded active running Command Scheduler
dbus.service                       loaded active running D-Bus System Message Bus
dracut-shutdown.service            loaded active exited Restore /run/initramfs on shutdown
firewalld.service                  loaded active running firewalld - dynamic
firewall daemon
getty@tty1.service                 loaded active running Getty on tty1
gssproxy.service                   loaded active running GSSAPI Proxy Daemon
```

```
irqbalance.service    loaded active    running    irqbalance daemon
iscsid.service        loaded activating start    start Open-iSCSI
```

显示服务状态

如果您需要显示某个服务的状态，可执行如下命令：

```
systemctl status name.service
```

相关状态显示参数说明如表 5-4 所示。

表5-4 状态参数说明

参数	描述
Loaded	说明服务是否被加载，并显示服务对应的绝对路径以及是否启用。
Active	说明服务是否正在运行，并显示时间节点。
Main PID	相应的系统服务的 PID 值。
CGroup	相关控制组（CGroup）的其他信息。

如果您需要鉴别某个服务是否运行，可执行如下命令：

```
systemctl is-active name.service
```

is-active 命令的返回结果如下：

表5-5 is-active 命令的返回结果

状态	含义
active(running)	有一只或多只程序正在系统中执行
active(exited)	仅执行一次就正常结束的服务，目前并没有任何程序在系统中执行。举例来说，开机或者是挂载时才会进行一次的 quotaon 功能
active(waiting)	正在执行当中，不过要等待其他的事件才能继续处理。例如：打印的队列相关服务 就是这种状态，虽然正在启动中，不过也需要真的有队列进来（打印作业）这样他才会继续唤醒打印机 服务来进行下一步打印的功能
inactive	这个服务没有运行

同样，如果您需要判断某个服务是否被启用，可执行如下命令：

```
systemctl is-enabled name.service
```

is-enabled 命令的返回结果如下：

表5-6 is-enabled 命令的返回结果

状态	含义
"enabled"	已经通过 /etc/systemd/system/ 目录下的 Alias= 别名、.wants/ 或 .requires/ 软连接被永久启用。
"enabled-runtime"	已经通过 /run/systemd/system/ 目录下的 Alias= 别名、.wants/ 或 .requires/ 软连接被临时启用。
"linked"	虽然单元文件本身不在标准单元目录中，但是指向此单元文件的一个或多个软连接已经存在于 /etc/systemd/system/ 永久目录中。
"linked-runtime"	虽然单元文件本身不在标准单元目录中，但是指向此单元文件的一个或多个软连接已经存在于 /run/systemd/system/ 临时目录中。
"masked"	已经被 /etc/systemd/system/ 目录永久屏蔽(软连接指向 /dev/null 文件)，因此 start 操作会失败。
"masked-runtime"	已经被 /run/systemd/systemd/ 目录临时屏蔽(软连接指向 /dev/null 文件)，因此 start 操作会失败。
"static"	尚未被启用，并且单元文件的 "[Install]" 小节中没有可用于 enable 命令的选项。
"indirect"	尚未被启用，但是单元文件的 "[Install]" 小节中 Also= 选项的值列表非空(也就是列表中的某些单元可能已被启用)、或者它拥有一个不在 Also= 列表中的其他名称的别名软连接。对于模版单元来说，表示已经启用了不同于 DefaultInstance= 的实例。
"disabled"	尚未被启用，但是单元文件的 "[Install]" 小节中存在可用于 enable 命令的选项
"generated"	单元文件是被单元生成器动态生成的。被生成的单元文件可能并未被直接启用，而是被单元生成器隐含的启用了。
"transient"	单元文件是被运行时 API 动态临时生成的。该临时单元可能并未被启用。
"bad"	单元文件不正确或者出现其他错误。 is-enabled 不会返回此状态，而是会显示一条出错信息。 list-unit-files 命令有可能会显示此单元。

例如查看 gdm.service 服务状态，命令如下：

```
# systemctl status gdm.service
gdm.service - GNOME Display Manager   Loaded: loaded
```

```
(/usr/lib/systemd/system/gdm.service; enabled) Active: active (running) since Thu
2013-10-17 17:31:23 CEST; 5min ago
Main PID: 1029 (gdm)
  CGroup: /system.slice/gdm.service
          └─1029 /usr/sbin/gdm
             └─1037 /usr/libexec/gdm-simple-slave --display-id /org/gno...
└─1047 /usr/bin/Xorg :0 -background none -verbose -auth /r...Oct 17 17:31:23
localhost systemd[1]: Started GNOME Display Manager.
```

运行服务

如果您需要运行某个服务，请在 **root** 权限下执行如下命令：

```
systemctl start name.service
```

例如运行 **httpd** 服务，命令如下：

```
# systemctl start httpd.service
```

关闭服务

如果您需要关闭某个服务，请在 **root** 权限下执行如下命令：

```
systemctl stop name.service
```

例如关闭蓝牙服务，命令如下：

```
# systemctl stop bluetooth.service
```

重启服务

如果您需要重启某个服务，请在 **root** 权限下执行如下命令：

```
systemctl restart name.service
```

执行命令后，当前服务会被关闭，但马上重新启动。如果您指定的服务，当前处于关闭状态，执行命令后，服务也会被启动。

例如重启蓝牙服务，命令如下：

```
# systemctl restart bluetooth.service
```

启用服务

如果您需要在开机时启用某个服务，请在 **root** 权限下执行如下命令：

```
systemctl enable name.service
```

例如设置 **httpd** 服务开机时启动，命令如下：

```
# systemctl enable httpd.service
ln -s '/usr/lib/systemd/system/httpd.service' '/etc/systemd/system/multi-
user.target.wants/httpd.service'
```

禁用服务

如果您需要在开机时禁用某个服务，请在 **root** 权限下执行如下命令：

```
systemctl disable name.service
```

例如在开机时禁用蓝牙服务启动，命令如下：

```
# systemctl disable bluetooth.service
Removed /etc/systemd/system/bluetooth.target.wants/bluetooth.service.
Removed /etc/systemd/system/dbus-org.bluez.service.
```

5.4 改变运行级别

Target 和运行级别

systemd 用目标（target）替代了运行级别的概念，提供了更大的灵活性，如您可以继承一个已有的目标，并添加其他服务，来创建自己的目标。表 5-7 列举了 systemd 下的目标和常见 runlevel 的对应关系。

表5-7 运行级别和 systemd 目标

运行级别	systemd 目标 (target)	描述
0	runlevel0.target, poweroff.target	关闭系统。
1, s, single	runlevel1.target, rescue.target	单用户模式。
2, 4	runlevel2.target, runlevel4.target, multi- user.target	用户定义/域特定运行级别。默认等同于 3。
3	runlevel3.target, multi- user.target	多用户，非图形化。用户可以通过多个控制台或网络登录。
5	runlevel5.target, graphical.target	多用户，图形化。通常为所有运行级别 3 的服务外加图形化登录。
6	runlevel6.target, reboot.target	重启系统。
emergenc y	emergency.target	紧急 Shell

查看系统默认启动目标

查看当前系统默认的启动目标，命令如下：

```
systemctl get-default
```

查看当前系统所有的启动目标

查看当前系统所有的启动目标，命令如下：

```
systemctl list-units --type=target
```

改变默认目标

改变系统默认的目标，在 root 权限下执行如下命令：

```
systemctl set-default name.target
```

改变当前目标

改变当前系统的目标，在 root 权限下执行如下命令：

```
systemctl isolate name.target
```

切换到救援模式

改变当前系统为救援模式，在 root 权限下执行如下命令：

```
systemctl rescue
```

这条命令和“systemctl isolate rescue.target”类似。命令执行后会在串口有如下打印信息：

```
You are in rescue mode. After logging in, type "journalctl -xb" to view system logs,
"systemctl reboot" to reboot, "systemctl default" or "exit" to boot into default
mode.
```

```
Give root password for maintenance
(or press Control-D to continue):
```

说明

用户需要重启系统，从救援模式进入正常模式。

切换到紧急模式

改变当前系统为紧急模式，在 root 权限下执行如下命令：

```
systemctl emergency
```

这条命令和“systemctl isolate emergency.target”类似。命令执行后会在串口有如下打印信息：

```
You are in emergency mode. After logging in, type "journalctl -xb" to view system
logs, "systemctl reboot" to reboot, "systemctl default" or "exit" to boot into
default mode.
```

```
Give root password for maintenance
(or press Control-D to continue):
```

说明

用户需要重启系统，从紧急模式进入正常模式。

5.5 关闭、暂停和休眠系统

systemctl 命令

systemd 通过 systemctl 命令可以对系统进行关机、重启、休眠等一系列操作。当前仍兼容部分 Linux 常用管理命令，对应关系如表 5-8。建议用户使用 systemctl 命令进行操作。

表5-8 命令对应关系

Linux 常用管理命令	systemctl 命令	描述
halt	systemctl halt	关闭系统
poweroff	systemctl poweroff	关闭电源
reboot	systemctl reboot	重启

关闭系统

关闭系统并下电，在 root 权限下执行如下命令：

```
systemctl poweroff
```

关闭系统但不下电机器，在 root 权限下执行如下命令：

```
systemctl halt
```

执行上述命令会给当前所有的登录用户发送一条提示消息。如果不想让 systemd 发送该消息，您可以添加 “--no-wall” 参数。具体命令如下：

```
systemctl --no-wall poweroff
```

重启系统

重启系统，在 root 权限下执行如下命令：

```
systemctl reboot
```

执行上述命令会给当前所有的登录用户发送一条提示消息。如果不想让 systemd 发送该消息，您可以添加 “--no-wall” 参数。具体命令如下：

```
systemctl --no-wall reboot
```

使系统待机

使系统待机，在 root 权限下执行如下命令：

```
systemctl suspend
```


使系统休眠

使系统休眠，在 root 权限下执行如下命令：

```
systemctl hibernate
```

使系统待机且处于休眠状态，在 root 权限下执行如下命令：

```
systemctl hybrid-sleep
```

6 管理进程

本章介绍了 Linux 内核的进程管理方式，然后以实例的方式讲解了 Linux 提供的常用的进程控制命令、at 和 cron 服务，以及进程查看命令。

6.1 管理系统进程

6.2 查看进程

6.1 管理系统进程

操作系统管理多个用户的请求和多个任务。大多数系统都只有一个 CPU 和一个主要存储，但一个系统可能有多个二级存储磁盘和多个输入/输出设备。操作系统管理这些资源并在多个用户间共享资源，当用户提出一个请求时，造成好像系统被用户独占的假象。实际上操作系统监控着一个等待执行的任务队列，这些任务包括用户任务、操作系统任务、邮件和打印任务等。本节将从用户的角度讲述如何控制进程。

6.1.1 调度启动进程

有时候需要对系统进行一些比较费时而且占用资源的维护工作，这些工作适合在深夜进行，这时候用户就可以事先进行调度安排，指定任务运行的时间或者场合，到时候系统会自动完成这些任务。要使用自动启动进程的功能，就需要掌握以下几个启动命令。

6.1.1.1 定时运行一批程序 (at)

at 命令

用户使用 at 命令在指定时刻执行指定的命令序列。该命令至少需要指定一个命令和一个执行时间。at 命令可以只指定时间，也可以时间和日期一起指定。

at 命令的语法格式如下：

```
at [-V] [-q 队列] [-f 文件名] [-mldbv] 时间
at -c 作业 [作业...]
```

设置时间

`at` 允许使用一套相当复杂的时间指定方法，比如：

- 接受在当天的 `hh:mm`（小时：分钟）式的时间指定。如果该时间已经过去，那么就放存第二天执行。
- 使用 `midnight`（深夜）、`noon`（中午）、`teatime`（饮茶时间，一般是下午 4 点）等比较模糊的词语来指定时间。
- 采用 12 小时计时制，即在时间后面加上 `AM`（上午）或者 `PM`（下午）来说明是上午还是下午。
- 指定命令执行的具体日期，指定格式为 `month day`（月日）或者 `mm/dd/yy`（月/日/年）或者 `dd.mm.yy`（日.月.年）。指定的日期必须跟在指定时间的后面。

上面介绍的都是绝对计时法，其实还可以使用相对计时法，这对于安排不久就要执行的命令是很有好处的。指定格式为 `now+count time-units`，`now` 就是当前时间，`time-units` 是时间单位，这里可以是 `minutes`（分钟）、`hours`（小时）、`days`（天）、`weeks`（星期）。`count` 是时间的数量，究竟是几天，还是几小时等。还有一种计时方法就是直接使用 `today`（今天）、`tomorrow`（明天）来指定完成命令的时间。下面通过一些例子来说明具体用法。

例如指定在今天下午 4:30 执行某个命令。假设现在时间是中午 12:30，2019 年 6 月 7 日，可用命令格式如下：

```
at 4:30pm
at 16:30
at 16:30 today
at now+4 hours
at now+ 240 minutes
at 16:30 7.6.19
at 16:30 6/7/19
at 16:30 Jun 7
```

以上这些命令表达的意义是完全一样的，所以在安排时间的时候完全可以根据个人喜好和具体情况自由选择。一般采用绝对时间的 24 小时计时法可以避免由于用户自己的疏忽造成计时错误，例如上例可以写成：`at 16:30 6/7/19`。

执行权限

对于 `at` 命令来说，需要定时执行的命令是从标准输入或者使用 `-f` 选项指定的文件中读取并执行的。如果 `at` 命令是从一个使用 `su` 命令切换到用户 `shell` 中执行的，那么当前用户被认为是执行用户，所有的错误和输出结果都会送给这个用户。但是如果邮件送出的话，收到邮件的将是原来的用户，也就是登录时 `shell` 的所有者。

例如在 6 月 8 日上午 10 点执行 `slocate -u` 命令。命令如下：

