inspur Power Systems

浪潮K-UX操作系统
用户手册

尊敬的浪潮 K-UX 系统用户：

衷心感谢您选用浪潮 K-UX 系统！

本手册介绍了浪潮 K-UX 操作系统的基本功能说明，有助于您了解和使用此操作系统。

请将我方产品的包装物交废品收购站回收利用，以利于污染预防，造福人类。

本手册中的内容如有变动恕不另行通知。

如果您对本手册有疑问或建议，请向浪潮垂询。

浪潮

二零二一年八月

版本说明

文档版本：V1.0

日期：2021 年 8 月

文档说明：第 1 次正式发行。

目录

前言

本手册用于指导 INSPUR K-UX 用户的基本功能使用。

第 1 章 系统启动过程

1.1 K-UX 启动过程简介

在系统启动过程中要涉及多个不同的组件。按下开机按钮后，首先 BIOS/UEFI 做最基本的硬件自检与初始化，然后加载预设/手动选择的磁盘/网络上的引导加载器(例如 GRUB2)，引导加载器进一步从磁盘/网络上加载操作系统内核(例如 Linux)。

对于 Linux 来说，内核将会(可选的)解压一个 initrd(initial RAM disk)镜像(可以用 dracut 类的工具生成)，并执行由"rdinit="内核引导参数指定的 init 程序(例如 systemd)以寻找并挂载根文件系统。

完成根文件系统的挂载之后，内核启动由"init="内核引导参数指定的 init 程序(例如 systemd)以接管系统的控制权。

该 init 程序将会负责检测所有其他的硬件设备、挂载必要的文件系统、启动所有必要的服务等等。

1.2 systemd 进程

当成功挂载了"root="内核引导参数指定的根文件系统之后，内核将启动由"init="内核引导参数指定的 init 程序，从这个时间点开始，即进入了"常规启动流程"：检测硬件设备并加载驱动、挂载必要的文件系统、启动所有必要的服务等等。

对于 systemd 系统来说，上述"init 程序"就是 systemd 进程，而整个"常规启动流程"也以几个特殊的 target 单元(详见)作为节点，被划分为几个阶段性步骤。

在每个阶段性步骤内部，任务是高度并行的，所以无法准确预测其中的单元的顺序，但是不同阶段之间的先后顺序总是固定的。

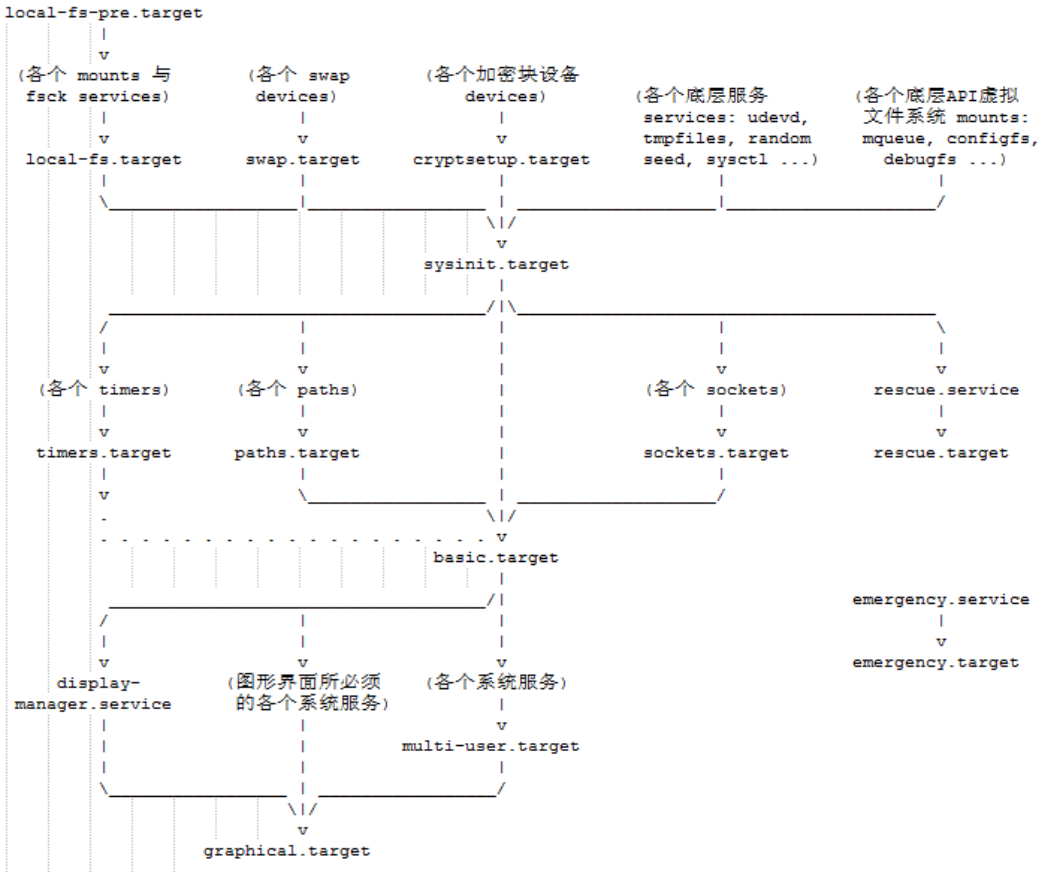
当启动系统时，systemd 将会以 default.target 为启动目标，借助单元之间环环相扣的依赖关系，即可完成"常规启动流程"。

通常，default.target 只是一个指向 graphical.target(图形界面) 或 multi-user.target(文本控制台) 的软连接。

为了强制启动流程的规范性以及提高单元的并行性，预先定义了一些具有特定含义的 target 单元。

下面的图表解释了这些具有特定含义的 target 单元之间的依赖关系以及各自在启动流程中的位置。

图中的箭头表示了单元之间的依赖关系与先后顺序，整个图表按照自上而下的时间顺序执行。



第 2 章 基本开机和登录

2.1 登录和退出系统

默认安装后，系统中的管理员帐户的用户名为 root， 这个帐户对系统的一切都有完全的访问权限。在 Login 提示后键入 root，然后按 Enter 键，如果敲入中有错，可用 Del 或 BackSpace 来纠正错误。按 Enter 键后会出现 Password:提示，输入 root 的口令，按[Enter]键，就应该看到类似下面的信息：

```
[root@inspur ~]#
```

进入如上的交互界面后，就可以执行命令进行维护操作。root 是用来执行系统管理任务，如创建新的用户帐户、关机等等。因为 root 的权限不受限制，如果不慎输入了错误的命令，则可能会导致灾难性的后果。所以在以 root 登录时，必须格外小心。 在完成操作以后，用户需要退出 K-UX。虽然大多数 shell 有 logout 命令，但多数人输入 Ctrl-D 或者键入 exit，这些命令都会使用户退出登录。

2.2 如何关闭计算机

如果当前用户是 root，可以执行 shutdown -h now 关闭计算机。

```
[root@inspur ~]# shutdown -h now
```

如果需要重新启动计算机，可以执行 shutdown -r now 或者 reboot 命令。

```
[root@inspur ~]# shutdown -r now
```

如果是 root 用户可以执行 poweroff 命令关闭计算机。

```
[root@inspur ~]# poweroff
```

shutdown 命令的更多功能和另外一些相关的命令，比如 init 0 ， init 6 ， halt 等等，请查看帮助文件。(man shutdown)

第 3 章 管理帐户和口令

3.1 账户管理概述

3.1.1 账户实质

K-UX 操作系统是多用户的操作系统，它允许多个用户同时登录到系统上，使用系统资源。当多个用户能同时使用系统时，为了使所有用户的工作都能顺利进行，保护每个用户的文件和进程，也为了系统自身的安全和稳定，必须建立起一种秩序，使每个用户的权限都能得到规范。为此，首先就需要区分不同的用户，这就产生了用户账户。账户实质上就是一个用户在系统上的标识，系统依据账户来区分每个用户的文件、进程、任务，给每个用户提供特定的工作环境，使每个用户的工作都能独立不受干扰地进行。

3.1.2 用户和组

广义上讲，K-UX 的账户包括用户账户和组账户两种。

K-UX 系统下的用户账户(简称用户)有两种，普通用户账户和超级用户账户(或管理员账户)。普通用户在系统上的任务是进行普通工作，管理员在系统上的任务是对普通用户和整个系统进行管理。管理员账户对系统具有绝对的控制权，能够对系统进行一切操作，如操作不当很容易对系统造成损坏。因此即使系统只有一个用户使用，也应该在管理员账户之外建立一个普通用户账户，在用户进行普通工作的时候以普通用户账户登录系统。

除了用户账户之外，在 K-UX 下还存在组账户(简称组)。组是用户的集合。在 K-UX 中组有两种类型:私有组和标准组。当创建一个新用户时，若没有指定他所属于的组，K-UX 就建立一个和该用户同名的私有组。此私有组中只包含这个用户自己。标准组可以容纳多个用户，若使用标准组，在创建一个新的用户时就应该指定他所属于的组。

从另一方面讲，同一个用户可以同属于多个组，例如某单位有领导组和技术组等，Tom 是该单位的技术主管，所以他既应该属于领导组又应该属于技术组。

当一个用户属于多个组时，其登录后所属的组称为主组，其他的组称为附加组。

3.1.3 账户文件管理

K-UX 下的账户系统文件主要有 /etc/passwd、/etc/shadow、/etc/group 和 /etc/gshadow 四个文件。

1. 用户文件

/etc/passwd 文件中每行定义一个用户账户，一行中又划分为多个字段定义用户账户的不同属性，各字段间用“:”分隔。例如:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
```

下表描述了这些字段的意义：

字段	说明
用户名	这是用户登录系统时使用的用户名，它在系统中是唯一的。
口令	此字段存放加密的口令。在此文件中的口令是 X，这表示用户的口令是被/etc/shadow 文件保护的，所有加密的口令以及和口令有关的设置都保存在/etc/shadow 中。
用户标识号	是一个整数，系统内部用它来标识用户。每个用户的 UID 都是唯一的。root 用户的 UID 是 0，从 1 到 999 是系统的标准账户。普通用户的 UID 从 1000 开始

组标识号	是一个整数，系统内部用它来标识用户所属的组。每个用户账户在建立好后都会有一个主组。主组相同的账户其 GID 相同。
注释性描述	例如存放用户全名等信息。
自家目录	用户登录系统后所进入的目录

2. 用户密码文件

/etc/passwd 文件对任何用户均可读，为了增加系统的安全性，用户的口令通常用 shadow passwords 保护。/etc/shadow 只对 root 用户可读。在安装系统时，会询问用户是否启用 shadow passwords 功能。在安装好系统后也可以用 pwconv 命令和 pwunconv 命令来启动或取消 shadow passwords 的保护。

UX 默认使用 shadow passwords 保护。经过 shadow passwords 保护的账户口令和相关设置信息保存在/etc/shadow 文件里。

shadow 文件的内容形式如下:

```
root:$1$BRcizYwG$XQJzEPcaUp0eKXgMSFQZB.:21914:0:99999:7:::
bin:*:16426:0:80:7:::
daemon:*:16426:0:80:7:::
adm:*:16426:0:80:7:::
lp:*:16426:0:80:7:::
sync:*:16426:0:80:7:::
shutdown:*:16426:0:80:7:::
...
```

其中各字段的意义如表所示。

字段	说明
用户名	用户的账户名
口令	用户的口令，是加密过的
最后一次修改的时间	从 1970 年 1 月 1 日起，到用户最后一次更改口令的天数

最小时间间隙	从 1970 年 1 月 1 日起，到用户可以更改口令的天数
最大时间间隙	从 1970 年 1 月 1 日起，到用户必须更改口令的天数
警告时间	在用户口令过期之前多少天提醒用户更新
不活动时间	在用户口令过期之后到禁用账户的天数
失效时间	从 1970 年 1 月 1 日起，到账户被禁用的天数
标志	保留位

3. 用户组文件

将用户分组是 K-UX 中对用户进行管理及控制访问权限的一种手段。每个用户都属于某一个组；个组中可以有多个用户，一个用户也可以属于不同的组。当一个用户同时是多个组的成员时，在/etc/passwd 文件中记录的是用户所属的主组，也就是登录时所属的默认组，而其他组称为附加组。用户要访问附加组的文件时，必须首先使用 newgrp 命令使自己成为所要访问的组的成员。组的所有属性都存放在/etc/group 文件中。/etc/group 文件对任何用户均可读。下面是一个/etc/group 文件的例子:

```
root:x:0:root
bin:x:1:root，bin，daemon
daemon:x:2:root，bin，daemon
sys:x:3:root，bin，adm
adm:x:4:root，adm，daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon，lp
...
```

和/etc/passwd 文件类似，其中每一行记录了一个组的信息。每行包括四个字段，不同字段之间用冒号隔开。其中各字段的内容说明见下表。

字段	说明
组名	该组的名称
组口令	用户组口令，由于安全性原因，已不使用该字段保存口令，用“x”占位
GID	组的识别号，和 UID 类似，每个组都有自己独有的识别号，不同组的 GID 不会相同
组成员	属于这个组的成员

4. 用户组文件

/etc/gshadow 文件用于定义用户组口令、组管理员等信息，该文件只有 root 用户可以读取。下面是一个/etc/gshadow 文件的例子:

```
root:::root
bin:::root，bin，daemon
daemon:::root，bin，daemon
sys:::root，bin，adm
adm:::root，adm，daemon
tty:::
disk:::root
lp:::daemon，lp
mem:::
```

和/etc/group 文件类似，其中每一行记录了一个组的信息。每行包括四个字段，不同字段之间用冒号隔开。其中各字段的内容说明见下表。

字段	说明
组名	用户组名称，该字段与 group 文件中的组名称对应
组口令	用户组口令，该字段用于保存已加密的口令
组的管理员账户	组的管理员账户，管理员有权对该组添加删除账户
组成员	属于该组的用户成员列表，列表中多个用户间用“，”分隔

3.2 新建用户和组

除了 root 用户以外，我们可以创建其他普通用户来维护不同的业务。但是只有 root 用户拥有创建用户的权力。

3.2.1 新建用户

useradd 用于添加用户账户或设置添加用户使用的默认信息。格式如下:

```
格式 1:useradd [options] LOGIN
格式 2:useradd -D
格式 3:useradd -D [options]
```

说明：

格式 1:用于添加用户账户，LOGIN 为用户登录账户

格式 2:用于显示添加用户使用的默认信息

格式 3:用于设置添加用户使用的默认信息

添加用户选项:

选项	说明
-u UID	指定新用户的 UID，默认为使用当前最大的 UID 加 1
-g GROUP	指定新用户的主组
-G GROUP1[，GROUP2，...[，GROUPN]]]	指定新用户的附加组
-d HOME_DIR	指定新用户的登录目录
-s SHELL	指定新用户使用的 Shell，默认为 bash
-c COMMENT	说明用户的附加信息，如全名等
-e EXPIRE_DATE	指定用户的登录失效时间，格式为

	YYYY-MM-DD
-f INACTIVE	指定在密码过期后多少天即关闭该账户， 默认值为 -1，即不做限制
-k SKEL_DIR	指 定 skel 目 录 ， 默 认 存 放 在 /etc/skel/
-m	创建新用户的自家目录，默认值
-M	不创建新用户的自家目录

改变用户默认值选项:

选项	说明
-b BASE_DIR	定义使用者所属目录的上级目录。用户名称会附加在 BASE_DIR 后面用来建立新使用者的目录。 当然使用-d 后则此选项无效。
-e EXPIRE_DATE	定义使用者账户失效日期。
-f INACTIVE	定义在密码过期后多少天即关闭该账户。
-g GROUP	定义新账户起始组名或 GID。组名必须为现已存在的组名。GID 也必须为现已存在的 GID。
-s SHELL	定义用户默认使用的 shell。

在配置文件 /etc/login.defs 和 /etc/default/useradd 中存放了用户默认数据。

假设我们要创建一个名字为 test 的帐户，命令如下：

```
[root@inspur ~]# useradd test
```

3.2.2 新建组

groupadd 命令用于将新组加入系统。格式如下：

```
groupadd [-g gid] [-o] [-r] [-f] groupname
```

说明：

-g gid：指定组 ID 号。

-o：允许组 ID 号，不必唯一。

-r：加入组 ID 号，低于 499 系统账户。

-f：如果加入的组已经存在，仅以成功状态退出。

例如，建立一个新组，并设置组 ID 加入系统：

```
[root@inspur ~]# groupadd -g 344 test
```

此时在/etc/group 文件中产生一个组 ID（GID）是 344 的项目。

3.3 设置用户密码

创建了新帐户之后，我们还必须为新帐户设置口令，否则用户仍然无法使用这个帐户登录。passwd 命令可以用来为新创建的用户分配口令、修改已存在的用户的口令和修改您登录的用户的口令。

设置用户口令的命令是 passwd，命令格式是:

```
passwd [选项] [用户登录名]
```

参数说明：

选项	说明
-d	删除口令。仅管理员才能使用。
-k	设置只有在口令过期失效后方能更新。
-l	锁定用户账户
-u	解除已锁定账户。
-S	列出口令的状态信息。

1. 在输入口令时，屏幕上不会回显。口令的选取至少用六个字符，最好大小写字母和数字及特殊字符搭配使用，尽量不要用英文单词作口令。

2. 只有管理员账户(root)可以更改其他用户的口令，普通用户只能更改自己的口令，且在更改口令之前，系统会要求用户输入现在的口令。

3. 超级用户也可以使用不带任何参数的 passwd 命令修改自己的口令。

```
[root@inspur ~]# passwd test
Changing password for user test.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

3.4 口令时效

目前已有更强大的硬件大大地缩短了利用自动运行的程序来猜测口令的时间。口令时效是系统管理员用来防止机构内不良口令的一种技术。防止口令被攻击的方法就是要经常地改变口令。为安全起见，要求用户定期改变他们的口令是明智之举。口令时效意味着过了一段预先设定的时间后，用户会被提示创建一个新口令。它所根据的理论是，如果用户被强制定期改变口令，某个破译的口令对入侵者来说就只有有限的利用机会。这种用来强制用户在一段时间之后更改口令的机制称为口令时效。

3.4.1 设置新添用户的口令时效

通过编辑 /etc/login.defs，可以指定几个参数，来设置口令时效的默认设定:

```
# Password aging controls:
#
#    PASS_MAX_DAYS    Maximum number of days a password may be used.
#    PASS_MIN_DAYS    Minimum number of days allowed between password
changes.
#    PASS_MIN_LEN    Minimum acceptable password length.
#    PASS_WARN_AGE    Number of days warning given before a password
expires.
#
PASS_MAX_DAYS    99999
PASS_MIN_DAYS    0
PASS_MIN_LEN    5
```

```
PASS_WARN_AGE     7
```

PASS_MAX_DAYS:设定在多少天后要求用户修改口令。默认口令时效的天数为 99999，即关闭了口令时效。更明智的设定一般是 60 天(每 2 个月)强制更改一次口令。

PASS_MIN_DAYS:设定在本次口令修改后，下次允许更改口令之前所需的最少天数。

PASS_MIN_LEN :设定口令的最小字符数。

PASS_WARN_AGE:设定在口令失效前多少天开始通知用户更改口令(一般在用户刚刚登陆系统时就会收到警告通知)。

也可以编辑 /etc/default/useradd 文件中 INACTIVE 和 EXPIRE 的设置。

```
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

INACTIVE:指明在口令失效后多久时间内，如果口令没有进行更改，则将账户更改为失效状态。默认值为-1，即关闭了口令时效。更明智的设定一般是 60 天(每 2 个月)强制更改一次口令。

EXPIRE:为设置所有新用户设定一个口令失效的明确时间(具体格式为“YYYY-MM-DD”)。

3.4.2 设置已存在用户的口令时效

对系统已存在用户设置口令时效是通过 chage 命令来管理的。chage 命令的格式是:

```
chage [选项] [用户登录名]
```

参数说明：

选项	说明
-m days	指定用户必须改变口令所间隔的最少天数。如果值为 0，口令就不会过期。(PASS_MIN_DAYS)
-M days	指定口令有效的最多天数。当该选项指定的天数加上-d 选项指定的天数小于当前的日期，用户在使用该账户前就必须改变口令。(PASS_MAX_DAYS)
-d days	指定自从 1970 年 1 月 1 日起，口令被改变的天数。
-I days	指定口令过期后，账户被锁前不活跃的天数。如果值为 0，账户在口令过期后就不会被锁。
-E date	指定账户被锁的日期，日期格式为 YYYY-MM-DD。若不用日期，也可以使用自 1970 年 1 月 1 日后经过的天数。
-W days	指定口令过期前要警告用户的天数。(PASS_WARN_AGE)
-l	列出指定用户当前的口令时效信息，以确定账户何时过期。

3.5 用户切换和用户状态命令

系统管理员应该养成良好的习惯:以一个普通用户登录系统进行不同操作，当需要超级用户身份进行系统管理时再切换超级用户执行系统管理命令。有如下两种方式：su 和 sudo。

3.5.1 su

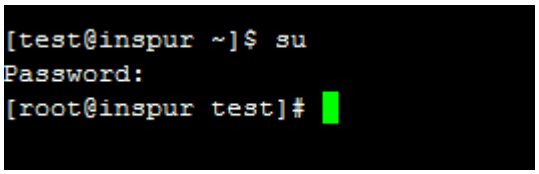
出于安全方面的考虑，管理员在大多数时间是用普通用户的帐户登录远程系统， 如果需要执行只有 root 才能执行的命令，可以使用 su 命令切换到 root，这种方式应注意：

a.直接切换为超级用户。

b.普通用户要切换为超级用户必须知道超级用户的口令。

c.适用于系统中只有单个系统管理员的情况。

输入 su 命令后，会要求用户输入 root 的口令，正确地输入了口令后，用户的当前登录就会“变成”root 或其他用户的登录，提示符由“$”变为“#”。



3.5.2 sudo

sudo 允许系统管理员(root)为几个用户或组委派权利，使之能运行部分或全部由 root (或另一个)用户执行的命令。sudo 设计者的宗旨是:给用户尽可能少的权限但仍允许完成他们的工作。sudo 是设置了 SUID 位的执行文件，这种方式应注意：

a.直接使用 sudo 命令前缀执行系统管理命令。

b.执行系统管理命令时无需知道超级用户的口令，使用普通用户自己的口令即可。

c.由于执行系统管理命令时无需知晓超级用户口令，所以适用于系统中有多个系统管理员的情况，因为这样不会泄露超级用户口令。当然，系统只有单个系统管理员时也可以使用。

使用 sudo 进行系统管理具有以下特点:

a. sudo 能够限制指定用户在指定主机上运行某些命令。

b. sudo 可以提供日志，忠实地记录每个用户使用 sudo 做了些什么，并且能将日志传到中心主机或者日志服务器。

c. sudo 为系统管理员提供配置文件，允许系统管理员集中地管理用户的使用权限和使用的主机。它默认的存放位置是 /etc/sudoers。

d. sudo 使用时间戳文件来完成类似“检票”的系统。当用户执行 sudo 并且输入密码后，用户获得了一张默认存活期为 5 分钟的“入场券”(默认值可以在编译的时候改变)。超时以后，用户必须重新输入密码。

用户执行 sudo 的执行过程:

1. 在 /var/run/sudo/$HOME 目录中查找时间戳文件

I. 若时间戳已过期，提示用户输入自己的口令

a. 若口令输入错误则退出 sudo 的执行

b. 若口令输入正确则继续 sudo 的执行过程

II. 若文件存在继续 sudo 的执行过程

2. 读取配置文件 /etc/sudoers，判断用户是否具有执行此 sudo 命令的权限

I. 若有权限执行则执行 sudo 后面的命令

II. 若无权执行则退出 sudo 的执行

3.5.3 sudo 命令格式

sudo 命令的格式为:

```
sudo -V | -h | -k | -l | -v
sudo [-Hb] [-u username|#uid] { -i | -s | <command> }
```

其中:

-V：显示版本信息，并退出。

-h：显示帮助信息。

-l：显示当前用户(执行 sudo 的使用者)的权限，只有在 /etc/sudoers 里的用户才能使用该选项。

-v：延长密码有效期限 5 分钟。

-k：将会强迫使用者在下一次执行 sudo 时问密码(不论有没有超过 5 分钟)。

-H：将环境变数中的 $HOME 指定为要变更身份的使用者家目录(如不加 -u 参数就是 /root) 。

-b：在后台执行指令。

-u username|#uid：以指定的用户作为新的身份。忽略此参数表示以 root 的身份执行指令。

-i：模拟一个新用户身份的初始 Shell。

-s：执行环境变量 $SHELL 所指定的 shell ，或是 /etc/passwd 里所指定的 shell。

command：以新用户身份要执行的命令。

3.6 修改用户属性

使用 usermod 可以修改用户属性。如若修改 test 用户的 home 目录为/home/ceshi，命令如下：

```
[root@inspur ~]# usermod -d /home/ceshi/ test
```

usermod 命令还有很多其他参数，列表如下，也可以查看 man (man usermod)

参数	说明
-u <uid>	修改用户 uid
-g <group>	修改用户组
-d <home>	修改 home 目录
-s <shell>	修改交互 SHELL

userdel 用于删除系统已存在的组账户。格式如下:

```
userdel [-r] LOGIN
```

其中:

LOGIN:为要删除的用户账户名

-r:同时删除用户的家目录和 mail 的假脱机文件

3.7 用户状态命令

常用的用户状态命令包括:whoami、id、groups、newgrp 等。

a. whoami:用于显示当前用户的名称

b. groups:用于显示指定用户所属的组

c. id:用户显示用户身份

e. newgrp:用户转换用户的当前组到指定附加组，用户必须属于该组才可以进行。

3.8 文件系统

3.8.1 文件系统概述

在 K-UX 系统中，所有的程序、库、系统文件和用户文件都存放在文件系统之中，系统和用户所创建和保存的数据也都在文件系统当中，什么是文件系统呢？文件系统是操作系统用于明确磁盘分区上的文件的方法和数据结构，即文件在磁盘上的组织方法。换句话说，文件系统规定了如何在存储设备上存储数据以及如何访问存储在设备上的数据。一个文件系统在逻辑上是独立的实体，他能单独地被操作系统管理和使用。

K-UX 的内核采用了称之为虚拟文件系统(Virtual File System ，VFS)的技术，因此 K-UX 可以支持多种不同的文件系统类型。每一种类型的文件系统都提供一个公共的软件接口 VFS。K-UX 文件系统的所有细节由软件进行转换，因而从 K-UX 的内核以及在 K-UX 中运行的程序来看，所有类型的文件系统都没有差别，K-UX 的 VFS 允许用户同时不受干扰地安装和使用多种不同类型的文件系统。所以 K-UX 核心的其它部分及系统中运行的程序将看到统一的文件系统。

K-UX 的虚拟文件系统允许用户同时能透明地安装许多不同的文件系统。虚拟文件系统是为 K-UX 用户提供快速且高效的文件访问服务而设计的。随着 K-UX 的不断发展，它所支持的文件格式系统也在迅速扩充。特别是 Linux 2.4 内核正式推出后，出现了大量新的文件系统，其中包括日志文件系统 ext3、ReiserFS、XFS、JFS 和其它文件系统。K-UX 系统核心可以支持十多种文件系统类型:JFS、 ReiserFS、ext、ext2、ext3、ISO 9660、XFS、Minx 、MSDOS、UMSDOS、VFAT、NTFS、HPFS、NFS、SMB、SysV 等。

3.8.2 文件系统布局

文件系统是 K-UX 下的所有文件和目录的集合，这些文件和目录结构是以一个树状的结构来组织的，这个树状结构构成了文件系统。也就是说当用户使用 K-UX 文件系统时所面对的就是这个树状结构，在这个结构中包含大量的文件和目录。使用 K-UX ，用户可以设置目录和文件的权限，以便允许或拒绝其他人对其

进行访问。用户可以浏览整个系统，可以进入任何一个已授权进入的目录，访问那里的文件。

K-UX 继承了 Unix 操作系统结构清晰的特点，在 K-UX 下的文件结构非常有条理。现在就把 K-UX 下的目录结构简单介绍一下：

基本目录和文件操作命令

目录	说明
/bin	K-UX 常用的命令所在的目录
/etc	这个目录下存放了系统管理时要用到的各种配置文件和子目录。
/lib	这个目录是用来存放系统动态连接共享库的。
/sbin	这个目录是用来存放系统管理员的系统管理程序。
/tmp	用来存放不同程序执行时产生的临时文件
/boot	在这个目录下存放的都是系统启动时要用到的程序。我们在使用 grub 引导 K-UX 的时候，会用到这里的一些信息。
/home	用户的 home 目录
/mnt	用户放置挂载项的目录。
/media	另一个用户放置挂载项的目录。
/root	root 用户的 home 目录。
/usr	通常用于存放用户的应用程序和文件。
/dev	因为在这个目录中包含了所有 K-UX 系统中使用的外部设备。
/proc	这是一个显示系统信息的伪文件系统，里面的文件可以显示当前内存中的系统信息。

3.8.3 重要文件系统介绍

1. /etc 文件系统

目录	说明
/etc/X11	X Windows 的设置目录
/etc/alternatives	存储具有相同功能程序的二/多选一链接的目录
/etc/init.d	守护进程启动脚本目录
/etc/logrotate.d	日志滚动脚本的配置目录
/etc/lvm	LVM2 的配置目录
/etc/opt	/opt 应用程序的配置目录
/etc/pam.d	PAM 配置目录
/etc/rc.d	启动、或改变运行级时运行的 scripts 目录
/etc/skel	普通用户初始环境目录
/etc/ssh	ssh 的配置目录
/etc/yum	yum 的配置目录
/etc/yum.repos.d	yum 源的配置目录

2. /usr 文件系统

目录	说明
/usr/bin	存放了许多用户命令
/usr/games	存放游戏和教育类软件
/usr/include	存放 K-UX 下开发和编译应用程序所需要的头文件
/usr/lib	放一些常用的动态链接共享库和静态档案库
/usr/local	供给本地用户的/usr 目录，在这里安装本地的应用软件
/usr/sbin	存放 root 超级用户使用的管理程序

/usr/share	系统共用的东西存放地，如:手册、文档、字体等
/usr/src	是内核源码存放的目录

3. /var 文件系统

目录	说明
/var/cache	应用程序缓存的数据目录
/var/lib	存储系统或各个应用程序运行时的状态信息数据
/var/lock	存储程序运行时的锁定文件的目录。许多程序遵循在/var/lock 中产生一个锁定文件的约定，以支持他们正在使用某个特定的设备或文件。其他程序注意到这个锁定文件，将不试图使用这个设备或文件
/var/log	系统日志存放，分析日志要看这个目录的东西
/var/mail	用户 mailbox 文件存储目录
/var/opt	存储 /opt 目录下应用程序的经常变化的数据
/var/run	存储到下次引导前有效的关于系统的信息文件
/var/spool	打印机、邮件、代理服务器等假脱机目录
/var/tmp	存放临时文件

显示目录内容的命令是 ls，显示当前目录的命令是 pwd，更改当前目录的命令是 cd。通过这些命令就可以切换到各个不同的目录。ls 也可以用于显示文件内容，因为在 K-UX 中所有的东西(包括设备，文件，目录，SOCKET 接口等等)都以文件形式出现。目录只是一个特殊的文件，对文件的操作是最基本的命令。常用的命令如下： ll，cp，mv，cat，chmod，chown 等等命令。

命令	说明
ll <filename>	显示文件的大小，权限和属主。
cp <filename1> <filename2>	拷贝< filename1>到 <filename2>
mv <filename1> <filename2>	把< filename1>改名为< filename2>

chown <options> <filename>	修改文件权限。
chmod <username> <filename>	修改文件的属主。
cat <filename>	显示文件内容。
less <filename>	分页的形式显示文件内容。
stat <filename>	显示文件的节点信息。

3.9 使用 vi 编辑文件

vi 是 Unix 世界里极为普遍的文书编辑器。在系统维护过程中，需要经常使用编辑软件修改文本文件。熟悉 vi 能够带来很大的方便。 vi 有两种主要的工作模式——输入模式和命令模式。用户可以通过 Esc 键在这两种模式之间进行切换。输入模式是用来输入文字资料的，而命令模式则是用来下达一些编排文件、存档、以及离开 vi 等等的操作指令。当执行 vi 后，会先进入命令模式，此时输入的任何字符都被视为命令。要进入 vi 可以直接在系统提示符下键入 vi <filename> ， vi 就可以自动载入所要编辑的文件或是打开一个新的文件。为了进入输入状态可以使用下表的命令：

键击	说明
a	从游标所在位置后面添加新文本
A	从当前行的末尾添加文本
i	从游标所在位置前添加新文本
I	从当前行的行首插入新文本
o	在当前行的下面打开一行以添加文本
O	在当前行的上面打开一行以添加文本

配合键盘上的功能键，如：方向键、Insert 、Delete 等等，就可以利用 vi 来处理文字资料了。对文件进行更正与修改就可能要删除文本。使用 vi，用户可以删除一个字符、一个字几个字或一整行。下表描述了这些删除命令：

键击	说明
x	删除游标所在字符
dd	删除游标所在的一行
r	修改游标所在字符，r 后接着要修正的字符
R	进入取代状态，新增文本会覆盖原先文本，直到按[ESC]回到命令模式下为止

在编辑完文件之后，用户可以有几种退出 vi 的方法。要退出 vi 编辑环境，必须在命令模式中。要改变到命令模式，按 Esc 键。下表列出可用于退出 vi 的命令。 退出 vi 的方法 ：

命令	说明
:q	对缓冲区没有做任何修改后退出，或在缓冲区被改变并保存到文本中之后退出
:q!	退出，并且放弃自缓冲区最后一次保存到文件以后的所有对缓冲区的改变
:wq、:x 或 ZZ	把缓冲区写入工作文件，然后退出

除了我们上面对 vi 的简要介绍，用户还应该使用 vi 的 man page 来获得有关 vi 编辑器使用的详细知识。

第 4 章 系统监控

4.1 系统监视概述

为了更好地维护系统，管理员经常要收集一些系统信息，诸如进程、内存、文件系统、硬件等使用信息。然后通过这些信息对系统的正常与否做出判断，并通过这些信息对系统故障做出正确判断。

下面再列出一些系统监视的常用工具，这些工具涉及的软件包也一并列出，若您的系统中没有这些工具可以使用 dnf install 命令进行安装。

名称	说明
coreutils	系统核心工具包
/bin/df	报告系统的磁盘空间用量。
/bin/uname	显示系统信息。
procps	系统进程工具包
/bin/ps	显示系统进程。
/usr/bin/pgrep	过滤显示系统进程
/usr/bin/free	显示系统内存的使用。
/usr/bin/vmstat	报告虚拟内存的统计信息。
/usr/bin/tload	在终端上显示系统平均负载。
/usr/bin/uptime	显示系统的运行时间。
/usr/bin/top	动态显示系统进程任务。
/usr/bin/slabtop	动态显示内核 slab 缓存信息
/usr/bin/watch	以全屏幕方式周期性地执行指定的命令。
lsof	显示进程打开文件的工具包
/usr/sbin/lsof	查看正在运行中的进程打开了哪些文件、目录和套接

	字。
psacct	用户与进程的统计工具包
/usr/bin/ac	登录帐户的简要信息。
/usr/bin/lastcomm	显示已执行过的命令。
/usr/sbin/accton	打开或关闭进程帐户记录功能。
/usr/sbin/dump-acct	输出 pacct 文件的内容。
/usr/sbin/dump-utmp	输出 utmp 文件的内容。
/usr/sbin/sa	进程帐户记录信息的摘要。
sysstat	系统状态工具包
/usr/bin/iostat	用于输出 CPU、I/O 系统和磁盘分区的统计信息。可以用来分析磁盘 I/O，带宽等信息。
/usr/bin/mpstat	用于输出 CPU 的各种统计信息。可以用来分析程序运行时在内核态和用户态的工作情况。
/usr/bin/sar	用于定时搜集系统的各种状态信息，然后对系统各个时间点的状态进行监控。
/usr/bin/sadf	显示被 sar 通过多种格式收集的二进制数据。
pciutils	系统 PCI 设备的工具包
/sbin/lspci	显示 PCI 设备。
/sbin/setpci	配置 PCI 设备。
/sbin/update-pciids	下载新版本的 PCI ID 列表。
usbutils	系统 USB 设备的工具包
/sbin/lsusb	显示 USB 设备。

4.2 收集基本的系统信息

显示该版本的内核信息:

```
[root@inspur ~]# uname -srvmo
Linux 4.18.0-240.10.1.kux.ppc64le #1 SMP Thu Sep 10 15:59:54 CST 2020 ppc64le GNU/Linux
```

显示系统进程的运行时间和负载:

```
[root@inspur ~]# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.2 188608 20864 ?        Ss   05:20   0:02 /usr/lib/systemd/systemd --
root           2  0.0  0.0      0     0 ?        S    05:20   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   05:20   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   05:20   0:00 [rcu_par_gp]
root           6  0.0  0.0      0     0 ?        I<   05:20   0:00 [kworker/0:0H-kblockd]
root           8  0.0  0.0      0     0 ?        I<   05:20   0:00 [mm_percpu_wq]
root           9  0.0  0.0      0     0 ?        S    05:20   0:00 [ksoftirqd/0]
root          10  0.0  0.0      0     0 ?        I    05:20   0:00 [rcu_sched]
root          11  0.0  0.0      0     0 ?        S    05:20   0:00 [migration/0]
root          12  0.0  0.0      0     0 ?        S    05:20   0:00 [watchdog/0]
root          13  0.0  0.0      0     0 ?        S    05:20   0:00 [cpuhp/0]
root          14  0.0  0.0      0     0 ?        S    05:20   0:00 [cpuhp/1]
root          15  0.0  0.0      0     0 ?        S    05:20   0:00 [watchdog/1]
root          16  0.0  0.0      0     0 ?        S    05:20   0:00 [migration/1]
root          17  0.0  0.0      0     0 ?        S    05:20   0:00 [ksoftirqd/1]
root          19  0.0  0.0      0     0 ?        I<   05:20   0:00 [kworker/1:0H-kblockd]
root          20  0.0  0.0      0     0 ?        S    05:20   0:00 [cpuhp/2]
root          21  0.0  0.0      0     0 ?        S    05:20   0:00 [watchdog/2]
root          22  0.0  0.0      0     0 ?        S    05:20   0:00 [migration/2]
root          23  0.0  0.0      0     0 ?        S    05:20   0:00 [ksoftirqd/2]
root          24  0.0  0.0      0     0 ?        I    05:20   0:00 [kworker/2:0-events]
root          25  0.0  0.0      0     0 ?        I<   05:20   0:00 [kworker/2:0H-kblockd]
```

显示系统的物理内存和交换区的使用:

```
[root@inspur ~]#  free
              total        used        free      shared  buff/cache   available
Mem:        7810496     2330752     4445504       35264     1034240     4898496
Swap:       4194240           0     4194240
```

显示系统的磁盘空间用量:

```
[root@inspur ~]# df -h
Filesystem              Size  Used Avail Use% Mounted on
devtmpfs                3.7G     0  3.7G   0% /dev
tmpfs                   3.8G     0  3.8G   0% /dev/shm
tmpfs                   3.8G   29M  3.7G   1% /run
tmpfs                   3.8G     0  3.8G   0% /sys/fs/cgroup
/dev/mapper/ik-root      35G  8.3G   27G  24% /
/dev/sda2               976M  238M  672M  27% /boot
tmpfs                   763M  1.6M  762M   1% /run/user/42
tmpfs                   763M  4.0M  759M   1% /run/user/1000
tmpfs                   763M   64K  763M   1% /run/user/0
```

显示的磁盘分区:

```
[root@inspur ~]# fdisk -l
Disk /dev/sda: 40 GiB, 42949672960 bytes, 83886080 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x17963e72

Device     Boot   Start      End  Sectors Size Id Type
/dev/sda1  *       2048    10239     8192   4M 41 PPC PReP Boot
/dev/sda2         10240  2107391  2097152   1G 83 Linux
/dev/sda3       2107392 83886079 81778688  39G 8e Linux LVM


Disk /dev/sdb: 4 GiB, 4294967296 bytes, 8388608 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x5d01f0c7

Device     Boot Start     End Sectors Size Id Type
/dev/sdb1        2048 8388607 8386560   4G fd Linux raid autodetect


Disk /dev/sdc: 4 GiB, 4294967296 bytes, 8388608 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa05552a5

Device     Boot Start     End Sectors Size Id Type
/dev/sdc1        2048 8388607 8386560   4G fd Linux raid autodetect
```

4.3 top 命令

top 命令显示了当前正运行的进程以及它们的重要信息，包括它们的内存和 CPU 用量。该列表既是真实时间的也是互动的。以下提供了一个 top 的输出示例:

[root@inspur ~]# top -s

```
[root@inspur ~]# top -s
top - 06:50:25 up  1:29,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 232 total,   1 running, 231 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.3 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7627.4 total,   4335.4 free,   2279.9 used,   1012.1 buff/cache
MiB Swap:   4095.9 total,   4095.9 free,      0.0 used.   4779.8 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 root      20   0  188608  20864  11072 S   0.0   0.3   0:02.34 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-kblockd
      8 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
      9 root      20   0       0      0      0 S   0.0   0.0   0:00.02 ksoftirqd/0
     10 root      20   0       0      0      0 I   0.0   0.0   0:00.61 rcu_sched
     11 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/0
     12 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 watchdog/0
     13 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     15 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 watchdog/1
     16 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/1
     17 root      20   0       0      0      0 S   0.0   0.0   0:00.02 ksoftirqd/1
     19 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-kblockd
     20 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/2
     21 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 watchdog/2
     22 root      rt   0       0      0      0 S   0.0   0.0   0:00.00 migration/2
     23 root      20   0       0      0      0 S   0.0   0.0   0:00.01 ksoftirqd/2
     24 root      20   0       0      0      0 I   0.0   0.0   0:00.11 kworker/2:0-mm_percpu_wq
     25 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/2:0H-kblockd
```

top 可使用的互动命令包括:

h：显示帮助屏幕

Space：立即刷新显示

k：杀死指定的进程

n：改变要显示的进程数量

u：按用户排序

m：按内存用量排序

p：按 CPU 用量排序

q：退出

4.4 watch 命令

系统还提供了一个 watch 命令。它以固定的时间周期来执行所指定的指令。预设的执行间隔是 2 秒，当然这是可以用参数来调整的。 watch 命令以全屏幕方式显示输出，持续执行到使用者自行将其使用 Ctrl+C 中断为止。 watch 命令常用来观察一个连续不断更新的文件。

watch 命令的格式为:

```
watch [options] <command>
```

常用参数:

-d : 高亮显示更新时差异的内容。

-n : 设定间格时间，以”秒”为单位(预设值为 2 秒)。

command : 要持续执行的命令，可以是一般的外部指令或 shell 的内建指令，但不能是 alias。

举例:

```
# watch tail /var/log/messages
```

第 5 章 内存监控

内存管理系统是操作系统中最为重要的部分，通过查看内存的使用情况了解系统是否能够顺畅工作。

5.1 使用 free 查看内存

free 命令用来显示内存的使用情况，使用权限是所有用户。

```
[root@inspur ~]# free -m
              total        used        free      shared  buff/cache   available
Mem:           7627        2265        4354          34        1007        4795
Swap:          4095           0        4095
```

如上显示了系统总共拥有 7627M 内存，其中 2265M 已经被使用。在实际的应用系统上，通常会使用 swap 空间来交换物理内存。这种交换虽说使得系统的可用内存总数得到了提高，但是会因为大量的磁盘 IO 降低系统的性能。应用实例 free 命令是用来查看内存使用情况的主要命令。和 top 命令相比，它的优点是使用简单，并且只占用很少的系统资源。通过－S 参数可以使用 free 命令不间断地监视有多少内存在使用，这样可以把它当作一个方便实时监控器。

```
[root@inspur ~]# free -m -s1
              total        used        free      shared  buff/cache   available
Mem:           7627        2265        4353          34        1007        4794
Swap:          4095           0        4095

              total        used        free      shared  buff/cache   available
Mem:           7627        2266        4353          34        1007        4794
Swap:          4095           0        4095

              total        used        free      shared  buff/cache   available
Mem:           7627        2266        4353          34        1007        4794
Swap:          4095           0        4095
```

使用这个命令后终端会连续不断地报告内存使用情况(以 M 字节为单位)，每 1 秒更新一次。

5.2 使用 top 监视系统资源

top 命令可提供有关系统加载、内存使用和 CPU 利用率的日常报告。该命令是作为 procps 程序包的一部分而发布的。

第 6 章 管理磁盘空间

用户需要了解一些磁盘的使用方法，以便使自己和其它用户更有效地使用有限的磁盘空间。为了达到这个目的，管理员必须知道如何决定磁盘空间的使用，如何更有效的管理磁盘空间以及清理磁盘。

6.1 df 命令

在 K-UX 中，可以使用 df（disk filesystem）命令来检查文件系统的自由空间数量。df 命令在文件系统中是使用最广泛的统计工具。它可以显示系统中所有文件系统的各种信息。如：文件系统可使用的自由块数和自由节点数，系统中全部文件系统的自由空间等等。下面是使用 df 命令的一个例子：

```
[root@inspur ~]# df
Filesystem           1K-blocks    Used Available Use% Mounted on
devtmpfs               3839488       0   3839488   0% /dev
tmpfs                  3905216       0   3905216   0% /dev/shm
tmpfs                  3905216   29056   3876160   1% /run
tmpfs                  3905216       0   3905216   0% /sys/fs/cgroup
/dev/mapper/ik-root   36674052 8694248  27979804  24% /
/dev/sda2               999320  242892    687616  27% /boot
tmpfs                   780992    1600    779392   1% /run/user/42
tmpfs                   780992    4096    776896   1% /run/user/1000
tmpfs                   780992      64    780928   1% /run/user/0
```

从 df 命令的输出可以看出，该系统有三个分区已经被 mount 上。其中/dev/sda2 分区，大小为 999320k，已经使用了 242892k，还剩余空间 687616k，使用的空间占总分区的 27%，该分区安装在/boot 目录下。

df 命令使用户可以很容易的得到 K-UX 系统的所有分区的容量和它们的使用情况。通常当系统空间的利用率大于 80%的或者可用空间过小的时候，就应该整理文件系统。备份并且移出不在使用的文件。或者更换更大容量的硬盘。下面这条命令可以显示文件系统的索引节点数：这条命令与上一条命令显示的是同一个系统，而它显示的是 i 节点总数、已使用数量、自由数以及所使用的 i 节点占总数的百分比。每当有一个文件被使用时，就要使用一个 i 节点，如果用户存储了很多的小文件，i 节点表就会溢出，但此时用户可能还有很大的硬盘空间。因为磁盘空间和 i 节点数之间没有什么必然的联系，所以使用 df 命令时，用户应该将上面这两种情况都显示出来。

```
[root@inspur ~]# df -i
Filesystem             Inodes  IUsed    IFree IUse% Mounted on
devtmpfs                59992    403    59589    1% /dev
tmpfs                   61019      1    61018    1% /dev/shm
tmpfs                   61019    767    60252    2% /run
tmpfs                   61019     17    61002    1% /sys/fs/cgroup
/dev/mapper/ik-root 18345984 114749 18231235    1% /
/dev/sda2               65536    248    65288    1% /boot
tmpfs                   61019     23    60996    1% /run/user/42
tmpfs                   61019     33    60986    1% /run/user/1000
tmpfs                   61019     11    61008    1% /run/user/0
```

6.2 du 命令

du 命令可用来显示硬盘的使用情况，可以显示在文件目录中所用到的块的数量。使用这个命令能发现一些过大的目录和文件。

du 命令的一般使用格式为：# du directory

其中，选项 directory 必须是所安装的文件系统中目录的名字。例如，键入命令：

```
[root@inspur ~]# du -s /var/ -h
209M    /var/
```

这个命令可以显示当前目录下所有文件的总的大小。这时将只输出该目录所占的总硬盘块数，而不显示目录下其它的子目录。这条命令在检查系统中各用户所占的磁盘空间是很有用的。

6.3 fsck 命令

K-UX 的启动程序会检查安装好的文件系统有没有损坏或破坏，这种检查在系统每次重启时自动进行。但是，如果不经常关机或硬盘有磁盘错误，那就需要手工检查文件系统。

【注意】在生产环境中，绝对不要使用 fsck 检测在线的文件系统。所以，如果要检查一个文件系统，请先将它卸载(umount)，之后再运行 fsck 命令。在检查根文件系统时，最好使用安装光盘把系统引导到 RESCUE 模式下。然后再执行 fsck。

fsck 命令后面要跟上要检查的文件系统的设备名或安装点。例如，/dev/sda1

被安装在/usr 目录下，则下面命令可以正确地检查该文件系统：

```
[root@inspur ~]# fsck /dev/sda1
```

fsck 命令有很多有用的命令行参数，

参数	说明
-A	一次性检查所有的文件系统。这个选项通常用在 K-UX 引导期间检查所有正常安装的文件系统。如果使用了-A，那么就不能再使用 filesysc 参数
-V	长格式模式。打印有关 fsck 工作状况的附加信息
-t (type)	指定要检查的文件系统的类型
filesys	指定要检查哪个文件系统。这个参数可以是块特殊设备名（如/dev/sda1），或是一个安装点（如/usr）
-a	在不询问任何问题的情况下。自动的修复在文件系统中发现的任何问题。使用这个选项必须要小心
-l	列出文件系统中所有的文件名
-r	在修复文件前请求确认
-s	在检查文件系统前列出管理块的信息

6.4 badblocks 命令

检查磁盘装置中损坏的区块。通常你的硬件提供商也会为你的磁盘设备提供类似的检测工具， 建议使用专业的测试软件，作为一种替代的方法，badblocks 可以提供 K-UX 环境下的坏块检测。 这个命令的用法如下： badblocks [-svw][-b <区块大小>][-o <输出文件>][磁盘装置][磁盘区块数][启始区块] ，例如：

```
[root@inspur ~]# badblocks -v /dev/sda1
Checking blocks 0 to 204799
Checking for bad blocks (read-only test): done
Pass completed, 0 bad blocks found.
```

【注意】建议先 umount 设备，然后再进行坏道检测。仅当/etc/mtab 出现误

报设备挂载错误的时候可以使用-f 选项。

6.5 检查文件系统中的文件有无修改

完整性是安全要求的基本要求之一，为了防止其他人有意的修改你安装的文件，可以通过如下方法检查你的软件系统完整。 Tripware 是一个功能比较强大的软件，可以在系统安装完毕的时候对系统生成一个快照，然后在这些文件有改动的时候通知管理员。更加简单的方法是使用 rpm 命令来检查，rpm 在安装软件包的时候记录了其中文件的 MD5 信息，通过 rpm -V 可以核对系统中当前的文件和原始文件是否一致，并且生成报告。

```
[root@inspur ~]# rpm -V bash
S.5......    /bin/bash
```

上述命令检查的是 bash 软件包，可以发现 bash 的大小和 MD5 值都和原来不同，说明文件被修改。

6.6 磁盘分区工具

在安装 K-UX 操作系统的过程中需要对磁盘进行分区。另外，在系统硬盘空间不够用的情况下需要为系统添加新硬盘，此时就需要对磁盘进行分区操作。

K-UX 环境下通常使用 fdisk 或 parted 工具对磁盘进行分区。

6.6.1 fdisk

fdisk 命令的常用格式是:

```
(1)# fdisk <硬盘设备名>
```

进入 fdisk 的交互操作方式，对指定的硬盘进行分区操作。

```
(2)# fdisk -l <硬盘设备名>
```

在命令行方式下显示指定硬盘的分区表信息。

在 fdisk 的交互操作方式下可以使用若干子命令，见下表。

命令	说明

a	调整硬盘的启动分区
d	删除一个硬盘分区
l	列出所有支持的分区类型
m	列出所有命令
n	创建一个新的分区
p	列出硬盘分区表
q	退出 fdisk，不保存更改
t	更改分区类型
u	切换所显示的分区大小的单位
w	把设置写入硬盘分区表，然后退出

6.6.2 parted

GNU Parted 具有丰富的功能，它除了能够进行分区的添加、删除等常见操作外，还可以进行移动分区、创建文件系统、调整文件系统大小、复制文件系统等操作。

parted 同时还支持 fdisk 所不支持的 GUID 分区表(GUID Partition Table)。

GUID 分区表 (GPT) 是一个实体硬盘的分区表的结构布局的标准。它是可扩展固件接口（EFI）标准（被 Intel 用于替代个人计算机的 BIOS）的一部分，被用于替代 BIOS 系统中的一个扇区来存储逻辑块地址和大小信息的主开机纪录（MBR）分区表。与主启动记录 (MBR) 分区方法相比，GPT 具有更多的优点，因为它允许每个磁盘有多达 128 个分区，支持高达 18 千兆兆字节的卷大小，允许将主磁盘分区表和备份磁盘分区表用于冗余，还支持唯一的磁盘和分区 ID (GUID)。 更多解释请参考 GUID 分区表。

parted 有两种运行模式:命令行模式和交互模式。与 fdisk 的交互模式不同，在 parted 的交互模式下执行命令，一旦按回车键确认，命令就马上执行，对磁盘的更改就立刻生效。

parted 命令的常用格式是:

```
(1)# parted [选项] <硬盘设备名>
(2)# parted [选项] <硬盘设备名> <子命令> [<子命令参数>]
```

格式(1)用于进入 parted 的交互模式，在该模式下输入 parted 的子命令对指定的硬盘进行分区等操作。quit 命令用于退出交互模式。

格式(2)直接在命令行方式下对指定的硬盘进行分区等操作。

其中常用的选项为:

-h：help，显示求助信息

-i：interactive，在必要时提示用户

-l：list，显示所有磁盘设备的分区表

-s：script，从不提示用户

-v：version，显示版本

无论哪种模式，在 parted 中都可以使用若干子命令，见下表。

命令	说明
help [COMMAND]	打印命令的帮助信息，或指定命令的帮助信息。
print [free|NUMBER|all]	显示分区表， 指定编号的分区， 或所有设备的分区表。
mkpart PART-TYPE [FS-TYPE] START END	创建新分区。PART-TYPE 是以下类型之一:primary(主分区)、extended(扩展分区)、logical(逻辑分区)。START 和 END 是新分区开始和结束的具体位置。
rm NUMBER	删除指定编号 NUMBER 的分区。
set NUMBER FLAG STATE	对指定编号 NUMBER 的分区设置分区标记 FLAG。对于 PC 常用的 msdos 分区表来说， 分区标记 FLAG 可有如下取值:"boot"(引导)，"hidden"(隐藏)， "raid"(软 RAID 磁盘阵)，"lvm"(逻辑卷)， "lba" (LBA， Logic Block

	Addressing 模式)。 状态 STATE 的取值是:on 或 off 。
unit UNIT	设置默认输出时表示磁盘大小的单位为 UNIT，UNIT 的常用取值可以为:‘MB’、‘GB’、‘%’(占整个磁盘设备的百分之多少)、‘compact’(人类易读方式，类似于 df 命令中 -h 参数的作用)、‘s’(扇区)、‘cyl’(柱面)、‘chs’ (柱面 cylinders:磁头 heads:扇区 sectors 的地址) 。
mkfs NUMBER FS-TYPE	对指定编号 NUMBER 的分区创建指定类型 FS-TYPE 的文件系统。
mkpartfs PART-TYPE FS-TYPE START END	创建新分区同时创建文件系统。FS-TYPE 是以下类型之一:ext2、fat16、fat32、NTFS、reiserfs、ufs 等。
cp [FROM-DEVICE] FROM-NUMBER TO-NUMBER	将分区 FROM-NUMBER 上的文件系统完整地复制到分区 TO-NUMBER 中，作为可选项还可以指定一个来源硬盘的设备名称 FROM-DEVICE，若省略则在当前设备上进行复制。
move NUMBER START END	将指定编号 NUMBER 的分区移动到从 START 开始 END 结束的位置上。注意:(1)只能将分区移动到空闲空间中。(2)虽然分区被移动了，但它的分区编号是不会改变的。
resize NUMBER START END	对指定编号 NUMBER 的分区调整大小。分区的开始位置和结束位置由 START 和 END 决定。
check NUMBER	检查指定编号 NUMBER 分区中的文件系统是否有什么错误。
rescue START END	恢复靠近位置 START 和 END 之间的分区。

mklabel，mktable LABEL-TYPE	创建一个新的 LABEL-TYPE 类型的空磁盘分区表，对于 PC 而言 msdos 是常用的 LABEL-TYPE。若是用 GUID 分区表，LABEL-TYPE 应该为 gpt。
name NUMBER NAME	为指定编号 NUMBER 的分区命名为 NAME。

6.7 挂载和卸装文件系统

6.7.1 挂载文件系统

当在磁盘分区或逻辑卷上创建了文件系统后，还需要把新建立的文件系统挂载到系统上才能使用。挂载是文件系统中的概念，将所有的文件系统挂载到统一的 / 目录树中。使用 mount 命令可以灵活的挂载系统可识别的所有文件系统。

mount 的命令格式是:

```
格式 1:# mount [-lhV]
格式 2:# mount -a [-t <文件系统类型>] [-O <挂载选项>]
格式 3:# mount [-o <挂载选项> ] <设备名> 或 <挂载点>
格式 4:# mount [-t <文件系统类型>] [-o <挂载选项>] <设备名> <挂载点>
```

其中:

格式 1 用于显示相关消息

a. 不带参数的 mount 命令用于显示当前已经挂载的文件系统

b. -l:在显示当前已经挂载的文件系统时，若是 ext3，ext4 and XFS 文件系统则显示卷标(labels)

c. -h:显示命令帮助

d. -V:显示 mount 工具的版本

格式 2 用于挂载 /etc/fstab 中的所有(-a)不包含 noauto 挂载选项的文件系统

a. -t:若指定此参数，则只挂载 /etc/fstab 中指定类型的文件系统

b. -O :用于指定挂载 /etc/fstab 中包含指定挂载选项的文件系统

c. 若同时指定-t 和-O，则为“或者”的关系

格式 3 用于挂载 /etc/fstab 中已列出的文件系统

a. 选择使用 <设备名> 或 <挂载点> 之一即可

b. 若-o 省略，则使用 /etc/fstab 中该文件系统的挂载选项

格式 4 用于挂载 /etc/fstab 中未列出的文件系统

a. 使用-t 选项可以指定文件系统类型

b. 若-t 选项省略，mount 命令将依次试探 /proc/filesystems 中不包含“nodev”的行

c. 必须同时指定<设备名> 和 <挂载点>

6.7.2 自动挂载文件系统

如果要在系统启动时自动挂载某个文件系统，可以在文件 /etc/fstab 中作相应的设置。

系统挂载表配置文件 /etc/fstab 记录着系统启动时要挂载的文件系统。例如:

```
[root@inspur ~]# cat /etc/fstab

#
# /etc/fstab
# Created by anaconda on Tue Nov  3 00:47:52 2020
#
# Accessible filesystems, by reference, are maintained under '/dev/disk/'.
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info.
#
# After editing this file, run 'systemctl daemon-reload' to update systemd
# units generated from this file.
#
/dev/mapper/ik-root     /                       xfs     defaults        0 0
UUID=aee5b6d6-bf71-45e9-9dc7-0d34a3f68422 /boot                   ext4    defaults        1 2
/dev/mapper/ik-swap     swap                    swap    defaults        0 0
```

/etc/fstab 文件每一行书写一个文件系统的挂载情况，以 # 开头的行为注释行。 文件中的每一行包含如下的 6 列:

a. filesystem :要挂载的设备

b. mount point:挂载点

c. type :挂载的文件系统类型

d. options:挂载选项，多个选项间用逗号隔开

e. dump:使用 dump 命令备份文件系统的频率，空白或者值为零时，系统认为不需备份

f. pass:开机时 fsck 命令会自动检查文件系统，此列规定了检查的顺序。挂

载到/分区的文件系统，此栏位应是 1，其余是 2，0 表示不需检查。

例如，要在系统过程中挂载 xfs 格式的逻辑卷 /dev/wwwVG/www，可以在 /etc/fstab 文件中添加:

