

# Gnu/Linux 运维实战

Emacs & L<sup>A</sup>T<sub>E</sub>X

天道酬勤 编著



2019 年 10 月 16 日

# 目录

<b>第一部分 Gnu/Linux 基础知识篇</b>	<b>7</b>
<b>第一章 关于本书</b>	<b>9</b>
1.1 Gnu 计划	9
1.1.1 图片欣赏	10
1.2 本书使用的操作系统及环境	11
<b>第二章 常用命令使用</b>	<b>13</b>
2.1 命令行下的快捷键	13
2.1.1 常用快捷键介绍	13
2.2 使用 man page 获得帮助	15
2.3 echo 与终端颜色	15
2.3.1 终端颜色	16
2.4 date 命令的使用	18
2.5 yum 命令的使用	18
2.6 parted 命令的使用	20
2.7 mount 命令的使用	20
2.8 grep 命令的使用	21
2.8.1 常用选项	21
2.8.2 一些实例	21
2.9 crontab 命令的使用	22
2.10 find 命令的使用	22
2.11 top 命令的使用	24
2.12 free 命令的使用	25
2.12.1 常用选项	25
2.12.2 一些实例	25
2.13 xargs 命令的使用	25
2.14 tr 命令的使用	25
2.15 tar 命令的使用	25
2.16 read 命令的使用	27
2.16.1 常用选项	27
2.16.2 一些实例	27
2.17 cut 命令的使用	27
2.17.1 常用选项	27

2.17.2 一些实例 . . . . .	27
2.18 sort 命令的使用 . . . . .	28
2.18.1 常用选项 . . . . .	28
2.18.2 一些实例 . . . . .	28
2.19 lsof 命令的使用 . . . . .	28
2.19.1 恢复删除的文件 . . . . .	30
2.20 netstat 命令的使用 . . . . .	31
2.21 tcpdump 命令的使用 . . . . .	31
2.22 traceroute 命令的使用 . . . . .	33
2.23 wget 命令的使用 . . . . .	33
2.24 screen 命令的使用 . . . . .	33
2.24.1 screen 常用参数 . . . . .	33
2.24.2 使用 screen . . . . .	33
2.25 iptables 的使用 . . . . .	33
2.25.1 内网机器通过 iptables 访问互联网 . . . . .	36
2.25.2 iptables 之端口转发 . . . . .	37
2.26 qperf 命令的使用 . . . . .	38
2.26.1 参数说明及示例 . . . . .	39
2.27 iperf 命令的使用 . . . . .	39
2.27.1 参数说明及示例 . . . . .	40
2.28 vmstat 命令的使用 . . . . .	40
2.29 iostat 命令的使用 . . . . .	41
2.30 sar 命令的使用 . . . . .	41
2.31 ip 命令的使用 . . . . .	45
2.31.1 显示 IP 信息 . . . . .	45
<b>第三章 vim 编辑器</b>	<b>47</b>
3.1 vim 的几种模式 . . . . .	47
3.1.1 输入模式 . . . . .	47
3.1.2 命令模式 . . . . .	48
3.1.3 vim 的其他一些指令 . . . . .	49
3.2 vim 的一些小技巧 . . . . .	49
3.3 vim 复制粘贴及剪切板 . . . . .	51
3.4 vim 配置文件 . . . . .	51
<b>第四章 Bash 脚本</b>	<b>55</b>
4.1 正则表达式 . . . . .	55
4.1.1 正则表达式语法 . . . . .	55
4.1.2 一些实例 . . . . .	55
4.2 awk . . . . .	55
4.2.1 初步使用 . . . . .	56
4.2.2 awk 程序结构 . . . . .	57

4.2.3	BEGIN 与 END	58
4.2.4	域和记录	58
4.2.5	内置变量	58
4.2.6	One-liners awk 程序	58
4.2.7	条件和循环	58
4.2.8	if-else 语句	59
4.2.9	while 循环	59
4.2.10	for 循环	60
4.2.11	do 循环	60
4.2.12	影响流控制的语句	60
4.2.13	数组	60
4.2.14	表达式	61
4.2.15	函数	63
4.2.16	总结	63
4.3	sed	63
4.3.1	匹配	65
4.3.2	变量定义	65
4.3.3	特殊变量	65
4.3.4	数组	65
4.3.5	删除	65
4.3.6	替换	65
4.3.7	追加、插入和更改	66
4.3.8	模式空间和保留空间	66
4.3.9	流控制	66
4.3.10	地址范围	66
4.3.11	调用外部变量	66
4.3.12	总结	66
4.4	语法介绍	66
4.4.1	变量定义	66
4.4.2	特殊变量	66
4.4.3	变量赋值和替换	67
4.4.4	本地变量与全局变量	67
4.4.5	引用变量	67
4.4.6	数组	67
4.4.7	特殊字符	68
4.5	基本流程	68
4.5.1	if 结构	68
4.5.2	for 结构	69
4.5.3	while 结构	70
4.6	操作字符串	71
4.7	函数	71

4.8 信号捕捉 . . . . .	72
4.9 开机脚本启动顺序 . . . . .	72
4.10 一个实例 . . . . .	72
 <b>第二部分 数据库篇</b>	 <b>77</b>
 <b>第五章 MySQL 数据库基本知识</b>	 <b>79</b>
5.1 存储设备配置 . . . . .	79
5.1.1 构建文件系统 . . . . .	79
5.1.2 挂载文件系统 . . . . .	80
5.2 安装 MySQL . . . . .	80
5.2.1 创建 MySQL 用户 . . . . .	80
5.2.2 安装 MySQL . . . . .	80
5.3 操作系统配置 . . . . .	82
5.3.1 进程文件数配置 . . . . .	82
5.3.2 sysctl 配置 . . . . .	82
5.3.3 cgroup 配置 . . . . .	82
5.3.4 cgrule 配置 . . . . .	84
5.4 MySQL 配置 . . . . .	84
5.4.1 程序文件和目录配置 . . . . .	84
5.4.2 多实例配置 . . . . .	84
5.5 数据库日常管理 . . . . .	86
5.5.1 实例切换 . . . . .	86
5.5.2 MySQL 启动 . . . . .	86
5.5.3 MySQL 关闭 . . . . .	86
5.6 如何进入 MySQL 数据库 . . . . .	86
5.7 如何建库建表 . . . . .	87
5.8 简单 sql 语句的使用 . . . . .	87
5.8.1 create 语句的使用 . . . . .	87
5.8.2 insert 语句的使用 . . . . .	88
5.8.3 select 语句的使用 . . . . .	88
5.8.4 update 语句的使用 . . . . .	89
5.9 数据库文件的备份 . . . . .	91
5.9.1 数据库文件的导出 . . . . .	91
5.9.2 数据库文件的导入 . . . . .	91
5.10 常用命令总结 . . . . .	91
5.10.1 忘记 MySQL 密码 . . . . .	93
 <b>第六章 MySQL 复制</b>	 <b>95</b>
6.1 复制如何工作 . . . . .	95
6.2 MySQL 复制原理 . . . . .	96

6.3	MySQL 同步细节 . . . . .	96
6.4	MySQL 半同步配置 . . . . .	96
<b>第七章</b>	<b>MySQL+KeepAlived</b>	<b>99</b>
7.1	KeepAlived 介绍 . . . . .	99
7.2	测试同步 . . . . .	103
7.3	安装配置 KeepAlived . . . . .	104
7.4	启动 KeepAlived . . . . .	106
<b>第八章</b>	<b>MySQL+LVS</b>	<b>121</b>
<b>第九章</b>	<b>MHA</b>	<b>123</b>
9.1	实验环境说明 . . . . .	123
9.1.1	SSH 无密码通信设置 . . . . .	123
9.2	安装 MySQL . . . . .	123
9.3	安装及配置 MHA . . . . .	123
9.3.1	安装 node 节点 . . . . .	123
9.3.2	安装 manager 节点 . . . . .	123
<b>第三部分</b>	<b>基本服务篇</b>	<b>125</b>
<b>第十章</b>	<b>DHCP</b>	<b>129</b>
<b>第十一章</b>	<b>DNS</b>	<b>131</b>
11.1	测试环境 . . . . .	131
11.2	安装及配置主 DNS . . . . .	131
11.2.1	启动 DNS 服务 . . . . .	133
<b>第十二章</b>	<b>FTP</b>	<b>135</b>
<b>第十三章</b>	<b>NFS</b>	<b>137</b>
13.1	配置 NFS 服务器 . . . . .	137
13.2	配置 NFS 客户端 . . . . .	137
<b>第十四章</b>	<b>Kickstart</b>	<b>139</b>
14.0.1	安装相关软件包 . . . . .	139
<b>第十五章</b>	<b>Samba</b>	<b>143</b>
<b>第十六章</b>	<b>Apache</b>	<b>145</b>
16.1	安装及配置 Apache . . . . .	145
<b>第十七章</b>	<b>Nginx</b>	<b>147</b>
17.1	关于 Nginx . . . . .	147
17.2	Nginx 的安装与启动 . . . . .	147

17.3 Nginx 的基本配置 . . . . .	148
17.3.1 Nginx 主配置概述 . . . . .	148
17.3.2 Nginx 虚拟主机配置 . . . . .	152
17.3.3 安全的连接 https . . . . .	152
17.4 Nginx 日志管理 . . . . .	152
17.5 Nginx 访问控制 . . . . .	152
17.6 Nginx 反向代理 . . . . .	152
17.6.1 Nginx 与 Lua 结合 . . . . .	154
<b>第十八章 LAMP . . . . .</b>	<b>155</b>
18.1 安装依赖包 . . . . .	155
18.2 安装额外的包 . . . . .	155
18.3 编译安装 MySQL . . . . .	156
18.4 编译安装 Apache . . . . .	157
18.5 编译安装 PHP . . . . .	157
18.6 安装 Zend 加速器 . . . . .	158
18.7 整合 Apache 与 PHP . . . . .	158
<b>第十九章 多网卡绑定 bonding . . . . .</b>	<b>161</b>
19.1 bonding 的几种模式 . . . . .	161
19.2 RHEL 下配置 bonding . . . . .	161
19.3 SuSE 下配置 bonding . . . . .	163
19.4 单网卡多 IP 配置 . . . . .	164
<b>第二十章 RAID 技术 . . . . .</b>	<b>165</b>
20.1 RAID 基础知识 . . . . .	165
20.1.1 RAID 解决了什么问题 . . . . .	165
20.2 RAID 实现方式 . . . . .	166
20.2.1 RAID0 数据组织原理 . . . . .	166
20.2.2 RAID1 数据组织原理 . . . . .	166
20.2.3 RAID10 数据组织原理 . . . . .	167
20.2.4 RAID5 数据组织原理 . . . . .	167
20.3 MegaRAID Cli 工具基本使用 . . . . .	167
20.3.1 制作 RAID . . . . .	167
20.3.2 删除 RAID . . . . .	168
<b>第四部分 集群方案篇 . . . . .</b>	<b>169</b>
<b>第二十一章 集群基础知识 . . . . .</b>	<b>173</b>
21.1 集群概述 . . . . .	173
21.2 集群类型 . . . . .	173

<b>第二十二章 Keepalived</b>	<b>175</b>
22.1 Keepalived 简介	175
22.2 Keepalived 安装部署	175
22.2.1 环境准备	175
22.2.2 开始安装	175
22.2.3 Keepalived 配置介绍	175
22.3 运行服务与故障模拟	175
<b>第二十三章 LVS+Keepalived 负载均衡集群</b>	<b>177</b>
23.1 LVS 调度算法	178
23.2 安装 LVS	180
23.2.1 环境准备	180
<b>第二十四章 Heartbeat 高可用集群</b>	<b>181</b>
24.1 安装 Heartbeat	181
24.2 配置 Heartbeat	182
24.3 测试	184
<b>第二十五章 Nginx 负载均衡与高可用集群</b>	<b>187</b>
25.1 upstream 模块介绍	187
25.2 Nginx 负载均衡方法介绍	187
25.3 准备工作	187
25.4 Nginx+KeepAlived 高可用	187
<b>第二十六章 HAproxy 负载均衡与高可用</b>	<b>189</b>
26.1 关于 HAproxy	189
26.2 环境准备	189
26.2.1 测试环境准备	189
26.2.2 安装 EPEL 源	189
26.3 HAproxy 的高可用	189
<b>第五部分 监控篇</b>	<b>191</b>
<b>第二十七章 Zabbix</b>	<b>195</b>
27.1 Zabbix 简介	195
27.2 Zabbix 监控方案介绍	195
<b>第二十八章 Prometheus</b>	<b>197</b>
<b>第六部分 自动化运维篇</b>	<b>199</b>
<b>第二十九章 PXE 批量系统安装</b>	<b>203</b>
29.1 PXE 简介	203



29.2 PXE 工作原理 . . . . .	203
29.2.1 PXE 工作原理框图 . . . . .	203
29.2.2 PXE 工作原理示意图 . . . . .	203
29.3 开始搭建 PXE 服务器 . . . . .	205
29.3.1 安装及配置 DHCP 服务 . . . . .	205
29.4 安装及配置 TFTP 服务 . . . . .	206
29.5 安装及配置 NFS 服务 . . . . .	206
29.6 SLEL11sp2 系统镜像文件 . . . . .	207
29.6.1 目录结构 . . . . .	207
29.6.2 镜像挂载 . . . . .	207
29.6.3 复制系统内核文件 . . . . .	207
29.6.4 创建 default 文件 . . . . .	207
29.6.5 AutoYast 自动化安装脚本 . . . . .	208
<b>第三十章 Cobbler 自动化装机</b>	<b>209</b>
30.1 Cobbler 介绍 . . . . .	209
30.2 Cobbler 安装及配置 . . . . .	209
<b>第三十一章 Omnitty</b>	<b>211</b>
31.1 Omnitty 的简单使用 . . . . .	211
31.1.1 添加机器 . . . . .	211
31.1.2 激活机器 . . . . .	212
31.1.3 开始操作 . . . . .	212
<b>第三十二章 fabric</b>	<b>213</b>
<b>第三十三章 Ansible</b>	<b>215</b>
33.1 关于 Ansible . . . . .	215
33.2 安装 Ansible . . . . .	215
33.3 测试 Ansible . . . . .	215
33.3.1 command 模块 . . . . .	216
33.3.2 shell 模块 . . . . .	217
33.3.3 package 模块 . . . . .	217
33.3.4 service 模块 . . . . .	218
33.3.5 file 模块 . . . . .	219
33.3.6 copy 模块 . . . . .	220
33.3.7 lineinfile 模块 . . . . .	222
33.3.8 setup 模块 . . . . .	222
33.3.9 获得模块的帮助信息 . . . . .	223
33.4 Ansible 工作流程 . . . . .	224
33.5 playbooks . . . . .	224
33.6 一个综合实例 . . . . .	224

<b>第三十四章 SaltStack</b>	<b>229</b>
34.1 初识 SaltStack	229
34.2 安装及运行 SaltStack	229
34.2.1 环境准备	229
34.2.2 安装 SaltStack Master	230
34.2.3 安装 SaltStack Minion	230
34.3 SaltStack 配置	231
34.3.1 Master 端配置	231
34.3.2 Minion 端配置	231
34.4 基本使用	231
34.4.1 单 master 设置	231
34.4.2 基本命令	231
34.4.3 密钥管理	231
34.4.4 定位 minions	231
34.5 理解 YAML	233
34.6 SaltStack 配置管理实例	233
34.6.1 安装 JDK	235
34.6.2 安装 Tomcat	236
34.6.3 安装 Nginx	236
34.6.4 安装 MySQL	236
34.6.5 安装 PHP	236
34.6.6 安装 Redis	236
34.6.7 安装 ELK Stack	236
34.6.8 安装 OpenStack	236
<b>第三十五章 Puppet</b>	<b>237</b>
35.1 关于 Puppet	237
35.2 服务器端 puppetca 安装配置	237
35.2.1 配置 NTP	237
35.2.2 增加虚拟 IP	237
35.2.3 主机名 IP 静态解析配置和验证	237
35.2.4 安装新版本 puppet 软件包	237
35.2.5 修改配置并验证结果	237
35.3 客户端 puppet 安装配置	237
35.4 服务器端 rabbitmq-server 安装配置	237
35.5 服务器端 mcollective 安装配置	237
35.6 客户端 mcollective 安装配置	237
<b>第三十六章 Git</b>	<b>239</b>
36.1 git 简介	239
36.2 安装 git	239
36.3 创建版本库	239

36.4 本地仓库 . . . . .	240
36.4.1 版本回退 . . . . .	242
36.4.2 工作区和暂存区 . . . . .	242
36.4.3 管理修改 . . . . .	242
36.4.4 撤销修改 . . . . .	242
36.4.5 删除文件 . . . . .	242
36.5 远程仓库 . . . . .	242
36.5.1 添加远程库 . . . . .	242
36.5.2 克隆远程库 . . . . .	242
36.6 分支管理 . . . . .	242
36.6.1 创建与合并分支 . . . . .	242
36.6.2 解决冲突 . . . . .	242
36.6.3 分支管理策略 . . . . .	242
36.6.4 Bug 分支 . . . . .	242
36.6.5 Feature 分支 . . . . .	242
36.6.6 多人协作 . . . . .	242
36.7 标签管理 . . . . .	242
36.7.1 创建标签 . . . . .	242
36.7.2 操作标签 . . . . .	242
36.8 使用 GitHub . . . . .	242
36.9 自定义 Git . . . . .	242
36.9.1 忽略特殊文件 . . . . .	242
36.9.2 配置别名 . . . . .	242
36.9.3 搭建 Git 服务器 . . . . .	242
 第七部分 虚拟化及云计算部分	 243
 第三十七章 KVM	 247
37.1 关于 KVM . . . . .	247
37.1.1 KVM 管理工具 libvirt 简介 . . . . .	247
37.1.2 libvirt 中的一些术语 . . . . .	248
37.1.3 检查宿主机是否支持 KVM 虚拟化 . . . . .	248
37.2 安装前的准备工作 . . . . .	249
37.2.1 KVM 测试环境 . . . . .	249
37.2.2 安装 EPEL 源 . . . . .	249
37.2.3 安装 KVM 管理工具 . . . . .	249
37.3 开始部署 KVM 虚拟机 . . . . .	249
37.3.1 创建虚拟机镜像 . . . . .	249
37.3.2 安装虚拟机 . . . . .	250
37.4 KVM 虚拟机管理 . . . . .	251
37.4.1 libvirt 的配置和使用 . . . . .	251

37.4.2 虚拟机拷贝 . . . . .	252
37.4.3 虚拟机克隆 . . . . .	253
37.4.4 增加虚拟机硬盘空间 . . . . .	254
37.4.5 虚拟机硬盘格式转换 . . . . .	254
37.4.6 虚拟机迁移 . . . . .	255
37.4.7 创建虚拟机快照 . . . . .	255
37.5 KVM 虚拟机桥接网络 . . . . .	257
37.6 Libvirt API . . . . .	260
37.6.1 Libvirt API 简介 . . . . .	260
37.6.2 C API 示例 . . . . .	260
37.6.3 Python API 示例 . . . . .	262
<b>第三十八章 OpenStack . . . . .</b>	<b>267</b>
38.1 OpenStack 简介 . . . . .	267
38.1.1 OpenStack 常用组件介绍 . . . . .	267
38.2 安装前的准备工作 . . . . .	267
<b>第三十九章 Docker . . . . .</b>	<b>269</b>
39.1 关于 Docker . . . . .	269
39.2 容器 VS. 虚拟机 . . . . .	269
39.3 Docker 简介 . . . . .	270
39.3.1 Docker 与传统虚拟机建构对比 . . . . .	270
39.3.2 应用容器虚拟化定位 . . . . .	270
39.3.3 Docker 有哪些优势 . . . . .	270
39.3.4 Docker 应用场景 . . . . .	270
39.3.5 Docker 组成 . . . . .	270
39.4 安装及启动第一台 Docker 容器 . . . . .	270
39.5 测试环境 . . . . .	270
39.5.1 安装 Docker . . . . .	270
39.5.2 运行第一台 Docker 容器 . . . . .	271
39.6 容器及镜像管理 . . . . .	271
39.6.1 容器管理 . . . . .	271
39.6.2 镜像管理 . . . . .	271
39.7 构建 Docker 镜像 . . . . .	271
39.7.1 手工构建 . . . . .	271
39.7.2 使用 Dockerfile 构建 . . . . .	271
<b>第四十章 公有云 . . . . .</b>	<b>273</b>
40.1 公有云简介 . . . . .	273

目录	1
<b>第八部分 网络部分</b>	<b>275</b>
<b>第四十一章 TCP/IP</b>	<b>279</b>
41.1 OSI 网络参考模型	279
41.1.1 TCP/IP 三次握手	280
41.1.2 TCP/IP 四次挥手	280
<b>第四十二章 H3C 交换机配置</b>	<b>281</b>
42.1 配置 telnet 方式登录	281
42.1.1 认证方式为 Scheme 时的登录配置	281
42.2 以太网接口配置	281
42.2.1 打开或关闭以太网接口	281
42.2.2 以太网端口基本属性配置	282
42.3 以太网接口的配置显示和维护	282
<b>第九部分 附录部分</b>	<b>283</b>
<b>附录 A 书中的源码</b>	<b>285</b>
A.1 第 1 章插图源码	285
A.2 第 2 章插图源码	287
A.3 第 14 章插图源码	297



## 插图

1.1	Unix/Linux 结构图（测试图形）	10
1.2	美图欣赏	10
2.1	终端颜色效果	17
2.2	kvm 桥接方式的网络拓扑	38
3.1	vim 粘贴板一览	51
3.2	vim 配置文件效果	53
4.1	awk 工作流程	56
4.2	sed 工作流程	63
4.3	for 工作流程	69
4.4	do 工作流程	70
4.5	while 工作流程	71
6.1	MySQL 复制的工作原理	96
14.1	PXE WorkFlow	139
14.2	PXE 工作流程图	140
17.1	两种代理模式比较	153
24.1	heartbeat 实验拓扑	181
29.1	PXE 工作原理示意图	204
33.1	网络七层模型	215
33.2	Playbook	224
34.1	salt 通信示意图	229
37.1	Libvirt 支持的虚拟化类型	248
37.2	节点、域与 Hypervisor 之间的关系	248
39.1	容器架构模型	269
39.2	虚拟化架构模型	270
39.3	系统架构	271

41.1 OSI 七层模型 . . . . .	279
42.1 Linux 虚拟文件（测试图形） . . . . .	281



## 表格

2-1	颜色表 [1]	16
2-2	yum 常用命令选项	20
2-3	grep 常用选项	21
2-4	free 常用选项	25
2-5	tar 通用选项	26
2-6	tar 压缩选项	26
2-7	tar 解压缩选项	26
2-8	cut 常用选项	27
2-9	sort 常用选项	28
2-10	tcpdump 常用选项	32
2-11	iptables 实验环境	36
2-12	sar 常用选项	42
3-1	vim 进入插入模式指令	48
3-2	vim 方向控制键	48
3-3	vim 撤销指令按键	48
3-4	vim 删除指令按键	49
4-1	特殊变量的含义	67
11-1	DNS 演示环境机器一览	131
22-1	KeepAlived 演示环境机器列表	175
29-1	PXE 目录结构	207
33-1	Ansible 测试环境	216
34-1	SaltStack 测试环境机器列表	230
39-1	Docker 演示环境机器一览	270



## 第一部分

### **Gnu/Linux** 基础知识篇



# 第一章 关于本书

GNU/Linux 是 GNU 计划的支持者与开发者，特别是其创立者 Richard Stallman<sup>1</sup> 对于一个以 Linux 闻名的类 Unix 操作系统的称呼。

由林纳斯·托瓦兹及其他人士开发的 Linux 并不是一个完整的操作系统，而仅仅是一个类 Unix 内核。事实上，Linux 一开始是以完成 Minix 内核的功能为目标，Linus 想做一个“比 Minix 更好的 Minix”。而 GNU 计划始于 1984 年，终极目标是完成一套基于自由软件的完整作业操作系统。到 1991 年 Linux 的第一个版本公开发行时，GNU 计划已经完成除了操作系统内核之外的大部分软件，其中包括了一个壳程序（shell），C 语言程序库以及一个 C 语言编译器。林纳斯·托瓦兹及其他早期 Linux 开发人员加入了这些工具，而完成了 Linux 操作系统。但是尽管 Linux 是在 GNU 通用公共许可证下发行，它却不是 GNU 计划的一部分。

正是由于 Linux 使用了许多 GNU 程序，Richard Stallman 认为应该将该操作系统称为“GNU/Linux”比较恰当。[2]

## 1.1 Gnu 计划

GNU 计划，有译为“革奴计划”，是由理查德·斯托曼在 1983 年 9 月 27 日公开发起的，它的目标是创建一套完全自由的操作系统。理查德·斯托曼最早是在 net.unix-wizards 新闻组上公布该消息，并附带一份《GNU 宣言》等解释为何发起该计划的文章，其中一个理由就是要“重现当年软件界合作互助的团结精神”。

UNIX 是一种广泛使用的商业操作系统的名称。由于 GNU 将要实现 UNIX 系统的接口标准，因此 GNU 计划可以分别开发不同的操作系统。GNU 计划采用了部分当时已经可自由使用的软件，例如  $\text{\TeX}$  排版系统和 X Window 视窗系统等。不过 GNU 计划也开发了大批其他的自由软件，这些软件也被移植到其他操作系统平台上，例如 Microsoft Windows、BSD 家族、Solaris 及 MacOS。

为保证 GNU 软件可以自由地“使用、复制、修改和发布”，所有 GNU 软件都包含一份在禁止其他人添加任何限制的情况下，授权所有权利给任何人的协议条款，GNU 通用公共许可证（GNU General Public License，GPL）。这个就是被称为‘公共版权’的概

---

<sup>1</sup>理查德·马修·斯托曼，美国自由软件运动的精神领袖、GNU 计划以及自由软件基金会的创立者。作为一个著名的黑客，他的主要成就包括 Emacs 及后来的 GNU Emacs，GNU C 编译器及 GDB 调试器。他所写作的 GNU 通用公共许可证是世上最广为采用的自由软件许可证，为 copyleft 观念开拓出一条崭新的道路。

1990 年代中期，斯托曼作为一个政治运动者，为自由软件辩护，对抗软件专利及版权法的扩张。他对程式设计方面的投入都放在 GNU Emacs 上。他从演讲中获得的收入，已足够维持自己的生活。他最大的影响是为自由软件运动竖立道德、政治及法律框架。他被许多人誉为当今自由软件的斗士、伟大的理想主义者。

念。GNU 也针对不同场合，提供 GNU 宽通用公共许可证（与 GNU 自由文档许可证这两种协议条款）[5]。

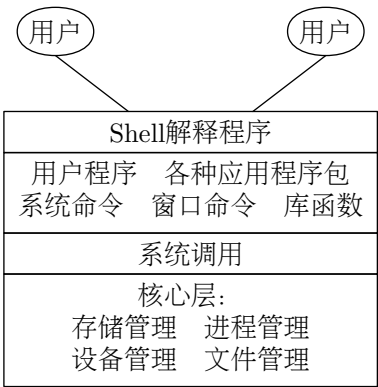


图 1.1 Unix/Linux 结构图（测试图形）

1.1.1 图片欣赏

下面给出几张美图看看。它们分别是 Gnu 的 logo，内核 Linux 的 logo，还有我最喜欢的理查德的照片，最后一张是我在网上找到的，自己稍作了处理，嘻嘻……！



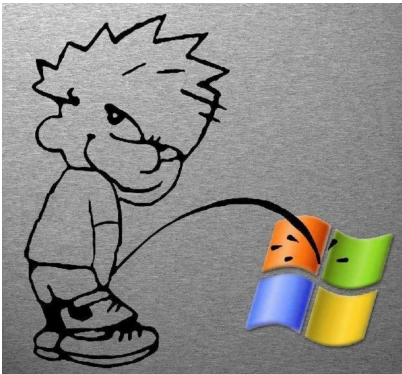
(a) Gnu Logo



(b) Linux Logo



(c) Richard



(d) 网友恶搞

图 1.2 美图欣赏

## 1.2 本书使用的操作系统及环境

本书的所有操作是在作者笔记本上的虚拟机上演示，虚拟机操作系统为 CentOS7u6 64 位。





## 第二章 常用命令使用

### 2.1 命令行下的快捷键

经常在命令行下工作的小伙伴们，可能用的最多的就是两个上下方向键，主要用来调出历史命令；使用左右箭头使光标向后或向前移动以修改上次使用过的命令。其实这样做效率并不是很高，有了快捷键可以让我们的效率提高数倍，而且看起来还更专业、更加 Awesome、更加 Geek。掌握了这些快捷键，我们可以做到手不离主键盘区域，完全可以忽略掉键盘上的四个可爱的箭头。当我们熟练之后，会越发喜欢这种方式。

#### 2.1.1 常用快捷键介绍

下面介绍一些作者在命令行下经常使用的快捷键，这些快捷键在 Emacs 下面是有同样的效果的，不信？你可以试试看。其实，Emacs 是 Gnu/Linux 系统下的命令行编辑器，通过 `/etc/profile` 或 `/etc/bashrc` 等文件都可以找到相关的设置。

##### 1. Ctrl+A 快捷键

这里的 A 可以理解为 Head。当我们按下此组合键时，光标就从当前位置移动到了命令行的起始位置。别只顾着看，动手试试！

##### 2. Ctrl+B 快捷键

这里的 B 可以理解为 Backward，向后的意思。有时在命令行上，我们把某个命令的参数或路径写错了，一般的做法是，使用左箭头，使光标移动到指定的位置，然后修改。其实我们完全可以使用 Ctrl+B 的方式以达到同样的效果。别只顾着看，动手试试！

##### 3. Ctrl+C 快捷键

这个组合键是用来终止当前正在运行的前台进程。在 UNIX 环境高级编程一书上看到了一个用来终止当前运行进程的组合键，是 Ctrl+[3]。别只顾着看，动手试试！

##### 4. Ctrl+D 快捷键

这个组合键的用途也很广，我主要用此组合键来退出某个程序，如 Python、MySQL 等等。在命令行下意思就不同啦，此时的 D 可以理解为 Delete。按下此组合键，会删除当前光标处的字符。别只顾着看，动手试试！

##### 5. Ctrl+E 快捷键

这里的 E 可以理解为 **End**。当在命令行按下此组合键时，我们的可爱的光标就乖乖地跑到了当前命令行的最后。

这是边注一个

#### 6. Ctrl+F 快捷键

这里的 F 可以理解为 **Forward**，向前的意思，等同于按下右箭头。别只顾着看，动手试试！

#### 7. Ctrl+H 快捷键

此组合键相当于键盘上的 **Backspace** 键。按下此组合键，它会从当前光标处开始向后删除字符。别只顾着看，动手试试！

#### 8. Ctrl+J 快捷键

此组合键相当于键盘的回车键。按下此组合键，相当于按了一次回车键。在 **Windows** 的命令行下，**Ctrl+M** 好像是等同于回车键。别只顾看着，动手试试！

#### 9. Ctrl+K 快捷键

这里的 K 可以理解为 **Kill**。按下此组合键，会删除从当前光标到本命令行的结束的位置的所有字符。别只顾着看，动手试试！

#### 10. Ctrl+L 快捷键

这里的 L 可以理解为 **Clear**。按下此组合键相当于执行了 **clear** 这条命令，清除当前屏幕上的内容。别只顾着看，动手试试！

#### 11. Ctrl+N 快捷键

这里的 N 可以理解为 **Next**。这个组合键的作用是用来调出下一条历史命令，与之对应的快捷键 **Ctrl+P** 是调出上一条历史命令。代替了向下的箭头。别只顾着看，动手试试！

#### 12. Ctrl+P 快捷键

这里的 N 可以理解为 **Previous**。这个组合键的作用是用来调出上一条历史命令，与之对应的快捷键 **Ctrl+N** 是调出下一条历史命令。代替了向上的箭头。别只顾着看，动手试试！

#### 13. Ctrl+R 快捷键

这个组合键是用来搜索之前的历史命令。这里的 R 可以理解为 **Reverse**，反向的意思。在 **Emacs** 里为向后搜索，与之对应的是 **Ctrl+S** 快捷键是向前搜索。不过 **Ctrl+S** 在命令行里却不是这个作用，而是用来锁屏的。别只顾着看，动手试试！

#### 14. Ctrl+S 快捷键

这个组合键在 **Emacs** 里为向后搜索，与之对应的是 **Ctrl+S** 快捷键是向前搜索。不过 **Ctrl+S** 在命令行里却不是这个作用，而是用来锁屏的。别只顾着看，动手试试！锁了之后怎么解锁呢？可以是试试 **Ctrl+Q** 组合键。

#### 15. Ctrl+T 快捷键

此组合键是交换两个相邻字符的位置。交换的是当前光标处字符及其当前光标前面的字符。比如我们不小心把 `clear` 命令写成了 `clera`，此时我们也不用把 `ra` 两个字符删掉，然后再写上正确的。此时使我们的光标位于字符 `a` 上，让后按下此组合键，是不是神奇的事情发生了？当然，如果光标在行尾，按下此组合键，它会交换光标前的两个连续的字符。在 Emacs 下面，使用 `Ctrl+X` 与 `Ctrl+T` 两个组合键<sup>1</sup>，可以交换当前光标行与上一行的位置。别只顾着看，动手试试！

#### 16. `Ctrl+W` 快捷键

此组合键在 Emacs 中的作用是剪切选中区域的文本。在命令行上使用该组合键则是往后删除一个字符组合。也就是说，删除光标左边的一个字母组合或单词。比如，我们在此命令行上使用了命令如下，“`service network restart`”，让我们的光标位于字符串的 `restart` 的后面，按下该组合键，看看有何效果？是不是变成“`service network`”了？确实是这样，如果我们使用 `Backspace` 键的话，则需要使用 7 次的按键才能达到一个 `Ctrl+W` 的组合键的效果。嗯，别只顾着看，动手试试？

#### 17. `Alt+.` 快捷键

此组合键是调出上一条命令的最后一个参数。如上一条命令是“`service network restart`”，则“`restart`”就是最后一个参数。如果我们接下来要敲的命令需要用到上一条命令的最后一个参数，则可使用此快捷键，而不需要手工输入“`restart`”了，而且不会出错，节省敲击键盘的次数。如果我们接下来想重启 `httpd` 服务，则只需要输入“`service httpd`”，然后按下“`Alt+.`”即可补全上一条命令的“`restart`”。在有些终端上，按“`Alt+.`”组合键可能会没有效果，这时可以使用“`ESC+.`”组合键代替。在 Emacs 中，`ESC` 键与 `Alt` 键是等价的。可以动手试试该组合键的效果。

## 2.2 使用 *man page* 获得帮助

当我们遇到不会用的系统命令时，该怎么办呢？或许你第一个想到的是 Baidu 或 Google，这样想很正常，可以节省很多时间。如果每次遇到不会的命令，都去找网络，个人觉得这不是一件好的事情，不利于我们的提高。

如果不依靠互联网，该怎么解决呢？那就是依赖系统自带的 *man page* 了。通过它我们可以获取绝大部分的帮助信息，这正是锻炼我们的时候。当然我们也可以使用强大的 `info` 工具来查看帮助。

## 2.3 `echo` 与终端颜色

`echo` 会将输入的字符串送往标准输出。输出的字符串间以空白字符隔开，并在最后加上换行号。

---

<sup>1</sup>先按下 `Ctrl+X`，然后松开 `X`，继续按着 `Ctrl` 键，然后再按下 `T` 键，即可完成两个组合键的操作。别嫌麻烦，习惯就好了。

参数:

1. `n` 不要在最后自动换行
2. `e` 若字符串中出现以下字符, 则特别加以处理, 而不会将它当成一般文字输出:

```
\a 发出警告声;
\b ***前一个字符;
\c 最后不加上换行符号;
\f 换行但光标仍旧停留在原来的位置;
\n 换行且光标移至行首;
\r 光标移至行首, 但不换行;
\t 插入tab;
\v 与\f相同;
\\ 插入\字符;
\nnn 插入nnn(八进制)所代表的ASCII字符;
-help 显示帮助
-version 显示版本信息
```

### 2.3.1 终端颜色

`echo` 字体颜色和背景颜色

`-e` enable interpretation of the backslash-escaped characters listed below

字背景颜色范围:40-47

R	G	B	Color
0	0	0	Black
0	0	1	Blue
0	1	0	Green
0	1	1	Cyan
1	0	0	Red
1	0	1	Magenta
1	1	0	Yellow
1	1	1	White

表 2-1 颜色表 [1]

```
40: 黑
41: 深 红
42: 绿
43: ***
44: 蓝 色
45: 紫 色
46: 深 绿
```

47: 白色

字颜色:30-37

ANSI 控制码的说明:

```
\e[0m 关闭所有属性
\e[1m 设置高亮度
\e[4m 下划线
\e[5m 闪烁
\e[7m 反显
\e[8m 消隐
\e[30m — \e[37m 设置前景色
\e[40m — \e[47m 设置背景色
\e[nA 光标上移n行
\e[nB 光标下移n行
\e[nC 光标右移n行
\e[nD 光标左移n行
\e[y;xH设置光标位置
\e[2J 清屏
\e[K 清除从光标到行尾的内容
\e[s 保存光标位置
\e[u 恢复光标位置
\e[?25l 隐藏光标
\e[?25h 显示光标
```

下面看一个例子:

```
for i in `seq 0 7` ; do echo -e "\033[30;4${i}m      \033[0m"; \
done
```

输出结果为:

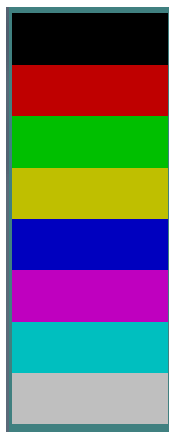


图 2.1 终端颜色效果

## 2.4 date 命令的使用

```
# 设日期
date -s 20091112

# 设时间
date -s 18:30:50

# 7天前日期
date -d "7 days ago" +%Y%m%d

# 5分钟前时间
date -d "5 minute ago" +%H:%M

# 一个月前
date -d "1 month ago" +%Y%m%d

# 日期格式转换
date +%Y-%m-%d -d '20110902'

# 日期和时间
date +%Y-%m-%d_%X

# 纳秒
date +%N

# 换算成秒计算(1970年至今的秒数)
date -d "2012-08-13 14:00:23" +%s

# 将时间戳换算成日期
date -d "@1363867952" +%Y-%m-%d-%T

# 将时间戳换算成日期
date -d "1970-01-01 UTC 1363867952 seconds" +%Y-%m-%d-%T

# 格式化系统启动时间(多少秒前)
date -d "`awk -F. '{print $1}' /proc/uptime` second ago" +%Y-%m-%d %H:%M:%S"
```

## 2.5 yum 命令的使用

安装好系统时,在/etc/yum.repos.d 目录下回有一个 rhel-debuginfo.repo 的文件,我们这里以 redhat 系统为例进行讲解。不管这个配置文件的名字如何,但文件的扩展名须为.repo,如 redhat.repo 也是可以的。我们需要做一些准备工作。

准备系统镜像文件并挂载到本地:

```
[root@iLiuc ~]# ls
```

```
rhel-server-5.5-i386-dvd.iso
```

```
[root@iLiuc ~]# mount -o loop rhel-server-5.5-i386-dvd.iso /media
```

复制镜像里的文件到本地目录:

```
[root@iLiuc ~]# mkdir /iso
```

```
[root@iLiuc ~]# cp -r /media/* /iso
```

修改这个配置文件:

```
[root@iLiuc ~]# cat /etc/yum.repos.d/rhel-debuginfo.repo
[rhel-debuginfo]
name=Red Hat Enterprise Linux $releasever - $basearch - Debug
baseurl=file:///iso/Server
enabled=1
gpgcheck=0
```

几点说明:

1. [rhel-debuginfo] 中括号里的内容可以随意
2. name 这一行可有可无
3. baseurl 这行要指定我们的资源在哪里
4. file://说明我们使用什么协议，也可以是 ftp://等
5. /iso/Server 指明我们的源在/iso/Server 目录下

配置好之后，如何使用呢？直接看操作吧：

- ## 1. 列出我们有哪些 yum 仓库

```
[root@iLiuc ~]# yum repolist
```

- ## 2. 列出仓库里的包

```
[root@iLiuc ~]# yum list
```

Deployment_Guide-en-US.noarch	5.2-11	installed
GConf2.i386	2.14.0-9.el5	installed
ImageMagick.i386	6.2.8.0-4.el5_1.1	installed
MAKEDEV.i386	3.23-1.2	installed
NetworkManager.i386	1:0.7.0-10.el5	installed
NetworkManager-glib.i386	1:0.7.0-10.el5	installed
NetworkManager-gnome.i386	1:0.7.0-10.el5	installed
ORBit2.i386	2.14.3-5.el5	installed
....		
yum-utils.noarch	1.1.16-13.el5	rhel-debuginfo
yum-verify.noarch	1.16-13.el5	rhel-debuginfo

yum-versionlock.noarch	1.1.16-13.el5	rhel-debuginfo
zisoofs-tools.i386	1.0.6-3.2.2	rhel-debuginfo
zsh.i386	4.2.6-3.el5	rhel-debuginfo
zsh-html.i386	4.2.6-3.el5	rhel-debuginfo

一些说明：

- 1. 第一列是我们的软件包名
  - 2. 第二列是对应软件包的版本号
  - 3. 第三列
- + **installed** 表明该软件包已安装
- + **rhel-debuginfo** 表明包未安装

几个常用的 yum 命令：

表 2-2 yum 常用命令选项

命令	说明
repolist	列出我们有哪些 yum 仓库
list	列出仓库里有哪些软件包
install	安装软件包的命令
groupinstall	安装软件包组
erase	移除一个或多个软件包

2.6 parted 命令的使用

Gnu/Linux 系统的分区工具通常可以使用 fdisk 与 parted。我们用的比较多的工具就是 fdisk 了，这里不介绍它的使用了。这里简单的介绍如何使用 parted 工具，对于分区表通常有 MBR 分区表和 GPT 分区表对于磁盘大小小于 2T 的磁盘，我们可以使用 fdisk 和 parted 命令工具进行分区对于 MBR 分区表的特点（通常使用 fdisk 命令进行分区）所支持的最大磁盘大小：2T 最多支持 4 个主分区或者是 3 个主分区加上一个扩展分区对于 GPT 分区表的特点（使用 parted 命令进行分区）支持最大卷：18EB（1EB=1024TB）最多支持 128 个分区

对于 parted 命令工具分区的介绍

最后，fdisk 与 parted 有些差异。fdisk 分区完毕后，需要使用“w”命令才能保存之前所做的一些操作；而 parted 则是实时的，每一步操作不需要保存，即时生效。

2.7 mount 命令的使用

如何挂载 iso 镜像文件呢？我们可以使用一下 mount 命令：

```
[root@iLiuc ~]# mount -o loop rhel-server-5.5-i386-dvd.iso /mnt
意思是把挂rhel-server-5.5-i386-dvd.iso载到/mnt目录下，不过
你得事先有这个镜像文件
```



## 2.8 grep 命令的使用

grep(global regular expression pattern 的缩写)。其实可以把它理解为过滤关键字用的一个程序。具体怎么用，还是看一个实例吧，然后结束本节内容。

### 2.8.1 常用选项

表 2-3 grep 常用选项

-A NUM	打印出紧随匹配的行之后的下文 NUM 行
-B NUM	打印出匹配的行之前的上文 NUM 行
-C NUM	打印出匹配的行的上下文前后各 NUM 行
-b	在输出的每行前面同时打印出当前行在输入文件中的字节偏移量
-c	显示匹配的行数
-f file	从文件 file 中获取模式，每行一个
-H	为每个匹配的文件打印文件名
-I	不搜索二进制文件
-i	忽略大小写
-l	只显示有匹配的文件的文件名
-L	只显示未匹配的文件的文件名
-n	输出行号
-o	只显示匹配字段
-q	quiet 静默模式
-v	只显示不匹配的行

### 2.8.2 一些实例

去掉文件里的注释行和空白行

```
# cat filename | grep -v ^$ | grep -v ^# | sudo tee squid.conf
```

这里我们使用 grep 命令及 cut 命令一起把 eth0 上的 IP 给取出来，看操作：

```
[root@iLiuc ~]# ifconfig eth0
eth0 Link encap:Ethernet HWaddr 5A:B6:4E:85:55:44
inet addr:192.168.18.18 Bcast:192.168.18.255 Mask:255.255.255.0
inet6 addr: fe80::58b6:4eff:fe85:5544/64 Scope:Link
UP BROADCAST RUNNING MULuTICAST MTU:1500 Metric:1
RX packets:11655331 errors:0 dropped:0 overruns:0 frame:0
TX packets:1074797 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:4001012028 (3.7 GiB) TX bytes:628740073 (599.6 MiB)
Interrupt:185
```

```
# 使用grep命令，匹配关键字，缩小范围
[root@iLiuc ~]# ifconfig eth0 |grep "inet addr"
inet addr:192.168.18.18 Bcast:192.168.18.255 Mask:255.255.255.0

# 使用cut命令，把范围再次缩小些
[root@iLiuc ~]# ifconfig eth0 |grep "inet addr" | cut -d: -f2
192.168.18.18 Bcast

# 是不是快出来了，再使用cut一次，IP地址就出来了
[root@iLiuc ~]# 自己写出来吧，我已经写得很多了！
```

## 2.9 crontab 命令的使用

假想这样一个场景：每天的凌晨两点，领导都会要求你重启服务器（当然，这有点变态）。这时，你该怎么办？你是不是每天凌晨两点都要从温暖的被窝里爬出来，然后远程连接服务器，然后重启服务器，然后重新钻进被窝，然后失眠了…。每天都如此，我想你一定会奔溃的。

**crontab 命令可以解救你！crontab 几个字段的说明：**

field	allowed values
-----	-----
minute	0-59
hour	0-23
day of month	1-31
month	1-12 (or names, see below)
day of week	0-7 (0 or 7 is Sun, or use names)

# 查看当前用户的crontab

```
[root@iLiuc ~]# crontab -l
```

```
*/2 * * * * /usr/lib/clear-server/cleargard/cleargard.sh
```

上面语句的意思是，每2分钟去执行/usr/lib/clear-server/cleargard目录下的cleargard.sh脚本，只要系统一直运行，它就会循环往复的执行。

# 编辑crontab

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

## 2.10 find 命令的使用

**find 命令很强大，强大到可以写很多东西。这里就介绍如何简单的使用。直接看例子吧：**

```
# linux文件无创建时间
```

```
# Access 使用时间
```

```
# Modify 内容修改时间
# Change 状态改变时间(权限、属主)
# 时间默认以24小时为单位,当前时间到向前24小时为0天,向前48-72小时为2天
# -and 且 匹配两个条件 参数可以确定时间范围 -mtime +2 -and -mtime -4
# -or 或 匹配任意一个条件

# 按文件名查找
find /etc -name http

# 查找某一类型文件
find . -type f

# 按照文件权限查找
find / -perm

# 按照文件属主查找
find / -user

# 按照文件所属的组来查找文件
find / -group

# 文件使用时间在N天以内
find / -atime -n

# 文件使用时间在N天以前
find / -atime +n

# 文件内容改变时间在N天以内
find / -mtime -n

# 文件内容改变时间在N天以前
find / -mtime +n

# 文件状态改变时间在N天前
find / -ctime +n

# 文件状态改变时间在N天内
find / -ctime -n

# 查找文件长度大于1M字节的文件
find / -size +1000000c -print

# 按名字查找文件传递给-exec后命令
find /etc -name "passwd*" -exec grep "root" {} \;

# 查找文件名,不取路径
```

```
find . -name 't*' -exec basename {} \;
```

