



中标软件

中标麒麟高级服务器操作系统 V7

系统管理员手册

中标软件有限公司

2016. 09. 23

目录

1 基本系统配置.....	1
1.1 系统地区和键盘配置.....	1
1.1.1 配置系统地区.....	1
1.1.2 配置键盘布局.....	3
1.1.3 其他资源.....	4
1.2 日期和时间配置.....	4
1.2.1 Timedatectl 工具使用说明.....	4
1.2.2 Date 工具使用说明.....	7
1.2.3 hwclock 工具使用说明.....	8
1.3 用户和组群配置.....	9
1.3.1 添加新用户.....	9
1.3.2 修改用户属性.....	10
1.3.3 命令行配置.....	11
1.3.4 添加用户的详细过程.....	12
1.4 获取特权.....	14
1.4.1 Su 命令工具.....	14
1.4.2 Sudo 命令工具.....	14
1.5 License 注册与管理.....	15
1.5.1 字符授权管理工具.....	15
1.5.2 图形授权管理工具.....	18
2 安装和管理软件.....	26
2.1 NKUC-客户端套件.....	26
2.1.1 图形工具注册.....	26
2.1.2 命令行注册.....	38
2.1.3 软件包升级.....	39
2.2 yum.....	41
2.2.1 检查和升级软件包.....	41
2.2.2 管理软件包.....	46
2.2.3 管理软件包组.....	54
2.2.4 软件包操作记录管理.....	57
2.2.5 配置 yum 和 yum 仓库.....	62
2.2.6 Yum 插件.....	69

3 基础服务.....	71
3.1 使用 systemd 管理系统服务	71
3.1.1 Systemd 介绍.....	71
3.1.2 管理系统服务.....	74
3.1.3 管理目标.....	81
3.1.4 关闭暂停挂起系统.....	85
3.1.5 在远程机器上使用 systemd	87
3.1.6 创建和修改 systemd 单元文件	87
3.2 OpenSSH.....	108
3.2.1 SSH 协议	108
3.2.2 配置 OpenSSH.....	112
3.2.3 不只是一个安全的 Shell	127
3.3 TigerVNC.....	129
3.3.1 VNC 服务端	130
3.3.2 共享一个已存在的桌面.....	133
3.3.3 VNC 查看器	133
4 服务器.....	139
4.1 Web 服务器	139
4.1.1 Apache HTTP 服务器.....	139
4.2 邮件服务器.....	169
4.2.1 电子邮件协议.....	169
4.2.2 电子邮件软件分类.....	173
4.2.3 邮件传输代理.....	174
4.2.4 邮件投递代理.....	177
4.2.5 邮件用户代理.....	185
4.3 目录服务器.....	187
4.3.1 OpenLDAP	187
4.3.2 安装 OpenLDAP 组件	190
4.3.3 配置 OpenLDAP 服务器	192
4.3.4 使用 LDAP 应用的 SELinux 策略.....	204
4.3.5 运行 OpenLDAP 服务	205
4.3.6 配置系统使用 OpenLDAP 作为验证	206
4.4 文件和打印服务器.....	207
4.4.1 Samba.....	208

4.4.2 FTP.....	225
4.4.3 打印设置.....	230
4.5 使用 chrony 套件配置 NTP.....	247
4.5.1 chrony 套件介绍	248
4.5.2 理解 CHRONY 及其配置.....	249
4.5.3 使用 chrony	254
4.5.4 为不同的环境设置 chrony	259
4.5.5 使用 chronyc.....	261
4.6 配置 NTP 使用 NTPD.....	262
4.6.1 NTP 介绍	262
4.6.2 NTP 分层	263
4.6.3 理解 NTP.....	263
4.6.4 理解 drift 文件.....	263
4.6.5 UTC, TIMEZONES 和 DST	264
4.6.6 NTP 身份验证选项.....	264
4.6.7 在虚拟机中管理时间.....	264
4.6.8 理解闰秒.....	264
4.6.9 理解 ntpd 配置文件	264
4.6.10 理解 ntpd 的 sysconfig 文件	266
4.6.11 禁止 chrony.....	266
4.6.12 检查 NTP 守护进程是否安装.....	267
4.6.13 ntpd 的安装	267
4.6.14 检查 ntp 的状态	267
4.6.15 配置防火墙允许 ntp 包进入	268
4.6.16 配置 ntpdate 服务器.....	269
4.6.17 配置 ntp	269
4.6.18 配置硬件时钟更新.....	275
4.6.19 配置时钟源.....	275
4.7 使用 ptp4l 配置 PTP	276
4.7.1 PTP 介绍.....	276
4.7.2 使用 PTP.....	277
4.7.3 和多个接口使用 PTP.....	281
4.7.4 指定一个配置文件.....	281
4.7.5 使用 PTP 管理客户端.....	282

4.7.6 同步时钟.....	283
4.7.7 验证时间同步.....	284
4.7.8 使用 NTP 服务 PTP 时间.....	288
4.7.9 使用 PTP 服务 NTP 时间.....	289
4.7.10 使用 timemaster 同步 PTP 或 NTP 时间.....	289
4.7.11 提高准确性.....	293
5 监控和自动化.....	294
5.1 系统监控工具.....	294
5.1.1 查看系统进程.....	294
5.1.2 查看内存使用情况.....	298
5.1.3 查看 CPU 使用.....	299
5.1.4 查看块设备和文件系统.....	300
5.1.5 查看硬件信息.....	306
5.1.6 检查硬件错误.....	310
5.1.7 使用 Net-SNMP 监控性能.....	311
5.2 OpenLMI.....	327
5.2.1 关于 OpenLMI.....	328
5.2.2 安装 OpenLMI.....	329
5.2.3 为 OpenPegasus 配置 SSL 证书.....	331
5.2.4 使用 LMI Shell.....	336
5.2.5 使用 OpenLMI 脚本.....	380
5.3 查看和管理日志文件.....	380
5.3.1 日志文件的位置.....	381
5.3.2 Rsyslog 的基本配置.....	381
5.3.3 使用新的配置格式.....	397
5.3.4 使用 Rsyslog 队列.....	399
5.3.5 在日志服务器上配置 rsyslog.....	410
5.3.6 使用 Rsyslog 模块.....	415
5.3.7 Syslogd 服务和日志的交互.....	419
5.4 Syslogd 日志结构.....	420
5.4.1 从日志中导入数据.....	421
5.4.2 过滤结构化消息.....	422
5.4.3 解析 JSON.....	423
5.4.4 向 MongoDB 中存储消息.....	423

5.5 调试 Rsyslog.....	424
5.6 使用日志.....	424
5.6.1 查看日志文件.....	425
5.6.2 访问控制.....	426
5.6.3 使用 Live view	427
5.6.4 过滤消息.....	427
5.6.5 使能持续存储.....	430
5.7 在图形界面管理日志.....	430
5.7.1 查看日志文件.....	430
5.7.2 添加日志文件.....	433
5.7.3 监控日志文件.....	434
5.8 自动化系统任务.....	435
5.8.1 Cron 和 Anacron.....	435
5.8.2 安装 Cron 和 Anacron.....	436
5.8.3 运行 Crond 服务.....	436
5.8.4 配置 Anacron 任务.....	437
5.8.5 配置 Cron 任务.....	440
5.8.6 控制对 Cron 的访问.....	442
5.8.7 Cron 任务的黑白名单.....	443
5.8.8 At 和 Batch	443
5.9 自动程序错误报告工具 (ABRT)	448
5.9.1 ABRT 简介	448
5.9.2 安装 ABRT 并启动服务	449
5.9.3 配置 ABRT	451
5.9.4 检测软件问题.....	459
5.9.5 处理检测到的问题.....	462
5.10 Oprofile	465
5.10.1 工具概览.....	466
5.10.2 使用 operf	468
5.10.3 使用遗留模式配置 OProfile.....	471
5.10.4 启动和停止 OProfile 遗留模式.....	478
5.10.5 在遗留模式下保存数据.....	479
5.10.6 分析数据.....	479
5.10.7 理解/dev/oprofile/目录.....	487

5.10.8 OProfile 对 Java 的支持.....	488
5.10.9 图形界面.....	489
5.10.10 Oprofile 和 SystemTap	491
6 kernel、module 和驱动配置.....	493
6.1 使用 GRUB2 启动加载工具	493
6.1.1 GRUB2 简介	493
6.1.2 配置 GRUB2 引导加载工具	495
6.1.3 临时修改 GRUB2 选项	495
6.1.4 用 grubby 工具永久修改 GRUB2 选项	496
6.1.5 自定义 GRUB2 配置文件	498
6.1.6 GRUB2 密码保护	504
6.1.7 重新安装 GRUB2	508
6.1.8 GRUB2 配置串口控制台	509
6.1.9 开机时的终端菜单编辑.....	511
6.1.10 安全启动统一可扩展固件接口（UEFI）	517
6.2 手动升级内核.....	518
6.2.1 内核软件包概述.....	519
6.2.2 升级准备工作.....	520
6.2.3 下载升级内核.....	522
6.2.4 准备升级内核.....	522
6.2.5 验证初始 RAM 磁盘镜像	522
6.2.6 验证引导加载工具.....	527
6.3 内核模块.....	527
6.3.1 查看当前加载的模块.....	528
6.3.2 查看模块信息.....	529
6.3.3 加载模块.....	535
6.3.4 卸载内核模块.....	536
6.3.5 设置模块参数.....	537
6.3.6 自动加载模块.....	538
6.3.7 从驱动更新磁盘安装模块.....	538
6.3.8 数字签名内核模块.....	542
7 附录 RPM.....	550
7.1 RPM 设计目标	551
7.2 使用 RPM.....	552

7.2.1 安装和升级软件包.....	553
7.2.2 卸载软件包.....	556
7.2.3 更新软件包.....	557
7.2.4 查询软件包.....	557
7.2.5 校验软件包.....	558
7.3 查找并校验 RPM 软件包	559
7.3.1 查找 RPM 软件包	559
7.3.2 校验软件包签名.....	560
7.4 RPM 用法的常用示例	561

1 基本系统配置

这部分涵盖了基本的系统管理任务,如键盘配置、日期和时间配置、用户和组群配置以及授权配置。

1.1 系统地区和键盘配置

系统地区配置是指系统服务和用户界面的语言环境配置。键盘布局配置是指文本控制台和图形用户界面的键盘布局规则。这些设置可以通过修改 `/etc/locale.conf` 配置文件或使用 `localectl` 命令。此外,您可以在用户图形界面来执行任务,详情请参考安装手册。

1.1.1 配置系统地区

系统地区配置文件为 `/etc/locale.conf`, 在系统启动时引导 `systemd` 守护进程。这个配置文件可以被每一个服务或者用户继承,单个服务或者用户也可修改配置文件。例如语言为英语,地区为德国的 `/etc/locale` 文件的配置内容如下,:

```
LANG=de_DE.UTF-8
LC_MESSAGES=C
```

`LC_MESSAGES` 选项决定了诊断消息的标准输出文本格式。其他选项说明总结在表格 1-1 在所示。

表格 1-1 在 `/etc/locale.conf` 文件中可配置项

配置项	描述
<code>LANG</code>	提供系统时区的默认值
<code>LC_COLLATE</code>	定义该环境的排序和比较规则
<code>LC_CTYPE</code>	用于字符分类和字符串处理,控制所有字符的处理方式,包括字符编码,字符是单字节还是多字节,如何打印等。是最最重要的一个环境变量。
<code>LC_NUMERIC</code>	非货币的数字显示格式
<code>LC_TIME</code>	时间和日期格式

LC_MESSAGES

提示信息语言。

1.1.1.1 显示当前配置

Localectl 命令可用于配置语言环境和键盘布局。显示当前配置，可使用如下命令，并输出示例：

```
# localectl status

System Locale: LANG=en_US.UTF-8
VC Keymap: us
X11 Layout: n/a
```

1.1.1.2 显示可用地区列表

显示可用地区列表可使用如下命令，并输出示例：

```
# localectl list-locales | grep en_

en_AG
en_AG.utf8
en_AU
en_AU.iso88591
en_AU.utf8
en_BW
en_BW.iso88591
en_BW.utf8
output truncated
```

1.1.1.3 配置地区

配置系统默认地区，需要以 root 用户身份运行：

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

用户可以配置适合的地区标示符以代替 locale，可通过 localectl list-locales 检索适合的地区。

1.1.2 配置键盘布局

键盘布局配置是指文本控制台和图形用户界面的键盘布局规则。

1.1.2.1 显示当前配置

Localectl 命令可用于配置语言环境和键盘布局。显示当前配置，可使用如下命令，并输出示例：

```
# localectl status

System Locale: LANG=en_US. UTF- 8

VC Keymap: us

X11 Layout: n/a
```

1.1.2.2 显示可用键盘布局列表

显示可用地区列表可使用如下命令，并输出示例：

```
# localectl list-keymaps | grep cz

cz

cz- cp1250

cz- lat2

cz- lat2- prog

cz- qwerty

cz- us- qwertz

sunt5- cz- us

sunt5- us- cz
```

1.1.2.3 配置键盘

配置系统默认键盘布局，需要以 root 用户身份运行：

```
# localectl set-keymap map
```

用户可以配置适合的键盘布局标示符以代替 *map*，可通过 *localectl list-keymaps* 检索适合的键盘布局。该命令还可用于配置 X11 窗口的键盘布局映射，但使用 *--no-convert* 参数的话则不生效。同样也可用一下命令单独配置 X11 窗口的键盘布局：

```
# localectl set-x11-keymap map
```

如果用户希望 X11 窗口和命令行终端的键盘布局不一样，可以使用如下命令：

```
# localectl --no-convert set- x11-keymap map
```

1.1.3 其他资源

其他官方配置系统地区和键盘布局的内容可以参考安装手册。同时还可参考 1.4 获取特权章节和 3.1 章节。

1.2 日期和时间配置

操作系统区分以下两种时区：

- 实时时间（RTC），通常作为物理时钟，它可以独立于系统当前状态计时，在主机关机情况下也可计时。
- 系统时间，是基于实时时间的由操作系统内核维护的软件时间。等系统启动内核初始化系统时间后，系统时间就独立于实时时间自行计时。

系统时间通常还保持一套世界统一时间(UTC)，用于转换系统的不同时区，本地时间就是用户所在时区的真实时间。

操作系统提供了三种命令行时间管理工具，timedatectl、date 和 hwclock。以下将分别介绍各个工具的使用。

1.2.1 Timedatectl 工具使用说明

1.2.1.1 显示当前日期和时间

命令 timedatectl 可以显示当前系统时间和机器的物理时间及其详细信息。如下示例显示的是未启用 NTP 时钟同步的系统时间：

```
# timedatectl

Local time: Mon 2013- 09- 16 19: 30: 24 CEST

Universal time: Mon 2013- 09- 16 17: 30: 24 UTC

Timezone: Europe/Prague ( CEST, +0200)

NTP enabled: no
```

```
NTP synchronized: no
RTC in local TZ: no
DST active: yes
Last DST change: DST began at
Sun 2013- 03- 31 01: 59: 59 CET
Sun 2013- 03- 31 03: 00: 00 CEST
Next DST change: DST ends ( the clock j umps one hour backwards) at
Sun 2013- 10- 27 02: 59: 59 CEST
Sun 2013- 10- 27 02: 00: 00 CET
```

变更 `chrony` 或 `ntpd` 服务状态不会主动通知 `timedatectl` 工具，如果想要更新服务的配置信息，请执行一下命令：

```
# systemctl restart systemd – timedated.services
```

1.2.1.2 变更当前时间

以 `root` 用户运行以下命令可以修改当前时间：

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

其中 `HH` 代表小时，`MM` 代表分钟，`SS` 代表秒数，均需两位表示。这个命令同样可以更新系统时间和物理时间，效果类似于 `date –set` 和 `hwclock –systohc` 命令。

系统默认时间配置基于 `UTC`，如果想基于本地时间来配置系统时间，需要以 `root` 用户运行以下命令修改。

```
# timedatectl set-local-rtc boolean
```

如果基于本地时间，需要将 `boolean` 配置为 `yes`（或者 `y`，`true`，`t` 或者 `1`）。如果使用 `UTC` 时间，则要将 `boolean` 配置为 `no`（或者 `n`，`false`，`f` 或者 `0`）。系统默认 `boolean` 为 `no`。

1.2.1.3 变更当前日期

以 `root` 用户运行以下命令可以修改当前日期：

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

其中 YYYY 代表年份，需 4 位数表示；MM 代表月份，需两位数表示；DD 代表日期，需两位表示。如果还需要配置时间，可以补充上时间参数，示例如下：

```
# timedatectl set- time ' 2015- 06-02 23:26:00'
```

1.2.1.4 修改时区

执行以下命令可以显示当前时区：

```
# timedatectl list-timezones
```

以 root 用户运行以下命令可以修改当前时区：

```
# timedatectl set-timezone time_zone
```

修改时区示例如下：

```
# timedatectl list-timezones | grep Europe
Europe/Amsterdam
Europe/Andorra
Europe/Athens
Europe/Belgrade
Europe/Berlin
Europe/Bratislava
...
# timedatectl set- timezone Europe/Prague
```

1.2.1.5 同步系统与远程服务器时间

以 root 用户运行以下命令可以启用/禁用时间同步服务：

```
# timedatectl set-ntp boolean
```

启用与禁用需要配置 boolean 值为 yes 或者 no。例如需要自动同步一个远程时间服务器，可以执行一下命令：

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

1.2.2 Date 工具使用说明

1.2.2.1 显示当前日期和时间

命令 **date** 可以显示当前系统时间、时区、日期等信息。并可以通过参数 **--utc** 显示当前时区时间。通过 “**format**” 标示符来输出特定状态。常用的 **format** 说明如下：

表格 1-2 参数介绍

参数	描述
%H	以 HH 格式输出当前小时
%M	以 MM 格式输出当前分钟
%S	以 SS 格式输出当前秒数
%d	以 DD 格式输出当前日期
%m	以 MM 格式输出当前月份
%Y	以 YYYY 格式输出当前年份
%Z	显示时区制式，例如 C EST
%F	以 YYYY-MM-DD 格式输出当前年月日，等价于参数 %Y- %m- %d
%T	以 HH:MM:SS 格式输出当前时间，等价于参数 %H: %M: %S

示例如下：

```
# date
Mon Sep 16 17: 30: 24 CEST 2013

# date -utc
Mon Sep 16 15: 30: 34 UTC 2013

# date + "%Y-%m-%d%H%M"
2013- 09- 16 17: 30
```

1.2.2.2 变更当前时间

以 **root** 用户运行以下命令可以修改当前时间：

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

其中 HH 代表小时，MM 代表分钟，SS 代表秒数，均需两位表示。这个命令同样可以更新系统时间和物理时间，效果类似于 `hwclock --systohc` 命令。

系统默认时间配置基于本地时间，如果想基于 UTC 时间来配置系统时间，需要以 root 用户运行以下命令修改。

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

1.2.2.3 变更当前日期

以 root 用户运行以下命令可以修改当前日期：

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

其中 YYYY 代表年份，需 4 位数表示；MM 代表月份，需两位数表示；DD 代表日期，需两位表示。如果还需要配置时间，可以补充上时间参数，示例如下：

```
# date --set 2013-06 - 02 23:26 :00
```

1.2.3 hwclock 工具使用说明

1.2.3.1 显示当前日期和时间

命令 `hwclock` 可以显示当前系统时间、时区、日期等信息。并可以通过参数 `--utc` 或 `--localtime` 显示当前 UTC 时区时间和本地时间。示例如下：

```
# hwclock
```

```
Tue 15 Apr 2014 04: 23: 46 PM CEST - 0. 329272 seconds
```

1.2.3.2 变更当前日期和时间

以 root 用户运行以下命令可以修改当前时间：

```
#hwclock - -set - -date "dd mmm yyyy HH: MM"
```

其中 dd 代表日期 HH 代表小时，MM 代表分钟，SS 代表秒数，均需两位表示。Mmm 代表月份，以月份英文三位字母简写表示，yyyy 代表年份，以四位数字表示。这个命令通过参数 `--utc` 或 `--localtime` 区分配置当前 UTC 时区时间和本地时间

基于 UTC 时间来配置系统时间，需要以 root 用户运行以下命令修改，示例如下。

```
# hwclock --set --date "21 Oct 2014 21: 17" --utc
```

1.2.3.3 同步系统与远程服务器时间

以 root 用户运行以下命令同步远程时间：

```
# hwclock - - systohc
```

1.3 用户和组群配置

用户管理者允许您查看、修改、添加和删除本地用户和组群。

要使用用户管理者，您必须具备 root 特权。要从桌面启动用户管理器，点击面板上的【应用程序】→【系统工具】→【设置】→【用户】，弹出用户配置对话框。



图 1-1 用户管理者

要查看包括系统内本地用户的明显，点击【用户】标签。在具体的用户标签页可以编辑各项参数。

通过点击【+】【-】可以添加删除用户。

1.3.1 添加新用户

要添加新用户，点击【+】按钮。一个如图 1-2 所示的窗口就会出现。在适

当的字段内键入新用户的用户名和完整姓名。在【密码】和【验证】字段内键入密码。密码必须至少包含八个字符。

【账户类型】可以配置用户管理员权限和普通用户权限。【用户名】会配置用户主目录，默认的主目录是 `home/用户名/`。您可以改变为用户创建的主目录。

点击【添加】来创建该用户。



The image shows a 'Add User' dialog box with the following fields and options:

- Buttons:** '取消' (Cancel), '添加用户' (Add User), and '添加(A)' (Add).
- Account Type (T):** A dropdown menu currently set to '标准' (Standard).
- Full Name (F):** A text input field.
- Username (U):** A text input field with a dropdown arrow on the right. Below it, a note states: '通常是您主文件夹的名字，该名字不可改变。' (Usually the name of your main folder, this name cannot be changed).
- Password Section:**
 - 密码 (P):** A text input field with a gear icon on the right. Below it, a note states: '混合使用大写和小些字母及 1 到 2 个数字。' (Mix uppercase and lowercase letters and 1 to 2 digits).
 - Options:**
 - ☒ 允许用户下次登录时更改密码 (Allow user to change password next login)
 - ☐ 现在设置密码 (Set password now)
- Verification (V):** A text input field.
- Enterprise Login (E):** A button at the bottom left.

图 1-2 创建新用户

1.3.2 修改用户属性

要查看某个现存用户的属性，点击【用户】标签，从用户列表中选择该用户，右侧可现实用户的详细信息，并可在右侧编辑这些属性。

【账户类型】区分标准用户还是管理员用户。

【语言】用户登录默认语言选项。

【密码】用户登录密码，单击即可进行修改配置。

【最近登录】选显示该用户最近一次登录系统的时间。

1.3.3 命令行配置

如果您更喜欢使用命令行工具，请参考本节来配置用户和组群。

表格 1-3 常用的命令行

选项	描述
id	显示用户和组群 id
useradd,usermod,userdel	添加、修改、删除用户
Groupadd.groupmod,groupdel	添加、修改、删除组群
gpasswd	修改/etc/group 配置文件
pwck, grpck	检查用户和组群密码文件完整性。
pwconv, pwunconv	开/关用户影子密码
grpconv, grpunconv	开/关组群影子密码

1.3.3.1 添加用户

要在系统上添加用户：

使用 useradd 命令来创建一个锁定的用户账号：

```
useradd <username>
```

使用 passwd 命令，通过指派密码和密码过期规则来给某账号解锁：

```
passwd <username>
```

useradd 的命令行选项在表格 1-4 中被列出。

表格 1-4 useradd 命令行选项

选项	描述
-c comment	用户的注释
-d home-dir	用来取代默认的 /home/username/ 主目录

选项	描述
-e date	禁用账号的日期，格式为：YYYY-MM-DD
-f days	密码过期后，账号被禁用前要经过的天数（若指定了 0，账号在密码过期后会被立刻禁用。若指定了 -1，密码过期后，账号将不会被禁用）
-g group-name	用户默认组群的组群名或组群号码（该组群在指定前必须存在）
-G group-list	用户是其中成员的额外组群名或组群号码（默认以外的）的列表，用逗号分隔（组群在指定前必须存在）
-m	若主目录不存在则创建它
-M	不要创建主目录
-n	不要为用户创建用户私人组群
-r	创建一个 UID 小于 500 的不带主目录的系统账号
-p password	使用 crypt 加密的密码
-s	用户的登录 shell，默认为 /bin/bash
-u uid	用户的 UID，它必须是独特的，且大于 499

1.3.3.2 添加组群

要给系统添加组群，使用 `groupadd` 命令：

```
groupadd <group-name>
```

`groupadd` 的命令行选项在表格 1-5 中被列出。

表格 1-5 `groupadd` 命令行选项

选项	描述
-g gid	组群的 GID，它必须是独特的，且大于 499
-r	创建小于 500 的系统组群
-f	若组群已存在，退出并显示错误（组群信息不会被改变）。若指定了 -g 和 -f 选项，但是组群已存在，-g 选项就会被忽略

1.3.4 添加用户的详细过程

下列步骤演示了在启用屏蔽密码的系统上使用 `useradd juan` 命令后的情形：

1) 在 `/etc/passwd` 文件中新添了有关 `juan` 的一行。这一行的特点如下：

- a) 它以用户名 `juan` 开头。
 - b) 密码字段有一个“x”，表示系统使用屏蔽密码。
 - c) 500 或 500 以上的 UID 被创建。（在中标麒麟服务器操作系统中，500 以下的 UID 和 GID 被保留给系统使用。）
 - d) 500 或 500 以上的 GID 被创建。
 - e) 可选的 GECOS 信息被留为空白。
 - f) `juan` 的主目录被设为 `/home/juan/`。
 - g) 默认的 shell 被设为 `/bin/bash`。
- 2) 在 `/etc/shadow` 文件中新添了有关 `juan` 的一行。这一行的特点如下：
- a) 它以用户名 `juan` 开头。
 - b) 出现在 `/etc/shadow` 文件中密码字段内的两个叹号(!!)会锁住账号。
 - c) 如果某个加密的密码使用了 `-p` 选项被传递，这个密码会被放置在 `/etc/shadow` 文件中用于该用户的那一行中，密码被设置为永不过期。
- 3) 在 `/etc/group` 文件中新添了一行有关 `juan` 组群的信息。和用户名相同的组群叫做用户私人组群 (user private group)。在 `/etc/group` 文件中新添的这一行具有如下特点：
- a) 它以组群名 `juan` 开头。
 - b) 密码字段有一个“x”，表示系统使用屏蔽密码。
 - c) GID 与列举 `/etc/passwd` 文件中用户 `juan` 行中的相同。
- 4) 在 `/etc/gshadow` 文件中新添了有关 `juan` 组群的一行。这一行的特点如下：
- a) 它以组群名 `juan` 开头。
 - b) 出现在 `/etc/gshadow` 文件中密码字段内的一个叹号(!)会锁住该组群。
 - c) 所有其它字段均为空白。
- 5) 用于用户 `juan` 的目录被创建在 `/home/` 目录之下。该目录为用户 `juan` 和组群 `juan` 所有。它的读写和执行权限仅为用户 `juan` 所有。所有其它权限都被拒绝。

6) /etc/skel/ 目录（包含默认用户设置）内的文件被复制到新建的 /home/juan/ 目录中。

这时候，系统上就存在了一个叫做 **juan** 的被锁的账号。要激活它，管理员必须使用 **passwd** 命令给账号设置一个密码，它还可以设置密码过期规则。

1.4 获取特权

系统普通用户的权限有不同的限制，某些情况下普通需用需要执行管理员用户权限才能执行的命令，此时可以通过 **su** 或者 **sudo** 命令获得管理员权限特权。

1.4.1 Su 命令工具

用户使用 **su** 命令时，需要输入 **root** 用户密码，验证通过后可以获取 **root** 的脚本环境。一旦通过 **su** 命令登入，这个用户的所有操作均视为 **root** 用户操作。由于 **su** 可以获取 **root** 全部权限，并因此获取其他用户的权限，可能存在一定安全问题。因此可以通过管理员组群 **wheel** 来进行限制。以 **root** 用户执行以下命令：

```
# usermod - G wheel username
```

当年将用户加入 **wheel** 组群后，可以限制只有这个组群的用户可以使用 **su** 命令访问。配置 **su** 的 **PAM** 可以编辑 **/etc/pam.d/su** 文件，通过添加删除 **#** 字符来确认添加或删除相应内容。

```
#auth required pam_wheel. so use_uid
```

上述内容表示管理员组群 **wheel** 内的用户可以通过 **su** 访问其他用户。

1.4.2 Sudo 命令工具

Sudo 命令允许系统管理员让普通用户执行一些或者全部的 **root** 命令。当可信用户执行 **sudo** 命令时，需要提供他们自己的用户密码，然后以 **root** 权限执行命令。

基本的 **sudo** 命令如下：

```
#sudo command
```

Sudo 命令有很大的弹性，只有在 **/etc/sudoers** 文件中被允许的用户可以执行

在他们自己的 shell 环境中执行 `sudo` 命令，而不是 `root` 的 shell 环境。这意味着在 7 系列中 `root` 的 shell 环境是被禁止访问的。

配置 `sudo` 必须通过编辑 `/etc/sudoers` 文件，而且只有管理员用户才可以修改它，必须使用 `visudo` 编辑。之所以使用 `visudo` 有两个原因，一是它能够防止两个用户同时修改它；二是它也能进行一些的语法检查。以 `root` 身份用 `visudo` 打开配置文件，输入以下内容：

```
#juan  ALL=(ALL)  ALL
```

这条信息意思是 `juan` 用户可以以任何主机连接并通过 `sudo` 执行任何命令。

下面这条信息说明 `users` 用户可以本地主机可以执行 `/sbin/shutdown - h now` 命令：

```
%users localhost=/sbin/shutdown - h now
```

1.5 License 注册与管理

1.5.1 字符授权管理工具

本节将会指导您从字符界面来使用授权管理工具，您可以从中了解和掌握具体使用步骤和操作方法。

1.5.1.1 生成验证码文件

在字符终端中，以 `root` 身份输入 `nklicadm -g`，就会显示如图 1-3：

```
[root@localhost tmp]# nklicadm -g
```

反馈码生成成功，并被保存到 `/root/feedback.txt`。

其值为： 03609ecec00463cb75f0de26bf8d037a1aa0a
请发电子邮件到 `register@cs2c.com.cn` 以获得您的授权。

```
[root@localhost tmp]# █
```

图 1-3 生成验证码文件界面

其中的反馈码保存到/root/feedback.txt 文件中，是一个 37 位长的字符。您在获得这个码以后，将其发送给 support-server@cs2c.com.cn，以获取授权文件。

或者发送给您中标软件的销售人员、商务人员以获取授权文件。

1.5.1.2 查看系统授权状态

在系统安装后，默认有 60 天的试用期。这时您可以用 nklicadm 的-s 选项来查看系统的状态，查看剩余天数。如果您的系统在试用期内，状态如图 1-4:

```
[root@localhost tmp]# nklicadm -s
认证状态:  *** 试用期授权文件 ***
剩余试用天数: 59 天。
[root@localhost tmp]#
```

图 1-4 试用期界面

这说明您的系统试用期还有 59 天，59 天以后就会显示信息如图 1-5:

```
[root@localhost 桌面]# nklicadm -s
认证状态:  *** 试用过期 ***
很抱歉，您的中标麒麟高级服务器操作系统是不合法的。
```

图 1-5 试用期过期界面

如果您的系统是合法状态，会显示如下图 1-6:

```
[root@localhost ~]# nklicadm -s
认证状态:  *** 认证通过 ***
恭喜您!
您已被授权在这台机器上使用中标麒麟高级服务器操作系统!
[root@localhost ~]#
```

图 1-6 获得授权界面

1.5.1.3 导入授权文件

在试用期或者试用期过期以后，在字符终端中，以 root 身份输入 nklicadm -i license-file.dat，导入合法的授权文件，如果 license.dat 文件合法，就会显示如图 1-7:

```
[root@localhost ~]# nklicadm -i /tmp/03609ecec00463cb75f0de26bf8d037a1aa0a.dat
```

授权文件成功导入！

```
[root@localhost ~]# nklicadm -s
```

认证状态: *** 认证通过 ***

恭喜您！

您已被授权在这台机器上使用中标麒麟高级服务器操作系统！

```
[root@localhost ~]#
```

图 1-7 导入授权文件界面

然后查看系统状态，就变成合法。

如果您导入时使用了错误的授权文件、非授权文件(将无法成功导入)，就会显示如下图 1-8:

```
[root@localhost tmp]# nklicadm -i /var/log/messages
无效的授权文件: /var/log/messages
[root@localhost tmp]# nklicadm -i /tmp/aa.dat
无效的授权文件: /tmp/aa.dat
[root@localhost tmp]#
```

图 1-8 导入错误授权文件界面

如果您的系统已经是合法的，重复导入授权文件，就会出现以下图 1-9:

```
[root@localhost ~]# nklicadm -i /tmp/03609ecec00463cb75f0de26bf8d037a1aa0a.dat
请不要重复导入授权文件
[root@localhost ~]#
```

图 1-9 重复导入界面

1.5.1.4 查看字符授权管理工具帮助

在字符终端下，当您输入 `nklicadm -h`，就会显示如下图 1-10:

```
[root@localhost tmp]# nklicadm -h
用法: nklicadm [选项] [文件]...
```

选项	GNU长选项	含义
-h, -?	--help	显示本帮助信息
-g	--generate	生成验证码
-i <lic_file>	--import <lic_file>	导入授权文件: license.dat
-s	--status	显示您的中标麒麟操作系统的认证状态
-v	--version	显示本软件的版本信息

```
[root@localhost tmp]#
```


图 1-10 帮助信息界面

1.5.1.5 显示工具版本信息

在字符终端中，输入 `nklicadm -v`，会显示您系统上的授权管理工具的版本信息，如图 1-11：

```
[root@localhost tmp]# nklicadm -v
nklicadm - 中标麒麟授权管理工具2.0版
[root@localhost tmp]#
```

图 1-11 版本信息界面

1.5.2 图形授权管理工具

本章将会指导您从图形界面来使用授权管理工具，您可以从中了解和掌握具体使用步骤和操作方法。

注意：

请检查并确认保证当前 **Gnome** 图形登录的系统用户，对当前预导入授权文件有读写权限。

1.5.2.1 图形工具的显示

在 **GNOME** 桌面启动以后，在您系统的右下角找到蓝色圆形按钮图标，如图 1-12：



图 1-12Gnome 桌面界面

鼠标点击蓝色圆形按钮图标，会显示一个【**钥匙**】，如下如图 1-13：

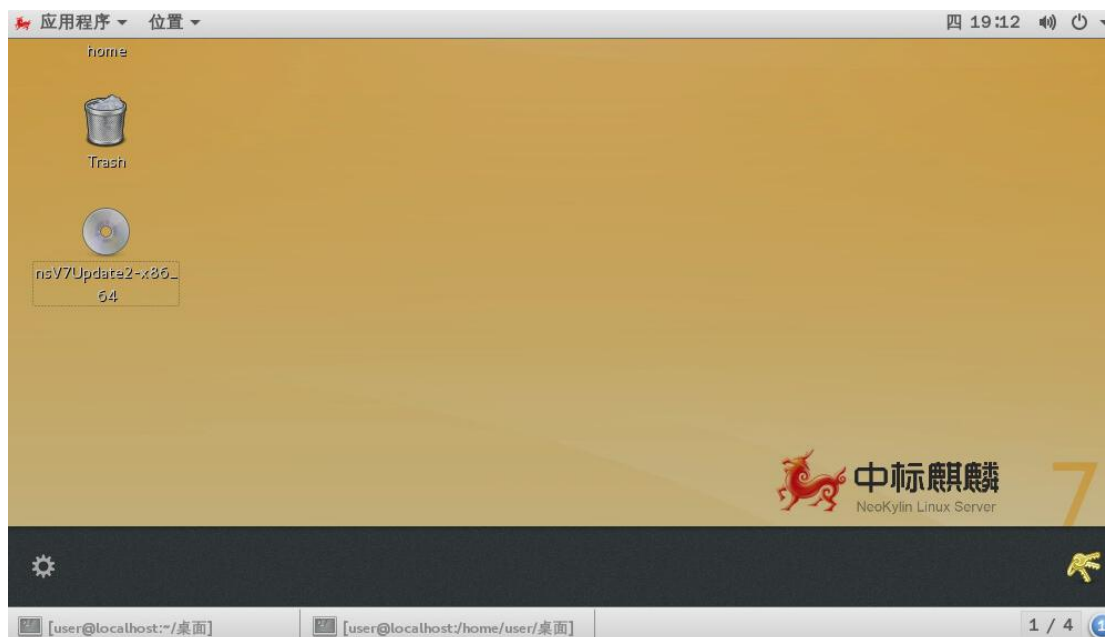


图 1-13 图形工具按钮界面

如果在试用期范围内，钥匙按钮的颜色是黄色；如果过了试用期，钥匙按钮是红色；如果系统是经过授权的，则显示为绿色。

1.5.2.2 授权状态查看

在桌面上，您点击【**钥匙**】，如果您的系统是试用期会弹出如下如图 1-14：

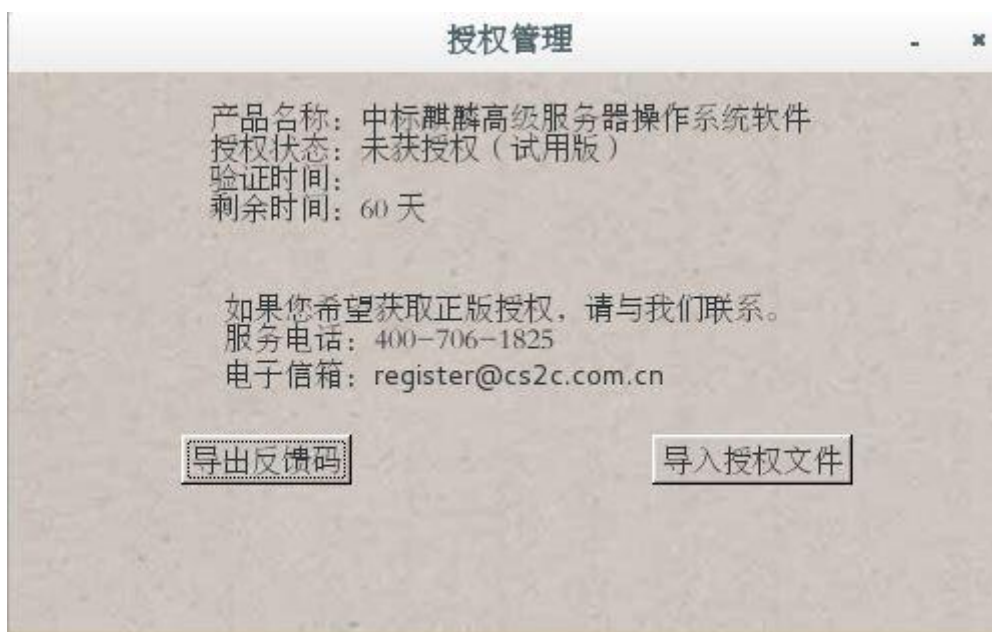


图 1-14 试用期界面

如果您的系统试用期已经过期，会显示如如图 1-15：



图 1-15 试用期过期界面

如果您的系统已经获得授权，会显示如图 1-16：



图 1-16 获得授权界面

当您获得授权以后，就不能再导出注册文件和导入授权文件。

1.5.2.3 导出反馈码文件

在试用期内，您可以通过点击【**钥匙**】→【**导出反馈码**】，弹出如图 1-17 所示的对话框：



图 1-17 导出注册文件界面

然后点击【导出】按钮，出现如下图 1-19 对话框，需要输入 root 密码如图 1-18:



图 1-18 导出反馈码文件界面

保存反馈码文件界面如图 1-19:

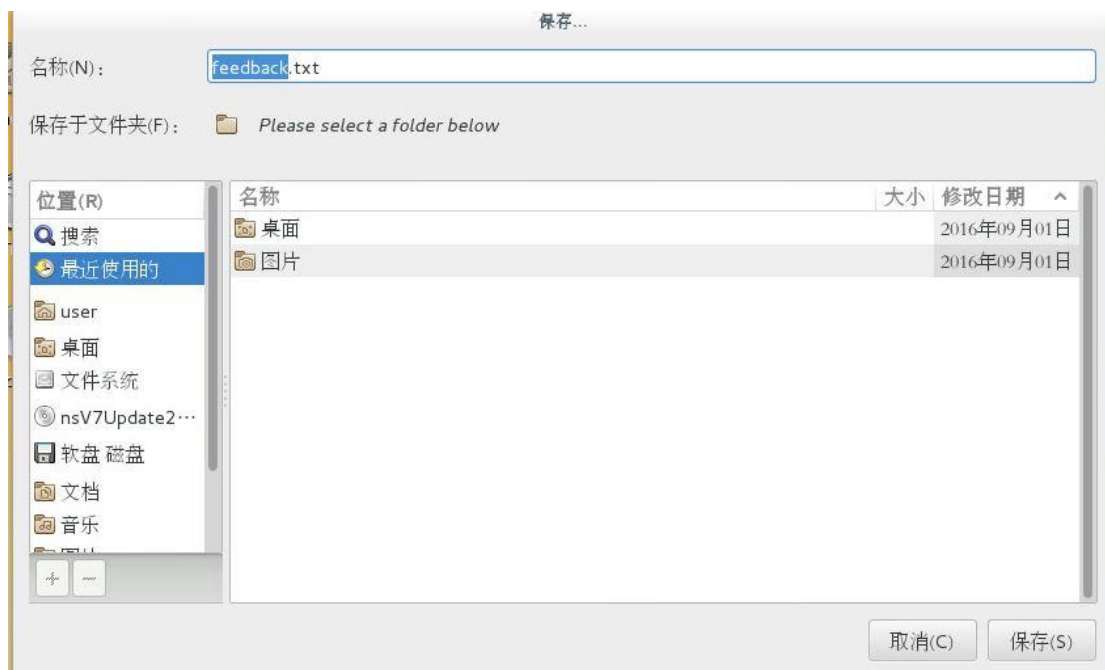


图 1-19 保存反馈码文件界面

您可以选择您希望的保存路径，请您确保对保存目录有写的权限，默认的文件名是 feedback.txt。

如果存放路径具有写的权限，那么就会弹出如图 1-20 的对话框：



图 1-20 保存成功界面

1.5.2.4 导入授权文件

在试用期内，您可以通过点击【**钥匙**】→【**导入授权文件**】，会弹出如图 1-16 的对话框：

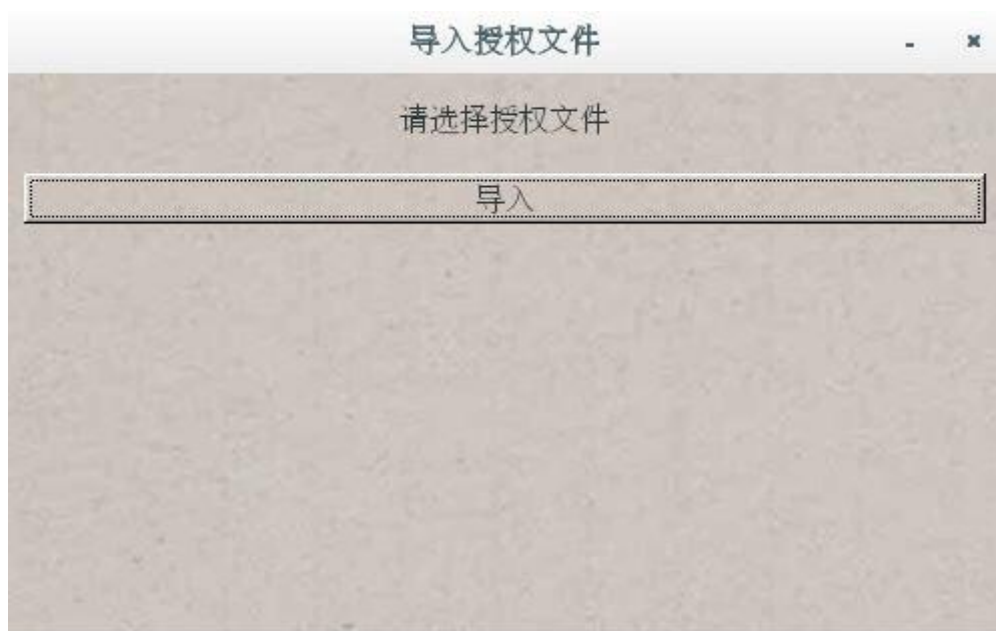


图 1-21 导入授权文件界面

然后再点击【**导入**】按钮，出现如图 1-22 所示的对话框，需要输入 root 密码：



图 1-22 选择授权文件界面

选择您获取的授权文件（必须保证当前 Gnome 登录用户对当前文件有读写权限）如图 1-23：



图 1-23 选择授权文件界面

选择您获取的授权文件(文件后缀为：.dat)所存放的路径，然后点击【打开】

按钮。如果导入的 license 文件是不合法的，那么会弹出如图 1-24 所示的对话框：



图 1-24 导入失败界面

如果导入成功，会弹出如图 1-25 所示的对话框：



图 1-25 导入成功界面

1.5.2.5 试用期过期管理

如果您的系统试用期过期，会每间隔一分钟，弹出如图 1-26 对话框：



图 1-26 试用期过期弹出窗口

2 安装和管理软件

2.1 NKUC-客户端套件

本章将介绍将中标麒麟服务器操作系统注册到升级服务中心的步骤及通过连接升级服务中心实现软件包的安装、卸载和升级。

2.1.1 图形工具注册

在桌面上，点击【应用程序】→【系统工具】→【中标麒麟注册向导】，或执行命令 `nkuc_register`，即可启动中标麒麟注册向导，将您的系统注册到中标麒麟产品升级服务中心，如图 2-1：

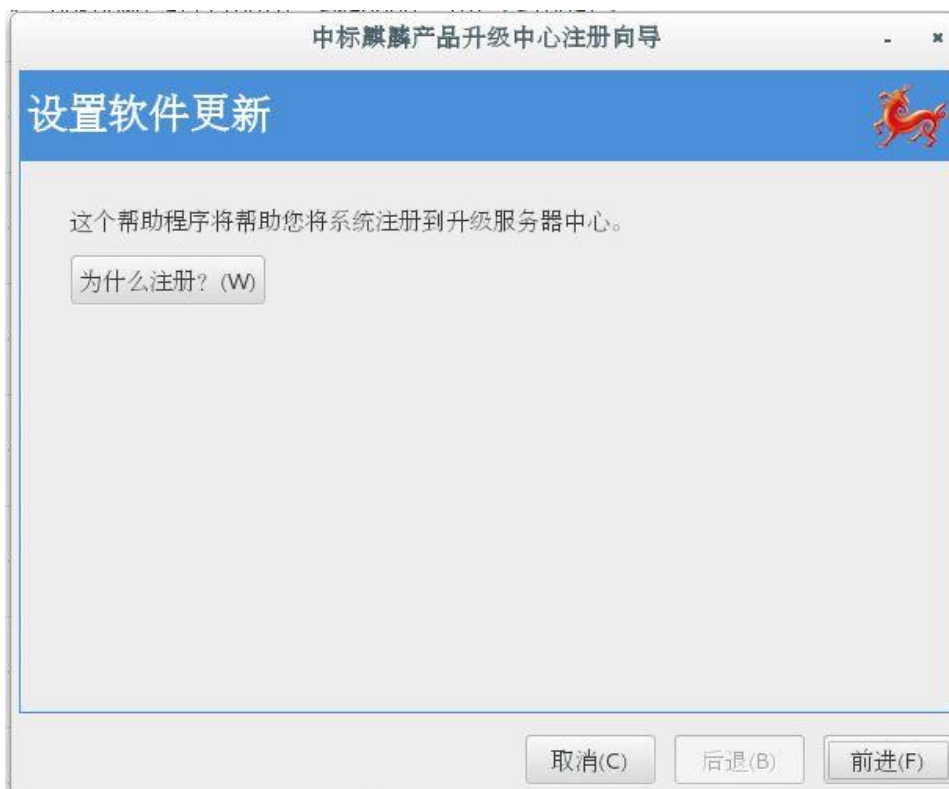


图 2-1 中标麒麟产品升级中心注册向导

中标麒麟注册向导，是一个图形化的注册工具。它可以将您的操作系统注册到中标麒麟产品升级服务中心，注册完成后，您可以接收中标麒麟产品升级服务器中心提供的软件包升级信息，使您的系统能够及时享受软件包更新服务。同时您也能进行软件包的在线安装/卸载，简化系统应用开发和应用软件包的管理。

注册过程包括以下几步：

1. 选择升级服务中心
2. 选择安全证书（默认安全证书正确则不需要手动进行选择）
3. 输入激活码
4. 选择要发送的系统信息
5. 注册

注册到中标软件默认升级服务中心：

设置升级服务中心地址，如图 2-2：

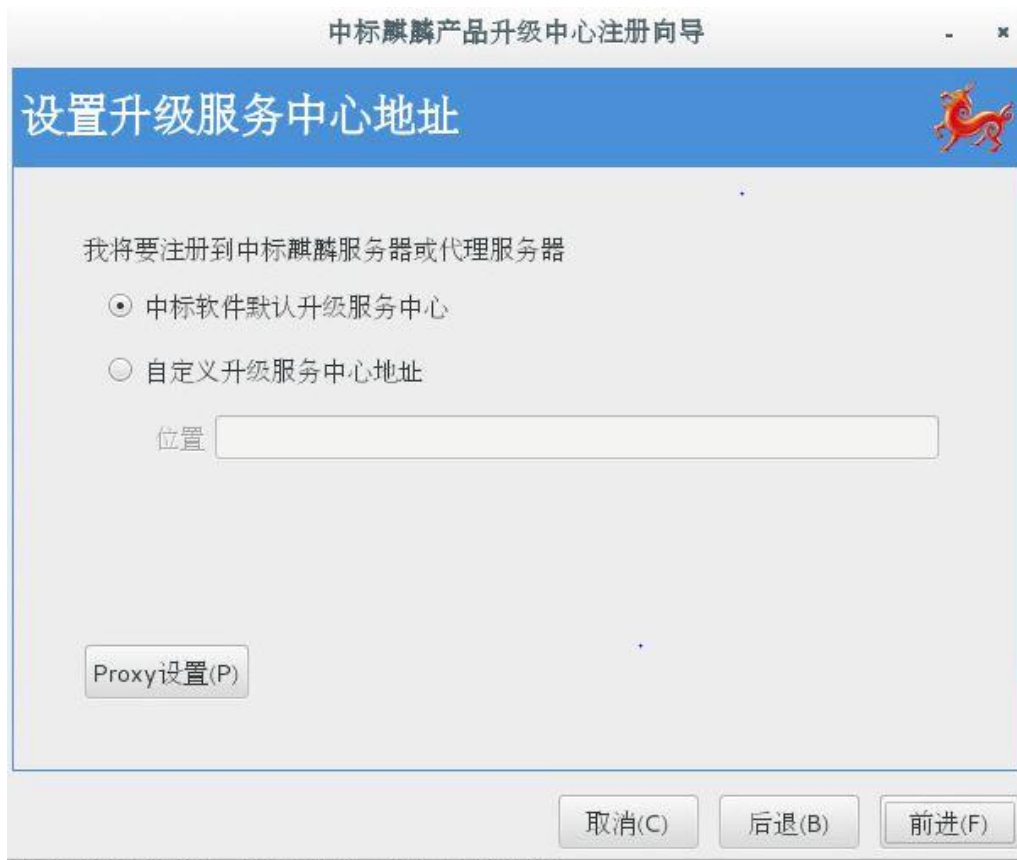


图 2-2 设置升级服务中心地址

“中标麒麟默认升级服务中心”是中标软件有限公司的产品升级中心，包含有中标麒麟的产品升级，将您的系统注册到中标软件升级服务中心，您就可以接收中标麒麟最新的产品信息，升级您的系统到最新版本。“中标麒麟默认升级服务中心”主要用于您的系统可以连接公网，从中标软件有限公司来接收最新的软件包更新信息。

选择“中标麒麟默认升级服务中心”后，点击【前进】，当您系统中的证书

/usr/share/nkuc/NKUC-CA-CERT 与升级中心中的证书匹配时,将进入激活码界面,如图 2-3:

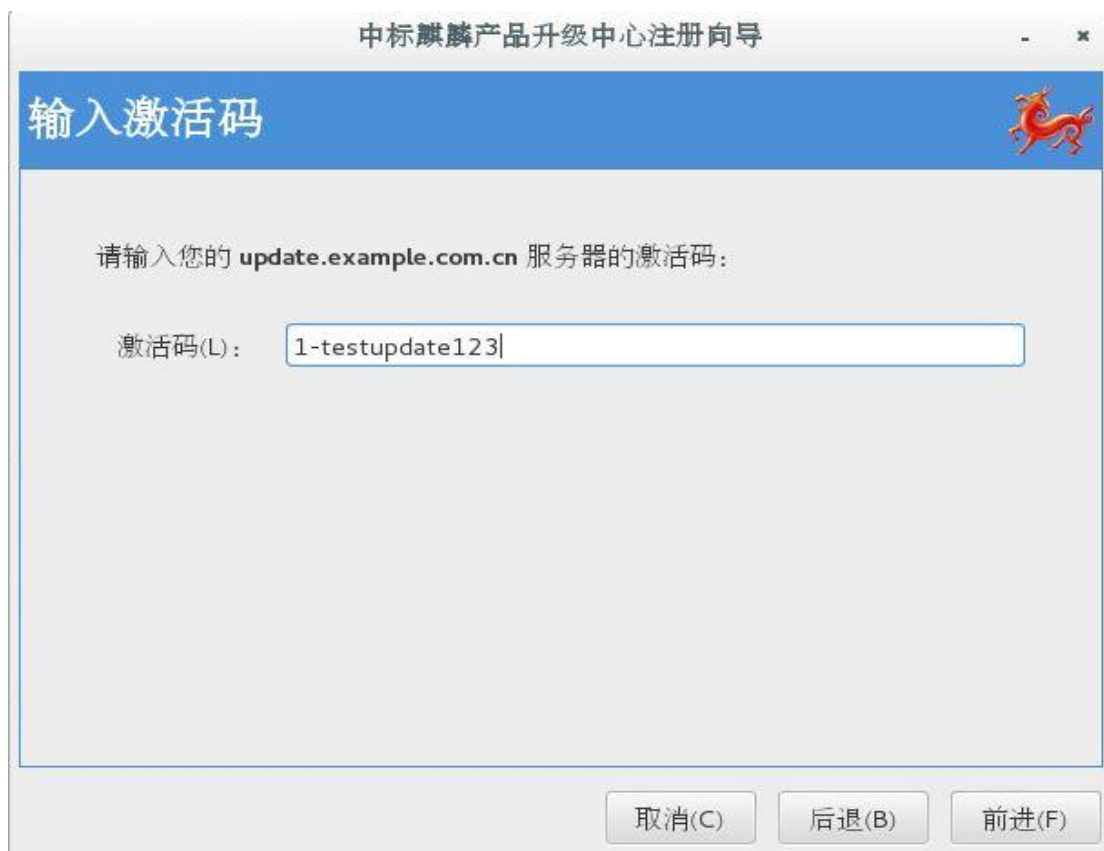


图 2-3 输入激活码

激活码通过中标软件来获取:

服务信箱: support@cs2c.com.cn

支持热线: 400-706-1825

注意: 如果您的安全证书被误删了, 您可以执行以下命令下载证书

`get-neokylin-cert`

下载完成之后再执行注册工具。如果您将证书文件放在了其他路径下, 请手动选择证书文件, 如图 2-4:



图 2-4 选择安全证书

在输入正确的激活码后，点击【前进】，进入创建概要界面，如图 2-5：

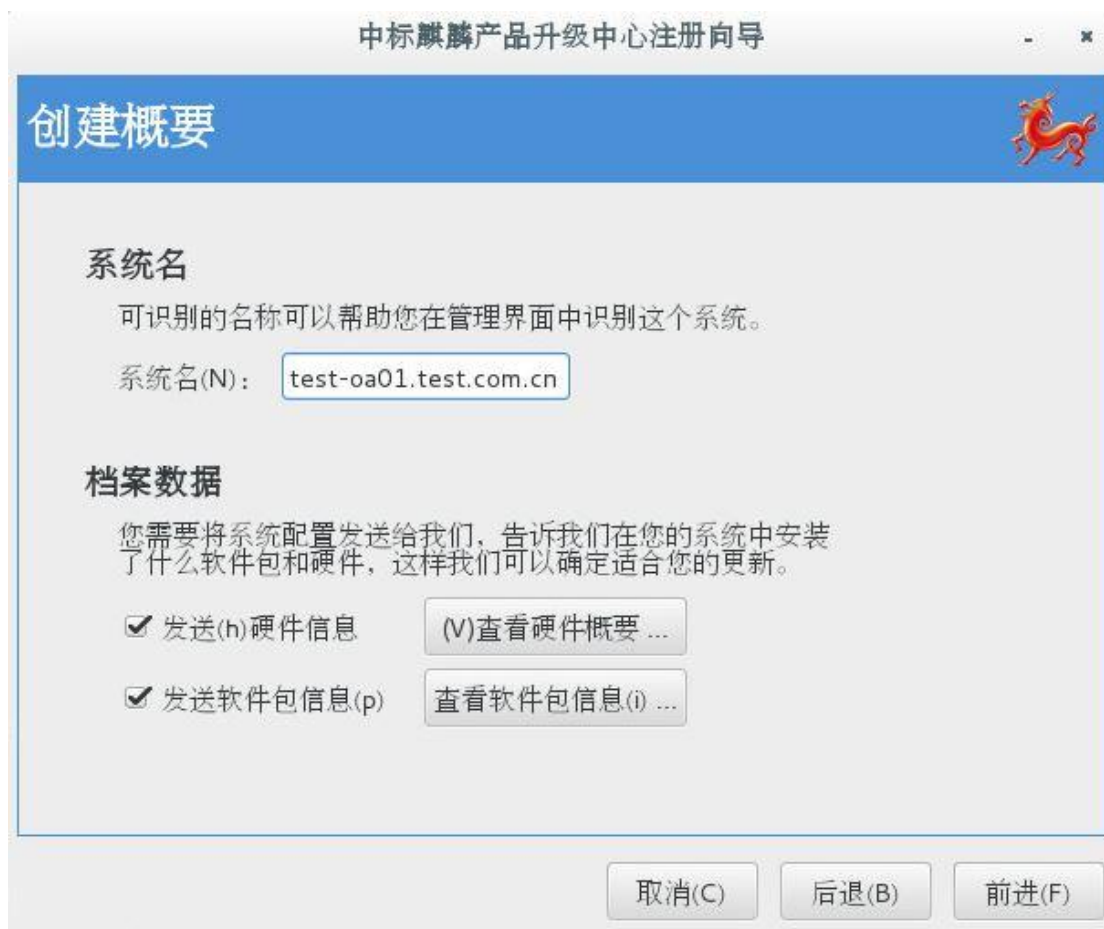


图 2-5 创建概要

在此处您可以选择是否将您的系统硬件信息和软件包信息发送到升级中心。系统名称用来在升级中心中显示您的系统名称，默认使用您的系统名称，可手动修改自定义。配置信息主要包括了硬件信息和您安装的软件包信息。硬件信息包括您安装的中标麒麟高级服务器操作系统版本、主机名称、IP 地址、CPU 型号、CPU 个数和内存等信息（如图 2-5），软件包信息包括您安装的所有的 rpm 软件包。建议将硬件信息和软件包信息发送到升级中心，以便于自动接收来自升级中心的更新信息，否则只能手动来获取升级信息。



图 2-6 查看硬件信息

注册过程如图 2-6，会根据您的选择来发送硬件信息和软件包信息。

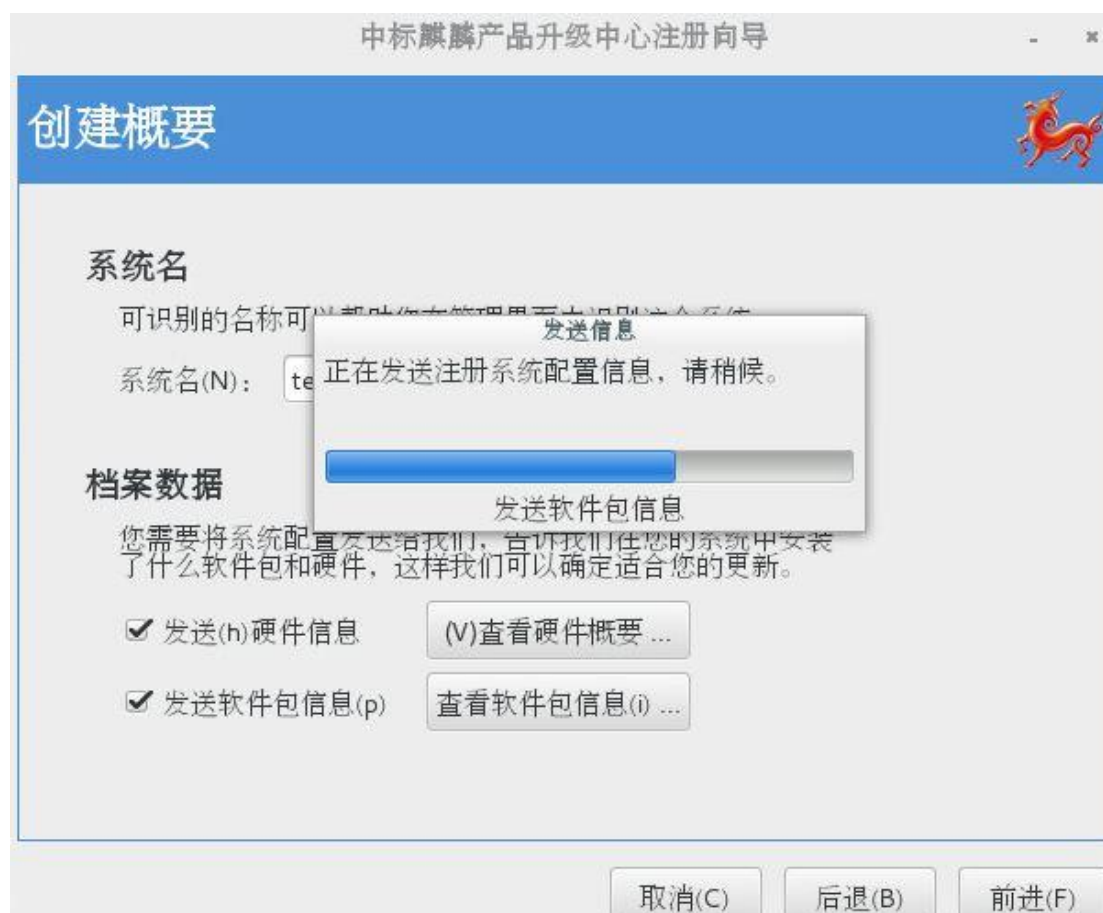


图 2-7 注册中

注册成功如图 2-8:



图 2-8 注册成功

注册到自定义升级服务中心：

“自定义升级服务中心地址”是用户在中标麒麟高级服务器操作系统上安装升级服务中心相关软件包组件，构建的独立升级服务中心，主要用于用户有软件升级需求，但依据管理要求不能连接公网，或者需要自建升级服务中心在内网环境中进行软件包的升级更新、在线安装/卸载、配置文件的分发等。构建和管理自定义服务中心的方法可以参照《NKUC 安装部署手册》和《NKUC 用户手册》。如图 2-9 设置一个自定义升级服务中心地址。

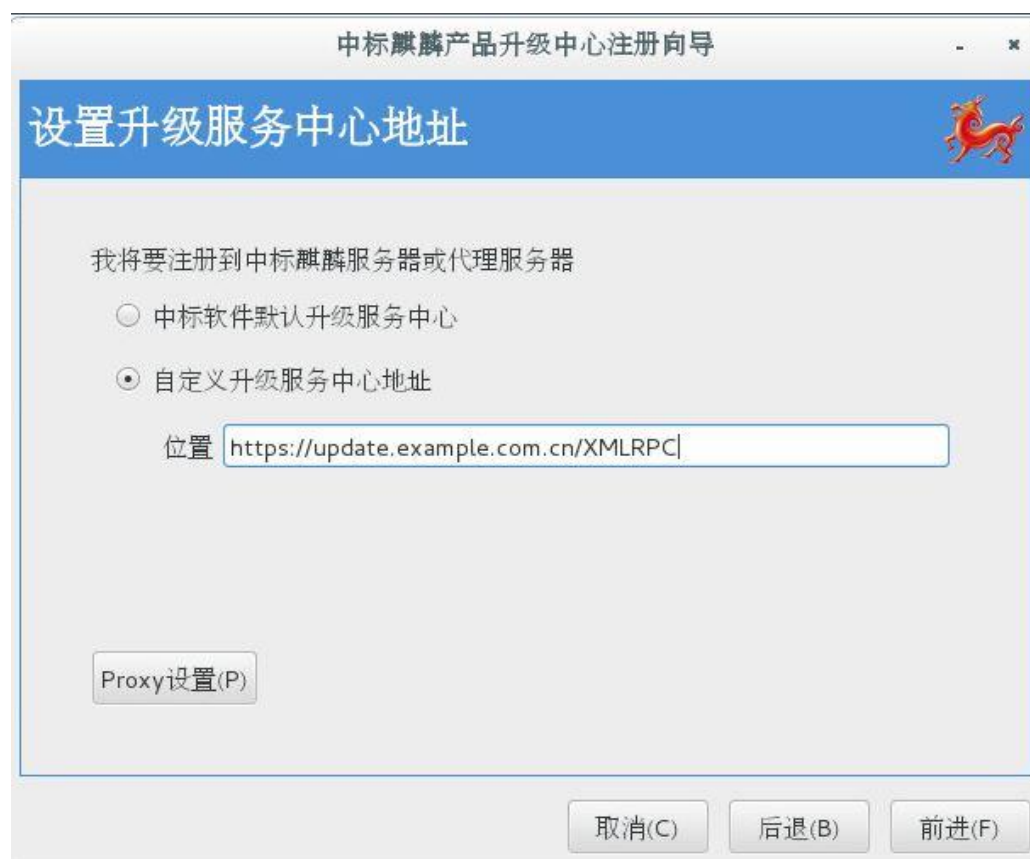


图 2-9 输入自定义升级服务中心地址

输入“自定义升级服务中心地址”后点击【前进】，将对您系统下的
/usr/share/nkuc/NKUC-ORG-TRUSTED-SSL-CERT

证书与您输入的升级服务中心的证书进行适配验证，验证证书正确后进入输入激活码界面，如图 2-10。此时您需要输入您自定义升级服务中心的激活码，此激活码从您的升级服务中心管理员处获取。

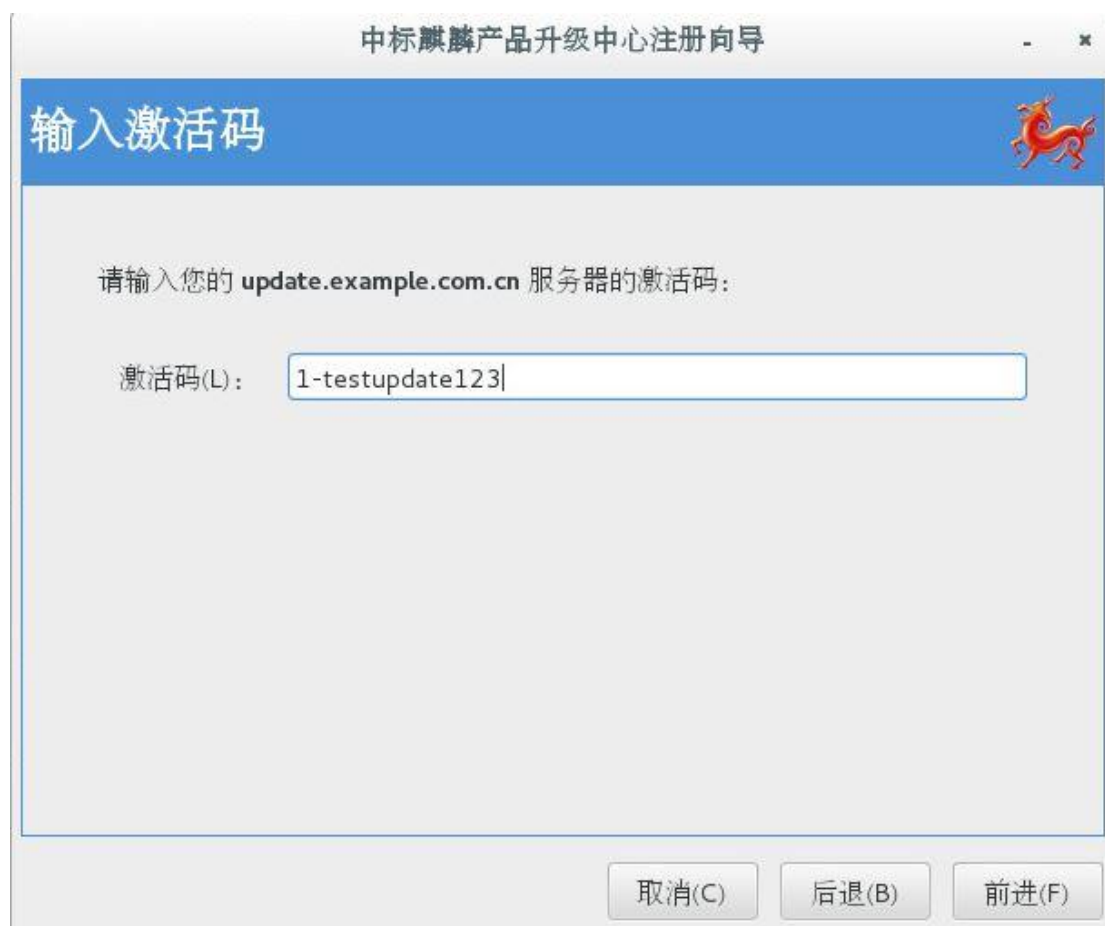


图 2-10 输入激活码

注意：如果未从自定义的升级服务中心下载证书，可以从自定义升级服务中心下载证书，执行以下命令下载证书：

```
get-self-cert [serverUrl]
```

如果在下载自定义升级服务中心证书时出现错误，请与您的升级服务中心管理员联系，确保升级中心的证书文件路径和名称为：

```
/var/www/html/pub/NKUC-ORG-TRUSTED-SSL-CERT
```

证书下载完成后再次运行注册工具并选择下载的证书文件，如图2-11 所示。



图 2-11 选择自定义升级服务中心证书

在输入正确的激活码后，点击【前进】，进入创建概要界面，如图 2-12。

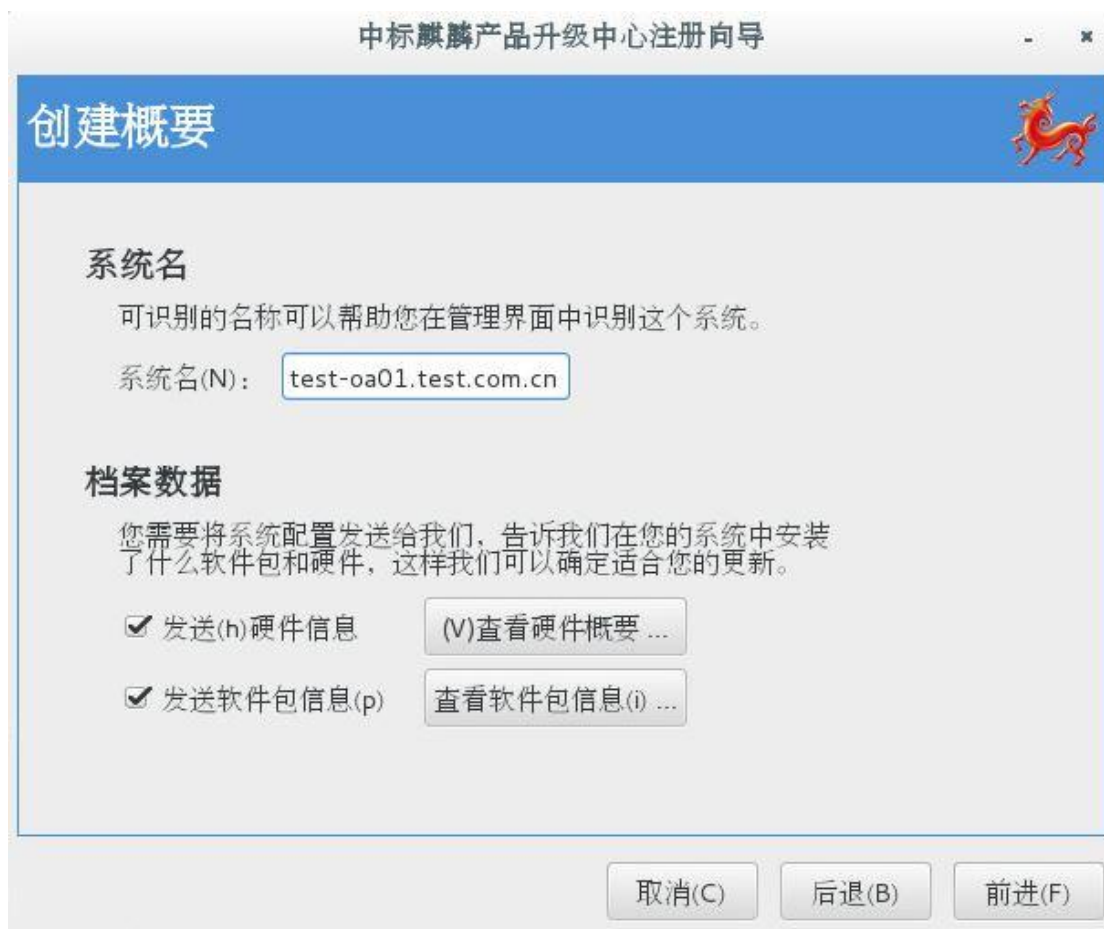


图 2-12 创建概要

在此处您可以选择是否将您的系统硬件信息和软件包信息发送到升级中心。系统名称用来在升级中心中显示您的系统名称，默认使用您的系统名称，可手动修改自定义。配置信息主要包括了硬件信息和您安装的软件包信息。硬件信息包括您安装的中标麒麟高级服务器操作系统版本、主机名称、IP 地址、CPU 型号、CPU 个数和内存等信息（如图 6-35），软件包信息包括您安装的所有的 rpm 软件包。建议将硬件信息和软件包信息发送到升级中心，以便于自动接收来自升级中心的更新信息，否则只能手动来获取升级信息。



图 2-13 查看硬件信息

注册过程如图 2-14，会根据您的选择来发送硬件信息和软件包信息。



图 2-14 注册中

注册成功如图 2-15：



图 2-15 注册成功

查看系统连接仓库如图 2-16:

```
[root@test-0a01 rhn]# yum repolist
已加载插件: langpacks, nkucplugin
This system is receiving updates from NKUC Classic or NeoKylin Satellite.
源标识          源名称          状态
ns7update2-lic-x86_64-os-base NeoKylin Linux Advanced Server V7Update2(x8 10,538
repolist: 10,538
[root@test-0a01 rhn]# hostname
test-0a01.test.com.cn
[root@test-0a01 rhn]#
```

图 2-16 查看系统连接仓库

2.1.2 命令行注册

如果您使用的系统没有安装桌面工具或者您想通过命令行来进行注册，可以通过命令 `nkucreg_ks` 来实现注册。执行以下命令来查看帮助信息：

```
nkucreg_ks --help
```

在使用命令注册时，必需的参数有以下几个：

`--serverUrl` 指定使用的服务器，例如：<https://update.cs2c.com.cn/XMLRPC>

`--activationkey` 指定使用的激活码信息

--sslCACert 指定安装证书文件

注册到中标麒麟升级服务中心，命令如下：

```
nkureg_ks --serverUrl=https://update.cs2c.com.cn/XMLRPC
```

```
--activationkey=1-testnkuc123
```

```
--sslCACert=/usr/share/nkuc/NKUC-CA-CERT
```

注册到自定义升级服务中心地址，命令如下：

```
nkureg_ks --serverUrl=[serverUrl] --activationkey=[activationkey]
```

```
--sslCACert=[SSL Cert Path]
```

成功将系统注册到升级中心后，可以通过 yum 命令来管理本地软件包。

2.1.3 软件包升级

对系统中的软件包升级可以通过图形工具和 yum 命令来完成。

1. 图形工具升级软件包

可以通过【应用程序】→【系统工具】→【Software Update】如图 2-20。

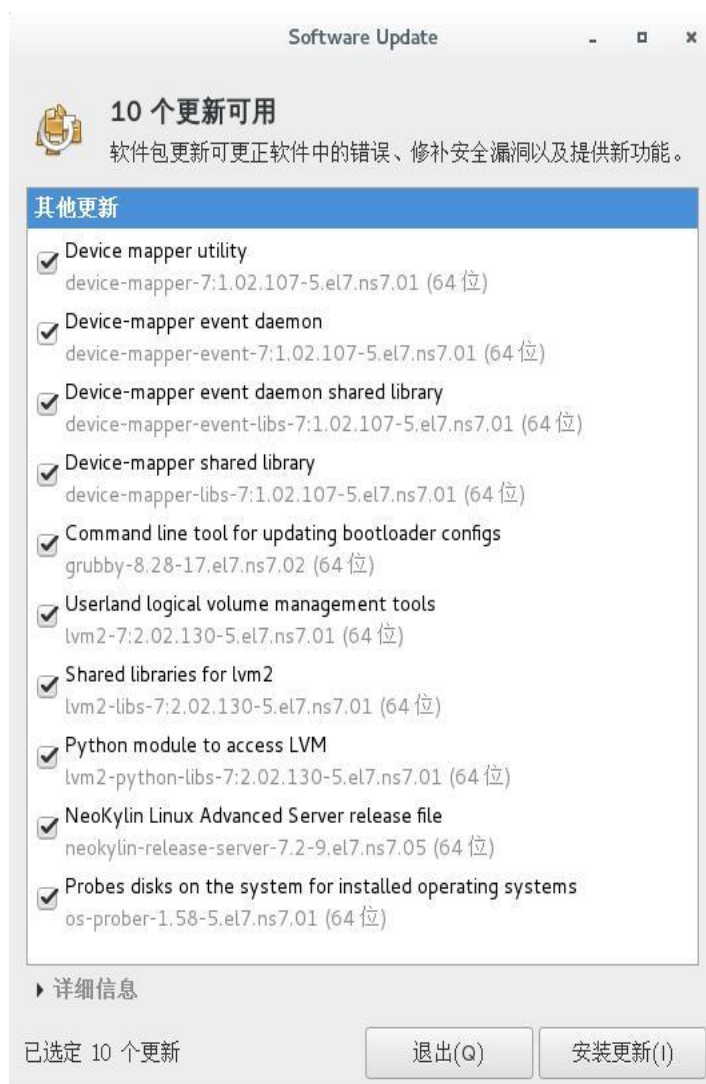


图 2-17 可用软件更新

选择软件包可点击【安装更新】即可更新软件包。此工具只用于检测可更新的软件包，对于未安装的软件包不进行检测，也无法显示。可使用命令 `gpk-update-viewer` 打开此工具。更多详情请查看 `Update Viewer` 帮助手册。

2. yum 命令更新软件包

执行下列命令查看可用的更新：

```
yum check-update
```

升级软件包可执行：

```
yum update 【package】
```

如果不指定软件包将升级所有的可用软件包，指定软件包将只升级一个软件包。

2.2 yum

Yum 是基于 RPM 包的前端软件包管理器，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、安装。

用户可以创建、添加和删除软件包安装源，yum 可以覆盖 rpm 工具的全部功能，并可通过简洁命令优化功能。Yum 还可以简便的管理在本机上安装和下载的 rpm 包，也可利用自己的特性下载软件包。

2.2.1 检查和升级软件包

2.2.1.1 软件包升级检查

查看系统里已经安装的软件包有哪些可以升级可以执行以下命令，示例如下：

```
# yum check-update

Loaded plugins: langpacks, product-id
Updating Neokylin repositories.
INFO: rhsm- app. repolib: repos updated: 0
PackageKit. x86_64 0. 5. 8- 2. el6 neokylin
PackageKit- glib. x86_64 0. 5. 8- 2. el6 neokylin
PackageKit- yum. x86_64 0. 5. 8- 2. el6 neokylin
PackageKit- yum- plugin. x86_64 0. 5. 8- 2. el6 neokylin
glibc. x86_64 2. 11. 90- 20. el6 neokylin
glibc- common. x86_64 2. 10. 90- 22 neokylin
kernel. x86_64 2. 6. 31- 14. el6 neokylin
rpm. x86_64 4. 7. 1- 5. el6 neokylin
rpm- libs. x86_64 4. 7. 1- 5. el6 neokylin
rpm- python. x86_64 4. 7. 1- 5. el6 neokylin
yum. noarch 3. 2. 24- 4. el6 neokylin
... ..
```

示例说明：

- PackageKi t——软件包名称;
- x86_64——该软件包支持的 CPU 架构;
- 0.5.8——可升级的软件包版本
- neokyin——可升级的软件包所存储仓库

上述输出可以看出还可升级内核 kernel, yum 和 rpm 包本身以及它们的依赖关系。

2.2.1.2 升级软件包

Yum 支持一次升级单个/批量软件包, 并同时安装/更新相应的依赖包。

1. 升级单一软件包命令:

```
# yum update package_name
```

升级 rpm 软件包示例:

```
#yum update rpm
Loaded plugins: langpacks, product- id, subscription- manager
Updating Red Hat repositories.
INFO: rhsm- app. repolib: repos updated: 0
Setting up Update Process
Resolving Dependencies
- - > Running transaction check
- - - > Package rpm. x86_64 0: 4. 11. 1- 3. el7 will be updated
- - > Processing Dependency: rpm = 4. 11. 1- 3. el7 for package: rpm-
libs-
4. 11. 1- 3. el7. x86_64
- - > Processing Dependency: rpm = 4. 11. 1- 3. el7 for package: rpm-
python-
4. 11. 1- 3. el7. x86_64
- - > Processing Dependency: rpm = 4. 11. 1- 3. el7 for package: rpm-
build-
```


4. 11. 1- 3. el7. x86_64

--> Package rpm. x86_64 0: 4. 11. 2- 2. el7 will be an update

--> Running transaction check

...

--> Finished Dependency Resolution

Dependencies Resolved

=====

=====

=====

Package Arch Version Repository

Size

=====

=====

=====

Updating:

rpm x86_64 4. 11. 2- 2. el7 rhel

1. 1 M

Updating for dependencies:

rpm- build x86_64 4. 11. 2- 2. el7 rhel

139 k

rpm- build- libs x86_64 4. 11. 2- 2. el7 rhel

98 k

rpm- libs x86_64 4. 11. 2- 2. el7 rhel

261 k

rpm- python x86_64 4. 11. 2- 2. el7 rhel

74 k

Transaction Summary

```
=====
=====

=====

Upgrade 1 Package ( +4 Dependent packages)

Total size: 1.7 M

Is this ok [y/d/N]:
```

上述输出的说明如下：

- a) Loaded plugins: langpacks, product-id——显示已安装和可用的 yum plug-ins 信息。
- b) rpm.x86_64: 用户需要下载升级的软件包和依赖软件包。
- c) Yum 默认会显示升级软件包的基本信息，并提示是否确认安装，用户可以在使用 yum 命令是添加参数 -y，效果等同于出现 Is this ok [y/d/N]: 时输入 yes。输入 d 则进行软件包下载。
- d) 安装过程中如果出现错误导致安装过程终止，可以使用 yum history 命令查看详细描述。

如果需要安装一组软件包，可以以 root 用户执行命令：

```
# yum group update group_name
```

2. 批量升级软件包及其依赖

如果需要升级系统所有软件包，可以使用以下命令：

```
# yum update
```

2.2.1.3 利用系统光盘与 yum 离线升级系统

当系统处于离线状态或者无法访问官方更新源时，可以利用更新的系统光盘创建本地 yum 源并进行升级。步骤如下：

1. 创建系统光盘挂载目录，以 root 用户执行：

```
# mkdir mount_dir
```

2. 将系统安装光盘挂载至该目录，以 root 用户执行

```
# mount -o loop iso_name mount_dir
```

3. 将系统光盘中的 media.repo 文件从挂载目录拷贝至/etc/yum.repo s.d/目录下:

```
# cp mount_dir/media.repo /etc/yum.repos.d/new.repo
```

4. 编辑/etc/yum.repos.d/new.repo 配置文件以添加光盘路径:

```
# baseurl=file://mount_dir
```

5. 更新 yum 源并进行升级, 以 root 用户执行:

```
# yum update
```

6. 升级成功后, 卸载系统光盘挂载目录:

```
# umount mount_dir 或者 rmdir mount_dir
```

如果不再使用这个 yum 源进行安装和升级, 可以以 root 用户删除文件:

```
# rm /etc/yum.repos.d/new.repo
```

将系统从 7.0 升级到 7.2 的示例如下:

1. 创建系统光盘挂载目录, 以 root 用户执行:

```
# mkdir /media/nkas7.2
```

2. 将系统安装光盘挂载至该目录, 以 root 用户执行

```
# mount -o loop nkas-server-7.2-x86_64-dvd.iso /media/nkas7.2
```

3. 将系统光盘中的 media.repo 文件从挂载目录拷贝至/etc/yum.repo s.d/目录下:

```
# cp /media/nkas7.2/media.repo /etc/yum.repos.d/nkas7.2.repo
```

4. 编辑/etc/yum.repos.d/nkas7.repo 配置文件以添加光盘路径:

```
# baseurl=file:///media/nkas7.2/
```

5. 更新 yum 源并进行升级, 以 root 用户执行:

```
# yum update
```

6. 升级成功后，卸载系统光盘挂载目录：

```
# umount /media/nkas7.2/或者 rmdir /media/nkas7.2/
```

7. 如果不再使用这个 yum 源进行安装和升级，可以以 root 用户删除文件：

```
# rm /etc/yum. repos. d /nkas7.2.repo
```

2.2.2 管理软件包

Yum 提供了完整操作系统软件包管理功能，包括检索、查看信息、安装和删除。

2.2.2.1 检索软件包

执行 yum search 命令可以检索软件包，例如检索包含“meld”和“kompare”字段的软件包示例如下：

```
# yum search meld kompare
Loaded plugins: langpacks, langpacks, product- id
Updating NeoKylin repositories.
INFO: rhsm- app. repolib: repos updated: 0
===== N/S matched: kompare
=====
ko mpare. x86_64 : Diff tool
...
Name and summary matches mo stl y, use " search all" for everything.
Warning: No matches found for: meld
```

如果 yum 检测的结果繁多，可以通过 shell 本身的 grep 或者正则表达式进行过滤。

2.2.2.2 安装包列表

显示已安装和可安装的软件包列表可以执行以下命令：

```
# yum list all
```

显示包括某些字符的已安装和可安装软件包列表可以执行以下命令：

```
# yum list glob_expression...
```

显示 `abrt` 相关软件包列表的示例如下：

```
# yum list abrt- addon\* abrt- plugin\*
Loaded plugins: langpacks, product- id
Updating NeoKylin repositories.
INFO: rhsm- app. repolib: repos updated: 0
Installed Packages
abrt- addon- ccpp. x86_64 1. 0. 7- 5. el6
@neokylin
abrt- addon- kerneloops. x86_64 1. 0. 7- 5. el6
@ neokylin
abrt- addon- python. x86_64 1. 0. 7- 5. el6
@ neokylin
abrt- plugin- bugzilla. x86_64 1. 0. 7- 5. el6
@ neokylin
abrt- plugin- logger. x86_64 1. 0. 7- 5. el6
@neokylin
abrt- plugin- sosreport. x86_64 1. 0. 7- 5. el6
@neokylin
abrt- plugin- ticketuploader. x86_64 1. 0. 7- 5. el6
@ neokylin
```

显示包括某些字符的已安装软件包列表可以执行以下命令：

```
# yum list installed glob_expression...
```

显示包括 krb 的所有已安装软件包示例如下：

```
# yum list installed "krb?- *"

Loaded plugins: langpacks, product- id
Updating NeoKylin repositories.
INFO: rhsm- app. repolib: repos updated: 0

Installed Packages

krb5- libs. x86_64 1. 8. 1- 3. el6

@neokylin

krb5- workstation. x86_64 1. 8. 1- 3. el6

@ neokylin
```

显示包括某些字符的可安装软件包列表可以执行以下命令：

```
# yum list available glob_expression...
```

显示所有可用的 gstreamer plug-ins 软件包列表：

```
# yum list availableg streamer\*plugin\*

Loaded plugins: langpacks, product- id
Updating NeoKylin repositories.
INFO: rhsm- app. repolib: repos updated: 0

Available Packages

gstreamer- plugins- bad- free. i686 0. 10. 17- 4. el6

rhel

gstreamer- plugins- base. i686 0. 10. 26- 1. el6

neokylin

gstreamer- plugins- base- devel. i686 0. 10. 26- 1. el6

neokylin

gstreamer- plugins- base- devel. x86_64 0. 10. 26- 1. el6
```

```
neokylin
gststreamer- plugins- good. i686 0. 10. 18- 1. el6
neokylin
```

查看软件仓库

成功注册后，可使用 yum 来管理软件包。

查看可用的软件仓库可以使用以下命令：

```
# yum repolist
```

如果想显示更多信息可以加上-v 选项，或者用 yum repoinfo 命令输出信息。

```
# yum repolist -v
```

```
# yum repoinfo
```

如果需要显示所有可用和不可用的软件仓库，可以使用以下命令：

```
# yum repolist all
```

2.2.2.3 显示软件包信息

显示一个或多个软件包可以使用以下命令：

```
# yum info package_name...
```

显示软件包 abrt 详细信息的示例：

```
# yum info abrt

Loaded plugins: langpacks, product- id
Updating NeoKylinrepositories.
INFO: rhsm- app. repolib: repos updated: 0

Installed Packages

Name : abrt
Arch : x86_64
```

```
Version : 1. 0. 7
Release : 5. el6
Size : 578 k
Repo : installed
From repo : rhel
Summary : Automatic bug detection and reporting tool
URL : https://fedorahosted.org/abrt/
License : GPLv2+
Description: abrt is a tool to help users to detect defects in
applications
: and to create a bug report with all informations needed
by
: maintainer to fix it. It uses plugin system to extend its
: functionality.
```

用户还可以通过查询 yum 数据库查询软件包相关信息:

```
# yumdb info package_name
```

显示软件包 yum 详细信息的示例:

```
# yum info yum
Loaded plugins: langpacks, product-id
yum- 3. 2. 27- 4. el6. noarch
checksum_data =
23d337ed51a9757bbfbdceb82c4eaca9808ff1009b51e9626d540f44fe95f7
71
checksum_type = sha256
from_repo = rhel
from_repo_revision = 1298613159
```



```
from_repo_timestamp = 1298614288

installed_by = 4294967295

reason = user

releasever = 6.1
```

2.2.2.4 安装软件包

用户可以以 root 用户使用以下命令安装软件包

```
# yumdb install package_name
```

安装 sqlite 的 i686 架构的软件包示例：

```
# yum i nstall sqlite.i686
```

除了安装软件包，还可以安装具体的二进制文件，你可以输入文件地址，以 root 用户执行安装：

```
# yum install /usr/sbin/named
```

安装流程示例：

```
# yum install httpd
Loaded plugins: langpacks, product-id
Resolving Dependencies
--> Running transaction check
---> Package httpd. x86_64 0: 2.4.6-12.el7 will be updated
---> Package httpd. x86_64 0: 2.4.6-13.el7 will be an update
--> Processing Dependency: 2.4.6-13.el7 for package: httpd- 2.4.6-13.el7. x86_64
--> Running transaction check
---> Package httpd- tools. x86_64 0: 2.4.6-12.el7 will be updated
---> Package httpd- tools. x86_64 0: 2.4.6-13.el7 will be an update
--> Finished Dependency Resolution
```

Dependencies Resolved

Package Arch Version Repository

Size

Updating:

httpd x86_64 2.4.6-13.el7 rhel-x86_64-server-7

1.2 M

Updating for dependencies:

httpd-tools x86_64 2.4.6-13.el7 rhel-x86_64-server-7

77 k

Transaction Summary

Upgrade 1 Package (+1 Dependent package)

Total size: 1.2 M

Is this ok [y/d/N]:

Downloading packages:

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

```
Updating : httpd- tools- 2. 4. 6- 13. el7. x86_64
```

```
1/4
```

```
Updating : httpd- 2. 4. 6- 13. el7. x86_64
```

```
2/4
```

```
Cleanup : httpd- 2. 4. 6- 12. el7. x86_64
```

```
3/4
```

```
Cleanup : httpd- tools- 2. 4. 6- 12. el7. x86_64
```

```
4/4
```

```
Verifying : httpd- 2. 4. 6- 13. el7. x86_64
```

```
1/4
```

```
Verifying : httpd- tools- 2. 4. 6- 13. el7. x86_64
```

```
2/4
```

```
Verifying : httpd- tools- 2. 4. 6- 12. el7. x86_64
```

```
3/4
```

```
Verifying : httpd- 2. 4. 6- 12. el7. x86_64
```

```
4/4
```

```
Updated:
```

```
httpd. x86_64 0: 2. 4. 6- 13. el7
```

```
Dependency Updated:
```

```
httpd- tools. x86_64 0: 2. 4. 6- 13. el7
```

```
Complete!
```

如果要安装本地软件包，可以执行：

```
# yum localinstall path
```

2.2.2.5 下载软件包

在执行安装流程中，显示以下选项是：

```
...
```

```
Total size: 1.2 M
```

```
Is this ok [y/d/N]:
```

```
...
```

输入 `d`，可以执行软件包下载。软件包默认下载路径为 `/var/cache/yum/$basearch/$releasever/packages/`。

2.2.2.6 删除软件包

删除软件包可以执行以下命令：

```
yum remove package_name...
```

删除 `totem` 软件包示例：

```
yum remove totem
```

2.2.3 管理软件包组

软件包组可以搜集一系列特定功能软件包，比如系统工具和视频软件包组。安装软件包组可以一起安装其依赖。

2.2.3.1 软件包组列表

`Summary` 选项可以显示软件包组的基本信息：

```
# yum groups summary
```

以下为输出示例：

```
# yum groups summary

Loaded plugins: langpacks, product- id, subscription- manager

Available Environment Groups: 12

Installed Groups: 10

Available Groups: 12
```

显示某个软件包组的全部信息可以用以下命令：

```
# yum groups info glob_expression...
```

以下为办公软件包组输出示例：

```
# yum group info LibreOffice

Loaded plugins: langpacks, product- id

Group: LibreOffice

Group- Id: libreoffice

Description: LibreOffice Productivity Suite

Mandatory Packages:

=libreoffice- calc

libreoffice- draw

- libreoffice- emailmerge

libreoffice- graphicfilter

=libreoffice- impress

=libreoffice- math

=libreoffice- writer

+libreoffice- xsltfilter

Optional Packages:

libreoffice- base

libreoffice- pyuno
```

说明如下：

“-” 软件包没有安装并且也不会作为包组内容安装。

“+” 软件包没有安装但是进行软件包或包组升级时将会被安装。

“=” 软件包已经安装且作为包组的一部分。

No symbol: 软件包已经安装，且不属于包组。

2.2.3.2 安装软件包组

每个软件包组都有自己的组 ID，要显示包组 id 可以使用以下命令：

```
# yum group list ids
```

查找 KDE 桌面软件包组列表的示例：

```
# yum group list ids kde\*

Available environment groups:

KDE Plasma Workspaces ( kde- desktop- environment)

Done
```

有些包组是作为因此软件仓库的，例如，使用因此命令选项示例：

```
# yum group list hidden ids kde\*

Loaded plugins: product- id

Available Groups:

KDE ( kde- desktop)

Done
```

软件包组的安装可以通过软件包组名称安装，也可通过包组 id 安装。

```
# yum group install "group name"

# yum group install groupid
```

也可用通过以下两种命令安装：

```
# yum install @group

# yum install @^group
```

下面是 4 中安装 KDE 桌面软件分组的示例：

```
# yum group install "KDE Desktop"

# yum group install kde-desktop

# yum install @"KDE Desktop"

# yum install @kde-desktop
```

2.2.3.3 删除软件包组

可以通过软件包组名或者软件包组 id 删除软件包。以 root 权限执行：

```
# yum group remove group_name

# yum group remove groupid
```

如果软件分组有@标签，也可用以下命令删除。以 root 身份执行：

```
# yum remove @group
# yum remove @^group
```

删除 KDE 桌面软件分组示例：

```
# yum group remove "KDE Desktop"
# yum group remove kde- desktop
# yum remove @"KDE Desktop"
# yum remove @kde-desktop
```

2.2.4 软件包操作记录管理

Yum 所有的操作记录默认存在放在/va/lib/yum/history/目录，并可以使用 yum history 命令进行管理操作。

2.2.4.1 查看操作

显示以往 20 条 yum 操作记录，可以使用以下命令。以 root 权限执行：

```
# yum history list
```

如果想显示所有 yum 操作记录，可以使用以下命令。以 root 权限执行：

```
# yum history list all
```

如果想显示某一部分 yum 操作记录，可以使用以下命令。以 root 权限执行：

```
# yum history list start_id. . end_id
```

显示过去 5 条 yum 信息示例如下：

```
# yum history list 1..5
Loaded plugins: langpacks, product- id
ID | Login user | Date and time | Action( s)
| Altered
-----
-----
```

```

-----

5 | User <user> | 2013- 07- 29 15: 33 | Install |
1
4 | User <user> | 2013- 07- 21 15: 10 | Install |
1
3 | User <user> | 2013- 07- 16 15: 27 | I, U |
73
2 | System <unset> | 2013- 07- 16 15: 19 | Update |
1
1 | System <unset> | 2013- 07- 16 14: 38 | Install |
1106
history list

```

以上 yum history list 输出显示内容说明如下：

ID——识别特定记录的标示数；

Login user——区别 yum 命令的操作用户；

Date and time——该条记录的日期和时间；

Action(s)——操作的具体内容；

Altered——记录操作影响的条目数；

下表是 Action 的不同说明：

表格 2-1 Action 说明

Action	缩写	描述
Downgrade	D	下载更新包
Erase	E	删除软件包
Install	I	安装软件包
Obsoleting	O	软件包标注废弃
Reinstall	R	软件包重装

Update	U	升级软件包
--------	---	-------

如果想要同步 rpmdb 或者 yumdb 数据库，可以使用以下命令：

```
# yum history sync
```

如果想显示 yum 历史及数据库状态信息可以使用以下命令：

```
# yum history stats
```

输出 yum history stats 示例如下：

```
# yum history stats

Loaded plugins: langpacks, product- id
File : //var/lib/yum/history/history- 2012- 08- 15. sqlite
Size : 2, 766, 848
Transactions: 41
Begin time : Wed Aug 15 16: 18: 25 2012
End time : Wed Feb 27 14: 52: 30 2013
Counts :
NEVRAC : 2, 204
NEVRA : 2, 204
NA : 1, 759
NEVR : 2, 204
rpm DB : 2, 204
yum DB : 2, 204
history stats
```

Yum 还支持提供过去记录的总结信息，以 root 身份执行以下信息：

```
# yum history summary
```

如果只想查看某一阶段的记录总结信息，以 root 身份执行以下信息：

```
# yum history summary start_id. . end_id
```

显示最后 5 条记录总结信息示例如下：

```
# yum history summary 1..5

Loaded plugins: langpacks, product-id
Login user | Time | Action(s) |
Altered
-----
-----

Jaromir... <j hradilek> | Last day | Install |
1J
aromir... <j hradilek> | Last week | Install |
1J
aromir... <j hradilek> | Last 2 weeks | I, U |
73
System <unset> | Last 2 weeks | I, U |
1107
history summary
```

Yum 记录显示还支持通过软件包进行查找，命令如下：

```
# yum history package- list glob_expression...
```

2.2.4.2 审查操作

需要显示某条操作记录的具体综述信息，可以执行以下命令：

```
# yum history summary id
```

其中 *id* 是操作的 *id*。

如果需要显示某条操作记录的详细信息，可以使用以下命令：

```
# yum history info id
```

如果需要显示某一阶段操作记录的详细信息，可以使用以下命令：

```
# yum history info start_id. . end_id
```

示例如下：

```
# yum history info 4 . 5
Loaded plugins: langpacks, product-id
Transaction ID : 4. . 5
Begin time : Thu Jul 21 15: 10: 46 2011
Begin rpmdb : 1107: 0c67c32219c199f92ed8da7572b4c6df64eacd3a
End time : 15: 33: 15 2011 ( 22 minutes)
End rpmdb : 1109: 1171025bd9b6b5f8db30d063598f590f1c1f3242
User : Jaromir Hradilek <j hradilek>
Return- Code : Success
Command Line : install screen
Command Line : install yum- plugin- aliases
Transaction performed with:
Installed rpm- 4. 8. 0- 16. el6. x86_64
Installed yum- 3. 2. 29- 17. el6. noarch
Installed yum- metadata- parser- 1. 1. 2- 16. el6. x86_64
Packages Altered:
Install screen- 4. 0. 3- 16. el6. x86_64
Install yum- plugin- aliases- 1. 1. 30- 6. el6. noarch
history info
```

同样的，用户还可以查询附件信息，命令如下：

```
# yum history addon- infoid
```

以及显示最后的记录附加信息命令：

```
# yum history addon- info last
```

示例如下：

```
# yum history addon- info 4

Loaded plugins: langpacks, product- id

Transaction ID: 4

Available additional history information:

config- main

config- repos

saved_tx

history addon- info
```

2.2.4.3 恢复与重复操作

如果想要撤销某个 yum 操作，可以以 root 权限执行一下操作：

```
# yum history undo id
```

如果需要重复某个 yum 操作，可以以 root 权限执行一下操作：

```
# yum history redo id
```

2.2.4.4 启用新的操作历史

Yum 的操作记录存为一个单独的 SQLite 数据文件。如果需要启用新的操作记录，可以以 root 权限执行命令：

```
# yum history new
```

这个命令会在/var/lib/yum/history/目录创建新的数据库文件，旧的数据库文件也会保留。

2.2.5 配置 yum 和 yum 仓库

Yum 的配置文件为/etc/yum.conf。其中【main】字段是必填选项，用于配置 yum 的基本规则。它也包含放在/etc/yum.repos.d/目录的.repo 文件作为仓库。

配置【main】选项

```
# [main]
```

```
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum. log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=3
[comments abridged]
# PUT YOUR REPOS HERE OR IN separate files named file. repo
# in /etc/yum. repos. d
```

各字段说明如下：

Cachedir: yum 缓存的目录，yum 在此存储下载的 rpm 包和数据库，默认是 /var/cache/yum。

Debuglevel: 除错级别，0-10,默认是 2。

Exactarch: 有两个选项 1 和 0,代表是否只升级和当前安装软件包 cpu 体系一致的包，如果设为 1，则如你安装了一个 i386 的 rpm，则 yum 不会用 1686 的包来升级。

Gpgchkeck: 有 1 和 0 两个选择，分别代表是否进行 gpg 校验，如果没有这一项，默认是检查的。

Keepcache: 缓存是否保存，1 保存，0 不保存。

Logfile: yum 的日志文件，默认是/var/log/yum.log。

Obsoletes: 这是一个 update 的参数，相当于 upgrade，允许更新陈旧的 RPM 包。默认为 1 启动更新。

Pkgpolicy: 包的策略。一共有两个选项，newest 和 last，这个作用是如果你设置了多个 repository，而同一软件在不同的 repository 中同时存 在，yum 应该安装哪一个，如果是 newest，则 yum 会安装最新版本。如果是 last，则 yum

会将服务器 id 以字母表排序，并选择最后的那个服务器上的软件安装。一般都是选 newest。

distroverpkg: 指定一个软件包，yum 会根据这个包判断你的发行版本，默认是 redhat-release，也可以是安装的任何针对自己发行版的 rpm 包。

Retries: 网络连接发生错误后的重试次数，如果设为 0，则会无限重试。

配置 **【repository】** 选项

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

每一个仓库选项 **【repository】** 都必须包括以下内容：

name=repository_name: 软件源的名称，将被 yum 获取并识别。

baseurl = repository_url: 是服务器设置中最重要的部分，只有设置正确，才能从上面获取软件。它的格式是：

baseurl=url://server1/path/to/repository/

url://server2/path/to/repository/

url://server3/path/to/repository/

其中 url 支持的协议有 http:// ftp:// file://三种。baseurl 后可以跟多个 url，你可以自己改为速度比较快的镜像站，但 baseurl 只能有一个，也就是说不能像如下格式：

baseurl=url://server1/path/to/repository/

baseurl=url://server2/path/to/repository/

baseurl=url://server3/path/to/repository/

其中 url 指向的目录必须是这个 repository header 目录的上一级，它 also 支持 \$releasever \$basearch 这样的变量。

enabled= value: 这个选项表示这个 repo 中定义的源是否启用的，0 为禁用，1 为启用。

gpgcheck= value: 这个选项表示这个 repo 中下载的 rpm 将进行 gpg 的校验，已确定 rpm 包的来源是有效和安全的。1 为启用，0 为禁用。

以下为一个 repo 文件示例：

```
[ns7-adv-x64-os]
name=NeoKylin Linux Advanced Server 7 - Os
baseurl=http://IP-address /NS/V7Update2/os/adv/$basearch/
gpgcheck=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-neokylin-release
enabled=1

[ns7-adv-x64-updates]
name=NeoKylin Linux Advanced Server 7 - Updates
baseurl=http://IP-address /NS/V7Update2/os/updates/adv/$basearch/
gpgcheck=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-neokylin-release
enabled=0

[ns7-adv-x64-addons]
name=NeoKylin Linux Advanced Server 7 - Addons
baseurl=http:// IP-address /NS/V7Update2/addons/adv/$basearch/
gpgcheck=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-neokylin-release
```

使用 Yum 变量

配置 yum 文件也可使用变量参数，说明如下：

\$releasever: 发行版的版本，从[main]部分的 distroverpkg 获取，如果没有，则根据 redhat-release 包进行判断。

\$arch: cpu 体系结构，如 i586, i686, x86_64 等

\$basearch, cpu 的基本体系组，如 i686 和 athlon 同属 i386, alpha 和 alphaev6 同属 alpha。

查看当前配置

查看 yum 当前所有配置，可以执行以下命令：

```
yum-config-manager
```

如果只想查看某个或某些字段，可以执行以下命令：

```
yum-config-manager section...
```

如果想查看匹配某些配置的字段，可以执行以下命令：

```
yum-config-manager glob_expression...
```

以下是显示【main】字段配置信息的示例：

```
#yum-config-manager main \*
Loaded plugins: langpacks, product-id
===== main
=====

[main]
alwaysprompt = True
assumeyes = False
bandwidth = 0
bugtracker_url = https://bugzilla.redhat.com/enter_bug.cgi?
product=Red%20Hat%20Enterprise%20Linux%206&component=yum
cache = 0

[output truncated]
```

添加、启用和禁用 yum 软件仓库

Yum 支持用户添加软件仓库，添加软件仓库的命令如下，以 root 身份执行：

```
yum-config-manager --add-repo repository_url
```

其中 repository_url 是.repo 文件的链接。

下边是添加 `http://www.example.com/example.repo` 软件仓库的示例：

```
# yum-config-manager --add-repo http://www.
example.com/example.repo

Loaded plugins: langpacks, product-id
adding repo from: http://www.example.com/example.repo
grabbing file http://www.example.com/example.repo to
/etc/yum.repos.d/example.repo
example.repo | 413 B
00:00

repo saved to /etc/yum.repos.d/example.repo
```

启用 yum 软件仓库命令如下，以 root 权限执行：

```
# yum-config-manager --enable repository...
```

如果只想启用某一个字段，可以以 root 权限执行：

```
# yum-config-manager --enable glob_expression...
```

例如，启用在 `/etc/yum.conf` 中配置的【example】【example-debuginfo】和【examplesource】字段。

```
# yum-config-manager--enable example\*

Loaded plugins: langpacks, product-id

===== repo: example
=====

[example]

bandwidth = 0

base_persistdir = /var/lib/yum/repos/x86_64/7Server

baseurl = http://www.example.com/repo/7Server/x86_64/

cache = 0

cachedir = /var/cache/yum/x86_64/7Server/example
```

```
[output truncated]
```

如果想启用/etc/yum.repos.d /目录下的所有软件仓库和/etc/yum.conf 文件中的仓库，示例如下：

```
# yum-config-manager --enable \*

Loaded plugins: langpacks, product-id

===== repo: example
=====

[example]

bandwidth = 0

base_persistdir = /var/lib/yum/repos/x86_64/7Server

baseurl = http://www.example.com/repo/7Server/x86_64/

cache = 0

cachedir = /var/cache/yum/x86_64/7Server/example

[output truncated]
```

禁用 yum 软件仓库

禁用软件仓库命令如下，以 root 权限执行：

```
# yum-config-manager --enable repository...
```

如果只想启用某一个字段，可以以 root 权限执行：

```
# yum-config-manager --disable glob_expression...
```

如果想禁用/etc/yum.repos.d /目录下的所有软件仓库和/etc/yum.conf 文件中的仓库，示例如下：

```
# yum-config-manager --disable \*

Loaded plugins: langpacks, product-id

===== repo: example
=====
```

```
[example]

bandwidth = 0

base_persistdir = /var/lib/yum/repos/x86_64/7Server

baseurl = http://www.example.com/repo/7Server/x86_64/

cache = 0

cachedir = /var/cache/yum/x86_64/7Server/example

[output truncated]
```

创建 yum 软件仓库

创建软件仓库步骤如下：

1. 安装 createrepo 软件包：

```
# yum install createrepo
```

将想要创建仓库的所有软件包复制到一个目录，例如/mnt/local_repo/。

2. 创建软件仓库

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

2.2.6 Yum 插件

查看 yum 插件命令如下：

```
# yum info yum

Loaded plugins: langpacks, product-id

[output truncated]
```

启用、配置和禁用 yum 插件

启用 yum 插件需要修改/etc/yum.conf 文件的【main】字段，确保 plugins 的值为 1：

```
# plugins=1
```

如果禁用则赋值为 0：

```
# plugins=0
```

每个已安装插件都有自己的配置文件目录/etc/yum/pl uginconf.d/, 以下是 aliases 插件的配置文件 aliases.conf 示例:

```
[main]

enabled=1
```

如果想使用一条命令禁用一个或多个插件, 可以通过参数 - -d isableplugin=plugin_name 实现。例如禁用 aliases 插件如下:

```
# yum update --disableplugin= aliases
```

禁用 aliases 和 lang 相关插件示例如下:

```
## yum update --disableplugin= aliases
```

安装附加 yum 插件

安装附加插件通常以 yum- pl ugin-plugin_name 规则命令, 例如 kabi 的插件包名为 kabi-yum- plugins, 安装插件软件包与安装 rpm 类似。Aliases 插件安装示例如下:

```
# yum install yum-plugin-aliases
```

管理 yum 插件

下述是一些常用的 yum 插件管理命令:

search-disabled-repos(subscription-manager)

search-disabled-repos 插件允许临时或永久的启用软件库用于解决软件一览。启用该插件后, 当 yum 安装时因为缺少依赖包失败是, 插件可以提供临时的检索软件库并重试安装, 之后用户可以自行配置该软件库是否永久启用。

product- id(subscription-manager)

product- id 插件可以管理产品标示认证。product- id 是默认安装的。

langpacks(yum-langpacks)

langpacks 插件用于检索软件包安装语言, langpacks 是默认安装的。

aliases(yum-plugin-aliases)

aliases 插件用于为 yum 命令添加 aliases 命令行选项。

yum-changelog(yum-plugin-changelog)

yum-changelog 插件用于提供变更的 changelog，通过参数--changelog 命令行添加。

yum-tmprepo(yum-plugin-tmprepo)

yum-tmprepo 插件用于检查确认软件仓库安全性，默认情况下该插件不允许禁用 gpg 检查。

yum-verify(yum-plugin-verify)

yum-verify 插件用于添加 verify, verify-rpm 以及 verify-all 命令行，以便提供系统的认证信息。

3 基础服务

该章节介绍如何配置系统服务、后台 daemon 以及远程访问 NeoKylin Linux Advanced Server V7 系统。

3.1 使用 systemd 管理系统服务

3.1.1 Systemd 介绍

systemd 是 Linux 下一个与 SysV 和 LSB 初始化脚本兼容的系统和服管理器。systemd 使用 socket 和 D-Bus 来开启服务，提供基于守护进程的按需启动策略，保留了 Linux cgroups 的进程追踪功能，支持快照和系统状态恢复，维护挂载和自挂载点，实现了各服务间基于从属关系的一个更为精细的逻辑控制，拥有前卫的并行性能。systemd 无需经过任何修改便可以替代 sysvinit。

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 有以下几种类型：

表格 3-1 unit 类型介绍

Unit 类型	文件后缀	描述
Service unit	.service	守护进程的启动、停止、重启和重载是此类型

Target unit	.target	此类 unit 为其他 unit 进行逻辑分组。
Automount unit	.automount	此类 unit 封装系统结构层次中的一个自挂载点。
Device unit	.device	此类 unit 封装一个存在于 Linux 设备树中的设备。
Mount unit	.mount	此类 unit 封装系统结构层次中的一个挂载点。
Path unit	.path	此类 unit 为文件系统中的文件或者目录。
Scope unit	.scope	此类 unit 为外部创建进行。
Slice unit	.slice	此类 unit 为一组管理系统进程的分层 unit。
Snapshot unit	.snapshot	与 target unit 相似，快照本身不做什么，唯一的目的是引用其他 unit
Socket unit	.socket	此类 unit 封装系统和互联网中的一个 socket。
Swap unit	.swap	此类 unit 封装 swap 设备或者 swap 文件。
Timer unit	.timer	此类 unit 封装系统时间

Systemd unit 的文件目录说明如下：

表格 3-2 Systemd unit 介绍

目录	描述
/usr/lib/systemd/system/	RPM 包安装时发布的 Systemd units。
/run/systemd/system/	运行时创建的 Systemd units，该目录任务会覆盖安装时自带的 Systemd units。
/etc/systemd/system/	系统创建与管理的 Systemd units，该目录任务会覆盖运行时 Systemd units。

主要特性

systemd 提供以下特性：

- 基于 socket 的并行性能：为了加速整个系统启动和并行启动更多的进程，systemd 在实际启动守护进程之前创建监听 socket，然后传递 socket 给守护进程。在系统初始化时，首先为所有守护进程创建 socket，然后再启动所有的守护进程。如果一个服务因为需要另一个服务的支持而没有完全启动，而这个连接可能正在提供服务的队列中排队，那么这个客户

端进程在这次请求中就处于阻塞状态。不过只会有这一个客户端进程会被阻塞，而且仅是在这一次请求中被阻塞。服务间的依赖关系也不再需要通过配置来实现真正的并行启动(因为一次开启了所有的 socket，如果一个服务需要其他的服务，它显然可以连接到相应的 socket)。

- **D-Bus 激活策略启动服务：**通过使用总线激活策略，服务可以在接入时马上启动。同时，总线激活策略使得系统可以用微小的消耗实现 D-Bus 服务的提供者与消费者的同步开启请求。(同时开启多个服务，如果一个比总线激活策略中其他服务快就在 D-Bus 中排队其请求，直到其他管理确定自己的服务信息为止)。
- 提供守护进程的按需启动策略。
- 保留了使用 Linux cgroups 进程的追踪功能：每一个执行了的进程获得它自己的一个 cgroup，配置 systemd 使其可以存放在 cgroup 中已经经过外部配置的服务非常简单。(如使用 libcgroups utilities)。
- 支持快照和系统状态恢复：快照可以用来保存/恢复系统初始化时所有的服务和 unit 的状态。它有两种主要的使用情况：允许用户暂时进入一个像 "Emergency Shell" 的特殊状态，终止当前的服务；提供一个回到先前状态的简单方法，重新启动先前暂时终止的服务。
- 维护挂载和自挂载点：systemd 监视所有的挂载点的进出情况，也可以用来挂载或卸载挂载点。/etc/fstab 也可以作为这些挂载点的一个附加配置源。通过使用 comment=fstab 选项你甚至可以标记 /etc/fstab 条目使其成为由 systemd 控制的自挂载点。
- 现了各服务间基于依赖关系的一个精细的逻辑控制：systemd 支持服务(或 unit)间的多种依赖关系。在 unit 配置文件中使用 After/Before、Requires 和 Wants 选项可以固定 unit 激活的顺序。Requires 和 Wants 表示一个正向(强制或可选)的需求和依赖关系，Conflicts 表示一个负向的需求和依赖关系。其他选项较少用到。如果一个 unit 需要启动或关闭，systemd 就把它和它的依赖关系添加到临时执行列表，然后确认它们的相互关系是否一致(或所有 unit 的先后顺序是否含有循环)。如果答

案是否的话，systemd 将尝试修复它，删除可以消除循环的无用工作。

兼容性

- 与 SysV 初始化脚本兼容：如果可能，它会利用 LSB 和 chkconfig 的头信息内容，否则，就使用其他可用信息，如：/etc/rc.d 。这些初始化脚本仅仅是作为一个附加的配置源，以减少 systemd 服务固有的路径数目。
- /etc/fstab 配置文件：这只是另一个配置源。通过使用 comment= fstab 选项标记 /etc/fstab 条目，使 systemd 可以控制自挂载点。
- 支持简单的模板/实例机制：例如只有一个作为示例的 getty@.service 文件，而不是为六个 getty 都准备一个配置文件。接口部分甚至可以被直接继承，也就是说，可以简单的调用 avahi-autoipd@eth0.service 服务配置 dhcpcd@eth0.service ，使得字符串 eth0 的值可以直接通过通配符匹配得到。
- 在一定程度上兼容 /dev/initctl 。这个兼容性实际上是为了执行 FIFO-activated 服务。(只是简单地把原先的请求转换为 D-Bus 请求)事实上，这也意味着旧的 Upstart 和 sysvinit 中的 shutdown、poweroff 和其他相似命令可以在 systemd 中继续使用。
- 与 utmp 和 wtmp 兼容。
- 它可以控制由它催生的每一个程序。
- 本地配置文件使用与.desktop 文件相近的语法：很多软件架构中都有这个简单的语法的分析器。它也可以借用已有的国际化的服务描述工具，语法都是相似的，没有必要再学习新的语法。

3.1.2 管理系统服务

systemctl 是最主要的工具。它融合 service 和 chkconfig 的功能于一体。你可以使用它永久性或只在当前会话中启用/禁用服务。

表格 3-3 service 与 systemctl 的区别

service	systemctl	描述
---------	-----------	----

service name start	systemctl start name.service	用来启动一个服务 (并不会重启现有的)
service name stop	systemctl stop name.service	用来停止一个服务 (并不会重启现有的)。
service name restart	systemctl restart name.service	用来停止并启动一个服务。
service name condrestart	systemctl try-restart name.service	重启一个正在运行的服务
service name reload	systemctl reload name.service	当支持时, 重新装载配置文件而不中断等待操作。
service name status	systemctl status name.service systemctl is-enabled name.service	查询服务状态
service --status-all	systemctl list-units --type service --all	显示所有服务状态

表格 3-4 chkconfig 与 systemctl 的区别

在下次启动时或满足其他触发条件时设置服务为启用	chkconfig name on	systemctl enable name.service
在下次启动时或满足其他触发条件时设置服务为禁用	chkconfig name off	systemctl disable name.service
用来检查一个服务在当前环境下被配置为启用还是禁用。	chkconfig --list name	systemctl status name.service systemctl is-enabled name.service
输出在各个运行级别下服务的启用和禁用情况	chkconfig --list	systemctl list-unit-files --type service
显示 unit 前置启动服务	chkconfig --list	systemctl list-dependencies --after
显示 unit 后导启动服务	chkconfig --list	systemctl list-dependencies --before

显示服务

输出激活的单元：

```
# systemctl
```

以下命令等效：

```
# systemctl list-units
```

如需输出激活服务的单元，执行以下命令：

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

输出加载服务的单元，执行以下命令：

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

输出所有服务的单元，执行以下命令：

```
# systemctl list-units-files --type service
```

以下为显示当前所有激活服务的示例：

```
# systemctl list-units --type service
UNIT LOAD ACTIVE SUB DESCRIPTION
abrt- ccpp. service loaded active exited Install ABRT
coredump hook
abrt- oops. service loaded active running ABRT kernel log
watcher
abrt- vmcore. service loaded active exited Harvest vmcores
for ABRT
abrt- xorg. service loaded active running ABRT Xorg log
watcher
abrt. service loaded active running ABRT Automated
Bug Reporting Tool
...
```

systemd- vconsole- setup. service loaded active exited Setup Virtual Console

tog- pegasus. service loaded active running OpenPegasus CIM Server

LOAD = Reflects whether the unit definition was properly loaded.

