（京峰教育）

2019 京峰教育 Linux 云计算

高级运维教程

2019 年 2 月 12 日 v5 版

Contents

第1章　Linux 零基础入门篇

1.1　为什么要学习 Linux

我们为什么要学习 Linux？我们现在的处境是什么？我们想达到什么样的目标？

在谈到这三个问题，相信我们每个人都有自己的答案，我们来自不同的家庭，各种经历都不一样，但是最终的目标都是希望自己有一样本事，能在这个社会立足。

这个社会是一个竞争残酷的社会，没有人会同情你跑的慢，那我们难道停滞不前吗，我们对自己负责吗，对父母负责吗？我们哪什么跟我们的初心交代，想想我们刚步入小学堂的时候，心里就狠狠下决心，以后一定要有一番出息，可是 20、30 多年时光过去了，我们得到了什么，我们到底是在追求什么？目标方向又在哪里呢？

我们在生活中经历各种挫折，感情、生活，很多小的纠结把我们缠在里面，当我们某天悔悟，时光已不在。所以今天下定决心了，就要去行动，去改变；人最可怕的事情在自以为稳稳当当地位待得太久，等到巨变来的时候，已经晚了。

1.2 Linux 操作系统简介

Linux 是一套免费使用和自由传播的类 Unix 操作系统，是一个基于 POSIX 和 UNIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。它能运行主要的 UNIX 工具软件、应用程序和网络协议。它支持 32 位和 64 位 CPU 硬件。Linux 继承了 Unix 以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

严格来讲，Linux 这个词本身只表示 Linux 内核，人们已经习惯了用 Linux 来形容整个基于 Linux 内核，并且使用 GNU（自由软件协议）工程各种工具和数据库的操作系统。

1991 年的 10 月 5 日，Linux 创始人林纳斯·托瓦兹(Linus Torvalds)在 comp.os.minix 新闻组上发布消息，正式向外宣布 Linux 内核的诞生，1994 年 3 月，Linux 1.0 发布，代码量 17 万行，当时是按照完全自由免费的协议发布，随后正式采用 GPL（General Public License 的缩写，是一份 GNU 通用公共授权）协议。

Linux 具有如下优点：

- 稳定、免费或者花费少
- 安全性高
- 多任务，多用户
- 耗资源少
- 由于内核小，所以它可以支持多种电子产品，如：Android 手机、PDA 等。

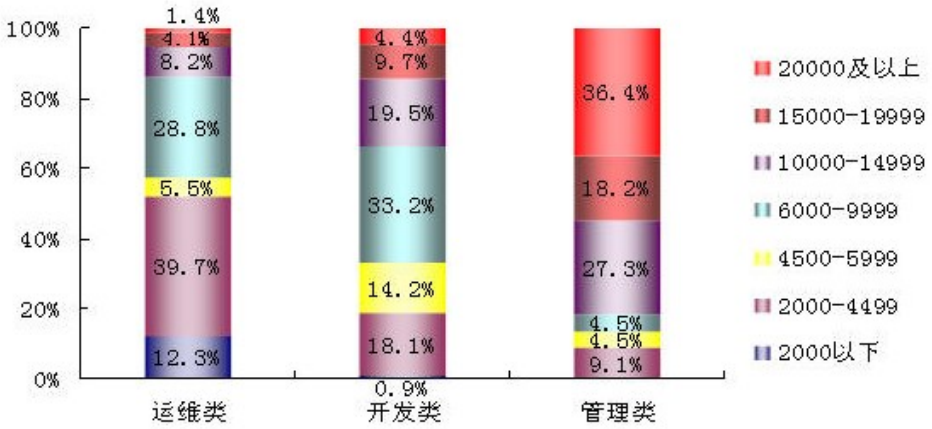
目前主流 Linux 操作系统有 RedHat、AIX、CentOS、Ubuntu、Debian、suse 等，其中在企业中用的最多的是 CentOS6.5 centos7 操作系统。

1.3 Linux 技术发展趋势

随着 IT 产业的不断发展，用户对网站体验要求也越来越高，而目前主流网站后端承载系统都是 Linux 系统，目前 Android 手机全部基于 Linux 内核研发。企业大数据、云存储、虚拟化等先进技术都是基于 Linux 系统。

1.4 Linux 企业就业薪资

2010 年据有关权威部门统计：将来几年内我国软件行业的从业机会十分庞大，中国每年对软件人才的需求将达到 50 万人左右。而对于 Linux 专业人才的就业前景，更是广阔；据悉在未来 5-10 年内 Linux 专业人才的需求将达到 120 万+！尤其是有经验的资深的 Linux 工程师目前非常的缺乏，薪资也是非常诱人，平均月薪都是 15-20K，能力强的薪资更高。



所以机会对每个人都是公平的，关键是我们每个人如何去行动，选择大于努力。

1.5 Windows 系统简介

在安装 Linux 系统之前，先来了解 windows 系统结构，windows 系统一般是安装在 C 盘系统盘，同样 Linux 也有类似的系统盘（/boot 根分区），Linux 通常分区为（根分区/、swap 分区），**Linux 系统以文件的存储方式**，所有的文件都是存储在某个目录下的，类似于 windows 的文件夹。

对于文件系统的属性来说，windows 文件系统类型一般是 ntfs、fat32 等，而 Linux 文件系统类型则为 ext2、ext3、ext4\xfs 等**（文件系统：是操作系统用于明确磁盘或分区上的文件的方法和数据结构，文件系统由三部分组成：与文件管理有关软件、被管理文件以及实施文件管理所需数据结构。）**

1.6 计算机硬盘分区简介

硬盘分区有三种，**主磁盘分区、扩展磁盘分区、逻辑分区。**

一个硬盘主分区至少有 1 个，最多 4 个，扩展分区可以没有，最多 1 个。且主分区+扩展分区总共不能超过 4 个，逻辑分区可以有若干个。

在 windows 下激活的主分区是硬盘的启动分区，他是独立的，也是硬盘的第一个分区，正常分的话就是 C 区。

在 linux 下主分区和逻辑分区都可以用来放系统，引导 os 开机，grub 会兼容 windows 系统开机启动。分出主分区后，其余的部分可以分成扩展分区，一般是剩下的部分全部分成扩展分区，也可以不全分，那剩的部分就浪费了。

扩展分区是不能直接用的，他是以逻辑分区的方式来使用的，所以说扩展分区可分成若干逻辑分区。他们的关系是包含的关系，所有的逻辑分区都是扩展分区的一部分。

在 linux 中第一块硬盘分区为 hda 分区(或者是 **sda** 分区)，主分区编号为 hda1-4，逻辑分区从 5 开始。

硬盘的容量=主分区的容量+扩展分区的容量　扩展分区的容量=各个逻辑分区的容量之和。

主分区也可成为“引导分区”，会被操作系统和主板认定为这个硬盘的第一个分区。所以 C 盘永远都是排在所有磁盘分区的第一的位置上。

MBR（主引导记录）的分区表（主分区表）只能存放 4 个分区，如果要分更多的分区的话就要一个扩展分区表（EBR），扩展分区表放在一个系统 ID 为 0x05 的主分区上，这个主分区就是扩展分区，扩展分区能可以分若干个分区，每个分区都是个逻辑分区

1.7　Windows 与 Linux 区别

安装 Linux 系统是每一个初学者的第一个门槛。在这个过程中间，最大的困惑莫过于给硬盘进行分区。虽然现在各种发行版本的 Linux 已经提供了友好的图形交互界面，但是很多人还是感觉无从下手。这其中的原因主要是不清楚 Linux 的分区规定。就好比如果我们了解了 windows 分区的规则，系统盘 C、数据盘 D 等，就很好分区了。

在 Linux 中规定，每一个硬盘设备最多只能有 4 个主分区（其中包含扩展分区）构成，任何一个扩展分区都要占用一个主分区号码，也就是在一个硬盘中，主分区和扩展分区一共最多是 4 个。

Microsoft Windows,是美国微软公司研发的一套操作系统，它问世于 1985 年，起初仅仅是 Microsoft-DOS 模拟环境，后续的系统版本由于微软不断的更新升级，不但易用，也慢慢的成为家家户户人们日常办公和使用的最喜爱的操作系统。

而 Linux 是一套免费使用和自由传播的类 Unix 操作系统，是一个基于 POSIX 和 UNIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。它能运行主要的 UNIX 工具软件、应用程序和网络协议。它支持 32 位和 64 位硬件。Linux 继承了 Unix 以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统，主要用于服务器领域。

1.8 虚拟机软件安装及配置

下面正式来安装 Linux 系统，安装系统前需要下载如下软件：

- ✓ **VMware workstation 10.0**
- ✓ **CentOS 7.x x86_64**

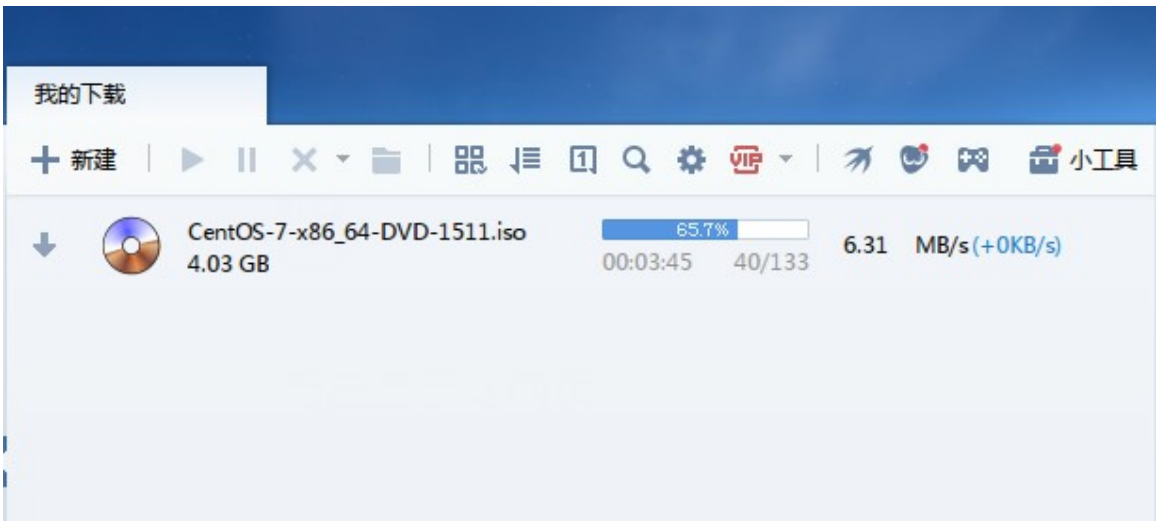
安装图解如下：

Vmware10.0 下载路径:

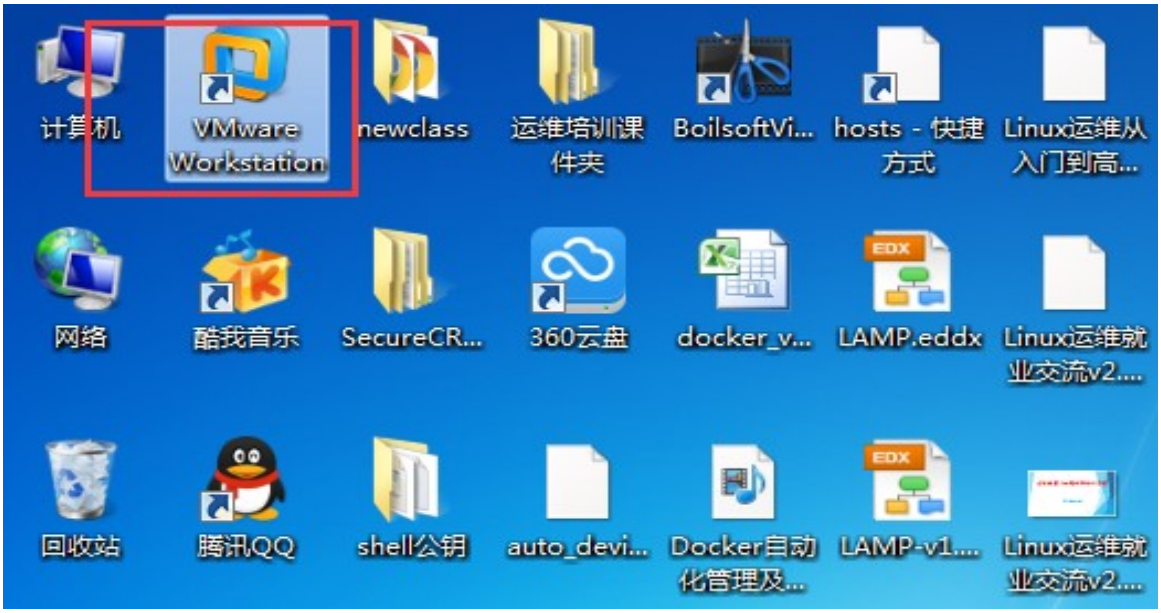
http://download3.vmware.com/software/wkst/file/VMware-workstation-full-10.0.1-1379776.exe

ISO 下载路径：

http://124.205.69.165/files/706900000291EB25/mirror.math.princeton.edu/pub/centos/7/isos/x86_64/CentOS-7-x86_64-DVD-1511.iso

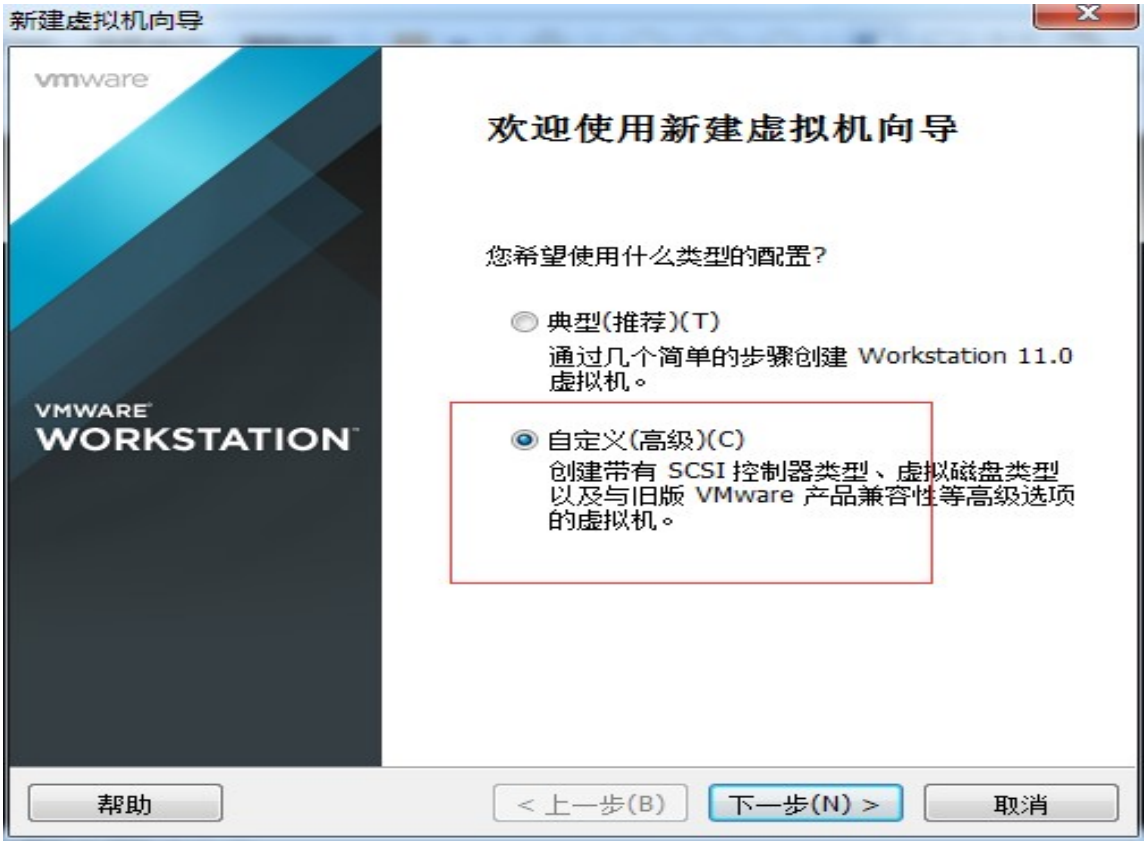


第一步，打开并新建虚拟机如下图：

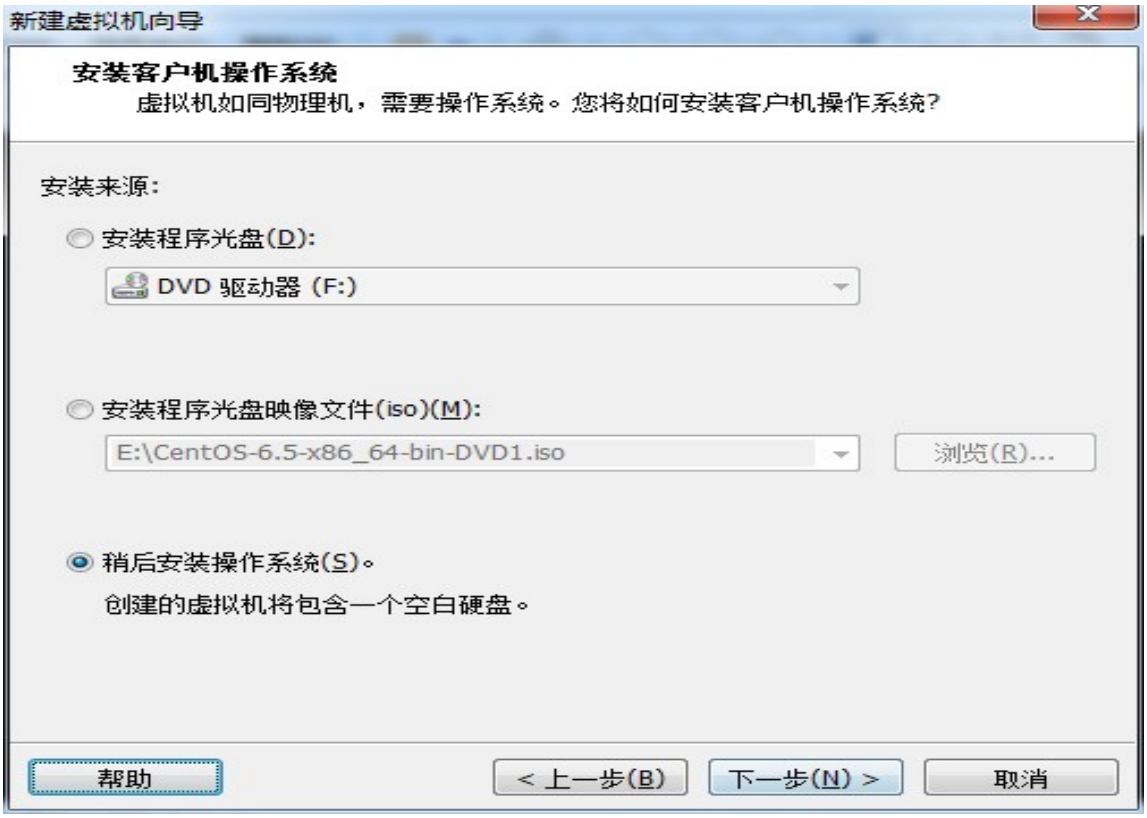


点击创建新的虚拟机：

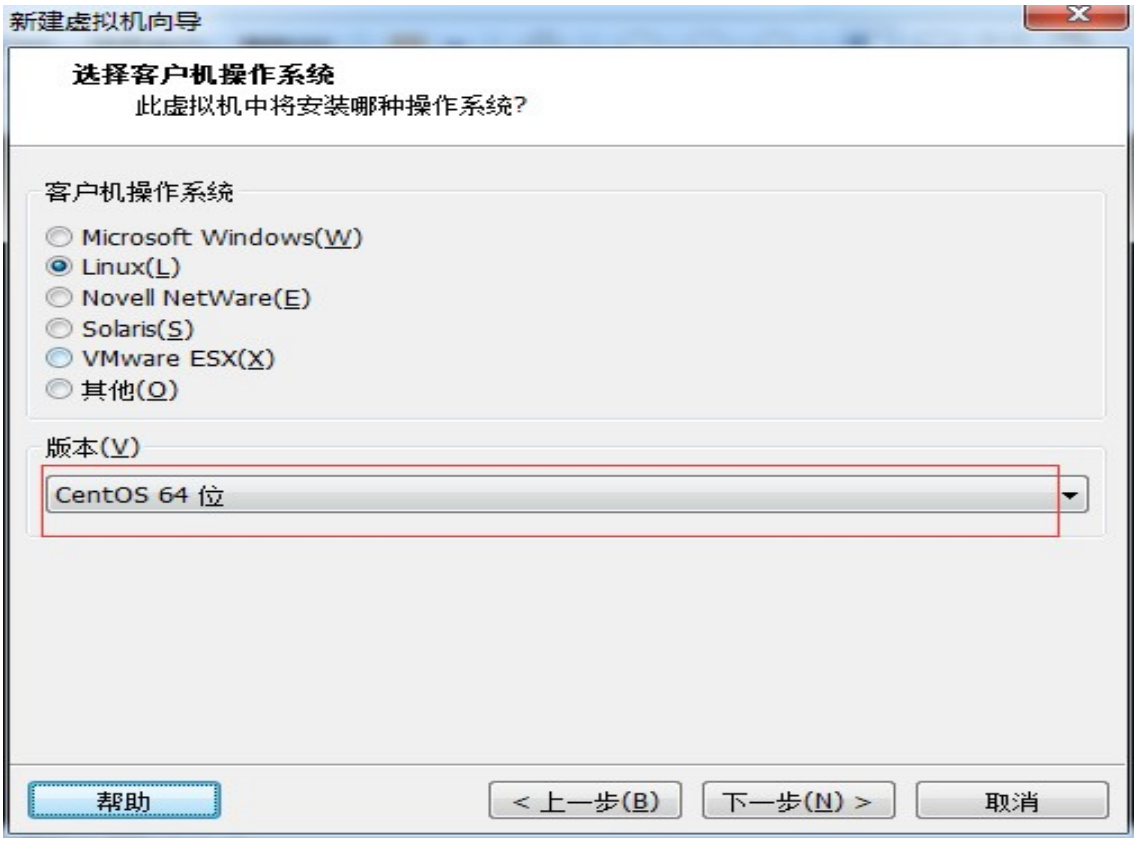
第二步，选择自定义，如下图：



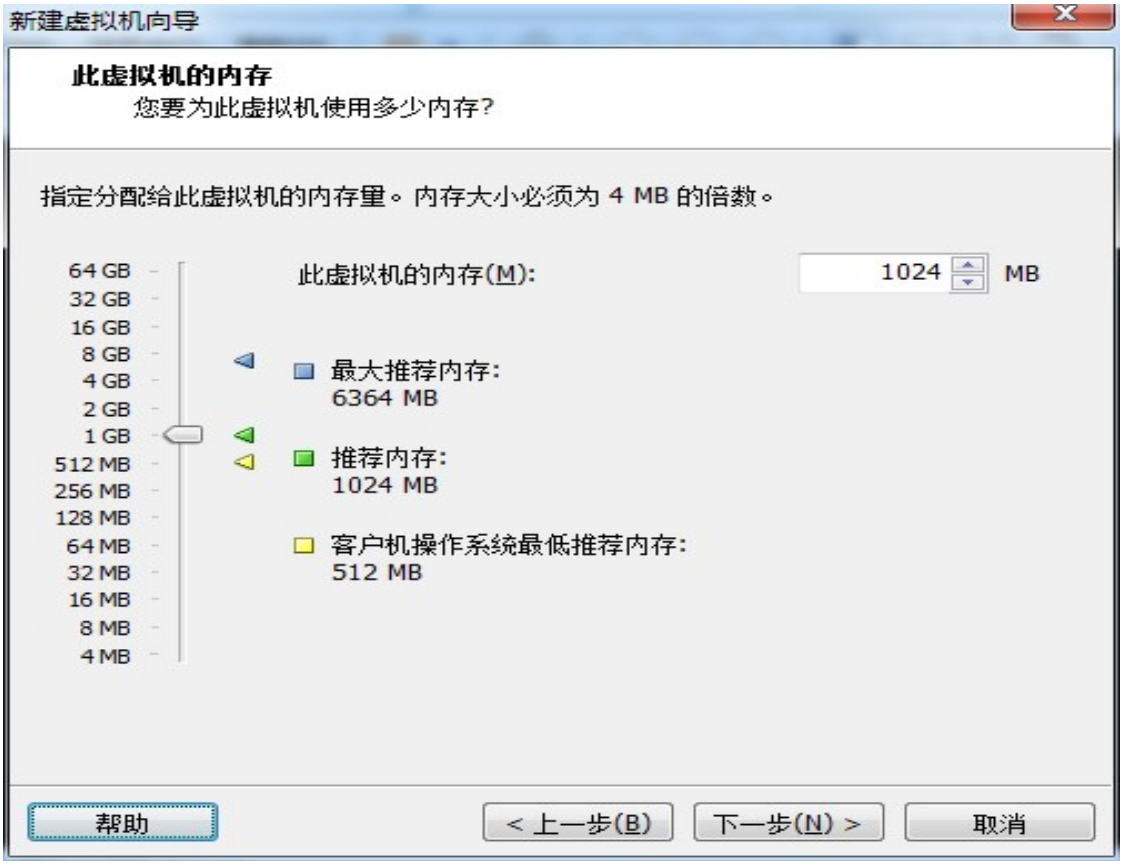
第三步选择“稍后安装操作系统”，如下图:



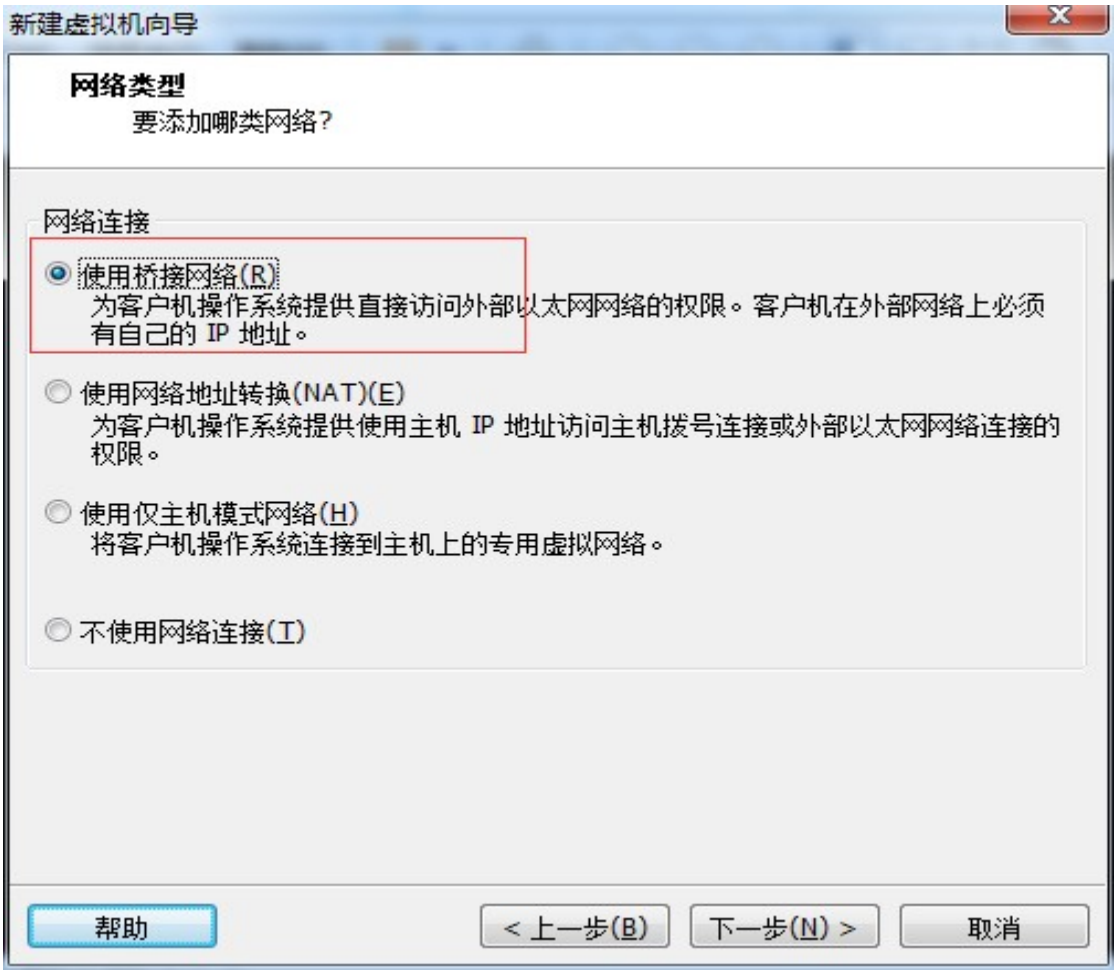
第四步，选择客户机操作系统类型如下图:



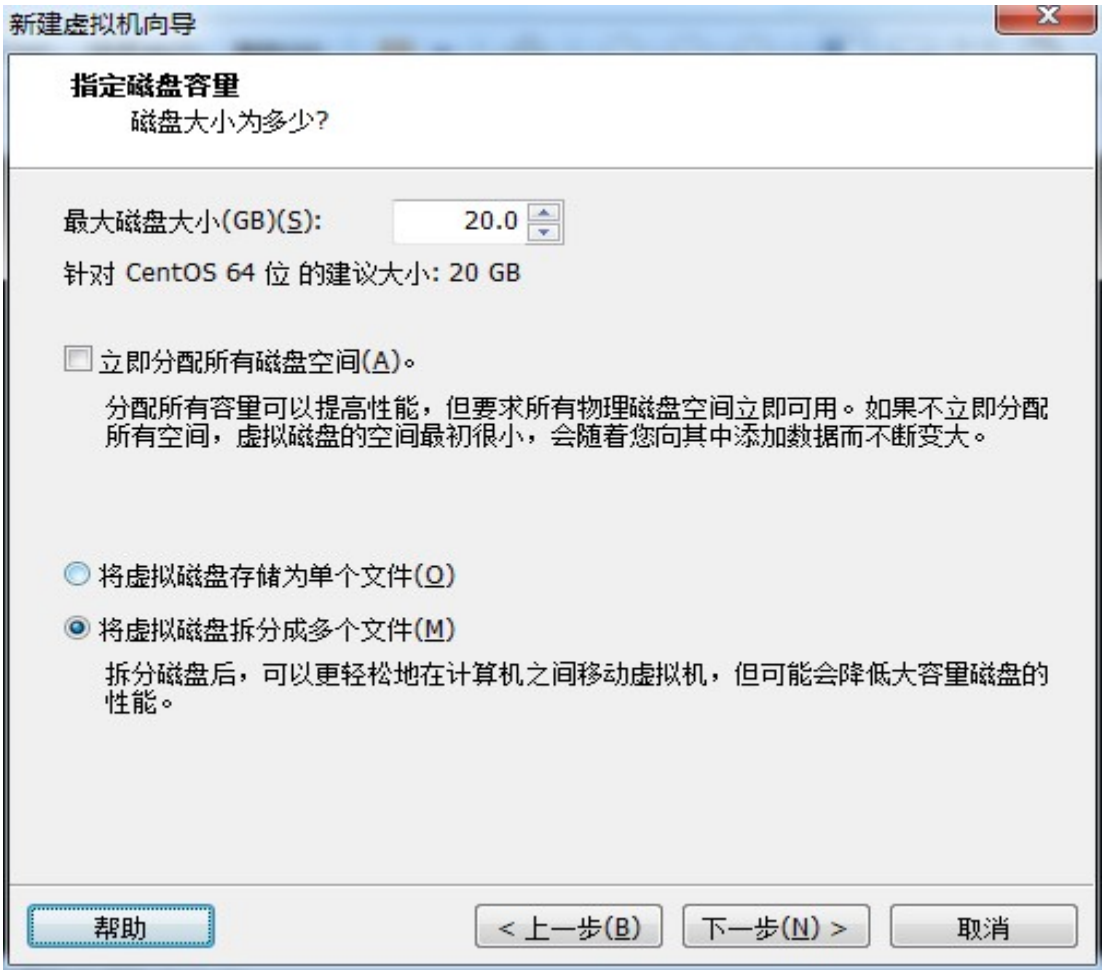
第五步，选择内存大小



第六步，选择虚拟机网卡模式

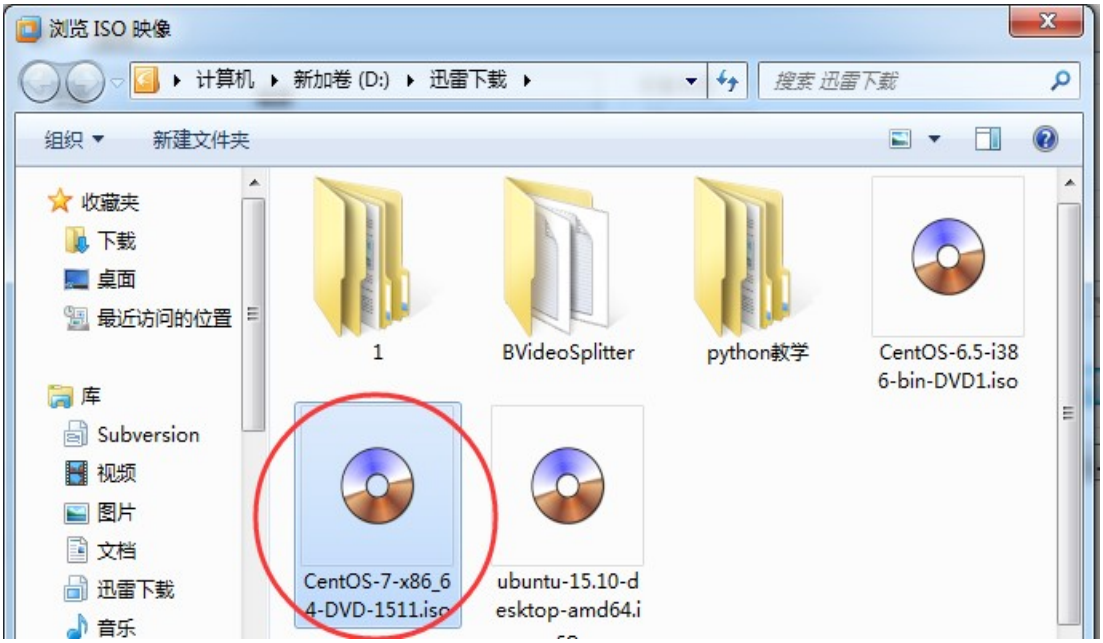


第五步，设置虚拟机硬盘大小为 20G，最低不能小于 5G，如下图:



第六步，虚拟机新建完成，如下图：

第七步，添加 CentOS7.2 ISO 镜像，如下图：

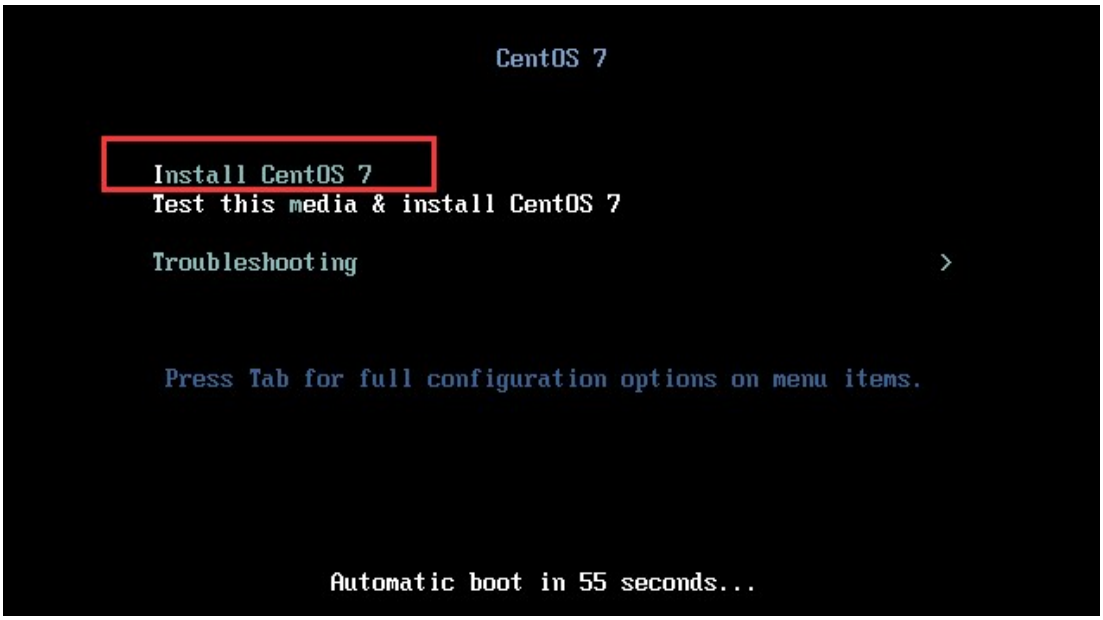


自此，虚拟机新建完成，接下来点击“启动此虚拟机”

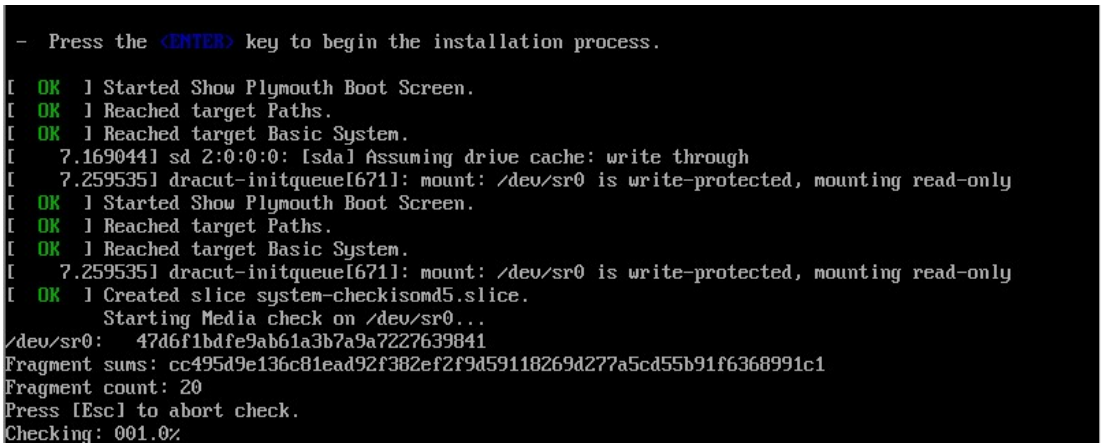
1.9 CentOS7 系统部署步骤

进行 Linux 系统安装，Linux 系统安装图解如下：

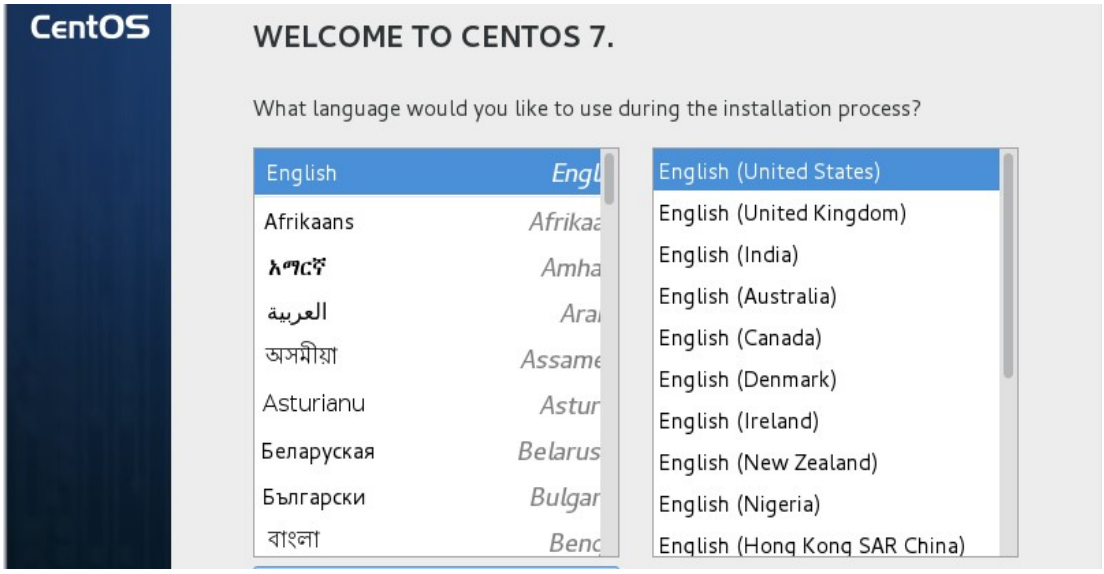
第一步，进入安装界面，直接按 Enter 回车键即可。



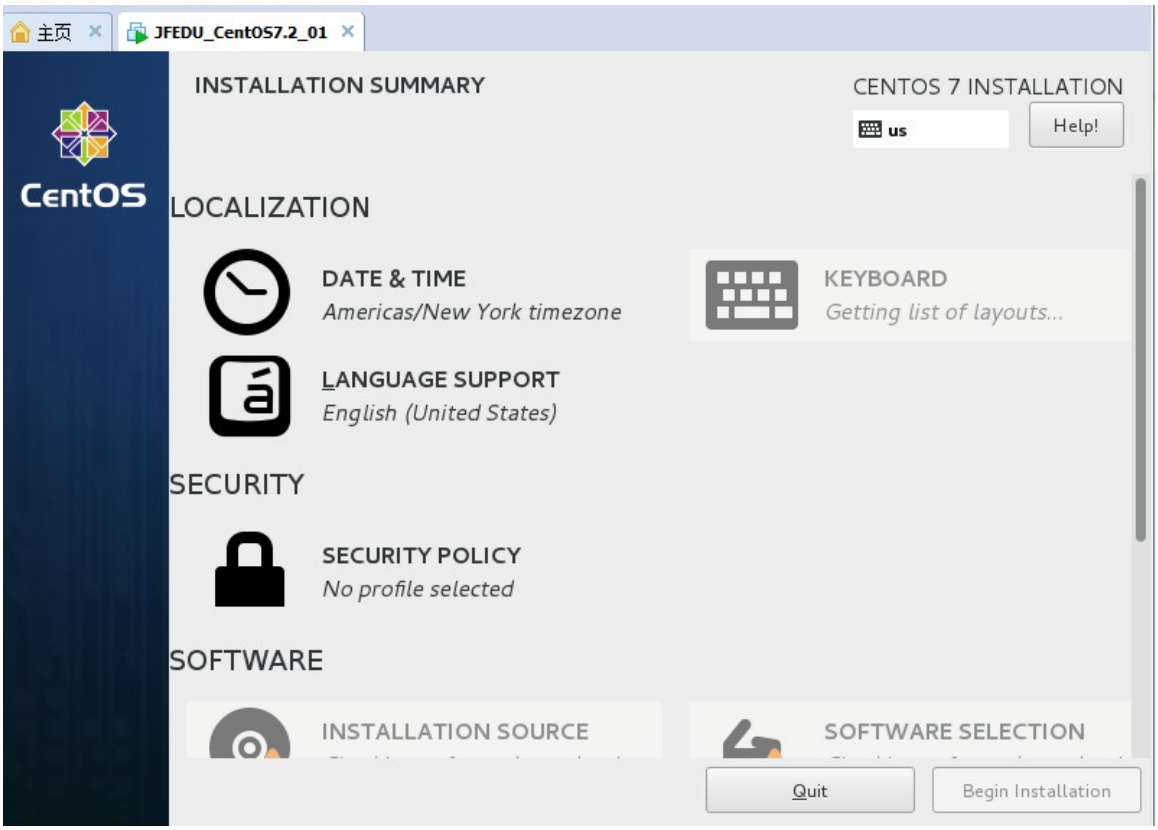
第二步，光盘检测，按 esc 跳过。



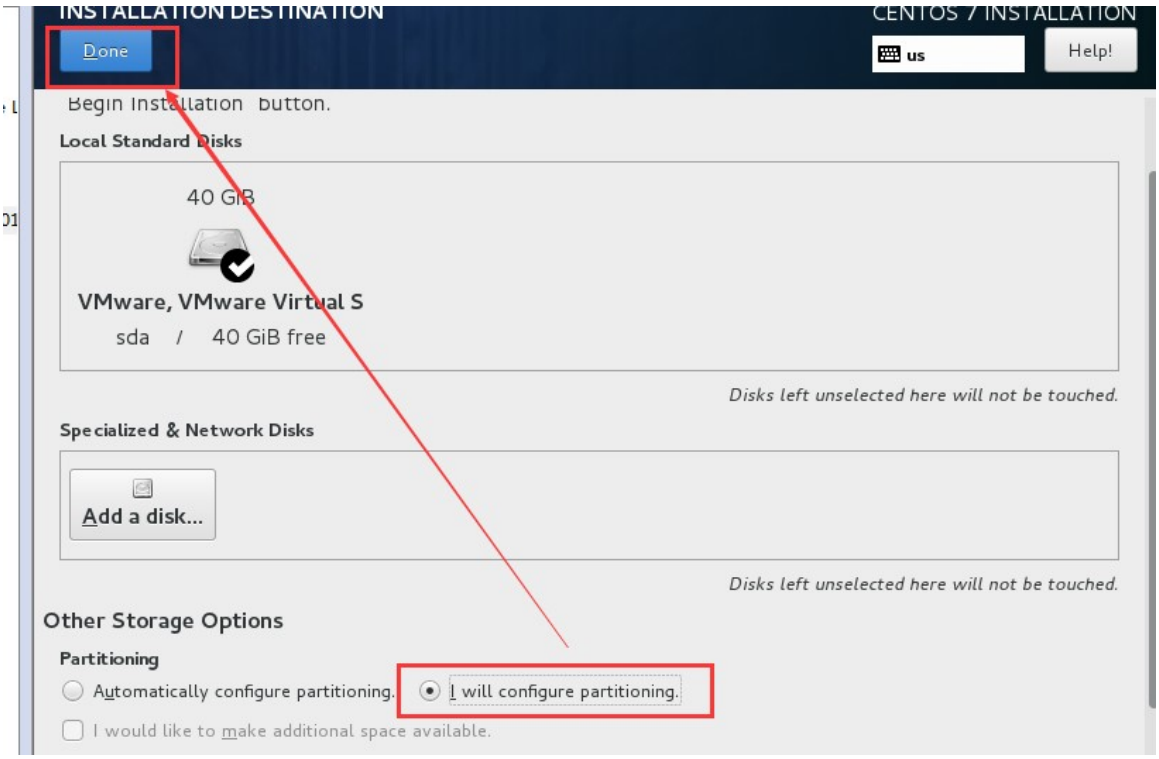
第三步，选择安装过程中的语言，初学者可以选择“简体中文”或者默认英文。



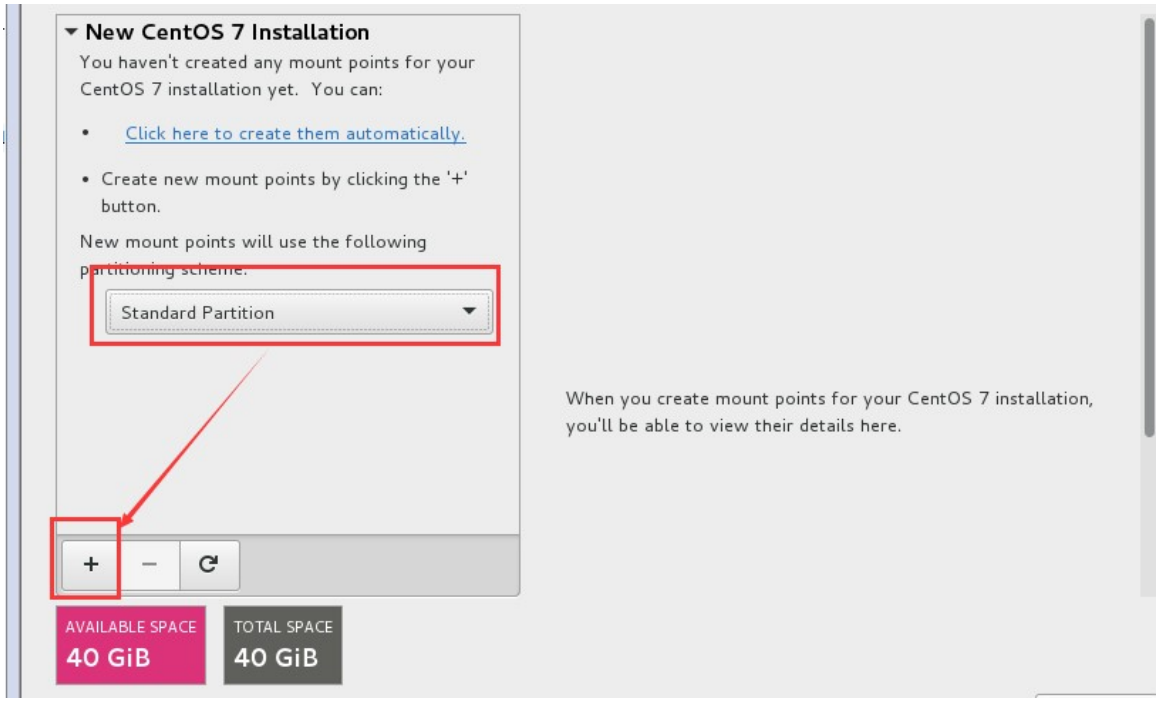
第四步，设置系统时间、磁盘分区、软件包信息



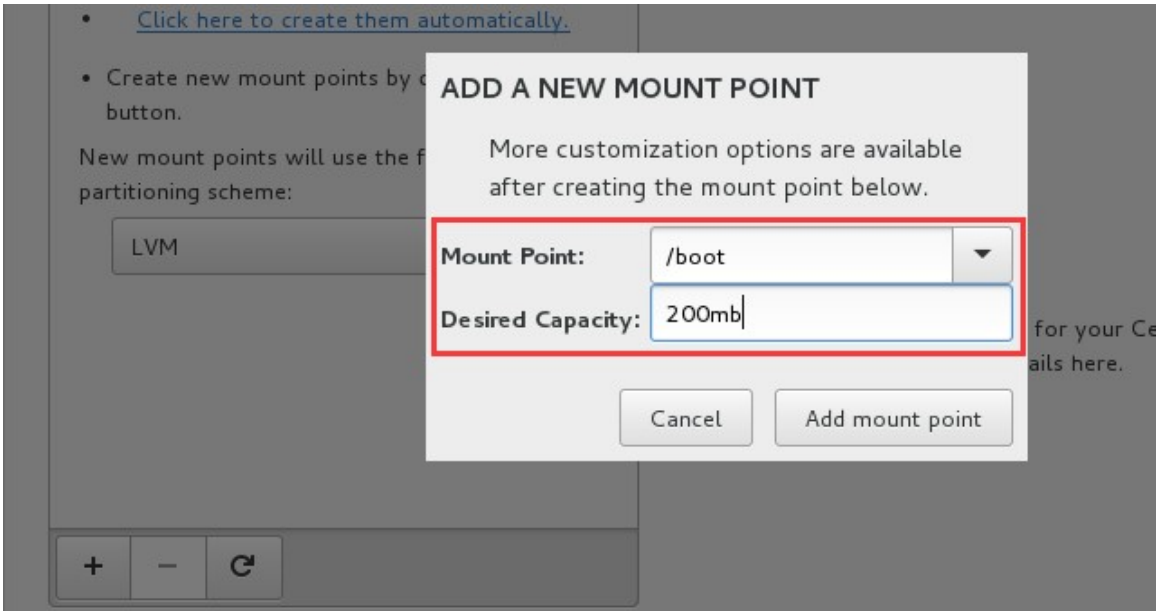
选择我将手动配置分区，不要写自动分区哦，然后点击 done



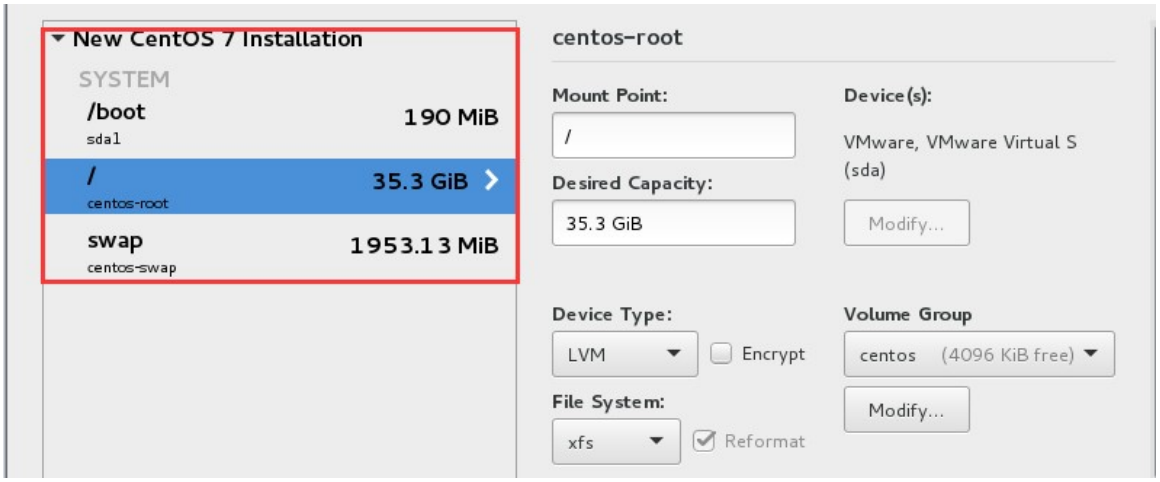
然后点击标准分区，加号+是新建分区



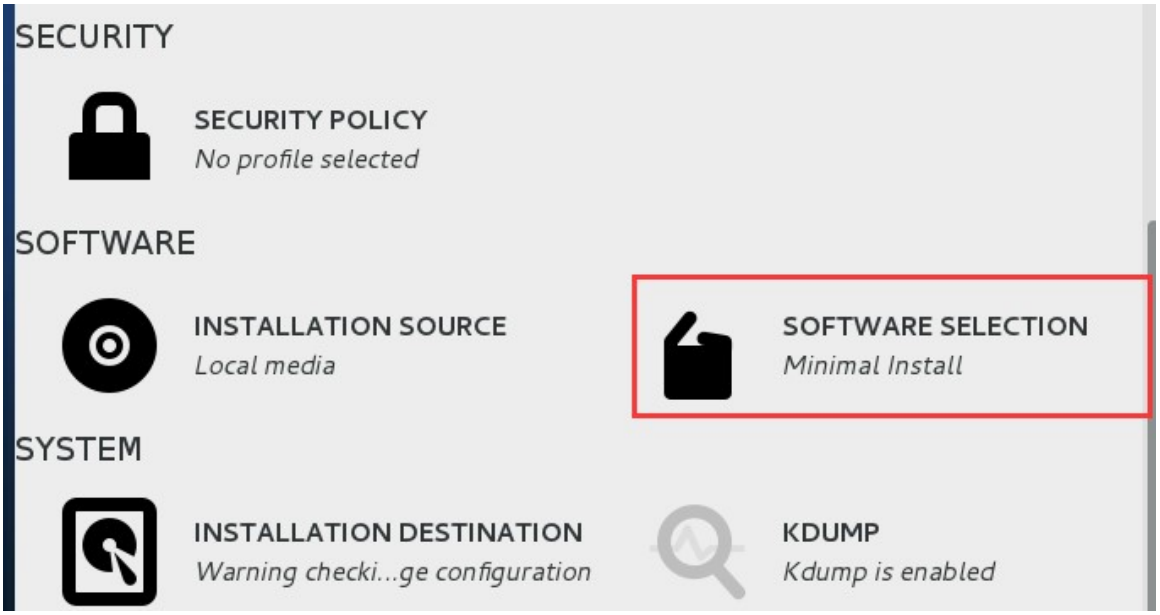
新建/boot 分区，/分区，swap 分区



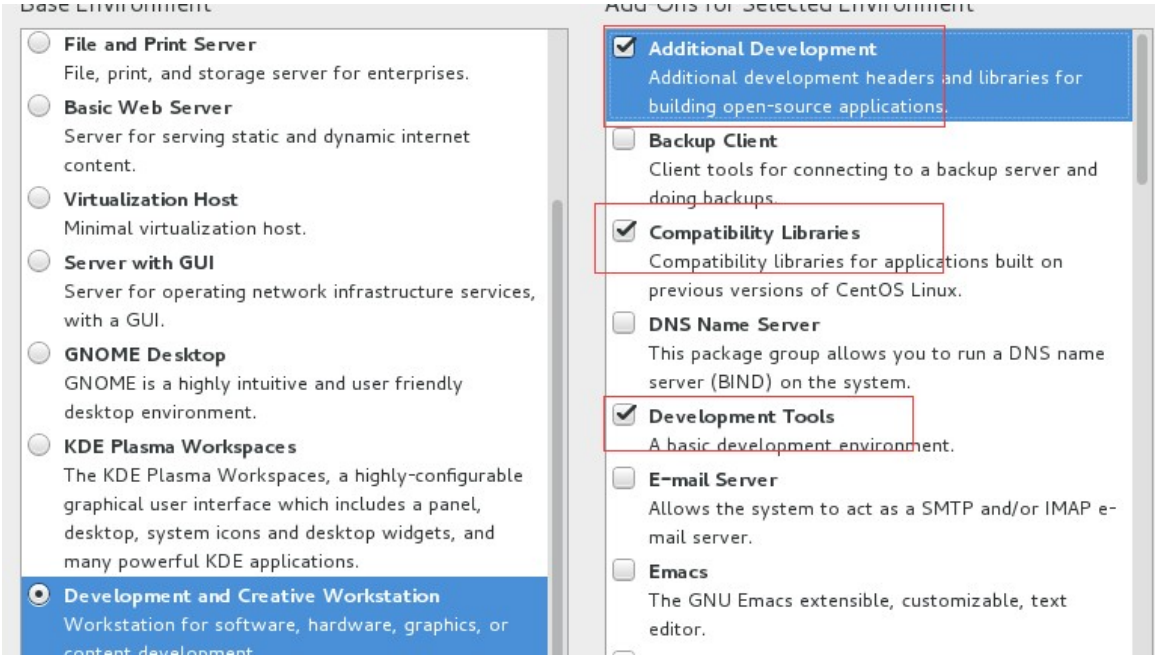
创建完，如下图所示，磁盘分区类型可以改成 EXT4 或者 xfs



第六步，现在需要安装的软件

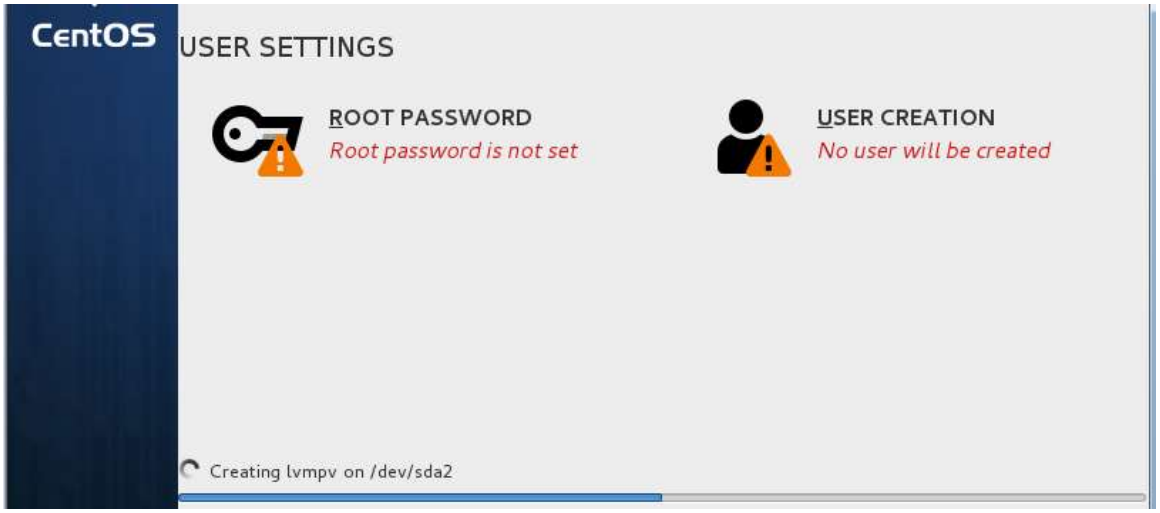


选择开发包，开发库

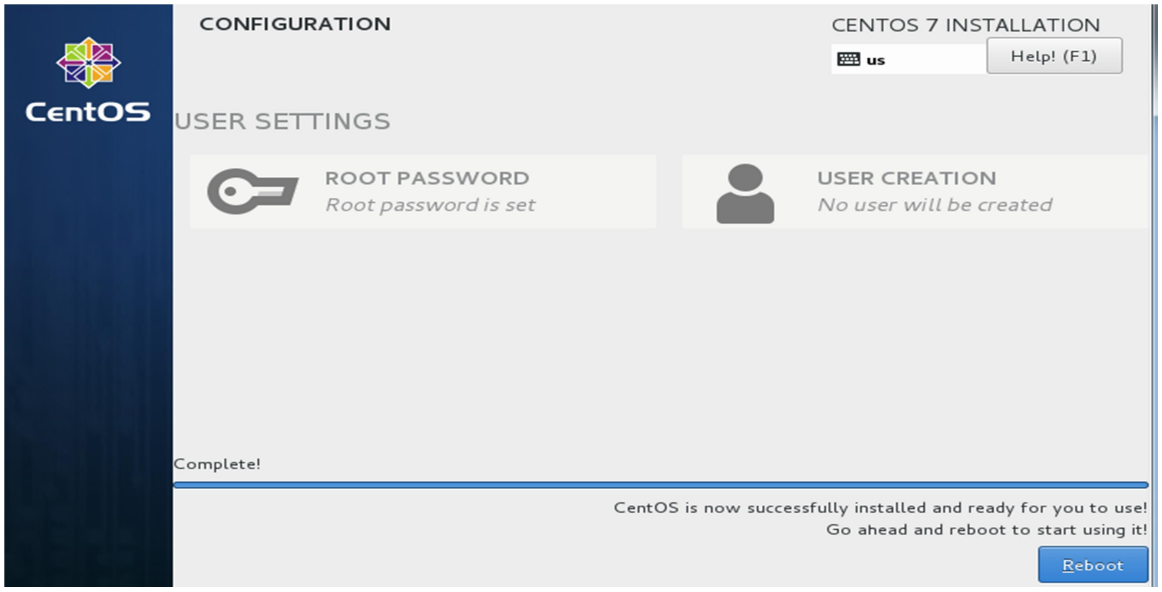


第七步，选择所在城市及选用 UTC 时间

第八步，如上配置完毕后，点击开始安装，然后设置根用户的密码（即 root 用户）



第九步，重启服务器



第十步，服务器窗口

```
CentOS Linux 7 (Core)
Kernel 3.10.0-327.el7.x86_64 on an x86_64

localhost login: root
Password:
Last login: Thu Aug 18 12:25:10 on tty1
[root@localhost ~]#
[root@localhost ~]# cat /etc/redhat-release
CentOS Linux release 7.2.1511 (Core)
[root@localhost ~]# uname -a
Linux localhost.localdomain 3.10.0-327.el7.x86_64 #1 SMP Thu Nov 19 22:10:57 UTC 2015 x86_64 x
[root@localhost ~]# pwd
/root
[root@localhost ~]# ls
anaconda-ks.cfg
```

1.10 菜鸟学好 Linux 大绝招

1) 初学者可以自己安装虚拟机，然后把 linux 常用命令例如 cd、ls、chmod、useradd、vi、cat、rm、mv、cp、find、more、vim、chown、awk、sed、grep、pwd 等等多练习几十遍，把自己敲打命令的熟练程度提升上来。

2) 最好有系统的参考资料和文档，根据文档搭建 Linux 下常见的各种服务（DHCP、SAMBA、DNS、Apache、Mysql 等），遇到问题后可以在 google 搜索，搜索的时候多看几篇文章，综合最好的文章来解决问题。

3) 要明白每个命令，每个服务的用途，为什么我要配置这个服务，只有带着目标去学习才能更快的成长，而不是看一点，学一点，没有系统和规划。

4) 能够熟练的搭建服务后，理解每个服务的完整配置和优化，可以拓展思维。例如 LAMP，我们一般是把所有服务放在一台机器上，如果分开多台该如何部署呢？等等。

5) 平时多积累 shell 编程，可以在网上查找前辈们写的非常好的 shell，自己下载下来多练习几遍，从中吸取，不断提高。

6) 建立一个自己的学习博客，把平时工作学习中的知识都记录在里面，这样也可以供别人来参考同时也能提高自己的编写文档及方案的能力。

7) 学习是一个长期的过程，一定要坚持，遇到各种错误、问题可以百度、Google，如果解决不了，可以跟老师请教，或者咨询同学和朋友。

8) 通过以上学习能够满足企业的一般应有，需要达到资深级别，还需要深入学习集群架构、负载均衡、自动化运维、运维开发等知识。

9) 最后还是一句话：多练习才是硬道理！实践出真知！

第2章　CentOS7 系统管理实战篇

通过前两章的学习，我们已经能够独立安装Linux系统，已经掌握了Linux学习的技巧，那接下来，我们将系统的来了解 Linux 系统各目录、权限及常用命令的使用。CentOS7 和 CentOS6 在系统管理、命令方面有什么区别呢？我们一起来遨游在 7 的世界里。

2.1　操作系统引导常用概念理解

1) BIOS 概念理解

BIOS 是英文"Basic Input Output System"的缩略语，中文名称是"基本输入输出系统"它是一组固化到计算机内主板上一个 ROM 芯片上的程序，它保存着计算机最重要的基本输入输出的程序、系统设置信息、开机后自检程序和系统自启动程序。 其主要功能是为计算机提供最底层的、最直接的硬件设置和控制。它是一段固件程序，主板上面 CMOS 芯片里写好的(CMOS 芯片是一种低耗电存储器，其主要作用是用来存放 BIOS 中的设置信息以及系统时间日期。）

2) MBR 概念理解

主引导记录（MBR，Main Boot Record）是位于磁盘最前边的一段引导（Loader）代码。它负责磁盘操作系统(OS)对磁盘进行读写时分区合法性的判别、分区引导信息的定位，它由磁盘操作系统(OS)在对硬盘进行初始化时产生的。

位于整个硬盘的 0 磁道 0 柱面 1 扇区。在总共 512 字节的主引导扇区中，MBR 只占用了其中的 446 个字节，另外的 64 个字节交给了 DPT(Disk Partition Table 硬盘分区表)，最后两个字节“55，AA”是分区的结束标志。这个整体构成了硬盘的主引导扇区。

0000-0088	Master Boot Record 主引导程序	主引导 程序
0089-01BD	出错信息数据区	数据区
01BE-01CD	分区项 1（16 字节）	分区表
01CE-01DD	分区项 2（16 字节）	
01DE-01ED	分区项 3（16 字节）	
01EE-01FD	分区项 4（16 字节）	
01FE	55	结束标志
01FF	AA	

主引导记录中包含了硬盘的一系列参数和一段引导程序（grub）。其中的硬盘引导程序的主要作用是检查分区表是否正确并且在系统硬件完成自检以后引导具有激活标志的分区上的操作系统，并将控制权交给启动程序。MBR 是由分区程序(如 Fdisk)所产生的，它不依赖任何操作系统，而且硬盘引导程序也是可以改变的，从而实现多系统共存。

MBR 是计算机最先执行的硬盘上的程序，只有 512 字节大小。因为只有 512 字节，不能载入操作系统的核心， 所以只能先载入一个可以载入计算机核心的程序，这个程序叫做引导程序。grub 就是一个引导程序。Grub 可以引导多种系统启动；

MBR 分区表只能识别磁盘前面的 2TB 左右的空间，对于后面的多余空间只能浪费掉了，如果你的硬盘容量 10TB 的磁盘，那就很浪费了。而且只支持 4 个分区；

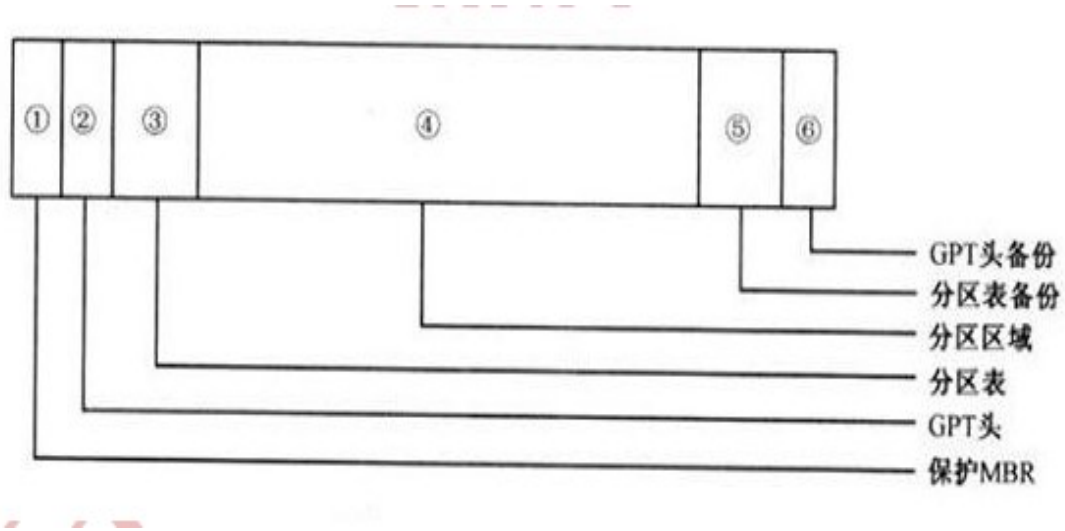
3) GPT 概念理解

GPT 名为 GUID 分区表。全局唯一的标识符（globally unique identifier，GUID）。这是一个正逐渐取代 MBR 的新标准。它和 UEFI 相辅相成——UEFI 用于取代老旧的 BIOS，

而 GPT 则取代老旧的 MBR。之所以叫作“GUID 分区表”，是因为你的驱动器上的每个分区都有一个全局唯一的标识符（globally unique identifier，GUID）。

GPT 磁盘的第一个扇区中同样有一个与 MBR（主引导记录）类似的标记，叫做 PMBR。PMBR 的作用是，当使用不支持 GPT 的分区工具时，整个硬盘将显示为一个受保护的分区，以防止分区表及硬盘数据遭到破坏。而其中存储的内容和 MBR 一样。GPT 优点：

支持超过 2T 的磁盘（64 位寻址空间）。fdisk 最大只能建立 2TB 大小的分区，创建一个大于 2TB 的分区使用 parted，向后兼容 MBR，必须在支持 uEFI 的硬件上才能使用（Intel 提出，用于取代 BIOS），必须使用 64 位系统，Mac、Linux 系统都能支持 GPT 分区格式，Windows 7/8 64bit、Windows Server 2008 64bit 支持 GPT



4) GRUB 概念理解

GNU GRUB（GRand Unified Bootloader 简称“GRUB”）是一个来自 GNU 项目的多操作系统启动程序。GRUB 是多启动规范的实现，它允许用户可以在计算机内同时拥有多个操作系统，并在计算机启动时选择希望运行的操作系统。GRUB 可用于选择操作系统分区上的不同内核，也可用于向这些内核传递启动参数。它是一个多重操作系统启动管理器。用来引导不同系统，如 windows，linux。

2.2 Linux 服务器启动流程详解

启动第一步 - - 加载 BIOS

当你打开计算机电源，计算机会首先加载 BIOS 信息，BIOS 信息是如此的重要，以至于计算机必须在最开始就找到它。这是因为 BIOS 中包含了 CPU 的相关信息、设备启动顺序信息、硬盘信息、内存信息、时钟信息、PnP 特性等等。在此之后，计算机心里就有谱了，知道应该去读取哪个硬件设备了。

启动第二步 - - 读取 MBR

众所周知，硬盘上第 0 磁道第一个扇区被称为 MBR，也就是 Master Boot Record，即主引导记录，它的大小是 512 字节，别看地方不大，可里面却存放了预启动信息、分区表信息。

系统找到 BIOS 所指定的硬盘的 MBR 后，就会将其复制到 0x7c00 地址所在的物理内存中。其实被复制到物理内存的内容就是 Boot Loader，而具体到你的电脑，那就是 lilo 或者 grub 了。

启动第三步 - - Boot Loader

Boot Loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核做好一切准备。

Boot Loader 有若干种，其中 Grub、Lilo 和 spfdisk 是常见的 Loader。

我们以 Grub 为例来讲解吧，毕竟用 lilo 和 spfdisk 的人并不多。

系统读取内存中的 grub 配置信息（一般为 menu.lst 或 grub.lst），并依照此配置信息来启动不同的操作系统。

启动第四步 - - 加载内核

根据 grub 设定的内核映像所在路径，系统读取内存映像，并进行解压缩操作。此时，屏幕一般会输出“Uncompressing Linux”的提示。当解压缩内核完成后，屏幕输出“OK, booting the kernel”。

系统将解压后的内核放置在内存之中，并调用 start_kernel()函数来启动一系列的初始化函数并初始化各种设备，完成 Linux 核心环境的建立。至此，Linux 内核已经建立起来了，基于 Linux 的程序应该可以正常运行了。

启动第五步 - - 用户层 init 依据 inittab 文件来设定运行等级

内核被加载后，第一个运行的程序便是/sbin/init，该文件会读取/etc/inittab 文件，并依据此文件来进行初始化工作。

其实/etc/inittab 文件最主要的作用就是设定 Linux 的运行等级，其设定形式是“：id:5:initdefault:”，这就表明 Linux 需要运行在等级 5 上。Linux 的运行等级设定如下：

0：关机

1：单用户模式

2：无网络支持的多用户模式

3：有网络支持的多用户模式

4：保留，未使用

5：有网络支持有 X-Window 支持的多用户模式

6：重新引导系统，即重启

启动第六步 - - init 进程执行 rc.sysinit

在设定了运行等级后，Linux 系统执行的第一个用户层文件就是/etc/rc.d/rc.sysinit 脚本程序，它做的工作非常多，包括设定 PATH、设定网络配置（/etc/sysconfig/network）、启动 swap 分区、设定/proc 等等。如果你有兴趣，可以到/etc/rc.d 中查看一下 rc.sysinit 文件。

启动第七步 - - 启动内核模块

具体是依据/etc/modules.conf 文件或/etc/modules.d 目录下的文件来装载内核模块。

启动第八步 - - 执行不同运行级别的脚本程序

根据运行级别的不同，系统会运行 rc0.d 到 rc6.d 中的相应的脚本程序，来完成相应的初始化工作和启动相应的服务。

启动第九步 - - 执行/etc/rc.d/rc.local

你如果打开了此文件，里面有一句话，读过之后，你就会对此命令的作用一目了然：

This script will be executed *after* all the other init scripts.

You can put your own initialization stuff in here if you don’ t

want to do the full Sys V style init stuff.

touch test.txt

reboot

rc.local 就是在一切初始化工作后，Linux 留给用户进行个性化的地方。你可以把你想设置和启动的东西放到这里。

启动第十步 - - 执行/bin/login 程序，进入登录状态

2.3 CentOS6.x 与 CentOS7.x 区别

Linux 操作系统的启动首先从 BIOS 开始，接下来进入 boot loader，由 bootloader 载入内核，进行内核初始化。内核初始化的最后一步就是启动 pid 为 1 的 init 进程。这个进程是系统的第一个进程。它负责产生其他所有用户进程。init 以守护进程方式存在，是所有其他进程的祖先。init 进程非常独特，能够完成其他进程无法完成的任务。

Sysvinit 就是 system V 风格的 init 系统,Sysvinit 用术语 runlevel 来定义"预订的运行模式"。Sysvinit 检查 '/etc/inittab' 文件中是否含有 'initdefault' 项。 这告诉 init 系统是否有一个默认运行模式。**Sysvinit 使用脚本，文件命名规则和软链接来实现不同的 runlevel,串行启动各个进程及服务。**

Systemd 是 Linux 系统中最新的初始化系统（init），它主要的设计目标是克服 sysvinit 固有的缺点，提高系统的启动速度。systemd 和 ubuntu 的 upstart 是竞争对手，预计会取代 UpStart。Systemd 的目标就是尽可能启动更少的进程，尽可能将更多进程并行启动。

A	B	C	D
编号	系统功能	CentOS6	CentOS7
1	init系统	sysvinit	systemd
2	桌面系统	GNOME 2.x	GNOME 3.x/GNOME Shell
3	文件系统	ext4	xfs
4	内核版本	2.6.x	3.10.x
5	启动加载器	GRUB Legacy (+efibootmgr)	GRUB2
6	防火墙	iptables	firewalld
7	数据库	mysql	MariaDB
8	文件目录	/bin, /sbin, /lib, and /lib64在/根下	/bin, /sbin, /lib, and /lib64在/usr下
9	主机名	/etc/sysconfig/network	/etc/hostname
10	时间同步	ntp, ntpq -p	chrony,chronyc sources

11	修改时间	#vi /etc/sysconfig/clock
		ZONE="Asia/Tokyo"
		UTC=fales
		#ln -s
		/usr/share/zoneinfo/Asia/Tokyo
		/etc/localtime
			#timedatectl set-timezone Asia/Tokyo
			#timedatectl status

12	区域及字符设置	/etc/sysconfig/i18n	/etc/locale.conf localectl set-locale LANG=zh_CN.utf8 localectl status
13	启动停止服务	#service service_name start #service service_name stop #service sshd restart/status/reload	#systemctl start service_name #systemctl stop service_name #systemctl restart/status/reload sshd
14	自动启动	chkconfig service_name on/off	#systemctl enable service_name #systemctl disable service_name
15	服务列表	chkconfig --list	#systemctl list-unit-files #systemctl --type service
16	Kill服务	kill -9 <PID>	systemctl kill --signal=9 sshd
17	网络及端口信息	netstat	ss
18	IP信息	ifconfig	ip address show
19	路由信息	route -n	ip route show
20	关闭停止系统	shutdown -h now	systemctl poweroff
21	单用户模式	init S	systemctl rescue
22	运行模式	vim /etc/inittab id:3:initdefault:	systemctl set-default graphical.target systemctl set-default multi-user.target

文件系统的区别,Centos6.x 普遍采用 ext3\ext4 文件系统格式，而 centos7 默认采用 xfs 格式：

EXT3 支持的最大 16TB 文件系统和最大 2TB 文件，Ext4 分别支持 1EB（1,048,576TB， 1EB=1024PB， 1PB=1024TB）的文件系统，以及 16TB 的文件。Ext3 目前只支持 32,000 个子目录，而 Ext4 支持无限数量的子目录。

EXT4 是第四代扩展文件系统 (英语：Fourth EXtended filesystem，缩写为 ext4) 是 Linux 系统下的日志文件系统，是 ext3 文件系统的后继版本。

Ext4 的文件系统容量达到 1EB，而单个文件容量则达到 16TB，这是一个非常大的数字了。对一般的台式机和服务器而言，这可能并不重要，但对于大型磁盘阵列的用户而言，这就非常重要；（磁盘结构原因 Ext4 的 inode 个数限制(32 位数)最多只能有大概 40 多亿文件，而且 Ext4 的单个文件大小最大只能支持到 16T(4K block size)）

XFS 是一个 64 位文件系统，最大支持 8EB 减 1 字节的单个文件系统，实际部署时取决于宿主操作系统的最大块限制。对于一个 32 位 Linux 系统，文件和文件系统的大小会被限制在 16TB。

2.4 Linux 网络配置管理

熟悉了常用的命令和 Linux 权限，那接下来如何让所在的 Linux 系统上网呢？管理 linux 服务器网络有哪些命令呢?

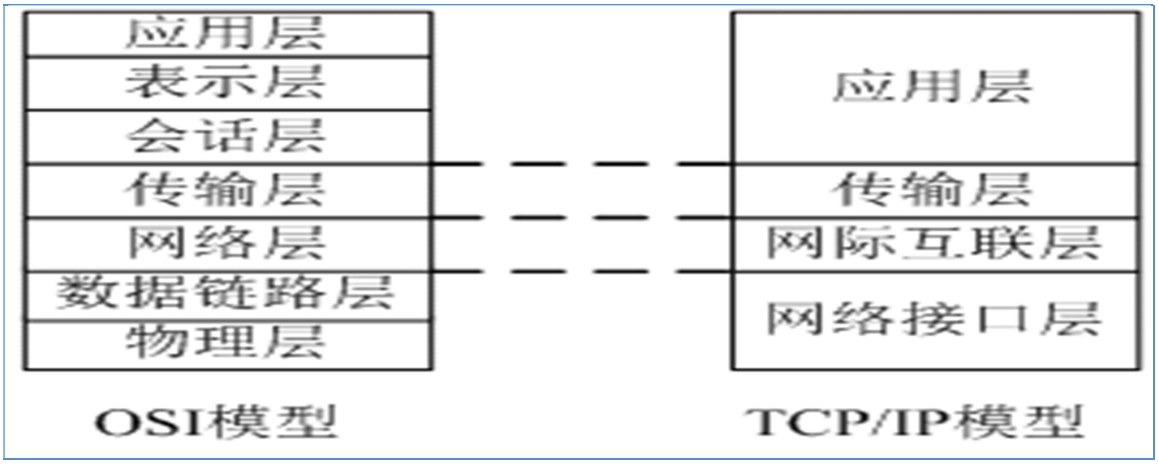
Linux 服务器默认网卡配置文件在/etc/sysconfig/network-scripts/下，命名的名称一般为:ifcfg-eth0 ifcfg-eth1 ，eth0 表示第一块网卡，eth1 表示第二块网卡，依次类推。一般 DELL R720 标配有 4 块千兆网卡。

2.5 TCP/IP 协议概述

TCP/IP（Transmission Control Protocol/Internet Protocol 的简写），中译名为传输控制协议/因特网互联协议，又名网络通讯协议，是 Internet 最基本的协议、Internet 国际互联网络的基础，由网络层的 IP 协议和传输层的 TCP 协议组成。**TCP/IP 定义了电子设备如何连入因特网，以及数据如何在它们之间传输的标准。协议采用了 4 层的层级结构，每一层都呼叫它的下一层所提供的协议来完成自己的需求。**

通俗而言，TCP 负责发现传输的问题，一有问题就发出信号，要求重新传输，直到所

有数据安全正确地传输到目的地。而 IP 是给因特网的每一台联网设备规定一个地址。基于 TCP/IP 的参考模型将协议分成四个层次，它们分别是网络接口层、网际互连层（IP 层）、传输层（TCP 层）和应用层。如下为 TCP/IP 跟 OSI 参考模型的对比图



OSI 模型与 TCP/IP 模型协议对照表

OSI 7层	每层功能	TCP/IP协议簇
应用层	文件传输，电子邮件，文件服务，虚拟终端	TFTP、HTTP、SNMP、FTP、SMTP、DNS、Telnet
表示层	数据格式化，代码转换，数据加密	没有协议
会话层	解除或建立与别的接点的联系	没有协议
传输层	提供端对端的接口	TCP、UDP
网络层	为数据包选择路由	IP、ICMP、OSPF、BGP、IGMP 、ARP、RARP
数据链路层	传输有地址的帧以及错误检测功能	SLIP、PPP、MTU
物理层	以二进制数据形式在物理媒体上传输数据	ISO2110、IEEE802、IEEE802.2

2.6 IP 地址及网络常识

1) IP 地址

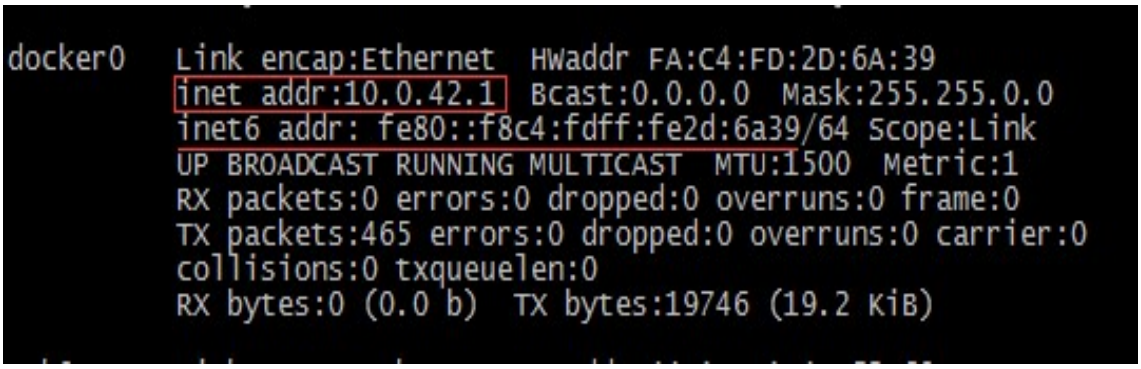
IP 地址是指互联网协议地址（Internet Protocol Address，又译为网际协议地址），是 IP Address 的缩写。IP 地址是 IP 协议提供的一种统一的地址格式，它为互联网上的每一个网络和每一台主机分配一个逻辑地址，以此来屏蔽物理地址的差异。IP 地址被用来给 Internet 上的电脑一个编号。大家日常见到的情况是每台联网的 PC 上都需要有 IP 地址，才能正常通信。

IP 地址是一个 32 位的二进制数，通常被分割为 4 个“8 位二进制数”（也就是 4 个字节）。IP 地址通常用“点分十进制”表示成（a.b.c.d）的形式，其中，a,b,c,d 都是 0~255 之间的十进制整数。

常见的 IP 地址，分为 IPv4 与 IPv6 两大类。IP 地址编址方案：IP 地址编址方案将 IP 地址空间划分为 A、B、C、D、E 五类，其中 A、B、C 是基本类，D、E 类作为多播和保留使用。

IPV4 就是有 4 段数字，每一段最大不超过 255。由于互联网的蓬勃发展，IP 位址的需求量愈来愈大，使得 IP 位址的发放愈趋严格，各项资料显示全球 IPv4 位址可能在 2005 至 2010 年间全部发完(实际情况是在 2011 年 2 月 3 日 IPv4 位地址分配完毕）。

地址空间的不足必将妨碍互联网的进一步发展。为了扩大地址空间，拟通过 IPv6 重新定义地址空间。IPv6 采用 128 位地址长度。在 IPv6 的设计过程中除了一劳永逸地解决了地址短缺问题以外，还考虑了在 IPv4 中解决不好的其它问题。



A 类 IP 地址

一个 A 类 IP 地址是指， 在 IP 地址的四段号码中，第一段号码为网络号码，剩下的三段号码为本地计算机的号码。如果用二进制表示 IP 地址的话，A 类 IP 地址就由 1 字节的网络地址和 3 字节主机地址组成，网络地址的最高位必须是“0”。A 类 IP 地址中网络的标识长度为 8 位，主机标识的长度为 24 位，A 类网络地址数量较少，有 126 个网络，每个网络可以容纳主机数达 1600 多万台。

A 类 IP 地址 地址范围 1.0.0.0 到 127.255.255.255（二进制表示为:00000001 00000000 00000000 00000000 - 01111110 11111111 11111111 11111111）。最后一个是广播地址。A 类 IP 地址的子网掩码为 255.0.0.0，每个网络支持的最大主机数为 256 的 3 次方-2=16777214 台。

B 类 IP 地址

一个 B 类 IP 地址是指，在 IP 地址的四段号码中，前两段号码为网络号码。如果用二进制表示 IP 地址的话，B 类 IP 地址就由 2 字节的网络地址和 2 字节主机地址组成，网络地址的最高位必须是“10”。B 类 IP 地址中网络的标识长度为 16 位，主机标识的长度为 16 位，B 类网络地址适用于中等规模的网络，有 16384 个网络，每个网络所能容纳的计算机数为 6 万多台。B 类 IP 地址地址范围 128.0.0.0-191.255.255.255 （二进制表示为：10000000 00000000 00000000 00000000----10111111 11111111 11111111 11111111）。 最后一个是广播地址。B 类 IP 地址的子网掩码为 255.255.0.0，每个网络支持的最大主机数为

256 的 2 次方-2=65534 台。

C 类 IP 地址

一个 C 类 IP 地址是指，在 IP 地址的四段号码中，前三段号码为网络号码，剩下的一段号码为本地计算机的号码。如果用二进制表示 IP 地址的话，C 类 IP 地址就由 3 字节的网络地址和 1 字节主机地址组成，网络地址的最高位必须是“110”。C 类 IP 地址中网络的标识长度为 24 位，主机标识的长度为 8 位，C 类网络地址数量较多，有 209 万余个网络。适用于小规模的局域网络，每个网络最多只能包含 254 台计算机。

C 类 IP 地址范围 192.0.0.0-223.255.255.255[3]　（二进制表示为: 11000000 00000000 00000000 00000000 - 11011111 11111111 11111111 11111111）。

C 类 IP 地址的子网掩码为 255.255.255.0，每个网络支持的最大主机数为 256-2=254 台

D 类 IP 地址

D 类 IP 地址在历史上被叫做多播地址(multicast address)，即组播地址。在以太网中，多播地址命名了一组应该在这个网络中应用接收到一个分组的站点。多播地址的最高位必须是“1110”，范围从 224.0.0.0 到 239.255.255.255。

特殊的网址

每一个字节都为 0 的地址（“0.0.0.0”）对应于当前主机，IP 地址中的每一个字节都为 1 的 IP 地址（“255．255．255．255”）是当前子网的广播地址；

IP 地址中凡是以“11110”开头的 E 类 IP 地址都保留用于将来和实验使用。

IP 地址中不能以十进制“127”作为开头，该类地址中数字 127．0．0．1 到 127．255．255．255 用于回路测试，如：127.0.0.1 可以代表本机 IP 地址，用“http://127.0.0.1”就可以测试本机中配置的 Web 服务器。网络 ID 的第一个 8 位组也不能全置为“0”，全“0”表示本地网络。

2) 子网掩码

子网掩码(subnet mask)又叫网络掩码、地址掩码、子网络遮罩，它是一种用来指明一个 IP 地址的哪些位标识的是主机所在的子网，以及哪些位标识的是主机的位掩码。子网掩码不能单独存在，它必须结合 IP 地址一起使用。子网掩码只有一个作用，就是将某个 IP 地址划分成网络地址和主机地址两部分。

子网掩码是一个 32 位地址，用于屏蔽 IP 地址的一部分以区别网络标识和主机标识，并说明该 IP 地址是在局域网上，还是在远程网上。

子网掩码——屏蔽一个 IP 地址的网络部分的“全 1”比特模式。对于 A 类地址来说，默认的子网掩码是 255.0.0.0；对于 B 类地址来说默认的子网掩码是 255.255.0.0；对于 C 类地址来说默认的子网掩码是 255.255.255.0。

互联网是由各种小型网络构成的，每个网络上都有许多主机，这样便构成了一个有层次的结构。IP 地址在设计时就考虑到地址分配的层次特点，将每个 IP 地址都分割成网络号和主机号两部分，以便于 IP 地址的寻址操作。

IP 地址的网络号和主机号各是多少位呢？如果不指定，就不知道哪些位是网络号、哪些是主机号，这就需要通过子网掩码来实现。

3) 网关地址

网关（Gateway）就是一个网络连接到另一个网络的“关口”， 网关实质上是一个网络通向其他网络的 IP 地址。主要用于不同网络传输数据，例如我们一般上网，需要在本机配置网关，内网服务器的数据通过网关，网关把数据转发到其他的网络的网关，直至找到对方的主机网络，然后返回数据。

4) MAC 地址

MAC（Media Access Control 或者 Medium Access Control）地址，意译为媒体访

问控制，或称为物理地址、硬件地址，用来定义网络设备的位置。在 OSI 模型中，第三层网络层负责 IP 地址，第二层数据链路层则负责 MAC 地址。因此一个主机会有一个 MAC 地址，而每个网络位置会有一个专属于它的 IP 地址。

IP 地址工作在 OSI 参考模型的第三层网络层。两者之间分工明确，默契合作，完成通信过程。IP 地址专注于网络层，将数据包从一个网络转发到另外一个网络；而 MAC 地址专注于数据链路层，将一个数据帧从一个节点传送到相同链路的另一个节点。

（OSI 模型是 TCP/IP 通信的标准，该模型定义了不同计算机互联的标准，是设计和描述计算机网络通信的基本框架。OSI 模型把网络通信的工作分为 7 层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。）

在一个稳定的网络中，IP 地址和 MAC 地址是成对出现的。如果一台计算机要和网络中另一外计算机通信，那么要配置这两台计算机的 IP 地址，MAC 地址是网卡出厂时设定的，这样配置的 IP 地址就和 MAC 地址形成了一种对应关系。

在数据通信时，IP 地址负责表示计算机的网络层地址，网络层设备（如路由器）根据 IP 地址来进行操作；MAC 地址负责表示计算机的数据链路层地址，数据链路层设备（如交换机）根据 MAC 地址来进行操作。IP 和 MAC 地址这种映射关系由 ARP（Address Resolution Protocol，地址解析协议）协议完成。

2.7 Linux 服务器 IP 命名规范

修改网卡的 IP，可以使用命令: vi /etc/sysconfig/network-scripts/ifcfg-eth0（**ifcfg-eno16777736**）如果是 DHCP 获取的 IP，默认配置如下：

 # Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE]

DEVICE=eth0

BOOTPROTO=dhcp

HWADDR=00:0c:29:52:c7:4e

ONBOOT=yes

TYPE=Ethernet

如果是静态配置的 IP，ifcfg-eth0 网卡配置内容如下：

 # Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE]

DEVICE=eth0

BOOTPROTO=static

HWADDR=00:0c:29:52:c7:4e

ONBOOT=yes

TYPE=Ethernet

IPADDR=192.168.1.103

NETMASK=255.255.255.0

GATEWAY=192.168.33.1

网卡参数详解如下：

DEVICE=eth0　#物理设备名

ONBOOT=yes　# [yes|no]（重启网卡是否激活设备）

BOOTPROTO=static #[none|static|bootp|dhcp]（不使用协议|静态分配|BOOTP 协议|DHCP 协议）

TYPE=Ethernet　#网卡类型

IPADDR=192.168.1.103 #IP 地址

NETMASK=255.255.255.0 #子网掩码

GATEWAY=192.168.33.1 #网关地址

网卡配置完毕，重启网卡，命令: /etc/init.d/network restart 即可。

查看 ip 命令：ifconfig 查看当前服务器所有网卡的 IP，可以单独指定，ifconfig eth0 查看 eth0 的 IP 地址。

如果没有 ifconfig 命令，可以用 ip addr list，也可以安装 ifconfig 命令软件包如下图：

yum install net-tools -y

```
[root@localhost ~]# ifconfig
-bash: ifconfig: command not found
[root@localhost ~]# yum install net-tools -y
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirrors.btte.net
 * extras: mirror.bit.edu.cn
 * updates: mirrors.tuna.tsinghua.edu.cn
Resolving Dependencies
--> Running transaction check
---> Package net-tools.x86_64 0:2.0-0.17.20131004git.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

2.8 Linux 服务器网卡及主机名命名

Centos7 服务器，默认网卡为 ifcfg-enxxxx，如果我们想改成 eth0，可以使用如下步骤：

1) 修改文件：vim /etc/sysconfig/grub

在倒数第二行 quiet 后面加入：net.ifnames=0 biosdevname=0

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto rhgb quiet net.ifnames=0 biosdevname=0"
GRUB_DISABLE_RECOVERY="true"
~
~
~
```

2) 然后执行命令如下图

grub2-mkconfig -o /boot/grub2/grub.cfg

```
[root@localhost ~]#
[root@localhost ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0-327.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-327.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-6fa6fd8dd8874932be8afaae5b91800f
Found initrd image: /boot/initramfs-0-rescue-6fa6fd8dd8874932be8afaae5b91800f
done
[root@localhost ~]#
```

3) 重命名网卡名称

执行命令 mv ifcfg-eno16777736 ifcfg-eth0，然后修改 eth0 文件里面设备也为 eth0 才行

```
[root@localhost network-scripts]# ls
ifcfg-eth0   ifdown-isdn      ifup          ifup-plip      ifup-tunn
ifcfg-lo     ifdown-post      ifup-aliases  ifup-plusb     ifup-wire
ifdown       ifdown-ppp       ifup-bnep     ifup-post      init.ipv6
ifdown-bnep  ifdown-routes    ifup-eth      ifup-ppp       network-f
ifdown-eth   ifdown-sit       ifup-ib       ifup-routes    network-f
ifdown-ib    ifdown-Team      ifup-ippp     ifup-sit
ifdown-ippp  ifdown-TeamPort  ifup-ipv6     ifup-Team
ifdown-ipv6  ifdown-tunnel    ifup-isdn     ifup-TeamPort
[root@localhost network-scripts]# mv ifcfg-eno16777736 ifcfg-eth0
```

4) 重启服务器验证

reboot 重启完后，如下图：

```
192.168.1.103 ×
Last login: Sat Aug 20 13:41:10 2016 from 192.168.1.101
[root@localhost ~]#
[root@localhost ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 15
        inet 192.168.1.103  netmask 255.255.255.0  broad
        inet6 fe80::20c:29ff:fe1c:9220  prefixlen 64  sc
        ether 00:0c:29:1c:92:20  txqueuelen 1000  (Ether
        RX packets 39  bytes 5322 (5.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 47  bytes 8499 (8.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  co

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
```

2.9 Linux 服务器上网 DNS 设置

网卡配置完毕，如果来配置 DNS，首先要知道 DNS 配置在哪个目录文件下，

vi /etc/resolv.conf 文件:

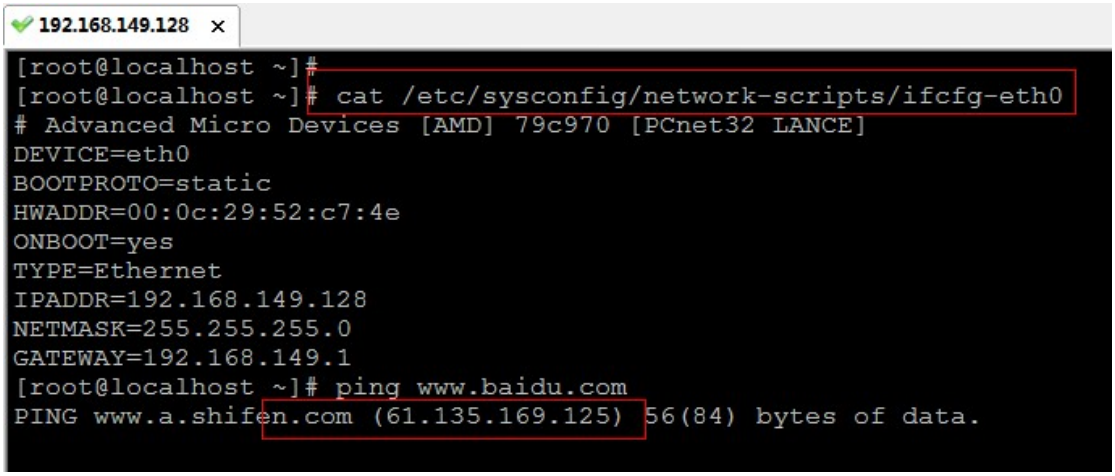
在该文件里面添加如下两条：

nameserver 202.106.0.20

nameserver 8.8.8.8

从上到下，分别表示主 DNS，备 DNS。配置完毕后，不需要重启网卡,DNS 立即生效。

可以 ping www.baidu.com 看看效果:

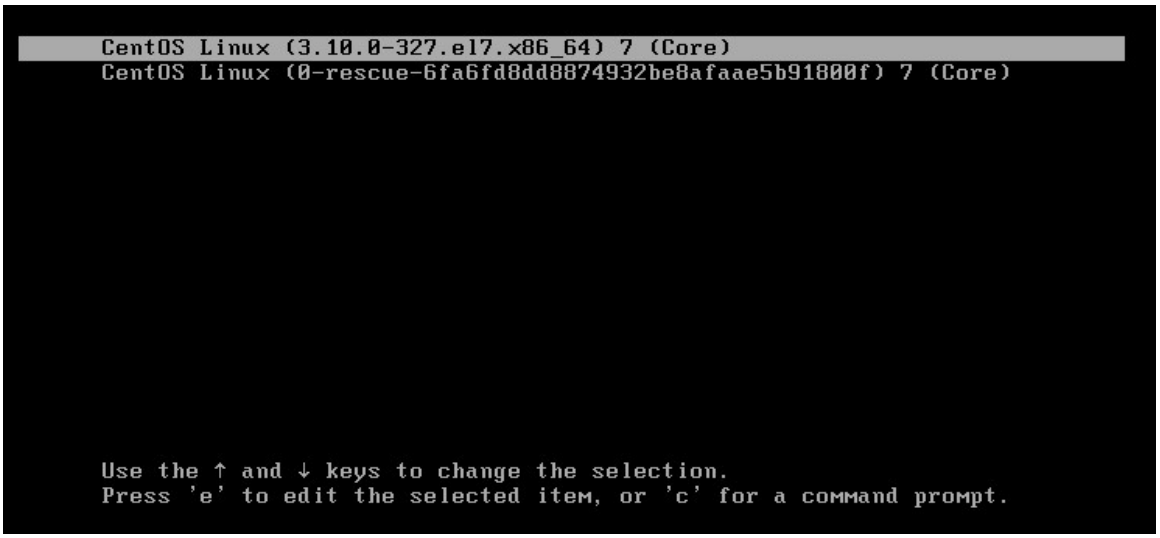


IP 配置完毕。

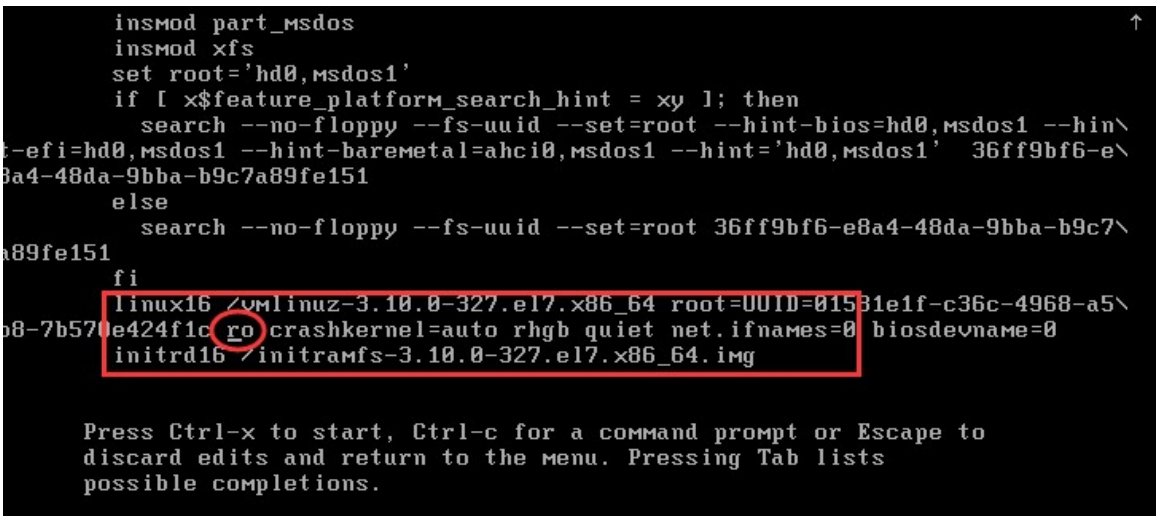
2.10 Centos7.x ROOT 密码重置方法

大家之前学习 centos6 的 ROOT 密码重置，然后用来重置 centos7 发现行不通，那么 centos7 怎么重置密码呢，为什么要重置密码呢?让我们一步一步来操作：

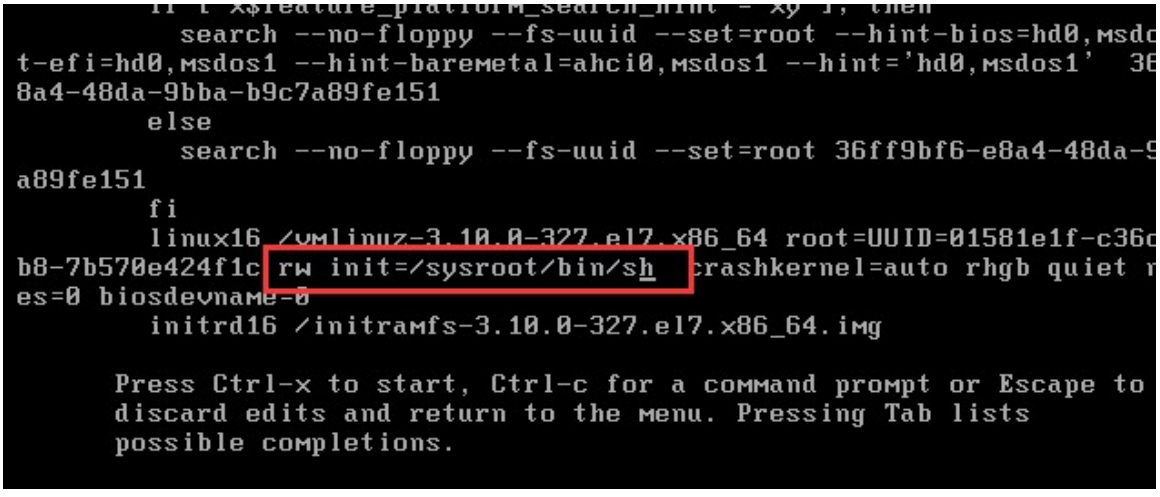
Reboot 重启系统，进入修改密码步骤



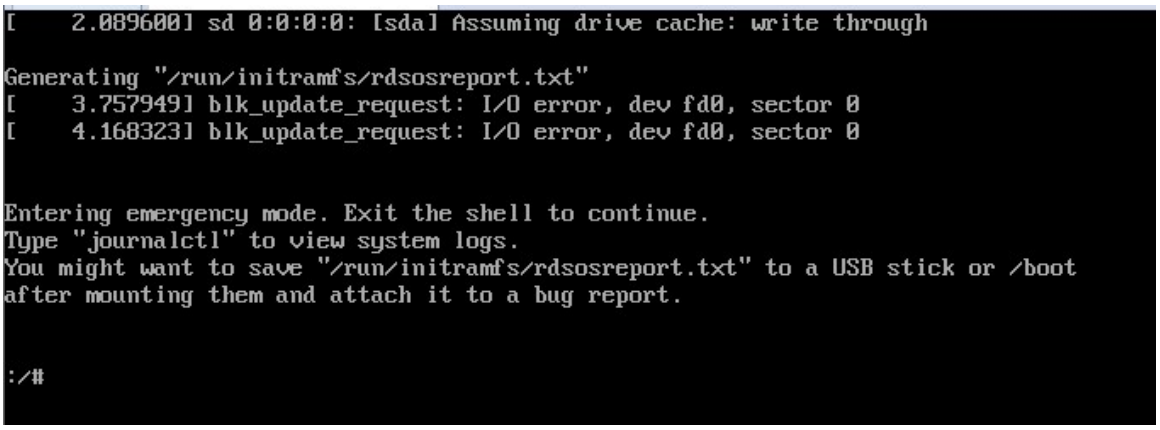
然后按 e 进行编辑，找到 ro　crashkernel=xxx，将 ro 改成 rw init=/sysroot/bin/sh



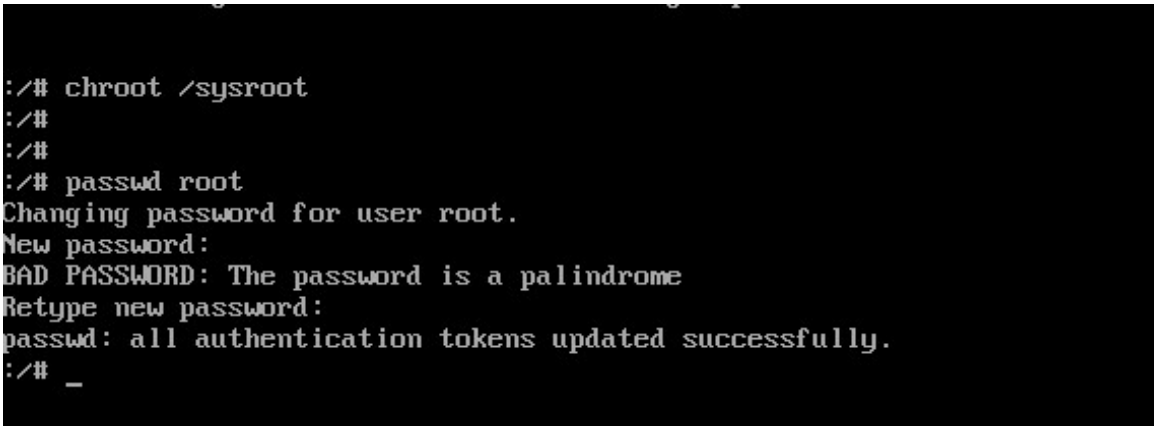
改成如下图：



然后按 ctrl+x 按钮进入单用户模式



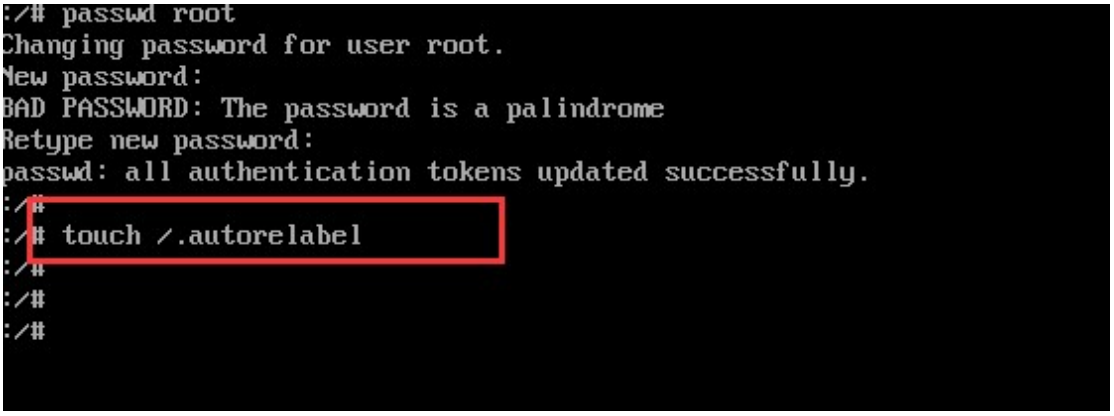
使用命令 chroot /sysroot 访问系统并 passwd 修改 root 密码即可。



然后更新系统信息：

touch /.autorelabel

(执行命令 touch /.autorelabel ,在/下创建一个.autorelabel 文件，有这个文件存在，系统在重启时就会对整个文件系统进行 relabeling 重新标记，也可以理解为对文件进行底层权限的控制和标记)，如果 selinux 属于关闭状态则不需要执行这条命令。

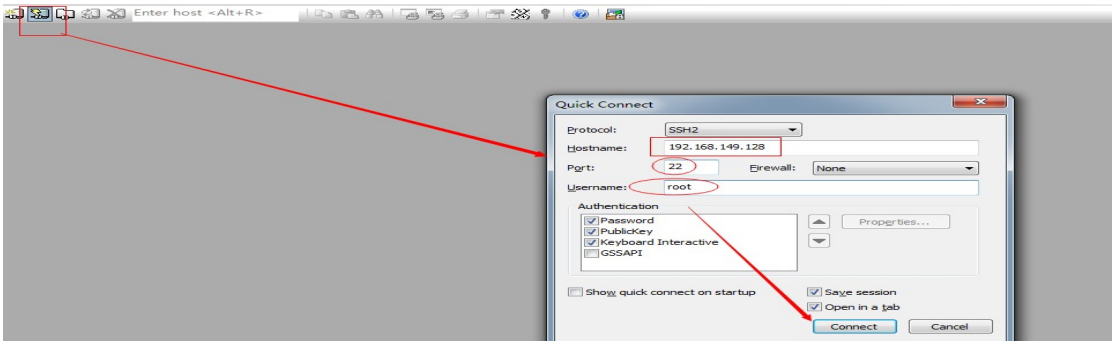


2.11 企业实战管理 Linux 服务器

通过远程工具来连接 Linux 服务器，常见的 Linux 远程连接工具有:putty、SecureCRT（主流）、xshell、xmanger 等工具。

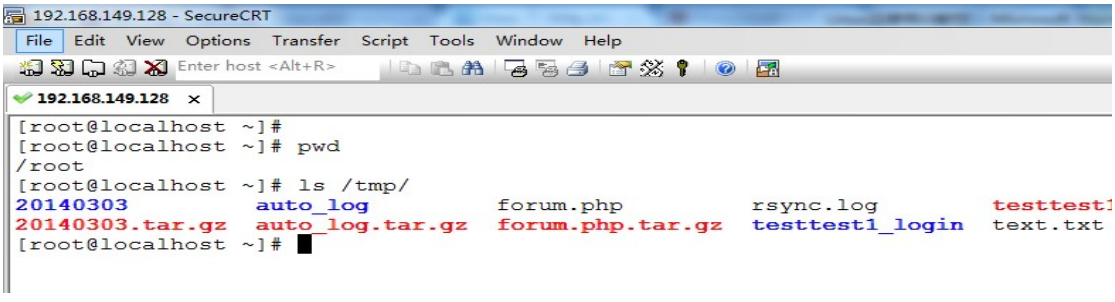
下载安装 secureCRT，打开工具，然后如图配置：

点击左上角 quick connect 快速连接，弹出界面，然后输入 IP，用户名，端口默认是 22，然后点击下方的 connect 连接，会提示输入密码，输入即可。



弹出输入密码框：

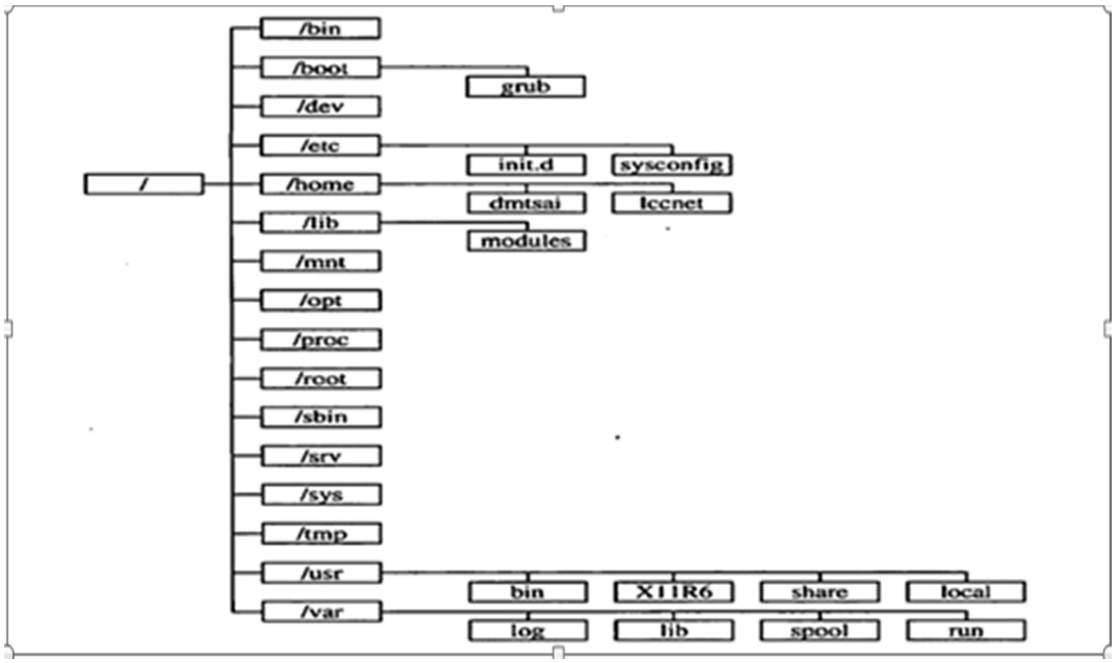
进入远程界面，与服务器真实登录一样，然后可以执行命令：



2.12 Linux 操作系统目录一览表

通过前面的学习,我们已经能够独立安装完一个 linux 系统，那接下来我们来熟悉一下 Linux 系统里面的各个目录文件夹的大致功能：

主要的目录树的有/、/root、/home、/usr、/bin 等目录。下面是一个典型的 linux 目录结构如下： （附图表）



2.13 Linux 操作系统各目录功能

- / 根目录；
- /bin 存放必要的命令；
- /boot 存放内核以及启动所需的文件；
- /dev 存放硬件设备文件；
- /etc 存放系统配置文件；
- /home 普通用户的宿主目录，用户数据存放在其主目录中 ；
- /lib 存放必要的运行库；

- /mnt 存放临时的映射文件系统，通常用来挂载使用；
- /proc 存放存储进程和系统信息；
- /root 超级用户的主目录；
- /sbin 存放系统管理程序；
- /tmp 存放临时文件；
- /usr 存放应用程序，命令程序文件、程序库、手册和其它文档；

  /var 系统默认日志存放目录。

第3章 Linux 必备命令实战篇

Linux 系统启动默认为字符界面，一般不会启动图形界面，所以对命令行的熟练程度能更加方便、高效的管理 Linux 系统。

本章向读者介绍 Linux 系统必备命令各项参数及功能场景，Linux 常见命令包括：cd、ls、pwd、mkdir、rm、cp、mv、touch、cat、head、tail、chmod、vim 等。

3.1 Linux 命令集

初学者完成 Linux 系统安装以后，学习 Linux 操作系统必备的指令，基于 Linux 指令管理 Linux 操作系统，必备 Linux 指令有哪些?

- 基础命令相关一：

Cd、ls、pwd、help、man、if、for、while、case、select、read、test、ansible、iptables、firewall-cmd、salt、mv、cut、uniq、sort、wc、source、sestatus、setenforce；

- 基础命令相关二：

Date、ntpdate、crontab、rsync、ssh、scp、nohup、sh、bash、hostname、hostnamectl、

source、ulimit、export、env、set、at、dir、db_load、diff、dmsetup、declare；

- 用户权限相关：

Useradd、userdel、usermod、groupadd、groupmod、groupdel、Chmod、chown、chgrp、umask、chattr、lsattr、id、who、whoami、last、su、sudo、w、chpasswd、chroot；

- 文件管理相关：

Touch、mkdir、rm、rmdi、vi、vim、cat、head、tail、less、more、find、sed、grep、awk、echo、ln、stat、file；

- 软件资源管理：

Rpm、yum、tar、unzip、zip、gzip、wget、curl、rz、sz、jar、apt-get、bzip2、service、systemctl、make、cmake、chkconfig；

- 系统资源管理：

Fdisk、mount、umount、mkfs.ext4、fsck.ext4、parted、lvm、dd、du、df、top、iftop、free、w、uptime、iostat、vmstat、iotop、ps、netstat、lsof、ss、sar；

- 网络管理相关：

Ping、ifconfig、ip addr、ifup、ifdown、nmcli、route、nslookup、traceroute、dig、tcpdump、nmap、brctl、ethtool、setup、arp、ab、iperf；

- Linux 系统开关机：

Init、reboot、shutdown、halt、poweroff、runlevel、login、logout、exit；

3.2 cd 命令详解

cd 命令主要用于目录切换，例如：cd /home 切换至/home 目录，cd /root 表示切

换至/root 目录 ； cd ../切换至上一级目录；cd ./切换至当前目录。

其中.和..可以理解为相对路径，例如 cd ./test 表示以当前目录为参考，表示相对于当前，而 cd /home/test 表示完整的路径，理解为绝对路径），如图 4-1 所示：

```
[root@www-jfedu-net tmp]# cd
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# cd /home/
[root@www-jfedu-net home]#
[root@www-jfedu-net home]# cd /root/
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# cd /
[root@www-jfedu-net /]#
[root@www-jfedu-net /]# cd /usr/local/sbin/
[root@www-jfedu-net sbin]#
[root@www-jfedu-net sbin]# cd /root/
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# cd /opt/
[root@www-jfedu-net opt]#
[root@www-jfedu-net opt]#
```

图 4-1 Linux cd 命令操作

3.3 ls 命令详解

ls 命令主要用于浏览目录下的文件或者文件夹，使用方法参考：ls ./ 查看当前目录所有的文件和目录，ls -a 查看所有的文件，包括隐藏文件,以.开头的文件，常用参数详解如下:

-a, --all	不隐藏任何以. 开始的项目；
-A, --almost-all	列出除. 及.. 以外的任何项目；
--author	与-l 同时使用时列出每个文件的作者；
-b, --escape	以八进制溢出序列表示不可打印的字符；
--block-size=大小	块以指定大小的字节为单位；
-B, --ignore-backups	不列出任何以"~"字符结束的项目；
-d, --directory	当遇到目录时列出目录本身而非目录内的文件；

-D, --dired	产生适合 Emacs 的 dired 模式使用的结果；
-f	不进行排序，-aU 选项生效，-lst 选项失效；
-i, --inode	显示每个文件的 inode 号；
-I, --ignore=PATTERN	不显示任何符合指定 shell PATTERN 的项目；
-k	即--block-size=1K；
-l	使用较长格式列出信息；
-n, --numeric-uid-gid	类似 -l，但列出 UID 及 GID 号；
-N, --literal	输出未经处理的项目名称（如不特别处理控制字符）；
-r, --reverse	排序时保留顺序；
-R, --recursive	递归显示子目录；
-s, --size	以块数形式显示每个文件分配的尺寸；
-S	根据文件大小排序；
-t	根据修改时间排序；
-u	同-lt 一起使用：按照访问时间排序并显示； 同-l 一起使用：显示访问时间并按文件名排序； 其他：按照访问时间排序；
-U	不进行排序；按照目录顺序列出项目；
-v	在文本中进行数字(版本)的自然排序。

3.4 pwd 命令剖析

pwd 命令主要用于显示或者查看当前所在的目录路径，如图 4-2 所示：

```
[root@www-jfedu-net ~]# cd
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# pwd
/root
[root@www-jfedu-net ~]# cd /home/
[root@www-jfedu-net home]#
[root@www-jfedu-net home]# pwd
/home
[root@www-jfedu-net home]#
[root@www-jfedu-net home]# cd /tmp/
[root@www-jfedu-net tmp]#
[root@www-jfedu-net tmp]# pwd
/tmp
[root@www-jfedu-net tmp]#
[root@www-jfedu-net tmp]#
```

图 4-2 pwd 命令查看当前目录

3.5 mkdir 命令剖析

mkdir 命令主要用于创建目录，用法 mkdir dirname，命令后接目录的名称，常用参数详解如下:

用法：mkdir [选项]... 目录；若指定目录不存在则创建目录；	
长选项必须使用的参数对于短选项时也是必需使用的；	
-m, --mode=模式	设置权限模式(类似 chmod)，而不是 rwxrwxrwx 减 umask；
-p, --parents	需要时创建目标目录的上层目录，但即使这些目录已存在也不当作错误处理；
-v, --verbose	每次创建新目录都显示信息；
-Z, --context=CTX	将每个创建的目录的 SELinux 安全环境设置为 CTX；
--help	显示此帮助信息并退出；
--version	显示版本信息并退出。

3.6 rm 命令剖析

rm 命令主要用于删除文件或者目录，用法 rm –rf test.txt（-r 表示递归，-f 表示强制），常用参数详解如下:

用法：rm [选项]... 文件...删除（unlink）文件。

-f, --force 强制删除。忽略不存在的文件，不提示确认；

-i 在删除前需要确认；

-I 在删除超过三个文件或者递归删除前要求确认。此选项比-i 提示内容更少，但同样可以阻止大多数错误发生；

-r, -R, --recursive 递归删除目录及其内容；

-v, --verbose 详细显示进行的步骤；

--help 显示此帮助信息并退出；

--version 显示版本信息并退出；

默认时，rm 不会删除目录，使用--recursive(-r 或-R)选项可删除每个给定的目录，以及其下所有的内容；

要删除第一个字符为"-"的文件（例如"-foo"），请使用以下方法之一:

rm -- -foo

rm ./-foo

3.7 cp 命令剖析

cp 命令主要用于拷贝文件，用法,cp old.txt /tmp/new.txt ，常用来备份，如果拷贝目录需要加-r 参数，常用参数详解如下:

用法：cp [选项]... [-T] 源文件 目标文件

或：cp [选项]... 源文件... 目录

或：cp [选项]... -t 目录 源文件...

将源文件复制至目标文件，或将多个源文件复制至目标目录。

长选项必须使用的参数对于短选项时也是必需使用的。

-a, --archive 等于-dR --preserve=all；

--backup[=CONTROL 为每个已存在的目标文件创建备份；

-b 类似--backup 但不接受参数；

--copy-contents 在递归处理是复制特殊文件内容；

-d 等于--no-dereference --preserve=links；

-f, --force 如果目标文件无法打开则将其移除并重试(当 -n 选项；

存在时则不需再选此项)；

-i, --interactive 覆盖前询问(使前面的 -n 选项失效)；

-H 跟随源文件中的命令行符号链接；

-l, --link 链接文件而不复制；

-L, --dereference 总是跟随符号链接；

-n, --no-clobber 不要覆盖已存在的文件(使前面的 -i 选项失效)；

-P, --no-dereference 不跟随源文件中的符号链接；

-p 等于--preserve=模式,所有权,时间戳；

--preserve[=属性列表 保持指定的属性(默认：模式,所有权,时间戳)，如果；

	可能保持附加属性：环境、链接、xattr 等；
-c	same as --preserve=context；
--sno-preserve=属性列表	不保留指定的文件属性；
--parents	复制前在目标目录创建来源文件路径中的所有目录；
-R, -r, --recursive	递归复制目录及其子目录内的所有内容。

3.8 mv 命令剖析

mv 命令主要用于重命名或者移动文件或者目录，用法, mv old.txt new.txt，常用参数详解如下:

用法：mv [选项]... [-T] 源文件 目标文件；	
或：mv [选项]... 源文件... 目录；	
或：mv [选项]... -t 目录 源文件；	
将源文件重命名为目标文件，或将源文件移动至指定目录。长选项必须使用的参数对于短选项时也是必需使用的。	
--backup[=CONTROL]	为每个已存在的目标文件创建备份；
-b	类似--backup 但不接受参数；
-f, --force	覆盖前不询问；
-i, --interactive	覆盖前询问；
-n, --no-clobber	不覆盖已存在文件，如果您指定了-i、-f、-n 中的多个，仅最后一个生效；
--strip-trailing-slashes	去掉每个源文件参数尾部的斜线；

-S, --suffix=SUFFIX	替换常用的备份文件后缀；
-t, --target-directory=DIRECTORY	将所有参数指定的源文件或目录；
	移动至 指定目录；
-T, --no-target-directory	将目标文件视作普通文件处理；
-u, --update	只在源文件文件比目标文件新，或目标文件；
	不存在时才进行移动；
-v, --verbose	详细显示进行的步骤；
--help	显示此帮助信息并退出；
--version	显示版本信息并退出。

3.9 touch 命令剖析

touch 命令主要用于创建普通文件，用法为 touch test.txt，如果文件存在，则表示修改当前文件时间，常用参数详解如下:

用法：touch [选项]... 文件...

将每个文件的访问时间和修改时间改为当前时间；

不存在的文件将会被创建为空文件，除非使用-c 或-h 选项；

如果文件名为"-"则特殊处理，更改与标准输出相关的文件的访问时间；

长选项必须使用的参数对于短选项时也是必需使用的；

-a	只更改访问时间；
-c, --no-create	不创建任何文件；
-d, --date=字符串	使用指定字符串表示时间而非当前时间；
-f	(忽略)；

-h, --no-dereference	会影响符号链接本身，而非符号链接所指示的目的地；
	(当系统支持更改符号链接的所有者时，此选项才有用)；
-m	只更改修改时间；
-r, --reference=文件	使用指定文件的时间属性而非当前时间；
-t STAMP	使用[[CC]YY]MMDDhhmm[.ss] 格式的时间而非当前时间；
--time=WORD	使用 WORD 指定的时间：access、atime、use 都等于-a；
	选项的效果，而 modify、mtime 等于-m 选项的效果；
--help	显示此帮助信息并退出；
--version	显示版本信息并退出。

3.10 cat 命令剖析

cat 命令主要用于查看文件内容，用法 cat test.txt 可以查看 test.txt 内容，常用参数详解如下:

用法：cat [选项]... [文件]...	
将[文件]或标准输入组合输出到标准输出。	
-A, --show-all	等于-vET；
-b, --number-nonblank	对非空输出行编号；
-e	等于-vE；
-E, --show-ends	在每行结束处显示"$"；

-n, --number	对输出的所有行编号；
-s, --squeeze-blank	不输出多行空行；
-t	与-vT 等价；
-T, --show-tabs	将跳格字符显示为^I；
-u	(被忽略)；
-v, --show-nonprinting	使用^ 和 M- 引用，除了 LFD 和 TAB 之外；
--help	显示此帮助信息并退出；
--version	显示版本信息并退出。

cat 还有一种用法，cat ...EOF...EOF，表示追加内容至/tmp/test.txt 文件中，如下：

```
cat >>/tmp/test.txt<<EOF
My Name is JFEDU.NET
I am From Bei jing.
EOF
```

cat test.txt |more 分页显示 text 内容，|符号是管道符，用于把|前的输出作为后面命令的输入。More 命令常用于分页查看某文件或者内容。

3.11 head 命令剖析

head 命令主要用于查看文件内容，通常查看文件前 10 行，head -10 /var/log/messages 可以查看该文件前 10 行的内容，常用参数详解如下:

```
用法：head [选项]... [文件]...
将每个指定文件的头 10 行显示到标准输出。
如果指定了多于一个文件，在每一段输出前会给出文件名作为文件头。
```

如果不指定文件，或者文件为"-"，则从标准输入读取数据，长选项必须使用的参数对于短选项时也是必需使用的；

-q, --quiet, --silent　　不显示包含给定文件名的文件头；

-v, --verbose　　总是显示包含给定文件名的文件头；

--help　　显示此帮助信息并退出；

--version　　显示版本信息并退出；

-c, --bytes=[-]K　　显示每个文件的前 K 字节内容，如果附加"-"参数，则除了每个文件的最后 K 字节数据外显示剩余全部内容；

-n, --lines=[-]K　　显示每个文件的前 K 行内容，如果附加"-"参数，则除了每个文件的最后 K 行外显示剩余全部内容。

3.12 tail 命令剖析

tail 命令主要用于查看文件内容，通常查看末尾 10 行，tail –fn 100 /var/log/messages 可以实时查看该文件末尾 100 行的内容，常用参数详解如下:

用法：tail [选项]... [文件]...

显示每个指定文件的最后 10 行到标准输出。

若指定了多于一个文件，程序会在每段输出的开始添加相应文件名作为头。

如果不指定文件或文件为"-"，则从标准输入读取数据。

长选项必须使用的参数对于短选项时也是必需使用的。

-n, --lines=K　　输出的总行数，默认为 10 行；

-q, --quiet, --silent　　不输出给出文件名的头；

--help　　显示此帮助信息并退出；

```
--version                                显示版本信息并退出；

-f, --follow[={name|descriptor}]         即时输出文件变化后追加的数据；

                -f, --follow 等于--follow=descriptor

-F              即--follow=name –retry

-c, --bytes=K                            输出最后 K 字节；另外，使用-c +K 从每

个文件的第 K 字节输出。
```

3.13 chmod 命令剖析

chmod 命令主要用于修改文件或者目录的权限，例如 chmod o+w test.txt，赋予 test.txt 其他人 w 写权限，常用参数详解如下:

```
用法：chmod [选项]... 模式[,模式]... 文件...

  或：chmod [选项]... 八进制模式 文件...

  或：chmod [选项]... --reference=参考文件 文件，将每个文件的模式更改为指定值。

-c, --changes                    类似 --verbose，但只在有更改时才显示结果

    --no-preserve-root           不特殊对待根目录(默认)；

    --preserve-root              禁止对根目录进行递归操作；

-f, --silent, --quiet            去除大部份的错误信息；

-R, --recursive                  以递归方式更改所有的文件及子目录；

--help                           显示此帮助信息并退出；

--version                        显示版本信息并退出；

-v, --verbose                    为处理的所有文件显示诊断信息；

--reference=参考文件             使用指定参考文件的模式，而非自行指定权限
```

模式。

3.14 chown 命令剖析

chown 命令主要用于文件或者文件夹宿主及属组的修改，命令格式例如 chown –R root.root /tmp/test.txt，表示修改 test.txt 文件的用户和组均为 root，常用参数详解如下:

用法：chown [选项]... [所有者][:[组]] 文件...

或：chown [选项]... --reference=参考文件 文件...

更改每个文件的所有者和/或所属组。

当使用 --referebce 参数时，将文件的所有者和所属组更改为与指定参考文件相同。

-f, --silent, --quiet 去除大部份的错误信息

--reference=参考文件	使用参考文件的所属组，而非指定值；
-R, --recursive	递归处理所有的文件及子目录；
-v, --verbose	为处理的所有文件显示诊断信息；
-H	命令行参数是一个通到目录的符号链接，则遍历符号链接；
-L	历每一个遇到的通到目录的符号链接；
-P	历任何符号链接(默认)；
--help	显示帮助信息并退出；
--version	显示版本信息并退出。

3.15 echo 命令剖析

echo 命令主要用于打印字符或者回显，例如输入 echo ok，会显示 ok，echo ok >

test.txt 则会把 ok 字符覆盖 test.txt 内容。>表示覆盖，原内容被覆盖，>>表示追加，原内容不变。

例如 echo ok >> test.txt,表示向 test.txt 文件追加 OK 字符,不覆盖原文件里的内容，常用参数详解如下:

使用-e 扩展参数选项时，与如下参数一起使用，有不同含义，例如:

\a 发出警告声

\b 删除前一个字符

\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;

\033[30m 黑色字 \033[0m

\033[31m 红色字 \033[0m

\033[32m 绿色字 \033[0m

\033[33m 黄色字 \033[0m

\033[34m 蓝色字 \033[0m

\033[35m 紫色字 \033[0m

\033[36m 天蓝字 \033[0m

\033[37m 白色字 \033[0m

\033[40;37m 黑底白字 \033[0m

```
\033[41;37m 红底白字 \033[0m

\033[42;37m 绿底白字 \033[0m

\033[43;37m 黄底白字 \033[0m

\033[44;37m 蓝底白字 \033[0m

\033[45;37m 紫底白字 \033[0m

\033[46;37m 天蓝底白字 \033[0m

\033[47;30m 白底黑字 \033[0m
```

echo 颜色打印扩展，auto_lamp_v2.sh 内容如下：

```
echo -e "\033[36mPlease Select Install Menu follow:\033[0m"

echo -e "\033[32m1)Install Apache Server\033[1m"

echo "2)Install MySQL Server"

echo "3)Install PHP Server"

echo "4)Configuration index.php and start LAMP server"

echo -e "\033[31mUsage: { /bin/sh $0 1|2|3|4|help}\033[0m"
```

执行结果如图 4-3 所示：

```
[root@www-jfedu-net ~]# sh auto_lamp_v2.sh
-----------------------------

Please Select Install Menu follow:
1)Install Apache Server
2)Install MySQL Server
3)Install PHP Server
4)Configuration index.php and start LAMP server
Usage: { /bin/sh auto_lamp_v2.sh 1|2|3|4|help}
[root@www-jfedu-net ~]#
```

图 4-3 echo –e 颜色打印

3.16 df 命令剖析

df 命令常用于磁盘分区查询，常用命令 df –h，查看磁盘分区信息，常用参数详解如下:

用法：df [选项]... [文件]...

显示每个文件所在的文件系统的信息，默认是显示所有文件系统。

长选项必须使用的参数对于短选项时也是必需使用的。

-a, --all	显示所有文件系统的使用情况，包括虚拟文件系统；
-B, --block-size=SIZE	使用字节大小块；
-h, --human-readable	以人们可读的形式显示大小；
-H, --si	同-h，但是强制使用 1000 而不是 1024；
-i, --inodes	显示 inode 信息而非块使用量；
-k	即--block-size=1K；
-l, --local	只显示本机的文件系统；
--no-sync	取得使用量数据前不进行同步动作(默认)；
-P, --portability	使用 POSIX 兼容的输出格式；
--sync	取得使用量数据前先进行同步动作；
-t, --type=类型	只显示指定文件系统为指定类型的信息；
-T, --print-type	显示文件系统类型；
-x, --exclude-type=类型	只显示文件系统不是指定类型信息；
--help	显示帮助信息并退出；
--version	显示版本信息并退出。

3.17 du 命令剖析

du 命令常用于查看文件在磁盘中的使用量，常用命令 du -sh，查看当前目录所有文件及文件及的大小，常用参数详解如下:

用法：du [选项]... [文件]...

