

# LINUX 基础

## Linux文件基本操作

### 终端和控制台的区别

- 区别标志：
  - **直接交互的标志**：控制台通常是指提供直接与操作系统交互的界面，尤其是在系统级别上，如直接查看和管理系统启动过程、内核消息等。
  - **远程和非系统级交互**：相比之下，终端可以是本地的也可以是远程的，提供用户级别的命令行界面，用于执行各种命令和程序，但不一定提供直接的系统级别交互。
- 自我理解：
  - 如果我使用vmware安装了一个linux系统的虚拟机，那么可以说，直接在vmware上打开这个虚拟机的界面是控制台，而ssh链接这个虚拟机的远程链接界面是终端，但不是控制台，控制台和终端的区别标志是是否直接与系统交互
- Linux中终端是一个设备文件，本质就是一个文件，位置在/dev/pts/0 (终端编号，从0开始)
- 查看当前终端设备

```
tty
```

### 登录前提示

```
/etc/issue # 里面的内容会在登录前出现在终端加载页面
```

### 编辑用户展示公告（登录后提示）

- 概述：
  - /etc/motd 文件的全称是 "Message of the Day"（当日消息）。在Linux和Unix系统中，这个文件的内容会在用户登录到系统后显示在他们的终端上。这为系统管理员提供了一个简单的方法来为登录到系统的用户展示公告、重要消息或任何其他信息。
  - 配置文件

```
/etc/motd
```

- 在VIM中输出彩色字体

- 步骤：

- 打开一个新的或现有的 vim 编辑器：vim filename
- 按下 i 进入插入模式。
- 按 Ctrl+v 进入 vim 的特殊字符插入模式。
- 接着按 Ctrl+[。这会插入一个 ^[，这实际上是一个转义字符。
- 然后直接输入 [32;1m。
- 继续输入你的文本，如：Hello, welcome To My Home - Mystical。
- 然后再次按 Ctrl+v 和 Ctrl+[ 插入另一个转义字符。
- 接着输入 [0m 来结束彩色输出。

- 示例：

```
^[[32;1mHello, welcome To My Home - Mystical^[[0m
```

## Prompt提示符

# 格式如下:

```
PS1="提示符格式命令"
```

# 格式命令如下:

```
\[ <提示符样式> \] # 这个提示符样式外的\[ \] 可以省略
```

# 提示符样式如下:

```
\e[ 样式 + 内容 \e[0m # 以 \e[ 开始, \e[0m 表示关闭设置
```

# 样式分3部分

格式: F;B;Sm

F: 表示前景色

30 黑色, 31 红色, 32 绿色, 33 黄色, 34 蓝色, 35 紫色, 36 青色, 37 白色

B: 表示背景色, 背景色数值 = F + 10

S: 显示的样式如下

0: 表示关闭颜色设置

1: 表示粗体

4: 表示加下划线

5: 表示闪烁

7: 表示前景色, 背景色互换

8: 表示隐藏

# 内容:

\u: 表示当前用户

\h: 表示主机名简称

\w: 表示当前工作目录基名

\\$: 表示提示符(普通用户 \$ root用户#) 实测在rocky9不好用

\H: 表示完整主机名

\w: 表示完整工作路径

\t: 表示24小时时间格式为: 时: 分: 秒

\A: 表示24小时时间格式为: 时: 分

\#: 表示开机命令历史数

\d: 表示日期, 格式为: 星期 月 日

\v: 表示BASH的版本信息

示例:

```
PS1="\e[32;40;1m[\d \t \e[31;40;1m\#] \e[33;40;1m\u@\h:\w \$\e[0m"
```

注意:

实测\\$不好用, 可以直接在root的目录下的.bashrc文件中改为#

```
PS1="\e[32;40;1m[\d \t \e[31;40;1m\#] \e[33;40;1m\u@\h:\w \$\e[0m"
```

注意:

实测\\$不好用, 可以直接在root的目录下的.bashrc文件中改为#

- 使其永久生效, 可以在下列文件中修改配置项

```
/etc/profile.d/env.sh #针对CentOS生效  
  
/usr/share/bash-completion/bash_completion  
# 写在该文件最下方，对所有普通用户生效  
  
~/.bashrc #针对当前目录的提示符效果生效
```

## 查看用户登录信息

- whoami命令：显示当前登录有效用户
- who命令：显示当前所有的登录会话
- w：显示系统当前所有的登录会话及其所做的操作

```
[Fri Oct 13 22:16:15 59] root@rocky9:/ #w  
22:20:10 up 1 day, 4:39, 2 users, load average: 0.00, 0.00, 0.00  
USER TTY LOGIN@ IDLE JCPU PCPU WHAT  
root pts/0 21:40 0.00s 0.07s 0.00s w  
root pts/1 21:56 7:22 0.03s 0.01s vim ps_demo.txt
```

## 操作系统版本

- 查看操作系统版本的配置文件

```
/etc/os-release # 操作系统详细信息  
  
/etc/redhat-release # 操作系统版本
```

- 扩展
  - 可以在字符串中直接添加shell命令运行结果

```
echo `cat /etc/redhat-release` is os-version
```

## 查看硬件信息

### 查看CPU

- command

```
lscpu  
  
cat /proc/cpuinfo
```

## 查看内存大小

```
lsmem  
  
free -h  
free -h -s 1 # 每秒更新一次内存数据  
free -h -c 2 # 总共更新几次，默认一秒一次  
  
cat /proc/meminfo
```

## 查看硬盘及分区情况

```
lsblk  
  
cat /proc/partitions
```

## 查看系统版本信息

### 查看系统架构

```
arch
```

### 查看内核版本

```
uname -r
```

### 查看操作系统发行版本

```
# CentOS  
cat /etc/redhat-release  
  
cat /etc/os-release  
  
lsb_release -a  
  
#Ubuntu  
cat /etc/os-release  
  
cat /etc/issue
```

## 查看日期时间

### 系统时间

```
date # 查看系统时间  
  
date -R # 显示时区信息  
  
date +%s # 显示时间戳（从1970年1月1日到当前时间，经过的秒数）  
  
date +"%F %T" # 时间戳格式化，年月日时分秒
```

```
# -d的基本用法:  
# date -d "string"  
  
date -d "yesterday"  
date -d "now"  
date -d "next friday"  
date -d "last month"  
date -d "2 weeks ago"  
# 格式化输出  
date -d "2024-01-01" +"%Y-%m-%d"  
date -d "next year" +"%A, %d %B %Y"  
# 时间计算  
date -d "-3 day" +%F
```

## 硬件时间

### - 主板上BIOS的时间

```
clock # 显示系统时钟  
  
hwclock # 显示硬件时钟
```

#### • 硬件时钟和系统时钟的区别和含义

##### ◦ 硬件时钟 (Real-Time Clock, RTC) :

###### ▪ 物理设备:

硬件时钟是计算机主板上的一个实际的物理设备，有时被称为 CMOS 时钟。

###### ▪ 独立供电:

它通常由一个小电池供电，这意味着即使计算机断电或关闭，硬件时钟也会继续运行。

###### ▪ 持久性:

硬件时钟保存了日期和时间信息，并在系统启动时提供给操作系统。这个时间通常在计算机启动时由 BIOS 或 UEFI 读取。

###### ▪ 精度:

硬件时钟的精度相对较低，可能会因电池老化或其他原因逐渐偏离准确时间。

## 对钟

```
hwclock -s | --hctosys # 以硬件时钟为准，校正系统时间  
  
hwclock -w | --sysowc # 以系统时钟为准，矫正硬件时间
```

## 设置时区

```
timedatectl list-timezones # 列出所有时区  
  
timedatectl set-timezone <时区> # 设置时区  
# 示例: timedatectl set-timezone Asia/Shanghai
```

## 显示日历

```
cal  
  
cal 2 2024 # 显示指定月份日历  
  
cal 2024 # 显示指定年份的12个月的所有日历
```

## 关机与重启

- 关机

```
halt  
poweroff  
init 0  
shutdown -h now
```

- 重启

```
reboot  
init 6  
shutdown -r now
```

- shutdown

```
shutdown      # 一分钟后关机  
shutdown +10   # 十分钟后关机  
shutdown 01:02 # 1点过2分关机  
shutdown -r|--reboot # 一分钟后重启  
shutdown -r now # 现在重启  
shutdown -H|--halt # 一分钟后调用halt关机  
shutdown -P|--poweroff # 一分钟后调用poweroff关机  
shutdown -c      # 取消关机计划
```

## 会话管理

- screen
- Tmux
  - Tmux安装

```
# 软件安装  
# Mac  
$ brew install tmux  
  
# Ubuntu 或 Debian  
$ sudo apt-get install tmux  
  
# CentOS 或 Fedora可以使用yum/dnf/brew等方式安装, brew版本更高些  
$ yum install tmux  
  
# 下载并加载字体  
$ git clone https://github.com/powerline/fonts.git --depth=1
```

```
$ cd fonts
$ ./install.sh
$ cd ..
$ rm -rf fonts

# 安装风格包
# Clone项目代码
$ git clone https://github.com/odedlaz/tmux-onedark-theme

# 删除原~/.tmux.conf 文件
$ rm ~/.tmux.conf

# 安装 tpm (opens new window)与tmux-resurrect (opens new window)插件
$ git clone https://github.com/tmux-plugins/tpm ~/.tmux/plugins/tpm
$ git clone https://github.com/tmux-plugins/tmux-resurrect
~/tmux/plugins/resurrect

# 然后新建 ~/.tmux.conf 文件添加以下内容然后新建 ~/.tmux.conf 文件添加以下内容
run-shell ~/tmux-onedark-theme/tmux-onedark-theme.tmux

#set -g @onedark_widgets "Mystical Recluse #(ip)"
set -g @onedark_widgets "Mystical #(ip)"
set -g @onedark_time_format "%I:%M"
set -g @onedark_date_format "%m:%d"
#set -g @onedark_date_format "%m:%d"
set-option -g default-terminal "screen-256color"
set -g default-terminal "screen-256color"

# 解决neovim中esc响应慢
set -s escape-time 0
set-option -g status-position bottom

# 自动保存会话
set -g @plugin 'tmux-plugins/tpm'
set -g @plugin 'tmux-plugins/tmux-sensible'
set -g @plugin 'tmux-plugins/tmux-resurrect'
set -g @plugin 'tmux-plugins/tmux-continuum'
set -g @continuum-save-interval '15'
set -g @continuum-restore 'on'
set -g @resurrect-capture-pane-contents 'on'
run '~/tmux/plugins/tpm/tpm'

run-shell ~/.tmux/plugins/resurrect/resurrect.tmux

# 解除默认前缀
unbind C-b
# 设置自定义前缀
set -g prefix C-f
# 采用vim的操作方式
setw -g mode-keys vi
# 窗口序号从1开始计数
set -g base-index 1
# 开启鼠标模式
set-option -g mouse on

# 通过前缀+KJHL快速切换pane
```

```
#up  
bind-key k select-pane -U  
#down  
bind-key j select-pane -D  
#left  
bind-key h select-pane -L  
#right  
bind-key l select-pane -R
```

- tmux常用热键

```
# 新建会话  
tmux new -s hdcms  
# 查看会话  
ctrl+b s  
# 重命名会话  
Ctrl+b $  
  
# 创建窗口  
ctrl+b c  
# 切换到2号窗口  
ctrl+b 2  
# 重命名窗口  
ctrl+b ,  
# 关闭窗口  
ctrl+b &  
  
# 水平拆分出一个新窗格  
ctrl+b %  
# 垂直拆分窗格  
ctrl+b "  
# 切换到下一个窗格  
ctrl+b o  
# 关闭窗格  
ctrl+b x
```

## 显示模式切换

- 查看显示模式

```
runlevel
```

- 切换显示模式

```
init 3 # 切换到字符界面  
  
init 5 # 切换到图形化界面
```

## 设置主机名hostname

- 配置文件

```
/etc/hostname
```

- 注意：主机名不要使用下划线

```
# 临时设置主机名  
hostname 新主机名
```

- ubuntu直接通过命令修改hostname配置文件

```
hostnamectl set-hostname ubuntu1804.magedu.org
```

- 在不重启的情况下，是hostname在prompt上生效，最快的方法是开启一个新的终端会话

## 显示字符echo

- 说明：echo会将字符串显示在标准输出即屏幕上。
- 语法：echo [SHORT-OPTION] [STRING]
- SHORT-OPTION说明：
  - n: 不自动附加换行符；即都在一行显示
  - e: 启用转义符，能使用转义符\
- 注意：echo后面的字符串建议用单引号括住

```
echo -e hello\nworld # 显示hellonworld  
echo -e 'hello\nworld' # 此时转义符生效，符合预期
```

## 命令分类

- 判断命令的类别

```
# 使用type命令
```

```
[Sat Oct 14 08:04:11 4] root@rocky9:~ #type type  
type is a shell builtin  
# 出现这个提示，即可判断该命令是内部命令，type本身也是一个内部命令
```

```
[Sat Oct 14 08:04:21 5] root@rocky9:~ #type hostname  
hostname is /usr/bin/hostname  
# 上述提示，即可判断hostname是外部命令
```

- 内部命令

- 概述：指集成在特定shell中的命令，当用户登陆时，会自动启用shell，而对应的shell程序中包含一些常见工具。默认的/bin/bash shell中就集成了很多内部命令，可以通过enable命令查看所有内部命令

```
# 通过enable命令查看所有内部命令
```

```
[Sat Oct 14 08:07:33 8] root@rocky9:bin #enable
enable .
enable :
enable [
enable alias
enable bg
enable bind
enable break
enable builtin
enable caller
enable cd
enable command
enable compgen
enable complete
enable compopt
enable continue
enable declare
enable dirs
enable disown
enable echo
enable enable
enable eval
enable exec
enable exit
enable export
enable false
enable fc
enable fg
enable getopt
enable hash
enable help
enable history
enable jobs
...
# enable -n <指定命令>
# 禁用指定内部命令, enable -n是查看禁用的内部命令列表
```

## 扩展问题

- echo是shell内置命令，但是/usr/bin/echo，也存在独立与shell的外部命令echo，存在这个现象的原因

独立的非内置的 `echo` 存在有几个原因，主要是为了提供兼容性、标准化以及在特定情况下的灵活性。以下是一些具体的原因：

### 1. 标准化和兼容性

**POSIX 标准：**`/usr/bin/echo` 程序通常会遵循 **POSIX** 或其他相关标准，确保在不同的 **Unix-like** 系统上有一致的行为。

**脚本兼容性：**在编写可移植的 `shell` 脚本时，依赖于标准的 `/usr/bin/echo` 可能比使用特定 `shell` 的内置 `echo` 更可靠。

### 2. 独立于 `shell`

**不依赖于特定的 `shell`：**独立的 `echo` 可以在没有启动 `shell` 的情况下直接执行，这对于某些脚本或系统程序很有用，尤其是在资源有限或需要避免启动完整 `shell` 会话的环境中。

### 3. 特定功能的需要

**功能差异：**不同的 `shell` 可能在其内置的 `echo` 中实现不同的功能或选项。独立的 `/usr/bin/echo` 程序提供了一个统一的选项集，这可能在跨不同 `shell` 环境工作时非常有用。

### 4. 历史原因

**遗留系统：**在早期的 **Unix** 系统中，很多现在被认为是内置命令的工具最初是作为独立的程序存在的。随着时间的推移，为了性能和便利性，许多这样的程序被内置到了 `shell` 中。但是，独立的版本仍然被保留，以保证兼容性和符合标准。

## POSIX标准

**POSIX** 标准的主要内容包括：

**系统调用和库：**定义了一系列的 **API**，包括文件操作、进程控制、线程管理、输入/输出等。

**Shell 和工具：**规定了 `shell` 的行为和一些基本的命令行工具，如 `ls`、`echo` 和 `grep` 等。

**程序编程接口（API）：**为 `C` 语言提供了一套标准的库函数接口，用于实现系统服务。

**用户和软件环境：**包括了用户环境的配置、软件包管理等方面的标准。

**POSIX** 标准的意义：

**兼容性：****POSIX** 标准化了 **UNIX** 系统的核心接口，使得开发者可以编写可在不同 **UNIX** 系统之间移植的程序。

**一致性：**通过遵循 **POSIX** 标准，操作系统厂商可以确保他们的系统提供一致的行为和服务。

**可移植性：**对于软件开发者来说，**POSIX** 提供了一套稳定的、不依赖于特定系统的接口，大大提高了代码的可移植性。

在实际应用中，虽然大多数类 **UNIX** 系统都遵循 **POSIX** 标准的大部分内容，但很少有系统是完全符合所有 **POSIX** 规范的。许多系统提供了超出 **POSIX** 标准的额外功能和扩展，但核心接口和服务通常保持一致。因此，**POSIX** 标准是理解和使用 **UNIX** 系统的基础，并且对于确保不同系统之间软件的兼容性和可移植性至关重要。

## 指令执行过程

1. 先判断是内部命令还是外部命令
2. 如果是内部命令：直接执行

如果是外部命令：先去hash里找，是否有该命令记录，如果没有，去PATH路径下找，如果还没有，则报错，**command not found**；如果找到，则直接执行，并将可执行文件的路径记录到hash中

3. 如果hash中有该指令路径，但是该指令路径已经转移，即使转移到了PATH路径下，仍然会报错，不存在该文件/目录，此时应清空hash值，重新执行指令

4. 清空hash值的方法

更新（更改）PATH路径：会自动清空hash记录

bash: hash -r

    hash -d <指定路径>

    hash -l 查看hash表详细数据

csh: rehash

- 外部命令

- 概述：所谓外部命令，就是没有集成在shell程序中。具体表现为一个独立的可执行文件。所以外部命令都能在磁盘中找到对应文件

- 系统查找外部命令的方式：

- 在系统中有一个叫PATH的变量，里面保存外部命令存放的路径。具体路径可以通过下面命令查看，当执行一个外部命令时，系统会按PATH中存放的目录路径顺序来查找，一旦在某个目录中找到，就停止继续往下找，并执行此外部命令

```
[Sat Oct 14 08:15:25 9] root@rocky9:bin #echo $PATH  
/root/.local/bin:/root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
```

- 查看外部命令存放路径

- which

- whereis：除了命令外，还显示和命令相关的帮助文档等文件路径

```
[Sat Oct 14 08:21:50 29] root@rocky9:bin #which gcc  
/usr/bin/gcc
```

```
[Sat Oct 14 08:23:59 30] root@rocky9:bin #whereis gcc  
gcc: /usr/bin/gcc /usr/lib/gcc /usr/libexec/gcc /usr/share/man/man1/gcc.1.gz  
/usr/share/info/gcc.info.gz
```

- which和where命令的区别

which命令找到相关的二进制程序是否已经在搜索路径中

whereis， 该命令会搜索shell的搜索路径之外更大范围的系统目录

- 当第一次执行外部命令后，系统会自动将外部的路径记录到内存缓存区中，下次再执行此外部命令，将会从缓存区中找到路径，直接到对应的磁盘路径找到此命令并执行。通过hash命令可以查看到已执行过的外部命令及路径

```
[Sat Oct 14 08:24:24 31] root@rocky9:bin #hash
hits      command
7        /usr/bin/ls
3        /usr/bin/whereis
```

- shell 的 hash 表机制主要用于跟踪和缓存 \$PATH 环境变量指定的目录中找到的命令的位置。  
非\$PATH记录的路径下的程序，执行后也不会记录在hash中
- 别名
  - 概述：所谓别名，就是将一些常用的内部或外部命令，起一个较短的名称，这样每次执行这些常用命令时，就可以用别名替代
  - 管理和查看别名

```
# 查看所有别名
$ alias

# 查看指定别名
$ alias 别名

# 定义别名
$ alias 别名="命令"

# 查看指定别名
$ alias 别名

# 定义别名
$ alias 别名="命令"

# 取消别名
$ unalias 别名
```

- 上述命令都是使别名临时生效，如果要使别名永久生效，需要写入配置文件(.bashrc)中
  - 仅对当前用户有效，写入 ~/.bashrc
  - 对所有人有效，写入 /etc/.bashrc
  - 启用配置文件
    - source 文件名 或 . 文件名
- 执行和别名相同的命令时，需要 \别名 或 !别名！
- 执行和别名相同的命令时，需要 \别名 或 !别名！或 "别名" 或 command 别名

## 命令的使用帮助

- Whatis-查看命令简要说明
  - 概述：Whatis可以快速查看到命令或相关内容的简短功能
  - 注意：在使用 whatis 之前，需要先使用 mandb 创建数据库
- 内部命令的使用帮助
  - help COMMAND
- 外部命令的使用帮助

- `COMMAND --help | COMMAND -h`
- man帮助
  - 大部分命令对应的手册通常存储在`/usr/share/man`里，几乎每个外部命令都有man手册
  - man命令的配置文件
    - `/etc/man_db.conf`
  - man命令的语法格式

```
man 命令语法格式;
man [section] WORD
```

格式说明：

**section:** 表示1-9的章节数

1	-	用户命令
2	-	系统调用
3	-	C库调用
4	-	设备文件及特殊文件
5	-	配置文件格式
6	-	游戏
7	-	杂项
8	-	管理类命令
9	-	Linux内核API

**WORD:** 查看帮助的关键字，如：命令，文件名，函数名

```
man -f COMMAND
```

# 如果有多个相同的命令的话，可以使用-f分别进行查看  
# 比如查看C语言的printf和bash命令的printf

## Linux目录结构

- 文件系统的目录结构
  - bin: 给普通用户使用的工具
  - boot: 开启启动的文件，包含linux内核
    - linux内核：`linux-5.14.0-284.11.1.el9_2.x86_64`

**WORD:** 查看帮助的关键字，如：命令，文件名，函数名

```
man -f COMMAND
```

# 如果有多个相同的命令的话，可以使用-f分别进行查看  
# 比如查看C语言的printf和bash命令的printf

```
man -k [keyword]
```

# 查找man手册里的关键词

- 使用彩色man手册

```
sudo apt install most # 使用most打开man
.bashrc配置文件中，添加
export MANPAGER="most -s"

# 之后执行source ./bashrc
```

# Linux目录结构

- 文件系统的目录结构
  - bin: 给普通用户使用的工具(二进制可执行文件)
  - boot: 开启启动的文件, 包含linux内核
    - linux内核: `vm1linuz-5.14.0-284.11.1.el9_2.x86_64`
    - grub, 开机引导加载程序
  - dev: 硬件设备, 比如: 硬盘
  - etc: 类似于注册表, 核心! 各种配置文件
  - home: 用户的数据, 各个用户在家目录
  - root: root用户的家目录
  - run: 运行过程中生成的临时文件
  - sbin: 给管理员使用的工具
  - sbin: 给管理员使用的工具 (二进制可执行文件)
  - tmp: 临时文件
  - usr: 操作系统下自带的文件, 大多在usr
  - var: 网页文件, 日志等不断会变化的文件
  - lib/lib64: 库文件, 很多应用程序共同依赖的库文件
  - mnt/media: 实现外围设备的挂载用的
  - proc/sys: 内存中的数据
  - opt/srv: 外部下载的一些程序软件, 如果不下载的话, 一般为空
  - proc/sys: 内存中的数据, 虚拟文件系统, 内存映射到硬盘的数据
  - opt: 外部下载的一些程序软件, 如果不下载的话, 一般为空
  - srv: 系统上运行的服务用到的数据

## 文件类型

- 概述:
  - 磁盘中存放的每个文件可以分为两个部分
    - 一部分为文件的内容: 即文件的数据部分, 此部分内容存放在磁盘中专门的数据空间(data block)中。
    - 一部分为文件的属性信息, 即元数据(meta data), 比如: 文件的大小, 类型, 节点号, 权限, 时间等, 此部分内容存放在磁盘中专门的节点空间(inode block)中
- 普通文件 (白色)
  - 纯文本文件:
    - `ls -l /etc/issue`
  - 二进制可执行文件 (绿色):
    - 概述: 二进制可执行文件是有特殊格式的可执行程序, 其文件内容表现为不可直接读懂的字符, 用cat查看, 会出现乱码。在Linux中有很多二进制可执行文件, 比如很多的外部命令都是二进制可执行文件
    - `ls -l /bin/cat`

- 数据格式文件
  - 概述：数据格式文件是一些程序在运行过程中需要读取的存放在某些特定格式的数据文件，比如：图片文件，压缩文件，日志文件。通常需要特定的工具打开
  - 举例：用户登录时，系统会将登录的信息记录在/var/log/wtmp文件中，这个就是一个数据文件。需要使用last命令打开此文件查看内容
    - `ls -l /var/log/wtmp` → last (直接在/var/log目录下使用last命令)
- 目录文件（蓝色）
  - 概述：目录文件即文件夹，通过ls -l查看文件属性时，第一个属性表现为d
- 链接文件（浅蓝色）
  - 概述：即将两个文件建立关联关系，这种操作实际上是给系统中已有的某个文件指定另外一个可用于访问它的不同文件名称。
    - `ls -l`查看文件属性时，第一个属性表现为l
  - 分类：
    - 硬链接
    - 软链接
    - 关于硬链接和软链接的区别和定义，后面详解
- 管道文件（暗黄色）
  - 概述：管道pipe文件是一种特殊的文件类型，其本质是一个伪文件（本质是内核缓冲区）。其主要目的是实现进程间通讯的问题。由于管道文件是一个与进程没有“血缘关系”的，真正独立的文件，所以它可以在任意进程之间实现通信。
  - 局限性：
    - 自己写的数据不能自己读
    - 数据一旦被读后，便不在管道中存在，不可反复读取
    - 管道采用半双工通信方式
  - `ls -l`查看文件属性时，第一个属性表现为p
  - FIFO：队列的数据结构，先进先出
  - 更多细节后续详解
- 字符设备文件（明黄色）
  - 通常是一些串行接口设备在用户空间的体现，像键盘、鼠标。字符设备是按字符为单位进行输入输出的，且按一定的顺序进行
  - `ls -l`查看文件属性时，第一个属性表现为c
  - 举例：我们登录到Linux主机，系统会提供一个终端文件tty供我们登录。
- 块设备文件（明黄色）
  - 块文件设备，就是一些以“块为单位”，如：4096个字节，访问数据，提供随机访问的接口设备，例如磁盘、硬盘、U盘
  - `ls -l`查看文件属性时，第一个属性表现为b
- 套接字文件（粉色）
  - 概述：数据接口文件，通常被用在基于网络的数据通讯使用。
  - 当两个进程在同一台主机上，但是像通过网络方式通信，可基于socket方式进行数据通信，可基于全双工方式实现，即可支持同时双向传输数据。

- `ls -l` 查看文件属性时，第一个属性表现为s

## 管理目录类文件相关命令

### 文件类型颜色的配置文件

```
# CentOS  
/etc/DIR_COLORS
```

# Ubuntu

ubuntu 中与颜色设置相关的文件和命令包括：

`~/.dircolors` 或 `~/.dir_colors`:

用户级别的配置文件。如果存在，`dircolors` 命令会使用这个文件中的配置。如果你想定制自己的颜色配置，可以在你的用户目录中创建这个文件。

`/etc/dircolors`:

系统级别的默认配置文件。这个文件可能在某些系统中不存在，或者命名可能有所不同。

`dircolors` 命令：

这个命令用于初始化颜色配置。它会检查 `~/.dircolors` 或 `~/.dir_colors` 文件，如果这些文件不存在，它会使用默认的颜色配置。你通常会在你的 shell 初始化文件中（比如 `~/.bashrc`）看到类似于以下的命令：

```
test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
```

如果你想要调整 Ubuntu 中 `ls` 命令输出的颜色，你可以创建或编辑 `~/.dircolors` 文件，并在该文件中定义你的颜色配置。然后，确保你的 shell 初始化文件（如 `~/.bashrc`）中包含处理 `dircolors` 的命令。这样，每次你打开一个新的 shell 时，都会应用这些颜色设置。

## 管理目录类文件相关命令

### • 查看当前目录

- 命令：`pwd`

```
pwd -P # 输出真实物理路径  
pwd -L # 默认，输出链接路径
```

### • 基名与文件名

```
basename <dir> # 只输出文件名  
# 示例：  
basename `which cat`  
  
dirname <dir> # 只输出路径  
# 示例：  
dirname `which cat`
```

### • 路径间移动

- 命令：`cd`

```
cd -P # 移动到真实物理路径
# 示例
cd -P /bin # 实际移动到/usr/bin

cd -L # 默认, 移动到链接路径

cd ~ # 移动到家目录
cd ~username # 移动到指定用户的家目录

cd - # 移动到上次所在的目录, 之所以能移动到上次所在目录是因为有系统变量记录了这个数据
# $OLDPWD 记录上次所在目录; $PWD 记录当前所在目录
```

- 查看目录

- 命令: `tree`

```
# 查看指定目录数的层级
tree -L 1 /


# 每个文件和目录前显示完整的相对路径
tree -f
[Sun Oct 15 10:08:22 7] root@rocky9:~ #tree -f /storage/
/Storage
└── /Storage/test
    ├── /Storage/test/baidu.html
    ├── /Storage/test/ps_demo.txt
    ├── /Storage/test/rename.txt
    └── /Storage/test/robots.txt

1 directory, 4 files
```

```
1 directory, 4 files
```

```
# 每个文件和目录前显示最新更改时间
tree -D
[Sun Oct 15 10:10:36 11] root@rocky9:~ #tree -D /storage/
/Storage/
└── [Oct 14 09:12] test
    ├── [Sep 27 12:03] baidu.html
    ├── [Oct 13 20:48] ps_demo.txt
    ├── [Jan 3 2020] rename.txt
    └── [Jan 3 2020] robots.txt
```

```
1 directory, 4 files
```

```
1 directory, 4 files
```

```
# 每个文件和目录前显示文件大小
tree -s
[Sun Oct 15 10:08:30 8] root@rocky9:~ #tree -s /storage/
/Storage/
└── [        79] test
    └── [     2381] baidu.html
```

```

├── [      270] ps_demo.txt
├── [      2814] rename.txt
└── [      2814] robots.txt

# 每个文件和目录前显示文件/目录拥有者
tree -u
[Sun Oct 15 10:09:28 9] root@rocky9:~ #tree -u /Storage/
/Storage/
└── [root      ] test
    ├── [root      ] baidu.html
    ├── [root      ] ps_demo.txt
    ├── [root      ] rename.txt
    └── [root      ] robots.txt

# 每个文件和目录前显示权限标示
tree -p
[Sun Oct 15 10:11:18 12] root@rocky9:~ #tree -p /Storage/
/Storage/
└── [drwxr-xr-x] test
    ├── [-rw-r--r--] baidu.html
    ├── [-rw-r--r--] ps_demo.txt
    ├── [-rw-r--r--] rename.txt
    └── [-rw-r--r--] robots.txt

# 使用通配符对tree的目录进行筛选
tree -P pattern 这里的pattern不支持正则表达式，仅支持通配符
[Sun Oct 15 10:33:09 26] root@rocky9:~ #tree -P 'r*.txt' /Storage/
/Storage/
└── test
    ├── rename.txt
    └── robots.txt

1 directory, 2 files

```

1 directory, 2 files

常用通配符：

- \* 匹配任意数量的字符（包括零个）。
- ? 匹配任意一个字符。
- [...] 匹配方括号中的任意一个字符。

## • 创建目录

- 命令：`mkdir`

语法格式：`mkdir [pv] [-m mode] directory_name...`

```
# mkdir在指定路径创建目录
mkdir /Storage/test   # 在Storage目录下创建一个test目录

# 默认在当前路径创建目录
mkdir dir1   # 在当前目录下创建名为dir1的目录
```

```

# 一次创建多个同级目录，每个目录间用空格隔开
mkdir dir1 dir2 dir3

# 创建多级目录
mkdir -p dir1/dir2/dir3

# mkdir在指定路径创建目录
mkdir /Storage/test # 在Storage目录下创建一个test目录

# 默认在当前路径创建目录
mkdir dir1 # 在当前目录下创建名为dir1的目录

# 一次创建多个同级目录，每个目录间用空格隔开
mkdir dir1 dir2 dir3

# 创建多级目录
mkdir -p dir1/dir2/dir3

# -v 会显示创建每个目录的详细信息
[Sun Oct 15 11:12:00 39] root@rocky9:/ #mkdir -pv
/Storage/test/dir1/dir2/dir3
mkdir: created directory '/Storage/test/dir1'
mkdir: created directory '/Storage/test/dir1/dir2'
mkdir: created directory '/Storage/test/dir1/dir2/dir3'

# -m mod 指定创建文件的权限
[Sun Oct 15 11:19:39 45] root@rocky9:demo #mkdir -m 644 demo2
[Sun Oct 15 11:20:12 46] root@rocky9:demo #ll
total 0
drwxr-xr-x. 2 root root 6 Oct 15 11:19 demo1
drw-r--r--. 2 root root 6 Oct 15 11:20 demo2
drwxr-xr-x. 3 root root 18 Oct 15 10:43 dir1

```

- **删除目录**

- 命令：`rmdir`
- 注意：用于删除空目录，此命令要删除的目录内，不能有文件存在

```

# 删除单一目录，注意：删除目录内不能有文件
rmdir <directory_name>

# 同时删除同级多个目录，每个目录用空格隔开
rmdir dir1 dir2 dir3

# -p 删除多级目录
rmdir -p dir1/dir2/dir3 # 同时删除dir1及其子目录dir2,dir3

```

## 目录的本质

- 在Unix和类Unix的文件系统中，每个文件或目录都有一个与之关联的inode（索引节点）。
- 这个inode包含了关于文件的元数据，例如文件的权限、大小、修改时间、拥有者、所用的数据块的位置等，但注意，**它不包含文件名**。
- 目录中的数据部分，包含了文件名与inode编号的映射关系

```
fileA -> inode34  
fileB -> inode57
```

- 因此目录下文件的元数据（非文件名的改变）并不会导致目录中数据部分的内容发生改变，因为：文件元数据的改变，会导致inode的数据部分发生变化，但是inode的编号/值不变。这样，文件名和对应的inode编号的映射关系就没有发生变化，所以目录数据内容无变化
- 目录的大小跟文件大小无关，仅跟目录的数据部门，即目录下文件和inode映射关系的大小有关

## 创建初始目录的时候，硬链接数初始为2的原因

- 在UNIX和Linux文件系统中，目录的硬链接数从2开始是有特定的原因的。当你创建一个目录（例如dir1），初始的两个硬链接代表：
  - 引用该目录的名字：这就是你所创建的目录名，如dir1。自身是一个硬链接
  - .（点）：每个目录都有一个特殊的名字.，它引用自身。当你进入dir1并列出内容时，你会看到一个.目录，它实际上指向dir1自身。
- 当你在dir1内创建子目录时，dir1的硬链接数会增加。这是因为每个子目录都有一个名为..（双点）的特殊目录名，它指向其父目录。因此，每当你在dir1内创建一个子目录，dir1的硬链接数就会增加1。

