第9章　核心级系统服务

只要是Linux系统，不管使用的发行版本、网络配置以及系统全局设计有什么不同，都有如下几个核心系统服务：init、inetd、syslog和cron。这些服务提供的功能比较简单，但同时它们也是其他操作的基础。如果没有它们，Linux就不会有现在的威力。

本章将对每一个核心系统服务、相应的配置文件以及推荐的应用方式（如果有的话）逐个进行介绍。你将会发现对这些简单服务的讨论并不是长篇大论，但是千万不要轻视这些内容。我们强烈推荐：务必要花上一些时间熟悉这些服务的实现原理（可以利用每天早晨上班时的塞车时间嘛）。通过使用这些服务，人们已经设计出了许多富有创意的问题解决方案。在本章中我们将向大家介绍其中的一些。

9.1　init服务

init进程是所有进程的发起者和控制者。因为在任何基于 UNIX的系统（比如Linux）中它都是第一个运行的进程，所以init的进程编号（process ID，PID）永远是1。如果init出现了问题，系统的其余部分也就随之而垮掉了。

init进程有两个作用：第一是扮演终极父进程的角色。因为 init进程永远不会被终止，所以系统总是可以确信它的存在，并在必要的时候以它为参照。如果某个进程在它衍生出来的全部子进程结束之前被终止，经常就会出现必须以 init为参照的情况。此时那些失去了父进程的子进程就都会以 init作为它们的父进程。快速执行一下 ps -af 命令可以列出许多父进程 ID（parent process ID，PPID）为1的进程来。

init的第二个角色是在进入某个特定的运行级别（runlevel）时运行相应的程序，以此对各种运行级别进行管理。它的这个作用是由 /etc/inittab文件定义的。

9.1.1　/etc/inittab文件

/etc/inittab文件中包括了所有 init启动运行级别所必须的信息。这个文件中的每一行语句的格式如下所示：

```
id : runlevels: action : process
```

注意　以#开始的语句是注释语句。看看你自己的 /etc/inittab文件就可以发现其中充斥了大量的注释语句。如果确实需要对 /etc/inittab文件进行什么改动（一般不会出现这种情况的），记住加上一些注释语句解释为什么要做那些修改的原因。

表9-1中列出了/etc/inittab文件语句格式组成部分的说明。

现在来看看/etc/inittab文件中的一个示例语句：

```
# If power was restored before the shutdown kicked in , cancel it .
pr : 12345 : powerokwait : /sbin/shutdown -c " Power Restored;Shutdown Cancelled ."
```

在这个例子中，

- pr是独一无二的标识符。
- 1、2、3、4和5是进程可以从中被调用的运行级别。
- powerokwait是运行这个进程的条件。
- /sbin/shutdown . . .命令是需要运行的进程。

表9-1 /etc/inittab文件语句格式组成部分的说明

/etc/inittab元素	说 明
id	由1到4个字符组成的、在/etc/inittab文件中对本数据项进行定义的独一无二的字符串
runlevel	必须调用此进程的运行级别。有的活动是如此的特殊，以至于在全部运行级别中都可以被捕获（比如用于重启动的 Ctrl+Alt+Del组合键）。如果想表示某个活动在全部运行级别中都需要处理，将 runlevel项空着就可以了。如果想定义在某几个运行级别中发生一些事情，只需要把这些运行级别全部列在该数据域中就可以了。举例来说，如果 runlevel数据域的值是123，就表示在运行级别1、2、和3中需要进行操作
action	定义需要进行的操作。这个数据域的参数马上会在下面说明
process	定义进入某个运行级别后需要执行的进程（程序）