```
/dev/wwwVG/www        /www        xfs        defaults        0        0
```

修改/etc/fstab 后执行命令

mount -a

就可以在当前生效。

6.7.3 挂载选项

在 mount 命令中使用 -o 参数可以指定挂载选项，在 /etc/fstab 文件的<options>列中也可以指定挂载选项，常用的挂载选项包括:

选项	说明
defaults	使用 rw，suid，dev，exec，auto，nouser 和 async 挂载设备
async	以非同步方式(延迟写)执行文件系统的输入输出操作
atime	每次访问文件时都更新文件的访问时间，此为默认值
noatime	不更新文件的访问时间，这会提高文件系统的访问速度
auto	使用 mount -a 时自动挂载，开机时也会自动挂载
noauto	使用 mount -a 时不自动挂载，开机时也不会自动挂载，一定要用 mount 命令手动挂载
dev	可解读文件系统上的字符或区块设备
nodev	不能解读文件系统上的字符或区块设备
exec	可执行文件系统上的二进制文件
noexec	不能执行文件系统上的二进制文件
suid	开启 SUID 和 SGID 设置位

nosuid	禁用 SUID 和 SGID 设置位
user	允许指定用户挂载这个文件系统，系统默认只允许 root 挂载，包含 noexec、nosuid、nodev 选项
nouser	使普通用户无法挂载/卸载，只能由 root 用户进行
users	使一般用户可以挂载/卸载，用于桌面环境，包含 noexec、nosuid、nodev 选项
rw	以读写方式挂载文件系统。
ro	以只读方式挂载文件系统。
remount	重新挂载已挂载的文件系统
iocharset=cp936，codepage=936	使挂载的文件系统能支持 GBK 中文文件名，对于繁体中文使用 950，多用于 Windows 文件系统
utf8	对于 ISO9660、ntfs、vfat 文件系统提供 utf8 文件名支持，也可以使用 iocharset=utf8

6.7.4 卸装文件系统

文件系统可以被挂载，也可以被卸装。卸装文件系统的命令是 umount，该命令可以把文件系统从 K-UX 系统中的挂载点分离。umount 命令的格式为:

```
umount <设备名或挂载点>
```

要卸装一个文件系统，可以指定要卸装的文件系统的目录名(挂载点)或设备名。

例如:

umount /dev/sdb1

可以将上面挂载的文件系统从挂载点卸装，也可以用以下命令:

umount /opt

如果一个文件系统处于“busy”状态的时候，不能卸装该文件系统。如下情况将导致文件系统处于“busy”状态:

1. 文件系统上面有打开的文件

2. 某个进程的工作目录在此文件系统上

3. 文件系统上面的缓存文件正在被使用

最典型的错误是在挂载点目录下实施卸装操作，此时文件系统处于“busy”状态。

6.7.5 使用镜像文件

ISO 文件是光盘镜像文件，其文件系统类型是 iso9660，可以用来刻录光盘，然后通过光驱来读取。ISO 文件也可以直接使用，在 Windows 操作系统下使用 ISO 文件需要安装虚拟光驱软件，在 K-UX 下则要简单得多。

先在系统中建立一个挂载点。

mkdir /media/iso

用 mount 命令加 -o loop 选项挂载光盘镜像文件。

mount -o loop K-UX-5.5-ppc64le-dvd1.iso /media/iso

K-UX-5.5-ppc64le-dvd1.iso 是 K-UX 的安装光盘的镜像文件，挂载到 /media/iso 后就成了系统中的一个目录，可以直接读取上面的文件了。

3. 卸装可用命令。

umount /media/iso

6.8 格式化 mkfs

使用方式 ：mkfs [-V] [-t fstype] [fs-options] filesys [blocks] [-L Lable]

说明 ： 在特定的 partition 上建立文件系统。

参数	说明
device	预备检查的硬盘 partition，例如：/dev/sda1
-V	详细显示模式
-t	给定档案系统的型式，K-UX 的预设值为 ext2
-c	在制做档案系统前，检查该 partition 是否有坏道

-l bad_blocks_file	将有坏道的 block 资料加到 bad_blocks_file 里面
block	给定 block 的大小
-L	建立 lable

说明：mkfs 本身并不执行建立文件系统的工作，而是去调用相关的程序来执行。例如，若在"-t" 参数中指定 ext2，则 mkfs 会调用 mke2fs 来建立文件系统，使用时如省略指定【块数】参数，mkfs 会自动设置适当的块数。

6.9 修改档案属性

修改一个档案的属性与权限命令如下所示：

chgrp:改变档案所属群组

chown:改变档案拥有者

chmod:改变档案的权限，SUID，SGID，SBIT 等等的特性

6.9.1 改变所属群组 chgrp

#chgrp [-R] dirname/filename …

选项与参数：

-R：进行递归(recursive)的持续变更，亦即连同次目录下的所有档案、目录都更新成为这个群组之意。常常用在变更某一目录内所有的档案之情况。

6.9.2 改变档案拥有者 chown

#chown [-R] 账户名称 档案或目录

#chown [-R] 账户名称:组名 档案或目录

选项与参数：

-R: 进行递归(recursive)的持续变更，亦即连同次目录下的所有档案都变更

6.9.3 改变权限 chmod

权限的设定方法有两种，分别可以使用数字或者是符号来进行权限的变更。

1. 数字类型改变档案权限

K-UX 档案的基本权限就有九个，分别是 owner/group/others 三种身份各有自己的 read/write/execute 权限。

举例：档案的权限字符为 -rwxrwxrwx 这九个权限是三个三个一组的！其中，可以使用数字来代表各个权限，各权限的分数对照表如下：

r:4 w:2 x:1

每种身份(owner/group/others)各自的三个权限(r/w/x)分数是需要累加的，例如当权限为： [-rwxrwx---] 分数则是：

owner = rwx = 4+2+1 = 7

group = rwx = 4+2+1 = 7

others= --- = 0+0+0 = 0

所以我们设定权限的变更时，该档案的权限数字就是 770， 变更权限的指令 chmod 的语法是这样的：

```
# chmod [-R] xyz 档案或目录
选项与参数：
xyz : 就是刚刚提到的数字类型的权限属性，为 rwx 属性数值的相加。
-R : 进行递归(recursive)的持续变更，亦即连同次目录下的所有档案都会变更
```

2.符号类型改变档案权限

还有一个改变权限的方法，基本上就九个权限分别是(1)user (2)group (3)others 三种身份.那么就可以由 u， g， o 来代表三种身份的权限！此外， a 则代表 all 亦即全部的身份！那么读写的权限就可以写成 r， w， x.也就是可以使用底下的方式来看：

chmod	u	+(加入)	r	
	g	-（出去）	w	档案
	o	=（设定）	x	目录
	a			

第 7 章 查看磁盘 IO

系统的输入输出，特别是磁盘的 IO 对系统性能的影响非常大。通过查看 IO 的情况，判断系统能够满足业务需求。也可以为调整 IO 参数提供帮助。

7.1 使用 vmstat 查看 IO 情况

vmstat 反映了进程的虚拟内存、虚拟内存、磁盘、trap 和 cpu 的活动情况,在多 cpu 系统中，vmstat 在输出结果中平均了 cpu 数量。如下是命令显示。可以看到 swap in, swap out 等 SWAP 交换分区的读写情况，也可以看到设备的 block in, block out 的情况和 cpu 的利用率。需要注意的几个地方： 注意 CPU 的 wa 项数值，这个数值表明 CPU 处理 iowait 所占用的时间比率。数值高代表系统用于处理 io 的时间长。

```
[root@inspur ~]# vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 4312704   3648 1065088    0    0     3     0   51   45  0  0 100  0  0
 0  0      0 4310016   3648 1065088    0    0     0     0  231  223  0  0 100  0  0
 0  0      0 4310016   3648 1065088    0    0     0     0  239  207  0  0 100  0  0
 0  0      0 4310016   3648 1065088    0    0     0     0  249  231  0  0 100  0  0
 0  0      0 4309760   3648 1065088    0    0     0     0  238  223  0  0 100  0  0
```

7.2 使用 iostat 命令

UX 系统中的 iostat 是 I/O statistics（输入/输出统计）的缩写，iostat 工具将对系统的磁盘操作活动进行监视。它的特点是汇报磁盘活动统计情况，同时也会汇报出 CPU 使用情况。同 vmstat 一样，iostat 也有一个弱点，就是它不能对某个进程进行深入分析，仅对系统的整体情况进行分析。iostat 属于 sysstat 软件包。可以用 dnf install sysstat 直接安装。

由 iostat 命令生成的第一份报告提供了关于自从系统被引导后的时间统计信息。后继的每一份报告都包含自上一次报告以来的时间。每次运行 iostat 命令时，就报告所有的统计信息。报告由紧接着一行 tty 和 CPU 统计信息的 tty 和 CPU 头行组成。在多处理器系统上，CPU 统计信息是系统范围计算的，是所有处理器的平均值。iostat 命令用来确定一个物理卷是否正在形成一个性能瓶颈，

以及是否有可能改善这种情况。

7.2.1 命令格式

iostat [参数][时间][次数]

7.2.2 命令功能

通过 iostat 方便查看 CPU、网卡、tty 设备、磁盘、CD-ROM 等等设备的活动情况，负载信息。

7.2.3 命令参数

名称	说明
-c	显示 CPU 使用情况
-d	显示磁盘使用情况
-k	以 KB 为单位显示
-m	以 M 为单位显示
-N	显示磁盘阵列(LVM) 信息
-n	显示 NFS 使用情况
-p[磁盘]	显示磁盘和分区的情况
-t	显示终端和 CPU 的信息
-x	显示详细信息
-V	显示版本信息

7.2.4 使用实例

实例 1：显示所有设备负载情况

命令：iostat

输出：

```
[root@inspur ~]# iostat
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 C

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.10    0.01    0.03    0.01    0.00   99.85

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.44        14.82         3.42     826408     190724
sdb               0.00         0.12         0.00       6697          0
sdc               0.00         0.09         0.00       4961          0
dm-0              0.46        13.61         3.52     758995     196075
dm-1              0.00         0.05         0.00       2816          0
md0               0.00         0.03         0.00       1536          0
loop0             0.00         0.17         0.00       9254          0
```

cpu 属性值说明：

名称	说明
%user	CPU 处在用户模式下的时间百分比
%nice	CPU 处在带 NICE 值的用户模式下的时间百分比
%system	CPU 处在系统模式下的时间百分比
%iowait	CPU 等待输入输出完成时间的百分比
%steal	管理程序维护另一个虚拟处理器时，虚拟 CPU 的无意识等待时间百分比
%idle	CPU 空闲时间百分比

备注：如果%iowait 的值过高，表示硬盘存在 I/O 瓶颈，%idle 值高，表示 CPU 较空闲，如果%idle 值高但系统响应慢时，有可能是 CPU 等待分配内存，此时应加大内存容量。%idle 值如果持续低于 10，那么系统的 CPU 处理能力相对较低，表明系统中最需要解决的资源是 CPU。

实例 2：定时显示所有信息

命令：iostat 2 3

输出：

```
[root@inspur ~]# iostat 2 3
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.10    0.01    0.03    0.01    0.00   99.85

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.44        14.81         3.42     826408     190724
sdb               0.00         0.12         0.00       6697          0
sdc               0.00         0.09         0.00       4961          0
dm-0              0.46        13.61         3.51     758995     196075
dm-1              0.00         0.05         0.00       2816          0
md0               0.00         0.03         0.00       1536          0
loop0             0.00         0.17         0.00       9254          0

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.00         0.00         0.00          0          0
sdb               0.00         0.00         0.00          0          0
sdc               0.00         0.00         0.00          0          0
dm-0              0.00         0.00         0.00          0          0
dm-1              0.00         0.00         0.00          0          0
md0               0.00         0.00         0.00          0          0
loop0             0.00         0.00         0.00          0          0
```

说明：每隔 2 秒刷新显示，且显示 3 次

实例 3：显示指定磁盘信息

命令：iostat -d sda1

输出：

```
[root@inspur ~]# iostat -d sda
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.44        14.74         3.40     826408     190802
```

实例 4：显示 tty 和 cpu 信息

命令：iostat -t

输出：

```
[root@inspur ~]# iostat -t
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

11/03/2020 08:56:05 PM
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.10    0.01    0.03    0.01    0.00   99.85

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.44        14.73         3.40     826408     190802
sdb               0.00         0.12         0.00       6697          0
sdc               0.00         0.09         0.00       4961          0
dm-0              0.45        13.53         3.50     758995     196160
dm-1              0.00         0.05         0.00       2816          0
md0               0.00         0.03         0.00       1536          0
loop0             0.00         0.16         0.00       9254          0
```

实例 5：以 M 为单位显示所有信息

命令：iostat -m

输出：

```
[root@inspur ~]# iostat -m
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.10    0.01    0.03    0.01    0.00   99.85

Device             tps    MB_read/s    MB_wrtn/s    MB_read    MB_wrtn
sda               0.44         0.01         0.00        807        186
sdb               0.00         0.00         0.00          6          0
sdc               0.00         0.00         0.00          4          0
dm-0              0.45         0.01         0.00        741        191
dm-1              0.00         0.00         0.00          2          0
md0               0.00         0.00         0.00          1          0
loop0             0.00         0.00         0.00          9          0
```

实例 6：查看 TPS 和吞吐量信息

命令：iostat -d -k 1 1

输出：

```
[root@inspur ~]# iostat -d -k 1 1
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

Device             tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               0.44        14.70         3.39     826408     190878
sdb               0.00         0.12         0.00       6697          0
sdc               0.00         0.09         0.00       4961          0
dm-0              0.45        13.50         3.49     758995     196243
dm-1              0.00         0.05         0.00       2816          0
md0               0.00         0.03         0.00       1536          0
loop0             0.00         0.16         0.00       9254          0
```

说明：

tps：该设备每秒的传输次数。“一次传输”意思是“一次 I/O 请求”。多个逻辑请求可能会被合并为“一次 I/O 请求”。“一次传输”请求的大小是未知的。

kB_read/s：每秒从设备（drive expressed）读取的数据量；

kB_wrtn/s：每秒向设备（drive expressed）写入的数据量；

kB_read：读取的总数据量；

kB_wrtn：写入的总数据量；

这些单位都为 Kilobytes。

实例 7：查看设备使用率（%util）,响应时间（await）

命令：iostat -d -x -k 1 1

输出：

```
[root@inspur ~]# iostat -d -x -k 1 1
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm r_await w_await aqu-sz rareq-sz wareq-sz  svctm  %util
sda              0.29    0.15     14.63      3.39     0.00     0.02   0.11  12.44    2.61    0.80    0.00    49.94    23.20   2.25   0.10
sdb              0.00    0.00      0.12      0.00     0.00     0.00   0.00   0.00    0.25    0.00    0.00    59.27     0.00   2.21   0.00
sdc              0.00    0.00      0.09      0.00     0.00     0.00   0.00   0.00    1.45    0.00    0.00    55.74     0.00   3.03   0.00
dm-0             0.29    0.17     13.44      3.48     0.00     0.00   0.00   0.00    2.59    0.30    0.00    46.77    21.03   2.16   0.10
dm-1             0.00    0.00      0.05      0.00     0.00     0.00   0.00   0.00    3.33    0.00    0.00    72.21     0.00   3.33   0.00
md0              0.00    0.00      0.03      0.00     0.00     0.00   0.00   0.00    0.00    0.00    0.00    73.14     0.00   0.00   0.00
loop0            0.00    0.00      0.16      0.00     0.00     0.00   0.00   0.00    2.65    0.00    0.00    59.70     0.00   1.61   0.00
```

disk 属性说明：

名称	说明
r/s	每秒完成的读 I/O 设备次数，即 rio/s
w/s	每秒完成的写 I/O 设备次数，即 wio/s
rkB/s	每秒读 K 字节数，是 rsect/s 的一半，因为每扇区大小为 512 字节。
wkB/s	每秒写 K 字节数，是 wsect/s 的一半。
rrqm/s	每秒进行 merge 的读操作数目，即 rmerge/s
wrqm/s	每秒进行 merge 的写操作数目，即 wmerge/s
%rrqm	合并读请求的百分比
%wrqm	合并写请求的百分比
r_await	读请求处理完成等待时间，包含队列中的等待时间和设备实际处理的时间（毫秒）
w_await	写请求处理完成等待时间，包含队列中的等待时间和设备实际处理的时间（毫秒）
aqu-sz	平均 I/O 队列长度
rareq-sz	平均读请求大小（kB）
wareq-sz	平均写请求大小（kB）
svctm	平均每次设备 I/O 操作的服务时间 (毫秒)
%util	磁盘处理 I/O 的时间百分比

备注：如果 %util 接近 100%，说明产生的 I/O 请求太多，I/O 系统已经满负荷，该磁盘可能存在瓶颈。svctm 一般要小于 await (因为同时等待的请求的等待时间被重复计算了)，svctm 的大小一般和磁盘性能有关，CPU/内存的负荷也会对其有影响，请求过多也会间接导致 svctm 的增加。await 的大小一般取决于服务时间(svctm) 以及 I/O 队列的长度和 I/O 请求的发出模式。如果 svctm 比较接近 await，说明 I/O 几乎没有等待时间；如果 await 远大于 svctm，说明 I/O 队列太

长，应用得到的响应时间变慢，如果响应时间超过了用户可以容许的范围，这时可以考虑更换更快的磁盘，调整内核 elevator 算法，优化应用，或者升级 CPU。

队列长度(aqu-sz)也可作为衡量系统 I/O 负荷的指标，但由于 aqu-sz 是按照单位时间的平均值，所以不能反映瞬间的 I/O 洪水。

实例 8：查看 cpu 状态

命令：iostat -c 1 3

输出：

```
[root@inspur ~]# iostat -c 1 3
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.10    0.01    0.03    0.01    0.00   99.86


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00


avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00
```

实例 9：查看更多细节信息

命令：iostat -x /dev/sda

```
[root@inspur ~]# iostat -x /dev/sda
Linux 4.18.0-193.kux.ppc64le (inspur)   11/03/2020      _ppc64le_       (16 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.10    0.01    0.03    0.01    0.00   99.86

Device            r/s     w/s     rkB/s     wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm r_await w_await aqu-sz rareq-sz wareq-sz  svctm  %util
sda              0.28    0.14     13.81      3.24     0.00     0.02   0.11  12.36    2.61    0.79   0.00    49.94    22.71   2.25   0.09
```

如果想查看某个磁盘上所有分区的读写情况，那么使用如下命令。命令会每隔 1 秒输出 IO 情况。

```
[root@inspur ~]# iostat -x /dev/sda 1
```

第 8 章 逻辑卷 LVM

8.1 LVM 简介

LVM(Logical volume Manager)是逻辑卷管理的简称。它是 K-UX 环境下对磁盘分区进行管理的一种机制。

LVM 的工作原理其实很简单，它就是通过将底层的物理硬盘抽象的封装起来，然后以逻辑卷的方式呈现给上层应用。在传统的磁盘管理机制中，上层应用是直接访问文件系统，从而对底层的物理硬盘进行读取，而在 LVM 中，其通过对底层的硬盘进行封装，对底层的物理硬盘进行操作时，其不再是针对于分区进行操作，而是通过一个叫做逻辑卷的东西来对其进行底层的磁盘管理操作。

LVM 最大的特点就是可以对磁盘进行动态管理。因为逻辑卷的大小是可以动态调整的，而且不会丢失现有的数据。

8.2 LVM 的原理

理解好 LVM 的原理，必须要掌握 4 个基本的逻辑卷概念。

PE ： physical extend，物理块

PV ： physical volume，物理卷

VG ： volume group，卷组

LV ： logical volume，逻辑卷

使用 LVM 对磁盘进行动态管理以后，以逻辑卷的方式呈现给上层的服务的。所有的操作目的，其实就是去创建一个 LV(Logical Volume)， 逻辑卷就是用来取代之前的分区，通过对逻辑卷进行格式化，然后进行挂载操作就可以使用了。

LVM 的工作原理总结：

(1)物理磁盘被格式化为 PV，空间被划分为一个个的 PE

(2)不同的 PV 加入到同一个 VG 中，不同 PV 的 PE 全部进入到了 VG 的 PE 池内

(3)LV 基于 PE 创建，大小为 PE 的整数倍，组成 LV 的 PE 可能来自不同的物理磁盘

(4)LV 现在就直接可以格式化后挂载使用了

(5)LV 的扩充缩减实际上就是增加或减少组成该 LV 的 PE 数量，其过程不会丢失原始数据

8.3 创建 LVM 逻辑卷

8.3.1 物理硬盘格式化

使用 pvcreate 命令

这里已经事先虚拟化了 3 块物理硬盘，每块硬盘的大小为 2G，通过 fdisk -l 命令可以查看。

```
[root@inspur ~]# fdisk -l

Disk /dev/sda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0007faf6

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          39      307200   83  Linux
Partition 1 does not end on cylinder boundary.
/dev/sda2              39        2358    18631680   83  Linux
/dev/sda3            2358        2611     2031616   82  Linux swap / Solaris

Disk /dev/sdb: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xf15e576c

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1         261     2096451   fd  Linux raid autodetect

Disk /dev/sdc: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disk identifier: 0x450342a2

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1         261     2096451   fd  Linux raid autodetect

Disk /dev/sdd: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0dad147b

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1               1         261     2096451   fd  Linux raid autodetect
```

根据上表所示，先将/dev/sdb，/dev/sdc 两块硬盘格式化为物理卷（PV）。

```
[root@inspur ~]# pvcreate /dev/sdb1 /dev/sdc1
  Physical volume "/dev/sdb1" successfully created
  Physical volume "/dev/sdc1" successfully created
```

创建 PV 完成后，可以使用 pvdisplay(显示详细信息)、pvs 命令来查看当前 pv 的信息：

```
[root@inspur ~]# pvdisplay
  "/dev/sdb1" is a new physical volume of "2.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdb1
  VG Name
  PV Size               2.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               F1fpDS-DX8h-Z1kZ-KDPr-A71T-zs1I-CojWqk

  "/dev/sdc1" is a new physical volume of "2.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/sdc1
  VG Name
  PV Size               2.00 GiB
  Allocatable           NO
  PE Size               0
  Total PE              0
  Free PE               0
  Allocated PE          0
  PV UUID               2iFfuh-NVSb-17SF-nBWJ-qaP1-PQyB-UYL4Fe
```

```
[root@inspur ~]# pvs
  PV         VG   Fmt  Attr PSize PFree
  /dev/sdb1       lvm2 a--  2.00g 2.00g
  /dev/sdc1       lvm2 a--  2.00g 2.00g
```

通过这两个命令可以看到已经创建好的 PV 的信息，两个 PV 都是 2G，目前还没有使用，PFree 都是 2G。

8.3.2 创建卷组

在创建完 PV 以后，这时候需要通过 vgcreate 命令创建一个卷组（VG），然后将 PV 都加入到这个卷组当中，在创建卷组时要给该卷组起一个名字。

```
[root@inspur ~]# vgcreate inspur /dev/sdb1 /dev/sdc1
  Volume group "inspur" successfully created
```

同样，创建 VG 完成后，也可以使用 vgdisplay 或者 vgs 命令来查看 VG 的信息。

```
[root@inspur ~]# vgs
  VG     #PV #LV #SN Attr   VSize VFree
  inspur   2   0   0 wz--n- 3.99g 3.99g
```

8.3.3 基于卷组创建逻辑卷

因为创建好的 PV、VG 都是底层的东西，上层使用的是逻辑卷，所以要基于 VG

通过 lvcreate 命令创建逻辑卷（LV）才行。

```
[root@inspur ~]# lvcreate -n mylv -L 2G inspur
  Logical volume "mylv" created
```

通过 lvcreate 命令基于 VG 创建好的逻辑卷，名字为 mylv，大小为 2G，同样可以使用 lvdisplay 或者 lvs 命令来查看创建好的逻辑卷的信息。

```
[root@inspur ~]# lvdisplay
  --- Logical volume ---
  LV Path                /dev/inspur/mylv
  LV Name                mylv
  VG Name                inspur
  LV UUID                UJqy7s-gPUi-pOCi-xbek-p9q6-ihI7-Rb98qu
  LV Write Access        read/write
  LV Creation host, time inspur, 2015-09-07 01:51:57 -0700
  LV Status              available
  # open                 0
  LV Size                2.00 GiB
  Current LE             512
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0

[root@inspur ~]# lvs
  LV   VG     Attr      LSize Pool Origin Data%  Move Log Cpy%Sync Convert
  mylv inspur -wi-a---- 2.00g
```

这样逻辑卷已经创建好，此时通过 vgs 和 pvs 命令查看一下 PV 与 VG 的信息。

```
[root@inspur ~]# vgs
  VG     #PV #LV #SN Attr   VSize VFree
  inspur   2   1   0 wz--n- 3.99g 1.99g
[root@inspur ~]# pvs
  PV         VG     Fmt  Attr PSize PFree
  /dev/sdb1  inspur lvm2 a--  2.00g    0
  /dev/sdc1  inspur lvm2 a--  2.00g 1.99g
```

【注意】每创建完一个 LV 时，VG 与 PV 的信息都是时时在变化的，并且创建 LV 的大小是根据当前 VG 的大小来决定的，不能超过当前 VG 的剩余大小！

每创建好一个逻辑卷，都会在 /dev 目录下出现一个以该卷组命名的文件夹，基于该卷组创建的所有的逻辑卷都是存放在这个文件夹下面，可以查看一下。

```
[root@inspur ~]# ls /dev/inspur/mylv
/dev/inspur/mylv
```

每创建一个新的逻辑卷，该 VG 目录下都会多出这么一个设备。

8.4 格式化并使用逻辑卷

已经创建好了的 PV、VG 以及 LV，如果要使用逻辑卷，就必须将其格式化成需要用的文件系统，并将其挂载起来，然后就可以像使用分区一样去使用逻辑卷 了 。

```
[root@inspur ~]# mkfs.ext4 /dev/inspur/mylv
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
131072 inodes, 524288 blocks
26214 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=536870912
16 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 27 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

格式化逻辑卷以后，就可以使用 mount 命令将其进行挂载，将其挂载到 /mnt 目录下。

```
[root@inspur ~]# mount /dev/inspur/mylv /mnt
[root@inspur ~]# mount
/dev/sda2 on / type ext4 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw,rootcontext="system_u:object_r:tmpfs_t:s0")
/dev/sda1 on /boot type ext4 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
vmware-vmblock on /var/run/vmblock-fuse type fuse.vmware-vmblock (rw,nosuid,nodev,defaul
t_permissions,allow_other)
/dev/mapper/inspur-mylv on /mnt type ext4 (rw)
[root@inspur ~]# ls
anaconda-ks.cfg  install.log  install.log.syslog
[root@inspur ~]# cd /mnt
[root@inspur mnt]# ls
lost+found
[root@inspur mnt]# touch inspur.txt
[root@inspur mnt]# ls
inspur.txt  lost+found
```

卷组已经挂载好了，并且可以像使用分区一样来对其进行文件操作了。

8.5 删除逻辑卷

在创建好逻辑卷后可以通过创建文件系统，挂载逻辑卷来使用它，如果说不想用了也可以将其删除掉。

【注意】对于创建物理卷、创建卷组以及创建逻辑卷是有严格顺序的，同样，对于删除逻辑卷、删除卷组以及删除物理卷也有严格的顺序要求。

1. 通过 umount 命令将正在使用的逻辑卷卸载；
2. 通过 lvremove 命令将逻辑卷先删除；
3. 通过 vgremove 命令删除卷组；
4. 通过 pvremove 命令删除物理卷。

```
[root@inspur ~]# umount /mnt
[root@inspur ~]# lvremove /dev/inspur/mylv
Do you really want to remove active logical volume mylv? [y/n]: y
  Logical volume "mylv" successfully removed
[root@inspur ~]# vgremove inspur
  Volume group "inspur" successfully removed
[root@inspur ~]# pvremove /dev/sdb1
  Labels on physical volume "/dev/sdb1" successfully wiped
```

此时刚创建的逻辑卷 mylv，卷组 inspur 以及物理卷 /dev/sdb 已经从当前操作系统上删除掉了，通过 lvs、vgs、pvs 命令可以查看一下。

```
[root@inspur ~]# pvremove /dev/sdb1
  Labels on physical volume "/dev/sdb1" successfully wiped
[root@inspur ~]# lvs
  No volume groups found
[root@inspur ~]# vgs
  No volume groups found
[root@inspur ~]# pvs
  PV         VG   Fmt  Attr PSize PFree
  /dev/sdc1       lvm2 a--  2.00g 2.00g
```

8.6 扩容逻辑卷

相比于传统磁盘管理方式的各种问题，使用 LVM 逻辑卷可以对我们的磁盘进行动态的管理。在传统的磁盘管理方式中，我们如果出现分区大小不足的情况下，我们此时只能通过加入一块物理硬盘，然后对其进行分区，因为加入的硬盘作为独立的文件系统存在，所有对原有分区并没有影响，如果此时需要扩大分区，就只能先将之前的分区先卸载掉，然后将所有的信息转移到新的分区下，最后在将新的分区挂载上去，如果是在生产环境下，这样是不可想象的，正因为如此，才出现了 LVM 的磁盘管理方式，可以动态的对我们的磁盘进行管理。

对逻辑卷的扩容，其实际就是向逻辑卷中增加 PE 的数量，而 PE 的数量是由 VG 中剩余 PE 的数量所决定的。

逻辑卷的扩容是可以在线进行的，所有要先将逻辑卷挂载上，并在使用情况下动态扩容逻辑卷。

```
[root@inspur ~]# mount /dev/inspur/mylv /mnt
[root@inspur ~]# cd /mnt/
[root@inspur mnt]# vim inspur.txt
[root@inspur mnt]# cat inspur.txt
this is a test
[root@inspur mnt]# ls
inspur.txt  lost+found
```

查看当前的 VG 的信息，保证 VG 中有足够的空闲空间，通过 vgdispaly 或者 vgs 命令。

```
[root@inspur mnt]# vgdisplay
  --- Volume group ---
  VG Name               inspur
  System ID
  Format                lvm2
  Metadata Areas        2
  Metadata Sequence No  2
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                2
  Act PV                2
  VG Size               3.99 GiB
  PE Size               4.00 MiB
  Total PE              1022
  Alloc PE / Size       512 / 2.00 GiB
  Free  PE / Size       510 / 1.99 GiB
  VG UUID               0Vql2K-uaU7-jjc5-GPxw-FCbf-hn8F-GlTVZb

[root@inspur mnt]# vgs
  VG     #PV #LV #SN Attr   VSize VFree
  inspur   2   1   0 wz--n- 3.99g 1.99g
```

在 vg 中还有足够的空闲空间时，就可以动态的对逻辑卷进行扩容操作。

通过 lvextend 命令扩充逻辑卷。 比如这里要对 mylv 逻辑卷扩充 1G 的大小，此时就可以使用# lvextend -L +1G /dev/inspur/mylv 命令来执行操作。

```
[root@localhost ~]# lvs
  LV   VG     Attr      LSize Pool Origin Data%  Move Log Cpy%Sync Convert
  mylv inspur -wi-ao--- 2.00g
[root@localhost ~]# lvextend -L +1G /dev/inspur/mylv
  Extending logical volume mylv to 3.00 GiB
  Logical volume mylv successfully resized
[root@localhost ~]#
```

查看完成扩容后 LV 大小，扩容完成后可以使用 lvdisplay 或者 lvs 命令来查看一下当前 lv 的信息。

```
[root@inspur ~]# lvdisplay
  --- Logical volume ---
  LV Path                /dev/inspur/mylv
  LV Name                mylv
  VG Name                inspur
  LV UUID                ch0pm2-AAtl-WAfP-4xlj-ZpzL-kQX7-EAOW57
  LV Write Access        read/write
  LV Creation host, time inspur, 2015-09-07 02:33:53 -0700
  LV Status              available
  # open                 1
  LV Size                3.00 GiB
  Current LE             768
  Segments               2
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0

[root@inspur ~]# lvs
  LV   VG     Attr      LSize Pool Origin Data%  Move Log Cpy%Sync Convert
  mylv inspur -wi-ao--- 3.00g
```

这时发现扩容后的逻辑卷大小变成 3G，此时正在使用逻辑卷，并没有卸载掉该逻辑卷，同时查看逻辑卷里面的内容，发现里面的文件还在，并且没有受到损害。

```
[root@inspur ~]# cd /mnt
[root@inspur mnt]# ls
inspur.txt  lost+found
[root@inspur mnt]# cat inspur.txt
this is a test
```

通过 resize2fs 命令更新文件系统。

在对逻辑卷进行扩容以后，通过 df -h 命令可以查看一下当前的文件系统信息。

```
[root@inspur mnt]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2              18G  2.5G   15G  15% /
tmpfs                 491M   84K  491M   1% /dev/shm
/dev/sda1             291M   33M  243M  12% /boot
/dev/mapper/inspur-mylv
                      2.0G   68M  1.9G   4% /mnt
```

发现当前挂载的逻辑卷的文件系统大小还是 2G，并没有变成 3G，其原因就是文件系统是在创建完 LV 以后就马上格式化的，此后对逻辑卷进行扩容后，其并不会改变当前的文件系统，所以这个时候必须更新文件系统，通过 resize2fs 命令

```
[root@inspur mnt]# resize2fs /dev/inspur/mylv
resize2fs 1.41.12 (17-May-2010)
Filesystem at /dev/inspur/mylv is mounted on /mnt; on-line resizing required
old desc_blocks = 1, new_desc_blocks = 1
Performing an on-line resize of /dev/inspur/mylv to 786432 (4k) blocks.
The filesystem on /dev/inspur/mylv is now 786432 blocks long.

[root@inspur mnt]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2              18G  2.5G   15G  15% /
tmpfs                 491M   84K  491M   1% /dev/shm
/dev/sda1             291M   33M  243M  12% /boot
/dev/mapper/inspur-mylv
                      3.0G   68M  2.8G   3% /mnt
```

这时文件系统也已经更新了，大小变成了 3G。通过上面的步骤可以发现，扩容一个逻辑卷其实是非常简单的，首先就是保持 VG 中有足够的空闲空间，其次对逻辑卷进行动态扩容，最后在扩容完逻辑卷以后还必须要更新文件系统。同时对逻辑卷进行扩容并不需要先卸载掉逻辑卷，可以在线进行，并且逻辑卷里面的文件内容都不会发生变化。

8.7 扩容卷组

如果此时 VG 中的 PE 数量不足。如果需要扩容逻辑卷，发现卷组中的空间已经不够了，这个时候就必须对卷组进行扩容，使得卷组中有足够的空闲空间，最后再来扩容逻辑卷。卷组其实就是将多块 PV 加入到 VG 当中，所以卷组的扩容也非常简单，只需要增加一块物理硬盘，将其格式化成 PV，然后再将这个 PV 加入到该卷组中即可。

这里首先模拟一下将 VG 中的剩余空间全部扩容到逻辑卷中，然后通过增加一块物理硬盘，来对卷组进行扩容操作。

```
[root@inspur mnt]# vgs
  VG     #PV #LV #SN Attr   VSize VFree
  inspur   2   1   0 wz--n- 3.99g 1016.00m
[root@inspur mnt]# lvextend -L +1016M /dev/inspur/mylv
  Extending logical volume mylv to 3.99 GiB
  Logical volume mylv successfully resized
[root@inspur mnt]# resize2fs /dev/inspur/mylv
resize2fs 1.41.12 (17-May-2010)
Filesystem at /dev/inspur/mylv is mounted on /mnt; on-line resizing required
old desc_blocks = 1, new_desc_blocks = 1
Performing an on-line resize of /dev/inspur/mylv to 1046528 (4k) blocks.
The filesystem on /dev/inspur/mylv is now 1046528 blocks long.

[root@inspur mnt]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda2              18G  2.5G   15G  15% /
tmpfs                 491M   84K  491M   1% /dev/shm
/dev/sda1             291M   33M  243M  12% /boot
/dev/mapper/inspur-mylv
                      4.0G   68M  3.7G   2% /mnt
```

这个时候如果还有对逻辑卷进行扩容，但是此时 VG 中空闲空间的大小以及不够用，这个时候就需要对卷组进行动态扩容了。

将要添加到 VG 的硬盘格式化成 PV 通过 pvcreate 命令。

```
[root@inspur mnt]# pvcreate /dev/sdd1
  Physical volume "/dev/sdd1" successfully created
[root@inspur mnt]# pvs
  PV         VG     Fmt  Attr PSize PFree
  /dev/sdb1  inspur lvm2 a--  2.00g    0
  /dev/sdc1  inspur lvm2 a--  2.00g    0
  /dev/sdd1         lvm2 a--  2.00g 2.00g
```

将新的 PV 添加到指定的 VG 当中，通过 vgextend 命令。

```
[root@inspur mnt]# vgextend inspur /dev/sdd1
  Volume group "inspur" successfully extended
[root@inspur mnt]# pvs
  PV         VG     Fmt  Attr PSize PFree
  /dev/sdb1  inspur lvm2 a--  2.00g    0
  /dev/sdc1  inspur lvm2 a--  2.00g    0
  /dev/sdd1  inspur lvm2 a--  2.00g 2.00g
```

查看当前 VG 信息，通过 vgdisplay 或 vgs 命令。

```
[root@inspur mnt]# vgdisplay
  --- Volume group ---
  VG Name               inspur
  System ID
  Format                lvm2
  Metadata Areas        3
  Metadata Sequence No  5
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                3
  Act PV                3
  VG Size               5.99 GiB
  PE Size               4.00 MiB
  Total PE              1533
  Alloc PE / Size       1022 / 3.99 GiB
  Free  PE / Size       511 / 2.00 GiB
  VG UUID               0Vql2K-uaU7-jjc5-GPxw-FCbf-hn8F-GlTVZb

[root@inspur mnt]# vgs
  VG     #PV #LV #SN Attr   VSize VFree
  inspur   3   1   0 wz--n- 5.99g 2.00g
```

第 9 章 软件包的管理

9.1 使用 RPM 管理软件包

使用 RPM 最大的好处在于它提供快速之安装,减少编译安装的排错困扰。对于最终用户来说,RPM 所提供的众多功能使维护系统要比以往容易的多。安装、卸载和升级 RPM 软件包均只需一条命令即可完成。RPM 维护一个所有已安装的软件包和文件的数据库,可以进行功能强大的软件包查询和验证工作。在软件包升级过程中,RPM 会对配置文件进行特别处理,绝对不会丢失以往的定制信息——这对于直接使用.tar.gz 文件是不可能的。对于程序员，RPM 可以让您连同软件的源代码打包成源代码和二进制软件包供最终用户使用。这个过程十分简单,整个过程由一个主文件和可能的补丁程序组成。RPM 在软件的新版本发布时,这种“原始”源代码,补丁程序和软件生成指令的清晰描述简化了软件包的维护工作。

9.1.1 RPM 的功能

简言之,RPM 具有如下五大功能:

安装——将软件从包中解出来,并且安装到硬盘。

卸载——将软件从硬盘清除。

升级——替换软件的旧版本。

查询——查询软件包的信息。

验证——检验系统中的软件与包中软件的区别。

9.1.2 RPM 命令用法

rpm 的完整语法参见 rpm 命令手册，下面只列出较常见的用法。

命令	说明
rpm -i <.rpm file name>	安装指定的.rpm 文件
rpm -U <.rpm file name>	用指定的.rpm 文件升级同名包

rpm -e <package-name>	删除指定的软件包
rpm -q <package-name>	查询指定的软件包在系统中是否安装
rpm -qa	查询系统中安装的所有 RPM 软件包
rpm -qf </path/to/file>	查询系统中指定文件所属的软件包
rpm -qi <package-name>	查询一个已安装软件包的描述信息
rpm -ql <package-name>	查询一个已安装软件包里所包含的文件
rpm -qc <package-name>	查看一个已安装软件包的配置文件位置
rpm -qd <package-name>	查看一个已安装软件包的文档安装位置
rpm -qR <package-name>	查询一个已安装软件包的最低依赖要求
rpm -qpi <.rpm file name>	查询一个未安装的 RPM 文件的描述信息
rpm -qpl <.rpm file name>	查询一个未安装的 RPM 文件里所包含的文件
rpm -qpc <.rpm file name>	查看一个未安装的 RPM 文件的配置文件位置
rpm -qpd <.rpm file name>	查看一个未安装的 RPM 文件的文档安装位置
rpm -qpR <.rpm file name>	查询一个未安装的 RPM 文件的最低依赖要求
rpm -V <package-name>	校验指定的软件包
rpm -V </path/to/file>	校验包含指定文件的软件包
rpm -Vp <.rpm file name>	校验指定的未安装的 RPM 文件
rpm -Va	校验所有已安装的软件包
rpm --rebuilddb	重新创建系统的 RPM 数据库,用于不能安装和查询的情况
rpm --import <key file>	导入指定的签名文件
rpm -Kv --nosignature <.rpm file name>	检查指定的 RPM 文件是否已损坏或被恶意篡改(验证包的 MD5 校验和)
rpm -K <.rpm file name>	检查指定 RPM 文件的 GnuPG 签名

a.在安装/升级时,还可以使用 -vh 参数,其中:v 表示在安装过程中将显示较详细的信息;h 表示显示水平进度条

b.在使用 rpm -qa 命令时,还可以使用 |more 或 |grep 进行过滤

c.所有的 <.rpm filename > 既可以是本地文件,也可以是远程文件

d.校验软件包将检查软件包中的所有文件是否与系统中所安装的一致性。包括校验码文件大小,存取权限和属主属性都将根据数据库进行校验。该操作可在用户安装了新程序以后怀疑某些文件遭到破坏时使用。

9.1.3 RPM 使用实例

1. 安装一个新的 rpm 软件包

在安装完一个全新系统后，并不是所有的软件包都被安装到这个系统中。工作中很可能会出现一个命令或者服务没有的情况，那么这时，就需要使用命令安装软件包。首先，需要准备好该命令或者服务的软件包，然后进入到该目录中，使用命令如下：

```
[root@inspur Server]# rpm -ivh vim-minimal-7.0.109-4.el5_2.4z.ia64.rpm
```

2. 删除一个已经存在的软件包

当不需要一个软件包时，K-UX 可以随时将其卸载，使用命令如下：

```
[root@inspur Server]# rpm -e vim-minimal-7.0.109-4.el5_2.4z.ia64.rpm
```

更新一个软件包

如果需要对系统中一个软件包进行更新，使用命令如下：

```
[root@inspur Server]# rpm -U vim-minimal-7.0.109-4.el5_2.4z.ia64.rpm
```

查看软件包

查看系统中安装的所有软件包，使用命令如下：

```
[root@inspur ~]# rpm -qa
```

9.2 使用 YUM 管理软件包

yum 具有如下特点:

a.自动解决包的倚赖性问题能更方便的添加/删除/更新 RPM 包

b.便于管理大量系统的更新问题

c.可以同时配置多个资源库(Repository)

d.简洁的配置文件(/etc/yum.conf)

e.保持与 RPM 数据库的一致性

f.有一个比较详细的 log,可以查看何时升级安装了什么软件包等

g.使用方便

9.2.1 配置本地 YUM 服务器

1. 放入 K-UX 5.2 光盘或找到镜像文件，使用命令如下：

```
[root@inspur ~]# mount /work/KUX-5.2.1-ppc64le-dvd1.iso /mnt/iso
```

2. 编辑系统中的 repo 文件，使用命令如下：

```
[root@inspur ~]# vim /etc/yum.repos.d/inspur-Media.repo
```

3. 重写 repo 文件，内容如下：

```
[BaseOS]
name=K-UX-BaseOS
baseurl=file:///mnt/iso/BaseOS
gpgcheck=0
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
[AppStream]
name=K-UX-AppStream
baseurl=file:///mnt/iso/AppStream
gpgcheck=0
enabled=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-centosofficial
```

4. 清除 YUM 缓存，使用命令如下：

```
[root@inspur ~]# yum clean all
```

5、建立新缓存，使用命令如下

```
[root@inspur ~]# yum makecache
```

至此，本地 YUM 配置完毕。

9.2.2 YUM 命令用法

yum 完整语法参见其命令手册,下面只列出较常见的用法。

命令	功能
yum check-update	检查可更新的所有软件包
yum update	下载更新系统已安装的所有软件包
yum upgrade	大规模的版本升级,与 yum update 不同的是,连旧的淘汰的包也升级
yum install <packages>	安装新软件包
yum update <packages>	更新指定的软件包
yum remove <packages>	卸载指定的软件包
yum groupinstall <groupnames>	安装指定软件组中的软件包
yum groupupdate <groupnames>	更新指定软件组中的软件包
yum groupremove <groupnames>	卸载指定软件组中的软件包
yum grouplist	查看系统中已经安装的和可用的软件组
yum list	列出资源库中所有可以安装或更新以及已经安装的 rpm 包
yum list <regex>	列出资源库中与正则表达式匹配的可以安装或更新以及已经安装的 rpm 包
yum list available	列出资源库中所有可以安装的 rpm 包
yum list available <regex>	列出资源库中与正则表达式匹配的所有可以安装的 rpm 包

yum list updates	列出资源库中所有可以更新的 rpm 包
yum list updates <regex>	列出资源库中与正则表达式匹配的所有可以更新的 rpm 包
yum list installed	列出资源库中所有已经安装的 rpm 包
yum list installed <regex>	列出资源库中与正则表达式匹配的所有已经安装的 rpm 包
yum list extras	列出已经安装的但是不包含在资源库中的 rpm 包
yum list extras <regex>	列出与正则表达式匹配的已经安装的但是不包含在资源库中的 rpm 包
yum list recent	列出最近被添加到资源库中的软件包
yum search <regex>	检测所有可用的软件的名称、描述、概述和已列出的维护者,查找与正则表达式匹配的值
yum provides <regex>	检测软件包中包含的文件以及软件提供的功能,查找与正则表达式匹配的值
yum clean packages	清除缓存中 rpm 包文件
yum clean all	清除缓存中的 rpm 头文件和包文件
yum deplist <packages>	显示软件包的依赖信息

9.2.3 YUM 使用实例

1、安装软件包，使用命令如下：

```
[root@inspur ~]# yum install vim-minimal-8.0.1763-13.kux.ppc64le
```

2、卸载软件包，使用命令如下：

```
[root@inspur ~]# yum remove vim-minimal-8.0.1763-13.kux.ppc64le
```

3、更新软件包，使用命令如下：

```
[root@inspur ~]# yum update vim-minimal-8.0.1763-13.kux.ppc64le
```

9.3 使用 DNF 管理软件包

dnf 具有如下特点:

a.使用 libsolv 来解决依赖关系，由 SUSE 开发和维护；

b.由 C、C++、Python 编写；

c.支持各种扩展；

d.API 有完整的文档，很容易创建新功能；

e.在同步存储库的元数据时，使用内存较少；

f.使用满足性算法来解决依赖关系解析；

g.安装包的依赖关系不更新；

h.在 dnf 更新过程中，如果包关的依赖，。

9.3.1 DNF 命令用法

下面列出 dnf 命令较常见的用法。

命令	功能
dnf --version	查看系统中的 DNF 包管理器的版本
dnf check-update	检查系统系统中所有软件包的更新
dnf update dnf upgrade	升级系统中所有有可用升级的软件包
dnf install <packages>	安装软件包及其所需的所有依赖
dnf update <packages>	升级指定的软件包
dnf remove <packages>	卸载指定的软件包
dnf groupinstall <groupnames>	安装指定软件包组
dnf groupupdate <groupnames>	升级指定软件包组中的软件包
dnf groupremove <groupnames>	卸载指定软件包组中的软件包
dnf grouplist	查看系统中已经安装的和可用的软件包组

dnf repolist	查看系统中可用的 DNF 库
dnf repolist all	查看系统中可用的和不可用的 DNF 库
dnf search <packages>	查找系统中可用的 rpm 包
dnf list dnf list <regex>	列出资源库中所有可以安装或更新以及已经安装的 rpm 包 列出资源库中与正则表达式匹配的可以安装或更新以及已经安装的 rpm 包
dnf list available	列出资源库中所有可以安装的 rpm 包
dnf list available <regex>	列出资源库中与正则表达式匹配的所有可以安装的 rpm 包
dnf list updates	列出资源库中所有可以更新的 rpm 包
dnf list updates <regex>	列出资源库中与正则表达式匹配的所有可以更新的 rpm 包
dnf list installed dnf list installed <regex>	列出资源库中所有已经安装的 rpm 包 列出资源库中与正则表达式匹配的所有已经安装的 rpm 包
dnf list extras	列出已经安装的但是不包含在资源库中的 rpm 包
dnf list extras <regex>	列出与正则表达式匹配的已经安装的但是不包含在资源库中的 rpm 包
dnf list recent dnf info <packages>	列出最近被添加到资源库中的软件包 查看软件包详情
dnf clean packages dnf clean all dnf deplist <packages>	清除缓存中 rpm 包文件 清除缓存中的 rpm 头文件和包文件 显示软件包的依赖信息

9.3.2 DNF 使用实例

1. 安装软件包，使用命令如下：

```
[root@inspur ~]# dnf install vim-minimal-8.0.1763-13.kux.ppc64le
```

2. 卸载软件包，使用命令如下：

```
[root@inspur ~]# dnf remove vim-minimal-8.0.1763-13.kux.ppc64le
```

3. 更新软件包，使用命令如下：

```
[root@inspur ~]# dnf update vim-minimal-8.0.1763-13.kux.ppc64le
```

9.3.3 DNF module 模块用法

K-U 系统中 dnf 具有 module 功能，该功能主要用于切换不同版本的软件，其主要用于快速替换升级当前使用软件版本。

基础使用方法：

命令	功能
dnf module enable [模块名称]	启用模块
dnf module info [模块名称]	查询模块信息
dnf module remove [模块名称]	卸载模块
dnf module provides [模块名称]	查询模块的提供软件库信息
dnf module list [模块名称]	查询模块的详细信息
dnf module update [模块名称]	更新模块
dnf module reset [模块名称]	重置模块
dnf module disable [模块名称]	禁用模块

9.3.4 DNF module 使用实例

1、查询所有软件流

```
[root@inspur ~]# dnf module list
```

2、安装 php 模块

```
[root@inspur ~]# dnf module install php
```

3、安装指定版本的 php 模块

```
[root@inspur ~]# dnf module isntall php:7.2
```

4、重置 php 模块

```
[root@inspur ~]# dnf module reset php
```

第 10 章 查看硬件配置命令

10.1 系统

选项	说明
uname -a	查看内核/操作系统/CPU 信息
cat /etc/inspur-release	查看操作系统版本
cat /proc/cpuinfo	查看 CPU 信息
hostname	查看计算机名
lspci -tv	列出所有 PCI 设备
lsusb -tv	列出所有 USB 设备
lsmod	列出加载的内核模块
env	查看环境变量

10.2 资源

选项	说明
free -m	查看内存使用量和交换区使用量
df -h	查看各分区使用情况
du -sh <目录名>	查看指定目录的大小
grep MemTotal /proc/meminfo	查看内存总量
grep MemFree /proc/meminfo	查看空闲内存量
uptime	查看系统运行时间、用户数、负载
cat /proc/loadavg	查看系统负载

10.3 磁盘和分区

选项	说明

mount | column -t	查看挂接的分区状态
fdisk -l	查看所有分区
swapon -s	查看所有交换分区
hdparm -I /dev/hda	查看磁盘参数(仅适用于 IDE 设备)
dmesg | grep IDE	查看启动时 IDE 设备检测状况

10.4 网络

选项	说明
nmcli connection show	查看所有网络连接配置
nmcli connection show --active	仅列出处于活动状态的网络配置
nmcli connection up name	激活一个网卡
nmcli connection down name	停用一个网卡
firewall-cmd --list-all	查看防火墙规则
firewall-cmd --list-port	查看防火墙所有打开的端口
firewall-cmd --list-services	查看防火墙所有打开的服务
route -n	查看路由表
netstat -lntp	查看所有监听端口
netstat -antp	查看所有已经建立的连接
netstat -s	查看网络统计信息

10.5 进程

选项	说明
ps -ef	查看所有进程
top	实时显示进程状态

10.6 用户

选项	说明
w	查看活动用户
id <用户名>	查看指定用户信息
last	查看用户登录日志
cut -d: -f1 /etc/passwd	查看系统所有用户
cut -d: -f1 /etc/group	查看系统所有组
crontab -l	查看当前用户的计划任务

10.7 服务

选项	说明
systemctl list-units	列出所有系统服务
systemctl list-units --type=service	列出所有启动的系统服务

第 11 章 编译工具

11.1 GCC 简介

GCC 不仅能支持 C 语言；它现在还支持 Ada 语言、C++ 语言、Java 语言、Objective C 语言、Pascal 语言、COBOL 语言，以及支持函数式编程和逻辑编程的 Mercury 语言，等等。

K-UX 的 gcc 版本为 gcc version 8.3.1 20191121 (K-UX 8.3.1-5) (GCC)

11.2 简单编译

示例如下：

```
1 //test.c
2 #include <stdio.h>
3 int main(void)
4   {
5       printf("Hello World!\n");
6       return 0;
7    }
```

这个程序，一步到位的编译指令是:

```
#gcc test.c -o test
```

实质上，上述编译过程是分为四个阶段进行的，即预处理(也称预编译，Preprocessing)、编译(Compilation)、汇编 (Assembly)和连接(Linking)。

11.2.1 预处理

```
#gcc -E test.c -o test.i
```

可以输出 test.i 文件中存放着 test.c 经预处理之后的代码。gcc 的-E 选项，可以让编译器在预处理后停止，并输出预处理结果。在本例中，预处理结果就是将 stdio.h 文件中的内容插入到 test.c 中。

11.2.2 编译为汇编代码

预处理之后，可直接对生成的 test.i 文件编译，生成汇编代码：

```
#gcc -S test.i -o test.s
```

gcc 的-S 选项，表示在程序编译期间，在生成汇编代码后，停止，-o 输出汇编代码文件。

11.2.3 汇编

对于上一小节中生成的汇编代码文件 test.s，gas 汇编器负责将其编译为目标文件，如下:

```
#gcc -c test.s -o test.o
```

11.2.4 连接

gcc 连接器是 gas 提供的，负责将程序的目标文件与所需的所有附加的目标文件连接起来，最终生成可执行文件。附加的目标文件包括静态连接库和动态连接库。对于上一小节中生成的 test.o，将其与 C 标准输入输出库进行连接，最终生成程序 test。

```
#gcc test.o -o test
```

在命令行窗口中，执行./test。

11.3 多个程序文件的编译

通常整个程序是由多个源文件组成的，相应地也就形成了多个编译单元，使用 GCC 能够很好地管理这些编译单元。假设有一个由 test1.c 和 test2.c 两个源文件组成的程序，为了对它们进行编译，并最终生成可执行程序 test，可以使用下面这条命令：

```
#gcc test1.c test2.c -o test
```

如果同时处理的文件不止一个，GCC 仍然会按照预处理、编译和链接的过

程依次进行。如果深究起来，上面这条命令大致相当于依次执行如下三条命令：

```
#gcc -c test1.c -o test1.o
#gcc -c test2.c -o test2.o
#gcc test1.o test2.o -o test
```

第 12 章 守护进程

K-UX 系统在启动时就启动很多进程(例如:init 进程、等待用户登录的进程 login、等待 FTP 客户连接的 vsftpd 等)，这些进程向本地和网络用户提供了 K-UX 的系统功能接口，直接面向应用程序和用户。将这些进程称为守护进程(daemon)。

守护进程是指在后台运行而又没有终端或登录 shell 与之结合在一起的进程。由于此类程序运行在后台，除非程序异常终止或者人为终止，否则它们将一直运行下去直至系统关闭。一般地，守护进程在系统引导装入时启动，在系统关闭时终止。一个实际运行中的系统一般会有多个这样的守护进程在运行。Windows 系统中的守护进程被称为“服务”。

按照服务类型可以分为如下两类:

系统守护进程 :如 atd、cron、lpd、syslogd、login 等。

网络守护进程 :如 sshd、httpd、sendmail、xinetd 等。

12.1 网络守护进程

在 Client/Server 模型中，服务器监听(Listen)在一个特定的端口上等待客户的连接。连接成功之后客户机与服务器通过端口进行数据通讯。

守护进程的工作就是打开一个端口，并且等待(Listen)进入的连接。如果客户提请了一个连接，守护进程就创建(fork)子进程来响应此连接，而父进程继续监听更多的服务请求。正因为如此，每个守护进程都可以处理多个客户服务请求。

12.2 超级服务器的引入

运行 K-UX 的计算机一般都作为服务器使用，提供了许多不同协议的服务。但是，一台繁忙的服务器也可能是专用于某个任务的，比如传输邮件、响应 DNS 请求等。

从守护进程的概念，我们可以看出，对于系统所要提供的每一种服务，都必须运行一个监听某个端口连接发生的守护程序。无论如何，在提供多种服务的 K-UX 系统中，系统内存中同时运行二、三十种不同的守护进程是对资源的浪费。解决这个问题的方法是使用超级服务器 (SuperServer)。

几乎所有的类 UNIX 系统都运行了一个“超级服务器”，它为众多服务创建套接字(Socket)，并且使用 Socket 系统调用同时监听多个端口。当远程系统请求一个服务时，超级服务器监听到这个请求后会产生该端口的服务器程序为客户提供服务。

使用最广泛的超级服务器程序是 xinetd ，即“扩展网络守护进程”。

xinetd 在运行时读取 /etc 下文本配置文件，在文件中指出超级服务器需要监听的端口以及在数据包到达端口时需要启动的程序。 xinetd 具有更先进的配置模式和更好的安全性。

12.3 守护进程的运行方式

由于引入了超级服务器，因此守护进程有如下两种运行方式:

1. 独立运行的(stand-a lone)守护进程

   a.独立运行的守护进程由 init 脚本负责管理

   b.独立运行的守护进程的脚本存放在 /etc/init.d/ 目录下

   c.所有的系统服务都是独立运行的。如:cron、syslogd 等。

2. 由超级服务器(SuperServer)运行的守护进程

   a.要运行的守护进程由 inetd/xinetd 启动。

   b.由 xinetd 管理的守护进程的配置文件存放在 /etc/xinetd.d/ 目录下，默认的 xinetd 的主配置文件是/etc/xinetd.conf。

   c.inetd/xinetd 本身是独立运行的守护进程。

为了节省资源，引入了超级服务器用于监控网络服务，如 telnet、talk 等。使用超级服务器启动网络服务虽然可以节省资源，但是对于服务量很大的守护进程(如 HTTP 服务、FTP 服务)将影响到其他服务的运行，同时也影响所提供服务的响应速度。为此，某些常用的知名网络服务的守护进程需要单独启动。

12.4 管理守护进程

12.4.1 查看守护进程树

可以使用 pstree 命令查看守护进程树。

pstree 还支持许多参数，请查看 man pstree。

12.4.2 守护进程的启用和停止

在服务器运行期间，可以随时启用和停止守护进程。另外，当修改了守护进程的配置文件之后，也要重新启动守护进程。

独立运行的守护进程的启用和停止

直接运行 /etc/init.d/ 目录下的脚本管理守护进程。

```
# systemctl start|stop|restart|reload server-name
```

例如:

启动 apache 服务，使用如下命令:

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

停止 apache 服务，使用如下命令:

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

重新启动 apache 服务，使用如下命令:

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

重新加载 apache 的配置文件，使用如下命令:

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

12.4.3 使用 systemctl 管理服务

可以使用 systemctl 命令检查、设置系统的各种服务。此命令通过在 Systemd 中使用 service 类型的 unit 文件（.service 文件）对系统的各种服务进行管理。

systemctl 命令具有如下功能:

启动一个服务，使用如下命令:

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

关闭一个服务，使用如下命令:

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

重启一个服务，使用如下命令:

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

显示一个服务的状态，使用如下命令:

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

在开机时启动一个服务，使用如下命令:

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

在开机时禁用一个服务，使用如下命令:

```
#systemctl disable name.service
```

查看服务是否开机启动，使用如下命令:

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

重载服务，使用如下命令:

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

查看已启动的服务列表，使用如下命令:

```
#systemctl list-unit-files | grep enabled
```

查看启动失败的服务列表，使用如下命令:

```
#systemctl --failed
```

重新加载配置文件，使用如下命令:

```
#ystemctl condrestart name
```

列出系统所有服务及状态，使用如下命令:

```
#systemctl list-units
```

12.5 管理子系统

12.5.1 CPUSET 的介绍

CPUSET 是一个集成到内核中的、直接使用内核中的进程调度，负载平衡的底层函数以及 CGROUP 提供的功能的一个工具。它包括两个部分：内核态部分和用户态部分。内核态部分对用户是透明的，在此不做讨论。本文仅讲述用户态部分的内容。

12.5.2 CPUSET 概述

CPUSET 提供一种类似于“容器”的虚拟结构，进程运行在这个“容器”中，并受到这个“容器”的相关属性的限制。在使用 CPUSET 前，需要先用命令创建并命名一个“容器”，其后，可以对这个“容器”进行操作：重命名、修改属性、删除等。在创建容器后，可以对进程进行操作：把进程迁移进入“容器”中，或者把“容器"中的进程迁移走。另外，如果一个进程在某个“容器”中运行时，那么它的所有的子进程都将在这个“容器”中运行，除非显示地以 CPUSET 命令把进程迁移走。

