

第 22 章 服务器基础知识

在正式讨论各种服务器的配置之前，首先了解一些和服务器有关的基础知识。本章主要讨论两个基本的守护进程 `init` 和 `inetd/xinetd`（严格来说，前者要比后者“基本”得多）。相对而言，本章的理论知识偏多，缺少相关经验的读者理解起来或许会有困难。作为建议，读者也可以选择跳过本章，首先实践几个服务器的配置，再回过头来补这些“基础知识”。

22.1 系统引导

计算机的启动和关闭并不是表面上那么简单。从打开电源到操作系统准备就绪，普通用户并不知道计算机已经完成了一项多么巨大的工程。系统引导是一整套复杂的任务流程，系统管理员没有必要知道其中的每一个细节，但大致了解一些是有帮助的。

22.1.1 Linux 启动的基本步骤

要完整讲述 Linux 的启动过程，需要追溯到按下电源开关的那一刻。PC 引导的第一步是执行存储在 ROM（只读存储器）中代码，这种引导代码通常被称为 BIOS（基本输入输出系统，Basic Input/Output System）。BIOS 知道和引导有关的硬件设备的信息，包括磁盘、键盘、串行口、并行口等，并根据设置选择从哪一个设备引导。

确定引导设备后（通常是第一块硬盘），计算机就尝试加载该设备开头 512 个字节的信息，包含这 512 个字节的段被称作 MBR（Master Boot Record，主引导记录）。MBR 的主要任务是告诉计算机从什么地方加载下一个引导程序，“下一个”引导程序被称为“引导加载器（Boot Loader）”。引导加载器负责加载操作系统的内核，Grub 和 LILO 就是 Linux 上最著名的两个引导加载器。

接下来发生的事情就随操作系统的不同而不同了。对于 Linux 而言，基本的引导步骤包括以下几个阶段。

- （1）加载并初始化 Linux 内核。
- （2）配置硬件设备。
- （3）内核创建自发进程。
- （4）由用户决定是否进入手工引导模式。
- （5）（由 `init` 进程）执行系统启动脚本。
- （6）进入多用户模式。

可见，Linux 内核总是第一个被加载的东西。内核执行包括硬件检测在内的一切基础操作，然后创建几个进程。这些内核级别的进程被称做“自发”进程。本章（或许也是整

个系统) 最重要的 `init` 进程就是在这个阶段创建的。

事情到这里还没有完。内核创建的进程只能执行最基本的硬件操作和调度, 而那些执行用户级操作的进程(诸如接受登录)还没有创建。这些任务最后都被内核“下放”给 `init` 进程来完成, 因此, `init` 进程是系统上除了几个内核自发进程之外所有进程的祖先。

22.1.2 `init` 和运行级

`init` 定义了一些被称做“运行级”的东西, 这里的“级”是级别的意思, 用一些整数表示。进入某一个运行级意味着使用某种特定的系统资源组合。“系统资源”是一个很宽泛的概念, 由于几乎所有的进程都是由 `init` 创建的, 因此理论上可以完全控制在某个运行级下应该运行哪些进程。从某种意义上, `init` 的运行级有点快餐店里“套餐”的味道, 顾客可以说“来一份 1 号套餐”, 于是服务员就端上汉堡、薯条和可乐。

Linux 的 `init` 进程总共支持 10 个运行级, 但实际定义的运行级只有 7 个。表 22.1 显示了这些运行级及其对应的系统状态。

表 22.1 运行级及其对应的系统状态

运 行 级	系 统 状 态
0	系统关闭
1 或 S	单用户模式
2	功能受限的多用户模式
3	完整的多用户模式
4	一般不用, 留作用户自己定义
5	多用户模式, 运行 X 窗口系统
6	重新启动

目前绝大部分的 Linux 发行版本默认都启动计算机至运行级 5, 也就是带有 X 窗口系统的多用户模式。服务器通常不需要运行 X, 因此常常被设置进入运行级 3。运行级 4 被保留, 方便管理员根据实际情况定义特殊的系统状态。

单用户模式是关于系统救援的。在这个运行级下, 所有的多用户进程都被关闭, 系统保留最小软件组合。引导系统进入单用户模式后, 系统会要求用户以 `root` 身份登录到系统中。在 2.4 节提到的“救援模式”就是典型的单用户模式。