表9-2 定义在/etc/inittab文件中action数据域中的参数

/etc/inittab文件里action数据域中的参数	说 明
respawn	被终止时立刻需要重新启动的进程
wait	一进入某个运行级别就必须运行的进程， init会等待它运行结束
once	一进入某个运行级别就必须运行的进程， init不需要等待它运行结束就可以执行需要在此运行级别中运行的其他程序
boot	需要在系统引导时运行的进程。此时运行级别域中的数据将不起作用
bootwait	需要在系统引导时运行的进程， init需要等待它运行结束之后才能继续到下一个需要运行的进程
ondemand	当某个特定运行级别请求出现时需要运行的进程（这些运行级别是 a、b和c），运行级别不发生变化
initdefault	定义init启动时的缺省运行级别。如果没有定义缺省值，就会在控制台上提示用户输入一个运行级别
sysinit	系统引导过程中，需要在任何其他 boot或者bootwait数据项之前运行的进程
powerwait	如果init从另外一个进程收到电源出现问题的信号，则将运行这个进程。 init会等到这个进程结束之后再继续向下执行
powerfail	相当于powerwait，但init不必等到这个进程结束之后才继续向下执行
powerokwait	如果init收到与powerwait类型相同的信号，并在 etc/powerstatus文件中保存有字符串“ OK ”，就将运行这个进程。 init会等到这个进程结束之后才继续向下执行
ctrlaltdel	如果init收到一个信号，表明用户按下了 CTRL-ALT-DEL组合键，就将运行这个进程。需要注意的是大多数 X-Windows服务器会捕获这个组合键，因此如果激活了X-Windows，init就不会收到这个信号

9.1.2 telinit命令

现在是谜底揭晓的时候了：通知 init在什么时候切换系统运行级别的神秘力量实际上就是

telinit命令。这个命令有两个命令行参数：一个参数用来通知 init准备切换过去的运行级别；另外一个是-t sec，其中的sec是在通知init之前需要等待的以秒计算的时间。

注意 init是否真的切换运行级别是由它自己决定的。很明显，它经常切换，否则这个命令就不会那么有用了。

注意 在大多数UNIX操作系统的具体实现（包括Linux）中，telinit命令实际上只是一个对init程序的符号链接。基于此，许多人更喜欢使用 init直接切换到他们想去的运行级别而不是使用 telinit。就个人而言，我发现使用 telinit切换运行级别更便于理解和记忆。

9.2 inetd进程

inetd程序是一个守护进程。读者可能已经知道守护进程是一些特殊的程序：它们在被启动之后，自愿放弃对调用自己终端的控制权。守护进程与系统其余部分的接口只有依靠进程间通信（interprocess communication，IPC）通道、或者依靠向系统全局性日志文件（log file）才能发送数据项（我们将在本章的后面讨论另外一个守护进程 syslogd）。

inetd的角色是作为telnet和ftp等与网络服务器相关的进程的“超级服务器”。这是一个简单的道理：并不是全部的服务器进程（包括那些接受新的 telnet和ftp连接的进程）都会如此频繁地被调用，以至于必须要有一个程序随时运行在内存中。因此为了避免出现可能有几十种服务都运行在内存中准备被使用的情况，它们都列在 inetd的配置文件/etc/inetd.conf中。而代替它们的是inetd监听着进入的连接。这样只需要有一个进程在内存中就可以了。

inetd的另外一个优点是程序员并不想把需要网络连接的进程都编写到系统中去。 inetd程序将处理网络代码，并把进入的网络数据流作为各个进程的标准输入（ standard-in，即stdin）传递到其中。这些进程的输出（stdout）将会被送回连接到该进程的主机去。

注意 除非你正在进行编程，否则是不需要连接到 inetd的stdin/stdout功能上。从另一方面来说，如果有人打算编写一个简单的命令脚本程序并让它出现在网络中，就值得深入研究这个极为强大的功能。

9.2.1 etc/inetd.conf文件

etc/inetd.conf文件是inetd的配置文件。它的结构很简单：每一行语句代表一种服务。服务定义语句的格式如下所示：

```
srvce_name  sock_type  protocol  [no]wait  user  srvr_prog  srvr_prog_args
```