## 管理文件的相关命令

- 查看文件列表

- 命令：`ls`

```
语法格式: ls [OPTION]... [FILE]...
```

```
# -a 显示包含隐藏文件在内的所有内容  
ls -a  
  
# -i 显示文件索引节点(inode)  
ls -i  
[Sun Oct 15 11:39:16 65] root@rocky9:test #ls -i  
136601235 baidu.html 137507906 ps_demo.txt 136601225 rename.txt 136601224  
robots.txt  
  
# -l 以长格式显示目录下内容列表  
# 长格式输出信息：文件名、文件类型、权限、硬链接数、所有者、组、文件大小、修改时间  
ls -l  
[Sun Oct 15 11:39:28 67] root@rocky9:test #ls -l  
total 16  
-rw-r--r--. 1 root root 2381 Sep 27 12:03 baidu.html  
-rw-r--r--. 1 root root 270 Oct 13 20:48 ps_demo.txt  
-rw-r--r--. 1 root root 2814 Jan 3 2020 rename.txt  
-rw-r--r--. 1 root root 2814 Jan 3 2020 robots.txt  
  
# 用文件目录的更改时间排序  
ls -t  
[Sun Oct 15 11:45:06 73] root@rocky9:test #ls -tl  
total 16  
-rw-r--r--. 1 root root 270 Oct 13 20:48 ps_demo.txt  
-rw-r--r--. 1 root root 2381 Sep 27 12:03 baidu.html  
-rw-r--r--. 1 root root 2814 Jan 3 2020 rename.txt  
-rw-r--r--. 1 root root 2814 Jan 3 2020 robots.txt
```

```

# 按文件大小，从大到小排序
ls -S
mystical@mystical 0101 #11-Sh
total 60K
-rwxrwxr-x 1 mystical mystical 17K Jan 1 23:05 a.out
-rw-rw-r-- 1 mystical mystical 1.6K Jan 1 22:29 7.struct.c
-rw-rw-r-- 1 mystical mystical 861 Jan 1 21:12 6.ifdef.c
-rw-rw-r-- 1 mystical mystical 536 Jan 1 23:05 8.union.c
-rw-rw-r-- 1 mystical mystical 529 Jan 1 14:49 2.array.c
-rw-rw-r-- 1 mystical mystical 521 Jan 1 10:47 1.demo.c
-rw-rw-r-- 1 mystical mystical 445 Jan 1 16:45 3.string.c
-rw-rw-r-- 1 mystical mystical 404 Jan 1 17:14 4.pointer.c
-rw-rw-r-- 1 mystical mystical 402 Jan 1 20:53 5.ifdef.c
-rw-rw-r-- 1 mystical mystical 375 Jan 1 14:55 3.address.c
-rw-rw-r-- 1 mystical mystical 44 Jan 2 15:00 website.txt

# ls后面支持通配符过滤，不加单引号
[sun Oct 15 11:49:26 80] root@rocky9:test #ls -l *.txt
-rw-r--r--. 1 root root 270 Oct 13 20:48 ps_demo.txt
-rw-r--r--. 1 root root 2814 Jan 3 2020 rename.txt
-rw-r--r--. 1 root root 2814 Jan 3 2020 robots.txt

ls -l <file>
#如果file是目录，则直接查询该目录下的内容，要查询目录使用
ls -d1 <file>
# 如果file是普通文件，则正常查看list

```

- 文件的时间属性

- atime: 记录最后一次的访问时间
  - atime的更新策略：连续在24小时内访问读取atime,24小时内不会更新atime；但在更改文件内容的时候会顺便更新atime
- mtime: 记录最后一次文件数据部分的修改时间
- ctime: 记录最后一次文件元数据的修改时间
- 注意：mtime的改变一定会引起ctime的改变
- 对于目录这种特殊文件
  - 其目录文件的数据部分(data block)存放的就是目录中的文件名等信息。所以在目录中创建，删除文件会改变目录的mtime，而目录的mtime的改变一定会引起ctime的改变，但其文件内容的改变，并不会引起目录mtime和ctime的改变
  - 当你访问一个目录（例如列出其内容）时，目录的 atime（访问时间）会被更新。如果你修改了目录中的一个文件，那么在大多数文件系统配置下，为了访问并修改该文件，你首先需要“访问”该目录，从而导致目录的 atime 被更新。所以修改一个目录下的文件，那么这个目录的atime通常情况下，是会更新的。但是...
  - 出于性能原因，一些现代的文件系统或挂载选项可能会默认禁用 atime 的更新。这种设置被称为noatime，它可以减少磁盘I/O，特别是在频繁读取文件但不经常修改它们的系统上。因此，如果文件系统是以 noatime 选项挂载的，那么访问文件或目录不会更新其 atime。
- ls查看文件的3个时间属性

```
# 默认显示文件的mtime
```

```
ls -l
```

```
# 显示文件的ctime
```

```
ls -l --time=ctime
```

```
# 显示文件的atime
```

```
ls -l --time=atime
```

- 关于atime的挂载选项

- 'noatime'

- 访问文件/目录不会更新atime

- 'relatime'

- 满足两个条件之一才更新atime

- 文件的atime时间超过一天

- 文件的mtime比atime更晚

- 查看文件属性信息

- 命令: `stat`

- 作用: 用于显示文件的详细属性

语法格式: `stat [文件或目录]`

```
# 查看文件属性
```

```
[Sun Oct 15 14:35:09 93] root@rocky9:test #stat rename.txt
File: rename.txt
Size: 2814          Blocks: 8           IO Block: 4096   regular file
Device: fd00h/64768d  Inode: 136601225  Links: 1
Access: (0664/-rw-rw-r--) Uid: (    0/      root)  Gid: (    0/      root)
Context: unconfined_u:object_r:default_t:s0
Access: 2023-10-13 20:36:29.807941125 +0800
Modify: 2020-01-03 16:33:48.000000000 +0800
Change: 2023-10-15 14:34:30.473235676 +0800
Birth: 2023-09-27 11:57:14.168869373 +0800
```

- 命令: `file`

- 作用: 使用 `file` 辨识文件的类型

语法格式: `file [OPTION] file_name`

```
# -i 查看文件的MIME类型
```

```
[Sun Oct 15 14:46:28 102] root@rocky9:test #file -i test.log
test.log: text/plain; charset=us-ascii
```

```
# -b 省略文件名称, 直接打印结果
```

```
mystical@ubuntu2204:~/C_coding/2024/jan/0128$ file 1.for.c
1.for.c: C source, Unicode text, UTF-8 text
```

```
mystical@ubuntu2204:~/C_coding/2024/jan/0128$ file -b 1.for.c
C source, Unicode text, UTF-8 text
```

```
# -f 从一个文件中，获取数据进行处理
mystical@ubuntu2204:~/test$ file -f studyvim.txt
/bin:           symbolic link to /usr/bin
/etc/passwd:   ASCII text
/home/:        directory
```

- windows与unix格式文本之间的相互转换

- Windows和Unix文本差异：
  - Windows每行末尾是回车符加换行符
  - Unix的每行末尾只有换行符结束
- 相互转换需要使用dos2unix

```
sudo apt install dos2unix
# Windows文本格式转Unix
dos2unix test.txt

# Unix文本格式转Windows
unix2dos test.txt
```

- 创建或刷新文件

- 命令： touch

```
# 如果文件存在则刷新时间，如果不存在则创建空文件

touch -a    # 改变atime, ctime
touch -m    # 改变mtime, ctime
touch -h    # 刷新链接文件本身，默认刷新目标文件
touch -c    # 只刷新已存在的文件，如果文件不存在，也不会创建文件
touch --time=STRING # 修改指定时间，如：--time=atime
touch -r    # 使用某个文件的修改时间作为当前文件的修改时间
# 改变atime和mtime并刷新ctime
touch -t    # 修改atime,mtime到指定日期时间
# 比如01020304，指2024-01-02 03:04:00
# 比如0102030405，指2001-02-03 04:05:00

# 示例
touch `date +%F-%T`.txt
2024-01-30-18:08:58.txt
```

- 复制文件

- 命令： cp

语法格式： cp [OPTION] SOURCE DEST

```
# -b 覆盖已存在的目标前先对其做备份，后缀为~
[sun Oct 15 15:03:16 108] root@rocky9:test #cp -b newtest.txt test.log
cp: overwrite 'test.log'? y
[sun Oct 15 15:03:32 109] root@rocky9:test #ls
baidu.html  dir1  dir2  dir3  newtest.txt  ps_demo.txt  rename.txt
robots.txt  test.log  test.log~
```

```
# -S 指定备份文件的后缀名
```

```
[Sun Oct 15 15:10:53 123] root@rocky9:test #cp -s .bak dir1/cptext.txt cptext.txt
cp: overwrite 'cptext.txt'? y
[Sun Oct 15 15:11:10 124] root@rocky9:test #tree .
.
├── baidu.html
├── cptext.txt
├── cptext.txt.bak
└── dir1
    └── cptext.txt
└── dir2
└── dir3
└── dir_cp
    └── cptext.txt
└── newtest.txt
└── ps_demo.txt
└── rename.txt
└── robots.txt
└── test.log
└── test.log~

4 directories, 11 files
```

# -i 覆盖前会先询问用户（推荐使用）  
`cp -i file cp_file`

# -r 递归处理，将目录及其中的文件一同复制  
`cp -r dir cp_dir`

# -a 复制特殊文件，使用-a  
`cp -a /dev/zero /home/mystical`

- 移动及重命名文件

- 命令：`mv`
  - 语法：`mv 目标文件 目标路径`
  - 语法2：`mv -t 目标路径 目标文件`
  - 语法3：`mv -bi 目标文件 目标路径`
    - i: 如果会覆盖文件则提示
    - b: 覆盖文件时会备份被覆盖的文件
- 命令：`rename`

关于批量创建和批量修改文件名

```
# rocky
# 批量创建文件与批量重命名
# rename <要改的字段> <改之后的字段> <使用通配符表示改的程度>
[Sun Oct 15 15:34:07 129] root@rocky9:py_test #touch pydemo{1..9}.txt
[Sun Oct 15 15:34:35 130] root@rocky9:py_test #ls
pydemo1.txt  pydemo2.txt  pydemo3.txt  pydemo4.txt  pydemo5.txt  pydemo6.txt
pydemo7.txt  pydemo8.txt  pydemo9.txt
[Sun Oct 15 15:34:38 131] root@rocky9:py_test #rename .txt .py *.txt
[Sun Oct 15 15:35:23 132] root@rocky9:py_test #ls
```

```
pydemo1.py  pydemo2.py  pydemo3.py  pydemo4.py  pydemo5.py  pydemo6.py  
pydemo7.py  pydemo8.py  pydemo9.py  
[Sun Oct 15 15:35:25 133] root@rocky9:py_test #rename py python py*  
[Sun Oct 15 15:35:43 134] root@rocky9:py_test #ls  
pythondemo1.py  pythondemo2.py  pythondemo3.py  pythondemo4.py  
pythondemo5.py  pythondemo6.py  pythondemo7.py  pythondemo8.py  
pythondemo9.py
```

- **删除文件**

- 命令: `rm`

语法格式: `rm [OPTION]...FILE...`

# -f 强制删除文件, 即在删除文件时不提示确认, 并自动忽略不存在的文件

# -r 递归删除, 目标是目录的话, 整个目录文件全部删除

- `rm` 是危险命令, 建议用以下命令替换

```
alias rm='dir=/Storage/backup/data`date +%F%T`;mkdir $dir;mv -t $dir'
```

# 将所有要删除的文件, 移动到创建的垃圾箱目录中

## 文件元数据和节点表结构

- 作用: `df` 命令用于显示文件系统的磁盘空间使用情况
- 查看不同分区的节点编号使用情况
  - 命令: `df -i`

生产案例1: 提示空间满NO space left on device, 但`df`可以看到空间很多, 为什么

答:

节点编号不足, 一个文件能被创建需要同时满足两个前提  
足够的空间, 以及该文件系统下还有剩余的节点编号

生产案例2: 为什么`cp /dev/zero /boot/test.img`会把`/boot`的空间撑满

答:

1. `/dev/zero` 是一个特殊的设备文件, 它可以生成无限的零字节。当你尝试从它读取数据时, 它会持续不断地返回零字节。

2. `cp` 命令的作用是复制文件或目录。在这种情况下, 它从 `/dev/zero` 复制数据并尝试写入 `/boot/test.img`。

3. 因为 `/dev/zero` 提供了无限的零字节, `cp` 会持续写入数据到 `/boot/test.img`, 直到 `/boot` 分区没有更多的空间可用。

生产案例3: 当`test.img`被访问时, 管理员在主服务器删除`test.img`后, 为什么, 空间依然是满的

答:

因为当一个文件被使用时, 在另一侧删除该文件, 该空间并不会被立即释放, 只有当这个文件不被使用时, 才会释放这个空间

解决方法:

```
cat /dev/null > /boot/test.img; rm -rf /boot/test.img  
把文件清空后删除即可。  
echo -n '' > /boot/test.img 结果和上述cat /dev/null...相同
```

## 硬链接与软链接

- 硬链接：
  - 概述：本质上是多个文件名共用一个inode
  - 命令：`ln a.txt aa.txt`
  - 注意：
    - 因为本质是共用一个inode，所以不能跨分区创建硬链接，因为不同分区有独立的inode表
    - 同理，为了防止inode循环利用，所以目录也不能创建硬链接，但是在创建目录及其子目录的时候，系统会自动创建..和.这种目录的硬链接
    - 硬链接数本质上是inode计数器的值
- 软链接：
  - 概述：也叫符号链接，软链接的本质是创建了一个新文件，该文件的内容是源文件的路径，所以访问软连接文件，实质上系统访问指向了源文件
  - 命令：`ln -s 目标文件 软链接文件`
    - 注意：根据软链接的本质，软连接文件中的内容实际上是指向目标文件的路径，因此目标文件的路径如果是相对路径，那么一定是相对软链接的路径
    - 注意2：删除软链接的时候，不要加tab键补全，如果软连接文件后跟/，删除的时候，比如`rm -rf /Storage/test/test/`实际上是把原始目录中的内容一起删除

## Bash Shell特性

### 命令历史

- 概述：
- 使用bash shell，执行过的命令会记录到history对应的缓存区中。当用户注销时，会将history缓存区中的命令历史追加到`~/.bash_history`文件中。
- 当下次登录shell时，系统会读取命令历史文件中记录的命令到命令缓存区。

### history命令

```
# 语法:  
history -c  
# 清空历史命令，仅清空命令缓存区的命令，不影响.bash_history  
  
history -d offset  
[Tue Oct 17 10:52:59 22] root@rocky9:~ #history | tail -n 10  
1011 cat .bash_history  
1012 cat oldfile.txt  
1013 getent passwd root  
1014 getent passwd | tail -n 10  
1015 getent passwd  
1016 history  
1017 history -d 999
```

```

1018 history | tail -n 20 # 1018
1019 history | tail -n 30
1020 history | tail -n 10
[Tue Oct 17 10:54:14 23] root@rocky9:~ #history -d 1017
[Tue Oct 17 10:54:35 24] root@rocky9:~ #history | tail -n 10
1012 cat oldfile.txt
1013 getent passwd root
1014 getent passwd | tail -n 10
1015 getent passwd
1016 history
1017 history | tail -n 20 # 1017
1018 history | tail -n 30
1019 history | tail -n 10
1020 history -d 1017
1021 history | tail -n 10

```

# 删除命令缓存区中指定编号的历史命令

# 删除后，后面的命令编号会依次往前提

**history n** # 显示最近的n条命令，等同于**history|tail -n <num>**

**history -a** # 立即追加命令缓存区中的命令到历史文件中

**history -w** # 将命令缓存区的当前内容覆盖到**.bash\_history**文件。

**history -w <new\_file>** # 将命令缓存区中的内容存储到指定文件中

**history -r** # 从**.bash\_history**读取命令到命令缓存区，通常在开始新会话时使用。

**history -r <new\_file>** # 从指定文件中读取命令到缓存区

**history -p <指定历史命令>** # 将指定的数据显示在标准输出

# 输出的指令不会执行，也不会出现在历史缓存区中

[Tue Oct 17 14:13:14 50] root@rocky9:Storage #history

```

1 ls
2 touch test2.txt
3 echo "hello" >> test2.txt
4 cat test2.txt
5 history

```

[Tue Oct 17 14:13:22 51] root@rocky9:Storage #history -p \!-2

**cat** test2.txt

**history -s** # 将参数作为单独的条目添加到历史列表的末尾。

# 这允许你将一个或多个命令手动添加到历史记录中。

**history -s "echo hello"** # 将**echo hello**加入历史缓存区，但是不会执行

- 详细描述数据在历史缓冲区和**.bash\_history**的过程

- 在一个新的bash会话中开始时，**.bash\_history**文件中的历史命令会被加载到命令缓存区。（假设**.bash\_history**初始没有数据）
- 执行了10个bash命令。
- 退出bash，这10个命令写入到**.bash\_history**
- 开始一个新的bash会话，这10个命令加载到命令缓存区。
- 新的bash会话中，你执行了一个新的命令

- 退出bash，命令缓存区中的11个命令会替换.bash\_history文件中的内容。
  - 最终.bash\_history中共有11条命令
- 问题2：history -a 的原理
    - 纯粹从底层操作角度看，history -a是追加到文件的，不是覆盖。
    - 当我们在日常使用中考虑整个会话的上下文时，情况会变得复杂。每当开始一个新的bash会话，它都会从.bash\_history读取历史记录到命令缓存区。这意味着，命令缓存区现在包含了从.bash\_history文件加载的命令加上在当前会话中输入的新命令。
    - 所以，尽管history -a实际上是追加行为，但在一个常规的bash会话中，由于命令缓存区已经从.bash\_history加载了历史，看起来就像是被覆盖了。
    - 但就底层原理来说：尽管history -a实际上是追加行为，但在一个常规的bash会话中，由于命令缓存区已经从.bash\_history加载了历史，看起来就像是被覆盖了(真正的覆盖操作是history -w)。
  - 命令历史的快捷方式使用
    - ! 历史扩展的开始标识符。
      - 在配合 history -p 使用时，需要使用\! 转义
      - !! 重复执行前一个命令
      - !:0 执行前一个不含参数的命令
      - !n 执行缓冲区序号n的命令
      - !-n 执行缓冲区序号倒数第n个命令
      - !string 执行前一个以string开头的命令
      - !?string 执行前一个包含string的命令
      - :p 打印命令，不执行
        - !string:p
        - !\$:p 打印输出 !\$ (上一个命令的最后一个参数)的内容
        - !\*:p 打印输出 !\* (上一个命令的所有参数)的内容
      - ^ 删除替换
        - ^string 删除上一个命令的第一个string
        - ^string1^string2 将上一个命令的string1替换为string2
        - !:gs/string1/string2 将上一个命令的所有string1替换为string2
    - 调用历史参数
      - command !^ 调用上一个命令的第一个参数作为command的参数
      - command !\$ 调用上一个命令的最后一个参数
      - command !\* 调用上一个命令的所有参数
      - command !:n 调用上一个命令的第n个参数
      - command !n:^ 调用第n条命令的第一个参数
      - command !n:\$ 调用第n条命令的最后一个参数
      - command !n:m 调用第n条命令的第m个参数
      - command !n:.\* 调用第n条命令的第\*个参数
      - command !string:^ 调用以string开头的命令，获取它的第一个参数

■ ...

- 历史命令快捷键
  - `ctrl + R` 进入历史命令搜索模块
  - `ctrl + G` 退出历史命令搜索
- history命令相关的shell变量
  - \$HISTSIZE: 历史缓存区的条数限制, 默认1000
  - \$HISTFILE: 指定历史文件, 默认为`~/.bash_history`
  - \$HISTFILESIZE: 历史文件的条数限制, 默认1000
  - \$HISTTIMEFORMAT: 显示时间, 示例: `HISTTIMEFORMAT="%F %T"`
  - \$HISTIGNORE: 历史缓存区忽略指定命令, 多个命令用冒号:分隔
    - 示例: `export HISTIGNORE = "ls:pwd"`
  - \$HISTCONTROL:控制历史命令记录的方式
    - 值为ignoredups: 默认, 忽略重复的命令, 连续且相同认为重复
    - 值为ignorespace: 忽略所有以空白开头的命令
    - 值为ignoreboth: 相同于ignoreboth: 相当于ignoredups和ignorespace组合
    - 值为erasedups: 删除重复命令
- 提示: 如果要持久保存上述变量, 需要将上面指令存放在`etc/profile`或者`~/.bash_profile`中

## 命令行展开

- 所谓命令行展开, 即把命令行中给定的特殊符号自动替换为相应字符串的机制。在Bash Shell中有些符号有特殊含义
  - `~`: 自动替换为用户家目录
  - `~USERNAME`: 自动替换为指定用户的家目录
  - `{}`: 可包含一个以逗号分隔的字符串或序列, 能够将其展开为多个字符串

```
a{d,c,b}e # ade, ace, abe
/tmp/{a,b,c} # /tmp/a, /tmp/b, /tmp/c
/tmp/{a,b}/z # /tmp/a/z, /tmp/b/z
{1..6} # 1,2,3,4,5,6
{1..10..2} # 1,3,5,7,9
{a..d} # a,b,c,d
```

## 标志I/O重定向和管道

### 三种I/O设备

- Linux给程序提供三种I/O设备
  - 标准输入(STDIN) -0 默认接受来自终端窗口的输入
  - 标准输出(STDOUT) -1 默认输出到终端窗口
  - 标准错误(STDERR) -2 默认输出到终端窗口
- 每创建一个新的会话, `/dev/pts`下就会创建一个新的伪终端, 以数字序号命名, 三种I/O设备都会指向这些伪终端

- 数据流向有输入流(称为标准输入)、输出流，默认数据的输入流由键盘实现。默认的输出流是到当前的终端屏幕上，Linux系统里输出流又分为错误输出(称为标准错误输出)和正确的输出(称为标准输出)
- 当一个程序刚启动时，会自动打开三个I/O设备文件：标准输入文件STDIN(STD即standard标准，IN即input输入)，标准输出文件STDOUT，标准错误输出文件STDERR。分别得到文件描述符0, 1, 2。如果此时打开一个新的文件，它的文件描述符会是3
- 因为0, 1, 2都指向终端窗口，即

```
1rwx-----. 1 root root 64 Oct 22 21:01 0 -> /dev/pts/3
1rwx-----. 1 root root 64 Oct 22 21:01 1 -> /dev/pts/3
1rwx-----. 1 root root 64 Oct 22 21:01 2 -> /dev/pts/3
```

所以输入输出都在终端窗口显示

## 文件描述符

- 在Linux中每一个打开的文件，系统都会给这个打开的文件分配一个唯一的文件描述符(本质上是一个软链接)，指向正在打开的文件。
- 文件描述符是一个非负整数，是内核为了高效管理已被打开的文件所创建的索引。所有执行I/O操作的系统调用都需要文件描述符
- 因此可以通过查询一个的程序中的文件描述符，来发现这个程序打开了几个文件

```
cd /proc
# 在proc中能看到所有的进程pid的目录，进入这个目录
cd /5005
# 里面有允许该程序的文件路径和fd目录(即打开的文件描述符目录)
cd /fd
# 进入fd目录，就可以发现多个软链接，指向程序打开的文件
```

- 扩展：
  - 使用 \$\$ 可以发现当前打开shell的pid
  - 当打开一个新的会话，，创建一个伪终端，就是进入shell程序，使用 echo \$SHELL 可以查询当前使用的shell类型

## 重定向技术

- 概述：更改数据流向
- 标准输出和错误的重定向
  - STDOUT和STDERR可以被重定向到指定文件，而非默认的当前终端
  - 格式：

命令	操作符号	文件名
----	------	-----

- 支持的操作符号包括

```
1> 或 >    把STDOUT重定向到文件  
>|          强制覆盖（允许在set -C的情况下，强制覆盖）  
2>          把STDERR重定向到文件  
&>          把所有输出(标准输出和错误输出)重定向到文件
```

以上如果文件已经存在，文件内容会被覆盖

**set -C** 禁止将内容覆盖已有文件，但可追加

**set +C** 允许覆盖，默认

# 该知识点仅作了解

- 追加: >>可以在原有内容上，追加内容
- 合并多个程序进行重定向：
  - (CMD1;CMD2...) 合并多个程序的STDOUT
  - {CMD1;CMD2;...} 使用花括号，记得结尾加分号；
- 将标准输出和标准错误同时输出到指定文件

方法1:

```
ls python/ err/ &> all.log
```

方法2:

```
ls python/ err/ > all.log 2>&1
```

- & 符号的作用：
  - 表示要操作的是文件描述符而非文件或命令。
  - 例如&1这里指的是文件描述符1
  - 文件描述符的本质是软链接，软链接的本质是指向目标文件的路径，而1指向的是/dev/pts/0即伪终端，因此: 2>&1 的意思就是标准错误输出到描述符1指向的路径中
- 标准输入的重定向
  - 标准输入符号：< 后面各跟文件

```
bc < file
```

# 将需要计算的内容输入到指定文件中，然后标准输入到bc中

- 管道

- 管道符：|

```
seq -s+ 1 100 | bc # 5050
```

- 单行重定向与多行重定向

单行重定向:

```
cat > cat.log
```

多行重定向:

```
cat > cat2.log <<EOF # EOF是结束符
```

# 单行重定向在回车启用命令是，就已经创建了文件，然后每输入一行，回车后，内容就流入指定文件

# 多行重定向是在所有命令都输入完之后，遇到EOF结束时，才一起执行所有命令

## 支持标准输入的常用命令

- `seq`

- 作用：用于生成某个范围内的整数
- 语法：`seq [选项]...[首数][增量][尾数]`
- 选项：`-s`, `-f`, `-w`
- 示例：

```
# -f: 格式化数据，使用%g表示整数，例如%g aaa%03g。  
root@ubuntu2004:/Storage$ seq -f "id:%06g" 1 5  
id:000001  
id:000002  
id:000003  
id:000004  
id:000005
```

```
# 使所有数据同宽，位数小的前面补零，不能和-f同时使用  
root@ubuntu2004:/Storage$ seq -w 2 30 140  
002  
032  
062  
092  
122
```

```
# -s: 指定字符分隔产生的所有数字，默认为\n  
root@ubuntu2004:/Storage$ seq -s '+' 1 9  
1+2+3+4+5+6+7+8+9
```

- `s-nail` (原mailx)

- 配置（与外网邮箱互通）
- 在家目录下，创建`.mailrc`配置文件

```
# .mailrc内容如下  
set v15-compat  
set smtp-auth=login  
set from="昵称<qq号@qq.com>"  
set mta=https://qq号:qq邮箱授权码@smtp.qq.com:465
```

- 文件配置好后后，使用`s-nail`发送到外网的邮件，发件人为配置的qq邮箱
- 本机的邮箱默认位置

```
/var/spool/mail/
```

- **tr**

- 作用: 转换和删除字符
- 语法 `tr [OPTION].. SET1 [SET2]`
- 选项:

```
-d --delete: 删除所有属于第一字符集的字符  
-s --squeeze-repeats: 把连续重复的字符以单独一个字符表示, 即去重  
-t --truncate-set1: 将 第一个字符集对应字符转换为第二个字符集对应的字符, 如果第一个字符集的字符数量多于第二字符集数量, 超出部分忽略  
-c -C --complement: 取字符集的补集
```

```
[alnum]: 字母和数字  
[alpha]: 字母  
[digit]: 数字  
[lower]: 小写字母  
[upper]: 大写字母  
[space]: 空白字符  
[print]: 可打印字符  
[:punct:]: 标点符号  
[:graph:]: 图形字符  
[:cntrl:]: 控制(非打印)字符  
[:xdigit:]: 十六进制字符
```

- 示例:

```
tr 'a-z' 'A-Z' < /etc/issue  
  
tr [:lower:] [:upper:] < /etc/issue  
  
tr -d 'abc' # 删除字符串中的abc  
  
tr -dc 'abc' # 只保留字符串中的abc, 程序完全结束后返回结果  
  
df|tr -s ' ' # 将df的返回结果中的空格去掉
```

- **tee**

- 作用: 在 Linux 中用于从标准输入读取数据, 并将其内容输出到标准输出和一个或多个文件中。这使得你能够查看数据的同时, 将其保存到文件中。
- 基本用法

```
command1 | tee file1
```

- 示例

```
ls -l | tee output1.txt  
  
ls -l | tee -a output1.txt # 追加内容到output1.txt, 而不是覆盖  
  
ls -l /bin | tee output1.txt|cat -n  
# 同时将标准输入和标准错误通过管道符, 输入给tr进行处理  
  
ls /data /err |& tr 'a-z' 'A-Z' # 相对比较新的写法
```

- 案例2

```
# 设置用户口令  
echo magedu | passwd --stdin wang &> /dev/null  
# --stdin read new tokens from stdin (root only)
```

## 用户组和权限管理

### Linux安全模型

- 3A资源分派
  - **Authentication:** 认证, 验证用户身份
    - 常见的通过用户名和口令, 来区分验证用户信息
  - **Authorization:** 授权, 不同的用户设置不同的权限
    - 比如, 某个文件, 张三有访问权限, 而李四没有
  - **Accounting|Auditon:** 审计
    - 记录不同用户的操作记录
- 当用户登录成功时, 系统会自动分配命令token, 包括用户标识和组成员等信息

### 用户和组相关文件

- 用户: Linux中每个用户是通过 User Id(UID)来唯一标识的
  - 管理员: root, UID=0
  - 普通用户: 1-60000自动分配
    - 系统用户: 1-499(Centos 6以前), 1-999(Centos 7以后) 对守护进程获取资源进行权限分配
    - 登录用户: 500+(Centos 6以前), 1000+(Centos 7以后) 给用户进行交互式登录使用

## 守护进程的名称来源

"**Daemon**" 这个词在计算领域中指的是后台运行的进程，但它的起源和意义比计算领域的用法要古老得多。

在古希腊语中，"**δαίμων**" (*daimōn*) 的意思是神、命运之神或守护之神。它通常被描述为一种位于神和凡人之间的超自然存在，有时候是好的，有时候是中性的，但它通常在人们察觉不到的情况下起作用，影响人类的命运或行为。这与计算机中的守护进程的概念相似，后者在后台静默地运行，而用户通常不直接与之互动。

在计算的早期历史中，MIT的研究人员开始使用"**daemon**"这个词来描述在背后处理各种任务的计算机进程。有一个有趣的历史小故事称，它是由 PDP-10 的操作员 **Fernando J. Corbató** 和 **Victor A. Vyssotsky** 创造的。他们将这种程序称为 "**disk and execution MONitor**"，并使用 "**DAEMON**" 作为其缩写。无论如何，该术语已经在计算社区中被广泛接受，并被用来描述后台进程。

因此，"**daemon**" 这个词用于描述后台进程是因为这些进程默默地、在后台工作，而不需要用户的直接干预，很像古希腊神话中的守护之神在幕后影响事物的方式。

- 用户组：Linux中可以将一个或多个用户加入用户组中，用户组是通过Group ID(GID)来唯一标识的。(不起实质作用)
- 用户和组的关系
  - 用户的主要组(primary group)：用户必须属于一个且只有一个主组，默认创建用户时自动创建和用户名同名的组，做为用户的主要组，由于此组中只有一个用户，又称为私有组
  - 用户的附加组 (supplementary group)：一个用户可以属于零个或多个辅助组，附属组
- 安全上下文
  - Linux安全上下文Context：运行中的程序，即进程 (process)，以进程发起者的身份运行，进程所能访问资源的权限取决于进程的运行者身份；
  - 比如：分别以root和wang的身份运行/bin/cat /etc shadow，得到的结果是不同的，资源能否被访问，是由运行者的身份决定，非程序本身

## 用户和组管理命令

- Linxu中和用户有关的文件

```
ll /etc/passwd /etc/shadow

getent passwd = cat /etc/passwd #直接在屏幕上打印出passwd的信息
getent passwd <用户名>

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
...
后续省略
```

```
//每一个上都是一个用户信息，有几行，就有几个用户，大部分是程序服务作为用户的信息
//这里的每个信息都用“:”隔开，一共7个信息，分别是：
//name, password(密码被迁到别的地方存储), UID, GID, GECOS (描述), directory(家目录), shell类型

/etc/shadow #存放指令的文件，密码，日期

root@ubuntu2004:/etc# cat shadow
root:$6$CNC6y8cEgE3pNLSo$U6Nk1J2hmcmAF8y4yAPZLoXY12swtBvo62z1l670jetuR0Ndv9CL
29saSn7ZXrJFRHu0jgE5aJso1NYvopYgG0:19503:0:99999:7:::
daemon:*:19430:0:99999:7:::
bin:*:19430:0:99999:7:::
sys:*:19430:0:99999:7:::
sync:*:19430:0:99999:7:::
games:*:19430:0:99999:7:::
man:*:19430:0:99999:7:::
lp:*:19430:0:99999:7:::
mail:*:19430:0:99999:7:::
...
此处省略
mystical:$6$74dN0t1hGHDSqcyd$/tZTWDJMRdMdYREiF9npwh9nSHhbKb4hvW/ka15jCyzBtocg
p1s3MhDP21hnsULwOaBkb3.QXfEZNIk1Topuh/:19503:0:99999:7:::
lxde!:19503::::::
rtkit:*:19529:0:99999:7:::
dnsmasq:*:19529:0:99999:7:::
avahi:*:19529:0:99999:7:::
cups-pk-helper:*:19529:0:99999:7:::
pulse:*:19529:0:99999:7:::
geoclue:*:19529:0:99999:7:::
saned:*:19529:0:99999:7:::
colord:*:19529:0:99999:7:::
gdm:*:19529:0:99999:7:::
...
此处省略

// 密码的组成:
同时使用三个$符号分隔成3字段。
// 第一字段代表加密使用的单向加密算法，使用数字表示，1表示MD5算法，6表示SHA512算法。
// 第二字段代表加密算法使用的随机因子，一般称为salt盐，salt和加密算法一起构成了密码最终的加密方式。
// 第三字段代表加密后的结果，即密文
// 组成：密码（加密后）
// 更改/创建口令时间（以1970年以基准到更改密码的天数）
// 最短有效期，即有效期内无法更改口令（0代表可随时更改）（该有效期仅针对普通用户，对root用户无效）
// 密码有效期，
// 密码以前几天提醒，
// 超过密码有效期，再过几天账号锁定
// 账号有效期

/etc/group

root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
```

```

adm:x:4:syslog,mystical
tty:x:5:syslog
disk:x:6:
lp:x:7:
mail:x:8:
news:x:9:
...
后续省略

//组的文件组成: 组名, 密码, id, 成员

/etc/gshadow

root:*::
daemon:*::
bin:*::
sys:*::
adm:*::syslog,mystical
tty:*::syslog
...
后续省略

//组名, 组的口令, 管理员账号 (管理员可以在组中增删用户)
// 附加组成员