或：du [选项]... --files0-from=F

计算每个文件的磁盘用量，目录则取总用量。

长选项必须使用的参数对于短选项时也是必需使用的。

-a, --all 输出所有文件的磁盘用量，不仅仅是目录；

--apparent-size 显示表面用量，而并非是磁盘用量；虽然表面用量通常会小一些，但有时它会因为稀疏文件间的"洞"、内部碎片、非直接引用的块等原因而变大；

-B, --block-size=大小 使用指定字节数的块；

-b, --bytes 等于--apparent-size --block-size=1；

-c, --total 显示总计信息；

-H 等于--dereference-args (-D)；

-h, --human-readable 以可读性较好的方式显示尺寸(例如：1K 234M 2G)；

--si 类似-h，但在计算时使用 1000 为基底而非 1024；

-k 等于--block-size=1K；

-l, --count-links 如果是硬连接，就多次计算其尺寸；

-m 等于--block-size=1M；

-L, --dereference	找出任何符号链接指示的真正目的地；
-P, --no-dereference	不跟随任何符号链接(默认)；
-0, --null	将每个空行视作 0 字节而非换行符；
-S, --separate-dirs	不包括子目录的占用量；
-s, --summarize	只分别计算命令列中每个参数所占的总用量；
-x, --one-file-system	跳过处于不同文件系统之上的目录；
-X, --exclude-from=文件	排除与指定文件中描述的模式相符的文件；
-D, --dereference-args	解除命令行中列出的符号连接；
--files0-from=F	计算文件 F 中以 NUL 结尾的文件名对应占用的磁盘空间，如果 F 的值是"-"，则从标准输入读入文件名。

如上为 Linux 初学者必备命令，当然 Linux 命令还有很多，后面章节会随时学习新的命令。

第4章 VIM 编辑器实战篇

4.1 VIM 编辑器简介

VI 是一个命令行界面下的文本编辑工具，最早在 1976 年由 Bill Joy 开发，当时名字叫做 ex。vi 支持绝大多数操作系统（最早在 BSD 上发布），并且功能已经十分强大

1991 年 Bram Moolenaar 基于 vi 进行改进，发布了 vim，加入了对 GUI 的支持。

随着 VIM 更新发展，VIM 已经不是普通意义上的文本编辑器，而是被广泛的作为在文本编辑、文本处理、代码开发等用途，Linux 中主流的文本编辑器包括：VI、Vim、Sublime、Emacs、Light Table、Eclipse、Gedit 等。

4.2 VIM 编辑器键盘

Vim 强大的编辑能力中很大部分是来自于其普通模式命令。vim 的设计理念是命令的组合。

- “5dd” 5 表示总共 5 行，删除光标所在后的 5 行，包含光标行；
- “d$” $"代表行尾,删除到行尾的内容，包含光标；
- “2yy” 表示复制光标及后 2 行，包括光标行；
- “%d” %代表全部或者全局,%d 表示删除文本所有的内容，也即是清空文档所有的内容。

VIM 是一个主流开源的编辑器，其默认执行 vim 命令，会显示帮助乌干达贫困的孩子，如图 4-4 为 vim 与键盘键位功能对应关系：

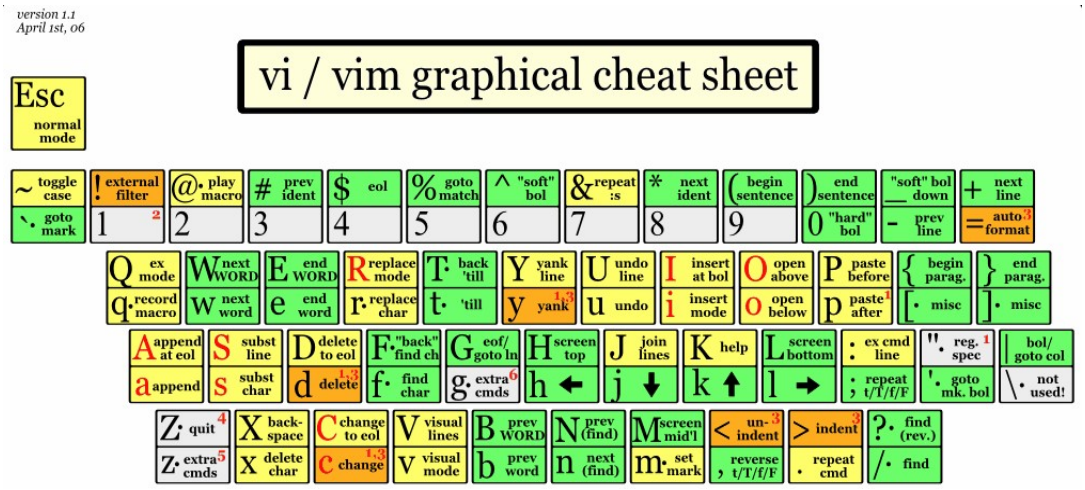


图 4-4 vim 与键盘位置对应关系

4.3 VIM 编辑器三种模式

Vim 编辑器模式常用有三种，分别是：

- 命令行模式；
- 文本输入模式；

- 末行模式。

vim 是 vi 的升级版本，它是安装在 Linux 操作系统中的一个软件，官网为：www.vim.org

在 Linux Shell 终端下默认执行 vim 命令，按 Enter 键后：

- 默认进入命令行模式；
- 在命令行模式按 i 进入文本输入模式；
- 按 ESC 进入命令行模式；
- 按:进入末行模式。

4.4 VIM 命令行模式实战

VIM 编辑器最强大的功能，就在于内部命令及规则使用，如下为 VIM 编辑器最常用的语法及规则：

- 命令行模式：可以删除、复制、粘贴、撤销，可以切换到输入模式，输入模式跳转至命令行模式：按 ESC 键。

yy	复制光标所在行；
nyy	复制 n 行；
3yy	复制 3 行；
p,P	粘贴；
yw	复制光标所在的词组，不会复制标点符号；
3yw	复制三个词组；
u	撤消上一次；
U	撤消当前所有；
dd	删除整行；

ndd	删除 n 行；
x	删除一个字符；
u	逐行撤销；
dw	删除一个词组；
a	从光标所在字符后一个位置开始录入；
A	从光标所在行的行尾开始录入；
i	从光标所在字符前一个位置开始录入；
I	从光标所在行的行首开始录入；
o	跳至光标所在行的下一行行首开始录入；
O	跳至光标所在行的上一行行首开始录入；
R	从光标所在位置开始替换；

4.5 VIM 末行模式实战

- 末行模式主要功能包括：查找、替换、末行保存、退出等；

:w	保存；
:q	退出；
:s/x/y	替换 1 行；
:wq	保存退出；
1,5s/x/y	替换 1,5 行；
:wq!	强制保存退出；
1,$sx/y	从第一行到最后一行；

:q!	强制退出；
:x	保存；
/word	从前往后找，正向搜索；
?word	从后往前走，反向搜索；
:s/old/new/g	将 old 替换为 new，前提是光标一定要移到那一行；
:s/old/new	将这一行中的第一次出现的 old 替换为 new，只替换第一个；
:1,$s/old/new/g	第一行到最后一行中的 old 替换为 new；
:1,2,3s/old/new/g	第一行第二行第三行中的 old 改为 new；
vim +2 jfedu.txt	打开 jfedu.txt 文件，并将光标定位在第二行；
vim +/string jfedu.txt	打开 jfedu.txt 文件，并搜索关键词。

4.6 本章小结

通过对本章内容的学习，读者对 Linux 操作系统引导有了进一步的理解，能够快速解决 Linux 启动过程中的故障，同时学习了 CentOS6 与 CentOS7 系统的区别，理解了 TCP/IP 协议及 IP 地址相关基础内容。

学会了 Linux 初学必备的 16 个 Linux 命令，能使用命令熟练的操作 Linux 系统，通过对 VIM 编辑器的深入学习，能够熟练编辑、修改系统中任意的文本及配置文件。对 Linux 系统的认识及操作有了更进一步的飞跃。

4.7 同步作业

1. 修改密码的命令默认为 passwd，需要按 Enter 键两次，如何一条命令快速修改密码呢？
2. 企业服务器，某天发现系统访问很慢，需要查看系统内核日志，请写出查看系统内核日

志的命令；

3. 如果在 Linux 系统/tmp/目录，快速创建 1000 个目录，目录名为：jfedu1、jfedu2、jfedu 依次类推，不断增加；

4. Httpd.conf 配置文件中存在很多以#号开头的行，请使用 vim 相关指令删除#开头的行；

第5章 Linux 用户及权限管理篇

Linux 是一个多用户的操作系统，引入用户，可以更加方便管理 Linux 服务器，系统默认需要以一个用户的身份登入，而且在系统上启动进程也需要以一个用户身份去运行，用户可以限制某些进程对特定资源的权限控制。

本章向读者介绍 Linux 系统如何管理创建、删除、修改用户角色、用户权限配置、组权限配置及特殊权限深入剖析。

5.1 Linux 用户及组

Linux 操作系统对多用户的管理，是非常繁琐的，所以用组的概念来管理用户就变得简单，每个用户可以在一个独立的组，每个组也可以有零个用户或者多个用户。

Linux 系统用户是根据用户 ID 来识别的，从默认 ID 编号从 0 开始，但是为了和老式系统兼容，用户 ID 限制在 60000 以下，Linux 用户分总共分为三种，分别如下：

- root 用户 （ID 0）
- 系统用户 （ID 1-499）
- 普通用户 （ID 500 以上）

Linux 系统中的每个文件或者文件夹，都有一个所属用户及所属组，使用 id 命令可以显示当前用户的信息，使用 passwd 命令可以修改当前用户密码。Linux 操作系统用户的特

点如下：

- 每个用户拥有一个 UserID，操作系统实际读取的是 UID，而非用户名；
- 每个用户属于一个主组，属于一个或多个附属组，一个用户最多有 31 个附属组；
- 每个组拥有一个 GroupID；
- 每个进程以一个用户身份运行，该用户可对进程拥有资源控制权限；
- 每个可登陆用户拥有一个指定的 Shell 环境。

5.2 Linux 用户管理

Linux 用户在操作系统可以进行日常管理和维护，涉及到的相关配置文件如下：

- /etc/passwd 保存用户信息
- /etc/shdaow 保存用户密码（以加密形式保存）
- /etc/group 保存组信息
- /etc/login.defs 用户属性限制,密码过期时间,密码最大长度等限制
- /etc/default/useradd 显示或更改默认的 useradd 配置文件

如需创建新用户,可以使用命令 useradd,执行命令 useradd jfedu1 即可创建 jfedu1 用户，同时会创建一个同名的组 jfedu1，默认该用户属于 jfedu1 主组。

Useradd jfedu1 命令默认创建用户 jfedu1，会根据如下步骤进行操作：

- 读取/etc/default/useradd，根据配置文件执行创建操作；
- 在/etc/passwd 文件中添加用户信息；
- 如使用 passwd 命令创建密码，密码会被加密保存在/etc/shdaow 中；
- 为 jfedu1 创建家目录：/home/jfedu1；
- 将/etc/skel 中的.bash 开头的文件复制至/home/jfedu1 家目录；
- 创建与用户名相同的 jfedu1 组，jfedu1 用户默认属于 jfeud1 同名组；

- Jfedu1 组信息保存在/etc/group 配置文件中。

在使用 useradd 命令创建用户时，可以支持如下参数：

```
用法：useradd [选项] 登录

useradd -D

useradd -D [选项]

选项：

-b, --base-dir BASE_DIR                 指定新账户的家目录；

-c, --comment COMMENT                   新账户的 GECOS 字段；

-d, --home-dir HOME_DIR                 新账户的主目录；

-D, --defaults                          显示或更改默认的 useradd 配置；

-e, --expiredate EXPIRE_DATE            新账户的过期日期；

-f, --inactive INACTIVE                 新账户的密码不活动期；

-g, --gid GROUP                         新账户主组的名称或 ID；

-G, --groups GROUPS                     新账户的附加组列表；

-h, --help                              显示此帮助信息并推出；

-k, --skel SKEL_DIR                     使用此目录作为骨架目录；

-K, --key KEY=VALUE                     不使用 /etc/login.defs 中的默认值；

-l, --no-log-init                       不要将此用户添加到最近登录和登录失败数
据库；

-m, --create-home                       创建用户的主目录；

-M, --no-create-home                    不创建用户的主目录；

-N, --no-user-group                     不创建同名的组；
```

-o, --non-unique	允许使用重复的 UID 创建用户；
-p, --password PASSWORD	加密后的新账户密码；
-r, --system	创建一个系统账户；
-R, --root CHROOT_DIR	chroot 到的目录；
-s, --shell SHELL	新账户的登录 shell；
-u, --uid UID	新账户的用户 ID；
-U, --user-group	创建与用户同名的组；
-Z, --selinux-user SEUSER	为 SELinux 用户映射使用指定 SEUSER。

Useradd 案例演示：

（1） 新建 jfedu 用户，并加入到 jfedu1，jfedu2 附属组；

```
useradd -G jfedu1,jfedu2 jfedu
```

（2） 新建 jfedu3 用户，并指定新的家目录，同时指定其登陆的 SHELL；

```
useradd jfedu3 -d /tmp/ -s /bin/bash
```

5.3 Linux 组管理

所有的 Linux 或者 Windows 系统都有组的概念，通过组可以更加方便的管理用户，组的概念应用于各行行业，例如企业会使用部门、职能或地理区域的分类方式来管理成员，映射在 Linux 系统，同样可以创建用户，并用组的概念对其管理。

Linux 组有如下特点：

- 每个组有一个组 ID；
- 组信息保存在/etc/group 中；

- 每个用户至少拥有一个主组，同时还可以拥有 31 个附属组。

通过命令 groupadd、groupdel、groupmod 来对组进行管理，详细参数使用如下：

```
groupadd 用法
-f, --force                    如果组已经存在则成功退出；
                               并且如果 GID 已经存在则取消 –g；
-g, --gid GID                  为新组使用 GID；
-h, --help                     显示此帮助信息并推出；
-K, --key KEY=VALUE            不使用 /etc/login.defs 中的默认值；
-o, --non-unique               允许创建有重复 GID 的组；
-p, --password PASSWORD        为新组使用此加密过的密码；
-r, --system                   创建一个系统账户；
groupmod 用法
-g, --gid GID                  将组 ID 改为 GID；
-h, --help                     显示此帮助信息并推出；
-n, --new-name NEW_GROUP       改名为 NEW_GROUP；
-o, --non-unique               允许使用重复的 GID；
-p, --password PASSWORD        将密码更改为(加密过的) PASSWORD；
groupdel 用法
groupdel jfedu                 删除 jfedu 组；
```

Groupadd 案例演示：

（1） groupadd 创建 jingfeng 组

```
groupadd jingfeng
```

（2） groupadd 创建 jingfeng 组，并指定 GID 为 1000；

```
groupadd -g 1000 jingfeng
```

（3） groupadd 创建一个 system 组，名为 jingfeng 组

```
groupadd  -r  jingfeng
```

Groupmod 案例演示：

（4） groupmod 修改组名称，将 jingfeng 组名，改成 jingfeng1；

```
groupmod -n jingfeng1 jingfeng
```

（5） groupmod 修改组 GID 号，将原 jingfeng1 组 gid 改成 gid 1000；

```
groupmod –g 1000 jingfeng1
```

5.4 Linux 用户及组案例

Useradd 主要用于新建用户,而用户新建完毕，可以使用 usermod 来修改用户及组的属性，如下为 usermod 详细参数：

```
用法：usermod [选项] 登录

选项：

-c, --comment 注释                  GECOS 字段的新值；

-d, --home HOME_DIR                 用户的新主目录；

-e, --expiredate EXPIRE_DATE        设定帐户过期的日期为 EXPIRE_DATE；

-f, --inactive INACTIVE             过期 INACTIVE 天数后，设定密码为失效状

态；

-g, --gid GROUP                     强制使用 GROUP 为新主组；

-G, --groups GROUPS                 新的附加组列表 GROUPS；
```

-a, --append GROUP	将用户追加至上边 -G 中提到的附加组中，并不从其它组中删除此用户；
-h, --help	显示此帮助信息并推出；
-l, --login LOGIN	新的登录名称；
-L, --lock	锁定用户帐号；
-m, --move-home	将家目录内容移至新位置（仅于 -d 一起使用)；
-o, --non-unique	允许使用重复的(非唯一的) UID；
-p, --password PASSWORD	将加密过的密码（PASSWORD）设为新密码；
-R, --root CHROOT_DIR	chroot 到的目录；
-s, --shell SHELL	该用户帐号的新登录 shell 环境；
-u, --uid UID	用户帐号的新 UID；
-U, --unlock	解锁用户帐号；
-Z, --selinux-user SEUSER	用户账户的新 SELinux 用户映射。

Usermod 案例演示：

（1） 将 jfedu 用户属组修改为 jfedu1，jfedu2 附属组；

```
usermod -G jfedu1,jfedu2 jfedu
```

（2） 将 jfedu 用户加入到 jfedu3，jfedu4 附属组，-a 为添加新组，原组保留；

```
usermod –a -G jfedu3,jfedu4 jfedu
```

（3） 修改 jfedu 用户，并指定新的家目录，同时指定其登陆的 SHELL；

```
usermod -d /tmp/ -s /bin/sh jfedu
```

（4） 将 jfedu 用户名修改为 jfedu1；

```
usermod -l jfedu1 jfedu
```

（5） 锁定 jfedu1 用户及解锁 jfedu1 用户方法；

```
usermod –L  jfedu1

usermod   -U  jfedu1
```

Userdel 案例演示：

使用 userdel 可以删除指定用户及其用户的邮箱目录或者 Selinux 映射环境：

- userdel jfedu1 保留用户的家目录；
- userdel -r jfedu1 删除用户及用户家目录，用户 login 系统无法删除；
- userdel -rf jfedu1 强制删除用户及该用户家目录，不论是否 login 系统。

5.5 Linux 权限管理

Linux 权限是操作系统用来限制对资源访问的机制，权限一般分为读、写、执行。系统中每个文件都拥有特定的权限、所属用户及所属组，通过这样的机制来限制哪些用户或用户组可以对特定文件进行相应的操作。

Linux 每个进程都是以某个用户身份运行，进程的权限与该用户的权限一样，用户的权限越大，则进程拥有的权限就越大。

Lnux 中有的文件及文件夹都有至少权限三种权限，常见的权限如表 5-1 所示:

权限	对文件的影响	对目录的影响
r（读取）	可读取文件内容	可列出目录内容
w（写入）	可修改文件内容	可在目录中创建删除内容
x（执行）	可作为命令执行	可访问目录内容
目录必须拥有 x 权限，否则无法查看其内容		

表 5-1 Linux 文件及文件及权限

Linux 权限授权，默认是授权给三种角色，分别是 user、group、other，Linux 权限与用户之间的关联如下：

- U 代表 User，G 代表 Group，O 代表 Other；
- 每个文件的权限基于 UGO 进行设置；
- 权限三位一组（rwx），同时需授权给三种角色，UGO；
- 每个文件拥有一个所属用户和所属组，对应 UGO，不属于该文件所属用户或所属组使用 O 来表示；

在 Linux 系统中，可以通过 ls –l 查看 jfedu.net 目录的详细属性，如图 5-1 所示：

drwxrwxr-x	2	jfedu1 jfedu1	4096	Dec 10 01:36	jfedu.net

图 5-1 Linux jfedu.net 目录详细属性

jfedu.net 目录属性参数详解如下：

- d 表示目录，同一位置如果为-则表示普通文件；
- rwxrwxr-x 表示三种角色的权限，每三位为一种角色，依次为 u，g，o 权限，如上则表示 user 的权限为 rwx，group 的权限为 rwx，other 的权限为 r-x；

- 2 表示文件夹的链接数量，可理解为该目录下子目录的数量；
- 从左到右，第一个 jfedu1 表示该用户名，第二个 jfedu1 则为组名，其他人角色默认不显示；
- 4096 表示该文件夹占据的字节数；
- Dec 10 01:36 表示文件创建或者修改的时间；
- Jfedu.net 为目录的名，或者文件名。

5.6 Chown 属主及属组

修改某个用户、组对文件夹的属主及属组，用命令 chown 实现，案例演示如下：

（1） 修改 jfedu.net 文件夹所属的用户为 root，其中-R 参数表示递归处理所有的文件及子目录。

```
chown -R  root  jfedu.net
```

（2） 修改 jfedu.net 文件夹所属的组为 root。

```
chown  -R  :root  jfedu.net 或者 chgrp  –R  root  jfedu.net
```

（3） 修改 jfedu.net 文件夹所属的用户为 root，组也为 root。

```
chown  -R  root:root  jfedu.net
```

5.7 Chmod 用户及组权限

修改某个用户、组对文件夹的权限，用命令 chmod 实现，其中以代指 ugo，、-、=代表加入、删除和等于对应权限，具体案例如下：

（1） 授予用户对 jfedu.net 目录拥有 rwx 权限

```
chmod  –R  u+rwx  jfedu.net
```

（2） 授予组对 jfedu.net 目录拥有 rwx 权限

```
chmod  –R  g+rwx  jfedu.net
```

（3） 授予用户、组、其他人对 jfedu.net 目录拥有 rwx 权限

```
chmod  –R  u+rwx,g+rwx,o+rwx  jfedu.net
```

（4） 撤销用户对 jfedu.net 目录拥有 w 权限

```
chmod  –R  u-w  jfedu.net
```

（5） 撤销用户、组、其他人对 jfedu.net 目录拥有 x 权限

```
chmod  –R  u-x,g-x,o-x  jfedu.net
```

（6） 授予用户、组、其他人对 jfedu.net 目录只有 rx 权限

```
chmod  –R  u=rx,g=rx,o=rx  jfedu.net
```

5.8 Chmod 二进制权限

Linux 权限默认使用 rwx 来表示，为了更简化在系统中对权限进行配置和修改，Linux 权限引入二进制表示方法，如下代码：

```
Linux 权限可以将 rwx 用二进制来表示，其中有权限用 1 表示，没有权限用 0 表示；

Linux 权限用二进制显示如下：

rwx=111

r-x=101

rw-=110

r--=100

依次类推，转化为十进制，对应十进制结果显示如下：

rwx=111=4+2+1=7

r-x=101=4+0+1=5
```

```
rw-=110=4+4+0=6

r--=100=4+0+0=4

得出结论，用 r=4,w=2,x=1 来表示权限。
```

使用二进制方式来修改权限案例演示如下，其中默认 jfedu.net 目录权限为 755：

（1） 授予用户对 jfedu.net 目录拥有 rwx 权限

```
chmod  –R  755  jfedu.net
```

（2） 授予组对 jfedu.net 目录拥有 rwx 权限

```
chmod  –R  775  jfedu.net
```

（3） 授予用户、组、其他人对 jfedu.net 目录拥有 rwx 权限

```
chmod  –R  777  jfedu.net
```

（4） 撤销用户对 jfedu.net 目录拥有 w 权限

```
chmod  –R  555  jfedu.net
```

（5） 撤销用户、组、其他人对 jfedu.net 目录拥有 x 权限

```
chmod  –R  644  jfedu.net
```

（6） 授予用户、组、其他人对 jfedu.net 目录只有 rx 权限

```
chmod  –R  555  jfedu.net
```

5.9 Linux 特殊权限及掩码

Linux 权限除了常见的 rwx 权限之外，还有很多特殊的权限，细心的读者会发现，为什么 Linux 目录默认权限 755，而文件默认权限为 644 呢，这是因为 Linux 权限掩码 umask 导致。

每个 Linux 终端都拥有一个 umask 属性，umask 熟悉可以用来确定新建文件、目录的

默认权限，默认系统权限掩码为 022。在系统中每创建一个文件或者目录，文件默认权限是 666，而目录权限则为 777，权限对外开放比较大，所以设置了权限掩码之后，默认的文件和目录权限减去 umask 值才是真实的文件和目录的权限。

- 对应目录权限为：777-022=755；
- 对应文件权限为：666-022=644；
- 执行 umask 命令可以查看当前默认的掩码，umask -S 023 可以设置默认的权限掩码。

在 Linux 权限中，除了普通权限外，还有如下表 5-2 所示，三个特殊权限：

权限	对文件的影响	对目录的影响
Suid	以文件的所属用户身份执行，而非执行文件的用户	无
sgid	以文件所属组身份去执行	在该目录中创建任意新文件的所属组与该目录的所属组相同
sticky	无	对目录拥有写入权限的用户仅可以删除其拥有的文件，无法删除其他用户所拥有的文件

表 5-2 Linux 三种特殊权限

Linux 中设置特殊权限方法如下：

- 设置 suid： chmod u+s jfedu.net
- 设置 sgid： chmod g+s jfedu.net

- 设置 sticky：　　chmod o+t jfedu.net

特殊权限与设置普通权限一样，可以使用数字方式表示：

- SUID　=　4
- SGID　=　2
- Sticky　=　1

可以通过 chmod 4755 jfedu.net 对该目录授予特殊权限为 s 的权限，Linux 系统中 s 权限的应用常见包括：su、passwd、sudo，如图 5-2 所示：

```
[root@www-jfedu-net ~]# ll /bin/su
-rwsr-xr-x 1 root root 34904 5月  11 2016 /bin/su
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll /usr/bin/pa
package-cleanup          pango-querymodules-64    paste
package-stash-conflicts  pango-view               patch
pal2rgb                  passwd                   patchperl
[root@www-jfedu-net ~]# ll /usr/bin/passwd
-rwsr-xr-x 1 root root 30768 11月 24 2015 /usr/bin/passwd
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll /usr/bin/sudo
sudo        sudoedit    sudoreplay
[root@www-jfedu-net ~]# ll /usr/bin/sudo
---s--x--x 1 root root 123832 12月  7 08:36 /usr/bin/sudo
You have new mail in /var/spool/mail/root
[root@www-jfedu-net ~]#
```

图 5-2 Linux 特殊权限 s 应用

第6章　Linux 软件企业实战篇

通过前几章的学习，读者掌握了 Linux 系统基本命令、用户及权限等知识，Linux 整个体系的关键不在于系统本身，而是在于可以基于 Linux 系统去安装和配置企业中相关的软件、数据及应用程序，所以对软件的维护是运维工程师的重中之重。

本章向读者介绍 Linux 系统软件的安装、卸载、配置、维护以及如何构建企业本地 YUM 光盘源及 HTTP 本地源。

6.1 RPM 软件包管理

Linux 软件包管理大致可分为二进制包、源码包，使用的工具也各不相同。Linux 常见软件包分为两种，分别是源代码包（Source Code）、二进制包（Binary Code），源代码包是没有经过编译的包，需要经过 GCC、C++编译器环境编译才能运行，二进制包无需编译，可以直接安装使用。

通常而言，可以通过后缀简单区别源码包和二进制包，例如.tar.gz、.zip、.rar 结尾的包通常称之为源码包，以.rpm 结尾的软件包称之为二进制包。真正区分是否为源码还是二进制还得基于代码里面的文件来判断，例如包含.h、.c、.cpp、.cc 等结尾的源码文件，称之为源码包，而代码代码里面存在 bin 可执行文件，称之为二进制包。

CentOS 操作系统中有一款默认软件管理的工具，红帽包管理工具（Red Hat Package Manager，RPM）。

使用 RPM 工具可以对软件包实现快速安装、管理及维护。RPM 管理工具适用的操作系统包括：CentOS，RedHat，Fedora，SUSE 等，RPM 工具常用于管理.rpm 后缀结尾的软件包。

RPM 软件包命令规则详解如下：

RPM 包命名格式为：

name-version.rpm

name-version-noarch.rpm

name-version-arch.src.rpm

如下软件包格式：

epel-release-6-8.noarch.rpm

perl-Pod-Plainer-1.03-1.el6.noarch.rpm

yasm-1.2.0-4.el7.x86_64.rpm

RPM 包格式解析如下：

- name 软件名称，例如 yasm、perl-pod-Plainer；
- version 版本号，1.2.0 通用格式：“主版本号.次版本号.修正号”；
  4 表示是发布版本号，该 RPM 包是第几次编译生成的；
- arch 适用的硬件平台，RPM 支持的平台有：i386、i586、i686、x86_64、sparc、alpha 等。
- .rpm 后缀包表示编译好的二进制包，可用 rpm 命令直接安装；
- .src.rpm 源代码包，源码编译生成.rpm 格式的 RPM 包方可使用；
- el* 软件包发行版本，el6 表示该软件包适用于 RHEL 6.x/CentOS 6.x；
- devel： 开发包；
- noarch： 软件包可以在任何平台上安装。

RPM 工具命令详解如下：

RPM 选项 PACKAGE_NAME

-a, --all 查询所有已安装软件包；

-q，--query 表示询问用户，输出信息；

-l, --list 打印软件包的列表；

-f, --file FILE 查询包含 FILE 的软件包；

-i, --info 显示软件包信息，包括名称，版本，描述；

-v, --verbose 打印输出详细信息；

-U, --upgrade 升级 RPM 软件包；

-h,--hash	软件安装，可以打印安装进度条；
--last	列出软件包时，以安装时间排序，最新的在上面；
-e, --erase	卸载 rpm 软件包
--force	表示强制，强制安装或者卸载；
--nodeps	RPM 包不依赖
-l, --list	列出软件包中的文件；
--provides	列出软件包提供的特性；
-R, --requires	列出软件包依赖的其他软件包；
--scripts	列出软件包自定义的小程序。

RPM 企业案例演示：

rpm -q	httpd	检查 httpd 包是否安装；
rpm -ql	httpd	查看软件安装的路径；
rpm -qi	httpd	查看软件安装的版本信息；
rpm -e	httpd	卸载 httpd 软件；
rpm -e --nodeps	httpd	强制卸载 httpd；
rpm -qa|grep	httpd	检查 httpd 相关的软件包是否安装。
rpm -ivh	httpd-2.4.10-el7.x86_64.rpm	安装 httpd 软件包；
rpm -Uvh	httpd-2.4.10-el7.x86_64.rpm	升级 httpd 软件；
rpm -ivh --nodeps	httpd-2.4.10-el7.x86_64.rpm	不依赖其他软件包；

6.2 Tar 命令参数详解

Linux 操作系统除了使用 RPM 管理工具对软件包管理之外，还可以通过 tar、zip、jar 等工具进行源码包的管理。

-A, --catenate, --concatenate	将存档与已有的存档合并
-c, --create	建立新的存档
-d, --diff, --compare	比较存档与当前文件的不同之处
--delete	从存档中删除
-r, --append	附加到存档结尾
-t, --list	列出存档中文件的目录
-u, --update	仅将较新的文件附加到存档中
-x, --extract, --get	解压文件
-j, --bzip2, --bunzip2	有 bz2 属性的软件包；
-z, --gzip, --ungzip	有 gz 属性的软件包；
-b, --block-size N	指定块大小为 Nx512 字节（缺省时 N=20)；
-B, --read-full-blocks	读取时重组块；
-C, --directory DIR	指定新的目录；
--checkpoint	读取存档时显示目录名；
-f, --file [HOSTNAME:]F	指定存档或设备，后接文件名称；
--force-local	强制使用本地存档，即使存在克隆；
-G, --incremental	建立老 GNU 格式的备份；

-g, --listed-incremental	建立新 GNU 格式的备份；
-h, --dereference	不转储动态链接，转储动态链接指向的文件；
-i, --ignore-zeros	忽略存档中的 0 字节块（通常意味着文件结束）；
--ignore-failed-read	在不可读文件中作 0 标记后再退出；
-k, --keep-old-files	保存现有文件；从存档中展开时不进行覆盖；
-K, --starting-file F	从存档文件 F 开始；
-l, --one-file-system	在本地文件系统中创建存档；
-L, --tape-length N	在写入 N*1024 个字节后暂停，等待更换磁盘；
-m, --modification-time	当从一个档案中恢复文件时，不使用新的时间标签；
-M, --multi-volume	建立多卷存档,以便在几个磁盘中存放；
-O, --to-stdout	将文件展开到标准输出；
-P, --absolute-paths	不要从文件名中去除 '/'；
-v, --verbose	详细显示处理的文件；
--version	显示 tar 程序的版本号；
--exclude	FILE 不把指定文件包含在内；
-X, --exclude-from FILE	从指定文件中读入不想包含的文件的列表。

6.3 TAR 企业案例演示

tar	-cvf	jfedu.tar.gz	jfedu 打包 jfedu 文件或者目录，打包后名称 jfedu.tar.gz；
tar	-tf	jfedu.tar.gz	查看 jfedu.tar.gz 包中内容；

tar -rf jfedu.tar.gz jfedu.txt 将 jfedu.txt 文件追加到 jfedu.tar.gz 中
tar -xvf jfedu.tar.gz 解压 jfedu.tar.gz 程序包；
tar -czvf jfedu.tar.gz jfedu 使用 gzip 格式打包并压缩 jfedu 目录；
tar -cjvf jfedu.tar.bz2 jfedu 使用 bzip2 格式打包并压缩 jfedu 目录；
tar -czf jfedu.tar.gz * -X list.txt 使用 gzip 格式打包并压当前目录所有文件，排除 list.txt 中记录的文件；
tar -czf jfedu.tar.gz * --exclude=zabbix-3.2.4.tar.gz --exclude=nginx-1.12.0.tar.gz 使用 gzip 格式打包并压当前目录所有文件及目录，排除 zabbix-3.2.4.tar.gz 和 nginx-1.12.0.tar.gz 软件包。

6.4 TAR 实现 Linux 操作系统备份

Tar 命令工具除了用于日常打包、解压源码包或者压缩包之外，最大的亮点是还可以用于 Linux 操作系统文件及目录的备份，使用 tar -g 可以基于 GNU 格式的增量备份，备份原理是基于检查目录或者文件的 atime、mtime、ctime 属性是否被修改。文件及目录时间属性详解如下：

- 文件被访问的时间(Access time,atime)；
- 文件内容被改变的时间（Modified time，mtime)；
- 文件写入、权限更改的时间（Change time，ctime)。

总结，更改文件内容 mtime 和 ctime 都会改变，但 ctime 可以在 mtime 未发生变化

时被更改，例如修改文件权限，文件 mtime 时间不变，而 ctime 时间改变。TAR 增量备份案例演示步骤如下：

（1） /root 目录创建 jingfeng 文件夹，同时在 jingfeng 文件夹中，新建 jf1.txt，jf2.txt 文件，如图 6-1 所示：

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# mkdir jingfeng
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# cd jingfeng/
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# pwd
/root/jingfeng
[root@www-jfedu-net jingfeng]# ls
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# touch jf1.txt jf2.txt
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# ll
total 0
-rw-r--r-- 1 root root 0 May  3 17:29 jf1.txt
-rw-r--r-- 1 root root 0 May  3 17:29 jf2.txt
[root@www-jfedu-net jingfeng]#
```

图 6-1 创建 jingfeng 目录及文件

（2） 使用 tar 命令第一次完整备份 jingfeng 文件夹中的内容，-g 指定快照 snapshot 文件，第一次没有该文件则会自动创建，如图 6-2 所示：

```
cd /root/jingfeng/
tar -g /data/backup/snapshot -czvf /data/backup/2017jingfeng.tar.gz
```

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# cd /root/jingfeng/
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# ls
jf1.txt  jf2.txt
[root@www-jfedu-net jingfeng]# tar  -g  /data/backup/snapshot  -czvf  /data/backu
g.tar.gz  *
jf1.txt
jf2.txt
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# ll /data/backup/
total 8
-rw-r--r-- 1 root root 126 May  3 17:37 2017jingfeng.tar.gz
-rw-r--r-- 1 root root  36 May  3 17:37 snapshot
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# cat /data/backup/snapshot
GNU tar-1.23-2
14938042225483817 99[root@www-jfedu-net jingfeng]#
```

图 6-2 tar 备份 jingfeng 目录中文件

（3） 使用 tar 命令第一次完整备份 jingfeng 文件夹中之后，会生成快照文件：/data/backup/snapshot，后期增量备份会以 snapshot 文件为参考，在 jingfeng 文件夹中再创建 jf3.txt jf4.txt 文件，然后通过 tar 命令增量备份 jingfeng 目录所有内容，如图 6-3 所示：

```
cd  /root/jingfeng/

touch jf3.txt jf4.txt

tar -g /data/backup/snapshot -czvf /data/backup/2017jingfeng_add1.tar.gz *
```

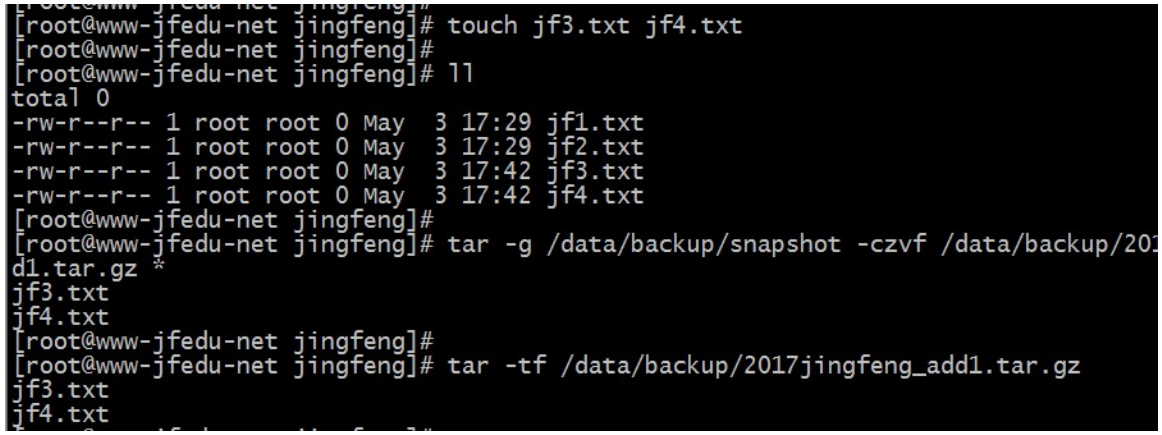


图 6-3 tar 增量备份 jingfeng 目录中文件

如上图 6-3 所示，增量备份时，需-g 指定第一次完整备份的快照 snapshot 文件，同时增量打包的文件名不能跟第一次备份后的文件名重复，通过 tar –tf 可以查看打包后的文件内容。

6.5 Shell+TAR 实现增量备份

企业中日常备份的数据包括/boot、/etc、/root、/data 目录等，备份的策略参考：每周 1-6 执行增量备份，每周日执行全备份。同时在企业中备份操作系统数据均使用 Shell 脚本完成，此处 auto_backup_system.sh 脚本供参考，后面章节会系统讲解 Shell 脚本，

脚本内容如下：

```
#!/bin/bash

#Automatic Backup Linux System Files

#By Author www.jfedu.net

#Define Variables

SOURCE_DIR=(

    $*

)

TARGET_DIR=/data/backup/

YEAR=`date +%Y`

MONTH=`date +%m`

DAY=`date +%d`

WEEK=`date +%u`

FILES=system_backup.tgz

CODE=$?

if

    [ -z $SOURCE_DIR ]; then

    echo -e "Please Enter a File or Directory You Need to Backup:\n----------\nExample $0 /boot /etc ....."

    exit

fi

#Determine Whether the Target Directory Exists
```

```
if
    [ ! -d $TARGET_DIR/$YEAR/$MONTH/$DAY ]; then
    mkdir -p $TARGET_DIR/$YEAR/$MONTH/$DAY
    echo "This $TARGET_DIR Created Successfully !"
fi
#EXEC Full_Backup Function Command
Full_Backup()
{
if
    [ "$WEEK" -eq "7" ]; then
    rm -rf $TARGET_DIR/snapshot
    cd $TARGET_DIR/$YEAR/$MONTH/$DAY ; tar -g $TARGET_DIR/snapshot
-czvf $FILES `echo ${SOURCE_DIR[@]}`
    [ "$CODE" == "0" ]&&echo -e
"-----------------------------------------\nFull_Backup System Files Backup
Successfully !"
fi
}
#Perform incremental BACKUP Function Command
Add_Backup()
{
   cd $TARGET_DIR/$YEAR/$MONTH/$DAY ;
```

```
if
    [ -f $TARGET_DIR/$YEAR/$MONTH/$DAY/$FILES ];  then
    read -p "$FILES Already Exists, overwrite confirmation yes or no ? : " SURE
    if [ $SURE == "no" -o $SURE == "n" ];  then
    sleep 1  ;  exit 0
    fi
#Add_Backup Files System
    if
        [ $WEEK -ne "7" ];  then
        cd  $TARGET_DIR/$YEAR/$MONTH/$DAY  ;  tar  -g
$TARGET_DIR/snapshot -czvf $$_$FILES `echo ${SOURCE_DIR[@]}`
        [  "$CODE"  ==  "0"  ]&&echo  -e
"----------------------------------------\nAdd_Backup  System  Files  Backup
Successfully !"
   fi
else
   if
       [ $WEEK -ne "7" ];  then
       cd $TARGET_DIR/$YEAR/$MONTH/$DAY  ;  tar -g $TARGET_DIR/snapshot
-czvf $FILES `echo ${SOURCE_DIR[@]}`
       [  "$CODE"  ==  "0"  ]&&echo  -e
"----------------------------------------\nAdd_Backup  System  Files  Backup
```

```
Successfully !"
    fi
fi
}
Full_Backup；Add_Backup
```

6.6 ZIP 软件包管理

ZIP 也是计算机文件的压缩的算法，原名 Deflate（真空），发明者为菲利普·卡兹（Phil Katz)），他于 1989 年 1 月公布了该格式的资料。ZIP 通常使用后缀名“.zip”。

主流的压缩格式包括 tar、rar、zip、war、gzip、bz2、iso 等。从性能上比较，TAR、WAR、RAR 格式较 ZIP 格式压缩率较高，但压缩时间远远高于 ZIP，Zip 命令行工具可以实现对 zip 属性的包进行管理，也可以将文件及文件及打包成 zip 格式。如下为 ZIP 工具打包常见参数详解：

-f	freshen：只更改文件；
-u	update：只更改或新文件；
-d	从压缩文件删除文件；
-m	中的条目移动到 zipfile（删除 OS 文件）；
-r	递归到目录；
-j	junk（不记录）目录名；
-l	将 LF 转换为 CR LF（-11 CR LF 至 LF）；
-1	压缩更快 1-9 压缩更好；
-q	安静操作，不输出执行的过程；

-v	verbose 操作/打印版本信息；
-c	添加一行注释；
-z	添加 zipfile 注释；
-o	读取名称使 zip 文件与最新条目一样旧；
-x	不包括以下名称；
-F	修复 zipfile（-FF 尝试更难）；
-D	不要添加目录条目；
-T	测试 zip 文件完整性；
-X	eXclude eXtra 文件属性；
-e	加密 - 不要压缩这些后缀；
-h2	显示更多的帮助。

ZIP 企业案例演示：

（1） 通过 zip 工具打包 jingfeng 文件夹中所有内容，如图 6-4 所示：

```
zip  -rv  jingfeng.zip /root/jingfeng/
```

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# zip  -rv  jingfeng.zip /root/jingfeng/
updating: root/jingfeng/          (in=0) (out=0) (stored 0%)
updating: root/jingfeng/jf4.txt (in=0) (out=0) (stored 0%)
updating: root/jingfeng/jf1.txt (in=0) (out=0) (stored 0%)
updating: root/jingfeng/jf2.txt (in=0) (out=0) (stored 0%)
updating: root/jingfeng/jf3.txt (in=0) (out=0) (stored 0%)
total bytes=0, compressed=0 -> 0% savings
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll jingfeng.zip
-rw-r--r-- 1 root root 858 May  3 18:16 jingfeng.zip
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# zip -T jingfeng
test of jingfeng.zip OK
[root@www-jfedu-net ~]#
```

图 6-4 zip 对 jingfeng 目录打包备份

（2） 通过 zip 工具打包 jingfeng 文件夹中所有内容，排除部分文件，如图 6-5 所示：

```
zip   -rv   jingfeng.zip   *   -x   jf1.txt

zip   -rv   jingfeng.zip   *   -x   jf2.txt -x jf3.txt
```

```
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# zip -rv jingfeng.zip * -x jf1.txt
excluding jf1.txt
file matches zip file -- skipping
updating: jf2.txt       (in=0) (out=0) (stored 0%)
updating: jf3.txt       (in=0) (out=0) (stored 0%)
total bytes=0, compressed=0 -> 0% savings
[root@www-jfedu-net jingfeng]#
[root@www-jfedu-net jingfeng]# zip -rv jingfeng.zip * -x jf2.txt -x jf3.txt
excluding jf2.txt
excluding jf3.txt
file matches zip file -- skipping
updating: jf1.txt       (in=0) (out=0) (stored 0%)
total bytes=0, compressed=0 -> 0% savings
[root@www-jfedu-net jingfeng]#
```

图 6-5 zip 对 jingfeng 目录打包备份，排除部分文件

（3） 通过 zip 工具删除 jingfeng.zip 中 jf3.txt 文件，如图 6-6 所示

```
zip jingfeng.zip -d jf3.txt
```

（4） 通过 unzip 工具解压 jingfeng.zip 文件夹中所有内容，如图 6-6 所示：

```
unzip   jingfeng.zip

unzip   jingfeng.zip   -d   /data/backup/  可以-d 指定解压后的目录
```

```
[root@www-jfedu-net ~]# ll jingfeng.zip
-rw-r--r-- 1 root root 858 May  3 18:16 jingfeng.zip
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# unzip jingfeng.zip
Archive:  jingfeng.zip
   creating: root/jingfeng/
 extracting: root/jingfeng/jf4.txt
 extracting: root/jingfeng/jf1.txt
 extracting: root/jingfeng/jf2.txt
 extracting: root/jingfeng/jf3.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# unzip jingfeng.zip -d /data/backup/
Archive:  jingfeng.zip
   creating: /data/backup/root/jingfeng/
 extracting: /data/backup/root/jingfeng/jf4.txt
 extracting: /data/backup/root/jingfeng/jf1.txt
 extracting: /data/backup/root/jingfeng/jf2.txt
 extracting: /data/backup/root/jingfeng/jf3.txt
```

图 6-6 unzip 对 jingfeng 目录解压

6.7 源码包软件安装

通常使用 RPM 工具管理.rpm 结尾的二进制包，而标准的.zip、tar 结尾的源代码包则不能使用 RPM 工具去安装、卸载及升级，源码包安装有三个步骤，如下：

- ./configure 预编译，主要用于检测系统基准环境库是否满足，生成 MakeFile 文件；
- make 编译，基于第一步生成的 makefile 文件，进行源代码的编译；
- make install 安装，编译完毕之后，将相关的可运行文件安装至系统中；

使用 make 编译时，Linux 操作系统必须有 GCC 编译器，用于编译源码。

源码包安装通常需要./configure、make、make install 三个步骤，某些特殊源码可以只有三步中的其中一个步骤，或者两个步骤。

以 CentOS 7 Linux 系统为基准，在其上安装 Nginx 源码包，企业中源码安装的详细步骤如下：

（1） Nginx.org 官网下载 Nginx-1.13.0.tar.gz 包

```
wget http://nginx.org/download/nginx-1.13.0.tar.gz
```

（2） Nginx 源码包解压

```
tar  -xvf nginx-1.13.0.tar.gz
```

（3） 源码 Configure 预编译，需进入解压后的目录执行./configure 指令，分号“；”表示连接多个命令。

```
cd nginx-1.13.0；./configure
```

（4） make 编译

```
make
```

（5） make install 安装

```
make    install
```

通过以上五个步骤，源码包软件安装成功，源码包在编译及安装时，可能会遇到各种错误，需要把错误解决之后，然后再进行下一步安装即可，后面章节会重点针对企业使用的软件进行案例演练。

6.8 YUM 软件包管理

前端软件包管理器（Yellow Updater Modified，YUM）适用于 CentOS、Fedora、RedHat 及 SUSE 中的 Shell 命令行，主要用于管理 RPM 包，于 RPM 工具使用范围类似，YUM 工具能够从指定的服务器自动下载 RPM 包并且安装，还可以自动处理依赖性关系。

使用 RPM 工具管理和安装软件时，会发现 rpm 包有依赖，需要逐个手动下载安装，而 YUM 工具的最大便利就是可以自动安装所有依赖的软件包，从而提升效率，节省时间。

6.9 YUM 工作原理

学习 YUM，一定要理解 YUM 工作原理，YUM 正常运行，需要依赖两个部分，一是 YUM 源端，二是 YUM 客户端，也即用户使用端。

YUM 客户端安装的所有 RPM 包都是来自 YUM 服务端，YUM 源端通过 HTTP 或者 FTP 服务器发布。而 YUM 客户端能够从 YUM 源端下载依赖的 RPM 包是由于在 YUM 源端生成了 RPM 包的基准信息，包括 RPM 包版本号、配置文件、二进制信息、依赖关系等。

YUM 客户端需要安装软件或者搜索软件，会查找/etc/yum.repos.d 下以.repo 结尾文件，CentOS Linux 默认的.repo 文件名为 CentOS-Base.repo，该文件中配置了 YUM 源端的镜像地址，所以每次安装、升级 RPM 包，YUM 客户端均会查找.repo 文件。

YUM 客户端如果配置了 CentOS 官方 repo 源，客户端操作系统必须能联外网，满足

网络条件，才能下载软件并安装，如果没有网络，也可以构建光盘源或者内部 YUM 源。在只要 YUM 客户端时，YUM 客户端安装软件，默认会把 YUM 源地址、Header 信息、软件包、数据库信息、缓存文件存储在/var/cache/yum 中，每次使用 YUM 工具，YUM 优先通过 Cache 查找相关软件包，Cache 中不存在，然后在访问外网 YUM 源。

6.10 YUM 企业案例演练

由于 YUM 工具的使用简便、快捷、高效，在企业中得到广泛的使用，得到众多 IT 运维、程序人员的青睐，要能熟练使用 YUM 工具，需要先掌握 YUM 命令行参数的使用，如下为 YUM 命令工具的参数详解及实战步骤：

YUM 命令工具指南，YUM 格式为：
YUM [command] [package] -y|-q 其中的[options]是可选。-y 安装或者卸载出现 YES 时，自动确认 yes；-q 不显示安装的过程。
yum install httpd 安装 httpd 软件包；
yum search YUM 搜索软件包；
yum list httpd 显示指定程序包安装情况 httpd；
yum list 显示所有已安装及可安装的软件包；
yum remove httpd 删除程序包 httpd；
yum erase httpd 删除程序包 httpd；
yum update 内核升级或者软件更新；
yum update httpd 更新 httpd 软件；
yum check-update 检查可更新的程序；
yum info httpd 显示安装包信息 httpd；

yum provides	列出软件包提供哪些文件；
yum provides "*/rz"	列出 rz 命令由哪个软件包提供；
yum grouplist	查询可以用 groupinstall 安装的组名称；
yum groupinstall "Chinese Support"	安装中文支持；
yum groupremove "Chinese Support"	删除程序组 Chinese Support；
yum deplist httpd	查看程序 httpd 依赖情况；
yum clean packages	清除缓存目录下的软件包；
yum clean headers	清除缓存目录下的 headers；
yum clean all	清除缓存目录下的软件包及旧的 headers。

（1） 基于 CentOS 7 Linux，执行命令 yum install httpd -y，安装 httpd 服务，如图 6-7 所示：

```
[root@www-jfedu-net ~]# yum install httpd -y
Loaded plugins: fastestmirror, langpacks
Repository epel is listed more than once in the configuration
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
---> Package httpd.x86_64 0:2.4.6-45.el7.centos.4 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package          Arch              Version                          Reposi
================================================================================
Installing:
 httpd            x86_64            2.4.6-45.el7.centos.4            update

Transaction Summary
================================================================================
Install  1 Package
```

图 6-7 YUM 安装 httpd 软件

（2） 执行命令 yum grouplist，检查 groupinstall 的软件组名，如图 6-8 所示：

```
[root@www-jfedu-net ~]# yum grouplist|more
Loaded plugins: fastestmirror, langpacks
Repository epel is listed more than once in the configuration
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)
Loading mirror speeds from cached hostfile
Available Environment Groups:
   Minimal Install
   Compute Node
   Infrastructure Server
   File and Print Server
   MATE Desktop
   Basic Web Server
   Virtualization Host
   Server with GUI
   GNOME Desktop
   KDE Plasma Workspaces
   Development and Creative Workstation
```

图 6-8 YUM Grouplist 显示组安装名称

（3） 执行命令 yum groupinstall "GNOME Desktop" -y，安装 Linux 图像界面，如图 6-9 所示:

```
[root@www-jfedu-net ~]# yum groupinstall "GNOME Desktop" -y
Loaded plugins: fastestmirror, langpacks
Repository epel is listed more than once in the configuration
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)
Loading mirror speeds from cached hostfile
Resolving Dependencies
--> Running transaction check
---> Package ModemManager.x86_64 0:1.6.0-2.el7 will be installed
--> Processing Dependency: libqmi-utils for package: ModemManager-1.6.0-2.
--> Processing Dependency: libmbim-utils for package: ModemManager-1.6.0-2
--> Processing Dependency: libqmi-glib.so.5()(64bit) for package: ModemMan
_64
--> Processing Dependency: libmbim-glib.so.4()(64bit) for package: ModemMa
6_64
---> Package NetworkManager-adsl.x86_64 1:1.4.0-19.el7_3 will be installed
```

图 6-9 GNOME Desktop 图像界面安装

（4） 执行命令 yum install httpd php php-devel php-mysql mariadb mariadb-server -y，安装中小企业 LAMP 架构环境，如图 6-10 所示:

```
[root@www-jfedu-net ~]# yum install httpd php php-devel php-mysql mariadb m
Loaded plugins: fastestmirror, langpacks
Repository epel is listed more than once in the configuration
Loading mirror speeds from cached hostfile
Package httpd-2.4.6-45.el7.centos.4.x86_64 already installed and latest ver
Package php-5.4.16-42.el7.x86_64 already installed and latest version
Package php-devel-5.4.16-42.el7.x86_64 already installed and latest version
Package php-mysql-5.4.16-42.el7.x86_64 already installed and latest version
Package 1:mariadb-5.5.52-1.el7.x86_64 already installed and latest version
Resolving Dependencies
--> Running transaction check
---> Package mariadb-server.x86_64 1:5.5.52-1.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved
```

图 6-10 LAMP 中小企业架构安装

（5） 执行命令 yum remove ntpdate -y，卸载 ntpdate 软件包，如图 6-11 所示:

```
[root@www-jfedu-net ~]# yum  remove  ntpdate -y
Loaded plugins: fastestmirror, langpacks
Repository epel is listed more than once in the configuration
Resolving Dependencies
--> Running transaction check
---> Package ntpdate.x86_64 0:4.2.6p5-25.el7.centos.2 will be erased
--> Processing Dependency: ntpdate = 4.2.6p5-25.el7.centos.2 for package:
entos.2.x86_64
--> Running transaction check
---> Package ntp.x86_64 0:4.2.6p5-25.el7.centos.2 will be erased
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package          Arch           Version                                 Repo
--------------------------------------------------------------------------------
```

图 6-11 卸载 NTPDATE 软件

（6） 执行命令 yum provides rz 或者 yum provides "*/rz"，查找 rz 命令的提供者，

如图 6-12 所示:

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# yum provides rz
Loaded plugins: fastestmirror, langpacks
Repository epel is listed more than once in the configuration
Loading mirror speeds from cached hostfile
lrzsz-0.12.20-36.el7.x86_64 : The lrz and lsz modem communications program
Repo        : os
Matched from:
Filename    : /usr/bin/rz


[root@www-jfedu-net ~]#
```

图 6-12 查找 RZ 命令的提供者

（7） 执行命令 yum update -y，升级 Linux 所有可更新的软件包或 Linux 内核升级，

如图 6-13 所示：

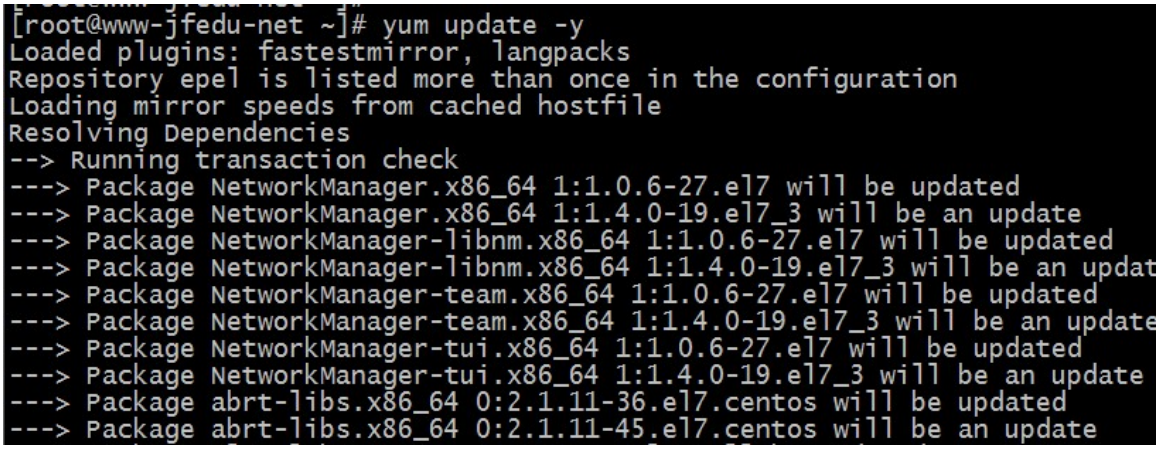


图 6-13 软件包升级或内核升级

6.11 YUM 优先级配置实战

基于 YUM 安装软件时,通常会配置多个 Repo 源，而 Fastest mirror 插件是为拥有多个镜像的软件库配置文件而设计的。它会连接到每一个镜像，计算连接所需的时间，然后将镜像按快到慢排序供 YUM 应用。

默认 CentOS Linux 系统，Fastestmirror 插件是开启的，所以安装软件会从最快的镜像源安装，但是由于 Repo 源很多，而在这些源中都存在某些软件包,但有些软件有重复,甚至冲突,能否可以优先从一些 Repo 源中去查找，如果找不到，再去其他源中找呢?

可以使用 YUM 优先级插件解决该问题，YUM 提供的插件 yum-plugin-priorities，直接 YUM 安装即可，命令如下：

yum install -y yum-plugin-priorities

```
Installed size: 28 k
Downloading packages:
yum-plugin-priorities-1.1.31-46.el7_5.noarch.rpm
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : yum-plugin-priorities-1.1.31-46.el7_5.noarch
  Verifying  : yum-plugin-priorities-1.1.31-46.el7_5.noarch

Installed:
  yum-plugin-priorities.noarch 0:1.1.31-46.el7_5

Complete!
```

修改 YUM 源优先级配置文件，设置为 Enabled，开启优先级插件，1 为开启，0 为禁止；

vim /etc/yum/pluginconf.d/priorities.conf

enabled = 1

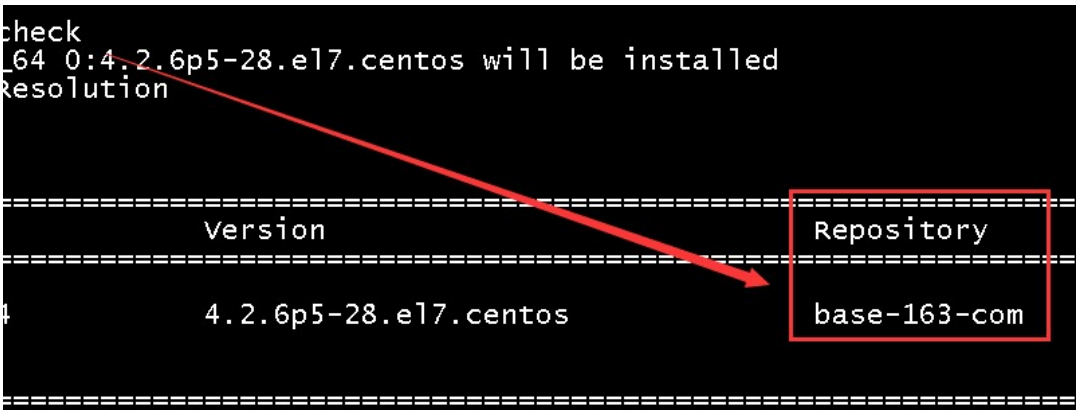
```
root@www-jfedu-net ~]# vim /etc/yum
yum/          yum.conf      yum.repos.d/
[root@www-jfedu-net ~]# vim /etc/yum/
fssnap.d/             protected.d/          version-groups.c
pluginconf.d/         vars/
[root@www-jfedu-net ~]# vim /etc/yum/pluginconf.d/
fastestmirror.conf  langpacks.conf      priorities.conf
[root@www-jfedu-net ~]# vim /etc/yum/pluginconf.d/prioriti
[main]
enabled = 1
~
~
```

vim 修改/etc/yum.repos./xx.repo 文件，在 base 段中加入如下指令：（优先级为 1 表示优先被查找，越大其反而被后续查找）

priority=1

```
# remarked out baseurl= line instead.
#
#
[base-163-com]
name=CentOS-$releasever - Base - 163.com
#mirrorlist=http://mirrorlist.centos.org/?release=$release
baseurl=http://mirrors.163.com/centos/$releasever/os/$base
gpgcheck=1
priority=1
gpgkey=http://mirrors.163.com/centos/RPM-GPG-KEY-CentOS-7

#released updates
[updates]
```

基于 YUM 安装 ntpdate 软件，测试已经优先从 163 源中查找；



6.12 基于 ISO 镜像构建 YUM 本地源

通常而言，YUM 客户端使用前提是必须联外网，YUM 安装软件时，检查 repo 配置文件查找相应的 YUM 源仓库，企业 IDC 机房很多服务器为了安全起见，是禁止服务器上外网的，所以不能使用默认的官方 YUM 源仓库。

构建本地 YUM 光盘源，其原理是通过查找光盘中的软件包，实现 YUM 安装，配置步骤如下：

（1） 将 CentOS-7-x86_64-DVD-1511.iso 镜像加载至虚拟机 CD/DVD 或者放入服务器 CD/DVD 光驱中，并将镜像文件挂载至服务器/mnt 目录，如图 6-14 所示，挂载命令：

```
mount        /dev/cdrom      /mnt/
```

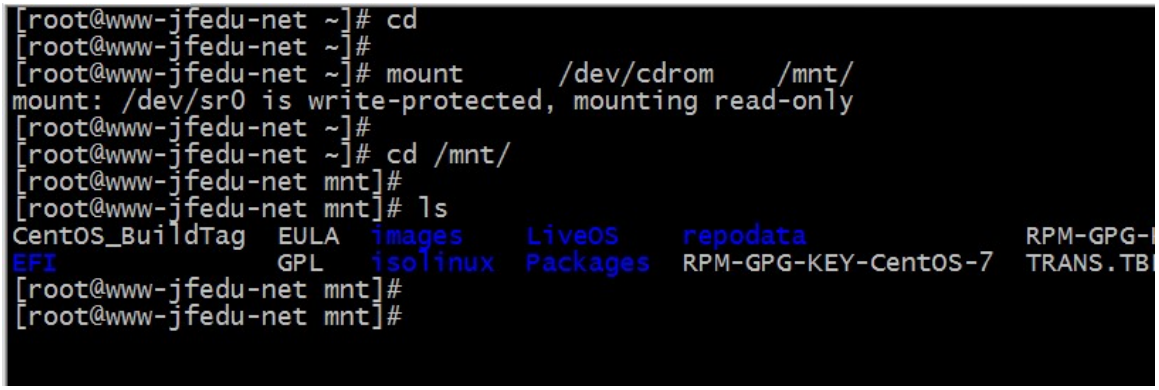


图 6-14 CentOS ISO 镜像文件挂载

（2） 备份/etc/yum.repos.d/CentOS-Base.repo 文件为 CentOS-Base.repo.bak，同时在/etc/yum.repos.d 目录下创建 media.repo 文件，并写入如下内容：

```
[yum]
name=CentOS7
baseurl=file:///mnt
enabled=1
gpgcheck=1
gpgkey=file:///mnt/RPM-GPG-KEY-CentOS-7
```

Media.repo 配置文件详解：

```
name=CentOS7                                    YUM 源显示名称；
baseurl=file:///mnt                             ISO 镜像挂载目录；
gpgcheck=1                                      是否检查 GPG-KEY；
enabled=1                                       是否启用 YUM 源；
gpgkey=file:///mnt/RPM-GPG-KEY-CentOS-7         指定载目录下的 GPG-KEY 文件验证。
```

（3） 运行命令 yum clean all 清空 YUM Cache，执行 yum install screen –y 安装 screen 软件如图 6-15 所示：

```
[root@www-jfedu-net ~]# yum clean all
Loaded plugins: fastestmirror
Cleaning repos: yum
Cleaning up everything
Cleaning up list of fastest mirrors
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# yum install screen -y
Loaded plugins: fastestmirror
yum
(1/2): yum/group_gz
(2/2): yum/primary_db
Determining fastest mirrors
Resolving Dependencies
--> Running transaction check
---> Package screen.x86_64 0:4.1.0-0.21.20120314git3c2946.el7 will be insta
--> Finished Dependency Resolution

Dependencies Resolved
```

图 6-15　YUM 安装 Screen 软件

（4）　至此 YUM 光盘源构建完毕，在使用 YUM 源时，会遇到部分软件无法安装，原因是因为光盘中软件包不完整导致，同时光盘源只能本机使用，其他局域网服务器无法使用。

6.13 基于 HTTP 构建 YUM 网络源

YUM 光盘源默认只能本机使用，局域网其他服务器无法使用 YUM 光盘源，如果想使用的话，需要在每台服务器上构建 YUM 本地源，该方案在企业中不可取，所以需要构建 HTTP 局域网 YUM 源解决，可以通过 CreateRepo 创建本地 YUM 源端，repo 即为 Repository。

构建 HTTP 局域网 YUM 源方法及步骤如下：

（1）　挂载光盘镜像文件至/mnt

```
mount　　/dev/cdrom　　/mnt/
```

（2）　拷贝/mnt/Packages 目录下所有软件包至/var/www/html/centos/

```
mkdir　-p　/var/www/html/centos/

cp　　-R　　/mnt/Packages/*　/var/www/html/centos/
```

（3） 使用 Createrepo 创建本地源，执行如下命令会在 Centos 目录生成 repodata 目录，目录内容如图 6-16 所示：

```
yum install createrepo* -y

cd /var/www/html

createrepo centos/
```

```
[root@www-jfedu-net html]# pwd
/var/www/html
[root@www-jfedu-net html]#
[root@www-jfedu-net html]# cd centos/
[root@www-jfedu-net centos]#
[root@www-jfedu-net centos]#
[root@www-jfedu-net centos]# cd repodata/
[root@www-jfedu-net repodata]#
[root@www-jfedu-net repodata]# ls
40bac61f2a462557e757c2183511f57d07fba2c0dd63f99b48f0b466b7f2b8d2-other.xml.gz
4cdf96c7618bdbb2dfa543c29496faf76d940f0c04f316c8a3476426c65c81cf-primary.sqlite.bz2
9710c85f1049b4c60c74ae5fd51d3e98e4ecd50a43ab53ff641690fb164a6d63-other.sqlite.bz2
cfa741341d5d270d5b42d6220e2908d053c39a2d8346986bf48cee360e6f7ce8-filelists.xml.gz
d216d0fba4167a2d086c80ac8c708670a7e45ee3295f27ac2702784fc56623b4-primary.xml.gz
d863fcc08a4e8d47382001c3f22693ed77e03815a76cedf34d8256d4c12f6f0d-filelists.sqlite.bz2
repomd.xml
```

图 6-16 Createrepo 生成 repodata 目录

（4） 利用 HTTP 发布 YUM 本地源

本地 YUM 源通过 CreateRepo 搭建完毕，需要借助 HTTP WEB 服务器发布/var/www/html/centos/中所有软件，YUM 或者 RPM 安装 HTTP WEB 服务器，并启动 httpd 服务。

```
yum install httpd httpd-devel -y 安装 HTTP WEB 服务；

useradd apache -g apache 创建 apache 用户和组；

systemctl restart httpd.service 重启 HTTPD 服务；

setenforce 0 临时关闭 SeLinux 应用级安全策略；

systemctl stop firewalld.service 停止防火墙；

ps -ef |grep httpd 查看 HTTPD 进程是否启动。
```

（5） 在 YUM 客户端，创建/etc/yum.repos.d/http.repo 文件，写入如下内容：

```
[base]
```

```
name="CentOS7 HTTP YUM"

baseurl=http://192.168.1.115/centos/

gpgcheck=0

enabled=1

[updates]

name="CentOS7 HTTP YUM"

baseurl=http://192.168.1.115/centos

gpgcheck=0

enabled=1
```

（6） 至此在 YUM 客户端上执行如下命令，如图 6-17 所示：

yum clean all	清空 YUM Cache；
yum install ntpdate -y	安装 NTPDATE 软件。

```
[root@www-jfedu-net ~]# yum clean all
Loaded plugins: fastestmirror
Cleaning repos: base updates
Cleaning up everything
Cleaning up list of fastest mirrors
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# yum install ntpdate -y
Loaded plugins: fastestmirror
base
updates
(1/2): base/primary_db
(2/2): updates/primary_db
Determining fastest mirrors
Resolving Dependencies
--> Running transaction check
```

图 6-17 HTTP YUM 源客户端验证

第7章 Linux 磁盘管理实战篇

Linux 系统一切以文件的方式存储于硬盘，应用程序数据需要时刻读写硬盘，所以企业

生产环境中对硬盘的操作变得尤为重要，对硬盘的维护和管理也是每个运维工程师必备工作之一。

本章向读者介绍硬盘简介、硬盘数据存储方式、如何在企业生产服务器添加硬盘、对硬盘进行分区、初始化以及对硬盘进行故障修复等。

7.1 计算机硬盘简介

硬盘是计算机主要存储媒介之一，由一个或者多个铝制或者玻璃制的碟片组成，碟片外覆盖有铁磁性材料，硬盘内部由磁道、柱面、扇区、磁头等部件组成，如图 7-1 所示：

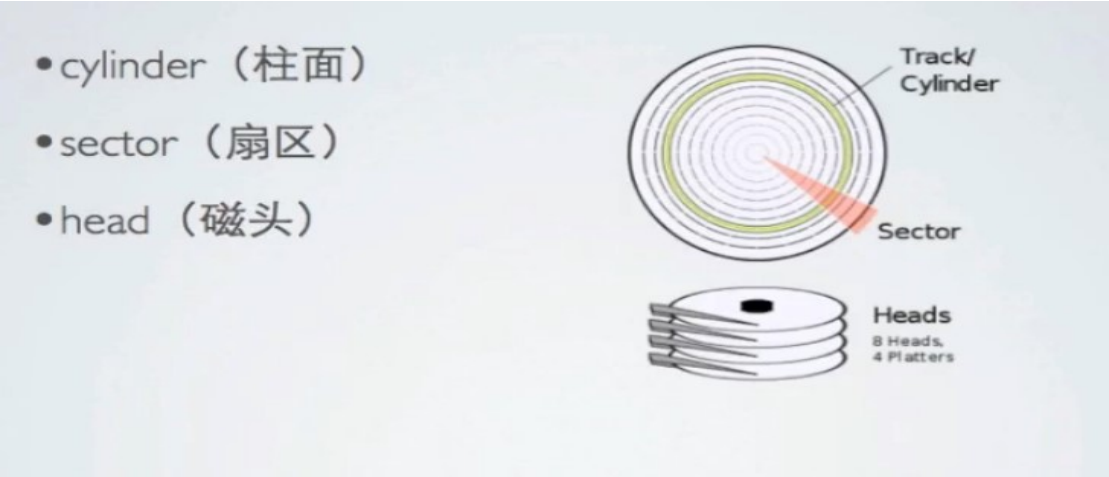


图 7-1 硬盘内部结构组成

Linux 系统中硬件设备相关配置文件存放在/dev/下，常见硬盘命名：/dev/hda、/dev/sda、/dev/sdb、/dev/sdc、/dev/vda。不同硬盘接口，在系统中识别的设备名称不一样。

IDE 硬盘接口在 Linux 中设备名为/dev/hda，SAS、SCSI、SATA 硬盘接口在 Linux 中设备名为 sda，高效云盘硬盘接口会识别为/dev/vda 等。

文件储存在硬盘上，硬盘的最小存储单位叫做 Sector（扇区），每个 Sector 储存 512 字节。操作系统在读取硬盘的时候，不会逐个 Sector 的去读取，这样效率非常低，为了提

升读取效率，操作系统会一次性连续读取多个 Sector，即一次性读取多个 Sector 称为一个 Block（块）。

由多个 Sector 组成的 Block 是文件存取的最小单位。Block 的大小常见的有 1KB、2KB、4KB，Block 在 Linux 中常设置为 4KB，即连续八个 Sector 组成一个 Block。

/boot 分区 Block 一般为 1KB，而/data/分区或者/分区的 Block 为 4K。可以通过如下三种方法查看 Linux 分区的 Block 大小：

```
dumpe2fs /dev/sda1 |grep "Block size"

tune2fs -l /dev/sda1 |grep "Block size"

stat /boot/|grep "IO Block"
```

例如创建一个普通文件，文件大小为 10Bytes，而默认设置 Block 为 4K，如果有 1 万个小文件，由于每个 Block 只能存放一个文件，如果文件的大小比 Block 大，会申请更多的 Block，相反如果文件的大小比默认 Block 小，仍会占用一个 Block，这样剩余的空间会被浪费掉。

- 1 万个文件理论只占用空间大小：10000x10=100000Bytes=97.65625MBytes；
- 1 万个文件真实占用空间大小：

  10000x4096Bytes=40960000Bytes=40000MBytes=40GB。
- 根据企业实际需求，此时可以将 Block 设置为 1K，从而节省更多的空间。

7.2 硬盘 Block 及 Inode 详解

通常而言，操作系统对于文件数据的存放包括两个部分：文件内容、权限及文件属性。操作系统文件存放是基于文件系统，文件系统会将文件的实际内容存储到 Block 中，而将权限与属性等信息存放至 Inode 中。

在硬盘分区中，还有一个超级区块（SuperBlock），SuperBlock 会记录整个文件系统的整体信息，包括 Inode、Block 总量、使用大小、剩余大小等信息，每个 inode 与 block 都有编号对应，方便 Linux 系统快速定位查找文件。

- Superblock：记录文件系统的整体信息，包括 inode 与 block 的总量、使用大小、剩余大小， 以及文件系统的格式与相关信息等；
- Inode：记录文件的属性，权限，同时会记录该文件的数据所在的 block 编号；
- Block：存储文件的内容，如果文件超过默认 Block 大小，会自动占用多个 Block。

因为每个 inode 与 block 都有编号，而每个文件都会占用一个 inode ，inode 内则有文件数据放置的 block 号码。如果能够找到文件的 inode，就可以找到该文件所放置数据的 block 号码，从而读取该文件内容。

操作系统进行格式化分区时，操作系统自动将硬盘分成两个区域。一个是数据 Block 区，用于存放文件数据；另一个是 Inode Table 区，用于存放 inode 包含的元信息。

每个 inode 节点的大小，可以在格式化时指定，默认为 128Bytes 或 256Bytes，/boot 分区 Inode 默认为 128Bytes，其他分区默认为 256Bytes，查看 Linux 系统 Inode 方法如下：

```
dumpe2fs     /dev/sda1 |grep " Inode size "

tune2fs    -l    /dev/sda1 |grep " Inode size"

stat    /boot/|grep "Inode"
```

格式化磁盘时，可以指定默认 Inode 和 Block 的大小，-b 指定默认 Block 值，-I 指定默认 Inode 值，如图 7-2 所示，命令如下：

```
mkfs.ext4 -b 4096 -I 256 /dev/sdb
```

```
[root@www-jfedu-net ~]# mkfs.ext4 -b 4096 -I 256 /dev/sdb
mke2fs 1.42.9 (28-Dec-2013)
/dev/sdb is entire device, not just one partition!
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
2621440 inodes, 10485760 blocks
524288 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2157969408
320 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
```

图 7-2 格式化硬盘指定 Inode 和 Block

7.3 硬链接介绍

一般情况下，文件名和 inode 编号是一一对应的关系，每个 inode 号码对应一个文件名。但 UNIX/Linux 系统多个文件名也可以指向同一个 inode 号码。这意味着可以用不同的文件名访问同样的内容，对文件内容进行修改，会影响到所有文件名。但删除一个文件名，不影响另一个文件名的访问。这种情况就被称为硬链接（hard link）。

创建硬链接的命令为：ln　jf1.txt　jf2.txt，其中 jf1.txt 为源文件，jf2.txt 为目标文件。如上命令源文件与目标文件的 inode 号码相同，都指向同一个 inode。inode 信息中有一项叫做"链接数"，记录指向该 inode 的文件名总数，这时会增加 1，变成 2，如图 7-3 所示：

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# touch jf1.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll jf1.txt
-rw-r--r-- 1 root root 0 May  5 17:19 jf1.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ln jf1.txt jf2.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll jf1.txt
-rw-r--r-- 2 root root 0 May  5 17:19 jf1.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll jf2.txt
-rw-r--r-- 2 root root 0 May  5 17:19 jf2.txt
[root@www-jfedu-net ~]#
```

图 7-3 jf1.txt jf2.txt 硬链接 inode 值变化

同样删除一个 jf2.txt 文件，就会使得 jf1.txt inode 节点中的"链接数"减 1。如果该 inode

值减到 0，表明没有文件名指向这个 inode，系统就会回收这个 inode 号码，以及其所对应 block 区域，如图 7-4 所示：

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll jf1.txt jf2.txt
-rw-r--r-- 2 root root 0 May  5 17:19 jf1.txt
-rw-r--r-- 2 root root 0 May  5 17:19 jf2.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# rm -rf jf2.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# ll jf1.txt
-rw-r--r-- 1 root root 0 May  5 17:19 jf1.txt
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
```

图 7-4 删除 jf2.txt 硬链接 inode 值变化

实用小技巧：硬链接不能跨分区链接，硬链接只能对文件生效，对目录无效，也即是目录不能创建硬链接。硬链接源文件与目标文件共用一个 inode 值，从某种意义上来，节省 inode 空间。不管是单独删除源文件还是删除目标文件，文件内容始终存在。同时链接后的文件不占用系统多余的空间。

7.4 软链接介绍

除了硬链接以外，还有一种链接-软链接。文件 jf1.txt 和文件 jf2.txt 的 inode 号码虽然不一样，但是文件 jf2.txt 的内容是文件 jf1.txt 的路径。读取文件 jf2.txt 时，系统会自动将访问者导向文件 jf1.txt。

无论打开哪一个文件，最终读取的都是文件 jf1.txt。这时，文件 jf2.txt 就称为文件 jf1.txt 的"软链接"（soft link）或者"符号链接（symbolic link）。

文件 jf2.txt 依赖于文件 jf1.txt 而存在，如果删除了文件 jf1.txt，打开文件 jf2.txt 就会报错："No such file or directory"。

软链接与硬链接最大的不同是文件 jf2.txt 指向文件 jf1.txt 的文件名，而不是文件 jf1.txt

的 inode 号码，因此文件 jf1.txt 的 inode 链接数不会发生变化，如图 7-5 所示：

```
[root@localhost ~]# ls -li jf1.txt
790403 -rw-r--r-- 1 root root 0 May  5 17:50 jf1.txt
[root@localhost ~]#
[root@localhost ~]# ln -s jf1.txt jf2.txt
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# ll -li jf1.txt jf2.txt
790403 -rw-r--r-- 1 root root 0 May  5 17:50 jf1.txt
795230 lrwxrwxrwx 1 root root 7 May  5 17:50 jf2.txt -> jf1.txt
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# rm -rf jf1.txt
[root@localhost ~]#
[root@localhost ~]# ll -li jf2.txt
795230 lrwxrwxrwx 1 root root 7 May  5 17:50 jf2.txt -> jf1.txt
[root@localhost ~]#
```

图 7-5 删除 jf1.txt 源文件链接数不变

实用小技巧：软链接可以跨分区链接，软链接支持目录同时也支持文件的链接。软链接源文件与目标文件 Inode 不相同，从某种意义上来，会消耗省 inode 空间。不管是删除源文件还是重启系统，该软链接还存在，但是文件内容会丢失，一旦新建源同名文件名，软链接文件恢复正常。

7.5 磁盘实战操作命令

企业真实场景由于硬盘常年大量读写，经常会出现坏盘，需要更换硬盘。或者由于磁盘空间不足，需添加新硬盘，新添加的硬盘需要经过格式化、分区才能被 Linux 系统所使用，虚拟机 CentOS 7 Linux 模拟 DELL R730 真实服务器添加一块新硬盘，不需要关机，直接插入用硬盘即可，一般硬盘均支持热插拔功能。企业中添加新硬盘的操作流程如下：

（1） 检测 Linux 系统识别的硬盘设备，新添加硬盘被识别为/dev/sdb，如果有多块硬盘，会依次识别成/dev/sdc、/dev/sdd 等设备名称，如图 7-6 所示：

```
fdisk  -l
```

```
[root@www-jfedu-net ~]# fdisk -l

Disk /dev/sda: 42.9 GB, 42949672960 bytes, 83886080 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x00077233

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      411647      204800   83  Linux
/dev/sda2          411648     1460223      524288   82  Linux swap / Solar
/dev/sda3         1460224    78217215    38378496   83  Linux

Disk /dev/sdb: 42.9 GB, 42949672960 bytes, 83886080 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

图 7-6 Fdisk 查看 Linux 系统硬盘设备

（2） 基于新硬盘/dev/sdb 设备，创建磁盘分区/dev/sdb1，如图 7-7 所示：

```
fdisk /dev/sdb
```

```
[root@www-jfedu-net ~]# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x3e5bdbfe.

Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   g   create a new empty GPT partition table
   G   create an IRIX (SGI) partition table
   l   list known partition types
   m   print this menu
```

图 7-7 Fdisk /dev/sdb 分区

（3） fdisk 分区命令参数如下，常用参数包括 m、n、p、e、d、w。

b	编辑 bsd disklabel；
c	切换 dos 兼容性标志；
d	删除一个分区；
g	创建一个新的空 GPT 分区表；

G	创建一个 IRIX（SGI）分区表；
l	列出已知的分区类型；
m	打印帮助菜单；
n	添加一个新分区；
o	创建一个新空 DOS 分区表；
p	打印分区表信息；
q	退出而不保存更改；
s	创建一个新的空的 Sun 磁盘标签；
t	更改分区的系统 ID；
u	更改显示/输入单位；
v	验证分区表；
w	将分区表写入磁盘并退出；
x	额外功能。

（4） 创建/dev/sdb1 分区方法，fdisk /dev/sdb，然后按 n-p-1-Enter 键- +20G-Enter 键-w，最后执行 fdisk –l|tail -10，如图 7-8（a）、图 7-8（b）所示：

```
[root@www-jfedu-net ~]# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x3cf5bf9c.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-83886079, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-83886079, default 83886079): +20G
Partition 1 of type Linux and of size 20 GiB is set

Command (m for help): w
```

图 7-8（a） Fdisk /dev/sdb 创建/dev/sdb1 分区

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# fdisk -l|tail -10

Disk /dev/sdb: 42.9 GB, 42949672960 bytes, 83886080 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x3cf5bf9c

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1            2048    41945087    20971520   83  Linux
[root@www-jfedu-net ~]#
```

图 7-8（b） Fdisk –l 查看/dev/sdb1 分区

（5） mkfs.ext4 /dev/sdb1 格式化磁盘分区，如图 7-9 所示：

```
[root@www-jfedu-net ~]# mkfs.ext4 /dev/sdb1
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
1310720 inodes, 5242880 blocks
262144 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=2153775104
160 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 265
        4096000
```

图 7-9 mkfs.ext4 格式化磁盘分区

（6） /dev/sdb1 分区格式化，使用 mount 命令挂载到/data/目录，如图 7-10 所示：

mkdir -p /data/	创建/data/数据目录
mount /dev/sdb1 /data	挂载/dev/sdb1 分区至/data/目录
df -h	查看磁盘分区详情
echo "mount /dev/sdb1 /data" >>/etc/rc.local	将挂载分区命令加入/etc/rc.local 开机启动

```
[root@www-jfedu-net ~]# mkdir -p /data/
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# mount  /dev/sdb1   /data
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        37G  5.7G   31G  16% /
devtmpfs        484M     0  484M   0% /dev
tmpfs           493M     0  493M   0% /dev/shm
tmpfs           493M   13M  480M   3% /run
tmpfs           493M     0  493M   0% /sys/fs/cgroup
/dev/sda1       197M  103M   94M  53% /boot
tmpfs            99M     0   99M   0% /run/user/0
/dev/sdb1        20G   45M   19G   1% /data
[root@www-jfedu-net ~]# echo "mount  /dev/sdb1   /data" >>/etc/rc.local
```

图 7-10　MOUNT 挂载/dev/sdb1 磁盘分区

（7）　自动挂载分区除了可以加入到/etc/rc.local 开机启动之外，还可以加入到 /etc/fstab 文件中，如图 7-11 所示：

/dev/sdb1		/data/	ext4	defaults
0 0				
mount	-o	rw,remount	/	重新挂载/系统，检测/etc/fstab 是否有误。

```
[root@www-jfedu-net ~]# vim /etc/fstab
# /etc/fstab
# Created by anaconda on Sat Aug 20 13:10:05 2016
#
# Accessible filesystems, by reference, are maintained under '/dev/disk'
# See man pages fstab(5), findfs(8), mount(8) and/or blkid(8) for more info
#
UUID=01581e1f-c36c-4968-a5b8-7b570e424f1c /                       xfs     defaults
UUID=36ff9bf6-e8a4-48da-9bba-b9c7a89fe151 /boot                   xfs     defaults
UUID=fae320d7-1e4c-487d-87ea-8a79e1d0885a swap                    swap    defaults
/dev/sdb1                                 /data/                  ext4    defaults
~
```

图 7-11 /dev/sdb1 磁盘分区加入/etc/fstab 文件

7.6　基于 GPT 格式磁盘分区

MBR 分区标准决定了 MBR 只支持在 2TB 以下的硬盘，为了支持能使用大于 2T 硬盘空间，需使用 GPT 格式进行分区。创建大于 2TB 的分区，需使用 parted 工具。

在企业真实环境中，通常一台服务器有多块硬盘，整个硬盘容量为 10T，需要基于 GTP

格式对 10T 硬盘进行分区，操作步骤如下：

parted -s	/dev/sdb mklabel gpt	设置分区类型为 gpt 格式；
mkfs.ext3	/dev/sdb	基于 Ext3 文件系统类型格式化；
mount	/dev/sdb /data/	挂载/dev/sdb 设备至/data/目录。

（1） 如图 7-12 所示，假设/dev/sdb 为 10T 硬盘，使用 GPT 格式来格式化磁盘：

```
/dev/sda3               536        6528    48126976

Disk /dev/sdb: 53.7 GB, 53687091200 bytes
255 heads, 63 sectors/track, 6527 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 b
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

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

图 7-12 假设/dev/sdb 为 10T 设备

（2） 执行命令：parted -s /dev/sdb mklabel gpt，如图 7-13 所示：

```
[root@localhost ~]# parted -s /dev/sdb  mklabel gpt
[root@localhost ~]#
[root@localhost ~]# fdisk -l |tail

WARNING: GPT (GUID Partition Table) detected on '/dev/sdb'! The
 Use GNU Parted.


Disk /dev/sdb: 53.7 GB, 53687091200 bytes
255 heads, 63 sectors/track, 6527 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1        6528    52428799+  ee  GPT
[root@localhost ~]#
```

图 7-13 设置/dev/sdb 为 GPT 格式磁盘

（3） 基于 mkfs.ext3 /dev/sdb 格式化磁盘，如图 7-14 所示：

```
[root@localhost ~]# mkfs.ext3 /dev/sdb
mke2fs 1.41.12 (17-May-2010)
/dev/sdb is entire device, not just one partition!
无论如何也要继续? (y,n) y
文件系统标签=
操作系统:Linux
块大小=4096 (log=2)
分块大小=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
3276800 inodes, 13107200 blocks
655360 blocks (5.00%) reserved for the super user
第一个数据块=0
Maximum filesystem blocks=4294967296
```

图 7-14　格式/dev/sdb 磁盘

parted 命令行也可以进行分区，如图 7-15（a）、7-15（b）、7-15（c）所示：

parted→select /dev/sdb→mklabel gpt→mkpart primary 0　-1→print

mkfs.ext3　/dev/sdb1

mount　　/dev/sdb1　/data/

```
[root@localhost ~]# parted
GNU Parted 2.1
使用 /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of c
(parted) select /dev/sdb                  选择/dev/sdb硬盘
使用 /dev/sdb
(parted)
(parted) mklabel gpt                      格式类型为GPT
警告: The existing disk label on /dev/sdb will be dest
lost. Do you want to continue?
是/Yes/否/No? yes
(parted)
(parted) mkpart primary 0 -1              将整块硬盘分为一个分区
警告: The resulting partition is not properly aligned
忽略/Ignore/放弃/Cancel?
忽略/Ignore/放弃/Cancel? ignore
(parted)
(parted) print                            打印我们刚分区的磁盘信息
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 53.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
```

图 7-15（a）parted 工具执行 GPT 格式分区

```
[root@localhost ~]# mkfs.ext3  /dev/sdb1
mke2fs 1.41.12 (17-May-2010)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
1310720 inodes, 5242631 blocks
262131 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=4294967296
160 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632
```

图 7-15（b）parted 工具执行 GPT 格式分区

```
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# mount      /dev/sdb1     /data/
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        30G  4.8G   23G  18% /
tmpfs           242M     0  242M   0% /dev/shm
/dev/sda1       194M   44M  141M  24% /boot
/dev/sdb1        20G  173M   19G   1% /data
[root@localhost ~]#
[root@localhost ~]#
```

图 7-15（c） parted 工具执行 GPT 格式分区

7.7 MOUNT 命令工具

Mount 命令工具主要用于将设备或者分区挂载至 Linux 系统目录下，Linux 系统在分区时，也是基于 mount 机制将/dev/sda 分区挂载至系统目录，将设备与目录挂载之后，Linux 操作系统方可进行文件的存储。

7.8 Mount 命令参数详解

如下为企业中 Mount 命令常用参数详解：

mount [-Vh]

mount -a [-fFnrsvw] [-t vfstype]

mount [-fnrsvw] [-o options [,...]] device | dir

mount [-fnrsvw] [-t vfstype] [-o options] device dir

-V: 显示 mount 工具版本号;

-l: 显示已加载的文件系统列表;

-h: 显示帮助信息并退出;

-v: 输出指令执行的详细信息;

-n: 加载没有写入文件/etc/mtab 中的文件系统;

-r: 将文件系统加载为只读模式;

-a: 加载文件/etc/fstab 中配置的所有文件系统;

-o: 指定 mount 挂载扩展参数，常见扩展指令：rw、remount、loop 等，其中-o 相关指令如下:

-o atime: 系统会在每次读取文档时更新文档时间;

-o noatime: 系统会在每次读取文档时不更新文档时间;

-o defaults: 使用预设的选项 rw,suid,dev,exec,auto,nouser 等;

-o exec 允许执行档被执行;

-o user、-o nouser: 使用者可以执行 mount/umount 的动作;

-o remount: 将已挂载的系统分区重新以其他再次模式挂载;

-o ro: 只读模式挂载;

-o rw: 可读可写模式挂载;

-o loop 使用 loop 模式，把文件当成设备挂载至系统目录。
-t: 指定 mount 挂载设备类型，常见类型 nfs、ntfs-3g、vfat、iso9660 等，其中-t 相关指令如下:
iso9660 光盘或光盘镜像;
msdos Fat16 文件系统;
vfat Fat32 文件系统;
ntfs NTFS 文件系统;
ntfs-3g 识别移动硬盘格式;
smbfs 挂载 Windows 文件网络共享;
nfs Unix/Linux 文件网络共享。

7.9 企业常用 Mount 案例

Mount 常用案例演示如下:

mount /dev/sdb1 /data 挂载/dev/sdb1 分区至/data/目录
mount /dev/cdrom /mnt 挂载 Cdrom 光盘至/mnt 目录;
mount -t ntfs-3g /dev/sdc /data1 挂载/dev/sdc 移动硬盘至/data1 目录;
mount -o remount,rw / 重新以读写模式挂载/系统;
mount -t iso9660 -o loop centos7.iso /mnt 将 centos7.iso 镜像文件挂载至/mnt 目录;

mount -t fat32 /dev/sdd1 /mnt 将 U 盘/dev/sdd1 挂载至/mnt/目录； mount -t nfs 192.168.1.11:/data/ /mnt 将 远 程 192.168.1.11:/data 目录挂载至本地/mnt 目录。

7.10 Linux 硬盘故障修复

企业服务器运维中，经常会发现操作系统的分区变成只读文件系统，错误提示信息为“Read-only file system”，出现只读文件系统，会导致只能读取，而无法写入新文件、新数据等操作。

造成该问题的原因包括：磁盘老旧长期大量的读写、文件系统文件被破坏、磁盘碎片文件、异常断电、读写中断等等。

以企业 CentOS 7 Linux 为案例，来修复文件系统，步骤如下：

（1） 远程备份本地其他重要数据，出现只读文件系统，需先备份其他重要数据，基于 rsync|scp 远程备份，其中/data 为源目录，/data/backup/2017/为目标备份目录。

```
rsync -av /data/ root@192.168.111.188:/data/backup/2017/
```

（2） 可以重新挂载/系统，挂载命令如下，测试文件系统是否可以写入文件。

```
mount -o remount,rw /
```

（3） 如果重新挂载/系统无法解决问题，则需重启服务器，以 CD/DVD 光盘引导进入 Linux Rescue 修复模式，如图 7-16（a）、7-16（b）所示，光标选择“Troubleshooting”，按 Enter 键，然后选择“Rescue a CentOS system”，按 Enter 键。

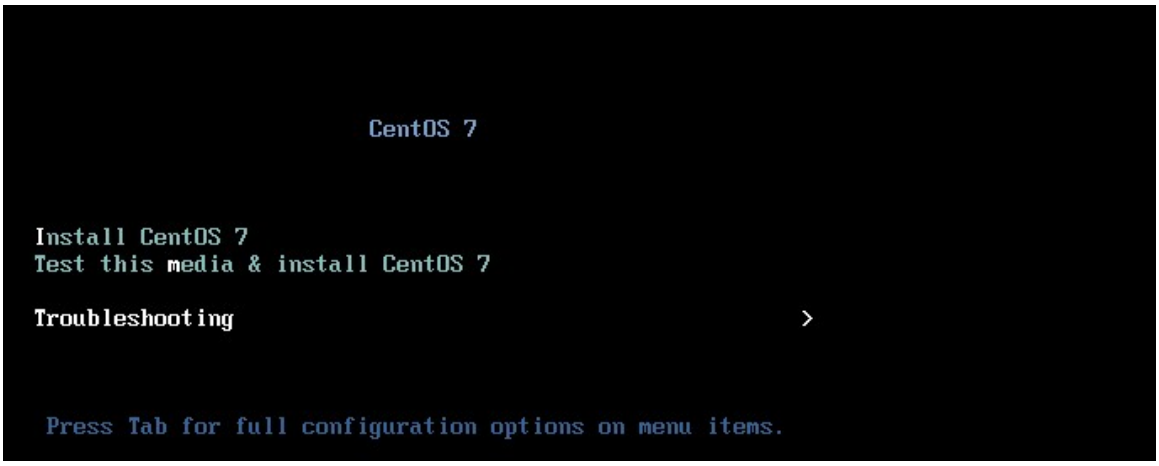


图 7-16（a） 光盘引导进入修复模式

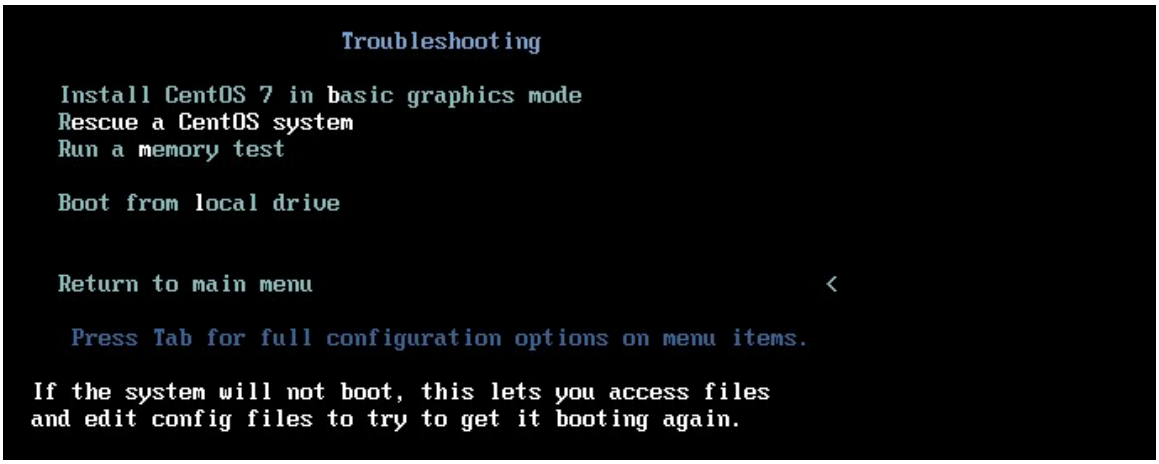


图 7-16（b） 光盘引导进入修复模式

(4) 选择“1）Continue”继续操作，如图 7-17 所示：

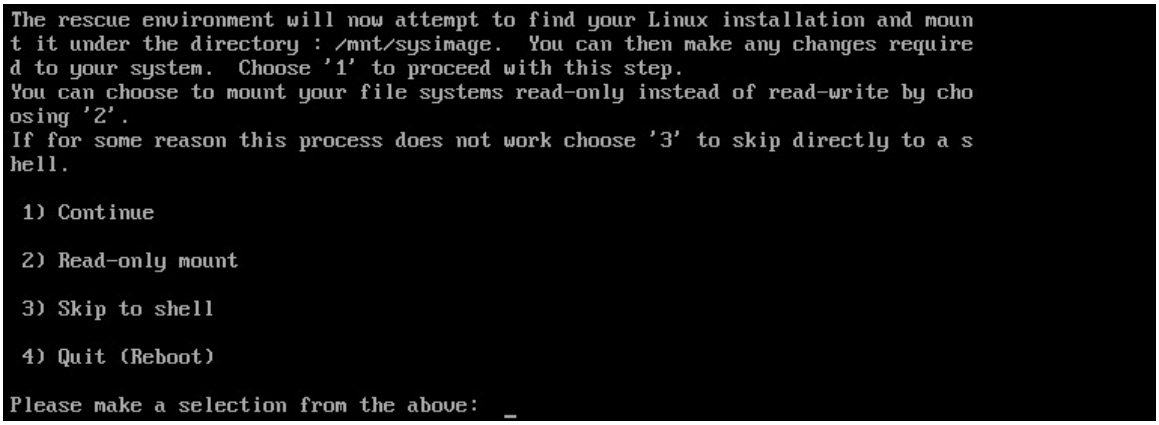


图 7-17 选择 Continue 继续进入系统

(5) 登录修复模式，执行如下命令，df –h 显示原来的文件系统，如图 7-18 所示：

```
chroot /mnt/sysimage
```

```
df  -h
```

```
sh-4.2# chroot   /mnt/sysimage/


bash-4.2#
bash-4.2#
bash-4.2#
bash-4.2# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        37G  5.7G   31G  16% /
devtmpfs        474M     0  474M   0% /dev
tmpfs           493M     0  493M   0% /dev/shm
tmpfs           493M   13M  481M   3% /run
/dev/sda1       197M  103M   94M  53% /boot
/dev/sdb1        20G   45M   19G   1% /data
bash-4.2#
```

图 7-18 切换原分区目录

(6) 对有异常的分区进行检测并修复，根据文件系统类型，执行相应的命令如下：

```
umount      /dev/sda3

fsck.ext4   /dev/sda3      –y
```

(7) 修复完成之后，重启系统即可

```
reboot
```

第8章 Linux 基础服务实战篇

8.1 NFS 文件服务器简介

NFS 是 Network File System 的缩写，即网络文件系统。一种使用于分散式文件系统的协定，由 Sun 公司开发，于 1984 年向外公布。功能是通过网络让不同的机器、不同的操作系统能够彼此分享个别的数据，让应用程序在客户端通过网络访问位于服务器磁盘中的数据，是在类 Unix 系统间实现磁盘文件共享的一种方法。

NFS 在文件传送或信息传送过程中依赖于 RPC 协议，RPC 远程过程调用（Remote Procedure Call）是能使客户端执行其他系统中程序的一种机制，NFS 本身是没有提供信息

传输的协议和功能的。

NFS 应用场景，常用于高可用文件共享，多台服务器共享同样的数据，可扩展性比较差，本身高可用方案不完善，取而代之的数据量比较大的可以采用 MFS、TFS、HDFS、GFS 等等分布式文件系统。

8.2 NFS 文件服务器原理

NFS（网络文件系统）：让网络上的不同 linux/unix 系统机器实现文件共享

nfs 本身只是一种文件系统，没有提供文件传递的功能，但却能让我们进行文件的共享，原因在于 NFS 使用 RPC 服务，用到 NFS 的地方都需要启动 RPC 服务，无论是 NFS 客户端还是服务端

nfs 和 rpc 的关系：nfs 是一个文件系统，负责管理分享的目录;rpc 负责文件的传递

nfs 启动时至少有 rpc.nfsd 和 rpc.mountd2 个 daemon

rpc.nfsd 主要是管理客户机登陆 nfs 服务器时，判断改客户机是否能登陆，和客户机 ID 信息。

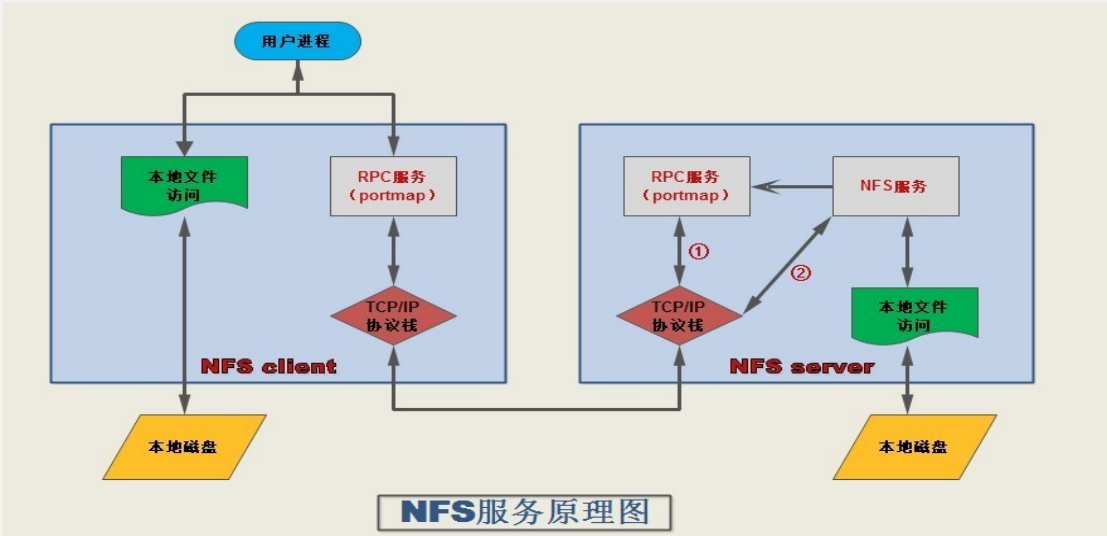
Rpc.mountd 主要是管理 nfs 的文件系统。当客户机顺利登陆 nfs 服务器时，会去读/etc/exports 文件中的配置，然后去对比客户机的权限。

协议使用端口：

RPC：111 tcp/udp

nfsd： 2049 tcp/udp

mountd：RPC 服务在 nfs 服务启动时默认会为 mountd 动态选取一个随机端口（32768--65535）来进行通讯 ，可以在/etc/nfsmount.conf 文件中指定 mountd 的端口



8.3 NFS 文件服务器实战

yum install nfs* -y 如下图，安装成功即可。

```
[root@node1 ~]# yum install nfs* -y
Loaded plugins: fastestmirror, security
Loading mirror speeds from cached hostfile
 * addons: mirror.bit.edu.cn
 * base: mirror.bit.edu.cn
 * epel: mirrors.neusoft.edu.cn
 * extras: mirrors.btte.net
 * updates: mirror.bit.edu.cn
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package nfs-utils.x86_64 1:1.0.9-70.el5 set to be updated
--> Processing Dependency: initscripts >= 8.45.43 for package: nfs-utils
---> Package nfs-utils-lib.i386 0:1.0.8-7.9.el5 set to be updated
--> Processing Dependency: libgssapi.so.2(libgssapi_CITI_2) for package: nfs-utils-lib
--> Processing Dependency: libgssapi.so.2 for package: nfs-utils-lib
---> Package nfs-utils-lib.x86_64 0:1.0.8-7.9.el5 set to be updated
---> Package nfs-utils-lib-devel.i386 0:1.0.8-7.9.el5 set to be updated
---> Package nfs-utils-lib-devel.x86_64 0:1.0.8-7.9.el5 set to be updated
---> Package nfs4-acl-tools.x86_64 0:0.3.3-3.el5 set to be updated
--> Running transaction check
---> Package initscripts.x86_64 0:8.45.44-3.el5.centos set to be updated
---> Package libgssapi.i386 0:0.10-2 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved
```

NFS 安装完毕，需要创建共享目录，共享目录在 vi /etc/exports 文件里面配置，可配置参数如下：

/data/ *(rw,no_root_squash,no_all_squash,sync)

配置文件中添加如上一行，然后重启 Portmap，NFS 服务即可

service portmap restart ;service nfs restart

第一列/data/表示需要共享的目录。

IP 表示允许哪个客户端访问。

IP 后括号里的设置表示对该共享文件的权限。

- ro　　只读访问；
- rw　　读写访问；
- sync　　所有数据在请求时写入共享；
- all_squash　　共享文件的 UID 和 GID 映射匿名用户 anonymous；
- no_all_squash　　保留共享文件的 UID 和 GID（默认）；
- root_squash　　root 用户的所有请求映射成如 anonymous 用户一样的权限；
- no_root_squash　　root 用户具有根目录的完全管理访问权限。

Linux 客户端，如何想使用这个 NFS 文件系统，需要在客户端挂载，挂载命令为：

mount –t nfs 192.168.1.103:/data/ /mnt 即可。

如果有报错根据错误信息排查。常见问题有 rpc 服务没有启动、防火墙没关闭、selinux 未关闭等问题。（拓展* 有兴趣的童鞋可以研究 MFS（分布式文件系统）。）

8.4 进程与线程概念及区别

Linux 系统各种软件和服务，存在于系统，必然会占用系统各种资源，系统资源是如何分配及调度的呢，本节将给读者展示系统进程、资源及调度相关的内容。

进程（Process）是计算机中的软件程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。

在早期面向进程设计的计算机结构中，进程是程序的基本执行实体，在当代面向线程设计的计算机结构中，进程是线程的容器。软件程序是对指令、数据及其组织形式的描述，

而进程是程序的实体，通常而言，把运行在系统中的软件程序称之为进程。

除了进程，读者通常会听到线程的概念，线程也被称为轻量级进程(Lightweight Process，LWP)，是程序执行流的最小单元。一个标准的线程由线程ID，当前指令指针(PC)，寄存器集合和堆栈组成。

线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有操作系统资源，但是该线程可与同属进程的其它线程共享该进程所拥有的全部资源。

程序、进程、线程三者区别如下：

- 程序：程序并不能单独执行，是静止的，只有将程序加载到内存中，系统为其分配资源后才能够执行；
- 进程：程序对一个数据集的动态执行过程，一个进程包含一个或者更多的线程，一个线程同时只能被一个进程所拥有，进程是分配资源的基本单位。进程拥有独立的内存单元，而多个线程共享内存，从而提高了应用程序的运行效率。
- 线程：线程是进程内的基本调度单位，线程的划分尺度小于进程，并发性更高，线程本身不拥有系统资源，但是该线程可与同属进程的其它线程共享该进程所拥有的全部资源。每一个独立的线程，都有一个程序运行的入口、顺序执行序列、和程序的出口。

如图 8-1 所示，程序、进程、线程三者的关系拓扑图：

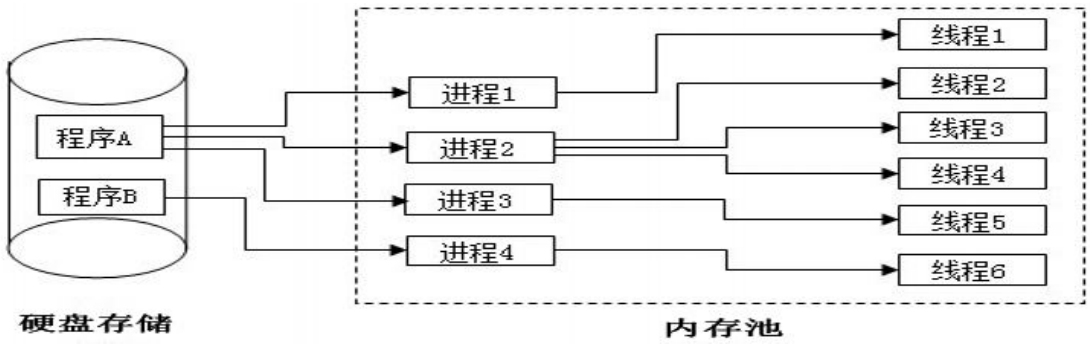


图 8-1　程序、进程、线程关系图

如上图 8-1 所示，多进程、多线程的区别如下：

- 多进程，每个进程互相独立，不影响主程序的稳定性，某个子进程崩溃对其他进程没有影响，通过增加 CPU 可以扩充软件的性能，可以减少线程加锁/解锁的影响，极大提高性能。缺点是多进程逻辑控制复杂，需要和主程序交互，需要跨进程边界，进程之间上下文切换比线程之间上下文切换代价大。
- 多线程，无需跨进程，程序逻辑和控制方式简单，所有线程共享该进程的内存和变量等。缺点是每个线程与主程序共用地址空间，线程之间的同步和加锁控制比较麻烦，一个线程的崩溃会影响到整个进程或者程序的稳定性。

8.5 Vsftpd 服务器企业实战

文件传输协议（File Transfer Protocol，FTP），基于该协议 FTP 客户端与服务端可以实现共享文件、上传文件、下载文件。 FTP 基于 TCP 协议生成一个虚拟的连接，主要用于控制 FTP 连接信息，同时再生成一个单独的 TCP 连接用于 FTP 数据传输。用户可以通过客户端向 FTP 服务器端上传、下载、删除文件，FTP 服务器端可以同时提供给多人共享使用。

FTP 服务是 Client/Server（简称 C/S）模式，基于 FTP 协议实现 FTP 文件对外共享及传输的软件称之为 FTP 服务器源端，客户端程序基于 FTP 协议，则称之为 FTP 客户端，FTP 客户端可以向 FTP 服务器上传、下载文件。

8.6 FTP 工作原理&传输模式

FTP 基于 C/S 模式，FTP 客户端与服务器端有两种传输模式，分别是 FTP 主动模式、FTP 被动模式，主被动模式均是以 FTP 服务器端为参照。主被动模式如图 8-2（a）、8-2（b）所示，主被动模式详细区别如下：

- FTP 主动模式：客户端从一个任意的端口 N（N>1024）连接到 FTP 服务器的 port

21 命令端口，客户端开始监听端口 N+1，并发送 FTP 命令“port N+1”到 FTP 服务器，FTP 服务器以数据端口（20）连接到客户端指定的数据端口（N+1）。

- FTP 被动模式：客户端从一个任意的端口 N (N>1024) 连接到 FTP 服务器的 port 21 命令端口，客户端开始监听端口 N+1，客户端提交 PASV 命令，服务器会开启一个任意的端口（P >1024），并发送 PORT P 命令给客户端。客户端发起从本地端口 N+1 到服务器的端口 P 的连接用来传送数据。

在企业实际环境中，如果 FTP 客户端与 FTP 服务端均开放防火墙，FTP 需以主动模式工作，这样只需要在 FTP 服务器端防火墙规则中，开放 20、21 端口即可。关于防火墙配置后面章节会讲解。

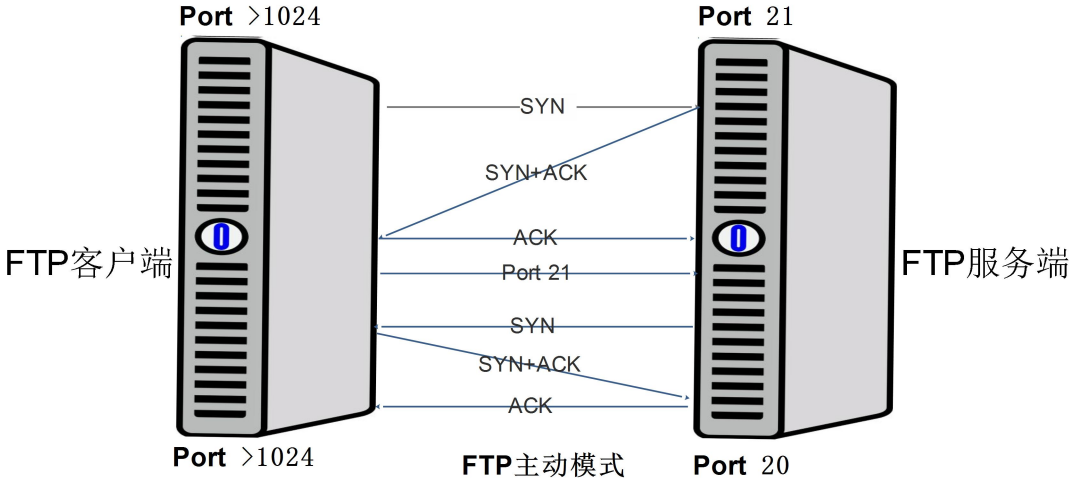


图 8-2（a） FTP 主动模式

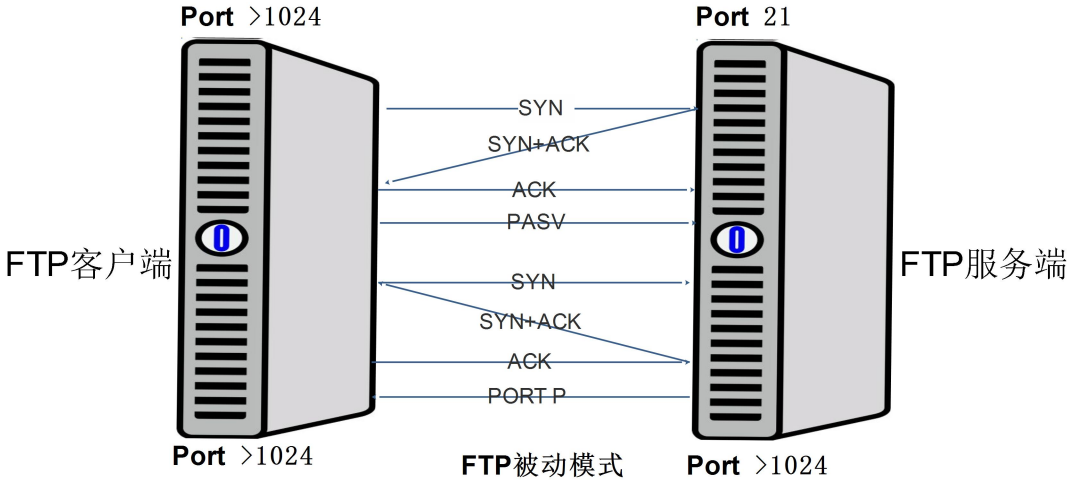


图 8-2（b） FTP 被动模式

8.7 Vsftpd 服务器简介

目前主流的 FTP 服务器端软件包括：Vsftpd、ProFTPD、PureFTPd、Wuftpd、Server-U FTP、 FileZilla Server 等软件，其中 Unix/Linux 使用较为广泛的 FTP 服务器端软件为 Vsftpd 。

非常安全的 FTP 服务进程（Very Secure FTP daemon，Vsftpd），Vsftpd 在 Unix/Linux 发行版中最主流的 FTP 服务器程序，优点小巧轻快，安全易用、稳定高效、满足企业跨部门、多用户的使用（1000 用户）等。

Vsftpd 基于 GPL 开源协议发布，在中小企业中得到广泛的应用，Vsftpd 可以快速上手，基于 Vsftpd 虚拟用户方式，访问验证更加安全。Vsftpd 还可以基于 MYSQL 数据库做安全验证，多重安全防护。

8.8 Vsftpd 服务器安装配置

Vsftpd 服务器端安装有两种方法，一是基于 YUM 方式安装，而是基于源码编译安装，最终实现效果完全一致，本文采用 YUM 安装 Vsftpd，步骤如下：

（1） 在命令行执行如下命令，如图 8-3 所示：

```
yum    install    vsftpd*    -y
```

```
[root@www-jfedu-net ~]# yum install vsftpd* -y
已加载插件：fastestmirror, product-id, search-disabled-repos, subscrip
This system is not registered with Subscription Management. You can us
base
extras
updates
updates/7/x86_64/primary_db
Loading mirror speeds from cached hostfile
 * base: mirrors.btte.net
 * extras: centos.ustc.edu.cn
 * updates: ftp.jaist.ac.jp
正在解决依赖关系
--> 正在检查事务
---> 软件包 vsftpd.x86_64.0.3.0.2-11.el7_2 将被 升级
---> 软件包 vsftpd.x86_64.0.3.0.2-21.el7 将被 更新
---> 软件包 vsftpd-sysvinit.x86_64.0.3.0.2-21.el7 将被 安装
```

图 8-3 YUM 安装 Vsftpd 服务端

（2） 打印 vsftpd 安装后的配置文件路径、启动 Vsftpd 服务及查看进程是否启动，如图 8-4 所示：

```
rpm -ql vsftpd|more

systemctl restart vsftpd.service

ps -ef |grep vsftpd
```

```
[root@www-jfedu-net ~]# rpm -ql vsftpd|more
/etc/logrotate.d/vsftpd
/etc/pam.d/vsftpd
/etc/vsftpd
/etc/vsftpd/ftpusers
/etc/vsftpd/user_list
/etc/vsftpd/vsftpd.conf
/etc/vsftpd/vsftpd_conf_migrate.sh
/usr/lib/systemd/system-generators/vsftpd-generator
/usr/lib/systemd/system/vsftpd.service
/usr/lib/systemd/system/vsftpd.target
/usr/lib/systemd/system/vsftpd@.service
/usr/sbin/vsftpd
/usr/share/doc/vsftpd-3.0.2
/usr/share/doc/vsftpd-3.0.2/AUDIT
/usr/share/doc/vsftpd-3.0.2/BENCHMARKS
```

图 8-4 打印 Vsftpd 软件安装后路径

（3） Vsftpd.conf 默认配置文件详解如下：

```
anonymous_enable=YES 开启匿名用户访问；

local_enable=YES 启用本地系统用户访问；
```

write_enable=YES　　本地系统用户写入权限；

local_umask=022　　本地用户创建文件及目录默认权限掩码；

dirmessage_enable=YES　　打印目录显示信息，通常用于用户第一次访问目录时，信息提示；

xferlog_enable=YES　　启用上传/下载日志记录；

connect_from_port_20=YES　　FTP 使用 20 端口进行数据传输；

xferlog_std_format=YES　　日志文件将根据 xferlog 的标准格式写入；

listen=NO　　Vsftpd 不以独立的服务启动，通过 Xinetd 服务管理，建议改成 YES；

listen_ipv6=YES　　启用 IPV6 监听；

pam_service_name=vsftpd　　登录 FTP 服务器，依据/etc/pam.d/vsftpd 中内容进行认证；

userlist_enable=YES　　vsftpd.user_list 和 ftpusers 配置文件里用户禁止访问 FTP；

tcp_wrappers=YES　　设置 vsftpd 与 tcp wrapper 结合进行主机的访问控制，Vsftpd 服务器检查/etc/hosts.allow 和/etc/hosts.deny 中的设置，来决定请求连接的主机，是否允许访问该 FTP 服务器。

（4）　启动 Vsftpd 服务后，通过 Windows 客户端资源管理器访问 Vsftp 服务器端，如图 8-5 所示：

ftp://192.168.111.131/

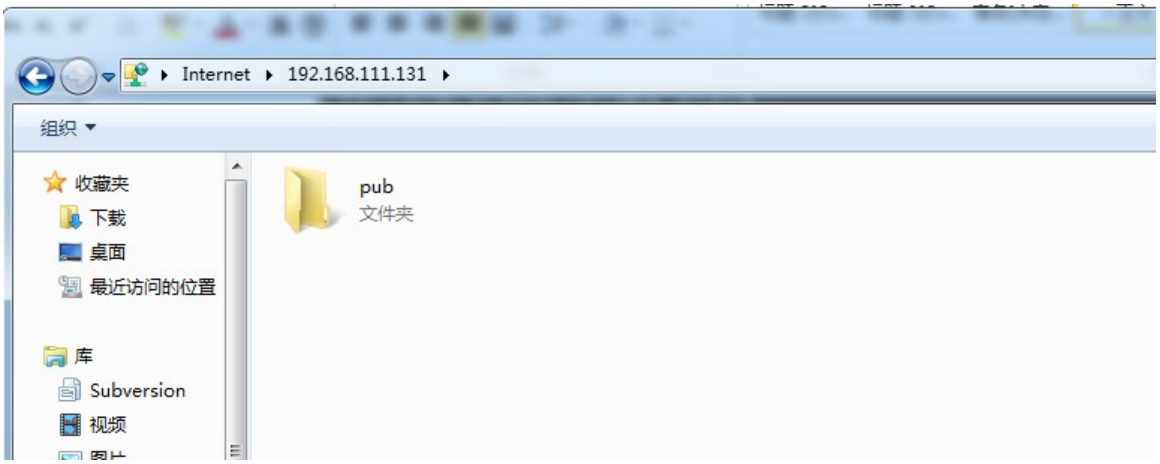


图 8-5 匿名用户访问 FTP 默认目录

FTP 主被动模式，默认为主动模式，设置为被动模式使用端口方法如下：

```
pasv_enable=YES

pasv_min_port=60000

pasv_max_port=60100
```

8.9 Vsftpd 匿名用户配置

Vsftpd 默认以匿名用户访问，匿名用户默认访问的 FTP 服务器端路径为：/var/ftp/pub，匿名用户只有查看权限，无法创建、删除、修改。如需关闭 FTP 匿名用户访问，需修改配置文件 /etc/vsftpd/vsftpd.conf ， 将 anonymous_enable=YES 修改为 anonymous_enable=NO，重启 Vsftpd 服务即可。

如果允许匿名用户能够上传、下载、删除文件，需在/etc/vsftpd/vsftpd.conf 配置文件中加入如下代码：

```
anon_upload_enable=YES                允许匿名用户上传文件；

anon_mkdir_write_enable=YES           允许匿名用户创建目录；

anon_other_write_enable=YES           允许匿名用户其他写入权限。
```

匿名用户完整 vsftpd.conf 配置文件代码如下：

```
anonymous_enable=YES

local_enable=YES

write_enable=YES

local_umask=022

anon_upload_enable=YES

anon_mkdir_write_enable=YES

anon_other_write_enable=YES

dirmessage_enable=YES

xferlog_enable=YES

connect_from_port_20=YES

xferlog_std_format=YES

listen=NO

listen_ipv6=YES

pam_service_name=vsftpd

userlist_enable=YES

tcp_wrappers=YES
```

由于默认 Vsftpd 匿名用户有两种：anonymous、ftp，所以匿名用户如果需要上传文件、删除及修改等权限，需要 ftp 用户对/var/ftp/pub 目录有写入权限，使用如下 chown 和 chmod 任意一种即可，设置命令如下：

```
chown   -R   ftp        pub/

chmod              o+w        pub/
```

如上 Vsftpd.conf 配置文件配置完毕，同时权限设置完，重启 vsftpd 服务即可，通过

Windows 客户端访问，能够上传文件、删除文件、创建目录等操作，如图 8-6 所示：

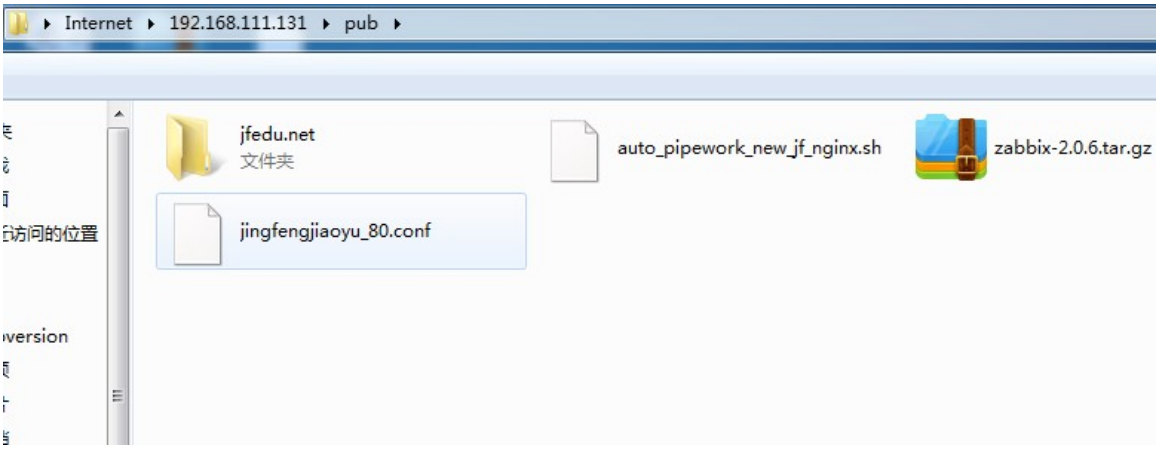


图 8-6 匿名用户访问上传文件

8.10 Vsftpd 系统用户配置

Vsftpd 匿名用户设置完毕，匿名用户，任何人都可以查看 FTP 服务器端的文件、目录，甚至可以修改、删除，此方案如适合存放私密文件在 FTP 服务器端，如何保证文件或者目录专属拥有者呢，Vsftpd 系统用户可以实现该需求。

实现 Vsftpd 系统用户方式验证，只需在 Linux 系统中创建多个用户即可，创建用户使用 useradd，同时给用户设置密码，即可通过用户和密码登录 FTP，进行文件上传、下载、删除等操作。Vsftpd 系统用户实现方法步骤如下：

（1） Linux 系统中创建系统用户 jfedu1、jfedu2，分别设置密码为 123456：

```
useradd    jfedu1

useradd    jfedu2

echo 123456|passwd --stdin   jfedu1

echo 123456|passwd --stdin   jfedu2
```

（2） 修改 vsftpd.conf 配置文件代码如下：

```
anonymous_enable=NO
```

```
local_enable=YES

write_enable=YES

local_umask=022

dirmessage_enable=YES

xferlog_enable=YES

connect_from_port_20=YES

xferlog_std_format=YES

listen=NO

listen_ipv6=YES

pam_service_name=vsftpd

userlist_enable=YES

tcp_wrappers=YES
```

（3） 通过 Windows 资源客户端验证，使用 jfedu1、jfedu2 用户登录 FTP 服务器，即可上传文件、删除文件、下载文件，jfedu1、jfedu2 系统用户上传文件的家目录在 /home/jfedu1、/home/jfedu2 下，如图 8-7（a）、8-7（b）所示：

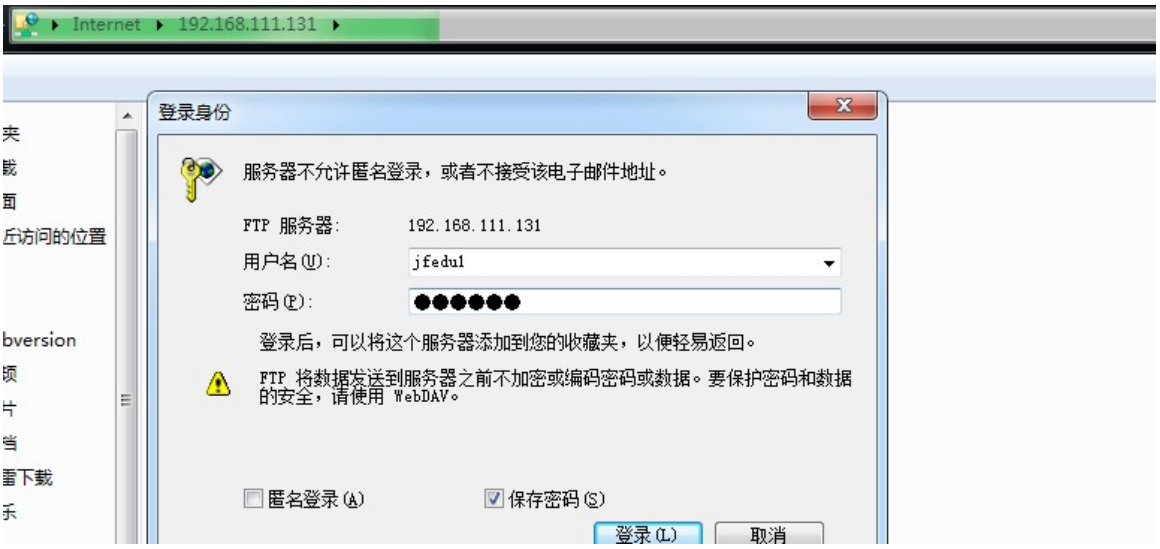


图 8-7（a） jfedu1 用户登录 FTP 服务器

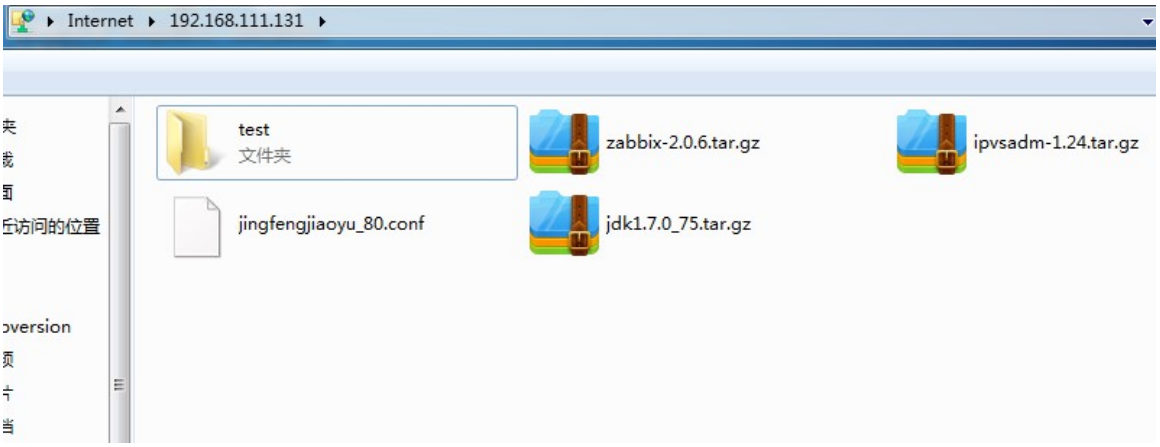


图 8-7（b） jfedu1 登录 FTP 服务器上传文件

8.11 Vsftpd 虚拟用户配置

如果基于 Vsftpd 系统用户访问 FTP 服务器，系统用户越多越不利于管理，而且不利于系统安全管理，鉴于此，为了能更加的安全使用 VSFTPD，需使用 Vsftpd 虚拟用户方式。

Vsftpd 虚拟用户原理：虚拟用户就是没有实际的真实系统用户，而是通过映射到其中一个真实用户以及设置相应的权限来实现访问验证，虚拟用户不能登录 Linux 系统，从而让系统更加的安全可靠。

Vsftpd 虚拟用户企业案例配置步骤如下：

（1） 安装 Vsftpd 虚拟用户需用到的软件及认证模块：

```
yum install pam* libdb-utils libdb* --skip-broken -y
```

（2） 创建虚拟用户临时文件/etc/vsftpd/ftpusers.txt，新建虚拟用户和密码，其中 jfedu001、jfedu002 为虚拟用户名，123456 为密码，如果有多个用户，依次格式填写即可：

```
jfedu001
123456
jfedu002
```

```
123456
```

（3） 生成 Vsftpd 虚拟用户数据库认证文件，设置权限 700：

```
db_load -T -t hash -f /etc/vsftpd/ftpusers.txt
/etc/vsftpd/vsftpd_login.db
chmod 700 /etc/vsftpd/vsftpd_login.db
```

（4） 配置 PAM 认证文件，/etc/pam.d/vsftpd 行首加入如下两行：

```
auth required pam_userdb.so db=/etc/vsftpd/vsftpd_login
account required pam_userdb.so db=/etc/vsftpd/vsftpd_login
```

（5） 所有 Vsftpd 虚拟用户需要映射到一个系统用户，该系统用户不需要密码，也不需要登录，主要用于虚拟用户映射使用，创建命令如下：

```
useradd -s /sbin/nologin ftpuser
```

（6） 完整 vsftpd.conf 配置文件代码如下：

```
#global config Vsftpd 2017
anonymous_enable=YES
local_enable=YES
write_enable=YES
local_umask=022
dirmessage_enable=YES
xferlog_enable=YES
connect_from_port_20=YES
xferlog_std_format=YES
listen=NO
```

```
listen_ipv6=YES

userlist_enable=YES

tcp_wrappers=YES

#config virtual user FTP

pam_service_name=vsftpd

guest_enable=YES

guest_username=ftpuser

user_config_dir=/etc/vsftpd/vsftpd_user_conf

virtual_use_local_privs=YES
```

如上 Vsftpd 虚拟用户配置文件参数详解：

```
#config virtual user FTP

pam_service_name=vsftpd                          虚拟用户启用 pam 认证；

guest_enable=YES                                 启用虚拟用户；

guest_username=ftpuser                           映射虚拟用户至系统用户 ftpuser；

user_config_dir=/etc/vsftpd/vsftpd_user_conf     设置虚拟用户配置文件所在的

目录；

virtual_use_local_privs=YES                      虚拟用户使用与本地用户相同的权

限。
```

（7） 至此，所有虚拟用户共同基于/home/ftpuser 主目录实现文件上传与下载，可以在/etc/vsftpd/vsftpd_user_conf 目录创建虚拟用户各自的配置文件，创建虚拟用户配置文件主目录:

```
mkdir  -p     /etc/vsftpd/vsftpd_user_conf/
```

（8） 如下分别为虚拟用户 jfedu001、jfedu002 用户创建配置文件：

vim /etc/vsftpd/vsftpd_user_conf/jfedu001，同时创建私有的虚拟目录，代码如下：

```
local_root=/home/ftpuser/jfedu001
write_enable=YES
anon_world_readable_only=YES
anon_upload_enable=YES
anon_mkdir_write_enable=YES
anon_other_write_enable=YES
```

vim /etc/vsftpd/vsftpd_user_conf/jfedu002，同时创建私有的虚拟目录，代码如下：

```
local_root=/home/ftpuser/jfedu002
write_enable=YES
anon_world_readable_only=YES
anon_upload_enable=YES
anon_mkdir_write_enable=YES
anon_other_write_enable=YES
```

虚拟用户配置文件内容详解：

local_root=/home/ftpuser/jfedu002	jfedu002 虚拟用户配置文件路径；
write_enable=YES	允许登陆用户有写权限；
anon_world_readable_only=YES	允许匿名用户下载，然后读取文件；
anon_upload_enable=YES	允许匿名用户上传文件权限，只有在
write_enable=YES 时该参数才生效；	
anon_mkdir_write_enable=YES	允许匿名用户创建目录，只有在

write_enable=YES 时该参数才生效；

anon_other_write_enable=YES　　允许匿名用户其他权限，例如删除、重命名等。

（9）　创建虚拟用户各自虚拟目录：

```
mkdir -p /home/ftpuser/{jfedu001,jfedu002} ; chown -R ftpuser:ftpuser /home/ftpuser
```

重启 Vsftpd 服务，通过 Windows 客户端资源管理器登录 Vsftpd 服务端，测试结果如图 8-8（a）、8-8（b）所示：

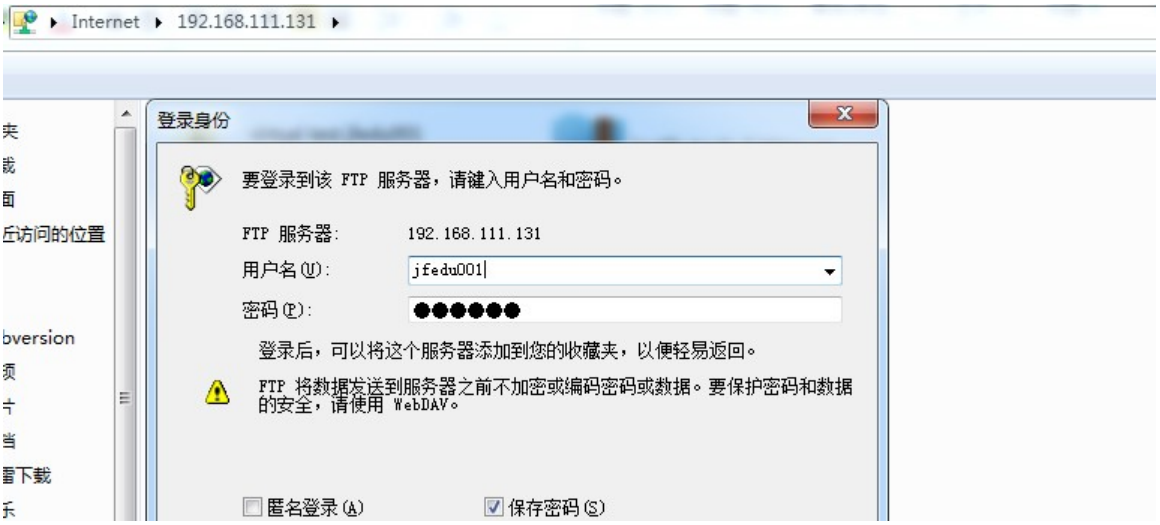


图 8-8（a） jfedu001 虚拟用户登录 FTP 服务器

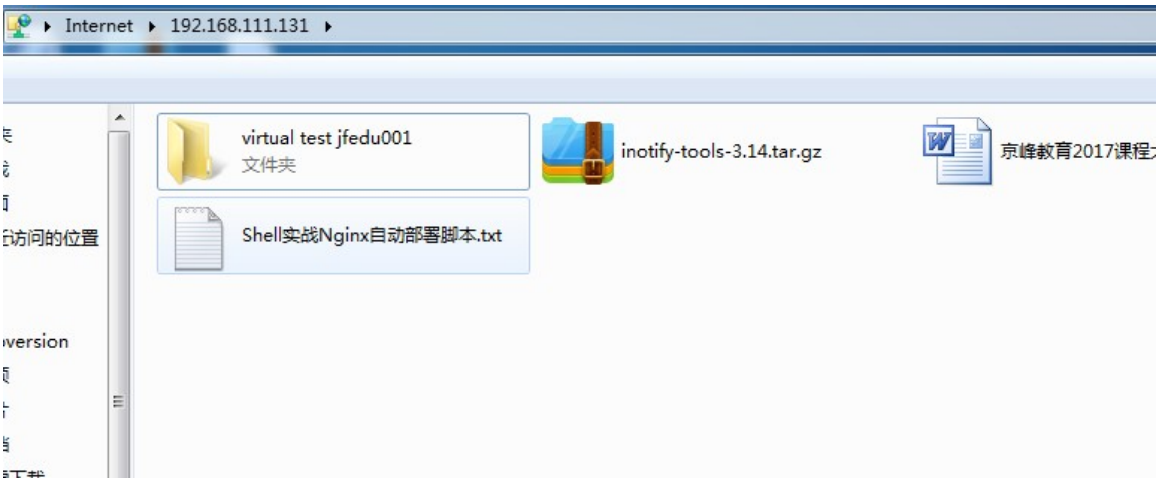


图 8-8（b） jfedu001 虚拟用户上传下载文件

第9章　HTTP 协议原理剖析篇

超文本传输协议（HyperText Transfer Protocol，HTTP)是互联网上应用最为广泛的一种网络协议。所有的 WWW 服务器都基于该协议。HTTP 设计最初的目的是为了提供一种发布 WEB 页面和接收 WEB 页面的方法。

本章向读者介绍 TCP、HTTP 协议、HTTP 资源定位、HTTP 请求及响应头详细信息、HTTP 状态码及 MIME 类型详解等。

9.1　TCP 协议与 HTTP 协议

1960 年美国人 Ted Nelson 构思了一种通过计算机处理文本信息的方法，并称之为超文本（hypertext）,为 HTTP 超文本传输协议标准架构的发展奠定了根基。Ted Nelson 组织协调万维网协会（World Wide Web Consortium）和互联网工程工作小组（Internet Engineering Task Force)共同合作研究,最终发布了一系列的 RFC,其中著名的 RFC 2616 定义了 HTTP 1.1。

很多读者对 TCP 协议与 HTTP 协议存在疑问,这两者有什么区别呢,从应用领域来说，TCP 协议主要用于数据传输控制，而 HTTP 协议主要用于应用层面的数据交互，本质上两者没有可比性。

HTTP 协议属于应用层协议，是建立在 TCP 协议基础之上，HTTP 协议以客户端请求和服务器端应答为标准，浏览器通常称为客户端，而 WEB 服务器称之为服务器端。客户端打开任意一个端口向服务端的指定端口（默认为 80）发起 HTTP 请求，首先会发起 TCP 三次握手，TCP 三次握手的目的是建立可靠的数据连接通道，TCP 三次握手通道建立完毕，进行 HTTP 数据交互，如图 9-1（a）、9-1（b）所示:

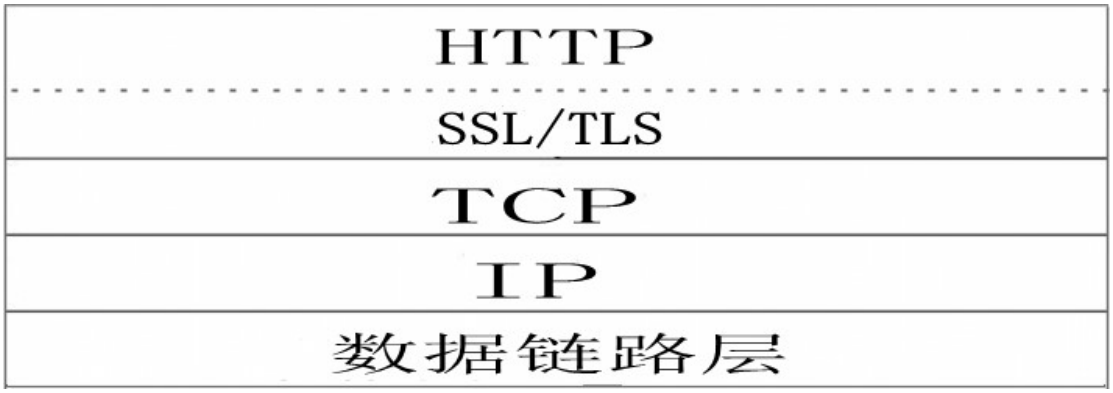


图 9-1（a） HTTP 与 TCP 关系结构图

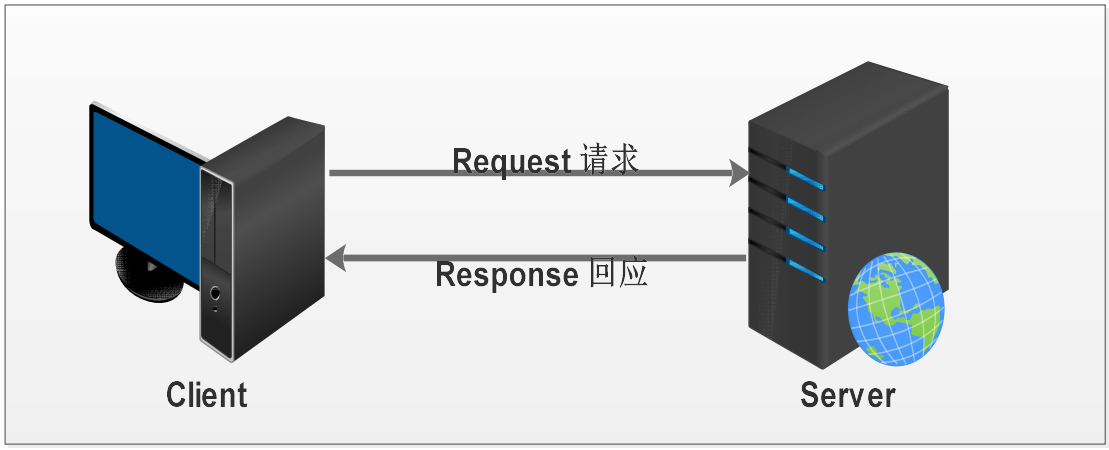


图 9-1（b） HTTP 客户端与服务器

当客户端请求的数据接收完毕后，HTTP 服务器端会断开 TCP 连接，整个 HTTP 连接过程非常短。HTTP 连接也称为无状态的连接，无状态连接是指客户端每次向服务器发起 HTTP 请求时，每次请求都会建立一个新的 HTTP 连接，而不是在一个 HTTP 请求基础上进行所有数据的交互。

9.2 资源定位标识符

发起 HTTP 请求的内容资源由统一资源标示符（Uniform Resource Identifiers，URI）来标识，关于资源定位及标识有三种：URI、URN、URL，三种资源定位详解如下：

- 统一资源标识符（uniform resource identifier，URI），用来唯一标识一个资源；
- 统一资源定位器（uniform resource locator，URL），是一种具体的 URI。URL 可以用来标识一个资源，而且访问或者获取该资源；

- 统一资源命名（uniform resource name，URN），通过名字来标识或识别资源。

如图 9-2 所示，可以直观区分 URI、URN、URL 的区别：

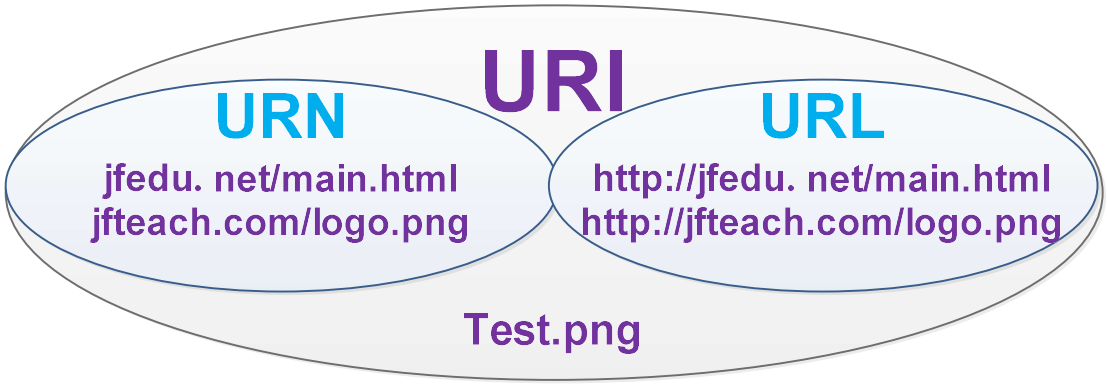


图 9-2 URI、URN、URL 关联与区别

三种资源标识，其中 URL 资源标识方式使用最为广泛，完整的 URL 标识格式如下：

protocol://host[:port]/path/.../[?query-string][#anchor]	
protocol	基于某种协议，常见协议：http、https、ftp、rsync 等；
host	服务器的 IP 地址或者域名；
port	服务器的端口号，如果是 HTTP 80 端口，默认可以省略。
path	访问资源在服务器的路径；
query-string	传递给服务器的参数及字符串；
anchor-	锚定结束；

Http URL 案例演示如下：

http://www.jfedu.net/newindex/plus/list.php?tid=2#jfedu	
protocol:	http 协议；
host:	www.jfedu.net；
path:	/newindex/plus/list.php
Query String:	tid=2

Anchor:	jfedu

9.3 HTTP 与端口通信

HTTP WEB 服务器默认在本机会监听 80 端口，不仅 HTTP 会开启监听端口，其实每个软件程序在 Linux 系统中运行，会以进程的方式启动，程序就会启动并监听本地接口的端口，为什么会引入端口这个概念呢?

端口是 TCP/IP 协议中应用层进程与传输层协议实体间的通信接口，端口是操作系统可分配的一种资源，应用程序通过系统调用与某个端口绑定后，传输层传给该端口的数据会被该进程接收，相应进程发给传输层的数据都通过该端口输出。

在网络通信过程中，需要唯一识别通信两端设备的端点，就是使用端口识别运行于某主机中的应用程序。如果没有引入端口，则只能通过 PID 进程号进行识别，而 PID 进程号是系统动态分配的，不同的系统会使用不同的进程标识符，应用程序在运行之前没有明确的进程号，如果需要运行后再广播进程号则很难保证通信的顺利进行。

而引入端口后，就可以利用端口号识别应用程序，同时通过固定端口号来识别和使用某些公共服务，例如如 HTTP 默认使用 80 端口，而 FTP 使用 21、20 端口，MYSQL 则使用 3306 端口。

使用端口还有一个原因是随着计算机网络技术的发展，物理机器上的硬件接口已不能满足网络通信的要求，而 TCP/IP 协议模型作为网络通信的标准就解决了这个通信难题。

TCP/IP 协议中引入了一种被称为套接字（Socket）的应用程序接口。基于 Socket 接口技术，一台计算机就可以与任何一台具有 Socket 接口的计算机进行通信，而监听的端口在服务器端也称之为 Socket 接口。

9.4 HTTP Request 与 Response 详解

客户端浏览器向 WEB 服务器发起 Request，Web 服务器接到 Request 后进行处理，会生成相应的 Response 信息返给浏览器，客户端浏览器收到服务器返回的 Response 信息，会对信息进行解析处理，最终用户看到浏览器展示 WEB 服务器的网页内容。

客户端发起 Request，Request 消息分为三个部分，分别包括：Request line、Request header、Body，如图 9-3 所示：

图 9-3 HTTP Request Message 组成

Unix/Linux 系统中执行 CURL -v 命令可以打印访问 WEB 服务器的 Request 及 Response 详细处理流程，如图 9-4 所示：

```
curl   -v   http://192.168.111.131/index.html
```

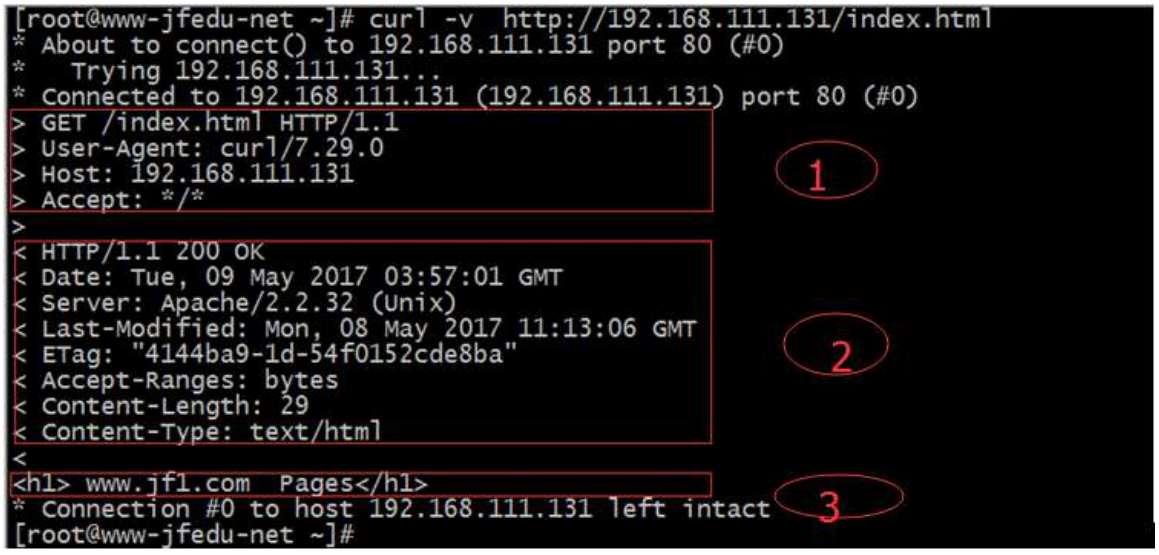


图 9-4 Request 及 Response 请求回应流程

（1） Request 信息详解如表 9-1 所示：

<table>
<tr><td>GET /index.html
HTTP/1.1</td><td>请求行</td><td rowspan="4">Request Message</td></tr>
<tr><td>User-Agent: curl/7.19.7
Host: 192.168.111.131
Accept: */*
……</td><td>请求头部</td></tr>
<tr><td>></td><td>空行</td></tr>
<tr><td>></td><td>请求 Body</td></tr>
<tr><td colspan="3">第一部分：请求行，指定请求类型，访问的资源及使用的 HTTP 协议版本。
GET 表示 Request 请求类型为 GET；/index.html 表示访问的资源；HTTP/1.1 表示协议版本。
第二部分：请求头部，请求行下一行起，指定服务器要使用的附加信息；
User-Agent 表示用户使用的代理软件，常指浏览器；HOST 表示请求的目的主机。
第三部分：空行，请求头部后面的空行表示请求头发送完毕。
第四部分：请求数据也叫 Body，可以添加任意的数据，Get 请求的 Body 内容默认为空。</td></tr>
</table>

表 9-1 Request 请求头详解

（2） Response 信息详解如表 9-2 所示：

<table>
<tr><td>HTTP/1.1 200 OK</td><td>响应行</td><td rowspan="2">Response Message</td></tr>
<tr><td>Server: nginx/1.10.1
Date: Thu, 11 May 2017
Content-Type: text/html</td><td>响应头部</td></tr>
</table>

……		
>	空行	
www.jf1.com Pages	响应 Body	

第一部分：响应状态行，包括 HTTP 协议版本号、状态码、状态消息。

HTTP/1.1 表示 HTTP 协议版本号；200 表示返回状态码；OK 表示状态消息。

第二部分：消息报头，响应头部附加信息。

Date 表示生成响应的日期和时间，Content-Type 表示指定 MIME 类型的

HTML(text/html)，编码类型是 UTF-8，记录文件资源的 Last-Modified 时间。

第三部分：空行，表示消息报头响应完毕。

第四部分：响应正文，服务器返回给客户端的文本信息。

表 9-2 Request 请求头详解

（3） Request 请求方法根据请求的资源不同，有如下请求方法：

GET 方法，向特定的资源发出请求，获取服务器端数据；

POST 方法，向 WEB 服务器提交数据进行处理请求，常指提交新数据；

PUT 方法，向 WEB 服务器提交上传最新内容，常指更新数据；

DELETE 方法，请求删除 Request-URL 所标识的服务器资源；

TRACE 方法，回显服务器收到的请求，主要用于测试或诊断；

CONNECT 方法，HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器；

OPTIONS 方法，返回服务器针对特定资源所支持的 HTTP 请求方法；

HEAD 方法，HEAD 方法跟 GET 方法相同，只不过服务器响应时不会返回消息体。

9.5 HTTP 1.0/1.1 协议区别

HTTP 协议定义服务器端和客户端之间文件传输的沟通方式 HTTP1.0 运行方式，如图 9-5 所示：

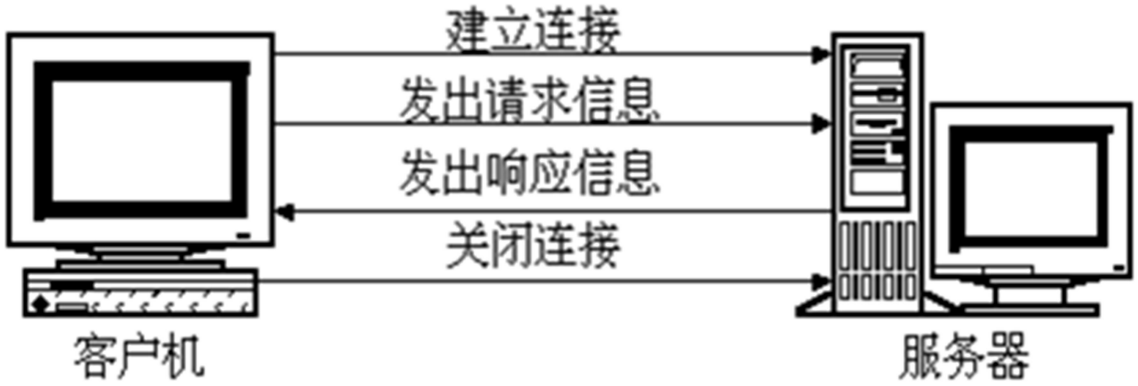


图 9-5 HTTP1.0 客户端、服务器传输模式

- 基于 HTTP 协议的客户/服务器模式的信息交换过程，如图所示，它分四个过程，建立连接、发送请求信息、发送响应信息、关闭连接；
- 浏览器与 WEB 服务器的连接过程是短暂的，每次连接只处理一个请求和响应。对每一个页面的访问，浏览器与 WEB 服务器都要建立一次单独的连接；
- 浏览器到 WEB 服务器之间的所有通讯都是完全独立分开的请求和响应。

HTTP1.1 运行方式，如图 9-6 所示：

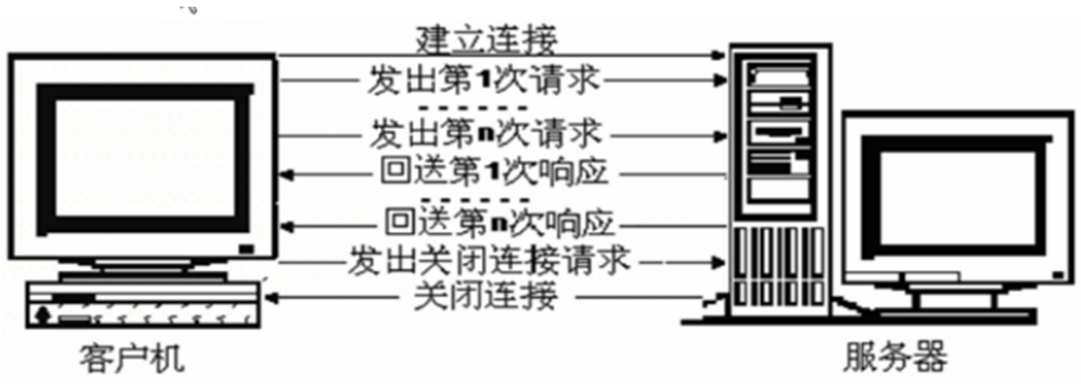


图 9-6 HTTP1.1 客户端、服务器传输模式

- 在一个 TCP 连接上可以传送多个 HTTP 请求和响应；
- 多个请求和响应过程可以重叠；
- 增加了更多的请求头和响应头，比如 Host、If-Unmodified-Since 请求头等。

9.6 HTTP 状态码详解

HTTP 状态码（HTTP Status Code）是用来表示 WEB 服务器 HTTP Response 状态的 3 位数字代码，常见的状态码范围分类：

100-199	用于指定客户端应相应的某些动作；
200-299	用于表示请求成功；
300-399	已移动的文件且被包含在定位头信息中指定新的地址信息；
400-499	用于指出客户端的错误；
500-599	用于支持服务器错误。

HTTP 协议 Response 常用状态码详解表 9-3 所示：

HTTP 状态码	状态码英文含义	状态码中文含义
100	Continue	HTTP/1.1 新增状态码，表示继续，客户端继续请求 HTTP 服务器；
101	Switching Protocols	服务器根据客户端的请求切换协议，切换到 HTTP 的新版本协议；
200	OK	HTTP 请求完成，常用于 GET、POST 请求中；
301	Moved Permanently	永久移动，请求的资源已被永久的移动到新 URI；

302	Found	临时移动，资源临时被移动，客户端应继续使用原有 URI；
304	Not Modified	文件未修改，请求的资源未修改，服务器返回此状态码时，常用于缓存；
400	Bad Request	客户端请求的语法错误，服务器无法解析或者访问；
401	Unauthorized	请求要求用户的身份认证；
402	**Payment Required**	此状态码保留，为以后使用；
403	Forbidden	服务器理解请求客户端的请求，但是拒绝执行此请求；
404	Not Found	服务器没有该资源，请求的文件找不到；
405	Method Not Allowed	客户端请求中的方法被禁止；
406	Not Acceptable	服务器无法根据客户端请求的内容特性完成请求；
499	**Client has closed connection**	服务器端处理的时间过长；
500	Internal Server Error	服务器内部错误，无法完成请求；
502	Bad Gateway	服务器返回错误代码或者代理服务器错误的网关；
503	Service Unavailable	服务器无法响应客户端请求，或者后端服务器异常；
504	Gateway Time-out	网关超时或者代理服务器超时；

505	HTTP Version not supported	服务器不支持请求的 HTTP 协议的版本，无法完成处理。

表 9-3 HTTP 常用状态码

9.7 HTTP MIME 类型支持

浏览器接收到 WEB 服务器的 Response 信息，浏览器会进行解析，在解析页面之前，浏览器必须启动本地相应的应用程序来处理获取到的文件类型。

基于多用途互联网邮件扩展类型（Multipurpose Internet Mail Extensions，MIME），可以明确某种文件在客户端用某种应用程序来打开，当该扩展名文件被访问的时候，浏览器会自动使用指定应用程序来打开，设计之初是为了在发送电子邮件时附加多媒体数据，让邮件客户程序能根据其类型进行处理。然而当它被 HTTP 协议支持之后，它使得 HTTP 传输的不仅是普通的文本，可以支持更多文件类型、多媒体音、视频等。

在 HTTP 协议中，HTTP Response 消息，MIME 类型被定义在 Content-Type header 中，例如：Content-Type: text/html，表示默认指定该文件为 html 类型，在浏览器端会以 HTML 格式来处理。

在最早的 HTTP 协议中，并没有附加的数据类型信息，所有传送的数据都被客户程序解释为超文本标记语言 HTML 文档，为了支持多媒体数据类型，新版 HTTP 协议中就使用了附加在文档之前的 MIME 数据类型信息来标识数据类型，如表 9-4 所示：

Mime-Types(MIME 类型)	Dateiendung (扩展名)	Bedeutung
application/msexcel	*.xls *.xla	Microsoft Excel Dateien
application/mshelp	*.hlp *.chm	Microsoft Windows Hilfe Dateien

application/mspowerpoint	*.ppt *.ppz *.pps *.pot	Microsoft Powerpoint Dateien
application/msword	*.doc *.dot	Microsoft Word Dateien
application/octet-stream	*.exe	exe
application/pdf	*.pdf	Adobe PDF-Dateien
application/post******	*.ai *.eps *.ps	Adobe Post******-Dateien
application/rtf	*.rtf	Microsoft RTF-Dateien
application/x-httpd-php	*.php *.phtml	PHP-Dateien
application/x-java******	*.js	serverseitige Java******-Dateien
application/x-shockwave-flash	*.swf *.cab	Flash Shockwave-Dateien
application/zip	*.zip	ZIP-Archivdateien
audio/basic	*.au *.snd	Sound-Dateien
audio/mpeg	*.mp3	MPEG-Dateien
audio/x-midi	*.mid *.midi	MIDI-Dateien
audio/x-mpeg	*.mp2	MPEG-Dateien
audio/x-wav	*.wav	Wav-Dateien
image/gif	*.gif	GIF-Dateien
image/jpeg	*.jpeg *.jpg *.jpe	JPEG-Dateien

image/x-windowdump	*.xwd	X-Windows Dump
text/css	*.css	CSS Stylesheet-Dateien
text/html	*.htm *.html *.shtml	-Dateien
text/java******	*.js	Java******-Dateien
text/plain	*.txt	reine Textdateien
video/mpeg	*.mpeg *.mpg *.mpe	MPEG-Dateien
video/vnd.rn-realvideo	*.rmvb	realplay-Dateien
video/quicktime	*.qt *.mov	Quicktime-Dateien
video/vnd.vivo	*viv *.vivo	Vivo-Dateien

表 9-4　HTTP MIME 类型详解

第10章　Apache WEB 服务器实战篇

10.1　Apache WEB 软件简介

Apache HTTP Server 是 Apache 软件基金会的一个开源的网页服务器，是世界使用排名第一的 Web 服务器软件，可以运行在几乎所有广泛使用的计算机平台上，由于其跨平台和安全性被广泛使用，是目前最流行的 Web 服务器端软件之一。

Apache 服务器是一个多模块化的服务器，经过多次修改，成为目前世界使用排名第一的 Web 服务器软件。Apache 取自“A Patchy Server”的读音，即充满补丁的服务器，因为 Apache 基于 GPL 发布，大量开发者不断为 Apache 贡献新的代码、功能、新的特性、修改原来的缺陷。

Apache 服务器的特点是使用简单、速度快、性能稳定，可以做负载均衡及代理服务器

来使用。

10.2 Prefork 的工作原理

如果不用“——with-mpm”显式指定某种 MPM,prefork 就是 Unix 平台上缺省的 MPM.它所采用的预派生子进程方式也是 Apache1.3 中采用的模式.prefork 本身并没有使用到线程,2.0 版使用它是为了与 1.3 版保持兼容性;另一方面,prefork 用单独的子进程来处理不同的请求,进程之间是彼此独立的,这也使其成为最稳定的 MPM 之一.

prefork 的工作原理是,控制进程在最初建立“StartServers”个子进程后,为了满足 MinSpareServers 设置的需要创建一个进程,等待一秒钟,继续创建两个,再等待一秒钟,继续创建四个……如此按指数级增加创建的进程数,最多达到每秒 32 个,直到满足 MinSpareServers 设置的值为止.这就是预派生（prefork）的由来.这种模式可以不必在请求到来时再产生新的进程,从而减小了系统开销以增加性能.

10.3 Worker 的工作原理

相对于 prefork,worker 是 2.0 版中全新的支持多线程和多进程混合模型的 MPM.由于使用线程来处理,所以可以处理相对海量的请求,而系统资源的开销要小于基于进程的服务器.但是,worker 也使用了多进程,每个进程又生成多个线程,以获得基于进程服务器的稳定性.这种 MPM 的工作方式将是 Apache2.0 的发展趋势.

worker 的工作原理是,由主控制进程生成“StartServers”个子进程,每个子进程中包含固定的 ThreadsPerChild 线程数,各个线程独立地处理请求.同样,为了不在请求到来时再生成线程,MinSpareThreads 和 MaxSpareThreads 设置了最少和最多的空闲线程数;而

MaxClients 设置了所有子进程中的线程总数.如果现有子进程中的线程总数不能满足负载,控制进程将派生新的子进程.

Worker 模式下所能同时处理的请求总数是由子进程总数乘以 ThreadsPerChild 值决定的,应该大于等于 MaxClients.如果负载很大,现有的子进程数不能满足时,控制进程会派生新的子进程.默认最大的子进程总数是 16,加大时也需要显式声明 ServerLimit（最大值是 20000）

需要注意的是,如果显式声明了 ServerLimit,那么它乘以 ThreadsPerChild 的值必须大于等于 MaxClients,而且 MaxClients 必须是 ThreadsPerChild 的整数倍,否则 Apache 将会自动调节到一个相应值（可能是个非期望值）.

在大多数平台上，Prefork MPM 在效率上要比 Worker MPM 要高，但是内存使用大得多。prefork 的无线程设计在某些情况下将比 worker 更有优势：它可以使用那些没有处理好线程安全的第三方模块，并且对于那些线程调试困难的平台而言，它也更容易调试一些。

Worker 模式：Worker MPM 使用多个子进程，每个子进程有多个线程。每个线程在某个确定的时间只能维持一个连接。通常来说，在一个高流量的 HTTP 服务器上，Worker MPM 是个比较好的选择，因为 Worker MPM 的内存使用比 Prefork MPM 要低得多。

Worker MPM 也由不完善的地方，如果一个线程崩溃，整个进程就会连同其所有线程一起"死掉".由于线程共享内存空间，所以一个程序在运行时必须被系统识别为"每个线程都是安全的"。

10.4 Apache 源码安装编译配置

- 源码安装 Apache

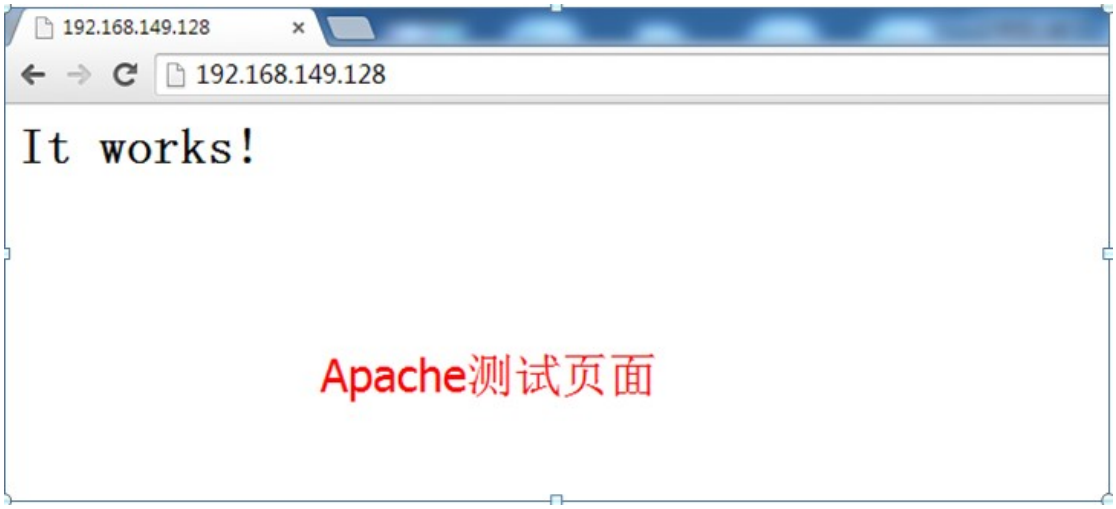
官方下载目前稳定版本，

http://mirrors.hust.edu.cn/apache/httpd/httpd-2.2.32.tar.bz2

解压安装如下，安装 apache 之前，需要先安装 apr apr-util。