```
# at 10:00 6/8/19
at> slocate -u
at>
[1]+  Stopped      at 10:00 6/8/19
```

上面的结果中，输入 `at` 命令之后，会出现提示符 `at>`，提示用户输入命令，在此输入了 `slocate -u`，然后按回车键。还可以输入多条命令，当所有要执行的命令输入结束后，按 `Ctrl+d` 键结束 `at` 命令。

在任何情况下，管理员账户都可以使用这个命令。对于其他用户来说，是否可以就取决于/etc/at.allow 和/etc/at.deny 文件。

6.1.1.2 周期性运行一批程序（cron）

前面介绍 at 命令都会在一定时间内完成一定任务，但是它只能执行一次。也就是说，当指定了运行命令后，系统在指定时间完成任务，以后就不再执行了。但是在很多情况下需要周期性重复执行一些命令，这时候就需要使用 cron 命令来完成任务。

运行机制

首先 cron 命令会搜索/var/spool/cron 目录，寻找以/etc/passwd 文件中的用户名命名的 crontab 文件，被找到的这种文件将装入内存。比如一个用户名为 globus 的用户，对应的 crontab 文件应该是/var/spool/cron/globus，即以该用户命名的 crontab 文件存放在 /var/spool/cron 目录下面。

cron 命令还将搜索/etc/crontab 文件，这个文件是用不同的格式写成的。cron 启动以后，它将首先检查是否有用户设置了 crontab 文件，如果没有就转入睡眠状态，释放系统资源。所以该后台进程占用资源极少，它每分钟被唤醒一次，查看当前是否有需要运行的命令。

命令执行结束后，任何输出都将作为邮件发送给 crontab 的所有者，或者是/etc/crontab 文件中 MAILTO 环境变量中指定的用户。这是 cron 的工作原理，但是 cron 命令的执行不需要用户干涉，用户只需要修改 crontab 中要执行的命令。

crontab 命令

crontab 命令用于安装、删除或者显示用于驱动 cron 后台进程的表格。用户把需要执行的命令序列放到 crontab 文件中以获得执行，而且每个用户都可以有自己的 crontab 文件。

crontab 命令的常用方法如下：

- crontab -u //设置某个用户的 cron 服务，root 用户在执行 crontab 时需要此参数。
- crontab -l //列出某个用户 cron 服务的详细内容。
- crontab -r //删除某个用户的 cron 服务。
- crontab -e //编辑某个用户的 cron 服务。

例如 root 查看自己的 cron 设置。命令如下：

```
crontab -u root -l
```

crontab 文件

在 crontab 文件中输入需要执行的命令和时间。该文件中每行都包括 6 个域，其中前 5 个域是指定命令被执行的时间，最后一个域是要被执行的命令。每个域之间使用空格或者制表符分隔。格式如下：

```
minute hour day-of-month month-of-year day-of-week commands
```

对于每一项的说明如所示。

表6-1 参数说明

参数	描述
minute	分钟（0~59）。
hour	小时（0~23）。
day-of-month	一个月的第几天（1~31）。
month-of-year	一年的第几个月（1~12）。
day-of-week	一周的星期几（0~6），0代表星期天。
commands	需要执行的命令。

这些项都不能为空，必须指定值。除了数字还有几个特殊的符号“*”、“/”和“-”、“,”。其中，*代表所有的取值范围内的数字，/代表每的意思，“*/5”表示每5个单位，“-”代表从某个数字到某个数字，“,”分开几个离散时数字。对于要执行的命令，调用的时候需要写出命令的完整路径。

例如晚上 18 点到 22 点之间每两个小时，在/tmp/test.txt 文件中加入 sleepy 文本。在 crontab 文件中对应的行如下：

```
* 18-22/2 * * * echo "sleepy" >> /tmp/test.txt
```

每次编辑完某个用户的 cron 设置后，cron 自动在/var/spool/cron 下生成一个与此用户同名的文件。此用户的 cron 信息都记录在这个文件中，这个文件是不可以直接编辑的，只可以用 crontab -e 来编辑。用户也可以另外建立一个文件，使用“cron 文件名”命令导入 cron 设置。

假设有个用户名为 globus，它需要为自己创建一个 crontab 文件。步骤如下：

1. 首先可以使用任何文本编辑器建立一个新文件，并将向该文件加入需要运行的命令和要定期执行的时间，假设该文件为 ~/globus.cron。
2. 然后使用 crontab 命令安装这个文件，使用 crontab 命令使之成为该用户的 crontab 文件。命令如下：

```
crontab globus. ~/globus.cron
```

这样 crontab 文件就建立好了，可以转到/var/spool/cron 目录下面查看，发现多了一个 globus 文件。这个文件就是所需的 crontab 文件。

📖 说明

cron 启动后，每过一分钟读一次 crontab 文件，检查是否要执行里面的命令。因此该文件被修改后不需要重新启动 cron 服务。

编辑配置文件

cron 服务每分钟不仅要读一次/var/spool/cron 内的所有文件，还需要读一次 /etc/crontab，因此通过配置这个文件也能得到 cron 的服务。用 crontab 配置是针对某个用户的，而编辑/etc/crontab 是针对系统的任务。此文件的文件格式如下：

```
SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root //如果出现错误, 或者有数据输出, 数据作为邮件发给这个账号
HOME=/
# run-parts
01 * * * * root run-parts /etc/cron.hourly //每小时执行一次/etc/cron.hourly 里的脚本
02 4 * * * root run-parts /etc/cron.daily //每天执行一次/etc/cron.daily 里的脚本
22 4 * * 0 root run-parts /etc/cron.weekly //每周执行一次/etc/cron.weekly 里的脚本
42 4 1 * * root run-parts /etc/cron.monthly //每月执行一次/etc/cron.monthly 里的脚本
```

📖 说明

如果去掉 `run-parts` 参数, 其后面就是运行的某个脚本名, 而不是目录名。

6.1.2 挂起/恢复进程

作业控制允许进程挂起并可以在需要时恢复进程的运行, 被挂起的作业恢复后将中止处开始继续运行。只要在键盘上按 `Ctrl+Z` 键, 即可挂起当前的前台作业。在键盘上按 `Ctrl+Z` 键后, 将挂起当前执行的命令 `cat`。使用 `jobs` 命令可以显示 `shell` 的作业清单, 包括具体的作业、作业号以及作业当前所处的状态。

恢复进程执行时, 有两种选择: 用 `fg` 命令将挂起的作业放回到前台执行; 用 `bg` 命令将挂起的作业放到后台执行。灵活使用上述命令, 将给自己带来很大的方便。

6.2 查看进程

Linux 是一个多任务系统, 经常需要对这些进程进行一些调配和管理。要进行管理, 首先就要知道现在的进程情况: 有哪些进程、进程的状态如何等。Linux 提供了多种命令来了解进程的状况。

who 命令

`who` 命令主要用于查看当前系统中的用户情况。如果用户想和其他用户建立即时通讯, 比如使用 `talk` 命令, 那么首先要确定的就是该用户确实在线上, 不然 `talk` 进程就无法建立起来。又如, 系统管理员希望监视每个登录的用户此时此刻的所作所为, 也要使用 `who` 命令。`who` 命令应用起来非常简单, 可以比较准确地掌握用户的情况, 所以使用非常广泛。

例如查看系统中的用户及其状态。使用如下:

```
# who
admin    tty1      Jul 28 15:55
admin    pts/0    Aug  5 15:46 (192.168.0.110)
admin    pts/2    Jul 29 19:52 (192.168.0.110)
root     pts/3    Jul 30 12:07 (192.168.0.110)
root     pts/4    Jul 31 10:29 (192.168.0.144)
root     pts/5    Jul 31 14:52 (192.168.0.11)
root     pts/6    Aug  6 10:12 (192.168.0.234)
root     pts/8    Aug  6 11:34 (192.168.0.234)
```

ps 命令

ps 命令是最基本又非常强大的进程查看命令。使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵尸、哪些进程占用了过多的资源等，大部分进程信息都是可以通过执行该命令得到的。

ps 命令最常用的还是用来监控后台进程的工作情况，因为后台进程是不与屏幕、键盘这些标准输入/输出设备进行通信的，所以如果需要检测其状况，就可使用 ps 命令。ps 命令的常见选项如表 6-2 所示。

表6-2 选项说明

选项	描述
-e	显示所有进程。
-f	全格式。
-h	不显示标题。
-l	使用长格式。
-w	宽行输出。
-a	显示终端上的所有进程，包括其他用户的进程。
-r	只显示正在运行的进程。
-x	显示没有控制终端的进程。

例如显示系统中终端上的所有进行进程。命令如下：

```
# ps -a
  PID TTY          TIME CMD
 12175 pts/6        00:00:00 bash
 24526 pts/0        00:00:00 vsftpd
 29478 pts/5        00:00:00 ps
 32461 pts/0        1-01:58:33 sh
```

top 命令

top 命令和 ps 命令的基本作用是相同的，显示系统当前的进程和其他状况，但是 top 是一个动态显示过程，即可以通过用户按键来不断刷新进程的当前状态，如果在前台执行该命令，它将独占前台，直到用户终止该程序为止。其实 top 命令提供了实时的对系统处理器的状态监视。它将显示系统中 CPU 的任务列表。该命令可以按 CPU 使用、内存使用和执行时间对任务进行排序，而且该命令的很多特性都可以通过交互式命令或者在定制文件中进行设定。

top 命令输出的实例如图 6-1 所示：

图6-1 top 显示

```
top - 19:04:08 up 9 days, 3:09, 8 users, load average: 2.17, 2.08, 2.06
Tasks: 242 total, 8 running, 234 sleeping, 0 stopped, 0 zombie
Cpu(s): 8.3%us, 0.2%sy, 0.0%ni, 91.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 19983M total, 19777M used, 206M free, 567M buffers
Swap: 2053M total, 10M used, 2043M free, 12326M cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
32757	root	20	0	4462m	3.0g	5440	S	100	15.3	1542:40	qemu-kvm
32461	root	20	0	11580	1380	1120	R	100	0.0	1563:47	sh
31437	root	20	0	4626m	2.4g	5436	R	4	12.1	14:36.89	qemu-kvm
29553	root	20	0	17256	1392	932	R	0	0.0	0:00.02	top
31438	root	20	0	0	0	0	S	0	0.0	0:12.80	vhost-31437
32758	root	20	0	0	0	0	S	0	0.0	0:25.21	vhost-32757
1	root	20	0	10540	796	748	S	0	0.0	0:04.59	init
2	root	20	0	0	0	0	S	0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0	0.0	0:01.64	ksoftirqd/0
6	root	RT	0	0	0	0	S	0	0.0	0:01.08	migration/0
7	root	RT	0	0	0	0	S	0	0.0	0:01.66	watchdog/0
8	root	RT	0	0	0	0	S	0	0.0	0:01.09	migration/1
9	root	20	0	0	0	0	S	0	0.0	0:05.58	kworker/1:0
10	root	20	0	0	0	0	S	0	0.0	0:01.31	ksoftirqd/1
11	root	20	0	0	0	0	S	0	0.0	0:50.48	kworker/0:1
12	root	RT	0	0	0	0	S	0	0.0	0:01.27	watchdog/1
13	root	RT	0	0	0	0	S	0	0.0	0:01.64	migration/2
14	root	20	0	0	0	0	S	0	0.0	0:00.00	kworker/2:0
15	root	20	0	0	0	0	S	0	0.0	1:01.89	ksoftirqd/2
16	root	RT	0	0	0	0	S	0	0.0	0:01.38	watchdog/2
17	root	RT	0	0	0	0	R	0	0.0	0:01.12	migration/3
18	root	20	0	0	0	0	S	0	0.0	0:00.00	kworker/3:0
19	root	20	0	0	0	0	S	0	0.0	0:22.84	ksoftirqd/3
20	root	RT	0	0	0	0	S	0	0.0	0:01.33	watchdog/3
21	root	RT	0	0	0	0	S	0	0.0	0:01.56	migration/4
22	root	20	0	0	0	0	S	0	0.0	0:00.00	kworker/4:0
23	root	20	0	0	0	0	S	0	0.0	0:00.01	ksoftirqd/4
24	root	RT	0	0	0	0	S	0	0.0	0:01.29	watchdog/4

kill 命令

当需要中断一个前台进程的时候，通常是使用“Ctrl+c”组合键，而对于后台进程不能用组合键来终止，这时就可以使用 kill 命令。该命令可以终止前台和后台进程。终止后台进程的原因包括：该进程占用 CPU 的时间过多、该进程已经死锁等。

kill 命令是通过向进程发送指定的信号来结束进程的。如果没有指定发送的信号，那么默认值为 TERM 信号。TERM 信号将终止所有不能捕获该信号的进程。至于那些可以捕获该信号的进程可能就需要使用 KILL 信号（它的编号为 9），而该信号不能被捕捉。

kill 命令的语法格式有以下两种方式：

```
kill [-s 信号 | -p] [-a] 进程号...
kill -l [信号]
```

其中进程号可以通过 ps 命令的输出得到。-s 选项是给程序发送指定的信号，详细的信号可以用“kill -l”命令查看；-p 选项只显示指定进程的 ID 号。

杀死 pid 为 1409 的进程，示例如下：


```
# kill -9 1409
```

显示所有的信号及其编号对应关系，示例如下：

```
# kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS     8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD   18) SIGCONT    19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU   23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF   28) SIGWINCH   29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

7 配置网络

- 7.1 配置 IP
- 7.2 配置主机名
- 7.3 配置网络绑定
- 7.4 IPv6 使用差异说明 (vs IPv4)

7.1 配置 IP

7.1.1 使用 nmcli

7.1.1.1 nmcli 介绍

nmcli 是 NetworkManager 的一个命令行工具，它提供了使用命令行配置由 NetworkManager 管理网络连接的方法。nmcli 命令的基本格式为：

```
nmcli [OPTIONS] OBJECT { COMMAND | help }
```

其中，OBJECT 选项可以是 general、networking、radio、connection 或 device 等。在日常使用中，最常使用的是 -t, --terse（用于脚本）、-p, --pretty 选项（用于用户）及 -h, --help 选项，用户可以使用 “ nmcli help ” 获取更多参数及使用信息。