0 和 6 是两个比较特殊的运行级, 系统实际上并不能停留在这两个运行级中。进入这两个运行级别意味着关机和重启。使用 `telinit` 命令可以强制系统进入某个运行级。运行下面这条命令后, 系统就进入运行级 6, 也就是关闭计算机, 然后再启动。

```
sudo telinit 6
```

尽管表 22.1 明确地列出了所有 7 个运行级代表的系统状态, 但事实上这只代表了大部分系统的习惯做法。在某一台特定的计算机上, 管理员可能会根据实际情况调整配置: 例如让运行级 3 也能启动 X 窗口系统。`init` 的配置文件是 `/etc/inittab`, 这个文件中定义了每个运行级上需要做的事情。下面是 `opensuse Linux` 中 `inittab` 文件的一部分。

```
# runlevel 0 is System halt (Do not use this for initdefault!)
# runlevel 1 is Single user mode
```

```
# runlevel 2 is Local multiuser without remote network (e.g. NFS)
# runlevel 3 is Full multiuser with network
# runlevel 4 is Not used
# runlevel 5 is Full multiuser with network and xdm
# runlevel 6 is System reboot (Do not use this for initdefault!)
#
10:0:wait:/etc/init.d/rc 0
11:1:wait:/etc/init.d/rc 1
12:2:wait:/etc/init.d/rc 2
13:3:wait:/etc/init.d/rc 3
#14:4:wait:/etc/init.d/rc 4
15:5:wait:/etc/init.d/rc 5
16:6:wait:/etc/init.d/rc 6
```

以“#”开头的行是注释行，紧跟在后面的这些行定义了在每个运行级下应该做的事情。`inittab` 文件通常并不会一一列出所有应该执行的脚本，而是调用 `rc` 脚本（通常是 `/etc/init.d/rc`）改变运行级。`rc` 脚本随后根据传给它的参数查找与运行级有关的目录，并执行其中的脚本。

这些“与运行级有关”的目录总是以 `rclevel.d` 的形式出现，其中 *level* 就是运行级编号。例如所有要在运行级 1 下执行的脚本都保存在 `rc1.d` 目录下，而为了进入运行级 3，那么就执行位于 `rc3.d` 目录下的脚本。通常，这些目录不是在 `/etc` 目录下，就是在 `/etc/init.d` 目录下。

```
$ ls -d /etc/rc*                                ##列出/etc目录下所有以rc开头的目录
/etc/rc0.d  /etc/rc1.d  /etc/rc2.d  /etc/rc3.d  /etc/rc4.d  /etc/rc5.d
/etc/rc6.d  /etc/rcS.d
```

很显然，为了改变某个运行级所使用的系统资源组合，可以在这些目录下添加/删除相应的脚本。`rclevel.d` 目录下的脚本文件有自己一套独特的命名和实现方法，将在 22.1.3 节讨论。

很容易改变 Linux 的默认运行级。在 `/etc/inittab` 文件中找到下面这一行：

```
id:5:initdefault:
```

这一行设置将 Linux 默认启动到运行级 5。如果要让 Linux 默认启动到运行级 3，可以把它改成下面这样：

```
id:3:initdefault:
```

22.1.3 服务器启动脚本

用于启动服务器应用程序（更确切地说是服务器守护进程）的脚本全部位于 `/etc/init.d` 目录下，每个脚本控制一个特定的守护进程（这个概念将在 22.2.1 节具体介绍）。所有的脚本都应该认识 `start` 和 `stop` 参数，分别表示启动和停止服务器守护进程。下面这条命令启动了 SSH 服务器的守护进程。

```
$ sudo /etc/init.d/sshd start
Starting SSH daemon                                     done
```

与此相对的，下面这条命令停止 SSH 服务器的守护进程。

```
$ sudo /etc/init.d/sshd stop
Shutting down SSH daemon                               done
```

大部分启动脚本还认识 `restart` 参数。顾名思义，接收到这个参数的脚本首先关闭服务器守护进程，然后再启动它。

```
$ sudo /etc/init.d/sshd restart
Shutting down SSH daemon           done
Starting SSH daemon                done
```