12.5.3 CPUSET 工具程序使用

1、创建一个新的 cpuset 容器：mkdir -p /sys/fs/cgroup/cpuset/test

2、进入新创建的 test 容器中，输入指定的 CPU 号，例如：echo 2 > cpuset.cpus

3、输入指定的使用的节点内存号，例如：echo 0 > cpuset.mems

4、运行一个程序并获取 PID，例如： nohup sha1sum /dev/zero &

5、输入需要绑定的运行进程号，例如：echo 8000 > tasks

描述

CPUSET 提供多种功能，可以用来设置进程的与 CPU 的绑定关系，迁移进程至指定的 CPU 上执行，指定一个程序在某个 CPU 上运行等。

容器中的配置文件作用如下：

1. cpuset.cpus：指定任务绑定的 cpu 号

2. cpuset.mems：指定任务绑定的内存节点号

3. cpuset.cpu_exclusive：布尔值，默认 0，指定当前容器是否独占该 cpu，需要配合 cpus

4. cpuset.mem_exclusive：布尔值，默认 0，指定当前容器是否独占该节点内存，需要配合 mems

5. cpuset.mem_hardwall：布尔值，默认 0，内核为该容器分配的进程是否应该仅仅在指定的内存节点上

6. cpuset.memory_migrate：布尔值，默认 0，指定当内存节点变化是，原内存页面是否迁移到新的内存节点上

7. cpuset.memory_pressure：统计了该容器内存压力的平均值（仅在 cpuset.memory_pressure_enabled 启用时有效）

8. cpuset.memory_pressure_enabled：布尔值，默认 0

9. cpuset.memory_spread_page：布尔值，默认 0，是否均衡使用该容器的内存节点

10. cpuset.memory_spread_slab：布尔值，默认 0，是否均衡使用该容器的 cpu 节点

11. cpuset.sched_load_balance：布尔值，默认 1，是否平均分配该容器的 cpu 负载到该容器的节点上，注意，如果父 group 启用了这项，那么当前项就不在有效。

12. cpuset.sched_relax_domain_level：一个-1 至 5；代表系统试图进行负载均衡的类型（仅在 cpuset.sched_load_balance 启用时有效）

-1：使用系统默认值

0：定期负载均衡

1：实时在同一内核线程间进行负载均衡

2：实时在同一内核包间负载均衡

3：实时在同一 cpu 节点或者刀片上负载均衡

4：实时在多个 CPU（NUMA）节点负载均衡

5：实时在所有 cpu 间负载均衡

12.6 内存管理子系统

12.6.1 针对 NUMA 结构的内存分配

NUMA API 主要任务是管理 NUMA 的内存策略。NUMA 策略通过几个子系统的协同工作来实现。内核管理进程的内存分配机制以及特殊的内存映射。NUMA API 通过新引入的 3 个内核系统调用来实现这一点。在用户空间中，NUMA API 通过 libnuma 库提供了统一的接口供用户空间程序使用。相对于系统调用，libnuma 接口更加清晰易用。同时 NUMA API 还提供了命令行工具 numactl 和 numastat 来帮助系统管理员实现进程级别的策略管理。

在 K-UX 上 NUMA API 支持四种内存分配策略：

1. 缺省(default)：总是在本地节点分配（分配在当前线程运行的节点上）；
2. 绑定(bind)：分配到指定节点上；
3. 交织(interleave)：在所有节点或者指定的节点上交织分配 K-UX；
4. 优先(preferred)：在指定节点上分配失败则在其他节点上分配。

绑定和优先的区别是，在指定节点上分配失败时（如无足够内存），绑定策略会报告分配失败，而优先策略会尝试在其他节点上进行分配。强制使用绑定有可能会导致前期的内存短缺，并引起大量换页。在 libnuma 库中，优先和绑定是组合在一起的。通过对线程调用 uma_set_strict 函数，可以在两种策略间切换。缺省的策略是更加普适的优先策略。

策略可以基于进程或内存区域设定。进程策略对整个进程内的内存分配都有效，而内存区域策略作用于指定的内存区域，其优先级比进程策略要高。

进程策略，作用于所有由内核分配的内存页，包括 malloc， 系统调用中使用的内核级的分配以及文件缓冲区等。唯一的例外是，中断分配的内存总是在当前节点中。当子进程 Fork 时，会继承父进程的进程策略。

内存区域策略，又称为 VMA 策略，它允许一个进程为自己地址空间里的一块内存设置策略。内存区域策略比进程策略具有更高的优先级。它的主要优点在于能够在分配发生前进行设置。目前，内存区策略只支持一部分内存机制，如：SYSV 共享内存，shmem 和 tmpfs 文件映射以及 hugetlbfs 文件系统。在共享内存段或文件映射被删除前，共享内存的区域策略会一直有效。

12.6.2 NUMA 策略控制工具

K-UX 系统提供命令行及编程 API 两级用户空间工具来对策略进行控制。

numactl 是设定进程 NUMA 策略的命令行工具。对于那些无法修改和重新编译的程序，它可以进行非常有效的策略设定。numactl 使管理员可以通过简单的命令行调用来设定进程的策略， 并可以集成到管理脚本中。

numactl 的主要功能包括：

1. 设定进程的内存分配基本策略
2. 限定内存分配范围，如某一特定节点或部分节点集合
3. 对进程进行节点或节点集合的绑定
4. 修改命名共享内存，tmpfs 或 hugetblfs 等的内存策略
5. 获取当前策略信息及状态
6. 获取 NUMA 硬件拓扑

下面是使用 numactl 设定进程策略的实例：

#numactl --cpubind=0 --membind=0，1 program

其意义为：在节点 0 上的 CPU 运行名为 program 的程序，并且只在节点 0，1 上分配内存。cpubind 的参数是节点编号，而不是 cpu 编号。对于每个节点上有多个 CPU 的系统，编号的定义顺序可能会不同。

下面是使用 numactl 更改共享内存段的分配策略的实例：

#numactl --length=1G --file=/dev/shm/interleaved --interleave=all

其意义为：对命名共享内存 interleaved 进行设置，其策略为全节点交织分配，

大小为 1G。

numastat 是获取 NUMA 内存访问统计信息的命令行工具。对于系统中的每个节点，内核维护了一些有关 NUMA 分配状态的统计数据。numastat 命令会基于节点对内存的申请，分配，转移，失败等等做出统计，也会报告 NUMA 策略的执行状况。这些信息对于测试 NUMA 策略的有效性是非常有用的。

12.6.3 libnuma--NUMA 策略的应用程序编程接口

尽管 numactl 能够用作进程级别的内存控制，但其缺点也很明显：分配策略作用于整个进程，无法指定到线程或者特定内存区域。libnuma 为更加精细的控制提供了 API 接口，安装 numactl-devel 包可以创建 numa.h 头文件。

应用程序只需在代码中引用 numa.h 头文件， 并在连接时如下连接 libnuma 的共享库即可方便使用 libnuma：

```
#include <numa.h>

……<代码段>

cc ... -lnuma
```

在开始使用 NUMA API 更改策略或分配内存之前，首先需要调用 numa_available()函数。之后， 则可以使用 libnuma 的接口对进程策略进行更改，或分配内存。libnuma 库的函数包括以下几组：

1. 环境信息：包括一组用于获取系统内存和 CPU 拓扑信息的函数，如系统节点数目，特定 节点的内存大小等等
2. 进程策略：包括一组用于获取，设定和更改进程级策略的函数
3. 内存区域策略：包括一组用于设定特定内存区域策略的函数
4. 节点绑定：将线程绑定到指定节点或节点组的函数
5. 分配函数：忽略当前进程策略，直接使用特定的策略进行分配的一组函数
6. 其他辅助函数

通过使用这些接口，可以非常灵活的配置程序内存分配的方式和策略， 以

达到优化性能的目的。通常的基于 NUMA 的内存分配流程为：

1．用 numa_available()判定系统是否支持 NUMA

2．使用进程策略函数定义进程的整体策略

3．使用节点绑定函数合理绑定线程

4．使用普通的分配函数(如 malloc)进行普通分配

5．对于特定性能需求的代码，使用 NUMA 分配函数做指定分配

6．对于内存区域，使用内存区域策略函数设定其分配策略

第 13 章 重定向和管道

13.1 重定向

K-UX 命令在执行时常常期望接收输入数据,命令执行后又期望将产生的数据结果输出。K-UX 的大部分命令都具有标准的输入/输出设备端口。下表列出了标准设备。

名称	文件描述符	代表意思	设备	说明
STDIN	0	标准输入	键盘	命令在执行时所要的输入数据通过它来取得
STDOUT	1	标准输出	显示器	命令执行后的输出结果从该端口送出
STDERR	2	标准错误	显示器	命令执行时的错误信息通过该端口送出

所谓重定向,就是不使用系统的标准输入端口、标准输出端口或标准错误端口,而进行重新的指定,所以重定向分为输出重定向、输入重定向和错误重定向。通常情况下重定向到一个文件。在 Shell 中,要实现重定向主要依靠重定向符实现,即 Shell 是检查命令行中有无重定向符来决定是否需要实施重定向。下表列出了常用的重定向符。

重定向符	说 明
<	实现输入重定向。输入重定向并不经常使用,因为大多数命令都以参数的形式在命令行上指定输入文件的文件名。尽管如此,当使用一个不接受文件名为输入参数的命令,而需要的输入又是在一个已存在的文件里时,就能用输入重定向解决问题
«!.....!	实现输入重定向的特例,即 here 文件
>或»	实现输出重定向。输出重定向比输入重定向更常用。输出重定向使用户能把一个命令的输出重定向到一个文件里,而不是显示在屏幕上。

	很多情况下都可以使用这种功能。例如,如果某个命令的输出很多,在屏幕上不能完全显示,即可把它重定向到一个文件中,稍后再用文本编辑器来打开这个文件
2>或 2»	实现错误重定向
&>	同时实现输出重定向和错误重定向

下面举几个使用重定向的例子:

1. 将 ls 命令生成的/tmp 目录的一个清单存到当前目录中的 dir 文件中。

```
# ls -l /tmp >dir
```

2. 将 ls 命令生成的/etc 目录的一个清单以追加的方式存到当前目录中的 dir 文件中。

```
# ls -l /etc >> dir
```

3. 将/etc/passwd 文件的内容作为 wc 命令的输入。

```
# wc < /etc/passwd
```

4. 将命令随后输入的文本作为 wc 命令的输入。

```
# wc << !
> this text forms the content of the heredocument ,
> which continues until the end of text delimiter
> !
```

5. 获得 apache 软件包的安装文件清单并存入指定的文件。

```
# rpm -ql apache > apache.list
```

6. .用 echo 命令和输出重定向建立简单的文本文件。

```
# echo “Please call me:62488888”>message
```

7. 利用 cat 命令、here 文档和输出重定向建立简单的文本文件。

```
# cat <<! >mytext
> This text forms the content of the heredocument ,
> which continues until the end of text delimiter
> !
```

8. 将命令 myprogram 的错误信息保存在当前目录下的 err_file 文件中。

```
# myprogram 2> err_file
```

9. 将命令 myprogram 的输出信息和错误信息保存在当前目录下的 output_file 文件中。

```
# myprogram &> output_file
```

13.2 管道

许多 K-UX 命令具有过滤特性,即一条命令通过标准输入端口接受一个文件中的数据,命令执行后产生的结果数据又通过标准输出端口送给后一条命令,作为该命令的输入数据。后一条命令也是通过标准输入端口而接受输入数据。

Shell 提供管道命令“|”将这些命令前后衔接在一起,形成一个管道线,格式为:

```
命令 1 | 命令 2 | .....|命令 n
```

管道线中的每一条命令都作为一个单独的进程运行,每一条命令的输出作为下一条命令的输入。由于管道线中的命令总是从左到右顺序执行的,因此管道线是单向的。

管道线的实现创建了 K-UX 系统管道文件并进行重定向,但是管道不同于 I/O 重定向,输入重定向导致一个程序的标准输入来自某个文件,输出重定向是将一个程序的标准输出写到一个文件中,而管道是直接将一个程序的标准输出与另一个程序的标准输入相连接,不需要经过任何中间文件。

下面举几个使用管道的例子:

以长格式递归的方式分屏显示/etc 目录下的文件和目录列表。

```
# ls -Rl /etc |more
```

1. 分屏显示文本文件/etc/passwd 的内容。

```
# cat /etc/passwd | more
```

2. 统计文本文件/etc/passwd 的行数、字数和字符数。

```
# cat /etc/passwd | wc
```

3. 查看是否存在 zly 用户账户。

```
# cat /etc/passwd | grep zly
```

4. 查看引导信息中关于第 1 块网卡的信息。

```
# dmesg | grep ens0
```

5. 查看系统是否安装了 apache 软件包。

```
# rpm -qa | grep apache
```

6. 解压缩 tar 名为 xyz.tar.gz 的软件包。

```
# gzip -dc xyz.tar.gz | tar -xv
```

7. 以排序方式查看 K-UX 系统中目录的磁盘占据情况。

```
# du -S | sort -n
```

8. 快速移动整个目录。

```
# (cd /source/directory && tar cf - . ) | (cd /dest/directory && tar xvfp -)
```

9. 把 man 的信息存为文本文件。

```
# man bash | col -b > bash.txt
```

13.3 find 命令

13.3.1 find 命令格式

find 命令用于在文件系统中查找满足条件的文件。find 命令功能强大,提供了相当多的查找条件。find 命令还可以对查找到的文件做操作,如执行 Shell 命令等。

find 命令的格式是:

```
find [<起始目录> ...] [<选项表达式>] [<条件匹配表达式>] [<动作表达式>]
```

其中:

<起始目录>:对每个指定的 <起始目录> 递归搜索目录树

若在整个文件系统范围内查找,则起始目录是“/”

若在当前目录下寻找,则起始目录是“.”,省略 <起始目录 >表示当前目录

<选项表达式>:控制 find 命令的行为

<条件匹配表达式>:根据匹配条件查找文件

<动作表达式>:指定对查找结果的操作,默认为显示在标准输出(-print)

不带任何参数的 find 命令将在屏幕上递归显示当前目录下的文件列表。下面给出一些常用的表达式的解释。

13.3.2 选项表达式

表达式	说明
-follow	如果遇到符号链接文件,就跟踪链接所指的文件
-regextype TYPE	指定-regex 和-iregex 使用的正则表达式类型,默认为 emacs,还可选择 posix-awk, posix-basic, posix-egrep 和 posix-extended
-depth	查找进入子目录前先查找完当前目录的文件
-mount	查找文件时不跨越文件系统

-xdev	查找文件时不跨越文件系统
-maxdepth LEVELS	设置最大的查找深度
--help	显示 find 命令帮助信息
--version	显示 find 的版本

13.3.3 条件匹配表达式

表达式	说明
-name PATTERN	匹配文件名
-iname PATTERN -lname PATTERN	匹配文件名(忽略大小写) 匹配符号链接文件名
-ilname PATTERN	匹配符号链接文件名(忽略大小写)
-path PATTERN	匹配文件的完整路径(不把‘/’和‘.’作为特殊字符) PATTERN 使用 Shell 的匹配模式,可以使用 Shell 的通配符(*、?[]),要用“”或‘’括起来
表达式	说明
-regex PATTERN -iregex PATTERN	以正则表达式匹配文件名 以正则表达式匹配文件名(忽略大小写)
表达式	说明
-amin N	查找 N 分钟以前被访问过的所有文件

-atime N	查找 N 天以前被访问过的所有文件
-cmin N -ctime N	查找 N 分钟以前文件状态被修改过的所有文件(比如权限修改) 查找 N 天以前文件状态被修改过的所有文件 (比如权限修改)
-mmin N	查找 N 分钟以前文件内容被修改过的所有文件
-mtime N	查找 N 天以前文件内容被修改过的所有文件
-uid N	查找属于 ID 号为 N 用户的所有文件
-gid N -inum N -links N -size N[bcwkMG]	查找属于 ID 号为 N 组的所有文件 查找 i-node 是 N 的文件 查找硬链接数为 N 的文件 查找大小为 N 的文件,b(块)默 认 单 位 ; c(字节) ; w(双字节)
	N 可以有三种输入方式,+N 或 -N 或 N。假设 N 为 20,则:(1) +20:表示 20 以上(21,22,23 等);(2) -20:表示 20 以内(19,18,17 等);(3) 20:表示正好是 20。
表达式	说明
-perm MODE	精确匹配权限模式为 MODE 的文件。MODE : 与 chown 命令的书写方式一致,既可以使用字符模式也可以使用 8 进制模式
-perm -MODE	匹配权限模式至少为 MODE 的文件
表达式	说明
-anewer FILE	查找所有比 FILE 的访问时间新的文件
-cnewer FILE	查找所有比 FILE 的状态修改时间新的文件(比如权限修改)

-newer FILE	查找所有比 FILE 的内容修改时间新的文件
-samefile FILE	查找与 FILE 具有相同 i-node 的文件 (硬链接)
表达式	说明
-fstype TYPE	只查找指定类型的文件系统
-type [bcdpfls]	查找指定类型的文件 [块设备, 字符设备,目录, 管道, 普通文件, 符号链接, socket 套接字]
-empty	内容为空的文件
-user NAME	查找用户名为 NAME 的所有文件
-group NAME	查找组名为 NAME 的所有文件
-nouser	文件属于不在 /etc/passwd 文件中的用户
-nogroup	文件属于不在 /etc/group 文件中的组

13.3.4 动作表达式

表达式	说明
-print	在标准输出上列出查找结果(每行一个文件)
-print0	在标准输出上列出查找结果(取消间隔符) 同样与 |xargs -0 连用
-fprint FILE	与 -print 一致,只是输出到文件 FILE
-fprint0 FILE	与 print0 一致,只是输出到文件 FILE
-ls	使用 ‘ls -dils’ 在标准输出上列出查找结果
-fls FILE	与 -ls 一致,只是输出到文件 FILE
-prune	忽略对某个目录的查找
-exec COMMAND {} \;	对符合查找条件的文件执行 K-UX 命令

-ok COMMAND {} \;	对符合查找条件的文件执行 K-UX 命令;与 -exec 不同的是,它会询问用户是否需要执行

13.3.5 组合条件表达式

在书写表达式时,可以使用逻辑运算符与、或、非组成的复合条件,并可以用()改变默认的操作符优先级。下面以优先级由高到低列出可用的逻辑操作符。若以空格作为各个表达式的间隔符,则各个表示式之间是与关系。

操作符	说明
(EXPR)	改变操作符优先次序,一些 UNIX 版的 find 命令要使用 \(EXPR \) 形式
! EXPR	表示对表达式取反
EXPR1 EXPR2 EXPR1 -a EXPR2	与逻辑,若 EXPR1 为假,将不再评估 EXPR2 与 EXPR1 EXPR2 功能一致
EXPR1 -o EXPR2	逻辑或,若 EXPR1 为真,将不再评估 EXPR2
EXPR1 , EXPR2	若 EXPR1 为假,继续评估 EXPR2

13.3.6 find 命令使用举例

1. find 的版本和使用帮助信息

```
$ find --help        #显示 find 命令帮助信息
$ find --version    #显示 find 的版本
```

2. 不指定匹配表达式,显示所有文件

```
$ find       #递归显示当前目录的文件列表
$ find /     #递归显示 / 目录的文件列表
$ find / -maxdepth 3     #递归显示 / 目录的文件列表(仅限于 3 层目录)
```

```
$ find / -xdev              #递归显示 / 目录的文件列表(仅限于 / 文件系统)

$ find /home /www /srv      #递归显示 /home、/www、/srv 目录的文件列表
```

3. 按文件名/路径名查找

```
$ find -name myfile      #查找特定的文件名

$ find -name 'd*'      #使用通配符查找特定的文件名

$ find -path '*server'      #匹配文件路径名

$ find -regex '.*'      #以正则表达式匹配文件路径名
```

4.按文件属性查找

```
$ find . -type f      #查找普通文件

$ find . -type l      #查找符号链接文件

$ find /home -links +1      #查找硬连接数大于 1 的文件或目录

$ find /tmp -size -10M      #查找 /tmp 目录下小于 10M 的文件

$ find /home -size +1G      #查找 /home 目录下大于 1G 的文件

$ find / -empty      #查找系统中为空的文件或者目录
```

第 14 章 进程和作业控制

14.1 进程的概念

进程(Process)是一个程序在其自身的虚拟地址空间中的一次执行活动。之所以要创建进程，就是为了使多个程序可以并发的执行，从而提高系统的资源利用率和吞吐量。

进程和程序的概念不同，下面是对这两个概念的比较:

程序只是一个静态的指令集合；而进程是一个程序的动态执行过程，它具有生命期，是动态的产生和消亡的。

进程是资源申请、调度和独立运行的单位，因此，它使用系统中的运行资源；而程序不能申请系统资源、不能被系统调度、也不能作为独立运行的单位，因此，它不占用系统的运行资源。

程序和进程无一一对应的关系。一方面一个程序可以由多个进程所共用，即一个程序在运行过程中可以产生多个进程；另一方面，一个进程在生命期内可以顺序的执行若干个程序。

K-UX 操作系统是多任务的，如果一个应用程序需要几个进程并发地协调运行来完成相关工作，系统会安排这些进程并发运行，同时完成对这些进程的调度和管理任务，包括 CPU、内存、存储器等系统资源的分配。

14.1.1 K-UX 中的进程

在 K-UX 系统中总是有很多进程同时在运行，每一个进程都有一个识别号，叫做 PID(Process ID)，用以区分不同的进程。系统启动后的第一个进程是 systemd，它的 PID 是 1。systemd 是唯一一个由系统内核直接运行的进程。新的进程可以用系统调用 fork 来产生，就是从一个已经存在的旧进程中分出一个新进程来，旧的进程是新产生的进程的父进程，新进程是产生它的进程的子进程，除了 init 之外，每一个进程都有父进程。当系统启动以后，systemd 进程会创建 login 进程等待用户登录系统，login 进程是 systemd 进程的子进程。当用户登录系统后，login 进程就会为用户启动 shell 进程，shell 进程就是 login 进程的子进程，而此

后用户运行的进程都是由 shell 衍生出来的。

除了进程识别号外，每个进程还有另外四个识别号。它们是实际用户识别号(realuser ID)、实际组识别号以及有效用户识别号(effectuser ID)，和有效组识别号(effect group ID)。实际用户识别号和实际组识别号的作用是识别正在运行此进程的用户和组。一个进程的实际用户识别号和实际组识别号就是运行此进程的用户的识别号(UID)和组的识别号(GID)。有效用户识别号和有效组识别号的作用是确定一个进程对其访问的文件的权限和优先权。除了产生进程的程序被设置 UID 位和 GID 位之外，一般有效用户识别号和有效组识别号和实际用户识别号及实际组识别号相同。如果程序被设置了 UID 位或 GID 位，则此进程相应的有效用户识别号和有效组识别号，将和运行此进程的文件的所属用户的 UID 或所属组的 GID 相同。

例如，一个可执行文件/usr/bin/passwd，其所属用户是 root(UID 为 0)，此文件被设置了 UID 位。则当一个 UID 为 500、GID 为 501 的用户执行此命令时，产生的进程的实际用户识别号和实际组识别号分别是 500 和 501，而其有效用户识别号是 0，有效组识别号是 501。

所有这些设计都是为了在一个多用户、多任务的操作系统中，所有用户的工作都能够安全可靠地进行，这也是 K-UX 操作系统的优秀性所在。

14.1.2 进程的类型

可以将运行在 K-UX 系统中的进程分为三种不同的类型:

交互进程:由一个 Shell 启动的进程。交互进程既可以在前台运行，也可以在后台运行。

批处理进程:不与特定的终端相关联，提交到等待队列种顺序执行的进程。

守护进程:在 K-UX 在启动时初始化，需要时运行于后台的进程。

以上三种进程各有各的特点、作用和不同的使用场合。

14.1.3 进程的启动方式

启动一个进程有两个主要途径：手工启动和调度启动。

1. 手工启动:由用户输入命令，直接启动一个进程便是手工启动进程。手工启动进程又可以分为前台启动和后台启动。

I. 前台启动:是手工启动一个进程的最常用的方式。一般地，用户键入一个命令“ls -l”，这就已经启动了一个进程，而且是一个前台的进程。

II. 后台启动:直接从后台手工启动一个进程用得比较少一些，除非是该进程甚为耗时，且用户也不急着需要结果的时候。假设用户要启动一个需要长时间运行的格式化文本文件的进程。为了不使整个 shell 在耗时进程的运行过程中都处于“瘫痪”状态，从后台启动这个进程是明智的选择。在后台启动一个进程，可以在命令行后使用&命令，例如:

```
# ls -R / >list &
```

调度启动方式是事先进行设置，根据用户要求让系统自行启动。

14.2 使用 ps 显示系统进程

K-UX 是一个多用户多任务的操作系统，对多用户的状态查看可以使用 who 命令和 w 命令来进行。要对系统中的进程进行监测和控制，首先就要了解进程的当前运行情况，即查看进程。普通用户和管理员都可以查看系统中正在运行的进程，和这些进程的相关信息。在 K-UX 中，使用 ps 命令对进程进行查看。ps 是一个功能非常强大的进程查看命令。使用该命令使用户可以确定有哪些进程正在执行和执行的状态、进程是否结束、进程有没有僵死、哪些进程占用了过多的系统资源等等。总之，大部分信息都可以通过运行 ps 命令来获得。下面介绍 ps 命令的格式和常用选项。ps 命令的格式如下:

参数	说明
-a	显示终端上的所有进程，包括其他用户的进程。
-e	显示所有进程。
-f	全格式。
-h	不显示标题。
-l	长格式。

-x	显示没有控制终端的进程。

下表列出了 ps 命令输出的重要信息的含义。

输出项	说明
PID	进程号
PPID	父进程的进程号
TTY STAT START	进程从那个终端启动 进程当前状态 进程开始执行的时间
VSZ	进程所占用的虚拟内存的空间，以 kb 为单位
RSS	进程所占用的内存的空间，以 kb 为单位
TIME	进程自从启动以来占用 CPU 的总时间
COMMAND/CMD USER %CPU	进程的命令名 用户名 占用 CPU 时间与总时间的百分比
%MEM	占用内存与系统内存总量的百分比
SIZE	进程代码大小+数据大小+栈空间大小(单位:KB)

其中，在进程状态(STAT)一栏里，表示状态的字符的意义见下表。

字符	说明
R	进程正在执行中(进程排在执行队列里，随时都会被执行)
S	进程处于睡眠状态(sleeping)
T	追踪或停止
Z	僵尸进程(zombie)，进程已经被终止，但其父进程并不知道，没有妥善处理，导致其处于僵尸状态

W	进程没有固定的 pages
<	高优先级的进程
N	低优先级的进程

系统管理员可以从上面的输出获得进程的运行信息。例如:当某个进程占用了过多的 CPU 和 MEM 资源，系统管理员就应该检查该进程是否为一个合法进程。例如:

```
# ps                          # 显示出当前用户在 shell 下所运行的进程
# ps -u osmond                # 只查看用户 osmond 的进程
# ps -aux                     # 列出系统中正在运行的所有进程的详细信息
# ps -auxf                    # 显示系统进程树
```

1. 如果想看清所运行的进程的完整命令行，可以使用 w 参数

2. ps 命令经常同管道命令连用，如下面的形式:

```
# ps -aux|more
# ps -aux|grep httpd
```

14.3 使用 top 命令查看进程状态

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

```
top - 13:45:11 up 35 days,  3:07,  3 users,  load average: 0.01, 0.02, 0.00
Tasks: 717 total,   2 running, 715 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.5%us,  0.4%sy,  0.0%ni, 99.1%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  65948160k total, 44965404k used, 20982756k free,   743760k buffers
Swap:  2047992k total,        0k used,  2047992k free, 33492524k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21873 root      20   0 1900m 166m  21m S  6.3  0.3   1953:00 python
 2205 qemu      20   0 2582m 2.0g 5120 S  4.0  3.3 117:46.34 qemu-kvm
 3575 root      20   0 1007m  16m 5496 S  4.0  0.0   1203:28 libvirtd
24928 qemu      20   0 2643m 2.0g 5116 S  4.0  3.2 115:00.08 qemu-kvm
 4198 qemu      20   0 2642m 2.0g 5068 S  2.3  3.2   1081:26 qemu-kvm
 3399 root      20   0  134m  60m  16m S  0.7  0.1 124:54.26 Xvnc
20227 root      20   0 15432 1792  984 R  0.7  0.0   0:00.09 top
  338 root      39  19     0    0    0 S  0.3  0.0 100:53.83 kipmi0
 2227 root      20   0     0    0    0 S  0.3  0.0   0:05.41 vhost-2205
 2561 root      20   0     0    0    0 S  0.3  0.0   3:52.76 kondemand/14
 4110 qemu      20   0 2604m 1.5g 5064 S  0.3  2.4 214:30.72 qemu-kvm
    1 root      20   0 19360 1540 1228 S  0.0  0.0   0:17.70 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.07 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:27.96 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:08.38 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:04.15 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:23.26 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:12.00 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:03.22 watchdog/1
   11 root      RT   0     0    0    0 S  0.0  0.0   0:17.23 migration/2
   12 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/2
   13 root      20   0     0    0    0 S  0.0  0.0   0:06.15 ksoftirqd/2
   14 root      RT   0     0    0    0 S  0.0  0.0   0:03.13 watchdog/2
   15 root      RT   0     0    0    0 S  0.0  0.0   0:17.81 migration/3
   16 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/3
```

可以看到目前的系统状态，其中比较重要的几个项目如下：

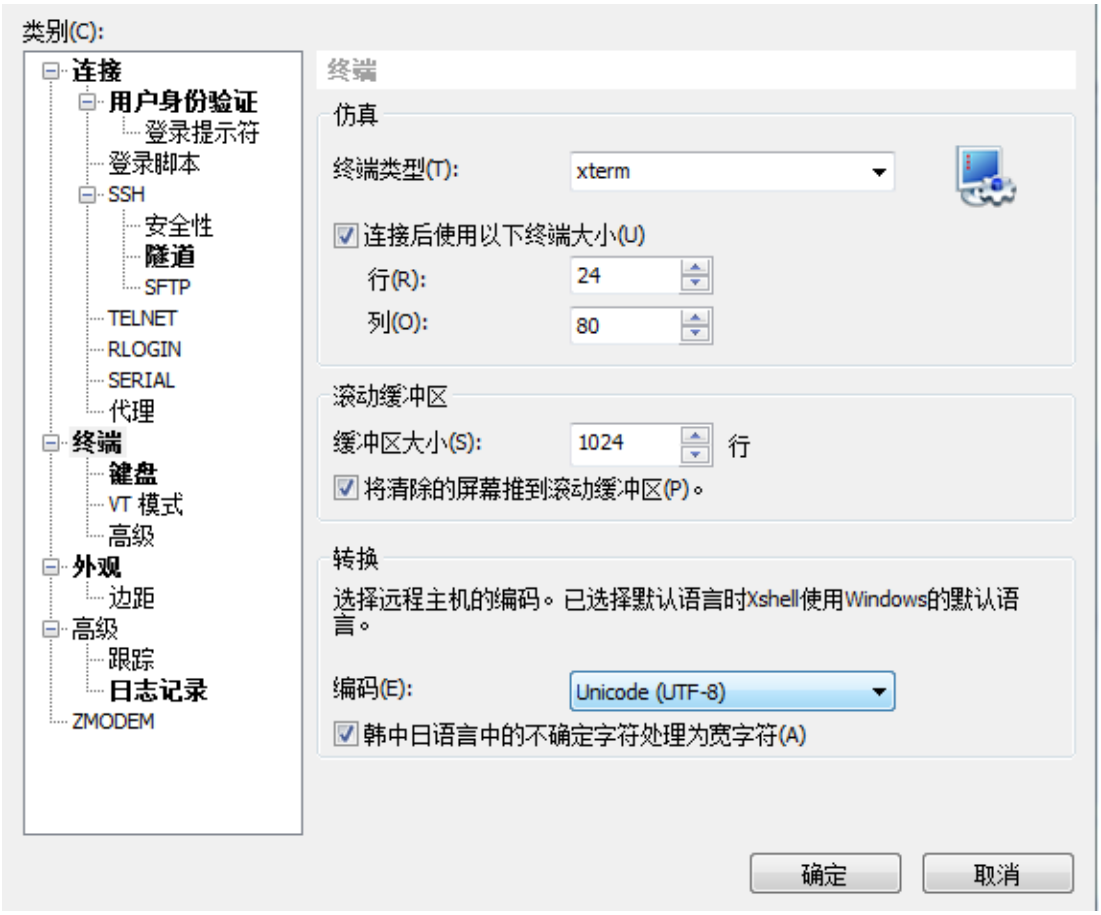
项目	作用
load average	三个数字分别显示了最近不同时间范围的系统负载情况。
Mem	显示了内存使用情况。
Cpu(s)	显示了 CPU 的负载情况。
Swap	显示了 SWAP 空间的使用情况。
	其他内容请查阅 man top

【注意事项】由于服务器 CPU 总数多，在 TOP 命令的输出结果缺省第三行 cpu(s)，键盘按 1 可以再 CPU 总的使用情况与每个 CPU 具体情况之间来回切换。但是当 CPU 个数较多时，按 1 提示 Sorry，terminal is not big enough，如下：

```
top - 22:16:36 up 8 days, 20:05,  6 users,  load average: 0.00, 0.00, 0.00
Tasks: 449 total,   1 running, 448 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  133560144k total, 30711584k used, 102848560k free,  2670304k buffers
Swap: 135561184k total,        0k used, 135561184k free, 23462240k cached
 Sorry, terminal is not big enough
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6037 root      20   0  5120 2816 1952 R  0.3  0.0   0:00.58 top
    1 root      20   0  4496 1680 1360 S  0.0  0.0   0:03.03 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.01 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.30 migration/0
    4 root      15  -5     0    0    0 S  0.0  0.0   0:00.01 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.16 migration/1
    6 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/1
    7 root      RT  -5     0    0    0 S  0.0  0.0   0:00.36 migration/2
    8 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/2
    9 root      RT  -5     0    0    0 S  0.0  0.0   0:00.12 migration/3
   10 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/3
   11 root      RT  -5     0    0    0 S  0.0  0.0   0:00.26 migration/4
   12 root      15  -5     0    0    0 S  0.0  0.0   0:00.07 ksoftirqd/4
   13 root      RT  -5     0    0    0 S  0.0  0.0   0:00.09 migration/5
```

原因：

xshell 等终端是 80 列 24 行的，详见 xshell 文件->属性->终端->连接后使用以下终端大小->行 24，列 80。



CPU 数只要超过 24 个核就报错了，调整行数即可，Putty 没有限制。

除了设置新的行数，还有其他方法：

使用命令 mpstat -P ALL

查看 CPU 逻辑核数：

```
[root@inspur ~]# grep "processor" /proc/cpuinfo |wc -l
32
```

使用命令 mpstat -P　ALL 查看总体及每个 CPU 的性能统计信息：

```
[root@localhost ~]# mpstat -P ALL
Linux 2.6.28.10-vs2.1.4 (localhost.localdomain)         07/01/2015

10:25:33 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal  %idle    intr/s
10:25:33 PM  all    0.21    0.00    0.05    0.00    0.00    0.00    0.00   99.73   3249.84
10:25:33 PM    0    0.15    0.00    0.04    0.00    0.00    0.00    0.00   99.81    100.37
10:25:33 PM    1    0.47    0.00    0.06    0.00    0.00    0.03    0.00   99.43    142.04
10:25:33 PM    2    0.19    0.00    0.05    0.04    0.00    0.01    0.00   99.72    101.10
10:25:33 PM    3    0.29    0.00    0.04    0.00    0.00    0.00    0.00   99.67    100.28
10:25:33 PM    4    0.35    0.00    0.18    0.06    0.00    0.01    0.00   99.39    102.37
10:25:33 PM    5    0.26    0.00    0.06    0.03    0.00    0.01    0.00   99.65    101.10
10:25:33 PM    6    0.15    0.00    0.05    0.00    0.00    0.00    0.00   99.80    101.34
10:25:33 PM    7    0.22    0.00    0.04    0.00    0.00    0.00    0.00   99.74    100.24
10:25:33 PM    8    0.15    0.00    0.04    0.00    0.00    0.00    0.00   99.81    100.25
10:25:33 PM    9    0.36    0.00    0.04    0.00    0.00    0.00    0.00   99.60    100.03
10:25:33 PM   10    0.15    0.00    0.04    0.00    0.00    0.00    0.00   99.81    100.03
10:25:33 PM   11    0.24    0.00    0.03    0.00    0.00    0.00    0.00   99.73    100.03
10:25:33 PM   12    0.15    0.00    0.01    0.00    0.00    0.00    0.00   99.84    100.02
10:25:33 PM   13    0.24    0.00    0.02    0.00    0.00    0.00    0.00   99.73    100.02
10:25:33 PM   14    0.15    0.00    0.02    0.00    0.00    0.00    0.00   99.83    100.02
10:25:33 PM   15    0.24    0.00    0.02    0.00    0.00    0.00    0.00   99.74    100.03
10:25:33 PM   16    0.14    0.00    0.02    0.00    0.00    0.00    0.00   99.84    100.17
10:25:33 PM   17    0.21    0.00    0.03    0.00    0.00    0.00    0.00   99.76    100.04
10:25:33 PM   18    0.24    0.00    0.04    0.00    0.00    0.00    0.00   99.72    100.02
10:25:33 PM   19    0.22    0.00    0.04    0.00    0.00    0.00    0.00   99.74    100.02
10:25:33 PM   20    0.15    0.00    0.06    0.00    0.00    0.00    0.00   99.78    100.03
10:25:33 PM   21    0.21    0.00    0.06    0.00    0.00    0.00    0.00   99.73    100.02
10:25:33 PM   22    0.16    0.00    0.07    0.00    0.00    0.00    0.00   99.77    100.02
10:25:33 PM   23    0.19    0.00    0.07    0.00    0.00    0.00    0.00   99.73    100.02
10:25:33 PM   24    0.15    0.00    0.07    0.00    0.00    0.00    0.00   99.77    100.04
10:25:33 PM   25    0.20    0.00    0.10    0.00    0.00    0.00    0.00   99.70    100.02
10:25:33 PM   26    0.16    0.00    0.02    0.00    0.00    0.00    0.00   99.83    100.02
10:25:33 PM   27    0.21    0.00    0.03    0.00    0.00    0.00    0.00   99.76    100.02
10:25:33 PM   28    0.24    0.00    0.03    0.00    0.00    0.00    0.00   99.73    100.03
10:25:33 PM   29    0.24    0.00    0.05    0.00    0.00    0.00    0.00   99.71    100.03
10:25:33 PM   30    0.17    0.00    0.05    0.00    0.00    0.00    0.00   99.79    100.02
10:25:33 PM   31    0.22    0.00    0.06    0.00    0.00    0.00    0.00   99.72    100.02
```

14.4 调度启动-cron 命令

有时候需要对系统进行一些比较费时而且占用资源的维护工作，这些工作适合在深夜进行，这时候用户就可以事先进行调度安排，指定任务运行的时间或者场合，到时候系统会自动完成这一切工作。要使用自动启动进程的功能。 cron 可

以不断重复的执行一些命令，比如：某任务是每个月 1 日晚上 23:00 要进行数据备份，这时候就需要使用 cron 命令来完成任务了。若想设置一个上述的 cron 备份任务，可以执行：

```
[root@inspur ~]# crontab -e
```

在打开的对话框中输入如下内容：

```
0 23 1 * * backup.sh
~
```

第一项是分钟，第二项是小时，第三项是一个月的第几天，第四项是一年的第几个月，第五项是一周的星期几，第六项是要执行的命令。这些项都不能为空，必须填入。如果用户不需要指定其中的几项，那么可以使用*代替。因为*是统配符，可以代替任何字符，所以就可以认为是任何时间，也就是该项被忽略了。

14.5 进程的挂起及恢复命令 bg、fg

作业控制允许将进程挂起并可以在需要时恢复进程的运行，被挂起的作业恢复后将从中止处开始继续运行。只要在键盘上按 ctrl+z，即可挂起当前的前台作业。 在键盘上按 ctrl+z 后，将挂起当前执行的命令 cat。使用 jobs 命令可以显示 shell 的作业清单，包括具体的作业、作业号以及作业当前所处的状态。恢复进程执行时，有两种选择：用 fg 命令将挂起的作业放回到前台执行；用 bg 命令将挂起的作业放到后台执行。灵活使用上述命令，将给自己带来很大的方便。bg，fg 的帮助文档可以查看 man bg， man fg。

14.6 kill 命令

当需要中断一个前台进程的时候，通常是使用 Ctrl+c 组合键；但是对于一个后台进程恐怕就不是一个组合键所能解决的了，这时就必须求助于 kill 命令。该命令可以终止后台进程。至于终止后台进程的原因很多，或许是该进程占用的 CPU 时间过多； 或许是该进程已经挂死。总之这种情况是经常发生的。kill 命令是通过向进程发送指定的信号来结束进程的。如果没有指定发送信号，那么默认值为 TERM 信号。TERM 信号将终止所有不能捕获该信号的进程。至于那些可以捕获该信号的进程可能就需要使用 kill(9)信号了，该信号是不能被捕捉的。

kill 命令的语法格式很简单，大致命令的格式是:

```
#kill [-signal] PID
```

其中:PID 是进程的识别号；signal 是向进程发出的进程信号，下表列出了一些常用信号的说明。

信号	数值	用途
SIGHUP	1	从终端上发出的结束信号
SIGINT	2	从键盘上发出的中断信号(ctrl+c)
SIGQUT	3	从键盘上发出的退出信号(ctrl+\)
SIGFPE	8	浮点异常(如:被 0 除)
SIGKILL	9	结束接受信号的进程(强行杀死进程)
SIGTERM	15	kill 命令默认的终止信号
SIGCHLD	17	子进程中止或结束的信号
SIGSTOP	19	从键盘来执行的信号(ctrl+d)

要终止一个进程首先要知道它的 PID，这就需要用到上面介绍过的 ps 命令。例如，用户的 xterm 突然停止响应了，无法接受用户的输入，也无法关闭，可以进行如下操作。

(1)找到 xterm 对应的进程的 PID

$ ps aux | grep xterm

osmond 1621 0.0 1.3 6980 1704 tty1 S Aug01 0:01 [xterm]

osmond 1920 0.0 1.9 6772 2544 tty1 S 00:41 0:00 [xterm]

osmond 1921 0.0 0.5 3528 664 pts/1 R 00:41 0:00 grep xterm

(2)杀死进程

$ kill 1621

可以看到用户共启动了两个 xterm ，可以通过两个 xterm 启动的先后顺序来判断哪个进程对应的是要杀死的 xterm ，因为先启动的进程的 PID 总是要小于后启动的进程的 PID。默认情况下，kill 命令发送给进程的终止信号是 15，有些进程会不理会这个信号，这时可以用信号 9 来强制杀死进程，信号 9 是不会被

忽略的强制执行信号。例如，如果上面的命令没有能够杀死 xterm ，可以用信号 9 来结束它。

```
# kill -9 1621
```

14.7 killall 命令

用户也可以用 killall 命令来杀死进程，和 kill 命令不同的是，在 killall 命令后面指定的是要杀死的进程的命令名称，而不是 PID；和 kill 命令相同的是，用户也可以指定发送给进程的终止信号。

例如，要删除所有 apache 进程，可以用如下命令:

killall -9 apache

发送给进程的终止信号可以是信号的号码，也可以用信号的名称。

由于 killall 使用进程名称而不是 PID，所以所有的同名进程都将被杀死。

14.8 作业控制

作业控制是指控制当前正在运行的进程的行为，也称为进程控制。作业控制是 Shell 的一个特性，使用户能在多个独立进程间进行切换。例如，用户可以挂起一个正在运行的进程，稍后再恢复它的运行。bash 记录所有启动的进程并保持对所有已启动的进程的跟踪，在每一个正在运行的进程的生命期内的任何时候，用户可以任意地挂起进程或重新启动进程恢复运行。

例如，当用户使用 vim 编辑一个文本文件，并需要中止编辑做其他事情时，利用作业控制，用户可以让编辑器暂时挂起，返回 Shell 提示符开始做其他的事情。其他事情做完以后，用户可以重新启动挂起的编辑器，返回到刚才中止的地方，就像用户从来没有离开编辑器一样。这只是一个例子，作业控制还有许多其他实际的用途。

下表列出了作业控制的常用命令或操作快捷键。

命令或快捷键	功能说明
cmd &	命令后的&符号表示将该命令放到后台运行，以免霸占终端

<Ctrl+d>	终止一个正在前台运行的进程(含有正常含义)
<Ctrl+c>	终止一个正在前台运行的进程(含有强行含义)
<Ctrl+z>	挂起一个正在前台运行的进程
jobs	显示后台作业和被挂起的进程
bg	重新启动一个挂起的作业， 并且在后台运行
fg	把一个在后台运行的作业放到前台来运行

这些命令经常用于用户需要在后台运行，而却意外地把它放到了前台启动运行的时候。当一个命令在前台被启动运行时，它会禁止用户与 Shell 的交互，直到该命令结束。由于大多数命令的执行都能很快完成，所以一般情况下不会有什么问题。但是如果要运行的命令要花费很长时间的话，我们通常会把它放到后台，以便能在前台继续输入其他命令。此时，上面的命令就会派上用场了。

在进行作业控制时经常使用如下的作业标识符:

作业标识符	说明
%N	第 N 号作业
%S	以字符串 S 开头的被命令行调用的作业
%?S	包含字符串 S 的被命令行调用的作业
%+	默认作业(前台最后结束的作业， 或后台最后启动的作业)，等同于 %%
%-	第二默认作业

14.9 nohup 命令

用途：不挂断地运行命令

语法：nohup command [Arg…] [&]

描述：nohup 命令运行由 command 参数和任何相关的 Arg 参数指定的命令，忽略所有挂断（SIGHUP）信号。在注销后使用 nohup 命令运行后台中的程序。要运行后台中的 nohup 命令，添加 & （ 表示“and”的符号）到命令的尾部。

如果不将 nohup 命令的输出重定向，输出将附加到当前目录的 nohup.out 文件中。如果当前目录的 nohup.out 文件不可写，输出重定向到 $HOME/nohup.out 文件中。如果没有文件能创建或打开以用于追加，那么 Command 参数指定的命令不可调用。如果标准错误是一个终端，那么把指定的命令写给标准错误的所有输出作为标准输出重定向到相同的文件描述符。

退出状态：该命令返回下列值：

126 可以查找但不能调用 Command 参数指定的命令。

127 nohup 命令发生错误或不能查找由 Command 参数指定的命令。

否则，nohup 命令的退出状态是 Command 参数指定命令的退出状态。

nohup 命令及其输出文件 。

nohup 命令：如果你正在运行一个进程，而且你觉得在退出帐户时该进程还不会结束，那么可以使用 nohup 命令。该命令可以在你退出帐户/关闭终端之后继续运行相应的进程。nohup 就是不挂起的意思(no hang up)。

该命令的一般形式为：nohup command &；

使用 nohup 命令提交作业：

如果使用 nohup 命令提交作业，那么在缺省情况下该作业的所有输出都被重定向到一个名为 nohup.out 的文件中，除非另外指定了输出文件：

nohup command > myout.file 2>&1 &

在上面的例子中，输出被重定向到 myout.file 文件中。

使用 jobs 查看任务。

使用 fg %n 关闭。

第 15 章 时钟同步守护进程

15.1 K-UX 的时钟

在 K-UX 中有硬件时钟(Real Time Clock ，简称 RTC)与系统时钟(System Clock)两种时钟。硬件时钟是指主机板上的由电池供电的硬件时钟设备，也就是通常可在 BIOS 画面设定的时钟；系统时钟则是指 kernel 中的时钟。当 K-UX 启动时，系统时钟会去读取硬件时钟的设定，之后系统时钟即独立运作。所有 K-UX 相关指令与函数均读取系统时钟的设定。

15.2 设置系统时钟

使用 date 命令可以查看并设置系统时钟。例如:

```
# date                                    # 查看系统时钟
# date -s "2020-10-30 19:09:00"           # 设置系统时钟
```

15.3 设置硬件时钟

使用 hwclock 命令可以查看并设置硬件时钟。例如:

```
# hwclock                                 # 查看硬件时钟
# hwclock --set --date="30/10/2020 19:09:00"   # 设置硬件时钟
```

15.4 同步系统时钟和硬件时钟

使用 hwclock 命令也可以同步系统时钟和硬件时钟。例如:

```
# hwclock --hctosys # Hardware Clock to System clock
# hwclock --systohc # System clock to Hardware Clock
```

第 16 章 安全登录守护进程

16.1 OpenSSH 和密钥认证协议

管理员时常需要同时管理通过网络相连的分散于各处的多台主机，而类 UNIX 操作系统最大的特色就是可以进行远程登录并进行管理。UNIX 系统很早就有(传统命令：rlogin、rsh、rcp)用于实现远程登录、远程命令执行、远程文件传输的功能。但遗憾的是他们都不安全。

SSH(Secure SHell)协议是 C/S 模式协议，即区分客户端和服务器端。一次成功的 ssh 会话需要两端通力合作来完成。 所有使用 SSH 协议的通讯，包括口令，都会被加密传输。传统的 telnet 和 ftp 之所以不安全，就是因为他们使用纯文本口令，并被明文发送。这些信息可能会被截取，口令可能会被检索，然后未经授权的人员可能会使用截取的口令登录进你的系统而对你的系统造成危害。

与使用传统命令和 telnet 命令最大的不同之处就是 SSH 还添加了密钥认证机制。主机密钥用于识别主机的身份(安装后自动生成，一般无需变更)；用户个人的密钥用于识别用户身份(需要用户自己生成)。

OpenSSH 是 SSH 协议的免费开源实现。它用安全、加密的网络连接工具(ssh、scp、sftp)代替了 telnet、 rlogin、rsh、rcp 和 ftp 等工具。你应该尽可能地使用 OpenSSH 的工具集合来避免这些安全问题。OpenSSH 支持 SSH 协议的版本 1.3、1.5、和 2。自从 OpenSSH 的版本 2.9 以来，默认的协议是版本 2，该协议支持 RSA 和 DSA，默认使用 RSA 密钥。OpenSSH 既支持基于 PAM 的用户口令认证同时也支持用户密钥认证。

16.2 密钥认证协议

OpenSSH 使用非对称密钥的 RSA 和 DSA 认证协议来认证用户。RSA 和 DSA 认证承诺不必提供密码就能够同远程系统建立连接，这是 SSH 的主要魅力之一。

RSA/DSA 密钥认证协议的基本工作原理:

1.密钥由一对组成:一把专用密钥(亦称私钥)和一把公用密钥(亦称公钥)。

2.密钥对由客户端生成，私钥由用户自己保管，并将公钥散播到需要认证之处(登录服务器端)。

3.公钥用于对消息进行加密，只有拥有私钥的人才能对该消息进行解密。

4.公钥只能用于加密，而私钥只能用于对由匹配的公钥编码的消息进行解密。

使用 RSA/DSA 密钥认证协议的 ssh 登录过程:

1.本地主机使用 ssh 客户命令告诉远程主机的 sshd 守护进程想使用 RSA/DSA 认证协议登录。

2.远程主机的 sshd 守护进程会生成一个随机数，并使用存储于该机上的公钥对这个随机数进行加密。

3.远程主机的 sshd 守护进程把加密了的随机数发回给正在本地主机上运行的 ssh 客户。

4.本地主机的 ssh 客户用密钥对中的私钥对这个随机数进行解密后，再把它发回给远程主机的 sshd 守护进程。

5. 远程主机的 sshd 守护进程进行判断，若密钥对匹配，则允许登录。

16.3 OpenSSH 及其相关文件

K-UX 默认安装了 openssh 的客户端和服务器端软件包，您也可以使用 dnf 来安装:

```
# dnf install openssh-client openssh-server
```

OpenSSH 服务器配置文件、客户端系统配置文件和客户端用户配置文件的位置和文件名称:

OpenSSH 服务器配置文件 — /etc/ssh/sshd_config

OpenSSH 客户端系统配置文件 — /etc/ssh/ssh_config

OpenSSH 客户端用户配置文件 — $HOME/.ssh/config

客户命令会依次读取如下条目(优先级由高到低)决定其行为:

命令行参数选项

用户自家目录 $HOME/.ssh/config 配置文件

系统客户配置文件 /etc/ssh/ssh_config

密钥相关文件:

主机密钥相关

/etc/ssh/ssh_host_ed25519_key : 主机 ED25519 认证私钥

/etc/ssh/ssh_host_ed25519_key.pub: 主机 ED25519 认证公钥

/etc/ssh/ssh_host_ecdsa_key : 主机 ECDSA 认证私钥

/etc/ssh/ssh_host_ecdsa_key.pub : 主机 ECDSA 认证公钥

/etc/ssh/ssh_host_rsa_key : 主机 RSA 认证私钥

/etc/ssh/ssh_host_rsa_key.pub : 主机 RSA 认证公钥

用户密钥相关

root/.ssh/id_rsa : 用户默认的 RSA 身份认证私钥

root/.ssh/id_rsa.pub : 用户默认的 RSA 身份认证公钥

root/.ssh/id_ecdsa : 用户默认的 ECDSA 身份认证私钥

root/.ssh/id_ecdsa.pub : 用户默认的 ECDSA 身份认证公钥

root/.ssh/id_ed25519 : 用户默认的 ED25519 身份认证私钥

root/.ssh/id_ed25519.pub : 用户默认的 ED25519 身份认证公钥

root/.ssh/authorized_keys : 用于存放所有已知用户的公钥。

16.4 配置 OpenSSH 服务器

在安装了 openssh-server 软件包之后，默认的配置即可运行良好。

您可以使用 systemctl 命令启动、停止或重启 sshd 守护进程。

```
# systemctl start sshd.service
# systemctl stop sshd.service
# systemctl restart sshd.service
```

16.5 使用 OpenSSH 客户端

OpenSSH 客户端包括 ssh、scp、sftp 命令。有关这些命令的基本使用方法参见 man 手册。下面重点说说用户密钥认证客户配置过程。

1. 服务器方创建目录和文件(若不存在)

```
[root@inspur ~]# mkdir ~/.ssh
[root@inspur ~]# chmod 700 ~/.ssh
```

```
[root@inspur ~]# touch ~/.ssh/authorized_keys
[root@inspur ~]# chmod 600 ~/.ssh/authorized_keys
```

2. 客户端生成密钥对

```
[root@inspur ~]# ssh-keygen -t dsa
Generating public/private dsa key pair.
Enter file in which to save the key (/root/.ssh/id_dsa): /root/.ssh/id_dsa
Enter passphrase (empty for no passphrase):            #输入私钥保护短语
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_dsa.
Your public key has been saved in /root/.ssh/id_dsa.pub.
The key fingerprint is:
10:d5:12:17:ba:6d:81:09:55:30:5a:9f:c2:31:0f:47 root@inspur
The key's randomart image is:
+--[ DSA 1024]----+
|          oo@*E.         |
|           *.&..         |
|          o *.=          |
|           . + .         |
|             S o         |
|               .         |
|                         |
|                         |
|                         |
+-----------------+
[root@inspur ~]# ll .ssh
总用量 16
-rw-------. 1 root root   672 5 月      4 15:18 id_dsa
-rw-r--r--. 1 root root   601 5 月      4 15:18 id_dsa.pub
```

```
-rw-r--r--. 1 root root 5527 4 月    23 15:31 known_hosts
```

3. 客户端生成的公钥分发到服务器端

```
[root@inspur ~]# ssh-copy-id 100.2.127.199
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_ed25519.pub"
The authenticity of host '100.2.127.199 (100.2.127.199)' can't be established.
ECDSA key fingerprint is SHA256:DakuUGXCQxRind+LOmAOw7+HapAp4CXGZnVueJug68s.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s)， to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@100.2.127.199's password:
Number of key(s) added: 1
Now try logging into the machine， with: "ssh '100.2.127.199'"
and check to make sure that only the key(s) you wanted were added.
```

4.密钥认证方式进行远程登录

```
[root@inspur ~]# ssh 100.2.127.199
Activate the web console with: systemctl enable --now cockpit.socket
Last login: Wed Oct 21 22:15:57 2020 from 100.2.93.66
```

若您在使用 ssh-keygen 命令生成密钥对时没有设置私钥保护短语(即直接回车)，则使用 ssh 登录时将直接进入远程系统。这在 cron 任务中包含有 s 族命令、 rsync 等命令时很有用 。

第 17 章 安排周期性任务

17.1 安排周期性任务概述

系统中有两个守护进程 cron 和 anacron 用于自动执行周期性任务。cron 与 anacron 是完全不同的两个用于定期执行任务的守护进程。

cron 假定服务器是 24*7 全天候运行的,当系统时间变化或有一段关机时间就会遗漏这一时间段应该执行的 cron 任务。

anacron(anachronistic cron) 是 cron 的一个连续时间版本,它不会因为时间不连续而导致的任务不执行。

anacron 是针对非全天候运行而设计的,当 anacron 发现时间不连续时,也会执行这一时间段内该执行的任务,这样就不会遗漏计划任务的执行。

每个用户都可以安排自己的 cron 任务。超级用户可以管理系统的 cron 任务和 anacron 任务。

17.2 安排用户自己的周期性任务

/var/spool/cron 目录 cron 守护进程还将搜索 /etc/crontab 文件,这个文件是系统安装时设置好的自动安排的进程任务的 crontab 文件。

17.2.1 cron 简介

cron 守护进程启动以后,它将首先检查是否有用户设置了 crontab 文件,cron 守护进程首先会搜索 /var/spool/cron 目录,寻找以 /etc/passwd 文件中的用户名命名的 crontab 文件,被找到的这种文件将载入内存。例如一个用户名为 osmond 的用户,它所对应的 crontab 文件就应该是 /var/spool/cron/osm ond。也就是说,以该用户命名的 crontab 文件存放在 /var/spool/cron 目录下面。如果 cron 守护进程没有发现相应的 crontab 文件就转入“休眠”状态,释放系统资源,所以该后台进程占用资源极少。cron 守护进程每分钟唤醒一次,当 crontab 中的时间和日期与系统的当前时间和日期相同时,就执行相应的 cron 任务。cron 任务执行结束后,

任何输出都将作为邮件发送给安排 cron 任务的所有者,或者是 crontab 中 MAILTO 环境变量中指定的用户。

cron 守护进程的执行不需要用户干涉;只需要用户安排 crontab 文件,在该文件中要执行的时间和命令序列,下面介绍 crontab 命令。

17.2.2 crontab 命令

每个用户都可以设置自己的 crontab 文件以便执行用户自己需要的自动运行的任务。用户自己的 crontab 文件位于 /var/spool/cron/ 目录,但用户不能直接编辑这些文件,用户必须使用 crontab 命令编辑它。

crontab 命令用于安装、删除或者列出用于驱动 cron 后台进程的 crontab 任务。crontab 的命令格式如下:

```
crontab [-u user] file
或者
crontab [-u user] [-l|-r|-e]
```

下表是 crontab 命令的选项说明。

选项	说明
-u user	指定具体哪个用户的 crontab 文件将被修改。如果不指定该选项,crontab 将默认为是操作者本人的 crontab,也就是执行该 crontab 命令的用户的 crontab 文件将被修改。当使用了 su 命令后执行 crontab 命令就应该指定此参数,以免出现混乱。
file	是一个已经编辑好的 crontab 文件。如果使用“-”号作为文件名,那就意味着使用标准输入。 用于将一个已经编辑好的 crontab 文件装载到 /var/spool/cron/ 目录。
-l	该选项将使在标准输出上显示当前的 crontab。
-r -e	删除当前的 crontab 任务。 使用 VISUAL 或 EDITER 环境变量指定的编辑器编辑当前的 crontab 文件。当结束编辑离开时,编辑后的文件将自动安装。

crontab 文件中的每一行格式为:

```
minute hour day-of-month month-of-year day-of-week [ username] commands
```

每行中都由用空格间隔的七个字段组成。下表说明了各个字段的含义和取值范围。

字段	说明	取值范围
minute	一小时中的哪一分钟	0~59
hour	一天中的哪个小时	0~23
day-of-month	一月中的哪一天	1~31
month-of-year	一年中的哪一月	1~12
day-of-week	一周中的哪一天	0~7(0 和 7 均表示周日)
username	以指定的用户身份执行 commands,省略此字段时表示以安排本任务的用户身份执行 commands。	
commands	执行的命令(可以是多行命令或者是脚本调用)。	

