第十五章：系统管理

尚硅谷云计算 Linux 课程

版本：V1.0

讲师：沈超

一、进程管理

1　进程简介

1）、什么是进程管理？

那么到底什么是进程呢？进程是正在执行的一个程序或命令，每一个进程都是一个运行的实体，都有自己的地址空间，并占用一定的系统资源。那么什么是程序呢？程序是人使用计算机语言编写的可以实现特定目标或解决特定问题的代码集合。这么讲很难理解，那我们换一种说法：

- ✧ 程序是人使用计算机语言编写的，可以实现一定功能，并且可以执行的代码集合。
- ✧ 进程是正在执行当中的程序。程序被执行时，执行人的权限和属性、以及程序的代码都会被加载如内存，操作系统给这个进程分配一个 ID 号，我们成为 PID（进程 ID）。

2）、进程管理的作用

- ✧ 判断服务器健康状态：运维工程师最主要的工作就是保证服务器安全稳定的运行。理想的状态是，在服务器出现问题，但是还没有造成服务器宕机或停止服务时，就人为干预解决了问题。进程管理最主要的工作就是判断服务器当前运行是否健康，是否需要人为干预。如果服务器的 CPU 占用率、内存占用率过高，就需要人为介入解决问题了。
- ✧ 查看系统中所有的进程：我们需要查看系统中所有正在运行的进程，通过这些进程可以判断系统中运行了哪些服务，是否有非法服务运行。
- ✧ 杀死进程：这是进程管理中最不常用的手段，当我需要停止服务时，会通过正确关闭命令来停止服务（如 apache 服务可以通过 service httpd stop 来关闭）。只有当正确终止进程的手段失效的情况下，才会考虑使用 kill 命令杀死进程（你不是杀手，不要什么进程都用 kill 来终止，否则非常容易导致服务器崩溃）。

2　进程的查看

1）、　ps 命令

ps 命令是用来静态显示系统中进程的命令。不过这个命令有些特殊，它的部分命令的选项不能加入“-”，比如命令“ps　aux”，其中“aux”是选项，但是这个选项不能加入“-”。这是因为 ps 命令的部分选项需要遵守 BSD 操作系统的格式。所以 ps 命令的常用选项的组合是固定的。命令如下：

```
[root@localhost ~]# ps aux
#查看系统中所有进程，使用 BSD 操作系统格式
[root@localhost ~]# ps -le
#查看系统中所有进程，使用 Linux 标准命令格式。
```

选项：

a：	显示一个终端的所有进程，除了会话引线
u：	显示进程的归属用户及内存的使用情况
x：	显示没有控制终端的进程

```
    -l:          长格式显示。显示更加详细的信息
    -e:          显示所有进程，和-A 作用一致
```

```
[root@localhost ~]# ps aux
#查看系统中所有的进程
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2   2872  1416 ?        Ss   Jun04   0:02 /sbin/init
root         2  0.0  0.0      0     0 ?        S    Jun04   0:00 [kthreadd]
```

解释下这个命令的输出：

- ✧ USER：该进程是由哪个用户产生的；
- ✧ PID：进程的 ID 号；
- ✧ %CPU：该进程占用 CPU 资源的百分比，占用越高，进程越耗费资源；
- ✧ %MEM：该进程占用物理内存的百分比，占用越高，进程越耗费资源；
- ✧ VSZ：该进程占用虚拟内存的大小，单位 KB；
- ✧ RSS：该进程占用实际物理内存的大小，单位 KB；
- ✧ TTY：该进程是在哪个终端中运行的。其中 tty1-tty7 代表本地控制台终端（可以通过 alt+F1-F7 键切换不同的终端），tty1-tty6 是本地的字符界面终端，tty7 是图形终端。pts/0-255 代表虚拟终端，一般是远程连接的终端，第一个远程连接占用的是 pts/0 终端，第二个远程连接占用 pts/1，依次增长。
- ✧ STAT：进程状态。常见的状态有：
  - ➢ D：不可被唤醒的睡眠状态，通常用于 I/O 情况
  - ➢ R：该进程正在运行
  - ➢ S：该进程在睡眠状态，可被唤醒
  - ➢ T：停止状态，可能是在后台暂停或进程在除错状态
  - ➢ W：内存交互状态（从 2.6 内核开始无效）
  - ➢ X：死掉的进程（应该不会出现）
  - ➢ Z：僵尸进程。进程已经终止，但是部分程序还在内存当中。
  - ➢ <：高优先级（以下状态在 BSD 格式当中出现）
  - ➢ N：低优先级
  - ➢ L：被锁入内存
  - ➢ s：包含子进程
  - ➢ l：多线程（小写 L）
  - ➢ +：位于后台
- ✧ START：该进程的启动时间
- ✧ TIME：该进程占用 CPU 的运算时间，注意不是系统时间
- ✧ COMMAND：产生此进程的命令名

```
[root@localhost ~]# ps -le
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   718 -      ?        00:00:02 init
1 S     0     2     0  0  80   0 -     0 -      ?        00:00:00 kthreadd
```

也来解释下这个命令的输出：

- ✧ F：进程标志，说明进程的权限，常见的标志有：

- 1：进程可以复制，但是不能执行
- 4：进程使用超级用户权限

✧ S：进程状态。具体的状态和“ps aux”命令中 STAT 状态一致
✧ UID：进程是哪个 UID 用户调用运行的
✧ PID：进程的 ID 号
✧ PPID：父进程的 ID 号
✧ C：该进程的 CPU 使用率，单位是百分比
✧ PRI：进程的优先级，数值越小该进程优先级越高，越快被 CPU 执行
✧ NI：进程的优先级，也是数值越小越早被执行
✧ ADDR：该进程在内存的哪个位置
✧ SZ：该进程占用多大内存
✧ WCHAN：该进程是否运行。“-”代表正在运行
✧ TTY：该进程由哪个终端产生
✧ TIME：该进程占用 CPU 的运算时间，注意不是系统时间
✧ CMD：产生此进程的命令名