在改变运行级（包括系统启动和关闭）的时候，系统执行的是 `rclevel.d` 目录下的脚本文件。仍然以 `SSH` 为例，使用 `ls -l` 命令可以清楚地看到 `init.d` 和 `rclevel.d` 这两个目录下脚本文件之间的关系。

```
$ ls -l /etc/init.d/rc5.d/ | grep sshd
lrwxrwxrwx 1 root root 7 11-09 17:55 K12sshd -> ../sshd
lrwxrwxrwx 1 root root 7 11-09 17:55 S10sshd -> ../sshd
```

`/etc/init.d/rc5.d` 目录下的两个脚本文件 `K12sshd` 和 `S10sshd`，实际上都是指向 `/etc/init.d/sshd` 的符号链接。`init` 在执行脚本的时候，会给以字母 `S` 开头的脚本文件传递 `start` 参数，而给以字母 `K` 开头的脚本文件传递 `stop` 参数。例如 `init` 运行 `K12sshd` 时，实际执行的是下面这条命令。

```
/etc/init.d/rc5.d/K12sshd stop
```

由于 `K12sshd` 脚本是 `/etc/init.d/sshd` 的符号链接，因此又等价于下面这条命令。

```
/etc/init.d/sshd stop
```

脚本文件名中的数字描述了脚本运行的先后顺序，数字较小的脚本首先被执行。下面的例子反映了这一点。当进入运行级 5 的时候，`S05network` 在 `S10sshd` 之前执行（因为 $5 < 10$ ）；类似地，当退出运行级 5 的时候，`K12sshd` 在 `K17network` 之前执行（因为 $12 < 17$ ）。

```
$ ls -l /etc/init.d/rc5.d/ | egrep 'ssh|network'
lrwxrwxrwx 1 root root 7 11-09 17:55 K12sshd -> ../sshd
lrwxrwxrwx 1 root root 10 11-09 17:50 K17network -> ../network
lrwxrwxrwx 1 root root 10 11-09 17:50 S05network -> ../network
lrwxrwxrwx 1 root root 7 11-09 17:55 S10sshd -> ../sshd
```

这样安排的用意很明显，供远程登录使用的 `SSH` 服务器不应该在网络接口启动之前运行。在向 `rclevel.d` 目录下手动添加脚本的时候应该格外注意这些依赖关系。下面列出了在笔者的 `openSUSE` 系统上启动服务器脚本的顺序。

```
$ ls -l /etc/init.d/rc5.d/
...
lrwxrwxrwx 1 root root 8 11-09 17:50 S01acpid -> ../acpid
lrwxrwxrwx 1 root root 7 11-09 17:50 S01dbus -> ../dbus
lrwxrwxrwx 1 root root 14 11-09 17:50 S01earlysyslog -> ../earlysyslog
lrwxrwxrwx 1 root root 8 11-09 17:50 S01fbset -> ../fbset
lrwxrwxrwx 1 root root 16 11-09 17:50 S01microcode.ctl -> ../microcode.ctl
lrwxrwxrwx 1 root root 9 11-09 17:50 S01random -> ../random
lrwxrwxrwx 1 root root 9 11-09 17:50 S01resmgr -> ../resmgr
lrwxrwxrwx 1 root root 21 11-09 18:10 S01SuSEfirewall2_init -> ../SuSEfirewall2_init
lrwxrwxrwx 1 root root 13 11-09 17:50 S02consolekit -> ../consolekit
lrwxrwxrwx 1 root root 12 11-09 17:50 S03haldaemon -> ../haldaemon
lrwxrwxrwx 1 root root 11 11-09 17:50 S04earlyxdm -> ../earlyxdm
lrwxrwxrwx 1 root root 10 11-09 17:50 S05network -> ../network
```

```

lrwxrwxrwx 1 root root 9 11-09 17:50 S06syslog -> ../syslog
lrwxrwxrwx 1 root root 9 11-09 17:55 S07auditd -> ../auditd
lrwxrwxrwx 1 root root 10 11-09 17:55 S07portmap -> ../portmap
lrwxrwxrwx 1 root root 8 11-27 13:06 S07smbfs -> ../smbfs
lrwxrwxrwx 1 root root 15 11-09 17:55 S07splash_early -> ../splash_early
lrwxrwxrwx 1 root root 12 11-09 17:55 S10alsasound -> ../alsasound
lrwxrwxrwx 1 root root 15 11-09 17:55 S10avahi-daemon -> ../avahi-daemon
lrwxrwxrwx 1 root root 7 11-09 17:55 S10cups -> ../cups
lrwxrwxrwx 1 root root 19 11-09 17:55 S10java.binfmt_misc -> ../java.
binfmt_misc
lrwxrwxrwx 1 root root 6 11-09 17:55 S10kbd -> ../kbd
lrwxrwxrwx 1 root root 7 11-09 17:55 S10nscd -> ../nscd
lrwxrwxrwx 1 root root 13 11-09 17:56 S10powersaved -> ../powersaved
lrwxrwxrwx 1 root root 9 11-09 17:55 S10splash -> ../splash
lrwxrwxrwx 1 root root 7 11-09 17:55 S10sshd -> ../sshd
lrwxrwxrwx 1 root root 15 11-09 17:56 S10vmware-guest -> ../vmware-guest
lrwxrwxrwx 1 root root 17 11-09 17:55 S11avahi-dnscfnd -> ../avahi-
dnscfnd
lrwxrwxrwx 1 root root 12 12-21 05:25 S11nfsserver -> ../nfsserver
lrwxrwxrwx 1 root root 10 11-09 17:55 S11postfix -> ../postfix
lrwxrwxrwx 1 root root 6 11-09 17:55 S11xdm -> ../xdm
lrwxrwxrwx 1 root root 7 11-09 17:55 S12cron -> ../cron
lrwxrwxrwx 1 root root 9 11-09 17:57 S12smartd -> ../smartd
lrwxrwxrwx 1 root root 9 11-09 14:15 S12xinetd -> ../xinetd
lrwxrwxrwx 1 root root 15 11-09 17:50 S21stopblktrace -> ../stopblktrace
lrwxrwxrwx 1 root root 22 11-09 18:10 S21SuSEfirewall2_setup -> ../S-
uSEfirewall2_setup