```
[root@node1 ~]# cd soft/
[root@node1 soft]#
[root@node1 soft]# tar xzf httpd-2.2.27.tar.gz
[root@node1 soft]#
[root@node1 soft]# ls
httpd-2.2.27  httpd-2.2.27.tar.gz  zabbix
[root@node1 soft]# cd httpd-2.2.27
[root@node1 httpd-2.2.27]# ls
ABOUT_APACHE  buildconf      config.status  httpd.dep   InstallBin.dsp  Makefile      modules.o
acinclude.m4  buildmark.o    configure      httpd.dsp   LAYOUT          Makefile.in   NOTICE
Apache.dsw    CHANGES        configure.in   httpd.mak   libhttpd.dep    Makefile.win  NWGNUmakefile
build         config.layout  docs           httpd.spec  libhttpd.dsp    modules       os
BuildAll.dsp  config.log     emacs-style    include     libhttpd.mak    modules.c     README
BuildBin.dsp  config.nice    httpd          INSTALL     LICENSE         modules.lo    README.platforms
[root@node1 httpd-2.2.27]# yum install apr apr-util apr-devel apr-util-devel
[root@node1 httpd-2.2.27]#
[root@node1 httpd-2.2.27]#
[root@node1 httpd-2.2.27]# ./configure --prefix=/usr/local/apache2 --enable-rewrite --enable-so
[root@node1 httpd-2.2.27]#
[root@node1 httpd-2.2.27]#
[root@node1 httpd-2.2.27]# make
[root@node1 httpd-2.2.27]#
[root@node1 httpd-2.2.27]#
[root@node1 httpd-2.2.27]# make install
[root@node1 httpd-2.2.27]#
[root@node1 httpd-2.2.27]#
```

然后启动 apache 服务: /usr/local/apache2/bin/apachectl start



查看 apache 进程及端口:

```
[root@node1 ~]#
[root@node1 ~]# ps -ef |grep httpd
root      1585     1  0 15:32 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
daemon    1586  1585  0 15:32 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
daemon    1587  1585  0 15:32 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
daemon    1588  1585  0 15:32 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
daemon    1589  1585  0 15:32 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
daemon    1590  1585  0 15:32 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
daemon    1594  1585  0 15:32 ?        00:00:00 /usr/local/apache2/bin/httpd -k start
root      1614  5801  0 15:35 pts/0    00:00:00 grep httpd
[root@node1 ~]#
[root@node1 ~]#
[root@node1 ~]# netstat -tnl |grep 80
tcp        0      0 :::80                       :::*                        LISTEN
[root@node1 ~]#
[root@node1 ~]# lsof -i :80
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
httpd    1585   root    3u  IPv6  43197      0t0  TCP *:http (LISTEN)
httpd    1586 daemon    3u  IPv6  43197      0t0  TCP *:http (LISTEN)
httpd    1587 daemon    3u  IPv6  43197      0t0  TCP *:http (LISTEN)
httpd    1588 daemon    3u  IPv6  43197      0t0  TCP *:http (LISTEN)
httpd    1589 daemon    3u  IPv6  43197      0t0  TCP *:http (LISTEN)
httpd    1590 daemon    3u  IPv6  43197      0t0  TCP *:http (LISTEN)
httpd    1594 daemon    3u  IPv6  43197      0t0  TCP *:http (LISTEN)
[root@node1 ~]#
```

源码包安装 Apache 默认发布目录为:**/usr/local/apache2/htdocs/**下。

10.5 Apache 虚拟主机企业应用

修改 vi /usr/local/apache2/conf/extra/httpd-vhosts.conf 虚拟主机配置文件内容如下:

```
NameVirtualHost *:80

<VirtualHost *:80>

    ServerAdmin wgkgood@163.com

    DocumentRoot  "/data/webapps/www1"

    ServerName www.wugk1.com

  <Directory "/data/webapps/www1">

    AllowOverride All

    Options -Indexes FollowSymLinks

    Order allow,deny

    Allow from all
```

```
    </Directory>
        ErrorLog   logs/error_log
        CustomLog logs/access_log common
</VirtualHost>
<VirtualHost *:80>
        ServerAdmin wgkgood@163.com
        DocumentRoot "/data/webapps/www2"
        ServerName www.wugk2.com
    <Directory "/data/webapps/www2">
        AllowOverride All
        Options -Indexes FollowSymLinks
        Order allow,deny
        Allow from all
    </Directory>
        ErrorLog   logs/error_log
        CustomLog logs/access_log common
</VirtualHost>
```

参数详解：

1) 使用<Directory>…</Directory>设置指定目录的访问权限，其中可包含：

Options

AllowOverride

Order

Allow

Deny

如上五个属性，在后面可以跟如下参数设置：

2) AllowOverride 设置为 None 时，.htaccess 文件将被完全忽略。指令设置为 All 时，所有具有 ".htaccess" 作用域将生效.

3) Options -Indexes FollowSymLinks 禁止显示目录，如果要以目录显示需要改成 Options Indexes FollowSymLinks

4) Order allow,deny ,默认情况下禁止所有客户机访问;Order deny,allow 默认情况下允许所有客户机访问。

5) **Allow from all** 允许所有客户机访问。

6) **NameVirtualHost** 指令，指定一个基于域名的虚拟主机将使用哪个 IP 地址来接受请求。

10.6 Apache Rewrite 规则讲解

Rewirte 主要的功能就是实现 URL 的跳转，它的正则表达式是基于 Perl 语言。可基于服务器级的(httpd.conf)和目录级的（.htaccess)两种方式。如果要想用到 rewrite 模块，必须先安装或加载 rewrite 模块。方法有两种一种是编译 apache 的时候就直接 安装 rewrite 模块，别一种是编译 apache 时以 DSO 模式安装 apache,然后再利用源码和 apxs 来安装 rewrite 模块。

基于服务器级的(httpd.conf)有两种方法，一种是在 httpd.conf 的全局下 直接利用 RewriteEngine on 来打开 rewrite 功能;另一种是在局部里利用 RewriteEngine on 来打开 rewrite 功能,下面将会举例说明，需要注意的是,必须在每个 virtualhost 里用

RewriteEngine on 来打开 rewrite 功能。否则 virtualhost 里没有 RewriteEngine on 它里面的规则也不会生效。

基于目录级的(.htaccess),要注意一点那就是必须打开此目录的 FollowSymLinks 属性且在.htaccess 里要声明 RewriteEngine on。

Rewrite URL 重定向就是实现 URL 的跳转和隐藏真实地址，可以把复杂的 URL 变成简洁直观的 URL，对 seo 优化有很大的帮助。如下几个简单的举例：

把所有配置的域名都跳转到一个域名：

RewriteEngine on　　//启用 rewrite 引擎

RewriteCond　%{HTTP_HOST} ^wugk1.com [NC]　　//匹配以 wugk1.com 开头的域名,NC 忽略大小写。

RewriteRule ^/(.*)$　http://www.wugk1.com/$1　[L]

//匹配上面条件，然后跳转到 http://www.wugk1.com

1) R 强制外部重定向。

2) F 禁用 URL,返回 403HTTP 状态码。

3) G 强制 URL 为 GONE，返回 410 HTTP 状态码。

4) P 强制使用代理转发。

5) L 表明当前规则是最后一条规则，停止分析以后规则的重写。

6) N 重新从第一条规则开始运行重写过程。

7) C 与下一条规则关联。

10.7 Apache 必备目录功能

1) |- /etc/httpd/conf/httpd.conf → (Apache 的主配文件)

httpd.conf 是 Apache 的主配文件，整个 Apache 也不过就是这个配置文件，里面几乎包含了所有的配置。有的 distribution 都将这个文件拆分成数个小文件分别管理不同的参数。但是主要配置文件还是以这个文件为主。只要找到这个文件名就知道如何设置了。后面会详细解释 Apache 主配文件的每一行配置

2) |- /etc/httpd/conf.d/*.conf ((include 文件)

如果你不想要修改原始配置文件 httpd.conf 的话，那么可以将你自己的额外参数文件独立出来，注意以.conf 结尾放在/etc/httpd/conf.d/目录下。重启 Apache 的时候，这个配置文件就会被读入主配文件之中了。他的好处就是当你在进行系统升级的时候，几乎不需要改动原本的配置文件，只要将你自己的额外参数文件复制到正确的地点即可，维护起来非常方便。

3) |- /usr/lib64/httpd/modules/

|- /etc/httpd/modules/

Apache 支持很多的外挂模块，例如 PHP 以及 SSL 都是 Apache 外挂的一种。所有你想要使用的模块文件默认是放置在这个目录当中的。

4) |- /var/www/html/ ((网站根目录)

这个目录就是 Apache 默认的存放首页的目录（默认是 index.html）

5) |- /var/www/error/

当因为服务器设置错误，或是浏览器要求的数据错误时，在浏览器上出现的错误信息就以这个目录的默认信息为主

|- /var/www/icons/

这个目录提供 Apache 默认给予的一些小图示，可以随意使用。

|- /var/www/cgi-bin/

默认给一些可执行的 CGI（网页程序）程序放置的目录。

6) |- /var/log/httpd/

默认的 Apache 日志文件都放在这里，对于流量比较大的网站来说，这个目录要格外注意，这里的数据文件可能会非常大。

7) |- /usr/sbin/apachectl

这个就是 Apache 的主要执行文件，这个执行文件其实就是一个 Shell Script 而已，他可以主动地侦测系统上面的一些设置值，好让你启动 Apache 时更简单一些。

8) |- /usr/sbin/httpd

这个是主要的 Apache 二进制执行文件。

9) |- /usr/bin/htpasswd ((Apache 密码保护)

在当你想要登入某些网页时你需要输入帐号与密码，那 Apache 本身就提供一个最基本的密码保护方式，该密码的产生就是通过这个命令来实现的。

10.8 Apache 配置文件详解及优化

ServerTokens OS

#这个项目仅仅是在告知客户端我们服务器的版本和操作系统而已，不需要改动他

#如果不在乎你的系统信息被远程用户查询到，则可以将这个项目注释掉（不建议）

ServerRoot "/etc/httpd"

#服务器设置的最顶层目录，有点类似于 chroot 那种感觉。包括 logs , modules 等

#的数据都应该要放置在此目录下面（如果这些配置没有声明成绝对路径的话）

PidFile run/httpd.pid

#放置 PID 的文件，可方便 apache 软件的管理。只有相对路径考虑 ServerRoot 设置值，

#所以文件在/etc/httpd/run/httpd.pid

Timeout 60

#不论接收或发送，当持续连接等待超过 60 秒则该次连接就中断

#一般来说，此数值在 300 秒左右即可，不需要修改这个原始值

KeepAlive Off

#这里最好把默认值“Off”修改为“On”

#这里表示是否允许持续性的连接，也就是一个 TCP 连接可以具有多个文件资料传送的要求

#举例来说，如果你的网页内含有很多图片文件，那么这一次连接就会将所有的数据传送完

#而不必每一个图片都需要进行一次 TCP 连接。

MaxKeepAliveRequests 100

#可以将默认的 100 改成 500 或更高

#与上一个设置的值 KeepAlive 有关，当 KeepAlive 的值设置为 On 的时候，这个数值可以

决定

#该次连接能够传输的最大传输数量。为了提高效率则可以改大一点。0 代表不限制

KeepAliveTimeout 65

#在 KeepAlive 设置为“On”的情况下，该次连接在最后一次传输后等待延迟的秒数

#当超过该秒数的时候该连接中断。保持默认值 15 即可，如果设置的值太高（等待时间较

长）

#在较忙碌的系统上面将会有较多的 Apache 程序占用资源，可能有效率方面的问题。

<IfModule prefork.c>

StartServers 8 #启动 Apache 的时候,唤醒几个 PID 来处理服务的。

#Apache 使用了进程预派生的技术来处理请求，大大提高了响应速度，

MinSpareServers 5 #最小预备使用的 PID 数量

MaxSpareServers 20 #最大预备使用的 PID 数量

ServerLimit 4096 #服务器的限制

MaxClients 4096 #最多可以有多少个客户端同时连接到 Apache

#最大的同时连接数量，也就是 process 不会超过这一数值。

#这个 MaxClients 设置值可以控制同时连上 www 服务器的总连接要求数量，

#也可以将其看作是最高实时在线人数。不过要注意的是：这个值并非越大越好

#因为他会消耗物理内存（与 process 有关），所以如果你设置太高导致超出物理内存

#能够容许的范围，那么效率就会大大降低（因为会跑 SWAP），此外，MaxClients 也在

#Apache 编译的时候就指定最大值了，所以你也无法超出系统最大值，除非你重新编译

Apache

MaxRequestsPerChild 4000

#每个程序能够提供的最大传输次数要求。举例来说：如果有个用户连上服务器之后，

#要求数百个网页，当他的要求数量超过这个值的时候则该程序会被丢弃，

#另外切换一个新程序。这个设置可以有效地管理每个 process 在系统上存活的时间。

#根据观察所得，新程序的效果较好。

</IfModule>

<IfModule worker.c>

StartServers 8

MaxClients 4000

MinSpareThreads 25

MaxSpareThreads 75

ThreadsPerChild 75

MaxRequestsPerChild 0

</IfModule>

#上面的 prefork 和 worker 其实就是两个与服务器连接资源有关的设置项目。

#默认的项目配置对于一般中小型网站来说已经很够用了，不过如果网站的流量

#比较大，也可以修订一下里面的数值。这两个模块都是用在提供用户连接资源，

#设置的数值越大代表系统会启动越多的程序来提供 Apache 的服务，反映速度就比较快

#Redhat 和 CentOS 将这两个模块分别放到了不同的执行文件中，分别是

|- /usr/sbin/httpd 使用 prefork 模块

|- /usr/sbin/httpd.worker 使用 worker 模块

#/etc/sysconfig/httpd 这个文件决定了 Apache 使用哪一个模块，可以通过。

#修改这个文件来切换不同的工作模式。

LoadModule cgi_module modules/mod_cgi.so

LoadModule version_module modules/mod_version.so

#Apache 提供了非常多的模块供我们使用，以上就是加载的模块

Include conf.d/*.conf

ServerAdmin root@localhost

#系统管理员的邮箱，当网站出现问题的时候，错误信息会显示的联系邮箱

DocumentRoot "/var/www/html"

#上面这一行的配置指定了放置首页的目录

<Directory />

    Options FollowSymLinks

    AllowOverride None

</Directory>

#Directory 指定后面的路径是系统中的绝对路径

#这个设置是针对 www 服务器的默认环境而来的，因为是针对“/”的设置

#建议保留上面的默认值

<Directory "/var/www/html">

#使用 Directory 指定了一个绝对路径的目录

Options -Indexes FollowSymLinks

#Options（目录参数）

#此设置值表示在这个目录内能够让 Apache 进行的操作，也就是针对 Apache 的程序的权限设置。

#主要的参数值有：

Indexes：如果在此目录下找不到首页文件（默认为 index.html）时，

就显示整个目录下的文件名，至于首页文件名则与 DirectoryIndex 设置的值有关

建议注释掉 Indexes

FollowSymLinks：这是 Fllow Symolic Links 的缩写，字面意义是让连接文件可以生效。

我们知道首页的目录是在/var/www/html，既然是 WWW 的根目录，理论上就像被 chroot

一般。一般说来说被 chroot 的程序无法离开其目录，也就是说，默认的情况下，你在

/var/www/html 下面的连接文件只要链接到非此目录的其他地方，则该连接文件默认是失效的。

但是使用这个设置可以让链接有效的离开本目录

ExecCGI：让此目录具有执行 CGI 的权限，非常重要。举例来说，OpenWebMail 使用了

很多 Perl 程序，你要让 OpenWebMail 可以执行，就需要在该程序所在目录拥有 ExecCGI

的权限才行。但是要注意：不要让所有的目录均可以使用 ExecCGI

Includes：让一些 Server-Side Include 程序可以运行。建议可以加上去

MultiViews：这个有点像是多国语言的支持，与语言数据有关。在错误信息的回报内容中

最常见，在同一台主机中，可以依据客户端的语言而给予不同的语言显示。默认在回报

信息中存在，你可以检查一下/var/www/error/目录下的数据。

AllowOverride None

#允许覆盖参数功能

#表示是否允许额外配置文件.htaccess 的某些参数覆盖。在 httpd.conf 内设置好所有的权限

#不过这样一来，若用户自己的个人网页想要修改权限时将会对管理员造成困扰。因此，Apache 默认

#可以让用户以目录下的.htaccess 文件内覆盖<Direcoty>内的某些功能参数。这个项目则是在规定

#.htaccess 可以覆盖的权限类型有哪些。常见的有以下几种：

#　ALL：全部的权限均可以覆盖

#　AuthConfig：仅有网页认证（帐号与密码）可以覆盖

#　Indexes：仅允许 Indexes 方面的覆盖

#　Limits：允许用户利用 Allow、Deny 与 Order 管理可浏览的权限

#　None：不可覆盖，也就是让.htaccess 文件失效

#使用.htaccess 会严重影响到 Apache 的性能，如果不是特殊需要，建议关闭

#

Controls who can get stuff from this server.

#

    Order allow,deny

    Allow from all

#能否登陆浏览的权限

#决定此目录是否可被 Apache 的 PID 所浏览的权限设置。

#能否被浏览主要有两种判断的方式：

#　deny,allow　以 deny 优先处理，但没有写入规则的默认为 allow

#　allow,deny　以 allow 为优先处理，但是没有写入规则的默认为 deny

#所以在默认的情况下，因为是 allow,deny 所以默认为 deny（不可浏览）

#不过在下一行有个 allow from all，allow 优先处理，因此全部客户端均可浏览

10.9 Apache 配置文件权限操作

#在权限配置一块让人一头雾水，下面我整理了一下相关的文档，希望能拨开云雾

#Apache 内部的 Order 可以处理相关权限的限制，其中有两个值，Allow 和 Deny

#Order deny,allow　可以理解为拒绝所有，开放特定

#Order allow,deny　可以理解为开放所有，拒绝特定

#当 allow 与 deny 中有重复的规则出现，则最后一条的配置起到了决定性的作用

#举个例子来说，我们要允许所有人访问除了 192.168.61.61

#Order allow,deny

#allow from all

#deny 192.168.61.61

#以上这个例子很明显是允许所有拒绝特定的配置

#第二行定义了允许的规则，开放所有

#第三行定义了拒绝的规则，拒绝了一个 IP，这个 IP 包含在第二行的 all 当中，

#所以它的权限就默认由最后一行配置决定，最后一行是 deny，所以 61.61 被成功被拒之门外

#接下来举一个只允许 1.1 访问的例子

#Order deny,all

#deny from all

#allow 192.168.1.1

#这个例子的第一行声明了它是拒绝所有而允许特定

#第二行拒绝了所有访问

#第三行配置了一个允许的 IP，这个 IP 当然也是包含在第二行的 all 之中，出现了重复定义

#所以它的规则默认按照最后一条执行，最后一条是放行，所以 1.1 成功被释放

```
</Directory>


</IfModule>
DirectoryIndex   jfedu.php index.html index.html.var
#网站默认的首页文件的名称
#如果客户端在地址栏中只输入到目录，例如 http://localhost/时，那么 Apache
#将会拿哪一个文件作为首页来显示呢？这个文件的文件名就是在这里定义的了
#如果上面的文件全部存在的话，就会按照设置的顺序显示排在最前面的首页
#这个与之前在 Option 中谈到的 Indexes 有关。
Alias /icons/ "/var/www/icons/"
#Alias  网址延伸  实际 Linux 目录
#制作了一个目录的别名


<Directory "/var/www/icons">
    Options Indexes MultiViews FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
#这个 Alias 很有趣，可以制作出类似于链接文件的东西。当你输入 http://localhost/icons
时
#其实你的/var/www/html/中并没有 icons 那个目录，不过由于 Alias 别名的关系，会让
```

该网址

#直接链接到/var/www/icons/下

#因为设置了一个新的可浏览的目录，所以多了个<Dirctory>来限定权限

#

ScriptAlias /cgi-bin/ "/var/www/cgi-bin/"

#ScriptAlias　网址延伸　实际 Linux 目录

<Directory "/var/www/cgi-bin">

    AllowOverride None

    Options None

    Order allow,deny

    Allow from all

</Directory>

#与上面的 icons 类似，不过这边却是以 ScriptAlias（可执行脚本的别名）为设置值

#这个设置值可以指定该目录下面为“具有 ExecCGI 能力”的目录所在。所以你可以

#将类似 OpenWebMail 的程序放置到/var/www/cgi-bin/内，就不必额外设置其他的

#目录来放置你的 CGI 程序了。

Some examples:

#ErrorDocument 500 "The server made a boo boo."

#ErrorDocument 404 /missing.html

#ErrorDocument 404 "/cgi-bin/missing_handler.pl"

#ErrorDocument 402 http://www.example.com/subscription_info.html

#

#错误页面的设置

#404 的第一个例子中使用的是相对路径，必须放置在首页目录下

#402 的例子直接跳转到了另一个页面

#NameVirtualHost *:80

#如果需要开启虚拟主机的功能，需要首先把这一行的注释拿掉

#这里的*号意味着监听每一个 IP 的 80 端口

#<VirtualHost *:80>

#这里一定要注意，这个标签后面的 IP 定义一定要和上面 NameVirtualHost 定义的一致

#　　ServerAdmin webmaster@dummy-host.example.com

#这一行不是必须的，指定了管理员的邮箱

#　　DocumentRoot /www/docs/dummy-host.example.com

#这一行是必须的配置，它指定这个虚拟主机的网页存放目录

#　　ServerName dummy-host.example.com

#ServerName 是必须要有的配置，这里需要定义一个能被 DNS 或 Hosts 解析的完整域名

#　　ErrorLog logs/dummy-host.example.com-error_log

#以上这一行的配置意为将这个主机发生的错误日志写入到指定的文件中，而不是写入到默认的文件中

#默认的错误日志路径为：/var/log/httpd/errorlog

#　　CustomLog logs/dummy-host.example.com-access_log common

#以上这一行的配置意为将访问日志写入到指定的文件中，而不写入到默认的 /var/log/httpd/accesslog 中

#</VirtualHost>

#上面一块是一个虚拟主机的配置实例

#在虚拟主机之上还能设置很多的功能，不过最精简的配置是需要有

#ServerName 和 DocumentRoot 这两个配置

#在添加了虚拟主机的配置之后需要把中心主机的信息也单独写成一个虚拟机的配置

#不然中心主机名的服务就不知道丢到哪里去了，这一步骤在 Apache 帮助文档中称做：取消中心主机

#当然，如果你想有多个域名都指向到同一个虚拟主机是可以通过在<VirtualHost>块中

#配置 ServerAlias 功能来实现的。

#注意点一：DNS 指向问题

#　如果你设置了别名，多个域名指向同一个虚拟主机，那么一定要保证 DNS 能正常解析的到

#注意点二：<VirtualHost>段配置指定作用域的问题

#　你可以把其他一些指令放入<VirtualHost>段中，以更好的配置一个虚拟主机。

#　大部分指令都可以放入这些<VirtualHost>段中以改变相应虚拟主机配置。

#　如果您想了解一个特定的指令是否可以这样运用，请查看帮助手册中指令的作用域。

#　主服务器(main server)范围内的配置指令(在所有<VirtualHost>配置段之外的指令)

#　仅在它们没有被虚拟主机的配置覆盖时才起作用。

#注意点三：虚拟主机名的问题

#　当一个请求到达的时候，服务器会首先检查它是否使用了一个能和 NameVirtualHost 相匹配

#　的 IP 地址。如果能够匹配，它就会查找每个与这个 IP 地址相对应的<VirtualHost>段，

#　并尝试找出一个与请求的主机名相同的 ServerName 或 ServerAlias 配置项。

如果找到了，它就会使用这个服务器。

否则，将使用符合这个 IP 地址的第一个列出的虚拟主机。

顺序展示：客户端发起一个访问域名的请求--->DNS 解析到目标主机

--->检查是否开启了虚拟主机的功能--->检查是否能和 NameVirtualHost 相匹配

--->查找出每个与该 IP 对应的虚拟主机段配置--->尝试找出与请求的完整域名相同的 ServerName 或 ServerAlias

--->如果找到就使用这个虚拟主机的配置--->如果配置中与中心主机的配置不冲突则优先使用中心主机的配置

如果找不到与之相匹配的完整域名的虚拟主机配置--->使用符合这个 IP 地址的第一个虚拟主机

#综上所述，第一个列出的虚拟主机充当了默认虚拟主机的角色。

#当一个 IP 地址与 NameVirtualHost 指令中的配置相符的时候，

#主服务器中的 DocumentRoot 将永远不会被用到。

#所以，如果你想创建一段特殊的配置用于处理不对应任何一个虚拟主机的请求的话，

#你只要简单的把这段配置放到<VirtualHost>段中，并把它放到配置文件的最前面就可以了。

第11章 构建 MySQL 数据库企业实战篇

11.1 Mysql 数据库入门及简介

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 旗下公司。MySQL 最流行的关系型数据库管理系统，在 WEB 应用方面

MySQL 是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件之一。

MySQL 是一种关联数据库管理系统，关联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 软件采用了双授权政策，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 作为网站数据库。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。

（RDBMS 即关系数据库管理系统(Relational Database Management System)，是将数据组织为相关的行和列的系统，而管理关系数据库的计算机软件就是关系数据库管理系统，常用的数据库软件有 Oracle、SQL Server 等。）

RDBMS 的特点：

1) 数据以表格的形式出现；

2) 每行为各种记录名称；

3) 每列为记录名称所对应的数据域；

4) 许多的行和列组成一张表单；

5) 若干的表单组成 database；

对应目前主流的 LAMP(Linux+Apache+Mysql+PHP)架构来说，Mysql 更是得到各位 IT 运维、DBA 的青睐，目前 mysql 已被 orcacle 收购，不过好消息是原来 mysql 创始人已独立出来自己重新开发了一个 MariaDB，而且使用的人数越来越多。而且 MariaDB 兼容 mysql 所有的功能和相关参数。

11.2 Mysql 数据库引擎详解

MySQL 是我们比较常用的一种数据库软件。它有着诸多的优点，如开源的，免费的等等。其实它还有一个很好的特点，那就是有多种引擎可以供你选择。如果赛车手能根据不同的路况，地形随手更换与之最适宜的引擎，那么他们将创造奇迹。

MyISAM MySQL 5.0 之前的默认数据库引擎，最为常用。拥有较高的插入，查询速度，但不支持事务；

InnoDB 事务型数据库的首选引擎，支持 ACID 事务，支持行级锁定, MySQL 5.5 起成为默认数据库引擎；

BDB 源自 Berkeley DB，事务型数据库的另一种选择，支持 Commit 和 Rollback 等其他事务特性；

Memory 所有数据置于内存的存储引擎，拥有极高的插入，更新和查询效率。但是会占用和数据量成正比的内存空间。并且其内容会在 MySQL 重新启动时丢失；

Mysql 常用的两大引擎有 MyISAM 和 innoDB，那他们有什么明显的区别呢，什么场合使用什么引擎呢?

MyISAM 类型的表强调的是性能，其执行数度比 InnoDB 类型更快，但不提供事务支持，如果执行大量的 SELECT(查询)操作，MyISAM 是更好的选择，支持表锁。

InnoDB 提供事务支持事务，外部键等高级 数据库功能，执行大量的 INSERT 或 UPDATE，出于性能方面的考虑，应该使用 InnoDB 表，支持行锁。

11.3 Mysql 数据库应用索引

索引是一种特殊的文件（InnoDB 数据表上的索引是表空间的一个组成部分），它们包含着对数据表里所有记录的引用指针。索引不是万能的，索引可以加快数据检索操作，但会

使数据修改操作变慢。每修改数据记录，索引就必须刷新一次。常见索引类别如下：

1) 普通索引

普通索引（由关键字 KEY 或 INDEX 定义的索引）的唯一任务是加快对数据的访问速度。因此，应该只为那些最经常出现在查询条件（WHERE column =）或排序条件（ORDER BY column）中的数据列创建索引。只要有可能，就应该选择一个数据最整齐、最紧凑的数据列（如一个整数类型的数据列）来创建索引。

2) 唯一索引

普通索引允许被索引的数据列包含重复的值。比如说，因为人有可能同名，所以同一个姓名在同一个“员工个人资料”数据表里可能出现两次或更多次。

如果能确定某个数据列将只包含彼此各不相同的值，在为这个数据列创建索引的时候就应该用关键字 UNIQUE 把它定义为一个唯一索引。这么做的好处：一是简化了 MySQL 对这个索引的管理工作，这个索引也因此而变得更有效率；二是 MySQL 会在有新记录插入数据表时，自动检查新记录的这个字段的值是否已经在某个记录的这个字段里出现过了；如果是，MySQL 将拒绝插入那条新记录。也就是说，唯一索引可以保证数据记录的唯一性。事实上，在许多场合，人们创建唯一索引的目的往往不是为了提高访问速度，而只是为了避免数据出现重复。

3) 主索引

在前面已经反复多次强调过：必须为主键字段创建一个索引，这个索引就是所谓的“主索引”。主索引与唯一索引的唯一区别是：前者在定义时使用的关键字是 PRIMARY 而不是 UNIQUE。

4) 外键索引

如果为某个外键字段定义了一个外键约束条件，MySQL 就会定义一个内部索引来帮

助自己以最有效率的方式去管理和使用外键约束条件。

5) 复合索引

索引可以覆盖多个数据列，如像 INDEX (columnA, columnB) 索引。这种索引的特点是 MySQL 可以有选择地使用一个这样的索引。如果查询操作只需要用到 columnA 数据列上的一个索引，就可以使用复合索引 INDEX(columnA，columnB)。不过，这种用法仅适用于在复合索引中排列在前的数据列组合。比如说，INDEX (A，B，C) 可以当做 A 或 (A,B) 的索引来使用，但不能当做 B、C 或 (B,C) 的索引来使用。

11.4 Mysql 数据库安装方式

MySQL 安装方式有两种，一种是 yum/rpm 安装，另外一种是 tar 源码安装。

Yum 安装方法很简单，执行命令如下即可:

yum install –y mysql-server mysql-devel mysql

源码安装 MySQL 方式:

cd /usr/src ;

wget http://downloads.mysql.com/archives/mysql-5.1/mysql-5.1.63.tar.gz ;tar xzf mysql-5.1.63.tar.gz ;cd mysql-5.1.63 ;./configure --prefix=/usr/local/mysql --enable-assembler &&make -j8&&make -j8 install

```
make[3]: Nothing to be done for `install-data-am'.
make[3]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
make[2]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
Making install in instance-manager
make[2]: Entering directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[3]: Entering directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
test -z "/usr/local/mysql/libexec" || /bin/mkdir -p "/usr/local/mysql/libexec"
  /bin/sh ../../libtool --preserve-dup-deps   --mode=install /usr/bin/install -c 'mysqlmanager' '/usr/local/m
r'
libtool: install: /usr/bin/install -c mysqlmanager /usr/local/mysql/libexec/mysqlmanager
make[3]: Nothing to be done for `install-data-am'.
make[3]: Leaving directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[2]: Leaving directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[1]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
Making install in win
make[1]: Entering directory `/usr/src/mysql-5.1.63/win'
make[2]: Entering directory `/usr/src/mysql-5.1.63/win'
make[2]: Nothing to be done for `install-exec-am'.
make[2]: Nothing to be done for `install-data-am'.
make[2]: Leaving directory `/usr/src/mysql-5.1.63/win'
make[1]: Leaving directory `/usr/src/mysql-5.1.63/win'
[root@node2 mysql-5.1.63]#
[root@node2 mysql-5.1.63]# ./configure  --prefix=/usr/local/mysql --enable-assembler &&make &&make install
```

配置 Mysql 服务为系统服务：

cp /usr/local/mysql/share/mysql/my-medium.cnf /etc/my.cnf

cp /usr/local/mysql/share/mysql/mysql.server /etc/rc.d/init.d/mysqld

chkconfig --add mysqld

chkconfig --level 35 mysqld on

service mysqld restart

源码启动方式：

cd /usr/local/mysql

useradd mysql

chown -R mysql.mysql /usr/local/mysql

/usr/local/mysql/bin/mysql_install_db --user=mysql

--datadir=/usr/local/mysql/var --basedir=/usr/local/mysql/

chown -R mysql.mysql var

/usr/local/mysql/bin/mysqld_safe --user=mysql &

11.5 Mysql 数据库必备命令操作

MySQL 日常操作命令：

show databases; 查看数据库

create database test_db; 创建名为 test_db 数据库

use test_db; 进入 test_db 数据库。

show tables; 查看数据库里有多少张表。

```
Server version: 5.0.95-log Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> create database test_db;
Query OK, 1 row affected (0.06 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| mysql_ab_test      |
| test               |
| test_db            |
+--------------------+
5 rows in set (0.13 sec)

mysql> use test_db;
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

create table test01 (id varchar(20),name varchar(20));创建名为 test01 表，并创建两个字段，id、name、数据长度（用字符来定义长度单位。）

insert into test01 values ("001","wugk1"); 向表中插入数据。

select * from test01; 查看 test01 表数据内容。

Select * from test01 where id=1 and age =’ jfedu’ ;多个条件查询。

desc test01;查看 test01 表字段内容；

alter table test01 modify column name varchar(20);修改 name 字段的长度；

update test01 set name='jfedu.net' where id=1; 修改 name 字段的内容；

```
mysql> create table test01 (id varchar(20),name varchar(20));
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+------------------+
| Tables_in_test_db |
+------------------+
| test01           |
+------------------+
1 row in set (0.00 sec)

mysql> insert into test01 values ("001","wugk1");
Query OK, 1 row affected (0.02 sec)

mysql> insert into test01 values ("002","wugk2");
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> select * from test01;
+------+-------+
| id   | name  |
+------+-------+
| 001  | wugk1 |
| 002  | wugk2 |
+------+-------+
2 rows in set (0.00 sec)
```

grant all on test_db.* to test@localhost identified by '123456';

grant all on test_db.* to test@localhost identified by '123456';

grant select,insert,update,delete on *.* to test@ ” % ” identified by ‘123456’ ;

给 mysql 数据库授权：

flush privileges;刷新权限

mysqldump –uroot –p123456

test_db >/tmp/test.db.sql ;MySQL 备份或导出

mysql –uroot –p123456 test_db < /tmp/test.db.sql ;

MySQL 导入

mysqladmin –uroot –p123456 password newpassword ;

修改 MySQL root 密码

drop database test_db；删除数据库

drop table test01； 删除表

delete from test01；清空表内容

show variables like '%char%'; 查看数据库字符集

11.6 Mysql 数据库字符集及密码破解

- **修改 Mysql 字符集为 UTF-8 的方法：**

在/etc/my.cnf 对应如下配置段加入相应命令。

[client]字段里加入 default-character-set=utf8

[mysqld]字段里加入 character-set-server=utf8

[mysql]字段里加入 default-character-set=utf8

然后重启 MySQL 服务即可。

- **Mysql 忘记密码如何破解：**

首先停止 mysql 服务，然后以跳过权限方式后台启动：

/usr/bin/mysqld_safe --user=mysql --skip-grant-tables &

然后执行 mysql 回车进入 mysql，然后修改密码。

```
[root@localhost ~]# /usr/bin/mysqld_safe --user=mysql --skip-grant-tables &
[2] 2103
[root@localhost ~]# 141126 08:40:58 mysqld_safe Logging to '/var/log/mysqld.log'.
141126 08:40:58 mysqld_safe Starting mysqld daemon with databases from /var/lib/mysql

[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

修改 Mysql 密码：

Use mysql 数据库，然后执行如下命令：

```
mysql> use mysql
Database changed
mysql> update user set password=password('123456') where user='root';
Query OK, 4 rows affected (0.00 sec)
Rows matched: 4  Changed: 4  Warnings: 0

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

修改完后，停止 Mysql，然后再以正常方式启动：

```
[root@localhost ~]# /etc/init.d/mysqld start
Starting MySQL.. SUCCESS!
[root@localhost ~]#
[root@localhost ~]# mysql -uroot -p123456
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.5.20 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql>
```

再次以新的密码登陆即可进入 Mysql 数据库。

11.7 Mysql 数据库参数详解(my.cnf)

[mysqld]　//服务器端配置

datadir=/data/mysql　//数据目录

socket=/var/lib/mysql/mysql.sock //socket 通信设置

user=mysql //使用 mysql 用户启动

Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0　//是否支持快捷方式

log-bin=mysql-bin　//开启 bin-log 日志

server-id = 1　//mysql 服务 ID

auto_increment_offset=1　　//

auto_increment_increment=2

(mysql 中有自增长字段，在做数据库的主主同步时需要设置自增长的两个相关配置：

auto_increment_offset 和 auto_increment_increment。

auto_increment_offset 表示自增长字段从那个数开始，他的取值范围是 1 -65535

auto_increment_increment 表示自增长字段每次递增的量，其默认值是 1，取值范围是 1-65535

在主主同步配置时，需要将两台服务器的 auto_increment_increment 增长量都配置为 2，而要把 auto_increment_offset 分别配置为 1 和 2,这样才可以避免两台服务器同时做更新时自增长字段的值之间发生冲突。)

[mysqld_safe]　//mysql 服务安全启动配置

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

key_buffer_size 指定索引缓冲区的大小，它决定索引处理的速度，尤其是索引读的速

度。一般为内存的 50%

show variables like ‘key_buffer_size‘;

max_connections = 3000

每个客户端连接最大的错误允许数量,如果达到了此限制,这个客户端将会被 MySQL 服务阻止直到执行了”FLUSH HOSTS”或者服务重启.

innodb_buffer_pool_size

对于 InnoDB 表来说，innodb_buffer_pool_size 的作用就相当于 key_buffer_size 对于 MyISAM 表的作用一样。InnoDB 使用该参数指定大小的内存来缓冲数据和索引。

对于单独的 MySQL 数据库服务器，最大可以把该值设置成物理内存的 80%。

内存 32G，24G

根据 MySQL 手册，对于 2G 内存的机器，推荐值是 1G（50%）。

basedir = path # 使用给定目录作为根目录(安装目录)。

datadir = path # 从给定目录读取数据库文件。

pid-file = filename # 为 mysqld 程序指定一个存放进程 ID 的文件(仅适用于 UNIX/Linux 系统);

[mysqld]

socket = /tmp/mysql.sock # 为 MySQL 客户程序与服务器之间的本地通信指定一个套接字文件(Linux 下默认是/var/lib/mysql/mysql.sock 文件)

port = 3306 # 指定 MsSQL 侦听的端口

key_buffer = 384M # key_buffer 是用于索引块的缓冲区大小，增加它可得到更好处理的索引(对所有读和多重写)。索引块是缓冲的并且被所有的线程共享，key_buffer 的大小视内存大小而定。

table_cache = 512 # 为所有线程打开表的数量。增加该值能增加mysqld 要求的文件描述符的数量。可以避免频繁的打开数据表产生的开销

sort_buffer_size = 2M # 每个需要进行排序的线程分配该大小的一个缓冲区。增加这值加速ORDER BY或GROUP BY操作。注意:该参数对应的分配内存是每连接独占!如果有 100 个连接，那么实际分配的总共排序缓冲区大小为 100×6=600MB

read_buffer_size = 2M # 读查询操作所能使用的缓冲区大小。和sort_buffer_size 一样，该参数对应的分配内存也是每连接独享。

query_cache_size = 32M # 指定 MySQL 查询结果缓冲区的大小

read_rnd_buffer_size = 8M # 改参数在使用行指针排序之后，随机读用的。

myisam_sort_buffer_size = 64M # MyISAM 表发生变化时重新排序所需的缓冲

thread_concurrency = 8 # 最大并发线程数，取值为服务器逻辑 CPU 数量×2，如果 CPU 支持 H.T 超线程，再×2

thread_cache = 8 # #缓存可重用的线程数

skip-locking # 避免 MySQL 的外部锁定，减少出错几率增强稳定性。

[mysqldump]

max_allowed_packet =16M # 服务器和客户端之间最大能发送的可能信息包

11.8 MySQL 引擎 MyISAM 与 InnoDB

1) MyISAM 引擎:

默认表类型，它是基于传统的 ISAM 类型,ISAM 是 Indexed Sequential Access Method (有索引的顺序访问方法) 的缩写,它是存储记录和文件的标准方法.不是事务安全的，而且不支持外键，如

果执行大量的 select， MyISAM 比较适合。

2) InnoDB 引擎：

支持事务安全的引擎，支持外键、行锁、事务是他的最大特点。Innodb 最初是由 innobase Oy 公司开发，2005 年 10 月由 oracle 公司并购，目前 innodb 采用双授权，一个是 GPL 授权，一个是商业授权。如果有大量的 update 和 insert，建议使用 InnoDB,特别是针对多个并发和 QPS 较高的情况。

3) 总体来讲：

InnoDB 和 MyISAM 是在使用 MySQL 最常用的两个表类型，各有优缺点，视具体应用而定。基本的差别为：MyISAM 类型不支持事务处理等高级处理，而 InnoDB 类型支持。MyISAM 类型的表强调的是性能，其执行数度比 InnoDB 类型更快，但是不提供事务支持，而 InnoDB 提供事务支持已经外部键等高级数据库功能。

① MyISAM 适合：(1)做很多 count 的计算；(2)插入不频繁，查询非常频繁；(3)没有事务。

② InnoDB 适合：(1)可靠性要求比较高，或者要求事务；(2)表更新和插入都相当的频繁，并且表锁定的机会比较大的情况。

4) 如何查看数据库引擎：

一般情况下，MySQL 会默认提供多种存储引擎，可以通过下面的查看:

① 查看 MySQL 现在已提供什么存储引擎: mysql> show engines;

② 查看 MySQL 当前默认的存储引擎: mysql> show variables like '%storage_engine%';

③ 查看某个表用了什么引擎(在显示结果里参数 engine 后面的就表示该表当前用的存储引擎): mysql> show create table 表名;

```
mysql> show engines;
+------------+---------+----------------------------------------------------------------+------
| Engine     | Support | Comment                                                        | Trans
+------------+---------+----------------------------------------------------------------+------
| CSV        | YES     | CSV storage engine                                             | NO
| MRG_MYISAM | YES     | Collection of identical MyISAM tables                          | NO
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables      | NO
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance         | NO
+------------+---------+----------------------------------------------------------------+------
4 rows in set (0.00 sec)
```

查看表使用的引擎：show create table test_t0;

```
mysql> show create table test_t0;
+---------+-----------------------------------------------------
--------------------+
| Table   | Create Table
                     |
+---------+-----------------------------------------------------
--------------------+
| test_t0 | CREATE TABLE `test_t0` (
  `id` varchar(20) DEFAULT NULL,
  `name` varchar(20) DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
+---------+-----------------------------------------------------
--------------------+
1 row in set (0.00 sec)

mysql>
```

5) 修改 MySQL 表空间引擎：

设置 InnoDB 为默认引擎：在配置文件 my.cnf 中的 [mysqld] 下面加入 default-storage-engine=INNODB 然后重启 mysqld 服务即可。

可以修改表引擎方法如下：

alter table t1 engine=myisam;

alter table t1 engine=innodb;

如果添加 innodb 引擎报错，需要执行如下命令：

删除/mysql/data 目录下的 ib_logfile0,ib_logfile1 文件即可。

11.9 MySQL 索引及慢查询案例讲解

MySQL 索引用来快速地寻找那些具有特定值的记录，所有 MySQL 索引都以 B-树的形式保存。如果没有索引，执行查询时 MySQL 必须从第一个记录开始扫描整个表的所有记录，直至找到符合要求的记录。表里面的记录数量越多，这个操作的代价就越高。

如果作为搜索条件的列上已经创建了索引，MySQL 无需扫描任何记录即可迅速得到目标记录所在的位置。如果表有 1000 个记录，通过索引查找记录至少要比顺序扫描记录快 100 倍。

常见索引类型：

1) normal：表示普通索引
2) unique：表示唯一的，不允许重复的索引，如果该字段信息保证不会重复例如身份证号用作索引时，可设置为 unique
3) full text: 表示 全文搜索的索引。 FULLTEXT 用于搜索很长一篇文章的时候，效果最好。用在比较短的文本，如果就一两行字的，普通的 INDEX 也可以。

总结，索引的类别由建立索引的字段内容特性来决定，通常 normal 最常见。

创建索引命令：

ALTER TABLE 用来创建普通索引、UNIQUE 索引或 PRIMARY KEY 索引。（我这里以 t1 表为例来讲解）

ALTER TABLE t1 ADD INDEX index_name (name)

ALTER TABLE t1 ADD UNIQUE (name)

ALTER TABLE t1 ADD PRIMARY KEY (name)

或者使用 create 创建

CREATE INDEX index_name ON t1 (name)

CREATE UNIQUE INDEX index_name ON t1 (name)

删除索引

DROP INDEX index_name ON talbe_name

ALTER TABLE t1 DROP INDEX index_name

ALTER TABLE t1 DROP PRIMARY KEY;

查看索引

show index from t1;

show keys from t1;

MySQL 慢查询：

慢查询对于跟踪有问题的查询很有用，可以分析出当前程序里那些 Sql 语句比较耗费资源。

1）查看当前 mysql 慢查询

show variables like "%slow%";

```
mysql>
mysql> show variables like  '%slow%';
+---------------------+---------------------------------+
| Variable_name       | Value                           |
+---------------------+---------------------------------+
| log_slow_queries    | OFF                             |
| slow_launch_time    | 2                               |
| slow_query_log      | OFF                             |
| slow_query_log_file | /var/lib/mysql/wugkapp-slow.log |
+---------------------+---------------------------------+
4 rows in set (0.00 sec)

mysql>
```

| slow_launch_time | 2 | 超过 2 秒定义为慢查询。

| slow_query_log | OFF | 慢查询关闭状态。

| slow_query_log_file | /data/mysql/var/db-Test2-slow.log | 慢查询日志的文件。

2）开启慢查询日志方法

Mysql 数据库里执行：set global slow_query_log=on;

在 my.cnf 中添加，如下：

log-slow-queries = /data/mysql/var/db-Test2-slow.log #日志目录。

long_query_time = 0.1 #记录下查询时间查过 1 秒。

log-queries-not-using-indexes #表示记录下没有使用索引的查询。

3）mysqldumpslow 分析日志

可用 mysql 提供的 mysqldumpslow，使用很简单，参数可-help 查看

-s：排序方式。

c ， t ， l ， r 表示记录次数、时间、查询时间的多少、返回的记录数排序；

ac ， at ， al ， ar 表示相应的倒叙；

-t：返回前面多少条的数据；

-g：包含什么，大小写不敏感的；

mysqldumpslow -s r -t 10 /data/mysql/var/db-Test2-slow.log

第12章 MYSQL&LAMP 架构优化实战篇

12.1 MySQL 数据库配置并发优化

Mysql 优化是一项非常重要的工作，而且是一项长期的工作，曾经有一个为位 DBA 前辈说过:mysql 的优化，三分配置的优化，七分 sql 语句的优化。

Mysql 的优化：一般分为配置的优化、sql 语句的优化、表结构的优化、索引的优化，而配置的优化：一般包括系统内核优化、mysql 本身配置文件的优化。

MySQL 常见的优化参数详解：

硬件上的优化：增加内存和提高磁盘读写速度，都可以提高 MySQL 数据库的查询，更新的速度。另一种提高 MySQL 性能的方式是使用多块磁盘来存储数据。因为可以从多块磁盘上并行读取数据，这样可以提高读取数据的速度。

MySQL 参数的优化：内存中会为 MySQL 保留部分的缓冲区。这些缓冲区可以提高 MySQL 的速度。缓冲区的大小都是在 MySQL 的配置文件中进行设置的。

下面对几个重要的参数进行详细介绍：

1) key_buffer_size：表示索引缓存的大小。这个值越大，使用索引进行查询的速度就越快。
2) table_cache：表示同时打开的表的个数。这个值越大，能同时打开的表的个数就越多。这个值不是越大越好，因为同时打开的表过多会影响操作系统的性能。
3) query_cache_size：表示查询缓冲区的大小。使用查询缓存区可以提高查询的速度。这个方式只使用与修改操作少且经常执行相同的查询操作的情况；默认值是 0。
4) Query_cache_type：表示查询缓存区的开启状态。0 表示关闭，1 表示开启。
5) Max_connections：表示数据库的最大连接数。这个连接数不是越大越好，因为连接会浪费内存的资源。
6) sort_buffer_size：排序缓存区的大小，这个值越大，排序就越快。
7) Innodb_buffer_pool_size：表示 InnoDB 类型的表和索引的最大缓存。这个值越大，查询的速度就会越快。这个值太大了就会影响操作系统的性能。

当然了 Mysql 是一个长期的优化过程，所以在日常的运维工作中，需要不断去总结和学习。

附一个真实环境 MySQL 配置 my.cnf 内容，可以根据实际情况修改：

```
[client]
port = 3306
socket = /tmp/mysql.sock
[mysqld]
user = mysql
server_id = 10
port = 3306
socket = /tmp/mysql.sock
datadir = /data/mysql/data1
old_passwords = 1
lower_case_table_names = 1
character-set-server = utf8
default-storage-engine = MYISAM
log-bin = bin.log
log-error = error.log
pid-file = mysql.pid
long_query_time = 2
slow_query_log
slow_query_log_file = slow.log
binlog_cache_size = 4M
binlog_format = mixed
```

max_binlog_cache_size = 16M

max_binlog_size = 1G

expire_logs_days = 30

ft_min_word_len = 4

back_log = 512

max_allowed_packet = 64M

max_connections = 4096

max_connect_errors = 100

join_buffer_size = 2M

read_buffer_size = 2M

read_rnd_buffer_size = 2M

sort_buffer_size = 2M

query_cache_size = 64M

table_open_cache = 10000

thread_cache_size = 256

max_heap_table_size = 64M

tmp_table_size = 64M

thread_stack = 192K

thread_concurrency = 24

local-infile = 0

skip-show-database

skip-name-resolve

```
skip-external-locking
connect_timeout = 600
interactive_timeout = 600
wait_timeout = 600
#*** MyISAM
key_buffer_size = 512M
bulk_insert_buffer_size = 64M
myisam_sort_buffer_size = 64M
myisam_max_sort_file_size = 1G
myisam_repair_threads = 1
concurrent_insert = 2
myisam_recover
#*** INNODB
innodb_buffer_pool_size = 16G
innodb_additional_mem_pool_size = 32M
innodb_data_file_path = ibdata1:1G;ibdata2:1G:autoextend
innodb_read_io_threads = 8
innodb_write_io_threads = 8
innodb_file_per_table = 1
innodb_flush_log_at_trx_commit = 2
innodb_lock_wait_timeout = 120
innodb_log_buffer_size = 8M
```

```
innodb_log_file_size = 256M
innodb_log_files_in_group = 3
innodb_max_dirty_pages_pct = 90
innodb_thread_concurrency = 16
innodb_open_files = 10000
#innodb_force_recovery = 4
#*** Replication Slave
read-only
#skip-slave-start
relay-log = relay.log
log-slave-updates
```

12.2 MySQL 数据库集群拓展

MySQL 是一个开放源码的小型关联式数据库管理系统，开发者为瑞典 MySQL AB 公司，目前属于 Oracle 公司,MySQL 被广泛地应用在 Internet 上的中小型网站中。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，许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。

对应目前主流的 LAMP 架构来说,Mysql 更是得到各位 IT 运维、DBA 的青睐，目前 mysql 已被 orcacle 收购，不过好消息是原来 mysql 创始人已独立出来自己重新开发了一个 MariaDB，而且使用的人数越来越多。而且 MariaDB 兼容 mysql 所有的功能和相关参数。

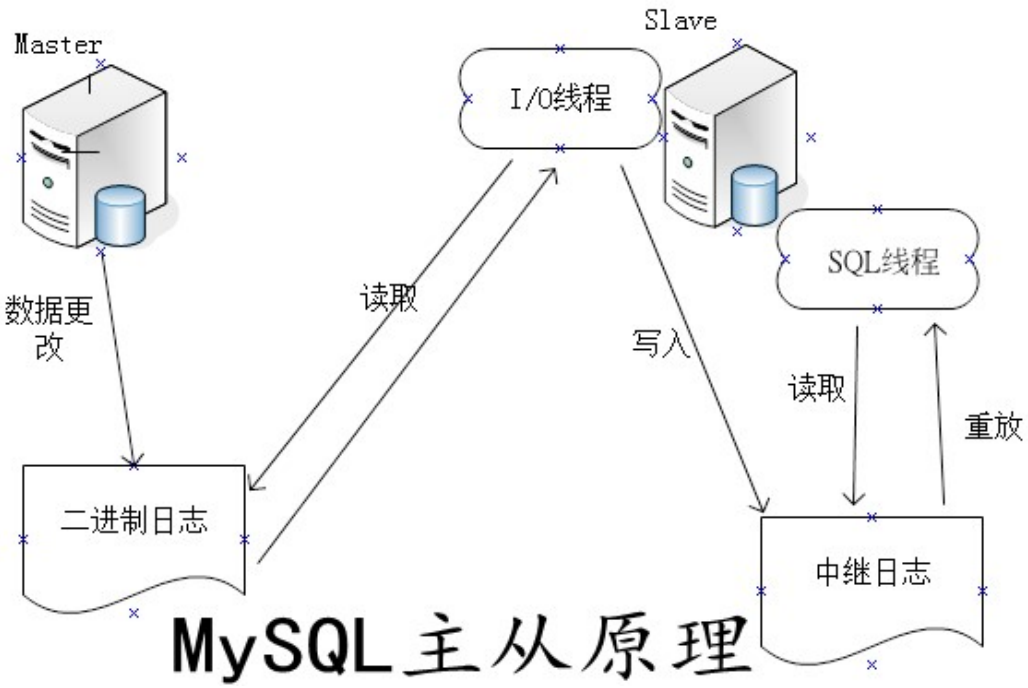
Mysql 常用的两大引擎有 MyISAM 和 innoDB，那他们有什么明显的区别呢，什么场

合使用什么引擎呢?

MyISAM 类型的表强调的是性能，其执行数 度比 InnoDB 类型更快，但不提供事务支持，如果执行大量的 SELECT 操作，MyISAM 是更好的选择，支持表锁。

InnoDB 提供事务支持事务，外部键等高级 数据库功能，执行大量的 INSERT 或 UPDATE，出于性能方面的考虑，应该使用 InnoDB 表，支持行锁。

随着访问量的不断增加，Mysql 数据库压力不断增加，需要对 mysql 进行优化和架构改造，可以使用高可用、主从复制、读写分离来、拆分库、拆分表进行优化。下面我们来学习**MySQL 主从复制**高可用如何来实现。



12.3 MySQL 主从复制原理剖析

Mysql 主从同步其实是一个异步复制的过程，要实现复制首先需要在 master 上开启 bin-log 日志功能，整个过程需要开启 3 个线程，分别是 Master 开启 IO 线程，slave 开启 IO 线程和 SQL 线程。

a) **在从服务器执行 slave start，从服务器上 IO 线程会通过授权的用户连接上 master，并请求 master 从指定的文件和位置之后发送 bin-log 日志内容。**

b) Master 服务器接收到来自 slave 服务器的 IO 线程的请求后，master 服务器上的 IO 线程根据 slave 服务器发送的指定 bin-log 日志之后的内容，然后返回给 slave 端的 IO 线程。（返回的信息中除了 bin-log 日志内容外，还有本次返回日志内容后在 master 服务器端的新的 binlog 文件名以及在 binlog 中的下一个指定更新位置。）

c) Slave 的 IO 线程接收到信息后，将接收到的日志内容依次添加到 Slave 端的 **relay-log** 文件的最末端，并将读取到的 Master 端的 bin-log 的文件名和位置记录到 master.info 文件中，以便在下一次读取的时候能够清楚的告诉 Master “我需要从某个 bin-log 的哪 个位置开始往后的日志内容，请发给我”；

d) Slave 的 Sql 线程检测到 **relay-log 中新增加了内容后**，会马上解析 relay-log 的内容成为在 Master 端真实执行时候的那些可执行的内容，并在自身执行。

12.4 MySQL 主从复制架构实战

环境准备：192.168.1.103 为 master 主服务器，192.168.33.11 为 slave 从服务器。

在主和从服务器都安装 mysql 相关软件，命令如下：

```
yum  install -y  mysql mysql-devel mysql-server mysql-libs
```

安装完毕后，在 Master 修改 vi /etc/my.cnf 内容为如下：

```
[mysqld]
 datadir=/data/mysql
 socket=/var/lib/mysql/mysql.sock
```

user=mysql

Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

log-bin=mysql-bin

server-id = 1

auto_increment_offset=1

auto_increment_increment=2

[mysqld_safe]

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

replicate-do-db =jfedu

创建/data/mysql 数据目录，mkdir –p /data/mysql ;chown –R mysqld:mysqld /data/mysql

启动 mysql 即可，service mysqld restart

然后修改 slave Mysql 数据库 my.cnf 配置文件内容如下：

[mysqld]

datadir=/data/mysql

socket=/var/lib/mysql/mysql.sock

user=mysql

Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

server-id = 2

auto_increment_offset=2

auto_increment_increment=2

[mysqld_safe]

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

在 Master 数据库服务器上设置权限，执行如下命令：

grant replication slave on *.* to 'tongbu'@'%' identified by '123456';

在 Master 数据库执行如下命令：

mysql> show master status;

```
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000006 |       98 |              |                  |
+------------------+----------+--------------+------------------+
```

1 row in set (0.00 sec)

然后在 slave 服务器指定 master IP 和同步的 pos 点：

change master to

master_host='192.168.1.103',master_user='tongbu',master_password='123456

',master_log_file='mysql-bin.000006',master_log_pos=98;

在 slave 启动 slave start，并执行 **show slave status\G** 查看 Mysql 主从状态：

```
[root@node2 ~]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.0.95-log Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> slave start;
Query OK, 0 rows affected (0.00 sec)

mysql> show slave status\G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.149.128
                Master_User: tongbu
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000006
        Read_Master_Log_Pos: 98
             Relay_Log_File: mysqld-relay-bin.000004
              Relay_Log_Pos: 235
      Relay_Master_Log_File: mysql-bin.000006
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB:
```

Slave_IO_Running: Yes

Slave_SQL_Running: Yes 两个状态为 YES，代表 slave 已经启动两个线程，一个为 IO 线程，一个为 SQL 线程。

然后在 Master 服务器创建一个数据库和表，命令如下：

192.168.149.128 × 192.168.149.129

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

mysql> create database mysql_ab_test charset=utf8;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| mysql_ab_test      |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql> use mysql_ab_test;
Database changed
mysql>
mysql> create table t0 (id varchar(20),name varchar(30));
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+-------------------------+
| Tables_in_mysql_ab_test |
+-------------------------+
| t0                      |
+-------------------------+
1 row in set (0.00 sec)
```

然后去 slave 服务器查看是否有 mysql_ab_test 数据库和相应 t0 的表，如果存在则代表

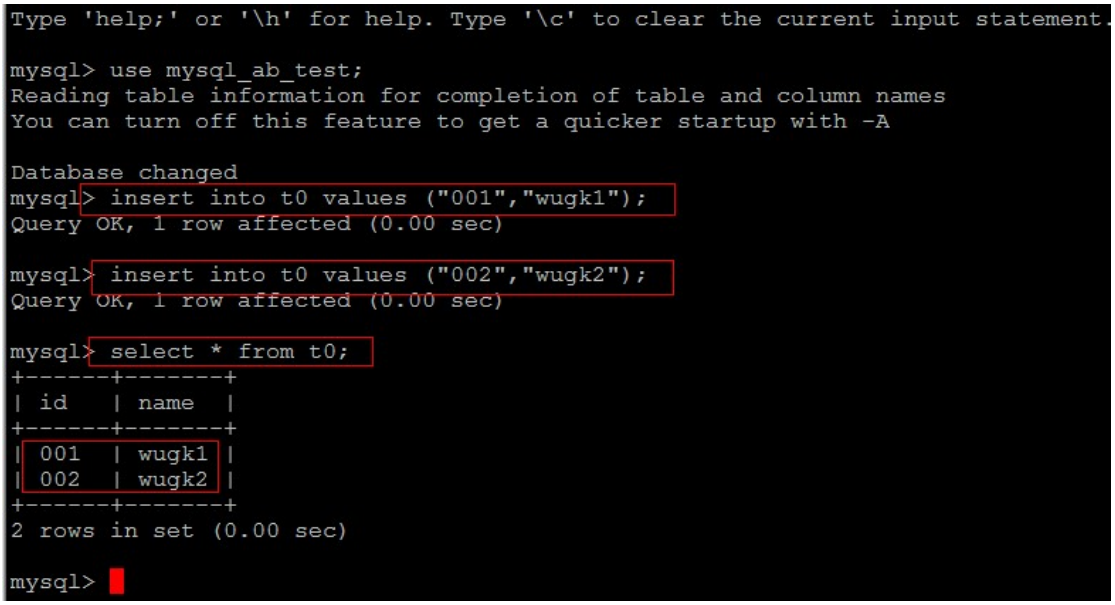
Mysql 主从同步搭建成功：

192.168.149.128 192.168.149.129 ×

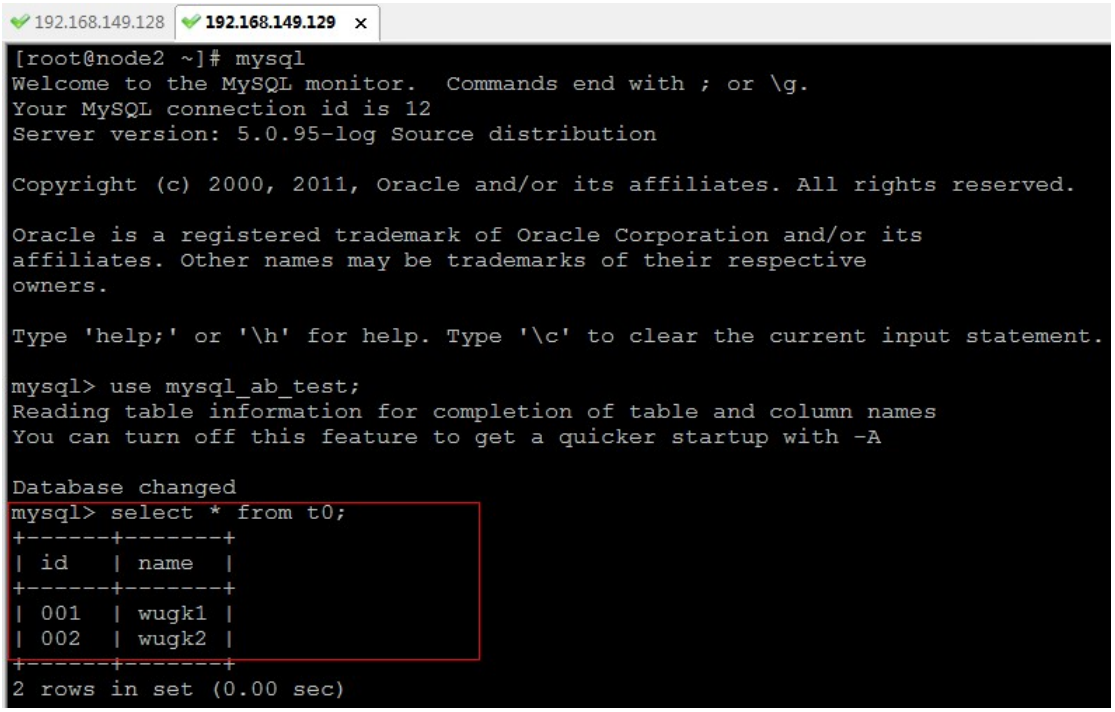
```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| mysql_ab_test      |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql> use mysql_ab_test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-------------------------+
| Tables_in_mysql_ab_test |
+-------------------------+
| t0                      |
+-------------------------+
1 row in set (0.00 sec)
```

同样还可以测试在 master 服务器插入两条数据，在 slave 查看 insert 数据是否已同步：

128 master 上执行如下图：



129 slave 上执行如下图，在 master 插入的数据已经同步到 slave 上：



自此 Mysql 主从搭建完毕，现在有一个问题，如果 master 服务器 down 机了，如何快速恢复服务呢?

可以通过两种方法：

第一种方法，如果程序连接的是 master 的 IP，直接在 slave 服务器上添加 master 的

IP 即可。这个手动去操作，而且需要花费时间比较长，可能还会出现误操作的情况，不推荐。

第二种方法，可以使用 keepalived、heartbeat 作为 HA 检测软件，检查 MySQL 服务是否正常，不正常则自动切换到 slave 上，推荐使用。

12.5 MySQL 主从注意事项

mysql 主从同步的原理：

1、在 master 上开启 bin-log 日志功能，记录更新、插入、删除的语句。

2、必须开启三个线程，主上开启 io 线程，从上开启 io 线程和 sql 线程。

3、从上 io 线程去连接 master，master 通过 io 线程检查有 slave 过来的请求，请求日志、postsion 位置。

4、master 将这些相应的日志返回给 slave，slave 自己去下载到本地的 realy_log 里面，写入一个 master-info

日志记录同步的点。

5、slave 的 sql 线程检查到 realy-log 日志有更新，然后在本地去 exec 执行。

6、主从同步是属于异步方式。

20133

insert into table values ('wgkgood','wgkgood@163.com');

20134

insert into t1 values ('Linux 系统安装方法'，'dsfjsdklsdfjsdlk');

20135

change master to

master_host='192.168.1.12',master_user='tongbu',master_password='123456',mas

ter_log_file='mysql-bin.000001',master_log_pos=272;

二、Mysql 主从同步，master 突然 down 机，如何恢复：

1、slave 数据库必须启动，在 slave 上授权网站 IP 对数据库的访问权限。

2、修改网站服务器 config 目录下:

config_global.php config_ucenter.php 把原先 master ip 改成 slave ip 地址。

3、重启 httpd 服务，切换成功。

作业：

1、在 master 上已经创建好了 discuz 数据库，同时有论坛，新建从，怎么同步数据?

mysqldump >mysql.sql ；导入进去，再同步。

12.6 MySQL 主从同步故障解决方案

方法一：忽略错误后，继续同步

该方法适用于主从库数据相差不大，或者要求数据可以不完全统一的情况，数据要求不严格的情况

解决：

Master 上执行：flush tables with read lock;

Slave 上执行:

stop slave;

#表示跳过一步错误，后面的数字可变

set global sql_slave_skip_counter =1;

start slave;

之后再用 mysql> show slave status\G　查看：

Slave_IO_Running: Yes

Slave_SQL_Running: Yes

ok，现在主从同步状态正常了。。。

方式二：重新做主从，完全同步

该方法适用于主从库数据相差较大，或者要求数据完全统一的情况

解决步骤如下：

1)先进入主库，进行锁表，防止数据写入

使用命令：

mysql> **flush tables with read lock;**

注意：该处是锁定为只读状态，语句不区分大小写

2)进行数据备份

#把数据备份到 mysql.bak.sql 文件

[root@server01 mysql]#**mysqldump -uroot -p -hlocalhost > mysql.sql**

这里注意一点：数据库备份一定要定期进行，可以用 shell 脚本或者 python 脚本，都比较方便，确保数据万无一失

3)查看 master 状态

mysql> show master status;

```
+------------------+----------+--------------+------------------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB             |
+------------------+----------+--------------+------------------------------+
| mysqld-bin.000001 |      272 |              |
```

+------------------+----------+-------------+------------------------------+

1 row in set (0.00 sec)

4)把 mysql 备份文件传到从库机器，进行数据恢复

#使用 scp 命令

[root@server01mysql]# scp mysql.sql root@192.168.1.13:/tmp/

5)停止从库的状态

mysql> stop slave;

6)然后到从库执行 mysql 命令，导入数据备份

mysql> source /tmp/mysql.sql

7)设置从库同步，注意该处的同步点，就是主库 show master status 信息里的| File| Position 两项

change master to master_host = '192.168.1.12', master_user = 'tongbu',master_password='123456', master_log_file = 'mysqld-bin.000001', master_log_pos=272;

8)重新开启从同步

mysql> start slave;

9)查看同步状态

mysql> show slave status\G　查看：

Slave_IO_Running: Yes

Slave_SQL_Running: Yes

10)在 master 上解锁：

unlock tables;

12.7 LAMP 企业架构集群讲解

Linux+Apache+Mysql/MariaDB+Perl/PHP/Python 一组常用来搭建动态网站或者服务器的开源软件，本身都是各自独立的程序，但是因为常被放在一起使用，拥有了越来越高的兼容度，共同组成了一个强大的 Web 应用程序平台。

随着开源潮流的蓬勃发展，开放源代码的 LAMP 已经与 J2EE 和.Net 商业软件形成三足鼎立之势，并且该软件开发的项目在软件方面的投资成本较低，因此受到整个 IT 界的关注。

目前 LAMP 架构是大多数中小企业**最青睐的 PHP 架构选择**，也是众多 Linux SA 喜欢选择的一套架构。那接下来我们就实战来操作一下，如果来搭建这样一套架构，当然可以使用 yum 方法，安装命令很简单，一条命令搞定所有。

yum install httpd httpd-devel mysql mysql-server mysql-devel php php-devel php-mysql -y

这一条命令 LAMP 环境即可安装成功，只需要重启 apache、mysql 服务即可。

如果想要更多功能和自定义模块，需要使用源码包的方式来安装 LAMP 架构。如下我们使用源码包来实现 LAMP 架构安装与配置：

- 源码安装 LAMP 之 Apache

yum install apr-devel apr-util-devel -y；

cd /usr/src ; wget http://mirror.bit.edu.cn/apache/httpd/httpd-2.2.31.tar.gz ;tar xzf httpd-2.2.31.tar.gz ;cd httpd-2.2.31 ;./configure --prefix=/usr/local/apache --enable-so --enable-rewrite &&make &&make install

- 源码安装 LAMP 之 MySQL

cd /usr/src ;

wget http://downloads.mysql.com/archives/mysql-5.1/mysql-5.1.63.tar.gz ;tar xzf mysql-5.1.63.tar.gz ;cd mysql-5.1.63 ;./configure --prefix=/usr/local/mysql --enable-assembler &&make &&make install

cmake

```
make[3]: Nothing to be done for `install-data-am'.
make[3]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
make[2]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
Making install in instance-manager
make[2]: Entering directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[3]: Entering directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
test -z "/usr/local/mysql/libexec" || /bin/mkdir -p "/usr/local/mysql/libexec"
  /bin/sh ../../libtool --preserve-dup-deps   --mode=install /usr/bin/install -c 'mysqlmanager' '/usr/local/m
r'
libtool: install: /usr/bin/install -c mysqlmanager /usr/local/mysql/libexec/mysqlmanager
make[3]: Nothing to be done for `install-data-am'.
make[3]: Leaving directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[2]: Leaving directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[1]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
Making install in win
make[1]: Entering directory `/usr/src/mysql-5.1.63/win'
make[2]: Entering directory `/usr/src/mysql-5.1.63/win'
make[2]: Nothing to be done for `install-exec-am'.
make[2]: Nothing to be done for `install-data-am'.
make[2]: Leaving directory `/usr/src/mysql-5.1.63/win'
make[1]: Leaving directory `/usr/src/mysql-5.1.63/win'
[root@node2 mysql-5.1.63]#
[root@node2 mysql-5.1.63]# ./configure  --prefix=/usr/local/mysql --enable-assembler &&make &&make install
```

配置 Mysql 服务为系统服务：

cp /usr/local/mysql/share/mysql/my-medium.cnf /etc/my.cnf

cp /usr/local/mysql/share/mysql/mysql.server /etc/rc.d/init.d/mysqld

chkconfig --add mysqld

chkconfig --level 35 mysqld on

cd /usr/local/mysql

useradd mysql

chown -R mysql.mysql /usr/local/mysql

/usr/local/mysql/bin/mysql_install_db --user=mysql

chown -R mysql var

/usr/local/mysql/bin/mysqld_safe --user=mysql &

- 源码安装 LAMP 之 PHP

cd /usr/src ;wget http://mirrors.sohu.com/php/php-5.3.28.tar.bz2 ;tar jxf php-5.3.28.tar.bz2 ;cd php-5.3.28 ;

./configure --prefix=/usr/local/php5 --with-config-file-path=/usr/local/php5/etc

--with-apxs2=/usr/local/apache/bin/apxs

--with-mysql=/usr/local/mysql/

```
Installing PHP CLI man page:      /usr/local/php5/man/man1/
Installing build environment:     /usr/local/php5/lib/php/build/
Installing header files:          /usr/local/php5/include/php/
Installing helper programs:       /usr/local/php5/bin/
  program: phpize
  program: php-config
Installing man pages:             /usr/local/php5/man/man1/
  page: phpize.1
  page: php-config.1
Installing PEAR environment:      /usr/local/php5/lib/php/
[PEAR] Archive_Tar    - installed: 1.3.11
[PEAR] Console_Getopt - installed: 1.3.1
warning: pear/PEAR requires package "pear/Structures_Graph" (recommended version 1.0.4)
warning: pear/PEAR requires package "pear/XML_Util" (recommended version 1.2.1)
[PEAR] PEAR           - installed: 1.9.4
Wrote PEAR system config file at: /usr/local/php5/etc/pear.conf
You may want to add: /usr/local/php5/lib/php to your php.ini include_path
[PEAR] Structures_Graph- installed: 1.0.4
[PEAR] XML_Util       - installed: 1.2.1
/usr/src/php-5.3.28/build/shtool install -c ext/phar/phar.phar /usr/local/php5/bin
ln -s -f /usr/local/php5/bin/phar.phar /usr/local/php5/bin/phar
Installing PDO headers:          /usr/local/php5/include/php/ext/pdo/
[root@node2 php-5.3.28]#
```

- 源码安装 Apache+PHP 整合

整合 Apache+php 环境，修改 httpd.conf 配置文件，然后加入如下语句：

LoadModule php5_module modules/libphp5.so (默认已存在)

AddType application/x-httpd-php .php

DirectoryIndex index.php index.html (把 index.php 加入 index.html 之前)

然后在/usr/local/apache/htdocs 目录下创建 index.php 测试页面，执行如下命令：

Vi /usr/local/apache/htdocs/index.php

<?php

phpinfo();

?>

重新启动 apache 服务，通过 IP 访问界面如下图，即代表 LAMP 环境搭建成功。

System	Linux node2 2.6.18-308.el5 #1 SMP Tue Feb 21 20:06:06 EST 2012 x86_64
Build Date	May 26 2014 14:34:36
Configure Command	'./configure' '--prefix=/usr/local/php5' '--with-config-file-path=/usr/local/php/etc' '--with-apxs2=/usr/local/apache/bin/apxs' '--with-mysql=/usr/local/mysql/'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/php/etc
Loaded Configuration File	(none)
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626

- 源码安装 DISCUZ 论坛

下载 discuz 源码包文件，然后解压：

cd /usr/src ;wget

http://download.comsenz.com/DiscuzX/3.1/Discuz_X3.1_SC_UTF8.zip

解压 discuz 程序包：unzip Discuz_X3.1_SC_UTF8.zip -d /usr/local/apache/htdocs/

重命名程序文件：cd /usr/local/apache/htdocs/ ;mv upload/* .

赋予 discuz 目录完全访问权限：cd /usr/local/apache/htdocs/ ;chmod 777 -R data/ uc_server/ config/ uc_client/

然后访问 IP 安装 discuz 论坛，如下图，选择“我同意”

进入如下界面，数据库安装，如果不存在则需要新建数据库并授权。

数据库创建及授权命令如下：

create database discuz charset=utf8;

grant all on discuz.* to root@'localhost' identified by "123456";

```
[root@node2 ~]# /usr/local/mysql/bin/mysql -uroot -p123456
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.1.63-log Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> create database discuz charset=utf8;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on discuz.* to root@'localhost' identified by "123456";
Query OK, 0 rows affected (0.00 sec)

mysql>
```

点击下一步，直至安装完成，进入等待已久的论坛画面：

自此 LAMP 环境整合并搭建成功，通过 IP 直接访问即可。

12.8 LAMP 企业架构拓展实战

当我们在一台服务器搭建完 LAMP 架构后，随着访问量的不断增加，一台服务器压力逐渐增加,那如何来拆分 LAMP 架构呢,怎么把 apache 和 mysql 分开放在不同的机器呢，用 binlog 怎么恢复 mysql 数据，mysql 慢查询又是什么呢?

1、LAMP 多主机部署

Apache+php 在一台机器，mysql 在另外一台机器。

1) Yum 方式：在 apache 网站端，只需要执行：

yum install httpd httpd-devel mysql php-devel php 即可。

2) 源码方式：Apache 默认编译，php 编译的时候添加如下参数：

./configure --prefix=/usr/local/php5

--with-mysql=mysqlnd --with-mysqli=mysqlnd --with-pdo-mysql=mysqlnd

--with-apxs2=/usr/local/apache2/bin/apxs ;make -j8;make -j8 install

(Mysql Native Driver 简称 mysqlnd)，在企业中，一般都是使用 hosts 方式，通过域名去访问 mysql 程序，例如在 apache 服务器端 hosts 配置如下记录：

Vi /etc/hosts

192.168.1.12 mysql.jfedu.net

然后安装的时候，数据库地址改成 mysql.jfedu.net，或者修改程序配置文件数据库的地址为 mysq.jfedu.net。

2、PHP 空白

short_open_tag = On

除此之外，还有可能是 php 版本兼容性导致的问题哦。

3、mysql-binlog 数据恢复：

查看 binlog 日志的命令：mysqlbinlog mysql-bin.000001 |more

备份：cp mysql-bin.000001 /data/back/20150411/

恢复 1 ：**mysqlbinlog --start-position=215 --stop-position=336 mysql-bin.000001** |mysql -uroot -p

336--776 插入了 4 条记录。

恢复 2：mysqlbinlog --start-position=336 --stop-position=776 mysql-bin.000001

|mysql -uroot -p

mysqldump -uroot -p123456 jfedu > /data/backup/jfedu.sql

4、Mysql 慢查询

查看慢查询日志是否开启：

show variables like "%slow%";

查询一条语句花费的时间超过规定的时间，然后将该条语句记录到慢查询日志里面。

select * from t1 limit 1；超过 10s

如何去优化，这条语句的功能是获取用户名，优化后的语句：

select name from t1 limit 1;

如何去开启慢查询功能:

在 mysql 里面执行：set global slow_query_log=on;

在/etc/my.cnf 配置如下：

log-slow-queries = /var/run/mysqld/slow.log

long_query_time = 5

log-queries-not-using-indexes

然后重启 mysql 服务即可。

12.9 LAMP+Redis 企业实战

Redis 是一个开源的使用 ANSI C 语言编写、支持网络、可基于内存亦可持久化的日志型、Key-Value 数据库，并提供多种语言的 API。

Redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和 hash

（哈希类型）。

Redis 是一种高级 key-value 数据库。它跟 memcached 类似，不过数据可以持久化，而且支持的数据类型很丰富。有字符串，链表，集 合和有序集合。支持在服务器端计算集合的并，交和补集(difference)等，还支持多种排序功能。所以 Redis 也可以被看成是一个数据结构服务 器。

Redis 的所有数据都是保存在内存中，然后不定期的通过异步方式保存到磁盘上(这称为“半持久化模式”)；也可以把每一次数据变化都写入到一个 usr/localend only file(aof)里面(这称为“全持久化模式”)。

```
wget  http://download.redis.io/releases/redis-2.8.13.tar.gz

tar zxf redis-2.8.13.tar.gz

cd redis-2.8.13

make PREFIX=/usr/local/redis install

cp  redis.conf  /usr/local/redis

/root/.bash_profile 添加以下 1 行内容:

export PATH=/usr/local/redis/bin:$PATH

启动 Redis:

/usr/local/redis/bin/redis-server /usr/local/redis/redis.conf

/usr/local/redis/bin/redis-server /usr/local/redis/redis.msg.conf

停止 Redis:

/usr/local/redis/bin/redis-cli -p 6379 shutdown

/usr/local/redis/bin/redis-cli -p 6380 shutdown
```

12.10 PHP 添加 Redis 扩展

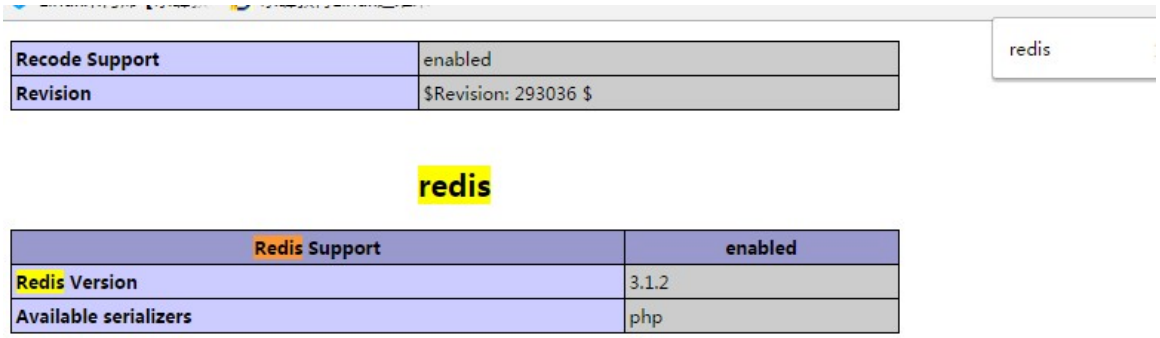
要确保 PHP 代码能够连接 Redis,需要添加 Redis 扩展程序,安装方法如下:

```
wget https://github.com/phpredis/phpredis/archive/3.1.2.tar.gz

tar xzf 3.1.2.tar.gz

cd phpredis-3.1.2/

./configure --with-php-config=/usr/bin/php-config --enable-redis

make

make install
```

修改 php.ini 配置文件，加载 redis.so 模块：

```
extension_dir = "/usr/lib64/php/modules/"

extension=redis.so
```

重启 httpd 服务器，测试 phpinfo 文件，检查 redis 模块：



Recode Support	enabled
Revision	$Revision: 293036 $

redis

Redis Support	enabled
Redis Version	3.1.2
Available serializers	php

修改 discuz 网站全局配置文件，修改 redis server IP，如图所示：

```
// ------------------------  CONFIG MEMORY  ------------------------ /
$_config['memory']['prefix'] = 'IOkLan_';
$_config['memory']['redis']['server'] = '192.168.111.128';
$_config['memory']['redis']['port'] = 6379;
$_config['memory']['redis']['pconnect'] = 1;
$_config['memory']['redis']['timeout'] = '0';
$_config['memory']['redis']['requirepass'] = '';
$_config['memory']['redis']['serializer'] = 1;
$_config['memory']['memcache']['server'] = '';
$_config['memory']['memcache']['port'] = 11211;
$_config['memory']['memcache']['pconnect'] = 1;
$_config['memory']['memcache']['timeout'] = 1;
$_config['memory']['apc'] = 1;
$_config['memory']['xcache'] = 1;
$_config['memory']['eaccelerator'] = 1;
$_config['memory']['wincache'] = 1;
```

访问论坛网站，同时登陆服务器，进入 redis，redis-cli，如图：

```
[root@www-jfedu-net ~]# redis-cli
127.0.0.1:6379>
127.0.0.1:6379> KEYS *
 1) "IOkLan_forum_index_page_1"
 2) "IOkLan_style_default"
 3) "IOkLan_magic"
 4) "IOkLan_usergroups"
 5) "IOkLan_creditrule"
 6) "IOkLan_common_member_field_home_2"
 7) "IOkLan_usergroup_7"
 8) "IOkLan_userstats"
 9) "IOkLan_cronnextrun"
10) "IOkLan_plugin"
11) "IOkLan_common_member_count_1"
12) "IOkLan_onlinelist"
13) "IOkLan_adminmenu"
14) "IOkLan_onlinerecord"
15) "IOkLan_common_member_field_home_1"
```

查看到 redis 数据库中存在 IOKLan 开头的 key 值，则证明 LAMP+Redis 整合成功！

12.11 Redis 配置文件详解

#daemonize no 默认情况下， redis 不是在后台运行的，如果需要在后台运行，把该项的值更改为 yes daemonize yes # 当 redis 在后台运行的时候， Redis 默认会把 pid 文件放在 /var/run/redis.pid ，你可以配置到其他地址。 # 当运行多个 redis 服务时，需要指定不同的 pid 文件和端口

```
pidfile /var/run/redis_6379.pid

#  指定 redis 运行的端口，默认是 6379

port 6379

#  在高并发的环境中，为避免慢客户端的连接问题，需要设置一个高速后台日志

tcp-backlog 511

#  指定 redis 只接收来自于该 IP 地址的请求，如果不进行设置，那么将处理所有请求

# bind 192.168.1.100 10.0.0.1

# bind 127.0.0.1

#  设置客户端连接时的超时时间，单位为秒。当客户端在这段时间内没有发出任何指令，

那么关闭该连接

# 0 是关闭此设置

timeout 0

# TCP keepalive

#  在 Linux 上，指定值（秒）用于发送 ACKs 的时间。注意关闭连接需要双倍的时间。

默认为 0 。

tcp-keepalive 0

#  指定日志记录级别，生产环境推荐 notice

# Redis 总共支持四个级别： debug 、 verbose 、 notice 、 warning ，默认为

verbose

# debug       记录很多信息，用于开发和测试

# varbose     有用的信息，不像 debug 会记录那么多

# notice      普通的 verbose ，常用于生产环境
```

```
# warning    只有非常重要或者严重的信息会记录到日志

loglevel notice

#   配置 log 文件地址

#   默认值为 stdout ，标准输出，若后台模式会输出到 /dev/null 。

logfile /var/log/redis/redis.log

#   可用数据库数

#   默认值为 16 ，默认数据库为 0 ，数据库范围在 0- （ database-1 ）之间

databases 16

################################## 快 照

###################################

#   保存数据到磁盘，格式如下：

#    save <seconds> <changes>

#     指出在多长时间内，有多少次更新操作，就将数据同步到数据文件 rdb 。

#     相当于条件触发抓取快照，这个可以多个条件配合

#     比如默认配置文件中的设置，就设置了三个条件

#    save 900 1   900 秒内至少有 1 个 key 被改变

#    save 300 10   300 秒内至少有 300 个 key 被改变

#    save 60 10000   60 秒内至少有 10000 个 key 被改变

# save 900 1

# save 300 10

# save 60 10000

#   后台存储错误停止写。
```

```
stop-writes-on-bgsave-error yes

#  存储至本地数据库时（持久化到 rdb 文件）是否压缩数据，默认为 yes

rdbcompression yes

# RDB 文件的是否直接偶像 chcksum

rdbchecksum yes

#  本地持久化数据库文件名，默认值为 dump.rdb

dbfilename dump.rdb

#  工作目录

#  数据库镜像备份的文件放置的路径。

#  这里的路径跟文件名要分开配置是因为 redis 在进行备份时，先会将当前数据库的状态写入到一个临时文件中，等备份完成，

#  再把该该临时文件替换为上面所指定的文件，而这里的临时文件和上面所配置的备份文件都会放在这个指定的路径当中。

# AOF 文件也会存放在这个目录下面

#  注意这里必须制定一个目录而不是文件

dir /var/lib/redis/

################################## 复 制 ##################################

#  主从复制 . 设置该数据库为其他数据库的从数据库 .

#  设置当本机为 slav 服务时，设置 master 服务的 IP 地址及端口，在 Redis 启动时，它会自动从 master 进行数据同步

# slaveof <masterip> <masterport>
```

```
#  当 master 服务设置了密码保护时（ 用 requirepass 制定的密码 ）
# slave 服务连接 master 的密码
# masterauth <master-password>
#  当从库同主机失去连接或者复制正在进行，从机库有两种运行方式：
# 1)  如果 slave-serve-stale-data 设置为 yes( 默认设置 )，从库会继续响应客户端的请求
# 2)  如果 slave-serve-stale-data 是指为 no ，出去 INFO 和 SLAVOF 命令之外的任何请求都会返回一个
#       错误 "SYNC with master in progress"
slave-serve-stale-data yes
#  配置 slave 实例是否接受写。写 slave 对存储短暂数据（在同 master 数据同步后可以很容易地被删除）是有用的，但未配置的情况下，客户端写可能会发送问题。
#  从 Redis2.6 后，默认 slave 为 read-only
slaveread-only yes
#  从库会按照一个时间间隔向主库发送 PINGs. 可以通过 repl-ping-slave-period 设置这个时间间隔，默认是 10 秒
# repl-ping-slave-period 10
# repl-timeout   设置主库批量数据传输时间或者 ping 回复时间间隔，默认值是 60 秒
#  一定要确保 repl-timeout 大于 repl-ping-slave-period
# repl-timeout 60
#  在 slave socket 的 SYNC 后禁用 TCP_NODELAY
#  如果选择 “ yes ” ,Redis 将使用一个较小的数字 TCP 数据包和更少的带宽将数据
```

发送到 slave ， 但是这可能导致数据发送到 slave 端会有延迟，如果是 Linux kernel 的默认配置，会达到 40 毫秒 .

如果选择 "no" ，则发送数据到 slave 端的延迟会降低，但将使用更多的带宽用于复制 .

repl-disable-tcp-nodelay no

设置复制的后台日志大小。

复制的后台日志越大， slave 断开连接及后来可能执行部分复制花的时间就越长。

后台日志在至少有一个 slave 连接时，仅仅分配一次。

repl-backlog-size 1mb

在 master 不再连接 slave 后，后台日志将被释放。下面的配置定义从最后一个 slave 断开连接后需要释放的时间（秒）。

0 意味着从不释放后台日志

repl-backlog-ttl 3600

如果 master 不能再正常工作，那么会在多个 slave 中，选择优先值最小的一个 slave 提升为 master ，优先值为 0 表示不能提升为 master 。

slave-priority 100

如果少于 N 个 slave 连接，且延迟时间 <=M 秒，则 master 可配置停止接受写操作。

例如需要至少 3 个 slave 连接，且延迟 <=10 秒的配置：

min-slaves-to-write 3

min-slaves-max-lag 10

设置 0 为禁用

```
#    默认 min-slaves-to-write 为 0 （禁用）， min-slaves-max-lag 为 10

################################## 安 全

##################################

#   设置客户端连接后进行任何其他指定前需要使用的密码。

#   警告：因为 redis 速度相当快，所以在一台比较好的服务器下，一个外部的用户可以在一秒钟进行 150K 次的密码尝试，这意味着你需要指定非常非常强大的密码来防止暴力破解

# requirepass foobared

#   命令重命名 .

#   在一个共享环境下可以重命名相对危险的命令。比如把 CONFIG 重名为一个不容易猜测的字符。

#   举例 ：

# rename-command CONFIG b840fc02d524045429941cc15f59e41cb7be6c52

#   如果想删除一个命令，直接把它重命名为一个空字符 "" 即可，如下：

# rename-command CONFIG ""

################################### 约 束

###################################

#设置同一时间最大客户端连接数，默认无限制，

#Redis 可以同时打开的客户端连接数为 Redis 进程可以打开的最大文件描述符数，

#如果设置  maxclients 0 ，表示不作限制。

#当客户端连接数到达限制时， Redis 会关闭新的连接并向客户端返回 max number of clients reached 错误信息
```

```
# maxclients 10000

#  指定 Redis 最大内存限制， Redis 在启动时会把数据加载到内存中，达到最大内存后， Redis 会按照清除策略尝试清除已到期的 Key

#  如果 Redis 依照策略清除后无法提供足够空间，或者策略设置为 " noeviction" ，则使用更多空间的命令将会报错，例如 SET, LPUSH 等。但仍然可以进行读取操作

#  注意： Redis 新的 vm 机制，会把 Key 存放内存， Value 会存放在 swap 区

#  该选项对 LRU 策略很有用。

# maxmemory 的设置比较适合于把 redis 当作于类似 memcached 的缓存来使用，而不适合当做一个真实的 DB 。

#  当把 Redis 当做一个真实的数据库使用的时候，内存使用将是一个很大的开销

# maxmemory <bytes>

#  当内存达到最大值的时候 Redis 会选择删除哪些数据？有五种方式可供选择

# volatile-lru ->  利用 LRU 算法移除设置过过期时间的 key (LRU: 最近使用 Least RecentlyUsed )

# allkeys-lru ->  利用 LRU 算法移除任何 key

# volatile-random ->  移除设置过过期时间的随机 key

# allkeys->random -> remove a randomkey, any key

# volatile-ttl ->  移除即将过期的 key(minor TTL)

# noeviction ->  不移除任何可以，只是返回一个写错误

#  注意：对于上面的策略，如果没有合适的 key 可以移除，当写的时候 Redis 会返回一个错误

#  默认是 :  volatile-lru
```

```
# maxmemory-policy volatile-lru

# LRU 和 minimal TTL 算法都不是精准的算法，但是相对精确的算法（为了节省内存），随意你可以选择样本大小进行检测。

# Redis 默认的灰选择 3 个样本进行检测，你可以通过 maxmemory-samples 进行设置

# maxmemory-samples 3

##############################

AOF################################

# 默认情况下， redis 会在后台异步的把数据库镜像备份到磁盘，但是该备份是非常耗时的，而且备份也不能很频繁，如果发生诸如拉闸限电、拔插头等状况，那么将造成比较大范围的数据丢失。

# 所以 redis 提供了另外一种更加高效的数据库备份及灾难恢复方式。

# 开启 append only 模式之后， redis 会把所接收到的每一次写操作请求都追加到 appendonly.aof 文件中，当 redis 重新启动时，会从该文件恢复出之前的状态。

# 但是这样会造成 appendonly.aof 文件过大，所以 redis 还支持了 BGREWRITEAOF 指令，对 appendonly.aof 进行重新整理。

# 你可以同时开启 asynchronous dumps 和 AOF

appendonly no

# AOF 文件名称 （默认："appendonly.aof"）

# appendfilename appendonly.aof

# Redis 支持三种同步 AOF 文件的策略：

# no: 不进行同步，系统去操作 . Faster.
```

```
# always: always 表示每次有写操作都进行同步 . Slow, Safest.
# everysec:  表示对写操作进行累积，每秒同步一次 . Compromise.
#  默认是 "everysec" ，按照速度和安全折中这是最好的。
#  如果想让 Redis 能更高效的运行，你也可以设置为 "no" ，让操作系统决定什么时候去执行
#  或者相反想让数据更安全你也可以设置为 "always"
#  如果不确定就用  "everysec".
# appendfsync always
appendfsync everysec
# appendfsync no
# AOF 策略设置为 always 或者 everysec 时，后台处理进程（ 后台保存或者 AOF 日志重写 ）会执行大量的 I/O 操作
#  在某些 Linux 配置中会阻止过长的 fsync() 请求。注意现在没有任何修复，即使 fsync 在另外一个线程进行处理
#  为了减缓这个问题，可以设置下面这个参数 no-appendfsync-on-rewrite
no-appendfsync-on-rewrite no
# AOF  自动重写
#  当 AOF 文件增长到一定大小的时候 Redis 能够调用  BGREWRITEAOF  对日志文件进行重写
#  它是这样工作的：Redis 会记住上次进行些日志后文件的大小（ 如果从开机以来还没进行过重写，那日子大小在开机的时候确定 ）
#  基础大小会同现在的大小进行比较。如果现在的大小比基础大小大制定的百分比，重
```

写功能将启动

同时需要指定一个最小大小用于 AOF 重写，这个用于阻止即使文件很小但是增长幅度很大也去重写 AOF 文件的情况

设置 percentage 为 0 就关闭这个特性

auto-aof-rewrite-percentage 100

auto-aof-rewrite-min-size 64mb

################################## LUASCRIPTING

###############################

一个 Lua 脚本最长的执行时间为 5000 毫秒（ 5 秒），如果为 0 或负数表示无限执行时间。

lua-time-limit 5000

################################LOW

LOG################################

Redis Slow Log 记录超过特定执行时间的命令。执行时间不包括 I/O 计算比如连接客户端，返回结果等，只是命令执行时间

可以通过两个参数设置 slow log ：一个是告诉 Redis 执行超过多少时间被记录的参数 slowlog-log-slower-than(微妙)，

另一个是 slow log 的长度。当一个新命令被记录的时候最早的命令将被从队列中移除

下面的时间以微妙为单位，因此 1000000 代表一秒。

注意指定一个负数将关闭慢日志，而设置为 0 将强制每个命令都会记录

slowlog-log-slower-than 10000

```
#   对日志长度没有限制，只是要注意它会消耗内存

#   可以通过   SLOWLOG RESET  回收被慢日志消耗的内存

#   推荐使用默认值  128  ，当慢日志超过  128  时，最先进入队列的记录会被踢出

slowlog-max-len 128

################################          事      件      通      知

##############################

#   当事件发生时，  Redis  可以通知  Pub/Sub  客户端。

#   可以在下表中选择  Redis  要通知的事件类型。事件类型由单个字符来标识：

# K        Keyspace  事件，以  _keyspace@<db>_  的前缀方式发布

# E        Keyevent  事件，以  _keysevent@<db>_  的前缀方式发布

# g        通用事件（不指定类型），像  DEL, EXPIRE, RENAME,  ...

# $        String  命令

# s        Set  命令

# h        Hash  命令

# z        有序集合命令

# x        过期事件（每次  key  过期时生成）

# e        清除事件（当  key  在内存被清除时生成）

# A        g$lshzxe  的别称，因此 ” AKE”  意味着所有的事件

# notify-keyspace-events  带一个由  0  到多个字符组成的字符串参数。空字符串意思是

通知被禁用。

#   例子：启用  list  和通用事件：

# notify-keyspace-events Elg
```

```
#  默认所用的通知被禁用，因为用户通常不需要改特性，并且该特性会有性能损耗。
#  注意如果你不指定至少 K 或 E 之一，不会发送任何事件。
notify-keyspace-events  ""
################################  高  级  配  置  ################################
#  当 hash 中包含超过指定元素个数并且最大的元素没有超过临界时，
# hash 将以一种特殊的编码方式（大大减少内存使用）来存储，这里可以设置这两个临界值
# Redis Hash 对应 Value 内部实际就是一个 HashMap ，实际这里会有 2 种不同实现，
#  这个 Hash 的成员比较少时 Redis 为了节省内存会采用类似一维数组的方式来紧凑存储，而不会采用真正的 HashMap 结构，对应的 valueredisObject 的 encoding 为 zipmap,
#  当成员数量增大时会自动转成真正的 HashMap, 此时 encoding 为 ht 。
hash-max-zipmap-entries 512
hash-max-zipmap-value 64
#  和 Hash 一样，多个小的 list 以特定的方式编码来节省空间。
# list 数据类型节点值大小小于多少字节会采用紧凑存储格式。
list-max-ziplist-entries 512
list-max-ziplist-value 64
# set 数据类型内部数据如果全部是数值型，且包含多少节点以下会采用紧凑格式存储。
set-max-intset-entries 512
```

```
#  和 hashe 和 list 一样，排序的 set 在指定的长度内以指定编码方式存储以节省空间
# zsort 数据类型节点值大小小于多少字节会采用紧凑存储格式。
zset-max-ziplist-entries 128
zset-max-ziplist-value 64
# Redis 将在每 100 毫秒时使用 1 毫秒的 CPU 时间来对 redis 的 hash 表进行重新 hash ，可以降低内存的使用
#  当你的使用场景中，有非常严格的实时性需要，不能够接受 Redis 时不时的对请求有 2 毫秒的延迟的话，把这项配置为 no 。
#  如果没有这么严格的实时性要求，可以设置为 yes ，以便能够尽可能快的释放内存
activerehashing yes
# 客户端的输出缓冲区的限制，因为某种原因客户端从服务器读取数据的速度不够快，
# 可用于强制断开连接（一个常见的原因是一个发布 / 订阅客户端消费消息的速度无法赶上生产它们的速度）。
#  可以三种不同客户端的方式进行设置：
# normal ->  正常客户端
# slave  -> slave 和 MONITOR 客户端
# pubsub ->  至少订阅了一个 pubsub channel 或 pattern 的客户端
#  每个 client-output-buffer-limit 语法 ：
# client-output-buffer-limit <class><hard limit> <soft limit> <soft seconds>
#  一旦达到硬限制客户端会立即断开，或者达到软限制并保持达成的指定秒数（连续）。
#  例如，如果硬限制为 32 兆字节和软限制为 16 兆字节 /10 秒，客户端将会立即断
```

开

如果输出缓冲区的大小达到 32 兆字节，客户端达到 16 兆字节和连续超过了限制 10 秒，也将断开连接。

默认 normal 客户端不做限制，因为他们在一个请求后未要求时（以推的方式）不接收数据，

只有异步客户端可能会出现请求数据的速度比它可以读取的速度快的场景。

把硬限制和软限制都设置为 0 来禁用该特性

client-output-buffer-limit normal 0 0 0

client-output-buffer-limit slave 256mb 64mb60

client-output-buffer-limit pubsub 32mb 8mb60

Redis 调用内部函数来执行许多后台任务，如关闭客户端超时的连接，清除过期的 Key ，等等。

不是所有的任务都以相同的频率执行，但 Redis 依照指定的“ Hz ”值来执行检查任务。

默认情况下，“ Hz ”的被设定为 10 。

提高该值将在 Redis 空闲时使用更多的 CPU 时，但同时当有多个 key 同时到期会使 Redis 的反应更灵敏，以及超时可以更精确地处理。

范围是 1 到 500 之间，但是值超过 100 通常不是一个好主意。

大多数用户应该使用 10 这个预设值，只有在非常低的延迟的情况下有必要提高最大到 100 。

hz 10

当一个子节点重写 AOF 文件时，如果启用下面的选项，则文件每生成 32M 数据进

行同步。 aof-rewrite-incremental-fsync yes

12.12 LAMP 企业架构优化实战

当我们把 LAMP 架构配置好后，如何让服务器承担更大的访问量呢？今天我们将一起来学习 LAMP 如何来优化。

1) 优化 PHP 服务器

eAccelerator、APC、Xcache、ZEND 属于一个免费的开源 php 加速、优化、编译和动态缓存项目，原理和 apc 类似，都是通过缓存 php 编译后的 opcode 代码来提高 php 脚本的执行性能。

目前最新的版本下载地址:

https://codeload.github.com/eaccelerator/eaccelerator/legacy.tar.gz/master

tar -xzf master ;cd

eaccelerator-eaccelerator-42067ac/;phpize ;./configure --enable-eaccelerator=shared --with-php-config=/usr/bin/php-config

make && make install

然后在/etc/php.ini 末尾加入如下代码:

extension="/usr/lib64/php/modules/eaccelerator.so"

eaccelerator.shm_size="64"

eaccelerator.cache_dir="/data/eaccelerator"

eaccelerator.enable="1"

eaccelerator.optimizer="1"

eaccelerator.check_mtime="1"

eaccelerator.debug="0"

eaccelerator.filter=""

eaccelerator.shm_max="0"

eaccelerator.shm_ttl="0"

eaccelerator.shm_only="0"

eaccelerator.compress="1"

eaccelerator.compress_level="9"

如上参数详解：

eaccelerator.shm_size

指定 eAccelerator 能够使用的共享内存数量，单位：MB。

"0" 代表操作系统默认，默认值为 "0"，一般设置为 64 或 128。

eaccelerator.cache_dir

用户磁盘缓存的目录。eAccelerator 在该目录中存储预编译代码、session 数据、内容等。

相同的数据也可以存储于共享内存中（以获得更快的存取速度）。默认值为

"/tmp/eaccelerator"。

eaccelerator.enable

开启或关闭 eAccelerator。"1" 为开启，"0" 为关闭。默认值为 "1"。

eaccelerator.optimizer

开启或关闭内部优化器，可以提升代码执行速度。"1" 为开启，"0" 为关闭。默认值为 "1"。

eaccelerator.debug

开启或关闭调试日志记录。"1" 为开启，"0" 为关闭。默认值为 "0"。

eaccelerator.check_mtime

开启或关闭 PHP 文件改动检查。"1" 为开启，"0" 为关闭。如果您想要在修改后重新编译 PHP 程序则需要设置为 "1"。默认值为 "1"。

eaccelerator.filter

判断哪些 PHP 文件必须缓存。您可以指定缓存和不缓存的文件类型（如 "*.php *.phtml" 等），如果参数以 "!" 开头，则匹配这些参数的文件被忽略缓存。默认值为 ""，即，所有 PHP 文件都将被缓存。

eaccelerator.shm_max

当使用 " eaccelerator_put() " 函数时禁止其向共享内存中存储过大的文件。该参数指定允许存储的最大值，单位：字节（10240, 10K, 1M)。"0" 为不限制。默认值为 "0"。

eaccelerator.shm_ttl

当 eAccelerator 获取新脚本的共享内存大小失败时，它将从共享内存中删除所有在最后 "shm_ttl" 秒内无法存取的脚本缓存。默认值为 "0"，即：不从共享内春中删除任何缓存文件。

eaccelerator.shm_prune_period

当 eAccelerator 获取新脚本的共享内存大小失败时，他将试图从共享内存中删除早于 "shm_prune_period" 秒的缓存脚本。默认值为 "0"，即：不从共享内春中删除任何缓存文件。

eaccelerator.shm_only

允许或禁止将已编译脚本缓存在磁盘上。该选项对 session 数据和内容缓存无效。默认值为 "0"，即：使用磁盘和共享内存进行缓存。

eaccelerator.compress

允许或禁止压缩内容缓存。默认值为"1"，即：允许压缩。

eaccelerator.compress_level

指定内容缓存的压缩等级。默认值为"9"，为最高等级。

第13章 Zabbix 企业级监控实战篇

企业服务器对用户提供服务，作为运维工程师最重要的事情就是保证该网站正常稳定的运行，需要实时监控网站、服务器的运行状态，并且有故障及时去处理。

监控网站无需人工时刻去访问 WEB 网站或者登陆服务器去检查，可以借助开源监控软件例如 Zabbix、Cacti、Nagios、Ganglia 等来实现对网站的 7x24 小时的监控，并且做到有故障及时报警通知 SA 解决。

本章向读者介绍企业级分布式监控 Zabbix 入门、Zabbix 监控原理、最新版本 Zabbix 安装实战、Zabbix 批量监控客户端、监控 MYSQL、WEB 关键词及微信报警等。

13.1 Zabbix 监控系统入门简介

Zabbix 是一个基于 WEB 界面的提供分布式系统监控的企业级的开源解决方案，Zabbix 能监视各种网络参数，保证服务器系统的安全稳定的运行，并提供灵活的通知机制以让 SA 快速定位并解决存在的各种问题。Zabbix 分布式监控系统的优点如下：

- 支持自动发现服务器和网络设备；
- 支持底层自动发现；
- 分布式的监控体系和集中式的 WEB 管理；
- 支持主动监控和被动监控模式；

- 服务器端支持多种操作系统：Linux, Solaris, HP-UX, AIX, FreeBSD, OpenBSD, MAC 等；
- Agent 客户端支持多种操作系统：Linux, Solaris, HP-UX, AIX, FreeBSD,Windows 等；
- 基于 SNMP、IPMI 接口方式也可以监控 Agent；
- 安全的用户认证及权限配置；
- 基于 WEB 的管理方法，支持自由的自定义事件和邮件发送；
- 高水平的业务视图监控资源，支持日志审计，资产管理等功能；
- 支持高水平 API 二次开发、脚本监控、自 Key 定义、自动化运维整合调用。

13.2 Zabbix 监控组建及流程

Zabbix 监控组建如图 13-1 所示，主要由三大组件，分别是 Zabbix server 端、Zabbix Proxy、Agent 客户端，其中 Zabbix Server 端包括：WEB GUI、Database、Zabbix_Server。

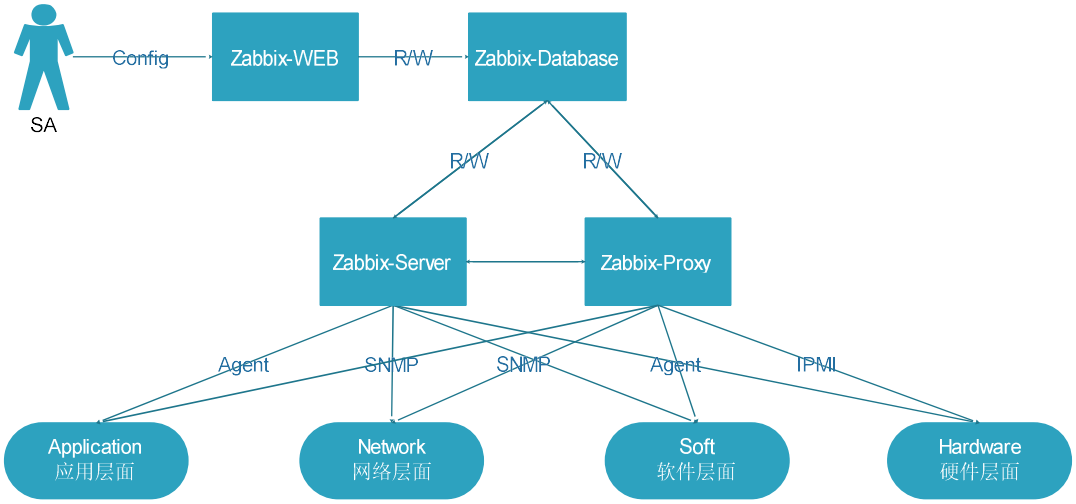


图 13-1 Zabbix 监控组建

Zabbix 监控系统具体监控系统流程如图 13-2 所示：

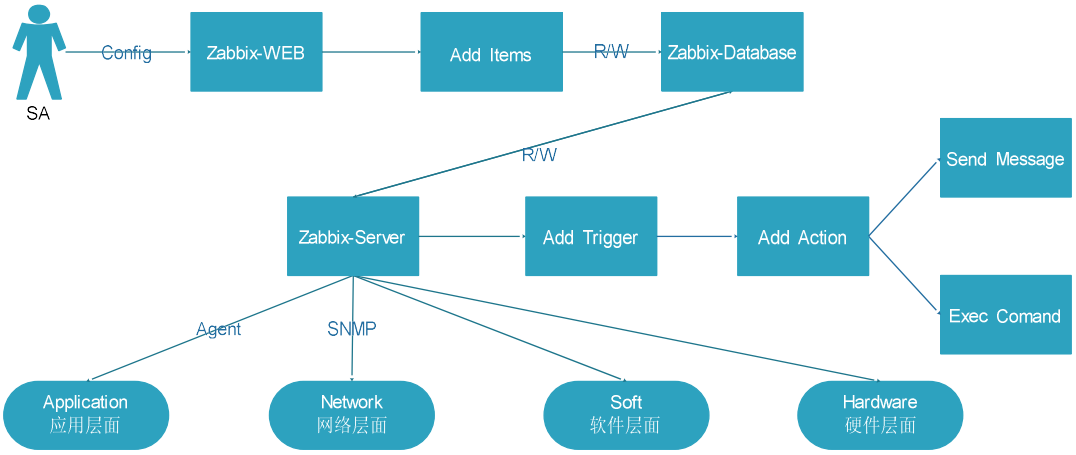


图 13-2 Zabbix 监控流程图

Zabbix 监控完整流程包括：Agentd 安装在被监控的主机上，Agent 负责定期收集客户端本地各项数据，并发送到 Zabbix Server 端，Zabbix Server 收到数据，将数据存储到数据库中，用户基于 Zabbix WEB 可以看到数据在前端展现图像。

当 Zabbix 监控某个具体的项目，该项目会设置一个触发器阀值，当被监控的指标超过该触发器设定的阀值，会进行一些必要的动作，动作包括：邮件、微信报警或者执行命令等操作。如下为 Zabbix 完整监控系统，各个部分负责的工作：

- Zabbix Server：负责接收 agent 发送的报告信息的核心组件，所有配置，统计数据及操作数据均由其组织进行；
- Database Storage：专用于存储所有配置信息，以及存储由 Zabbix 收集到的数据；
- Web interface：Zabbix 的 GUI 接口，通常与 Server 运行在同一台主机上；
- Proxy：常用于分布监控环境中，代理 Server 收集部分被监控端的监控数据并统一发往 Server 端；
- Zabbix Agent：部署在被监控主机上，负责收集本地数据并发往 Server 端或 Proxy 端；

Zabbix 监控部署在系统中，会包含常见的四个程序：zabbix_server 、zabbix_get、zabbix_agentd 、zabbix_proxy、zabbix_sender 等。四个程序启动后分别对应四个进程，如下为每个进程的功能：

- Zabbix_server：Zabbix 服务端守护进程，其中 zabbix_agentd、zabbix_get、zabbix_sender、zabbix_proxy 的数据最终均是提交给 Zabbix_Server；
- Zabbix_Agentd：客户端守护进程，负责收集客户端数据，例如收集 cpu 负载、内存、硬盘使用情况等；
- Zabbix_get：Zabbix 数据获取工具，单独使用的命令，通常在 server 或者 proxy 端执行获取远程客户端信息的命令；
- Zabbix_sender：zabbix 数据发送工具，用于发送数据给 server 或者 proxy，通常用于耗时比较长的检查。很多检查非常耗时间，导致 zabbix 超时。于是我们在脚本执行完毕之后，使用 sender 主动提交数据；
- Zabbix_proxy：zabbix 分布式代理守护进程，分布式监控架构需要部署 Zabbix_Proxy。

13.3 Zabbix 监控方式及数据采集

Zabbix 分布式监控系统监控客户端的方式常见有三种，分别是 Agent 方式、SNMP、IPMI 方式，三种方式特点如下：

- Agent：Zabbix 可以基于自身 zabbix_agent 客户端插件监控 OS 的状态，例如 CPU、内存、硬盘、网卡、文件等。
- SNMP：Zabbix 可以通过简单网络管理协议（Simple Network Management Protocol，SNMP)协议监控网络设备或者 windows 主机等，通过设定 SNMP 的参数将相关监控数据传送至服务器端，交换机、防火墙等网络设备一般都支持 SNMP 协议。
- IPMI：智能平台管理接口(Intelligent Platform Management Interface，IPMI)即主要应用于设备的物理特性，包括：温度、电压、电扇工作状态、电源供应以及机箱入侵等。IPMI 最大的优势在于无论 OS 在开机还是关机的状态下，只要接通电源就可以实现对服务器的监控。

Zabbix 监控客户端分为主动监控与被动监控，主被动模式以客户端为参照，Zabbix 监控客户端默认为被动模式，可以修改为主动模式，只需要在客户端配置文件中添加。可以关闭被动模式的方法：在配置文件中加入 StartAgents=0，即为关闭被动模式。主被动监控模式区别如下：

- Zabbix 主动模式：Agent 主动请求 server 获取主动的监控项列表，并主动将监控项内需要检测的数据提交给 server/proxy，zabbix agent 首先向 ServerActive 配置的 IP 请求获取 active items，获取并提交 active tiems 数据值 server 或者 proxy；
- Zabbix 被动模式：Server 向 agent 请求获取监控项的数据，agent 返回数据，Server 打开一个 TCP 连接，Server 发送请求 agent.ping，Agent 接收到请求并且响应，Server 处理接收到的数据。

13.4 Zabbix 监控组件概念

Zabbix 监控系统包括很多监控概念，掌握 Zabbix 监控概念能对 Zabbix 监控快速的理解，如下为 Zabbix 常用术语及解释。

主机（host）：	被监控的网络设备，可以写 IP 或者 DNS；
主机组（host group）：	主机组用于管理主机，可以批量设置权限；
监控项（item）：	具体监控项，items 值有独立的 keys 进行识别；
触发器（trigger）：	为某个 items 设置触发器，达到触发器会执行 action 动作；
事件（event）：	例如达到某个触发器，称之为一个事件；
动作（action）：	对于特定事件事先定义的处理方法，默认可以发送信息及发送命令；
报警升级（escalation）：	发送警报或执行远程命令的自定义方案，如隔 5 分钟发送

一次警报，共发送 5 次等。

媒介（media）：　　发送通知的方式，可以支持 Mail、SMS、Scripts 等；

通知（notification）：　　通过设置的媒介向用户发送的有关某事件的信息；

远程命令　　达到触发器，可以在被监控端执行命令；

模板（template）：　　可以快速监控被监控端，模块包含：item、trigger、graph、screen、application；

web 场景（web scennario）用于检测 web 站点可用性，监控 HTTP 关键词；

web 前端（frontend）：　Zabbix 的 web 接口；

图形（graph）　　监控图像；

屏幕（screens）　　屏幕显示；

幻灯（slide show）　　幻灯显示。

13.5 Zabbix 监控平台部署

Zabbix监控平台部署，至少需要安装四个组件，分别是Zabbix_Server、Zabbix_Web、Databases、Zabbix_Agent，如下为 Zabbix 监控平台安装配置详细步骤：

（1）　系统环境

- Server 端：192.168.149.128
- Agent 端：192.168.149.129

（2）　下载 zabbix 版本，各个版本之间安装方法相差不大，可以根据实际情况选择安装版本，本文版本为 Zabbix-3.2.6.tar.gz。

```
wget http://sourceforge.net/projects/zabbix/files/ZABBIX%20Latest%20Stable/3.2.6/zabbix-3.2.6.tar.gz/download
```

（3） Zabbix Server 端和 Zabbix Agent 执行如下代码：

```
yum -y install  curl  curl-devel net-snmp net-snmp-devel perl-DBI
groupadd zabbix ;useradd -g zabbix zabbix;usermod -s /sbin/nologin zabbix
```

（4） Zabbix Server 端配置

创建 zabbix 数据库，执行授权命令：

```
create  database  zabbix  charset=utf8;
grant all on zabbix.* to zabbix@localhost identified by '123456';
flush privileges;
```

解压 zabbix 软件包并将 Zabbix 基础 SQL 文件导入数据至 Zabbix 数据库：

```
tar  zxvf  zabbix-3.2.6.tar.gz
cd  zabbix-3.2.6
mysql -uzabbix -p123456 zabbix <database/mysql/schema.sql
mysql -uzabbix -p123456 zabbix <database/mysql/images.sql
mysql -uzabbix -p123456 zabbix < database/mysql/data.sql
```

切换至 Zabbix 解压目录，执行如下代码，安装 Zabbix_server：

```
./configure  --prefix=/usr/local/zabbix/  --enable-server  --enable-agent
--with-mysql --enable-ipv6 --with-net-snmp --with-libcurl
make
make install
ln -s /usr/local/zabbix/sbin/zabbix_*  /usr/local/sbin/
```

Zabbix server 安装完毕，cd /usr/local/zabbix/etc/目录，如图 13-3 所示：

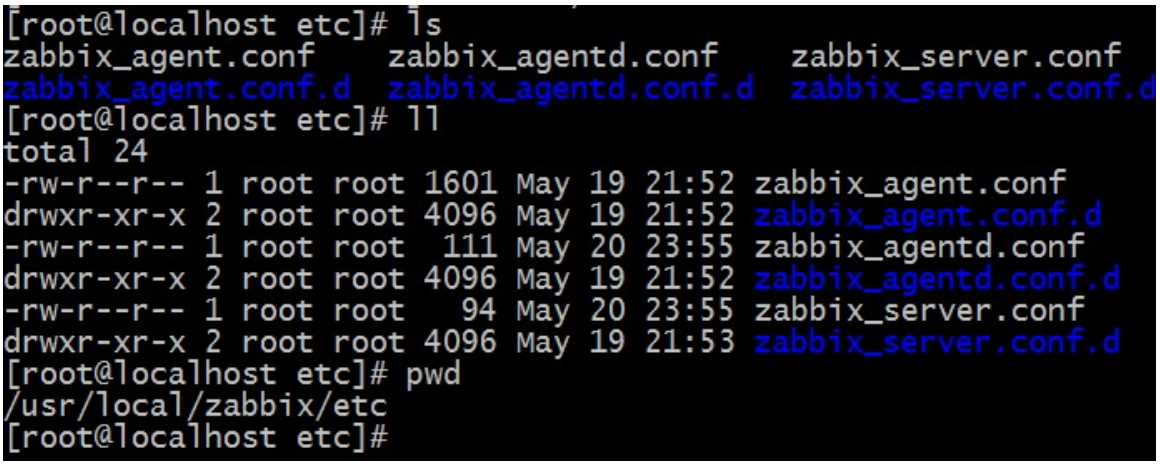


图 13-3 Zabbix 监控流程图

备份 Zabbix server 配置文件，代码如下：

```
cp  zabbix_server.conf  zabbix_server.conf.bak
```

将 zabbix_server.conf 配置文件中代码设置为如下：

```
LogFile=/tmp/zabbix_server.log

DBHost=localhost

DBName=zabbix

DBUser=zabbix

DBPassword=123456
```

同时 cp zabbix_server 启动脚本至/etc/init.d/目录，启动 zabbix_server，Zabbix_server 默认监听端口为 10051。

```
cd  zabbix-3.2.6

cp  misc/init.d/tru64/zabbix_server  /etc/init.d/zabbix_server

chmod  o+x  /etc/init.d/zabbix_server
```

配置 Zabbix interface Web 页面，安装 HTTP WEB 服务器，将 Zabbix WEB 代码发布至 Apache 默认发布目录，由于 Zabbix3.2+ PHP 版本需要使用 PHP5.4.0 版本，请将本机 PHP 版本升级至 5.4.0+，PHP5.3 升级至 PHP5.6，代码如下：

```
rpm -Uvh http://repo.webtatic.com/yum/el6/latest.rpm
yum remove php*
yum install php56w.x86_64 php56w-cli.x86_64 php56w-common.x86_64 php56w-gd.x86_64 php56w-ldap.x86_64 php56w-mbstring.x86_64 php56w-mcrypt.x86_64 php56w-mysql.x86_64 php56w-pdo.x86_64 -y
yum install httpd httpd-devel httpd-tools -y
cp -a /root/zabbix-3.2.6/frontends/php/* /var/www/html/
sed -i '/date.timezone/i date.timezone = PRC' /etc/php.ini
```

重新启动 Zabbix Server、HTTP、MYSQL 服务，代码如下：

```
/etc/init.d/zabbix_server restart
/etc/init.d/httpd restart
/etc/init.d/mysqld restart
```

（5） Zabbix WEB GUI 安装配置

通过浏览器 Zabbix_WEB 验证，通过浏览器访问 http://192.168.149.128/，如图 13-4 所示：

图 13-4 Zabbix WEB 安装界面

单击下一步，出现如图 13-5 所示，如果有错误提示，需要把错误依赖解决完，方可进行下一步操作。

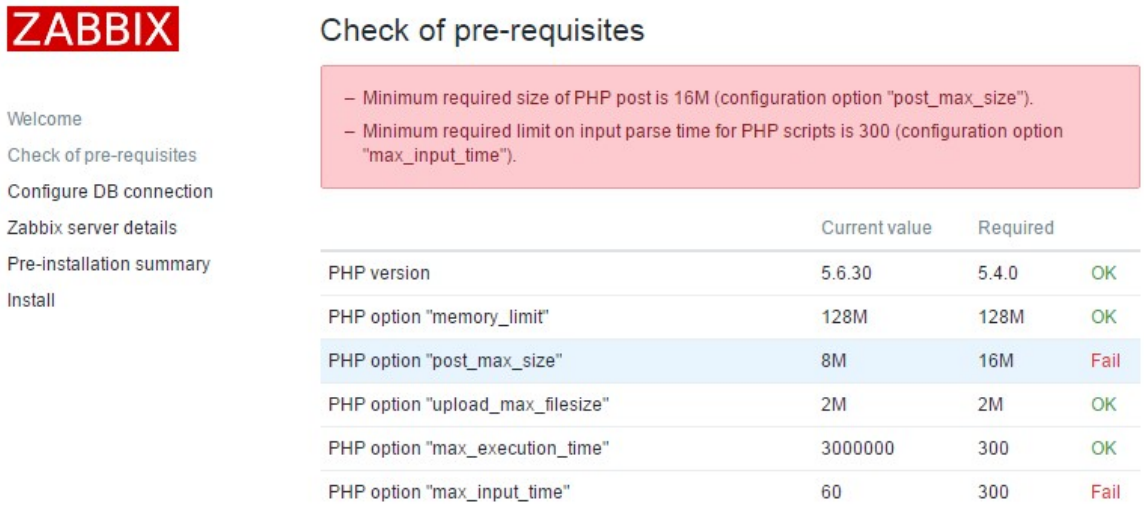


图 13-5 Zabbix WEB 安装错误提示

如上异常错误解决方法代码如下，安装缺失的软包，并修改 php.ini 对应参数的值即可，如图 13-6 所示：

```
    yum install php56w-mbstring php56w-bcmath php56w-gd php56w-xml -y
    yum install gd gd-devel -y
    sed -i '/post_max_size/s/8/16/g;/max_execution_time/s/30/300/g;/max_input_time/s/60/300/g;s/\;date.timezone.*/date.timezone \= PRC/g;s/\;always_populate_raw_post_data/always_populate_raw_post_data/g' /etc/php.ini
    /etc/init.d/httpd restart
```

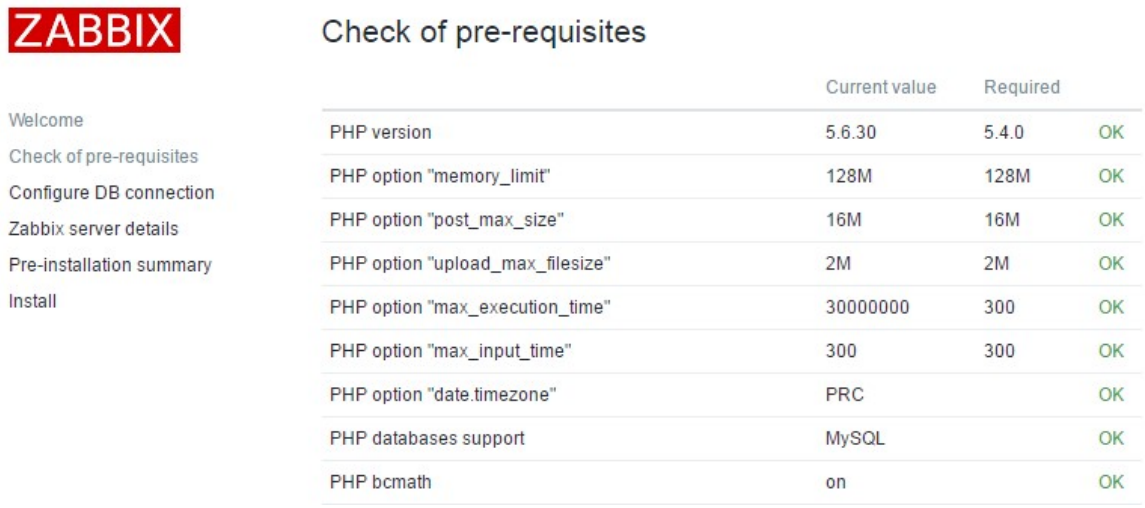


图 13-6 Zabbix WEB 测试安装环境

单击下一步，如图 13-7 所示，配置数据库连接，输入数据库名、用户、密码，单击 Test connection，显示 OK，单击下一步即可。

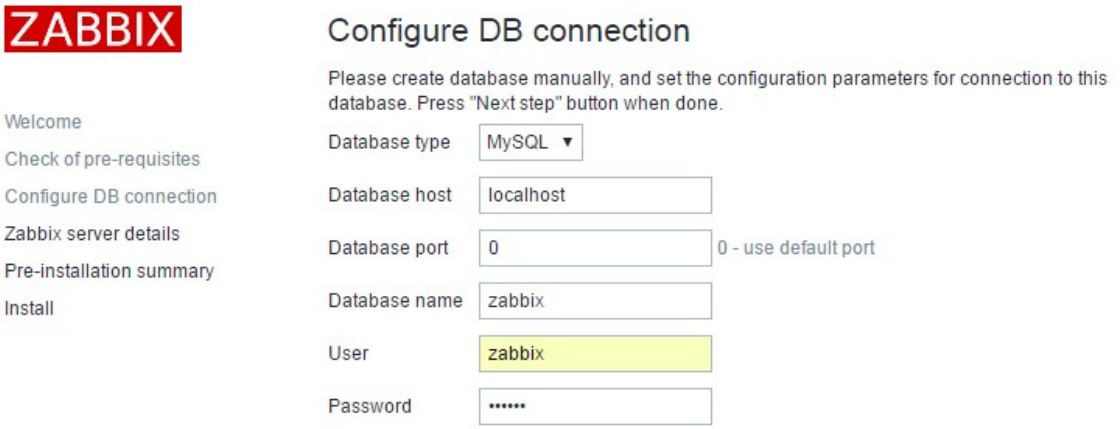


图 13-7 Zabbix WEB 数据库配置

继续单击下一步出现如图 13-8 所示，填写 Zabbix Title 显示，可以为空，可以输入自定义的名称。

图 13-8 Zabbix WEB 详细信息

单击下一步，如图 13-9 所示，需修创建 zabbix.conf.php 文件，执行如下命令，或者单击“Download the configuration file”下载 zabbix.conf.php 文件，并将该文件上传至/var/www/html/conf/，并设置可写权限，刷新 WEB 页面，zabbix.conf.php 内容代码如下，最后单击 Finish 即可：

```
<?php
// Zabbix GUI configuration file.
global $DB;
$DB['TYPE']     = 'MYSQL';
$DB['SERVER']   = 'localhost';
$DB['PORT']     = '0';
$DB['DATABASE'] = 'zabbix';
$DB['USER']     = 'zabbix';
$DB['PASSWORD'] = '123456';
// Schema name. Used for IBM DB2 and PostgreSQL.
$DB['SCHEMA'] = '';
$ZBX_SERVER      = 'localhost';
```

```
$ZBX_SERVER_PORT = '10051';

$ZBX_SERVER_NAME = '京峰教育-分布式监控系统';

$IMAGE_FORMAT_DEFAULT = IMAGE_FORMAT_PNG;
```

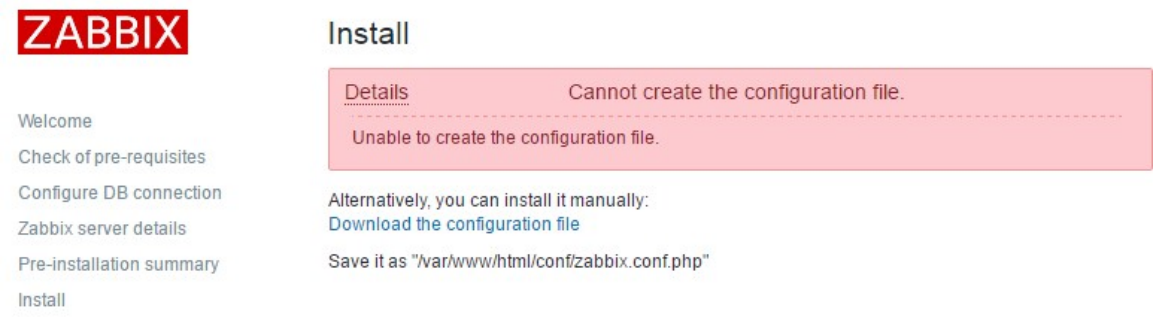


图 13-9 Zabbix WEB 配置文件测试

登录 Zabbix WEB 界面，默认用户名和密码为：admin/zabbix，如图 13-10（a）、13-10（b）所示：

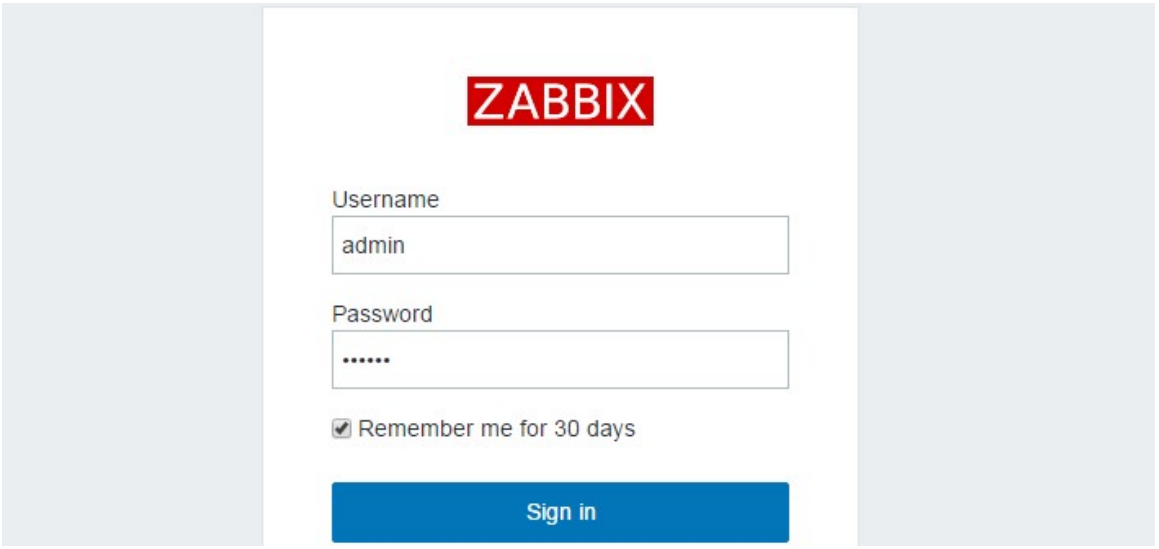


图 13-10（a） Zabbix WEB 登录界面

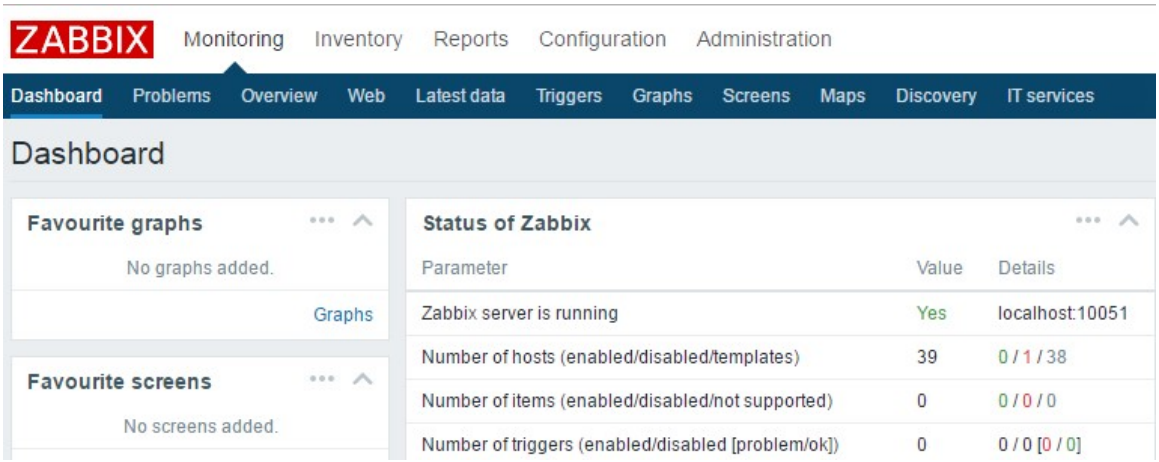


图 13-10（b） Zabbix WEB 后台界面

（6） Agent 客户端安装配置

解压 zabbix-3.2.6.tar.gz 源码文件，切换至解压目录，编译安装 Zabbix，命令如下：

```
./configure  --prefix=/usr/local/zabbix  --enable-agent

make

make install

ln  -s  /usr/local/zabbix/sbin/zabbix_*  /usr/local/sbin/
```

修改 zabbix_agentd.conf 客户端配置文件，执行如下命令，zabbix_agentd.conf 内容，指定 server IP，同时设置本地 Hostname 为本地 IP 地址或者 DNS 名称：

```
LogFile=/tmp/zabbix_agentd.log

Server=192.168.149.128

ServerActive=192.168.149.128

Hostname = 192.168.149.129
```

同时 cp zabbix_agentd 启动脚本至/etc/init.d/目录，启动 zabbix_agentd 服务即可，Zabbix_agentd 默认监听端口为 10050。

```
cd zabbix-3.2.6

cp misc/init.d/tru64/zabbix_agentd /etc/init.d/zabbix_agentd
```

```
chmod o+x /etc/init.d/zabbix_agentd

/etc/init.d/zabbix_agentd    start
```

（7）　Zabbix 监控客户端

Zabbix 服务端和客户端安装完毕之后，需通过 Zabbix Server 添加客户端监控，Zabbix WEB 界面添加客户端监控的操作步骤如下，如图 13-11 所示：

Zabbix-WEB →configuration →hosts →Create host →Host name 和 Agent interfaces，同时选择添加 templates 模板→选择 Add →勾选 Template OS Linux-选择 Add 提交；

注*此处 Host name 名称与 Agentd.conf 配置文件中 Hostname 保持一致，否则会报错。

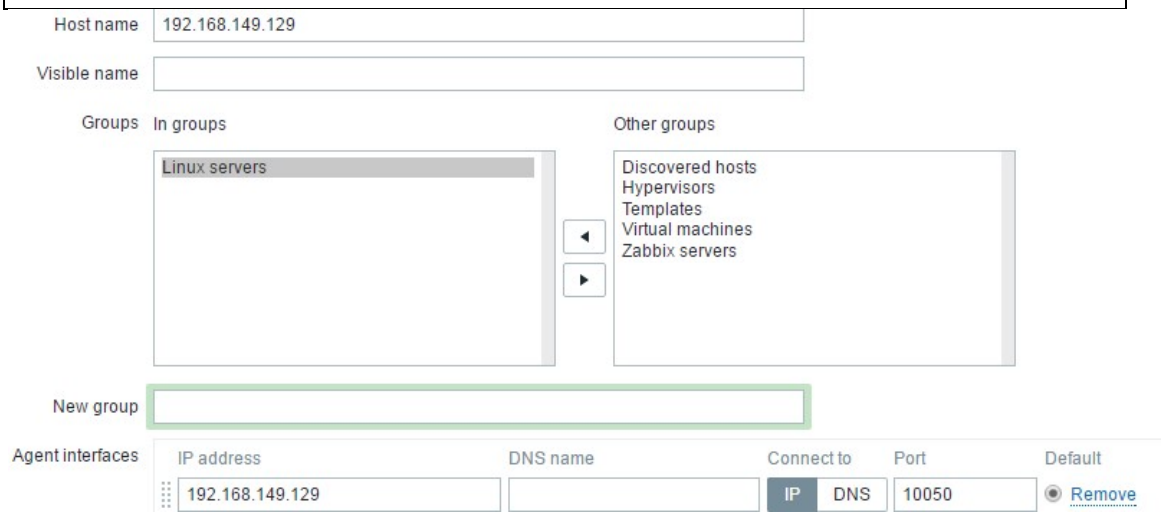


图 13-11 Zabbix 添加客户端监控

将客户端主机链接至“Template OS Linux”，启用模板完成主机默认监控，单击 Add，继续单击 Update 即可，如图 13-12 所示：

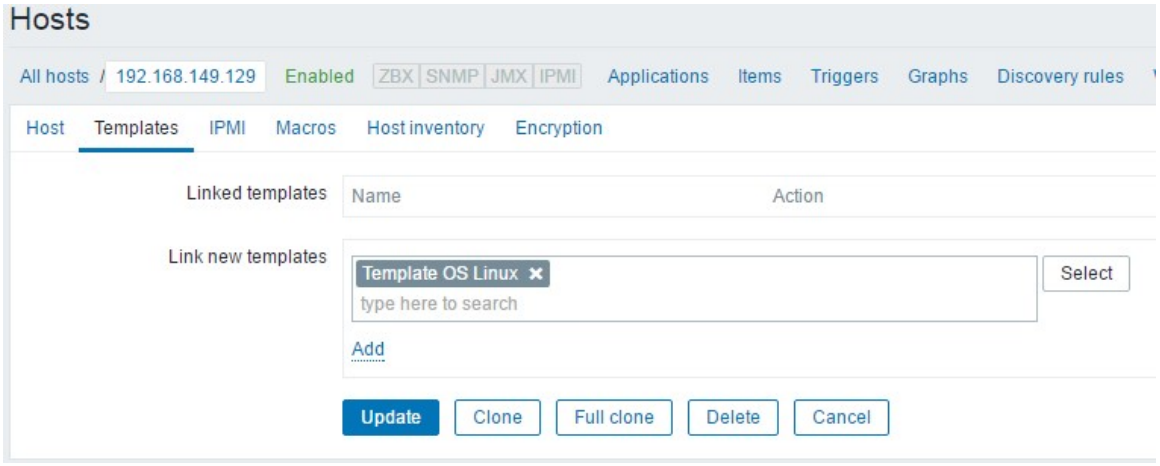


图 13-12 Zabbix 为客户端监控添加模板

单击 Zabbix WEB→Monitoring→Graphs→Group→Host→Graph，监控图像如图 13-13（a）、13-13（b）所示:

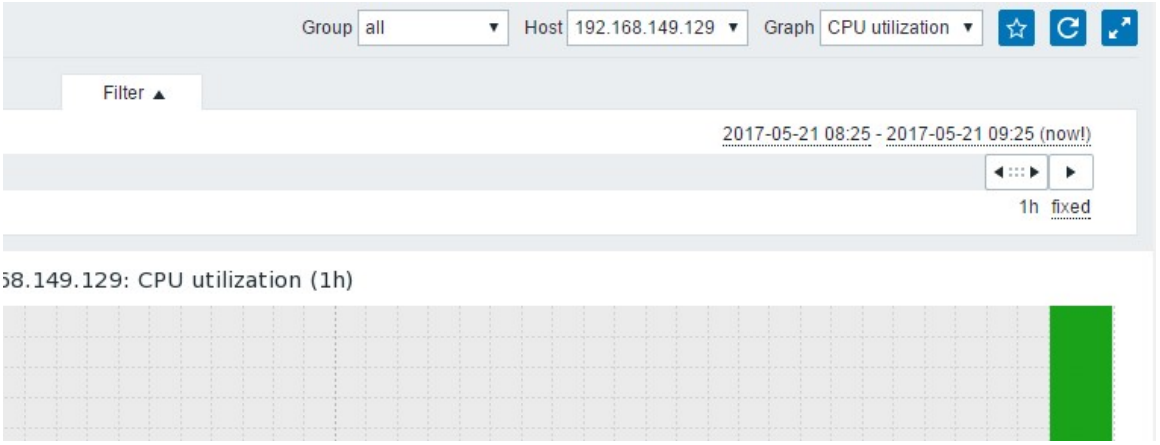


图 13-13（a） Zabbix 客户端监控图像

图 13-13（b） Zabbix 客户端监控图像

如果无法监控到客户端，可以在 Zabbix Server 端，执行命令获取 Agent 的 items KEY

值是否有返回，例如 system.uname 为返回客户端的 uname 信息，监测命令如下：