# 批量改名(查找err替换为ERR {}文件)

```
find . -type f -name "err*" -exec rename err ERR {} \;
```

# 查找任意一个关键字

```
find 路径 -name *name1* -or -name *name2*
```

## 2.11 top 命令的使用

top 命令可动态显示服务器的进程信息，用户可以通过按键来刷新当前状态。别的不多说，给个例子看看：

```
Tasks: 202 total,   2 running, 199 sleeping,   0 stopped,   1 zombie
Cpu(s):  7.9%us,   1.9%sy,   0.0%ni, 89.5%id,   0.7%wa,   0.0%hi,   0.0%si,   0.0%st
Mem:   6003152k total,  1909420k used,  4093732k free,    73688k buffers
Swap:  6180860k total,    0k used,  6180860k free,   893544k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3852	richard	20	0	1329m	127m	31m	S	22	2.2	2:11.41	vlc
2891	richard	9	-11	418m	6988	4660	S	6	0.1	0:37.86	pulseaudio
2880	richard	20	0	1669m	90m	35m	S	5	1.6	1:35.32	gnome-shell
2452	root	20	0	303m	74m	61m	S	4	1.3	1:23.90	Xorg
3051	richard	20	0	872m	200m	52m	S	1	3.4	1:32.25	firefox
10	root	20	0	0	0	0	S	0	0.0	0:00.57	rcuos/2
77	root	20	0	0	0	0	R	0	0.0	0:01.30	kworker/3:1
909	root	-51	0	0	0	0	S	0	0.0	0:10.11	irq/48-iwlwifi
3025	richard	20	0	581m	19m	11m	S	0	0.3	0:01.72	gnome-terminal
3942	richard	20	0	99.6m	15m	5028	S	0	0.3	0:02.35	python
4025	richard	20	0	17460	1408	980	R	0	0.0	0:00.04	top
1	root	20	0	24740	2620	1352	S	0	0.0	0:00.88	init
2	root	20	0	0	0	0	S	0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0	0.0	0:00.08	ksoftirqd/0
5	root	0	-20	0	0	0	S	0	0.0	0:00.00	kworker/0:0H
6	root	20	0	0	0	0	S	0	0.0	0:01.68	kworker/u16:0
7	root	20	0	0	0	0	S	0	0.0	0:01.60	rcu_sched
8	root	20	0	0	0	0	S	0	0.0	0:01.24	rcuos/0
9	root	20	0	0	0	0	S	0	0.0	0:00.45	rcuos/1
11	root	20	0	0	0	0	S	0	0.0	0:00.28	rcuos/3
12	root	20	0	0	0	0	S	0	0.0	0:00.00	rcuos/4
13	root	20	0	0	0	0	S	0	0.0	0:00.00	rcuos/5
14	root	20	0	0	0	0	S	0	0.0	0:00.00	rcuos/6

```

15 root      20    0    0    0    0 S    0  0.0    0:00.00 rcuos/7
16 root      20    0    0    0    0 S    0  0.0    0:00.00 rcu_bh

```

## 2.12 free 命令的使用

buffer: 缓存磁盘上的数据

cached: 缓存的是将要写往磁盘中的数据

buffer=used-buffer-cached

cached=free+buffer+cached

### 2.12.1 常用选项

下面介绍几个常用的 cut 命令的选项，

表 2-4 free 常用选项

选项	说明
-c	按照字符进行分割
-d	指定分割字段的分隔符，默认是 tab
-f	指定要显示的列

### 2.12.2 一些实例

## 2.13 xargs 命令的使用

## 2.14 tr 命令的使用

## 2.15 tar 命令的使用

tar 命令是一个打包与解包的一个工具，功能很强大。下面介绍一些常用选项及使用示例。

通用选项：

压缩时用的选项：

解压缩用的选项：

举例说明：

```

[root@iLiuc ~]# ls
360fy          CLEAR-VOD-INSTALLPACKAGE.V.2.0.8.tar

```

这里有两个文件，第一个为目录，第二个为压缩包

1. 创建tar包，不压缩

```

[root@iLiuc ~]# tar -cvf 360fy.tar 360fy

```

表 2-5 tar 通用选项

选项	说明
j	使用 bzip2 的压缩方式
t	列出压缩包里有哪些文件，并不解压
z	使用 gzip 的压缩方式
f	指定输出的结果文件。该选项是必选的，不管是压缩还是解压缩
p	保留文件的所有权限
v	压缩或解压缩时，查看其打包过程

表 2-6 tar 压缩选项

选项	说明
c	打包时用的选项，选项 c 与 x 不能同时出现

```
360fy/  
360fy/clearVodMS_360fy.tar.gz  
360fy/vod_yuezizhongxin.tar.gz  
360fy/clear_360fy.sql
```

上面的例子我们使用-v选项，使我们可以看到过程。其中360fy.tar是我们创建的tar包，是360fy这个目录的，后面可以是一个或多个文件。

2. 创建tar包，以gzip方式压缩

```
[root@iLiuc ~]# tar -czvf 360fy.tar.gz 360fy
```

3. 创建tar包，以bzip2的方式压缩

```
[root@iLiuc ~]# tar -cjvf 360fy.tar.bz2 360fy
```

4. 查看压缩包的内容，并不解压缩

```
[root@iLiuc ~]# tar -tf 360fy.tar  
360fy/  
360fy/clearVodMS_360fy.tar.gz  
360fy/vod_yuezizhongxin.tar.gz  
360fy/clear_360fy.sql
```

5. 解压缩

不管是tar包，还是以gzip或bz2压缩的方式，我们使用一下这条命令都是通用的

```
[root@iLiuc ~]# tar -xf 360fy.tar
```

表 2-7 tar 解压缩选项

选项	说明
x	解包时用的选项，选项 c 与 x 不能同时出现

```
[root@iLiuc ~]# tar -xf 360fy.tar.gz
[root@iLiuc ~]# tar -xf 360fy.tar.bz2
```

这里我们没有加-v选项，可以加上-v选项以看到解压过程

## 2.16 read 命令的使用

### 2.16.1 常用选项

### 2.16.2 一些实例

## 2.17 cut 命令的使用

cut 命令有的时候很有用，比如要获得指定的以某个分隔符分割的列时，它就可以做到。其实还有比它更强大的工具，如 sed 及 awk 等，这里并不介绍它们，后续章节有介绍。这里就不提及了，跳过它们吧。下面直接看例子吧：

### 2.17.1 常用选项

下面介绍几个常用的 cut 命令的选项，

表 2-8 cut 常用选项

选项	说明
-c	按照字符进行分割
-d	指定分割字段的分隔符，默认是 tab
-f	指定要显示的列

### 2.17.2 一些实例

接下来演示 cut 命令的一些常用示例，比如我们要查看/etc/passwd 文件中用户名，由于用户名是位于/etc/passwd 文件中的每一行的第一个以冒号为分割的字段，因此，可以使用 cut 命令轻松实现取出用户名的需求，

```
# cut -d: -f1 /etc/passwd
```

-d 指定区分列的定界，默认是tab

-f 指定要显示的列

-c 按字符来切割，如

```
# cut -c1-6 file （取文件第一行的前6个字符）
```

实验：取IP地址

```
# ifconfig wlan0 | grep 'inet addr' | cut -d: -f2 | cut -d' ' -f1
# hostname -i[I]
```

```
# 显示系统中总内存量
# free |tr -s ' ' |sed '/^Mem/!d' |cut -d" " -f2

[root@iLiuc ~]# cat test.txt
chuanchuan:goodboy
chuanchuan goodboy

# 以冒号为分割，显示第一列
[root@iLiuc ~]# cut -d: -f1 test.txt
chuanchuan
chuanchuan goodboy

# 以冒号为分割，显示第二列
[root@iLiuc ~]# cut -d: -f2 test.txt
goodboy
chuanchuan goodboy

# 以空格为分割，显示第一列
[root@iLiuc ~]# cut -d" " -f1 test.txt
chuanchuan:goodboy
chuanchuan
```

我想，知道这么多，应该就可以了。

2.18 sort 命令的使用

2.18.1 常用选项

下面介绍几个常用的 cut 命令的选项，

表 2-9 sort 常用选项

选项	说明
-c	按照字符进行分割
-d	指定分割字段的分隔符，默认是 tab
-f	指定要显示的列

2.18.2 一些实例

2.19 lsof 命令的使用

lsof 是“list open files”的缩写，是一个列出当前系统打开文件的工具。在 linux 环境下，任何事物都以文件的形式存在，通过文件不仅仅可以访问常规数据，还可以访问网



络连接和硬件。所以如传输控制协议 (TCP) 和用户数据报协议 (UDP) 套接字等，系统在后台都为该应用程序分配了一个文件描述符，无论这个文件的本质如何，该文件描述符为应用程序与基础操作系统之间的交互提供了通用接口。因为应用程序打开文件的描述符列表提供了大量关于这个应用程序本身的信息，因此通过 `lsof` 工具能够查看这个列表对系统监测以及排错将是很有帮助的。

`lsof` 不加任何选项，默认输出所有活动进程打开的文件。

```
show all connections with -i
```

```
# lsof -i
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
dhcpcd	6061	root	4u	IPv4	4510		UDP	*:bootpc
sshd	7703	root	3u	IPv6	6499		TCP	*:ssh (LISTEN)
sshd	7892	root	3u	IPv6	6757		TCP	10.10.1.5:ssh->192.168.1.5:49901

```
# Get only IPv6 traffic with -i 6
```

```
# lsof -i 6
```

```
# show only tcp connections (works the same for udp)
```

```
# lsof -iTCP
```

```
# show networking related to a given port using -i :port
```

```
# lsof -i :22
```

```
# show connections to a specific host using @host
```

```
# lsof -i@172.16.25.18
```

```
# show connections based on the host and the port using @host:port
```

```
# lsof -i@172.16.25.18:22
```

```
# Find listening ports
```

```
# find ports that are awaiting connections
```

```
# lsof -i -sTCP:LISTEN
```

```
# User Information
```

```
# We can also get information on various users and what they're doing
```

```
# on the system, including their activity on the network, their
```

```
# interactions with files, etc.
```

```
# show what a given user has open using -u
```

```
# lsof -u postfix
```

```
# show what all users are doing except a certain user using -u ^user
# lsof -u ^postfix

# Kill everything a given user is doing
# kill -9 `lsof -t -u postfix`
```

### 2.19.1 恢复删除的文件

有一次做研发的一位同事，因为系统根目录空间不足，想释放磁盘空间，结果使用 `find` 命令找到了单个文件大于 1GB 的文件，发现 `/var/log/messages` 和 `/var/log/warn` 文件每个都是 1.3GB。他的系统根目录空间只有 50GB 的容量，结果把 `/var/log/messages` 日志文件及 `/var/log/warn` 日志文给删除了。删除之后，却没有达到的目的，磁盘使用空间依然是 100%。

殊不知，当进程打开了某个文件时，只要该进程保持打开该文件，即使将其删除，它依然存在于磁盘中。这意味着，进程并不知道文件已经被删除，它仍然可以向打开该文件时提供给它的文件描述符进行读取和写入。除了该进程之外，这个文件是不可见的，因为已经删除了其相应的目录条目。

拿 `/var/log/messages` 文件为例，看看如何在故意删除后如何找回来。我们先看一下 `/var/log/messages` 文件是什么进程打开的。

```
[root@iLiuc ~]# lsof /var/log/messages
COMMAND      PID USER   FD   TYPE DEVICE SIZE/OFF      NODE NAME
syslog-ng 1493 root    10w   REG   8,2  57599903 3080573 /var/log/messages
```

该命令的输出表明，`/var/log/messages` 文件由 `syslog-ng` 进程打开，当前的进程号为 1493，用户身份是 `root`，打开的文件描述符为 10 且该文件处于只写模式，对应的 `TYPE`（类型）为 `REG`（常规）文件、磁盘位置、文件大小、索引节点。下面一一验证：

```
[root@iLiuc ~]# stat /var/log/messages
File: `/var/log/messages'
Size: 57603503    Blocks: 112632    IO Block: 4096    regular file
Device: 802h/2050d Inode: 3080573    Links: 1
Access: (0640/-rw-r-----)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2015-03-12 15:33:28.000000000 +0800
Modify: 2015-03-12 16:02:01.000000000 +0800
Change: 2015-03-12 16:02:01.000000000 +0800
Birth: -
```

现在，我们可以删除该文件以模拟误删除，

```
[root@iLiuc ~]# rm -f /var/log/messages
[root@iLiuc ~]# lsof -n |grep '(deleted)'
syslog-ng 1493  root 10w  REG   8,2  57605375    3080573 /var/log/messages (de
```

文件已被删除，看怎么恢复吧！从上面的输出信息可以看出之前打开该文件的进程号及文件描述符，有了这两个信息就足够了。接下来，我们去 cat 一下 /proc 目录中相应的目录中的文件描述符，

```
[root@iLiuc ~]# cat /proc/1493/fd/10 |head
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
Oct  9 03:55:32 linux syslog-ng[5747]: syslog-ng starting up; version='2.0.9'
```

通过文件描述符查看了相应的数据，那么就可以使用 I/O 重定向将其复制到文件中，如 cat /proc/1493/fd/10 > /tmp/messages。此时，可以中止该守护进程（这将删除 FD，从而删除相应的文件），将这个临时文件复制到所需的位置，然后重新启动该守护进程。

```
[root@iLiuc ~]# cat /proc/1493/fd/10 > /tmp/messages
[root@iLiuc ~]# /etc/init.d/syslog stop
[root@iLiuc ~]# cp /tmp/messages /var/log/messages
[root@iLiuc ~]# /etc/init.d/syslog start
[root@iLiuc ~]# wc -l /var/log/messages
452113 /var/log/messages

[root@iLiuc ~]# lsof /var/log/messages
COMMAND      PID  USER   FD   TYPE DEVICE SIZE/OFF      NODE NAME
syslog-ng    3819 root    4w    REG    8,2  57608271 3080449 /var/log/messages
```

我们可以看到，已删除的 /var/log/messages 文件已回来了！对于许多应用程序，尤其是日志文件和数据库，这种恢复删除文件的方法非常有用。

## 2.20 netstat 命令的使用

## 2.21 tcpdump 命令的使用

tcpdump 是一款基于命令行的工具，可以通过不同的命令行选项来改变其状态、捕获数据的数量及捕获数据的方法。tcpdump 提供的丰富选项可以使你很容易的改变程序的运行方式。

下面列举一部分比较常用的选项，

表 2-10 tcpdump 常用选项

选项	说明
i	指定侦听的网络接口
v	指定详细模式输出详细的报文信息
vv	指定非常详细的模式输出及非常详细的报文信息
vvv	指定更加详细的模式输出及更详细的报文信息
x	规定 tcpdump 以 16 进制数格式显示数据包
X	规定 tcpdump 以 hex 及 ASCII 格式显示输出
XX	同上，并显示以太网头部信息
n	在捕获过程中不需要向 DNS 查询 IP 地址（显示 IP 地址及端口号）
F	从指定的文件中读取表达式
D	显示 tcpdump 可以侦听的网络接口列表
c	指定捕获多少数据包，然后停止捕获
w	把捕获到的信息写到一个文件中
s	设置捕获数据包的长度为 length

第一种是关于类型的关键字，主要包括 host, net, port, 例如 host 210.27.38.1, 指明 210.27.38.1 是一台主机，net 202.0.0.0 指明 202.0.0.0 是一个网络地址，port 23 指明端口是 23。如果没有指定类型，缺省的类型是 host。

# 想要截获所有210.27.38.1的主机收到的和发出的所有的数据包：

```
# tcpdump host 210.27.38.1
```

# 对本机的udp 123端口进行监视，123为ntp的服务端口

```
# tcpdump udp port 123
```

# 如果想要获取主机210.27.38.1接收或发出的telnet包，使用如下命令：

```
# tcpdump tcp port 23 host 210.27.38.1
```

# 想要获取主机210.27.38.1和主机210.27.38.2或210.27.38.3的通信，使用命令：

# 在命令行中使用括号时，一定要转义

```
# tcpdump host 210.27.38.1 and \(210.27.38.2 or 210.27.38.3\)
```

# 如果想要获取主机210.27.38.1除了和主机210.27.38.2之外所有主机通信的ip包，则：

```
# tcpdump ip host 210.27.38.1 and ! 210.27.38.2
```

```
# tcpdump icmp
```

```
# tcpdump port 3306
```

```
# tcpdump src port 1025
```

```
# tcpdump dst port 389
```

```
# tcpdump src port 1025 and tcp
```

```
# tcpdump udp and src port 53
```

## 2.22 traceroute 命令的使用

## 2.23 wget 命令的使用

通常用来在命令行下面下载文件用的一个命令。

## 2.24 screen 命令的使用

运维人员经常需要 SSH 到远程登录 Unix/Linux 服务器，经常执行一些需要很长时间才能完成的任务，比如作者最近在测试 PCIe SSD 卡的稳定性、InfiniBand 网卡的带宽、系统备份等等。通常，我们会为每一个任务打开一个远程终端，在执行任务期间，必须等待命令执行完毕，表明该任务正常结束。在此期间，不能关闭终端或断开链接，否则这个任务一起被终止。

### 2.24.1 screen 常用参数

### 2.24.2 使用 screen

1. 创建一个新窗口
2. 查看已创建的窗口
3. 会话分离与恢复

## 2.25 iptables 的使用

### 1.1) 设定 INPUT 为 ACCEPT

```
# iptables -P INPUT ACCEPT
```

### 1.2) 设定 OUTPUT 为 ACCEPT

```
# iptables -P OUTPUT ACCEPT
```

### 1.3) 设定 FORWARD 为 ACCEPT

```
# iptables -P FORWARD ACCEPT
```

### 2) 定制源地址访问策略

#### 2.1) 接收来自 192.168.0.3 的 IP 访问

```
# iptables -A INPUT -i eth0 -s 192.168.0.3 -j ACCEPT
```

#### 2.2) 拒绝来自 192.168.0.0/24 网段的访问

```
# iptables -A INPUT -i eth0 -s 192.168.0.0/24 -j DROP
```

3) 目标地址192.168.0.3的访问给予记录,并查看/var/log/message

```
# iptables -A INPUT -s 192.168.0.3 -j LOG
```

4) 定制端口访问策略

4.1) 拒绝任何地址访问本机的111端口

```
# iptables -A INPUT -i eth0 -p tcp --dport 111 -j DROP
```

4.2) 拒绝192.168.0.0/24网段的1024-65534的源端口访问SSH

```
# iptables -A INPUT -i eth0 -p tcp -s 192.168.0.0/24 \
--sport 1024:65534 --dport ssh -j DROP
```

5) 定制CLIENT端的防火墙访问状态

5.1) 清除所有已经存在的规则;

```
# iptables -F
```

5.2) 设定预设策略,除了INPUT设为DROP,其他为ACCEPT;

```
# iptables -P INPUT DROP
# iptables -P OUTPUT ACCEPT
# iptables -P FORWARD ACCEPT
```

5.3) 开放本机的lo可以自由访问;

```
# iptables -A INPUT -i lo -j ACCEPT
```

5.4) 设定有相关的封包状态可以进入本机;

```
# iptables -A INPUT -i eth0 -m state \
--state RELATED,ESTABLISHED -j ACCEPT
# iptables -A INPUT -m state --state INVALID -j DROP
```

6) 定制防火墙的MAC地址访问策略

6.1) 清除所以已经存的规则

```
# iptables -F
# iptables -X
# iptables -Z
```

6.2) 将INPUT设为DROP

```
# iptables -P INPUT DROP
```

6.3) 将目标计算机的MAC设为ACCEPT

```
# iptables -A INPUT -m mac --mac-source \
    00-C0-9F-79-E1-8A -j ACCEPT
```

7) 设定ICMP包，状态为8的被DROP掉

```
# iptables -A INPUT -i eth0 -p icmp \
    --icmp-type 8 -j DROP
```

8) 定制防火墙的NAT访问策略

8.1) 清除所有策略

```
# iptables -F
```

8.2) 重置ip\_forward为1

```
# cat "1" > /proc/sys/net/ipv4/ip_forward
```

8.3) 通过MASQUERADE设定来源于192.168.6.0网段的IP通过192.168.6.217转发出去

```
# iptables -t nat -A POSTROUTING -s 192.168.6.0 -o \
    192.168.6.217 -j MASQUERADE
```

8.4) 通过iptables观察转发的数据包

```
# iptables -L -nv
```

9) 定制防火墙的NAT访问策略

9.1) 清除所有NAT策略

```
# iptables -F -t nat
```

9.2) 重置ip\_forward为1

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
```

9.3) 通过SNAT设定来源于192.168.6.0网段通过eth1转发出去

```
# iptables -t nat -A POSTROUTING -o eth1 \
    -j SNAT --to-source 192.168.6.217
```

9.4) 用iptables观察转发的数据包

```
# iptables -L -nv
```

## 10) 端口转发访问策略

## 10.1) 清除所有NAT策略

```
# iptables -F -t nat
```

## 10.2) 通过DNAT设定为所有访问192.168.6.217的22端口，都访问到192.168.6.191的22

```
# iptables -t nat -A PREROUTING -d 192.168.6.217 \  
-p tcp --dport 22 -j DNAT --to-destination 192.168.6.191:22
```

## 10.3) 设定所有到192.168.6.191的22端口的数据包都通过FORWARD转发

```
# iptables -A FORWARD -p tcp -d 192.168.6.191 --dport 22 -j ACCEPT
```

## 10.4) 设定回应数据包，即通过NAT的POSTROUTING设定，使通讯正常

```
# iptables -t nat -I POSTROUTING -p tcp --dport 22 -j MASQUERADE
```

```
=====
```

```
# iptables -A INPUT -m state --state NEW -p tcp --dport 25 -j ACCEPT
```

```
# iptables -A INPUT -m state --state NEW -j DROP
```

## 修改目标地址在路由之前

```
# iptables -t nat -A PREROUTING -d 2.2.2.2 -p tcp \  
--dport 80 -j DNAT --to-destination 192.168.1.1:80
```

## 修改源地址在路由之后

```
# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 \  
-o eth1 -j SNAT --to-source 1.1.1.1
```

### 2.25.1 内网机器通过 iptables 访问互联网

内网客户机通过一台 Gnu/Linux 服务器访问互联网。服务器的 eth0 网卡可以访问互联网，服务器的 eth1 网卡与内网客户机相连。客户机通过该服务器访问互联网。

实验环境，

表 2-11 iptables 实验环境

角色	IP	网卡	网关
服务端	10.11.1.71/24	eth0	10.11.1.1
	192.168.56.109	eth1	192.168.56.1
客户端	192.168.56.101	eth0	192.168.56.109

实验环境已具备，接下来配置网关服务器。只需要简单的几步就可完成内网客户机的上网需求。首先，开启服务器的内核转发功能，使其具备路由功能，设置如下，



```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

或则,

```
# sysctl -w net.ipv4.ip_forward=1
# sysctl -p
```

设置完毕, 可以在内网客户机上测试与服务器的连通性,

```
[root@client ~]# ping -c 2 192.168.56.109
PING 192.168.56.109 (192.168.56.109) 56(84) bytes of data.
64 bytes from 192.168.56.109: icmp_seq=1 ttl=64 time=5.76 ms
64 bytes from 192.168.56.109: icmp_seq=2 ttl=64 time=0.303 ms

--- 192.168.56.109 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.303/3.034/5.765/2.731 ms

[root@client ~]# ping -c 2 10.11.1.71
PING 10.11.1.71 (10.11.1.71) 56(84) bytes of data.
64 bytes from 10.11.1.71: icmp_seq=1 ttl=64 time=0.331 ms
64 bytes from 10.11.1.71: icmp_seq=2 ttl=64 time=0.331 ms

--- 10.11.1.71 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.331/0.331/0.331/0.000 ms
```

其次, 配置 NAT 规则。由上一步骤的配置后, 我们可以 ping 通服务器的各个网卡的 IP 地址, 但是内网主机还是无法访问互联网。内网机器需要通过地址转换后, 才能访问互联网。接下来配置 NAT 规则,

```
[root@server ~]# iptables -t nat -A POSTROUTING -s 192.168.56.0/24 \
> -o eth0 -j SNAT --to-source 10.11.1.71
[root@server ~]# iptables -A FORWARD -i eth1 -j ACCEPT
```

### 2.25.2 iptables 之端口转发

当我们的 Web 服务器在局域网内部, 而且没有可在 Internet 上使用的真实 IP 地址, 那就可以使用 DNAT 让防火墙把所有到它自己 HTTP 端口的包转发给局域网内部真正的 Web 服务器。目的地址可以是一个范围, 这样的话, DNAT 会为每一个流随机分配一个地址。

注意, DNAT 只能用在 nat 表的 PREROUTING 和 OUTPUT 链中, 或者是被两条链调用的链里。DNAT 的选项为 --to-destination, 一个例子为,

```
# iptables -t nat -A PREROUTING -p tcp -d 116.236.245.210 \
--dport 22 -j DNAT --to-destination 10.10.7.153-10.10.7.158
```

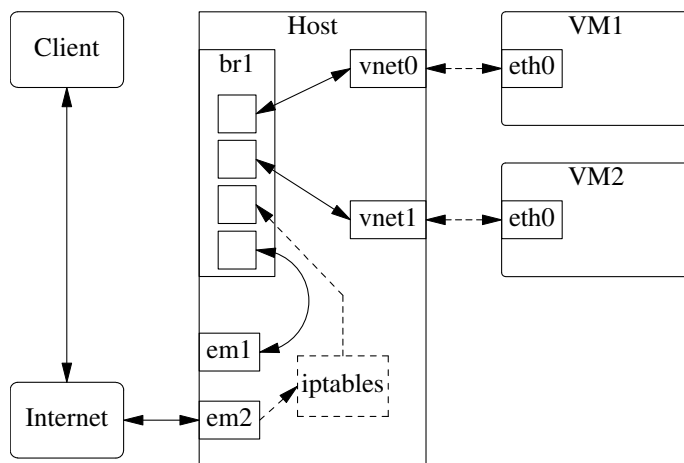
解释：指定要写入 IP 头的地址，这也是包要被转发到的地方。上面的例子就是把所有发往地址 116.236.245.210 的包都转发到一段局域网使用的私有地址中，即 10.10.7.153 到 10.10.7.158。如前所述，在这种情况下，每个连接都会被随机分配到一个要转发到的地址，但同一个连接流总是使用同一个地址。我们可以只指定一个 IP 地址作为参数，这样所有的包都被转发到同一台机器。我们还可以在地址后指定一个或一个范围的端口。如：  
 --to-destination 10.10.7.158:80 或 --to-destination 10.10.7.158:80-100。  
 要注意，只有先用 --protocol 指定了 TCP 或 UDP 协议，才能使用端口。

下面来一个具体的例子，大致理解一下它是如何工作的。比如，我们想通过 Internet 发布我们的网站，但是 HTTP 服务器在我们的内网里，而且我们对外只有一个公有 IP 地址，就是防火墙那个对外的 IP，即 INET\_IP。防火墙还有一个内网的 IP，即 LAN\_IP，HTTP 的 IP 是 HTTP\_IP（这也是内网地址）。为了完成我们设想，要做的第一件事就是把下面的这个简单的规则加入到 nat 表中的 PREROUTING 链中：

```
# iptables -t nat -A PREROUTING --dst INET_IP \
-p tcp --dport 80 -j DNAT --to-destination HTTP_IP
```

有了这条规则，所有从 Internet 来的并防火墙的 80 端口的数据包都会被转发到内网的 HTTP 服务器上。下面是数据包

图 2.2 kvm 桥接方式的网络拓扑



## 2.26 qperf 命令的使用

我们在做网络服务器的时候，通常会很关心网络的带宽和延迟。因为我们的很多协议都是 request-response 协议，延迟决定了最大的 QPS，而带宽决定了最大的负荷。通常我们知道自己的网卡是什么型号，交换机什么型号，主机之间的物理距离是多少，理论上知道带宽和延迟是很多的。但是现实的情况是，真正的带宽和延迟情况会有很多变数的，比如说网卡驱动，交换机跳数，丢包率，协议栈配置，就实际速度而言，都很大的影响了数值的估算。所以我们需要找到工具来实际测量下。

SUSE11sp2 发行版里面自带，方便安装，专业有效，能够针对 TCP 和 RDMA 进行带宽和延迟的详细测试。

```
# zypper install -y qperf
```

由于我们需要测试 Infiniband 的传输速率，在安装之前请先确认安装了 InfiniBand 的相关包，比如 librdmacm, libibverbs 等。另外，也可以选择使用源码包编译和安装 qperf，但是需要注意，在安装之前也需要将 infiniband 相关的包先安装上，否则 RDMA 的相关测试也将无法进行。

```
# zypper install -y librdmacm libibverbs
```

### 2.26.1 参数说明及示例

qperf 分为服务器端和客户端。客户端通过发送请求并获得响应来获得服务器端和客户端之间的网络带宽以及延迟等信息。

参数名	参数说明
<server_ip>	指定服务器的地址
time	指定网络测试时间。默认单位为秒，单位可以通过后缀为 m,h,d 指定为分钟，小时，天
conf	测试输出中显示本地和远端服务器和操作系统配置
use_bits_per_sec	使用 b(bit) 而不是 B(byte) 来显示网络速度
precision 2	设置显示小数点后几位。这里设置为显示小数点后两位
verbose_more	显示更详细的配置和状态信息
loop msg_size:1:1025k:*2	loop 表示对指定的指标值进行轮询。这里设置为对 msg_size 轮询 1, 2, 4, 8...1024k，获得对应的测试结果，下次测试的指标值是上次测试指标值的 *2 倍
tcp_bw	对 tcp 的带宽进行测试
tcp_lat	对 tcp 的延迟进行测试
udp_bw	对 udp 的带宽进行测试
udp_lat	对 udp 的延迟进行测试
sdp_bw	对 sdp 的延迟进行测试
sdp_lat	对 sdp 的延迟进行测试

## 2.27 iperf 命令的使用

iperf 工具我们主要

首先到官网获取 iperf 工具，并把该工具放到合适的位置。

```
# wget https://iperf.fr/download/iperf_2.0.2/iperf_2.0.2-4_amd64
```

```
# chmod +x iperf_2.0.2-4_amd64
# mv iperf_2.0.2-4_amd64 /usr/bin/iperf
```

### 2.27.1 参数说明及示例

参数名	参数说明
-server	以服务端模式运行
-udp	指定测试 UDP，默认为 TCP 带宽测试
-client <host>	以客户端模式运行，并连接 <host>
-bandwidth	指定测试中所使用的带宽，单位为 [KM]，默认为 1Mbit/sec
-time	指定测试时间，单位为秒
-interval	指定多少时间间隔来报告测试结果，时间单位为秒
-format [kmKM]	指定报告的输出格式，单位分别为 Kbits, Mbits, Kbytes, Mbytes

#### 1. 以太网 UDP 丢包率测试

```
A0304010:~ # iperf --server --udp
A0305010:~ # iperf --udp --client 172.16.25.39 --interval 1 \
--time 120 --bandwidth 900M
```

#### 2. InfiniBand 网络 UDP 丢包率测试

```
# iperf --server --udp
# iperf --udp --client 11.11.11.39 --interval 1 \
--time 120 --bandwidth 1024M
```

#### 3. 如果不指定-udp 选项，默认就是测试 TCP 带宽

```
A0304010:~ # iperf --server
A0305010:~ # iperf --client 172.16.25.39 -f M
```

## 2.28 vmstat 命令的使用

Linux 下 vmstat 输出释疑：

Vmstat

```
procs -----memory----- ---swap-- -----io----- --system-- ----cpu-
r b   swpd free buff cache          si so      bi bo      in cs    us sy ic
0 0   100152 2436 97200 289740          0 1      34 45      99 33    0 0 99
```

procs r 列表示运行和等待 cpu 时间片的进程数，如果长期大于 cpu 个数，说明 cpu 不足，需要增加 cpu。

b 列表示在等待资源的进程数，比如正在等待 I/O、或者内存交换等。

memory swpd 切换到内存交换区的内存数量 (k 表示)。如果 swpd 的值不为 0, 或者比较大, 比如超过了 100m, 只要 si、so 的值长期为 0, 系统性能还是正常

free 当前的空闲页面列表中内存数量 (k 表示)

buff 作为 buffer cache 的内存数量, 一般对块设备的读写才需要缓冲。

cache: 作为 page cache 的内存数量, 一般作为文件系统的 cache, 如果 cache 较大, 说明用到 cache 的文件较多, 如果此时 IO 中 bi 比较小, 说明文件系统效率比较好。

swap si 由内存进入内存交换区数量。

so 由内存交换区进入内存数量。

IO bi 从块设备读入数据的总量 (读磁盘) (每秒 kb)。

bo 块设备写入数据的总量 (写磁盘) (每秒 kb)

system 显示采集间隔内发生的中断数

in 列表示在某一时间间隔中观测到的每秒设备中断数。

cs 列表示每秒产生的上下文切换次数, 如当 cs 比磁盘 I/O 和网络信息包速率高得多, 都应进行进一步调查。

cpu 表示 cpu 的使用状态

us 列显示了用户方式下所花费 CPU 时间的百分比。us 的值比较高时, 说明用户进程消耗的 cpu 时间多, 但是如果长期大于 50%, 需要考虑优化用户的程序。

sy 列显示了内核进程所花费的 cpu 时间的百分比。这里 us + sy 的参考值为 80%, 如果 us+sy 大于 80% 说明可能存在 CPU 不足。

wa 列显示了 IO 等待所占用的 CPU 时间的百分比。这里 wa 的参考值为 30%, 如果 wa 超过 30%, 说明 IO 等待严重, 这可能是磁盘大量随机访问造成的, 也可能磁盘或者磁盘访问控制器的带宽瓶颈造成的 (主要是块操作)。

id 列显示了 cpu 处在空闲状态的时间百分比

## 2.29 iostat 命令的使用

## 2.30 sar 命令的使用

sar 命令行的常用格式:

```
sar [options] [-A] [-o file] t [n]
```

在命令行中, n 和 t 两个参数组合起来定义采样间隔和次数, t 为采样间隔, 是必须有的参数, n 为采样次数, 是可选的, 默认值是 1, -o file 表示将命令结果以二进制格式存放在文件中, file 在此处不是关键字, 是文件名。options 为命令行选项, sar 命令的选项很多, 下面只列出常用选项:

选项	说明
-A	所有报告的总和
-u	CPU 利用率
-v	进程、I 节点、文件和锁表状态
-d	硬盘使用报告
-r	没有使用的内存页面和硬盘块
-g	串口 I/O 的情况 (centos 5 中无此选项)
-b	缓冲区使用情况
-a	文件读写情况
-c	系统调用情况
-R	进程的活动情况
-y	终端设备活动情况
-w	系统交换活动

表 2-12 sar 常用选项

例一：使用命令行 `sar -u t n`

例如，每 60 秒采样一次，连续采样 5 次，观察 CPU 的使用情况，并将采样结果以二进制形式存入当前目录下的文件 `lavenliu` 中，需键入如下命令：

```
# sar -u -o lavenliu 60 5
SCO_SV      scosysv  3.2v5.0.5  i80386      10/01/2001
14:43:50      %usr        %sys        %wio        %idle(-u)
14:44:50        0            1            4            94
14:45:50        0            2            4            93
14:46:50        0            2            2            96
14:47:50        0            2            5            93
14:48:50        0            2            2            96
Average        0            2            4            94
```

在显示内容包括：

- %usr：CPU处在用户模式下的时间百分比。
- %sys：CPU处在系统模式下的时间百分比。
- %wio：CPU等待输入输出完成时间的百分比。
- %idle：CPU空闲时间百分比。

在所有的显示中，我们应主要注意%wio和%idle，%wio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。

如果要查看二进制文件 `lavenliu` 中的内容，则需键入如下 `sar` 命令：

```
# sar -u -f lavenliu
```

可见，sar 命令即可以实时采样，又可以对以往的采样结果进行查询。

例二：使用命令 `sar -v t n`

例如，每 30 秒采样一次，连续采样 5 次，观察核心表的状态，需键入如下命令：

```
# sar -v 30 5
SCO_SV scosysv 3.2v5.0.5 i80386 10/01/2001
10:33:23 proc-sz ov inod-sz ov file-sz ov lock-sz      (-v)
10:33:53 305/ 321 0 1337/2764 0 1561/1706 0 40/ 128
10:34:23 308/ 321 0 1340/2764 0 1587/1706 0 37/ 128
10:34:53 305/ 321 0 1332/2764 0 1565/1706 0 36/ 128
10:35:23 308/ 321 0 1338/2764 0 1592/1706 0 37/ 128
10:35:53 308/ 321 0 1335/2764 0 1591/1706 0 37/ 128
```

显示内容包括：

**proc-sz:** 目前核心中正在使用或分配的进程表的表项数，由核心参数 **MAX-PROC** 控制。**inod-sz:** 目前核心中正在使用或分配的 i 节点表的表项数，由核心参数 **MAX-INODE** 控制。**file-sz:** 目前核心中正在使用或分配的文件表的表项数，由核心参数 **MAX-FILE** 控制。**ov:** 溢出出现的次数。**Lock-sz:** 目前核心中正在使用或分配的记录加锁的表项数，由核心参数 **MAX-FLCKRE** 控制。

显示格式为：实际使用表项/可以使用的表项数

显示内容表示，核心使用完全正常，三个表没有出现溢出现象，核心参数不需调整，如果出现溢出时，要调整相应的核心参数，将对应的表项数加大。

例三：使用命令 `sar -d t n`

例如，每 30 秒采样一次，连续采样 5 次，报告设备使用情况，需键入如下命令：

```
# sar -d 30 5
SCO_SV scosysv 3.2v5.0.5 i80386 10/01/2001
11:06:43 device %busy      avque      r+w/s      blks/s      avwait avserv (-d)
11:07:13 wd-0    1.47        2.75        4.67        14.73        5.50 3.14
11:07:43 wd-0    0.43        18.77       3.07        8.66        25.11 1.41
11:08:13 wd-0    0.77        2.78        2.77        7.26        4.94 2.77
11:08:43 wd-0    1.10        11.18       4.10        11.26       27.32 2.68
11:09:13 wd-0    1.97        21.78       5.86        34.06       69.66 3.35
Average wd-0    1.15        12.11       4.09        15.19       31.12 2.80
```

显示内容包括：

**device:** sar 命令正在监视的块设备的名字。  
**%busy:** 设备忙时，传送请求所占时间的百分比。  
**avque:** 队列站满时，未完成请求数量的平均值。  
**r+w/s:** 每秒传送到设备或从设备传出的数据量。

blks/s: 每秒传送的块数, 每块512字节。

avwait: 队列占满时传送请求等待队列空闲的平均时间。

avserv: 完成传送请求所需平均时间(毫秒)。

在显示的内容中, wd-0 是硬盘的名字, %busy 的值比较小, 说明用于处理传送请求的有效时间太少, 文件系统效率不高, 一般来讲, %busy 值高些, avque 值低些, 文件系统的效率比较高, 如果%busy 和 avque 值相对比较高, 说明硬盘传输速度太慢, 需调整。

例四: 使用命令 `sar -b t n`

例如, 每 30 秒采样一次, 连续采样 5 次, 报告缓冲区的使用情况, 需键入如下命令:

```
# sar -b 30 5
SCO_SV scosysv 3.2v5.0.5 i80386 10/01/2001
14:54:59 bread/s lread/s %rcache bwrite/s lwrite/s %wcache pread/s pwrite/s (-
14:55:29 0      147      100      5      21      78      0      0
14:55:59 0      186      100      5      25      79      0      0
14:56:29 4      232       98      8      58      86      0      0
14:56:59 0      125      100      5      23      76      0      0
14:57:29 0       89      100      4      12      66      0      0
Average  1      156       99      5      28      80      0      0
```

显示内容包括:

bread/s: 每秒从硬盘读入系统缓冲区buffer的物理块数。

lread/s: 平均每秒从系统buffer读出的逻辑块数。

%rcache: 在buffer cache中进行逻辑读的百分比。

bwrite/s: 平均每秒从系统buffer向磁盘所写的物理块数。

lwrite/s: 平均每秒写到系统buffer逻辑块数。

%wcache: 在buffer cache中进行逻辑读的百分比。

pread/s: 平均每秒请求物理读的次数。

pwrite/s: 平均每秒请求物理写的次数。

在显示的内容中, 最重要的是%cache 和%wcache 两列, 它们的值体现着 buffer 的使用效率, %rcache 的值小于 90 或者%wcache 的值低于 65, 应适当增加系统 buffer 的数量, buffer 数量由核心参数 NBUF 控制, 使%rcache 达到 90 左右, %wcache 达到 80 左右。但 buffer 参数值的多少影响 I/O 效率, 增加 buffer, 应在较大内存的情况下, 否则系统效率反而得不到提高。

例五: 使用命令 `sar -g t n` 例如, 每 30 秒采样一次, 连续采样 5 次, 报告串口 I/O 的操作情况, 需键入如下命令:

```
# sar -g 30 5
SCO_SV scosysv 3.2v5.0.5 i80386 11/22/2001
17:07:03 ovsiohw/s ovsiodma/s ovclist/s (-g)
17:07:33 0.00 0.00 0.00
17:08:03 0.00 0.00 0.00
```



17:08:33	0.00	0.00	0.00
17:09:03	0.00	0.00	0.00
17:09:33	0.00	0.00	0.00
Average	0.00	0.00	0.00

显示内容包括：

ovsiohw/s: 每秒在串口 I/O 硬件出现的溢出。

ovsiodma/s: 每秒在串口 I/O 的直接输入输出通道高速缓存出现的溢出。

ovclist/s : 每秒字符队列出现的溢出。

在显示的内容中，每一列的值都是零，表明在采样时间内，系统中没有发生串口 I/O 溢出现象。

sar 命令的用法很多，有时判断一个问题，需要几个 sar 命令结合起来使用，比如，怀疑 CPU 存在瓶颈，可用 sar -u 和 sar -q 来看，怀疑 I/O 存在瓶颈，可用 sar -b、sar -u 和 sar -d 来看。

## 2.31 ip 命令的使用

### 2.31.1 显示 IP 信息

同样，在我的 SUSE 11sp2 64bit 上运行也是同样的输出，输出内容略。以下几个包是 SUSE 的 OEM 厂商给出的 Bash 最新的升级包。



## 第三章 vim 编辑器

当我们基本熟悉了操作系统的使用，是不是还少了什么件事？恭喜你猜对了，那就是在系统里面编辑一些东西，我们总该留点什么东西吧，你说是不是呢？

传说中，蔚蓝的地球上流传着两大神器，一个是 vi<sup>1</sup>，一个是 emacs。据说 Vim 是编辑器之神，而 Emacs 是神的编辑器。追求独步天下的高手和低手们争着一睹它们的风采，可看到它们朴素单薄的界面后，不禁心下怀疑：这就是神器吗？至有人生了轻视之心。

本文就向你展示如何使用 vim，掌握它最基本的使用即可。作者刚开始使用 vim 的时候，总是不由自主地想动动旁边的鼠标，大概用了两个晚上的时间才熟练的掌握。Emacs 可以根据自己兴趣看一看，作者使用了一年多的时间，也没觉得自己比那些使用 vim 的家伙强<sup>2</sup>，我的 vi 用的比 Emacs 要熟练。嘿嘿！

### 3.1 vim 的几种模式

vim 有两种模式，一个为输入模式，一个为命令模式。在系统提示字符 (如 \$、#) 下敲入 vim filename，如果没有 filename 这个文件则会创建，若有则打开。vim 可以自动帮你载入所要编辑的文件或是开启一个新文件（如果该文件不存在或缺少文件名）。进入 vim 后屏幕左方会出现波浪符号，凡是列首有该符号就代表此列目前是空的。

如上所述，vim 存在两种模式：指令模式和输入模式。在指令模式下输入的按键将做为指令来处理：如输入 i，vim 即认为是在当前位置插入字符。而在输入模式下，vim 则把输入的按键当作插入的字符来处理。指令模式切换到输入模式只需键入相应的输入命令即可（如 a，A），而要从输入模式切换到指令模式，则需在输入模式下键入 ESC 键，如果不晓得现在是处于什么模式，可以多按几次 ESC，系统如发出哔哔声就表示已处于指令模式下了。

#### 3.1.1 输入模式

如何进入输入模式？下面列出几个进入插入模式的指令：

大家注意到了没有，大写与小写分别控制反向与正向。如何退出呢？在插入模式下，需要按下键盘的 ESC 键，进入命令模式，再从命令模式下，输入” :wq “即可退出。wq 的意思是保存并退出，如果不想保存，只需输入” :q “即可。如果修改后，发现改错了文件，并不想保存了，在命令模式下输入” :q! “即可，加上” ! “作用是强制退出不保存，其实也可以强制保存并退出的，该怎么输入？想想看。

---

<sup>1</sup>vim 是 vi 的升级版，一般认为 vi 既 vim，vim 既 vi。也可理解为 vi 为 vim 的一个别名。

<sup>2</sup>这里作者并不想卷入编辑器之战，不想得罪任何一方阵营，两种编辑器我都会使用，上班时用 vi，不上班时用 Emacs；如果真的得罪了某方阵营，请原谅我的无知。

表 3-1 vim 进入插入模式指令

指令	说明
i	在光标所在处进入插入模式
I	在当前行的行首进入插入模式
a	在光标的后面进入插入模式
A	在当前行的行尾进行插入模式
o	在光标所在行的下面新开一行
O	在光标所在行的上面新开一行
s	删除光标所在字符并进入插入模式
S	删除光标所在行并进入输入模式

### 3.1.2 命令模式

在刚打开 vim 编辑器时，这时所处的模式是命令模式；从输入模式返回到命令模式，只需要按下键盘上的 ESC 键即可返回到命令模式。下面介绍几组很实用的指令，主要涉及删除、复制及粘贴的指令，还有几个控制光标移动的指令。

光标方向控制：

表 3-2 vim 方向控制键

指令	说明
h	光标向前移动
j	光标向下移动
k	光标向上移动
l	光标想后移动
G	按下 shift 与 G 组合，则跳到文件末尾
gg	连续按两次 G，跳到文件的首行

撤销指令：

表 3-3 vim 撤销指令按键

指令	说明
u	按一次，撤销上一次的操作，按两次呢？哈哈
:e!	直接回到了刚打开时的状态

删除指令：

表 3-4 vim 删除指令按键

指令	说明
dd	删除当前光标所在行
ndd	删除当前光标所在行及以下行共 n 行，其中 n 为数字

### 3.1.3 vim 的其他一些指令

## 3.2 vim 的一些小技巧

# 打开文件定位到指定行

```
vim +24 file
```

# 打开多个文件

```
vim file1 file2
```

# 垂直分屏

```
vim -O file1 file2
```

# 水平分屏

```
vim -o file1 file2
```

# 上下分割打开新文件

```
sp filename
```

# 左右分割打开新文件

```
vsp filename
```

# 多个文件间操作 大写W

# 操作：

# 关闭当前窗口c

# 屏幕高度一样=

# 增加高度+

# 移动光标所在屏 右l 左h 上k 下j 中h 下一个w

Ctrl+W [操作]

# 编辑下一个文件

```
:n
```

# 编辑下二个文件

```
:2n
```

```
# 编辑前一个文件
```

```
:N
```

```
# 回到首文件
```

```
:rew
```

```
# 打开行号
```

```
:set nu
```

```
# 取消行号
```

```
:set nonu
```

```
# 跳转到200
```

```
200G
```

```
# 取消高亮
```

```
:nohl
```

```
# 设置自动缩进
```

```
:set autoindent
```

```
# 查看文本格式
```

```
:set ff
```

```
# 改为unix格式
```

```
:set binary
```

```
# 向前翻页
```

```
ctrl+ U
```

```
# 向后翻页
```

```
ctrl+ D
```

```
# 全部替换
```

```
% s/字符1/字符2/g
```

```
# 文档加密
```

```
X
```

### 3.3 vim 复制粘贴及剪切板

复制粘贴指令：

指令	说明
yy	复制当前光标所在行
nyy	复制当前行及以下行共 n 行，n 为数字
p	将复制的内容粘贴到当前光标的下面一行，可以与数字一起使用，如 np
P	将复制的内容粘贴到当前光标的上面一行，可以与数字一起使用，如 nP

vim 有 12 个粘贴板，分别是 0、1、2、...、9、a、“、+；用:reg 命令可以查看各个粘贴板里的内容。效果如图所示：

```

Terminal
File Edit View Search Terminal Help
Python
R
setfont.txt
unpv13e
[No Name]^I[+] [unix/] [ASCII=111/HEX=6F] 15,1 65%
:reg
--- Registers ---
"  oracle^J
"0  ^I^ISimSun: <font size="5" face="SimSun">沁园春</font><br />^J
"1  ^J
"2  #include "apue.h"^J#include <errno.h>^I^I/* for definition of errno */^J#i
"3  static void^Ierr_doit(int, int, const char *, va_list);^J^J/*^J * Nonfatal
"4  #include "error.h"^J
"5  #include "apue.h"^J
"6  ^J
"7  ^J
"8  ^J
"9  ^J
"a  Nodejs^J
"b  oracle^J
"-
":  reg
"/  [:lower:]
Press ENTER or type command to continue

```

图 3.1 vim 粘贴板一览

在 vim 中简单用 y 只是复制到”(双引号) 粘贴板里，同样用 p 粘贴的内容也是这个粘贴板里的内容；要将 vim 的内容复制到某个粘贴板，需要退出编辑模式，进入正常模式后，选择要将复制的内容，然后按”Ny(注意带引号) 完成复制，其中 N 为粘贴板号(注意是按一下双引号后按粘贴板号再按 y)，比如要把内容复制到粘贴板 a，选中内容后按”ay 就可以了。

有两点需要说明一下：”号粘贴板(临时粘贴板) 比较特殊，直接按 y 就复制到这个粘贴板中了，直接按 p 就粘贴这个粘贴板中的内容；+ 号粘贴板是系统粘贴板，用”+ y 将内容复制到该粘贴板后可以使用 Ctrl+V 将其粘贴到其他文档(如 firefox、gedit) 中，同理，要把在其他地方用 Ctrl+C 或右键复制的内容复制到 vim 中，需要在正常模式下按”+p；

### 3.4 vim 配置文件

我相信，每个有想法的家伙肯定不会满足默认的配置，他们总喜欢捣鼓。这个时候我们就可以定制自己的编辑器了，以满足我们日常编辑的需求。Emacs 的定制配置更是令

人蛋疼，诸多选项、诸多值……。作者曾经也是为此事而蛋疼不已，只是学会了在 Emacs 里配置 L<sup>A</sup>T<sub>E</sub>X 的一些配置项，其它神马的，作者是一窍不通。

令人蛋疼的事就不多说，我比较喜欢简洁的配置。本人也不会开发，一些简单的配置就能满足日常需要了，再说这个东东用的也不多。其它高大上的配置东东对我来说也是浮云。下面是我常用的 vim 的配置文件<sup>3</sup>，配置如下：

```
[root@iLiuc ~]# cat ~/.vimrc
set nocompatible
filetype indent on
filetype plugin on
set tabstop=4
set shiftwidth=4
set autoindent
set cindent
set smartindent
set hlsearch
syntax enable
set dict=/usr/share/dict/words

if has('statusline')
    set laststatus=2

    " Broken down into easily includeable segments
    set statusline=%<%f\           " Filename
    set statusline+=%w%h%m%r      " Options
    set statusline+=\ [%{&ff}]/%Y  " filetype

    " ASCII/Hexadecimal value of under current cursor
    set statusline+=\ [ASCII=\%03.3b/HEX=\%02.2B]

    " Right aligned file nav info
    set statusline+=%=-14.(%l,%c%V%)\ %p%%
endif
endif
```

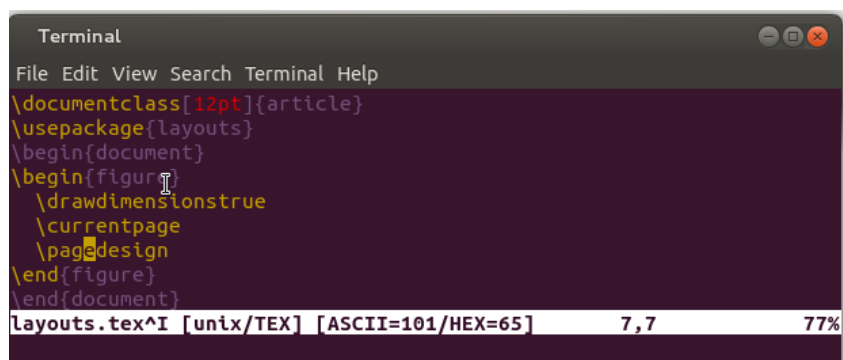
效果如下：

哦！天呢。根本停不下来了。就说这么多吧，如果我要再写下去，就不止这么多。呵呵！只不过是两年没用认真的使用了。呵呵！vim 的使用就介绍这么多，我想仅仅学会这些就能让初学者的效率有很大的提高，但还不够高，这里并没有介绍正则表达式等。

---

<sup>3</sup>我的配置文件基本上是随身携带或者上传到云盘，然后把配置文件放到公司服务器上自己的家目录





```
Terminal
File Edit View Search Terminal Help
\documentclass[12pt]{article}
\usepackage{layouts}
\begin{document}
\begin{figure}
  \drawdimensionstrue
  \currentpage
  \page design
\end{figure}
\end{document}
layouts.tex^I [unix/TEX] [ASCII=101/HEX=65] 7,7 77%
```

图 3.2 vim 配置文件效果



## 第四章 Bash 脚本

无聊地重复一件事确实惹人厌！当然了，数次重复一些命令是不是也是很不爽的一件事呢？如果是请跟随我往下看。当我们熟悉了操作系统后，以期望有更好的提高，那就是写脚本了。

### 4.1 正则表达式

Regular expression (abbreviated regex or regexp) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"like operations.