```

22.1.4 Ubuntu 和 Debian 的 init 配置

Ubuntu 和 Debian 的启动配置有一点特殊，这两个发行版使用 `upstart` 而不是 `init` 来管理启动脚本。在默认情况下，Ubuntu 和 Debian 没有 `inittab` 文件，而是使用 `/etc/event.d/rc-default` 来确定启动的默认运行级。但奇怪的是，`rc-default` 脚本依然会试图寻找 `/etc/inittab`。如果找到了，它就按照 `inittab` 文件的配置来设置运行级；如果没有找到，它就把系统启动到运行级 2。

这又和人们的常识不太一样，为什么是运行级 2 而不是 5？Debian 的 FAQ（常见问题）回答了这个问题，如表 22.2 所示。

表 22.2 Ubuntu和Debian的运行级默认设置

运 行 级	系 统 状 态
0	关闭系统
1	单用户模式
2~5	完整的多用户模式
6	重新启动


也就是说，Ubuntu 和 Debian 默认情况下并没有区分运行级 2~5。这意味着用户必须手动定制每个运行级应该包含的启动脚本。举例来说，如果想要启动到不包含图形界面的多用户模式，应该依次执行下面这些步骤。

- （1）选择一个运行级来完成这个任务，假设是运行级 3。
- （2）新建 `/etc/inittab`，内容为 “`id:3:initdefault:`”。

(3) 把/etc/rc3.d/S30gdm (KDE 是 S30kdm) 移动到其他地方备份起来。

(4) 重新启动系统。

当然, 如果愿意使用运行级 4 或 5 来表示“不包含图形界面的多用户模式”也没有问题, 只是不太符合习惯。

 **注意:** S30gdm (S30kdm) 中字母 S 后紧跟的数字随系统实际安装的软件不同而不同。

22.2 管理守护进程

本节开始介绍和服务器管理有关的另一个 (应该是两个) 重要的进程 `inetd` 和 `xinetd`。读者将会接触一些和服务器有关的内容, 包括守护进程的概念和服务器的运行方式。最后讨论如何配置 `inetd` 和 `xinetd`, 在后面几章的服务器配置中还会举例讲解这部分的内容。

22.2.1 什么是守护进程

守护进程 (daemon) 是一类在后台运行的特殊进程, 用于执行特定的系统任务。很多守护进程在系统引导的时候启动, 并且一直运行直到系统关闭; 另一些只在需要的时候才启动, 完成任务后就自动结束。举例来说, `/etc/sbin/sshd` (注意, 不是 `/etc/init.d/sshd`) 就是 SSH 服务的守护进程, 这个进程启动后会一直运行, 在后台监听 22 号端口, 等待并响应来自客户机的 SSH 连接请求。