```
# nmcli help
```

常用命令使用举例如下：

- 显示 NetworkManager 状态：

```
nmcli general status
```

- 显示所有连接：

```
nmcli connection show
```

- 只显示当前活动连接，如下所示添加 -a, --active：

```
nmcli connection show --active
```

- 显示由 NetworkManager 识别到设备及其状态：

```
nmcli device status
```

- 使用 `nmcli` 工具启动和停止网络接口，例如：

```
nmcli connection up id enp3s0
nmcli device disconnect enp3s0
```

7.1.1.2 设置网络连接

列出目前可用的网络连接：

```
# nmcli con show

NAME      UUID                                  TYPE      DEVICE
enp4s0    5afce939-400e-42fd-91ee-55ff5b65deab  ethernet  enp4s0
enp3s0    c88d7b69-f529-35ca-81ab-aa729ac542fd  ethernet  enp3s0
virbr0    ba552da6-f014-49e3-91fa-ec9c388864fa  bridge    virbr0
```

📖 说明

输出结果中的 `NAME` 字段代表连接 ID（名称）。

添加一个网络连接会生成相应的配置文件，并与相应的设备关联。检查可用的设备，方法如下：

```
# nmcli dev status

DEVICE      TYPE      STATE      CONNECTION
enp3s0      ethernet  connected  enp3s0
enp4s0      ethernet  connected  enp4s0
virbr0      bridge    connected  virbr0
lo          loopback  unmanaged  --
virbr0-nic  tun       unmanaged  --
```

7.1.1.2.1 配置动态 IP 连接

配置 IP

要使用 DHCP 分配网络时，可以使用动态 IP 配置添加网络配置文件，命令格式如下：

```
nmcli connection add type ethernet con-name connection-name ifname interface-name
```

例如创建名为 `net-test` 的动态连接配置文件，使用以下命令：

```
# nmcli connection add type ethernet con-name net-test ifname enp3s0
Connection 'net-test' (a771baa0-5064-4296-ac40-5dc8973967ab) successfully added.
```

NetworkManager 会将参数 `connection.autoconnect` 设定为 `yes`，并将设置保存到 `“/etc/sysconfig/network-scripts/ifcfg-net-test”` 文件中，在该文件中会将 `ONBOOT` 设置为 `yes`。

激活连接并检查状态

使用以下命令激活网络连接，使用以下命令：

```
# nmcli con up net-test
Connection successfully activated (D-Bus active
path:/org/freedesktop/NetworkManager/ActiveConnection/5)
```

检查这些设备及连接的状态，使用以下命令：

```
# nmcli device status

DEVICE      TYPE      STATE      CONNECTION
enp4s0      ethernet  connected  enp4s0
enp3s0      ethernet  connected  net-test
virbr0      bridge    connected  virbr0
lo          loopback  unmanaged  --
virbr0-nic  tun       unmanaged  --
```

7.1.1.2.2 配置静态 IP 连接

配置 IP

添加静态 IPv4 配置的网络连接，可使用以下命令：

```
nmcli connection add type ethernet con-name connection-name ifname interface-name
ip4 address gw4 address
```

📖 说明

如果要添加 IPv6 地址和网关信息，使用 `ip6` 和 `gw6` 选项。

例如创建名为 `net-static` 的静态连接配置文件，使用以下命令：

```
# nmcli con add type ethernet con-name net-static ifname enp3s0 ip4 192.168.0.10/24
gw4 192.168.0.254
```

还可为该设备同时指定 IPv6 地址和网关，示例如下：

```
# nmcli con add type ethernet con-name test-lab ifname enp3s0 ip4 192.168.0.10/24
gw4 192.168.0.254 ip6 abbe::**** gw6 2001::***:*
Connection 'net-static' (63aa2036-8665-f54d-9a92-c3035bad03f7) successfully added.
```

NetworkManager 会将其内部参数 `ipv4.method` 设定为 `manual`，将 `connection.autoconnect` 设定为 `yes`，并将设置写入 `/etc/sysconfig/network-scripts/ifcfg-my-office` 文件，其中会将对应 `BOOTPROTO` 设定为 `none`，将 `ONBOOT` 设定为 `yes`。

设定两个 IPv4 DNS 服务器地址，使用以下命令：

```
# nmcli con mod net-static ipv4.dns "*. *.* *.*.*"
```

设置两个 IPv6 DNS 服务器地址，使用以下命令：

```
# nmcli con mod net-static ipv6.dns "2001:4860:4860::**** 2001:4860:4860::****"
```

激活连接并检查状态

激活新的网络连接，使用以下命令：

```
# nmcli con up net-static ifname enp3s0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/6)
```

检查这些设备及连接的状态，使用以下命令：

```
# nmcli device status

DEVICE      TYPE      STATE      CONNECTION
enp4s0      ethernet  connected  enp4s0
enp3s0      ethernet  connected  net-static
virbr0      bridge    connected  virbr0
lo          loopback  unmanaged  --
virbr0-nic  tun       unmanaged  --
```

查看配置的连接详情，使用以下命令（使用 `-p, --pretty` 选项在输出结果中添加标题和分段）：

```
# nmcli -p con show net-static
=====
Connection profile details (net-static )
=====
connection.id:                net-static
connection.uuid:              b9f18801-6084-4aee-af28-c8f0598ff5e1
connection.stable-id:         --
connection.type:              802-3-ethernet
connection.interface-name:    enp3s0
connection.autoconnect:       yes
connection.autoconnect-priority: 0
connection.autoconnect-retries: -1 (default)
connection.multi-connect:     0 (default)
connection.auth-retries:      -1
connection.timestamp:         1578988781
connection.read-only:         no
connection.permissions:       --
connection.zone:              --
connection.master:            --
connection.slave-type:        --
connection.autoconnect-slaves: -1 (default)
connection.secondaries:       --
connection.gateway-ping-timeout: 0
connection.metered:           unknown
connection.lldp:              default
connection.mdns:              -1 (default)
connection.llmnr:             -1 (default)
```

7.1.1.2.3 添加 Wi-Fi 连接

使用以下命令查看可用 Wi-Fi 访问点：

```
# nmcli dev wifi list
```

使用以下命令生成使用的静态 IP 配置，但允许自动 DNS 地址分配的 Wi-Fi 连接：

```
# nmcli con add con-name Wifi ifname wlan0 type wifi ssid MyWifi ip4
192.168.100.101/24 gw4 192.168.100.1
```

请使用以下命令设定 WPA2 密码，例如 “answer”：

```
# nmcli con modify Wifi wifi-sec.key-mgmt wpa-psk
# nmcli con modify Wifi wifi-sec.psk answer
```

使用以下命令更改 Wi-Fi 状态：

```
# nmcli radio wifi [ on | off ]
```

7.1.1.2.4 更改属性

请使用以下命令检查具体属性，比如 `mtu`：

```
# nmcli connection show id 'Wifi' | grep mtu
802-11-wireless.mtu: auto
```

使用如下命令更改设置的属性：

```
# nmcli connection modify id 'Wifi' 802-11-wireless.mtu 1350
```

使用如下命令确认更改：

```
# nmcli connection show id 'Wifi' | grep mtu
802-11-wireless.mtu: 1350
```

7.1.1.3 配置静态路由

- 使用 `nmcli` 命令为网络连接配置静态路由，使用命令如下：

```
# nmcli connection modify enp3s0 +ipv4.routes "192.168.122.0/24 10.10.10.1"
```

- 使用编辑器配置静态路由，使用如下命令：

```
# nmcli con edit type ethernet con-name enp3s0
===| nmcli interactive connection editor |===
Adding a new '802-3-ethernet' connection
Type 'help' or '?' for available commands.
Type 'describe [<setting>.<prop>]' for detailed property description.
You may edit the following settings: connection, 802-3-ethernet (ethernet),
802-1x, ipv4, ipv6, dcb
nmcli> set ipv4.routes 192.168.122.0/24 10.10.10.1
nmcli>
nmcli> save persistent
Saving the connection with 'autoconnect=yes'. That might result in an immediate
activation of the connection.
Do you still want to save? [yes] yes
Connection 'enp3s0' (1464ddb4-102a-4e79-874a-0a42e15cc3c0) successfully saved.
nmcli> quit
```

7.1.2 使用命令行

7.1.2.1 通过 `ifcfg` 文件配置网络

配置静态网络

以 `enp4s0` 网络接口进行静态网络设置为例，通过修改 `ifcfg` 文件实现，在 `/etc/sysconfig/network-scripts/` 目录中生成名为 `ifcfg-enp4s0` 的文件中，修改参数配置，示例如下：

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
```

```
BOOTPROTO=none
IPADDR=192.168.0.10
PREFIX=24
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=enp4s0static
UUID=08c3a30e-c5e2-4d7b-831f-26c3cdc29293
DEVICE=enp4s0
ONBOOT=yes
```

配置动态网络

要通过 `ifcfg` 文件为名为 `em1` 的接口配置动态网络，请按照如下操作在 `/etc/sysconfig/network-scripts/` 目录中生成名为 `ifcfg-em1` 的文件，示例如下：

```
DEVICE=em1
BOOTPROTO=dhcp
ONBOOT=yes
```

要配置一个向 DHCP 服务器发送不同的主机名的接口，请在 `ifcfg` 文件中新增一行内容，如下所示：

```
DHCP_HOSTNAME=hostname
```

要配置忽略由 DHCP 服务器发送的路由，防止网络服务使用从 DHCP 服务器接收的 DNS 服务器更新 `/etc/resolv.conf`。请在 `ifcfg` 文件中新增一行内容，如下所示：

```
PEERDNS=no
```

要配置一个接口使用具体 DNS 服务器，请将参数 `PEERDNS=no`，并在 `ifcfg` 文件中添加以下行：

```
DNS1=ip-address
DNS2=ip-address
```

其中 `ip-address` 是 DNS 服务器的地址。这样就会让网络服务使用指定的 DNS 服务器更新 `/etc/resolv.conf`。

7.1.2.2 使用 ip 命令配置网络

使用 `ip` 命令为接口配置地址，命令格式如下，其中 `interface-name` 为网卡名称。

```
ip addr [ add | del ] address dev interface-name
```

配置静态地址

在 `root` 权限下，配置设置 IP 地址，使用示例如下：

```
# ip address add 192.168.0.10/24 dev enp3s0
```

查看配置结果，使用如下命令：

```
# ip addr show dev enp3s0
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 52:54:00:aa:ad:4a brd ff:ff:ff:ff:ff:ff
    inet 192.168.202.248/16 brd 192.168.255.255 scope global dynamic noprefixroute
enp3s0
    valid_lft 9547sec preferred_lft 9547sec
    inet 192.168.0.10/24 scope global enp3s0
    valid_lft forever preferred_lft forever
    inet6 fe80::32e8:cc22:9db2:f4d4/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

配置多个地址

`ip` 命令支持为同一接口分配多个地址，可重复多次使用 `ip` 命令实现分配多个地址。使用示例如下：

```
# ip address add 192.168.2.223/24 dev enp4s0
# ip address add 192.168.4.223/24 dev enp4s0
# ip addr

3: enp4s0: <BROADCAST,MULTICAST,UP,LOWER UP> mtu 1500 qdisc fq_codel state UP group
default qlen 1000
    link/ether 52:54:00:aa:da:e2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.203.12/16 brd 192.168.255.255 scope global dynamic noprefixroute
enp4s0
    valid_lft 8389sec preferred_lft 8389sec
    inet 192.168.2.223/24 scope global enp4s0
    valid_lft forever preferred_lft forever
    inet 192.168.4.223/24 scope global enp4s0
    valid_lft forever preferred_lft forever
    inet6 fe80::1eef:5e24:4b67:f07f/64 scope link noprefixroute
    valid_lft forever preferred_lft forever
```

📖 说明

在系统重启后，通过 `ip` 命令进行的配置会丢失。

7.1.2.3 静态路由及默认网关

配置静态路由

如果需要静态路由，可使用 `ip route add` 命令在路由表中添加，使用 `ip route del` 命令删除。最常使用的 `ip route` 命令格式如下：

```
ip route [ add | del | change | append | replace ] destination-address
```

使用 `ip route` 命令显示当前的 IP 路由表。示例如下：

```
# ip route

default via 192.168.0.1 dev enp3s0 proto dhcp metric 100
default via 192.168.0.1 dev enp4s0 proto dhcp metric 101
192.168.0.0/16 dev enp3s0 proto kernel scope link src 192.168.202.248 metric 100
192.168.0.0/16 dev enp4s0 proto kernel scope link src 192.168.203.12 metric 101
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1 linkdown
```


在主机地址中添加一个静态路由，在 `root` 权限下，使用以下命令格式：

```
ip route add 192.168.2.1 via 10.0.0.1 [dev interface-name]
```

其中 `192.168.2.1` 是用点分隔的十进制符号中的 IP 地址，`10.0.0.1` 是下一个跃点，`interface-name` 是进入下一个跃点的退出接口。

要在网络中添加一个静态路由，即代表 IP 地址范围的 IP 地址，请作为 `root` 运行以下命令格式：

```
ip route add 192.168.2.0/24 via 10.0.0.1 [dev interface-name]
```

其中 `192.168.2.1` 是目标网络的 IP 地址，`10.0.0.1` 是网络前缀，`interface-name` 为网卡名称。

配置默认网关

在确定默认网关时，首先解析 `/etc/sysconfig/network` 文件，然后解析 `ifcfg` 文件，将最后读取的 `GATEWAY` 的取值作为路由表中的默认路由。

在动态网络环境中，使用 `NetworkManager` 管理主机时，建议设置为由 `DHCP` 来分配。

7.2 配置主机名

7.2.1 简介

`hostname` 有三种类型：`static`、`transient` 和 `pretty`。

- `static`：静态主机名，可由用户自行设置，并保存在 `/etc/hostname` 文件中。
- `transient`：动态主机名，由内核维护，初始是 `static` 主机名，默认值为“`localhost`”。可由 `DHCP` 或 `mDNS` 在运行时更改。
- `pretty`：灵活主机名，允许使用自由形式（包括特殊/空白字符）进行设置。静态/动态主机名遵从域名的通用限制。

📖 说明

`static` 和 `transient` 主机名只能包含 `a-z`、`A-Z`、`0-9`、“`-`”、“`_`”和“`.`”，不能在开头或结尾处使用句点，不允许使用两个相连的句点，大小限制为 64 个字符。

7.2.2 使用 `hostnamectl` 配置主机名

查看所有主机名

查看当前的主机名，使用如下命令：