```
/usr/local/zabbix/bin/zabbix_get  -s  192.168.149.130  -k  system.uname
```

13.6 Zabbix 配置文件详解

Zabbix 监控系统组件分为 Server、Proxy、Agentd 端，对参数的详细了解，能够更加深入理解 Zabbix 监控功能，及对 Zabbix 进行调优，如下为三个组建常用参数详解：

（1） Zabbix_server.conf 配置文件参数详解：

DBHost	数据库主机地址；
DBName	数据库名称；
DBPassword	数据库密码；
DBPort	数据库端口，默认为 3306；
AlertScriptsPath	告警脚本存放路径；
CacheSize	存储监控数据的缓存；
CacheUpdateFrequency	更新一次缓存时间；
DebugLevel	日志级别；
LogFile	日志文件；
LogFileSize	日志文件大小，超过自动切割；
LogSlowQueries	数据库慢查询记录，单位毫秒；
PidFile	PID 文件；
ProxyConfigFrequency	Proxy 被动模式下，Server 多少秒同步配置文件至 proxy；
ProxyDataFrequency	被动模式下，Server 间隔多少秒向 proxy 请求历史数

据；

StartDiscoverers 发现规则线程数；

Timeout 连接 Agent 超时时间；

TrendCacheSize 历史数据缓存大小；

User Zabbix 运行的用户；

HistoryCacheSize 历史记录缓存大小；

ListenIP 监听本机的 IP 地址；

ListenPort 监听端口；

LoadModule 模块名称；

LoadModulePath 模块路径。

（2） Zabbix_Proxy.conf 配置文件参数详解：

ProxyMode Proxy 工作模式，默认为主动模式，主动发送数据至 Server；

Server 指定 Server 端地址；

ServerPort Server 端 PORT；

Hostname Proxy 端主机名；

ListenPort Proxy 端监听端口；

LogFile Proxy 代理端日志路径；

PidFile PID 文件的路径；

DBHost Proxy 端数据库主机名；

DBName Proxy 端数据库名称；

DBUser Proxy 端数据库用户；

DBPassword	Proxy 端数据库密码；
DBSocket	Proxy 数据库 SOCKET 路径；
DBPort	Proxy 数据库端口号；
DataSenderFrequency	Proxy 向 Server 发送数据的时间间隔；
StartPollers	Proxy 程池数量；
StartDiscoverers	Proxy 端自动发现主机的线程数量；
CacheSize	内存缓存配置；
StartDBSyncers	同步数据线程数；
HistoryCacheSize	历史数据缓存大小；
LogSlowQueries	慢查询日志记录，单位为毫秒；
Timeout	超时时间。

（3） Zabbix_agentd.conf 配置文件参数详解：

EnableRemoteCommands	运行服务端远程至客户端执行命令或者脚本；
Hostname	客户端主机名；
ListenIP	监听的 IP 地址；
ListenPort	客户端监听端口；
LoadModulePath	模块路径；
LogFile	日志文件路径；
PidFile	PID 文件名；
Server	指定 Server IP 地址；
ServerActive	Zabbix 主动监控 server 的 ip 地址；
StartAgents	Agent 启动进程，如果设置为 0，表示禁用被动监控；

Timeout	超时时间
User	运行 Zabbix 的用户；
UserParameter	用户自定义 key；
BufferSize	缓冲区大小；
DebugLevel	Zabbix 日志级别。

13.7 Zabbix 自动发现及注册

熟练通过 Zabbix 监控平台监控单台客户端之后，企业中有成千上万台服务器，如果手工添加会非常耗时间，造成大量的人力成本的浪费，有没有什么好的自动化添加客户端的方法呢?

Zabbix 自动发现就是为了解决批量监控而设计的功能之一，什么是自动发现呢，简单来说就是 Zabbix Server 端可以基于设定的规则，自动批量的去发现局域网若干服务器，并自动把服务器添加至 Zabbix 监控平台，省去人工手动频繁的添加，节省大量的人力成本。

Zabbix 相当于 Nagios、Cacti 监控来说，如果要想批量监控，Nagios、Cacti 需要手动单个添加设备、分组、项目、图像，也可以使用脚本，但是不能实现自发方式添加。

Zabbix 最大的特点之一就是可以批量自动主机并监控，利用发现(Discovery)模块，实现自动发现主机、自动将主机添加到主机组、自动加载模板、自动创建项目（Items）、自动创建监控图像，操作步骤如下：

（1） Configuration→discovery → Create discovery rule，如图 13-14 所示：

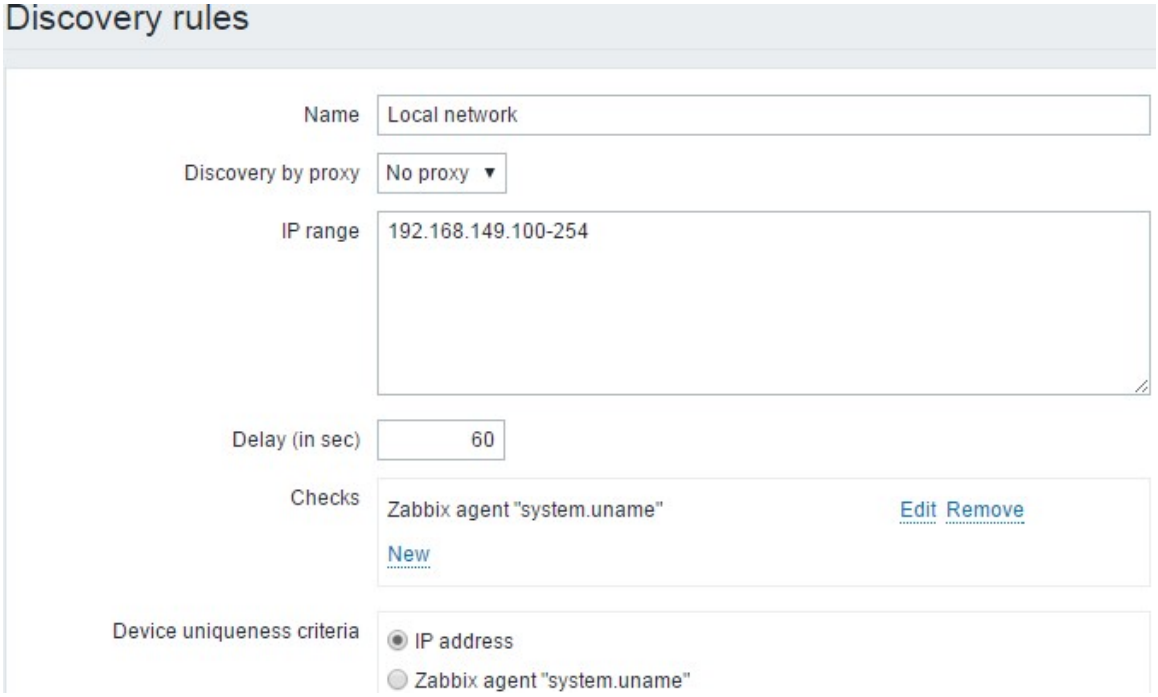


图 13-14 创建客户端发现规则

Name: 规则名称；

Discovery by proxy：通过代理探索；

IP range： zabbix_server 探索区域的 IP 范围；

Delay： 搜索一次的时间间隔；

Checks： 检测方式，如用 ping 方式去发现主机，zabbix_server 需安装 fping，此处使用 Agent 方式发现；

Device uniqueness criteria: 以 IP 地址作为被发现主机的标识。

（2） Zabbix 客户端安装 Agent

由于发现规则里选择 checks 方式为 Agent，所以需在所有被监控的服务器安装 zabbix Agent，安装的方法可以手动安装，也可以使用 Shell 脚本，附 Zabbix 客户端安装脚本，脚本运行方法：sh auto_install_zabbix.sh。

```
#!/bin/bash

#auto install zabbix
```

```
#by jfedu.net 2017
#############
ZABBIX_SOFT="zabbix-3.2.6.tar.gz"
INSTALL_DIR="/usr/local/zabbix/"
SERVER_IP="192.168.149.128"
IP=`ifconfig|grep Bcast|awk '{print $2}'|sed 's/addr://g'`
AGENT_INSTALL(){
yum -y install curl curl-devel net-snmp net-snmp-devel perl-DBI
groupadd zabbix  ;useradd -g zabbix zabbix;usermod -s /sbin/nologin zabbix
tar -xzf $ZABBIX_SOFT;  cd `echo $ZABBIX_SOFT|sed 's/.tar.*//g'`
./configure   --prefix=/usr/local/zabbix   --enable-agent&&make install
if [ $? -eq 0 ];  then
     ln -s /usr/local/zabbix/sbin/zabbix_* /usr/local/sbin/
fi
cd -  ;  cd zabbix-3.2.6
cp   misc/init.d/tru64/zabbix_agentd   /etc/init.d/zabbix_agentd  ;chmod o+x
/etc/init.d/zabbix_agentd
#config zabbix agentd
cat >$INSTALL_DIR/etc/zabbix_agentd.conf<<EOF
LogFile=/tmp/zabbix_agentd.log
Server=$SERVER_IP
ServerActive=$SERVER_IP
```

```
Hostname = $IP

EOF

#start zabbix agentd

/etc/init.d/zabbix_agentd restart

/etc/init.d/iptables stop

setenforce 0

}

AGENT_INSTALL
```

（3） 创建发现 Action

Zabbix 发现规则创建完毕，客户端 Agent 安装完后，被发现的 IP 主机不会自动添加至 Zabbix 监控列表，需要添加发现动作，添加方法如下：

Configuration→ Actions → Event source(选择 Discovery) → Create action

添加规则时，系统默认存在一条发现规则，可以新建规则，也可以编辑默认规则，如图 13-15（a）、13-15（b）、13-15（c）所示，编辑默认发现规则，单击 Operations 设置发现操作，分别设置 Add host、Add to host groups、Link to templates，最后启用规则即可：

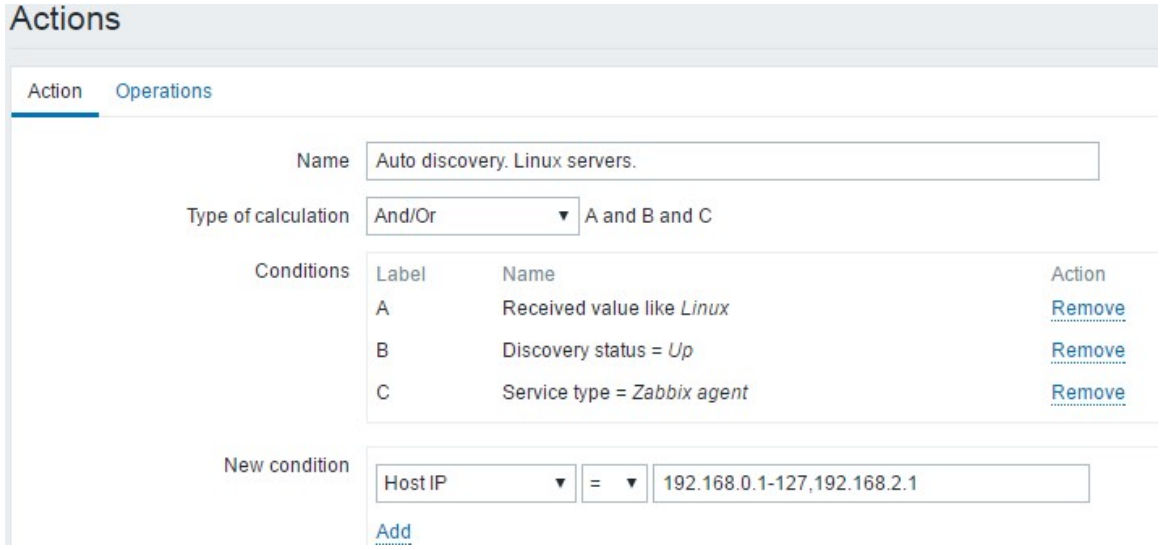


图 13-15（a） 创建客户端发现动作

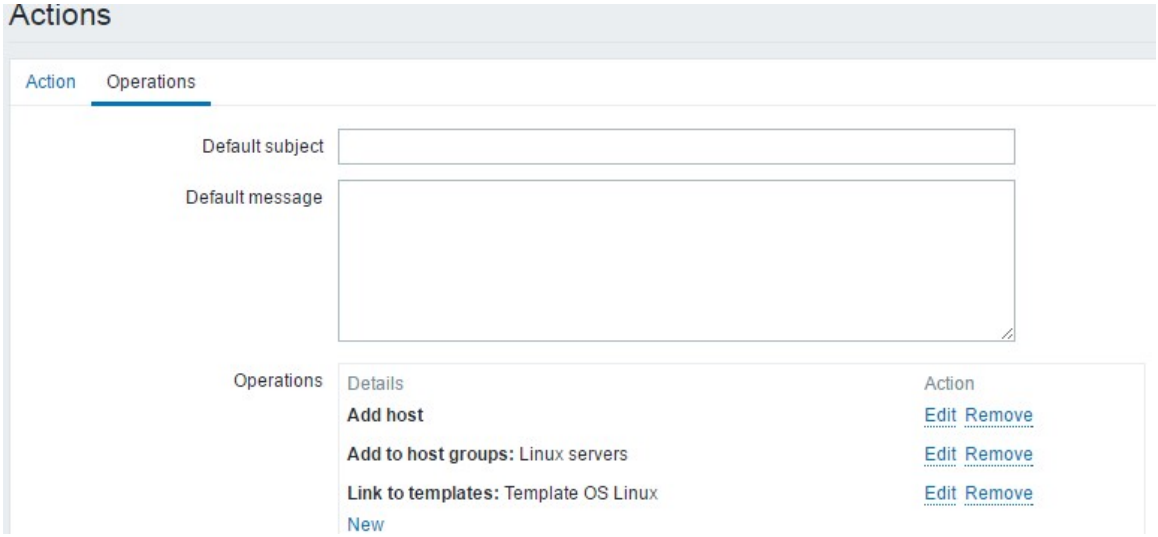


图 13-15（b） 客户端发现自动添加至 Zabbix

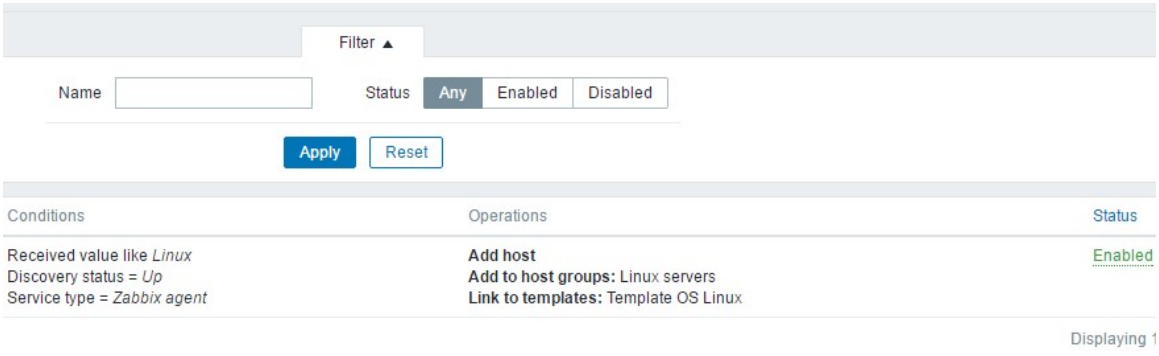


图 13-15（c） 客户端发现自动添加至 Zabbix

Montoring→Discovery，查看通过发现规则找到的服务器 IP 列表，如图 13-16 所示：

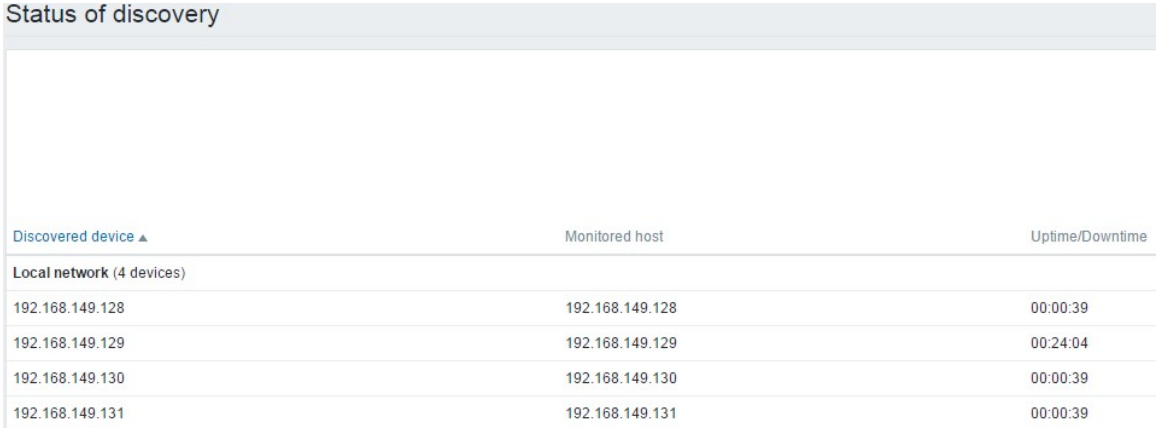


图 13-16 被发现的客户端列表

Configuration→Hosts，查看 4 台主机是否被自动监控至 Zabbix 监控平台，如图 13-17 所示：

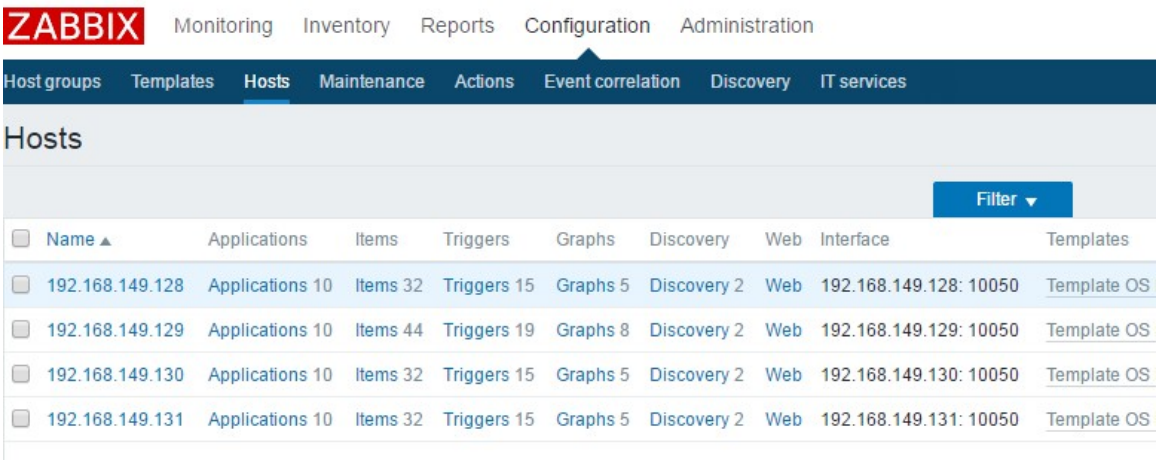


图 13-17 自动发现的主机被添加至 Hosts 列表

Monitoring→Graphs，监控图像查看，如图 13-18（a）、13-18（b）所示，可以选择 Host、Graph 分别查看各种的监控图像：

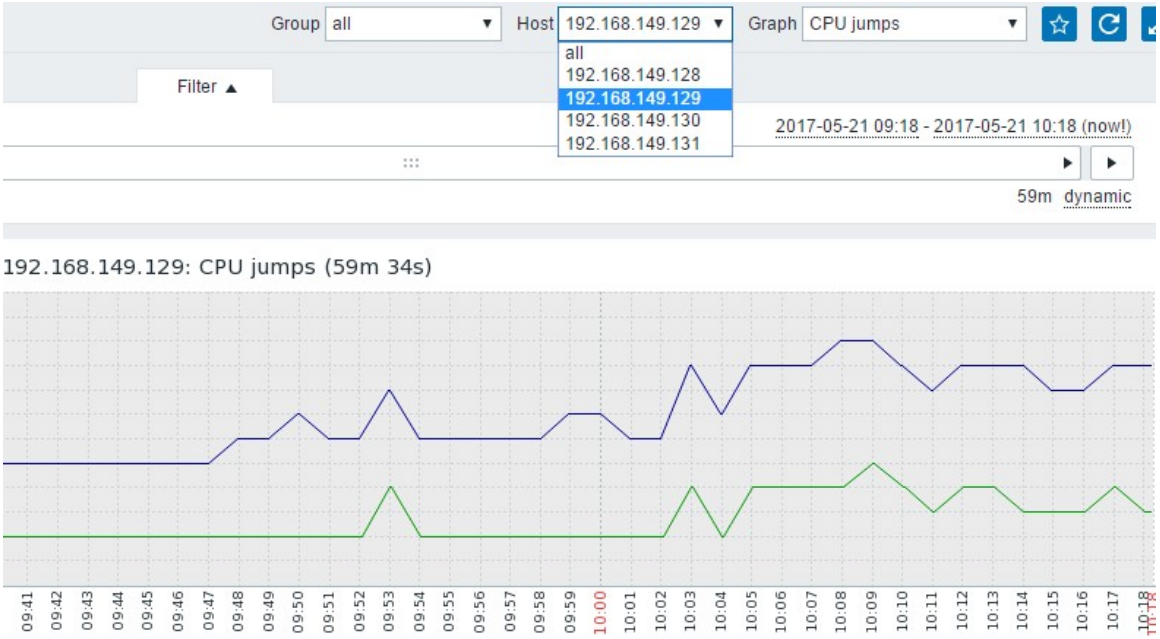


图 13-18（a） 客户端监控图像

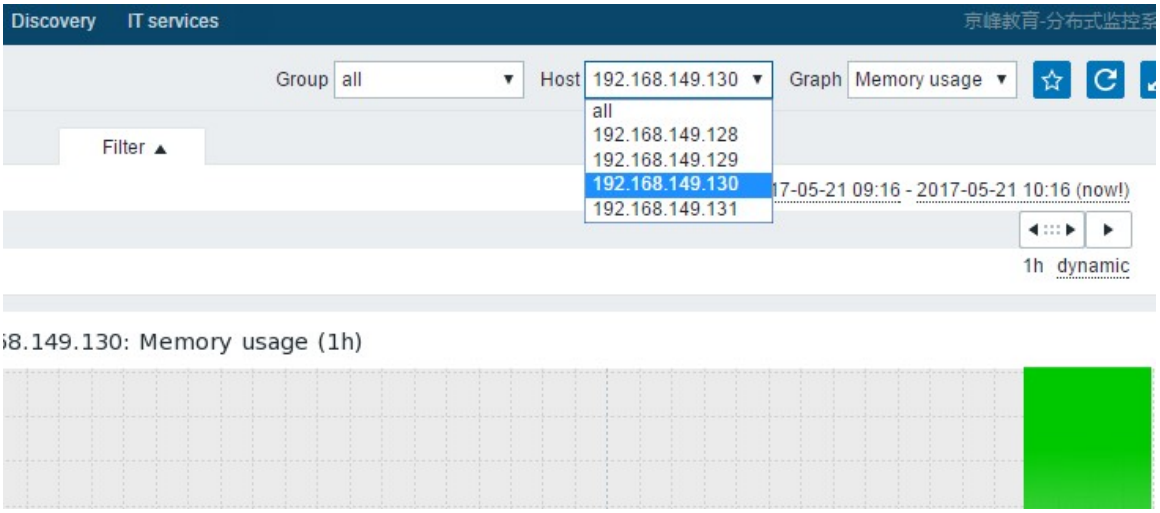


图 13-18（b） 客户端监控图像

13.8 Zabbix 监控 MYSQL 主从实战

Zabbix 监控除了可以使用 Agent 监控客户端服务器状态、CPU、内存、硬盘、网卡流量，同时 Zabbix 还可以监控 MySQL 主从用、监控 LAMP、Nginx WEB 服务器等等，如下为 Zabbix 监控 MYSQL 主从复制的步骤：

（1） 在 Zabbix Agent 端/data/sh 目录创建 Shell 脚本：mysql_ab_check.sh，写入如下代码：

```
#!/bin/bash
/usr/local/mysql/bin/mysql -uroot -e 'show slave status\G' |grep -E
"Slave_IO_Running|Slave_SQL_Running"|awk '{print $2}'|grep -c Yes
```

（2） 在客户端 Zabbix_agentd.conf 配置文件中加入如下代码：

```
UserParameter=mysql.replication,sh /data/sh/mysql_ab_check.sh
```

（3） Zabbix 服务器端获取监控数据，如果返回值为 2，则证明从库 I/O、SQL 线程均为 YES，表示主从同步成功：

```
/usr/local/zabbix/bin/zabbix_get -s 192.168.149.129 -k mysql.replication
```

（4） Zabbix WEB 平台，在 192.168.149.129 hosts 中创建 item 监控项，如图 13-24（a）、13-24（b）所示，单击右上角 create item，Key 填写 Zabbix Agentd 配置文件中的 mysql.replication 即可：

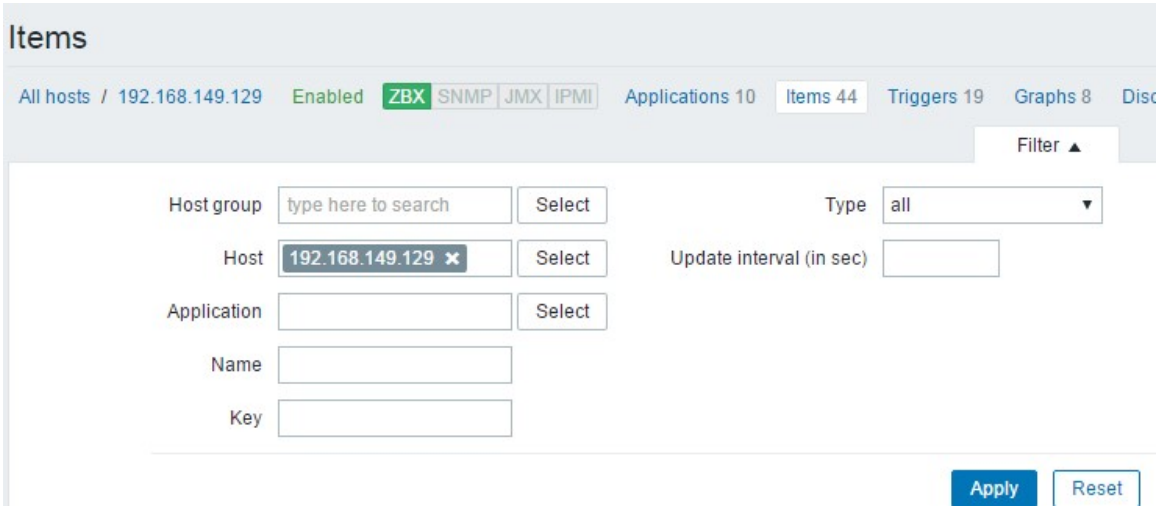


图 13-24（a） Zabbix 添加 MYSQL 主从 item

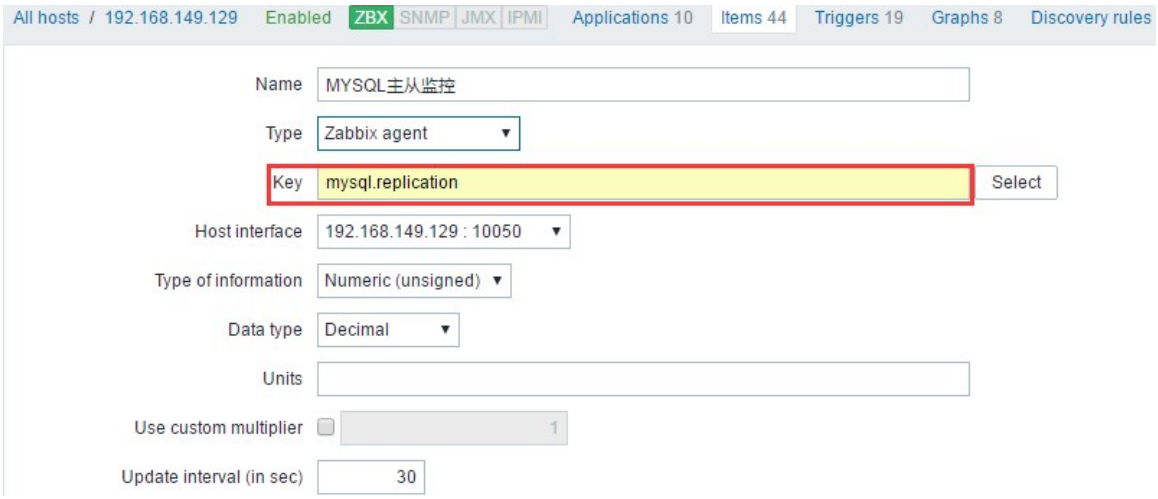


图 13-24（b） Zabbix 添加 MYSQL 主从 item

MYSQL 主从监控项创建 Graph 图像，如图 13-25（a）、13-25（b）所示：

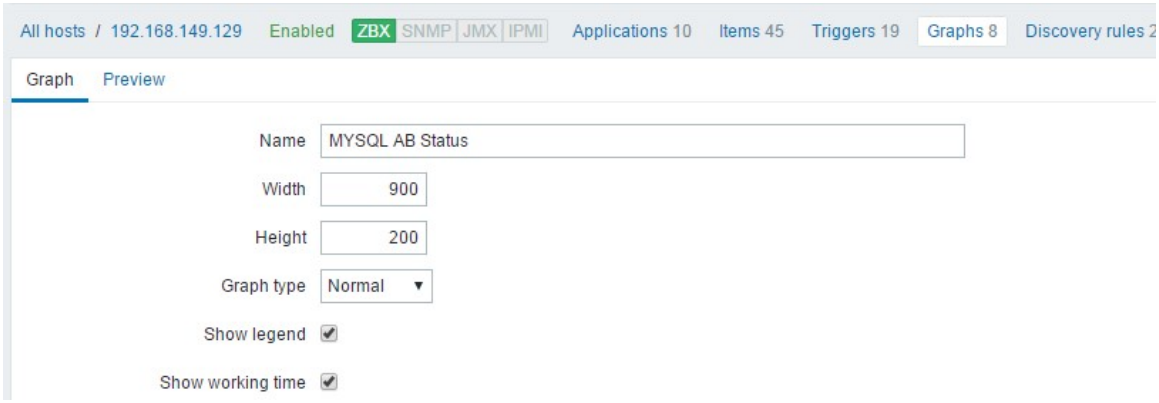


图 13-25（a） 创建 MYSQL 主从监控图像

图 13-25（b） 创建 MYSQL 主从监控图像

MYSQL 主从监控项创建触发器，如图 13-26（a）、13-26（b）所示，MYSQL 主从状态监控，设置触发器条件为 key 值不等于 2 即可，不等于 2 即表示 MYSQL 主从同步状态异常，匹配触发器，执行 Actions：

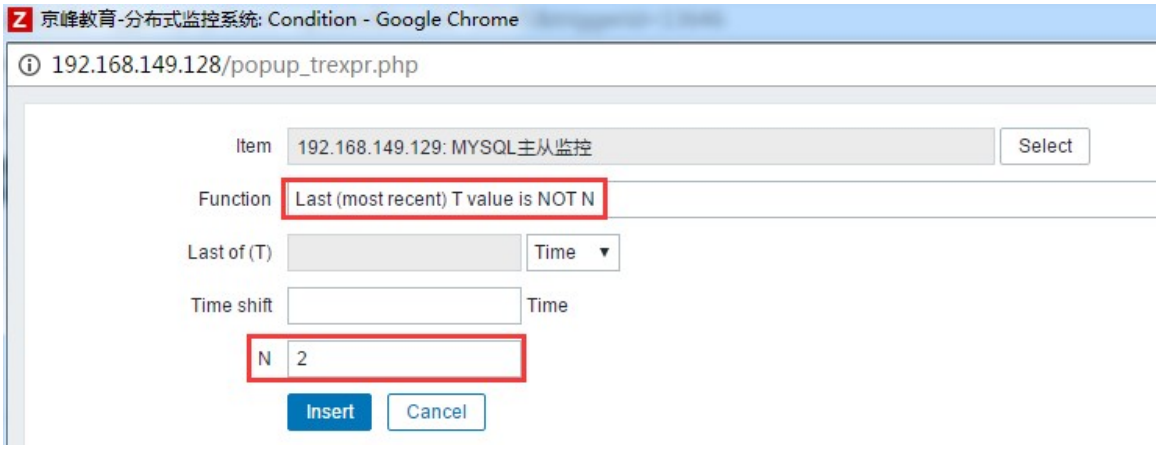


图 13-26（a） 创建 MYSQL 主从监控触发器

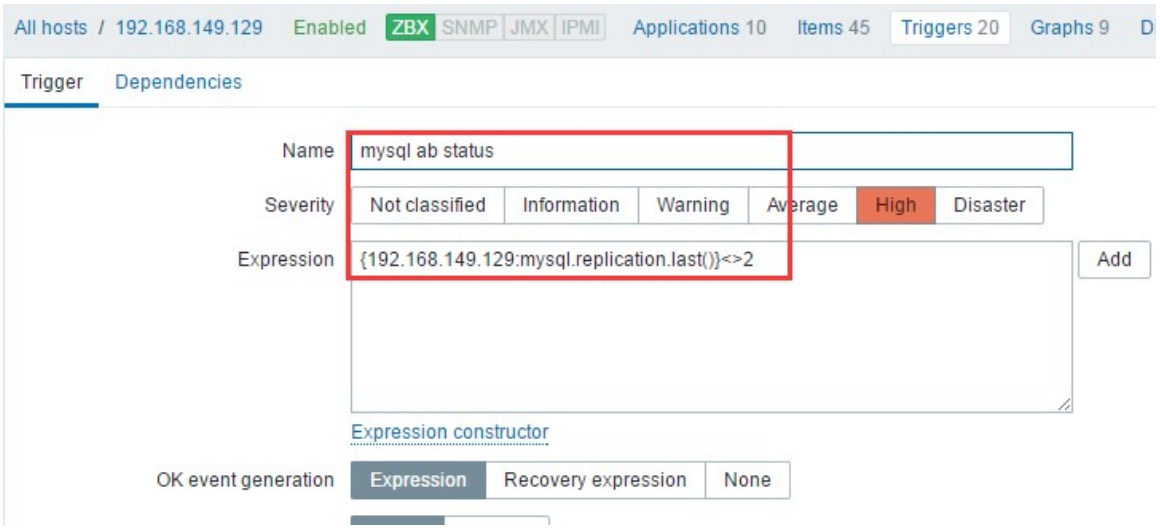


图 13-26（b） 创建 MYSQL 主从监控触发器

如果主从同步状态异常，Key 值不等于 2，会触发邮件报警，报警信息如图 13-27 所示：

图 13-27 MYSQL 主从监控报警邮件

13.9 Zabbix 故障解决&排错

Zabbix 可以设置中文汉化，如果出现乱码解决办法，如果访问 zabbix 出现如下历史记录乱码，WEB 界面乱码，原因是因为数据库导入前不是 UTF-8 字符集，需要修改为 UTF-8 模式，如图 13-28 所示：

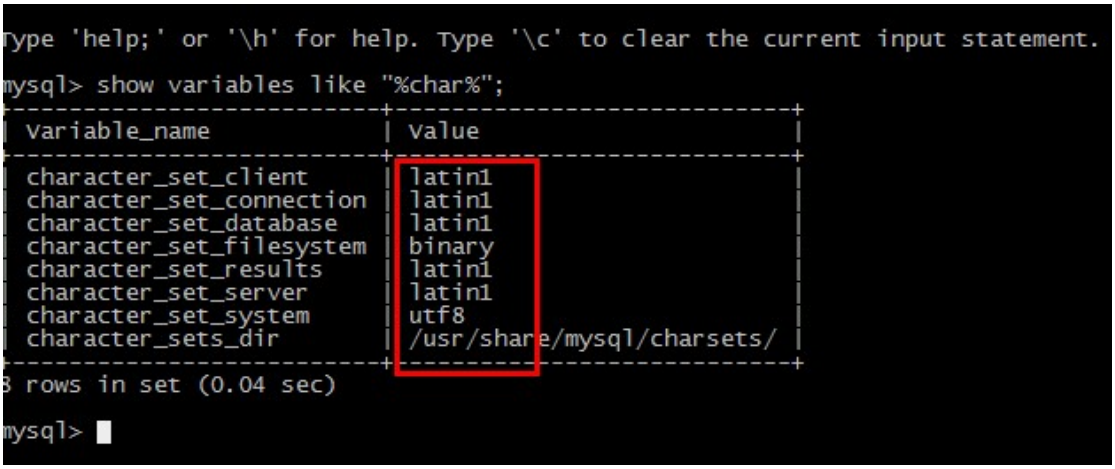


图 13-28 数据库原字符集 latin1

MYSQL 数据库修改字符集方法，vim /etc/my.cnf 在配置段加入如下代码：

```
[mysqld]

character-set-server= utf8

[client]

default-character-set = utf8

[mysql]

default-character-set = utf8
```

备份 zabbix 数据库，并删除原数据库，重新创建，再导入备份的数据库，修改导入的 zabbix.sql 文件里面的 latin1 为 utf8，然后再导入到 zabbix 库，乱码问题解决。

```
sed   -i   's/latin1/utf8/g'   zabbix.sql
```

如果在查看 graphs 监控图像界面的时候时候出现乱码，如图 13-29 所示：

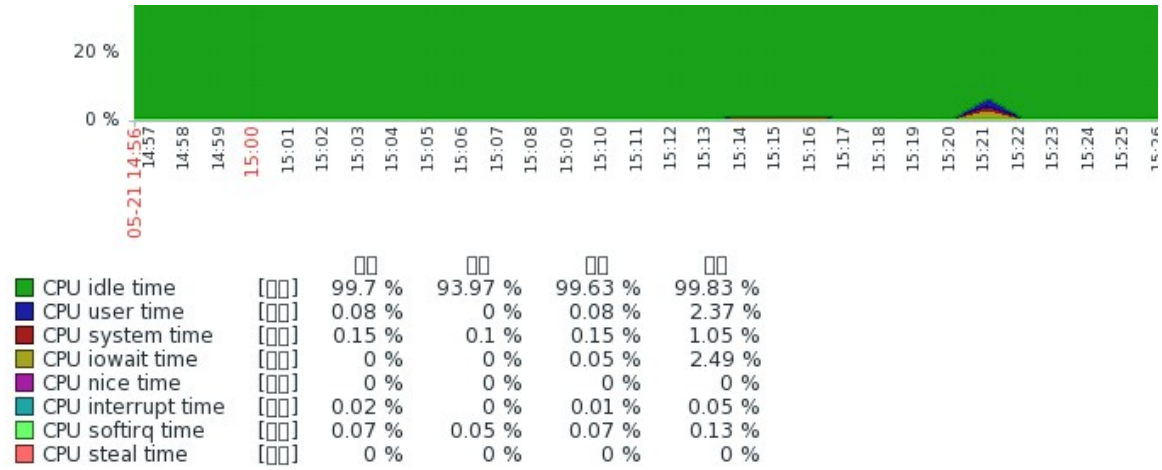


图 13-29 Graphs 图像乱码

从 windows 下控制面板->字体->选择一种中文字库，例如“楷体”，如图 13-30 所示：

图 13-30 上传 Windows 简体中文字体

将字体文件 cp 至 zabbix 服务 dauntfonts 目录下，/var/www/html/zabbix/fonts，并且将 STKAITI.TTF 重命名为 DejaVuSans.ttf，最好刷新 Graph 图像，乱码问题解决，如图 13-31（a）、13-31（b）所示：

```
[root@localhost ~]# cd /var/www/html/fonts/
[root@localhost fonts]#
[root@localhost fonts]#
[root@localhost fonts]# ls
DejaVuSans.ttf
[root@localhost fonts]#
[root@localhost fonts]#
[root@localhost fonts]# rz -y
rz waiting to receive.
 zmodem trl+C ȡ
  100%   12437 KB 12437 KB/s 00:00:01       0 Errors

[root@localhost fonts]# ls
DejaVuSans.ttf  stkaiti.ttf
[root@localhost fonts]# mv stkaiti.ttf DejaVuSans.ttf
mv: overwrite `DejaVuSans.ttf'? y
[root@localhost fonts]#
[root@localhost fonts]# ls
DejaVuSans.ttf
```

图 13-31（a） 上传 Windows 简体中文字体

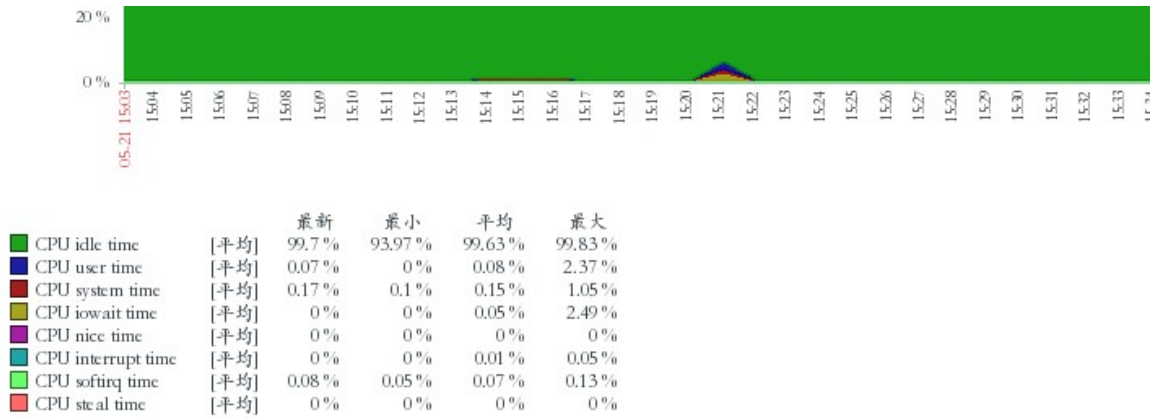


图 13-31（b） Graph 图像乱码问题解决

13.10 Zabbix 触发命令及脚本

Zabbix 监控在对服务或者设备进行监控的时候，如果被监控客户端服务异常，满足触发器，默认可以发送邮件报警、短信报警及微信报警。Zabbix 还可以远程执行命令或者脚本对部分故障实现自动修复。具体可以执行的任务包括：

- 重启应用程序，例如 Apache、Nginx、MySQL、Tomcat 服务等；
- 通过 IPMI 接口重启服务器；
- 删除服务器磁盘空间及数据；

- 执行脚本及资源调度管理；
- 远程命令最大长度为 255 字符；
- 同时支持多个远程命令；
- Zabbix 代理不支持远程命令。

使用 Zabbix 远程执行命令，首先需在 zabbix 客户端配置文件开启对远程命令的支持，在 zabbix_agentd.conf 行尾加入如下代码，并重启服务，如图 13-32 所示：

```
EnableRemoteCommands = 1
```

```
[root@localhost ~]# cd /usr/local/zabbix/etc/
[root@localhost etc]#
[root@localhost etc]# ls
zabbix_agentd.conf  zabbix_agentd.conf.d
[root@localhost etc]#
[root@localhost etc]# vim zabbix_agentd.conf
LogFile=/tmp/zabbix_agentd.log
Server=192.168.149.128
ServerActive=192.168.149.128
Hostname = 192.168.149.129
UserParameter=mysql.replication,sh /data/sh/mysql_ab_check.sh
EnableRemoteCommands = 1
~
```

图 13-32 客户端配置远程命令支持

创建 Action，Configuration→Actions→Triggers，如图 13-33（a）、13-33（b）所示，类型选择“Remote Command”，steps 表示执行命令 1-3 次， step duration 设置每次命令间隔时间的 5 秒执行一次，执行命令方式选择 Zabbix agent，基于 sudo 执行命令即可：

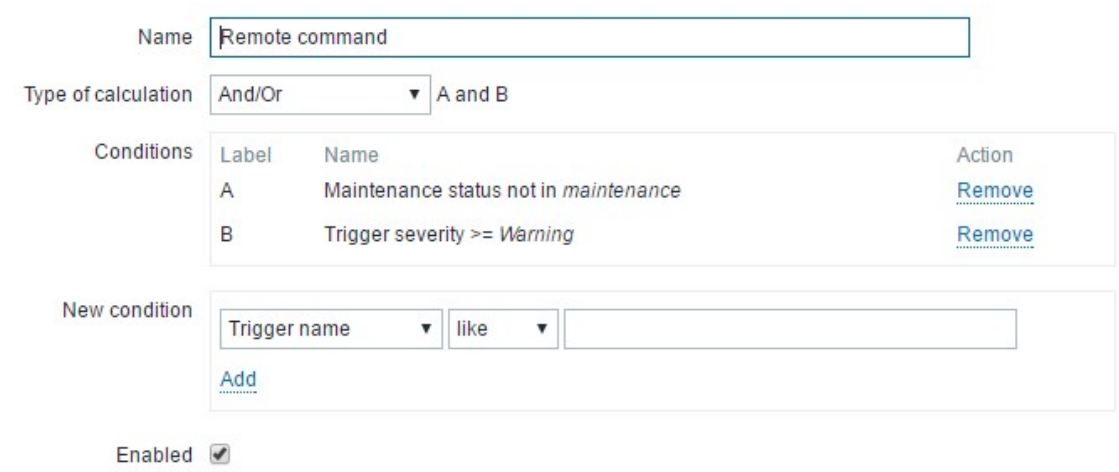


图 13-33（a） 客户端触发器满足条件

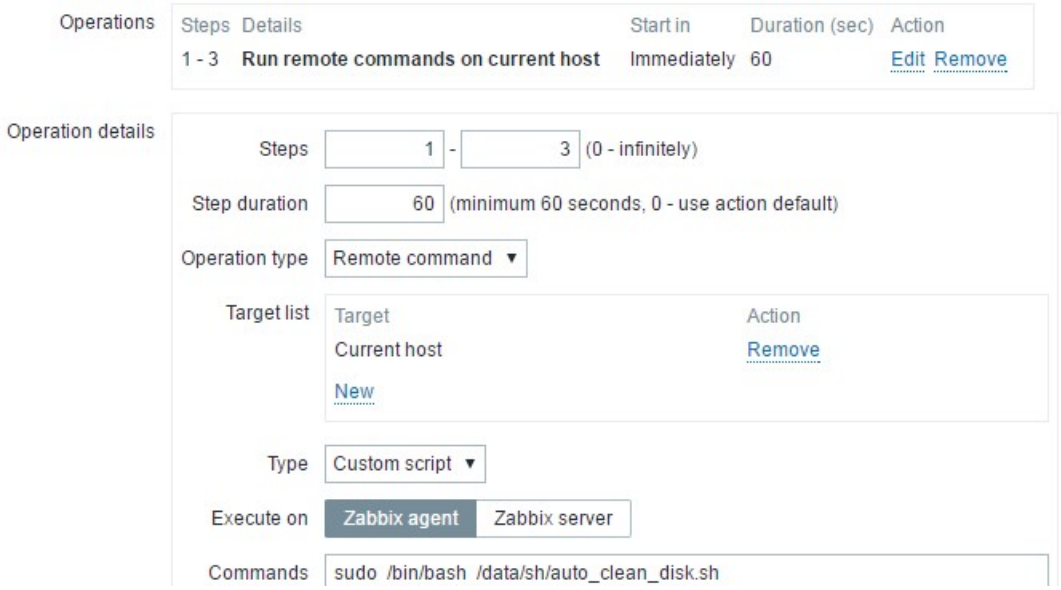


图 13-33（b） Operations 类型选择 Remote Command

Zabbix 客户端 Sudoer 配置文件中添加 zabbix 用户拥有执行权限且无需密码登录:

```
Defaults:zabbix          !requiretty

zabbix    ALL=(ALL)       NOPASSWD: ALL
```

Zabbix 客户端/data/sh/，创建 auto_clean_disk.sh，脚本代码如下:

```
#!/bin/bash

#auto clean disk space

#2017 年 6 月 21 日 10:12:18
```

```
#by author jfedu.net

rm -rf /boot/test.img

find /boot/ -name "*.log" -size +100M -exec rm -rf {} \;
```

将 192.168.149.129 服务器/boot 目录，临时写满，然后满足触发器，实现远程命令执行，查看问题事件命令执行结果，如图 13-34（a）、13-34（b）所示：

图 13-34（a） Remote Command 执行成功

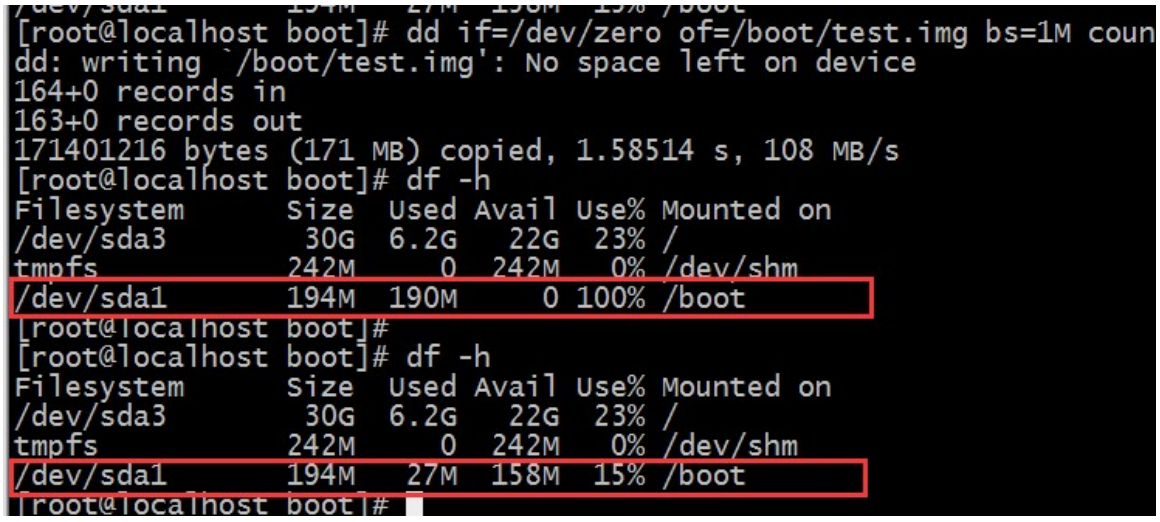


图 13-34（b） Remote Command 执行磁盘清理成功

如果 Zabbix 客户端脚本或者命令没有执行成功，http 服务没有停止,可以在 Zabbix server 端执行如下命令，如图 13-35 所示：

```
/usr/local/zabbix/bin/zabbix_get -s 192.168.149.129 -k "system.run[sudo /etc/init.d/httpd restart]"
```

```
ifconfig eth0
:Ethernet  HWaddr 00:0C:29:3F:F8:29
192.168.149.128  Bcast:192.168.149.255  Mask:255.255.255.0
: fe80::20c:29ff:fe3f:f829/64 Scope:Link
ST RUNNING MULTICAST  MTU:1500  Metric:1
:453673 errors:0 dropped:0 overruns:0 frame:0
:538109 errors:0 dropped:0 overruns:0 carrier:0
:0 txqueuelen:1000
68319859 (160.5 MiB)  TX bytes:239519442 (228.4 MiB)

/usr/local/zabbix/bin/zabbix_get -s 192.168.149.129 -k "system.run[sud
K  ]
d: Could not reliably determine the server's fully qualified domain na
```

图 13-35 测试 Remote Command 命令

13.11 Zabbix 分布式配置实战

Zabbix 是一个分布式监控系统，它可以以一个中心点、多个分节点的模式运行，使用 Proxy 能大大的降低 Zabbix Server 的压力，Zabbix Proxy 可以运行在独立的服务器上，如图 13-36 所示：

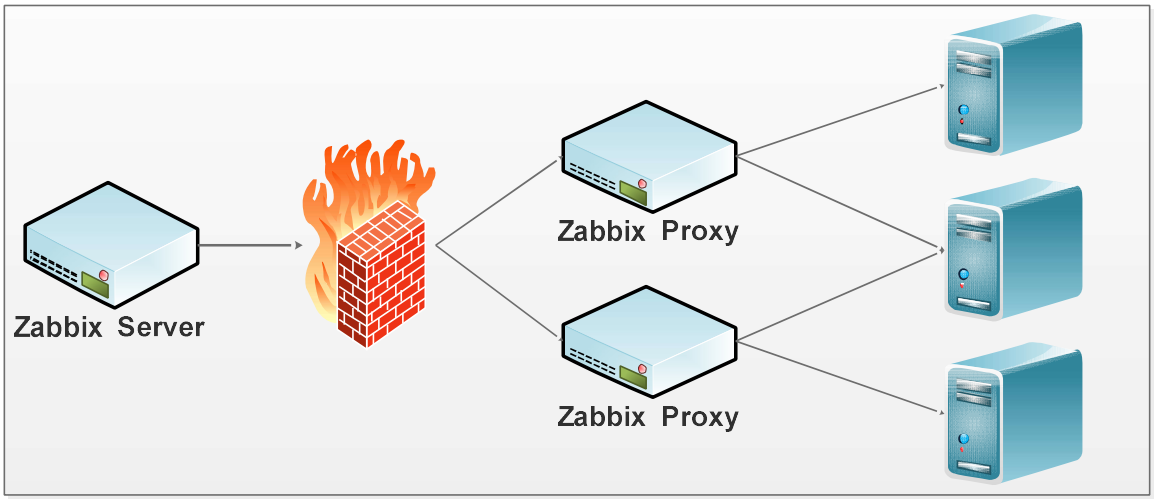


图 13-36 Zabbix Proxy 网络拓扑图

安装 Zabbix Proxy，基于 Zabbix-3.2.6.tar.gz 软件包，同时需要导入 zabbix 基本框架库，具体实现方法如下：

（1） 下载 Zabbix 软件包，代码如下：

```
wget  http://sourceforge.net/projects/zabbix/files/ZABBIX%20Latest%20Stab
```

```
le/3.2.6/zabbix-3.2.6.tar.gz/download
```

（2） Zabbix Proxy 上执行如下代码：

```
yum -y install curl curl-devel net-snmp net-snmp-devel perl-DBI

groupadd zabbix ;useradd -g zabbix zabbix;usermod -s /sbin/nologin zabbix
```

（3） Zabbix Proxy 端配置

创建 zabbix 数据库，执行授权命令：

```
create database zabbix_proxy charset=utf8;

grant all on zabbix_proxy.* to zabbix@localhost identified by '123456';

flush privileges;
```

解压 zabbix 软件包并将 Zabbix 基础 SQL 文件导入数据至 Zabbix 数据库：

```
tar zxvf zabbix-3.2.6.tar.gz

cd zabbix-3.2.6

mysql -uzabbix -p123456 zabbix_proxy <database/mysql/schema.sql

mysql -uzabbix -p123456 zabbix_proxy <database/mysql/images.sql
```

切换至 Zabbix 解压目录，执行如下代码，安装 Zabbix_server：

```
./configure --prefix=/usr/local/zabbix/ --enable-proxy --enable-agent
--with-mysql --enable-ipv6 --with-net-snmp --with-libcurl

make

make install

ln -s /usr/local/zabbix/sbin/zabbix_* /usr/local/sbin/
```

Zabbix Proxy 安装完毕，cd /usr/local/zabbix/etc/目录，如图 13-37 所示：

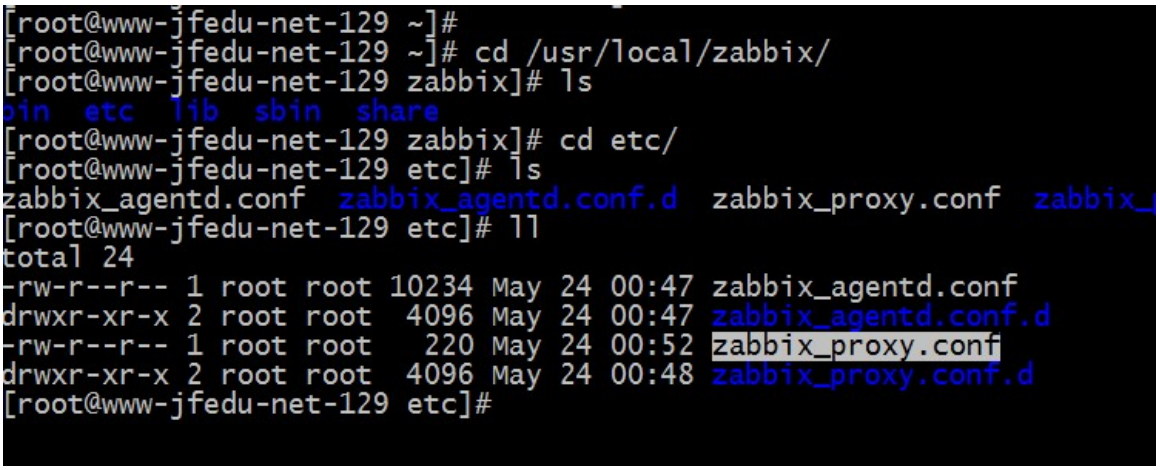


图 13-37 Zabbix Proxy 安装目录

（4） 备份 Zabbix Proxy 配置文件，代码如下：

```
cp  zabbix_proxy.conf  zabbix_proxy.conf.bak
```

（5） 将 zabbix_proxy.conf 配置文件中代码设置为如下：

```
Server=192.168.149.128

Hostname=192.168.149.130

LogFile=/tmp/zabbix_proxy.log

DBName=zabbix_proxy

DBUser=zabbix

DBPassword=123456

Timeout=4

LogSlowQueries=3000

DataSenderFrequency=30

HistoryCacheSize=128M

CacheSize=128M
```

（6） Zabbix 客户端安装 Agent，同时配置 Agent 端 Server 设置为 Proxy 服务器的 IP

地址或者主机名，zabbix_agentd.conf 配置文件代码：

```
LogFile=/tmp/zabbix_agentd.log

Server=192.168.149.130

ServerActive=192.168.149.130

Hostname = 192.168.149.131
```

（7） Zabbix Server WEB 端添加 Proxy，实现集中管理和分布式添加监控，如图 13-38（a）、13-38（b）、13-38（c）所示：

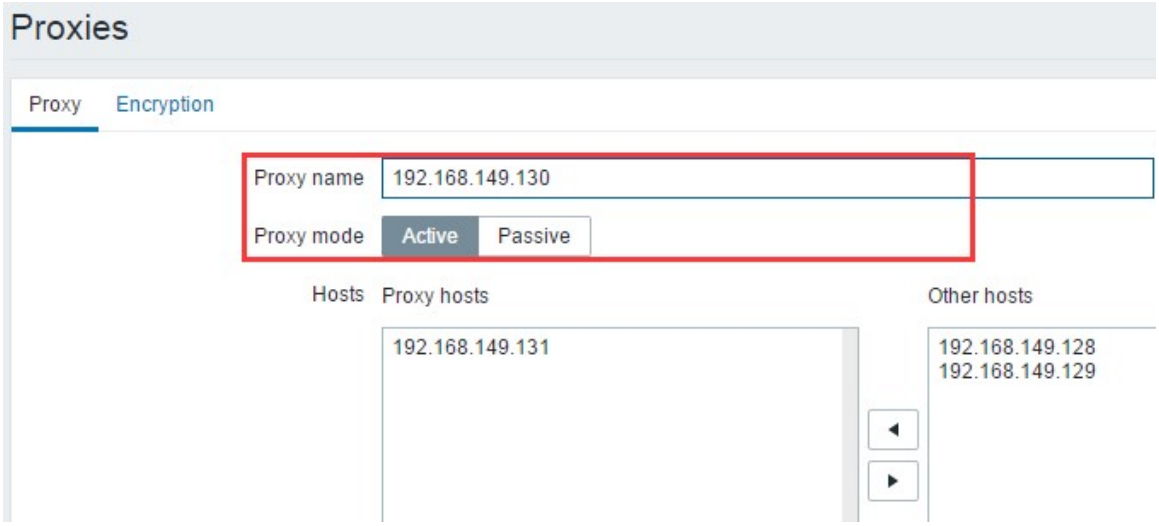


图 13-38（a） Zabbix Proxy WEB 添加

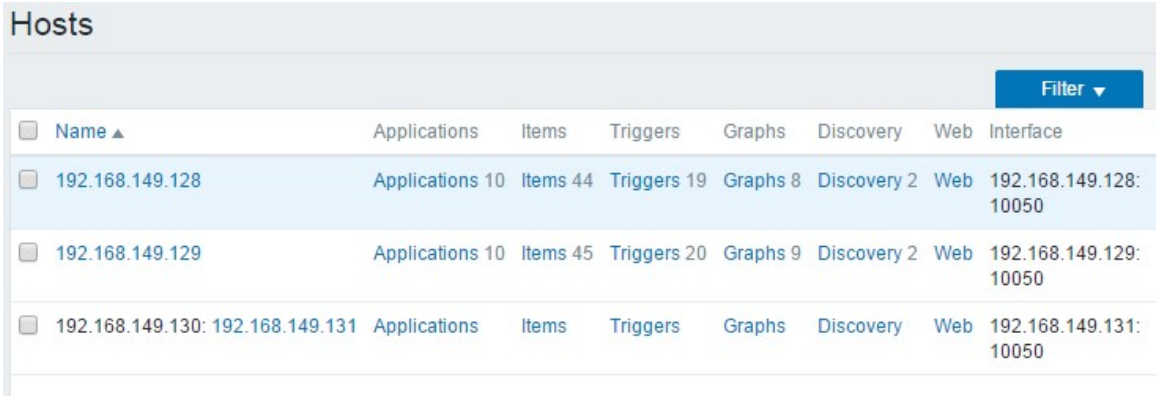


图 13-38（b） Zabbix Proxy 监控客户端

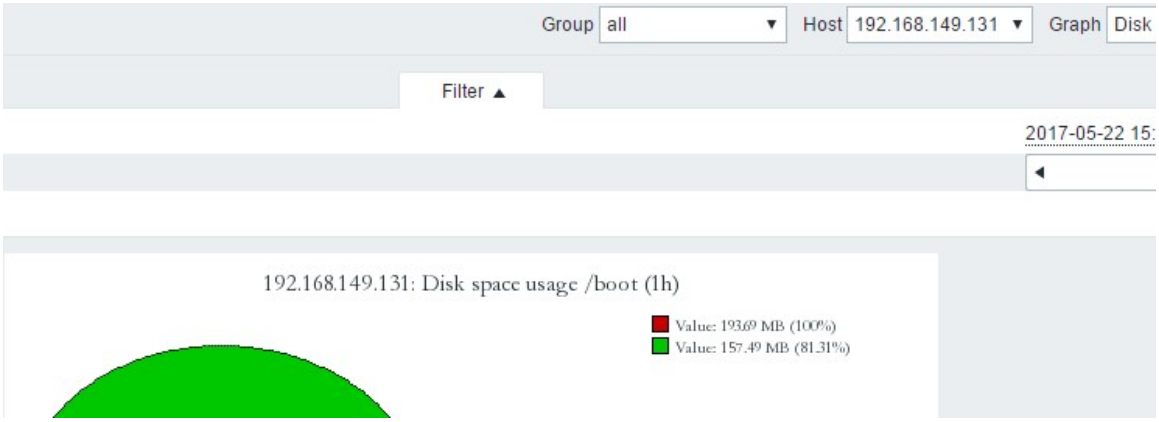


图 13-38（c） Zabbix Proxy 监控客户端图像

13.12 Zabbix 邮件报警实战

Zabbix 监控服务端、客户端都已经部署完成，被监控主机已经添加，Zabiix 监控运行正常，通过查看 Zabbix 监控服务器，可以了解服务器的运行状态是否正常，运维人员不会时刻登录 Zabbix 监控平台刷新，查看服务器的状态。

可以在 Zabbix 服务端设置邮件报警，当被监控主机宕机或者达到设定的触发器预设值时，不管任何时候，会自动发送报警邮件、微信信息到指定的人员，运维人员收到信息有利于第一时间解决故障。Zabbix 邮件报警设置步骤如下：

（1） 设置邮件模板及邮件服务器

Administration→Media types→Create media type，填写邮件服务器信息，根据提示设置完毕，如图 13-19（a）、13-19（b）所示：

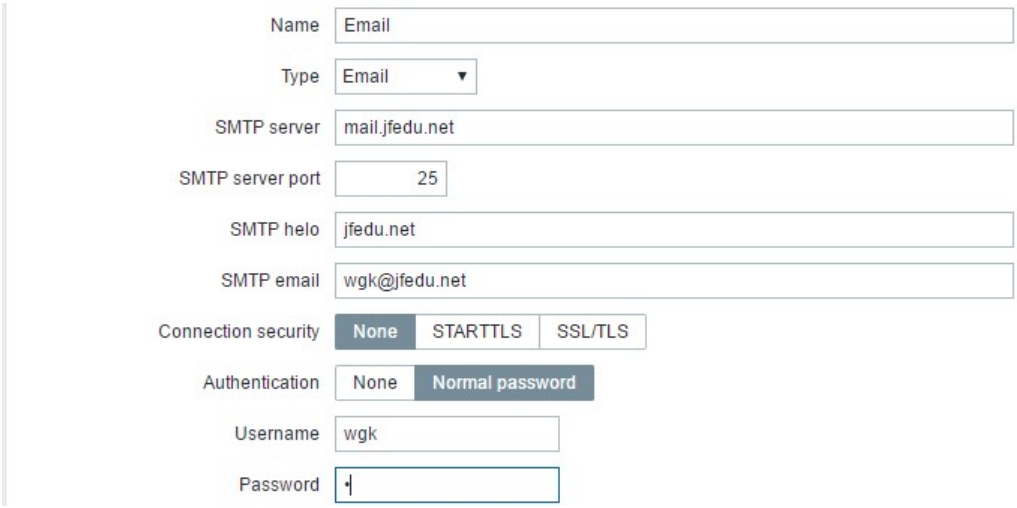


图 13-19（a） Zabbix 邮件报警邮箱设置

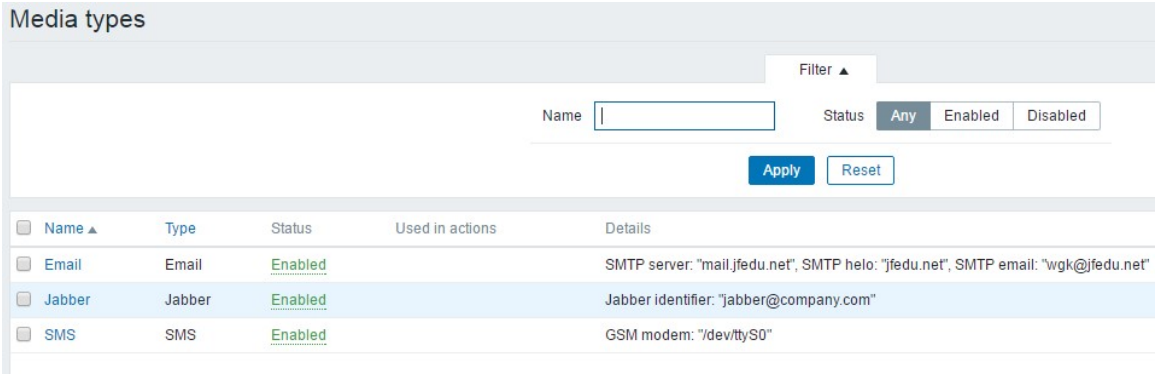


图 13-19（b） Zabbix 邮件报警邮箱设置

（2） 配置接收报警的邮箱

Administration-user-Admin (Zabbix Administrator)-user-admin，选择 Media，单击 Add 添加发送邮件的类型“Email”，同时指定接收邮箱地址：wgkgood@163.com，根据实际需求改成自己的接收人，如图 13-20 所示：

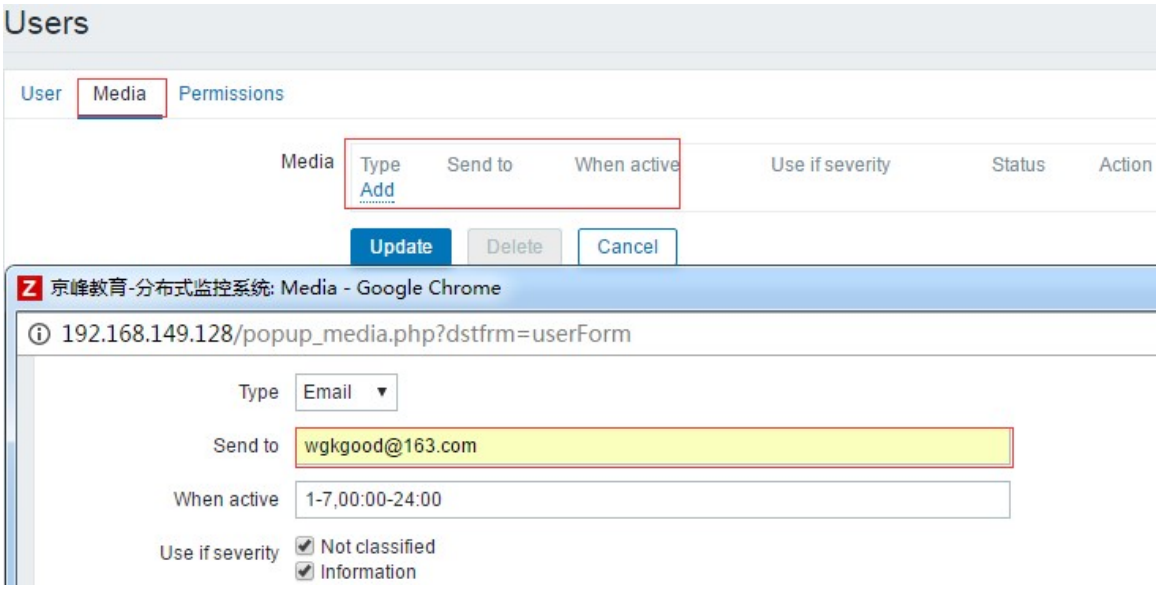


图 13-20 Zabbix 邮件报警添加接收人

（3） 添加报警触发器

Configuration→Actions→Action→ Event source→Triggers-Create Action，如图 13-21（a）、13-21（b）、13-21（c）所示，分别设置 Action、Operations、Recovery operations。

- Action→New condition 选择“Trigger serverity>=Warning”；
- Operations→设置报警间隔为 60s，自定义报警信息，报警信息发送至 administrators 组；
- Recovery operations →自定义恢复信息，恢复信息发送至 administrators 组。

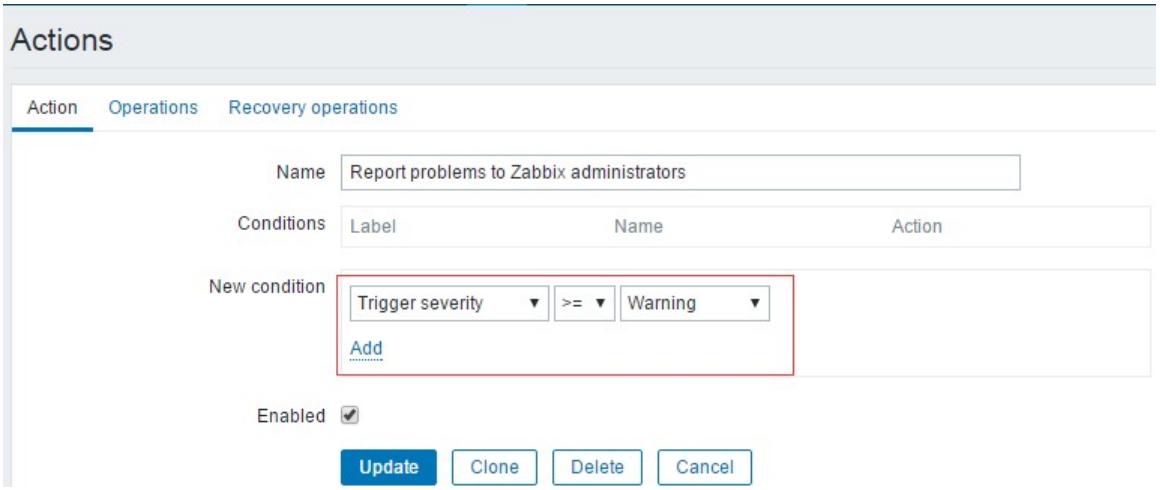


图 13-21（a） 邮件报警 Action 设置

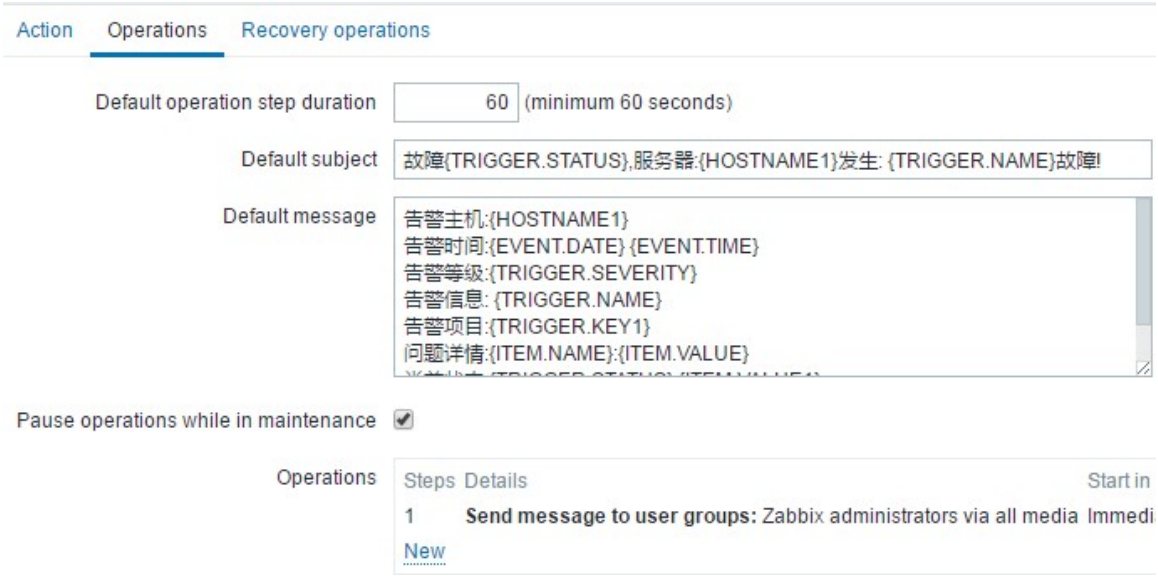


图 13-21（b） 邮件报警 Operations 设置

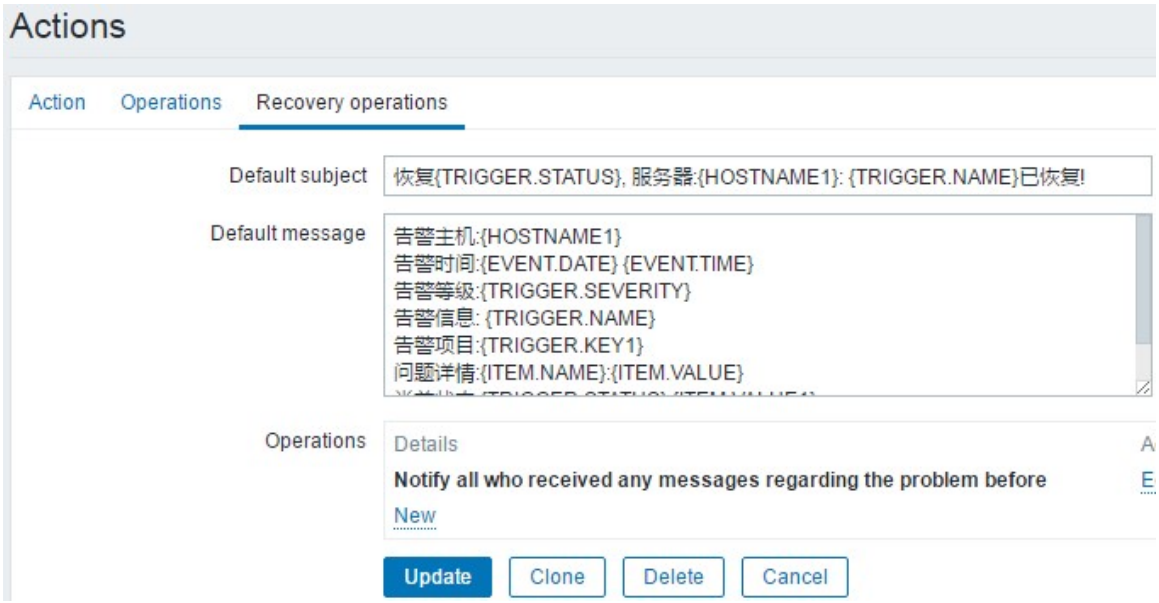


图 13-21（c） 邮件报警 Recovery Operations 设置

报警邮件标题可以使用默认信息，亦可使用如下中文报警内容：

名称：Action-Email

默认标题：故障{TRIGGER.STATUS},服务器:{HOSTNAME1}发生: {TRIGGER.NAME}故障!

默认信息：

告警主机:{HOSTNAME1}

告警时间:{EVENT.DATE} {EVENT.TIME}

告警等级:{TRIGGER.SEVERITY}

告警信息: {TRIGGER.NAME}

告警项目:{TRIGGER.KEY1}

问题详情:{ITEM.NAME}:{ITEM.VALUE}

当前状态:{TRIGGER.STATUS}:{ITEM.VALUE1}

事件 ID:{EVENT.ID}

恢复邮件标题可以使用默认信息，亦可使用如下中文报警恢复内容：

恢复标题：恢复{TRIGGER.STATUS}，服务器:{HOSTNAME1}: {TRIGGER.NAME}已恢复!

恢复信息：

告警主机:{HOSTNAME1}

告警时间:{EVENT.DATE} {EVENT.TIME}

告警等级:{TRIGGER.SEVERITY}

告警信息: {TRIGGER.NAME}

告警项目:{TRIGGER.KEY1}

问题详情:{ITEM.NAME}:{ITEM.VALUE}

当前状态:{TRIGGER.STATUS}:{ITEM.VALUE1}

事件 ID:{EVENT.ID}

Monitoring→Problems，检查有问题的 Action 事件，单击 Timex 下方时间，如图 13-22（a）、13-22（b）所示，可以看到邮件是否执行成功或者失败：

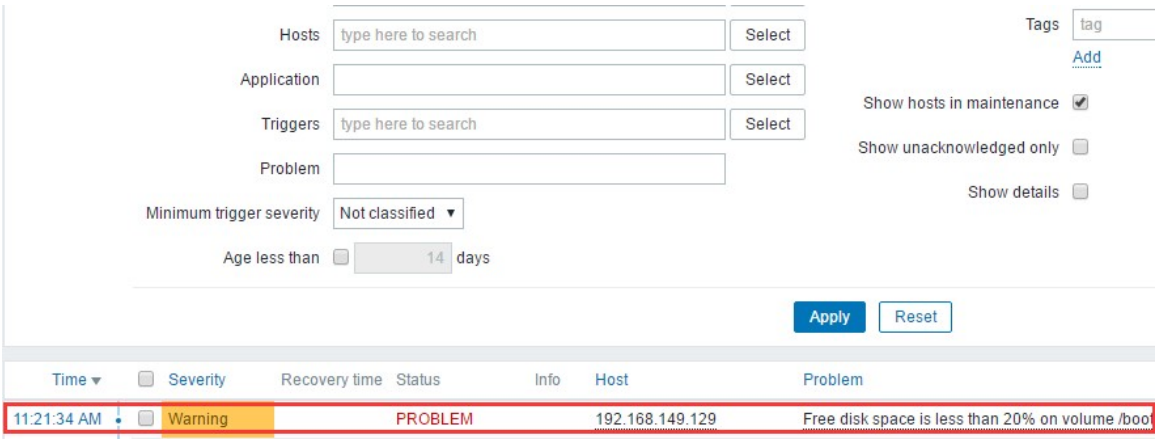


图 13-22（a） Zabbix 查看有问题的事件

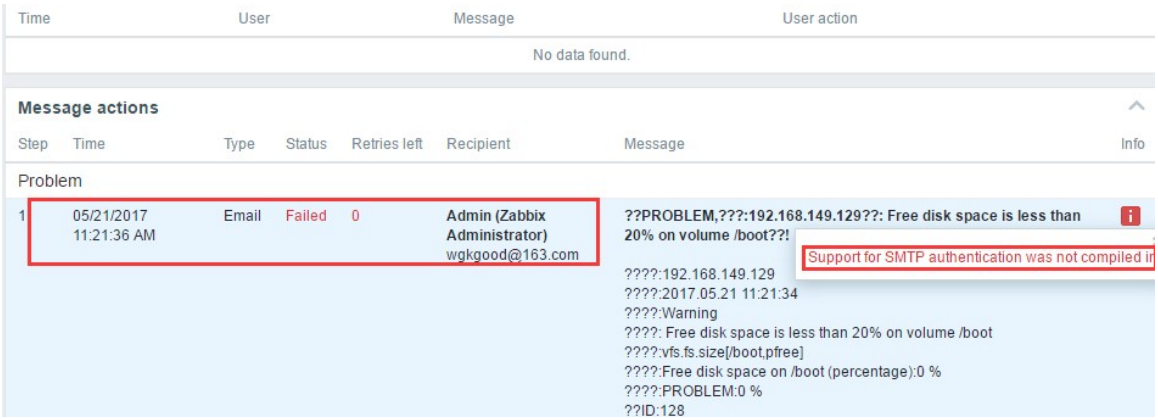


图 13-22（b） Zabbix 有问题的事件执行任务

Zabbix 邮件发送失败，报错 Support for SMTP authentication was not compiled in，原因是由于 Zabbix CURL 版本要求至少是 7.20+版本，升级 CURL，升级方法：

```
wget http://mirror.city-fan.org/ftp/contrib/yum-repo/city-fan.org-release-1-13.rhel6.noarch.rpm
rpm -ivh city-fan.org-release-1-13.rhel6.noarch.rpm
yum upgrade libcurl –y
curl -V
```

CURL 升级完毕之后，测试邮件发送，还是报同样的错误，原因是因为需要重新将 Zabbix_Server 服务通过源码编译安装一遍，安装完 zabbx_server，重启服务，乱码问题

是由于数据库字符集需改成UTF-8格式,同时将数Zabbix库导出,然后修改latin1为utf8,再将 SQL 导入，重启 Zabbix 即可，最终如图 13-23（a）、13-23（b）、13-23（c）所示:

Time	Type	Status	Retries left	Recipient	Message
m					
05/21/2017 12:15:03 PM	Email	Sent		Admin (Zabbix Administrator) wgkgood@163.com	故障PROBLEM,服务器:192.168.149.130发生: Free disk spa than 20% on volume /boot故障! 告警主机:192.168.149.130 告警时间:2017.05.21 12:15:01 告警等级:Warning 告警信息: Free disk space is less than 20% on volume /boot 告警项目:vfs.fs.size[/boot,pfree] 问题详情:Free disk space on /boot (percentage):0 % 当前状态:PROBLEM:0 % 事件ID:226

图 13-23（a） Zabbix 事情发送邮件进程

故障PROBLEM,服务器:192.168.149.130发生: Free disk space is less than 20% on volume /boot故障!

发件人：wgk<wgk@jfedu.net>

收件人：我<wgkgood@163.com>

时　间：2017年05月21日 12:15 (星期日)

告警主机:192.168.149.130
告警时间:2017.05.21 12:15:01
告警等级:Warning
告警信息: Free disk space is less than 20% on volume /boot
告警项目:vfs.fs.size[/boot,pfree]
问题详情:Free disk space on /boot (percentage):0 %
当前状态:PROBLEM:0 %
事件ID:226

图 13-23（b） Zabbix 监控故障 item 发送报警邮件

恢复OK, 服务器:192.168.149.130: Free disk space is less than 20% on volume /boot已恢复!

发件人：wgk<wgk@jfedu.net>

收件人：我<wgkgood@163.com>

时　间：2017年05月21日 12:13 (星期日)

告警主机:192.168.149.130
告警时间:2017.05.21 12:03:01
告警等级:Warning
告警信息: Free disk space is less than 20% on volume /boot
告警项目:vfs.fs.size[/boot,pfree]
问题详情:Free disk space on /boot (percentage):85.74 %
当前状态:OK:85.74 %
事件ID:194

图 13-23（c） Zabbix 监控故障 item 恢复发送邮件

13.13 Zabbix 微信报警实战

Zabbix 除了可以使用邮件报警之外，还可以通过多种方式把告警信息发送到指定人，例如短信报警方式，越来越多的企业开始使用 Zabbix 结合微信作为主要的告警方式，因为每个人每天都在使用微信，这样可以及时有效的把告警信息推送到接收人，方便告警的及时处理。Zabbix 微信报警怎么设置呢，设置的步骤有哪些呢，方法步骤如下：

（1） 微信企业号注册

企业号注册地址：https://qy.weixin.qq.com/填写企业注册信息，等待审核完，并且微信扫描登录企业公众号，如图 13-39（a）、13-39（b）所示：

图 13-39（a） 微信企业公众号注册

图 13-39（b） 微信企业公众号登录

（2） 通讯录添加运维部门及人员

登录新建的企业号，通过提前把企业成员信息添加到组织或者部门，需要填写手机号、微信号或邮箱，通过这样方式让别人扫码关注企业公众号，为了后面企业号推送消息给企业成员，如图 13-40（a）、13-40（b）所示：

图 13-40（a） 微信企业公众号通讯录

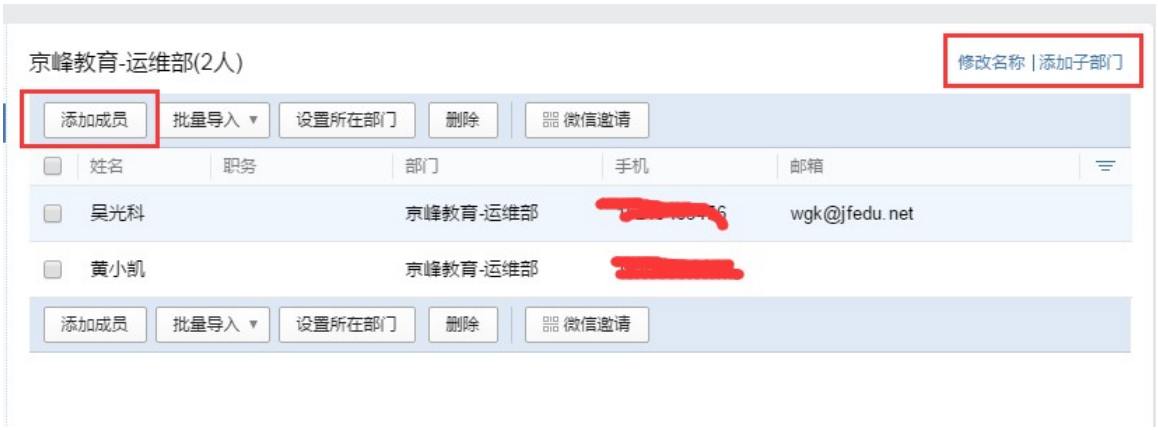


图 13-40（b） 微信企业公众号通讯录

（3） 企业应用-创建应用

除了对个人添加微信报警之外,还可以添加不同管理组，接受同一个应用推送的消息，成员账号，组织部门 ID，应用 Agent ID，CorpID 和 Secret，调用 API 接口需要用到这些信息，如图 13-41（a）、13-41（b）、13-41（c）所示:

图 13-41（a） 微信企业公众号创建应用

图 13-41（b） 微信企业公众号创建应用

图 13-41（c） 微信企业公众号创建应用

（4） 获取企业 CorpID，单击企业公众号首页“我的企业”，即可看到，如图 13-42 所示：

图 13-42 微信企业公众号 CorpID

（5） 微信接口调试，调用微信接口需要一个调用接口的凭证：Access_token 通过 CorpID 和 Secret 可以获得 Access_token，微信企业号接口调试地址：http://qydev.weixin.qq.com/debug，如图 13-43（a）、13-43（b）所示：

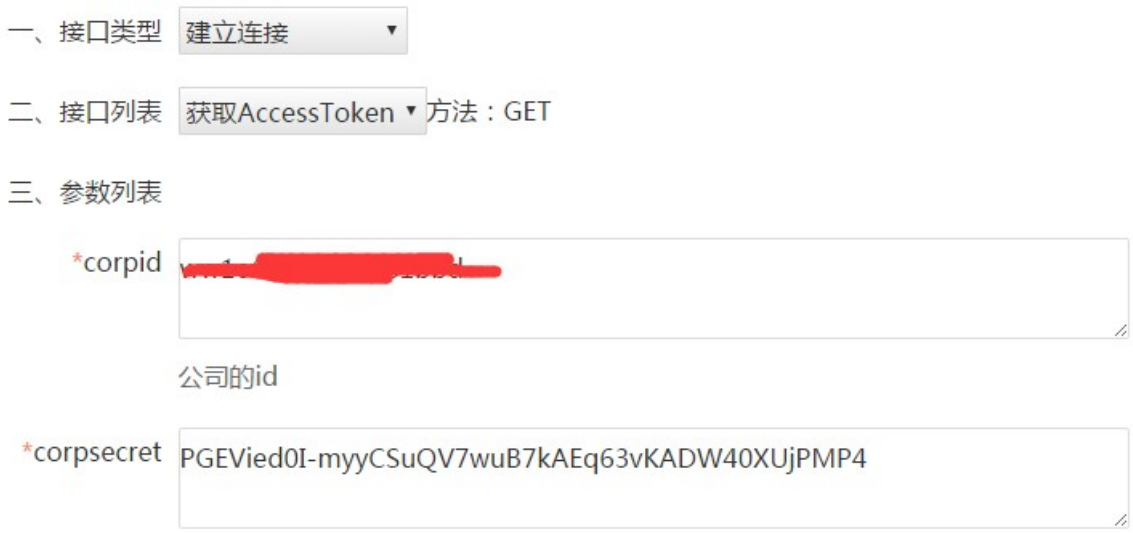


图 13-43（a） 微信企业公众号调试

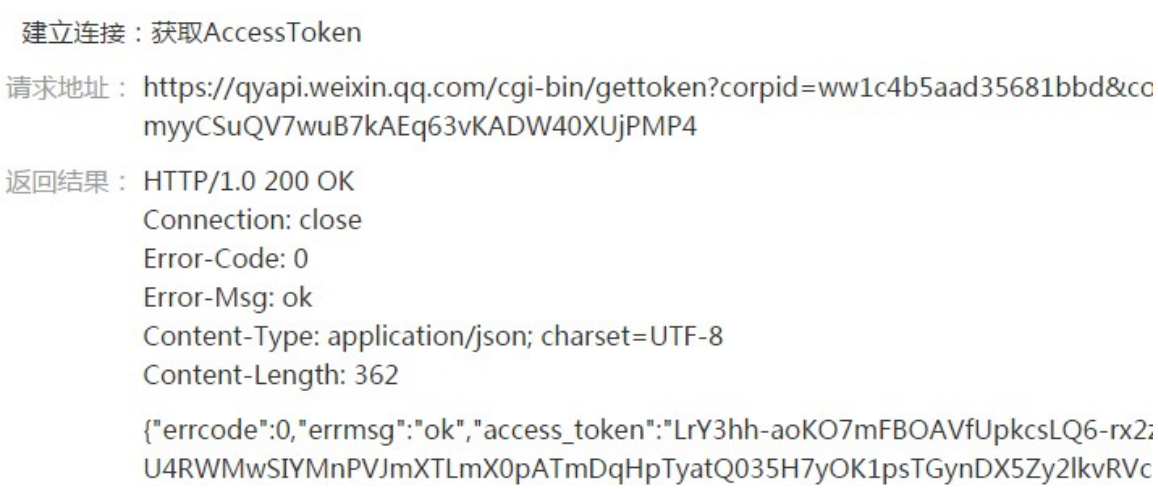


图 13-43（b） 微信企业公众号调试

（6） 获取微信报警工具

```
mkdir  –p /usr/local/zabbix/alertscripts

cd  /usr/local/zabbix/alertscripts

wget http://dl.cactifans.org/tools/zabbix_weixin.x86_64.tar.gz
```

```
tar   zxvf zabbix_weixin.x86_64.tar.gz

mv   zabbix_weixin/weixin   .

chmod o+x weixin

mv zabbix_weixin/weixincfg.json /etc/

rm -rf zxvf zabbix_weixin.x86_64.tar.gz

rm -rf zabbix_weixin/
```

修改/etc/ weixincfg.json 配置文件中 corpid、secret、agentid，并测试脚本发送信息，如图 13-44（a）、13-44（b）所示：

```
cd   /usr/local/zabbix/alertscripts

./weixin   wuguangke 京峰教育报警测试   Zabbix 故障报警

./weixin   contact    subject    body

标准信息格式：

Contact，为你的微信账号，注意不是微信号，不是微信昵称，可以把用户账号设置成微信号或微信昵称，Subject 告警主题，Body    告警详情。
```

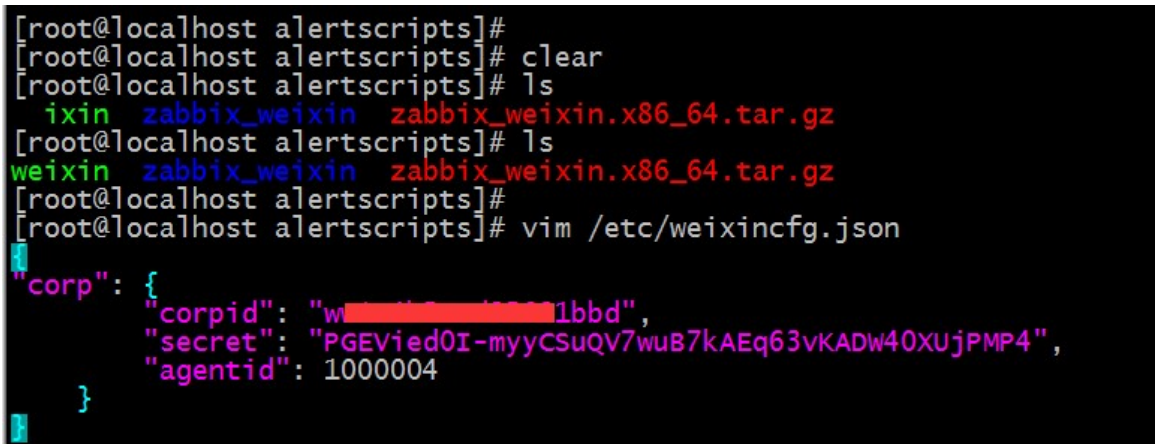


图 13-44（a） Zabbix Server 端微信配置文件

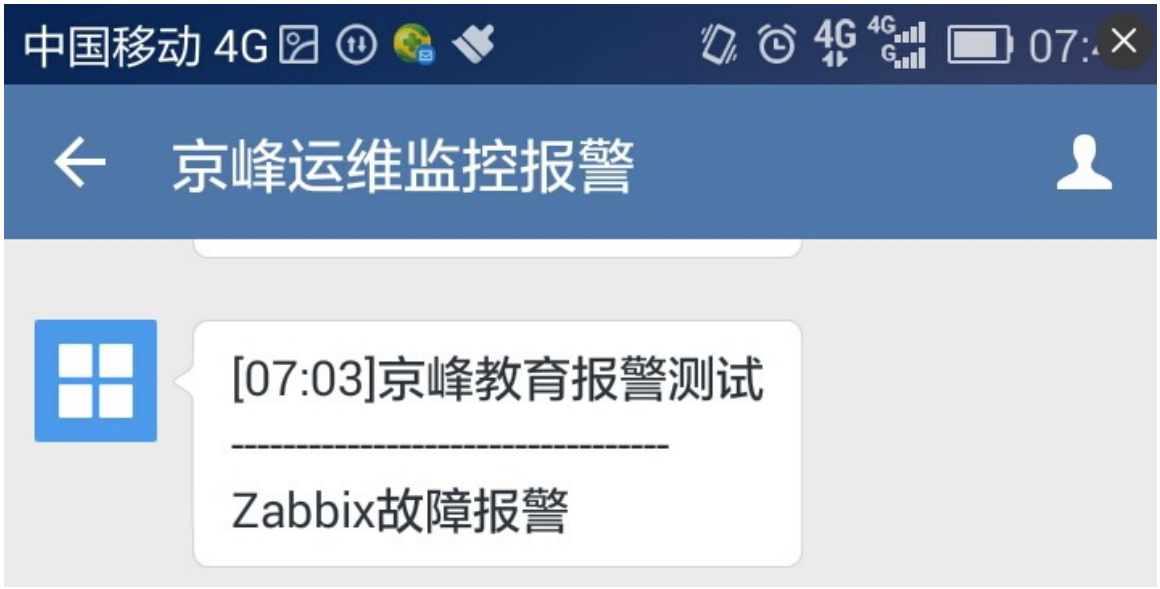


图 13-44（b） Zabbix Server 端微信配置文件

（7） 脚本调用设置

Zabbix_Server 端设置脚本执行路径，编辑 zabbix_server.conf 文件，添加如下内容：

```
AlertScriptsPath=/usr/local/zabbix/alertscripts
```

（8） Zabbix WEB 端配置，设置 Actions 动作，并设置触发微信报警，如图 13-45（a）、13-45（b）、13-45（c）所示：

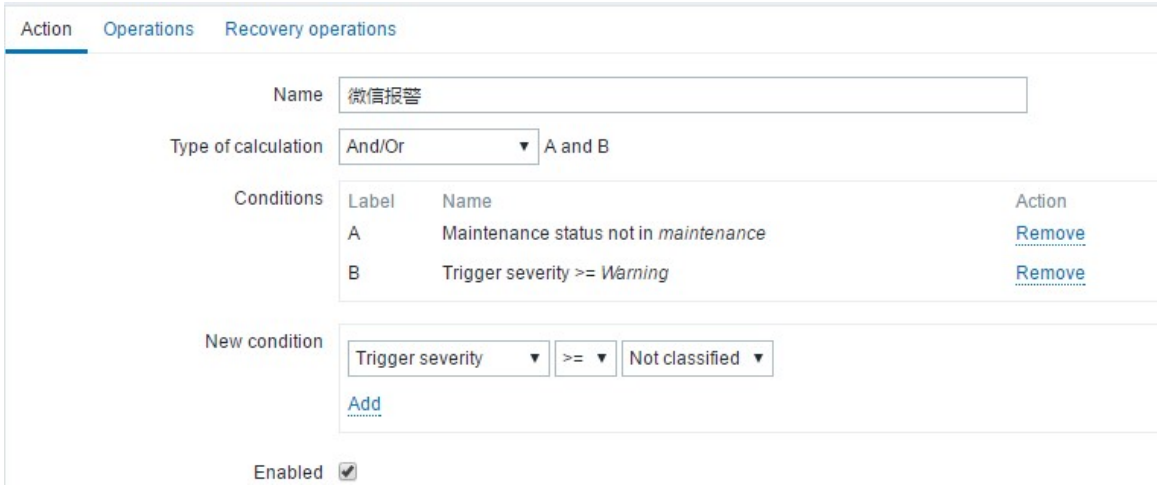


图 13-45（a） Zabbix Server Action 动作配置

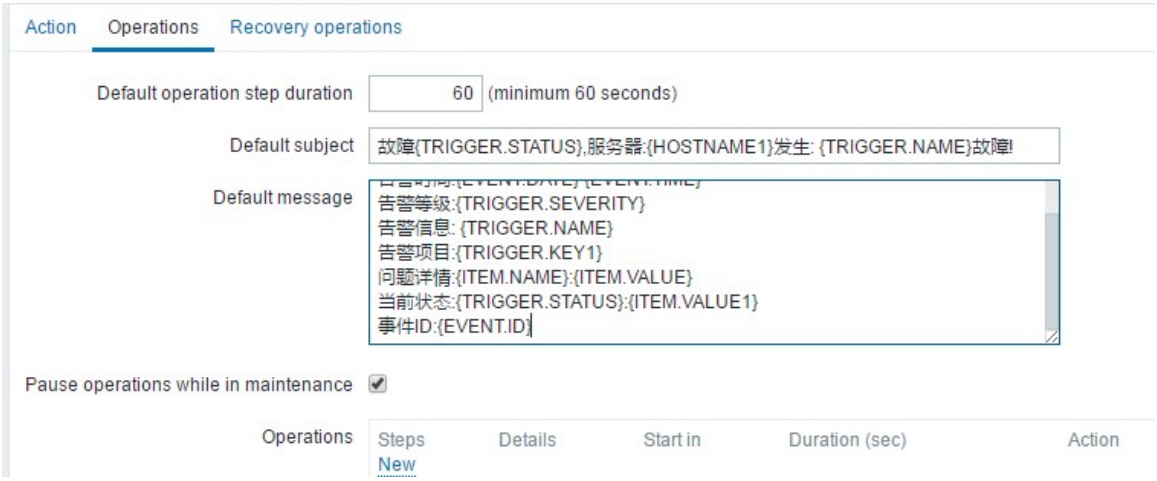


图 13-45（b） Zabbix Server Action 动作配置

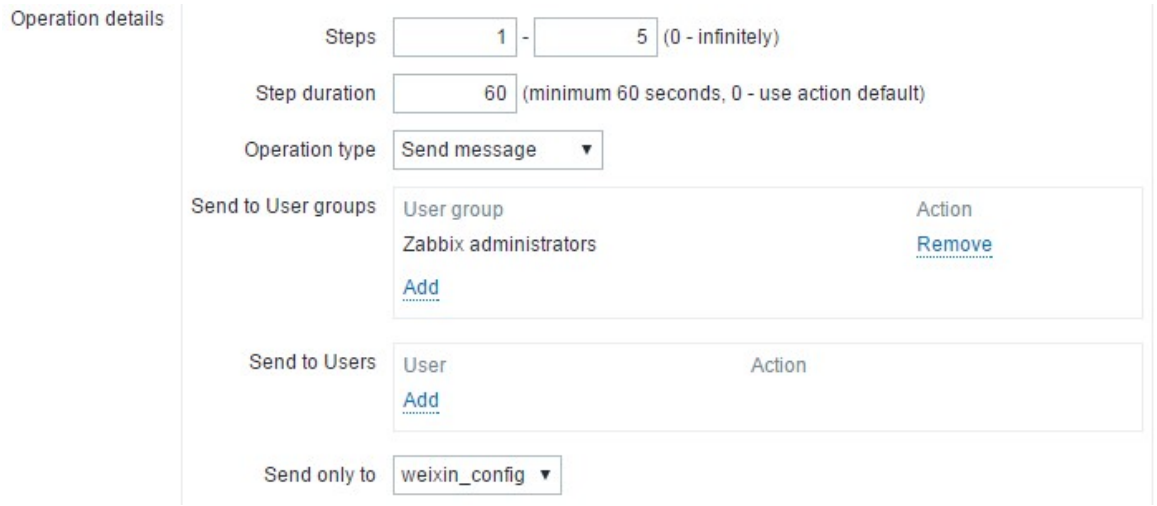


图 13-45（c） Zabbix Server Action 动作配置

（9） 配置 Media Type 微信脚本，Administration→Media Types→Create Media Type 如图 13-46 所示，脚本加入三个参数：{ALERT.SENDTO}、{ALERT.SUBJECT}、{ALERT.MESSAGE}：

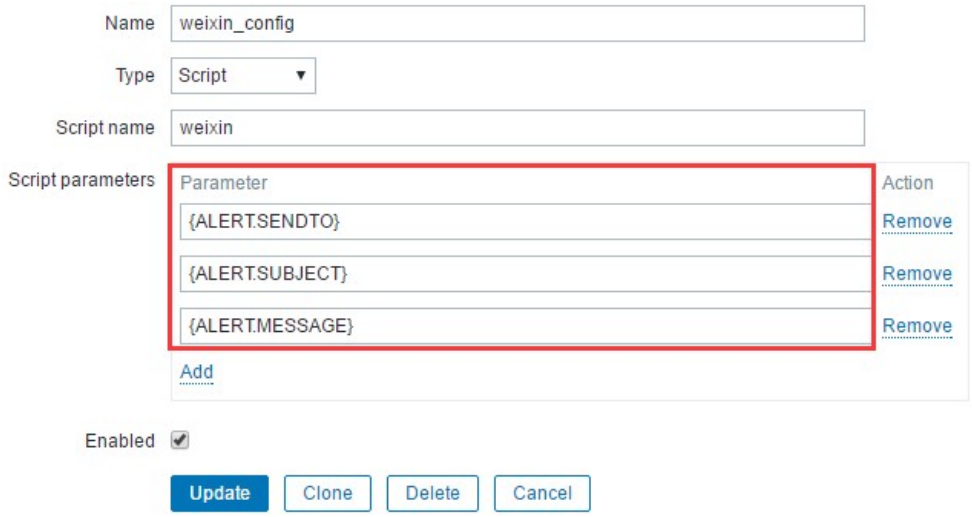


图 13-46 Zabbix Server Media Types 配置

（10） 配置接收微信信息的用户，Administration→Users→Admin→Media 如图 13-47 所示：

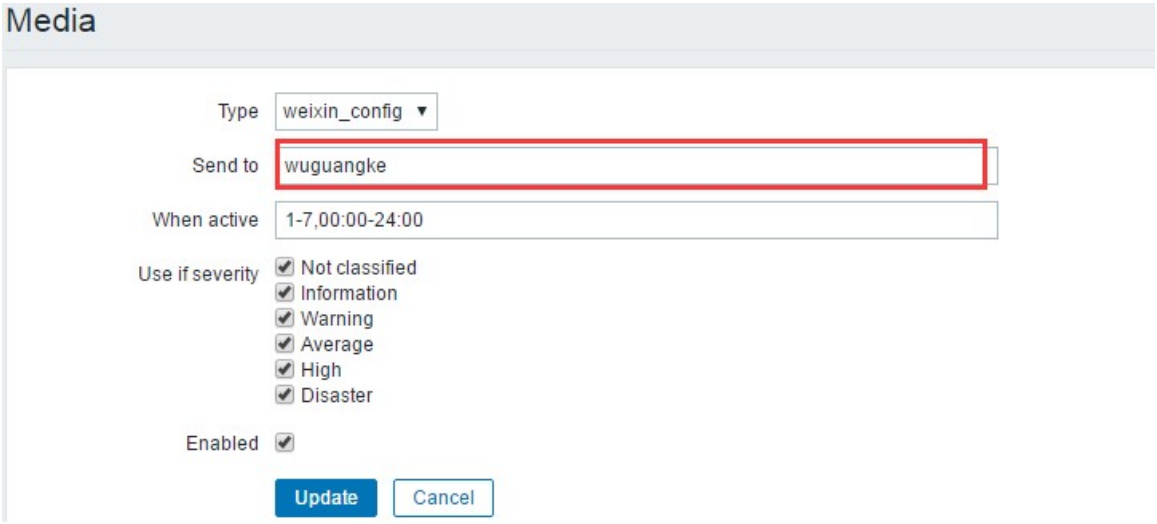


图 13-47 Zabbix Server Users Media

（11） 微信报警信息测试，磁盘容量剩余不足 20%，会触发微信报警，如图 13-48（a）、13-48（b）、13-48（c）所示：

Message actions

Step	Time	Type	Status	Retries left	Recipient	Message
Problem						
1	05/23/2017 07:23:36 AM	weixin_config	Sent		Admin (Zabbix Administrator) wuguangke	**故障PROBLEM,服务器:192.168.149.129发生: Free less than 20% on volume /boot故障!** 告警主机:192.168.149.129 告警时间:2017.05.23 07:23:34 告警等级:Warning 告警信息: Free disk space is less than 20% on volu 告警项目:vfs.fs.size[/boot,pfree] 问题详情:Free disk space on /boot (percentage):0 % 当前状态:PROBLEM:0 % 事件ID:2664

图 13-48（a）Zabbix 微信报警信息

图 13-48（b）Zabbix 微信报警故障信息

图 13-48（c）Zabbix 微信报警恢复信息

13.14 Zabbix 监控网站关键词

随着公司网站系统越来越多,不能通过人工每天手动去刷新网站来检查网站代码及页面是否被传该,通过 zabbix 监控可以实现自动去检查 WEB 网站是否被串改,例如监控某个客户端网站页面中关键词“ATM”是否被修改，通过脚本监控的方法如下：

（1） Agent 端编写 Shell 脚本监控网站关键词,/data/sh/目录 Shell 脚本内容如下，如图 13-49 所示:

```
#!/bin/bash
#2017 年 5 月 24 日 09:49:48
#by author jfedu.net
####################
WEBSITE="http://192.168.149.131/"
NUM=`curl -s $WEBSITE|grep -c "ATM"`
echo $NUM
```

```
[root@localhost sh]# cat check_http_word.sh
#!/bin/bash
#2017年5月24日09:49:48
#by author jfedu.net
####################
WEBSITE="http://192.168.149.131/"
NUM=`curl -s $WEBSITE|grep -c "ATM"`
echo $NUM
[root@localhost sh]#
[root@localhost sh]#
[root@localhost sh]# sh check_http_word.sh
1
[root@localhost sh]#
[root@localhost sh]#
```

图 13-49 Zabbix 客户端脚本内容

（2） 客户端 Zabbix_agentd.conf 内容中加入如下代码，并重启 Agentd 服务即可，如图 13-50 所示：

```
UserParameter=check_http_word,sh /data/sh/check_http_word.sh
```

```
[root@localhost sh]# cat /usr/local/zabbix/etc/zabbix_agentd.conf
LogFile=/tmp/zabbix_agentd.log
Server=192.168.149.128
ServerActive=192.168.149.128
Hostname = 192.168.149.131
UserParameter=check_http_word,sh /data/sh/check_http_word.sh
[root@localhost sh]#
[root@localhost sh]# pwd
/data/sh
[root@localhost sh]#
```

图 13-50 Zabbix 客户端脚本执行结果

（3） 服务器端获取客户端的关键词 KEY，输入 1，则表示 ATM 关键词存在，如果不为 1 则表示 ATM 关键词被串改。

```
/usr/local/zabbix/bin/zabbix_get -s 192.168.149.131 -k check_http_word
```

（4） Zabbix WEB 端添加客户端的 items 监控项，如图 13-51 所示：

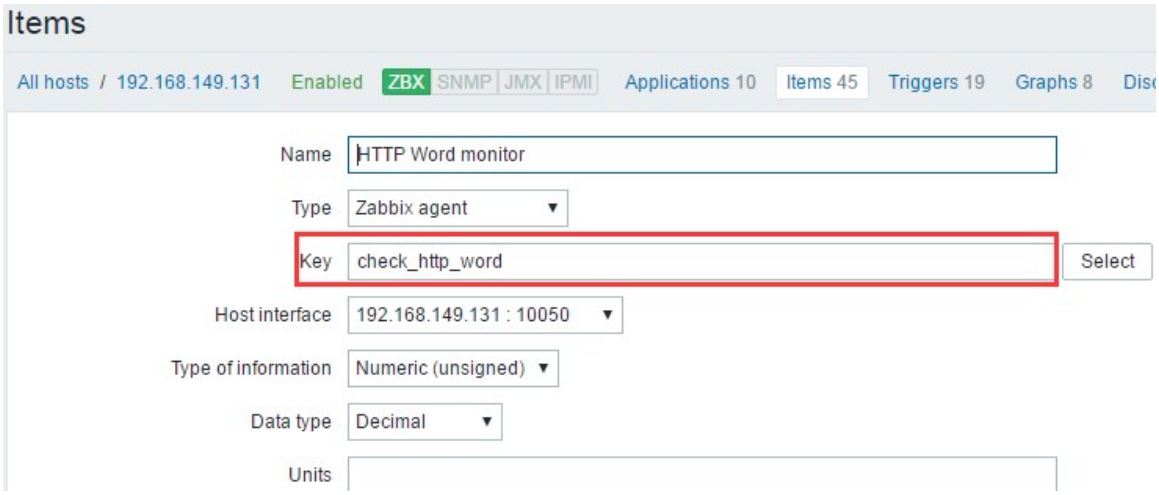


图 13-51 Zabbix 客户端 KEY 添加

（5） 创建 check_http_word 监控 Graphs 图像，如图 13-52（a）、13-52（b）所示：

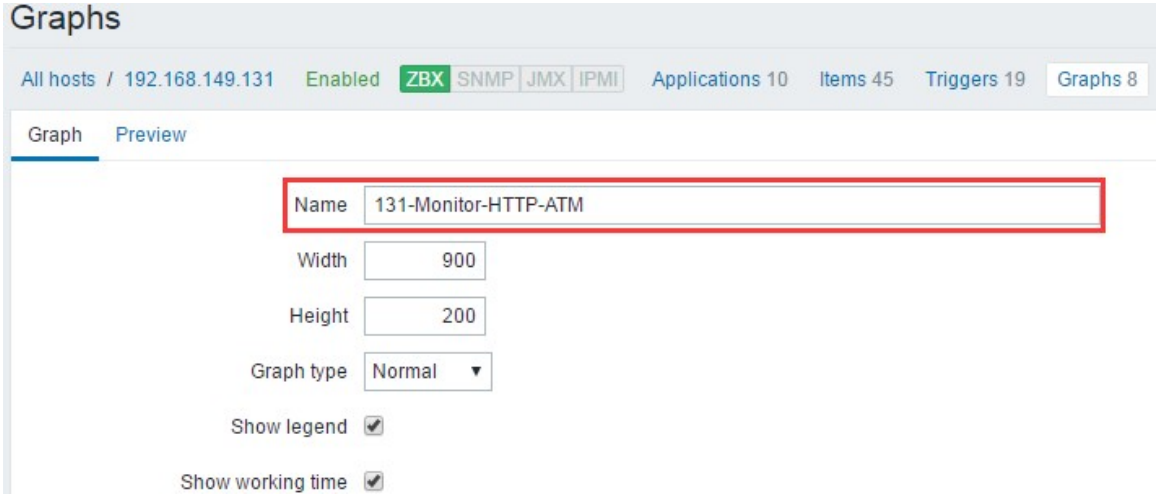


图 13-52（a） Zabbix 客户端添加 Graphs

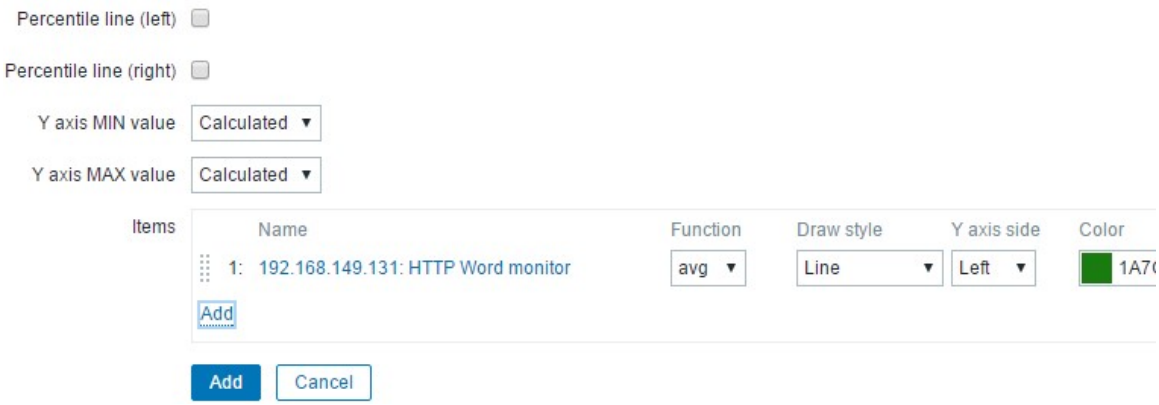


图 13-52（b） Zabbix 客户端添加 Graphs

（6） 创建 check_http_word 触发器，如图 13-53（a）、13-53（b）所示：

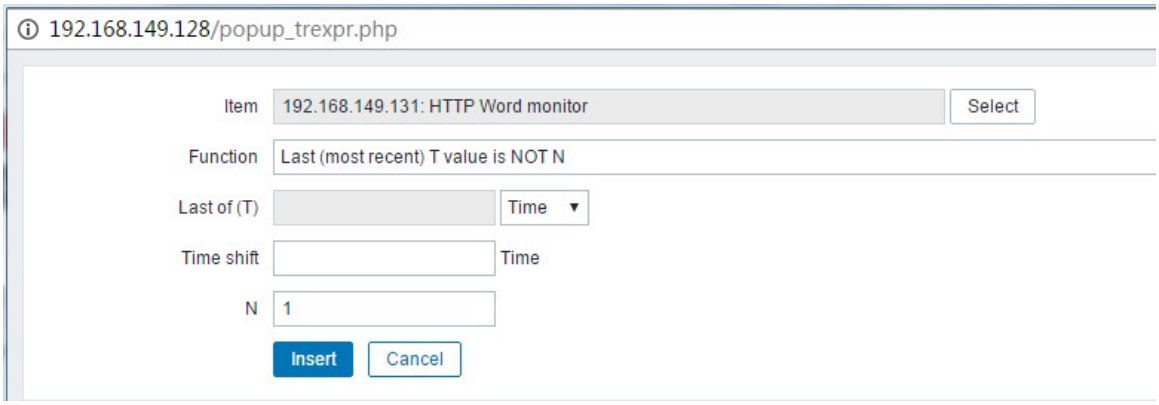


图 13-53（a） Zabbix 客户端创建触发器

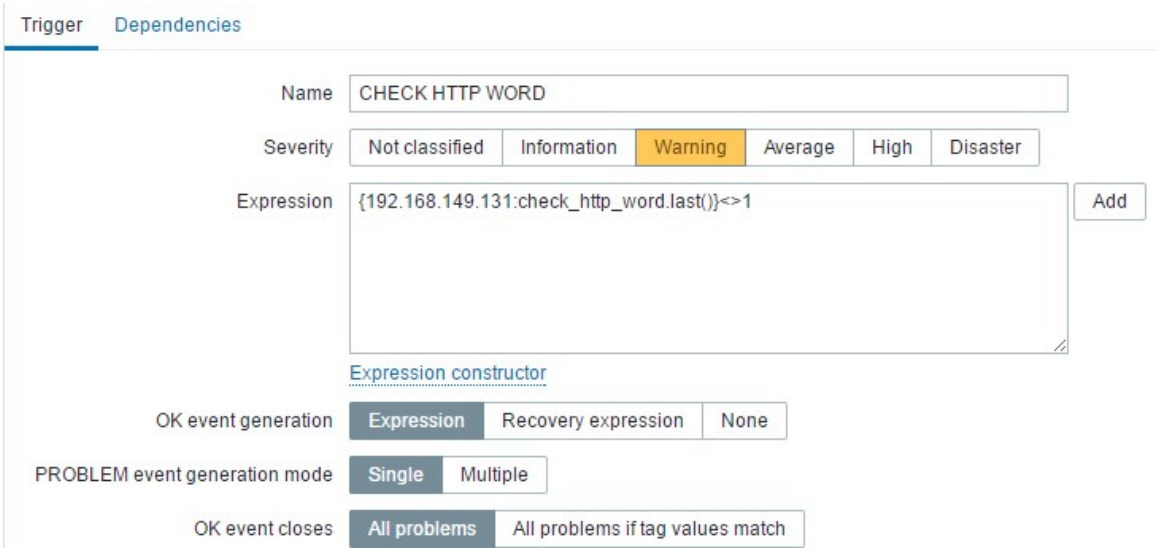


图 13-53（b） Zabbix 客户端创建触发器

（7） 查看 Zabbix 客户端监控图像，如图 13-54（a）、13-54（b）所示：

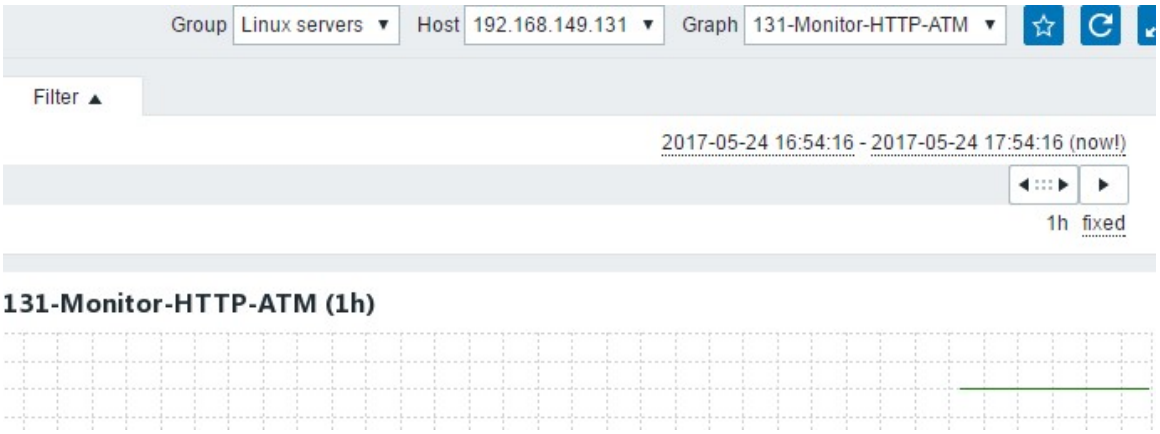


图 13-54（a） Zabbix Http word monitor 监控图

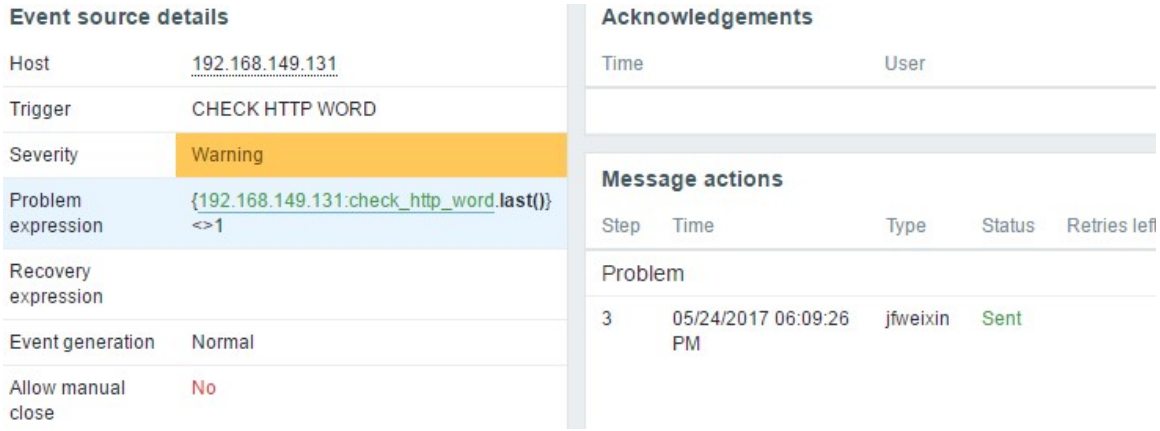


图 13-55（b） Zabbix Http word monitor 触发器微信报警

除了使用如上 Shell 脚本方法，还可以通过 Zabbix WEB 界面配置 Http URL 监控，方

法如下：

（1） Configuration→Hosts→WEB，创建 WEB 监控场景，基于 Chrome38.0 访问 HTTP WEB 页面，如图 13-55（a）、13-55（b）、13-55（c）、13-55（d）、13-55（e）所示：

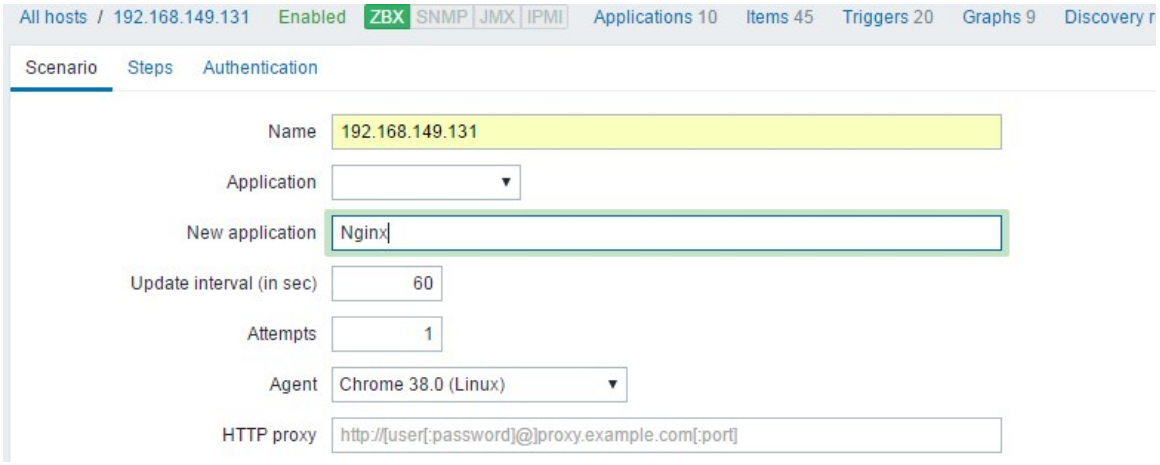


图 13-55（a）Zabbix WEB 场景配置

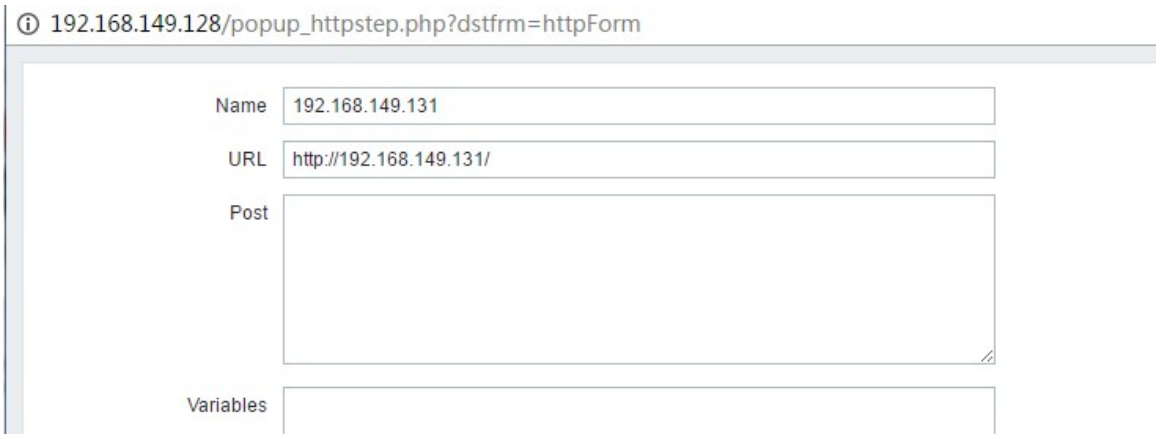


图 13-55（b）Zabbix WEB 场景配置

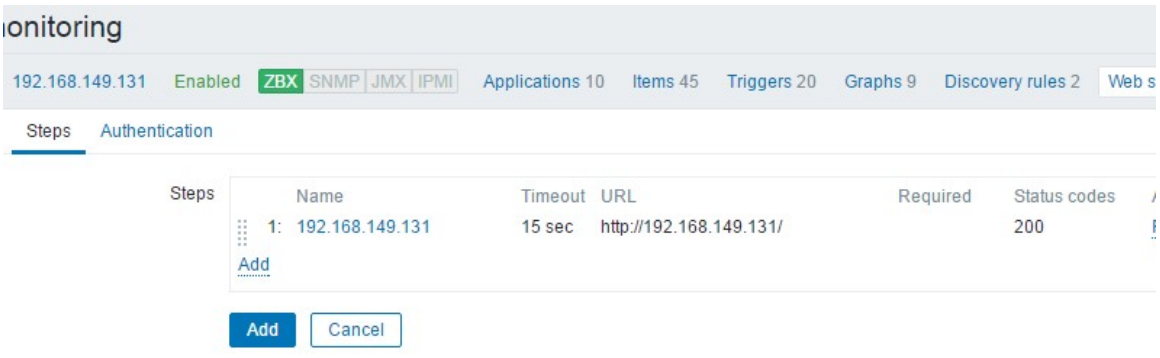


图 13-55（c）Zabbix WEB 场景配置

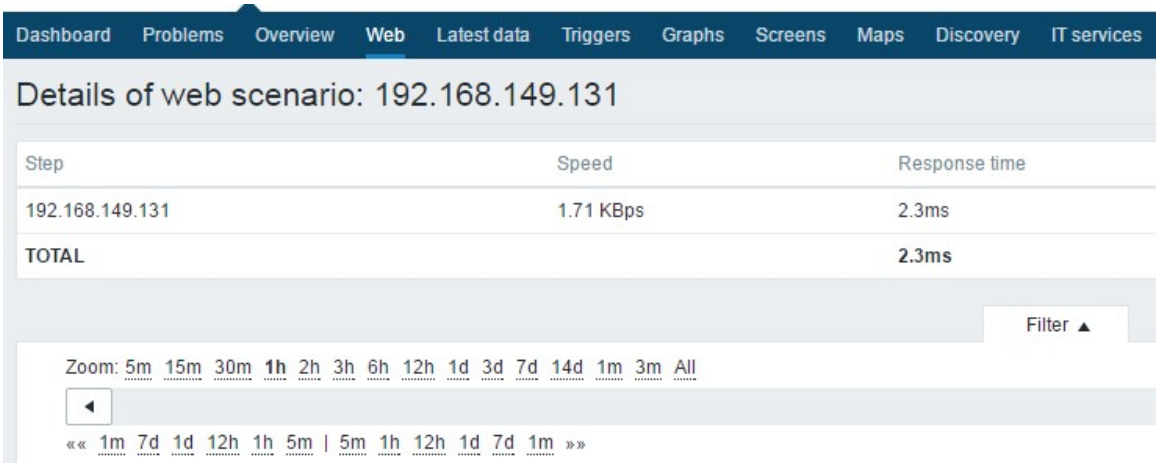


图 13-55（d）Zabbix WEB 监控图

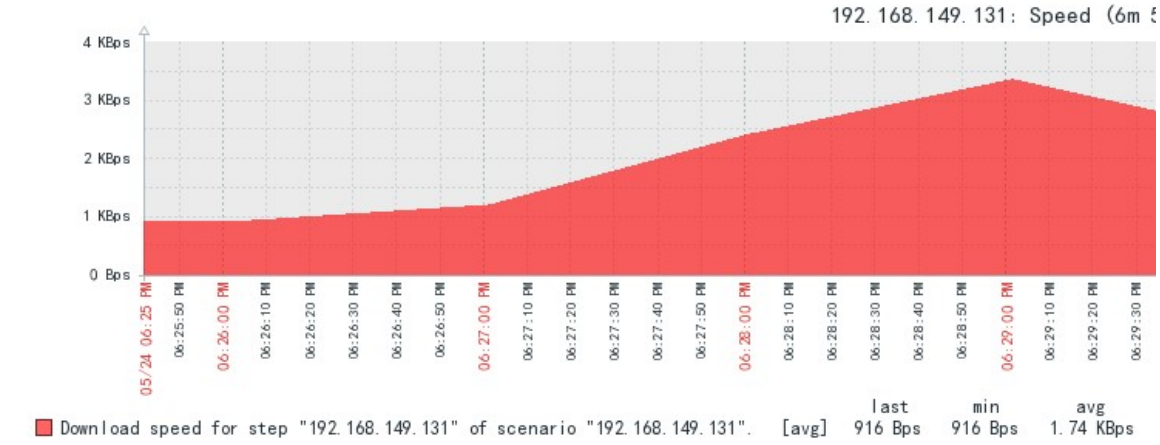


图 13-55（e）Zabbix WEB 监控图

第14章 CentOS7 下 Kickstart 实战篇

14.1 Kickstart 使用背景介绍

随着公司业务不断增加，经常需要采购新服务器，并要求安装 Linux 系统，并且要求 Linux 版本要一致，方便以后的维护和管理，每次人工安装 linux 系统会浪费掉更多时间，如果我们有办法能节省一次一次的时间岂不更好呢?

大中型互联网公司一次采购服务器上百台,如果采用人工手动一台一台的安装,一个人得搞坏 N 张光盘,得多少个加班加点才能完成这项”艰巨”的任务呢,看到全人工来完成这样的

工作太浪费人力了,有没有自动化安装平台呢,通过一台已存在的系统然后克隆或者复制到新的服务器呢。Kickstart 可以毫不费力的完成这项工作。

PXE(preboot execute environment，预启动执行环境)是由 Intel 公司开发的最新技术，工作于 Client/Server 的网络模式，支持工作站通过网络从远端服务器下载映像，并由此支持通过网络启动操作系统，在启动过程中，终端要求服务器分配 IP 地址，再用 TFTP（trivial file transfer protocol）协议下载一个启动软件包到本机内存中执行。

要使用 Kickstart 安装平台，包括的完整架构为：

Kickstart+DHCP+NFS(HTTP)+TFTP+PXE，从架构可以看出，大致需要安装的服务，例如 dhcp、tftp、httpd、kickstart/pxe 等。

14.2 Kickstart 企业实战配置

基于 YUM 安装 DHCP、TFTP、HTTPD 服务，指令如下：

yum install httpd httpd-devel tftp-server xinetd dhcp* -y

配置 tftp 服务，开启 tftp 服务；

```
cat>/etc/xinetd.d/tftp<<EOF
service tftp
{
disable = no
socket_type = dgram
protocol = udp
wait = yes
user = root
```

```
server = /usr/sbin/in.tftpd
server_args = -u nobody -s /tftpboot
per_source = 11
cps = 100 2
flags = IPv4
}
EOF
```

只需要把 disable = yes 改成 disable = no 即可，基于 sed 命令也可以实现：

sed -i '/disable/s/yes/no/g' /etc/xinetd.d/tftp

14.3 TFTP+PXE 配置实战

要实现远程安装系统，需要在 TFTPBOOT 目录指定相关 PXE 内核模块及相关参数，配置步骤如下：

#挂载本地光盘

mount /dev/cdrom /mnt

#安装 syslinux 必备文件

yum install syslinux syslinux-devel -y

#软链接至/根系统下；

```
ln -s /var/lib/tftpboot /
mkdir -p /var/lib/tftpboot/pxelinux.cfg/
\cp /mnt/isolinux/isolinux.cfg /var/lib/tftpboot/pxelinux.cfg/default
\cp /usr/share/syslinux/vesamenu.c32 /var/lib/tftpboot/
```

```
\cp /mnt/images/pxeboot/vmlinuz /var/lib/tftpboot/
\cp /mnt/images/pxeboot/initrd.img /var/lib/tftpboot/
\cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot/
chmod 644 /var/lib/tftpboot/pxelinux.cfg/default
```

14.4 配置 TFTPBOOT 引导案例

```
cat>/tftpboot/pxelinux.cfg/default<<EOF
default vesamenu.c32
timeout 10
display boot.msg
menu clear
menu background splash.png
menu title CentOS Linux 7
label linux
  menu label ^Install CentOS Linux 7
  menu default
  kernel vmlinuz
  append  initrd=initrd.img  inst.repo=http://192.168.0.131/centos7
quiet ks=http://192.168.0.131/ks.cfg
label check
  menu label Test this ^media & install CentOS Linux 7
  kernel vmlinuz
```

```
  append                                                initrd=initrd.img
inst.stage2=hd:LABEL=CentOS\x207\x20x86_64 rd.live.check quiet
EOF
```

配置文件详解：

192.168.0.131 是 kickstart 服务器，/centos7 是 HTTPD 共享 linux 镜像的目录，即 linux 存放安装文件的路径：

ks.cfg 是 kickstart 主配置文件;

设置 timeout 10 /*超时时间为 10S */;

ksdevice=ens33 代表当我们有多块网卡的时候，要实现自动化需要设置从 ens33 安装。

TFTP 配置完毕，由于是 TFTP 是非独立服务，需要依赖 xinetd 服务来启动，启动命令为：

chkconfig tftp --level 35 on && service xinetd restart

14.5 Apache+Kickstart 配置实战

远程系统安装，客户端需要下载系统所需的软件包，所以需要使用 NFS 或者 httpd 把镜像文件共享出来。

mkdir -p /var/www/html/centos7/

mount /dev/cdrom /var/www/html/centos7/

#cp /dev/cdrom/* /var/www/html/centos7/ （可选配置）

配置 kickstart，可以使用 system-kickstart 系统软件包来配置，ks.cfg 配置文件内容如下：

```
cat>/var/www/html/ks.cfg<<EOF
install
text
keyboard 'us'
rootpw www.jfedu.net
timezone Asia/Shanghai
url --url=http://192.168.0.131/centos7
reboot
lang zh_CN
firewall --disabled
network   --bootproto=dhcp --device=ens33
auth   --useshadow   --passalgo=sha512
firstboot --disable
selinux    disabled
bootloader --location=mbr
clearpart --all --initlabel
part /boot --fstype="ext4" --size=300
part / --fstype="ext4" --grow
part swap --fstype="swap" --size=512
%packages
@base
@core
```

```
%end

EOF
```

14.6 DHCP 服务器配置实战

DHCP 服务配置文件代码如下：

```
cat>/etc/dhcp/dhcpd.conf<<EOF

ddns-update-style interim;

ignore client-updates;

next-server 192.168.0.131;

filename "pxelinux.0";

allow booting;

allow bootp;

subnet 192.168.0.0 netmask 255.255.255.0 {

#default gateway

option routers                  192.168.0.1;

option subnet-mask          255.255.255.0;

range dynamic-bootp 192.168.0.180 192.168.0.200;

host ns {

hardware ethernet    00:1a:a0:2b:38:81;

fixed-address 192.168.0.101;}

}

EOF
```

重启各个服务，启动新的客户端验证测试：

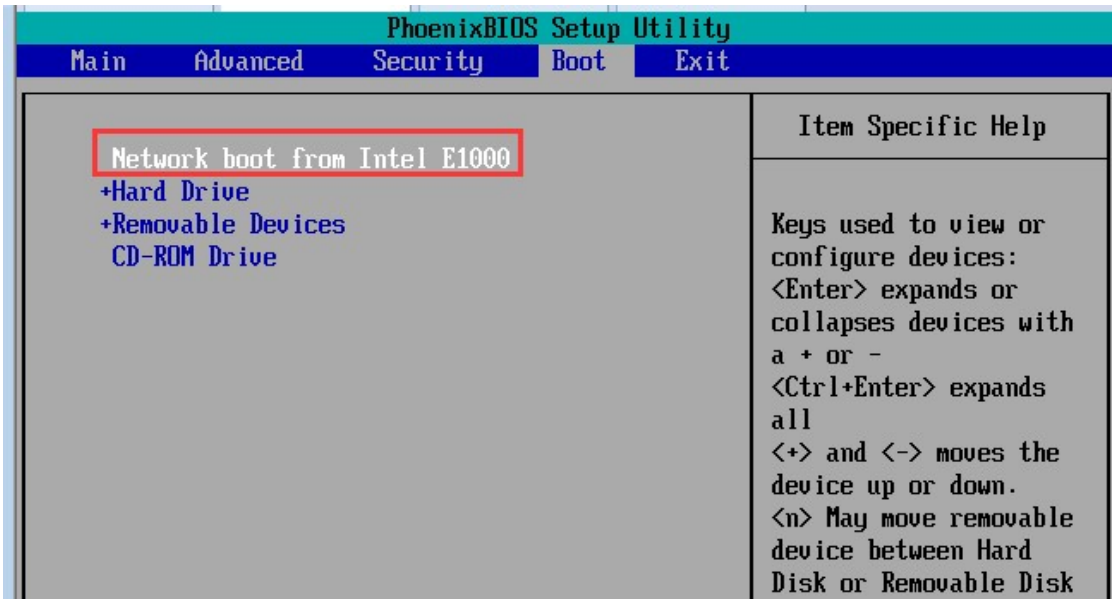
service httpd restart

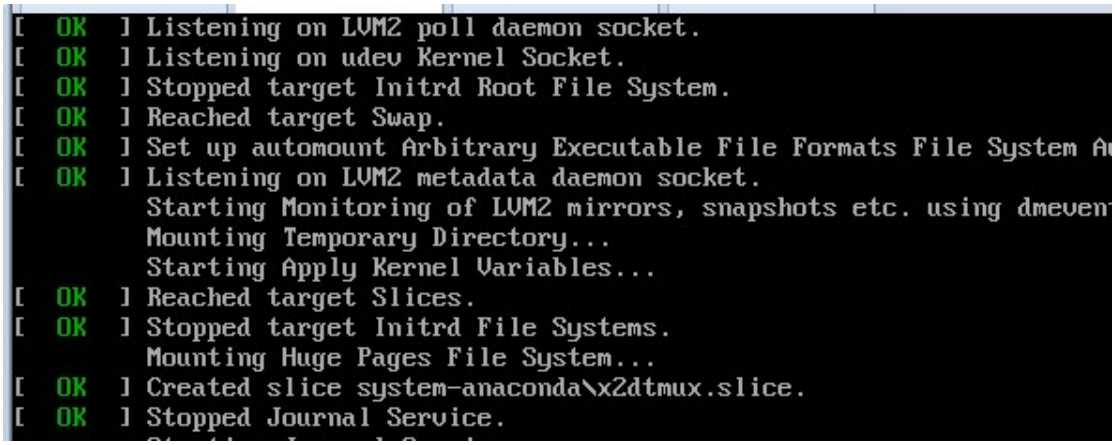
service dhcpd restart

service xinetd restart

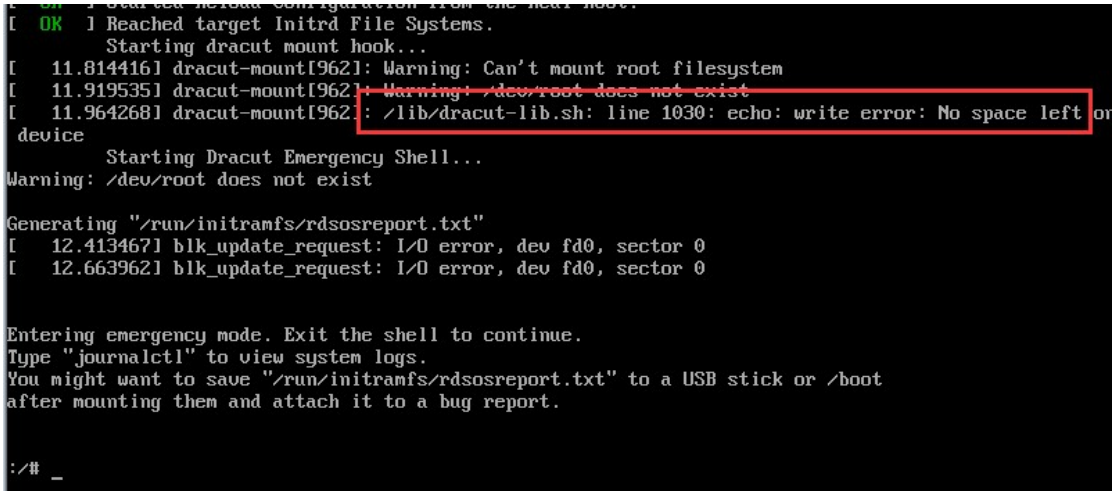
```
[root@Jfedu-Server-131 ~]#
[root@Jfedu-Server-131 ~]# service httpd restart
service dhcpd restart
service xinetd restartRedirecting to /bin/systemctl restart  httpd.service
[root@Jfedu-Server-131 ~]# service dhcpd restart
Redirecting to /bin/systemctl restart  dhcpd.service
[root@Jfedu-Server-131 ~]# service xinetd restart
Redirecting to /bin/systemctl restart  xinetd.service
[root@Jfedu-Server-131 ~]#
[root@Jfedu-Server-131 ~]# ps -ef|grep -E "httpd|dhcpd|xinetd"
root      16583     1  0 18:08 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    16586 16583  0 18:08 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    16587 16583  0 18:08 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    16588 16583  0 18:08 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    16589 16583  0 18:08 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    16590 16583  0 18:08 ?        00:00:00 /usr/sbin/httpd -DFOREGROUND
dhcpd     16609     1  0 18:08 ?        00:00:00 /usr/sbin/dhcpd -f -cf /etc/d
 dhcpd -group dhcpd --no-pid
root      16629     1  0 18:08 ?        00:00:00 /usr/sbin/xinetd -stayalive -
etd.pid
root      16631 15348  0 18:09 pts/0    00:00:00 grep --color=auto -E httpd|dh
[root@Jfedu-Server-131 ~]#
```

14.7 Kickstart 客户端测试

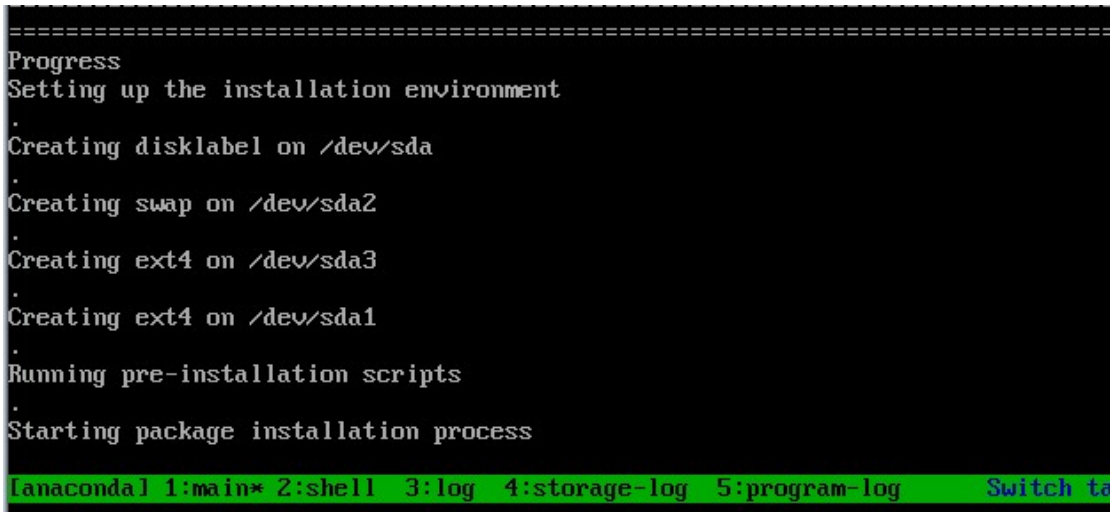




如果安装时报错如下：



需要调整客户端虚拟机的内存设置为 2G+；



14.8 Kickstart 企业生产环境扩展

在真实环境中，通常我们会发现一台服务器好几块硬盘，做完 raid，整个硬盘有等 10T，如果来使用 kickstart 自动安装并分区呢；一般服务器硬盘超过 2T，如何来使用 kickstart 安装配置呢？这里就不能使用 MBR 方式来分区，需要采用 GPT 格式来引导并分区。需要在 ks.cfg 末尾添加如下命令来实现需求：

```
%pre
parted  -s  /dev/sdb  mklabel  gpt
%end
```

为了实现 kickstart 安装完系统后，自动初始化系统等等工作，在系统安装完后，自动执行定制的脚本，需要在 ks.cfg 末尾加入如下配置：