`init` 是系统中第一个启动、也是最重要的守护进程。`init` 会持续工作, 保证启动和登录的顺利进行, 并且适时地“杀死”那些没有响应的进程。只要系统还在运行, 就可以看到 `init` 守护进程。

```
$ ps aux | grep init                                     ##在进程列表中搜索 init 进程
root      1  0.0  0.0   4020   888 ?        Ss   13:17   0:00 /sbin/init
```

`xinetd` 和 `inetd` 是管理其他守护进程 (例如 `sshd`) 的守护进程。引入这两个守护进程的目将在 22.2.2 节中介绍。

22.2.2 服务器守护进程的运行方式

运行一个服务 (例如 SSH) 最简单的办法就是让它的守护进程在引导的时候就启动, 然后一直运行, 监听并处理来自客户机的请求。在刚开始, 这样的设置不会有什么问题。但随着服务的增多, 这些运行在后台的守护进程会大量消耗系统资源 (因为它们一直在运行!), 这种消耗常常是没有必要的。举例来说, SSH 服务一天内可能只会被一个管理员用到几次, 这样, `/etc/sbin/sshd` 每天空闲的时间甚至接近 20 个小时。

`inetd` 和 `xinetd` 就是为了解决这种矛盾而诞生的。`inetd` 最初由伯克利的专家们开发, 这个特殊的守护进程能够接管其他服务器守护进程使用的网络端口, 在监听到客户端请求后启动相应的守护进程, 并为这个服务器守护进程建立一条通往指定端口的输入/输出通道。

inetd 的意义在于，系统上不用同时运行多个“没有事做”的守护进程。像 SSH、FTP 这样平时不怎么用到的服务可以配置为使用 inetd，这样它们可以把监听端口的任务交给 inetd。当出现一条 FTP 连接时，inetd 就启动 FTP 服务的守护进程；同样，当管理员有事找 SSH 的时候，inetd 就把 sshd 叫醒。

inetd 最初在 UNIX 系统上被设计，后来被移植到了 Linux 上。现在绝大多数 Linux 已经使用了更好的 xinetd。相对于 inetd 而言，xinetd 有以下优点：

- ❑ 更多的安全特性；
- ❑ 针对拒绝服务攻击的更好的解决方案；
- ❑ 更强大的日志管理功能；
- ❑ 更灵活清晰的配置语法。

尽管如此，一些 Linux 系统仍然在使用 inetd。因此在详细讨论 xinetd 配置之后，本章还会对 inetd 做简单的介绍。

现在可以把本节的标题补充完整了。服务器守护进程的运行方式有两种：一种是随系统启动而启动，并持续在后台监听连接请求；另一种是借助于 inetd/xinetd，在需要的时候启动，完成任务后把监听任务交还给 inetd/xinetd。通常，前者被称为 standalone 模式，后者被称为 inetd/xinetd 模式。（尽管这种叫法听上去有点别扭，但既然大家都这么说，就随大流吧。）

并不是所有的服务器守护进程都支持 inetd 和 xinetd，应用程序必须在编写的时候就加入对这种模式的支持。一些服务器守护进程（例如 sshd、apache2）既支持 standalone 模式，也能支持 inetd/xinetd 模式。在接下来几章的服务器配置中会涉及这两种运行模式的选择。

inetd/xinetd 模式的确有很多优点，但事情总不能一概而论。对大型 Web 站点而言就不应该使用 inetd/xinetd 模式运行 Apache（当前最流行的 Web 服务器软件），因为这些服务器访问量巨大，每分每秒都会有新的连接请求，让 inetd/xinetd 如此频繁地启动和关闭 Apache 守护进程会非常糟糕。

对于桌面版本的 Linux 而言，inetd 和 xinetd 通常都需要手动安装。Ubuntu Linux 在其安装源中提供了 inetd 和 xinetd，而 openSUSE 只提供了 xinetd。

22.2.3 配置 xinetd

xinetd 守护进程依照/etc/xinetd.conf 的配置行事。如今的 Linux 发行版都不鼓励通过直接编辑/etc/xinetd.conf 来添加服务，相反用户应该为每个服务单独开辟一个文件，存放在/etc/xinetd.d 目录下。查看 xinetd.conf 可以看到这一点：

```
$ cat /etc/xinetd.conf                                ##查看/etc/xinetd.conf
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
    # Please note that you need a log_type line to be able to use log_on_success
    # and log_on_failure. The default is the following :
    log_type = SYSLOG daemon info
```

```
}
includedir /etc/xinetd.d
```