ACTIVE = The high- level unit activation state, i. e. generalization of SUB.

SUB = The low- level unit activation state, values depend on unit type.

4 6 l o a d e d u n i t s l i s t e d . P a s s - - a l l t o s e e l o a d e d b u t i n a c t i v e u n i t s , t o o .

To show all installed unit files use 'systemctl list- unit- files'

显示所有已安装服务单元示例如下：

```
# systemctl list-unit-files --type service
```

UNIT FILE STATE

abrt- ccpp. service enabled

abrt- oops. service enabled

abrt- vmcore. service enabled

abrt- xorg. service enabled

abrt-d. service enabled

...

wpa_supplicant. service disabled

ypbind. service disabled

208 unit files listed.

显示服务状态

显示服务状态命令：

```
# systemctl status name.service
```

表格 3-5 服务单元信息列表

文件	描述
Loaded	服务是否价值
Active	服务是否激活
Main PID	当前系统服务的 PID
Status	当前系统服务的附加信息
Process	相关进程附加信息
CGroup	相关 CGroup 附加信息

显示 gdm.service.服务状态示例：

```
# 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.
```

显示 gdm.service 服务启动前置依赖服务示例：

```
# systemctl list-dependencies --after gdm.service

gdm. service
├─dbus. socket
├─getty@tty1. service
├─livesys. service
├─plymouth- quit. service
├─system. slice
├─systemd- j ournald. socket
├─systemd- user- sessions. service
└─basic. target

[output truncated]
```

显示 gdm.service 服务启动后导服务示例：

```
# systemctl list-dependencies --before gdm.service

gdm. service
├─dracut- shutdown. service
├─graphical. target
│   ├─systemd- readahead- done. service
│   ├─systemd- readahead- done. timer
│   └─systemd- update- utmp- runlevel. service
└─shutdown. target
    ├─systemd- reboot. service
    └─final. target
        └─systemd- reboot. service
```

启动服务

启动服务命令：

```
# systemctl start name.service
```

启动 httpd 服务示例：

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

停止服务

停止服务命令：

```
# systemctl stop name.service
```

停止 httpd 服务示例：

```
# systemctl stop httpd.service
```

重启服务

重启服务命令：

```
# systemctl restart name.service
```

重启 httpd 服务示例：

```
# systemctl restart httpd.service
```

仅重启正在运行的服务命令：

```
# systemctl try-restart name.service
```

重载服务命令：

```
# systemctl reload name.service
```

重载 httpd 服务示例：

```
# systemctl reload httpd.service
```

启用服务

启用服务命令：

```
# systemctl enable name.service
```

重新启用服务命令：

```
# systemctl reenale name.service
```

启用 httpd 服务示例：

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

禁用服务

禁用服务命令：

```
# systemctl enable name.service
```

禁用 bluetooth 服务示例：

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

3.1.3 管理目标

启动级别（runlevel）是一个旧的概念。现在，systemd 引入了一个和启动级别功能相似又不同的概念——目标（target）。不像数字表示的启动级别，每个目标都有名字和独特的功能，并且能同时启用多个。一些目标继承其他目标的服务，并启动新服务。systemd 提供了一些模仿 sysvinit 启动级别的目标，仍可以使用旧的 telinit 启动级别命令切换。

启动级别 0、1、3、5、6 都被赋予特定用途，并且都对应一个 systemd 的目标。要实现用户自定义启动级别功能，可以以原有的启动级别为基础，创建一个新的目标 /etc/systemd/system/<新目标>（可以参考 /usr/lib/systemd/system/graphical.target），创建 /etc/systemd/system/<新目标>.wants 目录，向其中加入额外服务的链接（指向 /usr/lib/systemd/system/ 中的单元文件）。

表格 3-6 SysV 启动级别与 systemd 目标对比

运行级别	目标单元	描述
0	runlevel0.target, poweroff.target	中断系统（halt）
1	runlevel1.target,	单用户模式

	rescue.target	
2	runlevel2.target,	用户自定义启动级别，通常识别为级别 3。
3	runlevel3.target, multi-user.target	多用户，无图形界面。用户可以通过终端或网络登录。
4	runlevel4.target, multi-user.target	用户自定义启动级别，通常识别为级别 3。
5	runlevel5.target, graphical.target	多用户，图形界面。继承级别 3 的服务，并启动图形界面服务。
6	runlevel6.target, reboot.target	重启
emergency	emergency.target	急救模式（Emergency shell）

表格 3-7 SysV 初始化命令与 systemctl 对比

旧命令	新命令	描述
run level	systemctl list-units --type target	显示当前目标
telini trunlevel	systemctl isolate name.target	变更当前目标

3.1.3.1 查看默认目标

查看默认目标命令：

```
# systemctl get-default
```

以下为查看默认目标单元的示例：

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

3.1.3.2 查看当前目标

查看当前目标命令：

```
# systemctl list-units --type target
```

输出加载目标的单元，执行以下命令：


```
# systemctl list-units --type target --all
```

以下为显示当前所有激活目标的示例：

```
# systemctl list-units --type target

UNIT LOAD ACTIVE SUB DESCRIPTION
basic.target loaded active active Basic System
cryptsetup.target loaded active active Encrypted Volumes
getty.target loaded active active Login Prompts
graphical.target loaded active active Graphical Interface
local-fs-pre.target loaded active active Local File Systems ( Pre)
local-fs.target loaded active active Local File Systems
multi-user.target loaded active active Multi- User System
network.target loaded active active Network
paths.target loaded active active Paths
remote-fs.target loaded active active Remote File Systems
sockets.target loaded active active Sockets
sound.target loaded active active Sound Card
spice-vdagentd.target loaded active active Agent daemon for Spice
guests
swap.target loaded active active Swap
sysinit.target loaded active active System Initialization
time-sync.target loaded active active System Time Synchronized
timers.target loaded active active Timers

LOAD = Reflects whether the unit definition was properly loaded.
ACTIVE = The high- level unit activation state, i. e. generalization of
SUB.
SUB = The low- level unit activation state, values depend on unit
type.
```

```
17 loaded units listed. Pass --all to see loaded but inactive units,
too.
```

```
To show all installed unit files use 'systemctl list-unit-files'.
```

3.1.3.3 变更默认目标

变更默认目标命令：

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

以下变更多用户目标示例：

```
# systemctl set-default multi-user.target
rm '/etc/systemd/system/default.target'
ln -s '/usr/lib/systemd/system/multi-user.target'
'/etc/systemd/system/default.target'
```

3.1.3.4 变更当前目标

变更当前目标命令：

```
# systemctl isolate name.target
```

以下变更多用户目标示例：

```
# systemctl isolate multi-user.target
```

3.1.3.5 切换救援模式

`systemd.unit=rescue.target` 是一个设置基本系统和救援 shell 的特殊 target unit (与运行级 1 相似)；

进入救援模式命令：

```
# systemctl rescue
```

如果需要进入救援模式且不输出日志，输以下命令：

```
# systemctl --no-wall rescue
```

切换救援模式示例：

```
# systemctl rescue

Broadcast message from root@localhost on pts/0 ( Fri 2013- 10- 25 18:
23: 15

CEST) :

The system is going down to rescue mode NOW!
```

3.1.3.6 切换紧急模式

systemd.unit=emergency.target 与传递保留参数的 init=/bin/sh 给系统使系统从该状态启动相似。

进入紧急模式命令：

```
# systemctl emergency
```

如果需要进入紧急模式且不输出日志，输以下命令：

```
# systemctl --no-wall emergency
```

3.1.4 关闭暂停挂起系统

系统使用 systemctl 工具进行系统的关闭暂停挂起等操作，与以前电源管理对比如下：

表格 3-8 电源管理命令对比

旧命令	新命令	描述
halt	systemctl halt	关闭系统
poweroff	systemctl poweroff	退出系统并停止电源
reboot	systemctl reboot	重启
suspend	systemctl suspend	待机
hibernate	systemctl hibernate	休眠
hybridsleep	systemctl hybrid-sleep	混合休眠模式（同时休眠到硬盘并待机）

3.1.4.1 系统关机

使用 systemctl 命令：

退出系统并停止电源：

```
# systemctl poweroff
```

退出系统但不停止电源：

```
# systemctl halt
```

如果不需要输出日志，可以加上参数--no-wall。

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

使用 shutdown 命令：

某时某刻（hh:mm）退出系统并停止电源：

```
# shutdown --poweroff hh:mm
```

m 分钟后退出系统但不停止电源：

```
# shutdown --halt +m
```

取消关机命令：

```
# shutdown -c
```

3.1.4.2 重启系统

重启系统命令：

```
# systemctl reboot
```

如果不需要输出日志，可以加上参数--no-wall。

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

3.1.4.3 挂起系统

挂起系统命令：

```
# systemctl suspend
```

3.1.4.4 休眠系统

休眠系统命令：

```
# systemctl hibernate
```

混合休眠系统（同时休眠到硬盘并待机）命令

```
# systemctl hybrid-sleep
```

3.1.5 在远程机器上使用 systemd

连接远程机器使用 systemd 命令如下：

```
# systemctl --host user_name@host_name command
```

远程管理示例：

```
# systemctl -H root@server-01.example.com status httpd.service
>>>>>> systemd unit files - - update
root@server-01.example.com's password:
httpd.service - The Apache HTTP Server
Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled)
Active: active (running) since Fri 2013-11-01 13:58:56 CET; 2h
48min
ago
Main PID: 649
Status: "Total requests: 0; Current requests/sec: 0; Current
traffic: 0 B/sec"
CGroup: /system.slice/httpd.service
```

3.1.6 创建和修改 systemd 单元文件

3.1.6.1 单元文件介绍

单元文件通常包括三个部分：

【Unit】：通用配置项，包括该 unit 的基本信息。

【Unit type】：Unit 类型，不同类型定义可以参考 systemd 简介内容。

【Install】：包括需要被安装、启用和禁用的服务内容。

表格 3-9 【Unit】字段配置项说明

配置项	描述
Description	一些描述，显示给用户界面看的，可以是任何字符串，一般是关于服务的说明。
Documentation	指定参考文档的列表，以空格分开的 URI 形式，如 http://, https://, file:, info:,man，这是有顺序的，最好是先解释这个服务的目的是什么，然后是它是如何配置的，再然后是其它文件，这个选项可以多次指定，会将多行的合并，如果指定了一个空的，那么会重置此项，前的配置不在起作用。
After/Before	配置服务间的启动顺序，比如一个 foo.service 包含了一行 Before=bar.service，那么当他们同时启动时，bar.service 会等待 foo.service 启动完成后才启动。注意这个设置和 Requires= 的相互独立的，同时包含 After= 和 Requires= 也是常见的。此选项可以指定一次以上，这时是按顺序全部启动。
Requires	指定此服务依赖的其它服务，如果本服务被激活，那么 Requires 后面的服务也会被激活，反之，如果 Requires 后面的服务被停止或无法启动，则本服务也会停止。这个选项可以指定多次，那么就要求所有指定的服务都被激活。需要注意的是这个选项不影响启动或停止的顺序，启动顺序使用单句的 After= 和 Before= 来配置。例如，如果 foo.service 依赖 bar.service，但是只配置了 Requires= 而没有 After= 或 Before=，那么 foo.service 启动时会同时激活 foo.service 和 bar.service。通常使用 Wants= 代替 Requires= 是更好的选择，因为系统会更好的处理服务失败的情况。
Wants	相对弱化的 Requires=，这里列出的服务会被启动，但如果无法启动或无法添加到事务处理，并不影响本服务做为一个整体的启动。这是推荐的两个服务关联的方式。这种依赖也可以配置文件外，通过 .wants/ 目录添加，具体可以看上面的说明。
Conflicts	配置一个依赖冲突，如果配置了些项，那么，当一个服务启动时，或停

	止此处列出的服务，反过来，如果这里列出的服务启动，那么本服务就会停止，即后启动的才起作用。注意，此设置和 After= 和 Before= 是互相独立的。如果服务 A 和 B 冲突，且在 B 启动的时候同时启动，那么有可能会启动失败（两都都是必需的）或修改以修复它（两者之一或两都都不是必需的），后一种情况，会将不需要的依赖删除，或停止冲突。
--	---

表格 3-10 【Service】字段配置项

配置项	描述
Type	<p>设置进程的启动类型，必须是下列值之一：simple, forking, oneshot, dbus, notify, idle</p> <ul style="list-style-type: none">➤ 如果设为"simple"(设置了 ExecStart= 但未设置 BusName= 时的默认值)，那么表示 ExecStart= 所设定的进程就是该服务的主进程。➤ 如果此进程需要为其他进程提供服务，那么必须在该进程启动之前先建立好通信渠道(例如套接字)，以加快后继单元的启动速度。➤ "dbus"(设置了 ExecStart= 与 BusName= 时的默认值)与 "simple"类似，不同之处在于该进程需要在 D-Bus 上获得一个由 BusName= 指定的名称。systemd 将会在启动后继单元之前，首先确保该进程已经成功的获取了指定的 D-Bus 名称。设置为此类型相当于隐含的依赖于 dbus.socket 单元。➤ "oneshot"(未设置 ExecStart= 时的默认值)与"simple"类似，不同之处在于该进程必须在 systemd 启动后继单元之前退出。此种类型通常需要设置 RemainAfterExit= 选项。➤ 如果设为"forking"，那么表示 ExecStart= 所设定的进程将会在启动过程中使用 fork() 系统调用。这是传统 UNIX 守护进程的经典做法。也就是当所有的通信渠道都已建好、启动亦已成功之后，父进程将会退出，而子进程将作为该服务的主进程继续运行。对

	<p>于此种进程，建议同时设置 <code>PIDFile=</code> 选项，以帮助 <code>systemd</code> 准确定位该服务的主进程，进而加快后继单元的启动速度。</p> <ul style="list-style-type: none"> ➤ <code>"notify"</code>与<code>"simple"</code>类似，不同之处在于该进程将会在启动完成之后通过 <code>sd_notify(3)</code> 之类的接口发送一个通知消息。<code>systemd</code> 将会在启动后继单元之前，首先确保该进程已经成功的发送了这个消息。如果设置为此类型，那么 <code>NotifyAccess=</code> 将只能设置为<code>"all"</code>或者<code>"main"</code>(默认)。注意，目前 <code>Type=notify</code> 尚不能在 <code>PrivateNetwork=yes</code> 的情况下正常工作。 ➤ <code>"idle"</code>与<code>"simple"</code>类似，不同之处在于该进程将会被延迟到所有的操作都完成之后再执行。这样可以避免控制台上的状态信息与 <code>shell</code> 脚本的输出混杂在一起。
ExecStart	<p>在启动该服务时需要执行的命令行(命令+参数)。</p> <p>仅在设置了 <code>Type=oneshot</code> 的情况下，才可以设置任意个命令行(包括零个)，否则必须且只能设置一个命令行。</p> <p>多个命令行既可以在同一个 <code>ExecStart=</code> 中设置，也可以通过设置多个 <code>ExecStart=</code> 来达到相同的效果。</p> <p>如果设为一个空字符串，那么先前设置的所有命令行都将被清空。如果不设置任何 <code>ExecStart=</code> 指令，那么必须确保设置了 <code>RemainAfterExit=yes</code> 指令。</p> <p>命令行必须以一个绝对路径表示的可执行文件开始，并且其后的那些参数将依次作为<code>"argv[1] argv[2] ..."</code>传递给被执行的进程。</p> <p>如果在绝对路径前加上可选的<code>"@"</code>前缀，那么其后的那些参数将依次作为<code>"argv[0] argv[1] argv[2] ..."</code>传递给被执行的进程。</p> <p>如果在绝对路径前加上可选的<code>"-"</code>前缀，那么即使该进程以失败状态(例如非零的返回值或者出现异常)退出，也会被视为成功退出。可以同时使用<code>"-"</code>与<code>"@"</code>前缀，且顺序任意。</p>

	<p>如果设置了多个命令行，那么这些命令行将以其在单元文件中出现的顺序依次执行。</p> <p>如果某个无 "-" 前缀的命令行执行失败，那么剩余的命令行将不会被执行，同时该单元将变为失败(failed)状态。</p> <p>当未设置 <code>Type=forking</code> 时，这里设置的命令行所启动的进程将被视为该服务的主守护进程。</p>
ExecStop	<p>这是一个可选的指令，用于设置当该服务被要求停止时所执行的命令行。语法规则与 <code>ExecStart=</code> 完全相同。</p> <p>执行完此处设置的命令行之后，该服务所有剩余的进程将会根据 <code>KillMode=</code> 的设置被杀死(参见 <code>systemd.kill(5)</code> 手册)。</p> <p>如果未设置此选项，那么当此服务被停止时，该服务的所有进程都将会根据 <code>KillMode=</code> 的设置被立即全部杀死。</p> <p>与 <code>ExecReload=</code> 一样，也有一个特殊的环境变量 <code>\$MAINPID</code> 可以用于表示主进程的 PID</p> <p>一般来说，仅仅设置一个结束服务的命令，而不等待其完成，是不够的。</p> <p>因为当此处设置的命令执行完之后，剩余的进程会被 <code>SIGKILL</code> 信号立即杀死，这可能会导致数据丢失。</p> <p>因此，这里设置的命令必须是同步操作，而不能是异步操作。</p>
ExecReload	<p>这是一个可选的指令，用于设置当该服务被要求重新载入配置时所执行的命令行。语法规则与 <code>ExecStart=</code> 完全相同。</p> <p>另外，还有一个特殊的环境变量 <code>\$MAINPID</code> 可以用于表示主进程的 PID，例如可以这样使用：</p> <pre>/bin/kill -HUP \$MAINPID</pre>

	<p>注意，像上例那样，通过向守护进程发送复位信号，强制其重新加载配置文件，并不是一个好习惯。</p> <p>因为这是一个异步操作，所以不适用于需要按照特定顺序重新加载配置文件的服务。</p> <p>我们强烈建议将 <code>ExecReload=</code> 设置为一个能够确保重新加载配置文件的操作同步完成的命令行。</p>
Restart	<p>当服务进程正常退出、异常退出、被杀死、超时的时候，是否重新启动该服务。</p> <p>"服务进程"是指 <code>ExecStart=</code>, <code>ExecStartPre=</code>, <code>ExecStartPost=</code>, <code>ExecStop=</code>, <code>ExecStopPost=</code>, <code>ExecReload=</code> 中设置的进程。</p> <p>当进程是由于 <code>systemd</code> 的正常操作(例如 <code>systemctl stop restart</code>)而被停止时，该服务不会被重新启动。</p> <p>"超时"可以是看门狗的"keep-alive ping"超时，也可以是 <code>systemctl start reload stop</code> 操作超时。</p> <p>该选项可以取下列值之一： <code>no</code>, <code>on-success</code>, <code>on-failure</code>, <code>on-abnormal</code>, <code>on-watchdog</code>, <code>on-abort</code>, <code>always</code></p> <p>"no"(默认值)表示不会被重启。"always"表示会被无条件的重启。</p> <p>"on-success"表示仅在服务进程正常退出时重启，所谓"正常退出"是指：退出码为"0"，或者进程收到 <code>SIGHUP</code>, <code>SIGINT</code>, <code>SIGTERM</code>, <code>SIGPIPE</code> 信号并且退出码符合 <code>SuccessExitStatus=</code> 的设置。</p> <p>"on-failure"表示仅在服务进程异常退出时重启，所谓"异常退出"是指：退出码不为"0"，或者进程被强制杀死(包括"core dump"以及收到 <code>SIGHUP</code>, <code>SIGINT</code>, <code>SIGTERM</code>, <code>SIGPIPE</code> 之外的其他信号)，或者进程由于看门狗或者 <code>systemd</code> 的操作超时而杀死。</p>
RemainAfterExit	<p>可设为"yes"或"no"(默认值)，表示当该服务的所有进程全部退出之后，是否依然将此服务视为活动(active)状态。</p>

表格 3-11 【install】字段配置项

配置项	描述
Alias	在安装使用应该使用的额外名字（即别名）。名字必须和服务本身有同样的后缀（即同样的类型）。这个选项可以指定多次，所有的名字都起作用，当执行 <code>systemctl enable</code> 命令时，会建立相当的链接。
RequiredBy WantedBy	在 <code>.wants/</code> 或 <code>.requires/</code> 子目录中为服务建立相应的链接。这样做的效果是当列表中的服务启动，本服务也会启动。
Also	当此服务安装时同时需要安装的附加服务。 如果用户请求安装的服务中配置了此项，则 <code>systemctl enable</code> 命令执行时会自动安装本项所指定的服务。
DefaultInstance	表示 unit 启用时默认的实例

下面是 postfix.service 单元文件的示例：

```
[Unit]
Description=Postfix Mail Transport Agent
After=syslog.target network.target
Conflicts=sendmail.service exim.service

[Service]
Type=forking
PIDFile=/var/spool/postfix/pid/master.pid
EnvironmentFile=- /etc/sysconfig/network
ExecStartPre=- /usr/libexec/postfix/aliasesdb
ExecStartPre=- /usr/libexec/postfix/chroot-update
ExecStart=/usr/sbin/postfix start
ExecReload=/usr/sbin/postfix reload
ExecStop=/usr/sbin/postfix stop

[Install]
WantedBy=multi-user.target
```

这个示例中，【Unit】字段表述服务名称与依赖冲突信息。【Service】包括基本的服务信息。EnvironmentFile 表述预定义的该服务的环境变量，PIDFile 表示服务使用静态的PID。最后，【Install】字段显示依赖该服务的内容。

3.1.6.2 创建自定义单元文件

创建自定义单元文件的步骤：

- 1) 准备自定义服务的执行文件。

可执行文件可以是脚本，也可以是软件提供者的程序，如果需要，为自定义服务的主进程准备一个PID文件，一保证PID保持不变。另外还可能需要的配置环境变量的脚本，确保所以脚本都有可执行属性并且不需要交互。

- 2) 在/etc/systemd/system/目录创建单元文件，并且保证只能被 root 用户编辑：

```
# touch /etc/systemd/system/name.service
# chmod 664 /etc/systemd/system/name.service
```

文件不需要执行权限。

- 3) 打开 name.service 文件，添加服务配置，各种变量如何配置视所添加的服务类型而定，下面是一个依赖网络服务的配置例子：

```
[Unit]
Description=service_description
After=network.target

[Service]
ExecStart=path_to_executable
Type=forking
PIDFile=path_to_pidfile

[Install]
WantedBy=default.target
```

- 4) 通知 systemd 有个新服务添加：

```
# systemctl daemon-reload
# systemctl start name.service
```

创建 emacs.service 文件的例子：

1) 创建文件，并确保正确权限：

```
# touch /etc/systemd/system/emacs.service
# chmod 664 /etc/systemd/system/emacs.service
```

2) 添加配置信息：

```
[Unit]
Description=Emacs:the extensible, self-documenting text editor

[Service]
Type=forking
ExecStart=/usr/bin/emacs --daemon
ExecStop=/usr/bin/emacsclose --eval "(kill-emacs)"
Environment=SSH_AUTH_SOCK=%t/keyring/ssh
Restart=always

[Install]
WantedBy=default.target
```

3) 通知 systemd 并开启服务：

```
# systemctl daemon-reload
# systemctl start emacs.service
```

创建 sshd-second 服务的例子：

1) 拷贝 sshd_config 文件

```
# cp /etc/ssh/sshd_config /etc/ssh/sshd-second_config
```

- 2) 编辑 sshd-second_config 文件，添加 22220 的端口，和 PID 文件：

```
Port22220
```

```
PidFile/var/run/sshd-second.pid
```

如果还需要修改其他参数，请阅读帮助。

- 3) 拷贝单元文件：

```
#cp/usr/lib/systemd/system/sshd{,-second}.service
```

- 4) 编辑单元文件 sshd-second.service

修改描述字段

```
Description=OpenSSHserversecondinstancedaemon
```

添加 sshd.service 服务在 After 关键字之后：

```
After=syslog.targetnetwork.targetauditd.servicesshd.service
```

移除 sshdkey 创建：

```
ExecStartPre=/usr/sbin/sshd-keygen
```

移除这一行

在执行脚本里，添加第二 sshd 服务的配置文件：

```
ExecStart=/usr/sbin/sshd-D-f/etc/ssh/sshd-second_config$OPTIONS
```

修改后的 sshd-second.service 文件内容如下：

```
ExecStart=/usr/sbin/sshd-D-f/etc/ssh/sshd-second_config$OPTIONS
```

- 5) 如果使用 SELinux，添加 tcp 端口，负责 sshd-second 服务的端口就会被拒绝绑定：

```
#semanage port -a -tssh_port_t -p tcp 22220
```

- 6) 设置开机启动并测试：

```
#systemctl enable sshd-second.service
```

```
#ssh -p 22220 user@server
```

确保防火墙端口也开放。

3.1.6.3 SysV Init 脚本与 unit 文件转换

传统方式,Unix 和 Linux 服务(daemons)由 SysV 的 init 脚本启动.这些服务写成 Bourne Shell 脚本,通常放在在像/etc/rc.d/init.d/这样的目录里,调用时有一个或几个参数,比如 start, stop 或 restart, 用来控制服务的开始,停止和重启。开始服务包括调用守护程序的代码, 守护程序再 fork 一个后台进程。Shell 脚本很慢,难以阅读,冗长而脆弱。 尽管 shell 脚本极其灵活（毕竟，脚本本身就是代码）有些事儿很难拿脚本来完成，比如：安排并发执行的顺序，正确的监督进程或只是详尽的配置执行上下文。systemd 可以与脚本兼容，在安装所有守护进程的时候推荐安装原生的 systemd 服务文件。SysV 的启动脚本需要根据发行版的不同做出调整，systemd 的服务文件则兼容任何发行版中运行的 systemd。

接下来是一个简洁的入门手册：如何把 SysV 的 shell 脚本转换成一个 systemd 服务文件。理想情况下，上游软件应该在 tar 包里发行和安装 systemd 的服务文件。如果你已经根据入门手册成功的把某个上游软件 SysV 脚本转换成功了，一个不错的做法是把转换成功的文件作为补丁提交给那个上游软件。

作为例子,我们将把 ABRT 守护进程的 init 脚本转换成 systemd 的服务文件。ABRT 是自动 Bug 报告工具（Automatic Bug Reporting Tool）的缩写，比如，它提供收据 crash dumps 服务。它的 SysV 脚本如下：

```
#!/bin/bash

# Starts the abrt daemon

#

# chkconfig: 35 82 16

# description: Daemon to detect crashing apps

# processname: abrt

### BEGIN INIT INFO

# Provides: abrt
```

```
# Required-Start: $syslog $local_fs
# Required-Stop: $syslog $local_fs
# Default-Stop: 0 1 2 6
# Default-Start: 3 5
# Short-Description: start and stop abrt daemon
# Description: Listen to and dispatch crash events
#### END INIT INFO

# Source function library.
. /etc/rc.d/init.d/functions
ABRT_BIN="/usr/sbin/abrttd"
LOCK="/var/lock/subsys/abrttd"
OLD_LOCK="/var/lock/subsys/abrt"
RETVAL=0

#
# Set these variables if you are behind proxy
#
#export http_proxy=
#export https_proxy=

#
# See how we were called.
#

check() {
    # Check that we're a privileged user
```



```
[ "`id -u`" = 0 ] || exit 4

# Check if abrt is executable
test -x $ABRT_BIN || exit 5
}

start() {

    check

    # Check if it is already running
    if [ ! -f $LOCK ] && [ ! -f $OLD_LOCK ]; then
        echo -n "Starting abrt daemon: "
        daemon $ABRT_BIN
        RETVAL=$?
        [ $RETVAL -eq 0 ] && touch $LOCK
        echo
    fi
    return $RETVAL
}

stop() {

    check

    echo -n "Stopping abrt daemon: "
    killproc $ABRT_BIN
```

```
RETVAL=$?  
[ $RETVAL -eq 0 ] && rm -f $LOCK  
[ $RETVAL -eq 0 ] && rm -f $OLD_LOCK  
echo  
return $RETVAL  
}  
  
restart() {  
    stop  
    start  
}  
  
reload() {  
    restart  
}  
  
case "$1" in  
start)  
    start  
    ;;  
stop)  
    stop  
    ;;  
reload)  
    reload  
    ;;
```

```

force-reload)

    echo "$0: Unimplemented feature."

    RETVAL=3

    ;;

restart)

    restart

    ;;

condrestart)

    if [ -f $LOCK ]; then

        restart

    fi

    # update from older version

    if [ -f $OLD_LOCK ]; then

        restart

    fi

    ;;

status)

    status abrttd

    RETVAL=$?

    ;;

*)

    echo                $"Usage:                $0

{ start|stop|status|restart|condrestart|reload|force-reload}"

    RETVAL=2

esac

exit $RETVAL

```

转换脚本的第一步是阅读它的代码并从中提取有用信息。几乎所有的 init 脚

本中都有模板化的代码，通常这些代码就是从另一些脚本中拷贝过来的。下面开始抽取上面链接中脚本的有用信息。

- 文件中的一个描述句子是“发现崩溃程序的守护进程”。其他很多注释是多余描述，它们不怎么描述服务本身，却描述启动服务的脚本文件。`systemd` 服务也包含描述，描述的应该是服务内容而非服务文件。
- LSB 头[1] 包含服务的依赖关系信息，`systemd` 是基于 `socket` 激活进行设计的，通常不需要（或者需要很少量的）手动配置依赖关系。（对 `socket` 激活的详细阐述请见这篇博客。本例所依赖的`$syslog`（表明 `abrt` 需要用到 `syslog` 守护进程），只是个变量信息。头部列出的另一依赖关系（`$local_fs`）对 `systemd` 来说是冗余项，因为系统服务开始时总是要求本地文件系统可用。
- LSB 头建议服务应该运行在运行级别 3（多用户）或 5（图形界面）
- 守护进程的二进制文件是`/usr/sbin/abrt`

以上是脚本中的重要信息。115 行 `shell` 脚本的其他部分不是模板化的东西就是冗余代码：用来应付同步和顺序启动（比如，锁文件相关代码），或是输出状态信息（比如调用 `echo` 语句的代码），或者进行参数分析（比如 `case` 块儿）。

从上述抽取出的信息出发，现在可以写新的 `systemd` 服务文件了。

```
[Unit]

Description=Daemon to detect crashing apps
After=syslog.target


[Service]

ExecStart=/usr/sbin/abrt

Type=forking


[Install]

WantedBy=multi-user.target
```

这一文件的简单解释：`[Unit]` 节包含服务的通用信息. `systemd` 不止管理系统

服务，而且管理设备，mount 点，计时器，和其它系统组件。systemd 中这些对象的通用名称是一个 unit，[Unit]节中包含的信息不但可能用于服务，也可能用于 systemd 维护的其它 unit 类型。本例中进行如下的 unit 设置：设置描述项，并配置守护进程在 Syslog 后启动[2]，很像原 init 脚本中的 LSB 头。针对 Syslog 依赖关系，在 systemd 的 Unit 节创建了一个依赖类型 After=syslog.target。syslog.target 为 systemd 中专用的目标 unit。它是 syslog 的标准化命名，对标准化命名的更多信息可见 systemd.special(7)。注意键入 After=来创建的依赖关系只是一种建议顺序，它本身没有强制性：当 abrted 启动时并不能引起 syslog 启动--这正是我们想要的，因为 abrted 实际上在没有 syslog 时也能正常工作。但是，如果两者都启动了（通常情况如此），那么这一依赖关系决定了启动顺序。

接下来一节是[Service]用来配置服务本身的信息。它包含只用于服务本身的设置，而非其它 systemd 所维护的 units（mount 点，设备，定时器等）。这里用了两个设置：ExecStart=定位服务启动时二进制文件的路径。Type=配置服务启动完成后如何通知 init 系统。由于传统的 Unix 守护进程 fork 完毕并完成后台守护进程初始化后返回父进程，在这里设置 forking 类型。这告诉 systemd 等到启动的二进制文件返回后再考虑守护进程后面仍然运行的进程

最后一节是[Install]。用于对如何安装给出建议，比如，服务在何种环境下启动，如何被触发。本例中服务将在 multi-user.target unit 激活后启动。multi-user.target 是一个专用 unit，基本上相当于 SysV 的运行级 3[3]。WantedBy= 设置项对运行时的守护进程没啥作用。systemd 有个推荐的激活服务的命令，即 systemctl enable，它会读取 WantedBy=设置项。此命令保证只要需要 multi-user.target，制定的这个小服务就能自动启用。正常的启动都需要 multi-user.target[4]。

现在有了个最小的能工作的 systemd 服务文件，把它拷贝到 /etc/systemd/system/abrted.service 然后执行 systemctl daemon-reload。这会使 systemd 发现它，现在可以启动服务：systemctl start abrted.service。 可以查看服务状态：systemctl status abrted.service。 可以停止服务：systemctl stop abrted.service。以及在系统启动时自动激活：systemctl enable abrted.service。

3.1.6.4 修改已经存在的单元文件

systemd 单元配置文件默认保存在/usr/lib/systemd/system/目录，系统管理员不建议直接修改这个目录下的文件，自定义的文件在/etc/systemd/system/目录下，如果有扩展的需求，可以使用以下方案：

创建一个目录/etc/systemd/system/unit.d/，这个是最推荐的一种方式，可以参考初始的单元文件，通过附件配置文件来扩展默认的配置，对默认单元文件的升级会被自动升级和应用。

从/usr/lib/systemd/system/拷贝一份原始配置文件到/etc/systemd/system/，然后修改。复制的版本会覆盖原始配置，这种方式不能增加附件的配置包，用于不需要附加功能的场景。

如果需要恢复到默认的配置文件，只需要删除/etc/systemd/system/下的配置文件就可以了，不需要重启机器，使用如下命令应用改变就可以：

```
# systemctl restart name.service
```

扩展默认单元配置文件

为了扩展默认的单元文件配置，需要先在/etc/systemd/system/下创建一个目录，用 root 执行类似下面的命令：

```
# mkdir/etc/systemd/system/name.service.d
```

在刚才创建的目录之下创建配置文件，必须以.conf 文件结尾。

例如创建一个自定义的依赖文件，内容如下：

```
[Unit]

Requires=new_dependency

After=new_dependency
```

另外一个例子，可以配置重启的时候，在主进程退出后 30 秒在重启，配置例子如下：

```
[Unit]

Requires=new_dependency
```

```
After=new_dependency
```

另外一个例子，可以配置重启的时候，在主进程退出后 30 秒在重启，配置例子如下：

```
[Service]
Restart=always
RestartSec=30
```

推荐每次只产生一个小文件，每个文件只聚焦完善一个功能，这样配置文件很容易被移除或者链接到其他服务对的配置目录中。

为了应用刚才的修改，使用 root 执行以下操作：

```
systemctl daemon-reload
systemctl restart name.service
```

例子：扩展 httpd.service 服务配置

为了使 httpd 服务启动的时候执行用户自定义的脚本，需要修改 httpd 的单元配置文件，执行以下几步操作，首先创建一个自定义文件的目录及自定义文件：

```
#mkdir/etc/systemd/system/httpd.service.d
#touch/etc/systemd/system/httpd.service.d/custom_script.conf
```

假设自定义文件位置在 /usr/local/bin/custom.sh，将这个信息添加到 custom_script.conf 自定义脚本中：

```
[Service]
ExecStartPost=/usr/local/bin/custom.sh
```

应用更改：

```
#systemctl daemon-reload
#systemctl restart httpd.service
```

3.1.6.5 单元文件实例管理

在现实中，往往有一些应用需要被复制多份运行，例如在一个负载均衡实例

后面运行的多个相同的服务实例。但是按照之前的例子，每个服务都需要一个单独的 Unit 文件，这样复制多份相同文件的做显然不便于服务的管理。为此 Systemd 定义了一种特殊的 Service Unit 文件，称为 Unit 模板。

模板文件的主要特点是，文件名以@符号结尾，而启动的时候指定的 Unit 名称为模板名称附加一个参数字符串。例如，将之前的例子第二个 Unit 文件修改为可以用于启动多个实例的模板。

1) 首先修改文件名，添加一个@符号

例如原来的文件名是 `apache.service`，那么可以将它修改为 `apache@.service`，这样做的目的是表面这个文件是一个模板文件。而在服务启动时可以在@后面放置一个用于区分服务实例的附加字符串参数，通常这个参数会使用监听的端口号或使用的控制台 TTY 编号等。例如 “`systemctl start apache@8080.service`”。

2) 然后修改 Unit 文件内容

Unit 文件中可以获取服务启动时的附加参数，因此通常需要修改 Unit 文件中不应固定的部分，例如服务监听的 IP 和端口，替换为从附加参数中获取。

```
[Unit]
Description=My Advanced Service Template
After=etcd.servicedocker.service

[Service]
TimeoutStartSec=0

ExecStartPre=-/usr/bin/docker kill apache%i
ExecStartPre=-/usr/bin/docker rm apache%i
ExecStartPre=/usr/bin/docker pull coreos/apache
ExecStart=/usr/bin/docker run --name apache%i -p %i:80 coreos/apache
/usr/sbin/apache2ctl -D FOREGROUND

ExecStartPost=/usr/bin/etcdctl set /domains/example.com/%H:%i
running

ExecStop=/usr/bin/docker stop apache1

ExecStopPost=/usr/bin/etcdctl rm /domains/example.com/%H:%i
```


[Install]

WantedBy=multi-user.target

仔细观察一下变化了的地方，上面使用到了占位符 %H 和 %i，常用的占位符有 6 种，这些占位符会在 Unit 启动时被实际的值动态的替换掉。

表格 3-12 占位符说明

占位符	作用
%n	完整的 Unit 文件名字，包括 .service 后缀名
%m	实际运行的节点的 Machine ID，适合用来做 Etcd 路径的一部分，例如 /machines/%m/units
%b	作用类似 Machine ID，但这个值每次节点重启都会改变，称为 Boot ID
%H	实际运行节点的主机名
%p	Unit 文件名中在 @ 符号之前的部分，不包括 @ 符号
%i	Unit 文件名中在 @ 符号之后的部分，不包括 @ 符号和 .service 后缀名

3) 启动 Unit 模板的服务实例

模板服务的启动对于 Systemd 和 Fleet 大致相同。

Systemd 的情况略简单一些，只需要运行时加上后缀参数。例如 “systemctl start apache@8080.service”。Systemd 首先会在其特定的目录下寻找名为 apache@8080.service 的文件，如果没有找到，而文件名中包含@字符，它就会尝试去掉后缀参数匹配模板文件。例如没有找到 apache@8080.service，那么 Systemd 会找到 apache@.service，并将它通过模板文件中实例化。

3.2 OpenSSH

SSH (Secure Shell) 是一个使用客户端-服务端架构, 使得两个系统之间的安全通信变得容易的协议, 它能够让用户远程登录到服务端主机系统中。和其它诸如 FTP 或者 Telnet 的远程通信协议不同, SSH 加密了登录会话, 致使入侵者难以通过连接获取未加密的密码。

ssh 程序被设计用于替换诸如 telnet 或者 rsh, 这些比较旧的、安全性不高的、用来登录远程主机系统的终端应用程序。与其相关的一个叫做 scp 的程序, 用于替换诸如 rcp 这样用来在主机之间复制文件的老程序。因为这些老旧的应用程序不会加密在客户端和服务端之间传递的密码, 所以应该尽可能地避免使用它们。使用安全方法登录远程系统, 能够同时降低客户端系统和远程主机系统的风险。

中标麒麟高级服务器操作系统包含了通用的 OpenSSH 安装包——openssh, 以及 OpenSSH 服务端安装包——openssh-server 和 OpenSSH 客户端安装包——openssh-clients。注意, OpenSSH 安装包依赖于 OpenSSL 的安装包 openssl-lib, 这个包安装了几个重要的加密库, 使得 OpenSSH 能够提供加密通信。

3.2.1 SSH 协议

3.2.1.1 为什么使用 SSH?

潜在的入侵者有许多可以使用的工具, 能够让他们中断、拦截和重新路由网络流量, 从而试图获取对一个系统的访问权限。一般来说, 这些威胁可以分为以下几类:

1) 拦截两个系统之间的通信

攻击者可能位于通信双方所在网络中的某处, 复制他们之间传递的任何信息。他可能会拦截并保留信息, 或者修改信息后将其发送给计划中的接收者。

这种攻击通常是通过使用数据包嗅探器来进行的, 数据包嗅探器是一种非常常见的网络工具, 可以用来捕获网络流中的数据包, 并分析其内容。

2) 冒充特定主机

攻击者的系统被配置来冒充传送的预期接收者。如果攻击者的策略成功

了，用户系统将不会发现它其实是在跟一个错误的主机进行通信。

这种攻击可以通过使用一种名为“DNS 缓存投毒”的技术，或者通过所谓的“IP 欺骗”来实现。在第一种示例中，入侵者使用一台被攻破的 DNS 服务器来将客户系统指向一台恶意复制主机。在第二种示例中，入侵者发送伪造的网络数据包，让这个数据包看起来好像是从一台可信任的主机发出的。

这些技术都能够拦截可能存在的敏感信息，而且如果这些拦截是出于怀有敌意的原因，那么结果将是灾难性的。如果使用 SSH 来进行远程登录和文件复制，那么就能够极大地减少这些安全威胁。这是因为 SSH 客户端和服务端使用数字签名来验证身份。此外，客户端和服务端系统之间的所有通信都是加密的。不管尝试在通信的哪一方进行身份欺骗都是行不通的，因为每一个数据包都是使用一个只有本地系统和远程系统才知道的密钥来加密的。

3.2.1.2 主要特性

SSH 协议提供了以下保护措施：

1) 无法伪装预期的服务端

在初始连接之后，客户端能够验证它当前所连接的服务端的确是它之前所连接过的同一个服务端。

2) 无法捕获认证信息

客户端使用强 128 位加密算法来将它的认证信息传递给服务端。

3) 无法拦截通信

在一个会话中所有发送和接收的数据都是使用 128 位加密算法加密的，使得拦截到的传输内容要解密阅读是极度困难的。

此外，SSH 协议还提供了以下选项：

1) 提供了在网络上使用图形化应用程序的安全手段

通过使用名为“X11 转发”的技术，客户端能够从服务端转发 X11（X 窗口系统）应用程序。

2) 提供了一种保护其它不安全协议的方式

SSH 协议对它发送和接收的一切进行加密。通过使用名为“端口转发”的技术，SSH 服务端可以成为一个保护其它不安全协议（例如 POP）的导管，

从而增加整体系统和数据的安全性。

3) 可以用来创建安全通道

OpenSSH 服务端和客户端可以被配置来为服务端和客户端机器之间的网络流，创建一个类似于虚拟专用网络的隧道。

4) 支持 Kerberos 认证

OpenSSH 服务端和客户端可以被配置为使用 Kerberos 网络认证协议的 GSSAPI（通用安全服务应用程序编程接口）实现来进行认证。

3.2.1.3 协议版本

SSH 目前存在两个版本：版本 1 和较新的版本 2。中标麒麟高级服务器操作系统中的 OpenSSH 套件使用 SSH 版本 2，该版本具有增强的密钥交换算法，不易受到版本 1 中已知漏洞的攻击。然而，为了兼容性的原因，OpenSSH 套件同样也支持版本 1 的连接。

重要说明：

为了确保您的连接的最佳安全性，建议您只要可能就使用只兼容 SSH 版本 2 的服务端和客户端。

3.2.1.4 SSH 连接的事件序列

以下一系列事件可以保护两个主机之间的 SSH 通信的完整性。

- 1) 进行加密握手，以便客户端能够验证它正在跟正确的服务端进行通信。
- 2) 采用对称加密算法对客户端和远程主机之间的连接的传输层进行加密。
- 3) 客户端向服务端进行自我认证。
- 4) 客户端通过加密连接和远程主机进行交互。

3.2.1.4.1 传输层

传输层的主要角色是确保两个主机之间的通信是安全可靠的，包括在认证的时候以及在随后的通信期间。传输层通过对数据进行加密和解密处理，以及通过提供对数据包发送和接收时的完整性保护，来实现这一目标。传输层也提供压缩、加速信息传输的功能。

SSH 客户端联系服务端时，将进行密钥交换，使得两个系统之间能够正确地构建传输层。密钥交换时的步骤如下：

- 1) 交换密钥
- 2) 确定公钥加密算法
- 3) 确定对称加密算法
- 4) 确定消息认证算法
- 5) 确定哈希算法

在密钥交换期间，服务端用一个唯一的主机密钥向客户端表明自己的身份。如果客户端以前从未和这个特定的服务端通信过，服务端的主机密钥对于客户端来说是未知的，客户端将不会连接。OpenSSH 通过接受服务端的主机密钥来规避这个问题。用户知晓，并且新的主机密钥已经被接受和验证之后，继续后续过程。在之后的连接中，客户端会用已保存的版本来检查服务端的主机密钥，以确保客户端确实是在和预期的服务端通信。如果在将来主机密钥发生了变化，用户必须在连接之前删除客户端已经保存的版本。

重要说明：

攻击者可能会在最初的联系阶段伪装成 SSH 服务端，因为本地系统并不知道预期的服务端和攻击者设置的假服务端之间有什么区别。为了防止这样的事情发生，请在第一次连接或者主机密钥不匹配时，联系服务端管理员以验证 SSH 服务端的真实完整性。

SSH 被设计成几乎可以和任何类型的公钥算法或者编码格式兼容。在最初的密钥交换创建了一个用于交换的哈希值和一个共享的密值后，两个系统立即开始计算新的密钥和算法，以保护将在连接上发送的认证和数据。

在使用指定的密钥和算法传输一定量（准确的量取决于 SSH 实现）的数据之后，将会发生又一次密钥交换，生成另一套哈希值和一个新的共享密值。即使攻击者能够确定哈希值和共享密值，这一信息也仅在非常有限的一段时间内有用。

3.2.1.4.2 认证

一旦传输层构建好一个安全隧道在两个系统之间传递信息，服务端告知客户端其所支持的不同的认证方法，例如使用一个私钥编码的签名，或者输入一个密码。然后客户端尝试使用所支持的其中一种方法向服务端进行认证。

SSH 服务端和客户端可以配置使用不同类型的认证方式,让各方都具有最佳的控制度。服务端可以决定基于其安全模型,它可以支持哪些加密方法;客户端可以从可用的选项中选择尝试认证方法的顺序。

3.2.1.4.3 通道

在 SSH 传输层完成一次成功的认证之后,将会通过被称为“多路技术”(多路复用连接由多个信号组成,这些信号通过一个共享的、通用的介质进行发送。对 SSH 来说,不同的通道都在一个共同的安全连接中发送)的一种技术打开多重通道。每一条通道负责处理不同的终端会话和转发 X11 会话。

不论是客户端还是服务端都可以创建新的通道。每一个通道将在连接的两端被赋予一个编号。当客户端尝试打开一个新的通道时,客户端把请求和通道编号一起发送出去。这些信息被服务端保存起来,用于将通信导向对应的通道。如此一来,不同类型的会话不会相互影响,并且当一个指定的会话结束之后,关闭它的通道不会中断主要的 SSH 连接。

通道也可以支持流控制,可以让通道以一种有秩序的方式发送和接收数据。以这种方式,只有当客户端接收到通道已经打开的消息,数据才会通过通道发送。

客户端和服务端自动协商每条通道的特征,取决于客户端请求的服务类型以及用户连接到网络的方式。这样可以在不必改变协议的基础设施的情况下,为处理不同类型的远程连接带来很大的灵活性。

3.2.2 配置 OpenSSH

3.2.2.1 配置文件

配置文件有两个不同的系列,一系列用于客户端程序(例如 ssh、scp 和 sftp),另外一系列用于服务端(sshd 守护进程)。

系统范围的 SSH 配置信息保存在/etc/ssh/目录下,详见表格 3-13 系统范围的配置文件。特定用户的 SSH 配置信息保存在~/.ssh/目录下(该目录在特定用户的家目录里),详见表格 3-14 特定用户的配置文件。

表格 3-13 系统范围的配置文件

文件	描述
/etc/ssh/moduli	包含了 Diffie-Hellman 密钥交换需要使用的 Diffie-Hellman 组，Diffie-Hellman 密钥交换对于构建安全的传输层是关键。当密钥在一个 SSH 会话的最初阶段被交换的时候，一个共享的密值被创建，该密值无法由任何单独的一方所确定。这个密值随后被用来提供主机认证。
/etc/ssh/ssh_config	默认的 SSH 客户端配置文件。注意，如果 ~/.ssh/config 存在的话，该文件的配置将被 ~/.ssh/config 覆盖。
/etc/ssh/sshd_config	sshd 守护进程的配置文件。
/etc/ssh/ssh_host_ecdsa_key	sshd 守护进程所使用的 ECDSA 私钥。
/etc/ssh/ssh_host_ecdsa_key.pub	sshd 守护进程所使用的 ECDSA 公钥。
/etc/ssh/ssh_host_key	SSH 协议版本 1 的 sshd 守护进程所使用的 RSA 私钥。
/etc/ssh/ssh_host_key.pub	SSH 协议版本 1 的 sshd 守护进程所使用的 RSA 公钥。
/etc/ssh/ssh_host_rsa_key	SSH 协议版本 2 的 sshd 守护进程所使用的 RSA 私钥。
/etc/ssh/ssh_host_rsa_key.pub	SSH 协议版本 2 的 sshd 守护进程所使用的 RSA 公钥。
/etc/pam.d/sshd	sshd 守护进程的 PAM 配置文件。
/etc/sysconfig/sshd	sshd 服务的配置文件。

表格 3-14 特定用户的配置文件

文件	描述
~/.ssh/authorized_keys	为服务端保留一个授权的公钥列表。当客户端连接到服务端时，服务端通过检查保存在该文件中的它的签名公钥来认证客户端。
~/.ssh/id_ecdsa	包含用户的 ECDSA 私钥。
~/.ssh/id_ecdsa.pub	用户的 ECDSA 公钥。
~/.ssh/id_rsa	SSH 协议版本 2 的 ssh 所使用的 RSA 私钥。
~/.ssh/id_rsa.pub	SSH 协议版本 2 的 ssh 所使用的 RSA 公钥。
~/.ssh/identity	SSH 协议版本 1 的 ssh 所使用的 RSA 私钥。
~/.ssh/identity.pub	SSH 协议版本 1 的 ssh 所使用的 RSA 公钥。
~/.ssh/known_hosts	包含用户访问的 SSH 服务端的主机密钥。该文件对于确保 SSH 客户端正在连接正确的 SSH 服务端是很重要的。

获取有关 SSH 配置文件中所使用的各种指令的信息，请参考 ssh_config(5) 和 sshd_config(5)手册页面。

3.2.2.2 启动 OpenSSH 服务端

您必须安装 openssh-server 包之后才能运行 OpenSSH 服务端（参见 2.2.2.4 安装软件包。了解如何在中标麒麟高级服务器操作系统 V7 中安装新的包）。

要在当前会话中启动 sshd 守护进程，请以 root 用户在 shell 命令行提示符下输入以下命令：

```
~]# systemctl start sshd . servi ce
```

要在当前会话中停止运行 sshd 守护进程，请以 root 用户在 shell 命令行提示符下输入以下命令：

```
~]# systemctl stop sshd . servi ce
```

如果您想在系统启动时自动启动守护进程，请以 root 用户在 shell 命令行提

示符下输入以下命令：

```
~]# systemctl enable sshd . service
ln -s '/usr/lib/systemd/system/sshd.service'
/etc/systemd/system/multiuser.
target.wants/sshd.service'
```

获取如何在中标麒麟高级服务器操作系统中管理系统服务的信息，请参见：

3.1 使用 systemd 管理系统服务。

注意，如果您重装了系统，将会创建一系列新的身份密钥。这将导致所有在该系统重装前使用 OpenSSH 工具连接过该系统的客户端看到以下信息：

```
@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @

@ WARNING: REMOTE HOST IDENTIFICATION HAS
CHANGED! @

@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @

IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING
NASTY!

Someone could be eavesdropping on you right now (man-in-the-middle
attack)!

It is also possible that the RSA host key has just been changed.
```

为了避免这种情况的发生，你可以备份/etc/ssh/目录下的相关文件（参见表格 3-13 系统范围的配置文件获取文件的完整列表），并在重装系统之后恢复这些文件。

3.2.2.3 使用 SSH 进行远程连接

为了让 SSH 真正发挥作用，应该禁止使用不安全的连接协议。否则，用户

的密码可能在一个会话中使用 SSH 时被保护得很好，但是却在之后使用 Telnet 登录时被捕获了。需要禁用的服务包括 telnet、rsh、rlogin 和 vsftpd。

获取如何配置 vsftpd 服务的信息，请参见 4.4.2FTP。要了解如何在中标麒麟高级服务器操作系统 V7 中管理系统服务，请参见 3.1 使用 systemd 管理系统服务。

3.2.2.4 使用基于密钥的认证

为了更进一步的提高系统安全，可以生成 SSH 密钥对，然后强制使用基于密钥的认证，并禁用密码认证。要这样做，请在 vi 或者 nano 等文本编辑器中打开/etc/ssh/sshd_config 配置文件，并将 PasswordAuthentication 选项修改为如下内容：

```
PasswordAuthentication no
```

如果您不是在一个新的默认安装的系统中进行操作，请检查配置文件确保没有设置 PubkeyAuthentication no 选项。如果是远程连接上的，而不是使用的控制台或者带外访问，建议在禁用密码认证之前先测试一下基于密钥的登录过程。

为了能够使用 ssh、scp 或者 sftp 从一个客户端机器连接到服务端，请按照以下步骤生成一个授权密钥对。注意，这些密钥必须针对每个用户分别生成。

中标麒麟高级服务器操作系统 V7 默认使用版本 2 的 SSH 协议和 RSA 密钥（参见 3.2.1.3 协议版本获取更多信息）。

重要说明：

如果您以 root 的身份完成了步骤，那么只有 root 用户能够使用这些密钥。

备注：

如果您要重装您的系统，又想保留之前生成的密钥对，请备份 ~/.ssh/ 目录。在重装后，将其复制回您的家目录。这一过程需要让系统上的所有用户执行，包括 root 用户。

3.2.2.5 生成密钥对

要生成 SSH 协议版本 2 的 RSA 密钥对，请按以下步骤操作：

- 1) 在 shell 命令行提示符下输入以下命令生成 RSA 密钥对：

```
~]$ ssh-keygen -t rsa
```

Generating public/private rsa key pair.

Enter file in which to save the key (/home/USER/.ssh/id_rsa):

- 2) 按回车键确认新创建的密钥的默认路径，即~/.ssh/id_rsa。
- 3) 输入一个口令，并且在提示确认的时候再次输入该口令。为了安全起见，请不要使用和您登录您的账户相同的密码。

之后，您会看到如下所示的类似信息：

```
Your identification has been saved in /home/USER/.ssh/id_rsa.
Your public key has been saved in /home/USER/.ssh/id_rsa.pub.
The key fingerprint is:
e7:97:c7:e2:0e:f9:0e:fc:c4:d7:cb:e5:31:11:92:14
USER@ penguin.example.com
The key's randomart image is:
+--[ RSA 2048]-----+
|           E. |
|          .. |
|         o . |
|          . .|
|         S .  .|
|        + o o ..|
|       * * +oo|
|        O +..=|
|       o* o.|
+-----+
```

- 4) 默认情况下，将~/.ssh/目录的权限设置为 `rwX-----` 或者以八进制标注表示的 `700`。这是为了确保只有对应的用户 `USER` 能够查看其内容。如果有必要，可以使用以下命令来进行确认：

```
~]$ ls -ld ~/.ssh
```

```
drwx-----. 2 USER USER 54 Nov 25 16:56 /home/USER/.ssh/
```

- 5) 使用以下格式的命令，将公钥复制到一台远程机器上：

```
ssh-copy-id user@hostname
```

如果公钥尚未安装的话，该命令会复制最近一次修改的`~/.ssh/id*.pub` 公钥。
可选的，您也可以指定公钥的文件名：

```
ssh-copy-id -i ~/.ssh/id_rsa.pub user@hostname
```

该命令会将`~/.ssh/id_rsa.pub` 的内容复制到您想连接的机器的
`~/.ssh/authorized_keys` 文件中。如果`authorized_keys` 文件已经存在了，密钥将会
追加到该文件的末尾。

要生成 SSH 协议版本 2 的 ECDSA 密钥对，请按以下步骤操作：

- 1) 在 shell 命令行提示符下输入以下命令生成 ECDSA 密钥对：

```
~]$ ssh-keygen -t ecdsa
```

```
Generating public/private ecdsa key pair.
```

```
Enter file in which to save the key (/home/USER/.ssh/id_ecdsa):
```

- 2) 按回车键确认新创建的密钥的默认路径，即`~/.ssh/id_ecdsa`。
3) 输入一个口令，并且在提示确认的时候再次输入该口令。为了安全起见，
请不要使用和您登录您的账户相同的密码。

之后，您会看到如下所示的类似信息：

```
Your identification has been saved in /home/USER/.ssh/id_ecdsa.
```

```
Your public key has been saved in /home/USER/.ssh/id_ecdsa.pub.
```

```
The key fingerprint is:
```

```
fd:1d:ca:10:52:96:21:43:7e:bd:4c:fc:5b:35:6b:63
```

```
USER@ penguin.example.com
```

The key's randomart image is:

```
+--[ECDSA 256]---+
|      .+ +o      |
|      .  =.o      |
|      o o +   ..|
|      + + o   +|
|      S o o oE.|
|      + oo+.|
|      + o   |
|              |
|              |
+-----+
```

- 4) 默认情况下，将~/.ssh/目录的权限设置为 `rwX-----` 或者以八进制标注表示的 `700`。这是为了确保只有对应的用户 **USER** 能够查看其内容。如果有必要，可以使用以下命令来进行确认：

```
~]$ ls -ld ~/.ssh
drwx-----. 2 USER USER 54 Nov 25 16:56 /home/USER/.ssh/
```

- 5) 使用以下格式的命令，将公钥复制到一台远程机器上：

```
ssh-copy-id user@hostname
```

如果公钥尚未安装的话，该命令会复制最近一次修改的~/.ssh/id*.pub 公钥。可选的，您也可以指定公钥的文件名：

```
ssh-copy-id -i ~/.ssh/id_ecdsa.pub user@hostname
```

该命令会将 ~/.ssh/id_ecdsa.pub 的内容复制到您想连接的机器的 ~/.ssh/authorized_keys 文件中。如果 authorized_keys 文件已经存在了，密钥将会追加到该文件的末尾。

请参见 3.2.2.6 配置 ssh-agent 了解如何设置您的系统记住口令。

重要说明:

私钥仅供您个人使用，绝不要把它给任何人，这一点是非常重要的。

3.2.2.6 配置 ssh-agent

要保存您的口令，以便在您初始化一个到远程机器的连接时，不用每次都输入口令，您可以使用 `ssh-agent` 认证代理。如果您正在使用 **GNOME**，您可以配置它在您登录时提示输入您的口令，并在整个会话中记住该口令。否则，您可以为某个确定的 `shell` 终端保存口令。

要在您的 **GNOME** 会话期间保存您的口令，请按以下步骤进行操作：

- 1) 确保您已经安装了 `openssh-askpass` 包。如果没有安装，请参见 2.2.2.4 安装软件包。了解如何在中标麒麟高级服务器操作系统中安装新的包。
- 2) 按下【应用程序】右侧【**Super 键**】（Windows 徽标键）【**进入活动概览**】页面，在搜索栏输入【**Startup Applications**】并按回车键，将会弹出【**Startup Application Preferences**】工具。默认情况下，将会显示包含了一个可用的启动程序列表的标签页。**Super** 键可能存在很多伪装，这取决于键盘和其它硬件，但它通常要么是 Windows 徽标键，要么是 **Command** 键，并且一般是在空格的左边。



图 3-1 Startup Applications Preferences

- 3) 点击右侧的【添加】按钮，然后在【命令】文本框中输入/usr/bin/ssh-add。

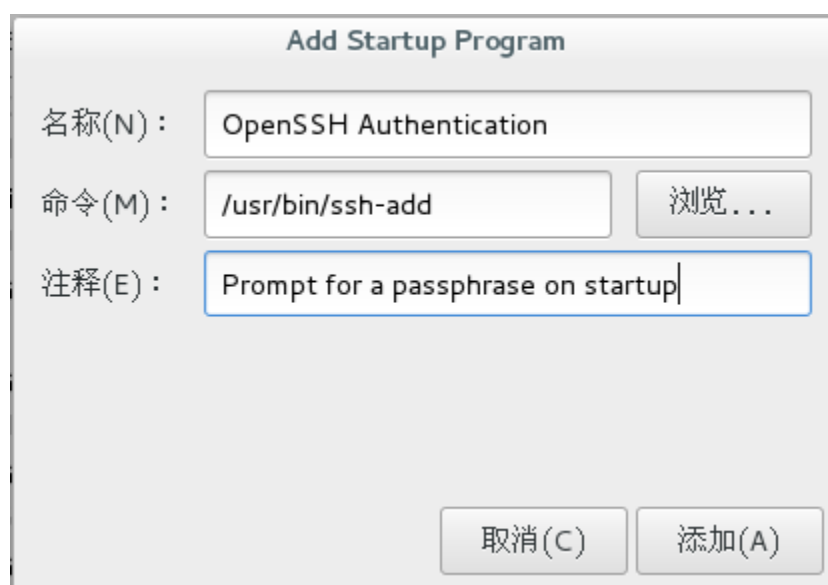


图 3-2 添加新的程序

- 4) 点击【添加】按钮，并确保新添加的条目旁边的复选框是选中的。

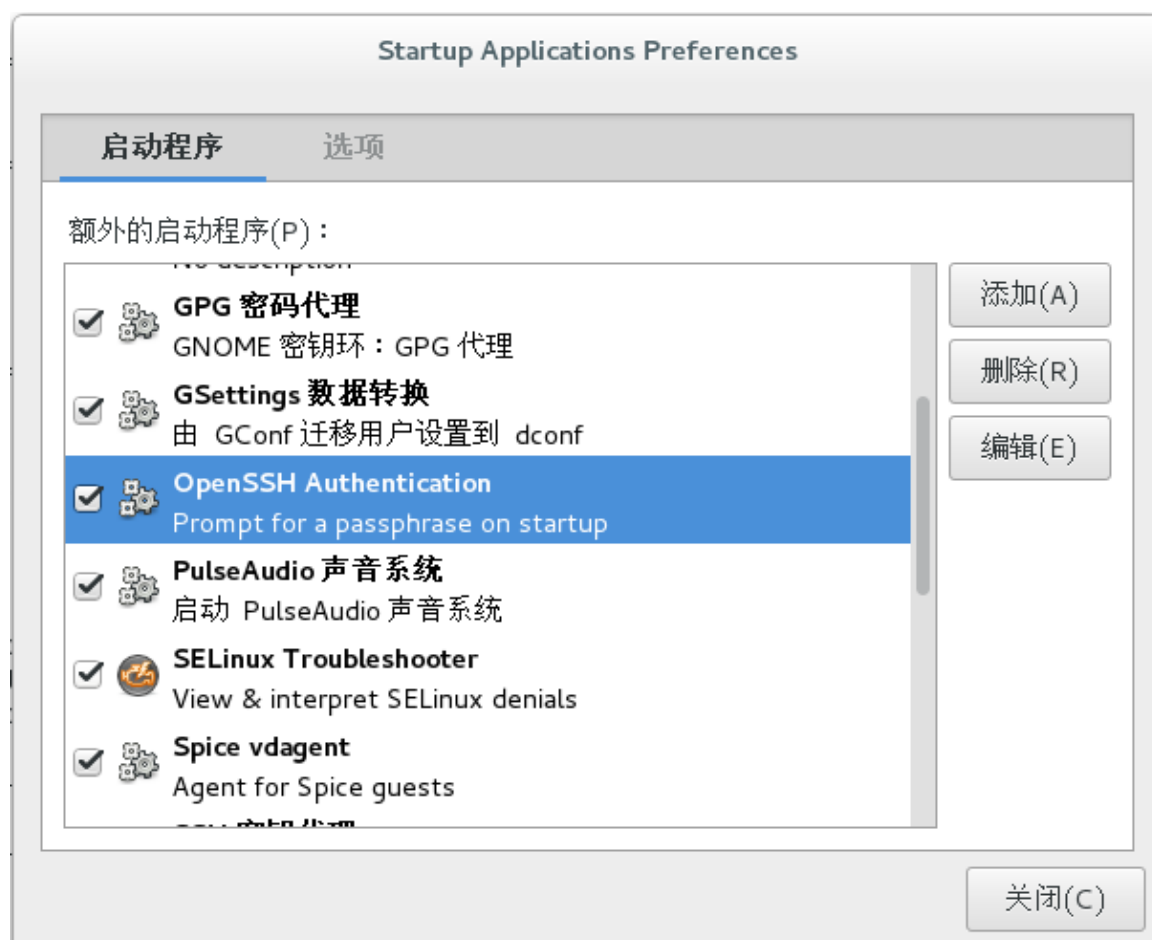


图 3-3 启用应用程序

- 5) 注销用户然后重新登录。将会弹出一个对话框提示您输入您的口令。从此刻起，您将不会再被 ssh、scp 或者 sftp 提示输入密码。



图 3-4 输入口令

要为某个确定的 `shell` 终端保存口令，可以使用以下命令：

```
~]$ ssh-add  
Enter passphrase for /home/USER/.ssh/id_rsa:
```

注意，当您登出 `shell` 时，您的口令将被忘记。您必须在每次登录一个虚拟控制台或终端窗口时执行此命令。

3.2.2.7 OpenSSH 客户端

您必须安装 `openssh-clients` 包后，才能从客户端机器连接到一个 `OpenSSH` 服务端（请参见 2.2.2.4 安装软件包。了解如何在中标麒麟高级服务器操作系统中安装新的包）。

3.2.2.8 使用 `ssh` 工具

`ssh` 工具可以让您登录到一台远程机器上，并在上面执行命令。它是对 `rlogin`、`rsh` 和 `telnet` 程序的一个安全替换。

和 `telnet` 命令相似，使用以下命令登录到一台远程机器上：

```
ssh hostname
```

例如，要登录到一台名为 `penguin.example.com` 的远程主机，可以在 `shell` 命令行提示符下输入以下命令：

```
~]$ ssh penguin.example.com
```

该命令将会以您正在使用的本地机器的用户名登录。如果您想指定一个不同的用户名，请使用以下命令：

```
ssh username@hostname
```

例如，以 `USER` 登录到 `penguin.example.com`，请输入以下命令：

```
~]$ ssh USER@penguin.example.com
```

您第一次初始连接时，将会看到和如下内容相似的信息：

```
The authenticity of host 'penguin.example.com' can't be established.  
ECDSA key fingerprint is 256  
da:24:43:0b:2e:c1:3f:a1:84:13:92:01:52:b4:84:ff.  
Are you sure you want to continue connecting (yes/no)?
```

在回答对话框中的问题之前，用户应该始终检查指印是否正确。用户可以询问服务端的管理员以确认密钥是正确的。这应该以一种安全的、事先约定好的方式进行。如果用户可以使用服务端的主机密钥，可以使用以下 `ssh-keygen` 命令来检查指印：

```
~]# ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key.pub  
256 da:24:43:0b:2e:c1:3f:a1:84:13:92:01:52:b4:84:ff (ECDSA)
```

输入 `yes` 接受密钥并确认连接。您将会看到一个有关服务端已经被添加到已知的主机列表中的通告，以及一个输入密码的提示：

```
Warning: Permanently added 'penguin.example.com' (ECDSA) to the list  
of  
known hosts.  
USER@ penguin.example.com's password:
```

重要说明：

如果 SSH 服务端的主机密钥改变了，客户端将会通知用户连接不能继续，除非将服务端的主机密钥从 `~/.ssh/known_hosts` 文件中删除。然而，在进行此操作之前，请联系 SSH 服务端的系统管理员，验证服务端没有收到攻击。

要从 `~/.ssh/known_hosts` 文件中删除一个密钥，可以使用如下命令：

```
~]# ssh-keygen -R penguin.example.com  
# Host penguin.example.com found: line 15 type ECDSA  
/home/USER/.ssh/known_hosts updated.
```

Original contents retained as
/home/USER/.ssh/known_hosts.old

在输入密码之后，您将会进入远程主机的 shell 命令行提示符下。

可选地，ssh 程序可以用来在远程主机上执行一条命令，而不用登录到 shell 命令行提示符下：

ssh *[username@]hostname command*

例如，/etc/redhat-release 文件提供有关操作系统版本的信息。要查看 penguin.example.com 上该文件的内容，输入：

```
~]$ ssh USER@penguin.example.com cat /etc/redhat-release
USER@penguin.example.com's password:
NeoKylin Linux Advanced Server release V7Update2 (Potassium)
```

在您输入正确的密码之后，将会显示远程主机的操作系统版本信息，然后您将返回到本地的 shell 命令行提示符下。

3.2.2.9 使用 scp 工具

scp 可以用来在主机之间通过一个安全加密的连接传输文件。在设计上，它和 rcp 非常相似。

要传输一个本地文件到远程系统中，可以使用如下形式的命令：

scp *localfile username@hostname:remotefile*

例如，如果您想将 taglist.vim 传输到名为 penguin.example.com 的远程主机上，可以在 shell 命令行提示符下输入以下命令：

```
~]$ scp taglist.vim
USER@penguin.example.com:~: vim/plugin/taglist.vim
USER@penguin.example.com's password:
taglist.vim                                100%  144KB
```

```
144.5KB/s
```

```
00:00
```

一次可以指定多个文件。要传输.vim/plugin/目录下的内容到远程主机penguin.example.com的相同目录下，可以输入以下命令：

```
~]$ scp .vim/plugin/* USER@penguin.example.com:.vim/plugin/
```

```
USER@ penguin.example.com's password:
```

```
closetag.vim                                100%  13KB 12.6KB/s
```

```
00:00
```

```
snippetsEmu.vim                            100%  33KB 33.1KB/s
```

```
00:00
```

```
taglist.vim                                100% 144KB 144.5KB/s
```

```
00:00
```

要将一个远程的文件传输到本地系统上，可以使用以下语法：

```
scp username@hostname:remotefile localfile
```

例如，要从远程主机上下载.vimrc 配置文件，可以输入：

```
~]$ scp USER@penguin.example.com:.vimrc .vimrc
```

```
USER@penguin.example.com's password:
```

```
.vimrc                                100% 2233
```

```
2.2KB/s
```

```
00:00
```

3.2.2.10 使用 sftp 工具

sftp 工具可以用来打开一个安全的、交互式的 FTP 会话。在设计上，它类似于 ftp，不同之处在于 sftp 使用了一个安全的加密连接。

要连接到远程系统中，可以使用如下形式的命令：

```
sftp username@hostname
```

例如，要使用 USER 用户登录到名为 penguin.example.com 的远程主机上，可以输入：

```
~]$ sftp USER@penguin.example.com
USER@penguin.example.com's password:
Connected to penguin.example.com.
sftp>
```

当您输入正确的密码之后，您将会看到一个 sftp 的命令行提示符。sftp 工具可以使用一系列和 ftp 类似的命令（参见表格 3-15）。

表格 3-15 常用的 sftp 命令

命令	描述
ls [<i>directory</i>]	列出一个远程目录的内容。如果没有提供任何目录名称，默认使用当前的工作目录。
cd <i>directory</i>	切换远程工作目录到指定的目录下。
mkdir <i>directory</i>	创建一个远程目录。
rmdir <i>path</i>	删除一个远程目录。
put <i>localfile</i> [<i>remotefile</i>]	将本地文件传输到远程主机上。
get <i>remotefile</i> [<i>localfile</i>]	将远程主机上的文件下载到本地主机上。

要获取可用命令的完整列表，请参考 sftp(1)用户手册页面。

3.2.3 不只是一个安全的 Shell

一个安全的命令行界面只不过是 SSH 众多使用方式的开端。给予合适的带宽，可以通过 SSH 通道进行 X11 会话的转发。或者，通过 TCP/IP 转发，系统间以前的不安全端口连接可以映射到指定的 SSH 通道上。

3.2.3.1 X11 转发

要通过 SSH 连接打开 X11 会话，可以使用以下形式的命令：

```
ssh -Y username@hostname
```

例如，要使用 USER 用户登录到名为 penguin.example.com 的远程主机上，

可以输入：

```
~]$ ssh -Y USER@penguin.example.com  
USER@penguin.example.com's password:
```

当从安全 shell 命令行提示符下运行一个 X 程序时，SSH 客户端和服务端会创建一个新的安全通道，X 程序数据通过这个通道透明地发送到客户端机器上。

注意，在发生 X11 转发之前，必须在远程系统中安装好 X 窗口系统。以 root 用户输入以下命令可以安装 X11 软件包分组：

```
~]# yum group install "X Window System"
```

要了解关于软件包分组的更多信息，请参见 2.2.2 管理软件包。

X11 转发非常有用。例如，X11 转发可以用来为【打印设置】工具创建一个安全的交互式会话。要这样做，先使用 ssh 连接到服务端，然后输入：

```
~]$ system-config-printer &
```

【打印设置】工具将会显示出来，让远程用户可以在远程主机上安全地配置打印。

3.2.3.2 端口转发

SSH 可以通过端口转发安全加固其他不安全的 TCP/IP 协议。在使用这一技术时，SSH 服务端成为了 SSH 客户端的一个加密导管。

端口转发的工作原理是将客户端的一个本地端口映射到服务端的一个远程端口上。SSH 可以从服务端将任意端口映射到客户端的任意端口上。这一技术并不需要端口号互相匹配。

重要说明：

要设置端口转发监听 1024 以下的端口，需要 root 级别的访问权限。

要创建一个监听 localhost 上的连接的 TCP/IP 端口转发通道，可以使用以下形式的命令：

```
ssh -L local-port:remote-hostname:remote-port username@hostname
```

例如，要使用 POP3 通过一个加密连接检查名为 mail.example.com 的服务器上的邮件，可以使用以下命令：

```
~]$ ssh -L 1100:mail.example.com:110 mail.example.com
```

一旦客户端机器和邮件服务器之间的端口转发通道准备就绪后，就可以使用一个 POP3 邮件客户端在 localhost 上使用 1100 端口来检查新邮件了。任何在客户端系统上发往 1100 端口的请求，都将被安全地导向 mail.example.com 服务器。

如果 mail.example.com 没有运行 SSH 服务端，但是同一网络中的另一台机器运行了 SSH 服务端，那么 SSH 仍然可以用来对连接进行安全加固。当然，使用的命令会略有不同：

```
~]$ ssh -L 1100:mail.example.com:110 other.example.com
```

在这一示例中，从客户端机器的 1100 端口发出的 POP3 请求，将通过 22 端口上的 SSH 连接被转发到 SSH 服务端 other.example.com。然后，other.example.com 连接到 mail.example.com 上的 110 端口来检查新邮件。注意，在使用这一技术时，只有客户端和 other.example.com 服务端之间的连接是安全的。

端口转发也可以用来通过网络防火墙安全地获取信息。如果防火墙配置为允许放行使用标准端口（即 22 端口）的 SSH 数据流，但是阻塞了对其它端口的访问，那么在要在两台主机之间使用被阻塞端口建立连接仍然是可能的，只要将它们之间的通信通过一条建立好的 SSH 连接进行重定向即可。

重要说明：

以这种方式来使用端口转发技术对连接进行转发，将允许客户端系统上的任何用户都能连接到相应的服务上。如果客户端系统被入侵，攻击者也同样能够访问转发的服务。

担心端口转发的系统管理员，可以在服务端将/etc/ssh/sshd_config 文件中的 AllowTCPForwarding 选项设置为 No，并重启 sshd 服务，来禁用端口转发功能。

3.3 TigerVNC

TigerVNC（Tiger Virtual Network Computing）是一个图形化桌面共享系统，

可以让您远程控制其它计算机。

TigerVNC 采用服务端-客户端架构：服务端共享它的输出（vncserver），客户端（vncviewer）连接到客户端。

重要说明：

和以前的中标麒麟高级服务器操作系统发行版不一样，中标麒麟高级服务器操作系统 V7 中的 TigerVNC 使用 systemd 系统管理守护进程进行配置。配置文件/etc/sysconfig/vncserver 被替换成了/etc/systemd/system/vncserver@.service。

3.3.1 VNC 服务端

vncserver 工具用来启动一个 VNC（Virtual Network Computing）桌面。它将使用适当的选项运行 Xvnc，并在 VNC 桌面中启动一个窗口管理器。vncserver 允许用户在一台机器上并行地运行多个独立的会话，之后这些会话可以被任意数量的客户端从任意位置访问。

3.3.1.1 安装 VNC 服务端

要安装 TigerVNC 服务端，请以 root 用户执行以下命令：

```
~]# yum install tigervnc-server
```

3.3.1.2 配置 VNC 服务端

VNC 服务端可以被配置了为一个或多个用户（倘若这些用户账户存在于系统上）启动一个显示，可以配置诸如显示设置、网络地址和端口，以及安全设置等可选参数。

实例：为单一用户配置 VNC 显示

- 1) 需要一个名为/etc/systemd/system/vncserver@.service 的配置文件。要创建该文件，可以以 root 用户复制/lib/systemd/system/vncserver@.service 文件。

```
~]# cp /lib/systemd/system/vncserver@.service  
/etc/systemd/system/vncserver@.service
```

没有必要在文件名中包含显示编号，因为 systemd 会在需要时自动在内存中创建适当命名的实例，用显示编号替代服务文件中的'%i'。对于单一用户来说，

没有必要重命名该文件。对于多用户来说，有必要为每个用户创建一个唯一命名的服务文件，例如，以某种方式在文件名中添加上用户名。具体细节请参见为 0 两个用户配置 VNC 服务端。

- 1) 编辑/etc/systemd/system/vncserver@.service 文件，用实际的用户名替换 USER。文件的剩余部分保持原样。-geometry 参数指定将要创建的 VNC 桌面的大小，默认情况下，大小为 1024x768。

```
ExecStart=/sbin/runuser -l USER -c "/usr/bin/vncserver %i -geometry
1280x1024"

PIDFile=/home/USER/.vnc/%H%i.pid
```

- 2) 保存修改。
- 3) 要让修改立即生效，请执行以下命令：

```
~ ]# systemctl daemon-reload
```

- 4) 为配置文件中定义的用户设置密码。注意，首先您需要从 root 用户切换到 USER 用户。

```
~ ]# su - USER

~ ]$ vncpasswd

Password:

Verify:
```

重要说明：

保存的密码是未加密的；任何能够访问密码文件的人都能找到明文密码。

为两个用户配置 VNC 服务端

如果您想在同一个机器上配置一个以上的用户，请创建不同的服务文件，每个用户对应一个文件。

- 1) 创建两个服务文件，例如 vncserver-USER_1@.service 和 vncserver-USER_2@.service。请用正确的用户名替换这些文件名中的

USER。

2) 为两个用户设置密码:

```
~ ]$ su - USER_1
```

```
~ ]$ vncpasswd
```

Password:

Verify:

```
~ ]$ su - USER_2
```

```
~ ]$ vncpasswd
```

Password:

Verify:

3.3.1.3 启动 VNC 服务端

要启动或者开机自启动 VNC 服务, 请在命令行中指定显示编号。在 3.3.1.2 配置 VNC 服务端下的示例: “为单一用户配置 VNC 显示” 中使用的配置文件担任着模板的角色, 文件中的%i 将被 systemd 用显示编号替换。使用一个有效的显示编号来执行以下命令:

```
~ ]# systemctl start vncserver@:display_number.service
```

您也可以让服务在系统启动时自动启动。之后, 当您登录时, vncserver 已经自动启动了。以 root 用户执行以下命令:

```
~ ]# systemctl enable vncserver@:display_number.service
```

这时候, 其他用户可以使用一个 VNC 查看程序, 以及定义好的显示编号和密码来连接到 VNC 服务端。假若已经安装好了一个图形化桌面, 将会显示该桌面的一个实例。这个实例和目标机器上正在显示的实例是不相同的。

为两个用户和两个不同的显示配置 VNC 服务端

对于两个已经配置好的 VNC 服务端 vncserver-USER_1@.service 和 vncserver-USER_2@.service, 您可以启用不同的显示编号。例如, 以下命令将会使 USER_1 的 VNC 服务端在显示编号 3 上启动, 而 USER_2 的 VNC 服务端将在显示编号 5 上启动:

```
~]# systemctl start vncserver-USER_1@:3.service
~]# systemctl start vncserver-USER_2@:5.service
```

3.3.1.4 终止 VNC 会话

类似于启用 vncserver 的开机自启动，您也可以禁用该服务的开机自启动：

```
~]# systemctl disable vncserver@:display_number.service
```

或者，当您的系统正在运行时，您可以以 root 用户执行以下命令来停止 VNC 服务：

```
~]# systemctl stop vncserver@:display_number.service
```

3.3.2 共享一个已存在的桌面

默认情况下，一个已登录的用户拥有一个由 X 服务端提供的、在显示编号 0 上的桌面。用户可以使用 TigerVNC 服务端的 x0vncserver 来共享他们的桌面。

实例：共享一个 X 桌面

要使用 x0vncserver 共享一个已登录用户的桌面，请按以下步骤操作：

- 1) 以 root 用户执行以下命令：

```
~]# yum install tigervnc-server
```

- 2) 为用户设置 VNC 密码：

```
~]$ vncpasswd
Password:
Verify:
```

- 3) 以已登录用户的身份执行以下命令：

```
~]$ x0vncserver -PasswordFile=.vnc/passwd -AlwaysShared=1
```

倘若防火墙已经配置了允许连接 5900 端口，远程查看器现在就可以连接到显示编号 0 上，并查看已登录用户的桌面了。请参见 3.3.3.2 为 VNC 配置防火墙。

3.3.3 VNC 查看器

vncviewer 是一个用来显示图形用户界面并远程控制 vncserver 的程序。

为了操作 `vncviewer`，有一个包含许多条目的弹出菜单，通过这些条目可以执行诸如切换到/切换出全屏模式、退出查看器等多种操作。可选地，您可以通过终端来操作 `vncviewer`。在命令行中输入 `vncviewer -h` 可以列出 `vncviewer` 的参数。

3.3.3.1 安装 VNC 查看器

要安装 TigerVNC 的客户端 `vncviewer`，请以 root 用户执行以下命令：

```
~]# yum install tigervnc
```

连接到 VNC 服务端

一旦配置好了 VNC 服务端，您就可以从任何 VNC 查看器连接到该服务端了。

实例：使用图形用户界面连接到 VNC 服务端

- 1) 不带任何参数输入 `vncviewer` 命令，将会显示标题为【VNC Viewer: Connection Details】的工具。它会提示输入一个要连接的 VNC 服务端。
- 2) 如果需要的话，可以设置禁止断开连接到同一个显示编号上的任何已存在的 VNC 连接，请按以下步骤选择相应的选项允许桌面共享：
 - a. 选择【Options】按钮。
 - b. 选择【Misc.】标签页。
 - c. 选择【Shared】复选框。
 - d. 按下【OK】按钮返回到主界面。
- 3) 输入一个要连接的地址和显示编号：

```
address:display_number
```

- 4) 点击【Connect】按钮，连接 VNC 服务端的显示编号。
- 5) 您将会被提示输入 VNC 密码。这个密码就是对应显示编号的用户的 VNC 密码，除非设置了一个全局默认的 VNC 密码。

之后您将会看到一个显示 VNC 服务端桌面的窗口。注意，这个桌面不是普通用户所看到的桌面，它是一个 Xvnc 桌面。

实例：使用命令行连接到 VNC 服务端

- 1) 以地址和显示编号为参数输入 `vncviewer` 命令：

```
vncviewer address:display_number
```

这里的 `address` 是 IP 地址或者主机名。

- 2) 输入 VNC 密码进行身份认证。这个密码就是对应显示编号的用户的 VNC 密码，除非设置了一个全局默认的 VNC 密码。
- 3) 之后您将会看到一个显示 VNC 服务端桌面的窗口。注意，这个桌面不是普通用户所看到的桌面，它是一个 Xvnc 桌面。

3.3.3.2 为 VNC 配置防火墙

当使用非加密连接时，`firewalld` 可能会阻止连接。为了让 `firewalld` 允许 VNC 数据包通过，您可以开放指定的 TCP 数据流端口。当使用 `-via` 选项时，数据流通过 SSH 做了重定向，而 SSH 的端口在 `firewalld` 中默认是开放的。

备注：

VNC 服务端的默认端口是 5900。要得到远程桌面将被访问的端口号，可以用默认端口号加上用户分配的显示编号。例如，对于第二个显示屏： $2 + 5900 = 5902$ 。

对于显示编号 0 到 3，可以直接利用 `firewalld` 对 VNC 服务的支持来操作，具体是通过以下将描述的 `service` 选项来进行的。注意，对于大于 3 的显示编号，其对应的端口请按本节中的例子“”中描述的方法来开放。

实例：在防火墙中启用 VNC 服务

- 1) 执行以下命令来查看有关 `firewalld` 的设置信息：

```
~ ]$ firewall-cmd --list-all
```

- 2) 要放行从一个指定地址发起的所有 VNC 连接，可以执行以下命令：

```
~]# firewall-cmd --add-rich-rule='rule family="ipv4" source  
address= "192.168.122.116" service name=vnc-server accept'
```

```
success
```

注意，这些修改不是持久的，下一次系统启动这些修改就无效了。

要对防火墙做永久性的修改，请加上--permanent 选项重新执行上述命令。

参见《NeoKylin Linux Advanced Server V7 安全手册》了解防火墙丰富语言命令的使用。

3) 要验证以上设置，请执行以下命令：

```
~]# firewall-cmd --list-all
public (default, active)
  interfaces: bond0 bond0.192
  sources:
  services: dhcpv6-client ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
rule family="ipv4" source address="192.168.122.116" service
name="vnc-server" accept
```

要开放指定的端口或端口范围，请使用 `firewall-cmd` 命令行工具的--add-port 选项。例如，VNC 显示编号 4 要求开放 TCP 数据流的 5904 端口。

实例：在 firewalld 中开放端口

1) 要在 public 区域为 TCP 数据流开放一个端口，以 root 用户执行以下命令：

```
~]# firewall-cmd --zone=public --add-port=5904/tcp
Success
```

- 2) 要查看 **public** 区域当前所开放的端口，可以执行以下命令：

```
~]# firewall-cmd --zone=public --list-ports  
  
5904/tcp
```

已开放端口可使用 `firewall-cmd --zone=zone --remove-port=number/protocol` 命令删除。

注意，这些修改不是持久的，下一次系统启动这些修改就无效了。要对防火墙做永久性的修改，请加上 `--permanent` 选项重新执行上述命令。参见《NeoKylin Linux Advanced Server V7 安全手册》了解更多关于在 `firewalld` 中开放和关闭端口的信息。

3.3.3.3 使用 SSH 连接到 VNC 服务端

VNC 是一个很明显的文本网络协议，在通信中没有相应的安全机制来应对可能的攻击。为了使通信安全，您可以通过使用 `-via` 选项来加密您的服务端-客户端连接。这将会在 VNC 服务端和客户端之间创建一个 SSH 隧道。

加密 VNC 服务端-客户端连接的命令格式如下：

```
vncviewer -via user@host:display_number
```

实例：使用 `-via` 选项

- 1) 要使用 SSH 来连接到 VNC 服务端，输入以下命令：

```
~ ]$ vncviewer -via USER_2@192.168.2.101:3
```

- 2) 当提示您输入密码时，输入正确的密码，并按回车键确认。
3) 一个带有远程桌面的窗口将会显示在您的屏幕上。

限制 VNC 访问

如果您倾向于只使用加密连接，您可以在 `systemd.service` 文件的 `ExecStart` 这一行中使用 `-localhost` 选项，来禁用未加密连接：

```
ExecStart=/sbin/runuser -l user -c "/usr/bin/vncserver -localhost %i"
```

这样一来，除了本地连接和加了 `-via` 选项所导致的使用 SSH 来发送的端口

转发连接外，`vncserver` 将停止接受所有其他连接。

有关使用 SSH 的更多信息，请参见 3.2OpenSSH。

4 服务器

4.1 Web 服务器

web 服务器能够为用户提供一种基于 web 的网络服务,通常以网页形式呈现给用户所需的网络信息。基于插件机制,现在可以浏览更多的各种文档了。因为 web 服务器使用的是超文本传输协议 (HTTP),所以也称 web 服务器为 HTTP 服务器。

4.1.1 Apache HTTP 服务器

NeoKylin Linux Advanced Server V7 中提供的可用 web 服务器是 Apache HTTP 服务器 (httpd), httpd 的版本为 2.4, 它是由 Apache 软件基金会开发的一种开源 web 服务器。

如果当前的 NeoKylin Linux Advanced Server 系统是升级而来的,则将需要相应地更新 httpd 服务的配置。本节主要回顾一些新增加的特性,简要介绍一下 2.4 版本和 2.2 版本的重要区别,以及对旧版本配置文件的变更。

4.1.1.1 重要变更

下面给出 Apache HTTP 服务器,在 NeoKylin Linux Advanced Server V7 中,针对 NeoKylin Linux Advanced Server 6 所做的一些变更。

httpd 服务控制

随着迁移废弃了 SysV 初始化脚本,系统管理员应该使用 apachectl 和 systemctl 命令来管理服务,而不是使用 service 命令了。下面给出了查看 httpd 服务的例子:

命令

```
service httpd graceful
```

被替换成了

```
apachectl graceful
```

针对 httpd 的 systemd 单元文件和初始化脚本,有如下所示的不同操作:

- 当重新加载服务时,默认使用的是 graceful 重启方式

➤ 当停止服务时，默认使用的是 graceful 停止方式

命令

```
service httpd configtest
```

被替换成了

```
apachectl configtest
```

私用 /tmp

为了提高系统安全性，systemd 单元文件运行 httpd 守护进程时，所使用的 /tmp 是私用的，和系统的/tmp 是分开的。

配置布局

当前系统中，用于模块加载的配置文件都存放在了 /etc/httpd/conf.modules.d/目录下。为 httpd 提供的一些可加载附加模块的配置文件也存放于此，如：php。/etc/httpd/conf/httpd.conf 文件中的主要配置项 Include 就是用来描述/etc/httpd/conf.modules.d/目录下的 include 文件的。这就意味着在 httpd.conf 的主体之前，先要处理 conf.modules.d 下的全部配置文件。在/etc/httpd/conf.d 目录下的文件，需要由 IncludeOptional 指示项，在 httpd.conf 文件的最后加以描述，也即是说，这些文件会在 httpd.conf 主体之后被处理。

由 httpd 软件包提供的一些其它配置文件：

- /etc/httpd/conf.d/autoindex.conf --- 配置 mod_autoindex 目录索引
- /etc/httpd/conf.d/userdir.conf --- 配置可访问用户目录，例如：
http://example.com/~username/；为了安全起见，应该禁用这种默认访问。
- /etc/httpd/conf.d/welcome.conf --- 在使用早期版本时，当 http://localhost/ 没有内容的情况下，就将其设置成了欢迎页面。

默认配置

默认情况下，提供的 httpd.conf 文件是最小配置。许多常见的配置项，如：以前默认配置中的 Timeout 或 KeepAlive 都不再被明确设置了；硬编码设

置也被取代了。针对所有配置指示项的硬编码设置在手册中有详细说明。
不兼容的语法变更

如果要从现有的 httpd 2.2 配置迁移到 httpd 2.4，则存在有大量前后不兼容的 httpd 配置语法需要修改。更多有关升级部分的 Apache 文档，请参看 <http://httpd.apache.org/docs/2.4/upgrading.html> 。

处理模型

在 NeoKylin Linux Advanced Server 的早期版本中，不同的多处理模型（MPM）提供了不同的 httpd 二进制文件。如：基于子进程模型可以用“prefork”代表/usr/sbin/httpd；基于线程模型可以用“worker”代表/usr/sbin/httpd.worker 。

在 NeoKylin Linux Advanced Server V7 中，只使用了单一的 httpd。3 个 MPM 作为可加载模块还是有效的：worker，prefork（default）和 event，当需要加载一个 MPM 模块时，可以通过编辑/etc/httpd/conf.modules.d/00-mpm.conf 配置文件，把相关 MPM 模块前的注释符#去掉，就可以实现 MPM 模块的加载了。

包的变更

由独立的分包 mod_ldap 实现模块的 LDAP 身份验证和授权。由新的分包 mod_session 提供模块 mod_session 和 helper。由新的分包 mod_proxy_html 提供模块 mod_proxy_html 和 mod_xml2enc。这些软件包都可以从可选频道中获得。

打包文件系统布局

不再使用/var/cache/mod_proxy/目录，取而代之的是/var/cache/httpd/目录下的 proxy 和 ssl 子目录 。

httpd 软件包所提供的打包内容已经从/var/www 迁移到了/usr/share/httpd/。

- /usr/share/httpd/icons/ --- 包含了一组用于目录索引的图标。早期版本在/var/www/icons/目录下，现在迁移到了/usr/share/httpd/icons 目录下。在默认情况下，http://localhost/icons/ 是有效的。在/etc/httpd/conf.d/autoindex.conf 文件中，可以配置图标的位置和可用性。

- `/usr/share/httpd/manual/` --- `/var/www/manual/` 目录已经迁移到了 `/usr/share/httpd/manual/` 目录。这个目录包含了来自于 `httpd-manual` 软件包的内容。包含了 `httpd` 的 HTML 版手册。如果安装了这个软件包，则 `http://localhost/manual/` 是有效的。在 `/etc/httpd/conf.d/manual.conf` 文件中，可以配置手册的位置和可用性。
- `/usr/share/httpd/error/` --- `/var/www/error/` 目录已经迁移到了 `/usr/share/httpd/error/` 目录。在默认配置中，已删除了自定义的多国语言 HTTP 错误信息包的配置。在 `/usr/share/doc/httpd-VERSION/httpd-multilang-errordoc.conf` 文件中，提供了配置文件样例。

身份验证，授权和访问控制

用于控制身份验证、授权和访问控制的配置指令已发生了明显的改变。在现有配置文件中使用的 `Order`，`Deny` 和 `Allow` 指令应适用于新的 `Require` 语法。有关 Apache 文档的更多详细信息，请查看 <http://httpd.apache.org/docs/2.4/howto/auth.html>。

suexec

为了提高系统的安全性，已不再安装 `suexec` 二进制了，取而代之的是文件系统能力位集，允许更严格的权限设置。结合这一改变，`suexec` 也不再使用 `/var/log/httpd/suexec.log` 日志文件了。相反，日志信息都送到了 `syslog`；默认情况下，将出现在 `/var/log/secure` 日志文件中。

模块接口

由于 `httpd` 改变了模块接口定义，因此，基于 `httpd 2.2` 构建的第 3 方模块对 `httpd 2.4` 是不兼容的。这样，就需要根据 `httpd 2.4` 模块接口定义，对那些模块代码做必要的修改，最后，再重构。有关 `httpd2.4` API 变更的详细信息列表，请参看 http://httpd.apache.org/docs/2.4/developer/new_api_2_4.html。

用于从源构建模块的 `apxs` 可执行文件已经从 `/usr/sbin/apxs` 迁移到了 `/usr/bin/apxs`。

被删除模块

在 NeoKylin Linux Advanced Server V7 中，已经被删除的 httpd 模块列表如下：

`mod_auth_mysql`, `mod_auth_pgsql`

在 `mod_authn_dbd` 中，httpd 2.4 提供了内部的 SQL 数据库认证机制。

`mod_perl`

在 httpd 2.4 中，不再正式支持 `mod_perl` 了。

`mod_authz_ldap`

在 httpd 2.4 的分包 `mod_ldap` 中，用 `mod_authz_ldap` 提供了对 LDAP 的支持。

4.1.1.2 更新配置

为了更新 Apache HTTP Server version 2.2 配置文件，需要完成以下步骤：

1. 由于模块名称有可能变更了，因此，需要先确认所有模块名称是否正确。针对已经变更了名称的每个模块，需要有针对性地调整 `LoadModule` 指示项。
2. 在需要加载第三方模块前，需要重新编译它们。这通常意味着需要身份验证和授权模块。
3. 如果要使用 `mod_userdir` 模块，则在 `UserDir` 指示项中，需要提供目录名称（通常为 `public_html`）。
4. 如果要使用 Apache HTTP 安全服务器的话，则有关启用安全套接字层（SSL）协议的更多重要信息，请参看 4.1.1.8 启用 `mod_ssl` 模块。

可以用以下命令，检查配置文件中可能出现的错误。

```
~]# apachectl configtest
```

```
Syntax OK
```

有关如何将 Apache HTTP 服务器的配置从 2.2 版本升级 2.4 版本，请参看 <http://httpd.apache.org/docs/2.4/upgrading.html> 。

4.1.1.3 运行 httpd 服务

本章描述如何启动，停止和重启 Apache HTTP，以及如何检查 Apache HTTP

的当前状态。为了能使用 httpd 服务,应先用以下命令确认是否已经安装了 httpd。

```
~]# yum install httpd
```

通常,在 NeoKylin Linux Advanced Server V7 中,有关一些目标概念和如何管理系统服务的更多信息,请参看 3.1 使用 systemd 管理系统服务。

启动服务

由 root 用户执行以下命令,启动 httpd 服务:

```
~]# systemctl start httpd.service
```

执行以下命令,可以使得在系统引导时,自动启动 httpd 服务:

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

备注:

如果要以安全服务器方式启动 Apache HTTP 服务器,则操作系统启动后,会提示需要一个用 SSL 私钥加密的密码。

停止服务

由 root 用户执行以下命令,停止 httpd 服务:

```
~]# systemctl stop httpd.service
```

执行以下命令,可以避免 httpd 服务随操作系统自启:

```
~]# systemctl disable httpd.service  
  
rm '/etc/systemd/system/multi-user.target.wants/httpd.service'
```

重启服务

有以下 3 种方法,可以重启 httpd 服务:

1. 由 root 用户执行以下命令,完全重启服务

```
~]# systemctl restart httpd.service
```

这里,首先要停止这个运行着的 httpd 服务,然后再重启它。通常,在安装或删除一个动态加载模块(如:PHP)时,会用这个命令。

2. 由 root 用户执行以下命令，可以重新加载配置：

```
~]# systemctl reload httpd.service
```

这将会导致运行着的 httpd 服务去重新加载它的配置文件。当前正在处理的任何请求都将会被中断，客户端浏览器上也会看到有错误信息提示或页面不完整。

3. 为了重新加载配置而又不影响当前执行着的请求，可以由 root 用户执行以下命令：

```
~]# apachectl graceful
```

这将会导致运行着的 httpd 服务去重新加载它的配置文件，当前正在处理的任何请求都将会沿用旧的配置继续处理。

在 NeoKylin Linux Advanced Server V7 中，有关如何管理系统服务的更多信息，请参看 3.1 使用 systemd 管理系统服务。

验证服务状态

在 shell 提示符下，执行以下命令，可以检查运行着的 httpd 服务的状态：

```
~]# systemctl is-active httpd.service  
active
```

4.1.1.4 编辑配置文件

默认情况下，当启动 httpd 时，会读取表格 4-1 httpd 服务配置文件中列出的 httpd 服务配置文件。

表格 4-1 httpd 服务配置文件

路 径	描 述
/etc/httpd/conf/httpd.conf	主要配置文件
/etc/httpd/conf.d/	包含在主配置文件中的其它配置文件的所在目录

虽然，默认配置能适合于大多数情况的使用，然而，熟悉一些其它更重要的配置选项也是很有必要的。注意：为了使任一修改都能生效，完成修改后必须重启 web 服务器。想了解如何重启 httpd 服务的更多信息，请参看 0 重启服务。

可以用以下命令，检查配置文件中可能出现的错误：

```
~]# apachectl configtest  
Syntax OK
```

解决错误的简单方法就是将配置文件恢复到修改前的状态。

4.1.1.5 使用模块

作为模块化结构的应用，httpd 服务器发布时带有大量的动态共享对象(DSOs)，根据需要它们可以在运行中被动态加载或卸载。在 NeoKylin Linux Advanced Server V7 中，这些模块被存放在/usr/lib64/httpd/modules/目录下。

加载模块

为了加载指定的 DSO 模块，需要用到 LoadModule 指示项。注意：以独立软件包形式提供的模块，通常，在/etc/httpd/conf.d/目录下都有它自己的配置文件。

实例：加载模块

```
LoadModule ssl_module modules/mod_ssl.so
```

完成后，需要重启 web 服务器来加载模块。有关如何重启 httpd 服务的更多信息，请参看 0 重启服务。

写模块

如果要创建新的 DSO 模块，则需要安装 httpd-devel 软件包。执行以下命令，就可以完成安装。

```
~]# yum install httpd-devel
```

这个软件包包含有构建模块所需的 include 文件，header 文件和编译生成工具 Apache eXtenSion(apxs)应用程序。

一旦写好源码，就可用以下命令来构建模块了：

```
~]# apxs -i -a -c module_name.c
```

模块生成后，就可以用加载 Apache HTTP 服务器其它模块的方法来加载它。

4.1.1.6 设置虚拟主机

可以使用 Apache HTTP 服务器的内置虚拟主机特性，实现基于不同 IP，主机名和端口号提供的不同网络信息服务。

为了设置基于主机名的虚拟主机，可以拷贝 /usr/share/doc/httpd-VERSION/httpd-vhosts.conf 配置文件到/etc/httpd/conf.d/目录下，并替换掉@@Port@@和@@ServerRoot@@占位符。根据需要自定义的选项，如下图实例所示。

实例：虚拟主机配置实例

```
<VirtualHost *:80>

    ServerAdmin webmaster@penguin.example.com

    DocumentRoot "/www/docs/penguin.example.com"

    ServerName penguin.example.com

    ServerAlias www.penguin.example.com

    ErrorLog "/var/log/httpd/dummy-host.example.com-error_log"

    CustomLog "/var/log/httpd/dummy-host.example.com-access_log" common

</VirtualHost>
```

注意 ServerName 必须是有效的 DNS 名。<VirtualHost>容器的可自定义性很好，可以接受主服务器中大多数指示项的配置。容器中包含的 User 和 Group 在此不支持了，取而代之的是 SuexecUserGroup。

备注：

如果配置的虚拟主机监听端口不是默认的，则需要确认是否根据要求，修改了/etc/httpd/conf/httpd.conf 中的全局指示项 Listen。

为了激活新配置的虚拟主机，需要重启 web 服务器。有关如何重启 httpd 服务的更多信息，请参看 0 重启服务。

4.1.1.7 创建 SSL 服务器

安全套接字层（SSL）是一种能使服务器和客户端进行安全数字通讯的加密协议。传输层安全（TLS）协议就是 SSL 的扩展和改进版本，能够确保隐私和数据的安全性和完整性。Apache HTTP 服务器和 mod_ssl 模块的结合，使得使用了 OpenSSL 工具包的模块，能够提供对 SSL/TLS 的支持，通常称它为 SSL 服务器。

NeoKylin Linux Advanced Server 也支持基于 TLS 实现的 Mozilla NSS。mod_nss 模块提供了对 Mozilla NSS 的支持。

不像 HTTP 连接，任何能够拦截到它的人都可以读和修改它。所谓 HTTPS 就是在 HTTP 上增加了对 SSL/TLS 的支持，它能保护传输内容不被窃听和修改。本章主要介绍在 Apache HTTP 服务器配置上，如何启用此类模块的一些基本方法，指导如何生成私钥和自签证书的过程。

概述证书和安全

密钥主要用于安全通讯。在传统或对称加密技术中，事务的两端运用相同的密钥来解码彼此的传输。然而，在公共或非对称加密技术中，有 2 个密钥共存：一个是要保密的私钥；另一个是可以共享的，公共公钥。使用公钥编码的数据只能用对应的私钥来解码；用私钥编码的数据只能用对应的公钥来解码。

为了基于 SSL 实现安全通讯，SSL 服务器必须要使用由证书颁发机构(CA)签名的数字证书。证书中包含有服务器的各种属性（例如：服务器主机名；公司名；地理位置等等），数字签名使用的是 CA 的私钥。这个签名可以确保特定证书颁发机构已签明了证书，并且不能以任何方式修改它。

当浏览器基于 web 要建立新的 SSL 连接时，会检查 web 服务器提供的证书。如果证书没有被可信 CA 机构签名，或者证书中的主机名和连上来的主机名不匹配，web 服务器会拒绝建立通讯，通常用户也会收到相应的错误信息。

默认情况下，大多数浏览器都配置有一组被广泛使用的合法签名证书。正因为如此，设置安全服务器时，总可以选择一个合适的 CA 证书，这样，最终用户就可以建立可信连接了，否则，会收到相应的错误信息，这时，需要人为接受一个证书。由于用户可以忽略证书错误，但是，却会使得攻击者可以拦截连接，因此，强力推荐尽可能使用可信 CA。相关的更多信息，请参看表表格 4-2 查看浏览器中通常可用的 CA。

表格 4-2 查看浏览器中通常可用的 CA

web 浏览器	链 接
Mozilla Firefox	Mozilla root CA 列表

Opera	Opera 可用的根证书
Internet Explorer	Microsoft Windows 可用的根证书
Chromium	Chromium project 可用的根证书

在建立 SSL 服务器时，需要生成一个证书请求和一个密钥，然后，发送证书请求，公司的身份证明和支付到证书颁发机构。一旦 CA 验证了你的证书请求和身份证明，它将会发送一个可以和服务器一起使用的签名证书给你。另外，可以创建一个自签名的证书，它不含有 CA 的数字签名，因此，这仅仅适用于以测试为目的场合。

4.1.1.8 启用 mod_ssl 模块

如果想基于 mod_ssl 模块，建立一个 SSL 或 HTTPS 服务器的话，则不能有另外一个应用或模块（如：mod_nss）使用相同的端口。端口 443 是默认 HTTPS 端口。

为了使用 mod_ssl 模块和 OpenSSL 工具包，建立一个 SSL 服务器，则需要安装 mod_ssl 和 openssl 软件包。由 root 用户执行以下命令就可以完成安装：

```
~]# yum install mod_ssl openssl
```

此时，会创建 mod_ssl 的配置文件/etc/httpd/conf.d/ssl.conf，默认情况下，它包含在 Apache HTTP 服务器的主配置文件中。为了加载模块，需要重启 httpd 服务，请参看 0 重启服务。

重要说明：

正如 POODLE: SSLv3 vulnerability (CVE-2014-3566)中所描述的，NeoKylin 不推荐启用 ssl，建议仅使用 TLSv1.1 或 TLSv1.2。使用 TLSv1.1 可以实现向后兼容。虽然许多产品多已支持使用 SSLv2 或 SSLv3 协议了，并默认启用了它们，但是，现在还是不适合强力推荐使用它们。

在 mod_ssl 中启用和禁用 SSL 和 TLS

为了启用和禁用指定版本的 SSL 和 TLS 协议，既可以在配置文件中，通过 在 “## SSL Global Context” 部分添加/删除 SSLProtocol 指示项来全局启用/禁用 SSL，也可以在每个 “VirtualHost”的 “# SSL Protocol support”中单独编辑默认项来实现。如果没有在每个域的主机部分指定它，则将会继承全局部分的设置。为

了禁用一个协议版本，管理员既可以仅在“SSL Global Context”部分来指定 SSLProtocol，也可以在每个域的虚拟主机部分指定它，注意要带上参数 all。

实例：禁用 SSLv2 和 SSLv3

为了禁用 SSL 2 和 SSL 3，可以在每个 VirtualHost 部分,启用除 SSL 2 和 SSL 3 意外的全部其它协议。处理过程如下所示：

- 1) 由 root 用户，打开/etc/httpd/conf.d/ssl.conf 文件，针对每个实例搜索 SSLProtocol 指示项。默认情况下，配置文件中只包含了一个，如下所示：

```
~]# vi /etc/httpd/conf.d/ssl.conf  
  
#    SSL Protocol support:  
  
# List the enable protocol levels with which clients will be able to  
# connect.  Disable SSLv2 access by default:  
  
SSLProtocol all -SSLv2
```

这一部分是在 VirtualHost 内。

- 2) 编辑 SSLProtocol 行，如下所示：

```
#    SSL Protocol support:  
  
# List the enable protocol levels with which clients will be able to  
# connect.  Disable SSLv2 access by default:  
  
SSLProtocol all -SSLv2 -SSLv3
```

上例，在 SSLProtocol 行中添加了-SSLv3。针对每个 VirtualHost，重复这个操作。最后，保存并关闭此文件。

- 3) 执行以下命令，验证针对 SSLProtocol 指示项的所有修改都已正确完成：

```
~]# grep SSLProtocol /etc/httpd/conf.d/ssl.conf  
  
SSLProtocol all -SSLv2 -SSLv3
```

如果有更多的 VirtualHost 部分，则这一步验证就显得格外重要。

- 4) 使用以下命令，重启 Apache 守护进程

```
~]# systemctl restart httpd
```

注意任一会话都将被中断。

实例：禁用除 TLS1 及更高版本以外的所有 SSL 和 TLS 协议

使用以下过程，可以禁用除 TLSv1 及更高版本以外的所有 SSL 和 TLS 协议。

- 1) 由 root 用户，打开 /etc/httpd/conf.d/ssl.conf 文件，针对每个实例搜索 SSLProtocol 指示项。默认情况下，配置文件中只包含了一个，如下所示：

```
~]# vi /etc/httpd/conf.d/ssl.conf

#    SSL Protocol support:

# List the enable protocol levels with which clients will be able to
# connect.  Disable SSLv2 access by default:

SSLProtocol all -SSLv2
```

- 2) 编辑 SSLProtocol 行，如下所示：

```
#    SSL Protocol support:

# List the enable protocol levels with which clients will be able to
# connect.  Disable SSLv2 access by default:

SSLProtocol -all +TLSv1 +TLSv1.1 +TLSv1.2
```

最后，保存并关闭此文件。

- 3) 验证过程，如下所示：

```
~]# grep SSLProtocol /etc/httpd/conf.d/ssl.conf

SSLProtocol -all +TLSv1 +TLSv1.1 +TLSv1.2
```

- 4) 使用以下命令，重启 Apache 守护进程

```
~]# systemctl restart httpd
```

注意任一会话都将被中断。

实例：检查 SSL 和 TLS 协议的状态

可以使用 openssl s_client -connect 命令，验证启用了/禁用了的 SSL 和 TLS 的版本，命令格式如下所示：

```
openssl s_client -connect hostname:port -protocol
```

这里：*port* 是待测试端口；*protocol* 是待测试协议版本。只需将 localhost 设

为主机名，就可以检查本地的 SSL 服务器是否正在运行。例如：为了检查是否在默认端口上建立了安全的 HTTPS 连接，需要查看 443 端口及验证 SSLv3 是否启用。完成这个检查任务，需要执行如下命令：

1)

```
~]# openssl s_client -connect localhost:443 -ssl3
CONNECTED(00000003)
139809943877536:error:14094410:SSL routines:SSL3_READ_BYTES:ssl3
alert handshake failure:s3_pkt.c:1257:SSL alert number 40
139809943877536:error:1409E0E5:SSL routines:SSL3_WRITE_BYTES:ssl
handshake failure:s3_pkt.c:596:
output omitted
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol : SSLv3
output truncated
```