正则表达式就是由一系列特殊字符组成的字符串, 其中每个特殊字符都被称为元字符, 这些元字符并不表示为它们字面上的含义, 而会被解释为一些特定的含义. 举个例子, 比如引用符号, 可能就是表示某人的演讲内容, 同上, 也可能表示为我们下面将要讲到的符号的元-含义. 正则表达式其实是由普通字符和元字符共同组成的集合, 这个集合用来匹配 (或指定) 模式.

一个正则表达式会包含下列一项或多项:

1. 一个字符集. 这里所指的字符集只包含普通字符, 这些字符只表示它们的字面含义. 正则表达式的最简单形式就是只包含字符集, 而不包含元字符.
2. 锚. 锚指定了正则表达式所要匹配的文本在文本行中的位置. 比如, `^` 和 `$` 就是锚.
3. 修饰符. 它们扩大或缩小 (修改) 了正则表达式匹配文本的范围. 修饰符包含星号, 括号, 和反斜杠.

#### 4.1.1 正则表达式语法

#### 4.1.2 一些实例

### 4.2 awk

如果要格式化报文或从一个大的文本文件中抽取数据包, 那么 `awk` 可以完成这些任务. 它在文本浏览和数据的熟练使用上性能优异. 与其它大多数 `Unix/Linux` 命令不同的是, 从名字上看, 我们一眼看不出 `awk` 的功能: 它既不是具有独立意义的英文单词, 也不是几个相关单词的缩写. 事实上, `awk` 是三个人名的缩写, 他们是: Aho、(Peter) Weinberg 和 (Brain)Kernighan. 正是这三个家伙创造了 `awk`, 一个优秀的样式扫描与处理工具. 本章只介绍 Gnu 的 `awk` 版本 `gawk`, 其他 `awk` 版本不做介绍.

awk 的工作流程:

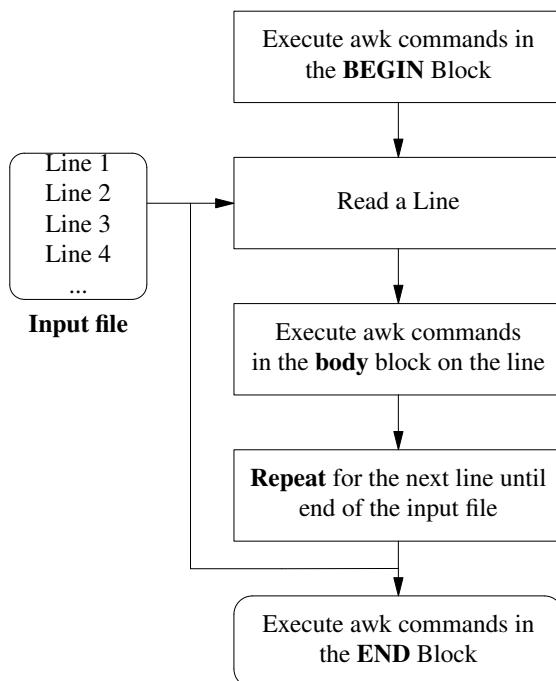


图 4.1 awk 工作流程

### 4.2.1 初步使用

假如我们有文件 `employee.txt`，内容如下：

```
Beth 4.00 0
Dan 3.75 0
Kathy 4.00 10
Mark 5.00 20
Mary 6.50 22
Susie 6.25 18
```

第一列表示员工姓名，第二列表示每小时加班所得的人民币，第三列为加班多少小时。我们现在就来打印出那些加班大于 0 个小时的员工及其加班费，在命令行只需执行这样一行命令即可：

```
# awk '$3 > 0 { print $1, $2 * $3 }' employee.txt
```

我们会得到这样的输出：

```
Kathy 40
Mark 100
Mary 143
Susie 112.5
```

上面的命令告诉系统运行 `awk`，使用单引号里的程序，并从 `employee.txt` 文件里获得自己的数据。单引号里的部分完全就是 `awk` 程序。它包含了单个的 `pattern-action`（模式-动作）语句。动作 `$3 > 0`，匹配输入的每一行的第三列或域，如果该列大于 0，然后就执行下面的动作

```
{ print $1, $2 * $3 }
```

对于每匹配到的行，就会打印第一个域并同时计算第二个域与第三个域的乘积。

如果我们想看看哪个家伙这个月没有加班，可以使用下面的命令：

```
# awk '$3 == 0 { print $1 }' employee.txt
```

这里的模式，`$3 == 0`，将会匹配每一输入行的第三个域，如果等于 0，就会执行下面的动作

```
{ print $1 }
```

打印第一个域。

### 4.2.2 awk 程序结构

`awk` 的基本操作是对输入的行做逐一扫描，搜索这些行中匹配到的任意模式，然后执行相应的动作。然后下一行被读进来并从头开始匹配到行末，直到输入文件的所有行被读进来。就 `$3 > 0` 来说，当满足时，说明该条件为真。

在上面的例子中，只有一个模式和动作，可以称为单 `pattern-action` 语句；对于每一行，匹配到第三个字段等于 0 时，第一个字段将会被打印。

另外，模式和动作并不总是成对出现的，就是说，在 `pattern-action` 语句中，可能只有模式或者只有动作。如果一个 `pattern` 没有相应的 `action`，例如，

```
$3 == 0
```

然后，每一被匹配的行都将会被打印，

```
Beth      4.00      0
Dan       3.75      0
```

如果一个 `action` 没有 `pattern` 时，例如，

```
{ print $1 }
```

此时，将会打印每一行的第一个域。由上可知，`patterns` 和 `actions` 都是可选的。`actions` 使用大括号括起来以与 `patterns` 区分。

### 4.2.3 BEGIN 与 END

特殊的 BEGIN 模式在第一个输入文件的第一行被匹配，而 END 则是在最后一个文件的最后被处理后匹配。下面的程序将会打印一个头部：

```
BEGIN { print "NAME      RATE      HOURS"; print "" }
      { print }
```

我们将会得到这样的输出：

NAME	RATE	HOURS
Beth	4.00	0
Dan	3.75	0
Kathy	4.00	10
Mark	5.00	20
Mary	6.50	22
Susie	6.25	18

我们可以在一行同时输入多条语句，这些语句之间需要用分号分开。注意到，*print* "" 意思是打印一个空行，它与 *print* 不一样，仅仅一个 *print* 语句将打印整行。

### 4.2.4 域和记录

awk 执行时，其浏览域标记为 \$1, \$2,...\$n。这种方法称为域标识。使用这些域标识将更容易对域进一步处理。

### 4.2.5 内置变量

awk 有一些内置的变量，变量如下：

### 4.2.6 One-liners awk 程序

这一节介绍几个很有用的、短小精悍的 awk 程序。

1. 打印输入文件的总行数
2. 打印指定的行
3. 打印输入行的最后一个字段
4. 打印最后一行的最后一个字段
5. 打印每一输入行并擦除第二个字段

### 4.2.7 条件和循环

这一节包含了一些基本的编程结构。它覆盖了 awk 程序设计语言中的所有控制结构。在 awk 中，这些条件和循环的语法用起来比较简单。实际上，awk 中的条件和循环结构的语法借鉴 C 语言。因此，通过学习 awk，我们也同样在学习 C 语言。

变量	说明
ARGC	命令行中参数的个数
ARGV	包含命令行参数的数组
CONVFMT	用于数字的字符串转换格式（默认格式为%.6g）
ENVIRON	环境变量的关联数组
FILENAME	当前文件名
NR	当前记录的个数（当前的行号）
FNR	当前记录的个数（仅为当前文件，而非全部输入文件）
FS	字段分割符（默认为空格）
NF	当前记录中的字段个数
OFMT	数字的输出格式（默认格式为%.6g）
OFS	输出字符的分割符（默认为空格）
ORS	输出记录分割符（默认为换行）

### 4.2.8 if-else 语句

awk 提供一个 *if-else* 语句，这些语句只能用在 **actions** 中。下面的这个程序使用的还是??节的例子。在这个例子中，我们会计算那些每小时加班费大于 6 块的家伙，他们的总加班费及平均加班费。

```
# cat progfile
$2 > 6 { n = n + 1; pay = pay + $2 * $3 }
END    { if (n > 0)
        print n, "employees, total pay is", pay,
            "average pay is", pay / n
        else
        print "no employees are paid more than Y6/hour"
    }

# awk -f progfile employee.txt
```

我们将会得到下面的输出结果：

```
2 employees, total pay is 255.5 average pay is 127.75
```

### 4.2.9 while 循环

**while** 语句有一个 **condition** 和一个 **body**。当 **condition** 为真时，**body** 中的语句将被重复执行。我们使用下面的程序计算这样的一个公式  $value = amount(1 + rate)^{years}$

```
# cat interest1
# interest1 - compute compound interest
#    formula: value = amount(1+rate)^years
```

```
# input: amount rate years
# output: compounded value at the end of each year

{ i = 1
  while (i <= $3) {
    printf("\t%.2f\n", $1 * (1 + $2) ^ i)
    i = i + 1
  }
}
```

如何执行该程序？看操作，

```
# awk -f interest1 <- 回车
1000 .06 5 <- 输入三组数字后，回车
1060.00
1123.60
1191.02
1262.48
1338.23
```

#### 4.2.10 for 循环

我们使用 for 语句改写第4.2.9节的 while 程序。如下：

```
# cat interest2
# interest2 - compute compound interest
# formula: value = amount(1+rate)^years
# input: amount rate years
# output: compounded value at the end of each year

{ for (i = 1; i <= $3; i = i + 1)
  printf("\t%.2f\n", $1 * (1 + $2) ^ i)
}
```

#### 4.2.11 do 循环

#### 4.2.12 影响流控制的语句

#### 4.2.13 数组

数组是可以用来存储一组数据的变量。通常这些数据之间具有某种关系。数组中的每一个元素通过它们在数组中的下标来访问。每个下标用方括号括起来。下面的语句表示为数组中的一个元素赋值。

```
array[subscript] = value
```



在 `awk` 中不必指明数组的大小，只需要为数组指定标示符。向数组元素赋值比较容易。例如，下面的例子中为数组 `hr` 的一个元素指定了一个字符串 “laven”。

```
hr[1] = "laven"
```

这个数组元素的下标是 “1”，下面的语句将打印字符串 “laven”。

```
print hr[1]
```

当然，我们可以用循环向数组中写入或取出元素。例如，如果数组 `hr` 有 10 个元素，可以使用下面的循环来打印每个元素：

```
hr_count = 10
for (x = 1; x <= hr_count; ++x)
print hr[x]
```

我们来看一个例子，依然使用第??节中的输入文件，让 `employee.txt` 文件反序输出，

```
# cat reverse
# reverse - print input in reverse order by line
{ line[NR] = $0 } # remember each input line
END { i = NR      # print lines in reverse order
      while (i > 0) {
        print line[i]
        i = i - 1
      }
}
```

```
# awk -f reverse employee.txt
Susie    6.25    18
Mary     6.50    22
Mark     5.00    20
Kathy    4.00    10
Dan       3.75    0
Beth     4.00    0
```

#### 4.2.14 表达式

```
# cat awk_file
gold      1      1986  USA      American Eagle
gold      1      1908  Austria-Hungary  Franz Josef 100 Korona
silver    10      1981  USA      ingot
gold      1      1984  Switzerland  ingot
gold      1      1979  RSA      Krugerrand
```

gold	0.5	1981	RSA	Krugerrand
gold	0.1	1986	PRC	Panda
silver	1	1986	USA	Liberty dollar
gold	0.25	1986	USA	Liberty 5-dollar piece
silver	0.5	1986	USA	Liberty 50-cent piece
silver	1	1987	USA	Constitution dollar
gold	0.25	1987	USA	Constitution 5-dollar piece
gold	1	1988	Canada	Maple Leaf

```
# This is an awk program that summarizes a coin collection.
#
/gold/      { num_gold++; wt_gold += $2 }      # Get weight of gold.
/silver/    { num_silver++; wt_silver += $2 }# Get weight of silver.
END { val_gold = 485 * wt_gold;                # Compute value of gold.
      val_silver = 16 * wt_silver;              # Compute value of silver.
      total = val_gold + val_silver;
      print "Summary data for coin collection:"; # Print results.
      printf ("\n");
      printf ("    Gold pieces:                %2d\n", num_gold);
      printf ("    Weight of gold pieces:        %5.2f\n", wt_gold);
      printf ("    Value of gold pieces:            %7.2f\n", val_gold);
      printf ("\n");
      printf ("    Silver pieces:                %2d\n", num_silver);
      printf ("    Weight of silver pieces:            %5.2f\n", wt_silver);
      printf ("    Value of silver pieces:            %7.2f\n", val_silver);
      printf ("\n");
      printf ("    Total number of pieces:            %2d\n", NR);
      printf ("    Value of collection:                %7.2f\n", total);
}
```

```
[root@unionpay bash]# awk -f myawk awk_file
Summary data for coin collection:
```

Gold pieces:	9
Weight of gold pieces:	6.10
Value of gold pieces:	2958.50

Silver pieces:	4
Weight of silver pieces:	12.50
Value of silver pieces:	200.00

Total number of pieces:	13
-------------------------	----

Value of collection: 3158.50

#### 4.2.15 函数

#### 4.2.16 总结

### 4.3 sed

Sed 是一个超级流式编辑器。picture a stream flowing through a pipe. Okay, you can't see a stream if it's inside a pipe. That's what I get for attempting a flowing analogy. You want literature, read James Joyce.

Anyhow, sed is a marvelous utility. Unfortunately, most people never learn its real power. The language is very simple, but the documentation is terrible. The Solaris on-line manual pages for sed are five pages long, and two of those pages describe the 34 different errors you can get. A program that spends as much space documenting the errors than it does documenting the language has a serious learning curve.

Do not fret! It is not your fault you don't understand sed. I will cover sed completely. But I will describe the features in the order that I learned them. I didn't learn everything at once. You don't need to either.

sed 的工作流程:

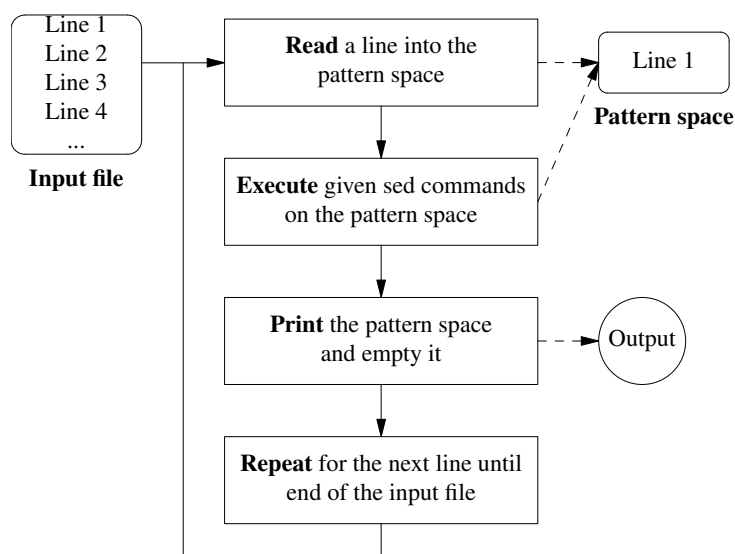


图 4.2 sed 工作流程

命令列表:

命令	说明
a	在当前行后面追加文本
b label	分支到脚本中带有标号的地方, 如果标号不存在就分支到脚本的末尾
c	用新的文本改变或者替代本行的文本
d	从模板块 (Pattern space) 位置删除行
D	删除模板块的第一行
i	在当前行上面插入文本
h	拷贝模板块的内容到内存中的缓冲区
H	追加模板块的内容到内存中的缓冲区
g	获得内存缓冲区的内容, 并替代当前模板块中的文本
G	获得内存缓冲区的内容, 并追加到当前模板块文本的后面
l	列表不能打印字符的清单
n	读取下一个输入行, 用下一个命令处理新的行而不是用第一个命令
N	追加下一个输入行到模板块后面并在二者之间嵌入一个新的行, 改变当前行的号码
p	打印模板块的行
P	打印模板块的第一行
q	退出 sed
r file	从 file 中读行
t label	if 分支, 从最后一行开始, 条件一旦被满足或者 T 命令或者 t 命令, 将导致分支到带有标号的命令处, 或者到脚本的末尾
T label	错误分支, 从最后一行开始, 一旦发生错误或者 T 命令或者 t 命令, 将导致分支到带有标号的命令处, 或者到脚本的末尾
w file	写并追模板块到 file 末尾
W file	写并追模板块的第一行到 file 末尾
!	表示后面的命令对所有没有被选定的行发生作用
s/re/string/	用 string 替换正则表达式 re
=	打印当前行号码
注释 command	把注释扩展到下一个换行符以前替换标记
g	行内全面替换
p	打印行
w	把行写入一个文件
x	互换模板块中的文本和缓冲区中的文本
y	把一个字符翻译为另外的字符 (但是不能用于正则表达式)

一些选项:

-e command	允许多点编辑
-expression=command	同上
-h,-help	打印命令行选项摘要, 并显示 bug 列表的地址
-n,-quiet,-silent	取消默认输出
-f,	引导 sed 脚本文件名
-filr=script-file	同上
-V,-version	打印版本和版权信息

### 4.3.1 匹配

### 4.3.2 变量定义

### 4.3.3 特殊变量

### 4.3.4 数组

### 4.3.5 删除

删除 (d) 编辑命令采用一个地址, 如果行匹配这个地址就删除模式空间的内容。删除命令 (d) 还是一个可以改变脚本中的控制流的命令。这是因为一旦执行这个命令, 那么在“空的”模式空间中就不会再有命令执行。删除命令 (d) 会导致读取新输入行, 而编辑脚本则从头开始新一轮。重要的是, 如果某行匹配这个地址, 那么就删除整个行, 而不只是删除行中匹配的部分。

### 4.3.6 替换

Sed 有众多命令, 但经常被用到的就是最基本的替换命令“s”了, 替换命令有四部分组成, 如下表所示:

s	所使用的命令是替换命令
/././	所使用的分隔符, 可以自行改变, 不过三个符号要一致
old	这是将要被替换的字符串或正则表达式
new	这是替换后的字符串

```
sed s/day/night/ <old>new
```

```
eg: sed 's/local/host/[g]' /etc/hosts
```

Or another way (for Unix beginners),

```
sed s/day/night/ old>new
```

我们也可以如下这样写,

```
echo day | sed s/day/night/
```

将会输出“night”。

我们在这里并没有使用引号把这些参数给引用起来, 也能得出想要的结果, 因为这个例子是不需要它们的, 嘿嘿! 然而, 在使用过程中出现了原字符, 这时候引号是必须

要的。如果我们不确定何时要使用引号，那么，最好的习惯就是“每用必带”引号即可，不用纠结那么多。

```
sed 's/day/night/' <old >new
```

I must emphasize that the sed editor changes exactly what you tell it to. So if you executed

```
eg: # echo Sunday | sed 's/day/night/'
```

```
output: Sunnight
```

This would output the word "Sunnight" because sed found the string "day" in the input.

The search pattern is on the left hand side and the replacement string is on the right hand side.

We've covered quoting and regular expressions. That's 90% of the effort needed to learn the substitute command. To put it another way, you already know how to handle 90% of the most frequent uses of sed. There are a ... few fine points that an future sed expert should know about. (You just finished section 1. There's only 63 more sections to cover. :-) Oh. And you may want to bookmark this page, .... just in case you don't finish.

#### 4.3.7 追加、插入和更改

追加 (a)、更改 (c)、插入 (i) 编辑命令提供了类似于 vi 交互式编辑器的编辑功能。

追加命令 (a) 将文本放置在当前行之后。更改命令 (c) 用所指定的文本取代模式空间的内容。插入命令 (i) 将所提供的文本放置在模式空间的当前行之前。这些命令中的每一个都要求后面跟一个反斜杠用于转义第一个行尾。如要输入多行文本，每个连续的行都必须用反斜杠结束，最后一行除外。而且，如果文本包含一个字面含义的反斜杠，要再添加一个反斜杠来转义它。

#### 4.3.8 模式空间和保留空间

#### 4.3.9 流控制

#### 4.3.10 地址范围

#### 4.3.11 调用外部变量

#### 4.3.12 总结

### 4.4 语法介绍

#### 4.4.1 变量定义

#### 4.4.2 特殊变量

在脚本里有以下几个特殊的变量，如下

表 4-1 特殊变量的含义

变量	说明
\$0	代表该脚本的名字
\$n	代表该程序的第 $n$ 个参数值, $n=1..9$
\$*	代表该脚本的所有参数
\$#	代表该脚本的参数个数
\$\$	代表该脚本的 PID
#!	代表上一个指令的 PID
\$?	代表上一个指令的返回值

### 4.4.3 变量赋值和替换

### 4.4.4 本地变量与全局变量

### 4.4.5 引用变量

### 4.4.6 数组

Bash 只支持一维数组, 但参数个数没有限制。

数组赋值:

```
1. array=(var1 var2 var3 ... varN)
2. array=( [0]=var1 [1]=var2 [2]=var3 ... [n]=varN)
3. array[0]=var1
   array[1]=var2
   array[2]=var3
   ...
   array[N]=varN
```

declare -

```
MYARRAY=( [0]=tom [1]=laven [2]=liu [5]=jim)
```

数组的元素的长度:

`${#MYARRAY}` 指的是数组 MYARRAY 的第 0 个元素的长度, 与  
`${#MYARRAY[0]}` 等价  
`${#MYARRAY[n]}` 是第  $n+1$  个元素,

数组的元素个数:

```
${#MYARRAY[*]}
${#MYARRAY[@]}
```

一个例子, 随机生成 10 个整型元素的数组, 并找出该数组中的最大值。

---

```
1  #!/bin/bash
2  #
3  for i in {0..9}
4  do
5      ARRAY[$i]=$RANDOM
6      echo -n "${ARRAY[$i]} "
7      sleep 1
8  done
9
10 echo
11
12 declare -i MAX=${ARRAY[0]}
13 INDEX=$(( ${#ARRAY[*]} - 1 ))
14 for i in `seq 1 $INDEX`
15 do
16     if [ $MAX -lt ${ARRAY[$i]} ] ; then
17         MAX=${ARRAY[$i]}
18     fi
19 done
20
21 echo ${MAX}
```

---

#### 4.4.7 特殊字符

### 4.5 基本流程

Bash 与其他编程语言一样，也有自己的程序处理逻辑。接下来的这个章节主要介绍 Bash 的脚本几种基本流程。

#### 4.5.1 if 结构

先看一下 bash 的 man page 是如何定义 if 结构的，

```
if list; then
    list;
[ elif list; then list; ]
...
[ else list; ]
fi
```

当 *if list* 执行成功并且返回状态是 0 时，相应的 *then list* 就会被执行；否则，*elif list* 执行，且返回状态为 0 时，相应的 *then list* 就会被执行；之后命令执行结束。如果前面的 *if list* 及 *elif list* 都不能成功执行，那么将执行最后一个 *else list* 语句。返回状态就是上一



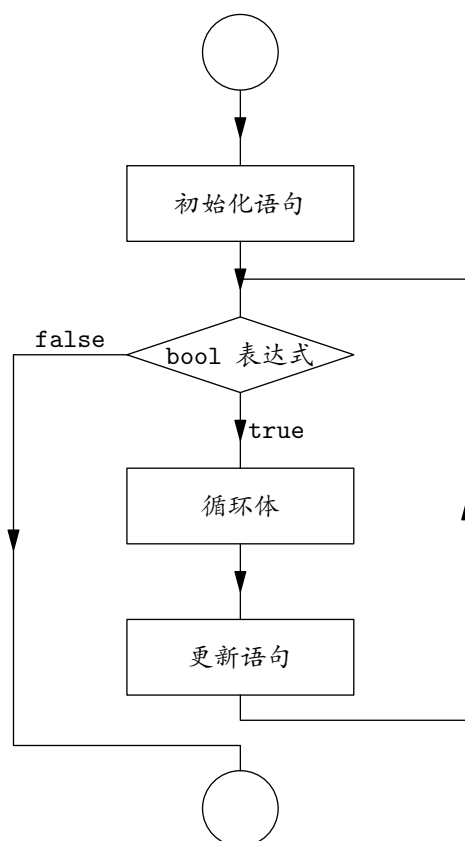


图 4.3 for 工作流程

条命令执行成功与否，执行成功就返回 0，不成功返回非 0。不成功的原因有很多，成功返回就一个。

一个示例：

src/shell01.sh

```

1 #!/bin/bash
2 #: Title      : xxxx
3 #: Date       : 2012-10-12
4 #: Author    : "Laven Liu" ldczz2008@sina.com
5 #: Version    : 1.0
6 #: Description : get the maximum number
7 #: Options    : None
8
9 NUM=1
10
11 if [ $NUM -gt 0 ]; then
12     echo "NUM is great than 0"
13 else
14     echo "NUM is less than 0"
15 fi
  
```

## 4.5.2 for 结构

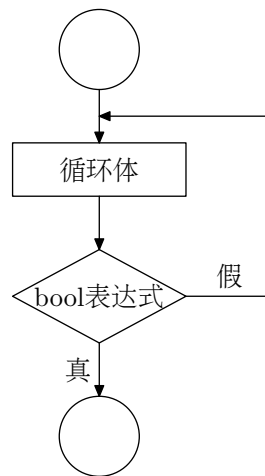


图 4.4 do 工作流程

---

```

1 #!/bin/sh
2
3 # numberlines - A simple alternative to cat -n, etc.
4
5 for filename
6 do
7     linecount="1"
8     while read line
9     do
10         echo "${linecount}: $line"
11         linecount="$((linecount + 1))"
12     done < $filename
13 done
14 exit 0
  
```

---

### 4.5.3 while 结构

---

src/shell03.sh

---

```

1 #!/bin/bash
2
3 # Written By: LavenLiu
4 # Date: 2014-02-13
5 # Mail: air.man.six@gmail.com
6
7 # 把服务器IP地址的最后一个字段存放到一个数组中
8 server_list=(90 110 225 231 233 235 249 251 252)
9
10 # 计算服务器数量
11 how_many=$(( ${#server_list[@]} - 1 ))
12
  
```

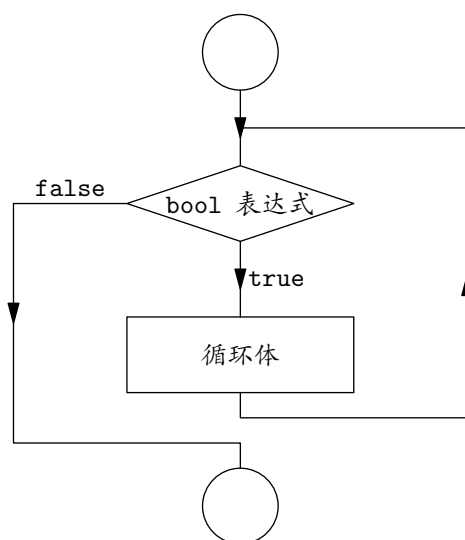


图 4.5 while 工作流程

```

13 myloop=0
14
15 while [ ${myloop} -lt ${how_many} ] ;
16 do
17   ping -c1 192.168.18.${server_list[${myloop}]} &> /dev/null
18   if [ "$?" -eq 0 ] ; then
19     echo "Server 192.168.18.${server_list[${myloop}]} is alive"
20   else
21     echo "Server 18.${server_list[${myloop}]} is down! Contact the
22       administrator..."
23   fi
24   let "myloop = ${myloop} + 1"
25 done

```

我们可以将 while 循环和专用命令 “:” 结合使用来定义。由于循环固有的一个特性，当条件永远不被满足时，就会产生一个无限循环。定义一个 while 无限循环可以使用如下三种命令：

1. true 命令 不做任何事情，表示成功，总是返回退出状态码 0
2. false 命令 不做任何事情，表示失败，总是返回退出状态码 1
3. : 命令 无作用，此命令也不做任何事情，总是返回退出状态码 0

## 4.6 操作字符串

### 4.7 函数

当我们的脚本大到一定程度时，使用函数则可以简化脚本，使程序结构更为清晰。

## 4.8 信号捕捉

trap 可以捕捉信号，根据捕捉到的信号，执行响应的操作。

语法：

```
trap 'action' SIGNAL
```

## 4.9 开机脚本启动顺序

### 4.10 一个实例

src/shell04.sh

```
1 #!/bin/bash
2 # Written By: LavenLiu
3 # Date: 2013-01-08
4 def_colors () {
5     # Attributes
6     normal='\033[0m';bold='\033[1m';dim='\033[2m';under='\033[4m'
7     italic='\033[3m'; notalic='\033[23m'; blink='\033[5m';
8     reverse='\033[7m'; conceal='\033[8m'; nobold='\033[22m';
9     nounder='\033[24m'; noblink='\033[25m'
10
11     # Foreground
12     black='\033[30m'; red='\033[31m'; green='\033[32m'; yellow='\033[33
13     m'
14     blue='\033[34m'; magenta='\033[35m'; cyan='\033[36m'; white='\
15     \033[37m'
16
17     # Background
18     bblack='\033[40m'; bred='\033[41m'
19     bgreen='\033[42m'; byellow='\033[43m'
20     bblue='\033[44m'; bmagenta='\033[45m'
21     bcyan='\033[46m'; bwhite='\033[47m'
22 }
23
24 def_colors
25 clear
26 hostname=`cat /proc/sys/kernel/hostname`
27 echo
28 echo -e "System Report for $white$hostname$normal on `date`"
29 echo
30 processor=`grep 'model name' /proc/cpuinfo | cut -d: -f2 | cut -c2-`
31 nisdomain=`cat /proc/sys/kernel/domainname`
32 cache=`grep 'cache size' /proc/cpuinfo | awk '{print $4,$5}'`
33 bogomips=`grep 'bogomips' /proc/cpuinfo | awk '{print $3}'`
34 vendor=`grep 'vendor_id' /proc/cpuinfo`
```

```

33 echo -e "Hostname: $white$hostname$normal NIS Domain:
    $white$nisdomain$normal"
34 if [ "`echo $vedner | grep -i intel`" ]
35 then
36     cpu_color=$blue
37 elif [ "`echo $vender | grep -i amd`" ]
38 then
39     cpu_color=$green
40 fi
41
42 echo -e "Processor: $cpu_color$processor$normal"
43 echo -e "Running at $white$bogomips$normal bogomips with \
44 $white$cache$normal cache"
45 echo
46 osrelease=`cat /proc/sys/kernel/osrelease`
47 rev=`cat /proc/sys/kernel/version | awk '{print $1}'`
48 da_date=`cat /proc/sys/kernel/version | cut -d\ -f2-`
49 upsec=`awk '{print $1}' /proc/uptime`
50 uptime=`echo "scale=2;$upsec/86400" | bc`
51 echo -e "OS Type: $white$ostype$normal Kernel: \
52 $white$osrelease$normal"
53 echo -e "Kernel Compile $white$rev$normal on \
54 $white$da_date$normal"
55 echo -e "Uptime: $magenta$uptime$normal days"
56 set `grep MemTotal /proc/meminfo`
57 tot_mem=$2; tot_mem_unit=$3
58 set `grep MemFree /proc/meminfo`
59 free_mem=$2; fre_mem_unit=$3
60 perc_mem_used=$((100-(100*free_mem/tot_mem)))
61 set `grep SwapTotal /proc/meminfo`
62 tot_swap=$2; tot_swap_unit=$3
63 perc_swap_used=$((100-(100*free_swap/tot_swap)))
64 if [ $perc_mem_used -lt 80 ]
65 then
66     mem_color=$green
67 elif [ $perc_mem_used -ge 80 -a $perc_mem_used -lt 90 ]
68 then
69     mem_color=$yellow
70 else
71     mem_color=$red
72 fi
73 if [ $perc_swap_used -lt 80 ]
74 then
75     swap_color=$green
76 elif [ $perc_swap_used -ge 80 -a $perc_swap_used -lt 90 ]
77 then

```

```

78     swap_color=$yellow
79 else
80     swap_color=$red
81 fi
82
83 echo -e "Memory: $white$tot_mem$normal $tot_mem_unit Free:
      $white$free_mem$normal \
84 $fre_mem_unit $Used: $mem_color$perc_mem_used$normal"
85 echo -e "Swap: $white$tot_swap$normal $tot_swap_unit Free:
      $white$free_swap$normal \
86 $fre_swap_unit $Used: $swap_color$perc_swap_used$normal"
87 echo
88 set `cat /proc/loadavg`
89 one_min=$1
90 five_min=$2
91 fifteen_min=$3
92 echo -n "Load Avderage:"
93 for ave in $one_min $five_min $fifteen_min
94 do
95     int_ave=`echo $ave | cut -d. -f1`
96     if [ $int_ave -lt 1 ] ; then
97         echo -en "$green$ave$normal "
98     elif [ $int_ave -ge 1 -a $int_ave -lt 5 ] ; then
99         echo -en "$yellow$ave$normal "
100     else
101         echo -en "$red$ave$normal"
102     fi
103 done
104 echo
105 running=0; sleeping=0; stopped=0; zombie=0
106 for pid in /proc/[1-9]*
107 do
108     procs=$((procs+1))
109     stat=`awk '{print $3}' $pid/stat`
110     case $stat in
111         R) running=$((running+1));;
112         S) sleeping=$((sleeping+1));;
113         T) stopped=$((stopped+1));;
114         Z) zombie=$((zombie+1));;
115     esac
116 done
117 echo -n "Process Count: "
118 echo -e "$white$process$normal total $white$running$normal running \
119 $white$sleep$normal sleeping $white$stopped$normal stopped \
120 $white$zombie$normal zombie"
121 echo

```

---

下面是该脚本的输出结果:

---

```
System Report for richard on Tue Sep 23 21:34:22 CST 2014

Hostname: richard NIS Domain: (none)
Processor:
Running at 4585.16
4585.16
4585.16
4585.16 bogomips with 3072 KB
3072 KB
3072 KB
3072 KB cache

OS Type: Linux Kernel:3.13.0-35-generic
Kernel Compile #62-Ubuntu on
Uptime: .08 days
Memory: 6003456 kB Free:    : 100
Swap: 6180860 kB Free:    : 100

Load Average:0.63 0.69 0.61
Process Count:  total 1 running sleeping 0 stopped 0 zombie
```

---





## 第二部分

### 数据库篇



## 第五章 MySQL 数据库基本知识

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 公司。MySQL 是一个命途多舛的产品，2008 年 2 月 26 日被 Sun 完全收购，2009 年 4 月 20 日 Oracle 并购了 Sun 公司。业界一片哗然，一时众说纷纭，褒贬不一。在此过程中衍生出了两个版本的数据库，一个是 Percona Server 及 MariaDB，这些版本的衍生，是考虑到 Oracle 将其闭源的潜在可能。

关系型数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权策略，它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为其网站的数据库。由于其社区版的性能卓越，搭配 PHP 和 Apache 可组成良好的开发环境。

### 5.1 存储设备配置

#### 5.1.1 构建文件系统

使用 xfs 文件系统来提供 MySQL 数据的存储。使用 xfs 的话，需要添加管理配置工具包，如下：

```
A0305010:~ # zypper -y install xfsdump xfsprogs
A0305010:~ # modprobe -v xfs
insmod /lib/modules/3.0.13-0.27-default/kernel/fs/xfs/xfs.ko
```

使用 mkfs.xfs 创建文件系统如下：

```
A0305010:~ # fdisk /dev/sdb
Command (m for help): n
Command action
  e   extended
  p   primary partition (1-4)
p
Partition number (1-4, default 1):
Using default value 1
First sector (2048-121634815, default 2048):
Using default value 2048
```

```
Last sector, +sectors or +size{K,M,G} (2048-121634815, default 121634815):
Using default value 121634815
```

```
Command (m for help): w
The partition table has been altered!
```

```
Calling ioctl() to re-read partition table.
Syncing disks.
```

```
A0305010:~ # partprobe /dev/sdb
A0305010:~ # mkfs.xfs -f -i size=512,attr=2 -l lazy-count=1 \
> -d su=64k,sw=10 -L /data1 /dev/sdb1
```

注：这里的 `sdb` 是我们使用的一个样例。不同 SSD 厂商的设备在系统里被识别的标识不一定是 `/dev/sdb`，根据实际情况选择合适的标示符（如，Fusion-io 的设备在本次测试中被识别为 `/dev/fioa`；LSI 的设备在本次测试中被识别为 `/dev/sdb`；Intel 的设备在本次测试中被识别为 `/dev/nvme0n1`；宝存的设备在本次测试中被识别为 `/dev/dfa`）。

### 5.1.2 挂载文件系统

```
A0305010:~ # mkdir /data1
A0305010:~ # cd /data1
A0305010:~ # mkdir mysqldata1 mysqldata2 mysqldata3 mysqldata4
A0305010:~ # mount -t xfs -o defaults,rw,noatime,nodiratime,\
> noikeep,nobarrier,allocsize=8M,attr2,largeio,\
> inode64,swalloc /dev/sdb1 /data1
```

在 `/etc/fstab` 中添加自动挂载参数如下：

```
/dev/sdb1 /data1 xfs defaults,rw,noatime,nodiratime,noikeep,nobarrier,allocsize=8M,attr2,largeio,inode64,swalloc 0 0
A0305010:~ # mount -a
```

## 5.2 安装 MySQL

### 5.2.1 创建 MySQL 用户

我们需要添加专门的 MySQL 用户，并设置它的 `uid` 及 `gid` 为 601。

```
[root@iLiuc ~]# groupadd -g 601 mysql
[root@iLiuc ~]# useradd -c "mysql software owner" -g mysql \
> -u 601 -m -d /home/mysql mysql
```

### 5.2.2 安装 MySQL

采用二进制方案安装，步骤如下：

1. 首先获取与生产环境数据库相同的二进制安装文件

```
A0305010:~ # mkdir /home/mysql/program
A0305010:~ # cd /home/mysql/program
A0305010:~ # ls
p16901968_55_Linux-x86-64.zip
```

## 2. 解压

```
A0305010:~ # unzip p16901968_55_Linux-x86-64.zip
A0305010:~ # ls
README.txt
keepalived-1.2.7-8.1.x86_64.rpm
mysql-advanced-5.5.32-linux2.6-x86_64.tar.gz
mysql-advanced-5.5.32-linux2.6-x86_64.tar.gz.asc
mysql-advanced-5.5.32-linux2.6-x86_64.tar.gz.md5
p16901968_55_Linux-x86-64.zip
A0305010:~ # tar -xf mysql-advanced-5.5.32-linux2.6-x86_64.tar.gz
```

## 3. 创建数据目录

```
A0305010:~ # mkdir /home/mysql/data
A0305010:~ # cd /home/mysql/data
A0305010:~ # mkdir conf innodb_log innodb_ts log mydata \
> relaylog slowlog sock tmpdir undo binlog
```

## 4. 初始化 MySQL 元数据

```
A0305010:~ # cd /home/mysql/program/mysql-advanced-5.5.32-linux2.6-x86_64
A0305010:~ # ./scripts/mysql_install_db \
> --defaults-file=/home/mysql/conf/my1.cnf \
> --user=mysql \
> --skip-name-resolve
```

## 5. 链接到/usr/local/mysql

```
A0305010:~ # ln -s /home/mysql/program/mysql-advanced-5.5.32-linux2.6-x86_64
> /usr/local/mysql
A0305010:~ # ln -s /data1/mysqldata1 /home/mysql/data/mysqldata1
A0305010:~ # ln -s /data1/mysqldata2 /home/mysql/data/mysqldata2
A0305010:~ # ln -s /data1/mysqldata3 /home/mysql/data/mysqldata3
A0305010:~ # ln -s /data1/mysqldata4 /home/mysql/data/mysqldata4
```

```
[root@iLiuc ~]# groupadd -g 601 mysql
[root@iLiuc ~]# useradd -c "mysql software owner" -g mysql \
> -u 601 -m -d /home/mysql mysql
```

## 5.3 操作系统配置

### 5.3.1 进程文件数配置

由于 MySQL 采用多线程形式，每个进程打开的文件数要求比较高。所以我们需要增加 `ulimit` 进程文件数的限制。在 `/etc/rc.local` 中添加：

```
A0305010:~ # /etc/init.d/sshd stop
A0305010:~ # ulimit -HSn 65535
```

把上述配置写到配置文件 `/etc/security/limits.conf`，找到 “# End of file”，在其上添加 3 行

```
A0305010:~ # vi /etc/security/limits.conf
* hard memlock unlimited
* soft memlock unlimited
* -nofile 65535
A0305010:~ # /etc/init.d/sshd start
```

### 5.3.2 sysctl 配置

修改 `sysctl`，添加 `swappiness` 并设置 `TIME_WAIT` 的相关参数。避免内存被交换并且减少 `TIME_WAIT` 状态时间。

```
A0305010:~ # echo "vm.swappiness = 0" >>/etc/sysctl.conf
A0305010:~ # echo "net.ipv4.tcp_tw_reuse=1" >>/etc/sysctl.conf
A0305010:~ # echo "net.ipv4.tcp_tw_recycle=1" >>/etc/sysctl.conf
A0305010:~ # echo "net.ipv4.tcp_fin_timeout = 30" >>/etc/sysctl.conf
A0305010:~ # sysctl -p
```

### 5.3.3 cgroup 配置

我们只对 `cpu` 和内存做限制，所以只需要挂载两个子系统。另外，我们在本服务器上计划启动 4 个实例，这四个实例分别属于四个组，`mysql_g1`，`mysql_g2`，`mysql_g3`，`mysql_g4`。目前我们限制它们都在 4 个超线程并且每个实例的内存不超过 24G。如下：

```
A0305010:~ # zypper install -y libcgroupl
A0305010:~ # cat /etc/cgconfig.conf
mount {
    cpuset    = /cgroup/cpuset;
    memory    = /cgroup/memory;
}
group mysql_g1 {
    cpuset {
        cpuset.cpus = 17-20;
```

```
        cpuset.mems = 0;
    }
    memory {
        memory.limit_in_bytes=25769803776;
        memory.swappiness=0;
    }
}
group mysql_g2 {
    cpuset {
        cpuset.cpus = 12-16;
        cpuset.mems = 0;
    }
    memory {
        memory.limit_in_bytes=25769803776;
        memory.swappiness=0;
    }
}
group mysql_g3 {
    cpuset {
        cpuset.cpus = 5-8;
        cpuset.mems = 0;
    }
    memory {
        memory.limit_in_bytes=25769803776;
        memory.swappiness=0;
    }
}
group mysql_g4 {
    cpuset {
        cpuset.cpus = 1-4;
        cpuset.mems = 0;
    }
    memory {
        memory.limit_in_bytes=25769803776;
        memory.swappiness=0;
    }
}
```

```
A0305010:~ # /etc/init.d/cgconfig start
```