最后一行使用 `includedir` 命令把目录 `/etc/xinetd.d` 下的文件包含进来。这样设置的好处是，如果有很多服务需要依靠 `xinetd`，那么把它们全部写入 `xinetd.conf` 中势必会让整个结构看起来一团糟，把服务器配置分类存放有助于管理员理清头绪。

`xinetd.conf` 中的 `defaults` 配置段设置了 `xinetd` 一些参数的默认值。在上面的例子中，`log_type` 的值被设置为 `SYSLOG daemon info`，该变量的含义将在后文解释。

安装 `xinetd` 后会在 `/etc/xinetd.d` 中自动生成一些服务的配置文件。作为例子，下面显示了 `time` 服务的配置信息（在 `/etc/xinetd.d/time` 文件中配置）。

```
service time
{
    disable      = yes
    type         = INTERNAL
    id           = time-stream
    socket_type  = stream
    protocol    = tcp
    user        = root
    wait        = no
}
```

每个服务总是以关键字 `service` 开头，后面跟着服务名。对该服务的配置包含在一对花括号中，以“参数=值”的形式，每个参数占一行。表 22.3 列出了 `xinetd` 配置的常用参数。

表 22.3 xinetd配置的常用参数

参 数	取 值	含 义
<code>id</code>	有意义的字符串	该服务的唯一名称
<code>type</code>	<code>RPC/INTERNAL/UNLISTED</code>	指定特殊服务的类型。 <code>RPC</code> 用于 <code>RPC</code> 服务； <code>INTERNAL</code> 用于构建到 <code>xinetd</code> 内部的服务； <code>UNLISTED</code> 用于非标准服务
<code>disable</code>	<code>yes/no</code>	是否禁用该服务
<code>socket_type</code>	<code>stream/dgram</code>	网络套接口类型。 <code>TCP</code> 服务用 <code>stream</code> ， <code>UDP</code> 服务用 <code>dgram</code>
<code>protocol</code>	<code>tcp/udp</code>	连接使用的通信协议
<code>wait</code>	<code>yes/no</code>	<code>xinetd</code> 是否等待守护进程结束才重新接管该端口
<code>server</code>	路径	服务器二进制文件的路径
<code>server_args</code>	参数	提供给服务器二进制文件的命令行参数
<code>port</code>	端口号	该服务所在的端口
<code>user</code>	用户名	服务器进程应该由哪个用户身份运行
<code>nice</code>	数字	服务器进程的谦让度。参考 10.7 节
<code>instances</code>	数字/ <code>UNLIMITED</code>	同时启动的响应数量。 <code>UNLIMITED</code> 表示没有限制
<code>max_load</code>	数字	调整系统负载阈值。如果实际负载超过该阈值，就停止服务
<code>only_from</code>	IP 地址列表	只接受来自该地址的连接请求
<code>no_access</code>	IP 地址列表	拒绝向该 IP 地址提供服务
<code>log_on_failure</code>	列表值	连接失败时应该记录到日志中的信息
<code>log_on_success</code>	列表值	连接成功时应该记录到日志中的信息

参数 `id` 用于唯一标识服务，这意味着可以为同一个服务器守护进程配置不同的协议。上文中的 `time` 服务就拥有两个版本的 `xinetd` 配置，另一个用于 `UDP` 协议。

参数 `disable` 设置是否要禁用该服务。有些时候，管理员只是想列出将来可能会用到的服务，而不是现在就启用它。对这么多行进行注释会让人感到厌烦，将 `disable` 设置为 `yes` 就可以简单地禁用该服务。不过，管理员偶尔也会忘记在启用服务的时候把这个选项改回来，如果正在奇怪为什么某项服务没有被 `xinetd` 加载，那么应该首先检查 `disable` 选项是否已经被正确地设置为 `no` 了。

将 `wait` 参数设置为 `yes` 意味着由 `xinetd` 派生出的守护进程一旦启动就接管端口，`xinetd` 会一直等待，直到该守护进程自己退出；`wait=no` 表示 `xinetd` 会连续监视端口，每次接到一个请求就启动守护进程的一个新副本。管理员应该参考守护进程的手册，或者 `xinetd` 的配置样例来确定使用何种配置。