```
%post
mount  -t  nfs 192.168.0.79:/centos/init  /mnt
cd  /mnt/ ;/bin/sh  auto_init.sh
%end
```

KICKSTART 所有配置就此告一段落，真实环境需要注意，新服务器跟 kickstart

最后独立在一个网络，不要跟办公环境或者服务器机房网络混在一起，如果别的机器以网卡就会把它的系统重装成 Linux 系统。

第15章　Linux SHELL 编程基础篇

说到 Shell 编程，很多从事 Linux 运维工作的朋友都不陌生，都对 Shell 有基本的了解，读者可能刚开始接触 Shell 的时候，有各种想法，感觉编程非常困难，SHELL 编程是所有编程语言中最容易上手，最容易学习的编程脚本语言。

本章向读者介绍 Shell 编程入门、Shell 编程变量、If、While、For、Case、Select 基本语句案例演练及 Shell 编程四剑客 Find、Grep、Awk、Sed 深度剖析等。

15.1 SHELL 编程入门简介

曾经有人说过，学习 Linux 不知道 Shell 编程，那就是不懂 Linux，现在细细品味确实是这样。Shell 是操作系统的最外层，Shell 可以合并编程语言以控制进程和文件，以及启动和控制其它程序。

Shell 通过提示您输入，向操作系统解释该输入，然后处理来自操作系统的任何结果输出，简单来说 Shell 就是一个用户跟操作系统之间的一个命令解释器。

Shell 是用户与 Linux 操作系统之间沟通的桥梁，用户可以输入命令执行，又可以利用 Shell 脚本编程去运行，如图 17-1 所示：

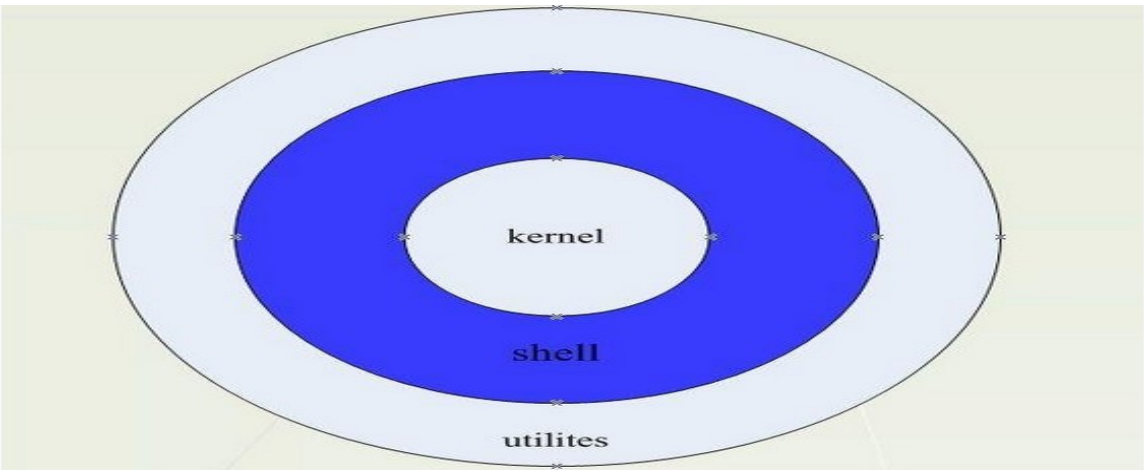
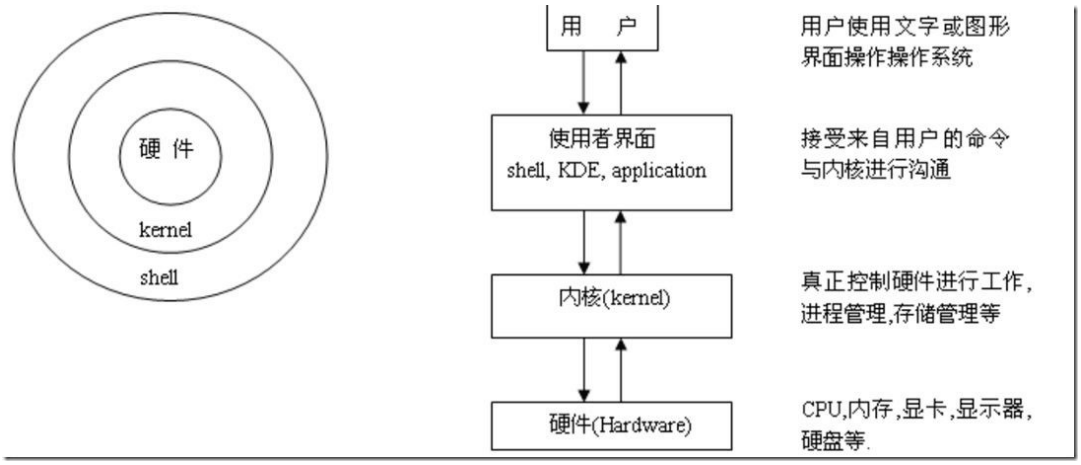


图 17-1 Shell、用户及 Kernel 位置关系



Linux Shell 种类非常多，常见的 SHELL 如下：

- Bourne Shell（/usr/bin/sh 或/bin/sh）
- **Bourne Again Shell（/bin/bash)**
- C Shell（/usr/bin/csh）
- K Shell（/usr/bin/ksh）
- Shell for Root（/sbin/sh）

不同的 Shell 语言的语法有所不同，一般不能交换使用，最常用的 shell 是 Bash，也就是 Bourne Again Shell。Bash 由于易用和免费，在日常工作中被广泛使用，也是大多数 Linux 操作系统默认的 Shell 环境。

Shell、Shell 编程、Shell 脚本、Shell 命令之间都有什么区别呢？简单来说 Shell 是一

个整体的概念，Shell 编程与 Shell 脚本统称为 Shell 编程，Shell 命令是 Shell 编程底层具体的语句和实现方法。

15.2 SHELL 脚本及 Hello World

要熟练掌握 Shell 编程语言，需要大量的练习，初学者可以用 Shell 打印“Hello World”字符，寓意着开始新的启程！

Shell 脚本编程需要如下几个事项：

- Shell 脚本名称命名一般为英文、大写、小写；
- 不能使用特殊符号、空格来命名；
- Shell 脚本后缀以.sh 结尾；
- 不建议 Shell 命名为纯数字，一般以脚本功能命名。
- Shell 脚本内容首行需以#!/bin/bash 开头；
- Shell 脚本中变量名称尽量使用大写字母，字母间不能使用“-”，可以使用“_”；
- Shell 脚本变量名称不能以数字、特殊符号开头。

如下为第一个 Shell 编程脚本，脚本名称为：first_shell.sh，代码内容如下：

```
#!/bin/bash
#This is my First shell
#By author jfedu.net 2017
echo  “Hello  World ”
```

First_shell.sh 脚本内容详解如下：

#!/bin/bash	固定格式，定义该脚本所使用的 Shell 类型；
#This is my First shell	#号表示注释，没有任何的意义，SHELL 不会解析它；

```
#By author jfedu.net 2017 表示脚本创建人，#号表示注解；

echo  “Hello  World !”  Shell 脚本主命令，执行该脚本呈现的内容。
```

Shell 脚本编写完毕，如果运行该脚本，运行用户需要有执行权限，可以使用 chmod o+x first_shell.sh 赋予可执行权限。然后./first_shell.sh 执行即可，还可以直接使用命令执行: /bin/sh first_shell.sh 直接运行脚本，不需要执行权限，最终脚本执行显示效果一样。

初学者学习 Shell 编程，可以将在 Shell 终端运行的各种命令依次写入到脚本内容中，可以把 Shell 脚本当成是 Shell 命令的堆积。

15.3 Shell 编程之变量详解

Shell 是非类型的解释型语言，不像 C++、JAVA 语言编程时需要事先声明变量，Shell 给一个变量赋值，实际上就是定义了变量，在 Linux 支持的所有 shell 中，都可以用赋值符号(=)为变量赋值，Shell 变量为弱类型，定义变量不需要声明类型，但在使用时需要明确变量的类型，可以使用 Declare 指定类型，Declare 常见参数有：

```
+/-  "-"可用来指定变量的属性，"+"为取消变量所设的属性；

-f   仅显示函数；

r    将变量设置为只读；

x    指定的变量会成为环境变量，可供 shell 以外的程序来使用；

i    指定类型为数值，字符串或运算式。
```

Shell 编程中变量分为三种，分别是系统变量、环境变量和用户变量，Shell 变量名在定义时，首个字符必须为字母（a-z，A-Z），不能以数字开头，中间不能有空格，可以使用下划线（_），不能使用（-），也不能使用标点符号等。

例如定义变量 A=jfedu.net，定义这样一个变量，A 为变量名，jfedu.net 是变量的值，变量名有格式规范，变量的值可以随意指定。变量定义完成，如需要引用变量，可以使用

$A。

如下脚本 var.sh 脚本内容如下：

```
#!/bin/bash

#By author jfedu.net 2017

A=123

echo    “Printf variables is $A.”
```

执行该 Shell 脚本，结果将会显示：Printf variables is jfedu.net。

15.4 Shell 编程之系统变量

Shell 常见的变量之一系统变量，主要是用于对参数判断和命令返回值判断时使用，系统变量详解如下：

$0	当前脚本的名称；
$n	当前脚本的第 n 个参数,n=1,2,…9；
$*	当前脚本的所有参数(不包括程序本身)；
$#	当前脚本的参数个数(不包括程序本身)；
$?	命令或程序执行完后的状态，返回 0 表示执行成功；
$$	程序本身的 PID 号。

15.5 Shell 编程之环境变量

Shell 常见的变量之二环境变量，主要是在程序运行时需要设置，环境变量详解如下：

PATH	命令所示路径，以冒号为分割；
HOME	打印用户家目录；
SHELL	显示当前 Shell 类型；

USER	打印当前用户名；
ID	打印当前用户 id 信息；
PWD	显示当前所在路径；
TERM	打印当前终端类型；
HOSTNAME	显示当前主机名。

15.6 Shell 编程之用户变量

Shell 常见的变量之三用户变量，用户变量又称为局部变量，主要用在 Shell 脚本内部或者临时局部使用，系统变量详解如下：

A=jfedu.net	自定义变量 A；
N_SOFT=nginx-1.12.0.tar.gz	自定义变量 N_SOFT；
BACK_DIR=/data/backup/	自定义变量 BACK_DIR；
IP1=192.168.1.11	自定义变量 IP1；
IP2=192.168.1.12	自定义变量 IP2。

创建 Echo 打印菜单 Shell 脚本，脚本代码如下：

```
#!/bin/bash

#auto install httpd

#By author jfedu.net 2017

echo -e '\033[32m----------------------------\033[0m'

FILE=httpd-2.2.31.tar.bz2

URL=http://mirrors.cnnic.cn/apache/httpd/

PREFIX=/usr/local/apache2/
```

```
echo -e "\033[36mPlease Select Install Menu:\033[0m"

echo

echo "1)官方下载 Httpd 文件包."

echo "2)解压 apache 源码包."

echo "3)编译安装 Httpd 服务器."

echo "4)启动 HTTPD 服务器."

echo -e '\033[32m----------------------------\033[0m'

sleep 20
```

运行脚本，执行结果如图 17-2 所示：

图 17-2 Echo 打印菜单脚本

15.7 If 条件语句实战

Linux Shell 编程中，if、for、while、case 等条件流程控制语句用的非常多，熟练掌握以上流程控制语句及语法的实验，对编写 Shel 脚本有非常大的益处。

If 条件判断语句，通常以 if 开头，fi 结尾。也可加入 else 或者 elif 进行多条件的判断，if 表达式如下：

```
if  (表达式)

    语句 1
```

```
else
    语句 2
fi
```

If 语句 Shell 脚本编程案例如下：

（1） 比较两个整数大小。

```
#!/bin/bash
#By author jfedu.net 2017
NUM=100
if (( $NUM > 4 )) ; then
    echo "The Num $NUM more than 4."
else
    echo "The Num $NUM less than 4."
fi
```

（2） 判断系统目录是否存在。

```
#!/bin/bash
#judge DIR or Files
#By author jfedu.net 2017
if [ ! -d /data/20140515 -a ! -d /tmp/2017/ ]; then
        mkdir -p /data/20140515
fi
```

If 常见判断逻辑运算符详解：

```
-f                    判断文件是否存在  eg: if [ -f filename ];
```

-d	判断目录是否存在 eg: if [-d dir];
-eq	等于，应用于整型比较 equal;
-ne	不等于，应用于整型比较 not equal;
-lt	小于，应用于整型比较 letter;
-gt	大于，应用于整型比较 greater;
-le	小于或等于，应用于整型比较;
-ge	大于或等于，应用于整型比较;
-a	双方都成立（and） 逻辑表达式 –a 逻辑表达式;
-o	单方成立（or） 逻辑表达式 –o 逻辑表达式;
-z	空字符串;
||	单方成立;
&&	双方都成立表达式。

（3） if 多个条件测试分数判断。

```
#!/bin/bash
#By author jfedu.net 2017
scores=$1
if  [[ $scores -eq 100 ]];  then
    echo "very good!";
elif [[ $scores -gt 85 ]];  then
    echo "good!";
elif [[ $scores -gt 60 ]];  then
    echo "pass!";
```

```
elif [[ $scores -lt 60 ]];    then
    echo "no pass!"
fi
```

15.8 SHELL 编程括号详解

Shell 编程中，尤其是使用 if 语句时，经常会使用()、(())、[]、[[]]、{}等括号，如下为几种括号简单区别对比：

```
(  )
用于多个命令组、命令替换、初始化数组，多用于 SHELL 命令组，例如：JF=(jf1 jf2 jf3)，其中括号左右不保留空格；
(( ))
整数扩展、运算符、重定义变量值，算术运算比较,例如：((i++))、((i<=100))，其中括号左右不保留空格；
[ ]
bash 内部命令，[ ]与 test 是等同的，正则字符范围、引用数组元素编号，不支持+-*/数学运算符，逻辑测试使用-a、-o，通常用于字符串比较、整数比较以及数组索引，其中括号左右要保留空格；
[[ ]]
bash 程序语言的关键字，不是一个命令，[[ ]]结构比[ ]结构更加通用，不支持+-*/数学运算符，逻辑测试使用&&、||，通常用于字符串比较、逻辑运算符等，其中括号左右要保留空格；
{}
```

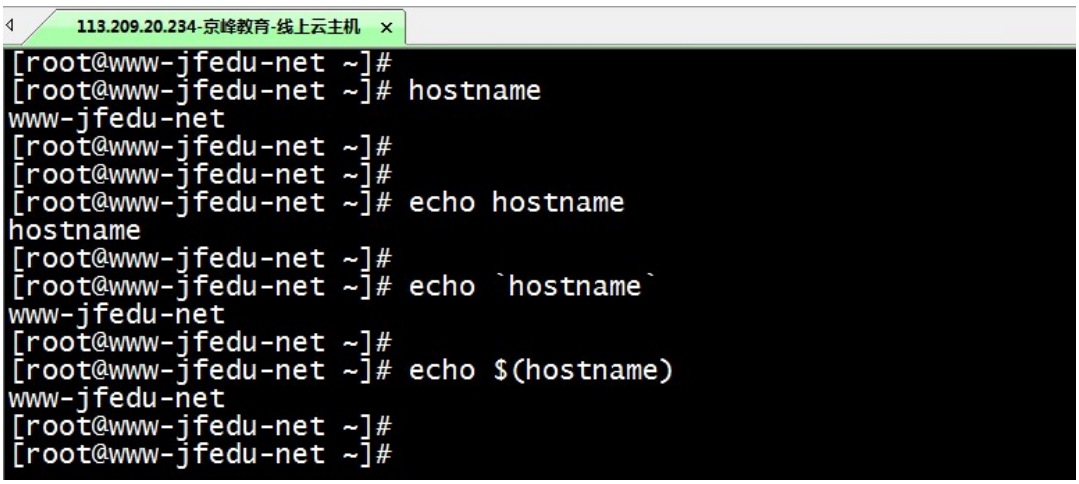
主要用于命令集合或者范围，例如 mkdir -p /data/201{7,8}/，其中括号左右不保留空格；

15.9 SHELL 编程符号详解

Shell 编程中，不管是使用变量、编程时，经常会使用$、\、单引号、双引号、反引号等符号，如下为几种符号简单区别对比：

- 美元符号$，主要是用于引用 SHELL 编程中变量，例如定义变量 JF=www.jfedu.net，引用值，需要用$JF;
- \反斜杠，主要是用于对特定的字符实现转义，保留原有意义，例如 echo “\$JF”结果会打印$JF,而不会打印 www.jfedu.net；

- 单引号（' '），单引号，不具有变量置换的功能，所有的任意字符还原为字面意义，实现屏蔽 Shell 元字符的功能；
- 双引号(" ")，双引号，具有变量置换的功能，保留$（使用变量前导符），\(转义符)，`(反向引号)元字符的功能；
- 反向引号(` `)，反引号，位于键盘 Tab 键上面一行的键，用作命令替换（相当于$(...)）；



15.10 MySQL 数据库备份脚本

MySQL 数据库备份是运维工程师的工作之一，如下为自动备份 Mysql 数据库脚本。

```
#!/bin/bash

#auto backup mysql

#By author jfedu.net 2017

#Define PATH 定义变量

BAK_DIR=/data/backup/mysql/`date +%Y-%m-%d`

MYSQLDB=webapp

MYSQLPW=backup

MYSQLUSR=backup

#must use root user run scripts 必须使用 root 用户运行，$UID 为系统变量

if

    [ $UID -ne 0 ]；then

    echo This script must use the root user ! ! !
```

```
    sleep 2
    exit 0
fi
#Define DIR and mkdir DIR 判断目录是否存在，不存在则新建
if
    [ !  -d  $BAKDIR ]; then
    mkdir  -p  $BAKDIR
fi
#Use mysqldump backup Databases
/usr/bin/mysqldump          -u$MYSQLUSR          -p$MYSQLPW          -d
$MYSQLDB >$BAKDIR/webapp_db.sql
echo  "The mysql backup successfully "
```

15.11 LAMP 一键自动化安装脚本

通过前面章节对 if 语句和变量的学习，现基于所学知识，编写一键源码安装 LAMP 脚本，

编写脚本可以养成先分解脚本的各个功能的习惯，有利于快速写出脚本，写出更高效的脚本。

一键源码安装 LAMP 脚本，可以拆分为如下功能：

（1） LAMP 打印菜单：

- 安装 apache WEB 服务器；
- 安装 Mysql DB 服务器；
- 安装 PHP 服务器；
- 整合 LAMP 架构

- 启动 LAMP 服务；

（2） Apache 服务器安装部署：

Apache 官网下载 httpd-2.2.31.tar.gz 版本，解压，进入安装目录，configure、make 、make install。

（3） Mysql 服务器的安装：

Mysql 官网下载 mysql-5.5.20.tar.gz 版本，解压，进入安装目录，configure、make 、make install。

（4） PHP 服务器安装：

PHP 官网下载 php-5.3.8.tar.gz 版本，解压，进入安装目录，configure、make 、make install。

（5） LAMP 整合及服务启动：

```
vi   /usr/local/apache2/htdocs/index.php
<?php
phpinfo();
?>
/usr/local/apache2/bin/apachectl restart
service mysqld   restart
```

一键源码安装 LAMP 脚本，auto_install_lamp.sh 内容如下：

```
#!/bin/bash
#auto install LAMP
#By author jfedu.net 2017
#Httpd define path variable
```

```
H_FILES=httpd-2.2.31.tar.bz2

H_FILES_DIR=httpd-2.2.31

H_URL=http://mirrors.cnnic.cn/apache/httpd/httpd-2.2.31.tar.bz2

H_PREFIX=/usr/local/apache2/

#MySQL define path variable

M_FILES=mysql-5.5.20.tar.gz

M_FILES_DIR=mysql-5.5.20

M_URL=http://down1.chinaunix.net/distfiles/ mysql-5.5.20.tar.gz

M_PREFIX=/usr/local/mysql/

#PHP define path variable

P_FILES=php-5.3.28.tar.bz2

P_FILES_DIR=php-5.3.28

P_URL=http://mirrors.sohu.com/php/

P_PREFIX=/usr/local/php5/

echo -e '\033[32m----------------------------\033[0m'

echo

if [ -z "$1" ];  then

      echo -e "\033[36mPlease Select Install Menu follow:\033[0m"

      echo -e "\033[32m

1)编译安装 Apache 服务器\033[1m"

      echo "2)编译安装 MySQL 服务器"

      echo "3)编译安装 PHP 服务器"
```

```
    echo "4)配置 index.php 并启动 LAMP 服务"
    echo -e "\033[31mUsage: { /bin/sh $0 1|2|3|4|help}\033[0m"
    exit
fi
if [[ "$1" -eq "help" ]]; then
        echo -e "\033[36mPlease Select Install Menu follow:\033[0m"
        echo -e "\033[32m1)编译安装 Apache 服务器\033[1m"
        echo "2)编译安装 MySQL 服务器"
        echo "3)编译安装 PHP 服务器"
        echo "4)配置 index.php 并启动 LAMP 服务"
        echo -e "\033[31mUsage: { /bin/sh $0 1|2|3|4|help}\033[0m"
        exit
fi
#Install httpd web server
if [[ "$1" -eq "1" ]]; then
    wget -c $H_URL/$H_FILES && tar -jxvf $H_FILES && cd $H_FILES_DIR
&&./configure --prefix=$H_PREFIX
    if [ $? -lt 0 ]; then
        make && make install
    fi
fi
#Install Mysql DB server
```

```
if [[ "$1" -eq "2" ]]; then

wget -c $M_URL/$M_FILES && tar -xzvf $M_FILES && cd $M_FILES_DIR

&&yum install cmake -y ; cmake . -DCMAKE_INSTALL_PREFIX=$M_PREFIX \

-DMYSQL_UNIX_ADDR=/tmp/mysql.sock \

-DMYSQL_DATADIR=/data/mysql \

-DSYSCONFDIR=/etc \

-DMYSQL_USER=mysql \

-DMYSQL_TCP_PORT=3306 \

-DWITH_XTRADB_STORAGE_ENGINE=1 \

-DWITH_INNOBASE_STORAGE_ENGINE=1 \

-DWITH_PARTITION_STORAGE_ENGINE=1 \

-DWITH_BLACKHOLE_STORAGE_ENGINE=1 \

-DWITH_MYISAM_STORAGE_ENGINE=1 \

-DWITH_READLINE=1 \

-DENABLED_LOCAL_INFILE=1 \

-DWITH_EXTRA_CHARSETS=1 \

-DDEFAULT_CHARSET=utf8 \

-DDEFAULT_COLLATION=utf8_general_ci \

-DEXTRA_CHARSETS=all \

-DWITH_BIG_TABLES=1 \

-DWITH_DEBUG=0

make && make install
```

```
/bin/cp support-files/my-small.cnf    /etc/my.cnf

/bin/cp support-files/mysql.server /etc/init.d/mysqld

chmod +x /etc/init.d/mysqld

chkconfig --add mysqld

chkconfig mysqld on

if [ $? -eq 0 ];  then

      make && make install

      echo -e "\n\033[32m-----------------------------------------------\033[0m"

                  echo    -e    "\033[32mThe    $M_FILES_DIR    Server    Install
Success !\033[0m"

            else

                  echo  -e  "\033[32mThe  $M_FILES_DIR  Make  or  Make  install
ERROR,Please Check....."

                  exit 0

            fi

fi

#Install PHP server

if [[ "$1" -eq "3" ]];  then

            wget -c $P_URL/$P_FILES &&   tar -jxvf $P_FILES && cd $P_FILES_DIR
&&./configure     --prefix=$P_PREFIX     --with-config-file-path=$P_PREFIX/etc
--with-mysql=$M_PREFIX --with-apxs2=$H_PREFIX/bin/apxs

            if [ $? -eq 0 ];  then
```

```
                make ZEND_EXTRA_LIBS='-liconv' && make install
                echo                                                    -e
"\n\033[32m-----------------------------------------\033[0m"
                echo    -e    "\033[32mThe    $P_FILES_DIR    Server    Install
Success !\033[0m"
        else
                echo -e "\033[32mThe $P_FILES_DIR Make or Make install
ERROR,Please Check....."
                exit 0
        fi
fi
if [[ "$1" -eq "4" ]]; then
    sed     -i     '/DirectoryIndex/s/index.html/index.php     index.html/g'
$H_PREFIX/conf/httpd.conf
    $H_PREFIX/bin/apachectl restart
    echo                                                    "AddType
application/x-httpd-php .php" >>$H_PREFIX/conf/httpd.conf
    IP=`ifconfig eth1|grep "Bcast"|awk '{print $2}'|cut -d: -f2`
    echo "You can access http://$IP/"
cat >$H_PREFIX/htdocs/index.php <<EOF
<?php
phpinfo();
```

```
?>

EOF

fi
```

15.12 For 循环语句实战

for 循环语句主要用于对某个数据域进行循环读取、对文件进行遍历，通常用于需要循环某个文件或者列表。其语法格式以 for…do 开头，done 结尾。语法格式如下：

```
For  var  in  （表达式）

do

    语句 1

done
```

For 循环语句 Shell 脚本编程案例如下：

（1） 循环打印 BAT 企业官网：

```
#!/bin/bash

#By author jfedu.net 2017

for  website  in  www.baidu.com www.taobao.com www.qq.com

do

    echo  $website

done
```

（2） 循环打印 1 至 100 数字，seq 表示列出数据范围：

```
#!/bin/bash

#By author jfedu.net 2017
```

```
for    i    in   `seq 1 100`
do
     echo    “NUM is $i”
done
```

（3）　For 循环求 1-100 的总和：

```
#!/bin/bash
#By author jfedu.net 2017
#auto sum 1 100
j=0
for   ((i=1；i<=100；i++))
do
      j=`expr $i + $j`
done
echo $j
```

（4）　对系统日志文件进行分组打包：

```
#!/bin/bash
#By author jfedu.net 2017
for    i    in   `find /var/log   -name   “*.log” `
do
     tar   -czf   2017_log$i.tgz   $i
done
```

（5）　For 循环批量远程主机文件传输：

```
#!/bin/bash

#auto scp files for client

#By author jfedu.net 2017

for i in `seq 100 200`

do

        scp -r /tmp/jfedu.txt root@192.168.1.$i:/data/webapps/www

done
```

（6）　For 循环批量远程主机执行命令：

```
#!/bin/bash

#auto scp files for client

#By author jfedu.net 2017

for i in `seq 100 200`

do

          ssh -l    root 192.168.1.$i     'ls /tmp'

done
```

（7）　For 循环打印 10 秒等待提示：

```
for ((j=0；j<=10；j++))

do

        echo    -ne    "\033[32m-\033[0m"

        sleep 1

done

echo
```

15.13 While 循环语句实战

While 循环语句与 for 循环功能类似，主要用于对某个数据域进行循环读取、对文件进行遍历，通常用于需要循环某个文件或者列表，满足循环条件会一直循环，不满足则退出循环，其语法格式以 while…do 开头，done 结尾。语法格式如下：

```
while   (表达式)
do
        语句 1
done
```

While 循环语句 Shell 脚本编程案例如下：

（1） 循环打印 BAT 企业官网，read 指令用于读取行或者读取变量：

```
#!/bin/bash
#By author jfedu.net 2017
while read line
do
        echo $line
done <jfedu.txt
```

其中 jfedu.txt 内容为:

```
www.baidu.com
www.taobao.com
www.qq.com
```

（2） While 无限每秒输出 Hello World：

```
#!/bin/bash

#By author jfedu.net 2017

while sleep 1

do

        echo -e "\033[32mHello World.\033[0m"

done
```

其中 jfedu.txt 内容为:

```
www.baidu.com

www.taobao.com

www.qq.com
```

（3） 循环打印 1 至 100 数字，expr 用于运算逻辑工具：

```
#!/bin/bash

#By author jfedu.net 2017

i=0

while ((i<=100))

do

          echo   $i

          i=`expr $i + 1`

done
```

（4） While 循环求 1-100 的总和：

```
#!/bin/bash

#By author jfedu.net 2017
```

```
#auto sum 1 100

j=0

i=1

while ((i<=100))

do

        j=`expr $i + $j`

        ((i++))

done

echo $j
```

（5）　While 循环逐行读取文件：

```
#!/bin/bash

#By author jfedu.net 2017

while read line

do

      echo    $line;

done    < /etc/hosts
```

（6）　While 循环判断输入 IP 正确性：

```
#!/bin/bash

#By author jfedu.net 2017

#Check    IP    Address

read -p "Please enter ip Address,example 192.168.0.11 ip": IPADDR

echo $IPADDR|grep -v "[Aa-Zz]"|grep --color -E "([0-9]{1,3}\.){3}[0-9]{1,3}"
```

```
while [ $? -ne 0 ]
do
        read -p "Please enter ip Address,example 192.168.0.11 ip": IPADDR
        echo $IPADDR|grep -v "[Aa-Zz]"|grep --color -E "([0-9]{1,3}\.){3}[0-9]{1,3}"
done
```

（7） 每 5 秒循环判断/etc/passwd 是否被非法修改：

```
#!/bin/bash
#Check File to change.
#By author jfedu.net 2017
FILES="/etc/passwd"
while true
do
        echo "The Time is `date +%F-%T`"
        OLD=`md5sum $FILES|cut -d" " -f 1`
        sleep 5
        NEW=`md5sum $FILES|cut -d" " -f 1`
        if [[ $OLD != $NEW ]]; then
                echo "The $FILES has been modified."
        fi
done
```

（8） 每 10 秒循环判断 jfedu 用户是否登录系统：

```
#!/bin/bash
#Check File to change.
#By author jfedu.net 2017
USERS="jfedu"
while true
do
        echo "The Time is `date +%F-%T`"
        sleep 10
        NUM=`who|grep "$USERS"|wc -l`
        if [[ $NUM -ge 1 ]]; then
                echo "The $USERS is login in system."
        fi
done
```

15.14 Case 选择语句实战

Case 选择语句，主要用于对多个选择条件进行匹配输出，与 if elif 语句结构类似，通常用于脚本传递输入参数，打印出输出结果及内容，其语法格式以 Case…in 开头，esac 结尾。语法格式如下：

```
#!/bin/bash
#By author jfedu.net 2017
case  $1  in
    Pattern1)
```

```
    语句 1
    ；；
    Pattern2)
    语句 2
    ；；
    Pattern3)
    语句 3
    ；；
esac
```

Case 条件语句 Shell 脚本编程案例如下：

（1） 打印 Monitor 及 Archive 选择菜单：

```
#!/bin/bash
#By author jfedu.net 2017
case $1 in
        monitor)
        monitor_log
        ；；
        archive)
        archive_log
        ；；
        help )
        echo –e "\033[32mUsage:{$0 monitor | archive |help }\033[0m"
```

```
        ；；
        *)
        echo –e "\033[32mUsage:{$0 monitor | archive |help }\033[0m "
esac
```

（2） 自动修改 IP 脚本菜单：

```
#!/bin/bash
#By author jfedu.net 2017
case $i in
                modify_ip)
                change_ip
                ；；
                modify_hosts)
                change_hosts
                ；；
                exit)
                exit
                ；；
                *)
                echo -e "1) modify_ip\n2) modify_ip\n3)exit"
esac
```

15.15 Select 选择语句实战

Select 语句一般用于选择，常用于选择菜单的创建，可以配合 PS3 来做打印菜单的输

出信息，其语法格式以 select...in do 开头，done 结尾：

```
select i in （表达式）
do
        语句
done
```

Select 选择语句 Shell 脚本编程案例如下：

（1） 打印开源操作系统选择：

```
#!/bin/bash
#By author jfedu.net 2017
PS3="What you like most of the open source system?"
select i in CentOS RedHat Ubuntu
do
        echo "Your Select System: "$i
done
```

（2） 打印 LAMP 选择菜单

```
#!/bin/bash
#By author jfedu.net 2017
PS3="Please enter you select install menu:"
select i in http php mysql quit
do
case $i in
            http)
```

```
        echo Test Httpd.
        ;;
        php)
        echo Test PHP.
        ;;
        mysql)
        echo Test MySQL.
        ;;
        quit)
        echo The System exit.
        exit
esac
done
```

15.16 Shell 编程函数实战

Shell 允许将一组命令集或语句形成一个可用块，这些块称为 Shell 函数，Shell 函数的用于在于只需定义一次，后期随时使用即可，无需在 Shell 脚本中添加重复的语句块，其语法格式以 function name（）{开头，以}结尾。

Shell 编程函数默认不能将参数传入（）内部，Shell 函数参数传递在调用函数名称传递，例如 name args1 args2。

```
function name (){
        command1
```

```
        command2

        ........

}

name args1 args2
```

（1） 创建 Apache 软件安装函数，给函数 Apache_install 传递参数 1：

```
#!/bin/bash

#auto install LAMP

#By author jfedu.net 2017

#Httpd define path variable

H_FILES=httpd-2.2.31.tar.bz2

H_FILES_DIR=httpd-2.2.31

H_URL=http://mirrors.cnnic.cn/apache/httpd/

H_PREFIX=/usr/local/apache2/

function Apache_install（）

{

#Install httpd web server

if [[ "$1" -eq "1" ]]；then

    wget -c $H_URL/$H_FILES && tar -jxvf $H_FILES && cd $H_FILES_DIR

&&./configure --prefix=$H_PREFIX

    if [ $? -eq 0 ]；then

        make && make install

        echo                                                    -e
```

```
"\n\033[32m-------------------------------------------\033[0m"
        echo -e "\033[32mThe $H_FILES_DIR Server Install Success !\033[0m"
    else
        echo -e "\033[32mThe $H_FILES_DIR Make or Make install ERROR,Please Check....."
        exit 0
    fi
fi
}
Apache_install 1
```

（2） 创建 judge_ip 判断 IP 函数：

```
#!/bin/bash
#By author jfedu.net 2017
judge_ip(){
        read -p "Please enter ip Address,example 192.168.0.11 ip": IPADDR
        echo $IPADDR|grep -v "[Aa-Zz]"|grep --color -E "([0-9]{1,3}\.){3}[0-9]{1,3}"
}
judge_ip
```

15.17 Shell 数组编程实战

数组是相同数据类型的元素按一定顺序排列的集合，把有限个类型相同的变量用一个名

字命名，然后用编号区分他们变量的集合，这个名称称之为数组名，编号成为下标。Linux Shell 编程中常用一维数组。

数组的设计其实了为了处理方便，把具有相同类型的若干变量按有序的形式组织起来的一种形式，以减少重复频繁的单独定义。如图 17-3 所示：

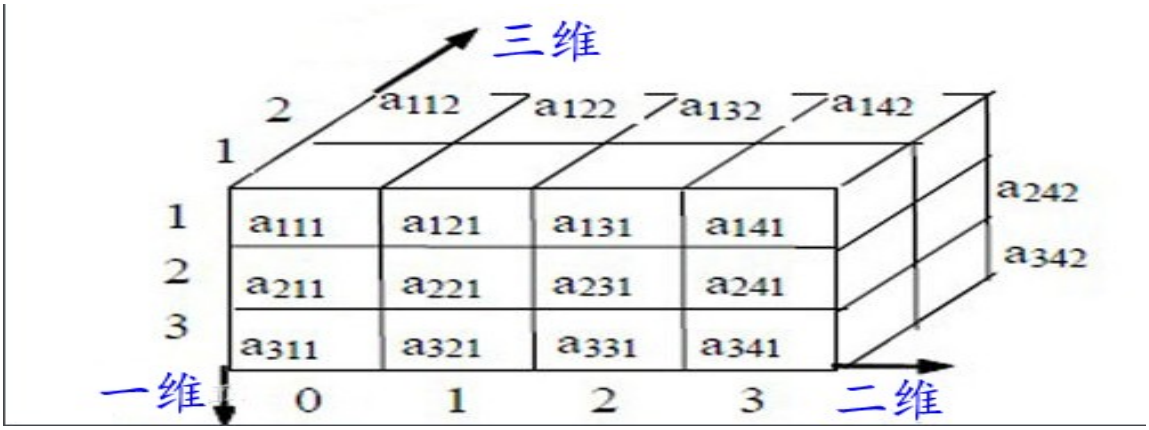


图 17-3 一维、二维、三维数组

定义数组一般以小括号的方式来定义，数组的值可以随机指定，如下为一维数组的定义、统计、引用和删除操作，：

（1） 一唯数组定义及创建：

```
JFTEST=(
        test1
        test2
        test3
)
LAMP=(httpd  php  php-devel php-mysql mysql mysql-server)
```

（2） 数组下标一般从 0 开始，如下为引用数组的方法：

```
echo    ${JFTEST[0]}                引用第一个数组变量，结果打印 test1；
echo    ${JFTEST[1]}            引用第二个数组变量；
```

```
echo     ${JFTEST[@]}          显示该数组所有参数；

echo     ${#JFTEST[@]}         显示该数组参数个数；

echo      ${#JFTEST[0]}         显示 test1 字符长度；

echo      ${JFTEST[@]:0}        打印数组所有的值；

echo      ${JFTEST[@]:1}        打印从第二个值开始的所有值；

echo      ${JFTEST[@]:0:2}      打印从第一个值与第二个值；

echo      ${JFTEST[@]:1:2}      打印从第二个值与第三个值。
```

（3）　数组替换操作：

```
JFTEST=( [0]=www1 [1]=www2 [2]=www3 )          数组赋值；

echo   ${JFTEST[@]/test/jfedu}                 将数组值 test 替换为 jfedu；

NEWJFTEST=`echo   ${JFTEST[@]/test/jfedu}`     将结果赋值新数组。
```

（4）　数组删除操作：

```
unset array[0]                  删除数组第一个值；

unset array[1]                  删除数组第二个值；

unset array                     删除整个数组。
```

（5）　数组 Shell 脚本企业案例一，网卡 bond 绑定脚本：

```
#!/bin/bash

#Auto Make KVM Virtualization

#Auto config bond scripts

#By author jfedu.net 2017

eth_bond()

{
```

```
NETWORK=(
  HWADDR=`ifconfig eth0 |egrep "HWaddr|Bcast" |tr "\n" " "|awk '{print $5,$7,$NF}'|sed -e 's/addr://g' -e 's/Mask://g'|awk '{print $1}'`
  IPADDR=`ifconfig eth0 |egrep "HWaddr|Bcast" |tr "\n" " "|awk '{print $5,$7,$NF}'|sed -e 's/addr://g' -e 's/Mask://g'|awk '{print $2}'`
  NETMASK=`ifconfig eth0 |egrep "HWaddr|Bcast" |tr "\n" " "|awk '{print $5,$7,$NF}'|sed -e 's/addr://g' -e 's/Mask://g'|awk '{print $3}'`
  GATEWAY=`route -n|grep "UG"|awk '{print $2}'`
)
cat >ifcfg-bond0<<EOF
DEVICE=bond0
BOOTPROTO=static
${NETWORK[1]}
${NETWORK[2]}
${NETWORK[3]}
ONBOOT=yes
TYPE=Ethernet
NM_CONTROLLED=no
EOF
```

（6） 数组 Shell 脚本企业案例二，定义 IPv4 值:

```
#!/bin/bash
#auto Change ip netmask gateway scripts
```

```
#By author jfedu.net 2017

ETHCONF=/etc/sysconfig/network-scripts/ifcfg-eth0

HOSTS=/etc/hosts

NETWORK=/etc/sysconfig/network

DIR=/data/backup/`date +%Y%m%d`

NETMASK=255.255.255.0

echo "----------------------------"

count_ip(){

        count=(`echo    $IPADDR|awk    -F.    '{print $1,$2,$3,$4}'`)

        IP1=${count[0]}

        IP2=${count[1]}

        IP3=${count[2]}

        IP4=${count[3]}

}
```

15.18 Shell 编程四剑客之 Find

通过如上基础语法的学习，读者对 Shell 编程有了更近一步的理解，Shell 编程不再是简单命令的堆积，而是演变成了各种特殊的语句、各种语法、编程工具、各种命令的集合。

在 Shell 编程工具中，四剑客工具的使用更加的广泛，Shell 编程四剑客包括：find、sed、grep、awk，熟练掌握四剑客会对 Shell 编程能力极大的提升。

四剑客之 Find 工具实战，Find 工具主要用于操作系统文件、目录的查找，其语法参数格式为：

```
find    path    -option    [    -print ]    [ -exec    -ok    command ]    { }   \;
```

其 option 常用参数详解如下：

```
-name     filename                  #查找名为 filename 的文件；
-type     b/d/c/p/l/f               #查是块设备、目录、字符设备、管道、符号链接、普
通文件；
-size       n[c]                    #查长度为 n 块[或 n 字节]的文件；
-perm                               #按执行权限来查找；
-user     username                  #按文件属主来查找；
-group    groupname                 #按组来查找；
-mtime     -n +n                    #按文件更改时间来查找文件，-n 指 n 天以内，+n
指 n 天以前；
-atime     -n +n                        #按文件访问时间来查找文件；
-ctime     -n +n                        #按文件创建时间来查找文件；
-mmin      -n +n                        #按文件更改时间来查找文件，-n 指 n 分钟以
内，+n 指 n 分钟以前；
-amin      -n +n                        #按文件访问时间来查找文件；
-cmin      -n +n                        #按文件创建时间来查找文件；
-nogroup                                #查无有效属组的文件；
-nouser                                 #查无有效属主的文件；
-newer     f1 !f2                   #找文件，-n 指 n 天以内，+n 指 n 天以前；
-depth                                  #使查找在进入子目录前先行查找完本目录；
-fstype                             #查更改时间比 f1 新但比 f2 旧的文件；
```

```
-mount                          #查文件时不跨越文件系统 mount 点;
-follow                         #如果遇到符号链接文件，就跟踪链接所指的文件;
-cpio                           #查位于某一类型文件系统中的文件;
-prune                          #忽略某个目录;
-maxdepth                       #查找目录级别深度。
```

(1) Find 工具-name 参数案列:

```
find /data/ -name "*.txt"          #查找/data/目录以.txt 结尾的文件;
find /data/ -name "[A-Z]*"         #查找/data/目录以大写字母开头的
文件;
find /data/ -name "test*"          #查找/data/目录以 test 开头的文
件;
```

(2) Find 工具-type 参数案列:

```
find /data/ -type d                #查找/data/目录下的文件夹;
find /data/ ! -type d              #查找/data/目录下的非文件夹;
find /data/ -type l                #查找/data/目录下的链接文件。
find /data/ -type d|xargs chmod 755 -R  #查目录类型并将权限设置为 755;
find /data/ -type f|xargs chmod 644 -R  #查文件类型并将权限设置为 644;
```

(3) Find 工具-size 参数案列:

```
find /data/ -size +1M              #查文件大小大于 1Mb 的文件;
find /data/ -size 10M              #查文件大小为 10M 的文件;
find /data/ -size -1M              #查文件大小小于 1Mb 的文件;
```

(4) Find 工具-perm 参数案列:

```
find    /data/    -perm    755          #查找/data/目录权限为 755 的文件或者
目录；
find    /data/    -perm    -007         #与-perm 777 相同，表示所有权限；
find    /data/    -perm    +644         #文件权限符号 644 以上；
```

（5） Find 工具-mtime 参数案列：

```
atime,access time    文件被读取或者执行的时间；
ctime,change time    文件状态改变时间；
mtime,modify time    文件内容被修改的时间；
find /data/ -mtime +30   -name   "*.log"    #查找 30 天以前的 log 文件；
find /data/ -mtime -30   -name   "*.txt"    #查找 30 天以内的 log 文件；
find /data/ -mtime 30    -name    "*.txt"   #查找第 30 天的 log 文件；
find /data/ -mmin   +30  -name    "*.log"    #查找 30 分钟以前修改的 log 文
件；
find /data/ -amin   -30   -name    "*.txt"   #查找 30 分钟以内被访问的 log 文
件；
find /data/ -cmin   30    -name    "*.txt"   #查找第 30 分钟改变的 log 文件。
```

（6） Find 工具参数综合案列：

```
#查找/data 目录以.log 结尾，文件大于 10k 的文件，同时 cp 到/tmp 目录；
find /data/ -name "*.log"   –type f   -size +10k -exec cp {} /tmp/ \;
#查找/data 目录以.txt 结尾，文件大于 10k 的文件，权限为 644 并删除该文件；
find /data/ -name "*.log"   –type f   -size +10k   -m perm 644 -exec rm –rf {} \;
#查找/data 目录以.log 结尾，30 天以前的文件，大小大于 10M 并移动到/tmp 目录；
```

```
find /data/ -name "*.log"  –type f  -mtime +30 –size +10M -exec mv {} /tmp/
\;
```

15.19 Shell 编程四剑客之 SED

SED 是一个非交互式文本编辑器，它可对文本文件和标准输入进行编辑，标准输入可以来自键盘输入、文本重定向、字符串、变量，甚至来自于管道的文本，与 VIM 编辑器类似，它一次处理一行内容，Sed 可以编辑一个或多个文件，简化对文件的反复操作、编写转换程序等。

在处理文本时把当前处理的行存储在临时缓冲区中，称为“模式空间”（pattern space），紧接着用 SED 命令处理缓冲区中的内容，处理完成后把缓冲区的内容输出至屏幕或者写入文件。

逐行处理直到文件末尾，然而如果打印在屏幕上，实质文件内容并没有改变，除非你使用重定向存储输出或者写入文件。其语法参数格式为：

```
sed    [-Options]    [ ‘Commands’ ]    filename;
sed 工具默认处理文本，文本内容输出屏幕已经修改，但是文件内容其实没有修改，需要加-i 参数即对文件彻底修改；
x                          #x 为指定行号；
x,y                        #指定从 x 到 y 的行号范围；
/pattern/                  #查询包含模式的行；
/pattern/pattern/              #查询包含两个模式的行；
/pattern/,x                #从与 pattern 的匹配行到 x 号行之间的行；
x,/pattern/                #从 x 号行到与 pattern 的匹配行之间的行；
```

x,y!	#查询不包括 x 和 y 行号的行；
r	#从另一个文件中读文件；
w	#将文本写入到一个文件；
y	#变换字符；
q	#第一个模式匹配完成后退出；
l	#显示与八进制 ASCII 码等价的控制字符；
{}	#在定位行执行的命令组；
p	#打印匹配行；
=	#打印文件行号；
a\	#在定位行号之后追加文本信息；
i\	#在定位行号之前插入文本信息；
d	#删除定位行；
c\	#用新文本替换定位文本；
s	#使用替换模式替换相应模式；
n	#读取下一个输入行，用下一个命令处理新的行；
N	#将当前读入行的下一行读取到当前的模式空间。
h	#将模式缓冲区的文本复制到保持缓冲区；
H	#将模式缓冲区的文本追加到保持缓冲区；
x	#互换模式缓冲区和保持缓冲区的内容；
g	#将保持缓冲区的内容复制到模式缓冲区；
G	#将保持缓冲区的内容追加到模式缓冲区。

常用 SED 工具企业演练案列：

（1） 替换 jfedu.txt 文本中 old 为 new：

```
sed 's/old/new/g' jfedu.txt
```

（2） 打印 jfedu.txt 文本第一行至第三行：

```
sed -n '1,3p' jfedu.txt
```

（3） 打印 jfedu.txt 文本中第一行与最后一行：

```
sed -n '1p；$p' jfedu.txt
```

（4） 删除 jfedu.txt 第一行至第三行、删除匹配行至最后一行：

```
sed '1,3d' jfedu.txt
sed '/jfedu/,$d' jfedu.txt
```

（5） 删除 jfedu.txt 最后 6 行及删除最后一行：

```
for i in `seq 1 6`; do sed -i '$d' jfedu.txt ; done
sed '$d' jfedu.txt
```

（6） 删除 jfedu.txt 最后一行：

```
sed '$d' jfedu.txt
```

（7） 在 jfedu.txt 查找 jfedu 所在行，并在其下一行添加 word 字符，a 表示在其下一行添加字符串：

```
sed '/jfedu/aword' jfedu.txt
```

（8） 在 jfedu.txt 查找 jfedu 所在行，并在其上一行添加 word 字符，i 表示在其上一行添加字符串：

```
sed '/jfedu/iword' jfedu.txt
```

（9） 在 jfedu.txt 查找以 test 结尾的行尾添加字符串 word，$表示结尾标识，&在 Sed 中表示添加：

```
sed    's/test$/&word/g'       jfedu.txt
```

（10） 在 jfedu.txt 查找 www 的行，在其行首添加字符串 word，^表示起始标识，&在 Sed 中表示添加：

```
sed    '/www/s/^/&word/'      jfedu.txt
```

（11） 多个 sed 命令组合，使用-e 参数：

```
sed   -e   '/www.jd.com/s/^/&1./'   -e   's/www.jd.com$/&./g'   jfedu.txt
```

（12） 多个 sed 命令组合，使用分号“；”分割：

```
sed   -e   '/www.jd.com/s/^/&1./；s/www.jd.com$/&./g'   jfedu.txt
```

（13） Sed 读取系统变量，变量替换：

```
WEBSITE=WWW.JFEDU.NET

Sed    “s/www.jd.com/$WEBSITE/g”   jfedu.txt
```

（14） 修改 Selinux 策略 enforcing 为 disabled，查找/SELINUX/行，然后将其行 enforcing 值改成 disabled、!s 表示不包括 SELINUX 行：

```
sed   -i    '/SELINUX/s/enforcing/disabled/g' /etc/selinux/config

sed   -i    '/SELINUX/!s/enforcing/disabled/g' /etc/selinux/config
```

通常而言，SED 将待处理的行读入模式空间，脚本中的命令逐行进行处理，直到脚本执行完毕，然后该行被输出，模式空间请空；然后重复刚才的动作，文件中的新的一行被读入，直到文件处理完备。

如果用户希望在某个条件下脚本中的某个命令被执行，或者希望模式空间得到保留以便下一次的处理，都有可能使得 sed 在处理文件的时候不按照正常的流程来进行。这时可以使用 SED 高级语法来满足用户需求。总的来说，SED 高级命令可以分为三种功能：

- N、D、P：处理多行模式空间的问题；

- H、h、G、g、x：将模式空间的内容放入存储空间以便接下来的编辑；
- :、b、t：在脚本中实现分支与条件结构。

(1) 在 jfedu.txt 每行后加入空行，也即每行占永两行空间，每一行后边插入一行空行、两行空行及前三行每行后插入空行：

```
sed '/^$/d；G' jfedu.txt

sed '/^$/d；G；G' jfedu.txt

sed '/^$/d；1,3G；' jfedu.txt
```

(2) 将 jfedu.txt 偶数行删除及隔两行删除一行：

```
sed 'n；d' jfedu.txt

sed 'n；n；d' jfedu.txt
```

(3) 在 jfedu.txt 匹配行前一行、后一行插入空行以及同时在匹配前后插入空行：

```
sed '/jfedu/{x；p；x；}' jfedu.txt

sed '/jfedu/G' jfedu.txt

sed '/jfedu/{x；p；x；G；}' jfedu.txt
```

(4) 在 jfedu.txt 每行后加入空行，也即每行占永两行空间，每一行后边插入空行：

```
sed '/^$/d；G' jfedu.txt
```

(5) 在 jfedu.txt 每行后加入空行，也即每行占永两行空间，每一行后边插入空行：

```
sed '/^$/d；G' jfedu.txt
```

(6) 在 jfedu.txt 每行前加入顺序数字序号、加上制表符\t 及.符号：

```
sed = jfedu.txt| sed 'N；s/\n/ /'

sed = jfedu.txt| sed 'N；s/\n/\t/'

sed = jfedu.txt| sed 'N；s/\n/\./'
```

（7） 删除 jfedu.txt 行前和行尾的任意空格：

```
sed 's/^[ \t]*//；s/[ \t]*$//' jfedu.txt
```

（8） 打印 jfedu.txt 关键词 old 与 new 之间的内容：

```
sed -n '/old/,/new/'p       jfedu.txt
```

（9） 打印及删除 jfedu.txt 最后两行：

```
sed    '$!N；$!D'                jfedu.txt

sed    'N；$!P；$!D；$d'         jfedu.txt
```

（10） 合并上下两行，也即两行合并：

```
sed     '$!N；s/\n/ /'           jfedu.txt

sed     'N；s/\n/ /'             jfedu.txt
```

15.20 Shell 编程四剑客之 AWK

AWK 是一个优良的文本处理工具，Linux 及 Unix 环境中现有的功能最强大的数据处理引擎之一，以 Aho、Weinberger、Kernighan 三位发明者名字首字母命名为 AWK，AWK 是一个行级文本高效处理工具，AWK 经过改进生成的新的版本有 Nawk、Gawk，一般 Linux 默认为 Gawk，Gawk 是 AWK 的 GNU 开源免费版本。

AWK 基本原理是逐行处理文件中的数据，查找与命令行中所给定内容相匹配的模式，如果发现匹配内容，则进行下一个编程步骤，如果找不到匹配内容，则 继续处理下一行。其语法参数格式为，AWK 常用参数、变量、函数详解如下：

```
awk     'pattern    +    {action}'       file
```

（1） AWK 基本语法参数详解：

- 单引号' '是为了和 shell 命令区分开；

- 大括号{ }表示一个命令分组；
- pattern 是一个过滤器，表示匹配 pattern 条件的行才进行 Action 处理；
- action 是处理动作，常见动作为 Print；
- 使用#作为注释，pattern 和 action 可以只有其一，但不能两者都没有。

(2) AWK 内置变量详解：

- FS 分隔符，默认是空格；
- OFS 输出分隔符；
- NR 当前行数，从 1 开始；
- NF 当前记录字段个数；
- $0 当前记录；
- $1~$n 当前记录第 n 个字段（列）。

(3) AWK 内置函数详解：

- gsub(r,s)：在$0 中用 s 代替 r；
- index(s,t)：返回 s 中 t 的第一个位置；
- length(s)：s 的长度；
- match(s,r)：s 是否匹配 r；
- split(s,a,fs)：在 fs 上将 s 分成序列 a；
- substr(s,p)：返回 s 从 p 开始的子串。

(4) AWK 常用操作符，运算符及判断符：

- ++ -- 增加与减少(前置或后置)；
- ^ ** 指数(右结合性)；
- ! + - 非、一元(unary) 加号、一元减号；

- + - * / %　　加、减、乘、除、余数；
- < <= == != > >=　　数字比较；
- &&　　逻辑 and；
- ||　　逻辑 or；
- = += -= *= /= %= ^= **=　　赋值。

(5)　AWK 与流程控制语句：

- if(condition) { } else { };
- while { };
- do{ }while(condition);
- for(init；condition；step){ };
- break/continue。

常用 AWK 工具企业演练案列：

(1)　AWK 打印硬盘设备名称，默认以空格为分割：

```
df　　-h|awk　'{print　$1}'
```

(2)　AWK 以空格、冒号、\t、分号为分割：

```
awk　-F '[ :\t；]'　'{print　$1}'　　jfedu.txt
```

(3)　AWK 以冒号分割，打印第一列，同时将内容追加到/tmp/awk.log 下：

```
awk　-F:　'{print $1 >>"/tmp/awk.log"}'　jfedu.txt
```

(4)　打印 jfedu.txt 文件中的第 3 行至第 5 行，NR 表示打印行，$0 表示文本所有域：

```
awk 'NR==3,NR==5　{print}'　　jfedu.txt

awk 'NR==3,NR==5　{print $0}'　　jfedu.txt
```

(5)　打印 jfedu.txt 文件中的第 3 行至第 5 行的第一列与最后一列：

```
awk 'NR==3,NR==5 {print $1,$NF}'          jfedu.txt
```

（6）　打印 jfedu.txt 文件中，长度大于 80 的行号：

```
awk    'length($0)>80 {print NR}'          jfedu.txt
```

（7）　AWK 引用 Shell 变量，使用-v 或者双引号+单引号即可：

```
awk -v STR=hello   '{print STR,$NF}'          jfedu.txt

STR="hello"；echo| awk   '{print "'${STR}'"；}'
```

（8）　AWK 以冒号切割，打印第一列同时只显示前 5 行：

```
cat   /etc/passwd|head -5|awk   -F:    '{print $1}'

awk   -F:   'NR>=1&&NR<=5 {print $1}'   /etc/passwd
```

（9）　Awk 指定文件 jfedu.txt 第一列的总和：

```
cat jfedu.txt |awk '{sum+=$1}END{print sum}'
```

（10）　AWK NR 行号除以 2 余数为 0 则跳过该行，继续执行下一行，打印在屏幕：

```
awk   -F:   'NR%2==0 {next} {print NR,$1}'   /etc/passwd
```

（11）　AWK 添加自定义字符：

```
ifconfig   eth0|grep "Bcast"|awk '{print "ip_"$2}'
```

（12）　AWK 格式化输出 passwd 内容，printf 打印字符串，%格式化输出分隔符，s 表示字符串类型，-12 表示 12 个字符，-6 表示 6 个字符：

```
awk -F:   '{printf "%-12s %-6s %-8s\n",$1,$2,$NF}'   /etc/passwd
```

（13）　AWK OFS 输出格式化\t：

```
netstat   -an|awk   '$6   ~   /LISTEN/&&NR>=1&&NR<=10   {print   NR,$4,$5,$6}'

OFS="\t"
```

（14）　AWK 与 if 组合实战，判断数字比较：

```
echo 3 2 1 | awk '{ if(($1>$2)||($1>$3)) { print $2} else {print $1} }'
```

（15） AWK 与数组组合实战，统计 passwd 文件用户数：

```
awk -F ':' 'BEGIN {count=0；} {name[count] = $1；count++；}； END{for (i = 0；
i < NR； i++) print i, name[i]}' /etc/passwd
```

（16） awk 分析 Nginx 访问日志的状态码 404、502 等错误信息页面，统计次数大于 20 的 IP 地址。

```
awk '{if ($9~/502|499|500|503|404/) print $1,$9}' access.log|sort|uniq –c|sort –
nr | awk '{if($1>20) print $2}'
```

（17） 用/etc/shadow 文件中的密文部分替换/etc/passwd 中的"x"位置,生成新的/tmp/passwd 文件。

```
awk 'BEGIN{OFS=FS=":"} NR==FNR{a[$1]=$2}NR>FNR{$2=a[$1];
print >>"/tmp/passwd"}' /etc/shadow /etc/passwd
```

（18） Awk 统计服务器状态连接数：

```
netstat -an | awk '/tcp/ {s[$NF]++} END {for(a in s) {print a,s[a]}}'
netstat -an | awk '/tcp/ {print $NF}' | sort | uniq -c
```

15.21 Shell 编程四剑客之 GREP

全面搜索正则表达式（Global search regular expression(RE)，GREP）是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。

Unix/Linux 的 grep 家族包括 grep、egrep 和 fgrep，其中 egrep 和 fgrep 的命令跟 grep 有细微的区别，egrep 是 grep 的扩展，支持更多的 re 元字符， fgrep 是 fixed grep 或 fast grep 简写，它们把所有的字母都看作单词，正则表达式中的元字符表示其自身的字

面意义，不再有其他特殊的含义，一般使用比较少。

目前 Linux 操作系统默认使用 GNU 版本的 grep。它功能更强，可以通过-G、-E、-F 命令行选项来使用 egrep 和 fgrep 的功能。其语法格式及常用参数详解如下：

grep -[acinv] 'word' Filename

Grep 常用参数详解如下：

-a	以文本文件方式搜索；
-c	计算找到的符合行的次数；
-i	忽略大小写；
-n	顺便输出行号；
-v	反向选择，即显示不包含匹配文本的所有行；
-h	查询多文件时不显示文件名；
-l	查询多文件时只输出包含匹配字符的文件名；
-s	不显示不存在或无匹配文本的错误信息；
-E	允许使用 egrep 扩展模式匹配。

学习 Grep 时，需要了解通配符、正则表达式两个概念，很多读者容易把彼此搞混淆，通配符主要用在 Linux 的 Shell 命令中，常用于文件或者文件名称的操作，而正则表达式用于文本内容中的字符串搜索和替换，常用在 AWK、GREP、SED、VIM 工具中对文本的操作。

通配符类型详解：

*	0 个或者多个字符、数字；
?	匹配任意一个字符；
#	表示注解；

|	管道符号；
;	多个命令连续执行；
&	后台运行指令；
!	逻辑运算非；
[ ]	内容范围，匹配括号中内容；
{ }	命令块，多个命令匹配。

正则表达式详解：

*	前一个字符匹配 0 次或多次；
.	匹配除了换行符以外任意一个字符；
.*	代表任意字符；
^	匹配行首，即以某个字符开头；
$	匹配行尾，即以某个字符结尾；
\(..\)	标记匹配字符；
[]	匹配中括号里的任意指定字符，但只匹配一个字符；
[^]	匹配除中括号以外的任意一个字符；
\	转义符，取消特殊含义；
\<	锚定单词的开始；
\>	锚定单词的结束；
{n}	匹配字符出现 n 次；
{n,}	匹配字符出现大于等于 n 次；
{n,m}	匹配字符至少出现 n 次，最多出现 m 次；
\w	匹配文字和数字字符；

\W	\w 的反置形式，匹配一个或多个非单词字符；
\b	单词锁定符；
\s	匹配任何空白字符；
\d	匹配一个数字字符，等价于[0-9]。

常用 GREP 工具企业演练案列：

grep	-c "test"	jfedu.txt	统计 test 字符总行数；
grep	-i "TEST"	jfedu.txt	不区分大小写查找 TEST 所有的行；
grep	-n "test"	jfedu.txt	打印 test 的行及行号；
grep	-v "test"	jfedu.txt	不打印 test 的行；
grep	"test[53]"	jfedu.txt	以字符 test 开头，接 5 或者 3 的行；
grep	"^[^test]"	jfedu.txt	显示输出行首不是 test 的行；
grep	"[Mm]ay"	jfedu.txt	匹配 M 或 m 开头的行；
grep	"K...D"	jfedu.txt	匹配 K，三个任意字符，紧接 D 的行；
grep	"[A-Z][9]D"	jfedu.txt	匹配大写字母，紧跟 9D 的字符行；
grep	"T\{2,\}"	jfedu.txt	打印字符 T 字符连续出现 2 次以上的行；
grep	"T\{4,6\}"	jfedu.txt	打印字符 T 字符连续出现 4 次及 6 次的行；
grep	-n "^$"	jfedu.txt	打印空行的所在的行号；
grep	-vE "#|^$"	jfedu.txt	不匹配文件中的#和空行；
grep	--color -ra -E "db|config|sql"	*	匹配包含 db 或者 config 或者 sql 的文件；
grep	--color -E "\<([0-9]{1,3}\.){3}([0-9]{1,3})\>"	jfedu.txt	匹配 IPV4 地址。

第16章　Linux SHELL 编程高级篇

企业生产环境中，服务器规模成百上千，如果依靠人工去维护和管理是非常吃力的，基于 Shell 编程脚本管理和维护服务器变得简单、从容，而且对企业自动化运维之路的建设起到极大的推动作用。

本章向读者介绍企业生产环境 Shell 编程案例、自动化备份 MySQL 数据、服务器信息收集、防止恶意 IP 访问、LAMP+MySQL 主从实战、千台服务器 IP 修改、Nginx+Tomcat 高级自动化部署脚本、Nginx 虚拟主机配置、Docker 管理平台等。

16.1 Shell 编程实战系统备份脚本

日常企业运维中，需要备份 Linux 操作系统中重要的文件，例如/etc、/boot 分区、重要网站数据等，在备份数据时，由于数据量非常大，需要指定高效的备份方案，如下为常用的备份数据方案：

- 每周日进行完整备份，周一至周六使用增量备份；
- 每周六进行完整备份，周日至周五使用增量备份。

企业备份数据的工具主要有：tar、cp、rsync、scp、sersync、dd 等工具。如下为基于开源 tar 工具实现系统数据备份方案：

Tar 工具手动全备份网站，-g 参数指定新的快照文件：

```
tar -g /tmp/snapshot -czvf /tmp/2017_full_system_data.tar.gz /data/sh/
```

Tar 工具手动增量备份网站，-g 参数指定全备已生成的快照文件，后续增量备份基于上一个增量备份快照文件：

```
tar -g /tmp/snapshot -czvf /tmp/2014_add01_system_data.tar.gz /data/sh/
```

Tar 工具全备、增量备份网站，Shell 脚本实现自动打包备份编写思路如下：

- 系统备份数据按每天存放；
- 创建完整备份函数块；
- 创建增量备份函数块；
- 根据星期数判断完整或增量；
- 将脚本加入 Crontab 实现自动备份；

Tar 工具全备、增量备份网站，Shell 脚本实现自动打包备份，代码如下：

```
#!/bin/bash
#Auto Backup Linux System Files
#By author jfedu.net 2017
#Define Path variables
SOURCE_DIR=(
    $*
)
TARGET_DIR=/data/backup/
YEAR=`date +%Y`
MONTH=`date +%m`
DAY=`date +%d`
WEEK=`date +%u`
A_NAME=`date +%H%M`
```

```
FILES=system_backup.tgz
CODE=$?
if
    [ -z "$*" ]; then
    echo -e "\033[32mUsage:\nPlease Enter Your Backup Files or Directories\n----------\n\nUsage: { $0 /boot /etc}\033[0m"
    exit
fi
#Determine Whether the Target Directory Exists
if
    [ ! -d $TARGET_DIR/$YEAR/$MONTH/$DAY ]; then
    mkdir -p $TARGET_DIR/$YEAR/$MONTH/$DAY
    echo -e "\033[32mThe $TARGET_DIR Created Successfully !\033[0m"
fi
#EXEC Full_Backup Function Command
Full_Backup()
{
if
    [ "$WEEK" -eq "7" ]; then
    rm -rf $TARGET_DIR/snapshot
    cd $TARGET_DIR/$YEAR/$MONTH/$DAY ; tar -g $TARGET_DIR/snapshot
```

```
-czvf $FILES ${SOURCE_DIR[@]}
    [ "$CODE" == "0" ]&&echo -e
"----------------------------------------\n\033[32mThese Full_Backup System
Files Backup Successfully !\033[0m"
fi
}
#Perform incremental BACKUP Function Command
Add_Backup()
{
    if
        [ $WEEK -ne "7" ]; then
        cd $TARGET_DIR/$YEAR/$MONTH/$DAY ; tar -g
$TARGET_DIR/snapshot -czvf $A_NAME$FILES ${SOURCE_DIR[@]}
        [ "$CODE" == "0" ]&&echo -e
"----------------------------------------\n\033[32mThese Add_Backup System
Files $TARGET_DIR/$YEAR/$MONTH/$DAY/${YEAR}_$A_NAME$FILES Backup
Successfully !\033[0m"
    fi
}
sleep 3
Full_Backup; Add_Backup
```

Crontab 任务计划中添加如下语句，每天凌晨 1 点整执行备份脚本即可：

```
0  1  *  *  * /bin/sh /data/sh/auto_backup.sh /boot /etc/ >> /tmp/back.log
2>&1
```

16.2 Shell 编程实战收集服务器信息脚本

在企业上产环境中，经常会对服务器资产进行统计存档，单台服务器可以手动去统计服务器的 CPU 型号、内存大小、硬盘容量、网卡流量等，如果服务器数量超过百台、千台，使用手工方式就变得非常吃力。

基于 Shell 脚本实现自动化服务器硬件信息的收集，并将收集的内容存放在数据库，能更快、更高效的实现对服务器资产信息的管理。Shell 脚本实现服务器信息自动收集，编写思路如下：

- 创建数据库和表存储服务器信息；
- 基于 Shell 四剑客 awk、find、sed、grep 获取服务器信息；
- 将获取的信息写成 SQL 语句；
- 定期对 SQL 数据进行备份；
- 将脚本加入 Crontab 实现自动备份；

创建数据库表，创建 SQL 语句如下：

```
CREATE TABLE `audit_system` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `ip_info` varchar(50) NOT NULL,
  `serv_info` varchar(50) NOT NULL,
  `cpu_info` varchar(50) NOT NULL,
  `disk_info` varchar(50) NOT NULL,
```

```
  `mem_info` varchar(50) NOT NULL,
  `load_info` varchar(50) NOT NULL,
  `mark_info` varchar(50) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ip_info` (`ip_info`),
  UNIQUE KEY `ip_info_2` (`ip_info`)
  );
```

Shell 脚本实现服务器信息自动收集，代码如下：

```
#!/bin/bash
#Auto get system info
#By author jfedu.net 2017
#Define Path variables
echo -e "\033[34m \033[1m"
cat <<EOF
++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++Welcome to use system Collect+++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++
EOF
ip_info=`ifconfig |grep "Bcast"|tail -1 |awk '{print $2}'|cut -d: -f 2`
cpu_info1=`cat /proc/cpuinfo |grep 'model name'|tail -1 |awk -F: '{print $2}'|sed
's/^ //g'|awk '{print $1,$3,$4,$NF}'`
cpu_info2=`cat /proc/cpuinfo |grep "physical id"|sort |uniq -c|wc -l`
```

```
serv_info=`hostname |tail -1`
disk_info=`fdisk -l|grep "Disk"|grep -v "identifier"|awk '{print $2,$3,$4}'|sed 's/,//g'`
mem_info=`free -m |grep "Mem"|awk '{print "Total",$1,$2"M"}'`
load_info=`uptime |awk '{print "Current Load: "$(NF-2)}'|sed 's/\,//g'`
mark_info='BeiJing_IDC'
echo -e "\033[32m----------------------------------------\033[1m"
echo IPADDR:${ip_info}
echo HOSTNAME:$serv_info
echo CPU_INFO:${cpu_info1} X${cpu_info2}
echo DISK_INFO:$disk_info
echo MEM_INFO:$mem_info
echo LOAD_INFO:$load_info
echo -e "\033[32m----------------------------------------\033[0m"
echo -e -n "\033[36mYou want to write the data to the databases? \033[1m" ;
read ensure
if [ "$ensure" == "yes" -o "$ensure" == "y" -o "$ensure" == "Y" ]; then
    echo "----------------------------------------"
    echo -e '\033[31mmysql -uaudit -p123456 -D audit -e "' "insert into audit_system values('','${ip_info}','$serv_info','${cpu_info1} X${cpu_info2}','$disk_info','$mem_info','$load_info','$mark_info')" '" \033[0m '
    mysql -uroot -p123456 -D test -e "insert into audit_system
```

```
values('','${ip_info}','$serv_info','${cpu_info1}
X${cpu_info2}','$disk_info','$mem_info','$load_info','$mark_info')"
else
    echo "Please wait，exit....."
    exit
fi
```

手动读取数据库服务器信息命令：

```
mysql -uroot -p123 -e 'use wugk1 ; select * from audit_audit_system; '|sed
's/-//g'|grep -v "id"
```

16.3 Shell 编程实战拒绝恶意 IP 登录脚本

企业服务器暴露在外网，每天会有大量的人使用各种用户名和密码尝试登陆服务器，如果让其一直尝试，难免会猜出密码，通过开发 Shell 脚本，可以自动将尝试登陆服务器错误密码次数的 IP 列表加入到防火墙配置中。

Shell 脚本实现服务器拒绝恶意 IP 登陆，编写思路如下：

- 登陆服务器日志/var/log/secure；
- 检查日志中认证失败的行并打印其 IP 地址；
- 将 IP 地址写入至防火墙；
- 禁止该 IP 访问服务器 SSH 22 端口；
- 将脚本加入 Crontab 实现自动禁止恶意 IP；

Shell 脚本实现服务器拒绝恶意 IP 登陆，代码如下：

```
#!/bin/bash
```

```
#Auto drop ssh failed IP address
#By author jfedu.net 2017
#Define Path variables
SEC_FILE=/var/log/secure
IP_ADDR=`awk '{print $0}' /var/log/secure|grep -i "fail"| egrep
-o "([0-9]{1,3}\.){3}[0-9]{1,3}" | sort -nr | uniq -c |awk '$1>=15 {print $2}'`
IPTABLE_CONF=/etc/sysconfig/iptables
echo
cat <<EOF
++++++++++++++welcome to use ssh login drop failed
ip++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++
++++++++++++++++-------------------------------------------+++++++++++++
+++++
EOF
echo
for ((j=0; j<=6; j++)) ; do echo -n "-"; sleep 1 ; done
echo
for i in `echo $IP_ADDR`
do
   cat $IPTABLE_CONF |grep $i >/dev/null
```

```
if
   [ $? -ne 0 ]；then
    sed -i "/lo/a -A INPUT -s $i -m state --state NEW -m tcp -p tcp --dport 22 -j
DROP" $IPTABLE_CONF
fi
done
NUM=`find /etc/sysconfig/   -name iptables -a -mmin -1|wc -l`
          if [ $NUM -eq 1 ]；then
                    /etc/init.d/iptables restart
          fi
```

16.4 Shell 编程实战 LAMP 一键安装脚本

LAMP 是目前互联网主流 WEB 网站架构，通过源码安装、维护和管理对于单台很轻松，如果服务器数量多，手工管理就非常困难，基于 Shell 脚本可以更快速的维护 LAMP 架构。

Shell 脚本实现服务器 LAMP 一键源码安装配置，编写思路如下：

- 脚本的功能，实现安装 LAMP 环境、论坛网站；
- Apache 安装配置、MYSQL、PHP 安装；
- 源码 LAMP 整合配置；
- 启动数据库，创建数据库并授权；
- 重启 LAMP 所有服务， 验证访问；

Shell 脚本实现服务器 LAMP 一键源码安装配置，代码如下：

```
#!/bin/bash
```

```
#Auto install LAMP

#By author jfedu.net 2017

#Define Path variables

#Httpd define path variable

H_FILES=httpd-2.2.32.tar.bz2

H_FILES_DIR=httpd-2.2.32

H_URL=http://mirrors.cnnic.cn/apache/httpd/

H_PREFIX=/usr/local/apache2/

#MySQL define path variable

M_FILES=mysql-5.5.20.tar.gz

M_FILES_DIR=mysql-5.5.20

M_URL=http://down1.chinaunix.net/distfiles/

M_PREFIX=/usr/local/mysql/

#PHP define path variable

P_FILES=php-5.3.28.tar.bz2

P_FILES_DIR=php-5.3.28

P_URL=http://mirrors.sohu.com/php/

P_PREFIX=/usr/local/php5/

function httpd_install(){

if [[ "$1" -eq "1" ]]; then

      wget -c $H_URL/$H_FILES &&    tar -jxvf $H_FILES && cd $H_FILES_DIR

&&./configure --prefix=$H_PREFIX
```

```
    if [ $? -eq 0 ];  then
        make && make install
    fi
fi
}
function mysql_install(){
if [[ "$1" -eq "2" ]];  then
wget -c $M_URL/$M_FILES &&   tar -xzvf $M_FILES && cd $M_FILES_DIR
&&yum install cmake ncurses-devel -y  ;  cmake .
-DCMAKE_INSTALL_PREFIX=$M_PREFIX \
-DMYSQL_UNIX_ADDR=/tmp/mysql.sock \
-DMYSQL_DATADIR=/data/mysql \
-DSYSCONFDIR=/etc \
-DMYSQL_USER=mysql \
-DMYSQL_TCP_PORT=3306 \
-DWITH_XTRADB_STORAGE_ENGINE=1 \
-DWITH_INNOBASE_STORAGE_ENGINE=1 \
-DWITH_PARTITION_STORAGE_ENGINE=1 \
-DWITH_BLACKHOLE_STORAGE_ENGINE=1 \
-DWITH_MYISAM_STORAGE_ENGINE=1 \
-DWITH_READLINE=1 \
-DENABLED_LOCAL_INFILE=1 \
```

```
-DWITH_EXTRA_CHARSETS=1 \

-DDEFAULT_CHARSET=utf8 \

-DDEFAULT_COLLATION=utf8_general_ci \

-DEXTRA_CHARSETS=all \

-DWITH_BIG_TABLES=1 \

-DWITH_DEBUG=0

if [ $? -eq 0 ];  then

      make && make install

      echo -e "\n\033[32m-----------------------------------------------\033[0m"

                  echo -e "\033[32mThe $M_FILES_DIR Server Install

Success !\033[0m"

            else

                  echo -e "\033[32mThe $M_FILES_DIR Make or Make install

ERROR,Please Check....."

                  exit 0

fi

/bin/cp support-files/my-small.cnf    /etc/my.cnf

/bin/cp support-files/mysql.server /etc/init.d/mysqld

chmod +x /etc/init.d/mysqld

chkconfig --add mysqld

chkconfig mysqld on

fi
```

```
}
function php_install(){
if [[ "$1" -eq "3" ]]; then
        yum install   libxml2-devel   perl-devel   perl   libtool* -y
        wget -c $P_URL/$P_FILES &&   tar -jxvf $P_FILES && cd $P_FILES_DIR
&&./configure --prefix=$P_PREFIX --with-config-file-path=$P_PREFIX/etc
--with-mysql=$M_PREFIX --with-apxs2=$H_PREFIX/bin/apxs
        if [ $? -eq 0 ]; then
                make ZEND_EXTRA_LIBS='-liconv' && make install
                echo -e
"\n\033[32m-----------------------------------------\033[0m"
                echo -e "\033[32mThe $P_FILES_DIR Server Install
Success !\033[0m"
        else
                echo -e "\033[32mThe $P_FILES_DIR Make or Make install
ERROR,Please Check....."
                exit 0
        fi
fi
}
function lamp_config(){
if [[ "$1" -eq "4" ]]; then
```

```
    sed -i '/DirectoryIndex/s/index.html/index.php index.html/g'
$H_PREFIX/conf/httpd.conf
    $H_PREFIX/bin/apachectl restart
    echo "AddType
application/x-httpd-php .php" >>$H_PREFIX/conf/httpd.conf
    IP=`ifconfig eth0|grep "Bcast"|awk '{print $2}'|cut -d: -f2`
    echo "You can to access http://$IP/"

cat >$H_PREFIX/htdocs/index.php<<EOF
<?php
phpinfo();
?>
EOF
fi
}
PS3="Please enter you select install menu:"
select i in http mysql php config quit
do

case $i in
    http)
    httpd_install 1
```

```
    ;;
    mysql)
    mysql_install 2
    ;;
    php)
    php_install 3
    ;;
    config)
    lamp_config 4
    ;;
    quit)
    exit
esac
done
```

16.5 Shell 编程实战 MYSQL 主从复制脚本

MYSQL 数据库服务器应用主要应用于与动态网站结合，存放网站必要的数据，例如订单、交易、员工表、薪资等记录，为了实现数据备份，需引入 MYSQL 主从架构，MYSQL 主从架构脚本可以实现自动化安装、配置和管理。

Shell 脚本实现服务器 MYSQL 一键 YUM 安装配置，编写思路如下：

MYSQL 主库的操作：

- 主库上安装 MYSQL,设置 server-id、bin-log；
- 授权复制同步的用户，对客户端授权；

- 确认 bin-log 文件名、position 位置点。

MYSQL 丛库的操作：

- 从库上安装 MYSQL,设置 server-id；
- change master 指定主库和 bin-log 名和 position；
- start slave； 启动丛库 IO 线程；
- show slave status\G 查看主从的状态。

Shell 脚本实现服务器 MYSQL 一键 YUM 安装配置，需要提前手动授权主库可以免密码登录丛库服务器，代码如下：

```
#!/bin/bash
#Auto install Mysql AB Repliation
#By author jfedu.net 2017
#Define Path variables
MYSQL_SOFT="mysql mysql-server mysql-devel php-mysql mysql-libs"
NUM=`rpm -qa |grep -i mysql |wc -l`
INIT="/etc/init.d/mysqld"
CODE=$?
#Mysql To Install 2017
if [ $NUM -ne 0 -a -f $INIT ]; then
    echo -e "\033[32mThis Server Mysql already Install.\033[0m"
    read -p "Please ensure yum remove Mysql Server,YES or NO": INPUT
    if [ $INPUT == "y" -o $INPUT == "yes" ]; then
        yum remove $MYSQL_SOFT -y ; rm -rf /var/lib/mysql /etc/my.cnf
```

```
        yum install $MYSQL_SOFT -y
    else
        echo
    fi
else
    yum remove $MYSQL_SOFT -y ； rm -rf /var/lib/mysql /etc/my.cnf
    yum install $MYSQL_SOFT -y
    if [ $CODE -eq 0 ]； then
        echo -e "\033[32mThe Mysql Install Successfully.\033[0m"
    else
        echo -e "\033[32mThe Mysql Install Failed.\033[0m"
        exit 1
    fi
fi
my_config(){
cat >/etc/my.cnf<<EOF
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
symbolic-links=0
log-bin=mysql-bin
```

```
server-id = 1

auto_increment_offset=1

auto_increment_increment=2

[mysqld_safe]

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

EOF

}

my_config

/etc/init.d/mysqld restart

ps -ef |grep mysql

MYSQL_CONFIG(){

#Master Config Mysql

mysql -e "grant  replication  slave  on *.* to  'tongbu'@'%'  identified by

'123456';  "

MASTER_FILE=`mysql -e "show master status;  "|tail -1|awk '{print $1}'`

MASTER_POS=`mysql -e "show master status;  "|tail -1|awk '{print $2}'`

MASTER_IPADDR=`ifconfig eth0|grep "Bcast"|awk '{print $2}'|cut -d: -f2`

read -p "Please Input Slave IPaddr: " SLAVE_IPADDR

#Slave Config Mysql

ssh -l root $SLAVE_IPADDR "yum remove $MYSQL_SOFT -y  ;  rm -rf

/var/lib/mysql /etc/my.cnf  ;  yum install $MYSQL_SOFT -y"
```

```
ssh -l root $SLAVE_IPADDR "$my_config"

#scp -r /etc/my.cnf root@192.168.111.129:/etc/

ssh -l root $SLAVE_IPADDR "sed -i 's#server-id = 1#server-id = 2#g'

/etc/my.cnf"

ssh -l root $SLAVE_IPADDR "sed -i '/log-bin=mysql-bin/d' /etc/my.cnf"

ssh -l root $SLAVE_IPADDR "/etc/init.d/mysqld restart"

ssh -l root $SLAVE_IPADDR "mysql -e \"change master to

master_host='$MASTER_IPADDR',master_user='tongbu',master_password='123

456',master_log_file='$MASTER_FILE',master_log_pos=$MASTER_POS;  \""

ssh -l root $SLAVE_IPADDR "mysql -e \"slave start;  \""

ssh -l root $SLAVE_IPADDR "mysql -e \"show slave status\G;  \""

}


read -p "Please ensure your Server is Master and you will config mysql

Replication?yes or no": INPUT

if [ $INPUT == "y" -o $INPUT == "yes" ];  then

      MYSQL_CONFIG

else

      exit 0

fi
```

16.6 Shell 编程实战修改 IP 及主机名脚本

企业中服务器 IP 地址系统通过自动化工具安装完系统，IP 均是自动获取的，而服务器

要求固定的静态 IP，百台服务器手工去配置静态 IP 是不可取的，可以基于 Shell 脚本自动修改 IP、主机名等信息。

Shell 脚本实现服务器 IP、主机名自动修改及配置，编写思路如下：

- 静态 IP 修改；
- 动态 IP 修改；
- 根据 IP-生成主机名并配置；
- 修改 DNS 域名解析；

Shell 脚本实现服务器 IP、主机名自动修改及配置，代码如下：

```
#!/bin/bash
#Auto Change ip netmask gateway scripts
#By author jfedu.net 2017
#Define Path variables
ETHCONF=/etc/sysconfig/network-scripts/ifcfg-eth0
HOSTS=/etc/hosts
NETWORK=/etc/sysconfig/network
DIR=/data/backup/`date +%Y%m%d`
NETMASK=255.255.255.0
echo "----------------------------"
judge_ip(){
      read -p "Please enter ip Address,example 192.168.0.11 ip": IPADDR
      echo $IPADDR|grep -v "[Aa-Zz]"|grep --color -E "([0-9]{1,3}\.){3}[0-9]{1,3}"
}
```

```
count_ip(){
    count=(`echo $IPADDR|awk -F. '{print $1,$2,$3,$4}'`)
    IP1=${count[0]}
    IP2=${count[1]}
    IP3=${count[2]}
    IP4=${count[3]}
}
ip_check()
{
judge_ip
while [ $? -ne 0 ]
do
    judge_ip
done
count_ip
while [ "$IP1" -lt 0 -o "$IP1" -ge 255 -o "$IP2" -ge 255 -o "$IP3" -ge 255 -o
"$IP4" -ge 255 ]
do
    judge_ip
    while [ $? -ne 0 ]
    do
        judge_ip
```

```
        done
        count_ip
done
}
change_ip()
{
if [ ! -d $DIR ];  then
        mkdir -p $DIR
fi
echo "The Change ip address to Backup Interface eth0"
cp $ETHCONF    $DIR
grep "dhcp"    $ETHCONF
if [ $? -eq 0 ];  then
        read -p "Please enter ip Address:" IPADDR
        sed -i 's/dhcp/static/g' $ETHCONF
        echo -e "IPADDR=$IPADDR\nNETMASK=$NETMASK\nGATEWAY=`echo
$IPADDR|awk -F. '{print $1"."$2"."$3}'`.2" >>$ETHCONF
        echo "The IP configuration success. !"
else
        echo -n    "Static IP has been configured,please confirm whether to
modify,yes or No":
        read i
```

```
fi

if   [ "$i" == "y" -o "$i" == "yes" ];  then

      ip_check

      sed -i -e '/IPADDR/d' -e '/NETMASK/d' -e '/GATEWAY/d' $ETHCONF

      echo -e "IPADDR=$IPADDR\nNETMASK=$NETMASK\nGATEWAY=`echo
$IPADDR|awk -F. '{print $1"."$2"."$3}'`.2" >>$ETHCONF

      echo "The IP configuration success. !"

      echo

else

      echo "Static IP already exists,please exit."

      exit $?

fi

}

change_hosts()

{



if [ ! -d $DIR ];  then

      mkdir -p $DIR

fi

cp $HOSTS $DIR

ip_check

host=` echo $IPADDR|sed 's/\./-/g'|awk '{print "BJ-IDC-"$0"-jfedu.net"}'`
```

```
cat $HOSTS |grep "$host"
if [ $? -ne 0 ];  then
      echo "$IPADDR       $host" >> $HOSTS
      echo "The hosts modify success "
fi
grep "$host" $NETWORK
if [ $? -ne 0 ];  then
      sed -i "s/^HOSTNAME/#HOSTNAME/g" $NETWORK
      echo "NETWORK=$host" >>$NETWORK
      hostname $host;  su
fi
}
PS3="Please Select configuration ip or configuration host:"
select i in    "modify_ip" "modify_hosts" "exit"
do
      case $i in
                  modify_ip)
                  change_ip
            ;  ;
                  modify_hosts)
                  change_hosts
            ;  ;
```

```
                exit)
                exit
        ;;
                *)
                echo -e "1) modify_ip\n2) modify_ip\n3)exit"
    esac

done
```

16.7 Shell 编程实战 Zabbix 配置脚本

Zabbix 是一款分布式监控系统，基于 C/S 模式，需在服务器安装 Zabbix_server，在客户端安装 Zabbix_agent，通过 Shell 脚本可以更快速的实现该需求。

Shell 脚本实现 Zabbix 服务器端和客户端自动安装，编写思路如下：

- Zabbix 软件的版本源码安装、路径、--enable-server、--enable-agent；
- cp zabbix_agentd 启动进程-/etc/init.d/zabbix_agentd、给执行 x 权限；
- 配置 zabbix_agentd.conf 文件，指定 server IP 变量；
- 指定客户端的 Hostname 其实可以等于客户端 IP 地址；
- 启动 zabbix_agentd 服务，创建 zabbix user。

Shell 脚本实现 Zabbix 服务器端和客户端自动安装，代码如下：

```
#!/bin/bash
#Auto install zabbix server and client
#By author jfedu.net 2017
```

```
#Define Path variables

ZABBIX_SOFT="zabbix-3.2.6.tar.gz"

INSTALL_DIR="/usr/local/zabbix/"

SERVER_IP="192.168.111.128"

IP=`ifconfig|grep Bcast|awk '{print $2}'|sed 's/addr://g'`

SERVER_INSTALL(){

yum -y install curl curl-devel net-snmp net-snmp-devel perl-DBI

groupadd zabbix ; useradd -g zabbix zabbix; usermod -s /sbin/nologin zabbix

tar -xzf $ZABBIX_SOFT; cd `echo $ZABBIX_SOFT|sed 's/.tar.*//g'`

./configure  --prefix=/usr/local/zabbix --enable-server --enable-agent

--with-mysql --enable-ipv6 --with-net-snmp --with-libcurl &&make install

if [ $? -eq 0 ]; then

    ln -s /usr/local/zabbix/sbin/zabbix_* /usr/local/sbin/

fi

cd - ; cd zabbix-3.2.6

cp  misc/init.d/tru64/{zabbix_agentd,zabbix_server}  /etc/init.d/ ; chmod o+x

/etc/init.d/zabbix_*

mkdir -p /var/www/html/zabbix/; cp -a  frontends/php/*

/var/www/html/zabbix/

#config zabbix server

cat >$INSTALL_DIR/etc/zabbix_server.conf<<EOF

LogFile=/tmp/zabbix_server.log
```

```
DBHost=localhost

DBName=zabbix

DBUser=zabbix

DBPassword=123456

EOF

#config zabbix agentd

cat >$INSTALL_DIR/etc/zabbix_agentd.conf<<EOF

LogFile=/tmp/zabbix_agentd.log

Server=$SERVER_IP

ServerActive=$SERVER_IP

Hostname = $IP

EOF

#start zabbix agentd

/etc/init.d/zabbix_server restart

/etc/init.d/zabbix_agentd restart

/etc/init.d/iptables stop

setenforce 0

}

AGENT_INSTALL(){

yum -y install curl curl-devel net-snmp net-snmp-devel perl-DBI

groupadd zabbix ； useradd -g zabbix zabbix；usermod -s /sbin/nologin zabbix
```

```
tar -xzf $ZABBIX_SOFT;  cd `echo $ZABBIX_SOFT|sed 's/.tar.*//g'`

./configure   --prefix=/usr/local/zabbix   --enable-agent&&make install

if [ $? -eq 0 ];  then

      ln -s /usr/local/zabbix/sbin/zabbix_* /usr/local/sbin/

fi

cd -  ;  cd zabbix-3.2.6

cp   misc/init.d/tru64/zabbix_agentd   /etc/init.d/zabbix_agentd  ;  chmod o+x

/etc/init.d/zabbix_agentd

#config zabbix agentd

cat >$INSTALL_DIR/etc/zabbix_agentd.conf<<EOF

LogFile=/tmp/zabbix_agentd.log

Server=$SERVER_IP

ServerActive=$SERVER_IP

Hostname = $IP

EOF

#start zabbix agentd

/etc/init.d/zabbix_agentd restart

/etc/init.d/iptables stop

setenforce 0

}



read -p "Please confirm whether to install Zabbix Server,yes or no? " INPUT
```

```
if [ $INPUT == "yes" -o $INPUT == "y" ]；then
    SERVER_INSTALL
else
    AGENT_INSTALL
fi
```

16.8 Shell 编程实战 Nginx 虚拟主机脚本

Nginx WEB 服务器的最大特点在于 Nginx 常被用于负载均衡、反向代理，单台 Nginx 服务器配置多个虚拟主机，百台服务器配置 N 多虚拟主机，基于 Shell 脚本可以更加高效的配置虚拟主机及添加、管理。

Shell 脚本实现 Nginx 自动安装及虚拟主机的维护，编写思路如下：

- 脚本指定参数 v1.jfedu.net；
- 创建 v1.jfedu.net 同时创建目录/var/www/v1；
- 将 Nginx 虚拟主机配置定向到新的目录；
- 重复虚拟主机不再添加。

Shell 脚本实现 Nginx 自动安装及虚拟主机的维护，代码如下：

```
#!/bin/bash
#Auto config Nginx virtual Hosts
#By author jfedu.net 2017
#Define Path variables
NGINX_CONF="/usr/local/nginx/conf/"
NGINX_MAKE="--user=www --group=www --prefix=/usr/local/nginx
```

```
--with-http_stub_status_module --with-http_ssl_module"

NGINX_SBIN="/usr/local/nginx/sbin/nginx"

NGINX_INSTALL(){

#Install Nginx server

NGINX_FILE=nginx-1.12.0.tar.gz

NGINX_DIR=`echo $NGINX_FILE|sed 's/.tar*.*//g'`

if [ ! -e /usr/local/nginx/ -a ! -e /etc/nginx/ ]; then

    pkill nginx

    wget -c http://nginx.org/download/$NGINX_FILE

    yum install pcre-devel pcre -y

    rm -rf $NGINX_DIR  ;  tar xf $NGINX_FILE

    cd $NGINX_DIR;  useradd www;  ./configure $NGINX_MAKE

    make &&make install

    grep -vE "#|^$" $NGINX_CONF/nginx.conf >$NGINX_CONF/nginx.conf.swp

    \mv    $NGINX_CONF/nginx.conf.swp $NGINX_CONF/nginx.conf

    for i in `seq 1 6`;  do sed -i '$d' $NGINX_CONF/nginx.conf;  done

    echo "}" >>$NGINX_CONF/nginx.conf

    cd ../

fi

}

NGINX_CONFIG(){

#config tomcat nginx vhosts
```

```
grep "include domains" $NGINX_CONF/nginx.conf >>/dev/null
if [ $? -ne 0 ];  then
      #sed -i '$d' $NGINX_CONF/nginx.conf
      echo -e "\ninclude domains/*;  \n}" >>$NGINX_CONF/nginx.conf
      mkdir -p $NGINX_CONF/domains/
fi
VHOSTS=$1
ls $NGINX_CONF/domains/$VHOSTS>>/dev/null 2>&1
if [ $? -ne 0 ];  then
      #cp -r xxx.jfedu.net $NGINX_CONF/domains/$VHOSTS
      #sed -i "s/xxx/$VHOSTS/g" $NGINX_CONF/domains/$VHOSTS
      cat>$NGINX_CONF/domains/$VHOSTS<<EOF
      #vhost server $VHOSTS
      server {
            listen          80;
            server_name    $VHOSTS;
            location / {
                  root     /data/www/$VHOSTS/;
                  index    index.html index.htm;
            }
        }
EOF
```

```
    mkdir -p /data/www/$VHOSTS/
    cat>/data/www/$VHOSTS/index.html<<EOF
    <html>
    <h1><center>The First Test Nginx page.</center></h1>
    <hr color="red">
    <h2><center>$VHOSTS</center></h2>
    </html>
EOF
    echo -e "\033[32mThe $VHOSTS Config success,You can to access
http://$VHOSTS/\033[0m"
    NUM=`ps -ef |grep nginx|grep -v grep|grep -v auto|wc -l`
    $NGINX_SBIN -t >>/dev/null 2>&1
    if [ $? -eq 0 -a $NUM -eq 0 ];  then
        $NGINX_SBIN
    else
        $NGINX_SBIN -t >>/dev/null 2>&1
        if [ $? -eq 0 ];  then
            $NGINX_SBIN -s reload
        fi
    fi
else
    echo -e "\033[32mThe $VHOSTS has been config,Please exit.\033[0m"
```

```
fi

}

if [ -z $1 ]；then

      echo -e "\033[32m--------------------\033[0m"

      echo -e "\033[32mPlease enter sh $0 xx.jf.com.\033[0m"

      exit 0

fi

NGINX_INSTALL

NGINX_CONFIG $1
```

16.9 Shell 编程实战 Nginx+Tomcat 脚本

Tomcat用于发布JSP WEB页面，根据企业实际需求，会在单台服务器配置N个Tomcat实例，同时手动将 Tomcat 创建后的实例加入至 Nginx 虚拟主机中，同时重启 Nginx，开发 Nginx、Tomcat 自动创建 Tomcat 实例及 Nginx 虚拟主机管理脚本能让大大的减轻人工的干预，实现快速的交付。

Shell 脚本实现 Nginx 自动安装、虚拟主机及自动将 Tomcat 加入至虚拟主机，编写思路如下：

- 手动拷贝 Tomcat 与脚本一致目录（可自动修改）；
- 手动修改 Tomcat 端口为 6001、7001、8001（可自动修改）；
- 脚本指定参数 v1.jfedu.net；
- 创建 v1.jfedu.net Tomcat 实例；
- 修改 Tomcat 实例端口，保证 Port 唯一；
- 将 Tomcat 实例加入 Nginx 虚拟主机；

- 重复创建 Tomcat 实例，端口自动增加，并加入原 Nginx 虚拟主机，实现负载均衡；

Shell 脚本实现 Nginx 自动安装、虚拟主机及自动将 Tomcat 加入至虚拟主机，代码如下：

```
#!/bin/bash
#Auto config Nginx and tomcat cluster
#By author jfedu.net 2017
#Define Path variables
NGINX_CONF="/usr/local/nginx/conf/"
install_nginx(){
    NGINX_FILE=nginx-1.10.2.tar.gz
    NGINX_DIR=`echo $NGINX_FILE|sed 's/.tar*.*//g'`
    wget -c http://nginx.org/download/$NGINX_FILE
    yum install pcre-devel pcre -y
    rm -rf $NGINX_DIR ； tar xf $NGINX_FILE
    cd $NGINX_DIR；useradd www；./configure --user=www --group=www
--prefix=/usr/local/nginx2 --with-http_stub_status_module
--with-http_ssl_module
    make &&make install
    cd ../
}
install_tomcat(){
```

```
    JDK_FILE="jdk1.7.0_25.tar.gz"
    JDK_DIR=`echo $JDK_FILE|sed 's/.tar.*//g'`
    tar -xzf $JDK_FILE   ;  mkdir -p /usr/java/  ;  mv $JDK_DIR /usr/java/
    sed -i '/JAVA_HOME/d;  /JAVA_BIN/d;  /JAVA_OPTS/d' /etc/profile
    cat >> /etc/profile <<EOF
    export JAVA_HOME=/export/servers/$JAVA_DIR
    export JAVA_BIN=/export/servers/$JAVA_DIR/bin
    export PATH=\$JAVA_HOME/bin:\$PATH
    export CLASSPATH=.:\$JAVA_HOME/lib/dt.jar:\$JAVA_HOME/lib/tools.jar
    export JAVA_HOME JAVA_BIN PATH CLASSPATH
EOF
    source /etc/profile;  java -version
    #install tomcat start
    ls tomcat
}
config_tomcat_nginx(){
    #config tomcat nginx vhosts
    grep "include domains" $NGINX_CONF/nginx.conf >>/dev/null
    if [ $? -ne 0 ];  then
        sed -i '$d' $NGINX_CONF/nginx.conf
        echo -e "\ninclude domains/*;  \n}" >>$NGINX_CONF/nginx.conf
        mkdir -p $NGINX_CONF/domains/
```

```
    fi
    VHOSTS=$1
    NUM=`ls /usr/local/|grep -c tomcat`
    if [ $NUM -eq 0 ];  then
        cp -r tomcat /usr/local/tomcat_$VHOSTS
        cp -r xxx.jfedu.net $NGINX_CONF/domains/$VHOSTS
        #sed -i "s/VHOSTS/$VHOSTS/g" $NGINX_CONF/domains/$VHOSTS
        sed -i "s/xxx/$VHOSTS/g" $NGINX_CONF/domains/$VHOSTS
        exit 0
    fi
    #-------------------------------
    #VHOSTS=$1
    VHOSTS_NUM=`ls $NGINX_CONF/domains/|grep -c $VHOSTS`
    SERVER_NUM=`grep -c "127" $NGINX_CONF/domains/$VHOSTS`
    SERVER_NUM_1=`expr $SERVER_NUM + 1`
    rm -rf /tmp/.port.txt
    for i in `find /usr/local/ -maxdepth 1 -name "tomcat*"`;  do
        grep "port" $i/conf/server.xml |egrep -v "\--|8080|SSLEnabled"|awk
'{print $2}'|sed 's/port=//g;  s/\"//g'|sort -nr >>/tmp/.port.txt
    done
    MAX_PORT=`cat /tmp/.port.txt|grep -v 8443|sort -nr|head -1`
    PORT_1=`expr $MAX_PORT - 2000 + 1`
```

```
    PORT_2=`expr $MAX_PORT - 1000 + 1`
    PORT_3=`expr $MAX_PORT + 1`
    if [ $VHOSTS_NUM -eq 1 ];  then
        read -p "The $VHOSTS is exists,You sure create mulit Tomcat for the
$VHOSTS? yes or no " INPUT
        if [ $INPUT == "YES" -o $INPUT == "Y" -o $INPUT == "yes" ];  then
            cp -r tomcat /usr/local/tomcat_${VHOSTS}_${SERVER_NUM_1}
            sed -i "s/6001/$PORT_1/g"
/usr/local/tomcat_${VHOSTS}_${SERVER_NUM_1}/conf/server.xml
            sed -i "s/7001/$PORT_2/g"
/usr/local/tomcat_${VHOSTS}_${SERVER_NUM_1}/conf/server.xml
            sed -i "s/8001/$PORT_3/g"
/usr/local/tomcat_${VHOSTS}_${SERVER_NUM_1}/conf/server.xml
            sed -i "/^upstream/a        server 127.0.0.1:${PORT_2} weight=1
max_fails=2 fail_timeout=30s;  " $NGINX_CONF/domains/$VHOSTS
            exit 0
        fi
        exit
    fi
    cp -r tomcat /usr/local/tomcat_$VHOSTS
        cp -r xxx.jfedu.net $NGINX_CONF/domains/$VHOSTS
        sed -i "s/VHOSTS/$VHOSTS/g" $NGINX_CONF/domains/$VHOSTS
```

```
        sed -i "s/xxx/$VHOSTS/g" $NGINX_CONF/domains/$VHOSTS
    sed -i "s/7001/${PORT_2}/g" $NGINX_CONF/domains/$VHOSTS
    #######config tomcat
        sed -i "s/6001/$PORT_1/g"
/usr/local/tomcat_${VHOSTS}/conf/server.xml
        sed -i "s/7001/$PORT_2/g"
/usr/local/tomcat_${VHOSTS}/conf/server.xml
        sed -i "s/8001/$PORT_3/g"
/usr/local/tomcat_${VHOSTS}/conf/server.xml
}
if [ ! -d $NGINX_CONF -o ! -d   /usr/java/$JDK_DIR ]; then
    install_nginx
    install_tomcat
fi
config_tomcat_nginx $1
```

16.10 Shell 编程实战 Docker 管理脚本

Docker 虚拟化是目前主流的虚拟化解决方案，越来越多的企业在使用 Docker 轻量级虚拟化，构建、维护和管理 Docker 虚拟化平台是运维人员非常重要的一个环节，开发 Docker Shell 脚本可以在命令行界面快速管理和维护 Docker。

Shell 脚本实现 Docker 自动安装、自动导入镜像、创建虚拟机、指定 IP 地址、将创建的 Docker 虚拟机加入 Excel 存档或者加入 MYSQL 数据库，编写思路如下：

- 基于 CentOS6.5+或者 7.x YUM 安装 Docker；

- Docker 脚本参数指定 CPU、内存、硬盘容量；
- Docker 自动检测局域网 IP 并赋予 Docker 虚拟机；
- Docker 基于 pipework 指定 IP；
- 将创建的 Docker 虚拟机加入至 CSV（Excel）或者 MYSQL 库；

Shell 脚本实现 Docker 自动安装、自动导入镜像、创建虚拟机、指定 IP 地址、将创建的 Docker 虚拟机加入 CSV（Excel）存档或者加入 MYSQL 数据库，代码如下：

```
#!/bin/bash
#Auto install docker and Create VM
#By author jfedu.net 2017
#Define Path variables
IPADDR=`ifconfig|grep -E "\<inet\>"|awk '{print $2}'|grep "192.168"|head -1`
GATEWAY=`route -n|grep "UG"|awk '{print $2}'|grep "192.168"|head -1`
IPADDR_NET=`ifconfig|grep -E "\<inet\>"|awk '{print $2}'|grep "192.168"|head
-1|awk -F. '{print $1"."$2"."$3"."}'`
LIST="/root/docker_vmlist.csv"
if [ ! -f /usr/sbin/ifconfig ]; then
    yum install net-tools* -y
fi
for i in `seq 1 253`; do ping -c 1 ${IPADDR_NET}${i} ; [ $? -ne 0 ]&&
DOCKER_IPADDR="${IPADDR_NET}${i}" &&break; done >>/dev/null 2>&1
echo "###################"
echo -e "Dynamic get docker IP,The Docker IP address\n\n$DOCKER_IPADDR"
```

```
NETWORK=(
    HWADDR=`ifconfig eth0|grep ether|awk '{print $2}'`
    IPADDR=`ifconfig eth0|grep -E "\<inet\>"|awk '{print $2}'`
    NETMASK=`ifconfig eth0|grep -E "\<inet\>"|awk '{print $4}'`
    GATEWAY=`route -n|grep "UG"|awk '{print $2}'`
)
if [ -z "$1" -o -z "$2" ];  then
    echo -e "\033[32m--------------------------------\033[0m"
    echo -e "\033[32mPlease exec $0 CPU(C) MEM(G),example $0 4 8\033[0m"
    exit 0
fi
#CPU=`expr $2 - 1`
if [ ! -e /usr/bin/bc ];  then
    yum install bc -y >>/dev/null 2>&1
fi
CPU_ALL=`cat /proc/cpuinfo |grep processor|wc -l`
if [ ! -f $LIST ];  then
    CPU_COUNT=$1
    CPU_1="0"
    CPU1=`expr $CPU_1 + 0`
    CPU2=`expr $CPU1 + $CPU_COUNT - 1`
    if [ $CPU2 -gt $CPU_ALL ];  then
```

```
            echo -e "\033[32mThe System CPU count is $CPU_ALL,not more than
it.\033[0m"
            exit
        fi
else
        CPU_COUNT=$1
        CPU_1=`cat $LIST|tail -1|awk -F"," '{print $4}'|awk -F"-" '{print $2}'`
        CPU1=`expr $CPU_1 + 1`
        CPU2=`expr $CPU1 + $CPU_COUNT - 1`
        if [ $CPU2 -gt $CPU_ALL ];  then
            echo -e "\033[32mThe System CPU count is $CPU_ALL,not more than
it.\033[0m"
            exit
        fi
fi
MEM_F=`echo $2 \* 1024|bc`
MEM=`printf "%.0f\n" $MEM_F`
DISK=20
USER=$3
REMARK=$4
ping $DOCKER_IPADDR -c 1 >>/dev/null 2>&1
if [ $? -eq 0 ];  then
```

```
        echo -e "\033[32m--------------------------------\033[0m"
        echo -e "\033[32mThe IP address to be used,Please change other
IP,exit.\033[0m"
        exit 0
fi
if [ ! -e /usr/bin/docker ];  then
        yum install docker* device-mapper*    -y
        mkdir -p /export/docker/
        cd /var/lib/  ;  rm -rf docker  ;  ln -s /export/docker/ .
        mkdir -p /var/lib/docker/devicemapper/devicemapper
        dd if=/dev/zero of=/var/lib/docker/devicemapper/devicemapper/data
bs=1G count=0 seek=2000
        service docker start
        if [ $? -ne 0 ];  then
                echo "Docker install error ,please check."
                exit
        fi
fi
cd    /etc/sysconfig/network-scripts/
        mkdir -p /data/backup/`date +%Y%m%d-%H%M`
        yes|cp ifcfg-eth* /data/backup/`date +%Y%m%d-%H%M`/
if
```

```
    [ -e /etc/sysconfig/network-scripts/ifcfg-br0 ]；then
    echo
else
    cat >ifcfg-eth0 <<EOF
    DEVICE=eth0
    BOOTPROTO=none
    ${NETWORK[0]}
    NM_CONTROLLED=no
    ONBOOT=yes
    TYPE=Ethernet
    BRIDGE="br0"
    ${NETWORK[1]}
    ${NETWORK[2]}
    ${NETWORK[3]}
    USERCTL=no
EOF
    cat >ifcfg-br0 <<EOF
    DEVICE="br0"
    BOOTPROTO=none
    ${NETWORK[0]}
    IPV6INIT=no
    NM_CONTROLLED=no
```

```
        ONBOOT=yes
        TYPE="Bridge"
        ${NETWORK[1]}
        ${NETWORK[2]}
        ${NETWORK[3]}
        USERCTL=no
EOF
        /etc/init.d/network restart
fi
echo 'Your can restart Ethernet Service: /etc/init.d/network restart !'
echo '------------------------------------------------------'

cd -
#######create docker container
service docker status >>/dev/null
if [ $? -ne 0 ];  then
        service docker restart
fi
NAME="Docker_`echo $DOCKER_IPADDR|awk -F"." '{print $(NF-1)"_"$NF}'`"
IMAGES=`docker images|grep -v "REPOSITORY"|grep -v "none"|grep
"jfedu"|head -1|awk '{print $1}'`
if [ -z $IMAGES ];  then
```

```
    echo "Plesae Download Docker Centos Images,you can to be use docker
search centos,and docker pull centos6.5-ssh,exit 0"
    if [ ! -f jfedu_centos68.tar ]; then
        echo "Please upload jfedu_centos68.tar for docker server."
        exit
    fi
    cat jfedu_centos68.tar|docker import - jfedu_centos6.8
fi
IMAGES=`docker images|grep -v "REPOSITORY"|grep -v "none"|grep
"jfedu"|head -1|awk '{print $1}'`
CID=$(docker run -itd --privileged --cpuset-cpus=${CPU1}-${CPU2} -m
${MEM}m --net=none --name=$NAME $IMAGES /bin/bash)
echo $CID
docker ps -a |grep "$NAME"
pipework br0 $NAME   $DOCKER_IPADDR/24@$IPADDR
docker exec $NAME /etc/init.d/sshd start
if [ ! -e $LIST ]; then
    echo "编号,容器 ID,容器名称,CPU,内存,硬盘,容器 IP,宿主机 IP,使用人,备注
" >$LIST
fi
####################
NUM=`cat $LIST |grep -v CPU|tail -1|awk -F, '{print $1}'`
```

```
if [[ $NUM -eq "" ]]；then
        NUM="1"
else
        NUM=`expr $NUM + 1`
fi
##################
echo -e "\033[32mCreate virtual client Successfully.\n$NUM `echo $CID|cut -b
1-12`,$NAME,$CPU1-$CPU2,${MEM}M,${DISK}G,$DOCKER_IPADDR,$IPADDR,$
USER,$REMARK\033[0m"
if [ -z $USER ]；then
    USER="NULL"
    REMARK="NULL"
fi
echo $NUM,`echo $CID|cut -b
1-12`,$NAME,$CPU1-$CPU2,${MEM}M,${DISK}G,$DOCKER_IPADDR,$IPADDR,$
USER,$REMARK >>$LIST
rm -rf /root/docker_vmlist_*
iconv -c -f utf-8 -t gb2312 $LIST   -o /root/docker_vmlist_`date +%H%M`.csv
```

16.11 Shell 编程实战 Bind 管理脚本

Bind 主要应用于企业 DNS 构建平台，而 DNS 用于将域名与 IP 进行解析，用户在浏览器只需输入域名，即可访问服务器 IP 地址的虚拟主机网站。

Bind 难点在于创建各种记录，例如 A 记录、mail 记录、反向记录、资源记录，基于

Shell 脚本可以减轻人工的操作，节省大量的时间成本。

Shell 脚本实现 Bind 自动安装、初始化 Bind 环境、自动添加 A 记录、反向记录、批量添加 A 记录，编写思路如下：

- YUM 方式自动安装 Bind；
- 自动初始化 Bind 配置；
- 创建安装、初始化、添加记录函数；
- 自动添加单个 A 记录及批量添加 A 记录和反向记录；

Shell 脚本实现 Bind 自动安装、初始化 Bind 环境、自动添加 A 记录、反向记录、批量添加 A 记录，代码如下：

```
#!/bin/bash
#Auto install config bind server
#By author jfedu.net 2017
#Define Path variables
BND_ETC=/var/named/chroot/etc
BND_VAR=/var/named/chroot/var/named
BAK_DIR=/data/backup/dns_`date +%Y%m%d-%H%M`
##Backup named server
if
        [ ! -d    $BAK_DIR ];  then
        echo "Please waiting    Backup Named Config ....."
        mkdir     -p    $BAK_DIR
        cp -a    /var/named/chroot/{etc,var}     $BAK_DIR
```

```
        cp -a    /etc/named.* $BAK_DIR
fi
##Define Shell Install Function
Install ()
{
   if
        [ ! -e /etc/init.d/named ];  then
        yum install bind* -y
else
        echo ----------------------------------------------
        echo "The Named Server is exists ,Please exit ....."
        sleep 1
 fi
}
##Define Shell Init Function
Init_Config ()
{
          sed    -i -e 's/localhost;  /any;  /g' -e '/port/s/127.0.0.1/any/g'
/etc/named.conf
          echo ----------------------------------------------
          sleep 2
          echo "The named.conf config Init success !"
```

```
}

##Define Shell Add Name Function

Add_named ()

{

##DNS name

        read -p    "Please    Insert Into Your Add Name ,Example 51cto.com :"

NAME

        echo $NAME |grep -E "com|cn|net|org"

        while

         [ "$?" -ne 0 ]

           do

         read -p    "Please    reInsert Into Your Add Name ,Example 51cto.com :"

NAME

         echo $NAME |grep -E "com|cn|net|org"

      done

## IP address

        read -p    "Please    Insert Into Your Name Server IP ADDress:" IP

        echo $IP |egrep -o "([0-9]{1,3}\.){3}[0-9]{1,3}"

        while

        [ "$?" -ne "0" ]

         do

         read -p    "Please    reInsert Into Your Name Server IP ADDress:" IP
```

```
        echo $IP |egrep -o "([0-9]{1,3}\.){3}[0-9]{1,3}"
      done
        ARPA_IP=`echo $IP|awk -F. '{print $3"."$2"."$1}'`
        ARPA_IP1=`echo $IP|awk -F. '{print $4}'`
        cd  $BND_ETC
        grep  "$NAME" named.rfc1912.zones
if
          [ $? -eq 0 ]; then
          echo "The $NAME IS exist named.rfc1912.zones conf ,please exit ..."
          exit
else
        read -p  "Please  Insert Into SLAVE Name Server IP ADDress:" SLAVE
        echo $SLAVE |egrep -o "([0-9]{1,3}\.){3}[0-9]{1,3}"
        while
        [ "$?" -ne "0" ]
        do
            read -p  "Please  Insert Into SLAVE Name Server IP ADDress:"
SLAVE
            echo $SLAVE |egrep -o "([0-9]{1,3}\.){3}[0-9]{1,3}"
        done
            grep  "rev" named.rfc1912.zones
      if
```

```
            [ $? -ne 0 ];  then
                cat >>named.rfc1912.zones <<EOF
#`date +%Y-%m-%d` Add $NAME CONFIG
zone "$NAME" IN {
            type master;
            file "$NAME.zone";
            allow-update { none;    };
};
zone "$ARPA_IP.in-addr.arpa" IN {
            type master;
            file "$ARPA_IP.rev";
            allow-update { none;    };
};
EOF
        else
            cat >>named.rfc1912.zones <<EOF

#`date +%Y-%m-%d` Add $NAME CONFIG
zone "$NAME" IN {
            type master;
            file "$NAME.zone";
            allow-update { none;    };
```

```
};
EOF
    fi
fi
        [ $? -eq 0 ]&& echo "The $NAME config name.rfc1912.zones success !"
        sleep 3  ;  echo "Please waiting config $NAME zone File ....."
        cd   $BND_VAR
        read -p "Please insert Name DNS A HOST ,EXample   www or mail :"
HOST
        read -p "Please insert Name DNS A NS IP ADDR ,EXample
192.168.111.130 :" IP_HOST
        echo $IP_HOST |egrep -o "([0-9]{1,3}\.){3}[0-9]{1,3}"
        ARPA_IP2=`echo $IP_HOST|awk -F. '{print $3"."$2"."$1}'`
        ARPA_IP3=`echo $IP_HOST|awk -F. '{print $4}'`
        while
        [ "$?" -ne "0" ]
do
        read -p "Please Reinsert Name DNS A IPADDRESS ,EXample
192.168.111.130 :" IP_HOST
        echo $IP_HOST |egrep -o "([0-9]{1,3}\.){3}[0-9]{1,3}"
done
        cat >$NAME.zone <<EOF
```

```
\$TTL    86400
@               IN SOA   localhost.      root.localhost. (
                                   43              ;   serial (d. adams)
                                   1H              ;   refresh
                                   15M             ;   retry
                                   1W              ;   expiry
                                   1D )            ;   minimum

              IN   NS           $NAME.
EOF
      REV=`ls   *.rev`
      ls   *.rev >>/dev/null
if
      [ $? -ne 0 ]; then
      cat >>$ARPA_IP.rev <<EOF
\$TTL    86400
@       IN      SOA     localhost.     root.localhost.   (
                                  1997022703  ;   Serial
                                  28800       ;   Refresh
                                  14400       ;   Retry
                                  3600000     ;   Expire
                                  86400 )     ;   Minimum
```

```
                IN    NS    $NAME.
EOF
        echo    "$HOST                   IN    A
$IP_HOST" >>$NAME.zone
        echo    "$ARPA_IP3               IN    PTR
$HOST.$NAME." >>$ARPA_IP.rev
        [ $? -eq 0 ]&& echo -e "The $NAME config success:\n$HOST          IN
A               $IP_HOST\n$ARPA_IP3             IN    PTR             $HOST.$NAME."
else
        sed -i    "9a IN    NS    $NAME." $REV
        echo    "$HOST                   IN    A
$IP_HOST" >>$NAME.zone
        echo    "$ARPA_IP3               IN    PTR             $HOST.$NAME." >>$REV
        [ $? -eq 0 ]&& echo -e "The $NAME config success1:\n$HOST          IN
A               $IP_HOST\n$ARPA_IP3             IN    PTR             $HOST.$NAME."
fi
}
##Define Shell List A Function
Add_A_List ()
{
if
```

```
        cd   $BND_VAR
        REV=`ls   *.rev`
        read -p   "Please   Insert Into Your Add Name ,Example 51cto.com :"
NAME
        [ ! -e "$NAME.zone" ];  then
        echo "The $NAME.zone File is not exist ,Please ADD $NAME.zone File :"
        Add_named  ;
else
        read -p "Please Enter List Name A NS File ,Example /tmp/name_list.txt: "
FILE
    if
        [ -e $FILE ];  then
        for i in   `cat $FILE|awk '{print $2}'|sed "s/$NAME//g"|sed 's/\.$//g'`
        #for i in   `cat $FILE|awk '{print $1}'|sed "s/$NAME//g"|sed 's/\.$//g'`
do
        j=`awk -v I="$i.$NAME" '{if(I==$2)print $1}' $FILE`
        echo ------------------------------------------------------------
        echo "The $NAME.zone File is exist ,Please Enter insert NAME HOST ...."
        sleep 1
        ARPA_IP=`echo $j|awk -F. '{print $3"."$2"."$1}'`
        ARPA_IP2=`echo $j|awk -F. '{print $4}'`
        echo   "$i                IN   A              $j" >>$NAME.zone
```

```
        echo    "$ARPA_IP2          IN    PTR          $i.$NAME." >>$REV
        [ $? -eq 0 ]&& echo -e "The $NAME config success:\n$i          IN    A
$j\n$ARPA_IP2              IN    PTR              $i.$NAME."
done
      else
        echo "The $FILE List File IS Not Exist .....,Please exit ..."
      fi
fi
}
##Define Shell Select Menu
PS3="Please select Menu Name Config: "
select i in "自动安装 Bind 服务"  "自动初始化 Bind 配置" "添加解析域名"  "批量添
加 A 记录"
do
case    $i    in
        "自动安装 Bind 服务")
        Install
;;
        "自动初始化 Bind 配置")
        Init_Config
;;
        "添加解析域名")
```

```
        Add_named
;  ;
        "批量添加 A 记录")
        Add_A_List
;  ;
        * )
        echo ----------------------------------------------------
        sleep 1
        echo "Please exec: sh   $0   { Install(1)   or Init_Config(2) or
Add_named(3) or Add_config_A(4) }"
;  ;
esac
done
```

第17章　Linux 企业 WEB 高级篇

17.1　Nginx WEB 入门简介

Nginx [engine x]是 Igor Sysoev 编写的一个 HTTP 和反向代理服务器，另外它也可以作为邮件代理服务器。 它已经在众多流量很大的俄罗斯网站上使用了很长时间，这些网站包括 Yandex、Mail.Ru、VKontakte，以及 Rambler。

据 Netcraft 统计，在 2012 年 8 月份，世界上最繁忙的网站中有 11.48%使用 Nginx 作为其服务器或者代理服务器。目前互联网主流公司 360、百度、新浪、腾讯、阿里等都在

使用 nginx 作为自己的 web 服务器。

Nginx 由内核和模块组成，其中，**内核的设计非常微小和简洁，完成的工作也非常简单，仅仅通过查找配置文件将客户端请求映射到一个 location block (location 是 Nginx 配置中的一个指令，用于 URL 匹配），而在这个 location 中所配置的每个指令将会启动不同的模块去完成相应的工作。**

Nginx 相对于 Apache 优点：

1) 高并发响应性能非常好，官方 Nginx 处理静态文件并发 5w/s

2) 反向代理性能非常好。（可用于负载均衡）

3) 内存和 cpu 占用率低。（为 Apache 的 1/5-1/10）

4) 功能较 Apache 少（常用功能均有）

5) 对 php 可使用 cgi 方式和 fastcgi 方式。

17.2 Nginx WEB 安装配置

首先需要安装 pcre 库，然后再安装 Nginx：

#安装 pcre 支持 rewrite 库,也可以安装源码，注*安装源码时，指定 pcre 路径为解压源码的路径，而不是编译后的路径，否则会报错。

（make[1]: *** [/usr/local/pcre/Makefile] Error 127 错误）

yum install pcre-devel pcre -y

#下载 Nginx 源码包

cd /usr/src ;wget -c http://nginx.org/download/nginx-1.4.2.tar.gz

#解压 Nginx 源码包

tar -xzf nginx-1.4.2.tar.gz

#进入解压目录，然后 sed 修改 Nginx 版本信息为 WS

cd nginx-1.4.2 ; **sed -i -e 's/1.4.2//g' -e 's/nginx\//WS/g' -e**

's/"NGINX"/"WS"/g' src/core/nginx.h

#预编译 Nginx

useradd www ;./configure --user=www --group=www

--prefix=/usr/local/nginx --with-http_stub_status_module

--with-http_ssl_module

#.configure 预编译成功后，执行 make 命令进行编译

make

#make 执行成功后，执行 make install 正式安装

make install

#自此 Nginx 安装完毕

/usr/local/nginx/sbin/nginx -t 检查 nginx 配置文件是否正确，返回 OK 即正确。

[root@localhost ~]# **/usr/local/nginx/sbin/nginx -t**

nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok

nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful

[root@localhost ~]#

然后启动 nginx，/usr/local/nginx/sbin/nginx 回车即可。查看进程是否已启动：

[root@localhost ~]# ps -ef |grep nginx

nobody 5381 30285 0 May16 ? 00:04:31 nginx: worker process

root 30285 1 0 2014 ? 00:00:00 nginx: master process

/usr/local/nginx/sbin/nginx

root 32260 32220 0 12:34 pts/0 00:00:00 grep nginx

[root@localhost ~]#

17.3 Nginx 虚拟主机配置

在真实的服务器环境，为了充分利用服务器资源，一台 nginx web 服务器同时会配置 N 个虚拟域名主机，即多个域名对于同样一个 80 端口。然后服务器 IP 数量很多，也可以配置基于多个 IP 对应同一个端口。

vi 修改 nginx.conf server 段配置内容如下：

```
#virtual hosts config 2014/5/18
server {
        listen          80;
        server_name    www.a.com;
        #access_log    logs/host.access.log    main;
        location / {
            root     html/a;
            index    index.html index.htm;
        }
server {
        listen          80;
        server_name    www.b.com
        #access_log    logs/host.access.log    main;
```

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

创建两个不同的目录 mkdir –p /usr/local/nginx/html/{a,b}，然后分别在两个目录创建两个不同的 index.html 网站页面即可。通过客户端配置 hosts 指向两个域名，然后在 IE 浏览器访问测试效果。

17.4 Nginx 性能优化实战

随着访问量的不断增加，需要对 Nginx 和内核做相应的优化来满足高并发用户的访问，那下面在单台 Nginx 服务器来优化相关参数。

1) Nginx.conf 配置优化：

worker_processes 8;

nginx 进程数，建议按照 cpu 数目来指定，一般为它的倍数。

worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;

为每个进程分配 cpu，上例中将 8 个进程分配到 8 个 cpu，当然可以写多个，或者将一个进程分配到多个 cpu。

worker_rlimit_nofile 102400;

这个指令是指当一个 nginx 进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与 nginx 进程数相除，但是 nginx 分配请求并不是那么均匀，所以最好与 ulimit -n 的值保持一致。

use epoll;

使用 epoll 的 I/O 模型。epoll 是 Linux 内核为处理大批量文件描述符而作了改进的 poll，它能显著提高程序在大量并发连接中只有少量活跃的情况下的系统 CPU 利用率。

worker_connections 102400;

每个进程允许的最多连接数，理论上每台 nginx 服务器的最大连接数为 **worker_processes*worker_connections。**

keepalive_timeout 60;

keepalive 超时时间，客户端到服务器端的连接持续有效时间,当出现对服务器的后继请求时,keepalive-timeout 功能可避免建立或重新建立连接。

client_header_buffer_size 4k;

客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过 1k，不过由于一般系统分页都要大于 1k，所以这里设置为分页大小。分页大小可以用命令 getconf PAGESIZE 取得。

open_file_cache max=102400 inactive=20s;

这个将为打开文件指定缓存，默认是没有启用的，max 指定缓存数量，建议和打开文件数一致，inactive 是指经过多长时间文件没被请求后删除缓存。

open_file_cache_valid 30s;

这个是指多长时间检查一次缓存的有效信息。

open_file_cache_min_uses 1;

open_file_cache 指令中的 inactive 参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在 inactive

时间内一次没被使用，它将被移除。

2) Linux 内核参数优化：

net.ipv4.tcp_max_tw_buckets = 10000

timewait 的数量，默认是 180000。

net.ipv4.ip_local_port_range = 1024 65000

允许系统打开的端口范围。

net.ipv4.tcp_tw_recycle = 1

启用 timewait 快速回收。

net.ipv4.tcp_tw_reuse = 1

开启重用。允许将 TIME-WAIT sockets 重新用于新的 TCP 连接。

net.ipv4.tcp_syncookies = 1

开启 SYN Cookies，当出现 SYN 等待队列溢出时，启用 cookies 来处理。

17.5 Nginx 参数深入理解

Nginx 常用配置参数有 upstream，主要用于均衡后端多个实例：

Nginx 的 upstream 目前支持 5 种算法分配方式：

1) 轮询（默认 rr round robin）

每个请求按时间顺序逐一分配到后端不同的服务器，如果后端某台服务器 down 掉，自动剔除，待恢复自动添加上。

2) Weight 权重

指定轮询权重，权重越高，处理的请求就越多，weight 和访问比率成正比，用于后端服务器性能不均的情况。

3) ip_hash

每个请求根据访问的 IP 的 hash 结果分配，这样每个访客固定访问一个后端服务器，可以解决 session 的问题，一般用于登录会话。

4) fair（第三方）

按后端服务器的响应时间来分配请求，响应时间短的优先分配。

5) url_hash（第三方）

upstream 的 **fail_timeout 和 max_fails** 参数是用来判断负载均衡 upstream 中的某个 server 是否失效。

在 fail_timeout 的时间内，nignx 与 upstream 中某个 server 的连接尝试失败了 max_fails 次，则 nginx 会认为该 server 已经失效。在接下来的 fail_timeout 时间内，nginx 不再将请求分发给失效的 server。

例如在 nginx.conf 里面配置如下的 tdt_app 均衡：

```
upstream tdt_app {
    server     10.10.1.11:8080 weight=1 max_fails=2 fail_timeout=30s;
    server     10.10.1.12:8080 weight=1 max_fails=2 fail_timeout=30s;
}
```

Tdt_app 均衡两台后端 JAVA 服务，在 30 秒内 nginx 会与后端的某个 server 通信检测，如果检测连接失败 2 次，则 Nginx 会认为该 server 已经失效，然后踢出转发列表，然后在接下来的 30s 内，nginx 不再讲请求转发给失效的 server。

另外，fail_timeout 设置的时间对响应时间没影响，这个响应时间是用 proxy_connect_timeout 和 proxy_read_timeout 来控制的。

proxy_connect_timeout : Nginx 与后端服务器连接的超时时间，发起握手等候响应超时时间。

proxy_read_timeout：连接成功后_等候后端服务器响应时间，其实已经进入后端的排队之中等候处理（也可以说是后端服务器处理请求的时间）。

proxy_send_timeout :后端服务器数据回传时间，在规定时间之内后端服务器必须传完所有的数据。

keepalive_timout：一个 http 产生的 tcp 连接在传送完最后一个响应后，还需要等待多少秒后，才关闭这个连接。

Rewrite 规则实战

Rewrite 规则含义就是某个 URL 重写成特定的 URL，从某种意义上说为了美观或者对搜索引擎友好，提高收录量及排名等。

Rewrite 规则的最后一项参数为 flag 标记，支持的 flag 标记主要有以下几种：

1) last ：相当于 Apache 里德(L)标记，表示完成 rewrite；
2) break；本条规则匹配完成后，终止匹配，不再匹配后面的规则
3) redirect：返回 302 临时重定向，浏览器地址会显示跳转后的 URL 地址
4) **permanent**：返回 301 永久重定向，浏览器地址栏会显示跳转后的 URL 地址
5) last 和 break 用来实现 URL 重写，浏览器地址栏 URL 地址不变。

a) 例如用户访问 www.test.com，想直接跳转到网站下面的某个页面，www.test.com/new.index.html 如何来实现呢?

使用 Nginx Rewrite 来实现这个需求，具体如下：

在 server 中加入如下语句即可：

rewrite ^/$ http://www.wugk.com/index.html permanent;

***代表前面 0 或更多个字符**

+代表前面 1 或更多个字符

? 代表前面 0 或 1 个字符

^代表字符串的开始位置

$代表字符串结束的位置

。为通配符，代表任何字符

b) 例如多个域名跳转到同一个域名，nginx rewrite 规则写法如下：

```
server
{
    listen 80;
    server_name www.wugk.com wugk.com;
    if ( $host != 'www.wugk.com' ) {
    rewrite ^/(.*)$ http://www.wugk.com/$1 permanent;
}
```

更多深入的 rewrite 可以继续学习。

17.6 Nginx location 规则匹配

1. “= ”，字面精确匹配， 如果匹配，则跳出匹配过程。（不再进行正则匹配）

2. “^~ ”，最大前缀匹配，如果匹配，则跳出匹配过程。（不再进行正则匹配）

3. / 不带任何前缀：最大前缀匹配，举例如下：

location / 代表以"/"开头的搜索匹配， 在没有正则表达式匹配的情况下才进行这个匹配（优先级最低）

4. “~ ”，大小写相关的正则匹配

5. “~* ” ， 大小写无关的正则匹配

6. “@”， Named location 不是普通的 location 匹配，而是用于 location 内部重定向的变量。

Location @apache

其中： 1、2、3 三种情况属于 location using literal string， 即使用普通字符串的 location 匹配；

4、5 二种情况属于 location using regular expresstion，即使用正则表达式的 location 匹配；

location 匹配的优先级(与 location 在配置文件中的顺序无关)

= 精确匹配会第一个被处理。如果发现精确匹配，nginx 停止搜索其他匹配。

普通字符匹配，正则表达式规则和长的块规则将被优先和查询匹配，也就是说如果该项匹配还需去看有没有正则表达式匹配和更长的匹配。

^~ 则只匹配该规则，nginx 停止搜索其他匹配，否则 nginx 会继续处理其他 location 指令。

最后匹配理带有"~"和"~*"的指令，如果找到相应的匹配，则 nginx 停止搜索其他匹配；当没有正则表达式或者没有正则表达式被匹配的情况下，那么匹配程度最高的逐字匹配指令会被使用。

```
location  = / {
```

```
  # 只匹配"/".
  [ configuration A ]
}
location / {
  # 匹配任何请求，因为所有请求都是以"/"开始
  # 但是更长字符匹配或者正则表达式匹配会优先匹配
  [ configuration B ]
}
location = /images/ {
  # 匹配任何以 /images/ 开始的请求，并停止匹配 其它 location
  [ configuration C ]
  root /tmp/;
}
location ~* \.(gif|jpg|jpeg)$ {
  # 匹配以 gif, jpg, or jpeg 结尾的请求.
  # 但是所有 /images/ 目录的请求将由 [Configuration C]处理.
  [ configuration D ]
```

```
  Root  /var/www/html/


}
```

请求 URI 例子:

- / -> 符合 configuration A
- /documents/document.html -> 符合 configuration B
- /images/1.gif -> 符合 configuration C
- /documents/1.jpg ->符合 configuration D

正常的优先级别为:

(location =) > (location 完整路径 >) >(location ^~ 路径) >(location ~* 正则) >(location 路径)

17.7 Nginx Rewrite 规则企业案例

Rewrite 规则含义就是某个 URL 重写成特定的 URL，从某种意义上说**为了美观或者对搜索引擎友好，提高收录量及排名等。**

Rewrite 规则的最后一项参数为 flag 标记，支持的 flag 标记主要有以下几种:

1) last ：相当于 Apache 里的(L)标记，表示完成 rewrite;

2) break;本条规则匹配完成后，终止匹配，不再匹配后面的规则;

3) redirect：返回 302 临时重定向，浏览器地址会显示跳转后的 URL 地址，Nginx 返回 response 状态码 302。

4) permanent：返回 301 永久重定向，浏览器地址栏会显示跳转

后的新的 URL 地址。 Nginx 返回 response 状态码 301。

5) **last 和 break 用来实现 URL 重写，浏览器地址栏 URL 地址不变。**

a) 例如用户访问 www.test.com，想直接跳转到网站下面的某个页面，www.test.com/new.index.html 如何来实现呢?

使用 Nginx Rewrite 来实现这个需求，具体如下:

在 server 中加入如下语句即可:

rewrite ^/$ http://www.test.com/index01.html permanent;

***代表前面 0 或更多个字符**

+代表前面 1 或更多个字符

? 代表前面 0 或 1 个字符

^代表字符串的开始位置

$代表字符串结束的位置

$1 代表字符串第一个参数

$2 代表字符串第二个参数

.为通配符，代表任何一个字符

b) 例如多个域名跳转到同一个域名，nginx rewrite 规则写法如下:

```
server
{
    listen 80;
    server_name  www.wugk.com wugk.com;
    if ($host  !=  'www.wugk.com'  ) {
```

```
        rewrite ^/(.*)$    http://www.wugk.com/$1    permanent;
}
1.当访问的文件和目录不存在时，重定向到某个 php 文件
if( !-e $request_filename )
{
rewrite    ^/(.*)$    index.php    last;
}
2.目录对换  /xxxx/123456 ---/xxxx?id=123456
rewrite    ^/(.+)/(\d+)   /$2?id=$1    last;
3.浏览器请求头跳转
if( $http_user_agent   ~    MSIE)
{
        rewrite ^(.*)$ /ie/$1 break;
}
4.禁止访问以.sh,.flv,.mp3 为文件后缀名的文件
location ~ .*\.(sh|flv|mp3|xml|rar|zip)$
{
        return 403;
}
5.将 jfedu.net 跳转到 www.jfedu.net
if ($host = 'jfedu.net' ) {
            rewrite ^/(.*)$ http://www.jfedu.net/$1 permanent;
```

}

6.匹配用户浏览器代理信息：

```
if ( $http_user_agent ~* "(Android)|(iPhone)|(Mobile)|(WAP)|(UCWEB)" )
{
    rewrite ^/$    http://m.jfedu.net/ permanent;
}
```

7.Nginx-BBS 论坛 rewrite 规则配置

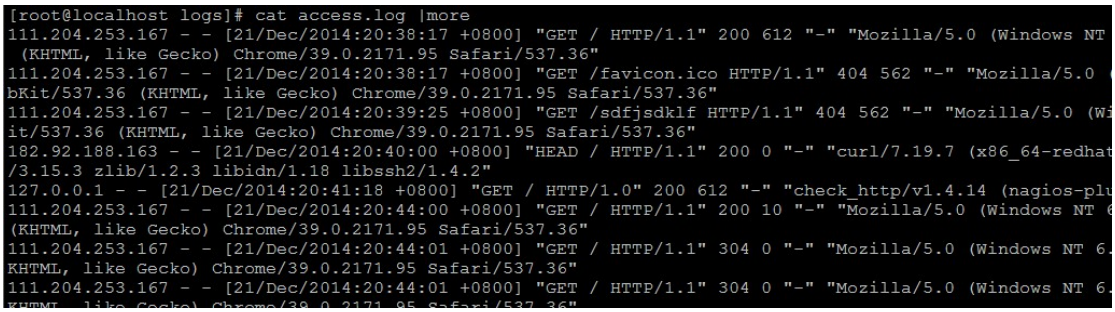
```
rewrite ^([^\.]*)/group-([0-9]+)-([0-9]+)\.html$    $1/forum.php?mod=group&fid=$2&page=$3 last;
```

更多深入的 rewrite 可以继续学习。

17.8 Nginx 日志分析及脚本编写

在我们日常的运维中，当 Nginx 服务器正常运行后，我们会经常密切关注 Nginx 访问日志的相关情况，发现有异常的日志信息需要进行及时处理。

那今天我将跟大家一起来研究和分析 Nginx 日志，nginx 默认日志路径为：/usr/local/nginx/logs/access.log 和 error.log 文件。如下图查看 nginx 日志：cat access.log |more



Nginx 日志的格式一般在 nginx.conf 里面配置，常见的格式配置如下：

```
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;
```

格式说明：

$remote_addr，$http_x_forwarded_for 记录客户端 IP 地址

$remote_user 记录客户端用户名称

$request 记录请求的 URL 和 HTTP 协议

$status 记录请求状态，默认状态：200、301、302、400、401、403、404、406、499、500、502、503、504；

$body_bytes_sent 发送给客户端的字节数，不包括响应头的大小；

$bytes_sent 发送给客户端的总字节数。

$connection_requests 当前通过一个连接获得的请求数量。

$http_referer 记录从哪个页面链接访问过来的

$http_user_agent 记录客户端浏览器相关信息

$request_length 请求的长度（包括请求行，请求头和请求正文）。

$request_time 请求处理时间，单位为秒，精度毫秒； 从读入客户端的第一个字节开始，直到把最后一个字符发送给客户端后进行日志写入为止。

Nginx 日志分析需求：

1) 分析截止目前为止访问量最高的 IP 排行。

awk '{print $1}' /usr/local/nginx/logs/access.log|sort |uniq -c |sort -nr |head -20

2) 分析从早上 9 点至中午 12 点总的访问量。

sed -n "/2016:09:00/,/2016:12:00/"p access_20161121.log

awk '/2017:08:00/,/2017:10:00/ {print $1}' access.log|sort |uniq -c|sort -nr|head -20

3) 分析上一秒的访问请求数。

sed -n "/2016:09:00:00/"p access_20161121.log

4) 找到当前日志中 502 或者 404 错误的页面并统计。

awk '{print $0}' /usr/local/nginx/logs/access.log|egrep

"404|502"|awk '{print $1,$7,$9}'|more

通过编写脚本实现自动分析 Nginx 异常访问并加入防火墙：

1）首先要判断什么是恶意访问呢？

定义为某个时间段请求一个 URL 超过 20 次的 IP 都可以封掉哦。

17.9 Nginx 日志切割案例讲解

Nginx 是一个非常轻量的 Web 服务器，体积小、性能高、速度快等诸多优点。但不足的是也存在缺点，比如在产生的访问日志文件一直就是一个，不会自动地进行切割，如果访问量很大的话，将会导致日志文件容量非常大，不便于管理。当然了，我们也不希望看到这么庞大的一个访问日志文件，那需要手动对这个文件进行切割。

如果访问日志非常大，不便于我们每天查看相关的网站异常日志。

在 Linux 平台上 Shell 脚本丰富，使用 Shell 脚本加 crontab 命令能非常方便地进行切割。

今天我将跟大家一起来写一个企业环境中使用的 Nginx 自动切换脚本。

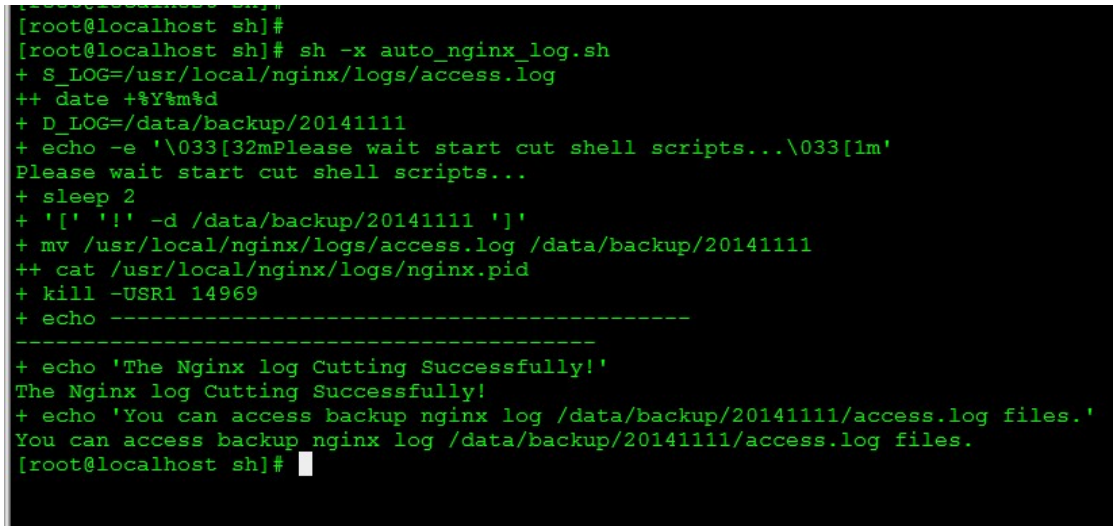
脚本的功能，就是定时切割日志，这里切割的方法我们直接使用 mv 即可。

直接上脚本内容如下：

```
#!/bin/bash
#auto mv nginx log shell
#by author wugk
```

```
S_LOG=/usr/local/nginx/logs/access.log
D_LOG=/data/backup/`date +%Y%m%d`
echo -e "\033[32mPlease wait start cut shell scripts...\033[1m"
sleep 2
if [ ! -d $D_LOG ];then

        mkdir -p  $D_LOG
fi

mv  $S_LOG  $D_LOG

kill -USR1 `cat /usr/local/nginx/logs/nginx.pid`

echo "----------------------------------------"
echo "The Nginx log Cutting Successfully!"
echo "You can access backup nginx log $D_LOG/access.log files."
```

执行如下图：



最后在 crontab 中添加如下代码即可，每天晚上自动去切割日志：

0 0 * * *

/bin/sh

/data/sh/auto_nginx_log.sh >>/tmp/nginx_cut.log

2>&1

17.10 Nginx 防盗链配置案例配置

防盗链的含义：网站内容部署自己服务器上，而通过技术手段，绕过别人放广告有利益的最终页，直接在自己的有广告有利益的页面上向最终用户提供此内容。 常常是一些名不见经传的小网站来盗取一些有实力的大网站的地址(比如一些音乐、图片、软件的下载地址)然后放置在自己的网站中，通过这种方法盗取大网站的空间和流量。

这样的话，我们会看到每天访问量很大，占用很多不必要的带宽，浪费资源，所以我们需要做一些限制。

防盗链其实就是采用服务器端编程，通过 url 过滤技术实现的防止盗链的软件。

比 如 http://chinaapp.wugk2.com/download/keepalived.conf 这个下载地址，如果没有装防盗链，别人就能轻而易举的在他的网站上引用这个地址。

防盗链的定义此内容不在自己服务器上，而通过技术手段，绕过别人放广告有利益的最终页，直接在自己的有广告有利益的页面上向最终用户提供此内容。 常常是一些名不见经传的

小网站来盗取一些有实力的大网站的地址（比如一些音乐、图片、软件的下载地址）然后放置在自己的网站中，通过这种方法盗取大网站的空间和流量。

Nginx server 防盗链配置如下：

```
server {
        listen          80;
        server_name   localhost   www.wugk2.com;
        location / {
            root     html/b;
            index   index.html index.htm;
        }
        location ~* \.(gif|jpg|png|swf|flv)$ {
                valid_referers      none      blocked
*.jfedu.net;
                root     html;
```

```
            if ($invalid_referer) {
                    return 403;
                }
            }

        }
```

第一行：gif|jpg|png|swf|flv 表示对 gif、jpg、png、swf、flv 后缀的文件实行防盗链

第二行：wugk2.com　表示对 wugk2.com 这个来路进行判断

if{}里面内容的意思是，如果来路不是指定来路就跳转到错误页面，当然直接返回 403 也是可以。

或者如下设置也可以：

```
location ~* \.(gif|jpg|png|swf|flv)$
if ($host　!=’*.wugk2.com’) {
return 403;
}
```

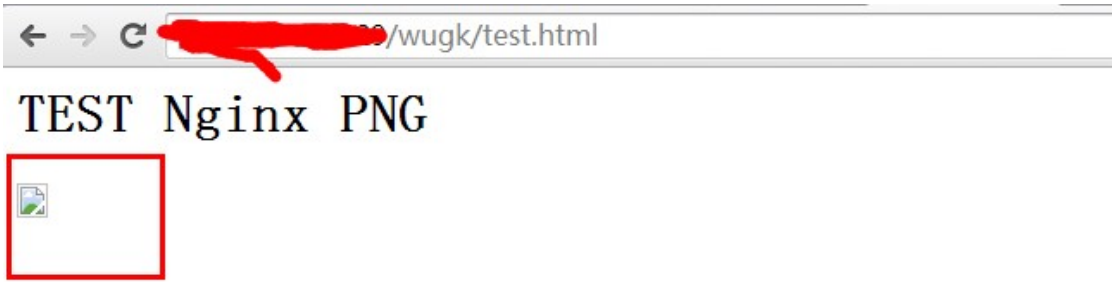
防盗链测试：

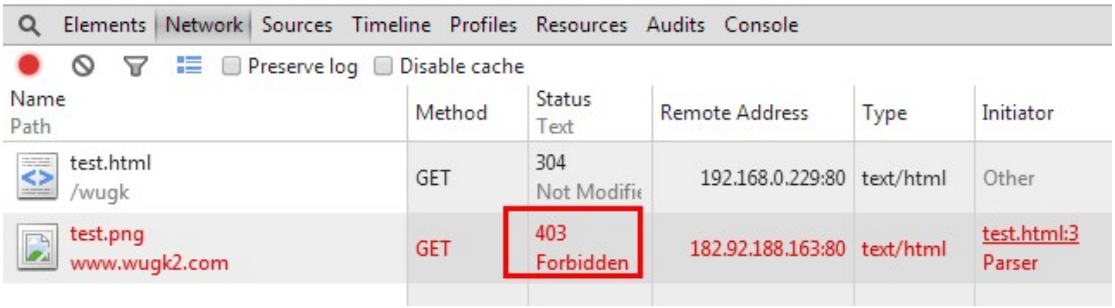
找一台测试机器，新建如下 html 页面，代码如下：

```
<html>
<h1>TEST Nginx PNG</h1>
<img src="http://www.wugk2.com/test.png">
</html>
```

然后在本地配置 hosts 指向 182.92.188.163 www.wugk2.com 即可。

然后访问如下图即代表 Nginx 防盗链配置成功：





17.11 Nginx 性能压测及评估

常见的压测工具包括:ab、webbench、loadrunner、jmeter.

ab 工具网站压力测试命令:

格式： ./ab [options] [http://]hostname[:port]/path

-n 测试会话中所执行的请求个数，默认时，仅执行一个请求

-c 一次产生的请求个数。默认是一次一个

-t 测试所进行的最大秒数

-v 设置显示信息的详细程度 - 4 或更大值会显示头信息， 3 或更大值可以显示响应代码(404, 200 等), 2 或更大值可以显示警告和其他信息。 -V 显示版本号并退出。

4 个比较长用的参数 若有其他需要 man 下吧

一般工作中我们只用-n 和 -c：

例：./ab -c 10 -n 1000 http://www.jfedu.net/index.php

-n 1000 表示总请求数为 1000

-c 10 表示并发用户数为 10

http://www.jfedu.net/index.php 表示这些请求的目标 url。

```
Server Software:        ShareKuWang/1.0
Server Hostname:        www.jfedu.net
Server Port:            80

Document Path:          /
Document Length:        71189 bytes

Concurrency Level:      10
Time taken for tests:   4.496 seconds
Complete requests:      10
Failed requests:        0
Write errors:           0
Total transferred:      717500 bytes
HTML transferred:       711890 bytes
Requests per second:    2.22 [#/sec] (mean)
Time per request:       4496.127 [ms] (mean)
Time per request:       449.613 [ms] (mean, across all concurrent requests)
Transfer rate:          155.84 [Kbytes/sec] received
```

Document Path: /

Document Length: 315 bytes HTTP 响应数据的正文长度

Concurrency Level: 800

Time taken for tests: 0.914 seconds 所有这些请求处理完成所花费的时间

Complete requests: 800 完成请求数

Failed requests: 0 失败请求数

Write errors: 0

Non-2xx responses: 800

Total transferred: 393600 bytes 网络总传输量

HTML transferred: 252000 bytes HTML 内容传输量

Requests per second: 875.22 [#/sec] (mean) 吞吐量-每秒请求数

Time per request: 914.052 [ms] (mean) 服务器收到请求，响应页面要花费的时间

Time per request: 1.143 [ms] (mean, across all concurrent requests) 并发的每个请求平均消耗时间

Transfer rate: 420.52 [Kbytes/sec] received 平均每秒网络上的流量，可以帮助排除是否存在网络流量过大导致响应时间延长的问题。

第18章 Tomcat/Resin JAVA 服务器实战

Tomcat 是由 Apache 软件基金会下属的 Jakarta 项目开发的一个 Servlet 容器，按照 Sun Microsystems 提供的技术规范，实现了对 Servlet 和 JavaServer Page（JSP）的支持，，Tomcat 本身也是一个 HTTP 服务器可以单独使用，apache 是一个以 C 语言编写的 HTTP 服务器。Tomcat 主要用来解析 JSP 语言。目前最新版本为 8.0。

18.1 Tomcat 企业安装配置

安装 tomcat 之前需要安装 jdk (Java Development Kit) 是 Java 语言的软件开发工具包(SDK))，这里选择 jdk-6u18-linux-x64-rpm.bin，bin 文件安装跟 sh 文件方法一样，

sh ./ jdk-6u18-linux-x64-rpm.bin，回车即可，默认安装到/usr/java/jdk1.6.0_18 目录下。

配置 java 环境变量，vi /etc/profile 添加如下语句：