```
# hostnamectl status
```

📖 说明

如果命令未指定任何选项，则默认使用 `status` 选项。

设定所有主机名

在 root 权限下，设定系统中的所有主机名，使用如下命令：

```
# hostnamectl set-hostname name
```

设定特定主机名

在 root 权限下，通过不同的参数来设定特定主机名，使用如下命令：

```
# hostnamectl set-hostname name [option...]
```

其中 option 可以是--pretty、--static、--transient 中的一个或多个选项。

如果--static 或--transient 与--pretty 选项一同使用时，则会将 static 和 transient 主机名简化为 pretty 主机名格式，使用“-”替换空格，并删除特殊字符。

当设定 pretty 主机名时，如果主机名中包含空格或单引号，需要使用引号。命令示例如下：

```
# hostnamectl set-hostname "Stephen's notebook" --pretty
```

清除特定主机名

要清除特定主机名，并将其还原为默认形式，在 root 权限下，使用如下命令：

```
# hostnamectl set-hostname "" [option...]
```

其中 "" 是空白字符串，option 是--pretty、--static 和--transient 中的一个或多个选项。

远程更改主机名

在远程系统中运行 hostnamectl 命令时，要使用-H，--host 选项，使用如下命令：

```
# hostnamectl set-hostname -H [username]@hostname new_hostname
```

其中 hostname 是要配置的远程主机，username 为自选项，new_hostname 为新主机名。hostnamectl 会通过 SSH 连接到远程系统。

7.2.3 使用 nmcli 配置主机名

查询 static 主机名，使用如下命令：

```
# nmcli general hostname
```

在 root 权限下，将 static 主机名设定为 host-server，使用如下命令：

```
# nmcli general hostname host-server
```

要让系统 hostnamectl 感知到 static 主机名的更改，在 root 权限下，重启 hostnamed 服务，使用如下命令：

```
# systemctl restart systemd-hostnamed
```

7.3 配置网络绑定

7.3.1 使用 nmcli

- 创建名为 `mybond0` 的绑定，使用示例如下：

```
# nmcli con add type bond con-name mybond0 ifname mybond0 mode active-backup
```

- 添加从属接口，使用示例如下：

```
# nmcli con add type bond-slave ifname enp3s0 master mybond0
```

要添加其他从属接口，重复上一个命令，并在命令中使用新的接口，使用示例如下：

```
# nmcli con add type bond-slave ifname enp4s0 master mybond0
Connection 'bond-slave-enp4s0' (05e56afc-b953-41a9-b3f9-0791eb49f7d3)
successfully added.
```

- 要启动绑定，则必须首先启动从属接口，使用示例如下：

```
# nmcli con up bond-slave-enp3s0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/14)
# nmcli con up bond-slave-enp4s0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/15)
```

现在可以启动绑定，使用示例如下：

```
# nmcli con up bond-mybond0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/16)
```

7.3.2 使用命令行

7.3.2.1 检查是否已安装 Bonding 内核模块

在系统中默认已加载相应模块。在 `root` 权限下，要载入绑定模块，可使用如下命令：

```
# modprobe --first-time bonding
```

显示该模块的信息，可使用如下命令：

```
# modinfo bonding
```

更多命令请使用 `modprobe --help` 查看。

7.3.2.2 创建频道绑定接口

要创建绑定接口，可通过在 `/etc/sysconfig/network-scripts/` 目录中创建名为 `ifcfg-bondN` 的文件（使用接口号码替换 `N`，比如 `0`）。

根据要绑定接口类型的配置文件来编写相应的内容，比如网络接口。接口配置文件示例如下：

```
DEVICE=bond0
NAME=bond0
TYPE=Bond
BONDING_MASTER=yes
```

```
IPADDR=192.168.1.1
PREFIX=24
ONBOOT=yes
BOOTPROTO=none
BONDING_OPTS="bonding parameters separated by spaces"
```

7.3.2.3 创建从属接口

创建频道绑定接口后，必须在从属接口的配置文件中添加 **MASTER** 和 **SLAVE** 指令。

例如将两个网络接口 **enp3s0** 和 **enp4s0** 以频道方式绑定，其配置文件示例分别如下：

```
TYPE=Ethernet
NAME=bond-slave-enp3s0
UUID=3b7601d1-b373-4fdf-a996-9d267d1cac40
DEVICE=enp3s0
ONBOOT=yes
MASTER=bond0
SLAVE=yes
TYPE=Ethernet
NAME=bond-slave-enp4s0
UUID=00f0482c-824f-478f-9479-abf947f01c4a
DEVICE=enp4s0
ONBOOT=yes
MASTER=bond0
SLAVE=yes
```

7.3.2.4 激活频道绑定

要激活绑定，则需要启动所有从属接口。请在 **root** 权限下，运行以下命令：

```
# ifup enp3s0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/7)
# ifup enp4s0
Connection successfully activated (D-Bus active path:
/org/freedesktop/NetworkManager/ActiveConnection/8)
```

说明

对于已经处于“up”状态的接口，请首先使用“**ifdown enp3s0**”命令修改状态为 down，其中 **enp3s0** 为实际网卡名称。

完成后，启动所有从属接口以便启动绑定（不将其设定为“down”）。

要让 **NetworkManager** 感知到系统所做的修改，在每次修改后，请在 **root** 权限下，运行以下命令：

```
# nmcli con load /etc/sysconfig/network-scripts/ifcfg-device
```

查看绑定接口的状态，请运行以下命令：

```
# ip link show

1: lo: <LOOPBACK,UP,LOWER UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode
DEFAULT group default qlen 1000
    link/ether 52:54:00:aa:ad:4a brd ff:ff:ff:ff:ff:ff
3: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode
DEFAULT group default qlen 1000
    link/ether 52:54:00:aa:da:e2 brd ff:ff:ff:ff:ff:ff
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
mode DEFAULT group default qlen 1000
    link/ether 86:a1:10:fb:ef:07 brd ff:ff:ff:ff:ff:ff
5: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc fq_codel master virbr0 state
DOWN mode DEFAULT group default qlen 1000
    link/ether 52:54:00:29:35:4c brd ff:ff:ff:ff:ff:ff
```

7.3.2.5 创建多个绑定

系统会为每个绑定创建一个频道绑定接口，包括 `BONDING_OPTS` 指令。使用这个配置方法可让多个绑定设备使用不同的配置。请按照以下操作创建多个频道绑定接口：

- 创建多个 `ifcfg-bondN` 文件，文件中包含 `BONDING_OPTS` 指令，让网络脚本根据需要创建绑定接口。
- 创建或编辑要绑定的现有接口配置文件，添加 `SLAVE` 指令。
- 使用 `MASTER` 指令工具在频道绑定接口中分配要绑定的接口，即从属接口。

以下是频道绑定接口配置文件示例：

```
DEVICE=bondN
NAME=bondN
TYPE=Bond
BONDING MASTER=yes
IPADDR=192.168.1.1
PREFIX=24
ONBOOT=yes
BOOTPROTO=none
BONDING_OPTS="bonding parameters separated by spaces"
```

在这个示例中，使用绑定接口的号码替换 `N`。例如要创建两个接口，则需要使用正确的 IP 地址创建两个配置文件 `ifcfg-bond0` 和 `ifcfg-bond1`。

7.4 IPv6 使用差异说明（vs IPv4）

7.4.1 约束限制

- `chrony` 支持全局地址（`global address`），不支持链路本地地址（`link-local address`）。
- `Firefox` 支持通过 `http/https` 协议访问全局地址（`global address`），不支持链路本地地址（`link-local address`）。

7.4.2 配置说明

7.4.2.1 设置接口设备 MTU 值

概述

IPv6 场景中会发现整个路由路径中的最小 mtu 的值作为当前链接的 PMTU 的值，源端根据 PMTU 的值确定是否进行分片发送，而在整个路径中的其它设备将不再需要进行分片处理，从而可以降低中间路由设备的负载大小。其中 IPv6 PMTU 设置的最小值为 1280。

设置接口设备的 mtu

如果在配置了 IPv6 地址的接口上设置 mtu 的值小于 1280（IPv6 PMTU 设置的最小值），则会导致该接口的 IPv6 地址被删除。并且无法再次添加 IPv6 地址。所以在 IPv6 场景中，对接口设备的 mtu 的配置一定要大于等于 1280。具体现象如下：

```
# ip addr show enp3s0
3: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic
enp3s0
    valid_lft 38663sec preferred_lft 38663sec
    inet6 2001:222::2/64 scope global
    valid_lft forever preferred_lft forever
# ip link set dev enp3s0 mtu 1200
# ip addr show enp3s0
3: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1200 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic
enp3s0
    valid_lft 38642sec preferred_lft 38642sec
# ip addr add 2001:222::2/64 dev enp3s0
RTNETLINK answers: No buffer space available
# ip link set dev enp3s0 mtu 1500
# ip addr show enp3s0
3: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic
enp3s0
    valid_lft 38538sec preferred_lft 38538sec
# ip addr add 2001:222::2/64 dev enp3s0
# ip addr show enp3s0
3: enp3s0: <BROADCAST,MULTICAST,UP,LOWER UP> mtu 1500 qdisc pfifo fast state UP
group default qlen 1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic
enp3s0
    valid_lft 38531sec preferred_lft 38531sec
```

```
inet6 2001:222::2/64 scope global
    valid_lft forever preferred_lft forever
```

7.4.2.2 有状态自动配置 IPv6 地址

概述

IPv6 与 IPv4 都可以通过 DHCP 的方式获得 IP 地址。IPv6 地址有两种配置方式：无状态自动配置和有状态自动配置。

- 无状态自动配置
不需要 DHCP 服务进行管理，设备根据网络 RA（路由公告）获得网络前缀，或者 link-local 地址为固定 fe80::。而接口 ID 则根据 ifcfg 配置 IPV6_ADDR_GEN_MODE 的具体设置来进行自动获得：
 - a. IPV6_ADDR_GEN_MODE="stable-privacy" 则根据设备及网络环境来确定一个随机接口 ID。
 - b. IPV6_ADDR_GEN_MODE="EUI64" 则根据设备 MAC 地址来确定接口 ID。
- 有状态自动配置：需要 DHCP 服务器进行管理分配，服从 DHCPv6 协议来从 DHCPv6 服务器端租赁 IPv6 地址。

在有状态自动配置 IPv6 地址时，DHCPv6 服务端可以通过客户端设置的 vendor class 将客户端进行分类，不同类别分配不同地址段的 IPv6 地址。在 IPv4 场景中，客户端可以直接用 dhclient 的 -V 选项来设置 vendor-class-identifier，DHCP 服务端在配置文件中根据 vendor-class-identifier 来对客户端进行分类处理。而 IPv6 场景中，如果使用同样的方法对客户端分类，则分类并不会生效。

```
dhclient -6 <interface> -V <vendor-class-identifier string> <interface>
```

这是由于 DHCPv6 和 DHCP 协议存在较大差异，DHCPv6 的可选项中使用 vendor-class-option 替代了 DHCP 中的 vendor-class-identifier。而 dhclient 的 -V 选项并不能设置 vendor-class-option。

有状态自动配置 IPv6 地址时 dhclient 设置 vendor class 方法

- 在客户端使用配置文件方式添加对 vendor class 的设置，使用方法如下：
客户端配置文件（/etc/dhcp/dhclient6.conf），文件位置可以自定义，在使用时需要通过 dhclient -cf 选项来指定配置文件：

```
option dhcp6.vendor-class code 16 = {integer 32, integer 16, string};
interface "enp3s0" {
    send dhcp6.vendor-class <Enterprise-ID number> <vendor class string
length> <vendor class string>;
}
```

📖 说明

- <Enterprise-ID number>，32 位整型数字，企业标识号，企业通过 IANA 注册。
- <vendor class string length>，16 位整型数字，vendor class 字符串长度。
- <vendor class string>，要设置的 vendor class 字符串，例如：“HWHW”。

客户端使用方法：

```
dhclient -6 <interface> -cf /etc/dhcp/dhclient6.conf
```

- DHCPv6 服务端配置文件 (/etc/dhcp/dhcpd6.conf)，需要 dhcpd -cf 选项来指定该配置文件：

```
option dhcp6.vendor-class code 16 = {integer 32, integer 16, string};
subnet6 fc00:4:12:ffff::/64 {
    class "hw" {
        match if substring ( option dhcp6.vendor-class, 6, 10 ) = "HWHW";
    }
    pool6 {
        allow members of "hw";
        range6 fc00:4:12:ffff::ff10 fc00:4:12:ffff::ff20;
    }
    pool6 {
        allow unknown clients;
        range6 fc00:4:12:ffff::100 fc00:4:12:ffff::120;
    }
}
```

说明

substring (option dhcp6.vendor-class, 6, 10) 其中子字符串的开始位置为 6，因为前面包含 4 个字节的<Enterprise-ID number>和 2 个字节的<string length>。而子字符串的结束位置位：6+<vendor class string length>。这里 vendor class string 为 “HWHW”，字符串的长度为 4，所以子字符串的结束位置为 6+4=10。用户可以根据实际需要来确定<vendor class string>及相应的<vendor class string length>。

服务端使用方法：