参数 `port` 在绝大多数情况下是不需要的。`xinetd` 根据服务名从 `/etc/service` 文件中查找信息，确定该服务使用的端口和网络协议。如果没有在 `/etc/service` 文件中登记该服务，那么也应该手动添加，而不是使用 `port` 参数——把信息集中起来管理总是能省去不少麻烦。下面截取了 `/etc/service` 文件中的一部分。

```
ftp      21/tcp
fsp      21/udp      fspd
ssh      22/tcp
ssh      22/udp      # SSH Remote Login Protocol
telnet   23/tcp
smtp     25/tcp      mail
```

`/etc/service` 中的每一行对应一个服务，从左到右依次表示：

- ❑ 服务名称。例如 `ssh`；
- ❑ 该服务使用的端口号。例如 `22`；
- ❑ 该服务使用的传输协议。例如 `tcp`；
- ❑ 别名（或者叫“绰号”？）。例如 `fspd`；
- ❑ 注释。例如 `# SSH Remote Login Protocol`。

参数 `user` 设置应该以哪个用户身份运行该服务器进程，大部分服务都使用 `root`。有些时候从安全的角度考虑会使用非特权用户（例如 `nobody`），但这只适用于那些不需要 `root` 权利的守护进程。

`xinetd` 会记录连接失败/成功时的信息，用户可以通过定制 `log_on_failure` 和 `log_on_success` 这两个参数指导 `xinetd` 记录哪些信息。表 22.4 列出了和这两个参数有关的取值。

表 22.4 和日志记录有关的取值

值	适用于	描述
HOST	二者皆可	记录远程主机的地址
USERID	二者皆可	记录远程用户的 ID
PID	<code>log_on_success</code>	记录服务器进程的 PID
EXIT	<code>log_on_success</code>	记录服务器进程的退出信息
DURATION	<code>log_on_success</code>	记录任务持续的时间
ATTEMPT	<code>log_on_failure</code>	记录连接失败的原因
RECORD	<code>log_on_failure</code>	记录连接失败的额外的信息

 **注意：**USERID 标志会向远程主机询问建立连接的用户信息，这样总会造成明显的延时，因此应该尽可能避免使用 USERID。

完成对服务配置的后，使用下面这条命令重新启动 xinetd 守护进程。

```
$ sudo /etc/init.d/xinetd restart
```

22.2.4 举例：通过 xinetd 启动 SSH 服务

作为例子，本节将带领读者配置 SSH 服务的 xinetd 实现。总的来说，在 xinetd 中添加服务无非是下面这几步：

- (1) 修改（增加）配置文件；
- (2) 停用该服务的守护进程；
- (3) 重启 xinetd 使配置生效；
- (4) 如果需要，从相应的 rc 目录中移除该服务的启动脚本。

下面就来逐一实现以上各个步骤。首先在/etc/xinetd.d 目录下建立文件 ssh，包含下面这些内容。

```
service ssh
{
    socket_type      = stream
    protocol        = tcp
    wait            = no
    user            = root
    server          = /usr/sbin/sshd
    server_args     = -i
    log_on_success  += DURATION
    disable         = no
}
```

注意 log_on_success 参数允许使用“+”这样的赋值方式，表示在原有默认值的基础上添加，而不是推倒重来。类似地，也可以使用“-”在默认值的基础上减去一些值。参数的默认值通常在/etc/xinetd.conf 中设置。

下一步停用 SSH 守护进程，为 xinetd 接管 22 端口铺平道路。

```
$ sudo /etc/init.d/ssh stop
* Stopping OpenBSD Secure Shell server sshd [ OK ]
```

重新启动 xinetd 使配置生效。

```
$ sudo /etc/init.d/xinetd restart
* Stopping internet superserver xinetd [ OK ]
* Starting internet superserver xinetd [ OK ]
```

运行 netstat -tulnp 命令查看 22 端口的情况，发现 xinetd 已经顺利接管了 SSH 通信端口。

```
$ sudo netstat -tulnp | grep 22
tcp        0      0 0.0.0.0:22        0.0.0.0:*        LISTEN
8356/xinetd
```