表9-3给出了服务定义语句格式中每一个元素的说明。

表9-3 etc/inetd.conf文件中服务定义语句格式中的元素

名 称	说 明
srvce_name	所提供服务的名称。这个名称是与 /etc/services文件中的定义相关联的，该文件把服务的名称与提供该服务的端口联系在一起。如果打算在 etc/inetd.conf文件中添加新的内容，那么还必须在 /etc/services文件中添加一个相应的数据项（不用有什么顾虑，因为/etc/services文件的格式是不连续的）

（续）

名 称	说 明
sock-type	套接字类型可以是stream或者dgram。stream值代表面向连接的（即TCP）数据流（比如：telnet和ftp）。dgram值代表数据报（datagram，即UCP）流，比如：tftp服务就是一个基于数据报的协议）。不同于TCP/IP范围以外的协议确实是存在的；但是，很少会遇到它们
protocol	这是一个在/etc/protocols文件中定义的值——典型值是tcp、udp，或者是定义在RPC基础上的TCP服务的rpc/tcp，以及定义在RPC基础上的UDP服务的rpc/udp
[no]wait	可以是wait或者nowait。对任何流（TCP）式连接，必须使用nowait。对其他连接来说，这个值取决于进程支持的数据报连接的类型。 wait代表的意思是：客户将连接到服务器，在断开连接之前必须等待一个回答（这通常被称为“单线”服务器，因为它一次只能处理一个请求）。nowait代表的意思是：客户在发送完自己的数据之后就立刻断开连接
user	进程将作为本数据项中定义的服务运行。使用这个数据项的时候要十分谨慎——一个不正确的选择有可能导致安全方面的漏洞。一般来说，选择服务的时候要尽量选那些能够以非根用户的身份来运行的。如果你必须运行一个只能以根用户身份启动的服务，一定要保证程序有良好的设计，并且要阅读全部的文档以保证已经对它进行了正确的配置
srvr_prog	这是连接到本语句定义的服务时将要执行的程序的完整路径
srvr_prog_args	程序的名称，加上该程序的参数。举例来说，我们假设srvr_prog参数定义（见上面的说明）的完整路径是/usr/bin/in.fingerd，并且准备把-l参数传递给它。那么，srvr_prog参数就将是/usr/bin/in.fingerd，而srvr_prog_args就是in.fingerd -l

下面是/etc/inetd.conf文件中的一个数据项示例：

```
#
# These are standard services.
#
ftp     stream  tcp     nowait  root    /usr/sbin/tcpd  in.ftpd -l -a
telnet  stream  tcp     nowait  root    /usr/sbin/tcpd  in.telnetd
```

在上面的例子中，我们看到有两个数据项：ftp和telnet。它们两个都是流式TCP服务，在被调用的时候将立刻作为根用户运行。需要注意的是我们并没有直接调用这两个服务，它们是通过被称为“TCP打包器”的/usr/sbin/tcpd程序调用的。TCP打包器程序将接受对该程序的连接，记录连接请求，并检查/etc/hosts.allow和/etc/hosts.deny两个文件，确定客户是否被允许连接到这些服务上。如果一切检查合格，该连接的控制权将传递到相应的守护进程中。

9.2.2 安全性与inetd.conf文件

你将会发现在大多数的Linux安装中，许多服务在缺省的情况下是打开的。如果你的系统将向因特网开放（包括通过拨号点对点协议（PPP）被连通），你想做的第一件事情就会是把一切都关闭！决不要假设因为你的系统没有对公众进行宣传，别人就不会找到它。从相反的方向看——扫描大型网络，寻找存在安全性攻击隐患的系统的工具软件却是既容易找到又容易使用的。