下面重点说明一下前五个时间字段的语法:

a.不能为空,可以使用统配符*表示任何时间。

b.可以指定多个值,它们之间用逗号间隔。例如:1,3,7。

c.可以指定时间段,用减号间隔。例如:0-6。

d.可以用/n 表示步长。例如:8-18/2 表示时间序列 8,10,12,14,16,18。

1. 可以在 crontab 文件中定义并使用环境变量。

2. 如果要执行多条命令可以将其写入一个脚本中,在 commands 字段中只需写脚本文件名即可。

17.2.3 控制安排 cron 任务的人员

是否每个用户都可以安排 cron 任务,需要受如下两个文件的限制:

a./etc/cron.allow

b./etc/cron.deny

1. 当用户每次安排 cron 任务时,系统会先查找 /etc/cron.allow 文件,若该文件存在,则只有包含在此文件中的用户允许使用 cron。

2. 若 /etc/cron.allow 文件不存在,系统继续查找 /etc/cron.deny 文件,若该文件存在,则只有包含在此文件中的用户禁止使用 cron。

3. 有一个例外:无论 root 是否包含在 /etc/cron.allow 文件或 /etc/cron.deny 文件中,root 都可以使用 cron。

/etc/cron.allow 文件和 /etc/cron.deny 文件的格式很简单,每行只能包含一个用户名,且不能有空格字符。

17.3 安排系统的周期性任务

17.3.1 系统的 cron 任务

cron 守护进程在搜索 /var/spool/cron 目录下用户的 crontab 文件的同时,还将搜索/etc/crontab 文件,这个文件是系统安装时设置好的自动安排的进程任务的 crontab 文件。这为系统管理员安排 cron 任务提供了方便。

K-UX 默认的 /etc/crontab 文件的内容为:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

# 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
```

/etc/cron.hourly,/etc/cron.daily,/etc/cron.weekly 和/etc/cron.monthly 是四个目录,分别放置系统每天、每个月、每周和每个小时要执行的任务的脚本文件。

a.通常系统管理员无需编辑 /etc/crontab 文件,只要直接编写 /etc/cron.daily、/etc/cron.monthly、/etc/cron.weekly 和 /etc/cron.hourly 目录下的脚本文件即可。

b.一旦编辑了 /etc/crontab 文件,为了使之立即生效,需要执行如下的命令:

```
# systemctl status crond.service
```

17.3.2 系统的 anacron 任务

anacron 在 /var/spool/anacron 中保留时间戳文件,记录作业运行的时间。当 anacron 运行时,它检查自作业上一次运行以来是否已经经过了所需的天数,如果需要,就运行作业。

anacron 的任务表存储在/etc/anacrontab 文件中。K-UX 默认的 /etc/anacrontab 文件的内容为:

```
# /etc/anacrontab: configuration file for anacron

# See anacron(8) and anacrontab(5) for details.

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
```

此处 /etc/anacrontab 的配置目的是确保每日、每周、和每月的 cron 任务都被运行。

/etc/anacrontab 文件格式与 /etc/crontab 略有不同。与/etc/crontab 一样，/etc/anacrontab 可以包含环境设置。每个任务有四个字段:

```
period    delay    job-identifier    command
周期      延迟      作业标识符       命令
```

说明:

a.周期(period):命令执行的时间间隔(天数)

b.延迟(delay):延迟是在作业符合运行条件之后,到实际启动它之前等待的时间(分钟)。可以使用这个设置防止在系统启动时集中执行作业。

c.作业标识符(job-identifier):任务的描述,用在 anacron 的消息中,并作为作业时间戳文件的名称,只能包括非空白的字符(除斜线外)。

d.命令(command):实际执行的任务。

对于每项任务,anacron 先判定该任务是否已在配置文件的周期字段中指定的期间内被执行了。如果它在给定周期内还没有被执行,anacron 会等待延迟字段中指定的分钟数,然后再次尝试执行命令字段中指定的命令。

当任务完成后,anacron 会将此日期记录在 /var/spool/anacron 目录的时间戳(Time stamp)文件中，默认的时间戳文件有三个:cron.daily,cron.monthly 和 cron.weekly。时间戳文件的记录内容都很简单,下面是作业标识符为 cron.daily 的时间戳文件示例:

```
20300127
```

/etc/crontab 和 /etc/anacrontab 都通过直接编辑进行更新。不使用 crontab 命

令更新这些文件或 /etc/cron.d 目录中的文件。

第 18 章 日志及其查看

日志的主要用途是系统审计、监测追踪和分析统计。

为了保证 K-UX 系统正常运行、准确解决遇到的各种各样的系统问题，认真地读取日志文件是管理员的一项非常重要的任务。

K-UX 内核由很多子系统组成，包括网络、文件访问、内存管理等。子系统需要给用户传送一些消息，这些消息内容包括消息的来源及其重要性等。所有的子系统都要把消息送到一个可以维护的公用消息区，于是，就有了 rsyslog。

syslog 是一个综合的日志记录系统。它的主要功能是:方便日志管理和分类存放日志.syslog 使程序设计者从繁重的、机械的编写日志文件代码的工作中解脱出来，使管理员更好地控制日志的记录过程。在 syslog 出现之前，每个程序都使用自己的日志记录策略。管理员对保存什么信息或是信息存放在哪里没有控制权。syslog 能设置成根据输出信息的程序或重要程度将信息排序到不同的文件。例如，由于核心信息更重要且需要有规律地阅读以确定问题出在哪里，所以要把核心信息与其他信息分开来，单独定向到一个分离的文件中。rsyslog 可以理解为增强版的 syslog，在 syslog 的基础上扩展了很多其他功能，如数据库支持(Mysql，PostgreSQL、Oracle 等)、日志内容筛选、定义日志格式模板等。除了默认的 udp 协议外，rsyslog 还支持 tcp 协议来接收日志。

18.1 rsyslogd 的配置文件

rsyslogd 的配置文件 /etc/rsyslog.conf 规定了系统中需要监视的事件和相应的日志的保存位置。使用如下命令:

```
[root@inspur ~]# cat /etc/rsyslog.conf
```

可以查看此文件的内容为:

```
# rsyslog configuration file
```

```
# For more information see /usr/share/doc/rsyslog-*/rsyslog_conf.html
# If you experience problems，  see http://www.rsyslog.com/doc/troubleshoot.html
#### MODULES ####
# The imjournal module bellow is now used as a message source instead of imuxsock.
$ModLoad imuxsock # provides support for local system logging (e.g. via logger command)
$ModLoad imjournal # provides access to the systemd journal
#$ModLoad imklog # reads kernel messages (the same are read from journald)
#$ModLoad immark  # provides --MARK-- message capability
# Provides UDP syslog reception
#$ModLoad imudp
#$UDPServerRun 514
# Provides TCP syslog reception
#$ModLoad imtcp
#$InputTCPServerRun 514
#### GLOBAL DIRECTIVES ####
# Where to place auxiliary files
$WorkDirectory /var/lib/rsyslog
# Use default timestamp format
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
# File syncing capability is disabled by default. This feature is usually not required，
# not useful and an extreme performance hit
#$ActionFileEnableSync on
# Include all config files in /etc/rsyslog.d/
$IncludeConfig /etc/rsyslog.d/*.conf
# Turn off message reception via local log socket；
# local messages are retrieved through imjournal now.
$OmitLocalLogging on
# File to store the position in the journal
```

```
$IMJournalStateFile imjournal.state
#### RULES ####
# Log all kernel messages to the console.
# Logging much else clutters up the screen.
#kern.*                                                 /dev/console

# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info；mail.none；authpriv.none；cron.none                /var/log/messages

# The authpriv file has restricted access.
authpriv.*                                              /var/log/secure

# Log all the mail messages in one place.
mail.*                                                  -
/var/log/maillog



# Log cron stuff
cron.*                                                  /var/log/cron

# Everybody gets emergency messages
*.emerg                                                 :omusrmsg:*

# Save news errors of level crit and higher in a special file.
uucp，news.crit                                         /var/log/spooler

# Save boot messages also to boot.log
local7.*
```

```
/var/log/boot.log



# ### begin forwarding rule ###
# The statement between the begin ... end define a SINGLE forwarding
# rule. They belong together，  do NOT split them. If you create multiple
# forwarding rules，  duplicate the whole block!
# Remote Logging (we use TCP for reliable delivery)
#
# An on-disk queue is created for this action. If the remote host is
# down，  messages are spooled to disk and sent when it is up again.
#$ActionQueueFileName fwdRule1 # unique name prefix for spool files
#$ActionQueueMaxDiskSpace 1g      # 1gb space limit (use as much as possible)
#$ActionQueueSaveOnShutdown on # save messages to disk on shutdown
#$ActionQueueType LinkedList      # run asynchronously
#$ActionResumeRetryCount -1       # infinite retries if host is down
# remote host is: name/ip:port，  e.g. 192.168.0.1:514，  port optional
#*.* @@remote-host:514
# ### end of the forwarding rule ###
```

该配置文件的每一行的格式如下:

```
facility.priority       action
设备.级别          动作
```

其中:

1. 设备字段用来指定需要监视的事件。它可取的值如下:

设备字段	说明
authpriv	报告认证活动。通常，口令等私有信息不会被记录
cron	报告与 cron 和 at 有关的信息
daemon	报告与 xinetd 有关的信息

kern	报告与内核有关的信息。通常这些信息首先通过 klogd 传送
lpr	报告与打印服务有关的信息
mail	报告与邮件服务有关的信息
mark	在默认情况下每隔 20 分钟就会生成一次表示系统还在正常运行的消息。Mark 消息很像经常用来确认远程主机是否还在运行的“心跳信号”(Heartbeat)。Mark 消息另外的一个用途是用于事后分析，能够帮助系统管理员确定系统死机发生的时间。
news	报告与网络新闻服务有关的信息
syslog	由 syslog 生成的信息
user	报告由用户程序生成的任何信息，是可编程缺省值
uucp	由 UUCP 生成的信息
local0- local7	与自定义程序一起使用
*	*代表除了 mark 之外的所有功能

2. 级别字段用于指明与每一种功能有关的级别和优先级。它可取的值如下:

级别字段	说明
emerg	出现紧急情况使得该系统不可用，有些需广播给所有用户
alert	需要立即引起注意的情况
crit	危险情况的警告
err	除了 emerg、alert、crit 的其他错误
warning	警告信息
notice	需要引起注意的情况，但不如 err、warning 重要
info	值得报告的消息
debug	由运行于 debug 模式的程序所产生的消息

none	用于禁止任何消息
*	所有级别，除了 none

3. 动作字段用于描述对应功能的动作。它可取的值如下:

动作字段	说明
file	指定一个绝对路径的日志文件名记录日志信息
username	发送信息到指定用户，*表示所有用户
device	将信息发送到指定的设备中，如/dev/console
@hostname	将信息发送到可解析的远程主机 hostname，且该主机必须正在运行 syslogd 并可以识别 syslog 的配置文件

syslog 可以为某一事件指定多个动作，也可以同时指定多个功能和级别，它们之间用分号间隔。

18.2 查看日志的工具

日志文件通常存放在/var/log 目录下。在该目录下除了包括 syslogd 记录的日志之外，同时还包含所有应用程序的日志。

为了查看日志文件的内容必须要有 root 权限。日志文件中的信息很重要，只能让超级用户有访问这些文件的权限。

管理员可以使用下面的命令，查看系统中使用的日志文件

```
[root@inspur ~]# ls /var/log/
```

常用的日志文件如表所示：

日志文件	说明
audit/	存储 auditd 审计守护进程的日志目录
conman/	存储 ConMan 串行终端管理守护进程的日志目录
cups/	存储 CUPS 打印系统的日志目录
httpd/	记录 apache 的访问日志和错误日志目录

mail/	存储 mail 日志的目录
news/	存储 INN 新闻系统的日志目录
pm/	存储电源管理的日志目录
ppp/	存储 pppd 的日志目录
prelink/	prelink 的日志目录
samba/ squid/ vbox/	记录 Samba 的每个用户的日志目录 记录 Squid 的日志目录 ISDN 子系统的日志目录
acpid anaconda.*	存储 acpid 高级电源管理守护进程的日志 K-UX 安装程序 anaconda 的日志
boot.log	记录系统启动日志
btmp	记录登陆未成功的信息日志
cron	记录守护进程 crond 的日志
dmesg lastlog	记录系统启动时的消息日志 记录最近几次成功登录的事件和最后一次不成功的登录
maillog messages secure	记录邮件系统的日志 由 rsyslogd 记录的 info 或更高级别的消息日志 由 rsyslogd 记录的认证日志
spooler	由 rsyslogd 记录的 uucp 和 news 的日志
vsftpd.log	记录 vsftpd 的日志
wtmp	一个用户每次登录进入和退出时间的永久记录
yum.log	记录 yum 的日志

18.2.1 查看文本日志文件

绝大多数日志文件是纯文本文件，每一行就是一个消息。只要是在 K-UX 下能够处理纯文本的工具都能用来查看日志文件。可以使用 cat、tac、more 、less、tail 和 grep 进行查看。

下面以 /var/log/messages 为例，说明其日志文件的格式。

a.该文件中每一行表示一个消息，而且都由四个域的固定格式组成:

b.时间标签(Time stamp):表示消息发出的日期和时间。

c.主机名(Hostname):表示生成消息的计算机的名字。

d.生成消息的子系统的名字:可以是“Kernel”，表示消息来自内核或者是进程的名字，表示发出消息的程序的名字。在方括号里的是进程的 PID。

e.消息(Message)，即消息的内容。

18.2.2 查看非文本日志文件

也有一些日志文件是二进制文件，需要使用相应的命令进行读取。

1. lastlog 命令

使用 lastlog 命令来检查某特定用户上次登录的时间，并格式化输出上次登录日志 /var/log/lastlog 的内容。

例如：

```
[root@inspur log]# lastlog
Username         Port     From             Latest
root             pts/0    10.166.14.8      Mon Jan 28 00:11:08 +0800 2030
bin                                        **Never logged in**
daemon                                     **Never logged in**
```

2. last 命令

last 命令往回搜索 /var/log/wtmp 来显示自从文件第一次创建以来登录过的用户。例如:

```
[root@inspur log]# last
```

```
root     ttyS0                          Mon Jan 28 00:35    still logged in
root     pts/0        10.166.14.8       Mon Jan 28 00:35    still logged in
reboot   system boot  2.6.28.10-vs2.1. Mon Jan 28 00:34          (00:02)
root     ttyS0                          Mon Jan 28 00:20 - down    (00:02)
reboot   system boot  2.6.28.10-vs2.1. Mon Jan 28 00:16          (00:06)
root     pts/0        10.166.14.8       Mon Jan 28 00:11 - down    (00:00)
root     ttyS0                          Mon Jan 28 00:10 - down    (00:01)
reboot   system boot  2.6.28.10-vs2.1. Mon Jan 28 00:06          (00:05)
```

3. lastb 命令

lastb 命令搜索 /var/log/btmp 来显示登录未成功的信息。例如:

```
[root@inspur log]# lastb
root     ssh:notty    10.166.14.8       Mon Jan 28 00:35 - 00:35    (00:00)
AT S7=45 ttyS0                          Sat Jan 26 22:25 - 22:25    (00:00)
AT S7=45 ttyS0                          Tue Jan 22 18:18 - 18:18    (00:00)
AT S7=45 ttyS0                          Tue Jan 15 17:57 - 17:57    (00:00)
1*root   ttyS0                          Sun Jan 13 17:50 - 17:50    (00:00)
AT S7=45 ttyS0                          Sun Jan 13 17:49 - 17:49    (00:00)
AT S7=45 ttyS0                          Sun Jan 13 17:36 - 17:36    (00:00)
```

4. who 命令

who 命令查询 wtmp 文件并报告当前登录的每个用户。who 命令的缺省输出包括用户名、终端类型、登录日期及远程主机。例如:

```
[root@inspur log]# who
root     ttyS0        2030-01-28 00:35
root     pts/0        2030-01-28 00:35 (10.166.14.8)
```

18.2.3 命令 less

用法：less <filename> less 命令可以分页的查看文件，可以进行查找。使用“q”来停止查看文件。使用“h”来获得 less 的使用帮助。

18.2.4 命令 tailf

用法： tailf <filename> 这个命令可以显示命令的最后几行，并且可以动态的更新，当日志增加新的内容时， 将继续显示新的输出。 ctrl+c 可以结束显示。

18.3 常见的系统日志

大部分日志记录文件被保存在/var/log 目录中，在这个目录中除了保存系统生成日志之外，还包括一些应用软件的日志文件。当然/var 目录下的其他子目录中也会记录下一些其他种类的日志记录文件。 下面就介绍一些查看日志的方法和日志文件。

18.3.1 dmesg

输入 dmesg 命令可以查看系统引导日志 dmesg 是一个命令，使用 dmesg 命令可以查看最后一次系统引导的引导日志。这个日志文件中保存了模块加载，服务启动，设备改变等信息。通常可以使用如下命令查看

```
[root@inspur ~]# dmesg |more
```

18.3.2 /var/log/messages

/var/log/messages messages 日志是核心系统日志文件。它包含了系统启动时的引导消息，以及系统运行时的其他状态消息。IO 错误、网络错误和其他系统错误都会记录到这个文件中。其他信息，比如某个人的身份切换为 root，也在这里列出。如果服务正在运行，比如 DHCP 服务器，您可以在 messages 文件中观察它的活动。/var/log/messages 是您在做故障诊断时首先要查看的文件。 通常可以使用 tail 命令或者 less 命令查看这个日志。

18.3.3 /var/log/secure

这个文件记录了用户登录的信息。它也能在日志中记录所有通过 sudo 做的

命令操作。

18.4 救援模式

K-UX 进入救援（rescue）模式：系统启动时，选择第二项，救援模式

18.5 单用户模式

启动操作系统，在第一个选项处按“e”进入编辑模式，然后在 tuned_params 后面添加上 rw 和 init=/bin/bash，按下 control + x 启动

等待一段时间后，即进入单用户模式，此时可以进行修改密码操作。

1、首先输入：passwd

2、输入密码

3、再次输入密码

4、运行命令：touch /.autorelabel　　#使 SElinux 生效，密码生效

5、运行命令 exec /sbin/init　#快速启动

第 19 章 备份数据

19.1 备份简介

不管系统多么可靠，总会发生一些意想不到的事情，致使系统数据丢失。因此使用备份来保护数据不丢失是一种非常重要的手段，尤其在系统数据非常重要的时候。经常进行数据备份能够减少偶然破坏造成的损失，保证系统能够从错误中恢复正常运行。

硬件故障、软件损坏、病毒侵袭、黑客骚扰、错误操作以及其他意想不到的原因时时都在威胁着服务器，随时可能使系统崩溃而无法工作。避免损失的行之有效、有时甚至是惟一的办法就是备份！备份是系统管理工作中十分重要的一个环节。

备份就是把一个文件系统或其部分文件存储到另外的介质中，使得通过这些介质中的记录信息可以恢复原有的文件系统或其中的某些文件。

备份数据的过程就是拷贝重要的数据到其他的介质之上，以保证在原始数据丢失的情况下可以恢复数据。一次备份可能是简单的 cp 命令，将一个文件复制到其他目录下，也可能是使用特定的程序将数据流写进一个特定的设备中的复杂过程。

在 K-UX 环境下可以通过各种各样的方法来执行备份。所涉及的技术从非常简单的脚本驱动的方法，到精心设计的商业化软件。备份可以保存到远程网络设备、磁带驱动器和其他可移动存储介质上。

要实施备份，系统管理员要考虑如下几个因素:

1. 选择备份介质
2. 选择备份策略
3. 选择要备份的数据
4. 选择合适的备份工具
5. 选择是否进行远程备份或网络备份
6. 备份的自动化(备份周期和备份文件的存放周期)

19.2 备份策略

一般可以采取三种可用的备份策略:

完全(Full)备份

每隔一段时间对系统进行一次完全的备份，这样在备份时间间隔内，一旦系统发生故障使得数据丢失时，就可以用上一次的备份数据恢复到上一次备份时的情况。

增量(Incremental)备份

首先进行一次完全备份，然后每隔一个较短时间进行一次备份，但仅备份在这个期间更改的内容。这样一旦发生数据丢失，首先恢复到前一个完全备份，然后按日期逐个恢复每天的备份，就能恢复到前一天的情况。这种备份方法比较有效。

差分(Differential)备份

差分备份也称累计备份。这种备份方法与增量备份相似，首先每月进行一次完全备份，然后备份从上次进行完全备份后更改的全部数据文件。一旦发生数据丢失，使用一个完全备份和一个差分备份就可以恢复故障以前的状态。差分备份只需两次恢复，因此它的恢复工作相对简单。

增量备份和差分备份都能以比较经济的方式对系统进行备份。如果系统数据更新不是太频繁的话，可以选用差分备份。如果系统数据更新太快，使每个备份周期后的几次差分备份的数据量相当大，这时候可以考虑增量备份或混用差分备份和增量备份的方式，或者缩短备份周期。

三种备份策略的比较

下表对这三种备份策略做了比较。

备份方式	备份内容	工作量	恢复步骤	备份速度	恢复速度	优缺点
完全备份	全部内容	大	一次操作	慢	很快	占用空间大，恢复快
增量备份	每次修改后	小	多次操作	很快	中	占用空间

	的单个内容					小，恢复麻烦
差分备份	每次修改后的所有内容	中	二次操作	快	快	占用空间较小，恢复快

19.3 确定要备份的数据

K-UX 区别于其他大多数操作系统的一个方面是操作系统和大多数应用程序一次被安装，而 Windows 或者其他 UNIX 系统则是应用程序与操作系统是分开来安装的，首先是安装操作系统，然后才逐渐安装各个应用程序，对于这样的系统，备份整个系统才是必要的，这些操作系统在初次安装时需要花费大量的时间和精力。而对于 K-UX 来说，初次或再次安装一个基本系统(包括绝大多数应用程序)是非常简单和快速的。系统中的大部分内容都是非常稳定的，而不稳定部分主要在:

/etc:包含所有配置文件。

/var:包含系统守护进程(服务)所使用的信息，包括 DNS 配置、DHCP 租期、邮件缓冲文件、默认 HTTP 服务器文件等等。

/srv:包含本地服务文件。

/usr/local:包含那些相对系统来说“本地化”的内容。

/root:根用户的主目录。

/opt:是安装许多非系统文件的地方。

/home :包含所有普通用户的用户主目录。

一般只要备份这几部分就可以了，其余的系统内容可以通过安装盘获得。由于系统数据并不经常发生改变，所以一般只有当系统内容发生变化时才进行。

以上只是粗略地列出了要备份的目录，当然您可以进行更详细的筛选，如:Apache 的配置、postfix 的配置、vsftpd 的配置、MySQL 的配置；网站数据、邮件数据、FTP 站点数据等。

管理员应该针对所选定的要备份的数据实施备份策略、安排备份计划，例如:每月进行一次完全备份，每周进行一次差分备份，每天做一次增量备份。

当备份您的系统时，不要包括如下的文件系统或目录:

/proc 伪文件系统。

/mnt 或 /media 文件系统，除非您确定有从 CD-ROM，软盘，网络共享的文件系统或其他可移动存储设备上备份的特殊需求。

/dev 设备文件目录。因为通常将备份还原到一个已安装的 K-UX 系统，此目录已经存在。

19.4 使用 tar 做备份

19.4.1 tar 命令

tar 是一个已移植到 K-UX 中的经典 UNIX 命令。tar 是 TapeARchive (磁带归档)的缩写，最初设计用于将文件打包到磁带上。它是一个基于文件的命令，它本质上是连续地、首尾相连地堆放文件。

使用 tar 可以打包整个目录树，这使得它特别适合用于备份。归档文件可以全部还原，或从中展开单独的文件和目录。备份可以保存到基于文件的设备或磁带设备上。文件可以在还原时重定向，以便将它们重新放到一个与最初保存它们的目录(或系统)不同的目录(或系统)。tar 是与文件系统无关的，它可以使用在 ext2、ext3、jfs、Reiser 和其他文件系统上。

K-UX 使用 tar 命令进行备份，此时将涉及 tar 命令的一些参数。 tar 命令的完整格式是:

```
tar <operation> [options] <files_to_backup_or_restore>
```

其中:

peration:用于指定 tar 要进行的操作

options:用于指定一系列的选项

files_to_backup_or_restore:用于给出要备份或要恢复的文件或目录名，在指定目录时也包括了这个目录下的子目录。

下表给出了 tar 命令的操作说明。

操作	说明

[-]A	连接多个归档文件为一个归档文件。
[-]c	用于创建一个新的存档文件
[-]x	从归档文件中恢复备份文件。
[-]t	用于列出一个存档文件中的文件名。
[-]u	仅仅添加比存档文件中更新的文件。即，用新增的文件取代原备份文件，如果在归档文件中找不到要更新的文件，则把它追加到备份文件的最后。
[-]d	将归档文件的内容与文件系统上的当前文件作比较。
[-]r	将文件追加到指定的归档文件中。
--delete	从归档文件中删除指定的文件。

下表给出了 tar 命令的常用选项说明。

选项	说明
-f name	使用 name 指定存档文件名或设备名。
-v	列出处理的详细信息。
-z	用 GNU 的 gzip 压缩文件或解压。
-j	用 GNU 的 bzip2 压缩文件或解压。
-C directory	将当前目录切换到 directory。
-M	创建/列出/恢复多卷存档文件，以便在几个备份介质中存放。
-N DATE	指定仅对那些比 DATE 新的文件进行操作。
-p	表示希望保留文件许可权限。
-P	保留文件的绝对路径，即不去掉/。
-w	要求等待用户确认每一个操作。
-W	表示在写入备份内容到备份设备以后再读出来进行验证

	以提高可靠性。
-T filename	从指定的文件中读需要备份或恢复的文件名。
-X filename	不处理给定文件中列出的文件。
--exclude=PATTERN	不处理指定的文件。

19.4.2 使用 tar 备份文件

通常将备份文件存储在单独的分区中，可以是系统本地硬盘中的一个分区，也可以是另外挂载的移动设备中的一个分区。因此，在备份之前，应该创建挂载点目录，并挂载文件系统。

```
[root@inspur ~]# mkdir /backups
[root@inspur ~]# mkdir /backups/logs /backups/last-full
```

指定要备份的文件或目录

1. 使用如下命令备份指定的一个或多个目录到 /backups 目录的一个归档文件

```
[root@inspur ~]# tar -zcvpf /backups/full-backups.tar.gz /home
[root@inspur ~]# tar -zcvpf /backups/full-backups.tar.gz /home /etc
```

2. 使用命令替换生成要备份的目录

下面的命令备份整个 / 系统，除了 mnt、media 、dev、proc、backups 目录和 lost+found 目录。

```
[root@inspur ~]# tar -zcvpf /backups/full-backup.tar.gz -C / \
>$(ls| egrep -v "backups|mnt|media|dev|lost+found|proc")
```

3. 用 exclude 选项剔除无需备份的文件或目录

1) 下面的命令备份整个 / 系统，除了 mnt、media 、dev、proc、backups、var/spool/squid 目录和所有的 lost+found 目录。

```
[[root@inspur ~]#    tar -zcvpf /backups/full-backup.tar.gz -C / \
> --exclude=mnt --exclude=media --exclude=dev --exclude=pro \
> --exclude=backups --exclude=*/lost+found \
```

```
> --exclude=var/spool/squid
```

2) 下面的命令备份 etc、home 、usr/local 和 var/spool(不包括 var/spool/squid) 目录。

```
[root@inspur ~]# tar -zcvpf /backups/full-backup.tar.gz -C / \
> --exclude=var/spool/squid \
> etc home usr/local var/spool
```

4. 将要备份的文件或目录名放入文本文件

```
[root@inspur ~]# tar -zcvpf /backups/full-backups.tar.gz -T whatsbackup.txt
```

1. -T 参数后指定的文件中，不能使用文件通配符

2. 可以使用 ls 或 find 命令生成 whatsbackup.txt 文件

19.4.3 为归档文件名添加时间

在归档文件名中使用带有命令替换的 date 命令 。

1. 添加日期

```
[root@inspur ~]# tar -zcvpf /backups/full-backup_$(date +%F).tar.gz /etc
[root@inspur ~]# ls -l /backups/
-rw-r--r--. 1 root root    10686211 5 月      6 15:29 full-backup_2015-05-06.tar.gz
```

2. 添加日期、小时和分钟

```
[root@inspur ~]# tar -zcvpf /backups/full-backup_$(date +%Y%m%d-%H%M).tar.gz /etc
[root@inspur ~]# ls -l /backups/
-rw-r--r--. 1 root root    10686211 5 月      6 15:31 full-backup_20150506-1531.tar.gz
```

19.4.4 增量备份

1. 使用带 N 选项的 tar 命令实现增量备份

下面的命令将备份 /etc 目录自 2015-05-06 以来修改过的文件

```
[root@inspur ~]# tar -N 2015-05-06 -zcvpf /backups/inc-backup_$(date +%F).tar.gz
```

```
/etc
```

下面的命令将备份 /etc 目录昨天以来修改过的文件

```
[root@inspur backups]# tar -N $(date -d yesterday "+%F") -zcvpf /backups/inc-backup_$(date +%F).tar.gz /etc
```

以下两种书写方式均可，但含义有所不同:

-N yesterday：比昨天的当前时间新的文件，例如:若当前时间为 2:31，则表示自昨天 2:31 以来的新文件

-N $(date -d yesterday ”+%F”)：自昨天 0:00 以来的新文件

2. 用 find 命令获取增量备份的文件列表

使用如下命令找出 n 天(如:7 天)内修改过的文件，生成备份内容的文件列表

```
[root@inspur ~]# find /home /etc /root/ -mtime -7 -print > /backups/logs/inc-backup_$(date +%F).log
```

对指定文件列表中的文件实现增量备份

```
[root@inspur ~]# tar -zcvpf /backups/inc-backup_$(date + %F).tar.gz -T /backups/logs/inc-backup_2015-05-06.log
```

19.4.5 使用 tar 恢复文件

当建立一个归档时， tar 会将文件路径前面的 / (斜线) 去掉。因此，默认恢复文件时将文件释放到当前目录下。 若要将文件释放到指定的目录下，可以使用 -C 选项指定。

1. 建议在释放文件之前先使用 -t 选项替换 -x 选项，进行检查。

2. 一个更安全的方法是在不同的目录释放文件 (例如您的 home 目录)，然后比对确认后，再将释放的文件移动到原始位置。

恢复全部文件

```
[root@inspur ~]# tar -zxvpf /backups/full-backup_2015-05-06.tar.gz -C /
```

恢复指定文件

```
[root@inspur ~]# tar -zxvpf /backups/full-backup_2015-05-06.tar.gz -C / etc/passwd
etc/shadow
```

19.5 使用 cpio 进行备份

cpio 工具像 tar 一样从命令提示行启动程序。与 tar 相比 cpio 更复杂，但是也更为可靠。因为如果一个 tar 文件中某处有一个坏块，就不能对备份文件的其它部分进行访问，而使用 cpio，只有坏块不能被访问。 cpio 创建一个称为 copy-outmode 的备份，备份存档中包含了文件和所有者、时间及访问许可等信息。cpio 具有如下几个优点。 第一，它对数据的压缩要比 tar 命令更有效。 第二，它是为备份任何文件集而设计的（ tar 旨在备份子目录）. 第三，cpio 能够处理跨多个磁带的备份。 第四，cpio 能够跳过磁带上的坏区继续工作，而 tar 却不能。使用 cpio 备份/home/test/rpm 到 rpm.cpio

解 data.cpio 到/home/test/rpm

```
[root@inspur tmp]# find /home/test/rpm |cpio -oc > rpm.cpio
1643 blocks
```

```
[root@inspur tmp]# cpio -im < rpm.cpio
```

如果有磁带机的话，可以使用如下命令备份：

```
[root@inspur ~]# find /home/test/rpm |cpio -oc > /dev/st0
```

把磁带的内容恢复到/data：

```
[root@inspur ~]# cpio -im < /dev/st0
```

19.6 备份策略

定期对系统进行备份是必不可少的，在进行备份之前，首先要选择合适的备份策略，这将决定何时需要备份，以及出现故障时进行恢复的方式。 完全备份：每隔一定时间就对系统进行一次全面的备份，这样在备份间隔期间出现数据丢失等问题，可以使用上一次的备份数据恢复到前次备份时数据状况。这是最基本的备份方式，但是每次都需要备份所有的数据。 完全备份的缺点是它耗费的时间比较多。对于只需恢复单个文件来说；使用它就有些得不偿失。有些时候需要执

行完全备份，有时候却不必。对于一个良好的备份和恢复方案来说，应该知道什么时间需要完全备份，什么时间增量备份会更好。 增量备份：首先进行一次完全备份，然后每隔一个较短时间进行一次备份，但仅仅备份在这个期间更改的内容。使用一个计划来定期地存储关键的信息，如帐目、工作计划，以防止财产的损失。在设计了备份计划之后，应坚持使用。

19.7 网络监控

K-UX 的网络服务功能非常强大，以高效性和灵活性而著称。通过对网络的监控，可以了解服务器的工作情况。

19.8 使用 netstat 监控 TCP/IP 网络连接

在 K-UX 中使用 netstat 命令来监控 TCP/IP 网络。它可以显示内核路由表、活动的网络状态、和每个网络接口的有用的统计数字。下表列出了部分 netstat 的通用命令行参数。要得到更多的信息请参看 netstat 的 man page netstat 的通用命令行参数

参数	说明
-a	显示所有 Internet 连接的有关信息，包括那些正在监听的信息
-i	显示所有网络设备的统计数字
-c	不断显示网络的更新状态。这个参数使 netstat 每秒一次的输出网络状态列表，直到该程序被中断
-n	以数字/原始形式显示远程地址、本地地址和端口信息，而不是解析主机名和服务器
-o	显示计时器的终止时间和每个网络连接的回退（back off）情况
-r	显示内核路由表
-t	只显示 TCP socket 信息，包括正在监听的信息
-u	只显示 UDP socket 信息

-v	显示 netstat 版本信息
-w	显示原始（raw） socket 信息
-x	显示 UNIX 域 socket 信息

下面用几个例子来说明 netstat 的用法，首先使用 netstat 来显示活动的网络连接： 使用不带任何参数的 netstat 命令将在显示器上列出您计算机上活动网络连接的列表下面这个例子是 netstat 命令的默认输出的一部分，可以看出目前该系统在活动的网络连接。

```
[root@inspur ~]# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0    216 inspur:ssh             100.2.93.66:55558       ESTABLISHED
tcp        0      0 inspur:ssh             100.2.93.66:55481       ESTABLISHED
tcp        0      0 inspur:ssh             100.2.93.66:55334       ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  3      [ ]         DGRAM                    11779    /run/systemd/notify
unix  2      [ ]         DGRAM                    11781    /run/systemd/cgroups-agent
unix  26     [ ]         DGRAM                    11791    /run/systemd/journal/dev-log
unix  8      [ ]         DGRAM                    11805    /run/systemd/journal/socket
unix  2      [ ]         DGRAM                    51814    /run/user/0/systemd/notify
unix  2      [ ]         DGRAM                    46518    /run/user/1000/systemd/notify
unix  2      [ ]         DGRAM                    24007    /var/run/chrony/chronyd.sock
unix  2      [ ]         DGRAM                    28397    /run/user/42/systemd/notify
unix  3      [ ]         STREAM     CONNECTED     74647    /run/user/1000/wayland-0
unix  3      [ ]         STREAM     CONNECTED     48768
```

查看本地系统那些 TCP 端口处于监听(Listen)状态，通过这种方法可以知道那些 TCP 服务正在运行，使用命令 netstat -lt。

```
[root@inspur ~]# netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp        0      0 localhost:ipp           0.0.0.0:*               LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 localhost:ipp           [::]:*                  LISTEN
```

netstat 命令输出的各个字段说明：

名称	说明
Proto	连接所使用的协议， TCP 或 DUP
Recv-Q	在这个 scoket 上收到的但还没有被用户程序拷贝的字节数
Send-Q	送到远程主机但还没有被确认的字节数

Local Address	分配给这个连接的外部主机名或端口号。
Foreign Address	分配给这个连接的外部主机名。
State	socket 的状态。它可以是下述状态的一种： ESTABLISTED 本连接完全地建立 SYN_SENT socket 现在正试图对远程主机进行连接 SYN_RECV 正在初始化连接 FIN_WAIT1 Socket 已关闭并且正在等待关闭本连接 FIN_WAIT2 本连接已关闭，Socket 正在等待从远程主机上关闭 TIME_WAIT scocket 是关闭的并且它正在等待远程主机关闭转发信息 COLSED socket 不再使用 CLOSE_WAIT 远程主机已关闭它的连接，本地主机正在等待 socket 关闭 LAST_ACK 远程主机被关闭并且 socket 也是关闭的，本地主机正在等待确认 LISTEN socket 正在侦听到来的连接企图 UNKNOWN 不知道 socket 的状态 User 拥有这个 socket 的用户的登录 ID

第二部分显示活动的 UNIX 域 socket。UNIX 域 socket 是一个 IPC（进程间通讯）机制，它把 UNIX 文件系统作为约会系统。各进程在文件系统中建立特殊的文件，然后由想要通讯的机器上的其它进程打开这些特殊文件。

检查内核路由表 使用带-r 选项的 netstat 将在显示器上输出内核路由表，格式与 route 命令相同。

```
[root@inspur ~]# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         _gateway        0.0.0.0         UG        0 0          0 enp0s1
100.2.126.0     0.0.0.0         255.255.254.0   U         0 0          0 enp0s1
```

显示网络接口统计数据 在命令行中输入 netstat -i 将在显示器上输出每个活动的网络接口的使用情况的统计数据。 使用这个命令很容易看到何时分组被丢掉了、何时分组超时了等等。下面是一个使用 netstat -i 的例子，而下表解释了其

中的每一个字段：

```
[root@inspur ~]# netstat -i
Kernel Interface table
Iface             MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
enp0s1           1500  1876135      0  37471 0         27573      0      0      0 BMRU
enp0s4           1500    13164      0      0 0             0      0      0      0 BMRU
enp0s5           1500  1854359      0  37471 0             0      0      0      0 BMRU
lo              65536      866      0      0 0           866      0      0      0 LRU
```

内核接口表中字段的说明

字段	说明
iface	网络接口名
MTU	该接口在一次传输时所能传输的最大字节数
Met	这个接口的度量值
RX-OK	接收到的无错误的分组的数量
RX-ERR	接收到的有错误的分组数量
RX-OVR	有超时错误的分组数量
TX-OK	无错传输的分组数量
TX-ERR	有错传输的分组数量
TX-DRP	在传输间丢失的分组数量
TX-OVE	由于超时而丢失的分组数量
Flags	Flags 字段中将会有下面这些标志 A 这个接口接收多点传送地址的分组。
	B 这个接口接收广播分组。
	D 这个接口的调试特性现在是活动的。
	L 这是回送接口。
	M 这个接口处于混合模式。

19.9 网络配置

19.9.1 设置网络参数

可以使用 ifconfig 命令来配置并查看网络接口的配置情况。ifconfig 命令格式是:

#ifconfig <网络接口> <IP 地址> [<netmask 子网掩码> <broadcast 广播地址>]

例如:要配置 enp0s1 的网络参数，可以使用下面的命令。

```
# ifconfig enp0s1 192.168.0.222
```

此命令将启动 enp0s1 接口，并设置其 IP 地址为 192.168.0.222，子网掩码为 255.255.255.0，广播地址为 192.168.0.255。

当 IP 地址使用标准 A、B、C 类地址时，广播地址和子网掩码可以省略，系统会自动判断广播地址和子网掩码的值并进行设置。否则必须指出广播地址和子网掩码，例如:

```
# ifconfig enp0s1  10.0.0.222 mask 255.255.255.0 broadcast 10.0.0.255
```

也可以用 ifconfig 命令配置 enp0s1 别名设备，为 enp0s1 绑定多个 IP 地址。例如:

```
# ifconfig ens0:0 192.168.0.250

# ifconfig ens1:0 192.168.1.3

# ifconfig ens1:1 192.168.2.3
```

使用 ifconfig 命令设置网络参数会立即生效，但不会修改网络接口配置文件，这将导致所配置的参数在重新启动系统后失效。

19.9.2 网络接口的启用和停用

可以使用 ifconfig 命令来启用和停用网络接口， 命令格式是:

```
# ifconfig <网络接口> up

# ifconfig <网络接口> down
```

例如:要启用 enp0s1 并停用 enp0s1 ，可以使用下面的命令。

```
# ifconfig enp0s1 up

# ifconfig enp0s1 down
```

19.9.3 查看网络参数配置

可以使用 ifconfig 命令查看当前的网络参数配置。例如:

```
# ifconfig ens0   # 查看指定的网络接口
```

```
[root@inspur ~]# ifconfig enp0s1
enp0s1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 100.2.127.187  netmask 255.255.254.0  broadcast 100.2.127.255
        inet6 fe80::a491:3b1d:cdc1:ffe8  prefixlen 64  scopeid 0x20<link>
        ether 56:6f:35:92:00:10  txqueuelen 1000  (Ethernet)
        RX packets 162991  bytes 10235110 (9.7 MiB)
        RX errors 0  dropped 3195  overruns 0  frame 0
        TX packets 825  bytes 316896 (309.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
# ifconfig -a     #查看所有网络接口 （-a 参数可以省略）
```

```
[root@inspur ~]# ifconfig -a
enp0s1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 100.2.127.187  netmask 255.255.254.0  broadcast 100.2.127.255
        inet6 fe80::a491:3b1d:cdc1:ffe8  prefixlen 64  scopeid 0x20<link>
        ether 56:6f:35:92:00:10  txqueuelen 1000  (Ethernet)
        RX packets 167795  bytes 10537319 (10.0 MiB)
        RX errors 0  dropped 3278  overruns 0  frame 0
        TX packets 840  bytes 318574 (311.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 56:6f:35:92:00:11  txqueuelen 1000  (Ethernet)
        RX packets 1188  bytes 182943 (178.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp0s5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 56:6f:35:92:00:12  txqueuelen 1000  (Ethernet)
        RX packets 167010  bytes 10473943 (9.9 MiB)
        RX errors 0  dropped 3278  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 408  bytes 34896 (34.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 408  bytes 34896 (34.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

ifconfig 命令显示信息说明:

输出项目	说明

Link encap	网络接口类型，如以太网或 PPP 等
HWadrr	网卡的 Mac 地址。每一块网卡都有自己的编号，用于在以太网协议下定位网络主机
inet addr	此接口对应的 IP 地址
网络接口状态标志	UP — 网络接口被启用
	RUNNING — 接口正在运行
	BROADCAST — 支持广播 IP 寻址方式
	MULTICAST — 支持多播 IP 寻址方式
	LOOPBACK — 表示本地回环设备接口
MTU	Message transfer unit， 此接口所能传输的最大 frame 数
Metric	此接口的 Metric 数，用于引导路由决策
Bcast	广播地址，通常是网络的最后一个 IP 地址
Mask	子网掩码
RX packets	接收的封包总数、错误数、遗失数和溢流数
TX packets	发送的封包总数、错误数、遗失数和溢流数
collisions	冲突数(当多个 NIC 同时使用网线传输数据时会产生冲突)
txqueuelen	指出网络接口可以存储的数据包的个数
RX bytes	与 RX packets 类似，表示接收的具体字节数
TX bytes	与 TX packets 类似，表示发送的具体字节数
Interrupt	网卡使用的中断(IRQ)
Base address	网卡使用的内存地址

19.9.4 直接修改配置文件配置以太网

K-UX 在 /etc/sysconfig/network-scripts 目录下存储网络接口配置文件。每个网络接口有各自的配置文件，配置文件以 ifcfg- 为前缀后接网络接口名。例如，接口 ens0 的配置文件名为 ifcfg-enp0s1 。下面是 ens0 接口的配置文件。

cat /etc/sysconfig/network-scripts/ifcfg-enp0s1

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=enp0s1
UUID=b2863d35-a8c6-46d0-90a9-ac2d54db072b
DEVICE=enp0s1
ONBOOT=yes
IPADDR=100.2.127.187
GATEWAY=100.2.126.1
NETMASK=255.255.254.0
```

您可以根据需要修改此配置文件改变 enp0s1 的配置。要设置 enp0s4 的配置文件，您可以复制 ifcfg-enp0s1 为 ifcfg-enp0s4 然后做适当修改。下面给出一个为 ifcfg-enp0s1 设置动态获得 IP 地址的配置文件。

vim /etc/sysconfig/network-scripts/ifcfg-enp0s1

```
TYPE=Ethernet
DEVICE=enp0s1
HWADDR=00:0c:29:f1:15:8f
BOOTPROTO=dhcp
```

配置文件修改完毕，必须使用如下命令重新启动网络服务，使配置生效。

```
# systemctl restart network.service
```

19.9.5 nmcli 基础

K-UX 系统的网络服务是由 NetworkManager 提供（简称 NM），这是动态控制及配置网络的守护进程，它用于当前网络设备及连接处于工作状态，同时也支持传统的 ifcfg 类型的配置文件。nmcli 是命令行的 NetworkManager 工具，自动把配置写到/etc/sysconfig/network-scripts/目录下面。

1. 基本命令及解析

命令	说明
nmcli	查看所有 IP
nmcli device status	查看左右接口的简略信息
nmcli device show	查看所有接口的详细信息
nmcli device show interface-name	查看特定接口的详细信息
nmcli connection show	显示所有连接的简略信息
nmcli connection show --active	显示激活状态的连接
nmcli connection show connection-name	显示一个网卡的详细信息
nmcli connection up connection-name	激活一个网卡
nmcli connection down connection-name	关闭一个网卡

2. 创建动态获取 ip 地址的连接

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

说明：add 表示添加连接，type 后面是指定创建连接时候必须指定的类型。类型有很多，可以通过 nmcli c add type -h 看到，con-name 指创建连接的名字，ifname 后面是指定物理设备，网络接口。

例子：nmcli connection add type ethernet con-name ifcfg-enp0s5 ifname enp0s5

3. 创建静态 ip 地址连接

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

```
name ipv4.method manual ipv4.addresses address ipv4.gateway address
```

说明：ipv4.addresses 后面指定网卡 ipv4 的地址，ipv4.gateway 后面指定网卡的 ipv4 网关。

例子：nmcli connection add type ethernet con-name ifcfg-enp0s5 ifname enp0s5 ipv4.method manual ipv4.addresses 100.2.127.188/23 ipv4.gateway 100.2.127.1

4. 修改连接配置

```
# nmcli connection modify connection-name ipv4.addresses 100.2.127.188
```

说明：上述命令为添加一个 ip 地址，如果已经存在 ip 会更改现有 ip

```
# nmcli connection modify connection-name ipv4.addresses 100.2.127.188/23
```

说明：上述命令给一个网卡添加一个子网掩码（NETMASK）

```
# nmcli connection modify connection-name ipv4.method manual
```

说明：ip 获取方式设置为手动（BOOTPROTO=static/none）

```
# nmcli connection modify connection-name ipv4.method auto
```

说明：ip 获取方式设置为自动（BOOTPROTO=dhcp）

```
# nmcli connection modify connection-name ipv4.dns 114.114.114.114
```

说明：添加 DNS

```
# nmcli connection modify connection-name -ipv4.dns 114.114.114.114
```

说明：删除 DNS

```
# nmcli connection modify connection-name ipv4.gateway 100.2.127.1
```

说明：添加一个网关（GATEWAYl）

```
# nmcli connection modify connection-name connection.autoconnect no/on
```

说明：修改连接是否随开机激活

19.9.6 设置本地主机名

临时修改主机名，可以使用如下的命令:

hostname INSPUR

永久修改主机名，编辑 /etc/sysconfig/network 文件中的如下配置行:

```
HOSTNAME=yourhostname # 将 yourhostname 修改为您的主机名
```

配置文件修改完毕，在下次重新启动时就会生效。

不要忘记还需要修改 /etc/hosts 文件中的主机名。

19.9.7 设置 DNS 客户和本地主机解析

设置 DNS 客户

DNS 客户端配置文件为/etc/resolv.conf，使用如下命令添加 DNS 服务器解析的指向。

```
# echo "nameverver 208.67.222.222" > /etc/resolv.conf

# echo "nameverver 208.67.220.220" >> /etc/resolv.conf
```

修改 "Hosts 表" 实现静态 DNS 解析

要实现域名解析，即可以使用 DNS 服务器，也可以使用/etc/hosts ，使用如下的命令修改:

vim /etc/hosts

```
127.0.0.1          localhost localhost.localdomain localhost
  4 ::1            localhost6.localdomain6 localhost6
```

19.9.8 路由表和静态路由

查看 K-UX 内核路由表

使用下面的 route 命令可以查看 K-UX 内核路由表。

route

Destination	Gateway	Genmask	Flags	Metric	Ref	Use
Iface						
169.254.0.0	*	255.255.0.0	U	0	0	0
ens1						
10.152.0.0	*	255.255.0.0	U	0	0	0

ens1						
default	localhost	0.0.0.0	UG	0	0	0
ens1						

route 命令的输出项说明

输出项	说明
Gateway	网关地址，”*” 表示目标是本主机所属的网络，不需要路由
Genmask	网络掩码
Flags	标记。一些可能的标记如下:
	U — 路由是活动的
	H — 目标是一个主机
	G — 路由指向网关
	R — 恢复动态路由产生的表项
	D — 由路由的后台程序动态地安装
	M — 由路由的后台程序修改
	! — 拒绝路由
Metric	路由距离，到达指定网络所需的中转数(K-UX 内核中没有使用)
Ref	路由项引用次数(K-UX 内核中没有使用)
Use	此路由项被路由软件查找的次数
Iface	该路由表项对应的输出接口

19.9.9 配置静态路由

1. route 命令

设置和查看路由表都可以用 route 命令，设置内核路由表的命令格式是:

#route [add|del] [-net|-host] target [netmask Nm] [gwGw] [[dev] If]

其中：

参数	说明
add	添加一条路由规则
del	删除一条路由规则
-net	目的地址是一个网络
-host	目的地址是一个主机
target	目的网络或主机
netmask	目的地址的网络掩码
gw	路由数据包通过的网关
dev	为路由指定的网络接口

route 命令使用举例

添加到主机的路由：

```
# route    add -host 192.168.1.2 dev ens0
```

添加到网络的路由：

```
# route    add -net 10.20.30.40 netmask 255.255.255.248 ens0
```

添加默认路由：

```
# route add default gw 192.168.1.1
```

删除路由：

```
# route del -host 192.168.1.2 dev ens0
```

设置包转发

在 K-UX 中默认的内核配置已经包含了路由功能，但默认并没有在系统启动时启用此功能。开启 K-UX 的路由功能可以通过调整内核的网络参数来实现。要配置和调整内核参数可以使用 sysctl 命令。例如:要开启 K-UX 内核的数据包

转发功能可以使用如下的命令。

sysctl -w net.ipv4.ip_forward=1

这样设置之后，当前系统就能实现包转发，但下次启动计算机时将失效。为了使在下次启动计算机时仍然有效，需要将下面的行写入配置文件/etc/sysctl.conf。

vim /etc/sysctl.conf

```
net.ipv4.ip_forward = 1
```

用户还可以使用如下的命令查看当前系统是否支持包转发。

sysctl net.ipv4.ip_forward

第 20 章 NFS 服务

20.1 NFS 简介

NFS 是 Network File System 的缩写，即网络文件系统。一种使用于分散式文件系统的协定，功能是通过网络让不同的机器、不同的操作系统能够彼此分享个别的数据，让应用程序在客户端通过网络访问位于服务器磁盘中的数据，是在类 Unix 系统间实现磁盘文件共享的一种方法。

NFS 的基本原则是“容许不同的客户端及服务端通过一组 RPC 分享相同的文件系统”，它是独立于操作系统，容许不同硬件及操作系统的系统共同进行文件的分享。

NFS 在文件传送或信息传送过程中依赖于 RPC 协议。RPC，远程过程调用 (Remote Procedure Call) 是能使客户端执行其他系统中程序的一种机制。NFS 本身是没有提供信息传输的协议和功能的，但 NFS 却能让我们通过网络进行资料的分享，这是因为 NFS 使用了一些其它的传输协议。而这些传输协议用到这个 RPC 功能的。可以说 NFS 本身就是使用 RPC 的一个程序。或者说 NFS 也是一个 RPC SERVER。所以只要用到 NFS 的地方都要启动 RPC 服务，不论是 NFS SERVER 或者 NFS CLIENT。这样 SERVER 和 CLIENT 才能通过 RPC 来实现 PROGRAM PORT 的对应。可以这么理解 RPC 和 NFS 的关系：NFS 是一个文件系统，而 RPC 是负责负责信息的传输。

20.2 安装 NFS 服务

NFS 的安装是非常简单的，只需要两个软件包即可，而且在通常情况下，是作为系统的默认包安装的。

a.nfs-utils-* ： 包括基本的 NFS 命令与监控程序

b.portmap-* ： 支持安全 NFS RPC 服务的连接

1. 首先查看 K-UX 系统是否已安装 NFS。

```
[root@localhost home]# rpm -qa | grep nfs
libnfsidmap-2.3.3-31.kux.ppc64le
nfs-utils-2.3.3-31.kux.ppc64le
nfs4-acl-tools-0.3.5-3.kux.ppc64le
```

当前系统默认已安装了 nfs-utils libnfsidmap nfs4-acl-tools 三个软件包。

2. 如果当前系统中没有安装 NFS 所需的软件包，需要使用 dnf 进行安装。

dnf install nfs-utils nfs4-acl-tools libnfsidmap -y

20.3 NFS 服务器的配置

NFS 服务器的配置相对比较简单，只需要在相应的配置文件中进行设置，然后启动 NFS 服务器即可。

NFS 的常用目录：

/etc/exports	NFS 服务的主要配置文件
/usr/sbin/exportfs	NFS 服务的管理命令
/usr/sbin/showmount	客户端的查看命令
/var/lib/nfs/etab	记录 NFS 分享出来的目录的完整权限设定值

NFS 服务的配置文件为 /etc/exports，这个文件是 NFS 的主要配置文件，不过系统并没有默认值，所以这个文件不一定会存在，可能要使用 vim 手动建立，然后在文件里面写入配置内容。

/etc/exports 文件内容格式：

```
<输出目录> [客户端 1 选项（访问权限，用户映射，其他）] [客户端 2 选项（访问权限，用户映射，其他）]
```

a. 输出目录：

输出目录是指 NFS 系统中需要共享给客户机使用的目录；

b. 客户端：

客户端是指网络中可以访问这个 NFS 输出目录的计算机

客户端常用的指定方式

- 指定 ip 地址的主机：192.168.0.200
- 指定子网中的所有主机：192.168.0.0/24 192.168.0.0/255.255.255.0

- 指定域名的主机：david.bsmart.cn
- 指定域中的所有主机：*.bsmart.cn
- 所有主机：*

c.选项：

选项用来设置输出目录的访问权限、用户映射等。

20.4 NFS 服务器的启动和停止

在对 exports 文件进行了正确的配置后，就可以启动 NFS 服务器。

1. 启动 NFS 服务器。为了使 NFS 服务器能正常工作，需要启动 rpcbind 和 nfs 两个服务，并且 rpcbind 一定要先于 nfs 启动。

```
# systemctl start rpcbind.service
# systemctl start nfs-server.service
```

2. 查询 NFS 服务器状态。

```
#systemctl status nfs-server.service
```

3. 停止 NFS 服务器。要停止 NFS 运行时，需要先停止 nfs 服务再停止 portmap 服务，对于系统中有其他服务(如 NIS)需要使用时，不需要停止 rpcbind 服务

```
# systemctl stop rpcbind.service
# systemctl stop nfs-server.service
```

4. 设置 NFS 服务器的自动启动状态。

对于实际的应用系统，每次启动系统后都手工启动 nfs 服务器是不现实的，需要设置系统在指定的运行级别自动启动 rpcbind 和 nfs 服务。

```
# systemctl list-unit-files |grep rpcbind
# systemctl list-unit-files |grep nfs-server
```

设置 rpcbind 和 nfs 服务开机启动。

```
# systemctl enable rpcbind.service
# systemctl enable nfs-server.service
```

第 21 章 Chrony 服务

21.1 Chrony 服务简介

Chrony 是网络时间协议（NTP）的通用实现，它可以将系统时钟与 NTP 服务器、参考时钟（例如 GPS 接收器）和使用手表和键盘的手动输入同步。它还可以作为 NTPv4（RFC 5905）服务器和对等服务器运行，为网络中的其他计算机提供时间服务，它被设计成在各种条件下都能很好的运行，包括断续的网络连接，严重拥挤的网络、不断变化的温度（普通的计算机时钟对温度非常敏感），以及不断运行或者在虚拟机上运行的系统。

21.2 Chrony 安装

默认情况下 K-UX 中已经安装好 chrony 软件包，可以通过 rpm -q chrony 确认是否存在，如果不存在则使用下边的命令安装即可。

```
# dnf  install  chrony  -y
```

21.3 相关配置文件

/etc/chrony.conf ： chrony 服务的主要配置文件，所有的更改全部在这里。

/usr/bin/chronyc ： chronyc 是一个命令行交互式接口程序，可用于监视 chronyd 的性能，并在运行时更改各种操作参数。

/usr/sbin/chronyd ： chronyd 是一个可以在启动时启动的守护程序。

/usr/share/zoneinfo ：由 tzdata 所提供，规定了各主要时区的时间设定文件，例如中国的时区设置文件是/usr/share/zoneinfo/Asia/Chongqing。

/etc/sysconfig/clock：K-UX 的主要时区设定文件。每次启动后操作系统会自动读取这个文件来设定系统预设要显示的时间。如这个文件内容为”ZONE=Asia/Chongqing”，则表示 K-UX 操作系统的时间设定使用/usr/share/zoneinfo/Asia/Chongqing 这个文件。

/etc/localtime ： 本地系统的时间设定文件，如果 clock 文件中规定了使用的

时间设定文件为/usr/share/zoneinfo/Asia/Chongqing，K-UX 操作系统就会将 Chongqing 那个文件复制一份为/etc/localtime，所以系统的时间显示就会以 Chongqing 那个时间设定文件为准。

21.4 设置 Chrony 服务器

直接编辑/etc/chrony.conf 文件即可

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
#pool 2.centos.pool.ntp.org iburst
local stratum 10
allow 100.2.0.0/16
# Record the rate at which the system clock gains/losses time.
#driftfile /var/lib/chrony/drift
#
## Allow the system clock to be stepped in the first three updates
## if its offset is larger than 1 second.
#makestep 1.0 3
#
## Enable kernel synchronization of the real-time clock (RTC).
#rtcsync
#
## Enable hardware timestamping on all interfaces that support it.
##hwtimestamp *
#
## Increase the minimum number of selectable sources required to adjust
## the system clock.
##minsources 2
#
## Allow NTP client access from local network.
##allow 192.168.0.0/16
```

取消 local stratum 10 这行注释，allow 100.2.0.0/16 为手动添加，说明允许该 IP 段内的客户端通过这台服务器获取时间。

更改后重新启动 chrony 服务器，并配置开机自启动

```
#systemctl restart chronyd  && systemctl  enable chronyd
```

21.5 客户端测试

21.5.1 安装 Chrony 软件包

```
# dnf  install  chrony  -y
```

21.5.2 配置 chrony 服务

对于客户端编辑/etc/chrony.conf 文件。

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
#pool 2.centos.pool.ntp.org iburst
server 100.2.127.199 iburst
#server 100.2.126.26 iburst
# Record the rate at which the system clock gains/losses time.
#driftfile /var/lib/chrony/drift
#
## Allow the system clock to be stepped in the first three updates
## if its offset is larger than 1 second.
#makestep 1.0 3
#
## Enable kernel synchronization of the real-time clock (RTC).
#rtcsync
#
## Enable hardware timestamping on all interfaces that support it.
##hwtimestamp *
#
## Increase the minimum number of selectable sources required to adjust
## the system clock.
##minsources 2
#
## Allow NTP client access from local network.
##allow 192.168.0.0/16
```

server 100.2.127.199 iburst 为手动添加，说明该机器从 IP 为 100.2.127.199 的机器同步时间。

21.5.3 查看同步结果

通过以下命令查看同步状态：

```
#chronyc sources -v
```

```
210 Number of sources = 1

  .-- Source mode  '^' = server, '=' = peer, '#' = local clock.
 / .- Source state '*' = current synced, '+' = combined , '-' = not combined,
| /   '?' = unreachable, 'x' = time may be in error, '~' = time too variable.
||                                                 .- xxxx [ yyyy ] +/- zzzz
||      Reachability register (octal) -.           |  xxxx = adjusted offset,
||      Log2(Polling interval) --.      |          |  yyyy = measured offset,
||                                \     |          |  zzzz = estimated error.
||                                 |    |           \
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* 100.2.127.199                11   6    17     1   -225ns[+5138ns] +/- 8073us
```

说明：当状态为^*时说明同步时间成功。

第 22 章 FTP 服务

22.1 配置 vsftp 服务的宿主

```
#useradd vsftpdadmin -s /sbin/nologin -M
```

这个 vsftpdadmin 只是用来替换 root 的，并不需要登录。

22.2 建立 ftp 虚拟宿主账户