```
dhcpd -6 -cf /etc/dhcp/dhcpd6.conf <interface>
```

7.4.2.3 内核支持 socket 相关系统调用

概述

IPv6 地址长度扩展到 128 比特，所以有足够的 IPv6 地址可供分配使用。同时 IPv6 头相比 IPv4 头进行了简化，并增强了 IPv6 的自动配置功能。IPv6 地址分为单播地址，组播地址和任意播地址。常用的单播地址又包含：链路本地地址（link-local address），唯一本地地址（Unique local address）和全局地址（global address）。由于 IPv6 的全局地址十分充足，唯一本地地址一般不被使用（其前身为站点本地地址（site-local address），已于 2004 年被废弃）。当前主要使用的单播地址为：链路本地地址（link-local address）和全局地址（global address）。当前内核支持 socket 系统调用，在使用单播地址的链路本地地址和全局地址时存在差异。

link-local 地址和 global 地址在 socket 调用时的差异

RFC 2553: Basic Socket Interface Extensions for IPv6 定义 sockaddr_in6 的数据结构如下：

```
struct sockaddr_in6 {
    uint8_t      sin6_len;      /* length of this struct */
    sa_family_t  sin6_family;   /* AF_INET6 */
    in_port_t    sin6_port;     /* transport layer port # */
    uint32_t     sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr;   /* IPv6 address */
    uint32_t     sin6_scope_id; /* set of interfaces for a scope */
};
```


说明

`sin6_scope_id`: 32 位整型，对于链路本地地址（link-local address），对于链路范围的 `sin6_addr`，它可以用来标识指定的接口索引号。如果是站点范围的 `sin6_addr`，则用来作为站点的标识符（站点本地地址已被抛弃）。

在使用 link-local 地址进行 socket 通信时，在构造目的地址时，需要制定该地址所对应的接口索引号。一般可以通过 `if_nametoindex` 函数将接口名转化为接口索引号。具体方式如下，

```
int port = 1234;
int sk fd;
int iff index = 0;
char iff name[100] = "enp3s0";
char * ll addr[100] = "fe80::123:456:789";
struct sockaddr_in6 server_addr;

memset(&server_addr, 0, sizeof(struct sockaddr_in6));
iff_index=if_nametoindex(iff_name);

server_addr.sin6_family=AF_INET6;
server_addr.sin6_port=htons(port);
server_addr.sin6_scope_id=iff_index;
inet_pton(AF_INET6, ll_addr, &(server_addr.sin6_addr));

sk_fd=socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
connect(sk_fd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr_in6));
```

7.4.2.4 IPv4 的 dhclient 守护进程持久化配置

概述

通过 NetworkManager 服务来管理网络服务时，如果接口 `ifcfg-<interface-name>` 配置文件中配置了 DHCP 方式获得 IP 地址，则相应地 NetworkManager 服务会拉起 `dhclient` 守护进程来通过 DHCP 协议方式来从 DHCP 服务器获取 IP 地址。

`dhclient` 提供了 "-1" 选项来决定 `dhclient` 进程在未获得 DHCP 服务响应时，是会不断持久化尝试请求地址还是会尝试时间超时后退出。针对 IPv4 的 `dhclient` 守护进程，可以在 `ifcfg-<interface-name>` 配置文件中设置 `PERSISTENT_DHCLIENT` 来决定是否设置 IPv4 的 `dhclient` 进程的持久化。

约束限制

1. 当 `dhclient` 进程在运行中被杀死，`network` 服务无法自动将其拉起，可靠性需要用户自己保障。
2. 配置了持久化选项 `PERSISTENT_DHCLIENT`，需要确保有相应的 DHCP 服务器。如果在拉起 `network` 时无可用 DHCP 服务器，`dhclient` 进程不断尝试发送请求包但无回应，则会导致 `network` 服务卡死直到 `network` 服务超时失败。由于 `network` 服务在拉起多个网卡的 IPv4 `dhclient` 进程时，是通过串行的方式来拉起的。如果有网卡配置了持久化而 DHCP 服务器没有准备好，则会导致 `network` 服务在给该网卡获取 IPv4 地址超时卡死，进而导致后续网卡无法获得 IPv4/IPv6 地址。

以上两种约束限制是特殊的应用场景，需要用户自己进行可靠性保障。

IPv4 DHCP 和 IPv6 DHCPv6 方式获取地址的配置差异

可以通过配置接口 `ifcfg-<interface-name>` 参数来分别实现 IPv4 和 IPv6 通过 DHCP/DHCPv6 协议来动态获取 IP 地址，具体配置说明如下：

```
BOOTPROTO=none|bootp|dhcp
DHCPV6C=yes|no
PERSISTENT_DHCLIENT=yes|no|1|0
```

- **BOOTPROTO:** none 表示静态配置 IPv4 地址，bootp|dhcp 则会拉起 DHCP dhclient 来动态获取 IPv4 地址。
- **DHCPV6C:** no 表示静态配置 IPv6 地址，yes 则会拉起 DHCPv6 dhclient 来动态获取 IPv6 地址。
- **PERSISTENT_DHCLIENT:** no|0 表示 IPv4 的 dhclient 进程配置为“非持久化”，当 dhclient 向 DHCP 服务器发送一次请求报文而无响应，则会间隔一段时间后退出，退出值为 2。yes|1 则表示 IPv4 的 dhclient 进程配置为“持久化”，dhclient 会向 DHCP 服务器反复发送请求报文。**如果没有配置 PERSISTENT_DHCLIENT 项，则 IPv4 的 dhclient 会默认设置为“持久化”。**

📖 说明

PERSISTENT_DHCLIENT 配置只针对 IPv4 生效，对 IPv6 相关 dhclient -6 进程不生效，IPv6 默认不进行持久化配置。

7.4.2.5 iproute 相关命令配置 IPv4 与 IPv6 时的差异说明

概述

由于 IPv4 和 IPv6 是两个不同的协议标准，iproute 相关命令在使用方法上存在一定的差异。本章节主要梳理 iproute 包中用户经常使用到命令在 IPv4 和 IPv6 使用方面的差异，从而可以更好地指导用户使用 iproute 包中相关命令。

IPv6 地址的生命周期

IPv6 状态	解释
tentative	临时状态：刚添加地址还处于地址重复检测 DAD 过程
preferred	首选状态：完成 DAD 过程，没有收到相应的 NA 报文，表示该地址没有冲突。
deprecated	弃用状态：地址有一定的使用时限（valid_lft 和 preferred_lft），preferred_lft 到期后地址会变化 deprecated 状态。 该状态下的地址不能用于创建新的连接，但是原有的连接可以继续使用。
invalid	无效状态：使用时限超过 preferred_lft 一段时间后仍然没有成功进行租约续约，则 valid_lft 时间到后地址状态会被设置为 invalid，表示该地址不可以再被使用。

其它说明：

- **preferred_lft:** preferred lifetime, 地址为首选状态的寿命, **preferred_lft** 没有到期的地址可以用于正常通信使用, 若有多个 **preferred** 地址则按照内核具体机制选择地址。
- **valid_lft:** valid lifetime, 地址有效的寿命, 在[**preferred_lft**, **valid_lft**]时间段内该地址不能被用于新建连接, 已经创建的连接继续有效。

ip link 命令

命令:

```
ip link set IFNAME mtu MTU
```

IPv6 中 PMTU 的最小值为 1280, 如果 **mtu** 值设置小于 1280 则会导致 IPv6 地址丢失。其它设备无法 ping 通该 IPv6 地址。

ip addr 命令

1. 命令:

```
ip [-6] addr add IFADDR dev IFNAME
```

添加 IPv6 地址可以选择添加 -6 选项也可以不添加, **ip addr** 命令会根据具体地址类型来判断是 ipv4 地址还是 IPv6 地址。

如果指定“-6”选项, 但是 **IFADDR** 是 ipv4 地址则会有错误返回。

2. 命令:

```
ip [-6] addr add IFADDR dev IFNAME [home|nodad]
```

[**home|nodad**] 选项只针对 IPv6 地址有效。

- **home:** 将该地址指定为 RFC 6275 中定义的家庭地址。(这是移动节点从家庭链路获取的地址, 是移动节点的永久地址, 如果移动节点保持在相同的归属链路中, 则各种实体之间的通信照常进行。)
- **nodad:** 配置该项(仅限 IPv6)添加此地址时不执行重复地址检测 DAD (RFC 4862)。如果一台设备上多个接口通过 **nodad** 配置了多个相同的 IPv6 地址, 则会按照接口顺序使用该 IPv6 地址。同一个接口上不能添加一个 **nodad** 一个非 **nodad** 的相同 IPv6 地址。因为两个地址是一样的, 所以会报“RTNETLINK answers: File exists”。

3. 命令:

```
ip [-6] addr del IFADDR dev IFNAME
```

删除 IPv6 地址可以选择添加 -6 选项也可以不添加, **ip addr del** 命令会根据具体地址类型来判断是 ipv4 地址还是 IPv6 地址。

4. 命令:

```
ip [-6] addr show dev IFNAME [tentative|-tentative|deprecated|-  
deprecated|dadfailed|-dadfailed|temporary]
```

- 不指定 -6 选项, 则会同时打印 IPv4 和 IPv6 地址。指定 -6 选项则只打印 IPv6 地址。
- [**tentative|-tentative|deprecated|-deprecated|dadfailed|-dadfailed|temporary**], 这些选项只针对 IPv6, 可以根据 IPv6 地址状态对地址进行筛选查看。
 - i. **tentative:** (仅限 IPv6) 仅列出尚未通过重复地址检测的地址。

- ii. `-tentative`: (仅限 IPv6) 仅列出当前未处于重复地址检测过程中的地址。
- iii. `deprecated`: (仅限 IPv6) 仅列出已弃用的地址。
- iv. `-deprecated`: (仅限 IPv6) 仅列出未弃用的地址。
- v. `dadfailed`: (仅限 IPv6) 仅列出重复地址检测失败的地址。
- vi. `-dadfailed`: (仅限 IPv6) 仅列出未重复地址检测失败的地址。
- vii. `temporary`: (仅限 IPv6) 仅列出临时地址

ip route 命令

1. 命令:

```
ip [-6] route add ROUTE [mtu lock MTU]
```

- `-6` 选项: 添加 IPv6 路由可以选择添加 `-6` 选项也可以不添加, `ip route` 命令会根据具体地址类型来判断是 IPv4 地址还是 IPv6 地址。
- `mtu lock MTU`: 锁定路由的 MTU 值。如果不锁定 MTU, 则 MTU 的值则可能在 PMTUD 过程中被内核改变。如果锁定 MTU, 则不会尝试 PMTUD, 所有 IPv4 包都将不设置 DF 位发出, IPv6 包则会按照 MTU 进行分段处理。

2. 命令:

```
ip [-6] route del ROUTE
```

删除 IPv6 路由可以选择添加 `-6` 选项也可以不添加, `ip route` 命令会根据具体地址类型来判断是 IPv4 地址还是 IPv6 地址。

ip rule 命令

1. 命令:

```
ip [-6] rule list
```

`-6` 选项: 设置 `-6` 选项打印 IPv6 的策略路由, 不设置 `-6` 选项打印 IPv4 的策略路由。所以需要根据具体协议类型来配置 `-6` 选项。

2. 命令:

```
ip [-6] rule [add|del] [from|to] ADDR table TABLE pref PREF
```

`-6` 选项: IPv6 相关的策略路由表项需要设置 `-6` 选项, 否则会报错: “Error: Invalid source address.”。相应地, IPv4 相关的策略路由表项不可以设置 `-6` 选项, 否则会报错: “Error: Invalid source address.”。

7.4.2.6 NetworkManager 服务配置差异说明

概述

NetworkManager 服务使用 `ifup/ifdown` 的逻辑接口定义进行高级网络设置。其参数大多数都是在 `/etc/sysconfig/network` 和 `/etc/sysconfig/network-scripts/ifcfg-<interface-name>` 两个配置文件设置。前者为全局设置, 后者为指定网卡的设置, 当两者有冲突时, 后者生效。

配置差异说明

其中在 `/etc/sysconfig/network` 下的配置差异有:

IPv4	IPv6	含义说明
NA	IPV6FORWARDING=yes no	IPv6 转发，默认不转发。
NA	IPV6_AUTOCONF=yes no	IPv6 转发打开是 no，否则是 yes。
NA	IPV6_ROUTER=yes no	IPv6 转发打开是 yes，否则是 no。
NA	IPV6_AUTOTUNNEL=yes no	指定 Tunnel 为自动隧道模式，默认是 no。
GATEWAY	IPV6_DEFAULTGW=<IPv6 address[%interface]> (optional)	在 IPv6 中设置默认网关。
NA	IPV6_DEFAULTDEV=<interface > (optional)	指定默认转发的网卡。
NA	IPV6_RADVD_PIDFILE=<pid-file> (optional)	默认 ipv6_radvd_pid 路径： /var/run/radvd/radvd.pid。
NA	IPV6_RADVD_TRIGGER_ACTION=start stop reload restart SIGHUP (optional)	radvd 默认触发动作。

而在/etc/sysconfig/network-scripts/ifcfg-<interface-name>下的差异主要有：

IPv4	IPv6	含义说明
IPADDRn	IPV6ADDR=<IPv6 address>[/<prefix length>]	ip 地址。
PREFIXn	NA	网络前缀，网络别名和 ppp 无效，优先级高于 NETMASK。
NETMASKn	NA	子网掩码，仅用于别名和 ppp。
GATEWAY	IPV6_DEFAULTGW=<IPv6 address[%interface]> (optional)	默认网关。
MTU	IPV6_MTU=<MTU of link> (optional)	默认 MTU。
IPV4_FAILURE_FATAL=yes no	IPV6_FAILURE_FATAL	默认值是 no。若设置为 yes，dhclient 失败 ifup-eth 会直接退出。
NA	IPV6_PRIVACY=rfc3041	默认禁用。
NA	IPV6INIT=yes no	默认开启 IPv6。

IPv4	IPv6	含义说明
NA	IPV6FORWARDING=yes no	默认关闭，已废弃。

7.4.3 FAQ

7.4.3.1 iscsi-initiator-utils 不支持登录 fe80 IPv6 地址

问题现象

客户端通过 IPv6 登录 iscsi 服务端时，使用如 “iscsiadm -m node -p ipv6address -l” 的命令格式登录，如果是全局地址（global address），直接替换将命令范例中的 “ipv6address” 替换为全局地址即可；但如果是链路本地地址（link-local address，fe80 开头的 IPv6 地址）则无法使用，因为 iscsi-initiator-utils 目前机制还不支持用链路本地地址（link-local address）地址登录 iscsi 服务端。

原因分析

如果使用格式如 “iscsiadm -m node -p fe80::xxxx -l” 登录，会登录超时返回，这是因为使用链路本地地址必须指定接口，否则使用 iscsi_io_tcp_connect 函数调用 connect 函数会失败，并且产生标准错误码 22。

如果使用格式如 “iscsiadm -m node -p fe80::xxxx%enp3s0 -l” 登录时，iscsi_addr_match 函数会将地址 “fe80::xxxx%enp3s0” 与服务端返回的 node 信息中的地址 “fe80::xxxx” 对比，对比结果不匹配，导致登录失败。

因此，iscsi-initiator-utils 目前机制还不支持用链路本地地址（link-local address）地址登录 iscsi 服务端。

7.4.3.2 网卡 down 掉之后，IPv6 地址丢失

问题现象

通过 ip link down+up 网卡或 ifconfig down+up 网卡命令，将网卡 down 掉之后再上线，查看网卡上配置的 ip 地址，发现 ipv4 地址不丢失，而配置的 IPv6 地址丢失。

原因分析

内核中的处理逻辑为如果网卡设置为 down 状态，会清空所有 IPv4 及 IPv6 地址，将网卡重新 up 之后，ipv4 地址自动恢复，网卡上自动配置的 IPv6 链路本地地址也会恢复，但是其他 IPv6 地址默认会丢失。如果需要保留这些 IPv6 地址，可以通过 “sysctl -w net.ipv6.conf.<网卡名>.keep_addr_on_down=1” 来实现。

7.4.3.3 bond 口已具有多个 IPv6 地址时，添加或删除 IPv6 地址耗时过久

问题现象

下列方式配置或删除（包括 flush）IPv6 地址方式，X 为动态变化的低 16 位，并且配置在 bond 口时，耗时会随已配置的 IPv6 地址数量成数倍增加。例如由 4 个物理网卡组成的 bond 口添加 IPv6 地址时，单线程添加删除 3000 IPv6 地址均需大概 5 分钟，而普通物理网卡耗时在 10 秒内。