关闭服务的第一个步骤是把/etc/inetd.conf文件里所有用不着的服务性说明语句都改为注

释语句。例如，你是否有必要提供一个 gopher服务器？你会用到它吗？你的用户会用到它吗？如果都不会，把它改为注释语句。

注意 一般来说，你会发现下面的方法更容易使用：先把全部东西都改为注释语句（彻底关闭网络服务），当有机会仔细考虑过这个方法之后再有选择地打开需要的服务。

在完成对/etc/inetd.conf文件的修改之后，需要向守护进程报告其配置文件已经被修改了。这是通过向该守护进程发送 HUP信号来实现的。先使用下面的命令找出 inetd.conf对应的进程ID：

```
[ root@ford /root ] # ps auxw | grep inetd | grep -v grep
```

这个命令的输出类似于下面的内容：

```
root     359  0.0  0.1  1232  168  ?     S    Jun21    0 : 00 inetd
```

输出中的第二列告诉我们进程 ID号（这里就是359）。为了发送HUP信号，我们需要使用kill命令（把这个程序叫做 kill多少有些误导。实际上，它只是向进程发送信号而已。缺省的情况下，它会发出请求某个程序终止运行的信号）。

下面是使用kill命令发送HUP信号的方法：

```
kill -1 359
```

你应该把上面命令中的359换成从你的系统上得到的进程编号。

9.3 syslogd守护进程

在同一时间会发生许许多多的事情，而在终端窗口中断开连接的网络服务就更是如此了。因此，提供一个记录特殊事件和消息的标准机制就非常有必要了。 Linux使用syslogd守护进程来提供这个服务。

syslogd守护进程提供了一个对系统活动和消息进行记录的标准方法。许多其他种类的UNIX操作系统也使用了兼容的守护进程，这就提供了一个在网络中跨平台记录的方法。在大型的网络环境里这更具有价值。因为在那样的环境里，集中收集各种记录数据以获得系统运转的准确情况是很有必要的。你可以把这种记录功能子系统比作 Windows NT的System Logger。

syslogd保存数据用的记录文件都是简明的文本文件，一般都存放在 /var/log子目录中。每个数据项构成一行，包括日期、时间、主机名、进程名、进程的 PID、以及来自该进程的消息。标准C函数库中的一个全局性的函数提供了生成记录消息的简单机制。如果你不喜欢编写程序代码，但是又想在记录文件中生成数据项，可以选择使用 logger命令。

可以想像，像syslogd这样重要的工具应该是作为开机引导命令脚本程序的一部分来启动的。你准备在服务器环境中使用的任何一个 Linux发行版本都已经为你设置好了。

9.3.1 调用syslogd

如果需要手动启动 syslogd，或者需要修改开机引导时启动它的命令脚本程序，你就必须注意syslogd的命令行参数，如表9-4所示。

表9-4 syslogd的命令行参数

参 数	说 明
-d	调试状态。在启动的时候，syslogd一般会交出当前终端的控制权，让自己在后台运行。使用了-d参数之后，syslogd将保留对终端的控制权，并在记录消息的同时显示调试信息。一般几乎没有机会用到这个参数
-f *config*	定义另外一个配置文件代替缺省的/etc/syslog.conf文件
-h	在缺省的情况下，syslogd不会把发送给它但实际目的地是另外一个主机的消息转发出去。如果使用了这个参数，就有可能被用来作为实施拒绝服务攻击的一个部分
-l hostlist	这个参数列出愿意记录其活动的主机名单。每个主机使用的是它的简单名称，不是它的完全授权域名（fully qualified domain name，FQDN）。允许列出多个主机，彼此用冒号隔开，如下所示： `-l toybox : ford : oid`
-m interval	缺省情况下，syslogd每隔20分钟生成一个记录数据项，以此作为“让你知道我正在运行”的消息。这只适用于并不繁忙的系统（如果你正在观察系统的记录，要是过了20分钟还没有看到一条消息，则说明有什么地方出现了问题）。如果定义了一个数值作为时间间隔，就可以指定syslogd需要每隔多少分钟生成一条消息
-r	缺省情况下，作为安全预警措施，syslogd守护进程会拒绝从网络上发送给它的消息。这个参数将激活本功能
-s domainlist	如果你正在接收的syslogd数据项给出了FQDN，可以让syslogd滤除域名，只保留主机名。只需要在一个以冒号隔开的清单中简单地列出域名作为 -s选项的参数就可以了。如下所示： `-s x-files.com : conspiracy.com : wealthy.com`