```

- 用户管理命令配置文件

- `/etc/login.defs`
- `/etc/login.defs`文件在Linux系统中扮演着重要角色，它提供了系统管理员用于配置系统全局用户和组设置的默认值。这个文件被`login`程序和大多数用于添加和管理用户的工具读取，如`useradd`、`userdel`、`usermod`、`passwd`等。`/etc/login.defs`文件包含了一系列的配置指令，这些指令定义了用户账户和密码策略的各种方面，比如密码过期时间、密码复杂度要求、新用户的默认家目录权限等。
- 以下是`/etc/login.defs`文件中一些常见配置项的说明：
  - **MAIL\_DIR:** 定义用户邮件存放的目录。通常设为`/var/mail`。
  - **PASS\_MAX\_DAYS:** 账户密码的最大有效期。过了这个期限，用户必须更改密码。
  - **PASS\_MIN\_DAYS:** 两次密码更改之间需要等待的最少天数。这防止用户立即更改密码以绕过密码历史策略。
  - **PASS\_WARN\_AGE:** 密码过期前，系统开始警告用户的天数。
  - **UID\_MIN和UID\_MAX:** 分配给新用户的UID范围。这通常用来区分系统账户和普通用户账户。
  - **GID\_MIN和GID\_MAX:** 分配给新用户组的GID范围。
  - **CREATE\_HOME:** 是否为新用户自动创建家目录。通常设置为yes，以确保每个用户都有自己的家目录。
  - **UMASK:** 定义了新创建的用户文件的默认权限掩码。通常设置为022或027，以防止新文件和目录对其他用户是可写的。
  - **USERGROUPS\_ENAB:** 如果设置为yes，当创建一个新用户时，系统也会创建一个与用户名相同的用户组，并将此用户添加到该组。

- **ENCRYPT\_METHOD**: 定义用于加密用户密码的算法。常见的值包括MD5、SHA256、SHA512。
- **CREATE\_MAIL\_SPOOL**: 定义是否为每个新创建的用户创建一个邮件池文件。
  - 这个文件还包含其他多个设置，可以根据系统管理员的需求来调整。编辑/etc/login.defs文件时应格外小心，因为错误的配置可能会影响系统安全和用户管理策略。修改完毕后，强烈建议对配置进行检查，以确保没有意外的更改会影响系统操作。
- useradd命令默认配置文件
  - `cat /etc/default/useradd`

```
[root@rocky8 ~]# cat /etc/default/useradd
# useradd defaults file
GROUP=100
# useradd不指定组,且/etc/login.defs中的USERGROUPS_ENAB为no或
# useradd -N时, group 为100
HOME=/home
INACTIVE=-1
表示不锁定
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
#默认家目录父目录
#对应/etc/shadow文件第7列, 即用户密码过期后的帐号锁定的宽限期,-1
#对应/etc/shadow文件第8列, 即用户帐号的有效期
#默认bash
#用于生成新建用户家目录的模版文件
CREATE_MAIL_SPOOL=yes
```

- 用户管理常用命令

<code>useradd</code>	新建用户
----------------------	------

```
# useradd wileson -- 创建一个新用户wileson
#
# id wileson -- 使用id 用户名, 可以查看该用户是否存在
uid=1001(wilson) gid=1001(wilson) 组=1001(wilson)
#
# 参数
# useradd -d 路径 -- 指定家目录 ‘ -d ’
# useradd -s /sbin/nologin -- 指定shell类型 ‘ -s ’
-- nologin还是给服务使用的, 用户一般是使用/bin/bash
# useradd -g postfix -- 指定主组 ‘ -g ’;postfix是指定的组名
# useradd -G [,GROUP2...] -- 指定附加组 ‘ -G ’
-- 注意添加的时候, 要保证组存在
# useradd -N -- 不创建和用户同名的私用组作为主组
# useradd -r -- 创建系统用户, Centos 6: id<500, Centos7及以上: id<1000
# useradd -M -- 不创建家目录 ‘ -M ’
# useradd -m -- 创建家目录 ‘ -m ’
# useradd -u 1088 -- 指定UID ‘ -u ’
# useradd -c 'COMMENT' -- 新的注释信息
总结:
useradd [option...] user_name
```

!!!! (面试题) 所有的新建用户, 系统自带的配置文件, 都是复制 /etc/skel 的文件,

因此，如果在这个目录下新增文件，则所有的新建账户下，都会默认自带新增文件

**userdel**      删除用户

```
# userdel wilson -- 删除指定用户（保留家目录）
# userdel -r wilson -- 删除用户（不保留家目录）
# userdel -rf wilson -- 强制删除
```

**passwd**      修改用户密码

```
# passwd wilson -- 更改指定用户的密码
# passwd -- 直接用passwd， 默认更改root用户密码
```

**usermod**      修改用户属性

-- usermod -d 指定路径 用户名  
# usermod -md /home/wile1 wilson -- 更改指定用户家目录，新家目录不会自动创建  
-- 如果要创建新家目录并移动原家数据，同时使用-m选项  
-- -md 原家目录被删除，替换为新指定的目录并转移数据  
-- 如果不想要创建家目录的时候自带的配置文件，可以创建的时候使用-M，然后使用usermod -d 指定目录  
-- 效果：  
-- 1. 登录后自动进入该家目录  
-- 2. 对用户相关的配置文件也放在了该家目录

```
# usermod -u UID -- 更改uid
# usermod -g GID -- 更改gid
# usermod -G [..GROUP2...GROUPN] -- 更改附加组
-- 原来的附加组会被覆盖；若要保留原有的附加组，则要同时使用-a选项
# usermod -s SHELL -- 更改shell类型
# usermod -l login_name -- 更改新的用户名
# usermod -e YYYY-MM-DD -- 指明用户账号过期日期
# usermod -f INACTIVE -- 设置非活动期限，即宽限时间
# usermod -L -- Lock指用户，在/etc/shadow密码栏增加！
```

**chage**      修改用户密码策略

```
# chage -l <username> -- 查看指定用户的密码管理策略
```

- useradd的默认属性的文件

```
[Mon Oct 30 16:10:21 104] root@rocky9:spool $ cat /etc/default/useradd
# useradd defaults file
GROUP=100 # 100表示users组的编号
# 如果使用useradd -N, 不创建私用组的话, 就用users(GID100)作为主组
HOME=/home
INACTIVE=-1 # -1表示其实口令到期也不锁账号, 如果是10表示口令到期10天后锁账号
EXPIRE= #-- 表示账户有效期
SHELL=/bin/bash
SKEL=/etc/skel # /etc/skel目录下的内容表示创建新用户时, 家目录自带的文件
# /etc/skel实际上是新建账号的模板文件夹
CREATE_MAIL_SPOOL=yes # 默认创建账号邮箱
```

- 新建用户的相关文件

```
/etc/default/useradd
/etc/skel/*
/etc/login.defs # 控制账号口令, 即shadow后面配置的默认定义
```

- 批量创建用户

```
newusers newusers file
```

- 范例:

```
[root@ubuntu2204 ~]# cat user.txt
u1:123456:1024:1024::/home/u1:/bin/bash
u2:123456:1025:1025::/home/u2:/bin/bash
[root@ubuntu2204 ~]# newusers user.txt
[root@ubuntu2204 ~]# id u1
uid=1024(u1) gid=1024(u1) groups=1024(u1)
[root@ubuntu2204 ~]# id u2
uid=1025(u2) gid=1025(u2) groups=1025(u2)
```

- 批量更改用户口令

```
cat <格式文件> | chpasswd
```

格式文件内容:

```
user_name : password
```

示例:

```
mystical:12346
user1:admin123
```

在ubuntu中, 由于Ubuntu不支持--stdin的选项, 因此无法像Centos一样使用如下命令

```
echo password | passwd --stdin <user_name>
```

因此, 可以使用如下口令实现非交互式更改口令:

```
echo user_name:password | chpasswd
```

```
[root@ubuntu2204 ~]# cat pwd.txt
u1:1234567
u2:1234567
```

```
#标准输入重定向  
[root@ubuntu2204 ~]# chpasswd < pwd.txt  
  
#多行重定向  
[root@ubuntu2204 ~]# chpasswd <<EOF  
> u1:1234567  
> u2:1234567  
> EOF
```

- 查看用户相关的ID信息
  - id命令可以查看用户的UID, GID等信息

```
id [OPTION]... [USER]
```

- 常见选项

```
-u : 显示UID  
-g : 显示GID  
-G : 显示用户所属组ID  
-n : 显示名称, 需配合ugG使用, 例如: -un;-gn;-Gn
```

- 组管理命令

groupadd 新建用户组

格式: groupadd [OPTION]... group\_name

常见选项:

-g GID 指明GID号; [GID\_MIN,GIDMAX]  
-r 创建系统组, CentOS6之前: ID<500, CentOS 7以后: ID<1000

注意:

如果你知道你要创建的是一个系统组, 并且你想确保它在系统组的 **GID** 范围内, 那么使用 **-r** 选项是一个好的实践。如果你只是想创建一个具有特定 **GID** 的组, 不管它是否是系统组, 那么只使用 **-g** 选项就足够了。

添加 **-r** 选项是为了明确表达你的意图, 并确保组被正确地分类为系统组。不过, 如果你手动指定了一个在系统组 **GID** 范围内的 **GID**, 即使没有使用 **-r** 选项, 该组在某种程度上也被视为系统组。

范例:

```
groupadd -g 48 -r apache
```

groupmod 组属性修改

格式: groupmod [OPTION]... group

常见选项:

-n <新组名> <原组名>: 新名字  
-g GID : 新的GID

示例: groupmod -n www apache

```
groupdel group_name    删除用户组
```

- 关于脚本中写组和用户的创建的示例

```
getent group apache > /dev/null || groupadd -g 48 -r apache
getent passwd apache > /dev/null || \
useradd -r -u 48 -g apache -s /sbin/nologin -d /var/www -c "Apache" apache
exit 0
```

## 理解并设置文件权限

- Linux指令权限管理

```
su <用户名>      -- 切换为用户名的身份权限
-- 不完全切换，身份切换了，但是环境还是之前的root路径下
-- 而且环境变量依然是之前切换前的环境变量，对于依赖个人用户环境配置的程序执行，可能会出现非预期的现象
su - <用户名>    -- 完全切换，身份和所在路径都切换了
su - <用户名> -c 'cmd'
-- 以切换的指定身份执行命令，但本身不切换身份，依然是当前用户
exit      -- 切换回上一个身份
```

---

su 和 sudo 的区别

```
su          -- 切换用户
su -USERNAME -- 使用Login shell方式切换用户

sudo         -- 以其他用户身份执行命令
visudo       -- 设置需要使用sudo的用户(组)
```

**sudo**的意义：

可以授权普通用户使用**sudo**命令去执行指定的**root**权限才能进行的被允许普通用户也可以执行的命令

场景一

```
-- root用户执行:
# shutdown -h 30    -- 30分钟后系统自动关闭
-- 如果普通用户想要停止这个操作，需要执行
# shutdown -c     -- 停止关闭指令
-- 但是普通用户没有这个命令的执行权限
-- 因此需要root用户通过sudo配置给普通用户授权
```

具体的授权操作：

使用**visudo**进入配置文件  
配置操作共3段指令

第一段：确定授权的用户/组；

```
-- 用户：直接输入名称
-- 组：‘%’加‘组名’
```

第二段：给哪些用户授权哪些命令

```
-- 命令形式
-- 远程登录的话：
ALL=授权的指令
```

-- 本地服务器登录：  
**localhost**=授权的指令  
-- 如果是多条命令的话，中间用逗号‘，’隔开  
-- 被授权的命令需要填写具体的命令地址，地址的查询使用**which**指令  
例如：授权的指令为**shutdown**，但是不知道**shutdown**指令的地址  
**# which shutdown;**  
**/sbin/shutdown** --返回结果  
-- 查询 **shutdown** 的地址为**/sbin/shutdown**，这个地址就可以写在授权命令中

第三段：判定普通用户在使用管理员授权的命令时，是否需要输入密码

-- 不需要的话：  
**NOPASSWD: ALL** (不建议)

整个示例的命令为：

```
# zhangyifeng ALL=/sbin/shutdown -c
```

最后保存退出

总结：使用**visudo**，配置**visudo**的信息，可以达到给普通用户授权使用**root**权限指定命令的效果。很好用哦

#### • Linux文件权限管理

```
root@clem:~# ll
total 32
drwx----- 4 root root 4096 Jun 27 08:54 .
drwxr-xr-x 20 root root 4096 Jun 28 20:32 ../
-rw----- 1 root root 458 Jun 28 22:20 .bash_history
-rw-r--r-- 1 root root 3204 Jun 27 08:38 .bashrc
-rw-r--r-- 1 root root 161 Dec 5 2019 .profile
drwx----- 3 root root 4096 Dec 21 2022 snap/
drwx----- 2 root root 4096 Dec 21 2022 .ssh/
```

解析：-文件类型 权限 链接数 所属账号 所属主组 大小 时间 文件名

#### chown -- 更改文件用户权限/更改所有权限

```
# chown user_name file_name
-- 将file文件的用户权限改为user_name

# chown user_name.group_name file_name
# chown user_name:group_name file_name
-- 将该文件所属的用户名，组名一起变更。

# chown -R user_name.group_name dir
-- 将文件夹下，所有文件的所属账号和组都一起变更，危险命令
```

#### chgrp -- 仅更改文件所属组权限

```
# chgrp group_name file_name
-- 更改所属组
```

#### • 文件权限类型

- r Readable 读
- w Writable 写
- x eXecutable 执行

示例:

```
-rw-r--r-- 1 root root 161 Dec 5 2019 .profile
```

```
- rw- r-- r--
```

第一个‘-’不在权限标识中，仅指文件类型

第一组‘rw-’ 标明所属用户权限

第二组‘r--’ 同组其他用户权限

第三组‘r--’ 其他用户权限，通用权限，所有人基本都能用

- rwx对于目录的权限意义（与文件不同）
  - r: 可以使用ls查看此目录中文件名列表，但无法看到文件的属性meta信息，包括inode号，不能查看文件的内容
  - w: 可以在此目录中创建文件，也可以删除此目录中的文件，而和此被删除的文件的权限无关。
  - x: 如果，没有该目录的执行权限，用户将无法访问这个目录下的所有文件，所以执行权限是目录访问的基本权限，没有执行就无法进入，是的，连目录进都进不去！
    - 如果只有x,没有r的话，对于目录来说，就是只能访问，但是看不到ls，就是没有访问目录下文件名的权限，但是如果这个文件你知道名称，且这个文件的通用权限有读权限，那么对于普通用户来说，只是无法浏览目录下文件名及文件元信息，但是依然可以cat到文件内的内容
- 更改文件的读写权限

`chmod -- 模式法和数字法`

模式法:

```
chmod who opt per file
-- who: u,g,o,a
-- u(所属者), g(所属组), o(other),a(all)
-- opt:+,-,=
-- per:r,w,x
```

示例:

```
chmod o+r file -- 表示file文件的通用权限中增加r权限
```

数字法:

```
rwx rw- --- a.txt
111 110 000 a.txt
7   6   0
chmod 760 a.txt
```

- 默认权限

- 定义：当创建一个文件或文件夹时，会默认一个权限，这个默认权限时如何产生的，如何修改

`umask -- 这个指令可以修改新建文件/文件夹的权限`

```
# umask -- 查看当前umask的值
```

```
-- root权限的默认umask值022  
-- 普通用户的默认umask值002
```

-----  
修改默认权限的实现方式

指定新建文件的默认权限

**666**-umask, 如果所得结果某位存在执行（奇数）权限，则将其权限+1，偶数不变

将权限+1的原因：

文件的执行时危险的！！！，如果没有执行权限，root也无法直接执行，但是没有读写权限，root依然能够进行读写

基于安全考虑，默认新建的文件不允许有执行权限！！

umask 的内在机制

```
666  
123    -- umask值
```

```
110110110  
001010011      -- mask  
110100100  
644
```

-----  
指定新建目录的默认权限

**777**-umask

-----  
修改默认权限

# umask <更改后的数字> -- 临时修改

永久修改：

root目录下，.bashrc文件内修改，添加umask <数值>，保存退出后，. .bashrc或者重启

-- 全局设置：/etc/bashrc 不建议，这里修改会影响全局所有用户

-- 用户设置：~/ .bashrc 只影响当前用户

## • 特殊权限

### SUID

用于二进制可执行文件，执行命令时取得文件属主权限或root权限

表现形式

```
# ls -l /usr/bin/passwd  
-rwsr-xr-x. 1 root root 33424 Apr 20 2022 /usr/bin/passwd
```

-- 表示这个文件，即使普通用户也能执行修改

实现SUID的指令

```
# chmod 4755 file_name
```

### SGID

用于目录，在该目录下创建新的文件和目录，权限自动更改为该目录的属组

### SBIT

保证该文件只有root和自己可以删除150

实现SBIT的指令

```
# chmod 1777 file_name
```

- 特殊属性 (限制管理员-root)

```
# chattr +i a.txt -- 添加i属性限制root  
  
# lsattr a.txt -- 查看i属性  
  
# chattr -i -- 删除i属性  
  
# chattr +a -- 只能追加，不能修改，不能删除
```

以下是一些chattr命令的常用属性：

#### 1. a (Append only)

设置后，文件只能被追加内容，不能被删除或覆盖。这对于日志文件非常有用。

使用方法: chattr +a filename

#### 2. i (Immutable)

设置后，文件变为不可修改，即不能被删除、修改、重命名，也不能添加链接。即使是root用户也不能绕过这一限制。

使用方法: chattr +i filename

#### 3. s (Secure deletion)

设置后，当文件被删除时，其占用的磁盘空间会被立即覆盖，以确保数据不能被恢复。适用于包含敏感数据的文件。

使用方法: chattr +s filename

#### 4. S (Synchronous updates)

设置后，对文件的修改立即写入磁盘，类似于使用sync命令。这对于需要即时保存更改的重要文件很有帮助。

使用方法: chattr +S filename

#### 5. u (Undeletable)

设置后，文件内容在被删除后可以被恢复。这提供了一种简单的文件恢复机制。

使用方法: chattr +u filename

#### 6. A (No atime updates)

设置后，访问文件时不更新文件的访问时间。这可以提高对于频繁访问但不需要保持访问记录的文件的性能。

使用方法: chattr +A filename

#### 7. D (Synchronous directory updates)

设置后，对目录的更改会立即写入磁盘，这适用于需要高数据一致性的目录。

使用方法: chattr +D dirname

## ACL(Access Control List) 访问控制列表

- ACL权限功能：

- rwx 权限体系中，仅仅只能将用户分成三种角色，如果要对单独用户设置额外的权限，则无法完成；而ACL可以单独对指定的用户设定各不相同的权限；提供颗粒度更细的权限控制
- CentOS7 默认创建的xfs和ext4文件系统具有ACL功能
- CentOS7 之前版本，默认手工创建的ext4文件系统无ACL功能，需手动增加

```
tune2fs -o acl /dev/sdb1  
mount -o acl /dev/sdb1 /mnt/test
```

- ACL安装

```
sudo apt install acl
```

- ACL生效顺序:

所有者, 自定义用户, 所属组, 自定义组, 其他人

- ACL相关命令

- getfacl 可查看设置的ACL权限

```
# Display the file access control list:  
getfacl {{path/to/file_or_directory}}  
  
# Display the file access control list with numeric user and group IDs:  
getfacl -n {{path/to/file_or_directory}}  
  
# Display the file access control list with tabular output format:  
getfacl -t {{path/to/file_or_directory}}
```

- setfacl 可设置ACL权限

```
setfacl [-bkndRLPvh] [{-m|-x} acl_spec] [{-M|-X} acl_file] file ...  
#常用选项  
-m|--modify=acl          #修改acl权限  
-M|--modify-file=file    #从文件读取规则  
-x|--remove=acl           #删除文件acl 权限  
-X|--remove-file=file    #从文件读取规则  
-b|--remove-all           #删除文件所有acl权限  
-k|--remove-default       #删除默认acl规则  
--set=acl                 #用新规则替换旧规则, 会删除原有ACL项, 用新的替代, 一定要包含UGO的设置, 不能象 -m一样只有 ACL  
--set-file=file            #从文件读取新规则  
--mask                     #重新计算mask值  
-n|--no-mask               #不重新计算mask值  
-d|--default                #在目录上设置默认acl  
-R|--recursive              #递归执行  
-L|--logical                #将acl 应用在软链接指向的目标文件上, 与-R一起使用  
-P|--physical               #将acl 不应用在软链接指向的目标文件上, 与-R一起使用
```

- setfacl示例:

```
#设置 tom 无任何权限
[root@ubuntu2204 tmp]# setfacl -m u:tom:- f1
[root@ubuntu2204 tmp]# getfacl f1
# file: f1
# owner: root
# group: root
user::rw
user:tom:--
group::r-
mask::r-
other::r-
#查看文件，多了一个小 +
[root@ubuntu2204 tmp]# ll f1-rw-r--r--+ 1 root root 5 May  9 23:22 f1
```

- 编辑ACL规则文件

- ACL规则文件是一个文本文件，其中每一行都包含一个ACL规则。这些规则的格式通常如下：

```
[类型]:[用户/组]:[权限]
```

## 文本处理

### 查看文本内容

#### cat

- cat 可以查看文本内容

```
cat [OPTION]... [FILE]...

#常见选项
-E | --show-ends          #显示行结束符$
-A | --show-all            #显示所有控制符
-n | --number               #对显示出的每一行进行编号
-b | --number-nonblank      #非空行编号
-s | --squeeze-blank        #压缩连续的空行成一行
```

#### tac

- tac 逆向显示文本内容，行倒序显示
- 格式

```
tac [option]... file...

# 常用选项
-s      # 以指定分隔符进行逆序，默认以换行符为分隔逆序
-r      # 配合-s使用，可以用正则表达式指代复杂规则的分隔符
```

```
-b      # 将分隔符视为行的一部分，并将其放在每行的开头，默认情况下，分隔符被认为是行的末尾部分

示例:
seq 10| tac

cat f1 # 1-2-3-4-5-
echo -n `cat f1` | tac -s '-' #-5-4-3-2-1

cat f2 # 1--2---3-4---5-
echo -n `cat f2` | tac -r -s '-+' #-5----4-3---2--1

cat f3 # ,1,2,3,4,5
echo -n `cat f3` | tac -b -s ',' # ,5,4,3,2,1
```

## rev

- rev: 同一行的文本内容，反转显示

```
echo "12345" | rev
>> 54321
```

## 查看非文本文件内容

### hexdump

```
hexdump [option] file

# 常用option:
-C          # 规范的十六进制和ASCII显示。这可能是最常用的选项
-n length   # 显示文件的前length个字节
-s offset    # 从指定偏移量offset处开始显示

# 示例:
hexdump -n 100 -C /dev/sda
hexdump -C < <(echo {a..z}|tr -d ' ')
# <(command) 括号内会临时生成一个临时文件，然后传给前面的指令
```

## 分页查看文件内容

### more

- 可以实现分页查看文件，可以配合管道实现输出信息的分页
- 格式

```
more [option] file

# 常用选项
-d          # 在底部显示提示
-s          # 压缩连续空行
```

- 命令选项

```
空格键      # 翻页  
回车键      # 下一行  
!command    # 执行指令  
h           # 显示帮助  
:f          # 显示文件名和当前行号  
=           # 显示行号
```

## less

- less 也可以实现分页查看文件或STDIN输出, less 命令是man命令使用的分页器
- 配置(.bashrc)

```
# 默认man指令分页器  
# 配置彩色man页面, 使用使用less和groff  
# 确保man命令使用less作为分页器。这通常是默认配置, 但你可以通过设置MANPAGER或PAGER环境变量来  
明确指定  
export MANPAGER='less -R'  
export PAGER='less -R'  
# -R选项告诉less解释颜色编码, 这是显示颜色输出的关键  
  
# 指定颜色样式  
export LESS_TERMCAP_mb=$(printf '\e[01;31m')      # 开始闪烁  
export LESS_TERMCAP_md=$(printf '\e[01;38;5;74m')  # 开始粗体  
export LESS_TERMCAP_me=$(printf '\e[0m')            # 结束模式  
export LESS_TERMCAP_se=$(printf '\e[0m')            # 结束强调模式  
export LESS_TERMCAP_so=$(printf '\e[38;5;246m')     # 开始强调模式  
export LESS_TERMCAP_ue=$(printf '\e[0m')            # 结束下划线  
export LESS_TERMCAP_us=$(printf '\e[04;38;5;146m')  # 开始下划线
```

- 命令选项
  - 类似与VIM中的操作

## 显示文本前面或后面的行内容

### head

- head 可以显示文件或标准输入的前面行
- 格式

```
head [option]... file...  
  
# 常用选项  
-c | --bytes=N      # 指定获取前N个字节  
-n | --lines=N      # 指定获取前N行,N如果为负数, 表示从文件头取到倒数第N前
```

- 应用

```
# 设置随机10位密码并记录  
cat /dev/urandom | tr -dc '[:alnum:]' | head -c 10 | tee -a pass.log | passwd --  
stdin mage
```

## tail

- tail: tail 和 head 相反, 查看文件或标准输入的倒数行
- 格式

```
tail [option]... file...

# 常用选项
-c      # 指定获取后N字节
-n      # 指定获取后N行, 如果写成+N, 表示从第N行开始到文件结束
-f      # #跟踪显示文件fd新追加的内容, 常用日志监控
       # 当删除再新建同名文件, 将无法继续跟踪
-F      # 跟踪文件名, 当删除文件再新建同名文件, 可继续追踪
```

## 按列抽取cut

- cut 命令可以提取文本文件或STDIN数据的指定列
- 格式

```
cut [option] file

# 常用选项
-b      # 以字节分割, 指定要显示的列
-c      # 以字符分割, 指定要显示的列
-d      # 以指定分割符分割
-f      # 显示指定的列 eg:-f1; f1,2; f1-10
--output-delimiter=String  # 用指定字符替代分隔符
```

- 示例:

```
df | head -n 2| tail -n 1| tr -s " "| cut -d " " -f5
```

## 合并多个文件paste

- paste 合并多个文件同行号的行到一行
- 格式

```
paste [option] file

# 常用选项
-d      # 指定分隔符, 默认tab
-s      # 合成一行显示, 默认用tab分割
-z      # 以NULL 字符而非换行符作为行尾分隔符
```

- 示例: 批量修改密码

```
[root@ubuntu2204 ~]# cat user.txt
tom
jerry
[root@ubuntu2204 ~]# cat pass.txt
123456
654321
[root@ubuntu2204 ~]## paste -d: user.txt pass.txt
tom:123456
jerry:654321
[root@ubuntu2204 ~]# paste -d: user.txt pass.txt | chpasswd
```

## 文本折叠fold

- fold命令是一个在Unix和类Unix系统中用于折叠文本的实用工具，它能够将较长的文本行折叠（或分割）成多个较短的行，使得文本适合在限定宽度的显示区域中查看。这在处理长行文本文件或输出时特别有用，以便更易于阅读或符合特定格式要求。
- 格式

```
fold [option]... file...

# 常用选项
-w, --width=WIDTH      # 设置每行的目标宽度
-s, --spaces             # 在空格处断行
-b, --bytes              # 按字节计数而非按列计数。这对于处理包含多字节字符的文本（如UTF-8编码的文本）时特别有用，以确保正确的宽度计算。
```

## 分析文本工具

### 文本数据统计wc

- wc命令可用于统计文件的行总数、单词总数、字节总数和字符总数，可以对文件或STDIN中的数据统计
- 格式

```
wc [option] file
cat file | wc [option]

# 常用选项
-l|--lines    #只计数行数
-w|--words    #只计数单词总数
-c|--bytes    #只计数字节总数
-m|--chars    #只计数字符总数
-L|--max-line-length #显示文件中最长行的长度
```

### 文本排序sort

- 把整理过的文本显示在STDOUT，不改变原始文件
- 格式

```
sort [option] file

# 常用选项
-n      # 以数字大小排序
-R      # 随机排序
-r      # 倒序排序
-t      # 指定列分隔符
-k      # 指定排序列
-u      # 去重
```

## uniq

- uniq 命令从输入中删除前后相接的重复的行，常和 sort 配合使用
- 格式

```
uniq [option]... file...

# 常用选项
-c      # 显示每行出现的次数
-d      # 仅显示重复行
-u      # 仅显示不重复的行
```

## 比较文件

### diff

- diff 命令比较两个文件之间的区别
- 格式

```
diff [option]... file...

# 常用选项
-u      # 详细显示
```

### patch

- 使用diff生成的文件和其中一个源文件修复另一个源文件

```
patch [option]... file1 file2

# 常规选项
-b      # 备份原文件，防止覆盖后丢失

# 使用场景
diff f1 f2 > diff.log

# 使用diff.log和f1可以生成f2文件
patch -b f1 diff.log
# 此时f1文件的内容变为f2，原f1的内容在备份文件f1.orig中
```

## cmp

- 查看二进制文件的不同

```
cmp file1 file2
```

示例：

```
mystical@ubuntu2204:~/test$ cmp /bin/ls /bin/dir  
/bin/ls /bin/dir differ: byte 25, line 1  
# 表示在25字节后，出现不同，使用hexdump进行查看  
  
mystical@ubuntu2204:~/test$ hexdump -s 20 -Cn 20 /bin/dir  
00000014 01 00 00 00 90 6a 00 00 00 00 00 00 40 00 00 00  
# 90  
mystical@ubuntu2204:~/test$ hexdump -s 20 -Cn 20 /bin/ls  
00000014 01 00 00 00 b0 6a 00 00 00 00 00 00 40 00 00 00  
# b0
```

## 文本处理三剑客之 grep

- 作用：
  - 文本搜索工具，根据用户指定的“模式”对目标文本逐行进行匹配检查；打印匹配到的行
- 模式：
  - 由正则表达式字符及文本字符所编写的过滤条件
- 格式

```
grep [option]... PATTERN [FILE]...
```

#常用选项

```
--color=auto      # 对匹配到的文本着色处理  
-m N             # 匹配N次后停止  
-v               # 取反  
-i               # 忽略大小写  
-n               # 显示行号  
-c               # 统计匹配次数  
-o               # 仅显示匹配到的字符，使用o的时候，后面一般跟正则表达式  
-q               # 静默模式，什么都不输出，但是匹配成功与否可以通过$?的数值看出来  
-A N             # 匹配到的行的后N行  
-B N             # 匹配到的行的前N行  
-C N             # 匹配到的行的前后各N行  
-e               # 表示或关系，进行查询  
  
# 示例：  
grep -e false -e bash /etc/passwd  
-w               # 表示匹配的是单词，而不是仅仅包含该字符  
-E               # 表示使用ERE  
-F               # 表示不支持正则表达式  
-f file         # 将文件中的每行内容作为匹配的正则规则  
                 # 也可以用来判断两个文件中的相同行  
-r               # 递归处理，不处理链接文件  
-R               # 递归处理，处理链接文件
```

# Awk

## Awk基础

- Awk语法

```
awk [option] 'pattern[action]' file ...
# options: awk可选参数
# pattern: 模式
# {action}: 动作; 最常用的动作是 print 和 printf
# file: 文件/数据
```

- Awk场景

```
[Mon Aug 14 09:16:05 -bash 24] root@parrot:Storage #cat prac1.txt
root x 0 0 root /root /bin/bash
daemon x 1 1 daemon /usr/sbin /usr/sbin/nologin
bin x 2 2 bin /bin /usr/sbin/nologin
sys x 3 3 sys /dev /usr/sbin/nologin
sync x 4 65534 sync /bin /bin/sync
games x 5 60 games /usr/games /usr/sbin/nologin
man x 6 12 man /var/cache/man /usr/sbin/nologin
lp x 7 7 lp /var/spool/lpd /usr/sbin/nologin
mail x 8 8 mail /var/mail /usr/sbin/nologin
news x 9 9 news /var/spool/news /usr/sbin/nologin
inetsim x 132 138 /var/lib/inetsim /usr/sbin/nologin
_gvm x 133 140 /var/lib/openvas /usr/sbin/nologin
beef-xss x 134 141 /var/lib/beef-xss /usr/sbin/nologin
mystical x 1000 1003 mystical /home/mystical /bin/bash
saned x 135 142 /var/lib/saned /usr/sbin/nologin
[Mon Aug 14 09:16:21 -bash 25] root@parrot:Storage #cat prac1.txt | awk '{print
$1}'
root
daemon
bin
sys
sync
games
man
lp
mail
news
inetsim
_gvm
beef-xss
mystical
saned
```

- 简单规则:

- awk默认以空格为分隔符，且多个空格也识别为一个空格，作为分隔符
- awk是按行处理文件，一行处理完毕，处理下一行，根据用户指定的分隔符去工作，没有指定则默认空格

- awk内置变量

内置变量	解释
\$n	指定分隔符后，当前记录的第n个字段
\$0	表示整行
FS	字段分隔符，默认是空格
NF(Number of fields)	分割后，当前一共有多少个字段
NR(Number of records)	当前记录数，行数
更多详情查看man手册	man awk

- 一次性输出多列信息
  - 示例：awk '{print \$1,\$4,\$5}' prac1.txt
  - ','逗号代表空格分隔显示
- 自定义输出内容
  - awk，必须外层用单引号，内层用双引号
  - 示例：awk '{print "第一列",\$1,"第二列",\$2}' prac1.txt
  - 内层双引号代表字符串
- awk参数

参数	解释
-F	指定分割字段符
-v	定义或修改一个awk内部的变量
-f	从脚本文件中读取awk命令

- 输出指定行信息
  - 示例：awk 'NR==5{print \$0}' prac1.txt 输出第5行数据
  - 示例2：awk 'NR==5,NR==10{print \$0}' prac1.txt 输出第5-10行数据
  - 示例3：awk 'NR==5,NR==10{print NR,\$0}' prac1.txt 输出第5-10行数据，并给每一行的内容添加行号
- awk变量
  - 内置变量

内置变量	解释
\$n	指定分隔符后，当前记录的第n个字段
\$0	表示整行
NF(Number of fields)	分割后，当前一共有多少个字段
NR(Number of records)	当前记录数，行数
FS	输入字段分隔符，默认为空格
OFS	输出字段分隔符，默认为空格

内置变量	解释
RS	输入记录分隔符，指定输入时的换行符
ORS	输出记录分隔符，输出指定的换行符
FNR	各文件分别计数的行号
FILENAME	FILENAME: 当前文件名
ARGC	命令行参数个数
ARGV	数组，保存命令行所给定的各参数

- 自定义变量
  - 方法一： -v varName=value
  - 示例： awk -v myname="峰哥" 'BEGIN{print "我的名字是？ ",myname}'
  - 方法二： 在程序中直接定义
- awk格式化
  - printf 格式化输出
  - printf 和 print 的区别

#### format的使用

要点：

1. 其与print命令的最大不同是，printf需要指定format
2. format用于指定后面的每个item的输出格式
3. printf语句不会自动打印换行符：\n

format格式的指示符都以%开头，后跟一个字符；如下：

%c： 显示字符的ASCLL码；  
 %d, %i： 十进制整数；  
 %e, %E： 科学计数法显示数值；  
 %f： 显示浮点数；  
 %s： 显示字符串；  
 %u： 显示无符号整数；  
 %%： 显示%自身

#### printf修饰符

-： 左对齐， 默认右对齐；  
 +： 显示数值符号； printf"%+d"

- awk模式pattern
  - BEGIN： 处理文本前，先执行BEGIN模式指定的动作
  - END： 处理完指定文本后，需要执行的动作

关系运算符	解释	示例
<	小于	x < y

关系运算符	解释	示例
<=	小于等于	x <= y
==	等于	x == y
!=	不等于	x != y
>=	大于等于	x >= y
>	大于	x > y
~	匹配正则	x~/正则/
!~	不匹配正则	x!~/正则/