```
ip a add/del 192:168::18:X/64 dev DEVICE
```

原因分析

bond 口在添加 IPv6 地址时，会生成 IPv6 组播地址，并进行同步到所有的物理网卡上，此耗时会随 IPv6 数量增加而增加，导致耗时过长。

解决方法

IPv6 的组播地址是由 IPv6 地址的低 24 位与 33-33-ff 组合生成，组播地址过多会导致添加删除耗时增加，如果生成的组播地址为少量，耗时不会受此影响。

建议添加 IPv6 地址时，可保持低 24 位一致，保持高位变动，单网卡中仅需一个网段的一个地址即可与外部正常通信，此配置更符合常规使用。

7.4.3.4 Rsyslog 在 IPv4 和 IPv6 混合使用场景中日志传输延迟

问题现象

rsyslog 客户端配置文件同时配置 IPv4 和 IPv6 地址，且端口配置相同的情况下，服务端收集 log 时会概率性出现日志打印延迟。

原因分析

延迟是因为 rsyslog 内部存在缓冲队列机制，默认情况下需要缓冲区队列达到一定数量才会写入文件。

解决方法

可通过配置 Direct 模式，关闭缓冲队列机制解决该问题。在 rsyslog 远程传输服务端的 /etc/rsyslog.d 目录下新增的远程传输配置文件中，最开头增加如下配置：

```
$ActionQueueType Direct  
$MainMsgQueueType Direct
```

📖 说明

- Direct 模式减少队列大小为 1，所以在队列中会保留 1 条日志到下次日志打印；
- Direct 模式会降低服务器端的 rsyslog 性能。

8 搭建服务

8.1 搭建 repo 服务器

8.2 搭建 FTP 服务器

8.3 搭建 web 服务器

8.1 搭建 repo 服务器

8.1.1 概述

将 openEuler 提供的镜像 openEuler-1.0-base-aarch64-dvd.iso 创建为 repo 源，如下以使用 nginx 进行 repo 源部署，提供 http 服务为例进行说明。

8.1.2 创建/更新本地 repo 源

使用 mount 挂载，将 openEuler 的镜像 openEuler-1.0-base-aarch64-dvd.iso 创建为 repo 源，并能够对 repo 源进行更新。

8.1.2.1 获取 ISO 镜像

获取包路径

请从如下网址获取 openEuler 软件包：

<https://openeuler.org/zh/download.html>

8.1.2.2 挂载 ISO 创建 repo 源

使用 mount 命令挂载镜像文件。

示例如下：

```
mount /home/openEuler/openEuler-1.0-base-aarch64-dvd.iso /mnt/
```

挂载好的 mnt 目录如下：


```
.
├── boot.catalog
├── EFI
├── images
├── Packages
├── repodata
├── TRANS.TBL
└── RPM-GPG-KEY-openEuler
```

其中，Packages 为 rpm 包所在的目录，repodata 为 repo 源元数据所在的目录，RPM-GPG-KEY-openEuler 为 openEuler 的签名公钥。

8.1.2.3 创建本地 repo 源

可以拷贝镜像中相关文件至本地目录以创建本地 repo 源，示例如下：

```
mount /home/openEuler/openEuler-1.0-base-aarch64-dvd.iso /mnt/
mkdir -p /srv/repo/
cp -r /mnt/Packages /srv/repo/
cp -r /mnt/repodata /srv/repo/
cp -r /mnt/RPM-GPG-KEY-openEuler /srv/repo/
```

从而本地 repo 目录如下：

```
.
├── Packages
├── repodata
└── RPM-GPG-KEY-openEuler
```

Packages 为 rpm 包所在的目录，repodata 为 repo 源元数据所在的目录，RPM-GPG-KEY-openEuler 为 openEuler 的签名公钥。

8.1.2.4 更新 repo 源

更新 repo 源有两种方式：

- 通过新版本的 ISO 更新已有的 repo 源，与创建 repo 源的方式相同，即挂载镜像或者重新拷贝镜像至本地目录
- 在 repo 源的 Packages 目录下添加 rpm 包，然后更新 repo 源，可通过 createrepo 命令更新 repo 源

```
dnf install createrepo
createrepo --update --workers=10 /srv/repo
```

其中，--update 表示更新，--workers 表示线程数，可自定义。

8.1.3 部署远端 repo 源

安装操作系统 openEuler1.0 (openEuler-1.0-base-aarch64-dvd.iso)，在 openEuler1.0 上通过 nginx 部署 repo 源。

8.1.3.1 nginx 安装与配置

1. 请自行下载 nginx 工具并安装 nginx。
2. 安装 nginx 之后，配置/etc/nginx/nginx.conf。

📖 说明

文档中的配置内容仅供参考，请用户根据实际情况（例如安全加固需要）进行配置。

```
user root;
worker processes auto;           # 建议设置为 core-1
error log /var/log/nginx/error.log warn; # log 存放位置
pid /var/run/nginx.pid;

events {
    worker connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default type application/octet-stream;

    log format main '$remote_addr - $remote_user [$time local] "$request" '
        '$status $body bytes sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;
    sendfile on;
    keepalive_timeout 65;

    server {
        listen 80;
        server_name localhost; # 服务器名 (url)
        client_max_body_size 4G;
        root /srv/repo; # 服务默认目录

        location / {
            autoindex on; # 开启访问目录下层文件
            autoindex_exact_size on;
            autoindex_localtime on;
        }
    }
}
```

8.1.3.2 启动 nginx 服务

1. 通过 systemd 启动 nginx 服务：

```
systemctl enable nginx
systemctl start nginx
```

2. nginx 是否启动成功可通过下面命令查看：

```
systemctl status nginx
```

- 图 8-1 表示 nginx 服务启动成功

图8-1 nginx 服务启动成功

```
[root@localhost ~]# systemctl status nginx
● nginx.service - SYSV: Nginx is an HTTP(S) server, HTTP(S) reverse proxy and IMAP/POP3 proxy server
   Loaded: loaded (/etc/rc.d/init.d/nginx)
   Active: active (running) since Wed 2016-12-21 05:20:31 EST; 2 days ago
     Docs: man:systemd-sysv-generator(8)
  Main PID: 1965 (nginx)
    CGroup: /system.slice/system-hostos.slice/nginx.service
           └─1965 nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.conf
             └─1967 nginx: worker process

Dec 21 05:20:30 localhost.localdomain systemd[1]: Starting SYSV: Nginx is an HTTP(S) s...
Dec 21 05:20:31 localhost.localdomain nginx[1446]: Starting nginx: [ OK ]
Dec 21 05:20:31 localhost.localdomain systemd[1]: Started SYSV: Nginx is an HTTP(S) se...
Hint: Some lines were ellipsized, use -l to show in full.
```

– 若 nginx 服务启动失败，查看错误信息：

```
systemctl status nginx.service --full
```

图8-2 nginx 服务启动失败

```
[root@localhost ~]# systemctl status nginx.service --full
● nginx.service - SYSV: Nginx is an HTTP(S) server, HTTP(S) reverse proxy and IMAP/POP3 proxy server
   Loaded: loaded (/etc/rc.d/init.d/nginx)
   Active: failed (Result: exit-code) since Thu 2016-12-08 06:13:45 EST; 3min 8s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 24340 ExecStart=/etc/rc.d/init.d/nginx start (code=exited, status=1/FAILURE)

Dec 08 06:13:45 localhost.localdomain systemd[1]: Starting SYSV: Nginx is an HTTP(S) server, HTTP(S) reverse proxy and IMAP/POP3 proxy server...
Dec 08 06:13:45 localhost.localdomain nginx[24340]: Starting nginx: nginx: [emerg] mkdir(
"/var/spool/nginx/tmp/client_body" failed (13: Permission denied)
Dec 08 06:13:45 localhost.localdomain nginx[24340]: [FAILED]
Dec 08 06:13:45 localhost.localdomain systemd[1]: nginx.service: control process exited, code=exited status=1
Dec 08 06:13:45 localhost.localdomain systemd[1]: Failed to start SYSV: Nginx is an HTTP(S) server, HTTP(S) reverse proxy and IMAP/POP3 proxy server.
Dec 08 06:13:45 localhost.localdomain systemd[1]: Unit nginx.service entered failed state.
Dec 08 06:13:45 localhost.localdomain systemd[1]: nginx.service failed.
```

如图 8-2 所示 nginx 服务创建失败，是由于目录/var/spool/nginx/tmp/client_body 创建失败，手动进行创建，类似的问题也这样处理：

```
mkdir -p /var/spool/nginx/tmp/client_body
mkdir -p /var/spool/nginx/tmp/proxy
mkdir -p /var/spool/nginx/tmp/fastcgi
mkdir -p /usr/share/nginx/uwsgi_temp
mkdir -p /usr/share/nginx/scgi_temp
```

8.1.3.3 repo 源部署

1. 创建 nginx 配置文件/etc/nginx/nginx.conf 中指定的目录/srv/repo:

```
mkdir -p /srv/repo
```

2. SELinux 设置为宽容模式:

```
setenforce permissive
```

说明

repo server 重启后，需要重新设置。

3. 设置防火墙规则，开启 nginx 设置的端口（此处为 80 端口），通过 firewall 设置端口开启：

```
firewall-cmd --add-port=80/tcp --permanent
firewall-cmd --reload
```

查询 80 端口是否开启成功，输出为 yes 则表示 80 端口开启成功：

```
firewall-cmd --query-port=80/tcp
```

也可通过 iptables 来设置 80 端口开启：

```
iptables -I INPUT -p tcp --dport 80 -j ACCEPT
```

4. nginx 服务设置好之后，即可通过 ip 直接访问网页，如图 8-3：

图8-3 nginx 部署成功



5. 通过下面几种方式将 repo 源放入到/srv/repo 下：

- 拷贝镜像中相关文件至在/srv/repo 下

```
mount /home/openEuler/openEuler-1.0-base-aarch64-dvd.iso /mnt/
cp -r /mnt/Packages /srv/repo/
cp -r /mnt/repodata /srv/repo/
cp -r /mnt/RPM-GPG-KEY-openEuler /srv/repo/
```

openEuler-1.0-base-aarch64-dvd.iso 存放在/home/openEuler 目录下。

- 在/srv/repo 下创建 repo 源的软链接

```
ln -s /home/openEuler/os /srv/repo/os
```

/home/openEuler/os 为已经创建好的 repo 源，/srv/repo/os 将指向 /home/openEuler/os。

8.1.4 使用 repo 源

repo 可配置为 yum 源，yum（全称为 Yellow dog Updater, Modified）是一个 Shell 前端软件包管理器。基于 RPM 包管理，能够从指定的服务器自动下载 RPM 包并且安装，可以自动处理依赖性关系，并且一次安装所有依赖的软件包，无须繁琐地一次次下载和安装。

8.1.4.1 repo 配置为 yum 源

构建好的 repo 可以配置为 yum 源使用，在/etc/yum.repos.d/目录下创建***.repo 的配置文件（必须以.repo 为扩展名），分为本地和 http 服务器配置 yum 源两种方式：

- 配置本地 yum 源

在/etc/yum.repos.d目录下创建 openEuler-1.0-Base.repo 文件，使用构建的本地 repo 作为 yum 源，openEuler-1.0-Base.repo 的内容如下：

```
[base]
name=base
baseurl=file:///srv/repo
enabled=1
gpgcheck=1
gpgkey=file:///srv/repo/RPM-GPG-KEY-openEuler
```

📖 说明

gpgcheck 可设置为 1 或 0，1 表示进行 gpg（GNU Private Guard）校验，0 表示不进行 gpg 校验，gpgcheck 可以确定 rpm 包的来源是有效和安全的。

gpgkey 为签名公钥的存放路径。

- 配置 http 服务器 yum 源

在/etc/yum.repos.d目录下创建 openEuler-1.0-Base.repo 文件，使用 http 服务端的 repo 作为 yum 源，openEuler-1.0-Base.repo 的内容如下：