### 5.3.4 cgrule 配置

我们设置使用 `mysqld_safe1` 启动的命令为 `mysql_g1` 组，这样就可以限制它的 CPU 和内存使用。其他的四个实例同样进行设置如下：

```
A0305010:~ # cat /etc/cgrules.conf
*:/usr/local/mysql/bin/mysqld_safe1      *      mysql_g1
*:/usr/local/mysql/bin/mysqld_safe2      *      mysql_g2
*:/usr/local/mysql/bin/mysqld_safe3      *      mysql_g3
*:/usr/local/mysql/bin/mysqld_safe4      *      mysql_g4

A0305010:~ # /etc/init.d/cgred start
```

## 5.4 MySQL 配置

### 5.4.1 程序文件和目录配置

### 5.4.2 多实例配置

#### 1. my.cnf 文件配置

我们在 `/home/mysql/conf` 下保存着 `my1.cnf`, `my2.cnf`, `my3.cnf`, `my4.cnf`。针对每一个数据库实例都有一个对应的配置文件。各个配置文件中都保存的是每个实例独立的配置。通过“实例切换配置”来切换到各个不同的实例。

#### 2. 实例切换配置

我们在 `/home/mysql/bin` 下存放有四个文件 `my1`, `my2`, `my3`, `my4`。用于切换到各个实例的配置环境中。并且我们在 `profile` 中添加了对应的别名：

```
alias my1=". /home/mysql/bin/my1"
alias my2=". /home/mysql/bin/my2"
alias my3=". /home/mysql/bin/my3"
alias my4=". /home/mysql/bin/my4"
```

如，`my1` 文件内容如下：

```
#!/bin/sh
#*****#
# ScriptName: my1
#           switch to mysql instance 1 environment
# Author:
# Create Date: 2013-03-27
# Modify Author:
# Modify Date: 2013-03-27
# Function:
#*****#
```



```
#set -x

#### variable should change every instance
export INSTANCE_NO=1

#### variable should change every instance
export MYSQL_PORT=$(expr 3305 + $INSTANCE_NO)
export MYSQL_PROGRAM=/usr/local/mysql
export DATA_DIR=/home/mysql/data/mysqldata$INSTANCE_NO/
export MYSQL_CONF=/home/mysql/conf/my$INSTANCE_NO.cnf
export PS1="\n\e[1;37m[\e[m\e[1;36mMySQL$INSTANCE_NO\e[m \$(date \"+%Y-%m-%d
alias stopmysql="$MYSQL_PROGRAM/bin/mysqladmin --defaults-file=$MYSQL_CONF -
alias my="$MYSQL_PROGRAM/bin/mysql --defaults-file=$MYSQL_CONF -uroot -p1111

#### variable should not change every version
export PATH=$MYSQL_PROGRAM/bin/:$PATH
export LD_LIBRARY_PATH=$MYSQL_PROGRAM/lib/
export PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME%%.*}"; echo -ne "

startmysql()
{
cd $MYSQL_PROGRAM/
$MYSQL_PROGRAM/bin/mysqld_safe$INSTANCE_NO --defaults-file=${MYSQL_CONF} & c
}

alias cdd="cd $DATA_DIR"
alias cdb="cd $MYSQL_PROGRAM"
alias cdm="cd /home/mysql/"
alias cdqm="cd $QGUARD_BASEDIR"
alias vi="vim"
alias vie="vim $DATA_DIR/log/error.log"
alias cate="cat $DATA_DIR/log/error.log"
alias psm="ps -ef|grep $MYSQL_PROGRAM/bin/mysqld |grep -v grep"
alias ll='ls -l -G'
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

直接输入命令 `my1` 就可以切换到 `mysql` 实例 1 的环境下。此时可以直接输入 `my1` 命令登录第一个实例的 `mysql` 环境，输入 `stopmysql` 就可以停掉 MySQL 数据库，输入 `startmysql` 就可以启动 MySQL 数据库。

另外，还做了一些简单方便的命令，以便管理：

cdd 用于切换到实例的数据目录 cdm 用于切换到 MySQL 根目录 cdb 用于切换到 MySQL 的程序文件目录 psm 用于输出 MySQL 数据库的进程和 cgroup 信息等

## 5.5 数据库日常管理

### 5.5.1 实例切换

### 5.5.2 MySQL 启动

### 5.5.3 MySQL 关闭

## 5.6 如何进入 MySQL 数据库

进入 MySQL 数据库很简单，首先确保 `mysqld` 进程已经启动。如果没有启动，请参照前面的命令启动之。只需要在命令行输入命令 `mysql` 回车即可，因为我们的 IPTV 系统里的 `mysql` 的 `root` 账户是没有密码的。

```
[root@iLiuc ~]# mysql
mysql> show databases;          <- 查看我们有哪些库
+-----+
| Database                |
+-----+
| information_schema       |
| clear_360fy              | <- 稍后我们将导出这个库
| clear_iptv2x             |
| clear_iptv_Skyworth      |
| clear_qingpu_school      |
| clear_vod_mingzhu        |
| clear_vod_new            |
| clear_vod_yiyuan         |
| clear_xianhuashan        |
| cleardb_str              |
| happyview               |
+-----+

mysql> use clear_360fy;          <- 选择要使用的库
Database changed                <- 提示数据库已改变

mysql> show tables;             <- 查看clear_360fy库中有哪些表
+-----+
| Tables_in_clear_360fy    |
+-----+
| config                   |
| directory                |
| drinks_info_t           |
| favorites                 |
```

```

| hotel_category      |
| hotel_info          |
| language            |
| livechannel         |
| log                 |
| ...                 |
| message             |
| program             |
| user_group_matrix   |
| users              |
| weather             |
| wel_message         |
+-----+
mysql> exit           <- 退出数据库
[root@iLiuc ~]#

```

## 5.7 如何建库建表

当我们的数据很少时，把这些信息可以手工的记录在纸片上或其他介质上，管理起来也是很方便的。现如今是信息的时代，我们总不能把数据还是记录在纸上或龟壳上吧，管理起来非把人给累死。把数据存在易于管理的数据库系统中是明智之举。

## 5.8 简单 sql 语句的使用

有了上面的介绍是不是想动手操练一番。首先确保你的系统中应该安装有 `mysql` 数据库软件；其次，要确保 `mysqld` 服务已经在运行；最后，要有数据库的用户名及密码。这里我们使用 `Clear` 的用户名及密码为 `123456` 为演示环境。

### 5.8.1 create 语句的使用

`create` 可以创建库和表，首先我们以一个具体的例子来演示该命令如何使用。接下来我们要创建这样一个库，库名为 `cc` (`clear company`)，表的名字为 `hr` (`human resource`)

id	name	deptname	salary
1	james	development	1000
2	wade	engineer	800
3	bosh	finance	600
4	richard	sale	600
5	laven	it	1010

```

[root@iLiuc ~]# mysql
mysql> create database if not exists cc;

```

```
mysql> use cc;
mysql> create table hr (
    id int not null primary key,
    name char(10) not null,
    deptname char(15) not null,
    salary float(5,2) not null);
```

```
mysql> desc hr;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
name	char(10)	NO		NULL	
deptname	char(15)	NO		NULL	
salary	float(5,2)	NO		NULL	

上面的具体含义，我就不详细讲解了，自己去 Google 上 Baidu 一下吧。上面只是把一个框架给搭建了起来，里面并没有任何数据，我们可以查看一下：

```
mysql> select * from hr;
```

Empty set (0.02 sec) <- 你将会看到这样的提示

### 5.8.2 insert 语句的使用

insert 就是往表里面添加数据的，废话不多说，直接看例子：

```
mysql> insert into hr values
    (1, "james", "development", 1000),
    (2, "wade", "engineer", 800),
    (3, "bosh", "finance", 600),
    (4, "richard", "sale", 600),
    (5, "laven", "it", 1010);
```

```
mysql> select * from hr;
```

id	name	deptname	salary
1	james	development	999.99
2	wade	engineer	800.00
3	bosh	finance	600.00
4	richard	sale	600.00
5	laven	it	999.99

### 5.8.3 select 语句的使用

select 应该是数据库中使用最多的命令了，接下来赶紧介绍如何写 select 语句。

```
[root@iLiuc ~]# mysql -uroot -p123456
mysql> use clear_360fy;
mysql> select * from maincontrol;
...
```

\* 是通配符，表示所有。上面的语句意思是查看 `maincontrol` 表的所有字段。如果只查看某个字段或某几个字段，该怎么办呢？各字段之间用逗号分开即可。每个表有哪些字段呢？使用 `desc` 查看即可。下面看演示：

```
mysql> desc maincontrol;

mysql> select * from hr;
+----+-----+-----+-----+
| id | name   | deptname | salary |
+----+-----+-----+-----+
| 1  | james  | development | 999.99 |
| 2  | wade   | engineer   | 800.00 |
| 3  | bosh   | finance    | 600.00 |
| 4  | richard | sale       | 600.00 |
| 5  | laven  | it         | 999.99 |
+----+-----+-----+-----+
```

#### 5.8.4 update 语句的使用

当有更新的操作时，`update` 命令就派上用场了。当我们忘记信息发布或 IPTV 后台的密码时，也是可用用到 `update` 语句的，接下来我们看看如何使用。

```
[root@iLiuc ~]# mysql
mysql> use clear_360fy;
mysql> show tables; <- 找到users
mysql> desc users; <- 查看users有哪些字段
+-----+
| Field          | 其他列我省略了 |
+-----+
| ID              |                  |
| NAME            | <- 这是用户名   |
| PASSWORD        | <- 这是密码     |
| EMAIL           |                  |
| TYPE            |                  |
| 此处省略若干行... |                  |
| IDCARD          |                  |
| ADDRESS         |                  |
| TEL             |                  |
+-----+

mysql> select name, password from users;
+-----+-----+
```

name	password
master	eb0a191797624dd3a48fa681d3061212
NULL	aaabf0d39951f3e6c3e8a7911df524c2
NULL	4124bc0a9335c27f086f24ba207a4912
newmanager	81dc9bdb52d04dc20036dbd8313ed055
firstmanger	202cb962ac59075b964b07152d234b70
123	202cb962ac59075b964b07152d234b70

这时，我们看到了后台用户及其密码。密码是使用 md5 方式加密的，不安全。我们可以在数据库里查看某个字符串的 md5 值，如：

```
mysql> select md5("123456");
```

md5("123456")
e10adc3949ba59abbe56e057f20f883e

发现没有，123456 的 md5 值跟上面 master 用户密码的 md5 值一样，那说明 maser 用户的密码为 123456。即使我们不知道 master 用户的密码，我们照样可以在数据库里更新它的密码。这里我们把它的密码改为“654321”，看操作：

```
mysql> select md5("654321");
```

md5("654321")
c33367701511b4f6020ec61ded352059

用鼠标左键选择“c33367701511b4f6020ec61ded352059”，别把双引号也给选上了。之后按下鼠标中键即可实现粘贴功能

```
mysql> update users set password="c33367701511b4f6020ec61ded352059"
> where name="master";
```

或者一步到位：

```
mysql> update users set password=(select md5("654321")) where name="master";
```

再次查询，验证是不是已经改变，这时可以使用新的密码登陆后台了。

```
mysql> select name, password from users where name="master";
```

name	password
master	c33367701511b4f6020ec61ded352059

## 5.9 数据库文件的备份

做完一个项目后的第一件事，就是做好现场的备份，包括前台、后台及数据库。首先，这是一个很好的习惯。如果不这样，万一出现问题，又要重新部署，岂不是 x 都碎了。

### 5.9.1 数据库文件的导出

数据库文件的导出，我们这里使用 `mysqldump` 命令。别的不多说，直接介绍该命令如何使用

```
[root@iLiuc ~]# mysqldump -u root -p password db_name > db_name.sql
```

上面这是标准的数据库备份命令，`root` 为数据的用户名，`password` 为 `root` 用户的密码，需要导出的数据库名字为 `db_name`，然后把数据保存到当前目录下的 `db_name.sql` 文件。在我们的 IPTV 系统里，`mysql` 的 `root` 账户是没有密码的，如果我们要备份 `clear_360fy` 的库文件，则输入一下命令即可：

```
[root@iLiuc ~]# mysqldump clear_360fy > /home/clear_360fy.sql
```

这条命令意思是把 `clear_360fy` 的库 dump 到 `/home` 目录下的 `clear_360fy.sql` 文件

### 5.9.2 数据库文件的导入

数据库文件导入的前提是准备要导入的 `sql` 文件，这里我们准备好 `clear_360fy.sql` 文件，然后创建相应的库，再然后把 `sql` 文件导入到刚创建的库中，这里只介绍一种方法，仅供参考。

```
[root@iLiuc ~]# mysqladmin create clear_360fy
```

```
[root@iLiuc ~]# mysql clear_360fy < /home/clear_360fy.sql
```

这条命令意思是先创建 `clear_360fy` 库，然后把 `/home` 目录下的 `clear_360fy.sql` 文件导入到该库中

## 5.10 常用命令总结

### 1. 刷新

```
flush privileges;
```

### 2. 显示所有数据库

```
show databases;
```

### 3. 打开数据库

```
use dbname;
```

### 4. 显示当前数据中所有的表

```
show tables;
```

### 5. 查看表结构

```
desc tables;
```

### 6. 删除数据库

```
drop database dbname;
```

### 7. 删除表

```
drop table tbname;
```

### 8. 创建数据库

```
create database dbname;
```

### 9. 查询

#### 10. 查看用户权限

```
# 查询
```

```
select 列名称 from 表名称;
```

```
# 查看用户权限
```

```
show grants for repl;
```

```
# 查看mysql进程
```

```
show processlist;
```

```
# 查看所有用户
```

```
select user();
```

```
# 查看主从状态
```

```
show slave status\G;
```

```
# 查看所有参数变量
```

```
show variables;
```

```
# 查看表的引擎状态
```

```
show table status
```

```
# 表存在就删除
```

```
drop table if exists user
```

```
# 表不存在就创建
```

```
create table if not exists user
```

```
# 查询用户权限 先用mysql
```

```
select host, user, password from user;
```



```
# 创建表
create table ka(ka_id varchar(6),qianshu int);

# 查看系统的字符集和排序方式的设定
show variables like 'character_set_%';

# 查看超时(wait_timeout)
show variables like '%timeout%';

# 删除空用户
delete from user where user='';

# 删除用户
delete from user where user='sss' and host='localhost' ;

# 改变现有的表使用的存储引擎
alter table mytable engine = MyISAM ;

# 查询表引擎
show table status from 库名 where Name='表名';

# 创建表指定存储引擎的类型(MyISAM或INNODB)
CREATE TABLE innodb (id int, title char(20)) ENGINE = INNODB

# 创建主从复制用户
grant replication slave on *.* to '用户'@'%' identified by '密码';

# 添加索引
ALTER TABLE player ADD INDEX weekcredit_faction_index (weekcredit, faction);

# 插入字段
alter table name add column accountid(列名) int(11) NOT NULL(字段不为空);

# 更新数据
update host set monitor_state='Y',hostname='username' where ip='192.168.1.1';

# 清除二进制文件
purge binary logs to 'mysql-bin.000051';
```

### 5.10.1 忘记 MySQL 密码

如果 MySQL 正在运行, 首先停掉服务, 然后终止进程; 在安全模式下进入 mysql 数据库去修改密码, 最后重启 mysqld 服务。具体操作为:

```
# service mysqld stop
# killall -TERM mysqld
```

```
# mysqld_safe --skip-grant-tables &  
# mysql  
mysql> use mysql;  
mysql> update user set password('new_pass') where user="root";  
mysql> exit  
# service mysqld restart
```

## 第六章 MySQL 复制

主服务器/从服务器设置增加了健壮性。主服务器出现问题时，可以切换到从服务器。

通过在主服务器和从服务器之间切分处理客户查询的负荷，可以得到更好的客户响应时间。`SELECT` 查询可以发送到从服务器以降低主服务器的查询处理负荷。但修改数据的语句仍然应发送到主服务器，以便主服务器和从服务器保持同步。

MySQL 提供了数据库的同步功能，这对实现数据库的冗余、备份、恢复、负载均衡等都是有极大帮助的。

### 6.1 复制如何工作

MySQL 复制数据，总的来说，有三个步骤：

1. 在主库上把数据更改记录到二进制日志（Binary Log）中（这些记录被称为二进制日志事件）。
2. 备库将主库上的日志复制到自己的中继日志（Relay Log）中。
3. 备库读取中继日志中的事件，将其重放到备库数据之上。

复制如何工作：

第一步是在主库上记录二进制日志。在每次准备提交事务完成数据更新前，主库将数据更新的事件记录到二进制日志中。MySQL 会按事务提交的顺序而非每条语句的执行顺序来记录二进制日志。在记录二进制日志后，主库会告诉存储引擎可以提交事务了。

下一步，备库将主库的二进制日志复制到其本地的中继日志中。首先，备库会启动一个工作线程，称为 I/O 线程，I/O 线程跟主库建立一个普通的客户端连接，然后在主库上启动一个特殊的二进制转储（binlog dump）线程（该线程没有对应的 SQL 命令），这个二进制转储线程会读取主库上二进制日志中的事件。它不会对事件进行轮询。如果该线程追赶上了主库，它将进入睡眠状态，直到主库发送信号量通知其有新的事件产生时才会被唤醒，备库 I/O 线程会将接收到的事件记录到中继日志中。

备库的 SQL 线程执行最后一步，该线程从中继日志中读取事件并在备库执行，从而实现备库数据的更新。当 SQL 线程追赶上 I/O 线程时，中继日志通常已经在系统缓存中了，所以中继日志的开销很低。SQL 线程执行的事件也可以通过配置选项来决定是否写入其自己的二进制日志中。

这种复制架构实现了获取事件的重放事件的解耦，允许这两个过程异步进行。也就是说 I/O 线程能够独立于 SQL 线程之外工作。但这种架构也限制了复制的过程，其中最重要的一点是在主库上并发运行的查询在备库上只能串行执行，因为只有一个 SQL 线程来重放中继日志的事件。

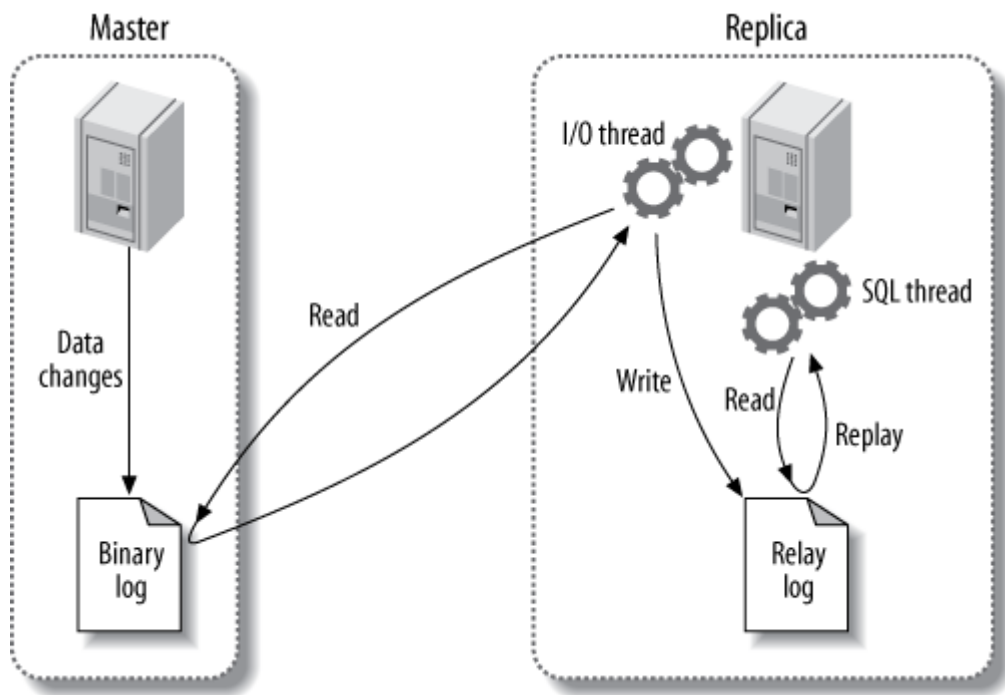


图 6.1 MySQL 复制的工作原理

## 6.2 MySQL 复制原理

MySQL 支持单向、双向复制、异步复制，复制过程中一个服务器充当主服务器，而一个或多个其他服务器充当从服务器。主服务器将更新写入一个二进制日志文件中，并创建一个索引文件以跟踪日志循环。这些日志文件可以将记录发送到从服务器以让从服务器保持与主服务器的数据一致。当一个从服务器连接主服务器时，日志文件会通知主服务器，从服务器在上次成功更新的位置处开始进入更新操作。更新完成后从服务器开始进入等待状态，等待主服务器后续的更新。

## 6.3 MySQL 同步细节

MySQL 同步功能由 3 个线程（Master 上 1 个 binlog dump，Slave 上 2 个，分别是 SQL 进程和 IO 进程）来实现。执行“START SLAVE”语句后，Slave 就创建

## 6.4 MySQL 半同步配置

### 1. 检查是否支持动态加载

登录 MySQL 并查询 `have_dynamic_loading` 变量

```
mysql_master> show variables like 'have_dynamic_loading';
mysql_slave> show variables like 'have_dynamic_loading';
+-----+
| Variable_name      | Value |
+-----+
| have_dynamic_loading | YES   |
```

```
+-----+-----+
1 row in set (0.00 sec)
```

## 2. 确认 lib/plugin 目录下有半同步插件 so

确认主库 MySQL 程序目录下有 lib/plugin/semisync\_master.so 文件, 确认备库 MySQL 程序目录下有 lib/plugin/semisync\_slave.so 文件

```
ls /usr/local/mysql/lib/plugin
adt_null.so          auth.so              auth_test_plugin.so
daemon_example.ini  libdaemon_example.so  qa_auth_client.so
qa_auth_server.so   semisync_slave.so    audit_log.so
auth_socket.so      authentication_pam.so debug
mypluglib.so        qa_auth_interface.so semisync_master.so
thread_pool.so
```

## 3. 主库动态加载半同步插件

登录 MySQL 并加载主库半同步插件, 并设置主库半同步有效

```
mysql_master> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so'
mysql_master> set global rpl_semi_sync_master_enabled=on
mysql_master> set global rpl_semi_sync_master_timeout=1000
```

## 4. 备库动态加载半同步插件

```
mysql_slave> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
mysql_slave> set global rpl_semi_sync_slave_enabled=on;
```

## 5. 主库 MySQL 配置文件修改

为了保证主库 MySQL 再次重启服务器后自动打开半同步, 需要增加主库 MySQL 配置如下:

```
rpl_semi_sync_master_enabled=1
rpl_semi_sync_master_timeout=1000
```

## 6. 备库 MySQL 配置文件修改

为了保证备库 MySQL 再次重启服务器后自动打开半同步, 需要增加备库 MySQL 配置如下:

```
rpl_semi_sync_slave_enabled=1
```

## 7. 备库 MySQL 重新连接

为了保证主备库是以半同步方式复制的, 建议备库上重启复制:

```
root@slave >stop slave ; start slave;
```



## 第七章 MySQL+KeepAlived

### 7.1 KeepAlived 介绍

Keepalived is a routing software written in C. The main goal of this project is to provide simple and robust facilities for loadbalancing and high-availability to Linux system and Linux based infrastructures. Loadbalancing framework relies on well-known and widely used Linux Virtual Server (IPVS) kernel module providing Layer4 loadbalancing. Keepalived implements a set of checkers to dynamically and adaptively maintain and manage loadbalanced server pool according their health. On the other hand high-availability is achieved by VRRP protocol. VRRP is a fundamental brick for router failover. In addition, Keepalived implements a set of hooks to the VRRP finite state machine providing low-level and high-speed protocol interactions. Keepalived frameworks can be used independently or all together to provide resilient infrastructures.

使用 MySQL 双 master+keepalived 是一种非常好的解决方案，在 MySQL-HA 环境中，MySQL 互为主从关系，这样就保证了两台 MySQL 数据的一致性，然后用 KeepAlived 实现虚拟 IP，通过 KeepAlived 自带的服务监控功能来实现 MySQL 故障时自动切换。

环境：

mysql\_01:

hostname A0304010 172.16.25.39

mysql\_02:

hostname A0305010 172.16.25.40

各个主机授权：

```
mysql_01> grant replication slave on *.* to 'repl01'@'%' identified by 'replpass'
mysql_01> flush privileges;
```

```
mysql_02> grant replication slave on *.* to 'repl02'@'%' identified by 'replpass'
mysql_02> flush privileges;
```

查看各主机的日志文件及日志位置：

```
mysql_01> show master status\G
```

```
***** 1. row *****
      File: mysql-bin.000076
      Position: 346
```

```

    Binlog_Do_DB:
Binlog_Ignore_DB:
1 row in set (0.00 sec)

```

```

mysql_02> show master status\G
***** 1. row *****
      File: mysql-bin.000076
      Position: 346
    Binlog_Do_DB:
Binlog_Ignore_DB:
1 row in set (0.00 sec)

```

各主机change master to:

```

mysql_01> change master to
-> master_host='172.16.25.40',
-> master_user='repl02',
-> master_password='replpass',
-> master_port=3313,
-> master_log_file='mysql-bin.000076',
-> master_log_pos=346;
Query OK, 0 rows affected (0.35 sec)

```

```

mysql_01> start slave;
mysql_01> show slave status\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
      Master_Host: 172.16.25.40
      Master_User: repl02
      Master_Port: 3313
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000076
      Read_Master_Log_Pos: 346
      Relay_Log_File: mysql-relay-bin.000002
      Relay_Log_Pos: 253
      Relay_Master_Log_File: mysql-bin.000076
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:

```



```

Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
    Last_Errno: 0
    Last_Error:
    Skip_Counter: 0
Exec_Master_Log_Pos: 346
Relay_Log_Space: 409
Until_Condition: None
Until_Log_File:
Until_Log_Pos: 0
Master_SSL_Allowed: No
Master_SSL_CA_File:
Master_SSL_CA_Path:
Master_SSL_Cert:
Master_SSL_Cipher:
Master_SSL_Key:
Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
    Last_IO_Errno: 0
    Last_IO_Error:
    Last_SQL_Errno: 0
    Last_SQL_Error:
Replicate_Ignore_Server_Ids:
Master_Server_Id: 4008
1 row in set (0.00 sec)

```

```

mysql_02> change master to
master_host='172.16.25.39',
master_user='repl01',
master_password='replpass',
master_port=3313,
master_log_file='mysql-bin.000076',
master_log_pos=346;
Query OK, 0 rows affected (0.20 sec)

```

```

mysql_02> start slave;
mysql_02> show slave status\G
***** 1. row *****
      Slave_IO_State: Waiting for master to send event
        Master_Host: 172.16.25.39
        Master_User: repl01

```

```
      Master_Port: 3313
      Connect_Retry: 60
      Master_Log_File: mysql-bin.000076
      Read_Master_Log_Pos: 346
      Relay_Log_File: mysql-relay-bin.000002
      Relay_Log_Pos: 253
      Relay_Master_Log_File: mysql-bin.000076
      Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
      Replicate_Do_DB:
      Replicate_Ignore_DB:
      Replicate_Do_Table:
      Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
      Replicate_Wild_Ignore_Table:
      Last_Errno: 0
      Last_Error:
      Skip_Counter: 0
      Exec_Master_Log_Pos: 346
      Relay_Log_Space: 409
      Until_Condition: None
      Until_Log_File:
      Until_Log_Pos: 0
      Master_SSL_Allowed: No
      Master_SSL_CA_File:
      Master_SSL_CA_Path:
      Master_SSL_Cert:
      Master_SSL_Cipher:
      Master_SSL_Key:
      Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
      Last_IO_Errno: 0
      Last_IO_Error:
      Last_SQL_Errno: 0
      Last_SQL_Error:
      Replicate_Ignore_Server_Ids:
      Master_Server_Id: 3908
1 row in set (0.00 sec)
```

## 7.2 测试同步

```
mysql_01> show databases;
```

```
+-----+
```

```
| Database |
```

```
+-----+
```

```
| information_schema |
```

```
| mysql |
```

```
| performance_schema |
```

```
| sbtest |
```

```
| test |
```

```
+-----+
```

```
5 rows in set (0.00 sec)
```

```
mysql_02> show databases;
```

```
+-----+
```

```
| Database |
```

```
+-----+
```

```
| information_schema |
```

```
| mysql |
```

```
| performance_schema |
```

```
| sbtest |
```

```
| test |
```

```
+-----+
```

```
5 rows in set (0.00 sec)
```

```
mysql_01> create database doublec;
```

```
mysql_02> show databases;
```

```
+-----+
```

```
| Database |
```

```
+-----+
```

```
| information_schema |
```

```
| liucc |
```

```
| mysql |
```

```
| performance_schema |
```

```
| sbtest |
```

```
| test |
```

```
+-----+
```

```
6 rows in set (0.00 sec)
```

```
mysql_02> drop database doublec;
```

```
mysql_01> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| mysql              |
| performance_schema |
| sbtest             |
| test               |
+-----+
5 rows in set (0.00 sec)
```

### 7.3 安装配置 KeepAlived

安装配置 keepalived:

A0304010:

```
rpm -qa |grep keepalived
```

keepalived-1.2.7-8.1

```
cat /etc/keepalived/keepalived.conf
```

! Configuration File for keepalived

```
global_defs {
    router_id LVS_DEVEL
}
```

```
vrrp_instance MySQL_Loadblancing {
    state BACKUP
    interface vlan0
    virtual_router_id 7
    priority 250
    advert_int 1
    nopreempt
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        172.16.25.7
    }
}
```

```
virtual_server 172.16.25.7 3313 {
    delay_loop 6
    lb_algo wrr
    lb_kind DR
    protocol TCP
    persistence_timeout 50

    real_server 172.16.25.39 3313 {
        weight 3
        notify_down /home/mysql/bin/kill_keepalived.sh
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3306
        }
    }
}
```

```
cat /home/mysql/bin/kill_keepalived.sh
#!/bin/bash
pkill keepalived
```

```
A0305010:
rpm -qa |grep keepalived
keepalived-1.2.7-8.1
```

```
cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived
```

```
global_defs {
    router_id LVS_DEVEL
}
```

```
vrrp_instance MySQL_Loadblancing {
    state BACKUP
    interface vlan0
    virtual_router_id 7
    priority 200
```

```
advert_int 1
authentication {
    auth_type PASS
    auth_pass 1111
}
virtual_ipaddress {
    172.16.25.7
}
}

virtual_server 172.16.25.7 3313 {
    delay_loop 6
    lb_algo wrr
    lb_kind DR
    protocol TCP
    persistence_timeout 50

    real_server 172.16.25.40 3313 {
        weight 3
        notify_down /home/mysql/bin/kill_keepalived.sh
        TCP_CHECK {
            connect_timeout 3
            nb_get_retry 3
            delay_before_retry 3
            connect_port 3313
        }
    }
}

cat /home/mysql/bin/kill_keepalived.sh
#!/bin/bash
pkill keepalived
```

## 7.4 启动 KeepAlived

启动 keepalived 服务，验证 VIP：

A0304010:

/etc/init.d/keepalived start

ip a

.....

```

8: vlan0@bond0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state U
    link/ether 6c:92:bf:0a:73:e3 brd ff:ff:ff:ff:ff:ff
    inet 172.16.25.39/24 brd 172.16.25.255 scope global vlan0
    inet 172.16.25.7/32 scope global vlan0
    inet6 fe80::6e92:bfff:fe0a:73e3/64 scope link
        valid_lft forever preferred_lft forever
.....

```

A0304010:

```
mysql_01> select host, user from mysql.user;
```

```

+-----+-----+
| host      | user    |
+-----+-----+
| %         | repl    |
| %         | repl01  |
| %         | sbtest  |
| localhost | root    |
+-----+-----+
4 rows in set (0.00 sec)

```

A0305010:

```
/etc/init.d/keepalived start
```

```
ip a
```

.....

```

8: vlan0@bond0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state U
    link/ether 6c:92:bf:0a:53:07 brd ff:ff:ff:ff:ff:ff
    inet 172.16.25.40/24 brd 172.16.25.255 scope global vlan0
    inet6 fe80::6e92:bfff:fe0a:5307/64 scope link
        valid_lft forever preferred_lft forever
.....

```

在其他机器上，测试VIP的连通性：

A0304006:

```
ping 172.16.25.7
```

```
PING 172.16.25.7 (172.16.25.7) 56(84) bytes of data.
```

```
64 bytes from 172.16.25.7: icmp_seq=1 ttl=64 time=0.174 ms
```

```
64 bytes from 172.16.25.7: icmp_seq=2 ttl=64 time=0.143 ms
```

```
64 bytes from 172.16.25.7: icmp_seq=3 ttl=64 time=0.151 ms
```

```
64 bytes from 172.16.25.7: icmp_seq=4 ttl=64 time=0.168 ms
```

```
64 bytes from 172.16.25.7: icmp_seq=5 ttl=64 time=0.174 ms
```

^C

--- 172.16.25.7 ping statistics ---

5 packets transmitted, 5 received, 0% packet loss, time 3998ms

rtt min/avg/max/mdev = 0.143/0.162/0.174/0.012 ms

在其他机器上登陆VIP, 进行验证:

A0304002:~ # mysql -h172.16.25.39 -usbtest -P3313 -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 217

Server version: 5.5.32-enterprise-commercial-advanced-log MySQL Enterprise

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql>mysql> use sbtest;

Database changed

mysql> show tables;

+-----+

| Tables\_in\_sbtest |

+-----+

| sbtest1 |

| sbtest10 |

| sbtest11 |

| sbtest12 |

| sbtest13 |

| sbtest14 |

| sbtest15 |

| sbtest16 |

| sbtest17 |

| sbtest18 |

| sbtest19 |

| sbtest2 |

| sbtest20 |

| sbtest21 |

| sbtest22 |

| sbtest23 |

| sbtest24 |

| sbtest3 |

| sbtest4 |

| sbtest5 |

| sbtest6 |



```
| sbtest7          |
| sbtest8          |
| sbtest9          |
+-----+
24 rows in set (0.00 sec)
```

```
root@A0304010 /data1]
```

```
$netstat -antup |grep 3313
```

```
tcp        0      0 0.0.0.0:3313          0.0.0.0:*             LISTEN
tcp        0      0 172.16.25.39:45087    172.16.25.40:3313     ESTABLISHED
tcp        0      0 172.16.25.39:3313     172.16.25.40:42982    ESTABLISHED
tcp        0      0 172.16.25.39:3313     172.16.25.32:32845    ESTABLISHED
```

```
MySQL8 2014-12-26 12:20:14 root@A0305010 /data1]
```

```
$netstat -antup |grep 3313
```

```
tcp        0      0 0.0.0.0:3313          0.0.0.0:*             LISTEN
tcp        0      0 172.16.25.40:3313     172.16.25.39:45087    ESTABLISHED
tcp        0      0 172.16.25.40:42982    172.16.25.39:3313     ESTABLISHED
```

停掉25.39的MySQL:

```
[MySQL8 2014-12-26 12:48:32 root@A0304010 /data1]
```

```
$netstat -antup |grep 3313
```

```
tcp        0      0 172.16.25.39:45087    172.16.25.40:3313     FIN_WAIT2
tcp        0      0 172.16.25.39:3313     172.16.25.40:42982    TIME_WAIT
tcp        0      0 172.16.25.39:3313     172.16.25.32:32845    FIN_WAIT2
```

```
/etc/init.d/keepalived status
```

```
Checking for Keepalived daemon                                unused
```

故障切换:

172.16.25.36为客户端, 连接VIP来进行查询,

首先, 停掉25.39的MySQL服务, 验证25.39上的keepalived是不是被停掉了。

```
ip a
```

```
.....
```

```
8: vlan0@bond0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state U
    link/ether 6c:92:bf:0a:73:e3 brd ff:ff:ff:ff:ff:ff
    inet 172.16.25.39/24 brd 172.16.25.255 scope global vlan0
```

```

inet 172.16.25.7/32 scope global vlan0
inet6 fe80::6e92:bfff:fe0a:73e3/64 scope link
    valid_lft forever preferred_lft forever
.....

```

然后，查看25.36客户端的连接情况。

```
A0304002:~ # mysql -h172.16.25.7 -usbtest -P3313 -p
```

```
Enter password: xxxxxx
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 83
```

```
Server version: 5.5.32-enterprise-commercial-advanced-log MySQL Enterprise
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement
```

```
mysql> show databases;
```

```
+-----+
```

```
| Database          |
```

```
+-----+
```

```
| information_schema |
```

```
| sbtest            |
```

```
| test              |
```

```
+-----+
```

```
3 rows in set (0.00 sec)
```

```
mysql> use sbtest;
```

```
mysql> show tables;
```

```
+-----+
```

```
| Tables_in_sbtest  |
```

```
+-----+
```

```
| sbtest1           |
```

```
| sbtest10          |
```

```
| sbtest11          |
```

```
| sbtest12          |
```

```
| sbtest13          |
```

```
| sbtest14          |
```

```
| sbtest15          |
```

```
| sbtest16          |
```

```
| sbtest17          |
```

```
| sbtest18          |
```

```
| sbtest19          |
```

```
| sbtest2           |
```

```

| sbtest20          |
| sbtest21          |
| sbtest22          |
| sbtest23          |
| sbtest24          |
| sbtest3           |
| sbtest4           |
| sbtest5           |
| sbtest6           |
| sbtest7           |
| sbtest8           |
| sbtest9           |
+-----+
24 rows in set (0.00 sec)

```

```

mysql> show tables;
ERROR 1053 (08S01): Server shutdown in progress

```

```

mysql> show tables;
No connection. Trying to reconnect...
ERROR 2003 (HY000): Can't connect to MySQL server on '172.16.25.7' (113)
ERROR:
Can't connect to the server

```

```

mysql> show tables;
ERROR 2013 (HY000): Lost connection to MySQL server during query
mysql> show tables;
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:      22
Current database: sbtest

```

```

+-----+
| Tables_in_sbtest |
+-----+
| sbtest1          |
| sbtest10         |
| sbtest11         |
| sbtest12         |
| sbtest13         |
| sbtest14         |

```

```

| sbtest15      |
| sbtest16      |
| sbtest17      |
| sbtest18      |
| sbtest19      |
| sbtest2       |
| sbtest20      |
| sbtest21      |
| sbtest22      |
| sbtest23      |
| sbtest24      |
| sbtest3       |
| sbtest4       |
| sbtest5       |
| sbtest6       |
| sbtest7       |
| sbtest8       |
| sbtest9       |
+-----+
24 rows in set (0.02 sec)

```

最后，查看25.40的keepalived状态及连接情况。

```
ip a
```

```
.....
```

```

8: vlan0@bond0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue st
    link/ether 6c:92:bf:0a:53:07 brd ff:ff:ff:ff:ff:ff
    inet 172.16.25.40/24 brd 172.16.25.255 scope global vlan0
    inet 172.16.25.7/32 scope global vlan0
    inet6 fe80::6e92:bfff:fe0a:5307/64 scope link

```

```
.....
```

```
$netstat -antup |grep 3313
```

```

tcp        0      0 0.0.0.0:3313          0.0.0.0:*             LISTEN
tcp        0      0 172.16.25.7:3313     172.16.25.32:43533    ESTABLIS

```

```
A0304010:
```

```
A0304010:~ # grep -v "^#" /home/mysql/conf/my8.cnf
```

```
[client]
```

```
loose_default-character-set = utf8
```

```
port=3313
socket=/home/mysql/data/mysqldata8/sock/mysql.sock
user=admin

[mysqld]
large-pages
read_only=off
memlock
default-storage-engine = INNODB
character-set-server=utf8
collation = utf8_bin

user=mysql
port=3313
socket=/home/mysql/data/mysqldata8/sock/mysql.sock
pid-file=/home/mysql/data/mysqldata8/sock/mysql.pid
datadir=/home/mysql/data/mysqldata8/mydata
tmpdir=/home/mysql/data/mysqldata8/tmpdir

skip-name-resolve
skip_external_locking

lower_case_table_names=1
event_scheduler=0
back_log=512
default-time-zone='+8:00'

max_connections = 3000
max_connect_errors=99999
max_allowed_packet = 64M
max_heap_table_size = 8M
max_length_for_sort_data = 16k

wait_timeout=172800
interactive_timeout=172800

net_buffer_length = 8K
read_buffer_size = 2M
read_rnd_buffer_size = 2M
sort_buffer_size = 2M
join_buffer_size = 4M
```

```
binlog_cache_size = 2M
```

```
table_open_cache = 2048
```

```
table_definition_cache = 2048
```

```
thread_cache_size = 512
```

```
tmp_table_size = 8M
```

```
query_cache_size=0
```

```
query_cache_type=OFF
```

```
log-error=/home/mysql/data/mysqldata8/log/error.log
```

```
long_query_time = 1
```

```
slow_query_log
```

```
slow_query_log_file=/home/mysql/data/mysqldata8/slowlog/slow-query.log
```

```
log_warnings
```

```
log_slow_slave_statements
```

```
server-id=03908
```

```
log-bin=/home/mysql/data/mysqldata8/binlog/mysql-bin
```

```
binlog-format=ROW
```

```
max_binlog_size = 512M
```

```
expire_logs_days=15
```

```
sync_binlog=1
```

```
auto_increment_increment=2
```

```
auto_increment_offset=2
```

```
relay-log=/home/mysql/data/mysqldata8/relaylog/mysql-relay-bin
```

```
slave-skip-errors=1022,1032,1062
```

```
report-port=3313
```

```
log_bin_trust_function_creators=1
```

```
log_slave_updates=1
```

```
key_buffer_size = 8M
```

```
bulk_insert_buffer_size = 8M
```

```
myisam_sort_buffer_size = 64M
```

```
myisam_max_sort_file_size = 10G
```

```
myisam_repair_threads = 1
```

```
myisam_recover
```

```
innodb_data_home_dir = /home/mysql/data/mysqldata8/innodb_ts
innodb_data_file_path = ibdata1:2048M:autoextend
innodb_file_per_table
innodb_file_format = barracuda
innodb_file_format_max = barracuda
innodb_file_format_check = ON
innodb_strict_mode = 1
innodb_flush_method = O_DIRECT
innodb_autoinc_lock_mode=2

innodb_additional_mem_pool_size = 8M
innodb_buffer_pool_size = 1G
innodb_max_dirty_pages_pct = 75
innodb_adaptive_flushing = ON
innodb_change_buffering = inserts
innodb_old_blocks_time = 1000

innodb_log_group_home_dir = /home/mysql/data/mysqldata8/innodb_log
innodb_log_buffer_size = 64M
innodb_log_file_size = 2000M
innodb_log_files_in_group = 2
innodb_flush_log_at_trx_commit = 1
innodb_support_xa = ON

innodb_thread_concurrency = 32
innodb_lock_wait_timeout = 120
innodb_rollback_on_timeout = 1
transaction_isolation = READ-COMMITTED

innodb_read_io_threads = 4
innodb_write_io_threads = 12
innodb_io_capacity = 40000
innodb_use_native_aio = 1

innodb_purge_threads = 1

[mysqldump]
quick
max_allowed_packet = 2G
default-character-set = utf8
```

```
[mysql]
no-auto-rehash
show-warnings
prompt="5.1 \u@\h : \d \r:\m:\s> "
default-character-set = utf8
```

```
[myisamchk]
key_buffer = 512M
sort_buffer_size = 512M
read_buffer = 8M
write_buffer = 8M
```

```
[mysqlhotcopy]
interactive-timeout
```

```
[mysqld_safe]
user=mysql
open-files-limit = 8192
```

```
-----

A0305010:
$grep -v "^#" /home/mysql/conf/my8.cnf
[client]
loose_default-character-set = utf8
port=3313
socket=/home/mysql/data/mysqldata8/sock/mysql.sock
user=admin
```

```
[mysqld]
large-pages
read_only=off
memlock
default-storage-engine = INNODB
character-set-server=utf8
collation = utf8_bin

user=mysql
port=3313
socket=/home/mysql/data/mysqldata8/sock/mysql.sock
```



```
pid-file=/home/mysql/data/mysqldata8/sock/mysql.pid
datadir=/home/mysql/data/mysqldata8/mydata
tmpdir=/home/mysql/data/mysqldata8/tmpdir

skip-name-resolve
skip_external_locking

lower_case_table_names=1
event_scheduler=0
back_log=512
default-time-zone='+8:00'

max_connections = 3000
max_connect_errors=99999
max_allowed_packet = 64M
max_heap_table_size = 8M
max_length_for_sort_data = 16k

wait_timeout=172800
interactive_timeout=172800

net_buffer_length = 8K
read_buffer_size = 2M
read_rnd_buffer_size = 2M
sort_buffer_size = 2M
join_buffer_size = 4M
binlog_cache_size = 2M

table_open_cache = 2048
table_definition_cache = 2048
thread_cache_size = 512
tmp_table_size = 8M

query_cache_size=0
query_cache_type=OFF

log-error=/home/mysql/data/mysqldata8/log/error.log
long_query_time = 1
slow_query_log
slow_query_log_file=/home/mysql/data/mysqldata8/slowlog/slow-query.log
log_warnings
```

log\_slow\_slave\_statements

server-id=04008

log-bin=/home/mysql/data/mysqldata8/binlog/mysql-bin

binlog-format=ROW

max\_binlog\_size = 512M

expire\_logs\_days=15

sync\_binlog=1

auto\_increment\_increment=2

auto\_increment\_offset=2

relay-log=/home/mysql/data/mysqldata8/relaylog/mysql-relay-bin

slave-skip-errors=1022,1032,1062

report-port=3313

log\_bin\_trust\_function\_creators=1

log\_slave\_updates=1

key\_buffer\_size = 8M

bulk\_insert\_buffer\_size = 8M

myisam\_sort\_buffer\_size = 64M

myisam\_max\_sort\_file\_size = 10G

myisam\_repair\_threads = 1

myisam\_recover

innodb\_data\_home\_dir = /home/mysql/data/mysqldata8/innodb\_ts

innodb\_data\_file\_path = ibdata1:2048M:autoextend

innodb\_file\_per\_table

innodb\_file\_format = barracuda

innodb\_file\_format\_max = barracuda

innodb\_file\_format\_check = ON

innodb\_strict\_mode = 1

innodb\_flush\_method = O\_DIRECT

innodb\_autoinc\_lock\_mode=2

innodb\_additional\_mem\_pool\_size = 8M

innodb\_buffer\_pool\_size = 8M

innodb\_max\_dirty\_pages\_pct = 75

innodb\_adaptive\_flushing = ON

innodb\_change\_buffering = inserts

innodb\_old\_blocks\_time = 1000

```
innodb_log_group_home_dir = /home/mysql/data/mysqldata8/innodb_log
innodb_log_buffer_size = 64M
innodb_log_file_size = 2000M
innodb_log_files_in_group = 2
innodb_flush_log_at_trx_commit = 1
innodb_support_xa = ON
```

```
innodb_thread_concurrency = 32
innodb_lock_wait_timeout = 120
innodb_rollback_on_timeout = 1
transaction_isolation = READ-COMMITTED
```

```
innodb_read_io_threads = 4
innodb_write_io_threads = 12
innodb_io_capacity = 40000
innodb_use_native_aio = 1
```

```
innodb_purge_threads = 1
```

```
[mysqldump]
quick
max_allowed_packet = 2G
default-character-set = utf8
```

```
[mysql]
no-auto-rehash
show-warnings
prompt="5.5 \u@\h : \d \r:\m:\s> "
default-character-set = utf8
```

```
[myisamchk]
key_buffer = 512M
sort_buffer_size = 512M
read_buffer = 8M
write_buffer = 8M
```

```
[mysqlhotcopy]
interactive-timeout
```

```
[mysqld_safe]
user=mysql
```

```
open-files-limit = 8192
```

## 第八章 MySQL+LVS



## 第九章 MHA

本文介绍如何一步步搭建 MHA 高可用环境，

### 9.1 实验环境说明

使用的 MySQL 版本为 5.5.32 企业版，MHA 版本为 0.56 版本，三台服务器。

#### 9.1.1 SSH 无密码通信设置

在所有机器上执行，设置 SSH 无密码通信：

```
# ssh-keygen -t rsa
# ssh-copy-id -i /root/.ssh/id_rsa.pub root@10.10.7.16
# ssh-copy-id -i /root/.ssh/id_rsa.pub root@10.10.7.17
# ssh-copy-id -i /root/.ssh/id_rsa.pub root@10.10.7.201
```