- awk使用正则语法
  - awk '/正则表达式/{动作} file

## Sed (Stream EDitor)

### Sed工作原理

- Sed是从文件或管道中读取一行，处理一行，输出一行；再读取一行，再处理一行，再输出一行，直到最后一行。每当处理一行时，把当前处理的行存储在临时缓冲区中，称为**模式空间(Pattern Space)**，接着用sed命令处理缓冲区中的内容，处理完成后，把缓冲区的内容送往屏幕。接着处理下一行，这样不断重复，直到文件末尾。

### Sed基本用法

- 格式：

```
sed [option]... 'script;script;...' [inputfile...]
```

- 常用选项

-n	不输出模式空间内容到屏幕，即不自动打印
-e	多点编辑
-f FILE	从指定文件中读取编辑脚本
-r,-E	使用扩展正则表达式
-i.bak	备份文件并原处编辑
-s	将多个文件视为独立文件，而不是单个连续的长文件流

#说明：

-ir	不支持
-i -r	支持
-ri	支持
-ni	危险选项，会清空文件

- script格式：

'地址格式'

1. 不给地址：对全文进行处理
2. 单地址：

```
#: 指定的行, $:最后一行
/pattern/: 被此处模式所能够匹配到的每一行
3. 地址范围:
  #,#      #从#行到第#行, 3, 6 从第3行到第6行
  #,+#     #从#行到+#行, 3, +4 表示从3行到第7行
  /pat1/,/pat2/
  #,/pat/
  /pat/,#
4. 步进: ~
  1~2 奇数行
  2~2 偶数行
```

#### '命令格式'

- -p 打印当前模式空间内容, 追加到默认输出之后
- -Ip 忽略大小写输出
- -d 删除模式空间匹配的行, 并立即启用下一轮循环
- a [\]text 在指定行后面追加文本, 支持使用\n实现多行追加
- i [\]text 在行前面插入文本
- c [\]text 替换行为单行或多行文本
- w file 保存模式匹配的行至指定文件
- r file 读取指定文件的文本至模式空间中匹配到的行后
- = 为模式空间中的行打印行号
- ! 模式空间中匹配行取反处理
- q 结束或退出sed

#### • 扩展:

sed命令:

eg:sed ''

在不写地址的时候, 默认对全文处理

在不写指令的时候, 默认自动打印

sed '' < /etc/passwd

在无任何地址, 指令的情况下, 默认打印输入的所有内容

使用-n, 可以关闭这种自动打印, 使其只打印指定内容

CentOS 安装后必须所做的初始化操作

```
#关闭SELINUX
sed -i '/^SELINUX=/c SELINUX=disabled' /etc/selinux/config
#关闭防火墙
systemctl disable --now firewalld
```

Ubuntu开启root远程登录功能

```
#sudo -i
#passwd root
#sed -i '/PermitRootLogin/c PermitRootLogin yes' /etc/ssh/sshd_config
#systemctl restart sshd
```

#### • 分组与后项引用

#查找替代

```
s/pattern/string/修饰符  
替换修饰符：  
g      行内全局替换  
p      显示替换成功的行  
w      /PATH/FILE 将替换成功的行保存至文件中  
I, i    忽略大小写
```

```
#查找替换，支持使用其他分隔符，可以是其他形式：s@@@,s###
```

```
#正则表达式：后项引用
```

```
示例：
```

```
echo 123456789 -> 456123789  
echo 123456789|sed -rn 's/(123)(456)(789)/\2\1\3/p'  
#用括号分组  
用反斜杠'\'+数字表示分组的代号
```

## AWK

### AWK整体格式

```
awk [options] 'program' var=value file...  
awk [options] -f programfile var=value file...
```

### Program格式

```
pattern {action statement;...}  
  
# pattern: 决定动作语句何时触发及触发事件，比如：BEGIN,END,正则表达式等  
# BEGIN: 数据处理前，先要执行BEGIN之后的动作  
# END: 数据都处理结束后，执行END的动作  
# action statement: 对数据进行处理，放在{}内指明，常见：print,printf
```

program通常被放在单引号中，并可以由三种部分组成

- BEGIN语句块
- 模式匹配的通用模块
- END语句块

### 常见选项 (Option)

- **-F** "分隔符" 指明输入时用到的字段分隔符，默认不限个数的空白分

```
# 多个分隔符的写法  
-F'[:space:]+'%'  
df | awk -F'[:space:]+' '{print $1,$5}'
```

- **-v var=value** 变量赋值

# 分隔符、域和记录

- 有分隔符分隔的字段（列column,域field）标记\$1,\$2...\$n称为域标识，\$0为所有域
- 文件的每一行称为记录record
- 如果省略action，则默认执行print \$0的操作

## print

# 文件查找与打包压缩

## 文件查找

- 非实时查找（数据库查找）：locate
- 实时查找：find

## locate

- locate简介
  - locate 查询系统上预建的文件索引数据库 /var/lib/mlocate/mlocate.db
  - 索引的构建是在系统较为空闲时自动进行(周期任务)，执行updatedb可以更新数据库
  - 索引构建过程需要遍历整个根文件系统，很消耗资源
  - locate和update命令来自于mlocate包
- 工作特点
  - 查找速度快
  - 模糊查询
  - 非实时查找
  - 搜索的是文件全路径，不仅仅是文件名
  - 可能只搜索用户具备读取和执行权限的目录
- locate安装

```
# CentOS
yum install -y mlocate

# Ubuntu
apt install -y plocate
```

- 格式

```
locate [option...] [PATTERN]...

# 手动更新数据库
updatedb

# 常用选项
-A                      # 输出所有能匹配到的文件名，不管文件是否存在
-b                      # 仅匹配文件名部份，而不匹配路径中的内容
```

```
-c          # 统计匹配到的数量
-d database # 指定数据库查找
-i          # 忽略大小写
-n N        # 只显示前N条匹配数据
-r          # 使用基本正则表达式
--regex     # 使用扩展正则表达式
```

## find

- find 是实时查找工具，通过遍历指定路径完成文件查找；
- 工作特点
  - 查找速度略慢
  - 精确查找
  - 实时查找
  - 查找条件丰富
  - 可能只搜索用户具备读取和执行权限的目录
- 格式

```
#find [-H] [-L] [-P] [-Olevel] [-D help|tree|search|stat|rates|opt|exec] [path...]
[expression]
```

```
find [OPTION]... [查找路径] [查找条件] [处理动作]
```

## 条件组合（-a,-o,-not）不加括号的情况下，默认处理动作只针对最后一个查找条件

# 查找路径：指定具体目标路径，默认当前路径  
# 查找条件：指定的查找条件，可以为文件名，大小，类型，权限等标准进行，默认找出指定路径下的所有文件  
# 处理动作：对符合条件的文件做操作，默认输出至屏幕

# 常用查找条件

```
-maxdepth      # 为最大搜索遍历深度
-mindepth      # 为最小搜索遍历深度
-name          # 指定文件名， 支持通配符
-depth         # 优先处理文件（用处不大）
```

# 根据文件名和inode查找

```
-name name
-iname name
-inum number
-samefiles name # 查找相同inode号的文件
-link n         # 链接数为n的文件
-regex "PATTERN" # 以PATTERN匹配整个路径，而非文件名
```

# 根据属主属组查找

```
-user USERNAME # 查找属主为指定用户（UID）的文件
-group GPRNAME # 查找属组为指定用户组（GID）的文件
-uid UserID
-gid GrpID
-nouser
-nogroup
```

```
# 根据文件类型查找
-type TYPE

# 查找空文件或空目录
-empty

# 组合条件
-a          # 与, 默认值
-o          # 或
-not | !   # 非

# 排除子目录
-prune      # 跳过, 排除指定目录, 必须配合-path使用

## 示例:
# 排除当前目录下的dir1目录下的.txt文件, 但是会输出dir1
find -path './dir1' -prune -o -name "*.txt"
# 去除dir1
find -path './dir1' -prune -o -name "*.txt" -print

# 根据文件大小查找
-size [-|+]N UNIT

10k: (9k, 10k]
-10k: (--, 9k]
+10k: (10k,++)

# 根据时间查找
## 以天为时间单位
-atime [-|+]N
-mtime [-|+]N
-ctime [-|+]N

## 以分钟为单位
-amin [-|+]N
-mmin [-|+]N
-cmin [-|+]N

# 根据权限查找
- perm [/|-] MODE
MODE    # 精确匹配权限
/MODE   # (u,g,o) 只要有一位匹配即可, or关系
+MODE   # 每一类对象都必须同时拥有指定权限, and关系
        # 权限位是0表示不关注该位置的权限

# 正则表达式
-regextype type      # 正则表达式类型
-regex pattern       # 正则表达式

# 处理动作
-print           # 将匹配到的内容输出到控制台
-print0          # 用空字符null代替换行符进行分界
-ls              # 将匹配到的内容ls -l显示出来
-fls             # 查找到的所有文件的长格式信息保存至指定文件中, 相当于 -ls > file
-delete          # 删除查找到的文件, 慎用!
-ok command {} \; # 对查找到的每个文件执行有command指定的命令, 并对每个文件进行交互式确认
```

```
-exec command {} \; #对查找到的每个文件执行由COMMAND指定的命令
{} # 用于引用查找到的文件名称自身
```

## 参数替换 - xargs

- 作用：
  - 使不支持标准输入的命令可以接受管道传递的参数
  - 许多命令不支持过多参数，可以使xargs分组传参
- 格式

```
xargs [OPTION]... COMMAND [INITIAL-ARGS]...

# 常用选项
-o          # 用 ascii 中的0或 null 作分隔符
# 示例：
# [root@ubuntu2204 ~]# find -type f -print0 |xargs -0 ls
# './a b' ./f-1.txt ./f-2.txt ./f-3.txt
# 适用于分隔文件中带空格的情况
-a          # 从文件中读入作为输入
-d          # 指定分隔符
-E END      # 指定结束符，执行到此处截止
-n          # 一次接受n个参数
-p          # 每次执行前确认
-t          # 显示过程
```

## 压缩与解压缩

### compress和uncompress

- 此工具来自于ncompress包，目前已很少使用
- 对应文件是.Z后缀
- 格式

```
compress...[option]... [file]...
uncompress [option]... [file]...

# 常用选项
-d          # 解压缩，相当于uncompress
-c          # 结果输出至标准输出，不删除源文件
# 保留源文件方法
# compress -c syslog > syslog.Z
-f          # 覆盖已存在目标文件
-v          # 显示过程
-r          # 递归压缩目录下所有内容

# 压缩比：1:5
```

## gzip和gunzip

- 来自于gzip包
- 对应的文件是.gz后缀
- 格式

```
gzip [option]... FILE...
gunzip [option]... FILE...

# 常用选项
-c          # 将压缩数据输出到标准输出中，并保留原文件
-d          # 解压缩，相当于gunzip
-k          # 保留原文件
-l          # 显示原文件大小，压缩文件大小，压缩比，压缩前文件名
-r          # 递归压缩目录下所有文件
-s          # 指定压缩文件后缀
-v          # 显示过程
-1          # 最快压缩，压缩比最小，但压缩时间块
-9          # 最慢压缩，压缩比最高，但压缩

# 压缩比：1:8
```

## bzip2和bunzip2

- 来自于bzip2包
- 对应文件是.bz2后缀
- 格式

```
bzip2 [option]... FILE...
bunzip2 [option]... FILE...
bzcat # 不解压，查看文件内容

# 常用选项
# 同gzip: -dkvfc19

# 压缩比：1:10
```

## xz和unxz

- 来自于xz包
- 对应文件是.xz包
- 格式

```
xz [option]... FILE...
unxz [option]... FILE...

# 常规选项
# 同gzip: -dzkvc19
-T          # 开多线程，默认为1
```

## zip和unzip

- zip可以实现打包目录和多个文件大包为一个文件并压缩，但可能会丢失文件属性信息（如属主属组）
- 对应的文件是.zip后缀
- 格式

```
zip [option]... zipfile [FILE]...
unzip [option]... zipfile [FILE]...
```

# zip常用选项

```
-f      # 仅更新，不追加
-u      # 有则更新，无则追加
-d      # 删除指定压缩包内文件
-m      # 将文件压缩之后，删除原始文件
-r      # 递归压缩目录
-1~9
-v      # 显示过程
-c      # 替每个压缩的文件添加注释
-z      # 给压缩包添加注释
-P      # 非交互设置密码
-e      # 交互式设置密码
-i      # 仅压缩指定文件
-x      # 压缩时排除指定文件
```

# unzip常用选项

```
-l      # 显示压缩文件内所包含的文件
-t      # 查看压缩文件完整性
-z      # 查看压缩包注释
-v      # 列出包内文件信息
-x      # 指定不需要解压缩的文件
-d      # 指定解压的目标目录
-n      # 压缩时不要覆盖原有文件
-o      # 直接覆盖
```

## zcat

- 其功能是在不解压的情况下查看压缩文件内容
- 格式

```
zcat [option]... FILE...
```

# 常用选项

```
-d      # 解压缩
-l      # 显示压缩文件内的文件列表
-r      # 递归操作
-t      # 测试文件完整性
```

# 打包和解包

## tar

- tar 即 Tape ARchive 磁带归档，可以对目录和多个文件打包成一个文件进行归档；其本身不具备压缩功能，但可以使用参数调用相应的压缩命令进行压缩；此命令可以保留文件属性，推荐使用；
- 对应文件后缀是.tar
- 格式

```
# 常用选项
-cvf      # 打包
-tvf      # 查看包内文件
-xvf      # 解包
-z        # gz压缩算法
-j        # bzip2压缩算法
-J        # xz压缩算法
```

# 磁盘存储和文件系统

## 磁盘结构

### 设备文件

- 设备文件：关联至一个设备驱动程序，进而能够与之对应硬件设备进行通信
- 设备号码：
  - 主设备号：major number, 标识设备类型
  - 次设备号：minor number, 标识同一类型下的不同设备
- 设备类型
  - 块设备：block, 存取单位"块", 磁盘
  - 字符设备：char, 存取单位"字符", 键盘
- 磁盘设备的设备文件命名

```
/dev/DEV_FILE
/dev/sdX    # SAS, SATA, SCSI, IDE, USB
/dev/nvme0n  # nvme协议硬盘, 如: 第一个硬盘: nvme0n1, 第二个硬盘: nvme0n2
```

- 虚拟磁盘

```
/dev/vd
/dev/xvd
```

- 不同磁盘标识：a-z, aa, ab..

```
/dev/sda, /dev/sdb...
```

- 同一设备的不同分区：1, 2....

```
/dev/sda1  
/dev/sda5
```

- 添加硬盘不重启识别

```
echo '---' > /sys/class/scsi_host/host0/scan  
echo '---' > /sys/class/scsi_host/host1/scan  
echo '---' > /sys/class/scsi_host/host2/scan
```

#### # 解读/sys/class/scsi\_host/host\*

scsi\_host文件夹下，代表了所有的scsi主机适配器接口，每个hostN都表示一个SCSI主机适配器接口

#### # 解读hostN目录下的文件

hostN目录包含了与该接口相关的一系列参数和操作接口。这些文件和目录提供了一种机制，允许用户空间程序查询和修改 SCSI 主机适配器的属性，以及执行特定的操作，如扫描新设备

##### # 举例

###### scan

> 这是一个写入接口，用于触发 SCSI 主机适配器扫描其管理的总线以识别新连接的设备。  
> 向这个文件写入特定的字符串（通常是 "---"）会指示适配器扫描所有可能的通道（channel）、目标（target）和逻辑单元号（LUN）

###### state

> 显示适配器的当前状态，如 "running" 或 "offline"

###### proc\_name

> 显示 SCSI 主机适配器的驱动程序名称  
> 知道了proc\_name，即驱动程序名称，可以通过"modinfo 驱动名称"查询驱动详细信息  
> 也可以通过lsmod|grep "驱动名称"查询该模块是否被加载  
> 使用modprobe可以手动加载或卸载驱动程序  
> 大多驱动程序以模块的形式被加载，模块（Kernel Module）是一种可以在系统运行时动态加载和卸载的代码块

###### unique\_id

> 为 SCSI 主机适配器提供一个唯一标识符

- SCSI主机适配器接口hostN放在sys目录下的原因

/sys 目录是 Linux 系统中的一个特殊文件系统，称为 sysfs。sysfs 提供了一种机制，通过它用户空间的程序可以与内核空间的数据结构进行交互，获取系统和硬件组件的信息，以及在某些情况下修改这些信息。

#### #为什么使用 sysfs

简化访问：

通过 sysfs，提供了一种标准和简单的方式来访问内核提供的信息和服务，这些服务以前可能需要特殊的系统调用或复杂的编程接口来访问。

统一接口：

sysfs 为许多不同类型的内核结构提供了一个统一的文件系统接口，包括设备、驱动程序和内核模块等，这使得管理和查询这些结构更加一致和简单。

#### 动态性：sysfs

是动态生成的，它的内容反映了当前系统的状态。当硬件设备被添加或移除时，sysfs 中相应的条目会相应地出现或消失，这提供了一种实时反映系统状态的机制。

- 扩展内容：检查和管理模块

**lsmod**: 列出当前加载的所有模块。

**modinfo**: 显示有关特定模块的详细信息。

**modprobe**: 智能地加载或卸载模块，包括处理模块依赖关系。

**insmod**: 加载一个模块到内核中，但不解决依赖关系。

**rmmod**: 从内核中卸载一个模块

- 创建设备文件

```
mknod /data/partition-sda1 b 8 1  
mount /data/partition-sda1 /mnt/
```

## dd指令

- 指令格式

**dd if=**输入文件 **of=**输出文件 [选项]

# **if=**文件名: 指定输入文件 (**input file**)，可以是设备文件（如磁盘分区 **/dev/sda**）或普通文件。

# **of=**文件名: 指定输出文件 (**output file**)，同样可以是设备文件或普通文件。

# [选项]: dd 命令提供了多种选项，用于控制数据的读取、写入和转换过程。

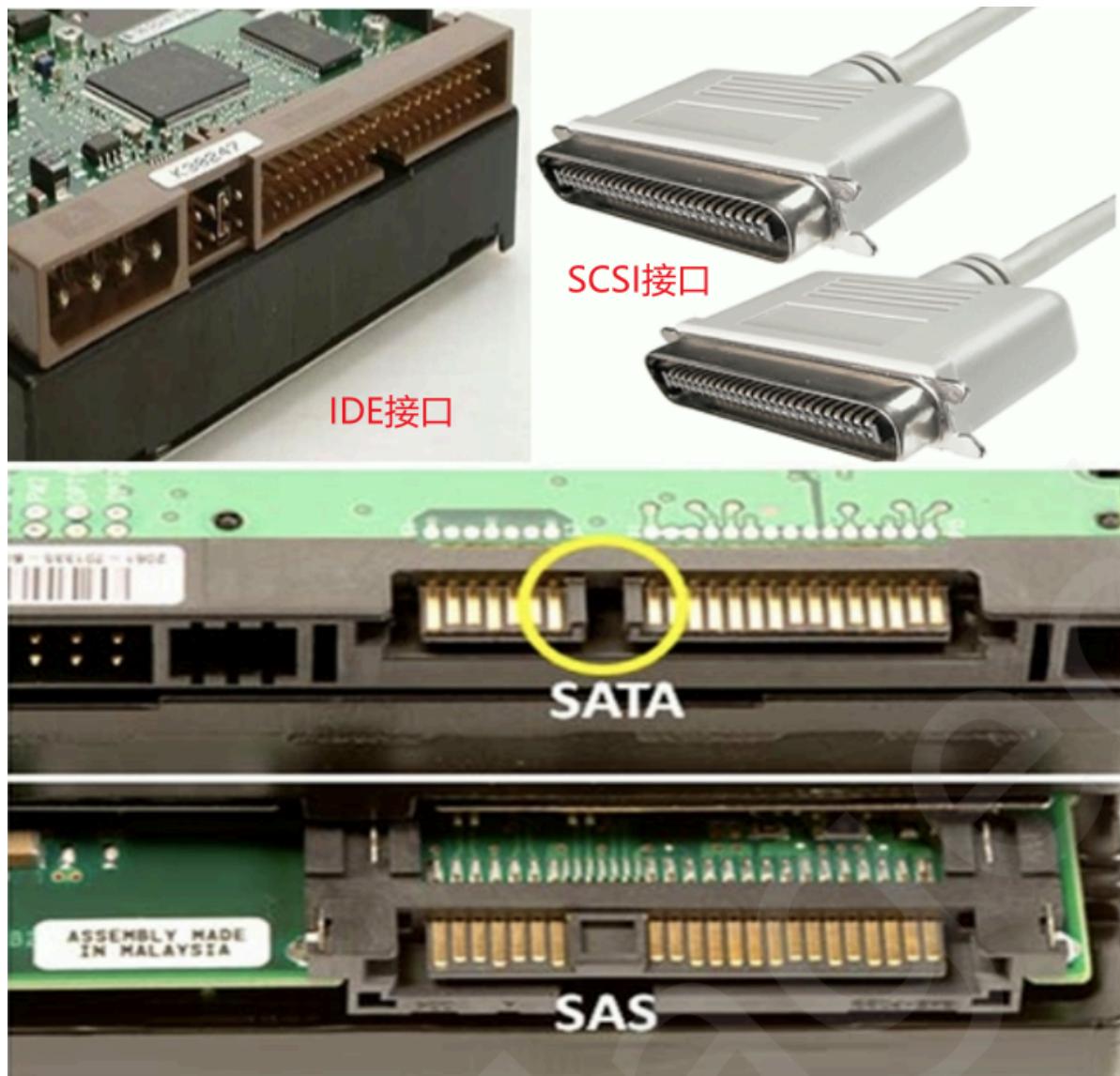
- 常用选项

- **bs=**大小: 设置块大小，例如 **bs=512** 表示每次读写512字节。bs 对性能有很大影响。
- **count=**数量: 复制指定的块数量，与 bs 一起使用可以控制复制的总数据量。
- **skip=**数量: 跳过输入文件开头的指定块数量。
- **seek=**数量: 跳过输出文件开头的指定块数量，用于在指定位置开始写入。
- **conv=**转换选项: 指定数据转换选项，如 **conv=notrunc** 避免截断输出文件，**conv=noerror** 在读取错误时继续处理。
- **status=**进度选项: 控制进度报告的输出，**status=progress** 显示处理过程中的进度信息。

- 示例

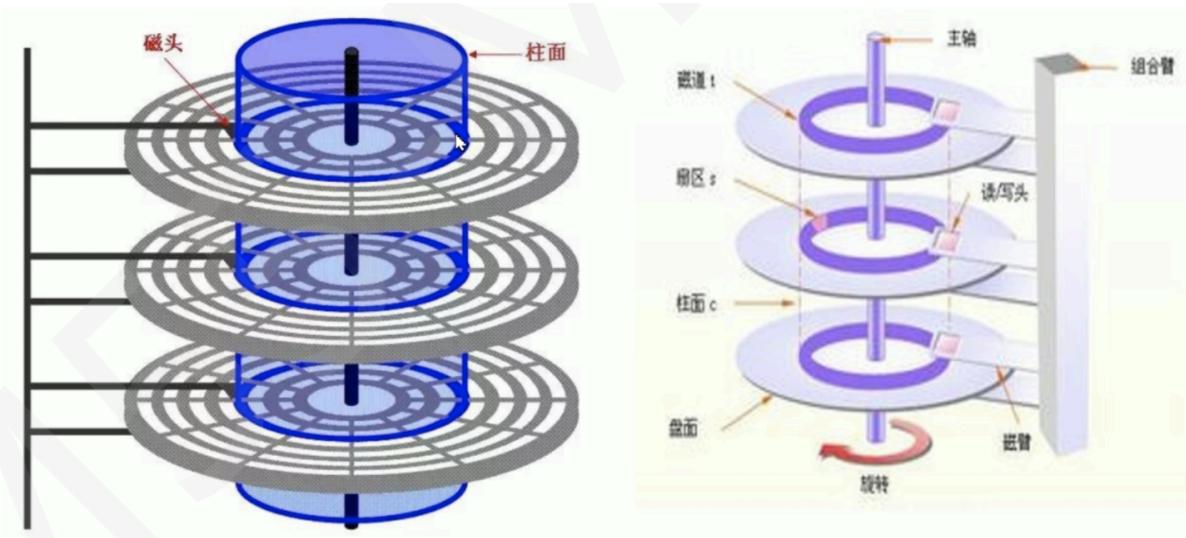
```
# 创建一个固定大小的文件  
dd if=/dev/zero of=文件名 bs=1G count=1  
  
# 备份和恢复硬盘分区  
dd if=/dev/sda1 of=/path/to/backup.img  
  
# 将备份文件恢复到分区  
dd if=/path/to/backup.img of=/dev/sda1  
  
# 复制光盘到ISO文件  
dd if=/dev/cdrom of=/path/to/cdimage.iso
```

## 硬盘类型



- 硬盘接口类型
  - IDE: 133MB/s, 并型接口, 早期家用服务器
  - SCSI: 640MB/s, 并行接口, 早期服务器
  - SATA: 6Gbps, SATA数据端口与电源端口是分开的, 即需要两条线, 一条数据线, 一条电源线
  - SAS: 6Gbps, SAS是一整条线, 数据端口与电源端口是一体化的, SAS中是包含供电线的, 而 SATA中不包含供电线。SATA标准其实是SAS标准的一个子集, 二者可兼容, SATA硬盘可以插入SAS主板上, 反之不行
  - USB: 480MB/s
  - 注意: 速度不是由单纯的接口类型决定, 支持Nvme协议硬盘速度是最快的

## 硬盘存储术语



- 硬盘存储术语CHS
  - head: 磁头; 磁头数=盘面数
  - track: 磁道, 磁道=柱面数
  - sector: 扇区, 512bytes
  - cylinder: 柱面 1柱面=512sector数/trackhead数=512\*32\*55=7.84M
- CHS(已基本淘汰)
  - CHS采用24bit位寻址
  - 其中前10位表示cylinder, 中间8位表示head, 后面6位表示sector
  - 最大寻址空间8G (8G计算方式:  $2^{24}=16M$ , 这个表示扇区的数量, 一个扇区512byte, 所以最大存储是 $16M*512=8G$ )
  - 对于机械硬盘, 外层磁道的数据, 读取更快, 因为单位时间内, 外层磁头划过的更长, 读取数据更多, **因此一种优化方式, 是把数据尽量存放到外磁道扇区, 对于机械硬盘, 数字越小, 扇区越靠外**
- LBA (Logic block addressing)
  - LBA是一个整数, 通过转换成CHS格式完成磁盘具体寻址
- 查看扇区信息

```
fdisk -l /dev/sda
# 默认只有扇区信息
Disk /dev/sda: 200 GiB, 214748364800 bytes, 419430400 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt # (分区方式)
Disk identifier: E6097981-0CC8-4229-AC57-CFFCAF3BC758

Device      Start      End    Sectors  Size Type
/dev/sda1    2048     4095     2048    1M BIOS boot
/dev/sda2    4096   4198399    4194304    2G Linux filesystem
/dev/sda3  4198400 419428351 415229952 198G Linux filesystem
root@mystical:/dev#
```

```

# 加参数可以得到更详细的信息
fdisk -u=cylinder -l /dev/sda
Disk /dev/sda: 200 GiB, 214748364800 bytes, 419430400 sectors
Disk model: VMware Virtual S
Geometry: 255 heads, 63 sectors/track, 26108 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
Disklabel type: gpt
Disk identifier: E6097981-0CC8-4229-AC57-CFFCAF3BC758

Device      Start    End   Size Type
/dev/sda1        1      1    1M BIOS boot
/dev/sda2        1    262    2G Linux filesystem
/dev/sda3     262 26109 198G Linux filesystem

```

## 分区类型

- Linux中使用磁盘的过程
  - 设备分区
  - 创建文件系统
  - 挂载该文件系统
- 分区的原因
  - 优化I/O性能
    - 比如：1000G空间中找一个文件和40G中找一个文件，明显40G中搜索查找更容易
  - 实现磁盘空间的配额限制
  - 提高修复速度
  - 隔离系统和程序
  - 安装多个操作系统
  - 采用不同的文件系统
- 分区方式
  - MBR
  - GPT

## MBR分区

- 定义：Master Boot Record，1982年，使用32位表示扇区数，分区不超过2T
- 查询

```

fdisk -l /dev/sda
# Disklabel type: dos (表示MBR分区)

```

- 0磁道0扇区：512byte
  - 446bytes: boot loader启动相关
  - 64bytes: 分区表，其中每16bytes表示一个分区

```

# 分区表信息
fdisk -l /dev/sda

# 这些为GPT分区表信息
Device      Start      End      Sectors  Size Type
/dev/sda1    2048     4095     2048    1M BIOS boot
/dev/sda2    4096   4198399   4194304    2G Linux filesystem
/dev/sda3  4198400 419428351 415229952 198G Linux filesystem

# MBR分区表信息
Device      Boot      Start      End      Sectors  Size Id Type
/dev/sda1    *       2048    2099199   2097152    1G 83 Linux
/dev/sda2        2099200 419430399 417331200 199G 8e Linux LVM
# *（星号）表示该分区为活动分区，分区信息第一字节为80

```

- 2bytes: 55AA标识符
- MBR分区特点
  - MBR分区中一个硬盘最多有4个主分区，也可以3个主分区+1个扩展（N个逻辑分区）
  - MBR分区：主和扩展分区对应的1-4，/dev/sda1-4,逻辑分区从5开始，/dev/sda5(逻辑分区从扩展分区中分，但是sda4代表扩展分区，逻辑分区从sda5开始)
  - **MBR最多4个分区的原因**

因为分区表一共64字节，而一个分区16字节，因此**MBR**分区方式最多4个分区

#### ◦ **总分区不能超过2T的原因**

一个分区的信息共16字节，其中4字节起始位置，4字节记录结束位置，（因为**MBR**中，使用32位，4字节记录分区地址），因此一共能记录的数量是 $2^{32}=4G$ 个扇区数量，一个扇区512字节，因此，一个分区空间最多不能超过2T，否则分区表无法记录

#### ◦ **详解16字节分区表中分区信息**

1. 第一个字节80表示活动分区，00表示非活动分区
2. 9-12字节，分区起始LBA地址
3. 13-16字节，分区结束LBA地址

# 以80 04 01 04 fe c2 ff 00 08 00 00 00 00 20 00举例  
80表示该分区为活动分区

后面的数据表示磁盘扇区位置和内存逻辑位置的地址信息

#### ◦ **扩展分区记录逻辑分区的方式**

在 MBR 的0扇区中，确实有64字节用于存储分区信息，这部分被分成4个分区条目，每个分区条目占用16字节。如果硬盘使用扩展分区，其中一个分区条目将用来定义扩展分区的起始位置（通常是在哪个扇区开始）。

这16字节的扩展分区条目指向第一个 EBR (Extended Boot Record)，这个 EBR 存在于扩展分区的起始位置。每个 EBR 也有自己的小型分区表，其中包括两个条目：

第一个条目定义了一个逻辑分区的起始位置和大小。

第二个条目指向下一个 EBR (如果有的话)，实际上是一个指向扩展分区中下一个逻辑分区的 EBR 的相对位置。

这样就形成了一种链表结构：

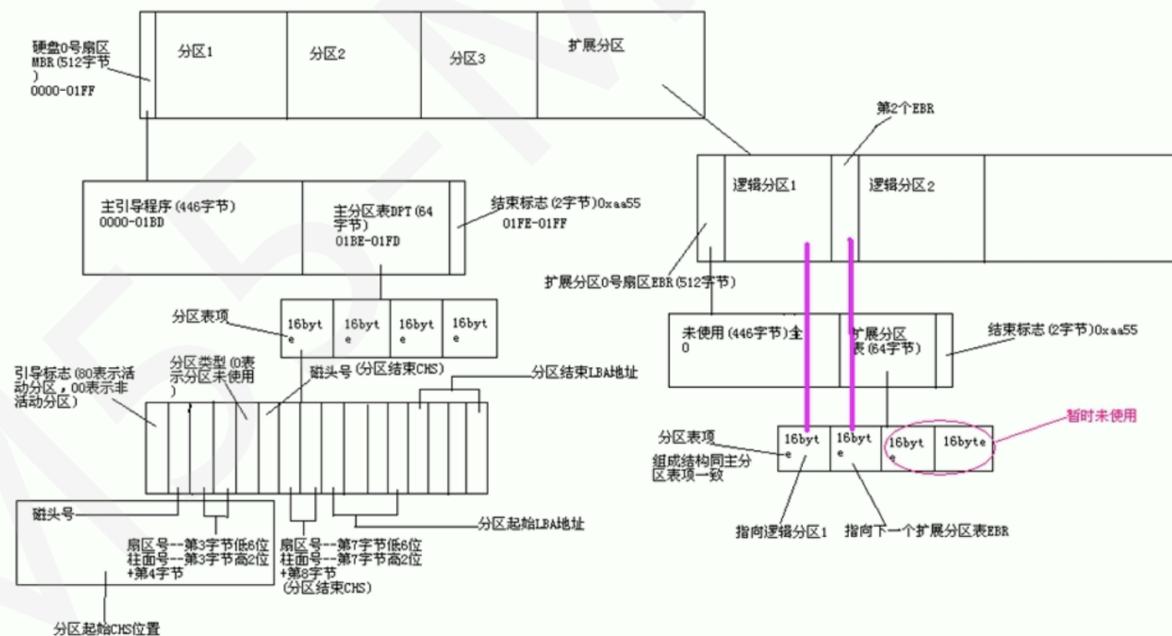
MBR 的扩展分区条目指向第一个 EBR。

第一个 EBR 定义第一个逻辑分区，并指向下一个 EBR。

第二个 EBR 定义第二个逻辑分区，并指向下一个 EBR。

以此类推，直到最后一个 EBR，其通常不再指向任何 EBR，表示链表的结束。

## MBR硬盘分区结构图



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	

- 详细解读

- 主引导程序 (偏移地址0000H--0088H) , 它负责从活动分区中装载, 并运行系统引导程序
- 出错信息数据区, 偏移地址0089H--00E1H为出错信息, 00E2H--01BDH全为0字节
- 分区表 (DPT,Disk Partition Table) 含4个分区项, 偏移地址01BEH--01FDH, 每个分区表项长16个字节, 共64字节为分区项1、分区项2、分区项3、分区项4
- 结束标志字, 偏移地址01FE--01FF的2个字节值为结束标志55AA