2)、 top 命令

```
[root@localhost ~]# top [选项]
选项：
    -d 秒数：    指定 top 命令每隔几秒更新。默认是 3 秒
    -b：         使用批处理模式输出。一般和“-n”选项合用，用于把 top 命令重定向到
                 文件中
    -n 次数：    指定 top 命令执行的次数。一般和“-b”选项合用
    -p：         指定 PID。只查看某个 PID 的进程
    -s：         使 top 在安全模式运行，避免在交互模式中出现错误
    -u 用户名：  只监听某个用户的进程
在 top 命令的交互模式当中可以执行的命令：
    ？或 h：     显示交互模式的帮助
    P：          以 CPU 使用率排序，默认就是此项
    M：          以内存的使用率排序
    N：          以 PID 排序
    T：          按照 CPU 的累积运算时间排序，也就是用 TIME+项排序
    k：          按照 PID 号，给予某个进程一个信号。一般用于终止某个进程，信号 9
                 是强制终止的信号
    r：          按照 PID 号，给某个进程重设优先级（Nice）值
    q：          退出 top
```

我们看看 top 命令的执行结果：

```
[root@localhost ~]# top
top - 12:26:46 up 1 day, 13:32,  2 users,  load average: 0.00, 0.00, 0.00
Tasks:  95 total,   1 running,  94 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.1%us,  0.1%sy,  0.0%ni, 99.7%id,  0.1%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:    625344k total,   571504k used,    53840k free,    65800k buffers
Swap:   524280k total,        0k used,   524280k free,   409280k cached
```

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
19002 root      20   0  2656 1068  856 R  0.3  0.2   0:01.87 top
    1 root      20   0  2872 1416 1200 S  0.0  0.2   0:02.55 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.03 kthreadd
```

我们解释下命令的输出，top 命令的输出内容是动态的，默认每 3 秒刷新一次。命令的输出主要分为两大部分：第一部分是前五行，显示的是整个系统的资源使用状况，我们就是通过这些输出判断服务器的健康状态；第二部分从第六行开始，显示的是系统中进程的信息。

- 我们先来说明第一部分的作用：
  - 第一行信息为任务队列信息，具体内容如表 15-1 所示：

内容	说明
12:26:46	系统当前时间
up 1 day, 13:32	系统的运行时间，本机已经运行 1 天 13 小时 32 分钟
2 users	当前登录了两个用户
load average: 0.00, 0.00, 0.00	系统在之前 1 分钟，5 分钟，15 分钟的平均负载。如果 CPU 是单核，则这个数超过 1，就是高负载。如果 CPU 是四核，则这个数超过 4，就是高负载。（这个平均负载完全是个人经验来进行判断的，一般认为不应该超过服务器 CPU 的核数。）

表 15-1 任务队列信息

  - 第二行为进程信息，具体内容如表 15-2 所示：

内容	说明
Tasks: 95 total	系统中的进程总数
1 running	正在运行的进程数
94 sleeping	睡眠的进程
0 stopped	正在停止的进程
0 zombie	僵尸进程。如果不是 0，需要手工检查僵尸进程

表 15-2 进程信息

  - 第三行为 CPU 信息，具体内容如表 15-3 所示：

内容	说明
Cpu(s): 0.1%us	用户模式占用的 CPU 百分比
0.1%sy	系统模式占用的 CPU 百分比
0.0%ni	改变过优先级的用户进程占用的 CPU 百分比
99.7%id	空闲 CPU 的 CPU 百分比
0.1%wa	等待输入/输出的进程的占用 CPU 百分比
0.0%hi	硬中断请求服务占用的 CPU 百分比
0.1%si	软中断请求服务占用的 CPU 百分比
0.0%st	st（Steal time）虚拟时间百分比。就是当有虚拟机时，虚拟 CPU 等待实际 CPU 的时间百分

	比。

表 15-3 CPU 信息

- 第四行为物理内存信息，具体内容如表 15-4 所示：

内容	说明
Mem: 625344k total	物理内存的总量，单位 KB
571504k used	已经使用的物理内存数量
53840k free	空闲的物理内存数量，我们使用的是虚拟机，总共只分配了 628MB 内存，所以只有 53MB 的空闲内存了
65800k buffers	作为缓冲的内存数量

表 15-4 物理内存信息

- 第五行为交换分区（swap）信息

内容	说明
Swap: 524280k total	交换分区（虚拟内存）的总大小
0k used	已经使用的交互分区的大小
524280k free	空闲交换分区的大小
409280k cached	作为缓存的交互分区的大小

表 15-5 交互分区信息

- 再来看 top 命令的第二部分输出，主要是系统进程信息。这部分和 ps 命令的输出比较类似，只是如果在终端中执行 top 命令不能看到所有的进程，而只能看到占比靠前的进程。
  - PID：进程 ID
  - USER：该进程所属的用户
  - PR：优先级，数值越小优先级越高
  - NI：优先级，数值越小优先级越高
  - VIRT：该进程使用的虚拟内存的大小，单位 KB
  - RES：该进程使用的物理内存的大小，单位 KB
  - SHR：共享内存大小，单位 KB
  - S：进程状态
  - %CPU：该进程占用 CPU 的百分比
  - %MEM：该进程占用内存的百分比
  - TIME+：该进程总共占用的 CPU 时间
  - COMMAND：进程的命令名

接下来我们举几个 top 命令常用的实例，来继续说明下 top 命令。比如我如果只想让 top 命令查看某一个进程，这时就可以使用“-p 选项”：

```
[root@localhost ~]# top -p 15273
#只查看 PID 为 15273 的 apache 进程
```

在 top 命令的交互界面中按“q”键会退出 top 命令。也可以按“？”或“h”得到 top 命令交互界面的帮助信息。也可以按“k”键终止某个进程，命令如下：

```
[root@localhost ~]# top
top - 14:10:15 up 1 day, 15:15,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  97 total,   1 running,  96 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:    625344k total,   574248k used,    51096k free,    66840k buffers
Swap:   524280k total,        0k used,   524280k free,   409324k cached
PID to kill:15273                          ← 按“k”键，会提示输入要杀死进程的 PID
```

如果在操作终端执行 top 命令，并不能看到系统中所有的进程，默认看到的只是 CPU 占比靠前的进程。如果我们想要看到所有的进程可以把 top 命令的结果重定向到文件当中即可。不过 top 命令是持续运行的，这时就需要“-b”和“-n”选项了，具体命令如下：