```
#useradd ftpuser -s /sbin/nologin -M
```

这 ftpuser 只个虚拟帐户的宿主，本身是不用登录的。

22.3 配置 vsftpd.conf

配置之前要先备份一下原来的配置文件。

```
# mv vsftpd.conf vsftpd.conf.bak
#vim /etc/vsftpd/vsftpd.conf
#anonymous_enable=YES  -->  anonymous_enable=NO  //不允许匿名用户访问，默认是允许。
#chroot_list_enable=YES  -->  chroot_list_enable=YES  //不允许 FTP 用户离开自己主目录
#user_config_dir=/etc/vsftpd/vconf/userlocal  #这一步非常重要，要记住这一步。一会要根据这个配置新建文件夹
ftp_data_port=4040  #修改端口号
reverse_lookup_enable=NO
pasv_enable=yes
pasv_min_port=48790
pasv_max_port=48800
listen_port=48796
```

#设定虚拟用户个人 Vsftp 的配置文件存放路径。也就是说，这个被指定的

目录里，将存放每个 Vsftp 虚拟用户个性的配置文件，一个需要注意的地方就是这些配置文件名必须和虚拟用户名相同。

22.4 建立虚拟用户文件

```
#mkdir /etc/vsftpd/vconf
#touch /etc/vsftpd/vconf/vir_user
```

22.5 建立虚拟用户

```
#vi /etc/vsftpd/vconf/vir_user
virtualuser          //用户名
12345678             //密码
```

22.6 生成数据库

```
#db_load -T -t hash -f /etc/vsftpd/vconf/vir_user /etc/vsftpd/vconf/vir_user.db
```

22.7 设置数据库文件访问权限

```
#chmod 600 /etc/vsftpd/vconf/vir_user
#chmod 600 /etc/vsftpd/vconf/vir_user.db
```

22.8 修改/etc/pam.d/vsftpd

```
#vim /etc/pam.d/vsftpd
auth sufficient pam_userdb.so db=/etc/vsftpd/vconf/vir_user
account sufficient pam_userdb.so db=/etc/vsftpd/vconf/vir_user
```

(要想同时使用系统用户和虚拟用户，就需要把 required 改成 sufficient)

根据 3 配置的 user_config_dir=/etc/vsftpd/vconf/userlocal，新建 userlocal 文件夹。

```
#mkdir /etc/vsftpd/vconf/userlocal
```

根据 5 建立的用户名建立一个文件。例如 5 建立的帐户是 virtualuser，则新建一个 virtualuser 文件。

```
#touch /etc/vsftpd/vconf/userlocal/virtualuser
```

编辑该用户访问的文件路径。

```
#vim /etc/vsftpd/vconf/userlocal/virtualuser
```

输入如下：

```
local_root=/root
anonymous_enable=NO #禁止匿名用户访问
write_enable=YES #开启写权限
local_umask=022 #上传后文件的权限掩码
anon_upload_enable=NO #关闭匿名下载
anon_mkdir_write_enable=NO #关闭匿名创建文件夹
idle_session_timeout=60 #会话自动关闭时间  60 是因分钟
data_connection_timeout=120 #数据延迟时间
max_clients=10 #最大连接数
max_per_ip=5 #同一个 ip 同时允许 5 个 IP 联机
local_max_rate=1048576 #实体用户传输速度限制，单位 B/s。0 代表不限制
```

22.9 启动 ftp 服务

配置就此完成，重启 vsftpd 服务：#systemctl restart vsftpd.service

查看系统端口状态：#netstat -tulnp。如果能看 48796 端口正在被 vsftpd 调用说明启动成功。

第 23 章 firewalld 基础

23.1 防火墙简介

防火墙的主要功能是依据策略对穿越防火墙自身的流量进行过滤，防火墙策略可以基于流量的源目地址、端口号、协议、应用等信息来定制，然后防火墙使用预先定制的策略规则监控出入的流量，若流量与某一条策略规则相匹配，则执行相应的处理，反之则丢弃。

23.2 策略规则

K-UX 系统中存在多款防火墙管理工具，其中 firewalld 服务是默认的防火墙配置管理工具。firewalld 不是真正的防火墙，它只是用来定义防火墙策略的防火墙管理工具，也可以认为是一种服务，firewalld 服务把配置好的防火墙策略交由内核层面的 nftables 包过滤框架来处理。相较于传统的防火墙管理配置工具，firewalld 支持动态更新技术并加入了区域（zone）的概念。简单来说，区域就是 firewalld 预先准备了几套防火墙策略集合（策略模板），用户可以根据生产场景的不同而选择合适的策略集合，从而实现防火墙策略之间的快速切换，从而极大地提升了防火墙策略的应用效率。

firewalld 中常见的区域名称（默认为 public）以及相应的策略规则：

区域	默认策略规则
trusted	允许所有的数据包
home	拒绝流入的流量，除非与流出的流量相关；而如果流量与 ssh、mdns、ipp-client、amba-client、dhcpv6-client 服务相关，则允许流量
internal	等同于 home 区域
work	拒绝流入的流量，除非与流出的流量相关；而如果流量与 ssh、ipp-client、dhcpv6-client 服务相关，则允许流量
public	拒绝流入的流量，除非与流出的流量相关；而如果流量与 ssh、dhcpv6-

	client 服务相关，则允许流量
external	拒绝流入的流量，除非与流出的流量相关；而如果流量与 ssh 服务相关，则允许流量
dmz	拒绝流入的流量，除非与流出的流量相关；而如果流量与 ssh 服务相关，则允许流量
block	拒绝流入的流量，除非与流出的流量相关
drop	拒绝流入的流量，除非与流出的流量相关

23.3 启动和停止

查看 firewalld 服务状态：

```
#systemctl status firewalld.service
```

启动/停止/重启 firewalld 服务：

```
#systemctl start firewalld.service
#systemctl stop firewalld.service
#systemctl restart firewalld.service
```

开机时启用/禁用 firewalld 服务：

```
#systemctl enable firewalld.service
#systemctl disable firewalld.service
```

锁定/取消锁定 firewalld 服务：

```
#systemctl mask firewalld.service
#systemctl unmask firewalld.service
```

23.4 配置 firewalld

firewall-cmd 是 firewalld 防火墙配置管理工具的 CLI（命令行界面）版本，它的参数一般都是以“长格式”来提供的。使用 firewalld 配置的防火墙策略默认为运行时（Runtime）模式，又称为当前生效模式，而且随着系统的重启会失效。如果想让配置策略一直存在，就需要使用永久（Permanent）模式，方法就是在用

firewall-cmd 命令正常设置防火墙策略时添加--permanent 参数，这样配置的防火墙策略就可以永久生效。但是，使用永久生效模式设置的策略只有在系统重启之后才能自动生效。如果想让配置的策略立即生效，需要手动执行 firewall-cmd --reload 命令。

查看防火墙版本：

```
# firewall-cmd --version
```

查看防火墙帮助：

```
# firewall-cmd --help
```

显示防火墙状态：：

```
# firewall-cmd --state
```

查看区域信息:

```
# firewall-cmd --get-active-zones
```

查看指定接口所属区域：

```
# firewall-cmd --get-zone-of-interface=ens33
```

拒绝所有包/取消拒绝状态/查看是否拒绝：

```
# firewall-cmd --panic-on
# firewall-cmd --panic-off
# firewall-cmd --state
```

更新防火墙规则(第一个无需断开连接，就是 firewalld 特性之一动态添加规则，第二个需要断开连接，类似重启服务)：

```
# firewall-cmd --reload
# firewall-cmd --complete-reload
```

将接口添加到区域（默认接口都在 public，永久生效再加上 --permanent 然后 reload 防火墙）：

```
# firewall-cmd --zone=public --add-interface=ens33
```

设置默认接口区域，立即生效无需重启：

```
# firewall-cmd --set-default-zone=public
```

查看所有打开的端口：

```
# firewall-cmd --zone=dmz --list-ports
```

加入一个端口到区域(若要永久生效方法同上)：

```
# firewall-cmd --zone=dmz --add-port=8080/tcp
```

打开一个服务，类似于将端口可视化，服务需要在配置文件中添加，/etc/firewalld 目录下有 services 文件夹:

```
# firewall-cmd --zone=public --add-service=https
```

移除服务:

```
# firewall-cmd --zone=public --remove-service=https
```

第 24 章 中断处理

24.1 简介

中断其实就是由硬件或软件所发送的一种称为 IRQ（中断请求）的信号。

中断允许让设备，如键盘，串口卡，并口等设备表明它们需要 CPU。

一旦 CPU 接收了中断请求，CPU 就会暂时停止执行正在运行的程序，并且调用一个称为中断处理器或中断服务程序（interrupt service routine）的特定程序。

中断服务程序或中断处理器可以在中断向量表中找到，而这个中断向量表位于内存中的固定地址中。中断被 CPU 处理后，就会恢复执行之前被中断的程序。

其实，在机器启动的时候，系统就已经识别了所有设备，并且也把相应的中断处理器加载到中断表中。

下面是请求 CPU 关注的两种方式：

1. 基于中断
2. 基于轮询

所有的操作系统都是基于中断驱动的。

24.2 查看中断/proc/interrupts 文件

在 K-UX 的机器上，/proc/interrupts 这个文件包含有关于哪些中断正在使用和每个处理器各被中断了多少次的信息。

```
#cat /proc/interrupts
 CPU0   CPU1   CPU2   CPU3
  0: 3710374484       0     0     0   IO-APIC-edge   timer
  1:         20       0     0     0   IO-APIC-edge   i8042
  6:          5       0     0     0   IO-APIC-edge   floppy
  7:          0       0     0     0   IO-APIC-edge   parport0
  8:          0       0     0     0   IO-APIC-edge   rtc
  9:          0       0     0     0   IO-APIC-level   acpi
 12:        240       0     0     0   IO-APIC-edge   i8042
```

```
14:   11200026        0     0     0   IO-APIC-edge   ide0
51:   61281329        0     0     0   IO-APIC-level   ioc0
59:          1        0     0     0   IO-APIC-level   vmci
67:   19386473        0     0     0   IO-APIC-level   ens0
75:   94595340        0     0     0   IO-APIC-level   ens1
NMI:          0        0     0     0
LOC: 3737150067 3737142382 3737145101 3737144204
ERR:          0
MIS:          0
```

对上面文件的输出，解释如下：

第一列表示 IRQ 号.

第二、三、四列表示相应的 CPU 核心被中断的次数。在上面的例子中，timer 表示中断名称（为系统时钟）。3710374484 表示 CPU0 被中断了 3710374484 次。i8042 表示控制键盘和鼠标的键盘控制器。

对于像 rtc（real time clock）这样的中断，CPU 是不会被中断的。因为 RTC 存在于电子设备中，是用于追踪时间的。

NMI 和 LOC 是系统所使用的驱动，用户无法访问和配置。

IRQ 号决定了需要被 CPU 处理的优先级。IRQ 号越小意味着优先级越高。例如，如果 CPU 同时接收了来自键盘和系统时钟的中断，那么 CPU 首先会服务于系统时钟，因为他的 IRQ 号是 0 。

IRQ0 ： 系统时钟（不能改变）

IRQ1 ： 键盘控制器（不能改变）

IRQ3 ： 串口 2 的串口控制器（如有串口 4，则其也使用这个中断）

IRQ4 ： 串口 1 的串口控制器（如有串口 3，则其也使用这个中断）

IRQ5 ： 并口 2 和 3 或 声卡

IRQ6 ： 软盘控制器

IRQ7： 并口 1。它被用于打印机或若是没有打印机，可以用于任何的并口。

而对于像操作杆（或称为游戏手柄）上的 CPU，它并不会等待设备发送中断。因为操作杆主要用于游戏，操作杆的移动必须非常快，因此使用轮询的方式

检测设备是否需要 CPU 的关注还是比较理想的。使用轮询方式的缺点是 CPU 就处于了忙等状态，因为 CPU 会不停的多次检查设备。但是需要注意的是在 K-UX 中，这种处理信号的方式也是必不可少的。

24.3 硬中断

对于上文所讨论的场景都是属于硬中断的例子。硬中断主要分为两种类别：

1. 非屏蔽中断(Non-maskable interrupts，即 NMI）：就像这种中断类型的字面意思一样，这种中断是不可能被 CPU 忽略或取消的。NMI 是在单独的中断线路上进行发送的，它通常被用于关键性硬件发生的错误，如内存错误，风扇故障，温度传感器故障等。

2. 可屏蔽中断（Maskable interrupts）：这些中断是可以被 CPU 忽略或延迟处理的。当缓存控制器的外部针脚被触发的时候就会产生这种类型的中断，而中断屏蔽寄存器就会将这样的中断屏蔽掉。我们可以将一个比特位设置为 0，来禁用在此针脚触发的中断。

24.4 软中断

这些中断是在 CPU 执行指令（也就是说在进程正在运行的时候）的时候产生的，因为在执行指令时，CPU（确切的说应是在 CPU 中的运算器）自身会产生一个异常（此处的异常也可理解为软中断）。

例如，一个数字除以 0（当然这是不可能的），此时就会导致一个 divide-by-zero 的异常，从而导致计算机将此计算取消或者显示一个错误的信息。

在文件/proc/stat 中，包含了一些关于系统内核的统计信息，也包含一些中断信息。

```
[root@inspur ~]# vim /proc/interrupts
You have new mail in /var/spool/mail/root
[root@inspur ~]# cat /proc/stat
cpu    5239037 18880 1169616 2427187699 114249 4732 49129 0 0
cpu0 116032 76 26901 75911065 1901 13 50 0 0
```

```
cpu1 359388 1126 44880 75622614 443 1225 26049 0 0
cpu2 141446 110 41232 75838533 28384 585 5435 0 0
cpu3 218822 1070 28651 75804954 1559 482 186 0 0
cpu4 270691 68 138555 75590564 46322 875 8649 0 0
cpu5 195164 1306 49062 75785147 19867 664 4514 0 0
cpu6 115771 32 35723 75903782 345 64 6 0 0
cpu7 166445 1282 32160 75854999 356 477 4 0 0
cpu8 113780 67 28143 75908175 3079 207 2272 0 0
cpu9 275924 1033 27486 75749900 1204 8 168 0 0
cpu10 117451 74 27342 75910538 304 0 13 0 0
cpu11 185184 1254 22309 75846643 326 0 6 0 0
cpu12 111352 16 11298 75932805 224 0 26 0 0
                    .
intr 2471812823
```

在 intr 这一行，显示了自从系统启动以来所产生的中断数。第一列表示所有被服务的中断数。后续的每一列都表示一个特定中断的总数。

SMP_AFFINITY

SMP 是指对称多处理器。smp_affinity 文件主要用于某个特定 IRQ 要绑定到哪个 CPU 核心上。在/proc/irq/IRQ_NUMBER/目录下都有一个 smp_affinity 文件，这个文件中，所表示的 CPU 核心以十六进制来表示的。例如，网卡的中断号是：

```
#grep /enp0s1 /proc/interrupts
67: 23834931 0 0 0 IO-APIC-level eth0
cat /proc/irq/67/smp_affinity
00000000，00000000，00000000，00000000，00000000，00000000，00000000，
00000001
```

上面的十六进制对应的十进制是 1，也就是说所有的和网卡驱动相关的中断都是有 CPU0 来提供服务的。

可以通过手动改变 smp_affinity 文件中的值来将 IRQ 绑定到指定的 CPU 核心上，或者启用 irqbalance 服务来自动绑定 IRQ 到 CPU 核心上。

IRQ Balance

irqbalance 是一个实用程序，它主要是用于分发中断请求到 CPU 核心上，有助于性能的提升。它的目的是寻求省电和性能优化之间的平衡。可以使用 yum 进行安装：

```
[root@inspur network-scripts]# rpm -qa|grep irqbalance
irqbalance-1.4.0-4.kux.ppc64le
```

启动 irqbalance 服务后，中断在 CPU 上的分布如下：

```
# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:  950901695          0          0          0   IO-APIC-edge   timer
  1:         13          0          0          0   IO-APIC-edge   i8042
  6:         96      10989        470          0   IO-APIC-edge   floppy
  7:          0          0          0          0   IO-APIC-edge   parport0
  8:          1          0          0          0   IO-APIC-edge   rtc
  9:          0          0          0          0   IO-APIC-level  acpi
 12:        109       1787          0          0   IO-APIC-edge   i8042
 15:         99   84813914          0          0   IO-APIC-edge   ide1
 51:      17371          0   46689970          0   IO-APIC-level  ioc0
 67:       1741          0          0  225409160   PCI-MSI   ens0
 83:          0          0          0          0   PCI-MSI   vmci
NMI:          0          0          0          0
LOC:  950902917  950903742  950901202  950901400
ERR:          0
MIS:          0
```

irqbalance 对于包含多个核心的系统来说是非常有用的。因为通常中断只被第一个 CPU 核心服务。

第 25 章 远程图形界面 VNC

25.1 配置 VNC 启动参数

K-UX 下的 VNC 可以同时启动多个 vncserver，各个 vncserver 之间用显示编号(display number)来区分，每个 vncserver 服务监听 3 个端口，它们分别是：

5800+显示编号: VNC 的 httpd 监听端口，如果 VNC 客户端为 IE，Firefox 等非 vncviewer 时必须开放。

5900+显示编号: VNC 服务端与客户端通信的真正端口，必须无条件开放。

6000+显示编号: X 监听端口，可选。

显示编号、开放的端口分别由 /etc/sysconfig/vncservers 文件中的 VNCSERVERS 和 VNCSERVERARGS 控制。

VNCSERVERS 的设置方式为“VNCSERVERS="显示编号 1:用户名 1 …"”，如：VNCSERVERS="1:root 2:aiezu"。

VNCSERVERARGS 的设置方式为 VNCSERVERARGS[显示编号 1]="参数一 参数值一 参数二 参数值二 ……"，如 VNCSERVERARGS[2]="-geometry 800x600 -nohttpd"，VNCSERVERARGS 的详细参数有：

-geometry：桌面分辨率，默认 1024x768；

-nohttpd ：不监听 HTTP 端口(58xx 端口)；

-nolisten tcp：不监听 X 端口(60xx 端口)；

-localhost：只允许从本机访问；

-AlwaysShared：默认只同时允许一个 vncviewer 连接，此参数允许同时连多个 vncviewer；

-SecurityTypes：None，登录不需要密码认证；VncAuth，默认值，要密码认证。

25.2 运行

第一次运行设置密码

输入命令：#vncpasswd 设置密码.然后输入两次密码.

```
[root@localhost bin]# vncpasswd
Password:
Verify:
Would you like to enter a view-only password (y/n)? y
Password:
Verify:
```

25.3 启动 vncserver

先关闭，输入命令：vncserver -kill :1

启动： vncserver

25.4 测试

在客户端用 vncviewer 连接测试。

第 26 章 Multipath

多路径 I/O 可以用于在服务器节点和存储阵列之间配置多路径 I/O 路径。这些 I/O 路径组成物理 SAN 连接，包括独立的电缆、交换机和控制器。多路径软件能够聚合 I/O 路径，并创建一个由聚合路径组成的新设备。

26.1 多路径 I/O 概述

多路径 I/O 能够提供：

路径冗余

多路径 I/O 在 active/passive 配置模式下可以提供故障转移。在 active/passive 配置模式下，任何时候只有一半的路径用于 I/O。如果任何一条 I/O 路径（包括电缆，交换机，控制器）失效，多路径 I/O 会自动切换到备用路径。

性能提升

多路径 I/O 可配置成 inactive/active 模式，此模式下，I/O 通过循环方式在路径传播。在某些配置中，多路径 I/O 可以检测 I/O 路径负载以及动态负载均衡。

26.2 安装多路径 I/O

多路径 I/O 包含编译的默认设置，这是比较合适的、常见的多路径配置。建立多路径是一个简单的过程。

用于配置系统多路径 I/O 的基本步骤如下：

安装 device-mapper 、device-mapper-multipath RPM 包。

编辑 multipath.conf 配置文件：

使用命令生成配置文件 multipath.conf，

命令为：mpathconf --user_friendly_names n

a、注释掉默认黑名单

编辑/etc/multipath.conf 配置文件，初始状态所有设备都处于黑名单中，进行如下注释后变成：

```
# blacklist {
```

```
# devnode "*"
# }
```

a. 跟据需要改变现有的默认设置

根据实际情况，修改默认配置。

b. 保存配置文件

1. 开启多路径 I/O 守护进程。

执行下列命令：

```
# modprobe dm-multipath
# systemctl start multipathd.service
# multipath -v2
```

multipath -v2 命令打印多路径 I/O 路径，显示哪些设备是多路径的。如果该命令没有打印任何信息，确保所有的 SAN 连接正确设置，并且系统是配置了多路径。

#systemctl enable multipathd

此命令保证系统启动会运行多路径守护进程。

2. 使用 multipath 命令创建多路径设备。

26.3 配置多路径 I/O

multipath 配置文件如下：

```
格式 1：
1． vim /etc/multipath.conf
注: 如果没有 multipath.conf 文件，需要把
/usr/share/doc/device-mapper-multipath/multipath.conf
拷贝到/etc 下
2． 修改 multipath.conf 文件：
blacklist {
    devnode "hda"
    wwid   SATA_ST91000640NS_9XG6JNN6(系统分区对应的 wwid：ll /dev/disk/by-
```

```
id/
)
}

如果系统安装在存储上，会映射出两块盘，在黑名单中要加入映射出来的两个 lun 的设备名称和 wwid 号，缺一不可
defaults {
        user_friendly_names no
}
multipaths {
      multipath {
    wwid    360080e50001ff03c00000f2f4ba97e4a
            alias               mpath1
            path_grouping_policy      multibus     #multibus  为负载均衡模式/failover  为主备模式
            path_checker          readsector0
            path_selector         "round-robin 0"
            failback          immediate
            rr_weight             priorities
            no_path_retry         5
        }
        multipath {
            wwid                  360080e50001fceb800000f5453e3f354
            alias               mpath2
            path_grouping_policy      multibus
            path_checker          readsector0
            path_selector         "round-robin 0"
            failback          immediate
            rr_weight             priorities
```

```
        no_path_retry       5
    }
}
```

格式 2：

```
blacklist {
     devnode "hda"
    wwid  SATA_ST91000640NS_9XG6GTJN(需手动获取本地硬盘 wwid 号：ll
/dev/disk/by-id/)
}
```

如果系统安装在存储上，会映射出两块盘，在黑名单中要加入映射出来的两个 lun 的设备名称和 wwid 号，缺一不可

```
defaults {
    user_friendly_names yes
}
defaults {
    udev_dir        /dev
    polling_interval       10
    selector          "round-robin 0"
    path_grouping_policy      multibus
    getuid_callout         "/sbin/scsi_id -g -u -s /block/%n"
    prio_callout           /bin/true
    path_checker         readsector0
    rr_min_io            100
    max_fds              8192
    rr_weight            priorities
```

```
        failback        immediate
        no_path_retry       fail
        user_friendly_names     yes
}
```

3. 配置完 multipath.conf 文件后，需要重启 multipath 服务：

systemctl restart multipathd

4. 初始化链路：multipath -v2. 此时会在目录/dev/mapper/下生成 mpath0，mpath1，使用 multipath -ll 命令会查看到链路情况

5. 在/dev/mapper 目录下可以看到有 mpath0，mpath1 等信息

6. 配置多路径模式的命令： multipath -p multibus # 配置为负载均衡模式

multipath -p failover # 配置为主备模式

多路径配置文件分为以下部分：

blacklist：列出不会被用于多路径的特殊设备。默认情况下，所有设备都被列入黑名单。通常黑名单部分默认会注释掉。如果有某个磁盘没有加入黑名单，那么会自动配置为多路径。

blacklist_exceptions：列出多路径候选者，这些候选者将根据黑名单中的参数被列入黑名单。

defaults：多路径 I/O 一般缺省设置。

multipaths：个人设置多路径设备的特点。这些值覆盖配置文件中 defaults 和 devices 部分的默认值。

devices：设置个人存储控制器。这些值覆盖配置文件中 defaults 部分。如果你使用的存储阵列默认不支持，你需要为阵列创建一个子设备段。

配置文件 blacklist：

配置文件黑名单部分指定的设备，不会在系统配置多路径设备时使用。黑名单中的设备，将不会被列入多路径设备的分组中。

默认情况下，所有设备都列入黑名单，如下所示的初始配置文件。

```
blacklist {
devnode "*"
```

```
}
```

要启用默认支持的所有设备上进行多路径，需要注释掉这些行。

注释掉黑名单后，您可以指定一般的设备类型和个别设备黑名单。可以根据下列条件将设备列入黑名单：

1. Blacklisting by WWID

您可以指定单个设备，根据其 WWID（World-Wide IDentification），在配置文件的黑名单部分加入对应 WWID 项。例如：

```
blacklist {
wwid 26353900f02796769
}
```

2. Blacklisting by Device Name

您可以通过设备名称将某些设备类型加入黑名单，通过指定的设备配置文件黑名单部分的 devnode 项，使他们不会加入到多路径分组中。例如：

```
blacklist {
devnode "^sd[a-z]"
}
```

3. Blacklisting by Device Type

您可以在配置文件的黑名单部分使用 device 条目指定特定的设备类型。下面的例子将所有 IBM DS4200 设备和所有 HP 设备列入黑名单。

```
blacklist {
device {
vendor "IBM"
product "3S42" #DS4200 Product 10
}
device {
vendor "HP"
product "*"
}
}
```

配置文件 defaults：

在/etc/multipath.conf 配置文件中，设置默认 user_friendly_names 参数设置为 YES，如下所示。

```
defaults {
user_friendly_names yes
}
```

这将覆盖该 user_friendly_names 参数的默认值。

配置文件包含了默认配置模板。这部分被注释掉了，如下所示。

```
#defaults {
#           udev_dir /dev
#           polling_interval 10
#           selector "round-robin 0"
#           path_grouping_policy multibus
#           getuid_callout "/sbin/scsi_id -g -u -s /block/%n"
#           prio_callout /bin/true
#           path_checker readsector0
#           rr_min_io 100
#           rr_weight priorities
#           failback immediate
#           no_path_retry fail
#           user_friendly_name yes
#}
```

为了覆盖任何配置参数的默认值，你可以从模板复制相关的行到 defaults 部分并取消其注释。例如，为了覆盖 path_grouping_policy 参数，使其配置成 multibus，而不是默认 failover，从模板中复制相应的行到配置文件的初始 defaults 部分，并取消其注释，如下所示。

```
defaults {
user_friendly_names yes
```

```
path_grouping_policy multibus
}
```

下面介绍了在 multipath.conf 配置文件中 defaults 部分的属性规定。这些值直接被多路径 I/O 使用，除非它们被 multipath.conf 配置文件的 devices 和 multipaths 部分的属性所覆盖。

属性	描述
udev_dir	指定设备节点所在的 udev 创建的目录。默认值为/udev 的。
polling_interval	指定两条路径 path checks 之间的间隔时间。默认值是 5 秒。
selector	指定使用的默认算法用于确定哪条路径用于下一个 I/O 操作。默认值是 round-robin 0。
path_grouping_policy	指定默认路径分组策略，适用于未指定的多路径。可能的值包括： failover = 每个路径组中一条路径 multibus = 一个路径组中所有有效路径 group_by_serial = 每个被检测到的序列号为一个路径组 group_by_prio = 每个路径优先级值为一个路径组 group_by_node_name = 每个目标节点名为一个路径组 默认值是故障转移。
getuid_callout	指定默认程序和参数用来调用以获得一个唯一的路径标识符。必须是绝对路径。默认值是/sbin/scsi_id - g -u -s。
prio_callout	指定默认程序和参数来调用以获得路径优先级值。例如，SPC-3 的 ALUA 位提供了一个可利用的优先级值。 “none”是一个有效值。默认值

	是 no callout，说明一切路径是平等的。
path_checker	指定默认的方法来确定状态的路径。可能的值包括：readsector0， tur， emc_clariion， hp_sw 和 directio。默认值是 readsector0。
rr_min_io	指定 I/O 请求的数量路由到一个路径前切换到在当前路径组的下一个路径。该默认值是 1000。
rr_weight	如果设置 priorities，而不是发送 rr_min_io 选择请求之前调用的路径选择下一路径，请求发送数量取决于 rr_min_io 倍的路径优先级，由 prio_callout 程序决定。目前，有重点标注唯一的设备，使用 group_by_prio 路径组策略对于设备具有优先级 callouts，这意味着路径组中所有路径具有相同的优先级。如果设置为 uniform，所有路径权重是相等的。默认值是 uniform。
failback	指定路径组故障恢复。0 或 immediate，指定只要有一个路径组优先级高于当前路径组则该系统切换到该路径组。数值大于零，指定故障恢复延迟，以秒表示。manual 指定故障恢复只可能操作干预发生。默认值是 immediate。
no_path_retry	此属性的数值指定系统应该尝试使用一个失效路径的次数。fail 值表示立即失效，没有排队。queue 值表示不应该停止，直到路径被修复。默认值是 null。
user_friendly_names	如果设置为 yes，指定系统应使用绑定文件 /var/lib/multipath/bindings 来指派一个持久性和独特的别名，形式如 mpathn。如果设置为 no，指定系统应使用多路径别名 WWID。默认值是 no。

多路径设备配置属性:

下列介绍了能够在 multipath.conf 配置文件 multipaths 部分设定的属性。这些属性仅适用于一个指定的多路径设备。这些默认值被多路径使用，并能覆盖 multipath.conf 配置文件的 defaults 和 devices 部分所设定的属性。

属性	描述
wwid	指定多路径设备 WWID，应用于 multipath 属性。
alias	指定多路径设备的符号名。
path_grouping_policy	指定默认路径分组策略，适用于未指定的多路径。可能的值包括： failover = 每个路径组中一条路径 multibus = 所有有效路径在一个路径组 group_by_serial = 每个被检测到的序列号为一个路径组 group_by_prio = 每个路径优先级值为一个路径组 group_by_node_name = 每个目标节点名为一个路径组
path_selector	指定使用的默认算法用于确定哪条路径用于下一个 I/O 操作。
failback	指定路径组故障恢复。0 或 immediate，指定只要有一个路径组优先级高于当前路径组则该系统切换到该路径组。数值大于零，指定故障恢复延迟，以秒表示。manual 指定故障恢复只可能操作干预发生。默认值是 immediate。
rr_weight	如果设置 priorities，然后而不是发送 rr_min_io 选择请求之前调用的路径选择下一路径，请求发送数量取决于 rr_min_io 倍的路径优先级，由 prio_callout 程序决定。目前，有重点标注唯一的设备，使用 group_by_prio 路径组策略对于设备具有优先级 callouts，这意味着路径组中所有路

	径具有相同的优先级。如果设置为 uniform，所有路径权重是相等的。默认值是 uniform。
no_path_retry	此属性的数值指定系统应该尝试使用一个失效路径的次数。fail 值表示立即失效，没有排队。queue 值表示不应该停止，直到路径被修复。默认值是 null。
rr_min_io	指定 I/O 请求的数量路由到一个路径前切换到在当前路径组的下一个路径。

下面的例子显示了配置文件中指定两个具体的多路径设备的多路径属性。第一个设备具有的 WWID 为 3600508b4000156d70001200000b0000 和黄色符号名。

示例中的第二个多路径设备拥有 1DEC_____321816758474 的 WWID 和红色符号名。在这个例子中，rr_weight 属性用来设置优先级属性。

```
multipaths {
multipath {
wwid 3600508b4000156d70001200000b0000
alias yellow
path_grouping_policy multibus
path_checker readsector0
path_selector "round-robin 0"
failback manual
rr_weight priorities
no_path_retry 5
}
multipath {
wwid 1DEC_____321816758474
alias red
rr_weight priorities
}
}
```

配置文件 devices：

下面介绍了可以设置每个独立存储设备在 multipath.conf 配置文件的 devices 部分的属性。这些属性由多路径使用，除非他们被 multipath.conf 配置文件 multipaths 部分所覆盖。这些属性会覆盖 multipath.conf 配置文件 defaults 部分的属性设置。

属性	描述
vendor	指定的存储设备供应商名称，为 device 属性所适用，例如 COMPAQ。
product	指定的存储设备产品名称，为 device 属性所适用，例如 HSV110（C）COMPAQ。
product_blacklist	指定正则表达式，用于设备的黑名单产品。
path_grouping_policy	指定默认路径分组策略，适用于未指定的多路径。可能的值包括： failover = 每个路径组中一条路径 multibus = 所有有效路径在一个路径组 group_by_serial = 每个被检测到的序列号为一个路径组 group_by_prio = 每个路径优先级值为一个路径组 group_by_node_name = 每个目标节点名为一个路径组
getuid_callout	指定默认程序和参数来调用以获得一个唯一的路径标识符。需要使用绝对路径。
prio_callout	指定默认程序和参数来调用以获得路径优先级。权重相加为每个路径组，以确定万一路径失效后使用下一个路径组。“none”是有效值。
path_checker	指定默认的方法来确定的状态的路径。可能的值包括：readsector0， tur， emc_clariion， hp_sw

	和 directio。
path_selector	指定使用的默认算法用于确定哪条路径用于下一个 I/O 操作。
failback	指定路径组故障恢复。0 或 immediate，指定只要有一个路径组优先级高于当前路径组则该系统切换到该路径组。数值大于零，指定故障恢复延迟，以秒表示。manual 指定故障恢复只可能操作干预发生。
features	多路径设备的额外功能。现有的唯一特点是 queue_if_no_path ， 这 是 作 为 设 置 相 同 no_path_retry 排队。
hardware_handler	指定一个用于执行交换路径或处理 I/O 错误的硬件模块。可能的值包括 0，1 emc、1 rdac、1 hp_sw 和 1 alua。该默认值是 0。
rr_min_io	指定 I/O 请求的数量路由到一个路径前切换到在当前路径组的下一个路径。
rr_weight	如果设置 priorities，然后而不是发送 rr_min_io 选择请求之前调用的路径选择下一路径，请求发送数量取决于 rr_min_io 倍的路径优先级，由 prio_callout 程序决定。目前，有重点标注唯一的设备，使用 group_by_prio 路径组策略对于设备具有优先级 callouts，这意味着路径组中所有路径具有相同的优先级。如果设置为 uniform，所有路径权重是相等的。默认值是 uniform。
no_path_retry	此属性的数值指定系统应该尝试使用一个失效路径的次数。fail 值表示立即失效，没有排队。queue 值表示不应该停止，直到路径被修复。默认值是 null。

下面的例子显示了一个在多路径配置文件中的 device 条目。

```
#   }
#   device {
#           vendor "COMPAQ "
#           product "MSA1000 "
#           path_grouping_policy multibus
#           path_checker tur
#           rr_weight priorities
#   }
#}
```

多路径管理和命令

1. 多路径查询命令

你可以使用 multipath 命令的-l 和-ll 选项来显示当前的多路径配置。-l 选项显示由 sysfs 和 device mapper 获得的多路径拓扑信息。使用- ll 选项显示除了- l 显示的信息，还包括系统的其他可用组件的信息。

当显示多路径配置，有三个详细级别，使用 multipath 命令可以通过- v 选项指定。指定- v0 没有输出。指定- v1 只输出创建或更新的多路径设备名称，然后您可以使用其他工具，如 kpartx。指定- v2 打印所有检测到的路径，多路径设备和设备映射关系。

下面的例子显示了 multipath -l 命令的输出。

```
# multipath -l
mpath1 (3600d0230003228bc000339414edb8101)
[size=10 GB][features="0"][hwhandler="0"]
\_ round-robin 0 [prio=1][active]
\_ 2:0:0:6 sdb 8:16 [active][ready]
\_ round-robin 0 [prio=1][enabled]
\_ 3:0:0:6 sdc 8:64 [active][ready]
```

1. 多路径命令选项

下表介绍了 multipath 命令的一些选项。

选项	描述
-l	显示当前多路径配置，从 sysfs 和 device mapper 获取信息。
-ll	显示当前多路径配置，从 sysfs、device mapper 以及系统其他所有可用组件获取信息。
-f device	删除已命名的多路径设备。
-F	删除所有多路径设备。

2. dmsetup 设备映射命令

您可以使用 dmsetup 命令找出哪些设备映射条目匹配多路径设备。

下面的命令显示所有的多路径设备以及它们的主、次设备号。次设备号决定 DM 设备的名称。例如，次设备号 3 对应的多路径设备/ dev/dm-3。

```
# dmsetup ls
mpath2 (253，  4)
mpath4p1 (253，  12)
mpath5p1 (253，  11)
mpath1 (253，  3)
mpath6p1 (253，  14)
mpath7p1 (253，  13)
mpath0 (253，  2)
mpath7 (253，  9)
mpath6 (253，  8)
VolGroup00-LogVol01 (253，  1)
mpath5 (253，  7)
VolGroup00-LogVol00 (253，  0)
mpath4 (253，  6)
mpath1p1 (253，  10)
mpath3 (253，  5)
```

3. multipathd 互动控制台

multipathd -k 命令是一个与 multipathd 守护进程的交互式接口。此命令将引入一个多路径互动控制台。输入该命令后，您可以输入 help 以获取可用命令的列表，你可以输入一个交互式命令，也可以输入 Ctrl-D 来退出。

multipathd 交互式控制台可用于解决您的系统可能遇到的问题。例如，下面的命令序列显示多路径配置，包括默认值，然后退出控制台。

```
# multipathd -k
> > show config
> > CTRL-D
```

下面的命令序列，确保多路径获取了 multipath.conf 配置文件的任何更改。

```
# multipathd -k
> > reconfigure
> > CTRL-D
```

使用下面的命令序列，以确保 path checker 是否工作正常。

```
# multipathd -k
> > show paths
> > CTRL-D
```

第 27 章 可插拔认证模块（PAM）

可插拔认证模块（PAM）机制采用模块化设计和插件功能，使用户可以轻易地在应用程序中插入新的认证模块或替换原先的组件，同时不必对应用程序做任何修改，从而使软件的定制，维护和升级更加轻松。因为认证和鉴别机制与应用程序之间相对独立，所以应用程序可以通过 PAM API 来方便地使用 PAM 提供的各种鉴别功能而不必了解太多的底层细节。此外 PAM 的易用性也较强，主要表现在它对上层屏蔽了鉴别和认证的具体细节，所以用户不必被迫学习各种各样的鉴别方式，也不必记住多个口令；又由于它实现了多鉴别认证机制的集成问题，所以单个程序可以轻易集成多种鉴别机制，但用户仍可以用同一口令而且感觉不到采用了各种不同的鉴别方法。

27.1 PAM 的配置文件

配置文件放在了应用接口层中，它与 PAM API 配合使用，从而达到了在应用中灵活插入所需鉴别模块的目的。它的作用主要是为应用选定具体的鉴别模式，模块间的组合以及规定模块的行为。下面是一个示例配置文件：

```
[root@localhost ~]# cat /etc/pam.d/system-auth
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 500 quiet
auth        required      pam_deny.so

account     required      pam_unix.so
account     sufficient    pam_succeed_if.so uid < 500 quiet
account     required      pam_permit.so

password    requisite     pam_cracklib.so try_first_pass retry=3 minlen=11 minclass=3 di
fok=8
password    sufficient    pam_unix.so md5 shadow nullok try_first_pass use_authtok
password    required      pam_deny.so

session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet use_uid
session     required      pam_unix.so
```

可以看到，配置文件由许多配置项（每行对应一个配置项）组成，每一行又分为四列（每列对应一栏）：

第一栏，认证鉴别接口类型：

指明程序所使用的 PAM 的认证接口类型，其对应了四类接口：

auth:表示鉴别类接口模块类型用于检查用户和密码，并分配权限；

这种类型的模块为用户验证提供两方面服务。让应用程序提示用户输入密码或者其他标记，确认用户合法性；通过它的凭证许可证，设定组成员关系或者其他优先权。

account:表示账户类接口，主要负责账户合法性检查，确认账户是否过期，是否有权限登录系统等；

这种模块执行的是基于非验证的账户管理。它主要用于限制/允许用户对某个服务的访问时间，当前有效的系统资源（最多可以多少用户），限制用户位置（如：root 只能通过控制台登录）。

多数情况下 auth 和 account 会一起用来对用户登录和使用服务的情况进行限制。这样的限制会更加完整。比如下面是一个具体的例子：Login 是一个应用程序。Login 要完成两件工作---首先查询用户，然后为用户提供所需的服务，例如提供一个 shell 程序。通常 Login 要求用户输入名称和密码进行验证。当用户名输入的时候，系统自然会去比对该用户是否是一个合法用户，是否存在于本地或者远程的用户数据库中。如果该账户确实存在，那么是否过期。

这些个工作是由 account 接口来负责的。

如果用户满足上述登录的前提条件，那么他是否具有可登录系统的口令，口令是否过期等。这个工作就要由 auth 接口来负责了，它通常会将用户口令信息加密并提供给本地（/etc/shadow）或者远程的（ldap，kerberos 等）口令验证方式进行验证。

如果用户能够登录程序，证明 auth 和 account 的工作已经完成。但整个验证过程并没有完全结束。因为还有一些其他的问题没有得到确认。例如，用户能够在服务器上同时开启多少个窗口登录，用户可以在登录之后使用多少终端多少时间，用户能够访问哪些资源和不能访问哪些资源等等。也就是说登录之后的后续验证和环境定义等还需要其他的接口。这就是下面要提到的两组接口：

session:会话类接口。实现从用户登录成功到退出的会话控制；

处理为用户提供服务之前/后需要做的一些事情。包括：开启/关闭交换数据的信息，监视目录等，设置用户会话环境等。也就是说这是在系统正式进行服务

之前的最后一道关口。

password：口令类接口。控制用户更改密码的全过程。也就是有些资料所说的升级用户验证标记。

注意，上述接口在使用的时候，每行只能指定一种接口类型，如果成行需要多种接口的话，可在多行中分别予以规定。

第二栏，control_flag 控制位：

规定如何处理 PAM 模块鉴别认证的结果，简而言之是鉴别认证成功或者失败之后会发生什么事，如何进行控制。单个应用程序可以调用多种底层模块，通常称为“堆叠”。对应于某程序按照配置文件中出现顺序执行的所有模块称为“堆”，堆中的各模块的地位与出错时的处理方式由 control_flag 栏的取值决定，它的四种可能取值分别为：required,requisite,sufficien 或 optional:

required:表示该行为以及所涉及模块的成功是用户通过鉴别的必要条件。换句话说，只有当对应于应用程序的所有带 required 标记的模块全部成功后，该程序才能通过鉴别。同时，如果任何带 required 标记的模块出现了错误，PAM 并不立刻将错误消息放回给应用程序，而是在所有模块对调用完毕后才将错误消息返回调用它的程序。就是必须将所有模块都执行一次，其中任何一个模块验证出错，验证都会继续进行，并在执行完成之后才返回错误信息。这样做的目的就是不让用户知道自己被哪个模块拒绝，通过一种隐藏的方式来保护系统服务。就像设置防火墙规则的时候讲拒绝类的规则都设置为 drop 一样，以至于用户在访问网络不成功的时候无法准确判断到底是被拒绝还是目标网络不可达。

requisite:与 required 相仿，只要带此标记的模块返回成功后，用户才能通过鉴别。不同之处在于其一旦失败就不再执行堆中后面的其他模块，并且鉴别过程到此结束，同时也会立即返回错误信息。与上面的 required 相比，似乎更显得光明正大一些。

sufficient:表示该行以及所涉及模块验证成功是用户通过鉴别的充分条件。也就是说只要标记为 sufficient 的模块一旦验证成功，那么 PAM 便立即向应用程序返回成功结果而不必尝试任何其他模块。即便后面的层叠模块使用了 requisite 或者 required 控制标志也是一样。当标记为 sufficient 的模块失败时，sufficient 模块会当做 optional 对待。因此拥有 sufficient 标志位的配置项在执行验证出错的时

候并不会导致整个验证失败，但执行验证成功之时则大门敞开。所以该控制位的使用务必慎重。

optional：它表示即便该行为所涉及的模块验证失败用户仍能通过认证。在 PAM 体系中，带有该标记的模块失败后将继续处理下一个模块。也就是说即使本行指定的模块验证失败，也运行用户享受应用程序提供的服务。使用该标志，PAM 框架会忽略这个模块产生的验证错误，继续顺序执行下一个层叠模块。

第三栏，module_path 即所使用模块的全路径名称。

以 K-UX 为例，值得注意的是在 i386/i686，x86_64 和 ia64 系统中模块的全路径名称是不一样的。所以当有的时候用户将一些 PAM 的配置文件从原来系统复制到新的系统时，如果两种系统架构不同，那么不修改模块路径名称则可能导致 PAM 报错。

第四栏，options 用于向特定模块传递相关的选项，然后由模块分析解释这些选项。

比如使用此栏打开模块调试模式，或向某模块传递诸如超时值之类的参数等。另外它还拥有支持下文所述的口令映射技术。

如果任一栏出现错误或者某模块没有找到，那么所在行被忽略并将其作为严重错误进行记录。

27.2 PAM 的工作原理与流程

以 K-UX 系统为例，当 pam 安装之后有两大部分：在/lib/security/目录下的各种 pam 模块以及/etc/pam.d 和/etc/pam.d 目录下的针对各种服务和应用已经定义好的 pam 配置文件。当某一个有认证需求的应用程序需要验证的时候，一般在应用程序中就会定义负责对其认证得 PAM 配置文件。以 vsftpd 为例，在它的配置文件/etc/vsftpd/vsftpd.conf 中就有这样一行定义：

```
114
115 pam_service_name=vsftpd
```

表示登录 FTP 服务器的时候进行认证是根据/etc/pam.d/vsftpd 文件定义的内容进行。

首先要声明一点：system-auth 是一个非常重要的 pam 配置文件，主要负责

用户登录系统的认证工作。而且该文件不仅仅只是负责用户登录系统认证，其他的程序和服务通过 include 接口也可以调用到它，从而节省了很多重新自定义配置的工作。所以应该说该文件是系统安全的总开关和核心 pam 配置文件。

下面是/etc/pam.d/system-auth 文件的全部内容：

```
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        sufficient    pam_fprintd.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        required      pam_deny.so

account     required      pam_unix.so
account     sufficient    pam_localuser.so
account     sufficient    pam_succeed_if.so uid < 1000 quiet
account     required      pam_permit.so

password    requisite     pam_pwquality.so try_first_pass local_users_only retry=3 authtok_type=
password    sufficient    pam_unix.so sha512 shadow nullok try_first_pass use_authtok
password    required      pam_deny.so

session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
-session     optional      pam_systemd.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet use_uid
session     required      pam_unix.so
```

第一部分表示，当用户登录的时候，首先会通过 auth 类接口对用户身份进行识别和密码认证。所有在该过程中验证会经过几个带 auth 的配置项。

其中的第一步是通过 pam_env.so 模块来定义用户登录之后的环境变量，pam_env.so 允许设置和更改用户登录的环境变量，默认情况下，若没有特别指定配置文件，将依据/etc/security/pam_env.conf 进行用户登录之后环境变量的设置。

然后通过 pam_unix.so 模块来提示用户输入密码，并将用户密码与/etc/shadow 中记录的密码信息进行对比，如果密码对比对结果正确则允许用户登录，而且该配置项的使用的是“sufficient”控制位，即表示只要该配置项的验证通过，用户即可完全通过认证而不用再去走下面的认证项。不过在特殊情况下，用户允许使用空密码登录系统，例如当将某个用户在/etc/shadow 中的密码字段删除之后，该用户可以只输入用户名直接登录系统。

下面的配置项中，通过 pam_succeed_if.so 对用户的登录条件做一些限制，表示允许 uid 大于 1000 的用户在通过密码验证的情况下登录，在 K-UX 系统中，一般系统用户的 uid 都在 1000 之内，所以该项即表示允许使用 useradd 命令以及默认选项建立的普通用户直接由本地控制台登录系统。

最后通过 pam_deny.so 模块对所有不满足上述任意条件的登录请求直接拒绝，

pam_deny.so 是一个特殊的模块，该模块返回值永远为否，类似于大多数安全机制的配置准则，在所有认证规则走完之后，对不匹配任何规则的请求直接拒绝。

第二部分的三个配置项主要表示通过 account 账户类接口来识别账户的合法性以及登录权限。

第一行仍然使用 pam_unix.so 模块来声明用户需要通过密码认证。第二行承认了系统中 uid 小于 1000 的系统用户的合法性。之后对所有类型的用户登录请求都开放控制台。

第三部分会通过 password 口令类接口来确认用户使用的密码或者口令的合法性。第一行配置项表示需要的情况下将调用 pam_cracklib 来验证用户密码复杂度。如果用户输入密码不满足复杂度要求或者密码错，最多将在三次这种错误之后直接返回密码错误的提示，否则期间任何一次正确的密码验证都允许登录。需要指出的是，pam_cracklib.so 是一个常用的控制密码复杂度的 pam 模块。带 pam_unix.so 和 pam_deny.so 的两行配置项的意思与之前类似。都表示需要通过密码认证并对不符合上述任何配置项要求的登录请求直接予以拒绝。不过用户如果执行的操作是单纯的登录，则这部分配置是不起作用的。

第四部分主要将通过 session 会话类接口为用户初始化会话连接。其中几个比较重要的地方包括，使用 pam_keyinit.so 表示当用户登录的时候为其建立相应的密钥环，并在用户登出的时候予以撤销。不过该行配置的控制位使用的是 optional，表示这并非必要条件。之后通过 pam_limits.so 限制用户登录时的会话连接资源，相关 pam_limit.so 配置文件是/etc/security/limits.conf，默认情况下对每个登录用户都没有限制。

通过对上述 system-auth 配置文件的修改，模块的增加和选项的变化，从很大的程度上增加了用户登录验证的安全性要求。另外也一定需要注意，在整个的 PAM 配置文件当中，配置项以及模块调用的逻辑顺序非常关键。因为 PAM 是按照配置项的先后顺序来进行验证。错误的模块调用顺序很可能导致严重的安全问题甚至系统错误。所以对 PAM 配置进行修改的时候务必要考虑这一点。

第 28 章 mdadm 管理

mdadm 有 6 种模式，前两种模式：Create，Assemble 用于设置和激活阵列；manage 模式用于操作在活动阵列中的设备；Follow 或 Monitor 模式允许管理员对活动阵列设置事件提醒和动作；Build 模式用于对旧阵列使用旧版本的 md 驱动；更有 Grow 模式能扩展阵列；剩下的是 Misc 模式，它包括对多种内部的任务和没有指定特别模式的一种操作。

28.1 部署

28.1.1 准备磁盘

只能使用 Sofware RAID 格式的磁盘才能组成阵列，所以，首先要把磁盘格式化。除了系统盘 sda 外，需要对 sdb，sdc，sdd 进行操作

a）对 sdb 进行分区

fdisk /dev/sdb

```
[root@inspur ~]# fdisk /dev/sdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0xf15e576c.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').
```

分区前状态：

```
Command (m for help): p

Disk /dev/sdb: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xf15e576c

   Device Boot      Start         End      Blocks   Id  System
```

n，划分区：

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-261, default 1):
Using default value 1
Last cylinder, +cylinders or +size{K,M,G} (1-261, default 261):
Using default value 261
```

t，修改分区格式为 fd：

```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): fd
Changed system type of partition 1 to fd (Linux raid autodetect)
```

w，保存：

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

b） 用同样的方法，对 sdc，sdd 进行分区和保存

最后状态如下：

```
Disk /dev/sdb: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0xf15e576c

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1         261     2096451   fd  Linux raid autodetect

Disk /dev/sdc: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x450342a2

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1         261     2096451   fd  Linux raid autodetect

Disk /dev/sdd: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0dad147b

   Device Boot      Start         End      Blocks   Id  System
/dev/sdd1               1         261     2096451   fd  Linux raid autodetect
```

28.1.2 准备阵列

mdadm 能支持 LINEAR，RAID0(striping)，RAID1(mirroring)，RAID4，RAID5，RAID6 和 MULTIPATH 的阵列模式。

创建命令格式如下：

```
mdamd [mode] [options]
```

例如：创建一个 RAID 0 设备：

```
#mdadm --create --verbose /dev/md0 --level=0 --raid-devices=3 /dev/sdb1 /dev/sdc1 /dev/sdd1
```

/dev/sdb1 /dev/sdc1 /dev/sdd1 表示创建的阵列模式，--raid-devices 表示参加阵列的磁盘数量。

```
[root@inspur ~]# mdadm --create --verbose /dev/md0 --level=0 --raid-devices=3 /dev/sdb1
/dev/sdc1 /dev/sdd1
mdadm: chunk size defaults to 512K
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

也能这样表达，意思是相同的：

```
# mdadm -C /dev/md0 -ayes -l0 -n3 /dev/sd[bcd]1
```

还能增加-c128 参数，指定 chunk size 为 128k(默认为 64k)。

28.1.3 设置文件

mdadm 不采用/etc/mdadm.conf 作为主要设置文件，它能完全不依赖该文件而不会影响阵列的正常工作。

该设置文件的主要作用是方便跟踪软 RAID 的设置。对该设置文件进行设置是有好处的，但不是必须的。推荐对该文件进行设置。

通常是这样来建立的：

```
# echo DEVICE /dev/sd[bcd]1 > /dev/mdadm.conf
# mdadm -Ds >> /etc/mdadm.conf
# mdadm --detail --scan >> /etc/mdadm.conf
```

28.1.4 格式化阵列

后续，只要把/dev/md0 作为一个独立的设备来进行操作即可：

```
# mkfs.ext3 /dev/md0
# mkdir /mnt/test
# mount /dev/md0 /mnt/test/
```

若要开机自动挂载，请加入/etc/fstab 中：

```
vim /etc/fstb
/dev/md0        /mnt/test       auto    defaults        0 0
```

28.2 监视管理

mdadm 能非常方便的对阵列进行监视和管理的操作，也包括了停止和启动阵列等常用维护。

28.2.1 查看

```
# cat /proc/mdstat
```

能查看所有使用 md 驱动的阵列的状态：

```
[root@inspur ~]# cat /proc/mdstat
Personalities : [raid0]
md0 : active raid0 sdd1[2] sdc1[1] sdb1[0]
      6288384 blocks super 1.2 512k chunks

unused devices: <none>
```

```
#mdadm -D /dev/md0
```

查看指定阵列的详细信息：

```
[root@inspur ~]# mdadm --detail /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Mon Sep  7 00:20:09 2015
     Raid Level : raid0
     Array Size : 6288384 (6.00 GiB 6.44 GB)
   Raid Devices : 3
  Total Devices : 3
    Persistence : Superblock is persistent

    Update Time : Mon Sep  7 00:20:09 2015
          State : clean
 Active Devices : 3
Working Devices : 3
 Failed Devices : 0
  Spare Devices : 0

     Chunk Size : 512K

           Name : inspur:0  (local to host inspur)
           UUID : 744e7910:76457c78:698eb75c:332dfcc8
         Events : 0

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       1       8       33        1      active sync   /dev/sdc1
       2       8       49        2      active sync   /dev/sdd1
```

28.2.2 停止

```
#mdadm -S /dev/md0
```

停止指定阵列，并释放磁盘(--stop):

```
[root@inspur ~]# mdadm -S /dev/md0
mdadm: stopped /dev/md0
[root@inspur ~]# cat /proc/mdstat
Personalities : [raid0]
unused devices: <none>
```

注意：停止后，原组成阵列的磁盘将处于空闲状态，一旦误操作这些磁盘，将不能再重启激活原阵列。

28.2.3 启动

```
# mdadm -A /dev/md0 /dev/sd[bcd]1
```

启动指定的阵列，也可以理解为将一个新阵列装配到系统中（--assemble）：

```
[root@inspur ~]# mdadm -A /dev/md0 /dev/sd[bcd]1
mdadm: /dev/md0 has been started with 3 drives.
```

28.2.4 测试

如果没有设置/etc/mdadm.conf 文件，而且又忘了某磁盘属于那个阵列，则使用检测(--examine)

```
# mdadm -E /dev/sdb1
```

```
[root@inspur ~]# mdadm -E /dev/sdb1
/dev/sdb1:
          Magic : a92b4efc
        Version : 1.2
    Feature Map : 0x0
     Array UUID : 744e7910:76457c78:698eb75c:332dfcc8
           Name : inspur:0  (local to host inspur)
  Creation Time : Mon Sep  7 00:20:09 2015
     Raid Level : raid0
   Raid Devices : 3

 Avail Dev Size : 4192886 (2047.65 MiB 2146.76 MB)
    Data Offset : 16 sectors
   Super Offset : 8 sectors
          State : clean
    Device UUID : 1ae7ee67:668b5f7a:12ef2fbc:e72f22ee

    Update Time : Mon Sep  7 00:20:09 2015
       Checksum : 313d8546 - correct
         Events : 0

     Chunk Size : 512K

   Device Role : Active device 0
   Array State : AAA ('A' == active, '.' == missing)
```

获得 UUID 后，也能这样激活阵列：

```
# mdadm -Av /dev/md0 --uuid=1ae7ee67:668b5f7a:12ef2fbc:e72f22ee /dev/sd*
```

能看到，只要磁盘没有损坏，这样装配时非常方便的：

```
[root@inspur ~]# mdadm -Av /dev/md0 --uuid=1ae7ee67:668b5f7a:12ef2fbc:e72f22ee /dev/sd*
mdadm: looking for devices for /dev/md0
mdadm: Cannot assemble mbr metadata on /dev/sda
mdadm: no recogniseable superblock on /dev/sda1
mdadm: no recogniseable superblock on /dev/sda2
mdadm: no recogniseable superblock on /dev/sda3
mdadm: Cannot assemble mbr metadata on /dev/sdb
mdadm: /dev/sdb1 has wrong uuid.
mdadm: Cannot assemble mbr metadata on /dev/sdc
mdadm: /dev/sdc1 has wrong uuid.
mdadm: Cannot assemble mbr metadata on /dev/sdd
mdadm: /dev/sdd1 has wrong uuid.
```

28.2.5 添加及删除磁盘

mdadm 能在 Manage 模式下，对运行中的阵列进行添加及删除磁盘。常用于标示 failed 磁盘，增加 spare(冗余)磁盘，及替换磁盘等。

例如：原来状态是：

```
          Name : inspur:0  (local to host inspur)
          UUID : 744e7910:76457c78:698eb75c:332dfcc8
        Events : 0

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       1       8       33        1      active sync   /dev/sdc1
       2       8       49        2      active sync   /dev/sdd1
```

则能使用--fail 指定坏磁盘，--remove 去掉：

```
# mdadm /dev/md0 --fail /dev/sdc1 --remove /dev/sdc1
```

```
[root@inspur ~]# mdadm /dev/md0 --fail /dev/sdc1 --remove /dev/sdc1
mdadm: set /dev/sdc1 faulty in /dev/md0
mdadm: hot removed /dev/sdc1 from /dev/md0
[root@inspur ~]# cat /proc/mdstat
Personalities : [raid0] [raid1]
md0 : active raid1 sdd1[2] sdb1[0]
      2095360 blocks super 1.2 [3/2] [U_U]

unused devices: <none>
```

```
[root@inspur ~]# mdadm -D /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Mon Sep  7 01:15:19 2015
     Raid Level : raid1
     Array Size : 2095360 (2046.59 MiB 2145.65 MB)
  Used Dev Size : 2095360 (2046.59 MiB 2145.65 MB)
   Raid Devices : 3
  Total Devices : 2
    Persistence : Superblock is persistent

    Update Time : Mon Sep  7 01:17:21 2015
          State : clean, degraded
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

          Name : inspur:0  (local to host inspur)
          UUID : 65fde0ab:369a71bd:80399993:ae216ed4
        Events : 19

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       1       0        0        1      removed
       2       8       49        2      active sync   /dev/sdd1
```

需要注意的是：对于某些阵列模式，如 RAID0 等，是不能用--fail 和--remove 的。

```
[root@inspur ~]# mdadm /dev/md0 --fail /dev/sdc1 --remove /dev/sdc1
mdadm: set device faulty failed for /dev/sdc1:  Device or resource busy
```

增加一个新的阵列用磁盘

```
# mdadm /dev/md0 --add /dev/sdc1
```

```
[root@inspur ~]# mdadm /dev/md0 --add /dev/sdc1
mdadm: added /dev/sdc1
[root@inspur ~]# mdadm -D /dev/md0
/dev/md0:
        Version : 1.2
  Creation Time : Mon Sep  7 01:15:19 2015
     Raid Level : raid1
     Array Size : 2095360 (2046.59 MiB 2145.65 MB)
  Used Dev Size : 2095360 (2046.59 MiB 2145.65 MB)
   Raid Devices : 3
  Total Devices : 3
    Persistence : Superblock is persistent

    Update Time : Mon Sep  7 01:24:32 2015
          State : clean
 Active Devices : 3
Working Devices : 3
 Failed Devices : 0
  Spare Devices : 0