## MBR分区表破坏与修复实验

- MBR分区表破坏

```
# 备份分区表(将数据备份到mbrtb.img文件)
dd if=/dev/sda of=/home/mystical/mbrtb.img bs=1 count=64 skip=446

# 将备份数据放到远程服务器保存(保存到服务器10.0.0.150)
scp 10.0.0.164:/home/mystical/mbrtb.img .

# 将分区表的64byte部分, 填充0
dd if=/dev/zero of=/dev/sda bs=1 count=64 seek=446

# Ubuntu分区修复
1. 关闭服务器, 进入Bios, 使用CD-ROM drive加载引导程序

2. 选择Try or Install Ubuntu Server进入安装界面
3. 按ctrl+alt+f2 进入救援模式的命令行
4. 将之前备份在远程服务器的备份文件复制到待修复的服务器中
scp root@10.0.0.150:/home/mystical/mbrtb.img

5. 加载过来后, 将数据恢复重启即可
dd if=mbrtb.img of=/dev/sda bs=1 count=64 seek=446
fdisk -l /dev/sda # 看到分区表信息, 即证明修复成功

6 使用bios复原加载启动项, 然后重启即可

# -----
# Rocky分区修复

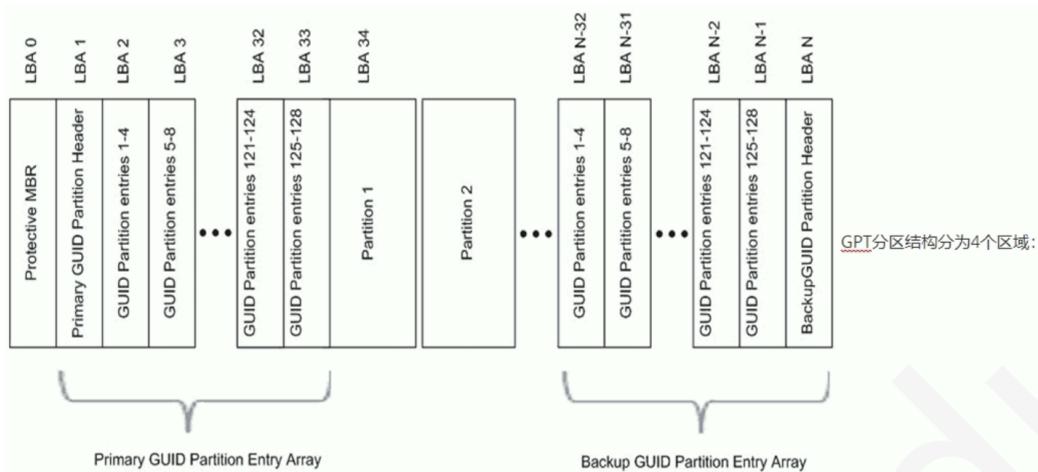
# 从光驱启动进救援模式
1. 使用CD-ROM drive启动, 进入安装界面后, 选择Troubleshooting
2. 选择Rescue a Rocky Linux System
3. 出现选择页面后, 选择1, 回车, 再回车, 拿到一个shell
4. 当前没有网络, 需要先配置网络, 然后远程将分区备份拷贝过来
ip address add 10.0.0.160/24 dev ens160
scp root@10.0.0.157:/root/mbrtb.img

5. 然后使用备份数据修复磁盘分区表后, 调整加载启动项, 重启即可
```

## GPT分区

- GPT:
  - 定义: GUID (Global Unique Identifiers) Partition table 支持128个分区, 使用64位, 支持8Z (512Byte/block) 64Z(4096Byte/block)

- 使用128位UUID (Universally unique Identifiers) 表示磁盘和分区，GPT分区表自动备份在头和尾两份，并有CRC校验位
- UEFI (同一可扩展硬件接口) 支持GPT，是的操作系统可以使用
- GPT分区结构



GPT分区结构分为4个区域：

- GPT头
- 分区表
- GPT分区
- 备份区域

## BIOS和UEFI

- BIOS：BIOS是固化在电脑主板上的程序，主要用于[开机系统自检](#)和[引导操作系统](#)。目前新式的电脑基本上都是UEFI启动

## 管理分区

- 列出块设备lsblk

```
lsblk

# 列出设备和挂载地址的完整路径
lsblk -p

# 列出每个分区的操作系统
lsblk -f

# 常用字段
NAME      # 设备名称
MAJ:MIN   # 主设备号:次设备号
RM        # 是否是可移动设备
SIZE      # 设备容量大小
RO        # 是否是只读设备
TYPE      # 设备类型
MOUNTPOINT # 挂载点
```

- 创建分区的命令

```
fdisk      # 管理MBR分区  
gdisk      # 管理GPT分区  
parted    # 高级分区操作，可以是交互或非交互方式  
  
partprobe  # 重新设置内存中的内核分区表版本，适合除CentOS 6之外的其他版本
```

## parted命令

- parted的操作都是实时生效的，没有交互式确认
- 由于parted命令，回车后直接生效，比较危险，因此生产中使用的比较少
- 格式：

```
parted [OPTION]... [DEVICE [COMMAND [PARAMETERS]...]]...  
  
# 常用选项  
-l | --list          # 显示所有硬盘分区信息  
-s | --script         # 不输出提示信息  
  
# 常用子命令  
help [COMMAND]        # 显示命令帮助  
print                # 显示  
quit                 # 退出（交互式操作时使用）  
mklabel|mktab le LABEL-TYPE # 指定磁盘的分区类型gpt|msdos(mbr)  
mkpart PART-TYPE [FS-TYPE] START END   # 新建分区，指定分区类型，文件系统，开始结束位置  
（默认单位是M）  
rm NUMBER            # 删除指定分区
```

- 示例：

```
root@ubuntu2204~# parted -l  
Model: VMware, VMware virtual s (scsi)  
  
Disk /dev/sda: 215GB  
Sector size (logical/physical): 512B/512B  
Partition Table: gpt  
Disk Flags:  
  
Number  Start   End     Size   File system  Name  Flags  
1       1049kB  2097kB  1049kB          bios_grub  
2       2097kB  2150MB  2147MB  ext4  
3       2150MB  215GB   213GB  
  
#-----  
  
Error: /dev/sdb: unrecognised disk label  
Model: VMware, VMware virtual s (scsi)  
Disk /dev/sdb: 21.5GB  
Sector size (logical/physical): 512B/512B  
Partition Table: unknown  
Disk Flags:
```

```
Model: Linux device-mapper (linear) (dm)
Disk /dev/mapper/ubuntu--vg-ubuntu--lv: 106GB
Sector size (logical/physical): 512B/512B
Partition Table: loop
Disk Flags:

Number  Start   End     Size   File system  Flags
 1       0.00B  106GB  106GB  ext4

#-----#
Warning: Unable to open /dev/sr0 read-write (Read-only file system).  /dev/sr0
has been opened read-only.
Error: /dev/sr0: unrecognised disk label
Model: NECVMWar VMware SATA CD01 (scsi)
Disk /dev/sr0: 2133MB
Sector size (logical/physical): 2048B/2048B
Partition Table: unknown
Disk Flags:
```

- 创建分区

```
# 创建分区表种类(msdos|gpt)
root@ubuntu2204~# parted /dev/sdb mklabel gpt
Information: You may need to update /etc/fstab.
# 只有将/etc/fstab文件同步更新，才能永久生效

root@ubuntu2204~# parted /dev/sdb print
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size   File system  Name   Flags
#-----#
# 创建分区
root@ubuntu2204~# parted /dev/sdb mkpart primary 1 300
Information: You may need to update /etc/fstab.

root@ubuntu2204~# parted /dev/sdb print
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size   File system  Name   Flags
 1       1049kB  300MB  299MB           primary
# 这里从1M即1024KB开始，但实际从1049的原因：
```

```
# parted 会默认对齐分区到硬盘的最优写入边界，这通常是硬盘的物理块大小的整数倍，或者对于更现代的硬盘来说，是最佳I/O性能的位置。这个对齐通常是为了性能考虑，确保分区边界与底层存储介质的物理布局相匹配，可以提高存取效率并减少潜在的读写错误。
```

```
root@ubuntu2204~# parted /dev/sdb mkpart primary 301 500
Information: You may need to update /etc/fstab.
```

```
root@ubuntu2204~# parted /dev/sdb print
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:
```

Number	Start	End	Size	File system	Name	Flags
1	1049kB	300MB	299MB		primary	
2	301MB	500MB	199MB		primary	

```
#-----
```

```
# 删除分区
root@ubuntu2204~# parted /dev/sdb rm 1
Information: You may need to update /etc/fstab.
```

```
root@ubuntu2204~# parted /dev/sdb print
Model: VMware, VMware Virtual S (scsi)
Disk /dev/sdb: 21.5GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:
```

Number	Start	End	Size	File system	Name	Flags
2	301MB	500MB	199MB		primary	

## 分区工具fdisk,gdisk

- fdisk

```
fdisk /dev/sdb

# p 打印分区表
Command (m for help): p
Disk /dev/sdb: 20 GiB, 21474836480 bytes, 41943040 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xe3be2582

# n 创建一个新的分区
# d 删除一个分区
删除主分区，磁盘编号不变
删除逻辑分区，磁盘编号可能会变，保证是从5往下依次顺延
```

```
# t 更改分区类型
```

```
# w 保存退出
```

```
# q 不保存退出
```

- gdisk

```
gdisk和fdisk用法基本相同
```

- 如果出现分区表信息分区后未更新，执行partprobe

```
partprobe # 更新分区表信息
```

- 非交互式创建分区

```
echo "n\np\nn\nn+2G\nw" | fdisk /dev/sdb
```

## 管理文件系统

- 文件系统

- 概念：文件系统是操作系统用于明确存储设备或分区上的文件的方法和数据结构；即在存储设备上组织文件的方法
- 操作系统中负责管理和存储文件信息的软件结构称为文件管理系统，简称文件系统
- 从系统角度来看，文件系统是对文件存储设备的空间进行组织和分配，负责文件存储并对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，安全控制，日志，压缩，加密等。

- ext4和xfs文件系统的比较

```
ext4最大支持16GB到16TB的文件，分区最大支持1EB
```

```
xfs最大支持8EB的文件，最大支持分区8EB
```

```
# 指 XFS 文件系统可以理论上管理和存储的最大数据量。
```

- 查看当前内核支持的文件系统

```
# rokcy8.6
ls /lib/modules/`uname -r`/kernel/fs
```

```
# ubuntu22.04
ls /lib/modules/`uname -r`/kernel/fs
```

- 查看当前系统可用的文件系统

```
cat /proc/filesystems
```

- 当前系统支持的文件系统和当前系统可用的文件系统是两回事，modules 中的文件系统在编译时选择了才是可用的，而可用的文件系统包含了默认支持的文件系统，如果需要使用某个文件系统，而该文件系统又不在proc 中，则需要重新编译内核；

对这句话的详细解读：

这个概念可以分为三个关键点：

支持但不可用：如果 **Linux** 内核的源代码包含了文件系统A的支持，这意味着理论上内核可以处理文件系统A。然而，如果这个支持没有在您当前运行的内核版本中被编译进去（无论是作为内核的一部分，还是作为可以动态加载的模块），那么文件系统A虽然被支持，但在实践中是不可用的。

添加模块和重新编译内核：为了使文件系统A在您的系统中可用，您需要确保内核编译过程中包括了文件系统A的模块。这通常意味着您需要获取内核的源代码，配置内核以包括文件系统A的支持（可以选择直接编译进内核，或者作为模块编译），然后重新编译并安装这个定制的内核。

使用文件系统：一旦您的系统运行了包含了文件系统A支持的内核（无论是静态还是动态模块），您就可以正常创建、挂载和使用文件系统A了。

## 文件系统类型

- Linux常用文件系统

文件系统	备注
ext2	Extended file system 适用于那些分区容量不是太大，更新也不频繁的情况，例如/boot 分区
ext3	ext2 的改进版本，其支持日志功能，能够帮助系统从非正常关机导致的异常中恢复
ext4	ext 文件系统的最新版。有很多新的特性，包括纳秒级时间戳、巨型文件 (16TB)、最大1EB的文件系统，以及速度的提升
xfs	SGI，支持最大8EB的文件系统
swap	交换分区专用的文件系统
iso9660	光盘文件系统
btrfs	Oracle公司开发

- Windows常用文件系统

/

文件系统	备注
FAT32	最多只能支持16TB的文件系统和4GB的文件
NTFS	最多只能支持16EB的文件系统和16EB的文件
extFAT	

- Unix常用文件系统

文件系统	备注
FFS(fast)	
UFS(unix)	UFS是UNIX文件系统的简称，几乎是大部分UNIX类操作系统默认的基于磁盘的文件系统
JFS2	

- 网络文件系统

文件系统	备注
NFS	Network File System, 即网络文件系统
CIFS	

- 集群文件系统

文件系统	备注
GFS2	基于X86_64, 最大文件系统可到100TB
OCFS2(oracle)	

- 分布式文件系统

文件系统	备注
faststdFS	
ceph	不仅仅是一个文件系统，还是一个有企业级功能的对象存储生态环境
mooseFS	
mogileFS	
glusterFS	
Lustre	

- Raw
  - 裸文件系统，未经处理或者未经格式化产生的文件系统
- 文件系统的组成部分
  - 内核中的模块：ext4,xfs,vfat
  - Linux虚拟文件系统：VFS
  - 用户空间的管理工具：mkfs.ext4, mkfs.xfs, mkfs.vfat
- 详解文件系统的三个组成部分的功能
  - 内核中的模块
    - 这些模块实际上是文件系统的核心实现，它们直接在内核空间运行。每个模块支持特定的文件系统类型（如 ext4、xfs、vfat 等），处理实际的数据存储、目录结构、文件操作（打开、读写、关闭）、权限检查、元数据管理等。这些模块使得内核能够理解和操作存储在磁盘或其他存储设备上的文件和目录结构。

- ext4: 是目前Linux上广泛使用的日志文件系统，提供了高性能、大容量、文件系统恢复和扩展属性等特性。
- XFS: 是一种高性能的日志文件系统，最初由Silicon Graphics为IRIX系统开发，特别适用于大文件处理和高并发环境。
- vfat: 是FAT文件系统的扩展，兼容于早期的FAT16和FAT32，提供了长文件名支持，通常用于与Windows系统交换数据。
- Linux虚拟文件系统: VFS
  - VFS 是一个抽象层，**它为不同的文件系统提供一个统一的接口。这意味着无论底层使用的是哪种文件系统 (ext4、XFS、vfat 或其他)，上层应用和用户都使用相同的标准系统调用 (如 open、read、write 等) 来操作文件和目录。** VFS 负责将这些调用转发到相应的文件系统模块进行处理。它也处理文件描述符、挂载点、缓存等核心概念，确保文件系统操作的高效和一致性。
  - VFS 的设计允许 Linux 系统同时支持多种文件系统，且能够灵活地挂载和使用它们，这是 Linux 系统灵活性和强大功能的基础之一。
- 用户空间的管理工具: mkfs.ext4, mkfs.xfs, mkfs.vfat
  - **这些工具运行在用户空间，为用户和管理员提供创建、检查、修复和调整文件系统的能力。每种文件系统类型通常都会有相应的管理工具。** 这些工具使用内核提供的接口与特定的文件系统模块交互，执行各种管理任务。
  - mkfs.ext4、mkfs.xfs、mkfs.vfat: 这些是格式化工具，用于在磁盘或分区上创建新的文件系统。例如，mkfs.ext4 /dev/sda1 会在 /dev/sda1 分区上创建一个新的 ext4 文件系统。
  - fsck 类工具：用于检查和修复文件系统错误。
  - tune2fs、xfs\_admin 等：用于调整文件系统参数和属性。

## 文件系统选择管理

- 创建文件系统
  - mkfs命令

```
# 创建ext4
# mkfs.ext4 或者mkfs -t ext4
# 后接要指定硬盘分区
mkfs -t ext4 /dev/sdb1

# 创建xfs
mkfs.xfs 或者mkfs -t xfs
mkfs.xfs /dev/sdb1

# 示例
root@ubuntu2204~# mkfs.ext4 /dev/sdb1
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 524288 4k blocks and 131072 inodes
Filesystem UUID: 93003898-536c-4f9d-a1ba-6bdaaa3ac155
Superblock backups stored on blocks:
            32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
```

```
writing superblocks and filesystem accounting information: done
```

- 该内容的详细解读

- Creating filesystem with 524288 4k blocks and 131072 inodes:
  - 这表明您创建的文件系统包含524,288个4KB大小的块，以及131,072个inode。每个块是文件系统存储数据的基本单位，而inode是文件系统用来存储文件元数据（如文件大小、权限、修改时间等）的数据结构。  
总计存储空间为  $524,288 \times 4\text{KB} = 2,097,152\text{KB} = 2\text{GB}$   
 $524,288 \times 4\text{KB} = 2,097,152\text{KB} = 2\text{GB}$ 。
- Filesystem UUID: 93003898-536c-4f9d-a1ba-6bdadaa3ac155:
  - 这是文件系统的全局唯一标识符（UUID），用于在系统中唯一地标识这个文件系统。
- Superblock backups stored on blocks: 32768, 98304, 163840, 229376, 294912:
  - 这列出了一些包含超级块备份的块的位置。超级块是文件系统的一个重要部分，包含了文件系统的整体信息（如块的大小、块的总数、inode的总数、空闲块和inode的数目等）。
  - 在多个不同位置存储超级块的备份，是为了在主超级块损坏时可以恢复文件系统。
- Allocating group tables: done:
  - 分配组表完成。组表用于管理文件系统中的块和inode分组，有助于优化存储空间的使用和提高访问效率。
- Writing inode tables: done:
  - 写入inode表完成。每个inode表包含了一组inode的信息，这对于文件系统来说是核心数据结构。
- Creating journal (16384 blocks): done:
  - 创建了一个由16,384个块组成日志（Journal）。ext4文件系统支持日志功能，可以通过记录对文件系统所做更改的日志，来提高文件系统的可靠性和恢复能力。
- Writing superblocks and filesystem accounting information: done:
  - 写入超级块和文件系统的会计信息完成。这是在文件系统创建的最后阶段，确保所有重要的文件系统信息都被正确记录。
- SuperBlack超级快概念
  - 超级块是文件系统中的一个关键数据结构，它存储了描述整个文件系统状态的信息，如：
    - 块大小
    - 文件系统中块和inode的总数
    - 空闲块和inode的数目
    - 文件系统的挂载状态
    - 最后一次检查（fsck）的时间等
  - 每个文件系统都有一个超级块，位于文件系统的开始位置。由于超级块对文件系统的健康至关重要，因此在多个位置保留其备份是一种常见的做法，以防止数据损坏导致文件系统不可用。
- 关于 Superblock backups stored on blocks: 32768, 98304, 163840, 229376, 29491 的解读
  - 这表示文件系统创建时在指定的块位置保存了超级块的备份。这些位置是相对于文件系统开始处的块号。在这个例子中，超级块的备份被存储在块号 32768、98304、163840、229376 和 294912。
  - 如何查看超级块的位置

- 块号：这些数字直接表示了存储超级块备份的块的位置。在文件系统中，一个块是数据存储的基本单位，其大小通常是 4KB（这个大小可以根据文件系统的创建时的设置而变化）。因此，当你看到块号 32768，这意味着备份超级块存储在从文件系统开始处的第 32768 个块的位置。
- 理解块号的意义：要理解这些块号的物理意义，你需要知道块的大小。如果块的大小是 4KB，那么块号 32768 实际上指的是距离文件系统开始  $32768 \times 4\text{KB} = 128\text{MB}$  的位置。这意味着第一个备份超级块位于文件系统开始后的 128MB 处。
- 如何使用备份超级块
  - 在某些情况下，如果主超级块受损，你可以使用这些备份超级块来恢复文件系统。例如，e2fsck 是一个检查和修复 ext 类文件系统的工具，可以指定备份超级块来执行文件系统检查，如

```
e2fsck -b 32768 /dev/sda1
# 这条命令告诉 e2fsck 使用 /dev/sda1 分区上块号 32768 处的超级块备份来进行文件系统检查。
```

## 查看和管理分区信息

- blkid

```
# 查看块设备属性
blkid [option]... [DEVICE]

# 常用选项
-U UUID          # 根据指定ID来查找对应设备
-L LABEL         # 根据指定LABEL来查找对应设备
```

- e2label

```
# 管理ext4系列文件系统的LABEL
e2label DEVICE [LABEL]      # 卷标，后面可以通过这个名称挂载，类似分区别名？

# 清除label
e2label DEVICE ""
```

- findfs

```
# 查找分区
findfs [options] {LABEL,UUID,PARTUUID,PARTLABEL}=<value>

# 示例：
findfs UUID="XXXXXX"
findfs LABEL="XXXXXX"
```

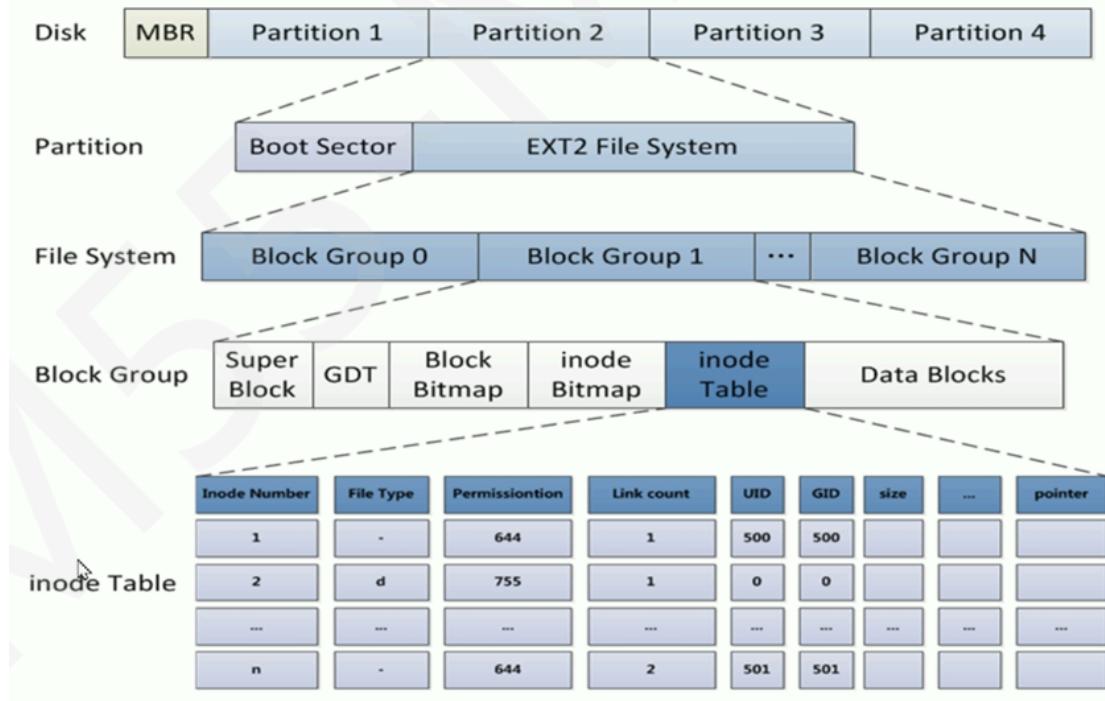
-tune2fs

```
# 查看ext4文件系统信息，无法查看xfs
tune2fs -l /dev/sdb1

# 常用选项
-l          # 查看指定文件系统信息
-L          # 修改卷标
-m N       # 修改预留root用户空间百分比，默认5%
```

## MBR分区和ext4文件系统的结构

- 超级块和inode table



- 块组(BLOCK GROUP)
  - 概念：将很多连续的块放到一个块组里
  - 查看块组命令——dump2fs

```
dump2fs /dev/sdb1
```

- Boot Sector和MBR的区别

**Boot Sector**, 也称为启动扇区, 是存储在存储设备上的一个特定区域, 通常位于硬盘或分区的最开始部分。它包含了计算机启动时所需的一些基本代码和系统启动信息。对于启动过程来说, **Boot Sector**是至关重要的, 因为它包含了启动加载程序 (**Bootloader**), 这是一个小程序, 它的任务是加载操作系统。

在一个物理存储设备中 (如硬盘), 可以有多个分区, 每个分区可以被视为独立的逻辑存储区域。理论上, 每个分区都可以有自己的**Boot Sector**, 尤其是当它们被配置为可启动分区时。这意味着, 如果你有一个硬盘被分成多个分区, 并且每个分区上都安装了操作系统, 那么每个分区的**Boot Sector**都包含了启动该分区上操作系统的引导代码。

\*\*主引导记录 (MBR) \*\*位于硬盘的最开始处, 它包含了一个小程序和分区表。这个小程序负责读取分区表, 找到被标记为活动 (或可启动) 的分区, 然后执行该分区的**Boot Sector**中的代码。

分区的**Boot Sector**位于分区的开始处, 它包含了启动分区上操作系统的具体代码。

因此, 虽然每个分区的开始都可以有一个**Boot Sector**, 但整个硬盘的最开始只有一个**MBR**, 其中包含了指向可启动分区的引导代码。这意味着**Boot Sector**并非只在第一个分区上有; 如果多个分区都被设置为包含操作系统, 则它们各自都会有自己的**Boot Sector**。

所以可以实现: 在虚拟机的一个硬盘上安装多个Linux系统, 每个系统独立于其他系统运行

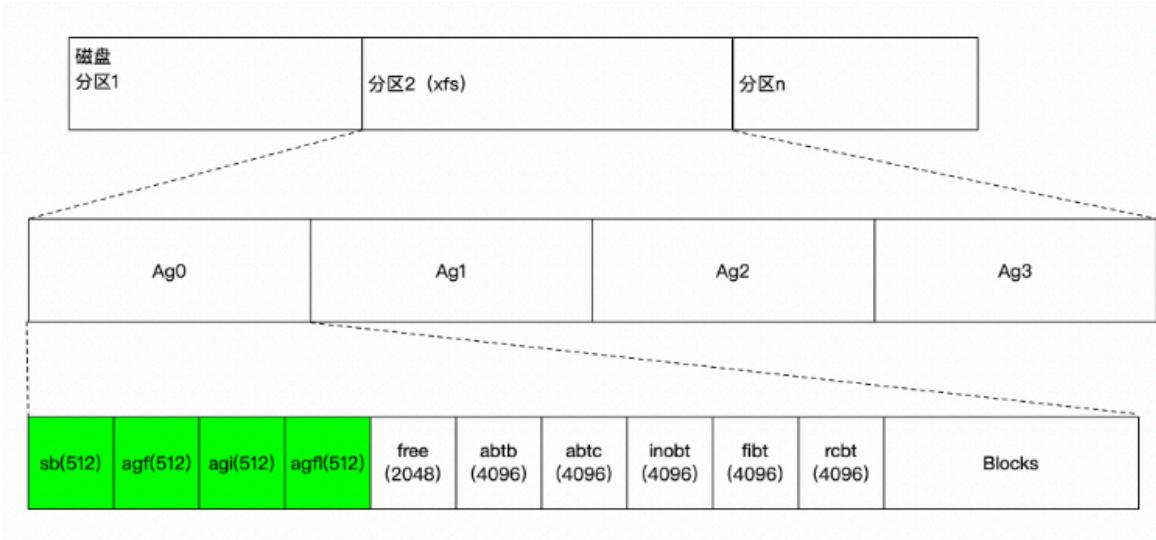
- 块组中的内容
  - Super Block
    - 超级块是文件系统中的一个关键数据结构, 它存储了描述整个文件系统状态的信息 (上文有详细说明)
  - 块组描述符表 (GDT)
    - ext文件系统每一个块组信息使用32字节描述, 这32个字节称为块组描述符, 所有块组的块组描述符组成块组描述符表GDT(group descriptor table)。虽然每个块组都需要块组描述符来记录块组的信息和属性元数据, 但是不是每个块组中都存放了块组描述符。将所有块组的块组信息组成一个GDT保存, 并将该GDT存放于某些块组中, 类似存放superblock和备份superblock的块
  - 块位图 (Block Bitmap)
    - 块位图是一个简单的数据结构, 用一系列的位 (bit) 来表示块组中每个块的状态——即它是空闲的还是已被占用。在块位图中:
      - 一个“0”位表示相应的块当前是空闲的;
      - 一个“1”位表示相应的块已经被占用。
    - 这个位图使得文件系统能够快速查找到块组中的空闲块, 当需要存储新数据时, 文件系统可以迅速定位到一个空闲的块, 并将其标记为已使用。
  - Inode 位图 (Inode Bitmap)
    - 和块位图类似, inode位图也是一个位图数据结构, 用来跟踪块组中的每个inode的状态。在inode位图中:
      - 一个“0”位表示相应的inode是空闲的;
      - 一个“1”位表示相应的inode已经被占用。
    - 这使得文件系统能够迅速找到空闲的inode来存储新文件或目录的元数据信息。
  - inode table
  - Data Blocks

### # Inode表中数据块映射的原理和实现

小文件：对于小文件，它的数据通常直接存储在 inode 中指向的数据块中。如果这些数据块足够存放文件的内容，则这些数据块可能位于同一个块组中。

大文件：对于较大的文件，文件内容可能会分布在多个数据块中，这些数据块可能位于不同的块组中。当文件内容超过直接块指针可以引用的大小时，文件系统会使用间接指针、二级间接指针，甚至三级间接指针来存储额外的数据块位置。这意味着一个大文件的数据块可以分散在文件系统的多个块组中。

## XFS文件系统结构



## 超级块和分配组

我们可以使用xfs\_info命令查看一个xfs文件系统的superblock信息

```
[root@ubuntu2204 /]#xfs_info /dev/sdb4
meta-data=/dev/sdb4          isize=512    agcount=4, agsize=655360 blks
                           =           sectsz=512   attr=2, projid32bit=1
                           =           crc=1     finobt=1, sparse=1, rmapbt=0
                           =           reflink=1  bigtime=0 inobtcount=0
data      =           bsize=4096   blocks=2621440, imaxpct=25
           =           sunit=0     swidth=0 blks
naming    =version 2          bsize=4096   ascii-ci=0, ftype=1
log       =internal log       bsize=4096   blocks=2560, version=2
           =           sectsz=512   sunit=0 blks, lazy-count=1
realtime =无                  extsz=4096   blocks=0, rtextents=0
```

superblock简称sb，我们可以看到显示的内容中包括很多当前文件系统的关键信息

- isize=512: 一个inode占用的空间大小
- bsize=4096: 一个block大小。由此可知当前文件系统中一个block可以存放8个inode
- agcount=4: ag是xfs文件系统的最初级组织结构，全称是allcation group。一个xfs文件系统是由若干个ag组成的。可以在格式化xfs的时候指定ag个数，默认一般是4个ag
- agsize=1310656: 每个ag所包含的block块数
- sectsz=512: 磁盘扇区大小
- inobt=1: 自 Linux 3.16 起，XFS 增加了 B+树用于索引未被使用的 inode。这是这个开关是否打开的选项。1表示打开

- blocks=5242624: 分区包含的所有block总数。

我们还可以使用xfs\_db 命令（交互式命令）查看sb的内容

```
xfs_db /dev/sdb4
xfs_db> sb
xfs_db> p
magicnum = 0x58465342
blocksize = 4096
dblocks = 2621440
rblocks = 0
rextents = 0
uuid = 36e5c9a8-0ff3-4bd6-bd06-294f5bb060d7
logstart = 2097158
rootino = 128
rbmino = 129
rsumino = 130
rextsize = 1
agblocks = 655360
agcount = 4
rbmblocks = 0
logblocks = 2560
versionnum = 0xb4a5
sectsize = 512
inodesize = 512
inopblock = 8
fname = "\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000"
blocklog = 12
sectlog = 9
inodeilog = 9
inopblog = 3
agblklog = 20
rextslog = 0
inprogress = 0
imax_pct = 25
icount = 64
ifree = 61
fdblocks = 2618848
frextents = 0
uquotino = null
gquotino = null
qflags = 0
flags = 0
shared_vn = 0
inoalignmt = 8
unit = 0
width = 0
dirblklog = 0
logsectlog = 0
logsectsize = 0
logsunit = 1
features2 = 0x18a
bad_features2 = 0x18a
features_compat = 0
features_ro_compat = 0x5
features_incompat = 0x3
```

```

features_log_incompat = 0
crc = 0x840bc531 (correct)
spino_align = 4
pquotino = null
lqn = 0x100000004
meta_uuid = 00000000-0000-0000-0000-000000000000

```

- rootinto=128：本文件系统第一个inode编号，一般这个inode就是这个文件系统的第一目录对应的inode编号
- agblocks=1310656：每个ag中的block个数
- ictcount=2432：目前已经分配的inode个数。这里要注意的是，与ext3/4文件系统不同，xfs的inode是动态分配的，所以这里的个数会随着文件个数的变化而变化
- ifree=136：表示inode的空闲空间
- xfs中inode总数等于ictcount+ifree

文件系统第一块除了sb占用前512字节外，后续还保存了其他相关本ag的重要数据结构信息，依次为：

- agf：ag本身的头部信息，包含了本ag的很多block索引和相关重要信息
- agi：ag本身的头部信息。包含了本ag的很多inode索引的相关重要信息，具体内容可在内核源代码中参考 `xfs_agi_t` 数据结构的定义。
- agfl：ag的freelist结构信息。具体可以参考内核源代码中的 `xfs_agfl_t` 结构体定义。

每一组ag的第一块中都包含sb, agf, agi, agfl四个结构，每个结构占用512字节。不同的是除了第一个ag的sb以外，其他的sb都用作备份。其他的ag中的agf、agi和agfl都记录各自的索引信息

第一块的后2048字节是未被占用的

- 在XFS文件系统中，每个分区（或更准确地说，每个XFS文件系统）由多个Allocation Groups (AGs) 组成。Allocation Group是XFS设计的核心概念之一，它将文件系统的存储空间划分为若干个较小、管理上相对独立的区块
- Allocation Group(AG)
  - Superblock
    - AG Superblock：每个Allocation Group都有自己的superblock，它存储了关于该AG的元数据，比如AG的大小、空闲块数量、空闲inode数量等。这个AG级别的superblock是整个文件系统superblock的一个子集，它允许文件系统在需要时只访问特定AG的信息，提高效率。
  - Inode Allocation Section
    - Inode表：管理该AG内的文件和目录的inode。inode包含了文件的元数据，如文件大小、权限、所有者、时间戳以及实际数据块的位置信息。通过在每个AG中独立管理inode，XFS能够提高文件创建和访问的速度。
  - Free Space Management
    - 空闲空间索引：XFS使用两种主要的数据结构来管理AG内的空闲空间——空间树（Space Trees）和空闲列表（Free Lists）。这些结构帮助XFS快速地找到足够大的连续空闲块来存储新文件或文件扩展。
  - Directory Block Allocation
    - 目录块：AG中还包含了目录的数据块，这些块存储了文件系统中目录的结构和信息。XFS优化了目录操作的性能，特别是在包含大量文件和子目录的目录中。

### 1. 目录项 (Directory Entries)

每个目录项代表目录中的一个文件或子目录，包含以下信息：

名称：文件或目录的名称。

Inode号：指向文件或目录inode的指针，inode中存储了关于文件的元数据，如大小、权限、所有者、时间戳以及数据块的位置等。

### 2. 目录块 (Directory Blocks)

目录块是存储目录项的物理单位。XFS为每个目录维护一个或多个目录块，具体取决于目录项的数量和大小。为了高效管理，这些目录块可以按需增长或缩减。

### 3. B树索引 (B-tree Index)

对于包含大量目录项的目录，XFS使用B树或B+树索引来组织目录块，这样可以加速目录项的查找过程。这种索引机制允许XFS以对数时间复杂度进行目录项的查找、添加和删除操作。