```
[root@localhost ~]# top -b -n 1 > /root/top.log
#让 top 命令只执行一次，让后把结果保存到 top.log 文件中。这样就能看到所有的进程了
```

3）、 pstree 命令

```
[root@localhost ~]# pstree [选项]
选项：
    -p:     显示进程的 PID
    -u:     显示进程的所属用户
```

3 进程的管理

系统中可以识别的信号较多，我们可以使用命令“kill -l”或“man 7 signal”来查询，命令如下：

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

这里我们介绍一下主要的信号，如表 15-6 所示：

信号代号	信号名称	说明
1	SIGHUP	该信号让进程立即关闭，然后重新读取配置文件之后重启。
2	SIGINT	程序终止信号，用于终止前台进程。相当于输出 ctrl+c 快捷键。
8	SIGFPE	在发生致命的算术运算错误时发出. 不仅包括浮点运算错误，还包括溢出及除数为 0 等其它所有的算术的错误。
9	SIGKILL	用来立即结束程序的运行. 本信号不能被阻塞、处理和忽略。一般用于强制终止进程。
14	SIGALRM	时钟定时信号，计算的是实际的时间或时钟时间. alarm 函数使

		用该信号。
15	SIGTERM	正常结束进程的信号，kill 命令的默认信号。有时如果进程已经发生问题，这个信号是无法正常终止进程的，我们才会尝试 SIGKILL 信号，也就是信号 9。
18	SIGCONT	该信号可以让暂停的进程恢复执行，本信号不能被阻断。
19	SIGSTOP	该信号可以暂停前台进程，相当于输入 ctrl+z 快捷键。本信号不能被阻断。

1）、 kill 命令

```
[root@localhost ~]# kill [信号] PID
```

例 2：使用“-1”信号，让进程重启

```
[root@localhost ~]# kill -1 2246
#使用“-1（数字一）”信号，让 httpd 的主进程重启动。
```

例 3：使用“-19”信号，让进程暂停。

```
[root@localhost ~]# vi test.sh
#使用 vi 命令编辑一个文件，不要退出

[root@localhost ~]# ps aux | grep "vi" | grep -v "grep"
root       2313  0.0  0.2   7116  1544 pts/1    S+   19:20   0:00 vi test.sh
#换一个不同的终端，查看下这个这个进程的状态。进程状态是 S（休眠）和+（位于后台），
#因为是在另外一个终端运行的命令

[root@localhost ~]# kill -19 2313
#使用-19 信号，让 PID 是 2313 的进程暂停。相当于在 vi 界面按“ctrl+z”快捷键。

[root@localhost ~]# ps aux | grep "vi" | grep -v "grep"
root       2313  0.0  0.2   7116  1580 pts/1    T    19:20   0:00 vi test.sh
#注意 2313 进程的状态，变成了 T（暂停）状态。这时我切换回 vi 的终端，发现 vi 命令已
#经暂停，又回到了命令提示符。
#不过 2313 进程就会卡在后台，如果想要恢复可以使用“kill -9 2313”强制终止进程，也
#可以利用 15.2 小节将要学习的工作管理来进行恢复。
```

2）、killall 命令

```
[root@localhost ~]# killall [选项][信号] 进程名
选项：
    -i：     交互式，询问是否要杀死某个进程
    -I：     忽略进程名的大小写
```

交互式杀死 sshd 进程

```
[root@localhost ~]# ps aux | grep "sshd"  | grep -v "grep"
root       1733  0.0  0.1   8508  1008 ?        Ss   19:47   0:00 /usr/sbin/sshd
```

```
root      1735  0.1  0.5  11452  3296 ?        Ss   19:47   0:00 sshd: root@pts/0
root      1758  0.1  0.5  11452  3296 ?        Ss   19:47   0:00 sshd: root@pts/1
#查询系统有 3 个 sshd 进程。1733 是 sshd 服务的进程，1735 和 1758 是我的两个远程连接
#的进程。

[root@localhost ~]# killall -i sshd
#交互式杀死 sshd 进程
杀死 sshd(1733) ? (y/N) n
#这个进程是 sshd 的服务进程，如果杀死，所有的 sshd 连接都不能登录。
杀死 sshd(1735) ? (y/N) n
#这是我当前登录终端，不能杀死我自己吧！
杀死 sshd(1758) ? (y/N) y
#把另外一个 sshd 登录终端踢出。
```

3）、pkill 命令

pkill 命令和 killall 命令非常类似，也是按照进程名来杀死进程。格式如下：

```
[root@localhost ~]# pkill [选项] [信号] 进程名
选项：
    -t 终端号：  按照终端号踢出用户
```

```
[root@localhost ~]# pkill -9 -t pts/1
#强制杀死从 pts/1 虚拟终端登录的进程
```

二、工作管理

1　工作管理简介

后台管理有几个事项需要大家注意：

- ✧ 前台是指当前可以操控和执行命令的这个操作环境，后台是指工作可以自行运行，但是不能直接用 ctrl+c 来终止它，只能使用 fg/bg 来调用工作；
- ✧ 当前的登录终端，只能管理当前终端的工作，而不能管理其他登录终端的工作。如 tty1 登录的终端是不能管理 tty2 终端中的工作的；
- ✧ 放入后台的命令必须可以持续运行一段时间，这样我们才能扑捉和操作这个工作。如果把 ls 命令放入后台执行，它很快就会执行完成，我们很难操作它。
- ✧ 放入后台执行的命令不能和前台用户有交互或需要前台输入，否则放入后台只能暂停，而不能执行。比如 vi 命令放入后台只能暂停，而不能执行，因为 vi 需要前台输入信息。top 命令也不能放入后台执行，而只能放入后台暂停，因为 top 命令需要和前台有交互。

2　如何把命令放入后台

那么我们如何把命令放入后台呢？有两种方法，分别如下：

1)、　　第一种方法是“命令 &”，把命令放入后台执行

第一种把命令放入后台的方法是在命令后面加入“空格&”，这种方法放入后台的命令，在后台是执行状态。但是注意，放入后台执行的命令不能与前台有交互，否则这个命令是不能在后台执行的。举个例子：

```
[root@localhost ~]# find / -name install.log &
[1] 1920
#[工作号] 进程号
#find 命令放入后台执行，每个后台命令会分配一个工作号，命令既然可以执行，就会有进
#程产生，所以也会有进程号。
```

这样的话，虽然 find 命令在执行，但是当前终端我还是可以进行其他命令操作的。如果在终端上出现：