### 9.2 安装 MySQL

使用二进制的方式安装 MySQL，

### 9.3 安装及配置 MHA

安装方式有两种，可以使用源代码编译安装或者使用 RPM 包进行安装。使用 RPM 包安装简单些，本次安装使用 RPM 包方式进行安装。安装包可以到这里 <https://code.google.com/p/mysql-master-ha/downloads/list> 下载。

#### 9.3.1 安装 node 节点

在所有机器上执行，安装 mha 的 node 节点，安装依赖包：

```
# yum install -y perl-DBD-MySQL
# rpm -ivh mha4mysql-node-0.56-0.el6.noarch.rpm
```

#### 9.3.2 安装 manager 节点

在 Slave 机器上执行，安装 mha 的 manager 节点，

```
# yum install -y perl-Config-Tiny
# yum install -y perl-DBD-MySQL
# yum install -y perl-MIME-Lite
# yum install -y perl-Params-Validate
# yum install -y perl-Time-HiRes
# rpm -ivh perl-Mail-Sender-0.8.16-3.el6.noarch.rpm
# rpm -ivh perl-Mail-Sendmail-0.79-12.el6.noarch.rpm
# rpm -ivh perl-Log-Dispatch-2.27-1.el6.noarch.rpm
# rpm -ivh perl-Parallel-ForkManager-0.7.9-1.el6.noarch.rpm
# rpm -ivh mha4mysql-manager-0.56-0.el6.noarch.rpm
```



## 第三部分

### 基本服务篇



本章介绍几种常见的服务。这里不注重概念的介绍，概念只做简短的描述，主要介绍如何实施及实现这些服务。



## 第十章 DHCP

随着网络规模的不断扩大和网络复杂度的提高，计算机的数量经常超过可分配的 IP 地址数量。同时随着便携机及无线网络的广泛使用，计算机的位置也经常变化，相应的 IP 地址也必须经常更新，从而导致网络配置越来越复杂。DHCP (Dynamic Host Configuration Protocol，动态主机配置协议) 就是为满足这些需求而发展起来的。

DHCP 提供安全、可靠且简单的 TCP/IP 网络设置，避免了 TCP/IP 网络中地址的冲突，同时也降低了管理 IP 地址设置的工作强度。使用 DHCP 主要的好处有以下几点：

1. 减少管理员的工作量
2. 减小输入错误的可能
3. 避免 IP 冲突
4. 当网段更改 IP 地址段时，不需要重新配置每台计算机的 IP
5. 计算机移动不必重新配置 TCP/IP 信息
6. 提高了 IP 地址的利用率

当一台 DHCP 客户端启动时，该客户端将在网络中请求 IP 地址，当 DHCP 服务器收到申请 IP 地址请求后，它将从可用的地址中选择一个提供给 DHCP 客户端。

客户端除了可以从 DHCP 服务器得到 IP 地址信息外，还可以得到子网掩码、默认网关、DNS 服务器等其他 TCP/IP 信息。一般这些信息并非永久使用，而是有一个使用的期限，这个期限被称为租约。所以 DHCP 服务器与客户端之间的通信分为两类，即租约产生及租约更新。

DHCP 的作用是为局域网中的每台计算机自动分配 TCP/IP 信息，包括 IP 地址、子网掩码、网关以及 DNS 服务器等。其优点是终端无须配置，且网络维护方便。

动态主机配置协议 (DHCP)，

1. 服务端端口 67/UDP
2. 客户端端口 68/UDP
3. 客户端发送 DHCPDISCOVER 在网络中寻求地址分配
4. 服务端回应 DHCPDISCOVER 请求
5. 客户端发送 DHCPREQUEST
6. 服务端发送 DHCPACK
7. 客户端得到地址

主配置文件 `/usr/share/doc/dhcp-<version>/dhcpcd.conf.sample` 包含以下 7 项服务器即可搭建

1. ddns-update-style
2. subnet
3. option
4. range dynamic-bootp
5. default-lease-time
6. max-lease-time
7. host

```
ddns-update-style interim;
ignore client-updates;
default-lease-time 600;
max-lease-time 7200;

subnet 192.168.0.0 netmask 255.255.255.0 {
    next-server 192.168.0.128;
    filename="pxelinux.0";
    option routers 192.168.0.128;
    option subnet-mask 255.255.255.0;
    option nis-domain "lavenliu.com";
    option domain-name "lavenliu.com";
    option ntp-servers 192.168.0.254;
    range dynamic-bootp 192.168.0.100 192.168.0.130;
    host stu1 {
        next-server lavenliu.com;
        hardware Ethernet 12:34:56:78:AB:CD;
        fixed-address 192.168.0.2;
    }
}
```

# 第十一章 DNS

在 TCP/IP 网络中，IP 地址是网络节点的标识。但是，IP 地址是点分十进制数字，难以记忆。联想到在现实生活中，名字比身份证号码更容易被人记住，那么是否可以拿名字来标识某个网络节点呢？答案是肯定的。DNS（Domain Name System，域名系统）是一种用于 TCP/IP 应用程序的分布式数据库，提供域名与 IP 地址之间的转换。

## 11.1 测试环境

以下测试是在 CentOS6U5 64 位系统上进行测试。系统均关闭 iptables 及 selinux，如果要开启 iptables，请设置允许开放本机的 53 端口。

表 11-1 DNS 演示环境机器一览

主机名	IP 地址	说明
master01.lavenliu.com	192.168.20.134	主 DNS
minion01.lavenliu.com	192.168.20.135	辅 DNS
minion02.lavenliu.com	192.168.20.136	客户端

## 11.2 安装及配置主 DNS

在 master01.lavenliu.com 上安装如下软件包，

```
[root@master01 ~]# yum install -y bind bind-utils
```

接下来配置主 DNS，有两个文件需要修改，一个是主配置文件/etc/named.conf 及/etc/named.rfc1912.zones 文件。首先配置/etc/named.conf 文件，配置内容如下，在 options 字段中做如下修改：

```
listen-on port 53 { 127.0.0.1; 192.168.20.134; }; // 192.168.20.134 is Master DNS
listen-on-v6 port 53 { ::1; };
directory "/var/named";
dump-file "/var/named/data/cache_dump.db";
statistics-file "/var/named/data/named_stats.txt";
memstatistics-file "/var/named/data/named_mem_stats.txt";
allow-query { localhost; 192.168.20.0/24; }; // IP range
allow-transfer { localhost; 192.168.20.135; }; // 192.168.20.135 is Slave DNS Server
```

接下来修改/etc/named.rfc1912.zones 文件，在该文件末尾追加如下内容：

```
cat >> /etc/named.rfc1912.zones <<EOF
zone "lavenliu.com" IN {
    type master;
    file "forward.lavenliu";
    allow-update { none; };
};

zone "20.168.192.in-addr.arpa" IN {
    type master;
    file "reverse.lavenliu";
    allow-update { none; };
};
EOF
```

由于我们在/etc/named.rfc1912.zones 文件中设置了两个区域文件，分别为 forward.lavenliu 及 reverse.lavenliu 两个区域文件。这两个文件分别是正向解析文件与反向解析文件，接下来就创建两个文件，这两个文件的位置默认是/var/named 目录下。

首先，创建正向解析文件，

```
# vi /var/named/forward.lavenliu
$TTL 86400
@ IN SOA master01.lavenliu.com. root.lavenliu.com. (
    2016051201 ;Serial
    3600 ;Refresh
    1800 ;Retry
    604800 ;Expire
    86400 ;Minimum TTL
)
@ IN NS master01.lavenliu.com.
@ IN NS minion01.lavenliu.com.
@ IN A 192.168.20.134
@ IN A 192.168.20.135
@ IN A 192.168.20.136
master01 IN A 192.168.20.134
minion01 IN A 192.168.20.135
minion02 IN A 192.168.20.136
```

其次，创建反向解析文件，

```
# vi /var/named/reverse.lavenliu
$TTL 86400
```



```

@    IN  SOA      master01.lavenliu.com. root.lavenliu.com. (
        2016051201   ;Serial
3600          ;Refresh
1800          ;Retry
604800        ;Expire
86400         ;Minimum TTL
)
@                IN  NS      master01.lavenliu.com.
@                IN  NS      minion01.lavenliu.com.
@                IN  PTR     lavenliu.com.
master01        IN  A       192.168.20.134
minion01        IN  A       192.168.20.135
minion02        IN  A       192.168.20.136
134            IN  PTR     master01.lavenliu.com.
135            IN  PTR     minion01.lavenliu.com.
136            IN  PTR     minion02.lavenliu.com.

```

### 11.2.1 启动 DNS 服务

有了以上步骤的配置，我们就可以启动主 DNS 了。在启动之前，最好检查一下我们的配置文件的语法是否正确，另外一个就是文件的权限问题了。由于我们是用 `root` 用户创建的正向及反向解析文件，所以这两个文件的所有者及所属组均为 `root` 用户和组，接下来的第一件事情就是修改这两个文件的权限，

```

[root@master01 named]# chown -R named.named /var/named
[root@master01 named]# ll /var/named/
total 36
drwxrwx--- 2 named named 4096 Mar 16 21:25 data
drwxrwx--- 2 named named 4096 Mar 16 21:25 dynamic
-rw-r--r-- 1 named named  531 May 14 21:54 forward.lavenliu
-rw-r----- 1 named named 2075 Apr 23  2014 named.ca
-rw-r----- 1 named named  152 Dec 15  2009 named.empty
-rw-r----- 1 named named  152 Jun 21  2007 named.localhost
-rw-r----- 1 named named  168 Dec 15  2009 named.loopback
-rw-r--r-- 1 named named  564 May 14 21:57 reverse.lavenliu
drwxrwx--- 2 named named 4096 Mar 16 21:25 slaves

```

接下来检查正向及反向解析文件的语法正确性，

```

[root@master01 ~]# named-checkconf /etc/named.conf
[root@master01 ~]# named-checkzone lavenliu.com /var/named/forward.lavenliu
zone lavenliu.com/IN: loaded serial 2016051201
OK

```

```
[root@master01 ~]# named-checkzone lavenliu.com /var/named/reverse.lavenliu.com
zone lavenliu.com/IN: loaded serial 2016051201
OK
```

看上去没有问题，接下来就可以启动 **named** 服务了，

```
[root@master01 ~]# /etc/init.d/named start
```

启动之后，最好检查一下 **named** 的进程是否启动成功及 53 端口是否监听，

```
ps -ef |grep named |grep -v grep
netstat -antup |grep named |grep -v grep
```

## 第十二章 FTP

在互联网中，人们经常需要在远端主机与本地服务器之间传输文件，文件传输协议提供的服务满足了人们的这种需求。FTP（File Transfer Protocol，文件传输协议）是互联网上文件传输的标准协议，FTP 使用 TCP 作为传输协议，支持用户的登录认证及访问权限的设置。互联网上另一种常用的文件传输协议是 TFTP（Trivial File Transfer Protocol，普通文件传输协议），TFTP 是一种简单的文件传输协议，不支持用户的登录认证，也不具备复杂的命令。TFTP 使用 UDP 作为传输协议，并具有重传机制。



## 第十三章 NFS

NFS（Network File System）网络文件系统，目前依然非常流行。NFS 的一个最大优点是具有广泛的支持：大部分的类 UNIX 都可以支持 NFS。NFS 通常比其他的网络文件系统更容易配置和使用。与其他网络文件系统一样，NFS 可以在服务器或任何一个客户端上修改文件，然后在其他所有系统上可以立即使用修改后的文件。

安装很简单：

### 13.1 配置 NFS 服务器

### 13.2 配置 NFS 客户端

客户端基本上不用配置就可以使用。



## 第十四章 Kickstart

Red Hat Linux 提供了一种非常方便的自动化系统安装方式，即为 **kickstart**。这种工具的出现，极大的方便了众多的系统管理员或装机攻城狮。有了它，我们就不用拿着光盘或 U 盘在机房来回乱窜地装机了，我们可以在办公室通过 **IPMI** 的方式来操作。不管这个工具有多优秀，相比之前单台的安装方式，效率提升了很多。当然，也有另外几种较优越的自动化系统安装工具，这里就不介绍了。

**kickstart** 文件包含了安装程序所使用的指令，在安装的过程中可以用来减少或者消除用户输入的麻烦。

在系统数目巨大且完全相同的时候，**kickstart** 文件非常有用。

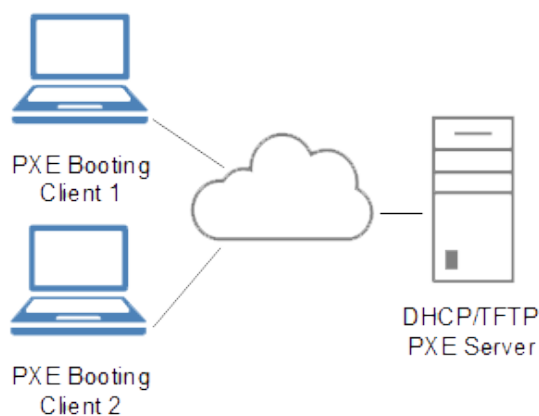


图 14.1 PXE Workflow

PXE 工作流程图：

### 14.0.1 安装相关软件包

```
rpm -ivh dhcpdxxx
rpm -ivh nfs-utilsxxx
rpm -ivh tftp-serverxxx
rpm -ivh xinetdxxx
rpm -ivh httpdxxx
rpm -ivh syslinuxxxx
rpm -ivh system-config-kickstartxxx
```

```
# chkconfig tftp on
# chkconfig xinetd on
```

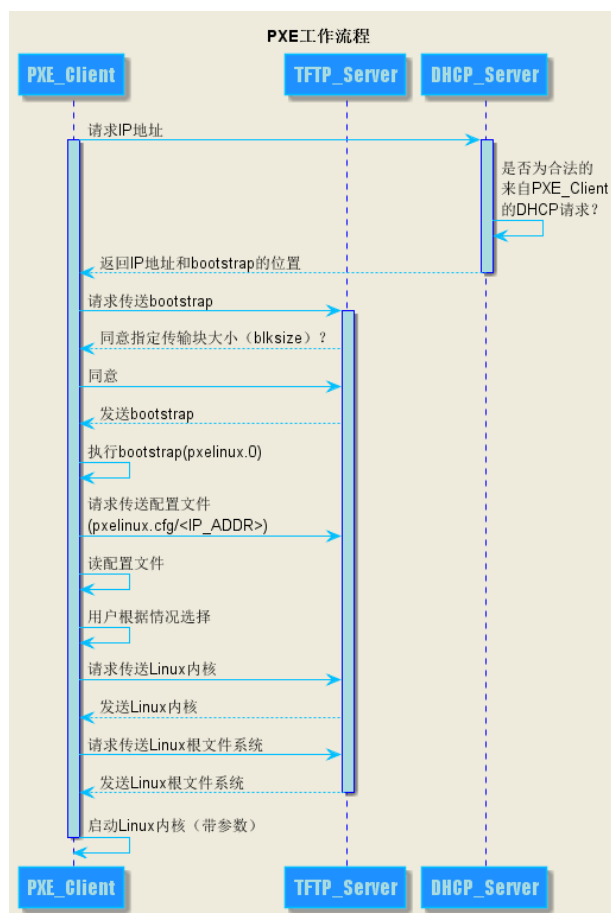


图 14.2 PXE 工作流程图

3. Stop some services, such as selinux, iptables setenforce 0 or vi /etc/sysconfig/selinux disabled service iptables stop

4. Copy the related files to the related path

```

# mount /dev/cdrom /media
# mkdir /var/ftp/pub/RedHat
# cp -a /media/* /var/ftp/pub/RedHat
# cp /usr/share/syslinux/pxelinux.0 /tftpboot
# cp /var/ftp/pub/RedHat/isolinux/vmlinuz /tftpboot
# cp /var/ftp/pub/RedHat/isolinux/initrd.img /tftpboot
# mkdir /tftpboot/pxelinux.cfg
# vi /tftpboot/pxelinux.cfg/default
    default install
    prompt 1
    timeout 60
    label local
localhost 1
    label install
kernel vmlinuz
append initrd=initrd.img ramdisk_size=8192 ks=http://192.168.0.254/ks/ks.conf

# mount /dev/cdrom /media
  
```



```
# mkdir /var/ftp/pub/RedHat
# cp -a /media/* /var/ftp/pub/CentOS
# cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot
# cp /var/ftp/pub/CentOS/isolinux/vmlinuz /var/lib/tftpboot
# cp /var/ftp/pub/CentOS/isolinux/initrd.img /var/lib/tftpboot
# mkdir /var/lib/tftpboot/pxelinux.cfg
# vi /var/lib/tftpboot/pxelinux.cfg/default
default install
prompt 1
timeout 60
label local
localhost 1
label install
kernel vmlinuz
append initrd=initrd.img ramdisk_size=8192 ks=http://192.168.0.254/ks/ks.conf
```

### 5. Modify the related configure files

```
# vi /etc/dhcp/dhcpd.conf
.....
.....
next-server ip_addr;
filename "pxelinux.0";
.....

# vi /etc/http/conf/httpd.conf
Allow from all
# chown -R apache.apache /var/www

# vi /etc/exports
/var/ftp/pub/RedHat 192.168.0.0/255.255.255.0(ro,sync)
```

### 6. Start services, and later the client can install OS

```
# service xinetd restart
# service nfs restart
# service vsftpd restart
# service httpd restart
# service dhcpd restart
```

#### NOTE:

In the ks.cfg, the keyword  
clearpart --none is default  
change this into:  
clearpart --all



## 第十五章 Samba



## 第十六章 Apache

Apache HTTP Server 是 Apache 软件基金会的一个开源 Web 服务器，可以在大多数操作系统中运行，由于其多平台和安全性被广泛使用，是最流行的 Web 服务器软件之一。

Apache 支持许多特性，而这些特性大部分通过编译的模块实现。这些特性从服务器端的编程语言支持到身份认证方案等包括目前所有流行的 Web 服务器应用。由于 Apache 良好的开放性，目前也有很多非官方的模块用以满足某些特殊的应用，在 Apache 2.x 中默认包含的模块如表所示：

### 16.1 安装及配置 Apache



## 第十七章 Nginx

### 17.1 关于 Nginx

Nginx 是由俄罗斯人 Igor Sysoev 为俄罗斯访问量第二 Rambler.ru 站点开发的，它已经在该站点运行超过两年半了。它的发音为“engine X”，是一个高性能的 HTTP 和反向代理服务器，同时也是一个 IMAP/POP3/SMTP 代理服务器。Igor Sysoev 在建立的项目时，使用基于 BSD 许可。自 Nginx 发布四年来，Nginx 已经因为它的稳定性、丰富的功能集、示例配置文件和低系统资源的消耗而闻名。

在俄罗斯许多大网站都已经使用它，且一直表现不凡。截至 2007 年 4 月，俄罗斯大约有 20% 左右的虚拟主机是由 Nginx 服务或代理的。Google 在线安全博客中统计 Nginx 服务或代理了大约所有 Internet 虚拟主机的 4%。而 Netcraft 的统计显示，Nginx 服务的主机在过去的一年里以四倍的速度增长并且在这几年里，它的排名还在不断上升，下图为 Netcraft 截止至 2010 年 5 月的统计。

### 17.2 Nginx 的安装与启动

Linux 下安装软件有三种方式，这里我以源代码编译安装为主。服务器最小化安装后，安装依赖包。

```
yum install -y pcre-devel \  
gcc \  
zlib-devel \  
openssl-devel
```

出于管理和安全的目的，我们希望使用一个指定的普通用户身份去运行我们的 Web 服务器。所以，我们首先增加一个普通用户用于运行我们的 Nginx。

```
groupadd nginx  
useradd -g nginx nginx  
  
service iptables stop  
chkconfig iptables off
```

然后下载、解压并编译安装我们的 Nginx，

```
wget http://nginx.org/download/nginx-1.8.0.tar.gz  
tar -xf nginx-1.8.0.tar.gz -C /usr/local/src
```

```
cd /usr/local/src/nginx-1.8.0
./configure --user=nginx \
> --group=nginx \
> --with-http_ssl_module \
> --with-http_sub_module
```

安装过程比较简单，`./configure` 过程会报出一些依赖关系，这里已经解决。下面来看看 `./configure` 后面几个常用的参数：

```
--prefix=<dir>           指定安装主目录,默认为/usr/local/nginx
--user=<user>             指定用户身份,如果没有指定则默认使用nobody
--group=<group>           指定组身份
--with-http_ssl_module    启用https支持
```

## 17.3 Nginx 的基本配置

Linux 下基本上每个服务都会有它的主配置文件，该文件会定义服务应该如果去运行，使用些什么参数，启用些什么功能，相关会涉及到的一些操作文件在哪，所以主配置文件对服务是至关重要的。下面我们来分析一下 Nginx 的主配置文件。

### 17.3.1 Nginx 主配置概述

Linux 下基本上每个服务都会有它的主配置文件，该文件会定义服务应该如果去运行，使用些什么参数，启用些什么功能，相关会涉及到的一些操作文件在哪，所以主配置文件对服务是至关重要的。Nginx 的主配置文件默认情况下位于 `/usr/local/nginx/conf/nginx.conf`，以下为 Nginx 配置文件一些参数的注释。

```
#user nobody;
#指定使用的用户
worker_processes 1;
#开启的进程数，一般设置 1-5
#error_log logs/error.log;
#error_log logs/error.log notice;
#error_log logs/error.log info;
#定义错误日志，以及记录的日志等级
#pid
logs/nginx.pid;
#定义 pid 文件位置
events {
# use [ kqueue | rtsig | epoll | /dev/poll | select | poll ];
#use epoll; #使用 epoll(linux2.6 的高性能方式)
#Nginx 支持如下处理连接的方法(I/O 复用方法)，这些方法可以通过 use 指令指定。
#select - 标准方法。 如果当前平台没有更有效的方法，它是编译时默认的方法。你
```



数 `-with-select_module` 和 `-without-select_module` 来启用或禁用这个模块。

`#poll` - 标准方法。如果当前平台没有更有效的方法,它是编译时默认的方法。你可以使用 `-with-poll_module` 和 `-without-poll_module` 来启用或禁用这个模块。

`#kqueue` - 高效的方法,使用于 FreeBSD 4.1+, OpenBSD 2.9+, NetBSD 2.0 和 MacOS X。处理器的 MacOS X 系统使用 `kqueue` 可能会造成内核崩溃。

`#epoll` - 高效的方法,使用于 Linux 内核 2.6 版本及以后的系统。在某些发行版本中,如有让 2.4 版本的内核支持 `epoll` 的补丁。

`#rtsig` - 可执行的实时信号,使用于 Linux 内核版本 2.2.19 以后的系统。可是从 Linux 2.6.6-mm2 开始,这个参数就不再使用了。

`#/dev/poll` - 高效的方法,使用于 Solaris 7 11/99+, HP/UX 11.22+ (eventport), IRIX Tru64 UNIX 5.1A+。

`#eventport` - 高效的方法,使用于 Solaris 10。为了防止出现内核崩溃的问题,有必要安装 `worker_connections 1024;`

`#worker_connections 51200;` #每个进程最大连接数(最大连接=连接数 x 进程数)

```

}
http {
include
mime.types;
#文件扩展名与文件类型映射表
default_type application/octet-stream;
#默认文件类型
#log_format main '$remote_addr - $remote_user [$time_local] "$request" '
# '$status $body_bytes_sent "$http_referer" '
# '"$http_user_agent" "$http_x_forwarded_for"';
#access_log logs/access.log main;
sendfile
on;
#开启高效文件传输模式
#tcp_nopush
on;
#该选项用于防止网络阻塞
#keepalive_timeout 0;
keepalive_timeout 65;
##长链接超时时间
#gzip on;
#打开 gzip 压缩
#fastcgi_connect_timeout 300;
#fastcgi_send_timeout 300;
#fastcgi_read_timeout 300;
#fastcgi_buffer_size 128k;
#fastcgi_buffers 4 256k;

```

```
#fastcgi_busy_buffers_size 256k;
#fastcgi_temp_file_write_size 256k;
#fastcgi_temp_path /dev/shm;
#fastcgi 连接超时时间和缓存
server {
listen
80;
server_name localhost;
#主机名
#charset koi8-r;
#默认字符编码 charset gb2312
#access_log logs/host.access.log main;
location / {
#pass 路径匹配 能够匹配路径中带“/”的 不过需要注意的是如果之后也有相关“/”
使用比此处更精确匹配符,则以之后表达式优先
root html;
index index.html index.htm;
}
#error_page 404
/404.html;
# redirect server error pages to the static page /50x.html
#
error_page 500 502 503 504 /50x.html;
location = /50x.html {
#精确的匹配,并且不再向下匹配
root html;
}
#
#location ~ /\.php$ {
#正则表达式匹配 一旦匹配则不再向下匹配
#
proxy_pass http://127.0.0.1;
#}
# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
#location ~ /\.php$ {
# root
html;
# fastcgi_pass 127.0.0.1:9000;
#指定 fastcgi 的地址端口
# fastcgi_index index.php;
```

```
# fastcgi_param SCRIPT_FILENAME /scripts$fastcgi_script_name;
# include fastcgi_params;
#}
# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#
deny all;
#不允许访问以.ht 开头的文件
#}
}
# another virtual host using mix of IP-, name-, and port-based configuration
#
#server {
# listen
8000;
# listen
somename:8080;
# server_name somename alias another.alias;
# location / {
# root html;
# index index.html index.htm;
#
}
#}
#以上在配置虚拟主机
# HTTPS server
#
#server {
# listen 443;
# server_name localhost;
# ssl on;
# ssl_certificate cert.pem;
# ssl_certificate_key cert.key;
# ssl_session_timeout 5m;
# ssl_protocols SSLv2 SSLv3 TLSv1;
# ssl_ciphers ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;
# ssl_prefer_server_ciphers on;

#以上为 ssl 配置
```

```
# location / {  
# root html;  
# index index.html index.htm;  
# }  
# }  
}
```

### 17.3.2 Nginx 虚拟主机配置

利用虚拟主机技术，可以把一台真正的主机分成许多虚拟的主机，每一台虚拟主机都具有独立的域名和 IP 地址，具有完整的 Internet 服务器（www，FTP，email）功能。虚拟主机之间完全独立，在外界看来，每一台虚拟主机和一台独立的主机完全一样。效果一样但费用却大不一样了。由于多台虚拟主机共享一台真实主机的资源，每个虚拟主机用户承受的硬件费用、网络维护费用、通信线路的费用均大幅度降低，Internet 真正成为人人用得起的网络！

虚拟主机共分为三种：基于 IP 的虚拟主机，基于端口的虚拟主机和基于名称的虚拟主机。前两种由于受到成本和客户使用习惯的限制，相对使用的没有基于名称的虚拟主机多，以下我们介绍一下三种虚拟主机的配置。

### 17.3.3 安全的连接 https

众所周知，我们在互联网上冲浪，一般使用的是 http 协议（超文本传输协议），默认情况下数据是明文传送的，这些数据在传输过程中都可能会被捕获和窃听，因此是不安全的。https 可以说是 http 协议的安全版，就是为了满足对安全性要求比较高的用户而设计的。

## 17.4 Nginx 日志管理

## 17.5 Nginx 访问控制

## 17.6 Nginx 反向代理

反向代理（Reverse Proxy）方式是指以代理服务器来接受 Internet 上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给 Internet 上请求连接的客户端，此时代理服务器对外就表现为一个服务器。

反向代理又称为 Web 服务器加速，是针对 Web 服务器提供加速功能的。它作为代理 Cache，但并不针对浏览器用户，而针对一台或多台特定 Web 服务器（这也是反向代理名称的由来）。代理服务器可以缓存一些 web 的页面，降低了 web 服务器的访问量，所以可以降低 web 服务器的负载。web 服务器同时处理的请求数少了，响应时间自然就快了。同时代理服务器也存了一些页面，可以直接返回给客户端，加速客户端浏览。实施反向代理，只要将反向代理设备放置在一台或多台 Web 服务器前端即可。当互联网用户访问某个 WEB 服务器时，通过 DNS 服务器解析后的 IP 地址是代理服务器的 IP 地址，而非

原始 Web 服务器的 IP 地址，这时代理服务器设备充当 Web 服务器，浏览器可以与它连接，无需再直接与 Web 服务器相连。因此，大量 Web 服务工作量被转载到反向代理服务上。不但能够很大程度上减轻 web 服务器的负担，提高访问速度，而且能够防止外部网主机直接和 web 服务器直接通信带来的安全隐患。

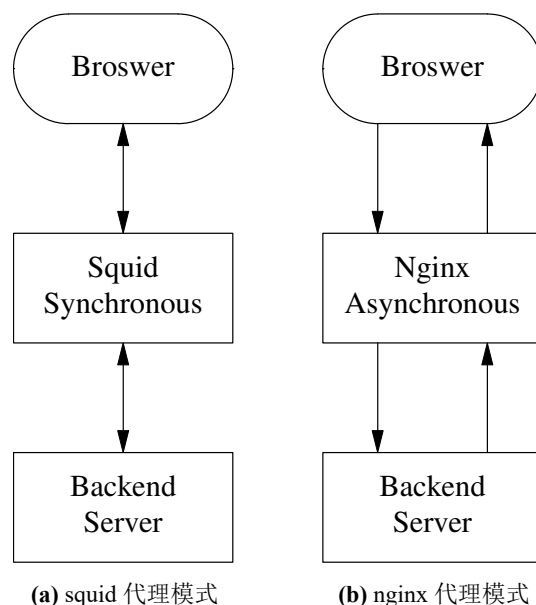


图 17.1 两种代理模式比较

**squid 同步传输：**浏览器发起请求，而后请求会立刻被转到后台，于是在浏览器和后台之间就建立了一个通道。在请求发起直到请求完成，这条通道都是一直存在的。

**nginx 异步传输：**浏览器发起请求，请求不会立刻转到后台，而是将请求数据 (header) 先收到 nginx 上，然后 nginx 再把这个请求发到后端，后端处理完之后把数据返回到 nginx 上，nginx 将数据流发到浏览器，这点和 lighttpd 有点不同，lighttpd 是将后端数据完全接收后才发送到浏览器。

那么这到底有什么好处呢？

1. 假设用户执行一个上传文件操作，因为用户网速又比较慢，因此需要花半个小时才能把文件传到服务器。**squid** 的同步代理在用户开始上传后就和后台建立了连接，半小时后文件上传结束，由此可见，后台服务器连接保持了半个小时；而 **nginx** 异步代理就是先将此文件收到 **nginx** 上，因此仅仅是 **nginx** 和用户保持了半小时连接，后台服务器在这半小时内没有为这个请求开启连接，半小时后用户上传结束，**nginx** 才将上传内容发到后台，**nginx** 和后台之间的带宽是很充裕的，所以只花了一秒钟就将请求发送到了后台，由此可见，后台服务器连接保持了一秒。同步传输花了后台服务器半个小时，异步传输只花一秒，可见优化程度很大。
2. 在上面这个例子中，假如后台服务器因为种种原因重启了，上传文件就自然中断了，这对用户来说是非常恼火的一件事情，想必我们也有上传文件传到一半被中断的经历。用 **nginx** 代理之后，后台服务器的重启对用户上传的影响减少到了极点，而 **nginx** 是非常稳定的并不需要常去重启它，即使需要重启，利用 `kill -HUP` 就可以做到不间断重启 **nginx**。

3. 异步传输可以令负载均衡器更有保障,为什么这么说呢?在其它的均衡器(lvs/haproxy/apache等)里,每个请求都是只有一次机会的,假如用户发起一个请求,结果该请求分到的后台服务器刚好挂掉了,那么这个请求就失败了;而nginx因为是异步的,所以这个请求可以重新发往下一个后台,下一个后台返回了正常的数据,于是这个请求就能成功了。还是用用户上传文件这个例子,假如不但用了nginx代理,而且用了负载均衡,nginx把上传文件发往其中一台后台,但这台服务器突然重启了,nginx收到错误后,会将这个上传文件发到另一台后台,于是用户就不用再花半小时上传一遍。
4. 假如用户上传一个10GB大小的文件,而后台服务器没有考虑到这个情况,那么后台服务器岂不要崩溃了。用nginx就可以把这些东西都拦在nginx上,通过nginx的上传文件大小限制功能来限制,另外nginx性能非常有保障,就放心的让互联网上那些另类的用户和nginx对抗去吧。

### 17.6.1 Nginx 与 Lua 结合

# 第十八章 LAMP

## 18.1 安装依赖包

Needless to say, you should install required system packages. So you can continue the next steps. :-)

Be sure these packages are installed:

```
gcc
gcc-c++
flex
bison
autoconf
automake
bzip2-devel
ncurses-devel
libjpeg-devel
libpng-devel
libtiff-devel
freetype-devel
pam-devel
```

## 18.2 安装额外的包

In this section, we need to compile and install four packages:

### 1. GD2

```
[root@lamp ~] # cd /usr/local/src
[root@lamp src] # tar xzvf gd-2.0.34.tar.gz
[root@lamp src] # cd gd-2.0.34
[root@lamp gd-2.0.34] # ./configure --prefix=/usr/local/gd2
[root@lamp gd-2.0.34] # make
[root@lamp gd-2.0.34] # make install
```

### 2. LibXML2

```
[root@lamp ~] # cd /usr/local/src
[root@lamp src] # tar xzvf libxml2-2.6.29.tar.gz
[root@lamp src] # cd libxml2-2.6.29
[root@lamp libxml2-2.6.29] # ./configure --prefix=/usr/local/libxml2
```

```
[root@lamp libxml2-2.6.29] # make
[root@lamp libxml2-2.6.29] # make install
```

### 3. LibMcrypt

```
[root@lamp ~] # cd /usr/local/src
[root@lamp src] # tar xjvf libmcrypt-2.5.8.tar.bz2
[root@lamp src] # cd libmcrypt-2.5.8
[root@lamp libmcrypt-2.5.8] # ./configure --prefix=/usr/local/libmcrypt
[root@lamp libmcrypt-2.5.8] # make
[root@lamp libmcrypt-2.5.8] # make install
```

### 4. OpenSSL

```
[root@lamp ~] # cd /usr/local/src
[root@lamp src] # tar xzvf openssl-0.9.8e.tar.gz
[root@lamp src] # cd openssl-0.9.8e
[root@lamp openssl-0.9.8e] # ./config --prefix=/usr/local/openssl
[root@lamp openssl-0.9.8e] # make
[root@lamp openssl-0.9.8e] # make test
[root@lamp openssl-0.9.8e] # make install
```

## 18.3 编译安装 MySQL

```
[root@lamp ~] #tar xzvf mysql-5.0.27.tar.gz
[root@lamp ~] # cd mysql-5.0.27
[root@lamp mysql-5.0.27] # ./configure \
"--prefix=/usr/local/mysql" \
"--localstatedir=/var/lib/mysql" \
"--with-mysqld-user=mysql" \
"--without-debug" \
"--with-big-tables" \
"--with-extra-charsets=all" \
"--with-pthread" \
"--enable-static" \
"--enable-thread-safe-client" \
"--with-client-ldflags=-all-static" \
"--with-mysqld-ldflags=-all-static" \
"--enable-asm" \
"--without-isam" \
"--without-innodb" \
"--without-ndb-debug"

[root@lamp msyql-5.0.27] # make
[root@lamp mysql-5.0.27] # make install
[root@lamp mysql-5.0.27] # useradd mysql
[root@lamp mysql-5.0.27] # cd /usr/local/mysql
[root@lamp mysql] # bin/mysql_install_db --user=mysql
[root@lamp mysql] # chown -R root:mysql .
```



```
[root@lamp mysql] # chown -R mysql /var/lib/mysql
[root@lamp mysql] # cp share/mysql/my-huge.cnf /etc/my.cnf
[root@lamp mysql] # cp share/mysql/mysql.server /etc/rc.d/init.d/mysqld
[root@lamp mysql] # chmod 755 /etc/rc.d/init.d/mysqld
[root@lamp mysql] # chkconfig --add mysqld
[root@lamp mysql] # chkconfig --level 3 mysqld on
[root@lamp mysql] # /etc/rc.d/init.d/mysqld start
[root@lamp mysql] # bin/mysqladmin -u root password 'clear123'
```

## 18.4 编译安装 Apache

```
[root@lamp src] # cd /usr/local/src
[root@lamp src] # tar xjvf httpd-2.2.4.tar.bz2
[root@lamp src] # cd httpd-2.2.4
[root@lamp httpd-2.2.4] # ./configure \
"--prefix=/usr/local/apache2" \
"--with-included-apr" \
"--enable-so" \
"--enable-deflate=shared" \
"--enable-expirers=shared" \
"--enable-rewrite=shared" \
"--enable-static-support" \
"--disable-userdir"
[root@lamp httpd2.2.4] # make
[root@lamp httpd2.2.4] # make install
[root@lamp httpd2.2.4] # echo '/usr/local/apache2/bin/apachectl start' \
> >> /etc/rc.local
```

## 18.5 编译安装 PHP

```
[root@lamp ~] # cd /usr/local/src
[root@lamp src] # tar xjvf php-5.2.3.tar.bz2
[root@lamp php-5.2.3] # cd php-5.2.3
[root@lamp php-5.2.3] # ./configure \
"--prefix=/usr/local/php" \
"--with-apxs2=/usr/local/apache2/bin/apxs" \
"--with-config-file-path=/usr/local/php/etc" \
"--with-mysql=/usr/local/mysql" \
"--with-libxml-dir=/usr/local/libxml2" \
"--with-gd=/usr/local/gd2" \
"--with-jpeg-dir" \
"--with-png-dir" \
"--with-bz2" \
"--with-freetype-dir" \
"--with-iconv-dir" \
"--with-zlib-dir" \
```

```

"--with-openssl=/usr/local/openssl" \
"--with-mcrypt=/usr/local/libmcrypt" \
"--enable-soap" \
"--enable-gd-native-ttf" \
"--enable-memory-limit" \
"--enable-ftp" \
"--enable-mbstring" \
"--enable-exif" \
"--disable-ipv6" \
"--disable-cgi" \
"--disable-cli"
[root@lamp php-5.2.3] # make
[root@lamp php-5.2.3] # make install
[root@lamp php-5.2.3] # mkdir /usr/local/php/etc
[root@lamp php-5.2.3] # cp php.ini-dist /usr/local/php/etc/php.ini

```

## 18.6 安装 Zend 加速器

```

[root@lamp ~]# cd /usr/local/src
[root@lamp ~]# tar xf ZendOptimizer-3.2.8-linux-glibc21-i386.tar.gz
[root@lamp ~]# ./ZendOptimizer-3.2.8-linux-glibc21-i386/install.sh

```

## 18.7 整合 Apache 与 PHP

We should modify configure file httpd.conf:

```
[root@lamp ~]# emacs /usr/local/apache2/conf/httpd.conf
```

Find this line,

```
AddType application/x-gzip .gz .tgz
```

Add on line under this line,

```
AddType application/x-httpd-php .php
```

Find these lines,

```

<IfModule dir_module>
DirectoryIndex index.html
</IfModule>

```

Change them like this,

```

<IfModule dir_module>
DirectoryIndex index.html index.htm index.php
</IfModule>

```

Find these lines and uncomment them,

```
#Include conf/extra/httpd-mpm.conf
#Include conf/extra/httpd-info.conf
#Include conf/extra/httpd-vhosts.conf
#Include conf/extra/httpd-default.conf
```

When finished, save it! Then restart apache service,

```
[root@lamp ~]# /usr/local/apache2/bin/apachectl restart
```



## 第十九章 多网卡绑定 bonding

Linux bonding 提供将多个网络接口设备捆绑为单个网络接口设置来使用，用于网络负载均衡及网络冗余。

以太网平衡设备即使用两个或两个以上的网络接口模拟一个虚拟的网络接口用以外部网络连接。这多个真实网络接口可以连接在同一交换设备或多个交换设备上，以达到多通路高可用性。在所有真实网络接口中可以指定其中某些真实网络接口活跃，其他真实网络接口在活跃的真实网络接口故障时接管网络传输；也可以指定所有真实网络接口活跃，分担全部网络传输带宽。

通道绑定 (Channel bonding) 需要服务器至少拥有 2 个以太网卡，当使用绑定功能的时候，bonding 模块会使用第一个实际网卡的 MAC 地址来通信，在侦测到这个网卡失败以后，它会把这个 MAC 地址指定到另一块网卡上。

### 19.1 bonding 的几种模式

bonding 只能提供链路监测，即从主机到交换机的链路是否接通。如果只是交换机对外的链路 down 掉了，而交换机本身并没有故障，那么 bonding 会认为链路没有问题而继续使用。

miimon 是用来进行链路监测的。比如:miimon=100，那么系统每 100ms 监测一次链路连接状态，如果有一条线路不通就转入另一条线路；mode 的值表示工作模式，他共有 0, 1, 2, 3, 4, 5, 6 七种模式，作者遇到的场景是使用的 1, 0, 6 三种，其他场景适合哪些模式作者也不清楚。

mode=1，表示 fault-tolerance (active-backup) 提供网络冗余功能，工作方式是主备模式，也就是说默认情况下只有一块网卡工作，另一块做备份。

mode=0，表示 load balancing (round-robin) 为负载均衡模式，两块网卡都工作，但是与网卡相连的交换机必须做特殊配置（这两个端口应该采取聚合方式），因为做 bonding 的这两块网卡是使用同一个 MAC 地址。

mode=6，表示 load balancing (round-robin) 为负载均衡方式，两块网卡都工作，但是该模式下无需配置交换机，因为做 bonding 的这两块网卡是使用不同的 MAC 地址。

### 19.2 RHEL 下配置 bonding

华为 RH1288 服务器共有 8 块网卡，eth0 与 eth1 做 bond0，对应管理网段 172.16.25.X，主备模式；eth2 与 eth3 做 bond1，此网段暂没有配置 IP 信息，主备模式；eth4 与 eth5 做 bond2，对应存储网段 10.10.2.X，主备模式。

编辑/etc/modprobe.d/dist.conf 文件，在文件末尾添加如下几行，开启 bonding 功能<sup>1</sup>。

```
# vim /etc/modprobe.d/dist.conf
alias bond0 bonding
options bond0 mode=1 miimon=100

alias bond1 bonding
options bond1 mode=1 miimon=100

alias bond2 bonding
options bond2 mode=1 miimon=100
```

编辑虚拟网络接口配置文件 bond0，

```
# vim /etc/sysconfig/network-scripts/ifcfg-bond0
DEVICE=bond0
ONBOOT=yes
BOOTPROTO=none
NETWORK=
IPADDR=
NETMASK=
GATEWAY=
USERCTL=no
```

分别编辑 eth0 与 eth1 网络接口文件 ifcfg-eth0 与 ifcfg-eth1，

```
# vim /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
BOOTPROTO=none
TYPE=Ethernet
ONBOOT=yes
USERCTL=no
MASTER=bond0
SLAVE=yes
```

```
# vim /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
BOOTPROTO=none
TYPE=Ethernet
ONBOOT=yes
USERCTL=no
MASTER=bond0
SLAVE=yes
```

---

<sup>1</sup>这是 RHEL6U4 的系统，如果是 RHEL5 系列的机器，配置文件则为/etc/modprobe.conf

由于该机器对应的交换机接口为 VLAN 210, 所以, 需要在 `bond0` 接口上配置 VLAN。

```
# vconfig add bond0 210
```

这样就产生了一个 `bond0.210` 的虚拟网络接口, 这样就可以启用该接口, 并配置 IP 信息,

```
# ifconfig bond0.210 up
# vim /etc/sysconfig/network-scripts/ifcfg-bond0.210
DEVICE=bond0.210
BOOTPROTO=none
ONBOOT=yes
USERCTL=no
IPADDR=172.16.25.93
NETMASK=255.255.255.0
GATEWAY=172.16.25.254
VLAN=yes
```

查看上面已配置的 VLAN 信息,

```
# cat /proc/net/vlan/config
VLAN Dev name | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
bond0.210      | 210 | bond0

# cat /proc/net/vlan/bond0.210
bond0.210  VID: 210  REORDER_HDR: 1  dev->priv_flags: 1
            total frames received      4663
            total bytes received      224809
Broadcast/Multicast Rcvd              38

            total frames transmitted    228
            total bytes transmitted    44140
            total headroom inc          0
            total encap on xmit         0
Device: bond0
INGRESS priority mappings: 0:0  1:0  2:0  3:0  4:0  5:0  6:0  7:0
EGRESS priority mappings:
```

## 19.3 SuSE 下配置 bonding

`bond0` 配置:

```
# cat /etc/sysconfig/network/ifcfg-bond0
DEVICE='bond0'
ONBOOT='yes'
BOOTPROTO='static'
IPADDR='0.0.0.0/24'
STARTMODE='auto'
BONDING_MASTER='yes'
BONDING_MODULE_OPTS='mode=1 miimon=100'
BONDING_SLAVE0='eth0'
BONDING_SLAVE1='eth1'
USERCONTROL='no'
```

**br0 配置:**

```
# cat /etc/sysconfig/network/ifcfg-br0
BOOTPROTO='static'
BRIDGE='yes'
BRIDGE_FORWARDDELAY='0'
BRIDGE_PORTS='bond1'
BRIDGE_STP='off'
IPADDR='0.0.0.0/24'      ##管理网ip
STARTMODE='auto'
USERCONTROL='no'
```

```
# cat /etc/sysconfig/network/ifcfg-vlan0      ##为网卡打Vlan Tag
BOOTPROTO='static'
ETHERDEVICE='br0'
IPADDR='145.240.21.11/24'
STARTMODE='auto'
USERCONTROL='no'
VLAN_ID='210'                ##上联核心交换机端口的Vlan号
```

```
# cat /proc/net/vlan/config      ##验证Vlan是否设置成功
VLAN Dev name | VLAN ID
Name-Type: VLAN_NAME_TYPE_RAW_PLUS_VID_NO_PAD
vlan0          | 210 | br0
```

## 19.4 单网卡多 IP 配置



## 第二十章 RAID 技术

RAID 技术有各种级别之分，包括 RAID0、RAID1、RAID2、RAID3、RAID4、RAID5、RAID5E、RAID5EE、RAID6、RAID10 等。作者接触最多的是 RAID0、RAID1、RAID5 及 RAID10 这些 RAID 级别，其他级别在实际工作当中并没有见到，这里就不在介绍。

### 20.1 RAID 基础知识

RAID 最初是 1987 年在加利福尼亚大学进行的一个科研项目，后来由伯克利分校的 D.A. Patterson 教授在 1988 年正式提出。

RAID 是 Redundant Array of Inexpensive Disks 的缩写，直译为“廉价冗余磁盘阵列”，最初是为了组合多块小容量的廉价磁盘来代替大容量的昂贵磁盘，同时希望在磁盘失效时不会对数据造成影响而开发出的一种磁盘存储技术。

后来随着磁盘研发技术的不断提升，硬盘容量越来越大，成本却不断下降，所以 RAID 中“*Inexpensive*（廉价）”一词已经失去意义，于是将这个词用“*Independent*（独立）”来替代，RAID 就成了“独立冗余磁盘阵列”，也简称为“磁盘阵列”，但这只是名称的变化，实质性的内容并没有改变。

#### 20.1.1 RAID 解决了什么问题

通俗地说，RAID 就是通过将多个磁盘按照一定的形式和方案组织起来，通过这样的形式能够获取比单个磁盘更高的速度、更好的稳定性、更大的存储能力的存储解决方案，我们不必关心磁盘阵列究竟由多少块磁盘组成，使用中整个阵列就如同一块磁盘一样。所以，RAID 技术能够为计算机系统提供以下三个方面的优异性能：

##### 1. 提供更大的存储空间

使用 RAID 技术，就可以把多块磁盘组成一个更大的存储空间供我们使用。比如，利用 RAID0 技术把 5 块 1TB 的硬盘组织起来，能够提供 5TB 的存储空间。