叶节点 (Leaf Nodes)：包含实际的目录项数据。在B树中，所有的目录项都存储在叶节点中。

内部节点 (Internal Nodes)：包含指向子节点的指针，用于导航B树。在较大的目录中，内部节点帮助快速定位到包含特定目录项的叶节点。

### 4. Extents

目录数据（包括目录块）可能被存储在连续的磁盘块中，这些连续块被称为extents。使用extents可以减少磁盘寻址时间，并提高读写效率。

- Extent Allocation Trees

- Extent树：文件和目录的实际数据是通过extents（一系列连续的块）来存储的。Extent分配树记录了文件数据块的分配情况，使文件系统能够有效地管理大文件的存储，同时减少碎片。

- 关于xfs更多详细知识，以及数据修复

- <https://zorrozou.github.io/docs/xfs/XFS%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F%E7%BB%93%E6%9E%84.html>

- 查看xfs信息

```
xfs_info /dev/sdb2
```

## 文件系统检测和修复

- ext4检测和修复

```
# fsck, 修复文件系统
fsck [options] -- [fs-options] [<filesystem> ...]
fsck.FS_type
fsck -t FS_type
```

```
# 示例
fsck /dev/sdb1 -y
# -y自动按yes, ext4可选, fsck可以自动识别ext文件系统
fsck.ext4 /dev/sdb1 -y
```

```
# e2fsck 和fsck使用方法类似
e2fsck /dev/sdb1 -y
```

- 破坏文件系统的超级块与修复

```

# 将格式化文件系统的分区挂载到指定目录
mount /dev/sdb1 /mnt

# 复制两个文件到该目录
cp /etc/passwd /mnt/f1
cp /etc/passwd /mnt/f2

# 然后破坏文件系统
dd if=/dev/zero of=/dev/sdb1 bs=1M count=1

# 此时tunefs -l /dev/sdb1会提示超级块被破坏，目录下文件也丢失
root@ubuntu2204:/mnt# tune2fs -l /dev/sdb1
tune2fs 1.46.5 (30-Dec-2021)
tune2fs: Bad magic number in super-block while trying to open /dev/sdb1

# 修复
# 先取消挂载
umount /mnt

# 使用fsck修复
fsck.ext4 /dev/sdb1 -y

# 修复后，使用tune2fs -l /dev/sdb1可正常查看文件系统信息，修复成功

# 重新挂载
mount /dev/sdb1 /mnt

# 即可在该目录下，找到源f1,f2文件（不保证每次都成功，存在数据丢失风险）

```

- xfs文件系统专用的检测修复工具——xfs\_repair

```
xfs_repair /dev/sdb2
```

## 挂载设备

- 挂载：将额外文件系统与根文件系统某现存的目录建立起关联关系，进而使得此目录做为其它文件访问入口的行为
- 卸载：为解除此关联关系的过程
- 挂载点下原有文件在挂载完成后会被临时隐藏，因此，挂载点目录一般为空，进程正在使用中的设备无法被卸载
- 挂载规则
  - 一个挂载点同一时间只能挂载一个设备
  - 一个挂载点同一时间挂载了多个设备，只能看到最后一个设备的数据，其它设备上的数据将被隐藏
  - 一个设备可以同时挂载到多个挂载点
  - 通常挂载点一般是已存在空的目录

## 挂载文件系统mount

```
mount device dir(挂载路径)

# -a 自动挂载fstab中的挂载配置，但是只在删除和新添时有效，对修改无效
mount -a

# --source选项值
-L LABEL          # 同LABEL=label
-U UUID           # 同UUID=uuid
--source device

# -o选项值(option)
ro/rw             # 只读/读写
remount          # 重新挂载
```

## 卸载

```
umount dir
umount device
# 效果相同
# 当有进程占有挂载空间时，无法卸载
```

## 查看挂载

```
mount          # 使用mount命令查看
cat /etc/mtab    # 通过查看/etc/mtab文件显示当前已挂载设备
cat /proc/mounts # 查看内核追踪到的已挂载的所有设备
```

## 查看挂载点情况

```
findmnt [options] <device> | <mountpoint>

# 默认查看整个系统挂载树
findmnt

# 查看指定设备或挂载点
findmnt <device> | <mountpoint>

# 查看正在访问指定文件系统的进程
lsof MOUNT_POINT
fuser -v MOUNT_POINT

# 终止所有正在访问指定的文件系统的进程
fuser -km MOUNT_POINT
```

## 永久挂载

- 直接 mount <device> <MOUNT\_POINT> 挂载点重启后会丢失
- 将挂载保存到 /etc/fstab 中可以下次开机时，自动启用挂载
- 手动启动挂载的方式

```
# 对fstab新添或删除  
mount -a  
  
# 对挂载项进行修改  
mount -o remount MOUNT_POINT
```

- /etc/fstab文件中的配置(一共六个字段)

- 要挂载的设备或伪文件系统设备文件(LABEL=label | UUID=uuid | /dev/sda1)
- 挂载点：必须是事先存在的目录
- 文件系统类型：ext4, xfs, iso9660, nfs, none
- 挂载选项：defaults , acl, bind, ro, rw 等

现代操作系统大部分默认支持acl，不需要额外启用acl

bind 选项用于将一个已存在的目录挂载到另一个位置，相当于做了一个镜像或链接。这在组织文件、兼容性调整或容器技术中非常有用

应用场景：当你需要在系统中的另一个位置访问某个目录内容时，可以使用bind挂载。这使得相同的内容可以在不同的路径下被访问，而无需复制数据。

代码示例：/var/www /home/user/www none bind 0 0

- 转储频率：0 不做备份；1 每天转储；2 每隔一天转储
- 现代Linux中，通常为0，不备份

- fsck检查的文件系统的顺序：0 不自检；1 首先自检，一般只有rootfs才用；2 非rootfs 使用0

在操作系统启动时，会自动执行fsck检测值为1的文件系统，通常是根文件系统

## 管理swap空间

### SWAP介绍

- SWAP交换分区是系统RAM的补充，swap 分区支持虚拟内存。当没有足够的 RAM 保存系统处理的数据时会将数据写入 swap 分区，当系统缺乏 swap 空间时，内核会因 RAM 内存耗尽而终止进程。
- SWAP交换分区实现过程
  - 创建交换分区或文件
  - 使用mkswap写入特殊签名
  - 在/etc/fstab文件中添加适当的条目
  - 使用swapon -a激活交换空间

### 创建SWAP

- 启动SWAP

```
swapon [option]

# 常用选项
-a          # 激活/etc/fstab中的所有交换区
-s          # 显示所有swap信息
```

- 禁用SWAP

```
swapoff -a          # 禁用所有SWAP

# swapoff -a ---> swapon -a    可以用来刷新swap
```

- 创建SWAP分区

```
# 创建swap文件系统
mkswap /dev/sdb3

# 修改/etc/fstab文件，添加swap行

# 刷新，即禁用后再次启用
swapoff -a ---> swapon -a
```

- 查看swap分区

```
cat /proc/swaps
swap -s
```

- 以文件作为swap分区
- 永久禁用SWAP

```
#删除swap行
[root@ubuntu2204 ~]# sed -i.bak '/swap/d' /etc/fstab

#或注释swap行
[root@ubuntu2204 ~]# sed -i.bak '/swap/s@^@#@" /etc/fstab

#禁用swap，由于修改了配置文件，所以重启也不会有SWAP
[root@ubuntu2204 ~]# swapoff -a
```

## SWAP使用策略

- /proc/sys/vm/swappiness 的值决定了当内存占用达到一定的百分比时，会启用swap分区的空间使用规则

当内存使用率达到100-swappiness时，会启用交换分区  
简单地说这个参数定义了系统对swap的使用倾向，此值越大表示越倾向于使用swap。  
可以设为0，这样做并不会禁止对swap的使用，只是最大限度地降低了使用swap的可能性

- 修改交换分区的使用策略

```
#修改
[root@rocky86 ~]# vim /etc/sysctl.conf
vm.swappiness=0

#生效
[root@rocky86 ~]# sysctl -p
vm.swappiness = 0

# 手动修改该配置文件后也可以生效
root@ubuntu2204 ~]# cat /proc/sys/vm/swappiness
60
```

## 光盘自动挂载

- 手动挂载光盘

```
mount /dev/sr0 /mnt/
```

- 自动挂载

```
apt install autofs

# 查看autofs服务的状态
systemctl status autofs

# 修改配置文件
vim /etc/auto.master
# 将auto.master中的/misc /etc/auto.misc注释取消
/mnt /etc/auto.mnt

# 创建auto.mnt文件
cd -fstype=iso9660,ro,nodev,nosuid :/dev/sr0

# 重启autofs服务
systemctl restart autofs

# 此时在目录下就能看到自动挂载的光盘
ls /mnt/cd
```

## 磁盘常见工具

### 文件系统查看工具df

```
df [option]... [file]...

# 常见选项
-T          # 显示文件系统类型
-h          # 显示文件系统大小
-l          # 只显示本机的文件系统
```

## 目录统计工具du

```
du [option]... [file]...

# 常用选项
-s          # 只显示外层目录
-h
```

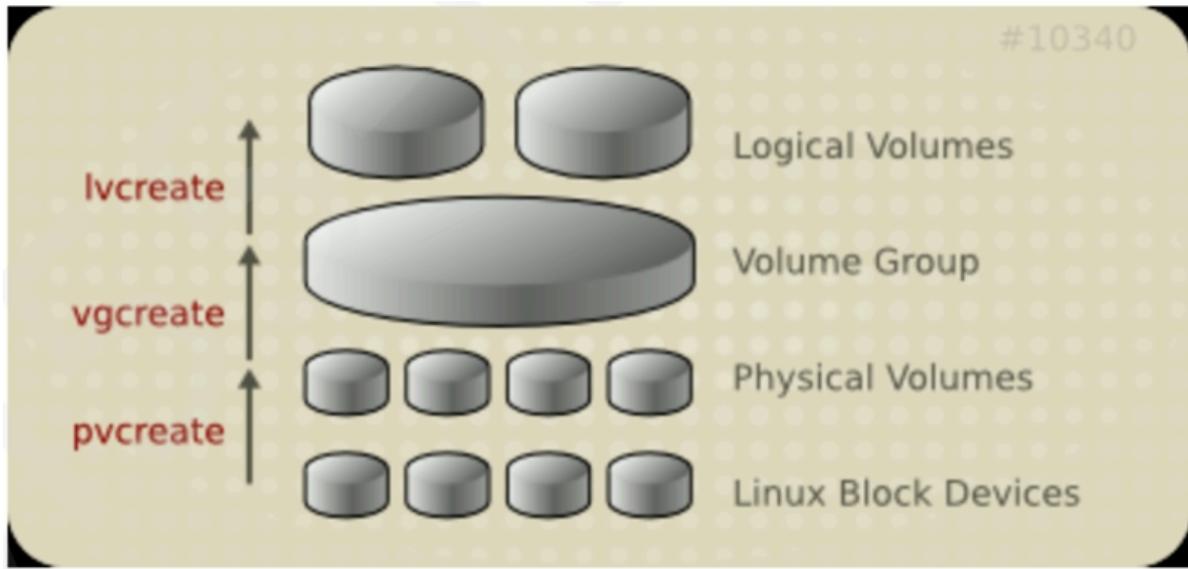
## RAID空间

- 在操作系统之外，进行磁盘阵列组合
- RAID-0
  - 数据同时向两个磁盘写
  - 大小为N\*min
- RAID-1
  - 镜像备份
- RAID-4
  - 最少3个磁盘
  - 其中一个磁盘是校验盘
- RAID-5
  - 最少3个磁盘
  - 校验数据分布在每个磁盘
- RAID-6
  - 最少4个磁盘
  - 双校验
- RAID-10
  - 先组RAID-1，再组RAID-0
- RAID-01
  - 先组RAID-0, 再组RAID-1
- RAID-50
- RAID-60
- JBOD
  - 单纯堆磁盘

## LVM管理

- LVM: Logical Volume Manager 可以允许对卷进行方便操作的抽象层，包括重新设定文件系统的大  
小，  
允许在多个物理设备间重新组织文件系统
- LVM可以弹性的更改LVM的容量

- 实现
  - 将设备指定为物理卷
  - 用一个或者多个物理卷来创建一个卷组，物理卷是用固定大小的物理区域（Physical Extent, PE）来定义的
  - 在物理卷上创建的逻辑卷，是由物理区域（PE）组成
  - 可以在逻辑卷上创建文件系统并挂载



- 第一个逻辑卷对应设备名：/dev/dm-#
- dm: device mapper, 将一个或多个底层设备组织成一个逻辑设备的模块
- 设备软连接：
  - /dev/mapper/VG-LV\_NAME
  - /dev/VG\_NAME/LV\_NAME
- 实现逻辑卷
  - 下载相关工具

```
apt install lvm2
```

- 将指定设备创建为物理卷Physical Volumes

- 查看物理卷

<b>pvs</b>	# 简要pv信息显示
<b>pvdisplay</b>	# 显示详细信息

- 创建物理卷

<b>pvcreate &lt;device&gt;</b>	#将指定设备格式化为物理卷
--------------------------------	---------------

- 删除物理卷

<b>pvremove &lt;device&gt;</b>	# 删除pv
--------------------------------	--------

- 卷组（Volume Group）管理

## ■ 显示卷组

```
vgs  
vgdisplay
```

## ■ 创建卷组

```
vgcreate [-s size] vgname pv1 [pv2...]  
# -s指定PE大小，类似于存储空间中的块的概念，
```

## ■ 管理卷组

```
# 卷组扩容，往卷组中增加新的物理卷  
vgextend vgname pv1 [pv2...]  
  
# 从卷组中移出物理卷  
vgreduce vgname pv1 [pv2...]
```

## ■ 删除卷组

```
# 删除vg之前，要先把对应的pv解除绑定（pvmove）  
vgremove vgname
```

### o lv (Logic volumes) 管理工具

## ■ 显示逻辑卷

```
lvs  
Lvdisplay
```

## ■ 创建逻辑卷

```
lvcreate {-L N[mMgGtT] -T N} -n NAME VolumeGroup  
  
# 常用选项  
-L N          # 指定大小  
-T N          # 指定PE大小，也可用百分比  
-n NAME       # 逻辑卷名
```

## ■ 扩展逻辑卷

```
lvextend {-L N[mMgGtT] -T N} LV_NAME  
  
# 常用选项  
-L [+SIZE[mMgGtT]]      # N个单位大小，也可以写成+10M  
-T [+Number[PE]]         # N个PE，也可以写成+10，表示在原基础上加10个PE大小  
(+100%free) 表示把剩下的空间都用完  
-r | --resizefs          # 自动重置文件系统大小
```

## ■ 缩减逻辑卷

```
# 逻辑卷缩减的前提是取消挂载
lvreduce {-L N[mMgGtT] -l N} LV_NAME
# xfs只支持缩减，不支持扩容

# 常用选项
-L|--size [-]SIZE[mMgGtT] # N个单位大小，也可写成-10M
-l|--extents [-]NUMBER[PE] # N个PE，也可以写成-10
```

#### ■ 删除逻辑卷

```
lvremove /dev/VG_NAME/LV_NAME
```

#### ■ 重设文件系统大小

```
# 修改了逻辑卷大小后，要同步文件系统
resize2fs [-f] [-F] [-p] [-P] [-M] # 只支持ext4文件系统
xfs_growfs /mountpoint
```

#### ■ 拆除卷组中的PC硬盘

```
pv
# 然后从卷组中取消
vgreduce <vgname> <device>

# 然后从物理卷中取消
pvremove <device>

# 然后就可以拔除硬盘
```

## LVM快照

### 逻辑卷快照原理

- 在生成快照时会分配给它一定的空间（小于等于原数据大小），但只有在原逻辑卷或快照有所改变时，才会使用这些空间（由于创建快照，最开始只是在卷组中分配一块空间，不会备份任何数据）因此快照创建速度快，瞬间完成。
- 快照的执行过程

逻辑卷快照是一种在特定时间点创建数据存储卷的副本的技术，通常用于备份或恢复目的。快照工作原理如下，以及对您提出的情况的解释：

**初始快照时刻：**当快照首次创建时，它会标记那一刻的文件系统状态，但不会立即复制数据。而是使用一种称为写时复制（**Copy-On-Write, COW**）的技术。这意味着快照初始时并不占用额外的空间。

**文件第一次改变（例如文件a）：**当原始逻辑卷中的文件（如文件a）在快照创建后第一次被修改时，快照机制会在将这些改动写入原始逻辑卷之前，将改动之前的原始数据（即文件a的原始状态）复制到快照空间中。这样，快照空间中保存了文件修改前的状态。

**后续改动同一文件（文件a）：**如果文件a在后续操作中继续被修改，快照空间不会更新。因为快照只保留了第一次修改之前的状态，即快照创建时或第一次修改前文件的状态。

**创建新文件（例如文件b）：**当在原始逻辑卷中创建新文件（如文件b）后，快照空间初始时不会改变，因为快照只关注快照创建时刻以前存在的数据。

**新文件后续更新（文件b）：**如果文件b在创建后发生了改动，且这是自文件b创建以来的第一次改动，则根据快照的工作原理，它的原始数据（即创建时的数据）会被复制到快照空间，因为这是对文件b的第一次修改。

**总结：**快照空间会包含所有在快照创建后首次修改的文件的原始状态。对于在快照创建之后新增的文件（如文件b），如果这些文件在快照创建后发生了修改，它们的原始状态（创建时的数据）也会被备份到快照空间，但只有在第一次修改时。

## 逻辑卷快照实现

```
# 前提条件：有一个逻辑卷/dev/testvg/lv3

# 在该逻辑卷上创建文件系统
mkfs.ext4 /dev/testvg/lv3

# 创建目录
mkdir /lv3{,_snapshot}

# 挂载
mount /dev/testvg/lv3 /lv3

# 写文件
向文件中写入文件

# 为该逻辑卷创建快照，创建前保证卷组有足够的空间
lvcreate -n lv3_snapshot -s -L 100M -p r /dev/testvg/lv3

# 选项解读
-n          # 指定逻辑卷快照的名称
-s          # 表示创建的逻辑卷是快照
-L          # 指定快照空间大小
-p r        # 表示该卷只读

# 挂载快照
mount /dev/testvg/lv3_snapshot /lv3_snapshot

# 利用快照恢复，逻辑卷快照是一次性的
# 将快照和源都取消挂载
umount /lv3; umount /lv3_snapshot
```

```
# 从快照中恢复  
lvconvert --merge /dev/testvg/lv3_snapshot  
  
# 将源逻辑卷挂载回去  
mount /dev/testvg/lv3 /lv3  
  
# 快照使用后，就没有了，一次性的
```

## 网络协议和管理

- OSI网络国际标准
  - 由ISO国际标准化组织定义
  - 应用层: Application
  - 表示层: Presentation
    - 实现资源的格式与转换, 包括压缩, 解压缩, 加密, 解密等
  - 会话层: Session
  - 传输层: Transport
    - 实现应用进程间的安全可靠的通信
  - 网络层: Network
    - 负责端到端的路径的选择, 同时实现跨网域通信
  - 数据链路层: Data Link
    - 根据目标和源的mac地址, 进行点与点之间的通信
  - 物理层: Physical
- Mac地址
  - 由48位二进制数或12位16进制数组成
  - 示例: 00:0c:29:21:ac:a1
- 网络通信的过程
  - 协议数据单元PDU: Protocol Data Unit, 协议数据单元是指对等层次之间传递的数据单位。
    - 物理层的PDU是数据位 bit
    - 数据链路层的PDU是数据帧 frame
    - 网络层的PDU是数据包 Packet
    - 传输层的PDU是数据段 segment
    - 应用层的PDU是message
  - 三种通讯模式
    - unicast: 单播: 目标设备是一个
    - broadcast: 广播, 目标设备是所有
    - multicast: 多播, 组播, 目标设备是多个
  - 冲突域和广播域
    - 冲突域: 两个网络设备同时发送数据, 如果发生了冲突, 则两个设备处于同一个冲突域, 反之, 则各自处于不同的冲突域。

- 广播域：一个网络设备发送广播，另一个设备收到了，则两个设备处于同一个广播域，反之，则各自处于不同的广播域。
- 三种通讯机制
  - 单工通信：只有一个方向的通信。比如：收音机
  - 半双工通信：通信双方都可以发送和接收信息，但不能同时发送，也不能同时接收，比如：对讲机
  - 全双工通信：通信双方可以同时发送和同时接收，比如：手机

```

mii-tool eth0(网卡名)
>>> eth0: negotiated 1000baseT-FD flow-control,link ok
<!!-- 可以查询到网卡是否连通 -->

ethtool eth0(网卡名)
<!!-- 能够返回网卡的状态 -->

```

## 局域网

- 标准：关注定义了数据链路层和物理层
- 局域网中主流的设备是交换机，如果需要连接外网，才需要路由器
- 集线器与交换机
  - 集线器(hub)：所有连接集线器得机器都在同一个冲突域中，也同时在同一个广播域中，所以集线器有很严重得安全隐患，因为，一台机器向另一个机器发送数据得时候，集线器会把数据广播给所有机器，工作在物理层
  - 交换机(二层设备)：只有第一次发送数据会产生广播，后续将机器mac地址和端口一一记录后，以后得数据发送，都是点对点得数据发送，避免了广播得问题
- 路由器router
  - 作用：把一个数据包从一个设备发送到不同网络里得另一个设备中（跨网域）。路由器只关心网络得状态和决定网络中的最佳路径。路由的实现依靠路由器中的路由表来完成。
  - 功能
    - 工作在网络层
    - 分隔广播域和冲突域
    - 选择路由表中到达目录最好的路径
    - 维护和检查路由信息
- 以太网技术
  - 概述：以太网(Ethernet)是一种产生较早且使用相当广泛的局域网，由美国Xerox(施乐)公司的Palo Alto研究中心(简称PARC)于20世纪70年代初期开始研究，并于1975年研制成功。
  - 以太网MAC帧格式

Field Length, in Bytes						Ethernet					
8	6	6	2	46-1500	4						
Preamble	Destination Address	Source Address	Type	Data	FCS						

- Preamble：8个字节前导信息（检测是否网络冲突）

- Destination Address: 6个字节的目标MAC地址
- Source Address: 6个字节的源MAC地址
- Type: 2个字节的要传给上层的协议类型
- Data: 46-1500字节的数据
- FCS: 4个字节的校验位
- MAC地址
  - 共6个字节，48位bit，前3个字节由网卡的厂商定义，后3个字节可由工厂自定义
  - 可以通过[standards-oui.ieee.org/oui/oui.txt](http://standards-oui.ieee.org/oui/oui.txt)查询各大厂商的mac前3字节的值
- 在虚拟机中，不同机器的相当于连接在集线器hub上，因此，可以在windows上通过抓包，把虚拟机之间的通信数据包抓取
- Linux中的抓包工具：tcpdump

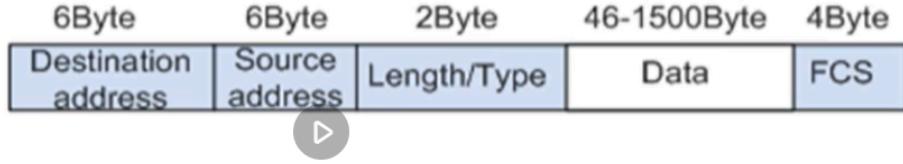
```
tcpdump -i eth0 -nn icmp
```

-i 指定抓取哪个网卡的包  
 -nn 表示ip以数字形式展示  
 icmp 表示抓取哪个协议的数据包

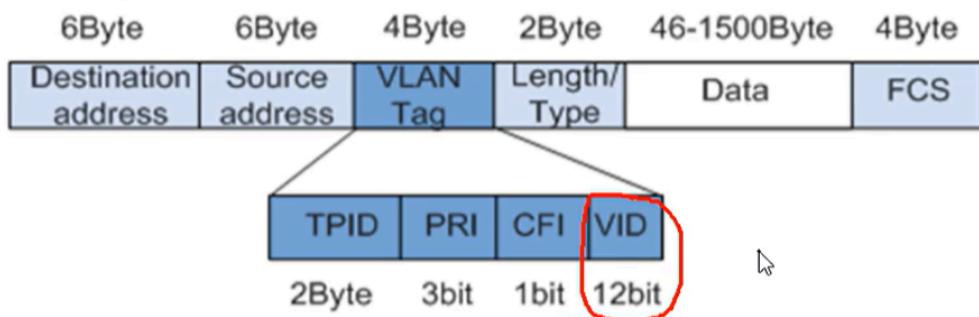
- VLan原理-虚拟局域网

- 虚拟局域网VLAN是由一些局域网网段构成的与物理位置无关的逻辑组，实实在在交换机内进行局域网隔离的效果
- IEEE802.1Q帧结构
  - 作用：在多个交换机之间的trunk干道中的数据包，带有vlan的标识，使其可以跨交换机实现vlan布局
  - 原理：在传统的以太网帧的里面添加一个vlan编号

传统的以太网数据帧



VLAN数据帧



- 在数据帧中，只预留了12位用来表示VLAN的编号，也就是说，最多可以创建4096个vlan，在当前的云环境中可能不够用，因此，出现了VXLAN技术
- VXLAN
- 云环境下，进行虚拟局域网隔离

# VXLAN

VXLAN解决的问题

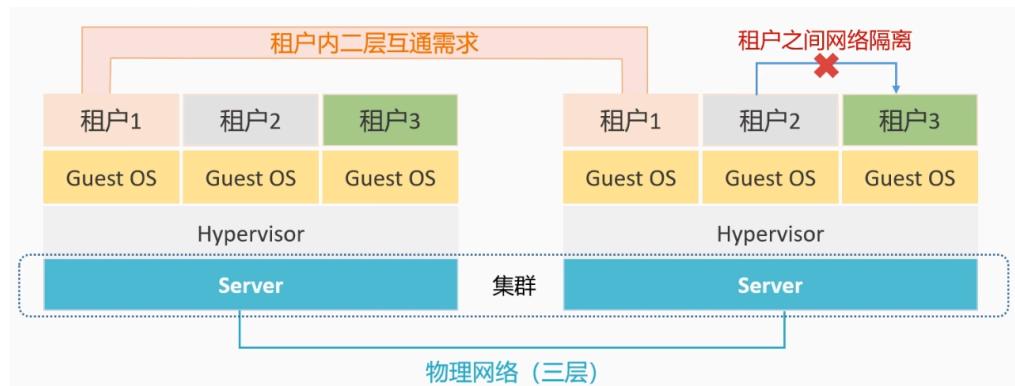
- 二层扩展

- 物理服务器可能分布在地理位置跨度非常大的机房，因此需要使用三层进行互联
- 而三层设备会隔离广播，如果设备有二层互通的需求，就需要使用VXLAN技术



- 多租户隔离

- 云化场景一般支持多租户，即不同用户共享物理资源。这对网络提出两个需求，租户间隔离和租户内互访
  - 租户间隔离：租户可能配置相同的MAC和IP地址，需要考虑网络承载隔离问题，并且存在海量的用户需要进行隔离
  - 租户内互访：租户内相同网段能够直接进行二层通信，即便处于不同物理位置的机房中。



传统网络面临的问题

- 虚拟机规模受设备表项规格限制

- 服务器虚拟化后，VM的数量比原有物理机发生了巨大的增长，而接入侧二层设备的MAC地址表规格较小，无法满足快速增长的VM数量

- 网络隔离能力限制

- VLAN Tag只有12bit (仅能表示4096个逻辑单元)
- 对于大型虚拟化云计算服务的场景而言，租户数目远大于VLAN可以个数
- 传统而成网络中的VLAN无法满足网络动态调整需求

- 虚拟机迁移范围受限

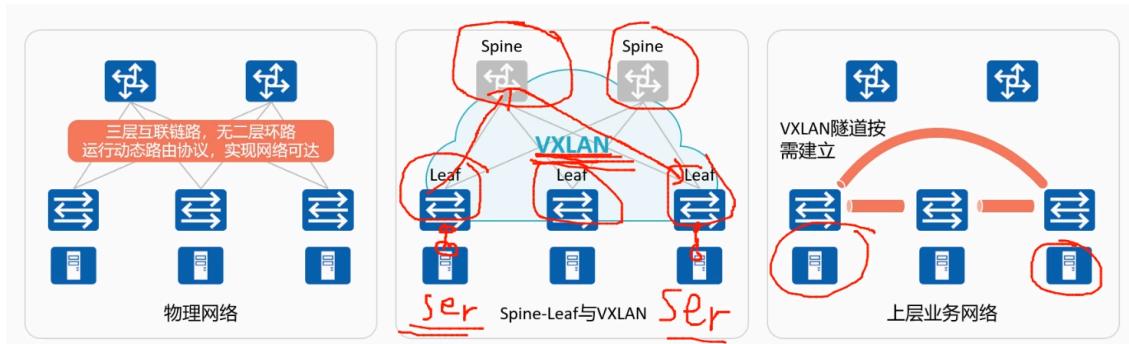
- 虚拟机迁移必须发生在一个二层网络中
- 传统的二层网络，将虚拟机迁移范围限制在了一个较小的局部范围内

## VXLAN简介

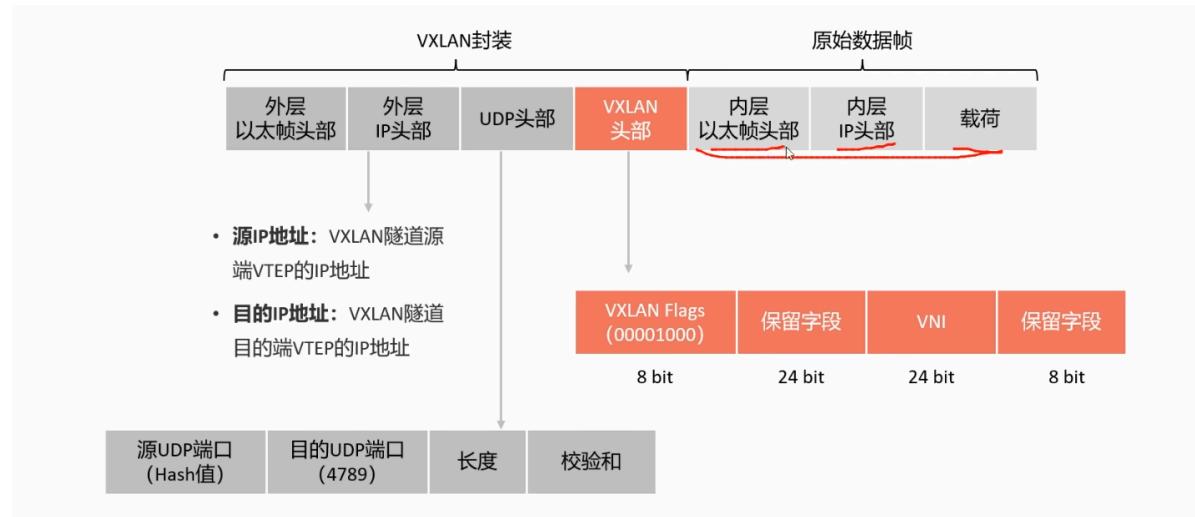
- VXLAN在本质上是一种VPN技术，能够在任意路由可达的物理网络（underlay网络）上叠加二层虚拟网络（Overlay网络），通过VXLAN网关之间的VXLAN隧道实现VXLAN网络内部的互通，同时，也可以实现与传统的非VXLAN网络的互通
  - 实质上是在arp报文上面加一个IP头部
- VXLAN通过采用MAC in UDP封装来延伸二层网络，将以太报文封装在IP报文之上，通过路由在网络中传输，无需关注虚拟机的MAC地址。且三层网络无网络结构限制，具备大规模扩展能力。通过路由网络，虚拟机迁移不受网络架构限制

## VXLAN在数据中心的应用

- 在数据中心中采用Spine-Leaf两层物理架构，结合VXLAN应用
- Spine节点执行路由转发，转发时不感知VXLAN。Leaf节点负责资源接入，完成VXLAN封装及解封装
- 这里Leaf上面会做关于VLAN到VXLAN的一个映射
- 数据中心的业务均由VXLAN承载



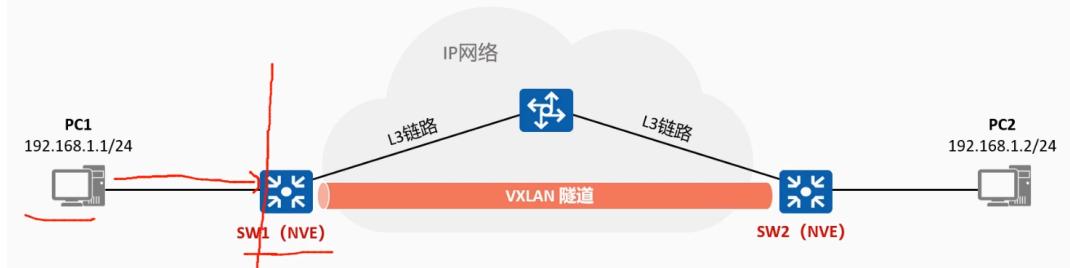
## VXLAN报文格式



## VXLAN基本概念

- NVE(Network Virtualization Edge, 网络虚拟边缘)
  - 是实现网络虚拟化功能的网络实体，可以是硬件交换机也可以是软件交换机

- NVE在三层网络上构建二层虚拟网络，是运行VXLAN的差别。图中SW1和SW2都是NVE



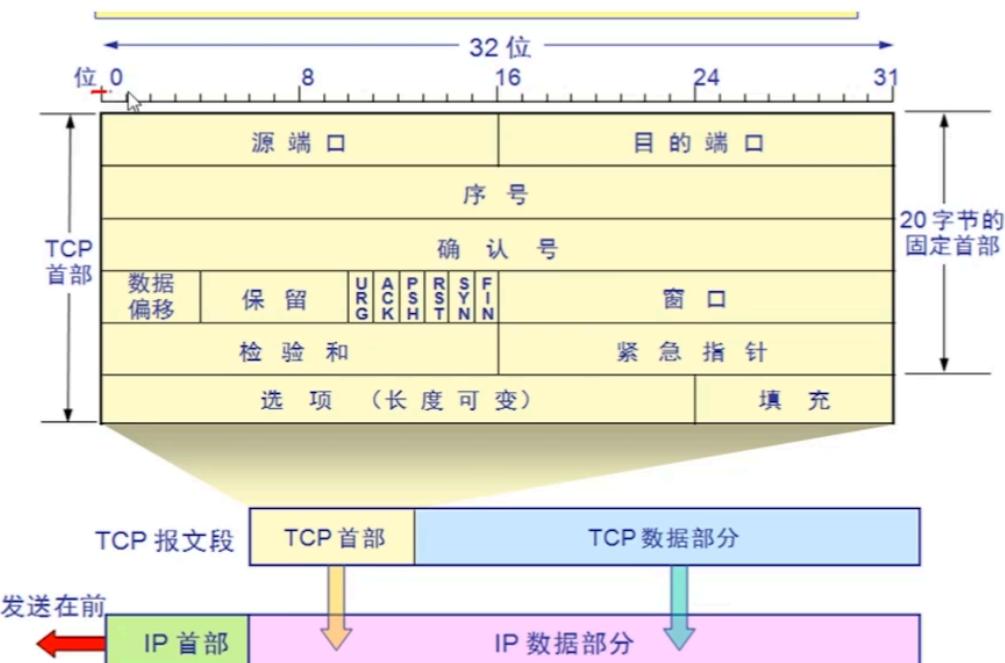
- VTEP(VXLAN Tunnel Endpoints, VXLAN隧道端点)