```
export JAVA_HOME=/usr/java/jdk1.6.0_18

export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib

export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$PATH:$HOMR/bin
```

source /etc/profile //使环境变量立刻生效。

java -version //查看 java 版本，显示版本为 1.6.0_18，证明安装成功。

在官网下载 tomcat 相应版本，这里下载的版本为 apache-tomcat-6.0.30.tar.gz，下载完后解压：

```
tar -xzf apache-tomcat-6.0.30.tar.gz

;mv apache-tomcat-6.0.30 /usr/local/tomcat 即可。

启动 tomcat，命令为：/usr/local/tomcat/bin/startup.sh

查看 ps -ef |grep tomcat 进程及端口是否存在
```

通过页面访问可以看到 tomcat 默认测试页面：

Apache Tomcat

Administration

Status
Tomcat Manager

Documentation

Release Notes
Change Log
Tomcat Documentation

Tomcat Online

If you're seeing this page via a web browser, it means y

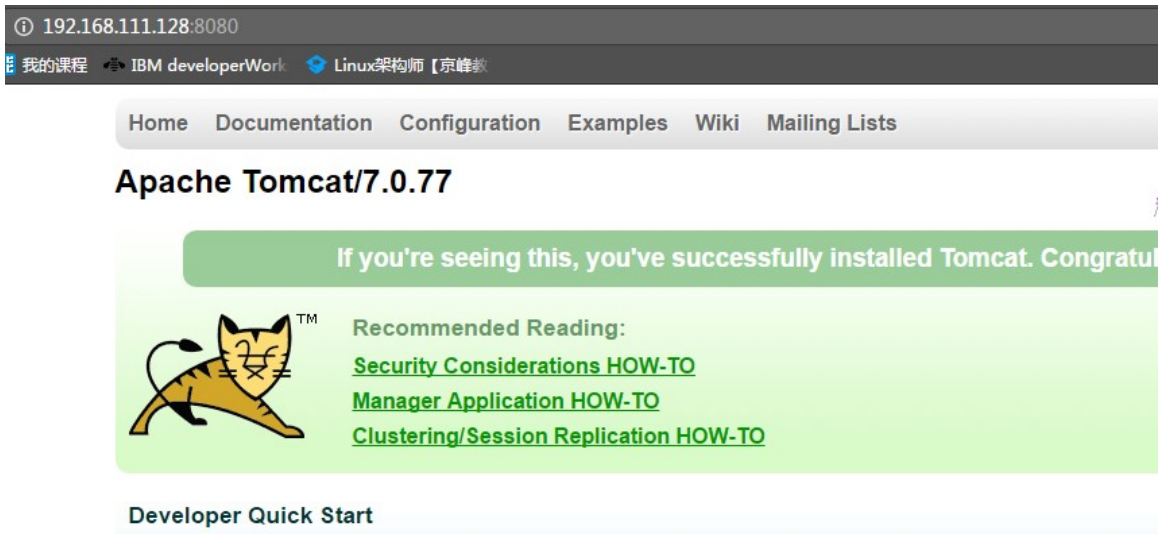
As you may have guessed by now, this is the default Tomcat home page. It can be

$CATALINA_HOME/webapps/ROOT/index.html

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you'r who has arrived at new installation of Tomcat, or you're an administrator who hasn the Tomcat Documentation for more detailed setup and administration informatior

NOTE: For security reasons, using the manager webapp is restricted to us users.xml.

Included with this release are a host of sample Servlets and JSPs (with associatec developing web applications.



可以编写 index.jsp 测试代码:

```
<html>
<body>
<h1>JSP Test Page</h1>
<%=new java.util.Date()%>
</body>
</html>
```

这个画面是默认网站，怎么来创建一个自己的网站页面呢，定义自己的发布目录，方法如下：

在 server.xml 配置文件末尾加入如下行：（附截图）

```
<Context path="/" docBase="/data/webapps/www"    reloadable="true"/>
```

```
        that are performed against this UserDatabase are immediately
        available for use by the Realm.  -->
    <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
           resourceName="UserDatabase"/>

    <!-- Define the default virtual host
         Note: XML Schema validation will not work with Xerces 2.2.
     -->
    <Host name="localhost"  appBase="webapps"
          unpackWARs="true" autoDeploy="true"
          xmlValidation="false" xmlNamespaceAware="false">

      <!-- SingleSignOn valve, share authentication between web applications
           Documentation at: /docs/config/valve.html -->
      <!--
      <Valve className="org.apache.catalina.authenticator.SingleSignOn" />
      -->

      <!-- Access log processes all example.
           Documentation at: /docs/config/valve.html -->
      <!--
      <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
             prefix="localhost_access_log." suffix=".txt" pattern="common" resolveHosts="false"/>
      -->
      <Context path="/" docBase="/data/webapps/www"  reloadable="true"/>

    </Host>
  </Engine>
```

在/data/webapps/www 目录下，创建自己的 jsp 代码，重启 tomcat 即可访问。

18.2 Tomcat 性能优化

线上环境使用默认 tomcat 配置文件，性能很一般，为了满足大量用户的访问，需要对 tomcat 进行参数性能优化，具体优化的地方如下：

- Linux 内核的优化
- 服务器资源配置的优化
- Tomcat 参数优化
- 配置负载集群优化

这里着重讲解 tomcat 参数的优化：server.xml 文件,关闭 DNS 查询、配置最大并发等参数。

maxThreads：tomcat 起动的最大线程数，即同时处理的任务个数，默认值为 200

acceptCount：当 tomcat 启动的线程数达到最大时，接受排队的请求个数，默认值为 100

当然这些值都不是越大越好，需要根据实际情况来设定。可以基于测试的基础上来不断的调优分析。

<Connector port="8080"

protocol="org.apache.coyote.http11.Http11NioProtocol"

connectionTimeout="20000"

redirectPort="8443"

maxThreads="5000"

minSpareThreads="20"

acceptCount="10000"

disableUploadTimeout="true"

enableLookups="false"

URIEncoding="UTF-8" />

Bin/catalina.sh JVM 参数优化，添加如下内容：

```
CATALINA_OPTS="$CATALINA_OPTS -Xms512M -Xmx1024M -Xmn100M -XX:SurvivorRatio=4 -XX:+UseConcMarkS
weepGC -XX:CMSInitiatingOccupancyFraction=82 -DLOCALE=UTF-16LE  -DRAMDISK=/ -DUSE_RAM_DISK=ture
 -DRAM_DISK=true -Djava.rmi.server.hostname=192.168.111.128 -Dcom.sun.management.jmxremote.port
=10000 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false"
```

配置多个 tomcat 实例，方法也很简单，只需要在服务器上 cp 多个 tomcat，然后修改三个端口和发布目录即可，然后分别启动即可。

为了提升整个网站的性能，还需要在 tomcat 前面架设 nginx web 反向代理服务器，用以提高用户高速访问。

18.3 Resin 安装配置

Resin 是 CAUCHO 公司的产品，是一个非常流行的 application server，对 servlet 和 JSP 提供了良好的支持，性能也比较优良，resin 自身采用 JAVA 语言开发。

resin 普通版本和 pro 版本主要区别是 pro 支持缓存和负载均衡。pro 因为有强大的 cache 功能，独立作为 web 服务器处理静态页面性能都可以和 apache 有一比。但普通版

本独立作为 web 服务器性能就要差一些。当然可以使用 apache+resin 的方案借助 apache 的缓存功能提高性能。

一般个人使用都使用开源免费版，如果想更高的性能，可以购买使用企业版 resin，售后服务有保障。

wget http://www.caucho.com/download/resin-4.0.33.tar.gz

tar –xzvf resin-4.0.33.tar.gz

cd resin-4.0.33 &&./configure --prefix=/usr/local/resin

\--with-resin-log=/data/logs/resin/ --with-java-home=/usr/java/jdk1.6.0_18/

make &&make install

安装完毕后，修改/usr/local/resin/conf/resin.xml 配置文件发布目录，如图：

```
<!-- define the servers in the cluster -->
<server id="" address="127.0.0.1" port="6800">
</server>

<!-- the default host, matching any host name -->
<host id="" root-directory=".">
  <!--
     - configures an explicit root web-app matching the
     - webapp's ROOT
    -->
  <web-app id="/" root-directory="/data/webapps/www"/>

  <!--
     - Administration application /resin-admin
    -->
  <web-app id="/resin-admin" root-directory="${resin.root}/doc/admin">
    <prologue>
      <resin:set var="resin_admin_external" value="false"/>
      <resin:set var="resin_admin_insecure" value="true"/>
    </prologue>
  </web-app>

  <!--
     - Resin documentation - remove for a live site
    -->
  <web-app id="/resin-doc" root-directory="${resin.root}/doc/resin-doc"/>

  <!--
     - <resin:LoadBalance regexp="^/load" cluster="backend-tier"/>
-- INSERT --
```

然后启动 resin，/usr/local/resin/bin/resin.sh start 测试成功访问如下图（发布目录未修改之前 resin 默认测试页面）：

18.4 Resin 性能优化

Resin 同 tomcat 一样，都需要优化 JVM 参数，resin 的 JVM 参数配置在 resin.xml 里面，配置最大最小内存，会话保持时间及并发数等如下所示：

```
<http address="*" port="8080"/>
        <!-- SSL port configuration: -->
        <http address="*" port="8443">
          <jsse-ssl self-signed-certificate-name="resin@localhost"/>
        </http>
          <jvm-arg>-Xms4000m</jvm-arg>
          <jvm-arg>-Xmx4000m</jvm-arg>
          <jvm-arg>-Xmn1000m</jvm-arg>
          <jvm-arg>-XX:PermSize=128m</jvm-arg>
          <jvm-arg>-XX:MaxPermSize=256m</jvm-arg>
          <thread-max>10000</thread-max>
          <socket-timeout>30s</socket-timeout>
          <keepalive-max>5000</keepalive-max>
          <keepalive-timeout>60s</keepalive-timeout>
```

<jvm-arg>-agentlib:resin</jvm-arg>

Resin 参数优化同样包括最大内存、最小内存，年轻带，最大并发，会话超时时间等。根据实际的应用来调节不同的参数。

18.5 Resin 多实例配置

为了资源最大利用，单台服务器可以配置多个 resin 实例，配置 resin 多实例的方式跟 tomcat 大部分一致，但还有一些区别：

cd /usr/local/resin/conf 下，然后 cp resin.xml resin1.xml ；cp resin.xml resin2.xml

修改两个配置文件如下图所示：

HTTP 端口为 8080 Resin1.xml 配置如下：

```
<!-- defaults for each server, i.e. JVM -->
<server-default>
  <!-- The http port -->
  <http address="*" port="8080"/>

  <!-- SSL port configuration: -->
  <http address="*" port="8443">
    <jsse-ssl self-signed-certificate-name="resin@localhost"/>
  </http>

  <!--
     - <jvm-arg>-Xmx512m</jvm-arg>
     - <jvm-arg>-agentlib:resin</jvm-arg>
    -->

</server-default>

<!-- define the servers in the cluster -->
<server id="1" address="127.0.0.1" port="6800">
</server>

<!-- the default host, matching any host name -->
<host id="" root-directory=".">
  <!--
     - configures an explicit root web-app matching the
     - webapp's ROOT
    -->
  <web-app id="/" root-directory="/data/webapps/www1"/>
```

HTTP 端口为 8081 Resin2.xml 配置如下：

```
<!-- defaults for each server, i.e. JVM -->
<server-default>
  <!-- The http port -->
  <http address="*" port="8081"/>

  <!-- SSL port configuration: -->
  <http address="*" port="8444">
    <jsse-ssl self-signed-certificate-name="resin@localhost"/>
  </http>

  <!--
     - <jvm-arg>-Xmx512m</jvm-arg>
     - <jvm-arg>-agentlib:resin</jvm-arg>
    -->

</server-default>

<!-- define the servers in the cluster -->
<server id="2" address="127.0.0.1" port="6801">
</server>

<!-- the default host, matching any host name -->
<host id="" root-directory=".">
  <!--
     - configures an explicit root web-app matching the
     - webapp's ROOT
    -->
  <web-app id="/" root-directory="/data/webapps/www2"/>
```

创建两个发布目录 mkdir -p /data/webapps/{www1,www2}写入测试 jsp 文件即可。

最后如下方法启动两个 resin 实例：

/usr/local/resin/bin/resin.sh -conf /usr/local/resin/conf/resin1.xml -server 1 start

/usr/local/resin/bin/resin.sh -conf /usr/local/resin/conf/resin2.xml -server 2 start

```
[root@node2 ~]#
[root@node2 ~]# /usr/local/resin/bin/resin.sh -conf /usr/local/resin/conf/resin1.xml -server 1 start
Resin/4.0.23 launching watchdog at 127.0.0.1:6600
Resin/4.0.23 started -server '1' for watchdog at 127.0.0.1:6600
[root@node2 ~]#
[root@node2 ~]# /usr/local/resin/bin/resin.sh -conf /usr/local/resin/conf/resin2.xml -server 2 start

Resin/4.0.23 started -server '2' for watchdog at 127.0.0.1:6600
[root@node2 ~]#
[root@node2 ~]#
[root@node2 ~]# tail -fn 5 /usr/local/resin/log/jvm-1.log
[14-05-21 11:03:15.212] {main}
[14-05-21 11:03:15.212] {main} server     = 127.0.0.1:6800 (app-tier:1)
[14-05-21 11:03:15.213] {main} stage      = production
[14-05-21 11:03:21.588] {main} WebApp[production/webapp/default/ROOT] active
[14-05-21 11:03:23.276] {main} WebApp[production/webapp/default/resin-admin] active
[14-05-21 11:03:27.444] {main} WebApp[production/webapp/default/resin-doc] active
[14-05-21 11:03:27.445] {main} Host[production/host/default] active
[14-05-21 11:03:27.447] {main} ProServer[id=1,cluster=app-tier] active
[14-05-21 11:03:27.447] {main}   JNI: file, nio keepalive (max=130816), socket
[14-05-21 11:03:27.447] {main}
[14-05-21 11:03:27.449] {main}
[14-05-21 11:03:27.454] {main} http listening to *:8080
```

真实环境，需要调整 jvm 参数，需要在 resin.xml 里面配置，同时需要开启启动 resin，只需要把上述脚本加入/etc/rc.local 即可。

18.6 Nginx Tomcat 动静分离实战

18.7 动静分离企业应用背景

Nginx 动静分离简单来说就是把动态跟静态请求分开，不能理解成只是单纯的把动态页面和静态页面物理分离。严格意义上说应该是动态请求跟静态请求分开，可以理解成使用 Nginx 处理静态页面，Tomcat、Resin 出来动态页面。

动静分离从目前实现角度来讲大致分为两种，一种是纯粹的把静态文件独立成单独的域名，放在独立的服务器上，也是目前主流推崇的方案；另外一种方法就是动态跟静态文件混合在一起发布，通过 nginx 来分开。这样也是本次课程要讲解的，具体怎么来实现呢，如下图，通过 location 指定不同的后缀名实现不同的请求转发。

通过 expires 参数设置，可以使浏览器缓存过期时间，减少与服务器之前的请求和流量。具体 Expires 定义：是给一个资源设定一个过期时间，也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可，所以不会产生额外的流量。

此种方法非常适合不经常变动的资源。（如果经常更新的文件，不建议使用 Expires 来缓存），我这里设置 3d，表示在这 3 天之内访问这个 URL，发送一个请求，比对服务器该文件最后更新时间没有变化，则不会从服务器抓取，返回状态码 304，如果有修改，则直接从服务器重新下载，返回状态码 200。

18.8 Nginx 动静分离服务器配置

```
####www.wuguangke.cn
server
  {
    listen       80;
    server_name  www.wuguangke.cn;
    index index.html index.htm;
#配置发布目录为/data/www/wugk
    root  /data/www/wugk;
    location /
    {
         proxy_next_upstream http_502 http_504 error timeout invalid_header;
         proxy_set_header Host  $host;
         proxy_set_header X-Real-IP $remote_addr;
         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
         proxy_pass http://tdt_wugk;
         expires      3d;
    }
#动态页面交给http://tdt_wugk，也即我们之前在nginx.conf定义的upstream tdt_wugk 均衡
    location ~ .*\.(php|jsp|cgi)?$
    {
         proxy_set_header Host  $host;
         proxy_set_header X-Real-IP $remote_addr;
         proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
         proxy_pass http://tdt_wugk;
    }
#配置Nginx动静分离，定义的静态页面直接从Nginx发布目录读取。
    location ~ .*\.(html|htm|gif|jpg|jpeg|bmp|png|ico|txt|js|css)$
    {
    root /data/www/wugk;
    #expires定义用户浏览器缓存的时间为3天，如果静态页面不常更新，可以设置更长，这样可以节省带宽和缓解服务器的压
    expires      3d;
    }
#定义Nginx输出日志的路径
    access_log  /data/logs/nginx_wugk/access.log main;
    error_log   /data/logs/nginx_wugk/error.log  crit;
}
```

如下为 nginx.conf 里面 server 配置段，直接添加在 nginx.conf 里即可。

```
####www.wuguangke.cn
server
  {
    listen       80;
    server_name  www.wuguangke.cn;
    index index.html index.htm;
#配置发布目录为/data/www/wugk
    root  /data/www/wugk;
    location /
    {
```

```
            proxy_next_upstream http_502 http_504 error timeout invalid_header;
            proxy_set_header    Host    $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass http://tdt_jfedu;
            expires          3d;
        }
#动态页面交给 http://tdt_jfedu，也即我们之前在 nginx.conf 定义的 upstream tdt_jfedu
均衡
        location ~ .*\.(php|jsp|cgi)?$
        {
            proxy_set_header Host    $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_pass http://tdt_jfedu;
        }
#配置 Nginx 动静分离，定义的静态页面直接从 Nginx 发布目录读取。
        location ~ .*\.(html|htm|gif|jpg|jpeg|bmp|png|ico|txt|js|css)$
        {
        root /data/www/wugk;
        #expires 定义用户浏览器缓存的时间为3天，如果静态页面不常更新，可以设置更长，
这样可以节省带宽和缓解服务器的压力
```

```
        expires          3d;
        }
#定义 Nginx 输出日志的路径
        access_log    /data/logs/nginx_wugk/access.log main;
        error_log     /data/logs/nginx_wugk/error.log    crit;
}
```

真实环境网站程序包只有一个，需要把这个程序包在 nginx 前端放一份，同时需要在 Tomcat、Resin 后端也放置一份，如果服务器涉及数量很多，那每台服务器都需要更新，可以使用批量更新方法。

18.9 LNAMP 高性能架构配置

LNAMP(Linux+Nginx+Apache+Mysql+PHP)架构受到很多 IT 企业的青睐,取代了原来认为很好的 LNMP(Linux+Nginx+Mysql+PHP)架构,那我们说 LNAMP 到底有什么优点呢,还得从 Nginx 和 apache 的优缺点说起。

Nginx 处理静态文件能力很强,Apache 处理动态文件很强而且很稳定,把二者综合在一块,性能提升很多倍。可能很多 Linux SA 在从事 LNMP 运维中，会发现 PHP（FastCGI）模式会出现一些 502 错误的现象，这是因为 Nginx+PHP（FastCGI）组合不稳定的原因造成的。

- 源码安装 LNAMP 之 Nginx

yum install prce-devel – y ;cd /usr/src ;wget http://nginx.org/download/nginx-1.6.0.tar.gz ;cd nginx-1.6.0 ;./configure –prefix=/usr/local/nginx && make &&make install

- 源码安装 LNAMP 之 Apache

yum install apr-devel apr-util-devel –y;

cd /usr/src ; wget http://mirror.bit.edu.cn/apache/httpd/httpd-2.2.31.tar.gz ;tar xzf httpd-2.2.31.tar.gz ;cd httpd-2.2.31 ;./configure --prefix=/usr/local/apache --enable-so --enable-rewrite &&make &&make install

- 源码安装 LNAMP 之 MySQL

cd /usr/src ;wget http://downloads.mysql.com/archives/mysql-5.1/mysql-5.1.63.tar.gz ;tar xzf mysql-5.1.63.tar.gz ;cd mysql-5.1.63 ;./configure --prefix=/usr/local/mysql --enable-assembler &&make &&make install

```
make[3]: Nothing to be done for `install-data-am'.
make[3]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
make[2]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
Making install in instance-manager
make[2]: Entering directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[3]: Entering directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
test -z "/usr/local/mysql/libexec" || /bin/mkdir -p "/usr/local/mysql/libexec"
  /bin/sh ../../libtool --preserve-dup-deps   --mode=install /usr/bin/install -c 'mysqlmanager' '/usr/local/m
r'
libtool: install: /usr/bin/install -c mysqlmanager /usr/local/mysql/libexec/mysqlmanager
make[3]: Nothing to be done for `install-data-am'.
make[3]: Leaving directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[2]: Leaving directory `/usr/src/mysql-5.1.63/server-tools/instance-manager'
make[1]: Leaving directory `/usr/src/mysql-5.1.63/server-tools'
Making install in win
make[1]: Entering directory `/usr/src/mysql-5.1.63/win'
make[2]: Entering directory `/usr/src/mysql-5.1.63/win'
make[2]: Nothing to be done for `install-exec-am'.
make[2]: Nothing to be done for `install-data-am'.
make[2]: Leaving directory `/usr/src/mysql-5.1.63/win'
make[1]: Leaving directory `/usr/src/mysql-5.1.63/win'
[root@node2 mysql-5.1.63]#
[root@node2 mysql-5.1.63]# ./configure  --prefix=/usr/local/mysql --enable-assembler &&make &&make install
```

配置 Mysql 服务为系统服务：

cp /usr/local/mysql/share/mysql/my-medium.cnf /etc/my.cnf

cp /usr/local/mysql/share/mysql/mysql.server /etc/rc.d/init.d/mysqld

chkconfig --add mysqld

chkconfig --level 345 mysqld on

cd /usr/local/mysql

useradd mysql

chown -R mysql.mysql /usr/local/mysql

/usr/local/mysql/bin/mysql_install_db --user=mysql

chown -R mysql var

/usr/local/mysql/bin/mysqld_safe --user=mysql &

- 源码安装 LNAMP 之 PHP

cd /usr/src ;wget http://mirrors.sohu.com/php/php-5.3.28.tar.bz2 ;tar jxf php-5.3.28.tar.bz2 ;cd php-5.3.28 ;./configure --prefix=/usr/local/php5 --with-config-file-path=/usr/local/php/etc --with-apxs2=/usr/local/apache/bin/apxs --with-mysql=/usr/local/mysql/

```
Installing PHP CLI man page:       /usr/local/php5/man/man1/
Installing build environment:      /usr/local/php5/lib/php/build/
Installing header files:           /usr/local/php5/include/php/
Installing helper programs:        /usr/local/php5/bin/
  program: phpize
  program: php-config
Installing man pages:              /usr/local/php5/man/man1/
  page: phpize.1
  page: php-config.1
Installing PEAR environment:       /usr/local/php5/lib/php/
[PEAR] Archive_Tar    - installed: 1.3.11
[PEAR] Console_Getopt - installed: 1.3.1
warning: pear/PEAR requires package "pear/Structures_Graph" (recommended version 1.0.4)
warning: pear/PEAR requires package "pear/XML_Util" (recommended version 1.2.1)
[PEAR] PEAR           - installed: 1.9.4
Wrote PEAR system config file at: /usr/local/php5/etc/pear.conf
You may want to add: /usr/local/php5/lib/php to your php.ini include_path
[PEAR] Structures_Graph- installed: 1.0.4
[PEAR] XML_Util       - installed: 1.2.1
/usr/src/php-5.3.28/build/shtool install -c ext/phar/phar.phar /usr/local/php5/bin
ln -s -f /usr/local/php5/bin/phar.phar /usr/local/php5/bin/phar
Installing PDO headers:          /usr/local/php5/include/php/ext/pdo/
[root@node2 php-5.3.28]#
```

- 源码安装 Apache+PHP 整合

整合 apache+php 环境，修改 httpd.conf 配置文件，然后加入如下语句：

LoadModule php5_module modules/libphp5.so (默认已存在)

AddType application/x-httpd-php .php

DirectoryIndex index.php index.html (把 index.php 加入 index.html 之前)

然后在/usr/local/apache/htdocs 目录下创建 index.php 测试页面，执行如下命令：

cat >>/usr/local/apache/htdocs/index.php <<EOF

<?php

phpinfo();

?>

EOF

重新启动 apache 服务，通过 IP 访问界面如下图，即代表 LAMP 环境搭建成功。

System	Linux node2 2.6.18-308.el5 #1 SMP Tue Feb 21 20:06:06 EST 2012 x86_64
Build Date	May 26 2014 14:34:36
Configure Command	'./configure' '--prefix=/usr/local/php5' '--with-config-file-path=/usr/local/php/etc' '--with-apxs2=/usr/local/apache/bin/apxs' '--with-mysql=/usr/local/mysql/'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/php/etc
Loaded Configuration File	(none)
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626

- 源码安装 DISCUZ 论坛

下载 discuz 源码包文件，然后解压：

cd /usr/src ;wget

http://download.comsenz.com/DiscuzX/3.1/Discuz_X3.1_SC_UTF8.zip

解压 discuz 程序包：unzip Discuz_X3.1_SC_UTF8.zip -d /usr/local/apache/htdocs/

重命名程序文件：cd /usr/local/apache/htdocs/ ;mv upload/* .

赋予 discuz 目录完全访问权限：cd /usr/local/apache/htdocs/ ;chmod 777 -R data/ uc_server/ config/ uc_client/

然后访问 IP 安装 discuz 论坛，如下图，选择“我同意”

进入如下界面，数据库安装，如果不存在则需要新建数据库并授权。

数据库创建及授权命令如下：

create database discuz charset=utf8;

grant all on discuz.* to root@'localhost' identified by "123456";

```
[root@node2 ~]# /usr/local/mysql/bin/mysql -uroot -p123456
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.1.63-log Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> create database discuz charset=utf8;
Query OK, 1 row affected (0.00 sec)

mysql> grant all on discuz.* to root@'localhost' identified by "123456";
Query OK, 0 rows affected (0.00 sec)

mysql>
```

点击下一步，直至安装完成，进入等待已久的论坛画面：

自此 LAMP 环境整合并搭建成功，那如何使用 Nginx 来整合 LAMP 呢?

- 源码安装 Nginx+LAMP 整合

先修改 apache 访问端口为 8080，Nginx 端口为 80。

然后修改 nginx 配置文件: vi /usr/local/nginx/conf/nginx.conf，server 配置段内容如下:

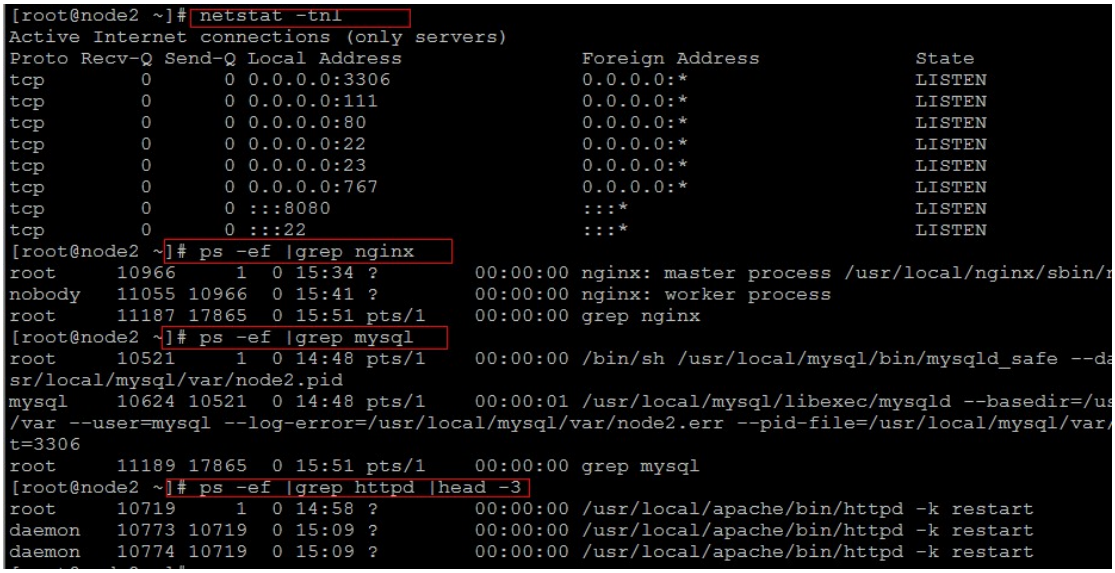
(定义 upstream 均衡模块，配置动静分离，动态转发至 apache，静态文件直接本地响应)

```
upstream app_lamp {
    server     127.0.0.1:8080 weight=1 max_fails=2 fail_timeout=30s;
    }
    server {
        listen          80;
        server_name    localhost;

        location / {
            root     /usr/local/apache/htdocs;
            index    index.php index.html index.htm;
        }
```

```
        location ~ .*\.(php|jsp|cgi)?$
    {
          proxy_set_header Host    $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_pass http://app_lamp;
    }
        location ~ .*\.(html|htm|gif|jpg|jpeg|bmp|png|ico|txt|js|css)$
       {
        root    /usr/local/apache/htdocs;
        expires         3d;
      }
    }
```

测试，访问 nginx ip+port 如下图所示：

查看系统启动的端口及进程如下图：



自此，LNAMP 全部整合完毕，接下来就是对系统内核、各个服务、架构进行优化，同样优化是一项长期的任务。

第19章　DNS 域名服务器实战篇

19.1　DNS 服务器工作原理

我们每天打开的网站，他是如何来解析，并且我们怎么能得到网站的内容反馈的界面呢？那什么是 DNS 呢（DNS（Domain Name service，域名服务，主要用于因特网上作为域名和 IP 地址相互映射）那今天我们将来学习 DNS 服务器的构建，DNS 服务可以算是 Linux 服务中比较难的一个了，尤其是配置文件书写，少一个字符都有可能造成错误。

那什么是 DNS 呢？简单的说就是完成域名到 IP 的解析过程。简洁的域名能让人们更方便记忆，不需要记那么长的 IP 访问某一个网站。

DNS 解析过程到底是怎样的呢？

第一步：客户机访问某个网站，请求域名解析,首先查找本地 HOSTS 文件，如果有对

应域名、IP 记录，直接返回给客户机。如果没有则将该请求发送给本地的域名服务器：

第二步：本地 DNS 服务器能够解析客户端发来的请求，服务器直接将答案返回给客户机。

第三步：本地 DNS 服务器不能解析客户端发来的请求，分为两种解析方法.

1、采用递归解析：本地 DNS 服务器向根域名服务器发出请求，根域名服务器对本地域名服务的请求进行解析，得到记录再给本地 DNS 服务器，本地 DNS 服务器将记录缓存，并将记录返给客户机。

2、采用迭代解析：本地 DNS 服务器向根域名服务器发出请求，根域名服务器返回给本地域名服务器一个能够解析请求的根的下一级域名服务器的地址，本地域名服务器再向根返回的 IP 地址发出请求，最终得到域名解析记录。

19.2 DNS 服务器种类

1) master（主 DNS 服务器）：拥有区域数据的文件，并对整个区域数据进行管理。

2) slave(从服务器或叫辅助服务器）：拥有主 DNS 服力器的区域文件的副本，辅助 DNS 服务器对客户端进行解析，当主 DNS 服务器坏了后，可以完全接替主服务器的工作。

3) forward:将任何查询请求都转发给其他服务器。起到一个代理的作用。

4) cache:缓存服务器。

5) hint：根 DNS internet 服务器集。

19.3 DNS 服务器安装配置

1) 首先安装 bind，执行 yum install bind* -y

2) 安装完毕修改 bind 配置文件/etc/named.conf 内容如下：

```
options {
        listen-on port 53 { any; };
        listen-on-v6 port 53 { any; };
        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     { any; };
        recursion yes;
        dnssec-enable yes;
        dnssec-validation yes;
        dnssec-lookaside auto;
        /* Path to ISC DLV key */
        bindkeys-file "/etc/named.iscdlv.key";
        managed-keys-directory "/var/named/dynamic";
};
logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};
```

zone "." IN {

type hint;

file "named.ca";

};

include "/etc/named.rfc1912.zones";

include "/etc/named.root.key";

19.4 DNS 主配置文件详解

options {

directory "/var/named"; //指定配置文件所在目录，必须配置此科

dump-file "/var/named/data/cache_dump.db;; //保存 DNS 服务器搜索到的对应 IP 地址的高速缓存

statistics_file "/var/named/data/named_stats.txt; //dns 的一些统计数据列出时就写入这个设置指定的文件中。即搜集统计数据。

pid-file "/var/run/named/named.pid ; //用于记录 named 程序的 PID 文件，可在 NAMED 启动、关闭时提供正确的 PID

allow_query (any;); //是否允许查询，或允许哪 些客户端查询，可以把 any 换上网段 地址，以设置允许查询的客户端)

allow_transfer(none;); //是否允许 MASTER 里的信息传到 SLAVE 服务器， 只有在同时拥有 MASTER 服务器和 SLAVE 服务器时才设置此项。none 为不允许

forwarders{ 192.168.3.11;192.168.3.44;} ;//设置向上查找的哪个“合法”的 DNS。地址

之间要用； 分隔。 （我的理解是此处定义的如同 windows 里定义的转发一样，当本地 DNS 服力器解析不了时，转发到你指定的一个 DNS 服务器上去解析）。

当不配置此项时，本机无法解析的都会用 name.ca 中配置的根服务器上查询，但如果配置了此项，本机查找不到的，就丢给此项中配置的 DNS 服务器处理。

forward only //让 DNS 服务器只作为转发服务器，自身不作查询。

motify //当主服务器变更时，向从服务器发送信息。 有两个选项，yes 和 no

};

以上是全局配置中常用到的选项，根据实际情况配置。

3) 修改 vi /etc/named.rfc1912.zones 配置文件（用于定义根区域和自定义区域），添加如下两段内容：

```
#add named by wugk
zone "wugk.com" IN {
        type master;
        file "wugk.com.zone";
        allow-update { none; };
};
zone "188.92.182.in-addr.arpa" IN {
        type master;
        file "wugk.com.arpa";
        allow-update { none; };
};
```

19.5 DNS 自定义区域详解

定义正向解析文件，此处以域 wugk.com 域为例。

```
  zone "wugk.com" IN {
  type master;     //定义服务器类型
  file "wugk.com";  //指定正向解析文件名。
};
```

定义反向解析文件

```
zone "1.168.192.in-addr.arpa" {
  type master;  //服务器类型
  file "named.192.168.9" ;   //反向解析文件名
};
```

4) 在/var/named/目录添加如下文件：

Wugk.com.zone 正向解析文件内容如下：

```
$TTL 86400
@     IN SOA    ns.wugk.com.    root (
                                42    ; serial
                                3H    ; refresh
                                15M ; retry
                                1W    ; expire
                                1D )       ; minimum
@                       IN NS                   ns.wugk.com.
```

ns IN A 182.92.188.163

www IN A 182.92.188.163

@ IN MX 10 mail.wugk.com.

mail IN A 182.92.188.163

Wugk.com.arpa 反向解析文件内容如下：

```
$TTL 86400
@    IN SOA   ns.wugk.com. root (
                        42   ; serial
                        3H   ; refresh
                        15M ; retry
                        1W   ; expires
                        1D )      ; minimum

@    IN NS       ns.wugk.com.
163 IN PTR       mail.wugk.com.
163 IN PTR       ns.wugk.com.
163 IN PTR       www.wugk.com.
```

正反向文件详解：

$TTL 86400 //外 DNS 服务器请求本 DNS 服务器的查询结果，在外 DNS 服务器上的缓存时间，以秒为单位.

@ IN SOA ns.wugk.com. root. (//格式为：

【主机名或域名】ttl] [calss] [type] [orgin] [mail]

主机名或域名：一般 用@代替。每个区域都有自已的 SOA 记录，此处的为指定的域名。用@表示当前的源，也可以手工指定域名。

SOA 记录 (起始授权机构) NS (Name Server) 记录 (域名服务器)

ttl:通常省略

class:类别，SOA 记录的类别对 internet 乖哦说缺省为 IN

type:类型，SOA 记录的类型就是 SOA,指明哪个 DNS 服务器对这个区域有授权。

origin：域区文件资 源，这个域区文件源就是这个域主 DNS 服务器的主机名，注意这里要求是完整的主机名，后面一定要加上“."。上例中：www.wugk.com. 而不是 www.wugk.com 如果没有加后面的点，结果将是：www.wugk.com.wugk.com

mail:一般指管理员的邮箱。但和一般 的邮箱不同，用"."代替了"@"，尾部也 要加上“."

2009121001 //作为版本控制，当域区文件修改时，序号就增加，辅助服务器对比发现与自已的不同后，就会做出更新，与主服力器同步

28800 //辅助服务器与主服务器进行更新的等待时间。间隔多久与主服务器进行更新，单位为秒。

14400 //重试间隔。当辅助服务器请求与主服务器更新失败后，再间隔多久重试传递

720000 //到期时间。当辅助服务器与主服务器之间刷新失败

后，辅助服务器还提供多久的授 权回答。因为当与主服务器失去联系一定时间后，（这个时间就是此处定义的时间），辅助服务器会把本地数据当作不可靠的数据，将停止提供查询。

如果主服务器恢复正常，则辅助服务器重新开始计时。

86400 ） //最小 TTL，即最小有效时间，表明客户端得到的回答在多长时间内有效。如果 TTL 时间长，客户端缓存保存时间长，客户端在收到查询结果时开始计时 (TTL)时间内有相同的查询周日不再查询服输务器，直接查自已的缓存。如果 TTL 时间短，则缓存更新的频率快

@ IN NS www.wugk.com. //ns 记录

www IN A 192.168.1.13 //A 记录

ftp IN CName www.wugk.com. //别名类型

mail IN MX 10 192.168.1.12 //邮件交换器

IN 表示后面的数据使用的是 INTERNET 标准

SOA 表示授权开始

@则代表相应的域名

test.com 授权主机

root.test.com 管理者信箱 root@test.com

NS：表示是这个主机是一个域名服务器，

A：定义了一条 A 记录，即主机名到 IP 地址的对应记录

MX 定义了一邮件记录

CNAME：定义了对应主机的一个别名

type 类型有三种，它们分别是 master,slave 和 hint 它们的含义分别是：

master:表示定义的是主域名服务器

slave :表示定义的是辅助域名服务器

hint:表示是互联网中根域名服务器

zone 关键字来定义域区的，一个 zone 关键字定义一个域区

PTR 记录用来解析 IP 地址对应的域名

注释二：

Serial：其格式通常会是“年月日+修改次序”(但也不一定如此，您自己能够记得就行)。当 slave 要进行资料同步的时候，会比较这个号码。如果发现在这里的号码比它那边的数值“大”，就进行更新，否则忽略。不过设 serial 有一个地方您要留意：不能超过 10 位数字!

Refresh：这里是是告诉 slave 要隔多久要进行资料同步(是否同步要看 Serial 的比较结果)。

Retry：如果 slave 在进行更新失败后，要隔多久再进行重试。

Expire：这是记录逾期时间：当 slave 一直未能成功与 master 取得联系，那到这里就放弃 retry，同时这里的资料也将标识为过期(expired)。

Minimum：这是最小默认 TTL 值，如果您在前面没有用“$TTL”来定义，就会以此值为准。

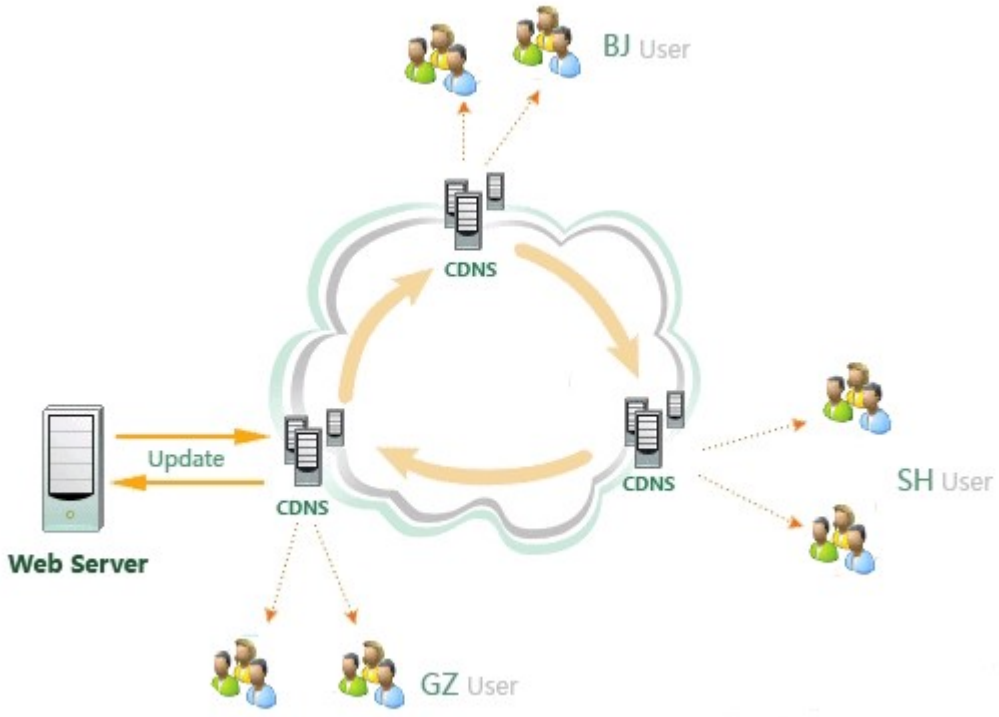
5) 测试 DNS 服务器：

找一台客户端，PC 或者其他服务器，将客户端的 DNS 修改为服务器端 DNS，访问解析的域名能正常访问即证明 DNS 服务器搭建成功。

19.6 CDN 技术入门简介

CDN（Content Delivery Network）即内容分发网络。通过在现有的 Internet 中增加一层新的网络架构，部署边缘服务器，将网站的内容发布到最接近用户的 Cache 服务器，使用户可以就近取得所需的内容，实现用户就近访问，有效提升网站的访问效果、安全性和稳定性。

附 CDN 简单拓扑图：



CDN 的实现需要依赖多种网络技术的支持，其中负载均衡技术、动态内容分发与复制技术、缓存技术是比较主要的几个，下面让我们简单看一下这几种技术。

负载均衡技术

负载均衡技术不仅仅应用于 CDN 中，在网络的很多领域都得到了广泛的应用，如服务器的负载均衡、网络流量的负载均衡。顾名思义，网络中的负载均衡就是将网络的流量尽可能均匀分配到几个能完成相同任务的服务器或网络节点上，由此来避免部分网络节点过载。

这样既可以提高网络流量，又提高了网络的整体性能。在 CDN 中，负载均衡又分为服务器负载均衡和服务器整体负载均衡(也有的称为服务器全局负载均衡)。GSLB

服务器负载均衡是指能够在性能不同的服务器之间进行任务分配，既能保证性能差的服务器不成为系统的瓶颈，又能保证性能高的服务器的资源得到充分利用。而服务器整体负载均衡允许 Web 网络托管商、门户站点和企业根据地理位置分配内容和服务。

通过使用多站点内容和服务来提高容错性和可用性，防止因本地网或区域网络中断、断电或自然灾害而导致的故障。

在 CDN 的方案中服务器整体负载均衡发挥着重要作用，其性能高低将直接影响整个 CDN 的性能。如图所示的全局调度服务器就可以看做一个服务器全局负载均衡。它根据网民的位置决定 CDN 系统哪个 Cache 服务器为网民服务。

在使用 CDN 服务的时候，全局调度服务器可以进行智能域名解析，网站需要将原来的域名 CNAME 到全局调度服务器智能解析的域名上，这样 CDN 的工作就做到了对网民的透明，网民可以访问到部署在边缘的 Cache 服务器。

具体到一个 Cache 节点，对网民看是一个的服务个体，其实内部实现也是一个服务器负载均衡。当单台服务器不能满足该地区的 Cache 业务时，负载均衡器扮演了重要角色，它将 Cache 服务集群联系到一起，其服务能力也是多台 Cache 服务能力的总和。

动态内容分发与复制技术

大家都知道，网站访问响应速度取决于许多因素，如网络的带宽是否有瓶颈、传输途中的路由是否有阻塞和延迟、网站服务器的处理能力及访问距离等。多数情况下，网站响应速度和访问者与网站服务器之间的距离有密切的关系。如果访问者和网站之间的距离过远的话，它们之间的通信一样需要经过重重的路由转发和处理，网络延误不可避免。

为了避免网络延误，就需要一个有效的方法将占网站主体的大部分静态网页、图像和流媒体数据分发复制到各地的加速节点上。同时在国内又有南北互联的问题，电信联通间的访问速度非常不好，所以动态内容分发和复制技术显得更为必要，因此动态内容分发与复制技术也是 CDN 所需的一个主要技术。

缓存技术

缓存技术已经不是一种新鲜技术。Web 缓存服务通过几种方式来改善用户的响应时间，如代理缓存服务、透明代理缓存服务、使用重定向服务的透明代理缓存服务等。通过 Web 缓存服务，用户访问网页时可以将广域网的流量降至最低。对于公司内联网用户来说，这意味着将内容在本地缓存，而无须通过专用的广域网来检索网页。对于 Internet 用户来说，这意味着将内容存储在他们的 ISP 的缓存器中，而无须通过 Internet 来检索网页。这样无疑会提高用户的访问速度。CDN 的核心作用正是提高网络的访问速度，所以，缓存技术将是 CDN 所采用的又一个主要技术。

如图示，各地的 Cache 服务器保存着源站静态内容的一份有效拷贝，网民无需直接访问源站，就可以在离自己最近的 Cache 服务器上获得新鲜正确的内容。目前缓存服务器可以有多种选择，大名鼎鼎的 squid，还有 ngnix（ncache），vanish 都可以用作 Cache 服务器。Cache 服务器的主要工作提高内容 HIT 率，使得大多数的访问都能在 Cache 设备获得，而不用 MISS 回源去取.技术要点是过期机制等内容更新管理，此外 Cache 服务器可以将源站的一些功能分担出来，实现起来更加灵活。

综上，CDN 从技术上解决由于网络带宽小、用户访问量大、网点分布不均等原因所造成的用户访问网站响应速度慢的问题，关注全国范围内不同网络中的用户都能得到优质的访问质量的网站可以采用 CDN 来提高网站的体验水平。

初期我国 CDN 市场发展缓慢，2006 年后市场陡然升温。国内整体 CDN 市场规模、运

营成熟度、服务能力和技术研发均较国外存在一定的差距。

常见的 CDN 厂商，例如：蓝汛、帝联、网宿及自建 CDN（阿里、腾讯、百度、新浪、乐视）等等。

第20章 Keepalived 高可用实战篇

20.1 MySQL 架构拓展背景

MySQL 是一个开放源码的小型关联式数据库管理系统，开发者为瑞典 MySQL AB 公司，目前属于 Oracle 公司,MySQL 被广泛地应用在 Internet 上的中小型网站中。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，许多中小型网站为了降低网站总体拥有成本而选择了 MySQL 作为网站数据库。

对应目前主流的 LAMP 架构来说,Mysql 更是得到各位 IT 运维、DBA 的青睐，目前 mysql 已被 orcacle 收购，不过好消息是原来 mysql 创始人已独立出来自己重新开发了一个 MariaDB，而且使用的人数越来越多。而且 MariaDB 兼容 mysql 所有的功能和相关参数。

MySQL 常用的两大引擎有 MyISAM 和 innoDB，那他们有什么明显的区别呢，什么场合使用什么引擎呢?

MyISAM 类型的表强调的是性能，其执行数 度比 InnoDB 类型更快，但不提供事务支持，如果执行大量的 SELECT 操作，MyISAM 是更好的选择，支持表锁。

InnoDB 提供事务支持事务，外部键等高级 数据库功能，执行大量的 INSERT 或 UPDATE，出于性能方面的考虑，应该使用 InnoDB 表，支持行锁。

随着访问量的不断增加，Mysql 数据库压力不断增加，需要对 mysql 进行优化和架构

改造，可以使用高可用、主从复制、读写分离来、拆分库、拆分表进行优化。下面我们来学习 MySQL 主从复制高可用如何来实现。

20.2 MySQL 数据库主从复制原理

Mysql 主从同步其实是一个异步复制的过程，要实现复制首先需要在 master 上开启 bin-log 日志功能，整个过程需要开启 3 个线程，分别是 Master 开启 IO 线程，slave 开启 IO 线程和 SQL 线程。

e) **在从服务器执行 slave start，从服务器上 IO 线程会通过授权的用户连接上 master，并请求 master 从指定的文件和位置之后发送 bin-log 日志内容。**

f) **Master 服务器接收到来自 slave 服务器的 IO 线程的请求后，master 服务器上的 IO 线程根据 slave 服务器发送的指定 bin-log 日志之后的内容，然后返回给 slave 端的 IO 线程。(返回的信息中除了 bin-log 日志内容外，还有本次返回日志内容后在 master 服务器端的新的 binlog 文件名以及在 binlog 中的下一个指定更新位置。)**

g) **Slave 的 IO 线程接收到信息后，将接收到的日志内容依次添加到 Slave 端的 relay-log 文件的最末端，并将读取到的 Master 端的 bin-log 的文件名和位置记录到 master-info 文件中，以便在下一次读取的时候能够清楚的告诉 Master "我需要从某个 bin-log 的哪 个位置开始往后的日志内容，请发给我"；**

h) **Slave 的 Sql 线程检测到 relay-log 中新增加了内容后，会马上解析 relay-log 的内容成为在 Master 端真实执行时候的那些可执行的内容，并在自身执行。**

- **MySQL 数据库主从配置**

  环境准备：192.168.1.103 为 master 主服务器，192.168.33.11 为 slave 从服务器。

  在主和从服务器都安装 mysql 相关软件，命令如下：

yum install -y mysql mysql-devel mysql-server mysql-libs

安装完毕后，在 Master 修改 vi /etc/my.cnf 内容为如下：

[mysqld]

datadir=/data/mysql

socket=/var/lib/mysql/mysql.sock

user=mysql

Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

log-bin=mysql-bin

server-id = 1

auto_increment_offset=1

auto_increment_increment=2

[mysqld_safe]

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

replicate-do-db =all

创建/data/mysql 数据目录，mkdir –p /data/mysql ;chown –R mysqld:mysqld /data/mysql

启动 mysql 即可，/etc/init.d/mysqld restart

然后修改 slave Mysql 数据库 my.cnf 配置文件内容如下：

[mysqld]

datadir=/data/mysql

socket=/var/lib/mysql/mysql.sock

user=mysql

Disabling symbolic-links is recommended to prevent assorted security risks

symbolic-links=0

log-bin=mysql-bin

server-id = 2

auto_increment_offset=2

auto_increment_increment=2

[mysqld_safe]

log-error=/var/log/mysqld.log

pid-file=/var/run/mysqld/mysqld.pid

master-host =192.168.1.103

master-user=tongbu

master-pass=123456

master-port =3306

master-connect-retry=60

replicate-do-db =all

在 Master 数据库服务器上设置权限，执行如下命令：

grant replication slave on *.* to 'tongbu'@'%' identified by '123456';

在 Master 数据库执行如下命令：

mysql> show master status;

```
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000006 |       98 |              |                  |
+------------------+----------+--------------+------------------+
```

1 row in set (0.00 sec)

然后在 slave 服务器指定 master IP 和同步的 pos 点：

change master to

master_host='192.168.1.103',master_user='tongbu',master_password='123456

',master_log_file='mysql-bin.000006',master_log_pos=98;

在 slave 启动 slave start，并执行 show slave status\G 查看 Mysql 主从状态：

```
[root@node2 ~]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.0.95-log Source distribution

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> slave start;
Query OK, 0 rows affected (0.00 sec)

mysql> show slave status\G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.149.128
                Master_User: tongbu
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000006
        Read_Master_Log_Pos: 98
             Relay_Log_File: mysqld-relay-bin.000004
              Relay_Log_Pos: 235
      Relay_Master_Log_File: mysql-bin.000006
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB:
```

Slave_IO_Running: Yes

Slave_SQL_Running: Yes 两个状态为 YES，代表 slave 已经启动两个线程，一个为 IO 线程，一个为 SQL 线程。

然后在 Master 服务器创建一个数据库和表，命令如下：

192.168.149.128 × 192.168.149.129

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

mysql> create database mysql_ab_test charset=utf8;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| mysql_ab_test      |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql> use mysql_ab_test;
Database changed
mysql>
mysql> create table t0 (id varchar(20),name varchar(30));
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+-------------------------+
| Tables_in_mysql_ab_test |
+-------------------------+
| t0                      |
+-------------------------+
1 row in set (0.00 sec)
```

然后去 slave 服务器查看是否有 mysql_ab_test 数据库和相应 t0 的表，如果存在则代表

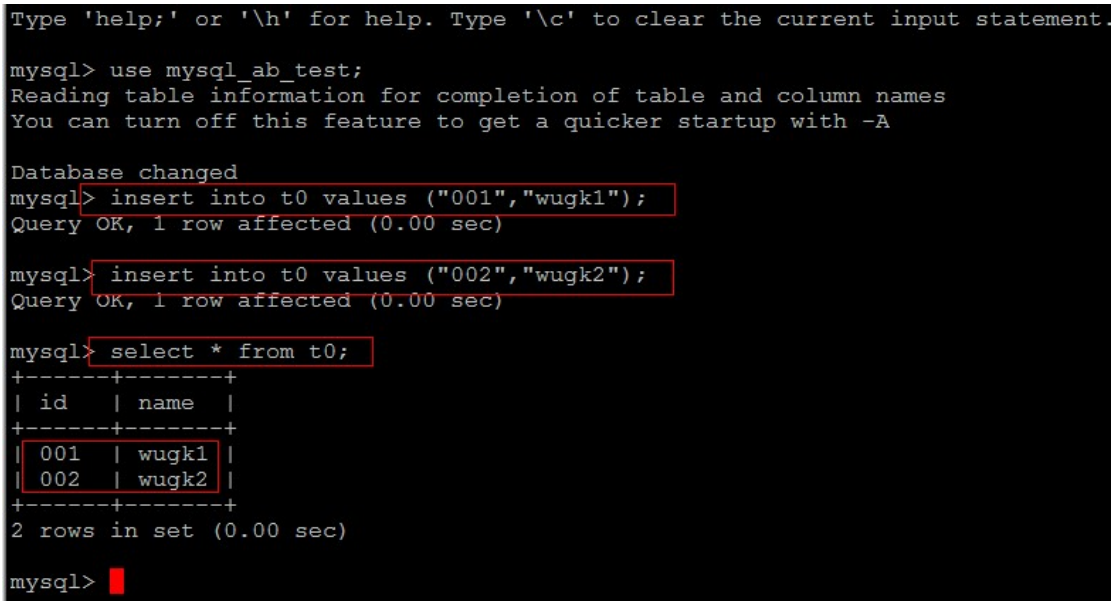
Mysql 主从同步搭建成功：

192.168.149.128 192.168.149.129 ×

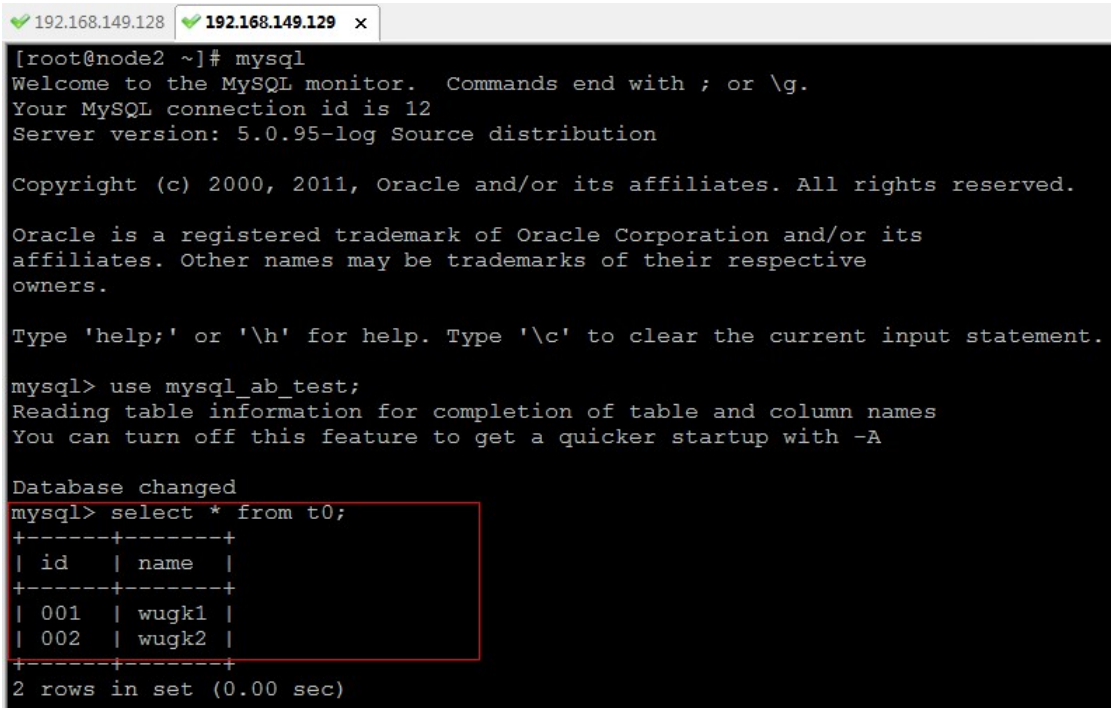
```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| mysql_ab_test      |
| test               |
+--------------------+
4 rows in set (0.00 sec)

mysql> use mysql_ab_test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-------------------------+
| Tables_in_mysql_ab_test |
+-------------------------+
| t0                      |
+-------------------------+
1 row in set (0.00 sec)
```

同样还可以测试在 master 服务器插入两条数据，在 slave 查看 insert 数据是否已同步：

128 master 上执行如下图：



129 slave 上执行如下图，在 master 插入的数据已经同步到 slave 上：



Mysql 主从搭建完毕，现在有一个问题，如果 master 服务器 down 机了，如何快速恢复服务呢?

可以通过两种方法：

第一种方法，如果程序连接的是 master 的 IP，直接在 slave 服务器上添加 master 的

IP 即可。这个手动去操作，而且需要花费时间比较长，可能还会出现误操作的情况，不推荐。

第二种方法，可以使用 keepalived、heartbeat 作为 HA 检测软件，检查 MySQL 服务是否正常，不正常则自动切换到 slave 上，推荐使用。

➢ MySQL+keepalived 高可用配置

继上一章节 MySQL 主从配置完毕后，接着配置 keepalived、heartbeat 服务，主要用于 Mysql 故障自动切换。那说到 keepalived，keepalived 到底是什么呢?

20.3 Keepalived 工作原理介绍

keepalived 是一个类似于 layer3, 4 & 7 交换机制的软件，也就是我们平时说的第 3 层、第 4 层和第 7 层交换。

Keepalived 的作用是检测 web 服务器的状态，如果有一台 web 服务器、Mysql 服务器宕机，或工作出现故障，Keepalived 将检测到后，会将有故障的 web 服务器或者 Mysql 服务器从系统中剔除，当服务器工作正常后 Keepalived 自动将 web、Mysql 服务器加入到服务器群中，这些工作全部自动完成，不需要人工干涉，需要人工做的只是修复故障的 WEB 和 Mysql 服务器。

Layer3,4&7 工作在 IP/TCP 协议栈的 IP 层，TCP 层，及应用层,原理分别如下：

Layer3：Keepalived 使用 Layer3 的方式工作式时，Keepalived 会定期向服务器群中的服务器发送一个 ICMP 的数据包（既我们平时用的 Ping 程序）,如果发现某台服务的 IP 地址没有激活，Keepalived 便报告这台服务器失效，并将它从服务器群中剔除，这种情况的典型例子是某台服务器被非法关机。Layer3 的方式是以服务器的 IP 地址是否有效作为服务器工作正常与否的标准。

Layer4:如果您理解了 Layer3 的方式，Layer4 就容易了。Layer4 主要以 TCP 端口的状态来决定服务器工作正常与否。如 web server 的服务端口一般是 80，如果 Keepalived 检测到 80 端口没有启动，则 Keepalived 将把这台服务器从服务器群中剔除。

Layer7：Layer7 就是工作在具体的应用层了，比 Layer3,Layer4 要复杂一点，在网络上占用的带宽也要大一些。Keepalived 将根据用户的设定检查服务器程序的 URL 运行是否正常，如果与用户的设定不相符，则 Keepalived 将把服务器从服务器群中剔除。

VRRP 介绍

keepalived 是 VRRP 的完美实现，因此在介绍 keepalived 之前，先介绍一下 VRRP 的原理。

在现实的网络环境中，两台需要通信的主机大多数情况下并没有直接的物理连接。对于这样的情况，它们之间路由怎样选择？主机如何选定到达目的主机的下一跳路由，这个问题通常的解决方法有二种：

1) 在主机上使用动态路由协议(RIP、OSPF 等)

2) 在主机上配置静态路由

很明显，在主机上配置动态路由是非常不切实际的，因为管理、维护成本以及是否支持等诸多问题。配置静态路由就变得十分流行，但路由器(或者说默认网关 default gateway)却经常成为单点。

VRRP 的目的就是为了解决静态路由单点故障问题。

VRRP 通过一竞选(election)协议来动态的将路由任务交给 LAN 中虚拟路由器中的某台 VRRP 路由器。

工作机制

在一个 VRRP 虚拟路由器中，有多台物理的路由器，但是这多台的物理的机器并不能同时工作，而是由一台称为 MASTER 的负责路由工作，其它的都是 BACKUP，MASTER 并非一成不变，VRRP 让每个 VRRP 路由器参与竞选，最终获胜的就是 MASTER。

MASTER 拥有一些特权，比如拥有虚拟路由器的 VIP 地址，我们的主机就是用这个 VIP 地址作为静态路由的。拥有特权的 MASTER 要负责转发发送给网关地址的包和响应 ARP 请求。

VRRP 通过竞选协议来实现虚拟路由器的功能，所有的协议报文都是通过 IP 多播(multicast)包(多播地址 **224.0.0.18**)形式发送的。虚拟路由器由 VRID(范围 0-255)和一组 IP 地址组成，对外表现为一个周知的 MAC 地址。**所以，在一个虚拟路由器中，不管谁是 MASTER，对外都是相同的 MAC 和 IP(称之为 VIP)。**客户端主机并不需要因为 MASTER 的改变而修改自己的路由配置，对他们来说，这种主从的切换是透明的。

在一个虚拟路由器中，**只有作为 MASTER 的 VRRP 路由器会一直发送 VRRP 广告包(VRRP Advertisement message)，BACKUP 不会抢占 MASTER**，除非它的优先级(priority)更高。当 MASTER 不可用时(BACKUP 收不到广告包)， 多台 BACKUP 中优先级最高的这台会被抢占为 MASTER。这种抢占是非常快速的(<1s)，以保证服务的连续性。由于安全性考虑，VRRP 包使用了加密协议进行加密。

20.4 Mysql+Keepalived 高可用实战

```
tar zxf keepalived-1.2.1.tar.gz

cd keepalived-1.2.1
```

```
./configure --with-kernel-dir=/usr/src/kernels/2.6.18-164.el5-i686/
&&make && make install
DIR=/usr/local/ ;\cp $DIR/etc/rc.d/init.d/keepalived  /etc/rc.d/init.d/ ;
\cp  $DIR/etc/sysconfig/keepalived  /etc/sysconfig/  &&  mkdir  -p
/etc/keepalived ;\cp $DIR/sbin/keepalived  /usr/sbin/
```

修改 Master 服务器上 keepalived.conf 代码如下:

```
! Configuration File for keepalived
global_defs {
    notification_email {
        wgkgood@139.com
    }
    notification_email_from wgkgood@139.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}
# VIP1  VRRP 协议 Config
vrrp_instance VI_1 {
     state BACKUP
     interface eth0
     lvs_sync_daemon_inteface eth0
     virtual_router_id 151
```

```
    priority 100
    advert_int 5
    nopreempt
    authentication {
        auth_type PASS
        auth_pass 2222
    }
    virtual_ipaddress {
        192.168.1.111
    }
}
virtual_server 192.168.1.111 3306 {
    delay_loop 6
    lb_algo wrr
    lb_kind DR
    persistence_timeout 60
    protocol TCP
    real_server 192.168.1.103 3306 {
        weight 100
        notify_down /data/sh/mysql.sh
        TCP_CHECK {
        connect_timeout 10
```

```
        nb_get_retry 3
        delay_before_retry 3
        connect_port 3306
        }
}
```

Mysql 从服务器配置 keepalived.conf 跟 master 一样，只需要把 Realserver IP 修改成 real_server 192.168.33.11 ；优先级从 100 改成 90 即可。

在 master、slave 数据库上创建/data/sh/mysql.sh 脚本，内容为：

/etc/init.d/keepalived stop

然后分别重启两台数据库上 keepalived 服务即可。最后测试停止 master Mysql 服务，是否会自动切换到 Backup 上。

关于 Mysql 集群高可用就在此告一段落，当然除了 keepalived 高可用之外，Mysql 优化还可以进行读写分离、Mysql+DRBD、拆分表等等优化，有兴趣的童鞋可以继续深入研究。

20.5 Keepalived 配置文件深入剖析

完整的 keepalived 的配置文件，其配置文件 keepalived.conf 可以包含三个文本块：全局定义块、VRRP 实例定义块及虚拟服务器定义块。全局定义块和虚拟服务器定义块是必须的，如果在只有一个负载均衡器的场合，就不须 VRRP 实例定义块。

```
#全局定义块
global_defs {
        notification_email {                    #指定 keepalived 在发生切换时需要发送 email 到的对象，一行一个
```

```
            wgkgood@gmail.com
        }
       notification_email_from    wgkgood@163.com      #指定发件人
       smtp_server 127.0.0.1                          #指定 smtp 服务器地址
       smtp_connect_timeout 3                         #指定 smtp 连接超时时间
       router_id LVS_DEVEL                            #运行 keepalived 机器的一个标识
}
#监控 Nginx 进程
vrrp_script    chk_nginx {
      script "/data/script/nginx.sh"                  #监控服务脚本;
      interval 2                                      #检测时间间隔(执行脚步间隔)
      weight 2
}
#VRRP 实例定义块
vrrp_sync_group VG_1{                                 监控多个网段的实例
           group {
   VI_1                                    实例名
   VI_2
 }
 notify_master /data/sh/nginx.sh                      #指定当切换到 master 时，执行的
脚本
 notify_backup /data/sh/nginx.sh                      #指定当切换到 backup 时，执行的
```

```
脚本
 notify /data/sh/nginx.sh                          #发送任何切换，均执行的脚本
 smtp_alert                                        #使用 global_defs 中提供的邮
件地址和 smtp 服务器发送邮件通知
}
vrrp_instance VI_1 {
    state BACKUP                                   #设置主机状态，MASTER|BACKUP
    nopreempt                                      #设置为不抢占
    interface eth0                                 #对外提供服务的网络接口
    lvs_sync_daemon_inteface eth0                  #负载均衡器之间监控接口;
    track_interface{                               #设置额外的监控，网卡出现问题都
会切换;
     eth0
     eth1
    }
    mcast_src_ip                                   #发送多播包的地址，如果不设置默
认使用绑定网卡的 primary ip
    garp_master_delay                              #在切换到 master 状态后，延迟进
行 gratuitous ARP 请求
    virtual_router_id 50                           #VRID 标记 ,路由 ID，可通过
#tcpdump vrrp 查看
    priority 90                                    #优先级，高优先级竞选为 master
```

```
    advert_int 1                                    #检查间隔，默认 1 秒
    preempt_delay                                   #抢占延时，默认 5 分钟
    debug                                           #debug 级别
    authentication {                                #设置认证
        auth_type PASS                                  #认证方式
        auth_pass 22222                             #认证密码
    }
    track_script {                                  #以脚本为监控 chk_nginx;
        chk_nginx
    }
    virtual_ipaddress {                         #设置 vip
        192.168.111.188
    }
}
```

注意：使用了脚本监控 Nginx 或者 MYSQL，不需要如下虚拟服务器设置块。

```
#虚拟服务器定义块
virtual_server 192.168.111.188 3306 {
    delay_loop 6                        #健康检查时间间隔
    lb_algo rr                          #调度算法 rr|wrr|lc|wlc|lblc|sh|dh
    lb_kind DR                          #负载均衡转发规则 NAT|DR|RUN
    persistence_timeout    5            #会话保持时间
    protocol TCP                        #使用的协议
```

```
    real_server 192.168.1.12 3306 {
                    weight 1                    #默认为 1,0 为失效
                    notify_up     <string> | <quoted-string> #在检测到 server up 后执行
脚本;
                    notify_down  <string> | <quoted-string> #在检测到 server down 后
执行脚本;
                    TCP_CHECK {
                    connect_timeout 3       #连接超时时间;
                    nb_get_retry 3          #重连次数;
                    delay_before_retry 3    #重连间隔时间;
                    connect_port 3306       #健康检查的端口的端口;
                    }
                   HTTP_GET{
                   url{                         #检查 url，可以指定多个
                   path /
                   digest ATM                   #检查后的摘要信息
                   status_code 200              #检查的返回状态码
                      }
        }
}
```

第21章　LVS 负载均衡实战篇

21.1　LVS 负载均衡入门简介

LVS 是 Linux Virtual Server 的简写，意即 Linux 虚拟服务器，是一个虚拟的服务器集群系统。本项目在 1998 年 5 月由章文嵩博士成立，是中国国内最早出现的自由软件项目之一。

LVS 简单工作原理：用户请求 LVS VIP，LVS 根据转发方式和算法，将请求转发给后端服务器，后端服务器接受到请求，返回给用户。对于用户来说，看不到 WEB 后端具体的应用。

LVS 特点如下：

可伸缩网络服务的几种结构，它们都需要一个前端的负载调度器（或者多个进行主从备份）。我们先分析实现虚拟网络服务的主要技术，指出 IP 负载均衡技术是在负载调度器的实现技术中效率最高的。在已有的 IP 负载均衡技术中，主要有通过网络地址转换（Network Address Translation）将一组服务器构成一个高性能的、高可用的虚拟服务器，我们称之为 VS/NAT 技术（Virtual Server via Network Address Translation）。

在分析 VS/NAT 的缺点和网络服务的非对称性的基础上，我们提出了通过 IP 隧道实现虚拟服务器的方法 VS/TUN （Virtual Server via IP Tunneling），和通过直接路由实现虚拟服务器的方法 VS/DR（Virtual Server via Direct Routing），它们可以极大地提高系统的伸缩性。VS/NAT、VS/TUN 和 VS/DR 技术是 LVS 集群中实现的三种 IP 负载均衡技术。

21.2 LVS 负载均衡工作原理

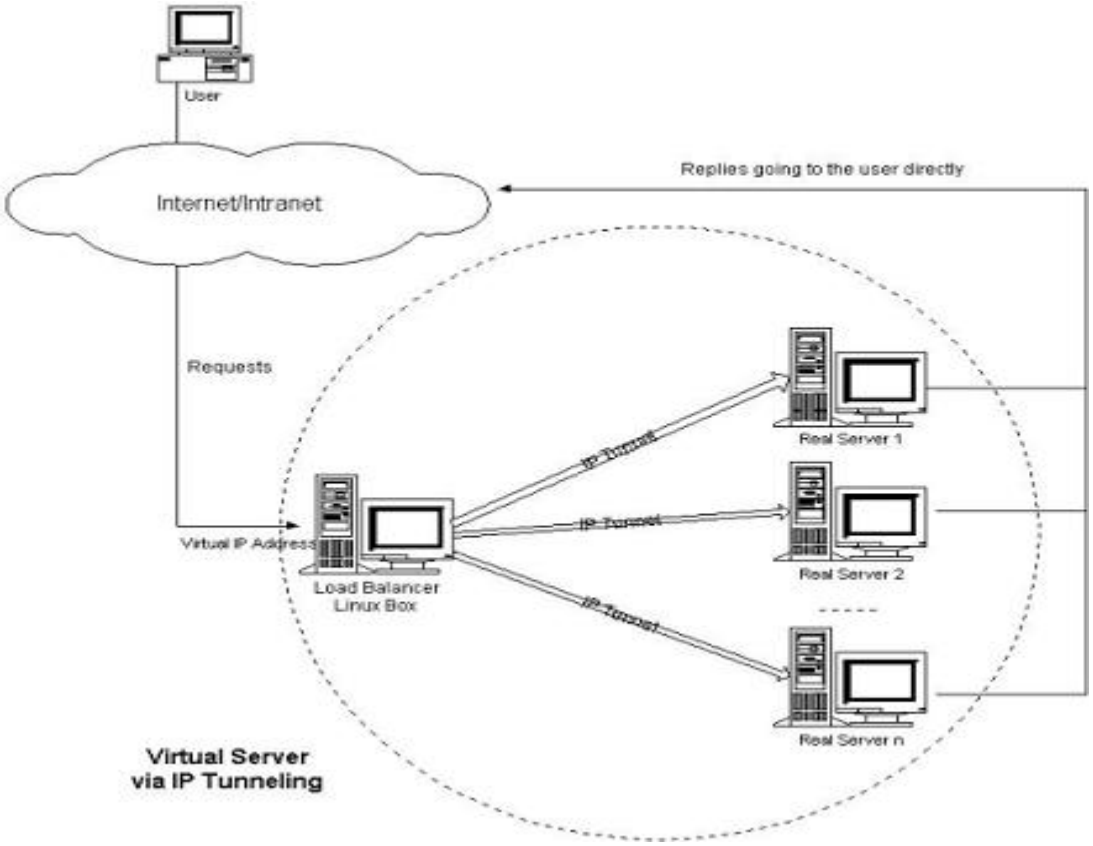
LVS 转发方式有三种，分别是 NAT、DR、TUN 模式，常用算法：RR(round-robin)、LC(least_connection)、W(weight)RR、WLC 模式等（RR 为轮询模式，LC 为最少连接模式）

LVS NAT 原理：用户请求 LVS 到达 director,director 将请求的报文的目标 IP 地址改成后端的 realserver IP 地址，同时将报文的目标端口也改成后端选定的 realserver 相应端口，最后将报文发送到 realserver，realserver 将数据返给 director，director 再把数据发送给用户。（两次请求都经过 director，所以访问大的话，director 会成为瓶颈）

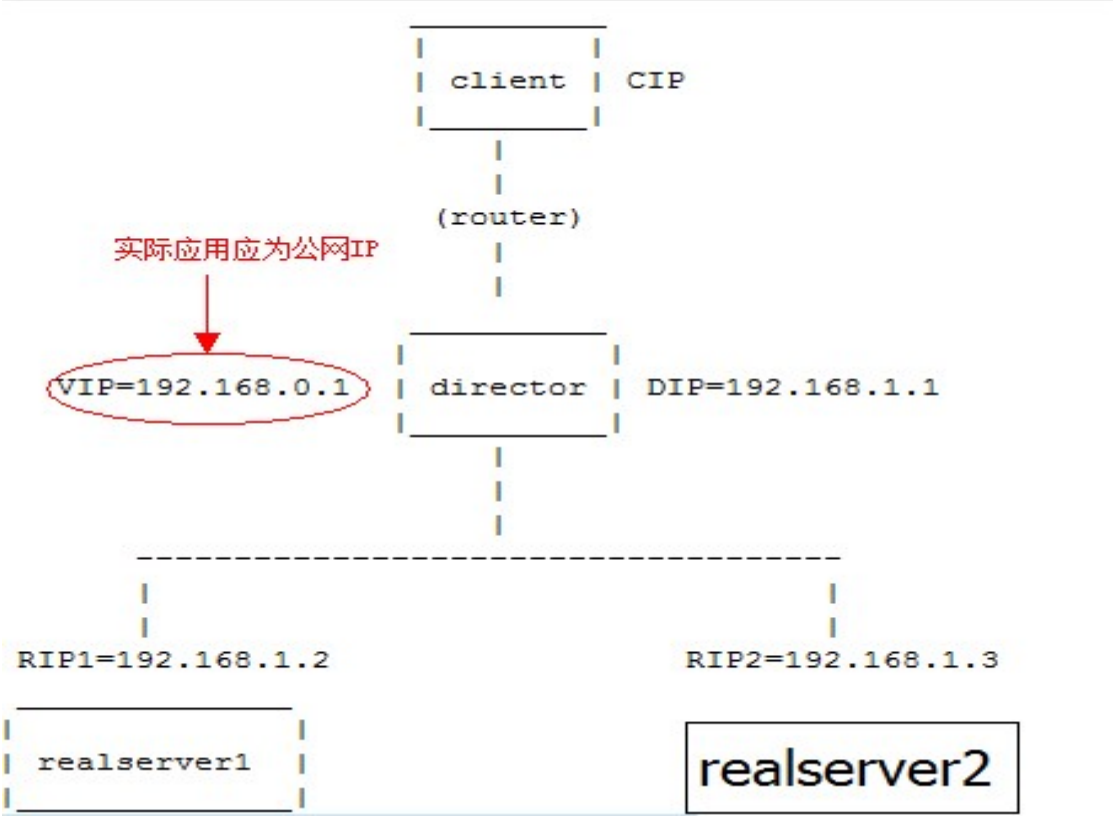
LVS DR 原理：用户请求 LVS 到达 director,director 将请求的报文的目标 MAC 地址改成后端的 realserver MAC 地址，目标 IP 为 VIP（不变），源 IP 为用户 IP 地址（保持不变），然后 Director 将报文发送到 realserver，realserver 检测到目标为自己本地 VIP，如果在同一个网段，然后将请求直接返给用户。如果用户跟 realserver 不在一个网段，则通过网关返回用户。（此种转发效率最高）

单块网卡流量 1000Mbs，100Mbs—12MB，40Gbs 网卡流量，LVS 前端集群，30 台 LVS 同时处理用户请求，600 台 LVS 量级，Docker 30 万台！

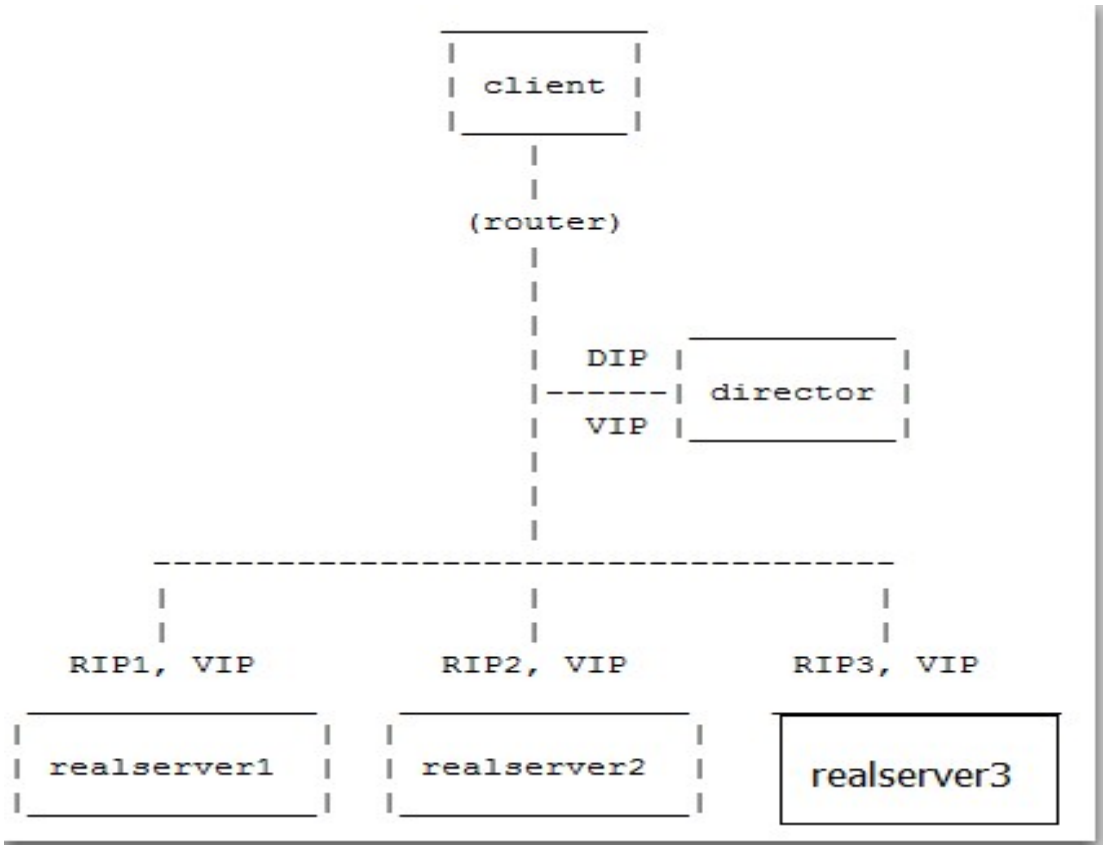
LVS TUN 原理：跟 LVS DR 类似，也是改变封装 MAC 地址，多了一层隧道加密。实施环境复杂，比 LVS DR 模式效率略低。（图一为 LVS 负载均衡图）



21.3 LVS 负载均衡 NAT 模式工作图解



21.4 LVS 负载均衡 DR 模式工作图解



LVS-DR 模式原理详解图：

原图：

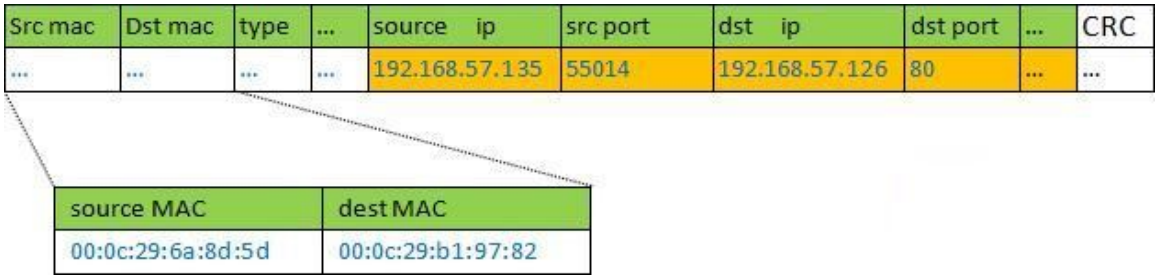
Director	VIP	RealServer	client/gateway
192.168.57.120(eth0) 00:0c:29:6a:8d:5d	192.168.57.126 00:0c:29:6a:8d:5d	192.168.57.122(eth0) 00:0c:29:b1:97:82	192.168.57.135 00:18:82:3c:e8:96

第一步请求：

Src mac	Dst mac	type	...	source ip	src port	dst ip	dst port	...	CRC
...	...	...	...	192.168.57.135	55014	192.168.57.126	80	...	...

source MAC	dest MAC
00:18:82:3c:e8:96	00:0c:29:6a:8d:5d

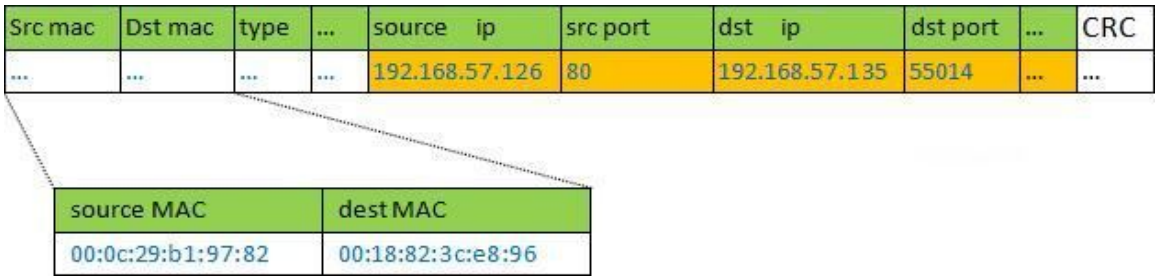
第二步传输：



Src mac	Dst mac	type	...	source ip	src port	dst ip	dst port	...	CRC
...	...	...	...	192.168.57.135	55014	192.168.57.126	80	...	...

source MAC	dest MAC
00:0c:29:6a:8d:5d	00:0c:29:b1:97:82

第三步数据返回：



Src mac	Dst mac	type	...	source ip	src port	dst ip	dst port	...	CRC
...	...	...	...	192.168.57.126	80	192.168.57.135	55014	...	...

source MAC	dest MAC
00:0c:29:b1:97:82	00:18:82:3c:e8:96

21.5 LVS 负载均衡实战配置

下载 LVS 所需软件 **ipvsadm-1.2.4.tar.gz** 软件，编译安装：

wget -c

http://www.linuxvirtualserver.org/software/kernel-2.6/ipvsadm-1.24.tar.gz

ln -s /usr/src/kernels/2.6.* /usr/src/linux //IPVS 模块编译进内核里，需要做软连接

tar xzvf ipvsadm-1.24.tar.gz &&cd ipvsadm-1.24 && make && make install

LVS 安装完毕之后，需要进行配置，配置的步骤有两步，第一步为定义端口服务，第二步为添加 realserver 后端服务。

ipvsadm -A -t 192.168.33.188:80 -s rr

ipvsadm -a -t 192.168.33.188:80 -r 192.168.33.12 -g -w 2

ipvsadm -a -t 192.168.33.188:80 -r 192.168.33.13 -g -w 2

可以用脚本自动部署（LVS 上一键安装）：

```
#!/bin/bash
SNS_VIP=$2
SNS_RIP1=$3
SNS_RIP2=$4
if [ "$1" == "stop" -a -z "$2" ];then
    echo "----------------------------------------"
    echo -e "\033[32mPlease Enter $0 stop LVS_VIP\n\nEXample:$0 stop 192.168.1.111\033[0m"
    echo
    exit
else
    if [ -z "$2" -a -z "$3" -a -z "$4" ];then
        echo "----------------------------------------"
        echo -e "\033[32mPlease Enter Input $0 start VIP REALSERVER1 REALSERVER2\n\nEXample:$0 start/stop 192.168.1.111 192.168.1.2 192.168.1.3\033[0m"
        echo
        exit 0
    fi
fi
. /etc/rc.d/init.d/functions
```

```
logger $0 called with $1

function IPVSADM(){

/sbin/ipvsadm --set   30 5 60

/sbin/ifconfig eth0:0 $SNS_VIP broadcast $SNS_VIP netmask 255.255.255.255

broadcast $SNS_VIP up

/sbin/route add -host $SNS_VIP dev eth0:0

/sbin/ipvsadm -A -t $SNS_VIP:80 -s wlc -p 120

/sbin/ipvsadm -a -t $SNS_VIP:80 -r $SNS_RIP1:80 -g -w 1

/sbin/ipvsadm -a -t $SNS_VIP:80 -r $SNS_RIP2:80 -g -w 1

}

case "$1" in

start)

IPVSADM

echo "----------------------------------------------------"

/sbin/ipvsadm -Ln

touch /var/lock/subsys/ipvsadm > /dev/null 2>&1

;;

stop)

/sbin/ipvsadm -C

/sbin/ipvsadm -Z

ifconfig eth0:0 down >>/dev/null 2>&1

route del $SNS_VIP >>/dev/null 2>&1
```

```
rm -rf /var/lock/subsys/ipvsadm > /dev/null 2>&1

echo "ipvsadm stopped!"

;;

status)

if [ ! -e /var/lock/subsys/ipvsadm ]

then

echo "ipvsadm stopped!"

exit 1

else

echo "ipvsadm started!"

fi

;;

*)

echo "Usage: $0 {start | stop | status}"

exit 1

esac

exit 0
```

2、也可以在 LVS 单独执行绑定的 VIP 语句：

VIP=192.168.111.190

ifconfig eth0:0 $VIP netmask 255.255.255.255 broadcast $VIP

/sbin/route add -host $VIP dev eth0:0

参数说明：

-A 增加一台虚拟服务器 VIP 地址。

-t 虚拟服务器提供的是 tcp 服务。

-s 使用的调度算法。

-a 在虚拟服务器中增加一台后端真实服务器。

-r 指定真实服务器地址。

-m 设置当前转发方式为 NAT 模式；-g 为直接路由模式；-i 模式为隧道模式。

-w 后端真实服务器的权重。

查看 LVS 转发列表命令为：ipvsadm –Ln

```
[root@node2 ~]#
[root@node2 ~]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  192.168.149.129:80 rr
  -> 192.168.149.131:80           Masq    2      0          0
  -> 192.168.149.130:80           Masq    2      0          0
[root@node2 ~]#
```

客户端 realserver 配置 VIP 脚本：

```
#!/bin/sh

#LVS Client Server

VIP=192.168.33.188

case  $1  in

start)

    ifconfig lo:0 $VIP netmask 255.255.255.255 broadcast $VIP

    /sbin/route add -host $VIP dev lo:0

    echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore

    echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
```

```
        echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
        sysctl -p >/dev/null 2>&1
        echo "RealServer Start OK"
        exit 0
;;
stop)
        ifconfig lo:0 down
        route del $VIP >/dev/null 2>&1
        echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore
        echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce
        echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
        echo "RealServer Stoped OK"
        exit 1
;;
*)
        echo "Usage: $0 {start|stop}"
;;
esac
```

我们会发现，如果这台 LVS 发生突发情况，down 机了，那后端所有的应用程序都访问不了。如何避免这种问题呢，这里需要用到故障切换，也就是如果有一台备用的 LVS 就

好了，主 down 了，自动切换到从，怎么实现这个需求，接下来讲解的 keepalived 软件就是专门用来做故障检测及切换的。

Keepalived 基于三层检测（IP 层，TCP 层，及应用层），主要用于检测 web 服务器的状态，如果有一台 web 服务器死机，或工作出现故障，Keepalived 检测到并将有故障的 web 服务器从系统中剔除；

当 web 服务器工作正常后 Keepalived 自动将 web 服务器加入到服务器群中，这些工作全部自动完成，不需要人工干涉，需要人工做的只是修复故障的 web 服务器。

需要注意一点，如果使用了 keepalived.conf 配置，就不需要再执行 ipvs –A 命令去添加均衡的 realserver 命令了，所有的配置都会在 keepalived.conf 里面，一个配置文件搞定所有，即只需要安装 ipvs 模块。

21.6 LVS+keepalived 高可用负载均衡

- Keepalived 安装配置

官方下载 keepalived 相应稳定版本：

cd /usr/src ;wget -c http://www.keepalived.org/software/keepalived-1.1.15.tar.gz

tar -xzvf keepalived-1.1.15.tar.gz &&cd keepalived-1.1.15 && ./configure && make && make install

安装完毕，配置 keepalived 服务为系统服务。

```
DIR=/usr/local/
cp $DIR/etc/rc.d/init.d/keepalived /etc/rc.d/init.d/ && cp $DIR/etc/sysconfig/keepalived /etc/sysconfig/ && mkdir -p /etc/keepalived && cp $DIR/sbin/keepalived /usr/sbin/
```

在 MASTER 上/etc/keepalived/目录创建 keepalived.conf 配置文件，并写入如下内容：

```
! Configuration File for keepalived
global_defs {
    notification_email {
        wgkgood@163.com
    }
    notification_email_from wgkgood@163.com
    smtp_server 127.0.0.1
    smtp_connect_timeout 30
    router_id LVS_DEVEL
}

# VIP1
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    lvs_sync_daemon_inteface eth0
    virtual_router_id 51
    priority 100
    advert_int 5
    nopreempt
    authentication {
```

```
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.33.188
    }
}
virtual_server 192.168.33.188 80 {
    delay_loop 6
    lb_algo wrr
    lb_kind DR
#     persistence_timeout 60
    protocol TCP
    real_server 192.168.33.12 80 {
        weight 100
        TCP_CHECK {
        connect_timeout 10
        nb_get_retry 3
        delay_before_retry 3
        connect_port 80
        }
    }
```

```
    real_server 192.168.33.13 80 {
        weight 100
        TCP_CHECK {
        connect_timeout 10
        nb_get_retry 3
        delay_before_retry 3
        connect_port 80


        }
    }
}
```

如上配置文件，红色标记的地方需要注意，state 状态主服务器设置 MASTER，从设置为 BACKUP，优先级备机设置比 MASTER 小，例如设置 90，使用 TCP 端口检测。

在 LVS BACKUP 服务器写入如下配置，需要注意的是客户端的配置要修改优先级及状态：

```
! Configuration Filefor keepalived
global_defs {
  notification_email {
   wgkgood@163.com
  }
  notification_email_from wgkgood@163.com
  smtp_server 127.0.0.1
  smtp_connect_timeout 30
```

```
  router_id LVS_DEVEL
}
# VIP1
vrrp_instance VI_1 {
  state BACKUP
  interface eth0
  lvs_sync_daemon_inteface eth0
  virtual_router_id 51
  priority 90
  advert_int 5
  authentication {
    auth_type PASS
    auth_pass 1111
  }
  virtual_ipaddress {
    192.168.33.11
  }
}
#REAL_SERVER_1
virtual_server 192.168.33.11 80 {
  delay_loop 6
  lb_algo wlc
```

```
  lb_kind DR
  persistence_timeout 60
  protocol TCP
  real_server 192.168.33.130 80 {
    weight 100
    TCP_CHECK {
    connect_timeout 10
    nb_get_retry 3
    delay_before_retry 3
    connect_port 80
    }
}
#REAL_SERVER_2
  real_server 192.168.33.131 80 {
    weight 100
    TCP_CHECK {
    connect_timeout 10
    nb_get_retry 3
    delay_before_retry 3
    connect_port 80
    }
  }
```

}

如上设置，LVS 主备配置完毕，接下来需要在 realserver 配置 LVS VIP，为什么要在 realserver 绑定 VIP 呢?

客户端访问 director 的 VIP，director 接收请求，将通过相应的算法将请求转发给相应的 realserver。在转发的过程中，会修改请求包的目的 mac 地址，目的 ip 地址不变。

Realserver 接收请求，并直接响应客户端。这时便出现一个问题，director 此时与 realserver 位于同一个网络中，当 director 直接将请求转发给 realserver 时，realserver 检测到该请求包的目的 ip 是 vip 而并非自己，便会丢弃，而不会响应。为了解决这个问题，所以需要在所有 Realserver 上都配上 VIP。

为什么一定要配置在 lo 接口上呢?

在 realserver 上的 lo 口配置 VIP,这样限制了 VIP 不会在物理交换机上产生 MAC 地址表，从而避免 IP 冲突。

21.7 LVS+keepalived 客户端脚本讲解

客户端启动 Realserver.sh 脚本内容：

```
#!/bin/sh
#LVS Client Server
VIP=192.168.33.188
case  $1  in
start)
    ifconfig lo:0 $VIP netmask 255.255.255.255 broadcast $VIP
    /sbin/route add -host $VIP dev lo:0
```

```
        echo "1" >/proc/sys/net/ipv4/conf/lo/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/lo/arp_announce
        echo "1" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "2" >/proc/sys/net/ipv4/conf/all/arp_announce
        sysctl -p >/dev/null 2>&1
        echo "RealServer Start OK"
        exit 0
;;
stop)
        ifconfig lo:0 down
        route del $VIP >/dev/null 2>&1
        echo "0" >/proc/sys/net/ipv4/conf/lo/arp_ignore
        echo "0" >/proc/sys/net/ipv4/conf/lo/arp_announce
        echo "0" >/proc/sys/net/ipv4/conf/all/arp_ignore
        echo "0" >/proc/sys/net/ipv4/conf/all/arp_announce
        echo "RealServer Stoped OK"
        exit 1
;;
*)
        echo "Usage: $0 {start|stop}"
;;
esac
```

21.8 LVS 负载均衡企业实战排错经验

当我们熟练构建 LVS 负载均衡配置了，在企业真实环境中，如何去排错呢，遇到问题我们该怎么办呢?

LVS 使用过程中，我们都会遇到很多的问题，但是遇到问题后，我们需要该如何处理呢？那这里分享我的解决思路。

LVS+Keepalived+Nginx 架构中，某天突然发现网站 www.jfedu.net 部分用户访问巨慢，甚至无法访问，那这个问题我们该如何定位呢？分两种情况：如果有监控，如果有报警短信再好不过了。然后可以很快的定位到某一台机器。如果没有监控，或者其他的原因没法看监控，那我们该如何排查呢？思路是什么呢?

1) 首先我们想到 ping www.jfedu.net ，通过 ping 返回数据部正常。DNS

2) 登录 LVS 服务器，ipvsadm –Ln 查看当前后端 web 连接信息，显示如下：

```
[root@LVS-Master keepalived]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
-> RemoteAddress:Port               Forward Weight ActiveConn InActConn
TCP  192.168.1.10:80 wlc
-> 192.168.1.6:80                   Route   100    2          13
-> 192.168.1.5:80                   Route   100    120        13
-> 192.168.1.4:80                   Route   100    1363       45
```

通过 LVS 信息,我们看到 LVS 选择的轮训方式为加权最少连接,而网站也是部分无法访问，猜测是其中一台 web 服务器无法访问或者访问巨慢导致，我们会想难道 LVS 不会自己判断吗？想法很好，那我们接下来查看 keepalived.conf 配置，部分截图如下：

```
real_server 192.168.1.4    80    {


                weight 100

                TCP_CHECK {

                connect_timeout 10

                nb_get_retry 3

                delay_before_retry 3

                connect_port 80

                }

}
```

通过配置文件我们发现 LVS 默认用的是 TCP 检测方式，只要 80 端口能通，请求就会转发到后端服务器。紧接着在 LVS /tmp 目 wget http://192.168.1.4/ 返回 502 超时，另外几台 nginx 返回正常，1.4 服务器 80 端口对于 LVS 来说是打开的，所以 LVS 会把请求转发给给它。

这就造成了为什么部分用户可以访问，有的用户无法访问的问题。登录 1.4 nginx 服务器，pkill nginx ，临时停止 nginx 保证服务保证正常访问，然后再查看 nginx 日志发现是后端程序连接一台数据库出现的问题。

回过头来我们会发现，LVS 不会检测你后端 502 超时错误，只关心 80 端口是否开启对于应用来说，这样检测明显不足，那我们需要如何处理呢？增加 LVS 对后端 Nginx URL 的检测，能访问 URL 则表示服务正常，直接看代码：

```
real_server 192.168.1.4 80 {
```

```
            weight 100

            HTTP_GET {

            url {

            path /monitor/warn.jsp

            status_code 200

            }

            connect_timeout 10

            nb_get_retry 3

            delay_before_retry 3

        }

}
```

我们对比之前的检测方式，从单纯的 80 端口到现在的 URL 检测，后端如果某台出现 502 超时错误，LVS 会自动踢出，等后端恢复后自动添加。自此这个故障就解决完毕，当然

还有很多故障，例如如下的问题：

某一天 www.jfedu.net 网站突然有用户反馈报 404，而我在本地访问正常，这又是怎么回事呢？

……

总结：LVS 网站故障排查经验

如果发现主网站无法访问，首先第一步 ping 网站域名是否能 ping 通，如果域名无法访问，试着使用 IP 能不能访问，如果 IP 能访问，首先排查到域名解析问题。

如果 IP 也无法访问，登录 LVS 服务器，使用命令 ipvsadm –Ln 查看当前连接状态和查看/var/log/messages 日志信息，可以在 LVS 上访问 realserver ip，进行排查。

如果 LVS 服务正常，后端 realserver 服务异常，然后查看 nginx 日志信息，是否有大量恶意访问，临时重启看是否能访问。

如果有恶意 ip 访问，找出恶意 ip，经确认可以关闭后，使用 iptables 防火墙临时关闭即可。

第22章 Linux 服务器安全加固篇

22.1 IPtables 入门简介

Iptables 是 Linux 内核集成的 IP 信息包过滤系统。如果 Linux 系统连接到因特网或 LAN、服务器或连接 LAN 和因特网的代理服务器，则该系统有利于在 Linux 系统上更好地控制 IP 信息包过滤和防火墙配置。

防火墙在做信息包过滤决定时，有一套遵循和组成的规则，这些规则存储在专用的**信息包过滤表**中，而这些表集成在 Linux 内核中。在信息包过滤表中，规则被分组放在我们所谓的链（chain）中。而 netfilter/iptables IP 信息包过滤系统是一款功能强大的工具，可用于添加、编辑和移除规则。

虽然 netfilter/iptables IP 信息包过滤系统被称为单个实体，但它实际上由两个组件 netfilter 和 iptables 组成。

netfilter 组件也称为内核空间（kernelspace），是内核的一部分，由一些信息包过滤表组成，这些表包含内核用来控制信息包过滤处理的规则集。

iptables 组件是一种工具，也称为用户空间（userspace），它使插入、修改和除去信息包过滤表中的规则变得容易。

22.2 IPtables 表与链功能详解

Iptables 的规则链分为三种：输入、转发和输出。

输入——这条链用来过滤目的地址是本机的连接。例如，如果一个用户试图使用 SSH 登陆到你的 PC/服务器，iptables 会首先匹配其 IP 地址和端口到 iptables 的输入链规则。

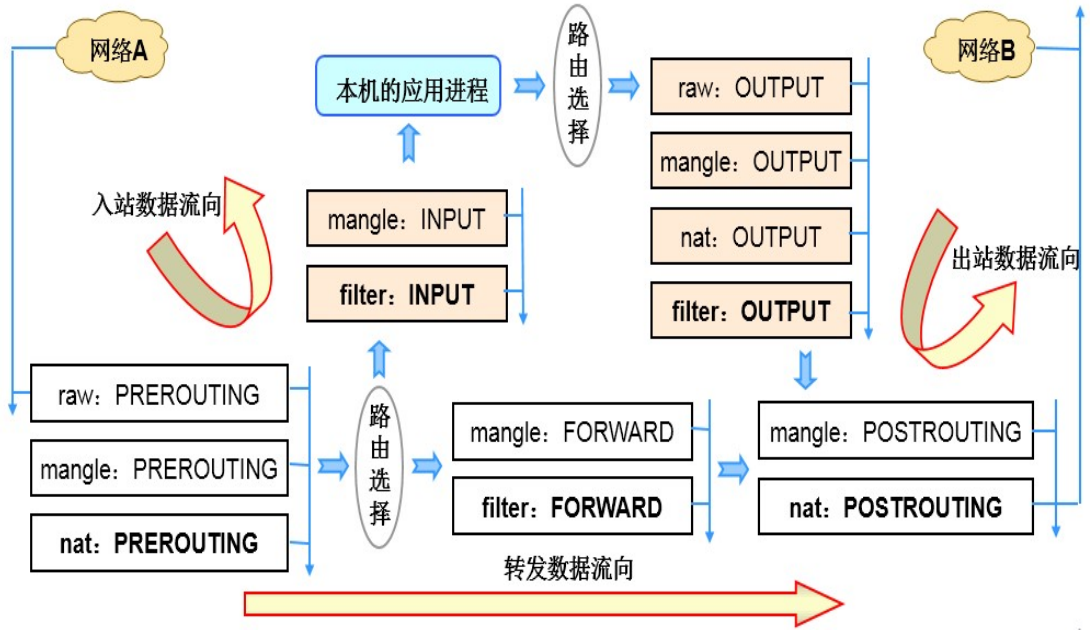
转发——这条链用来过滤目的地址和源地址都不是本机的连接。例如，路由器收到的绝大数数据均需要转发给其它主机。如果你的系统没有开启类似于路由器的功能，如 NATing，你就不需要使用这条链。

输出——这条链用来过滤源地址是本机的连接。例如，当你尝试 ping

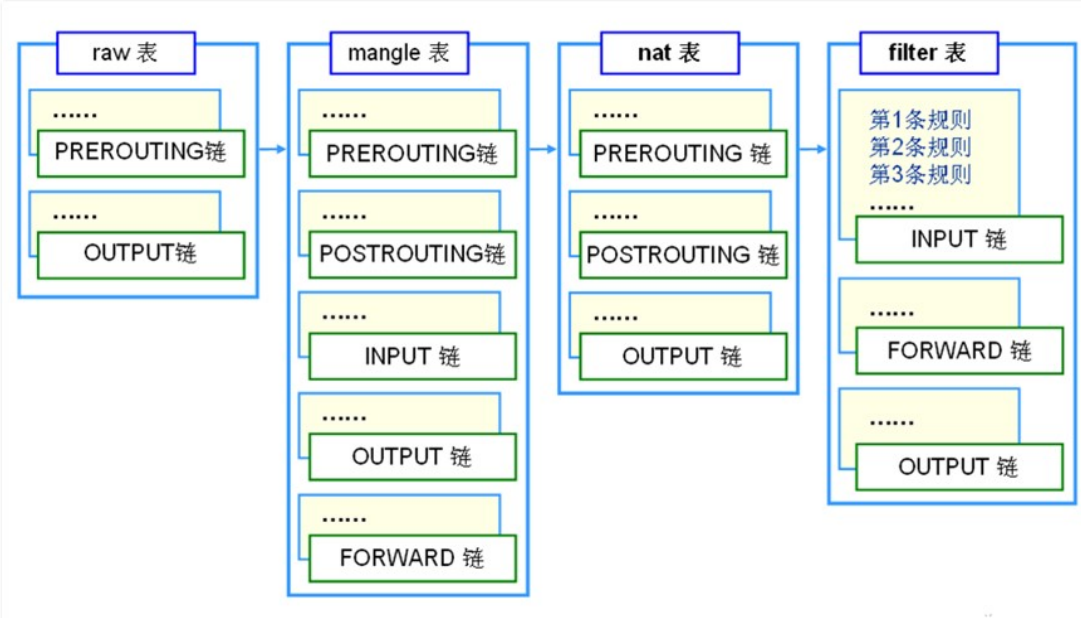
jfedu.net 时，iptables 会检查输出链中与 ping 和 jfedu.net 相关的规则，然后决定允许还是拒绝你的连接请求。

注意：当 ping 一台外部主机时，看上去好像只是输出链在起作用。但是请记住，外部主机返回的数据要经过输入链的过滤。当配置 iptables 规则时，请牢记许多协议都需要双向通信，所以你需要同时配置输入链和输出链。人们在配置 SSH 的时候通常会忘记在输入链和输出链都配置它。

IPTABLES 数据包流程:



IPTABLES 四张表，五条链接：



iptables 具有 Filter, NAT, Mangle, Raw 四种内建表。

22.3 IPtables Filter 表详解

Filter 表示 iptables 的默认表，因此如果你没有自定义表，那么就默认使用 filter 表，它具有以下三种内建链：

- **INPUT 链 – 处理来自外部的数据。**
- **OUTPUT 链 – 处理向外发送的数据。**
- **FORWARD 链 – 将数据转发到本机的其他网卡设备上。**

22.4 IPtables NAT 表详解

NAT 表有三种内建链：

- **PREROUTING 链** – 处理刚到达本机并在路由转发前的数据包。它会转换数据包中的目标 IP 地址（destination ip address），通常用于 DNAT(destination NAT)。
- **POSTROUTING 链** – 处理即将离开本机的数据包。它会转换数据包中的源 IP 地址（source ip address），通常用于 SNAT（source NAT）。

- **OUTPUT 链** – 处理本机产生的数据包。

22.5 IPtables Mangle 表详解

Mangle 表用于指定如何处理数据包。它能改变 TCP 头中的 QoS 位。Mangle 表具有 5 个内建链：

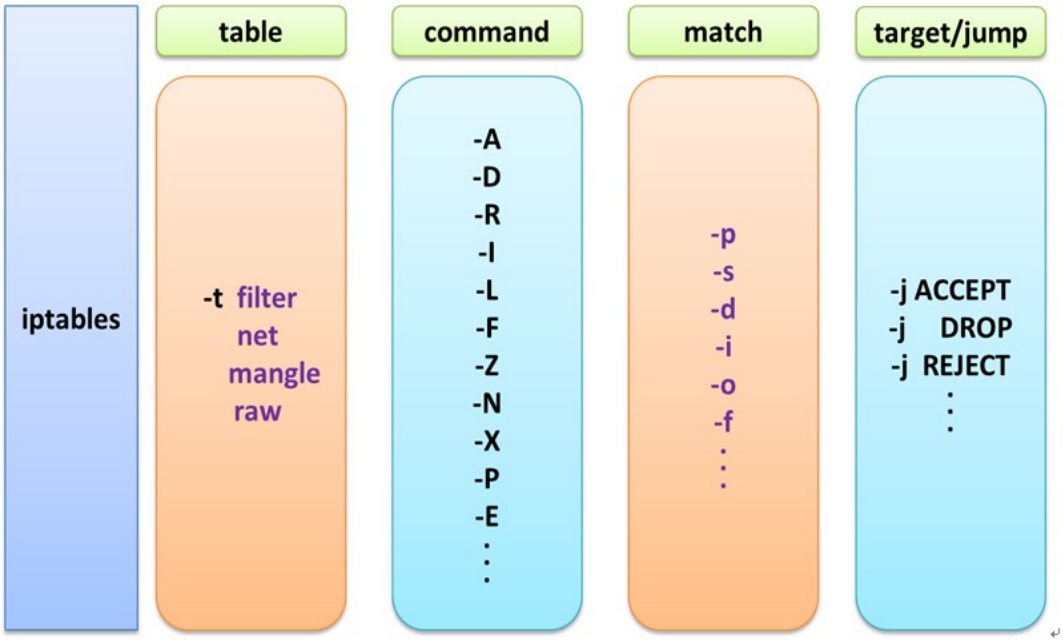
- PREROUTING
- OUTPUT
- FORWARD
- INPUT
- POSTROUTING

22.6 IPtables RAW 表详解

Raw 表用于处理异常，它具有 2 个内建链：

- PREROUTING chain
- OUTPUT chain

22.7 IPtables 常用命令图解



22.8 IPtables 常用命令剖析

1.命令：

-A 顺序添加，添加一条新规则

-I 插入，插入一条新规则 -I 后面加一数字表示插入到哪行

-R 修改， 删除一条新规则 -D 后面加一数字表示删除哪行

-D 删除，删除一条新规则 -D 后面加一数字表示删除哪行

-N 新建一个链

-X 删除一个自定义链,删除之前要保证次链是空的,而且没有被引用

-L 查看

@1.iptables -L -n 以数字的方式显示

@2. iptables -L -v 显示详细信息

@3. iptables -L -x 显示精确信息

-E　重命名链

-F 清空链中的所有规则

-Z　清除链中使用的规则

-P 设置默认规则

2.匹配条件：

隐含匹配：

-p　tcp udp icmp

--sport 指定源端口

--dport 指定目标端

-s 源地址

-d 目的地址

-i 数据包进入的网卡

-o 数据包出口的网卡

扩展匹配：

-m state --state　匹配状态的

-m mutiport --source-port　端口匹配 ,指定一组端口

-m limit --limit 3/minute　每三分种一次

-m limit --limit-burst　5　只匹配 5 个数据包

-m string --string --algo bm|kmp --string"xxxx"　匹配字符串

-mtime--timestart 8:00 --timestop 12:00　表示从哪个时间到哪个时间段

-mtime--days　　表示那天

-m mac --mac-sourcexx:xx:xx:xx:xx:xx 匹配源 MAC 地址

-m layer7 --l7proto qq　表示匹配腾讯 qq 的 当然也支持很多协议,这个默认是没有的,需要我们给内核打补丁并重新编译内核及 iptables 才可以使用 -m layer7 这个显示扩展匹配

3.动作:

-j

DROP 直接丢掉

ACCEPT 允许通过

REJECT 丢掉，但是回复信息

LOG --log-prefix"说明信息,自己随便定义"， 记录日志

SNAT　　源地址转换

DNAT　　目标地址转换

REDIRECT　重定向

MASQUERAED　地址伪装

保存 iptables 规则

service iptables save

重启 iptables 服务

service iptables stop

service iptables start

22.9 IPtables 企业案例解析

```
# Firewall configuration written by system-config-firewall
# Manual customization of this file is not recommended.
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
#-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 3306 -j ACCEPT
-A INPUT -s 180.171.238.0/24 -m state --state NEW -m tcp -p tcp --dport 80 -j DROP
-A INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT
~
~
~
~
```

含义：

:INPUT ACCEPT [0:0]# 该规则表示 INPUT 表默认策略是 ACCEP

:FORWARD ACCEPT [0:0]# 该规则表示 FORWARD 表默认策略是 ACCEPT

:OUTPUT ACCEPT [0:0]# 该规则表示 OUTPUT 表默认策略是 ACCEPT

-A INPUT -m state –state ESTABLISHED,RELATED -j ACCEPT# 意思是允许进入的数据包只能是刚刚我发出去的数据包的回应，ESTABLISHED：已建立的链接状态。RELATED：该数据包与本机发出的数据包有关。

-A INPUT -p icmp -j ACCEPT

-A INPUT -i lo -j ACCEPT# 意思就允许本地环回接口在 INPUT 表的所有数据通信，-i 参数是指定接口，接口是 lo，lo 就是 Loopback（本地环回接口）

-A INPUT -j REJECT –reject-with icmp-host-prohibited

-A FORWARD -j REJECT –reject-with icmp-host-prohibited

 # 这两条的意思是在 INPUT 表和 FORWARD 表中拒绝所有其他不符合上述任何一条规则的数据包。并且发送一条 host prohibited 的消息给被拒绝的主机。

下面来介绍一下，我添加的每个参数是什么意思，跟我没讲得允许 22 端口的一样

-A INPUT -m state –state NEW -m tcp -p tcp –dport 22 -j ACCEPT

-A 最后添加一条规则

-j 后面接动作，主要的动作有接受(ACCEPT)、丢弃(DROP)、拒绝(REJECT)及记录(LOG)

–dport 限制目标的端口号码。

-p 协定：设定此规则适用于哪种封包格式 主要的封包格式有： tcp, udp, icmp 及 all 。

-m state –state 模糊匹配一个状态，

NEW 用户发起一个全新的请求

ESTABLISHED 对一个全新的请求进行回应

RELATED 两个完整连接之间的相互关系，一个完整的连接，需要依赖于另一个完整的连接。

INVALID 无法识别的状态。

WEB 服务器,开启 80 端口.

[root@localhost ~]# iptables -A INPUT -p tcp --dport 80 -j ACCEPT

邮件服务器,开启 25,110 端口.

[root@localhost ~]# iptables -A INPUT -p tcp --dport 110 -j ACCEPT

[root@localhost ~]# iptables -A INPUT -p tcp --dport 25 -j ACCEPT

FTP 服务器,开启 21 端口

[root@localhost ~]# iptables -A INPUT -p tcp --dport 21 -j ACCEPT

[root@localhost ~]# iptables -A INPUT -p tcp --dport 20 -j ACCEPT

DNS 服务器,开启 53 端口

[root@localhost ~]# iptables -A INPUT -p tcp --dport 53 -j ACCEPT

允许 icmp 包通过,也就是允许 ping,

[root@localhost ~]# iptables -A OUTPUT -p icmp -j ACCEPT (OUTPUT 设置成 DROP 的话)

[root@localhost ~]# iptables -A INPUT -p icmp -j ACCEPT (INPUT 设置成 DROP 的话)

将本机的 8080 端口转发至其他主机，主机 IP：192.168.1.12，目标主机 IP 和端口：

192.168.1.13:8088，规则如下；

iptables -t nat -A PREROUTING -p tcp -m tcp --dport 8080 -j DNAT --to-destination

192.168.1.13:8088

iptables -t nat -A POSTROUTING -p tcp -m tcp --dport 8088 -j SNAT --to-source

192.168.1.12

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

同时开启 iptables forward 转发功能。

第23章 Squid 缓存服务器实战篇

23.1 Squid 服务器简介

随着网站访问人数越来越多，对体验的要求也越来越高，网站承受的并发和压力也越来越大，所以需要对网站和架构进行优化，优化的策略有很多，系统内核、程序、配置均衡、

加入缓存等，目前主流缓存服务器主要有 squid、varnish、nginx_cache、ATS（apache-trffic-server），那今天我们来讨论使用老牌 Squid 对架构进行缓存优化。

Squid cache (简称为 Squid) 是一个流行的自由软件，它符合 GNU 通用公共许可证。Squid 作为网页服务器的前置 cache 服务器，可以代理用户向 web 服务器请求数据并进行缓存；也可以用在局域网中，使局域网用户通过代理上网。Squid 主要设计用于在 Linux 一类系统运行。

23.2 Squid 服务器原理

简单的来说就是：用户请求 www 网站，经过 squid，squid 检查本地硬盘目录有没有这个文件的缓存；如果没有，squid 则去后端真实 web 服务器获取该页面，返回给用户，同时在自己本地缓存一份，如果另外一个用户再访问同样请求页面时，squid 直接从本地返回。

Squid 有 ufs, aufs, coss, diskd, null 五种存储机制，其中 ufs, aufs, diskd 都是在文件系统上面保存很多小文件，coss 是 squid 自己实现了一个简单的文件系统,可以使用一个大文件或者一个磁盘设备来存储。null 则是给不想要磁盘缓存的情况准备的，coss 看起来好像很不错，但是以前试验并不足够稳定,因此并不推荐使用。

对于一些老系统,使用 aufs 或者 diskd 是比较好的选择,如果系统的线程库比较好(如 Linux,Solaris),那么使用 aufs。

23.3 源码方式实战 Squid

安装 squid 也非常简单，可以用源码安装，也可以使用 rpm、yum 安装，这里使用 yum 安装，根据实际经验使用，可以考虑采用。

源码安装 squid 2.6.24 版本：squid-2.6.STABLE24.tar.bz2

tar -jxf squid-2.6.STABLE24.tar.bz2 ;cd squid-2.6.STABLE24

./configrue --prefix=/usr/local/squid/ --enable-storeio=ufs,diskd ;make ;make install

源码安装的配置文件内容如下：

http_port 80 accel vhost vport

cache_peer 192.168.33.12 parent 80 0 originserver name=wugk1

cache_peer 192.168.33.13 parent 80 0 originserver name=wugk2

cache_peer_domain wugk1 www.wugk1.com

cache_peer_domain wugk2 www.wugk2.com

visible_hostname localhost

forwarded_for off

via off

cache_vary on

acl manager proto cache_object

acl localhost src 127.0.0.1/32

acl to_localhost dst 127.0.0.0/8 0.0.0.0/32

acl localnet src 10.0.0.0/8

acl localnet src 172.16.0.0/12

acl localnet src 192.168.0.0/16

acl all src 0.0.0.0/0

acl CONNECT method CONNECT

```
http_access allow manager localhost

http_access deny manager

http_access allow localnet

http_access allow localhost

http_access allow all

acl PURGE method PURGE

http_access allow PURGE localhost

http_access deny PURGE

cache_mem 1000 MB

maximum_object_size 8 MB

maximum_object_size_in_memory 256 KB

hierarchy_stoplist cgi-bin ?

coredump_dir /usr/local/squid/var/cache

refresh_pattern ^ftp:           1440    20%     10080

refresh_pattern ^gopher:        1440    0%      1440

refresh_pattern -i (/cgi-bin/|\?) 0     0%      0

refresh_pattern                                 \.(jpg|png|gif|mp3|xml|html|htm|css|js)

1440    50%     2880    ignore-reload

refresh_pattern .               0       20%     4320
```

23.4 YUM 方式实战 Squid

CentOS 7 上 yum 安装 squid 方法：

yum install -y squid

Squid.conf 配置文件，内容如下：

```
http_port 80 accel vhost vport

cache_peer 192.168.33.130 parent 80 0 originserver name=wugk1

cache_peer 192.168.33.131 parent 80 0 originserver name=wugk2

cache_peer_domain wugk1 www.wugk1.com

cache_peer_domain wugk2 www.wugk2.com

visible_hostname localhost

forwarded_for off

via off

cache_vary on

#acl config

acl manager proto cache_object

acl localhost src 127.0.0.1/32

acl to_localhost dst 127.0.0.0/8 0.0.0.0/32

acl localnet src 10.0.0.0/8     # RFC1918 possible internal network

acl localnet src 172.16.0.0/12  # RFC1918 possible internal network

acl localnet src 192.168.0.0/16 # RFC1918 possible internal network

acl SSL_ports port 443

acl Safe_ports port 80 8080         # http

acl Safe_ports port 21          # ftp

acl Safe_ports port 443         # https
```

```
acl all src 0.0.0.0/0

acl CONNECT method CONNECT

http_access allow manager localhost

http_access deny manager

http_access deny !Safe_ports

http_access deny CONNECT !SSL_ports

http_access allow localnet

http_access allow localhost

http_access allow all

acl PURGE method PURGE

http_access allow PURGE localhost

http_access deny PURGE

#squid config 2014-03-25

cache_dir aufs /data/cache1 10240 16 256

cache_mem 4000 MB

maximum_object_size 8 MB

maximum_object_size_in_memory 256 KB

hierarchy_stoplist cgi-bin ?

coredump_dir /var/spool/squid

refresh_pattern ^ftp:           1440    20%     10080

refresh_pattern ^gopher:        1440    0%      1440

refresh_pattern -i (/cgi-bin/|\?) 0     0%      0
```

refresh_pattern \.(jpg|png|gif|mp3|xml|html|htm|css|js)

1440 50% 2880 ignore-reload

refresh_pattern . 0 20% 4320

23.5 Squid 配置参数剖析

#vhost 和 vport 表示支持虚拟主机和虚拟端口，如果再加上 transparent 表示支持透明代理

http_port 80 accel vhost vport

#cache_peer 表示如果本机缓存中找不到客户端请求的数据,则与后端主机联系,以 parent 类型进行联系;

使用 HTTP 协议进行联系，联系端口是 80，orginserver 表示此服务器是源服务器，name 表示别名。

cache_peer 192.168.1.103 parent 80 0 originserver name=wugk1

cache_peer 192.168.33.11 parent 80 0 originserver name=wugk2

#设置别名所对应的域名,如果 cache_peer 中使用域名而不是 IP 的话;

那么 cache_peer_domain 中一定要用相同的域名，否则无法访问。

cache_peer_domain wugk1 www.wugk1.com

cache_peer_domain wugk2 www.wugk2.com

#设置缓存服务器名称

visible_hostname localhost

forwarded_for off

via off

```
cache_vary on

#acl config

acl manager proto cache_object

acl localhost src 127.0.0.1/32

acl to_localhost dst 127.0.0.0/8 0.0.0.0/32

acl localnet src 10.0.0.0/8        # RFC1918 possible internal network

acl localnet src 172.16.0.0/12    # RFC1918 possible internal network

acl localnet src 192.168.0.0/16 # RFC1918 possible internal network

acl SSL_ports port 443

acl Safe_ports port 80 8080              # http

acl Safe_ports port 21               # ftp

acl Safe_ports port 443              # https

acl all src 0.0.0.0/0

acl CONNECT method CONNECT

http_access allow manager localhost

http_access deny manager

http_access deny !Safe_ports

http_access deny CONNECT !SSL_ports

http_access allow localnet

http_access allow localhost

##设置访问控制,允许所有客户端访问上面设置的两个网站

http_access allow all
```

#支持 purge 方式清除缓存

acl PURGE method PURGE

http_access allow PURGE localhost

http_access deny PURGE

#squid config 2014-03-25

#设置缓存文件夹的路径和参数，缓存机制为 aufs，10240 表示 10G，目录下面分为 16 级，每级有 256 个目录

cache_dir aufs /data/cache1 10240 16 256

#设置缓存内存大小，最大内存为 4g

cache_mem 4000 MB

#设置硬盘中可缓存的最大文件大小

maximum_object_size 8 MB

#设置内存中可缓存的最大文件大小

maximum_object_size_in_memory 256 KB

hierarchy_stoplist cgi-bin ?

#当 squid 突然挂掉的时候，或者突然出现什么故障的时候，将 squid 在内存中的资料写到硬盘中

coredump_dir /var/spool/squid

#<refresh_pattern> <regex> <最小时间> <百分比> <最大时间>

#refresh_pattern 用于确定缓存的类型,缓存过期时间,及百分比。

#如果希望内容缓存 cache 后不删除，直到被主动用 purge 清除，可以加 ignore-reload 选项

refresh_pattern ^ftp: 1440 20% 10080

refresh_pattern ^gopher: 1440 0% 1440

refresh_pattern -i (/cgi-bin/|\?) 0 0% 0

refresh_pattern \.(jpg|png|gif|mp3|xml|html|htm|css|js) 1440 50% 2880

ignore-reload

refresh_pattern . 0 20% 4320

23.6 Squid 必备命令实战

Squidclient 可以用来清理 squid 缓存，查看 squid 状态等功能。

使用方法：

*取得 squid 运行状态信息： squidclient -p 80 mgr:info

*取得 squid 内存使用情况： squidclient -p 80 mgr:mem

*取得 squid 已经缓存的列表： squidclient -p 80 mgr:objects

*取得 squid 的磁盘使用情况： squidclient -p 80 mgr:diskd

*强制更新某个 url：

squidclient -p 80 -m PURGE

http://www.wugk1.com/static/image/common/nv_a.png

查看缓存命中率：squidclient -p 80 mgr:info |egrep "(Request Hit Ratios|Byte Hit Ratios)"

第24章　Linux 自动化运维实战篇

24.1　自动化运维工具简介

曾有媒体报道，Facebook 一个运维人员管理上万台服务器，如果使用手工的方法去维护是很难做到的，基于自动化工具就可以轻松的实现管理上万台、甚至十万台。

如下为 IT 运维主流自动化管理工具 Puppet、saltstack、Ansible 各自优缺点：

24.2　Puppet 自动运维工具特点

Puppet 是早期的 Linux 自动化运维工具，是一种 Linux、Unix、Windows 平台的集中配置管理系统，发展至今目前已经非常成熟，可以批量管理远程服务器，模块丰富，配置复杂，基于 Ruby 语言编写。最典型的 C/S 模式，需要安装服务端与客户端。

puppet 采用 C/S 星状的结构，所有的客户端和一个或几个服务器交互，每个客户端周期的（默认半个小时）向服务器发送请求，获得其最新的配置信息，保证和该配置信息同步。

每个 puppet 客户端每半小时(可以设置)连接一次服务器端，下载最新的配置文件,并且严格按照配置文件来配置客户端. 配置完成以后，puppet 客户端可以反馈给服务器端一个消息，如果出错也会给服务器端反馈一个消息。

Puppet 适用于服务器管理的整个过程，比如初始安装、配置、更新以及系统下线。

24.3　Saltstack 自动运维工具特点

Saltstack 与 Puppet 均是 C/S 模式，需安装服务端与客户端，基于 Python 编写，加入 MQ 消息同步，可以使执行命令和执行结果高效返回，但其执行过程需等待客户端全部返回，如果客户端未及时返回或未响应的话，可能会导致部分机器没有执行结果。

24.4 Ansible 自动运维工具特点

Ansible 与 Saltstack 均是基于 Python 语言开发，Ansible 只需要在一台普通的服务器上运行即可，不需要在客户端服务器上安装客户端。因为 Ansible 是基于 SSH 远程管理，而 Linux 服务器大都离不开 SSH，所以 Ansible 不需要为配置工作添加额外的支持。

Ansible 安装使用非常简单，而且基于上千个插件和模块实现各种软件、平台、版本的管理，支持虚拟容器多层级的部署。很多读者在使用 Ansible 工具时，认为 Ansible 比 Saltstatck 执行效率慢，其实不是软件本身慢，是由于 SSH 服务慢，可以优化 SSH 连接速度及使用 Ansible 加速模块，满足企业上万台服务器的维护和管理。

24.5 Ansible 运维工具原理

Ansible 是一款极为灵活的开源工具套件，能够大大简化 Unix 管理员的自动化配置管理与流程控制方式。它利用推送方式对客户系统加以配置，这样所有工作都可在主服务器端完成。其命令行机制同样非常强大，允许大家利用商业许可 Web UI 实现授权管理与配置。

可以通过命令行或者 GUI 来使用 Ansible，运行 Ansible 的服务器这里俗称“管理节点”；通过 Ansible 进行管理的服务器俗称“受控节点”。权威媒体报道 Ansible 于 2015 年被 Red Hat 公司 1.5 亿美元收购，新版 Red Hat 内置 Ansible 软件。

本书以 Ansible 为案例，基于 Ansible 构建企业自动化运维平台，实现大规模服务器的快速管理和部署。Ansible 将平常复杂的配置工作变得简单，变得更加标准化更容易控制。

Ansible 自动运维管理工具优点：

- 轻量级，更新时，只需要在操作机上进行一次更新即可；
- 采用 SSH 协议；
- 不需要去客户端安装 agent；

- 批量任务执行可以写成脚本，而且不用分发到远程就可以执行；
- 使用 python 编写的，维护更简单；
- 支持 sudo 普通用户命令；
- 去中心化管理。

Ansible 自动运维管理工具工作原理拓扑，如图 21-1 所示：

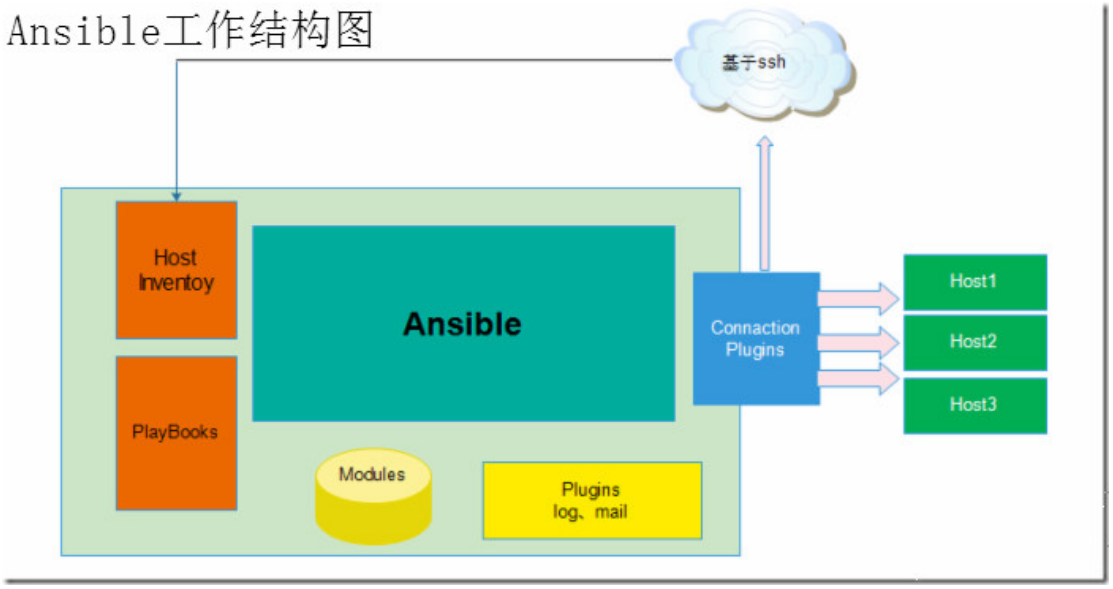


图 21-1 Ansible 工作原理图

24.6 Ansible 管理工具安装配置

Ansible 可以工作在 Linux、BSD、Mac OS X 等平台，对 Python 环境的版本最低要求为 Python2.6 以上，如果操作系统 Python 软件版本为 2.4，需要升级方可使用 Ansible 工具。

Red Hat、CentOS 操作系统可以直接基于 YUM 工具自动安装 Ansible，CentOS6.x 或者 CentOS7.x 安装前，需先安装 epel 扩展源，代码如下：

```
rpm -Uvh
http://mirrors.ustc.edu.cn/fedora/epel/6/x86_64/epel-release-6-8.noarch.rpm
yum install epel-release -y
```

```
yum    install    ansible          -y
```

Ansible 工具默认主目录为：/etc/ansible/，其中 hosts 文件为被管理机 IP 或者主机名列表，ansible.cfg 为 ansible 主配置文件，roles 为角色或者插件路径，默认该目录为空，如图 21-2 所示：

```
[root@localhost ansible]# ls
ansible.cfg  hosts  roles
[root@localhost ansible]# ll
total 28
-rw-r--r--. 1 root root 17718 May 22 20:59 ansible.cfg
-rw-r--r--. 1 root root   410 May 22 20:17 hosts
drwxr-xr-x. 2 root root  4096 Apr 20 05:08 roles
[root@localhost ansible]#
[root@localhost ansible]#
[root@localhost ansible]# pwd
/etc/ansible
[root@localhost ansible]#
[root@localhost ansible]# ls
ansible.cfg  hosts  roles
[root@localhost ansible]#
```

图 21-2 Ansible 主目录信息

Ansible 远程批量管理，其中执行命令是通过 Ad-Hoc 来完成，也即点对点单条执行命令，能够快速执行，而且不需要保存执行的命令。默认 hosts 文件配置主机列表，可以配置分组，可以定义各种 ip 及规则，hosts 列表默认配置如图 21-3 所示：

```
# This is the default ansible 'hosts' file.
## green.example.com
## blue.example.com
## 192.168.100.1
## 192.168.100.10

## [webservers]
## alpha.example.org
## beta.example.org
## 192.168.1.100
## 192.168.1.110

## www[001:006].example.com

# Ex 3: A collection of database servers in the 'dbservers' group
```

图 21-3 Hosts 主机列表文件内容

Ansible 基于多模块管理，常用的 Ansible 工具管理模块包括：command、shell、script、yum、copy、File、async、docker、cron、mysql_user、ping、sysctl、user、acl、add_host、

easy_install、haproxy 等。

可以使用 ansible-doc –l|more 查看 ansible 支持的模块，也可以查看每个模块的帮助文档，ansible-doc　module_name，如图 21-4 所示：

```
[root@www ~]# ansible-doc docker
> DOCKER    (/usr/lib/python2.7/site-packages/ansible/modules/cloud/doc

  This is the original Ansible module for managing the Docker container
  Additional and newer modules are available. For the latest on orchest
  Ansible visit our Getting Started with Docker Guide at
  https://github.com/ansible/ansible/blob/devel/docsite/rst/guide_docke

DEPRECATED:
In 2.2 use M(docker_container) and M(docker_image) instead.

Options (= is mandatory):

- cap_add
        Add capabilities for the container. Requires docker-py >= 0.5.0
        [Default: False]
```

图 21-4 Ansible-doc docker 帮助信息

24.7 Ansible 工具参数详解

基于 Ansible 批量管理之前，需将被管理的服务器 IP 列表添加至/etc/ansible/hosts 文件中，如图 21-5 添加 4 台被管理端 IP 地址，分成 web 和 db 两组，本机也可以是被管理机。

```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups
[web]
192.168.149.128
192.168.149.129
[db]
192.168.149.130
192.168.149.131
```

图 21-5 Ansible Hosts 主机列表

基于 Ansible 自动运维工具管理客户端案例操作，由于 Ansible 管理远程服务器基于

SSH，在登录远程服务器执行命令时需要远程服务器的用户名和密码，也可以加入-k 参数手动输入密码或者基于 ssh-keygen 生成免秘钥。

Ansible 自动化批量管理工具主要参数如下：

-v,–verbose	打印详细模式；
-i PATH,–inventory=PATH	指定 host 文件路径；
-f NUM,–forks=NUM	指定 fork 开启同步进程的个数，默认 5；
-m NAME,–module-name=NAME	指定 module 名称，默认模块 command；
-a MODULE_ARGS	module 模块的参数或者命令；
-k,–ask-pass	输入远程被管理端密码；
–sudo	基于 sudo 用户执行；
-K,–ask-sudo-pass	提示输入 sudo 密码与 sudo 一起使用；
-u USERNAME,–user=USERNAME	指定移动端的执行用户；
-C,–check	测试执行过程，不改变真实内容，相当于预演；
-T TIMEOUT,	执行命令超时时间，默认为 10 秒；
--version	查看 Ansible 软件版本信息。

24.8 Ansible ping 模块实战

Ansible 最基础的模块为 ping 模块，主要用于判断远程客户端是否在线，用于 ping 本身服务器，返回值为 changed、ping。

Ansible ping 模块企业常用案例如下：

（1） Ansible ping 服务器状态，如图 21-6 所示：

```
ansible -k all   -m ping
```

```
[root@localhost ~]# ansible -k all  -m ping
SSH password:
192.168.149.129 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.149.130 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.149.131 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.149.128 | SUCCESS => {
```

图 21-6 Ansible ping 服务器状态

24.9 Ansible command 模块实战

Ansible command 模块为 ansible 默认模块，主要用于执行 Linux 基础命令，可以执行远程服务器命令执行、任务执行等操作。Command 模块使用详解：

Chdir	执行命令前，切换到目录；
Creates	当该文件存在时，则不执行该步骤；
Executable	换用 shell 环境执行命令；
Free_form	需要执行的脚本；
Removes	当该文件不存在时，则不执行该步骤；
Warn	如果在 ansible.cfg 中存在告警，如果设定了 False，不会警告此行。

Ansible command 模块企业常用案例如下：

（1） Ansible command 模块远程执行 date 命令，执行结果如图 21-7 所示：

```
ansible -k -i /etc/ansible/hosts   all   -m command -a   "date"
```

```
[root@localhost ansible]# ansible -k -i /etc/ansible/hosts  all  -m co
SSH password:
192.168.149.128 | SUCCESS | rc=0 >>
Wed May 31 00:24:43 CST 2017

192.168.149.131 | SUCCESS | rc=0 >>
Sat May 20 09:34:37 CST 2017

192.168.149.130 | SUCCESS | rc=0 >>
Wed May 24 21:47:33 CST 2017

192.168.149.129 | SUCCESS | rc=0 >>
Wed May 31 00:25:09 CST 2017

[root@localhost ansible]#
```

图 21-7 Ansible command date 命令执行结果

（2） Ansible command 模块远程执行 ping 命令，执行结果如图 21-8 所示：

```
ansible  -k  all  -m command -a  "ping  -c  1  www.baidu.com"
```

```
[root@localhost ansible]# ansible -k all  -m command -a  "ping -c 1 w
SSH password:
192.168.149.129 | SUCCESS | rc=0 >>
PING www.a.shifen.com (119.75.218.70) 56(84) bytes of data.
64 bytes from 119.75.218.70: icmp_seq=1 ttl=128 time=6.49 ms

--- www.a.shifen.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 12ms
rtt min/avg/max/mdev = 6.498/6.498/6.498/0.000 ms

192.168.149.130 | SUCCESS | rc=0 >>
PING www.a.shifen.com (119.75.217.109) 56(84) bytes of data.
64 bytes from 119.75.217.109: icmp_seq=1 ttl=128 time=7.25 ms

--- www.a.shifen.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 22ms
rtt min/avg/max/mdev = 7.250/7.250/7.250/0.000 ms
```

图 21-8 Ansible command ping 命令执行结果

（3） Ansible Hosts 正则模式远程执行 df -h，执行结果如图 21-9 所示：

```
ansible -k  192.168.149.13*  -m  command  -a  "df -h"
```

```
[root@localhost ansible]# ansible -k  192.168.149.13*  -m  command
SSH password:
192.168.149.131 | SUCCESS | rc=0 >>
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        30G   15G   14G  52% /
tmpfs           242M     0  242M   0% /dev/shm
/dev/sda1       194M  190M     0 100% /boot

192.168.149.130 | SUCCESS | rc=0 >>
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda3        30G   16G   12G  57% /
tmpfs           242M     0  242M   0% /dev/shm
/dev/sda1       194M   27M  158M  15% /boot

[root@localhost ansible]#
```

图 21-9 Ansible command df -h 命令执行结果

24.10 Ansible copy 模块实战

Ansible copy 模块主要用于文件或者目录拷贝，支持文件、目录、权限、用户组功能，

copy 模块使用详解：

src	Ansible 端源文件或者目录，空文件夹不拷贝；
content	用来替代 src，用于将指定文件的内容，拷贝到远程文件内；
dest	客户端目标目录或者文件，需要绝对路径；
backup	拷贝之前，先备份远程节点上的原始文件；
directory_mode	用于拷贝文件夹，新建的文件会被拷贝，而老旧的不会被拷贝；
follow	支持 link 文件拷贝；
force	覆盖远程主机不一致的内容；
group	设定远程主机文件夹的组名；
mode	指定远程主机文件及文件及的权限；
owner	设定远程主机文件夹的用户名。

Ansible copy 模块企业常用案例如下：

（1） Ansible copy 模块操作，src 表示源文件，dest 表示目标目录或者文件，owner

指定拥有者，执行结果如图 21-10 所示：

```
ansible -k all -m copy  -a  'src=/etc/passwd  dest=/tmp/  mode=755
owner=root'
```

图 21-10 Ansible copy 拷贝文件

（2） Ansible copy 模块操作，content 文件内容，dest 目标文件，owner 指定拥有者，

执行结果如图 21-11 所示：

```
ansible -k all -m copy  -a  'content="Hello World"  dest=/tmp/jfedu.txt
mode=755  owner=root'
```

图 21-11 Ansible copy 追加内容

（3） Ansible copy 模块操作，content 文件内容，dest 目标文件，owner 指定拥有者，

backup=yes 开启备份，执行结果如图 21-12 所示：

```
ansible -k all -m copy  -a  'content="Hello World"  dest=/tmp/jfedu.txt

backup=yes mode=755  owner=root'
```

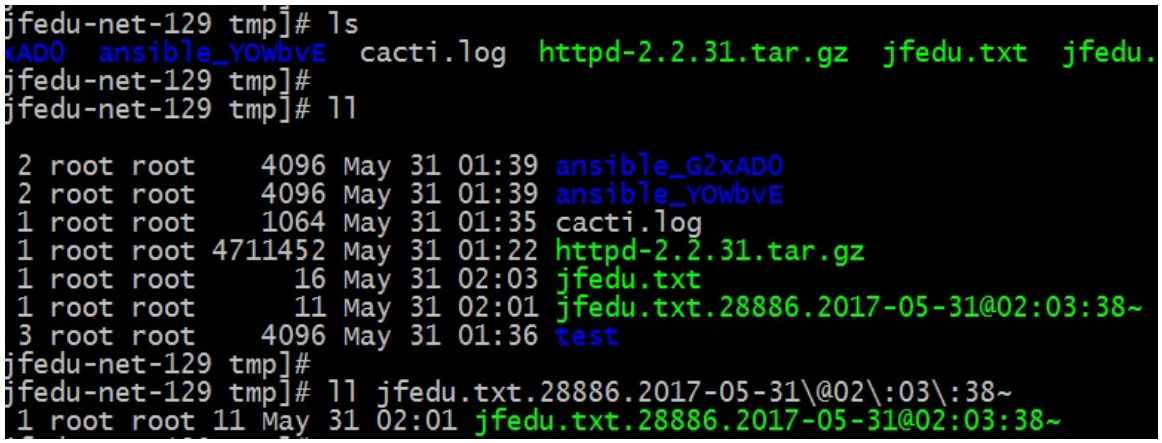


图 21-12 Ansible copy 客户端备份结果

24.11 Ansible yum 模块实战

Ansible yum 模块主要用于软件的安装、升级、卸载，支持红帽.rpm 软件的管理，YUM 模块使用详解：

conf_file	设定远程 yum 执行时所依赖的 yum 配置文件
disable_gpg_check	安装软件包之前是否坚持 gpg key；
name	需要安装的软件名称，支持软件组安装；
update_cache	安装软件前更新缓存；
enablerepo	指定 repo 源名称；
skip_broken	跳过异常软件节点；
state	软件包状态，包括：installed、present、latest、absent、removed。

Ansible yum 模块企业常用案例如下：

（1） Ansible yum 模块操作，name 表示需安装的软件名称，state 表示状态，常见 state= installed 表示安装软件，执行结果如图 21-13 所示：

```
ansible all -k -m yum -a "name=sysstat,screen state=installed"
```

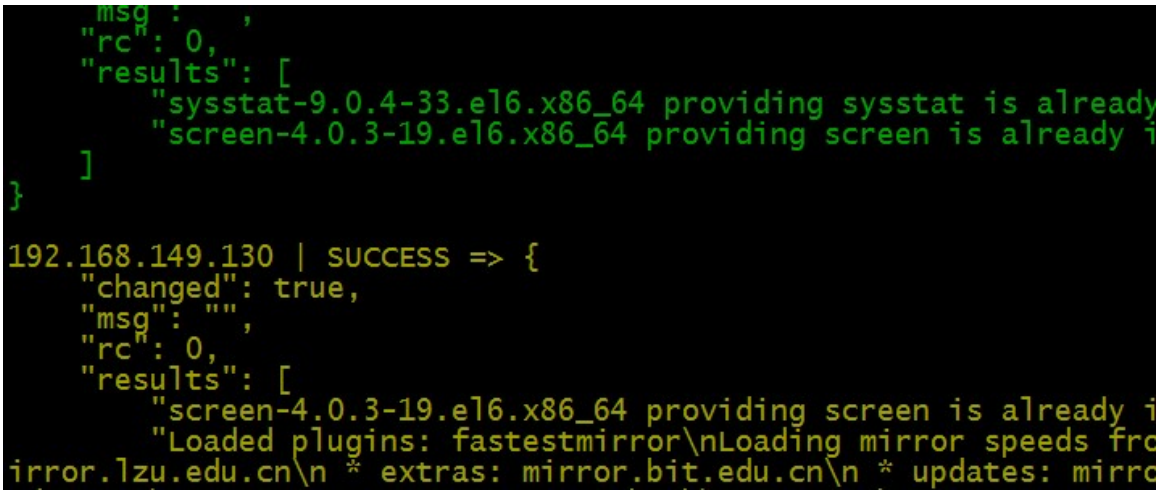


图 21-13 Ansible YUM 安装软件包

（2） Ansible yum 模块操作，name 表示需安装的软件名称，state 表示状态，常见 state= installed 表示安装软件，执行结果如图 21-14 所示：

```
ansible all -k -m yum -a "name=sysstat,screen state=absent"
```

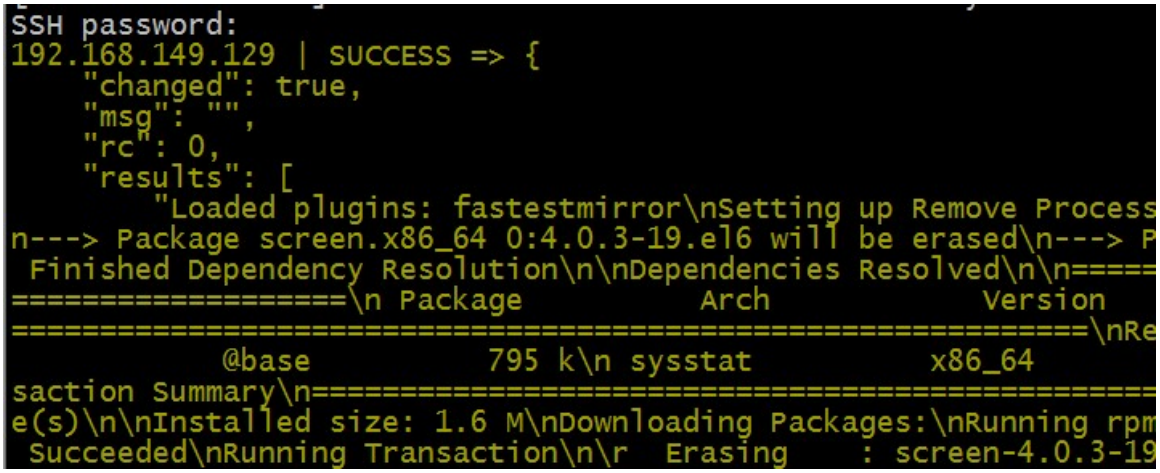


图 21-14 Ansible YUM 卸载软件包

（3） Ansible yum 模块操作，name 表示需安装的软件名称，state 表示状态，常见 state= installed，表示安装软件，disable_gpg_check=no 不检查 key，执行结果如图 21-15 所示：

```
ansible 192.168.149.129 -k -m yum -a "name=sysstat,screen
state=installed disable_gpg_check=no"
```