```
[1]+  Done                    find / -name install.log
```

证明后台这个任务已经完成了，当然命令如果有执行结果的话，也会显示到操作终端上。[1]是这个命令的工作号，+号代表这个任务是最近一个被放入后台的工作。

2)、 第二种方法是在命令执行过程中按 ctrl+z 快捷键，命令在后台是暂停状态

使用这种方法放入后台的命令，就算是不和前台有交互，能在后台执行的命令，也是暂停状态，ctrl+z 快捷键就是暂停的快捷键。举几个例子：

```
例 1：
[root@localhost ~]# top
#在 top 命令执行的过程中，按下 ctrl+z 快捷键
[1]+  Stopped                 top
#提示命令被放入后台，工作号是 1，状态是暂停。而且虽然 top 命令没有结束，也能取得
#控制台权限。

例 2：
[root@localhost ~]# tar -zcf etc.tar.gz /etc
#压缩一下/etc/目录
tar: 从成员名中删除开头的“/”
tar: 从硬连接目标中删除开头的“/”
^Z                            ← 执行的过程中，按下 ctrl+z 快捷键
[2]+  Stopped                 tar -zcf etc.tar.gz /etc
#命令被放入后台，工作号是 2，状态是暂停。
```

3 后台命令管理

1)、 查看后台的工作

```
[root@localhost ~]# jobs [-l]
选项：
    -l：    显示工作的 PID
```

2）、 将后台暂停的工作恢复到前台执行

```
[root@localhost ~]# fg %工作号
参数：
    %工作号：    %号可以省略，但是注意工作号和 PID 的区别
```

3）、 把后台暂停的工作恢复到后台执行

```
[root@localhost ~]# bg %工作号
```

4）、 后台命令脱离登录终端运行

我们已经知道把命令放入后台，只能在当前登录终端执行。那如果我是远程管理的服务器，在远程终端中执行了后台命令，这时我退出登录，这个后台命令还能继续执行吗？当然是不行的，这个后台命令会被终止。但是我们确实需要在远程终端中执行某些后台命令，该如何执行呢？

- ✧ 第一种方法是把需要后台执行的命令加入/etc/rc.local 文件，让系统在启动时执行这个后台程序。这种方法的问题是，服务器时不能随便重启的，万一有临时后台任务，就不能执行。
- ✧ 第二种方法是使用系统定时任务，让系统在指定的时间执行某个后台命令。这样放入后台的命令与终端无关，是不依赖登录终端的。
- ✧ 最后一种方法是使用 nohup 命令。

nohup 命令的作用就是让后台工作在离开操作终端时，也能够正确的在后台执行。命令的格式如下：

```
[root@localhost ~]# nohup [命令] &

例如：
[root@localhost ~]# nohup find / -print > /root/file.log &
[3] 2349                                    ← 用 find 命令，打印/下所有文件。放入后台执行
[root@localhost ~]# nohup: 忽略输入并把输出追加到"nohup.out"
#有提示信息
```

三、系统资源查看

1 vmstat 命令监控系统资源

```
[root@localhost ~]# vmstat [刷新延时 刷新次数]

例如：
[root@localhost proc]# vmstat 1 3
#使用 vmstat 检测，每 1 秒刷新一次，共刷新 3 次
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 407376  55772  84644    0    0     5     2    9   10  0  0 100  0  0
 0  0      0 407368  55772  84644    0    0     0     0   12   10  0  0 100  0  0
 0  0      0 407368  55772  84644    0    0     0     0   15   13  0  0 100  0  0
```

解释下这个命令的输出：

- ✧ procs：进程信息字段：
  - ➢ r：等待运行的进程数，数量越大，系统越繁忙。
  - ➢ b：不可被唤醒的进程数量，数量越大，系统月繁忙。
- ✧ memory：内存信息字段：
  - ➢ swpd：虚拟内存的使用情况，单位 KB。
  - ➢ free：空闲的内存容量，单位 KB。
  - ➢ buff：缓冲的内存容量，单位 KB。
  - ➢ cache：缓存的内存容量，单位 KB。

- ✧ swap：交换分区的信息字段：
  - ➢ si：从磁盘中交换到内存中数据的数量，单位 KB。
  - ➢ so：从内存中交换到磁盘中数据的数量，单位 KB。此两个数越大，证明数据需要经常在磁盘和内存之间交换，系统性能越差。
- ✧ io：磁盘读写信息字段：
  - ➢ bi：从块设备读入数据的总量，单位是块。
  - ➢ bo：写到块设备的数据的总量，单位是块。此两个数越大，代表系统的 I/O 越繁忙。
- ✧ system：系统信息字段：
  - ➢ in：每秒被中断的进程次数。
  - ➢ cs：每秒钟进行的事件切换次数。此两个数越大，代表系统与接口设备的通信非常繁忙。
- ✧ CPU：CPU 信息字段：
  - ➢ us：非内核进程消耗 CPU 运算时间的百分比。
  - ➢ sy：内核进程消耗 CPU 运算时间的百分比。
  - ➢ id：空闲 CPU 的百分比。
  - ➢ wa：等待 I/O 所消耗的 CPU 百分比。
  - ➢ st：被虚拟机所盗用的 CPU 占比。

2 dmesg 显示开机时内核检测信息

```
例如：
[root@localhost ~]# dmesg | grep CPU
#查看 CPU 信息
[root@localhost ~]# dmesg | grep eth0
#查询第一块网卡信息
```

3 free 命令查看内存使用状态

```
[root@localhost ~]# free [-b|-k|-m|-g]
选项：
    -b：    以字节为单位显示
    -k：    以 KB 为单位显示，默认就是以 KB 为单位显示
    -m：    以 MB 为单位显示
    -g：    以 GB 为单位显示