9.3.2 /etc/syslog.conf文件

/etc/syslog.conf文件包含了syslogd需要运行的配置信息。这个文件的格式有些不寻常，但是现有的缺省配置文件将足以满足使用需要了，除非你需要在特定的文件中查找特定的信息，或者需要把这些信息发送到远程记录计算机去。

1. 记录信息分类

在我们掌握/etc/syslog.conf文件格式本身之前，需要先了解记录消息是如何分类的。每个消息都有一个功能值（facility）和一个优先权值（priority）。功能值告诉我们这条消息是由哪个子系统产生的，而优先权值则告诉我们这个消息有多重要。这两个值由句号分隔。

这两个值都有等价的字符串，从而容易记忆。功能值和优先权值的等价字符串分别列在表9-5和表9-6中。

表9-5 /etc/syslog.conf文件中功能值的等价字符串

功能值等价字符串	说 明
auth	身份验证消息
authpriv	基本上与auth相同
cron	由cron子系统（参见后面的章节）产生的消息
daemon	各种服务守护进程的基本信息
kern	内核消息
lpr	打印子系统消息
mail	电子邮件子系统消息（包括对每一个电子邮件的记录）

（续）

功能值等价字符串	说　明
mark	已弃用，但是有的书里还介绍它；syslogd只是简单地忽略它
news	通过NNTP子系统的消息
security	与auth相同的东西（不应再使用）
syslog	来自syslog的内部消息
user	来自用户程序的消息
uucp	来自UUCP（UNIX to UNIX CoPy的缩写，意思是UNIX到UNIX的复制）的消息
local0 - local9	基本功能级别，它们的重要性可以根据需要来决定

表9-6　/etc/syslog.conf文件中优先权值的等价字符串

优先权级别等价字符串	说　明
debug	调试语句
info	杂项信息
notice	重要语句，但并不一定是坏消息
warning	潜在危险情况
warn	与warning相同（不应再使用）
err	出现一个错误
error	与err相同（不应再使用）
crit	严重事件
alert	指明发生重大事件的消息
emerg	紧急事件

注意　优先权级别是根据syslogd的严重性程度排列的。因此可以根本不考虑 debug级别消息的严重性，而 emerg是最严重的。举例来说，功能和优先权组合字符串mail.crit就表示在电子邮件子系统中出现一个严重的错误（比如它用尽了硬盘空间）。syslogd把这条消息看得比 mail.info消息重要得多，后一条消息可能只是说明又收到了一个新的电子邮件。

syslogd还知道通配符。因此，你可以定义整个一个级别的消息；举例来说， mail.*代表与电子邮件子系统有关的全部消息。

2. /etc/syslog.conf文件的格式

下面是配置文件里各语句的格式：

```
facility/priority  combinations separated by commas file/process/host to log to
```

举例如下：

```
kern.info     /ver/log/kerned
```

syslogd还可以灵活地把记录消息发送到多种不同的保存目的地去。它可以把消息保存为文件、把消息发送到FIFO队列、发送到一组用户、或者（在大型站点集中记录消息的情况下）发送到一个中心记录主机中。为了区分这些目的地，在目的地入口使用了下面的规则：

- 如果保存目的地的开始字符是斜杠字符（/），消息将发送到某个文件。
- 如果保存目的地的开始字符是垂直字符（|），消息将发送到某个FIFO队列。
- 如果保存目的地的开始字符是“ @ ”字符，消息将发送到某个主机。