- VTEP是VXLAN隧道端点，位于NVE中，用于VXLAN报文的封装和解封装
- VXLAN报文(外层IP头部)中源IP地址为源端VTEP的IP地址，目的IP地址为目的端VTEP的IP地址



## TCP/IP协议栈

- TCP包头结构



- 端口号

- 0-1023: 系统端口或特权端口（仅管理员可用），众所周知，永久分配给固定的系统应用使用，22/tcp(ssh), 80/tcp(http), 443/tcp(https)
- 1024-49151: 用户端口或注册端口，但要求并不严格，分配给程序员注册为某应用使用，1433/tcp(SqlServer), 1521/tcp(oracle), 3306/tcp(MySQL)
- 49151-65535: 动态或私有端口，客户端随意使用端口，范围定义：/proc/sys/net/ipv4/ip\_local\_port\_range（可以通过重定向更改这个端口值得范围）
- 查看当前正在使用的端口号

```

<!-- 去检索 /etc/services -->
nc -l 1023
# nc: Permission denied
# 1023以内的端口，只有管理员可以使用
nc -l 1024
# 可以正常使用

grep <端口号> /etc/services
# /etc/services记录了著名程序使用的端口号

```

监听端口号的使用

```
ss -tnl
```

- 序号(4个字节):

- 表示本报文段所发送数据的第一个字节编号。在TCP连接中所传诵的字节流的每个字节都会按顺序编号。由于序列号由32位表示，所以每 $2^{32}$ 个字节，就会出现序列回绕，再次从0开始

- 确认号:

- 表示接收方期望收到的发送方下一个报文段的第一个字节数据的编号
- 通过序号和确认号，保证数据包可靠有序的发送
- 如果发送方A发出的数据包的序号为100，接收方B会响应该数据包，发给A的响应包中，确认号为101表示希望发送方A下次方的包的序号为101

- 数据偏移(4bit)

- 表示tcp头部的长度

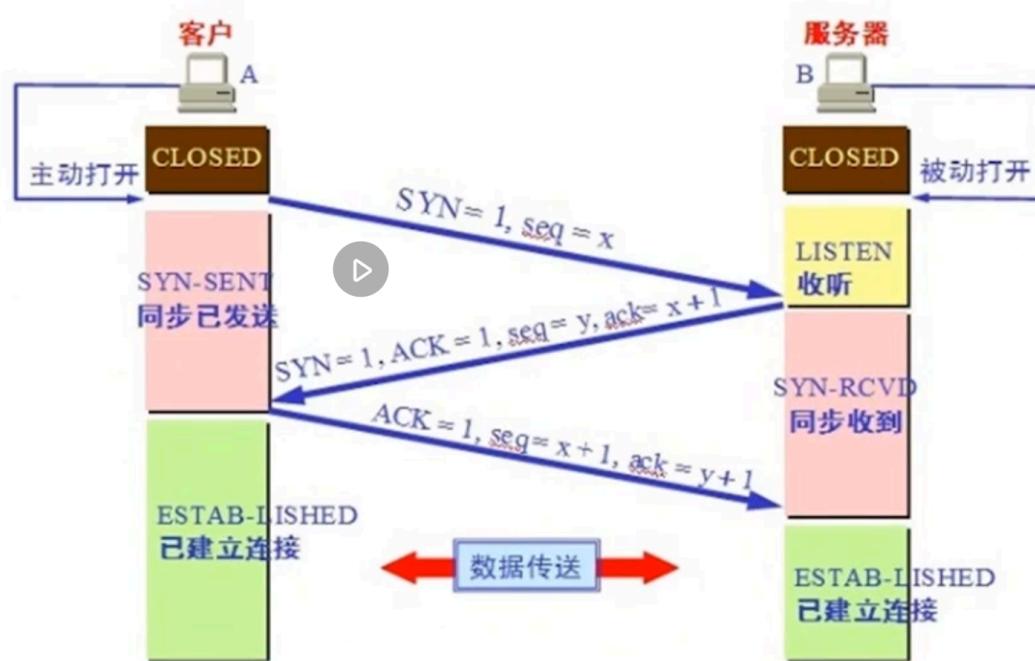
- 保留 (6bit) : 没用到的空间

- 关于tcp连接握手的重要的6位

- URG(紧急指针)
  - 如果URG为1表示紧急指针数据有效，为0则无效
- ACK
- PSH
  - 如果为1，表示接收到报文后不等待，立即发送给上层应用程序，如果为0，则报文会在缓冲区存放一段时间，在传给上层应用程序
- RST(重置位)
  - 如果是0,则正常通信，如果是1，表示报文有问题，需要重新发送
- SYN
- FIN

- 三次握手和四次挥手

### ○ 三次握手（建立连接）



- 第一次：客户端向服务端发送报文；此时， $SYN=1$ ,  $seq$ 即序列号为x
- 第二次：服务端收到报文，响应客户端，发送报文；此时， $SYN=1$ ,  $ACK=1$ , 序列号 $seq=y$ , 确认号 $ack=x+1$
- 第三次：客户端再次向服务端发送报文；此时 $ACK=1$ ,  $seq=x+1$ ,  $ack=y+1$
- 三次握手的原因：
  - 第一次握手，保证了A发送数据包，B接收数据包
  - 第二次握手，保证了A接收到B发送的响应包，证明了A端数据包的有去有回，但是此时无法保证B端的响应包能够有去有回，因此需要第三次握手
  - 第三次握手，A接收到B的响应包后，再次给B发送数据包，B接收数据包，实现了B端数据包的有去有回
  - 总结：因此，tcp面向连接，实现连接达成需要至少经历3次握手；核心在于，只有数据有去有回才能证明通信一端的达成
  - 除了数据包外，还要注意客户端和服务端的网络状态的改变，通过下面的命令可以查看到

```
ss -nalt
// 同时能够查看到处于监听状态的所有端口

ss -ntlp
lsof -i :<端口号>
// 能够查看到占用端口的具体程序
```

- 正常通信，SYN-SENT和SYN-RECV的状态，持续很短，很难看到
  - SYN-SENT观测到的方法：是服务端在接收到客户端发送过来的数据后，屏蔽掉客户端，不向客户端响应数据，这样客户端就会停在SYN-SENT的状态，不会向ESTABLISHED(建立连接的状态)转变

- SYN-RECV的观测方法：客户端不接受服务器端的响应，在服务器端在发送数据包后，由LISTEN状态变成SYN-RECV状态，后续由于没有收到客户端后续发送的数据包，因此维持下SYN-RECV的状态，不会向ESTAB转变
- 上述过程，需要使用到iptables的命令工具

- **三次握手过程中的窗口大小协商**

- 初始化窗口大小被称为接收窗口，它指定了接收方愿意接收的数据量，即发送方在等待接收方确认之前可以发送的最大数据量
- 第一次握手（SYN）：当客户端向服务器发起连接请求时，它会发送一个SYN（同步序列编号）包。这个SYN包中不仅包含客户端的初始序列号，还可以包含客户端支持的最大窗口大小

客户端 （SYN包） -----> 服务器

发送过程中，告诉服务器，我发的是第几个包，我能接收多少流量

- 第二次握手（SYN-ACK）：服务器收到客户端的SYN包，会回复一个SYN-ACK包，确认收到客户端的连接请求。这个SYN-ACK包同样包含服务器的初始序列号，并且会包含服务器的窗口大小，告知客户端服务器端愿意接收的数据量。

客户端 <----- (SYN-ACK) 服务器

响应过程中，服务器告诉客户端

我需要第几个包

我的包是第几个

我能接收多少流量

- 第三次握手（ACK）：客户端收到服务器的SYN-ACK包后，会发送一个ACK包作为响应，确认连接建立。这个ACK包也可以包含窗口大小的信息，但通常情况下，窗口大小是在前两个步骤中协调的
- 三次连接的意义
  - 第一次，第二次握手，可以证明服务端可以响应客户端的数据
  - 第二次，第三次握手，可以证明客户端可以响应服务端的数据

- **TCP滑动窗口（TCP Sliding Window）机制是在TCP三次握手建立连接之后发生的。这个机制是TCP流量控制和拥塞控制的关键部分，旨在优化网络通信的效率和可靠性。**

- 实现原理：

- TCP滑动窗口机制使用了窗口的概念来控制在任意时刻可以发送多少数据（未被确认的数据）。这个“窗口”是指发送方和接收方基于当前网络状况协商的数据量，可以理解为接收方告诉发送方在没有收到进一步确认前能接收多少字节的数据

- 运行过程

- **连接建立与窗口初始化：**在TCP三次握手过程中，发送方和接收方会初始化各自的发送和接收窗口大小。这个大小可以在后续的通信过程中根据网络条件进行调整
- **数据传输：**发送方基于当前的窗口大小发送数据。窗口大小表示发送方在接收到接收方的下一个确认（ACK）之前可以发送的最大数据量

数据传输：

客户端 -----> 服务端

客户端发送数据大小是由初始窗口大小决定

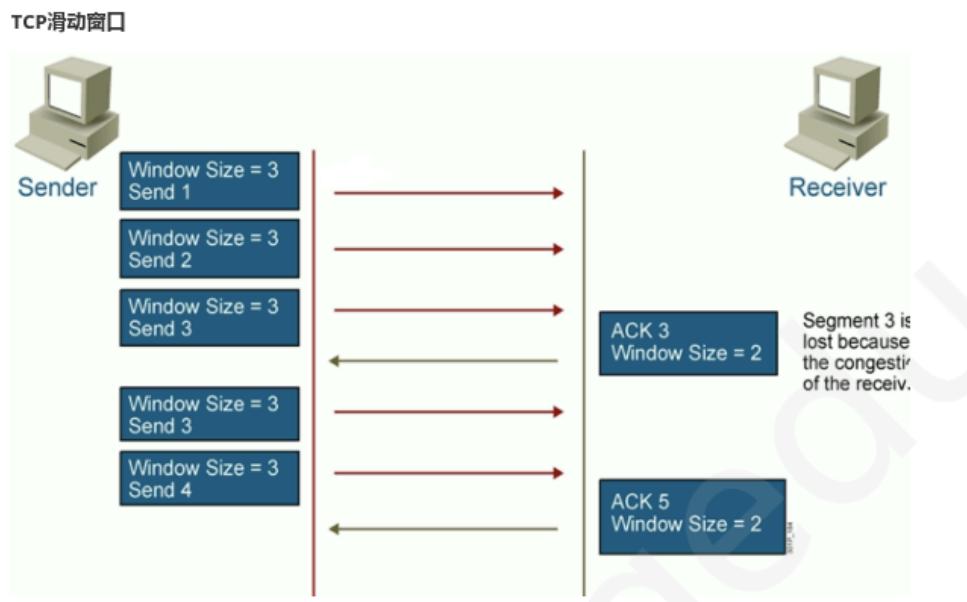
- **窗口滑动**: 当接收方收到数据后，他会发送一个确认 (ACK) 给发送方，并在确认消息中更新自己的接收窗口大小，告知发送方还可以接受多少新的数据。发送方在收到确认后，会根据接收方的窗口大小调整自己的发送窗口，并继续发送新的数据。这个过程中，窗口会根据确认的接收和新数据的发送“滑动”
- **流量控制**: 滑动窗口机制允许接收方控制发送方的数据发送速率，避免接收方的缓冲区被溢出。接收方可以通过调整其窗口大小（甚至将窗口设置为0）来减慢发送方发送的速度

在流量控制中，关注的是接收方的能力，确保发送方不会发送超过接收方缓冲区大小的数据。  
接收窗口大小 (**rwnd**) 是在TCP头部显示告知对方的，用于流量控制

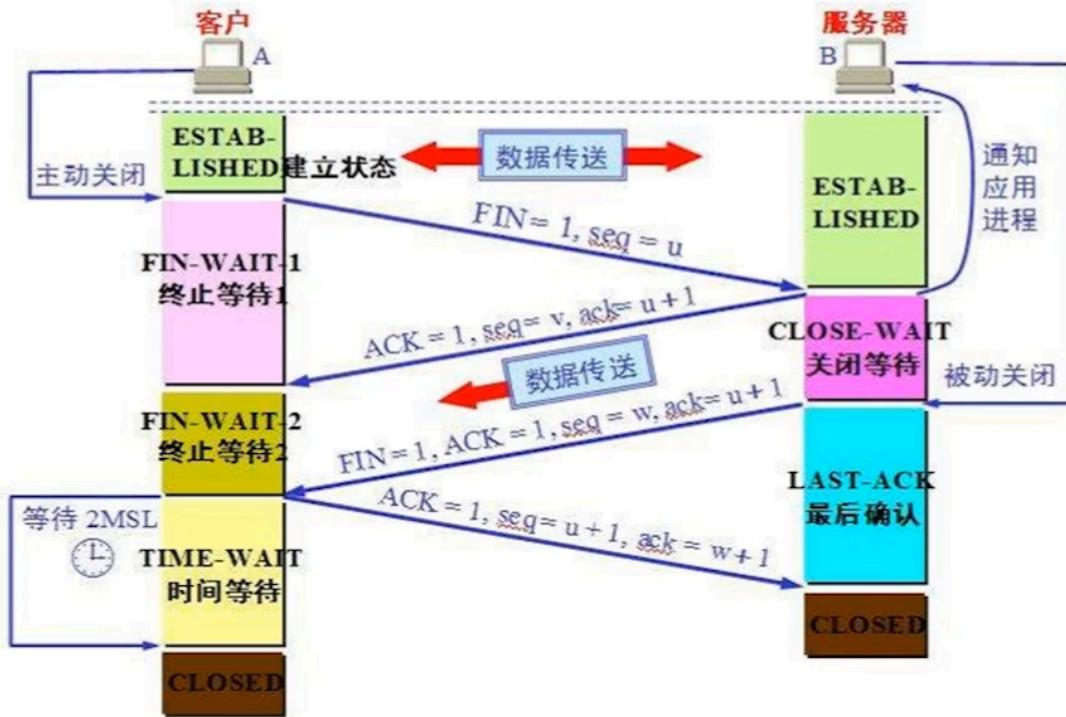
- **拥塞控制**: 除了基于接收方能力的流量控制外，TCP还实现了基于网络拥塞情况的窗口调整策略

拥塞控制使用的拥塞窗口 (**CWND**) 大小不通过TCP头部直接传递给接收方。  
发送方基于ACK返回的速率和可能出现的丢包等信号来评估网络状况，并调整**cwnd**的大小

- **实际发送窗口**: 实际的发送窗口大小由流量控制的接收窗口和拥塞控制的拥塞窗口中的较小值决定。



- 四次挥手（退出连接）



- 半连接队列和全连接队列

```
/proc/sys/net/ipv4/tcp_max_syn_backlog # 未完成连接队列大小，默认值128，建议调整大小为1024以上
```

```
/proc/sys/net/core/somaxconn # 完成连接队列大小，默认值128，建议调整大小为1024以上
```

- 半连接队列：在三次握手建立连接的时候，服务器收到了第一次握手时，客户端发来的数据包，并在第二次握手的时候进行了回复，在等待客服端再次发送数据包的服务器的第三次握手之前，会将这个时候的连接状态，放到syns queue的队列中，如果，服务器收到了第三次握手手中客户端发来的数据包，则会把半连接状态的数据从队列中清除，如果没收到第三次握手时，客户端发来的数据包，则此次半连接状态会一直缓存在syns queue即半连接队列中

#### “SYN洪水攻击”（SYN Flood Attack）

在 SYN 洪水攻击中，攻击者连续地发送大量的 SYN 请求到服务器，但故意不响应服务器的 SYN+ACK，从而不完成三次握手。因此，这些半建立的连接会占据半连接队列（SYN Queue）。如果这些请求足够多，会导致队列迅速填满。当半连接队列满时，新的正常用户请求也无法被放入队列，从而无法与服务器建立连接。

Linux 有一个特性叫做 SYN cookies，它是一个防止 SYN 洪水攻击的策略。当 SYN 队列快要满时，服务器可以选择不使用队列，而是为每个 SYN 包计算一个特殊的值（称为 SYN cookie）并发送回 SYN+ACK。这样，即使队列已满，服务器仍然可以响应新的连接请求。当服务器后续收到客户端的 ACK 时，它可以使用 SYN cookie 来验证并建立连接，而不需要查询 SYN 队列。

- 全连接队列：在三次握手完成后，服务器会将连接已建立的状态的信息数据放入accept queue即全连接队列中，用户在3次握手建立连接后，第四次发送数据包请求时，服务器端有一个accept()函数，来接收用户请求，接收到之后，会将accept queue队列中之前缓存的连接建立状态的数据包清除掉，如果accept()没有接收到第四次客户的请求，建立连接状态的数据包就会一直在accept queue中排队

连接泄露/连接耗尽攻击

攻击者通过正常地进行三次握手与服务器建立连接，但在连接建立后不进行实质性的数据交换或请求，从而占用服务器的资源。当攻击者建立了大量这样的“空闲”连接后，服务器的 ACCEPT 队列可能会被填满，导致正常用户无法与服务器建立新的连接。

简单地说，攻击者的策略是：

正常执行三次握手，与服务器建立连接。

在连接建立后，不进行实际的数据传输或请求，使连接保持空闲。

重复上述步骤，创建大量空闲连接，从而耗尽服务器的资源或填满 ACCEPT 队列。

这种策略的目的是耗尽服务器的资源（如文件描述符、内存）或填满 ACCEPT 队列，从而使得正常用户无法与服务器建立新的连接。与 SYN 洪水攻击不同，这种攻击已经完成了完整的三次握手，因此会被视为有效的连接，使得对其进行检测和防护更为困难。

为了防范此类攻击，以下是一些建议：

连接超时：为连接设置一个合理的超时，确保长时间未活跃的连接会被关闭。

限制源 IP 的连接数：可以配置防火墙或其他安全设备，限制来自单一源 IP 的并发连接数量。

监控：监控服务器的资源使用情况，如文件描述符、内存等。当资源使用接近上限时，可以及时采取措施。

负载均衡器和防火墙：使用负载均衡器或防火墙进行前端流量过滤，拒绝异常的连接请求。

- TCP超时重传

- 定义：异常网络状况下(开始出现超时或丢包)，TCP控制数据传输以保证其承诺的可靠服务

TCP服务必须能够重传超时时间内未收到确认的TCP报文段。为此，TCP模块为每个TCP报文段都维护一个重传定时器，该定时器在tcp报文段第一次被发送时启动。如果超时时间内未收到接收方的应答，TCP模块将重传TCP报文段并重置定时器。

至于下次重传的超时时间如何选择，以及最多执行多少次重传，就是TCP重传策略

与TCP超时重传相关的两个内核参数

```
/proc/sys/net/ipv4/tcp_retries1      # 指定在底层IP接管之前TCP最少执行的重传次数，默认是3
```

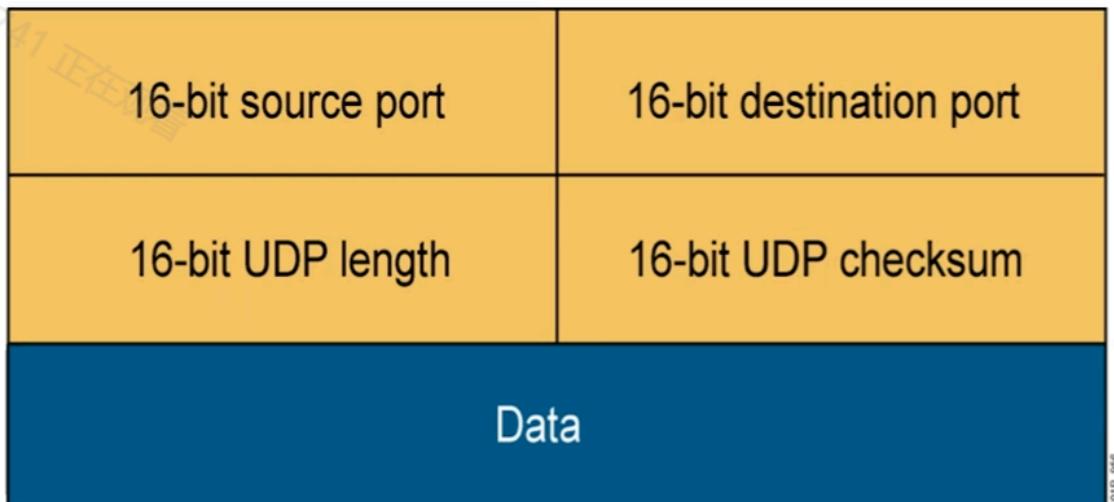
```
/proc/sys/net/ipv4/tcp_retries2      # 指定连接放弃前TCP最多可以执行的重传次数，默认值15(对应13-30min)
```

- 拥塞控制
- 内核TCP参数优化（建议后续深入了解）

## UDP协议(User Datagram Protocol)

- UDP特性
  - 工作在传输层
  - 提供不可靠的网络访问
  - 非面向连接协议
  - 有限的错误检查

- 传输性能高
- 无数据恢复特性
- UDP包头



## Internet层

### ICMP协议(Internet Control Message Protocol)

- 范例：利用ICMP协议判断网络状态

```
# 禁止本机被ping的配置文件
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all

ping 10.0.0.8
ping -f -s <ip地址> -c1
#-f 表示泛洪，就是最大功率无限制发送数据包
#-s 指定发送包的大小，最大不能超过65507
#-c(num) ping的次数
```

### ARP协议(Address Resolution Protocol)

- ARP协议（第二层协议）
  - 定义：APR地址解析协议，由互联网工程任务组(IETF)在1982年11月发布的RFC826中描述制定，是根据IP地址获取物理地址的一个TCP/IP协议
  - 作用：根据IP地址寻找对应的物理地址
  - 原理：显示发送一个包含有IP地址的请求广播给局域网的所有主机，对应IP的主机会返回一个数据包给发送请求的主机，该数据包中含有这个主机的MAC地址，交换机上记载着MAC地址对应的交换机接口，从而实现物理地址的寻址
  - 本质：在于IP地址到MAC地址的转换

```
查询ip地址与mac地址对应关系的指令
# arp -n
-- 只有发生过通讯，才会显示
-- 可以通过ping，来做实验，观察发生过通讯的虚拟机双方记录的ip和mac的对应关系
```

- 因为arp是通过广播实现的ip与mac地址的转换，因此，arp是不能跨网段的，因为路由器会屏蔽清除掉广播信息

- 判断广播的方法：数据链路层的目标地址是48个1，即fffff, 就说明是广播
- 详细过程：
  - 第一步：源主机广播发送一个请求数据包，询问谁是xxip,同时数据包携带源主机的IP和MAC地址
  - 第二步：目标主机收到数据包后，发现ip与自己一致，于是记录下源主机的IP和mac地址，同时向这个地址发送响应数据包，上面携带自己的ip和mac地址。注意!!! 这次目标主机对源主机的响应式单播
  - 第三步，源主机收到目标主机的响应包后，将目标主机的IP所对应的mac地址记录下来，实现ip到mac的转换
- ARP存在的安全问题
  - 由于是广播机制，同时没有任何校验，因此很容易被黑客进行arp欺骗，造成流量劫持，

```
KALI系统实现arp欺骗上网流量劫持

-- 启动路由转发功能
# echo 1 > /proc/sys/net/ipv4/ip_forward

-- 安装包
# apt-get install dsniff

-- 欺骗目标主机，本机是网关
# arpspoof -i eth0 -t 被劫持的目标主机ip 网关ip

-- 欺骗网关，本机是目标主机
# arpspoof -i eth0 -t 网关ip 被劫持目标主机ip
```

- 解决方法：
  - ARP静态绑定可以防止ARP欺骗，提前绑定好目标ip和mac地址的对应关系，直接进行单播通讯，同时也不会发送询问报文进行询问，避免arp欺骗

```
arp -s 10.0.0.6 00:0c:29:32:80:38
-- arp -s ip地址 mac地址

删除arp中ip和mac的对应关系
# arp -d ip地址
```

- Gratuitous ARP
  - 定义：也称为免费ARP
  - 作用：验证IP是否冲突
  - 原理：自问自答
    - 第一步：机器启动时，向网络中发送一个谁是xxip的报文，xxip是机器自己的ip
    - 第二步：如果无冲突ip，则机器再次发送一个响应ip，回复我是xxip,从而验证网络中无ip冲突
- Reverse Address Resolution Protocol
  - RARP即反向地址解析协议，将MAC地址转换为ip地址
  - 作用：主要用于无盘环境

# Internet协议

- IP PDU报头



- 版本：占4位，指IP协议的版本，目前的IP协议版本号是4
- IPV4的源地址是32位，所以，共有 $2^{32}$ 个地址，约为43亿个
- 首部长度：占4位，可表示的最大数值是15个单位，一个单位为4个字节，因此IP的首部长度最大值是60个字节
- 分区服务：占8位
- 总长度；占16位，是报文头+数据部分的总长度，单位是字节，因此数据包的最大长度是65535字节，总长度必须不超过最大传送单元MTU
- 标识：占16位，同一个打包，分片多个小包，每个分片的标识是一样的
- 标志：占3位，标志同一个包，分片的序号，目前只有后两位有意义
- 片偏移：13位，制定了数据包的偏移量，即较长的分组在分片后，该分片在原分组中的相对位置，片偏移以8个字节偏移单位
- 生存期：TTL(timetolive),标识数据包允许经过的路由器的数量，不是时间，例如TTL=64;说明这个数据包允许经过64个路由器
  - TTL常见的3个默认值：64(linux), 128, 255
  - 修改ttl默认值

```
# echo (num) > /proc/sys/net/ipv4/ip_default_ttl
```

- 协议：占8位，指出此数据包携带的数据，使用何种协议以便目的主机的ip层将数据部分上交到上层哪个协议处理（1 表示为ICMP协议，2表示为IGMP协议，6表示为TCP协议，17表示为UDP协议）
- 首部校验和：占16位，判断数据包是否被破坏
- 源地址：占32位
- 目的地址：占32位
- ipv4和ipv6的区别

**IPv4**和**IPv6**是两种不同的互联网协议，用于数据在网络中的传输。下面是它们之间的主要区别：

地址长度：

**IPv4**: 32位地址，通常以点分十进制格式显示，如**192.168.1.1**。

**IPv6**: 128位地址，通常以冒号分隔的十六进制格式显示，如  
**2001:0db8:85a3:0000:0000:8a2e:0370:7334**。

地址空间：

**IPv4**: 大约有43亿个地址。

**IPv6**: 具有近乎无限的地址空间，是**IPv4**的数千倍。

头部复杂性：

**IPv4**: 头部通常更加复杂，有多个字段。

**IPv6**: 头部设计得更简洁，许多不必要的字段被去掉或者移到扩展头中。

自动配置：

**IPv4**: 需要额外的协议如**DHCP**来实现自动地址配置。

**IPv6**: 支持地址的自动配置。

安全性：

**IPv4**: 安全是可选的，依赖于**IPSec**等技术。

**IPv6**: 设计时考虑到了安全性，**IPSec**的支持是必选的。

广播：

**IPv4**: 使用广播来发送数据到同一网络的所有设备。

**IPv6**: 没有传统的广播模式，而是使用多播和其他新机制来实现类似的功能。

报文头的具体区别：

**IPv4**报文头字段（至少包含以下内容）：

版本  
头部长度  
服务类型  
总长度  
标识  
标志  
片偏移  
生存时间（TTL）  
协议  
头部校验和  
源IP地址  
目的IP地址  
选项（可选）

**IPv6**报文头字段（固定大小为40字节，包含以下内容）：

版本  
通信量类别  
流标签  
有效载荷长度  
下一个头部  
跳数限制（相当于**IPv4**的TTL）  
源地址  
目的地址

**IPv6**相比**IPv4**更加先进和健壮，设计时考虑了未来的扩展性、安全性和其他网络挑战。但由于**IPv4**的广泛部署和兼容性问题，两种协议仍然共存，尽管大量的努力正在推动向**IPv6**的过渡。

- ip地址的组成

给一个网卡配置多个ip（临时配置，重启后回复）

```
# ip a a 10.0.0.7/24 dev eth0 label eth0:1
```

-- 给这台机器的网卡的子接口添加一个10.0.0.7的ip

解析：

/\* **ip**: 这是Linux下的一个命令，用于显示或操作路由、网络设备、策略路由和隧道。

它是**ifconfig**命令的现代替代者。 \*/

-- 第一个**a**: 这是**ip**命令的缩写形式，实际上应该是**addr**或**address**，意为“地址”，用于管理网络地址

-- 第二个**a**: 这是**add**的缩写，表示添加

/\* 10.0.0.7/24: 这是**CIDR**格式的**IP**地址。

10.0.0.7是**IP**地址，/24表示其子网掩码的前24位是1，对应于子网掩码255.255.255.0。 \*/

-- **dev**: 这是“**device**”的缩写，意为“设备”，表示接下来要指定的是哪个网络设备。

-- **eth0**: 这是网络设备(网卡)的名称，通常是**Ethernet**适配器的默认名称

-- **label**: 标签，用于给配置的**IP**地址提供一个别名或标识。这在你给单个物理接口配置多个**IP**地址时非常有用。

/\* **eth0:1**: 这是给新添加的**IP**地址10.0.0.7的标签。

它告诉系统这个地址是**eth0**的一个别名或虚拟接口。

通常，可以为一个物理接口配置多个虚拟接口，每个虚拟接口有其自己的**IP**地址。 \*/

```
# ip a      -- 查看网卡信息
```

- 探测ip冲突的情况

```
# arping ip地址
```

## ip地址

- ip地址的意义

- ip地址可以实现逻辑上的管理功能

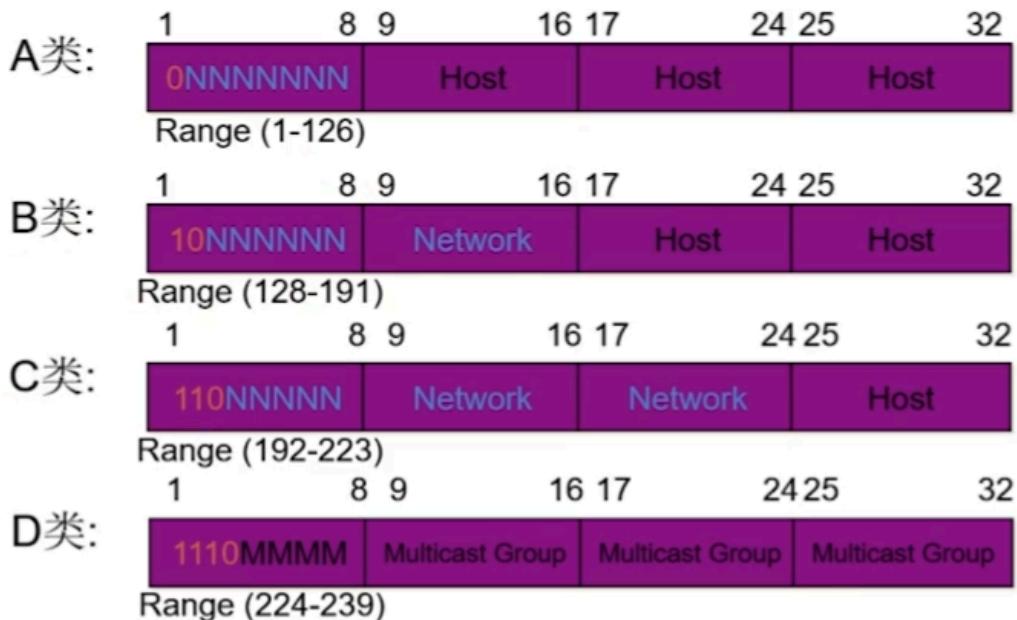
- ip地址组成

- 网络ID：表示网络，每个网段分配一个网络ID，处于高位

- 作用：体现ip所在的网段（位置）

- 主机ID：表示单个主机，由组织分配给各设备，处于低位

- IP地址的分类 (早期分类)



- A类: 1-126.X.X.X
  - 0开头, 表示未知地址, 不能用
  - 127开头, 表示回环地址, 不能用
- B类: 128-191.X.X.X
- C类: 192-223.X.X.X
- D类: 224-239.X.X.X (多播)
- E类: 略 (留给科学家使用)
- 公式
  - 网络数 =  $2^{\text{可变的网络ID数}}$
  - 主机数 =  $2^{(32-\text{可变网络ID数})}-2$ 
    - 主机数全为0, 则是该网段的编号
    - 主机数全为1, 则是该网段的广播ip
  - 网络ID = IP和子网掩码netmask做'与运算'
- 不分类的IP(现代方式)
  - 优势: 早期的分类方法过于僵化, 不利于ip地址的灵活管理
  - 网络ID的位数按需分配
  - CIDR表示法: 表示网络ID的位数
    - IP/网络ID的位数
  - 子网掩码表示: 可以用来表示网络ID的位数, 32位二进制, 对应于网络ID的位为1, 对应于主机ID的位为0
- 公共和私有IP地址

- 私有IP地址：不直接用于互联网，通常在局域网中使用

类	私有地址范围
A	10.0.0.0 到 10.255.255.255
B	172.16.0.0 到 172.31.255.255
C	192.168.0.0 到 192.168.255.255

- 公共IP地址：互联网上设备拥有的唯一地址

类	公共 IP 地址范围
A	1.0.0.0 到 9.255.255.255 11.0.0.0 到 126.255.255.255
B	128.0.0.0 到 172.15.255.255 172.32.0.0 到 191.255.255.255
C	192.0.0.0 到 192.167.255.255 192.168.0.0 到 223.255.255.255

- 公共IP地址世界唯一，可以通过互联网访问，私有地址不能通过互联网访问

- 特殊地址

- 0.0.0.0

0.0.0.0 不是一个真正意义上的IP地址。它表示所有不清楚的主机和目的网络

- 255.255.255.255

限制广播地址。对本机来说，这个地址指本网段内（同一广播域）的所有主机

- 224.0.0.0到239.255.255.255

组播地址，224.0.0.1特指所有主机，224.0.0.2特指所有路由器。224.0.0.5指OSPF路由器，地址多用于一些特殊的程序以及多媒体程序

- 169.254.x.x

如果Windows主机使用了DHCP自动分配IP地址，而又无法从DHCP服务器获取地址，系统会为主机分配这样的地址

- 判断两个IP是否在一个网段

- 示例A:10.0.1.1/16; B:10.0.2.2/24

- 分析步骤：
  - 如果A访问B，则A和自己的netmask做与运算
  - B和A的netmask做与运算
  - 两个结果进行比较，相同则在同一网段，反之不在。
  - 如果B访问A，则比较结果不相同，则不在同一网段

(如果在一个网段，则可以直接arp寻址通讯；如果不在同一网段，则需要借助路由网关进行寻址通讯)

- 划分子网
  - 定义：将一个大的网络(主机数多)划分成多个小的网络(主机数少)，主机ID位数变少，网络ID位数变多，网络ID位向主机ID位借n位，将划分 $2^n$ 个子网
  - 公式：网络ID向主机ID借位，如果借n位，则划分 $2^n$ 个子网

试题：中国移动10.0.0.0/8 给32个各省子公司划分对应子网

1. 每个省公司的子网netmask?