上述输出表明，握手失败，因此无从谈及密码。

2)

```
~]$ openssl s_client -connect localhost:443 -tls1_2
CONNECTED(00000003)
depth=0 C = --, ST = SomeState, L = SomeCity, O = SomeOrganization, OU =
SomeOrganizationalUnit, CN = localhost.localdomain, emailAddress =
root@localhost.localdomain
output omitted
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Server public key is 2048 bit
```

```
Secure Renegotiation IS supported
```

```
Compression: NONE
```

```
Expansion: NONE
```

```
SSL-Session:
```

```
    Protocol  : TLSv1.2
```

```
output truncated
```

上述输出表明，握手成功，并给出了密码。

有关 `openssl s_client` 命令的更多信息，请参看 `s_client(1)` 的 man 手册。

4.1.1.9 启用 mod_nss 模块

如果想基于 `mod_nss` 模块，建立一个 HTTPS 服务器的话，就不能安装 `mod_ssl` 软件包，因为默认 `mod_ssl` 要占用端口 443。由于 443 是 HTTPS 的默认端口，所以尽可能不要占用它。

执行下列命令，可以删除 `mod_ssl` 软件包：

```
~]# yum remove mod_ssl
```

备注：

如果因其它目的还需要使用 `mod_ssl` 的话，则要将 `/etc/httpd/conf.d/ssl.conf` 文件中的端口号改为非 443，以防 `mod_ssl` 和 `mod_nss` 在监听端口上发生冲突。一个模块仅能拥有一个端口，因此，为了使 `mod_ssl` 和 `mod_nss` 能共存，就需要为它们配置各自独立的端口。正因为如此，`mod_nss` 的默认端口设为了 8443，但是，HTTPS 的默认端口仍为 443。在配置文件中，除了在虚拟主机名或地址中设置外，还可以由 `Listen` 指示项来设置端口。

NSS 中的一切都与“令牌”有关。NSS 软令牌保存在 NSS 数据库中，但是，也可以在证书中含有物理令牌。使用 OpenSSL 时，离散的证书和私钥保存在了 PEM 文件中；使用 NSS 时，这些将保存于数据库中。每个证书和密钥都和令牌相关，每个令牌可以有一个密码保护它。这个密码是可选的，但是，如果使用了密码，为了在系统引导时无需用户干预，就能打开数据库，则 Apache HTTP 服务器就需要它的一个拷贝。

实例：配置 mod_nss

- 1) 由 root 用户安装 mod_nss

```
~]# yum install mod_nss
```

这将创建 mod_nss 的配置文件/etc/httpd/conf.d/nss.conf。在/etc/httpd/conf.d/目录下,默认包含有 Apache HTTP 服务器的主配置文件。请参看 0 重启服务,实现模块加载。

- 2) 由 root 用户, 编辑/etc/httpd/conf.d/nss.conf, 检查 Listen 指示项。编辑 Listen 8443 行, 如下所示:

```
Listen 443
```

443 是 HTTPS 的默认端口。

- 3) 编辑默认行 Virtual Host _default_: 8443, 如下所示:

```
VirtualHost _default_:443
```

如果还有非默认的虚拟主机部分, 则也需要编辑它们。最后, 保存并关闭此文件。

- 4) Mozilla NSS 保存证书于服务器的证书库中, 在/etc/httpd/conf.d/nss.conf 文件的 NSSCertificateDatabase 指示项中设置证书库位置。默认情况下, 在/etc/httpd/alias 中设置有目录。在安装时, 创建 NSS 数据库。

执行以下命令, 可以查看 NSS 数据库:

```
~]# certutil -L -d /etc/httpd/alias
```

Certificate Nickname	Trust
Attributes	
SSL,S/MIME,JAR/XPI	
cacert	
CTu,Cu,Cu	
Server-Cert	u,u,u

alpha

u,pu,u

上述命令输出中，Server-Cert 默认为 NSSNickname。-L 选项可列出证书库中的全部证书及一个证书的相关信息。-d 选项指定包含了证书的证书库目录及密钥库文件。有关更多的参数选项，请参看 certutil(1)的 man 手册。

- 5) 为了配置 mod_nss 使用非默认库，可以编辑/etc/httpd/conf.d/nss.conf 文件的 NSSCertificateDatabase 指示项来设定。在默认文件中的 VirtualHost 部分有如下所示的描述行：

```
# Server Certificate Database:
# The NSS security database directory that holds the certificates and
# keys. The database consists of 3 files: cert8.db, key3.db and secmod.db.
# Provide the directory that these files exist.
NSSCertificateDatabase /etc/httpd/alias
```

上述命令输出中，alias 是默认的 NSS 数据库所在目录/etc/httpd/alias/。

- 6) 可以用以下命令为 NSS 证书数据库设置密码：

```
~]# certutil -W -d /etc/httpd/alias
Enter Password or Pin for "NSS Certificate DB":
Enter a password which will be used to encrypt your keys.
The password should be at least 8 characters long,
and should contain at least one non-alphabetic character.

Enter new password:
Re-enter password:
Password changed successfully.
```

- 7) 在部署 HTTPS 服务器前，使用证书颁发机构（CA）签署的证书创建一个新的证书数据库。

实例：添加一个证书到 Mozilla NSS 数据库

使用以下 certutil 命令，可以添加一个 CA 证书到 NSS 数据库中：

```
certutil -d /etc/httpd/nss-db-directory/ -A -n "CA_certificate" -t CT,, -a -i  
certificate.pem
```

上述命令添加一个证书到 *certificate.pem* 文件中。-d 选项用于指定证书库及密钥库文件所在目录；-n 选项用于指定证书名；-t CT 表示证书为可用于 TLS 客户端和服务器的可信证书。-A 选项表示添加一个已有证书到证书库；如果证书数据库不存在，则会创建它。-a 选项表示允许使用 ASCII 格式的输入或输出。-i 选项表示允许将 *certificate.pem* 文件作为命令参数使用。有关 *certutil(1)* 的更多参数选项，请参看它的 man 手册

8) 为了密钥的安全性，为 NSS 数据库设置了密码保护。

实例： 为 Mozilla NSS 数据库设置一个密码

使用 *certutil* 工具能为 Mozilla NSS 数据库设置一个密码：

```
certutil -W -d /etc/httpd/nss-db-directory/
```

例如：由 root 用户执行以下命令为默认数据库设置一个密码：

```
~]# certutil -W -d /etc/httpd/alias  
Enter Password or Pin for "NSS Certificate DB":  
Enter a password which will be used to encrypt your keys.  
The password should be at least 8 characters long,  
and should contain at least one non-alphabetic character.  
  
Enter new password:  
Re-enter password:  
Password changed successfully.
```

9) 通过修改 *NSSPassPhraseDialog* 指示项，可以让 *mod_nss* 使用 NSS 内置软令牌。

```
~]# vi /etc/httpd/conf.d/nss.conf  
NSSPassPhraseDialog file:/etc/httpd/password.conf
```

这可以避免在系统启动时，需要手动输入密码。软令牌存在于 NSS 数据库中，但是，在证书中也还包含有一个物理令牌。

- 10) 如果包含在 NSS 数据库中的 SSL 服务器证书是一个 RSA 证书，则要确保 NSSNickname 参数未被注释掉，并与第 4 步所显示 nickname 的相一致。

```
~]# vi /etc/httpd/conf.d/nss.conf  
NSSNickname Server-Cert
```

如果包含在 NSS 数据库中的 SSL 服务器证书是一个 ECC 证书，则要确保 NSSECCNickname 参数未被注释掉，并与第 4 步所显示 nickname 的相一致。

```
~]# vi /etc/httpd/conf.d/nss.conf  
NSSECCNickname Server-Cert
```

要确保 NSSCertificateDatabase 参数未被注释掉，并指出在第 4 步中显示的 NSS 数据库目录，或者上述第 5 步中的配置。

```
~]# vi /etc/httpd/conf.d/nss.conf  
NSSCertificateDatabase /etc/httpd/alias
```

可以用所需证书目录替代/etc/httpd/alias。

- 11) 由 root 用户，创建/etc/httpd/password.conf 文件

```
~]# vi /etc/httpd/password.conf
```

按照下列格式所示，添加一行：

```
internal:password
```

用上述第 6 步中为 NSS 安全数据库设置的密码替代此处的密码。

- 12) 为/etc/httpd/password.conf 文件设置合适的所属和权限

```
~]# chgrp apache /etc/httpd/password.conf  
~]# chmod 640 /etc/httpd/password.conf  
~]# ls -l /etc/httpd/password.conf  
-rw-r-----. 1 root apache 10 Dec  4 17:13 /etc/httpd/password.conf
```

- 13) 在/etc/httpd/password.conf 中，为了让 mod_nss 使用 NSS 软令牌，可以编辑 /etc/httpd/conf.d/nss.conf 文件，如下所示：

```
~]# vi /etc/httpd/conf.d/nss.conf
```

- 14) 请参看 0 重启服务，重启 Apache 服务器使修改生效。

在 `mod_nss` 中启用和禁用 SSL 和 TLS

为了启用和禁用指定版本的 SSL 和 TLS 协议，既可以在配置文件中，通过在“`## SSL Global Context`”部分添加/删除 `SSLProtocol` 指示项来全局启用/禁用 SSL，也可以在每个“`VirtualHost`”的“`# SSL Protocol`”中单独编辑默认项来实现。如果没有在每个域的主机部分指定它，则将会继承全局部分的设置。为了禁用一个协议版本，管理员既可以仅在“`SSL Global Context`”部分来指定 `SSLProtocol`，也可以在每个域的虚拟主机部分都指定它，注意要带上参数 `all`。

实例： 在 `mod_nss` 中，禁用除 TLS1 及更高版本以外的所有 SSL 和 TLS 协议

使用以下过程，可以禁用除 TLSv1 或更高版本以外的所有 SSL 和 TLS 协议

- 1) 由 `root` 用户，打开 `/etc/httpd/conf.d/nss.conf` 文件，搜索针对所有实例的 `NSSProtocol` 指示项。默认情况下，配置文件中只包含了一个，如下所示：

```
~]# vi /etc/httpd/conf.d/nss.conf

#   SSL Protocol:
output omitted

#   Since all protocol ranges are completely inclusive, and no protocol in the
#       middle of a range may be excluded, the entry "NSSProtocol
SSLv3,TLSv1.1"
#   is identical to the entry "NSSProtocol SSLv3,TLSv1.0,TLSv1.1".
NSSProtocol SSLv3,TLSv1.0,TLSv1.1
```

这一部分是在 `VirtualHost` 内。

- 2) 编辑 `NSSProtocol` 行，如下所示：

```
#   SSL Protocol:

NSSProtocol TLSv1.0,TLSv1.1
```

针对每个 `VirtualHost`，重复这个操作。

- 3) 编辑 `Listen 8443` 行，如下所示：

```
Listen 443
```

- 4) 编辑默认的 VirtualHost _default_:8443 行，如下所示：

```
VirtualHost _default_:443
```

如果还有非默认的虚拟主机部分，则也需要编辑它们。最后，保存并关闭此文件。

- 5) 执行以下命令，验证针对 NSSProtocol 指示项的所有修改都已正确完成：

```
~]# grep NSSProtocol /etc/httpd/conf.d/nss.conf

# middle of a range may be excluded, the entry "NSSProtocol
SSLv3,TLSv1.1"

# is identical to the entry "NSSProtocol SSLv3,TLSv1.0,TLSv1.1".

NSSProtocol TLSv1.0,TLSv1.1
```

如果有更多的 VirtualHost 部分，则这一步验证就显得格外重要。

- 6) 使用以下命令，重启 Apache 守护进程

```
~]# service httpd restart
```

注意任一会话都将被中断。

实例：在 mod_nss 中检查 SSL 和 TLS 协议的状态

在 mod_nss 中，可以使用 openssl s_client -connect 命令，验证启用了/禁用了的 SSL 和 TLS 的版本。由 root 用户安装 openssl 软件包：

```
~]# yum install openssl
```

openssl s_client -connect 命令格式如下所示：

```
openssl s_client -connect hostname:port -protocol
```

这里：*port* 是待测试端口；*protocol* 是待测试协议版本。只需将 localhost 设为主机名，就可以检查本地的 SSL 服务器是否正在运行。例如：为了检查是否在默认端口上建立了安全的 HTTPS 连接，需要查看 443 端口及验证 SSLv3 是否启用，完成这个检查任务，需要执行如下命令：

- 1)

```
~]# openssl s_client -connect localhost:443 -ssl3

CONNECTED(00000003)

3077773036:error:1408F10B:SSL routines:SSL3_GET_RECORD:wrong version
```

```
number:s3_pkt.c:337:
output omitted
New, (NONE), Cipher is (NONE)
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol   : SSLv3
output truncated
```

上述输出表明，握手失败，因此无从谈及密码。

2)

```
~]$ openssl s_client -connect localhost:443 -tls1
CONNECTED(00000003)
depth=1 C = US, O = example.com, CN = Certificate Shack
output omitted
New, TLSv1/SSLv3, Cipher is AES128-SHA
Server public key is 1024 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol   : TLSv1
output truncated
```

上述输出表明，握手成功，并给出了密码。

有关 `openssl s_client` 命令的参数文档，请参看 `s_client(1)` 的 man 手册。

4.1.1.10 使用存在的密钥和证书

可以用已有密钥和证书来配置 SSL 服务器，而无需创建新的。但是，也有 2

种情形是不可以的：

1. 正在修改 IP 或域名

证书是针对特定的 IP 和域名而生成的。因此，只要其中之一发生了变化，证书就会无效。

2. 正在更改由 VeriSign 颁发了证书的服务器软件

VeriSign 是一个常用的证书颁发机构，它会针对特定的软件，IP 地址和域名颁发证书。一旦更改了软件产品，证书就会无效。

无论上述哪种情况发生，都需要重新生成证书。有关更多相关信息，请参看 4.1.1.10 生成新密钥和证书。

如果要使用已有的密钥和证书，则要分别迁移相关文件到/etc/pki/tls/private/和 /etc/pki/tls/certs/目录下。可以执行以下命令实现：

```
~]# mv key_file.key /etc/pki/tls/private/hostname.key
~]# mv certificate.crt /etc/pki/tls/certs/hostname.crt
```

然后，在/etc/httpd/conf.d/ssl.conf 配置文件中，添加下列行：

```
SSLCertificateFile /etc/pki/tls/certs/hostname.crt
SSLCertificateKeyFile /etc/pki/tls/private/hostname.key
```

为了使更新的配置马上生效，请参看 0。

例如：使用来自 NeoKylin 安全 web 服务器的密钥和证书，如下所示：

```
~]# mv /etc/httpd/conf/httpsd.key /etc/pki/tls/private/penguin.example.com.key
~]# mv /etc/httpd/conf/httpsd.crt /etc/pki/tls/certs/penguin.example.com.crt
```

4.1.1.11 生成新密钥和证书

为了生成新密钥和证书，在系统中必须安装了 crypto-utils 软件包。由 root 用户执行以下命令可以完成安装：

```
~]# yum install crypto-utils
```

这个软件包提供了一组生成和管理 SSL 证书和密钥的工具及 genkey 应用程序，它能帮助我们生成密钥。

重要说明：

如果想用一个新证书替换掉已有的有效证书，那就只需指定一个不同的序列号。这样，可以确保客户端浏览器知晓证书更新了，避免访问页面出错。为了用自定义序列号创建新证书，需要由 root 用户用下列命令替代 genkey 命令：

```
~]# openssl req -x509 -new -set_serial number -key hostname.key -out  
hostname.crt
```

备注：

在系统中，如果有针对特指主机名的密钥文件存在的话，则执行 genkey 应用程序会失败。此时，就要由 root 用户执行以下命令，删除此密钥文件：

```
~]# rm /etc/pki/tls/private/hostname.key
```

由 root 用户执行 genkey 应用程序时，需要为其选择一个合适的主机名（例如：penguin.example.com），如下所示：

```
~]# genkey hostname
```

执行以下步骤，可以生成密钥和证书：

- 1) 查看密钥和证书的存放目录。

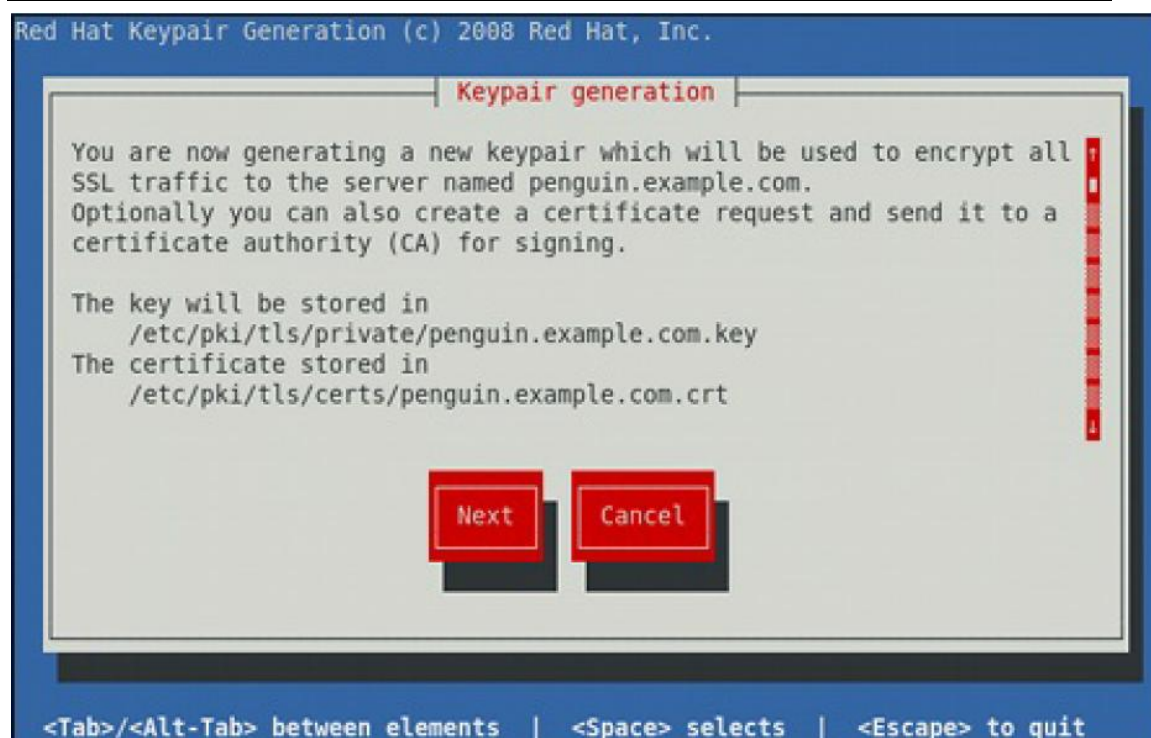


图 4-1 运行 genkey 应用程序

用 Tab 键导航到 Next 按钮，按下 Enter 键，进入下一屏。

- 2) 用 up 和 down 方向键，选择一个合适的密钥长度。注意：密钥越长越安全，当然服务器的响应时间也会加长。NIST 推荐密钥用 2048 位。更多信息，请参看 NIST Special Publication 800-131A。

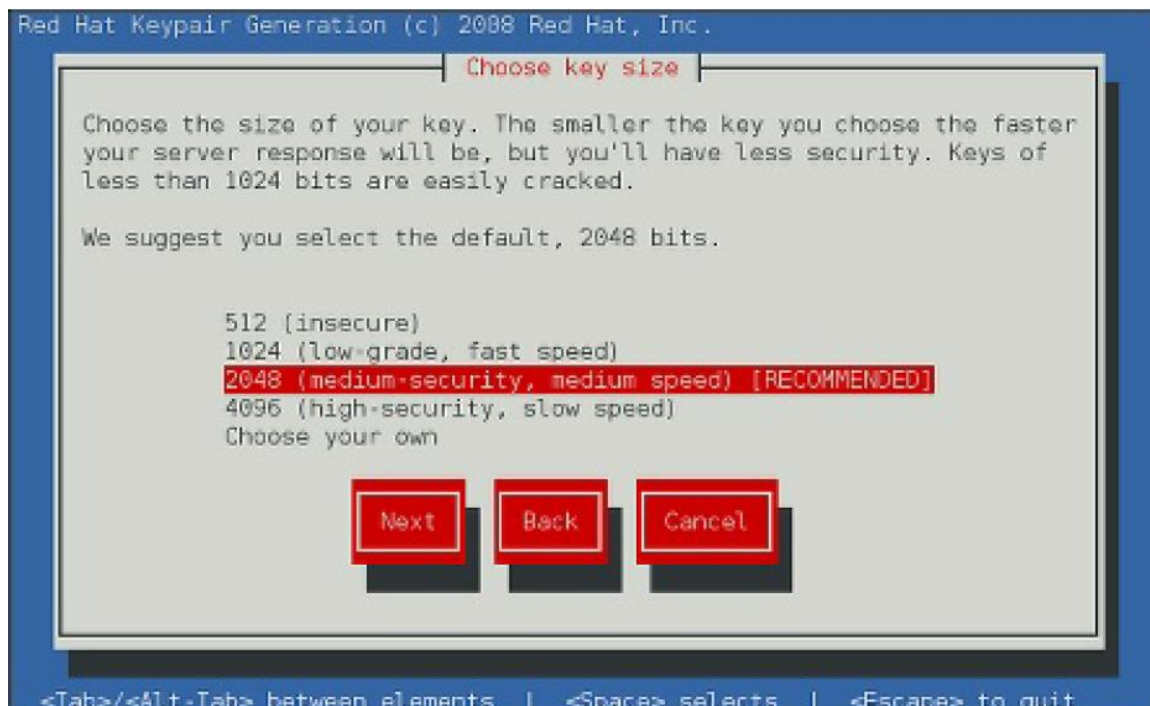


图 4-2 选择密钥长度

一旦完成，用 Tab 键导航到 Next 按钮，按下 Enter 键，开始生成密钥。根据所选密钥长度的不同，需要花费不同的时间来生成密钥。

3) 决定是否要将证书请求发送到证书颁发机构



图 4-3 生成证书请求

用 Tab 键导航到 Yes 按钮或 No 按钮。选择 Yes 生成证书请求；选择 No 生成自签名证书，然后，按下 Enter 键，再次确认你的选择。

4) 用 Spacebar 键选择启用 ([*]) 或禁用 ([]) 私钥加密

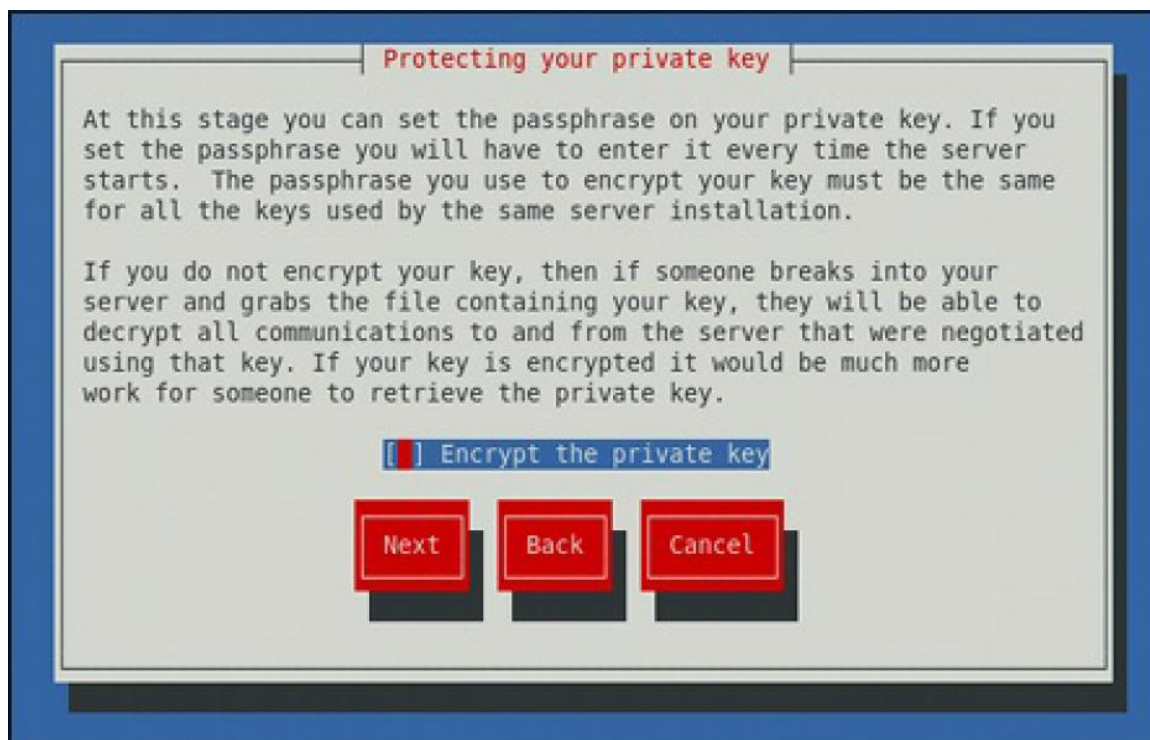


图 4-4 加密私钥

用 Tab 键导航到 Next 按钮，按下 Enter 键，进入下一屏。

5) 如果启用了私钥加密，那就要输入一个适当的密码。由于保密的原因，输入密码时，不会显示出来的。密码至少要有 5 个字符长。

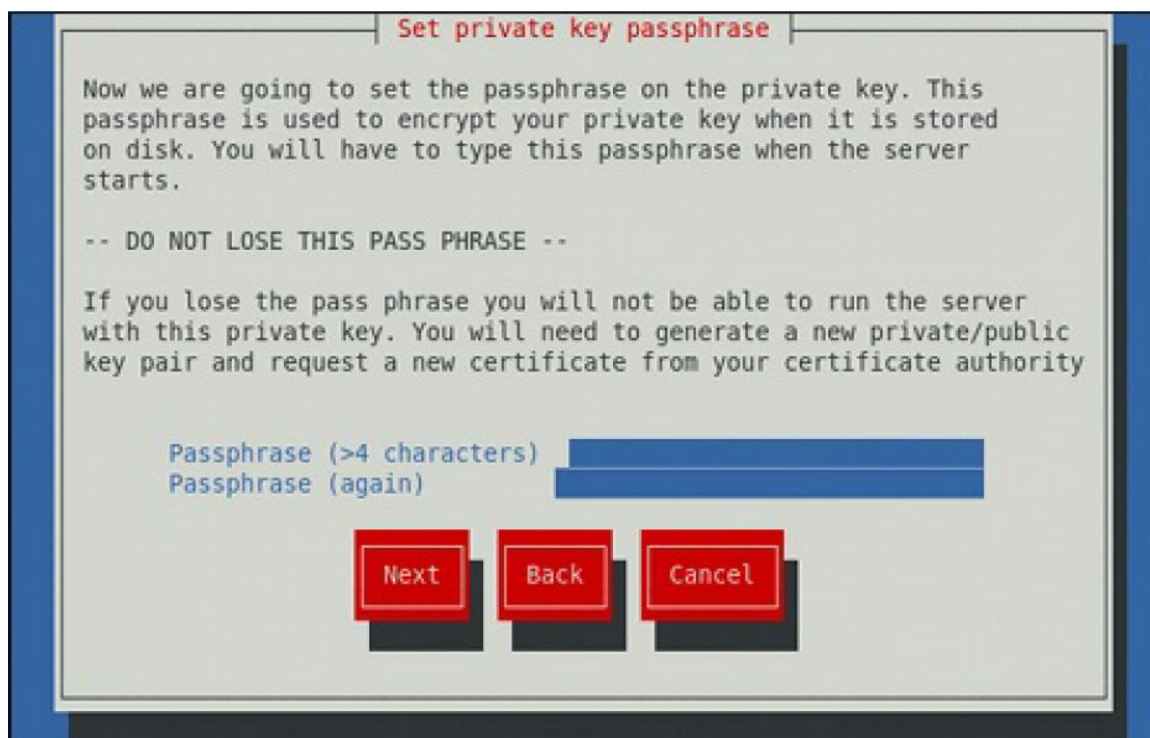


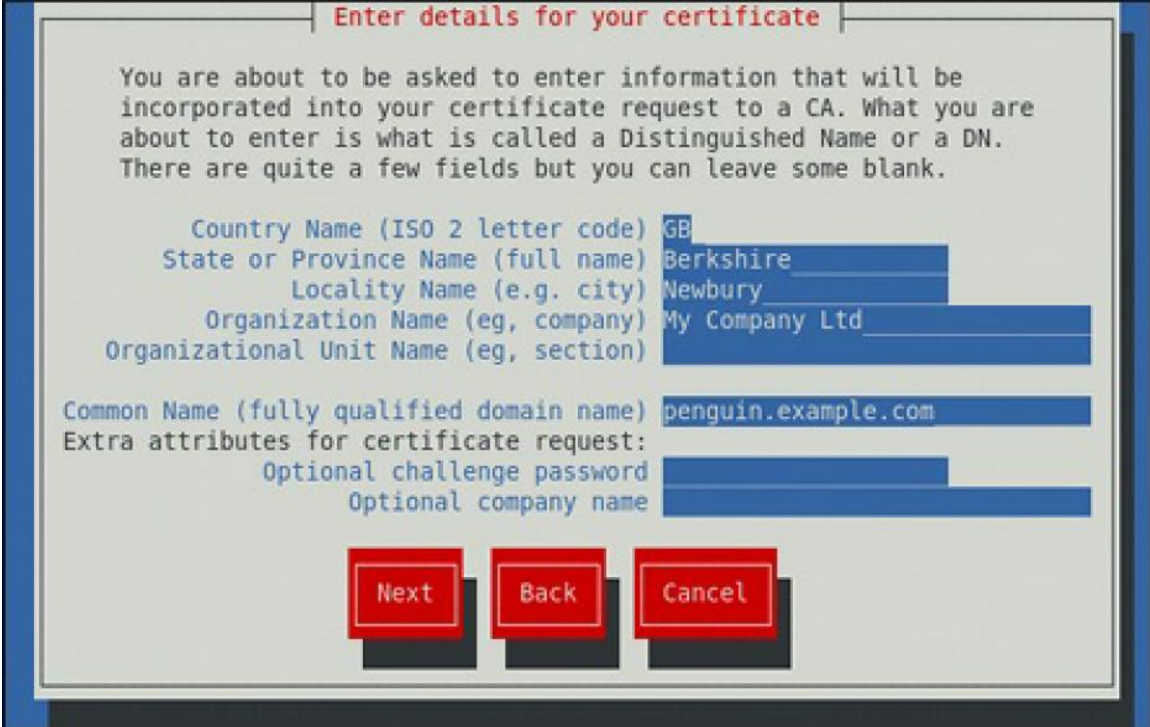
图 4-5 输入密码

用 Tab 键导航到 Next 按钮，按下 Enter 键，进入下一屏。

重要说明：

服务器启动时需要输入正确的密码。一旦密码丢失，那就需要重新生成密钥和证书了。

6) 自定义证书详细信息



Enter details for your certificate

You are about to be asked to enter information that will be incorporated into your certificate request to a CA. What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank.

Country Name (ISO 2 letter code) GB
 State or Province Name (full name) Berkshire
 Locality Name (e.g. city) Newbury
 Organization Name (eg, company) My Company Ltd
 Organizational Unit Name (eg, section)
 Common Name (fully qualified domain name) penguin.example.com
 Extra attributes for certificate request:
 Optional challenge password
 Optional company name

Next
Back
Cancel

图 4-6 指定证书信息

用 Tab 键导航到 Next 按钮，按下 Enter 键，生成密钥。

7) 如果前面选择了生成证书请求，则会提示将证书发送到证书颁发机构。

```

You now need to submit your CSR and documentation to your certificate
authority. Submitting your CSR may involve pasting it into an online
web form, or mailing it to a specific address. In either case, you
should include the BEGIN and END lines.

-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBqjCCARMCAQAwajELMAkGA1UEBhMCR0Ix EjAQBgNVBAgTCUJlcmtzaGlyZTEQ
MA4GA1UEBxMHTmV3YnVyeTEXMBUGA1UEChMOTXkgQ29tcGFueSBMdGQxHDAaBgNV
BAMTE3BlbmdlaW4uZXhhbXBsZS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJ
AoGBAjjw8bXq7WKGXNZsNZltEe9849wUMc4uAh+X8251b8x+ptJQCanGeNhLlXU
xiL5srY2TjoTSQ5DvyFgPQmFfe3cn7v//bKNgNqd4h0EbRFGaj/hDUG3fXnujukX
hP+9iY/eIAQZLHQsKABh/2egtIllpfDeRvsTUX376TnkIWLhAgMBAAGgADANBgkq
hkiG9w0BAQQFAAQBQgBUTjgjcnts1hZK070c5j+b4IfsBCwm4lnvGx3j0wpLdRq/
rHpx5cbHV99vcKnF3CwDrze9DgpTdjdbAccSCVgSG5GE8JZXWYD8EK8p2naJNQL1
YVX1KPiSMPLZuZ9cTb+k4K0cbug0IQiYaKNLNI/0zLE1VEWZXYFX0UBFM2gXYw==
-----END NEW CERTIFICATE REQUEST-----

A copy of this CSR has been saved in the file
/etc/pki/tls/certs/penguin.example.com.1.csr

Press return when ready to continue
    
```

图 4-7 说明如何发送证书请求

按下 Enter 键，返回到 shell 提示符下。

一旦生成了密钥和证书，就会在/etc/httpd/conf.d/ssl.conf 配置文件中添加密钥和证书的全路径名。如下所示：

```
SSLCertificateFile /etc/pki/tls/certs/hostname.crt
SSLCertificateKeyFile /etc/pki/tls/private/hostname.key
```

最后，为了使更新立即生效，请参看 0 重启服务，重启 httpd 服务。

4.1.1.12 为 HTTP 和 HTTPS 配置防火墙

默认情况下,NeoKylin Linux Advanced Server 是不启用 HTTP 和 HTTPS 的。把系统配置成一个 web 服务器后，利用 firewalld 服务，通过防火墙，就可以启用 HTTP 和 HTTPS 了。

由 root 用户执行以下命令，可以启用 HTTP：

```
~]# firewall-cmd --add-service http
success
```

由 root 用户执行以下命令，可以启用 HTTPS：

```
~]# firewall-cmd --add-service https
success
```

注意系统重启后，这些修改就失效了。为了使这些修改永久生效，只需要在执行上述命令时，加上--permanent 选项。

检查 HTTP 和 HTTPS 的网络访问许可

由 root 用户执行以下命令，可以检查通过防火墙允许访问的那些配置：

```
~]# firewall-cmd --list-all
public (default, active)
  interfaces: em1
  sources:
  services: dhcpv6-client ssh
output truncated
```

这是一个默认安装的例子，虽然防火墙启用了，但是，HTTP 和 HTTPS 是不允许通过防火墙被访问的。

一旦允许通过防火墙可以访问 HTTP 和 HTTPS 了，那么执行上述命令，就会看到 services 行上会有如下的信息：

```
services: dhcpv6-client http https ssh
```

有关如何启用防火墙服务，或者如何使用 firewalld 打开和关闭端口的更多信息。请参看《NeoKylin Linux Advanced Server V7 安全手册》。

4.2 邮件服务器

NeoKylin Linux Advanced Server 提供了许多很好的服务和访问电子邮件的应用。本章主要介绍当今主流的电子邮件协议，以及一些发送和接收电子邮件的应用。

4.2.1 电子邮件协议

当今，大多数电子邮件系统仍是基于客户端/服务器架构设计的。由邮件客户端程序来创建一个电子邮件，并发送邮件到一个服务器，然后，这个服务器再将邮件转发给收件人的电子邮件服务器，最终，该邮件会被收件人的电子邮件客户端所接收。

在整个邮件的收发过程中，由多种标准网络协议来支持不同的机器；不同的操作系统；使用不同的邮件程序来接收和发送电子邮件。

下面讨论在电子邮件的传输过程中最常用到的协议。

4.2.1.1 邮件传输协议

邮件从客户端应用程序投递到服务器以及从源服务器投递到目的服务器，都是要遵循简单邮件传输协议（SMTP）的。

SMTP

SMTP 的主要用途是在邮件服务器间传送电子邮件。当然，邮件客户端也是很重要的。为了发送电子邮件，邮件客户端需要发送邮件到一个发送邮件服务器，为投递邮件，这个邮件服务器需要再依次联系目的邮件服务器。正因为如此，在配置电子邮件客户端时，必须要指定 SMTP 服务器。

在 NeoKylin Linux Advanced Server 上，用户可以在本地配置一个 SMTP 服务器用于处理邮件的分发。也可以配置一个远程 SMTP 服务器来发送邮件。

重要的一点就是 **SMTP** 协议不需要身份验证。这就使得互联网上任何一人可以给别人或一组人发送邮件，无形中就会产生垃圾邮件和有可能产生垃圾邮件，这也成了 **SMTP** 的一个缺憾。我们可以强加一些限制，来制约用户在互联网上通过自己的 **SMTP** 服务器或其他服务器发送电子邮件。不强加这种限制的服务器称为开放式中继服务器。

NeoKylin Linux Advanced Server V7 提供了 Postfix 和 Sendmail 两个 **SMTP** 服务器软件。

4.2.1.2 邮件访问协议

电子邮件客户端软件从邮件服务器查询邮件，主要使用的协议是：**Post Office Protocol (POP)**和 **Internet Message Access Protocol (IMAP)**。

POP

在 NeoKylin Linux Advanced Server 上，默认使用的 **POP** 服务器是 Dovecot，它由 dovecot 软件包提供。

备注：

首先要确认 dovecot 软件包是否已经在系统上安装了，否则，无法正常使用 Dovecot。使用以下命令，可以安装 dovecot 软件包。

```
~]# yum install dovecot
```

有关如何用 yum 安装软件包的更多信息，请参看 2.2.2.4 安装软件包。

当使用了 **POP** 服务器后，电子邮件会被邮件客户端软件下载到本地。默认情况下，大多数 **POP** 邮件客户端会自动配置成当电子邮件从邮件服务器下载到本地后，就自动删除它们。然而，通常是要修改这种设置的。

POP 完全兼容一些重要的互联网信息标准，如：**Multipurpose Internet Mail Extensions (MIME)**，支持电子邮件附件。

POP 主要就是支持用户能在一个系统上读取电子邮件，还有就是不需要用户长久连接到互联网上或连接到有邮件服务器的网络中。不幸的是，对那些网速慢的用户，在验证时，要下载每个信件的全部内容。如果每个信件都带有大量的附件的话，那么，这个过程会花费很长时间。

标准 **POP** 协议当前最新版本为 **POP3**。然而，还有一些的很少使用 **POP** 协

议的变化，在此，就不加以介绍了。

为了提高安全性，尽量使用 SSL 加密客户端验证和会话中传输的数据。为此，需要启用 pop3s 服务或 stunnel 应用。有关安全电子邮件通信的更多信息，请参看 4.2.5.1 安全通信。

IMAP

在 NeoKylin Linux Advanced Server 上，默认使用的 IMAP 服务器是 Dovecot。它由 dovecot 软件包提供。有关如何安装 Dovecot 的更多信息，请参看 OPOP。

当用 IMAP 邮件服务器时，电子邮件会保留在服务器上，用户可以读取或删除它们。IMAP 也允许客户端应用在服务器上通过创建，重命名或删除邮件目录来组织和存储电子邮件。

IMAP 最可取之处就是允许用户能从多台机器上查看他们的电子邮件。因为 IMAP 邮件客户端能在打开邮件前只下载电子邮件头信息，所以能接受用户通过慢速网络连接到邮件服务器，节省了带宽。用户可以不必浏览和下载邮件而直接删除信件。

IMAP 客户端应用可以在本地缓存信件的备份，这样，用户可以不用直接连到服务器上而浏览到以前读过的信件。

和 POP 一样，IMAP 也完全兼容一些重要的互联网信息标准，如： MIME，支持电子邮件附件。

为了提高安全性，尽量使用 SSL 加密客户端验证和会话中传输的数据。为此，需要启用 imaps 服务或 stunnel 应用。有关安全电子邮件通信的更多信息，请参看 4.2.5.1 安全通信。

针对其它的免费或商业情况，IMAP 客户端和服务器在 IMAP 协议上的扩展和附加功能也都是可用的。

Dovecot

由 dovecot 守护进程产生的支持 IMAP 和 POP3 协议的 imap-login 和 pop3-login 进程。可以通过配置/etc/dovecot/dovecot.conf 文件来使用 IMAP 和 POP。默认情况下，基于 SSL dovecot 会同时启用 IMAP 和 POP3。为了配置 dovecot 使用 POP，需要完成以下步骤：

- 1) 编辑/etc/dovecot/dovecot.conf 配置文件，确保含有 pop3 参数的 protocols 变量所在行没有被注释掉（即：删除 protocols 行前的#）：

```
protocols = imap pop3 lmtp
```

- 2) 由 root 用户执行下列命令使修改生效

```
~]# systemctl restart dovecot
```

- 3) 执行下列命令使在下次开始，系统重启时，能自动启动 dovecot

```
~]# systemctl enable dovecot  
  
ln -s /usr/lib/systemd/system/dovecot  
/etc/systemd/system/multi-user.target.wants/dovecot
```

备注：

Dovecot 仅仅报告了启动了 IMAP 服务器，但是，实际也启动了 POP3 服务器。

不像 SMTP，IMAP 和 POP3 请求连接客户端是需要提供用户名和密码完成验证的。默认情况下，两个协议的密码是在非加密网上传送的。

为 dovecot 配置 SSL：

- 编辑/etc/dovecot/conf.d/10-ssl.conf 配置文件，确保含有!SSLv2 !SSLv3 参数的 ssl_protocols 变量所在行未被注释掉，如下所示：

```
ssl_protocols = !SSLv2 !SSLv3
```

这个设置确保 dovecot 不使用 SSL 版本 2 和版本 3，因为它们有漏洞，所以两者都被认为是不安全的。

- 根据用户意愿，编辑/etc/pki/dovecot/dovecot-openssl.cnf 配置文件。不过，典型安装是不需要修改它的。
- 重命名，移动或删除 /etc/pki/dovecot/certs/dovecot.pem 和 /etc/pki/dovecot/private/dovecot.pem。
- 执行/usr/libexec/dovecot/mkcert.sh 脚本创建 dovecot 的自签名证书。这些证书会被拷贝到/etc/pki/dovecot/certs 和/etc/pki/dovecot/private 目录下。为了使修改生效，需要由 root 用户执行以下命令：

```
~]# systemctl restart dovecot
```

有关 dovecot 的更多信息，请参看 <http://www.dovecot.org>。

4.2.2 电子邮件软件分类

通常，所有的电子邮件应用至少都会归属于以下三类之一。在移动和管理电子邮件的信件中都充当着不同的角色。虽然大多数用户只知道他们使用的特定电子邮件软件可以用来接收和发送信件，为了确保每个邮件能正确地到达的目的地，它们中的每一个都很重要。

4.2.2.1 邮件传输代理

Mail Transport Agent (MTA) 负责基于 SMTP 在主机间传递信件。一个信件的投递可能会涉及到多个 MTA，分别作为它投递的目的地。

系统之间的信件传递看似很简单，如果要求特定的 MTA 完成邮件的分发，那么整个邮件的传递过程就复杂得多了。另外，由于垃圾邮件的问题，使用特定的 MTA 会限制其配置，或者限制对 MTA 所在网络访问的配置。

在发送电子邮件时，许多现在的电子邮件客户端都是可以充当 MTA 角色的。然而，这一功能不应该与真正的 MTA 角色相混淆。电子邮件客户端能够像 MTA 一样发送电子邮件的唯一原因就是本地没有运行自己的 MTA。这个电子邮件客户端程序在非 UNIX 的操作系统上尤其如此。然而，这些客户端程序只发送外出的邮件到它们已授权的 MTA 上，而不直接将邮件发送到收件人的邮件服务器中。

NeoKylin Linux Advanced Server 提供了两个 MTA，分别是 Postfix 和 Sendmail。电子邮件客户端常常不需要充当 MTA 角色。NeoKylin Linux Advanced Server 也包含有具有特殊用途的 MTA，如：Fetchmail。在本手册中，主要介绍 Postfix，由于篇幅原因，Sendmail 和 Fetchmail 不再介绍了。

有关 Postfix 的更多信息，请参看 4.2.3 邮件传输代理。

4.2.2.2 邮件投递代理

由 MTA 调用 Mail Delivery Agent (MDA) 归档外来的电子邮件到一个合适的用户邮箱。大多数情况下，MDA 就是一个 Local Delivery Agent (LDA)，如：mail 或 Procmail。

实际上，任一可将电子邮件分发到一个目的地，邮件客户端能从其中读取到邮件，这样的邮件应用都可以认为是一个 MDA。正因为如此，在一些 MTA（如：Sendmail 和 Postfix）追加新电子信件到本地用户的邮件文件中时，充当了 MDA 的角色。通常，MDA 不在系统间传送信件，也不提供用户界面。一个电子邮件客户端应用程序访问本地计算机上的 MDA 实现信件的分发和排序。

4.2.2.3 邮件用户代理

Mail User Agent(MUA)是一个电子邮件客户端应用的同义词。最低限度，一个 MUA 应用也要能允许用户读取和撰写电子邮件。许多 MUA 能够通过 POP 或 IMAP 协议检索信件，设置收件箱存储信件，发送外来信件到 MTA。

MUA 对应的产品很多，有基于图形接口的，如：Evolution；也有基于简单文本接口的，如：Mutt。

4.2.3 邮件传输代理

NeoKylin Linux Advanced Server V7 提供了 2 个主要的 MTA，分别是：Postfix 和 Sendmail。Postfix 被配置成了默认的 MTA，Sendmail 被认为是过时的了。如果需要也可以将 Sendmail 配置成默认的，只需要由 root 用户执行以下命令就可以完成切换，不需要删除 Postfix。

```
~]# alternatives --config mta
```

你也可以用下列命令启用所需的服务：

```
~]# systemctl enable service
```

类似地，可以用以下命令禁用此服务：

```
~]# systemctl disable service
```

在 NeoKylin Linux Advanced Server V7 中，有关如何管理系统服务的更多信息，请参看 3.1 使用 systemd 管理系统服务。由于篇幅原因，Sendmail 和 Fetchmail 不在此介绍了。

4.2.3.1 Postfix

Postfix 最初是由 IBM 安全专家，程序员 Wietse Venema 开发的，是和 Sendmail 兼容的一种安全，快速和易于配置的 MTA。

为了提高安全性，Postfix 采用了模块化的结构设计，由一个主守护进程启动

一些小权限的小程序构成其运行模式。由较小的，较低权限的进程来完成一些邮件投递不同阶段中的特殊任务。在一个变化了的环境中运行，以防攻击。

在 Postfix 的配置文件中，只需做一些小小的修改，就可以把 Postfix 配置成可接受非本地网络连接的 MTA。然而，对于那些更复杂的需求，Postfix 提供了多种配置选项，以及第三方插件，可以使它成为一个非常灵活和功能齐全的 MTA。

Postfix 的配置文件是易读的，可支持超过 250 个的指示项。

4.2.3.1.1 默认安装 Postfix

Postfix 的可执行文件是 postfix。为了处理邮件的分发，postfix 守护进程需要启动一些相关进程。

Postfix 存储它的配置文件在/etc/postfix/目录下。下面列出了一些 Postfix 的常用文件：

- access --- 访问控制文件。在此文件中，配置了允许连接到 Postfix 的主机。
- main.cf --- 全局配置文件。主要的配置选项都在此文件中描述。
- master.cf --- 描述 Postfix 如何与各种相关进程协力完成邮件的分发。
- transport --- 将电子邮件地址映射到中继主机。

Postfix 和 Sendmail 可以共享存放于/etc/目录下的 aliases 文件。依据邮件协议要求所描述的可配置用户 ID 别名列表。

重点：

/etc/postfix/main.cf 不允许 Postfix 接受来自非本地的其它主机连接。如何将 Postfix 配置成邮件服务器，以便为更多客户端提供服务，请参看 12.3.1.3 Postfix 基本配置。

为了使/etc/postfix/目录下配置文件的变更生效，在完成修改后需要重启 postfix 服务，为此，需执行以下命令：

```
~]# systemctl restart postfix
```

4.2.3.1.2 升级

在 NeoKylin Linux Advanced Server V7 中，下列的设置不同于前期版本。

- disable_vrfy_command = no --- 默认禁用。如果设成 Yes，则可以禁用某些电子邮件地址的获取方法。

- `allow_percent_hack = yes` --- 默认启用。它允许删除邮件地址中的%字符。%黑客是一个传统的解决方法，允许由发送电子邮件的路由来控制。目前DNS 和邮件路由都相对比较可靠，但是，Postfix 仍然会被黑客继续利用。为了关闭%重写，需要设置 `allow_percent_hack = no` 。
- `smtpd_helo_required = no` --- 默认禁用。设置成 `yes` 后，就要求客户端在试图发送 MAIL, FROM 或 ETRN 命令之前要先发送 HELO 或 EHLO 命令。

4.2.3.1.3 Postfix 基本配置

默认情况下，Postfix 不接受来自非本地的其它主机连接。由 root 用户完成以下配置后，Postfix 就可以在网络中为其它主机分发邮件。

- 用文本编辑器（如：vi）编辑/etc/postfix/main.cf 文件。
- 取消 `mydomain` 行的注释（即：删除#），用邮件服务器所在域的域名替代 *domain.tld*，如：example.com。
- 取消 `myorigin = $mydomain` 行的注释
- 取消 `myhostname` 行的注释，用机器的主机名替代 *host.domain.tld* 。
- 取消 `mydestination = $myhostname, localhost.$mydomain` 行的注释
- 取消 `mynetworks` 行的注释，用能连到服务器主机的有效网络配置替代 *168.100.189.0/28* 。
- 取消 `inet_interfaces = all` 行的注释
- 注释掉 `inet_interfaces = localhost`
- 重启 postfix 服务

一旦完成了上述配置，主机就可实现外部电子邮件的分发了。

Postfix 有各种配置选项，学习如何配置 Postfix 的最好方法之一就是看 /etc/postfix/main.cf 配置文件的注释。其它相关的配置，整合 SpamAssassin 或 /etc/postfix/main.cf 中参数的更多描述，请参看官方 <http://www.postfix.org/> 在线帮助。

4.2.3.1.4 Postfix 与 LDAP 结合使用

Postfix 能把 LDAP 目录作为一种查询源（如：别名，虚拟的或规范等等），

这样，可以使用上 LDAP 的分层存储用户信息。在需要时，Postfix 可以方便地获得 LDAP 的查询结果，无需在本地保存这些信息，管理员也可以很容易地维护它。

4.2.3.1.5 /etc/aliases 查询实例

下例是使用 LDAP 查询/etc/aliases 文件的一个基本实例。首先确认/etc/postfix/main.cf 文件包含有以下内容：

```
alias_maps = hash:/etc/aliases, ldap:/etc/postfix/ldap-aliases.cf
```

确认/etc/postfix/ldap-aliases.cf 文件含有以下内容，否则，需要创建一个，并完成正确配置：

```
server_host = ldap.example.com
search_base = dc=example, dc=com
```

这里，ldap.example.com，example 和 com 都需要用当前使用的 LDAP 服务器的配置去替换。

备注：

在/etc/postfix/ldap-aliases.cf 文件中，可以指定各种参数，包括：启用 LDAP SSL 和 STARTTLS。更多相关信息，请查看 4.3.1 OpenLDAP 。

4.2.3.2 配置邮件传输代理

*Mail Transport Agent (MTA)*是发送邮件必不可少的，*Mail User Agent (MUA)*（如：Evolution 或 Mutt）是读取和撰写邮件的常用工具。当用户从 MUA 发送一封电子邮件时，信件首先被传送到 MTA，然后，再经过一系列 MTA 的转发，最终将信件送到目的地。

即使用户不打算从系统发送电子邮件，但是，有一些定时的自动管理任务或系统程序也会用 mail 命令给本地的 root 用户发送一些电子邮件的，如：发送日志信息和报警信息等。

NeoKylin Linux Advanced Server V7 提供了 2 个主要的 MTA，分别是：Postfix 和 Sendmail。Postfix 被配置成了默认的 MTA。Sendmail 已是过时的了。

4.2.4 邮件投递代理

NeoKylin Linux Advanced Server 提供了 2 个主要的 MDA，分别是：Procmail

和 mail。它们都被认为是 LDAs，都能将 MTA 邮件文件中的电子邮件移动到用户的邮箱中。然而，procmail 提供了更强大的过滤功能。

本节仅详细介绍了 procmail。有关 mail 命令的更多信息，请参看它的 man 手册。

Procmail 能投递和过滤存放于本地邮件文件中电子邮件。它是一个强大的系统资源，被广泛使用。在投递电子邮件中，Procmail 发挥着重要的作用。

可以用几种不同的方法来调用 Procmail。每当 MTA 投放一封电子邮件到邮件文件中，Procmail 就会被调用。然后，Procmail 过滤器和投寄来自于 MUA 的电子邮件，最终，退出。另外，MUA 也能被配置成只要接收到信件，就马上启动 Procmail，以便这些信件能被尽快地投寄到正确的邮箱中。默认情况下，每当 MTA 收到一封新信件，就会调用 Procmail，然后，在用户的主目录里创建 /etc/procmailrc 或 ~/.procmailrc（也称为 rc 文件）文件。

默认情况下，非系统的 rc 文件存放在 /etc 目录中；非 .procmailrc 文件存放在用户的主目录中。因此，为了使用 Procmail，每个用户必须构建一个 .procmailrc 文件，它包含有：设置的特定环境变量和规则描述。

Procmail 对电子邮件所起到的作用，完全依赖于信件在 rc 文件中所匹配的条件或规则。如果一封信件匹配了其中的一个规则，则这个电子邮件就会被放到指定文件中，或被删除，否则就会被处理。

当 Procmail 启动时，首先会读取电子邮件，从头部信息中分离出信件正文；其次，Procmail 会去查找 /etc/procmailrc 文件和 /etc/procmailrcs 目录下的 rc 文件，获取默认配置，系统范围，Procmail 环境变量及规则；然后，在用户主目录下搜索 .procmailrc 文件。许多用户还为 Procmail 创建了其他的 rc 文件，主要是指它们主目录中的 .procmailrc 文件。

4.2.4.1 配置 Procmail

Procmail 的配置文件主要包含一些重要的环境变量。这些变量特指一些操作规则，如：分类的信件和没有匹配上任何规则的信件该如何处理等等。

这些环境变量通常以下列形式出现在 ~/.procmailrc 文件的开头。

```
env-variable="value"
```


此例中，*env-variable* 为环境变量名；*value* 为变量值。

有许多环境变量 Procmail 用户是不使用的；重要的环境变量都已经被定义成了默认值。大多数情况下，会用到下列变量：

- **DEFAULT** --- 在信件不能匹配到任何设定的规则时，就使用它作为的默认收件箱。默认值和\$ORGMAIL 相同
- **INCLUDEDRC** --- 包含有一些额外的 rc 文件，这些文件含有针对信件检查用的许多规则。这就会将 Procmail 规则列表分解成充当不同角色的单个文件，如：拦截垃圾邮件及管理电子邮件列表。可以关掉它，或者用#注释掉用户 ~/.procmailrc 文件中的相关行。

例如：在用户的 ~/.procmailrc 文件中可能会如下一些行：

```
MAILDIR=$HOME/Msgs
INCLUDEDRC=$MAILDIR/lists.rc
INCLUDEDRC=$MAILDIR/spam.rc
```

用#注释掉第一行 INCLUDEDRC，这样，可以关掉 Procmail 中的电子邮件过滤列表，保留下垃圾邮件控制部分。注意：它用的是当前目录的相对路径。

- **LOCKSLEEP** --- 以秒为单位设置时间量。试图在使用一个特定的锁文件。默认值为 8 秒。
- **LOCKTIMEOUT** --- 以秒为单位设置时间量。必须在 Procmail 设定的锁文件过期被删除前，在规定时间内，完成锁文件的更新。默认值为 1024 秒。
- **LOGFILE** --- Procmail 的所有信息及错误日志都记录于此文件中。
- **MAILDIR** --- 设置 Procmail 的当前工作目录。一旦设置了，则 Procmail 的所有其它路径都是相对于它而设定。
- **ORGMAIL** --- 指定原始邮箱，或者，一旦信件不能投放到默认的和规则指定的邮箱时，就投递于此。
默认设定为： /var/spool/mail/\$LOGNAME
- **SUSPEND** --- 以秒为单位设置时间量。一旦系统资源（如：swap）无效了，Procmail 就会在此时间量过后被挂起。
- **SWITCHRC** --- 允许用户指定一个包含有 Procmail 规则的外部文件。很像

INCLUDERC 选项。除了在参考配置文件上的，规则检查实际是停止的，仅仅使用了 SWITCHRC 指定的文件。

➤ **VERBOSE** --- 产生过多的日志文件。这个选项对于调试是很有用的。

其它的重要环境变量，可以从 shell 中查到。例如：**LOGNAME** 设定登录名；**HOME** 设定主目录；**SHELL** 设定默认所使用的 shell。

所有环境变量的综合解释以及它们的默认值，可以参看 procmailrc 的 man 手册。

4.2.4.2 Procmail 规则

在使用 Procmail 中，新用户常常会在学习规则时遇到一些困难。最难学的部分其实就是和信件匹配的规则属性，也就是如何使用正则表达式来指定字符串匹配的条件。然而，正则表达式并不是很难编写，也不是很难理解。此外，Procmail 规则编写方式上的一致性，借助样例的学习，就会容易得多了。学习样例，了解 Procmail 规则，请参看 0 规则实例。

Procmail 规则采用以下格式：

```
:0 [flags] [: lockfile-name ]  
  
* [ condition_1_special-condition-character condition_1_regular_expression ]  
* [ condition_2_special-condition-character condition-2_regular_expression ]  
* [ condition_N_special-condition-character condition-N_regular_expression ]  
  
    special-action-character  
  
    action-to-perform
```

Procmail 规则的前两个字符是冒号和零。在零之后的各种标识用于表示 Procmail 进程是如何执行这些规则的。flags 部分后的冒号指出由信件创建的锁文件。一旦创建了锁文件，就能用此文件名替代 *lockfile-name*。

规则可以包含一些针对信件匹配的条件。如果它没有指定条件，则每个信件都可以匹配它。正则表达式中放置了一些条件，以便信件匹配。如果使用多个条件，则必须完成所有匹配的操作。基于规则文件第一行中的设置的标识来检查条件。可选的特殊字符将放置于星号（*）之后，用于进一步限制条件。

action-to-perform 参数表示一旦信件匹配上了某个条件后，需要采取的操作。

每个规则只能对应一个操作。多数情况下，邮箱名称被直接用于匹配信件需要进入哪个文件，以便有效地排序电子邮件。在指定的操作之前，还可以使用一些特殊的操作字符。更多信息请参看 0 特殊条件和操作。

投递和非投递规则

如果规则匹配了一封特殊的信件，那么对应的操作决定了是被认为投递规则还是非投递规则。一个投递规则包含有写信件到文件，发送信件到另一个程序，或者将信件转发到另一个电子邮件地址。一个非投递规则将覆盖任一其它操作，例如：嵌套块。一个嵌套块包含一组用{}括起来的操作，这组操作是指信件匹配了规则后，需要完成的相应操作。嵌套块可以嵌套在另一个里面，以便提供更多的控制来识别和操作信件。

当信件匹配了一个投递规则后，Procmail 就会完成其特指的操作，并停止针对其他规则的比较。如果信件匹配上的是非投递规则，则将会针对其他规则继续做比较。

标识

标识是必要的，以确定如何或者是否需要把一个规则的条件和信件做比较。

egrep 是用于内部条件匹配的工具。下列是一些常用的标识：

- **A** --- 指定此规则仅用于如果以前的规则不带 **A** 或者有一个标志也匹配此信件。
- **a** --- 指定此规则仅用于如果以前的规则带有 **A** 或者有一个标志也匹配此信件并且能成功完成。
- **B** --- 分析信件正文，找寻匹配条件。
- **b** --- 用于任一操作的信件正文，例如：写信件到一个文件或转发它。这是默认设置。
- **c** --- 生成电子邮件的副本。此项针对投递规则尤为重要。由于在邮件上有需要完成的操作，在 **rc** 文件中继续信件的拷贝。
- **D** --- 使 **egrep** 能区分大小写。默认设置，比较处理不识别大小写。
- **E** --- 类似于 **A** 标识。如果不带有 **E** 标识的立即处理规则没有被匹配，则规则中的条件仅仅和信件比较。当然，也能与其他操作相比。

- e --- 如果在立即处理规则中的操作失败了, 那么这个规则就完成了和信件的比较。
- f --- 用管道做过滤。
- H --- 分析信件头, 找寻匹配条件。这是默认设置。
- h --- 在产生的操作中使用头。这是默认设置。
- w --- 告诉 Procmail 需要等待指定的过滤和程序的完成。在考虑信件过滤前, 要报告是否成功。
- W --- 除了“程序失败”信件被禁止外, 其他和 w 完全相同。

更多其他的标识描述, 请参看 procmailrc 的 man 手册。

指定本地锁定文件

在 Procmail 中, 锁文件可以确保多个进程不会尝试同时修改一个信件, 所以它非常有用。在规则第一行的任一标识位后, 由冒号来指定本地锁定文件。无论是否已经设置 LOCKEXT 全局环境变量, 都将会基于目标文件名创建一个本地锁文件。

另外, 在冒号后指定一个本地锁文件用于此规则。

特殊条件和操作

在 Procmail 规则的条件和操作前使用特殊字符以改变它们原有的解释。

在规则的条件开始行的*之后可以使用下列一些字符:

- ! --- 在条件行, 这个字符反转条件。如果条件没有和信件匹配, 则这个字符就会导致一个匹配发生。
- < --- 检查信件是否在指定的字节数下。
- > --- 检查邮件是否在指定的字节数上。

用下列字符完成指定操作:

- ! --- 在操作行, 这个字符让 Procmail 将邮件转发到指定的电子邮件地址。
- \$ --- 参照 rc 文件中该变量的早期设置。常用来设置被各种规则使用的通用收件箱。
- | --- 启动指定程序来处理信件。
- {和} --- 构建一个嵌套块, 用于存放匹配信件的其它规则。

如果在操作行，没有使用指定字符，Procmail 会假设操作行指定的是写件箱。
规则实例

Procmail 是一个非常灵活的程序，但是，由于这种灵活性，要开始就组合 Procmail 规则，对于新用户来说还是很困难的。

开发构建 Procmail 规则条件技能的最佳方式是对正则表达式有较深刻的理解，然后，再结合去学习一些别人所建的例子。对正则表达式的完整解释超出了本节的范围，在此不多描述。在互联网的许多网站上，可以找到有关 Procmail 规则结构和一些有用的样例。通过查看这些规则的实例，可以学到正确的使用和编写正则表达式的方法。此外，更多介绍基本正则表达式规则的信息，请参看 grep 的 man 手册。

下面这个简单的例子说明 procmail 规则的一个基本结构，可以提供更复杂的基础结构。

一个基本的规则甚至可以不包含任何条件，如这个例子所示：

```
:0:
new-mail.spool
```

第一行给出本地所建锁文件，但是没有给出名称，因此，Procmail 会用目标文件名并在 LOCKEXT 环境变量中追加指出。此例中没有给出条件，因此任一封信件都能匹配这个规则，被放到 new-mail . spool 文件中，该文件所在目录在 MAILDIR 环境变量中设定。一个 MUA 就能在此文件中浏览到该信件。

如上例所示的一个基本规则，常常被追加在每个 rc 文件的最后，直接投递信件到默认位置。

下面的实例匹配特定电子邮件地址的信件，然后将它们投出去。

```
:0
* ^From: spammer@domain.com
/dev/null
```

此例说明，[任何由 spammer@domain.com](mailto:spammer@domain.com) 发出的信件都将被投递到/dev/null 中，实质是将其删除。

警示：

可以肯定的是，在发送信件到/dev/null，要永久删除它之前，规则就生效了。如果一个规则无意中捕捉到了意外的信件，并且这些信件消失了，那么要排查规则错误就变得困难了。

一个更好的解决办法就是将规则的操作导向到一个特殊邮箱，这样，就可以一次次地检查，以寻找到错误点。一旦满足了没有信件会偶然被匹配，就可以删除此邮箱，并直接将发送邮件的操作定向到/dev/null。

下面的规则将抓取从特定邮件列表发送的电子邮件，并将其存放于指定的文件夹中。

```
:0:
* ^(From|Cc|To).*tux-lug
tuxlug
```

上例说明，[任何来自于 tux-lug@domain.com 邮件列表的信件都将被自动放到 tuxlug 邮箱中](#)。注意：在实例中匹配信件的条件里，来自 From, Cc 或 To 行的邮件列表的邮件地址。

垃圾邮件过滤器

通过 Postfix 来接收新邮件，而 Procmail 可以用作拦截垃圾邮件的有力工具。

尤其当 Procmail 和 SpamAssassin 结合使用时，更是如此。它们可以快速识别出垃圾邮件，并排序或销毁它们。

SpamAssassin 使用报头分析、文本分析、黑名单、垃圾邮件跟踪数据库以及自学习 Bayesian 垃圾分析法，就能快速准确地识别和标记出垃圾邮件。

注意：

为了使用 SpamAssassin，首先要确保安装了 SpamAssassin 软件包，由 root 用户执行以下命令就可以安装它：

```
~]# yum install spamassassin
```

有关使用 yum 工具安装软件包的更多信息，请参看 7.2.4 安装软件包。

本地用户使用 SpamAssassin 的最简单方法是在 ~/.procmailrc 文件头部添加上以下行：

```
INCLUDERC=/etc/mail/spamassassin/spamassassin-default.rc
```

/etc/mail/spamassassin/spamassassin-default.rc 文件包含一个简单的 Procmail 规则用于针对所有邮件都激活 SpamAssassin。如果一个邮件被识别为垃圾邮件，那么它的标题就会被标记成以下格式：

```
*****SPAM*****
```

也可能因为在电子邮件的正文上加了一个运行标签，而导致它被诊断为垃圾邮件。

使用类似于以下的规则，就可能将电子邮件标记为垃圾邮件：

```
:0 Hw * ^X-Spam-Status: Yes spam
```

这个规则把所有电子邮件头部都标记为了垃圾邮件，并把它们归档到了垃圾邮箱中。

由于 SpamAssassin 是一个 Perl 脚本，因此，有必要在高负载服务器上使用二进制的 SpamAssassin 守护进程（spamd）和客户端应用（spamc）。完成这样的配置，需要 root 用户登录到主机上才能完成。

执行以下命令，可以启动 spamd 守护进程：

```
~]# systemctl start spamassassin
```

执行以下命令，可以使得 SpamAssassin 守护进程随操作系统自启：

```
systemctl enable spamassassin.service
```

要了解有关服务启动/停止的更多信息，请参看 8 用 systemd 管理服务。

在~ /. Procmailrc 文件顶部附近，添加上以下一行，就可以配置 Procmail 接收 SpamAssassin 客户端的请求，而不是 Perl 脚本。对于一个系统配置，可以将它添加到/etc/procmailrc 文件中。

```
INCLUDERC=/etc/mail/spamassassin/spamassassin-spamc.rc
```

4.2.5 邮件用户代理

NeoKylin Linux Advanced Server 提供了多种电子邮件程序客户端，有基于图形的电子邮件客户端程序，如：Evolution；也有基于文本的电子邮件客户端程序，如：Mutt。

本节的其余部分侧重于介绍如何确保客户端和服务器之间的安全通信。

4.2.5.1 安全通信

NeoKylin Linux Advanced Server 中包含有一些大家比较认可的邮件用户代理，如：Evolution 和 Mutt，它们能提供基于 SSL 加密的电子邮件会话。

像任何其他服务流经网络未加密一样，重要的邮件信息，如：用户名，密码及整个信件都可能在网被用户截获和浏览。此外，由于标准的 POP 和 IMAP 协议传递的是未加密的身份认证信息，因此，攻击者可以通过网络收集到用户传递的用户名和密码，以获得用户邮件账号的访问权限。

安全邮件客户

大多数 Linux 的 MUA 都被设计成能检查远程服务器上基于 SSL 加密的电子邮件。为了基于 SSL 来检索邮件，必须在邮件客户端和服务端上启用 SSL。

常常在邮件用户代理工具的配置窗口上，按下一个按钮，或者通过邮件用户代理工具配置文件上的一个选项，就都能很容易地启用 SSL 了。安全的 IMAP 和 POP 拥有众所周知的端口号（分别为：993 和 995），MUA 就是使用它们来鉴别身份和下载信件的。

安全邮件客户通信

在邮件服务器上，对 IMAP 和 POP 用户提供 SSL 加密其实是一件简单的事情。

首先，使用以下两种方法之一来创建一个 SSL 证书。方法一：向证书颁发机构（CA）申请一个 SSL 证书；方法二：创建一个自签名的证书。

警示：

自签名证书仅仅适用于测试环境。生产环境中的任一服务器都要使用 CA 签发的 SSL 证书。

为了给 IMAP 和 POP 创建自签名证书，需要由 root 用户，切换到/etc/pki/dovecot/目录下，编辑/etc/pki/dovecot/dovecot-openssl.cnf 配置文件中的证书参数，如下所示：

```
dovecot]# rm -f certs/dovecot.pem private/dovecot.pem  
dovecot]# /usr/libexec/dovecot/mkcert.sh
```

一旦完成，需要确认在/etc/dovecot/conf.d/10-ssl.conf 有以下配置：


```
ssl_cert = </etc/pki/dovecot/certs/dovecot.pem
ssl_key = </etc/pki/dovecot/private/dovecot.pem
```

执行以下命令重启 dovecot 守护进程：

```
~]# systemctl restart dovecot
```

另外，stunnel 命令可以作为一个标准的加密封装，非安全地连接到 IMAP 和 POP 服务。

Stunnel 实用程序使用的是包含在 NeoKylin Linux Advanced Server 内的外部 OpenSSL 库，它能提供强大的加密和保护网络连接功能。虽然推荐应用从 CA 获取的 SSL 证书，但也有可能用上自签名证书。

想了解如何安装 stunnel 和创建它的基本配置，请参看《NeoKylin Linux Advanced Server 安全指导》中的“使用 stunnel”章节。在/etc/stunnel/stunnel.conf 配置文件中，添加以下行，就可以为 IMAP 和 POP3S 配置 stunnel 成为一个封装。

```
[pop3s]
accept  = 995
connect = 110

[imaps]
accept  = 993
connect = 143
```

安全指导中也解释了如何启动和停止 stunnel。一旦启动了它，就有可能基于 SSL 加密使用 IMAP 和 POP 电子邮件客户端，从而连接到电子邮件服务器。

4.3 目录服务器

4.3.1 OpenLDAP

LDAP（轻量目录访问协议）是一组开放的协议，用来访问在网络上集中存储的信息。它基于 X.500 目录共享标准，但是更少的复杂度和资源密集型，所以，LDAP 被称作 X.500 标准基础上产生的一个简化版本。

像 X.500 一样，LDAP 采用目录组织分层方式。这些目录可以存储各种

信息，如名字，地址，和电话号码，甚至可以像类似于网络信息服务(NIS)来使用，确保任何人在任何机器上通过 LDAP 允许的网络都可以访问他们的账户。

LDAP 通常用于集中管理用户和组，用户身份验证或系统配置。它也能作为一个虚拟电话目录，允许用户方便的访问其他用户的信息。此外，它可以引用用户到其他 LDAP 服务器中，从而提供一个特别的全球信息的存储库。然而，它是最常见的是应用于单个组织机构，例如大学、政府部门和私人公司。

本节涵盖了 OpenLDAP 2.4 安装和配置，一个 LDAPv2 and LDAPv3 协议的开源实现。

LDAP 介绍

使用 client-server 体系结构，LDAP 创建一个通过网络可访问的中心信息目录。当客户端尝试修改这个目录里面的信息时，服务器端会验证用户是否有修改的权限，然后如果有需求会添加或更新入口。为了确保对话是安全的，Transport Layer Security (TLS)密码协议被用来防止通过截获传输来攻击。

LDAP 服务器支持多种数据库系统，管理员可以根据不同需求，有更多的选择。因为已经有定义好的编程接口，能够和 LDAP 服务器进行通信的应用程序有很多，并且在数量和质量上都在增加。

LDAP 术语

下面列出在本章中使用 LDAP-specific 术语：

entry

LDAP 目录的单一组件。每一个 entry 通过单独的分辨名来定义(DN)

attribute

信息是直接和 entry 关联的。例如，假如一个组织表示为一个 LDAP 的 entry，和该组织关联的 attributes 包括地址，传真机号等等。类似的，个人表示为一个 entry，包括个人电话号码或邮箱地址。

一个 attribute 可以是一个单一的值，或者是一组无序的值列表。虽然某些属性是可选的,但其他是必需的。必要的 attributes 必须使用 objectClass 来定义，并且在 /etc/openldap/slapd.d/cn=config/cn=schema/目录下的 schema 文件中被找到。

该 attribute 的声明和它的值被称为 Relative Distinguished Name (RDN)，不同于 DN，RDN 对于一个 entry 来说是独一无二的。

LDIF

LDAP 数据交换格式 (LDIF)，是 LDAP entry 的纯文本表现方式，如下所示：

```
[id] dn: distinguished_name
attribute_type: attribute_value...
attribute_type: attribute_value...
...
```

id 选项是一个应用定义的数字，用来编辑 entry。每一个 entry 可以包含多个 attribute_type 和 attribute_value 组，只有它们在相应的文件中被定义了。在 entry 最后，包含一空白行。

OpenLDAP 特性

OpenLDAP 提供了很多重要的特性：

- 支持 LDAPv3——在 LDAP 版本 2 当中，该协议中许多修改设计是用来保证 LDAP 更加安全。其它改进，包括支持 Simple Authentication and Security Layer (SASL), Transport Layer Security (TLS)和 Secure Sockets Layer (SSL) 等协议
- LDAP 使用 IPC——使用 inter-process communication (IPC)，加强了安全性通过消除通过网络进行对话的需求。
- IPv6 的支持——OpenLDAP 支持 IPv6 网络协议
- LDIFv1 的支持——OpenLDAP 支持 LDIF 版本 1
- 更新后的 C 接口——当前的 C 接口增强了程序员连接和使用 LDAP 目录服务器的方法
- 加强了独立的 LDAP 服务器——包括了更新了的访问控制系统，线程池，更好的工具等等。

OpenLDAP 服务器安装

在 NeoKylin 系统上安装 LDAP 服务器步骤如下：

- 1) 安装 OpenLDAP 组件，参见 4.3.2 安装 OpenLDAP 组件。
- 2) 配置参见 4.3.3 配置 OpenLDAP 服务器 11.。
- 3) 启动 slapd 服务，参见 4.3.5 运行 OpenLDAP 服务。
- 4) 使用 ldapadd 功能添加 entries 给 LDAP 目录。
- 5) 使用 LDAPsearch 功能确保 slapd 服务已经正确获得信息。

4.3.2 安装 OpenLDAP 组件

OpenLDAP 的函数库和工具由以下包提供：

表格 4-3 OpenLDAP 包列表

Package	Description
openldap	该包包含运行 OpenLDAP 服务器和客户端应用所需要的函数库
openldap-clients	该包包含了可以访问和修改 LDAP 服务器的命令程序
openldap-servers	该包包含了运行 LDAP 服务器的服务以及配置程序，包含了单独的 LDAP Daemon,， slapd
compat-openldap	该包包含了 OpenLDAP 的兼容库函数

此外，以下包通常和 LDAP 服务器使用：

表格 4-4 常用附加 LDAP 包列表

Package	Description
nss-pam-ldapd	该包包含 nslcd，一个本地的 LDAP 名称服务，允许用户执行本地查询
mod_ldap	该包包含 mod_authnz_ldap 和 mod_LDAP 模块，其中 mod_authnz_ldap 模块是为 Apache HTTP Server 的 LDAP 验证模块。该模块能针对 LDAP 目录够验证用户的证书，并且能够强制访问控制基于用户名，完全的 DN，组关系，一个任意 attribute，或者是一个完整的过滤字符串。这个 mod_LDAP

	模块包含在同一个包中，提供一个可配置的共享内存 cache，为了避免重复的 HTTP 请求，支持 SSL/TLS。
--	---

使用 yum 安装以下包：

```
yum install package...
```

例如，安装 LDAP 服务器

```
~]# yum install openldap openldap-clients openldap-servers
```

注意你必须拥有超级用户权限，使用 root 用户进行登录后运行命令。如果想知道更多安装新软件包的信息，请参考“安装软件包”。

OpenLDAP 服务器端程序

为了执行管理任务，这些 openldap-servers 包安装了以下组件和 slapd 服务：

表格 4-5 OpenLDAP 服务端工具列表

Command	Description
slapacl	允许您检查访问属性的列表
slapadd	允许您添加 entries 从 LDIF 文件到 LDAP 目录
slapauth	允许您检查认证和权限 ID 列表
slapcat	允许您从 LDAP 目录中以 LDIF 格式保存 entries
slapdn	允许您检查 DN 列表
slapindex	允许您根据当前内容重新编排 slapd 目录。当你修改配置文件相关参数，允许该组件
slappasswd	允许您创建一个加密的用户密码，为 ldapmodify 组件，或者是用在 slapd 配置文件中
slapschema	允许您通过相应的模式检查数据库是否符合规范
slaptest	允许您检查 LDAP 服务器配置

OpenLDAP 的客户端程序

openldap-clients 安装包包含以下组件，功能包括在 LDAP 目录下添加，修改和删除 entries:

表格 4-6 OpenLDAP 客户端工具列表

命令	描述
ladpadd	允许您给 LDAP 目录添加 entries，可以通过一个文件或者是标准输入，该命令是 ldapmodify -a 的一个链接
ldapcompare	允许您拿一个给定的 attribute 和 LDAP 目录 entry 进行比较
ladpdelete	允许您删除一个 LDAP 目录 entry
ldapexop	允许您使用 LDAP 扩展操作
ldapmodify	允许您修改 LDAP 目录 entry，通过文件或标准输入
ldapmodrdn	允许您修改修改 LDAP entry 的 RDN 值
ldappasswd	允许您修改 LDAP 用户密码
ldapsearch	允许您修改查找 LDAP entry
ldapurl	允许您生成或分解 LDAP URLS
ldapwhoami	允许您在 LDAP 服务器上执行我是谁操作

除了 ldapsearch 命令外，其它命令建议使用文件的方式进行修改，而不是直接使用命令进行修改。可以通过 man 命令查看文件格式。

LDAP 客户端应用介绍

虽然有许多 LDAP 客户端可以创建和修改服务器端目录，但是 NeoKylin Linux Advanced Server 没有包含任何一种。常见的能够以只读模式访问服务器目录的应用，包括 Mozilla, Thunderbird, Evolution, 或者 Ekiga。

4.3.3 配置 OpenLDAP 服务器

默认的，OpenLDAP 配置文件保存在/etc/openldap 目录下。下面的表格包含了最重要的一些文件和目录。

表格 4-7 OpenLDAP 配置目录和文件列表

路径	描述
/etc/openldap/ldap.conf	使用 OpenLDAP 库函数的客户端应用的配置文件，包括 ldapadd, ldapsearch, Evolution 等等
/etc/openldap/slapd.d/	Slapd 的配置文件

OpenLDAP 不在使用/etc/openldap/slapd.conf 配置文件，它使用一个配置数据库在/etc/openldap/slapd.d/目录。假如你之前的安装已经有了 slapd.conf 文件，你可以使用以下命令进行转换：

```
#slaptest -f /etc/openldap/slapd.conf -F /etc/openldap/slapd.d/
```

slapd 配置包括 LDIF entries，在一个分层的目录组织结构中，可以进行相应的编辑，参考 0 OpenLDAP 服务器端程序。

修改全局配置

LDAP 服务器的全局配置文件保存在/etc/openldap/slapd.d/cn=config.ldif 文件中。常用以下指令：

olcAllows

olcAllows 命令运行您指定哪些特性是可以使用的，使用以下格式：

```
olcAllows: feature
```

可用的特性，参照表格 4-8 可用的 olcAllows 选项。默认的选项是 bind_v2。

表格 4-8 可用的 olcAllows 选项

选项	描述
Bind_v2	LDAP 可接受的 bind 请求
Bind_anon_c red	当 DN 是空的时候接受匿名的 bind
Bind_anon_d	当 DN 是非空的时候接受匿名 bind

n	
Update_anon	接受匿名升级操作
Proxy_authz_anon	接受匿名代理控制

例如：使用 olcAllows 指令

```
olcAllows: bind_v2 update_anon
```

olcConnMaxPending

olcConnMaxPending 命令允许您指定匿名会话最大请求等待数，命令

如下：

```
olcConnMaxPending: 100
```

olcConnMaxPendingAuth

olcConnMaxPendingAuth 命令您指定验证过的会话最大请求等待数，

命令如下：

```
olcConnMaxPendingAuth : number
```

默认值是 1000。

例如：使用 olcConnMaxPendingAuth 命令

```
olcConnMaxPendingAuth : 1000
```

olcDisallows

olcDisallows 命令允许您指定哪些特性不可用。命令如下：

```
olcDisallows: feature...
```


可接受的特性列表参考表格 4-8 可用的 `olcAllows` 选项。没有默认不可使用的特性。

表格 4-9 可用 `olcDisallows` 选项

选项	描述
<code>bind_anon</code>	不允许接收匿名 <code>bind</code> 请求
<code>bind_simple</code>	不允许简单的 <code>bind</code> 验证机制
<code>tls_2_anon</code>	不允许增强匿名会话当收到 <code>STARTTLS</code> 命令
<code>tls_authc</code>	当已经验证后不允许 <code>STARTTLS</code> 命令

例如：使用 `olcDisallows` 命令

```
olcDisallows: bind_anon
```

`olcIdleTimeout`

`olcIdleTimeout` 命令允许您指定在关闭一个 `idle` 连接的时候，可以等待的时间。命令如下：

```
olcIdleTimeout: number
```

该选项默认值是不使用（意思是设置为 0）

例如：使用 `olcIdleTimeout` 命令

```
olcIdleTimeout: 180
```

`olcLogFile`

`olcLogFile` 命令指定一个文件记录日志信息，命令如下：

```
olcLogFile: file_name
```

日志信息默认使用错误标准输入格式

例如：使用 `olcLogFile` 命令

```
olcLogFile: /var/log/slapd.log
```

`olcReferral`

`olcReferral` 选项允许您指定一个服务器的 URL 来处理请求，命令如下：

```
olcReferral: URL
```

默认不使用

例如：使用 `olcReferral` 命令

```
olcReferral: ldap://root.openldap.org
```

`olcWriteTimeout`

`olcWriteTimeout` 选项允许您指定在关闭一个未完成的写请求连接的时候，可以等待的时间。格式如下：

```
olcWriteTimeout
```

默认不开启（意思是值为 0）

例如：使用 `olcWriteTimeout` 命令

```
olcWriteTimeout: 180
```

修改特定数据库配置

默认的，OpenLDAP 服务器使用 Berkeley DB(BDB)作为后端数据库。该数据库配置存储在 `/etc/openldap/slapd.d/cn=config/olcDatabase= {1}bdb.ldif` 文件。

以下为配置数据库命令：

`olcReadOnly`

`olcReadOnly` 命令允许您使用只读模式使用数据库，使用如下：

```
olcReadOnly: boolean
```

接收参数: TRUE(只读模式)或 FALSE(可修改模式), 默认为 FALSE

例如: 使用 `olcReadOnly` 命令

```
olcReadOnly: TRUE
```

`olcRootDN`

`olcRootDN` 命令可以为 LDAP 目录指定超级用户。使用如下:

```
olcRootDN: distinguished_name
```

接收 DN。默认值为 `cn= Manager,dn=my-domain,dc=com`。

例如: 使用 `olcRootDN` 命令

```
olcRootDN: cn=root, dn=example, dn=com
```

`olcRootPW`

`olcRootPW` 命令允许您为用户设置密码, 使用如下:

```
olcRootPW: password
```

可以接收没有加密的文本字符串, 或者是哈希值, 在 shell 终端, 使用如下命令生成哈希值

```
~]$ slappaswd
New password:
Re-enter new password:
{SSHA}WczWsyPEnMchFf1GRTweq2q7XJcvmSxD
```

例如: 使用 `olcRootPW` 命令

```
olcRootPW: {SSHA}WczWsyPEnMchFf1GRTweq2q7XJcvmSxD
```

`olcSuffix`

`olcSuffix` 命令允许您指定提供信息的域, 使用如下:

```
olcSuffix: domain_name
```

接收 FQDN，默认是 dc=my-domain, dc=com

例如：使用 olcSuffix 命令

```
olcSuffix: dc=example, dc=com
```

架构扩展

自从 OpenLDAP2.3 以来，/etc/openldap/slapd.d/目录包含了之前存放在 /etc/openldap/schema/目录下的一些 LDAP 定义。OpenLDAP 使用这些可以扩展方案，支持额外的 attribute 类型和对象类使用缺省架构文件。关于这个方面更多的信息，可以参考 <http://www.openldap.org/doc/admin/schema.html>。

建立安全连接

OpenLDAP 客户端和服务端使用 Transport Layer Security (TLS)框架来保证安全。TLS 是提供网络通信安全的加密协议。上面提到的，在 NeoKylin Linux Advanced Server V7 中 OpenLDAP 使用 Mozilla NSS 作为 TLS 的安装实现。

使用 TLS 建立安全连接，怎么通过 Mozilla NSS 使用 TLS/SSL，链接 <http://www.openldap.org/faq/data/cache/1514.html>。因此，需要在客户端和服务端进行相关配置。至少，服务器端需要配置 CA 证书和它本身的服务器证书和私钥。客户端需要配置包含可信任的 CA 证书文件。

典型的，服务器端需要指定一个 CA 证书，客户端想要连接到一个安全的服务器端，因此需要在其配置文件里面指定可信任的 CA。

服务器配置：

该章节列出了在 OpenLDAP 服务器中使用 TLS，需要对 slapd 命令进行全局配置，在/etc/openldap/slapd.d/cn=config.ldif 配置文件中配置。

老版本的配置使用单独的配置文件，通常是 /usr/local/etc/openldap/slapd.conf，新版的使用 slapd 后端数据库保存配置，保存目录/usr/local/etc/openldap/slapd.d/。

以下命令对于确立 SSL 来说也是有效的，除了 TLS 命令外，你需要在服

务器端给 SSL 打开一个端口，一般来说是 636 端口。编辑/etc/sysconfig/slapd 文件，添加 ldaps:///字符串，指定 SLAPD_URLS 命令 URLs。

olcTLSCACertificateFile

olcTLSCACertificateFile 命令指定了 Privacy-Enhanced Mail (PEM) 编码的文件，包含可信的 CA 证书。使用如下

```
olcTLSCACertificateFile: path
```

path 为包含 CA 证书的文件，或者如果使用 Mozilla NSS 的话，可以是证书名称。

olcTLSCACertificatePath

olcTLSCACertificatePath 命令指定在不同文件中包含单独 CA 证书存放的目录。该目录必须由 OpenSSL c_rehash 进行管理，生成指向实际证书文件的链接，一般而言，常使用 olcTLSCACertificateFile 作为替代

假如 Mozilla NSS 被使用,olcTLSCACertificatePath 接受 Mozilla NSS 数据库路径（就像下例所说）。在这种情况下，c_rehash 是不需要的。

使用如下

```
olcTLSCACertificatePath: path
```

例如：使用 olcTLSCACertificatePath Mozilla NSS

通过 Mozilla NSS，olcTLSCACertificatePath 指定目录路径，该目录 包含 NSS 证书和数据库文件

```
olcTLSCACertificatePath: sql:/home/nssdb/sharednssdb
```

certutil 命令用来给 NSS 数据库文件添加 CA 证书

```
certutil -d sql:/home/nssdb/sharednssdb -A -n  
"CA_certificate" -t CT,, -a -i certificate.pem
```

以上的命令添加了一个 CA 证书，以 PEM-formatted 格式保存，名字为 *certificate.pem*。-d 选项指定数据库目录，该目录 包含 NSS 证书和数据库文件，-n 选项设置证书名称，-t CT，意思是证书是可信任的，在 TLS 客户端和服务端使用。-A 添加已存在的证书至证书数据库中，-a 允许使用 ASCII 格式 作为输入输出，-i 将 *certificate.pem* 输入传递给命令。

olcTLSCertificateFile

olcTLSCertificateFile 命令指定包含 slapd 服务器证书的文件。

```
olcTLSCertificateFile: path
```

path 为包含 slapd 服务器证书的文件，如果使用 Mozilla NSS，改为证书名称。

例如：通过 Mozilla NSS 使用 olcTLSCertificateFile

当使用 Mozilla NSS 和证书关键数据库文件和 olcTLSCACertificatePath 命令，olcTLSCACertificatePath 用来指定证书的名称。首先，列出 NSS 数据库文件中可用的证书。

```
certutil -d sql:/home/nssdb/sharednssdb -L
```

选择一个证书，将它的名称传递给 olcTLSCertificateFile

```
olcTLSCertificateFile slapd_cert
```

olcTLSCertificateKeyFile

olcTLSCertificateKeyFile 命令指定文件，该文件包含私钥，私钥和存放在 olcTLSCertificateFile 的证书是匹配的。当前不支持加密的私钥，因此该文件必须被有效的保护。

```
olcTLSCertificateKeyFile: path
```

当使用 PEM 证书是时，**path** 替换为私钥文件路径。当使用 Mozilla NSS 时，**path** 替换为一个文件的名称，该文件包含 **olcTLSCertificateFile** 命令指定的证书的密码(参考下例使用 **olcTLSCertificateKeyFile** 和 Mozilla NSS)

例如： 使用 **olcTLSCertificateKeyFile** 和 Mozilla NSS

当使用 **Mozilla NSS** 时，该命令指定一个文件的名称，该文件包含 **olcTLSCertificateFile** 指定的证书的 key 的密码。

```
olcTLSCertificateKeyFile: slapd_cert_key
```

modutil 命令能够用来转变密码保护或改变 NSS 数据库文件密码

```
modutil -dbdir sql:/home/nssdb/sharednssdb -change pw
```

客户端配置

配置文件 **/etc/openldap/ldap.conf** 在系统中是全局的，也有单独的用户在 **~/.ldaprc** 配置中进行覆盖配置。

相同的指令可以创建一个 SSL 连接。在 OpenLDAP 命令比如 **ldapsearch**，**ldaps://**字符串必须替代 **ldap://**。这些命令使用服务器端默认的 SSL 端口 636。

TLS_CACERT

TLS_CACERT 命令指定一个文件，包含客户端识别的所有的证书。该功能等同于服务器的 **olcTLSCACertificateFile** 命令。**TLS_CACERT** 应该在文件 **/etc/openldap/ldap.conf** 中 **TLS_CACERTDIR** 选项前被指定。

```
TLS_CACERT path
```

path: CA 证书文件路径

TLS_CACERTDIR

TLS_CACERTDIR 命令指定在不同文件中包含单独 CA 证书存放的目录。跟 `olcTLSCACertificatePath` 命令类似。该目录必须由 **OpenSSL c_rehash** 进行管理，接受 **Mozilla NSS** 数据库文件路径，在这种情况下，`c_rehash` 是不需要的。

```
TLS_CACERTDIR directory
```

directory：包含 CA 证书的目录路径。使用 **Mozilla NSS** 时，为证书或关键数据库文件路径。

TLS_CERT

TLS_CERT 指定文件，包含客户端证书。该命令只有在用户 `~/.ldaprc` 文件中被指定。在 **Mozilla NSS** 下，该命令指定的是证书的名字，由 TLS_CACERTDIR 命令指定。

```
TLS_CERT path
```

path: 客户端证书文件路径，或者是 NSS 数据库证书的名称

TLS_KEY

TLS_KEY 指定包含私钥的文件，私钥是匹配存放在 TLS_CERT 指定的证书。该功能和服务器 `olcTLSCertificateFile` 类似，加密的文件是不支持的，所以该文件需要被小心保护，该选项只能在用户的 `~/.ldaprc` 文件指定。

当使用 **Mozilla NSS** 时，TLS_KEY 指定一个文件，包含私钥的密码，用来保护 TLS_CERT 指定的证书。功能和 `olcTLSCertificateKeyFile` 类似，你可以使用 `modutil` 命令管理密码。

TLS_KEY 使用如下

```
TLS_KEY path
```


path: 客户端证书文件路径, 或者是 NSS 数据库中的密码文件名称。

设置备份

备份是一个拷贝更新的进程, 从一个 LDAP 服务器 (*provider*) 至一个或多个其它的服务器或客户端 (*consumer*)。一个 *provider* 备份目录更新至 *consumers*, 接收到的更新能够被 *consumer* 传播至其它的服务器端, 因此一个 *consumers* 可以被当做一个 *provider*。一个 *consumers* 可以不是一个 LDAP 服务器端, 它可以仅仅是一个个客户端。在 OpenLDAP, 有多种备份模式, 常见的有 *mirror* 和 *sync*。更多 OpenLDAP 的备份模式信息, 可以参考《*OpenLDAP Software Administrator's Guide installed with openldap-servers package*》

为了使用选择好的备份模式, 需要在 *provider* 和 *consumers* 的 `/etc/openldap/slapd.d/` 选择以下其中一种模式:

olcMirrorMode

olcMirrorMode 使用 *mirror* 备份模式

```
olcMirrorMode on
```

其中 *serverID* 必须与 *syncrepl* 一起被指定。

olcSyncrepl

olcSyncrepl 使用 *sync* 备份模式

```
olcSyncrepl on
```

该配置详细被描述, 参见 *Syncrepl* 章节在《*OpenLDAP Software Administrator's Guide*》。

加载模块和后端

您可以使用动态加载模块用来增强 *slapd* 服务。在配置 *slapd* 的时候, 可以使用 `--enable-modules` 选项来配置那些可以支持的模块。模块保存在 `.la` 结

尾的文件中。

```
module_name.la
```

后端存储和数据检索应对 LDAP 请求。后端静态的编译至 slapd 或者当模块是被支持的，它们能够被动态的加载。对于后者，以下为命名约定

```
back_backend_name.la
```

为了加载模块和后端，在 /etc/openldap/slapd.d/ 选择：

olcModuleLoad

olcModuleLoad 指定动态加载的模块

```
olcModuleLoad: module
```

module：一个文件包含模块或后端，将要被加载。

4.3.4 使用 LDAP 应用的 SELinux 策略

SELinux 是一个在 linux 内核中实现的一个强制访问控制机制。默认的，SELinux 会组织应用访问 OpenLDAP 服务器。为了允许应用通过 LDAP 验证，SELinux 的 allow_ybind 必须被设为可用的。某些应用也需要 authlogin_nsswitch_use_ldap 是被允许的。以下是激活命令：

```
~]# setsebool -P allow_ybind=1  
~]# setsebool -P authlogin_nsswitch_use_ldap=1
```

-P 选项是是这次设置在系统重启一直保持有效。SELinux 的更多信息，参考 [《NeoKylin Linux Advanced Server V7 SELinux 用户及管理员手册》](#)。

4.3.5 运行 OpenLDAP 服务

该章节描述了怎样启动、停止、重启和检查当前 LDAP 服务的状态。

启动服务

使用 root 权限，开启 slapd 服务

```
~]# systemctl start slapd.service
```

使用 root 权限，设置开机启动 slapd 服务

```
~]# systemctl enable slapd.service

ln -s '/usr/lib/systemd/system/slapd.service'
'/etc/systemd/system/multi-user.target.wants/slapd.servic
e'
```

停止服务

停止服务

```
~]# systemctl stop slapd.service
```

设置开机不启动服务

```
~]# systemctl disable slapd.service

rm
'/etc/systemd/system/multi-user.target.wants/slapd.servic
e'
```

重启服务

重启服务

```
~]# systemctl restart slapd.service
```

该命令会先停止服务，马上再重启服务。使用该命令重新加载配置。

检查运行状态

检查 slapd 服务运行状态

```
~]$ systemctl is-active slapd.service  
  
active
```

4.3.6 配置系统使用 OpenLDAP 作为验证

为了配置系统使用 OpenLDAP 作为验证，确保所有的安装包在 LDAP 服务器和客户端机器上已经安装。怎么安装请参考章节 4.3.2 安装 OpenLDAP 组件和 4.3.3 配置 OpenLDAP 服务器。在客户端上，输入命令如下：

```
~]# yum install openldap openldap-clients nss-pam-ldapd
```

迁移旧的验证信息至 LDAP 格式

迁移工具包提供了许多 shell 和 perl 脚本，可以帮助迁移旧的验证信息至 LDAP 格式。安装这些包，命令如下：

```
~]# yum install migrationtools
```

安装完后，脚本存放目录： /usr/share/migrationtools/。编辑 /usr/share/migrationtools/migrate_common.ph 文件，修改以下行内容用来映射当前域。

```
# Default DNS domain  
  
$DEFAULT_MAIL_DOMAIN = "example.com";  
  
# Default base  
  
$DEFAULT_BASE = "dc=example,dc=com";
```

或者，使用命令行指定环境变量。例如，运行 migrate_all_online.sh 脚本，使用默认的基准，设置 dc=example,dc=com

```
~]# export DEFAULT_BASE="dc=example,dc=com" \  
/usr/share/migrationtools/migrate_all_online.sh
```

决定使用哪个脚本来运行迁移用户数据库，参考表格 4-10 常用的 LDAP 迁移脚本。

表格 4-10 常用的 LDAP 迁移脚本

已存在的 服务	LDAP 是否运 行	使用的脚本
/etc flat files	yes	migrate_all_online.sh
/etc flat files	no	migrate_all_offline.sh
NetInfo	yes	migrate_all_netinfo_online. sh
NetInfo	no	migrate_all_netinfo_offline .sh
NIS (YP)	yes	migrate_all_nis_online.sh
NIS (YP)	no	migrate_all_nis_offline.sh

怎么使用这些脚本，参考 README 和 migration-tools.txt 文件，存放在 /usr/share/doc/migrationtools-version/ 目录下。

4.4 文件和打印服务器

这个章节主要介绍 Samba 的安装和配置，一个 *Server Message Block (SMB)*

和 *common Internet file system* (CIFS)协议的开源实现, vsftpd, NeoKylin Linux Advanced Server V7 的 FTP 服务器。此外, 还介绍了怎么使用打印服务器工具去配置打印机。

4.4.1 Samba

Samba 是标准 linux 开源 windows 套件项目。它实现了 SMB 和 CIFS 协议。它允许不同的系统包括 Microsoft Windows®, Linux, UNIX 等, 访问基于 windows 的文件和打印机共享。Samba 的 SMB 使用类似 windows 服务器与 windows 客户端一样。

samba 安装

```
~]# yum install samba
```

samba 介绍

Samba 是一个很重要的组件, 作为无缝集成 Linux 服务器和桌面至 Active Directory (AD)环境。它可以作为一个域控制器, 或者是常规的域成员。

Samba 能够做什么:

- 服务目录树和 Linux, Unix 和 windows 客户端的打印机
- 协助网络浏览
- Windows 域进行身份验证登录
- 提供 *Windows Internet Name Service* (WINS)名称服务解决方案
- Windows NT®-style *Primary Domain Controller* (PDC)
- *Backup Domain Controller* (BDC) for a Samba-based PDC
- 域名服务器成员
- 加入 Windows NT/2000/2003/2008 PDC/Windows Server 2012

Samba 不能做什么:

- 作为 BDC 替代 windows PDC
- 作为域名服务器控制器

Samba 以及相关服务

Samba 是由三个守护进程组成(smbd, nmbd 和 winbindd)。三个服务(smb,

nmb 和 winbind) 控制着守护进程的启动, 停止和其它相关服务功能。这些服务作为不同的 init 脚本。以下详细介绍每一个守护进程。

smbd

服务器端 smbd 守护进程给 windows 客户端提供文件共享和打印服务。另外, 它负责用户身份验证, 资源锁定, 数据共享通过 SMB 协议。默认的, 服务器监控 SMB 传输端口为 TCP 端口 139 和 445。smbd 守护进程是被 smb 服务控制。

nmbd

nmbd 守护进程对于 NetBIOS 名称服务的请求 (SMB/CIFS in Windows-based systems) 进行理解和响应。这些系统包括 Windows 95/98/ME, Windows NT, Windows 2000, Windows XP, 和 LanManager 客户端。默认服务器监控 NMB 传输端口是 UDP 端口 137。

winbindd

winbind 服务解决了用户和组信息从运行在 Windows NT, 2000, 2003, Windows Server 2008, or Windows Server 2012 服务器上的接收问题。这使得 windows 用户和组信息能够被 UNIX 平台识别。这是被 Microsoft RPC calls, Pluggable Authentication Modules (PAM)和 the Name Service Switch (NSS)所实现。这允许 Windows NT 域和 Active Directory 用户被当做 UNIX 用户进行操作。虽然和 Samba 进行了捆绑, 但是 winbind 服务是和 smb 分开进行控制的。

连接 Samba 共享

您可以使用 **Nautilus** 或命令行连接可用的 Samba 共享。

实例: 使用 Nautilus 连接 Samba 共享

- 1) 选择【位置】->【浏览网络】, 或者是输入 smb:, 在【文件】->【输入位置】

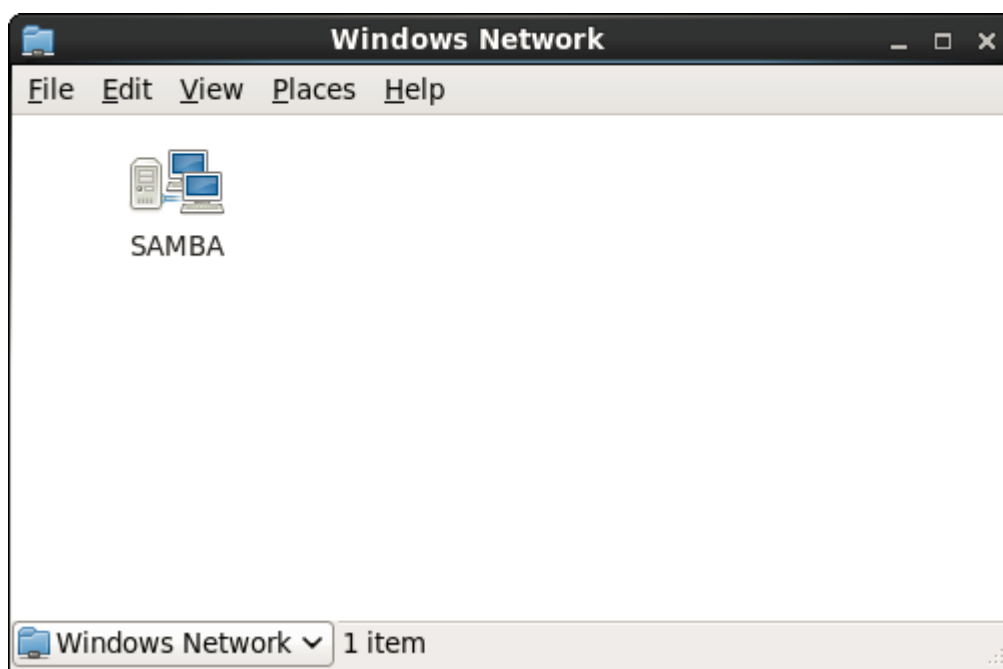


图 4-8 SMB 菜单

- 2) 双击菜单或域图标访问电脑列表。

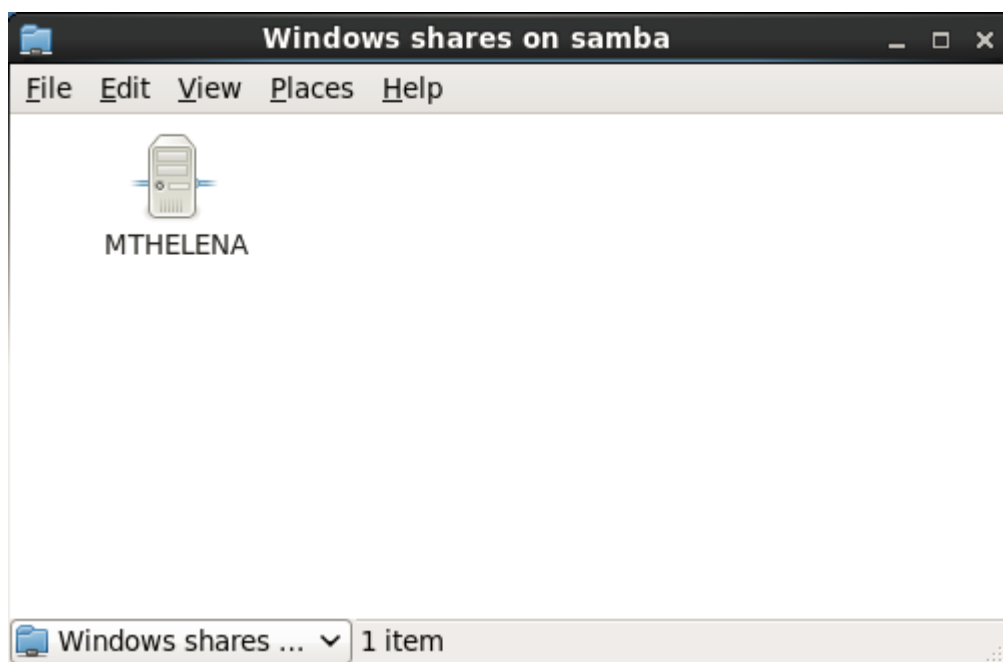


图 4-9 SMB 机器

- 3) 如图 4-9 SMB 机器，一个机器对应一个图标，双击图标，访问该机器上的 Samba 共享。如果需要，输入用户名和密码。
或者，你可以通过指定 Samba 服务器和共享名称来访问。


```
smb://servername/sharename
```

实例：使用命令行连接 Samba 共享

1) 在 shell 终端输入命令。

```
~]$ smbclient //hostname/sharename -U username
```

hostname: Samba 服务器主机名或 IP

sharename: 共享目录

username: Samba 用户

需要输入正确的用户密码

当看见 `smb:\>` 时表示你已经成功登入。输入 `help` 查看帮助

2) 如果想要退出，输入 `exit`。

挂载共享

有时候，需要对 Samba 共享进行挂载操作，需要使用 root 用户登录，命令如下：

```
mount -t cifs //servername/sharename /mnt/point/ -o  
username=username,password=password
```

该命令将 **sharename** 挂载至本地 `/mnt/point/` 目录

如果想知道更多关于 **samba** 共享挂载的信息，请查看 `mount.cifs(8)` 手册页面。

配置 Samba 服务器

默认的配置文件中 `/etc/samba/smb.conf` 允许用户访问它们的 `home` 目录作为 **samba** 共享。它可以共享所有配置好的打印机作为 **samba** 共享打印机。你可以在系统中附加一个打印机并且通过 windows 机器打印东西。

图形配置

配置 Samba 使用图形接口，可以参考 <http://www.samba.org/samba/GUI/>。

命令行配置

Samba 使用 `/etc/samba/smb.conf` 作为配置文件。修改这个配置文件后，需要重启 Samba 才能够生效

```
~]# systemctl restart smb.service
```

指定 windows 工作组和 Samba 服务器简短的描述，编辑 `/etc/samba/smb.conf` 文件

```
workgroup = WORKGROUPNAME  
  
server string = BRIEF COMMENT ABOUT SERVER
```

在 linux 系统上创建 Samba 共享目录，在 `/etc/samba/smb.conf` 文件中添加以下内容：

例如：Samba 服务器的配置

```
[sharename]  
  
comment = Insert a comment here  
  
path = /home/share/  
  
valid users = tfox carole  
  
writable = yes  
  
create mask = 0765
```

该例子允许用户 tfox 和 carole 通过 Samba 客户端对 Samba 服务器 `/home/share/` 目录进行读写操作

加密密码

加密密码是可以被使用的，因为加密的密码会更安全。创建一个使用加密密码的用户，命令如下

```
smbpasswd -a username
```

启动和关闭 Samba

启动

```
~]# systemctl start smb.service
```

关闭

```
~]# systemctl stop smb.service
```

重启

```
~]# systemctl restart smb.service
```

`condrestart` (特定条件下重启) 在当前正在运行的情况下才会启动 `smb`。这个选项对脚本是非常有用的，当守护进程没有运行的时候，将不会被启动。

```
~]# systemctl try-restart smb.service
```

重新载入 `/etc/samba/smb.conf` 配置文件，不需要进行服务的重启，命令如下

```
~]# systemctl reload smb.service
```

开机自启动

```
~]# systemctl enable smb.service
```

Samba 安全模式

Samba 有两种安全模式，*share-level* 和 *user-level*。*share-level* 已经被弃

用了，User-level 可以有三种不同的实现方式，这些方式被称作安全模式。

User-Level 安全

User-level security 是 Samba 默认推荐的配置。即使 `security = user` 没有在 `/etc/samba/smb.conf` 配置文件中列出，它照样会被使用。当服务器接收了客户端的用户名和密码，客户端在挂载共享的时候就能够不需要指定密码。Samba 也能够接收到基于用户名和密码的请求。客户端通过使用一个单独的 UID 维持已验证的状态。

`/etc/samba/smb.conf` 文件中，`security = user` 设置 user-level security

```
[GLOBAL]
...
security = user
...
```

Samba Guest 共享

上面已经提到过，share-level security 模式已经被放弃。怎么配置 Samba guest 共享，不使用 `security = share` 选项。参考以下步骤：

实例：配置 Samba Guest 共享

- 1) 创建用户映射文件，例如 `/etc/samba/smbusers`，添加以下行：

```
nobody = guest
```

- 2) 修改 `/etc/samba/smb.conf`，不要使用 valid users

```
[GLOBAL]
...
security = user
map to guest = Bad User
username map = /etc/samba/smbusers
```

```
...
```

username map : 1 步骤指定

3) 修改/etc/samba/smb.conf，不要使用 valid users

```
[SHARE]

...

guest ok = yes

...
```

下面将会介绍其它 user-level security 实现方式。

Domain Security Mode (User-Level Security)

在这种方式下，Samba 服务器有一个机器账号（domain security 信任的账号），所有验证的请求都需要通过域控制器。Samba 服务器要作为域成员，需要配置/etc/samba/smb.conf。

```
[GLOBAL]

...

security = domain

workgroup = MARKETING

...
```

Active Directory Security Mode (User-Level Security)

假如你有一个 Active Directory 环境，加入域作为一个本地成员是可能的。即使安全策略限制使用 NT-compatible 验证协议，Samba 服务器可以通过使用 Kerberos 加入 ADS。Samba 在 Active Directory member 模式下可以接受 Kerberos tickets。

修改/etc/samba/smb.conf

```
[GLOBAL]

...
```

```
security = ADS  
  
realm = EXAMPLE.COM  
  
password server = kerberos.example.com  
  
...
```

Share-Level 安全

在该模式下，服务器从客户端那仅接收一个没有明确用户名的密码。服务器期望一个密码可以给所有的共享，不依赖用户名。许多报告指出，Microsoft Windows 客户端兼容 **share-level security** 服务器。该模式已经被 Samba 抛弃。配置项 `security = share` 必须被升级成 **user-level security**。如果想使用，请参考 User-Level 安全中的例子：配置 Samba Guest 共享。

Samba 网络浏览

网络浏览使得 Windows 和 Samba 服务器出现在 Windows Network Neighborhood 中，在 Network Neighborhood 里面，服务器有自己的图标，打开图标，服务器可用的共享和打印机可以被看到。

网络浏览需要 NetBIOS 通过 TCP/IP。NetBIOS-based 网络使用 UDP 发送消息完成浏览列表管理。没有 NetBIOS 和 WINS 作为 TCP/IP 主机名称解决方案基本的方法，其它的例如静态文件（/etc/hosts）或 DNS 是必须要使用的。

一个域的主浏览服务器从所有子网中的本地主浏览服务器收集核对所有的浏览列表，因此在组和子网间可以相互浏览。

域浏览

默认的，一个 Windows 服务器 PDC 作为一个域，也是该域的主浏览服务器。Samba 服务器不应该被作为一个主浏览服务器。

在许多子网中，不包含 Windows 服务器 PDC，因此，Samba 服务器可以作为一个本地浏览服务器。配置 /etc/samba/smb.conf 文件作为一个本地主浏览服务器（或不作为），配置过程和组配置类似（参见 0 配置 Samba 服务器。）

WINS (Windows Internet Name Server)

Samba 和 Windows NT 服务器可以被作为一个 WINS 服务器。当 WINS 服务器和 NetBIOS 一起使用时，UDP 传播能够被转发在网络中允许使用名称转换。没有 WINS 服务器时，UDP 传播只能在当前子网传播不能够被转发至其它子网，组和域。假如需要 WINS 备份功能，不要使用 Samba 作为你的 WINS 服务器，因为 Samba 不支持 WINS 备份。

在一个 NT/2000/2003/2008 服务器和 Samba 混合环境中，推荐使用 Microsoft WINS。在一个只有 Samba 环境中，推荐使用 Samba 作为 WINS。

下面是一个使用 Samba 作为 WINS 服务器的例子。

例如：配置 WINS 服务器

```
[global]
wins support = yes
```

Samba Distribution Programs

net

```
net <protocol> <function> <misc_options>
<target_options>
```

net 命令的使用和在 Windows 和 MS-DOS 系统中类似。第一个参数指定命令使用的协议。protocol 选项可以是 ads, rap 或 rpc。Active Directory 使用 ads，Win9x/NT3 使用 rap，Windows NT4/2000/2003/2008 使用 rpc。假如 protocol 选项没指定，net 会自动尝试确定它。

下面例子显示了主机名为 wakko 的可用的共享：

```
~]$ net -l share -S wakko
```

Password:

Enumerating shared resources (exports) on remote server:

Share name	Type	Description
-----	----	-----
data	Disk	Wakko data share
tmp	Disk	Wakko tmp share
IPC\$	IPC	IPC Service (Samba Server)
ADMIN\$	IPC	IPC Service (Samba Server)

下面例子显示了 wakko 主机可用的 Samba 用户列表

```
~]$ net -l user -S wakko
```

root password:

User name	Comment

andriusb	Documentation
joe	Marketing
lisa	Sales

nmblookup

```
nmblookup <options> <netbios_name>
```

nmblookup 可以解决 NetBIOS 名称转换为 IP 地址。该项目在子网中会广播查询直到有目标主机回应。

下面的例子是显示 NetBIOS 名称 trek 的 IP 地址:

```
~]$ nmblookup trek
querying trek on 10.1.59.255
```


10.1.56.45 trek<00>

pdedit

pdedit <options>

pdedit 项目管理 SAM 数据库存储的账户。所有的后端包括支持 smbpasswd, LDAP 和 tdb 数据库

下面的例子是添加，删除和列出用户

```
~]$ pdedit -a kristin
new password:
retype new password:
Unix username:      kristin
NT username:
Account Flags:      [U          ]
User SID:
S-1-5-21-1210235352-3804200048-1474496110-2012
Primary Group SID:
S-1-5-21-1210235352-3804200048-1474496110-2077
Full Name: Home Directory:      \\wakko\kristin
HomeDir Drive:
Logon Script:
Profile Path:          \\wakko\kristin\profile
Domain:                WAKKO
Account desc:
Workstations: Munged
dial:
```

Logon time: 0
Logoff time: Mon, 18 Jan 2038 22:14:07 GMT
Kickoff time: Mon, 18 Jan 2038 22:14:07 GMT
Password last set: Thu, 29 Jan 2004 08:29:28
GMT Password can change: Thu, 29 Jan 2004 08:29:28 GMT
Password must change: Mon, 18 Jan 2038 22:14:07 GMT

~]\$ pdbedit -v -L kristin

Unix username: kristin
NT username:
Account Flags: [U]
User SID:

S-1-5-21-1210235352-3804200048-1474496110-2012

Primary Group SID:

S-1-5-21-1210235352-3804200048-1474496110-2077

Full Name:
Home Directory: \\wakko\kristin
HomeDir Drive:
Logon Script:
Profile Path: \\wakko\kristin\profile
Domain: WAKKO
Account desc:

Workstations: Munged

dial:

Logon time: 0
Logoff time: Mon, 18 Jan 2038 22:14:07 GMT
Kickoff time: Mon, 18 Jan 2038 22:14:07 GMT
Password last set: Thu, 29 Jan 2004 08:29:28 GMT

```

Password can change: Thu, 29 Jan 2004 08:29:28 GMT
Password must change: Mon, 18 Jan 2038 22:14:07 GMT
~]$ pdbedit -L
andriusb:505:
joe:503:
lisa:504:
kristin:506:
~]$ pdbedit -x joe
~]$ pdbedit -L
andriusb:505: lisa:504: kristin:506:
    
```

rpcclient

```
rpcclient <server> <options>
```

rpcclient 项目问题管理命令使用 Microsoft RPCs，这个是给知道 Microsoft RPCs 复杂性用户使用的。

smbcacls

```
smbcacls <//server/share> <filename> <options>
```

smbcacls 项目修改 Windows ACLs 文件和 Samba 共享的目录或者是 Windows 服务器。

smbclient

```
smbclient <//server/share> <password> <options>
```

smbclient 是 UNIX 系统通用的客户端，功能和 ftp 类似。

smbcontrol

```
smbcontrol -i <options>

smbcontrol <options> <destination> <messagetype>
<parameters>
```

smbcontrol 项目发送控制消息来运行 smbd, nmbd 或 winbindd 守护进程。smbcontrol -i 交互式运行，直到出现空行或者 'q' 被输入

smbpasswd

```
smbpasswd <options> <username> <password>
```

smbpasswd 项目管理加密密码。该项目能够被超级用户修改所有的用户密码还有普通用户修改自己的 Samba 密码。

smbspool

```
smbspool <job> <user> <title> <copies> <options>
<filename>
```

smbspool 项目是 Samba 一个 CUPS 兼容的打印接口。虽然被设置为 CUPS 打印机，smbspool 可以和非 CUPS 打印机一起使用。

smbstatus

```
smbstatus <options>
```

smbstatus 项目显示当前 Samba 连接状态

smbtar

```
smbtar <options>
```

smbtar 程序执行基于 Windows 共享文件和目录的备份和恢复。虽然和 tar 命令相似，两者不兼容。

testparm

```
testparm <options> <filename> <hostname IP_address>
```

testparm 程序检查 /etc/samba/smb.conf 文件的语法，假如你的 smb.conf 存储在默认位置 (/etc/samba/smb.conf)，则不需要指定。指定主机名和 IP 地址给 testparm 程序检查 hosts.allow 和 host.deny 文件是否配置正确。testparm 程序可以显示 smb.conf 文件和服务器规则在测试后。在调试的时候可以给丰富经验的管理员提供有用的信息。例如

```
~]$ testparm
Load smb config files from /etc/samba/smb.conf
Processing section "[homes]"
Processing section "[printers]"
Processing section "[tmp]"
Processing section "[html]"
Loaded services file OK.
Server role: ROLE_STANDALONE
Press enter to see a dump of your service definitions
<enter>
# Global parameters
[global]
    workgroup = MYGROUP
    server string = Samba Server
    security = SHARE
    log file = /var/log/samba/%m.log
    max log size = 50
```

```
socket options = TCP_NODELAY SO_RCVBUF=8192
SO_SNDBUF=8192

dns proxy = no

[homes]

comment = Home Directories

read only = no

browseable = no

[printers]

comment = All Printers

path = /var/spool/samba

printable = yes

browseable = no

[tmp]

comment = Wakko tmp

path = /tmp

guest only = yes

[html]

comment = Wakko www

path = /var/www/html

force user = andriusb

force group = users

read only = no

guest only = yes
```

wbinfo

```
wbinfo <options>
```

wbinfo 程序显示 winbindd 守护进程信息。winbindd 进程必须是运行的。

其他资源

安装文档

`/usr/share/doc/samba-<version-number>/`

可以查看 man 手册

- smb.conf(5)
- samba(7)
- smbd(8)
- nmbd(8)
- winbindd(8)

有用的网站

- <http://www.samba.org/>
- https://wiki.samba.org/index.php/User_Documentation
- <http://samba.org/samba/archives.html>

4.4.2 FTP

该章节将会介绍 FTP 协议以及 vsftpd（红帽 Linux 系统 FTP 服务器）

FTP 使用客户端-服务端架构模式来传输文件，使用 TCP 网络协议。因为 FTP 是一个较早的协议，他不支持加密用户和密码进行验证。基于这个原因，FTP 是被认为不安全的传输协议，除非特殊情况下不建议使用。因为 FTP 在网络上是非常流行的，它经常被用来分享文件。因此，作为一个管理员，需要知道它的独特特性。

这个章节描述了怎么配置 vsftpd 来建立安全连接通过 TLS 和怎么通过 SELinux 来加强 FTP 的安全性。FTP 的一个更好的取代品是 sftp，是由 OpenSSH 组件工具提供。要想获取更多 OpenSSH 配置以及 SSH 协议，请参考 3.2OpenSSH。

不同于其它协议，FTP 需要多个端口才能正常工作。当 FTP 客户端建立一

个到 FTP 服务器端的连接，它会打开端口 21 在服务器端——控制端口。这个端口被用来发送命令至服务器。所有从服务器端到客户端的数据请求是由专门的数据端口传输。数据连接端口，以及数据连接的初始化依赖于客户端请求数据方式：active 或 passive 模式。

active mode

Active mode 是 FTP 协议传输数据协议使用到的最原始的模式。当一个 active-mode 数据传输被 FTP 客户端初始化，服务器端会打开一个 20 端口给 IP 地址，和一个随机无特权的端口（大于 1024）。这样的安排意味着客户端机器必须被允许接受任何超过 1024 的端口连接。随着不安全网络的增长，使用防火墙来保护客户端机器现在是普遍的。因为这些客户端防火墙常常否定从主动模式传入的连接，所以 passive mode 被设计出来。

passive mode

像 active mode 一样，Passive mode 是被 FTP 客户端初始化的。当从服务器请求数据,FTP 客户端表明它想在 Passive mode 下访问数据，服务器在服务器上会提供一个 IP 地址和一个随机、无特权的端口(大于 1024)。然后客户端连接到该端口在服务器上下载请求的信息。

虽然 Passive mode 确实解决为客户端防火墙干扰数据连接问题,它可以使管理服务器端防火墙。你可以在一个服务器上通过限制的范围无特权的端口在 FTP 服务器上减少开放端口的数量。这也简化了服务器配置防火墙规则的过程。

vsftpd 服务器

vsftpd 设计为快、稳定和安全。vsftpd 是 NeoKylin Linux Advanced Server V7 中的 FTP 服务器。是因为其能够处理大量的连接数和安全性。

vsftpd 所使用的安全模型有三个主要方面：

- 强大的特权和非特权分离过程——独立的进程处理不同的任务,每一个进程运行的任务只需要最小权限
- 需要提升权限的任务可以使用最小权限进行处理——利用 libcap 库兼容性的特性，需要 root 特权权限的任务可以用很小特权的进程执行
- 大多数进程运行在 chroot 下——只要可能，进程可以改变程序执行时参

考的根目录位置为当前共享的目录。这个目录被认为是 **chroot** 目录。例如，假如 **/va/ftp/** 目录是共享目录，**vsftpd** 指定 **/va/ftp/** 为新的 **root** 目录，作为 **/**。这会减少黑客的攻击。

使用这些安全措施会对 **vsftpd** 如何处理请求产生影响，如下：

- 父进程运行时仅需最少的特权——父进程能够动态的计算最小特权等级将风险最小化。子进程处理直接与客户端交互和尽可能使用无权限运行。所有的需要特权的操作被一个小的父进程处理——就像 **Apache HTTP Server** 一样，**vsftpd** 分发无特权的子进程来处理传入的连接。这允许特权，父进程尽可能小处理任务。
- 所有从非特权子进程来的请求将会被父进程怀疑——和子进程进行对话是通过 **socket**，任何来自子进程的信息将会被检查。
- 大部分和 **FTP** 客户端交互式操作是在 **chroot** 环境下——因为子进程是非特权的，它们只有在共享目录下有登入权限，只允许攻击者能够访问共享目录。

启动和停止 **vsftpd**

以 **root** 用户登录

启动

```
~]# systemctl start vsftpd.service
```

停止

```
~]# systemctl stop vsftpd.service
```

重启

```
~]# systemctl restart vsftpd.service
```

有条件重启，当它已经在运行的时候，才能够执行

```
~]# systemctl try-restart vsftpd.service
```

设置开机自启动

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

开启 vsftpd 多路拷贝

有时候，一台计算机会被用来作为多路 FTP 域。这种技术叫做多归属。vsftpd 的多归属的用法是运行多个进程的拷贝，每一个进程拥有自己独立的配置文件。

首先，给所有的网络设备分配好 IP。

再次，FTP 的域的 DNS 服务器必须对应正确的机器。

当 vsftpd 响应不同 IP 的请求时，进程的多路拷贝必须是运行的。为了分发 vsftpd 进程的多路实例，一个特殊的服务 `systemd service unit` ([vsftpd@.service](#))是被 vsftpd 包提供的。

为了使用该服务，一个单独的 vsftpd 配置文件为各个 FTP 服务器实例是需要的，保存在 `/etc/vsftpd/` 目录。这些配置文件必须拥有独立的名称（例如 `/etc/vsftpd/vsftpd-site-2.conf`），并且拥有 root 的读写权限。

每一个配置文件，要有如下参数作为监听 IPv 网络

```
listen_address=N.N.N.N
```

`N.N.N.N` 为 FTP 站点的 IP 地址。假如使用的是 IPv6，则使用 `listen_address6`

假如配置文件已经存放在 `/etc/vsftpd/` 目录，单独的 vsftpd 进程，可以由以下命令启动

```
~]# systemctl start  
vsftpd@configuration-file-name.service
```

在以上的命令中，修改 *configuration-file-name* 为需要的配置文件名称。

例如 vsftpd-site-2，注意，.conf 后缀是不需要添加进来的。

如果想一次性启动多个 vsftpd 进程

```
~]# systemctl start vsftpd.target  
~]# systemctl enable vsftpd.target  
ln -s '/usr/lib/systemd/system/vsftpd.target'  
'/etc/systemd/system/multi-user.target.wants/vsftpd.target'
```

使用 TLS 加密 vsftpd 连接

为了对抗 FTP 固有的不安全性（使用明文传输）。vsftpd 进程可以使用 TLS 来对连接和传输进行加密。FTP 客户端必须支持 TLS。

在配置文件中 vsftpd.conf 设置 ssl_enable 为 YES 打开 TLS 的支持。

例如：配置 vsftpd 使用 TLS

vsftpd.conf 配置文件

```
ssl_enable=YES  
ssl_tlsv1=YES  
ssl_sslv2=NO  
ssl_sslv3=NO
```

重启 vsftpd service 生效

```
~]# systemctl restart vsftpd.service
```

可以查看 `vsftpd.conf(5)` 的 `man` 手册了解更多相关信息

vsftpd 的 SELinux 策略

SELinux 策略管理 `vsftpd` 守护进程(以及其他 `ftpd` 流程), 定义了一个强制访问控制, 为了允许 `FTP` 守护进程访问特定文件或目录, 需要分配给他们适当的标签。

例如, 为了能够匿名共享文件, `public_content_t` 标签必须分配给共享的文件和目录。你可以使用 `chcon` 命令

```
~]# chcon -R -t public_content_t /path/to/directory
```

/path/to/directory 是你想分配标签的目录路径, 如果你想建立一个上传文件的目录, 你必须分配一个特使的标签 `public_content_rw_t`。除此之外, `allow_ftpd_anon_write` 选项必须设置为 1, 使用 `setsebool` 命令设置

```
~]# setsebool -P allow_ftpd_anon_write=1
```

假如你想让本地用户通过 `FTP` 访问 `home` 目录, 在 `NeoKylin Linux Advanced Server V7` 中这个是缺省的设置, `ftp_home_dir` 选项必须设置为 1。假如 `vsftpd` 允许以独立模式运行, 在 `NeoKylin Linux Advanced Server V7` 中这个是缺省的设置, `ftpd_is_daemon` 必须设置为 1。

4.4.3 打印设置

打印设置工具用来打印机的配置, 维护打印机配置文件, 打印排队目录以及打印过滤和打印机管理类。

该工具是基于通用 `Unix` 印刷系统(`CUPS`)。如果你升级的系统是先前的 `NeoKylin Linux Advanced Server` 版本, 使用的是 `CUPS`, 在升级过程将会保存打印机信息。

启动打印机配置工具

从命令行启动打印机配置工具，输入 `system-config-printer`，打印机配置工具启动。或者是，当你使用 GNOME 桌面，按【super】键进入 Activities，输入 Print Settings 并按回车键。打印机配置工具启动。

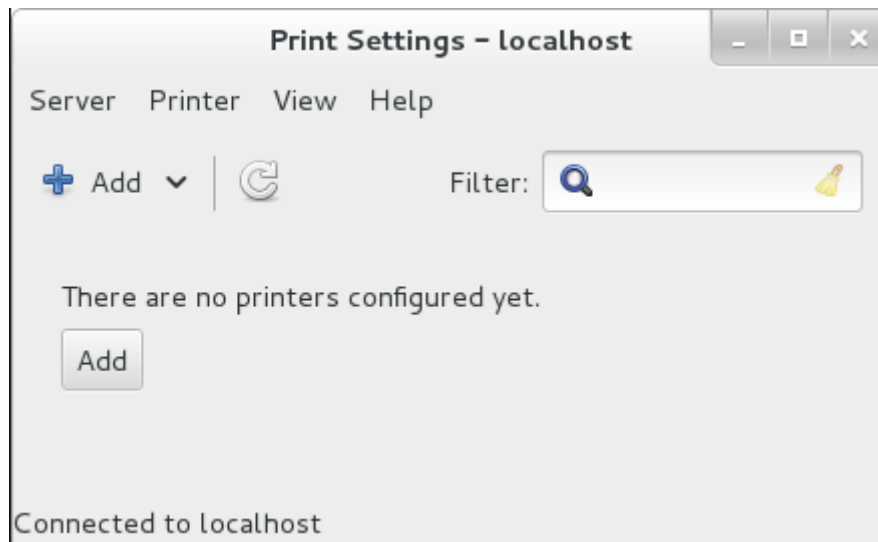


图 4-10 打印设置

启动打印机设置

打印机安装进程依赖于打印机队列类型。

假如你是设置一个通过 USB 连接的本地打印机，打印机会自动发现和添加。系统将提示您确认要安装的软件包,并提供一个管理员或 root 用户密码。本地打印机连接与其他端口类型和网络打印机需要手动设置。

以下是设置一个手动打印机的步骤：

- 1) 启动打印机配置工具（参见 0 启动打印机配置工具）。
- 2) 【服务器】->【新的】->【打印机】。
- 3) 在验证对话框中，输入管理员或 root 用户密码。假如这是你第一次配置一个远程的打印机，你将会被提示授权防火墙操作。
- 4) 选择打印机连接类型,在右边的区域提供细节。

添加一个本地打印机

添加本地打印机通过串行端口连接，步骤如下：

- 1) 打开添加打印机对话框（参考启动打印机设置）。
- 2) 假如设备没有自动出现，在左边的列表，选择打印机连接的端口（例如 **Serial Port #1** 或者 **LPT #1**）
- 3) 在右边,进入连接属性。

for other

URI（例如 file:/dev/lp0）

for **Serial Port**

Baud Rate

Parity

Data Bits

Flow Control

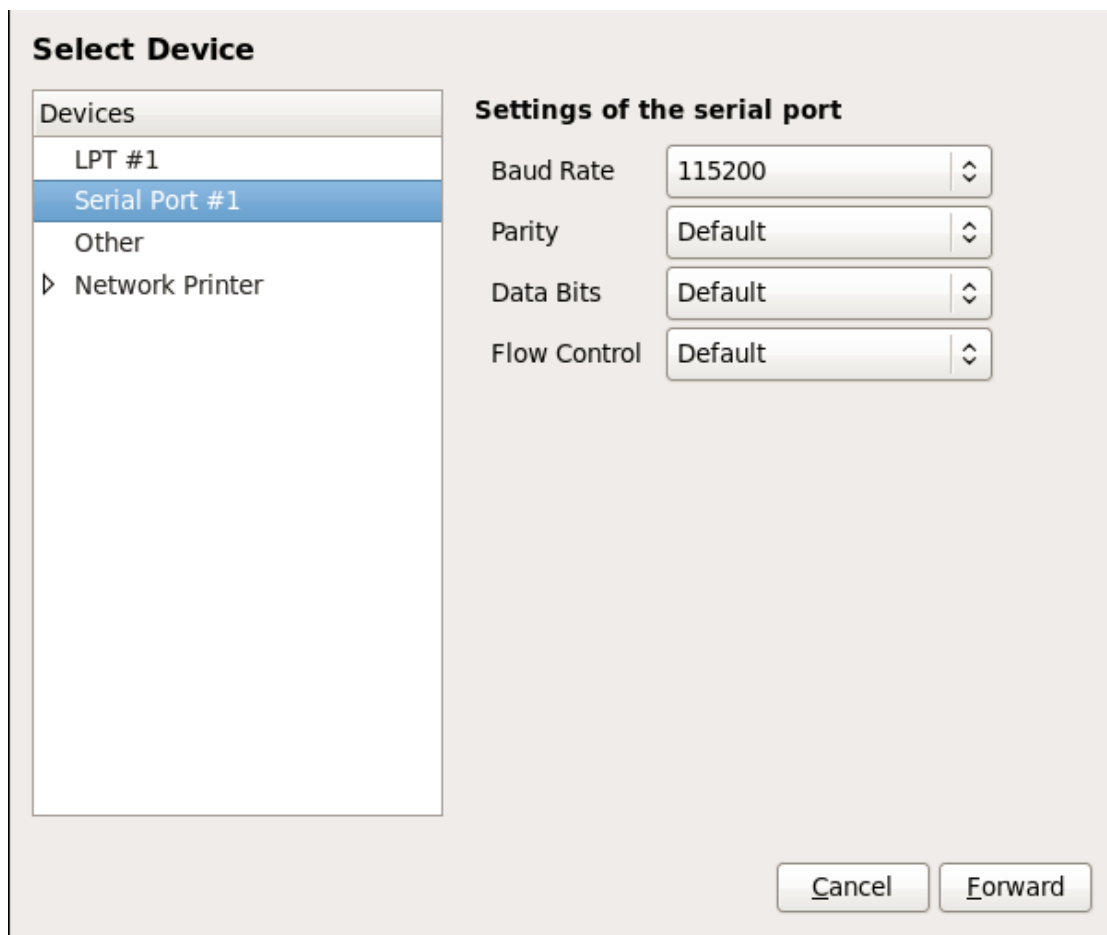


图 4-11 添加一个本地打印机

- 4) 点击【下一步】

5) 选择打印模式，参见 0 选择打印机模式并完成。

添加 AppSocket/HP JetDirect 打印机

步骤如下：

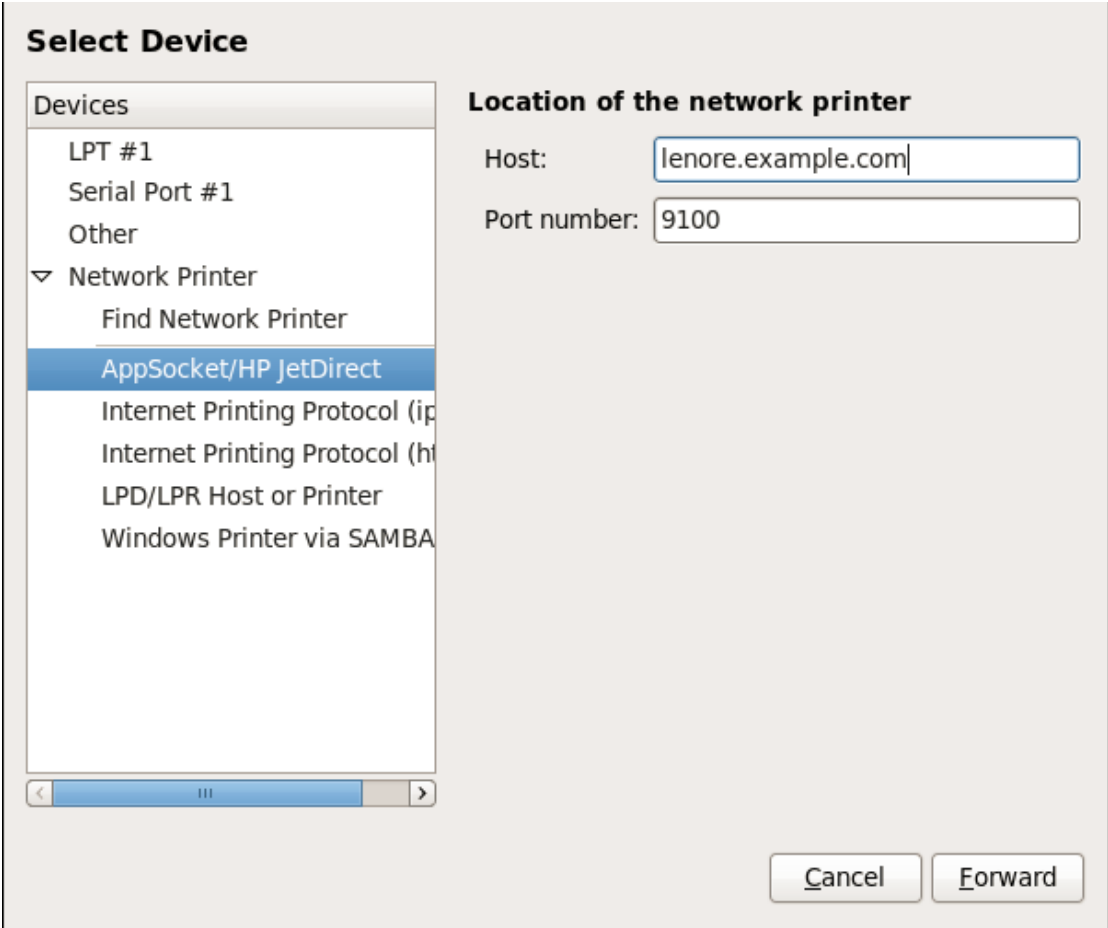
- 1) 打开添加打印机对话框（参见 0 启动打印机配置工具）。
- 2) 在左边的列表，选择 **Network Printer** → **AppSocket/HP JetDirect**。
- 3) 在右边，添加连接设置。

Hostname

打印机主机名或 IP 地址

Port Number

监听打印任务的打印机端口（默认 9100）



Select Device

Devices

- LPT #1
- Serial Port #1
- Other
- ▼ Network Printer
 - Find Network Printer
 - AppSocket/HP JetDirect**
 - Internet Printing Protocol (ip)
 - Internet Printing Protocol (hp)
 - LPD/LPR Host or Printer
 - Windows Printer via SAMBA

Location of the network printer

Host:

Port number:

- 4) 点击下一步。
- 5) 选择打印模式，参见 0 选择打印机模式并完成。

添加 IPP 打印机

IPP 打印机是通过 TCP/IP 网络连接到不同系统的打印机。在该系统上，

或允许 CUPs 或者是配置使用 IPP。

如果配置了防火墙，需要打开 TCP631 端口和 UDP631 端口

添加步骤如下：

- 1) 打开添加打印机对话框（参见 0 启动打印机设置）。
- 2) 在左边的设备列表里面，选择 **Network Printer** 和 **Internet Printing Protocol (ipp)** 或者 **Internet Printing Protocol (https)**。
- 3) 在右边，输入设置信息。

Host

IPP 打印机的主机名

Queue

给新的队列输入队列的名称（假如没有任何输入，会根据设备生产一个名称）。

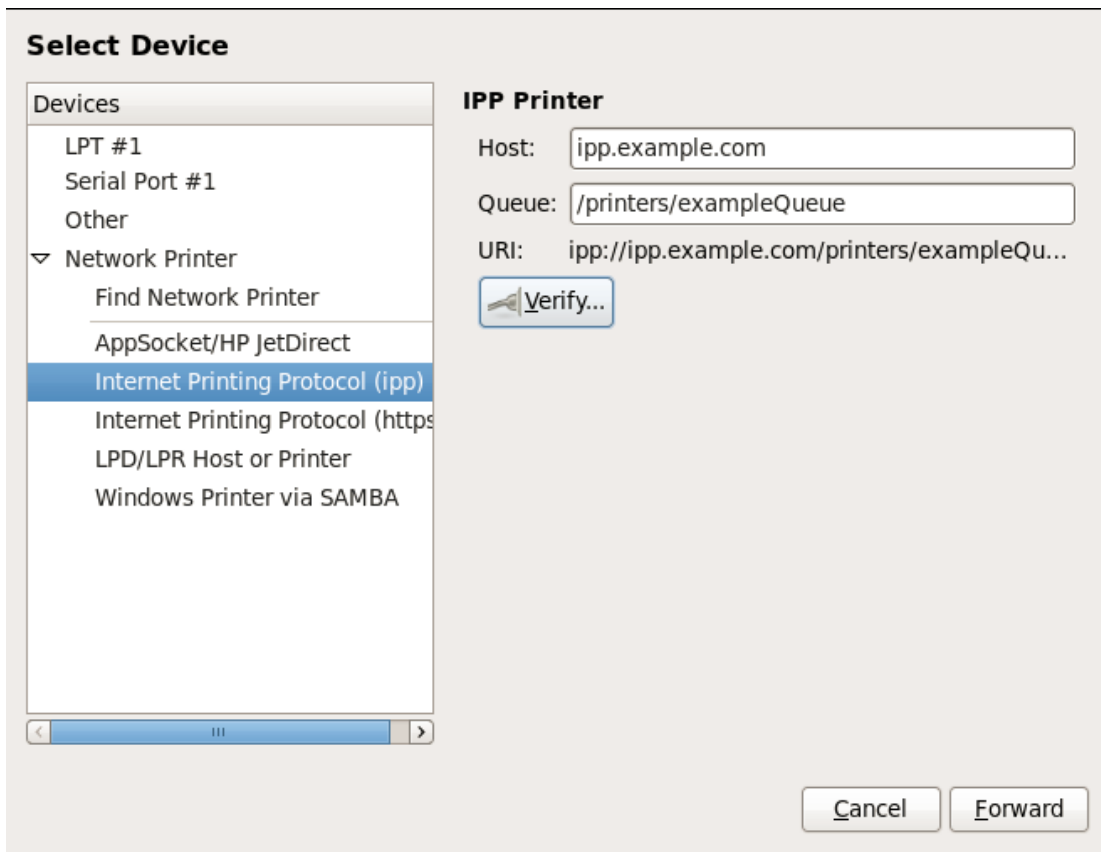


图 4-12 增加一个 IPP 打印机

- 4) 点击下一步继续。

5) 选择打印模式，参见 0 选择打印机模式并完成。

添加一个 LPD/LPR 主机或打印机

步骤如下：

- 1) 打开添加打印机对话框（参见 0 选择打印机模式并完成）。
- 2) 在左边的设备列表，选择 **Network Printer** → **LPD/LPR Host or Printer**。
- 3) 在右边，输入设置。

Host

LPD/LPR 打印机或主机的主机名

也可以，点击 **Probe** 查找 LDP 主机

Queue

给新的队列输入队列的名称（假如没有任何输入，会根据设备生产一个名称）

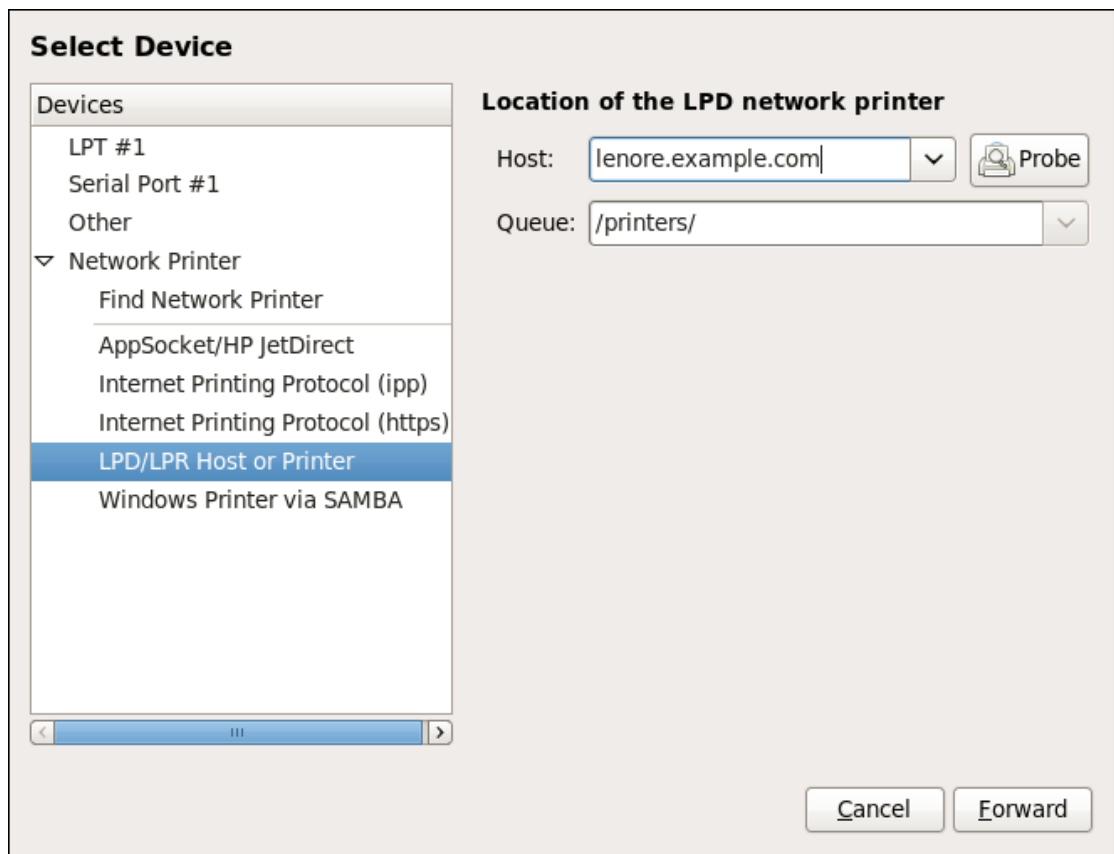


图 4-13 增加一个 LPD/LPR 打印机

- 4) 点击下一步继续。
- 5) 选择打印模式，参见 0 选择打印机模式并完成。

添加 Samba (SMB) 打印机

步骤如下：

- 1) 打开添加打印机对话框（参见 0 启动打印机设置）。
- 2) 在左边的设备列表，选择 **Network Printer** → **Windows Printer via SAMBA**
- 3) 输入 SMB 地址，smb://。使用格式主机名/共享打印机。例子的电脑名为 dellbox，共享打印机是 r2。

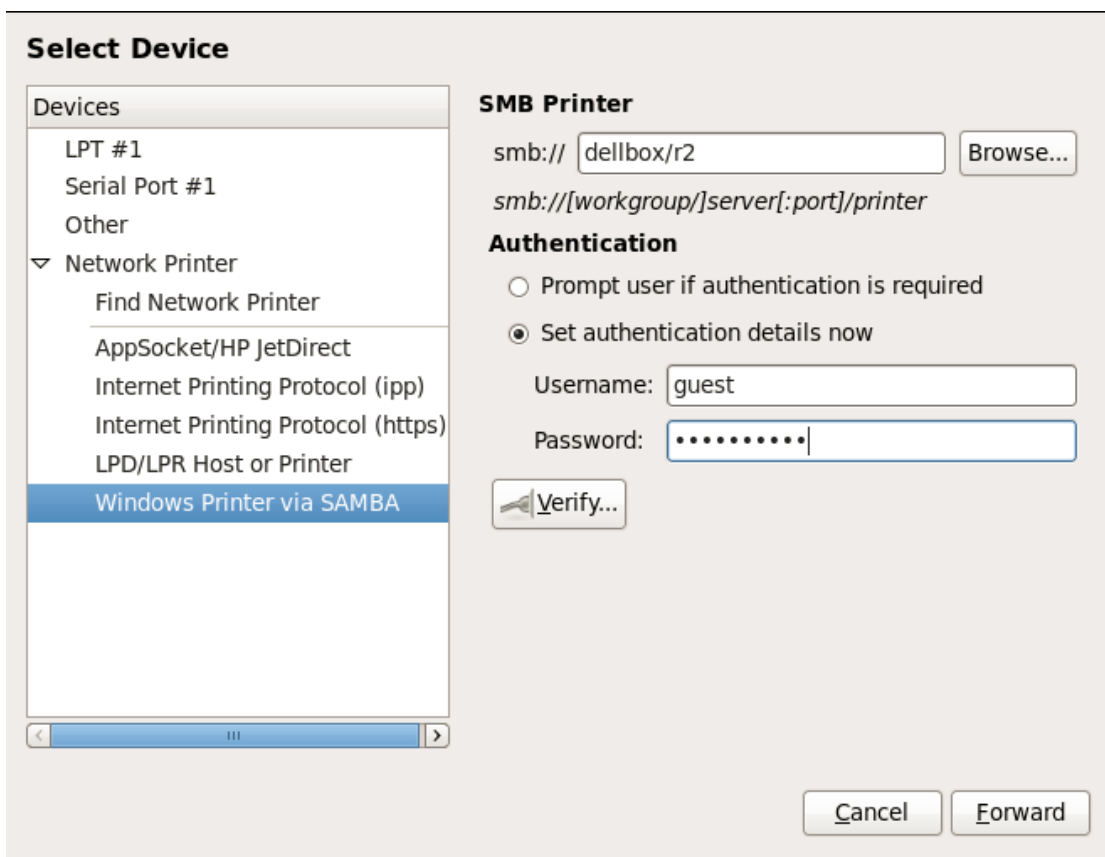


图 4-14 选择一个 SMB 打印机

- 4) 点击浏览查看可用的用户组/域。如果想显示单独的主机列表，输入主机名，点击浏览。
- 5) 选择的选项。
 - A. 提示用户是否需要身份验证：用户名和密码来自打印文档的用户
 - B. 设置身份验证细节:提供身份验证信息现在这不是必需的。在用户名字段中,输入用户名访问打印机。这对 SMB 系统用户必须存在,用户必须有权限访问该打印机。默认 Windows 服务器用户名通常是 guest，或者 Samba

服务器是 **nobody**。

- 6) 如果需要，输入用户名密码。
- 7) 单击验证测试连接。成功验证之后,出现一个对话框确认打印机共享的可访问性。
- 8) 点击【下一步】继续。
- 9) 选择打印模式，参见 0 选择打印机模式并完成。

选择打印机模式并完成

一旦你已经选择正确的打印机连接类型,系统尝试获取一个驱动。如果过程失败,您可以手动定位或寻找驱动资源。

步骤如下：

- 1) 在显示的窗口中自动驱动检测失败后,选择下列选项之一：
 - A. 从数据库选择打印机——系统会选择一个驱动基于您所选择的打印机。
如果未列出您的打印机模型，选择通用。
 - B. 提供 PPD 文件——系统使用 *PostScript Printer Description* (PPD) 文件来进行安装。一个 PPD 文件通常是由打印机的供应商提供，假如 PPD 文件是可用的，您可以选择此选项和使用浏览器栏以下选项描述选择 PPD 文件。
 - C. 搜索一个打印机驱动程序下载——在 **Make and model** 区域，输入打印机的生产型号和模型，用来在 OpenPrinting.org 搜索可用的包

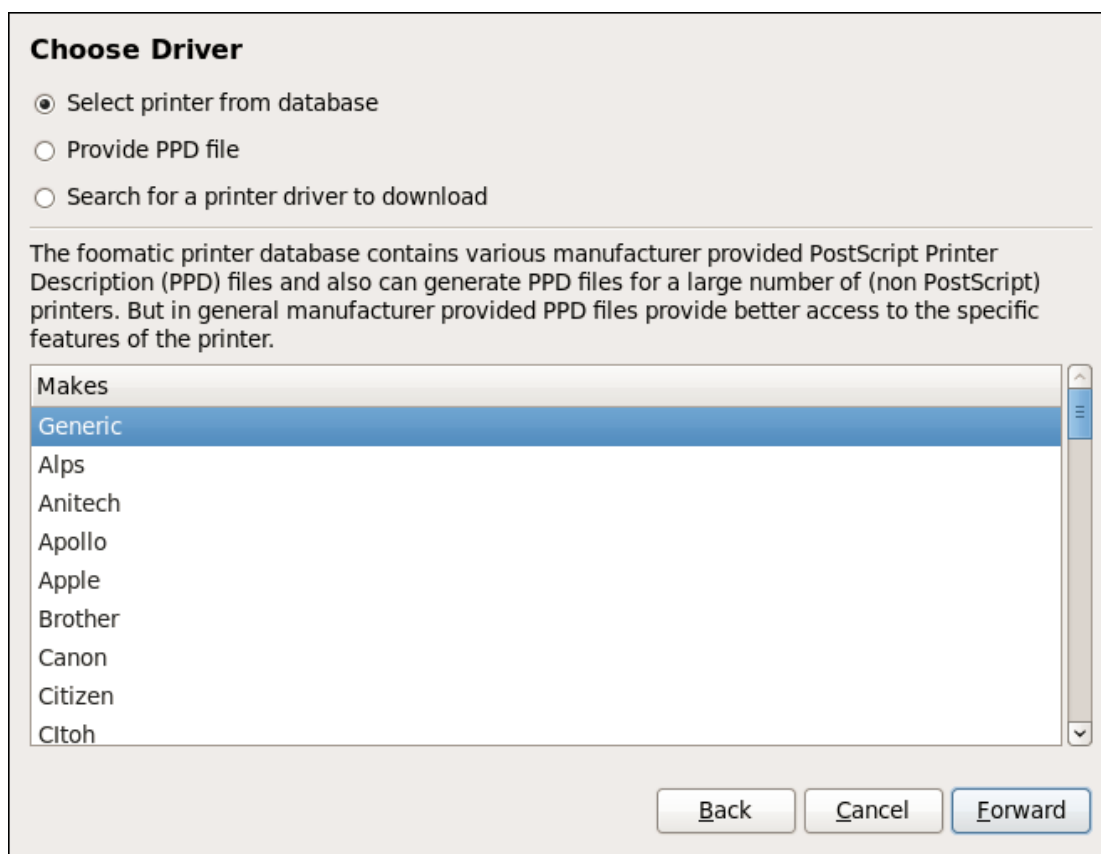


图 4-15 选择一个打印机品牌

- 2) 根据你之前的选择提供细节在下面显示的区域：
 - **Select printer from database** 选项：打印机品牌
 - **Provide PPD file** 选项：PPD 文件
 - **Search for a printer driver to download** 选项：打印机型号
- 3) 点击【下一步】继续。
- 4) 在左边选择相应的模型。

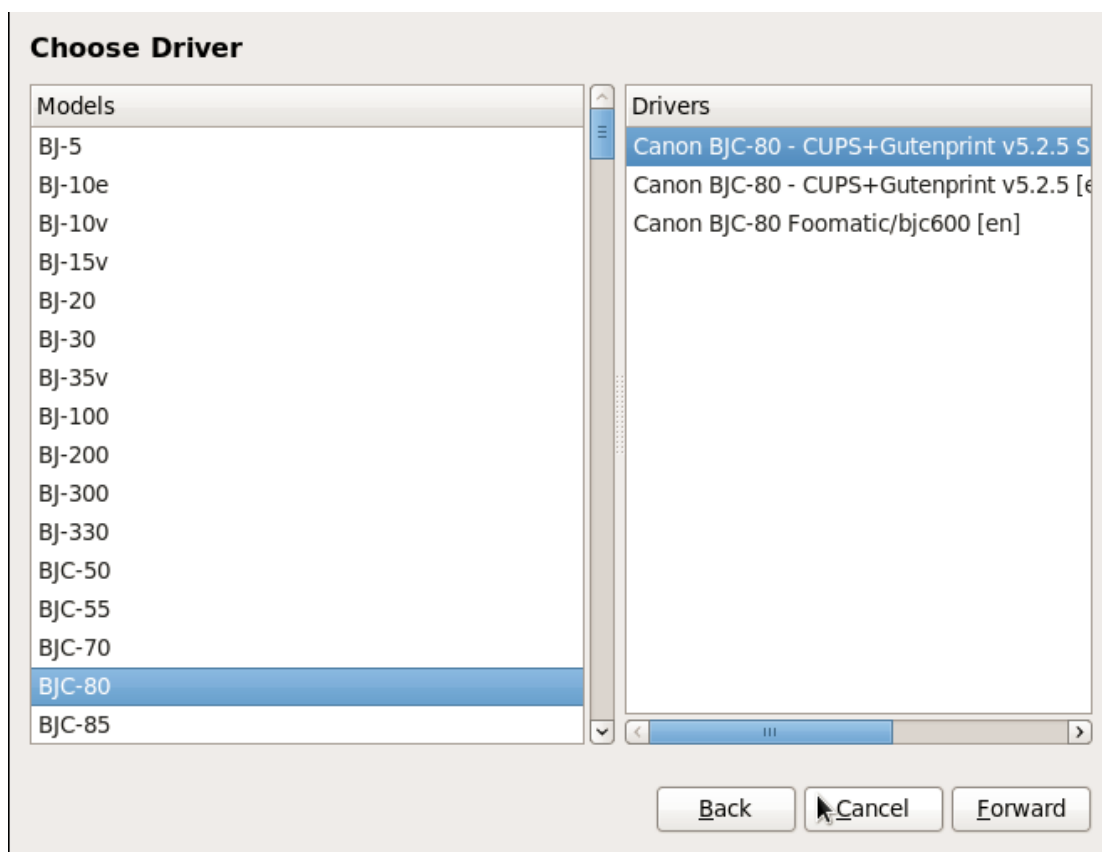
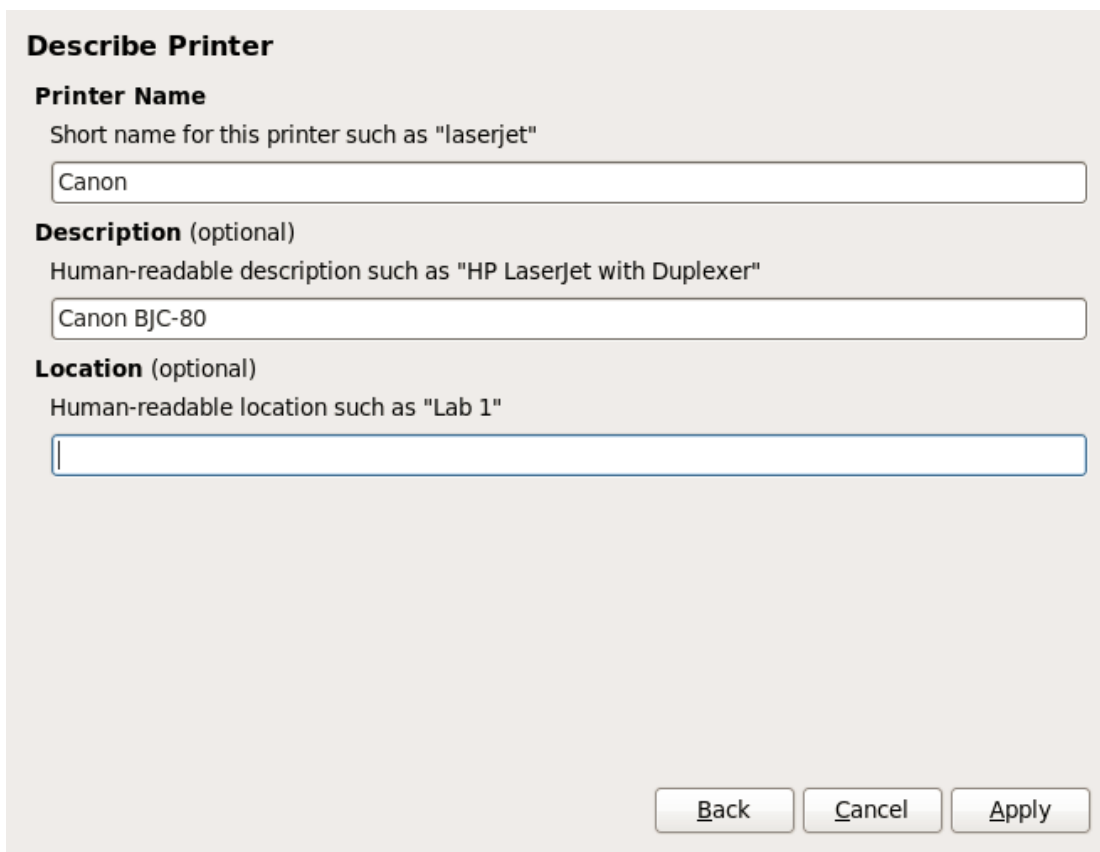


图 4-16 选择一个打印机型号

- 5) 点击【下一步】。
- 6) 在 Describe Printer 下边，在打印机名称字段，输入一个唯一的打印名称。打印机名称可以包含字母、数字破折号(-)和下划线(_);它不能包含空字符。你可以使用 Description 和 Location 区域添加更多打印机信息。这两个字段是可选的,可以为空。



The image shows a 'Describe Printer' dialog box. It has three sections: 'Printer Name' with a short name prompt and a text box containing 'Canon'; 'Description (optional)' with a human-readable description prompt and a text box containing 'Canon BJC-80'; and 'Location (optional)' with a human-readable location prompt and an empty text box. At the bottom right are three buttons: 'Back', 'Cancel', and 'Apply'.

图 4-17 打印机设置

- 7) 如果设置是正确的，单击 **Apply**，以确认您的打印机配置和添加打印队列。单击 **Back** 修改打印机配置。
- 8) 应用修改后，出现一个对话框允许您打印测试页。单击 **Yes** 打印测试页。参考打印测试页。

打印测试页

设置打印机或修改打印机配置后，打印测试页，确保打印机正常工作：

- 1) 打印机在打印窗口中点击鼠标右键，选择“属性”。
- 2) 在属性窗口中，点击左边的设置。
- 3) 在显示设置选项卡上，单击打印测试页按钮。

修改现有的打印机

删除现有的打印机，在打印设置窗口，选择打印机和打印机→删除。确认打印机删除。另外，按删除键。

设置默认打印机,在打印机列表中右键单击打印机,然后单击上下文菜单的设置为默认按钮。

设置页面

改变打印机驱动程序配置,双击打印机列表中对应的名称并单击左边的设置标签显示的设置页面。

您可以修改打印机设置例如打印机型号, 打印测试页,更改设备位置 (URI),等等。

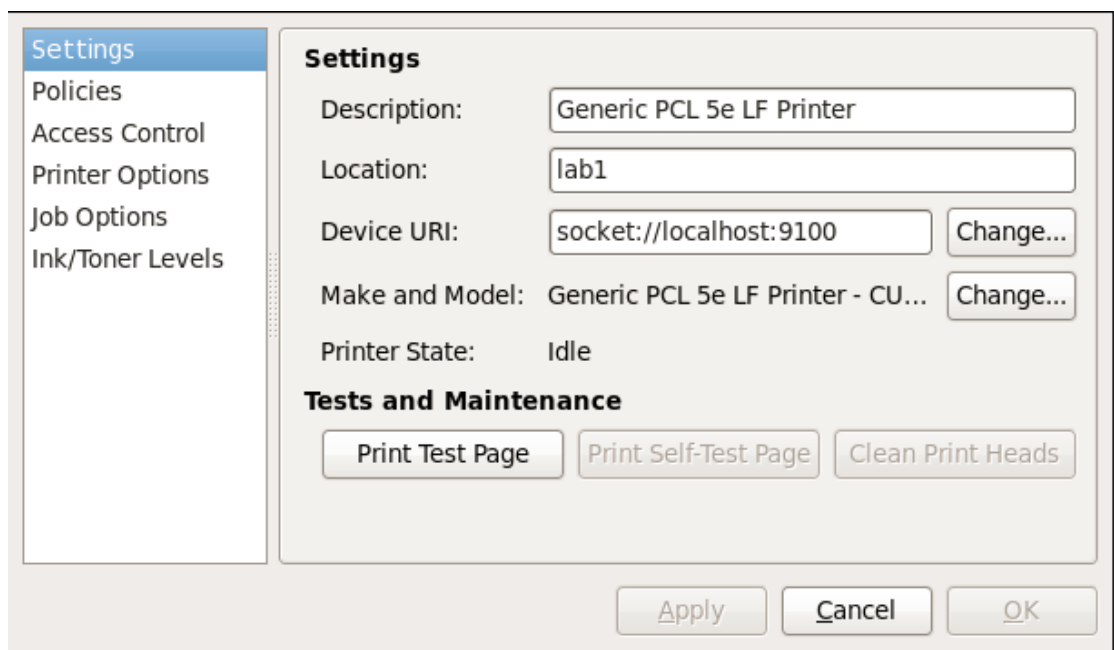


图 4-18 设置页

策略页面

点击 Policies 按钮, 修改打印机状态和打印机输出的设置

您可以选择打印机状态, 配置打印机的错误政策(你可以决定中止打印作业,重试,或停止它如果发生错误)。

你也可以创建一个 banner 页面 (这个页面描述了打印任务相关方面, 例如原打印机, 用户名, 打印文档的安全状态): 点击 Starting Banner 或者 Ending Banner 的下拉菜单, 选择最能描述打印作业的特性的选项 (例如 confidential)。

共享打印机

在策略页面，你可以标志一个打印机作为共享：假如一个打印机是共享的，网络用户可以使用它。为了能够使用共享功能，Server->Settings，选择共享打印机。

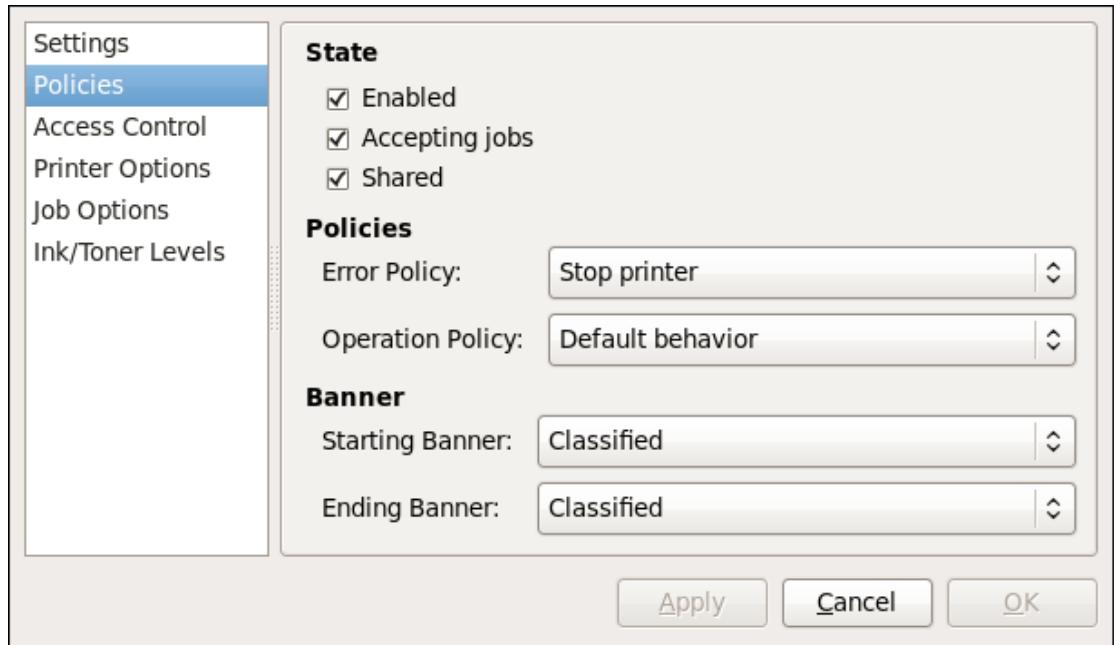


图 4-19 策略页

确保防火墙能够允许 TCP 接入端口 631 进入，该端口是 IPP 协议使用。

实例：在防火墙中开启 IPP 服务

- 1) 启动图形 firewall-config 工具,按 super 键进入活动概述、type firewall,然后按 enter 键。防火墙配置窗口打开。系统将提示您输入管理员或 root 密码或者使用命令行进入

```
~]# firewall-config
```

Firewall Configuration 窗口被打开。

寻找“连接”一词在左下角。这表明 firewall-config 工具连接到用户空间守护进程,firewalld。

为了马上修改当前防火墙的配置，确保 Configuration 标签下拉菜单设置为

Runtime。另一种选择，编辑配置，在下一次启动的时候应用，或者重启防火墙，在下拉菜单中选择 Permanent。

- 2) 选择 Zones 标签，选择防火墙区与要使用的网络接口。默认是公共区。选项卡显示了接口的接口已经被分配给一个区。
- 3) 选择 Services 标签，选择 IPP 服务可以分享。ipp-client 服务要求能够接收网络打印机。
- 4) 关闭防火墙配置工具。

访问控制页面

你可以在访问控制页面修改用户访问打印机的级别。点击【Access Control】标签。选择【Allow printing for everyone except these users】或者【Deny printing for everyone except these users】，输入用户名称，点击【add】按钮。

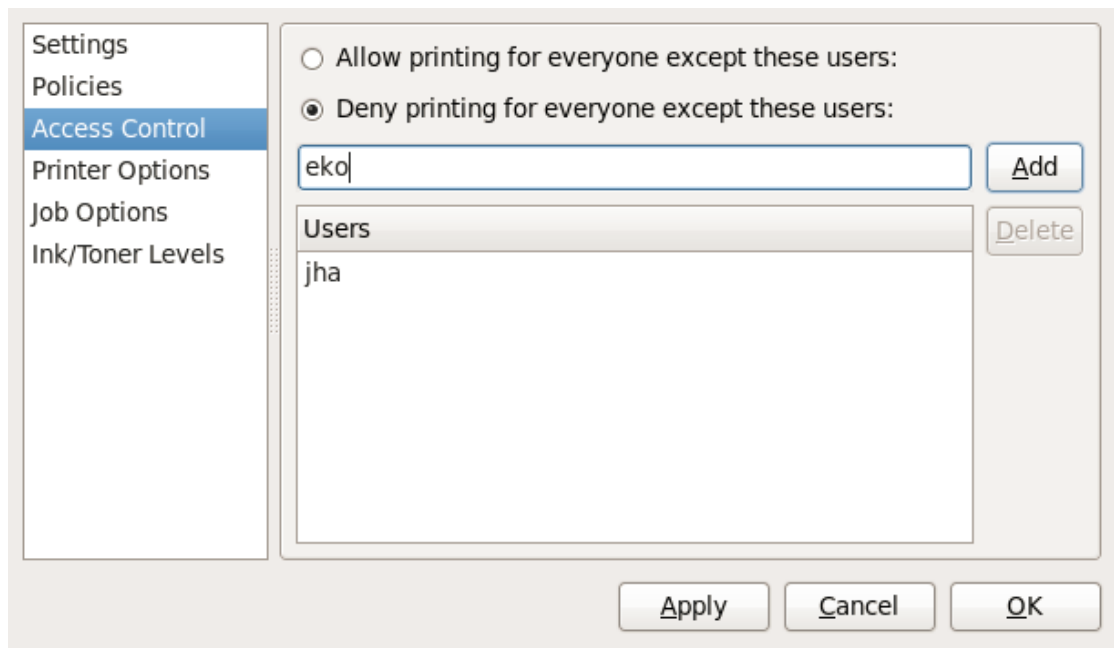


图 4-20 访问控制页

打印机选项页面

打印机选项页面包含各种配置选项的印刷媒体和输出,和它的内容可能会有所不同从打印机到打印机。它包含一般印刷、纸张、质量和印刷大小设置。

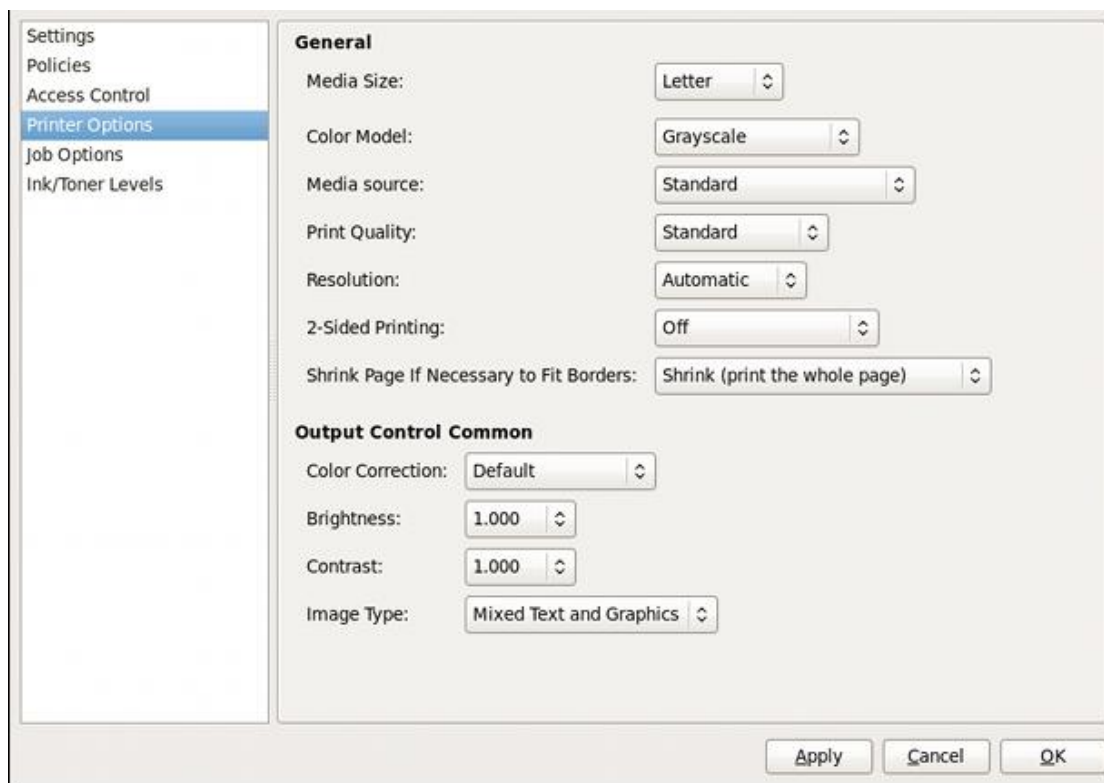


图 4-21 打印选项页

任务选项页面

在工作中选择页面上,您可以详细看到打印机工作的选项。点击左边的工作选择标签来显示页面。编辑默认设置应用自定义工作选项,如复制份数、定位、页面每一面,缩放(增加或减少可打印区域的大小,可以用来满足一个超大的打印区域到一个较小的物理表的打印介质),详细文本选项,选择和自定义工作。

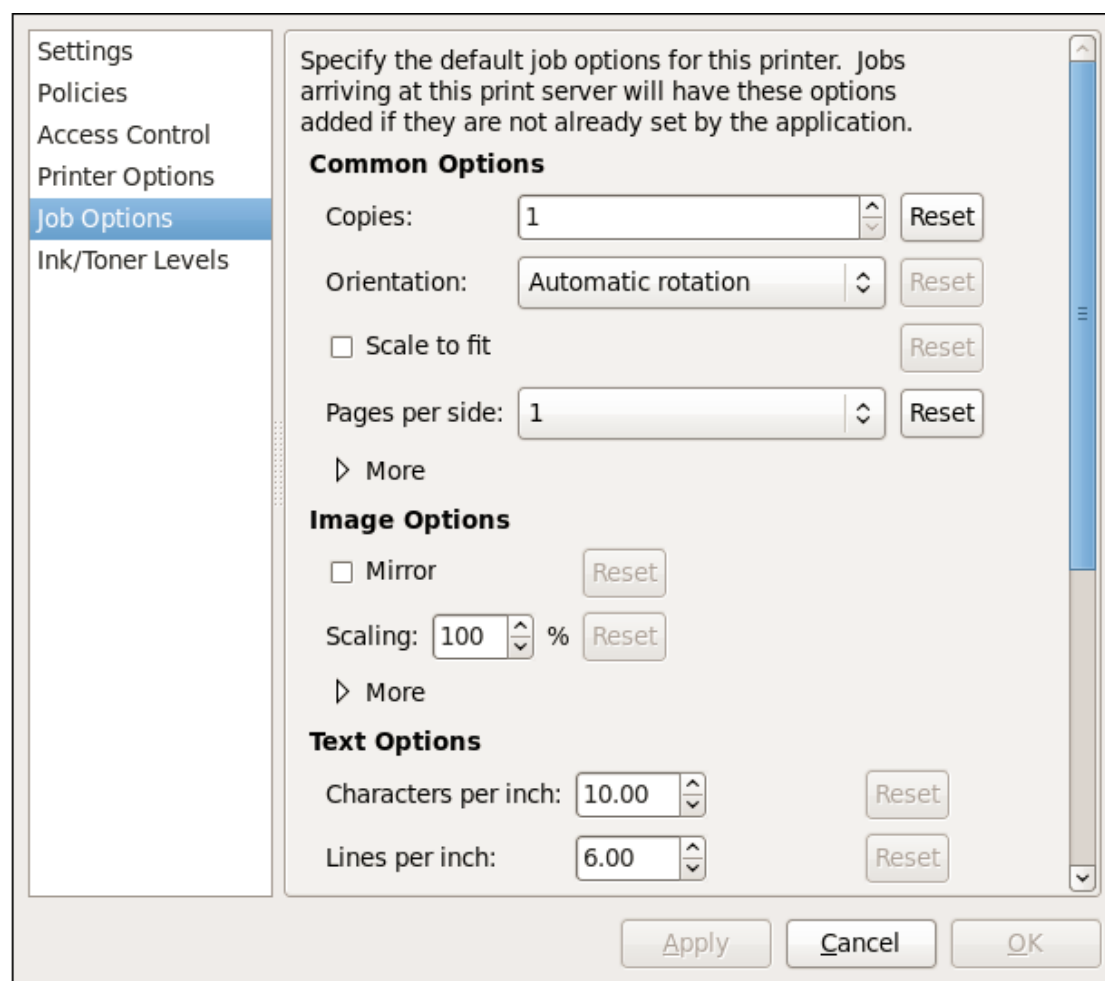


图 4-22 任务选项页

墨水或碳粉水平页面

墨水/碳粉水平页面包含墨粉状态细节，是否可用和打印机状态信息。点击左边的墨水/碳粉水平标签来显示页面。

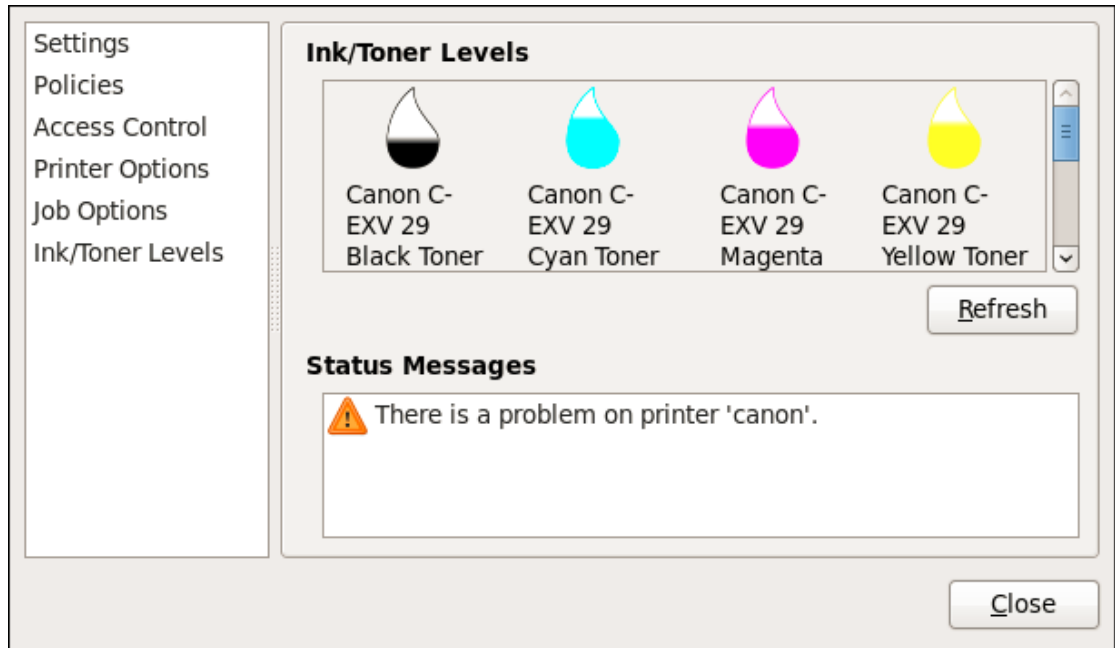





图 4-23Ink/Toner 页

4.4.3.1.1 管理打印作业

当你发送一个打印作业到打印机守护进程,如从 Emacs 打印文本文件或打印一个图像,打印的工作是添加到打印排队队列。打印排队队列是一个打印作业列表,发送到打印机和每个打印请求信息,如请求的状态,工作数量,等等。

在打印过程中,打印机状态图标出现在通知区域的面板。检查打印作业的状态,单击打印机地位,这类似于显示一个窗口。

File Job View					
Job	Document	Printer	Size	Time submitted	Status
2	Red Hat	Generic-PCL-5e-LF-...	5k	a minute ago	 Processing - Printer w...
1	Product Document...	Canon 	3k	22 hours ago	 Pending

Printer 'Generic-PCL-5e-LF-Printer': 'com.apple.print.recoverable'.

图 4-24GNOME 打印机状态

取消、持有、发布、转载或验证一个打印作业,在任务菜单,在 GNOME Print Status 选择任务, ,点击相应的命令。

使用 shell 命令查看打印任务列表,使用 `lpstat -o`

例如：lpstat -o 的输出。

```
$ lpstat -o
Charlie-60          twaugh          1024    Tue 08 Feb
2011 16:42:11 GMT
Aaron-61           twaugh          1024    Tue 08 Feb
2011 16:42:44 GMT
Ben-62             root            1024    Tue 08 Feb
2011 16:45:42 GMT
```

如果你想取消打印作业,使用请求命令 `lpstat - o` 找到任务号,然后使用命令 `cancel job number`。例如 `cancel 60` 将取消打印作业在上面例子中“lpstat -o 输出”。你不能使用 `cancel` 命令取消别人启动的打印作业。然而,你可以强制删除任务,通过 `cancel -U root job_number` 命令,为了防止这样的删除,修改打印操作策略验证进行强制 `root` 验证。

你可以通过 `shell` 命令打印文件。例如 `lp sample.txt`, 打印文件 `sample.txt`。

4.5 使用 chrony 套件配置 NTP

在 IT 行业,保持精确的时间是非常重要的,这有很多原因。例如,在网络上,包分发和日志是需要精确的时间戳的。在 linux 系统中, NTP 协议由守护进程运行在用户空间实现。

用户空间的守护进程更新运行在内核空间的系统时钟。系统时钟能够使用多种时钟资源保持时间准确。通常的使用 *Time Stamp Counter (TSC)*。TSC 是一个 CPU 寄存器用来计算周期数。它非常快,高分辨率,没有中断。

有两个守护进程的选择, `ntpd` 和 `chronyd`, 来自 `ntp` 和 `chrony` 包。本节描述 `chrony` 套件的使用的实用程序来更新系统时钟系统,不符合传统的永

久网络化。

4.5.1 chrony 套件介绍

Chrony 包含 chronyd, 一个运行在用户空间的守护进程, chronyc, 一个命令行程序, 对 chronyd 做出调整。系统, 如果不是永久的连接或经常开机, 会花费很多时间去校对系统时间通过 ntpd。

ntpd 和 chronyd 的差异

最主要的差异是用于控制计算机的时钟的算法。chronyd 能够比 ntpd 做的更好。

- 当外部基准时间只能够间歇性的访问时 chronyd 能够工作的很好。ntpd 需要可持续查询时间基准才能够工作的好。
- chronyd 能够在网络拥堵的时候工作的好
- chronyd 通常可以同步时钟更快和更好的时间精度
- chronyd 能够在时钟的速度突然变化时迅速适应,例如,由于晶体振荡器的温度的变化,而 ntpd 可能需要很长时间才能再次适应下来。
- 在默认配置中, 在系统启动后时间已经同步, chronyd 从未设置时间, 为了不打乱其他运行程序。ntpd 也可以配置为不同步时间, 但它必须使用不同的方式调整时钟,这会有一些缺点。
- chronyd 能够调整时间速率在 linux 操作系统上, 在一个很大的范围。这允许它在一个损坏或不稳定的机器上进行操作。例如, 在一些虚拟机上。

chronyd 能够做一些 ntpd 不能做的事情。

- chronyd 支持孤立网络时间校正的唯一方法是手动输入。例如,通过管理员看时钟。chronyd 能够在不同的更新中检查出错误信息, 评估计算机过多或丢失的时间速率。
- chronyd 提供支持工作的收益或损失的速度实时时钟,硬件时钟,维护计算机是关闭的时候。它可以使用这些数据在系统启动时设置系统时间使用调整后的实时时钟的时间的价值。写作时,仅可在 Linux。

ntpd 可以做 chronyd 不能做的事情。

- **ntpd** 支持 NTP 版本 4 (RFC 5905)，包括广播，多播，**manycast** 客户端和服务端，和孤儿模式。它还支持额外的身份验证方案基于公钥加密 (RFC 5906)。**chronyd** 使用 NTP 版本 3 (RFC 1305)，兼容版本 4。
- **ntpd** 包括许多参考时钟的驱动而 **chronyd** 依赖于其他项目,例如 **gpsd**,访问数据的参考时钟。

NTP 守护进程的选择

- **Chrony** 可以考虑在这些系统中使用，这些系统会经常暂停或间歇地断开连接和重新连接网络，例如移动和虚拟机系统。
- **NTP 守护进程 (ntpd)** 应该考虑用在经常保持不变的系统上。系统需要使用广播或多播 IP，或执行身份验证数据包的 **Autokey** 协议，应该考虑使用 **ntpd**。**chrony** 仅仅支持对称密钥身份验证，使用 MD5 消息身份验证代码(MAC)，SHA1 或者更强的哈希算法。而 **ntpd** 还支持 **Autokey** 认证协议，该协议可以利用 PKI 系统。**Autokey** 在 RFC5906 中有描述。

4.5.2 理解 CHRONY 及其配置

理解 chronyd

chronyd 是 **chrony** 的守护进程，运行在用户空间，调整运行在内核的系统时钟。它通过咨询外部时间源，使用 NTP 协议来进行调整。当外部引用并不可用，**chronyd** 将使用最后被计算存储在文件的数据。它还可以被 **chronyc** 手动进行修改。

理解 chronyc

chronyd 是 **chrony** 的守护进程，可以被命令行组件 **chronyc** 控制。这个工具提供了一个命令提示符输入一些命令可以对 **chronyd** 进行更改。默认的配置 **chronyd** 仅仅接收本地的 **chronyc** 命令，**chronyc** 可以配置为 **chronyd** 可以接收外部控制。**chronyc** 可以远程运行后第一个配置 **chronyd** 接受远程连接。IP 地址允许连接到 **chronyd** 应严格控制。

理解 chrony 配置命令

chronyd 默认的配置文件的 `/etc/chrony.conf`, `-f` 选项可以指定一个配置文件的路径。查看 `chronyd` 的 `man` 手册获取更多的信息。查看 <http://chrony.tuxfamily.org/manual.html#Configuration-file> 获取更多指令列表。

Comments

Comments should be preceded by `#`, `%`, `;` or `!`

allow

可选择的, 指定一个主机, 子网或者网络连接一台可以作为 NTP 服务器的机器。默认是不允许连接。

例如:

```
allow server1.example.com
```

通过主机名, 指定一个主机, 运行连接

```
allow 192.0.2.0/24
```

指定一个网络允许连接

```
allow 2001:db8::/32
```

指定一个 IPv6 地址

cmdallow

该命令和 `allow` 命令相似, 除了它允许特定的子网或主机的控制接入(而不是 NTP 客户端接入)。(控制接入, 意思是 `chronyc` 能够运行在其它主机上并且能够通过本地计算机连接到 `chronyd`) 他的语法是一样的。 `cmddeny all` 命令和 `cmdallow all` 命令类似。

dumpdir

保存 `chronyd` 服务重启的测量历史目录路径。

dumponexit

假如该命令是运行的，它表明 **chronyd** 必须为所有的时间源保存测量的历史。

local

local 关键字是允许 **chronyd** 通过客户端推送输入进行时间同步，即使它没有同步源。该选项经常被用来在 **master** 主机上使用，在一个独立的网络中，许多计算机需要同步，**master** 主机需要通过手动输入保持准确时间。

例如：

```
local stratum 10
```

log

log 命令表明某些信息将要被记录日记。它接收以下选项：

measurements

该选项记录了测量以及相关信息，生成 **measurements.log** 文件

statistics

该选项记录了回归处理相关信息，生成 **statistics.log** 文件

tracking

该选项记录了系统的收益或损失的估计，生成 **tracking.log** 文件

rtc

这个选项记录了系统的实时时钟信息

refclocks

这个选项记录了原始和过滤参考时钟测量，生成 **refclocks.log**。

tempcomp

这个选项记录温度测量和系统补偿速率，生成 **tempcomp.log** 文件
log 日志记录文件存放在 **logdir** 指定的目录

```
log measurements statistics tracking
```

logdir

指定日志文件存放的目录

```
logdir /var/log/chrony
```

makestep

通常，**chronyd** 将根据需求通过减慢或加速时钟，使得系统逐步纠正所有时间偏差。在某些特定情况下，系统时钟可能会漂移过快，导致该调整过程消耗很长的时间来纠正系统时钟。该指令强制 **chronyd** 在调整期大于某个阈值时步进调整系统时钟，但只有在因为 **chronyd** 启动时间超过指定限制（可使用负值来禁用限制），没有更多时钟更新时才生效。

```
makestep 1000 10
```

maxchange

该指令将指定最大修正偏移量，当偏移量大于指定的阈值，**chronyd** 将会放弃并且退出，将消息发送给 **syslog**

```
maxchange 1000 1 2
```

第一个时钟更新后，**chronyd** 将会检查每一个时钟偏移量的更新，将会忽略 2 次大于阈值的偏移量修正。

maxupdateskew

chronyd 的任务之一就是要相对于其源而言计算机的时钟运行的快或慢的程度。**maxupdateskew** 参数是确定估计是否是否可靠的阈值，缺省情况下，阈值是 1000ppm，语法格式如下：

```
maxupdateskew skew-in-ppm
```

noclientlog

这个命令没有参数，客户端不记录 log 日志

reselectdist

当 **chronyd** 选择同步源可用的来源,它将更喜欢最小的同步距离。然而,为了避免频繁的重新选择有来源相似的距离时,一个固定的距离被添加。默认情况下,距离是 100 微秒。

格式如下

```
reselectdist dist-in-seconds
```

stratumweight

stratumweight 指令设置当 **chronyd** 从可用源中选择同步源时,每个层应该添加多少距离到同步距离。默认情况下, CentOS 中设置为 0, 让 **chronyd** 在选择源时忽略源的层级。

格式如下

```
stratumweight dist-in-seconds
```

默认情况下, *dist-in-seconds* 为 1 秒

rtcfile

rtcfile 指令定义了一个文件的名称,该文件, **chronyd** 可以保存跟踪系统的实时时钟的准确性(RTC)参数。语法的格式是:

```
rtcfile /var/lib/chrony/rtc
```

rtcsync

rtcsync 指令将启用一个内核模式,在该模式中,系统时间每 11 分钟会拷贝到实时时钟 (RTC)。

chronyc 的安全性

和 Network Time Protocol (NTP) 相比, PTP 最主要的优点是硬件的支持,包括许多的 network interface controllers (NIC)和 network switches。极大的提

高了时间同步的准确性。

4.5.3 使用 chrony

安装 chrony

使用 root 用户登录

```
~]# yum install chrony
```

默认的 **chrony** 进程位置 `/usr/sbin/chronyd`。命令行组件安装在 `/usr/bin/chronyc`。

检查 **chronyd** 状态

```
~]$ systemctl status chronyd
chronyd.service - NTP client/server
    Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled)
    Active: active (running) since Wed 2013-06-12 22:23:16 CEST; 11h
ago
```

启动 **chronyd**

启动

```
~]# systemctl start chronyd
```

设置开机自启动

```
~]# systemctl enable chronyd
```

关闭 **chronyd**

关闭

```
~]# systemctl stop chronyd
```

取消开机自启动

```
~]# systemctl disable chronyd
```

检查 chrony 是否同步

为了检查 chrony 是否同步, 使用 `tracking`, `sources` 和 `sourcestats` 命令。

检查 chrony Tracking

命令如下

```
~]$ chronyc tracking
Reference ID      : 1.2.3.4 (a.b.c)
Stratum          : 3
Ref time (UTC)   : Fri Feb  3 15:00:29 2012
System time      : 0.000001501 seconds slow of NTP time
Last offset      : -0.000001632 seconds
RMS offset       : 0.000002360 seconds
Frequency        : 331.898 ppm fast
Residual freq    : 0.004 ppm
Skew             : 0.154 ppm
Root delay       : 0.373169 seconds
Root dispersion  : 0.024780 seconds
Update interval  : 64.2 seconds
Leap status      : Normal
```

字段如下:

Reference ID

服务器同步地址。假如是 127.127.1.1, 表示该计算机是没有跟外部源进行同步, 你正在使用 local 模式操作。

Stratum

拥有的层级数

Ref time

最后一次同步后测量的 UTC 时间

System time

系统时间

Last offset

最后一次预估的偏移量

RMS offset

平均偏移量

Frequency

如果 chronyd 没有进行纠错, 系统时间出错的速率, 例如: 1ppm 意思是系统时间 1 秒, 可能实际时间是 1.000001 秒。

Residual freq

显示当前选中源的剩余频率

Skew

frequency 的估计误差

Root delay

这是网络路径延迟到最终同步计算机时的 stratum-1 计算机的总和。在某些极端的情况下, 此值可以为负。

检查 chrony 来源

命令如下

```
~]$ chronyc sources
```

210 Number of sources = 3					
MS Name/IP address	Stratum Poll Reach LastRx Last sample				
=====					
=====					
#* GPS0	0	4	377	11	-479ns[-621ns] +/-
134ns					
^? a.b.c	2	6	377	23	-923us[-924us] +/-
43ms					
^+ d.e.f	1	6	377	21	-2629us[-2619us] +/-
86ms					

M

表示源的模式。^表示一个服务器，=表示一个 peer，#表示本地连接的参考时钟

S

表示源的状态。*表示正在同步，+表示已连接，-表示已被排除，? 表示连接已丢失。“x”表示一个时钟 chronyd 认为是 falseticker(与大多数其他来源的时间是不一致的)，“~”表示一个源的时间似乎有太多的变化。

Name/IP address

源的 IP 地址或名称

Stratum

源的层，层 1 表明计算机附带本地参考时钟，与层 1 同步的计算机在层 2，与层 2 同步的计算机在层 3，以此类推。

Poll

显示源被 polled 的速率，例如值为 6 的时候，就表示每 64s 进行一次测量。

Reach

显示最后一个收到的源的时间。一般是在几秒钟之间。字母 m h d 或 y

显示分钟,小时,几天或几年。10 年的值表示没有收到这个源。

LastRx

显示最后一个收到的源的时间。一般是在几秒钟之间。字母 m h d 或 y 显示分钟,小时,几天或几年。10 年的值表示没有收到这个源。

Last sample

此列显示本地时钟与最新的测量数据源之间的偏移量。

检查 chrony 来源统计

sourcestats 命令显示当前被检查的源的漂移速度和偏移量的信息。

-v 参数能够被指定,意思是冗长的。在这种情况下,额外的行显示为一个提醒列。

```
~]$ chronyc sourcestats
```

210 Number of sources = 1						
Name/IP Address	NP	NR	Span	Frequency	Freq	
Skew	Offset	Std Dev				
=====						
=====						
abc.def.ghi	11	5	46m	-0.001	0.045	
1us	25us					

Name/IP address

NTP 服务器的地址

NP

这是样本点的数量目前为服务器保留。漂移速率和电流偏移估计通过这些点进行线性回归。

NR

这是运行的 residuals 具有相同回归符号的数量。

Span

这是最古老的和最新的样本之间的时间间隔。如果没有显示单元的值,

则表示是在几秒钟内。在这个例子中,间隔是 46 分钟。

Frequency

估计的剩余频率,在这种情况下,计算机的时钟估计是 $1/10^9$ 相对于服务器来说

Freq Skew

估计的误差界限

Offset

估计的偏移量

Std Dev

这是估计样本标准差

手动调整系统时钟

命令如下

```
~]# chronyc
chrony> password commandkey-password
200 OK
chrony> makestep
200 OK
```

4.5.4 为不同的环境设置 chrony

为一个很少连接的系统设置 chrony

这个例子是用于系统使用 dial-on-demand 连接。正常的配置应满足移动和虚拟设备连接断断续续。首先,审查和确认/etc/chrony.默认设置配置类似于以下几点:

```
driftfile /var/lib/chrony/drift
commandkey 1
```

```
keyfile /etc/chrony.keys
```

`command key ID` 是在安装时候生成，应符合 `commandkey` 值，
`/etc/chrony.keys`。

添加 NTP 服务器

```
server 0.pool.ntp.org offline
server 1.pool.ntp.org offline
server 2.pool.ntp.org offline
server 3.pool.ntp.org offline
```

为一个独立网络系统设置 `chrony`

在一个从来不和外部网络进行连接的独立的网络中，可以选择一台计算机作为时间主机，其它主机作为主机的客户端。

在 `master` 主机上，编辑 `/etc/chrony.conf`

```
driftfile /var/lib/chrony/drift
commandkey 1
keyfile /etc/chrony.keys
initstepslew 10 client1 client3 client6
local stratum 8
manual
allow 192.0.2.0
```

`192.0.2.0` 是可以允许连接的子网地址

客户端机器上，编辑 `/etc/chrony.conf`

```
server master
```

```
driftfile /var/lib/chrony/drift
logdir /var/log/chrony
log measurements statistics tracking
keyfile /etc/chrony.keys
commandkey 24
local stratum 10
initstepslew 20 master
allow 192.0.2.123
```

192.0.2.123 是 master 主机的地址。

4.5.5 使用 chronyc

使用 chronyc 控制 chronyd

进入 chronyc 的交互界面

```
~]# chronyc -a
```

chronyc 必须以 root 用户执行。-a 选项是使用本地 keys 自动登录
chronyc 命令行界面

```
chronyc>
```

你可以使用 help 显示所有命令
也可以使用非交互界面进行输入

```
chronyc command
```

使用 chronyc 进行远程管理

配置 chrony 连接到远程 chronyd，格式如下：

```
~]$ chronyc -h hostname
```

指定端口

```
~]$ chronyc -h hostname -p port
```

port 是用来控制和监控远程 chronyd

第一条命令必须输入密码

```
chronyc> password password  
200 OK
```

密码不允许有空格

假如密码不是 MD5 哈希，哈希密码必须被 authhash 命令在前。

```
chronyc> authhash SHA1  
chronyc> password  
HEX:A6CFC50C9C93AB6E5A19754C246242FC5471BCDF  
200 OK
```

4.6 配置 NTP 使用 NTPD

4.6.1 NTP 介绍

NTP 是网络时间协议(Network Time Protocol)，它是用来同步网络中各个计算机的时间的协议。

4.6.2 NTP 分层

NTP 分层如下

Stratum 0:

原子钟和信号广播电台和 GPS

- GPS (Global Positioning System)
- Mobile Phone Systems
- Low Frequency Radio Broadcasts WWVB (Colorado, USA.), JJY-40 and JJY-60 (Japan), DCF77 (Germany), and MSF (United Kingdom)

这些信号可以被专用的设备接收，并经常被 RS-232 连接到系统作为一个组织或站点范围内的时间服务器

Stratum 1:

电脑附加了无线时钟、GPS 时钟或原子钟

Stratum 2:

从 Stratum 1 读取，作为更底层的服务器

Stratum 3:

从 Stratum 2 读取，作为更底层的服务器

Stratum n+1:

从 Stratum n 读取，作为更底层的服务器

Stratum 15:

从 Stratum 14 读取，作为更底层的服务器

4.6.3 理解 NTP

NeoKylin Linux Advanced Server V7 使用的 NTP 版本，在 RFC 1305 Network Time Protocol (Version 3) Specification, Implementation and Analysis 和 RFC 5905 Network Time Protocol Version 4: Protocol 和 Algorithms Specification 有详细描述。

4.6.4 理解 drift 文件

drift 文件记录保存着系统时间频率与 UTC 时钟源频率的偏移量。

4.6.5 UTC, TIMEZONES 和 DST

NTP 完全在 UTC (Universal Time, Coordinated)中, Timezones 和 DST (Daylight Saving Time) 被本地系统应用。/etc/localtime 是 /usr/share/zoneinfo 的拷贝或链接。RTC 可能在本地时间或者 UTC 中, 在 /etc/adjtime 第三行指定。这个文件可以知道 RTC 怎么被设置。用户可以很方便的使用 System Clock Uses UTC 在 Date and Time 图形配置界面进行配置的修改。

4.6.6 NTP 身份验证选项

在当前网络, 攻击者可以通过发送 NTP 包进行攻击。在使用公共的 NTP 源时, 为了减少风险, 在/etc/ntp.conf 文件中, 必须有 3 个公共源。假如只有一个源, ntpd 将会忽略该源。根据应用的风险需要, 可以选择开启或不开启身份验证。

默认的组播和广播是需要验证的。假如你决定关闭身份验证, 可以使用 disable auth 在 ntp.conf 文件中。

4.6.7 在虚拟机中管理时间

虚拟机不能使用真实的 hardware clock 并且虚拟机时间不够稳定。在 NeoKylin Linux Advanced Server V7 中, kvm 默认使用 kvm-clock。

4.6.8 理解闰秒

闰秒, 是指为保持协调世界时接近于世界时时刻, 由国际计量局统一规定在年底或年中(也可能在季末)对协调世界时增加或减少 1 秒的调整。由于地球自转的不均匀性和长期变慢性(主要由潮汐摩擦引起的), 会使世界时(民用时)和原子时之间相差超过到 ± 0.9 秒时, 就把世界时向前拨 1 秒(负闰秒, 最后一分钟为 59 秒)或向后拨 1 秒(正闰秒, 最后一分钟为 61 秒); 闰秒一般加在公历年末或公历六月末。

4.6.9 理解 ntpd 配置文件

守护进程 ntpd 在系统启动或重启时读取配置文件/etc/ntp.conf, 你可以通过下面命令查看配置文件

```
~]$ less /etc/ntp.conf
```

下面介绍默认的配置项

The driftfile entry

drift 文件路径

```
driftfile /var/lib/ntp/drift
```

The access control entries

以下行设置默认访问控制限制:

```
restrict default nomodify notrap nopeer noquery
```

nomodify 选项: 阻止修改配置文件

notrap 选项: 防止 ntpdc 控制消息协议陷阱

nopeer 选项: 防止 peer 联合的形成

noquery 选项: 防止 ntpq 和 ntpdc 查询, 但没有时间查询

有时候各种进程和应用需要 127.0.0.0/8 地址段。默认 restrict 行阻止所有非允许的接入

```
# the administrative functions.
```

```
restrict 127.0.0.1
```

```
restrict ::1
```

如果特别需要被另一个应用程序, 可以在下面添加地址。

主机在本地网络是不允许的, 因为默认的 restrict。为了进行修改, 例如, 允许 192.0.2.0/24 网络进行时间的查询

```
restrict 192.0.2.0 mask 255.255.255.0 nomodify notrap nopeer
```

如果只是允许某一台主机，例如 192.0.2.250/32

```
restrict 192.0.2.250
```

如果没有指定 mask，255.255.255.255 将会被指定

The public servers entry

默认的，ntp.conf 配置文件包含四个公共服务器

```
server 0.rhel.pool.ntp.org iburst
server 1.rhel.pool.ntp.org iburst
server 2.rhel.pool.ntp.org iburst
server 3.rhel.pool.ntp.org iburst
```

The broadcast multicast servers entry

默认的，ntp.conf 包含了很多被注释掉的例子，这些在很大程度上是自我解释。

4.6.10 理解 ntpd 的 sysconfig 文件

这个文件将会在 ntpd 启动服务初始化脚本的时候读取。默认的内容如下：

```
# Command line options for ntpd
OPTIONS="-g"
```

-g 选项可以使 ntpd 忽略 1000s 的偏移量限制，尝试同步时间即使偏移量大于 1000s，但是仅仅在系统启动的时候。当离开这个选项的时候，ntpd 在偏移量大于 1000s 的时候自动退出。

4.6.11 禁止 chrony

命令如下


```
~]# systemctl stop chronyd
```

取消 chrony 开机启动选项

```
~]# systemctl disable chronyd
```

查看状态

```
~]$ systemctl status chronyd
```

4.6.12 检查 NTP 守护进程是否安装

命令如下

```
~]# yum install ntp
```

4.6.13 ntpd 的安装

```
~]# yum install ntp
```

设置 ntpd 开机启动

```
~]# systemctl enable ntpd
```

4.6.14 检查 ntp 的状态

检查 ntpd 是否运行

```
~]$ systemctl status ntpd
```

获取 ntpd 的状态报告

```
~]$ ntpstat  
unsynchronised  
time server re-starting  
polling server every 64 s
```

```
~]$ ntpstat  
synchronised to NTP server (10.5.26.10) at stratum 2  
time correct to within 52 ms  
polling server every 1024 s
```

4.6.15 配置防火墙允许 ntp 包进入

ntp 使用 UDP 包，端口 123 。必须要能够允许通过防火墙。

检查防火墙时候允许 ntp 传输，使用图形界面 **Firewall Configuration** 工具

启动 **firewall-config**，点击 **Super** 键进入，输入 **firewall** 并且按回车键，防火墙配置窗口被打开。输入用户密码。

命令行进入

```
~]# firewall-config
```

寻找“连接”一词在左下角。这表明 **firewall-config** 工具连接到用户空间守护进程,firewalld。

修改 firewall 配置

为了能够让配置生效，确保下拉菜单 **Configuration** 是设置为 **Runtime**。

或者修改配置文件在下次启动生效，在下拉菜单中选择 **Permanent**

打开防火墙端口

打开 `firewall-config` 工具，选择网络，选择端口标签，点击“添加”按钮。

Port and Protocol 窗口被打开，输入端口 123，下拉菜单选择 **udp**

4.6.16 配置 ntpdate 服务器

`ntpdate` 服务是为了设置时间在系统启动的时候。

检查 `ntpdate` 服务是否运行

```
~]$ systemctl status ntpdate
```

设置开机启动

```
~]# systemctl enable ntpdate
```

在 NeoKylin Linux Advanced Server V7 系列，默认 `/etc/ntp/step-tickers` 文件包含 `0.rhel.pool.ntp.org`。添加额外的 `ntpdate` 服务器，编辑 `/etc/ntp/step-tickers`。服务器的数量是不重要的，因为 `ntpdate` 只是使用这个获取一次时间信息。假如你有一个外网的时间服务器，在第一行输入该服务器的地址。在第二行输入备份的服务器。

4.6.17 配置 ntp

修改默认的 `ntp` 服务器配置，编辑 `/etc/ntp.conf` 文件。该文件是和 `ntpd` 一起被安装。

配置 NTP 服务访问控制

编辑 `ntp.conf`，使用 `restrict` 命令

```
# Hosts on local network are less restricted.

#restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap
```

`restrict` 使用

restrict option

option: 可以是一个和多个

- ignore——所有的数据包将被忽略, 包括 ntpq 和 ntpdc 查询
- kod——一个“Kiss-o'-death”包发送以减少不必要的查询
- limited——如果数据包违反了速率限制, 不响应服务请求。ntpq 和 ntpdc 查询不受影响
- lowpriotrap——低优先级匹配主机设定的陷阱
- nomodify——阻止修改配置
- noquery——阻止 ntpq 和 ntpdc 查询, 不包括时间查询
- nopeer——阻止 peer 联合形成
- noserve——除了 ntpq 和 ntpdc 查询, 拒绝所有的包
- notrap——防止 ntpdc 控制消息协议陷阱
- notrust——拒绝没有密码身份验证数据包
- ntpport——修改算法, 只匹配 NTP UDP port123
- version——拒绝不匹配当前 NTP 版本的数据包

为了不相应所有的查询, 配置速率限制, restrict 可以使用 limited 选项。假如 ntpd 必须响应 KoD 数据包, restrict 必须使用 limited 和 kod 选项。

ntpq 和 ntpdc 查询可用于放大的攻击, 不要移除 noquery 选项。

配置连接 NTP 服务器的速率限制

给 restrict 加上 limited 选项, 如果你不想使用默认放弃的参数, 可以使用 discard 命令。

命令格式如下

```
discard [average value] [minimum value] [monitor value]
```

- average——指定允许的最小平均数据包间隔, 它接受一个参数 log2 秒。

默认值是 3 (2 的三次方, 相当于 8)

- `minimum`——指定允许的最低包间距, 在 \log_2 秒它接受一个参数。默认值是 1 (2 的一次方, 相当于 2 秒)
- `monitor`——指定数据包的丢弃概率, 一旦允许利率已经超过限制。默认值为 3000 秒。这个选项适用于服务器每秒获得 1000 或更多请求

`discard` 命令例子

```
discard average 4
```

```
discard average 4 minimum 2
```

添加 `peer` 地址

在 `ntp.conf` 配置文件添加 `peer` 命令

```
peer address
```

`address` 是一个 ip 地址或 DNS 可识别的名称。**`address`** 必须是在同一层的系统。**Peers** 必须拥有至少一个不同的时间源。

添加服务器地址

添加一个服务器地址, 该服务器是运行 NTP 服务且处在更高层。在 `ntp.conf` 文件, 使用 `server` 命令

```
server address
```

`address` 是可识别的 IP 地址或 DNS 识别的名称。

添加一个广播或多播服务器地址

添加一个广播或多播发送地址, 也就是说, 广播或多播 NTP 包到达的地址。在 `ntp.conf` 文件中, 使用 `broadcast`

使用如下

```
broadcast address
```

添加一个 Manycast 客户地址

添加 **manycast** 客户端地址，在 `ntp.conf` 文件中，添加 `manycastclient` 命令。

格式如下

```
manycastclient address
```

该命令配置系统作为一个 **NTP** 客户端。系统可以同时为客户端和服务端。

添加 Broadcast 客户端地址

添加 **broadcast** 客户端地址，在 `ntp.conf` 文件中，添加 `broadcastclient` 命令。

格式如下

```
broadcastclient
```

该命令配置系统作为一个 **NTP** 客户端。系统可以同时为客户端和服务端。

添加一个 Manycast 服务器地址

添加 **manycast** 服务器地址，在 `ntp.conf` 文件中，添加 `manycastserver` 命令

格式如下

```
manycastserver address
```

该命令配置系统作为一个 **NTP** 服务器。系统可以同时为客户端和服务端。

添加一个 **Multicast** 服务器地址

添加 **multicast** 服务器地址，在 `ntp.conf` 文件中，添加 `multicastclient` 命令

格式如下

```
multicastclient address
```

该命令配置系统作为一个 **NTP** 服务器。系统可以同时为客户端和服务端。

配置 **Burst** 选项

不要和公共 **NTP** 服务器一起使用该选项，该选项适用于在自己组织里的应用程序。

在服务器命令结尾添加该选项

```
burst
```

配置 **iburst** 选项

在服务器命令结尾添加该选项

```
iburst
```

使用 **key** 配置 **Symmetric Authentication**

在服务器或 `peer` 命令后，添加该选项

```
key number
```

number 是一个 1 到 65534 的数。该选项可以在数据包中使用 *message authentication code (MAC)* 。该选项和 *peer*, *server*, *broadcast*, 和 *manycastclient* 命令使用。

/etc/ntp.conf, 格式如下:

```
server 192.168.1.1 key 10
broadcast 192.168.1.255 key 20
manycastclient 239.255.254.254 key 30
```

配置 Poll Interval

修改默认的 poll interval, 在服务器或 *peer* 命令后, 添加该选项

```
minpoll value and maxpoll value
```

配置服务器优先级

指定一个高优先级的服务器, 在 *server* 或 *peer* 命令后添加该选项

```
prefer
```

为 NTP 数据包配置 Time-to-Live

在 *server* 或 *peer* 命令后添加该选项

```
ttl value
```

配置 NTP 使用版本

在 *server* 或 *peer* 命令后添加该选项


```
version value
```

4.6.18 配置硬件时钟更新

配置系统时间更新硬件时钟（RTC），在/etc/sysconfig/ntpdate 添加以下选项

```
SYNC_HWCLOCK=yes
```

命令使用如下

```
~]# hwclock --systohc
```

4.6.19 配置时钟源

在系统列出可用的时钟源

```
~]$ cd /sys/devices/system/clocksource/clocksource0/
clocksource0]$ cat available_clocksource
kvm-clock tsc hpet acpi_pm
clocksource0]$ cat current_clocksource
kvm-clock
```

可以看出，该内核使用的是 kvm-clock。因为这个是虚拟机。

覆盖默认的时钟源，在内核的 GRUB 里添加 clocksource，例如

```
~]# grubby --args=clocksource=tsc --update-kernel=DEFAULT
```

4.7 使用 ptp4l 配置 PTP

4.7.1 PTP 介绍

Precision Time Protocol (PTP) 是用于网络中时钟同步的协议。使用时结合硬件支持,要比普通的 NTP 更快。PTP 的支持被划分为内核和用户空间。红帽企业版 linux 内核支持 PTP 时钟。linuxptp 是该协议的实现。

这个 linuxptp 包包含 ptp4l 和 phc2sys 时钟同步项目。ptp4l 程序实现了 PTP 边界时钟和普通时钟。与硬件时间戳,它用于 PTP 硬件时钟与主时钟同步,与软件时间戳,它用于系统时钟与主时钟同步。phc2sys 程序只需与硬件时间戳,将系统时钟同步到 PTP 硬件时钟网络接口卡(NIC)。

理解 PTP

PTP 的时钟同步可以理解为主从层次结构。从节点与它们的主节点进行时钟同步,该主节点有可能有它们自己的主节点。该层次是由 *best master clock (BMC)*算法创建和更新,该算法运行在所有的时钟。当一个时钟只有一个端口,那么它可以是主或是从,被称作 *ordinary clock (OC)*。当有多个端口的时候,它既可以是主也可以是从,被称作 *boundaryclock (BC)*。最顶层的 master 被称作 *grandmaster clock*, 可以和 *Global Positioning System (GPS)*时间源进行同步。

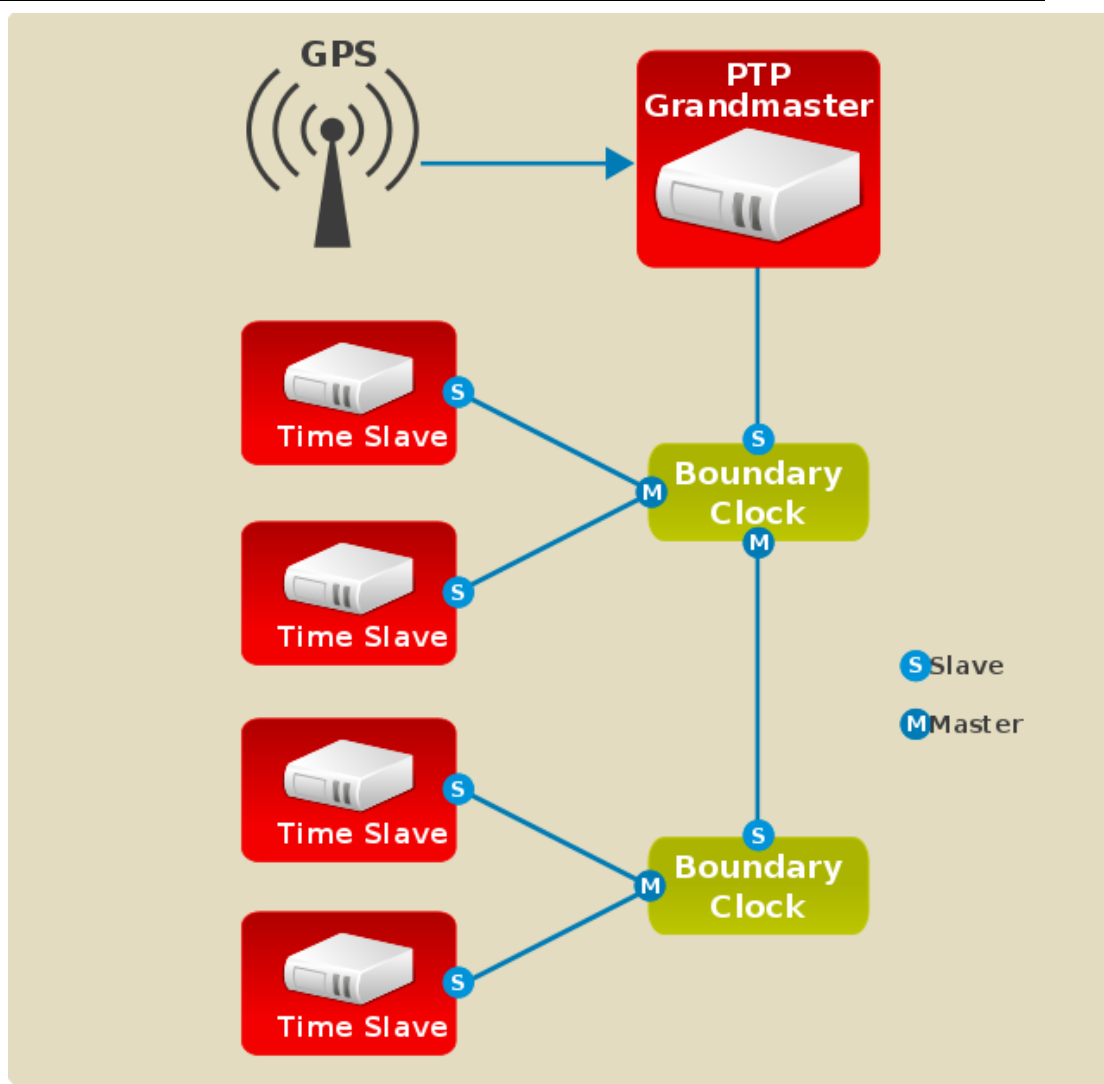


图 4-25 PTP grandmaster, boundary, and slave Clocks

PTP 的优点

和 Network Time Protocol (NTP) 相比，PTP 最主要的优点是硬件的支持，包括许多的 network interface controllers (NIC) 和 network switches。极大的提高了时间同步的准确性。

4.7.2 使用 PTP

为了使用 PTP，内核网络驱动必须支持软件或硬件时间戳。

检查驱动和硬件支持

使用 ethtool 命令查询

```
~]# ethtool -T eth3

Time stamping parameters for eth3:

Capabilities:

    hardware-transmit
(SOF_TIMESTAMPING_TX_HARDWARE)

    software-transmit
(SOF_TIMESTAMPING_TX_SOFTWARE)

    hardware-receive
(SOF_TIMESTAMPING_RX_HARDWARE)

    software-receive
(SOF_TIMESTAMPING_RX_SOFTWARE)

    software-system-clock
(SOF_TIMESTAMPING_SOFTWARE)

    hardware-raw-clock
(SOF_TIMESTAMPING_RAW_HARDWARE)

PTP Hardware Clock: 0

Hardware Transmit Timestamp Modes:

    off                (HWTSTAMP_TX_OFF)
    on                 (HWTSTAMP_TX_ON)

Hardware Receive Filter Modes:

    none              (HWTSTAMP_FILTER_NONE)
    all               (HWTSTAMP_FILTER_ALL)
```

eth3 是你想检查的接口

如果支持软时间戳，必须包含以下参数

- SOF_TIMESTAMPING_SOFTWARE
- SOF_TIMESTAMPING_TX_SOFTWARE

➤ SOF_TIMESTAMPING_RX_SOFTWARE

如果支持硬件时间戳，必须包含以下参数

➤ SOF_TIMESTAMPING_RAW_HARDWARE

➤ SOF_TIMESTAMPING_TX_HARDWARE

➤ SOF_TIMESTAMPING_RX_HARDWARE

安装 PTP

```
~]# yum install linuxptp
```

这将会安装 ptp4l 和 phc2sys。

启动 ptp4l

ptp4l 可以通过命令行或者作为服务启动。当 ptp4l 作为一个服务时，可以在 /etc/sysconfig/ptp4l 指定选项。不管是作为服务或是命令行启动，必须早 /etc/ptp4l.conf 指定选项。/etc/sysconfig/ptp4l 文件包含 -f /etc/ptp4l.conf 行，这会让 ptp4l 读取 /etc/ptp4l.conf 文件。

作为服务启动 ptp4l

```
~]# systemctl start ptp4l
```

使用命令行运行 ptp4l

```
~]# ptp4l -i eth3 -m
```

输出结果如下

```
~]# ptp4l -i eth3 -m
selected eth3 as PTP clock
port 1: INITIALIZING to LISTENING on INITIALIZE
port 0: INITIALIZING to LISTENING on INITIALIZE
port 1: new foreign master 00a069.ffe.0b552d-1
```

```
selected best master clock 00a069.ffe.0b552d
port 1: LISTENING to UNCALIBRATED on RS_SLAVE
master offset -23947 s0 freq +0 path delay      11350
master offset -28867 s0 freq +0 path delay      11236
master offset -32801 s0 freq +0 path delay      10841
master offset -37203 s1 freq +0 path delay      10583
master offset  -7275 s2 freq -30575 path delay   10583
port 1: UNCALIBRATED to SLAVE on
MASTER_CLOCK_SELECTED
master offset  -4552 s2 freq -30035 path delay   10385
```

记录 ptp4l 日志

默认情况下，日志文件是发送到/var/log/messages。-m 选项打开日志的标准输出，能够为 debugging 提供信息。

打开软时间戳，-s 选项，使用如下

```
~]# ptp4l -i eth3 -m -S
```

选择一个延迟测量机制

有 2 种不同的延迟测量机制，能够通过不同的选项进行选择：

-P

-P 选择 *peer-to-peer* (P2P) 延迟测量机制

P2P 机制是首选，因为它对网络拓扑的变化更快，可能比其他机制更准确测量延迟。

-E

-E 选择 *end-to-end* (E2E)延迟测量机制。这是默认的选项

E2E 也被认为是 *request-response* 延迟机制

-A

-A 打开延迟机制的自动选择

自动选项打开的是 ptp4l 的 E2E 模式。如有需要，可以改成 P2P 模式。

4.7.3 和多个接口使用 PTP

在不同网络的多接口中使用 PTP，有必要改变反向路径转发模式为松散模式。

sysctl 组件用来对内核进行读写。对正在运行的系统，可以使用命令行或者是修改/etc/sysctl.conf 文件

修改为 loose 模式，命令如下

```
~]# sysctl -w net.ipv4.conf.default.rp_filter=2
sysctl -w net.ipv4.conf.all.rp_filter=2
```

为了对每一个接口修改反向路径转发模式，使用 net.ipv4.interface.rp_filter 命令行，例如，em1 网络接口

```
~]# sysctl -w net.ipv4.conf.em1.rp_filter=2
```

为了让修改的配置永久有效，修改/etc/sysctl.conf 文件。例如：为了修改所有的网卡的模式，修改/etc/sysctl.conf，如下所示

```
net.ipv4.conf.all.rp_filter=2
```

如果是修改单一的某个网卡模式，如下所示：

```
net.ipv4.conf.interface.rp_filter=2
```

4.7.4 指定一个配置文件

没有默认的配置文件，所以需要在运行的时候指定，使用-f 选项

```
~]# ptp4l -f /etc/ptp4l.conf
```

一个配置文件内容，相当于-i eth3 -m -S 选项，如下所示

```
~]# cat /etc/ptp4l.conf
[global]
```

```

verbose                1

time_stamping          software

[eth3]
    
```

4.7.5 使用 PTP 管理客户端

PTP 的管理客户端，pmc 可以用来获取额外 ptp4l 信息。

```

~]# pmc -u -b 0 'GET CURRENT_DATA_SET'

sending: GET CURRENT_DATA_SET

90e2ba.ffe.20c7f8-0 seq 0 RESPONSE MANAGMENT
CURRENT_DATA_SET

stepsRemoved           1

offsetFromMaster       -142.0

meanPathDelay          9310.0
    
```

```

~]# pmc -u -b 0 'GET TIME_STATUS_NP'

sending: GET TIME_STATUS_NP

90e2ba.ffe.20c7f8-0 seq 0 RESPONSE MANAGMENT
TIME_STATUS_NP

master_offset          310

ingress_time

1361545089345029441

cumulativeScaledRateOffset +1.000000000

scaledLastGmPhaseChange  0

gmTimeBaseIndicator     0

lastGmPhaseChange

0x0000'0000000000000000.0000
    
```


gmPresent	true
gmIdentity	00a069.ffff.0b552d

查看 pmc 全部命令

```
~]# pmc help
```

4.7.6 同步时钟

phc2sys 程序用来将系统时间同步到 PTP 的 hardware clock (PHC)。

phc2sys 服务配置文件/etc/sysconfig/phc2sys。默认配置如下：

```
OPTIONS="-a -r"
```

-a 选项使得 phc2sys 能够同步来自 ptp4l 应用的时钟。它将跟随 PTP 端口状态的变化，相应调整网卡的硬件之间的同步时钟，系统时钟将不会被同步，除非-r 选项被指定。如果你想让系统时钟成为一个源，指定-r 选项两次。

修改/etc/sysconfig/phc2sys 后，重启 phc2sys 服务

```
~]# systemctl restart phc2sys
```

正常情况下，使用 systemctl 命令来启动，停止和重启 phc2sys 服务。

如果你不想使用服务启动 phc2sys，你可以通过命令行来启动。

```
~]# phc2sys -a -r
```

-a 选项使得 phc2sys 能够同步来自 ptp4l 应用的时钟。如果你想让系统时钟成为一个源，指定-r 选项两次。

使用-s 选项同步系统时钟至特定的 PTP 硬件时钟，例如：

```
~]# phc2sys -s eth3 -w
```

-w 选项等待运行的 ptp4l 应用同步 PTP 时钟，恢复 TAI 至 UTC 偏移量

正常情况下，PTP 操作在 *International Atomic Time* (TAI)，系统时间保持在 *Coordinated Universal Time* (UTC)。当前 TAI 和 UTC 的偏移量是 35s。当闰秒插入或删除的时候，偏移量会进行改变，这个变化每几年会发生一次。当

-w 选项没有使用时，可以使用-O 选项来进行偏移量的设置

```
~]# phc2sys -s eth3 -O -35
```

当 phc2sys servo 处于锁状态时，时钟将不会走，除非-S 选项能够被使用。这意味着 phc2sys 程序必须在 ptp4l 程序以及和 PTP 硬件时钟同步后启动。使用 -w 选项，phc2sys 不用在 ptp4l 后启动，因为它会等待，直到 ptp4l 已经同步。

phc2sys 可以作为服务启动

```
~]# systemctl start phc2sys
```

当以服务进行启动，可以将选项在/etc/sysconfig/phc2sys 文件指定。

4.7.7 验证时间同步

当 PTP 时间同步工作正常的时候，新的频率偏移量以及调整消息将会打印 ptp4l 和 phc2sys。这些信息可以在 /var/log/messages 文件查看。

```
ptp4l[352.359]: selected /dev/ptp0 as PTP clock
ptp4l[352.361]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[352.361]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[353.210]: port 1: new foreign master 00a069.ffff.0b552d-1
ptp4l[357.214]: selected best master clock 00a069.ffff.0b552d
ptp4l[357.214]: port 1: LISTENING to UNCALIBRATED on
RS_SLAVE
ptp4l[359.224]: master offset      3304 s0 freq      +0 path delay
9202
ptp4l[360.224]: master offset      3708 s1 freq     -29492 path delay
9202
ptp4l[361.224]: master offset     -3145 s2 freq     -32637 path delay
9202
```

```

ptp4l[361.224]: port 1: UNCALIBRATED to SLAVE on
MASTER_CLOCK_SELECTED

ptp4l[362.223]: master offset      -145 s2 freq  -30580 path delay
9202

ptp4l[363.223]: master offset      1043 s2 freq  -29436 path delay
8972

ptp4l[364.223]: master offset       266 s2 freq  -29900 path delay
9153

ptp4l[365.223]: master offset       430 s2 freq  -29656 path delay
9153

ptp4l[366.223]: master offset       615 s2 freq  -29342 path delay
9169

ptp4l[367.222]: master offset      -191 s2 freq  -29964 path delay
9169

ptp4l[368.223]: master offset       466 s2 freq  -29364 path delay
9170

ptp4l[369.235]: master offset        24 s2 freq  -29666 path delay
9196

ptp4l[370.235]: master offset      -375 s2 freq  -30058 path delay
9238

ptp4l[371.235]: master offset       285 s2 freq  -29511 path delay
9199

ptp4l[372.235]: master offset       -78 s2 freq  -29788 path delay
9204

```

phc2sys 输出的例子

```

phc2sys[526.527]: Waiting for ptp4l...
phc2sys[527.528]: Waiting for ptp4l...