##查看 22 端口状态


现在尝试连接本地的 SSH 服务。对于客户端而言，看上去和 standalone 方式没有什么

不同。

```
$ ssh localhost -l lewis
lewis@localhost's password:
```

如果在安装 SSH 服务器的时候选择了随系统启动（通常这是默认配置），那么接下来还要从相应的 rc 目录中移除 SSH 服务的启动脚本，否则下次启动系统的时候 xinetd 将无法运行。假设系统默认启动到运行级 5（可以参考 22.1 节获取有关运行级的详细信息）。

```
$ cd /etc/rc5.d/                ##进入相应的 rc 目录
$ ls | grep ssh                ##查找 SSH 启动脚本
S16ssh
$ sudo mv S16ssh ../rc_bak.d/S16ssh_rc5_bak  ##移动到另一个地方备份起来
```

 **注意：**不要随便删除启动脚本，而应该把它移动到另一个地方，并且取一个有意义的名字。这样在以后需要的时候可以方便地找回来。

22.2.5 配置 inetd

与 xinetd 类似，inetd 的配置文件是 /etc/inetd.conf。在参数的个数上，inetd 要比 xinetd 少很多，因此每个服务只需要一行就足够了。下面是从 /etc/inetd.conf 中截取的一部分配置信息。

```
#echo      stream  tcp     nowait  root    internal
#echo      dgram   udp     wait    root    internal
...
ident      stream  tcp     wait    identd  /usr/sbin/identd  identd
swat       stream  tcp     nowait.400 root    /usr/sbin/swat  swat
finger     stream  tcp     nowait  nobody  /usr/sbin/tcpd  in.fingerd -w
...
```

各个字段从左至右依次表示：

- ☐ 服务名称。和 xinetd 一样，inetd 通过查询 /etc/service 获得该服务的相关信息。
- ☐ 套接口类型。TCP 用 stream，UDP 用 dgram。
- ☐ 该服务使用的通信协议。
- ☐ inetd 是否等到守护进程结束才继续接管端口。wait 表示等待（相当于 xinetd 的 wait = yes），nowait 表示不等待，inetd 每次接到一个请求就启动守护进程的新副本（相当于 xinetd 的 wait = no）。
- ☐ 运行该守护进程的用户身份。
- ☐ 守护进程二进制文件的完整路径及其命令行参数。和 xinetd 不同，inetd 要求把服务器命令作为第一个参数（例如 in.fingerd），然后才是真正意义上的“命令行参数”（例如 -w）。关键字 internal 表示服务的实现由 inetd 自己实现。

完成对 /etc/inetd.conf 的编辑后，需要给 inetd 发送一个 HUP 信号，通知其重新读取配置文件。

```
$ ps aux | grep inetd          ##查找 inetd 的进程号
root      4414  0.0   0.1   1908   428  ?        S      13:38   0:00
/usr/sbin/inetutils-inetd
```

```
lewis      5186  0.0  0.3   3216   772 pts/0    R+   13:39   0:00 grep inetd
$ sudo kill -HUP 4414                                ##发送 HUP 信号
```

22.3 小 结

- ❑ PC 启动的第一步是执行 ROM 中的引导代码 BIOS。
- ❑ BIOS 中保存有硬件设备信息，并确定从哪一个设备开始引导。
- ❑ 引导设备开头 512 个字节的段称为 MBR，指导计算机加载下一个引导程序“引导加载器”。
- ❑ 引导加载器负责加载操作系统内核。Grub 和 LILO 是 Linux 上最著名的两个引导加载器。
- ❑ init 进程是整个系统最重要的进程之一。
- ❑ 运行级是对特定系统资源组合的抽象概念。
- ❑ 通过设置/etc/inittab 可以改变系统默认的运行级。
- ❑ 服务器的启动脚本位于/etc/init.d 目录下；rclevel.d 目录下保存了为特定运行级准备的启动脚本的符号链接。
- ❑ Ubuntu 和 Debian 默认启动到运行级 3。默认情况下各运行级（2~5 级）完全相同。
- ❑ 守护进程是一类在后台运行的特殊进程。
- ❑ 服务器守护进程有两种运行方式 standalone 方式和 inetd/xinetd 方式。
- ❑ 对于运行时负载较小的服务如 FTP、SSH 等，应该考虑使用 inetd/xinetd 方式。
- ❑ xinetd 的配置文件是/etc/xinetd.conf。从便于管理的角度考虑，添加服务应该在 /etc/xinetd.c 目录下添加相应的文件。