           Name : inspur:0  (local to host inspur)
           UUID : 65fde0ab:369a71bd:80399993:ae216ed4
         Events : 42

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       3       8       33        1      active sync   /dev/sdc1
       2       8       49        2      active sync   /dev/sdd1
```

需要注意的是：对于某些阵列模式，如 RAID0 等，是不能用--add 的。

28.2.6 删除阵列

```
#mdadm -S /dev/md0

# rm /dev/md0
```

修改/etc/mdadm.conf，/etc/fstab 的设+置文件，把相关的地方去掉；

最后，用 fdisk 对磁盘进行重新分区即可。

28.2.7 重建阵列

也可以在没有 fdisk 的情况下把使用过，将目前没有属于所有阵列的磁盘划分到新阵列中：

```
[root@inspur ~]# mdadm -Cv /dev/md0 -l5 -n2  -x1 /dev/sd[bcd]1
mdadm: layout defaults to left-symmetric
mdadm: layout defaults to left-symmetric
mdadm: chunk size defaults to 512K
mdadm: /dev/sdb1 appears to be part of a raid array:
    level=raid1 devices=3 ctime=Mon Sep  7 01:15:19 2015
mdadm: /dev/sdc1 appears to be part of a raid array:
    level=raid1 devices=3 ctime=Mon Sep  7 01:15:19 2015
mdadm: /dev/sdd1 appears to be part of a raid array:
    level=raid1 devices=3 ctime=Mon Sep  7 01:15:19 2015
mdadm: size set to 2095104K
Continue creating array? y
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

确认后即可。

第 29 章 Podman 容器

29.1 简介

Podman 用于管理和运行 Open Container Initiative(OCI)容器和容器镜像，通过 runC 容器运行时进程（不是守护进程）直接与镜像仓库、容器和镜像存储以及 Linux 内核进行交互，Podman 控制下的容器既可以由 root 用户运行，也可以由非特权用户运行。

通过以下命令可以查看版本信息：

```
#podman version
```

```
[root@inspur ~]# podman version
Version:            1.6.4
RemoteAPI Version:  1
Go Version:         go1.13.4
OS/Arch:            linux/ppc64le
```

29.2 Podman 管理镜像

Podman 可以从 Docker Hub 等流行的容器仓库以及私有仓库中拉取和推送容器像，使用 podman 命令可以运行，启动，停止，调查和删除容器镜像。

29.2.1 搜索镜像

使用 podman 命令可以在容器仓库中搜索镜像：

```
#podman search busybox
```

```
INDEX       NAME                                  DESCRIPTION
docker.io   docker.io/library/busybox             Busybox base image.
docker.io   docker.io/radial/busyboxplus          Full-chain, Internet enabled, busybox made f...
docker.io   docker.io/yauritux/busybox-curl       Busybox with CURL
docker.io   docker.io/vukomir/busybox             busybox and curl
docker.io   docker.io/arm64v8/busybox            Busybox base image.
docker.io   docker.io/ppc64le/busybox            Busybox base image.
docker.io   docker.io/odise/busybox-curl
docker.io   docker.io/s390x/busybox               Busybox base image.
docker.io   docker.io/prom/busybox                Prometheus Busybox Docker base images
docker.io   docker.io/arm32v7/busybox            Busybox base image.
docker.io   docker.io/amd64/busybox              Busybox base image.
docker.io   docker.io/joeshaw/busybox-nonroot    Busybox container with non-root user nobody
docker.io   docker.io/i386/busybox               Busybox base image.
docker.io   docker.io/p7ppc64/busybox            Busybox base image for ppc64.
```

29.2.2 拉取镜像

从容器仓库拉取容器镜像到本地系统：

```
#podman pull docker.io/library/busybox
```

```
[root@inspur ~]# podman pull docker.io/library/busybox
Trying to pull docker.io/library/busybox...
Getting image source signatures
Copying blob 9758c28807f2 done
Copying config f0b02e9d09 done
Writing manifest to image destination
Storing signatures
f0b02e9d092d905d0d87a8455a1ae3e9bb47b4aa3dc125125ca5cd10d6441c9f
```

29.2.3 显示镜像

将容器镜像拉取到本地之后，可查看容器镜像列表：

```
#podman images
```

```
[root@inspur ~]# podman images
REPOSITORY                  TAG      IMAGE ID       CREATED       SIZE
docker.io/library/busybox   latest   f0b02e9d092d   2 weeks ago   1.45 MB
```

29.2.4 检查镜像

通过查看镜像的详细信息，可以了解镜像的作用和镜像中包含的软件：

```
#podman inspect docker.io/library/busybox
```

```
[root@inspur ~]# podman inspect docker.io/library/busybox
[
    {
        "Id": "f0b02e9d092d905d0d87a8455a1ae3e9bb47b4aa3dc125125ca5cd10d6441
        "Digest": "sha256:a9286defaba7b3a519d585ba0e37d0b2cbee74ebfe590960b0
        "RepoTags": [
            "docker.io/library/busybox:latest"
        ],
        "RepoDigests": [
            "docker.io/library/busybox@sha256:a9286defaba7b3a519d585ba0e37d0
            "docker.io/library/busybox@sha256:c9249fdf56138f0d929e2080ae98ee
        ],
        "Parent": "",
        "Comment": "",
```

29.2.5 标记镜像

标记镜像，可以更直观地了解镜像包含的内容。实际上，标记镜像就是给镜

像添加别名，该别名可以由几个部分组成：registryhost/username/NAME:tag

```
#podman tag docker.io/library/busybox test

#podman tag docker.io/library/busybox test:1.0
```

```
[root@inspur ~]# podman tag f0b02e9d092d test
[root@inspur ~]# podman tag f0b02e9d092d test:1.0
[root@inspur ~]# podman images
REPOSITORY                  TAG      IMAGE ID       CREATED       SIZE
docker.io/library/busybox   latest   f0b02e9d092d   2 weeks ago   1.45 MB
localhost/test              latest   f0b02e9d092d   2 weeks ago   1.45 MB
localhost/test              1.0      f0b02e9d092d   2 weeks ago   1.45 MB
```

29.2.6 保存和加载镜像

可以将本地的容器镜像保存到存档文件或者目录中，便于存储、发送或者加载该镜像。

```
#podman save -o mypodman.tar docker.io/library/busybox:latest

#file mypodman.tar
```

```
[root@inspur ~]# podman save -o mypodman.tar docker.io/library/busybox:latest
[root@inspur ~]# file mypodman.tar
mypodman.tar: POSIX tar archive
```

若要使用保存的镜像压缩文件，可将其导入另一个容器环境：

```
#podman load -i mypodman.tar

#podman images
```

```
[root@inspur ~]# podman load -i mypodman.tar
Getting image source signatures
Copying blob d2421964bad1 skipped: already exists
Copying config f0b02e9d09 done
Writing manifest to image destination
Storing signatures
Loaded image(s): docker.io/library/busybox:latest
[root@inspur ~]# podman images
REPOSITORY                  TAG      IMAGE ID       CREATED       SIZE
docker.io/library/busybox   latest   f0b02e9d092d   2 weeks ago   1.45 MB
docker.io/library/busybox   1.0      f0b02e9d092d   2 weeks ago   1.45 MB
```

29.2.7 删除镜像

使用镜像 ID 作为选项可以删除不需要的镜像：

```
#podman rmi <镜像 ID>
```

```
[root@inspur ~]# podman rmi f0b02e9d092d
Untagged: docker.io/library/busybox:latest
Deleted: f0b02e9d092d905d0d87a8455a1ae3e9bb47b4aa3dc125125ca5cd10d6441c9f
[root@inspur ~]# podman images
REPOSITORY   TAG   IMAGE ID   CREATED   SIZE
```

29.3 Podman 管理容器

29.3.2 查看容器

1．容器列表

显示正在运行的容器列表

```
#podman ps
```

```
[root@inspur ~]# podman ps
CONTAINER ID  IMAGE                            COMMAND    CREATED         STATUS
628c2238d82f  docker.io/library/ubuntu:latest  /bin/bash  23 minutes ago  Up 8 minutes ago
```

显示所有正在运行或已停止的容器列表：

```
#podman ps -a
```

```
[root@inspur ~]# podman ps -a
CONTAINER ID  IMAGE                            COMMAND               CREATED
341fade60ebb  docker.io/library/ubuntu:latest  /bin/bash             3 minutes ago
628c2238d82f  docker.io/library/ubuntu:latest  /bin/bash             21 minutes ago
7b03c5a49e71  docker.io/library/ubuntu:latest  /bin/bash             24 minutes ago
6649f759f867  docker.io/library/ubuntu:latest  /bin/echo Hello I...  39 minutes ago
```

2．查看容器信息

通过 podman inspect 命令可以查看容器的信息：

```
#podman inspect <容器 ID>
```

```
[root@inspur ~]# podman inspect 0e816a3b0829
[
    {
        "Id": "0e816a3b082929309e9ed92935327fb964a4b588b4dd617d542a42139349944d",
        "Created": "2020-10-30T14:14:55.197106589+08:00",
        "Path": "/bin/bash",
        "Args": [
            "/bin/bash"
        ],
        "State": {
            "OciVersion": "1.0.1-dev",
            "Status": "running",
            "Running": true,
            "Paused": false,
```

容器的信息存储在层次结构中，可以查看容器的特定信息，如查看容器的进程 ID:

```
#podman inspect --format='{{.State.Pid}}' <容器 ID>
```

```
[root@inspur ~]# podman inspect --format='{{.State.Pid}}' 0e816a3b0829
66473
[root@inspur ~]# ps -ef | grep 66473
root       66473   66462  0 14:15 pts/0    00:00:00 /bin/bash
root       66523   29105  0 14:16 pts/2    00:00:00 grep --color=auto 66473
```

3．容器内部查看

使用 podman exec 命令可以进入正在运行的容器进程，不必中断容器在运行的应用程序，从而更好地查看容器的信息。

首先启动容器，确保其正在运行；使用 podman exec 命令访问正在运行的容器，然后查看容器的各项属性。

#podman exec -it <容器 ID> /bin/bash

```
[root@inspur ~]# podman ps
CONTAINER ID  IMAGE                            COMMAND    CREATED         STATUS
0e816a3b0829  docker.io/library/ubuntu:latest  /bin/bash  38 minutes ago  Up 38
[root@inspur ~]# podman exec -it 0e816a3b0829 /bin/bash
root@0e816a3b0829:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 06:15 pts/0    00:00:00 /bin/bash
root          24       0  0 06:53 pts/1    00:00:00 /bin/bash
root          31      24  0 06:54 pts/1    00:00:00 ps -ef
root@0e816a3b0829:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay          17G  4.8G   13G  28% /
tmpfs            64M     0   64M   0% /dev
tmpfs           901M   18M  883M   2% /etc/hosts
shm              63M     0   63M   0% /dev/shm
tmpfs           901M     0  901M   0% /sys/fs/cgroup
tmpfs           901M     0  901M   0% /proc/acpi
tmpfs           901M     0  901M   0% /proc/scsi
tmpfs           901M     0  901M   0% /sys/firmware
root@0e816a3b0829:/# uname -r
4.18.0-193.19.1.el8_2.x86_64
root@0e816a3b0829:/# free -m
              total        used        free      shared  buff/cache   available
Mem:           1800        1191         124          19         484         423
Swap:          2047          26        2021
root@0e816a3b0829:/# exit
exit
```

根据以上在容器内部运行的命令可以看出：进程表显示/bin/bash 命令的进程 ID 为 1；在主机进程表上可以看到/bin/bash 的进程（主机上的进程 ID 为 66473）但是容器内部看不到主机进程表中的正在运行的进程；容器中没有单独运行的内核；可以查看主机上的可用内存。

29.3.1 运行容器

运行容器，就是启动并根据容器镜像创建新的容器。podman run 所传递命令将容器内部视为运行环境，默认情况下，正在运行的应用程序可以查看：镜像提

供的文件系统和容器内部新的进程表。

查看用作容器基础的操作系统类型：

```
#podman run --rm docker.io/library/ubuntu cat /etc/os-release
```

```
[root@inspur ~]# podman run --rm docker.io/library/ubuntu cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04.1 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.1 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

在容器中运行 shell，查看容器内部并更改内容：

```
#podman run --name=mybash -it docker.io/library/ubuntu /bin/bash
```

```
[root@inspur ~]# podman run --name=mybash -it docker.io/library/ubuntu /bin/bas
root@628c2238d82f:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  2 03:06 pts/0    00:00:00 /bin/bash
root           8       1  0 03:06 pts/0    00:00:00 ps -ef
```

29.3.3 启动和停止容器

如果运行了某个容器却没有删除，该容器将会存储在本地系统上准备再次运行。通过 start 选项启动以前运行但未删除的容器，stop 选项停止正在运行的容器：

```
#podman start <容器 ID>

#podman stop <容器 ID>
```

```
[root@inspur ~]# podman start 628c2238d82f
628c2238d82f536a9001c12915efad1a0da3efbd2246ba7d103531459eab7f0c
[root@inspur ~]# podman ps
CONTAINER ID  IMAGE                                 COMMAND    CREATED
628c2238d82f  docker.io/library/ubuntu:latest  /bin/bash  26 minutes ago
[root@inspur ~]# podman stop 628c2238d82f
628c2238d82f536a9001c12915efad1a0da3efbd2246ba7d103531459eab7f0c
[root@inspur ~]# podman ps
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
```

29.3.4 删除容器

使用 podman rm 命令可以删除不需要的容器（添加-a 选项删除所有容器）：