```

phc2sys[528.528]: phc offset	55341 s0 freq	+0 delay	
2729			
phc2sys[529.528]: phc offset	54658 s1 freq	-37690 delay	2725
phc2sys[530.528]: phc offset	888 s2 freq	-36802 delay	2756
phc2sys[531.528]: phc offset	1156 s2 freq	-36268 delay	2766
phc2sys[532.528]: phc offset	411 s2 freq	-36666 delay	2738
phc2sys[533.528]: phc offset	-73 s2 freq	-37026 delay	2764
phc2sys[534.528]: phc offset	39 s2 freq	-36936 delay	2746
phc2sys[535.529]: phc offset	95 s2 freq	-36869 delay	2733
phc2sys[536.529]: phc offset	-359 s2 freq	-37294 delay	2738
phc2sys[537.529]: phc offset	-257 s2 freq	-37300 delay	2753
phc2sys[538.529]: phc offset	119 s2 freq	-37001 delay	2745
phc2sys[539.529]: phc offset	288 s2 freq	-36796 delay	2766
phc2sys[540.529]: phc offset	-149 s2 freq	-37147 delay	2760
phc2sys[541.529]: phc offset	-352 s2 freq	-37395 delay	2771
phc2sys[542.529]: phc offset	166 s2 freq	-36982 delay	2748
phc2sys[543.529]: phc offset	50 s2 freq	-37048 delay	2756
phc2sys[544.530]: phc offset	-31 s2 freq	-37114 delay	2748
phc2sys[545.530]: phc offset	-333 s2 freq	-37426 delay	2747
phc2sys[546.530]: phc offset	194 s2 freq	-36999 delay	2749

ptp4l 有一个命令 `summary_interval`，可以减少输出，默认情况下，每一秒会打印一条信息。可以修改为每 1024s 打印一次，在 `/etc/ptp4l.conf` 文件添加以下行：

```
summary_interval 10
```

一个 ptp4l 输出的例子，使用 `summary_interval 6`

```
ptp4l: [615.253] selected /dev/ptp0 as PTP clock
```

```

ptp4l: [615.255] port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l: [615.255] port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l: [615.564] port 1: new foreign master 00a069.ffe.0b552d-1
ptp4l: [619.574] selected best master clock 00a069.ffe.0b552d
ptp4l: [619.574] port 1: LISTENING to UNCALIBRATED on
RS_SLAVE
ptp4l: [623.573] port 1: UNCALIBRATED to SLAVE on
MASTER_CLOCK_SELECTED
ptp4l: [684.649] rms   669 max 3691 freq -29383 ± 3735 delay  9232 ±
122
ptp4l: [748.724] rms   253 max   588 freq -29787 ± 221 delay  9219 ±
158
ptp4l: [812.793] rms   287 max   673 freq -29802 ± 248 delay  9211 ±
183
ptp4l: [876.853] rms   226 max   534 freq -29795 ± 197 delay  9221 ±
138
ptp4l: [940.925] rms   250 max   562 freq -29801 ± 218 delay  9199 ±
148
ptp4l: [1004.988] rms  226 max   525 freq -29802 ± 196 delay  9228
± 143
ptp4l: [1069.065] rms  300 max   646 freq -29802 ± 259 delay  9214
± 176
ptp4l: [1133.125] rms  226 max   505 freq -29792 ± 197 delay  9225
± 159
ptp4l: [1197.185] rms  244 max   688 freq -29790 ± 211 delay  9201
± 162

```

减少从 phc2sys 的输出，可以使用 -u 选项

```
~]# phc2sys -u summary-updates
```

summary-updates 是时钟更新数，例子如下：

```
~]# phc2sys -s eth3 -w -m -u 60
phc2sys[700.948]: rms 1837 max 10123 freq -36474 ±4752 delay
2752 ± 16
phc2sys[760.954]: rms 194 max 457 freq -37084 ±174 delay 2753
± 12
phc2sys[820.963]: rms 211 max 487 freq -37085 ±185 delay 2750
± 19
phc2sys[880.968]: rms 183 max 440 freq -37102 ±164 delay 2734
± 91
phc2sys[940.973]: rms 244 max 584 freq -37095 ±216 delay 2748
± 16
phc2sys[1000.979]: rms 220 max 573 freq -36666 ±182 delay
2747 ± 43
phc2sys[1060.984]: rms 266 max 675 freq -36759 ±234 delay
2753 ± 17
```

4.7.8 使用 NTP 服务 PTP 时间

ntpd 守护进程可以配置从系统时间分发时间，系统时间是由 ptp4l 和 phc2sys 使用 LOCAL 时间驱动进行同步。为了防止 ntpd 调整系统时钟，ntp.conf 文件必须不能够指定任何 NTP 服务器。以下是一个例子：

```
~]# cat /etc/ntp.conf
server 127.127.1.0
fudge 127.127.1.0 stratum 0
```

4.7.9 使用 PTP 服务 NTP 时间

NTP 至 PTP 同步也是可能的。当 ntpd 是用来同步系统时间，ptp4l 可以使用 priority1 选项配置为 grandmaster 时钟，通过 PTP 系统时钟来分发时间。

```
~]# cat /etc/ptp4l.conf

[global]

priority1 127

[eth3]

# ptp4l -f /etc/ptp4l.conf
```

使用硬件时间戳，需要使用 phc2sys 来同步 PTP 硬件时钟至系统时钟。假如 phc2sys 是一个服务，编辑/etc/sysconfig/phc2sys 文件，默认的配置如下

```
OPTIONS="-a -r"
```

使用 root 用户，编辑

```
~]# vi /etc/sysconfig/phc2sys

OPTIONS="-a -r -r"
```

在这里，-r 选项被指定 2 次，允许 PTP 网卡硬件时钟从系统时钟进行同步。重启 phc2sys

```
~]# systemctl restart phc2sys
```

防止 PTP 时钟频率快速的变化，可以通过设置更小的 P (proportional) 和 I (integral)

```
~]# phc2sys -a -r -r -P 0.01 -I 0.0001
```

4.7.10 使用 timemaster 同步 PTP 或 NTP 时间

可以使用 timemaster 程序让系统时间跟可用的时间源进行同步。PTP 时

间是由 phc2sys 和 ptp4l 提供, 通过使用 shared memory driver。NTP 守护进程能够比较所有的源, 选择最优的源进行同步。

启动时, timemaster 会读取一个配置文件, 该配置文件会指定 NTP 和 PTP 时间源, 检查哪些网络接口有自己或共享的 PTP 硬件时钟, 生成 ptp4l 和 chronyd 或 ntpd 配置文件, 启动 ptp4l, phc2sys 和 phc2sys 或 ntpd。

作为一个服务启动 timemaster

使用 root 用户

```
~]# systemctl start timemaster
```

启动时, 会读取/etc/timemaster.conf 配置文件

理解 timemaster 配置文件

默认的配置如下所示:

```
~]$ less /etc/timemaster.conf

# Configuration file for timemaster

#[ntp_server ntp-server.local]

#minpoll 4

#maxpoll 4

#[ptp_domain 0]

#interfaces eth0

[timemaster]

ntp_program chronyd

[chrony.conf]
```



```
include /etc/chrony.conf
```

```
[ntp.conf]
```

```
includefile /etc/ntp.conf
```

```
[ptp4l.conf]
```

```
[chronyd]
```

```
path /usr/sbin/chronyd
```

```
options -u chrony
```

```
[ntpd]
```

```
path /usr/sbin/ntpd
```

```
options -u ntp:ntp -g
```

```
[phc2sys]
```

```
path /usr/sbin/phc2sys
```

```
[ptp4l]
```

```
path /usr/sbin/ptp4l
```

注意到部分命名如下

```
[ntp_server address]
```

这是一个 NTP 服务器区域的例子，ntp-server.local 是一个本地 LAN 的 NTP 服务器。

注意到部分命名如下

```
[ptp_domain number]
```

PTP domain 是一组 PTP 时钟，它们能够相互同步。它们可以或者不可以

和其它域进行同步。拥有相同域的数字组成一个域。这包括一个 PTP 的 grandmaster 时钟。

注意到部分命名如下

```
[timemaster]
```

默认的 **timemaster** 配置文件包含了 **ntpd** 和 **chrony** 配置 (/etc/ntp.conf 或者 /etc/chronyd.conf)，为了能够包含访问限制的配置和认证密钥。这意味着任何指定的 NTP 服务器能够和 **timemaster** 一起被使用。

配置 **timemaster** 选项

实例：编辑 **timemaster** 配置文件

- 1) 打开/etc/timemaster.conf 配置文件。
- 2) 对于每一个 NTP 服务器，你想控制使用 **timemaster**，创建[ntp_server address] 字段。
- 3) 添加要在域中使用的网卡，编辑#[ptp_domain 0]字段并添加网卡，例子如下：

```
[ptp_domain 0]
    interfaces eth0

[ptp_domain 1]
    interfaces eth1
```

- 4) 假如需要使用 **ntpd** 作为 NTP 守护进程，在[timemaster]字段修改默认的 entry，**chronyd** 改为 **ntpd**。
- 5) 假如使用 **chronyd** 作为 NTP 服务器，在[chrony.conf]字段添加额外的选项，在默认的 include /etc/chrony.conf entry 下。
- 6) 假如使用 **ntpd** 作为 NTP 服务器，在[chrony.conf]字段添加额外的选项，在默认的 include /etc/ntp.conf entry 下。
- 7) 在[ptp4l.conf]字段，添加任何将要拷贝到 **ptp4l** 生成的配置文件的选项。
- 8) 在[chronyd]字段，添加任何命令行，当被 **timemaster** 访问时该命令需要传递

至 chronyd。

- 9) 在[ntpd]字段，添加任何命令行，当被 timemaster 访问时该命令需要传递至 ntpd。
- 10) 在[phc2sys]字段，添加任何命令行，当被 timemaster 访问时该命令需要传递至 phc2sys。
- 11) 在[ptp4l]字段，添加任何命令行，当被 timemaster 访问时该命令需要传递至 ptp4l。
- 12) 保存配置文件，重启 timemaster。

```
~]# systemctl restart timemaster
```

4.7.11 提高准确性

ptp4l 和 phc2sys 应用能够配置为使用新的 adaptive servo。对于 PI servo 来说，它的优势在于不需要配置 PI 优化配置文件。在 ptp4l 中使用，需要在 /etc/ptp4l.conf 配置文件添加以下行：

```
clock_servo linreg
```

重启 ptp4l 服务

```
~]# systemctl restart ptp4l
```

在 phc2sys 中使用，需要在 /etc/sysconfig/phc2sys 配置文件添加以下行：

```
-E linreg
```

重启 phc2sys 服务

```
~]# systemctl restart phc2sys
```

5 监控和自动化

5.1 系统监控工具

在配置系统之外，掌握收集基本的系统信息的方法也很重要。譬如，您应该知道如何找出空闲内存的数量、可用硬盘空间，硬盘分区方案，以及正在运行进程的信息等等。本节将说明如何使用几个简单程序来从您的中标麒麟服务器操作系统中检索这类信息。

5.1.1 查看系统进程

使用 `ps` 命令

`Ps` 命令显示系统运行进程的信息。它会生成一个静态列表，列表里的进程是当你执行命令时系统运行的进程的一个快照。如果你想持续更新实时进程列表，你需要使用 `top` 命令和系统监控应用。

如果想列出当前系统中的所有进程，可以在 `shell` 里使用如下命令

```
ps ax
```

对于每一个列出的进程，`ps ax` 命令会显示进程的 **PID**,进程依附的终端，进程状态，进程占用的 **CPU** 时间，可执行文件的名字。例如：

```
~]$ ps ax

PID TTY STAT TIME COMMAND

1 ? Ss 0:01 /usr/lib/systemd/systemd --switched-root --system
--deserialize 23

2 ? S 0:00 [kthreadd]

3 ? S 0:00 [ksoftirqd/0]

5 ? S> 0:00 [kworker/0:0H]

[output truncated]
```

如果想显示进程的所有者，使用如下命令：

```
ps aux
```

除了 `ps ax` 命令提供的信息外，`ps aux` 还显示进程拥有者的名字，进程占用

CPU 和内存的百分比，以字节为单位显示虚拟内存大小和未交换物理内存的大小，进程开始运行的时间。如图：

```
~]$ ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.3 0.3 134776 6840 ? Ss 09:28 0:01
/usr/lib/systemd/systemd --switched-root --system --d
root 2 0.0 0.0 0 0 ? S 09:28 0:00 [kthreadd]
root 3 0.0 0.0 0 0 ? S 09:28 0:00 [ksoftirqd/0]
root 5 0.0 0.0 0 0 ? S> 09:28 0:00 [kworker/0:0H]

[output truncated]
```

您可以使用 `ps` 命令和 `grep` 命令的组合来查看某进程是否在运行。譬如，要判定 Emacs 是否在运行，使用下面这个命令：

```
~]$ ps ax | grep emacs

12056 pts/3 S+ 0:00 emacs
12060 pts/2 S+ 0:00 grep --color=auto emacs
```

有关可用命令行选项的完整列表，查看 `ps(1)` man 手册。

使用 `top` 命令

`Top` 命令显示系统运行中进程的实时列表。它还会显示附加信息包括系统更新时间，当前 CPU 和内存的使用率，运行的进程总数。这样你可以对进程做更进一步的操作，例如可以给进程排序或者杀死一个进程。

运行 `top` 命令，在 shell 里输入如下命令：

```
top
```

对于列出的进程，`top` 命令会显示进程的 PID,进程所有者的名字，进程优先级，进程 NICE 值，进程使用的虚拟内存，进程使用的未交换的物理内存，进程使用的共享内存，进程的状态，进程使用内存和 CPU 的百分比，进程占用 CPU 的时间和可执行文件的名字。

例如：

```
~]$ top
top - 16:42:12 up 13 min,  2 users,  load average: 0.67, 0.31, 0.19
Tasks: 165 total,  2 running, 163 sleeping,   0 stopped,   0 zombie
%Cpu(s): 37.5 us,  3.0 sy,  0.0 ni, 59.5 id,  0.0 wa,  0.0 hi,  0.0 si,
0.0 st
KiB Mem : 1016800 total,    77368 free,   728936 used,   210496
buff/cache
KiB Swap: 839676 total,   776796 free,    62880 used.  122628 avail
Mem

  PID USER      PR  NI   VIRT   RES    SHR S  %CPU  %MEM    TIME+
COMMAND
 3168 sjw      20   0 1454628 143240  15016 S   20.3   14.1   0:22.53 gnome-
shell
 4006 sjw      20   0 1367832 298876  27856 S   13.0   29.4   0:15.58
firefox
 1683 root      20   0  242204  50464   4268 S    6.0    5.0   0:07.76 Xorg
 4125 sjw      20   0  555148  19820  12644 S    1.3    1.9   0:00.48 gnome-
terminal-
  10 root      20   0         0         0         0 S    0.3    0.0   0:00.39
rcu_sched
 3091 sjw      20   0   37000   1468    904 S    0.3    0.1   0:00.31 dbus-
daemon
 3096 sjw      20   0  129688   2164   1492 S    0.3    0.2   0:00.14 at-
spi2-registr
 3925 root      20   0         0         0         0 S    0.3    0.0   0:00.05
kworker/0:0
   1 root      20   0  126568   3884   1052 S    0.0    0.4   0:01.61
systemd
   2 root      20   0         0         0         0 S    0.0    0.0   0:00.00
kthreadd
```

表格 5-1 交互 top 命令可以和 top 一起使用的互动命令包括（更多的信息查看 top(1)man 手册）：

表格 5-1 交互 top 命令

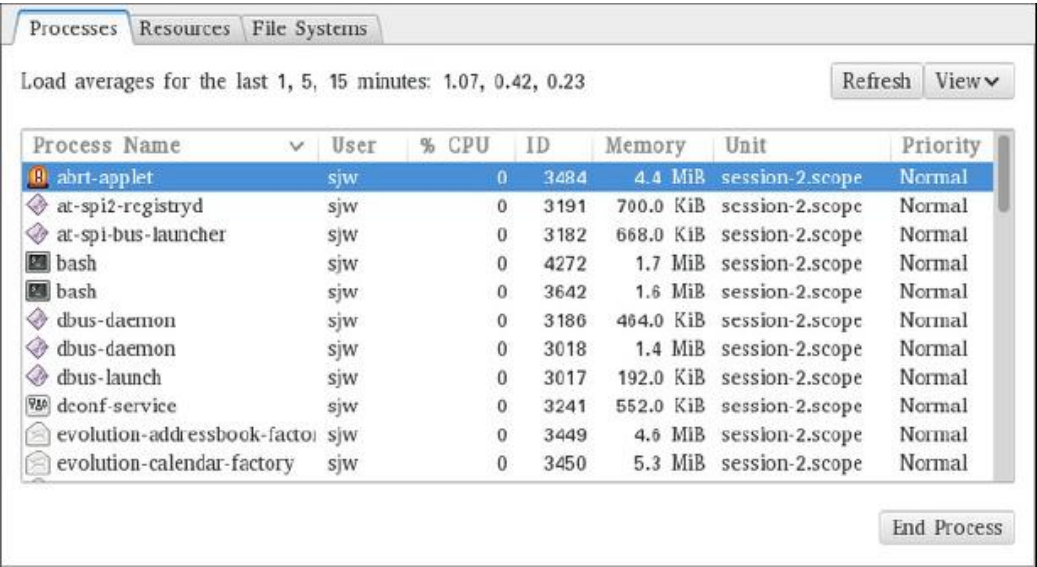
命令	描述
[Space]	立即刷新显示
[h]	显示帮助屏幕
[k]	杀死某进程。您会被提示输入进程 ID 以及要发送给它的信号
[n]	改变要显示的进程数量。您会被提示输入数量
[u]	按用户排序
[M]	按内存用量排序
[P]	按 CPU 使用率排序
[q]	退出 top

使用系统监控工具

用户可以在系统监控工具的图形界面上查看和查找进程，改变进程的优先级以及杀死进程。

在命令行启动系统监控工具需要在 shell 下输入 `gnome-system-monitor`。如果使用了 GNOME 桌面，按 Super 键进入活动概述，键入 System Monitor，然后按 Enter 键。出现系统监视器工具。Super 键存在多种伪装，这取决于键盘和其它硬件，但通常作为 Windows 或 command 键，并且通常在空格键的左侧。

点击【Processes】标签页查看运行的进程。



Process Name	User	% CPU	ID	Memory	Unit	Priority
abrt-applet	sjw	0	3484	4.4 MiB	session-2.scope	Normal
at-spi2-registryd	sjw	0	3191	700.0 KiB	session-2.scope	Normal
at-spi-bus-launcher	sjw	0	3182	668.0 KiB	session-2.scope	Normal
bash	sjw	0	4272	1.7 MiB	session-2.scope	Normal
bash	sjw	0	3642	1.6 MiB	session-2.scope	Normal
dbus-daemon	sjw	0	3186	464.0 KiB	session-2.scope	Normal
dbus-daemon	sjw	0	3018	1.4 MiB	session-2.scope	Normal
dbus-launch	sjw	0	3017	192.0 KiB	session-2.scope	Normal
deconf-service	sjw	0	3241	552.0 KiB	session-2.scope	Normal
evolution-addressbook-factory	sjw	0	3449	4.6 MiB	session-2.scope	Normal
evolution-calendar-factory	sjw	0	3450	5.3 MiB	session-2.scope	Normal

图 5-1 System Monitor — Processes

对于列出的进程，系统监控工具会显示进程的名字，状态，进程 NICE 值，进程使用内存和 CPU 的百分比，进程的 PID,进程等待的通道和进程会话的其它细节。通过点击进程名字栏可以将进程显示的信息进行升序或则降序排列。

默认系统监控工具会显示当前登录用户的进程列表，通过选择查看菜单下的不同选项，你可以做如下事情：

- 查看活动的进程
- 查看所有进程
- 查看当前登录用户的进程
- 查看进程依赖
- 另外，有两个按钮有如下作用：

- 刷新进程列表
- 选中进程后杀死进程

5.1.2 查看内存使用情况

使用 free 命令

使用 free 命令可以查看系统空闲和已经使用的内存，在 shell 下输入如下命令：

```
free
```

Free 命令可以提供物理内存和交换空间的信息。它可以显示内存总量，使用的内存大小，空闲内存大小，共享内存大小，buff 的大小和 cache 的大小以及是否可用。例子如下：

```
~]$ free
total used free shared buff/cache
available
Mem: 1016800 727300 84684 3500 204816
124068
Swap: 839676 66920 772756
```

默认 free 以字节显示大小，如果想以兆为单位可以加-m 选项，如下：

```
~]$ free -m
total used free shared buff/cache
available
Mem: 992 711 81 3 200
120
Swap: 819 65 754
```

有关可用命令行选项的完整列表，查看 free(1) man 手册。

使用系统监控工具

系统监控工具的资源标签页可以查看系统中空闲的内存大小和已经使用的大小。

在命令行启动系统监控工具可以在 shell 下输入 gnome-system-momitor。另

外,如果使用 GNOME 桌面,按【Super】键进入活动概述,键入【System Monitor】,然后按【Enter】键。出现系统监视器工具。【Super】键存在多种伪装,这取决于键盘和其它硬件,但通常作为 Windows 或 command 键,并且通常在空格键的左侧。

点击资源标签页查看系统内存使用情况。

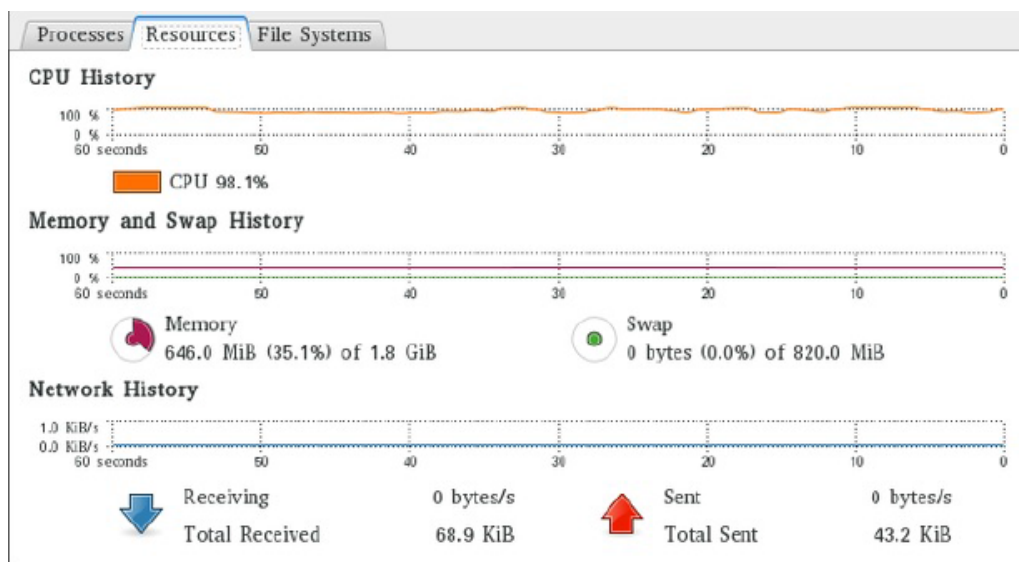


图 5-2 System Monitor — Resources

在内存和交换分区章节,系统监控工具显示内存和交换分区的历史使用图表和物理内存和交换分区的总大小以及使用率。

5.1.3 查看 CPU 使用

使用系统监控工具

使用系统监控工具的资源标签页可以查看当前系统的 CPU 使用情况。

在命令行启动系统监控工具可以在 shell 下输入 `gnome-system-momitor`。另外,如果使用 GNOME 桌面,按【Super】键进入活动概述,键入 System Monitor,然后按【Enter】键。出现系统监视器工具。【Super】键存在多种伪装,这取决于键盘和其它硬件,但通常作为 Windows 或 command 键,并且通常在空格键的左侧。

点击资源标签页查看系统的 CPU 使用情况。

在 CPU 章节,系统监控工具显示 CPU 的历史使用图表和 CPU 当前使用率的图表。

5.1.4 查看块设备和文件系统

使用 lsblk 命令

lsblk 命令可以显示系统可以使用的块设备。它比 blkid 命令提供更多的信息和输出格式控制。它从 udev 读取信息，因此非 root 用户都可以使用。显示块设备列表需要在 shell 输入如下命令：

```
lsblk
```

每一块输出的块设备，lsblk 都显示设备名，主次设备号，设备是否可以删除，设备文件大小，设备是否是只读，设备类型，设备挂载路径。例子如下：

```
~]$ lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sr0 11:0 1 1024M 0 rom
vda 252:0 0 20G 0 rom
|-vda1 252:1 0 500M 0 part /boot
`-vda2 252:2 0 19.5G 0 part
|-vg_kvm-lv_root (dm-0) 253:0 0 18G 0 lvm /
`-vg_kvm-lv_swap (dm-1) 253:1 0 1.5G 0 lvm [SWAP]
```

默认 lsblk 命令以树形结构输出块设备，如果想获取更多的设备信息添加-l 参数，例如：

```
~]$ lsblk -l

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sr0 11:0 1 1024M 0 rom
vda 252:0 0 20G 0 rom
vda1 252:1 0 500M 0 part /boot
vda2 252:2 0 19.5G 0 part
vg_kvm-lv_root (dm-0) 253:0 0 18G 0 lvm /
vg_kvm-lv_swap (dm-1) 253:1 0 1.5G 0 lvm [SWAP]
```

有关可用命令行选项的完整列表，查看 lsblk(8) man 手册。

使用 blkid 命令

blkid 命令可以显示可用块设备底层信息。它需要 root 权限，因此非 root 用户只能使用 lsblk 命令，以 root 身份在 shell 输入如下命令：

```
blkid
```

每一个列出的块设备，blkid 会显示它的可用属性，例如 UUID,文件系统类型，卷标签。例子如下：

```
~]# blkid
/dev/vda1: UUID="7fa9c421-0054-4555-b0ca-b470a97a3d84" TYPE="ext4"
/dev/vda2: UUID="7IvYzk-TnnK-oPjf-ipdD-cofz-DXaJ-gPdgBW"
TYPE="LVM2_member"
/dev/mapper/vg_kvm-lv_root:
UUID="a07b967c-71a0-4925-ab02-aebcad2ae824"
TYPE="ext4"
/dev/mapper/vg_kvm-lv_swap:
UUID="d7ef54ca-9c41-4de4-ac1b-4193b0c1ddb6"
TYPE="swap"
```

默认 blkid 命令会显示所有可用的块设备。如果想显示某一块设备，需要在命令后面加设备名：

```
blkid device_name
```

例如，想显示设备/dev/vda1 的信息，以 root 用户权限输入命令：

```
~]# blkid /dev/vd a1
/dev/vda1: UUID="7fa9c421-0054-4555-b0ca-b470a97a3d84" TYPE="ext4"
```

在上面的命令中加入 -p 和 -o 选项可以获取更多细节信息，命令同样需要 root 权限。

```
blkid -po device_name
```

例如：

```
~]# blkid -poud ev /dev/vda1

ID_FS_UUID=7fa9c421-0054-4555-b0ca-b470a97a3d84
ID_FS_UUID_ENC=7fa9c421-0054-4555-b0ca-b470a97a3d84
ID_FS_VERSION=1.0
ID_FS_TYPE=ext4
ID_FS_USAGE=filesystem
```

有关可用命令行选项的完整列表，查看 `blkid(8)` man 手册。

使用 `findmnt` 命令

`findmnt` 命令显示系统当前挂载的文件系统，在 shell 里输入如下命令：

```
findmnt
```

每一个列出的文件系统，`findmnt` 命令显示挂载点，原设备，文件系统类型和有关的挂载选项。

例如：

```
~]$ findmnt
TARGET                                SOURCE                                FSTYPE
OPTIONS
/                                     /dev/mapper/rhel-root
                                     xfs
rw,relatime,seclabel,attr2,inode64,noquota
|-/proc                               proc
rw,nosuid,nodev,noexec,relatime
| |-/proc/sys/fs/binfmt_misc          systemd-1
rw,relatime,fd=32,pgrp=1,timeout=300,minproto=5,maxproto=5,direct
|  |-/proc/fs/nfsd                    sunrpc
rw,relatime
                                     nfsd

|-/sys                                sysfs
rw,nosuid,nodev,noexec,relatime,seclabel
| |-/sys/kernel/security              securityfs
rw,nosuid,nodev,noexec,relatime
|  |-/sys/fs/cgroup                   tmpfs
rw,nosuid,nodev,noexec,seclabel,mode=755
[output truncated]
```

默认 `findmnt` 以树型结构输出文件系统，如果想获取更多的设备信息添加-l 参数，例如：

```
findmnt -l
```

例如：

```
~]$ findmnt -l

TARGET SOURCE FSTYPE OPTIONS

/proc proc proc
rw,nosuid,nodev,noexec,relatime

/sys sysfs sysfs
rw,nosuid,nodev,noexec,relatime,seclabel

/dev devtmpfs devtmpfs
rw,nosuid,seclabel,size=933372k,nr_inodes=233343,mode=755

/sys/kernel/security securityfs securityfs
rw,nosuid,nodev,noexec,relatime

/dev/shm tmpfs tmpfs
rw,nosuid,nodev,seclabel

/dev/pts devpts devpts
rw,nosuid,noexec,relatime,seclabel,gid=5,mode=620,ptmxmode=000

/run tmpfs tmpfs
rw,nosuid,nodev,seclabel,mode=755

/sys/fs/cgroup tmpfs tmpfs
rw,nosuid,nodev,noexec,seclabel,mode=755

[output truncated]
```

你可以选择只输出某种类型的文件系统，使用 `-t` 选项，后面写文件系统类型，例如：

```
findmnt -t type
```

例如，显示所有的 xfs 文件系统：

```
~]$ findmnt -t xfs

TARGET SOURCE FSTYPE OPTIONS

/dev/mapper/rhel-root xfs
rw,relatime,seclabel,attr2,inode64,noquota
```

```
└─/boot /dev/vda1 xfs  
rw,relatime,seclabel,attr2,inode64,noquota
```

有关可用命令行选项的完整列表，查看 `findmnt(8)` man 手册。

使用 df 命令

df 命令输出系统磁盘空间的使用报告，在 shell 里输入如下命令：

```
df
```

每一个列出的文件系统，df 命令都会显示文件系统的名字，大小（以 K 字节为单位），磁盘空间使用率和使用大小，磁盘空间剩余大小和文件挂载点。例子如下：

```
~]$ df  
  
Filesystem 1K-blocks Used Available Use% Mounted on  
  
/dev/mapper/vg_kvm-lv_root 18618236 4357360 13315112 25% /  
  
tmpfs 380376 288 380088 1% /dev/shm  
  
/dev/vda1 495844 77029 393215 17% /boot
```

默认 df 命令显示分区大小（以 K 字节为单位），磁盘空间使用总量，磁盘剩余空间可用量（以 K 字节为单位）。想以 M 字节和 G 字节显示需要使用 -h 选项，该选项可以以易读方式的格式显示磁盘空间大小。

例如：

```
~]$ df -h  
  
Filesystem Size Used Avail Use% Mounted on  
  
/dev/mapper/vg_kvm-lv_root 18G 4.2G 13G 25% /  
  
tmpfs 372M 288K 372M 1% /dev/shm  
  
/dev/vda1 485M 76M 384M 17% /boot
```

有关可用命令行选项的完整列表，查看 `df(1)` man 手册。

使用 du 命令

du 命令可以查看文件的大小。du 命令不加参数可以显示每个子目录中文件的大小。例如：

```
~]$ du
14972 ./Downloads
4 ./mozilla/extensions
4 ./mozilla/plugins
12 ./mozilla
15004 .
```

默认情况下，du 命令以千字节为单位显示磁盘的使用情况。如果想以易读方式（MB 和 GB）显示大小，可以添加参数-h。例如；

```
~]$ du -h
15M ./Downloads
4.0K ./mozilla/extensions
4.0K ./mozilla/plugins
12K ./mozilla
15M .
```

在列表末尾 du 命令会显示出当前目录所有文件大小的和，如果只显示目录中文件的总大小可以添加参数-s,例如：

```
~]$ du -sh
15M .
```

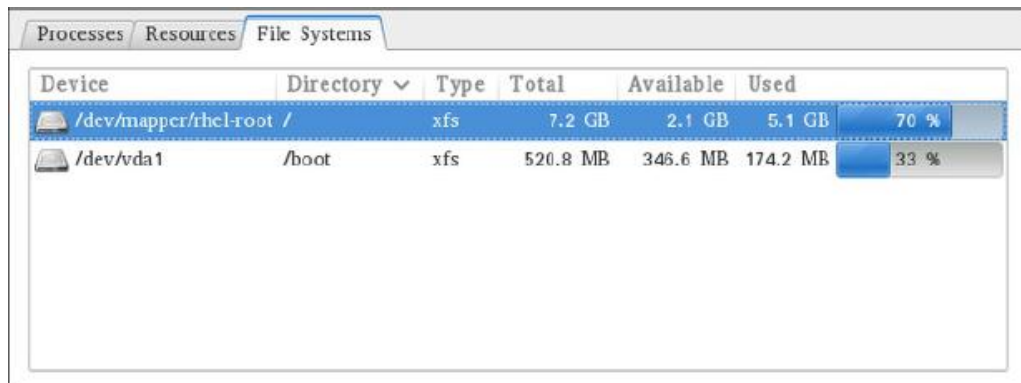
有关可用命令行选项的完整列表，查看 du(1) man 手册。

使用系统监控工具

在系统监控工具的文件系统标签页可以以图表的方式查看文件系统和磁盘空间的使用情况。

在命令行启动系统监控工具需要在 shell 下输入 gnome-system-monitor。另外，如果使用 GNOME 桌面，按【Super】键进入活动概述，键入 System Monitor，然后按【Enter】键。出现系统监视器工具。【Super】键存在多种伪装，这取决于键盘和其它硬件，但通常作为 Windows 或 command 键，并且通常在空格键的左侧。

点击文件系统标签页显示文件系统列表，如图：



Device	Directory	Type	Total	Available	Used	
/dev/mapper/rhel-root	/	xfs	7.2 GB	2.1 GB	5.1 GB	70 %
/dev/vda1	/boot	xfs	520.8 MB	346.6 MB	174.2 MB	33 %

图 5-3 System Monitor — File Systems

每一个列出的文件系统，系统监控工具都会显示原设备，目标挂载点，文件系统类型和大小，磁盘空间使用大小和未使用大小。

5.1.5 查看硬件信息

使用 `lspci` 命令

`lspci` 命令可以显示 PCI 总线上的设备信息，显示所有的 PCI 设备在 shell 里输入如下命令：

```

~]$ lspci

00:00.0 Host bridge: Intel Corporation 82X38/X48 Express DRAM
Controller

00:01.0 PCI bridge: Intel Corporation 82X38/X48 Express Host-Primary PCI
Express Bridge

00:1a.0 USB Controller: Intel Corporation 82801I (ICH9 Family) USB UHCI
Controller #4 (rev 02)

00:1a.1 USB Controller: Intel Corporation 82801I (ICH9 Family) USB UHCI
Controller #5 (rev 02)

00:1a.2 USB Controller: Intel Corporation 82801I (ICH9 Family) USB UHCI
Controller #6 (rev 02)

[output truncated]
    
```

你可以使用 `-v` 选项输出设备详细信息，使用 `-vv` 选项输出设备更详细的信息。


```
~]$ lspci -v

[output truncated]

01:00.0 VGA compatible controller: nVidia Corporation G84 [Quadro FX
370]

(rev a1) (prog-if 00 [VGA controller])

Subsystem: nVidia Corporation Device 0491

Physical Slot: 2

Flags: bus master, fast devsel, latency 0, IRQ 16

Memory at f2000000 (32-bit, non-prefetchable) [size=16M]

Memory at e0000000 (64-bit, prefetchable) [size=256M]

Memory at f0000000 (64-bit, non-prefetchable) [size=32M]

I/O ports at 1100 [size=128]

Expansion ROM at <unassigned> [disabled]

Capabilities: <access denied>

Kernel driver in use: nouveau

Kernel modules: nouveau, nvidiafb

[output truncated]
```

有关可用命令行选项的完整列表，查看 `lspci(8)` man 手册。

使用 `lsusb` 命令

`lsusb` 命令可以显示 `usb` 设备信息。显示所有的 `usb` 设备信息在 `shell` 里输入如下命令：

```
~]$ lsusb

Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub

[output truncated]

Bus 001 Device 002: ID 0bda:0151 Realtek Semiconductor Corp. Mass
Storage
```

Device (Multicard Reader)

Bus 008 Device 002: ID 03f0:2c24 Hewlett-Packard Logitech M-UAL-96
Mouse

Bus 008 Device 003: ID 04b3:3025 IBM Corp.

你可以添加-v 参数显示设备的详细信息。

```
~]$ lsusb -v
```

[output truncated]

Bus 008 Device 002: ID 03f0:2c24 Hewlett-Packard Logitech M-UAL-96
Mouse

Device Descriptor:

bLength 18

bDescriptorType 1

bcdUSB 2.00

bDeviceClass 0 (Defined at Interface level)

bDeviceSubClass 0

bDeviceProtocol 0

bMaxPacketSize0 8

idVendor 0x03f0 Hewlett-Packard

idProduct 0x2c24 Logitech M-UAL-96 Mouse

bcdDevice 31.00

iManufacturer 1

iProduct 2

iSerial 0

bNumConfigurations 1

Configuration Descriptor:

bLength 9

bDescriptorType 2

[output truncated]

有关可用命令行选项的完整列表，查看 `lsusb(8) man` 手册。

使用 `lscpu` 命令

`lscpu` 命令显示当前系统的 `cpu` 信息，信息包括 `cpu` 数目，架构，型号，家族，模式，`cpu` 高速缓存等等，在 `shell` 输入命令：

```
~]$ lscpu

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

CPU(s): 4

On-line CPU(s) list: 0-3

Thread(s) per core: 1

Core(s) per socket: 4

Socket(s): 1

NUMA node(s): 1

Vendor ID: GenuineIntel

CPU family: 6

Model: 23

Stepping: 7

CPU MHz: 1998.000

BogoMIPS: 4999.98

Virtualization: VT-x

L1d cache: 32K

L1i cache: 32K

L2 cache: 3072K

NUMA node0 CPU(s): 0-3
```

有关可用命令行选项的完整列表，查看 `lscpu(1) man` 手册。

5.1.6 检查硬件错误

NeoKylin Linux Advanced Server V7 引入了新的硬件事件报告机制。这个机制会为 DIMMs 收集内存错误和错误检查和纠错机制报告的错误，并且把错误向用户空间报告。用户空间的守护进程 `rasdaemon` 会捕获和处理这些由内核机制跟踪的有可靠性可用性可维护性的错误事件，并记录它们。这个功能以前由 `edac-utils` 提供，现在由守护进程 `rasdaemon` 提供。

安装 `rasdaemon` 需要 `root` 权限。

```
~]# yum install rasdaemon
```

启动服务命令如下

```
~]# systemctl start rasdaemon
```

输入下面的命令查看命令的各种选项

```
~]$ rasdaemon --help
Usage: rasdaemon [OPTION...] <options>
RAS daemon to log the RAS events.
□ Chapter 18 . System Monitoring Tools
317
-d, --disable disable RAS events and exit
-e, --enable enable RAS events and exit
-f, --foreground run foreground, not daemonize
-r, --record record events via sqlite3
output truncated
```

命令在 `rasdaemon(8)` man 手册里页也有描述。

`ras-mc-ctl` 工具提供了一种可以使用 EDAC 驱动的方法，输入如下命令可以查看命令的选项。

```
~]$ ras-mc-ctl --help
Usage: ras-mc-ctl [OPTIONS...]
--quiet Quiet operation.
```

```
--mainboard Print mainboard vendor and model for this hardware.  
  
--status Print status of EDAC drivers.  
  
output truncated
```

命令在 ras-mc-ctl (8) man 手册里页也有描述。

5.1.7 使用 Net-SNMP 监控性能

NeoKylin Linux Advanced Server V7 包括 Net-SNMP 软件套件，其中包括一个灵活的可扩展的简单网络管理协议（SNMP）代理。该代理及其关联的实用程序可以被用于从大量的系统中提供性能数据给一系列工具，这些工具都支持 SNMP 协议。

本节提供了关于配置 Net-SNMP 代理通过网络安全地提供性能数据，使用 SNMP 协议检索数据和扩展 SNMP 代理以提供自定义的性能指标。

安装 Net-SNMP

Net-SNMP 软件套件可作为中标麒麟高级服务器操作系统 V7 发行版的一组 RPM 软件包。表格 5-2 可用 Net-SNMP 软件包概述了每个包和它们的内容。

表格 5-2 可用 Net-SNMP 软件包

软件包	提供商
net-snmp	SNMP 代理守护进程和文档。输出性能数据需要这个包。
net-snmp-libs	netsnmp 库和捆绑的管理信息库（MIBs）。输出性能数据需要这个包。
net-snmp-utils	SNMP 客户端例如 snmpget 和 snmpwalk 。 需要该软件包以查询 SNMP 系统的性能数据。
net-snmp-perl	mib2c 程序和 NetSNMP Perl 模块。此包是由可选通道提供的。
net-snmp-python	Python 的 SNMP 客户端库。此包是

	由可选通道提供的。
--	-----------

安装任何一个软件包按如下方式使用 `yum` 命令：

```
yum install package...
```

例如，安装在本节的其余部分中使用的 **SNMP** 代理守护进程和 **SNMP** 客户端，以 `root` 身份在 `shell` 键入如下命令：

```
~]# yum install net-snmp net-snmp-libs net-snmp-utils
```

有关如何安装新的软件包在 NeoKylin Linux Advanced Server V7 的更多信息，请参见 2.2.2.4 安装软件包。

运行 Net-SNMP 守护进程

`net-snmp` 软件包中包含 **SNMP** 代理守护进程 `snmpd`。本节提供有关如何启动，停止和重新启动 `snmpd` 服务的信息。有关在中标麒麟高级服务器操作系统 V7 系统的管理服务的详细信息请参阅 3.1 使用 `systemd` 管理系统服务。

启动服务

运行 `snmpd` 服务以 `root` 身份在 `shell` 键入如下命令：

```
systemctl start snmpd.service
```

配置服务使其在系统启动后自动开始运行使用如下命令：

```
systemctl enable snmpd.service
```

停止服务

停止 `snmpd` 服务以 `root` 身份在 `shell` 键入如下命令：

```
systemctl stop snmpd.service
```

配置服务使其在系统启动后并不开始运行，使用如下命令：

```
systemctl disable snmpd.service
```

重启服务

重启 `snmpd` 服务以 `root` 身份在 `shell` 键入如下命令：

```
systemctl restart snmpd.service
```

该命令停止服务并快速重启服务。要仅重新加载配置，而无需停止服务，运行以下命令：

```
systemctl reload snmpd.service
```

这会运行 snmp 服务去重新加载配置。

配置 Net-SNMP

要修改 Net-SNMP 代理守护进程的配置，编辑/etc/snmp/snmpd.conf 配置文件。中标麒麟高级服务器操作系统 V7 包含的默认的 snmpd.conf 文件被大量注释，并可以作为一个很好的代理配置文件的起点。

本节主要介绍两种常见的任务：设置系统信息并配置认证。有关可用的配置指令的详细信息，请参阅 snmpd.conf(5)手册页。此外，在 net-snmp 软件包里有一个名为 snmpd.conf 的实用程序，可以用来以交互方式生成代理配置。

需要注意的是 net-snmp-util 软件包必须被安装才能使用本节所述的 snmpwalk 实用程序。

```
systemctl reload snmpd.service
```

设置系统信息

Net-SNMP 通过系统树提供了一些基本的系统信息。例如，下面的 snmpwalk 的命令显示具有默认代理配置系统树。

```
~]# snmpwalk -v2c -c public localhost system

SNMPv2-MIB::sysDescr.0 = STRING: Linux localhost.localdomain 3.10.0-
123.el7.x86_64 #1 SMP Mon May 5 11:16:57 EDT 2014 x86_64

SNMPv2-MIB::sysObjectID.0 =                                OID:
NET-SNMP-MIB::netSnmpAgentOIDs.10

DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (464) 0:00:04.64

SNMPv2-MIB::sysContact.0 = STRING: Root <root@ localhost> (configure
/etc/snmp/snmp.local.conf)
```

```
[output truncated]
```

缺省情况下，sysName 对象被设置为主机名。sysLocation 和 sysContact 对象可以在/etc/snmp/snmpd.conf 文件通过改变 syslocation 和 syscontact 的值被配置。例如：

```
syslocation Datacenter, Row 4, Rack 3  
syscontact UNIX Admin <admin@ example.com>
```

更改配置文件后，重新加载配置，并通过再次运行 snmpwalk 命令测试一下配置是否生效。

```
~]# systemctl reload snmp.service  
~]# snmpwalk -v2c -c public localhost system  
SNMPv2-MIB::sysDescr.0 = STRING: Linux localhost.localdomain 3.10.0-  
123.el7.x86_64 #1 SMP Mon May 5 11:16:57 EDT 2014 x86_64  
SNMPv2-MIB::sysObjectID.0 = OID:  
NET-SNMP-MIB::netSnmpAgentOIDs.10  
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (35424)  
0:05:54.24  
SNMPv2-MIB::sysContact.0 = STRING: UNIX Admin <admin@ example.  
com>  
SNMPv2-MIB::sysName.0 = STRING: localhost.localdomain  
SNMPv2-MIB::sysLocation.0 = STRING: Datacenter, Row 4 , Rack 3  
[output truncated]
```

配置权限

Net-SNMP 代理守护程序支持所有三个版本的 SNMP 协议。前两个版本（1 和 2c）通过使用字符串提供简单的身份验证。该字符串是代理人 and 任何客户端实用程序之间共享的秘密。该字符串在网络上用明文传递，这认为是不安全的。SNMP 协议第 3 版支持使用多种协议的用户认证和信息加密。Net-SNMP 代理还支持 SSH 通道，使用 X.509 证书的 TLS 认证和 Kerberos 认证。

配置 SNMP 版本 2c

要配置 SNMP 版本 2c 的社区，在/etc/snmp/snmpd.conf 配置文件下使用 RO 社区或 RW 社区指令。指令的格式是如下：

```
directive community [source [OID]]
```

其中，community 要使用社区字符串，source 是 IP 地址或子网，OID 对 SNMP 树提供访问属性。例如，下面的指令对客户端提供只读的系统树访问，该客户端使用本地计算机上的社区字符串“redhat”。

```
rocommunity redhat 127.0.0.1 .1.3.6.1.2.1.1
```

使用 snmpwalk 命令并添加-v 和-c 参数可以测试配置。

```
~]# snmpwalk -v2c -c redhat localhost system
SNMPv2-MIB::sysDescr.0 = STRING: Linux localhost.localdomain 3.10.0-
123.el7.x86_64 #1 SMP Mon May 5 11:16:57 EDT 2014 x86_64
SNMPv2-MIB::sysObjectID.0 =                               OID:
NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (101376)
0:16:53.76
SNMPv2-MIB::sysContact.0 = STRING: UNIX Admin <admin@
example.com>
SNMPv2-MIB::sysName.0 = STRING: localhost.localdomain
SNMPv2-MIB::sysLocation.0 = STRING: Datacenter, Row 4, Rack 3
[output truncated]
```

配置 SNMP 版本 3

要配置 SNMP 版本 3，使用 net-snmp-create-v3-user 命令。此命令添加条目到/var/lib/net-snmp/snmpd.conf 和/etc/snmp/snmpd.conf 文件，这两个文件可以创建用户并授权访问。注意，net-snmp-create-v3-user 命令只有代理没有运行时才能运行。下面的示例创建了“admin”用户，密码“redhatsnmp”：

```
~]# systemctl stop snmpd.service

~]# net-snmp-create-v3-user

Enter a SNMPv3 user name to create:

admin

Enter authentication pass-phrase:

redhatsnmp

Enter encryption pass-phrase:

[press return to reuse the authentication pass-phrase]


adding the following line to /var/lib/net-snmp/snmpd.conf:

createUser admin MD5 "redhatsnmp" DES

adding the following line to /etc/snmp/snmpd.conf:

rwuser admin

~]# systemctl start snmpd.service
```

`net-snmp-create-v3-user` 命令添加 `rwuser` 指令（或当提供的 `-ro` 命令行选项使用 `rouser`）到 `/etc/snmp/snmpd.conf` 中和添加 `rwcommunity` 和 `rocommunity` 的格式相同：

```
directive user [no auth|auth|priv] [OID]
```

其中，`user` 是用户名，`OID` 是 SNMP 树提供访问。默认情况下，Net-SNMP 代理仅允许已通过授权的请求（`auth` 选项）。`noauth` 选项可以允许未经授权的请求，`priv` 选项强制使用加密，`authpriv` 选项指定的请求必须通过授权和回复被加密。

例如，下面一行表示授予用户 “admin” 读写访问整个树：

```
rwuser admin authpriv .1
```

为了测试配置，在你的用户目录下创建一个 `snmp` 目录，在该目录下创建 `snmp.conf` 文件（`~/snmp/snmp.conf`），使用如下命令：

```
defVersion 3

defSecurityLevel authPriv
```

```
defSecurityName admin
defPassphrase redhatsnmp
```

snmpwalk 命令在查询代理时会使用这些授权设置。

```
~]$ snmpwalk -v3 localhost system
SNMPv2-MIB::sysDescr.0 = STRING: Linux localhost.localdomain 3.10.0-
123.el7.x86_64 #1 SMP Mon May 5 11:16:57 EDT 2014 x86_64
[output truncated]
```

检索数据性能

中标麒麟高级服务器操作系统 V7 的 Net-SNMP 代理在 SNMP 协议上提供了多种性能信息。此外，可以通过代理查询系统中安装 RPM 包的列表，系统当前运行的进程列表和系统的网络配置。

本节提供了关于 OIDs 在 SNMP 上的性能概述。它假定系统已安装 net-snmp-utils 软件包，并且用户被授予访问 SNMP 树，如 0 配置权限中描述的。

硬件配置

包括 Net-SNMP 的 Host Resources MIB 提供了主机客户端程序的硬件和软件配置。表格 5-3 可用的 OIDs 总结了在 MIB 下可以获取的不同的 OID 。

表格 5-3 可用的 OIDs

OID	描述
HOST-RESOURCES-MIB::hrSystem	包含一般系统信息，如运行时间，用户的数目，和运行的进程的数目。
HOST-RESOURCES-MIB::hrStorage	包含内存和文件系统使用情况数据。
HOST-RESOURCES-MIB::hrDevice	包含所有的处理器，网络设备和文件系统的列表。
HOST-RESOURCES-MIB::hrSWRun	包含所有正在运行的进程的列表。
HOST-RESOURCES-MIB::	包含进程表中的内存和 CPU 统计

hrSWRunPerf	信息，进程表来自于 HOST-RESOURCES-MIB::hrSWRun。
HOST-RESOURCES-MIB::hrSWIns talled	包含 RPM 数据库的列表。

在 Host Resources MIB 中，还有一些可用于检索的可用信息的 SNMP 表。

下面的例子列出了 HOST-RESOURCES-MIB:: hrFSTable:

```
~]$ snmptable -Cb localhost HOST-RESOURCES-MIB::hrFSTable
SNMP table: HOST-RESOURCES-MIB::hrFSTable

Index MountPoint RemoteMountPoint                               Type
Access Bootable StorageIndex LastFullBackupDate LastPartialBackupDate
1      "/"                               ""      HOST-RESOURCES-TYPES::hrFSLinuxExt2
readWrite      true                    31      0-1-1,0:0:0.0      0-1-1,0:0:0.0
5 "/dev/shm"                               ""      HOST-RESOURCES-TYPES::hrFSOther
readWrite      false                   35      0-1-1,0:0:0.0      0-1-1,0:0:0.0
6      "/boot"                             ""      HOST-RESOURCES-TYPES::hrFSLinuxExt2
readWrite      false                   36      0-1-1,0:0:0.0      0-1-1,0:0:0.0
```

关于 HOST -RESOURCES-MIB 的更多信息，可以查看/usr/share/snmp/mibs/HOST-RESOURCES-MIB.txt 文件。

CPU 和内存信息

UCD-SNMP-MIB 的大多数系统性能数据是可用的。systemStats OID 提供了一些处理器的使用计数器：

```
~]$ snmpwalk localhost UCD -SNMP -MIB: : systemStats

UCD-SNMP-MIB::ssIndex.0 = INTEGER: 1
UCD-SNMP-MIB::ssErrorName.0 = STRING: systemStats
UCD-SNMP-MIB::ssSwapIn.0 = INTEGER: 0 kB
UCD-SNMP-MIB::ssSwapOut.0 = INTEGER: 0 kB
UCD-SNMP-MIB::ssIOSent.0 = INTEGER: 0 blocks/s
UCD-SNMP-MIB::ssIOReceive.0 = INTEGER: 0 blocks/s
UCD-SNMP-MIB::ssSysInterrupts.0 = INTEGER: 29 interrupts/s
UCD-SNMP-MIB::ssSysContext.0 = INTEGER: 18 switches/s
UCD-SNMP-MIB::ssCpuUser.0 = INTEGER: 0
UCD-SNMP-MIB::ssCpuSystem.0 = INTEGER: 0
UCD-SNMP-MIB::ssCpuIdle.0 = INTEGER: 99
```

```
UCD-SNMP-MIB::ssCpuRawUser.0 = Counter32: 2278
UCD-SNMP-MIB::ssCpuRawNice.0 = Counter32: 1395
UCD-SNMP-MIB::ssCpuRawSystem.0 = Counter32: 6826
UCD-SNMP-MIB::ssCpuRawIdle.0 = Counter32: 3383736
UCD-SNMP-MIB::ssCpuRawWait.0 = Counter32: 7629
UCD-SNMP-MIB::ssCpuRawKernel.0 = Counter32: 0
UCD-SNMP-MIB::ssCpuRawInterrupt.0 = Counter32: 434
UCD-SNMP-MIB::ssIORawSent.0 = Counter32: 266770
UCD-SNMP-MIB::ssIORawReceived.0 = Counter32: 427302
UCD-SNMP-MIB::ssRawInterrupts.0 = Counter32: 743442
UCD-SNMP-MIB::ssRawContexts.0 = Counter32: 718557
UCD-SNMP-MIB::ssCpuRawSoftIRQ.0 = Counter32: 128
UCD-SNMP-MIB::ssRawSwapIn.0 = Counter32: 0
UCD-SNMP-MIB::ssRawSwapOut.0 = Counter32: 0
```

特别是， ssCpuRawUser, ssCpuRawSystem, ssCpuRawWait 和 ssCpuRawIdle
OIDs 提供的计数器在确定系统是否要花费了大量的处理器时间在内核空间或用
户空间或 I/O 时非常有用。 ssRawSwapIn 和 ssRawSwapOut 可确定系统内存是否
耗尽时非常有用。

UCD-SNMP-MIB:: memory OID 下很多可用的内存信息和 free 命令类似：

```
~]$ snmpwalk localhost UCD -SNMP -MIB: : memory
UCD-SNMP-MIB::memIndex.0 = INTEGER: 0
UCD-SNMP-MIB::memErrorName.0 = STRING: swap
UCD-SNMP-MIB::memTotalSwap.0 = INTEGER: 1023992 kB
UCD-SNMP-MIB::memAvailSwap.0 = INTEGER: 1023992 kB
UCD-SNMP-MIB::memTotalReal.0 = INTEGER: 1021588 kB
UCD-SNMP-MIB::memAvailReal.0 = INTEGER: 634260 kB
UCD-SNMP-MIB::memTotalFree.0 = INTEGER: 1658252 kB
UCD-SNMP-MIB::memMinimumSwap.0 = INTEGER: 16000 kB
```

```
UCD-SNMP-MIB::memBuffer.0 = INTEGER: 30760 kB
UCD-SNMP-MIB::memCached.0 = INTEGER: 216200 kB
UCD-SNMP-MIB::memSwapError.0 = INTEGER: noError(0)
UCD-SNMP-MIB::memSwapErrorMsg.0 = STRING:
```

UCD-SNMP-MIB 的负载均衡是可用的。SNMP 的 UCD-SNMP-MIB::laTable 会列出 1 分钟，5 分钟和 15 分钟的负载均衡值。

```
~]$ snmptable localhost UCD-SNMP-MIB::laTable
SNMP table: UCD-SNMP-MIB::laTable

laIndex laNames laLoad laConfig laLoadInt laLoadFloat laErrorFlag
laErrorMessage
    1 Load-1 0.00 12.00 0 0.000000 noError
    2 Load-5 0.00 12.00 0 0.000000 noError
    3 Load-15 0.00 12.00 0 0.000000 noError
```

文件系统和磁盘信息

Host Resources MIB 提供文件系统的大小和使用信息。每个文件系统（以及各内存池）在 HOST-RESOURCES-MIB::hrStorageTable 表中有一个入口：

```
~]$ snmptable -Cb localhost HOST-RESOURCES-MIB::hrStorageTable
SNMP table: HOST-RESOURCES-MIB::hrStorageTable

Index      Type      Descr
AllocationUnits Size Used AllocationFailures
1          HOST-RESOURCES-TYPES::hrStorageRam Physical memory
1024 Bytes 1021588 388064 ?
3          HOST-RESOURCES-TYPES::hrStorageVirtualMemory Virtual memory
1024 Bytes 2045580 388064 ?
6          HOST-RESOURCES-TYPES::hrStorageOther Memory buffers
1024 Bytes 1021588 31048 ?
7          HOST-RESOURCES-TYPES::hrStorageOther Cached memory
1024 Bytes 216604 216604 ?
10         HOST-RESOURCES-TYPES::hrStorageVirtualMemory Swap space
1024 Bytes 1023992 0 ?
31         HOST-RESOURCES-TYPES::hrStorageFixedDisk /
4096 Bytes 2277614 250391 ?
35         HOST-RESOURCES-TYPES::hrStorageFixedDisk /dev/shm
4096 Bytes 127698 0 ?
36         HOST-RESOURCES-TYPES::hrStorageFixedDisk /boot
1024 Bytes 198337 26694 ?
```

HOST-RESOURCES-MIB::hrStorageSize 和 HOST-RESOURCES-MIB::hrStorageUsed 中的 ODI 被用来计算挂载文件系统的剩余容量。

I/O 数据在 UCD-SNMP-MIB::systemStats 和 UCD-DISKIO-MIB::diskIOTable 表中都是可用的。后者提供了更详细的数据，后者的 OID 有 diskIONReadX 和 diskIONWrittenX 其提供计数器，用于在系统启动时记录读取和写入块设备的字节数。

```
~]$ snmptable -Cb localhost UCD-DISKIO-MIB::diskIOTable
SNMP table: UCD-DISKIO-MIB::diskIOTable

  Index Device      NRead  NWritten Reads Writes LA1 LA5 LA15  NReadX
  NWrittenX
...
    25   sda 216886272 139109376 16409   4894   ?   ?   ? 216886272
139109376
    26   sda1 2455552    5120    613     2   ?   ?   ? 2455552
5120
    27   sda2 1486848      0    332     0   ?   ?   ? 1486848
0
    28   sda3 212321280 139104256 15312   4871   ?   ?   ? 212321280
139104256
```

网络信息

Interfaces MIB 提供网络设备信息。IF-MIB::ifTable 提供了一个 SNMP 表，系统上每个接口及其配置和各种数据包计数器在这个表里都有一个条目。下面的例子显示 ifTable 的部分列，它显示了系统上的两个物理网络接口：

```
~]$ snmptable -Cb localhost IF-MIB::ifTable
SNMP table: IF-MIB::ifTable

Index Descr Type Mtu Speed PhysAddress AdminStatus
1 lo softwareLoopback 16436 10000000 up
2 eth0 ethernetCsmacd 1500 0 52:54:0:c7:69:58
up
3 eth1 ethernetCsmacd 1500 0 52:54:0:a7:a3:24 down
```

网络流量是根据 IF-MIB::ifOutOctets 和 IF-MIB::ifInOctets 提供。下面 SNMP 查询将检索此系统上的每一个接口的网络流量。

```
~]$ snmpwalk localhost IF-MIB::ifDescr
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: eth0
IF-MIB::ifDescr.3 = STRING: eth1

~]$ snmpwalk localhost IF-MIB::ifOutOctets
IF-MIB::ifOutOctets.1 = Counter32: 10060699
IF-MIB::ifOutOctets.2 = Counter32: 650
IF-MIB::ifOutOctets.3 = Counter32: 0

~]$ snmpwalk localhost IF-MIB::ifInOctets
```



```
IF-MIB::ifInOctets.1 = Counter32: 10060699
```

```
IF-MIB::ifInOctets.2 = Counter32: 78650
```

```
IF-MIB::ifInOctets.3 = Counter32: 0
```

扩展的 Net-SNMP

Net-SNMP 代理可以扩展到提供除原生系统度量的应用度量。这使得容量规划和性能问题的故障排除。例如，在测试中知道邮件系统每 15 分钟有 5 分钟处在负载均衡状态是有帮助的，但更有用的是知道邮件系统在每秒处理 8000 个消息时它 15 分钟负载均衡是多少。当应用程序负载的度量可通过相同的接口作为系统的度量，这也允许在不同的负载影响的情况下对系统性能可视化（例如，每增加 10,000 条信息会增加平均负载线至 100,000）。

包括中标麒麟高级服务器操作系统 V7 的一些应用程序通过 SNMP 扩展 Net-SNMP 代理以提供应用度量。有几种方法来扩展代理定制应用程序。本节介绍通过 shell 脚本和可选的 Perl 插件来扩展代理。假设系统中 net-snmp-utils 和 net-snmp-perl 包已经安装，同时用户被授权访问 SNMP 树在 0 配置权限。

使用 shell 脚本扩展 Net-SNMP

在 Net-SNMP 代理提供了一个扩展 MIB (NET-SNMP-EXTEND-MIB)，可以用来查询任意的 shell 脚本。要指定的 shell 脚本来运行，可以在 /etc/snmp/snmpd.conf 文件使用 extend 指令。一旦定义，代理将提供退出码和在 SNMP 上命令的任何输出。下面的例子演示了这种机制，该脚本确定在进程表中 httpd 进程的数量。

注意：

Net-SNMP 代理提供了一个内建的机制通过 proc 命令检查进程表。更多的信息可以查看 snmpd.conf(5)手册页。

下面 shell 脚本的退出码是在给定时间点的系统上运行的 httpd 进程数。

```
#!/bin/sh

NUMPIDS=`pgrep httpd | wc -l`

exit $NUMPIDS
```

为了使这个脚本在 SNMP 上可用，将脚本复制到系统路径上的某个位置，

设置可执行位，并添加 `extend` 指令到 `/etc/snmp/snmp.conf` 文件。添加 `extend` 指令的格式如下：

```
extend name prog args
```

`name` 是标识 `extend` 的字符串，`prog` 是要运行的程序，`args` 是传入程序的参数。例如，如果上面的 `shell` 脚本复制到 `/usr/local/bin/check_apache.sh`，下面的指令会添加脚本到 `SNMP` 树。

```
extend httpd_pids /bin/sh /usr/local/bin/check_apache.sh
```

脚本可以在 `NET-SNMP-EXTEND-MIB::nsExtendObjects` 中查询到。

```
~]$ snmpwalk localhost NET-SNMP-EXTEND-MIB::nsExtendObjects
NET-SNMP-EXTEND-MIB::nsExtendNumEntries.0 = INTEGER: 1
NET-SNMP-EXTEND-MIB::nsExtendCommand."httpd_pids" = STRING: /bin/sh
NET-SNMP-EXTEND-MIB::nsExtendArgs."httpd_pids" = STRING:
/usr/local/bin/check_apache.sh
NET-SNMP-EXTEND-MIB::nsExtendInput."httpd_pids" = STRING:
NET-SNMP-EXTEND-MIB::nsExtendCacheTime."httpd_pids" = INTEGER: 5
NET-SNMP-EXTEND-MIB::nsExtendExecType."httpd_pids" = INTEGER: exec(1)
NET-SNMP-EXTEND-MIB::nsExtendRunType."httpd_pids" = INTEGER: run-on-
read(1)
NET-SNMP-EXTEND-MIB::nsExtendStorage."httpd_pids" = INTEGER:
permanent(4)
NET-SNMP-EXTEND-MIB::nsExtendStatus."httpd_pids" = INTEGER: active(1)
NET-SNMP-EXTEND-MIB::nsExtendOutput1Line."httpd_pids" = STRING:
NET-SNMP-EXTEND-MIB::nsExtendOutputFull."httpd_pids" = STRING:
NET-SNMP-EXTEND-MIB::nsExtendOutNumLines."httpd_pids" = INTEGER: 1
NET-SNMP-EXTEND-MIB::nsExtendResult."httpd_pids" = INTEGER: 8
NET-SNMP-EXTEND-MIB::nsExtendOutLine."httpd_pids".1 = STRING:
```

请注意，退出码设置为整型（示例中退出码为 8），其它任何输出为字符串类型。为了显示整数的多重度量，`extend` 指令提供了不同的参数。例如，下面的脚本可以被用于确定匹配任意字符串的进程数，并且也将输出一个文本字符串来表示进程数：

```
#!/bin/sh

PATTERN=$1

NUMPIDS=`pgrep $PATTERN | wc -l`

echo "There are $NUMPIDS $PATTERN processes."

exit $NUMPIDS
```

当上面的脚本被拷贝到 `/usr/local/bin/check_proc.sh` 下，执行下面 `/etc/snmp/snmpd.conf` 文件中的指令会显示出 `httpd` 和 `snmpd` 的进程数。

```
extend httpd_pids /bin/sh /usr/local/bin/check_proc.sh httpd

extend snmpd_pids /bin/sh /usr/local/bin/check_proc.sh snmpd
```

下面的例子显示 snmpwalk 命令对 nsExtend Objects OID 的输出：

```
~]$ snmpwalk localhost NET-SNMP-EXTEND-MIB::nsExtendObjects
NET-SNMP-EXTEND-MIB::nsExtendNumEntries.0 = INTEGER: 2
NET-SNMP-EXTEND-MIB::nsExtendCommand."httpd_pids" = STRING: /bin/sh
NET-SNMP-EXTEND-MIB::nsExtendCommand."snmpd_pids" = STRING: /bin/sh
NET-SNMP-EXTEND-MIB::nsExtendArgs."httpd_pids" = STRING:
/usr/local/bin/check_proc.sh httpd
NET-SNMP-EXTEND-MIB::nsExtendArgs."snmpd_pids" = STRING:
/usr/local/bin/check_proc.sh snmpd
NET-SNMP-EXTEND-MIB::nsExtendInput."httpd_pids" = STRING:
NET-SNMP-EXTEND-MIB::nsExtendInput."snmpd_pids" = STRING:
...
NET-SNMP-EXTEND-MIB::nsExtendResult."httpd_pids" = INTEGER: 8
NET-SNMP-EXTEND-MIB::nsExtendResult."snmpd_pids" = INTEGER: 1
NET-SNMP-EXTEND-MIB::nsExtendOutLine."httpd_pids".1 = STRING: There are 8
httpd processes.
NET-SNMP-EXTEND-MIB::nsExtendOutLine."snmpd_pids".1 = STRING: There are 1
snmpd processes.
```

警告：

整数退出码被限制在 0-255 的范围。对于超过 256 的值，既可以使用脚本的标准输出（被认为是字符串）或扩展代理的不同方法。

这最后一个实例列出了系统空闲内存的大小和 httpd 进程的数量的查询结果。这个查询可以被用到性能测中来检查进程数目的变化对内存的影响：

```
~]$ snmpget localhost \
' NET -SNMP -EXT END -MIB: : nsExtend Result. "httpd _pids" \
UC D -SNMP -MIB: : memAvai l Real . 0
NET-SNMP-EXTEND-MIB::nsExtendResult."httpd_pids" = INTEGER: 8
UCD-SNMP-MIB::memAvailReal.0 = INTEGER: 799664 kB
```

使用 perl 脚本扩展 Net-SNMP

使用 extend 指令执行 shell 脚本用来在 SNMP 上自定义应用度量是一种受限的方法。Net-SNMP 代理提供了一个内嵌的 Perl 接口用于自定义对象。net-snmp-perl 软件包中设置可选项可以提供 perl 模块，该模块可以用来在 NeoKylin Linux Advanced Server V7 上写嵌入式的 perl 插件。

注意：

之前订阅的可选和补充渠道看“Scope of Coverage Details”。如果你决定从这些通道安装软件包，请到红帽客户门户网站按照文档 How to access Optional

and Supplementary channels, and -devel packages using Red Hat Subscription Manager (RHSM)?记录的步骤操作。

NetSNMP::agent 的 Perl 模块提供一个 agent 对象，该对象用于处理代理的 OID 树的一部分请求。agent 对象的构造函数可以运行并作为 snmpd 的子代理或独立的代理。需要创建一个嵌入式代理并不需要参数：

```
use NetSNMP::agent ('all');

my $agent = new NetSNMP::agent();
```

Agent 对象有一个 register 方法用来向某一个特殊的 OID 注册一个回调函数。该 register 方法需要一个名称，一个 OID 还有指向回调函数的指针。下面的例子会注册一个回调函数，函数名字叫 hello_handler，它会处理 OID . 1 . 3 . 6 . 1 . 4 . 1 . 8072 . 9 9 9 9 . 9 9 9 9 的请求。

```
$agent->register("hello_world", ".1.3.6.1.4.1.8072.9999.9999",
    \& hello_handler);
```

注意：

OID.1.3.6.1.4.1.8072.9999.9999 仅用于演示目的。如果您的组织还没有一个根 OID，您可以通过联系 ISO 名称注册机构（美国的 ANSI）得到。

该处理函数被调用时使用四个参数（HANDLER, REGISTRATION_INFO, REQUEST_INFO, 和 REQUESTS.）。REQUESTS 参数在被调用时包含一个请求的列表,同时列表会被初始化和迭代。列表中的 request 对象通过获取和设置方法，操作 OID 的值。例如，下面的调用将设置 request 对象的字符串值为“hello world”：

```
$request->setValue(ASN_OCTET_STR, "hello world");
```

处理函数应响应两种类型的 SNMP 请求：GET 请求和 GETNEXT 请求。请求的类型是通过调用 request_info 对象中的 getMode 方法，该方法作为处理函数的第三个参数被传递。如果请求是 GET 请求，调用者期望处理程序来设置 request 对象的值，这取决于 OID 的请求。如果请求是一个请求 GETNEXT，调用者也将期望处理程序设置请求到 OID 树下一个可用的 OID。下面是实例代码：

```
my $request;
my $string_value = "hello world";
my $integer_value = "8675309";

for($request = $requests; $request; $request = $request->next()) {
    my $oid = $request->getOID();
    if ($request_info->getMode() == MODE_GET) {
        if ($oid == new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.0")) {
            $request->setValue(ASN_OCTET_STR, $string_value);
        }
        elsif ($oid == new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.1")) {
            $request->setValue(ASN_INTEGER, $integer_value);
        }
    } elsif ($request_info->getMode() == MODE_GETNEXT) {
        if ($oid == new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.0")) {
            $request->setOID(".1.3.6.1.4.1.8072.9999.9999.1.1");
            $request->setValue(ASN_INTEGER, $integer_value);
        }
        elsif ($oid < new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.0")) {
            $request->setOID(".1.3.6.1.4.1.8072.9999.9999.1.0");
            $request->setValue(ASN_OCTET_STR, $string_value);
        }
    }
}
```

GetMode 返回 MODE_GET, 处理程序分析 GetOID 的值, 调用 request 对象。根据 OID 以 1.0 结尾还是以 1.1 结尾, 设置 string_value 或者 integer_value。如果 GetMode 返回 MODE_GETNEXT, 处理程序会判断 OID 名是否为 “.1.0”, 然后设置 OID 和值 “.1.1”。如果请求 OID 值小于 “.1.0”, 设置 OID 值为 “.1.0”。这实际上是返回树中的 “下一个” 的值, 所以像 snmpwalk 这样的程序可以遍历树。

变量的类型包含在 NetSNMP::ASN, 看 perldoc 中关于 NetSNMP::ASN 可设置类型的列表。

完整的 Perl 插件代码如下:

```
#!/usr/bin/perl

use NetSNMP::agent(':all');
use NetSNMP::ASN qw(ASN_OCTET_STR ASN_INTEGER);

sub hello_handler {
    my ($handler, $registration_info, $request_info, $requests) = @_;
    my $request;
    my $string_value = "hello world";
    my $integer_value = "8675309";

    for($request = $requests; $request; $request = $request->next()) {
        my $oid = $request->getOID();
        if ($request_info->getMode() == MODE_GET) {
            if ($oid == new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.0")) {
                $request->setValue(ASN_OCTET_STR, $string_value);
            }
        }
    }
}
```

```

    }
    elsif ($oid == new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.1"))
    {
        $request->setValue(ASN_INTEGER, $integer_value);
    }
} elsif ($request_info->getMode() == MODE_GETNEXT) {
    if ($oid == new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.0")) {
        $request->setOID(".1.3.6.1.4.1.8072.9999.9999.1.1");
        $request->setValue(ASN_INTEGER, $integer_value);
    }
    elsif ($oid < new NetSNMP::OID(".1.3.6.1.4.1.8072.9999.9999.1.0"))
    {
        $request->setOID(".1.3.6.1.4.1.8072.9999.9999.1.0");
        $request->setValue(ASN_OCTET_STR, $string_value);
    }
}
}
}

my $agent = new NetSNMP::agent();
$agent->register("hello_world", ".1.3.6.1.4.1.8072.9999.9999",
    \&hello_handler);

```

为了测试插件，拷贝上面的代码到/usr/share/snmp/hello_world.pl，然后把下面这行添加到/etc/snmp/snmpd.conf 文件

```
perl do "/usr/share/snmp/hello_world.pl"
```

SNMP 代理守护进程需要重启来使插件生效，重启后，snmpwalk 会返回新数据。

```

~]$ snmpwalk localhost NET -SNMP -MIB: : netSnmPlaypen
NET-SNMP-MIB::netSnmPlaypen.1.0 = STRING: "hello world"
NET-SNMP-MIB::netSnmPlaypen.1.1 = INTEGER: 8675309

```

snmpget 应该处理程序的其他模式。

```

~]$ snmpget localhost \
NET -SNMP -MIB: : netSnmPlaypen. 1. 0 \
NET -SNMP -MIB: : netSnmPlaypen. 1. 1
NET-SNMP-MIB::netSnmPlaypen.1.0 = STRING: "hello world"
NET-SNMP-MIB::netSnmPlaypen.1.1 = INTEGER: 8675309

```

5.2 OpenLMI

开放式 linux 管理设备，通常被简称为 OpenLMI，它是管理 linux 系统的通

用基础设备。它构建于现有工具基础之上，并为了隐藏来自系统管理员的底层系统大量复杂性而作为抽象层运转。OpenLMI 和一系列可本地或远程登陆的服务共同发行，提供多语言绑定、标准 APIs 和可用于管理监测硬件、操作系统、系统服务的标准脚本。

5.2.1 关于 OpenLMI

OpenLMI 旨在对在物理机和虚拟机上运行中标麒麟高级服务器操作系统 V7 系统的产品服务器提供一个通用管理界面，它由以下三个部分组成：

- 1) 系统管理代理——这些代理都安装了管理系统，实现一个呈现给标准对象中间商的对象模型。在 OpenLMI 实现的最初的代理包括存储配置和网络配置，但后来的工作会处理系统管理的附加元素。系统管理代理通常被称为通用信息模型提供商或者 CIM 提供者。
- 2) 一个标准对象中间商——对象中间商管理代理并给他们提供接口。标准对象中间商也被称为 CIM 对象监控或 CIMOM。
- 3) 客户端应用程序和脚本——客户端应用程序和脚本通过标准对象中间商调用系统管理代理。

OpenLMI 项目通过提供一个底层接口实施现有管理措施，这些底层接口可以被脚本或系统管理控制台使用。和 OpenLMI 一起发行的接口包括 C, C ++, Python, Java 和一个交互式命令行客户端，他们都提供相同的完全访问每个代理实现的功能。这可以确保您始终能够访问完全相同的功能，无论您决定使用哪个编程接口。

主要特性

下面是在你的系统上安装和使用 OpenLMI 的主要优势：

- OpenLMI 提供一个标准接口来配置，管理和监控本地和远程系统。
- 它允许你配置，管理和监控在物理机和虚拟机上运行的产品服务器。
- 和它共同发行的一系列 CIM 提供者可以使你配置，管理和监控

存储设备和进行复杂的网络工作。

- 它可以让你从 C, C ++, Python 和 Java 程序调用系统管理功能，包括 LMIShell，这提供了一个命令行界面。
- 它是一个基于开放的行业标准的免费软件。
-

管理能力

OpenLMI 的主要功能包括存储设备，网络，系统服务的管理，用户帐户，硬件和软件配置，电源管理和交互使用动态目录。随中标麒麟高级服务器操作系统 V7 绑定发行的 CIM 提供商的完整列表，请参阅表格 5-4 可用 CIM 提供商。

表格 5-4 可用 CIM 提供商

包名	描述
openlmi-account	用于管理用户帐户的 CIM 提供商。
openlmi-logicalfile	用于读取文件和目录的 CIM 提供商
openlmi-networking	用于网络管理的 CIM 提供商
openlmi-powermanagement	用于电源管理的 CIM 提供商
openlmi-service	用于管理系统服务的 CIM 提供商
openlmi-storage	用于存储管理的 CIM 提供商
openlmi-fan	用于控制 cpu 风扇的 CIM 提供商
openlmi-hardware	用于接收硬件信息的 CIM 提供商
openlmi-realmd	用于配置 realmd 的 CIM 提供商
openlmi-software [a]	用于软件管理的 CIM 提供商
[a] 在 NeoKylin Linux Advanced Server V7 中，OpenLMI 软件供应商是作为一个技术预览。此提供商是全功能的，但有一个性能问题，当列出了大量的软件包时，可能消耗的内存和时间过多。要解决这个问题，搜索返回尽可能少的包。	

5.2.2 安装 OpenLMI

OpenLMI 与一组 RPM 包绑定发行，其中包括 CIMOM，独立 CIM 提供商，

以及客户端应用程序。这使您可以区分管理端和客户端系统，只安装您需要的组件。

在管理端安装 OpenLMI

管理端是你通过使用 OpenLMI 客户端工具实施监控，管理的。

要在管理端上安装 OpenLMI，请完成以下步骤：

- 1) 安装 tog-pegasus 包，需要以 root 身份在 shell 里输入如下命令：

```
yum install tog-pegasus
```

此命令安装 OpenPegasus CIMOM 和它所有的依赖组件，并为 pegasus 使用者创建用户帐户。

- 2) 通过 root 身份运行以下命令，安装所需 CIM 提供商：

```
yum install openlmi-  
{storage,networking,service,account,powermanagement}
```

此命令安装 CIM 提供商，可用于存储，网络，服务，帐户和电源管理。随中标麒麟高级服务器操作系统 V7 绑定发行的 CIM 提供商的完整列表，请参阅表格 5-4 可用 CIM 提供商表格。

- 3) 编辑 the /etc/Pegasus/access.conf 自定义许连接到的 OpenPegasus CIMOM

用户的列表。默认情况下，只允许 pegasus 用户可以本地和远程访问 CIMOM。

要激活此用户帐户以 root 身份运行下面命令，并设置密码：

```
passwd pegasus
```

- 4) 通过激活 tog-pegasus.service 来启动 OpenPegasus CIMOM，以 root 身份运行如下命令：

```
systemctl start tog-pegasus.service
```

配置 tog-pegasus.service 服务开机自启动：

```
systemctl enable tog-pegasus.service
```

- 5) 如果你想从远程与管理系统进行交互，在端口 5989 使能 TCP 链接

(wbem-https)，以 ROOT 身份运行如下命令：


```
firewall-cmd --add -port 5989 /tcp
```

为 tcp 链接永久的打开 5989 端口：

```
firewall-cmd --permanent --add -port 5989 /tcp
```

现在，您可以通过使用 OpenLMI 客户端工具连接到管理系统，如 5.2.4 使用 LMI Shell 描述的。如果你打算直接在管理系统上执行 OpenLMI 操作，按照 0 在客户端安装 OpenLMI 中描述的步骤操作。

在客户端安装 OpenLMI

客户端系统是可以让你与管理系统交互的系统。在通常情况下，客户端系统和管理系统都安装在两台不同的机器上，但你也可以在管理系统上安装客户端工具和管理系统直接互动。

在客户端系统安装 OpenLMI 按照如下步骤：

1) 以 root 身份安装 openlmi-tools 包：

```
yum install openlmi-tools
```

这个命令安装了 LMIShell，它是一个交互式客户端和解释器用来访问由 OpenPegasus 提供的 CIM 对象，及其所有依赖。

2) 为 OpenPegasus 配置 SSL 证书如 5.2.3 为 OpenPegasus 配置 SSL 证书述的。

你现在可以使用 LMIShell 客户端正如 5.2.4 使用 LMI Shell。描述的。

5.2.3 为 OpenPegasus 配置 SSL 证书

OpenLMI 使用基于 Web 的企业级管理（WBEM）协议，其功能通过 HTTP 传输层。在这个协议中执行标准的 HTTP 基本认证，这意味着该用户名和密码和请求一同发送。

配置 OpenPegasus CIMOM 使用 HTTPS 进行通信以确保安全认证是必要的。管理系统上建立一个加密通道需要安全套接字层（SSL）或传输层安全（TLS）证书。

在系统中有两种方法管理 SSL/TLS 证书。

➤ 自签名证书需要较少的基础设施，但是更难以安全的部署到客户

机和管理端。

➤ 机构签发的证书更容易部署到客户端,但可能需要更大的初始投资。

当使用机构签发的证书,就必须在客户端系统上配置受信任的证书颁发机构。机构可以被用来签署的所有管理型系统的 CIMOM 证书。证书也可以是一个证书链的一部分,所以用于签名的管理系统的证书可能又被另一个更高一级的主管部门(如 Verisign, CAcert, RSA 及其他)签名。

默认的证书和信任的存储位置在表格 5-5 证书和信任的存储位置。

表格 5-5 证书和信任的存储位置

配置选项	位置	描述
sslCertificateFile Path	/etc/Pegasus/server .pem	CIMOM 的公共证书
sslKeyFilePath	/etc/Pegasus/file.p em	CIMOM 的私有 KEY
sslTrustStore	/etc/Pegasus/client. pem	文件或目录提供了信任证书 颁发机构的列表

重要说明:

如果您修改任何表格 5-5 证书和信任的存储位置提到的文件,重新启动 tog-peg asus 服务以确保它识别新证书。以 root 身份键入如下命令:

```
systemctl restart tog-pegasus.service
```

更多如何在 NeoKylin Linux Advanced Server V7 管理服务的内容请参考 3.1 使用 systemd 管理系统服务。

管理自签名证书

自签名证书使用自己的私钥签名本身并没有连接到任何信任链。在一个管理系统中,如果 tog-pegasus 服务启动之前管理员没有提供证书,一套自签名的证书将自动产生,该证书使用该系统的主机名做证书标题。

重要说明：

自动生成的自签名证书在默认情况下 10 年有效，但他们没有自动更新功能。任何修改都需要由 OpenSSL 或 Mozilla NSS 相关官方文档中提供的操作指南来手动创建新的证书。

在客户系统上配置信任自签名证书按下面步骤操作：

- 1) 从 管 理 系 统 拷 贝 /etc/Pegasus/server.Pem 证 书 到 客 户 机 的 /etc/pki/ca-trust/source/anchors/下，以 root 用户在 shell 里执行如下命令：

```
scp root@ hostname: /etc/Pegasus/server.pem /etc/pki /ca-trust/
source/anchors/pegasus-hostname.pem
```

替换管理系统的主机名。请注意，此命令只有 sshd 服务是在管理系统上运行才生效，管理系统被配置被允许 root 用户通过 SSH 协议登录。有关如何安装和配置 sshd 服务和使用 scp 命令通过 SSH 协议来传输文件的详细信息，请参见 3.2OpenSSH。

- 2) 通过对比原文件和新文件的校验值确认文件的完整性。在管理系统上计算 /etc/Pegasus/server.Pem 文件的校验值，以 root 身份运行如下命令：

```
sha1sum /etc/Pegasus/server.pem
```

在客户端校验/etc/pki/ca-trust/source/anchors/pegasus-hostname.Pem 文件输入如下命令：

```
sha1sum /etc/pki/ca-trust/source/anchors/pegasus-hostname.pem
```

用管理系统的主机名替代 hostname。

- 3) 在客户端更新信任存储需要以 root 用户运行如下命令：

```
update-ca-trust extract
```

管理机构签发的证书与身份管理(推荐)

NeoKylin Linux Advanced Server V7 的身份管理功能提供了简化的 SSL 证书中加入系统管理的域控制器。其中，身份管理服务器提供了一个嵌入式证书颁发机构。参考《NeoKylin Linux Advanced Server V7 Linux Domain 标识、身份验

证》或 FreeIPA 文档，获取更多关于如何加入到客户端和管理端的信息。

注册管理系统到身份管理是必要的;对客户端系统的注册是可选的。

在管理端执行如下步骤:

- 1) 安 装 ipa-client 包 , 像 《 NeoKylin Linux Advanced Server V7 Linux Domain 标识、身份验证》描述的注册系统到身份管理。
- 2) 拷贝身份管理签名到信任的存储系统, 以 root 身份执行如下命令:

```
cp /etc/ipa/ca.crt /etc/pki/ca-trust/source/anchors/ipa.crt
```

- 3) 更新存储以 root 身份执行如下命令:

```
update-ca-trust extract
```

- 4) 在身份管理域注册 Pegasus 服务, 以特权域用户执行如下命令:

```
ipaservice-add CIMOM/hostname
```

用管理系统的主机名替代 hostname.

这个命令可以从安装了 ipa-admintools 包中的身份管理域中的任何系统上运行。它创建一个可用于生成签名的 SSL 证书身份管理服务入口。

- 5) 备份本地的 PEM 文件到/etc/Pegasus/目录 (推荐)。
- 6) 以 root 用户执行以下命令检索签名证书:

```
ipa-getcert request -f /etc/Pegasus/server.pem -k  
/etc/Pegasus/file.pem -N CN= hostname -K CIMOM/hostname
```

用管理系统的主机名替代 hostname。

证书和密钥文件现在保存在适当的位置。通过 ipa-client-install 脚本安装 certmonger 守护进程来确保证书更新是必要的。

更多信息参考《NeoKylin Linux Advanced Server V7 Linux Domain 标识、身份验证》。

注册客户系统更新信任存储按如下步骤:

- 1) 安装 ipa-client 包, 参考《NeoKylin Linux Advanced Server V7 Linux Domain 标识、身份验证》描述的注册系统到身份管理。
- 2) 拷贝身份管理签名到信任的存储系统, 以 root 身份执行如下命令:

```
cp /etc/ipa/ca.crt /etc/pki/ca-trust/source/anchors/ipa.crt
```

3) 更新存储以 root 身份执行如下命令：

```
update-ca-trust extract
```

如果客户端系统并未在身份管理注册，请完成以下步骤来更新信任存储。

1) 以 root 用户从任何一个加入到同一个身份管理域的系统拷贝/etc/ipa/ca.crt 文件到信任存储域的/etc/pki/ca-trust/source/anchors/目录下：

2) 以 root 用户执行如下命令更新信任存储域：

```
update-ca-trust extract
```

手动管理机构签发的证书

使用其他机制管理机构签发的证书比使用身份管理机制需要更多的手动配置。

有必要以确保所有的客户端都信任将要签署的管理系统证书。

- 如果证书权限缺省可信，则不必执行任何特殊的步骤来完成此。
- 如果证书颁发机构默认是不信任的，证书必须在客户端由管理系统导入。

拷贝证书到信任的存储域需要以 root 身份执行如下命令：

```
cp /path/to/ca.crt /etc/pki/ca-trust/source/anchors/ca.crt
```

更新信任存储域需要以 root 身份执行如下命令：

```
update-ca-trust extract
```

在管理系统完成如下步骤：

1) 创建一个新的 SSL 配置文件/etc/Pegasus/ssl.cnf 存储证书的信息，文件的内容实例如下：

```
[ req ]
distinguished_name = req_distinguished_name
prompt = no
[ req_distinguished_name ]
C = US
ST = Massachusetts
L = Westford
O = Fedora
OU = Fedora OpenLMI
CN = hostname
```

使用管理系统的域名替代 hostname。

- 2) 以 root 身份执行如下命令在管理系统生成私有密钥:

```
openssl genrsa -out /etc/Pegasus/file.pem 1024
```

- 3) 以 root 身份执行如下命令生成证书:

```
openssl req -config /etc/Pegasus/ssl.cnf -new -key
/etc/Pegasus/file.pem -out /etc/Pegasus/server.csr
```

- 4) 将文件/etc/Pegasus/server.csr 发送到证书颁发机构进行签名。提交的文件的详细过程取决于特定认证机构。
- 5) 当从证书颁发机构收到签名证书,将文件存储为/etc/Pegasus/server. Pem。
- 6) 拷贝可信机构的证书到 Pegasus 信任的存储域,确保 Pegasus 能够相信自己的证书需要以 root 身份运行如下命令:

```
cp /path/to/ca.crt/etc/Pegasus/client.pem
```

完成所有描述的步骤后,即信任签名授权的客户都能够成功地与受管服务器的 CIMOM 通讯。

重要说明:

不同于身份管理解决方案的是如果证书过期,需要更新,所有描述的手动步骤,必须再次进行。建议在过期之前续订证书。

5.2.4 使用 LMI Shell

LMIShell 是一个交互式客户端同时其非交互式解释可用于访问由的 OpenPegasus CIMOM 提供的 CIM 对象。它是基于 Python 解释器,同时它还实现了附加的函数和类用来和 CIM 对象交互。

启动，使用，退出 LMIShell

类似于 Python 解释器，你可以使用 LMIShell 无论是作为一个交互式客户端或作为非交互式解释器来完成 LMIShell 脚本。

以交互模式启动 LMIShell

以交互模式启动 LMIShell，运行 `lmishell` 命令，无需其他参数。

```
lmishell
```

默认情况下，当 LMIShell 试图建立与 CIMOM 的连接，它验证由证书颁发机构信任存储的服务器端证书。要禁用此验证，命令使用 `--noverify` 或 `-n` 选项。

```
lmishell --noverify
```

使用 Tab 键完成

在交互模式下运行时，LMIShell 解释器允许您按 Tab 键完成基本的编程结构和 CIM 对象，包括命名空间，类，方法和对象属性。

浏览历史

默认情况下，LMIShell 在交互提示下存储所有你输入的命令到 `~/.lmishell_history` 文件。这可以让你浏览交互模式下命令历史记录，在交互模式下重新使用这些命令而无需再次键入。向后浏览的命令历史记录，按【上箭头】键或【Ctrl+ p】组合键。要向前浏览命令历史记录，按【下箭头】键或【Ctrl+ n】组合键。

LMIShell 还支持增量反向搜索。如果寻找特殊的命令历史记录，按【Ctrl+ r】某一线并开始键入命令的任何部分。 例如：

```
> (reverse-i-search)`co nnect': c = co nnect("server.example.com",  
"pegasus")
```

清空历史命令记录使用 `clear_history()` 函数。

```
clear_history()
```

您可以通过配置 `~/.lmishellrc` 文件中的 `history_leng` 值修改记录历史命令文件的行数。此外，你可以通过修改这个配置文件中的 `history_file` 选项的值更改此文件的位置。例如，设置历史文件位置为 `~/.lmishell_history` 的位置和最大行数为

1000。

```
history_file = "~/lmishell_history"

history_length = 1000
```

处理异常

默认情况下，LMIShell 解释器使用返回值处理所有的异常。要禁用此行为以处理代码中的所有异常，使用 `use_exceptions()` 函数：

```
use_exceptions()
```

要能使自动处理异常，如下方法：

```
use_exception(False)
```

通过修改 `~/lmishellrc` 配置文件中 `use_exceptions` 值你可以永久禁用异常处理。

```
use_exceptions = True
```

配置临时缓存

在默认配置下，LMIShell 连接对象使用一个临时的缓存存储 CIM 类名和 CIM 类以减少网络通信。要清除此临时缓存，使用 `clear_cache()` 方法，如下：

```
object_name.clear_cache()
```

用连接对象的名字代替 `object_name`。

要为特定的连接对象禁用临时缓存，请使用 `use_cache()` 方法，如下：

```
object_name.use_cache(False)
```

使能临时缓存：

```
object_name.use_cache(True)
```

通过修改 `~/lmishellrc` 配置文件中 `use_cache` 值你可以对连接对象永久禁用临时缓存。

```
use_cache = False
```

退出 LMIShell

要终止 LMIShell 解释器并返回到 shell 提示符下，按 **【Ctrl + d】** 组合键或发

出的 quit () 函数,如下:

```
> quit()

~]$
```

运行 LMIShell 脚本

运行 LMIShell 脚本，执行如下命令：

```
lmishell file_name
```

脚本名字代替 file_name，执行后检查的 LMIShell 脚本，可指定--interact 或 -i 命令行选项：

```
lmishell --interact file_name
```

LMIShell 脚本的首选文件扩展名是.lmi。

连接 CIMOM

LMIShell 允许您连接到的 CIMOM，要么是在同一个系统上本地运行要么在通过网络可以访问远程计算机上。

连接远程 CIMOM

要访问远程 CIMOM 提供的 CIM 对象，通过使用 connect () 函数，如下所示创建一个连接对象：

```
connect(host_name, user_name[, password])
```

用管理系统的主机名代替 host_name，用在该系统上可以连接 OpenPegasus CIMOM 的用户名代替 user_name,用户的密码替代 password。如果省略了密码，LMIShell 提示用户输入密码。该函数返回一个 LMICConnection 对象。

实例：连接远程 CIMOM

要以用户 pegasus 连接到运行在 server.example.com 上的 OpenPegasus CIMOM，以下交互提示：

```
> c = connect("server.example.com", "pegasus")

password:
```

```
>
```

连接本地 CIMOM

LMIShell 允许您使用 Unix 套接字连接到本地 CIMOM。对于这种类型的连接，你必须以 root 运行 LMIShell 解释器同时/var/run/tog-pegasus/cimxml.socket 套接字必须存在。

要访问本地 CIMOM 提供的 CIM 对象，通过使用 connect（）函数，如下所示创建一个连接对象：

```
connect(host_name)
```

用 localhost, 127.0.0.1, or::1 代替 hostname,函数返回一个连接对象或者空。

实例：连接一个本地 CIMOM

为了连接在本地以 root 身份运行的 OpenPegasus CIMOM，键入如下交互是提示：

```
> c = connect("localhost")
>
```

验证一个 CIMOM 连接

connect（）函数返回一个 LMICConnection 对象，如果连接不能建立返回空。此外，当 connect（）函数无法建立连接时，它输出错误消息到标准错误输出。

要验证到 CIMOM 的连接已经成功建立，使用 isinstance（）函数如下所示：

```
isinstance(object_name, LMICConnection)
```

用连接对象名替代 object_name，如果 object_name 是一个 LMICConnection 对象函数返回为 TRUE，否则返回 FALSE。

实例：验证一个 CIMOM 连接

要验证本章节例子连接远程 CIMOM: 中创建的 C 变量，包含 LMICConnection 对象，在互动提示请键入以下内容：

```
> isinstance(c, LMICConnection)

True

>
```

或者，你可以验证 c 不是 none.

```
> cis None

False

>
```

使用命令空间

LMIShell 命名空间提供了一种组织可用类的自然方法，并对其他的命名空间和类作为一个分层接入点。根命名空间是连接对象的第一个进入点。

列出可用的命名空间

要列出所有可用命名空间，请使用 `print_namespaces()` 方法，如下所示：

```
object_name.print_namespaces()
```

用对象名替代 `object_name`，这个方法会向标准输出打印出可用的命名空间。

获取命名空间列表，访问对象的命名空间属性。

```
object_name.namespaces
```

这个方法返回字符串列表。

实例：返回可用的命名空间

要检查本章节例子：连接远程 CIMOM 中创建的 C 连接对象的根命名空间的对象，并列出所有可用的命名空间，在交互提示下执行如下命令：

```
> c.root.print_namespaces()

cimv2

interop

PG_InterOp

PG_Internal

>
```

给一个变量 `root_namespaces` 分配命名空间列表，如下：

```
> root_namespaces = c.root.namespaces  
  
>
```

访问命名空间对象

要访问一个特定的命名空间使用如下命令：

```
object_name.namespace_name
```

要检查的对象名字替代 `object_name`，要访问的命名空间替代 `namespace_name`，命令返回 `LMINamespace` 对象。

实例：访问命名空间对象

要访问本章节例子：连接远程 CIMOM 中创建的 C 连接对象的 `cimv2` 命名空间，并将其分配给一个变量名为 `ns`，在交互提示键入如下：

```
> ns = c.root.cimv2  
  
>
```

使用类

`LMIShell` 类表示类由 CIMOM 提供。您可以访问并列出它们的属性，方法，实例，实例名称和 `ValueMap` 属性，打印他们的文档字符串，并创建新实例和实例名称。

列出可用的类

要列出特定名称空间所有的可用类，使用 `print_classes()` 方法，如下所示：

```
namespace_object.print_classes()
```

用命名空间对象名替代 `namespace_object`，此方法会打印所有可用类到标准输出。

使用 `classes()` 方法可以获取所有类。

```
namespace_object.classes()
```

这个方法返回字符串列表。

实例：列出可用类

要检查本章节实例访问命名空间对象创建的 ns 名称对象，并列出所有可用的类，键入如下：

```
> ns.print_classes()

CIM_CollectionInSystem
CIM_ConcreteIdentity
CIM_ControlledBy
CIM_DeviceSAPImplementation
CIM_MemberOfStatusCollection
...
>
```

分配类列表到变量 cimv2_classes:

```
> cimv2_classes = ns.classes()

>
```

访问类对象

要访问一个由 CIMOM 提供的特殊类对象，使用如下语法：

```
namespace_object.class_name
```

用命名空间对象名替代 namespace_object，要访问的对象名替代 class_name。

实例：访问类对象

要访问由本章节实例访问命名空间的 LMI_IP NetworkConnection 类对象，将其分配给一个变量名为 CLS，键入如下：

```
> cls = ns.LMI_IP NetworkConnection

>
```

检查类对象

所有类对象会存储有关他们的名字和他们所属的命名空间，以及详细类的文档资料。为了得到一个特定的类对象的名称，请使用以下语法：

```
class_object.classname
```

用类对象名替代 class_object，返回一个字符串来表示类名。

获取类所属命名空间的信息：

```
class_object.namespace
```

返回一个字符串表示命名空间。

显示类的详细信息文档使用 doc()方法：

```
class_object.doc()
```

实例：检查类对象

要检查本章节实例访问类对象中创建的 CLS 类对象，显示其名称和相应的命名空间，键入如下：

```
> cls.classname  
'LMI_IPNetworkConnection'  
  
> cls.namespace  
'root/cimv2'  
  
>
```

访问类文档：

```
> cls.doc()  
  
Class: LMI_IPNetworkConnection  
SuperClass: CIM_IPNetworkConnection  
[qualifier] string UMLPackagePath: 'CIM::Network::IP'  
[qualifier] string Version: '0.1.0'  
  
...
```

列出可用方法

要列出特定类的可用方法使用 print_methods()方法：

```
class_object.print_methods()
```

用类名替代 class_object，此方法打印可用的方法到标准输出：

获取可用方法列表用 methods()方法：

```
class_object.methods()
```

方法返回字符串列表。

实例：列出可用方法

要检查本章节实例访问类对象中创建的 cls 类对象，列出所有可用方法，键入如下：

```
> cls.print_methods()

RequestStateChange

>
```

分配方法列表到变量 service_methods

```
> service_methods = cls.methods()

>
```

列出可用属性

打印特定类的所有属性使用 print_properties()方法：

```
class_object.print_properties()
```

用类名替代 class_object，这个方法打印可用方法到标准输出。

获取可用属性列表使用 properties()方法：

```
class_object.properties()
```

方法返回字符串列表。

实例：列出可用属性

要检查本章节实例访问类对象中创建的 cls 类对象，列出所有可用的属性，键入如下：

```
> cls.print_properties()

RequestedState

HealthState

StatusDescriptions

TransitioningToState

Generation

...
```

```
>
```

分配类的列表给变量 `service_properties`。

```
> service_properties = cls.properties()
```

```
>
```

列出和查看 Value Map 属性

CIM 类包含 ValueMap 属性其在托管对象格式 (MOF) 中定义。ValueMap 属性包含常数，调用方法或检查返回值时是有用的。

列出一个特定类的所有 ValueMap 属性，使用 `print_valuemap_properties()` 方法，如下所示：

```
class_object.print_valuemap_properties()
```

用类对象名字替代 `class_object`，这个方法打印 ValueMap 到标准输出：

获取可用的 ValueMap 属性列表使用 `valuemap_properties()` 方法：

```
class_object.valuemap_properties()
```

方法返回字符串列表。

实例：列出可用的 ValueMap 属性

要检查本章节实例访问类对象中创建的 `cls` 类对象，列出所有可用的 ValueMap 的属性，键入如下：

```
> cls.print_valuemap_properties()
```

```
RequestedState
```

```
HealthState
```

```
TransitioningToState
```

```
DetailedStatus
```

```
OperationalStatus
```

```
...
```

```
>
```

分配 ValueMap 属性列表给变量 `service_valuemap_properties`


```
> service_valuemap_properties = cls.valuemap_properties()
>
```

访问特定的 ValueMap 属性，使用如下语法：

```
class_object.valuemap_propertyValues
```

用 ValueMap 属性名字替代 valuemap_property：

使用 print_values()方法列出所有可用的常量。

```
class_object.valuemap_propertyValues.print_values()
```

这种方法可打印可用的常量到标准输出。您也可以通过使用 values()方法提供常量列表。

```
class_object.valuemap_propertyValues.values()
```

方法返回字符串列表。

实例：访问 ValueMap 属性

本章节例子：列出可用的 ValueMap 属性提到了一个名为 RequestedState 的 ValueMap 属性。要检查这个属性，并列出可用的常量值，键入如下：

```
> cls.RequestedStateValues.print_values()
Reset
NoChange
NotApplicable
Quiesce
Unknown
...
>
```

分配常量值列表给 requested_state_values 变量：

```
> requested_state_values = cls.RequestedStateValues.values()
>
```

访问一个特定的常量使用如下语法：

```
class_object.valuemap_propertyValues.constant_value_name
```

用常量名替代 constant_value_name，另外，可以使用 value()方法如下：

```
class_object.valuemap_propertyValues.value("constant_value_name")
```

确定特定常量名使用 value_name()方法，如下：

```
class_object.valuemap_propertyValues.value_name("constant_value")
```

方法返回字符串列表。

实例：访问常量

本章节例子：访问 ValueMap 属性列出了 RequestedState 属性，其中一个常量名为 Reset，为访问这个常量，如下所示：

```
> cls.RequestedStateValues.Reset
11
> cls.RequestedStateValues.value("Reset")
11
>
```

确定常量的值名字：

```
> cls.RequestedStateValues.value_name(11)
u'Reset'
>
```

获取 CIMClass 类对象

很多类方法不需要访问 CIMClass 对象，这就是为什么 LMIShell 只有调用方法需要时才获取 CIMOM 这个对象。要手动获取 CIMClass 对象，使用 fetch () 方法，如下：

```
class_object.fetch()
```

用类名替代 class_object，需要注意的是需要访问 CIMClass 对象的方法自动获取它。

使用实例

LMIShell 实例表示由 CIMOM 提供实例。您可以获取并设置其属性，列出和调用它们的方法，其打印文档字符串，得到相关或关联对象的列表，推出修改的对象给 CIMOM，并且从 CIMOM 删除个别实例。

访问实例

为了得到一个特定的类对象的所有可用实例的列表，使用 `instances()` 方法，如下所示：

```
class_object.instances()
```

用类名替代 `class_object`，这个方法返回 `LMIIInstance` 对象列表。

访问类对象的第一个实例使用 `first_instance()` 方法：

```
class_object.first_instance()
```

方法返回 `LMIIInstance` 对象。

除了列出所有实例或返回第一个实例，`instances()` 和 `first_instance()` 支持可选参数，使您可以筛选结果：

```
class_object.instances(criteria)
class_object.first_instance(criteria)
```

用字典包含的键 - 值对替代 `criteria`，键代表实例的属性，值代表属性的值。

实例：访问实例

为了找到在本章节的例子：访问类对象中创建的 `cls` 类对象的第一个实例，其具有 `ElementName` 属性，属性值等于 `eth0`，把它分配给一个名为 `device` 变量，键入如下：

```
> device = cls.first_instance({"ElementName": "eth0 "})
>
```

测试实例

所有的实例对象存储的信息包括它们的类名和它们所属的命名空间，以及关于它们的属性和值的详细文档。此外，实例对象允许您检索一个唯一的识别对象。

获取特定实例对象的类名使用如下语法：

```
instance_object.classname
```

用实例对象名字替代 `instance_object`，返回类名字符串。

获取关于实例命名空间的信息，使用如下：

```
instance_object.namespace
```

这返回一个表示命名空间的字符串。

要检索唯一识别对象的实例对象，使用如下：

```
instance_object.path
```

返回一个 `LMIInstanceName` 对象。

最后，为了显示详细文档，使用 `doc()` 方法，如下所示：

```
instance_object.doc()
```

实例：测试实例

要检查本章节的例子：访问类实例中创建的设备实例对象，显示它的类名和相应的名称空间，键入如下：

```
> device.classname
u'LMI_IPNetworkConnection'
> device.namespace
'root/cimv2'
>
```

访问并列实例对象的文档，如下所示：

```
> device.doc()
Instance of LMI_IPNetworkConnection
[property] uint16 RequestedState = '12'
[property] uint16 HealthState
[property array] string [] StatusDescriptions
...
```

创建新实例

某些 CIM 允许您创建特定的类对象的新实例。要创建一个类对象的新实例，使用 `create_instance()` 方法，如下所示：

```
class_object.create_instance(properties)
```

使用类对象名替代 `class_object`，字典的键值对替代 `properties`，键代表实例的属性，值代表属性的值。此方法返回 `LMIInstance` 对象。

实例：创建新属性

`LMI_Group` 类表示系统组，`LMI_Account` 类表示在管理系统上的用户帐户。使用本章节例子：访问命名空间对象中创建的 `ns` 命名空间对象，为系统组 `pegasus` 和用户 `lmishell-user` 创建两个类的实例，并分别将它们分配到变量 `group` 和 `user`，如下所示：

```
> group = ns.LMI_Group.first_instance({"Name": "pegasus"})
> user = ns.LMI_Account.first_instance({"Name": "lmishell-user"})
>
```

为了获取 `lmishell-user` 用户的 `LMI_Identity` 类的实例，如下所示：

```
> identity = user.first_associator(ResultClass= "LMI_Identity")
>
```

`LMI_MemberOfGroup` 类表示系统组成员。要使用 `LMI_MemberOfGroup` 类的 `lmishell-user` 添加 `pegasus` 组，创建这个类的新实例如下所示：

```
> ns.LMI_MemberOfGroup.create_instance({
... "Member": identity.path,
... "Collection": group.path})
LMIInstance(classname="LMI_MemberOfGroup", ...)
>
```

删除单个实例

为了从 CIMOM 删除特定实例，使用 `delete()` 方法，如下所示：

```
instance_object.delete()
```

用要被删除的实例名替代 `nstance_object`，该方法返回一个布尔值。请注意，

删除一个实例之后，其属性和方法无法访问。

实例：删除单个实例

LMI_Account 类表示在管理系统上的用户帐户。使用本章节例子：访问命名空间对象中创建的 ns 命名空间，创建 LMI_Account 类中用户名为 lmishell-user 的一个实例，并将其分配给一个变量 user，如下所示：

```
> user = ns.LMI_Account.first_instance({"Name" : "lmishell-user"})  
>
```

为了删除这个实例，并从系统中移除 lmishell-user，如下所示：

```
> user.delete()  
  
True  
  
>
```

列出并访问可用属性

列出特殊实例对象的可用属性列表，使用 print_properties()方法如下所示：

```
instance_object.print_properties()
```

用实例名替代 instance_object，本方法提供打印可用属性到标准输出。

列出可用属性列表使用 properties()方法：

获取可用属性的列表，使用 properties()方法：

```
instance_object.properties()
```

方法返回字符串列表。

实例：列出可用属性

要检查本章节实例：访问命名空间对象中创建的设备实例对象并列出所有可用的属性，如下所示：

```
> device.print_properties()  
  
RequestedState  
HealthState  
StatusDescriptions  
TransitioningToState
```

```
Generation
```

```
...
```

```
>
```

分配属性列表给变量 `device_properties`。

```
> device_properties = device.properties()
```

```
>
```

获取特定属性的当前值使用如下语法：

```
instance_object.property_name
```

用属性名替代 `property_name`。

为了修改特定属性的值，分配一个值给该属性。

```
instance_object.property_name = value
```

用属性值替代 `value`，需要注意的是，为了更改通知到 CIMOM，还必须执行 `push()` 方法。

```
instance_object.push()
```

这个方法返回一个三元素元组，该元组由一个返回值，返回值的参数和一个错误字符串组成。

实例：访问单个属性

要检查本章节实例：访问命名空间对象中创建的设备实例对象，显示 `SystemName` 属性的值，如下所示：

```
> device.SystemName
```

```
u'server.example.com'
```

```
>
```

列出和使用可用方法

要列出特定实例对象的所有可用的方法，使用 `print_methods()` 方法，如下所示：

```
instance_object.print_methods()
```

用实例对象名替代 `instance_object`，此方法打印可用方法到标准输出。

列出可用方法用 `method ()`方法：

```
instance_object.methods()
```

方法返回字符串列表。

实例：列出可用方法

要检查本章节实例：访问命名空间对象中创建的设备实例对象，列出所有可用的方法，如下所示：

```
> device.print_methods()

RequestStateChange

>
```

分配方法列表给变量 `network_device_methods`：

```
> network_device_methods = device.methods()

>
```

调用特殊方法使用下面语法：

```
instance_object.method_name(
    parameter=value,
    ...)
```

将要使用的实例对象名替代 `instance_object`，要调用的方法替代 `method_name`，要设置的参数替代 `parameter`，要设置此参数的值替代 `value`，这个方法返回一个三元素元组，该元组由一个返回值，返回值的参数和一个错误字符串组成。

重要说明：

`LMIIInstance` 对象不自动刷新其内容（属性，方法，限定符，等等）。要做到这一点，使用 `refresh ()` 方法，如下所述。

实例：使用方法

`PG_ComputerSystem` 对象代表系统，要通过本章实例：访问命名空间对象中创建的 `ns` 命名空间，将其分配给一个名为 `sys` 的变量，创建这个类的一个实例，如下所示：


```
> sys = ns.PG_ComputerSystem.first_instance()

>
```

LMI_AccountManagementService 类实现方法，使您可以在系统中管理用户和组。创建这个类的一个实例，并将其分配给一个变量 acc，如下所示：

```
> acc = ns.LMI_AccountManagementService.first_instance()

>
```

在系统中使用 CreateAccount()方法创建一个新用户 lmishell-user，如下所示：

```
> acc. CreateAccount(Name= "lmishell -user", System= sys)

LMIReturnValue(rval=0, rparams=NocaseDict({u'Account':
LMIInstanceName(classname="LMI_Account"...), u'Identities':
[LMIInstanceName(classname="LMI_Identity"...),
LMIInstanceName(classname="LMI_Identity"...)]}), errorstr=")
```

LMIShell 支持同步的方法调用：当您使用同步方法，LMIShell 等待相应的作业对象知道其状态变为“已完成”，然后返回此作业的返回参数。如果给定的方法返回的对象是下面的其中之一，LMIShell 则能够执行同步方法调用。

- LMI_StorageJob
- LMI_SoftwareInstallationJob
- LMI_NetworkJob

LMIShell 首先尝试使用指示作为等待的方法。如果失败了，它采用了轮询方法来代替。要执行同步方法调用，使用以下语法：

```
instance_object.Syncmethod_name(
    parameter=value,
    ...)
```

要使用的实例对象的名字替代 instance_object，要调用的方法替代 method_name，要设置的参数替代 parameter，要设置参数的值替代 value，所有的同步方法名都有 sync 前缀，所有方法返回一个三元组，包括返回值，返回值参数和错误字符串。

你可以强制 LMIShell 强制只使用轮询方法，设置参数 PreferPolling 如下所

示:

```
instance_object.Syncmethod_name(  
    P referPolling=T rue  
    parameter=value,  
    ...)
```

列出并查看 Value Map 参数

CIM 方法在托管对象格式 (MOF) 定义中包含 ValueMap 参数。 ValueMap 参数包含常量。

要列出一个特定方法的所有可用 ValueMap 参数，使用 `print_valuemap_parameters()`方法，如下所示：

```
instance_object.method_name.print_valuemap_parameters()
```

实例对象名替代 `instance_object`，要检查的方法替代 `method_name`。方法打印可用的 ValueMap 参数到标准输出。

要获取 ValueMap 的可用参数使用 `valuemap_parameters()`方法，如下所示：

```
instance_object.method_name.valuemap_parameters()
```

方法返回字符串列表。

实例：列出 ValueMap 参数

要检查本章实例：使用方法中创建的 `acc` 实例对象，并列出 `CreateAccount` () 方法的所有可用 ValueMap 参数，键入以下交互提示：

```
> acc.CreateAccount.print_valuemap_parameters()  
CreateAccount  
>
```

分配了 ValueMap 参数列表给 `create_account_parameters` 变量：

```
> create_account_parameters = acc.CreateAccount.valuemap_parameters()  
>
```

访问一个特定的 ValueMap 参数使用如下语法：

```
instance_object.method_name.valuemap_parameterValues
```

使用要访问的 ValueMap 参数名替代 valuemap_parameter。

列出所有可用的常量，使用 print_values()方法，如下所示：

```
instance_object.method_name.valuemap_parameterValues.print_values()
```

这种方法打印可用命名的常量值到标准输出。您也可以通过使用 values()方法获取常量值列表。

```
instance_object.method_name.valuemap_parameterValues.values()
```

此方法返回字符串列表。

实例：访问 ValueMap 参数

在本章的列出 ValueMap 参数实例中，用到了 CreateAccount 参数。检测此参数并列出可用的常量值，如下所示：

```
> acc.CreateAccount. CreateAccountValues. print_values()
Operationunsupported
Failed
Unabletosetpasswordusercreated
Unabletocreatehomedirectoryusercreatedandpasswordset
Operationcompletedsuccessfully
>
```

分配常量列表给变量 create_account_values，如下所示：

```
> create_account_values =
acc.CreateAccount.CreateAccountValues.values()
>
```

访问一个特定常量值使用如下语法：

```
instance_object.method_name.valuemap_parameterValues.constant_value_name
```

使用常量名替代 constant_value_name，你也可以使用 value()方法，如下所示：

```
instance_object.method_name.valuemap_parameterValues.value("constant_val
ue_name")
```

确定特殊常量名使用 value_name()方法:

```
instance_object.method_name.valuemap_parameterValues.value_name("constant_value")
```

方法返回字符串列表。

实例：访问常量

在本章的访问 ValueMap 参数实例中，给 ValueMap 参数 CreateAccount 提供了一个常数 Failed。要访问此常量，键入以下交互提示：

```
> acc.CreateAccount.CreateAccountValues.Failed
2
> acc.CreateAccount.CreateAccountValues.value("Failed ")
2
>
```

确定常量的名字，如下所示：

```
> acc.CreateAccount.CreateAccountValues.value_name(2)
u'Failed'
>
```

刷新实例对象

使用 LMIShell 本地对象，代表了在 CIMOM 的旧的 CIM 对象，如果在使用 LMIShell 时，对象发生改变，需要更新特定实例对象的属性和方法，使用 refresh

() 方法，如下所示：

```
instance_object.refresh()
```

使用要刷新的对象名替代 instance_object。这个方法返回一个三元素元组，该元组由一个返回值，返回值的参数和一个错误字符串组成。

实例：刷新实例对象

键入以下交互提示，可以更新本章访问实例中创建的设备实例对象的方法和属性：

```
> device.refresh()

LMIReturnValue(rval=True, rparams=NocaseDict({}), errorstr=")

>
```

显示托管对象格式表示

为了显示托管对象格式表示的实例对象使用 `tomof()` 方法，如下所示：

```
instance_object.tomof()
```

使用要检查的实例对象名替代 `instance_object`。此方法打印托管对象格式表示到标准输出。

实例：显示托管对象格式表示

键入以下交互提示，可以显示本章访问实例中创建的设备实例对象的托管对象格式，：

```
> device.tomof()

instance of LMI_IPNetworkConnection {

RequestedState = 12;

HealthState = NULL;

StatusDescriptions = NULL;

TransitioningToState = 12;

...
}
```

使用实例名称

`LMIShell` 实例名称对象包含了一组键和值。这种类型的一个对象的识别一个实例。

访问实例名称

`CIMInstance` 对象被 `CIMInstanceName` 对象标识。要获得所有可用的实例名称对象的列表，使用 `instance_names()` 方法，如下：

```
class_object.instance_names()
```

使用要检查的类对象名替代 `class_object`。此方法返回 `LMIIInstanceName` 对象列表。

要访问类对象的第一个实例名称对象使用 `first_instance_name()`方法，如下所示：

```
class_object.first_instance_name()
```

方法返回一个 `LMIIInstanceName` 对象。

除了列出的所有实例名称对象或返回第一个实例名称对象，`instance_names()`和 `first_instance_name()`都支持可选参数，使您可以这些参数过滤结果。

```
class_object.instance_names(criteria)
class_object.first_instance_name(criteria)
```

使用字典中的一个键值对替代 `criteria`，键代表属性，值代表属性的值。

实例：访问实例名称

为了找到本章实例访问类对象中，创建的 `cls` 类对象的第一个实例名称，其属性为 `name`，属性值为 `eth0`，将其分配给 `device_name` 变量，键入以下交互提示：

```
> device_name = cls.first_instance_name({"Name": "eth0 "})
>
```

测试实例名称

所有的实例名对象存储他们的类名和他们所属的命名空间信息。

为了获取特定实例对象的名字使用如下语法：

```
instance_name_object.classname
```

用要检查的实例名对象名替代 `instance_name_object`。此方法返回一个类名字符串。

要获取有关实例名对象所属的命名空间的信息，如下所示：

```
instance_name_object.namespace
```

返回命名空间的字符串。

实例：测试实例名称

键入以下交互提示，可以检查本章访问实例名实例中，创建的 device_name 实例名对象，显示它的类名和相应的命名空间：

```
> device_name.classname
u'LMI_IPNetworkConnection'
> device_name.namespace
'root/cimv2'
>
```

创建新实例名

LMIShell 允许你创建一个新的包装 CIMInstanceName 的对象，如果你知道一个远程对象的所有主键。该实例名对象可用于检索整个实例对象。

要创建一个类对象的新实例名称，使用 new_instance_name() 方法，如下：

```
class_object.new_instance_name(key_properties)
```

类对象名替代 class_object，键值对替代 key_properties，键代表属性，值代表属性值。此方法返回一个 LMIInstanceName 对象。

实例：创建新实例名称

LMI_Account 类代表在管理系统上的用户帐户。键入以下交互提示，可以使用本章实例访问命名空间对象中，创建的 ns 命名空间，创建 LMI_Account 类的新实例名称，其代表在管理系统上的 lmishell-user 用户：

```
> instance_name = ns.LMI_Account.new_instance_name({
... "CreationClassName": "LMI_Account",
... "Name": "lmishell-user",
... "SystemCreationClassName": "PG_ComputerSystem",
... "SystemName": "server"})
>
```

列出并访问关键属性

列出特定实例名称的所有可用关键属性，使用 print_key_properties() 方法，如下所示：

```
instance_name_object.print_key_properties()
```

使用要检查的实例名对象替代 `instance_name_object`，此方法打印可用的关键属性到标准输出。

为了获取关键属性列表，使用 `key_properties()` 方法：

```
instance_name_object.key_properties()
```

方法返回字符串列表

实例：列出可用的关键属性

键入以下交互提示，可以检查本章访问实例名实例中，创建的 `device_name` 实例名对象，列出所有可用的关键属性：

```
> device_name.print_key_properties()
CreationClassName
SystemName
Name
SystemCreationClassName
>
```

将关键属性列表分配给变量 `device_name_properties`，如下：

```
> device_name_properties = device_name.key_properties()
>
```

获取特定关键属性的值使用以下语法：

```
instance_name_object.key_property_name
```

使用要访问的关键属性名替代 `key_property_name`。

实例：访问某一个关键属性

键入以下交互提示，可以检查本章实例访问实例名中创建的 `device_name` 实例名对象，显示其 `SystemName` 关键属性的值：

```
> device_name.SystemName
u'server.example.com'
```



```
>
```

将实例名称转化为实例

每一个实例名都可以转化为实例，使用 `to_instance()` 方法，如下：

```
instance_name_object.to_instance()
```

将要转化的实例名对象的名称替代 `instance_name_object`。此方法返回 `LMIInstance` 对象。

实例：将实例名称转化为实例

键入以下交互提示，可以将本章实例访问实例名中创建的 `device_name` 实例名对象，转换为一个实例对象，并将其分配给变量 `device`：

```
> device = device_name.to_instance()
>
```

使用被关联对象

通用信息模型定义了管理对象之间的关联关系。

访问被关联实例

为了获取特定实例对象的所有关联对象，使用 `associators()` 方法，如下：

```
instance_object.associators(
    AssocClass=class_name,
    ResultClass=class_name,
    ResultRole=role,
    IncludeQualifiers=include_qualifiers,
    IncludeClassOrigin=include_class_origin,
    PropertyList=property_list)
```

访问特定实例对象的第一个关联对象使用 `first_associator()` 方法。

```
instance_object.first_associator(
    AssocClass=class_name,
    ResultClass=class_name,
```

```
ResultRole=role,  
IncludeQualifiers=include_qualifiers,  
IncludeClassOrigin=include_class_origin,  
PropertyList=property_list)
```

将 instance_object 用要检查的实例对象名替代，你可以使用下面的参数来过滤结果。

- **AssocClass:** 每个返回的对象必须与源对象通过这个类或它的一个子类的实例相关联。默认值是 none。
- **ResultClass:** 每个返回的对象必须是这个类的实例或它的一个子类，或者必须是这个类或它的一个子类。默认值是 none。
- **Role:** 每个返回的对象必须与源对象通过在其中源对象扮演特定角色的关联相关联。关联类中的属性名指定的源对象的参数和值必须相匹配。默认值是 none。
- **ResultRole:** 每个返回的对象必须与源对象通过在其中返回的对象扮演特定角色的关联相关联。关联类中的属性名指定的返回对象的参数和值必须相匹配。默认值是 none。

剩余的参数参考如下：

- **IncludeQualifiers:** 一个布尔值，指示是否每个对象（包括任何对象返回属性的 qualifiers）的所有 qualifiers 在响应时都应包括在 QUALIFIER 元素中。默认值是 false。
- **IncludeClassOrigin:** 一个布尔值，指示 CLASSORIGIN 属性是否应该出现在每个返回对象的所有适当元素中。默认值是 false。
- **PropertyList:** 该列表的成员定义一个或多个属性名。返回的对象将不包括在这个列表中缺少任何一个属性的元素。如果 PropertyList 是一个空列表，没有任何属性都包含在返回的对象中。如果它是 none，没有附加的过滤器被定义。默认值是 none。

实例：访问关联实例

LMI_StorageExtent 类表示系统中可用的块设备。键入以下交互提示，可以利用本章实例访问命名空间对象中创建的 ns 命名空间，为/dev/vda 块设备创建 LMI_StorageExtent 类的一个实例，并将其分配给 vda 变量：

```
> vda = ns.LMI_StorageExtent.first_instance({
... "DeviceID" : "/dev/vda"})
>
```

获取这个块设备所有的磁盘分区列表，将其分配给变量 vda_partitions，使用 associators()方法，如下：

```
> vda_partitions = vda.associators(ResultClass="LMI_DiskPartition")
>
```

访问关联实例名称

获取特定实例对象的关联实例名称列表使用 associator_names()方法，如下：

```
instance_object.associator_names(
    AssocClass=class_name,
    ResultClass=class_name,
    Role=role,
    ResultRole=role)
```

访问特定实例对象的第一个关联实例名称，使用 first_associator_name()方法，如下：

```
instance_object.first_associator_name(
    AssocClass=class_object,
    ResultClass=class_object,
    Role=role,
    ResultRole=role)
```

用要检查的实例对象名替代 instance_object，你可以通过下面的参数过滤结果：

- **AssocClass**：每个返回的名称标识一个对象，该对象必须与源对象通过这个类的实例或它的一个子类相关联。默认值是 none。

➤ **ResultClass:** 每个返回的名称标识一个对象，该对象必须是这个类的实例或它的一个子类，或者必须是这个类或它的一个子类。默认值是 `none`。

➤ **Role:** 每个返回名称标识一个对象，该对象必须与源对象通过在其源对象扮演指定的角色的关联相关联。关联类中的属性名指定的源对象的参数和值必须相匹配。默认值是 `none`。

➤ **ResuleRole:** 每个返回名称标识一个对象，该对象必须与源对象通过在其返回的名称对象扮演指定的角色的关联相关联。关联类中的属性名指定的返回对象的参数和值必须相匹配。默认值是 `none`。

实例：访问关联实例名称

使用本章访问关联实例中创建的 `vda` 实例对象，获得其关联的实例名列表，并将其分配给一个变量 `vda_partitions`，如下：

```
> vda_partitions =  
vda. associator_names(ResultClass= "LMI_D i skP arti ti o n")  
>
```

使用关联对象

通用信息模型定义了管理对象之间的关联关系。关联对象定义其它两个对象之间的关系。

访问关联实例

获取一个特定目标对象的关联对象列表，使用 `references()`方法，如下：

```
instance_object.references(  
    ResultClass=class_name,  
    Role=role,  
    IncludeQualifiers=include_qualifiers,  
    IncludeClassOrigin=include_class_origin,  
    PropertyList=property_list)
```

访问特定目标对象的第一个关联对象使用 `first_reference()`方法：

```
instance_object.filter_reference(
... ResultClass=class_name,
... Role=role,
... IncludeQualifiers=include_qualifiers,
... IncludeClassOrigin=include_class_origin,
... PropertyList=property_list)
>
```

用要检查的实例对象名替代 `instance_object`，你可以通过下面的参数过滤结果：

- **ResultClass**：每个返回的对象必须是这个类的实例或它的一个子类，或者必须是这个类或它的一个子类。默认值是 `none`。
- **Role**：每个返回的对象表示的目标对象其需通过匹配属性参数的值。默认值是 `none`。

剩余的参数介绍如下：

- **IncludeQualifiers**：一个布尔值，指示是否每个对象（包括任何对象返回属性的 `qualifiers`）的所有 `qualifiers` 在响应时都应包括在 `QUALIFIER` 元素中。默认值是 `false`。
- **IncludeClassOrigin**：一个布尔值，指示 `CLASSORIGIN` 属性是否应该出现在每个返回对象的所有适当元素中。默认值是 `false`。
- **PropertyList**：该列表的成员定义一个或多个属性名。返回的对象将不包括在这个列表中缺少任何一个属性的元素。如果 `PropertyList` 是一个空列表，没有任何属性都包含在返回的对象中。如果它是 `none`，没有附加的过滤器被定义。默认值是 `none`。

实例：访问关联实例

`LMI_LANEndpoint` 类代表和某一个特定的网络接口设备相关联的通信端点。键入以下交互提示，可以使用本章访问命名空间实例中创建的 `ns` 命名空间，为网络接口设备 `eth0` 创建一个 `LMI_LANEndpoint` 类的实例，并将其分配给一个变量 `lan_endpoint`：

```
> lan_endpoint = ns.LMI_LANEndpoint.first_instance({  
... "Name": "eth0"})  
  
>
```

访问第一个关联对象 LMI_BindsToLANEndpoint 对象, 将其分配给变量 bind, 如下:

```
> bind = lan_endpoint.first_reference(  
... ResultClass= "LMI_BindsToLANEndpoint")  
  
>
```

你可以使用 Dependent 属性来访问 LMI_IPProtocolEndpoint 类, 该类代表网络接口设备的 IP 地址。

```
> ip = bind.Dependent.to_instance()  
  
> print ip.IPv4Address  
  
192.168.122.1  
  
>
```

访问关联实例名称

获取特定实例对象的关联实例名称列表使用 reference_names() 方法, 如下所示:

```
instance_object.reference_names(  
ResultClass=class_name,  
Role=role)
```

访问第一个特定实例对象的关联实例名称使用 first_reference_name() 方法:

```
instance_object.first_reference_name(  
ResultClass=class_name,  
Role=role)
```

用要检查的实例对象名替代 instance_object, 你可以通过下面的参数来过滤结果:

- **ResultClass:** 每个返回的对象的名称标识该类的实例或它的一个子类, 或者该类或其子类之一。默认值是 none。

➤ **Role:** 每个返回对象标识的对象指的是通过实例属性与参数匹配的目标。默认值是 none。

实例：访问关联实例名

要使用本章访问关联实例中创建的 `lan_endpoint` 实例对象，访问其第一个关联实例名 `LMI_BindsToLANEndpoint` 对象，并将其分配给变量 `bind`，如下：

```
> bind = lan_endpoint.first_reference_name(
... ResultClass= "LMI_BindsToLANEndpoint")
```

你可以使用 `Dependent` 属性来访问 `LMI_IPProtocolEndpoint` 类，该类代表网络接口设备的 IP 地址。

```
> ip = bind.Dependent.to_instance()
> print ip.IPv4Address
192.168.122.1
>
```

使用指示器

指示器是对某些特定事件的响应，这些事件是因为数据发生了特定的改变而响应的。`LMIShell` 可以订阅指示器以便接收这样的事件的响应。

订阅指示器

为了订阅一个指示器，使用 `subscribe_indication()`方法，如下所示：

```
connection_object.subscribe_indication(
    QueryLanguage="WQL",
    Query='SELECT * FROM CIM_InstModification',
    Name="cpu",
    CreationNamespace="root/interop",
    SubscriptionCreationClassName="CIM_IndicationSubscription",
    FilterCreationClassName="CIM_IndicationFilter",
    FilterSystemCreationClassName="CIM_ComputerSystem",
    FilterSourceNamespace="root/cimv2",
```

```

Hand l erC reati o nC l assName="C IM_Ind i cati o nHand l erC
IMXML",

Hand l erSystemC reati o nC l assName="C IM_C o mputerSystem",

D esti nati o n="http: //host_name: 59 88")

```

另外，你可以使用一个参数更少的版本：

```

connection_object.subscri be_i nd i cati o n(

Q uery=' SELEC T * FR O M C IM_InstMo d i f i cati o n' ,

Name="cpu",

D esti nati o n="http: //host_name: 59 88")

```

使用要连接的对象替代 connection_object，使用你想订阅指示器的系统主机名替代 host_name。

默认情况下，解释器终止时，由 LMIShell 解释创建的所有订阅将被自动删除。要改变这种行为，调用 subscribe_indication () 方法时通过设置关键字参数 Permanent=True，这将防止 LMIShell 删除预订。

实例：订阅指示器

键入以下交互提示，可以使用本章连接远程 CIMOM 实例中创建的 c 连接对象，订阅名为 CPU 的指示器：

```

> c. subscri be_i nd i cati o n(
... Q ueryLang uag e= "WQ L",
... Q uery= ' SELEC T * FR O M C IM_InstMo d i f i cati o n' ,
... Name= "cpu",
... C reati o nNamespace= "ro o t/i ntero p",
... Subscri pti o nC reati o nC l assName= "C IM_Ind i cati o nSubscri pti
o n",
... Fi l terC reati o nC l assName= "C IM_Ind i cati o nFi l ter",
... Fi l terSystemC reati o nC l assName= "C IM_C o mputerSystem",
... Fi l terSo urceNamespace= "ro o t/ci mv2",
... Hand l erC reati o nC l assName= "C IM_Ind i cati o nHand l erC

```



```
IMXML",
... Hand l erSystemC reati o nC l assName= "C IM_C o mputerSystem",
... D esti nati o n= "http: //server. exampl e. co m: 59 88")
LMIReturnValue(rval=True, rparams=NocaseDict({ }), errorstr=")
>
```

列出订阅的指示器

列出所有订阅的指示器使用 `print_subscribed_indications()`方法，如下：

```
connection_object.print_subscribed_indications()
```

使用要检查的关联对象名替代 `connection_object`，此方法打印被订阅的指示器到标准输出。

获取订阅指示器列表使用 `subscribed_indications()`方法：

```
connection_object.subscribed_indications()
```

方法返回字符串列表。

实例：列出订阅的指示器

键入以下交互提示，可以检查本章连接远程 CIMOM 实例中创建的 c 连接对象，列出所有订阅的指示器：

```
> c. pri nt_subscri bed _i nd i cati o ns()
>
```

分配指示器类别给变量 `indications`，如下所示：

```
> i nd i cati o ns = c. subscri bed _i nd i cati o ns()
>
```

退订指示器

默认情况下，解释器终止时，由 LMIShell 解释创建的所有订阅将被自动删除。要删除单个订阅，使用 `unsubscribe_indication()`方法，如下：

```
connection_object.unsubscribe_indication(indication_name)
```

使用要连接的对象名替代 `connection_object`，要删除的订阅名替代 `indication_name`。

要删除所有的订阅使用 `unsubscribe_all_indications()`方法:

```
connection_object.unsubscribe_all_indications()
```

实例：退订指示器

键入以下交互提示,可以使用本章连接远程 CIMOM 实例中创建的 c 连接的对象,退订从本章订阅指示器实例中创建的指示器:

```
> c.unsubscribe_indication('cpu')  
  
LMIReturnValue(rval=True, rparams=NocaseDict({}), errorstr="")  
  
>
```

实现一个指示器处理句柄

`subscribe_indication()` 方法允许你指定要订阅指示器系统的主机名。下面的示例演示如何实现的指示器处理句柄:

```
> def handler(ind, arg1, arg2, **kwargs):  
...     exported_objects = ind.exported_objects()  
...     do_something_with(exported_objects)  
> listener = LmiIndicationListener("0.0.0.0", listening_port)  
> listener.add_handler("indication-name-XXXXXXXX", handler, arg1, arg2,  
**kwargs)  
> listener.start()  
>
```

句柄函数的第一个参数是 `Lmi Ind i cati o n` 对象,其包含方法列表和指示器输出的对象。其它的参数是用户相关的,当被添加到句柄函数中用来监听是这些参数需要赋特定值。

上面的例子中,`add_handler()`方法调用使用了一个 8 个 x 的特殊的字符串。这些字符串会被监听值随机产生的字符串替代,用来避免名字冲突。为了使用随机字符串,首先启动监听指示器,然后订阅指示器,这样目标对象的 `Destination` 属性就包含如下值: `schema://host_name/random_string`.