```
[base]
name=base
baseurl=http://192.168.1.2/
enabled=1
gpgcheck=1
gpgkey=http://192.168.1.2/RPM-GPG-KEY-openEuler
```

📖 说明

“192.168.1.2”为示例地址，请用户根据实际情况进行配置。

8.1.4.2 repo 优先级

当有多个 repo 源时，可通过在.repo 文件的 priority 参数设置 repo 的优先级（如果不设置，默认优先级是 99，当相同优先级的源中存在相同 rpm 包时，会安装最新的版本）。其中，1 为最高优先级，99 为最低优先级，如给 openEuler-1.0-Base.repo 配置优先级为 2：

```
[base]
name=base
baseurl=http://192.168.1.2/
enabled=1
priority=2
gpgcheck=1
gpgkey=http://192.168.1.2/RPM-GPG-KEY-openEuler
```

📖 说明

gpgcheck 可设置为 1 或 0，1 表示进行 gpg（GNU Private Guard）校验，0 表示不进行 gpg 校验，gpgcheck 可以确定 rpm 包的来源是有效和安全的。

gpgkey 为签名公钥的存放路径。

8.1.4.3 dnf 相关命令

dnf 命令在安装升级时能够自动解析包的依赖关系，一般的使用方式如下：

```
dnf <command> <packages name>
```

常用的命令如下：

- 安装

```
dnf install <packages name>
```

- 升级

```
dnf update <packages name>
```

- 回退

```
dnf downgrade <packages name>
```

- 检查更新

```
dnf check-update
```

- 卸载

```
dnf remove <packages name>
```

- 查询

```
dnf search <packages name>
```

- 本地安装

```
dnf localinstall <absolute path to package name>
```

- 查看历史记录

```
dnf history
```

- 清除缓存目录

```
dnf clean all
```

- 更新缓存

```
dnf makecache
```

8.2 搭建 FTP 服务器

8.2.1 总体介绍

FTP 简介

FTP（File Transfer Protocol）即文件传输协议，是互联网最早的传输协议之一，其最主要的功能是服务器和客户端之间的文件传输。FTP 使用户可以通过一套标准的命令访问远程系统上的文件，而不需要直接登录远程系统。另外，FTP 服务器还提供了如下主要功能：

- 用户分类

默认情况下，FTP 服务器依据登录情况，将用户分为实体用户（real user）、访客（guest）、匿名用户（anonymous）三类。三类用户对系统的访问权限差异较大，实体用户具有较完整的访问权限，匿名用户仅有下载资源的权限。

- 命令记录和日志文件记录

FTP 可以利用系统的 syslogd 记录数据，这些数据包括用户历史使用命令与用户传输数据（传输时间、文件大小等），用户可以在 /var/log/ 中获得各项日志信息。

- 限制用户的访问范围

FTP 可以将用户的工作范围限定在用户主目录。用户通过 FTP 登录后系统显示的根目录就是用户主目录，这种环境被称为 `change root`，简称 `chroot`。这种方式可以限制用户只能访问主目录，而不允许访问 `/etc`、`/home`、`/usr/local` 等系统的重要目录，从而保护系统，使系统更安全。

FTP 使用到的端口

FTP 的正常工作需要使用到多个网络端口，服务器端会使用到的端口主要有：

- 命令通道，默认端口为 21
- 数据通道，默认端口为 20

两者的连接发起端不同，端口 21 主要接收来自客户端的连接，端口 20 则是 FTP 服务器主动连接至客户端。

vsftpd 简介

由于 FTP 历史悠久，它采用未加密的传输方式，所以被认为是一种不安全的协议。为了更安全地使用 FTP，这里介绍 FTP 较为安全的守护进程 `vsftpd`（Very Secure FTP Daemon）。

之所以说 `vsftpd` 安全，是因为它最初的发展理念就是构建一个以安全为中心的 FTP 服务器。它具有如下特点：

- `vsftpd` 服务的启动身份为一般用户，具有较低的系统权限。此外，`vsftpd` 使用 `chroot` 改变根目录，不会误用系统工具。
- 任何需要较高执行权限的 `vsftpd` 命令均由一个特殊的上层程序控制，该上层程序的权限较低，以不影响系统本身为准。
- `vsftpd` 整合了大部分 FTP 会使用到的额外命令（例如 `dir`、`ls`、`cd` 等），一般不需要系统提供额外命令，对系统来说比较安全。

8.2.2 使用 vsftpd

安装 vsftpd

使用 `vsftpd` 需要安装 `vsftpd` 软件，在已经配置 `yum` 源的情况下，通过 `root` 权限执行如下命令，即可完成 `vsftpd` 的安装。

```
# dnf install vsftpd
```

管理 vsftpd 服务

启动、停止和重启 `vsftpd` 服务，请在 `root` 权限下执行对应命令。

- 启动 `vsftpd` 服务

```
# systemctl start vsftpd
```

可以通过 `netstat` 命令查看通信端口 21 是否开启，如下显示说明 `vsftpd` 已经启动。

```
# netstat -tulnp | grep 21
tcp6      0      0  :::21                :::*                   LISTEN
19716/vsftpd
```

说明

如果没有 netstat 命令，可以执行如下命令安装后再使用 netstat：

```
dnf install net-tools
```

- 停止 vsftpd 服务

```
# systemctl stop vsftpd
```

- 重启 vsftpd 服务

```
# systemctl restart vsftpd
```

8.2.3 配置 vsftpd

8.2.3.1 vsftpd 配置文件介绍

用户可以通过修改 vsftpd 的配置文件，控制用户权限等。vsftpd 的主要配置文件和含义如表 8-1 所示，用户可以根据需求修改配置文件的内容。更多的配置参数含义可以通过 man 查看。

表8-1 vsftpd 配置文件介绍

配置文件	含义
/etc/vsftpd/vsftpd.conf	vsftpd 进程的主配置文件，配置内容格式为“参数=参数值”，且参数和参数值不能为空。 vsftpd.conf 的详细介绍可以使用如下命令查看： man 5 vsftpd.conf
/etc/pam.d/vsftpd	PAM（Pluggable Authentication Modules）认证文件，主要用于身份认证和限制一些用户的操作。
/etc/vsftpd/ftpusers	禁用使用 vsftpd 的用户列表文件。默认情况下，系统帐号也在该文件中，因此系统帐号默认无法使用 vsftpd。
/etc/vsftpd/user_list	禁止或允许登录 vsftpd 服务器的用户列表文件。该文件是否生效，取决于主配置文件 vsftpd.conf 中的如下参数： userlist_enable: 是否启用 userlist 机制，YES 为启用，此时 userlist_deny 配置有效，NO 为禁用。 userlist_deny: 是否禁止 user_list 中的用户登录，YES 为禁止名单中的用户登录，NO 为允许命令中的用户登录。 例如 userlist_enable=YES, userlist_deny=NO, 则 user_list 中的用户都无法登录。
/etc/vsftpd/chroot_list	是否限制在主目录下的用户列表。该文件默认不存在，需要手动建立。它是主配置文件 vsftpd.conf 中参数 chroot_list_file 的参数值。 其作用是限制还是允许，取决于主配置文件 vsftpd.conf 中的如下参数： <ul style="list-style-type: none"> • chroot_local_user: 是否将所有用户限制在主目录，YES 为启用，NO 禁用。

配置文件	含义
	<ul style="list-style-type: none"> chroot_list_enable: 是否启用限制用户的名单, YES 为启用, NO 禁用。 例如 chroot_local_user=YES, chroot_list_enable=YES, 且指定 chroot_list_file=/etc/vsftpd/chroot_list 时, 表示所有用户被限制在其主目录下, 而 chroot_list 中的用户不受限制。
/usr/sbin/vsftpd	vsftpd 的唯一执行文件。
/var/ftp/	匿名用户登录的默认根目录, 与 ftp 帐户的用户主目录有关。

8.2.3.2 默认配置说明

📖 说明

文档中的配置内容仅供参考, 请用户根据实际情况 (例如安全加固需要) 进行修改。

openEuler 系统中, vsftpd 默认不开放匿名用户, 使用 vim 命令查看主配置文件, 其内容如下:

```
# vim /etc/vsftpd/vsftpd.conf
anonymous_enable=NO
local_enable=YES
write_enable=YES
local_umask=022
dirmessage_enable=YES
xferlog_enable=YES
connect_from_port_20=YES
xferlog_std_format=YES
listen=NO
listen_ipv6=YES
pam_service_name=vsftpd
userlist_enable=YES
```

其中各参数含义如表 8-2 所示。

表8-2 参数说明

参数	含义
anonymous_enable	是否允许匿名用户登录, YES 为允许匿名登录, NO 为不允许。
local_enable	是否允许本地用户登入, YES 为允许本地用户登入, NO 为不允许。
write_enable	是否允许登录用户有写权限, YES 为启用上传写入功能, NO 为禁用。
local_umask	本地用户新增档案时的 umask 值。
dirmessage_enable	当用户进入某个目录时, 是否显示该目录需要注意的内容, YES

参数	含义
	为显示注意内容，NO 为不显示。
xferlog_enable	是否记录使用者上传与下载文件的操作，YES 为记录操作，NO 为不记录。
connect_from_port_20	Port 模式进行数据传输是否使用端口 20，YES 为使用端口 20，NO 为不使用端口 20。
xferlog_std_format	传输日志文件是否以标准 xferlog 格式书写，YES 为使用该格式书写，NO 为不使用。
listen	设置 vsftpd 是否以 stand alone 的方式启动，YES 为使用 stand alone 方式启动，NO 为不使用该方式。
pam_service_name	支持 PAM 模块的管理，配置值为服务名称，例如 vsftpd。
userlist_enable	是否支持/etc/vsftpd/user_list 文件内的账号登录控制，YES 为支持，NO 为不支持。
tcp_wrappers	是否支持 TCP Wrappers 的防火墙机制，YES 为支持，NO 为不支持。
listen_ipv6	是否侦听 IPv6 的 FTP 请求，YES 为侦听，NO 为不侦听。listen 和 listen_ipv6 不能同时开启。

8.2.3.3 配置本地时间

概述

openEuler 系统中，vsftpd 默认使用 GMT 时间（格林尼治时间），可能和本地时间不一致，例如 GMT 时间比北京时间晚 8 小时，请用户改为本地时间，否则服务器和客户端时间不一致，在上传下载文件时可能引起错误。

设置方法

设置 vsftpd 时间为本地时间的操作步骤如下：

步骤 1 打开配置文件 vsftpd.conf，将参数 use_localtime 的参数值改为 YES。命令如下：

```
# vim /etc/vsftpd/vsftpd.conf
```

配置内容如下：

```
use_localtime=YES
```

步骤 2 重启 vsftpd 服务。

```
# systemctl restart vsftpd
```

步骤 3 设置 vsftpd 服务开机启动。

```
# systemctl enable vsftpd
```

----结束

8.2.3.4 配置欢迎信息

正常使用 vsftpd 服务，需要存在欢迎信息文件。设置 vsftp 的欢迎信息 welcome.txt 文件的操作步骤如下：

步骤 1 打开配置文件 vsftpd.conf，加入欢迎信息文件配置内容后保存退出。

```
# vim /etc/vsftpd/vsftpd.conf
```

需要加入的配置行如下：

```
banner_file=/etc/vsftpd/welcome.txt
```

步骤 2 建立欢迎信息。即打开 welcome.txt 文件，写入欢迎信息后保存退出。

```
# vim /etc/vsftpd/welcome.txt
```

欢迎信息举例如下：

```
Welcome to this FTP server!
```

----结束

8.2.3.5 配置系统帐号登录权限

一般情况下，用户需要限制部分帐号的登录权限。用户可根据需要进行配置。

限制系统帐号登录的文件有两个，默认如下：

- /etc/vsftpd/ftpusers：受/etc/pam.d/vsftpd 文件的设置影响，由 PAM 模块掌管。
- /etc/vsftpd/user_list：由 vsftpd.conf 的 userlist_file 设置，由 vsftpd 主动提供。

两个文件的必须同时存在且内容相同，请参考/etc/passwd 文件，将 UID 小于 500 的帐号写入这两个文件，每一行代表一个帐号。

如果用户需要限制系统帐号登录，需要将对应帐号添加到/etc/vsftpd/ftpusers 和 /etc/vsftpd/user_list。

打开 user_list 可以查看当前文件中包含的帐号信息，命令和回显如下：

```
# vim /etc/vsftpd/user_list
```

```
root  
bin  
daemon  
adm  
lp  
sync  
shutdown  
halt  
mail  
news  
uucp  
operator
```

```
games
nobody
```

8.2.4 验证 FTP 服务是否搭建成功

可以使用 openEuler 提供的 FTP 客户端进行验证。命令和回显如下，根据提示输入用户名（用户为系统中存在的用户）和密码。如果显示 Login successful，即说明 FTP 服务器搭建成功。

```
# dnf install ftp
# ftp localhost
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
220-Welcome to this FTP server!
220
Name (localhost:root): USERNAME
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bye
221 Goodbye.
```

8.2.5 配置防火墙

如果要将 FTP 开放给 Internet 使用，需要对防火墙和 SELinux 进行设置。

```
# firewall-cmd --add-service=ftp --permanent
success
# firewall-cmd --reload
success
# setsebool -P ftpd_full_access on
```

8.2.6 传输文件

概述

这里给出 vsftpd 服务启动后，如何进行文件传输的指导。

连接服务器

命令格式

```
ftp [hostname | ip-address]
```

其中 hostname 为服务器名称，ip-address 为服务器 IP 地址。

操作说明

在 openEuler 系统的命令行终端，执行如下命令：