[root@localhost ~]# free
             total       used       free     shared    buffers     cached
Mem:        625344     219704     405640          0      56852      85276
-/+ buffers/cache:      77576     547768
Swap:       524280          0     524280
```

解释下这个命令：

第一行：total 是总内存数，used 是已经使用的内存数，free 是空闲的内存数，shared 是多个进程共享的内存总数，buffers 是缓冲内存数，cached 是缓存内存数。默认单位是 KB。

第二行：-/buffers/cache 的内存数，相当于第一行的 used-buffers-cached。+/buffers/cache 的内存数，相当于第一行的 free+buffers+cached。

第三行：total 是 swap 的总数；used 是已经使用的 swap 数，free 是空闲的 swap 数。默认单位

是 KB。

4　查看 CPU 信息

CPU 的主要信息保存在/proc/cpuinfo 这个文件当中，我们只要查看这个文件，就可以知道 cpu 的相关信息。命令如下：

```
[root@localhost ~]# cat /proc/cpuinfo
processor       : 0
#逻辑 CPU 编号
vendor_id       : GenuineIntel
#CPU 制造厂商
cpu family      : 6
#产品的系列代号
model           : 58
#CPU 系列代号
model name      : Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz
#CPU 系列的名字，编号，主频
stepping        : 9
#更新版本
cpu MHz         : 2394.649
#实际主频
cache size      : 6144 KB
#二级缓存
```

5　查看本机登陆用户信息

如果我们想要知道 Linux 服务器上目前已经登录的用户信息,可以使用 w 或 who 命令来进行查询。先看看 w 命令，命令格式如下：

```
[root@localhost ~]# w
 09:54:22 up  9:50,  2 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1     -                 09:53   57.00s  0.02s  0.02s -bash
root     pts/0    192.168.0.108     09:53    0.00s  0.09s  0.06s w
```

解释下这个命令的结果：

- ✧ 第一行其实和 top 命令的第一行非常类似，主要显示了系统当前时间，开机多久（up），有多少用户登陆（users），1 分钟、5 分钟、15 分钟前的平均负载（load average）。
- ✧ 第二行是项目的说明，第三行开始每行代表一个用户。这些项目是：
  - ➢ USER：登陆的用户名；
  - ➢ TTY：登陆终端；
  - ➢ FROM：从哪个 IP 地址登陆；
  - ➢ LOGIN@：登陆时间；
  - ➢ IDLE：用户闲置时间；
  - ➢ JCPU：指的是和该终端连接的所有进程占用的 CPU 运算时间。这个时间里并不包括过去的后台作业时间，但却包括当前正在运行的后台作业所占用的时间；
  - ➢ PCPU：是指当前进程所占用的 CPU 运算时间；

- WHAT：当前正在运行的命令。

who 命令比 w 命令稍微简单一点，也可以用来查看系统中已经登录的用户。命令如下：

```
[root@localhost ~]# who
root     tty1         2018-06-07 09:53
root     pts/0        2018-06-07 09:53 (192.168.0.108)
#用户名   登录终端      登录时间（登录来源 IP 地址）
```

6 uptime 命令

uptime 命令的作用就是显示系统的启动时间和平均负载，也就是 top 命令的第一行。其实 w 命令也能看到这一行的数据，具体原意使用哪个命令看个人习惯。命令如下：

```
[root@localhost ~]# uptime
 10:56:30 up 10:52,  2 users,  load average: 0.00, 0.00, 0.00
```

7 查看系统与内核相关信息

我们使用 uname 命令查看内核的相关信息，命令如下：

```
[root@localhost ~]# uname [选项]
选项：
    -a:     查看系统所有相关信息；
    -r:     查看内核版本；
    -s:     查看内核名称。
```

如果我们想要判断当前系统的位数，可以通过 file 命令来判断系统文件（主要是系统命令）的位数来推断系统的位数。命令如下：

```
[root@localhost ~]# file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
#很明显，当前系统是 32 位的。
```

如果我们想要查询当前 Linux 系统的发行版本，可以使用“lsb_release -a”命令查看，命令如下：

```
[root@localhost ~]# lsb_release -a
```

四、系统定时任务

1 at 一次性执行定时任务

1)、 at 服务管理与访问控制

at 命令要想正确执行，需要 atd 服务的支持。atd 服务是独立的服务，所以启动的命令是：

```
[root@localhost ~]# service atd start
正在启动 atd:                                              [确定]
```

如果想要让 atd 服务开机时自动启动，可以使用如下命令：

```
[root@localhost ~]# chkconfig atd on
```

atd 服务启动之后，at 命令才可以正常使用，不过我们还要学习下 at 命令的访问控制。这里的访问控制指的是允许哪些用户使用 at 命令设定定时任务，或者不允许哪些用户使用 at 命令。大家可

以想象成为设定黑名单或设定白名单，这样更容易理解。at 的访问控制是依靠/etc/at.allow 文件（白名单）和/etc/at.deny 文件（黑名单）这两个文件来实现的，具体规则如下：

- ✧ 如果系统中有/etc/at.allow 文件，那么只有写入/etc/at.allow 文件（白名单）中的用户可以使用 at 命令，其他用户不能使用 at 命令（/etc/at.deny 文件会被忽略，也就是说同一个用户既写入/etc/at.allow 文件，也写入/etc/at.deny 文件，那么这个用户是可以使用 at 命令的，因为/etc/at.allow 文件优先级更高。）。
- ✧ 如果系统中没有/etc/at.allow 文件，只有/etc/at.deny 文件，那么写入/etc/at.deny 文件（黑名单）中的用户不能使用 at 命令，其他用户可以使用 at 命令。不过这个文件对 root 用户不生效。
- ✧ 如果系统中这两个文件都不存在，那么只有 root 用户可以使用 at 命令。

系统中默认时只有/etc/at.deny 文件，而且这个文件是空的，这样的话系统中所有的用户都可以使用 at 命令。不过如果我们打算控制用户的 at 命令权限，把用户写入/etc/at.deny 文件即可

/etc/at.allow 文件的权限更高，如果/etc/at.allow 文件存在，则/etc/at.deny 文件失效。/etc/at.allow 管理行为更加严格，因为只有写入这个文件的用户才能使用 at 命令，如果需要禁用 at 命令的用户较多，则可以把少数用户写入这个文件。/etc/at.deny 文件的管理较为松散，如果允许使用 at 命令的用户较多，则可以把禁用的用户写入这个文件。不过这两个文件都不能对 root 用户生效。

2）、 at 命令