##### 2. 提供更快的传输速度

著名的摩尔定律告诉我们，CPU 的性能每隔 18 个月就会提高一倍，可见其速度增长之快。然而，机械硬盘作为计算机中最重要的存储设备，在容量飞速增长的同时，速度却提高缓慢，已经成为计算机速度发展的瓶颈。如果采用 RAID 技术，可以让很多机械硬盘同时传输数据，而这些硬盘在逻辑上又表现为一块硬盘，所以使用 RAID 可以达到单个磁盘几倍、甚

至 N 多倍的速率。也就是说，RAID 技术可以通过在多个磁盘上同时存储和读取数据的方式来大幅提高存储系统的数据吞吐量。

### 3. 提高更高的安全性

RAID 可以通过数据校验提供容错功能，在很多 RAID 模式中都有较为完备的冗余措施，甚至是直接相互的镜像备份，从而大大提高来 RAID 系统的容错性，让系统的稳定性更好、安全性更高。

## 20.2 RAID 实现方式

### 20.2.1 RAID0 数据组织原理

RAID0 是无冗余、无校验的磁盘阵列，实现 RAID0，至少需要两个以上硬盘，它将两个以上的硬盘合并成一块，数据同时分散在每块磁盘中，因为带宽加倍，所以读写速度加倍，RAID0 的理论速度是单块硬盘的 N 倍，但是由于数据并不是保存在一个硬盘上，而是分成数据块保存在不同的硬盘上，所以安全性也下降 N 倍，只要任何一块磁盘损坏就会丢失所有数据。

RAID0 是最简单的一种 RAID 形式，目的是把多块物理盘连接在一起形成一个容量更大的存储设备，RAID0 逻辑盘的容量等于物理盘的容量乘以成员盘的数目。

原理图

RAID0 只是单纯地提高读写性能，并没有数据的可靠性提供保证，而且其中的任何一个物理盘失效都将影响到所有数据。因此，RAID0 不能用于数据安全性要求高的场合。

### 20.2.2 RAID1 数据组织原理

RAID1 通过磁盘数据镜像实现数据的冗余，在两块磁盘上产生互为备份的数据，当其中一块成员盘出现故障时，系统还可以从另外一块成员盘中读取数据。因此，RAID1 可以提供更好的冗余性。

RAID1 又称为磁盘镜像，需要在两个物理盘共同构建，使用磁盘镜像技术，方法是在工作磁盘之外再加一额外的备份磁盘，两个磁盘所存储的数据完全一样，数据写入工作磁盘的同时亦写入备份磁盘，也就是将一块物理盘的内容完全复制到另一块物理盘上。所以，两块物理盘所构成的 RAID1 阵列，其容量仅等于一块磁盘的容量，其数据分布情况如图所示。

原理图

RAID1 是磁盘阵列中单位成本最高的，但提供来很高的数据安全性和可用性。当一个物理盘失效时，系统可以自动切换到镜像盘上读写，而不需要重组失效的数据。

RAID1 是所有 RAID 等级中实现成本最高的一种，尽管如此，我们还是选择 RAID1 来保存那些关键性的重要数据。

### 20.2.3 RAID10 数据组织原理

### 20.2.4 RAID5 数据组织原理

## 20.3 MegaRAID Cli 工具基本使用

我们都是使用过 LSI 的 Web 界面去配置 RAID，虽听起来很高大上，但整个配置过程是那么的令人蛋疼。配置完成后，我们还要按“Ctrl+Alt+Del”组合键来重启机器，步骤虽有些繁琐，但仍能令人接受。MegaRAID Cli 工具是在命令行模式下操作 RAID 控制器的，它的优点之一就是做完 RAID 之后，可以直接使用并不需要重启操作系统，而且操作简单方便。

写这一节的目的并不是推荐使用该工具，而是熟悉了原来的配置界面，不愿意再学新的东西。若不是同事的提示，还不知有这种很 xx 的工具，使用起来确实很酷，愿意跟大家分享一下使用过程。

要想使用该工具，首先系统上要安装相应的软件包，这里省略安装过程。安装完毕之后，工具默认会安装在/opt/MegaRAID/MegaCli 的目录下。

本次使用该命令行工具的场景：新到了两块 Intel 的 SATA 接口 400GB 的 SSD 硬盘，欲测其性能。原来的服务器上自带了 4 块 600GB 的磁盘并做了 RAID10，这四块磁盘分别占据了第 0、1、2、3 这四个硬盘插槽，在第 4、第 5 个插槽放置了 SSD 盘，操作系统使用的是 SLES 11.2。下面是具体的操作过程，

### 20.3.1 制作 RAID

#### 1. 查看 RAID 卡的设备号

```
# cd /opt/MegaRAID/MegaCli
# ./MegaCli64 -PDList -aAll |grep "Device ID"
Enclosure Device ID: 252
Enclosure Device ID: 252
Enclosure Device ID: 252
Enclosure Device ID: 252
Enclosure Device ID: 252
Enclosure Device ID: 252
```

说明：上面的输出，表示我们有一个 RAID 卡，因为这些 ID 是一样的。这个卡下面有 6 个盘，这里的 ID 号需要记下来，后面做 RAID 时需要用到。可以把该 RAID 的 ID 号理解为主设备号。

#### 2. 查看 Slot ID 以确认有无错序的情况

```
# ./MegaCli64 -PDList -aAll |grep "Slot"
Slot Number: 0
Slot Number: 1
Slot Number: 2
Slot Number: 3
Slot Number: 4
```

Slot Number: 5

说明：这里的 Slot Number 号需要记下来，后面做 RAID 时需要用到。可以把该 Slot 的 ID 理解为次设备号。

### 3. 查看 Foreign 信息

```
# ./MegaCli64 -PDList -aAll |grep "Foreign State"
Foreign State: None
Foreign State: None
Foreign State: None
Foreign State: None
Foreign State: Foreign
Foreign State: Foreign
```

说明：状态显示为“Foreign”的磁盘，说明是新添加进来的或者是未使用的。这两个为“Foreign”状态的正是我们新添加的 SSD 盘。接下来的操作就是清除这些“Foreign”状态的盘。

### 4. 清除盘的 Foreign 信息

```
# ./MegaCli64 -CfgForeign -Clear -a0
```

### 5. 新做 RAID，在 Slot4 和 Slot5 上做 RAID0

```
# ./MegaCli64 -CfgLdAdd -r0 [252:4,252:5] WT Direct -a0
```

说明：如果做 RAID1，只需要把 r0 改为 r1 即可。

## 20.3.2 删除 RAID

当测试完毕 RAID0 级别的 SSD 盘时，要开始测试 RAID1 级别下 Intel SATA SSD 的性能。所以，之前制作的 RAID0 要被删除了。在删除之前，我们需要知道被删除的 Target Id，然后方可删除该组 RAID。

查看有多少个 RAID 级别，找到我们想要删除的 Target Id，其中“Target Id: n”，n 即为第 n 组 RAID。

```
# ./MegaCli64 -LdInfo -Lall -aALL
```

删除阵列，

```
# ./MegaCli64 -CfgLdDel -L1 -a0
```

附录：名词解释磁盘缓存策略：

WT (Write Through)

WB (Write Back)

NORA (NO Read Ahead)

RA (Read Ahead)

ADRA (Adaptive Read Ahead)

## 第四部分

### 集群方案篇



本章介绍几种高可用的解决方案。





## 第二十一章 集群基础知识

### 21.1 集群概述

集群是一组协同工作的服务实体，用以提供比单一服务实体更具扩展性和可用性的服务平台。

在客户端看来，一个集群就是一个完整不可细分的实体，但事实上一个集群实体是由完成不同任务的服务节点个体所组成的。

集群实体的可扩展性是指，在集群运行中新的服务节点可以动态的加入集群实体从而提升集群实体的综合性能。

集群实体的高可用性是指，集群实体通过其内部的服务节点的冗余使客户端免于 OUT OF SERVICE 错误。简单的说，在集群中同一服务可以由多个服务节点提供，当部分服务节点失效后，其他服务节点可以接管服务。

集群实体地址是指客户端访问集群实体获得服务资源的唯一入口地址。

负载均衡是指集群中的分发设备（服务）将用户的请求任务比较均衡（不是平均）分发到集群实体中的服务节点计算、存储和网络资源中。一般我们将提供负载均衡分发的设备叫做负载均衡器。负载均衡器一般具备如下三个功能：

1. 维护集群地址
2. 负责管理各个服务节点的加入和退出
3. 集群地址向内部服务节点地址的转换

错误恢复是指集群中某个节点或某些服务节点（设备）不能正常工作（或提供服务），其他类似服务节点（设备）可以资源透明和持续的完成原有任务。具备错误恢复能力是集群实体高可用性的必要条件。

负载均衡和错误恢复都需要集群实体中各个服务节点中有执行同一任务的资源存在，而且对于同一任务的各个资源来说，执行任务所学的信息视图必须一致。

### 21.2 集群类型

多台同构或异构的计算机用某种方式连接起来协同完成特定的任务就构成了集群系统，目前 Linux 下的集群主要有三种类型：

1. HA（High Availability）
2. LB（Load Balancing）
3. HPC（High Performance Computing）



## 第二十二章 Keepalived

### 22.1 Keepalived 简介

KeepAlived 起初是专为 LVS 设计的，专门用来监控 LVS 集群系统中各个服务节点的状态，后来又加入了 VRRP 的功能，因此，除了配合 LVS 服务外，也可以作为其他服务（如 Nginx，HAProxy 等）的高可用软件。VRRP 是 Virtual Router Redundancy Protocol（虚拟路由冗余协议）的缩写，VRRP 出现的目的就是为了解决静态路由出现的单点故障问题，它能够保证网络的不间断、稳定地运行。所以，KeepAlived 一方面具有 LVS cluster nodes healthcheck 功能，另一方面也具有 LVS directors failover 功能。

### 22.2 Keepalived 安装部署

#### 22.2.1 环境准备

演示环境为 CentOS6.5 64 位系统，机器列表如下：

表 22-1 KeepAlived 演示环境机器列表

主机名	IP 地址	角色
lb01.lavenliu.com	192.168.20.150	KeepAlived（主）
lb02.lavenliu.com	192.168.20.151	KeepAlived（备）

#### 22.2.2 开始安装

```
yum install -y keepalived
```

#### 22.2.3 Keepalived 配置介绍

### 22.3 运行服务与故障模拟



## 第二十三章 LVS+Keepalived 负载均衡集群

LVS(Linux Virtual Server) is a cluster of servers

The Linux Virtual Server can be used to build scalable network services based on a cluster of two or more nodes. The active node of the cluster redirects service requests to a collection of server hosts that will actually perform the services. Supported features include two protocols (TCP and UDP), three packet-forwarding methods (NAT, tunneling, and direct routing), and eight load balancing algorithms (round robin, weighted round robin, least-connection, weighted least-connection, locality-based least-connection, locality-based least-connection with replication, destination-hashing, and source-hashing).

LVS: ipvsadm/ipvs

When a new connection is requested from a client to a service provided by the LVS (e.g. httpd), the director will choose a realserver from the client.

From then, all packets from the client will go through the director to that particular realserver.

The association between the client and the realserver will last for only the life of the tcp connection (or udp exchange).

For the next tcp connection, the director will choose a new realserver (which may or may not be the same as the first realserver).

Thus a web browser connecting to a LVS serving a webpage consisting of several hits (images, html page), may get each hit from a separate realserver.

LVS IP Address Name Conventions

Virtual IP (VIP) address: The IP address the Director uses to offer services to client computers

Real IP (RIP) address: The IP address used on the cluster nodes

Director's IP (DIP) address: The IP address the Director uses to connect to the D/RIP network

Client computer's IP (CIP) address: The IP address assigned to a client computer that it uses as a source IP address for requests sent to the cluster

Basic Properties of LVS-NAT

1. The cluster nodes need to be on the same network (VLAN or subnet) as the Director. 集群节点必须与调度器在同一个网络中
2. The RIP addresses of the cluster nodes are normally private, non-routable IP addresses used only for intracluster communication. RIP 通常是私有地址，仅用

于各集群节点间的通信

3. director 位于 client 与 real server 之间，并负责处理进出的所有通信
  4. realserver 必须将网关指向 DIP
  5. 支持端口映射
  6. real server 可以使用任何操作系统
  7. 较大规模应用场景中，director 易成为系统瓶颈
1. 集群节点跟 director 必须在同一物理网络中
  2. RIP 可以使用公网地址，实现便捷的远程管理和监控
  3. director 仅负责处理入站请求，响应报文则由 real server 直接发往客户端
  4. real server 不能讲网关指向 DIP
  5. 不支持端口映射

real server 必须能够隐藏 VIP

TUN: 1. 集群节点可以跨越 Internet 2. RIP 必须是公网地址 3. Director 仅处理入站请求，响应报文则由 real server 直接发往客户端 4. real server 网关不能指向 director 5. 只有支持隧道功能的 os 才能用于 real server 6. 不支持端口映射

## 23.1 LVS 调度算法

Director 在接收到来自于 Client 的请求时，会基于“schedule”从 RealServer 中选择一个响应给 Client。ipvs 支持以下调度算法：

1. 轮询 (round robin, rr), 加权轮询 (Weighted round robin, wrr)  
新的连接请求被轮流分配至各 RealServer；算法的优点是其简洁性，它无需记录当前所有连接的状态，所以它是一种无状态调度。轮叫调度算法假设所有服务器处理性能均相同，不管服务器的当前连接数和响应速度。该算法相对简单，不适用于服务器组中处理性能不一的情况，而且当请求服务时间变化比较大时，轮叫调度算法容易导致服务器间的负载不平衡。
2. 最少连接 (least connected, lc), 加权最少连接 (weighted least connection, wlc)  
新的连接请求将被分配至当前连接数最少的 RealServer；最小连接调度是一种动态调度算法，它通过服务器当前所活跃的连接数来估计服务器的负载情况。调度器需要记录各个服务器已建立连接的数目，当一个请求被调度到某台服务器，其连接数加 1；当连接中止或超时，其连接数减一。
3. 基于局部性的最少链接调度 (Locality-Based Least Connections Scheduling, lbic)  
针对请求报文的目标 IP 地址的负载均衡调度，目前主要用于 Cache 集群系统，因为在 Cache 集群中客户请求报文的目标 IP 地址是变化的。这里假设任何后端服务器都可以处理任一请求，算法的设计目标是在服务器的负载基本平衡情况下，将相同目标 IP 地址的请求调度到同一台服务器，来提高各台服务器的访问局部性和主存 Cache 命中率，从而整个集群系统的处理能力。LBLC 调度算法先根据请求的目

标 IP 地址找出该目标 IP 地址最近使用的服务器, 若该服务器是可用的且没有超载, 将请求发送到该服务器; 若服务器不存在, 或者该服务器超载且有服务器处于其一半的工作负载, 则用“最少链接”的原则选出一个可用的服务器, 将请求发送到该服务器。

#### 4. 带复制的基于局部性最少链接调度 (Locality-Based Least Connections with Replication Scheduling, lblcr)

也是针对目标 IP 地址的负载均衡, 目前主要用于 Cache 集群系统。它与 LBLC 算法的不同之处是它要维护从一个目标 IP 地址到一组服务器的映射, 而 LBLC 算法维护从一个目标 IP 地址到一台服务器的映射。对于一个“热门”站点的服务请求, 一台 Cache 服务器可能会忙不过来处理这些请求。这时, LBLC 调度算法会从所有的 Cache 服务器中按“最小连接”原则选出一台 Cache 服务器, 映射该“热门”站点到这台 Cache 服务器, 很快这台 Cache 服务器也会超载, 就会重复上述过程选出新的 Cache 服务器。这样, 可能会导致该“热门”站点的映像会出现在所有的 Cache 服务器上, 降低了 Cache 服务器的使用效率。LBLCR 调度算法将“热门”站点映射到一组 Cache 服务器 (服务器集合), 当该“热门”站点的请求负载增加时, 会增加集合里的 Cache 服务器, 来处理不断增长的负载; 当该“热门”站点的请求负载降低时, 会减少集合里的 Cache 服务器数目。这样, 该“热门”站点的映像不太可能出现在所有的 Cache 服务器上, 从而提供 Cache 集群系统的使用效率。LBLCR 算法先根据请求的目标 IP 地址找出该目标 IP 地址对应的服务器组; 按“最小连接”原则从该服务器组中选出一台服务器, 若服务器没有超载, 将请求发送到该服务器; 若服务器超载; 则按“最小连接”原则从整个集群中选出一台服务器, 将该服务器加入到服务器组中, 将请求发送到该服务器。同时, 当该服务器组有一段时间没有被修改, 将最忙的服务器从服务器组中删除, 以降低复制的程度。

#### 5. 目标地址散列调度 (Destination Hashing, dh)

算法也是针对目标 IP 地址的负载均衡, 但它是一种静态映射算法, 通过一个散列 (Hash) 函数将一个目标 IP 地址映射到一台服务器。目标地址散列调度算法先根据请求的目标 IP 地址, 作为散列键 (Hash Key) 从静态分配的散列表找出对应的服务器, 若该服务器是可用的且未超载, 将请求发送到该服务器, 否则返回空。

#### 6. 源地址散列调度 (Source Hashing, sh)

算法正好与目标地址散列调度算法相反, 它根据请求的源 IP 地址, 作为散列键 (Hash Key) 从静态分配的散列表找出对应的服务器, 若该服务器是可用的且未超载, 将请求发送到该服务器, 否则返回空。它采用的散列函数与目标地址散列调度算法的相同。除了将请求的目标 IP 地址换成请求的源 IP 地址外, 它的算法流程与目标地址散列调度算法的基本相似。在实际应用中, 源地址散列调度和目标地址散列调度可以结合使用在防火墙集群中, 它们可以保证整个系统的唯一出入口。

基于 DNS 的负载均衡方案性能可能会出现问题。DNS 的记录会缓存。

rsync 基于文件的同步, 效率低。

drbd 磁盘镜像, 让两个计算机的两块磁盘做镜像, 基于块级别的同步, 效率高。

## 23.2 安装 LVS

### 23.2.1 环境准备



## 第二十四章 Heartbeat 高可用集群

Heartbeat 提供了诸多集群基础架构服务，比如集群之间的消息传递、节点成员身份、IP 地址分配和迁移，以及服务的开启和停止。Heartbeat 可以用来为 Apache、Samba 和 Squid 等企业应用系统构建几乎任何一种高可用性的集群。此外，它可以结合负载均衡软件使用，那样入站请求就可以由所有集群节点来分担。

本文中的示例集群将由 2 台运行 Heartbeat 的服务器组成。我们测试故障切换机制的方法是，手动关闭服务器，检查它们服务的网站是不是仍然可用。下面是我们的测试拓扑结构：

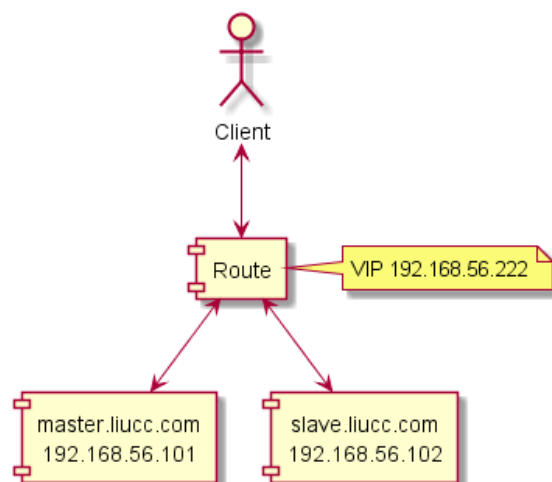


图 24.1 heartbeat 实验拓扑

映射服务所用的 IP 地址需要一直能够访问得到。通常，Heartbeat 会为你将指定的 IP 地址分配给主服务器上的虚拟网络接口卡。如果主服务器出现了故障，集群会自动将 IP 地址切换到另一台可用服务器上的虚拟网卡。如果主服务器恢复正常运行，它会再次将 IP 地址切换回到主服务器。由于具有迁移属性，这个 IP 地址被称为“浮动”地址。

### 24.1 安装 Heartbeat

在所有服务器上安装软件包

想组建集群，首先要使用 yum，在每一个节点上安装必要的软件包：

```
yum install PyXML cluster-glue cluster-glue-libs resource-agents
```

下一步，下载和安装官方 CentOS 软件库里面没有的两个 Heartbeat RPM 文件。

另外，你可以将 EPEL 软件库添加到源文件，并使用 yum 进行安装。

Heartbeat 会管理 Apache 的 httpd 服务的开启和停止，所以停止 Apache，并禁止它自动开启：

```
service httpd stop
chkconfig httpd off
```

设置主机名称

现在设置服务器的主机名称，为此编辑每个系统上的/etc/sysconfig/network，并更改 HOSTNAME 这一行：

HOSTNAME=xxxxx.liucc.com 新的主机名称会在服务器下一次启动时激活。你可以使用 hostname 命令立即激活它，不需要重启服务器：

hostname xxxxx.liucc.com 你可以在每一台服务器上运行 uname -n，以此证实主机名称已正确设置好。

## 24.2 配置 Heartbeat

想配置 Heartbeat，首先要将其默认配置文件从/usr 拷贝到/etc/ha.d/：

```
cp /usr/share/doc/heartbeat-3.0.4/authkeys /etc/ha.d/
cp /usr/share/doc/heartbeat-3.0.4/ha.cf /etc/ha.d/
cp /usr/share/doc/heartbeat-3.0.4/haresources /etc/ha.d/
```

然后，你还得改动全部集群节点上的所有三个文件，以便与你的需求相匹配。

authkeys 文件含有集群节点彼此联系时所使用的预共享密码。集群里面的每个 Heartbeat 消息都含有该密码，节点只处理拥有正确密码的那些消息。Heartbeat 支持 SHA1 密码和 MD5 密码。在 authkeys 文件中，下列指令将验证方法设置为 SHA1，并且定义了所使用的密码：

```
auth 2
2 sha1 pre-shared-password
```

保存该文件，然后使用命令

```
chmod 600 /etc/ha.d/authkeys
```

为该文件授予只有 root 用户可以读写的权限。

下一步，在 ha.cf 文件中，定义计时器、集群节点、消息传递机制、第 4 层端口及其他设置：

```
## 日志##
logfile /var/log/ha-log
logfacility local0hea
## 计时器##
## 所有计时器设成以秒为单位。如果你需要以毫秒为单位设置时间，就使用 ‘ms’ 。
```

```

## heartbeat 间隔时间##
keepalive 2

## 超过这个时间后，节点被认为已停滞##
deadtime 15
## 一些服务器花更长的时间来启动。该计时器定义了证实服务器宕机之前所等待的额外时间
## 该计时器的建议时间是停滞计时器的至少一倍。##
initdead 120
## 消息传递参数##
udpport 694
bcast eth0
## 你还可以使用多播或单播##
## 节点定义##
## 确保主机名称符合uname -n ##
node master.liucc.com
node slave.liucc.com

```

最后，文件 `haresources` 含有 **Heartbeat** 认为是主节点的那台服务器的主机名称，另外还含有浮动 IP 地址。该文件在所有服务器上都是一模一样，这点很重要。只要主节点在正常运行，它就服务所有请求；**Heartbeat** 停止其他所有节点上的高可用性服务。**Heartbeat** 检测到该主节点停机运行后，它会在集群中的下一个可用节点上自动开启服务。主节点恢复正常运行后，**Heartbeat** 会让它再次接手任务，服务所有请求。最后，该文件含有负责高可用性服务的脚本的名称：这里是 `httpd`。其他可能出现的值有 `squid`、`smb`、`nmb` 或 `postfix`，映射到通常位于 `/etc/init.d/` 目录中的服务启动脚本的名称。

在 `haresources` 中，定义 `master.liucc.com` 为主服务器，定义 `192.168.56.222` 为浮动 IP 地址，定义 `httpd` 为高可用性服务。你不需要创建任何接口，也不需要为任何接口手动分配浮动 IP 地址，**Heartbeat** 为我们处理这项任务：

```
master.liucc.com 192.168.56.222 httpd
```

每一台服务器上的配置文件准备就绪后，开启 **Heartbeat** 服务，并将它添加到系统启动项：

```
service heartebeat start
chkconfig heartbeat on
```

这时，可以查看一下 `master` 节点上的 IP 地址信息，

```

[root@master ~]# ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:4D:65:37
          inet addr:192.168.56.101  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe4d:6537/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

```

```

RX packets:13825 errors:0 dropped:0 overruns:0 frame:0
TX packets:10376 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:3095677 (2.9 MiB)  TX bytes:2253820 (2.1 MiB)

eth0:0    Link encap:Ethernet  HWaddr 08:00:27:4D:65:37
          inet addr:192.168.56.222  Bcast:192.168.56.255  Mask:255.255.255.
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

eth1      Link encap:Ethernet  HWaddr 08:00:27:BC:A6:E2
          inet addr:192.168.56.201  Bcast:192.168.56.255  Mask:255.255.255.
          inet6 addr: fe80::a00:27ff:febc:a6e2/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:21609 errors:0 dropped:0 overruns:0 frame:0
          TX packets:61 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:4926438 (4.6 MiB)  TX bytes:6922 (6.7 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:2696 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2696 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:5452373 (5.1 MiB)  TX bytes:5452373 (5.1 MiB)

```

你可以借助命令 `tailf /var/log/ha-log`，密切关注 Heartbeat 日志。

Heartbeat 可用于多项服务。比如说，`haresources` 中的下列指令将让 Heartbeat 同时管理 Apache 服务和 Samba 服务：

`master.liucc.com 192.168.56.222 httpd smb nmb` 不过，除非你还在运行 Pacemaker 之类的集群资源管理器（CRM），否则我不建议使用 Heartbeat 在单一集群中提供多项服务。要是没有 Pacemaker，Heartbeat 使用 IP 地址监测第 3 层中的集群节点。只要 IP 地址可以访问得到，Heartbeat 无视服务在服务器节点上可能遇到的任何崩溃或困难。

## 24.3 测试

一旦 Heartbeat 设置并运行起来，不妨对它测试一下。在所有三台服务器上创建单独的 `index.html` 文件，那样你就能看清哪台服务器在服务页面。浏览到 `192.168.56.222`，如果你设置好了 DNS，也可以浏览到相应域名。页面应该会从 `master.liucc.com.com` 加载，你可以查看服务器 1 中的 Apache 日志文件来核实这一点。试着刷新页面，证实该页面是否每次都从同一台服务器加载。

关闭主节点后，主节点产生的日志信息，

```
Apr 14 11:43:48 master.liucc.com heartbeat: [13928]: info: Heartbeat shutdown in
Apr 14 11:43:48 master.liucc.com heartbeat: [14300]: info: Giving up all HA reso
ResourceManager[14313]: 2015/04/14_11:43:48 info: Releasing resource group: mast
ResourceManager[14313]: 2015/04/14_11:43:48 info: Running /etc/init.d/httpd sto
ResourceManager[14313]: 2015/04/14_11:43:48 info: Running /etc/ha.d/resource.d/I
IPAddr[14386]: 2015/04/14_11:43:48 INFO: ifconfig eth0:0 down
IPAddr[14372]: 2015/04/14_11:43:48 INFO: Success
Apr 14 11:43:48 master.liucc.com heartbeat: [14300]: info: All HA resources reli
Apr 14 11:43:49 master.liucc.com heartbeat: [13928]: WARN: 1 lost packet(s) for
Apr 14 11:43:49 master.liucc.com heartbeat: [13928]: info: No pkts missing from
Apr 14 11:43:50 master.liucc.com heartbeat: [13928]: info: killing HBFIFO proces
Apr 14 11:43:50 master.liucc.com heartbeat: [13928]: info: killing HBWRITE proce
Apr 14 11:43:50 master.liucc.com heartbeat: [13928]: info: killing HBREAD proces
Apr 14 11:43:50 master.liucc.com heartbeat: [13928]: info: Core process 13932 ex
Apr 14 11:43:50 master.liucc.com heartbeat: [13928]: info: Core process 13931 ex
Apr 14 11:43:50 master.liucc.com heartbeat: [13928]: info: Core process 13930 ex
Apr 14 11:43:50 master.liucc.com heartbeat: [13928]: info: master.liucc.com Hear
```

此时，备节点上的日志信息为：

```
ResourceManager[13103]: 2015/04/14_11:43:49 info: Acquiring resource group: mast
IPAddr[13131]: 2015/04/14_11:43:49 INFO: Resource is stopped
ResourceManager[13103]: 2015/04/14_11:43:49 info: Running /etc/ha.d/resource.d/I
IPAddr[13194]: 2015/04/14_11:43:49 INFO: Using calculated nic for 192.168.56.222
IPAddr[13194]: 2015/04/14_11:43:49 INFO: Using calculated netmask for 192.168.56
IPAddr[13194]: 2015/04/14_11:43:49 INFO: eval ifconfig eth0:0 192.168.56.222 net
IPAddr[13180]: 2015/04/14_11:43:49 INFO: Success
ResourceManager[13103]: 2015/04/14_11:43:49 info: Running /etc/init.d/httpd sta
mach_down[13076]: 2015/04/14_11:43:49 info: /usr/share/heartbeat/mach_down: nice
mach_down[13076]: 2015/04/14_11:43:49 info: mach_down takeover complete for node
Apr 14 11:43:49 slave.liucc.com heartbeat: [12969]: info: mach_down takeover com
Apr 14 11:44:05 slave.liucc.com heartbeat: [12969]: WARN: node master.liucc.com:
Apr 14 11:44:05 slave.liucc.com heartbeat: [12969]: info: Dead node master.liucc
Apr 14 11:44:05 slave.liucc.com heartbeat: [12969]: info: Link master.liucc.com:
```

如果这一切进展良好，测试一下故障切换机制：停止 **master.liucc.com** 上的 **Heartbeat** 服务。浮动 IP 地址应该会迁移到服务器 **slave** 上，页面应该会从该服务器加载。迅速看一下 **slave** 的 **Apache** 日志，应该可以证实这一点。如果我们重启了服务器 **master** 上的 **heartbeat** 服务，浮动 IP 地址应该会按照 **haresources** 中的设置，从活动节点迁移到服务器 **slave** 上。

当我们把 **master** 上的 **heartbeat** 重新运行起来，这时，观察 **slave** 节点的日志信息，

```
ResourceManager[13654]: 2015/04/14_13:52:20 info: Releasing resource group:
ResourceManager[13654]: 2015/04/14_13:52:20 info: Running /etc/init.d/httpd
ResourceManager[13654]: 2015/04/14_13:52:21 info: Running /etc/ha.d/resource
IPAddr[13727]: 2015/04/14_13:52:21 INFO: ifconfig eth0:0 down
IPAddr[13713]: 2015/04/14_13:52:21 INFO: Success
Apr 14 13:52:21 slave.liucc.com heartbeat: [13641]: info: foreign HA resour
Apr 14 13:52:21 slave.liucc.com heartbeat: [12969]: info: Local standby pro
Apr 14 13:52:21 slave.liucc.com heartbeat: [12969]: WARN: 1 lost packet(s)
Apr 14 13:52:21 slave.liucc.com heartbeat: [12969]: info: remote resource t
Apr 14 13:52:21 slave.liucc.com heartbeat: [12969]: info: No pkts missing f
Apr 14 13:52:21 slave.liucc.com heartbeat: [12969]: info: Other node comple
```

正如我们所见，使用 **Heartbeat**，在 **RHEL** 下组建一个高可用性的 **Apache** 集群是件很容易的事。虽然我们使用了 2 台服务器，但 **Heartbeat** 在节点数量更多或更少的环境下应该同样没问题。**Heartbeat** 对节点数量没有任何限制，所以你可以根据需要扩展所设置环境的规模。

## 第二十五章 Nginx 负载均衡与高可用集群

Nginx 的负载均衡功能依赖于ngx\_upstream\_module模块，所支持的代理方式有，

1. proxy\_pass
2. fastcgi\_pass
3. memcached\_pass

### 25.1 upstream 模块介绍

### 25.2 Nginx 负载均衡方法介绍

### 25.3 准备工作

### 25.4 Nginx+KeepAlived 高可用

Nginx 负载均衡器总会存在单点故障的问题，所以，有必要对 Nginx 负载均衡器进行高可用配置。





## 第二十六章 HAproxy 负载均衡与高可用

### 26.1 关于 HAproxy

HAproxy 提供高可用性、负载均衡以及基于 TCP 和 HTTP 应用的代理，支持虚拟主机，它是免费、快速并且可靠的一种解决方案。HAproxy 特别适用于那些负载特大的 web 站点，这些站点通常又需要会话保持或七层处理。HAProxy 运行在当前的硬件上，完全支持数以万计的并发连接。并且它的运行模式使得它可以很简单安全的整合进您当前的架构中，同时可以保护你的 web 服务器不被暴露到网络上。

HAproxy 实现了一种事件驱动, 单一进程模型, 此模型支持非常大的并发连接数。多进程或多线程模型受内存限制、系统调度器限制以及无处不在的锁限制，很少能处理数千并发连接。事件驱动模型因为在有更好的资源和时间管理的用户空间 (User-Space) 实现所有这些任务，所以没有这些问题。此模型的弊端是，在多核系统上，这些程序通常扩展性较差。这就是为什么他们必须进行优化以使每个 CPU 时间片 (Cycle) 做更多的工作。[2]

### 26.2 环境准备

#### 26.2.1 测试环境准备

#### 26.2.2 安装 EPEL 源

### 26.3 HAproxy 的高可用



## 第五部分

### 监控篇



本章主要介绍几种常见的监控解决方案。在介绍之前，作者也说一下自己从工作以来所用过的监控软件。



## **第二十七章 Zabbix**

### **27.1 Zabbix 简介**

### **27.2 Zabbix 监控方案介绍**





## 第二十八章 Prometheus

比较流行的监控工具。



## 第六部分

### 自动化运维篇



本章介绍几种自动化运维的解决方案。



## 第二十九章 PXE 批量系统安装

### 29.1 PXE 简介

PXE (preboot execute environment) 是由 Intel 公司开发的最新技术, 工作于 Client/Server 的网络模式, 支持待安装服务器 (Client) 通过网络从远端安装服务器 (Server) 下载映像, 并由此支持来自网络的操作系统的启动安装过程, 其启动安装过程中, Client 端要求安装服务器动态分配 IP 地址, 再用 TFTP (Trivial File Transfer Protocol) 或 MTFTP (Multicast Trivial File Transfer Protocol) 协议下载一个启动软件包到本机内存中并执行, 由这个启动软件包完成 Client 端的基本软件设置, 从而引导安装预先安装在安装服务器上的 Client 端操作系统。

### 29.2 PXE 工作原理

#### 29.2.1 PXE 工作原理框图

#### 29.2.2 PXE 工作原理示意图

工作原理示意图说明:

1. Client 向 PXE Server 上的 DHCP 发送 IP 地址请求消息, DHCP 检测 Client 是否合法 (主要是检测 Client 的网卡 MAC 地址), 如果合法则返回 Client 的 IP 地址, 同时将启动文件 pxelinux.0 的位置信息一并传送给 Client;
2. Client 向 PXE Server 上的 TFTP 发送获取 pxelinux.0 请求消息, TFTP 接收到消息之后再向 Client 发送 pxelinux.0 大小信息, 试探 Client 是否满意, 当 TFTP 收到 Client 发回的同意大小信息之后, 正式向 Client 发送 pxelinux.0;
3. Client 执行接收到的 pxelinux.0 文件;
4. Client 向 TFTP 发送针对本机的配置信息 (记录在 TFTP 的 pxelinux.cfg 目录下), TFTP 将配置文件发回 Client, 继而 Client 根据配置文件执行后续操作;
5. Client 向 TFTP 发送 Linux 内核请求信息, TFTP 接收到消息之后将内核文件发送给 Client;
6. Client 向 TFTP 发送根文件请求信息, TFTP 接收到消息之后返回 Linux 根文件系统;
7. Client 启动 Linux 内核 (启动参数已经在 4 中的配置文件中设置好了);
8. Client 通过 NFS 下载镜像文件, 读取 autoyast 自动化安装脚本。至此, Client 正式进入自动化安装模式开始安装系统直到完成。

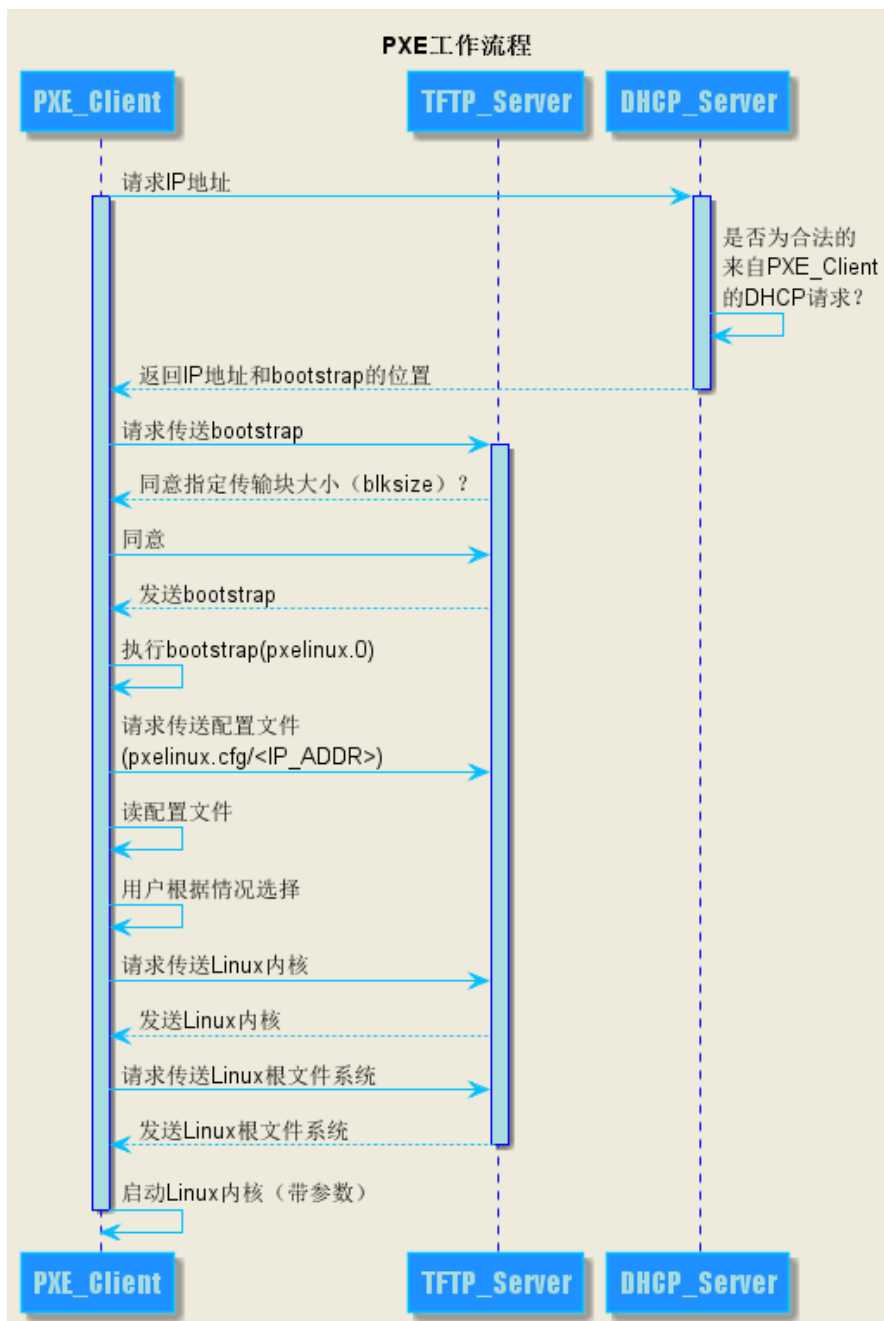


图 29.1 PXE 工作原理示意图



## 29.3 开始搭建 PXE 服务器

PXE 服务器包括 DHCP 服务、TFTP 服务与 NFS 服务。PXE 服务器安装完成后，配置 eth0 的 IP 地址为 192.168.241.1。其配置文件为：

```
# cat /etc/sysconfig/network/ifcfg-eth0
BOOTPROTO='static'
IPADDR='192.168.241.1/24'
STARTMODE='auto'
USERCONTROL='no'
```

### 29.3.1 安装及配置 DHCP 服务

安装 DHCP 服务端软件包，

```
# zypper install -y dhcp-server
```

配置 DHCP 服务端，修改/etc/dhcpd.conf 配置文件，

```
# vi /etc/dhcpd.conf
max-lease-time 7200;
ddns-updates off;
ddns-update-style none;
default-lease-time 600;
subnet 192.168.241.0 netmask 255.255.255.0 {
    option routers 192.168.241.254;
    next-server 192.168.241.1;
    filename "pxelinux.0";
    range 192.168.241.11 192.168.241.55;
    default-lease-time 14400;
    max-lease-time 172800;
}
host test1 {
    hardware ethernet 30:30:30:30:30:30;
    fixed-address 192.168.241.11;
}
```

修改/etc/sysconfig/dhcpd 配置文件，修改 DHCPD\_INTERFACE 的配置项，根据实际情况修改，

```
DHCPD_INTERFACE="eth0"
```

之后，启动 DHCP 服务，

```
# service dhcpd start
# rcdhcpd status
```

## 29.4 安装及配置 TFTP 服务

安装 TFTP 软件包,

```
# zypper install -y tftp
```

修改配置文件, 内容如下,

```
# cat /etc/xinetd.d/tftp
# default: off
# description: tftp service is provided primarily for booting or \
#   when a router need an upgrade. Most sites run this only on \
#   machines acting as "boot servers".
service tftp
{
    socket_type          = dgram
    protocol             = udp
    wait                 = yes
    flags                 = IPv6 IPv4
    user                 = root
    server                = /usr/sbin/in.tftpd
    server_args           = -s /opt/pxe/tftpboot
    disable              = no
}
```

启动 TFTP 服务,

```
# service xinetd restart
```

## 29.5 安装及配置 NFS 服务

安装 NFS 服务端软件包,

```
# zypper install -y nfs-kernel-server
```

NFS 服务配置,

```
# vi /etc/exports
/opt/pxe/iso *(ro,async)
/opt/pxe/autoinst *(ro,async)
```

启动 NFS 服务,

```
# rcnfsserver start
```

目录	说明
/opt/pxe/iso	镜像文件 SLES-11-SP2-DVD-x86_64-GM-DVD1.iso 放置目录
/opt/pxe/tftpboot	tftp 根目录
/opt/pxe/autoinst	自动化安装脚本 autoyast 放置目录

表 29-1 PXE 目录结构

## 29.6 SLEL11sp2 系统镜像文件

### 29.6.1 目录结构

目录结构为表29-1所示:

### 29.6.2 镜像挂载

```
# mkdir -p /opt/pxe/iso
# mount -o loop SLES-11-SP2-DVD-x86_64-GM-DVD1.iso /opt/pxe/iso
```

### 29.6.3 复制系统内核文件

```
# mkdir -p /opt/pxe/tftpboot/pxelinux.cfg
# cp /usr/share/syslinux/pxelinux.0 /opt/pxe/tftpboot
# cp /usr/share/syslinux/menu.c32 /opt/pxe/tftpboot
# cp /opt/pxe/iso/boot/x86_64/loader/linux /opt/pxe/tftpboot
# cp /opt/pxe/iso/boot/x86_64/loader/initrd /opt/pxe/tftpboot
```

### 29.6.4 创建 default 文件

```
PXEServer:~# cd /opt/pxe/tftpboot/pxelinux.cfg
PXEServer:~# vi default
default menu.c32
prompt 1
timeout 100
label local
localboot 0
label linux
kernel linux
append initrd=initrd install=nfs://192.168.241.1/opt/pxe/iso \
autoyast=nfs://192.168.241.1/opt/pxe/autoinst/autoinst.xml \
splash=silent showopt
```

### 29.6.5 AutoYast 自动化安装脚本

批量安装系统之前，首先手动安装一台 PXE 服务器，系统安装完成之后，在/root/目录下生成一个 autoinst.xml 文件，即为自动化安装脚本。

```
# mkdir -p /opt/pxe/autoinst
# cp /root/autoinst.xml /opt/pxe/autoinst
```

注意：

1. 若 autoinst.xml 自动化安装脚本中存在下列代码，须将下列代码删除；
2. 下列代码会把错误的网卡信息写入到/etc/udev/rules.d/70-persistent-net.rules 文件，在系统安装完成后自动获取 IP 地址过程中，导致网卡识别错误，系统获取不到 IP 地址；
3. 删除下列代码后，系统才能自动获取 IP 地址。

```
<net-udev config:type="list">
  <rule>
    <name>eth0</name>
    <rule>ATTR{address}</rule>
  </rule>
</net-udev>
```

保证 PXE 服务器和待安装服务器（Client）处于同一网络，在 Client 启动过程中根据提示手动按 F12 使后者以 PXE 模式启动，即可完成无人值守的自动化安装。

## 第三十章 Cobbler 自动化装机

### 30.1 Cobbler 介绍

号称补鞋匠都可以使用，大大降低了使用者的门槛，你懂的。

### 30.2 Cobbler 安装及配置

```
# yum install -y httpd dhcp tftp cobbler cobbler-web

# service httpd start
# service cobblerd start
# cobbler check # 检查是否通过，根据不通过的提示，进行修改配置文件
# vim /etc/cobbler/settings
    server: xxx.yyy.zzz.vvv
    next_server: xxx.yyy.zzz.vvv
# /etc/init.d/xinetd restart
# yum install -y pykickstart
# cobbler get-loaders
# openssl passwd -1 -salt 'lavenliu' 'laven'
fjsakjfsdkjfkdseri
# service cobblerd restart
# cobblerd check
# 如果检查没有问题，可以执行：
# cobbler sync
# vim /etc/cobbler/dhcp.template
#+END_EXAMPLE
```

配置完毕，接下来设置镜像：

```
BEGIN_EXAMPLE
mount /dev/cdrom /mnt
cobbler import --path=/mnt --name=CentOS-6.5-x86_64 --arch=x86_64
cobbler profile report # 可以找到默认ks配置文件
准备自定义的ks文件
cd /var/lib/cobbler/kickstarts
```

```
cobbler profile edit --name=CentOS-6.5-x86_64 --kickstart=/var/lib/cobbler/  
cobbler profile report  
每次改动，要执行sync动作  
cobbler sync
```

镜像文件被复制到了/var/www/cobbler/ks\_mirror/CentOS-6.5-x86\_64目录下。

## 第三十一章 Omnitty

Omnitty 是一个轻量级的批量运维管理工具，它是由德国人开发的。该工具目前最多支持同时操作 256 台远端机器，默认使用 SSH 协议进行通信<sup>1</sup>。它的安装很简单，下面给出安装步骤：

```
# cd /usr/local/src
# tar -zxf rote-0.2.8.tar.gz
# cd rote-0.2.8
# ./configure
# make
# make install
# cd ..

# cd /usr/local/src
# tar -zxf omnitty-0.3.0.tar.gz
# cd omnitty-0.3.0
# ./configure
# make
# make install

# omnitty
omnitty: error while loading shared libraries: librote.so.0: cannot open shared

# cd /usr/local/src/rote-0.2.8
# cp librote.so.0.2.8 /usr/lib64/
# cd /usr/lib64
# ln -s librote.so.0.2.8 librote.so.0
```

### 31.1 Omnitty 的简单使用

#### 31.1.1 添加机器

有两种方法可以向 Omnitty 里添加机器，一种是一台一台的进行手工添加；另一种就是使用文件的方式一次性的添加。如果机器数量较少，我们完全可以通过手工的方式

---

<sup>1</sup>只有 SSH 可供使用，使用 Omnitty 机器上的 SSH 客户端工具。

进行一台一台的添加，这样没有任何什么问题。如果要操作的机器数量大<sup>2</sup>，我们就可以通过使用文件的方式进行添加。我们把要操作的机器的 IP 或主机名<sup>3</sup>写入到一个文件中，然后 Omnitty 通过打开这个文件以达到加载机器的目的。

### 31.1.2 激活机器

在上面的小节中，我们把机器已经加载进来了，这时还不能使用。因为它们未被激活，显示的颜色是灰色的。如果要激活当前光标处的机器，可以按 F4 键即可激活该机器。只有激活的机器才响应我们的键盘操作。