实例：实现指示器处理句柄

下面的脚本说明了如何编写一个处理程序,监视位于 192.168.122.1 的管理系统和每当一个新的用户帐户创建时调用 `indication_callback()` 函数。

```
#!/usr/bin/lmishell

import sys
from time import sleep
from lmi.shell.LMIUtil import LMIPassByRef
from lmi.shell.LMIIndicationListener import LMIIndicationListener

# These are passed by reference to indication_callback
var1 = LMIPassByRef("some_value")
var2 = LMIPassByRef("some_other_value")

def indication_callback(ind, var1, var2):
    # Do something with ind, var1 and var2
    print ind.exported_objects()
    print var1.value
    print var2.value

c = connect("hostname", "username", "password")

listener = LMIIndicationListener("0.0.0.0", 65500)
unique_name = listener.add_handler(
    "demo-XXXXXXX",      # Creates a unique name for me
    indication_callback, # Callback to be called
    var1,                # Variable passed by ref
    var2                 # Variable passed by ref
)

listener.start()

print c.subscribe_indication(
    Name=unique_name,
    Query="SELECT * FROM LMI_AccountInstanceCreationIndication WHERE
SOURCEINSTANCE ISA LMI_Account",
    Destination="192.168.122.1:65500"
)

try:
    while True:
        sleep(60)
except KeyboardInterrupt:
    sys.exit(0)
```

使用示例

本节为不同的 CIM 提供者提供了一些例子，这些提供者和 OpenLMI 包一同被发布。在本节中的所有示例使用以下两个变量。

```
c = connect("host_name", "user_name", "password")
ns = c.root.cimv2
```

使用管理系统的主机名替代 host_name,允许被连接到运行 OpenPegasus CIMOM 的系统的用户名替代 user_name,用户的密码替代 password.

使用 OpenLMI 服务者

openlmi-service 包为管理系统服务安装了一个 CIM 提供者。下面的例子说明如何使用这个 CIM 提供者来列出可用的系统服务，以及如何启动，停止，启用和禁用它们。

实例：列出可用服务

要列出受控计算机上的所有可用的服务，以及有关服务是否已启动（TRUE）或停止（FALSE）信息和状态字符串，可以使用下面的代码片段：

```
for service in ns.LMI_Service.instances():  
    print "%s:\t%s" % (service.Name, service.Status)
```

要只列出默认使能的服务使用下面代码片段：

```
cls = ns.LMI_Service  
for service in cls.instances():  
    if service.EnabledDefault == cls.EnabledDefaultValues.Enabled:  
        print service.Name
```

注意 EnabledDefault 属性的值等于 2 时使能服务，等于 3 时关闭服务。

列出关于 cups 服务的信息使用下面代码：

```
cups = ns.LMI_Service.first_instance({"Name": "cups.service"})  
cups.doc()
```

实例：开始和停止服务

开始和停止 cups 服务，查看服务状态，使用下面的代码：

```
cups = ns.LMI_Service.first_instance({"Name": "cups.service"})  
cups.StartService()  
print cups.Status  
cups.StopService()  
print cups.Status
```

实例：使能和关闭服务

使能和关闭 cups 服务并显示 EnabledDefault 值，使用如下代码片段：

```
cups = ns.LMI_Service.first_instance({"Name": "cups.service"})  
cups.TurnServiceOff()  
print cups.EnabledDefault  
cups.TurnServiceOn()  
print cups.EnabledDefault
```

使用 OpenLMI 网络提供者

openlmi-networking 包为网络安装了 CIM 提供者。下面的例子说明如何使用这个 CIM 提供者列出与特定端口号相关联的 IP 地址，创建一个新的连接，配置静态 IP 地址，并激活连接。

实例：列出关联的给定端口的 IP 地址

列出关联 eth0 的所有 IP 地址使用如下代码片段：

```
device = ns.LMI_IPNetworkConnection.first_instance({'ElementName':
'eth0'})
for endpoint in
device.associators(AssocClass="LMI_NetworkSAPSAPDependency",
ResultClass="LMI_IPProtocolEndpoint"):
    if endpoint.ProtocolIFType ==
ns.LMI_IPProtocolEndpoint.ProtocolIFTypeValues.IPv4:
        print "IPv4: %s/%s" % (endpoint.IPv4Address,
endpoint.SubnetMask)
    elif endpoint.ProtocolIFType ==
ns.LMI_IPProtocolEndpoint.ProtocolIFTypeValues.IPv6:
        print "IPv6: %s/%d" % (endpoint.IPv6Address,
endpoint.IPv6SubnetPrefixLength)
```

这段代码使用了 LMI_IPProtocolEndpoint 类及相关联的 LMI_IPNetworkConnection 类。

使用以下代码，查看默认网关：

```
for rsap in
device.associators(AssocClass="LMI_NetworkRemoteAccessAvailableToElement",
ResultClass="LMI_NetworkRemoteServiceAccessPoint"):
    if rsap.AccessContext ==
ns.LMI_NetworkRemoteServiceAccessPoint.AccessContextValues.DefaultGateway:
        print "Default Gateway: %s" % rsap.AccessInfo
```

默认网关由 LMI_NetworkRemoteServiceAccessPoint 实例表示，其 AccessContext 属性等于 DefaultGateway。

获取 DNS 服务器列表，对象模式需要如下几步：

1. 通过使用 LMI_NetworkSAPSAPDependency 获取和 LMI_IPNetworkConnection 关联的 LMI_IPProtocolEndpoint 实例。
2. 对 LMI_DNSProtocolEndpoint 实例使用同样的关联。

LMI_NetworkRemoteServiceAccessPoint 实例的 AccessContext 属性等于 DNS 服务器，其关联的 LMI_NetworkRemoteAccessAvailableToElement 在 AccessInfo 属性里有 DNS 服务器地址。

有更多的可能的路径以到达 RemoteServiceAccessPath 同时入口可以被复制。使用 set () 函数可以从 DNS 服务器的列表中删除重复入口，代码如下：

```
dnsservers = set()
for ipendpoint in
device.associators(AssocClass="LMI_NetworkSAPSAPDependency",
ResultClass="LMI_IPProtocolEndpoint"):
    for dnsepoint in
ipendpoint.associators(AssocClass="LMI_NetworkSAPSAPDependency",
ResultClass="LMI_DNSProtocolEndpoint"):
        for rsap in
dnsepoint.associators(AssocClass="LMI_NetworkRemoteAccessAvailableToEle
ment", ResultClass="LMI_NetworkRemoteServiceAccessPoint"):
            if rsap.AccessContext ==
ns.LMI_NetworkRemoteServiceAccessPoint.AccessContextValues.DNSServer:
                dnsservers.add(rsap.AccessInfo)
print "DNS:", ", ".join(dnsservers)
```

实例：创建一个新的连接和配置静态 IP 地址

要为网络设备 eth0 创建一个新的配置。如：配置使用静态 IPv4 和无状态的 IPv6。可以使用以下代码实现：

```
capability = ns.LMI_IPNetworkConnectionCapabilities.first_instance({
'ElementName': 'eth0' })
result = capability.LMI_CreateIPSetting(Caption='eth0 Static',
IPv4Type=capability.LMI_CreateIPSetting.IPv4TypeValues.Static,

IPv6Type=capability.LMI_CreateIPSetting.IPv6TypeValues.Stateless)
setting = result.rparams["SettingData"].to_instance()
for settingData in
setting.associators(AssocClass="LMI_OrderedIPAssignmentComponent"):
    if setting.ProtocolIFType ==
ns.LMI_IPAssignmentSettingData.ProtocolIFTypeValues.IPv4:
        # Set static IPv4 address
        settingData.IPAddresses = ["192.168.1.100"]
        settingData.SubnetMasks = ["255.255.0.0"]
        settingData.GatewayAddresses = ["192.168.1.1"]
        settingData.push()
```

此段代码通过调用 LMI_IPNetworkConnectionCapabilities 实例中的 LMI_CreateIPSetting () 方法创建了一个新的配置，该实例是 LMI_IPNetworkConnection 通过 LMI_IPNetworkConnectionElementCapabilities 的关联实例。使用 push()方法来修改配置。

实例：激活一个连接

要使网络接口使用新的配置，调用 LMI_IPConfigurationService 类的 ApplySettingToIPNetworkConnection () 方法。此方法是异步并会返回一个作业。下面的代码片段演示如何同步调用此方法。

```
setting = ns.LMI_IPAssignmentSettingData.first_instance({ "Caption":
"eth0 Static" })
port = ns.LMI_IPNetworkConnection.first_instance({ 'ElementName':
'ens8' })
service = ns.LMI_IPConfigurationService.first_instance()
service.SyncApplySettingToIPNetworkConnection(SettingData=setting,
IPNetworkConnection=port, Mode=32768)
```


mode 参数影响配置，常用的 mode 参数的值如下所示：

- 1- 现在应用设置，使其自动激活。
- 2- 设置自动激活，现在不使用设置。
- 4- 断开并禁用自动激活。
- 5- 不改变设置状态，只禁用自动激活。
- 32768- 应用设置。
- 32767- 断开连接。

使用 OpenLMI 存储提供者

openlmi-storage 包为存储管理安装 CIM 提供者。下面的例子说明如何使用这个 CIM 提供者创建一个卷组，创建逻辑卷，建立一个文件系统，挂载文件系统，并列出系统已知的块设备。

除了 c 和 ns 变量，还可以使用以下变量：

```
MEGABYTE = 1024*1024
storage_service = ns.LMI_StorageConfigurationService.first_instance()
filesystem_service =
ns.LMI_FileSystemConfigurationService.first_instance()
```

实例：创建一个卷组

创建位于/dev/myGroup/的新的卷组，其有三个成员，默认扩展大小为 4M，使用下面的代码片段：

```
# Find the devices to add to the volume group
# (filtering the CIM_StorageExtent.instances()
# call would be faster, but this is easier to read):
sda1 = ns.CIM_StorageExtent.first_instance({"Name": "/dev/sda1"})
sdb1 = ns.CIM_StorageExtent.first_instance({"Name": "/dev/sdb1"})
sdc1 = ns.CIM_StorageExtent.first_instance({"Name": "/dev/sdc1"})

# Create a new volume group:
(ret, outparams, err) = storage_service.SyncCreateOrModifyVG(
    ElementName="myGroup",

    InExtents=[sda1, sdb1, sdc1])
vg = outparams['Pool'].to_instance()
print "VG", vg.PoolID, \
    "with extent size", vg.ExtentSize, \
    "and", vg.RemainingExtents, "free extents created."
```

实例：创建一个逻辑卷

创建两个 100M 大小的逻辑卷，可使用以下代码实现：

```
# Find the volume group:
vg = ns.LMI_VGStoragePool.first_instance({"Name":
"/dev/mapper/myGroup"})

# Create the first logical volume:
(ret, outparams, err) = storage_service.SyncCreateOrModifyLV(
    ElementName="Vol1",
    InPool=vg,
    Size=100 * MEGABYTE)
lv = outparams['TheElement'].to_instance()
print "LV", lv.DeviceID, \
    "with", lv.BlockSize * lv.NumberOfBlocks, \
    "bytes created."

# Create the second logical volume:
(ret, outparams, err) = storage_service.SyncCreateOrModifyLV(
    ElementName="Vol2",
    InPool=vg,
    Size=100 * MEGABYTE)
lv = outparams['TheElement'].to_instance()
print "LV", lv.DeviceID, \
    "with", lv.BlockSize * lv.NumberOfBlocks, \
    "bytes created."
```

实例：创建一个文件系统

使用以下代码，可以在本章创建的逻辑卷实例上，再创建一个 ext3 文件系统：

```
(ret, outparams, err) = filesystem_service.SyncLMI_CreateFileSystem(
    FileSystemType=filesystem_service.LMI_CreateFileSystem.FileSystemTypeValues.EXT3,
    InExtents=[lv])
```

实例：挂载一个文件系统

使用以下代码，可以挂载本章实例中创建的文件系统：

```
# Find the file system on the logical volume:
fs = lv.first_associator(ResultClass="LMI_LocalFileSystem")

mount_service = ns.LMI_MountConfigurationService.first_instance()
(rc, out, err) = mount_service.SyncCreateMount(
    FileSystemType='ext3',
    Mode=32768, # just mount
    FileSystem=fs,
    MountPoint='/mnt/test',
    FileSystemSpec=lv.Name)
```

实例：列出块设备

使用以下代码，可列出系统所识别的块设备：


```
devices = ns.CIM_StorageExtent.instances()
for device in devices:
    if lmi_isinstance(device, ns.CIM_Memory):
        # Memory and CPU caches are StorageExtents too, do not print
        them
        continue
    print device.classname,
    print device.DeviceID,
    print device.Name,
    print device.BlockSize*device.NumberOfBlocks
```

使用 OpenLMI 硬件提供者

openlmi-hardware 软件包为监控硬件安装的 CIM 提供者。下面的例子说明如何使用这个 CIM 提供者检索有关 CPU，内存，PCI 设备，以及制造商和型号的机器信息。

实例：查看 CPU 信息

查看 cup 的基本信息，例如名字，cpu 核数目，硬件线程数，使用下面代码：

```
cpu = ns.LMI_Processor.first_instance()
cpu_cap = cpu.associators(ResultClass="LMI_ProcessorCapabilities")[0]
print cpu.Name
print cpu_cap.NumberOfProcessorCores
print cpu_cap.NumberOfHardwareThreads
```

实例：查看内存信息

查看内存的基本信息例如内存大小，使用如下代码：

```
mem = ns.LMI_Memory.first_instance()
for i in mem.associators(ResultClass="LMI_PhysicalMemory"):
    print i.Name
```

实例：查看机箱信息

显示机箱的基本信息例如制造商或型号，使用如下代码：

```
chassis = ns.LMI_Chassis.first_instance()
print chassis.Manufacturer
print chassis.Model
```

实例：列出 PCI 设备

列出系统识别的 PCI 设备，使用如下代码：

```
for pci in ns.LMI_PCIDevice.instances():
    print pci.Name
```

5.2.5 使用 OpenLMI 脚本

LMIShell 解释器建立在 Python 模块上，可以用来开发自定义管理工具。OpenLMI 脚本项目提供了大量的 Python 库与 OpenLMI 提供商接口。此外，它和 lmi 一同发布，lmi 是一个可扩展工具，可以被用来与来自命令行的库交互。

在系统上安装 OpenLMI 脚本，在 shell 下输入如下命令：

```
easy_install --user openlmi-scripts
```

此命令在 ~/.local/ 目录下安装 Python 模块和 lmi 工具。为了扩展 lmi 工具的功能，安装附加的 OpenLMI 模块，使用如下命令：

```
easy_install --user package_name
```

完整可用模块的列表请参考 Python 官方网站，更多关于 OpenLMI 脚本的信息参考 OpenLMI 官方脚本文档。

5.3 查看和管理日志文件

日志文件是包含系统信息的文件，包括内核，服务及其上运行的应用程序。不同的日志文件包含不同的信息，例如，有一种默认系统日志文件，一种只是用于安全信息的日志文件，还有用于后台任务的日志文件。

当试图解决系统问题的时候，如试图载入内核驱动程序或寻找未授权的登录尝试系统时，日志文件是非常有用的。本章讨论在哪里可以找到日志文件，如何查看日志文件，以及在日志文件中查看什么。

一些日志文件被一个名为 rsyslogd 的守护进程控制。该 rsyslogd 守护进程是替代 syslogd 的一种增强型进程，并提供了扩展的过滤器，消息加密保护，各种配置选项，输入输出模块，通过 TCP 或 UDP 协议进行传输。需要注意的是 rsyslog 与 syslogd 是兼容的。

日志文件还可以由 journald 守护进程进行管理，该守护进程是 systemd 的组件。journald 守护进程捕获系统日志消息，内核日志消息，初始 RAM 磁盘和早期启动消息和写到标准输出的消息，以及所有服务的标准错误输出，把这些消息进行索引，并提供给用户。本地日志文件格式，它是一种结构化和索引二进制文件，改进了搜索，并提供更快的操作，而且它也存储像时间戳或用户 ID 的元数

据信息。由 `journald` 生成的日志文件在默认情况下不是一直存在的，日志文件只保存在内存或 `/run/log/journal/` 目录下的环形缓冲区。记录的数据量取决于可用内存，当你达到容量极限时，最早的记录将被删除。但是，设置可以被改变 - 请参阅 5.6.5 使能持续存储。有关期刊详细信息，请参阅 5.6 使用日志。

默认情况下，这两个记录工具并存在您的系统。`journald` 守护进程是解决问题的主要工具。它也提供了必要的用于创建结构化日志消息的附加数据。通过 `journald` 获取的数据被转发到 `/run/systemd/journal/syslog` 套接字，可以由 `rsyslogd` 进一步处理数据。然而，`rsyslogd` 实际整合默认通过 `imjournal` 输入模块，从而避免了上述的套接字。你还可以使用 `omjournal` 模块从 `rsyslogd` 到 `journald` 在相反方向上传输数据。参见 5.3.7 Syslogd 服务和日志的交互。集成使基于文本的日志文件中一致的格式可以用来维护，以确保可能的应用程序或配置依赖于 `rsyslogd` 的兼容性。另外，你可以用一致结构化的格式维护 `rsyslog` 日志。（参见 5.4 Syslogd 日志结构）

5.3.1 日志文件的位置

许多由 `rsyslogd` 维护的日志文件在 `/etc/rsyslog.conf` 配置文件里记录。大多数的日志文件在 `/var/log/` 目录下。一些应用程序例如 `httpd` 和 `samba` 在 `/var/log/` 目录下有自己的日志文件。

你可能会注意到在 `/var/log` 目录下有多个日志文件其名字后的数字不同（例如，`cron-20100906`）。这些数字代表添加轮替日志文件里的时间戳。日志文件被轮替，所以其大小不会太大。`Logrotate` 包包括了一个后台任务，它可以根据 `/etc/logrotate.conf` 配置来自动轮替日志文件，该配置文件在 `/etc/logrotate.d/` 目录下。

5.3.2 Rsyslog 的基本配置

`Rsyslog` 的主要配置文件是 `/etc/rsyslog.conf`。在这里，你可以指定全局指令，模块和过滤器的规则和动作部分。此外，您还可以以文本形式评论添加解释（以 `#` 号开头）。

过滤器

规则是由过滤器的一部分，它选择系统日志消息的一个子集，以及行动的一部分，它指定对选中的消息做什么。要在你的 `/etc/rsyslog.conf` 的配置文件定义规

则，在一行同时定义过滤器和动作，用一个或多个空格或制表符分隔。

根据所选属性，rsyslog 现在提供了不同的方式过滤系统日志消息。可用的过滤方法，可分为设施/基于优先级的，基于属性的，和基于正则表达式的过滤器。

设施/基于优先级的过滤器

最常用的和众所周知的方式来过滤系统日志消息是使用基于设施/基于优先级的过滤器，其过滤 syslog 消息基于两个条件：由点号分隔设施和优先级。要创建一个选择器，请使用以下语法

FACILITY.PRIORITY

➤ **FACILITY** 指定一个特定的子系统来生成日志消息。例如，邮件子系统处理所有与邮件相关的系统日志消息。**FACILITY** 可以由以下关键字之一（或由数字代码）来表示：kern (0), user (1), mail (2), daemon (3), auth (4), syslog (5), lpr (6), news (7), uucp (8), cron (9), authpriv (10), ftp (11), 和 local0 through local7 (16 - 23).

➤ **PRIORITY** 指定系统日志消息的优先级。**PRIORITY** 可以由以下关键字之一（或由数字代码）来表示：debug (7), info (6), notice (5), warning (4), err (3), crit (2), alert (1), 和 emerg (0).

上述语法根据定义或更高优先级选择系统日志消息。通过比较等号(=)两边的优先级，您指定的优先级的系统日志消息将被选中，所有其他优先级将被忽略。相反，优先关键字之前有一个(!)，选择除了该优先级的所有系统日志消息。

除上述指定的关键字，您也可以使用星号(*)来定义所有设施或优先级（这取决于你在那里放置星号，逗号之前或之后）。没有给出优先级设备的优先级指定其关键字为 none。设施和优先条件是不区分大小写。

要定义多个设施和优先级，用逗号(,)将它们分开。要在一行中定义多个选择，用分号(;)分隔它们。注意，在选择器在其字段能够覆盖前面的部分，它可以排除来自模式一些优先次序。

实例：设施/基于优先级的过滤器

以下是可在/etc/rsyslog.conf 中指定设施/基于优先级的过滤器的几个简单例子。要选择所有优先级的所有内核系统日志消息，添加下面的文字到配置文件：

```
kern.*
```

选择所有的邮件系统日志消息，使用优先级为 crit 或更高，使用如下形式：

```
mail.crit
```

选择所有的 cron 日志消息除了优先级为 info 和 debug 的。使用如下格式：

```
cron.!info,!debug
```

基于属性的过滤器

基于属性的过滤器可让您过滤系统日志消息的任何属性，如 timegenerated 或 syslogtag。有关属性的更多信息。请参考模板章节。您可以将每个指定属性和值比较，这些值使用表格 5-6 基于属性的比较操作 列出的比较操作。属性名和比较操作都是区分大小写的。

基于属性的过滤器以冒号（:）开始，定义该过滤器，使用如下语法：

```
:PROPERTY, [!]COMPARE_OPERATION, "STRING"
```

- PROPERTY 定义需要过滤的属性。
- 可选的感叹号（!）反向输出比较操作。基于属性的过滤器目前不支持其他布尔运算符。
- COMPARE_OPERATION 属性定义了比较操作，参考表格 5-6 基于属性的比较操作基于属性的比较操作 。
- 字符串属性指定比较值，其由属性的文字字符串提供。该值必须用引号括起来。为了使用某些字符的字符串中（例如一个引号（"）），用反斜杠字符（\）。

表格 5-6 基于属性的比较操作

Compare-operation	description
Contains	检查提供的字符串是否有任何

	部分和所提供的文本属性字符串相匹配。要执行大小写敏感的比较，使用 <code>contains_i</code> 。
<code>isequal</code>	比较提供的字符串和文本提供的属性字符串。这两个值必须匹配。
<code>startswith</code>	检查提供字符串和文本的属性字符串开头的匹配。要执行不区分大小写的比较，使用 <code>startswith_i</code> 。
<code>regex</code>	比较提供的 POSIX BRE（基本正则表达式）和文本所提供的属性字符串。
<code>ereregex</code>	比较提供的 POSIX ERE（扩展正则表达式）和文本所提供的属性字符串。
<code>isempty</code>	检查该属性是否为空。值被丢弃。在使用归一化的数据时，某一些字段可能被填充，这个操作则特别有用。

实例：基于属性的过滤器

以下是可在 `/etc/rsyslog.conf` 指定基于属性的过滤器的几个例子。在消息文本里要选择包含 `error` 字符串的日志，如下所示：

```
:msg, contains, "error"
```

下面的过滤器从主机名为 `host1` 条件选择日志消息。

```
:hostname, isequal, "host1"
```

选择的日志消息不包含 `fatal` 和 `error` 已经其之间的文本。

```
:msg, !regex, "fatal .* error"
```

基于表达式的过滤器

基于表达式过滤器根据定义的算术，布尔或字符串操作选择系统日志消息。基于表达式过滤器使用 rsyslog 自己的脚本语言调用 RainerScript 脚本，可以构建复杂的过滤器。

基本的基于表达式的过滤器使用如下：

if EXPRESSION then ACTION else ACTION

➤ **EXPRESSION** 属性表示要计算的表达式。例如：下面的表达式 `$msg startswi th ' DEVNAME' or $syslogfacility-text == ' local0'`你可以定义多个表达式，使用 **and** 或者 **or** 操作符。

➤ **ACTION** 属性代表如果表达式返回为真要执行的动作。这可以是一个单一的动作，或包含在大括号的任意复杂的脚本。

➤ 在一个新行的开始，基于表达式过滤器由关键字 **if** 指示。关键字 **then** 分开 **EXPRESSION** 和 **ACTION**。或者，也可以用关键字 **else** 来指定哪些动作是如果条件得不到满足将要执行。

基于表达式的过滤器，可以使用大括号嵌套使用条件就像例如：基于表达式的过滤器。该脚本可以让你在表达式中使用设备/基于优先级的过滤器。另一方面，基于属性的过滤器，不建议在这里。RainerScript 支持特定函数 `re_match()` 和 `re_extract()` 的正则表达式。

实例：基于表达式的过滤器

下面的表达式包含两个嵌套条件。`prog1` 程序创建的日志文件被分成基于消息中的“test”字符串的两个文件。

```
if $programname == 'prog1' then {
  action(type="omfile" file="/var/log/prog1.log")
  if $msg contains 'test' then
    action(type="omfile" file="/var/log/prog1test.log")
  else
    action(type="omfile" file="/var/log/prog1notest.log")
}
```

更多的关于基于表达式的过滤器参考在线文档。RainerScript 脚本是 rsyslog 新配置格式的基础，请参考 5.3.3 使用新的配置格式。

行为

行为指定对已经过滤的消息做什么。下面是一些可以在你的规则中定义的操作

作：

存储 syslog 消息到日志文件

主要行为指明 syslog 消息被保存哪个日志文件。这是由您指定文件路径完成。

FILTER PATH

FILTER 代表着用户选择的目标文件的路径。例如，下面的规则会选择所有的 **corn** syslog 消息然后将其存储到 **/var/log/cron.log** 文件中。

cron.* /var/log/cron.log

默认情况下，日志文件每次生成一个系统日志消息的时间同步。使用破折号标记（ - ）作为文件路径的前缀指定省略同步：

FILTER -PATH

请注意，你可能会失去信息，如果系统终止发生在写之前。但是，这种设置可以提高性能，特别是如果你运行的程序会产生非常详细的日志信息。

您指定的文件路径可以是静态或动态的。静态文件由一个固定文件路径，正如上所示的例子中表示的。动态文件路径根据所接收的消息是不同的。动态文件路径是由一个问号（ ? ）的前缀代表表示：

FILTER ?DynamicFile

DynamicFile 是一个名称。相当于预定义的模板其用于修改输出路径。您可以使用破折号做前缀（ - ）来禁用同步，你也可以使用一个冒号分隔的多个模板（ ; ）。有关模板的详细信息，请参见 5.3.2.3 中有关生成动态文件名的章节。

当你使用的是 **X Window** 系统的时候，如果您指定的文件是存在的终端或 **/dev/ console** 设备，系统日志消息发送到标准输出（使用特殊的终端处理）或控制台（使用特殊的 **/dev/ console** 的-处理）。

通过网络发生syslog消息

Rsyslog 允许您可以通过网络发送和接收系统日志消息。此功能允许您在一台机器管理多个主机的系统日志消息。要转发 **syslog** 消息到远程计算机，请使用以下语法：


```
@ [(zNUMBER)]HOST:[PORT]
```

- at 符号（@）表示 syslog 消息被转发到使用 UDP 协议的主机。
要使用 TCP 协议，使用两个 at 符号与他们之间没有空格（@@）。
- 可选项 zNUMBER 设置对系统日志消息的 zlib 压缩。该 NUMBER 属性指定的压缩级别（从 1 - 最低到 9 - 最大）。压缩增益自动由 rsyslogd 检查，如果有任何压缩增益消息会被压缩，如果消息小于 60 字节则不会被压缩。
- HOST 属性定义哪个主机选择 syslog 消息。
- PORT 属性定义主机端口。
- 当主机定义 IPV6 地址，用方括号括地址（[]）。

实例：通过网络发送 syslog 消息

以下是该通过网络（请注意，所有操作都前面带有一个选择器，其选择任何优先级的所有消息）转发系统日志信息的操作的一些例子。要通过 UDP 协议，将消息转发到 192.168.0.1。

```
*.* @ 192.168.0.1
```

使用 6514 端口和 TCP 协议将消息转发到 example.com:

```
*.* @ @ example.com:6514
```

下面压缩信息以 zlib（9 级压缩），并将其转发给[2001:db8::1],传输使用 UDP 协议。

```
*.* @ (z9)[2001:db8::1]
```

输出通道

输出通道主要用于指定日志文件可以增长到的最大大小。这对日志文件轮替非常有用的（有关详细信息，请参见 5.3.2.5 日志轮替）。输出通道基本上是关于输出操作的信息的集合。输出通道由 \$outchannel 指令定义。要定义 /etc/rsyslog.conf 输出通道，请使用以下语法：

```
$outchannel NAME, FILE_NAME, MAX_SIZE, ACTION
```

- NAME 属性指定了输出通道的名称。
- FILE_NAME 属性指定了输出文件名。输出通道只能写入到文件中，没有管道，终端，或其他类型的输出。
- MAX_SIZE 属性指定了文件可以增长的最大大小，以字节为单位。
- ACTION 属性定义当文件到最大大小的行为。
- 要使用定义的通道作为规则里的一个行为，如下：

```
FILTER :omfile:$NAME
```

实例：输出通道日志轮替

下面通过使用一个输出通道显示了一个简单的日志轮替。输出通道经由 \$ outchannel 指令定义：

```
$outchannel log_rotation, /var/log/test_log.log, 104857600,  
/home/joe/log_rotation_script
```

使用规则选择所有优先级的所有 syslog 消息，获取消息后执行预先定义的输出通道。

```
*,* :omfile:$log_rotation
```

一旦限制（在本例中 100 MB）被达到， /home/joe/ log_rotation_script 会被执行。该脚本可以包含任何从文件移动到其他文件夹内容，编辑具体的内容，或者干脆删除它。

发送 syslog 消息给特定的用户

rsyslog 通过指定用户名将 syslog 消息发送到指定的用户（如本章实例：指定多行为）。要指定多个用户，用逗号（,）分隔每个用户名。将消息发送给当前登录的所有用户，使用星号（*）。

执行一个程序

rsyslog 可以为选定的系统日志消息执行程序，并使用 system（）调用在 shell 中执行的程序。要指定一个程序来执行，用（^）字符前缀该命令。因此，指定一个模板其接收的消息的格式，并将其传递到指定的可执行文件作为一个行参数

(模板的更多信息, 请参见 5.3.2.3 模板)。

```
FILTER ^EXECUTABLE; TEMPLATE
```

这里过滤器状态的输出通过由 EXECUTABLE 指定的程序进行处理。这个程序可以是任何有效的可执行文件。用格式模板的名称替换 TEMPLATE。

实例：执行程序

在下面的例子中, 被选择的任何优先级的任何系统日志消息, 其格式经过与 template 模板匹配并作为参数传递给 test-program 程序。

```
*.* ^test-program;template
```

警告：

从任何主机接收消息时以及使用 shell 执行操作, 可能会受到命令注入。攻击者可以尝试将命令注入到您指定的行为要执行的程序。为了避免任何可能的安全威胁, 充分考虑使用的 shell 执行行为。

存储 syslog 消息到数据库

通过使用数据库写操作, 被选择的系统日志消息可以直接写入到数据库表。数据库写入使用的语法如下:

```
:PLUGIN:DB_HOST,DB_NAME,DB_USER,DB_PASSWORD:[TEMPLATE]
```

- PLUGIN 调用特定的插件, 这些插件可以处理数据库写操作。(例如, ommysql 插件)。
- DB_HOST 属性指定数据库的主机名。
- DB_NAME 属性指定数据库名。
- DB_USER 属性指定数据库用户。
- DB_PASSWORD 属性指定数据库用户的密码。
- TEMPLATE 属性指定一个修改 syslog 信息的模板。关于模板的详细信息参考模板。

重要：

目前, rsyslog 只支持 MySQL 和 PostgreSQL 数据库。为了使用 MySQL 和 PostgreSQL 数据库写入功能, 安装 rsyslog-mysql 和 rsyslog-pgsql 包。此外, 请

确保你在你的/etc/rsyslog.conf 配置文件加载相应的模块：

```
$ModLoad ommysql # Output module for MySQL support
$ModLoad ompgsql # Output module for PostgreSQL support
```

更多关于 rsyslog 的信息参考 5.3.6 使用 Rsyslog 模块。

另外，您也可以使用由 `omlibdb` 模块提供通用的数据库接口（支持：Firebird/Interbase, MS SQL, Sybase, SQLite, Ingres, Oracle, mSQL）。

丢弃 syslog 消息

想丢弃选择好的消息使用波形符（~）。

```
FILTER ~
```

丢弃行为主要是用来在进行任何进一步的处理之前，以筛选出的消息。如果你想省略一些重复的消息，这是非常有效的，否则重复消息将填充日志文件。丢弃操作的结果取决于所在的配置文件中的指定配置的位置，为得到最好的结果把这些行为放到行为列表的前面。请注意，一旦消息被丢弃在后面的配置文件中的行则没有办法来检索它。

例如，下面的规则丢弃了任何 `cron` 的 syslog 消息。

```
cron.* ~
```

定义多行为

对于每一个选择，你被允许指定多个行为。对一个选择要指定多个操作，每一个行为写在单独一行，并用符号它前面（&）字符：

```
FILTER ACTION
& ACTION
& ACTION
```

指定多个操作提高了所希望结果的整体性能，因此选择器需要被重新评估。

实例：指定多行为

在下面的例子中，所有的优先级为 `crit` 的内核 syslog 信息被发送到用户 `user1`，由模板 `temp` 处理通过后由 `test-program` 程序执行，并通过 `UDP` 协议转发到 `192.168.0.1`。

```
kern.=crit user1
& ^test-program;temp
& @ 192.168.0.1
```

任何行为后可以加模板，其用来格式化消息。要指定一个模板，以分号(;)隔开并指定模板的名称。有关模板的详细信息参考模板。

警告：

模板必须在其被用于一个行为前被定义，否则定义将被忽略。换句话说，模板定义应该总是先于/etc/rsyslog.conf 的规则定义。

模板

由 rsyslog 生成的任何输出可根据你的需求使用的模板进行修改和格式化。要创建一个模板，请在/etc/rsyslog.conf 文件使用以下语法：

```
$template TEMPLATE_NAME,"text %PROPERTY% more text",
[OPTION]
```

- \$template 是模板指令，指示后面的文本定义了一个模板。
- TEMPLATE_NAME 是模板名。
- 两个引号之间的任何内容 (“...”), 是真正的模板文本。在本文中，特殊字符可以被使用，如 \n 表示新行或 \r 表示回车。其他字符，比如 % 或者 “，如果你想使用这些字符必须进行转义。
- 两个百分比符号 (%) 之间指定的文本属性，通过此属性您可以访问系统日志消息的具体内容。有关属性的更多信息参考模板有关属性的章节。
- OPTION 属性指定修改模板功能的任何选项。当前支持的模板选项是 sql 和 stdsql，它作为一个 sql 查询被用于文本格式化。

注意：

需要注意的是数据库会写入器检查 sql 或 stdsql 选项，无论其是否在模板中被指定。如果不是，该数据库写入器不执行任何动作。这是为了防止任何可能的安全威胁，如 SQL 注入。

查看在数据库中存储 syslog 消息，更多信息参考行为章节。

生成动态文件名

模板可用于生成动态文件名。通过指定一个属性作为文件路径的一部分，一个新的文件将为每个唯一的属性而创建，这是一种方便的方式来 syslog 消息分类。

例如，使用 `timegenerated` 属性，这个属性是从消息中提取的时间戳。它将为每一个 syslog 消息生成一个唯一的文件名。

```
$template DynamicFile, "/var/log/test_logs/%timegenerated%-test.log"
```

请记住，`$template` 指令仅指定的模板。您必须在规则里使用，使其生效。在 `/etc/rsyslog.conf` 文件里，使用问号（`?`）的操作标识了动态文件名模板。

```
*.* ?DynamicFile
```

属性

模板中定义的（两个百分比符号之间（`%`））属性通过使用属性替代可以访问系统日志消息的各种内容。要定义一个模板内的属性（两个引号（之间的“`...`”）），请使用以下语法：

```
%PROPERTY_NAME[:FROM_CHAR:TO_CHAR:OPTION]%
```

➤ **PROPERTY_NAME** 属性指定了属性名。可用属性列表和详细的描述信息可以参考 `rsyslog.conf(5)` man 手册的 **Available Properties** 章节。

➤ **FROM_CHAR** 和 **TO_CHAR** 属性表示一个字符的范围用以限定其行为。另外，正则表达式可以用来指定一个字符范围。要做到这一点，请设置字母 **R** 作为的 **FROM_CHAR** 属性，并指定您的所需的正则表达式作为的 **TO_CHAR** 属性。

➤ **OPTION** 属性指定的任何属性选项，如 **lowercase** 选项将输入转换为小写。所有可用的属性选项及其详细描述您可在 `rsyslog.conf (5)` 手册页 **Property** 选项章节找到。

下面是简单属性的一些例子：

➤ 下面的属性获取系统日志消息的全部消息文本。

```
%msg%
```

➤ 下面的属性获取系统日志消息的整个消息文本和下降其最一行

的换行符。

```
%msg:::drop-last-lf%
```

➤ 下面的属性获取时间戳的前 10 个字符，该时间戳是接收系统日志消息时产生的，生成格式是遵循 RFC3999 日期标准的。

```
%timegenerated:1:10:date-rfc3339%
```

模板的例子

这章介绍了 rsyslog 模板的例子。

在本章实例“一个详细的 syslog 消息模板”中，给出了格式化系统日志消息的模板，所以其输出消息的重要性，设施，接收消息时的时间戳，主机名，消息标签，消息文本，并用一个新的行结束。

实例：一个详细的 syslog 消息模板

```
$template verbose, "%syslogseverity%, %syslogfacility%,  
%timegenerated%, %HOSTNAME%, %syslogtag%, %msg%\n"
```

在本章实例“一个墙体消息模板”中，给出了一个类似于传统墙体消息（一条消息发送到已登录系统的所有用户，它们拥有自己的 mesg（1）权限来设置为 yes）的模板。此模板在一个新行（使用 \r 和 \n 和 \7）输出的消息文本及其主机名，消息标记和时间戳。

实例：一个墙体消息模板”

```
$template wallmsg, "\r\n\7Message from syslogd@ %HOSTNAME% at  
%timegenerated% ... \r\n %syslogtag% %msg%\n\r"
```

在本章实例“一个数据库模式的消息模板”中，给出了格式化系统日志消息的模板，该模板可被用作数据库查询。注意在模板的末尾使用 sql 选项。此格式告诉数据库写入器将消息格式为 MySQL 的 sql 查询。

```
$template dbFormat, "insert into SystemEvents (Message, Facility,  
FromHost, Priority, DeviceReportedTime, ReceivedAt, InfoUnitID,  
SysLogTag) values ('%msg%', %syslogfacility%, '%HOSTNAME%',  
%syslogpriority%, '%timereported:::date-mysql%',  
'%timegenerated:::date-mysql%', %iut%, '%syslogtag%')", sql
```

rsyslog 现在还包含了一组可以根据 RSYSLOG_前缀识别的预定义模板。这个前缀是保留给系统日志的使用，最好创建模板时不要使用这个前缀，以避免

冲突。下面的列表显示了这些预定义模板以及它们的定义：

RSYSLOG_DebugFormat

用于解决属性问题的特殊格式。

```
"Debug line with all properties:\nFROMHOST: '%FROMHOST%',
fromhost-ip: '%fromhost-ip%', HOSTNAME: '%HOSTNAME%', PRI:
%PRI%,\nsyslogtag '%syslogtag%', programname: '%programname%',
APP-NAME: '%APP-NAME%', PROCID: '%PROCID%', MSGID:
'%MSGID%',\nTIMESTAMP: '%TIMESTAMP%',
STRUCTURED-DATA:
'%STRUCTURED-DATA%',\nmsg: '%msg%'\nescaped msg:
'%msg:::dropcc%'\n
nrawmsg: '%rawmsg%'\n\n"
```

RSYSLOG_SyslogProtocol23Format

格式在 IETF 的 Internet 草案 ietf-syslog-protocol-23 中指定，其假定成为新的 RFC 规定的系统日志标准。

```
"%PRI%1 %TIMESTAMP:::date-rfc3339% %HOSTNAME% %APP-NA
ME% %PROCID%
%MSGID% %STRUCTURED-DATA% %msg%\n"
```

RSYSLOG_FileFormat

现代风格的日志文件格式类似 TraditionalFileFormat，但具有高精度的时间戳和时区信息。

```
"%TIMESTAMP:::date-rfc3339% %HOSTNAME% %syslogtag% %msg:::
sp-if-no-
1st-sp% %msg:::drop-last-lf%\n"
```

RSYSLOG_TraditionalFileFormat

旧的默认日志文件格式使用低精度的时间戳。

```
"%TIMESTAMP% %HOSTNAME% %syslogtag% %msg:::sp-if-no-1st-
p% %
msg:::drop-last-lf%\n"
```

RSYSLOG_ForwardFormat

一个转发格式具有高精度的时间戳和时区信息。

```
"%PRI%%TIMESTAMP:::date-rfc3339% %HOSTNAME%
%syslogtag:1:32% %msg:::sp-if-no-1st-sp% %msg%"
```

RSYSLOG_TraditionalForwardFormat

用低精度的时间戳传统的转发格式：


```
"%PRI%%TIMESTAMP% %HOSTNAME% %syslogtag:1:32%%msg::s
p-if-no-1stsp%%
msg%\\"
```

全局指令

全局指令是适用于 rsyslogd 守护进程的配置选项。他们通常会指定特定预定义变量的值，该值会影响 rsyslogd 守护进程的行为或遵循的规则。所有全局指令必须以美元符号 (\$)。只有一个指令是每行指定。下面是一个全局指令的例子，其指定系统日志消息队列的最大大小：

```
$MainMsgQueueSize 50000
```

这个指令默认大小是 10000 个消息，它可以被上面的例子重新不同的数字。

您可以在/etc/rsyslog.conf 配置文件中定义多个指令。一个指令影响的所有配置选项的行为，直到同一个指令的另一事件被检测到。全局指令可以用来配置行为，队列和调试。所有可用的配置指令的完整列表可以参考在线文档。目前，一种新的配置格式已经开发出其可以替代\$基础的语法(5.3.3 使用新的配置格式)。然而，经典的全局指令仍然为旧格式的支持。

日志轮替

下面是一个简单的/etc/logrotate.conf 配置文件例子：

```
# rotate log files weekly
weekly
# keep 4 weeks worth of backlogs
rotate 4
# uncomment this if you want your log files compressed
compress
```

示例配置文件中所有的行的定义全局选项，其适用于每一个日志文件。在我们的例子，日志文件每周轮替，轮替的日志文件保留四个星期，所有的轮替日志文件由 gzip 压缩的成.gz 的格式。以#号开头的任何行是注释，不会被处理。

您可以为特定的日志文件定义配置选项，并将其放置在全局选项的下面。然而，最好是在/etc/logrotate.d/directory 目录下为特定的日志文件创建单独的配置文件，并在配置文件里配置选项。

下面是一个在/etc/logrotate.d/目录下配置文件的例子：

```
/var/log/messages {  
rotate 5  
weekly  
postrotate  
/usr/bin/killall -HUP syslogd  
endscript  
}
```

该文件中的配置选项只为/var/log/messages 日志文件配置。在可能的情况下，在此指定的设置将覆盖全局设置。因此，轮替的/var/log/messages 日志文件将保留五周，而不是全局选项定义定义的四周。

下面是一些指令的列表，你可以在你的日志轮替配置文件里修改。

➤ **weekly-** 指定日志轮替的周期为每周，类似的命令还包括如下：

daily

monthly

yearly

➤ **compress-**使能轮替文件的压缩功能，类似的命令还包括如下：

nocompress

compresscmd – 指定用于压缩的命令。

uncompresscmd

compressext – 指定用于压缩的扩展。

compressoptions – 指定用于传给程序的压缩选项。

delaycompress – 退出压缩直到下一个轮替周期。

➤ **rotate INTEGER** -指定了一个日志文件被轮替的最大数目，超过数目的日志文件被删除或邮寄到一个特定的地址。如果指定值 0，旧的日志文件被删除，而不会轮替。

➤ **mail ADDRESS** -此选项，已轮替多次的日志文件的邮件地址。

类似的指令包括：

nomail

maifirst-指定只是轮替的日志文件被邮寄，到期的日志文件不被邮寄。

Maillast- 指定到期的日志文件被邮寄，轮替的日志文件不邮寄。当

邮寄功能打开时，此选项是默认选项。

关于个配置选项的完整的指令列表参考 `logrotata(5)` 手册页。

5.3.3 使用新的配置格式

中标麒麟高级服务器操作系统 V7 安装的 `rsyslog` 包默认情况下是第 7 版，介绍其新的配置语法。这种新的配置格式的目标是更强大，更直观，通过不允许某些无效的结构防止常见的错误。语法增强的使能是通过 `RainerScript` 脚本配置处理器完成。遗留格式仍然得到支持，它默认在 `/etc/rsyslog.conf` 配置文件中使用。

`RainerScript` 是一种脚本语言，旨在用于处理网络事件和配置事件处理器，例如 `rsyslog`。`RainerScript` 最早是用来定义基于表达式的过滤器，见 5.3.2.1 中有关基于表达式过滤器的章节。`RainerScript` 在 `rsyslog 7` 的版本中实现了 `input()` 和 `ruleset()`，其允许所述 `/etc/rsyslog.conf` 配置文件被写入新的语法。新的语法不同之处主要在于，它是更结构化的;参数作为参数传递给语句，如输入，行为，模板，模块加载。选项的范围由块限制。这增强可读性，并降低所造成由错误配置的错误的数量。还有一个显著的性能优势。有些功能在新旧语法都可以用，有的只在新语法可以用。

与旧风格的参数配置比较：

```
$InputFileName /tmp/inputfile
$InputFileTag tag1:
$InputFileStateFile inputfile-state
$InputRunFileMonitor
```

同样的配置使用新的配置语句如下：

```
input(type="imfile" file="/tmp/inputfile" tag="tag1:"
statefile="inputfile-state")
```

这显著减少在配置中使用的参数的数目，提高了可读性，并且还提供了更高的运行速度。有关 `RainerScript` 语句和参数的详细信息，请参见在线文档。

规则集

除特殊指令之外，`rsyslog` 处理的消息由规则定义。规则由过滤条件和如果条件为真要执行的操作组成。在 `/etc/rsyslog.conf` 文件中的旧规则，所有的规则以每个输入消息的出场顺序来评估。此过程开始于第一规则，并继续，直到所有的规则都已经被处理或直到消息被其中某一规则丢弃。

然而，规则可被分成不同序列称为规则集。在规则集中，你可以限制某些规则的影响，只选择输入或通过定义一组绑定到特定的输入的行为提升 rsyslog 的性能。换句话说，某些类型的消息不可避免的被评估为假的，类似这样的过滤条件可以被跳过。在/etc/rsyslog.conf 中旧规则集定义如下所示：

```
$RuleSet rulesetname
rule
rule2
```

当另一个规则被定义，上一个规则就结束了，或者默认规则集被调用，如下：

```
$RuleSet RSYSLOG_DefaultRuleset
```

rsyslog 7 的新的配置格式，input () 和 ruleset () 语句执行被保留。新格式的规则集在/etc/rsyslog.conf 中定义，如下所示：

```
ruleset(name="rulesetname") {
rule
rule2
call rulesetname2
...
}
```

用你规则集的标识符替换 rulesetname。该规则集名称不能以 RSYSLOG_ 开始，因为这个名称空间是保留的，供 rsyslog 使用。RSYSLOG_DefaultRuleset 定义缺省设置的规则集，如果消息没有分配其他规则集将被执行。在上面提到的 rule 和 rule2 下，你可以定义过滤-操作的格式规则。对于调用参数，你可以嵌套规则集由内部或者规则集块调用它们。

创建规则集后，您需要指定它适用于什么输入：

```
input(type="input_type" port="port_num" ruleset="rulesetname");
```

这里可以通过 input_type 识别输入消息，这是所收集的信息，或者通过 port_num - 端口号。其他参数，如 file 或 tag 可以用于指定 input ()。用规则集的名称替换 rulesetname。万一某个输入消息未明确在规则集中指定，默认规则集会被触发。

你可以使用就的格式定义规则集，更多的信息参考在线文档。

实例：使用规则集

下面的规则集确保对于从不同端口进入的不同的远程消息进行不同处理。规则集添加到/etc/rsyslog.conf:

```
ruleset(name="remote-6514") {
    action(type="omfile" file="/var/log/remote-6514")
}
ruleset(name="remote-601") {
    cron.* action(type="omfile" file="/var/log/remote-601-cron")
    mail.* action(type="omfile" file="/var/log/remote-601-mail")
}
input(type="imtcp" port="6514" ruleset="remote-6514");
input(type="imtcp" port="601" ruleset="remote-601");
```

在上述例子显示的规则集为为从两个端口的远程输入定义日志目标地址。如果是 601 端口，消息是根据设施的进行排序。同时，TCP 输入被启用并绑定到规则集。请注意，您对于此配置工作必须加载所需的模块（imtcp）。

Sysklogd 服务的兼容性

rsyslog 第 5 版通过 -c 选项指定兼容模式，但在第 7 版不支持，同样，Sysklogd 风格的命令行选项已过时，应避免通过这些命令行选项配置 rsyslog。但是，您可以使用多个模板和指令配置 rsyslogd 来模仿像 sysklogd 似的行为。

更多关于不同的 rsyslogd 选项的信息参考 rsyslogd (8) 手册页。

5.3.4 使用 Rsyslog 队列

队列用于在 rsyslog 的组件之间传递内容，传递的主要内容是系统日志消息。在队列机制下，rsyslog 能够同时处理多个消息，并可以用多个行为对单个消息立刻反应。rsyslog 的数据流说明如下：

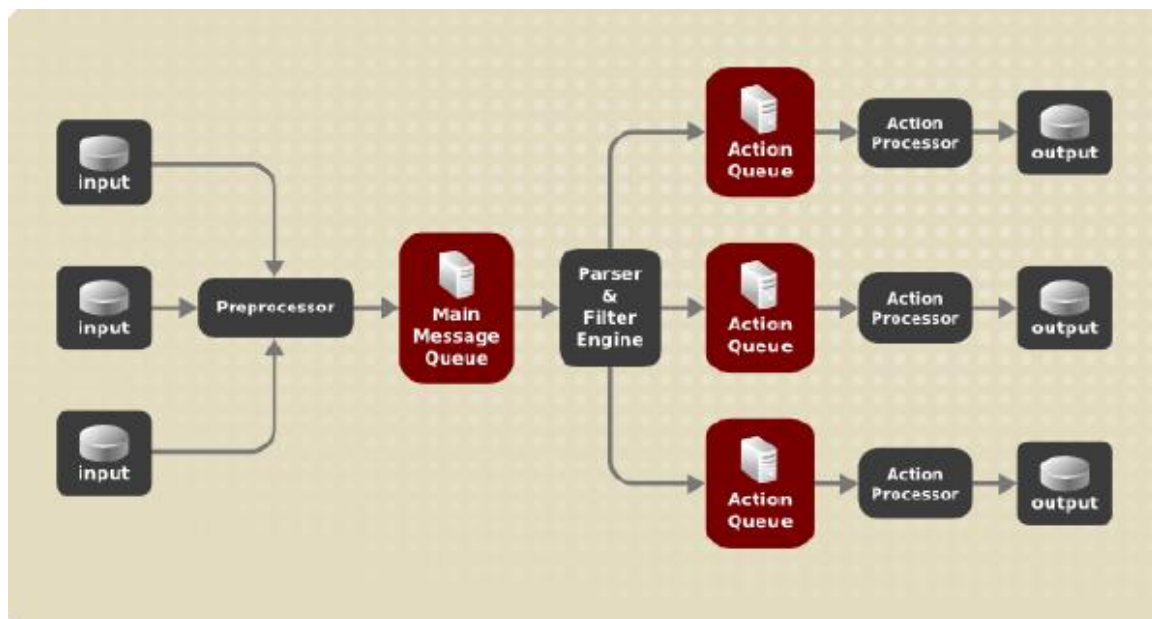


图 5-4 Rsyslog 中消息流

无论 rsyslog 什么时候收到消息，它都把这个消息给预处理，然后将其放置到主消息队列。消息等在那里待出队并传递到规则处理器。

规则处理器是一个解析和过滤引擎。在此，在/etc/rsyslog.conf 定义的规则被应用。基于这些规则，该规则处理器评估哪些操作被执行。每个行为都有它自己的执行队列。消息通过队列传递到行为处理器最终产生相应的输出。注意，在这一点上，几个行为可以同时对一个消息运行。为了这个目的，一个消息被复制并传递到多个行为处理器。

只有每个行为一个队列是可能的。根据配置的不同，该消息可以正确发送到行为处理器而不通过执行队列。这是直接队列（见下文）的行为。在输出操作失败的情况下，行为处理器通知执行队列，一段时间间隔后，行为处理器又会去取未处理 消息，再次尝试。

综上所述，rsyslog 队列有两个位置：无论是对与规则处理器作为一个主消息队列的前面或在各种类型的输出行为作为动作队列的前面。队列有两个主要优点既都会提高消息处理性能：

- 它们充当缓冲器，在 rsyslog 的结构中分离生产者和消费者。
- 它们允许并行执行消息。

除此之外，队列可以用几种不同的指令被配置来为系统提供最佳的性能。这

些配置选项在以下各节介绍。

警告：

如果输出插件无法传送一个消息，它被存储在前面的消息队列。如果在队列满时，输入阻塞，直到它不再满。这将阻止新的消息进入被阻塞的消息队列。在没有单独的执行队列的情况下，这会产生严重的后果，例如阻止 SSH 日志记录，这反过来又阻止 SSH 访问。因此，建议为那些通过网络转发到数据库的输出使用专用的行为的队列。

定义队列

根据消息在哪里存储，分为几种类型的队列：direct, in-memory, disk 和 disk-assisted,最广泛使用的队列是 in-memory。您可以选择这些类型的其中之一作为主要消息队列，也可以作为执行队列。将下面的语句添加到 / etc/ rsyslog.conf 文件：

```
$ObjectQueueType queue_type
```

在这里，你可以为主消息队列（用 MainMsg 替换 object）或执行队列（用 action 代替 object）申请设置。用 direct, linkedlist 或 fixedarray(in-memory 队列)，或 disk 更换 queue_type。

默认设置主消息队列是 FixedArray 队列,最大存储 10000 个消息。执行队列默认设置为 Direct 队列。

Direct 队列

对于许多简单的操作，例如，输出写入到本地文件，在执行前建立一个队列是不需要的。为了避免排队，使用如下：

```
$ObjectQueueType Direct
```

使用 MainMsg 或者 Action 替代 object,这个选项可以用在主消息队列或者执行消息队列。对于 direct 队列，消息直接传送，它会很快从生产者传送到消费者。

Disk 队列

disk 队列存储消息到硬盘驱动器，这使得它们高度可靠的，但也是所有可能的排队模式中最慢的。此模式可用于防止高度重要的日志数据的丢失。但是，大

多数用例中不建议使用 disk 队列。要设置 disk 队列，在/etc/rsyslog.conf 中键入以下内容：

```
$ObjectQueueType Disk
```

使用 MainMsg 或者 Action 替代 object,这个选项可以用在主消息队列或者执行消息队列。disk 队列写入部分，默认大小为 10 MB。此默认大小可以通过下面的配置指令进行修改：

```
$ObjectQueueMaxFileSize size
```

其中，size 表示 disk 队列的大小。定义的大小限制不是强制性的，rsyslog 总是写入一个完整的队列条目，即使违反的大小限制。disk 队列的每个部分有单独的文件相匹配。这些文件的命名指令如下所示：

```
$ObjectQueueFilename name
```

这为要设置的文件设置了文件名前缀。其文件名以 7 个数字开始，每增加一个文件数字也增加。

In-memory 队列

在 in-memory 队列中，排队的消息被保存在内存中，这使得这个过程非常快。如果电脑重新加电或关机，则排队的数据将丢失。但是，您可以使用 \$ ActionQueueSaveOnShutdown 设置关机前保存数据。有两种类型的 in-memory 队列：

- FixedArray queue –默认的主要消息队列，最大 10,000 个元素。这种类型的队列的使用固定预分配的数组保存队列元素的指针。由于这些指针，即使队列为空也会消耗一定量的存储器。然而，FixedArray 提供了最佳的运行时性能，当你期望排队的消息相对较少和高性能，此队列是最佳的。

- LinkedList queue –这里，一个链表里所有结构都是动态分配的，因此，存储器仅在需要时分配。LinkedList 的队列处理偶尔的突发消息非常好。

在一般情况下，当使用 LinkedList 队列相比于 FixedArray 队列，它消耗更少的内存，并降低了处理开销。

配置 in-memory 队列使用下面的语法：

\$ObjectQueueType LinkedList

\$ObjectQueueType FixedArray

使用 MainMsg 或者 Action 替代 object,这个选项可以用在主消息队列或者执行消息队列。

disk-assisted in-memory 队列

disk 和 in-memory 队列都有各自的优点,rsyslog 让你可以将其合并为 disk-assisted in-memory 队列。为此,配置一个正常的 in-memory 队列然后添加 \$ObjectQueueFileName 指令可定义为磁盘援助的文件名。这个队列就变成 disk-assisted,这意味着 in-memory 队列与 disk 队列通力协作。

如果 in-memory 队列已满或者需要关机后消息不丢的 disk 队列被激活。在 disk-assisted 队列中,可以同时设置 disk-specific 或 in-memory specific 配置参数。这种类型的队列可能是最常用的,它对可能长时间运行的和不可靠的操作特别有用。

指定 disk-assisted in-memory 队列使用 so-called 水印:

\$ObjectQueueHighWatermark number

\$ObjectQueueLowWatermark number

使用 MainMsg 或者 Action 替代 object,这个选项可以用在主消息队列或者执行消息队列。使用允许入队的最大消息数替代 number。当 in-memory 队列达到了高水位定义的数字,它开始将消息写入磁盘,并一直持续到 in-memory 队列大小下降到与低水印定义的数量。正确设置水印尽量减少不必要的磁盘写操作,同时也留下了消息的内存空间,因为写入磁盘文件是相当慢的。因此,高水位必须比整个队列容量设置 \$ObjectQueueSize 低。高水位和整体队列大小之间的差是专供消息突发而备用的存储器缓冲区。在另一方面,设定高水位过低会不必要的频繁的打开磁盘援助。

实例: 可靠的转发日志消息到服务器

rsyslog 经常被用来维护一个中心的日志系统,其中,日志消息在网络上被转发到服务器。当服务器不可用时为避免信息丢失,最好是配置一个执行队列中转发行为。这样一来,发送失败的信息存储在本地,直到服务器再次可达。请注

意，当连接使用 UDP 协议时，这样的队列是不可配置的。

程序 5.1 转发到一台单独的服务器

假设任务是从系统转发日志消息到主机名为 example.com 的服务器上，并配置一个执行队列缓冲消息以防服务器中断。为此，执行以下步骤：

- 在/etc/rsyslog.d.conf 文件中使用下面的配置或者在/etc/rsyslog.d/目录下创建一个文件，文件内容如下：

```
$ActionQueueType LinkedList
$ActionQueueFileName example_fwd
$ActionResumeRetryCount -1
$ActionQueueSaveOnShutdown on
*. * @ @ example.com:6514
```

`$ActionQueueType` 使能 LinkedList in-memory 队列

`$ActionFileName` 定义一个磁盘存储，备份文件名使用 example_fwd 前缀在/var/lib/rsyslog/目录下创建。

`$ActionResumeRetryCount` 设置为 -1，当尝试连接时，如果服务器没有响应，防止 rsyslog 丢失消息。

`$ActionQueueSaveOnShutdown` 是此项，当关机时存储 in-memory 队列中的数据。

最后一行转发所有收到的消息到日志服务器，端口规格可选。

具有上述配置，如果远程服务器不可达时，rsyslog 会在内存中保留消息。只有当 rsyslog 现在用完了所配置的内存队列空间或者需要关机，磁盘文件才会被创建，这有利于提升系统的性能。

程序 5.2 转发到多台服务器

转发日志消息到多个服务器的过程类似于前面的过程：

- 每个目标服务器需要一个单独的转发规则，执行队列规则，并在磁盘上备份文件。例如，在/etc/rsyslog.conf 文件使用下面的配置，或在/etc/rsyslog.d/目录中的以下内容创建一个文件：

```
$ActionQueueType LinkedList
$ActionQueueFileName example_fwd1
$ActionResumeRetryCount -1
$ActionQueueSaveOnShutdown on
*. * @ @ example1.com

$ActionQueueType LinkedList
$ActionQueueFileName example_fwd2
$ActionResumeRetryCount -1
$ActionQueueSaveOnShutdown on
*. * @ @ example2.com
```

为 rsyslog 文件创建目录

Rsyslog 服务以 syslogd 守护进程的方式运行，并被 SELinux 管理。因此，rsyslog 需要写的所有文件，必须有相应的 SELinux 文件上下文。

程序 5.3 创建一个工作目录

1. 如果需要使用不同的目录来存储工作文件，创建一个目录，如下所示：

```
~]# mkdir /rsyslog
```

2. 安装工具来管理 SELinux 策略

```
~]# yum install policycoreutils-python
```

3. 设置 SELinux 的上下文和 /var/lib/rsyslog/ 目录上下文相同：

```
~]# semanage fcontext -a -t syslogd_var_lib_t /rsyslog
```

4. 应用 SELinux 上下文：

```
~]# restorecon -R -v /rsyslog
restorecon reset /rsyslog context
unconfined_u:object_r:default_t:s0-
>unconfined_u:object_r:syslogd_var_lib_t:s0
```

5. 如果需要，检查 SELinux 上下文，如下：

```
~]# ls -Zd /rsyslog  
  
drwxr-xr-x. root root system_u:object_r:syslogd_var_lib_t:s0  
  
/rsyslog
```

6. 按要求创建子目录。例如：

```
~]# mkdir /rsyslog/work
```

子目录将与 SELinux 有相同的上下文，并将其作为父目录创建。

7. 在配置生效之前，添加下面这行到/etc/rsyslog.conf 文件中。

```
$WorkDirectory /rsyslog/work
```

该设置将一直有效，直到在解析配置文件遇到下一个 WorkDirectory 指令。

管理队列

所有类型的队列可以进一步配置以满足您的要求。您可以使用几种不同的指令来修改执行队列和主消息队列。目前，有 20 多个队列参数可用，见在线文档。其中的一些设置是常见的，其他诸如工作线程管理，提供对队列行为多的控制，保留给高级用户使用。在高级设置中，您可以优化 rsyslog 的性能，调度排队，或修改系统关闭队列的行为。

限制队列大小

您可以限制队列可以包含消息的数量，使用如下设置：

```
$ObjectQueueHighWatermark number
```

使用 MainMsg 或者 Action 替代 object,这个选项可以用在主消息队列或者执行消息队列。用队列可以包含的消息数目替代 number。可以设置队列的大小，而不是作为它们实际存储器的大小。主要消息队列和规则集队列默认队列大小为 10000，执行队列默认大小为 1000。

Disk assisted 队列在默认情况下是无限制的，不能用指令来强制大小，但可以以字节为单位保留他们的物理磁盘空间，使用以下设置：

```
$ObjectQueueMaxDiscSpace number
```

使用 MainMsg 或者 Action 替代 object。当队列被填满时，新消息被丢弃，

直到队列释放了足够的空间。

丢弃消息

当队列消息达到一定的数量，为了节省空间，你可以在队列中丢弃不太重要的信息，保留空间给优先级更高的消息。启动丢弃处理的阈值可设定 `discard mark`:

```
$SubjectQueueDiscardMark number
```

使用 `MainMsg` 或者 `Action` 替代 `object`,这个选项可以用在主消息队列或者执行消息队列。这里，`number` 表示要启动丢弃消息进程的消息数目。定义哪些消息要被丢弃，使用如下：

```
$SubjectQueueDiscardSeverity priority
```

用下面的关键字之一（或一些）替代 `priority`: `debug` (7), `info` (6), `notice` (5), `warning` (4), `err` (3), `crit` (2), `alert` (1) 和 `emerg` (0)。使用此设置，当达到丢弃数目后，比定义优先级较低的新来的和已排队消息从队列中删除。

使用时限

可以配置 `rsyslog` 在一个特定的时间段处理队列。有了这个选项，你可以转移部分工作到非高峰时段。要定义一个时间范围，请使用以下语法：

```
$SubjectQueueDequeueTimeBegin hour
```

```
$SubjectQueueDequeueTimeEnd hour
```

你可以指定绑定时间的框架，以小时为单位。使用 24 小时制，不支持设置分。

配置工作线程

工作线程执行消息入队的指定操作。例如，在主消息队列，一个工作线程的任务将过滤器逻辑应用到每个输入消息，并将其排队到相关执行队列。当消息到达时，一个工作线程将自动启动。当消息的数量达到一定数量时，另一个工作线程被启动。要指定此数，使用如下：

```
$SubjectQueueWorkerThreadMinimumMessages number
```

用消息的数量替代 `number` 将触发补充工作线程。例如，`number` 置为 100，当 100 多个消息到达时，一个新的工作线程开始。当超过 200 个消息到达时，第

三个工作线程启动等。然而，并行运行太多的工作线程是低效的，所以你可以限制它们的最大数量，如下所示：

`$SubjectQueueWorkerThreads number`

`number` 代表可并行运行的工作线程的最大数目。对于主消息队列，默认限制为 1 个线程。一旦一个线程工作已经开始，它会一直运行直到超时出现。要设置超时时长，如下所示：

`$SubjectQueueWorkerTimeoutThreadShutdown time`

用持续时间（以毫秒为单位）替换 `time`。如果没有设置 `time`，零超时会被应用，当它没有消息可以处理是，工作线程立即终止。如果您指定的时间为-1，没有线程将被关闭。

批量出队列

为了提高性能，您可以配置 `rsyslog` 同时多队列的消息数。设定出队列消息数目的上限，则使用如下命令：

`$SubjectQueueDequeueBatchSize number`

用可被同时出队列的消息的最大数目替换 `number`。请注意，出队列消息数目较高的设置和很多的工作线程同时运行会导致较大的内存消耗高。

终止队列

当要终止的队列中仍然包含信息，您可以尝试通过指定时间间隔使工作线程来完成队列处理，以减少数据丢失：

`$SubjectQueueTimeoutShutdown time`

指定的 `time` 以毫秒为单位。如果一段时间后仍然有一些排队的消息，工作线程完成当前的数据处理，然后终止队列。因此，未处理的消息都将丢失。另一个时间间隔可以让工作线程完成最后的数据处理：

`$SubjectQueueTimeoutActionCompletion time`

在设置的 `time` 超时的情况下，任何工作线程都被关闭。在关机时要保存数据，可以使用：

`$ObjectQueueTimeoutSaveOnShutdown time`

如歌按上面那样设置了，所有队列的数据在 `rsyslogd` 终止前都会被存储到磁盘。

使用 `rsyslog` 队列新语法

`rsyslog 7` 提供了新的语法，队列对象在 `action ()` 内部定义，既可以单独使用或在 `/etc/rsyslog.conf` 中一个规则集使用。执行队列的格式如下：

```
action(type="action_type" queue.size="queue_size"
queue.type="queue_type" queue.filename="file_name")
```

用要执行的操作的模块名称替换 `action_type`，用消息队列可以容纳的最大数目替代的 `queue_size`。对于 `queue_type`，可以选择 `disk` 队列或从 `in-memory` 队列中选择一个：`direct`，`LinkedList` 的或 `fixedarray`。对于 `file_name` 仅指定一个文件名，而不是路径。请注意，如果创建一个新的目录来保存日志文件，`SELinux` 必须被设置。参见 5.3.4.2 为 `rsyslog` 文件创建目录章节的例子。

实例：定义一个执行队列

要配置异步链表基础的执行队列中的输出行为，执行队列可容纳最多 10,000 条信息，输入命令如下：

```
action(type="omfile" queue.size="10000" queue.type="linkedlist"
queue.filename="logfile")
```

`Rsyslog 7` 对于 `direct` 执行队列的新语法如下：

```
*.* action(type="omfile" file="/var/lib/rsyslog/log_file
)
```

`Rsyslog 7` 对于执行队列使用多参数的新语法如下：

```
*.* action(type="omfile"
queue.filename="log_file"
queue.type="linkedlist"
queue.size="10000"
)
```

默认的工作目录，或者最后所设置的工作目录，将被使用。如果需要使

同的目录，在执行队列之前添加一行，如下所示：

```
global(workDirectory="/directory")
```

实例：使用新语法转发到一个单独服务器

下面的例子是建立在程序 5.1 转发到一台单独的服务器基础上的，为了展示旧语法和 rsyslog 7 新语法之间的差异的。omfwd 插件是用来提供在 UDP 或 TCP 转发消息的。默认使用 UDP。该插件被内置不需要被装载。

在/etc/rsyslog.conf 中使用下面的配置，或者在/etc/rsyslog.d/目录用如下内容创建一个文件：

```
*.* action(type="omfwd"
queue.type="linkedlist"
queue.filename="example_fwd"
action.resumeRetryCount="-1"
queue.saveOnShutdown="on"
target="example.com" port="6514" protocol="tcp"
)
```

- queue.type="linkedlist"使能 LinkedList in-memory 队列。
- queue.filename 定义一个磁盘存储。创建的备份文件名使用 example_fwd 前缀，通过预定义的全局的 workDirectory 指令指定工作目录。
- action.resumeRetryCount 如果设置为-1，当服务器无反应，syslog 尝试重新连接时，不能丢掉消息。
- queue.saveOnShutdown 如果设置为 on,如果 syslog 关闭，保存 in-memory 队列中数据。
- 最后一行转发所有收到的消息到日志服务器，端口规格可选。

5.3.5 在日志服务器上配置 rsyslog

rsyslog 服务提供的设施可用用来运行日志服务器和配置单个系统来发送日志文件到日志服务器。请参见实例“可靠的转发日志消息到服务器”，查看客户端 rsyslog 配置信息。

rsyslog 服务必须安装在你打算作为日志服务器的系统上，并且将所有的系

统配置为将日志发送给日志服务器。中标麒麟高级服务器操作系统 V7 默认情况下会安装 rsyslog，如果需要，以确保系统确实安装了 rsyslog，以 root 身份输入以下命令：

```
~]# yum install rsyslog
```

Syslog 流量默认的协议和端口是 UDP 和 514，其在 / etc/services 文件中列出。然而，rsyslog 默认使用 TCP 在端口 514。在配置文件 /etc/rsyslog.conf 中，TCP 是由 @@ 声明。

其它的端口在示例有时会用到，然而 SELinux 的默认配置为仅允许发送和接收在下面列出的端口中：

```
~]# semanag eport -l | grep syslog
syslogd_port_t tcp 6514, 601
syslogd_port_t udp 514, 6514, 601
```

semanage 实用程序由 policycoreutils-Python 包的一部分提供。如果需要的话，安装包如下：

```
~]# yum install policycoreutils-python
```

此外，在默认情况下 SELinux 的 rsyslog 类型是 rsyslogd_t，其被配置为允许发送和接收类型是 rsh_port_t 的远程 shell (RSH) 端口，其 TCP 默认端口是 514。因此，这是没有必要使用 semanage 的明确的允许 TCP 端口为 514。例如，要检查 SELinux 允许什么程序在端口 514 通信，输入命令如下：

```
~]# semanag eport -l | grep 514
output omitted
rsh_port_t tcp 514
syslogd_port_t tcp 6514, 601
syslogd_port_t udp 514, 6514, 601
```

更多关于 SELinux 的信息，请参考 SELinux 用户手册。

请在你打算做日志服务器的系统上，使用下列过程中的步骤。在这些过程中的所有步骤必须以 root 身份户执行命令。

程序 5.4 配置 SELinux 允许在某一个端口生成 rsyslog 流量

如果需要，rsyslog 流量使用新端口，在日志服务器和客户端上执行此过程。

例如，在端口 10514 发送和接收 TCP 流量，步骤如下：

1.

```
~]# semanag eport -a -t syslogd_port_t -p tcp 10 514
```

2. 输入如下命令查看 SELinux 端口使用情况：

```
~]# semanage port -l | grep syslog
```

3. 如果新的端口已经配置在/etc/rsyslog.conf，重新启动 rsyslog 使更改生效：

```
~]# service rsyslog restart
```

4. 确认 rsyslog 监听的端口

```
~]# netstat -tnl p | grep rsyslog
```

更多关于 semanage port 命令的信息查看 semanage-port (8) man 手册。

程序 5.5 配置 firewalld

配置 firewalld 允许 rsyslog 流量进入。例如，允许 TCP 流量在端口 10514 通过，步骤如下：

1.

```
~]# firewall -cmd --zone= zone --add-port= 10 514/tcp  
success
```

其中，zone 是接口要使用的区域。请注意，这些改变不会在下次系统启动后仍然存在。为了永久更改防火墙，添加--permanent 选项的命令。有关开和关 firewalld 端口的详细信息，请参阅中标麒麟高级服务器操作系统 V7 相关章节。

2. 核实上面的配置，使用如下命令：

```
~]# firewall -cmd --list-all  
public (default, active)  
interfaces: eth0
```

```
sources:

services: dhcpv6-client ssh

ports: 10 514 /tcp

masquerade: no

forward-ports:

icmp-blocks:

rich rules:
```

程序 5.6 配置 syslog 接收和排序远端的日志消息

1. 以文本模式打开/etc/rsyslog.conf 文件，执行如下步骤：
 - a) 在模块部分下面在提供 UDP syslog 接收部分上面添加这些行：

```
# Define templates before the rules that use them

### Per-Host Templates for Remote Systems ###

$template TmplAuthpriv,

"/var/log/remote/auth/%HOSTNAME%/%PROGRAMNAME:::secpathreplac
e%.
log"

$template TmplMsg,

"/var/log/remote/msg/%HOSTNAME%/%PROGRAMNAME:::secpathreplac
e%.
log"
```

- b) 用下面这些行替代提供的 TCP syslog 接收部分：

```
# Provides TCP syslog reception

$ModLoad imtcp

# Adding this ruleset to process remote messages

$RuleSet remote1

authpriv.* ?TmplAuthpriv

*.info;mail.none;authpriv.none;cron.none ?TmplMsg

$RuleSet RSYSLOG_DefaultRuleset #End the rule set by
```

```
switching back to the default rule set

$InputTCPServerBindRuleset remote1 #Define a new input and
bind it to the "remote1" rule set

$InputTCPServerRun 10514
```

保存配置到/etc/rsyslog.conf 文件中。

2. rsyslog 服务必须运行在日志服务器上或试图登录到日志服务器上的系统上。

a) 使用 systemctl 命令开启 rsyslog 服务。

```
~]# systemctl start rsyslog
```

b) 确保 rsyslog 服务开机自动运行，以 root 身份输入如下命令：

```
~]# systemctl enable rsyslog
```

在您的环境中，您的日志服务器现在配置为接收和存储其他系统的日志文件。
在日志服务器上使用模板

rsyslog 7 具有多种不同的模板样式。字符串模板最接近旧格式。用字符串格式生成上面例子中的模板，如下所示：

```
template(name="TmplAuthpriv" type="string"
string="/var/log/remote/auth/%HOSTNAME%/%PROGRAMNAME:::secpathr
eplace%.
log"
)
template(name="TmplMsg" type="string"
string="/var/log/remote/msg/%HOSTNAME%/%PROGRAMNAME:::secpathr
eplace%.
log"
)
```

这些模板也可以写成下面这样的格式列表：

```
template(name="TplAuthpriv" type="list") {
    constant(value="/var/log/remote/auth/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}

template(name="TplMsg" type="list") {
    constant(value="/var/log/remote/msg/")
    property(name="hostname")
    constant(value="/")
    property(name="programname" SecurePath="replace")
    constant(value=".log")
}
```

对这些 rsyslog 新规则来说，这个模板文本格式可能会更容易阅读，因此可以更容易应用。

要完成新语法的更改，我们需要重新生成模块加载命令，添加规则集，然后绑定协议，端口和规则集的规则：

```
module(load="imtcp")
ruleset(name="remote1"){
    authpriv.* action(type="omfile" DynaFile="TplAuthpriv")
    *.info;mail.none;authpriv.none;cron.none action(type="omfile"
    DynaFile="TplMsg")
}

input(type="imtcp" port="10514" ruleset="remote1")
```

5.3.6 使用 Rsyslog 模块

由于采用模块化设计，rsyslog 提供各种模块，这些模块提供额外的功能。需要注意的是模块可以由第三方开发。大多数模块提供额外的输入（见下文输入

模块)或输出(见下文输出模块)。其他模块提供具体到每个模块的特定功能。加载一个模块后,其可提供可用的附加配置指令。要加载模块,请使用以下语法:

```
$ModLoad MODULE
```

其中\$ModLoad 是加载指定的模块的全局指令。MODULE 表示你所需的模块。例如,如果你要加载的文本文件输入模块(imfile),使 rsyslog 将任何标准的文本文件转换成系统日志信息,在 /etc/rsyslog.conf 配置文件中指定以下行:

```
$ModLoad imfile
```

rsyslog 提供了许多模块,这些模块被分为以下类别:

- 输入模块-输入模块收集来自各种来源的消息。输入模块的名称总是以 im 前缀开始,如 imfile 和 imjournal。
- 输出模块-输出模块提供便利给各种目标发出消息,如通过网络发送,存储在数据库中,或者加密等消息。输出模块的名称总是以 om 前缀,如 omsnmp, omrel p 等等。
- 解析模块- 这些模块在创建自定义解析规则,或者解析异常的消息是有用的。使用 C 语言的中间层,你可以创建自己的消息解析器。解析器模块的名称总是以 pm 前缀开始,如 pmrfc5424, pmrfc3164, 等。
- 消息修改模块 - 消息修改模块修改系统日志消息的内容。这些模块的名称以 mm 的前缀开始。消息修改的模块如 mmanon, mmnormalize, 或 mmjsonparse 其用于匿名或消息的正常化。
- 字符串生成模块 - 串生成模块生成基于消息内容的字符串,并与 rsyslog 提供的模板功能相互协同。有关模板的详细信息,请参阅 5.3.2.3 模板。串生成模块的名称总是以 sm 前缀开始,如 smfile 或 smtrad 文件。
- 库模块 - 库模块为其他可加载模块提供功能。这些模块在 rsyslog 需要时自动被加载,不能由用户配置。

所有可用的模块和它们的详细描述完整列表可以在 http://www.rsyslog.com/doc/rsyslog_conf_modules.html 找到。

警告:

需要注意的是 `rsyslog` 加载的模块，这使他们可以访问自己的函数和数据。这会带来潜在的安全威胁。为了最大限度地减少安全风险，仅使用可信赖模块。

导入 text 文件

文本文件输入模块，简称 `imfile`，使 `rsyslog` 将任意文本文件转化成系统日志消息流。您可以使用 `imfile` 从创建自己的文本文件日志的应用程序中导入日志消息。要加载 `imfile`，在 `/etc/rsyslog.conf` 添加以下内容：

```
$ModLoad imfile
$InputFilePollInterval int
```

它足以加载 `imfile` 一次，导入多个文件时也是如此。`$InputFilePollInterval` 全局指令指定 `rsyslog` 检查被连接的文本文件变化的频率。默认的时间间隔为 10 秒，要改变它，用以秒为单位的时间间隔替换 `int`。

为了确定文本文件导入，在 `/etc/rsyslog.conf` 文件中使用的语法如下：

```
# File 1
$InputFileName path_to_file
$InputFileTag tag:
$InputFileStateFile state_file_name
$InputFileSeverity severity
$InputFileFacility facility
$InputRunFileMonitor

# File 2
$InputFileName path_to_file2

...
```

需要 4 个设置来指定一个输入文件：

- 用文本文件路径替代 `path_to_file`
- 用消息的 `tag` 名替代 `tag`
- 用唯一的状态文件名替代 `state_file_name`。状态文件都存储在 `rsyslog` 的工作目录，为监控的文件保持标记，标记已经被处理完的分区。如果删除

它们，整个文件将被重新读入。请确保您指定了一个不存在的名称。

➤ 添加\$InputRunFileMonitor 指令使能文件监控。没有这个设置，文本文件会被忽略。

除了所要求的指令，存在可以应用到文本输入的几个其它设置。通过用适当的關鍵字替换关键字 severity 来设置导入消息的严重性。用关键字替换 facility 来定义生成消息的子系统。关键字 severity 和 facility 都是一样，就像其在基于设备/优先级的过滤器中使用那样，请参见 5.3.2.1 过滤器。

实例：导入文本文件

Apache HTTP 服务器以文本格式创建日志文件。要应用 rsyslog 对 Apache 的错误信息的处理能力，请首先使用 imfile 模块导入消息。添加以下内容到 /etc/rsyslog.conf 文件：

```
$ModLoad imfile
$InputFileName /var/log/httpd/error_log
$InputFileTag apache-error:
$InputFileStateFile state-apache-error
$InputRunFileMonitor
```

从数据库导出消息

在数据库中，处理日志数据比处理文本文件可以更快，更方便。基于所使用的 DBMS 的类型，选择各种输出模块，例如 ommysql, ompgsql, omoracle, 或 ommongodb。作为替代，使用依赖于 libdbi 库的通用 omlibdbi 输出模块。omlibdbi 模块支持的数据库系统有 Firebird/ Interbase, MS SQL, Sybase, SQLite, ingres, 甲骨文, mSQL, MySQL 和 PostgreSQL。

实例：从数据库导出 rsyslog 消息

要存储 rsyslog 消息到 MySQL 数据库中，添加以下内容到/etc/rsyslog.conf:

```
$ModLoad ommysql
$ActionOmmysqlServerPort 1234
*. * :ommysql:database-server,database-name,database-userid,databasepasswo
rd
```


首先，输出模块被加载，则通信端口被所指定。其他信息，如服务器和数据库的名称和认证数据，在上述的例子中的最后一行被指定。

使能加密传输

网络传输的机密性和完整性可以通过 TLS 或 GSSAPI 加密协议来提供。

传输层安全性（TLS）是旨在提供通过网络通信的安全性的加密协议。当使用 TLS，rsyslog 消息被发送之前加密，发送者和接收者之间的需要相互认证。

通用安全服务 API（GSSAPI）是一种为程序访问安全服务的应用程序编程接口。为了在与 rsyslog 的连接中使用，你必须有一个正常的 Kerberos 环境。

5.3.7 Syslogd 服务和日志的交互

如上所述，rsyslog 和 journal，你系统上的两个日志应用程序，有几个鲜明的特点，使它们适用于特定的用例。在很多情况下（见 5.4 Syslogd 日志结构），它们可以相互合作，比如创建结构化的信息，并将它们存储在一个文件数据库中。需要这种合作的一个通信接口是由输入和输出模块提供的，这些模块由 Rsyslog 和 journal 的通信 socket 支持。

默认情况下，rsyslogd 使用 imjournal 模块作为日志文件的默认的输入模式。使用这个模块，你不仅可以导入消息，也可以导入由 Journald 提供的结构化数据。此外，旧的数据可以从 journald 导入（除非禁止用 \$ImjournalIgnorePreviousMessages 指令）。参考 5.4.1 关于 imjournal 的基本配置。

作为替代，配置 rsyslogd 来读取由 journal 提供的套接字作为一个基于 syslog 应用的输出。套接字的路径/run/systemd/journal/syslog。当你想维护 rsyslog 消息，请使用此选项。相比 imjournal，套接字输入提供更多的功能，如绑定规则集或过滤器。要通过套接字导入日志数据，在/etc/rsyslog.conf 中使用以下配置：

```
$ModLoad imuxsock
$OmitLocalLogging off
```

上述语法加载 imuxsock 模块并关闭 \$OmitLocalLogging 指令，使能通过系统套接字导入。套接字的路径分别在/etc/rsyslog.d/listen.conf 中指定，如下：

```
$SystemLogSocketName /run/systemd/journal/syslog
```

您也可以从 Rsyslog 输出消息到使用 omjournal 模块的 journal。在

/etc/rsyslog.conf 中配置输出如下：

```
$ModLoad omjournal
```

```
*.* :omjournal:
```

例如，下面的配置转发所有收到的信息通过 TCP 端口 10514 到 journal：

```
$ModLoad imtcp
```

```
$ModLoad omjournal
```

```
$RuleSet remote
```

```
*.* :omjournal:
```

```
$InputTCPServerBindRuleset remote
```

```
$InputTCPServerRun 10514
```

5.4 Syslogd 日志结构

在产生大量的日志数据的系统上，一个结构化格式的日志信息可以方便地被维护。通过结构化的信息，很容易搜索特定信息，生成统计信息和处理消息的改变和不一致。rsyslog 使用 JSON（JavaScript 对象符号）格式提供日志信息的结构化。

比较下面非结构化的日志消息：

```
Oct 25 10:20:37 localhost anacron[1395]: Jobs will be executed  
sequentially
```

下面是结构化的日志消息：

```
{"timestamp":"2013-10-25T10:20:37", "host":"localhost",  
  "program":"anacron", "pid":"1395", "msg":"Jobs will be executed  
sequentially"}
```

使用键值对搜索结构化数据比使用正则表达式搜索文本文件的速度更快更精确。结构化消息还可以让你搜索各种应用程序生成的相同的消息。另外，JSON 文件可以存储在文档数据库，如 MongoDB，它提供了额外的性能和分析功能。另一方面，一个结构化的消息需要比非结构化的消息更多的磁盘空间。

在 rsyslog，带有元数据的日志信息被使用 imjournal 模块的 journal 推出。使用 mmjsonparse 模块，可以解析来自 journal 和其他来源导入的数据，并进一步

对其进行处理，例如，作为一个数据库输出。为了使解析成功，mmjsonparse 要求输入消息是结构化的，向其在 lumberjack 项目定义的那样。

Lmberjack 项目的目的是以向后兼容的方式为 rsyslog 增加结构化日志记录。要确定一个结构化的消息，Lmberjack 指定 @ cee: 字符串，其前置 JSON 结构。另外，Lmberjack 定义了应该用在 JSON 串标准字段名称的列表。有关 Lmberjack 的更多信息，请参见在线文档。

下面是 lumberjack 格式消息的例子：

```
@ cee: { "pid":17055, "uid":1000, "gid":1000, "appname":"logger",
        "msg":"Message text." }
```

要在 Rsyslog 中构建这个结构，一个模板会被使用，请参见 5.4.2 过滤结构化消息 。应用程序和服务端可以采用 libumberlog 库来生成 lumberjack 兼容形式的消息。有关 libumberlog 的更多信息，请参见在线文档 。

5.4.1 从日志中导入数据

imjournal 模块是 rsyslog 的输入模块用来读取日志文件（见 5.3.7 Syslogd 服务和日志的交互）。作为其它的 rsyslog 消息，日志消息以文本格式被记录。然而，随着进一步的处理，其能够将由 journal 提供的元数据翻译成结构化的消息。

为了从 Journal 导入数据到 Rsyslog，在/etc/rsyslog.conf 文件中使用下面配置：

```
$ModLoad imjournal
$imjournalPersistStateInterval number_of_messages
$imjournalStateFile path
$imjournalRatelimitInterval seconds
$imjournalRatelimitBurst burst_number
$imjournalIgnorePreviousMessages off/on
```

- 使用 number_of_messages，您可以指定保存日志数据的频率。每当达到 number_of_messages 指定的消息数时，就会触发数据保存。
- 用状态文件的路径替代 path。此文件跟踪 journal 记录，其是处理的最后一个记录。
- 使用 seconds，设置的限制速率的时间间隔。此间隔期间处理的消息

的数量不能超过在 `burst_number` 指定的值。默认设置为每 600 秒 20000 的消息。Rsyslog 会丢弃在规定的时限内超过最大 `burst_number` 值的消息。

➤ 使用 `$ ImjournalIgnorePreviousMessages` 可以忽略当前在 `journal` 的消息和只导入新的消息，这些消息当未指定状态文件时被使用。默认设置为关闭。请注意，如果该设置是关闭的，没有状态文件，所有 `journal` 的消息会被处理，即使他们已经在 `rsyslog` 以前的会话中被处理过。

注意：

您可以同时使用 `imjournal` 和 `imuxsock` 模块，`imuxsock` 模块是旧的系统日志输入。然而，为了避免消息重复，你必须阻止 `imuxsock` 读取 `journal` 的系统套接字。要做到这一点，使用 `$ OmitLocalLogging` 指令：

```

$ModLoad imuxsock

$ModLoad imjournal

$OmitLocalLogging on

$AddUnixListenSocket /run/systemd/journal/syslog
    
```

您可以通过将存储在 `journal` 的数据和元数据翻译成结构化消息。其中的一些元数据项在本章实例“`journalctl` 的 `verbose` 输出”中列出，想获取 `journal` 域的完整列表，请参阅 `systemd.journal-fields` (7) 手册页。例如，可以将重点放在内核 `journal` 字段，该域被内核初始生成的消息使用。

5.4.2 过滤结构化消息

要创建一个 `rsyslog` 解析模块需要的 `lumberjack` 格式的消息，请使用以下模板：

```

template(name="CEETemplate"                                type="string"
string="%TIMESTAMP% %HOSTNAME%
%syslogtag% @ cee: %$!all-json%\n")
    
```

这个模板预先考虑 `@cee:` 可以应用的 `JSON` 字符串，例如，当创建使用 `omfile` 模块的输出文件时。要访问 `JSON` 字段名称，使用 `$!` 前缀。例如，搜索符合下面的筛选条件的消息，指定了消息的主机名和 `UID`。

```
(${!hostname} == "hostname" && ${!UID} == "UID")
```

5.4.3 解析 JSON

mmjsonparse 模块用于解析结构化消息。这些信息可以来自 journal 或其他输入源，并且必须由 lumberjack 项目中定义的方式进行格式化。这些消息是由 @cee: 字符串表示而被识别的。然后，mmjsonparse 会检查 JSON 结构是否是有效的，然后该消息被解析。

为了用 mmjsonparse 模块解析 lumberjack 格式的 JSON 消息，在 /etc/rsyslog.conf 文件下使用下面的配置：

```
$ModLoad mmjsonparse

*.* :mmjsonparse:
```

在这个例子中，mmjsonparse 模块在第一行被加载，所有的消息转发给它。目前，没有可用的配置参数用于 mmjsonparse。

5.4.4 向 MongoDB 中存储消息

rsyslog 支持通过 ommongodb 输出模块在 MongoDB 的文档型数据库中存储 JSON 日志。

要转发日志消息到 MongoDB 中，在 /etc/rsyslog.conf 使用以下语法（ommongodb 配置参数只适用于新配置格式；见 5.3.3 使用新的配置格式）

```
$ModLoad ommongodb

*.* action(type="ommongodb" server="DB_server" serverport="port"
db="DB_name" collection="collection_name" uid="UID" pwd="password")
```

➤ 用 MongoDB 服务器的地址或者名字替代 DB_server。配置端口需要从 MongoDB 的服务器选择一个非标准的端口。默认端口值是 0，并且通常没有必要改变该参数。

➤ 使用 DB_NAME，确定你想直接输出到 MongoDB 的服务器上的哪个数据库。用这个数据库集合的名称替换 collection_name。在 MongoDB 中，集合是一组文档，它和一个 RDBMS 表是等效的。

➤ 你通过设置 UID 和 password 来输入你的细节信息。

你可以使用模板来塑造最终数据库的输出形式。默认情况下，rsyslog 使用

的模板基于标准的 lumberjack 字段名称。

5.5 调试 Rsyslog

在 debug 模式运行 rsyslogd,使用如下命令:

```
rsyslogd -dn
```

使用此命令, rsyslogd 产生调试信息并打印到标准输出。-n 表示 “no fork”。您可以修改调试环境变量, 例如, 可以在日志文件中存储调试输出信息。在启动 rsyslogd 之前, 在命令行上执行以下操作:

```
export RSYSLOG_DEBUGLOG="path"  
export RSYSLOG_DEBUG="Debug"
```

用所需记录调试信息文件的位置替换 path。对于可用于 RSYSLOG_DEBUG 变量选项的完整列表, 请参阅 rsyslogd (8) 手册的相关章节。

检查在/etc/rsyslog.conf 文件中使用的语法是否有效:

```
rsyslogd -N 1
```

其中, 1 表示输出消息的详细程度的级别。这是前向兼容性选项, 因为目前, 只有一个级被提供。但是, 你必须添加此参数来运行验证。

5.6 使用日志

journal 是 systemd 的一个组件, 它负责查看和管理日志文件。它可并行使用或者代替一个旧的 syslog 守护进程, 如 rsyslogd。journal 的开发是为了解决与旧的日志记录有关的问题。它紧密地与系统的其余部分集成, 支持多种日志记录技术和访问管理日志文件。

记录数据由 journald 服务收集, 存储, 并处理。它创建和维护名为 journals 的二进制文件, 这些记录的信息来自内核, 用户进程, 标准输出, 标准错误输出, 或通过原生 API 的标准错误输出。这些 journals 被结构化和并被索引, 它提供了比较快的查询道时间。journal 条目有一个唯一的标识符。journald 服务收集每个日志消息众多的元数据字段。日志文件是安全的, 不能被手动编辑。

5.6.1 查看日志文件

为了访问 journal 日志，使用 journalctl 工具。以 root 身份查看日志类型的基本信息：

```
journalctl
```

此命令的输出是系统所有日志文件的列表，包括系统组件和用户产生的消息。该输出的结构类似于/var/log/messages 文件中消息结构，但有一定的改进：

- 记录优先级的标记是可见的。错误的优先级和更高的优先级使用了红色，通知和警告优先线使用加粗的字体。
- 时间戳转换为系统的本地时区。
- 所有记录数据都会显示，包括轮替的日志。
- 引导的开始将被标上专用线。

实例：journalctl 的输出

以下是由 journalctl 工具提供的示例输出。当不带参数调用时，列出的条目以一个时间戳开始，然后是主机名和执行操作的应用程序，然后是实际的消息。这个例子显示在 journal 日志中的前三条记录：

```
# journalctl
-- Logs begin at Thu 2013-08-01 15:42:12 CEST, end at Thu 2013-08-01
15:48:48 CEST. --
Aug 01 15:42:12 localhost systemd-journal[54]: Allowing runtime journal
files to grow to 49.7M.
Aug 01 15:42:12 localhost kernel: Initializing cgroup subsys cpuset
Aug 01 15:42:12 localhost kernel: Initializing cgroup subsys cpu
[...]
```

在许多情况下，只有 journal 日志中最新条目是有用的。减少 journalctl 输出最简单的方法是使用 -n 选项，列出最近的日志条目，条目数由 -n 的 number 指定：

```
journalctl -n Number
```

用希望显示的条目数目替代 number。如果没有指定 number，journalctl 现在

最近的 10 条信息。

Journalctl 命令可以使用下面的语法控制输出格式：

```
journalctl -o form
```

用希望输出格式的关键字替代 form。有多个选项，如 verbose，它可以返回所有字段的全部结构化的条目。还如 export，创造适合于备份和网络传输二进制流。还有 json,以 JSON 数据结构格式化消息。关于完整的关键字列表，请看 journalctl (1) man 手册页。

实例：journalctl 的 verbose 输出

看所有条目的全部元数据，请输入以下命令：

```
# journalctl -o verbose
[...]

Fri 2013-08-02 14:41:22 CEST
[s=e1021ca1b81e4fc688fad6a3ea21d35b;i=55c;b=78c81449c920439da57da7bd5c56a770;m=27cc
  _BOOT_ID=78c81449c920439da57da7bd5c56a770
  _PRIORITY=5
  _SYSLOG_FACILITY=3
  _TRANSPORT=syslog

  _MACHINE_ID=69d27b356a94476da859461d3a3bc6fd
  _HOSTNAME=localhost.localdomain
  _PID=562
  _COMM=dbus-daemon
  _EXE=/usr/bin/dbus-daemon
  _CMDLINE=/bin/dbus-daemon --system --address=systemd: --nofork
  --nopidfile --systemd-activation
  _SYSTEMD_CGROUP=/system/dbus.service
  _SYSTEMD_UNIT=dbus.service
  _SYSLOG_IDENTIFIER=dbus
  _SYSLOG_PID=562
  _UID=81
  _GID=81
  _SELINUX_CONTEXT=system_u:system_r:system_dbusd_t:s0-s0:c0.c1023
  MESSAGE=[system] Successfully activated service
  'net.reactivated.Fprint'
  _SOURCE_REALTIME_TIMESTAMP=1375447282839181

[...]
```

这个例子列出了标识单个日志条目字段。这些元数据可用于消息过滤，如 5.6.4 过滤消息中的高级过滤。对于所有字段的完整描述请参见 systemd.journal-fields (7) 手册页。

5.6.2 访问控制

默认情况下，没有 root 权限的 journal 用户，只能查看由它们产生的日志文件。系统管理员可以添加选定用户到 adm 组，授予他们访问完整日志文件的权

限。要做到这一点，以 root 用户输入如下：

```
usermod -a -G adm username
```

用要添加到 adm 组的用户名替代 username。该用户随后接收 journalctl 命令的输出同 root 用户获取的输出一样。请注意，访问控制只有当为 journal 启用了持久存储才会工作。

5.6.3 使用 Live view

当不带参数调用，journalctl 从收集到的最早的条目开始，显示条目的完整列表。有了 live view，你可以监控实时的日志消息，因为新的条目出现就会被打印出来。要开启 journalctl 的 live view 模式，输入：

```
journalctl -f
```

该命令返回十个最新的日志行列表。journalctl 工具将保持运行，等待新的消息并立即显示他们。

5.6.4 过滤消息

不带参数执行 journalctl 命令的输出信息是很多的，因此您可以使用各种过滤方法来提取信息，以满足您的需求。

通过优先级过滤

日志消息通常用于跟踪系统上错误的行为。要查看选定的或更高优先级消息，请使用以下语法：

```
journalctl -p priority
```

在这里，用下面的关键字之一（或数字）替换优先级：debug（7），info（6），notice（5），warning（4），err（3），crit（2），alert（1），和 emerg（0）。

实例：通过优先级过滤

查看优先级为 error 或更高的消息，使用：

```
journalctl -p err
```

通过时间过滤

要查看仅从当前引导的日志条目：

```
journalctl -b
```

如果您偶尔会重新启动系统，**-b** 不会显著减少 **journalctl** 的输出。在这样的情况下，基于时间的滤波是更有帮助：

```
journalctl --since=value --until=value
```

使用**--since** 和**--until** 选项。您可以查看指定时间范围内创建的日志信息。你可以以 **time** 的格式或者 **data** 的格式或者两者都可传递 **value** 给这两个选项，如下示例。

实例：通过优先级和时间过滤

根据具体的要求过滤选项可以组合，以减少的结果输出。例如，从某个时间点，查看警告或更高优先级的消息，如下：

```
journalctl -p warning --since= "20 13-3-16 23: 59 : 59 "
```

高级过滤

在本章实例“**journalctl** 的 **verbose** 输出”列出指定日志条目的一组字段，其都可以用于过滤。对于 **systemd** 可存储的元数据的完整说明，参见 **systemd.journal-fields** (7) 手册页。每个日志信息的元数据都被收集，而无需用户干预。值通常是基于文本的，但可以采取二进制和大的值；虽然不是很常见的，但是字段可以分配多个值。