表9-7给出了遵守上述规则的几个目的地示例。

表9-7 保存目的地数据项示例

目的地格式	说 明
/var/log/logfile	一个文件 注意，如果你在文件名前面加上短划线字符（-），则syslogd在完成了写操作之后不会同步文件系统。这也就是说如果使用了短划线字符（-），在系统有机会清空其缓冲区之前有可能会丢失一些数据。但是从另外一方面来说，如果某个应用程序需要记录的消息太多太噜嗦，使用这个参数可以改进性能 记住：如果用户打算把记录消息发送到控制台上，就必须指定 /dev/console设备
| /tmp/mypipe	一个管道。这种类型的文件是使用 mknode命令（参见第6章）建立的。当 syslogd在管道的一端读入数据的时候，可以运行另外一个程序从管道的另一端读取数据。这是使用程序分析记录消息输出、查找严重问题情况的有效方法，这样在必要的时候可以及时收到报警
@loghost	主机名。这个例子将把消息发送到 loghost主机。loghost主机上的syslogd守护进程将记录那些消息

如果在保存目的地的开始没有输入特殊字符， syslogd就将认为发送目的地是一个使用逗号分隔的用户清单，消息将发送到他们的显示器屏幕上。

如果使用了星号字符（*），syslogd就将把消息发送给全部已经登录上机的用户。

与其他情况相同，任何以井字号（#）开始的语句都是注释行。

现在，我们来看看配置文件数据项的几个示例：

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

在上面的例子中，我们的语句将把各种优先权级别的电子邮件功能消息都发送到/var/log/mailog文件中去。

再看看其他例子：

```
# Everybody gets emergency messages , plus log them on another
# machine .
*.emerg                                                    @loghost, sshah, hdc,
root
```

在上面的例子中，我们可以看到记录优先权级别为 emerg的全部功能消息都将被发送到另外一个运行着syslogd名为loghost的系统中去。同时，如果用户 hdc、sshah或者root已经登录上机的话，需要记录的消息也将出现在这些用户的显示器屏幕上。

在同一行上你可以对单个活动定义多个选择开关，如下所示：

```
*.info; mail.none; authpriv.none                                    /var/log/messages
```

3. /etc/syslog.conf文件示例

下面是一个完整的syslog.conf文件：

```
# 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                          /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
# Everybody gets emergency messages, plus log them on another
# machine.
*.emerg                                                 *

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

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

9.4 cron程序

cron程序允许系统中的任一用户安排某个程序在任何日期、时间、或者星期几准时运行，时间可以精确到分钟。使用 cron是一个极为有效的方法，可以使你的系统自动化、周期性地生成报告、以及执行其他定期任务（不太诚实的 cron使用方法还包括在你参加会议想逃会的时候调用某个程序给你打个传呼！）

和我们在本章中已经讨论的其他服务一样， cron是由开机引导命令脚本程序启动的，并且应该是已经配置好了的。对进程清单的一个快速检查就可以看到它默默地在后台运行着。

```
[root@ford /root]# ps auxw | grep cron | grep -v grep
root       341  0.0  0.0  1284  112 ?        S    Jun21   0:00 crond
[root@ford /root]#
```

cron服务的工作原理是：每隔一分钟唤醒一次，检查每个用户的 crontab文件。这个文件的内容是该用户希望在某些特定时刻执行的活动的清单。任何匹配当前日期和时间的活动都将被调入执行。

cron命令本身并不需要任何命令行参数，也不需要什么信号来表明其状态的改变。

用来编辑由cron调入执行的设置项工具是crontab。它的原理是：验证你是否有修改 cron设置项的权限，然后调入一个文本编辑器让你进行修改。修改完成后， crontab会把文件放到正确的位置，并把你带回到提示符下。

你是否具有适当的权限是由 crontab通过检查/etc/cron.allow和/etc/cron.deny这两个文件来决定的。不管这两个文件中的哪一个存在，只有当你明确地列在其中的时候才能使你的操作生效。举例来说，如果 /etc/cron.allow文件存在，那么只有当你的用户名列在这个文件中的时候才能允许你编辑 cron数据项。另一方面，如果 /etc/cron.deny文件存在而/etc/cron.allow文件不存在，那么只要你的用户名没有出现在这个文件中，就表明允许你编辑你的 cron数据项。

列有你cronjobs的文件（通常即指crontab文件）的格式如下所示，所有数值都必须是整数。