```
[root@localhost ~]# at [选项] 时间
选项：
    -m:             当 at 工作完成后，无论是否命令有输出，都用 email 通知执行 at
                命令的用户
    -c 工作号:      显示该 at 工作的实际内容
时间：
    at 支持的时间格式如下：
    HH:MM               在指定的“小时:分钟”执行命令，例如：02:30
    HH:MM YYYY-MM-DD    在指定的“小时:分钟 年-月-日”执行，例如 02:30 2018-07-25
    HH:MM[am|pm] [month] [date]     在指定的“小时:分钟[上午|下午][月][日]”执行，
                                    例如 02:30 July 25
    HH:MM[am|pm] + [minutes|hours|days|weeks]   在指定的时间“再加多久执行”，例
                                                如 now + 5 minutes，05am +2 hours
```

at 命令只要指定正确的时间，就可以输入需要在指定时间执行的命令了，这个命令可以是系统命令，也可以是 shell 脚本。举几个例子吧：

```
例子 1：
[root@localhost ~]# cat /root/hello.sh
#!/bin/bash
echo "hello world!!"
#该脚本会打印“hello workd！！”

[root@localhost ~]# at now +2 minutes
at> /root/hello.sh >> /root/hello.log
```

```
#执行 hello.sh 脚本，并把输出写入/root/hello.log 文件
at> <EOT>                                ←使用 ctrl+d 保存 at 任务
job 8 at 2018-07-25 20:54                ←这是第 8 个 at 任务，会在 2018 年 7 月 25 日 20:54
                                         ←分执行

[root@localhost ~]# at -c 8
#查询第 8 个 at 任务的内容
…省略部分内容…                           ←主要是定义系统的环境变量
/root/hello.sh >> /root/hello.log
#可以看到 at 执行的任务

例子 2：
[root@localhost ~]# at 02:00 2018-07-26
at> /bin/sync
at> /sbin/shutdown -h now
at> <EOT>
job 9 at 2018-07-26 02:00
#在指定的时间关机。在一个 at 任务中，是可以执行多个系统命令的
```

3）、 其他 at 管理命令

at 还有查询和删除命令，命令如下：

```
[root@localhost ~]# atq
#查询当前服务器上的 at 工作

例如：
[root@localhost ~]# atq
9       2018-07-26 02:00 a root
#说明 root 用户有一个 at 任务在 2018 年 7 月 26 日的 02:00 执行，工作号是 9

[root@localhost ~]# atrm [工作号]
#删除指定的 at 任务

例如：
[root@localhost ~]# atrm 9
[root@localhost ~]# atq
#删除 9 号 at 任务，再查询就没有 at 任务存在了
```

2 crontab 循环执行定时任务

1）、 crond 服务管理与访问控制

crontab 命令是需要 crond 服务支持的，crond 服务同样是独立的服务，所以启动和自启动方法如下：

```
[root@localhost ~]# service crond restart
停止 crond:                                                [确定]
```

```
正在启动 crond：                                           [确定]
#重启动 crond 服务

[root@localhost ~]# chkconfig crond on
#设定 crond 服务为开机自启动
```

crond 服务默认是自启动的，如果服务器上有循环执行的系统定时任务，就不要关闭 crond 服务了。

crontab 命令和 at 命令类似，也是通过/etc/cron.allow 和/etc/cron.deny 文件来限制某些用户是否可以使用 crontab 命令的。而且原则也非常相似：

- ✧ 当系统中有/etc/cron.allow 文件时，只有希尔此文件的用户可以使用 crontab 命令，没有写入的用户不能使用 crontab 命令。同样如果有此文件，/etc/cron.deny 文件会被忽略，/etc/cron.allow 文件的优先级更高。
- ✧ 当系统中只有/etc/cron.deny 文件时，则写入此文件的用户不能使用 crontab 命令，没有写入文件的用户可以使用 crontab 命令。

这个规则基本和 at 的规则一直，同样/etc/cron.allow 文件的优先级比/etc/cron.deny 文件的优先级高，Linux 中默认只有/etc/cron.deny 文件。

2）、 用户的 crontab 设置

每个用户都可以实现自己的 crontab 定时任务，只要是使用这个用户身份执行“crontab -e”命令即可，当然这个用户不能加入/etc/cron.deny 文件中。crontab 命令格式如下：

```
[root@localhost ~]# crontab [选项]
选项：
    -e:          编辑 crontab 定时任务
    -l:          查询 crontab 任务
    -r:          删除当前用户所有的 crontab 任务，如果有多个任务，只想删除一个，可
                 以所以用“crontab -e”
    -u 用户名:   修改或删除其他用户的 crontab 任务。只有 root 可用
```

```
[root@localhost ~]# crontab -e
#进入 crontab 编辑界面。会打开 vim 编辑你的工作。
* * * * * 执行的任务
```

项目	含义	范围
第一个“*”	一小时当中的第几分钟	0-59
第二个“*”	一天当中的第几小时	0-23
第三个“*”	一个月当中的第几天	1-31
第四个“*”	一年当中的第几月	1-12
第五个“*”	一周当中的星期几	0-7（0 和 7 都代表星期日）

在时间表示时，还有一些特殊符号需要学习，如表 15-8 所示：

特殊符号	含义
*	代表任何时间。比如第一个“*”就代表一小时中每分钟都执行一次的意思。

,	代表不连续的时间。比如“0 8,12,16 * * * 命令”，就代表在每天的 8 点 0 分，12 点 0 分，16 点 0 分都执行一次命令
-	代表连续的时间范围。比如“0 5 * * 1-6 命令”，代表在周一到周六的凌晨 5 点 0 分执行命令
*/n	代表每隔多久执行一次。比如“*/10 * * * * 命令”，代表每隔 10 分钟就执行一遍命令

举例

时间	含义
45 22 * * * 命令	在 22 点 45 分执行命令
0 17 * * 1 命令	每周 1 的 17 点 0 分执行命令
0 5 1,15 * * 命令	每月 1 号和 15 号的凌晨 5 点 0 分执行命令
40 4 * * 1-5 命令	每周一到周五的凌晨 4 点 40 分执行命令
*/10 4 * * * 命令	每天的凌晨 4 点，每隔 10 分钟执行一次命令
0 0 1,15 * 1 命令	每月 1 号和 15 号，每周 1 的 0 点 0 分都会执行命令。注意：星期几和几号最好不要同时出现，因为他们定义的都是天。非常容易让管理员混乱。

例子 1：让系统每隔 5 分钟，就向/tmp/test 文件中写入一行“11”，验证一下系统定时任务是否会执行