```
#podman rm <容器 ID>
#podman rm -a
```

```
[root@inspur ~]# podman rm 7b03c5a49e71
7b03c5a49e714509f171dfb3ddf04fad788ec6d0125b7470c3faaec398c405b7
[root@inspur ~]# podman ps -a
CONTAINER ID  IMAGE                           COMMAND               CREATED
6649f759f867  docker.io/library/ubuntu:latest  /bin/echo Hello I...  About an hour ago
```

第 30 章 Unicorn 系统

UNICORN 集成在 K-UX 中做为一种辅助管理工具，主要解决管理员对服务器的操作难度，在对 K-UX 等系统不熟悉的情况下，也能简单的管理服务器做日常的维护工作。主要功能是收集服务器系统数据和更改服务器配置。

UNICORN 平台其功能主要为通过浏览器方式进行目标服务器的管理，并为这些管理设置一定的权限。系统主要分为三层：客户端、管理平台、目标服务器。平台共分为三个大的模块，分别是：监控管理、日志分析和配置管理。三大模块间无调用关系。

监控管理：监控管理包括监控配置、定时任务、监控基本信息、监控 CPU 信息、监控内存信息、监控网络 I/O 信息、监控磁盘 I/O 信息、监控文件系统信息、监控风扇信息和监控进程信息，能够对相关监控内容进行配置，获取 CPU、内存、网络 I/O、磁盘 I/O、传感器和进程的实时监控信息。

日志分析：目前只提供内核日志和 UNICORN 日志的查看功能，能够在 UNICORN 运行过程中记录相关日志。

配置管理：对目标服务器进行配置和修改都此进行，能够管理防火墙、NFS 服务、FTP 服务、网络、设备驱动等。

30.1 登陆

打开本系统，如图 30-1，在该页面上输入用户名、密码，点击登录按钮进入系统界面。

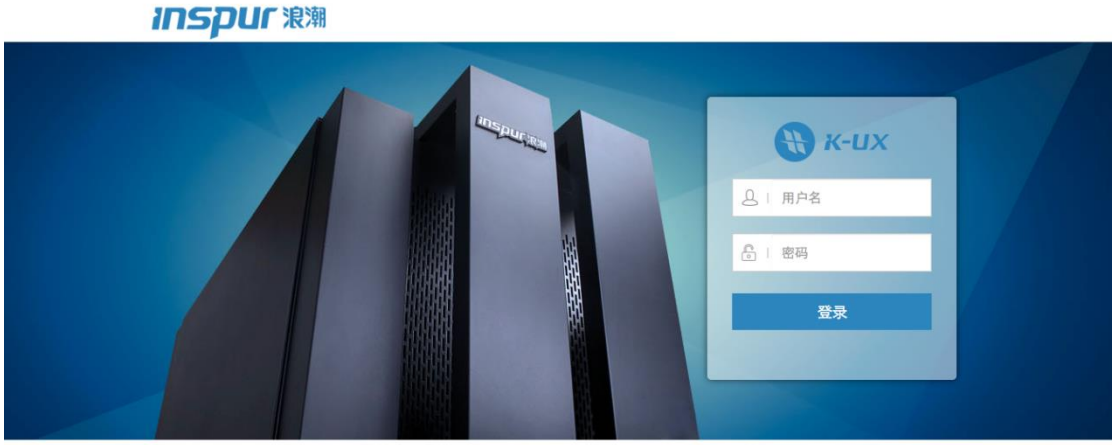


图 30- 1 登录

30.2 首页

本系统首页分为顶部菜单、监控信息、系统信息三部分。如图 30-2。

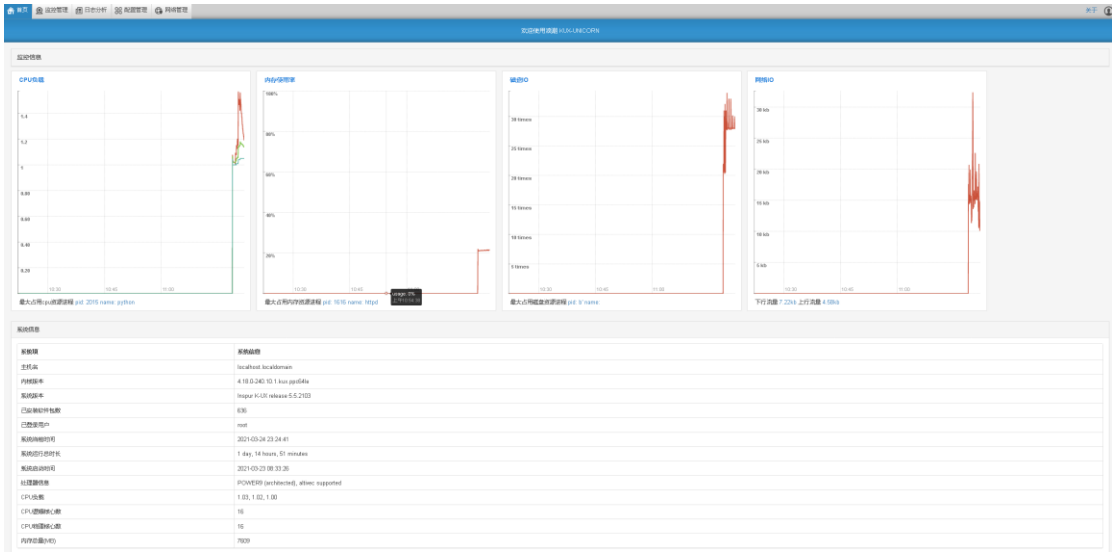


图 30- 2 首页

30.2.1 顶部菜单

顶部菜单左侧包含首页、监控管理、日志分析、配置管理、网络管理，如图 30-3。其中首页为独立连接，其他为独立模块，通过鼠标单击独立模块展开各自子模块。详细参见后续章节。

顶部菜单右侧包含关于和当前登录用户两个按钮。通过单击关于按钮打开关于界面，详细参见章节 7；通过单击当前登陆用户按钮可以注销当前登录并返回登录页面。

图 30-3 菜单

30.2.2 监控信息

监控信息集中展示监控模块中包含的四大主要子模块，其中包含 CPU 负载、内存使用率、磁盘 IO 和网络 IO 的重要信息，以折线图的方式给用户最直观的数据展示。如图 30-4。

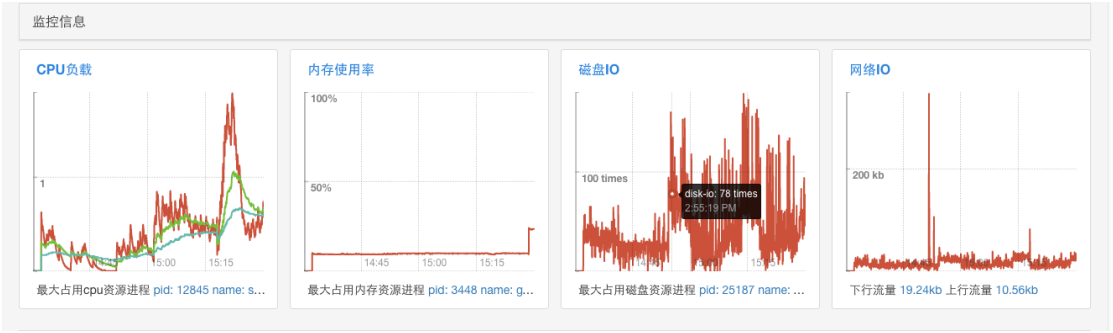


图 30-4 监控信息

30.2.3 系统信息

系统信息以列表的方式展示服务器的硬件类型、内核信息、OS 版本、运行时长等服务器与操作系统相关参数。如图 30-5。

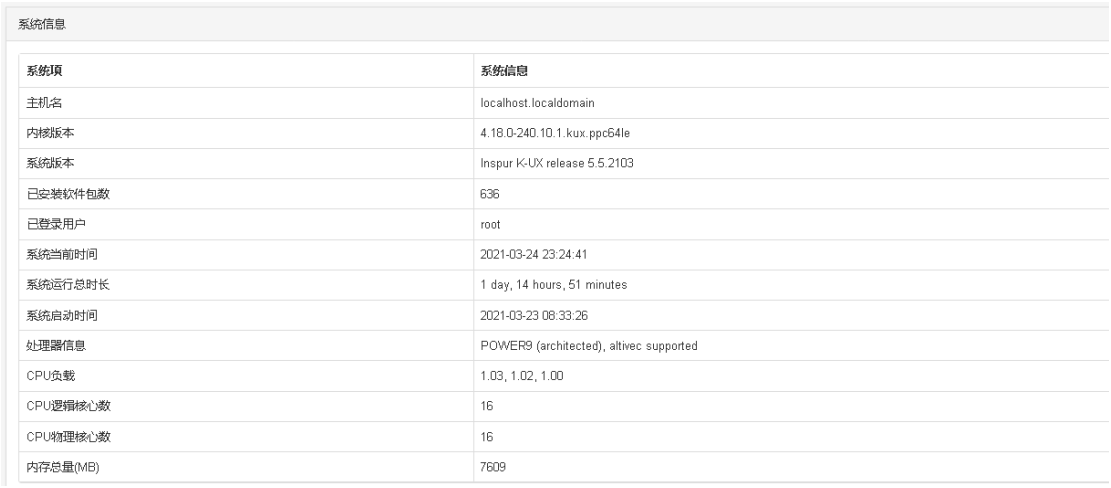


图 30-5 系统信息

30.3 监控管理

30.3.1 监控配置

本模块主要用于对实时监控进行监控开关配置。通过本模块可以方便的开启或关闭对以下模块的动态监控：基本信息配置、CPU 配置、内存配置、网络 I/O 配置、磁盘 I/O 配置、文件系统配置、传感器配置、进程配置。

点击开关处于 on 状态，则对应监控页面会显示出监控数据图像；

点击开关处于 off 状态，则对应监控页面的监控数据图像会被关闭。

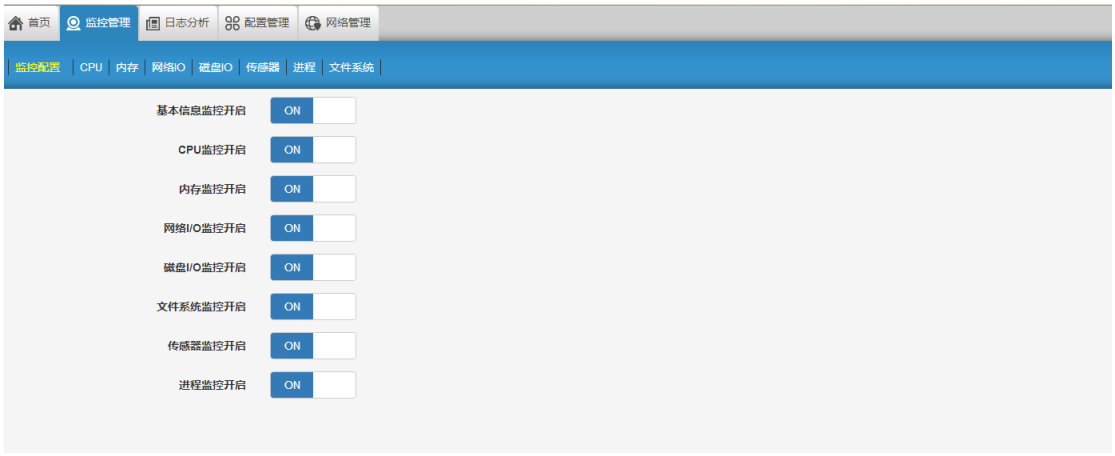


图 30-6 监控配置页面

30.3.2 CPU

本功能用于查看当前系统的 CPU 状态，用户可以通过本模块查看 CPU 负载、CPU 基本信息、各逻辑 CPU 使用信息、CPU 中断信息等。

1、CPU 占用率

点击菜单栏“监控管理”下的“CPU”子菜单，跳转到 CPU 占用率页面。通过此页面的信息，可以查询当前设备的 CPU 总核心数量以及每个核心的当前占用率。红色代表占用率大于 80%，黄色代表占用率位于 50%~80%之间，绿色代表占用率小于 50%。

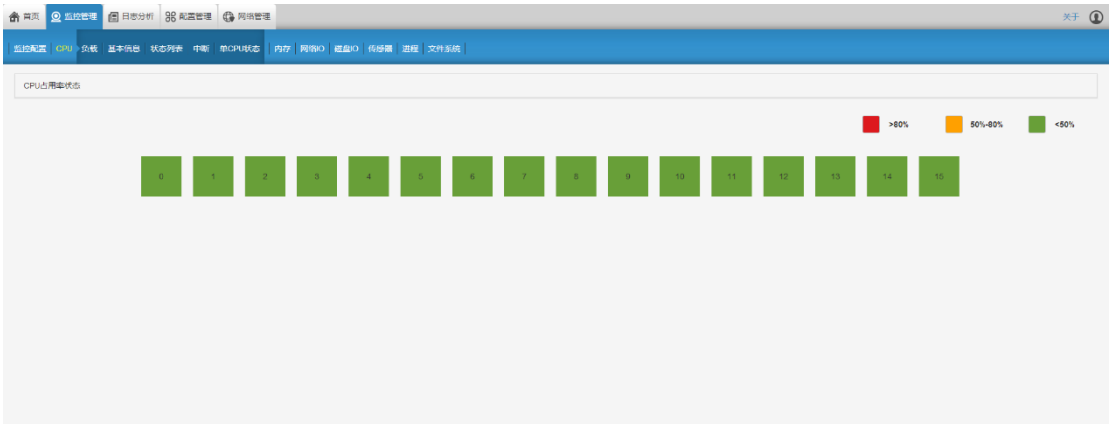


图 30-7 CPU 占用率页面

2、CPU 负载

点击菜单栏“监控管理”下的“CPU”子菜单中的“负载”，跳转到 CPU 负载监控页面，用户将获取到 CPU 的负载均值。监控数据折线图显示最近一分钟内的不同时间段的 CPU 平均负载。三种颜色分别代表了前 15 分钟、前 5 分钟和前 1 分钟的 CPU 负载均值。点击图像右侧的时间选项，可以选择想要观察的某个时间段内的 CPU 负载均值。

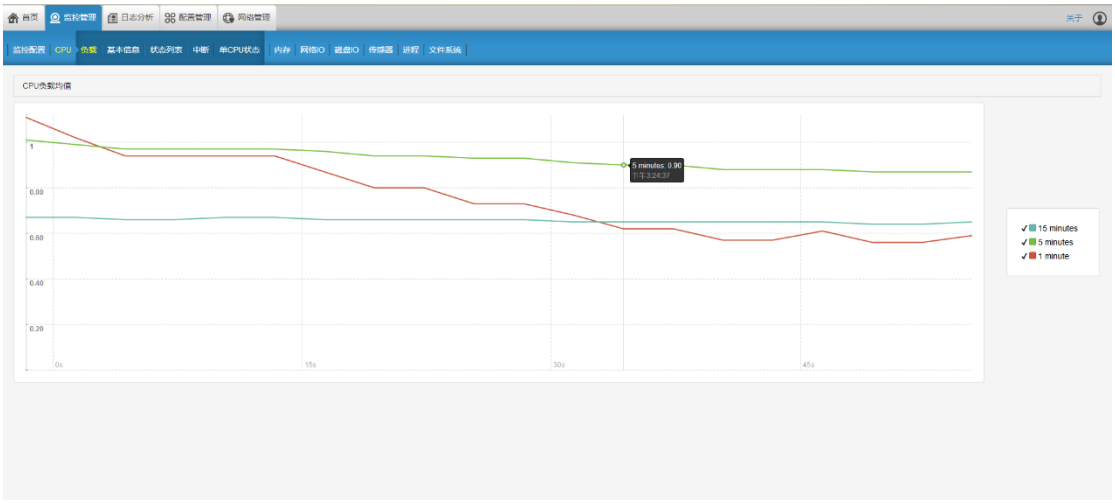


图 30-8 CPU 负载页面

3、CPU 基本信息

点击菜单栏“监控管理”下的“CPU”子菜单中的“基本信息”，跳转到 CPU 基本信息监控页面，获取系统中的 CPU 基本信息均值。监控数据折线图显示最近一分钟内的 CPU 基本信息均值。击图像右侧的参数选项，可以选择想要在图上观察的某个 CPU 参数信息。参数具体含义如下：

（1）user 表示从系统启动到当前时刻，处于用户态的运行时间。不包含 nice 值为负的进程。

（2）nice 表示从系统启动到当前时刻，nice 为负值进程占用的 cpu 时间

（3）system 表示从系统启动到当前时刻，处于内核态的运行时间

（4）idle 表示从系统启动到当前时刻，除了 iowait 外的等待时间

（5）iowait 表示从系统启动到当前时刻，io 等待时间

（6）irq 表示从系统启动到当前时刻，硬中断花费的时间

（7）softirq 表示从系统启动到当前时刻，软中断花费的时间

（8）steal 表示从系统启动到当前时刻，运行其他虚拟环境中的操作系统花费的时间

（9）guest 表示从系统启动到当前时刻，运行在通过 Linux 内核控制的客户操作系统上的虚拟 cpu 的时间

（10）guest_nice 表示从系统启动到当前时刻，运行在通过 Linux 内核控制的客户操作系统上的虚拟 cpu 的时间, nice 为负值进程

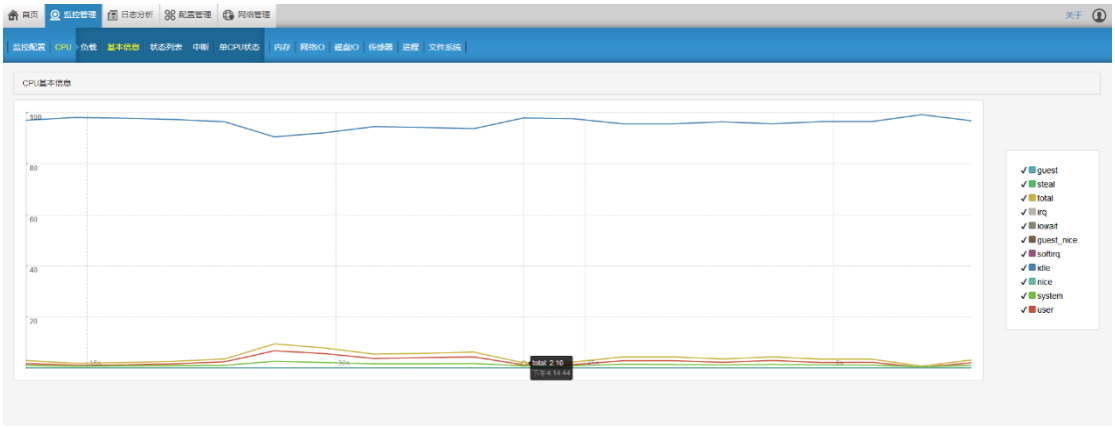


图 30-9 CPU 基本信息页面

4、 CPU 状态列表

点击菜单栏“监控管理”下的“CPU”子菜单中的“状态列表”，跳转到 CPU 状态列表监控页面，获取系统中的每个 CPU 核心基本信息均值的状态列表。表中的监控参数与 30-9 所述的参数完全一致。点击 cpuid 序列号，可分别查看单个 CPU 核心的基本信息。

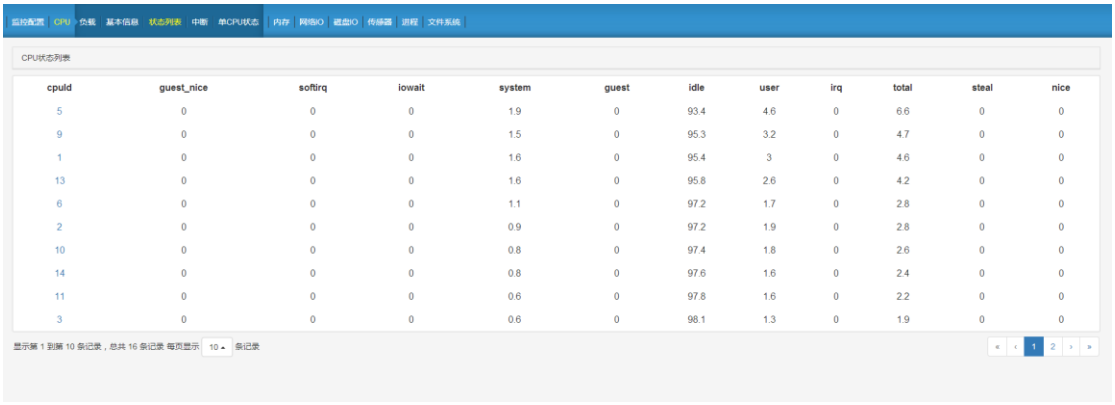


图 30-10 CPU 状态列表页面

5、CPU 中断信息

点击菜单栏“监控管理”下的“CPU”子菜单中的“中断”，跳转到 CPU 中断监控页面，获取系统中的 CPU 中断信息。根据物理 CPU 核心统计中断的次数、可编程中断控制器、设备名称等信息并进行展示，可查看到当前 CPU 核心中断次数以及系统总中断次数。通过选择“当前 CPU 编号”，可以查看不同的物理 CPU 核心中断信息。

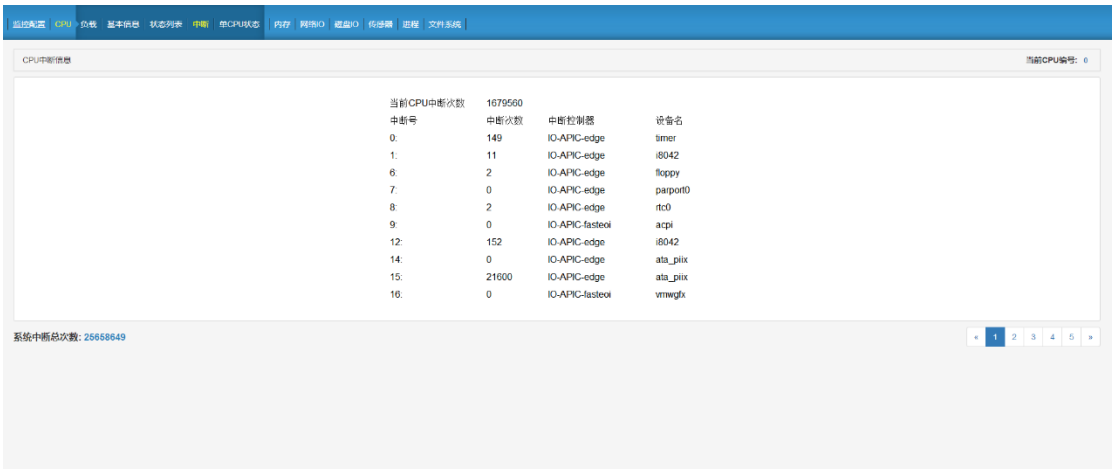


图 30-11 CPU 中断信息页面

6、单 CPU 状态

点击菜单栏“监控管理”下的“CPU”子菜单中的“单 CPU 状态”，跳转到单个 CPU 核心信息页面，获取一分钟内系统的单个物理 CPU 核心的基本信息。监控指标参数同 30-9 所属完全一致。选择“当前 CPU 编号”，可分别查看某个 CPU 核心的基本监控信息。点击图像右侧的参数选项，可以选择想要在图上观察的当前 CPU 核心的参数信息。

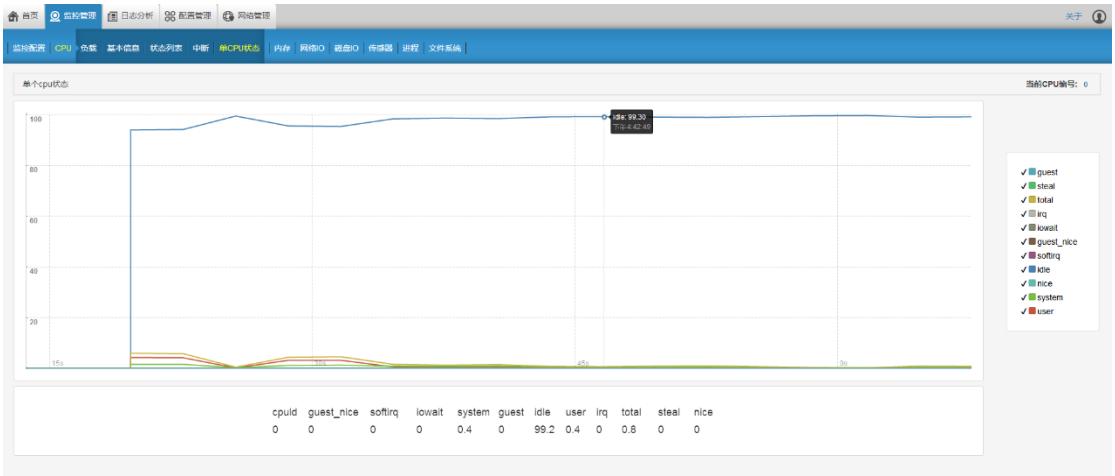


图 30-12 单 CPU 状态页面

30.3.3 内存

本模块主要提供对系统内存的监控，并对部分监控项提供动态监控的功能。

1、内存基本信息

点击菜单栏“监控管理”下的“内存”子菜单中的“基本信息”，跳转到内存基本信息监控页面，采集系统的内存基本信息，并以动态折线图和列表的方式进行展示，图像可展示三分钟内的监控数据信息，每 3 秒钟更新一次。点击右上角的选择按钮可以具体控制各个监控图像的显现。监控信息包括：

total：物理内存总量、

used：使用的物理内存总量、

free：空闲内存的大小、

shared：共享内存的大小、

buffers：作为 buffer cache 的内存、

cached：作为 page cache 的内存

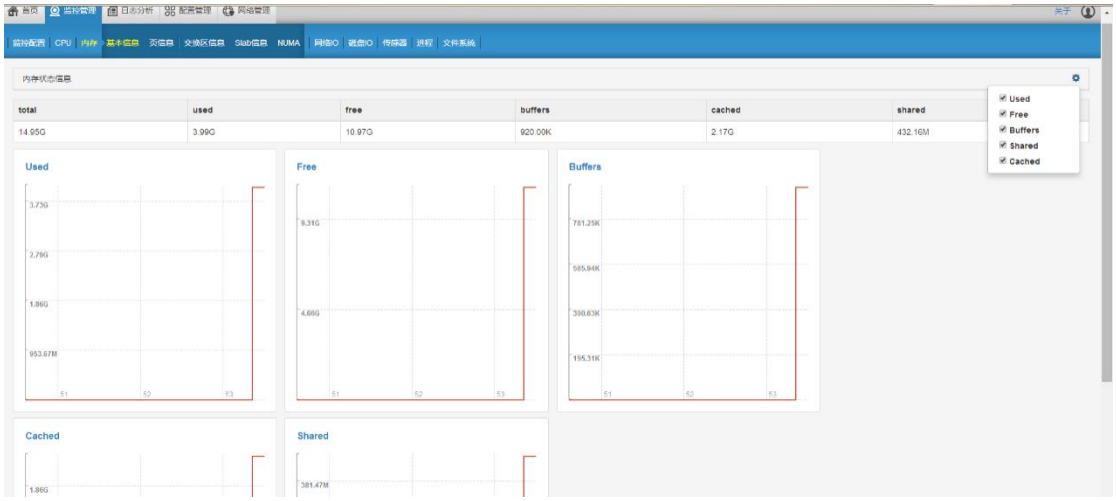


图 30-13 内存基本信息页面

2、页信息

点击菜单栏“监控管理”下的“内存”子菜单中的“页信息”，跳转到内存页信息监控页面，采集系统的内存页信息，并以动态折线图和列表的方式进行展示，图像可展示三分钟内的监控数据信息，每 3 秒钟更新一次。点击右上角的选择按钮可以具体控制各个监控图像的显现。监控信息包括：

pgfree/s:每秒系统中空闲的内存页面，

bufpg/s :每秒系统中用作缓冲区的附加内存页面，

campg/s:每秒系统中高速缓存的附加内存页面，

pgpgin/s:每秒硬盘读取速度，

pgpgout/s:每秒硬盘写出到硬盘速度，

fault/s:每秒内存页面失效（major+minor）数量，

majflt/s:每秒内存页面失效（major）数量，

pgsteal/s:每秒内存页面回收

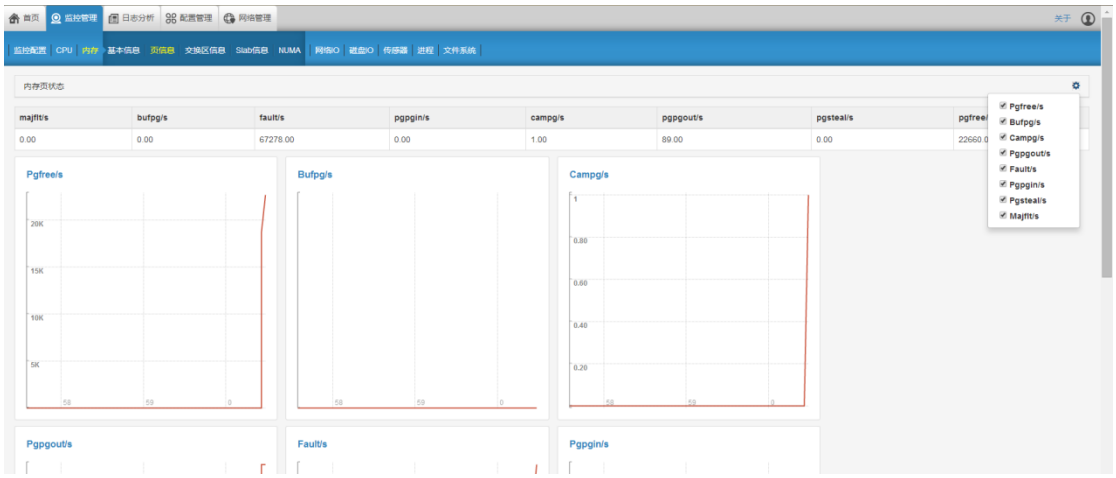


图 30-14 内存页信息页面

3、交换区信息

点击菜单栏“监控管理”下的“内存”子菜单中的“交换区信息”，跳转到内存交换区信息监控页面，采集系统的内存交换区信息，并以动态折线图和列表的方式进行展示，图像可展示三分钟内的监控数据信息，每 3 秒钟更新一次。点击右上角的选择按钮可以具体控制各个监控图像的显现。。监控信息包括：

total：交换区总量、

used：使用的交换区总量、

free：空闲的交换区总量、

pswpin/s：每秒系统换入的交换页面数量,

pswpout/s：每秒系统换出的交换页面数量

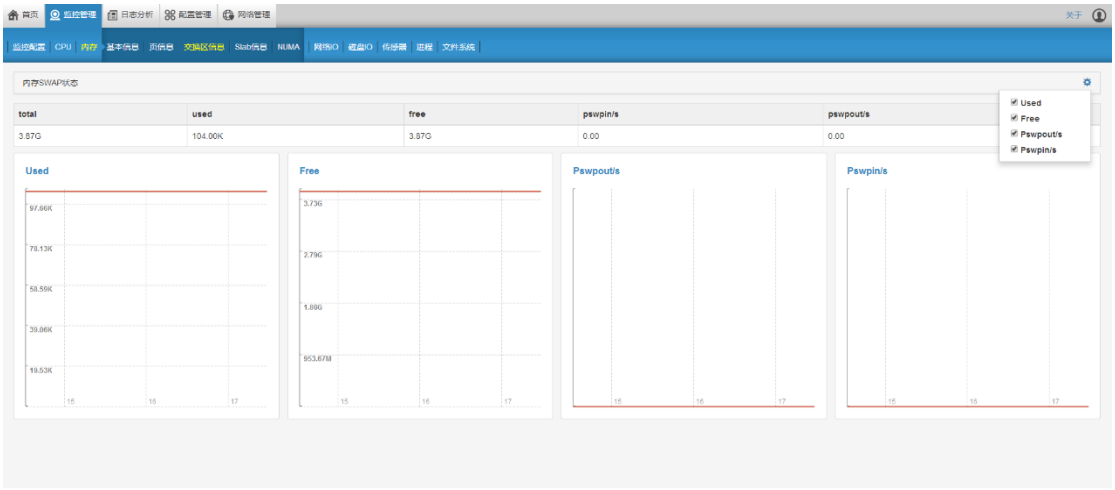


图 30-15 内存交换区信息页面

4、Slab 信息

点击菜单栏“监控管理”下的“内存”子菜单中的“Slab 信息”，跳转到内存 Slab 信息监控页面，采集系统的内存 Slab 信息，并以动态折线图和列表的方式进行展示，图像可展示三分钟内的监控数据信息，每 3 秒钟更新一次。点击右上角的选择按钮可以具体控制各个监控图像的显现。监控信息包括：

name :slab 缓存名称，

objs :slab 缓存数量，

slabs :slab 数量，

obj/slab:每个 slab 的对象，

active:活跃的 slab 数量，

obj_size:对象大小，

use :slab 使用占比

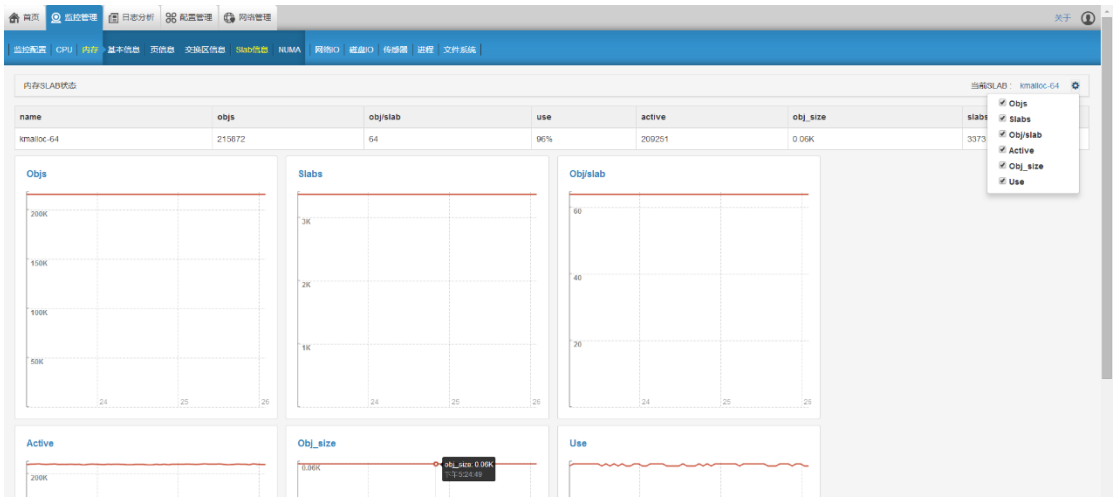


图 30-16 内存 Slab 信息

5、NUMA 信息

点击菜单栏“监控管理”下的“内存”子菜单中的“NUMA”，跳转到内存 NUMA 信息监控页面，采集系统的使用本地内存与远端内存的比例和各节点内存配置的相关信息，并以列表的方式进行展示。

umastat 信息包括：

numa_hit:成功分配至该节点的页面数量，

numa_miss:因预期节点内存不足分配至该节点的页面数量，

numa_foreign:原本预期分配至此节点，而改为分配至其他节点的页面数量，

interleave_hit:成功分配至此节点、交叉存取策略页面数量，

local_node:由节点上进程成功分配至该节点的页面数量，

other_node:由其他节点的进程分配至该节点的页面数量。

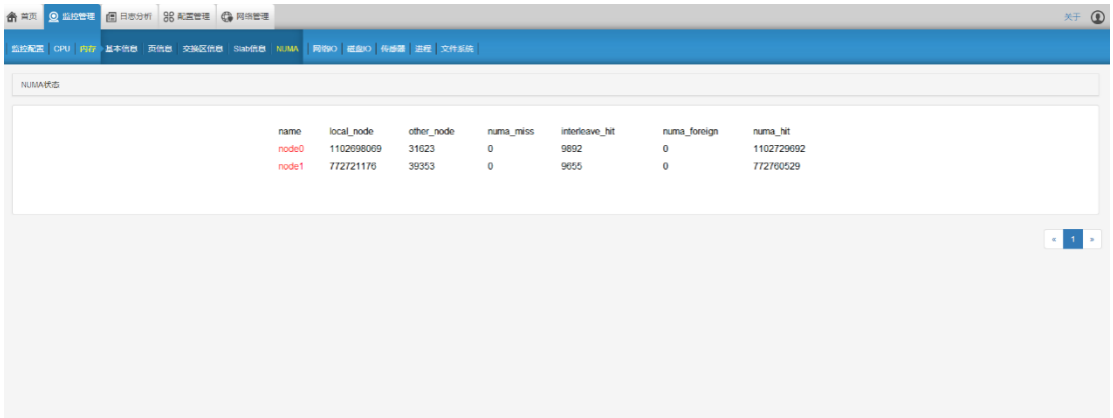


图 30-17 内存 NUMA 信息页面

30.3.4 网络 I0

本模块主要提供对网络 I/O 的监控，并对部分监控项提供动态监控的功能。主要监控内容有：监控网络吞吐-监控网络吞吐列表查看，监控网络吞吐-监控网络吞吐图形查看，监控网络 socket 信息-监控网络 socket 信息列表查看，监控网络 socket 信息-监控网络 socket 信息图形查看，监控网络 IP trafic 信息-监控网络 IP trafic 列表信息，监控网络 IP trafic 信息-监控网络 IP trafic 信息图形查看，监控网络 TCP 信息-监控网络 TCP 信息列表查看，监控网络 TCP 信息-监控网络 TCP 信息图形查看，监控网络 UDP 信息-监控网络 UDP 信息列表查看，监控网络 UDP 信息-监控网络 UDP 信息图形查看。

1、网络吞吐量

点击菜单栏“监控管理”下的“网络 IO”子菜单中的“吞吐量”，跳转到网络吞吐量信息监控页面，采集系统的网络吞吐量，并以图形和列表的形式展现当前主机所有网卡的吞吐量信息。监控图像展示了一分钟内所有网卡的吞吐信息；图像与列表均为每 3 秒钟更新一次数据。点击右上角的选择按钮可以具体控制各个监控图像的显现。

网络吞吐信息参数：

rxpck/s：每秒收到的包个数、

txpck/s：每秒传送的包格式、

rxkb/s：每秒收到的 kb、

txkb/s：每秒传送的 kb、

rxcmp/s：每秒收到的压缩后的包的个数 compressed packets、

txcmp/s：每秒传送的压缩后的包的个数、

rxmcst/s：每秒收到的广播的包的个数 multicast packets、

rxerr/s：每秒收到的坏包的个数 bad packets、

txerr/s：每秒发生的传送包时错误数 、

coll/s：每秒发生的传送包时冲突数、

rxdrop/s：每秒丢掉的已接收的包个数、

txdrop/s：每秒丢掉的已传送的包个数、

txcarr/s：每秒发生的 carrier-error 数，当发 送包时（bad cable）、

rxfram/s：每秒发生的接收包时的帧对齐错误数、

rxfifo/s：每秒发生的接收包时的 FIFO overrun 错误数、

txfifo/s：每秒发生的传送包是的 FIFO overrun 错误数

注：如 rxpck/s：每秒收到的包个数，rxpck/s 表示采集到的监控量，“每秒收到的包个数”表示对监控量的描述

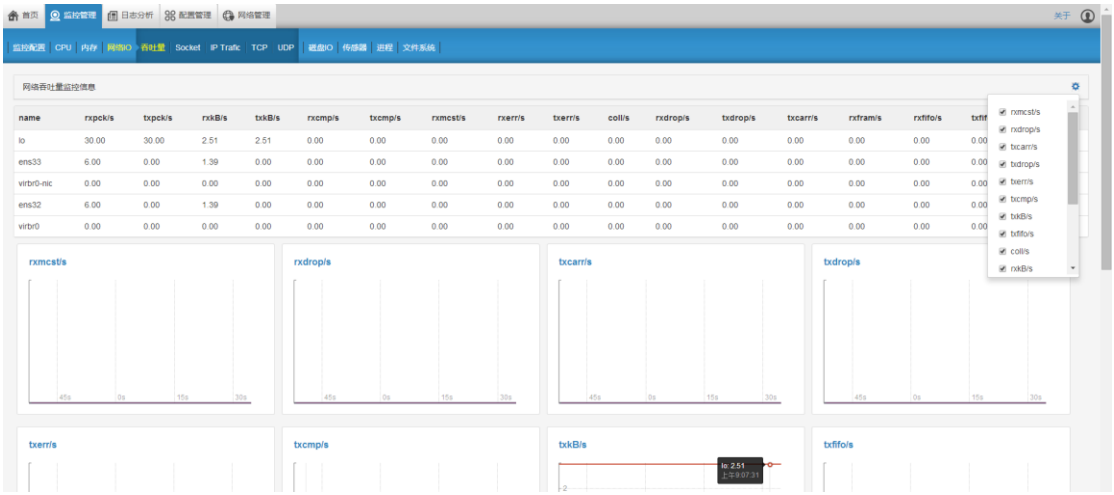


图 30-18 网络 IO 吞吐量监控页面

2、Socket 信息

点击菜单栏“监控管理”下的“网络 IO”子菜单中的“Socket”，跳转到网络 socket 信息监控页面，采集系统的网络 socket，并以图形和列表的形式展现当前主机的网络 socket 信息。监控图像展示了一分钟内的 socket 信息；图像与列表均为每 3 秒钟更新一次数据。点击右侧的参数选择可以具体控制监控图像上每种参数曲线的显现。

totsck：系统使用的 socket 总数、

tcpsck：正在使用的 tcp socket 数、

udpsck：正在使用的 udp socket 数、

rawsck：正在使用的 raw socket 数、

ip-frag：队列中的 ip 碎片树、

tcp-tw：在 TIME_WAIT 状态的 tcp socket 数

注：如 totsck：系统使用的 socket 总数，totsck 表示采集到的监控量，“系统使

用的 socket 总数”表示对监控量的描述。

图 30-19 网络 Socket 信息

3、IP Traffic

点击菜单栏“监控管理”下的“网络 IO”子菜单中的“IP Traffic”，跳转到网络 IP Traffic 信息监控页面，采集系统的网络 IP Traffic，并以图形和列表的形式展现当前主机的网络 IP Traffic 信息。监控图像展示了一分钟内当前主机的 IP Traffic 信息；图像与列表均为每 3 秒钟更新一次数据。点击右侧的参数选择可以具体控制监控图像上每种参数曲线的显现。

irec/s：每秒通过接口收到的报文总数 、

fwddgm/s：每秒通过接口收到的需要转发的报文总数、

idel/s：每秒收到的成功传递到 ip 用户协议的报文树、

orq/s：每秒要求传送出去的 ip 报文数、

asmrq/s：每秒接收的要求重新组合的 ip 碎片数、

asmok/s：每秒成功重新组合的 ip 报文数 、

fragok/s：每秒成功分割的 ip 报文数 、

fragcrt/s：每秒通过分割产生的 ip 碎片数 、

ihdrerr/s：每秒因为 ip 头错误丢弃的接收报文数、

iadrerr/s：每秒因为目的地址错误丢弃的接收报文数、

iukwnpr/s：每秒因为未知或不支持的协议丢弃的接收报文数、

idisc/s：每秒未发生错误但是丢弃的接收报文数，不包括在等待重新组合的时候

被丢弃的、

odisc/s：每秒未发生错误但是丢弃的传送报文数，包括 fwddgm/s 中满足此情况被丢弃的、

onort/s：每秒因为路由错误被丢弃的传送报文数、

asmf/s：每秒在进行重新组合报文碎片时发生的错误数、

fragf/s：每秒因为无法分割而丢弃的报文数，因为它们的 don't frag 标志置位

注：如 irec/s：每秒通过接口收到的报文总数，irec/s 表示采集到的监控量，“每秒通过接口收到的报文总数”表示对监控量的描述

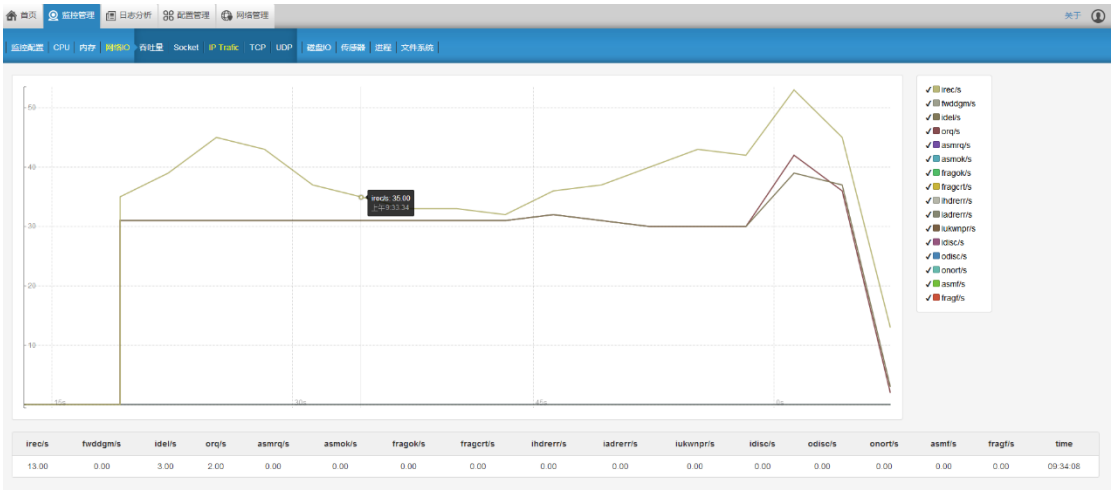


图 30-20 网络 IP Traffic 信息

4、TCP

点击菜单栏“监控管理”下的“网络 IO”子菜单中的“TCP”，跳转到网络 TCP 信息监控页面，采集系统的网络 TCP，并以图形和列表的形式展现当前主机的网络 TCP 信息。监控图像展示了一分钟内当前主机的 TCP 信息；图像与列表均为每 3 秒钟更新一次数据。点击右侧的参数选择可以具体控制监控图像上每种参数曲线的显现。

active/s：每秒 tcp 连接从 CLOSE 状态转换到 SYN_SENT 状态的次数、

passive/s：每秒 tcp 连接从 LISTEN 状态转换到 SYN_RCVD 状态的次数、

iseg/s：每秒收到的 segment 数量、

oseg/s：每秒传送的 segment 数量、

atmptf/s：每秒 tcp 链接从 SYN_SENT/RCVD 到 CLOSE/LISTEN 状态的直接转换

次数、

estres/s：每秒 tcp 链接从 ESTABLISHED 或 CLOSE_WAIT 到 CLOSED 状态的直接转换次数、

retrans/s：每秒重传的 tcp segments 数、

isegerr/s：每秒收到的错误的 segment 数量、

orst/s：每秒发送的包含 RST 标志的 tcp segment 数量

注：如 iseg/s：每秒收到的 segment 数量，iseg/s 表示采集到的监控量，“每秒收到的 segment 数量”表示对监控量的描述。

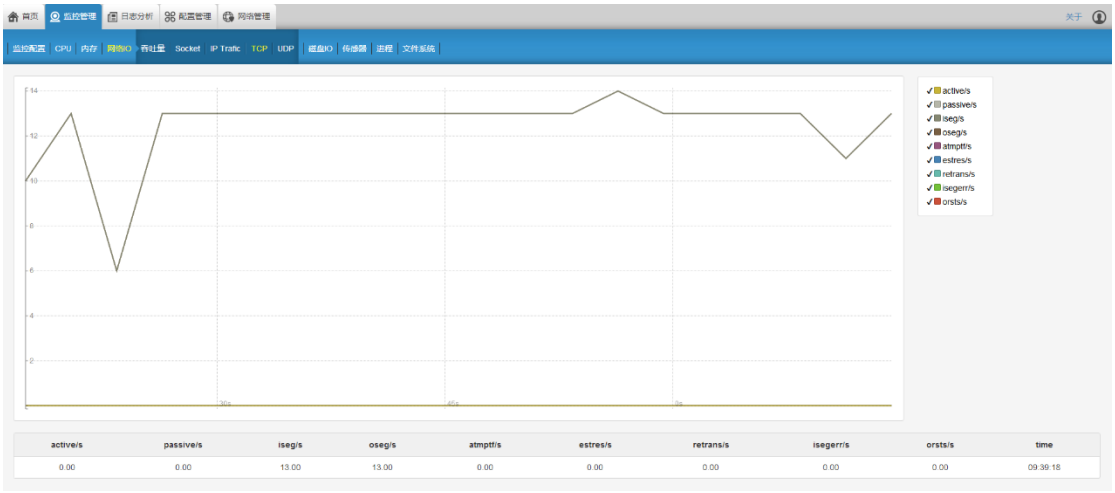


图 30-21 网络 TCP 信息

5、UDP

点击菜单栏“监控管理”下的“网络 IO”子菜单中的“UDP”，跳转到网络 UDP 信息监控页面，采集系统的网络 UDP，并以图形和列表的形式展现当前主机的网络 UDP 信息。监控图像展示了一分钟内当前主机的 UDP 信息；图像与列表均为每 3 秒钟更新一次数据。点击右侧的参数选择可以具体控制监控图像上每种参数曲线的显现。

idgm/s：每秒传递到 udp 用户的 udp 报文数、

odgm/s：每秒发送的 udp 报文数、

noport/s：每秒接收到的没有应用端口的 udp 报文数、

idgmerr/s：每秒接收到的不能传递的 udp 报文数，不包括没有应用端口的

注：如 odgm/s：每秒发送的 udp 报文数，odgm/s 表示采集到的监控量，“每秒

发送的 udp 报文数”表示对监控量的描述。

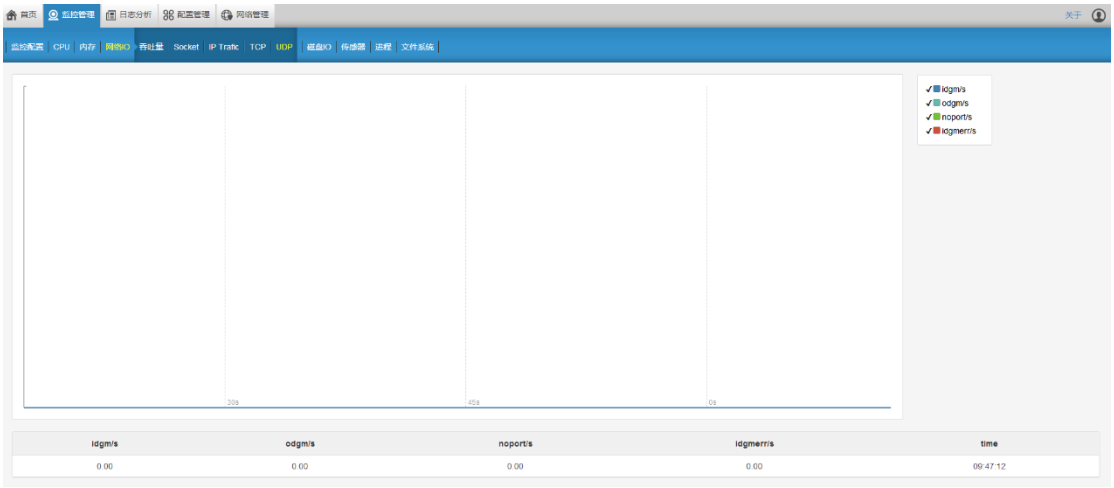


图 30-22 网络 UDP 信息

30.3.5 磁盘 IO

本模块用于查看当前系统的磁盘 IO 的信息，用户可以通过本模块查看磁盘 IO 信息、磁盘网络块设备信息。

1、磁盘信息

点击菜单栏“监控管理”下的“磁盘 IO”子菜单中的“磁盘信息”，跳转到磁盘信息监控页面，采集系统的磁盘信息，并以图形和列表的形式展现。磁盘信息监控图像和列表展示了一分钟内当前主机磁盘信息参数，每 3 秒钟更新一次；磁盘容量信息列表展示了磁盘名称、容量以及类型信息。点击右上角参数选择可以具体控制某种参数图像的显现。

tps：每秒提交给物理设备的传输数、

rtps：每秒提交给物理设备的读传输数、

wtps：每秒提交给物理设备的写传输数、

bread/s:每秒从设备读出的数据块数、

bwrtn/s:每秒写到设备的数据块数

注：如 tps：每秒提交给物理设备的传输数，tps 表示采集到的监控量，“每秒提交给物理设备的传输数”表示对监控量的描述。

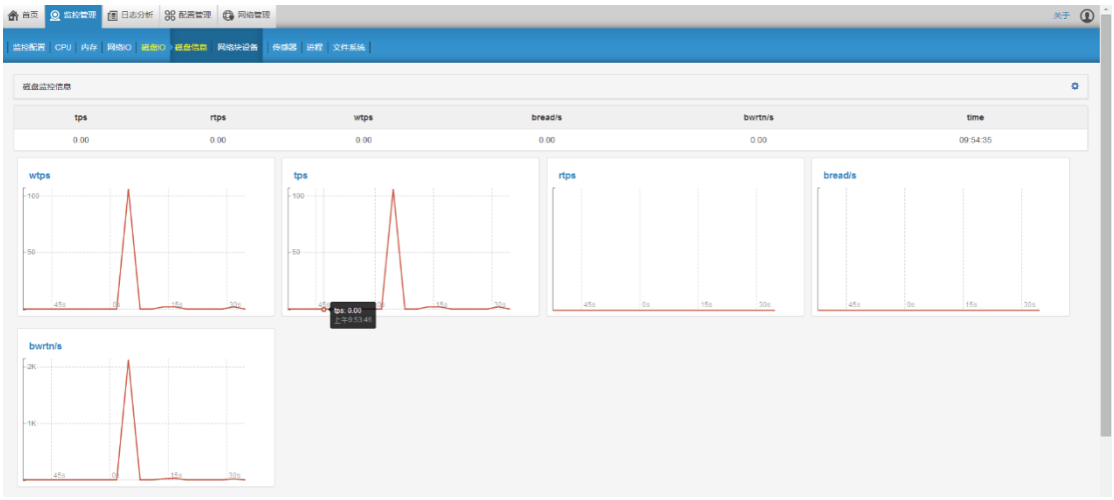


图 30-23 磁盘监控信息

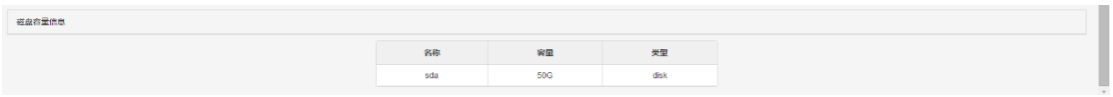


图 30-24 磁盘容量信息

2、网络块设备

点击菜单栏“监控管理”下的“磁盘 IO”子菜单中的“网络块设备”，跳转到磁盘网络块设备监控页面，采集系块设备信息，并以图形和列表的形式展现。网络块设备监控图像和列表展示了一分钟内当前主机磁盘信息参数，每 3 秒钟更新一次；点击右上角参数选择可以具体控制某种参数图像的显现。

tps:每秒提交给物理设备的传输数、

rd_sec/s:每秒读出的扇区数、

wr_sec/s:每秒写到设备的扇区数、

avgrq-sz:提交到设备的请求的平均大小（单位为扇区）、

avgqu-sz:提交到设备的请求的队列平均长度、

await:提交到设备的 IO 请求平均完成的时间（ms），包括等待和服务时间、

%util:设备带宽利用率、

rrqm/s:每秒合并的读请求、

wrqm/s:每秒合并的写请求

注：如 tps：每秒提交给物理设备的传输数，tps 表示采集到的监控量，“每秒提交给物理设备的传输数”表示对监控量的描述。

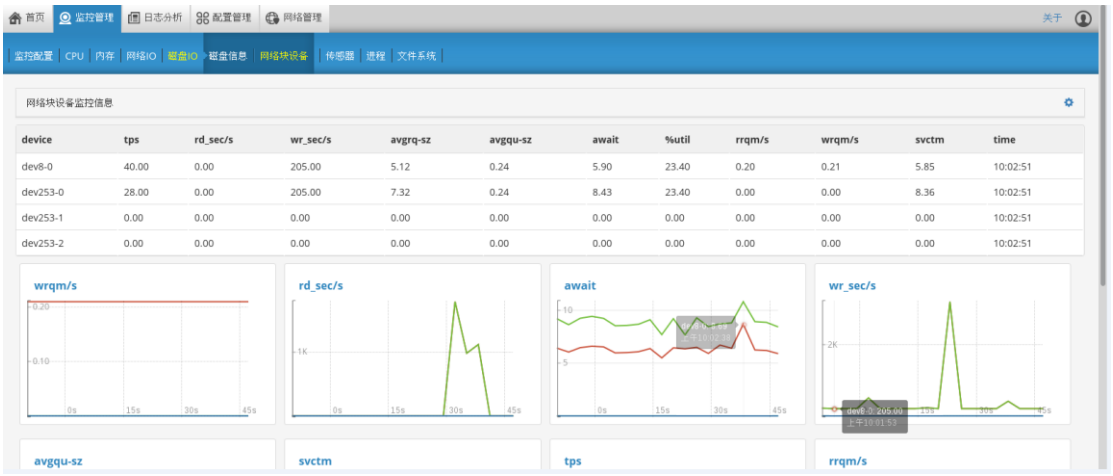


图 30-25 网络块设备监控页面

30.3.6 传感器

本模块用于查看当前系统的传感器相关信息，用户可以通过本模块查看 CPU 温度、风扇转速相关信息。

1、CPU 温度列表

该模块获取系统中的 CPU 温度信息，并呈现给用户。该模块对于温度区间进行区分，通过不同颜色标识当前温度范围，如图 30-26 所示，当温度擦后果 75℃表示为红色，超过 50℃且低于 75℃表示为黄色，低于 50℃为绿色。同时该模块为每个独立的 CPU 提供详细温度监控，用户可通过点击色块进入详细温度监控，如图 30-27 所示详细温度监控页面的左上角实时显示服务器某 CPU 温度，温度数值实时变化的同时，本模块自动对历史数值收集汇总绘制于本页面中央绘图区，生成动态折线图，便于用户观察时间段内 CPU 温度情况，分析服务器情况。

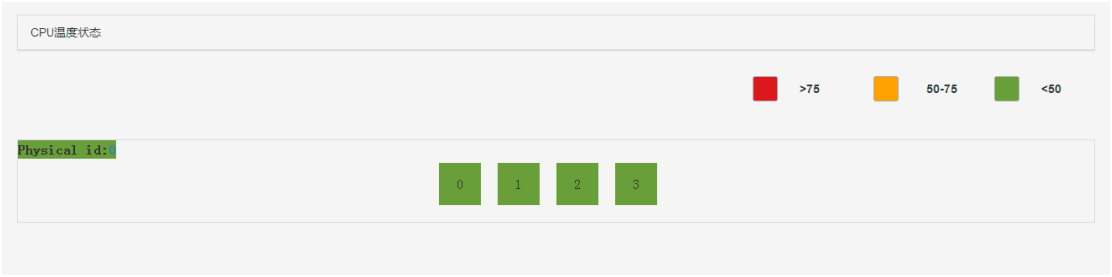


图 30-26 CPU 温度矩阵图

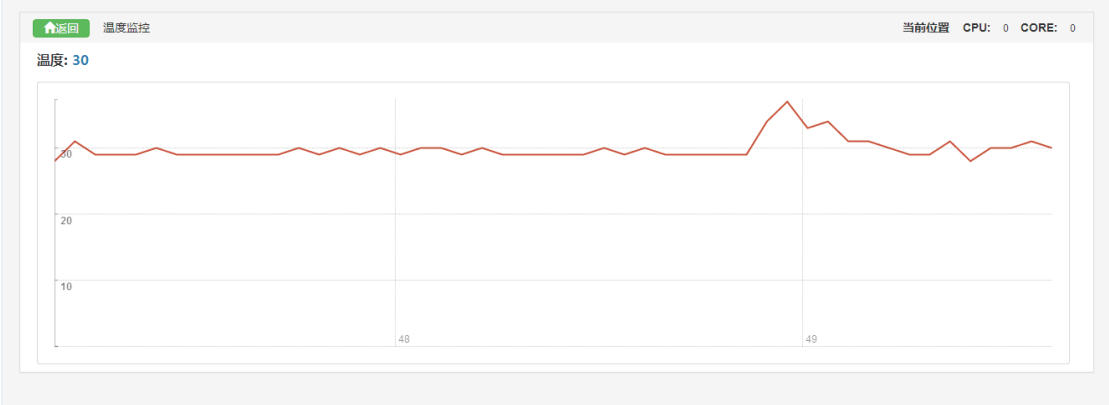


图 30-27 CPU 温度折线图

2、风扇

本模块可以监视实时的风扇传感器数据。在本模块的左上角实时显示服务器某风扇转速，服务器风扇一般存在多个，可通过右侧的下拉列表选择要监控的风扇。风扇转速数值实时变化的同时，本模块自动对历史数值收集汇总绘制于本模块的中央绘图区，生成动态折线图，便于用户观察时间段内风扇情况，分析服务器情况。如图 30-28。

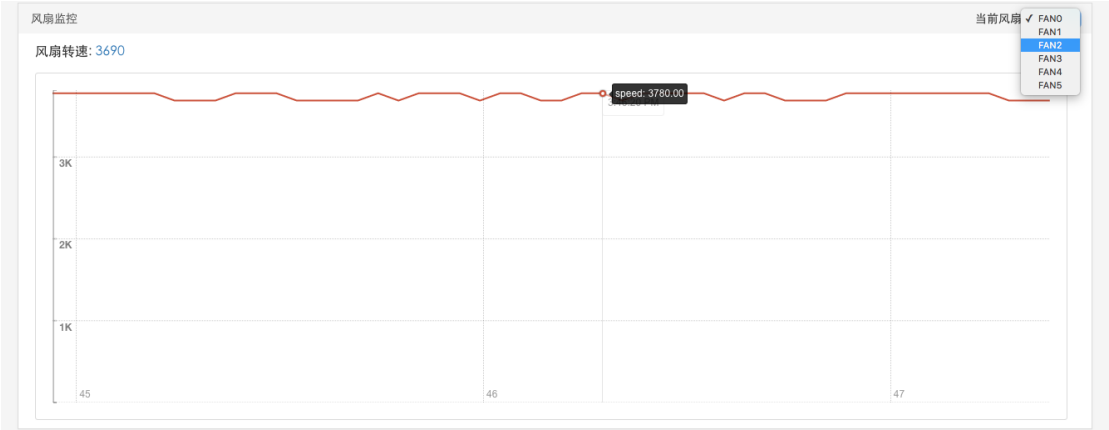


图 30-28 风扇

30.3.7 进程

本模块用于监控进程的信息，用户可以通过本模块查看进程的基本信息、进程的生命周期、进程 IO 信息、单个进程打开文件信息、单个进程的线程信息。以列表的方式展示上述信息给用户，并且实时同步最新的服务器进程信息反向更新该列表。列表中直观体现了进程的 pid、user、name、pcpu、pmem、state、time、

pri、vsz、rss、psr，并且提供四个详细信息查看按钮供用户查看单个进程相关的文件、线程、IO 信息、生命周期。如图 30-29。

图 30-29 进程

1、文件

本页面显示了相关进程打开的文件信息，以列表方式列举了文件的 name、type、node、user、device、command、fd、size。

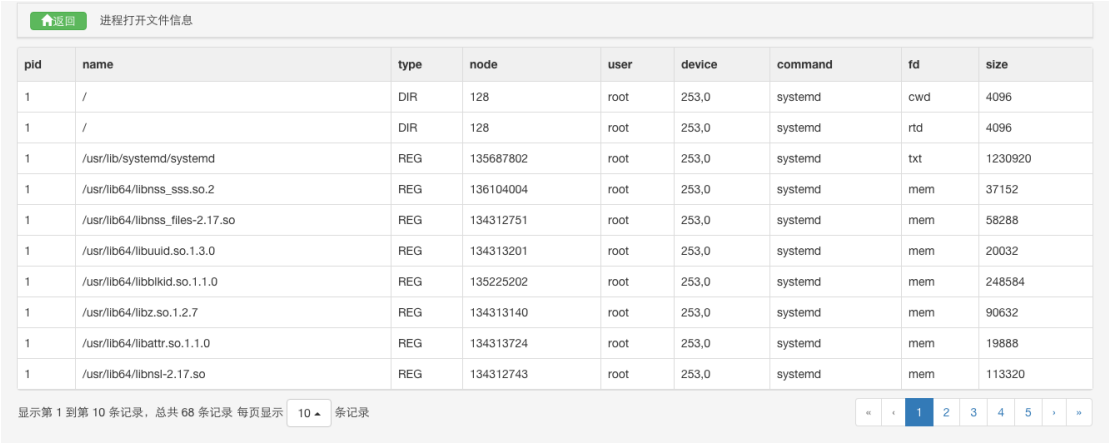


图 30-30 文件信息

2、线程

本页面显示了线程信息列表，以列表方式列举线程的 spid、cmd、time。

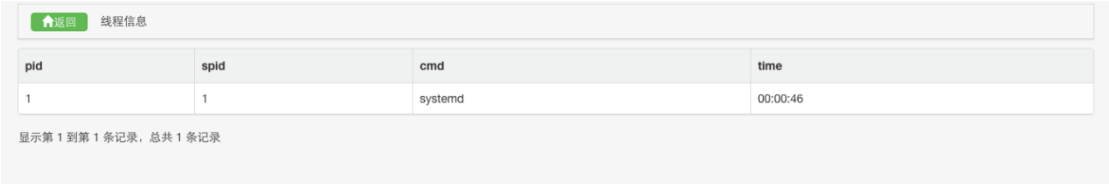


图 30-31 线程

3、IO 信息

本页面显示了相关进程的 IO 信息，以列表方式列举了 read_bytes、write_bytes、cancelled_write_bytes。

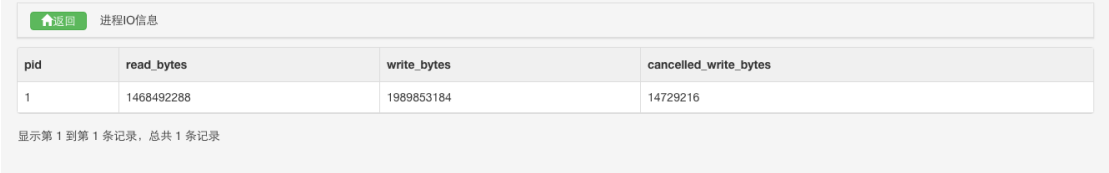


图 30-32 IO

4、生命周期

本页面显示了相关进程的生命周期，以列表方式列举了当前进程 name、flag、sched、stime、etime、nice。

图 30-33 进程生命周期

30.3.8 文件系统

文件系统模块以列表形式展现了当前服务器系统的文件系统信息，列举了文件系统名称、挂载点、总空间、已用空间、空闲空间、已用百分比。方便用户直观监控服务器的文件系统信息和各个文件系统的使用情况。

图 30-34 文件系统

30.4 日志分析

UNICORN 系统收集了系统的日志、UNICORN 系统日志、内核日志，通过该日志分析模块展示给用户，方便用户实时查看。

30.4.1 系统日志

系统日志包含了服务器操作系统各项日志，并且通过列表的形式分组显示给用户，如图 30-35，通过点击查看按钮，可以进入组日志和和查看具体的日志项，同时提供了日志内搜索筛选功能，方便用户快速定位到日志具体位置，利于高效分析。如图 30-36。

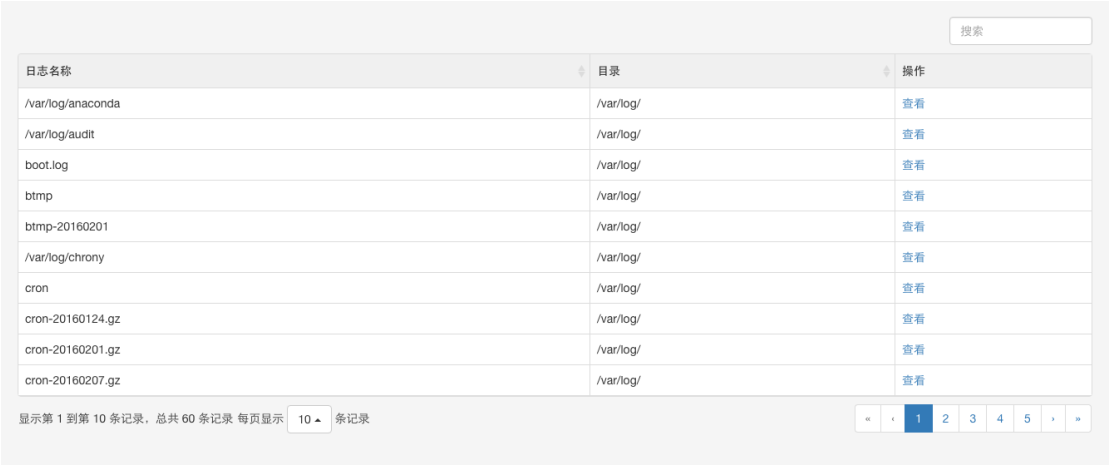


图 30-35 系统日志

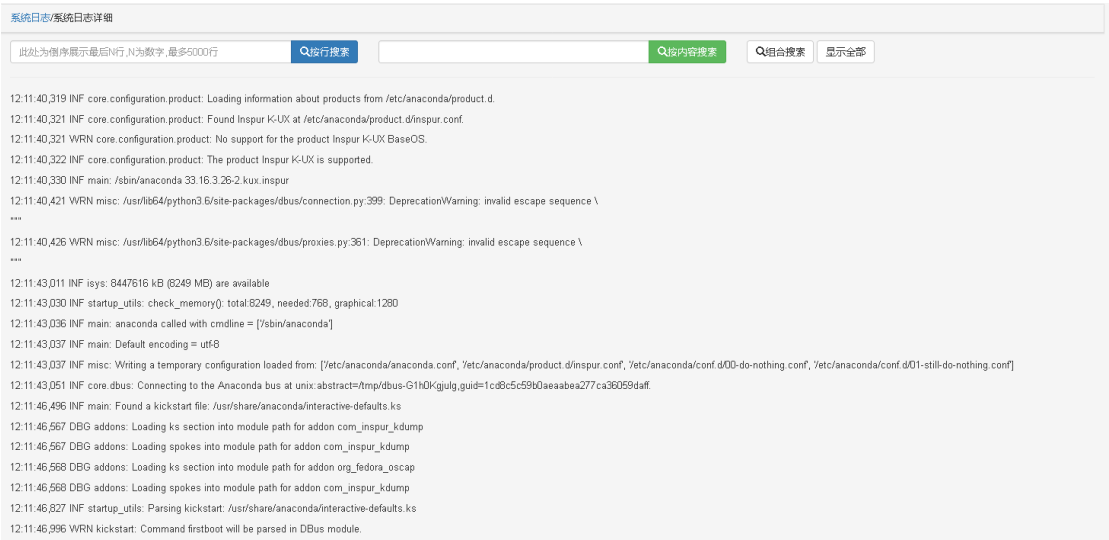
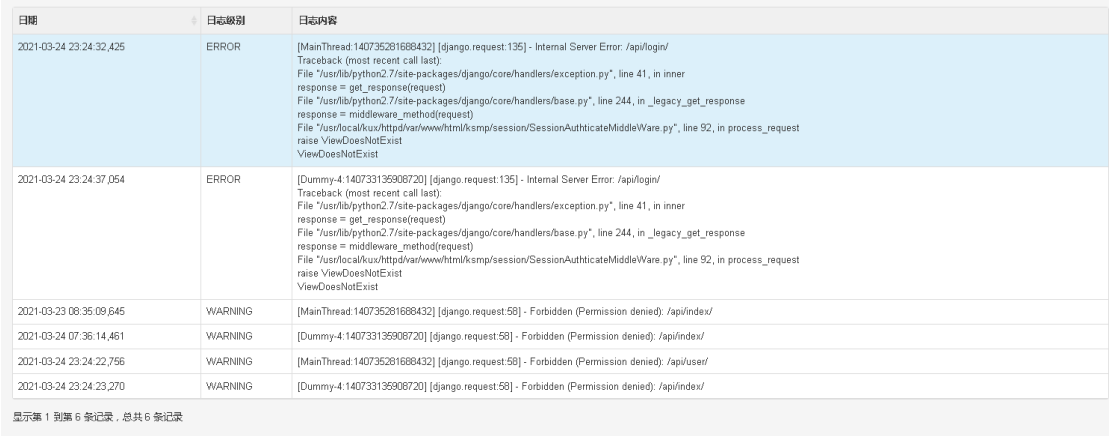


图 30-36 系统日志

30.4.2 UNICORN 日志

该子模块记录并展示了 UNICORN 的运行日志信息，通过详细记录 UNICORN 系统的操作、警告、错误日志，方便用户及时了解 UNICORN 系统运行情况，查看操作情况。以列表的形式展现了日志日期、日志级别、日志详细内容。如图 30-37。



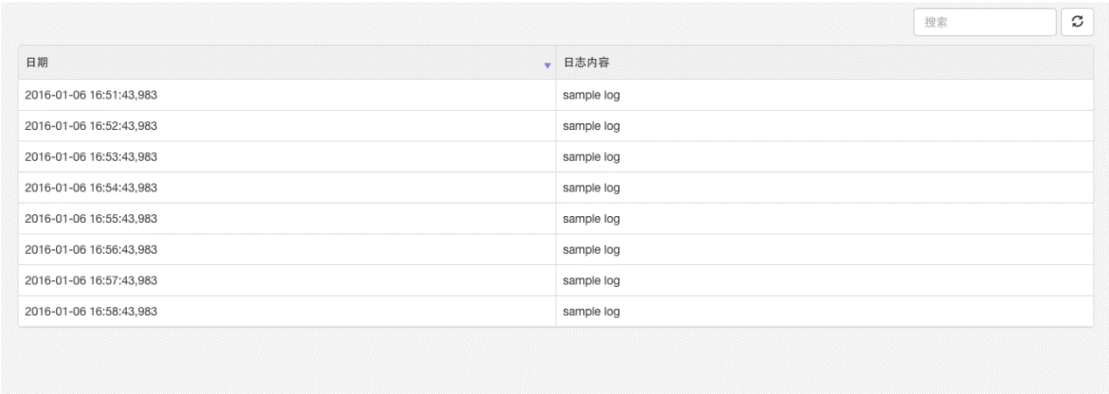
日期	日志级别	日志内容
2021-03-24 23:24:32,425	ERROR	[MainThread:140735281688432] [django.request:135] - Internal Server Error: /api/login/ Traceback (most recent call last): File "/usr/lib/python2.7/site-packages/django/core/handlers/exception.py", line 41, in inner response = get_response(request) File "/usr/lib/python2.7/site-packages/django/core/handlers/base.py", line 244, in _legacy_get_response response = middleware_method(request) File "/usr/local/kux/httpd/var/www/html/ksmp/session/SessionAuthticateMiddleWare.py", line 92, in process_request raise ViewDoesNotExist ViewDoesNotExist