```
Minute  Hour  Day  Month  DayOfWeek  Command
```

如果你打算在某一栏上设置多个数值（比如：你打算在早晨4：00、中午12：00和下午5：00三次运行同一个程序），就需要将这些时间数据用逗号在相应的栏里分隔开，但是在该栏中不能添加任何空格。对需要在早晨4：00、中午12：00和下午5：00三次运行同一个程序的情况，Hour栏的数据项应该是4、12、17。请注意cron使用的是军事时间写法。

在DayOfWeek数据项里，数值0表示星期天、1表示星期一、以此类推，直到6表示星期六。

如果某个数据项中出现了一个星号（*）通配符，就表示相应栏中任意的分钟、小时、日期、月份或者星期几都匹配要求。当文件中的日期和时间满足当前日期和时间的时候，这一行语句中设置的命令就会以设置该数据项用户的身份运行。所生成的任何输出都会以电子邮件E-mail的方式返回到那个用户。很明显，这可能会引起邮箱中的消息爆满，所以对邮件报告作出迅速反应是很重要的。对消息量加以控制的一个好办法是：只输出出错情况，而那些不可避免的其他输出就发送到/dev/null设备去。

我们来看一些例子。下面的数据项每隔四个小时运行一次/usr/bin/ping zaphod命令：

```
0  0,4,8,12,16,20  *  *  *  /usr/bin/ping zaphod
```

下面的数据项在每个星期五晚上的10：00运行程序/usr/local/scripts/backup-level-0：

```
0  22  *  *  5  /usr/local/scripts/backup_level_0
```

最后，下面的数据项在四月一日（不管那是星期几）早晨4：01分发出一封电子邮件：

```
1  4  1  4  *  /bin/mail  dad@domain.com < /home/sshah/joke
```

9.5 小结

本章讨论了每一个Linux系统都必须具备的四种核心服务。这些服务并不要求网络支持，并且会随着主机的不同而有不同的变化，这就使它们具有很高的使用价值，因为它们不管计算机系统本身是否在多用户模式下都可以工作得很好。

下面是对本章的快速回顾：init是系统中所有进程的父进程，它的PID永远是1。他还控制着运行级别，并且可以通过/etc/iniitab文件进行配置。

inetd可以说是一个超级服务器，它代替其他大量小型或者不经常使用的服务监听着服务器请求。当它收到对那些服务中的某一个请求时， inetd会调用那个实际需要的服务并在网络与实际服务之间默默地转发数据。它的配置文件是/etc/inetd.conf文件。

syslogd是系统全局性的日志记录守护进程。除了自己系统所产生的记录数据项以外，syslogd还可以记录来自网络上的消息（当然需要你先激活这项功能）。它的配置文件是/etc/syslog.conf文件。

最后介绍的cron服务允许用户安排活动在某个特定的日期和时间发生，这对类似于系统备份或者电子邮件备忘录之类定期出现的活动来说正好可以大显身手。它所依赖的各种配置文件都可以通过crontab程序来处理。

在本章的每一个小节里，我们都讨论了如何配置出加以变化的服务，甚至还在系统本身缺省的设置值以外建议了一些其他的用法。我强烈建议你仔细研究这些服务，使自己熟悉它们都可以用来完成哪些工作。我本人就编写过几个需要依靠 cron和syslog使用的数据收集和分析工具软件，当然还有一些相当可笑和没用的东西。一句话，别怕和它们打交道！