### 31.1.3 开始操作

---

<sup>2</sup>大于 10 台

<sup>3</sup>如果写主机名，要确保 Omnitty 主机可以解析。



## 第三十二章 fabric



## 第三十三章 Ansible

### 33.1 关于 Ansible

Ansible 是一款自动化 IT 工具。它可以配置系统、部署软件、编排更复杂任务，诸如连续部署或零停机滚动更新。

Ansible 使用 SSH 协议进行通信。它是无客户端的工作方式，被管理端只需要存在 SSH 客户端即可。

Application	Application
Presentation	Presentation
Session	Session
Translate	Translate
Network	Network
Data Link	Data Link
Physical	Physical

图 33.1 网络七层模型

### 33.2 安装 Ansible

Ansible 的安装很简单，配置好 EPEL 源后，就可以直接使用 yum 来进行安装了。

```
yum install -y ansible
```

### 33.3 测试 Ansible

确定我们的 ansible 可以正常使用，接下来，我们准备几台测试机器。由于 ansible 是无 agent 的配置管理工具，在客户端无需安装任何额外程序，使用 SSH 协议进行通信，而 SSH 是默认安装的。

为了今后方便进行配置管理，我们需要使用 SSH 无密码进行通信。本测试的几台主机及主机名为：

在主控端生成 SSH 密钥对，并把公钥分发到被控端的机器上。

```
# ssh-keygen -t rsa
# ssh-copy-id -i /root/.ssh/id_rsa.pub root@192.168.56.111
# ssh-copy-id -i /root/.ssh/id_rsa.pub root@192.168.56.112
```

主机	说明
192.168.56.150	主控端
192.168.56.111	被控端
192.168.56.112	被控端

表 33-1 Ansible 测试环境

以上配置完毕，我们来简单的使用一下。比如，ping 测试一下我们的被控端是否存活。

```
# cat /etc/ansible/hosts
[my1]
192.168.56.111
192.168.56.112

# ansible my1 -m ping
192.168.56.111 | success >> {
    "changed": false,
    "ping": "pong"
}

192.168.56.112 | success >> {
    "changed": false,
    "ping": "pong"
}
```

命令说明：

my1 -- ansible操作的对象，my1组里有2台机器  
-m -- 表示使用的是ansible的哪个模块，默认是command模块

### 33.3.1 command 模块

使用 **command** 模块，可以执行一些比较简单的命令。**command** 是 **ansible** 的默认模块，我们要在一些机器上执行命令，可以省略“**command**”关键字。如果命令里包含空格，需要使用引号以转义掉空格字符。较复杂的命令 **command** 模块并不支持，诸如，重定向操作、管道操作等复杂操作 **command** 模块就无能为力了。下面看一些简单的例子，

```
# ansible my1 -m command -a "uptime"
192.168.56.111 | success | rc=0 >>
    15:50pm up    1:01,  2 users,  load average: 0.04, 0.03, 0.05

192.168.56.112 | success | rc=0 >>
    15:50pm up    1:01,  2 users,  load average: 0.00, 0.01, 0.05
```

命令说明：

```
-m command -- 表示使用 command 模块
-a           -- 表示模块的参数
```

### 33.3.2 shell 模块

shell 模块与 command 模块类似，可以执行一些较为复杂的命令。支持诸如重定向及管道的操作。下面看一个简单的例子，

```
# ansible my1 -m shell -a "cat /etc/hosts |grep -v '^#'"
192.168.56.111 | success | rc=0 >>
127.0.0.1 localhost
192.168.56.111 puppetcamq1.local.site
192.168.56.112 puppetcamq2.local.site
192.168.56.202 puppetmaster
192.168.56.203 puppetca

192.168.56.112 | success | rc=0 >>
127.0.0.1      localhost
192.168.56.111 puppetslave
```

### 33.3.3 package 模块

Ansible 有几种常见的包管理模块。比如 Debian 系列的系统上，使用的是 apt 包管理机制；RHEL 系列的系统上，使用的是 yum 包管理机制；SLES 系列的系统上，使用的是 zypper 包管理机制。那么，对应到 Ansible 中，对应的三个模块是 apt 模块、yum 模块、zypper 模块。三种模块的使用方式一样，这里就使用 SUSE 的 zypper 包管理方式。下面来看一个简单的例子，

```
# ansible my1 -m shell -a "service snmpd status"
192.168.56.111 | FAILED | rc=1 >>
service: no such service snmpd

192.168.56.112 | FAILED | rc=1 >>
service: no such service snmpd
```

以上说明，这两台机器并没有安装 net-snmp 软件包。接下来，我们要安装之。

```
# ansible my1 -m zypper -a "name=net-snmp state=present"
192.168.56.112 | success >> {
    "changed": true,
    "name": [
        "net-snmp"
    ],
    "state": "present"
```

```

}

192.168.56.111 | success >> {
    "changed": true,
    "name": [
        "net-snmp"
    ],
    "state": "present"
}

```

以上，已经安装 `net-snmp` 软件包，但是还没有启动。验证一下？

```

# ansible my1 -m shell -a "service snmpd status"
192.168.56.111 | FAILED | rc=3 >>
Checking for service snmpd:..unused

192.168.56.112 | FAILED | rc=3 >>
Checking for service snmpd:..unused

```

说明：

在SUSE里，服务未启动，则是“unused”状态

如何启动刚刚的 `snmpd` 服务呢？接下来看看 `service` 模块。

### 33.3.4 service 模块

服务已经安装却迟迟不启动，这如何是好？使用 `service` 模块可以轻松搞定，

```

# ansible my1 -m service -a "name=snmpd state=running"
192.168.56.111 | success >> {
    "changed": true,
    "name": "snmpd",
    "state": "started"
}

192.168.56.112 | success >> {
    "changed": true,
    "name": "snmpd",
    "state": "started"
}

```

根据输出，看上去是很美好，但是启动成功了吗？不会是在忽悠我们吧？你觉得很有必要进行验证一把，我也是这么认为的。

```

# ansible my1 -m shell -a "service snmpd status"

```

```
192.168.56.111 | success | rc=0 >>
Checking for service snmpd:..running
```

```
192.168.56.112 | success | rc=0 >>
Checking for service snmpd:..running
```

看来，*Ansible* 并没有忽悠我们，确实把 *snmpd* 服务给启动了。但是这里的服务是要开机启动的，该怎么办呢？看看帮助信息有没有相关的设置呢？结果小白<sup>1</sup>找到了 *enabled* 选项，可以满足此要求，试试看？试之前，小白觉得还是有必要进行检查一下，*snmpd* 服务是不是已经开机自启了呢？我觉得不会，因为小白没有设置！

```
# ansible my1 -m shell -a "chkconfig -l |grep snmpd"
192.168.56.111 | success | rc=0 >>
snmpd          0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

```
192.168.56.112 | success | rc=0 >>
snmpd          0:off  1:off  2:off  3:off  4:off  5:off  6:off
```

*snmpd* 服务安装完毕，确实没有设置开机自启，看来小白很是不专业，这么简单的问题，却猜测不正确。接下来就查下帮助设置一下吧，可以使用 *enabled* 选项，

```
# ansible my1 -m service -a "name=snmpd enabled=yes"
# ansible my1 -m shell -a "chkconfig -l |grep snmpd"
192.168.56.111 | success | rc=0 >>
snmpd          0:off  1:off  2:on   3:on   4:off  5:on   6:off

192.168.56.112 | success | rc=0 >>
snmpd          0:off  1:off  2:on   3:on   4:off  5:on   6:off
```

### 33.3.5 file 模块

*file* 模块一般是对远程主机上的文件或目录进行操作的。可以创建软、硬链接，修改文件属组及权限等。下面看一个例子，把 */etc/hosts* 文件链接到 */tmp* 目录下，

```
# ansible my1 -m file -a "src=/etc/hosts dest=/tmp/hosts state=link"
192.168.56.111 | success >> {
    "changed": true,
    "dest": "/tmp/hosts",
    "mode": "0777",
    "owner": "root",
    "src": "/etc/hosts",
    "state": "link"
}
```

---

<sup>1</sup>小白就是 ldczz2008@163.com

```
192.168.56.112 | success >> {
    "changed": true,
    "dest": "/tmp/hosts",
    "mode": "0777",
    "owner": "root",
    "src": "/etc/hosts",
    "state": "link"
}
```

接下来，验证一下是不是已经创建软链接了，

```
# ansible my1 -m shell -a "egrep -v '^#|^$' /tmp/hosts"
192.168.56.111 | success | rc=0 >>
127.0.0.1      localhost
192.168.56.111 puppetcamq1.local.site
192.168.56.112 puppetcamq2.local.site
192.168.56.202 puppetmaster
192.168.56.203 puppetca

192.168.56.112 | success | rc=0 >>
127.0.0.1      localhost
192.168.56.111 puppetslave
```

有了以上输出，说明没有问题！file 模块就介绍这么多吧！如果我们需要向远程主机发送文件该怎么办呢？可以使用 copy 模块，接下来的一个小节，来感受一下 copy 模块。

### 33.3.6 copy 模块

copy 模块是用向远程主机推送文件或目录用的。在推送过程中，我们可以设置文件在远程主机上的一些属性，如所有者、所属组等。接下来看操作，首先在控制端准备 test.sh 文件，

```
# cat /root/test.sh
#!/bin/bash

echo -n "Today is: "
date +%F\ %H:%M:%S

echo -n "My name is: "
hostname
```

然后，把文件推送到远程主机的/tmp 目录下，并设置权限，



```
# ansible my1 -m copy -a "src=/root/test.sh \  
> dest=/tmp/test.sh owner=root group=root \  
> mode=0755"  
192.168.56.111 | success >> {  
    "changed": true,  
    "dest": "/tmp/test.sh"  
}  
  
192.168.56.112 | success >> {  
    "changed": true,  
    "dest": "/tmp/test.sh"  
}
```

根据输出，我们可以看出，在主控端的 `test.sh` 文件已经复制过去了，并且修改了相应的权限。小白不高兴再次去做验证了，接下来就直接执行这个脚本吧！

```
# ansible my1 -m shell -a "/tmp/test.sh"  
192.168.56.111 | success | rc=0 >>  
Today is: 2015-07-01 15:02:58  
My name is: puppetcamq1
```

```
192.168.56.112 | success | rc=0 >>  
Today is: 2015-07-01 15:03:01  
My name is: puppetslave
```

小白由于好奇，这个脚本之所以可以执行成功，是因为我们指定了其权限为 `755`，其文件的所有者、所属组及其他用户皆可执行该文件。接下来，设置其权限为 `644` 呢，是否可以执行呢？试试呗，试一下机器又不会爆炸，

```
# ansible my1 -m file -a "path=/tmp/test.sh mode=0644"  
192.168.56.111 | success >> {  
    "changed": true,  
    "mode": "0644",  
    "owner": "root",  
    "path": "/tmp/test.sh",  
    "state": "file"  
}  
  
192.168.56.112 | success >> {  
    "changed": true,  
    "mode": "0644",  
    "owner": "root",  
    "path": "/tmp/test.sh",
```

```
    "state": "file"
}
```

通过上述设置及其输出情况，我们可以看到，`test.sh` 文件的权限已经修改为了 644。执行一下吧，

```
# ansible my1 -m shell -a "/tmp/test.sh"
192.168.56.111 | FAILED | rc=126 >>
/bin/sh: /tmp/test.sh: Permission denied

192.168.56.112 | FAILED | rc=126 >>
/bin/sh: /tmp/test.sh: Permission denied
```

这样执行呢？

```
# ansible my1 -m shell -a "bash /tmp/test.sh"
192.168.56.111 | success | rc=0 >>
Today is: 2015-07-01 15:09:47
My name is: puppetcamq1

192.168.56.112 | success | rc=0 >>
Today is: 2015-07-01 15:09:50
My name is: puppetslave
```

这个例子很无聊，主要是小白想让大家熟悉一下 `file` 模块而已。`file` 模块是从本地主机向远程主机推送文件，可不可以从远程主机拉取文件呢？答案是可以的！可以使用 `fetch` 模块实现该需求。这里不再演示。

### 33.3.7 lineinfile 模块

### 33.3.8 setup 模块

`setup` 模块里包含了很多有用的变量。在执行 `playbooks` 时，这些变量可以被 `playbooks` 来调用，有了这些变量，可以编排复杂的任务。`setup` 模块默认是输出所有远程主机上的变量信息，我们可以通过 `filter` 来过滤我们想要的信息。举个例子看看，如只显示网卡信息，

```
# ansible my1 -m setup -a "filter=ansible_eth[0-2]"
192.168.56.112 | success >> {
    "ansible_facts": {
        "ansible_eth1": {
            "device": "eth1"
        },
        "ansible_eth2": {
            "device": "eth2"
```

```
    },
  }

192.168.56.111 | success >> {
  "ansible_facts": {
    "ansible_eth0": {
      "device": "eth0",
    },
    "ansible_eth1": {
      "device": "eth1"
    }
  }
}

# ansible my1 -m setup -a "filter=ansible_kernel"
192.168.56.112 | success >> {
  "ansible_facts": {
    "ansible_kernel": "3.0.13-0.27-default"
  },
  "changed": false
}

192.168.56.111 | success >> {
  "ansible_facts": {
    "ansible_kernel": "3.0.13-0.27-default"
  },
  "changed": false
}
```

### 33.3.9 获得模块的帮助信息

安装完毕 *ansible*，同时也安装了 *ansible* 的帮助文档。可以通过 *ansible-doc* 命令来查看，后面跟具体模块的名称。*ansible* 有哪些模块呢？可以使用“*ansible-doc -l*”命令来查看，由于输出内容较多，这里就不演示了。在本测试环境中，*ansible* 的模块有 259 个，已足够我们目前的使用了。

我们只介绍了如何进行安装服务及查询状态的操作，并没有介绍如何卸载服务、删除软硬链接、修改文件权限等操作，更多的内容留给大家自己动手去实践，完全可以通过帮助信息来自己完成。

## 33.4 Ansible 工作流程

### 33.5 playbooks

### 33.6 一个综合实例

由第一章的内容

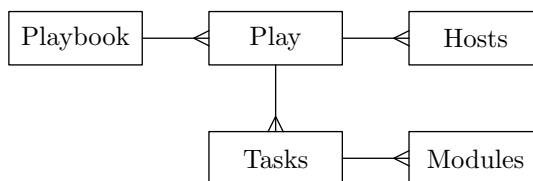


图 33.2 Playbook

该实例的目录结构为：

```
# tree playbooks/
playbooks/
|-- group_vars
|   |-- all
|-- hosts
|-- roles
|   |-- common
|       |-- handlers
|       |   |-- main.yml
|       |-- tasks
|       |   |-- main.yml
|       |-- templates
|       |   |-- net-snmp.j2
|       |   |-- ntp.conf.j2
|       |   |-- snmpd.conf.j2
|       |   |-- zypper.repo.j2
|       |-- vars
|       |-- main.yml
|-- site.yml

# cat playbooks/group_vars/all
---

# Variables listed here are applicable to all host groups
ntpserver: 172.16.25.35
```

该实例所执行的主机为，

```
# cat playbooks/hosts
```

```
[bigdata]
172.17.25.31
172.17.25.39
172.17.25.40

# cat playbooks/common/tasks/main.yml
---
- name: Prepare zypper repo file
  template: src=zypper.repo.j2 dest=/etc/zypper/repos.d/suse11sp2.repo

- name: Install net-snmp package
  zypper: pkg=net-snmp state=latest

- name: Modify snmp configuration file
  template: src=snmpd.conf.j2 dest=/etc/snmp/snmpd.conf

- name: Modify ntp configuration file
  template: src=ntp.conf.j2 dest=/etc/ntp.conf

- name: Restart ntp service
  shell: /etc/init.d/ntp restart

- name: Enable ntp service
  shell: chkconfig ntp on

- name: Restart snmpd service
  shell: service snmpd restart

- name: Enable snmpd service
  shell: chkconfig snmpd on

# cat playbooks/common/handlers/main.yml
---
- name: restart snmpd
  service name=snmpd state=restarted enabled=yes

- name: restart ntp
  service: name=ntp state=restarted enabled=yes

  snmp 日志轮询配置模板,

# cat playbooks/common/templates/net-snmp.j2
/var/log/net-snmpd.log {
```

```

daily
compress
dateext
maxage 10
logrotate 10
size=+1024k
notifempty
missingok
sharedscripts
postrotate
    /etc/init.d/snmpd reload ||:
if [ -x /etc/init.d/snmptrapd ] ; then \
    /etc/init.d/snmptrapd reload ||: ; \
fi
endscript
}

```

NTP 服务的配置模板,

```

# cat playbooks/common/templates/ntp.conf.j2
driftfile /var/lib/ntp/drift/ntp.drift
logfile /var/log/ntp
server {{ ntpserver }}

```

SNMP 服务的配置模板,

```

# cat playbooks/common/templates/snmpd.conf.j2
rocommunity public

```

YAST 源的配置模板,

```

# cat playbooks/common/templates/zypper.repo.j2
name=SUSE-Linux-Enterprise-Server-11-SP2 11.2.2-1.234
enabled=1
autorefresh=1
baseurl=ftp://{{ yastserver }}/
path=/
type=yast2
keeppackages=0

```

变量文件内容为,

```

# cat playbooks/common/vars/main.yml
---
ntpserver: 172.16.25.35
yastserver: 172.16.25.35

```

该实例的入口文件为，

```
# cat playbooks/site.yml
---
- name: apply common configuration to all nodes
  hosts: all
  roles:
    - common
```





## 第三十四章 SaltStack

This data coming from the minion (e.g., operating system) is called grains. But there is another type of data: pillar data. While grains are advertised by the minion back to the master, pillar data is stored on the master and is made available to each minion individually; that is, a minion cannot see any pillar data but its own. It is common for people new to Salt to ask about grains versus pillar data, so we will discuss them further in Chapter 5. For the moment, you can think of grains as metadata about the host (e.g., number of CPUs), while pillar is data the host needs (e.g., a database password). In other words, a minion tells the master what its grains are, while the minion asks the master for its pillar data. For now, just know that you can use either to define the target for a command.

### 34.1 初识 SaltStack

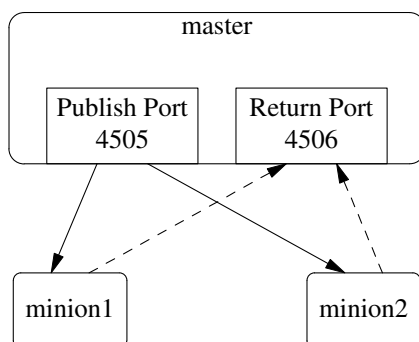


图 34.1 salt 通信示意图

### 34.2 安装及运行 SaltStack

#### 34.2.1 环境准备

以下测试是在 CentOS6U5 64 位系统上进行测试。系统均关闭 iptables 及 selinux，如果要开启 iptables，请设置允许开放本机的 53 端口。

假设已经做了 DNS 解析，如果没有做 DNS 解析，请在/etc/hosts 文件里设置静态主机名解析。这里使用 DNS 的方式进行主机名的解析工作，解析测试如下：

```
[root@master01 ~]# nslookup master01.lavenliu.com
Server: 192.168.20.134
```

表 34-1 SaltStack 测试环境机器列表

主机名	IP 地址	说明
master01.lavenliu.com	192.168.20.134	Salt Master
minion01.lavenliu.com	192.168.20.135	Salt Minion
minion02.lavenliu.com	192.168.20.136	Salt Minion

```
Address: 192.168.20.134#53
```

```
Name: master01.lavenliu.com
```

```
Address: 192.168.20.134
```

```
[root@master01 ~]# nslookup minion01.lavenliu.com
```

```
Server: 192.168.20.134
```

```
Address: 192.168.20.134#53
```

```
Name: minion01.lavenliu.com
```

```
Address: 192.168.20.135
```

```
[root@master01 ~]# nslookup minion02.lavenliu.com
```

```
Server: 192.168.20.134
```

```
Address: 192.168.20.134#53
```

```
Name: minion02.lavenliu.com
```

```
Address: 192.168.20.136
```

### 34.2.2 安装 SaltStack Master

使用 yum 进行 salt-master 的安装,

```
[root@master01 ~]# yum install -y salt-master
```

将 salt-master 加入开机自启动,

```
[root@master01 ~]# chkconfig salt-master on
```

### 34.2.3 安装 SaltStack Minion

使用 yum 进行 salt-minion 的安装, 在 minion01 及 minion02 两台机器上进行操作,

```
# yum install -y salt-minion
```

将 salt-minion 加入开机自启动, 在 minion01 及 minion02 两台机器上进行操作,

```
# chkconfig salt-minion on
```

## 34.3 SaltStack 配置

### 34.3.1 Master 端配置

### 34.3.2 Minion 端配置

## 34.4 基本使用

### 34.4.1 单 master 设置

### 34.4.2 基本命令

### 34.4.3 密钥管理

### 34.4.4 定位 minions

当我们有成千上万台机器时，这些机器承担着不同的角色。有的是 HTTP 服务器，有的是 DB 服务器，有的是文件共享服务器。而且它们分别运行不同的操作系统，比如，HTTP 服务器运行在 CentOS 系统之上；DB 服务器运行在 SUSE 系统之上；文件共享服务器运行在 FreeBSD 系统之上。当我们想在所有的服务器上增加 work 账号时，那将是很简单的一件事情。如果 WEB 服务器只安装 httpd，DB 服务器只安装 MySQL，文件共享服务器只安装 NFS 等，如何操作呢？Salt 提供了多种选项来帮助我们定位 minions。

#### Minion ID

最简单的方式就是通过指定具体的 minion ID 来定位目标机器。如，

```
# salt automatic01 test.ping
automatic01:
    True
```

#### List(-L) 选项

另外，我们可以使用 salt 命令的 -L 选项，向 salt 提供一组 Minion ID 的列表，如，

```
# salt -L mha-manager,automatic01 test.ping
mha-manager:
    True
automatic01:
    True
```

#### 通配符

我们还可以 shell-style 的通配符，通配符被替换成一组已签过名的 minions。表示所有已签名的 minions。如，

```
# salt '*' test.ping
windows01:
```

```
True
mha-manager:
    True
automatic01:
    True
debian:
    True
```

通过输出结果，可以看出，这些 `minions` 并不是按列表顺序返回的，而是按谁先返回数据的顺序返回的。

我们还可以使用通配符并结合 `minions` ID 一起使用，如，

```
# salt 'auto*' test.ping
automatic01:
    True
```

### Regular Expression(-E)

我们可以使用正则表达式来匹配更精确的 `minions`，如，

```
# salt -E '.*01' test.ping
windows01:
    True
automatic01:
    True
```

### Grains(-G)

`Grains` 是一组关于操作系统的信息，是在 `minions` 启动时加载的，是静态的数据。`Grains` 提供了操作系统名称和操作系统版本号等。我们可以基于此，只在 `CenOS6.5` 的机器上安装 `httpd` 软件包。

如何使用呢？如，

```
# salt -G 'os:CentOS' test.ping
automatic01:
    True
mha-manager:
    True
```

### Compound(-C)

接下来的这种方式更为强大一些。它允许我们以组合的形式来匹配 `minions`。接下来看一个简单的例子，如，

```
# salt -C 'auto* or G@os:Debian' test.ping
automatic01:
    True
debian:
    True

# salt -C 'auto* or L@debian, windows01' test.ping
windows01:
    True
automatic01:
    True
debian:
    True
```

## 34.5 理解 YAML

SLS 文件的默认渲染器是 YAML 渲染器。书写 SLS 文件只有简单的三条规则。

## 34.6 SaltStack 配置管理实例

接下来，就实际的运用以上的知识点，写出一些可工作的配置管理规则以进行配置管理。只要我们编排好 SLS 文件，Salt 就会按照我们的编排进行自动的配置管理。

看一下目录结构：

```
[root@master01 states]# pwd
/etc/salt/states
[root@master01 states]# tree
.
├── init
│   ├── files
│   │   ├── limits.conf
│   │   └── vimrc
│   ├── pkg.sls
│   ├── test.sls
│   └── vim.sls
├── prod
│   ├── elk
│   │   ├── elasticsearch.sls
│   │   ├── files
│   │   │   ├── elasticsearch-1.7.0.tar.gz
│   │   │   ├── elasticsearch_plugins.tar.gz
│   │   │   └── elasticsearch.yml
```



```
|   ├── pkg
|   │   └── pkg-init.sls
|   ├── redis
|   │   ├── files
|   │   │   └── redis.conf
|   │   └── server.sls
|   └── tomcat
|       ├── files
|       │   └── apache-tomcat-8.0.28.tar.gz
|       └── install.sls
└── top.sls
```

27 directories, 76 files

### 34.6.1 安装 JDK

```
[root@master01 states]# cat prod/jdk/install.sls
```

```
jdk-install:
```

```
  file.managed:
```

- name: /usr/local/src/jdk-8u65-linux-x64.tar.gz
- source: salt://prod/jdk/files/jdk-8u65-linux-x64.tar.gz
- user: root
- group: root
- mode: 644

```
  cmd.run:
```

- name: cd /usr/local/src && tar -xf jdk-8u65-linux-x64.tar.gz && mv jdk1.8.
- unless: test -d /usr/local/jdk

```
/etc/profile:
```

```
  file.append:
```

- text:
  - export JAVA\_HOME=/usr/local/jdk
  - export JRE\_HOME=\${JAVA\_HOME}/jre
  - CLASS\_PATH=\${JAVA\_HOME}/lib:\${JRE\_HOME}/lib
  - PATH=\$PATH:\$JAVA\_HOME/bin

**34.6.2 安装 Tomcat**

**34.6.3 安装 Nginx**

**34.6.4 安装 MySQL**

**34.6.5 安装 PHP**

**34.6.6 安装 Redis**

**34.6.7 安装 ELK Stack**

**34.6.8 安装 OpenStack**



## **第三十五章 Puppet**

### **35.1 关于 Puppet**

### **35.2 服务器端 puppetca 安装配置**

多台同构或异构的计算机用某种方式连接起来协同完成特定的任务就构成了集群系统，目前 Linux 下的集群主要有三种类型：

#### **35.2.1 配置 NTP**

#### **35.2.2 增加虚拟 IP**

#### **35.2.3 主机名 IP 静态解析配置和验证**

#### **35.2.4 安装新版本 puppet 软件包**

#### **35.2.5 修改配置并验证结果**

### **35.3 客户端 puppet 安装配置**

### **35.4 服务器端 rabbitmq-server 安装配置**

### **35.5 服务器端 mcollective 安装配置**

### **35.6 客户端 mcollective 安装配置**



## 第三十六章 Git

### 36.1 git 简介

Git 是一款分布式版本控制系统，有别于 CVS 和 SVN 等集中式版本控制系统，Git 可以让研发团队更加高效地协同工作，从而提高生产率。使用 Git，开发人员的工作不会因为频繁地遭遇提交冲突而中断，管理人员也无须为数据的备份而担心。

### 36.2 安装 git

```
# Debian系列
sudo apt-get install git
sudo apt-get install git-core
```

```
# RHEL系列
yum install git
```

### 36.3 创建版本库

设置用户名及邮件地址，

```
git config --global user.name "Laven Liu"
git config --global user.email "air.man.six@gmail.com"
git config --global color.ui true
```

查看已配置的信息，

```
git config --global user.name
Laven Liu

git config --global user.email
air.man.six@gmail.com

git config --list
git config --global --list
```

## 36.4 本地仓库

什么是版本库呢？版本库又名仓库，英文名 **repository**，我们可以简单理解成一个目录，这个目录里面的所有文件都可以被 **Git** 管理起来，每个文件的修改、删除，**Git** 都能跟踪，以便任何时刻都可以追踪历史，或者在将来某个时刻可以“还原”。

所以，创建一个版本库非常简单，首先，选择一个合适的地方，创建一个空目录：

```
$ mkdir learngit
$ cd learngit
$ pwd
/home/richard/learngit
```

**pwd** 命令用于显示当前目录。在我的 **Ubuntu** 上，这个仓库位于 **/home/richard/learngit**。

如果你使用 **Windows** 系统，为了避免遇到各种莫名其妙的问题，请确保目录名（包括父目录）不包含中文。

第二步，通过 **git init** 命令把这个目录变成 **Git** 可以管理的仓库：

```
$ git init
```

瞬间 **Git** 就把仓库建好了，而且告诉你是一个空的仓库（empty **Git repository**），细心的读者可以发现当前目录下多了一个 **.git** 的目录，这个目录是 **Git** 来跟踪管理版本库的，没事千万不要手动修改这个目录里面的文件，不然改乱了，就把 **Git** 仓库给破坏了。

首先这里再明确一下，所有的版本控制系统，其实只能跟踪文本文件的改动，比如 **TXT** 文件，网页，所有的程序代码等等，**Git** 也不例外。版本控制系统可以告诉你每次的改动，比如在第 5 行加了一个单词“**Linux**”，在第 8 行删了一个单词“**Windows**”。而图片、视频这些二进制文件，虽然也能由版本控制系统管理，但没法跟踪文件的变化，只能把二进制文件每次改动串起来，也就是只知道图片从 100KB 改成了 120KB，但到底改了啥，版本控制系统不知道，也没法知道。

不幸的是，**Microsoft** 的 **Word** 格式是二进制格式，因此，版本控制系统是没法跟踪 **Word** 文件的改动的，前面我们举的例子只是为了演示，如果要真正使用版本控制系统，就要以纯文本方式编写文件。

因为文本是有编码的，比如中文有常用的 **GBK** 编码，日文有 **Shift\_JIS** 编码，如果没有历史遗留问题，强烈建议使用标准的 **UTF-8** 编码，所有语言使用同一种编码，既没有冲突，又被所有平台所支持。

言归正传，现在我们编写一个 **readme.txt** 文件，内容如下：

```
$ cat readme.txt
Git is a version control system.
Git is free software.
```

一定要放到 **learngit** 目录下（子目录也行），因为这是一个 **Git** 仓库，放到其他地方 **Git** 再厉害也找不到这个文件。

和把大象放到冰箱需要 3 步相比，把一个文件放到 **Git** 仓库只需要两步。

第一步，用命令 **git add** 告诉 **Git**，把文件添加到仓库：

```
$ git add readme.txt
```

执行上面的命令，没有任何显示，这就对了，Unix 的哲学是“没有消息就是好消息”，说明添加成功。

第二步，用命令 `git commit` 告诉 Git，把文件提交到仓库：

```
$ git commit -m "wrote a readme file"
[master (root-commit) cb926e7] wrote a readme file
1 file changed, 2 insertions(+)
create mode 100644 readme.txt
```

简单解释一下 `git commit` 命令，`-m` 后面输入的是本次提交的说明，可以输入任意内容，当然最好是有意义的，这样你就能从历史记录里方便地找到改动记录。

嫌麻烦不想输入 `-m "xxx"` 行不行？确实有办法可以这么干，但是强烈不建议你这么干，因为输入说明对自己对别人阅读都很重要。实在不想输入说明的童鞋请自行 Google，我不告诉你这个参数。

`git commit` 命令执行成功后会告诉你，1 个文件被改动（我们新添加的 `readme.txt` 文件），插入了两行内容（`readme.txt` 有两行内容）。

为什么 Git 添加文件需要 `add`，`commit` 一共两步呢？因为 `commit` 可以一次提交很多文件，所以你可以多次 `add` 不同的文件，比如：

```
$ git add file1.txt
$ git add file2.txt
$ git add file3.txt
$ git commit -m "add 3 files."
```

**36.4.1 版本回退****36.4.2 工作区和暂存区****36.4.3 管理修改****36.4.4 撤销修改****36.4.5 删除文件****36.5 远程仓库****36.5.1 添加远程库****36.5.2 克隆远程库****36.6 分支管理****36.6.1 创建与合并分支****36.6.2 解决冲突****36.6.3 分支管理策略****36.6.4 Bug 分支****36.6.5 Feature 分支****36.6.6 多人协作****36.7 标签管理****36.7.1 创建标签****36.7.2 操作标签****36.8 使用 GitHub****36.9 自定义 Git****36.9.1 忽略特殊文件****36.9.2 配置别名****36.9.3 搭建 Git 服务器**

## 第七部分

### 虚拟化及云计算部分





虚拟化的主要目标是在一台主机上运行多个操作系统，以便充分利用大型机上昂贵的计算资源。随着 x86 处理器的性能提升以及应用的普及，虚拟化技术的发展也开始进入到 x86 架构领域。特别是到 20 世纪 90 年代末期，VMware 等虚拟化软件厂商为 x86 平台上虚拟化技术应用开辟了道路，提供了以 VMM（Virtual Machine Monitor）为中心，对 PC 服务器平台虚拟化的软件解决方案。纯软件解决方案在性能方面的瓶颈后来又进一步催生出了半虚拟化技术和硬件虚拟化技术，最终 Intel 的 VT（Virtualization Technology，虚拟化技术）和 AMD 的 SVM（Secure Virtual Machine，安全虚拟机）技术均在硬件级提供了对虚拟化的支持。虚拟化技术被列为 2009 年度最值得关注的 IT 技术之首，足可见虚拟化技术在目前企业计算和应用领域的重要性。



## 第三十七章 KVM

KVM (Kernel-based Virtual Machin) 的简称) 是一个开源的系统虚拟化模块, 自 Linux 2.6.20 之后集成在 Linux 的各个主要发行版本中。它使用 Linux 自身的调度器进行管理, 所以相对于 Xen, 其核心源码很少。KVM 目前已成为学术界的主流 VMM 之一。

### 37.1 关于 KVM

KVM 是开源软件, 全称是 Kernel-based virtual machine (基于内核的虚拟机), 是 x86 架构且硬件支持虚拟化技术 (如 Intel VT-x 或 AMD-V) 的 Linux 全虚拟化解决方案。KVM 包含一个为处理器提供底层虚拟化, 可加载的核心模块 `kvm.ko` (`kvm-intel.ko` 或 `kvm-amd.ko`)。KVM 还需要一个经过修改的 QEMU 软件 `qemu-kvm`, 作为虚拟机上层控制和界面。

KVM 能在不改变 Linux 或 Windows 镜像的情况下同时运行多个虚拟机 (多个虚拟机使用同一个镜像), 并为每一个虚拟机配置个性化硬件环境 (网卡、磁盘、图形适配器等)。

在主流的 Linux 内核, 如 2.6.20 以上的内核版本均已包含了 KVM 核心。

#### 37.1.1 KVM 管理工具 libvirt 简介

libvirt 是目前使用最为广泛的 KVM 虚拟机管理工具和应用程序接口 (API), 而且一些常用虚拟机的管理工具 (如 `virsh`、`virt-install` 及 `virt-manager` 等) 和云计算框架平台 (如 OpenStack、OpenNubla 等) 都在底层使用 libvirt 的应用程序接口。

libvirt 是为了更方便地管理平台虚拟化技术而设计的开放源代码的应用程序接口、守护进程和管理工具, 它不仅提供了对虚拟化客户机的管理, 也提供了对虚拟化网络和存储的管理。

libvirt 的应用程序接口已被广泛地用在基于虚拟化和云计算的解决方案中, 主要作为连接底层 Hypervisor 和上层应用程序的一个中间适配层。libvirt 对多种不同的 Hypervisor 的支持是通过一种基于驱动程序的架构来实现的。libvirt 对不同的 Hypervisor 提供了不同的驱动: 对 Xen 有 Xen 的驱动, 对 QEMU/KVM 有 QEMU 驱动; 对 VMware 有 VMware 的驱动。libvirt 作为中间适配层, 让底层 Hypervisor 对上层用户空间的管理工具是可以做到完全透明的, 因为 libvirt 屏蔽了底层各种 Hypervisor 的细节, 为上层管理工具提供了一个统一的 API 接口。通过 libvirt, 一些用户空间的管理工具可以管理各种不同的 Hypervisor 和其上面运行的客户机, 它们之间的交互拓扑如下图所示:

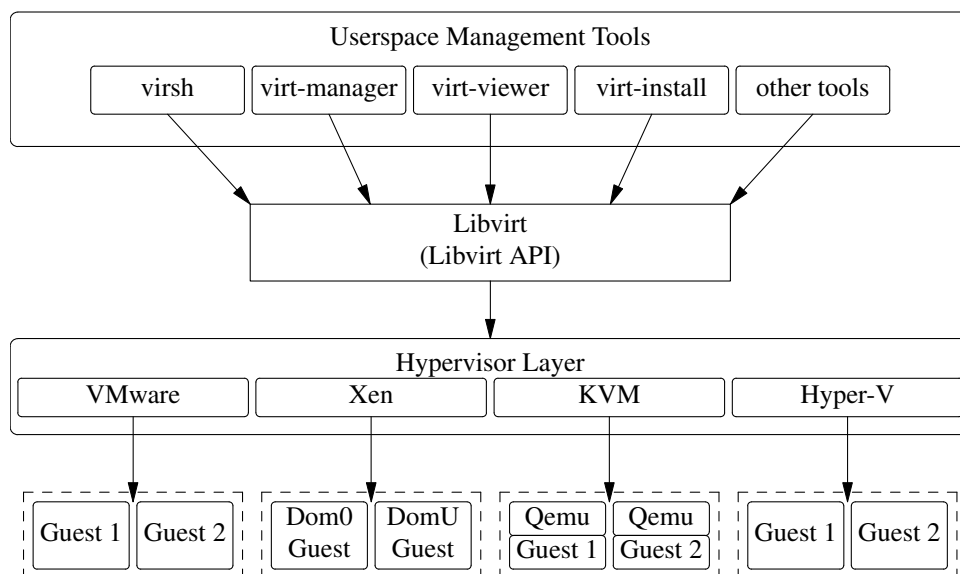


图 37.1 Libvirt 支持的虚拟化类型

### 37.1.2 libvirt 中的一些术语

1. 节点 (Node): 一台物理机; 上面可能运行多个虚拟客户机, Hypervisor 和 Domain 都运行在节点之上。

2. Hypervisor: 也称虚拟机监视器 (VMM), 如 KVM、Xen、VMware、Hyper-V 等, 是虚拟化中的一个底层软件层, 它可以虚拟化一个节点让其运行多个虚拟客户机 (不同的客户机有可能运行不同的操作系统)。

3. 域 (Domain): 是 Hypervisor 上运行的一个客户机操作系统实例。域也被称为实例、客户机操作系统 (Guest OS)、虚拟机 (Virtual Machine), 它们都是指同一个概念。

节点、Hypervisor 及域三者之间的关系如下图所示:

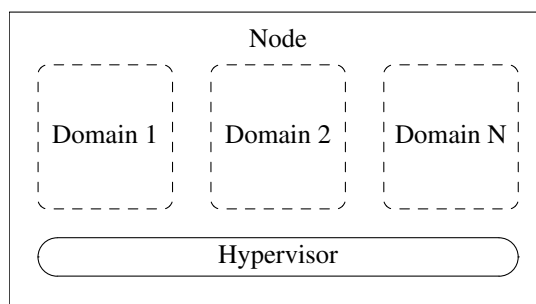


图 37.2 节点、域与 Hypervisor 之间的关系

### 37.1.3 检查宿主机是否支持 KVM 虚拟化

```
# grep -E --color '(vmx|svm)' /proc/cpuinfo
```

如果有红色的字符输出, 说明系统支持虚拟化, 然后接下来的一些操作才是有意义的。

## 37.2 安装前的准备工作

### 37.2.1 KVM 测试环境

下面的操作是在 VMWare 的两台虚拟机上进行的，  
测试环境为：  
安装 EPEL 源

### 37.2.2 安装 EPEL 源

```
# rpm -ivh http://mirrors.ustc.edu.cn/fedora/epel//6/x86_64/epel-release-6-8.noa
```

### 37.2.3 安装 KVM 管理工具

为了使用 KVM 虚拟化，操作系统至少要安装 `qemu-kvm` 及 `qemu-img` 两个软件包，这两个软件包在宿主机上为提使用者提供了用户层的 KVM 模拟器和硬盘镜像管理工具。因此，这两个软件包是必须安装的，操作如下：

```
# yum install -y qemu-kvm qemu-img
```

另外有几个建议安装的软件包，如下：  
安装以上建议的软件包：

```
# yum install virt-manager \
libvirt \
libvirt-python \
python-virtinst \
libvirt-client
```

## 37.3 开始部署 KVM 虚拟机

有了上面的准备工作，接下来就可以创建虚拟机了。

### 37.3.1 创建虚拟机镜像

虚拟机镜像文件就是 QEMU(KVM) 虚拟机使用的硬盘文件格式。

#### 创建 raw 格式镜像文件

这里新建一个 5G 的 RAW 格式的虚拟镜像文件，可以根据具体业务设置镜像文件大小，

```
# qemu-img create -f raw /opt/lavenliu.raw 5G
Formatting '/opt/lavenliu.raw', fmt=raw size=5368709120
```

创建完毕，可以查看刚才新建的虚拟机镜像文件，

```
# qemu-img info /opt/lavenliu.raw
image: /opt/lavenliu.raw
file format: raw
virtual size: 5.0G (5368709120 bytes)
disk size: 0
```

### 创建 qcow2 格式镜像文件

创建 qcow2 格式的镜像文件与创建 raw 格式的镜像文件步骤一样，只是 -f 选项后面指定的虚拟机镜像文件格式不一样而已。操作如下：

```
# qemu-img create -f qcow2 /opt/taoqi.qcow2 5G
Formatting '/opt/taoqi.qcow2', fmt=qcow2 size=5368709120 encryption=off clu
```

查看刚创建的虚拟机镜像文件，

```
# qemu-img info /opt/taoqi.qcow2
image: /opt/taoqi.qcow2
file format: qcow2
virtual size: 5.0G (5368709120 bytes)
disk size: 1.4G
cluster_size: 65536
```

format 指镜像的格式，常用的格式为 raw 和 qcow2，如果不声明，默认为 raw 模式，但推荐使用 qcow2 格式。

### 虚拟机镜像文件格式对比

1. raw 格式：可以简单、容易地导出到其它模拟器中，但是立即分配占用空间大。2. qcow2 格式：是 qcow 格式的升级版本，是目前最万能的格式。使用它可获得较小映像，也是虚拟池一直在使用的镜像格式，支持镜像快照，方便的恢复管理。

大量数据表明：qcow2 格式的文件虽然在性能上比 Raw 格式的有一些损失（主要体现在对于文件增量上，qcow2 格式的文件为了分配 cluster 多花费了一些时间），但是 qcow2 格式的镜像比 Raw 格式文件更小，只有在虚拟机实际占用了磁盘空间时，其文件才会增长，能方便的减少迁移花费的流量，更适用于云计算系统，同时，它还具有加密，压缩，以及快照等 raw 格式不具有的功能。

### 37.3.2 安装虚拟机

接下来就进行虚拟机的安装，分别演示 raw 格式与 qcow2 格式的虚拟机镜像文件系统的安装。两者的系统安装基本上是一样的，不同的只是 --disk 选项要指定不同的镜像文件格式而已。

### 安装 raw 格式的虚拟机系统

有了上面的虚拟机镜像文件，相当于我们已经有了一个虚拟的硬盘了，接下来就可以使用 CentOS 的 ISO 镜像文件，往这个新建的虚拟机镜像文件里安装 CentOS6u5 的系统了。作者已事先准备好了 CentOS6u5 64 位的 ISO 系统镜像文件，并存放在 `/opt/CentOS-6.5-x86_64-bin-DVD1.iso`。下面就演示如何进行安装，操作如下：

```
# virt-install --virt-type kvm --name lavenliu --ram 512 \
  --cdrom=/opt/CentOS-6.5-x86_64-bin-DVD1.iso \
  --disk path=/opt/lavenliu.raw \
  --network network=default \
  --graphics vnc,listen=0.0.0.0 \
  --noautoconsole --os-type=linux \
  --os-variant=rhel6
```

上述命令行参数的一些说明：

### 安装 qcow2 格式的虚拟机系统

```
# virt-install --virt-type kvm --name taoqi --ram 512 \
  --cdrom=/opt/CentOS-6.5-x86_64-bin-DVD1.iso \
  --disk path=/opt/taoqi.qcow2,format=qcow2 \
  --network network=default \
  --graphics vnc,listen=0.0.0.0 \
  --noautoconsole --os-type=linux \
  --os-variant=rhel6
```

上述命令行参数的一些说明：

安装过程中，`libvirtd` 守护进程会监听 5900 和 5901 的两个 VNC 端口（因为启动了两台虚拟机），供 VNC 客户端进行远程连接，可以查看 `kvm01` 服务器是否监听 5900 及 5901 端口<sup>1</sup>，

```
# netstat -natup |grep 59
tcp      0      0 0.0.0.0:5900      0.0.0.0:*        LISTEN   20249/qemu-kvm
tcp      0      0 0.0.0.0:5901      0.0.0.0:*        LISTEN   20268/qemu-kvm
```

## 37.4 KVM 虚拟机管理

### 37.4.1 libvirt 的配置和使用

`libvirtd` 是一个作为 `libvirt` 虚拟化管理系统中的服务器端的守护进程，如果要想某个节点能够用 `libvirt` 进行管理（无论是本地管理还是远程管理），都需要在这个节点上运行着 `libvirtd` 这个守护进程，以便让其他上层管理工具可以连接到该节点，`libvirtd` 负责

<sup>1</sup>如果同时安装 N 台虚拟机，则将分别监听 5900、5901、...、590N 端口。

执行其他管理工具发送到它的虚拟化管理操作指令。而 libvirt 的客户端工具（包括 virsh、virt-manager 等）可以连接到本地或远程的 libvirtd 进程，以便管理节点上的客户机（启动、停止、重启、迁移等）、收集节点上的宿主机和客户机的配置和资源使用状态。

默认情况下，libvirtd 监听在本地的 Unix Domain Socket 上，并没有监听基于网络的 TCP/IP socket，需要使用“-l”或“-listen”的命令行参数来开启对 libvirtd.conf 配置文件中对 TCP/IP socket 的配置。另外，libvirtd 进程的启动或停止，并不会直接影响正在运行的客户机。libvirtd 在启动或重启完成时，只要客户机的 XML 配置文件是存在的，libvirtd 会自动加载这些客户机的配置文件，获取它们的信息；当然，如果客户机没有基于 libvirt 格式的 XML 文件来运行，libvirtd 则不能发现它。

### 37.4.2 虚拟机拷贝

虚拟机的拷贝，其实可以理解为复制某一台虚拟机的 XML 文件为一个新的 XML 配置文件，然后重新 define 这个新的 XML 文件即可。不过为了能够让新的虚拟机正常工作，需要修改新的 XML 文件的几处配置，大致的步骤为：

dump 某台虚拟机的 XML 配置文件到新的 XML 配置文件

复制某台虚拟机的镜像文件到新的镜像文件

修改新的 XML 配置文件的相应配置

修改新的虚拟机的 MAC 地址

操作如下：

```
# virsh dumpxml lavenliu > lavenliu_new.xml
# cp /opt/lavenliu.raw /opt/lavenliu_new.raw
# sed -i 's#lavenliu#lavenliu_new#g' /opt/lavenliu_new.xml
# sed -i "s#<uuid>.*</uuid>#<uuid>\`uuidgen`</uuid>#g" /opt/lavenliu_new.xml
# sed -i "s@<mac address=.*@<mac address='`printf '00:0C:%02X:%02X:%02X:%02X`" /opt/lavenliu_new.xml"
```

注意，还要修改 MAC 地址，不然两台虚拟机的 MAC 地址是一样的。

使用这种复制虚拟机镜像文件的方式，需要修改修改虚拟机的 xml 配置文件。验证上述步骤是否成功，操作如下：

```
# virsh list --all
```

Id	Name	State
16	lavenliu	running
-	lavenliu_new	shut off # 已经创建成功，但未启动
-	taoqi	shut off

接下来启动 lavenliu\_new 虚拟机，是否可以启动，

```
# virsh list
```

Id	Name	State
16	lavenliu	running
17	lavenliu_new	running



### 37.4.3 虚拟机克隆

克隆虚拟机时，虚拟机必须要处于关闭或暂停状态。否则，将会出现下面的提示：