要查看发生在一个特定领域产生的唯一值的列表，请使用以下语法：

```
journalctl -F fieldname
```

用你想要的字段名替代 **fieldname**。

要显示出满足特定条件日志条目，请使用以下语法：

```
journalctl fieldname=value
```

用字段名和该字段包含的值替代 **fieldname** 和 **value**。结果，符合这个条件的行会输出。

注意：

如通过 **systemd** 存储的元数据字段的数量是相当大的，它很容易忘记感兴趣

的字段的确切名称。如果不能确定，请键入：

```
journalctl
```

按两次 `tab` 键。这会显示出可用的字段名。`Tab` 键名称补齐是基于该字段的上下文选项完成的，您可以输入一部分字母，然后按 `Tab` 键来自动完成这个名字。同样，你可以从一个字段中列出唯一值。 如下：

```
journalctl fieldname=
```

按 `tab` 键两次，就会像输入 `journalctl -F fieldname` 一样。

你可以在一个字段自定多个值，如下：

```
journalctl fieldname=value1 fieldname=value2 ...
```

同一个字段指定两个可以匹配的值，像逻辑 `or` 一样匹配。符合匹配值 1 或值 2 的条目会显示。

当然，你可以指定多个 `field-value` 对来进一步减少输出：

```
journalctl fieldname1=value fieldname2=value ...
```

如果指定了两个不同的字段名的匹配，它们将与逻辑 `AND` 组合。必须匹配这两个条件的条目才会显示。

使用`+`号，你可以设置逻辑 `or` 组合匹配多字段：

```
journalctl fieldname1=value + fieldname2=value ...
```

该命令返回至少匹配一个条件的条目，不仅是满足所有条件的条目。

实例：高级过滤器设置

要显示通过 `avahi-daemon` 或 `crond.service` 创建的条目。且该条目的用户 `UID` 为 70，请使用以下命令：

```
journalctl _UID = 70 _SYSTEMD_UNIT = avahi-daemon.service
_SYSTEMD_UNIT = crond.service
```

由于 `_SYSTEMD_UNIT` 字段可以设置两个值，匹配两者的结果将显示，但 `_UID= 70` 条件必须被满足。这可以简单地表示为 `(UID=70 and (avahi or cron))`。

你可以应用到前面提到的过滤器，其以 `live-view` 模式来跟踪选定组的日志

消息的变化:

```
journalctl -f fieldname=value ...
```

5.6.5 使能持续存储

默认情况下, `journal` 只在内存中或 `/run/log/journal/` 目录下的小环形缓冲区中存储日志文件。使用 `journalctl` 显示最近历史日志就足够了。该目录有易失性, 日志数据不会永久保存。在默认配置下, `syslog` 读取日志记录, 并将其存储在 `/var/log/` 目录下。若持续性日志记录开启, 日志文件存储在 `/var/log/journal`, 这意味着重新启动后他们仍然存在。对一些用户, `journal` 能代替 `rsyslog`。

使能持续性存储有下面的优点:

- 较长的时间的可用用来解决问题的更丰富的数据会被记录。
- 对于需要立刻解决的问题, 重启后可以获取更丰富的可用的数据。
- 服务器控制台当前从日志中读取数据, 而不是日志文件。

持续性存储也有某些缺点

- 持续性存储所存储的数据量取决于空闲的内存, 不保证可以覆盖特定的时间跨度。
- 需要更多的磁盘空间存储日志文件。

`journal` 启用持续性存储, 按下面所示的例子手动创建 `journal` 目录, 以 `root` 用户输入:

```
mkdir -p /var/log/journal
```

重启 `journald` 来是设置生效。

```
systemctl restart systemd-journald
```

5.7 在图形界面管理日志

作为一种替代上述命令行的实用程序, 中标麒麟高级服务器操作系统 V7 提供了一个可访问的图形用户界面来管理日志消息。

5.7.1 查看日志文件

大多数日志文件都以纯文本格式存储。你可以用任何文本编辑器如 `vi` 或

Emacs 来查看它们。某些日志文件可以被系统上所有用户查看;然而, 需要 root 权限来阅读大多数日志文件。

以交互的实时的方式查看系统日志文件的应用程序。可以使用 SystemLog。

注意:

为了可以使用 SystemLog, 首先确保你的系统安装了 gnome-system-log 包。

以 root 用户执行如下:

```
~]# yum install gnome-system-log
```

更多关于使用 yum 安装包的信息, 请查看 2.1.2.4 安装软件包。

在你安装完 gnome-system-log 包之后, 通过点击应用程序->系统工具->SystemLog 或者输入如下命令:

```
~]$ gnome-system-log
```

应用程序只会显示存在的日志文件, 因此列表也许和图 5-5 SystemLog 显示的有所不同。

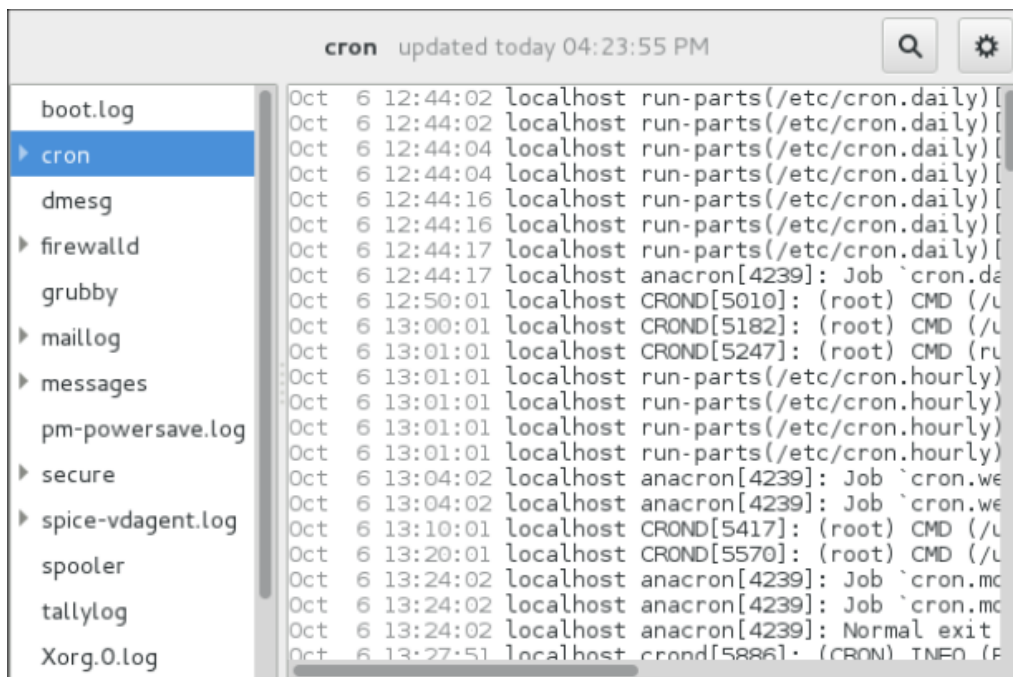


图 5-5 SystemLog

SystemLog 应用程序允许你过滤任何存在的日志文件。点击标有齿轮符号的按钮来查看菜单, 选择 Filters→Manage Filters 来定义或编辑所需的过滤器。

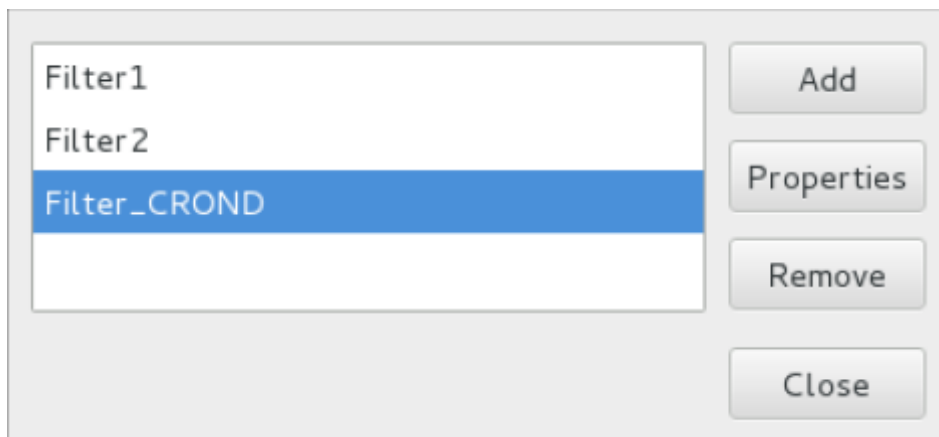


图 5-6 SystemLog –Filters

添加或编辑过滤器，您可以定义它的参数，该参数在图 5-7 SystemLog 定义过滤器中显示。

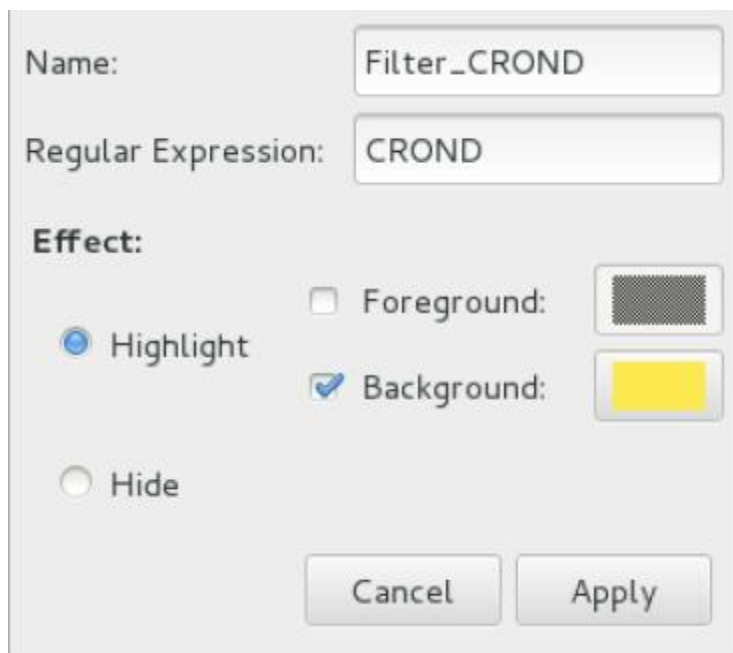


图 5-7 SystemLog 定义过滤器

当定义一个过滤器，下面的参数需要被编辑。

- Name – 定义过滤器的名字
- Regular Expression -指定被应用到日志文件的正则表达式，其会尝试匹配文本中任何可能的字符串。
- Effect
 - Highlight - 如果选中，找到的结果将用所选择的颜色突出显示。

可以选择是否要突出背景或文本的前景

- **Hide** - 如果选中, 发现的结果会 from 你正在查看的日志文件中被隐藏。

当你定义了至少一个过滤器, 它可以在 **Filters** 菜单中选择, 它会自动搜索您在过滤器中定义的字符串, 并突出显示或隐藏当前正在查看日志文件中每一个成功匹配的消息。

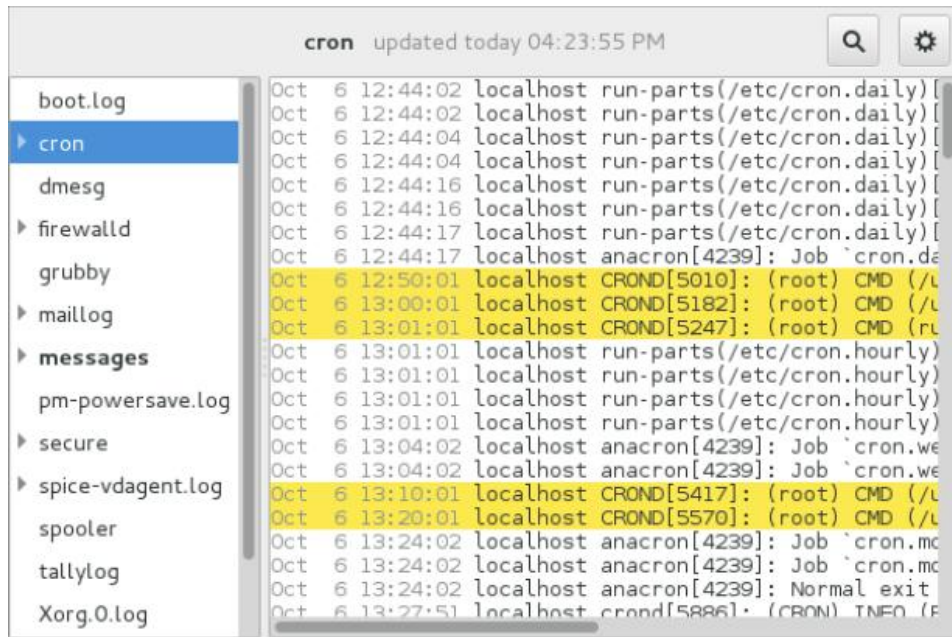


图 5-8 SystemLog – 启用过滤器

当您选择 **Show matches only** 选项, 只有匹配的字符串将显示在当前正在查看日志文件中。

5.7.2 添加日志文件

要添加一个日志文件到你想要查看的列表中, 选择 **File->Open**。这会显示 **Open Log** 窗口, 在这个窗口你可以选择你想查看的目录和日志文件。图 5-9 System Log – 添加日志文件显示了 **Open Log** 窗口。

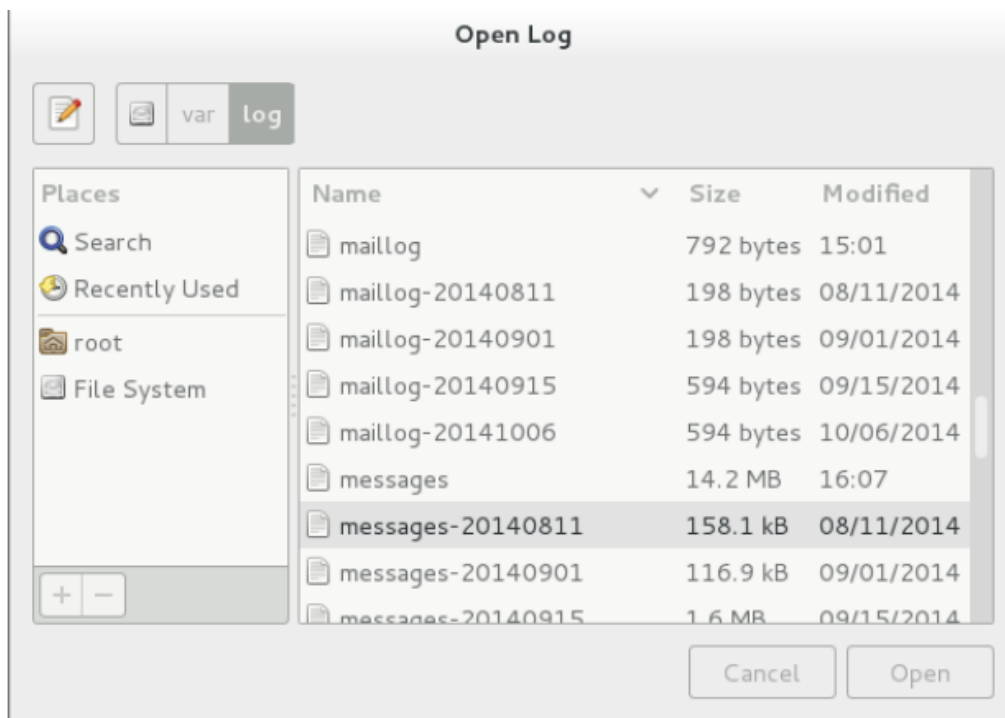


图 5-9 System Log – 添加日志文件

点击 **Open** 按钮来打开文件，文件会立即添加到你正查看的列表，这里你可以选中它查看他的内容。

注意：System Log 也允许你打开以.gz 格式压缩的日志文件。

5.7.3 监控日志文件

System Log 以在默认情况下监控所有打开的日志。如果一个新行被添加到受监视的日志文件，日志名称以粗体显示在日志列表中。如果日志文件被选中或显示，新的行以粗体显示在日志文件的底部。图 5-10 System Log - 新日志警报说明了一个新的警报在 cron 日志文件中和在 messages 日志文件中。点击 messages 日志文件,日志在文件中以新行粗体显示。

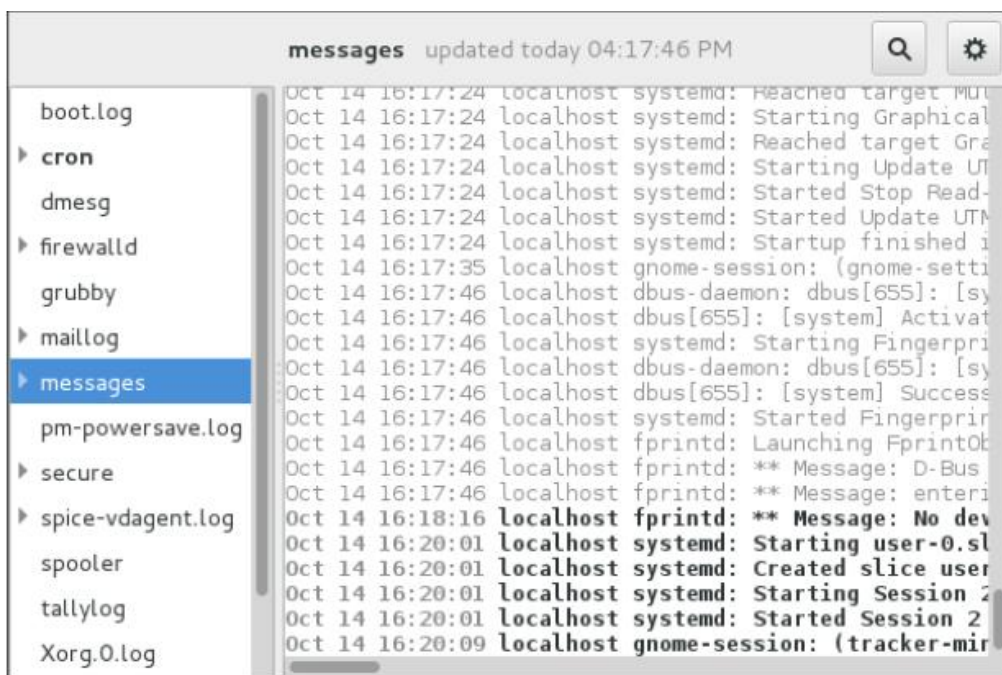


图 5-10 System Log - 新日志警报

5.8 自动化系统任务

任务，也被称为工作，可以通过配置，在一个指定的日期，在一个指定的时间周期内，或者当系统的平均负载低于 0.8 时，自动地运行。

中标麒麟高级服务器操作系统预先配置了运行一些重要的系统任务，以保持系统的更新。例如，locate 命令使用的 slocate 数据库每天都会更新。系统管理员可以使用自动任务来执行周期性的备份、监控系统、运行自定义脚本等等。

中标麒麟高级服务器操作系统提供了以下自动任务工具：**cron**、**anacron**、**at** 和 **batch**。

每一种工具都是为了调度不同的任务类型而被设计的，**Cron** 和 **Anacron** 调度重复发生的任务，**At** 和 **Batch** 调度一次性的任务（请分别参考 5.8.1Cron 和 Anacron 和 5.8.8At 和 Batch）。

中标麒麟高级服务器操作系统 V7 支持使用 **systemd.timer** 在一个指定的时间执行一个任务。参见 **systemd.timer(5)** 用户手册页面了解更多信息。

5.8.1 Cron 和 Anacron

Cron 和 **Anacron** 都是能够调度重复性任务在一个确定的时间点执行的守护进程，时间点是准确的时间、月份中的某天、月份、一周中的某天、周末定

义的。

Cron 的任务可以每分钟都运行。但是，这个工具假定系统是持续运行的，如果在任务被安排的时间系统并没有运行，则任务不会被执行。

另一方面，如果系统在任务被安排的时间被没有运行，Anacron 会记住这个已经被安排的任务。一旦系统开始运行，这个任务将被马上执行。然而，Anacron 对于一个任务一天只能运行一次。

5.8.2 安装 Cron 和 Anacron

要安装 Cron 和 Anacron，您需要安装 Cron 的 `crontab` 包和 Anacron 的 `crontab-anacron` 包（`crontab-anacron` 是 `crontab` 的一个子包）。

要确定您的系统上是否已经安装了相应的包，可以执行以下命令：

```
rpm -q crontab crontab-anacron
```

如果已经安装了，上述命令将返回 `crontab` 包和 `crontab-anacron` 包的完整名称，否则将通知您相应的包不可用。

要安装这些包，以 root 用户按照下面的格式来使用 `yum` 命令：

```
yum install package
```

例如，要同时安装 Cron 和 Anacron，可以在命令行提示符下输入以下命令：

```
~]# yum install crontab crontab-anacron
```

要了解如何在中标麒麟高级服务器操作系统中安装新的包，请参见 2.1.2.4 安装软件包。

5.8.3 运行 Crond 服务

`crontab` 和 `anacron` 任务都是由 `crond` 服务来控制的。本节将说明如何启动、停止和重启 `crond` 服务，以及如何配置让其开机自启动。要了解更多有关在中标麒麟高级服务器操作系统 V7 中如何管理服务的通用方法，请参见 3.1 使用 `systemd` 管理系统服务。

启动 Cron 服务

要确定服务是否正在运行，请使用以下命令：

```
systemctl status crond.service
```

要在当前会话中运行 **crond** 服务，请以 **root** 用户在命令行提示符下输入以下命令：

```
systemctl start crond.service
```

要配置服务开机自启动，请以 **root** 用户执行以下命令：

```
systemctl enable crond.service
```

停止 Cron 服务

要在当前会话中停止 **crond** 服务，请以 **root** 用户在命令行提示符下输入以下命令：

```
systemctl stop crond.service
```

要禁止服务开机自启动，请以 **root** 用户执行以下命令：

```
systemctl disable crond.service
```

重启 Cron 服务

要重启 **crond** 服务，请以 **root** 用户在命令行提示符下输入以下命令：

```
systemctl restart crond.service
```

该命令停止服务后将很快地再次启动服务。

5.8.4 配置 Anacron 任务

调度任务的主要配置文件是 **/etc/anacrontab** 文件，它只能通过 **root** 用户进行访问。该文件包含以下内容：

```
SHELL=/bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin

MAILTO=root

# the maximal random delay added to the base delay of the jobs

RANDOM_DELAY=45
```

```
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22

#period in days    delay in minutes    job-identifier    command
1          5          cron.daily          nice run-parts
/etc/cron.daily

7          25          cron.weekly          nice run-parts
/etc/cron.weekly

@monthly 45          cron.monthly          nice run-parts
/etc/cron.monthly
```

前三行定义用来配置 **anacron** 任务运行环境的相关变量：

- **SHELL**——用来运行任务的 **shell** 环境（示例中是 **Bash shell**）
- **PATH**——可执行程序的路径
- **MAILTO**——通过电子邮件接收 **anacron** 任务输出的用户的名称

如果没有定义 **MAILTO** 变量（**MAILTO=**），则不会发送邮件。

接下来的两个变量用来修改已定义任务的调度时间：

- **RANDOM_DELAY**——将会加到为每个任务指定的 **delay in minutes** 变量上的最大分钟数。

默认情况下，最小延迟的值被设置为 6 分钟。

例如，如果 **RANDOM_DELAY** 被设置为 12，那么这个特定的 **anacrontab** 中的每一个任务的 **delay in minutes** 值都将被加上 6 到 12 分钟。**RANDOM_DELAY** 的值也可以设置为 6 以下，包括 0。当设置为 0 的时候，则不会增加随机延迟。随机延迟被证明是很有用的，例如，当多台共享一个网络连接的计算机需要每天都下载相同的数据时。

- **START_HOURS_RANGE**——计划任务可以运行的时间段，以小时为单位。

万一错过了这个时间段，例如，由于停电等原因，则当天的计划任务不会被执行。

/etc/anacrontab 文件的剩余行代表计划任务，并且遵从以下格式：

period in days	delay in minutes	job-identifier	command
----------------	------------------	----------------	---------

- period in days——按天数计的任务执行频率

该属性值可以被定义为一个整数或者一个宏（@daily、@weekly 或者 @monthly），@daily 和整数 1 表示的是同一个值，@weekly 和整数 7 一样，而 @monthly 则表示任务一个月只运行一次，不管这个月的天数是多少。

- delay in minutes——在执行任务前，anacron 等待的分钟数

该属性值可以被定义为一个整数。如果该值被设置为 0，则表示没有延迟。

- job-identifier——用于日志文件中关联一个特定任务的唯一名称

- command——将被执行的命令

这里的命令既可以是一个类似 ls /proc >> /tmp/proc 的命令，也可以是一个执行自定义脚本的命令。

以井号（#）开头的所有行都是注释，不会被处理。

Anacron 任务示例

以下是一个简单的/etc/anacrontab 文件的示例：

```
SHELL=/bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin

MAILTO=root

# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=30

# the jobs will be started during the following hours only
START_HOURS_RANGE=16-20

#period in days    delay in minutes    job-identifier    command
```

1	20	dailyjob	nice run-parts
/etc/cron.daily			
7	25	weeklyjob	/etc/weeklyjob.bash
@monthly	45	monthlyjob	ls /proc >> /tmp/proc

在这个 `anacrontab` 文件中定义的所有任务，都将随机的延迟 6 到 30 分钟，并且将在 16:00 到 20:00 之间被运行。

第一个被定义的任务将在每天 16:26 到 16:50 间被触发（`RANDOM_DELAY` 是在 6 到 30 分钟之间；`delay in minutes` 属性值为 20 分钟）。该任务所指定的命令将使用 `run-parts` 脚本（`run-parts` 脚本接受一个目录作为命令行参数，并顺序地执行目录中的每一个程序）执行 `/etc/cron.daily/` 目录下的所有程序。参见 `run-parts` 用户手册页面了解更多有关 `run-parts` 脚本的信息。

第二个任务每周执行一次 `/etc/` 目录下的 `weeklyjob.bash` 脚本。

第三个任务运行一个命令，该命令把 `/proc` 的内容写到 `/tmp/proc` 文件里，每个月一次（`ls /proc >> /tmp/proc`）。

5.8.5 配置 Cron 任务

Cron 任务的配置文件是 `/etc/crontab` 文件，它只能通过 `root` 用户进行访问。该文件包含以下内容：

```
SHELL=/bin/bash

PATH=/sbin:/bin:/usr/sbin:/usr/bin

MAILTO=root

HOME=/

# For details see man 4 crontabs

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
```

```
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...

# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat

# | | | | |
# * * * * * user-name  command to be executed
```

前三行包含了和 `anacrontab` 文件一样的变量定义：`SHELL`、`PATH` 和 `MAILTO`。要了解更多有关这几个变量的信息，请参见 5.8.4 配置 Anacron 任务。

此外，该文件还可以定义 `HOME` 变量。`HOME` 变量定义一个目录，该目录在任务执行命令或脚本时将被作为家目录。

`/etc/crontab` 文件的剩余行代表计划任务，并遵从以下格式：

```
minute    hour    day    month    day of week    username
command
```

以下各项定义了任务将要运行的时间：

- `minute`——从 0 到 59 的任意整数
- `hour`——从 0 到 23 的任意整数
- `day`——从 1 到 31 的任意整数（如果指定了月份，则这里的日期必须是相对于该月有效的日期）
- `month`——从 1 到 12 的任意整数（或者月份的简短名称，例如 `jan` 或者 `feb`）
- `day of week`——从 0 到 7 的任意整数，0 或 7 代表星期日（或者使用每周各天的简短名称，例如 `sun` 或者 `mon`）

以下各项定义了任务的其它属性：

- `username`——指定执行任务的用戶。
- `command`——将要执行的命令。

这里的命令既可以是一个类似 `ls /proc >> /tmp/proc` 的命令，也可以是一个执行自定义脚本的命令

对于以上的各项属性值，星号（*）可以用来表示所有的有效值。例如，如

果您将 `month` 的值定义为星号，则该任务将在其他条件的约束下每个月都执行。

整数之间的连字号 (-) 代表了一个整数范围。例如，`1-4` 表示整数 1、2、3、4。

用逗号 (,) 分隔的值列表指定了一个列表。例如，`3,4,6,8` 就确实指明了这四个整数。


斜杠 (/) 可以用来指定步长值。指定了步长值的项将按步长所定义的整数来增长。例如，`minute` 项的值定义为 `0-59/2`，则表示每隔一分钟。步长值也可以和星号一起使用。例如，如果 `month` 项的值被定义为 `*/3`，则任务将每三个月执行一次（每隔两个月）。

以井号 (#) 开头的所有行都是注释，不会被处理。

非 root 用户可以使用 `crontab` 工具来配置 `cron` 任务。用户定义的 `crontabs` 被保存在 `/var/spool/cron/` 目录下，并且以创建它们的用户来执行。

要以一个特定用户来创建 `crontab`，请以该用户进行登录，并且输入命令 `crontab -e` 来使用 `VISUAL` 或者 `EDITOR` 环境变量所指定的编辑器对用户的 `crontab` 进行编辑。该文件使用和 `/etc/crontab` 完全相同的格式。当用户对 `crontab` 的修改被保存时，`crontab` 按照用户名来保存，并写入 `/var/spool/cron/username` 文件中。要列出当前用户的 `crontab` 文件的内容，使用 `crontab -l` 命令。

`/etc/cron.d/` 目录下包含的文件和 `/etc/crontab` 文件具有相同的语法。只有 root 用户被允许在该目录下创建和修改文件。

 `cron` 守护进程会每分钟检查 `/etc/anacrontab` 文件、`/etc/crontab` 文件、`/etc/cron.d/` 目录和 `/var/spool/cron/` 目录的变化，并把检测到的修改加载到内存中。因此，当一个 `anacrontab` 或 `crontab` 文件被修改后，并不需要重启守护进程。

5.8.6 控制对 Cron 的访问

要限制对 `Cron` 的访问，您可以使用 `/etc/cron.allow` 和 `/etc/cron.deny` 文件。这些访问控制文件使用相同的格式，都是每一行一个用户名。记住，两个文件都不允许使用空格。

如果存在 `cron.allow` 文件，只有在该文件中列出的用户才能使用 `cron`，并且

`cron.deny` 文件将被忽略。

如果 `cron.allow` 文件不存在，则 `cron.deny` 文件中列出的用户将被禁止使用 Cron。

如果访问控制文件被修改了，不必重启 Cron 守护进程（`crond`）。每次用户试图添加或删除一个 `cron` 任务时，都会检查访问控制文件。

不管访问控制文件中的用户名是什么，`root` 用户都始终能够使用 `cron`。

您也可以通过插入式认证模块（Pluggable Authentication Modules, PAM）来控制访问。相应的设置保存在 `/etc/security/access.conf` 文件中。例如，在文件中添加如下一行之后，将只有 `root` 用户能够创建 `crontabs`：

```
-.:ALL EXCEPT root :cron
```

被禁止的任务将被记录在适当的日志文件中，或者，当使用 `crontab -e` 命令时，返回到标准输出里。要了解更多信息，请参考 `access.conf.5` 用户手册页面。

5.8.7 Cron 任务的黑白名单

任务的黑白名单用来定义任务的某部分不需要被执行。当在一个 Cron 目录里（例如，`/etc/cron.daily/`）调用 `run-parts` 脚本时这很有用：如果用户把这个目录下的程序加入了黑名单，`run-parts` 脚本将不会执行这些程序。

要创建一个黑名单，请在 `run-parts` 脚本将要执行的目录中创建一个 `jobs.deny` 文件。例如，如果您需要从 `/etc/cron.daily/` 目录中忽略一个特定的程序，请创建 `/etc/cron.daily/jobs.deny` 文件。在这个文件中，指定需要在执行时被忽略的程序名称（只有位于同一个目录下的程序能够加入列表）。如果一项任务执行一个命令，该命令从 `/etc/cron.daily/` 目录执行程序，例如 `run-parts /ect/cron.daily`，则 `jobs.deny` 文件中定义的程序不会被执行。

要定义一个白名单，请创建 `jobs.allow` 文件。

`jobs.deny` 和 `jobs.allow` 的使用规则和 5.8.6 控制对 Cron 的访问中描述的 `cron.deny` 和 `cron.allow` 的使用规则一样。

5.8.8 At 和 Batch

Cron 被用来调度重复性任务，At 工具被用来在一个指定的时间调度一次性任务，Batch 工具被用来调度将在系统平均负载低于 0.8 时被执行的一次性任务。

安装 At 和 Batch

要确定您的系统上是否已经安装了 `at` 包，请执行以下命令：

```
rpm -q at
```

如果已经安装了，上述命令将返回 `at` 包的完整名称，否则将通知您包不可用。

要安装程序包，以 `root` 用户按照下面的格式来使用 `yum` 命令：

```
yum install package
```

例如，要同时安装 `At` 和 `Batch`，可以在命令行提示符下输入以下命令：

```
~]# yum install at
```

要了解如何在中标麒麟高级服务器操作系统中安装新的包，请参见 2.1.2.4 安装软件包。

运行 At 服务

`At` 和 `Batch` 任务都是由 `atd` 服务来控制的。本节将说明如何启动、停止和重启 `atd` 服务，以及如何配置让其开机自启动。要了解更多有关在中标麒麟高级服务器操作系统 V7 中如何管理服务的通用方法，请参见 3.1 使用 `systemd` 管理系统服务。

启动 At 服务

要确定服务是否正在运行，请使用以下命令：


```
systemctl status atd.service
```

要在当前会话中运行 `atd` 服务，请以 `root` 用户在命令行提示符下输入以下命令：

```
systemctl start atd.service
```

要配置服务开机自启动，请以 `root` 用户执行以下命令：

```
systemctl enable atd.service
```

 建议您将 `atd` 服务在您的系统中配置为开机自启动。

停止 At 服务

要在当前会话中停止 atd 服务，请以 root 用户在命令行提示符下输入以下命令：

```
systemctl stop atd.service
```

要禁止服务开机自启动，请以 root 用户执行以下命令：

```
systemctl disable atd.service
```

重启 At 服务

要重启 atd 服务，请以 root 用户在命令行提示符下输入以下命令：

```
systemctl restart atd.service
```

该命令停止服务后将很快地再次启动服务。

配置 At 任务

要使用 At 工具调度一次性任务在指定时间执行，请按以下步骤操作：

1. 在命令行中输入命令 `at TIME`，这里的 *TIME* 表示命令执行的时间。

TIME 参数可以使用以下任意一种格式定义：

- **HH:MM**：指定确切的小时和分钟，例如，04:00 表示上午 4 点。
- **midnight**：指定为午夜 12 点。
- **noon**：指定为中午 12 点。
- **teatime**：指定为下午 4 点。
- **MONTHDAYYEAR** 格式：例如，January 15 2012 表示 2012 年 1 月 15 日。年份的值是可选的。
- **MMDDYY**、**MM/DD/YY** 或者 **MM.DD.YY** 格式：例如，011512 表示 2012 年 1 月 15 日。
- **now + TIME**：这里的 *TIME* 由一个整数和 minutes、hours、days 或者 weeks 几个类型值来定义。例如，now + 5 days 表示命令将在从现在起五天后的同一时间被执行。

必须首先指定时间，其后可以指定可选的日期。要了解更多关于时间格式的信息，请参考 `/usr/share/doc/at-<version>/timespec` 文本文件。

如果指定的时间已经过了，则任务将在明天的同一时间被执行。

2. 在显示出的 `at>` 命令行提示符下，定义任务命令：

A. 输入任务应该执行的命令，并按下回车键。可选地，可以重复此步骤，提供多个命令。

B. 在命令行提示符下输入一个 `shell` 脚本，并在脚本的每一行之后按下回车键。

任务将使用用户的 `SHELL` 环境变量所设置的 `shell`、用户的登录 `shell`，或者 `/bin/sh`（无论先找到哪一个）。

3. 一旦输入结束，请在一个空行中按下 `Ctrl+D` 组合键，退出命令行提示符。

如果这一系列命令或者脚本试图在标准输出中显示信息，则输出将会以电子邮件的形式发送给用户。

要查看等待执行的任务列表，请使用 `atq` 命令。请参见 5.8.8.5 查看等待执行的任务了解更多信息。

您也可以限制 `at` 命令的使用。要了解更多信息，请参见 5.8.8.7 控制对 `At` 和 `Batch` 的访问。

配置 `Batch` 任务

`Batch` 程序在系统平均负载降低到 0.8 以下的时候执行已定义的一次性任务。

要定义 `Batch` 任务，请按以下步骤进行操作：

1. 在命令行中输入 `batch` 命令。

2. 在显示出的 `at>` 命令行提示符下，定义任务命令：

A. 输入任务应该执行的命令，并按下回车键。可选地，可以重复此步骤，提供多个命令。

B. 在命令行提示符下输入一个 `shell` 脚本，并在脚本的每一行之后按下回车键。

任务将使用用户的 SHELL 环境变量所设置的 shell、用户的登录 shell，或者/bin/sh（无论先找到哪一个）。

3. 一旦输入结束，请在一个空行中按下 **Ctrl+D** 组合键，退出命令行提示符。

如果这一系列命令或者脚本试图在标准输出中显示信息，则输出将会以电子邮件的形式发送给用户。

要查看等待执行的任务列表，请使用 **atq** 命令。请参见 5.8.8.5 查看等待执行的任务了解更多信息。

您也可以限制 **batch** 命令的使用。要了解更多信息，请参见 5.8.8.7 控制对 At 和 Batch 的访问。

查看等待执行的任务

要查看等待执行的 At 和 Batch 任务，可以执行 **atq** 命令。**atq** 命令将显示一个等待执行的任务的列表，每个任务单独显示为一行。每一行的格式为任务编号、日期、任务类型和用户名称。用户只能查看他们自己的任务。如果 **root** 用户执行 **atq** 命令，则会显示所有用户的所有任务。

额外的命令行选项

at 和 **batch** 命令的额外命令行选项如下所示：

表格 5-7 **at** 和 **batch** 命令行选项

选项	描述
-f	从一个文件中读取命令或 shell 脚本，而不是在命令行提示符下输入命令或脚本。
-m	当任务完成后向用户发送电子邮件。
-v	显示任务被执行的时间。

控制对 At 和 Batch 的访问

您可以使用/etc/at.allow 和/etc/at.deny 文件来限制对 **at** 和 **batch** 命令的访问。这些访问控制文件使用相同的格式，都是每一行一个用户名。记住，两个文件都不允许使用空格。

如果存在 `at.allow` 文件, 只有在该文件中列出的用户才能使用 `at` 或者 `batch`, 并且 `at.deny` 文件将被忽略。

如果 `at.allow` 文件不存在, 则 `at.deny` 文件中列出的用户将被禁止使用 `at` 或者 `batch`。

如果访问控制文件被修改了, 不必重启 `at` 守护进程 (`atd`)。每次用户试图执行 `at` 或 `batch` 命令时, 都会读取访问控制文件。

不管访问控制文件中的内容是什么, `root` 用户都始终能够执行 `at` 和 `batch` 命令。

5.9 自动程序错误报告工具 (ABRT)

5.9.1 ABRT 简介

自动程序错误报告工具, 通常简称为 **ABRT**, 是一个设计用于帮助用户检测和报告应用程序崩溃的工具集。它的主要目的是简化报告问题和查找解决方案的过程。


ABRT 由 `abrt` 守护进程和许多用于处理、分析和报告检测到的问题的系统服务和工具组成。守护进程大多数时间都静默地在后台运行, 当检测到一个应用程序崩溃或者内核故障时才活跃起来。然后守护进程搜集相关的问题数据, 例如核心文件 (如果有的话)、崩溃应用程序的命令行参数, 以及其它数据。

ABRT 目前支持对 `C`、`C++`、`Java`、`Python` 和 `Ruby` 所编写的应用程序崩溃的检测, 以及对 `X.Org` 崩溃、内核故障和内核严重错误的检测。要了解更多有关所支持的故障和崩溃类型的信息, 以及检测众多类型的崩溃的方式, 请参见 5.9.4 检测软件问题。

处理已存在的问题数据 (与创建新的问题数据不同) 的 **ABRT** 组件是一个名为 `libreport` 的独立项目的一部分。`libreport` 库提供了一个分析和报告问题的通用机制, 它不仅被 **ABRT** 使用, 也同样被应用程序所使用。然而, **ABRT** 和 `libreport` 的操作和配置已经紧密地整合在一起了, 因此, 在本文档中将作为一个整体进行讨论。

5.9.2 安装 ABRT 并启动服务

为了使用 ABRT，请确保您的系统中已经安装了 `abrt-desktop` 包或者 `abrt-cli` 包。`abrt-desktop` 安装包为 ABRT 提供了一个图形化的用户界面，`abrt-cli` 则包含一个在命令行使用 ABRT 的工具。您可以同时安装两个包。ABRT 图形用户界面和命令行工具的通用工作流是相似的，并且遵从相同的模式。

 注意，安装 ABRT 软件包将会覆写 `/proc/sys/kernel/core_pattern` 文件，该文件包含了用于命名内核转储文件的模板。该文件的内容将被覆写为：

```
|/usr/libexec/abrt-hook-ccpp %s %c %p %u %g %t e %P %I
```

参见 2.1.2.4 安装软件包了解如何在使用 Yum 包管理器安装软件包。

安装 ABRT 图形用户界面

ABRT 图形用户界面提供了一个在桌面环境中易于使用的前端工具。您可以作为 `root` 用户来执行以下命令安装所需要的包：

```
~]# yum install abrt-desktop
```

安装之后，ABRT 通知程序将被配置为当您的图形桌面会话启动后自动启动。您可以通过在终端中执行以下命令来验证 ABRT 小程序是否正在运行：

```
~]$ ps -el | grep abrt-applet
0 S  1000 23029 22660  0  80   0 - 112932 poll_s ?
00:00:00 abrt-applet
```

如果小程序没有运行，您可以通过运行 `abrt-applet` 程序来在您当前的桌面会话中手动启动它：

```
~]$ abrt-applet &
[1] 2261
```

安装 ABRT 命令行工具

命令行界面对于没有图形界面的机器、通过网络连接的远程系统或者在脚本中，是很有用的。您可以作为 `root` 用户来执行以下命令安装所需要的包：


```
~]# yum install abrt-cli
```

安装 ABRT 附属工具

要接收 ABRT 检测到的关于崩溃的电子邮件通知，您需要安装 `libreport-plugin-mailx` 软件包。您可以作为 `root` 用户来执行以下命令安装所需要的包：

```
~]# yum install libreport-plugin-mailx
```

默认情况下，它将在本地机器上给 `root` 用户发送通知。电子邮件目的地可以在 `/etc/libreport/plugins/mailx.conf` 文件中进行配置。

要在登录时将通知显示在您的控制台上，请安装上 `abrt-console-notification` 软件包。

ABRT 能够检测、分析并报告多种类型的软件故障。默认情况下，ABRT 安装支持最通用的故障类型，例如 C 和 C++ 应用程序的崩溃。对其它故障类型的支持通过独立软件包来提供。例如，要安装支持对 Java 语言编写的应用程序中的异常的检测，请以 `root` 用户执行以下命令：

```
~]# yum install abrt-java-connector
```

参见 5.9.4 检测软件问题获取 ABRT 所支持的预研和软件项目列表。该节也包括了一个要启用多种故障类型的检测所需要安装的对应的包的列表。

启动 ABRT 服务

`abrt-d` 守护进程被配置为开机自启动。您可以使用以下命令来验证它当前的状态：

```
~]$ systemctl is-active abrt-d.service  
active
```

如果 `systemctl` 命令返回了 `inactive` 或者 `unknown`，则表示守护进程没有运行。您可以作为 `root` 用户执行以下命令来在当前会话中启动它：

```
~]# systemctl start abrt-d.service
```


类似地，您可以通过相同的步骤来检查处理各种故障类型的服务的运行状态，并启动它们。例如，如果您希望 ABRT 能够检测 C 或者 C++ 崩溃，请确保 `abrt-ccpp` 服务正在运行。请参见 5.9.4 检测软件问题，获取所有可用的 ABRT 检测服务及它们各自对应的安装包的列表。

除了 `abrt-vmcore` 和 `abrt-pstoreoops` 服务是在内核严重故障或内核故障实际发生时才被启动外，所有的其它 ABRT 服务都是在它们各自的安装包被安装时，就被设置为开机启动，并立即启动的。就如 3.1 使用 `systemd` 管理系统服务所描述的那样，您可以使用 `systemctl` 工具禁用或启用任何 ABRT 服务。

测试 ABRT 崩溃检测

要测试 ABRT 是否能正确地工作，可以使用 `kill` 命令发送一个 `SEGV` 信号来终止一个进程。例如，按照下述方式启动一个 `sleep` 进程，并使用 `kill` 命令终止该进程：

```
~]$ sleep 100 &
[1] 2823
~]$ kill -s SEGV 2823
```

ABRT 在执行 `kill` 命令之后，很快就检测到了崩溃的发生，假若正在运行图形会话，用户将会被图形用户界面通知小程序通告检测到的问题。在命令行环境中，您可以使用 `abrt-cli list` 命令来检查是否检测到了崩溃的发生，或者通过检查在 `/var/tmp/abrt/` 目录里创建的崩溃转储来确定。请参见 5.9.5 处理检测到的问题了了解更多有关如何处理检测到的崩溃的信息。

5.9.3 配置 ABRT

在 ABRT 中，一个问题的生命周期是通过事件来驱动的。例如：

- 事件#1——创建问题数据目录。
- 事件#2——分析问题数据。
- 事件#3——报告问题。

无论何时检测到一个问题之后，ABRT 将该问题和所有已存在的问题数据进行比较，确定是否是已经被记录的相同的问题。如果是相同的问题，则更新已存在的问题数据，并且最近的（重复的）问题不会被再次记录。如果不是相同的问

题，则会创建一个问题数据目录。一个问题数据目录通常由以下文件组成：
analyzer、architecture、coredump、cmdline、executable、kernel、os_release、reason、
time 和 uid。

其它文件，例如 **backtrace**，将会在问题的分析过程中被创建，这取决于所使用的分析方法和它的配置设定。这些文件中的每一个都各自保存着有关系统和问题本身的特定信息。例如，**kernel** 文件里记录了崩溃内核的版本。

在创建了问题数据目录，并收集了问题数据之后，您可以通过 **ABRT** 图形用户界面或者 **abrt-cli** 命令行工具来处理问题。请参见 5.9.5 处理检测到的问题了解更多关于 **ABRT** 所提供的处理已记录问题的工具的信息。

配置事件

ABRT 事件使用插件来实现实际的报告操作。插件是事件用来调用处理问题数据目录的小工具。通过使用插件，**ABRT** 能够向不同的目的地报告问题，而几乎每一个报告目的地都需要一些配置。例如，**Bugzilla** 需要一个用户名、密码，以及一个指向 **Bugzilla** 服务实例的网址。

一些配置细节可以使用默认值（例如 **Bugzilla** 网址），但是其他配置细节则没有合理的默认值（例如用户名）。**ABRT** 在配置文件中查找这些设定，例如 **report_Bugzilla.conf**，该文件在 **/etc/libreport/events/** 或者 **\$HOME/.cache/abrt/events/** 目录中分别代表系统范围的设置和特定用户的设置。配置文件中包含了成对的指令和值。

这些文件是运行事件和处理问题数据所必需的。**gnome-abrt** 和 **abrt-cli** 工具从这些文件中读取配置数据，并传递给它们所运行的事件。

关于事件的额外信息（例如它们的描述、名称、能够作为环境变量传递给它们的参数类型，以及其它属性）保存在 **/usr/share/libreport/events/** 目录下的 **event_name.xml** 文件里。**gnome-abrt** 和 **abrt-cli** 都使用了这些文件，以使用户界面更友好。请不要编辑这些文件，除非您想改变标准安装。如果您确实打算这样做，请将要修改的文件复制到 **/etc/libreport/events/** 目录下，并修改新复制的文件。这些文件可能包含以下信息：

- 易于使用的事件名称和描述

- 问题数据目录中需要事件处理的条目列表
- 要发送或者不发送的条目的默认和强制选择
- 图形用户界面是否应该提示数据审查
- 存在的配置选项，它们的类型（string、Boolean 等）、默认值、

提示字符串等；这可以让图形用户界面构造出适当的配置对话框

例如，report_Logger 事件可以接受一个输出文件名作为参数。通过使用各自的 *event_name.xml* 文件，ABRT 图形用户界面可以确定哪些参数可以指定给一个选定的事件，并允许用户为这些参数设置参数值。这些值被 ABRT 图形用户界面保存，并在后续对这些事件的调用中重用。注意，ABRT 图形用户界面使用 GNOME Keyring 工具来保存配置选项，并且将它们传递给事件，这将覆盖从文本配置文件中获取的数据。

要打开图形化的【配置】窗口，请在 `gnome-abrt` 应用程序的运行实例中选择【自动程序错误报告工具】→【首选项】。这个窗口会显示在使用图形用户界面的报告过程中，可以被选择事件列表。当您选中了其中一个可配置的事件时，您可以点击【配置】按钮，为事件修改设定。

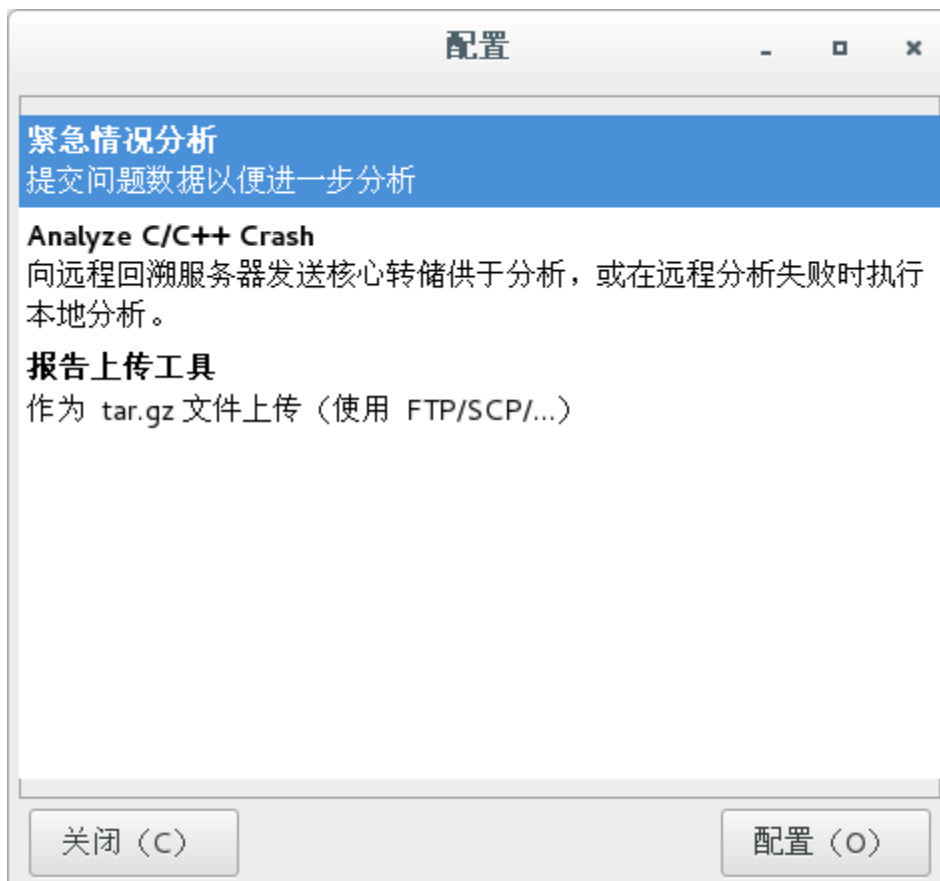



图 5-11 配置 ABRT 事件

 /etc/libreport/目录层级下的所有文件都是全面可读的，被用来作为全局设置。因此，不建议在其中保存用户名、密码或者任何其它敏感数据。针对用户的设置（在 GUI 应用程序中设置，只能被\$HOME 的所有者读取）安全地保存在 GNOME Keyring 中，如果使用的是 abrt-cli，这些设置也可以保存在\$HOME/.abrt/目录下的文本配置文件中。

下面的表格显示了 ABRT 标准安装所提供的对默认的分析、收集和报告事件的选择。表格中列出了每个事件的名称、标识符、/etc/libreport/events.d/目录下的配置文件，以及一个简要的描述。注意，由于配置文件使用的是事件标识符，ABRT 图形用户界面使用它们的名称来引用各个事件。另外需要注意的是，不是所有的事件都可以使用图形用户界面进行设置。要了解如何定义自定义事件，请参考“创建自定义事件”章节。

表格 5-8 标注 ABRT 事件

名称	标识符和配置文件	描述
uReport	report_uReport	上传一个 μ Report 到 FAF 服务器上。
Mailx	report_Mailx mailx_event.conf	通过 Mailx 工具把问题报告发送到一个指定的电子邮件地址。
Bugzilla	report_Bugzilla bugzilla_event.conf	将问题报告给指定的 Bugzilla。
Emergency analysis	report_EmergencyAnalysis emergencyanalysis_event.conf	上传一个压缩包到 FAF 服务器上进一步分析。在标准报告方法失败时使用。
Analyze C or C++ Crash	analyze_CCpp ccpp_event.conf	将核心转储发送到一个远程追踪服务器上进行分析，或者远程失效的话执行本地分析。
Report uploader	report_Uploader uploader_event.conf	使用 FTP 或 SCP 协议将包含问题数据的压缩归档文件（.tar.gz）上传到所选的目的地。
Analyze VM core	analyze_VMcore vmcore_event.conf	在内核故障的问题数据上运行 GDB(GNU 调试器)，生成一个内核回溯 (backtrace)。
Local GNU Debugger	analyze_localGDB ccpp_event.conf	在应用程序的问题数据上运行 GDB(GNU 调试器)，生成一个程序回溯

名称	标识符和配置文件	描述
		(backtrace)。
Collect.xsession-errors	analyze_xsession_errors ccpp_event.conf	将 ~/.xsession-errors 文件的相关行保存到问题报告中。
Logger	report_Logger print_event.conf	创建一个问题报告，并将其保存到一个指定的本地文件中。
Kerneloops.org	report_Kerneloops koops_event.conf	将内核问题发送到 kerneloops.org 上的故障追踪系统。

创建自定义事件

每一个事件都是在分别的配置文件中通过一个规则结构来定义的。配置文件通常保存在 /etc/libreport/events.d/ 目录下。这些配置文件都是通过主配置文件 /etc/libreport/report_event.conf 来加载的。该文件可以接受 shell 元字符 (*、\$、? 等)，并可以相对于其位置来解释相对路径。

每一个规则以一个非空格前导字符行开始，随后的所有以空格符或制表符开始的行都被认为是这个规则的一部分。每一个规则由两个部分组成，一个条件部分和一个程序部分。条件部分包含以下列形式之一来表示的条件：

- VAR=VAL
- VAR!=VAL
- VAL~=REGEX

在这里：

- VAR 要么是 EVENT 关键字，要么是一个问题数据目录元素的名
称（例如 executable、package、hostname 等）
- VAL 要么是一个事件名称，要么是一个问题数据元素
- REGEX 是一个正则表达式

程序部分由程序名称和 shell 可解释代码组成。如果条件部分的所有条件都

是有效的，则程序部分将在 **shell** 中运行。以下是一个事件的示例：

```
EVENT=post-create date > /tmp/dt
echo $HOSTNAME `uname -r`
```

这个事件将会用当前的日期和时间重写/tmp/dt 文件的内容，并将机器的主机名和它的内核版本打印到标准输出中。

接下来是一个更复杂的事件的示例，它实际是预先定义的事件之一。它将 ~/.xsession-errors 文件的相关行保存到 abrt-ccpp 服务用来处理的任何问题的问题报告中，这里假定崩溃的应用程序在崩溃时加载过任意 X11 库：

```
EVENT=analyze_xsession_errors analyzer=CCpp dso_list=~.*libX11.*
test -f ~/.xsession-errors || { echo "No ~/.xsession-errors"; exit 1; }
test -r ~/.xsession-errors || { echo "Can't read ~/.xsession-errors";
exit 1; }
executable=`cat executable` &&
base_executable=${executable##*/} &&
grep -F -e "$base_executable" ~/.xsession-errors | tail
-999 >xsession_errors &&
echo "Element 'xsession_errors' saved"
```

可能的事件集没有被限定。系统管理员可以根据他们的需要添加事件。目前，标准的 ABRT 和 libreport 安装提供了以下事件名称：

post-create

这个事件被 **abrt-d** 用来处理新创建的问题数据目录。当 **post-create** 事件运行时，**abrt-d** 检查新的问题数据的 **id** 是否和任何已存在的问题目录的 **id** 匹配。如果存在这样一个问题目录，则新的问题数据将被删除。

notify, notify-dup

notify 事件在 **post-create** 完成之后运行。当该事件运行时，用户可以确定问题是值得他们注意的。**notify-dup** 是类似的，只不过它用于相同问题的重复发生。

`analyze_name_suffix`

这里的 `name_suffix` 是事件名称的可替换部分。该事件用来处理收集的数据。例如，`analyze_LocalGDB` 事件使用 GNU 调试器（GDB）工具来处理一个应用程序的核心转储，并生成崩溃的回溯。

`collect_name_suffix`

这里的 `name_suffix` 是事件名称的可替换部分。该事件用来收集问题的额外信息。

`report_name_suffix`

这里的 `name_suffix` 是事件名称的可替换部分。该事件用来报告问题。

设置自动报告


ABRT 可以被配置来自动发送任何检测到的问题或者崩溃的最初的匿名报告或者 μ Reports，而不用任何用户干预。当自动报告被开启后，通常在崩溃报告过程的开始阶段发送的所谓的 μ Report，将在检测到崩溃后立即发送。这可以防止基于相同的崩溃进行重复支持。要启用自动报告功能，请以 `root` 用户执行以下命令：

```
~]# abrt-auto-reporting enabled
```

以上命令将 `/etc/abrt/abrt.conf` 配置文件中的 `AutoreportingEnabled` 指令设置为 `yes`。这一系统范围的设定将应用到系统中的所有用户。注意，通过启用该选项，自动报告在图形桌面环境中也将被启用。要只启用 ABRT 图形用户界面中的自动报告，请在【配置错误报告】窗口中将【自动发送 `uReport`】选项的开关切换为【YES】。要打开这个窗口，请在 `gnome-abrt` 应用程序的运行实例中选择【自动程序错误报告工具】→【ABRT 配置】。要运行该应用程序，请选择【应用程序】→【杂项】→【自动程序错误报告工具】。



图 5-12 配置 ABRT 问题报告

 **μReport**（微报告）是表示一个诸如二进制崩溃或者内核故障问题的 JSON 对象。这些报告设计得很简要、机器可读，并且完全匿名，这也是为什么它们能用于自动报告。**μReports** 使得对缺陷发生的追踪成为可能，但是它们通常不能提供足够的信息给工程师修复缺陷。

要改变自动报告功能发送 **μReport** 的默认行为，请修改 `/etc/abrt/abrt.conf` 配置文件中的 `AutoreportingEvent` 指令，让其指向一个不同的 ABRT 事件。请参见表格 5-8 标准 ABRT 事件了解标准事件的概览。

5.9.4 检测软件问题

ABRT 能够检测、分析和处理使用多种不同的编程语言编写的应用程序崩溃。许多支持检测不同类型的崩溃的软件包，在安装主要的 ABRT 软件包（`abrt-desktop`、`abrt-cli`）时，就已经被安装上了。请参见 5.9.2 安装 ABRT 并启动服务了解如何安装 ABRT。查看下面的表格了解所支持的崩溃类型和各自对应的软件包。

表格 5-9 支持的编程语言和软件项目

语言/项目	安装包
C 或者 C++	abrt-addon-ccpp
Python	abrt-addon-python
Ruby	rubygem-abrt
Java	abrt-java-connector
X.Org	abrt-addon-xorg
Linux（内核故障）	abrt-addon-kerneloops
Linux（内核严重故障）	abrt-addon-vmcore
Linux（永久存储）	abrt-addon-pstoreoops

检测 C 和 C++崩溃

abrt-ccpp 服务安装了它自己的核心转储处理程序，当该服务被启动后，将会覆盖内核的 `core_pattern` 变量的默认值，从而让 `abrt-d` 来处理 C 和 C++崩溃。如果您停止了 abrt-ccpp 服务，之前指定的 `core_pattern` 变量值将重新恢复为默认值。

默认情况下，`/proc/sys/kernel/core_pattern` 文件的内容包含了字符串 `core`，意味着内核将在崩溃进程的当前目录下生成带有“core.”前缀的文件。abrt-ccpp 服务将重写 `core_pattern` 文件的内容，使其包含如下命令：

```
|/usr/libexec/abrt-hook-ccpp %s %c %p %u %g %t e %P %I
```

该命令指示内核将核心转储用管道连接至 `abrt-hook-ccpp` 程序，该程序会将其保存到 ABRT 的转储位置，并通知 `abrt-d` 进程有新的崩溃。出于调试的目的，它也会从 `/proc/PID/` 目录中保存以下文件：`maps`、`limits`、`cgroup`、`status`。请参见 `proc(5)` 获取有关格式的描述和这些文件的含义。

检测 Python 异常

abrt-addon-python 安装包为 Python 应用程序安装了一个自定义的异常处理程序。Python 解释器将会自动导入安装到 `/usr/lib64/python2.7/site-packages/` 目录下的 `abrt.pth` 文件，`abrt.path` 文件又会导入 `abrt_exception_handler.py`。这样，Python

的默认的 `sys.excepthook` 将被自定义的处理程序覆盖，处理程序通过它的 `Socket` API 将未处理的异常转发给 `abrt`。

要禁用自动导入特定位置的模块，从而阻止 `Python` 程序在运行时使用 `ABRT` 自定义的异常处理程序，请传递 `-S` 选项给 `Python` 解释器：

```
~]$ python -S file.py
```

在上述命令中，请将 `file.py` 替换为在不使用特定位置模块的情况下您想执行的 `Python` 脚本的名称。

检测 Ruby 异常

`rubygem-abrt` 安装包使用 `at_exit` 特性注册了一个自定义的处理程序，该处理程序将在一个程序结束时被执行。这样可以允许对未处理异常进行检查。每次捕获到一个未处理异常时，`ABRT` 处理程序就会准备一个缺陷报告，该报告可通过标准的 `ABRT` 工具进行提交。

检测 Java 异常

`ABRT` `Java` 连接器是一个将未捕获的 `Java` 异常报告给 `abrt` 的 `JVM` 代理。此代理注册了多个 `JVMTI` 事件回调函数，并通过使用 `-agentlib` 命令行参数加载到 `JVM` 中。注意，处理注册的回调函数会影响应用程序的性能。可以使用以下命令来让 `ABRT` 从一个 `Java` 类中捕获异常：

```
~]$ java -agentlib:abrt-java-connector[=abrt=on] $MyClass  
-platform.jvmtiSupported true
```

在上述命令中，使用您想要测试的 `Java` 类的名称来替换 `$MyClass`。通过传递“`abrt=on`”选项给连接器，您可以确保异常将交由 `abrt` 来处理。如果您想让连接器将异常输出到标准输出中，请省略该选项。

检测 X.Org 崩溃

`abrt-xorg` 服务从 `/var/log/Xorg.0.log` 文件中收集并处理关于 `X.Org` 服务崩溃的信息。注意，如果加载了被加入黑名单的 `X.org` 模块，则不会生成报告。取而代之，将会在问题数据目录中创建一个带有适当解释的不可报告文件。您可以在 `/etc/abrt/plugins/xorg.conf` 文件中找到一个黑名单模块的列表。默认情况下，只有

专有的图形驱动模块会被加入黑名单。

检测内核故障和严重故障

通过检查内核日志的输出，ABRT 能够捕获和处理所谓的内核故障——Linux 内核的正确行为的非致命偏差。此功能由 `abrt-oops` 服务提供。


ABRT 也能够使用 `abrt-vmcore` 服务来检测和处理内核严重故障——致命的、不可恢复的错误，需要重启系统来解决。该服务只有当在 `/var/crash/` 目录中出现 `vmcore` 文件时才会启动。当找到核心转储文件时，`abrt-vmcore` 将在 `/var/tmp/abrt/` 目录中创建一个新的问题数据目录，并将核心转储文件移动到新创建的问题数据目录中。在搜索完 `/var/crash/` 目录之后，该服务被停止。

为了让 ABRT 能够检测内核严重故障，必须在系统中启用 `kdump` 服务。必须正确地设置好为 `kdump` 内核预留的内存量。您可以使用 `system-config-kdump` 图形化工具来设置它，也可以在 GRUB 菜单的内核选项列表中通过 `crashkernel` 参数来指定。要了解如何修改 GRUB 菜单，请参见 6.1 使用 GRUB2 启动加载工具。

在支持 `pstore` 的系统上，通过使用 `abrt-pstoreoops` 服务，ABRT 能够收集和处理有关内核严重故障的信息，该信息保存在自动挂载的 `/sys/fs/pstore/` 目录下。和平台相关的 `pstore` 接口（永久存储）提供了一种在系统重启时存储数据的机制，从而能够保留内核严重故障的信息。该服务将在 `/sys/fs/pstore/` 目录中出现内核崩溃转储文件时自动启动。

5.9.5 处理检测到的问题

`abrt-d` 所保存的问题数据可以被查看、报告和删除，您可以通过命令行工具 `abrt-cli` 或者图形化工具 `gnome-abrt` 来完成这些操作。

 注意，ABRT 通过将新的问题和所有本地保存的问题进行比较，来识别重复问题。对于重复的崩溃，ABRT 只需要您对其操作一次。然而，如果您删除了那个问题的崩溃转储，那么下一次同样的问题发生时，ABRT 将把它作为新的崩溃来处理：ABRT 将会向您通告该问题，提示您填充描述信息，并报告该问题。为了避免 ABRT 向您通告重复发生的问题，请不要删除它的问题数据。

使用命令行工具

在命令行环境中，用户在登录时被通告新的崩溃信息，这里假设用户已经安装了 `abrt-console-notification` 软件包。控制台通告看起来如下所示：

```
ABRT has detected 1 problem(s). For more info run: abrt-cli list --since
1398783164
```

要查看检测到的问题，请输入 `abrt-cli list` 命令：

```
~]$ abrt-cli list

id d3b541e2a3677456041e97522ed84d1b806dae24

reason:      sleep killed by SIGSEGV
time:        2015 年 12 月 23 日 星期三 10 时 28 分 56 秒
cmdline:     sleep 100
package:     coreutils-8.22-15.el7
uid:         0 (root)
count:       1
Directory:   /var/spool/abrt/ccpp-2015-12-23-10:28:56-21188
```

已禁用自动报告功能。请考虑启用该功能，方法是
作为有 `root` 特权的用户使用命令 `'abrt-auto-reporting enabled'`

在 `abrt-cli list` 命令的输出结果中列出的每一个崩溃，都有一个唯一标识符和一个目录，可以在使用 `abrt-cli` 做进一步的操作时用到它们。

要查看某个特定问题的信息，可以使用 `abrt-cli info` 命令：

```
abrt-cli info [-d] directory_or_id
```

要在使用 `list` 和 `info` 子命令时增加显示的信息量，请使用 `-d` (`--detailed`) 选项，该选项将会显示列出的问题的所有已保存的信息，包括各自的 `backtrace` 文件（如果已经生成了该文件的话）。

要分析并报告一个特定问题，请使用 `abrt-cli report` 命令：

```
abrt-cli report directory_or_id
```

如果您确定您不想报告某个特定的问题，您可以删除该问题。要删除一个问题，以便 ABRT 不再保留关于该问题的信息，请执行命令：

```
abrt-cli rm directory_or_id
```

要显示某个 `abrt-cli` 命令的帮助信息，请使用 `--help` 选项：

```
abrt-cli command --help
```

使用图形用户界面

无论何时一个问题报告被创建时，ABRT 守护进程将广播一个 D-Bus 消息。如果在一个图形化桌面环境中运行了 ABRT 小程序，它将捕获这个消息，并在桌面上显示一个通告对话框。您可以通过点击这个对话框上的【报告】按钮来打开 ABRT 的图形用户界面。您也可以通过选择【应用程序】→【杂项】→【自动程序错误报告工具】菜单项来打开 ABRT 的图形用户界面。

可选地，您可以在命令行中使用以下命令来运行 ABRT 的图形用户界面：

```
~]$ gnome-abrt &
```

ABRT 的图形用户界面窗口显示了一个已检测到问题的列表。每个问题条目由故障应用程序的名称、应用程序崩溃的原因，以及问题上一次发生的日期组成。



图 5-13 ABRT 图形用户界面

要访问更详细的问题描述，请双击要查看的问题或者选中列表中的问题后点击【报告】按钮。要删除一个问题，请点击【删除】按钮。

5.10 Oprofile

Oprofile 是一个低开销、系统范围的性能监视工具。它使用处理器上的性能监视硬件获取系统上关于内核和可执行程序的信息，例如当内存被引用的时候，L2 缓存的请求数，以及接收到的硬件中断数。在中标麒麟高级服务器操作系统中，要使用此工具必须安装上 oprofile 软件包。

许多处理器包含专用的性能监视硬件。该硬件使得检测什么时候发生了某些确定的事件成为可能（例如请求的数据不在缓存中）。该硬件通常表现为计数器的形式，每次一个事件发生时计数器的值就会增加。当计数器的值增加时，会产生一个中断，从而能够控制性能监视所产生的开销。

OProfile 使用这个硬件（如果没有性能监视硬件则使用一个基于计时器的替

代品) 来在每次计数器产生中断时收集性能相关的数据样本。这些取样将周期性地写入到磁盘中，之后，这些取样中所包含的数据能够被用来生成系统级和应用程序级的性能报告。

在使用 Oprofile 时请注意以下限制：

- 共享库的使用——共享库代码的取样没有被归属于特定的应用程序，除非使用了 `--separate=library` 选项。
- 性能监视取样是不精确的——当一个性能监视寄存器触发一次取样时，中断处理不想被零除异常那样精确。由于处理器指令的无序执行，取样可能被记录在一个邻近的指令中。
- `opreport` 没有正确地为内联函数关联取样——`opreport` 使用一个简单的地址范围机制来确定地址中的函数。内联函数取样没有被归属于内联函数本身，而是被归属于了内联函数被插入的那个函数。
- OProfile 多次运行后累积了数据——OProfile 是一个系统范围的评测器，通常会预期进程将启动和停止多次。于是，取样将会因为多次运行而累积。可以使用命令 `opcontrol --reset` 来清除之前的取样。
- 硬件性能计数器不适用于虚拟机操作系统——因为硬件性能计数器在虚拟机操作系统中不可用，所以您需要使用 `timer` 模式。输入命令 `opcontrol --deinit`，然后执行 `modprobe oprofile timer=1` 来启用 `timer` 模式。
- 非 CPU 受限性能问题——OProfile 面向于查找 CPU 受限进程的问题。OProfile 不会去识别因为正在等待锁或者其它一些事件发生（例如等待一个 IO 设备完成操作）而处于睡眠状态的进程。

5.10.1 工具概览

表格 5-10 OProfile 命令提供了 `oprofile` 安装包中最常用的工具的一个简要概览。

表格 5-10 常用命令

命令	描述
----	----

命令	描述
ophelp	显示系统处理器可用的事件，每个事件都有一个简要的描述。
opimport	将样本数据库文件从外部二进制格式转换为系统本地格式。只有在评测不同架构的样本数据库时需要使用此命令。
opannotate	如果应用程式是使用调试标志编译的，则为可执行文件创建带注释的源代码。参见 5.10.2 使用 operf 了解详情。
opcontrol	配置需要收集的数据。参见 5.10.3 使用遗留模式配置 OProfile 了解详情。
operf	推荐用来替换 opcontrol 的评测工具。参见 5.10.2 使用 operf 了解详情。要了解 operf 和 opcontrol 之间的不同请参见 5.10.1.1 operf 和 opcontrol。
opreport	获取评测数据。参见 5.10.6.1 使用 opreport 了解详情。
oprofiled	以守护进程来运行，周期性地把取样数据写到磁盘。

operf 和 opcontrol

在使用 OProfile 收集评测数据时，有两种互斥的方法。您可以使用较新的、更推荐使用的 operf，也可以使用 opcontrol。

operf

这是推荐的评测模式。operf 工具使用 Linux 性能事件子系统，因此不需要oprofile 内核驱动。operf 工具可以让您的性能评测更加精确，无论是对单个进程还是系统范围；也能够让 OProfile 和您系统上其它使用性能监视硬件的工具更好地共存。不像 opcontrol，operf 无需 root 权限就可以使用。不过，operf 在使用 --system-wide 选项进行系统范围的操作时，仍然需要 root 授权。

operf 不需要最初的配置。您可以在调用 operf 时使用命令行选项来指定您的评测设定。之后，您可以运行在 5.10.6 分析数据中描述的后处理工具。请参见 5.10.2 使用 operf 了解更多信息。

opcontrol

此模式由 opcontrol shell 脚本、oprofiled 守护进程和几个后处理工具组成。opcontrol 命令用来配置、启动和停止一个评测会话。一个 OProfile 内核驱动（通常被构建为一个内核模块）被用来收集样本，这些样本被 oprofiled 记录在样本文件中。只有当您具有 root 权限时，您才可以使用此遗留模式。在某些确定的情况下，例如当您需要对禁用中断请求的区域取样时，此模式是一个比较好的替换方案。

在 OProfile 使用此遗留模式运行前，必须先按照 5.10.3 使用遗留模式配置 OProfile 所展示的那样进行配置。之后启动 Oprofile 时将会应用这些设定（5.10.4 启动和停止 OProfile 遗留模式）。

5.10.2 使用 operf

operf 是推荐使用的评测模式，它在启动前不需要初始化设置。所有的设置都以命令行选项的形式指定，它也没有单独的启动评测进程的命令。要停止 operf，请按 Ctrl+C 组合键。典型的 operf 命令语法如下所示：

```
operf options range command args
```

使用需要的命令行选项替换 *options*，来指定您的评测设置。在 operf(1) 用户手册页面描述了完整的选项。使用以下之中的一个来替换 *range*：

--system-wide: 该选项允许进行全局评测。

--pid=*PID*: 该选项用来评测一个正在运行的应用程序，这里的 *PID* 是您想评测的进程的进程 ID。

通过 *command* 和 *args*，您可以定义一个要评测的命令或应用程序，以及该命令或应用程序需要输入的参数。*command* 需要 --pid 或者 --system-wide 选项，但是这两个选项不能同时使用。

当您在命令行中调用 operf 时不使用 *range* 选项，则将收集子进程的数据。

 要运行 operf --system-wide，您需要 root 授权。当评测完成后，

您可以使用 **Ctrl+C** 组合键来停止 **operf**。

如果您运行 **operf --system-wide** 时，是作为一个后台进程（使用了**&**）来运行的，需要以一种可控的方式来停止该进程，以便处理收集到的评测数据。要停止该后台进程，请使用以下命令：

```
kill -SIGINT operf-PID
```

在运行 **operf --system-wide** 命令时，建议您的当前工作目录应该是 **/root** 或者 **/root** 下的一个子目录，以便样本数据文件不会被存储在一个普通用户能够访问的位置。

指定内核

要监视内核，请执行以下命令：


```
operf --vmlinux=vmlinux_path
```

使用这个选项，您可以指定一个和正在运行的内核相匹配的 **vmlinux** 文件的路径。内核取样将会被归属于这个二进制文件，从而允许后处理工具将样本归属于合适的内核符号。如果没有指定该选项，则所有的内核取样都将被归属于一个名为“**no-vmlinux**”的伪二进制文件。

设定监视事件

大多数处理器包含了计数器，**OProfile** 用它们来监视特定的事件。如表格 5-12 **OProfile** 处理器和计数器所展示的，可用的计数器的数量取决于处理器。


每一个计数器的事件可以通过命令行或者图形界面来配置。要了解关于图形界面的更多信息，请参见 5.10.9 图形界面。如果计数器不能设定一个特定的事件，将会显示一个错误信息。

 底层的 **Linux** 性能事件子系统内核不支持一些比较旧的处理器模型，因此 **operf** 也不支持这些处理器模型。如果您在尝试使用 **operf** 时接收到如下所示的消息：

```
Your kernel's Performance Events Subsystem does not support your
processor type
```

可以试着用 **opcontrol** 来评测，看看也许 **OProfile** 的遗留模式能够支

持您的处理器类型。

 因为硬件性能计数器在虚拟机操作系统中不可用，您必须在虚拟机操作系统中启用 timer 模式来使用 `opperf`。要启用 timer 模式，请以 root 用户执行以下命令：

```
opcontrol --deinit

modprobe oprofile timer=1
```

要通过命令行为每个可配置的计数器设定事件，请使用如下命令：

```
opperf --events=event1,event2...
```

在这里，将为评测传递一个以逗号分隔的事件规范列表。每一个事件规范是一个以冒号分隔的属性列表，格式如下：

```
event-name:sample-rate:unit-mask:kernel:user
```

表格 5-11 事件规范事件规范对这些属性做了总结。最后三个属性是可选的，如果您省略了它们，它们将被设定为各自的默认值。注意，某些事件是需要单元掩码的。

表格 5-11 事件规范

属性	描述
event-name	从 <code>ophelp</code> 中获取到的确切的符号名称。
sample-rate	再次取样前需要等待的事件数。次数越小，取样就越频繁。对于不会频繁发生的事件，可能需要设置一个较小的值，以便在统计上能够捕获足够多的事件实例。另一方面，取样太频繁会使系统超载。默认情况下，OProfile 采用基于时间的事件设置，它会每个处理器每 100000 个时钟周期创建一个样本。
unit-mask	单元掩码用来进一步定义事件，可以用 <code>ophelp</code>

属性	描述
	命令来列举事件的单元掩码。您可以使用一个以“0x”开头的十六进制值，或者和 ophelp 命令中的单元掩码描述中的名称相匹配的字符串来指定单元掩码。通过名称来定义只对那些在 ophelp 的输出结果中显示的那样，具有“extra:”参数的单元掩码有效。这一类的单元掩码不能通过十六进制值来定义。注意，在某些架构中，可能会有多个单元掩码具有相同的十六进制值。在那种情况下，它们只能通过名称来指定。
kernel	指定是否评测内核代码（值为 0 或 1，默认值为 1）。
user	指定是否评测用户空间代码（值为 0 或 1，默认值为 1）。

可用的事件根据处理器类型的不同而不同。如果没有指定事件规范，则将使用正在运行的处理器类型的默认事件来进行评测。请参见表格 5-13 默认事件来了解默认事件。要确定可用于评测的事件，请使用 ophelp 命令：

ophelp

样本分类

--separate-thread 选项通过线程组 ID (tgid) 和线程 ID (tid) 来对样本分类。这在多线程应用程序中查看每个线程的取样时很有用。当其与--system-wide 选项联合使用时，--separate-thread 也很有用，比如在一次评测运行的过程中，有多个进程在执行同一个程序，该选项就可以查看每个进程（每个线程组）的取样。

--separate-cpu 选项通过 CPU 来对样本分类。

5.10.3 使用遗留模式配置 OProfile

OProfile 必须先加以配置才能以遗留模式运行。至少需要配置选择监视内核（或选择不监视内核）。下一小节描述如何使用 opcontrol 工具来配置 OProfile。当 opcontrol 命令被执行的时候，设定选项将被保存到/root/.oprofile/daemonrc 文

件中。

指定内核


首先，配置 OProfile 是否应该监视内核。这是在启动 OProfile 前唯一需要配置的选项。其它选项的配置都是可选的。

要监视内核，请以 root 用户执行以下命令：

```

opcontrol --setup --vmlinux=/usr/lib/debug/lib/modules/^uname
-r`/vmlinux

```

 为了监视内核，必须安装上包含了未压缩的内核的 kernel-debuginfo 安装包。

要配置 OProfile 不监视内核，请以 root 用户执行以下命令：

```

opcontrol --setup --no-vmlinux

```

该命令将会加载 oprofile 内核模块（如果该模块尚未加载的话），并创建 /dev/oprofile/ 目录（如果该目录还不存在的话）。请参见 5.10.7 理解/dev/oprofile/ 目录了解有关该目录的细节。

设置样本是否应该在内核中收集只会改变所收集的数据，而不会改变收集数据的方法或贮存地点。要为内核和应用程序库生成不同的样本文件，请参见 5.10.3.3 分离内核和用户空间评测。

设置要监视的事件

大多数处理器包含了计数器，OProfile 用它们来监视特定的事件。如表格 5-12 OProfile 处理器和计数器所展示的，可用的计数器的数量取决于处理器。

表格 5-12 OProfile 处理器和计数器

处理器	cpu_type	计数器数量
AMD64	x86-64/hammer	4
AMD Family 10h	x86-64/family10	4
AMD Family 11h	x86-64/family11	4
AMD Family 12h	x86-64/family12	4

处理器	cpu_type	计数器数量
AMD Family 14h	x86-64/family14	4
AMD Family 15h	x86-64/family15	6
Applied Micro X-Gene	arm/armv8-xgene	4
ARM Cortex A53	arm/armv8-ca53	6
ARM Cortex A57	arm/armv8-ca57	6
IBM eServer System i and IBM eServer System p	timer	1
IBM POWER4	ppc64/power4	8
IBM POWER5	ppc64/power5	6
IBM PowerPC 970	ppc64/970	8
IBM PowerPC 970MP	ppc64/970MP	8
IBM POWER5+	ppc64/power5+	6
IBM POWER5++	ppc64/power5++	6
IBM POWER6	ppc64/power6	6
IBM POWER7	ppc64/power7	6
IBM POWER8	ppc64/power8	8
IBM S/390 and IBM System z	timer	1
Intel Core i7	i386/core_i7	4
Intel Nehalem microarchitecture	i386/nehalem	4
Intel Westmere microarchitecture	i386/westmere	4
Intel Haswell microarchitecture (nonhyper-threaded)	i386/haswell	8
Intel Haswell microarchitecture (hyperthreaded)	i386/haswell-ht	4
Intel Ivy Bridge microarchitecture (nonhyper-threaded)	i386/ivybridge	8
Intel Ivy Bridge microarchitecture	i386/ivybridge-ht	4

处理器	cpu_type	计数器数量
(hyperthreaded)		
Intel Sandy Bridge microarchitecture (non-hyper-threaded)	i386/sandybridge	8
Intel Sandy Bridge microarchitecture	i386/sandybridge-ht	4
Intel Broadwell microarchitecture (nonhyper-threaded)	i386/broadwell	8
Intel Broadwell microarchitecture (hyperthreaded)	i386/broadwell-ht	4
Intel Silvermont microarchitecture	i386/silvermont	2
TIMER_INT	timer	1

使用表格 5-12 OProfile 处理器和计数器来确定您的 CPU 类型能够被同时监视的事件数量。如果处理器没有支持的性能监视硬件，计时器（timer）就会被用作处理器类型。

如果使用了计时器，就不能为任何处理器设置事件，因为硬件不支持硬件性能计数器。相反，计时器中断会被用来进行评测。

如果计时器没有被用作处理器类型，监视的事件就可以被改变，处理器的计数器 0 就会被默认设置为基于时间的事件。如果处理器上有多个计数器，0 以外的计数器默认不会被设置任何事件。被监视的默认事件显示在表格 5-13 默认事件中。

表格 5-13 默认事件

处理器	计数器的默认事件	描述
AMD Athlon and AMD 64	CPU_CLK_UNHALTED	处理器的时钟没有停止
AMD Family 10h,	CPU_CLK_UNHALTED	处理器的时钟没有停

处理器	计数器的默认事件	描述
AMD Family 11h, AMD Family 12h		止
AMD Family 14h, AMD Family 15h	CPU_CLK_UNHALTED	处理器的时钟没有停止
Applied Micro X-Gene	CPU_CYCLES	处理器周期
ARM Cortex A53	CPU_CYCLES	处理器周期
ARM Cortex A57	CPU_CYCLES	处理器周期
IBM POWER4	CYCLES	处理器周期
IBM POWER5	CYCLES	处理器周期
IBM POWER8	CYCLES	处理器周期
IBM PowerPC 970	CYCLES	处理器周期
Intel Core i7	CPU_CLK_UNHALTED	处理器的时钟没有停止
Intel Nehalem microarchitecture	CPU_CLK_UNHALTED	处理器的时钟没有停止
Intel Pentium 4 (hyperthreaded and nonhyper-threaded)	GLOBAL_POWER_EVENTS	处理器没有停止的时间
Intel Westmere microarchitecture	CPU_CLK_UNHALTED	处理器的时钟没有停止
Intel Broadwell microarchitecture	CPU_CLK_UNHALTED	处理器的时钟没有停止
Intel Silvermont microarchitecture	CPU_CLK_UNHALTED	处理器的时钟没有停止
TIMER_INT	(none)	每个计时器中断抽样


可以被同时监视的事件数量是由处理器的计数器数量决定的。不过，这不是

一对一的关系；在一些处理器上，某些事件必须被映射到指定的计数器上。要确定可用的计数器数量，请执行以下命令：

```
ls -ld /dev/oprofile/[0-9]*
```

可用的事件根据处理器类型的不同而不同。要确定可用于评测的事件，请使用 `ophelp` 命令。该命令返回的列表是针对系统处理器类型所特有的。

```
ophelp
```

 如果 OProfile 没有被正确地配置，`ophelp` 命令将会失败，并返回以下错误信息：

```
Unable to open cpu_type file for reading
Make sure you have done opcontrol --init
cpu_type 'unset' is not valid
you should upgrade oprofile or force the use of timer mode
```

要配置 OProfile，请参见 5.10.3 使用遗留模式配置 OProfile。

每一个计数器的事件可以通过命令行或者图形界面来配置。要了解关于图形界面的更多信息，请参见 5.10.9 图形界面。如果计数器不能设定一个特定的事件，将会显示一个错误信息。

要通过命令行为每个可配置的计数器设置事件，请使用 `opcontrol` 命令：

```
opcontrol --event=event-name:sample-rate
```

把 `event-name` 替换成 `ophelp` 中显示的确切事件名称，把 `sample-rate` 替换为取样之间的事件数。


取样率

默认设置会选择基于时间的事件设置。它会每个处理器每 100000 个时钟周期创建一个样本。如果使用了计时器中断，计时器就被设置成两幅画面的最小时间间隔率，而且还不能被用户设置。如果 `cpu_type` 不是 `timer`，就可以为每个事件设置了一个取样率。取样率是每次取样之间发生的事件数量。

在为计数器设置事件时，还可以指定一个取样率：

```
opcontrol --event=event-name:sample-rate
```

将 `sample-rate` 替换为再次取样前需要等待的事件数。次数越小，取样就越频繁。对于不会频繁发生的事件，可能需要设置一个较小的值，以便能够捕获足够多的事件实例。

 在设置取样率时请格外小心。取样太频繁会使系统超载，导致系统假死。

单元掩码

一些用户性能监视事件可能需要单元掩码来进一步定义事件。

每个事件的单元掩码能够通过 `ophelp` 命令列出。每个单元掩码的值以十六进制格式列出。要指定一个以上的单元掩码，十六进制值必须用按位或操作来组合。

```
opcontrol --event=event-name:sample-rate:unit-mask
```

注意，在某些架构中，可能会有多个单元掩码具有相同的十六进制值。在那种情况下，它们只能通过名称来指定。

分离内核和用户空间评测

默认情况下，会为每个事件收集内核模式和用户模式的信息。要配置 OProfile 在某个指定的计数器中忽略内核模式的事件，请执行以下命令：

```
opcontrol --event=event-name:sample-rate:unit-mask:0
```

执行以下命令让计数器再次开始评测内核模式：

```
opcontrol --event=event-name:sample-rate:unit-mask:1
```

要配置 OProfile 在某个指定的计数器中忽略用户模式的事件，请执行以下命令：

```
opcontrol --event=event-name:sample-rate:unit-mask:1:0
```

执行以下命令让计数器再次开始评测用户模式：

```
opcontrol --event=event-name:sample-rate:unit-mask:1:1
```

当 OProfile 守护进程将评测数据写入样本文件时，它可以把内核和库评测数据分离到单独的样本文件中。要配置守护进程写入样本文件的方式，请以 root 用户执行以下命令：

```
opcontrol --separate=choice
```

choice 参数可以是以下之一：

- none——不要分离评测数据（默认值）。
- library——为库生成每个应用程序的评测数据。
- kernel——为内核和内核模块生成每个应用程序的评测数据。
- all——为库生成每个应用程序的评测数据，为内核和内核模块生成每个应用程序的评测数据。

如果使用了 `--separate=library`，样本文件名在包括可执行文件名称的同时还包括库的名称。

 配置信息的修改在 OProfile 评测器重新启动时生效。

5.10.4 启动和停止 OProfile 遗留模式

要使用 OProfile 来开始监视系统，请以 root 用户执行以下命令：


```
opcontrol --start
```

所显示的输出和下面类似：

```
Using log file /var/lib/oprofile/oprofiled.log Daemon started. Profiler
running.
```

`/root/.oprofile/daemonrc` 中的设置将被使用。

OProfile 守护进程 `oprofiled` 被启动；它会周期性地把取样数据写到 `/var/lib/oprofile/samples/` 目录下。守护进程的日志文件位于 `/var/lib/oprofile/oprofiled.log`。

 在中标麒麟高级服务器操作系统 V7 中，`nmi_watchdog` 向 `perf` 子系统注册了。基于此，`perf` 子系统在启动时掌握了对性能计数器寄存器的控制权，使得 OProfile 不能正常工作。

要解决这个问题，要么以设置 `nmi_watchdog=0` 内核参数的方式来启

动系统，要么在运行时以 root 用户来执行以下命令禁用 nmi_watchdog:

```
echo 0 > /proc/sys/kernel/nmi_watchdog
```

要重新启用 nmi_watchdog，请以 root 用户执行以下命令:

```
echo 1 > /proc/sys/kernel/nmi_watchdog
```

要停止评测器，请以 root 用户执行以下命令:

```
opcontrol --shutdown
```

5.10.5 在遗留模式下保存数据

有时在一个指定的时间保存样本是很有用的。例如，在评测一个可执行文件时，根据不同的输入数据来收集不同的样本可能会很有用。如果要监视的事件数量超过了处理器可用的计数器数量，您可以运行多次 OProfile 来收集数据，每次都把样本数据保存到不同的文件中。

要保存当前的样本文件集合，请执行以下命令，把 *name* 替换成对当前会话的唯一描述性名称:

```
opcontrol --save=name
```

该命令会创建/var/lib/oprofile/samples/*name*/目录，并把当前的样本文件复制到该目录下。

要指定会话目录来保存样本数据，请使用--session-dir 选项。如果没有指定，则数据被保存到当前路径的 oprofile_data/目录下。

5.10.6 分析数据

无论您使用 *perf* 还是遗留模式下的 *opcontrol* 来收集您的评测，它们都使用相同的 OProfile 后处理工具。

默认情况下，*perf* 将评测数据保存在当前目录的 oprofile_data/目录下。您可以使用--session-dir 选项来指定一个不同的位置。通常的评测后分析工具，例如 *opreport* 和 *opannotate*，可以用来生成评测报告。这些工具首先在当前目录的 oprofile_data/目录下搜索样本。如果不存在这个目录，分析工具将使用标准会话目录/var/lib/oprofile/。统计数据，例如接收到的总样本以及丢失样本，将被写入

`session_dir/samples/operf.log` 文件。

当使用遗留模式时，OProfile 守护进程 `oprofiled`，将会周期性地收集样本，并把它写入到 `/var/lib/oprofile/samples/` 目录下。在读取数据之前，请以 `root` 用户执行以下命令，以确保所有数据都被写入了该目录：

```
opcontrol --dump
```

每个样本文件都是基于可执行文件的名称来命名的。例如，在 Pentium III 处理器上对 `/bin/bash` 进行默认事件评测的样本名称为：

```
\{root\}/bin/bash/\{dep\}/\{root\}/bin/bash/CPU_CLK_UNHALTED.100  
000
```

以下工具可以用来在收集好样本数据之后，对样本数据进行评测：

- `opreport`
- `opannotate`

用这些工具，以及被评测的二进制文件，可以生成能够被进一步分析报告。

⚠ 被评测的可执行文件必须和这些工具一起使用，来分析数据。如果在收集数据之后必须修改可执行文件，请备份用来创建样本的可执行文件以及样本文件。注意，样本文件和二进制文件的名称必须一致。如果这些名称不匹配，您将无法做备份。作为替代方案，`oparchive` 可以用来处理这个问题。

每个可执行文件的样本被写入一个单独的样本文件中。从每个动态链接库中取得的样本也被写入一个单独的样本文件中。当 OProfile 运行时，如果被监视的可执行文件改变了，并且存在一个该可执行文件的样本文件，则已存在的样本文件将被自动删除。因此，如果已存在的样本文件是需要的，则必须对其进行备份，并且用来创建它的可执行文件，在被新版本的可执行文件替换之前也必须进行备份。OProfile 分析工具在分析时会使用生成样本的可执行文件。如果可执行文件改变了，分析工具将无法分析和其相关的样本。请参见 5.10.5 在遗留模式下保存数据了解如何备份样本文件。

使用 opreport

opreport 工具可以提供被评测的所有可执行文件的一个概览。以下是 opreport 命令的部分样本输出：

```
~]$ opreport
Profiling through timer interrupt
TIMER:0|
samples|      %|
-----
25926  97.5212  no-vmlinux
359    1.3504   pi
65     0.2445   Xorg
62     0.2332   libvte.so.4.4.0
56     0.2106   libc-2.3.4.so
34     0.1279   libglib-2.0.so.0.400.7
19     0.0715   libXft.so.2.1.2
17     0.0639   bash
8      0.0301   ld-2.3.4.so
8      0.0301   libgdk-x11-2.0.so.0.400.13
6      0.0226   libgobject-2.0.so.0.400.7
5      0.0188   oprofiled
4      0.0150   libpthread-2.3.4.so
4      0.0150   libgtk-x11-2.0.so.0.400.13
3      0.0113   libXrender.so.1.2.2
3      0.0113   du
1      0.0038   libcrypto.so.0.9.7a
1      0.0038   libpam.so.0.77
1      0.0038   libtermcap.so.2.0.8
```

```
1 0.0038 libX11.so.6.2
1 0.0038 libgthread-2.0.so.0.400.7
1 0.0038 libwnck-1.so.4.9.0
```

每一个可执行文件都在列表中独占一行。第一列是为可执行文件所记录的样本数。第二列是可执行文件的样本数相对于总样本数所占的百分比。第三列是可执行文件的名称。

请参见 `opreport(1)` 用户手册页面查看可用的命令行选项列表，例如 `-r` 选项用来对输出结果按照可执行文件的样本数从小到大排序。您也可以使用 `-t` 或者 `--threshold` 选项来修剪 `opcontrol` 的输出结果。

对单个可执行文件使用 `opreport`

要取回关于一个特定的可执行文件的更详细的评测信息，请使用：

```
opreport mode executable
```

请用将被分析的可执行文件的全路径来替换 *executable*。*mode* 表示以下选项中的一个：

`-l`

该选项用来按符号列出样本数据。例如，执行下面这个命令：

```
~]# opreport -l /lib/tls/libc-version.so
```

将产生以下输出：

```
samples    %    symbol name
12   21.4286  __gconv_transform_utf8_internal
5    8.9286   _int_malloc 4 7.1429 malloc
3    5.3571   __i686.get_pc_thunk.bx
3    5.3571   _dl_mcount_wrapper_check
3    5.3571   mbrtowc
3    5.3571   memcpy
2    3.5714   _int_realloc
```



```

2    3.5714    _nl_intern_locale_data
2    3.5714    free
2    3.5714    strcmp
1    1.7857    __ctype_get_mb_cur_max
1    1.7857    __unregister_atfork
1    1.7857    __write_nocancel
1    1.7857    _dl_addr
1    1.7857    _int_free
1    1.7857    _itoa_word
1    1.7857    calc_eclosure_iter
1    1.7857    fopen@ @ GLIBC_2.1
1    1.7857    getpid
1    1.7857    memmove
1    1.7857    msort_with_tmp
1    1.7857    strcpy
1    1.7857    strlen
1    1.7857    vfprintf
1    1.7857    write
    
```

第一列是符号的样本数，第二列是符号的样本数相对于可执行文件的全部样本数的百分比，第三列是符号名称。

要按样本数从小到大（反序）对输出结果排序，可以让 `-r` 选项和 `-l` 选项结合使用。

`-i symbol-name`

该选项用来列出指定的符号名称的样本数据。例如，执行下面这个命令：

```

~]# oprofile -l -i __gconv_transform_utf8_internal
/lib/tls/libc-version.so
    
```

将返回以下输出：

```
samples    %    symbol name
12    100.000    __gconv_transform_utf8_internal
```

第一列是内存符号的样本数，第二列是内存地址的样本数相对于符号的总样本数的百分比，第三列是符号名称。

-d

该选项用来按符号列出样本数据，它比-l 选项的输出更详细。例如，执行下面这个命令：

```
~]# oprofile -d -i __gconv_transform_utf8_internal
/lib/tls/libc-version.so
```

将返回以下输出：

```
vma samples % symbol name
00a98640 12 100.000 __gconv_transform_utf8_internal
00a98640 1 8.3333
00a9868c 2 16.6667
00a9869a 1 8.3333
00a986c1 1 8.3333
00a98720 1 8.3333
00a98749 1 8.3333
00a98753 1 8.3333
00a98789 1 8.3333
00a98864 1 8.3333
00a98869 1 8.3333
00a98b08 1 8.3333
```

以上数据跟-l 选项一样，只不过对每个符号，还显示了其所使用的虚拟内存地址。对于每一个虚拟内存地址，显示了它的样本数，以及其样本数相对于符号的总样本数的百分比。

-e symbol-name...

使用该选项，您可以在输出结果中排除一些符号。请用以逗号分隔的，您想排除的符号列表来替换 *symbol-name*。

`session:name`

这里，您可以指定会话的全路径，一个相对于 `/var/lib/oprofile/samples/` 的目录，或者如果您使用的是 `operf`，则是一个相对于 `./oprofile_data/samples/` 的目录。

获取更详细的模块输出

OProfile 在系统范围内为运行在机器上的内核代码和用户空间代码收集数据。然而，一旦一个模块被加载到了内核中，关于该内核模块的起源信息就丢失了。该模块可能在启动时来自于 `initrd` 文件，可能来自于包含很多内核模块的目录，也可能来自于一个本地创建的内核模块。结果，当 OProfile 为一个模块记录样本时，它只是把该模块的样本列到了根目录下的一个可执行文件上，但这不可能是该模块实际代码的位置。您需要执行一些步骤来确保分析工具获取正确的可执行文件。

要获取模块动作的更详细视图，您可以安装上内核的 `debuginfo` 包。

请使用 `uname -a` 命令找出正在运行的内核，然后获取合适的 `debuginfo` 包，并将其安装到系统中。

之后请使用以下命令清理样本数据：

```
opcontrol --reset
```

要启动监视进程，例如，在一个 Westmere 处理器的机器上启动监视进程，可以使用以下命令：

```
~]# opcontrol --setup --vmlinux=/usr/lib/debug/lib/modules/^uname  
-r`/vmlinux --event=CPU_CLK_UNHALTED:500000
```

然后可以获取模块的详细信息，例如，可以使用以下命令获取 `ext4` 模块的详细信息：

```
~]# oprofile /ext4 -l --image-path /lib/modules/^uname -r`/kernel
```

```
CPU: Intel Westmere microarchitecture, speed 2.667e+06 MHz
(estimated)

Counted CPU_CLK_UNHALTED events (Clock cycles when not halted)
with a unit mask of 0x00 (No unit mask) count 500000

warning: could not check that the binary file
/lib/modules/2.6.32-191.el6.x86_64/kernel/fs/ext4/ext4.ko has not been
modified since the profile was taken. Results may be inaccurate.

samples    %    symbol name
1622    9.8381    ext4_iget
1591    9.6500    ext4_find_entry
1231    7.4665    __ext4_get_inode_loc
783    4.7492    ext4_ext_get_blocks
752    4.5612    ext4_check_dir_entry
644    3.9061    ext4_mark_iloc_dirty
583    3.5361    ext4_get_blocks
583    3.5361    ext4_xattr_get
479    2.9053    ext4_htree_store_dirent
469    2.8447    ext4_get_group_desc
414    2.5111    ext4_dx_find_entry
```

使用 opannotate

opannotate 工具试图将特定指令的样本匹配到源代码的对应行上。最终生成的文件在左侧会显示对应行的样本。它也会在每个函数的前面添加一个注释，其中列出该函数的总样本。

要使用该工具，需要在系统中安装可执行文件的适当的 debuginfo 包。在中标麒麟高级服务器操作系统中，debuginfo 包不会跟随包含可执行文件的安装包而被自动安装。您需要单独获取并安装 debuginfo 包。

opannotate 命令的通用语法如下所示：

```
opannotate --search-dirs src-dir --source executable
```

这些命令行选项是强制的。请用一个包含源代码的目录路径替换 *src-dir*，并指定将要被分析的可执行文件。请参见 `opannotate(1)` 用户手册页面了解其它命令行选项。

5.10.7 理解/dev/oprofile/目录

在使用 OProfile 遗留模式时，`/dev/oprofile/` 目录用来为 OProfile 保存文件系统。另一方面，`perf` 不需要 `/dev/oprofile/`。请使用 `cat` 命令来查看在这个文件系统中的虚拟文件的值。例如，以下命令显示 OProfile 检测到的处理器类型：

```
cat /dev/oprofile/cpu_type
```

在 `/dev/oprofile/` 目录中，每个计数器都存在一个对应的目录。例如，如果有 2 个计数器，则存在 `/dev/oprofile/0/` 和 `/dev/oprofile/1/` 两个目录。

每个计数器所对应的目录包含以下文件：

- `count`——取样间隔。
- `enabled`——如果值为 0，则计数器是关闭的，不会为其收集样本；如果值为 1，则计数器是打开的，将会为其收集样本。
- `event`——要监视的事件。
- `extra`——在具有 Nehalem 处理器的机器上用来进一步指定要监视的事件。
- `kernel`——如果值为 0，当处理器在内核空间时，就不会为这个计数器事件而收集样本；如果值为 1，即使处理器在内核空间时，也会收集样本。
- `unit_mask`——为计数器定义启用的单元掩码。
- `user`——如果值为 0，当处理器在用户空间时，就不会为这个计数器事件而收集样本；如果值为 1，即使处理器在用户空间时，也会收集样本。

这些文件的值可以用 `cat` 命令来获取，例如：

```
cat /dev/oprofile/0/count
```

示例用法

OProfile 可以被开发者用来分析应用程序性能，也可以被系统管理员用来执行系统分析。例如：

- 确定哪些应用程序和服务在系统上使用得最多——`opreport` 能够用来确定一个应用程序或服务使用了多少处理器时间。如果系统运行了多个服务，但是运行状况不佳，消耗处理器时间最多的服务可以移动到专用系统上。
- 确定处理器使用情况——可以通过监视 `CPU_CLK_UNHALTED` 事件来确定在一个给定的时间周期内处理器的负载。该数据之后可以用来确定是否额外的处理器或者更快的处理器能够提高系统性能。

5.10.8 OProfile 对 Java 的支持

OProfile 允许您评测 Java 虚拟机 (JVM) 的动态编译代码（也被称作即时编译代码）。中标麒麟高级服务器操作系统 V7 中的 OProfile 包括了对 Java 虚拟机工具接口 (JVM Tools Interface, JVMTI) 代理库的内建支持，可以支持 Java1.5 或更高版本。

评测 Java 代码


要使用 JVMTI 代理来评测 Java 虚拟机的即时编译代码，请添加以下 JVM 启动参数：

```
-agentlib:jvmti_oprofile
```

这里的 `jvmti_oprofile` 是 OProfile 代理的路径。对于 64 位的 JVM，路径如下所示：

```
-agentlib:/usr/lib64/oprofile/libjvmti_oprofile.so
```

当前，您可以添加 `--debug` 命令行选项来启用调试模式。

-  要使用 OProfile 来评测即时编译代码，必须在系统中安装 `oprofile-jit` 软件包。有了这个包，您就能够显示方法级别的信息了。

取决于您所使用的 JVM，您可能需要为 JVM 安装 `debuginfo` 包。对于


OpenJDK, debuginfo 包是需要安装的, 而 Oracle JDK 则没有 debuginfo 包。为了让调试信息安装包和它们各自的非调试安装包保持同步, 您还需要安装 yum-plugin-auto-update-debug-info 插件。这个插件会搜索调试信息仓库来寻找对应的更新。

成功配置之后, 您就可以按照前面的各小节中描述的那样, 使用标准的评测和分析工具了。

5.10.9 图形界面

一些 OProfile 首选项可以通过图形界面来设置。请确保在您的系统上已经安装了提供 OProfile 图形用户界面的 oprofile-gui 软件包。要启动该界面, 请在命令行提示符下以 root 用户执行 oprof_start 命令。

在对任何选项进行修改之后, 请点击 **【Save and quit】** 按钮保存修改。首选项将被写入 /root/.oprofile/daemonrc 文件中, 然后应用程序退出。

 退出应用程序不会让 OProfile 停止取样。

在 **【Setup】** 标签页, 要像 5.10.3.2 设置要监视的事件中讨论的那样为处理器计数器设置事件, 从下拉菜单中选择计数器, 然后从列表中选择事件。一个简要的事件描述将会出现在列表下方的文本框中。列表中只会显示指定的计数器和指定的架构可用的事件。界面也会显示评测器是否正在运行, 以及评测器的一些简要统计信息。

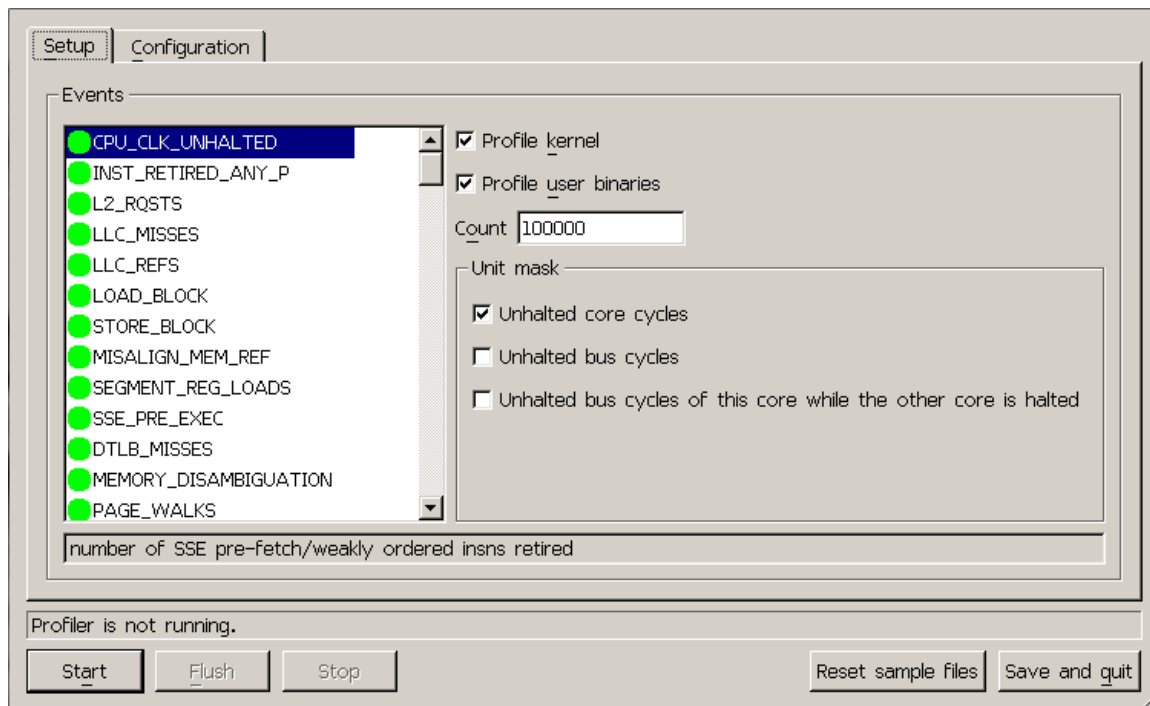


图 5-14 OProfile 设置

在标签页的右侧，选择【Profile kernel】选项来为当前选择的事件在内核模式下对事件进行计数，正如 5.10.3.3 分离内核和用户空间评测所讨论的那样。如果没有选择该选项，则不会为内核收集样本。

选择【Profile user binaries】选项来为当前选择的事件在用户模式下对事件进行计数，正如 5.10.3.3 分离内核和用户空间评测所讨论的那样。如果没有选择该选项，则不会为用户应用程序收集样本。

使用【Count】文本框来为当前选择的事件设置取样率，正如 5.10.3.2.1 取样率所讨论的那样。

如果当前选择的事件有任何可用的单元掩码，它们将被显示在【Setup】标签页的右侧【Unit mask】区域中。选择单元掩码旁边的复选框来为事件启用单元掩码。

在【Configuration】标签页，要评测内核，请在【Kernel image file】文本框中为要监视的内核输入 vmlinux 文件的名称和位置。要配置 OProfile 不监视内核，请选择【No kernel image】。

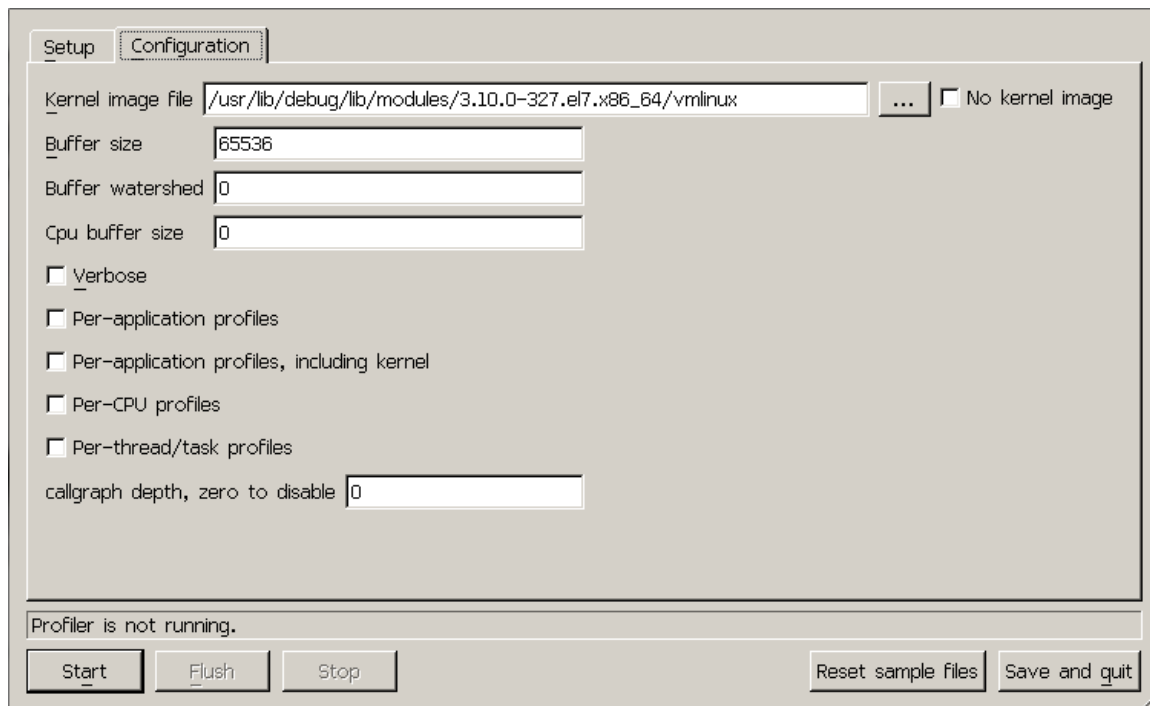


图 5-15 OProfile 配置

如果选择了【Verbose】选项，oprofiled 进程日志将包含更详细的信息。

如果选择了【Per-application profiles】，OProfile 将为库生成每个应用程序的评测。这相当于 `opcontrol --separate=library` 命令。如果选择了【Per-application profiles, including kernel】，OProfile 将为内核和内核模块生成每个应用程序的评测，正如 5.10.3.3 分离内核和用户空间评测所讨论的那样。这相当于 `opcontrol --separate=kernel` 命令。

要像 5.10.6 分析数据中讨论的那样，强制把数据写到样本文件中，请点击【Flush】按钮。这相当于 `opcontrol --dump` 命令。

要从图形界面启动 OProfile，请点击【Start】按钮。要停止评测器，请点击【Stop】按钮。退出应用程序不会让 OProfile 停止取样。

5.10.10 Oprofile 和 SystemTap

SystemTap 是一个追踪和探测工具，可以让用户比较详细地学习和监视操作系统的活动。它能够提供和 `netstat`、`ps`、`top`、`iostat` 等工具的输出类似的信息，不过，设计 SystemTap 是希望能够为收集到的信息提供更多的过滤和分析选项。

要了解为什么处理器在一个特定的代码区域花费时间以及具体在什么位置

花费时间，而需要收集数据时，建议使用 OProfile。而要找出为什么处理器总是处于空闲状态，就不太用得上 OProfile 了。

当您想在代码中的指定位置进行植入时，您可以使用 SystemTap。因为 SystemTap 可以让您进行代码植入，而不用停止或重启被植入的代码。它在向内核和守护进程中进行植入时特别有用。

6 kernel、module 和驱动配置

6.1 使用 GRUB2 启动加载工具

NeoKylin Linux Advanced Server V7 使用 GRUB2 (GRand 统一引导加载程序版本 2) 作为引导加载工具。在系统启动时, 用户可以使用 GRUB2 来选择启动的操作系统或内核, 同时, 可以使用 GRUB2 向内核传递参数。

6.1.1 GRUB2 简介

在传统基于 BIOS 引导的机器上, GRUB2 将 `/boot/grub2/grub.cfg` 作为它的配置文件; 在基于 UEFI 引导的机器上, GRUB2 将 `/boot/efi/EFI/neokylin /grub.cfg` 作为它的配置文件。这个配置文件包含了所需的选项信息。

GRUB2 的配置文件 `grub.cfg` 是在系统安装时生成的, 也可以用 `/usr/sbin/grub2-mkconfig` 来生成。每当有新内核安装后, 会有 `grubby` 自动更新此配置文件。当需要用 `grub2-mkconfig` 手动生成配置文件时, `grub2-mkconfig` 会利用 `/etc/grub.d/` 目录下提供的模板生成 `grub.cfg`, 用户的自定义设置将保存于 `/etc/default/grub` 文件中。任何一次使用 `grub2-mkconfig` 工具都将会刷新 `grub.cfg`, 因此, 要注意每次手动修改对 `/etc/default/grub` 产生的影响。

新内核的删除和增加都应该使用工具 `grubby` 来完成。针对脚本, 应该使用 `new-kernel-pkg` 工具。使用 `grubby` 工具对默认内核做的改变, 会在新内核安装时被继承。有关 `grubby` 更多的信息, 请参看 24.4 用 `grubby` 工具完成对 GRUB2 选项的永久修改。

在系统安装过程中, `anaconda` 常常会用到 `grub2-mkconfig` 所用的 `/etc/default/grub` 文件来创建 `grub.cfg`。在系统遭遇失败时, 也会常常用到它, 例如: 配置引导加载工具, 需要重建, 就要用到它。除了作为最后的办法, 通常, 不推荐手动运行 `grub2-mkconfig` 生成 `grub.cfg`。注意, 任何手动对 `/etc/default/grub` 的修改都会重建 `grub.cfg` 文件。

`grub.cfg` 中的选项

在众多的代码段和指示标志中, `grub.cfg` 文件包含了一个或多个 `menuentry` 块, 它们每个都代表 GRUB2 的一个引导项。这些块都以 `menuentry` 关键字开头,