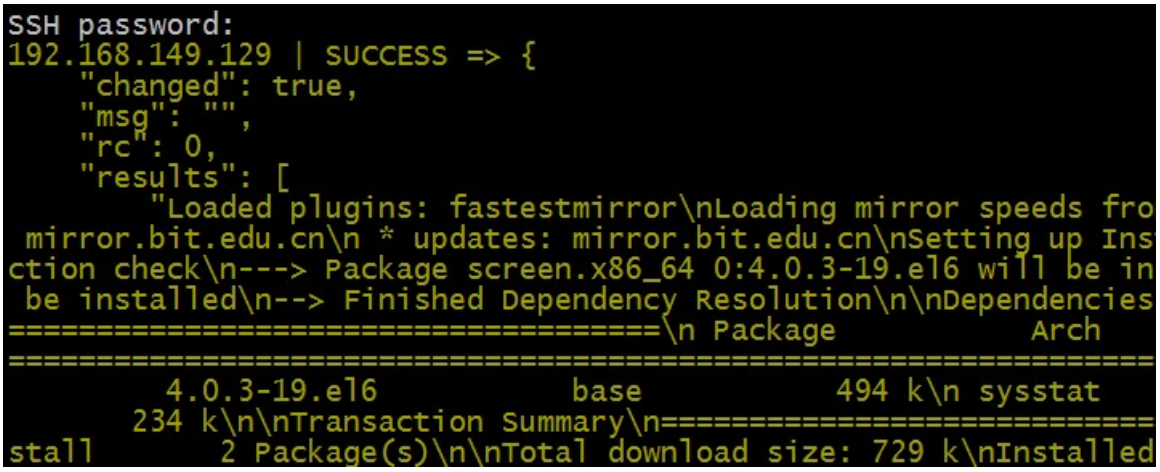


图 21-15 Ansible YUM 安装软件包，不检查 KEY

24.12 Ansible file 模块实战

Ansible file 模块主要用于对文件的创建、删除、修改、权限、属性的维护和管理，File 模块使用详解：

src	Ansible 端源文件或者目录；
follow	支持 link 文件拷贝；
force	覆盖远程主机不一致的内容；
group	设定远程主机文件夹的组名；
mode	指定远程主机文件及文件及的权限；
owner	设定远程主机文件夹的用户名；

path	目标路径，也可以用 dest,name 代替；
state	状态包括：file、link、directory、hard、touch、absent；
attributes	文件或者目录特殊属性。

Ansible file 模块企业常用案例如下：

（1） Ansible file 模块操作，path 表示目录的名称和路径， state=directory 表示创建目录，执行结果如图 21-16 所示：

```
ansible -k 192.168.* -m file -a "path=/tmp/`date +%F` state=directory mode=755"
```

```
[root@localhost ~]# ansible -k 192.168.* -m file -a "path=/tmp/test
SSH password:
192.168.149.129 | SUCCESS => {
    "changed": true,
    "dest": "/tmp/test.txt",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "size": 6,
    "state": "file",
    "uid": 0
}
192.168.149.130 | SUCCESS => {
    "changed": true,
```

图 21-16 Ansible file 创建目录

（2） Ansible file 模块操作，path 表示目录的名称和路径， state=touch 表示创建文件，执行结果如图 21-17 所示：

```
ansible -k 192.168.* -m file -a "path=/tmp/jfedu.txt state=touch mode=755"
```

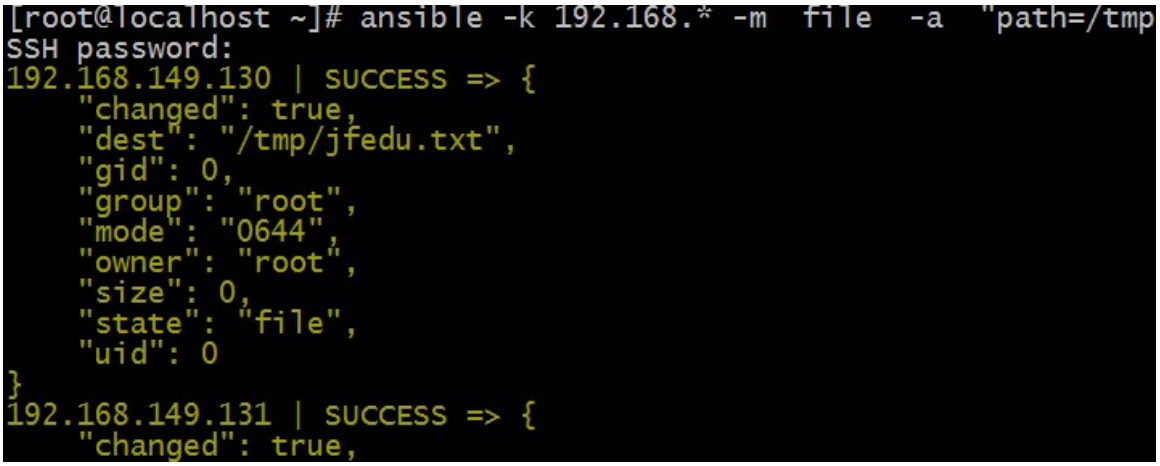


图 21-17 Ansible file 创建文件

24.13 Ansible user 模块实战

Ansible user 模块主要用于操作系统用户、组、权限、密码等操作，user 模块使用详解：

system	默认创建为普通用户，为 yes 则创建系统用户；
append	添加一个新的组；
comment	新增描述信息；
createhome	给用户创建家目录；
force	用于删除强制删除用户；
group	创建用户主组；
groups	将用户加入组或者附属组添加；
home	指定用户的家目录；
name	表示状态，是否 create、remove、modify；
password	指定用户的密码，此处为加密密码；
remove	删除用户；
shell	设置用户的 shell 登录环境；

uid	设置用户 id；
update_password	修改用户密码；
state	用户状态，默认为 present 表示新建用户。

Ansible user 模块企业常用案例如下：

（1） Ansible user 模块操作，name 表示用户名称，home 表示其家目录，执行结果如图 21-18 所示：

```
ansible -k 192.168.149.* -m user -a "name=jfedu home=/tmp/"
```

```
[root@localhost ~]# ansible -k 192.168.149.* -m user -a
SSH password:
192.168.149.129 | SUCCESS => {
    "changed": true,
    "comment": "",
    "createhome": true,
    "group": 504,
    "home": "/tmp/",
    "name": "jfedu",
    "shell": "/bin/bash",
    "state": "present",
    "stderr": "useradd: warning: the home directory already exist
,
    "stderr_lines": [
        "useradd: warning: the home directory already exists.",
```

图 21-18 Ansible user 创建新用户

（2） Ansible user 模块操作，name 表示用户名称，home 表示其家目录，执行结果如图 21-19 所示：

```
ansible -k 192.168.149.* -m user -a "name=jfedu home=/tmp/
shell=/sbin/nologin"
```

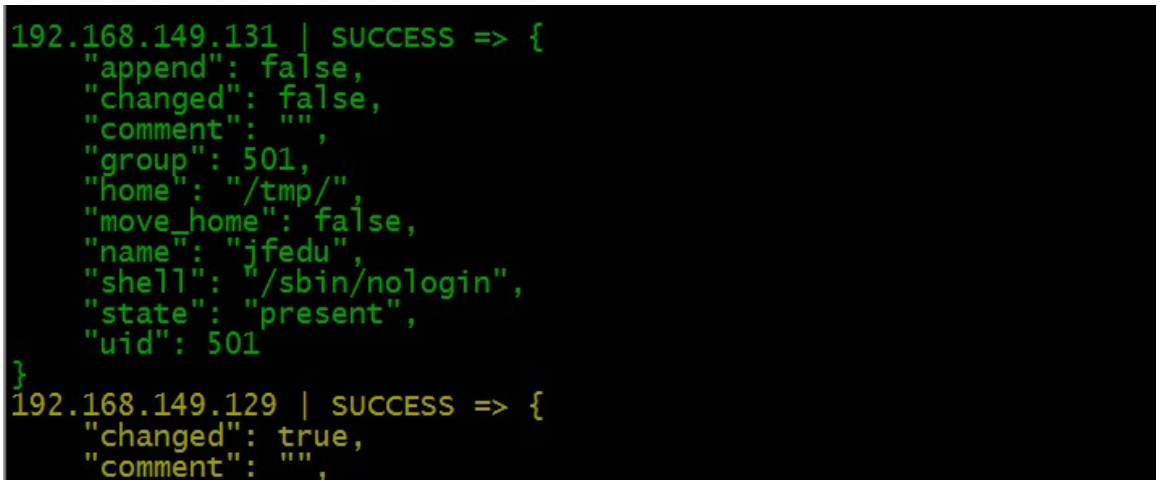


图 21-19 Ansible user 指定 Shell 环境

（3） Ansible user 模块操作，name 表示用户名称，state=absent 表示删除用户，执行结果如图 21-20 所示：

```
ansible -k 192.168.149.* -m user -a "name=jfedu state=absent force=yes"
```

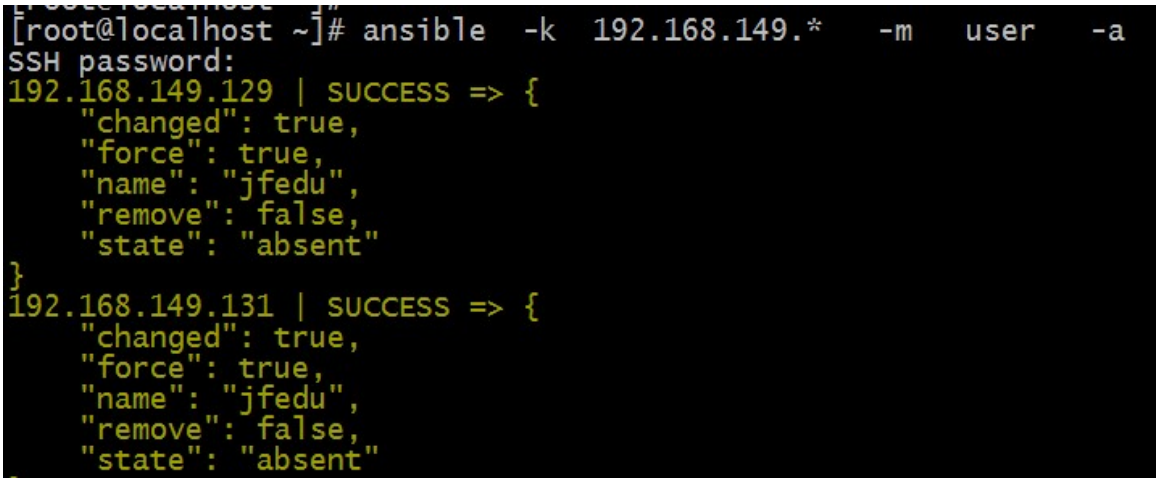


图 21-20 Ansible user 删除用户

24.14 Ansible cron 模块实战

Ansible cron 模块主要用于添加、删除、更新操作系统 Crontab 任务计划，Cron 模块使用详解：

name	任务计划名称；
cron_file	替换客户端该用户的任务计划的文件；
minute	分（ 0-59 ，* ，*/2 ）；
hour	时（ 0-23 ，* ，*/2 ）；
day	日（ 1-31 ，* ，*/2 ）；
month	月（ 1-12 ，* ，*/2 ）；
weekday	周（ 0-6 或 1-7 ，* ）；
job	任何计划执行的命令，state 要等于 present；
backup	是否备份之前的任务计划；
user	新建任务计划的用户；
state	指定任务计划 present、absent。

Ansible cron 模块企业常用案例如下：

（1） Ansible cron 模块操作，基于 cron 模块，创建 crontab 任务计划，执行结果如图 21-21 所示：

```
ansible -k all -m cron -a "minute=0 hour=0 day=* month=*
weekday=* name='Ntpdate server for sync time' job='/usr/sbin/ntpdate
139.224.227.121'"
```

```
[root@localhost ~]# ansible  -k  all  -m  cron  -a  "minute=0  ho
 sync time'   job='/usr/sbin/ntpdate 139.224.227.121'"
SSH password:
192.168.149.129 | SUCCESS => {
    "changed": true,
    "envs": [],
    "jobs": [
        "Ntpdate server for sync time"
    ]
}
192.168.149.130 | SUCCESS => {
    "changed": false,
    "envs": [],
    "jobs": [
        "Ntpdate server for sync time"
```

图 21-21 Ansible cron 添加任务计划

（2） Ansible cron 模块操作，基于 cron 模块，备份 crontab 任务计划，backup=yes 表示开启备份，备份文件存放于客户端/tmp/，执行结果如图 21-22 所示：

```
ansible -k all -m cron -a "minute=0 hour=0 day=* month=*
weekday=* name='Ntpdate server for sync time' backup=yes
job='/usr/sbin/ntpdate pool.ntp.org'"
```

```
[root@localhost ~]# ansible  -k  all  -m  cron  -a  "minute=0  ho
 sync time'  backup=yes  job='/usr/sbin/ntpdate pool.ntp.org'"
SSH password:
192.168.149.129 | SUCCESS => {
    "backup_file": "/tmp/crontab_Hp7sX",
    "changed": true,
    "envs": [],
    "jobs": [
        "Ntpdate server for sync time"
    ]
}
192.168.149.130 | SUCCESS => {
    "backup_file": "/tmp/crontabjzWpij",
    "changed": true,
    "envs": [],
```

图 21-22 Ansible cron 删除任务计划

（3） Ansible cron 模块操作，基于 cron 模块，删除 crontab 任务计划，执行结果如图 21-23 所示：

```
ansible -k all -m cron -a "name='Ntpdate server for sync time'
state=absent"
```

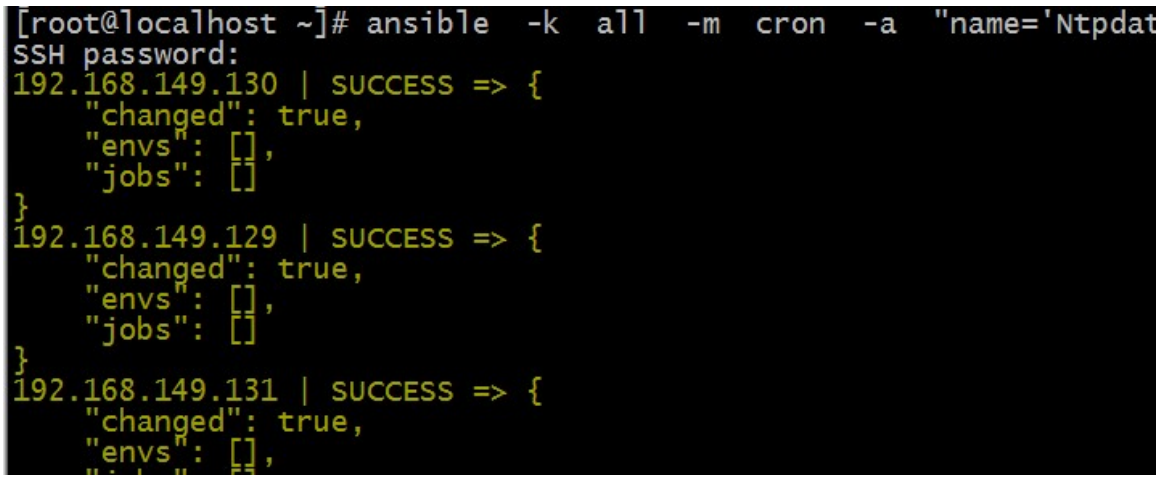


图 21-23 Ansible cron 删除任务计划

24.15 Ansible synchronize 模块实战

Ansible synchronize 模块主要用于目录、文件同步，基于 Rsync 命令同步目录，Synchronize 模块使用详解：

compress	开启压缩，默认为开启；
archive	是否采用归档模式同步，保证源和目标文件属性一致；
checksum	是否效验；
dirs	以非递归的方式传输目录；
links	同步链接文件；
recursive	是否递归 yes/no；
rsync_opts	使用 rsync 的参数；
copy_links	同步的时候是否复制连接；
delete	删除源中没有而目标存在的文件；

src	源目录及文件；
dest	目标目录及文件；
dest_port	目标接受的端口；
rsync_path	服务的路径，指定 rsync 命令来在远程服务器上运行；
rsync_timeout	指定 rsync 操作的 IP 超时时间；
set_remote_user	设置远程用户名；
--exclude=.log	忽略同步.log 结尾的文件；
mode	同步的模式，rsync 同步的方式 PUSH、PULL，默认都是推送 push。

Ansible synchronize 模块企业常用案例如下：

（1） Ansible synchronize 模块操作，src 源目录、dest 目标目录，执行结果如图 21-24 所示：

```
ansible   -k   all -m synchronize   -a 'src=/tmp/ dest=/tmp/'
```

```
192.168.149.129 | SUCCESS => {
    "changed": true,
    "cmd": "/usr/bin/rsync --delay-updates -F --compress --archiv
o StrictHostKeyChecking=no --out-format=<<CHANGED>>%i %n%L /tmp/
    "msg": ".d..t...... ./\n<f.st...... cacti.log\n<f+++++++++ cr
.txt\n<f..t...... test.txt\n<f+++++++++ tmp41pHom\n<f+++++++++ tm
\n<f+++++++++ tmpbKomV7\ncd+++++++++ ansible_RTiOm9/\n<f+++++++++
.zip\n<f+++++++++ ansible_RTiOm9/ansible_module_synchronize.py\n"
    "rc": 0,
    "stdout_lines": [
        ".d..t...... ./",
        "<f.st...... cacti.log",
        "<f+++++++++ crontabSqrvcV",
        "<f.st...... jfedu.txt",
        "<f..t...... test.txt"
```

图 21-24 Ansible 目录同步

（2） Ansible synchronize 模块操作，src 源目录、dest 目标目录、compress=yes 开

启压缩、delete=yes 数据一致、rsync_opts 同步参数、--exclude 排除文件，执行结果如图 21-25 所示：

```
ansible -k all -m synchronize -a 'src=/tmp/ dest=/tmp/ compress=yes
delete=yes rsync_opts=--no-motd,--exclude=.txt'
```

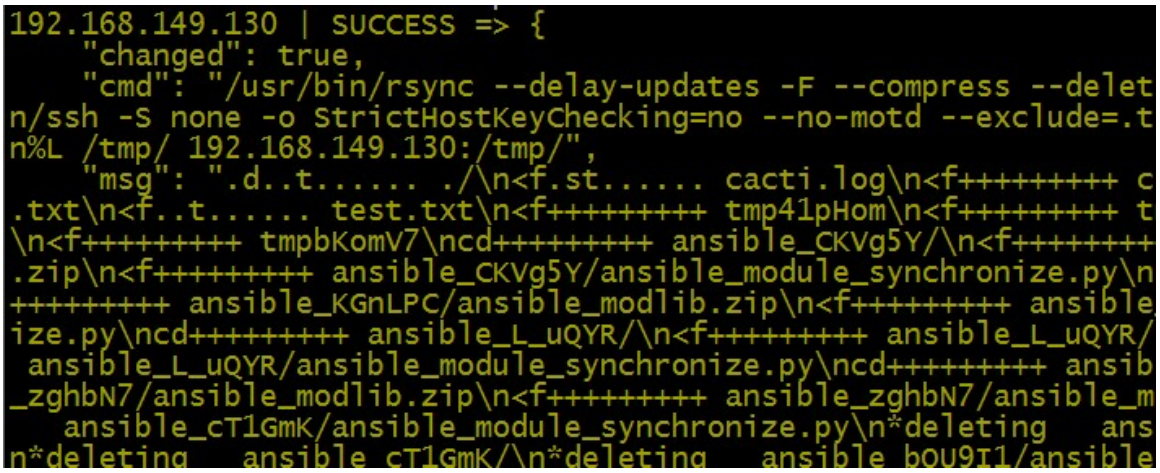


图 21-25 Ansible 目录同步排除.txt 文件

24.16 Ansible shell 模块实战

Ansible shell 模块主要用于远程客户端上执行各种 Shell 命令或者运行脚本，远程执行命令通过/bin/sh 环境来执行，支持比 command 更多的指令，Shell 模块使用详解：

Chdir	执行命令前，切换到目录；
Creates	当该文件存在时，则不执行该步骤；
Executable	换用 shell 环境执行命令；
Free_form	需要执行的脚本；
Removes	当该文件不存在时，则不执行该步骤；
Warn	如果在 ansible.cfg 中存在告警，如果设定了 False，不会警告此行。

Ansible shell 模块企业常用案例如下：

（1） Ansible shell 模块操作，-m shell 指定模块为 shell，远程执行 Shell 脚本，远程执行脚本也可采用 script 模块。并把执行结果追加至客户端服务器/tmp/var.log 文件，执行结果如图 21-26 所示：

```
ansible -k all -m shell -a "/bin/sh /tmp/variables.sh >>/tmp/var.log"
```

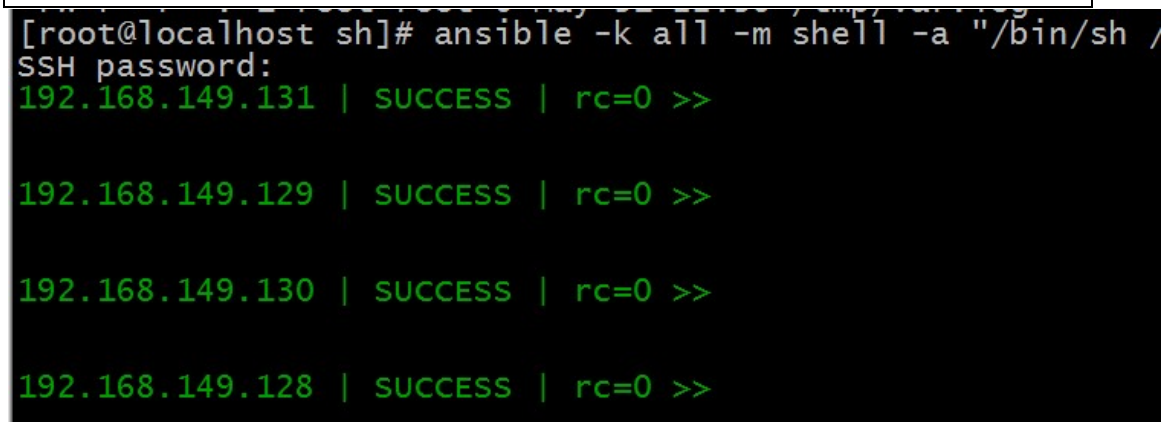


图 21-26 Ansible shell 远程执行脚本

（2） Ansible shell 模块操作，远程执行创建目录命令，执行之前切换在/tmp 目录，屏蔽警告信息，执行结果如图 21-27 所示：

```
ansible -k all -m shell -a "mkdir -p `date +%F` chdir=/tmp/ state=directory
warn=no"
```

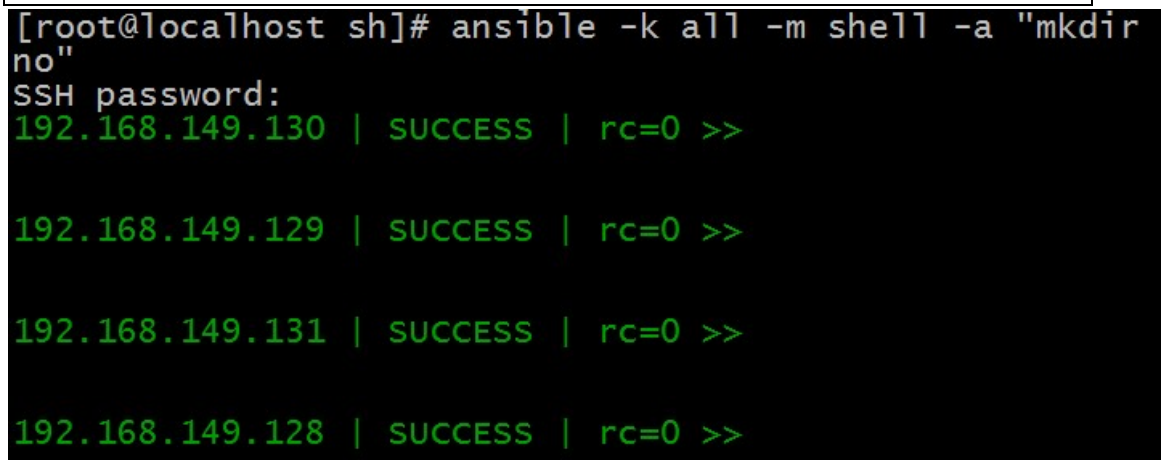


图 21-27 Ansible shell 远程执行脚本

（3） Ansible shell 模块操作，-m shell 指定模块为 shell，远程客户端查看 http 进程是否启动，执行结果如图 21-28 所示：

```
ansible -k all -m shell -a "ps -ef |grep http"
```

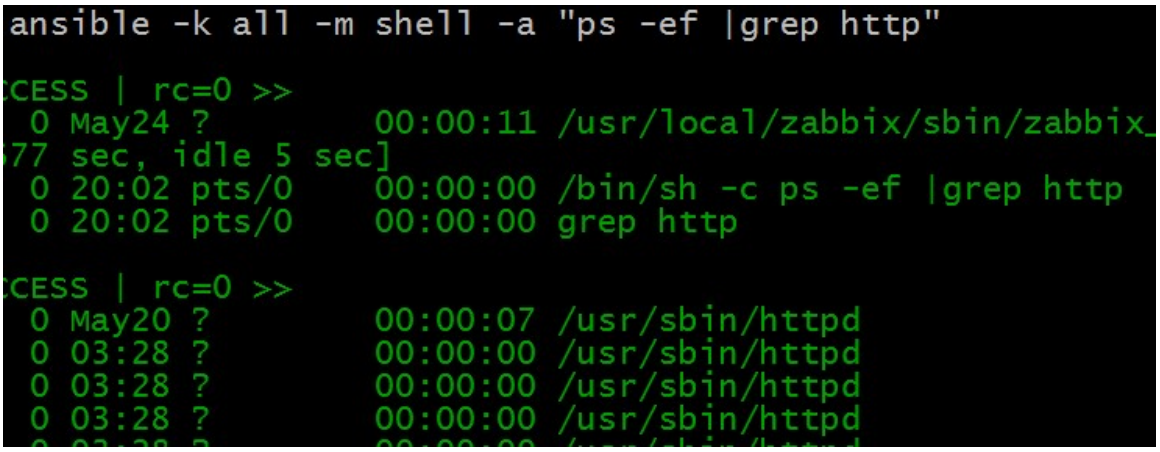


图 21-28 Ansible shell 远程查看进程

（4） Ansible shell 模块操作，-m shell 指定模块为 shell，远程客户端查看 crontab 任务计划，执行结果如图 21-29 所示：

```
ansible -k all -m shell -a "crontab -l"
```

图 21-29 Ansible shell 远程查看任务计划

24.17 Ansible service 模块实战

Ansible service 模块主要用于远程客户端各种服务管理，包括启动、停止、重启、重

新加载等，service 模块使用详解：

enabled	是否开启启动服务；
name	服务名称；
runlevel	服务启动级别；
arguments	服务命令行参数传递；
state	服务操作状态，状态包括 started，stopped，restarted，reloaded。

Ansible service 模块企业常用案例如下：

（1） Ansible service 模块操作，远程重启 httpd 服务，执行结果如图 21-30 所示：

```
ansible -k all -m service -a "name=httpd state=restarted"
```

```
[root@localhost sh]# ansible -k all -m service -a "name=
SSH password:
192.168.149.130 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
192.168.149.128 | SUCCESS => {
    "changed": true,
    "name": "httpd",
    "state": "started"
}
192.168.149.131 | SUCCESS => {
    "changed": true
```

图 21-30 Ansible service 重启 httpd 服务

（2） Ansible service 模块操作，远程重启网卡服务，指定参数 eth0，执行结果如图 21-31 所示：

```
ansible -k all -m service -a "name=network args=eth0 state=restarted"
```

```
[root@localhost sh]# ansible -k all  -m  service  -a  "na
SSH password:
192.168.149.129 | SUCCESS => {
    "changed": true,
    "name": "network",
    "state": "started"
}
192.168.149.128 | SUCCESS => {
    "changed": true,
    "name": "network",
    "state": "started"
}
192.168.149.131 | SUCCESS => {
    "changed": true,
```

图 21-31 Ansible service 重启 network 服务

（3） Ansible service 模块操作，远程开机启动 nfs 服务，设置 3,5 级别自动启动，执行结果如图 21-32 所示：

```
ansible -k all  -m  service  -a  "name=nfs  enabled=yes  runlevel=3,5"
```

```
[root@localhost sh]# ansible -k all  -m  service  -a  "
SSH password:
192.168.149.130 | SUCCESS => {
    "changed": false,
    "enabled": true,
    "name": "nfs"
}
192.168.149.131 | SUCCESS => {
    "changed": false,
    "enabled": true,
    "name": "nfs"
}
192.168.149.129 | SUCCESS => {
    "changed": true,
```

图 21-32 Ansible service 开机启动 nfs 服务

24.18 Ansible Playbook 应用

如上使用 Ad-hoc 方式点对点命令执行，可以管理远程主机，如果服务器数量很多，配置信息比较多，还可以利用 Ansible Playbook 编写剧本、从而以非常简便的方式实现任务处理的自动化与流程化。

Playbook 由一个或多个"play"组成的列表，play 的主要功能 Ansible 中的 Task 定义

好的角色，指定剧本对应的服务器组。

从根本上说，Task 是一个任务，Task 调用 Ansible 各种模块 module，将多个 paly 组织在一个 playbook 剧本中，然后组成一个非常完整的流程控制集合。

基于 Ansible Playbook 还可以收集命令、可以创建任务集，这样能够大大降低管理工作的复杂程度，Playbook 采用 YAML 语法结构，易于阅读、方便配置。

YAML（Yet Another Markup Language），是一种直观的能够被电脑识别的数据序列化格式，是一个可读性高并且容易被人类阅读，容易和脚本语言交互，用来表达资料序列的编程语言。它参考了其它多种语言，包括：XML、C 语言、Python、Perl 以及电子邮件格式 RFC2822，是类似于标准通用标记语言的子集 XML 的数据描述语言，语法比 XML 简单很多。

YAML 使用空白字符和分行来分隔资料，适合用 grep、Python、Perl、Ruby 操作。

（1） YAML 语言特性如下：

- 可读性强；
- 和脚本语言的交互性好；
- 使用实现语言的数据类型；
- 一致的信息模型；
- 易于实现；
- 可以基于流来处理；
- 可扩展性强。

（2） Playbooks 组件包括如下：

Target	定义 playbook 的远程主机组；
Variable	定义 playbook 使用的变量；

Task	定义远程主机上执行的任务列表；
Handler	定义 task 执行完成以后需要调用的任务，例如配置文件被改动，则启动 handler 任务重启相关联的服务。

（3） Target 常用参数如下：

hosts	定义远程主机组；
user	执行该任务的用户；
sudo	设置为 yes 的时候，执行任务的时候使用 root 权限；
sudo_user	指定 sudo 普通用户；
connection	默认基于 SSH 连接客户端；
gather_facks	获取远程主机 facts 基础信息。

（4） Variable 常用参数如下：

vars	定义格式，变量名:变量值；
vars_files	指定变量文件；
vars_prompt	用户交互模式自定义变量；
setup	模块去远程主机的值；

（5） Task 常用参数如下：

name	任务显示名称也即屏幕显示信息；
action	定义执行的动作；
copy	复制本地文件到远程主机；
template	复制本地文件到远程主机，可以引用本地变量；
service	定义服务的状态。

Ansible playbook 案例演示如下：

（1） 远程主机安装 Nginx WEB 服务，playbook 代码如下，执行结果如图 21-33 所示：

```
- hosts: all
  remote_user: root
  tasks:
  - name: Jfedu Pcre-devel and Zlib LIB Install.
    yum:  name=pcre-devel,pcre,zlib-devel state=installed
  - name: Jfedu  Nginx WEB  Server Install Process.
    shell:  cd  /tmp  ;  rm  -rf  nginx-1.12.0.tar.gz  ;  wget
http://nginx.org/download/nginx-1.12.0.tar.gz；tar xzf nginx-1.12.0.tar.gz；cd
nginx-1.12.0；./configure --prefix=/usr/local/nginx；make；make install
```

```
[root@localhost ~]# ansible-playbook nginx_install.yaml

PLAY [all] *********************************************

TASK [Nginx WEB Server Rewrite Install] ****************
ok: [192.168.149.129]
ok: [192.168.149.128]

TASK [Jfedu install Nginx WEB Server Process] **********
changed: [192.168.149.128]
changed: [192.168.149.129]

PLAY RECAP *********************************************
192.168.149.128            : ok=2    changed=1    unrea
192.168.149.129            : ok=2    changed=1    unrea
```

图 21-33 Ansible Playbook 远程 Nginx 安装

（2） 检测远程主机 Nginx 目录是否存在，不存在则安装 Nginx WEB 服务，安装完并启动 Nginx，playbook 代码如下，执行结果如图 21-34 所示：

```
- hosts: all
  remote_user: root
```

```
tasks:
    - name: Nginx server Install 2017
      file: path=/usr/local/nginx/ state=directory
      notify:
        - nginx install
        - nginx start
handlers:
    - name: nginx install
      shell: cd /tmp ; rm -rf nginx-1.12.0.tar.gz ; wget http://nginx.org/download/nginx-1.12.0.tar.gz; tar xzf nginx-1.12.0.tar.gz;cd nginx-1.12.0;./configure --prefix=/usr/local/nginx;make;make install
    - name: nginx start
      shell: /usr/local/nginx/sbin/nginx
```

```
[root@localhost ~]# ansible-playbook nginx.yaml

PLAY [all] *********************************************************

TASK [Nginx server Install 2017] ***********************************
changed: [192.168.149.129]

RUNNING HANDLER [nginx install] ************************************
changed: [192.168.149.129]

RUNNING HANDLER [nginx start] **************************************
changed: [192.168.149.129]

PLAY RECAP *********************************************************
192.168.149.129            : ok=3    changed=3    unreachable=0
```

图 21-34 Ansible Playbook Nginx 触发安装

（3） 检测远程主机内核参数配置文件是否更新，如果更新则执行命令 sysctl –p 使内核参数生效，playbook 代码如下，执行结果如图 21-35 所示：

```
- hosts: all
  remote_user: root
  tasks:
      - name: Linux kernel config 2017
        copy: src=/data/sh/sysctl.conf dest=/etc/
        notify:
            - source sysctl
  handlers:
      - name: source sysctl
        shell: sysctl -p
```

```
[root@localhost ~]# ansible-playbook sysctl.yaml

PLAY [all] *********************************************************

TASK [Linux kernel config 2017] ************************************
changed: [192.168.149.129]
changed: [192.168.149.128]

RUNNING HANDLER [source sysctl] ************************************
changed: [192.168.149.129]
changed: [192.168.149.128]

PLAY RECAP *********************************************************
192.168.149.128            : ok=2    changed=2    unreachable=0
192.168.149.129            : ok=2    changed=2    unreachable=0
```

图 21-35 Ansible Playbook 内核参数优化

（4） 基于列表 items 多个值创建用户，通过{{}}定义列表变量，with_items 选项传入变量的值，执行结果如图 21-36（a）、21-36（b）所示：

```
- hosts: all
  remote_user: root
  tasks:
```

```
- name: Linux system Add User list.
  user: name={{ item }} state=present
  with_items:
    - jfedu1
    - jfedu2
    - jfedu3
    - jfedu4
```

```
[root@localhost ~]# ansible-playbook user.yaml

PLAY [all] *********************************************************

TASK [Linux system Add User list.] *********************************
changed: [192.168.149.129] => (item=jfedu1)
changed: [192.168.149.129] => (item=jfedu2)
changed: [192.168.149.129] => (item=jfedu3)
changed: [192.168.149.129] => (item=jfedu4)

PLAY RECAP *********************************************************
192.168.149.129            : ok=1    changed=1    unreachable=0

[root@localhost ~]#
```

图 21-36（a） Ansible Playbook item 变量创建用户

```
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/no
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
apache:x:48:48:Apache:/var/www:/sbin/nologin
exim:x:93:93::/var/spool/exim:/sbin/nologin
sdfjsdklfskl
jfedu001:x:501:501::/home/jfedu001:/bin/bash
redis:x:496:496:Redis Server:/var/lib/redis:/sbin/nologin
a:x:502:502::/home/a:/bin/bash
zabbix:x:503:503::/home/zabbix:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
jfedu1:x:504:504::/home/jfedu1:/bin/bash
jfedu2:x:505:505::/home/jfedu2:/bin/bash
jfedu3:x:506:506::/home/jfedu3:/bin/bash
jfedu4:x:507:507::/home/jfedu4:/bin/bash
```

图 21-36（b） Ansible Playbook item 变量创建用户

（5） Ansible Playbook 可以自定义 template 模板文件，模板文件主要用于服务器需求不一致的情况，需要独立定义的，例如两台服务器安装了 Nginx，安装完毕之后将服务器 A 的 HTTP 端口改成 80，服务器 B 的 HTTP 端口改成 81，基于 tempalte 模块

轻松实现，方法步骤如下：

1. Ansible hosts 文件指定不同服务器不同 httpd_port 端口，代码如下：

```
[web]

192.168.149.128 httpd_port=80

192.168.149.129 httpd_port=81
```

2. Ansible 创建 nginx.conf jinja2 模板文件，cp nginx.conf nginx.conf.j2，并修改 listen 80 为 listen {{httpd_port}}，Nginx 其他配置项不变，代码如下：

```
cp nginx.conf nginx.conf.j2

listen   {{httpd_port}};
```

3. Ansible playbook 剧本 yaml 文件创建，代码如下：

```
- hosts: all

  remote_user: root

  tasks:

       - name: Nginx server Install 2017

         file: path=/usr/local/nginx/ state=directory

         notify:

              - nginx install

              - nginx config

  handlers:

       - name: nginx install

         shell: cd /tmp ; rm -rf nginx-1.12.0.tar.gz ; wget
http://nginx.org/download/nginx-1.12.0.tar.gz; tar xzf nginx-1.12.0
```

```
.tar.gz;cd nginx-1.12.0;./configure --prefix=/usr/local/nginx;make;make install

        - name: nginx config

          template:                                    src=/data/sh/nginx.conf.j2

dest=/usr/local/nginx/conf/nginx.conf
```

4. Ansible playbook 执行剧本文件，如图 21-37（a）、21-37（b）、21-37（c）所示：

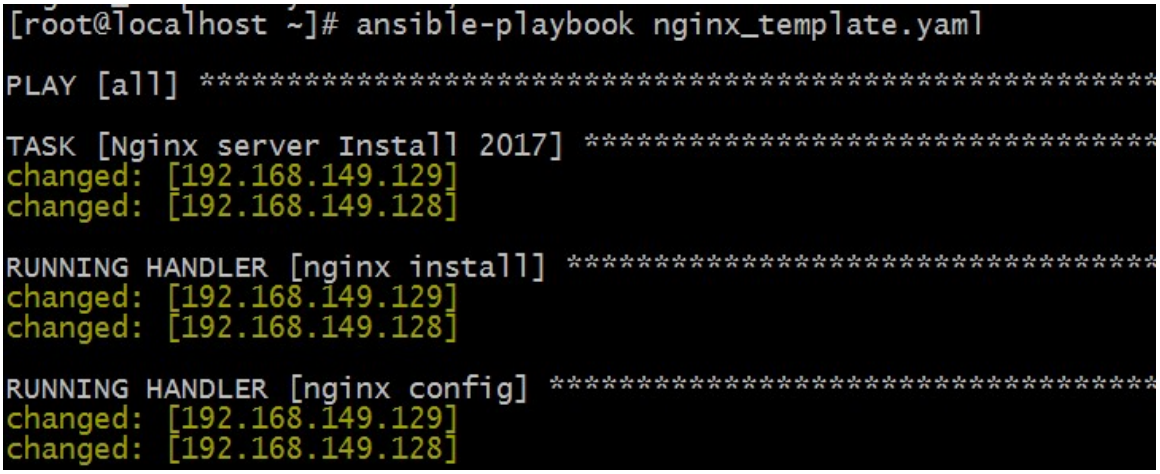


图 21-37（a） Ansible Playbook 执行模板 yaml

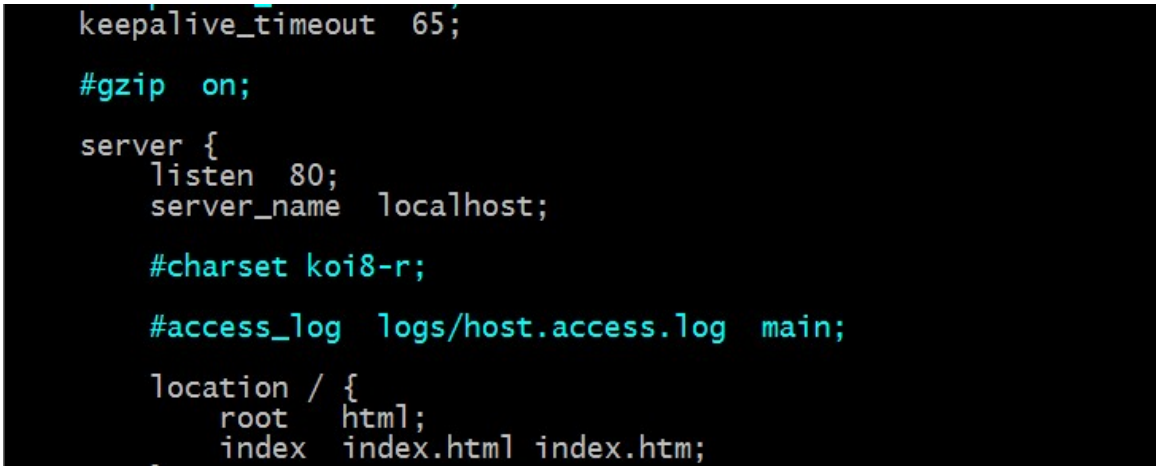


图 21-37（b） 149.128 服务器 Nginx HTTP Port 80

```
    keepalive_timeout  65;

    #gzip  on;

    server {
        listen  81;
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

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

图 21-37（c） 149.129 服务器 Nginx HTTP Port 81

24.19 Ansible 配置文件详解

Ansible 默认配置文件为/etc/ansible/ansible.cfg，配置文件中可以对 ansible 进行各项参数的调整，包括并发线程、用户、模块路径、配置优化等，如下为 Ansible.cfg 常用参数详解：

[defaults]	通用默认配置段；
inventory = /etc/ansible/hosts	被控端 IP 或者 DNS 列表；
library = /usr/share/my_modules/	Ansible 默认搜寻模块的位置；
remote_tmp = $HOME/.ansible/tmp	Ansible 远程执行临时文件；
pattern = *	对所有主机通信；
forks = 5	并行进程数；
poll_interval = 15	回频率或轮训间隔时间；
sudo_user = root	sudo 远程执行用户名；
ask_sudo_pass = True	使用 sudo，是否需要输入密码；
ask_pass = True	是否需要输入密码；
transport = smart	通信机制；

remote_port = 22	远程 SSH 端口；
module_lang = C	模块和系统之间通信的语言；
gathering = implicit	控制默认 facts 收集（远程系统变量）；
roles_path= /etc/ansible/roles	用于 playbook 搜索 Ansible roles；
host_key_checking = False	检查远程主机密钥；
#sudo_exe = sudo	sudo 远程执行命令；
#sudo_flags = -H	传递 sudo 之外的参数；
timeout = 10	SSH 超时时间；
remote_user = root	远程登陆用户名；
log_path = /var/log/ansible.log	日志文件存放路径；
module_name = command	Ansible 命令执行默认的模块；
#executable = /bin/sh	执行的 Shell 环境，用户 Shell 模块；
#hash_behaviour = replace	特定的优先级覆盖变量；
#jinja2_extensions	允许开启 Jinja2 拓展模块；
#private_key_file = /path/to/file	私钥文件存储位置；
#display_skipped_hosts = True	显示任何跳过任务的状态；
#system_warnings = True	禁用系统运行 ansible 潜在问题警告；
#deprecation_warnings = True	Playbook 输出禁用“不建议使用”警告；
#command_warnings = False	command 模块 Ansible 默认发出警

告；	
#nocolor = 1	输出带上颜色区别，开启/关闭：0/1；
pipelining = False	开启 pipe SSH 通道优化；
[accelerate]	accelerate 缓存加速。
accelerate_port = 5099	
accelerate_timeout = 30	
accelerate_connect_timeout = 5.0	
accelerate_daemon_timeout = 30	
accelerate_multi_key = yes	

24.20 Ansible 性能调优

Ansible 企业实战环境中，如果管理的服务器越来越多，Ansibe 执行效率会变得比较慢，可以通过优化 Ansible 提供工作效率，由于 Ansible 基于 SSH 协议通信，SSH 连接慢会导致整个基于 Ansible 执行变得缓慢，也需要对 Openssh 进行优化，具体优化的方法如下：

（1） Ansible SSH 关闭秘钥检测

默认以 SSH 登录远程客户端服务器，会检查远程主机的公钥(public key)，并将该主机的公钥记录在~/.ssh/known_hosts 文件中。下次访问相同主机时，OpenSSH 会核对公钥，如果公钥不同，OpenSSH 会发出警告，如果公钥相同，则提示输入密码。

SSH 对主机的 public_key 的检查等级是根据 StrictHostKeyChecking 变量来设定的，StrictHostKeyChecking 检查级别包括：no（不检查）、ask（询问）、yes（每次都检查）、False（关闭检查）。

Ansible 配置文件中加入如下代码，即可关闭 StrictHostKeyChecking 检查：

```
host_key_checking = False
```

（2） OpenSSH 连接优化

使用 OpenSSH 服务时，默认服务器端配置文件 UseDNS=YES 状态，该选项会导致服务器根据客户端的 IP 地址进行 DNS PTR 反向解析，得到客户端的主机名，然后根据获取到的主机名进行 DNS 正向 A 记录查询，并验证该 IP 是否与原始 IP 一致。关闭 DNS 解析代码如下：

```
sed -i '/^GSSAPI/s/yes/no/g ; /UseDNS/d ; /Protocol/aUseDNS no' /etc/ssh/sshd_config
/etc/init.d/sshd restart
```

（3） SSH pipelining 加速 Ansible

SSH pipelining 是一个加速 Ansible 执行速度的简单方法，SSH pipelining 默认是关闭的，关闭是为了兼容不同的 sudo 配置，主要是 requiretty 选项。

如果不使用 Sudo 建议开启该选项，打开此选项可以减少 Ansible 执行没有文件传输时，SSH 在被控机器上执行任务的连接数。使用 Sudo 操作的时候，必须在所有被管理的主机上将配置文件/etc/sudoers 中 requiretty 选项禁用。

```
sed -i '/^pipelining/s/False/True/g' /etc/ansible/ansible.cfg
```

（4） Ansible Facts 缓存优化

Ansible-playbook 在执行过程中，默认会执行 Gather facts，如果不需要获取客户端的 fact 数据的话，可以关闭获取 fact 数据功能，关闭之后可以加快 ansible-playbook 的执行效率。如需关闭 fact 功能，在 playbook yaml 文件中加入如下代码即可：

```
gather_facts: nogather_facts: no
```

Ansible facts 组件主要用于收集客户端设备的基础静态信息，这些信息可以在做配置

管理的时候方便引用。Facts 信息直接当做 Ansible Playbook 变量信息进行引用，通过定制 facts 以便收集我们想要的信息，同时可以通过 Facter 和 Ohai 来拓展 facts 信息，也可以将 facts 信息存入 Redis 缓存中，如下为 Facts 使用 Redis 缓存的步骤。

1. 部署 Redis 服务

```
wget    http://download.redis.io/releases/redis-2.8.13.tar.gz
tar     zxf             redis-2.8.13.tar.gz
cd      redis-2.8.13
make    PREFIX=/usr/local/redis  install
cp      redis.conf      /usr/local/redis/
```

将/usr/local/redis/bin/目录加入至环境变量配置文件/etc/profile 末尾，然后 Shell 终端执行 source /etc/profile 让环境变量生效。

```
export PATH=/usr/local/redis/bin:$PATH
```

启动及停止 Redis 服务命令：

```
nohup /usr/local/redis/bin/redis-server  /usr/local/redis/redis.conf  &
```

2. 安装 Python Redis 模块

```
easy_install pip
pip install redis
```

3. Ansible 整合 Redis 配置

在配置文件/etc/ansible/ansible.cfg 中 defaluts 段中加入代码，如果 redis 密码为 admin，则开启 admin 密码行：

```
gathering = smart
```

```
fact_caching = redis

fact_caching_timeout = 86400

fact_caching_connection = localhost:6379

#fact_caching_connection = localhost:6379:0:admin
```

4. 测试 Redis 缓存

Ansible-playbook 执行 nginx_wget.yaml 剧本文件，如图 21-38 所示：

```
ansible-playbook      nginx_wget.yaml
```

```
[root@localhost ~]# ansible-playbook nginx_wget.yaml

PLAY [192.168.*] ***************************************

TASK [Gathering Facts] *********************************
ok: [192.168.149.129]
ok: [192.168.149.130]
ok: [192.168.149.131]
ok: [192.168.149.128]

TASK [www.jfedu.net Centos Manager] ********************
```

图 21-38 ansible playbook 执行 yaml

检查 Redis 服务器，facts key 已存入 Redis 中，如图 21-39 所示：

```
[root@localhost ~]# redis-cli
127.0.0.1:6379>
127.0.0.1:6379> KEYS *
(empty list or set)
127.0.0.1:6379> KEYS *
1) "ansible_facts192.168.149.131"
2) "ansible_facts192.168.149.130"
3) "ansible_facts192.168.149.128"
4) "ansible_facts192.168.149.129"
5) "ansible_cache_keys"
127.0.0.1:6379>
```

图 21-39 Redis 缓存服务器缓存 facts 主机信息

（5） ControlPersist SSH 优化

ControlPersist 特性需要高版本的 SSH 支持，CentOS6 默认是不支持的，如果需要使

用，需要自行升级 Openssh。

ControlPersist 即持久化的 Socket，一次验证多次通信。并且只需要修改 SSH 客户端配置，也即 Ansible 被管理主机。

可使用 YUM 或者源码编译升级 OpenSSH 服务，升级完毕 ControlPersist 的设置办法如下，在其用户的家目录创建 config 文件，如果 ansible 以 root 用户登录客户端，至需要在客户端的/root/.ssh/config 目录中添加如下代码即可：

```
Host *
   Compression yes
   ServerAliveInterval 60
   ServerAliveCountMax 5
   ControlMaster auto
   ControlPath ~/.ssh/sockets/%r@%h-%p
   ControlPersist 4h
```

开启 ControlPersist 特性后，SSH 在建立 sockets 后，节省了每次验证和创建的时间，对 Ansible 执行速度提升是非常明显的。

第25章 MySQL+DRBD+Keepalived

25.1 DRBD 工作原理及入门

DRBD(Distributed Replicated Block Device)是一个基于块设备级别在远程服务器直接同步和镜像数据的开源软件，类似于 RAID1 数据镜像，通常配合 keepalived、heartbeat 等 HA 软件来实现高可用性。

DRBD 是一种块设备,可以被用于高可用(HA)之中.它类似于一个网络 RAID-1 功能，当你将数据写入本地文件系统时,数据还将会被发送到网络中另一台主机上.以相同的形式记录在一个文件系统中。

本地(master)与远程主机(backup)的保证实时同步,如果本地系统出现故障时,远程主机上还会保留有一份相同的数据,可以继续使用.在高可用(HA)中使用 DRBD 功能,可以代替使用一个共享盘阵.因为数据同时存在于本地主机和远程主机上,切换时,远程主机只要使用它上面的那份备份数据。

通过本次课程的学习，大家可以熟练构建企业级 MySQL+DRBD+Keepalived 高性能高可用架构，满足企业网站高效的访问，有突发故障及时切换。

25.2 内核优化及 DRBD 部署安装

一、实施环境

系统版本：CentOS 5.8

DRBD 版本: drbd-8.3.15

Keepalived：keepalived-1.1.15

Master：192.168.149.128

Backup：192.168.149.129

二、初始化配置

1) 在 128、129 两台服务器/etc/hosts 里面都添加如下配置：

192.168.149.128　node1

192.168.149.129　node2

2) 优化系统 kernel 参数，直接上 sysctl.conf 配置如下:

net.ipv4.ip_forward = 0

net.ipv4.conf.default.rp_filter = 1

net.ipv4.conf.default.accept_source_route = 0

kernel.sysrq = 0

kernel.core_uses_pid = 1

net.ipv4.tcp_syncookies = 1

kernel.msgmnb = 65536

kernel.msgmax = 65536

kernel.shmmax = 68719476736

kernel.shmall = 4294967296

net.ipv4.tcp_max_tw_buckets = 10000

net.ipv4.tcp_sack = 1

net.ipv4.tcp_window_scaling = 1

net.ipv4.tcp_rmem = 4096 87380 4194304

net.ipv4.tcp_wmem = 4096 16384 4194304

net.core.wmem_default = 8388608

net.core.rmem_default = 8388608

net.core.rmem_max = 16777216

net.core.wmem_max = 16777216

net.core.netdev_max_backlog = 262144

net.core.somaxconn = 262144

net.ipv4.tcp_max_orphans = 3276800

net.ipv4.tcp_max_syn_backlog = 262144

net.ipv4.tcp_timestamps = 0

net.ipv4.tcp_synack_retries = 1

net.ipv4.tcp_syn_retries = 1

net.ipv4.tcp_tw_recycle = 1

net.ipv4.tcp_tw_reuse = 1

net.ipv4.tcp_mem = 94500000 915000000 927000000

net.ipv4.tcp_fin_timeout = 1

net.ipv4.tcp_keepalive_time = 30

net.ipv4.ip_local_port_range = 1024 65530

net.ipv4.icmp_echo_ignore_all = 1

3)两台服务器分别添加一块设备，用于 DRBD 主设备存储，我这里为/dev/sdb 20G 硬盘；

执行如下命令：

mkfs.ext3 /dev/sdb ;dd if=/dev/zero of=/dev/sdb bs=1M count=1;sync

三、DRBD 安装配置

Yum 方式安装：

rpm -Uvh http://www.elrepo.org/elrepo-release-6-6.el6.elrepo.noarch.rpm

yum -y install drbd83* kmod-drbd83 ; modprobe drbd

源码安装方式：

http://oss.linbit.com/drbd/8.4/drbd-8.4.4.tar.gz

./configure --prefix=/usr/local/drbd --with-km

make KDIR=/usr/src/kernels/2.6.32-504.el6.x86_64/

```
make install

cp drbd/drbd.ko /lib/modules/`uname -r`/kernel/lib/
```

Yum 方式和源码方式都需要执行：modprobe drbd 加载 DRBD 模块。

安装完成并加载 drbd 模块后，vi 修改/etc/drbd.conf 配置文件，本文内容如下：

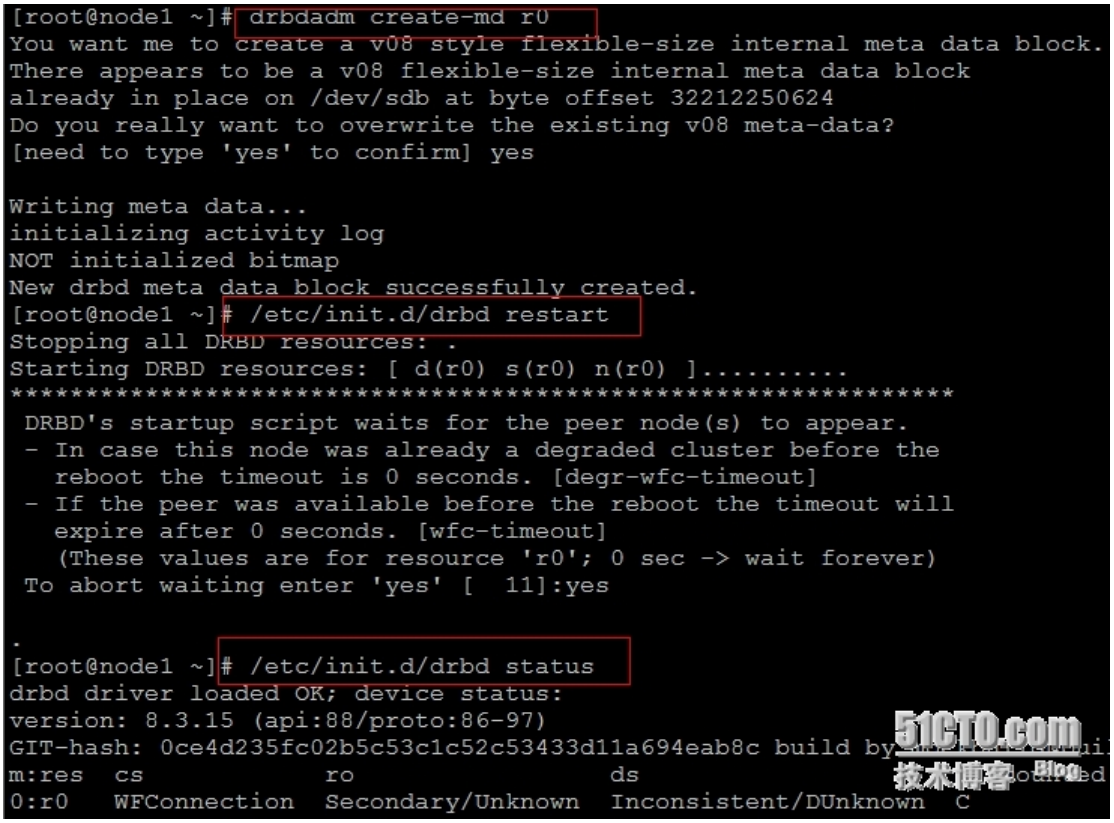
```
global {
      usage-count yes;
}
common {
syncer { rate 100M; }
}
resource r0 {
 protocol C;
 startup {


 }
 disk {
 on-io-error detach;
 #size 1G;
 }
 net {
 }
 on node1 {
```

```
 device /dev/drbd0;
 disk /dev/sdb;
 address 192.168.1.12:7898;
 meta-disk internal;
 }
 on node2 {
 device /dev/drbd0;
 disk /dev/sdb;
 address 192.168.1.13:7898;
 meta-disk internal;
 }
}
```

配置修改完毕后执行如下命令初始化：

drbdadm create-md r0 ;/etc/init.d/drbd restart ;/etc/init.d/drbd status

如下图:



以上步骤，需要在两台服务器都执行，两台都配置完毕后，在 node2 从上面执行如下命令：

/etc/init.d/drbd status 看到如下信息，表示目前两台都为从，我们需要设置 node1 为 master，命令如下：

drbdadm -- --overwrite-data-of-peer primary all

mkfs.ext4 /dev/drbd0

mkdir /app ;mount /dev/drbd0 /app

自此，DRBD 配置完毕，往/app 目录写入任何东西，当 master 出现宕机或者其他故障，手动切换到 backup，数据没有任何丢失，相当于两台服务器做网络 RAID1。

25.3 Keepalived+DRBD 配置整合实战

wget http://www.keepalived.org/software/keepalived-1.1.15.tar.gz ; tar -xzvf keepalived-1.1.15.tar.gz ;cd keepalived-1.1.15 ; ./configure ; make ;make install

DIR=/usr/local/ ;cp $DIR/etc/rc.d/init.d/keepalived /etc/rc.d/init.d/ ; cp $DIR/etc/sysconfig/keepalived /etc/sysconfig/ ;

mkdir -p /etc/keepalived ; cp $DIR/sbin/keepalived /usr/sbin/

两台服务器均安装 keepalived，并进行配置，首先在 node1（master）上配置，keepalived.conf 内容如下：

```
! Configuration File for keepalived
global_defs {
    router_id LVS_DEVEL
}
vrrp_script check_mysql {
            script "/data/sh/check_mysql.sh"
            interval 5
            }
vrrp_instance VI_1 {
      state MASTER
      interface eth0
      virtual_router_id 52
      priority 100
      advert_int 1
      authentication {
            auth_type PASS
            auth_pass 1111
```

```
    }
    virtual_ipaddress {
        192.168.149.100
         }
        track_script {
        check_mysql
        }
}
```

然后创建 check_mysql.sh 检测脚本，内容如下：

```
#!/bin/sh
A=`ps -C mysqld --no-header |wc -l`
if
[ $A -eq 0 ];then
/bin/umount /app/
drbdadm secondary r0
killall keepalived
fi
```

添加 node2（backup）上配置，keepalived.conf 内容如下：

```
! Configuration File for keepalived
global_defs {
   router_id LVS_DEVEL
}
```

```
vrrp_sync_group VI{
group {
        VI_1
}
notify_master /data/sh/master.sh
notify_backup /data/sh/backup.sh
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth0
    virtual_router_id 52
    priority 90
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {
        192.168.149.100
         }
}
```

25.4 Keepalived+DRBD 故障切换演练

创建 master.sh 检测脚本，内容如下：

```
#!/bin/bash
drbdadm primary r0
/bin/mount /dev/drbd0 /app/
/etc/init.d/mysqld start
```

创建 backup.sh 检测脚本，内容如下：

```
#!/bin/bash
/etc/init.d/mysqld stop
/bin/umount /dev/drbd0
drbdadm secondary r0
```

发生脑裂恢复步骤如下：

Master 执行命令：

```
drbdadm   secondary r0
drbdadm   --   --discard-my-data   connect r0
drbdadm   --   --overwrite-data-of-peer primary all
```

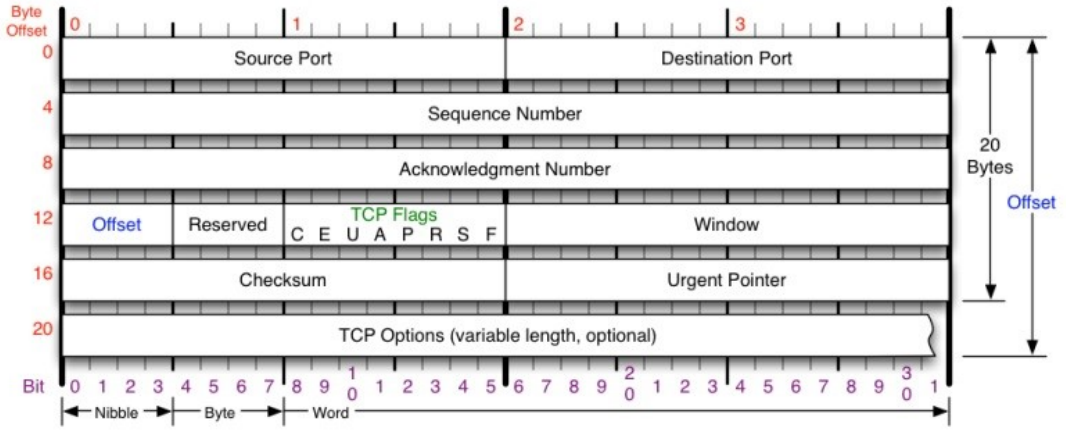
Backup 上执行命令：

```
drbdadm   secondary r0
drbdadm   connect r0
```

第26章 Linux 性能优化实战篇

26.1 TCP 协议详解

优化 Linux 内核，首先要掌握 TCP 协议相关信息，TCP/IP 数据报文的内容如下图所示:



IP 数据包详解如下:

- Source Port 和 Destination Port:分别占用 16 位，表示源端口号和目的端口号；用于区别主机中的不同进程，而 IP 地址是用来区分不同的主机的，源端口号和目的端口号配合上 IP 首部中的源 IP 地址和目的 IP 地址就能唯一的确定一个 TCP 连接；
- Sequence Number:用来标识从 TCP 发端向 TCP 收端发送的数据字节流，它表示在这个报文段中的的第一个数据字节在数据流中的序号；主要用来解决网络报乱序的问题；
- Acknowledgment Number:32 位确认序列号包含发送确认的一端所期望收到的下一个序号，因此，确认序号应当是上次已成功收到数据字节序号加 1。不过，只有当标志位中的 ACK 标志（下面介绍）为 1 时该确认序列号的字段才有效。主要用来解决不丢包的问题；
- Offset:给出首部中 32 bit 字的数目，需要这个值是因为任选字段的长度是可变的。这个字段占 4bit（最多能表示 15 个 32bit 的的字，即 4*15=60 个字节的首部长度），因此 TCP 最多有 60 字节的首部。然而，没有任选字段，正常的长度是 20 字节；

- TCP Flags:TCP 首部中有 6 个标志比特，它们中的多个可同时被设置为 1，主要是用于操控 TCP 的状态机的，依次为 URG，ACK，PSH，RST，SYN，FIN。每个标志位的意思如下：

① URG：此标志表示 TCP 包的紧急指针域（后面马上就要说到）有效，用来保证 TCP 连接不被中断，并且督促中间层设备要尽快处理这些数据；

② ACK：此标志表示应答域有效，就是说前面所说的 TCP 应答号将会包含在 TCP 数据包中；有两个取值：0 和 1，为 1 的时候表示应答域有效，反之为 0；

③ PSH：这个标志位表示 Push 操作。所谓 Push 操作就是指在数据包到达接收端以后，立即传送给应用程序，而不是在缓冲区中排队；

④ RST：这个标志表示连接复位请求。用来复位那些产生错误的连接，也被用来拒绝错误和非法的数据包；

⑤ SYN：表示同步序号，用来建立连接。SYN 标志位和 ACK 标志位搭配使用，当连接请求的时候，SYN=1，ACK=0；连接被响应的时候，SYN=1，ACK=1；这个标志的数据包经常被用来进行端口扫描。扫描者发送一个只有 SYN 的数据包，如果对方主机响应了一个数据包回来 ，就表明这台主机存在这个端口；但是由于这种扫描方式只是进行 TCP 三次握手的第一次握手，因此这种扫描的成功表示被扫描的机器不很安全，一台安全的主机将会强制要求一个连接严格的进行 TCP 的三次握手；

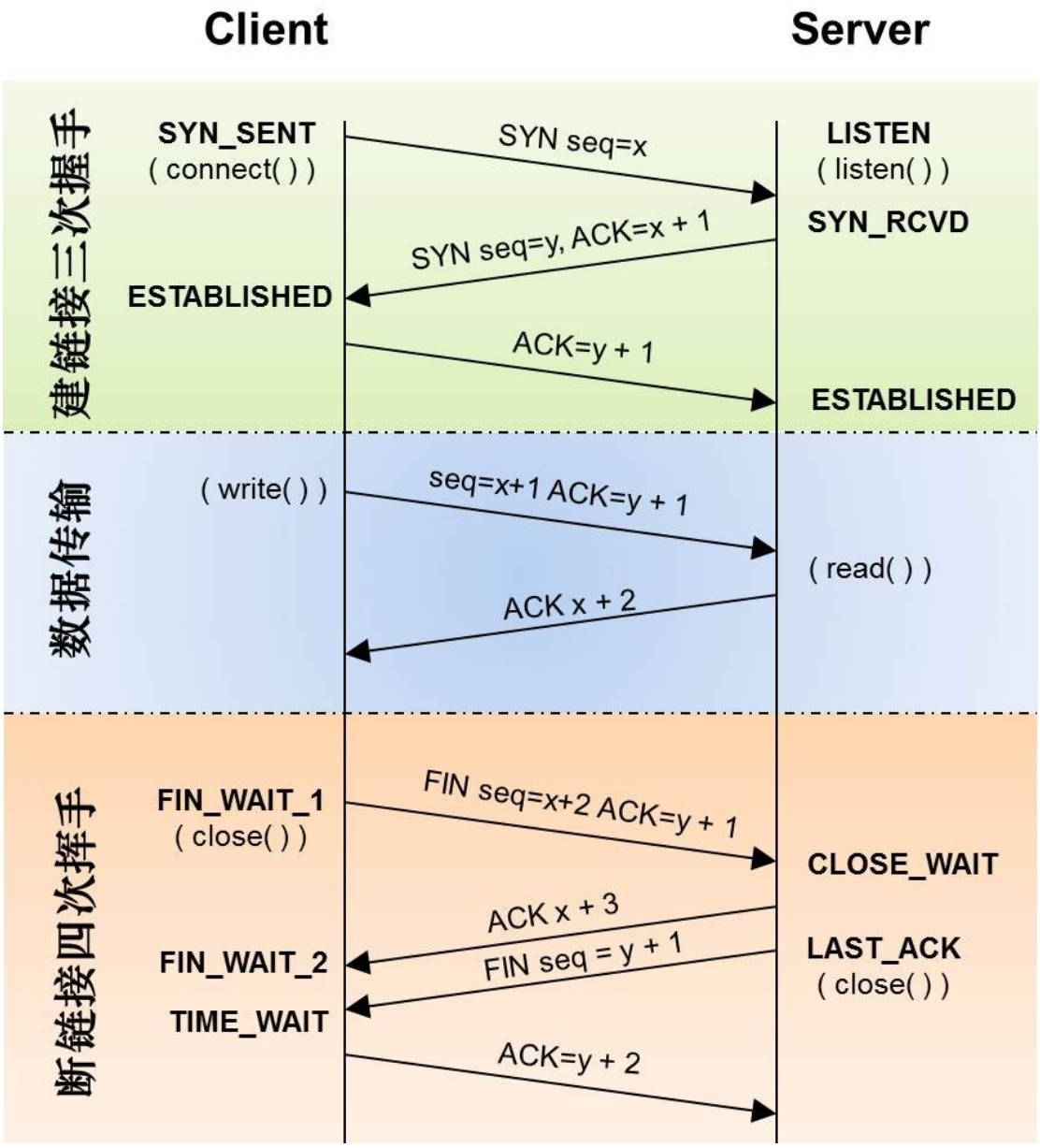
⑥ FIN： 表示发送端已经达到数据末尾，也就是说双方的数据传送完成，没有数据可以传送了，发送 FIN 标志位的 TCP 数据包后，连接将被断开。这个标志的数据包也经常被用于进行端口扫描。

- Window:窗口大小，也就是有名的滑动窗口，用来进行流量控制；

26.2 TCP 三次握手及四次断开

TCP 是面向连接的，无论哪一方向另一方发送数据之前，都必须先在双方之间建立一条连接。在 TCP/IP 协议中，TCP 协议提供可靠的连接服务，连接是通过三次握手进行初始

化的。三次握手的目的是同步连接双方的序列号和确认号并交换 TCP 窗口大小信息。



TCP 三次握手原理：

- 第一次握手：建立连接。客户端发送连接请求报文段，将 SYN 位置为 1，Sequence Number 为 x；然后客户端进入 SYN_SEND 状态，等待服务器的确认；
- 第二次握手：服务器收到 SYN 报文段。服务器收到客户端的 SYN 报文段，需要对这个 SYN 报文段进行确认，设置 Acknowledgment Number 为 x+1(Sequence Number+1)；同时，自己自己还要发送 SYN 请求信息，将 SYN 位置为 1，Sequence Number 为 y；服务器端将上述所有信息放到一个报文段（即 SYN+ACK 报文段）中，

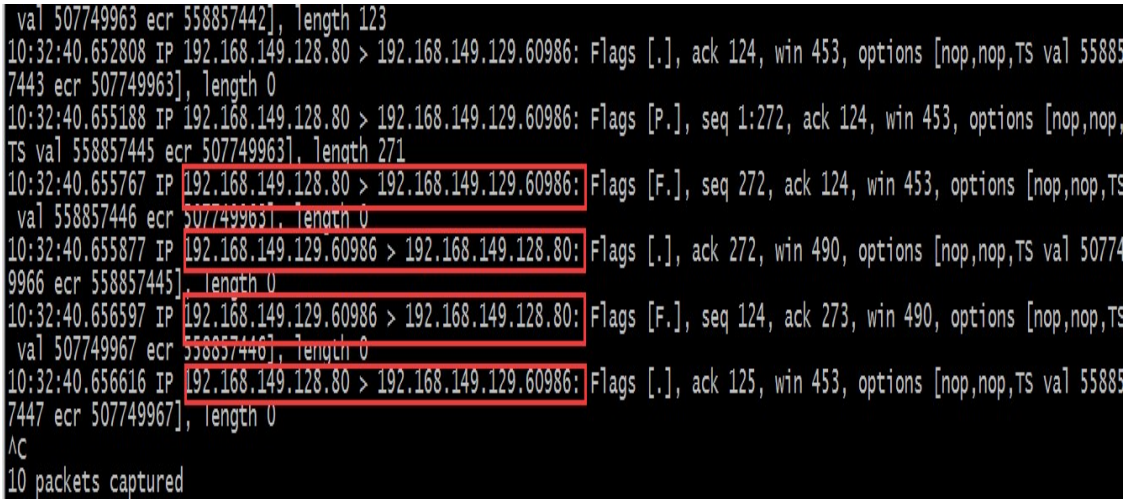
一并发送给客户端，此时服务器进入 SYN_RECV 状态；

- 第三次握手：客户端收到服务器的 SYN+ACK 报文段。然后将 Acknowledgment Number 设置为 y+1，向服务器发送 ACK 报文段，这个报文段发送完毕以后，客户端和服务器端都进入 ESTABLISHED 状态，完成 TCP 三次握手。

```
tcpdump -nn port 80 and host 192.168.149.129
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
10:25:02.979356 IP 192.168.149.129.60980 > 192.168.149.128.80: Flags [S], seq 1287430895, win 14600, options [mss 1460,sackOK,TS val 507292290 ecr 0,nop,wscale 5], length 0
10:25:02.979395 IP 192.168.149.128.80 > 192.168.149.129.60980: Flags [S.], seq 983948645, ack 1287430896, win 14480, options [mss 1460,sackOK,TS val 558399769 ecr 507292290,nop,wscale 5], length 0
10:25:02.979583 IP 192.168.149.129.60980 > 192.168.149.128.80: Flags [.], ack 1, win 457, options [nop,nop,TS val 507292290 ecr 558399769], length 0
10:25:02.979744 IP 192.168.149.129.60980 > 192.168.149.128.80: Flags [P.], seq 1:124, ack 1, win 457, options [nop,nop,TS val 507292290 ecr 558399769], length 123
10:25:02.979791 IP 192.168.149.128.80 > 192.168.149.129.60980: Flags [.], ack 124, win 453, options [nop,nop,TS val 558399770 ecr 507292290], length 0
10:25:02.986587 IP 192.168.149.128.80 > 192.168.149.129.60980: Flags [P.], seq 1:269, ack 124, win 453, options [nop,nop,TS val 558399776 ecr 507292290], length 268
10:25:02.986937 IP 192.168.149.129.60980 > 192.168.149.128.80: Flags [.], ack 269, win 490, options [nop,nop,TS val 507292297 ecr 558399776], length 0
10:25:02.987124 IP 192.168.149.128.80 > 192.168.149.129.60980: Flags [F.], seq 269, ack 124, win 453, options [nop,nop,TS val 558399777 ecr 507292297], length 0
10:25:02.987612 IP 192.168.149.129.60980 > 192.168.149.128.80: Flags [F.], seq 124, ack 270, win 490, options [nop,nop,TS val 507292298 ecr 558399777], length 0
10:25:02.987631 IP 192.168.149.128.80 > 192.168.149.129.60980: Flags [.], ack 125, win 453, options [nop,nop,TS val 558399778 ecr 507292298], length 0
^C
```

TCP 四次挥手原理：

- 第一次挥手：主机 A（可以使客户端，可以是服务器端），设置 Sequence Number 和 Acknowledgment Number，向主机 B 发送一个 FIN 报文段；此时，主机 A 进入 FIN_WAIT_1 状态；这表示主机 A 没有数据要发送给主机 B；
- 第二次挥手：主机 B 收到了主机 A 发送的 FIN 报文段，向主机 A 回一个 ACK 报文段，Acknowledgment Number 为 Sequence Number 加 1；主机 A 进入 FIN_WAIT_2 状态；主机 B 告诉主机 A，我“同意”你的关闭请求；
- 第三次挥手：主机 B 向主机 A 发送 FIN 报文段，请求关闭连接，同时主机 B 进入 LAST_ACK 状态；
- 第四次挥手：主机 A 收到主机 B 发送的 FIN 报文段，向主机 B 发送 ACK 报文段，然后主机 A 进入 TIME_WAIT 状态；主机 B 收到主机 A 的 ACK 报文段以后，就关闭连接；此时，主机 A 等待 2MSL 后依然没有收到回复，则证明 Server 端已正常关闭，那好，主机 A 也可以关闭连接。



26.3 优化 Linux 文件打开最大数

vi /etc/security/limits.conf

* soft noproc 60000

* hard noproc 65535

* soft nofile 65535

* hard nofile 65535

为了防止失控的进程破坏系统的性能,Unix 和 Linux 跟踪进程使用的大部分资源，允许用户和系统管理员使用对进程的资源限制,设置的限制有两种： 硬限制和软限制:

hard 硬限制是可以在任何时候任何进程中设置 但硬限制只能由超级用户修改。

soft 软限制是内核实际执行的限制，任何进程都可以将软限制设置为任意小于等于对进程限制的硬限制的值，(noproc)最大线程数和(nofile)文件数。

26.4 内核参数的优化

vi /etc/sysctl.conf

net.ipv4.tcp_max_tw_buckets = 6000

timewait 的数量，默认是 180000。

net.ipv4.ip_local_port_range = 1024 65000

允许系统打开的端口范围。

net.ipv4.tcp_tw_recycle = 1

启用 timewait 快速回收。

net.ipv4.tcp_tw_reuse = 1

开启重用。允许将 TIME-WAIT sockets 重新用于新的 TCP 连接。

net.ipv4.tcp_syncookies = 1

开启 SYN Cookies，当出现 SYN 等待队列溢出时，启用 cookies 来处理。

net.core.somaxconn = 262144

web 应用中 listen 函数的 backlog 默认会给我们内核参数的 net.core.somaxconn 限制到 128，而 nginx 定义的 NGX_LISTEN_BACKLOG 默认为 511，所以有必要调整这个值。

net.core.netdev_max_backlog = 262144

每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。

net.ipv4.tcp_max_orphans = 262144

系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤儿连接将即刻被复位并打印出警告信息。这个限制仅仅是为了防止简单的 DoS 攻击，不能过分依靠它或者人为地减小这个值，更应该增加这个值(如果增加了内存之后)。

net.ipv4.tcp_max_syn_backlog = 262144

记录的那些尚未收到客户端确认信息的连接请求的最大值。对于有 128M 内存的系统而言，缺省值是 1024，小内存的系统则是 128。

net.ipv4.tcp_timestamps = 0

时间戳可以避免序列号的卷绕。一个 1Gbps 的链路肯定会遇到以前用过的序列号。时间戳能够让内核接受这种“异常”的数据包。这里需要将其关掉。

net.ipv4.tcp_synack_retries = 1

为了打开对端的连接，内核需要发送一个 SYN 并附带一个回应前面一个 SYN 的 ACK。也就是所谓三次握手中的第二次握手。这个设置决定了内核放弃连接之前发送 SYN+ACK 包的数量。

net.ipv4.tcp_syn_retries = 1

在内核放弃建立连接之前发送 SYN 包的数量。

net.ipv4.tcp_fin_timeout = 1

如果套接字由本端要求关闭，这个参数决定了它保持在 FIN-WAIT-2 状态的时间。对端可以出错并永远不关闭连接，甚至意外当机。缺省值是 60 秒。2.2 内核的通常值是 180 秒，你可以按这个设置，但要记住的是，即使你的机器是一个轻载的 WEB 服务器，也有因为大量的死套接字而内存溢出的风险，FIN- WAIT-2 的危险性比 FIN-WAIT-1 要小，因为它最多只能吃掉 1.5K 内存，但是它们的生存期长些。

net.ipv4.tcp_keepalive_time = 30

当 keepalive 起用的时候，TCP 发送 keepalive 消息的频度。缺省是 2 小时。

完整的内核优化脚本：

```
net.ipv4.ip_forward = 0

net.ipv4.conf.default.rp_filter = 1
```

```
net.ipv4.conf.default.accept_source_route = 0

kernel.sysrq = 0

kernel.core_uses_pid = 1

net.ipv4.tcp_syncookies = 1

kernel.msgmnb = 65536

kernel.msgmax = 65536

kernel.shmmax = 68719476736

kernel.shmall = 4294967296

net.ipv4.tcp_max_tw_buckets = 10000

net.ipv4.tcp_sack = 1

net.ipv4.tcp_window_scaling = 1

net.ipv4.tcp_rmem = 4096            87380     4194304

net.ipv4.tcp_wmem = 4096            16384     4194304

net.core.wmem_default = 8388608

net.core.rmem_default = 8388608

net.core.rmem_max = 16777216

net.core.wmem_max = 16777216

net.core.netdev_max_backlog = 262144

net.core.somaxconn = 262144

net.ipv4.tcp_max_orphans = 3276800

net.ipv4.tcp_max_syn_backlog = 262144

net.ipv4.tcp_timestamps = 0
```

```
net.ipv4.tcp_synack_retries = 1

net.ipv4.tcp_syn_retries = 1

net.ipv4.tcp_tw_recycle = 1

net.ipv4.tcp_tw_reuse = 1

net.ipv4.tcp_mem = 94500000 915000000 927000000

net.ipv4.tcp_fin_timeout = 1

net.ipv4.tcp_keepalive_time = 30

net.ipv4.ip_local_port_range = 1024      65530

net.ipv4.icmp_echo_ignore_all = 1
```

26.5 常见内核报错解析

- net.ipv4.tcp_max_tw_buckets 错误：

Sep 23 04:45:54 localhost **kernel: possible SYN flooding on port 80. Sending cookies.**

Sep 23 04:45:55 localhost kernel: TCP: time wait bucket table overflow

Sep 23 04:45:55 localhost kernel: TCP: time wait bucket table overflow

Sep 23 04:45:55 localhost kernel: TCP: time wait bucket table overflow

Sep 23 04:45:55 localhost kernel: TCP: time wait bucket table overflow

Sep 23 04:45:55 localhost kernel: TCP: time wait bucket table overflow

Sep 23 04:45:55 localhost kernel: TCP: time wait bucket table overflow

如上错误是由于 net.ipv4.tcp_max_tw_buckets 设置过小导致，如果内核有如上错误，

我们需要增加 net.ipv4.tcp_max_tw_buckets 的值。

26.6 Too many open files 错误

如果后台报错，大量的 too many open files 错误，一般主要是 JAVA 应用出现这类错误比较多。我们需要设置内核打开文件最大数。

ulimit -SHn 51200 临时生效，如果想永久生效，需要写入到系统内核里面：

vi /etc/security/limits.conf

* soft nproc 65535

* hard nproc 65535

* soft nofile 65535

* hard nofile 65535

然后 exit 退出，重新登录即生效，也可以写在/etc/profile 文件里。

26.7 影响务器性能因素

影响 Linux 服务器性能的因素有很多，这里大致分为如下两类：

26.8 操作系统级

a) 内存

b) CPU

c) 磁盘 I/O

d) 网络 I/O 带宽

26.9 程序应用级

e) Nginx

f) Mysql

g) Tomcat

h) PHP

i) 应用程序代码

26.10 系统性能评估标准

影响性能因素	评判标准		
	好	坏	糟糕
CPU	user% + sys%< 70%	user% + sys%= 85%	user% + sys% >=90%
内存	Swap In（si）=0 Swap Out（so）=0	Per CPU with 10 page/s	More Swap In & Swap Out
磁盘	iowait % < 20%	iowait % =35%	iowait % >= 50%

其中：

%user：表示 CPU 处在用户模式下的时间百分比。

%sys：表示 CPU 处在系统模式下的时间百分比。

%iowait：表示 CPU 等待输入输出完成时间的百分比。

swap in：即 si，表示虚拟内存的页导入，即从 SWAP DISK 交换到 RAM

swap out：即 so，表示虚拟内存的页导出，即从 RAM 交换到 SWAP DISK。

26.11 系统性能分析工具

1.常用系统命令

vmstat、sar、iostat、netstat、free、ps、top、iftop 等

2.常用组合方式

vmstat、sar、iostat 检测是否是 CPU 瓶颈

free、vmstat 检测是否是内存瓶颈

iostat 检测是否是磁盘 I/O 瓶颈

netstat、iftop 检测是否是网络带宽瓶颈

26.12 Linux 性能评估与优化

1) 系统整体性能评估（uptime 命令）

[root@web1 ~]# uptime

16:38:00 up 118 days, 3:01, 5 users, load average: 1.22, 1.02, 0.91

这里需要注意的是：load average 这个输出值，这三个值的大小一般不能大于系统 CPU 的个数，例如，本输出中系统有 8 个 CPU,如果 load average 的三个值长期大于 8 时，说明 CPU 很繁忙，负载很高，可能会影响系统性能，但是偶尔大于 8 时，倒不用担心，一般不会影响系统性能。相反，如果 load average 的输出值小于 CPU 的个数，则表示 CPU 还有空闲的时间片，比如本例中的输出，CPU 是非常空闲的。

2) CPU 性能评估

1) 利用 vmstat 命令监控系统 CPU

该命令可以显示关于系统各种资源之间相关性能的简要信息，这里我们主要用它来看 CPU 一个负载情况。

下面是 vmstat 命令在某个系统的输出结果：

```
[root@PT-171123 ~]# vmstat 2 10
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 12529508 372016 16116888    0    0     0     2    0    0  0  0 100  0  0
 0  0      0 12529520 372016 16116888    0    0     0    40  137  214  0  0 100  0  0
 0  0      0 12529520 372016 16116888    0    0     0     0  135  219  0  0 100  0  0
 0  0      0 12529520 372016 16116888    0    0     0     0  120  205  0  0 100  0  0
 0  0      0 12529556 372016 16116888    0    0     0     8  139  232  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     0  159  250  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0    26  129  213  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     6  120  210  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     0  122  203  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     0  123  209  0  0 100  0  0
[root@PT-171123 ~]#
```

r 列表示运行和等待 cpu 时间片的进程数，这个值如果长期大于系统 CPU 的个数，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。

us 列显示了用户进程消耗的 CPU 时间百分比。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%，就需要考虑优化程序或算法。

sy 列显示了内核进程消耗的 CPU 时间百分比。Sy 的值较高时，说明内核消耗的 CPU 资源很多。

根据经验，us+sy 的参考值为 80%，如果 us+sy 大于 80%说明可能存在 CPU 资源不足。

2) 利用 sar 命令监控系统 CPU

sar 功能很强大，可以对系统的每个方面进行单独的统计，但是使用 sar 命令会增加系统开销，不过这些开销是可以评估的，对系统的统计结果不会有很大影响。

下面是 sar 命令对某个系统的 CPU 统计输出：

```
[root@PT-171123 ~]# sar -u 2 10
Linux 2.6.32-279.el6.x86_64 (PT-171123.360buy.com)      01/22/2015      _x86_64_        (16 CPU)

03:04:51 PM      CPU     %user     %nice   %system   %iowait    %steal     %idle
03:04:53 PM      all      0.00      0.00      0.03      0.00      0.00     99.97
03:04:55 PM      all      0.00      0.00      0.03      0.00      0.00     99.97
03:04:57 PM      all      0.03      0.00      0.03      0.00      0.00     99.94
03:04:59 PM      all      0.00      0.00      0.03      0.00      0.00     99.97
03:05:01 PM      all      0.00      0.00      0.03      0.00      0.00     99.97
03:05:03 PM      all      0.00      0.00      0.03      0.00      0.00     99.97
03:05:05 PM      all      0.00      0.00      0.00      0.00      0.00    100.00
03:05:07 PM      all      0.03      0.00      0.03      0.00      0.00     99.94
03:05:09 PM      all      0.03      0.00      0.03      0.00      0.00     99.94
```

对上面每项的输出解释如下：

%user 列显示了用户进程消耗的 CPU 时间百分比。

%nice 列显示了运行正常进程所消耗的 CPU 时间百分比。

%system 列显示了系统进程消耗的 CPU 时间百分比。

%iowait 列显示了 IO 等待所占用的 CPU 时间百分比

%steal 列显示了在内存相对紧张的环境下 pagein 强制对不同的页面进行的 steal 操作 。

%idle 列显示了 CPU 处在空闲状态的时间百分比。

问题

1） 你是否遇到过系统 CPU 整体利用率不高，而应用缓慢的现象？

在一个多 CPU 的系统中，如果程序使用了单线程，会出现这么一个现象，CPU 的整体使用率不高，但是系统应用却响应缓慢，这可能是由于程序使用单线程的原因，单线程只使用一个 CPU，导致这个 CPU 占用率为 100%，无法处理其它请求，而其它的 CPU 却闲置，这就导致了整体 CPU 使用率不高，而应用缓慢现象的发生。

3） 内存性能评估

1） 利用 free 指令监控内存

free 是监控 linux 内存使用状况最常用的指令，看下面的一个输出：

```
You have mail in /var/spool/mail/root
[root@PT-171123 ~]#
[root@PT-171123 ~]# free -m
             total       used       free     shared    buffers     cached
Mem:         32097      19861      12235          0        363      15739
-/+ buffers/cache:       3759      28337
Swap:          499          0        499
[root@PT-171123 ~]#
```

一般有这样一个经验公式：应用程序可用内存/系统物理内存>70%时，表示系统内存资源非常充足，不影响系统性能，应用程序可用内存/系统物理内存<20%时，表示系统内存资源紧缺，需要增加系统内存，20%<应用程序可用内存/系统物理内存<70%时，表示系统内存资源基本能满足应用需求，暂时不影响系统性能。

2) 利用 vmstat 命令监控内存

```
[root@PT-171123 ~]# vmstat 2 10
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 12529508 372016 16116888    0    0     0     2    0    0  0  0 100  0  0
 0  0      0 12529520 372016 16116888    0    0     0    40  137  214  0  0 100  0  0
 0  0      0 12529520 372016 16116888    0    0     0     0  135  219  0  0 100  0  0
 0  0      0 12529520 372016 16116888    0    0     0     0  120  205  0  0 100  0  0
 0  0      0 12529556 372016 16116888    0    0     0     8  139  232  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     0  159  250  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0    26  129  213  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     6  120  210  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     0  122  203  0  0 100  0  0
 0  0      0 12529556 372016 16116896    0    0     0     0  123  209  0  0 100  0  0
[root@PT-171123 ~]#
```

swpd 列表示切换到内存交换区的内存数量（以 k 为单位）。如果 swpd 的值不为 0，或者比较大，只要 si、so 的值长期为 0，这种情况下一般不用担心，不会影响系统性能。

free 列表示当前空闲的物理内存数量（以 k 为单位）

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

cache 列表示 page cached 的内存数量，一般作为文件系统 cached，频繁访问的文件都会被 cached，如果 cache 值较大，说明 cached 的文件数较多，如果此时 IO 中 bi 比较小，说明文件系统效率比较好。

si 列表示由磁盘调入内存，也就是内存进入内存交换区的数量。

so 列表示由内存调入磁盘，也就是内存交换区进入内存的数量。

一般情况下，si、so 的值都为 0，如果 si、so 的值长期不为 0，则表示系统内存不足。需要增加系统内存。

4) 磁盘 I/O 性能评估

1) 磁盘存储基础

熟悉 RAID 存储方式，可以根据应用的不同，选择不同的 RAID 方式。

尽可能用内存的读写代替直接磁盘 I/O，使频繁访问的文件或数据放入内存中进行操作处理，因为内存读写操作比直接磁盘读写的效率要高千倍。

将经常进行读写的文件与长期不变的文件独立出来，分别放置到不同的磁盘设备上。

对于写操作频繁的数据，可以考虑使用裸设备代替文件系统。

使用裸设备的优点有：

数据可以直接读写，不需要经过操作系统级的缓存，节省了内存资源，避免了内存资源争用。

避免了文件系统级的维护开销，比如文件系统需要维护超级块、I-node 等。

避免了操作系统的 cache 预读功能，减少了 I/O 请求。

使用裸设备的缺点是：

数据管理、空间管理不灵活，需要很专业的人来操作。

2) 利用 iostat 评估磁盘性能

```
[root@PT-171123 ~]#
[root@PT-171123 ~]# iostat -d 1 10
Linux 2.6.32-279.el6.x86_64 (PT-171123.360buy.com)     01/22/2015      _x86_64_        (16 CPU)

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               2.02         9.44        48.65   75171924  387376816

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               0.00         0.00         0.00          0          0
```

对上面每项的输出解释如下：

Blk_read/s 表示每秒读取的数据块数。

Blk_wrtn/s 表示每秒写入的数据块数。

Blk_read 表示读取的所有块数。

Blk_wrtn 表示写入的所有块数。

可以通过 Blk_read/s 和 Blk_wrtn/s 的值对磁盘的读写性能有一个基本的了解，如果 Blk_wrtn/s 值很大，表示磁盘的写操作很频繁，可以考虑优化磁盘或者优化程序，如果 Blk_read/s 值很大，表示磁盘直接读取操作很多，可以将读取的数据放入内存中进行操作。

对于这两个选项的值没有一个固定的大小，根据系统应用的不同，会有不同的值，但是有一个规则还是可以遵循的：长期的、超大的数据读写，肯定是不正常的，这种情况一定会影响系统性能。

3) 利用 sar 评估磁盘性能

通过 sar -d 组合，可以对系统的磁盘 IO 做一个基本的统计，请看下面的一个输出：

```
[root@PT-171123 ~]# sar -d 1 10
Linux 2.6.32-279.el6.x86_64 (PT-171123.360buy.com)      01/22/2015      _x86_64_        (16 CPU)

03:26:41 PM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
03:26:42 PM    dev8-0      6.00      0.00    120.00     20.00      0.00      0.50      0.50      0.30

03:26:42 PM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
03:26:43 PM    dev8-0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

03:26:43 PM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
03:26:44 PM    dev8-0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

03:26:44 PM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
03:26:45 PM    dev8-0      2.00      0.00     32.00     16.00      0.00      0.00      0.00      0.00

03:26:45 PM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
03:26:46 PM    dev8-0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00

03:26:46 PM       DEV       tps  rd_sec/s  wr_sec/s  avgrq-sz  avgqu-sz     await     svctm     %util
03:26:47 PM    dev8-0     13.86      0.00    110.89      8.00      0.00      0.00      0.00      0.00
```

需要关注的几个参数含义：

await 表示平均每次设备 I/O 操作的等待时间（以毫秒为单位）。

svctm 表示平均每次设备 I/O 操作的服务时间（以毫秒为单位）。

%util 表示一秒中有百分之几的时间用于 I/O 操作。

对以磁盘 IO 性能，一般有如下评判标准：

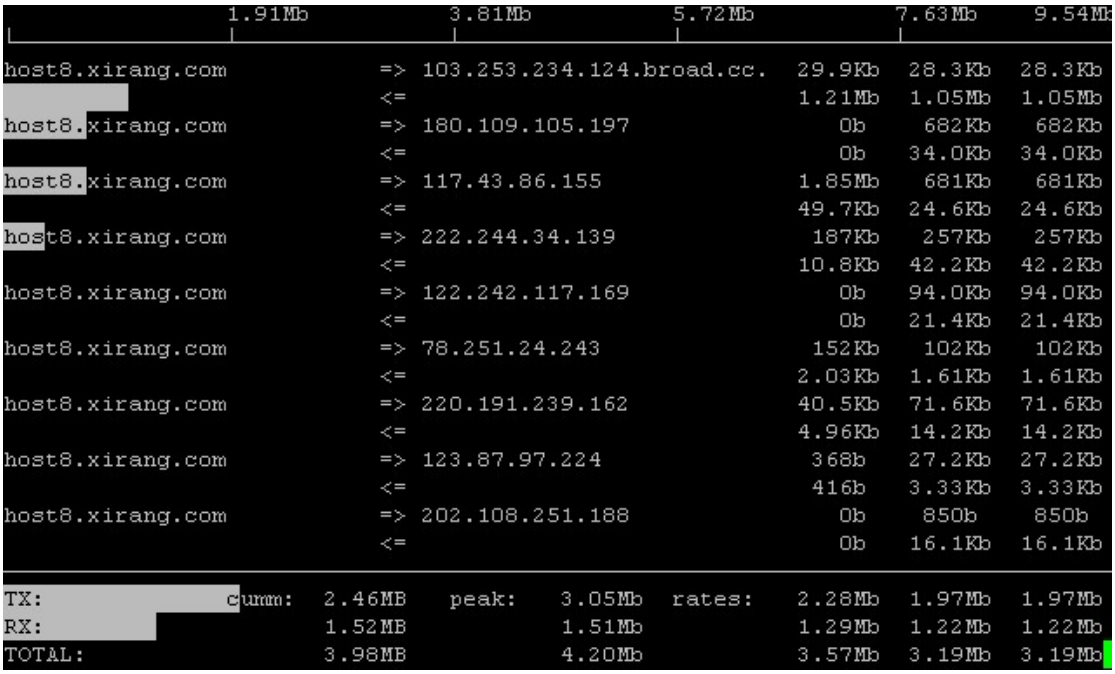
正常情况下 svctm 应该是小于 await 值的，而 svctm 的大小和磁盘性能有关，CPU、内存的负荷也会对 svctm 值造成影响，过多的请求也会间接的导致 svctm 值的增加。

await 值的大小一般取决与 svctm 的值和 I/O 队列长度以及 I/O 请求模式，如果 svctm 的值与 await 很接近，表示几乎没有 I/O 等待，磁盘性能很好，如果 await 的值远高于 svctm 的值，则表示 I/O 队列等待太长，系统上运行的应用程序将变慢，此时可以通过更换更快的硬盘来解决问题。

%util 项的值也是衡量磁盘 I/O 的一个重要指标，如果%util 接近 100%，表示磁盘产生的 I/O 请求太多，I/O 系统已经满负荷的在工作，该磁盘可能存在瓶颈。长期下去，势必影响系统的性能，可以通过优化程序或者通过更换更高、更快的磁盘来解决此问题。

5) 网络性能评估

- 通过 ping 命令检测网络的连通性；
- 通过 netstat –i 组合检测网络接口状况；
- 通过 netstat –r 组合检测系统的路由表信息；
- 通过 sar -n 组合显示系统的网络运行状态；
- 通过 iftop -i eth0 查看网卡流量；



第27章 Docker 虚拟化实战基础篇

27.1 虚拟化技术概述及简介

IT 行业发展到今天，已经从传统技术、传统运维发展到当下的主流技术、自动化运维，未来掌握核心技术是赶上时代变化，拉开跟别人差距的最关键因素。当下主流的 IT 三大技术：虚拟化、云计算、大数据之一的虚拟化技术。虚拟化技术也越来越广泛的应用在企业中，例如百度、阿里巴巴、腾讯、京东、Google 等。

通俗的说，虚拟化就是把物理资源转变为逻辑上可以管理的资源，以打破物理结构间的壁垒，计算元件运行在虚拟的基础上而不是真实的基础上，可以扩大硬件的容量，简化软件的重新配置过程。

虚拟化技术允许一个平台同时运行多个操作系统，并且应用程序都可以在相互独立的空间内运行而互不影响，从而显著提高计算机的工作效率，是一个为了简化管理，优化资源的解决方案。

虚拟化解决方案的底部是要进行虚拟化的物理机器，这台机器可能直接支持虚拟化，也可能不会直接支持虚拟化，那么就需要系统管理程序层的支持。虚拟机管理程序(Virtual machine monitor)，或称为 VMM，可以看作是平台硬件和操作系统的抽象化，VMM 本质是我们常说的虚拟化技术软件。

27.2 KVM 虚拟化概念

KVM 虚拟化全称为 kernel-based Virtual Machine，是一个开源的系统虚拟化模块，基于内核的虚拟机（KVM）是针对包含虚拟化扩展（Intel VT 或 AMD-V）的 x86 硬件上的 Linux 的完全原生的虚拟化解决方案。

KVM 是目前早期互联网广泛使用的虚拟化软件之一，最早由以色列的公司开发，现在已经被 RedHat 公司斥资 1.07 亿美元收购了 KVM 虚拟化管理程序厂商 Qumranet，KVM 虚拟化技术严格来讲，不是一个软件，而是 Linux 内核里面的一种加速虚拟机的功能扩展。

自 Linux 2.6.20 之后集成在 Linux 的各个主要发行版本中。KVM 的虚拟化需要硬件支持（如 Intel VT 技术或者 AMD V 技术)属于完全虚拟化。了解 KVM 之前，我们就需要了解 KVM 和 QEMU、Libvirt 有什么关系呢?

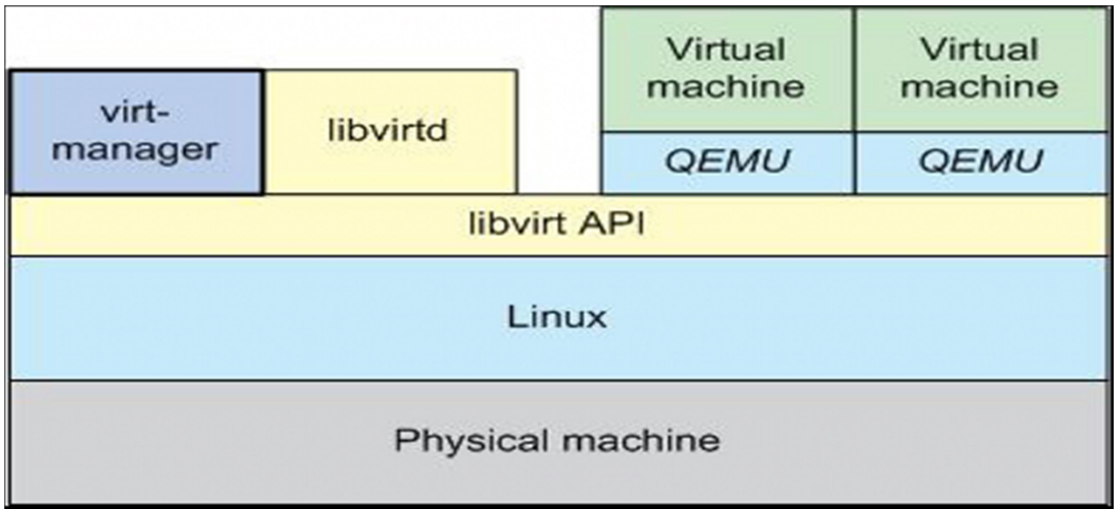
KVM 是一款支持虚拟机的技术，是 Linux 内核中的一个功能模块。它在 Linux2.6.20 之后的任何 Linux 分支中都被支持。它还有一个条件，对硬件要求的条件，必须达到一定标准的硬件架构。

Qemu 是什么呢。其实它也是一款虚拟化技术，就算不使用 kvm，单纯的 qemu 也可以完全实现一个虚拟机。

那为何还会有 Qemu-kvm 这个名词呢？虽然 kvm 的技术已经相当成熟而且可以对很多东西进行隔离，但是在某些方面还是无法虚拟出真实的机器。比如对网卡的虚拟，这个时候就需要另外的技术来做补充，而 qemu-kvm 则是这样一种技术。它补充了 kvm 技术的不足，而且在性能上对 kvm 进行了优化。

Libvirt 又是什么呢。它是一系列提供出来的库函数，用以其他技术调用，来管理机器上的虚拟机。包括各种虚拟机技术，kvm、xen 与 lxc 等，不同虚拟机技术就可以使用不同驱动，都可以调用 libvirt 提供的 api 对虚拟机进行管理。我们创建的各种虚拟机都是基于

libvirt 库及相关命令去管理的。



27.3 ESXI 虚拟化技术概念

Vmware 服务器虚拟化第一个产品命名为：ESX，后来 Vmware 在第 4 版本的时候推出了 ESXI，ESXI 和 ESX 的版本最大的技术区别是内核的变化。

从第 4 版本开始 VMware 把 ESX 及 ESXi 产品统称为 vSphere，但是 VMware 从 5 版本开始以后取消了原来的 ESX 版本，所以现在 VMware 虚拟化产品 vSphere 其实本质是 ESXI，只是两种叫法而已。官方称为 vSphere 虚拟化技术，个人也可以称为 ESXI 虚拟化技术。

VMware vSphere 是业界领先且最可靠的虚拟化平台。vSphere 将应用程序和操作系统从底层硬件分离出来，从而简化了 IT 操作。您现有的应用程序可以看到专有资源，而您的服务器则可以作为资源池进行管理。因此，您的业务将在简化但恢复能力极强的 IT 环境中运行。

VMware、vSphere、Essentials 和 Essentials Plus 套件专为工作负载不足 20 台服务器的 IT 环境而设计，只需极少的投资即可通过经济高效的服务器整合和业务连续性为小型企业提供企业级 IT 管理。结合使用 vSphere Essentials Plus 与 vSphere Storage Appliance 软件，无需共享存储硬件即可实现业务连续性。

VMware ESXI 虚拟化特点如下：

- 确保业务连续性和始终可用的 IT；
- 降低 IT 硬件和运营成本；
- 提高应用程序质量；
- 增强安全性和数据保护能力。

27.4 XEN 虚拟化技术概念

XEN 是一个基于 X86 架构、发展最快、性能最稳定、占用资源最少的开源虚拟化技术。Xen 可以在一套物理硬件上安全的执行多个虚拟机，与 Linux 是一个完美的开源组合，Novell SUSE Linux Enterprise Server 最先采用了 XEN 虚拟技术。它特别适用于服务器应用整合，可有效节省运营成本，提高设备利用率，最大化利用数据中心的 IT 基础架构。

实际上 XEN 出现的时间要早于 KVM 虚拟化，它是由剑桥大学开发的，严格来讲，XEN 是一个开源的虚拟机监视器，属于半虚拟化技术，其架构决定了它注定不是真正的虚拟机，只是自己运行了一个内核的例子。

XEN 虚拟化，同时区分 Xen+pv+和 Xen+hvm，其中 pv 只支持 Linux，而 hvm 则支持 Windows 系统。除此之外，XEN 还拥有更好的可用资源、平台支持、可管理性、实施、支持动态迁移和性能基准等优势。

27.5 Docker 虚拟化技术概念

Docker 是一款轻量级、高性能的虚拟化技术，是目前互联网使用最多的虚拟化技术，Docker 虚拟化技术的本质类似集装箱机制，最早集装箱没有出现的时候，码头上有许多搬运的工人在搬运货物，集装箱出现以后，码头上看到更多的不是工人，而且集装箱的搬运模式更加单一，更加高效，还有其他的好处。

将货物多打包在集装箱里面，可以防止货物之间相互影响。并且到了另外一个码头需要转运的话，有了在集装箱以后，直接把它运送到另一个码头即可，完全可以保证里面的货物是整体的搬迁，并且不会损坏货物本身。

Docker 技术机制跟集装箱类似，Docker 虚拟化 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。

Docker 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、框架或包括系统。

Docker 应该是近年最火爆的技术之一，如果没有听说过，那么你就 out 了，2019 年将开启新的跨越。

Docker 自开源后受到广泛的关注和讨论，以至于 dotCloud 公司后来都改名为 Docker Inc。

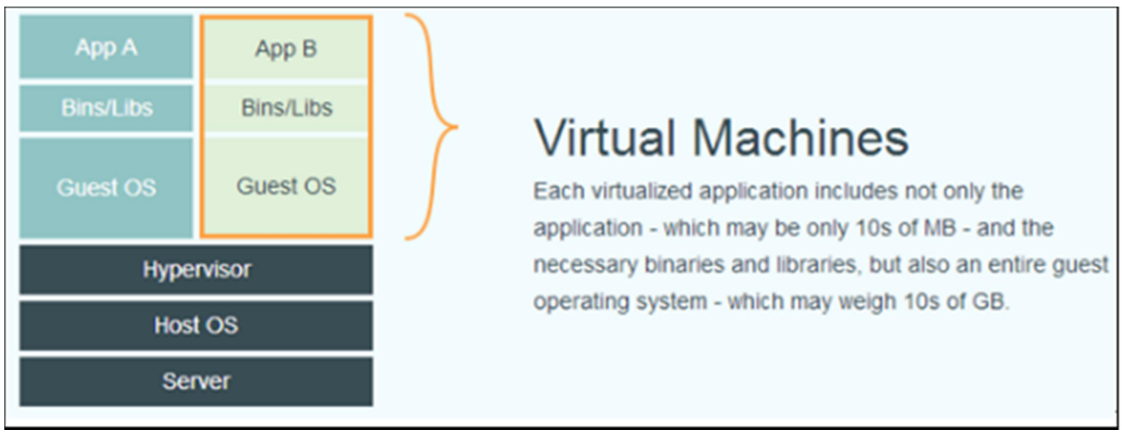
Redhat 已经在其 RHEL6.5 中集中支持 Docker；

Google 也在其 PaaS 产品中广泛应用，Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案。

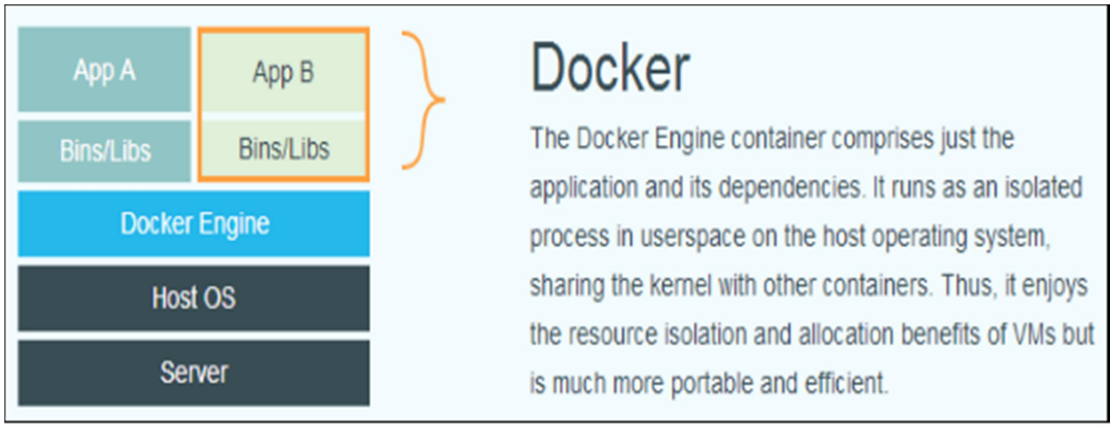
Docker 的基础是 Linux 容器（LXC）等技术。在 LXC 的基础上 Docker 进行了进一步的封装，让用户不需要去关心容器的管理，使得操作更为简便。用户操作 Docker 的容器就像操作一个快速轻量级的虚拟机一样简单。

下面对比了 Docker 和传统虚拟化（KVM、XEN、Hyper-V、ESXI）结构层级方式的不同之处，

图为传统虚拟化结构图：



图为 Docker 虚拟化结构图：



- Docker 虚拟化技术概念&总结：

Docker 虚拟化技术是在硬件的基础上，基于现有的操作系统层面上实现虚拟化，直接复用本地主机的操作系统，直接虚拟生成 Docker 容器，而 Docker 容器上部署相关的 APP 应用（Apache、MYSQL、PHP、JAVA）。

- 传统虚拟化技术概念&总结：

KVM、XEN、ESXI 传统虚拟化（完全、半虚拟化）是在硬件的基础上，基于现有的操作系统层面上实现虚拟化，但是不能复用本地主机的操作系统，而是必须虚拟出自己的 Guest OS 系统，然后在 Guest OS 系统上部署相关的 APP 应用（Apache、MYSQL、PHP、JAVA）。

Docker 虚拟化跟传统 VM 比较具有如下优点：

- 操作启动快：

运行时的性能可以获取极大提升，管理操作（启动，停止，开始，重启等等） 都是以秒或毫秒为单位的。

- 轻量级虚拟化：

你会拥有足够的“操作系统”，仅需添加或减小镜像即可。在一台服务器上可以布署 100~1000 个 Containers 容器。但是传统虚拟化，你虚拟 10-20 个虚拟机就不错了。

- 开源免费：

开源的，免费的，低成本的。由现代 Linux 内核支持并驱动。注* 轻量的 Container 必定可以在一个物理机上开启更多“容器”，注定比 VMs 要便宜。

- 前景及云支持；

正在越来越受欢迎，包括各大主流公司都在推动 docker 的快速发展，性能有很大的优势。随着 Go 语言越来越被人熟知，Docker 的使用也越来越广泛。

27.6 Docker LXC 及 Cgroup 原理剖析

Docker 虚拟化技术结构体系最早为 LXC (Linux Container) +AUFS (Another Union File System）结构组合，Docker 0.9.0 版本开始引入 LibContainer，可以视作 LXC 的替代品。

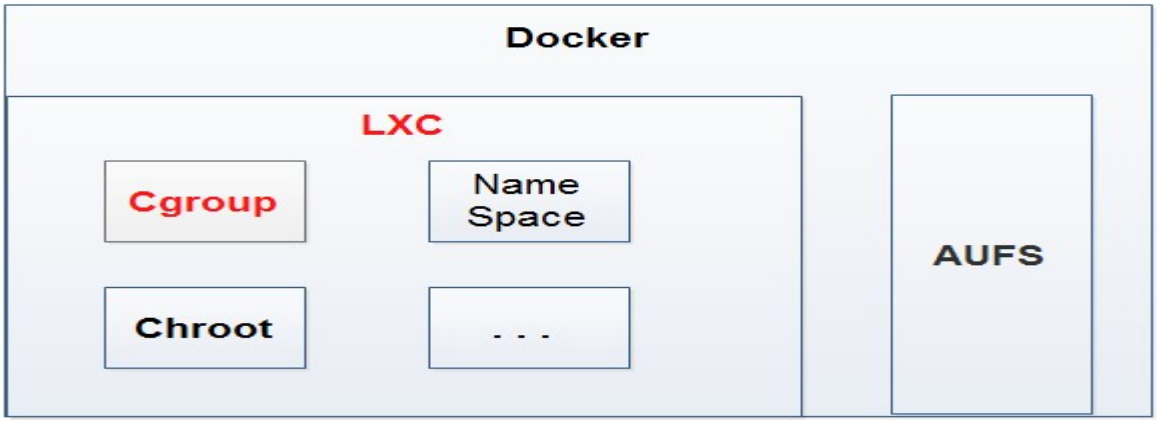
LXC 也是一种虚拟化的解决方案，跟 KVM、XEN、ESXI 虚拟化基于硬件层面虚拟化不同，LXC 是基于是内核级的虚拟化技术，Linux 操作系统软件服务进程之所以能够相互独立，并且系统能够控制每个服务进程的 CPU、内存资源也是得益于 LXC 容器技术。

Docker 虚拟化技术是在 LXC 技术上进一步的封装，比 LXC 技术更完善，并且提供了一系列完整的功能。在 Docker 虚拟化技术中，LXC 主要负责资源管理，而 AUFS 主要负责镜像管理，而 LXC 又包括 Cgroup、NameSpace、Chroot 等组件，并通过 Cgroup 进行资源管理。

从资源管理结构体系上来看，Docker、LXC、Cgroup 三者的关系：

Cgroup 在最底层落实资源管理，LXC 在 cgroup 上封装了一层，Docker 又在 LXC 封装了一层，要深入掌握 Docker 虚拟化技术，需要了解负责资源管理的 CGroup 和 LXC 相关概念和用途。

如图所示为 Docker、LXC、Cgroup、Aufs 结构图：



Cgroups 又名 Control groups，是 Linux 内核提供的一种可以限制、记录、隔离进程组所使用的物理资源（CPU、 Memory、IO、NET）的机制。

Cgroups 最初由 Google 的工程师提出，后来被整合进 Linux 内核中。Cgroups 也是 LXC 为实现虚拟化所使用的资源管理手段，一句话中间：没有 Cgroups 就没有 LXC，没有 LXC，就没有 Docker。

Cgroups 最初的目标是为资源管理提供的一个统一的框架，既整合现有的 Cpuset 等子系统，也为未来开发新的子系统提供接口。现在的 Cgroups 适用于多种应用场景，从单个进程的资源控制，到实现操作系统层次的虚拟化（OS Level Virtualization）。

Linux Container 容器技术可以提供轻量级的虚拟化，以便隔离进程和资源，而且不需要提供指令解释机制以及全虚拟化的其他复杂性。容器有效地将由单个操作系统管理的资源划分到孤立的组中，以更好地在孤立的组之间平衡有冲突的资源使用需求。

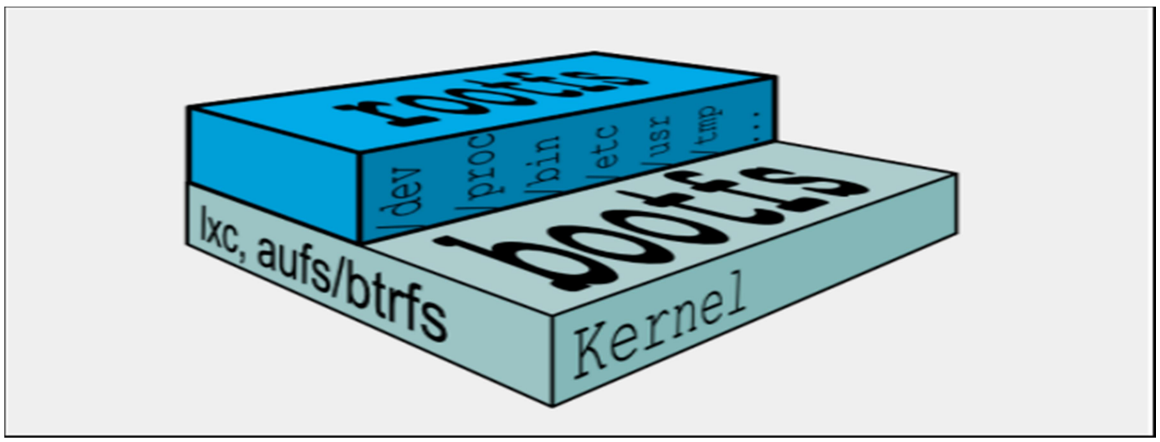
LXC 是建立在 CGroup 基础上，可以理解为：LXC = Cgroup+ namespace + Chroot + veth +用户态控制脚本。

LXC 利用内核的新特性(CGroup)来提供用户空间的对象，用来保证资源的隔离和对于应用或者系统的资源控制。

典型的 Linux 文件系统由 Bootfs 和 Rootfs 两部分组成：

Bootfs(boot file system)主要包含 bootloader 和 kernel，bootloader 主要是引导加载 kernel，当 kernel 被加载到内存中后 bootfs 就被 umount。

Rootfs (root file system) 包含的就是典型 Linux 系统中的/dev，/proc，/bin，/etc 等目录和文件。



Docker 容器的文件系统最早是建立在 Aufs 基础上的，Aufs 是一种 Union FS， 简单来说就是支持将不同的目录挂载到同一个虚拟文件系统下，并实现一种 layer 的概念。

由于 Aufs 未能加入到 Linux 内核，考虑到兼容性问题，加入了 Devicemapper 的支持。

Docker 虚拟化磁盘文件系统，默认使用 Devicemapper 方式，Docker 虚拟化目前支持六种文件系统：AUFS、Btrfs、Device mapper、OverlayFS、ZFS、VFS 等，如图所示：

Technology	Storage driver name
OverlayFS	`overlay`
AUFS	`aufs`
Btrfs	`btrfs`
Device Mapper	`devicemapper`
VFS	`vfs`
ZFS	`zfs`

27.7 AUFS 文件系统简介

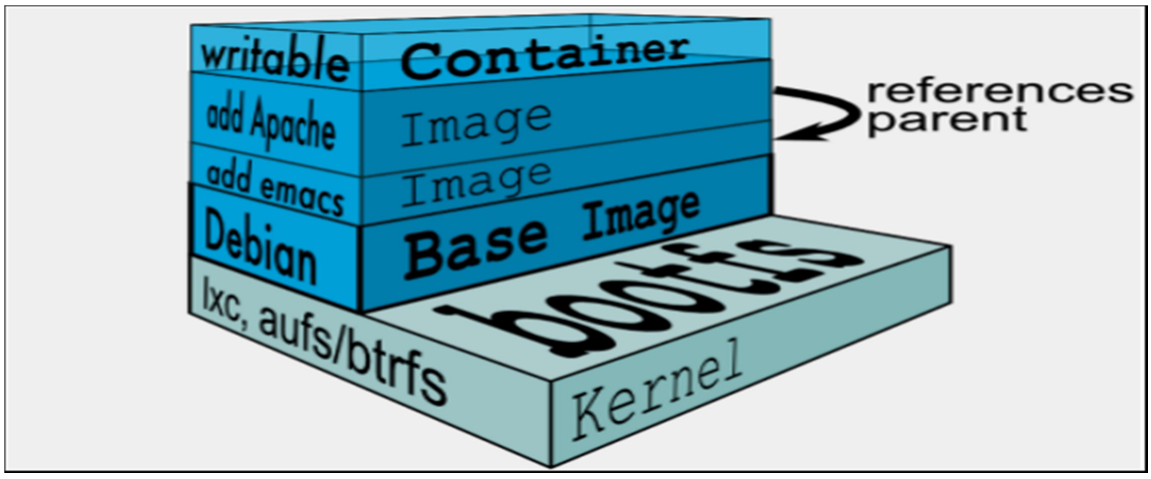
Aufs 将挂载到同一虚拟文件系统下的多个目录分别设置成 read-only，read-write 以及 whiteout-able 权限，对 read-only 目录只能读，而写操作只能实施在 read-write 目录中。重点在于，写操作是在 read-only 上的一种增量操作，不影响 read-only 目录。

当挂载目录的时候要严格按照各目录之间的这种增量关系，将被增量操作的目录优先于在它基础上增量操作的目录挂载，待所有目录挂载结束了，继续挂载一个 read-write 目录，如此便形成了一种层次结构。

传统的 Linux 加载 bootfs 时会先将 rootfs 设为 read-only，然后在系统自检之后将 rootfs 从 read-only 改为 read-write，然后我们就可以在 rootfs 上进行写和读的操作了。但 Docker 的镜像却不是这样，它在 bootfs 自检完毕之后并不会把 rootfs 的 read-only 改为 read-write。而是利用 union mount（UnionFS 的一种挂载机制）将一个或多个 read-only 的 rootfs 加载到之前的 read-only 的 rootfs 层之上。

在加载了这么多层的 rootfs 之后，仍然让它看起来只像是一个文件系统，在 Docker 的体系里把 union mount 的这些 read-only 的 rootfs 叫做 Docker 的镜像。但此时的每一层 rootfs 都是 read-only 的，我们此时还不能对其进行操作。当我们创建一个容器，也就是将 Docker 镜像进行实例化，系统会在一层或是多层 read-only 的 rootfs 之上分配一层空的 read-write 的 rootfs。

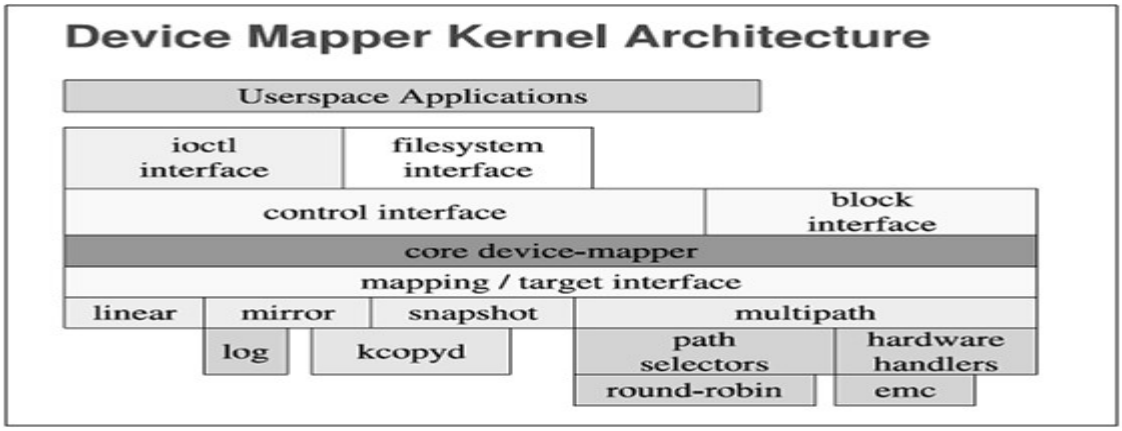
如图所示为一个完整的容器文件系统层级结构图：



27.8 Device Mappber 文件系统简介

Device Mapper 是 Linux2.6 内核中支持逻辑卷管理的通用设备映射机制，它为实现用于存储资源管理的块设备驱动提供了一个高度模块化的内核架构。

Device Mapper 的内核体系架构：



在内核中它通过一个一个模块化的 target driver 插件实现对 IO 请求的过滤或者重新定向等工作，当前已经实现的 target driver 插件包括软 raid、软加密、逻辑卷条带、多路径、镜像、快照等，图中 linear、mirror、snapshot、multipath 表示的就是这些 target driver。Device mapper 进一步体现了在 Linux 内核设计中策略和机制分离的原则，将所有与策略相关的工作放到用户空间完成，内核中主要提供完成这些策略所需要的机制。

Device mapper 用户空间相关部分主要负责配置具体的策略和控制逻辑，比如逻辑设备和哪些物理设备建立映射，怎么建立这些映射关系等等，而具体过滤和重定向 IO 请求的工作由内核中相关代码完成。因此整个 Device mapper 机制由两部分组成-内核空间的 Device mapper 驱动、用户空间的 Device mapper 库以及它提供的 dmsetup 工具。

27.9 OverlayFS 文件系统简介

OverlayFS 是目前使用比较广泛的层次文件系统，是一种类似 Aufs 的一种堆叠文件系统，于 2014 年正式合入 Linux 3.18 主线内核，OverlayFS 文件系统，实现简单，而且性能很好，可以充分利用不同或则相同 Overlay 文件系统的 Page Cache，具有：上下合并、同名遮盖、写时拷贝等特点。

Docker 虚拟化 Overlay 存储驱动利用了很多 OverlayFS 特性来构建和管理镜像与容器的磁盘结构。 从 Docker1.12 起，Docker 也支持 Overlay2 存储驱动，相比于 Overlay 来说，Overlay2 在 inode 优化上更加高效。但 Overlay2 驱动只兼容 Linux kernel 4.0+以上的版本。

OverlayFS 加入 Linux Kernel 主线后，在 Linux Kernel 模块中的名称从 Overlayfs 改名为 Overlay。在真实使用中，OverlayFS 代表整个文件系统，而 Overlay/Overlay2 表示 Docker 的存储驱动。

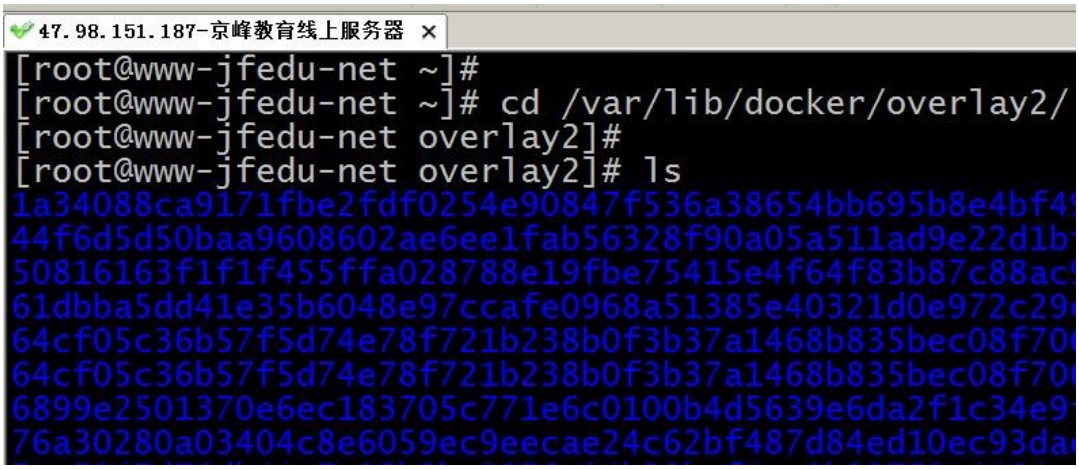
```
Images: 2
Server Version: 1.13.1
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: journald
Cgroup Driver: systemd
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
Swarm: inactive
Runtimes: docker-runc runc
Default Runtime: docker-runc
```

在 Docker 虚拟化技术中，OverlayFS 将一个 Linux 主机中的两个目录组合起来，一个在上，一个在下，对外提供统一的视图。这两个目录就是层 layer，将两个层组

合在一起的技术被成为联合挂载 Union mount。在 OverlayFS 中，上层的目录被称作 upperdir，下层的，目录被称作 lowerdir，对外提供的统一视图被称作 merged。

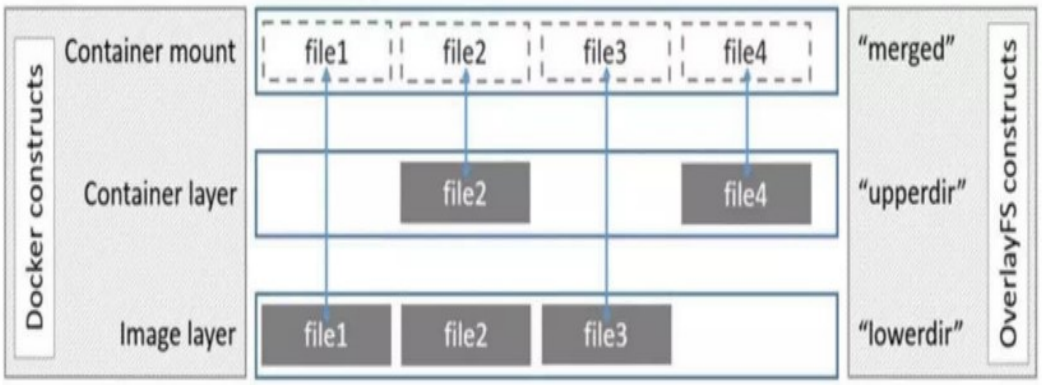
当需要修改一个文件时，使用 Copy On Write（写时复制）将文件从只读的 Lowerdir 复制到可写的 Upperdir 进行修改，结果也保存在 Upperdir 层。在 Docker 中，底下的只读层就是 image，可写层就是 Container。

如果镜像层和容器层可以有相同的文件，这中情况下，Upperdir 中的文件覆盖 Lowerdir 中的文件。Docker 镜像中的每一层并不对应 OverlayFS 中的层，而是镜像中的每一层对于应/var/lib/docker/overlay 中的一个文件夹，文件夹以该层的 UUID 命名。然后使用硬链接将下层的文件引用到上层。在一定程度上可以节省磁盘空间。



下图展示了容器和镜像的层与 OverlayFS 的 upperdir，lowerdir 以及 merged 之间的对应

关系：



27.10 Docker 虚拟化特点

跟传统 VM 比较具有如下优点：

- 操作启动快

运行时的性能可以获取极大提升，管理操作（启动，停止，开始，重启等等） 都是以秒或毫秒为单位的。

- 轻量级虚拟化

你会拥有足够的“操作系统”，仅需添加或减小镜像即可。在一台服务器上可以布署 100~1000 个 Containers 容器。但是传统虚拟化，你虚拟 10-20 个虚拟机就不错了。

- 开源免费

开源的，免费的，低成本的。由现代 Linux 内核支持并驱动。注* 轻量的 Container 必定可以在一个物理机上开启更多“容器”，注定比 VMs 要便宜。

- 前景及云支持

正在越来越受欢迎，包括各大主流公司都在推动 docker 的快速发展，性能有很大的优势。

- 跟传统 VM 比较具有如下缺点：

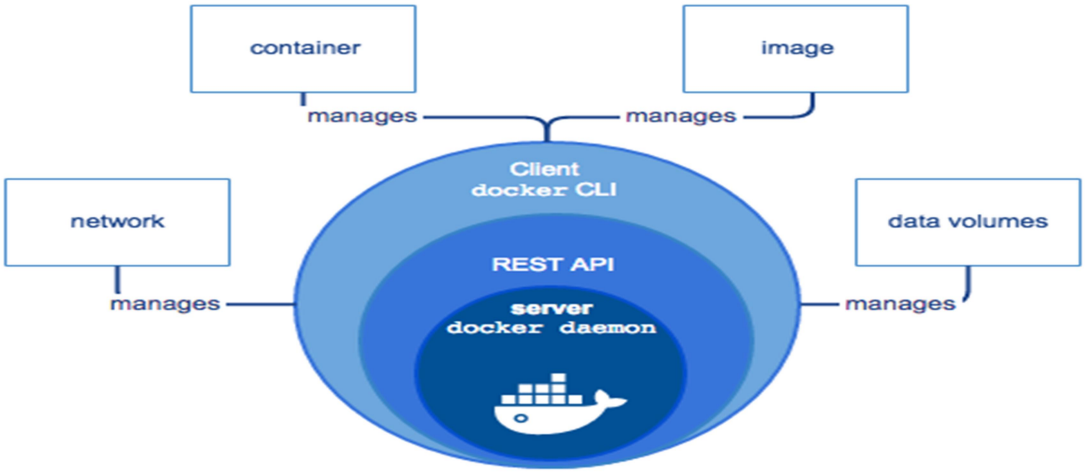
目前知道的人比较少；

相关的中文技术资料欠缺；

Go 语言还未完全成熟。

27.11 Docker 引擎架构

Docker 引擎是一个 C/S 结构的应用，组件如图所示：



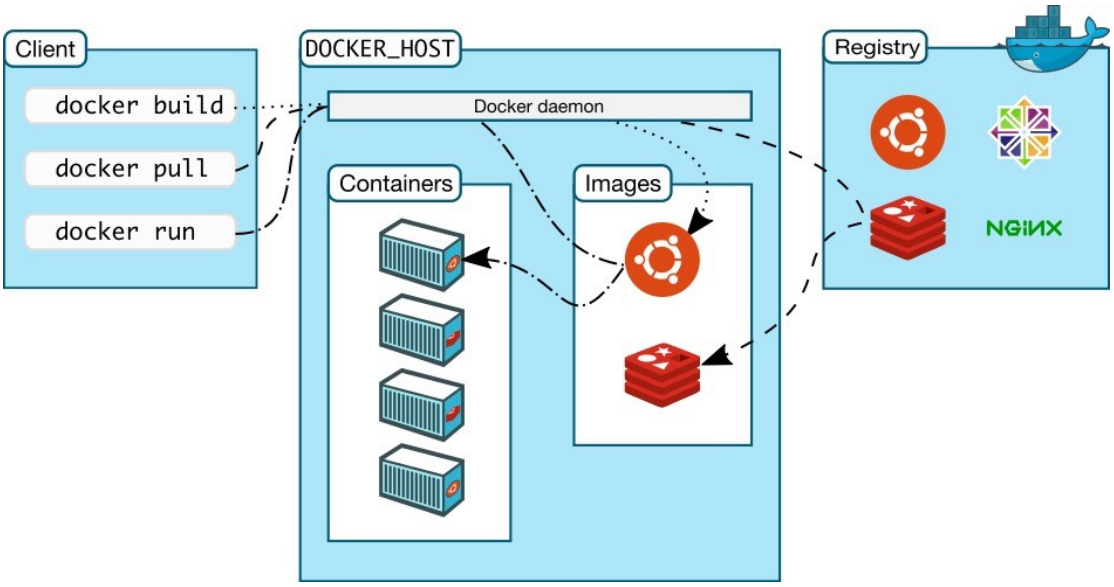
Server 是一个常驻进程；

REST API 实现了 client 和 server 间的交互协议；

CLI 实现容器和镜像的管理，为用户提供统一的操作界面。

Docker 使用 C/S 架构，Client 通过接口与 Server 进程通信实现容器的构建，运行和发布。

client 和 server 可以运行在同一台集群，也可以通过跨主机实现远程通信，架构如图所示：



27.12 Docker 镜像&容器&仓库

熟悉了 Docker 虚拟化简介、组件和工作原理之后，还需要掌握 Docker 虚拟化镜像原理&引擎架构等知识。

Docker 虚拟化技术有三个基础概念：Docker 镜像、Docker 容器、Docker 仓库，三个概念详解如下：

1）Docker 镜像

Docker 虚拟化最基础的组件为镜像，镜像跟我们常见的 Linux ISO 镜像类似，但是 Docker 镜像是分层结构的，是由多个层级组成，每个层级分别存储各种软件实现某个功能，Docker 镜像是静止的、只读的，不能对镜像进行写操作。

2）Docker 容器

Docker 容器是 Docker 虚拟化的产物，也是最早在生产环境使用的对象，Docker 容器的底层是 Docker 镜像，是基于镜像运行，并且在镜像最上层添加一层容器层

之后的实体，容器层是可读、可写的，容器层如果需用到镜像层中的数据，可以通过 JSON 文件读取镜像层中的软件和数据，对整个容器进行修改，只能作用于容器层，不能直接对镜像层进行写操作。

3）Docker 仓库

Docker 仓库是用于存放 Docker 镜像的地方，Docker 仓库分为两类，分别是：公共仓库（Public）和私有仓库（Private），国内和国外有很多默认的公共仓库，对外开放、免费或者付费使用，企业测试环境和生产环境推荐自建私有仓库，私有仓库的特点：安全、可靠、稳定、高效，能够更加自身的业务体系进行灵活升级和管理。

27.13 Docker 镜像原理剖析

完整的 Docker 镜像可以支撑一个 Docker 容器的运行，在 Docker 容器运行过程中主要提供文件系统数据支撑。Docker 镜像是分层结构的，是由多个层级组成，每个层级分别存储各种软件实现某个功能，Docker 镜像作为 Docker 中最基本的概念，有以下几个特性：

1）镜像是分层的，每个镜像都由一个或多个镜像层组成；

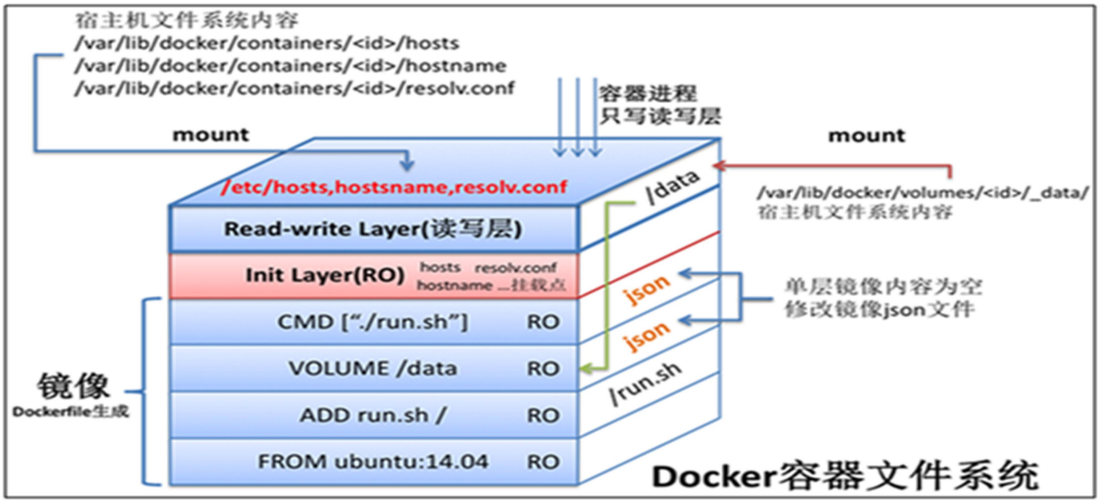
2）可通过在某个镜像加上一定的镜像层得到新镜像；

3）通过编写 Dockerfile 或基于容器 Commit 实现镜像制作；

4）每个镜像层拥有唯一镜像 ID，Docker 引擎默认通过镜像 ID 来识别镜像；

5）镜像在存储和使用时，共享相同的镜像层，在 PULL 镜像时，已有的镜像层会自动跳过下载；

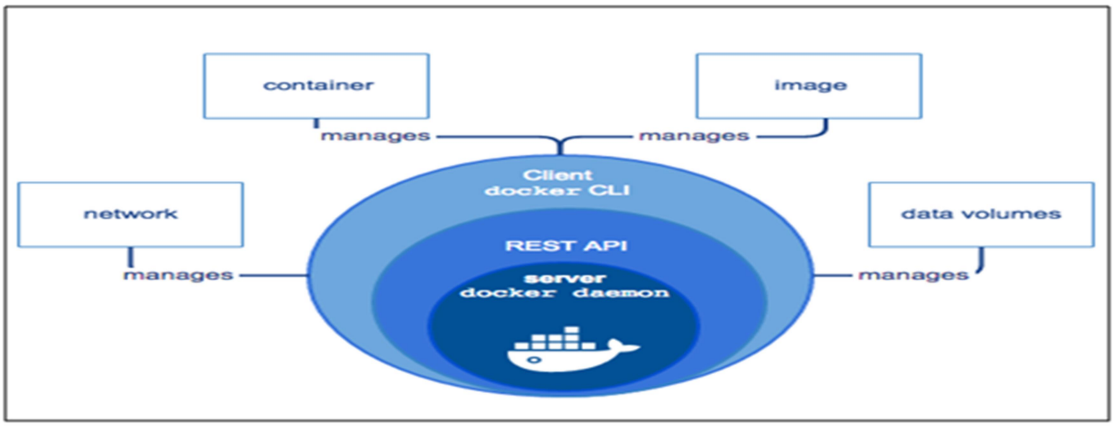
6）每个镜像层都是只读，即使启动成容器，也无法对其真正的修改，修改只会作用于最上层的容器层。



如图所示为一个完整的 Docker 容器系统，可以看出：

Docker 容器是一个或多个运行进程，而这些运行进程将占有相应的内存，相应的 CPU 计算资源，相应的虚拟网络设备以及相应的文件系统资源。Docker 容器所占用的文件系统资源，则通过 Docker 镜像的镜像层文件来提供。基于每个镜像的 Json 文件，可以通过解析 Docker 镜像的 json 的文件，获知应该在这个镜像之上运行什么样的进程，应该为进程配置什么样的环境变量，而 Docker 守护进程实现了从静态向动态的转变。

Docker 虚拟化引也是一个 C/S（Client/Server）结构的应用，如图所示：



该图为 Docker 虚拟化完整体系结构图，包括如下各个组件：

1）Docker Server 是一个常驻进程；

2）REST API 实现了 client 和 server 间的交互协议；

3）Docker CLI 实现容器和镜像的管理，为用户提供统一的操作界面；

4）Images 为容器提供了统一的软件、文件底层存储；

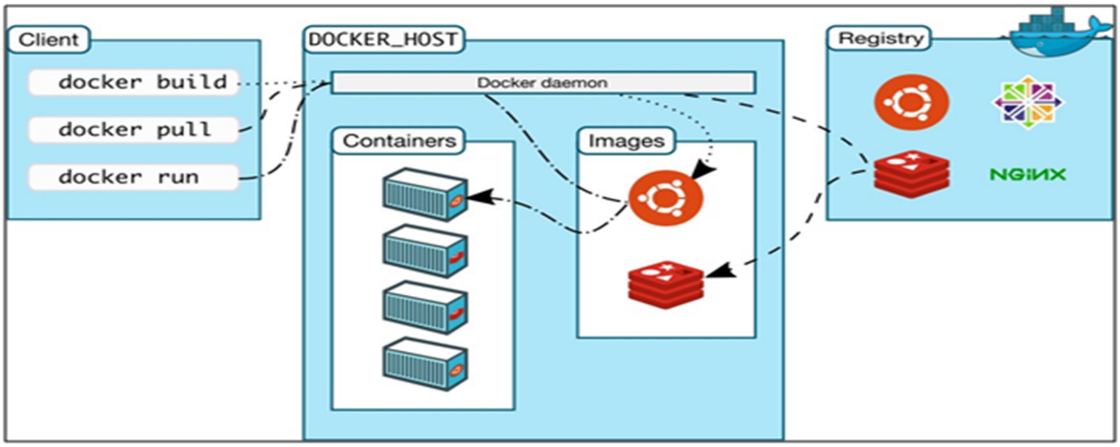
5）Container 是 Docker 虚拟化的产物，直接作为生产使用；

6）Network 为 Docker 容器提供完整网络通信；

7）Volume 为 Docker 容器提供额外磁盘、文件存储对象。

Docker 使用 C/S 架构，Client 通过接口与 Server 进程通信实现容器的构建，运行和发布。

client 和 server 可以运行在同一台集群，也可以通过跨主机实现远程通信，架构如图所示：



如上图可以清晰的看出 Docker 虚拟化整个生态体系：

1）基于 Docker Client 客户端-Rest API-操作 Docker Daemon；

2）Docker daemon 部署至 Docker 宿主机（通常为硬件物理机）；

3）基于 Docker pull 可以从 Registry 仓库获取各种镜像至 Docker Host 主机；

4）基于 Docker run 可以通过获取的镜像启动 Docker Container（容器）；

5）基于 Docker build 可以构建满足企业需求的各种 Docker images（镜像）；

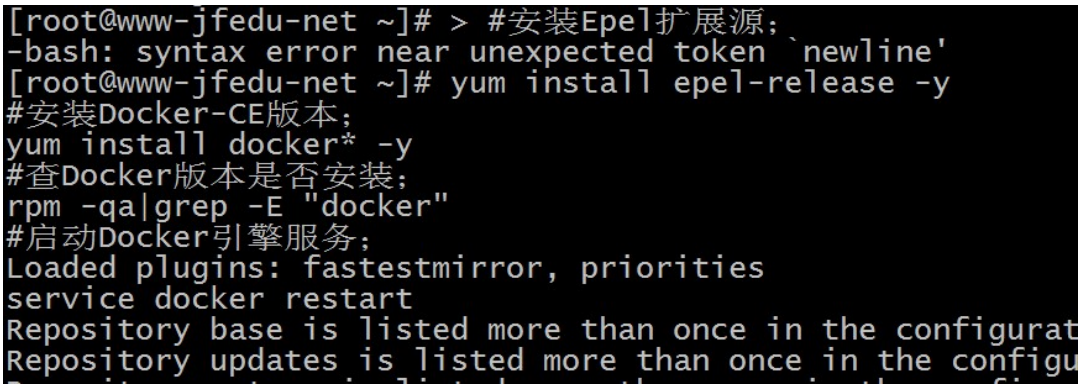
27.14 CentOS7 构建 Docker 平台实战

1）安装步骤和命令如下：

```
#安装国内阿里源；

wget  -P /etc/yum.repos.d/

http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo

#安装 Docker-CE 版本；

yum install docker-ce* -y

#查 Docker 版本是否安装；
```

```
rpm -qa|grep -E "docker"

#启动 Docker 引擎服务;

service docker restart

systemctl restart docker.service

#查看 Docker 服务进程;

ps -ef|grep docker
```

2）安装完成，如图所示：



3）查看启动进程如图所示：

4）查看 docker info 信息，如图所示：

```
[root@www-jfedu-net ~]# docker info|more
  WARNING: You're not using the default seccomp profile
Containers: 0
 Running: 0
 Paused: 0
 Stopped: 0
Images: 0
Server Version: 1.13.1
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
```

5）从 Docker 仓库下载 Nginx 镜像：

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# docker pull docker.io/nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
a5a6f2f73cd8: Extracting 6.652 MB/22.49 MB
a5a6f2f73cd8: Pull complete
67da5fbcb7a0: Pull complete
e82455fa5628: Pull complete
Digest: sha256:31b8e90a349d1fce7621f5a5a08e4fc519b634f7
Status: Downloaded newer image for docker.io/nginx:late
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
```

第28章　Docker 虚拟化实战高级篇

28.1　Docker 仓库源更新实战

Docker 默认连接的国外官方镜像，通常根据网络情况不同，访问时快时慢，大多时候获取速度非常慢，为了提示效率可以自建仓库或者先修改为国内仓库源，提升拉取镜像的速度。

Docker 可以配置的国内镜像有很多可供选择，例如：Docker 中国区官方镜像、阿里云、网易蜂巢、DaoCloud 等，这些都是国内比较快的镜像仓库。

从国外官网下载 Docker Tomcat 镜像，访问速度慢，如图所示：

```
[root@www-jfedu-net ~]# docker pull tomcat
Using default tag: latest
Trying to pull repository docker.io/library/tomcat ...
latest: Pulling from docker.io/library/tomcat
cc1a78bfd46b: Downloading 457.9 kB/45.32 MB
6861473222a6: Pulling fs layer
7e0b9c3b5ae0: Downloading 1.747 MB/4.336 MB
ae14ee39877a: Waiting
8085c1b536f0: Waiting
6e1431e84c0c: Waiting
ca0e3df5a1fd: Waiting
d2cb611ced6c: Waiting
268dc3e43e66: Waiting
79a7e8d254c7: Waiting
5c848af92738: Waiting
789b92e37607: Waiting
```

Docker 镜像修改方法，vim /etc/docker/daemon.json，执行如下命令即可：

```
cat>/etc/docker/daemon.json<<EOF

{

"registry-mirrors":["https://registry.docker-cn.com"]

}

EOF

service docker restart
```

重启 Docker 服务即可，修改仓库地址为国内仓库后，获取镜像速度非常快，如图所示：

```
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# service docker restart
Redirecting to /bin/systemctl restart docker.service

[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# systemctl restart docker.service
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# docker pull tomcat
Using default tag: latest
Trying to pull repository docker.io/library/tomcat ...
latest: Pulling from docker.io/library/tomcat
cc1a78bfd46b: Downloading  37.2 MB/45.32 MB
6861473222a6: Downloading 7.439 MB/10.77 MB
7e0b9c3b5ae0: Download complete
ae14ee39877a: Download complete
8085c1b536f0: Download complete
6e1431e84c0c: Download complete
ca0e3df5a1fd: Downloading  10.5 MB/122.1 MB
d2cb611ced6c: Waiting
```

28.2 Docker 典型命令演练

Docker 虚拟化平台部署完成，默认没有图形界面管理，作为运维人员、测试人员、开发人员来讲，需要通过 Docker-Client 命令行操作，如下为 Docker 平台下 30+操作指令，熟练指令的操作能够帮助我们对 Docker 进行高效的管理和维护，从而提高自己的技能。