```
[root@localhost ~]# crontab -e
#进入编辑界面
*/5 * * * * /bin/echo ”11” >> /tmp/test
```

例子 2：在系统在每周二的凌晨 5 点 05 分重启一次

```
[root@localhost ~]# crontab -e
5 5 * * 2 /sbin/shutdown -r now
```

例子 3：在每月的 1 号，10 号，15 号的凌晨 3 点 30 分都定时执行日志备份脚本 autobak.sh

```
[root@localhost ~]# crontab -e
30 3 1,10,15 * * /root/sh/autobak.sh
[root@localhost ~]# crontab -l
#查看 root 用户的 crontab 任务
[root@localhost ~]# crontab -r
#删除 root 用户所有的定时任务，如果只想删除某一个定时任务，
#可以“crontab -e”进入编辑模式手工删除
```

3）、 crontab 的注意事项

在书写 crontab 定时任务时，需要注意几个注意事项，这里我们再强调下：

- ✧ 六个选项都不能为空，必须填写。如果不确定使用“*”代表任意时间；
- ✧ crontab 定时任务，最小有效时间是分钟，最大时间范围是月。像 2018 年某时执行，3 点 30 分 30 秒这样的时间都不能识别；
- ✧ 在定义时间时，日期和星期最好不要在一条定时任务中出现，因为它们都是以天作为单位，非常容易让管理员混乱；
- ✧ 在定时任务中，不管是直接写命令，还是在脚本中写命令，最好都是用绝对路径。有时相对

路径的命令会报错。

4、 系统的 crontab 设置

“crontab -e”是每个用户执行的命令，也就是说不同的用户身份可以执行自己的定时任务。可是有些定时任务需要系统执行，这时我们就需要编辑/etc/crontab 这个配置文件了。当然，并不是说写入/etc/crontab 配置文件中的定时任务执行时，不需要用户身份，而是“crontab -e”命令定义定时任务时，默认用户身份是当前登录用户。而修改/etc/crontab 配置文件时，定时任务的执行着身份是可以手工指定的。这样定时任务的执行会更加灵活，修改起来也更加方便。

那我们打开这个文件看看吧：

```
[root@localhost ~]# vi /etc/crontab
SHELL=/bin/bash
#标示使用哪种 shell
PATH=/sbin:/bin:/usr/sbin:/usr/bin
#指定 PATH 环境变量，crontab 是使用自己的 PATH，而不是用系统默认的，所以在定时任务中出现的命令最好使用大写
MAILTO=root
#如果有报错输出，或命令结果又输出，会向 root 发信息
HOME=/
#标示主目录

# For details see man 4 crontabs
#提示大家可以去“man 4 crontabs”查看帮助

# 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
#分  时 日 月 周 执行者身份 命令
#列出文件格式，并加入了注释
```

在 CentOS 6.x 中，/etc/crontab 这个文件也发生了变化，在 CentOS 5.x 中这个文件的内容大概是这个样子：

```
#以下输出，是在 CentOS 5.5 当中
[root@sc ~]# vi /etc/crontab
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
```

```
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

在 CentOS 5.x 版本中，/etc/crontab 文件会使用 run-parts 脚本执行/etc/cron.{daily,weekly,monthly}目录中的所有文件。这个 run-parts 其实是一个 shell 脚本，保存在/usr/bin/run-parts，它的作用就是把其后面跟随的目录中的所有可执行文件依次执行。也就是说，如果我们想让某个脚本在每天都执行一次，可以不用手工去写定时任务，而只需要给这个脚本赋予执行权限，并把它放入/etc/cron.daily/目录中。这样这个脚本就会在每天的凌晨 4 点 02 分执行了。

可是在 CentOS 6.x 版本中，/etc/crontab 文件中不再有相关的段落，那么是否 run-ptars 这种定时任务执行方法不可用了呢？我们看到/etc/crontab 中有一句提示，让我们“man 4 crontabs”来查看帮助，那么我们就看看这个帮助吧！在这个帮助中，明确写明了“在旧版本的 crontab 配置文件中，是通过 run-parts 脚本来调用 cron.{daily,weekly,monthly}目录，定时执行这些目录中的脚本。在当前系统中，为了避免 cron 和 anacron 之间产生冲突，只要 anacron 已经安装，就使用 anacron 来执行这些目录中的脚本。具体可以查看 anacron（8）的帮助。”。对于 anacron 的用法，我们下一个小节来介绍。

其实对我们用户来讲，我们并不需要知道这个定时任务到底是由哪个程序调用的。我们需要知道的事情是如何使用系统的 crontab 设置。这个新老版本的 CentOS 没有区别，配置方法都有两种：

- ✧ 第一种方法就是我刚刚说过的，把你需要定时执行的工作写成脚本程序，并赋予执行权限，然后直接把这个脚本复制到/etc/cron.{daily,weekly,monthly}目录中的任意一个。比如我需要让某个脚本每周执行，那么就把这个脚本复制到/etc/cron.weekly/目中中。这样这个脚本就会每周执行一次，具体的执行时间要参考 anacron 的配置。
- ✧ 第二种方法就是修改/etc/crontab 这个配置文件，自己加入自己的定时任务，不过需要注意指定脚本的执行者身份。例如：

```
[root@localhost ~]# vi /etc/crontab
…省略部分输出…
* * * * * root run-parts /root/cron/
#让系统每分钟都执行一次/root/cron/目录中的脚本，脚本执行者是 root 用户。
#虽然在 CentOS 6.x 中，不是直接通过此配置文件调用/etc/cron.{daily,weekly,monthly}这些目录，
#但是 run-parts 脚本还是可以使用的。所以我完全可以参照 CentOS 5.x 的方法来写定时任务。
#使用 run-parts 脚本调用并执行/root/cron/目录中所有的可执行文件

[root@localhost ~]# mkdir cron
#建立/root/cron 目录
[root@localhost cron]# vi /root/cron/hello.sh
#/bin/bash
echo "hello" >> /root/cron/hello.log
#在/root/cron/hello.log 文件中写入 hello
[root@localhost cron]# chmod 755 hello.sh
#赋予执行权限。
#因为 hello.sh 脚本放入了/root/cron/目录，所以会每分钟执行一次。
```