```
ftp ip-address
```

根据提示输入用户名和密码，认证通过后显示如下，说明 ftp 连接成功，此时进入了连接到的服务器目录。

```
ftp>
```

在该提示符下，可以输入不同的命令进行相关操作：

- 显示服务器当前地址

```
ftp>pwd
```

- 显示本地路径，用户可以将该路径下的文件上传到 FTP 服务器对应位置

```
ftp>lcd
```

- 退出当前窗口，返回本地 Linux 终端

```
ftp>!
```

下载文件

通常使用 `get` 或 `mget` 命令下载文件。

get 使用方法

- 功能说明：将文件从远端主机中传送至本地主机中
- 命令格式：**get** [*remote-file*] [*local-file*]
其中 *remote-file* 为远程文件，*local-file* 为本地文件
- 示例：获取远程服务器上的 `/usr/your/openEuler.htm` 文件，命令如下：

```
ftp> get /usr/your/openEuler.htm
```

mget 使用方法

- 功能说明：从远端主机接收一批文件至本地文件
- 命令格式：**mget** [*remote-file*]
其中 *remote-file* 为远程文件
- 示例：获取服务器上 `/usr/your/` 目录下的所有文件，命令如下：

```
ftp> cd /usr/your/  
ftp> mget *.*
```

说明

- 此时每下载一个文件，都会有提示信息。如果要屏蔽提示信息，则在 **mget *.*** 命令前先执行 **prompt off**
- 文件都被下载到 Linux 主机的当前目录下。比如，在 `/usr/my/` 下运行的 `ftp` 命令，则文件都下载到 `/usr/my/` 下。

上传文件

通常使用 `put` 或 `mput` 命令上传文件。

put 使用方法

- 功能说明：将本地的一个文件传送到远端主机中
- 命令格式：**put** [*local-file*] [*remote-file*]
其中 *remote-file* 为远程文件，*local-file* 为本地文件

- 示例：将本地的 Euler.htm 传送到远端主机/usr/your/，并改名为 openEuler.htm，命令如下：

```
ftp> put Euler.htm /usr/your/openEuler.htm
```

mput 使用方法

- 功能说明：将本地主机中一批文件传送到远端主机
- 命令格式：**mput** [local-file]
其中 local-file 为本地文件
- 示例：将本地当前目录下所有 htm 文件上传到服务器/usr/your/下，命令如下：

```
ftp> cd /usr/your  
ftp> mput *.htm
```

删除文件

通常使用 delete 或 mdelete 命令删除文件。

delete 使用方法

- 功能说明：删除远程服务器上的一个或多个文件
- 命令格式：**delete** [remote-file]
其中 remote-file 为远程文件
- 示例：删除远程服务器上的 openFile 文件，命令如下：

```
ftp> delete openFile
```

mdelete 使用方法

- 功能说明：删除远程服务器上的文件，常用于批量删除
- 命令格式：**mdelete** [remote-file]
其中 remote-file 为远程文件
- 示例：删除所有 a 开头的文件，命令如下：

```
ftp> mdelete a*
```

断开服务器

断开与服务器的连接，使用 bye 命令，如下：

```
ftp> bye
```

8.3 搭建 web 服务器

8.3.1 概述

Web（World Wide Web）是目前最常用的 Internet 协议之一。目前在 Unix-Like 系统中的 web 服务器主要通过 Apache 服务器软件实现。为了实现运营动态网站，产生了 LAMP（Linux + Apache + MySQL + PHP）。web 服务可以结合文字、图形、影像以及声音等多媒体，并支持超链接（Hyperlink）的方式传输信息。

openEuler 系统中的 web 服务器版本是 Apache HTTP 服务器 2.4 版本，即 httpd，一个由 Apache 软件基金会发展而来的开源 web 服务器。

8.3.2 管理 httpd

概述

通过 `systemctl` 工具，可以对 httpd 服务进行管理，包括启动、停止、重启服务，以及查看服务状态等。本章介绍 Apache HTTP 服务的管理操作，以指导用户使用。

前提条件

- 为了能够使用 Apache HTTP 服务，请确保您的系统中已经安装 httpd 服务的 rpm 包。安装命令如下：

```
# dnf install httpd
```

更多关于管理服务的内容，请参见“管理服务”章节。

- 启动、停止和重启 httpd 服务，需要使用 root 权限。

启动服务

- 启动并运行 httpd 服务，命令如下：

```
# systemctl start httpd
```

- 假如希望在系统启动时，httpd 服务自动启动，则命令和回显如下：

```
# systemctl enable httpd
```

```
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service →  
/usr/lib/systemd/system/httpd.service.
```

说明

假如正在运行的 Apache HTTP 服务器作为一个安全服务器，系统开机启动后需要密码，这个密码使用的是加密的私有 SSL 密钥。

停止服务

- 停止运行的 httpd 服务，命令如下：

```
# systemctl stop httpd
```

- 如果希望防止服务在系统开机阶段自动开启，命令和回显如下：

```
# systemctl disable httpd
```

```
Removed /etc/systemd/system/multi-user.target.wants/httpd.service.
```

重启服务

重启服务有三种方式：

- 完全重启服务

```
# systemctl restart httpd
```

该命令会停止运行的 httpd 服务并且立即重新启动它。一般在服务安装以后或者去除一个动态加载的模块（例如 PHP）时使用这个命令。

- 重新加载配置

```
# systemctl reload httpd
```

该命令会使运行的 httpd 服务重新加载它的配置文件。任何当前正在处理的请求将会被中断，从而造成客户端浏览器显示一个错误消息或者重新渲染部分页面。

- 重新加载配置而不影响激活的请求

```
# apachectl graceful
```

该命令会使运行的 httpd 服务重新加载它的配置文件。任何当前正在处理的请求将会继续使用旧的配置文件。

验证服务状态

验证 httpd 服务是否正在运行

```
# systemctl is-active httpd
```

回显为“active”说明服务处于运行状态。

8.3.3 配置文件说明

当 httpd 服启动后，默认情况下它会读取如表 8-3 所示的配置文件。

表8-3 配置文件说明

文件	说明
/etc/httpd/conf/httpd.conf	主要的配置文件
/etc/httpd/conf.d	配置文件的辅助目录，这些配置文件也被包含在主配置文件当中 一个配置文件的辅助目录被包含在主要的配置文件中

虽然默认配置可以适用于多数情况，但是用户至少需要熟悉里面的一些重要配置项。配置文件修改完成后，可以使用如下命令检查配置文件可能出现的语法错误。

```
# apachectl configtest
```

如果回显如下，说明配置文件语法正确。

```
Syntax OK
```

说明

- 在修改配置文件之前，请先备份原始文件，以便出现问题时能够快速恢复配置文件。
- 需要重启 web 服务，才能使修改后的配置文件生效。

8.3.4 管理模块和 SSL

概述

httpd 服务是一个模块化的应用，它和许多动态共享对象 DSO（Dynamic Shared Objects）一起分发。动态共享对象 DSO，在必要情况下，可以在运行时被动态加载或

卸载。服务器操作系统中这些模块位于 `/usr/lib64/httpd/modules/` 目录下。本节介绍如何加载和写入模块。

加载模块

为了加载一个特殊的 DSO 模块，在配置文件中使用加载模块指示。独立软件包提供的模块一般在 `/etc/httpd/conf.modules.d` 目录下有他们自己的配置文件。

例如，加载 `asis` DSO 模块的操作步骤如下：

步骤 1 在 `/etc/httpd/conf.modules.d/00-optional.conf` 文件中，取消注释如下配置行。

```
LoadModule asis_module modules/mod_asis.so
```

步骤 2 加载完成后，请重启 `httpd` 服务以便于重新加载配置文件。

```
# systemctl restart httpd
```

步骤 3 加载完成后，使用 `httpd -M` 的命令查看是否已经加载了 `asis` DSO 模块。

```
# httpd -M | grep asis
```

回显如下，说明 `asis` DSO 模块加载成功。

```
asis_module (shared)
```

----结束

📖 说明

`httpd` 的常用命令

- `httpd -v`：查看 `httpd` 的版本号。
- `httpd -l`：查看编译进 `httpd` 程序的静态模块。
- `httpd -M`：查看已经编译进 `httpd` 程序的静态模块和已经加载的动态模块。

SSL 介绍

安全套接层 SSL (Secure Sockets Layer) 是一个允许服务端和客户端之间进行安全通信的加密协议。其中，传输层安全性协议 TLS (Transport Layer Security) 为网络通信提供了安全性和数据完整性保障。openEuler 支持 Mozilla NSS (Network Security Services) 作为安全性协议 TLS 进行配置。加载 SSL 的操作步骤如下：

步骤 1 安装 `mod_ssl` 的 rpm 包。

```
# dnf install mod_ssl
```

步骤 2 安装完成后，请重启 `httpd` 服务以便于重新加载配置文件。

```
# systemctl restart httpd
```

步骤 3 加载完成后，使用 `httpd -M` 的命令查看是否已经加载了 `SSL`。

```
# httpd -M | grep ssl
```

回显如下，说明 `SSL` 已加载成功。

```
ssl_module (shared)
```

----结束

8.3.5 验证 web 服务是否搭建成功

Web 服务器搭建完成后，可以通过如下方式验证是否搭建成功。

步骤 1 查看服务器的 IP 地址，命令如下：

```
# ifconfig
```

回显信息如下，说明服务器 IP 为 192.168.1.60。

```
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.1.60 netmask 255.255.255.0 broadcast 192.168.1.255
inet6 fe80::5054:ff:fe95:499f prefixlen 64 scopeid 0x20<link>
ether 52:54:00:95:49:9f txqueuelen 1000 (Ethernet)
RX packets 150713207 bytes 49333673733 (45.9 GiB)
RX errors 0 dropped 43 overruns 0 frame 0
TX packets 2246438 bytes 203186675 (193.7 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp4s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
ether 52:54:00:7d:80:9e txqueuelen 1000 (Ethernet)
RX packets 149937274 bytes 44652889185 (41.5 GiB)
RX errors 0 dropped 1102561 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 37096 bytes 3447369 (3.2 MiB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 37096 bytes 3447369 (3.2 MiB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

步骤 2 配置防火墙：

```
# firewall-cmd --add-service=http --permanent
success
# firewall-cmd --reload
success
```

步骤 3 验证 web 服务器是否搭建成功，用户可选择 Linux 或 Windows 系统进行验证。

- 使用 Linux 系统验证

执行如下命令，查看是否可以访问网页信息，服务搭建成功时，该网页可以正常访问。

```
curl https://192.168.1.60
```

执行如下命令，查看命令返回值是否为 0，返回值为 0，说明 httpd 服务器搭建成功。

```
echo $?
```

- 使用 Windows 系统验证

打开浏览器，在地址栏输入如下地址，如果能正常访问网页，说明 httpd 服务器搭建成功。

`https://192.168.1.60`

如果修改了端口号，输入地址格式如下：

`https://192.168.1.60:端口号`

----结束

9 FAQ

- 9.1 使用 `systemctl` 和 `top` 命令查询 `libvirtd` 服务占用内存不同
- 9.2 设置 RAID0 卷，参数 `stripsize` 设置为 4 时出错
- 9.3 使用 `rpmbuild` 编译 `mariadb` 失败

9.1 使用 `systemctl` 和 `top` 命令查询 `libvirtd` 服务占用内存不同

问题描述

使用 `systemctl` 和 `systemd-cgtop` 命令查询 `libvirtd` 服务占用内存超 1.5G，而使用 `top` 命令查询 `libvirtd` 服务占用内存仅 70M 左右。

原因分析

`systemd` 管理的服务（包括 `systemctl` 和 `systemd-cgtop`）中显示的内存通过查询 CGroup 对应的 `memory.usage_in_bytes` 得到。`top` 是直接统计 `/proc` 下内存相关信息计算得出。两者的统计方法不同，不能直接比较。

一般来说，业务进程使用的内存主要有以下几种情况：

- `anon_rss`: 用户空间的匿名映射页（Anonymous pages in User Mode address spaces），比如调用 `malloc` 分配的内存，以及使用 `MAP_ANONYMOUS` 的 `mmap`。当系统内存不够时，内核可以将这部分内存交换出去。
- `file_rss`: 用户空间的文件映射页（Mapped pages in User Mode address spaces），包含 `map file` 和 `map tmpfs`，前者比如指定文件的 `mmap`，后者比如 IPC 共享内存。当系统内存不够时，内核可以回收这些页，但回收之前可能需要与文件同步数据。
- `file_cache`: 文件缓存（page in page cache of disk file），发生在程序通过普通的读写（read/write）文件时。当系统内存不够时，内核可以回收这些页，但回收之前可能需要与文件同步数据。
- `buffer pages`: 属于 `page cache`，比如读取块设备文件时的相关缓存。

其中 `anon_rss` 和 `file_rss` 属于进程的 RSS，`file_cache` 和 `buffer pages` 属于 page cache。
简单来说：

`top` 里的 `RSS = anon_rss + file_rss`，`SHR = file_rss`。

CGroup 里的 `memory.usage_in_bytes = cache + RSS + swap`。

由上可知，`systemd` 相关命令和 `top` 命令的内存占用率含义不同，所以查询结果不同。

9.2 设置 RAID0 卷，参数 `stripsize` 设置为 4 时出错

问题现象

设置 RAID0 卷，参数 `stripsize` 设置为 4 时出错。

原因分析

64K 页表开启只能支持 64K 场景。

解决方法

不需要修改配置文件，`openeuler` 执行 `lvcreate` 命令时，条带化规格支持的 `stripsize` 最小值为 64KB，将参数 `stripsize` 设置为 64。

9.3 使用 `rpmbuild` 编译 `mariadb` 失败

问题描述

如果使用 `root` 账号登录系统，并在该账号下使用 `rpmbuild` 命令编译 `mariadb` 源代码，会出现编译失败现象，提示：

```
+ echo 'mysql can'\''t run test as root'
mysql can't run test as root
+ exit 1
```

原因分析

`mariadb` 数据库不允许使用 `root` 权限的账号进行测试用例执行，所以会阻止编译过程（编译过程中会自动执行测试用例）。

解决方案

使用 `vi` 等文本编辑工具，修改 `mariadb.spec` 文件中 `runtest` 变量的值。

修改前：

```
%global runtest 1
```

修改后：

```
%global runtest 0
```

该修改关闭了编译阶段执行测试用例的功能，但不会影响编译和编译后的 RPM 包内容。