28.3 Docker 虚拟化 30+命令实战剖析

docker search	在 docker hub 中搜索镜像;
docker pull	从 docker 镜像源服务器拉取指定镜像或者库镜像;
docker push	推送指定镜像或者库镜像至 docker 源服务器;
docker history	展示一个镜像形成历史;
docker images	列出系统当前镜像;
docker run	创建一个新的容器并运行一个命令;
docker start	启动容器;
docker stop	停止容器;
docker attach	当前 shell 下 attach 连接指定运行镜像;

docker build通过 Dockerfile 定制镜像;

docker commit 提交当前容器为新的镜像;

docker cp 从容器中拷贝指定文件或者目录到宿主机中;

docker create 创建一个新的容器，同 run，但不启动容器;

docker diff 查看 docker 容器变化;

docker events 从 docker 服务获取容器实时事件;

docker exec 在已存在的容器上运行命令;

docker export 导出容器的内容流作为一个 tar 归档文件[对应 import];

docker import 从 tar 包中的内容创建一个新的文件系统映像[对应 export];

docker info 显示系统相关信息;

docker inspect 查看容器详细信息;

docker kill 指定 docker 容器;

docker load 从一个 tar 包中加载一个镜像[对应 save];

docker login注册或者登陆一个 docker 源服务器;

docker logout Dockerregistry 退出;

docker logs 输出当前容器日志信息;

docker port 查看映射端口对应的容器内部源端口;

docker pause 暂停容器;

docker ps 列出容器列表;

docker restart 重启运行的容器;

docker rm 移除一个或者多个容器;

docker rmi 移除一个或多个镜像;

docker save 保存一个镜像为一个 tar 包[对应 load];
docker tag 给源中镜像打标签;
docker top 查看容器中运行的进程信息;
docker unpause 取消暂停容器;
docker version 查看 docker 版本号;
docker wait 截取容器停止时的退出状态值;

命令列表如下：

docker	search	在 docker hub 中搜索镜像;
docker	pull	从 docker 镜像源服务器拉取指定镜像或者库镜像;
docker	push	推送指定镜像或者库镜像至 docker 源服务器;
docker	history	展示一个镜像形成历史;
docker	images	列出系统当前镜像;
docker	run	创建一个新的容器并运行一个命令;
docker	start	启动容器;
docker	stop	停止容器;
docker	attach	当前 shell 下 attach 连接指定运行镜像;
docker	build	通过 Dockerfile 定制镜像;
docker	commit	提交当前容器为新的镜像;
docker	cp	从容器中拷贝指定文件或者目录到宿主机中;
docker	create	创建一个新的容器，同 run，但不启动容器;
docker	diff	查看 docker 容器变化;

docker	events	从 docker 服务获取容器实时事件;
docker	exec	在已存在的容器上运行命令;
docker	export	导出容器的内容流作为一个 tar 归档文件[对应 import];
docker	import	从 tar 包中的内容创建一个新的文件系统映像[对应 export];
docker	info	显示系统相关信息;
docker	inspect	查看容器详细信息;
docker	kill	指定 docker 容器;
docker	load	从一个 tar 包中加载一个镜像[对应 save];
docker	login	注册或者登陆一个 docker 源服务器;
docker	logout	Docker registry 退出;
docker	logs	输出当前容器日志信息;
docker	port	查看映射端口对应的容器内部源端口;
docker	pause	暂停容器;
docker	ps	列出容器列表;
docker	restart	重启运行的容器;
docker	rm	移除一个或者多个容器;
docker	rmi	移除一个或多个镜像;
docker	save	保存一个镜像为一个 tar 包[对应 load];
docker	tag	给源中镜像打标签;
docker	top	查看容器中运行的进程信息;

docker	unpause	取消暂停容器;
docker	version	查看 docker 版本号;
docker	wait	截取容器停止时的退出状态值。

28.4 Docker 网络深入剖析

Docker 虚拟化技术底层是基于 LXC+Cgroups+AUFS（Overlay）技术实现，而我们有熟知 Cgroups 是 Linux 内核提供的一种可以限制、记录、隔离进程组（Process Groups）所使用的物理资源的机制。

Docker 虚拟化的产物是 Docker 容器，基于 Docker Engine 启动容器时，默认会给容器指定和分配各种子系统：CPU 子系统、Memory 子系统、IO 子系统、NET 子系统等。

启动一个容器，会分配 Network Namespace（子系统）提供了一份独立的网络环境，包括网卡、路由、Iptables 规则等，容器跟其他容器的 Network Namespace 是相互隔离的。

通过 Docker run 创建 Docker 容器时，可以使用--net 选项指定 Docker 容器的网络模式，Docker 默认有四种网络模式：

- host 模式，使用--net=host 指定；
- container 模式，使用--net=container:NAME_or_ID 指定；
- none 模式，使用--net=none 指定；
- bridge 模式，使用--net=bridge 指定，默认设置。

28.5 Host 模式剖析

通常来讲，启动新的 Docker 容器，都会分配独立的 Network Namespace 隔离子系统，如果在运行是指定为 host 模式，那么 Docker 容器将不会获得一个独立的 Network Namespace，而是和宿主机共用一个 Network Namespace 子系统。

新创建的 Docker 容器不会创建自己的网卡，不会再虚拟出自己的网卡、IP、网关、路由等信息，而是和宿主机共享 IP 和端口等信息，其他的软件、目录还是相互独立的。两个容器除了网络方面相同之外，其他的如文件系统、进程列表等还是相互隔离的。

28.6 Container 模式剖析

Docker 容器网络，Container 模式是指定新创建的容器和已存在的某个容器共享一个 Network Namespace 子系统，而不是和宿主机共享 Namespace 子系统。

新创建的 Docker 容器不会创建自己的网卡，不会再虚拟出自己的网卡、IP、网关、路由等信息，而是和指定的 Docker 容器共享 IP 和端口等信息，其他的软件、目录还是相互独立的。两个容器除了网络方面相同之外，其他的如文件系统、进程列表等还是相互隔离的。如果依附的 Docker 容器关闭，新的 Docker 容器网络也会丢失。

28.7 None 模式剖析

None 模式与其他的模式都不同，如果 Docker 容器使用 None 模式，Docker 容器会拥有自己的 Network Namespace 子系统，但是 Docker 引擎并不会为新启动的 Docker 容器配置任何的网络信息。

即新创建的 Docker 容器不会虚拟出自己的网卡、IP、网关、路由等信息，而是需要手工为 Docker 容器添加网卡、配置 IP、路由等信息，在企业实战环境中，通常会使用 Pipework 工具为 Docker 容器指定 IP 等信息。

28.8 Bridge 桥接剖析

Docker 容器的 Bridge 模式也是 Docker 默认的网络模式，该模式会为每个容器分配 Network Namespace 子系统，会自动给每个容器虚拟出自己的网卡、IP、网关、路由等信息，无需手工添加。

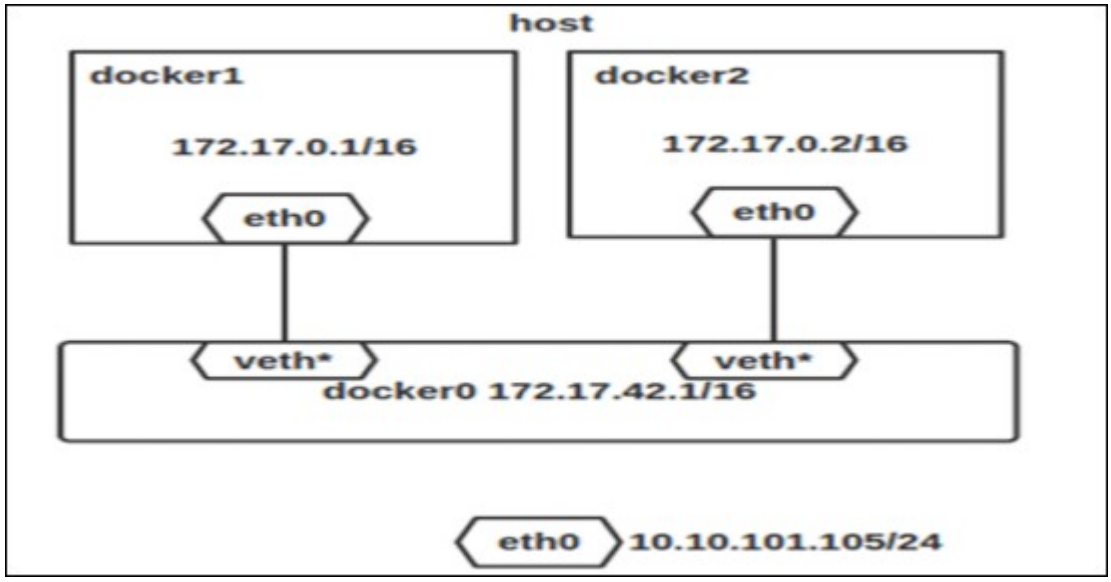
默认创建的 Docker 容器会统一通过一对 veth 虚拟网卡，连接到一个虚拟网桥交换机 Docker0 上，所有的容器的网络加入到一个二层交换机网络里面，即同一宿主机的所有容器之间都是可以相互联通和访问的。

28.9 Bridge 模式原理剖析

默认 Docker 引擎启动会在本地生成一个 Docker0 虚拟网卡。Docker0 是一个标准 Linux 虚拟网桥设备。在 Docker 默认的桥接网络工作模式中，docker0 网桥起到了至关重要的作用。物理网桥是标准的二层网络设备，标准物理网桥只有两个网口，可以将两个物理网络连接在一起。

但与物理层设备集线器等相比，网桥具备隔离冲突域的功能。网桥通过 MAC 地址学习和泛洪的方式实现二层相对高效的通信。随着技术的发展，标准网桥设备已经基本被淘汰了，替代网桥的是二层交换机。二层交换机也可以看成一个多口网桥。

如图为 Docker 容器采用 Bridge 模式结构图：



Docker Bridge 桥接模式创建过程：

1)启动一个 Docker 容器，指定模式为桥接模式时，Docker 引擎会创建一对虚拟网卡 veth pair 设备，veth 设备总是成对出现的，组成了一个数据的通道，数据从一个设备进入，就会从另一个设备出来，veth 设备常用来连接两个网络设备，可以把 veth 接口对认为是虚拟网线的两端。这个虚拟网线一端插在名为 docker0 的网桥上，另一端插到容器里。通过把每个 veth 接口绑定到 docker0 网桥，Docker 创建了一个虚拟子网，这个子网由宿主机和所有的 Docker 容器共享。

2)Docker 将 veth pair 设备的一块设备放在新创建的容器中，命名为 eth0，然后将另外一块设备放在宿主机中，以 vethxxx 类似的名称命名，并将这个网络设备加入到 docker0 网桥中。

3)Docker 引擎会从 docker0 子网中动态分配一个新的 IP 给容器使用，并

设置 docker0 的 IP 地址为容器的默认网关。

4）此时新创建的容器与宿主机能够通信，宿主机也可以访问容器中的 IP 地址，在 Bridge 模式下，连在同一网桥（交换机）上的容器之间可以相互通信，同时容器也可以访问外网（基于 iptables SNAT），但是其他物理机不能访问 docker 容器 IP，需要通过 NAT 将容器 IP 的 port 映射为宿主机的 IP 和 port。

28.10 Bridge 模式实战一

基于 Docker 引擎启动 Nginx WEB 容器，默认以 Bridge 方式启动 Docker 容器，会动态 DHCP 给 Docker 容器分配 IP、网关等信息，操作指令如下：

#查看镜像列表;

docker images

#运行新的 Nginx 容器;

docker run -itd docker.io/nginx:latest

#查看启动的 nginx 容器

docker ps

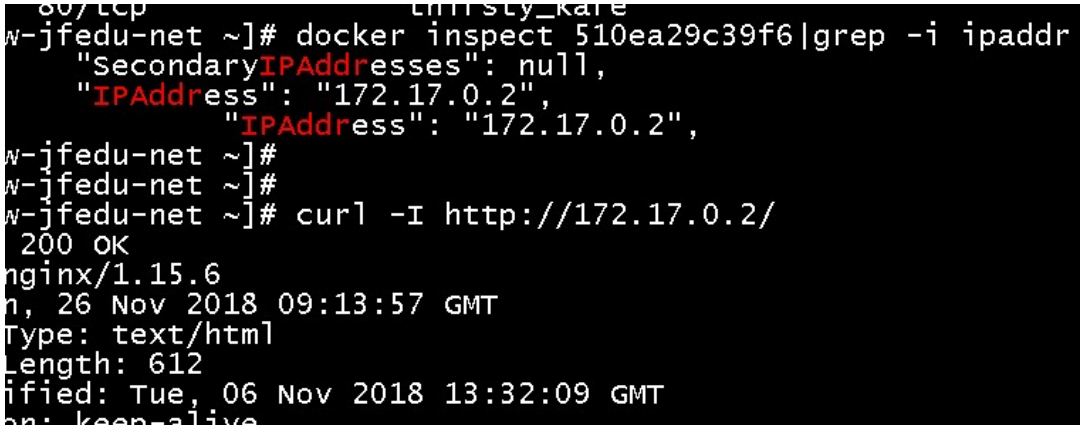
```
 ~]#
 ~]# docker images
             TAG                 IMAGE ID            CRE
             latest              6759d91a032b        2 w
             latest              62f816a209e6        2 w
centos6-ssh  latest              efd998bd6817        4 y
 ~]# docker run -itd docker.io/nginx:latest
ef86e2b4aa908aa68560bfc75b908c9d814bf9026221a
 ~]#
 ~]# docker ps
 IMAGE                  COMMAND                  CREATE
          NAMES
 docker.io/nginx:latest   "nginx -g 'daemon ..."   10 sec
          thirsty_kare
```

#查看 nginx 容器的 IP 地址；

docker inspect 510ea29c39f6|grep -i ipaddr

访问 nginx 容器 80 端口服务；

curl -I http://172.17.0.2/



28.11 Bridge 模式实战二

基于 Docker 引擎启动 Nginx WEB 容器，默认以 Bridge 方式启动 Docker 容器，此处使用 pipework 工具手工给容器指定桥接网卡，并且手工配置 IP 地址，操作指令如下：

#查看镜像列表；

docker images

#运行新的 Nginx 容器；

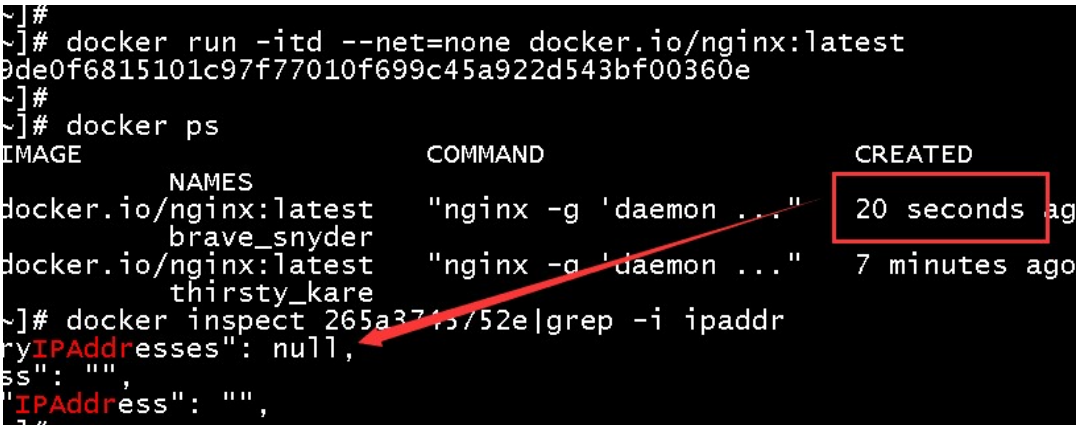
docker run -itd --net=none docker.io/nginx:latest

#查看启动的 nginx 容器

docker ps

#查看 nginx 容器的 IP 地址；（没有 IP 地址）

docker inspect 265a3745752e|grep -i ipaddr



安装 pipework IP 配置脚本工具，方法如下：

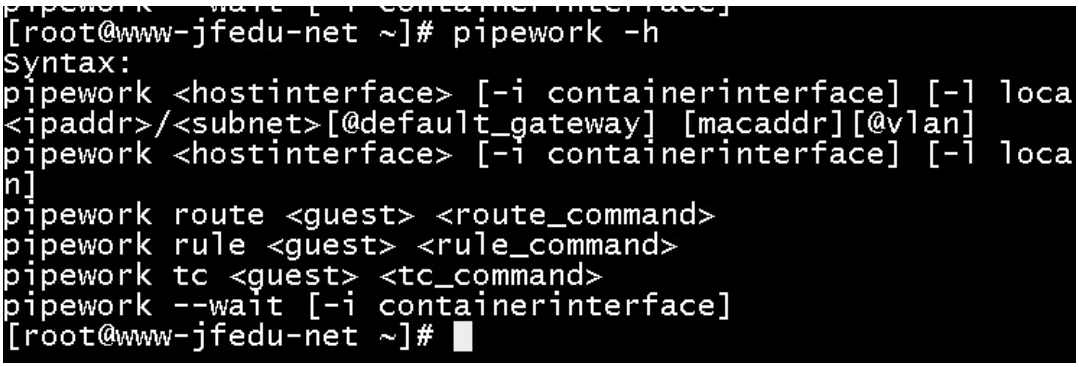
安装 pipework

git clone https://github.com/jpetazzo/pipework

cp ~/pipework/pipework /usr/local/bin/

查看 pipework 工具是否配置正确；

pipework -h



基于 pipework 工具手工指定容器的 IP，并且设置容器为桥接方式上网，命令如下：

（docker0 为网桥名称，172.17.0.18/16 为容器 IP 和掩码，172.17.0.1 为容器网关）

pipework docker0 265a3745752e 172.17.0.18/16@172.17.0.1

ping 172.17.0.18 -c 2

curl -I http://172.17.0.18/

```
[root@www-jfedu-net ~]# ping  172.17.0.18 -c 2
PING 172.17.0.18 (172.17.0.18) 56(84) bytes of data.
64 bytes from 172.17.0.18: icmp_seq=1 ttl=64 time=0.041 ms
64 bytes from 172.17.0.18: icmp_seq=2 ttl=64 time=0.057 ms

--- 172.17.0.18 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 99
rtt min/avg/max/mdev = 0.041/0.049/0.057/0.008 ms
[root@www-jfedu-net ~]# curl -I  http://172.17.0.18/
HTTP/1.1 200 OK
Server: nginx/1.15.6
Date: Mon, 26 Nov 2018 09:26:46 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 06 Nov 2018 13:32:09 GMT
```

28.12 Bridge 模式实战三

基于 Docker 引擎启动 Nginx WEB 容器，默认以 Bridge 方式启动 Docker 容器，而且 Docker0 的网桥 IP 为 172.17.0.0/16 网段，可以通过指令修改 Docker 网桥的 IP 网段，例如将网桥 IP 段修改为 10.10.0.1/16 段，操作指令如下：

#删除原有网络信息；

service docker stop

ip link set dev docker0 down

brctl delbr docker0

iptables -t nat -F POSTROUTING

#添加新的 docker0 网络信息；

brctl addbr docker0

ip addr add 10.10.0.1/16 dev docker0

ip link set dev docker0 up

#配置 Docker 的文件;

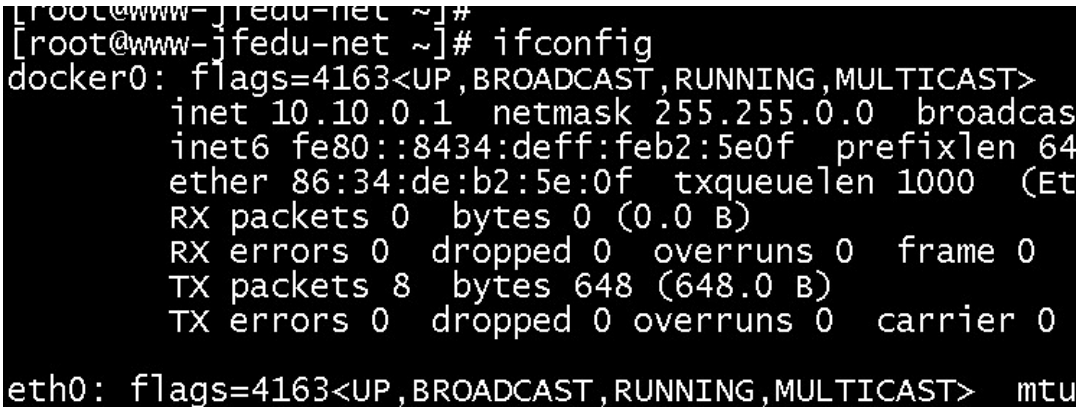
cat>/etc/docker/daemon.json<<EOF

{"registry-mirrors": ["http://docker-cn.docker.com"],

"bip": "10.10.0.1/16"

}

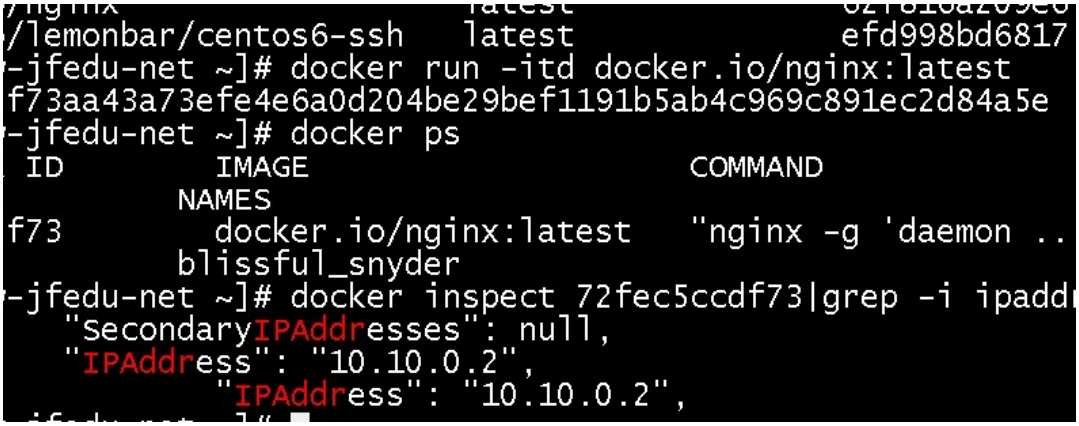
EOF



#启动新的 Docker 容器，查看容器桥接网络 IP 地址，如图所示:

docker run -itd docker.io/nginx:latest

docker inspect 72fec5ccdf73|grep -i ipaddr



28.13 Bridge 模式实战四

基于 Docker 引擎启动 Nginx WEB 容器，默认以 Bridge 方式启动 Docker 容器，而且 Docker0 的网桥 IP 为 172.17.0.0/16 网段，默认局域网的其他物理机是不能直接访问 Docker 容器的。

为了实现 Docker 容器跟局域网通信，并且实现局域网其他的物理机也可以访问容器的 IP（不配置 NAT 映射），可以自定义桥接网络 br0，将 br0 跟物理网卡 eth0 或者 ens33 桥接。

操作方法如下：

添加 ens33 网卡指定 bridge 桥接网卡名称 br0；

cd /etc/sysconfig/network-scripts/

#查看 ifcfg-ens33 网卡 IP 地址；

cat ifcfg-ens33

TYPE="Ethernet"

DEVICE="ens33"

ONBOOT="yes"

BRIDGE="br0"

IPADDR=192.168.0.141

NETMASK=255.255.255.0

GATEWAY=192.168.0.1

#查看 ifcfg-br0 网卡 IP 地址;

cat ifcfg-br0

DEVICE="br0"

BOOTPROTO=static

ONBOOT=yes

TYPE="Bridge"

IPADDR=192.168.0.141

NETMASK=255.255.255.0

GATEWAY=192.168.0.1

#重启 network 网络服务;

service network restart

```
[root@jfedu143 ~]# service network restart
Restarting network (via systemctl):
[root@jfedu143 ~]# ifconfig
br0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.143  netmask 255.255.255.0  broadcast
        inet6 fe80::20c:29ff:fe2d:ed7a  prefixlen 64  scopei
        ether 00:0c:29:2d:ed:7a  txqueuelen 1000  (Ethernet)
        RX packets 15  bytes 1104 (1.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17  bytes 1518 (1.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collis

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 00:0c:29:2d:ed:7a  txqueuelen 1000  (Ethernet)
```

#修改 docker 引擎，使其读取 br0 网桥；

cat /etc/sysconfig/docker-network

DOCKER_NETWORK_OPTIONS="-b=br0"

#启动 Docker 容器，设置为 none 模式，然后使用 br0 网桥，指令如下：

（br0 为网桥名称，192.168.0.11/24 为容器 IP 和掩码，192.168.0.141 为容器网关）

docker run -itd --net=none --name=nginx-v1 docker.io/nginx

pipework br0 nginx-v1 192.168.0.11/24@192.168.0.141

```
[root@jfedu141 ~]# ping 192.168.0.11 -c 2
PING 192.168.0.11 (192.168.0.11) 56(84) bytes of data.
64 bytes from 192.168.0.11: icmp_seq=1 ttl=64 time=0.19
64 bytes from 192.168.0.11: icmp_seq=2 ttl=64 time=0.05

--- 192.168.0.11 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time
rtt min/avg/max/mdev = 0.054/0.125/0.196/0.071 ms
[root@jfedu141 ~]#
[root@jfedu141 ~]# curl -I http://192.168.0.11/
HTTP/1.1 200 OK
Server: nginx/1.15.6
Date: Mon, 26 Nov 2018 10:46:37 GMT
```

28.14 CentOS6.x Docker 桥接网络实战

基于 CentOS6.x 构建 Docker 桥接网络，案例方法如下：

Docker0 桥接网络可以自定义，如下为自定义桥接网络的设置：

/etc/init.d/docker stop

关掉 docker0

ifconfig docker0 down

删除 docker

brctl delbr docker0

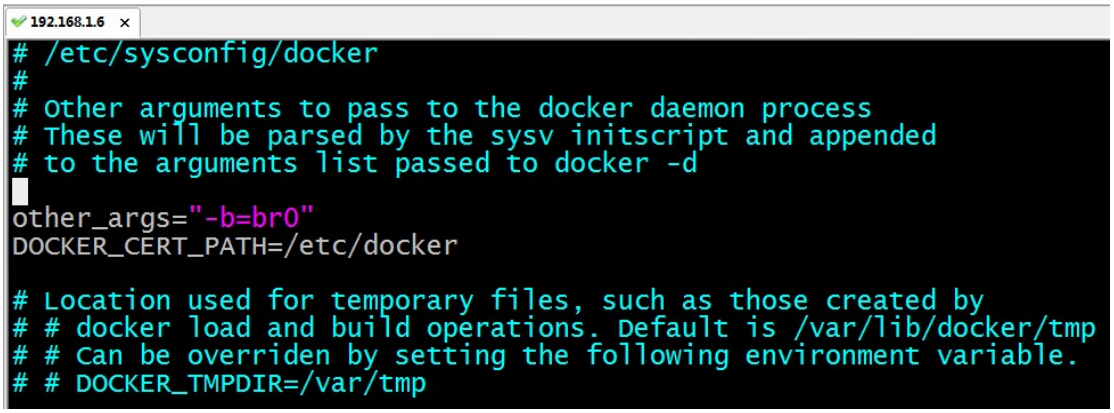
增加网桥 br0

yum install bridge-utils

CentOS6.5 下的配置：

vim /etc/sysconfig/docker

other_args="-b=br0"



除此之外，配置 bridge 桥接网络可以直接设置网卡配置文件：

/etc/sysconfig/network-scripts/下，修改 ifcfg-eth0 网卡配置，同时增加 ifcfg-br0 桥接网卡配置，操作流程如下：

vi ifcfg-eth0 内容修改为如下：

```
DEVICE=eth0

BOOTPROTO=none

NM_CONTROLLED=no

ONBOOT=yes

TYPE=Ethernet

BRIDGE="br0"

IPADDR=192.168.43.81

NETMASK=255.255.255.0

GATEWAY=192.168.43.1

USERCTL=no
```

vi　ifcfg-br0 内容如下：

```
DEVICE="br0"

BOOTPROTO=none

IPV6INIT=no

NM_CONTROLLED=no

ONBOOT=yes

TYPE="Bridge"

IPADDR=192.168.43.81

NETMASK=255.255.255.0

GATEWAY=192.168.43.1

USERCTL=no
```

```
[root@localhost network-scripts]# cat ifcfg-eth0
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
TYPE=Ethernet
BOOTPROTO=static
BRIDGE=br0
[root@localhost network-scripts]# cat ifcfg-br0
DEVICE="br0"
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=dhcp
TYPE=Bridge
[root@localhost network-scripts]#
```

启动 docker 服务

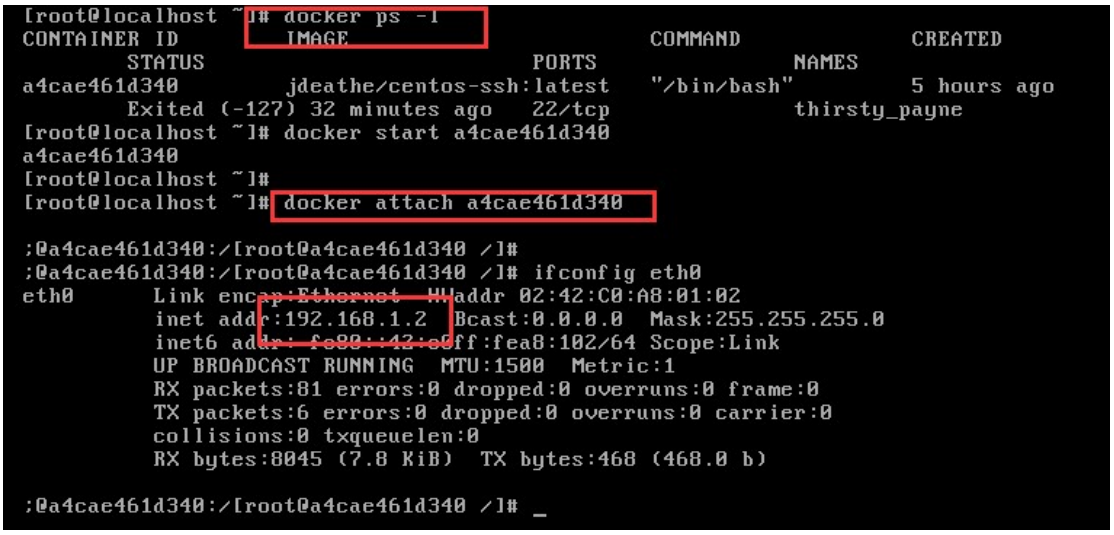
/etc/init.d/docker start

查看服务器网卡信息如下：

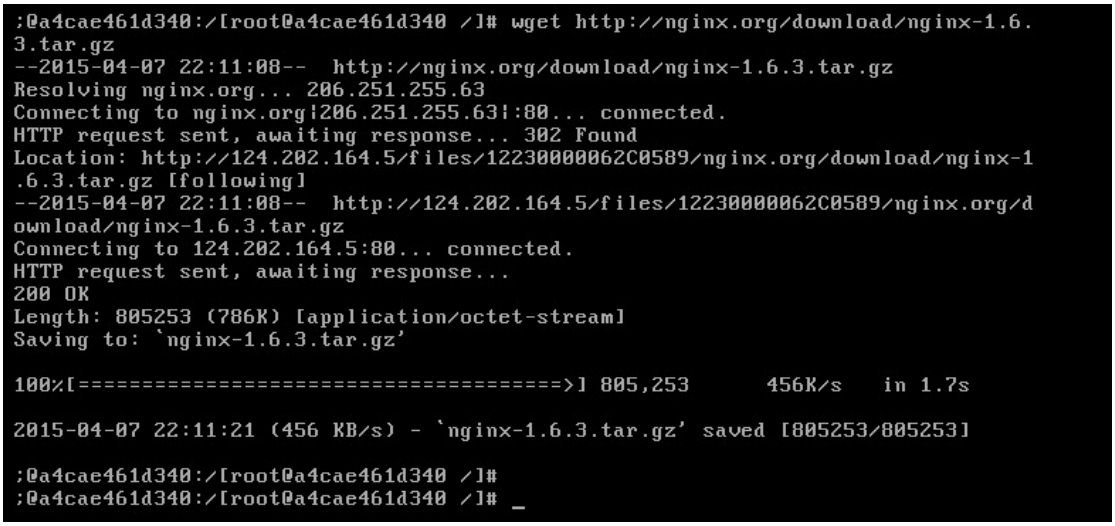
```
192.168.1.6 ×
br0       Link encap:Ethernet  HWaddr 00:0C:29:67:FA:D3
          inet addr:192.168.1.6  Bcast:192.168.1.255  Mask:255.255.
          inet6 addr: fe80::20c:29ff:fe67:fad3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:165 errors:0 dropped:0 overruns:0 frame:0
          TX packets:49 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:41513 (40.5 KiB)  TX bytes:6286 (6.1 KiB)

eth0      Link encap:Ethernet  HWaddr 00:0C:29:67:FA:D3
          inet6 addr: fe80::20c:29ff:fe67:fad3/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7943 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4158 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1168582 (1.1 MiB)  TX bytes:663488 (647.9 KiB)
          Interrupt:19 Base address:0x2000
```

启动一个新的 docker 客户端，查看客户端 ip 如下：

进入已经启动过的容器命令：docker attach　容器 ID 即可。



从外网下载 nginx 包：



28.15 CentOS7.x Docker 桥接网络实战

基于 CentOS7.x 构建 Docker 桥接网络，案例方法如下：

配置 bridge 桥接网络可以直接设置网卡配置文件：

/etc/sysconfig/network-scripts/下，修改 ifcfg-ens33 网卡配置，同时增加 ifcfg-br0 桥接网卡配置，操作流程如下：

vi ifcfg-ens33 内容修改为如下：

```
DEVICE=ens33
```

```
BOOTPROTO=none

NM_CONTROLLED=no

ONBOOT=yes

TYPE=Ethernet

BRIDGE="br0"

IPADDR=192.168.43.81

NETMASK=255.255.255.0

GATEWAY=192.168.43.1

USERCTL=no
```

vi　ifcfg-br0 内容如下：

```
DEVICE="br0"

BOOTPROTO=none

IPV6INIT=no

NM_CONTROLLED=no

ONBOOT=yes

TYPE="Bridge"

IPADDR=192.168.43.81

NETMASK=255.255.255.0

GATEWAY=192.168.43.1

USERCTL=no
```

启动 docker 服务，即可；

service docker start

Docker 默认提供了一个隔离的内网环境，启动时会建立一个 docker0 的虚拟网卡，每个容器都是连接到 docker0 网卡上的。而 docker0 的 ip 段为 172.17.0.1，如果想让容器与宿主机同一网段的其他机器访问，就必须在启动 docker 的时候将某个端口映射到宿主机的端口。

KVM 的桥接网络非常方便，其实 docker 也比较方便，至少不是自带的桥接而已，上次课程我们讲解了 docker 容器在 centos6.5 下的实现方法，今天我们来讲解 centos7 下如果快速实现 docker 容器桥接网络，并为容器分配外网 IP。如下为通过 pipework 工具配置容器 IP 方法：

安装 pipework

git clone https://github.com/jpetazzo/pipework

cp ~/pipework/pipework /usr/local/bin/

启动容器并设置网络

docker run -itd --net=none --name=lamp2 centos7 /bin/bash

pipework br0 lamp2 192.168.1.11/24@192.168.1.88

进入容器查看 ip

docker exe lamp2 ifconfig

```
bash-4.1# ifconfig
eth1      Link encap:Ethernet  HWaddr FA:B9:11:74:4D:07
          inet addr:192.168.1.11  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::f8b9:11ff:fe74:4d07/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1007 errors:0 dropped:0 overruns:0 frame:0
          TX packets:43 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:192809 (188.2 KiB)  TX bytes:3574 (3.4 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:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

28.16 基于 Docker WEB 管理 Docker 容器

通常来讲，开发人员和管理人员默认通过命令行来创建及运行 Docker 容器，但 Docker 的 Remote API 让他们可以通过充分利用 REST（代表性状态传输协议）的 API，运行相同的命令。

DockerUI 也是基于 API 方式管理宿主机的 Docker 引擎。Docker UI Web 前端程序让你可以处理通常通过 Web 浏览器的命令行来管理的许多任务。

主机上的所有容器都可以通过仅仅一条连接来处理，该项目几乎没有任何依赖关系。该软件目前仍在大力开发之中，但是它采用麻省理工学院（MIT）许可证，所以可以免费地重复使用。

Docker UI 不包含任何内置的身份验证或安全机制，所以务必将任何公之于众的 DockerUI 连接放在用密码来保护的系统后面。

1）下载 Docker UI 镜像；

只需要在宿主机 pull 相关的镜像即可，指令如下：

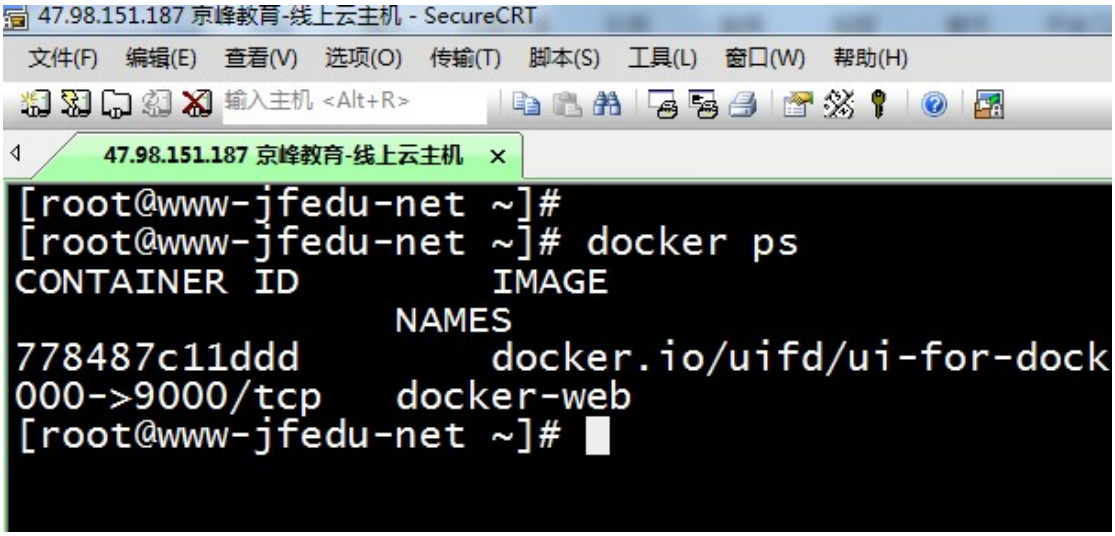
docker pull uifd/ui-for-docker

docker images

```
-net ~]#
-net ~]# docker pull uifd/ui-for-docker
ag: latest
repository docker.io/uifd/ui-for-docker ...
 from docker.io/uifd/ui-for-docker
fe371ff5a69549269b24073a5ab1244dd4c0b834cbadf24487057
s up to date for docker.io/uifd/ui-for-docker:latest
-net ~]# docker images
                TAG                 IMAGE ID
t               latest              690cb3b9c7d1
                latest              c82521676580
ui-for-docker   latest              965940f98fa5
-net ~]#
```

2）启动 docker-UI 服务，并且映射 9090 至容器 9090；

docker run -it -d --name docker-web -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock docker.io/uifd/ui-for-docker



3）如果启动 Docker 端口映射，报错信息如下：

docker0: iptables: No chain/target/match by that name.

解决方法：

如上报错信息是因为本地 iptables 规则策略没有匹配的链表，解决方案如下：

pkill docker

iptables -t nat -F

ifconfig docker0 down

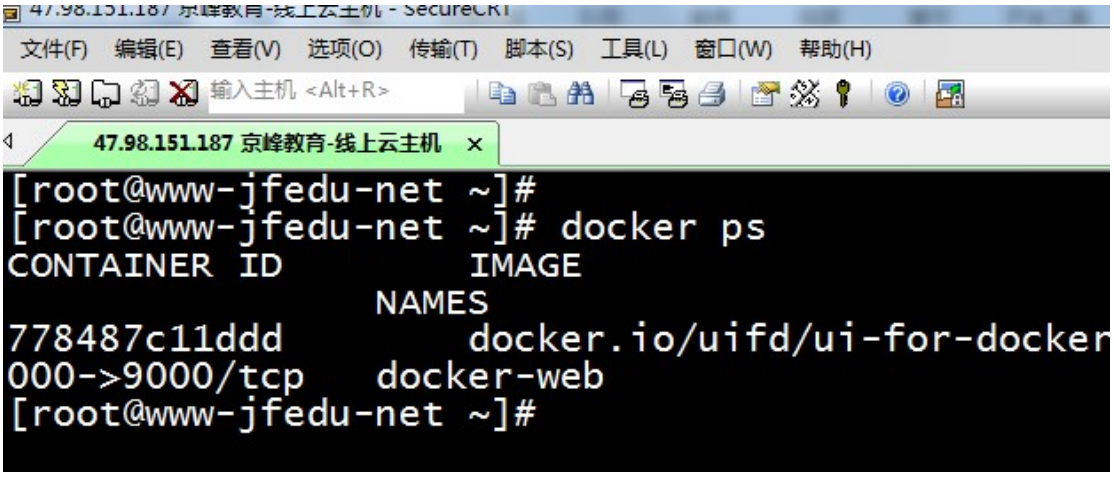
yum install -y bridge* -y

brctl delbr docker0

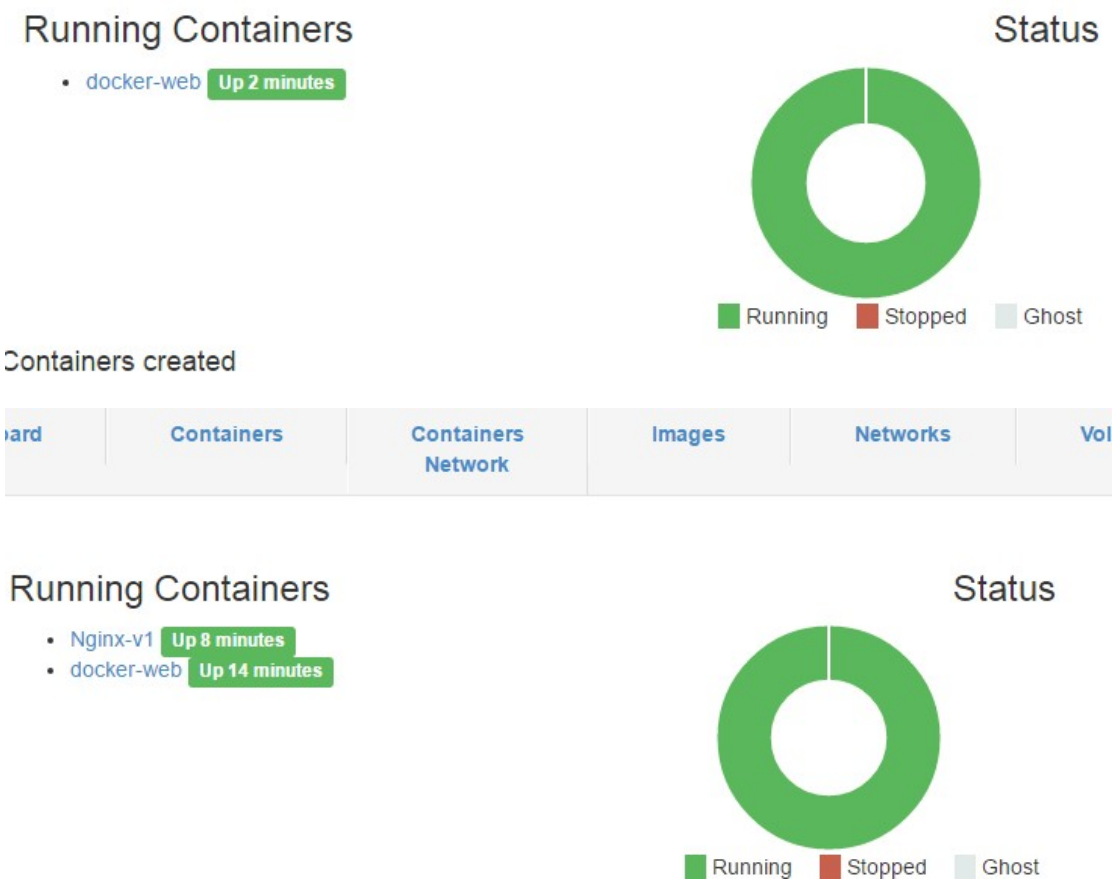
service docker restart

```
[root@www-jfedu-net ~]# docker ps -qa
e11075abd53a
[root@www-jfedu-net ~]# docker ps -aq
e11075abd53a
[root@www-jfedu-net ~]# docker ps -aq|xargs docker rm -
e11075abd53a
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]# docker run -it -d --name docker
cker.io/uifd/ui-for-docker
b510c42470c1120368100c80b5d5a375df74b8d9ad027305a5542b2
[root@www-jfedu-net ~]#
[root@www-jfedu-net ~]#
```

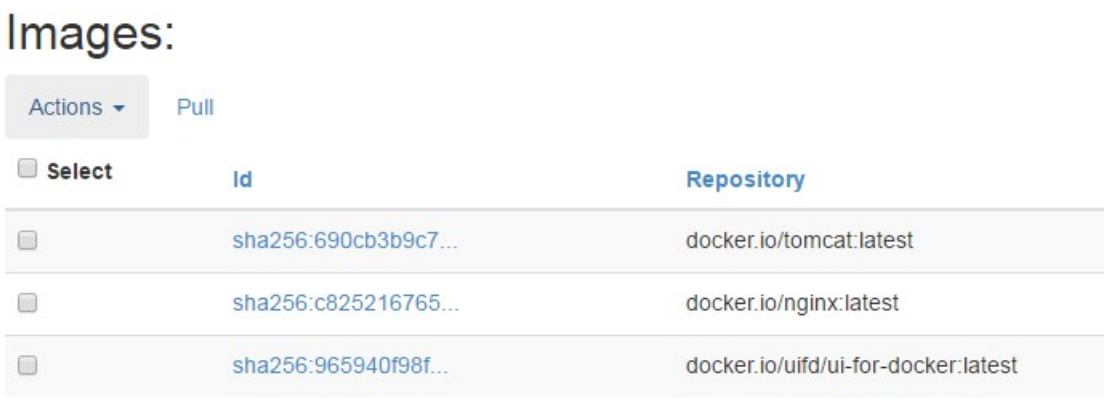
4）通过 docker ps 查看 Docker UI 状态；



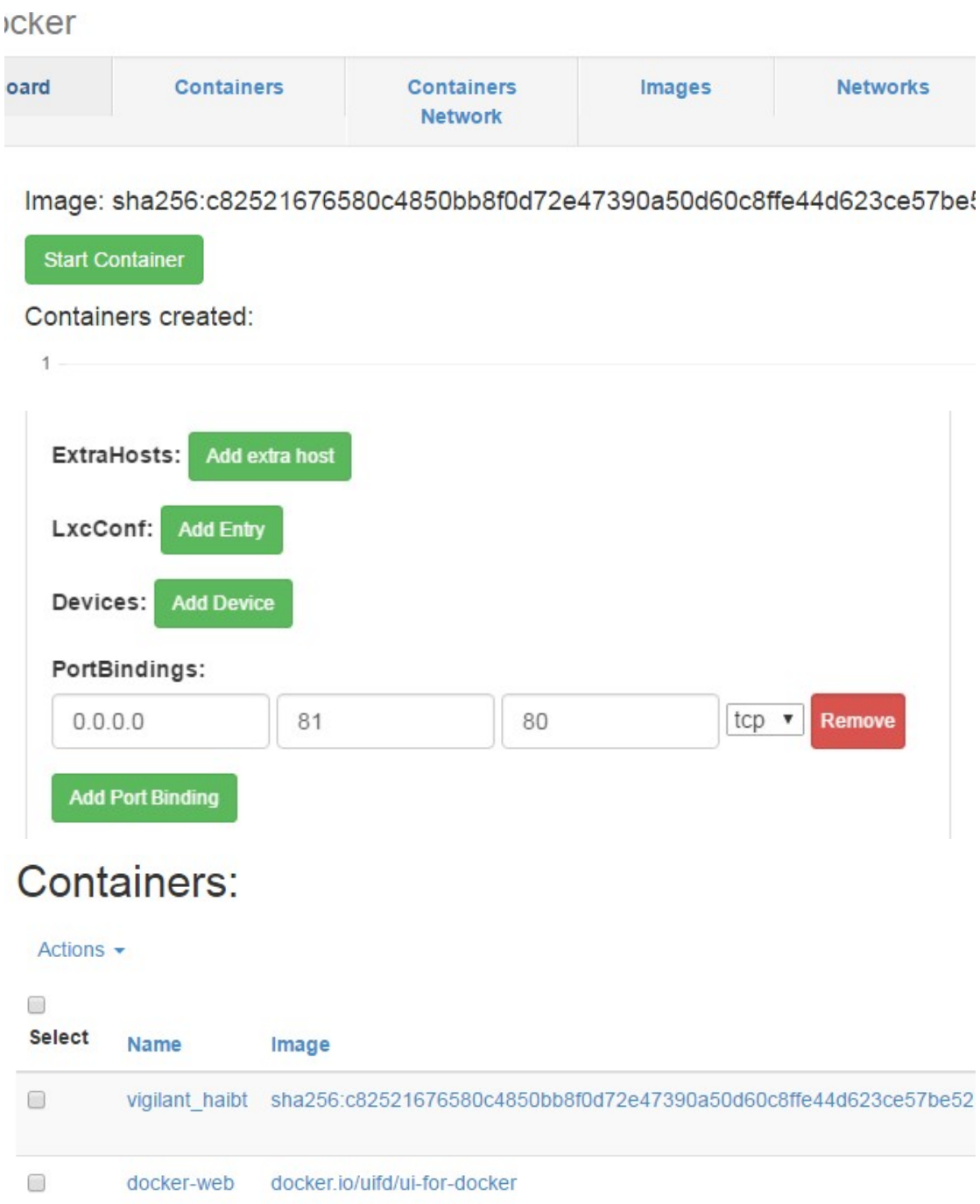
5）通过浏览器登录 WEB 9000 端口访问如图所示：



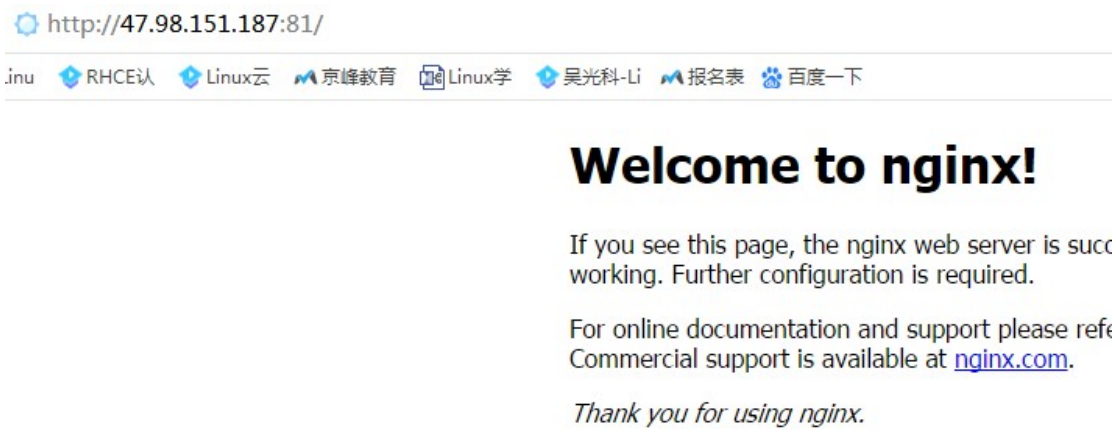
6）选择 WEB 界面 images 镜像列表，如图所示：



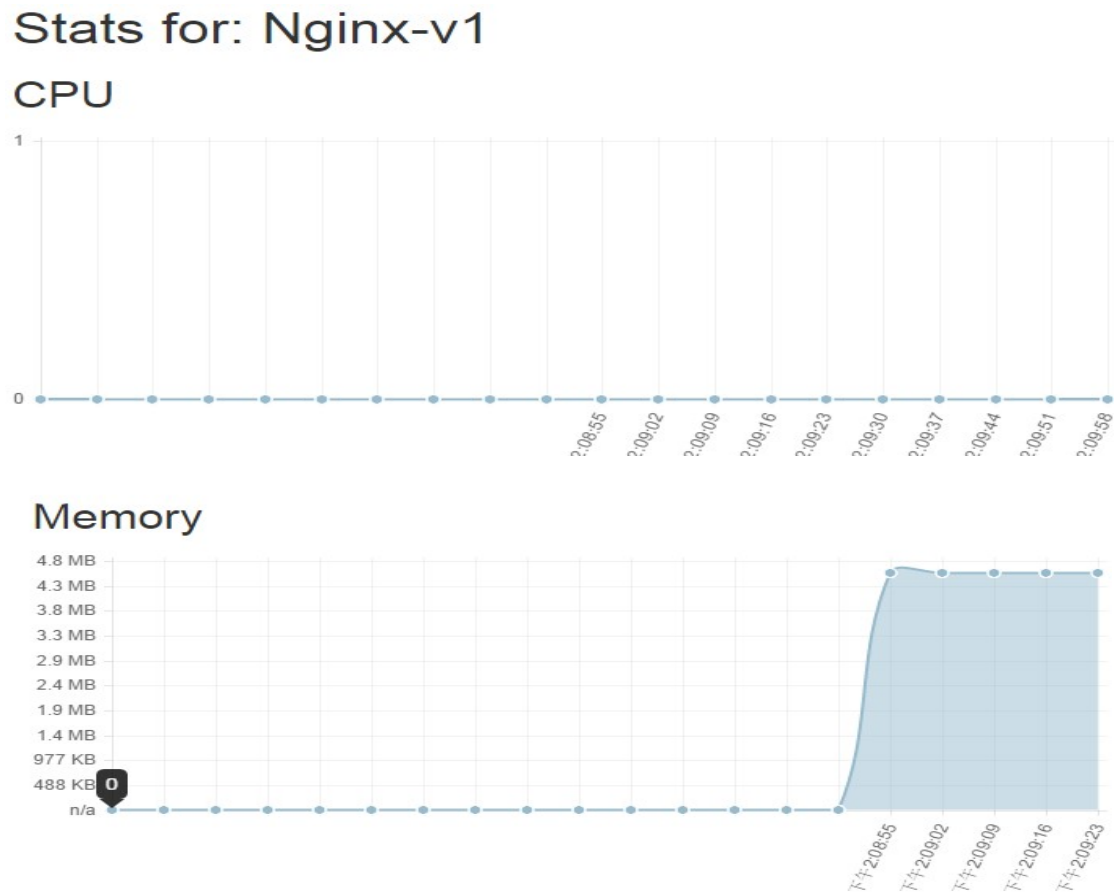
7）基于镜像启动 Docker 容器虚拟机，并且实现端口映射，如图所示：



8）创建 Docker 容器之后，通过浏览器实现访问 81 端口，如图所示：



9）Docker 容器资源 WEB 监控如图所示：



28.17 Dockerfile 企业案例演练

由于 Docker 官网公共仓库镜像大多不完整，无法真正满足企业的生产环境系统，此时需要我们自行定制镜像或者重新打包镜像。

Docker 镜像制作是管理员的必备工作之一，Docker 镜像制作的方法主要有两种，制作方法如下：

- Docker commit|export 将新容器提交至 Images 列表；
- 编写 Dockerfile，bulid 新的镜像至镜像列表；

28.18 Dockerfile 语法命令详解一

企业生产环境推荐使用 Dockerfile 制作镜像，Dockerfile 制作原理，将基于一个基础镜像，通过编写 Dockerfile 方式，将各个功能进行叠加，最终形成新的 Docker 镜像，是目前互联网企业中打包镜像最为推荐的方式。

Dockerfile 是一个镜像的表示，也是一个镜像的原材料，可以通过 Dockerfile 来描述构建镜像，并自动构建一个容器。

如下为 DockerFile 制作镜像，必备的指令和参数的详解：

FROM 指定所创建镜像的基础镜像;
MAINTAINER 指定维护者信息;
RUN 运行命令;
CMD 指定启动容器时默认执行的命令;
LABEL 指定生成镜像的元数据标签信息;
EXPOSE 声明镜像内服务所监听的端口;
ENV 指定环境变量;
ADD 赋值指定的<src>路径下的内容到容器中的<dest>路径下，<src>可以为 URL；如果为 tar 文件，会自动解压到<dest>路径下
COPY 赋值本地主机的<scr>路径下的内容到容器中的<dest>路径下；一般情况下推荐

使用 COPY 而不是 ADD;

ENTRYPOINT　指定镜像的默认入口;

VOLUME　创建数据挂载点;

USER　指定运行容器时的用户名或 UID;

WORKDIR　配置工作目录;

ARG　指定镜像内使用的参数(例如版本号信息等);

ONBUILD　配置当前所创建的镜像作为其他镜像的基础镜像时，所执行的创建操作的命令;

STOPSIGNAL　容器退出的信号;

HEALTHCHECK　如何进行健康检查;

SHELL　指定使用 SHELL 时的默认 SHELL 类型;

28.19 Dockerfile 制作规范及技巧

从企业需求出发，定制适合自己需求、高效方便的镜像，可以参考官方 Dockerfile 文件，也可以根据自身的需求，逐步的完善，在构建中不断优化 Dockerfile 文件；

Dockerfile 制作镜像规范和技巧如下：

- 精简镜像用途：尽量让每个镜像的用途都比较集中、单一，避免构造大而复杂、多功能的镜像；
- 选用合适的基础镜像：过大的基础镜像会造成构建出臃肿的镜像，一般推荐比较小巧的镜像作为基础镜像；
- 提供详细的注释和维护者信息： Dockerfile 也是一种代码，需要考虑方便后续扩展和他人使用；

- 正确使用版本号：使用明确的具体数字信息的版本号信息，而非 latest，可以避免无法确认具体版本号，统一环境；
- 减少镜像层数：减少镜像层数建议尽量合并 RUN 指令，可以将多条 RUN 指令的内容通过&&连接；
- 及时删除临时和缓存文件：这样可以避免构造的镜像过于臃肿，并且这些缓存文件并没有实际用途；
- 提高生产速度：合理使用缓存、减少目录下的使用文件，使用.dockeringore 文件等；
- 调整合理的指令顺序：在开启缓存的情况下，内容不变的指令尽量放在前面，这样可以提高指令的复用性；
- 减少外部源的干扰：如果确实要从外部引入数据，需要制定持久的地址，并带有版本信息，让他人可以重复使用而不出错。

28.20 DockerFile 企业案例一

DockerFile 企业案例一，将启动 Docker 容器，同时开启 Docker 容器对外的 22 端口的监听，实现通过 CRT 或者 Xshell 登录。

Docker 服务端创建 Dockerfile 文件，实现容器运行开启 22 端口，内容如下：

```
#设置基本的镜像，后续命令都以这个镜像为基础
FROM centos
#作者信息
MAINTAINER    WWW.JFEDU.NET
#安装依赖工具&删除默认 YUM 源，使用 YUM 源为国内 163 YUM 源；
```

```
RUN rpm --rebuilddb;yum install make wget tar gzip passwd openssh-server gcc
-y
RUN rm -rf /etc/yum.repos.d/*;wget -P /etc/yum.repos.d/
http://mirrors.163.com/.help/CentOS7-Base-163.repo
#配置 SSHD&修改 root 密码为 1qaz@WSX
RUN yes|ssh-keygen -q -t rsa -b 2048 -f /etc/ssh/ssh_host_rsa_key -N ''
RUN yes|ssh-keygen -q -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key -N ''
RUN yes|ssh-keygen -q -t ed25519 -f /etc/ssh/ssh_host_ed25519_key -N ''
RUN echo '1qaz@WSX' | passwd --stdin root
#启动 SSHD 服务进程，对外暴露 22 端口；
EXPOSE 22
CMD /usr/sbin/sshd -D
```

基于 Dockerfile 来创建生成镜像，命令如下：

用 docker build 根据 Dockerfile 创建镜像(centos:ssh)：

```
docker build -t centos:ssh - < Dockerfile
docker build -t centos:ssh .
```

28.21 DockerFile 企业案例二

DockerFile 企业案例二，开启 SSH 6379 端口，让 Redis 端口对外访问，Dockerfile 内容如下：

```
FROM centos:latest
#作者信息
```

```
MAINTAINER    WWW.JFEDU.NET

#安装依赖工具&删除默认 YUM 源，使用 YUM 源为国内 163 YUM 源；

RUN rpm --rebuilddb;yum install make wget tar gzip passwd openssh-server gcc

-y

RUN rm -rf /etc/yum.repos.d/*;wget -P /etc/yum.repos.d/

http://mirrors.163.com/.help/CentOS7-Base-163.repo

#配置 SSHD&修改 root 密码为 1qaz@WSX

RUN ssh-keygen -q -t rsa -b 2048 -f /etc/ssh/ssh_host_rsa_key -N ''

RUN ssh-keygen -q -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key -N ''

RUN ssh-keygen -q -t ed25519 -f /etc/ssh/ssh_host_ED25519_key -N ''

RUN echo '1qaz@WSX' | passwd --stdin root

#Redis 官网下载 Redis 最新版本软件；

RUN wget -P /tmp/ http://download.redis.io/releases/redis-5.0.2.tar.gz

#解压 Redis 软件包，并且基于源码安装,创建配置文件；

RUN cd /tmp/;tar xzf redis-5.0.2.tar.gz;cd redis-5.0.2;make;make

PREFIX=/usr/local/redis install;mkdir -p /usr/local/redis/etc/;cp redis.conf

/usr/local/redis/etc/

#创建用于存储应用数据目录/data/redis&修改 redis 配置文件 dir 路径；

RUN mkdir -p /data/redis/

RUN sed -i 's#^dir.*#dir /data/redis#g' /usr/local/redis/etc/redis.conf

#将应用数据存储目录/data/进行映射，可以实现数据持久化保存；

VOLUME ["/data/redis"]
```

```
#修改 Redis.conf 监听地址为 bind：0.0.0.0；

RUN sed -i '/^bind/s/127.0.0.1/0.0.0.0/g' /usr/local/redis/etc/redis.conf

#启动 Redis 数据库服务进程，对外暴露 22 和 6379 端口；

EXPOSE  22

EXPOSE  6379

CMD /usr/sbin/sshd;/usr/local/redis/bin/redis-server

/usr/local/redis/etc/redis.conf
```

28.22 DockerFile 企业案例三

DockerFile 企业案例三，基于 Dockerfile 开启 Nginx 80 端口，并远程连接服务器，dockerfile 内容如下：

```
FROM centos:latest

#作者信息

MAINTAINER  WWW.JFEDU.NET

#安装依赖工具&删除默认 YUM 源，使用 YUM 源为国内 163 YUM 源；

RUN rpm --rebuilddb;yum install make wget tar gzip passwd openssh-server gcc

pcre-devel open

ssl-devel net-tools -y

RUN rm -rf /etc/yum.repos.d/*;wget -P /etc/yum.repos.d/

http://mirrors.163.com/.help/CentOS7-Base-163.repo

#配置 SSHD&修改 root 密码为 1qaz@WSX
```

```
RUN ssh-keygen -q -t rsa -b 2048 -f /etc/ssh/ssh_host_rsa_key -N ''

RUN ssh-keygen -q -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key -N ''

RUN ssh-keygen -q -t ed25519 -f /etc/ssh/ssh_host_ED25519_key -N ''

RUN echo '1qaz@WSX' | passwd --stdin root

#Nginx 官网下载 Nginx 最新版本软件；

RUN wget -P /tmp/ http://nginx.org/download/nginx-1.14.2.tar.gz

#解压 Nginx 软件包，隐藏 WEB 服务器版本号；

RUN cd /tmp/;tar xzf nginx-1.14.2.tar.gz;cd nginx-1.14.2;sed -i -e 's/1.14.2//g' -e

's/nginx\//WS/g' -e 's/"NGINX"/"WS"/g' src/core/nginx.h

#基于源码安装,创建配置文件；

RUN cd /tmp/nginx-1.14.2;./configure --prefix=/usr/local/nginx

--with-http_stub_status_module --with-http_ssl_module;make;make install

#启动 Nginx 服务进程，对外暴露 22 和 80 端口；

EXPOSE   22

EXPOSE   80

CMD /usr/local/nginx/sbin/nginx;/usr/sbin/sshd -D
```

28.23 DockerFile 企业案例四

DockerFile 企业案例四，Docker 虚拟化中，如何来构建我们的 MYSQL 数据库服务器呢？答案很简单，是 dockerfile 来生成 mysql 镜像并启动运行即可。

```
FROM centos:v1

RUN groupadd -r mysql && useradd -r -g mysql mysql
```

```
RUN rpm --rebuilddb;yum install -y gcc zlib-devel gd-devel

ENV MYSQL_MAJOR 5.6

ENV MYSQL_VERSION 5.6.20

RUN

        && curl -SL

"http://dev.mysql.com/get/Downloads/MySQL-$MYSQL_MAJOR/mysql-$MYSQL_

VERSION-linux-glibc2.5-x86_64.tar.gz" -o mysql.tar.gz \

        && curl -SL

"http://mysql.he.net/Downloads/MySQL-$MYSQL_MAJOR/mysql-$MYSQL_VERSIO

N-linux-glibc2.5-x86_64.tar.gz.asc" -o mysql.tar.gz.asc \

        && mkdir /usr/local/mysql \

        && tar -xzf mysql.tar.gz -C /usr/local/mysql \

        && rm mysql.tar.gz* \

ENV PATH $PATH:/usr/local/mysql/bin:/usr/local/mysql/scripts

WORKDIR /usr/local/mysql

VOLUME /var/lib/mysql

EXPOSE 3306

CMD ["mysqld", "--datadir=/var/lib/mysql", "--user=mysql"]
```

28.24 Docker 本地私有仓库实战

Docker 虚拟化有三个基础概念：Docker 镜像、Docker 容器、Docker 仓库，三个概念详解如下：

1）Docker 镜像

Docker 虚拟化最基础的组件为镜像，镜像跟我们常见的 Linux ISO 镜像类似，但是 Docker 镜像是分层结构的，是由多个层级组成，每个层级分别存储各种软件实现某个功能，Docker 镜像是静止的、只读的，不能对镜像进行写操作。

2）Docker 容器：

Docker 容器是 Docker 虚拟化的产物，也是最早在生产环境使用的对象，Docker 容器的底层是 Docker 镜像，是基于镜像运行，并且在镜像最上层添加一层容器层之后的实体，容器层是可读、可写的，容器层如果需用到镜像层中的数据，可以通过 JSON 文件读取镜像层中的软件和数据，对整个容器进行修改，只能作用于容器层，不能直接对镜像层进行写操作。

3）Docker 仓库：

Docker 仓库是用于存放 Docker 镜像的地方，Docker 仓库分为两类，分别是：公共仓库（Public）和私有仓库（Private），国内和国外有很多默认的公共仓库，对外开放、免费或者付费使用，企业测试环境和生产环境推荐自建私有仓库，私有仓库的特点：安全、可靠、稳定、高效，能够更加自身的业务体系进行灵活升级和管理。

纵观 Docker 镜像、容器、仓库，其中最重要的，最基础的属 Docker 镜像，没有镜像的概念就没有容器，而且镜像是静止的、只读的模板文件层，是存储在 Docker 仓库中。

Docker 默认连接的国外官方镜像，通常根据网络情况不同，访问时快时慢，大多时候获取速度非常慢，为了提示效率可以自建仓库或者先修改为国内仓库源，提升拉取镜像的速度。

Docker 可以配置的国内镜像有很多可供选择，例如：Docker 中国区官方镜像、阿里云、网易蜂巢、DaoCloud 等，这些都是国内比较快的镜像仓库。

28.25 Docker 国内源实战

4）修改 Docker 默认镜像源方法：

vim /etc/docker/daemon.json，修改为如下内容：

```
{
"registry-mirrors":["https://registry.docker-cn.com"]
}
```

5）修改完成，重启 Docker 引擎服务即可；

```
service docker restart
```

28.26 Docker Registry 仓库源实战

Docker 仓库分为公共仓库和私有仓库，在企业测试环境、生产环境推荐自建内部私有仓库，使用私有仓库的优点：

- 节省网络带宽，针对于每个镜像不用去 Docker 官网仓库下载；
- 下载 Docker 镜像从本地私有仓库中下载；
- 组件公司内部私有仓库，方便各部门使用，服务器管理更加统一；
- 可以基于 GIT 或者 SVN、Jenkins 更新本地 Docker 私有仓库镜像版本。

官方提供 Docker Registry 来构建本地私有仓库，目前最新版本为 v2，最新版的 docker 已不再支持 v1，Registry v2 使用 Go 语言编写，在性能和安全性上做了很多优化，重新设计了镜像的存储格式。

1）登陆 Docker 仓库服务器，下载 Docker registry 镜像，命令如下：

docker pull registry

```
[root@jfedu123 ~]# docker pull registry
Using default tag: latest
Trying to pull repository docker.io/library/registry ...
latest: Pulling from docker.io/library/registry
cd784148e348: Pull complete
0ecb9b11388e: Pull complete
45793cf0ff93: Pull complete
d7eadb9e7675: Pull complete
4b2356bbbed3: Pull complete
Digest: sha256:a54bc9be148764891c44676ce8c44f1e53514c43b1bfb
Status: Downloaded newer image for docker.io/registry:latest
[root@jfedu123 ~]# docker images
REPOSITORY          TAG                 IMAGE ID
docker.io/registry  latest              116995fd6624
[root@jfedu123 ~]#
```

2） 启动私有仓库容器，启动命令如下：

mkdir -p /data/registry/

docker run -itd -p 5000:5000 -v /data/registry:/var/lib/registry

docker.io/registry

netstat -tnlp|grep -w 5000

```
[root@jfedu123 ~]# mkdir -p  /data/registry/
[root@jfedu123 ~]#
[root@jfedu123 ~]# docker run -itd  -p  5000:5000 -v /data/regist
c9c344cebaf1e15c9f375b75ff708e7c78051278103fae4c702e47027bf25115
[root@jfedu123 ~]#
[root@jfedu123 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND
                NAMES
c9c344cebaf1        docker.io/registry   "/entrypoint.sh /e..."
:5000->5000/tcp   pedantic_murdock
[root@jfedu123 ~]#
[root@jfedu123 ~]# netstat -tnlp|grep -w 5000
tcp6       0      0 :::5000                 :::*
[root@jfedu123 ~]#
```

默认情况下，会将仓库存放于容器内的/var/lib/registry 目录下，这样如果容器被删除，则存放于容器中的镜像也会丢失，所以我们一般情况下会指定本地一个目录挂载到容器内的/data/registry 下。

3） 上传镜像至本地私有仓库：

客户端上传镜像至本地私有仓库，如下以 busybox 镜像为例，将 busybox 上传至私有仓库服务器。


```
[root@jfedu123 ~]# mkdir -p  /data/registry/
[root@jfedu123 ~]#
[root@jfedu123 ~]# docker run -itd  -p  5000:5000 -v /data/regist
c9c344cebaf1e15c9f375b75ff708e7c78051278103fae4c702e47027bf25115
[root@jfedu123 ~]#
[root@jfedu123 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND
                NAMES
c9c344cebaf1        docker.io/registry   "/entrypoint.sh /e..."
:5000->5000/tcp   pedantic_murdock
[root@jfedu123 ~]#
[root@jfedu123 ~]# netstat -tnlp|grep -w 5000
tcp6       0      0 :::5000                 :::*
[root@jfedu123 ~]#
```


docker pull busybox

docker tag busybox 192.168.1.123:5000/busybox

docker push 192.168.1.123:5000/busybox

```
[root@jfedu123 ~]#
[root@jfedu123 ~]# docker   pull    busybox
Using default tag: latest
Trying to pull repository docker.io/library/busybox ...
latest: Pulling from docker.io/library/busybox
57c14dd66db0: Pull complete
Digest: sha256:7964ad52e396a6e045c39b5a44438424ac52e12e4d5a25d
Status: Downloaded newer image for docker.io/busybox:latest
[root@jfedu123 ~]# docker   tag    busybox  192.168.1.123:5000
[root@jfedu123 ~]# docker   push  192.168.1.123:5000/busybox
The push refers to a repository [192.168.1.123:5000/busybox]
Get https://192.168.1.123:5000/v1/_ping: http: server gave HTT
[root@jfedu123 ~]#
```

默认往 Docker 仓库，报错解决方法：

vim /etc/sysconfig/docker 配置文件：

注释或者删除以 OPTION 开头的行，然后加入如下两行代码：

OPTIONS='--selinux-enabled --log-driver=journald

--signature-verification=false --insecure-registry 192.168.1.123:5000'

ADD_REGISTRY='--add-registry 192.168.1.123:5000'

4） 检测本地私有仓库：

curl -XGET http://192.168.0.123:5000/v2/_catalog

curl -XGET http://192.168.0.123:5000/v2/busybox/tags/list

```
[root@jfedu123 ~]#
[root@jfedu123 ~]# docker   push  192.168.1.123:5000/busybox
The push refers to a repository [192.168.1.123:5000/busybox]
683f499823be: Layer already exists
latest: digest: sha256:bbb143159af9eabdf45511fd5aab4fd2475d4c0
[root@jfedu123 ~]#
[root@jfedu123 ~]# curl -XGET http://192.168.1.123:5000/v2/_ca
{"repositories":["busybox"]}
[root@jfedu123 ~]#
[root@jfedu123 ~]# curl -XGET http://192.168.1.123:5000/v2/bus
{"name":"busybox","tags":["latest"]}
[root@jfedu123 ~]#
[root@jfedu123 ~]#
```

5） 客户端使用本地私有仓库：

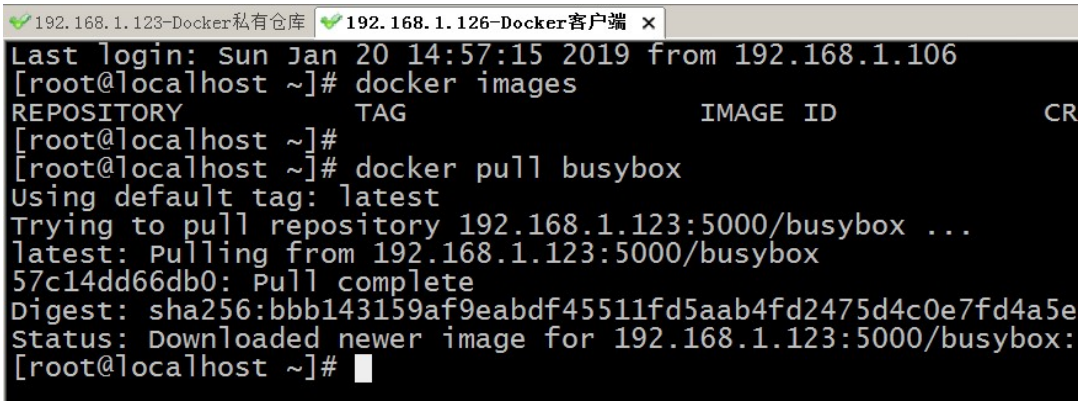
登陆 Docker 客户端，同样在其/etc/sysconfig/docker 配置文件添加如下代码，同时重启 docker 服务，获取本地私有仓库如图 24-3 所示：

OPTIONS='--selinux-enabled --log-driver=journald

--signature-verification=false --insecure-registry 192.168.1.123:5000'

ADD_REGISTRY='--add-registry 192.168.1.123:5000'

重启 Docker 服务，然后从 Docker 仓库下载 busybox 镜像，如图所示：



28.27 Docker 磁盘&内存&CPU 资源实战一

通过前面章节 Docker 内容的学习，我们知道 Docker 引擎启动的容器（虚拟机），默认会共享宿主机所有的硬件资源（CPU、内存、硬盘等）。

测试环境或生产环境，如果某一个 Docker 容器非常占资源，而且又没有对其做任何资源的限制，会导致其他的容器没有资源可用，甚至导致宿主机奔溃，为了防止意外或者错误产生，通常会对 Docker 容器进行资源隔离和限制，默认 Docker 基于 Cgroup 隔离子系统来实施资源隔离。

基于 Docker run 启动容器时，可以直接限制 CPU 和内存的资源，但是不能直接限制其对硬盘容量的使用。

1）CPU 和内存资源限制案例一：

基于 Docker 引擎启动一台 CentOS 容器，并且设置 CPU 为 2 核，内存为 4096MB，启动命令如下：

```
docker run -itd --privileged --cpuset-cpus=0-1 -m 4096m centos:latest
```

2）CPU 和内存资源限制案二:

基于 Docker 引擎启动一台 CentOS 容器，并且设置 CPU 为 4 核，内存为 8192MB，启动命令如下：

```
docker run -itd --privileged --cpuset-cpus=2-5 -m 8192m centos:latest
```

如上方法只能限制 Docker 容器 CPU 和内存的资源隔离，如果要实现硬盘容量的限制，没有默认参数设置，需要通过如下方法来实现。

3）基于 DeviceMapper 硬盘容量限制

限制 Docker 容器硬盘容量资源，不同的硬盘驱动方式，操作方法不一样，例如基于 DeviceMapper 驱动方式，Docker 容器默认分配硬盘的 rootfs 根分区的容量为 10G。

可以指定默认容器的大小（在启动容器的时候指定），可以在 docker 配置文件里通过 dm.basesize 参数指定，指定 Docker 容器 rootfs 容量大小为 20G：

```
docker -d --storage-opt dm.basesize=20G
```

也可以修改 Docker 引擎，默认存储配置文件 vim /etc/sysconfig/docker-storage，在 OPTIONS 参数后面，加入如下代码：

--storage-opt dm.basesize=20G

最终 docker-storage 文件，添加之后的代码如下：

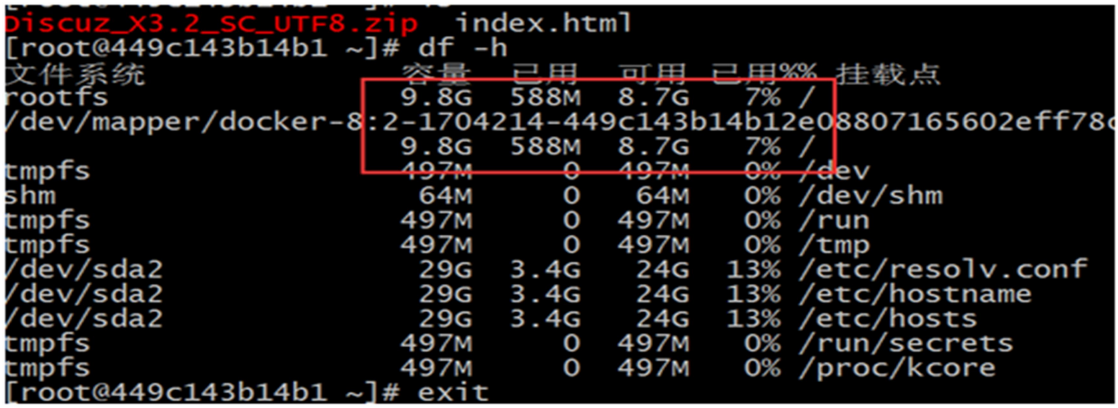
DOCKER_STORAGE_OPTIONS="--storage-driver devicemapper

--storage-opt dm.basesize=20G"

然后重启 docker 即可;

除了第一种默认方法，限制 Docker 容器硬盘容量之外，还可以基于现有容器在线扩容，宿主机文件系统类型支持：ext2、ext3、ext4、不支持 XFS。

查看原容器的磁盘空间大小：



查看 mapper 设备：

查看卷信息表：

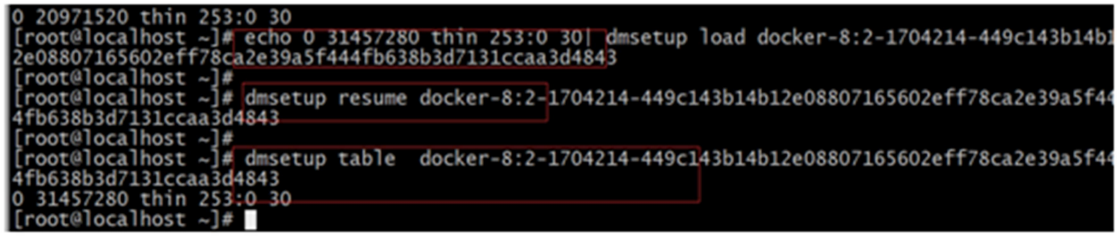
根据要扩展的大小，计算需要多少扇区：

第二个数字是设备的大小，表示有多少个 512 - bytes 的扇区. 这个值略高于 10GB 的大小。

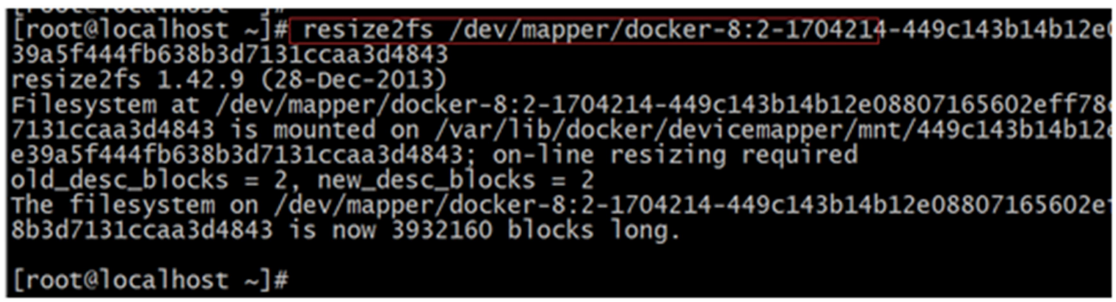
我们来计算一下一个 15GB 的卷需要多少扇区，

echo $((1510241024*1024/512)) 31457280

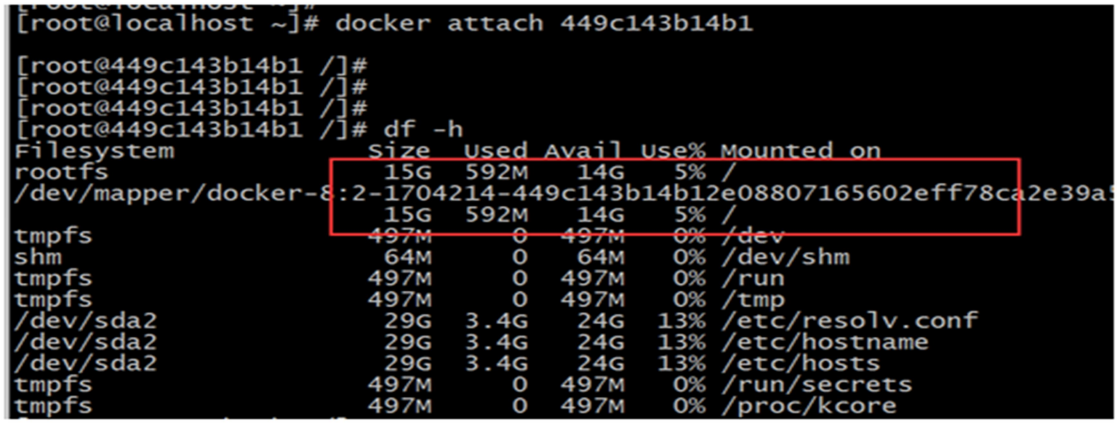
修改卷信息表--激活--并且验证（红色 3 个部分）



修改文件系统大小：



最后验证磁盘大小：



成功扩容，当然了以上步骤也可以写成脚本，然后使用脚本批量扩容分区大小。

28.28 Docker 磁盘&内存&CPU 资源实战二

Docker 容器默认启动的虚拟机，会占用宿主机的资源（CPU、内存、硬盘），例如默认 Docker 基于 Overlay2 驱动方式，容器硬盘的 rootfs 根分区空间是整个宿主机的空间大小。

可以指定默认容器的大小（在启动容器的时候指定），可以在 docker 配置文件：/etc/sysconfig/docker 中，OPTIONS 参数后面添加如下代码,指定 Docker 容器 rootfs 容量大小为 10G：

OPTIONS='--storage-opt overlay2.size=10G'

以上方法只适用于新容器生成，并且修改后需要重启 docker，无法做到动态给正在运行容器指定大小，如下图为默认容器大小：

```
[root@jfedu141 ~]# docker ps
CONTAINER ID        IMAGE                                COMMAN
   NAMES
0536d990db13        docker.io/lemonbar/centos6-ssh       "/bin/
   distracted_kilby
[root@jfedu141 ~]# docker exec 0536d990db13 df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs           20G  1.5G   19G   8% /
overlay          20G  1.5G   19G   8% /
tmpfs           493M     0  493M   0% /dev
tmpfs           493M     0  493M   0% /sys/fs/cgroup
/dev/sdb         20G  1.5G   19G   8% /etc/resolv.conf
```

修改 Docker 存储配置文件，加入如下代码：(默认如果已经为 overlay2，则无需修改)

vi /etc/sysconfig/docker-storage

DOCKER_STORAGE_OPTIONS="--storage-driver overlay2 "

然后重启 docker 即可;

```
[root@localhost sysconfig]# service docker restart
Redirecting to /bin/systemctl restart  docker.service
[root@localhost sysconfig]#
```

Overlay2 Docker 磁盘驱动模式，如果要调整其大小，通过如上的方法，会导致 Docker 引擎服务无法启动，还需要让 Linux 文件系统设置为 xfs，并且支持目录级别的磁盘配额功能;

CentOS7.x Xfs 磁盘配额配置，新添加一块硬盘，设置磁盘配额方法步骤如下：

1）添加新的硬盘如图所示：

```
[root@jfedu141 ~]# fdisk -l

Disk /dev/sda: 21.5 GB, 21474836480 bytes, 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000a39b8

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048      411647      204800   83  Linux
/dev/sda2          411648     1460223      524288   82  Linux swap
/dev/sda3         1460224    41943039    20241408   83  Linux

Disk /dev/sdb: 21.5 GB, 21474836480 bytes, 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

2）格式化硬盘为 xfs 文件系统格式，命令如下：

mkfs.xfs -f /dev/sdb

3）创建 data 目录，后续将作为 docker 数据目录；

mkdir /data/ -p

4）挂载 data 目录，并且开启磁盘配额功能（默认 xfs 支持配额功能）；

mount -o uquota,prjquota /dev/sdb /data/

```
[root@jfedu141 ~]# mount -o uquota,prjquota /dev/sdb /data/
[root@jfedu141 ~]#
[root@jfedu141 ~]# cd /data/
[root@jfedu141 data]# ls
[root@jfedu141 data]#
[root@jfedu141 data]# ll
total 0
[root@jfedu141 data]# mkdir docker
[root@jfedu141 data]# ls
docker
[root@jfedu141 data]#
[root@jfedu141 data]# mount|grep data
/dev/sdb on /data type xfs (rw,relatime,attr2,inode64,usrquota,prjquota)
[root@jfedu141 data]#
```

挂载配额类型如下：

- 根据用户(uquota/usrquota/quota)
- 根据组(gquota/grpquota)；
- 根据目录(pquota/prjquota)(不能与 grpquota 同时设定)。

5）查看配额-配置详情，命令如下：

xfs_quota -x -c 'report' /data/

```
[root@jfedu141 ~]# xfs_quota -x -c 'report' /data/
User quota on /data (/dev/sdb)
                                Blocks
User ID          Used       Soft       Hard    Warn/Grace
---------- --------------------------------------------------
root                0          0          0     00 [--------]

Project quota on /data (/dev/sdb)
                                Blocks
Project ID       Used       Soft       Hard    Warn/Grace
---------- --------------------------------------------------
#0                  0          0          0     00 [--------]

[root@jfedu141 ~]#
```

6）可以通过命令 xfs_quota 设置来为用户和目录分配配额，也可以通过命令来查看配额信息；

xfs_quota -x -c 'limit bsoft=10M bhard=10M jfedu' /data

xfs_quota -x -c 'report' /data/

```
[root@jfedu141 ~]# xfs_quota -x -c 'limit bsoft=10M bhard=10M jfedu' /data
[root@jfedu141 ~]#
[root@jfedu141 ~]#
[root@jfedu141 ~]# xfs_quota -x -c 'report' /data/
User quota on /data (/dev/sdb)
                                Blocks
User ID          Used       Soft       Hard    Warn/Grace
---------- --------------------------------------------------
root                0          0          0     00 [--------]
jfedu               0      10240      10240     00 [--------]

Project quota on /data (/dev/sdb)
                                Blocks
Project ID       Used       Soft       Hard    Warn/Grace
```

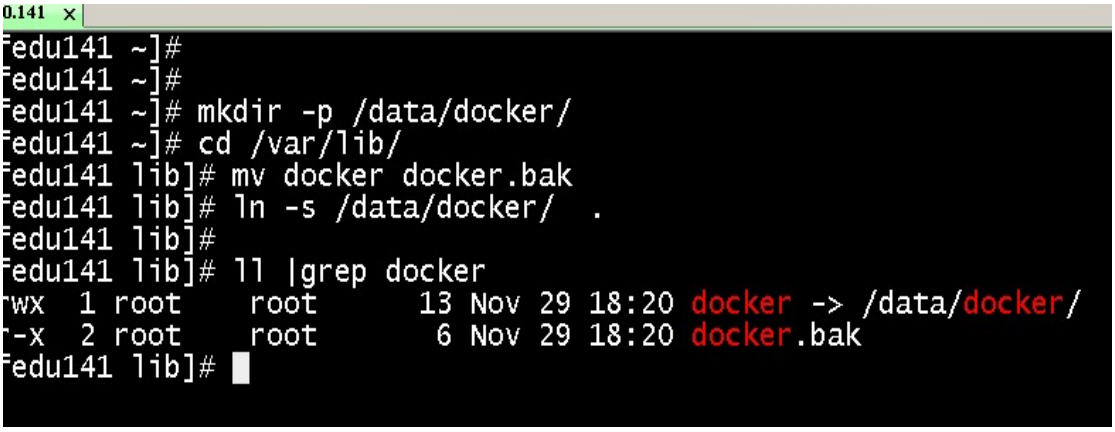
7）将 docker 引擎默认数据存储目录:/var/lib/docker 重命名，并且将/data/docker 目录软链接至/var/lib/下即可；

mkdir -p /data/docker/

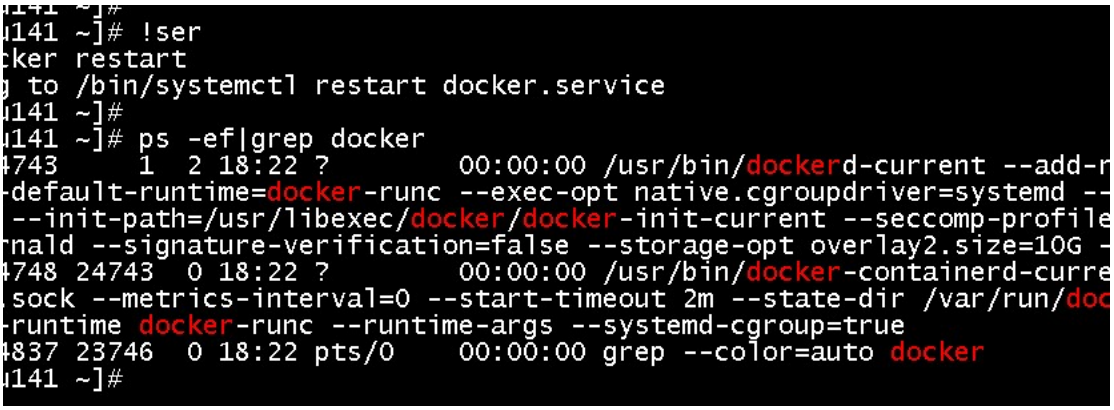
cd /var/lib/

mv docker docker.bak

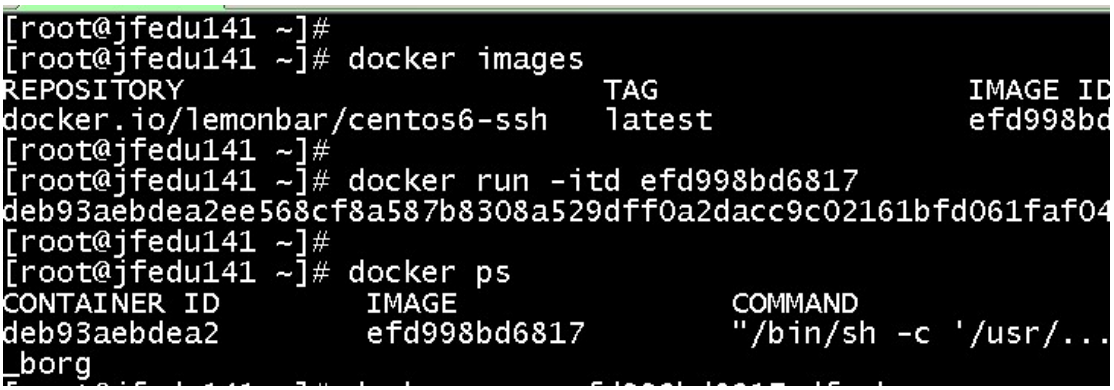
ln -s /data/docker/ .



8）重启 Docker 服务，并且查看进程，可以看到 docker overlay2.size 大小配置，如图所示：



9）基于 Docker 客户端指令启动 Docker 容器，并且查看最新容器的磁盘空间为 10G，则设置容器大小成功，如图所示：



```
[root@jfedu141 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND
deb93aebdea2        efd998bd6817        "/bin/sh -c '/usr/.
_borg
[root@jfedu141 ~]#
[root@jfedu141 ~]# docker exec deb93aebdea2 df -h
Filesystem      Size  Used Avail Use% Mounted on
rootfs           10G   12K   10G   1% /
overlay          10G   12K   10G   1% /
tmpfs           493M     0  493M   0% /dev
tmpfs           493M     0  493M   0% /sys/fs/cgroup
/dev/sdb         20G  347M   20G   2% /etc/resolv.conf
/dev/sdb         20G  347M   20G   2% /etc/hostname
/dev/sdb         20G  347M   20G   2% /etc/hosts
```

28.29 Docker-Compose 企业生产环境实战

Docker Compose 是 Docker 官方编排（Orchestration）项目之一，负责快速在集群中部署分布式应用。Compose 定位是“defining and running complex applications with Docker”，前身是 Fig，兼容 Fig 的模板文件。

28.30 Docker-Compose 简介

学了前面的章节内容，我们知道 Dockerfile 可以让用户管理一个单独的应用容器，而今天即将学习的 Compose 则是允许用户在一个模板（YAML 格式）中定义一组相关联的应用容器（被称为一个 project，即项目），例如：一台 Web 服务容器关联后端的数据库服务容器等。

Docker-Compose 将所管理的容器分为三层，分别是工程（project），服务（service）以及容器（container）。Docker-Compose 运行目录下的所有文件（docker-compose.yml，extends 文件或环境变量文件等）组成一个工程，若无特殊指定工程名即为当前目录名。一个工程当中可包含多个服务，每个服务中定义了容器运行的镜像、参数、依赖。

一个服务当中可包括多个容器实例，Docker-Compose 并没有解决负载均衡的问题，因此需要借助其它工具实现服务发现及负载均衡。

Docker-Compose 的配置文件默认为 docker-compose.yml，可通过环境变量 COMPOSE_FILE 或-f 参数自定义配置文件，其定义了多个有依赖关系的服务及每个服务运行的容器。

使用 Dockerfile 模板文件，可以让用户很方便的定义一个单独的应用容器。在工作中，经常会碰到需要多个容器相互配合来完成某项任务的情况。例如要实现一个 Web 项目，除了 Web 服务容器本身，往往还需要再加上后端的数据库服务容器，甚至还包括负载均衡容器等。

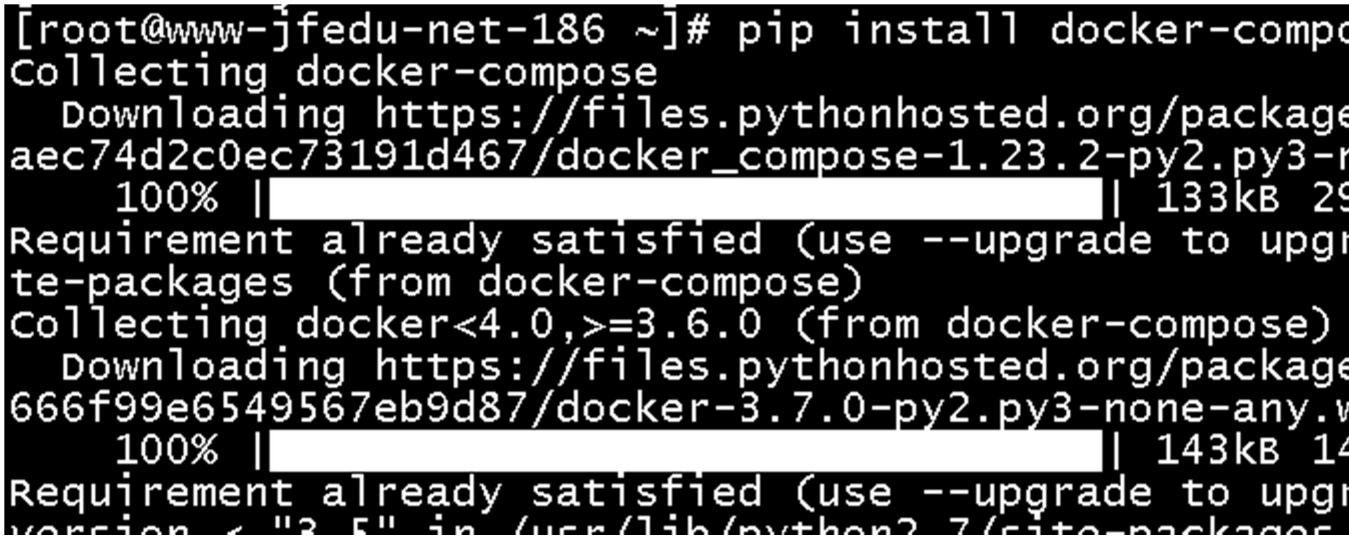
Docker-Compose 允许用户通过单独的 docker-compose.yml 模板文件（YAML 格式）来定义一组相关联的应用容器为一个项目（project）。

Docker-Compose 项目由 Python 编写，调用 Docker 服务提供的 API 来对容器进行管理。因此，只要所操作的平台支持 Docker API，就可以在其上利用 Compose 来进行编排管理。

28.31 Docker-Compose 部署安装

安装 Compose 之前，需要先安装 Docker 引擎服务，此处使用 PIP 安装 Compose 项目，要求系统必须提前安装 pip 工具，然后执行如下命令：

```
pip install --upgrade pip
pip install -U docker-compose
```



安装成功后，可以查看 docker-compose 命令的用法，执行如下指令：

docker-compose -h

```
[root@www-jfedu-net-186 ~]# docker-compose -h
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compo
                              (default: docker-compose.y
  -p, --project-name NAME     Specify an alternate proje
                              (default: directory name)
  --verbose                   Show more output
```

命令名称	命令功能	中文备注
build	Build or rebuild services	构建服务
config	Validate and view the Compose file	配置和查看
create	Create services	创建服务
down	Stop and remove containers	停止和移除
exec	Execute a command in a running container	执行命令
help	Get help on a command	帮助命令
images	List images	查看镜像
kill	Kill containers	杀掉容器
logs	View output from containers	查看日志
pause	Pause services	暂停服务
port	Print the public port for a port binding	打印端口
ps	List containers	列出容器
pull	Pull service images	下载镜像
push	Push service images	上传镜像
restart	Restart services	重启服务
rm	Remove stopped containers	删除容器
run	Run a one-off command	运行命令
start	Start services	启动服务
stop	Stop services	停止服务
top	Display the running processes	查看服务进程
up	Create and start containers	启动容器
version	Docker-Compose version information	查看版本信息

安装完成之后，可以添加 bash 补全命令。

curl -L

https://raw.githubusercontent.com/docker/compose/1.2.0/contrib/completion/bash/docker-compose > /etc/bash_completion.d/docker-compose

28.32 Docker-Compose 常见概念

- 服务（service）：一个应用容器，实际上可以运行多个相同镜像的实例。
- 项目(project)：由一组关联的应用容器组成的一个完整业务单元。

一个项目可以由多个服务（容器）关联而成，Compose 是 面向项目进行管理。

28.33 Docker-Compose 实战案例一

基于 docker-compose 构建 Nginx 容器，并且实现发布目录映射，通过浏览器实现访问，操作步骤如下：

1）编写 docker-compose.yml 文件，内容如下：

```
version: "3" services:
  nginx:
    container_name: www-nginx
    image: nginx:latest
    restart: always
    ports:
      - 80:80
    volumes:
    - /data/webapps/www/:/usr/share/nginx/html/
```

1）创建发布目录：/data/webapps/www/，并且在发布目录新建 index.html 页面，命令如下：

```
mkdir -p /data/webapps/www/

echo "<h1>www.jfedu.net Nginx Test

pages.</h1>" >>/data/webapps/www/index.html
```

3）启动和运行 docker-compose，启动 Nginx 容器，命令如下：

```
docker-compose up -d
```

```
[root@www-jfedu-net-186 compose]#
[root@www-jfedu-net-186 compose]# docker-compose up -d
Creating network "compose_default" with the default driver
Pulling nginx (nginx:latest)...
latest: Pulling from library/nginx
6ae821421a7d: Pull complete
da4474e5966c: Pull complete
eb2aec2b9c9f: Pull complete
Creating www-nginx ...
Pulling nginx (nginx:latest)...
Creating www-nginx ... done
[root@www-jfedu-net-186 compose]#
```

4）通过浏览器访问宿主机 80 端口，即可访问 Nginx 容器，如图所示：

5）docker-compose.yml 内容剖析：

version：版本号，通常写 2 和 3 版本；

service：**Docker** 容器服务名称；

container_name:容器的名称

restart：设置为 **always**，容器在停止的情况下总是重启；

image：**docker** 官方镜像上找到最新版的镜像。

ports：容器自己运行的端口号和需要暴露的端口号

volumes:数据卷。表示数据、配置文件等存放的位置。(- . 这个表示 **docker-compose**.yml 当前目录)

28.34 Docker-Compose 实战案例二

基于 docker-compose 构建 Nginx 容器和 Tomcat 容器，并且实现 Nginx 和 Tomcat 发布目录映射，同时实现 nginx 均衡 tomcat 服务，通过浏览器访问 Nginx 80 端口即访问 Tomcat 的 8080 端口，操作步骤如下：

1）编写 docker-compose.yml 文件，内容如下：

```
version: "3"services:
  tomcat01:
    container_name: tomcat01
    image: tomcat:latest
    restart: always
    ports:
      - 8080
  tomcat02:
    container_name: tomcat02
```

```
    image: tomcat:latest
    restart: always
    ports:
      - 8080
  nginx:
    container_name: www-nginx
    image: nginx:latest
    restart: always
    ports:
        - 80:80
    volumes:
        - /data/webapps/www/:/usr/share/nginx/html/
        - ./default.conf:/etc/nginx/conf.d/default.conf
    links:
        - tomcat01
        - tomcat02
```

2）创建发布目录：/data/webapps/www/，并且在发布目录新建 index.html 页面，命令如下：

```
mkdir -p /data/webapps/www/
echo "<h1>www.jfedu.net Nginx Test
pages.</h1>" >>/data/webapps/www/index.html
```

3）创建 nginx 默认配置文件：default.conf，内容如下：

```
upstream tomcat_web {
        server tomcat01:8080 max_fails=2 fail_timeout=15;
        server tomcat02:8080 max_fails=2 fail_timeout=15;
}
server {
    listen          80;
    server_name    localhost;
    location / {
        root     /usr/share/nginx/html;
        index    index.html index.htm;
        proxy_pass http://tomcat_web;
        proxy_set_header host $host;
    }
    error_page     500 502 503 504    /50x.html;
    location = /50x.html {
        root     /usr/share/nginx/html;
    }
}
```

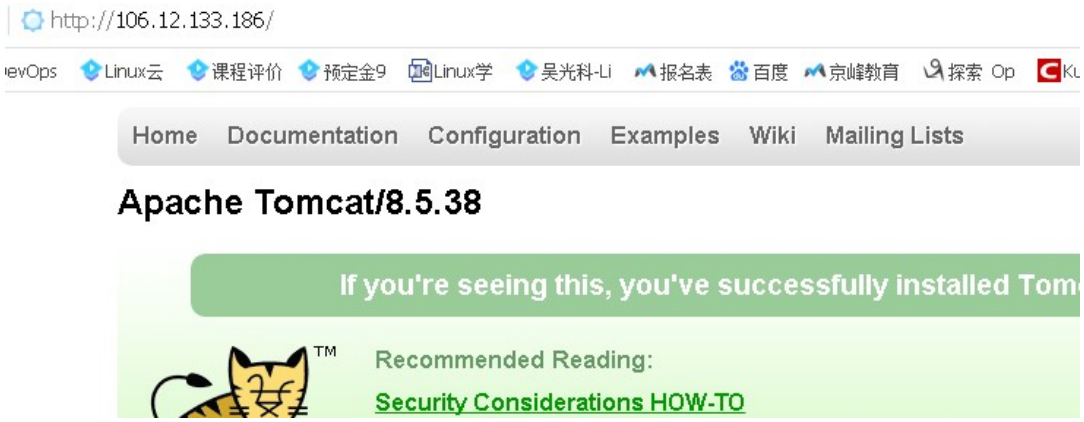
4）启动 Docker-compose，命令&如图所示：

docker-compose up -d

```
[root@www-jfedu-net-186 compose]# ls
default.conf  docker-compose.yml
[root@www-jfedu-net-186 compose]# docker-compose up -
Creating network "compose_default" with the default d
Creating tomcat01 ... done
Creating tomcat02 ... done
Creating www-nginx ... done
[root@www-jfedu-net-186 compose]#
```

```
Creating tomcat02 ... done
Creating www-nginx ... done
[root@www-jfedu-net-186 compose]# docker ps
CONTAINER ID        IMAGE               COMMAND
     PORTS                    NAMES
6b66efeef61b        nginx:latest        "nginx -g 'dae
     0.0.0.0:80->80/tcp       www-nginx
8494d9f97807        tomcat:latest       "catalina.sh r
     0.0.0.0:1028->8080/tcp   tomcat01
5c17437cde49        tomcat:latest       "catalina.sh r
     0.0.0.0:1027->8080/tcp   tomcat02
[root@www-jfedu-net-186 compose]#
```

5）通过浏览器直接访问 Nginx 容器，默认访问宿主机的 80 端口即可，如图所示：



6）查看运行的 docker 容器引用的状态和进程信息，命令如下：

docker-compose ps

docker-compose top

```
[root@www-jfedu-net-186 compose]#
[root@www-jfedu-net-186 compose]# docker-compose ps
  Name              Command               State           Ports
-------------------------------------------------------------------------
tomcat01    catalina.sh run              Up      0.0.0.0:1028->8080/tcp
tomcat02    catalina.sh run              Up      0.0.0.0:1027->8080/tcp
www-nginx   nginx -g daemon off;         Up      0.0.0.0:80->80/tcp
[root@www-jfedu-net-186 compose]#
[root@www-jfedu-net-186 compose]# docker-compose top
tomcat01
UID      PID     PPID     C    STIME    TTY      TIME
-------------------------------------------------------------------------
root    36343    36320    1    17:19    ?      00:00:06    /docker-java-
                                                            ng.config.fil
```

第29章　企业运维故障案例实战篇

29.1 企业运维故障案例

Nginx 是目前主流的 WEB 服务器发布软件，不仅可以作为强大的 web 服务器，也可以作为一个反向代理服务器，而且 nginx 还可以按照调度规则实现动静分离，可以按照轮询、ip_hash、URL 哈希、权重等多种方式对后端服务器做负载均衡，同时还支持后端服务器的健康检查。

```
upstream tdt_jfedu {
    ip_hash;
     server     10.10.141.32:8080 weight=1 max_fails=2 fail_timeout=30s;
     server     10.10.141.32:8081 weight=1 max_fails=2 fail_timeout=30s;
}
```

1、upstream 的 fail_timeout 和 max_fails，用来判断负载均衡 upstream 中的某个 server 是否失效。在 fail_timeout 的时间内，nignx 与 upstream 中某个 server 的连接尝试失败了 max_fails 次，则 nginx 会认为该 server 已经失效。在接下来的 fail_timeout 时间内，nginx 不再将请求分发给失效的 server。

2、fail_timeout 默认为10秒，max_fails 默认为1。是指在10秒内 nginx 与后端 server 连

接失败一次，如果在10秒内 nginx 与后端的连接失败达到一次，nginx 认为这个 server 已失效，在接下来的10秒内，nginx 将不会分发请求到这台后端。

3、如果 max_fails=0，即关闭后端服务器健康检查，如果权重一样，那么每次请求都会有机会发到后端不可用的服务器。另外，fail_timeout 设置的时间对响应时间没影响，这个响应时间是用接下来的 proxy_connect_timeout 和 proxy_read_timeout 来控制。

4、proxy_connect_timeout

nginx 与后端连接的超时时间，单位为秒，默认为60秒。我们在 nginx 错误日志里面看到的(110: Connection timed out)，就是指 nginx 与后端连接已经超时。

5、proxy_read_timeout

建立连接后，nginx 等候读取后端服务器响应的时间，默认为60秒。在一些比较繁忙的后端，比如线程数经常达到峰值了的 tomcat，这个值注意不要设得太低，虽然线程数已经用光，但请求已经进入待队列之中。

6、proxy_send_timeout

nginx 转发请求到后端的超时时间，默认为60秒，在这段时间内 nginx 没将请求数据发到后端将关闭连接。这个在网站有比较多像表单(post)之类的需要留意一下.

7、keepalive_timout 时间值意味着：一个 http 产生的 tcp 连接在传送完最后一个响应后，还需要等待 keepalive_timeout 秒后，才开始关闭这个连接。

1) Nginx 配置文件优化参数:

```
worker_processes 8;
```

nginx 进程数，建议按照 cpu 数目来指定，一般为它的倍数。

```
worker_cpu_affinity  00000001  00000010  00000100  00001000  00010000
```

```
00100000 01000000 10000000;
```

为每个进程分配 cpu，上例中将8个进程分配到8个 cpu，当然可以写多个，或者将一个进程分配到多个 cpu。

```
worker_rlimit_nofile 102400;
```

这个指令是指当一个 nginx 进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与 nginx 进程数相除，但是 nginx 分配请求并不是那么均匀，所以最好与 ulimit -n 的值保持一致。

```
use epoll;
```

使用 epoll 的 I/O 模型。

```
worker_connections 102400;
```

每个进程允许的最多连接数，理论上每台 nginx 服务器的最大连接数为 worker_processes*worker_connections。

```
keepalive_timeout 60;
```

keepalive 超时时间。

```
client_header_buffer_size 4k;
```

客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令 getconf PAGESIZE 取得。

```
open_file_cache max=102400 inactive=20s;
```

这个将为打开文件指定缓存，默认是没有启用的，max 指定缓存数量，建议和打开文件数一致，inactive 是指经过多长时间文件没被请求后删除缓存。

```
open_file_cache_valid 30s;
```

这个是指多长时间检查一次缓存的有效信息。

```
open_file_cache_min_uses 1;
```

open_file_cache 指令中的 inactive 参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在 inactive 时间内一次没被使用，它将被移除。

2) Nginx 浏览器缓存优化:

浏览器缓存（Browser Caching） 是为了加速浏览并节约网络资源，浏览器在用户磁盘上对最近请求过的文档进行存储。

nginx 可以通过 expires 指令来设置浏览器的 Header。

Expires 指令设置:

```
location~ \.(gif|jpg|jpeg|png|bmp|ico|txt|png|js|css|swf|doc)$ {
                expires 30d;
}
```

3) 系统 fstab 系统性能优化

当文件被创建，修改和访问时，Linux 系统会记录这些时间信息。当系统的读文件操作频繁时，记录文件最近一次被读取的时间信息，将是一笔不少的开销。

所以，为了提高系统的性能，可以在读取文件时不修改文件的 atime 属性。可以通过

在加载文件系统时使用 notime 选项来做到这一点。

当以 noatime 选项加载（mount）文件系统时，对文件的读取不会更新文件属性中的 atime 信息。设置 noatime 的重要性是消除了文件系统对文件的写操作，文件只是简单地被系统读取。

一般添加在/etc/fstab 里面，如下配置：

tmpfs /dev/shm tmpfs defaults 0 0

devpts /dev/pts devpts gid=5,mode=620 0 0

sysfs /sys sysfs defaults 0 0

proc /proc proc defaults,**noatime** 0 0

29.2 企业实战服务器 RAID 讲解及配置

目前在大部分企业里面用的最多的服务器就是 DELL 服务器，其中最有代表性的服务器例如 DELL R710、DELL R720 2U 系列或者 DELL R820 4U 服务器等等。(2U 大概为 8.74CM) 标准机柜为 42U，一个机柜可以放置 12 台 DELL R720 2U 服务器。

常见的企业采购服务器配置信息：

CPU 型号：Intel(R) Xeon(R) CPU E5-2630 2.60GHz （2 颗 4 核 CPU 或者 2 颗 6 核）。

内存型号：DDRIII 1666MHz 16G、32G、64G 等等。

硬盘型号：希捷 3.5 寸 2T SATA 7500 转速或者希捷 600G SAS 15000 转速。

DELL R720 服务器

在企业系统里面中，通常服务器归类一遍分为两个大类，一类为**应用服务器(例如 nginx、web、LVS、Tomcat 服务器)，和数据库服务器（Mysql、REDIS、Mongodb 等）。**

一般应用服务器硬盘 2x300G SAS，采用 Raid 1 技术。如果三块硬盘创建 RAID 5 技术。这样可以实现冗余，如果数据不重要，也可以采用 RAID 0 技术。

那什么是 Raid 技术呢?

磁盘阵列（Redundant Arrays of Independent Disks，RAID），有“独立磁盘构成的具有冗余能力的阵列”之意。原理是利用数组方式来作磁盘组，配合数据分散排列的设计，提升数据的安全性。

磁盘阵列还能利用同位检查（Parity Check）的观念，在数组中任一颗硬盘故障时，仍可读出数据，在数据重构时，将数据经计算后重新置入新硬盘中。

常见 RAID 特点：

RAID0：读写性能强，没有冗余功能，如果一个磁盘（物理）损坏，则所有的数据都无法使用。

RAID1：磁盘的利用率最高只能达到 50%(使用两块盘的情况下)，是所有 RAID 级别中最低的。

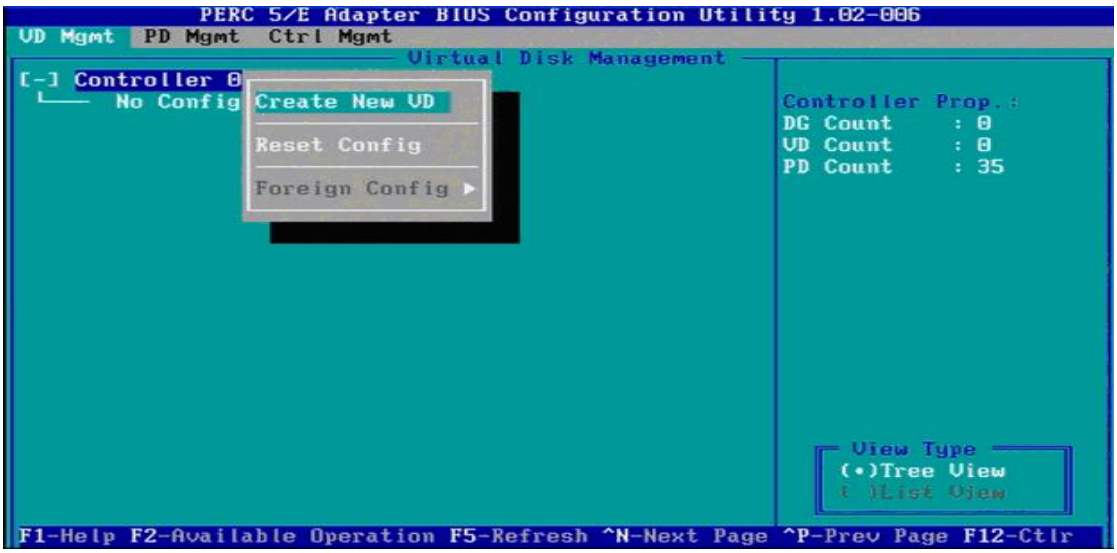
RAID5：奇偶校验码存在于所有磁盘上 ，RAID5 的读出效率很高，写入效率一般 ，磁盘容量为 n-1/n(最低需要三块硬盘)，最多允许坏一块硬盘。

RAID10：以理解为是 RAID 0 和 RAID 1 的折中方案。RAID 0+1 可以为系统提供数据安全保障，但保障程度要比 Mirror 低而磁盘空间利用率要比 Mirror 高。

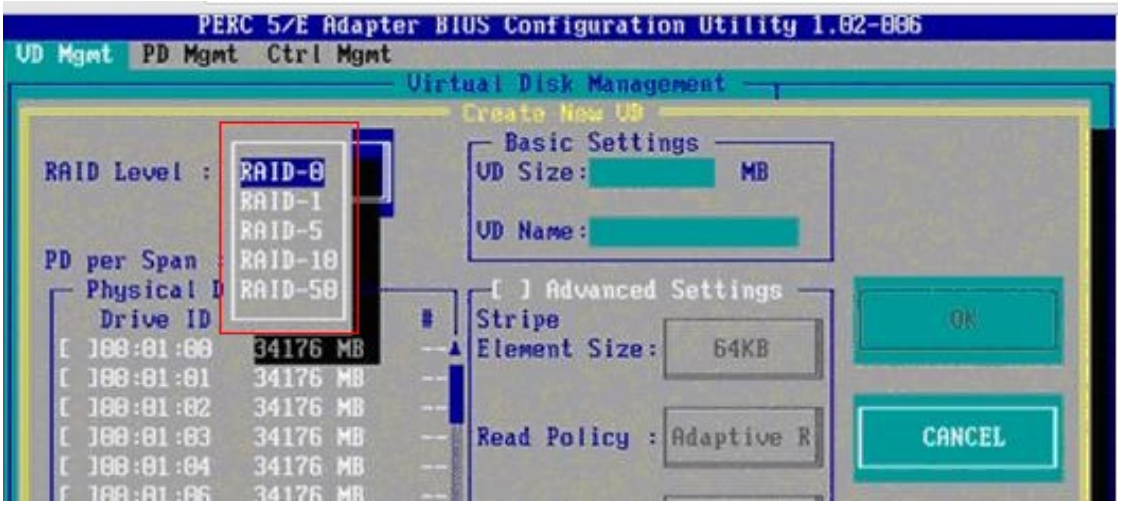
如下图为 DELL R720 制作 RAID 的方法，其他型号同理：

开机启动根据提示按 **ctrl + r** 进入 RAID 配置界面如下：

然后按 F2 或者回车进入选择“Create New VD”-->RAID 界面：



然后按回车可以看到 RAID 卡支持的几种 RAID：



然后按空格键选择下面 6 块磁盘，然后再点击最右边的”ok”即可，如果想划分为多个虚拟磁盘，只需要修改 VD Size 后面的大小就行，常用的企业环境可以这么划分：（超过 2T 磁盘划分如下方式）

VD SIZE1 ： 120G （专门用于安装系统）

VD SIZE2 ： 8653G（分区超过 2T，后面采用 GPT 格式格式化）

最后选择快速格式化即可，然后重启 BIOS，进入系统安装即可。

29.3 企业 Linux 系统规范标准及常见配置

在真实 Linux 运维中，我们每个人都经历过 Linux 系统的安装，安装的方法常用的有两种：光盘安装和 kickstart 自动化安装等。

那系统安装到底有什么标准呢？请让我娓娓道来：

- **系统分区**

- /boot 200M
- / 80G
- Swap 16G

- /data　剩余所有空间大小

➢ **系统软件选择**

选择上海时区--->安装包选择如下：

系统安装包选择，这里选择“现在定制“。

系统安装包选择，左侧选择“开发“----右侧选择”开发工具“和“开发库”，语言选择“支持中文 “，其他一概不选择。

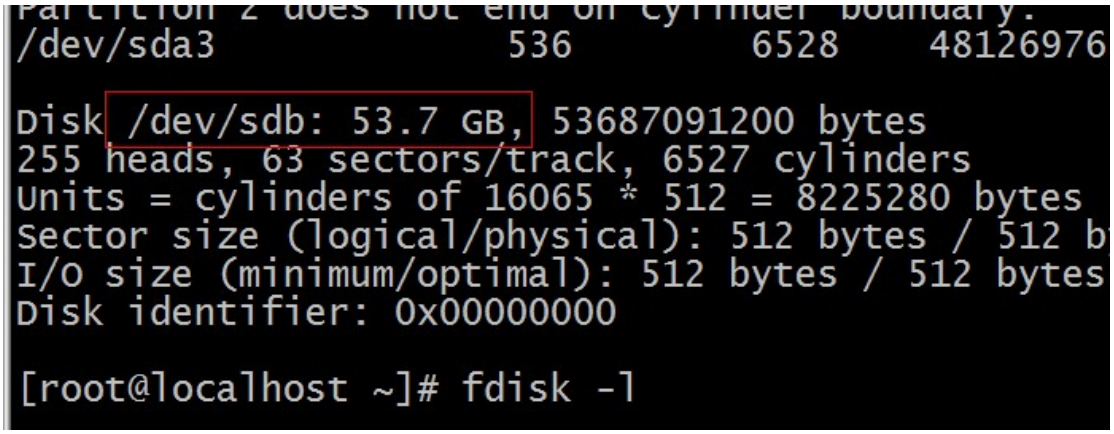
如上安装即可。

- **格式化大于 2T 硬盘**

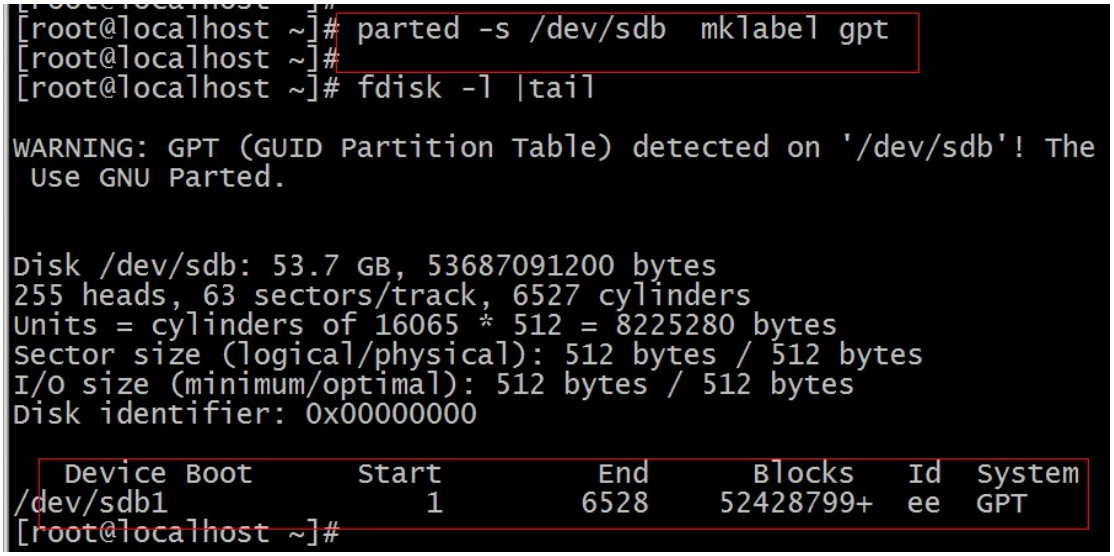
确保超过 2T 的硬盘为单独的磁盘，可以使用如下命令进行格式化：

- parted -s /dev/sdb mklabel gpt
- mkfs.ext3 /dev/sdb
- mount /dev/sdb /data/

如下图假设:/dev/sdb 为 10T 硬盘，我们现在才有 GPT 格式来格式化：



使用命令格式化为 GPT 格式后如下：



然后 mkfs.ext3 /dev/sdb 格式化：

```
[root@localhost ~]# mkfs.ext3 /dev/sdb
mke2fs 1.41.12 (17-May-2010)
/dev/sdb is entire device, not just one partition!
无论如何也要继续? (y,n) y
文件系统标签=
操作系统:Linux
块大小=4096 (log=2)
分块大小=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
3276800 inodes, 13107200 blocks
655360 blocks (5.00%) reserved for the super user
第一个数据块=0
Maximum filesystem blocks=4294967296
```

或者使用如下命令也可以：

parted-->select /dev/sdb---->mklabel gpt--->mkpart primary 0 -1 --->print 打印：

```
[root@localhost ~]# parted
GNU Parted 2.1
使用 /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of c
(parted) select /dev/sdb                  选择/dev/sdb硬盘
使用 /dev/sdb
(parted)
(parted) mklabel gpt                      格式类型为GPT
警告: The existing disk label on /dev/sdb will be dest
lost. Do you want to continue?
是/Yes/否/No? yes
(parted)
(parted) mkpart primary 0 -1              将整块硬盘分为一个分区
警告: The resulting partition is not properly aligned
忽略/Ignore/放弃/Cancel?
忽略/Ignore/放弃/Cancel? ignore
(parted)
(parted) print                            打印我们刚分区的磁盘信息
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 53.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
```

29.4 Linux 运维必备架构分析及制作能力培养

在企业运维中，我们经常会用到架构图，没有架构图，我们不能从直观上去快速了解整

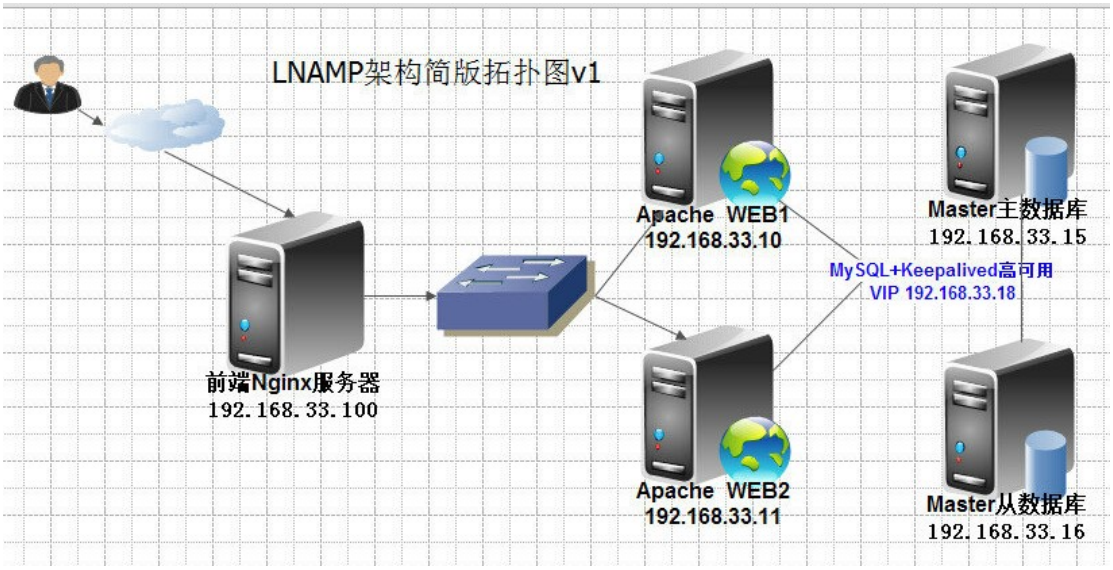
个系统的所有服务器及服务。所以规划和制定架构图是我们每个人运维SA必备的一项技术。

本次课程将给大家来构建一个主流企业完整的架构图，给大家分析架构图上各种含义：

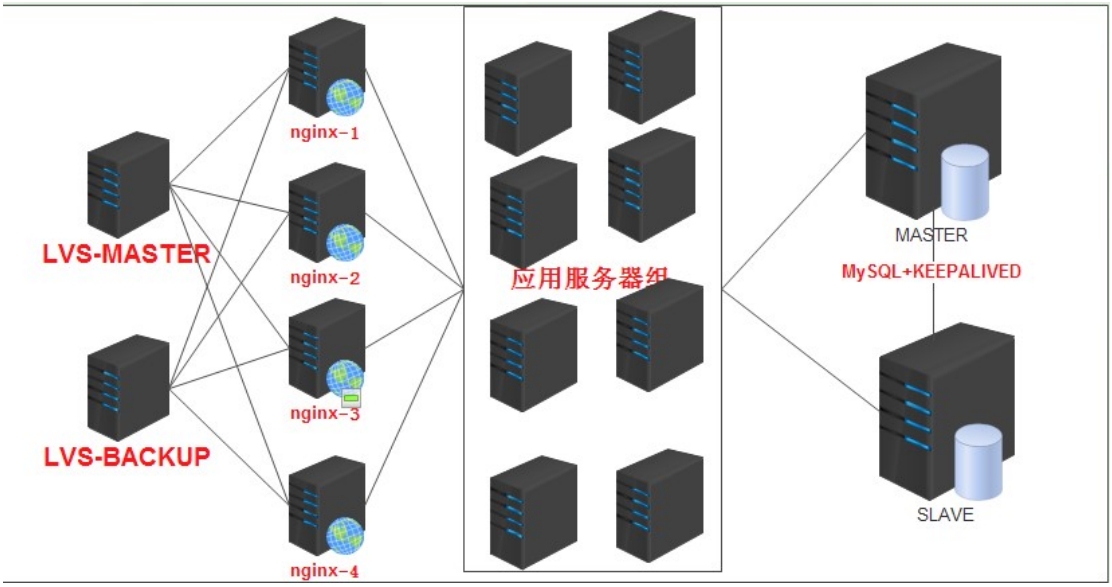
首先我们的架构包含的元素为：

LVS+KEEPALIVED+SQUID+Nginx+Tomcat+MySQL（REDIS/MEMCACHED）等。

如下案例为 LNAMP 简单架构图：（1 台 nginx---2 台 apache--2 台 Mysql 主从）



LVS+KEEPALIVED+SQUID+Nginx+Tomcat+MySQL 架构图如下：



29.5 Linux 运维部文档编写及逻辑总结

在我们日常运维中，编写文档的能力非常重要，所以我们大家在工作和学习中要注重培养相关的能力，这些都是通过不断练习和总结来的，不断去写，才会懂得很多，才能总结出精简同时能表达某个含义的语句。

今天我将跟大家一起来学习写文档，让我们在运维路上知识更加宽广，让我们的能力更加全面：

- **工具准备**
- **内容分解**
- **段落规划**
- **引入目录**
- **全面检查**
- **后期修改**

29.6 Linux 发展路线及面试技巧

29.7 Linux 发展方向及路线

对于从事 Linux 岗位的童鞋们,最关注的问题莫过于这个行业到底怎么样,能不能挣钱?我以后能做什么?

对于第一个问题:

随着互联网飞速的发展,用户对网站体验各方面都要求很高,所以作为网站底层承载的 linux 系统来说,得到大批量的应用,可以说大中型互联网公司 Linux 在服务器领域已经占到 7-80%,而且 Android 手机也是基于 Linux 来研发定制的。未来 Linux 会在各行各业得到普

遍的应用。

这里讨论 Linux 运维，如果是 Linux 开发的话，薪资更高，所以只要你技术熟练、精通，薪资根本不是问题，初级薪资一般都在 4-5K 以上，中间 6-8K，高级 Linux 运维薪资一般都在 10K+。对应 Linux 岗位管理方面薪资则 20K+左右。所有不要再问能给有多少钱，关键是你有多熟练，你的能力在哪里？

从 Linux 运维领域来说，可以努力学习的方向有：

- 熟练 Linux 系统的性能优化、网络日常管理；
- 高性能集群架构部署及优化等；
- 大并发网站运维及管理；
- Mysql、Oracle 数据库集群管理；
- 自动化运维平台开发与管理；
- 网站架构 GSLB、CDN 缓存等。

一个行业要想熟练、甚至精通至少要花上 5-10 年的时间，做一件事重在专一，即使现在不会，只要每天进步一点点，每天实践一点点，改变一点点，相信未来更美好。只有专注才能成功。

29.8 Linux 面试技巧总结

通过全面具体的学习，我们已经正式遨游进入了 Linux 运维世界，接下来我们就需要正式的找一份 Linux 岗位的工作，很多人谈到找工作就害怕，为什么呢，害怕面试不上、面试紧张、知识准备不充分等等。

通过这样一个完整的 Linux 高级运维的学习，我们了解了目前企业里面使用的技术和架构信息，那接下来我们来总结一下企业一般问什么问题？以及面试的过程中要注意哪些细

节?

总结日常面试的技巧（以正式讲课为准）：

1) 首先穿着要得体,最好标准的职业装面试,不能随意穿着;简单一点就是要让人一看你，就感觉清爽、能干、有活力。

2) 要准备充分，尽量提前 15 分钟到面试公司，提前翻阅资料了解公司的简单背景及相关文化。

3) 保持微笑，不要太古板，要随和，保持心态放松，不要抢话抢答；要懂礼貌，有时候细节决定成败。

4) 在回答问题上要简单明了，不要阐述一个问题绕来绕去，把自己都绕迷糊了；要说到恰到好处。该回答的回答，不该说尽量别说，做到有的放矢。

5) 要保持谦逊，遇到不会的题目，不知道就是不知道，不要非懂装懂；

6) 面试通常开始会让你做自我介绍，自我介绍说些什么？多长合适?

一般自我介绍，就介绍自己叫什么名字，毕业时间学校，已经之前工作经验，自己比较熟练的技能和自己的性格和优点等等；注 介绍完毕，最后说声介绍完毕，谢谢。

7) 面试要有自信，不要低着头，面试是双向的，你选择公司，公司也在选择你。机会是非常的多的，关键是看你自己是否能把握住，是否之前已经准备好。

8) 面试的心态一定要保持平静，不要因为一次面试不上，就觉得自己到处都是缺点，要总结自己上次面试的不足，然后下一次改变掉，相信坚持不懈一定能找到满意的工作。

9) 最后总结一点，做什么事情自信很重要，相信自己可以做到，然后勇敢的去做，结果一定让你倍感惊喜。

29.9 Linux 运维面试题目精讲

通过不断的面试，我们会总结到更多的知识和技巧，这里总结一下日常面试到的问题及简单回答方法：

1) 你平时在公司主要做什么？

2) 你们原来公司的网站架构是怎样的？

3) 你对哪一块比较熟练或者精通？

4) Squid、varnish 等缓存服务器维护过吗？squid 缓存代理的原理是什么？缓存命中率怎么查看及清空缓存？

5) LVS 的工作原理是什么？有哪些算法？

6) Nginx 日常的优化的参数都有哪些？Nginx 动静分离做过吗？描述简单的步骤。

7) Linux 内核优化，你都优化哪些参数？

8) 你在维护网站的过程中，曾经遇到过什么重大的问题？怎么解决的？

9) Shell 编程熟练吗？编写一个自动化备份 Mysql 数据库的脚本？

10) Mysql 主从架构的原理是什么？如果主从不同步，报错了，怎么恢复？

11) 如果备份大数据 Mysql 数据文件？Mysql 优化有哪些步骤？

12) FTP 主被动模式的区别是什么？

13) Apache 两种工作模式的区别及优化？

14) Nagios、cacti 维护过吗？平时都监控些什么？

15) 你们公司的网络出口带宽是多少？每天网站的 PV、UV 是多少？

16) 你觉得 Linux 运维工程师的职责是什么？

17) 你为什么离职，离职的原因是什么？

18) 你未来 5-10 年的职业规划是什么样的？

29.10 Linux 企业实战之备技巧

本次课程为额外增加课程，主要是为了给学员更多的指导，让我们的童鞋们在企业中更好的运维，熟悉日常运维的技巧满足我们企业的发展，同时让我们的运维更加的轻松，不再觉得运维是苦逼的活，真正去锻炼去成长去磨练。

让我们的运维更有"韵味",让我们的生活更加美好。 通过本次课程的学习，我会把我在日常运维中的心得和体会分享给大家，让大家得到真正的知识，然后应用在自己的运维工作中。

那我们需要注意什么呢：

1) 我们要明白学习 Linux 运维的目的，相信大家都是为了能找到一份非常好的工作，一个高薪的工作，不断的练习，不断的成长。

通过工作，让我们的生活更加的完整和充实。

2) 在明白自己的大的目标之后，我们需要分解大目标，接下来就是真正去行动，去朝着小目标努力，有哪些小目标呢?

- 计算机基础知识—硬件认识—Windows 系统日常操作—Linux 系统入门；
- Linux 目录及权限学习—linux 必备 20 个命令：

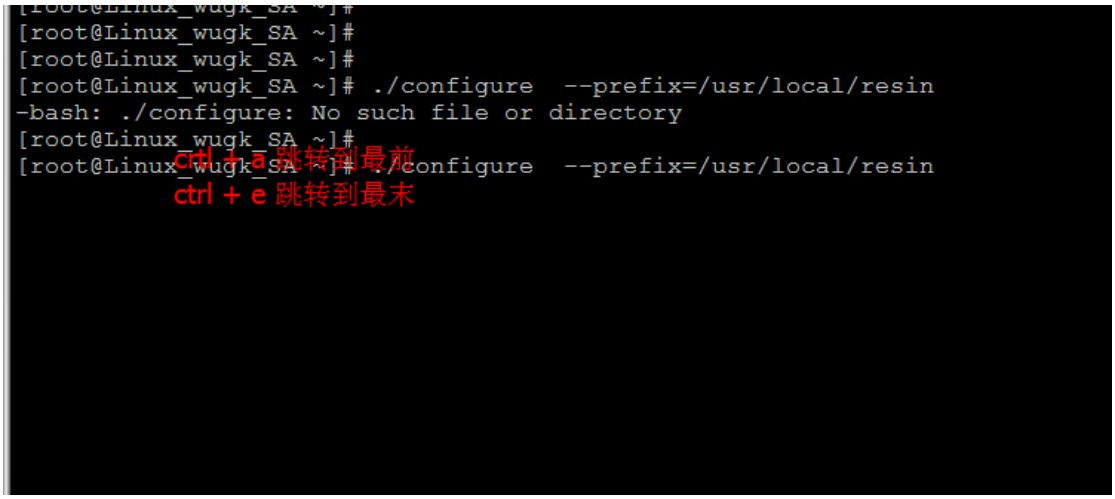
ls pwd cd cat useradd groupadd rm cp chown chmod vi find grep ps free top sed awk if for case wc yum rpm tar unzip more head tail 等。

- Linux 简单服务器搭建 (掌握 tar 常见文件解压方式，掌握安装软件的方法：yum install 方式安装；源码编译安装:./configure、make、make install；
- Apache 服务构建—Mysql 服务搭建—PHP 服务器搭建—LAMP 架构整合 discuz 论坛；

- Kickstart 自动化系统安装--cacti 监控部署—Shell 脚本编程(包括各种语句的学习，if for awk for while sed 等)；
- Linux 高级服务器搭建---Nginx WEB 服务器搭建—Tomcat 服务器搭建—resin 服务器搭建；
- Nginx 均衡 java 服务器—LNMP 架构搭建(yum/源码)--Nginx 动静分离；
- LVS+Keepalived 负载均衡部署---LVS+Keepalived+Nginx+Tomcat 均衡架构部署---高级 Shell 编写；
- 自动化运维学习（KVM、Puppet、ZABBIX、Ansible、Mysql+DRBD 等）

3) 编辑器命令技巧

熟悉命令行及 vi 编辑器的查找，匹配删除、跳转等等，例如在 shell 命令行里 ctrl +a 跳转到最前，ctrl +e 跳转到最末尾。



在 vi 编辑器里面：

Shift + ^跳转到开头，shift + $跳转到末尾。

匹配/word 字符，删除光标所在字符按 x 即可，跳转到文本最末行按 G，跳转到文本首行按 gg。

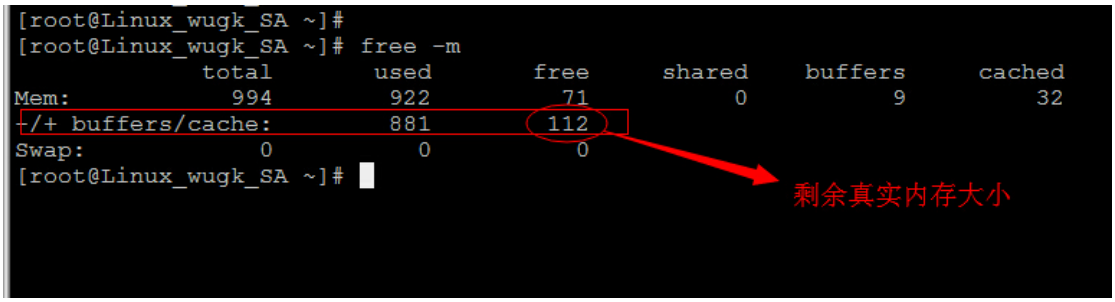
同时删除光标行至文本最后一行：dG

删除光标行至文本第一行：dgg

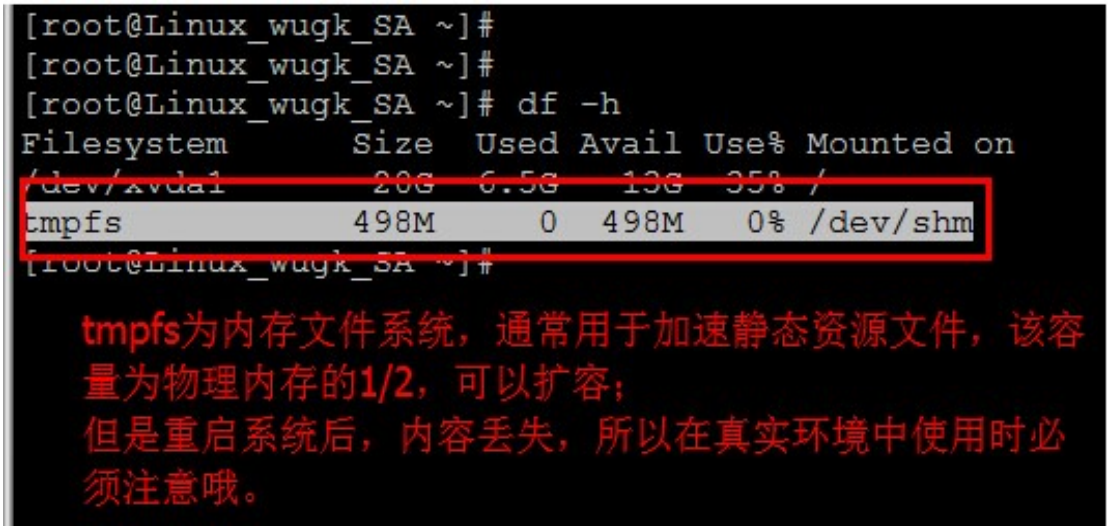
4) 系统运行状态监测

可以使用 free –m 查看内存剩余大小，通常看

-/+ buffers/cache: 881 **112（该值大约为真实内存值）**



可以使用 df –h 查看到 tmpfs 内存文件系统，加速静态文件及图片：



查看本地网卡流量技巧：

Yum install iftop –y

iftop -i eth0 查看结果如下图：

中间<= =>这两个左右箭头，表示的是流量的方向。

TX：发送流量。

RX：接收流量。

TOTAL：总流量。

Cumm：运行 iftop 到目前时间的总流量。

peak：流量峰值。

rates：分别表示过去 2s 10s 40s 的平均流量。

查看磁盘 IO 负载技巧：

vmstat 1 5 (每秒输出结果，总共输出 5 次)

```
[root@Linux_wugk_SA ~]# vmstat 1 5
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0  75400   8068  31844    0    0     8    14    5    3  1  0 99  0  0
 0  0      0  75400   8068  31872    0    0     0    84  227  223  1  0 99  0  0
 0  0      0  75400   8068  31872    0    0     0     0  202  197  0  1 99  0  0
 0  0      0  75400   8068  31872    0    0     0     0  210  216  1  0 99  0  0
 0  0      0  75400   8076  31868    0    0     0    28  189  175  0  0 99  1  0
[root@Linux_wugk_SA ~]#
```

r: 运行队列中进程数量

b: 等待 IO 的进程数量

Memory（内存）：

bi: 每秒读取的块数

bo: 每秒写入的块数

wa: 等待 IO 时间

注意*一般判断系统负载是否过高，IO 磁盘读写是否超高，查看 r、b 和 wa 的时间，当然是越小表是性能资源还有很多剩余，如果过大，我们就需要查看是由于什么操作导致的。

可以结合 iostat 查看更容易判断是不是磁盘读写导致 IO 很高。

```
[root@Linux_wugk_SA ~]#
[root@Linux_wugk_SA ~]# iostat -x 1 5
Linux 2.6.32-431.20.3.el6.x86_64 (Linux_wugk_SA)        11/11/2014      _x86_64_        (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.74    0.00    0.38    0.19    0.00   98.69

Device:         rrqm/s   wrqm/s     r/s     w/s   rsec/s   wsec/s avgrq-sz avgqu-sz   await  svctm  %util
xvda              0.03     1.76    0.66    1.65    16.53    27.33    18.94     0.04   16.18   1.01   0.23

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          1.01    0.00    0.00    0.00    0.00   98.99

Device:         rrqm/s   wrqm/s     r/s     w/s   rsec/s   wsec/s avgrq-sz avgqu-sz   await  svctm  %util
xvda              0.00     0.00    1.01    0.00    16.16     0.00    16.00     0.01    5.00   5.00   0.51

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          1.00    0.00    1.00    0.00    0.00   98.00
```

一般判断%util 的值，如果持续超过 75%以上就需要注意了，检查相关服务的访问是否异常，然后去一一解决。

服务后台启动：

常见的程序放在后台运行方法主要有：

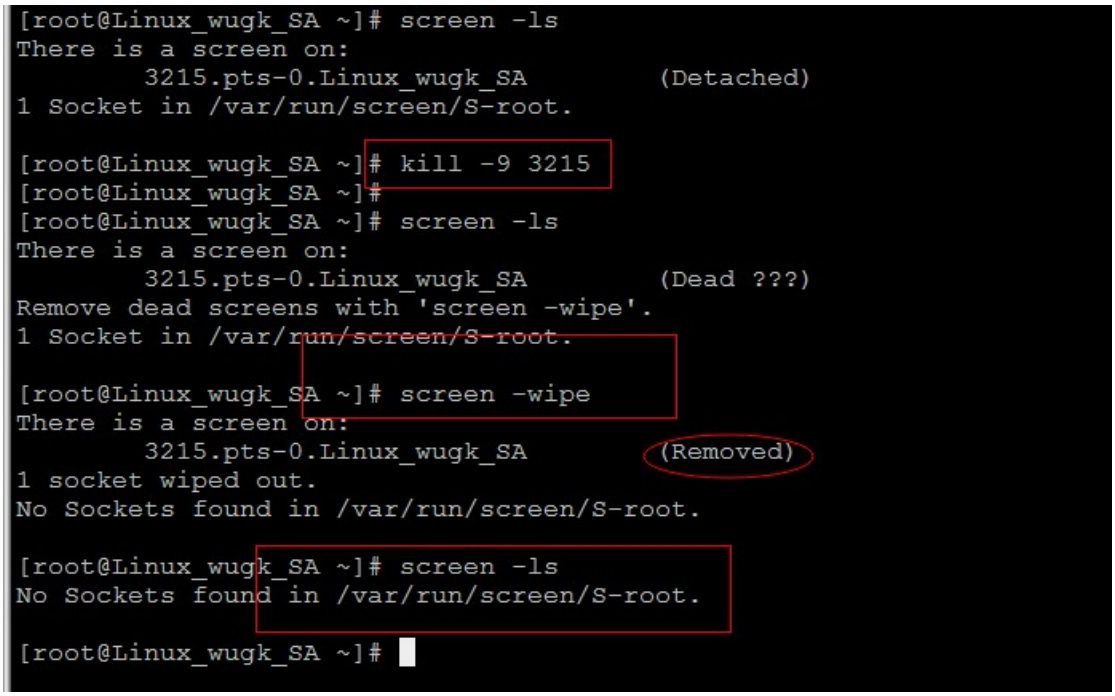
screen 后台运行：

在命令行执行 screen 回车，进入一个随机的 screen 后台，可以输入命令，然后按 ctrl +a+d 保存退出即可，这时程序已经在后台运行。

Screen –ls 可以查看当前运行 screen 后台列表,执行 screen –r 加 PID 可以进入相应的后台，再次退出还需要按 ctrl+a+d

```
[root@Linux_wugk_SA ~]#
[root@Linux_wugk_SA ~]# screen -ls
There is a screen on:
        3215.pts-0.Linux_wugk_SA        (Detached)
1 Socket in /var/run/screen/S-root.

[root@Linux_wugk_SA ~]# screen -r 3215
```

如何想要删除 screen，需要执行 kill -9 3215 ，然后执行 screen –wipe 即可删除。



除了 screen 之外，我们还可以使用 nohup 来后台运行程序：

nohup sh auto_nginx.sh & 即程序已经在后台运行，可以在当前目录查看 tail –fn 10 nohup.out 可以看到程序执行的相关信息，如果需要结束就直接 kill 进程就 OK。

最后祝愿大家早日找到合适的工作，在工作中不断成长，每个人都能独当一面。