接着就是标题栏和选项列表，选项是以花括号开始和结束的，并且要缩进排列。

例如：以下是有关 NeoKylin Linux Advanced Server V7（kernel 3.10.0-327.el7.x86_64）的 menuentry 块实例：

```
menuentry 'NeoKylin Linux Advanced Server (3.10.0-327.el7.x86_64) V7Update2 (Potassium)'
--class neoklyn --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option
'gnulinux-3.10.0-327.el7.x86_64-advanced-c883439f-45fb-42f2-8167-06036ff2eb0a' {
    load_video
    set gfxpayload=keep
    insmod gzio
    insmod part_msdos
    insmod xfs
    set root='hd0,msdos1'
    if [ x${feature_platform_search_hint} = xy ]; then
        search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-efi=hd0,msdos1
--hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1' 8c2d6f7d-4435-493e-8f17-884a0eaa0821
    else
        search --no-floppy --fs-uuid --set=root 8c2d6f7d-4435-493e-8f17-884a0eaa0821
    fi
    linux16 /vmlinuz-3.10.0-327.el7.x86_64
    root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a ro crashkernel=auto rhgb quiet
    LANG=zh_CN.UTF-8
    initrd16 /initramfs-3.10.0-327.el7.x86_64.img
}
```

在 64-bit IBM POWER Series 系统上，安装的 Linux Kernel 标识为：linux；在基于 BIOS x86_64 的系统上，安装的 Linux Kernel 标识为：linux16；在基于 UEFI 的系统上，安装的 Linux Kernel 标识为：linuxefi。然后，由 initrd 指出内核路径所在；initramfs 镜像关联到/boot。如上例：当根文件系统加载时，initrd16 /initramfs-3.10.0-327.el7.x86_64.img 行表示 initramfs 镜像的实际路径为：/boot/initramfs-3.10.0-327.el7.x86_64.img，同样，也是内核的路径。

在 linux16 /vmlinuz-kernel_version 中，给出的内核版本号必须和在 menuentry 块中 initrd /initramfs-kernel_version.img 给出的版本号一致。有关更多如何验证初始 RAM 盘镜像的信息，请参看 25.5 验证初始 RAM 磁盘镜像。

注意：

在 menuentry 块中，initrd 指示项必须指向和 initramfs 文件所对应的内核版本一

致的路径(如果在单独分区上，要关联到/boot/目录上)上。因为早期版本是使用 mkinitrd 来创建 RAM 磁盘镜像文件的，并命名为 initrd 文件，因此，这个指示项也就被称为了 initrd。grub.cfg 保留 initrd 是为了和其他工具兼容。遵循系统文件命名规则，使用 dracut 工具创建初始 RAM 磁盘镜像为 initramfs-kernel_version.img。

有关如何使用 dracut 的更多信息，请参看 25.5 验证初始 RAM 磁盘镜像。

6.1.2 配置 GRUB2 引导加载工具

在引导系统时，可以临时修改 GRUB2 选项；在系统运行中，可以永久修改单系统的 GRUB2 选项，或作为新的 GRUB2 配置文件的一部分。

- 要临时修改 GRUB2 选项，请参看 24.3 临时修改 GRUB2 选项
- 要永久修改 GRUB2 选项，请参看 24.4 用 grubby 工具永久修改 GRUB2 选项
- 要自定义 GRUB2 配置文件，请参看 24.5 自定义 GRUB2 配置文件

6.1.3 临时修改 GRUB2 选项

过程：临时修改内核选项

为了在单核系统启动过程中，只修改内核参数，需完成如下操作：

1. 启动系统，到达 GRUB2 引导界面，在其上移动光标到待编辑项，按下【e】e 键，方可编辑。
2. 移动键盘上的向下导航键到达内核命令行。内核命令行的起始标识因系统而异，在 64-bit IBM POWER Series 上，内核命令行起始标识为：linux；在基于 BIOS 的 x86_64 系统上，内核命令行起始标识为：linux16；在 UEFI 系统上，内核命令行起始标识为：linuxefi。
3. 移动光标到行尾

按下【Ctrl+a】和【Ctrl+e】可以分别跳到行首和行尾。在有些系统上，可以用【Home】键和【End】n 键来完成同样的操作。

4. 根据需要编辑内核参数。想让系统运行在应急方式下时，可以在 linux16 行尾加上 emergency 参数。

linux16 /vmlinuz-3.10.0-327.el7.x86_64

```
root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a ro crashkernel=auto rhgb  
quiet LANG=zh_CN.UTF-8 emergency
```

为了看到系统启动的详尽信息，可以去掉参数 **rhgb** 和 **quiet**。

6.1.4 用 grubby 工具永久修改 GRUB2 选项

使用 **grubby** 工具可以从 **grub.cfg** 文件中读取信息，并能使对它的修改永久生效。它能使针对 GRUB 参数选项所做的修改，在系统启动时，传到内核，并能在默认内核中生效。

在 NeoKylin Linux Advanced Server V7 中，如果执行 **grubby** 时，没有指定 GRUB 配置文件，那么，默认会使用 **/etc/grub2.cfg** 文件，其实该文件是到 **grub.cfg** 文件的一个连接，**grub.cfg** 文件所在目录依赖于系统的体系结构。如果没有找到 **grub.cfg** 文件，则会在默认的体系结构系统中搜索。

列出默认内核

可以使用以下命令，查看默认内核文件名：

```
~]# grubby --default-kernel  
  
/boot/vmlinuz-3.10.0-327.el7.x86_64
```

可以使用以下命令，查看默认内核索引号：

```
~]# grubby --default-index  
  
0
```

修改默认引导项

可以使用以下命令，指定某个内核作为默认引导内核：

```
~]# grubby --set-default /boot/vmlinuz-3.10.0-327.el7.x86_64
```

浏览内核 GRUB 选项

可以用以下命令列出所有内核选项：

```
~]# grubby --info=ALL
```

可以用以下命令，浏览指定内核的 GRUB 选项：

```
~]# grubby --info /boot/vmlinuz-3.10.0-327.el7.x86_64
index=0
kernel=/boot/vmlinuz-3.10.0-327.el7.x86_64
args="ro crashkernel=auto rhgb quiet LANG=zh_CN.UTF-8"
root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a
initrd=/boot/initramfs-3.10.0-327.el7.x86_64.img
title=NeoKylin Linux Advanced Server (3.10.0-327.el7.x86_64) V7Update2
(Potassium)
```

试着用标签可参看到/boot/目录下的所有有效内核。

在 GRUB 选项中添加/删除参数

在 grubby 命令中，联合使用选项--update-kernel 和--args，就可以为 GRUB 选项添加参数；若使用选项--remove-arguments，则可删除已设置参数。这些选项都可以用空格加以分开，外加双引号加以分组。下面格式的命令可同时实现从 grubby 命令中添加/删除参数。

```
grubby --remove-args="argX argY" --args="argA argB" --update-kernel /boot/kernel
```

为了给某个指定内核的 GRUB 选项添加/删除参数，可以使用以下命令：

```
~]# grubby --remove-args="rhgb quiet" --args=console=ttyS0,115200 --update-kernel
/boot/vmlinuz-3.10.0-327.el7.x86_64
```

这个命令删除了图形引导参数，启用显示引导信息，还添加了串口控制台。作为控制台参数，添加在了行尾，新的控制台配置将优先于任何其他控制台配置。

可以用如下命令带上参数--info 查看所做的修改：

```
~]# grubby --info /boot/vmlinuz-3.10.0-327.el7.x86_64
index=0
kernel=/boot/vmlinuz-3.10.0-327.el7.x86_64
args="ro crashkernel=auto LANG=zh_CN.UTF-8 console=ttyS0,115200"
```

```
root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a  
initrd=/boot/initramfs-3.10.0-327.el7.x86_64.img  
title=NeoKylin Linux Advanced Server (3.10.0-327.el7.x86_64) V7Update2  
(Potassium)
```

用相同参数修改所有内核选项

可以用以下命令为所有内核选项添加相同的内核引导参数：

```
~]# grubby --update-kernel=ALL --args=console=ttyS0,115200
```

--update-kernel 参数可以选择 DEFAULT，或者是用逗号分开的内核索引号。

修改内核参数

指定已设置内核参数值为某个需要值，即可实现对它的修改。例如：可以使用以下命令，修改虚拟控制台显示字符的大小：

```
~]# grubby --info /boot/vmlinuz-3.10.0-327.el7.x86_64  
index=0  
kernel=/boot/vmlinuz-3.10.0-327.el7.x86_64  
args="ro crashkernel=auto LANG=zh_CN.UTF-8 console=ttyS0,115200  
vconsole.font=latarcyrheb-sun32"  
root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a  
initrd=/boot/initramfs-3.10.0-327.el7.x86_64.img  
title=NeoKylin Linux Advanced Server (3.10.0-327.el7.x86_64) V7Update2  
(Potassium)
```

请参看 grubby (8) man 手册，可以了解该命令的更多参数选项。

6.1.5 自定义 GRUB2 配置文件

GRUB2 脚本会搜索用户的计算机，基于搜索结果来创建引导选项。一旦修改或添加了新内核，系统会自动修改引导选项来反映最新的系统启动变化。

可是，用户要创建一个含有指定选项或指定顺序选项的菜单项，GRUB2 会在屏幕上给用户一些实际可控的自定义基本选项。

GRUB2 用一组位于/etc/grub.d 目录下的脚本来创建引导选项。

00_header: 加载来自/etc/default/grub 文件的设置

01_users: 只有在 kickstart 文件中指定了引导加载密码后, 才能创建此文件。

10_linux: 指向默认引导分区的内核所在目录。

30_os_prober: 和创建在其它分区上的操作系统相关的项

40_custom: 用于创建附加选项的模板

/etc/grub.d 目录下的脚本是按字母顺序被读取的, 因此, 通过重命名这些脚本, 可以改变指定选项的引导顺序。

重点:

当/etc/default/grub 文件中的 GRUB_TIMEOUT 设置为 0 时, 在系统启动时 GRUB2 将不显示内核引导表。在启动时, 为了显示这个表, 当 BIOS 信息显示后, 要按下任一字母数字键并保持住, 就会看到带有 GRUB 菜单的 GRUB2 界面了。

修改默认引导项

默认情况下, /etc/default/grub 文件中的 GRUB_DEFAULT 被设置成了 saved。根据/boot/grub2/grubenv 目录下, GRUB2 环境文件中 saved_entry 的设置, GRUB2 选择引导内核。可以用 grub2-set-default 选择另一个 GRUB 记录作为默认的, 它会将这一选择更新到 GRUB2 环境文件中。

默认情况下, saved_entry 被设置成了最新安装的内核, 内核包类型为 kernel。它会在/etc/sysconfig/kernel 文件的 UPDATEDEFAULT 和 DEFAULTKERNEL 项中来指定。root 用户可以浏览/etc/sysconfig/kernel 文件, 如下:

```
~]# cat /etc/sysconfig/kernel
# UPDATEDEFAULT specifies if new-kernel-pkg should make
# new kernels the default
UPDATEDEFAULT=yes

# DEFAULTKERNEL specifies the default kernel package type
DEFAULTKERNEL=kernel
```

DEFAULTKERNEL 指出了将被用作默认内核引导包的类型。当将 DEFAULTKERNEL 设置成了 kernel 后，正在安装的 kernel-debug 包就不会再被设置成默认引导内核了。

也可以将 saved_entry 选项设成数字，用于表示操作系统加载时的默认顺序。为了指定某个操作系统为默认引导系统，只需将它所对应的数字作为 grub2-set-default 命令的参数，就可完成设置。例如：

```
~]# grub2-set-default 2
```

注意：在配置文件列表项中，以数字 0 作为起始序列号。因此，上述例子中，将引导的是第三项对应的系统，这个值将会被下次安装的内核名所取代。

在/etc/default/grub 文件中，当用某菜单项名作为 GRUB_DEFAULT 选项的值时，就可以强制一个系统启动时，总是能采用这个菜单项作为特殊菜单项。由 root 用户执行以下命令，就可以获得有效的菜单选项：

```
~]# awk -F' '$1=="menuentry " '{print $2}' /etc/grub2.cfg
```

/etc/grub2.cfg 是指向 grub.cfg 的一个连接文件，它所在目录依赖于系统体系结构。为了更加稳妥起见，在本章节中的其他例子里我们都不使用符号连接。当要写一个文件时，最好使用它的绝对路径，尤其是在修改一个系统配置文件时。

修改/etc/default/grub 后，会如下例重建 grub.cfg 文件：

➤ 基于 BIOS 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

➤ 基于 UEFI 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/efi/EFI/neokylin/grub.cfg
```

编辑菜单项

只要编辑/etc/default/grub 文件中的 GRUB_CMDLINE_LINUX 选项的值，就可以生成一个带有不同参数的新的 GRUB2 文件。注意：可以为 GRUB_CMDLINE_LINUX 指定多重参数，它类似于在 GRUB2 引导菜单中添加

参数。例如：

```
GRUB_CMDLINE_LINUX="console=tty0 console=ttyS0,9600n8"
```

这里，将 `console= tty0` 作为第一个虚拟端；将 `console=ttyS0` 作为串口终端来使用。

修改 `/etc/default/grub`，会如下例重建 `grub.cfg` 文件：

- 基于 BIOS 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- 基于 UEFI 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/efi/EFI/neokylin/grub.cfg
```

添加新项

当执行 `grub2-mkconfig` 命令时，GRUB2 会搜索 Linux 内核，会基于 `/etc/grub.d/` 目录下的脚本文件去搜索其它操作系统。`/etc/grub.d/10_linux` 脚本用于搜索安装于同一分区上的 Linux 内核，`/etc/grub.d/30_os-prober` 脚本用于搜索其它操作系统。当更新内核时，菜单项也会被自动添加到引导菜单中。

`/etc/grub.d/40_custom` 文件是自定义项参考模板，如下所示：

```
#!/bin/sh
exec tail -n +3 $0

# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.
```

该文件可以被编辑和拷贝。注意：必须包含的最小有效项如下：

```
menuentry "<Title>" {
<Data>
}
```

创建自定义菜单项

通过创建自定义菜单，可以避免自动修改菜单项。

重点:

在创建自定义菜单项前, 做好对/etc/grub.d 目录下内容的备份, 防止万一需要恢复到修改前的配置。

注意:

修改/etc/default/grub 文件不会对创建自定义菜单有任何影响。

1. 基于 BIOS 的系统, 追加 /boot/grub2/grub.cfg 文件的内容到 /etc/grub.d/40_custom 文件的最后; 基于 UEFI 的系统, 追加 /boot/efi/EFI/neokylin/grub.cfg 文件的内容到/etc/grub.d/40_custom 文件的最后。
40_custom 脚本的执行部分必须被保存下来。
2. 添加到/etc/grub.d/40_custom 文件的内容, 只有 menuentry 块是需要创建自定义项的。/boot/grub2/grub.cfg 文件和/boot/efi/EFI/neokylin/grub.cfg 文件可能包含了函数描述, 其它的上述内容以及下述的 menuentry 块。如果在前面步骤中拷贝了不必要的行, 则应删除掉。

这是一个自定义 40_custom 脚本的例子:

```
#!/bin/sh
exec tail -n +3 $0

# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.

menuentry 'First custom entry' --class neokylin --class gnu-linux --class gnu --class os
--unrestricted                                $menuentry_id_option
'gnulinux-3.10.0-327.el7.x86_64-advanced-c883439f-45fb-42f2-8167-06036ff2eb0a'
{
    load_video
    set gfxpayload=keep
    insmod gzio
    insmod part_msdos
```

```

insmod xfs

set root='hd0,msdos1'

if [ x$feature_platform_search_hint = xy ]; then
    search    --no-floppy    --fs-uuid    --set=root    --hint-bios=hd0,msdos1
--hint-efi=hd0,msdos1    --hint-baremetal=ahci0,msdos1    --hint='hd0,msdos1'
8c2d6f7d-4435-493e-8f17-884a0eaa0821
else
    search    --no-floppy    --fs-uuid    --set=root
8c2d6f7d-4435-493e-8f17-884a0eaa0821
fi

linux16    /vmlinuz-3.10.0-327.el7.x86_64
root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a    ro    crashkernel=auto
LANG=zh_CN.UTF-8 console=ttyS0,115200 vconsole.font=latacyrheb-sun32

initrd16 /initramfs-3.10.0-327.el7.x86_64.img
}

menuentry 'Second custom entry' --class neokylin --class gnu-linux --class gnu --class
os    --unrestricted    $menuentry_id_option
'gnulinux-0-rescue-ea311367055a4bcfb323ece036b6bf37-advanced-c883439f-45fb-4
2f2-8167-06036ff2eb0a' {
    load_video
    insmod gzio
    insmod part_msdos
    insmod xfs
    set root='hd0,msdos1'

    if [ x$feature_platform_search_hint = xy ]; then
        search    --no-floppy    --fs-uuid    --set=root    --hint-bios=hd0,msdos1
--hint-efi=hd0,msdos1    --hint-baremetal=ahci0,msdos1    --hint='hd0,msdos1'
8c2d6f7d-4435-493e-8f17-884a0eaa0821
    
```

```

else
    search          --no-floppy          --fs-uuid          --set=root
8c2d6f7d-4435-493e-8f17-884a0eaa0821
fi
linux16            /vmlinuz-0-rescue-ea311367055a4bcfb323ece036b6bf37
root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a ro crashkernel=auto rhgb quiet
initrd16 /initramfs-0-rescue-ea311367055a4bcfb323ece036b6bf37.img
}

```

3. 删除/etc/grub.d 目录下全部文件，尤其是下列这些文件：

- 00_header
- 40_custom
- 01_users（若存在）
- README

或者，通过对/etc/grub.d 目录下的文件执行 `chmod a-x <file_name>` 命令，可以达到既保留这些文件，又不让它们可执行。

4. 按照需要编辑，添加或删除 40_custom 文件中的菜单项

5. 根据不同系统体系结构，执行以下 `grub2-mkconfig -o` 命令，重建 grub.cfg 文件。

- 基于 BIOS 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- 基于 UEFI 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/efi/EFI/neokylin/grub.cfg
```

6.1.6 GRUB2 密码保护

在 GRUB 2 模板文件中，GRUB 2 支持纯文本密码和加密密码。为了启用密码功能，指定需要超级用户保护的选项，同时，还要指定其他用户也能访问它们。

为了引导时能用密码保护多个选项，需要加一个或多个用户可以到 6.1.6.1 指定

菜单项设置用户和密码保护 章节中描述的菜单项里完成。如果需要使用加密密码，请参看 6.1.6.2 加密密码章节。

警示：

如果使用了错误的菜单项格式，或使用了错误的方法修改了配置文件，则会导致系统启动失败。

在/etc/grub.d/00_header 和/etc/grub.d/01_users 文件中，由设定的超级用户完成一些修改，实现对所有菜单项的密码保护。00_header 文件比较复杂，如果可能的话，尽量不要修改它。菜单项应该存放在/etc/grub.d/40_custom 文件中；用户存放在/etc/grub.d/01_users 文件中。在 kickstart 模板中，当使用了 grub 引导加载密码后，由安装应用程序 anaconda 生成 01_users 文件。在这节的例子中，我们要遵循这个规定。

指定菜单项设置用户和密码保护

1. 在/etc/grub.d/01_users 文件中，通过添加下面的行来指定超级用户，这里 john 是指派的超级用户名，johnspassword 是超级用户密码：

```
cat <<EOF
set superusers="john"
password john johnspassword
EOF
```

2. 在/etc/grub.d/01_users 文件的最后，为每个用户添加上额外的描述，就可以使得这些用户也能访问这些项了。

```
cat <<EOF
set superusers="john"
password john johnspassword
password jane janespassword
EOF
```

3. 当设置用户和密码后，在/etc/grub.d/40_custom 文件中指定的密码保护项，就

和下图给出的例子格式类似：

```
menuentry 'NeoKylin Linux Advanced Server' --unrestricted {  
set root=(hd0,msdos1)  
linux    /vmlinuz  
}  
  
menuentry 'Fedora' --users jane {  
set root=(hd0,msdos2)  
linux    /vmlinuz  
}  
  
menuentry 'NeoKylin Linux Workstation' {  
set root=(hd0,msdos3)  
linux    /vmlinuz  
}
```

上例中：

- john 是可以访问所有项的超级用户，他能使用 GRUB2 命令行，在系统启动时，编辑 GRUB2 的选项。在上例中，John 既可以访问 NeoKylin Linux Advanced Server 和 Fedora，也可以访问 NeoKylin Linux Workstation。注意：只有 john 可以访问 NeoKylin Linux Workstation，因为它既没有使用--users 选项，也没有使用 unrestricted 选项。
- 因为在配置文件中给 jane 用户授权了，所以他能引导 Fedora。
- 因为在 NeoKylin Linux Advanced Server 的配置上有 unrestricted 选项，所以任何人都能引导 NeoKylin Linux Advanced Server，但是只有 john 能作为超级用户编辑菜单项。一旦定义了超级用户，则未经授权的更改就都无效了，并且对于未设定 unrestricted 的所有项都有针对引导的保护。

如果针对某菜单项没有指定用户，或带有 unrestricted 选项，则只有超级用户有权访问这个系统。

当针对模板文件做了修改，则 GRUB2 配置文件一定也会被更新。

执行以下 `grub2-mkconfig -o` 命令，可以重建 `grub.cfg` 文件：

- 基于 BIOS 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- 基于 UEFI 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/efi/EFI/neokylin/grub.cfg
```

加密密码

默认情况下，密码是以纯文本形式存放于 GRUB2 脚本中的。没有正确的密码，在系统启动时该文件是不能被访问的。通过使用 `grub2-mkpasswd-pbkdf2` 命令加密密码，可以更好地提高系统的安全性。`grub2-mkpasswd-pbkdf2` 命令可以转换密码成一个长哈希，并保存于 GRUB2 脚本中，而不再是保存密码的纯文本于文件中了。

1. 为了生成加密密码，需要由 root 用户以命令行方式执行 `grub2-mkpasswd-pbkdf2` 命令。
2. 当被提示需要输入密码和重复输入密码时，输入需要加密的密码，然后，命令则会以加密形式输出加密后的密码。
3. 拷贝粘贴这个长哈希到需要配置用户的模板文件中。也就是，`/etc/grub.d/01_users` 或 `/etc/grub.d/40_custom` 文件中。

将下面 “John” 用户的密码应用于 `01_users` 文件：

```
cat <<EOF
set superusers="john"
password_pbkdf2                                john
grub.pbkdf2.sha512.10000.19074739ED80F115963D984BDCB35AA671C24325755
377C3E9B014D862DA6ACC77BC110EED41822800A87FD3700C037320E51E932
6188D53247EC0722DDF15FC.C56EC0738911AD86CEA55546139FEBC366A393
DF9785A8F44D3E51BF09DB980BAFEF85281CBBC56778D8B19DC94833EA834
2F7D73E3A1AA30B205091F1015A85
EOF
```

下面的格式可以适用于 40_custom 文件：

```
set superusers="john"

password_pbkdf2                                john
grub.pbkdf2.sha512.10000.19074739ED80F115963D984BDCB35AA671C24325755
377C3E9B014D862DA6ACC77BC110EED41822800A87FD3700C037320E51E932
6188D53247EC0722DDF15FC.C56EC0738911AD86CEA55546139FEBC366A393
DF9785A8F44D3E51BF09DB980BAFEF85281CBBC56778D8B19DC94833EA834
2F7D73E3A1AA30B205091F1015A85
```

6.1.7 重新安装 GRUB2

为了确定一个 GRUB2 的错误安装，文件丢失或故障系统产生的原因，最简单的方法就是重新安装 GRUB2。再有需要重新安装 GRUB2 的其他原因如下：

- 升级 GRUB 版本
- 用户需要用 GRUB2 引导加载工具来控制安装了的操作系统。可是，有一些操作系统安装时，却带有它们自己的引导工具。这时，需要重新安装 GRUB2 再来控制所需的操作系统。
- 为其他驱动添加引导信息

重新安装 GRUB2 到基于 BIOS 引导的系统

当使用 `grub2-install` 命令来更新引导信息以及恢复丢失的文件时，注意，需要恢复的文件一定不能被损坏。

如果系统正常运行着，则可以使用 `grub2-install device` 命令来重新安装 GRUB2。例如：如果 `sda` 是你正在使用的系统所在的分区：

```
~]# grub2-install /dev/sda
```

重新安装 GRUB2 到基于 UEFI 引导的系统

当使用 `yum reinstall grub2-efi shim` 命令来更新引导信息以及恢复丢失的文件时，注意，需要恢复的文件一定不能被损坏。

如果系统正常运行着，则可以使用 `yum reinstall grub2-efi shim` 命令来重新安装 GRUB2。例如：

```
~]# yum reinstall grub2-efi shim
```

重新设置和安装 GRUB2

这个方法会完全删除 GRUB2 的配置和系统设置。应用这个方法将重新设置所有的配置到它们的默认值。删除配置文件和以后的重新安装 GRUB2 可以修复损害的文件和错误配置导致的失败。为此，需要由 root 以后完成以下步骤：

1. 运行 `rm /etc/grub.d/*` 命令
2. 运行 `rm /etc/sysconfig/grub` 命令
3. 仅对 EFI 系统运行下列命令：

```
~]# yum reinstall grub2-efi shim
```

4. 执行以下 `grub2-mkconfig -o` 命令，重建 `grub.cfg` 文件：

➤ 基于 BIOS 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

➤ 基于 UEFI 的系统，需要由 root 用户执行以下命令：

```
~]# grub2-mkconfig -o /boot/efi/EFI/neokylin/grub.cfg
```

5. 现在按照 24.7 重新安装 GRUB2 的描述，在/boot 分区上恢复了 GRUB2。

6.1.8 GRUB2 配置串口控制台

如果计算机不带有显示和键盘，则通过串口来控制机器就显得非常有用。

配置 GRUB2 选项

设置系统使用串口终端仅适用于当前引导的系统，因为修改的引导参数仅对当前引导的系统有效。当 GRUB2 引导菜单出现时，移动光标到需要启动的系统，按下 `e` 键去完成内核参数的修改。删除 `rhgb` 和 `quiet` 参数，并在 `linux16` 行的最后添加如下的控制台参数：

```
linux16 /vmlinuz-3.10.0-327.el7.x86_64
root=UUID=c883439f-45fb-42f2-8167-06036ff2eb0a ro crashkernel=auto
LANG=zh_CN.UTF-8 vconsole.font=latarcyrheb-sun32 console=ttyS0,115200
```

这些设置都是临时有效，仅适用于单次引导。

可以使用 `grubby` 工具使得对系统菜单项的修改永久生效。例如：使用以下命令可以更新默认引导内核项。

```
~]# grubby --remove-args="rhgb quiet" --args=console=ttyS0,115200
```

```
--update-kernel=DEFAULT
```

--update-kernel 参数也可以设成关键字 ALL，或用逗号分开的内核索引列表。有关 grubby 工具更多的详细信息，请参看 24.4 从 GRUB 菜单项中添加和删除参数。

在/etc/default/grub 文件中,添加以下 2 行就可以完成新 GRUB2 配置文件的生成。

```
GRUB_TERMINAL="serial"
GRUB_SERIAL_COMMAND="serial --speed=9600 --unit=0 --word=8 --parity=no
--stop=1"
```

上图中的第一行禁用了图形终端。将会用 GRUB_TERMINAL 值重写 GRUB_TERMINAL_INPUT 和 GRUB_TERMINAL_OUTPUT 的值。第二行调整了传输速率、奇偶校验值以及其他值,以适应环境和硬件。一个更高的传输速率,例如: 115200, 对于一些特殊的传输任务可能更为可取, 如: 日志文件的传输。一旦完成了/etc/default/grub 文件的修改, 就必须更新到 GRUB2 配置文件中。

执行以下 grub2-mkconfig -o 命令, 重建 grub.cfg 文件:

- 基于 BIOS 的系统, 需要由 root 用户执行以下命令:

```
~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- 基于 UEFI 的系统, 需要由 root 用户执行以下命令:

```
~]# grub2-mkconfig -o /boot/efi/EFI/neokylin/grub.cfg
```

注意:

为了通过串口连接访问 grub 终端, 一个附加选项必须要添加到内核定义中, 这样, 才能使得这种特殊的内核能监控到串口连接。例如:

```
console=ttyS0,9600n8
```

这里, console= ttyS0 表示当前使用的串口终端设备名 ttyS0; 波特率为 9600; n 表示无奇偶校验; 8 位数据位。一个更高的传输速率, 例如: 115200, 对于一些特殊的传输任务可能更为可取, 如: 日志文件的传输。要想了解更多有关串口控制台的信息, 请参看 26.9 可安装文件和外部文档。

用 screen 工具连接到串口控制台

用户可以将 screen 工具当作一个可用串口终端来使用。由 root 用户执行以下

命令来安装它：

```
~]# yum install screen
```

用下列命令创建串口控制台，并连接到你的机器：

```
screen /dev/console_port baud_rate
```

默认情况下，若不带参数，screen 用的是标准的 9600 波特率。为了设置更高的波特率，需要执行以下命令：

```
~]$ screen /dev/console_port 115200
```

这里，console_port 指 ttyS0 或 ttyUSB0 等。

为了结束 screen 会话，可以按下 Ctrl+a，并输入 quit，最后按下 Enter。

要想了解附加选项和更多详细信息，请参看 screen（1）的 man 手册。

6.1.9 开机时的终端菜单编辑

系统启动过程中，可以修改菜单项，传输参数给内核。在引导加载工具菜单里的任一菜单项上，按下 e 键，就可触发进入菜单编辑界面，开始编辑操作。按下 Esc 键将放弃任一修改，并重新回到标准菜单界面。按下 c 键进入命令行界面。

命令行界面是一个已授权的 GRUB 界面，也是最常用的控制界面。可以在命令行界面上，通过键入 GRUB 相关命令和 Enter 键来执行它们。这个界面还具有一些 shell 的高级功能，例如：**【Tab】** 键能基于上下文完成模糊匹配；**【Ctrl+a】** 移动光标到行首；**【Ctrl+e】** 移动光标到行尾。另外，上下左右的方向键，**【Home】** 键，**【End】** 键和 **【Delete】** 键的功能和 bash shell 中完全相同。

启动到救援模式

救援模式提供了方便的单用户环境，当不能正常完成系统启动时，可以允许你进入单用户环境，修复系统。在救援模式下，系统试图加载所有本地文件系统，并启动一些重要的系统服务，但它不能激活网络接口，允许多用户同时登录到系统。在 NeoKylin Linux Advanced Server V7 中，救援模式等效于单用户模式，进入它需要 root 密码。

1. 在系统启动时，进入救援模式。在 GRUB2 引导界面上，键入 **【e】** 键进入编辑。
2. 在 64 位 IBM Power 系列的系统上，在 linux 行尾添加上下列参数；在基于

BIOS 引导的 x86 64 位系统上, 在 `linux16` 行尾添加上下列参数; 在基于 UEFI 引导的系统上, 在 `linuxefi` 行尾上添加上下列参数:

```
systemd.unit=rescue.target
```

按下【**Ctrl+a**】和【**Ctrl+e**】可分别跳转到行首和行尾。在某些系统中,【**Home**】键和【**End**】键也能起到同样作用。

注意: 等效参数 `1`, `s` 和 `single` 也能被传递到内核。

3. 按下【**Ctrl+x**】将携带参数引导系统

启动到应急模式

应急模式提供了用于修复系统的最小可用环境, 甚至, 此时系统都不可能进入救援模式。在应急模式中, 系统为了读取根文件系统仅仅加载了它, 不会试图加载其它本地文件系统, 也不激活网络接口, 仅仅启动一些基本的服务。在 NeoKylin Linux Advanced Server V7 中, 进入应急模式需要 `root` 密码。

1. 在系统启动时, 进入应急模式。在 GRUB2 引导界面上, 键入 `e` 键进入编辑。
2. 在 64 位 IBM Power 系列的系统上, 在 `linux` 行尾添加上下列参数; 在基于 BIOS 引导的 x86 64 位系统上, 在 `linux16` 行尾添加上下列参数; 在基于 UEFI 引导的系统上, 在 `linuxefi` 行尾上添加上下列参数:

```
systemd.unit=emergency.target
```

按下 `Ctrl+a` 和 `Ctrl+e` 可分别跳转到行首和行尾。在某些系统中, `Home` 键和 `End` 键也能起到同样作用。

注意: 等效参数 `emergency` 和 `-b` 也能被传递到内核。

3. 按下 `Ctrl+x` 将携带参数引导系统

启动到 debug shell

在启动过程中, `systemd debug shell` 提供一个很早版本的 `shell`, 它被用来诊断和系统相关的启动问题。一旦进入 `debug shell`, 如: `systemctl list-jobs` 和 `systemctl list-units` 的一些 `systemctl` 命令就可以被用来查询导致启动失败的原因。另外, `debug` 选项也可被添加到内核命令行中, 这样, 会导致日志信息量的增加。针对 `systemd` 命令, 添加内核命令行选项 `debug` 的一个简洁的方式是 `systemd.log`

_level=debug。

过程：添加 Debug Shell 命令

仅仅为了在一次会话中接活 debug shell，需要完成以下操作：

1. 在 GRUB2 启动界面，移动光标到需要编辑的菜单项上，然后，按下 e 键进入编辑。
2. 在 64 位 IBM Power 系列的系统上，在 linux 行尾添加上下列参数；在基于 BIOS 引导的 x86 64 位系统上，在 linux16 行尾添加上下列参数；在基于 UEFI 引导的系统上，在 linuxefi 行尾上添加上下列参数：

systemd.debug-shell

可选择添加 debug 选项。

按下 Ctrl+a 和 Ctrl+e 可分别跳转到行首和行尾。在某些系统中，Home 键和 End 键也能起到同样作用。

3. 按下 Ctrl+x 将携带参数引导系统

如果需要的话，可以用 `systemctl enable debug-shell` 命令，设置在每次启动时启用 debug shell。另外，使用 grubby 工具也可以使得在 GRUB2 选项中，对内核命令行的修改永久生效。有关 grubby 使用上的更多信息，请参看 24.4 用 grubby 工具永久修改 GRUB2 选项。

警示：

因为使用 debug shell 是不需要授权的，所以永久启用 debug shell 是存在安全隐患的。当 debug 会话结束后禁用它。

过程：连接到 Debug Shell

在启动过程中，`systemd -debug -generator` 将配置 debug shell 到 TTY9 上。

1. 按下 Ctrl +Alt+F9 将连接到 debug shell。如果使用的是虚拟机，则发送这组键组合需要得到虚拟机应用的支持。如果使用的是虚拟机管理工具，则从菜单中的 send key 里选择 Ctrl +Alt+F9。
2. 因为 debug shell 不需要认证，因此，在 TTY9 上的系统提示类似于：
[root@localhost/]#
3. 可以输入以下命令来验证是否在 debug shell 状态下


```
/]# systemctl status $$
```

● debug-shell.service - Early root shell on /dev/tty9 FOR DEBUGGING ONLY

Loaded: loaded (/usr/lib/systemd/system/debug-shell.service; disabled; vendor preset: disabled)

Active: active (running) since Wed 2015-08-05 11:01:48 EDT; 2min ago

Docs: man:sushell(8)

Main PID: 450 (bash)

CGroup: /system.slice/debug-shell.service

└─ 450 /bin/bash

└─1791 systemctl status 450

4. 正常启动后，按下 Ctrl +Alt+F1 可以返回到默认 shell（如：bash shell）终端上。

为了诊断启动问题，在内核命令中，可以通过 `systemd.mask=unit_name` 命令一次或多次掩蔽掉某些 `systemd` 单元来分段排查。为了在系统启动过程中，启动其他单位，可以通过在内核命令中添加 `systemd.wants=unit_name` 来实现。在 `system-d ebug-generator(8)`手册中有一些相关选项的描述。

修改和重设 root 密码

在安装 NeoKylin Linux Advanced Server V7 过程中，设置 root 密码是一个强制性的操作。如果忘记或丢失了 root 密码，则可以用它来重新设置。然而，wheel 组的用户是可以改变 root 密码的，如下：

```
~]$ sudo passwd root
```

注意，在 GRUB2 中，不再支持 NeoKylin Linux Advanced Server V6 中单用户模式下的 root 密码重设方式。在进入 single-user 和 emergency 模式操作时，是需要 root 密码的。

在这里展示了 2 个重置 root 密码的过程：

- 用安装盘重设 root 密码，使用户不必编辑 grub 菜单，而是进入 shell 环境。

它是两个过程中较简洁的方法，也是推荐的方法。可以用引导盘或正常的 NeoKylin Linux Advanced Server V7 安装盘来完成 root 密码重置。

- 用 `rd.break` 重置 `root` 密码,它就是利用控制从 `initramfs` 转到 `systemd` 前,中断启动过程来完成 `root` 密码重置的。这个方法的缺点就是步骤多,包含:必要的 `GRUB` 项编辑;可以在耗时的 `SELinux` 文件重新标注和改变 `SELinux` 强制模式之间做出选择;启动完成后,需要恢复 `/etc/shadow` 的 `SELinux` 安全形式。

过程: 用安装盘重设 `root` 密码

1. 启动系统,当显示 `BIOS` 信息时,引导菜单,然后,选择从安装盘引导。
2. 选择故障排除
3. 选择救援 `NeoKylin Linux Advanced Server` 系统
4. 选择默认选项继续,如果此时发现有加密文件系统,则会提示输入密码。
5. 选择 `OK` 知晓一些回显信息,直到 `shell` 提示符出现。
6. 按如下方法改变根文件系统:

```
sh-4.2# chroot /mnt/sysimage
```

7. 执行 `passwd` 命令,根据此命令回显的提示,修改 `root` 密码。
8. 删除 `autorelabel` 文件,防止盘被 `SELinux` 重标而耗时
9. 执行 `exit` 命令即可退出 `chroot` 环境
10. 再执行 `exit` 命令返回继续初始化,完成系统启动

过程: 用 `rd.break` 重置 `root` 密码

1. 启动系统,在 `GRUB2` 引导界面,按下 `e` 键进入编辑
2. 在 `linux16` 行尾或靠近行尾处删除 `rhgb` 和 `quiet` 参数;在 `UEFI` 系统的 `linux16efi` 后删除 `rhgb` 和 `quiet` 参数。

按下 `Ctrl+a` 和 `Ctrl+e` 可以分别跳到行首和行尾。在有些系统上,可以用 `Home` 键和 `End` 键来完成同样的操作。

重点:

为了启用显示系统信息,必须删除 `rhgb` 和 `quiet` 参数。

3. 在 64 位 `IBM Power` 系列的系统上,在 `linux` 行尾添加上下列参数;在基于 `BIOS` 引导的 `x86 64` 位系统上,在 `linux16` 行尾添加上下列参数;在基于 `UEFI` 引导的系统上,在 `linuxefi` 行尾上添加上下列参数:

```
rd.break enforcing=0
```

添加 `enforcing=0` 选项略去耗时的 SELinux 标识。

在控制转到内核前，`initramfs` 要停止启用根文件系统工作。

注意：在 Linux 行中指出的最后一个控制台上出现 `initramfs` 提示符。

4. 按下 `Ctrl+x` 将用修改了的参数启动系统

在此，用加密文件系统需要密码。当日志信息覆盖时，就不出现密码提示符了。当忽略日志信息时，在按下 `Backspace` 键就能看到密码提示符，放开 `Backspace` 键，输入加密文件系统的密码即可。

出现 `initramfs switch_root` 提示。

5. 以只读方式加载文件系统到 `/sysroot` 下，如果文件系统不可写，则不允许修改密码。

以读写方式重新加载文件系统：

```
switch_root:/# mount -o remount,rw /sysroot
```

6. 以可写方式重新加载文件系统

用以下命令改变文件系统的根目录：

```
switch_root:/# chroot /sysroot
```

系统提示符会变成：`sh-4.2#`

7. 执行 `passwd` 命令，根据此命令回显的提示，修改 `root` 密码。

注意：如果系统不可写，则 `passwd` 命令会执行失败，给出以下错误信息：

```
Authentication token manipulation error
```

8. 更新密码文件会导致使用不正确的 SELinux 安全上下文。为了在下次启动系统时，能标注所有文件需要执行以下命令：

```
sh-4.2# touch /.autorelabel
```

另外，为了节省标注大磁盘所花费的时间，可以像第 3 步那样，加入 `enforcing=0` 而忽略这一步。

9. 以只读方式重新加载文件系统：

```
sh-4.2# mount -o remount,ro /
```

10. 执行 `exit` 命令即可退出 `chroot` 环境

11. 再执行 `exit` 命令返回继续初始化，完成系统启动。

在此，使用加密文件系统时，需要密码。当日志信息覆盖时，就不出现密码提示符了。当忽略日志信息时，在按下 `Backspace` 键看到密码提示符，放开 `Backspace` 键，输入加密文件系统密码即可。

注意：

`SELinux` 标注过程会花费较长时间。当整个过程结束后，系统会自动重启。如果在第 3 步添加了 `enforcing=0`；在第 8 步省略执行了 `touch /.autorelabel`，那么，执行下列命令就可以恢复 `/etc/shadow` 文件的 `SELinux` 安全形式了。

```
~]# restorcon /etc/shadow
```

执行下列命令，可以强制执行 `SELinux` 策略恢复和验证其正确性。

```
~]# setenforce 1
~]# getenforce
Enforcing
```

6.1.10 安全启动统一可扩展固件接口（UEFI）

统一可扩展固件接口（UEFI）安全启动技术要确保由系统固件去检查是否使用了存放于固件公共密钥库里的加密密钥，对系统启动加载工具进行数字签名。对下一阶段所使用的引导加载工具和内核的签名验证，可以防止没有被可信密钥签名的内核空间代码在系统中执行。

从固件到签名驱动和内核模块建立一个如下的信任链。引导加载的第一阶段，要把经过 UEFI 私钥签名和通过公钥（证书颁发机构签发的）认证的 `shim.efi` 存放于固件库中。`shim.efi` 包含有 NeoKylin 公钥“NeoKylin Secure Boot (CA key 1)”，它被用于 GRUB2 引导加载工具，`grubx64.efi` 和 NeoKylin 内核的认证。相应地，内核包含了一些用于认证驱动和模块的公钥。

安全引导是一个按照统一可扩展固件接口（UEFI）规范定义的引导路径验证组件。规范定义如下：

- 在非易失性存储中，用编程接口实现了 UEFI 变量的密码级保护
- 如何将可信的 X.509 根证书保存于 UEFI 变量中
- UEFI 应用的验证，如：引导加载工具和驱动的验证

➤ 取消一个恶意证书和应用哈希的过程

UEFI 安全引导既不妨碍第二阶段引导加载工具的安装或移出，也不需要明确的用户确认。在引导过程中，验证数字签名，而不是在引导加载工具的安装和更新时验证。因此，UEFI 安全引导不会停止引导路径的维护，这将有助于检测未经授权的更改。只要用系统可信密钥签名了，新的引导加载工具和内核就可以运行了。

支持 UEFI 安全引导

NeoKylin Linux Advanced Server V7 包括了对 UEFI 安全引导功能的支持，这意味着 NeoKylin Linux Advanced Server V7 能被安装和运行在启用了 UEFI 安全引导的硬件系统上。基于 UEFI 引导的系统，一旦启用了安全引导技术，则所有的驱动必须是被可信密钥签名过的，否则，系统将不接受它们。用 NeoKylin 的一个私钥签名的所有驱动必须用内核中和私钥一致的 NeoKylin 公钥来验证。

如果需要使用来自于 NeoKylin Linux Advanced Server DVD 以外的所建驱动，则必须确认已对这些驱动做过数字签名的。

有关签名自定义驱动的更多详细信息，请参看 26.8 为安全引导签名内核模块。

实施 UEFI 安全引导所涉及的限制

在 NeoKylin Linux Advanced Server V7 中，引入 UEFI 安全引导是为了确保系统仅仅运行已被验证签名通过的内核态代码，因此，会存在一些限制。

在不支持 GRUB2 模块签名和验证的架构系统中，不启用 GRUB 2 模块加载，这就意味着可以在安全引导定义的安全边界内，加载由不可信代码构成的模块。取而代之的是 NeoKylin 提供了一个签名了的 GRUB2 二进制码，它已经包含在 NeoKylin Linux Advanced Server V7 中了，支持所有的模块。

有关更多详细信息，请参看 NeoKylin 知识库里的《UEFI 安全引导实施限制》文章。

6.2 手动升级内核

由 NeoKylin Linux Advanced Server V7 核心组定制的 NeoKylin Linux Advanced Server 内核，必须要确保它的完整性和硬件兼容性。Neokylin 在发布

内核前，首先必须要通过一套严格的质量保证测试。

NeoKylin Linux Advanced Server 内核是以 rpm 软件包格式发布的，因此，能很容易地使用 yum 或 packagekit 软件包管理工具维护它们。

本章仅适合于用 rpm 命令手动更新内核的用户。

警示：

只要可能，用户还是要尽可能地用 yum 或 packagekit 工具安装新内核，而不是替代当前的内核，替代操作会带来系统引导失败的风险。

有关使用 yum 工具安装软件包的更多信息，请参看 7.1.2 更新包。

6.2.1 内核软件包概述

NeoKylin Linux Advanced Server 包含有下列内核软件包：

- kernel --- 包含有针对单处理器，多核和多处理器系统的内核；
- kernel-debug --- 包含有针对内核诊断可以启用的大量调试选项，启用它会降低一些系统性能；
- kernel-devel --- 包含有内核软件包模块构建时必须的头文件和 makefile 文件；
- kernel-debug-devel --- 包含有内核诊断用的大量调试选项的开发包，一旦启用调试功能会大大降低系统的性能；
- kernel-doc --- 针对内核的文档文件。文档涉及到内核各部分和设备驱动程序。这个包的安装提供了一个在内核加载时，可以传给内核模块的参考选项；
默认情况下，这些文件存放于/usr/share/doc/kernel-doc-kernel_version 目录中。
- kernel-headers --- 包含针对 Linux 内核，用户空间库和程序的指定接口 C 代码的头文件。在这些头文件中，定义了构建大多数标准程序所需的结构和常量说明。
- linux-firmware --- 包含有各种设备操作所需的固件文件。
- perf --- 包含启用 Linux 内核性能监控的 perf 工具。
- kernel-abi-whitelists --- 包含针对外部 Linux 内核模块所需的列表和辅助

yum 插件的内核 ABI 信息。

- kernel-tools --- 包含有内核操作工具和支持文档

6.2.2 升级准备工作

在升级内核前，建议做一些准备工作。

以防万一，首先确认系统的引导介质可用。如果引导加载工具不适合新内核的引导，则需要用这个介质先启动进入 NeoKylin Linux Advanced Server。

USB 介质常常来源于闪存设备，通常称为笔驱动器，拇指磁盘，或钥匙，也有作为外部链接的硬盘驱动。这种类型的所有介质都是采用 VFAT 文件系统的。你可以建成 ext2，ext3，ext4 或 VFAT 文件系统的 USB 引导盘。

需要将一个发布的镜像文件或最小的引导镜像文件拷贝到 USB 盘上。注意要确保 USB 盘上有足够的存储空间。一个发布的 DVD 镜像大约需要 4GB；一个发布的 CD 镜像大约需要 700MB；最小的引导介质镜像大约需要 10MB。

需要将 NeoKylin Linux Advanced Server DVD 安装盘或 CD-ROM 介质安装的第一张上的 boot.iso 文件拷贝到一个 VFAT 文件系统的 USB 盘上，这个 USB 盘大约需要 16MB。下面的操作过程不会影响到 USB 盘上已有的文件，除非拷贝到 USB 盘上的文件和盘中文件有相同的路径名。由 root 用户执行下列命令可以创建 USB 引导盘：

1. 如果系统中没有安装 syslinux 软件包的话，可以由 root 用户执行 `yum install syslinux` 命令来安装。
2. 在 USB 盘上，安装 SYSLINUX 引导加载工具。

```
~]# syslinux /dev/sdX1
```

这里，sdX 是设备名

3. 创建 boot.iso 和 USB 盘的加载点

```
~]# mkdir /mnt/isoboot /mnt/diskboot
```

4. 把 boot.iso 加载到/mnt/isoboot 目录

```
~]# mount -o loop boot.iso /mnt/isoboot
```

5. 把 USB 盘加载到/mnt/diskboot 目录

```
~]# mount /dev/sdX1 /mnt/diskboot
```

6. 拷贝 boot.iso 中的 ISOLINUX 文件到 USB 盘上

```
~]# cp /mnt/isoboot/isolinux/* /mnt/diskboot
```

7. 在 USB 盘中，利用 boot.iso 的 isolinux.cfg 文件生成 syslinux.cfg 文件

```
~]# grep -v local /mnt/isoboot/isolinux/isolinux.cfg > /mnt/diskboot/syslinux.cfg
```

8. 卸载 boot.iso 和 USB 盘

```
~]# umount /mnt/isoboot /mnt/diskboot
```

9. 在继续之前，需要用引导介质重启机器，以便验证能不能正常启动。

另外，带有软驱的系统，可以由 root 用户先安装 mkbootdisk 软件包，然后通过执行 mkbootdisk 命令来创建一个引导软盘。有关 mkbootdisk 命令的使用方法，请参看 mkbootdisk 命令的 man 手册。

在 shell 提示符下，执行 yum list installed "kernel-*" 命令可以查看已安装了的内核软件包。以下此命令的输出，另外，命令回显的信息还依赖于系统架构和版本号。

```
~]# yum list installed "kernel-*"

Loaded plugins: langpacks

Installed Packages

kernel.x86_64                                     3.10.0-327.el7
@anaconda/rawhide

kernel-devel.x86_64                               3.10.0-327.el7
@anaconda/rawhide

kernel-headers.x86_64                             3.10.0-327.el7
@anaconda/rawhide

kernel-tools.x86_64                               3.10.0-327.el7
@anaconda/rawhide

kernel-tools-libs.x86_64                           3.10.0-327.el7
@anaconda/rawhide
```

从上述命令的输出可以决定内核升级需要下载哪些包。对于单处理器系统，仅仅需要内核软件。有关不同软件包的描述，请参看 25.1 内核软件包概述。

6.2.3 下载升级内核

有几种方法，可以用来确定一个待更新的内核是否适用于当前系统。

➤ 安全勘误表

- NKUC --- 对于在 NKKUC 上注册了的系统，可以使用 yum 软件包管理工具下载最新的内核，然后，升级系统内核。如果需要的话，可以用 Dracut 应用程序创建一个初始 RAM 磁盘镜像，为引导新内核配置引导加载工具。有关从 NKUC 上如何安装软件包的更多信息，请参看 7 yum。有关在 NKUC 上如何注册系统的更多信息，请参看 NKUC 的相关管理手册。

如果想要使用 yum 命令，从 NKUC 上下载软件包来更新内核的话，则只需参看 6.2.5 验证初始 RAM 磁盘镜像和 6.2.6 验证引导加载工具的配置，它不会改变默认引导内核的。NKUC 会自动改变默认引导内核的版本。若需手动安装的话，请参看 6.2.4 准备升级内核。

6.2.4 准备升级内核

准备好所有相关的软件包后，就可以开始升级现有的内核了。

重点：

预防新内核万一有问题，强力推荐先保存好旧内核。

在 shell 提示符下，将 kernel 的 rpm 包所在目录设为当前目录，用带有 -i 参数的 rpm 命令来安装新内核，这样，可以保留旧内核；因为用 -U 参数会覆盖当前已安装的内核，可能会导致引导出问题，所以没选用 -U 参数。例如：

```
~]# rpm -ivh kernel-kernel_version.arch.rpm
```

下一步验证创建的初始 RAM 磁盘镜像。更多相关信息，请参看 25.5 验证初始 RAM 磁盘镜像。

6.2.5 验证初始 RAM 磁盘镜像

初始 RAM 磁盘镜像是用于加载块设备驱动模块的，如：IDE, SCSI 或 RAID，存放这些块设备驱动模块的根文件系统一定要是可加载和可访问的。在 NeoKylin Linux Advanced Server V7 中，无论用 Yum, PackageKit 还是 RPM 包管理工具安装的新内核，都会用安装脚本调用 Dracut 应用程序来创建 initramfs（初始 RAM 磁盘镜像）的。

除了 IBM eServer System i 以外的所有其它体系架构的系统，都可以通过执行 dracut 应用程序来创建 initramfs。不过，通常不需要手动创建 initramfs，如果内核及相关包都是来自于 NKUC 发布的 RPM 软件包的话，会在安装或升级内核时，自动创建 initramfs 文件的。

可以用下列过程验证 initramfs 和当前内核版本是否一致，配置的 grub.cfg 文件是否正确。

过程：验证初始 RAM 磁盘镜像

1. 由 root 用户列出 /boot 目录下的文件，找出最新版本的内核（vmlinuz-kernel_version）和 initramfs-kernel_version。

实例：确保内核与 initramfs 版本匹配

```
~]# ls /boot/
System.map-3.10.0-327.el7.x86_64
config-3.10.0-327.el7.x86_64
grub2
initramfs-0-rescue-ea311367055a4bcfb323ece036b6bf37.img
initramfs-3.10.0-327.el7.x86_64.img
initramfs-3.10.0-327.el7.x86_64kdump.img
initrd-plymouth.img
symvers-3.10.0-327.el7.x86_64.gz
vmlinuz-0-rescue-ea311367055a4bcfb323ece036b6bf37
vmlinuz-3.10.0-327.el7.x86_64
```

上例说明如下：

- 安装了 2 个内核（或更准确地说，在 /boot 目录下存在有 2 个内核文件）
- 最新版本的内核是 vmlinuz-3.10.0-327.el7.x86_64
- 和 最 新 版 本 的 内 核 相 匹 配 的 initramfs 文 件 是 initramfs-3.10.0-327.el7.x86_64.img

重点：

在 /boot 目录下，有几个 initramfs-kernel_versionkdump.img 文件，它们是为了调

试内核，而创建的 Kdump 机制文件，不能用于系统引导，所以可以忽略它，不会有问题。有关 kdump 的更多信息，请参看《NeoKylin Linux Advanced Server V7 内核崩溃转储指南》

2. （可选）在/boot 目录下，如果 initramfs-kernel_versionkdump.img 文件的版本和最新的内核版本不一致的话，则需要用 Dracut 应用程序来生成一个一致的 initramfs 文件。不用带任何参数，由 root 用户简单执行 dracut 命令就可以在 /boot 目录下，生成一个和最新内核版本相匹配的 initramfs 文件。

```
~]# dracut
```

如果想要用新创建的 initramfs 文件替代旧的相应文件，则需要执行 dracut 命令时带上--force 参数(例如: 如果现有 initramfs 已损坏, 可以重新创建它)。否则，会拒绝覆盖现有的 initramfs 文件。

```
[root@node1 ~]# dracut
Will not override existing initramfs (/boot/initramfs-3.10.0-327.el7.x86_64.img)
without --force

Broadcast message from systemd-journald@node1.test (Tue 2016-01-05 14:30:21
CST):

dracut[23908]:      Will      not      override      existing      initramfs
(/boot/initramfs-3.10.0-327.el7.x86_64.img) without --force

Message from syslogd@node1 at Jan  5 14:30:21 ...
dracut:Will      not      override      existing      initramfs
(/boot/initramfs-3.10.0-327.el7.x86_64.img) without --force
```

执行 dracut initramfs_name kernel_version 命令，可以在当前目录下创建 initramfs 文件。

```
~]# dracut "initramfs-$(uname -r).img" $(uname -r)
```

如果需要在预加载时指定内核模块的话，则可以按照 `add _dracutmodules+="module[more_modules]"` 格式，在 `/etc/dracut.conf` 配置文件中，添加上所需加载内核模块对应的 `*.ko` 文件（注意：去掉它们的后缀 `.ko`）。可以用 `lsinitrd initramfs_file` 命令查看由 `dracut` 命令生成的 `initramfs` 文件的内容。

```
~]# lsinitrd /boot/initramfs-3.10.0-327.el7.x86_64.img
```

```
Image: /boot/initramfs-3.10.0-327.el7.x86_64.img: 27M
```

```
Version: dracut-033-359.el7
```

```
Arguments: -f
```

```
dracut modules:
```

```
bash
```

```
nss-softokn
```

```
i18n
```

```
network
```

```
ifcfg
```

```
drm
```

```
plymouth
```

```
kernel-modules
```

```
fcoe
```

```
resume
```

```
rootfs-block
```

```
terminfo
```

```
udev-rules
```

```
biosdevname
```

```
systemd
```

```
usrmount
```

```
base
```

fs-lib				
shutdown				
drwxr-xr-x	12	root	root	0 Dec 31 08:49 .
crw-r--r--	1	root	root	5, 1 Dec 31 08:49 dev/console
crw-r--r--	1	root	root	1, 11 Dec 31 08:49 dev/kmsg
crw-r--r--	1	root	root	1, 3 Dec 31 08:49 dev/null
lrwxrwxrwx	1	root	root	7 Dec 31 08:49 bin -> usr/bin
drwxr-xr-x	2	root	root	0 Dec 31 08:49 dev
drwxr-xr-x	12	root	root	0 Dec 31 08:49 etc
drwxr-xr-x	2	root	root	0 Dec 31 08:49 etc/cmdline.d
drwxr-xr-x	2	root	root	0 Dec 31 08:49 etc/conf.d
-rw-r--r--	1	root	root	124 Dec 31 08:49 etc/conf.d/systemd.conf
-rw-r--r--	1	root	root	164 Sep 12 2013 etc/dhclient.conf
[output truncated]				

可以用 `man dracut` 和 `man dracut.conf`，了解有关 `dracut` 命令的更多信息。

3. 检查 `/boot/grub2/grub.cfg` 配置文件中的 `initramfs-kernel_version.img`，可以确认当前引导内核的版本。例如：

```

~]# grep initramfs /boot/grub2/grub.cfg

    initrd16 /initramfs-3.10.0-327.el7.x86_64.img

    initrd16 /initramfs-0-rescue-ea311367055a4bcfb323ece036b6bf37.img
    
```

查看 6.2.6 验证引导加载工具，可以了解更多信息。

在 IBM eServer System i 系统中，验证初始 RAM 磁盘镜像和内核

在 IBM eServer System i 系统中，由 `addRamDisk` 命令创建的初始 RAM 磁盘镜像和内核合二为一成了一个文件。如果内核及相关软件包都是来自于 NKUC 发布的 RPM 包的话，则会在安装或升级内核时，自动创建 `initramfs` 文件。由 `root` 用户执行下列命令来确认 `/boot/vmlinitrd-kernel_version` 文件的存在，验证 `initramfs` 文件的创建。

```
ls -l /boot/
```

`kernel_version` 应该和已安装的内核版本一致。

6.2.6 验证引导加载工具

当用 `rpm` 工具安装了新内核后，内核软件包会在引导加载工具的配置文件中添加上有关新内核的描述项。但是，`rpm` 工具是不会配置新内核作为默认引导内核的，需要用 `rpm` 工具在安装时手动完成设置。

强力推荐用 `rpm` 工具完成新内核的安装后，一定要反复检查确认配置的文件是否正确。一旦有误，会导致引导失败，此时，就需要重新修改相应的配置文件了。

6.3 内核模块

Linux 拥有模块化结构的内核，这样，可以通过动态加载模块来扩展整个系统的能力。内核模块可以提供：

- 添加新硬件的设备驱动
- 添加对某个文件系统的支持，例如：`btrfs` 和 `NFS`

大多数情况下，针对内核本身，可以使自定义参数和默认内核参数配合一起使用。使用工具可以查看当前加载到运行内核中的模块，结合参数设置可查询出所有有效的内核；在运行的内核中动态加载/卸载模块。此类工具中的大多数都是来自于 `kmod` 软件包，同时，也考虑到了模块间的依赖关系，当执行操作时，几乎不需要依赖手动跟踪。

当对内核模块有特殊要求时，系统总能以各种机制来自动加载它们。然而，也有一些特殊情况是需要手动加载或卸载模块的，例如：虽然两个模块都能提供类似的基本功能，但是其中有一个更合适；或者一个模块出现问题了。这些情况多会需要手动加载模块的。

- 用 `kmod` 工具可以显示，查询，加载和卸载内核模块，查询模块间的依赖关系；
- 用命令可以动态设置内核模块参数，一旦设置永久生效后，可以自定义内核模块的操作；
- 引导时自动加载模块；

注意:

在这一章中, 为了使用这些工具, 首先需要 root 用户确认 kmod 包是不是已经正确安装。安装 kmod 软件包的命令格式, 如下所示:

```
~]# yum install kmod
```

想了解如何使用 yum 安装包的更多信息, 请参看 7.2.4 安装包。

6.3.1 查看当前加载的模块

通过执行 lsmod 命令, 可以查看当前所有加载到系统中的内核模块。例如:

```
~]# lsmod

Module                Size  Used by
tcp_lp                12663  0
nls_utf8              12557  1
isofs                 39844  1
bnep                  19704  2
bluetooth             372944  5 bnep
rfkill                26536  3 bluetooth
fuse                  87741  3
xt_CHECKSUM           12549  1
ipt_MASQUERADE        12678  3
nf_nat_masquerade_ipv4 13412  1 ipt_MASQUERADE
tun                   27141  1
ip6t_rpfilter         12546  1
ip6t_REJECT           12939  2
ipt_REJECT            12541  4
xt_conntrack          12760  8
ebtable_nat           12807  0
ebtable_broute        12731  0
bridge               119560  1 ebtable_broute
stp                   12976  1 bridge
```

llc	14552	2	stp,bridge
ebtable_filter	12827	0	
ebtables	30913	3	ebtable_broute,ebtable_nat,ebtable_filter
ip6table_nat	12864	1	
nf_conntrack_ipv6	18738	5	
nf_defrag_ipv6	34768	1	nf_conntrack_ipv6
nf_nat_ipv6	14131	1	ip6table_nat
ip6table_mangle	12700	1	
ip6table_security	12710	1	
ip6table_raw	12683	1	
ip6table_filter	12815	1	
ip6_tables		27025	5
ip6table_filter,ip6table_mangle,ip6table_security,ip6table_nat,ip6table_raw			
[output truncated]			

lsmod 命令输出的逐行说明：

- 当前加载到内存中的内核模块名
- 模块所占内存的大小
- 所有正在使用这个模块的进程数总和，并列出了所有依赖于它的其它模块；
使用这个列表可以卸载一个模块的所有依赖模块。更多详细信息，请参看 26.4 卸载模块

lsmod 命令的输出比/proc/modules 文件的内容更简短易读。

6.3.2 查看模块信息

使用 modinfo module_name 可以显示更加详细的内核模块信息。

注意：

作为 kmod 工具参数的内核模块名是不带扩展名的。内核模块名是不含扩展名的，它的相关文件也是如此。

实例：用 lsmod 命令查看内核模块的相关信息

e1000e 是 Intel PRO/1000 网卡的驱动模块，可以由 root 用户执行以下命令获得该

模块的信息:

```
~]# modinfo e1000e
filename:
/lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/net/ethernet/intel/e1000e/e1000e.k
o
version:      3.2.5-k
license:      GPL
description:   Intel(R) PRO/1000 Network Driver
author:       Intel Corporation, <linux.nics@intel.com>
rhelversion:   7.2
srcversion:    7097C005F85B5C9D374D3FB
alias:         pci:v00008086d000015B8sv*sd*bc*sc*i*
alias:         pci:v00008086d000015B7sv*sd*bc*sc*i*
alias:         pci:v00008086d00001570sv*sd*bc*sc*i*
alias:         pci:v00008086d0000156Fsv*sd*bc*sc*i*
alias:         pci:v00008086d000015A3sv*sd*bc*sc*i*
alias:         pci:v00008086d000015A2sv*sd*bc*sc*i*
alias:         pci:v00008086d000015A1sv*sd*bc*sc*i*
alias:         pci:v00008086d000015A0sv*sd*bc*sc*i*
alias:         pci:v00008086d00001559sv*sd*bc*sc*i*
alias:         pci:v00008086d0000155Asv*sd*bc*sc*i*
alias:         pci:v00008086d0000153Bsv*sd*bc*sc*i*
alias:         pci:v00008086d0000153Asv*sd*bc*sc*i*
alias:         pci:v00008086d00001503sv*sd*bc*sc*i*
alias:         pci:v00008086d00001502sv*sd*bc*sc*i*
alias:         pci:v00008086d000010F0sv*sd*bc*sc*i*
alias:         pci:v00008086d000010EFsv*sd*bc*sc*i*
alias:         pci:v00008086d000010EBsv*sd*bc*sc*i*
```


alias:	pci:v00008086d000010EAsv*sd*bc*sc*i*
alias:	pci:v00008086d00001525sv*sd*bc*sc*i*
alias:	pci:v00008086d000010DFsv*sd*bc*sc*i*
alias:	pci:v00008086d000010DEsv*sd*bc*sc*i*
alias:	pci:v00008086d000010CEsv*sd*bc*sc*i*
alias:	pci:v00008086d000010CDsv*sd*bc*sc*i*
alias:	pci:v00008086d000010CCsv*sd*bc*sc*i*
alias:	pci:v00008086d000010CBsv*sd*bc*sc*i*
alias:	pci:v00008086d000010F5sv*sd*bc*sc*i*
alias:	pci:v00008086d000010BFsv*sd*bc*sc*i*
alias:	pci:v00008086d000010E5sv*sd*bc*sc*i*
alias:	pci:v00008086d0000294Csv*sd*bc*sc*i*
alias:	pci:v00008086d000010BDsv*sd*bc*sc*i*
alias:	pci:v00008086d000010C3sv*sd*bc*sc*i*
alias:	pci:v00008086d000010C2sv*sd*bc*sc*i*
alias:	pci:v00008086d000010C0sv*sd*bc*sc*i*
alias:	pci:v00008086d00001501sv*sd*bc*sc*i*
alias:	pci:v00008086d00001049sv*sd*bc*sc*i*
alias:	pci:v00008086d0000104Dsv*sd*bc*sc*i*
alias:	pci:v00008086d0000104Bsv*sd*bc*sc*i*
alias:	pci:v00008086d0000104Asv*sd*bc*sc*i*
alias:	pci:v00008086d000010C4sv*sd*bc*sc*i*
alias:	pci:v00008086d000010C5sv*sd*bc*sc*i*
alias:	pci:v00008086d0000104Csv*sd*bc*sc*i*
alias:	pci:v00008086d000010BBsv*sd*bc*sc*i*
alias:	pci:v00008086d00001098sv*sd*bc*sc*i*
alias:	pci:v00008086d000010BAsv*sd*bc*sc*i*
alias:	pci:v00008086d00001096sv*sd*bc*sc*i*

alias:	pci:v00008086d0000150Csv*sd*bc*sc*i*
alias:	pci:v00008086d000010F6sv*sd*bc*sc*i*
alias:	pci:v00008086d000010D3sv*sd*bc*sc*i*
alias:	pci:v00008086d0000109Asv*sd*bc*sc*i*
alias:	pci:v00008086d0000108Csv*sd*bc*sc*i*
alias:	pci:v00008086d0000108Bsv*sd*bc*sc*i*
alias:	pci:v00008086d0000107Fsv*sd*bc*sc*i*
alias:	pci:v00008086d0000107Esv*sd*bc*sc*i*
alias:	pci:v00008086d0000107Dsv*sd*bc*sc*i*
alias:	pci:v00008086d000010B9sv*sd*bc*sc*i*
alias:	pci:v00008086d000010D5sv*sd*bc*sc*i*
alias:	pci:v00008086d000010DAsv*sd*bc*sc*i*
alias:	pci:v00008086d000010D9sv*sd*bc*sc*i*
alias:	pci:v00008086d00001060sv*sd*bc*sc*i*
alias:	pci:v00008086d000010A5sv*sd*bc*sc*i*
alias:	pci:v00008086d000010BCsv*sd*bc*sc*i*
alias:	pci:v00008086d000010A4sv*sd*bc*sc*i*
alias:	pci:v00008086d0000105Fsv*sd*bc*sc*i*
alias:	pci:v00008086d0000105Esv*sd*bc*sc*i*
depends:	ptp
intree:	Y
vermagic:	3.10.0-327.el7.x86_64 SMP mod_unload modversions
signer:	Red Hat Enterprise Linux kernel signing key
sig_key:	
EA:14:CC:4A:E5:DC:B2:10:A3:B0:B5:77:00:54:A1:AD:34:71:10:A3	
sig_hashalgo:	sha256
parm:	debug:Debug level (0=none,...,16=all) (int)
parm:	copybreak:Maximum size of packet that is copied to a new buffer

on receive (uint)

parm:	TxIntDelay:Transmit Interrupt Delay (array of int)
parm:	TxAbsIntDelay:Transmit Absolute Interrupt Delay (array of int)
parm:	RxIntDelay:Receive Interrupt Delay (array of int)
parm:	RxAbsIntDelay:Receive Absolute Interrupt Delay (array of int)
parm:	InterruptThrottleRate:Interrupt Throttling Rate (array of int)
parm:	IntMode:Interrupt Mode (array of int)
parm:	SmartPowerDownEnable:Enable PHY smart power down (array of int)
parm:	KumeranLockLoss:Enable Kumeran lock loss workaround (array of int)
parm:	WriteProtectNVM:Write-protect NVM [WARNING: disabling this can lead to corrupted NVM] (array of int)
parm:	CrcStripping:Enable CRC Stripping, disable if your BMC needs the CRC (array of int)

针对上述 modinfo 命令输出中，域的描述说明：

filename

针对.ko 内核对象文件的绝对路径。可以 modinfo -n 作为一个仅输出 filename 域的简单命令。

description

针对模块的简短描述。可以 modinfo -d 作为一个仅输出 description 域的简单命令。

alias

一个模块可以有多个别名。从别名域可看出模块的多别名。如果没有别名，则可以完全省略。

depends

表示这个模块所依赖的模块

注意：

如果一个模块不依赖于任何模块，则 **depends** 域可以省略。

parm

模块参数格式：parameter_name:description 。这里：

- parameter_name: 必须准确使用。当需要将模块作为命令参数，或者作为 /etc/modprobe.d/*.conf 配置文件中的选择行时，都会用到 parameter_name。
- description: 一个参数的简要说明，在圆括号中的参数类型。（例如：整数型，单元或整型数组）

实例：查看模块参数

用参数 -p 可以查到模块支持的所有参数。但是，modinfo -p 输出中有些有用的数值类型会被省略掉，所以更多有用信息的输出，如下所示：

```
~]# modinfo e1000e | grep "^parm" | sort
parm:          CrcStripping:Enable CRC Stripping, disable if your BMC needs
the CRC (array of int)
parm:          IntMode:Interrupt Mode (array of int)
parm:          InterruptThrottleRate:Interrupt Throttling Rate (array of int)
parm:          KumeranLockLoss:Enable Kumeran lock loss workaround (array
of int)
parm:          RxAbsIntDelay:Receive Absolute Interrupt Delay (array of int)
parm:          RxIntDelay:Receive Interrupt Delay (array of int)
parm:          SmartPowerDownEnable:Enable PHY smart power down (array
of int)
parm:          TxAbsIntDelay:Transmit Absolute Interrupt Delay (array of int)
parm:          TxIntDelay:Transmit Interrupt Delay (array of int)
parm:          WriteProtectNVM:Write-protect NVM [WARNING: disabling
this can lead to corrupted NVM] (array of int)
parm:          copybreak:Maximum size of packet that is copied to a new buffer
on receive (uint)
parm:          debug:Debug level (0=none,...,16=all) (int)
```

6.3.3 加载模块

可以由 root 用户执行 `modprobe module_name` 命令来加载内核模块。例如：
要加载 `wacom` 模块，可以执行以下命令：

```
~]# modprobe wacom
```

默认情况下，`modprobe` 命令会试图加载 `/lib/modules/kernel_version/kernel/drivers/` 目录下的模块。此目录下的每种模块类型，都有自己的子目录，如：`net/`和 `scsi/` 目录分别存放网卡和 `SCSI` 接口的驱动。

有些模块是有依赖关系的，在加载一个模块前，必须先加载其所依赖的内核模块，否则，会加载失败。在完成加载的过程中，`modprobe` 命令会考虑到模块间的依赖关系，并将依赖软件包一起安装上。当使用 `modprobe` 命令加载指定内核模块时，`modprobe` 首先会去检查这个模块的依赖关系，如果存在有和其它模块间的依赖关系，并且这些被依赖的模块还没有被安装，则需要先安装它们，最后才能安装指定模块。`modprobe` 命令能递归解析依赖关系。如果需要的话，`modprobe` 会加载所有有依赖关系的模块，因此，必须要确认模块的所有依赖关系都能满足。

可以用 `modprobe` 命令带参数 `-v`（或 `--verbose`），参看正在加载模块的所有依赖关系的详细信息。

实例：`modprobe -v` 展示加载模块的依赖关系

执行以下命令加载 `Fibre Channel over Ethernet` 的多个模块：

```
~]# modprobe -v fcoe
insmod /lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/scsi/scsi_tgt.ko
insmod /lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/scsi/scsi_transport_fc.ko
insmod /lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/scsi/libfc/libfc.ko
insmod /lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/scsi/fcoe/libfcoe.ko
insmod /lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/scsi/fcoe/fcoe.ko
```

在这个例子中，可以看到在最终加载 `fcoe` 之前，我们加载了依赖的 `scsi_tgt`，`scsi_transport_fc`，`libfc` 和 `libfcoe`。也注意到了 `modprobe` 命令调用了早期的 `insmod` 命令，完成模块的插入。

重点:

虽然 `insmod` 命令也能被用来加载内核模块，但是它不能解析依赖关系。正因为如此，我们总是采用 `modprobe` 命令来替代 `insmod` 命令，完成内核模块加载。

6.3.4 卸载内核模块

可以由 root 用户执行 `modprobe -r module_name` 命令来卸载内核模块。例如：
假设 `wacom` 模块已经被加载到了内核中，我们可以执行以下命令卸载它：

```
~]# modprobe -r wacom
```

然而，如果一个进程涉及到以下情况，那么执行这个命令就会失败：

- 正在使用 `wacom` 模块
- 正在使用 `wacom` 所依赖的模块
- 正在使用 `wacom` 间接依赖的模块

参看 6.3.1 查看当前加载的模块，可以了解更多有关，如何使用 `lsmod` 命令获得模块名的信息，防止误删模块。

实例：卸载内核模块

如果要卸载 `firewire_ohci` 模块，则需要中断类似于下面的会话：

```
~]# modinfo -F depends firewire_ohci
firewire-core
~]# modinfo -F depends firewire_core
crc-itu-t
~]# modinfo -F depends crc-itu-t
```

这样，就可以找到加载 `Firewire` 模块时形成的依赖树了（在这个例子中没有分支）。

`firewire_ohci` 依赖于 `firewire_core`，`firewire_core` 又依赖于 `crc-itu-t`。

可以用 `modprobe -v -r module_name` 命令卸载掉 `firewire_ohci` 模块，这里的 `-r` 是 `--remove` 的缩写；`-v` 是 `--verbose` 的缩写。如下：

```
~]# modprobe -r -v firewire_ohci
rmmod firewire_ohci
rmmod firewire_core
rmmod crc_itu_t
```

命令输出了模块卸载的顺序，它和加载的顺序正好相反，直到没有进程再依赖于这些正在被卸载的任一模块。

重点：

虽然 `rmmod` 命令也能用来删除内核模块，但还是推荐使用 `modprobe -r` 命令。

6.3.5 设置模块参数

就像内核一样，模块也可以通过参数来选择其不同的功能。大多数情况下，默认就已经可以支持地很好了，但是，偶尔也会希望为模块自定义一些参数。加载到内核中的模块是不能动态修改参数的，下面有两个不同的方法用于设置参数：

1. 首先，使用 `modprobe -r` 命令，卸载掉所有设置了参数的相互依赖模块。然后，用 `modprobe` 命令及一组自定义参数列表来重新加载这些模块。此方法常用于依赖模块不多的情况下，或者测试参数的关联性，并不需要固化参数。这是本节所涵盖的方法。
2. 另外，可以查看 `/etc/modprobe.d/` 目录下，新定义参数或新建文件。此方法可以永久设定模块参数，以确保每次重新引导或用 `modprobe` 命令加载模块都使用上这些参数。有关此方法的相关信息，请参看 26.6 自动加载模块。

实例：模块加载时提供可选参数

用下列命令格式，可以基于自定义参数加载模块：

```
~]# modprobe module_name [parameter=value]
```

在命令行中，用自定义参数加载模块时，需要注意：

- 支持用空格分隔的多参数
- 有一些参数需要用逗号分隔它们的值。在输入值列表中，不可在逗号后带
有空格，避免 `modprobe` 错误地解释空格为其它参数值。
- `modprobe` 命令成功完成以下操作后的返回值应为 0
 - 成功加载模块
 - 模块已加载到内核

因此，在试图用自定义参数加载模块前，要先确认该模块没有被加载。

`modprobe` 命令不会自动加载模块，或者告知模块已经加载。

推荐按这里的步骤来设置自定义参数，然后，加载内核模块。这个演示过程

用的是 Intel PRO/1000 网卡的 e1000e 网卡驱动模块。例如：

过程：用自定义参数加载模块

1. 首先确认模块没有被加载到内核

```
~]# lsmod |grep e1000e  
~]#
```

输出若提示模块已经加载到了内核，此时，就要先卸载模块，然后再做以下操作。有关如何安全卸载模块，请参看 26.4 卸载内核模块

2. 在模块名后，列出全部自定义参数。例如：如果需要将参数中断节流率设置 3000，共 3 次来加载 Intel PRO/1000 网卡驱动，同时还要打开 debug 功能，则可以由 root 用户执行以下命令来实现。

```
~]# modprobe e1000e InterruptThrottleRate=3000,3000,3000 debug=1
```

这个实例演示了如何给一个参数传递用逗号分隔开的多个参数值，忽略参数间的空格。

6.3.6 自动加载模块

正如实例“查看当前加载的模块”所示，许多模块是在系统引导时被自动加载的。在/etc/modules-load.d/目录下，通过创建 program.conf（其中：program 是所选模块名称）可以让 systemd-modules-load.service 守护进程加载其指定的模块。在/etc/modules-load.d/目录下的文本文件，每一行对应一个待加载模块。

实例：加载模块的配置文件

创建/etc/modules-load.d/virtio-net.conf 文件的内容，如下所示。就可以在系统引导时，自动加载上 virtio-net 模块了。

```
# Load virtio-net.ko at boot  
virtio-net
```

想了解更多相关信息，请参看 modules-load.d(5)和 systemd-modules-load.service(8)的 man 手册。

6.3.7 从驱动更新磁盘安装模块

硬件驱动模块都是以驱动更新磁盘（DUD）形式提供给用户的。通常在安装时，使用驱动更新磁盘或 ISO 镜像来加载并安装任一需要的硬件驱动，这一过

程可以参看《NeoKylin Linux Advanced Server V7 安装指导》。然而，如果在安装后，需要用新的驱动模块，则就要用到下列过程。若已有 RPM 软件包，则可以直接跳到第 5 步。

过程：从驱动更新磁盘（DUD）安装新模块

按照以下系统安装后操作，完成从驱动更新磁盘（DUD）安装新硬件驱动模块：

1. 安装驱动更新磁盘
2. 由 root 用户创建加载目录和加载 DUD

```
~]# mkdir /run/OEMDRV
~]# mount -r -t iso9660 /dev/sr0 /run/OEMDRV
```

3. 查看 DUD，例如：

```
~]# ls /run/OEMDRV/
rhdd3  rpms  src
```

4. 将 DUD 中和系统体系架构一致的目录设置为当前目录，该目录路径中包含有 rpms 子目录，查看当前目录。例如：

```
~]# cd /run/OEMDRV/rpms/x86_64/
~]# ls
kmod-bnx2x-1.710.51-3.el7_0.x86_64.rpm
kmod-bnx2x-firmware-1.710.51-3.el7_0.x86_64.rpm  repodata
```

上述输出中，包版本号为 **1.710.51**；发行号为 3.el7_0。

5. 同时安装多个 RPM 文件，例如：

```
~]# yum install kmod-bnx2x-1.710.51-3.el7_0.x86_64.rpm
kmod-bnx2x-firmware-1.710.51-3.el7_0.x86_64.rpm
Loaded plugins: product-id, subscription-manager
This system is not registered to Red Hat Subscription Management. You can use
subscription-manager to register.
Examining kmod-bnx2x-1.710.51-3.el7_0.x86_64.rpm:
kmod-bnx2x-1.710.51-3.el7_0.x86_64
Marking kmod-bnx2x-1.710.51-3.el7_0.x86_64.rpm to be installed
```

```

Examining                                kmod-bnx2x-firmware-1.710.51-3.el7_0.x86_64.rpm:
kmod-bnx2x-firmware-1.710.51-3.el7_0.x86_64
Marking kmod-bnx2x-firmware-1.710.51-3.el7_0.x86_64.rpm to be installed
Resolving Dependencies
--> Running transaction check
---> Package kmod-bnx2x.x86_64 0:1.710.51-3.el7_0 will be installed
---> Package kmod-bnx2x-firmware.x86_64 0:1.710.51-3.el7_0 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====

=====

Package                Arch      Version                Repository
=====
=====

Installing:
  kmod-bnx2x                                x86_64      1.710.51-3.el7_0
/kmod-bnx2x-1.710.51-3.el7_0.x86_64
  kmod-bnx2x-firmware                        x86_64      1.710.51-3.el7_0
/kmod-bnx2x-firmware-1.710.51-3.el7_0.x86_64

Transaction Summary

=====

=====

Install 2 Packages

Total size: 1.6 M
    
```

Installed size: 1.6 M

Is this ok [y/d/N]:

6. 执行以下命令验证所有模块及依赖关系的完整性

```
~]# depmod -a
```

7. 执行以下命令完成初始 RAM 文件系统的备份

```
~]# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).img.$(date +%m-%d-%H%M%S).bak
```

8. 重建初始 RAM 文件系统

```
~]# dracut -f -v
```

9. 执行以下命令，查看初始 RAM 文件系统镜像的内容

```
~]# lsinitrd /boot/initramfs-3.10.0-327.el7.x86_64.img
```

上述命令输出很长，可以用 less 命令或 grep 命令，结合管道找出更新的模块。

例如：

```
~# lsinitrd /boot/initramfs-3.10.0-327.el7.x86_64.img | grep bnx
drwxr-xr-x 2 root root          0 Jun  9 11:25 usr/lib/firmware/bnx2x
-rw-r--r--  1 root root          164392 Nov  25 2014
usr/lib/firmware/bnx2x/bnx2x-e1-7.10.51.0.fw
-rw-r--r--  1 root root          173016 Nov  25 2014
usr/lib/firmware/bnx2x/bnx2x-e1h-7.10.51.0.fw
-rw-r--r--  1 root root          321456 Nov  25 2014
usr/lib/firmware/bnx2x/bnx2x-e2-7.10.51.0.fw
drwxr-xr-x  2 root root          0 Jun  9 11:25
usr/lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/net/ethernet/broadcom/bnx2x
-rw-r--r--  1 root root          1034553 Jan  29 19:11
usr/lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/net/ethernet/broadcom/bnx2x/b
nx2x.ko
```

10. 重启系统使更新生效

如果需要的话，可以执行以下命令查看当前内核驱动程序。

```

~]# modinfo bnx2x
filename:
/lib/modules/3.10.0-327.el7.x86_64/kernel/drivers/net/ethernet/broadcom/bnx2x/bnx2
x.ko
firmware:      bnx2x/bnx2x-e2-7.10.51.0.fw
firmware:      bnx2x/bnx2x-e1h-7.10.51.0.fw
firmware:      bnx2x/bnx2x-e1-7.10.51.0.fw
version:       1.710.51-0
license:       GPL
description:           Broadcom           NetXtreme           II
BCM57710/57711/57711E/57712/57712_MF/57800/57800_MF/57810/57810_MF/57
840/57840_MF Driver
author:         Eliezer Tamir
rhelversion:    7.1
    
```

6.3.8 数字签名内核模块

NeoKylin Linux Advanced Server V7 支持 UEFI 安全引导机制，这就表明在启用了 UEFI 安全引导的系统上，可以安装和运行 NeoKylin Linux Advanced Server V7。注意：在 UEFI 系统上，NeoKylin Linux Advanced Server V7 不需要使用安装引导。

当启用安全引导后，由 EFI 操作系统引导加载工具。NeoKylin Linux Advanced Server 内核和所有的内核模块都必须用私钥签名，用与之对应的公钥验证。NeoKylin Linux Advanced Server V7 发布了签名的引导加载工具，签名的内核和内核模块。另外，签名的第一阶段引导加载工具和签名内核都包含有嵌入的 NeoKylin 公钥。这些签名的可执行二进制代码和嵌入的密钥使得 NeoKylin Linux Advanced Server V7 在安装引导和运行时能使用 Microsoft UEFI 的安全引导 CA 密钥，这些密钥是由支持 UEFI 安全引导的系统 UEFI 固件提供的。注意：不是所有基于 UEFI 的系统都包含有对安全引导的支持。

本节描述在支持安全引导的 UEFI 系统中，如何由 NeoKylin Linux Advanced

Server V7 使用内置私钥为内核模块签名。还介绍了一些有效方法，利用它们，可以得到进入内核系统的公钥。

预配置

下表列出了一些可用于对外部模块签名的工具。

表格 6-1 需要的工具

工具	所在包		适用于	目的
openssl	openssl		构建系统	生成遵循 X.509 标准的一对公钥和私钥
sign-file	kernel-devel		构建系统	用于签名内核模块的 Perl 脚本
perl	perl		构建系统	用于运行签名脚本的 Perl 解释器
mokutil	mokutil		目标系统	可选工具，用于手动注册公钥
keyctl	keyctl		目标系统	可选工具，用于在系统密钥环内显示公钥

注意：构建系统中，在构建和签名内核模块时，不需要启用 UEFI 安全引导，甚至不需要基于 UEFI 系统。

验证内核模块

在 NeoKylin Linux Advanced Server V7 中，当加载内核模块时，会用内核系统密钥环中遵循 X.509 标准的公钥去检查模块的数字签名。不包括内核系统黑名单

单密钥环中的那些密钥。

验证内核模块的公钥源

在系统引导时，内核加载遵循 X.509 标准的密钥到系统密钥环中，或者加载到来自系统黑名单密钥环的一组持久密钥存储中，正如表格 6-2 系统密钥环源所示。

表格 6-2 系统密钥环源

X.509 密钥源	用户添加密钥的限制	UEFI 安全引导状态	系统引导中加载密钥
嵌入内核	No	-	.system_keyring
UEFI 安全引导 “db”	限制	未启用	No
		启用	.system_keyring
UEFI 安全引导 “dbx”	限制	未启用	No
		启用	.system_keyring
嵌入在 shim.efi 引导加载工具中	No	未启用	No
		启用	.system_keyring
机器自带密钥列表	Yes	未启用	No
		启用	.system_keyring

注意：如果不是基于 UEFI 系统或没有启用 UEFI 安全引导，则只有嵌入内核的密钥才被加载到系统密钥环中，并且不重构内核是不能增加密钥组的。系统黑名单密钥环是一个已经被取消的 X.509 列表组。如果模块是用黑名单密钥环中的密钥签名的，则即使公钥在系统密钥环中，验证也会失败。

可以用 keyctl 应用程序查看系统密钥环中密钥的信息。下例是一个来自没有启用 UEFI 安全引导的实例输出缩写：

```

~]# keyctl list %:.system_keyring
3 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Enterprise Linux kernel signing key:
    
```

```
4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key:
4d38fd864ebe18c5f0b7...
```

下例是一个来自启用了 UEFI 安全引导的实例输出缩写：

```
~]# keyctl list %::system_keyring
6 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Secure Boot (CA key 1):
4016841644ce3a810408050766e8f8a29...
...asymmetric: Microsoft Corporation UEFI CA 2011:
13adbf4309bd82709c8cd54f316ed...
...asymmetric: Microsoft Windows Production PCA 2011:
a92902398e16c49778cd90f99e...
...asymmetric: Red Hat Enterprise Linux kernel signing key:
4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key:
4d38fd864ebe18c5f0b7...
```

上面的输出中展示了两个额外的密钥，一个来自于 UEFI 安全引导 “db”；另一个来自于嵌入到 shim.efi 引导加载工具中的 Red Hat Secure Boot (CA key 1)。也能得到一些确定与 UEFI 安全引导密钥相关的内核控制信息，也就是 UEFI 安全引导 db，嵌入的 shim 和 MOK 列表。

```
~]# dmesg | grep 'EFI: Loaded cert'
[5.160660] EFI: Loaded cert 'Microsoft Windows Production PCA 2011: a9290239...
[5.160674] EFI: Loaded cert 'Microsoft Corporation UEFI CA 2011: 13adbf4309b...
[5.165794] EFI: Loaded cert 'Red Hat Secure Boot (CA key 1): 4016841644ce3a8...
```

内核模块验证需求

如果启用了 UEFI 安全引导或指定了 `module.sig_enforce` 内核参数，则仅需要用系统密钥环中的密钥验证这个签名内核模块，就可以成功加载它了，提供的

公钥不能来自于系统黑名单密钥环。如果没有启用 UEFI 安全引导或没有指定 module.sig_enforce 内核参数，则没有公钥也能成功加载未签名和签名的内核模块。总结归纳了表格 6-3 内核模块验证加载需求。

表格 6-3 内核模块验证加载需求

模块签名	公钥和有效签名	UEFI 安全引导状态	Module.sig_enforce	模块加载	损害内核
未签名	-	未启用	未启用	成功	Yes
		未启用	启用	失败	
		启用	-	失败	-
签名	No	未启用	未启用	成功	Yes
		未启用	启用	失败	-
		启用	-	失败	-
签名	Yes	未启用	未启用	成功	No
		未启用	启用	成功	No
		启用	-	成功	No

后续章节将描述如何生成遵循 X.509 标准的一对公钥和私钥；如何用私钥签名内核模块；如何注册成为系统密钥环中的公钥。

生成遵循 X.509 标准的一对公钥和私钥

遵循 X.509 标准，生成以后用于签名内核模块的一对公钥和密钥。在加载内核模块时，将使用对应的公钥来验证内核模块。

1. 在 NeoKylin Linux Advanced Server V7 中，可以用 openssl 工具生成满足特殊要求的密钥对。生成密钥的一些参数最好用配置文件制定。下列给出了一个配置文件实例：

```

~]# cat << EOF > configuration_file.config
[ req ]
default_bits = 4096
distinguished_name = req_distinguished_name
    
```



```
prompt = no
string_mask = utf8only
x509_extensions = myexts

[ req_distinguished_name ]
O = Organization
CN = Organization signing key
emailAddress = E-mail address

[ myexts ]
basicConstraints=critical,CA:FALSE
keyUsage=digitalSignature
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
EOF
```

2. 在创建配置文件后，就可以生成遵循 X.509 标准的密钥对了。公钥写入到 public_key.der 文件中；私钥写入到 private_key.priv 中。

```
~]# openssl req -x509 -new -nodes -utf8 -sha256 -days 36500 \ > -batch -config
configuration_file.config -outform DER \ > -out public_key.der \ > -keyout
private_key.priv
```

3. 在所有需要验证和加载内核模块的系统上，注册公钥。

警示：

请妥善保管您的私钥内容。无一丢失，这个密钥可以使得拥有公钥的系统出现安全隐患。

在目标系统上注册公钥

在启用了安全引导的 UEFI 系统上，引导 NeoKylin Linux Advanced Server V7 时，保存于安全引导密钥库（而非存于废弃密钥 dbx 数据库中）里的所有密钥，都将由内核加载到系统密钥环中，用于以后内核模块的验证。

包括公钥的原厂固件映像

为了方便验证系统内核模块，可以考虑要求系统供应商，将公钥注入到原厂固件镜像中的 UEFI 安全引导密钥库里。

借助可执行密钥注册镜像添加公钥

通过编写和提供一个 EFI 可执行注册镜像，可以添加一个密钥到现有的存活安全引导密钥库中。这样的注册镜像对要追加到安全引导密钥库中的密钥是有特殊要求的。这个要求就是已经在安全引导密钥交换密钥库（KEK）中的公钥必须和用私钥签名的数据对应上。另外，由私钥签名的这个 EFI 镜像必须和密钥库中已有公钥对应上。

运行于 NeoKylin Linux Advanced Server V7 之上，是可以生成一个注册镜像的。然而，用一个合适的私钥已签名的 NeoKylin Linux Advanced Server V7 镜像必须和一个在 KEK 库中已有的公钥对应上。

无论哪种密钥构建的注册镜像都需要得到平台厂商的支持。

系统管理员手动添加公钥到 MOK 列表中

NeoKylin Linux Advanced Server V7 支持硬件自带密钥（MOK），它可用于增加 UEFI 安全引导密钥库。在启用了安全引导的 UEFI 系统上，NeoKylin Linux Advanced Server V7 允许将 MOK 列表中的密钥添加到系统密钥环中，除非密钥来自于密钥数据库。MOK 中的密钥也可以永久保存，并且和在安全引导密钥库中的密钥一样安全，但是，这是两种机制。shim.efi, MokManager.efi, grubx64.efi 和 NeoKylin Linux Advanced Server V7 的 mokutil 应用程序都支持 MOK。

MOK 的主要功能是添加公钥到 MOK 列表中，而不需要将密钥链接到 KEK 库中的另一个密钥上。但是，需要在每个目标系统中的 UEFI 系统控制台上，由真正的系统用户手动交互来注册上 MOK 密钥。因此，MOK 提供了一个测试新生成密钥对和测试签名内核模块的很好方法。

下面是添加公钥到 MOK 列表的步骤：

1. 使用 NeoKylin Linux Advanced Server V7 应用程序添加公钥到 MOK 列表中所需要的其它信息

```
~]# mokutil --import my_signing_key_pub.der
```

在此，完成 MOK 注册需要输入和确认密码。

2. 重启系统

3. shim.efi 会关注将要注册的 MOK 密钥请求，并会启动 MokManager.efi 来完成在 UEFI 控制台上的注册。注册时，需要输入先前与此请求相关的密码。添加到 MOK 列表中的公钥会永久有效。

一旦一个密钥在 MOK 列表中了，它就会被自动传到系统密钥环上。当启用了 UEFI 安全引导后，以后会引导。

使用私钥签名内核模块

为了签名内核模块不需要完成额外的操作。按照常规方法创建内核模块就可以。配置一个合适的 Makefile 文件和相应的资源，按照下列步骤完成模块的创建和签名：

1. 按照通常的方法创建 my_module.ko 模块

```
~]# make -C /usr/src/kernels/$(uname -r) M=$PWD modules
```

2. 使用 Perl 脚本和私钥签名 my_module.ko 模块。在 Perl 脚本中，需要提供包含私钥和公钥的文件以及待签名的内核模块文件。

```
~]# perl /usr/src/kernels/$(uname -r)/scripts/sign-file \> sha256 \>  
my_signing_key.priv \> my_signing_key_pub.der \> my_module.ko
```

针对 ELF 镜像格式的内核模块，这个脚本计算并直接追加签名到 my_module.ko 文件的 ELF 镜像中。如果要有 modinfo 应用程序的话，可以用它来查看内核模块的签名信息。有关 modinfo 应用程序使用上的更多信息，请参看 26.2 查看模块信息。

注意：这个追加的签名不包含在 ELF 镜像节中，也就不是 ELF 镜像格式中的一部分了。因此，像 readelf 之类的工具就不能查看内核模块的数字签名了。

为了加载准备好的内核模块。即使在没有启用 UEFI 安全引导的系统上，也是可以加载签名内核模块的。这就意味着没必要提供签名和未签名两个版本的内核模块。

加载签名模块

一旦在系统密钥环中注册了公钥，通常的内核模块加载机制对于用户将会是透

明的。在下列例子中，将使用 `mokutil` 工具添加公钥到 `MOK` 列表中，并手动使用 `modprobe` 工具加载内核模块。

1. （可选）在注册公钥前，验证未加载内核模块。首先，在当前系统中，由 `root` 用户执行 `keyctl list %:.system_keyring` 命令来验证已加载到系统密钥环中的密钥。由于你的公钥还没有注册，因此，在命令的输出中是看不到的。

2. 注册公钥

```
~]# mokutil --import my_signing_key_pub.der
```

3. 重启系统，在 UEFI 控制台完成注册

```
~]# reboot
```

4. 重启系统后，再在系统密钥环中验证密钥

```
~]# keyctl list %:.system_keyring
```

5. 现在能成功加载内核模块了


```
~]# modprobe -v my_module  
insmod /lib/modules/3.10.0-327.el7.x86_64/extra/my_module.ko  
~]# lsmod | grep my_module  
my_module 12425 0
```

7 附录 RPM

RPM 包管理器是运行在中标麒麟高级服务器操作系统以及其它 Linux 和 UNIX 系统上的一个开源封装系统。我们鼓励其他供应商使用 RPM 来封装他们的产品。PRM 基于 GPL（通用公共许可证，GNU General Public License）协议发行。

RPM 包管理器只适用于以 RPM 格式构建的包。PRM 本身也是以预安装的 rpm 软件包来提供的。对于最终用户来说，RPM 能够让系统升级更容易。使用简短的命令就可以安装、卸载和升级 RPM 包。RPM 维护了一个记录已安装的包和它们的文件的数据库，因此您可以在您的系统中进行各种查询和验证。还有几款应用程序，例如 Yum 和 PackageKit，能够让使用 RPM 格式的安装包甚至变得


更容易。

 对于大部分包管理任务，Yum 包管理器提供了和 RPM 一样甚至更强大的功能。Yum 还负责执行和追踪复杂的系统依赖关系的解析。Yum 维护着系统的完整性，并且在使用其它应用程序，例如 RPM，来替代 Yum 安装或卸载软件包时，强制进行系统完整性检查。基于以上原因，我们高度推荐您在任何可能执行包管理任务的时候，使用 Yum 来替代 RPM。请参见 2.1 Yum。

如果您更喜欢图形界面，您可以使用 PackageKit 图形用户界面应用程序，它使用 Yum 作为后端，来管理您的系统的安装包。

在升级时，RPM 会细致地处理配置文件，以便您的自定义配置不会丢失——普通的.tar.gz 文件则不能做到这一点。

对于开发者来说，RPM 可以让软件源代码被封装到源码包和二进制包中，供最终用户使用。这个过程很简单，并且仅由一个单一的文件和您可能创建的一些补丁就能驱动这个过程。这种对原始源代码和您的补丁以及构建指令的清晰描述，可以让软件的新版本发布时，包的维护过程变得容易。

 因为 RPM 会对系统进行修改，所以在系统范围内执行诸如安装、升级、降级以及卸载二进制包的操作时，都需要 root 特权。

7.1 RPM 设计目标

理解 RPM 的设计目标，对于理解如何使用 RPM 来说很有帮助：

- 可升级性

使用 RPM，您可以升级您系统中的个别组件，而不用完全重新安装。当您获取到一个基于 RPM 的操作系统的新的发行版时，例如中标麒麟高级服务器操作系统，您不必在您的机器上重新安装该操作系统的一份全新拷贝（而对基于其它封装系统的操作系统，您是可能需要重新安装的）。RPM 可以智能化、自动地升级现有的系统。此外，软件包中的配置文件在升级时被保留，因此用户定制的配置信息不会丢失。在升级一个软件包时，并不需要什么特殊的升级文件，因为在系统中安装和升级软件包都是使用的相同的 RPM 文

件。

- 强大的查询功能

使用 **RPM**，您可以升级您系统中的个别组件，而不用完全重新安装。**RPM** 设计时提供了强大的查询功能。您可以在整个包管理数据库中搜索指定的软件包甚至文件。您还可以轻易地知道哪个文件属于哪个软件包，软件包来自哪里。每个 **RPM** 包有一个二进制头，其中包含软件包本身及其内容的信息，包中的文件被压缩存储，这样就允许快速简捷地查询软件包。

- 系统校验

RPM 另一项强大的功能是软件包校验。它可以让您验证系统中所安装的文件是否和指定的安装包中所提供的文件完全一样。如果检测到了不一致，**RPM** 会通知您，必要时您可以重装该软件包。修改过的配置文件在重装时会被 **RPM** 保留。

- 原始源码

一个重要的设计目标是允许使用原始软件源码，即由软件作者发布的没有打过任何补丁的源码。**RPM** 将原始源码、所用的补丁、以及完整的编译指令放在一个描述文件中，用来驱动将来的包管理过程。这是一个重要的优点，理由有几个，例如，如果软件有新版本推出，你不必从头开始整个编译过程，因为所有缺省的编译选项，补丁信息，要生成的二进制包中文件的列表等等都包含在描述文件了。

保持源码的原始性不仅对开发者重要，它也保证了给最终用户提供的软件的质量。

7.2 使用 RPM


RPM 有五种基本的操作模式（不包括包编译）：安装、卸载、升级、查询和校验。该小节包含了每种模式的概览。要了解完整的细节和选项，请使用 `rpm --help` 或者查看 `rpm(8)` 用户手册页面。

7.2.1 安装和升级软件包

RPM 软件包通常具有以下形式的文件名：

```
package_name-version-release-operating_system-CPU_architecture.rpm
```

例如，文件名 `tree-1.6.0-10.el7.x86_64.rpm` 包含了 `package_name`(`tree`)、`version` (1.6.0)、`release` (10)、`operating_system` (el7) 和 `CPU_architecture` (x86_64)。

 安装软件包的时候，请确保该软件包和您的操作系统以及处理器架构相兼容。这通常可以通过检查软件包名称来确定。例如，一个为 AMD64/Intel 64 计算机架构编译的 RPM 软件包，其文件名以 `x86_64.rpm` 结尾。

`-U`（或者`--upgrade`）选项有两个功能，它可以用来：

- 将您的系统上一个已存在的软件包升级到一个更新的版本
- 如果并没有安装较旧的版本，则直接安装该软件包


因此，命令 `rpm -U package.rpm` 要么升级要么安装，这取决于系统中是否存在该软件包的一个较旧版本。

假设 `tree-1.6.0-10.el7.x86_64.rpm` 软件包位于当前目录下，以 `root` 用户登录，并在 `shell` 命令行提示符下输入以下命令，将升级或安装 `tree` 软件包：

```
~]# rpm -Uvh tree-1.6.0-10.el7.x86_64.rpm
```

`-v` 和 `-h` 选项（和 `-U` 组合在一起的两个选项）让 `rpm` 打印更冗长的输出，并用井号显示一个进度条。如果升级或安装成功，将显示以下输出：

```
Preparing... #####
[100%]
Updating / installing...
1:tree-1.6.0-10.el7 ##### [100%]
```

 `rpm` 提供了两个不同的选项来安装软件包：前面提到的 `-U` 选项（历史上代表升级/`upgrade`），以及 `-i` 选项（历史上代表安装/`install`）。因为 `-U` 选项同时包含了安装和升级功能，因此除了 `kernel` 软件包以外，建议对所有软件包都使用 `rpm -Uvh` 命令。

您应该始终使用 `-i` 选项来安装一个新的 `kernel` 软件包，而不是对其进行升级。这是因为使用 `-U` 选项来升级 `kernel` 软件包时，将删除之前的（较旧的）`kernel` 软件包，这可能会导致如果新的内核有问题的话，系统无法启动。因此，使用 `rpm -i kernel_package` 命令来安装一个新的内核，而不替换任何旧的内核软件包。要了解更多信息安装内核软件包的信息，请参见 6.2 手动升级内核。

在安装或升级软件包时，将自动检查软件包的签名。签名用来确定软件包是由经授权的一方所签发的。如果签名验证失败，将显示一条错误信息。

如果您没有安装合适的密钥来校验签名，信息中将包含 `NOKEY` 这个词：

```
warning: tree-1.6.0-10.el7.x86_64.rpm: Header V3 RSA/SHA256
Signature,
key ID 431d51: NOKEY
```

请参见 7.3.1.2 校验软件包签名了解更多有关校验软件包签名的信息。

7.2.1.1 替换已安装的软件包

如果已经安装了一个名称和版本完全相同的软件包，将会显示以下输出：

```
准备中... #####
[100%]

软件包 tree-1.6.0-10.el7.x86_64 已经安装
```

无论如何都要安装这个包的话，可以使用 `--replacepkgs` 选项，该选项告诉 RPM 忽略此错误：

```
~]# rpm -Uvh --replacepkgs tree-1.6.0-10.el7.x86_64.rpm
```

如果从软件包中安装的文件被删除了，或者您想安装原始的配置文件的，那么这个选项是很有用的。

如果您想安装一个较旧版本的软件包（即已经安装了一个较新版本的软件包），RPM 会通知您已经安装了一个更新的版本。要让 RPM 强制执行降级，请使用 `--oldpackage` 选项：


```
rpm -Uvh --oldpackage older_package.rpm
```

7.2.1.2 解决文件冲突

如果您尝试安装一个软件包，该软件包中包含的一个文件已经被另一个软件包安装过了，则将会显示一条提示冲突的信息。要让 **RPM** 忽略这个错误，请使用 **--replacefiles** 选项：

```
rpm -Uvh --replacefiles package.rpm
```


7.2.1.3 满足未解决的依赖

RPM 软件包有时候会依赖于其它软件包，也就是说要正确地运行这些软件包，需要先安装其它软件包。如果您尝试安装一个具有未解决依赖的软件包，将会显示一条有关依赖不满足的信息。

在中标麒麟高级服务器操作系统的安装光盘或者通过其他途径，找到所提示的安装包，并把它加入到安装命令中。要确定哪个软件包包含了所需要的文件，请使用 **--whatprovides** 选项：

```
rpm -q --whatprovides "required_file"
```

如果包含了 **required_file** 的软件包存在于 **RPM** 数据库中，则将会显示该软件包的名称。

 尽管您能够让 **rpm** 强制安装一个具有未解决依赖的软件包（使用 **--nodeps** 选项），但不推荐这样做，而且这可能通常会导致安装的软件包不能正常运行。使用 **--nodeps** 选项安装软件包会引起应用程序行为异常或者意外终止。它也可能引起严重的包管理问题或者系统失效。基于这些原因，请注意依赖缺失的警告。**Yum** 包管理器会自动解决依赖关系，并从在线仓库中获取依赖包。

7.2.1.4 保留配置文件中的修改

由于 **RPM** 会对配置文件进行包的智能升级，您可能会看到以下信息：

```
saving /etc/configuration_file.conf as  
/etc/configuration_file.conf.rpmsave
```


这表示您之前对配置文件的修改不一定和新的配置文相兼容，因此 **RPM** 将您原来的配置文件保存了，并创建了一个新的配置文件。您应当调查清楚两个配置文件之间的差异，尽可能快地解决好配置文件的问题，以确保您的系统能够继续正确地运作。

可选地，**RPM** 也可能会保存新的配置文件，例如，将新的配置文件保存为 `configuration_file.conf.rpmnew`，而不去触动您修改过的配置文件。您仍然需要尽快地处理好您修改过的配置文件和新配置文件之间的冲突，通常可以通过将旧配置文件中的修改合并到新的配置文件中的方式来完成，例如可以使用 `diff` 程序。

7.2.2 卸载软件包


卸载软件包就像安装软件包一样简单。请以 `root` 用户在 `shell` 命令行提示符下输入以下命令：

```
rpm -e package
```

 注意，该命令只需要软件包的名称，不需要最初安装的软件包的文件名。如果您在尝试使用 `rpm -e` 命令来卸载一个软件包时，提供了原始的完整文件名，您将会接收到一个关于软件包名称的错误。

在卸载软件包时，如果另一个已安装的软件包依赖于您正要卸载的软件包，则您会遇到依赖错误。例如：

```
~]# rpm -e ghostscript
error: Failed dependencies:
ghostscript is needed by (installed) ghostscript-cups-9.07-16.el7.x86_64
ghostscript is needed by (installed) foomatic-4.0.9-6.el7.x86_64
libs.so.9()(64bit) is needed by (installed) libspectre-0.2.7-4.el7.x86_64
libijs-0.35.so()(64bit) is needed by (installed)
gutenprint-5.2.9-15.el7.x86_64
libijs-0.35.so()(64bit) is needed by (installed)
cups-filters-1.0.35-15.el7.x86_64
```

 尽管您能够让 `rpm` 强制卸载一个具有未解决依赖的软件包（使

用`--nodeps`选项），但不推荐这样做。使用`--nodeps`选项卸载软件包会引起依赖包被卸载的应用程序行为异常或者意外终止。它也可能引起严重的包管理问题或者系统失效。基于这些原因，请注意依赖不满足的警告。

7.2.3 更新软件包

更新和升级类似，只不过更新时只有已安装的软件包会被升级。请以 `root` 用户在 `shell` 命令行提示符下输入以下命令：

```
rpm -Fvh package.rpm
```

`-F`（或者`--freshen`）选项会比较命令行中指定的软件包的版本和系统中已经安装的软件包的版本。当`--freshen`选项处理的是已安装软件包的一个较新的版本时，它将被升级到这个较新的版本。然而，`--freshen`选项不会安装尚未在系统中安装过的软件包。这有别于普通的升级，因为升级将会安装所有指定的软件包，不管是否已经安装过一个旧版本的软件包。

更新可用于单个软件包或者软件包分组。例如，当您下载了很多不同的软件包，而您只想升级那些已经在系统中安装过的包时，可以使用更新功能。在这种情况下，使用`*.rpm`来执行以下命令：

```
~]# rpm -Fvh *.rpm
```

RPM 将自动升级那些已经在系统上安装过的软件包。

7.2.4 查询软件包

RPM 数据库保存了关于系统中所有已安装的软件包的信息。它保存在 `/var/lib/rpm/` 目录下，具有多种用途，包括查询安装了哪些包、每个包的版本是什么、以及计算安装软件包后对文件的修改。要查询这个数据库，请使用`-q`（或者`--query`）选项来执行 `rpm` 命令：

```
rpm -q package_name
```

该命令将显示已安装软件包 `package_name` 的名称、版本号以及发行号。例如：

```
~]$ rpm -q tree
```

```
tree-1.6.0-10.el7.x86_64
```

请参见 rpm(8)用户手册页面的【Package Selection Options】副标题，了解可以用来进一步改善或限制您的查询的选项列表。请使用【Package Query Options】副标题下列出的选项，来指定要显示关于查询到的软件包的哪些信息。

7.2.5 校验软件包

校验一个软件包就是对比该包安装到系统中的文件的信息和原始包中的相同信息。在其它参数中，校验过程对比每个文件的文件大小、MD5 和、权限、类型、所有者以及所属组。

请使用带-V（或--verify）选项的 rpm 命令来校验软件包。例如：

```
~]$ rpm -V tree
```

请参见 rpm(8)用户手册页面的【Package Selection Options】副标题，了解可以用来进一步改善或限制您的查询的选项列表。请使用【Verify Options】副标题下列出的选项，来指定要校验软件包的哪些特征。

如果一切校验都正常，则没有任何输出。如果有任何不符，都会被显示出来。输出结果类似以下内容：

```
~]# rpm -V abrt
S.5....T. c /etc/abrt/abrt.conf
.M..... /var/spool/abrt-upload
```

输出结果的格式是一个由 9 个字符、1 个可选的属性标记以及被处理的文件名组成的字符串。

前 9 个字符是对文件进行测试的结果。每一项测试就是把文件的一个属性值和 RPM 数据库中记录的对应属性值进行对比。一个单一的句点（.）表示该项测试通过了，问号（?）则表示无法执行该项测试。下表列出了用来指示特定差异的符号：

表格 7-1 RPM 校验符号

符号	描述
----	----

符号	描述
S	文件大小不同
M	模式不同（包括权限和文件类型）
5	MD5 和不同
D	设备 major/minor 编号不匹配
L	readLink(2)路径不匹配
U	用户所属关系不同
G	用户组所属关系不同
T	mtime 不同
P	能力不同

如果出现了属性标记，它描述所给出的文件的用途。下表列出了可用的属性标记：

表格 7-2 RPM 校验标记

标记	描述
c	配置文件（configuration file）
d	文档文件（documentation file）
l	许可证文件（license file）
r	自述文件（readme file）

如果您看到了任何输出，请用您最好的判断力来确定，您是否应该删除该软件包，并进行重新安装，或者是通过其他方式来解决该问题。

7.3 查找并校验 RPM 软件包

在使用任何 RPM 软件包之前，您必须知道从哪里可以找到它们，并能够验证是否能信任它们。

7.3.1 查找 RPM 软件包

尽管在互联网上有许多 RPM 仓库，但出于安全和兼容性的原因，您应该考虑只安装中标麒麟高级服务器操作系统官方提供的 RPM 软件包。下面是 RPM 软件包的来源列表：

- 官方的中标麒麟高级服务器操作系统安装光盘
 - 和 Yum 软件包管理器一起提供的官方 RPM 仓库。请参见 2.1 Yum 了解如何使用官方的软件包仓库。
 - EPEL (Extra Packages for Enterprise Linux, 企业级 Linux 额外软件包) 是一个致力于为企业级 Linux 提供高质量扩展软件包的社区。请参见 <http://fedoraproject.org/wiki/EPEL> 了解关于 EPEL RPM 软件包的详情。
 - 非官方的第三方 RPM 软件包仓库。
- ★ 当您考虑在中标麒麟高级服务器操作系统中使用第三方仓库时, 请仔细留意仓库网页上关于软件包兼容性的相关事项, 确定好之后再把该仓库加入到系统中作为软件包来源。可替换的软件包仓库可能会提供同一个软件的不同版本或者不兼容版本, 包括那些在中标麒麟高级服务器操作系统仓库中已经包含了的软件包。

7.3.2 校验软件包签名

RPM 软件包可以使用 GPG (GNU Privacy Guard) 来签名, 这可以帮助您确保下载的软件包是可信的。GPG 是一个安全通信工具。使用 GPG, 您能够验证文档的有效性, 并且对数据进行加解密。

要验证一个软件包没有被破坏或篡改, 可以使用 `rpmkeys` 命令配合 `-K` (或 `--checksig`) 选项来检查它的 GPG 签名:

```
rpmkeys -K package.rpm
```

注意, Yum 软件包管理器在安装和升级时, 会自动执行 GPG 签名的检查。

GPG 还有一系列用于校验的密钥软件包是默认安装的。要导入 RPM 使用的额外密钥, 请参见 7.3.1.2.1 导入 GPG 密钥。

7.3.2.1 导入 GPG 密钥

要校验中标麒麟的软件包, 需要安装一个中标麒麟的 GPG 密钥。默认已经安装了一套基本的密钥。要查看已经安装的密钥列表, 请在 `shell` 命令行提示符下执行以下命令:

```
~]$ rpm -qa gpg-pubkey*
```

要显示一个特定密钥的详细信息，请使用 `rpm -qi` 命令，后面加上上一条命令的输出。例如：

```
~]$ rpm -qi gpg-pubkey-fd431d51-4ae0493b
```

请使用 `rpmkeys` 命令和 `--import` 选项来安装 RPM 使用的一个新的密钥。保存 GPG 密钥的默认位置是 `/etc/pki/rpm-gpg/` 目录。要导入新的密钥，请以 `root` 用户执行以下命令：

```
~]# rpmkeys --import  
/etc/pki/rpm-gpg/RPM-GPG-KEY-neokylin-release
```

7.4 RPM 用法的常用示例

RPM 不管是对于管理您的系统来说，还是对于诊断和修复问题来说，都是很有用的工具。请参见以下的示例来了解其最常用的一些选项。

- 要校验您的整个系统，以查看缺失了哪些文件，请以 `root` 用户执行以下命令：

```
rpm -Va
```

如果某些文件缺失了，或者出现了损坏，请考虑重新安装相应的软件包。

- 要确定一个文件属于哪一个软件包，请输入：

```
rpm -qf file
```

- 要验证软件包拥有一个特定的文件，请以 `root` 用户输入：

```
rpm -Vf file
```

- 要定位一个文档文件属于哪一个软件包，请输入：

```
rpm -qdf file
```

- 要查找一个未安装的软件包文件的信息，请使用以下命令：

```
rpm -qip package.rpm
```

- 要列出一个软件中包含的文件，请使用以下命令：

```
rpm -qlp package.rpm
```

请参见 rpm(8)用户手册页面了解更多选项。