只要保存/etc/crontab 文件，这个定时任务就可以执行了，当然要确定 crond 服务是运行的。

这两种方法都是可以使用的，具体看个人的习惯吧！不过要想修改/etc/crontab 文件，当然我要

是 root 用户才可以，普通不能修改，只能使用用户身份的 crontab 命令。

3　anacron

anacron 是用来干什么的呢？我们的 Linux 服务器如果不是 24 小时开机的，那么刚好在关机的时间段之内有系统定时任务（cron）需要执行，那么这些定时任务是不会执行的。也就是说，假设我们需要在凌晨 5 点 05 分执行系统的日志备份，但是我们的 Linux 服务器不是 24 小时开机的，在晚上需要关机，白天上班之后才会再次开机。这个定时任务的执行时间我们的服务器刚好没有开机，那么这个任务就不会执行了。anacron 就是用来解决这个问题的。

anacron 会使用一天，七天，一个月作为检测周期，用来判断是否有定时任务在关机之后没有执行，如果有这样的任务，anacron 会在特定的时间重新执行这些定时任务。那么 anacron 是如何判断这些定时任务已经超过了执行时间呢？在系统的/var/spool/anacron/目录中存在 cron.{daily,weekly,monthly}文件，这些文件中都保存着 anacron 上次执行时的时间。anacron 会去读取这些文件中的时间，然后和当前时间做比较，若果两个时间的差值超过了 anacron 的指定时间差值（一般是 1 天，7 天和一个月），就说明有定时任务漏掉了没有被执行，这时 anacron 会介入而执行这个漏掉的定时任务，从而保证在关机时没有被执行的定时任务不会被漏掉。

在 CentOS 6.x 中，我们使用 cronie-anacron 取代了 *vixie-cron 软件包。而且在原先的 CentOS 版本中/etc/*cron.{daily,weekly,monthly}这些目录中的定时任务会同时被 cron 和 anacron 调用，这样非常容易出现重复执行同一个定时任务的错误。在现在的 CentOS 6.x 中，*/etc/*cron.{daily,weekly,monthly}目录中的定时任务程序只会被 anacron 调用，从而保证这些定时任务只会在每天、每周或每月被定时执行一次，而不会重复执行。这也是我们在上一个小节中介绍的 CentOS 6.x 的变化之一。

在 CentOS 6.x 中 anacron 还有一个变化，anacron 不再是单独的服务，而变成了系统命令。也就是说我们不再可以使用“service anacron restart”命令来管理 anacron 服务了。而是需要使用 anacron 命令来管理 anacron 工作，具体命令如下：

```
[root@localhost ~]# anacron [选项] [工作名]
选项：
    -s:     开始 anacron 工作，依据/etc/anacrontab 文件中的设定的延迟时间执行
    -n:     立即执行/etc/anacrontab 中所有的工作，忽略所有的延迟时间
    -u:     更新/var/spool/anacron/cron.{daily,weekly,monthly}文件中的时间戳，但
            不执行任何工作
参数：
    工作名:     是依据/etc/anacrontab 文件中定义的工作名
```

在我们当前的 Linux 中，其实不需要执行任何 anacron 命令，只需要配置好/etc/anacrontab 文件，系统就会依赖这个文件中的设定来通过 anacron 执行定时任务。那么关键就是/etc/anacrontab 文件的内容了，这个文件如下：

```
[root@localhost ~]# vi /etc/anacrontab
# /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
```

```
#前面的内容和/etc/crontab 类似

# the maximal random delay added to the base delay of the jobs
RANDOM_DELAY=45
#最大随机延迟。
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-22
#anacron 的执行时间范围是 3:00-22:00

#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
#天数   强制延迟（分）   工作名称              实际执行的命令
#时间差超过天数时，强制延迟多少分钟之后，就执行命令
```

这个文件中“RANDOM_DELAY”定义的是最大随机延迟，也就是说 cron.daily 任务如果超过 1 天没有执行，并不会马上执行，而是先延迟强制延迟时间之后，再延迟随机延迟时间之后再执行命令。“START_HOURS_RANGE”定义 anacron 的执行时间范围，anacron 只会在这个时间范围之内执行。

我们用 cron.daily 工作来说明下/etc/anacrontab 的执行过程：

- ✧ 首先读取/var/spool/anacron/cron.daily 中的上一次 anacron 执行的时间；
- ✧ 和当前时间比较，如果两个时间的差值超过 1 天，就执行 cron.daily 工作；
- ✧ 执行这个工作只能在 03:00-22:00 之间；
- ✧ 执行工作时强制延迟时间为 5 分钟，再随机延迟 0-45 分钟时间；
- ✧ 使用 nice 命令指定默认优先级，使用 run-parts 脚本执行/etc/cron.daily 目录中的所有可执行文件。

大家发现了吧，*/etc/*cron.{daily,weekly,monthly}中的脚本，在当前的 Linux 中是被 anacron 调用的，不再依靠 cron 服务。不过 anacron 不用设置多余的配置，我们只需要把需要定时执行的脚本放入*/etc/*cron.{daily,weekly,monthly}目录当中，就会每天、每周或每月执行，而且也不再需要启动 anacron 服务了。我们如果做修改的话，只用修改/etc/anacrontab 配置文件即可。比如我更加习惯让定时任务在凌晨 03:00-05:00 执行，就可以如下修改：

```
[root@localhost ~]# vi /etc/anacrontab
# /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=0
#随机延迟改为 0 分钟，不再随机延迟
# the jobs will be started during the following hours only
START_HOURS_RANGE=3-5
```

```
#执行时间范围为 03:00-05:00

#period in days   delay in minutes   job-identifier   command
1       0       cron.daily              nice run-parts /etc/cron.daily
7       0       cron.weekly             nice run-parts /etc/cron.weekly
@monthly 0      cron.monthly            nice run-parts /etc/cron.monthly
#强制延迟也改为 0 分钟，不再强制延迟
```

这样我们所有放入/etc/cron.{daily,weekly,monthly}目录中的脚本都会在指定时间运行了，而且也不怕服务器万一关机的情况了。