```
ERROR    Domain with devices to clone must be paused or shutdown.
```

克隆虚拟机操作如下：

```
# virt-clone -o lavenliu -n lavenliu_clone -f /opt/lavenliu_clone.raw
Cloning lavenliu.raw          | 5.0 GB      02:33
```

```
Clone 'lavenliu_clone' created successfully.
```

相应选项说明：

-o 指定源虚拟机  
-n 指定新虚拟机  
-f 指定新虚拟机的镜像文件

查看已克隆的虚拟机：

```
# ll /opt/
total 8217244
-rwxr-xr-x  1 root root 5368709120 Apr 26 18:11 lavenliu_clone.raw
-rw-r--r--  1 root root 5368709120 Apr 26 18:08 lavenliu.raw
drwxr-xr-x. 2 root root          4096 Nov 22  2013 rh
-rw-r--r--  1 root root 1517944832 Apr 26 17:16 taoqi.qcow2
```

能否启动呢？试试看：

```
# virsh list --all
 Id      Name                               State
-----
 18      lavenliu_new                       running
-        lavenliu                         shut off
-        lavenliu_clone                    shut off
-        taoqi                            shut off
```

接下来启动刚克隆的 `lavenliu_clone` 虚拟机，

```
# virsh start lavenliu_clone
Domain lavenliu_clone started
```

查看运行状态：

```
# virsh list --all
  Id      Name                               State
-----
  18      lavenliu_new                          running
  19      lavenliu_clone                       running # 启动成功
  -       lavenliu                             shut off
  -       taoqi                               shut off
```

#### 37.4.4 增加虚拟机硬盘空间

对硬盘做操作要谨慎。要做好备份再进行操作比较可靠。

```
# cd /opt
# qemu-img info lavenliu.raw
image: lavenliu.raw
file format: raw
virtual size: 5.0G (5368709120 bytes)
disk size: 1.4G
# qemu-img resize lavenliu.raw +1G
Image resized.
# qemu-img info lavenliu.raw
image: lavenliu.raw
file format: raw
virtual size: 6.0G (6442450944 bytes)
disk size: 1.4G
```

#### 37.4.5 虚拟机硬盘格式转换

把 raw 格式的硬盘转换为 qcow2 格式的，

```
# cd /opt
# qemu-img convert -c -f raw -O qcow2 lavenliu.raw new.qcow2
# 格式转换需要一段时间
# qemu-img check new.qcow2
No errors were found on the image.
Image end offset: 491454464
```

上述命令选项说明：

```
-c 表示压缩，对qcow2格式的镜像文件
-f 要被转换的格式
-O 转换后的格式
lavenliu.raw # 要被转换的镜像文件
new.qcow2 # 转换后的镜像文件
```

然后直接使用 `vim` 修改 `lavenliu.xml` 文件后, `reboot` 虚拟机后, 其磁盘格式仍然是 `raw` 格式。彻底更改使用“`virsh edit new`”。在生产环境中使用 `virsh edit xxx` 来修改设置。

### 37.4.6 虚拟机迁移

尽量避免使用动态迁移。可以使用复制的静态方式来迁移虚拟机, 然后重新定义虚拟机的 XML 文件。

### 37.4.7 创建虚拟机快照

要创建虚拟机快照, 首先要满足以下几个条件:  
虚拟机使用的镜像文件格式为 `qcow2` 格式;  
进行快照操作时, 虚拟机最好处于关闭或暂停的状态;  
关于快照相关的命令语法为:

```
# qemu-img snapshot [-l | -a snapshot | -c snapshot | -d snapshot] filename
```

命令行参数说明:

```
qemu-img snapshot
-l # 查看虚拟机快照
-a snapshot # 应用指定的快照
-c snapshot # 创建快照
-d snapshot # 删除指定的快照
```

#### 创建虚拟机快照

先来个错误的操作, 对 `raw` 格式的虚拟机镜像文件创建快照,

```
# qemu-img snapshot -c 1st_snapshot /opt/lavenliu.raw
Could not create snapshot '1st_snapshot': -95 (Operation not supported)
```

结果给出了错误提示: “Operation not supported”。

接下来对 `taoqi` 这台虚拟机先进行关闭操作, 然后再进行创建快照操作 (最好关闭或暂停虚拟机),

```
# virsh destroy taoqi
# qemu-img snapshot -c 1st_snapshot /opt/taoqi.qcow2
```

如果没有提示, 说明创建成功。在 UNIX 世界中, 没有消息就是好消息。接下来我们可以登录到 `taoqi` 这台虚拟机里, 创建一个测试文件, 如在 `/root` 目录下创建一个名为 `taoqi.iso` 的文件, 操作如下:

```
# dd if=/dev/zero of=/root/taoqi.iso bs=1M count=256
```

这时可以再次对 `taoqi` 这台虚拟机创建第 2 个快照, 操作如下:

```
# qemu-img snapshot -c 2nd_snapshot /opt/taoqi.qcow2
```

## 查看虚拟机快照

以上我们都创建了两个快照了,怎么查看呢?以及创建的快照文件存放在何处呢?接下来看操作:

```
# qemu-img snapshot -l /opt/taoqi.qcow2
Snapshot list:
ID          TAG          VM SIZE          DATE          VM CLOCK
1           1st_snapshot      0 2016-04-26 20:48:58    00:00:00.000
2           2nd_snapshot      0 2016-04-26 20:49:52    00:00:00.000
```

## 恢复指定快照

这时我们恢复到第一个快照,我们在做完第一个快照后,创建了一个/root/test.iso 的文件,那么当我们恢复到第一个快照时,在/root 目录下应该没有 test.iso 文件的,是不是这样的呢?接下来看操作:

```
# virsh destroy taoqi
# qemu-img snapshot -a 1st_snapshot /opt/taoqi.qcow2
# virsh start taoqi
```

登录到 taoqi 这台虚拟机进行查看,发现/root/test.iso 文件没有了(肯定没有改文件,因为在创建 1st\_snapshot 时,还没有创建/root/test.iso 文件的,所以恢复到 1st\_snaphost 快照时是看不到创建快照后的文件)。

恢复 2nd\_snapshot 快照,

```
# virsh destroy taoqi
# qemu-img snapshot -a 2nd_snapshot /opt/taoqi.qcow2
# virsh start taoqi
```

登录到 taoqi 这台虚拟机,查看/root 目录, test.iso 文件是不是又回来了。

## 删除虚拟机快照

删除快照很简单,把相应的选项换成-d就可以了,首先查看当前有哪些快照,操作如下:

```
# qemu-img snapshot -l /opt/taoqi.qcow2
Snapshot list:
ID          TAG          VM SIZE          DATE          VM CLOCK
1           1st_snapshot      0 2016-04-27 12:08:24    00:00:00.000
2           2nd_snapshot      0 2016-04-27 12:45:27    00:00:00.000
```

接下来删除第一个快照,

```
# qemu-img snapshot -d 1st_snapshot /opt/taoqi.qcow2
```

## 37.5 KVM 虚拟机桥接网络

默认情况下，虚拟机的网络使用的是 NAT 的方式。使用 `ifconfig` 命令查看宿主机的网络接口情况：

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:1C:1F:8E
          inet addr:192.168.19.134  Bcast:192.168.19.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe1c:1f8e/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:25557 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14455 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:25142063 (23.9 MiB)  TX bytes:834829 (815.2 KiB)

eth1      Link encap:Ethernet  HWaddr 00:0C:29:1C:1F:98
          inet addr:192.168.20.129  Bcast:192.168.20.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe1c:1f98/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:41832 errors:0 dropped:0 overruns:0 frame:0
          TX packets:39199 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3197973 (3.0 MiB)  TX bytes:13961170 (13.3 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:13 errors:0 dropped:0 overruns:0 frame:0
          TX packets:13 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2828 (2.7 KiB)  TX bytes:2828 (2.7 KiB)

virbr0    Link encap:Ethernet  HWaddr 52:54:00:F0:29:34
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:431 errors:0 dropped:0 overruns:0 frame:0
          TX packets:341 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:41451 (40.4 KiB)  TX bytes:42172 (41.1 KiB)

vnet0     Link encap:Ethernet  HWaddr FE:54:00:CC:94:60  # lavenliu这台虚拟机的MA
```

```

        inet6 addr: fe80::fc54:ff:fecc:9460/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:12 errors:0 dropped:0 overruns:0 frame:0
        TX packets:156 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:500
        RX bytes:1112 (1.0 KiB)  TX bytes:8548 (8.3 KiB)

vnet1    Link encap:Ethernet  HWaddr FE:54:00:18:DF:60  # taoqi这台虚拟机的
        inet6 addr: fe80::fc54:ff:fe18:df60/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:27 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1174 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:500
        RX bytes:2636 (2.5 KiB)  TX bytes:63308 (61.8 KiB)

```

查看系统的 ARP 缓存列表:

```

# arp
Address HWtype  HWaddress Flags Mask Iface
192.168.20.1 ether    00:50:56:c0:00:01 C   eth1
192.168.122.141 ether   52:54:00:cc:94:60 C   virbr0
192.168.19.2 ether    00:50:56:ea:9c:68 C   eth0
192.168.122.41 ether   52:54:00:18:df:60 C   virbr0

```

使用 `brctl` 命令查看:

```

# brctl show
bridge name bridge id STP enabled interfaces
virbr0 8000.525400f02934 yes virbr0-nic
vnet0
vnet1

```

当我们要在局域网中访问宿主机上的虚拟机时,这时使用桥接网络将显得很有必要。接下来在 `kvm02.lavenlilu.com` 主机上创建 `kvm-bridge` 虚拟机并使用桥接的方式,可以让局域网中的其他机器可以访问到它(如从 `kvm01.lavenliu.com` 主机可以访问 `kvm02` 上的 `kvm-bridge` 虚拟机)。操作如下,然后在宿主机上增加网桥设备 `br1`,

```

# virsh iface-bridge eth1 br1
Created bridge br1 with attached device eth1
Bridge interface br1 started

```

上面的命令执行完毕,将会在 `/etc/sysconfig/network-scripts/` 目录中产生 `ifcfg-br1` 网络接口文件,内容如下:

```
# cat /etc/sysconfig/network-scripts/ifcfg-br1
DEVICE="br1"
ONBOOT="yes"
TYPE="Bridge"
BOOTPROTO="none"
IPADDR="192.168.20.130"
NETMASK="255.255.255.0"
GATEWAY="192.168.20.1"
STP="on"
DELAY="0"
```

网卡接口文件 `ifcfg-eth1` 上的信息也会被 `virsh` 工具修改，IP 地址被删除了，内容如下：

```
# cat /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
ONBOOT=yes
BRIDGE="br1"
```

使用 `brctl show` 命令查看当前桥接情况：

```
# brctl show
bridge name bridge id STP enabled interfaces
br1 8000.000c2948b730 yes eth1
```

接下来创建新的 `kvm-bridge` 虚拟机镜像文件，并安装系统：

```
# qemu-img create -f qcow2 /opt/kvm-bridge.qcow2
# virt-install --virt-type kvm --name kvm-bridge \
--ram 512 \
--cdrom=/opt/CentOS-6.5-x86_64-bin-DVD1.iso \
--disk path=/opt/kvm-bridge.qcow2,format=qcow2 \
--network bridge=br1 \
--graphics vnc,listen=0.0.0.0 \
--noautoconsole \
--os-type=linux \
--os-variant=rhel6

Starting install...
Creating domain... | 0 B 00:01
Domain installation still in progress. You can reconnect to
the console to complete the installation process.
```

等待虚拟机操作系统安装完毕，可以查看当前宿主机的网络桥接情况，

```
# brctl show
bridge name bridge id                STP enabled interfaces
br1 8000.000c2948b730    yes                eth1
                        vnet0
```

可以在我们的 Windows 机器上进行 ping 测试，是否可以连通我们的 kvm-bridge 虚拟机（这里的虚拟机 IP 地址为 192.168.20.117），

```
ping 192.168.20.117
```

正在 Ping 192.168.20.117 具有 32 字节的数据：

来自 192.168.20.117 的回复：字节=32 时间<1ms TTL=64

来自 192.168.20.117 的回复：字节=32 时间<1ms TTL=64

来自 192.168.20.117 的回复：字节=32 时间<1ms TTL=64

来自 192.168.20.117 的回复：字节=32 时间<1ms TTL=64

192.168.20.117 的 Ping 统计信息：

数据包：已发送 = 4，已接收 = 4，丢失 = 0 (0% 丢失)，  
往返行程的估计时间(以毫秒为单位)：

最短 = 0ms，最长 = 0ms，平均 = 0ms

至此，虚拟机使用桥接网络可以被局域网中的其他机器访问。

## 37.6 Libvirt API

### 37.6.1 Libvirt API 简介

libvirt 的核心价值和主要目标就是提供了一套管理虚拟机的、稳定的、高效的应用程序接口（API）。libvirt API 本身是用 C 语言实现的，

### 37.6.2 C API 示例

在使用 libvirt API 之前，必须要在远程或本地节点上启动 libvirtd 守护进程。在使用 libvirt 的客户端，需要安装 libvirt-devel 开发包，如果没有安装，请执行如下操作进行安装：

```
# yum install -y libvirt-devel
```

编写源代码时，需要在源码的开头引入<libvirt/libvirt.h>头文件，编写完毕源代码，如何编译呢？

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



## 显示某个域的信息

代码如下:

```
# cat dominfo.c
/**
 * Get domain information via libvirt C API.
 * Tested with libvirt-devel- on CentOS6.5 host system
 */

#include <stdio.h>
#include <libvirt/libvirt.h>

int getDomainInfo(int id)
{
    virConnectPtr conn = NULL; /* the hypervisor connection */
    virDomainPtr dom = NULL; /* the domain being checked */
    virDomainInfo info; /* the information being fetched */

    /* NULL means connect to local QEMU/KVM hypervisor*/
    conn = virConnectOpenReadOnly(NULL);
    if (conn == NULL) {
        fprintf(stderr, "Failed to connect to hypervisor\n");
        return 1;
    }

    /* find the Domain by its ID*/
    dom = virDomainLookupByID(conn, id);
    if (dom == NULL) {
        fprintf(stderr, "Failed to find Domain %d\n", id);
        virConnectClose(conn);
        return 1;
    }

    /* Get virDomainInfo structure of the domain */
    if (virDomainGetInfo(dom, &info) < 0) {
        fprintf(stderr, "Failed to get information for Domain %d\n", id);
        virDomainFree(dom);
        virConnectClose(conn);
        return 1;
    }
}
```

```

/* Print some info of the domain*/
printf("Domain ID: %d\n", id);
printf("  vCPUs: %d\n", info.nrVirtCpu);
printf("  maxMem: %d KB\n", info.maxMem);
printf("  memory: %d KB\n", info.memory);

if (dom != NULL)
    virDomainFree(dom);
if (conn != NULL)
    virConnectClose(conn);

return 0;
}

int main(int argc, char *argv[])
{

    int dom_id = 5;
    printf("-- Get Domain info by ID via libvirt C API --\n");
    getDomainInfo(dom_id);
    return 0;
}

```

编译并运行:

```

# gcc dominfo.c -o dominfo -lvirt
./dominfo
-- Get Domain info by ID via libvirt C API --
Domain ID: 5
  vCPUs: 1
  maxMem: 524288 KB
  memory: 524288 KB

```

### 37.6.3 Python API 示例

许多种语言都提供了 libvirt 的绑定。Python 作为一种在 Linux 上比较流行的编程语言，它也提供了 libvirt API 的绑定。在使用 Python 调用 libvirt API 之前，需要安装 libvirt-python 软件包，如果没有安装则执行如下命令进行安装：

```
# yum install -y libvirt-python
```

查看宿主机正在运行的虚拟机

```
#!/usr/bin/env python
```

```
# coding: utf-8

# Get domain info via libvirt python API
# Tested with python2.6 and libvirt-python-0.10.2 on KVM host.

import libvirt
import sys

def createConnection():
    conn = libvirt.openReadOnly(None)
    if conn is None:
        print 'Failed to open connection to QEMU/KVM'
        sys.exit(1)
    else:
        print '-- Connection is created successfully --'
    return conn

def closeConnection(conn):
    """
    Arguments:
    - `conn`:
    """
    print
    try:
        conn.close()
    except:
        print 'Failed to close the connection'
        return 1
    print 'Connection is closed'

def getDomInfoByName(conn, name):
    """
    Arguments:
    - `conn`:
    - `NameError`:
    """
    print
    print '---- get domain info by name ----'
```

```

try:
    myDom = conn.lookupByName(name)
except:
    print 'Failed to find the domain with name "%s"' % name
    return 1

print "Dom id: %d name: %s" % (myDom.ID(), myDom.name())
print "Dom state: %s" % myDom.state(0)
print "Dom info: %s" % myDom.info()
print "memory: %d MB" % (myDom.maxMemory() / 1024)
print "memory status: %s" % myDom.memoryStats()
print "vCPUs: %d" % myDom.maxVcpus()

def getDomInfoByID(conn, id):
    """
    Arguments:
    - `conn`:
    - `id`:
    """
    print
    print '---- get domain info by ID ----'
    try:
        myDom = conn.lookupByID(id)
    except:
        print 'Failed to find the domain with ID "%d"' % id
        return 1
    print "Domain id is %d; Name is %s" % (myDom.ID(), myDom.name())

if __name__ == '__main__':
    name1 = "kvm-demo"
    name2 = "new"
    id1 = 5
    id2 = 6
    print '-- Get domain info via libvirt python API --'
    conn = createConnection()
    getDomInfoByName(conn, name1)
    getDomInfoByName(conn, name2)
    getDomInfoByID(conn, id1)

```

```
getDomInfoById(conn, id2)  
closeConnection(conn)
```



## **第三十八章   OpenStack**

### **38.1   OpenStack 简介**

#### **38.1.1   OpenStack 常用组件介绍**

### **38.2   安装前的准备工作**





## 第三十九章 Docker

### 39.1 关于 Docker

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。几乎没有性能开销，可以很容易地在机器和数据中心中运行。最重要的是，他们不依赖于任何语言、框架包括系统。

```
yum install -y docker.io
```

### 39.2 容器 VS. 虚拟机

容器为应用程序提供了隔离的运行空间：每个容器内都包含一个独享的完整用户环境空间，并且一个容器内的变动不会影响其他容器的运行环境，如图39.1所示。为了能达到这种效果，容器技术使用了一系列的系统级别的机制诸如利用 Linux namespaces 来进行空间隔离，通过文件系统的挂载点来决定容器可以访问哪些文件，通过 cgroups 来确定每个容器可以利用多少资源。此外容器之间共享同一个系统内核，这样当同一个库被多个容器使用时，内存的使用效率会得到提升。

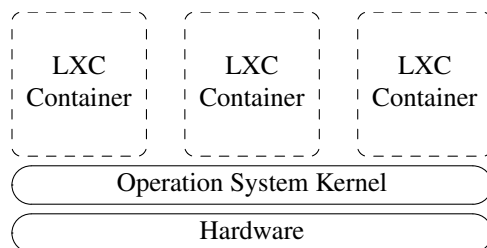


图 39.1 容器架构模型

对于系统虚拟化技术来说，虚拟层为用户提供了一个完整的虚拟机：包括内核在内的一个完整的系统镜像。CPU 虚拟化技术可以为每个用户提供一个独享且和其他用户隔离的系统环境，虚拟层可以为每个用户分配虚拟化后的 CPU、内存和 IO 设备资源，如图39.2所示。

```
git config --global user.name "Laven Liu"
git config --global user.email "air.man.six@gmail.com"
```

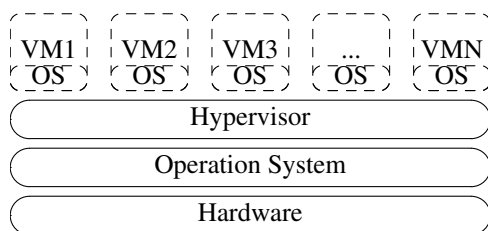


图 39.2 虚拟化架构模型

## 39.3 Docker 简介

### 39.3.1 Docker 与传统虚拟机建构对比

### 39.3.2 应用容器虚拟化定位

### 39.3.3 Docker 有哪些优势

### 39.3.4 Docker 应用场景

### 39.3.5 Docker 组成

## 39.4 安装及启动第一台 Docker 容器

## 39.5 测试环境

Docker 章节的演示在 Ubuntu14.04 LTS 64 位系统上进行演示<sup>1</sup>。演示环境所使用的机器列表如下：

表 39-1 Docker 演示环境机器一览

主机名	IP 地址	说明
master01.lavenliu.com	192.168.20.134	主 DNS
minion01.lavenliu.com	192.168.20.135	辅 DNS
minion02.lavenliu.com	192.168.20.136	客户端

### 39.5.1 安装 Docker

在安装 Docker 之前，需要设置 Docker 官方的镜像源。

```
% sudo apt-get update
% sudo apt-get install docker-engine
% sudo status docker
```

<sup>1</sup>在此之前，作者是在 CentOS6.5 64 位系统上进行演示的，遇到过 DeviceMapper 的问题，甚至有时 CentOS 还会发生 Kernel Panic 的现象，故使用了 Ubuntu 系统进行 Docker 章节的演示。

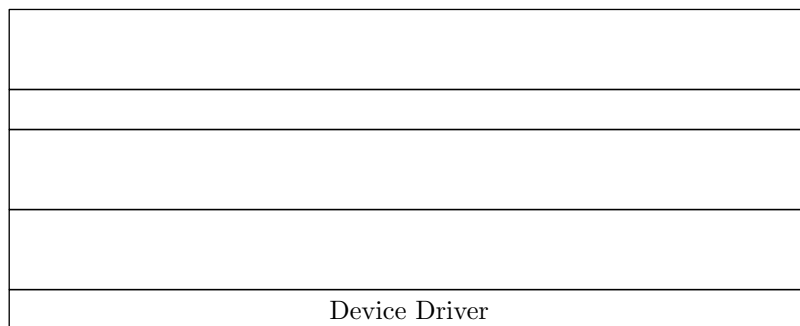


图 39.3 系统架构

### 39.5.2 运行第一台 Docker 容器

## 39.6 容器及镜像管理

### 39.6.1 容器管理

### 39.6.2 镜像管理

## 39.7 构建 Docker 镜像

### 39.7.1 手工构建

### 39.7.2 使用 Dockerfile 构建



## 第四十章 公有云

### 40.1 公有云简介



## 第八部分

## 网络部分





本部分主要介绍常用的网络协议及网络设备设备的基本使用。



# 第四十一章 TCP/IP

## 41.1 OSI 网络参考模型

应用层
表示层
会话层
传输层
网络层
数据链路层
物理层

图 41.1 OSI 七层模型

OSI 参考模型有 7 层。适用于这 7 层的基本原则简要概括如下：[4]

1. 应该在需要一个不同抽象体的地方创建一层。
2. 每一层都应该执行一个明确定义的功能。
3. 每一层功能的选择应该向定义国际标准化协议的目标看齐。
4. 层与层边界的选择应该使跨越接口的信息流最小。
5. 层应该足够多，保证不同的功能不会被混杂在同一层，但同时层数不能太多，以免体系结构变得过于庞大。

**物理层（physical layer）**：主要定义物理设备标准，如网线的接口类型、光纤的接口类型、各种传输介质的传输率等。它的主要作用是传输比特流<sup>1</sup>。这一层的数据叫做比特，网卡工作在此层。一般物理层较少关心网络入侵分析，而更关注于保证设备的电缆安全。

**数据链路层（data link layer）**：主要将从物理层接收的数据进行 MAC 地址（网卡的地址）的封装与解封装。这一层上的数据我们称为数据帧。在这一层工作的设备是交换机，数据通过交换机来传输。

**网络层（network layer）**：主要将从下层收到的数据进行 IP 地址的封装与解封装。在这一层工作的设备是路由器，我们称这一层的数据为数据包。

**传输层（transport layer）**：定义了一些传输数据的协议和端口号，比如 TCP<sup>2</sup>、UDP<sup>3</sup>。主要是将从下层接收的数据进行分段，到达目的地后进行数据重组。我们称这一层的数据

<sup>1</sup>即由 1、0 转化为电流强弱来进行传输，到达目的地后再转化为 1、0，也就是我们常说的数模转换与模数转换。

<sup>2</sup>传输控制协议，传输效率低，可靠性强，用于传输可靠性要求高且数据量大的数据。

<sup>3</sup>用户数据报协议，与 TCP 的特性恰恰相反，用于传输可靠性要求不高且数据量小的数据。

为段。

**会话层 (session layer):** 通过传输层 (端口号: 传输端口与接收端口) 建立数据传输的通路。主要在系统之间发起会话或接受会话请求 (设备之间可以通过 IP, 也可通过 MAC 或主机名来相互通信)。

**表示层 (presentation layer):** 主要是接收的数据进行解释、加密与解密、压缩与解压缩等 (也就是把计算机能够识别的信息转化为人可以识别的信息, 如图片、声音等)。

**应用层 (application layer):** 主要是一些终端应用, 可把它理解为应用层负责向用户或应用程序显示数据。

#### 41.1.1 TCP/IP 三次握手

#### 41.1.2 TCP/IP 四次挥手

## 第四十二章 H3C 交换机配置

### 42.1 配置 telnet 方式登录

#### 42.1.1 认证方式为 Scheme 时的登录配置

```
sudo apt-get install git
sudo apt-get install git-core
```

### 42.2 以太网接口配置

H3C 交换机的配置如下：  
配置方式与 Cisco 的很相似。

```
git config --global user.name "Laven Liu"
git config --global user.email "air.man.six@gmail.com"
```

#### 42.2.1 打开或关闭以太网接口

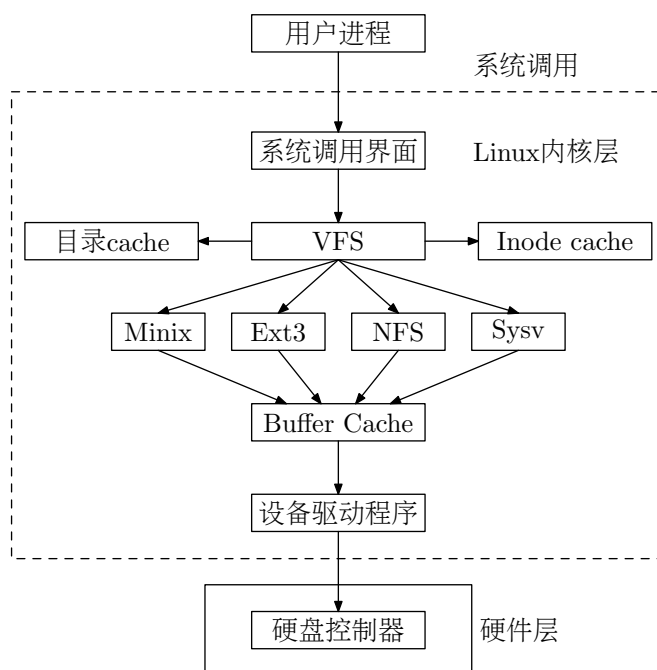


图 42.1 Linux 虚拟文件（测试图形）

#### 42.2.2 以太网端口基本属性配置

### 42.3 以太网接口的配置显示和维护

## 第九部分

## 附录部分





## 附录 A 书中的源码

小白在写书过程中，书中用到了很多插图及少量的 Bash 脚本。这些插图都是小白使用代码给绘制出来的。在此小白把自己所使用的工具及插图的源代码在本章里给出。有需要的同志，可以随意修改，成为自己所需的图形。脚本的代码这里不再给出。

为什么要使用这么古老的绘图工具？首先，小白使用的 Ubuntu 系统，上面没有安装 Office 套件。这种图形看起来很专业<sup>1</sup>；其次，绘制出的图形很精确<sup>2</sup>。然而，它也有很大缺点。首先，不可视化，很多人用不习惯（多用几次就好了）<sup>3</sup>；其次，不适合给领导做汇报用<sup>4</sup>。

### A.1 第 1 章插图源码

图1.1的源代码，使用的绘图工具为 metapost。如何编译生成 PDF 图形呢？

```
# mpost -tex=latex source.mp
```

源码如下：

```
verbatimtex
\documentclass[10pt]{article}
\usepackage{CJK}
\AtBeginDocument{\begin{CJK}{UTF8}{gbsn}}
\AtEndDocument{\end{CJK}}
\begin{document}
etex

input boxes;
input rboxes;

% system variables
ahangle := 40;

% fig0 is linux virtual file system topo
```

---

<sup>1</sup>科技之类的文章好像都是这种形式

<sup>2</sup>指哪画哪

<sup>3</sup>编译之后才能看到效果

<sup>4</sup>使用 PPT 的方式或许更好，PPT 越花哨越好

```

beginfig(0);
  boxit.a(btex Shell解释程序 etex);
  boxit.b(btex \begin{tabular}{c}
    用户程序\quad各种应用程序包 \\\
    系统命令\quad窗口命令\quad库函数
  \end{tabular} etex);
  boxit.c(btex 系统调用 etex);
  boxit.d(btex \begin{tabular}{c}
    核心层: \\\
    存储管理\quad进程管理 \\\
    设备管理\quad文件管理
  \end{tabular} etex);
  boxit.e(btex 硬件层 etex);
  a.sw = b.nw; a.se = b.ne;
  b.sw = c.nw; b.se = c.ne;
  c.sw = d.nw; c.se = d.ne;
  d.sw = e.nw; d.se = e.ne;
  a.se - a.sw = (140,0);
  b.se - b.sw = (140,0);
  c.se - c.sw = (140,0);
  d.se - d.sw = (140,0);
  e.se - e.sw = (140,0);

  circleit.cc(btex 用户 etex);
  circleit.dd(btex 用户 etex);

  cc.dx=cc.dy; dd.dx=dd.dy;

  cc.w = a.nw + (5,30);
  dd.e = a.ne - (5,-30);

  drawboxed(a,b,c,d,cc,dd);

  pair A[];
  A[1] = 1/3[a.nw,a.ne];
  A[2] = 2/3[a.nw,a.ne];
  draw cc.s -- A[1];
  draw dd.s -- A[2];

endfig;
end

```

## A.2 第2章插图源码

Libvirt 拓扑图,

```
.PS
h = .1
dh = .02
dw = .1
[
Userspacetools: [
boxht = 2.5*h; boxwid = 8*dw; boxrad = dh
movewid = 2*dh
A: box "virsh"; move
B: box "virt-manager"; move
C: box "virt-viewer"; move
D: box "virt-install"; move
E: box "other tools"
]
Userspace: [
boxht = 7*h; boxwid = 50*dw; boxrad = 2*dh
AA: box
] with .c at Userspacetools.c + (0,h/1.8)
F: "Userspace Management Tools" at last [].n - (0,h+2*dh)

Libvirt: box ht 4*h wid 25*dw "Libvirt" "(Libvirt API)" with .n at last [].s - (

Hypervisoroutline: [
Virtualtype: [
boxht = 2*h; boxwid = 12*dw; boxrad = dh
movewid = 2*dh
A: box "VMware"; move
B: box "Xen"; move
C: box "KVM"; move
D: box "Hyper-V"
]
Hypervisor: [
boxht = 5*h; boxwid = 50*dw; boxrad = 2*dh
AA: box
] with .c at Virtualtype.c + (0,h/1.8)
F: "Hypervisor Layer" at last [].n - (0,h+2*dh)
] with .n at Libvirt.s - (0,3*h)
```

```

XXX: [
VMwareoutline: [
VMware: [
boxht = 3.5*h; boxwid = 5*dw; boxrad = dh
movewid = 2*dh
VM1: box "Guest 1"; move
VM2: box "Guest 2"
] with .n at Hypervisoroutline.Virtualtype.A.s - (0,3*h)
box dashed ht last [].ht+dw wid last [].wid+dw at last []
]

move 5*dh

Xenoutline: [
Xen: [
boxht = 3.5*h; boxwid = 5*dw; boxrad = dh
movewid = 2*dh
VM1: box "Dom0" "Guest"; move
VM2: box "DomU" "Guest"
]
box dashed ht last [].ht+dw wid last [].wid+dw at last []
]

move 5*dh

Kvmoutline: [
Kvm: [
boxht = 1.75*h; boxwid = 5*dw; boxrad = dh
movewid = 2*dh
VM1: [
Qemu1: box "Qemu"
Guest01: box "Guest 1" with .n at Qemu1.s
]
move

VM2: [
Qemu1: box "Qemu"
Guest01: box "Guest 2" with .n at Qemu1.s
]
]
]

```

```

box dashed ht last [].ht+dw wid last [].wid+dw at last []
]

move 5*dh

Hypervoutline: [
Hyperv: [
boxht = 3.5*h; boxwid = 5*dw; boxrad = dh
movewid = 2*dh
VM1: box "Guest 1"; move
VM2: box "Guest 2"
]
box dashed ht last [].ht+dw wid last [].wid+dw at last []
]
] with .n at last [].s - (0,3*h)

arrow from Userspacetools.A.s to Libvirt.nw
arrow from Userspacetools.B.s to 1/2 <Libvirt.nw,Libvirt.n>
arrow from Userspacetools.C.s to Libvirt.n
arrow from Userspacetools.D.s to 1/2 <Libvirt.n,Libvirt.ne>
arrow from Userspacetools.E.s to Libvirt.ne

arrow from Libvirt.s to 3rd [].Hypervisor.n

arrow from Hypervisoroutline.Virtualtype.A.s to XXX.VMwareoutline.VMware.n
arrow from Hypervisoroutline.Virtualtype.B.s to XXX.Xenoutline.Xen.n
arrow from Hypervisoroutline.Virtualtype.C.s to XXX.Kvmoutline.Kvm.n
arrow from Hypervisoroutline.Virtualtype.D.s to XXX.Hypervoutline.Hyperv.n

]
.PE

    libvirt 与 node 与 hypervisor,

.PS
[
A: [
    box dashed wid 0.7 ht 0.75 rad 0.05 "Domain 1"
    move 0.2
    box dashed wid 0.7 ht 0.75 rad 0.05 "Domain 2"
    move same
    box dashed wid 0.7 ht 0.75 rad 0.05 "Domain N"

```

```

]

B: [
    box wid 2.5 ht 0.2 rad 0.1 "Hypervisor"
    ] with .n at A .s - (0,0.15)
]

box ht last [].ht+0.45 wid last [].wid+0.3 at last [] + (0,0.05)
"Node" at last box .n - (0,0.15)
.PE

```

KVM 桥接网络,

```

.PS
[
A: [
    box dashed wid 0.7 ht 0.75 rad 0.05 "Domain 1"
    move 0.2
    box dashed wid 0.7 ht 0.75 rad 0.05 "Domain 2"
    move same
    box dashed wid 0.7 ht 0.75 rad 0.05 "Domain N"
]

B: [
    box wid 2.5 ht 0.2 rad 0.1 "Hypervisor"
    ] with .n at A .s - (0,0.15)
]

box ht last [].ht+0.45 wid last [].wid+0.3 at last [] + (0,0.05)
"Node" at last box .n - (0,0.15)
.PE

```

saltstack 通讯模型,

```

.PS
scale = 2.54

define port { [
    Port: box wid $1 ht $2 $3 "$4"
    ]
}

Master: box wid 5.5 ht 2 rad 0.2;

```

```
"master" at Master.n below;
```

```
Port4505: [ port(2,1,"Publish Port",4505) ] with .se at Master.s - (0.25,0);
```

```
Port4506: [ port(2,1,"Return Port",4506) ] with .sw at Master.s + (0.25,0);
```

```
Minion1: [ box wid 1.5 ht 1 rad 0.1 "minion1" ] with .nw at Master.sw - (0,1.5);
```

```
Minion2: [ box wid 1.5 ht 1 rad 0.1 "minion2" ] with .ne at Master.se - (0,1.5);
```

```
line -> from 1/3 <Port4505.sw,Port4505.se> to 1/3 <Minion1.nw,Minion1.ne>;
```

```
line -> from 1/3 <Port4505.se,Port4505.sw> to 1/3 <Minion2.nw,Minion2.ne>;
```

```
line -> from 2/3 <Minion1.nw,Minion1.ne> to 1/3 <Port4506.sw,Port4506.se> dashed
```

```
line -> from 2/3 <Minion2.nw,Minion2.ne> to 2/3 <Port4506.sw,Port4506.se> dashed
```

```
.PE
```

ELK 拓扑图,

```
.PS
```

```
define shipper {
```

```
box wid $1 ht $2 rad $3 "Shipper" "LogStash";
```

```
}
```

```
[
```

```
shipper(0.85,0.4,0.02);
```

```
]
```

```
[
```

```
shipper(0.85,0.4,0.02);
```

```
] with .n at last [].s - (0,0.35);
```

```
[
```

```
shipper(0.85,0.4,0.02);
```

```
] with .n at last [].s - (0,0.35);
```

```
Broker: circle rad 0.5 "Borker" "Redis" "192.168.20.138" with .w at 2nd [].e + (
```

```
move right 0.35;
```

```
Indexer: box wid 1.15 ht 0.55 rad 0.02 "Indexer" "LogStash" "192.168.20.139";
```

```
move same;
```

```

Elastic: box wid 1.15 ht 0.75 rad 0.02 "Search &" "Storage" "ElasticSearch"

move same;

Kibana: box wid 1.15 ht 1.9 rad 0.02 "Web" "Interface" "LogStash" "192.168.

point01 = 1st [].e.y;
Line1: line right from 1st [].e to (Broker.n.x,point01);
line -> from Line1.end to Broker.n;

point02 = last [].e.y;
Line2: line right from last [].e to (Broker.s.x,point02);
line -> from Line2.end to Broker.s;

line -> from 2nd [] .e to Broker.w;
line -> from Broker.e to Indexer.w;
line -> from Indexer.e to Elastic.w;
line -> from Kibana.w to Elastic.e;
.PE

```

AWK 工作流程图,

```

.PS
A: box ht 0.8 rad 0.08 "Line 1" "Line 2" "Line 3" "Line 4" "..."
INFILE: "\fBInput file\fP" with .n at A.s - (0,0.15)
B: box width 1.8 "Read a Line" with .nw at A.ne + (0.5,0)
C: box width 1.8 "Execute awk commands" "in the \fBbody\fP block on the lin
D: box width 1.8 "\fBRepeat\fP for the next line until" "end of the input f
E: box width 1.8 rad0.1 "Execute awk commands in" "the \fBEND\fP Block" at
F: box width 1.8 "Execute awk commands in" "the \fBBEGIN\fP Block" at B +

L1: line chop 0.01 chop 0.9 from 1st box at 1/3 <A.e,A.ne> to B ->

L2: line down from B to C -> chop
L3: arrow down from C to D chop
L4: arrow down from D to E chop

L5: line down 2 from L1 .center
L6: line right from L5.end to L4.center
L7: arrow from F.s to B.n
.PE

```



SED 工作流程图,

.PS

A: box ht 0.8 rad 0.08 "Line 1" "Line 2" "Line 3" "Line 4" "..."

INFILE: "\fBInput file\fP" with .n at A.s - (0,0.15)

B: box width 1.8 "\fBRead\fP a line into the" "pattern space" with .nw at A.ne +

C: box width 1.8 "\fBExecute\fP given sed commands" "on the pattern space" at B

D: box width 1.8 "\fBPrint\fP the pattern space" "and empty it" at C - (0,0.8)

E: box width 1.8 "\fBRepeat\fP for the next line until" "end of the input file"

F: box width 0.75 ht 0.35 rad 0.05 "Line 1" with .w at B.e + (0.35,0)

PATTERN: "\fBPattern space\fP" with .n at F.s - (0,0.15)

G: circle "Output" with .w at D.e + (0.35,0)

L1: line chop 0.01 chop 0.9 from 1st box at 1/3 <A.e,A.ne> to B ->

L2: line down from B to C -> chop

L3: arrow down from C to D chop

L4: arrow down from D to E chop

L5: line dashed from B.e to F.w ->

L6: line dashed from C.e to F.w ->

L7: line dashed from D.e to G.w ->

L8: line down 2.85 from L1 .center to E.sw - (0.2,0.2)

L9: line right from L8.end to E.s - (0,0.2)

L10: line from E.s to L9.end

.PE

Linux 模型,

verbatimtex

\documentclass[10pt]{article}

\usepackage{CJK}

\AtBeginDocument{\begin{CJK}{UTF8}{gbsn}}

\AtEndDocument{\end{CJK}}

\begin{document}

etex

input boxes;

input rboxes;

% system variables

```

ahangle := 40;

% fig0 is linux virtual file system topo
outputtemplate := "vfs.mps";
beginfig(0);
    defaultfont:="ptmr8r";
    boxit.a(btex 用户进程 etex);
    boxit.b(btex 系统调用界面 etex);
    boxit.c(btex VFS etex);
    boxit.d(btex Ext3 etex);
    boxit.e(btex Buffer Cache etex);
    boxit.f(btex 设备驱动程序 etex);
    boxit.g(btex 硬盘控制器 etex);
    boxit.minix(btex Minix etex);
    boxit.nfs(btex NFS etex);
    boxit.sysv(btex Sysv etex);
    boxit.direc(btex 目录cache etex);
    boxit.inode(btex Inode cache etex);
    boxit.hard(btex etex);

    % Len is the box's length
    % Hig is the box's hight
    numeric Len;
    numeric Hig;
    Len := 65;
    Hig := 14pt;
    a.e - a.w = (Len,0); a.n - a.s = (0,Hig);
    b.e - b.w = (Len,0); b.n - b.s = (0,Hig);
    c.e - c.w = (Len,0); c.n - c.s = (0,Hig);
    d.e - d.w = (35,0); d.n - d.s = (0,Hig);
    minix.e - minix.w = (35,0); minix.n - minix.s = (0,Hig);
    nfs.e - nfs.w = (35,0); nfs.n - nfs.s = (0,Hig);
    sysv.e - sysv.w = (35,0); sysv.n - sysv.s = (0,Hig);
    e.e - e.w = (Len,0); e.n - e.s = (0,Hig);
    f.e - f.w = (Len,0); f.n - f.s = (0,Hig);
    g.e - g.w = (Len,0); g.n - g.s = (0,Hig);
    direc.e - direc.w = (Len,0); direc.n - direc.s = (0,Hig);
    inode.e - inode.w = (Len,0); inode.n - inode.s = (0,Hig);

    % Dis is the hight between the boxes
    numeric Dis;

```

```

Dis := 20;
a.s - b.n = (0,30);
b.s - c.n = (0,Dis);
c.s - d.ne = (5,Dis);
d.se - e.n = (-5,Dis);
c.s - nfs.nw = (-5,Dis);
d.w - minix.e = (10,0);
sysv.w - nfs.e = (10,0);
e.s - f.n = (0,Dis);
f.s - g.n = (0,30);
c.w - direc.e = (Dis,0);
c.e - inode.w = (-Dis,0);
hard.c = g.c;
hard.e - hard.w = (100,0);
hard.n - hard.s = (0,34);
drawboxed(a,b,c,d,e,f,g,minix,nfs,sysv,direc,inode,hard);

% draw the connectors
drawarrow a.s -- b.n;
drawarrow b.s -- c.n;

drawarrow c.s -- minix.n;
drawarrow c.s -- d.n;
drawarrow c.s -- nfs.n;
drawarrow c.s -- sysv.n;

pair A[];
A[1] = 1/5[e.nw,e.ne];
A[2] = 2/5[e.nw,e.ne];
A[3] = 3/5[e.nw,e.ne];
A[4] = 4/5[e.nw,e.ne];
drawarrow minix.s -- A[1];
drawarrow d.s -- A[2];
drawarrow nfs.s -- A[3];
drawarrow sysv.s -- A[4];

drawarrow e.s -- f.n;
drawarrow f.s -- g.n;

drawarrow c.w -- direc.e;
drawarrow c.e -- inode.w;

```

```

% draw the outline
pair B[];
B[1] = direc.w - (5,-56);
B[2] = inode.e - (-5,-56);
B[3] = inode.e - (-5,119);
B[4] = direc.w - (5,119);

draw B[1] -- B[2] -- B[3] -- B[4] -- cycle dashed evenly;
label.rt(btex 硬件层 etex,hard.e);
label.rt(btex Linux内核层 etex,b.e+(15,0));
label.rt(btex 系统调用 etex,a.se+(15,-5));
endfig;

outputtemplate := "os-arch.mps";
beginfig(1);
  boxjoin(a.nw=b.sw; a.ne=b.se);
  boxit.a(btex Device Driver etex);
  boxit.b();
  boxit.c();
  boxit.d();
  boxit.e();

  numeric Len[],Hig[];
  Len[1] := 300;
  Hig[1] := 15;
  Hig[2] := 30;
  a.e - a.w = (Len[1],0); a.n - a.s = (0,Hig[1]);
  b.n - b.s = (0,Hig[2]); c.n - c.s = (0,Hig[2]);
  d.n - d.s = (0,Hig[1]); e.n - e.s = (0,Hig[2]);
  drawboxed(a,b,c,d,e);

endfig;

outputtemplate := "osarch.mps";
beginfig(2);
  boxjoin(a.nw=b.sw; a.ne=b.se);
  boxit.a(btex Block Device Driver etex);
  boxit.b(btex Volume Manager etex);
  boxit.c(btex File Systems etex);
  boxit.d(btex VFS etex);

```

```

boxjoin(e.nw=f.sw; e.ne=f.se);
boxit.e(btex Ethernet etex);
boxit.f(btex IP etex);
boxit.g(btex TCP/UDP etex);
boxit.h(btex Socket etex);

e.nw = a.ne; e.sw = a.se;
numeric Len[],Hig[];
Len[1] := 95;
Hig[1] := 15;
Hig[2] := 30;
a.e - a.w = (Len[1],0); a.n - a.s = (0,Hig[1]);
b.n - b.s = (0,Hig[1]); c.n - c.s = (0,Hig[1]);
d.n - d.s = (0,Hig[1]); e.n - e.s = (0,Hig[1]);
f.n - f.s = (0,Hig[1]); g.n - g.s = (0,Hig[1]);

drawboxed(a,b,c,d,e,f,g,h);
endfig;
end

```

### A.3 第14章插图源码

图39.1的源码，使用的是pic工具进行绘制的。源码为：

```

.PS
define container { [
    box dashed wid 0.7 ht 0.75 rad 0.05 "$1" "Container"
] }

define layer {
    box wid 2.5 ht 0.2 rad 0.1 "$1"
}

V1: container(LXC)
move 0.2
V2: container(LXC)
move same
V3: container(LXC)

move to V1.sw - 0,0.15

```

```

L1: layer(Operation System Kernel)
move to L1.s - 0,0.05
down
L2: layer(Hardware)
.PE

```

图39.2的源码，使用的 `pic` 工具进行绘制的。源码为：

```

.PS
define vm { [
    box dashed wid 0.4 ht 0.4 rad 0.05 "$1"
    box dashed wid 0.4 ht 0.13 rad 0.05 "OS" with .s at last box .s
] }

define layer {
    box wid 2.5 ht 0.2 rad 0.1 "$1"
}

V1: vm(VM1)
move 0.125
V2: vm(VM2)
move same
V3: vm(VM3)
move same
V4: vm(...)
move same
V5: vm(VMN)

move to V1.sw - 0,0.15

L1: layer(Hypervisor)
move to L1.s - 0,0.05
down
L2: layer(Operation System)
move to L2.s - 0,0.05
down
L3: layer(Hardware)
.PE

```

如何编译呢？

```

pic /path/to/filename.pic | groff | ps2eps > /path/to/filename.eps
epstopdf /path/to/filename.eps

```

```
evince /path/to/filename.pdf
```

```
# 如果是Windows系统，则可以使用Adobe Reader或福昕阅读器进行查看
```





## 参考文献

- [1] Randal E. Bryant and David R. O'Hallaron. 深入理解计算机系统. 机械工业出版社, second edition, 2011.
- [2] Someone. 百度百科. 百度出版社, 2014.
- [3] W. Richard Stevens and Stephen A. Rago. *UNIX 环境高级编程*. 人民邮电出版社, second edition, 2011.
- [4] Andrew S. Tanenbaum David J. Wetherall. 计算机网络. 清华大学出版社, 第 5 版 edition, 2012.
- [5] 克莱因. 古今数学思想. 上海科学技术出版社, 2002.