2021-03-24 23:24:37,054	ERROR	[Dummy-4:140733135908720] [django.request:135] - Internal Server Error: /api/login/ Traceback (most recent call last): File "/usr/lib/python2.7/site-packages/django/core/handlers/exception.py", line 41, in inner response = get_response(request) File "/usr/lib/python2.7/site-packages/django/core/handlers/base.py", line 244, in _legacy_get_response response = middleware_method(request) File "/usr/local/kux/httpd/var/www/html/ksmp/session/SessionAuthticateMiddleWare.py", line 92, in process_request raise ViewDoesNotExist ViewDoesNotExist
2021-03-23 08:35:09,645	WARNING	[MainThread:140735281688432] [django.request:58] - Forbidden (Permission denied): /api/index/
2021-03-24 07:36:14,461	WARNING	[Dummy-4:140733135908720] [django.request:58] - Forbidden (Permission denied): /api/index/
2021-03-24 23:24:22,756	WARNING	[MainThread:140735281688432] [django.request:58] - Forbidden (Permission denied): /api/user/
2021-03-24 23:24:23,270	WARNING	[Dummy-4:140733135908720] [django.request:58] - Forbidden (Permission denied): /api/index/

显示第 1 到第 6 条记录，总共 6 条记录

图 30-37 unicorn 日志

30.4.3 内核日志

该子模块记录了系统内核日志，包含系统的设备信息启动和操作过程系统记录的日志。同样以列表形式展现。如图 30-38。



搜索

日期	日志内容
2016-01-06 16:51:43,983	sample log
2016-01-06 16:52:43,983	sample log
2016-01-06 16:53:43,983	sample log
2016-01-06 16:54:43,983	sample log
2016-01-06 16:55:43,983	sample log
2016-01-06 16:56:43,983	sample log
2016-01-06 16:57:43,983	sample log
2016-01-06 16:58:43,983	sample log

图 30-38 内核日志

该模块需要在服务器后端配置才能使用，配置方法如下：

a. 编辑/etc/rsyslog.conf 文件，添加 kern.* /var/log/kern.log

b. 新建日志文件 touch /var/log/kern.log

c. 重启服务：systemctl restart rsyslog

30.5 配置管理

30.5.1 用户和群组

该模块提供用户和组的基本管理功能。包括添加用户、删除用户、编辑用户、添加群组、删除群组、编辑群组的功能。

1、用户管理

用户管理界面以列表形式列出系统中全部用户，并按照系统用户、普通用户、系统管理员对用户进行分类显示，如图 30-39。

图 30-39 用户管理

（1）、添加用户。点击添加用户按钮，在添加用户界面输入用户名、密码等信息，然后点击添加按钮。其中用户名需要满足一定的要求，且用户名、用户全称、密码为必填项。

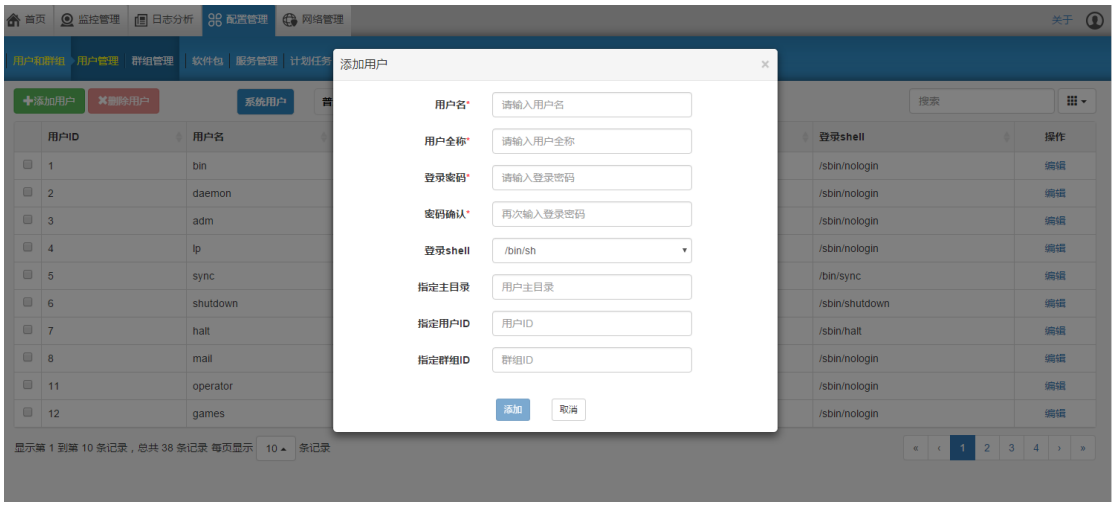


图 30-40 添加用户

（2）、编辑用户。点击用户后面的“编辑”按钮，在弹出的模态框中可以对用户名、用户全称、密码和登录 shell 进行修改。

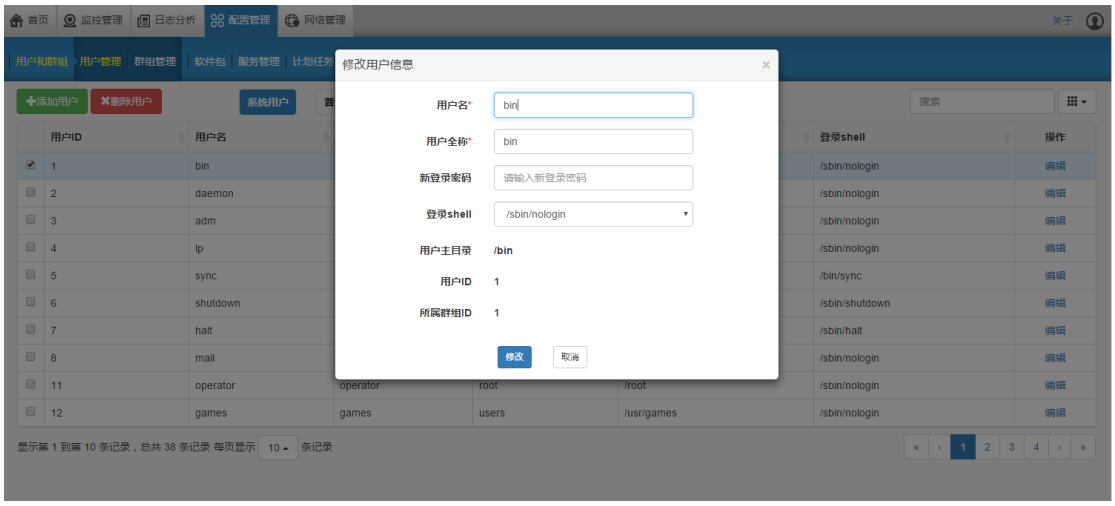


图 30-41 编辑用户

（3）、删除用户。选择要删除用户前面的选择框，然后点击“删除用户”按钮。

2、群组管理

点击菜单栏的“群组管理”进入群组界面。

（1）、添加群组。点击“添加群组”按钮，在弹出框中填写群组名和群组 ID，点击“添加”按钮，其中群组名为必填项。

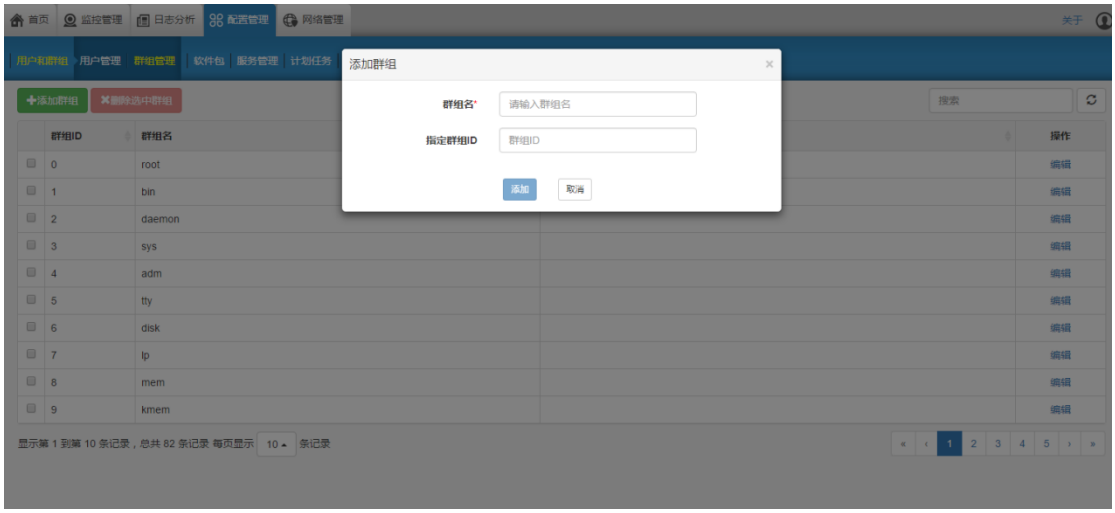


图 30-42 添加群组

（2）、编辑群组。点击群组后面的“编辑”按钮，弹出编辑群组界面。左边的框中为系统中没在组中的用户列表，右边的框为群组里的用户列表，选择用户后（按住‘Ctrl’键进行多选）点击“添加入组”和“移除用户”按钮可以将用户添加入组或者把组中用户删除。

图 30-43 编辑群组

（3）、删除群组。点击列表中想要删除的群组，该群组前面的复选框会被选中，然后点击“删除选中群组”按钮。

30.5.2 软件包

本模块提供软件包查找、显示、删除和安装功能。

1、管理

（1）、进入软件包管理界面，会以表格形式显示系统中所有 RPM 软件包的分组表格，如图 30-44，点击分组会显示该分组下的子分组或者软件包，点击表格上方的软件包导航可以返回到全部分组或上级分组。

（2）、见图 30-44，在搜索框中填入字符串，点击“搜索”按钮，会在下面表格中显示系统中所有相关软件包（调用系统命令“rpm –qa | grep ”进行搜索）。

（3）、搜索结果中的软件包或者分组下的软件包，点击选中后，再点击上面的“卸载”按钮可以卸载软件包，对有依赖的软件包，在弹出的确认对话框中选择“忽略可能的依赖信息”，可以强行卸载该软件包，如图 30-45。

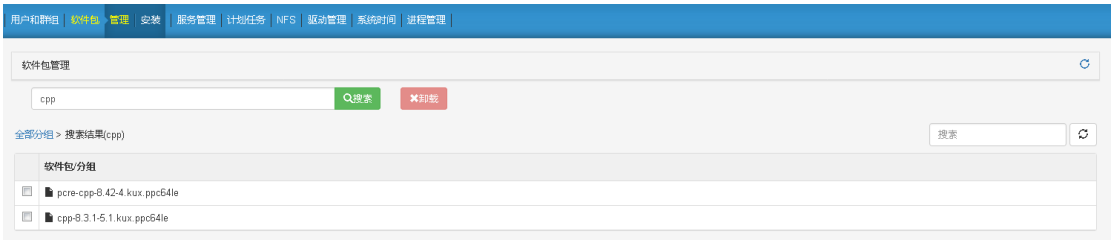


图 30-44 软件包管理

图 30-45 软件包删除

2、安装

（1）、进入软件包安装界面，如图 30-46，会以表格形式显示出系统“/home”目录下所有内容，包括文件和文件夹。如果为子文件夹，点击即可进入子文件夹，并显示文件夹下的所有内容；如果文件是 RPM 软件包，则在包名后面会有“安装”按钮，点击即可安装该软件包。

（2）、见图 30-46，界面上方有上传功能，点击“浏览”按钮选择本地 RPM 包，点击“上传文件”按钮会将软件包上传到服务器固定目录（“/home/filetrans”）下。

如果文件名不符合 RPM 包命名规则将无法上传。

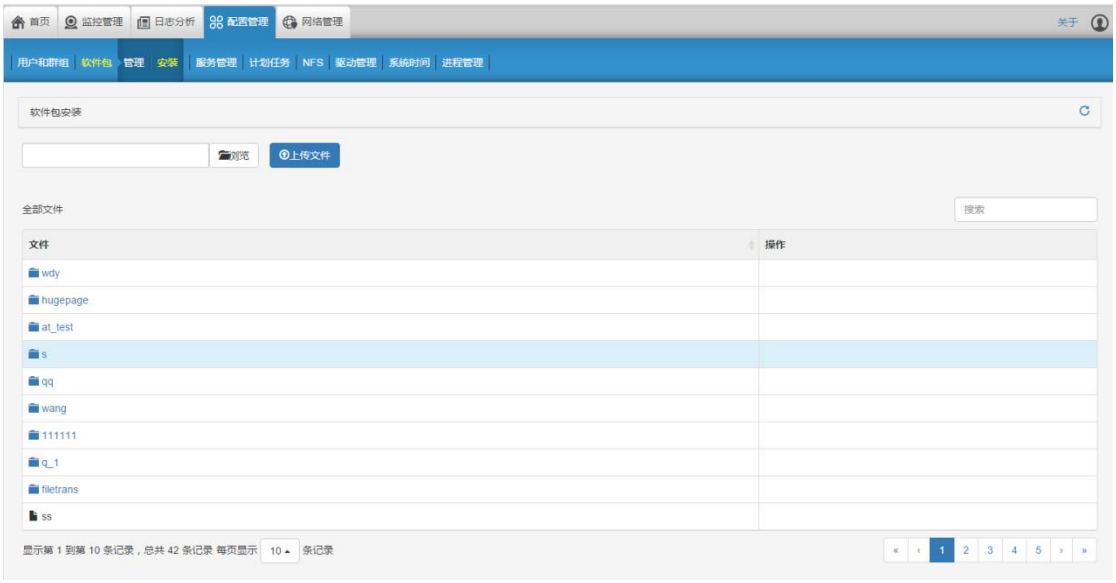


图 30-46 软件包安装

30.5.3 服务管理

服务管理界面显示系统中所有服务的信息，包括服务名、服务描述、加载状态、SUB 状态、当前状态和是否开机启动，如图 30-47。

1、改变服务的状态。在服务后面的“状态”栏中，点击状态弹出下拉列表，可以选择对该服务进行 start、stop 或 restart 操作。

2、设置服务开机启动。在服务后面的“开机启动”栏中，点击弹出下拉列表，可以选择 enable（开机启动）和 disable（开机不启动）操作。

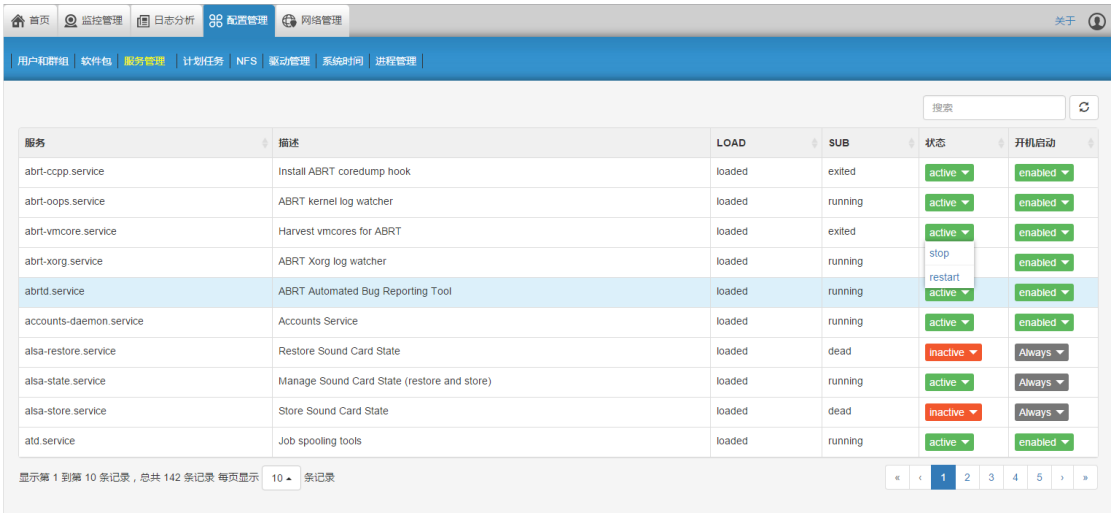


图 30-47 服务管理

30.5.4 计划任务

本模块提供计划任务的添加、删除和查看操作。

进入计划任务界面，如图 30-48，以列表形式显示系统中所有的计划任务信息，包括任务号、人物的执行用户、执行时间和对任务的查看操作按钮。

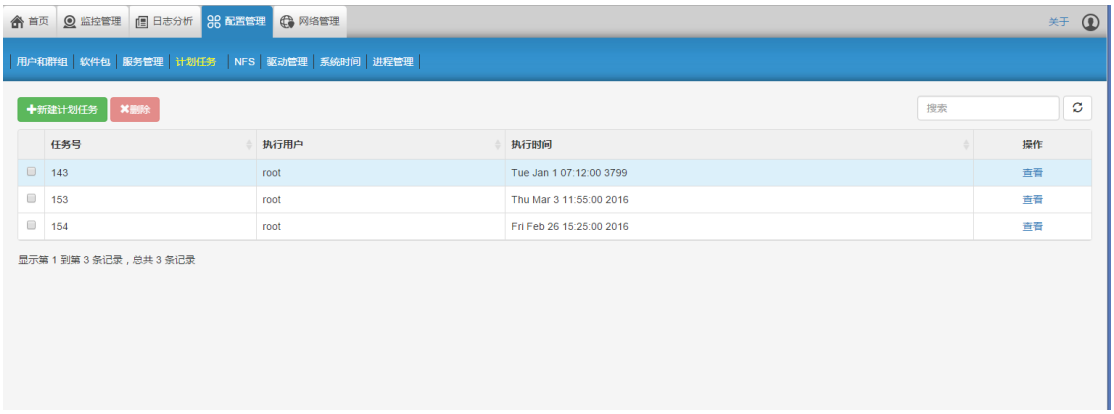


图 30-48 计划任务

1、新建计划任务。见图 30-49，点击“新建计划任务”按钮，进入新建任务界面。填写创建任务的信息，其中以此用户执行、执行日期、在该目录执行、执行命令都为必填项。点击“以此用户执行”后面的输入框，会在弹出框中显示系统当前所有用户的列表，点击用户名便可将该用户输入。填写完成后点击“创建”按钮。

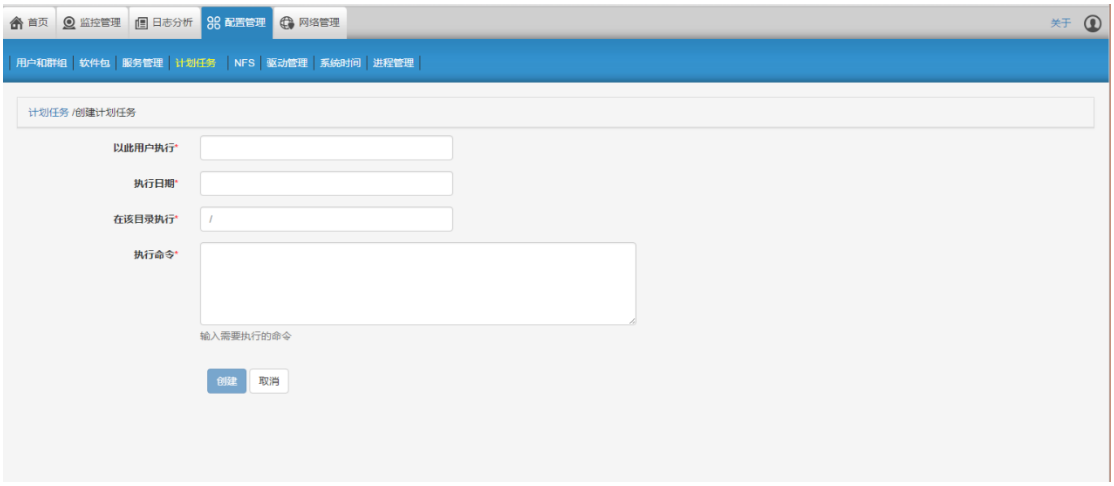


图 30-49 新建计划任务

2、查看计划任务。见图 30-48，在计划任务列表中点击任务后面的“查看”按

钮，即可进入任务详情界面，如图 30-50。

图 30-50 查看计划任务

3、删除计划任务。在任务列表中点击要删除的任务便可将任务选中，然后点击“删除”按钮即可。

30.5.5 NFS

本模块提供网络文件系统的管理功能，用户可以添加和删除一个共享目录，也可以切换一个已有共享目录的状态（激活或关闭）。

1、切换状态。进入 NFS 界面，如图 30-51，以列表的形式显示系统中所有的共享目录信息，包括目录的路径、该目录的共享规则以及共享目录状态。列表最后一列是状态开关，点击便可以打开/关闭共享目录状态。

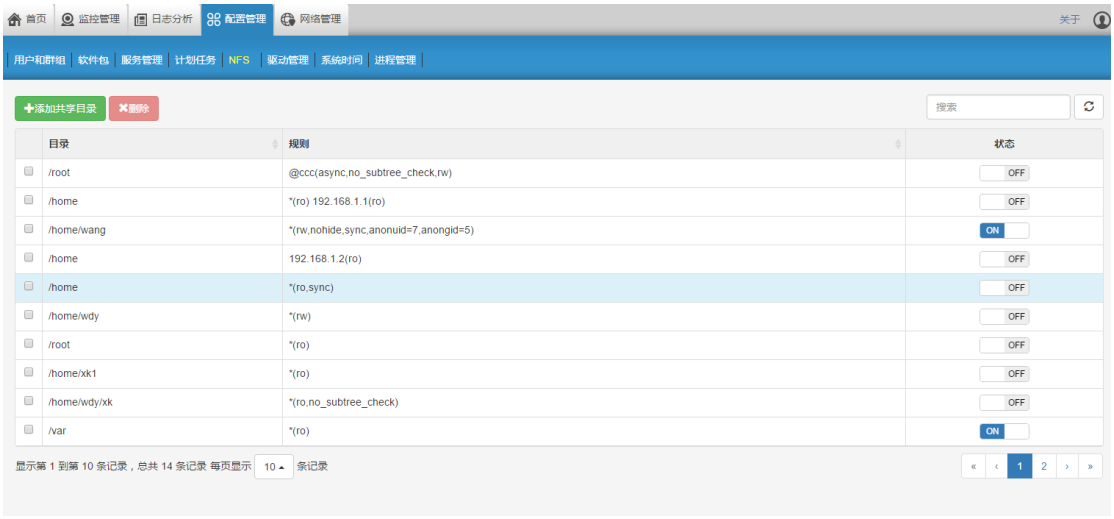


图 30-51 NFS

2、添加共享目录。见图 30-51，点击界面上方的“添加共享目录”按钮进入新建目录界面，填写共享目录信息，然后点击“创建”按钮。其中要共享的目录必须为当前系统上存在的文件夹；客户机范围的形式有四种类型，分别为：单个的 IP 地址、IP 网段、NIS 域名和所有地址；权限配置中的对待不可信用户/组，必须为当前系统上存在的用户/组 ID，否则验证不通过。

图 30-52 添加共享目录

3、删除共享目录。见图 30-51，点击列表中的目录，选中后点击上面的“删除”按钮。

30.5.6 驱动管理

本模块提供系统中驱动程序的信息查看，共分为六类：磁盘驱动器、网络适配器、PCI 设备驱动、PCI 信息、USB 信息和驱动模块。

1、磁盘驱动器。显示磁盘驱动程序的详细信息。

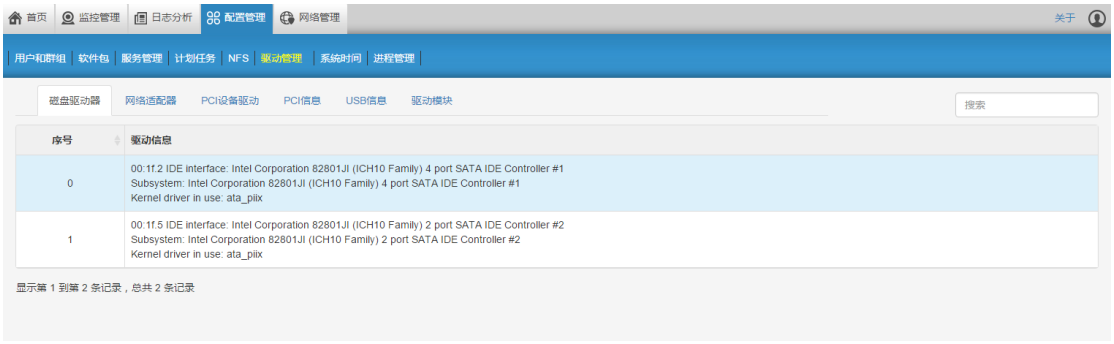


图 30-53 磁盘驱动

2、网络适配器。查看网卡驱动的详细信息，双击驱动信息可以查看完整内容。

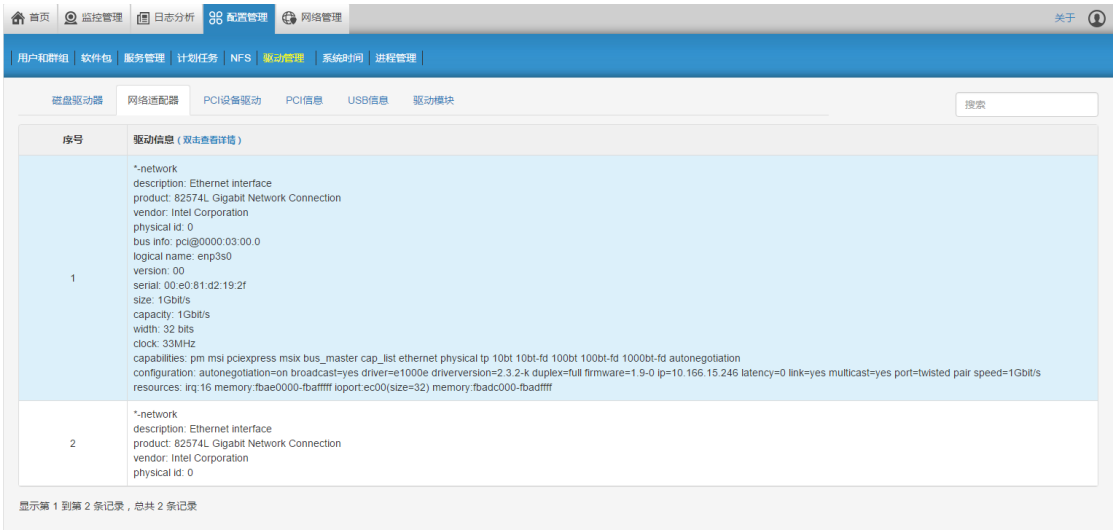


图 30-54 网络适配器

3、PCI 设备驱动。显示 PCI 设备的驱动程序信息。

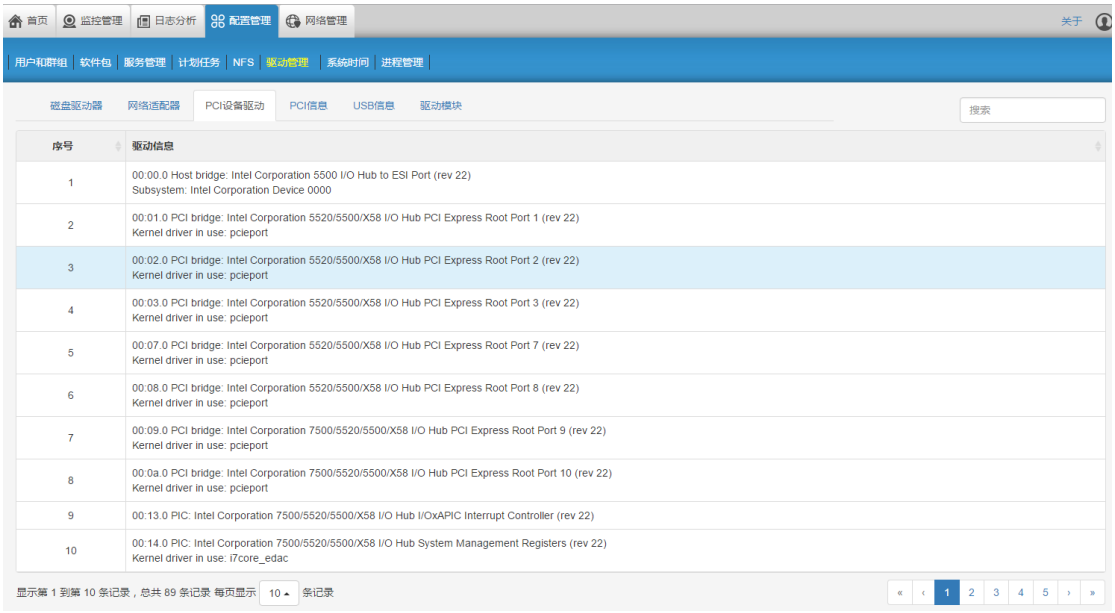


图 30-55 PCI 设备驱动

4、PCI 信息。显示系统中所有 PCI 总线的信息。

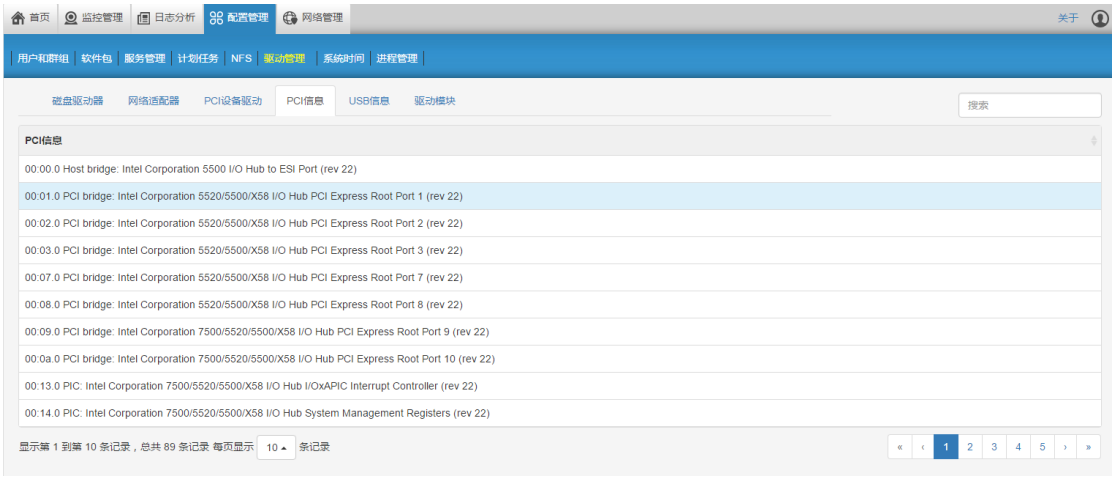


图 30-56 PCI 设备

5、USB 信息。显示系统中所有 USB 设备的信息。

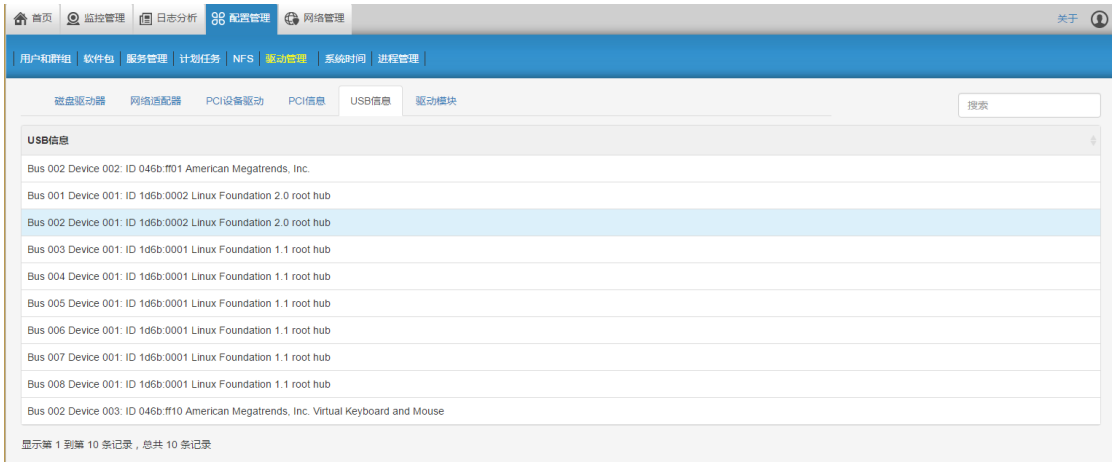


图 30-57 USB

6、驱动模块。显示系统中加载的所有设备及其驱动信息。

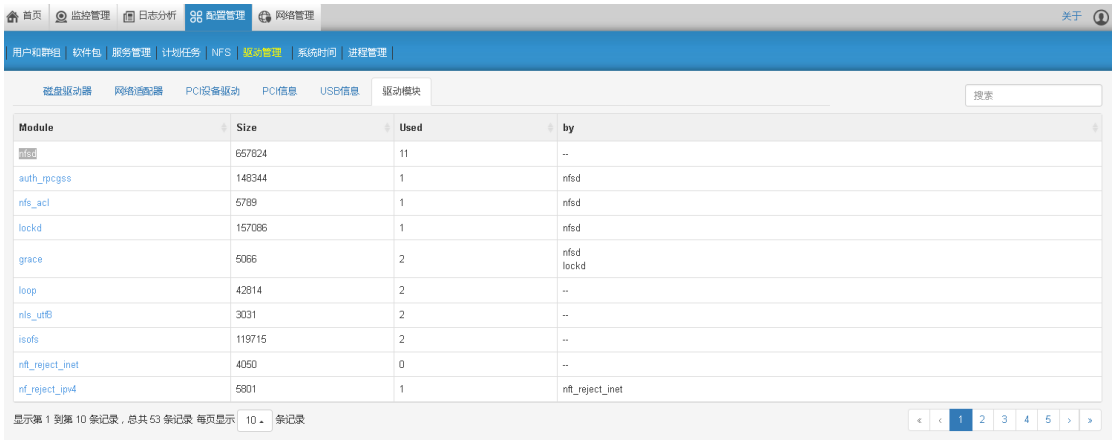


图 30-58 驱动模块

30.5.7 系统时间

本模块管理系统的时间和时区，提供硬件时间和系统时间的设置、时区的设置和与时间服务器同步的功能。

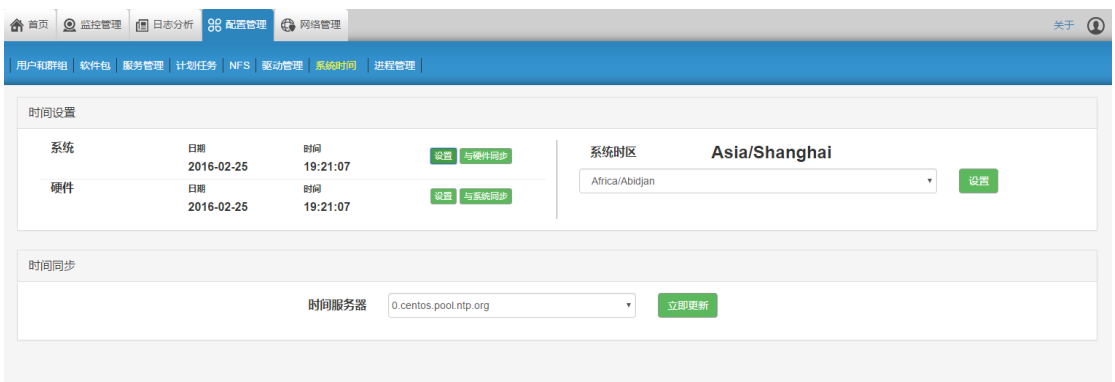


图 30-59 时间

1、时间设置。此功能分为系统时间设置和硬件时间设置。点击“设置”按钮即可弹出设置时间的模态框，如图 30-60，而后点击“新时间”输入框便可选择时间，选择完成后点击“确定”按钮便可设置成功。点击“与硬件同步”按钮可以把硬件时间同步到系统时间；点击“与系统同步”按钮可以把系统时间同步到硬件时间。

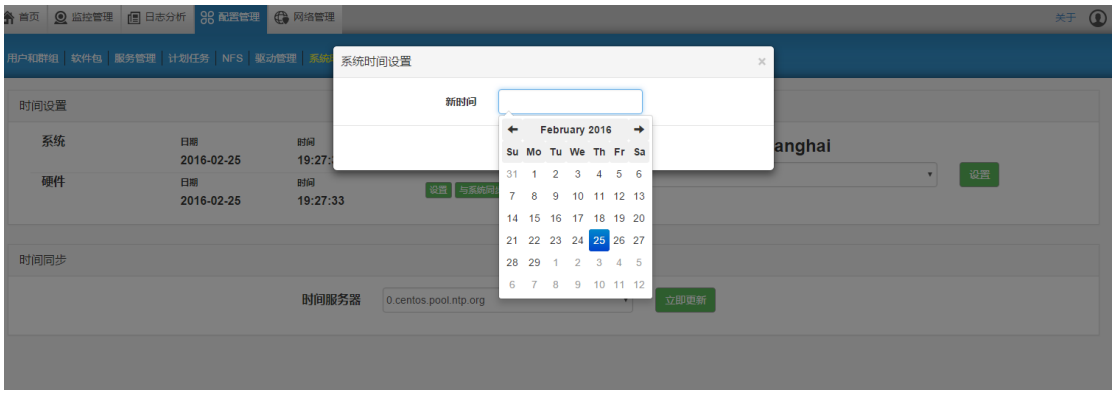


图 30-60 设置时间

2、时区设置。系统时区右边的内容是系统当前的时区，在下面下拉列表里选择时区并点击“设置”按钮便可设置其他时区。

3、时间同步。在时间服务器旁边的下拉列表里选择一个服务器，然后点击“立即更新”按钮，便可与服务器时间进行同步。

时间同步功能需要配置服务器后端，才能进行时间同步，配置方法如下：

a. 编辑/etc/chrony.conf 注释掉 pool 2.inspur.pool.ntp.org iburst

b. 在注释行下一行添加 server ntp-server-ip iburst

c. 重启服务：systemctl restart chronyd

30.5.8 进程管理

本模块提供系统进程的查看、关闭和搜索功能。

1、进程查看

点击“进程管理”按钮，默认便是进程查看界面。界面上以列表形式显示出系统中所有的进程信息。在列表上点击进程便可选中，然后点击上面的“删除”按钮

即可杀死进程。

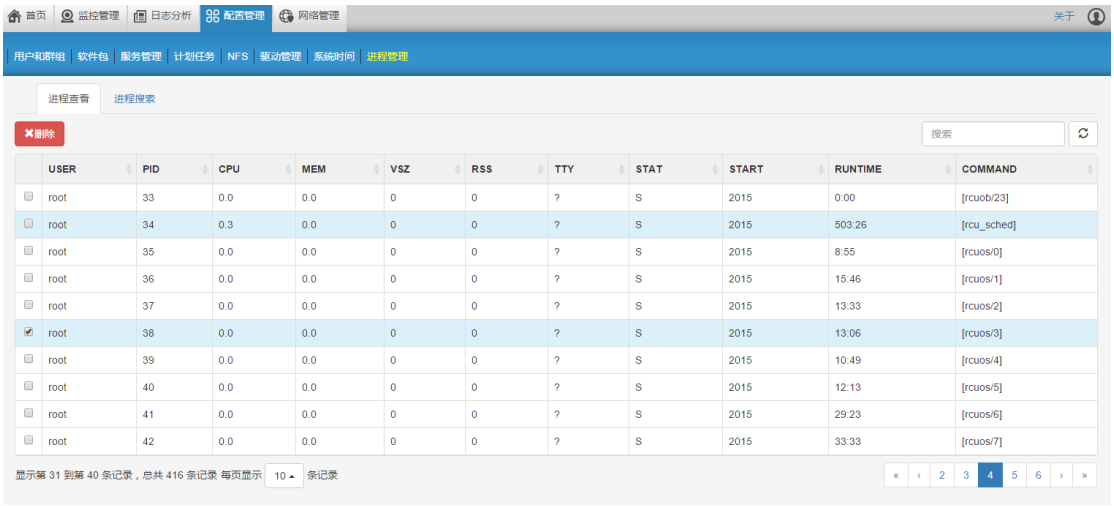


图 30-61 查看进程

2、进程搜索

本模块提供两种搜索形式：按进程所属用户（USER）和进程名搜索；按进程 ID 搜索。同时只能使用一种方法进行搜索。

（1）、按进程所属用户（USER）和进程名搜索。选择左边的复选框，填入用户名（USER）或进程名，或两项都填写，然后点击“搜索”按钮。

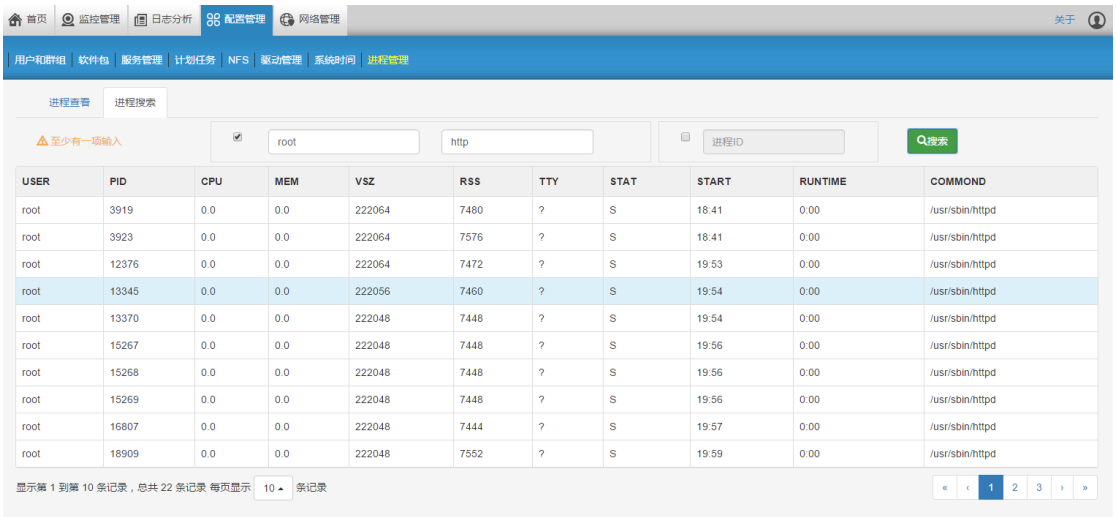


图 30-62 进程搜索

（2）、按进程 ID 搜索。选择右边的复选框，填入进程 ID，然后点击“搜索”按钮。

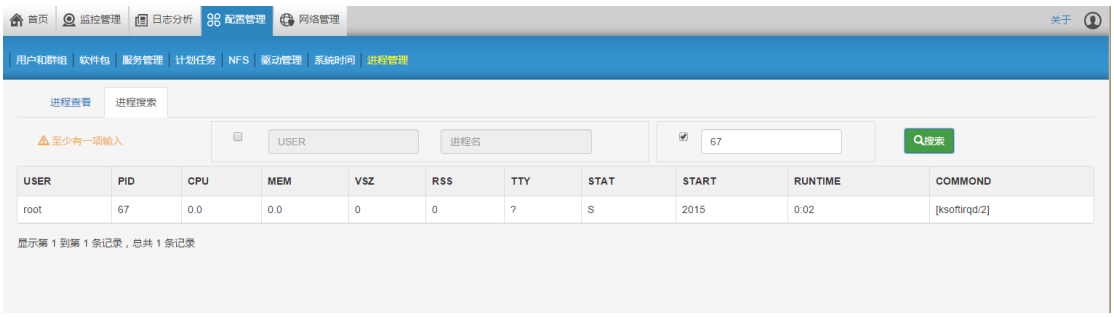


图 30-63 进程搜索

30.6 网络管理

30.6.1 网络接口

点击网络管理—网络接口，进入网络接口管理页面。

网络接口模块用于管理服务器的网络配置，包括网络设备和网络连接查询、单个有线连接配置信息管理、添加网络连接、设备状态和连接状态设置等。

1、网络设备列表

查询当前服务器上的网络设备基本信息，并列表显示。每个选项卡下面列出该网络设备的连接速度、网卡状态以及它的网络连接等信息，每个网络连接对应系统中的一个网络配置文件。

图 30-64 网络设备列表

2、网络设备状态设置

点击选项卡中网络设备名下方的“开启/关闭”按钮，可以开启或关闭某个网络设备。

3、网络连接状态设置

每个选项卡中列出了该设备可用的网络连接，点击某条连接中的“开启/关闭”按钮，可以开启或关闭该网络连接。

4、删除网络连接

每个选项卡中列出了该设备可用的网络连接，点击某条连接后面的“删除”按钮，可以删除该网络连接，并将对应的配置文件从系统中删除。

5、详细信息查询

查询某个网络设备活跃连接的详细信息，包括 IPV4 地址、IPV6 地址、硬件地址、默认路由、DNS 等信息。详细信息在每个选项卡中列出。

6、认证信息设置

点击网络设备选项卡中的网络连接名，进入认证设置。

认证设置用于修改某一网络连接的认证信息，包括连接名、MAC 地址、克隆 MAC 地址、最大传输单元 MTU、是否选自动连接、其他用户是否可用。MAC 地址从下拉菜单中选择当前系统中可用的网络设备的 MAC 地址。

填写完认证信息，点击“提交”按钮即可。

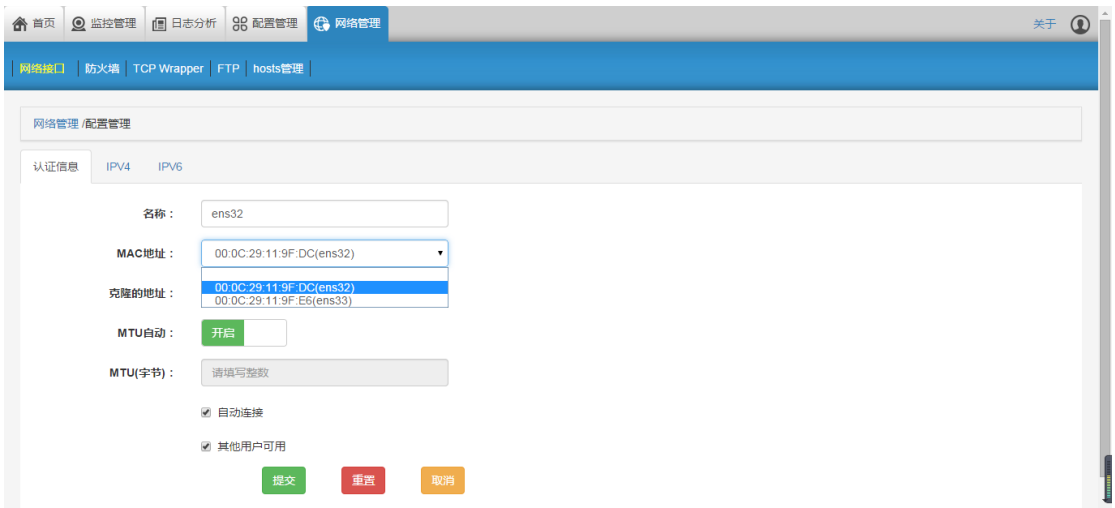


图 30-65 认证信息设置

7、IPV4 设置

点击网络设备选项卡中的网络连接名，进入 IPV4 设置。

IPV4 设置用来设置某一网络连接的 IPV4 信息，包括 IPV4 是否开启、地址类型、IPV4 地址信息、DNS 是否自动、DNS 地址、是否仅对此网络上的资源使用此链接。

地址类型在下拉菜单中选择。

填写完 IPV4 信息，点击“提交”按钮即可。

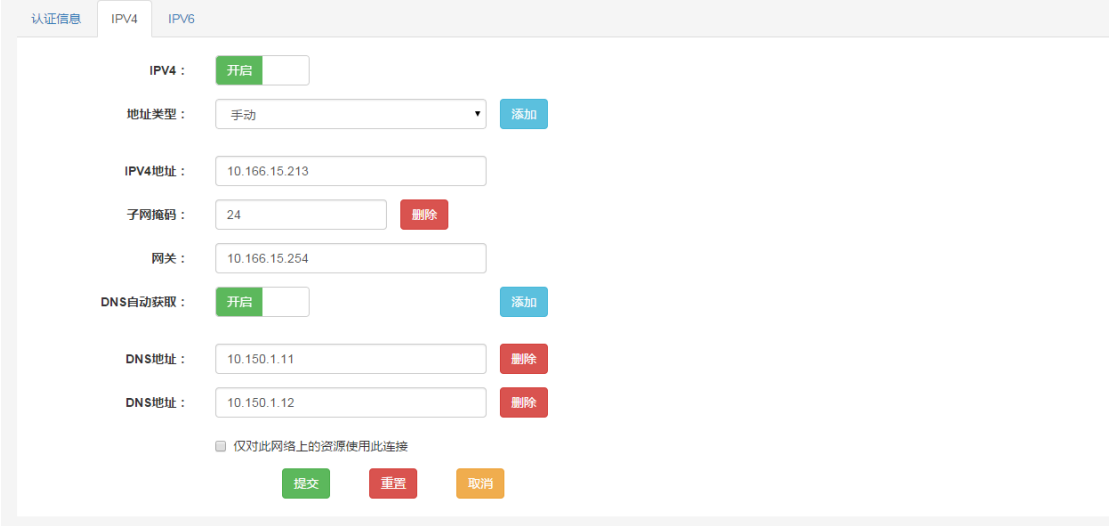


图 30-66 IPV4 信息设置

8、IPV6 设置

点击网络设备选项卡中的网络连接名，进入 IPV6 设置。同图 30-66。

IPV4 设置用来设置某一网络连接的 IPV6 信息，包括 IPV6 是否开启、地址类型、IPV6 地址信息、DNS 是否自动、DNS 地址、是否仅对此网络上的资源使用此链接。

地址类型在下拉菜单中选择。

填写完 IPV6 信息，点击“提交”按钮即可。

9、重置

点击网络设备选项卡中的网络连接名，进入连接信息配置页面，点击“重置”按钮，确定并提交，即可重置该网络连接的设置，恢复 IPv4、IPv6 自动获取 IP 地址，恢复 DNS 为自动。见图 30-66。

10、添加连接

在网络接口页面点击“添加”连接按钮，进入添加连接页面。

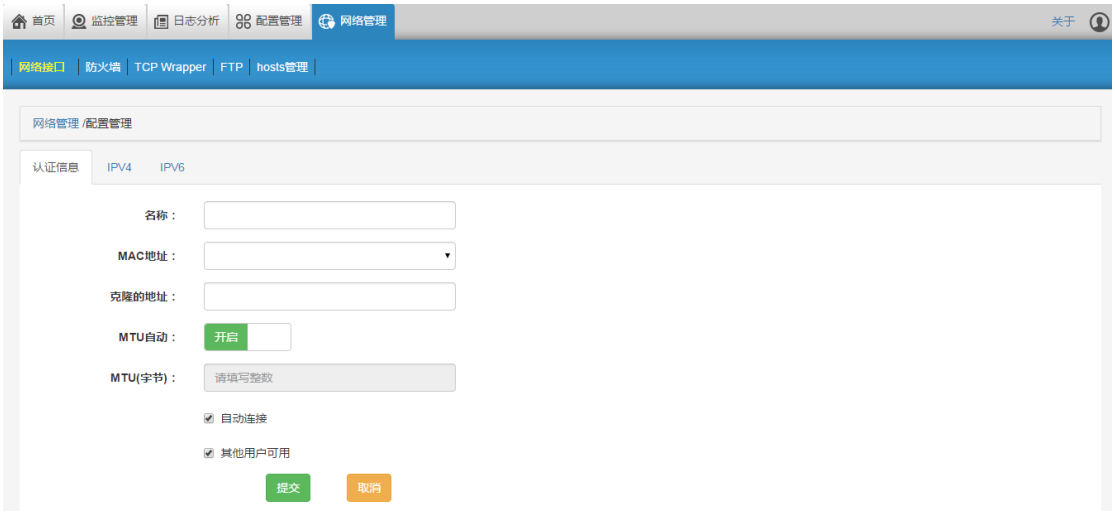


图 30-67 添加连接

填写完认证信息、IPV4 信息、IPV6 信息，点击“提交”按钮，即可添加一个新的连接。并在系统中创建对应的配置文件。IPV4、IPV6 默认地址类型为自动获取地址。

30.6.2 防火墙

防火墙模块同时提供对静态防火墙（即 iptables）规则的配置以及动态防火墙（即系统防火墙 firewalld）规则的配置。用户可以采用静态防火墙服务或者动态防火墙服务中的一种，二者不能同时使用（同时使用可能会造成系统崩溃）。静态防火墙可以显示防火墙规则、添加新规则、删除防火墙规则、应用防火墙规则。动态防火墙引入了“区域（zone）”的概念，可以进行区域管理、服务管理、端口、接口管理，并引入了应急模式。动态防火墙支持动态更新，不用重启服务。

1、iptables 管理

只有当前防火墙模式为静态防火墙时,才可以访问静态防火墙管理页面。点击网络管理—防火墙—iptables 进入 iptables 管理。

（1）显示防火墙规则

查看当前系统中的防火墙设置的基本信息，包括防火墙的类型,规则内容等信

息。可以根据表名和链名筛选防火墙规则。

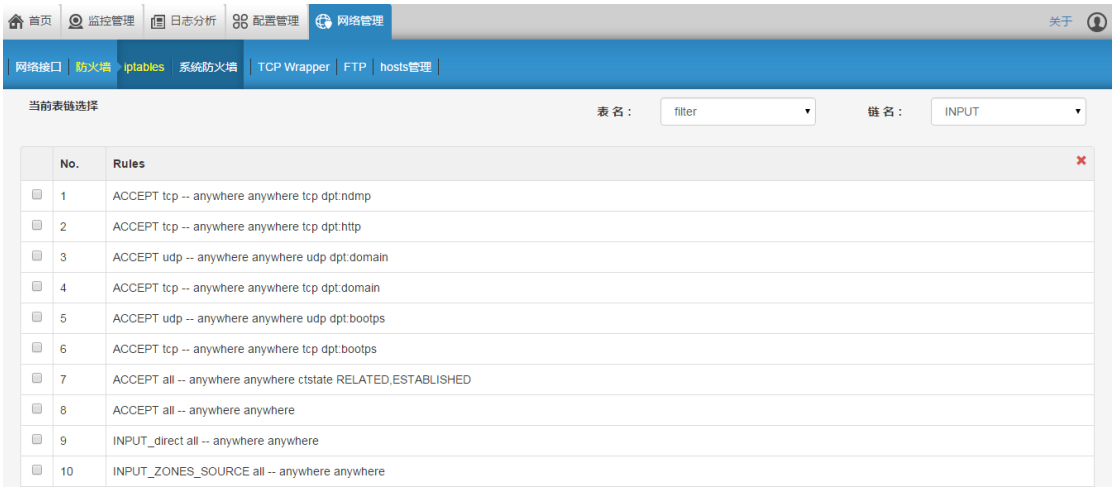


图 30-68 显示防火墙规则

（2）添加新规则

用户在规则添加文本框中输入 iptables 规则，点击“添加”按钮，即可完成新规则添加。

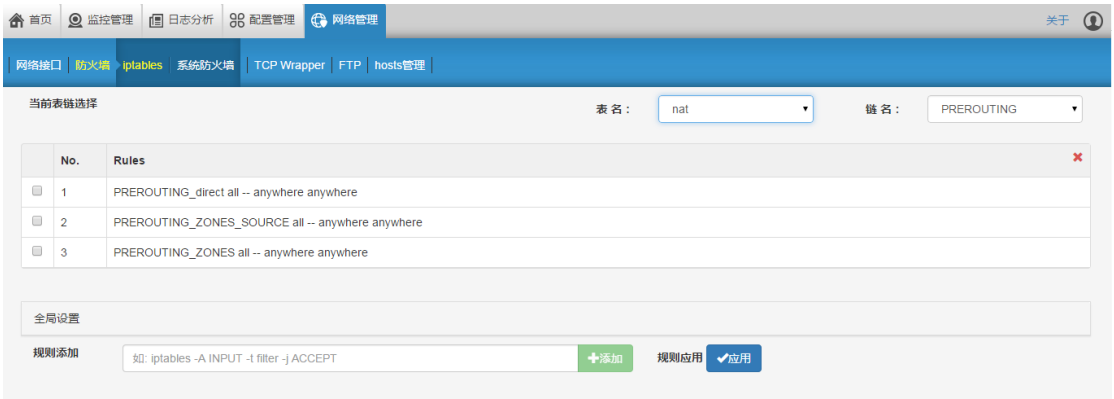


图 30-69 添加新规则

（3）删除防火墙规则

点击防火墙规则前面的复选框，选择要删除的规则，然后点击叉号，即可删除。

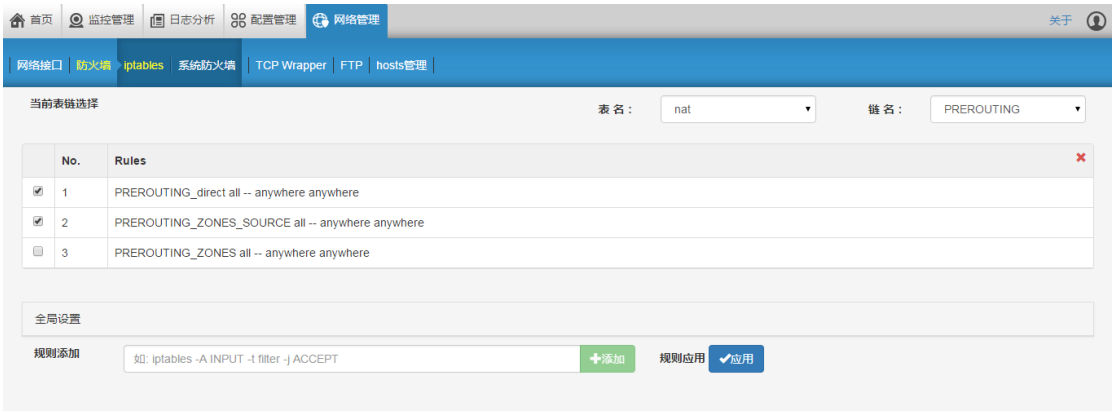


图 30-70 删除防火墙规则

（4）应用

点击页面下方的“应用”按钮，即可将防火墙规则应用到系统中。见图 30-70

2、系统防火墙管理

本模块用于查看当前系统的动态防火墙的状态、用户可以通过本模块完成防火墙的区域管理、服务管理、端口和协议管理、接口管理以及应急模式管理。

点击网络管理—防火墙—系统防火墙，进入系统防火墙管理页面。

（1）切换配置类型

用户在配置类型下拉菜单中可以完成配置类型切换：运行时或永久时。根据类型自动刷新防火墙配置。

图 30-71 切换配置类型

（2）重载防火墙

用户使用动态防火墙的重新加载功能，可以选择重新加载防火墙的方式：重

新加载和彻底加载，前者为保持防火墙状态的前提下，完成防火墙的重新装载；后者为丢弃当前的防火墙状态，完成防火墙的重新装载。

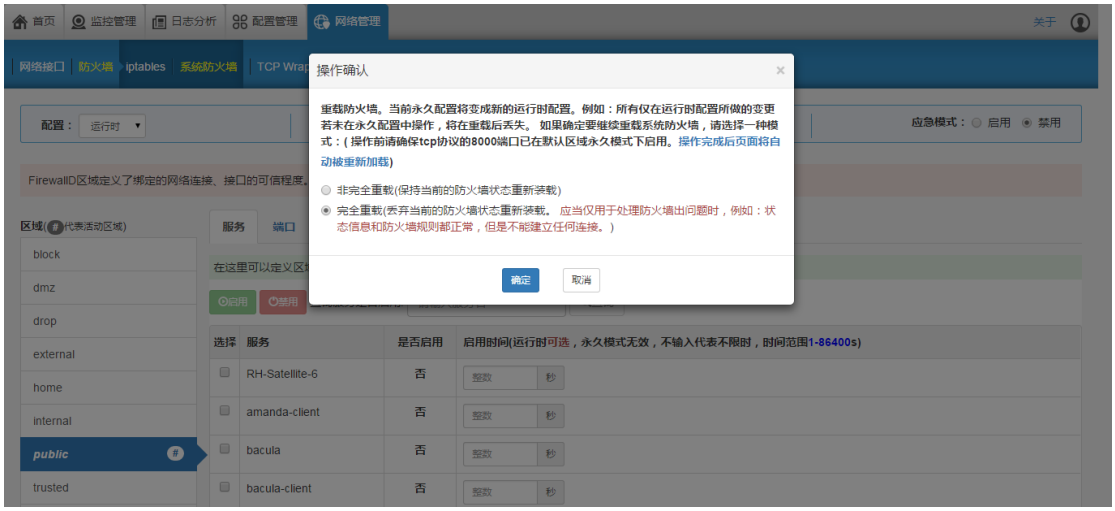


图 30-72 重载防火墙

点击“重载防火墙”按钮，可以选择重载防火墙的方式。

（3）区域列表

系统防火墙页面左侧显示当前系统防火墙支持的区域列表，活动区域用“#”标识，如下图。点击列表中的区域，自动刷新防火墙的配置。

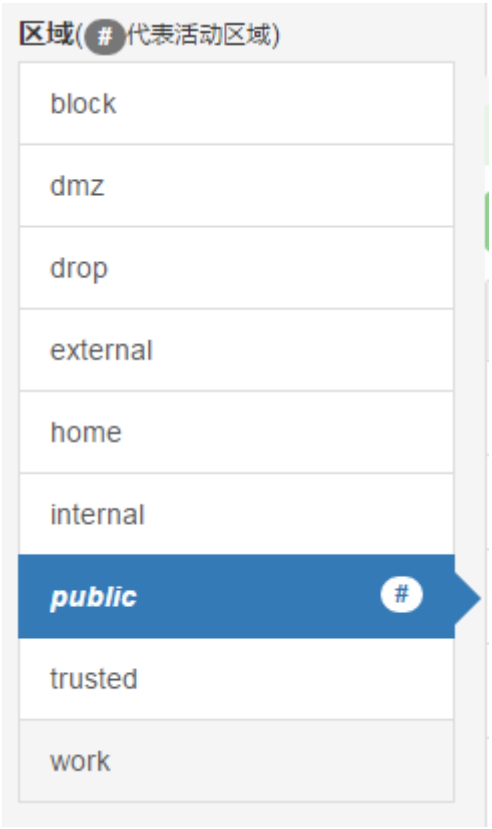


图 30-73 防火墙活动区域列表

（4）更改默认区域

防火墙页面上显示当前系统防火墙的默认区域。

图 30-74 防火墙默认区域

用户可以通过点击“更改默认区域”按钮，在下拉菜单中选择默认区域，更改系统防火墙默认区域。

图 30-75 更改默认区域

（5）区域服务管理

防火墙页面列出默认区域的服务基本信息，包括服务名称、是否启用、启用时间等。当切换区域时会自动刷新。

图 30-76 区域服务列表

点击服务名称前面的单选框，选中对应的服务，点击“启用”或“禁用”按钮，即可启用或者禁用选中的服务。

图 30-77 启用/禁用服务

（6）端口管理

点击系统防火墙页面端口选项卡，会列出某一区域的端口-协议列表。

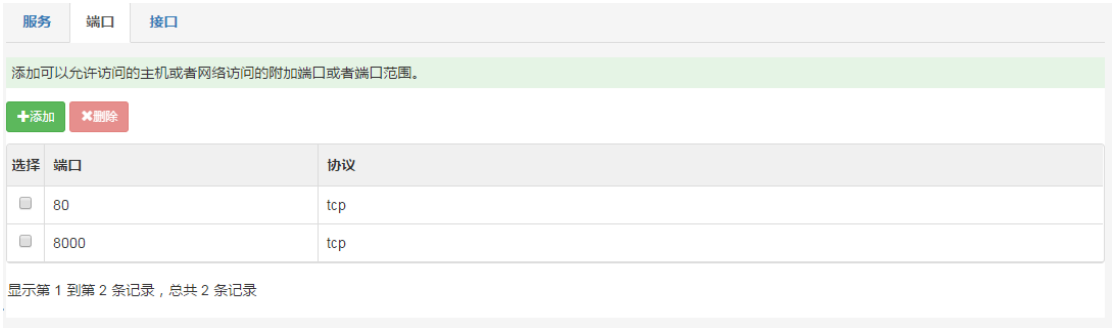


图 30-78 端口-协议列表

用户点击“添加”按钮，填写端口，选择端口协议，即可将端口添加到对应区域。需要注意的是，添加端口有两种配置类型：运行时和永久时，可以在配置类型下拉菜单中选择。

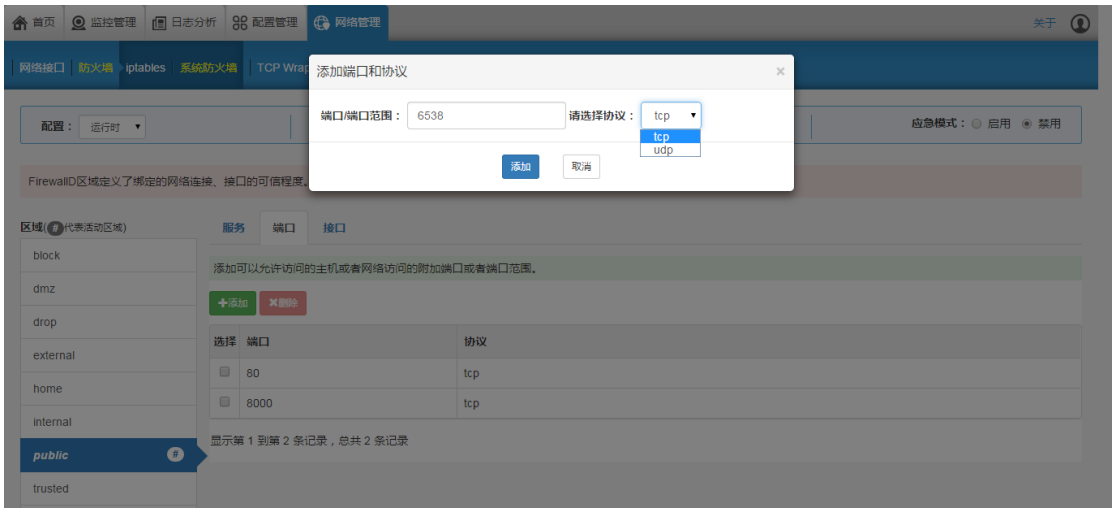


图 30-79 添加端口

用户点击端口前面的单选框，点击“删除”按钮，即可将选中的端口从系统中删除。需要注意的是，删除端口也对应两种配置类型：运行时和永久时。

（7）接口管理

用户可以通过接口管理查看当前系统中的接口，删除接口或者修改接口的所属区域，还可以添加一个接口到指定的区域。

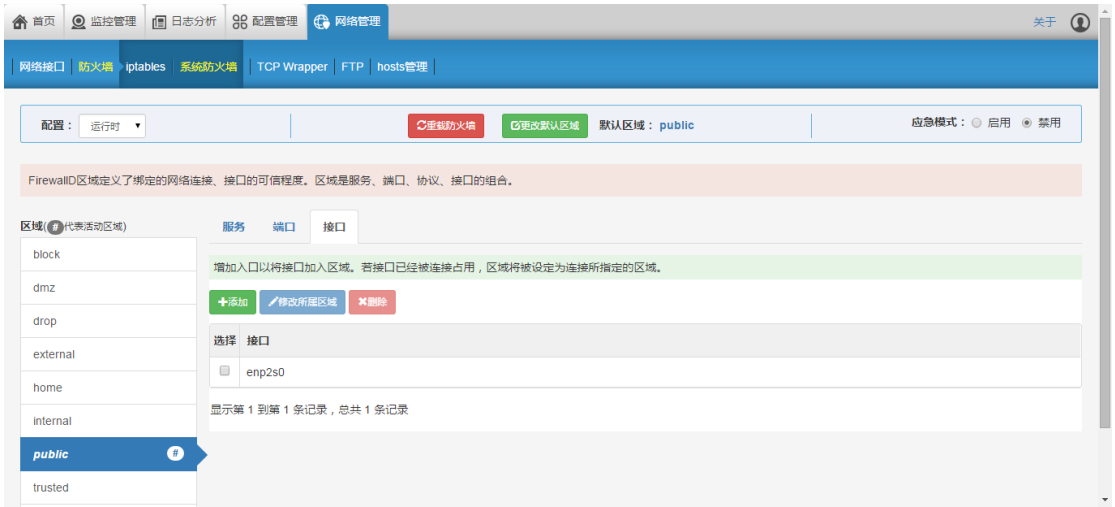


图 30-80 对应区域的接口

用户点击接口前面的单选框选中接口，可以执行两种操作：点击“删除”按钮，将接口从区域中删除；点击“修改所属区域”按钮，修改接口的所属区域。

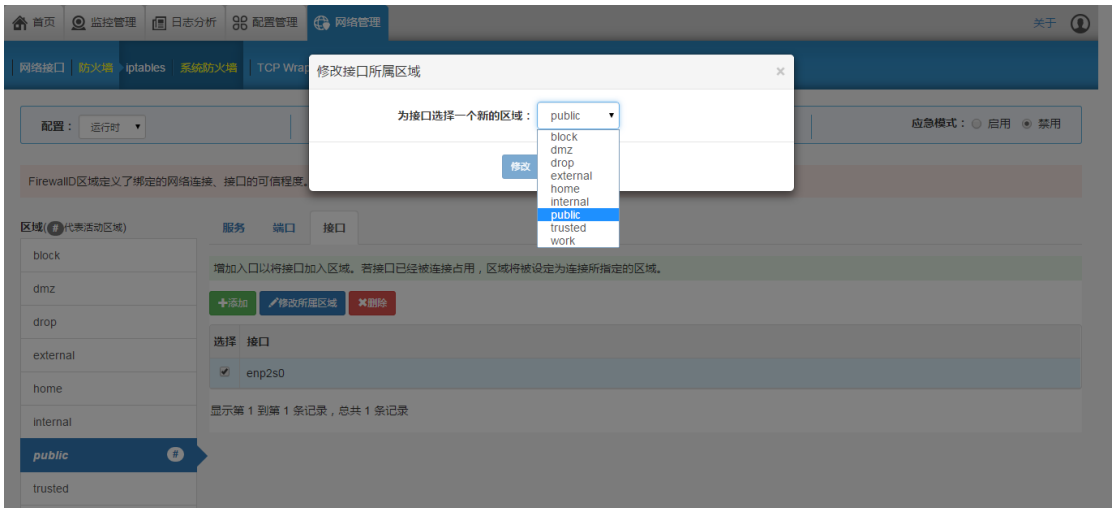


图 30-81 修改接口的所属区域

用户点击“添加接口”按钮，输入接口名称，点击添加即可将接口添加到系统中。

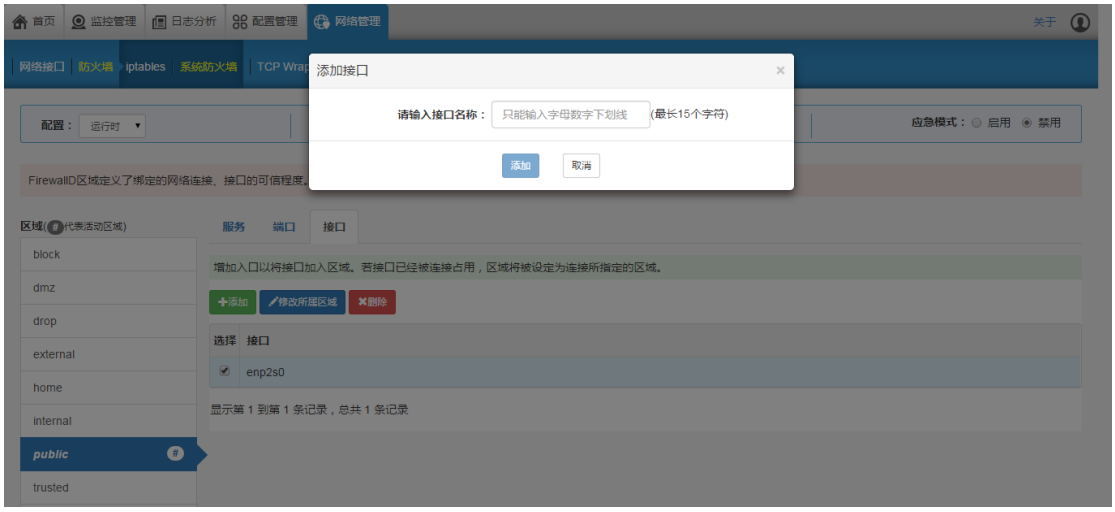


图 30-82 添加接口

（8）启用/禁用应急模式

应急模式是为了在特殊情况下对动态防火墙的一种管理方式，启用应急模式阻断所有网络连接，以防出现紧急状况。应急模式默认是禁用的，用户可以在紧急情况下启用。

图 30-83 启用/禁用应急模式

30.6.3 TCP Wrapper

TCP Wrappers 能够让系统管理员通过 inetd.conf 文件控制和记录对本地主机的基于 TCP 的连接。本模块提供对 TCP 访问规则的查看、添加和删除功能。

1、规则查看

点击网络和安全，选择 TCP Wrappers，进入列表显示界面，其中 Deny List 页面显示系统中所有拒绝连接的规则，Allow List 页面显示所有允许连接的规则。

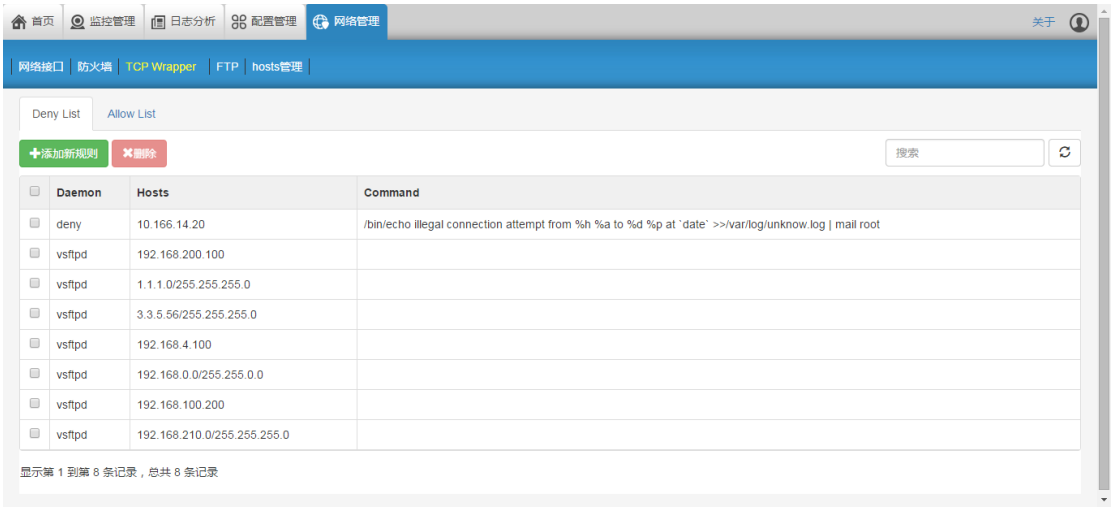


图 30-84 TCP Wrappers 页面

2、规则添加

在列表显示页面点击“添加新规则”按钮，进入添加规则界面。Service 名称只能输入数字、字母和下划线，最长 32 位。主机地址中选项的含义：

ALL 代表所有主机,或者所有服务；

LOCAL 表示本地主机，非 FQDN 主机；

KNOWN 表示可以被解析的主机；

UNKNOWN 表示反向可以被解析的主机；

PARANOID 表示正反向解析不匹配的主机

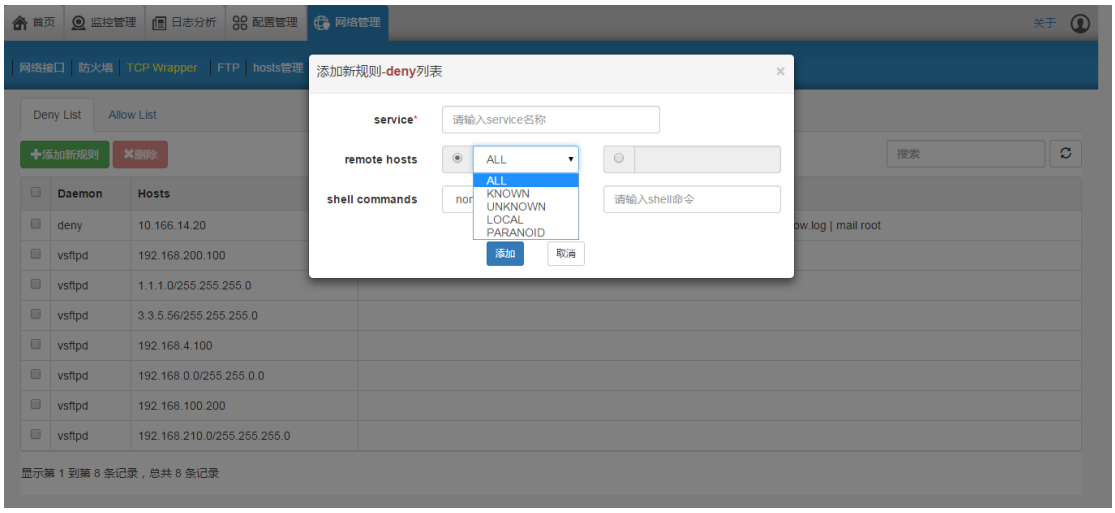


图 30-85 TCP Wrappers 添加规则页面

在添加规则时，shell commands 选项可以选择 none、spwan 或者 twist。

spawn:当访问被拒绝时不会向被拒绝的主机返回信息

twist:当访问被拒绝并需要向被拒绝的主机返回信息时使用 twist

3、规则删除

选择图 30-85 预览页面列表中的复选框，点击“删除”按钮，将删除选中的规则，并刷新预览页面。若不选择任何复选框，点击“删除”按钮，将出现警告信息。

30.6.4 FTP

本模块提供对系统 FTP 服务的管理功能。包括 FTP 虚拟用户管理、FTP 活跃连接监控、FTP 系统配置。

1、FTP 活跃连接

点击菜单栏“FTP”链接进入 FTP 主界面，如图 30-86，界面上以列表形式显示出系统中所有活跃的 FTP 连接，包括本地地址、远端地址、状态和 PID/程序名。界面左上角是 FTP 服务的开关，显示 FTP 服务当前状态，点击即可打开/关闭 FTP 服务。

图 30-86 FTP

2、FTP 用户管理

见图 30-86，点击图中“用户管理”按钮，进入用户管理界面。用户可以新建、删除、配置 FTP 用户。

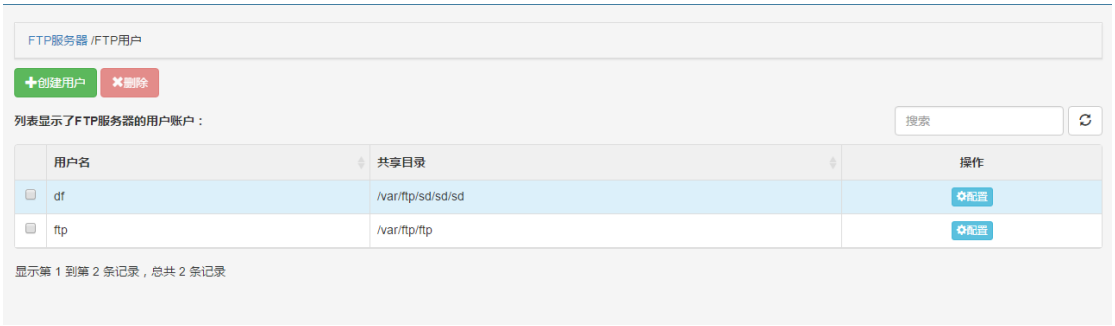


图 30-87 用户管理

（1）、新建用户。见图 30-87，点击“创建用户”按钮，进入创建用户界面。填入用户信息，点击“创建”按钮。其中用户名只能是字母、数字、下划线和横线的组合，且不能以数字开头；共享目录只能以斜杠+字母/数字/下划线开头。

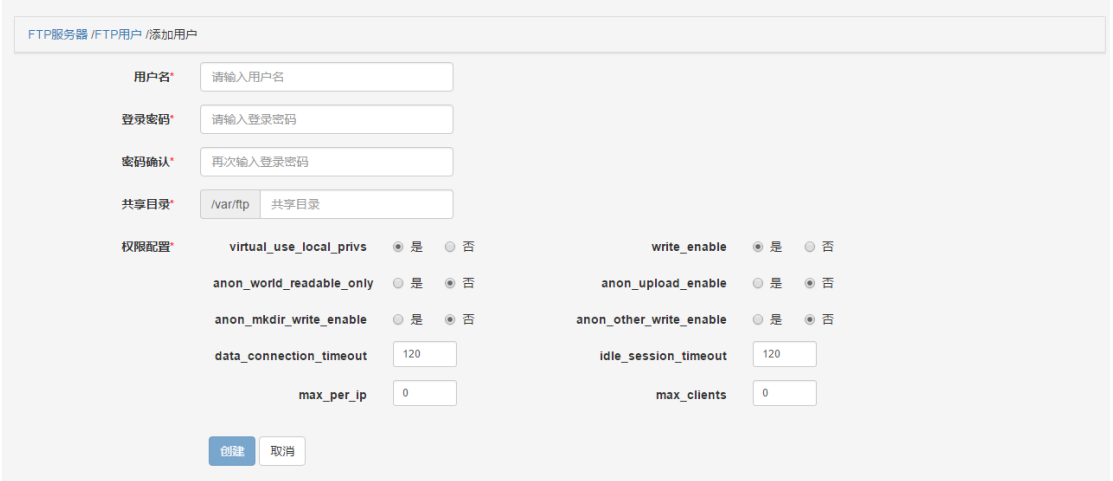


图 30-88 添加用户

（2）、配置用户。见图 30-87，点击用户列表中用户名后面的“配置”按钮，进入配置用户界面。用户可以修改密码和权限。

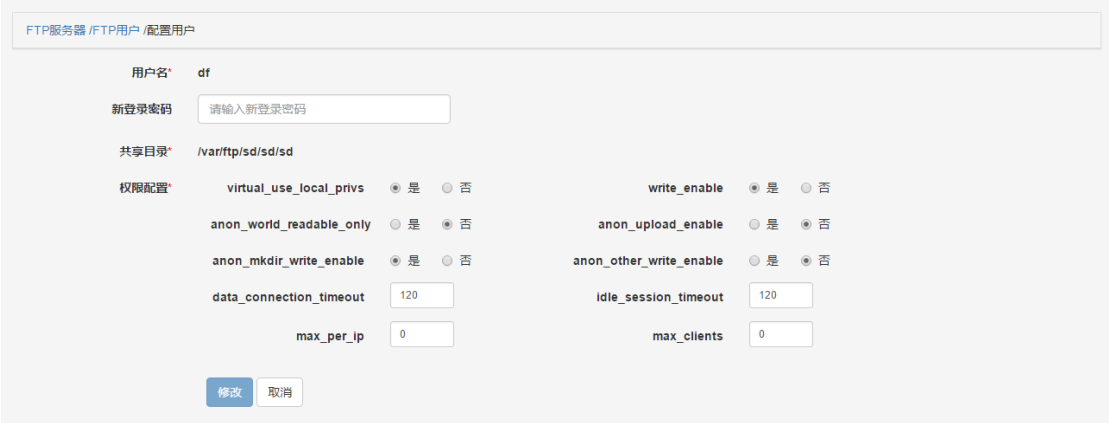


图 30-89 配置用户

（3）、删除用户。见图 30-87，在用户列表中单机用户以选中，然后点击上面的“删除”按钮。

3、FTP 配置

点击“FTP 配置”按钮进入 FTP 服务配置界面。用户可以配置 FTP 服务，并且可以配置禁止访问 FTP 服务器的 IP 地址。

（1）、服务配置。进入 FTP 配置界面后，默认是服务配置功能，如图 30-90。界面上前五行配置是只读的，不允许修改。用户配置完成后点击“修改”按钮即可完成配置，也可以点击“恢复默认设置”按钮回复 FTP 服务的默认设置。

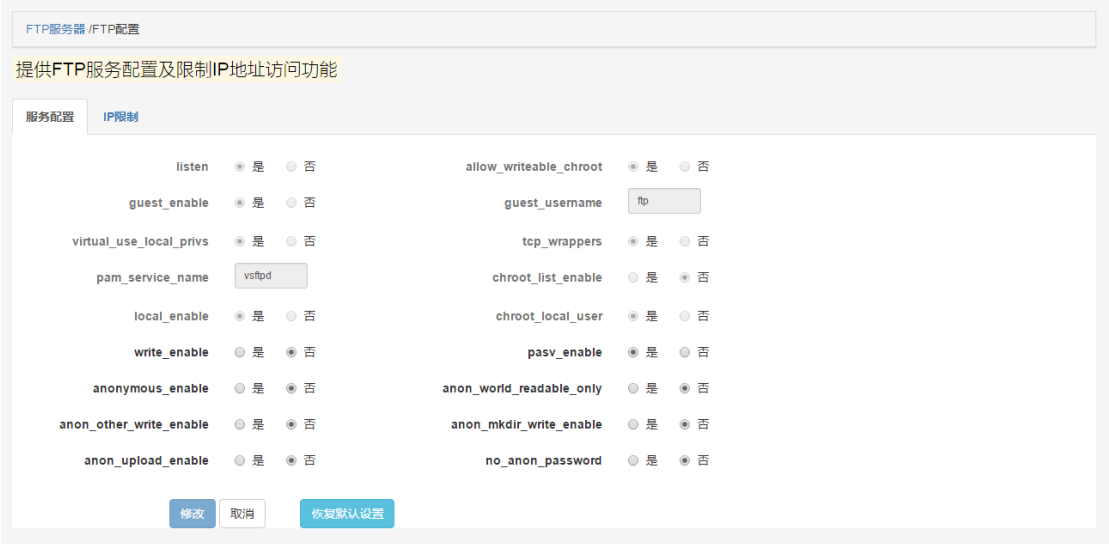


图 30-90 服务配置

（2）、IP 限制。点击“IP 限制”切换到 IP 地址界面。界面上以列表形式显示所有禁止访问 FTP 服务器的 IP 地址。用户可以添加、修改、删除 IP 地址。

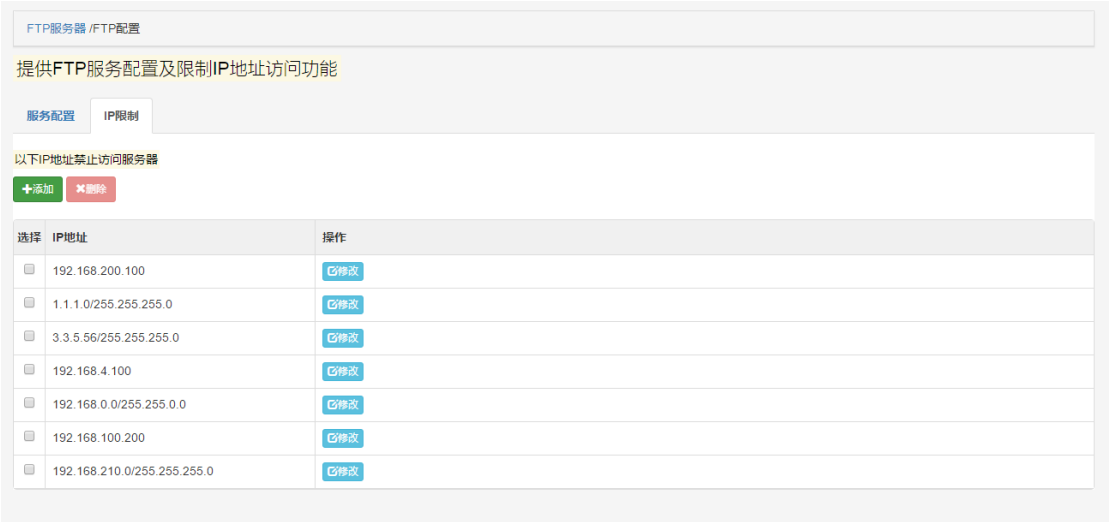


图 30-91 IP 限制

添加 IP 地址提供两种形式的 IP 地址：单个 IP 地址、IP 网段。

图 30-92 添加 IP

单击列表中的 IP 地址以选中，然后点击“删除”按钮即可删除。

30.6.5 hosts 管理

此模块用于管理系统中的主机地址设置，可实现查看、添加、删除三个功能。

1、查看主机地址

点击网络管理-hosts 管理，可查看当前系统中的主机地址设置，并以列表的形式显示（图 30-93）。

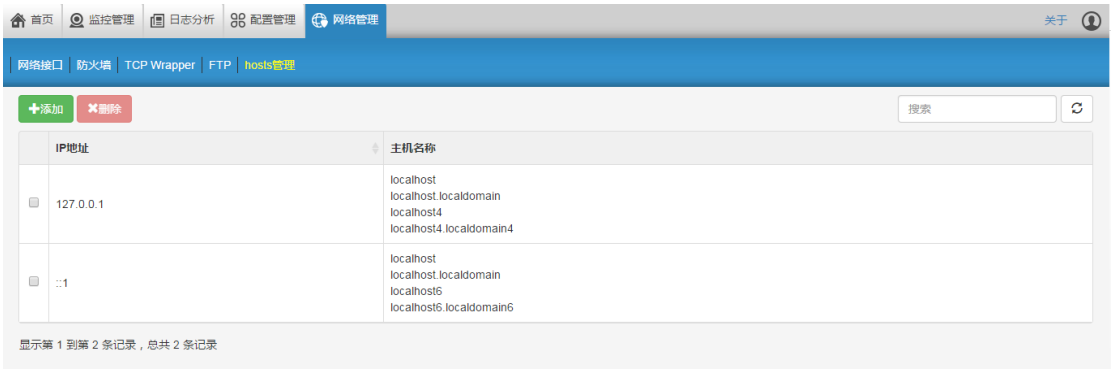


图 30-93 主机地址预览

2、添加主机地址

点击图 30-93 中的“添加”按钮，弹出添加主机地址对话框（图 30-94），可输入 IP 地址和主机名称信息，点击“添加”按钮，验证信息正确性，验证次序为 IP 地址、主机名称，其中 IP 地址符合 IP 地址的规则，主机名称为以空格分开的主机名，主机名不能包含汉字。本管理系统暂时不支持 Ipv6，所以，对于：：1 类型的 IP 地址不支持。若信息验证无误，则弹出创建成功或失败的结果，否则提示信息错误及原因。

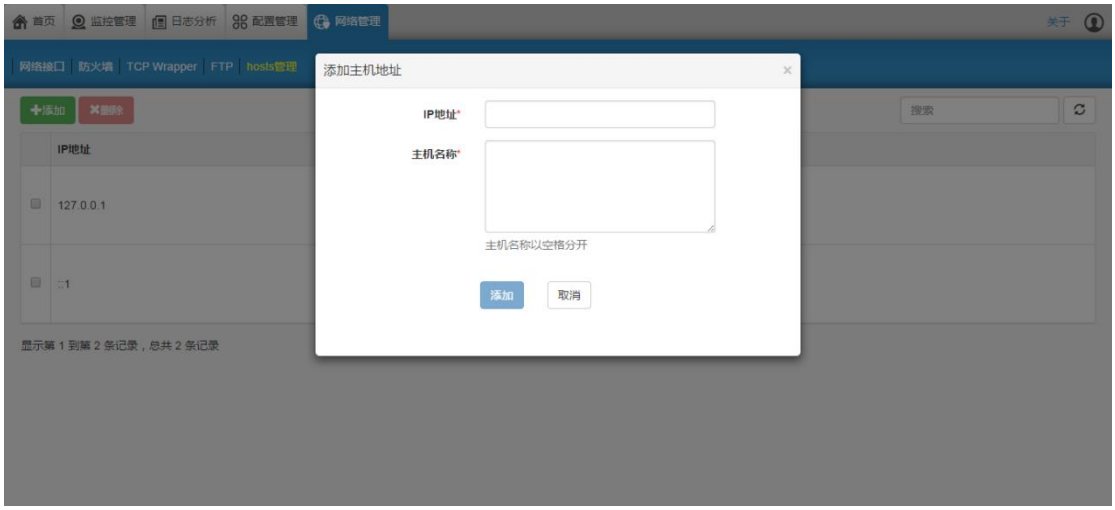


图 30-94 创建主机地址

3、删除主机地址

选择图 30-93 预览页面列表中的复选框，点击“删除”按钮，将删除选中的主机地址，并刷新预览页面。若不选择任何复选框，点击删除按钮，将出现警告信息。

30.7 关于

用户在登录状态下，点击本系统右上角的关于按钮，查看本系统关于信息。如图 30-95。

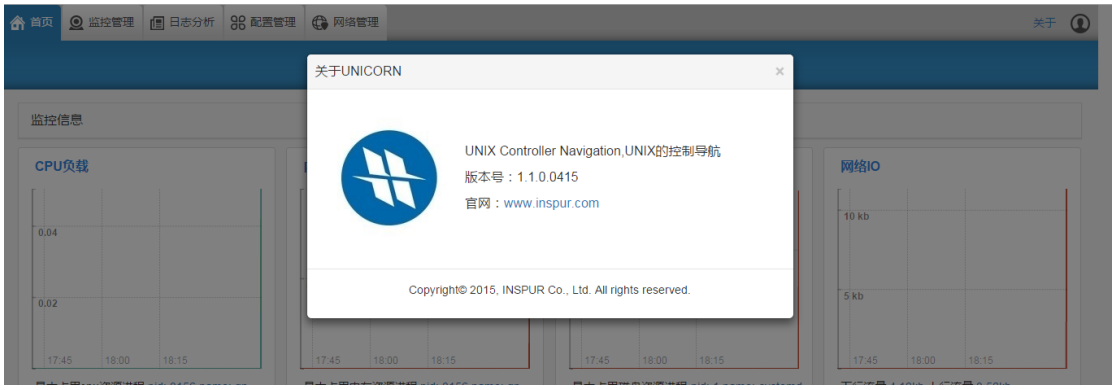


图 30-95 关于