-- 255.248.0.0

2. 每个省公司的子网的主机数是多少？

-- 524286

3. 河南省得到第10个子网，网络ID?

-- 10.72.0.0/13

4. 河南省得到第10个子网的最小IP和最大IP?

-- 最小IP: 10.72.0.1; 最大IP: 10.79.255.254

5. 所有子网中最大，最小的子网的net id?

-- 最小ip: 10.0.0.0/13; 最大ip: 10.248.0.0/13

- 优化IP地址分配
  - 合并超网：将多个网络合并成一个大网，主机ID位向网络ID位借位，主要实现路由聚合功能
- 动态主机配置协议DHCP
  - 服务器端：67端口
  - 客户端：68端口

## 网络配置

- 基本网络配置
  - 将Linux主机接入网络，需要配置网络相关设置，一般包括
    - 主机名
    - IP/netmask

- 路由：默认网关
- DNS服务器（名称解析）
  - 主DNS服务器
  - 次DNS服务器
  - 第三个DNS服务器

```
-- 配置主机名
# hostnamectl set-hostname <新主机名> -- centos7之后使用
# cat /etc/hostname -- centos7之后，主机名配置文件

-- centos6之前
# vim /etc/sysconfig/network -- 更改主机名配置文件
# hostname centos6.magedu.org -- 启用主机名
```

- 网卡名称

#### 更改网卡名

-- centos6及以前  
 永久修改网卡名 -> 更改文件  
`# vim /etc/udev/rules.d/70-persistent-net.rules`  
 -- 删除多余网卡信息  
 -- 将现有网卡信息的名称改为eth0

使用指令‘临时’修改网卡名  
`# ip link set <网卡名> down --禁用网卡`  
`# ip link set <旧网卡名> name <新网卡名> --更改网卡名`  
`# ip link set <新网卡名> up -- 启用网卡`

-- centos7及以后  
 永久修改网卡名 -> 修改配置文件  
`# vim /etc/default/grub`  
`GRUB_CMDLINE_LINUX="... net.ifnames=0"`  
 -- 进入文件，这行最后，添加`net.ifnames=0`

`# grub2-mkconfig -o /boot/grub2/grub.cfg`  
 -- 执行之后重启即可

-- ubuntu的修改和centos7类似

`# vim /etc/default/grub`  
`GRUB_CMDLINE_LINUX="net.ifnames=0"`  
 -- 进入文件，这行最后，添加`net.ifnames=0`

`# grub-mkconfig -o /boot/grub/grub.cfg`  
 -- 执行之后重启即可

- grub2-mkconfig的用法

```
grub2-mkconfig -o <output_file>

# grub2-mkconfig: 这是命令本身，用于生成GRUB2的配置文件
# -o: 这个选项后面跟着输出文件的路径，即生成的配置文件存放的位置
# <output_file>: 这里指定了输出文件的完整路径，即新的GRUB配置文件的位置
```

- ip地址修改

Centos网卡配置文件:

```
# /etc/sysconfig/network
-- 更改主机名hostname
# /etc/sysconfig/network-scripts/
-- 进入网卡配置目录
# ifcfg-eth0
-- 找到这个文件或新建这个文件（内容自己写）-网卡接口配置文件
-- 文件名必须以ifcfg-开头，横杠后叫什么无所谓，建议和网卡名一致
```

文件配置内容:

```
DEVICE=eth0 | HWADDR=00:0c:29:88:37:b8 (MAC地址) 二选一
-- 两种方式都可以用来表示选择配置哪块网卡
MACADDR=<mac地址> --(非必选)，可以用来更改mac地址
NAME=eth0
BOOTPROTO=dhcp --动态配置

BOOTPROTO=static 或者none --静态配置
IPADDR=10.0.0.88
NETMASK=255.255.255.0 或者 PREFIX=24
GATEWAY=10.0.0.2 --配置网关，根据网络规划写
-- 查看默认网关，执行ip route 或者 route -n
DNS1=10.0.0.2
DNS2=100.76.76.76
-- 填写DNS服务器地址，网上可以查到一些著名的DNS服务器地址，都可以用
-- 比如阿里的: 223.6.6.6
-- 百度的: 180.76.76.76
-- 移动的: 114.114.114.114
-- 谷歌的: 8.8.8.8
-- 验证DNS，查看文件/etc/resolv.conf
```

DOMAIN=<要修改的后缀名>

-- 默认ping www 后面自动补充一个域名，该域名是hostname的后缀，如果不修改hostname后缀，可以在网卡配置文件中加入:

DOMAIN=<要修改的后缀名>

-- 默认ping www 后面自动补充一个域名，该域名是hostname的后缀，如果不修改hostname后缀，可以在网卡配置文件中加入:

ONBOOT=yes --是否启动这个网卡，默认yes

-- 保存配置文件

--centos8及之上执行这两条命令

```
# nmcli connection reload
```

```
# nmcli connection up eth0 -- 启用设置好的网卡  
-- 如果是多块网卡同时设置后启用，可以用空格隔开  
-- nmcli connection up eth0 eth1  
  
--centos7执行  
# systemctl restart network
```

- ifconfig命令

查看启用网卡

```
# ifconfig
```

查看所有网卡

```
# ifconfig -a
```

启用网卡

```
# ifconfig eth0 up
```

禁用网卡

```
# ifconfig eth0 down
```

更改网卡ip

```
# ifconfig eth0 <新ip地址>
```

临时清空ip地址

```
# ifconfig eth0 0.0.0.0/0
```

保留一个网卡的情况下，增加新地址，实现一个网卡多个ip

```
# ifconfig eth0:1 <新ip地址> //网卡别名
```

查看网卡吞吐量和状态

```
# ifconfig -s
```

```
# watch -n1 ifconfig -s
```

删除网卡别名

```
# ifconfig eth0:1 down
```

清除网卡ip

```
# ifconfig eth0 0
```

- 路由route

- route: 路由表管理命令

- 路由表:作用是导航，地图，不仅仅在路由器有，在任何通信的主机都有

查看路由表

```
# route -n
```

```
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref    Use Iface
0.0.0.0         192.168.56.2   0.0.0.0        UG    100    0      0 eth0
# 默认路由, 在通向未知网段, 目标路由设为0.0.0.0
192.168.56.0   0.0.0.0       255.255.255.0  U     100    0      0 eth0
# 直连网段的路由会自动生成
192.168.122.0  0.0.0.0       255.255.255.0  U     0      0      0 virbr0
```

每一行描述的是一个网络的路径

**Destination:** 表示需要达到的目标网络, **Destination**配合**Genmask** (子网掩码) 表示具体网络地址  
0.0.0.0 表示未知网络

**iface**接口: 如果要达到目标网络**ID**, 需要从本机的那个(网卡)接口将数据包发出

**Gateway**网关: 如果目标网络不直接相连, 需要将数据包发送到下一个路由器邻近的接口的**IP**, 即网关; 如果网络和本主机直连, 则无需网关

网关的功能: 让主机通过网关访问别的网段的机器, 所以网关的接口地址, 一定和主机网卡在同一网段

**Metric**(费用); 数值越低, 优先级越高, 路径越优; 同一目标地址可能有多条路径, 根据**Metric**的数值, 可以看出最优路径

---

给路由表添加信息 (静态路由)

```
# route add [-net|-host|default] target[netmask Nm] [gw Gw] [[dev] If]
```

实例:

仅主机路由

```
# route add -host 192.168.1.3 gw 172.16.0.1 dev eth0
```

默认路由

```
#route add -net 0.0.0.0 netmask 0.0.0.0 gw 172.16.0.1
#route add default gw 172.16.0.1
```

```
# route add [-net|-host|default] target[netmask Nm] [gw Gw] [[dev] If]
```

实例:

仅主机路由

```
# route add -host 192.168.1.3 gw 172.16.0.1 dev eth0
```

默认路由

```
#route add -net 0.0.0.0 netmask 0.0.0.0 gw 172.16.0.1
#route add default gw 172.16.0.1
```

网络路由

```
# route add -net 179.20.0.0/16 gw 172.18.0.201 dev eth1
```

查看虚拟机转发功能状态, 默认不开启, 如果接收到的数据包, 目标地址不是该主机, 则直接抛弃, 不会转发

```
# cat /proc/sys/net/ipv4/ip_forward  
0
```

0表示不开启数据包转发功能

修改这个**ip\_forward**文件可以临时开启数据转发

永久开启

修改文件**sysctl.conf**

```
# vim /etc/sysctl.conf
```

在最后一行，添加: **net.ipv4.ip\_forward=1**

保存退出

```
# sysctl -p
```

使文件**sysctl.conf** 生效

使用**mtr**工具，可以查看主机到目标机器之间经过的路由

```
# mtr <目标ip>
```

使用**tracepath** 和**traceroute**，都能查看主机到目标经过的路由

```
# tracepath <目标ip>
```

```
# traceroute <目标ip>
```

配置动态路由（了解即可，后续有时间再深入了解）

通过守护进程获取动态路由

安装**quagga**包，通过命令**vtysh**配置

支持多种路由协议：**RIP**、**OSPF**和**BGP**

**RIP**协议算法：经过的路由器越少（步跳），路线越优

**OSPF**协议算法：除了路由数量，还会考虑带宽

---

**netstat**工具(和**ss**选项基本一致)

**netstat**和**ifconfig**都来自于**net-tools**

其中 **ifconfig**建议使用**ip**代替；**netstat**建议使用**ss**代替

**ip**和**ss**都来自于**iproute**

常用选项

**-t**: **tcp**协议相关

**-u**: **udp**协议相关

**-w**: **raw socket**相关

**-l**: 处于监听状态

**-a**: 所有状态

**-n**: 以数字显示**IP**和端口

**-e**: 扩展格式

**-p**: 显示相关进程及**PID**

常用组合

**-tan**

**-uan**

**-tnl**

**-unl**

```
显示路由表，类似于route -n
```

```
# netstat -rn
```

```
# netstat -I eth0
```

```
-I和网卡名之间没有空格，作用和ifconfig -s 相同，用于查看网卡吞吐量
```

```
-----  
ip 工具
```

```
链路层指令
```

```
# ip link      查看链路层，网卡的状态
```

```
# ip link set <网卡名> up/down    禁用/启动网卡
```

```
使用指令‘临时’修改网卡名
```

```
# ip link set <网卡名> down    --禁用网卡
```

```
# ip link set <旧网卡名> name <新网卡名>    --更改网卡名
```

```
# ip link set <新网卡名> up    -- 启用网卡
```

```
网络层指令
```

```
# ip address      显示网络层相关地址
```

```
# ip address add 10.0.0.100/24 dev eth0 label eth0:1
```

```
建议加label标签，这样可以和ifconfig兼容
```

```
新增网卡ip地址
```

```
# ip addr del 1.1.1.1/24 dev eth0    删除ip地址
```

```
# ip addr flush dev eth0    清空网卡下所有ip地址
```

```
# nmcli connection up eth0 重新加载eth0的原始ip地址
```

```
# ip route    查看路由表，相当于route -n
```

```
# ip route add 路由表信息    新增路由网段
```

```
# ip route del 路由表信息    删除路由网段信息
```

- 回环网卡：lo

  - 回环网卡的地址不会出现在路由表中

  - 所有和回环网卡同一网段的地址，都看作trewgkllfk环地址默认不参与网络通信

  - 但是如果给回环网卡加上路由信息，也能实现网络通讯

- ip地址的三种工作范围

  - global：全局有效

  - link：仅该网卡链路有效

  - host：仅主机有效

- 虚拟机的三种网络连接类型

  - 桥接(Bridge) - vmnet0:

- 桥接模式下的虚拟机将被视为与主机处于同一物理网络中的另一台独立的计算机。它会直接连接到主机的物理网络。
- 虚拟机会从网络的 DHCP 服务器（可能是家庭路由器或公司网络的 DHCP 服务器）获取其 IP 地址。
- 这种模式使得虚拟机可以像其他物理计算机一样与网络中的其他设备通信。
- NAT (Network Address Translation) - vmnet8:
  - NAT 模式下的虚拟机将与外部网络通信，但它不是直接连接的。而是通过 VMware 的 NAT 服务，该服务在主机上运行。
  - 虚拟机将从 VMware NAT 服务的 DHCP 获取 IP 地址，而不是从外部网络。
  - 这种设置提供了一种隔离，因为虚拟机不会在主网络上直接可见。但是，它仍然可以访问外部网络，例如进行互联网浏览或下载文件。
- 仅主机 (Host-only) - vmnet1:
  - 仅主机模式下的虚拟机只能与主机通信。它不能访问外部网络
  - 这提供了一种高度的隔离，适用于那些您不希望暴露给网络但仍想在主机上进行交互的虚拟机。
  - 虚拟机会从 VMware 仅主机网络的 DHCP 服务获取 IP 地址。

## 路由相关配置文件

```
/etc/sysconfig/network-scripts/route-IFACE

# route-IFACE 默认不存在，需要手工创建，IFACE是网卡名

两种风格：
(1) TARGET via GW
如：10.0.0.0/8 via 172.16.0.1

(2) 每三行定义一条路由 (#表示数字，不推荐)
ADDRESS#=TARGET
NETMASK#=mask
GATEWAT#=GW
```

- 重启网络服务
  - CentOS 6 : service network restart
    - CentOS 6 记得关闭NetworkManager
  - CentOS 7及以后: systemctl restart netwrok
- 注意:
  - 网卡别名必须配置静态地址

## 多网卡bonding

- 将多块网卡绑定同一IP地址对外提供服务，可以实现高可用或者负载均衡。直接给两块网卡设置同一IP是不可以的。通过bonding，虚拟一块网卡对外提供连接，物理网卡的被修改为相同的MAC地址
- 共7种模式

- Mode0(balance-rr): 轮询(Round-Robin)策略，从头到尾顺序的在每个slave接口上面发送数据包。本模式提供负载均衡和容错能力。
- Mode1(active-backup): 活动备份(主备)策略，只有一个slave被激活，当且仅当活动的slave接口失败时才会激活其他slave，为了避免交换机发生混乱，此时绑定的MAC地址只有一个外部端口上可见
- Mode3(broadcast): 广播策略，在所有的slave接口上传送所有的报文，提供容错能力
- 说明：
  - active-backup、balance-tlb和balance-alb模式不需要交换机的任何特殊配置。其他绑定模式需要配置交换机以便整合链接。如：Cisco交换机需要在模式0、2和3中使用Ether Channel，但在模式4中需要LACP和Ether Channel
- Bonding配置

创建bonding设备的配置文件

```
/etc/sysconfig/network-scripts/ifcfg-bond0
# 其中ifcfg-bond0文件需手工配置，文件名后面必须是bond加数字
```

```
TYPE=bond
DEVICE=bond0
BOOTPROTO=none
IPADDR=10.0.0.100
PREFIX=8
# miimon指定链路监测时间间隔。如果miimon=100，那么系统每100ms监测一次链路连接状态，如果有一条
线路不通就转入另一条线路
BONDING_OPTS="mode=1 miimon=100"
# BONDING_OPTS="mode=3 miimon=100"
# 这里相当于使用Mode3，使用广播模式
```

```
/etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
BOOTPROTO=none
MASTER=bond0
SLAVE=yes
ONBOOT=yes
```

```
/etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
BOOTPROTO=none
MASTER=bond0
SLAVE=yes
ONBOOT=yes
```

- 查看bond0状态：

- `/proc/net/bonding/bond0`

- 删除bond0

- `ifconfig bond0 down`
  - `rmmmod bonding`

# CentOS7以上版网络配置

- 概述：CentOS6之前，网络接口使用连续号码命名：eth0、eth1等，当增加或删除网卡时，名称可能会发生变化。CentOS7以上版使用基于硬件，设备拓扑和设备类型命名。
- 网卡命名机制
  - 如果Firmware（固件）或BIOS为主板上集成的设备提供的索引信息可用，且可预测，则根据此索引进行命名，如：eno1
    - 翻译：如果你的计算机主板上有一个内置的网络接口（网卡），系统可以从固件或BIOS中获取一些信息来给它命名。这种命名是基于该网络接口在主板上的位置或其他硬件信息。例如，它可以被命名为“eno1”，其中“eno”表示它是一个嵌入式网络输出，并且“1”是它的编号或索引。
  - 如果Firmware或BIOS为PCI-E扩展槽所提供的索引信息可用，且可预测，则根据此索引进行命名，如：ens1
    - 翻译：如果你在计算机上安装了一个额外的网络接口卡（例如，通过一个PCI-E扩展槽），系统也可以从固件或BIOS中获取一些信息来给它命名。这种命名是基于该网络接口卡在扩展槽中的位置或其他硬件信息。例如，它可以被命名为“ens1”，其中“ens”表示它是一个PCI-E网络输出，而“1”是它的编号或索引。
  - 如果硬件接口的物理位置信息可用，则根据此信息命名，如：enp2s0
    - 翻译：如果系统能够获取到网络接口在计算机内部的物理位置信息（例如，它连接到第2个PCI总线上的第0个插槽），它就会使用这些信息来给网络接口命名。在这个例子中，“enp2s0”是基于网络接口的物理位置来命名的，其中“p2”表示PCI总线2，而“s0”表示插槽0。
  - 如果用户显示启动，也可根据MAC地址进行命名，如：enx2387a1dc56
    - 翻译：如果用户明确要求（通常是通过配置文件设定），系统可以使用网络接口的MAC地址来命名它。MAC地址是一个网络接口的唯一标识符。在这个例子中，“enx2387a1dc56”是基于网络接口的MAC地址来命名的，其中“2387a1dc56”是该接口的MAC地址。
  - 上述均不可用时，使用传统命名机制
- 网卡组成格式
  - en：Ethernet 有线局域网
  - wl：wlan 无线局域网
  - ww：wwan 无线广域网
  - o<index>：集成设备的设备索引号
  - s<slot>：扩展槽的索引号
  - x<MAC>：基于MAC地址的命名
  - p<bus>s<slot>：enp2s1
- 更改网卡命名，使用传统命名方式
  - [更改方法](#)
- 主机名hostname（CentOS7及以上）
  - 配置文件：`/etc/hostname`
  - 默认没有此文件，通过DNS反向解析获取主机名，主机名默认为：localhost.localdomain
  - 设置主机名：`hostnamectl set-hostname centos7.magedu.com`

- 删除文件/etc/hostname，恢复主机名localhost.localdomain
- 显示主机名
  - `hostname`
  - `hostnamectl status`
- 网络配置工具nmcli
  - 依赖NetworkManager服务，此服务是管理和监控网络设置的守护进程
  - nmcli命令

```
nmcli connection # 查看网卡连接

# 更改网卡名称(name)
nmcli connection modify <旧网卡名> con-name <新网卡名> # 更改后自动生成配置文件

# 更改网卡配置
nmcli connection modify <网卡名> ipv4.addresses <ip地址> ipv4.gateway <网关地址>
ipv4.dns <dns地址> ipv4.method manual
# 这里ipv4.method manual表示采用静态地址，默认不写是auto，表示动态地址
# 示例：
nmcli connection modify eth1-home ipv4.addresses 192.168.0.100/24
ipv4.gateway 192.168.0.1 ipv4.dns 233.6.6.6 ipv4.method manual
```

配置改好后，重新加载网卡，然后重新启用配置好的网卡

```
nmcli connection reload
nmcli connection up eth1-home
```

```
# 删除配置的网卡
nmcli connection delete <网卡名>

# 使用nmcli看详细配置内容
nmcli connection show eth0
```

---

使用nmcli实现bonding模式

```
nmcli connection add con-name mybond0 ifname bond0 type bond mode active-
backup
# 添加一个bond类型的虚拟网卡，起名为mybond0，mode定义bond模式

nmcli connection add con-name mybond0 ifname bond0 type bond mode active-
backup ipv4.addresses 10.0.0.100/24 ...
# 配置的时候添加ip地址，方式，网关等

nmcli connection add con-name mybond0-eth1 ifname eth1 type bond-slave master
bond0
# 将eth1添加到bonding中

nmcli connection delete <eth1之前的配置文件>
# 删掉之前的eth1的配置文件，自动启用新配置的绑定bonding的eth1

cat /proc/net/bonding/bond0
# 查看bond0的绑定情况
```

- 网络组 NetWork Teaming

- 网络组：是将多个网卡聚合在一起方法，从而实现容错和提高吞吐量，网络组不同于旧版的 bonding技术，提供更好的性能和扩展性网络组由内核驱动和teamd守护进程实现
- 多种方式 runner
  - broadcast
  - roundrobin
  - activebackup
  - loadbalance
  - lacp(implements the 802.3ad Link Aggregation Control Protocol)
- 创建网络组示例：

```
# 创建网络组接口
nmcli con add type team con-name CNAME ifname INAME [config JSON]

CNAME 连接名
INAME 接口名
JSON 指定runner方式，格式: '{"runner":{"name":"METHOD"}}'
METHOD 可以是broadcast, roundrobin, activebackup, loadbalance, lacp

# 创建port接口
nmcli con add type team-slave con-name CNAME ifname INAME master TEAM
CNAME 连接名，连接名若不指定，默认为team-slave-IFACE
INAME 网络接口名
TEAM 网络组接口名

# 断开和启动
nmcli dev dis INAME
nmcli con up CNAME

-----  
网络组示例：
nmcli con add type team con-name myteam0 ifname team0 config '{"runner": {"name": "loadbalance"}}' ipv4.addresses 192.168.1.100/24 ipv4.method manual

nmcli con add con-name team0-eth1 type team-slave ifname eth1 master team0

nmcli con add con-name team0-eth2 type team-slave ifname eth2 master team0

nmcli con up myteam0
nmcli con up team0-eth1
nmcli con up team0-eth2

查看team状态
teamdctl team0 state
` `
```

- 网桥

- 作用：网桥的作用是将多个设备串联在一起，实现通讯，隔离冲突域，相当于交换机
- 配置实现网桥

```

# 下载bridge-utils Centos 8 无此包
yum -y install bridge-utils

# 查看网桥
brctl show

# 查看CAM(content addressable memory 内容可寻址存储器)表
brctl showmacs br0

# 添加和删除网桥
brctl addbr | delbr br0

# 添加和删除网桥中的网卡
brctl addif | delif br0 eth0

# 默认br0 是down, 必须启用
ifconfig br0 up

# 启用STP协议
brctl stp br0 on

```

- STP (生成树协议)
  - 交换机上的协议, 用来防止广播风暴(回环)

## TCPdump的使用

- 概述:
  - TCPdump,全称dump the traffic on a network, 是一个运行在linux平台可以根据使用需求对网络上传输的数据包进行捕获的抓包工具
- 功能:
  - 在Linux平台将网络中传输的数据包全部捕获过来进行分析
  - 支持网络层、传输层协议等协议捕获过滤
  - 数据发送和接收的主机、网卡和端口等各种过滤捕获数据规则
  - 提供and、or、not等语句进行逻辑组合捕获数据包或去掉不同的信息
  - 结合wireshark工具分析捕获的报文

```

# 不指定任何参数, 默认监听第一块网卡经过的数据包。
tcpdump

# 监听特定网卡
tcpdump -i eth0

# 监听特定主机, 监听主机10.0.0.100的通信包, 注意: 出入包都会被监听
tcpdump host 10.0.0.100

# 特定来源、目标地址的通信
# 特定来源
tcpdump src host hostname
# 特定目标
tcpdump dst host hostname
# 如果不指定src跟dst, 那么来源或者目标是hostname的通信都会被监听

```

```
tcpdump host hostname

# 特定端口
tcpdump port 3000

# 监听tcp/udp，服务器上不同服务分别用了tcp, udp作为传输层
tcpdump tpc # 监听tcp协议报文

# 来源主机+端口+tcp，监听来自主机10.0.0.100在端口22上的tcp数据包
tcpdump tcp port 22 and src host 10.0.0.100

# 监听特定主机之间的通信
tcpdump ip host 10.0.0.101 and 10.0.0.102

# 10.0.0.101和除了10.0.0.1之外的主机之间的通信
tcpdump ip host 10.0.0.101 and ! 10.0.0.1

# 将tcpdump的数据包，重定向到一个文件中
tcpdump -i eth0 -nn port ! 22 -w test.cap
sz test.cap # 将虚拟机的文件传到本地
```

### ### curl工具

- 介绍: curl命令来自于英文词组"Commandline URL"的缩写, 其功能是用于在shell终端界面中基于URL规则进行的文件传输工作, curl是一款综合的传输工具, 可以上传, 也可以下载, 支持HTTP, HTTPS, FTP等三十多种常见协议

```
- 语法: `curl [options] <url>`  
```shell  
curl <url>  
# 获取指定网站的网页源码, 默认GET请求  
  
curl -v <url>  
# 表示输出连接详情  
  
curl -o <url>  
# 下载指定网站中的文件  
  
curl -o filename <url>  
# 将网页源码保存到指定文件中  
  
curl -I https://www.baidu.com  
# 打印指定网站的HTTP响应头信息  
  
curl -Iv <url>  
# 同时打印HTTP请求和响应头信息  
  
curl -i -X <Method> <url>  
# 指定请求HTTP方法, 并返回响应信息
```

## wget工具

- 介绍: Linux下的下载命令
- 语法: `wget [options] <url>`

```
wget <url>
# 将链接文件直接下载到当前路径

wget -O filename <url>
# 表示output输出为文件, 后接文件名参数
```

## ubuntu网络配置

- 主机名

```
# 和Centos7之后版本的指令, 配置文件都一样
hostnamectl set-hostname ubuntu1804.magedu.org
# 用这个指令更改主机名后, 配置文件hostname里的主机名会自动被更改

cat /etc/hostname
ubuntu1804.magedu.org
```

- 网卡名称
  - 默认ubuntu的网卡名称和CentOS 7类似
  - 修改网卡名称为传统命名方式:

```
# 修改配置文件为下面形式
vim /etc/default/grub
GRUB_CMDLINE_LINUX="net.ifnames=0"
# 或者sed修改
sed -i.bak '/^GRUB_CMDLINE_LINUX=/s#"net.ifnames=0"' /etc/default/grub

# 生效新的grub.cfg文件
grub-mkconfig -o /boot/grub/grub.cfg # 本质上是修改grub.cfg文件, 在上面添加
net.ifnames=0的信息

# 本质: grub文件相当与一个修改模板, 通过grub-mkconfig指令使其调用grub模板去更改
grub.cfg文件
# 所以, 实际上可以只修改grub.cfg文件, 讲所有linux开头的行后添加net.ifnames=0, 是一样的

# 或者使用
update-grub # 作用等同于grub-mkconfig -o /boot/grub/grub.cfg

reboot # 最后重启生效
```

- 网卡配置

```
# ubuntu20.04的网卡配置文件采用yaml格式, 要求各级缩进必须严格统一

# 官方文档参考: https://ubuntu.com/server/docs/network-configuration

# ubuntu20.04网卡配置文件路径
```

```
/etc/netplan/01-netcfg.yaml # 配置文件命名格式: 数字-netcfg.yaml

# 配置文件内容 (严格控制缩进)
network:
  ethernets:
    ens33:      # 更改网卡名
      dhcp4: true # 这个就是自动分配, 动态网卡地址
    version: 2
    renderer: networkd # 选填

# 配置之后, 需要执行命令生效:
netplan apply # NAT模式下生效, 桥接模式, 需要配置动态地址

# yaml格斯的常见数据结构: 列表和字段
# ubuntu20.04静态地址配置
network:
  version: 2 #冒号后面要加空格
  renderer: networkd
  ethernets:
    eth0:
      addresses: [192.168.8.10/24, 10.0.0.10/8] # 或者用下面两行, 两种格式不能混用
      - 192.168.8.10/24 # 横线加空格, 也是yaml中列表中元素的一种表现形式
      - 10.0.0.10/8
      gateway4: 192.168.8.1
      nameservers:
        search: [magedu.com, magedu.org]
        addresses: [180.76.76.76, 8.8.8.8, 1.1.1.1]
```

- 查看ip和gateway

```
ip addr
route -n
```

- 查看DNS

```
# 使用指令来看
systemd-resolve --status
```

- 配置多网卡静态ip并添加静态路由

```
# 编辑配置文件
vim /etc/netplan/01-eth1cfg.yaml

# 内容
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
      addresses: [10.0.0.18/16]
      gateways: 10.0.0.2
      nameservers:
        addresses: [223.6.6.6]
```

```

eth1:
  dhcp4: no
  dhcp6: no
  addresses: [10.20.0.18/16]
  routes:
    - to: 10.30.0.0/16
      via: 10.20.0.1/16
    - to: 10.40.0.0/16
      via: 10.20.0.1/16
    - to: 10.50.0.0/16
      via: 10.20.0.1/16
    - to: 10.60.0.0/16
      via: 10.20.0.1/16

```

- 双网卡绑定

支持多网卡绑定其中模式

第一种模式: mod=0, 即: (**balance-rr**) **Round-robin policy** (平衡论循环策略)

特点: 传输数据包顺序是依次传输 (即: 第1个包走**eth0**, 下一个包就走**eth1**...一直循环下去, 知道最后一个传输完毕), 此模式提供负载平衡和容错能力

第二种模式: mod=1, 即: (**active-backup**) **Active-backup policy** (主-备份策略)

特点: 只有一个设备处于活动状态, 当一个宕机另一个马上由备份转换为主设备。mac地址是外部可见的, 从外面看来, **bond**的mac地址是唯一的, 以避免**switch**发生混乱。此模式只提供了容错能力; 由此可见此算法的优点是提供网络连接的可用性, 但是它的资源利用率较低, 只有一个接口处于工作状态, 在有N个网络接口的情况下, 资源利用率为1/N。

第三种模式: mod=2, 即: (**balance-xor**) **XOR policy** (平衡策略)

特点: 基于指定的传输HASH策略传输数据包。缺省的策略是: (源MAC地址 XOR 目标MAC地址) % slave数量。其他的传输策略可以通过**xmit\_hash\_policy**选项指定, 此模式提供负载平衡和容错能力

第四种模式: mod=3, 即: **broadcast** (广播策略)

特点: 在每个**slave**接口上传输每个数据包, 此模式提供了容错能力

第五种模式: mod=4, 即: (**802.3ad**) **IEEE 802.3ad Dynamic link aggregation** (**IEEE 802.3ad** 动态链接聚合)

特点: 创建一个聚合组, 它们共享同样的速率和双工设定。根据**802.3ad**规范将多个**slave**工作在同一个激活的聚合体下。

该模式必要条件:

条件1: **ethtool**持获取每个**slave**的速率和双工设定

条件2: **switch**支持**IEEE 802.3ad Dynamic link aggregation**

条件3: 多数**switch**需要经过特定配置才能支持**802.3ad**模式

第六种模式: mod=5, 即: (**balance-tlb**) **Adaptive transmit load balancing** (适配器传输负载均衡)

特点: 不需要任何特别的**switch**支持的通道**bonding**。在每个**slave**上根据当前的负载 (根据速度计算) 分配外出流量。如果正在接收数据的**slave**出故障了, 另一个**slave**接管失败的**slave**的MAC地址。

该模式的必要条件:

**ethtool**支持获取每个**slave**的速率

第七种模式: mod=6, 即: (**balance-alb**) **Adaption load balancing** (适配器适应性负载均衡)

特点: 该模式包含了**balance-tlb**模式, 同时加上了针对**IPV4**流量的接收负载均衡 (**receive load balance, rlb**), 而不需要任何**switch**的支持

```

# 在Ubuntu中启用多网卡绑定，需要直接写文件
vim /etc/netplan/01-netcfg.yaml

# 配置文件内容
# This file describes the network interfaces available on your system
# for more information, see netplan(5)
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      dhcp6: no
    eth1:
      dhcp4: no
      dhcp6: no

  bonds:
    bond0:
      interfaces:
        - eth0
        - eth1
      addresses: [10.0.0.18/16]
      gateways: 10.0.0.1
      nameservers:
        addresses: [223.6.6.6, 223.5.5.5]
      parameters:
        mode: active-backup
        mii-monitor-interval: 100 # 监控间隔

# 使其配置生效
netplan apply

```

```

# 查看生成的bond信息文件
cat /proc/net/bonding/bond0

```

- 网络测试诊断工具
  - 测试网络连通性
    - ping
  - 显示正确的路由表
    - ip route
  - 跟踪路由
    - traceroute
    - tracepath
    - mtr
  - 确定名称服务器使用
    - nslookup
    - host

- dig
- 抓包工具
  - tcpdump
  - wireshark
- 安全扫描工具
  - nmap
  - netcat: 网络界的瑞士军刀

## NAT (Network Address Translation) 技术。

- 前置知识科普
  - 私网ip:
    - 私有IP地址, 如192.168.x.x、10.x.x.x、172.16.x.x至172.31.x.x
    - 特点:
      - 只能在内网/局域网通信, 无法和互联网通信
  - 公网ip
    - 查询方式
      - Windows: 打开命令提示符 (cmd), 然后输入 `nslookup myip.opendns.com @resolver1.opendns.com`。这将使用OpenDNS的解析器来返回你的公网IP。
      - Linux/Unix/Mac: 打开终端, 然后输入 `dig +short myip.opendns.com @resolver1.opendns.com`。
- 概述:
  - NAT是一种在IP数据包通过路由器或防火墙时重新映射IP地址的技术。它允许一个内网 (通常使用私有IP地址, 如192.168.x.x、10.x.x.x、172.16.x.x至172.31.x.x) 的多台设备共享一个公网IP地址进行外部通信。这样可以实现资源共享, 也为IPv4地址的不足提供了临时解决方案。
- NAT(网络地址转换)的主要类型
  - 静态NAT
    - 为内部IP地址提供一个一对一的公网IP地址映射。通常用于需要对外提供服务的设备, 如web服务器。
  - 动态NAT
    - 从一个可用的公网IP地址池中为内部设备动态分配公网IP地址。
  - PAT (Port Address Translation) / NAPT (Network Address Port Translation)
    - 这是最常见的NAT类型, 也经常被称为"端口转发"。在此类型的NAT中, 多台内部设备共享单个公网IP地址。区分不同设备的方式是通过不同的源端口号。因此, 这种方法不仅会转换IP地址, 还会转换端口号。
    - 端口转发的工作流程
      - 当内部设备想要与外部世界通信时, 它会发送一个数据包到NAT设备 (通常是家用路由器)。
      - NAT设备将内部私有IP地址和端口号更改为公网IP地址和一个新的端口号, 然后将数据包发送到外部网络。
      - 当外部服务器响应时, 它会发送数据包回到路由器的公网IP地址和之前分配的端口号。

- 路由器使用NAT表查找原始的内部IP地址和端口，然后将响应转发给正确的内部设备。
- 端口映射的应用场景
  - 内网服务器对外发布
  - 内网设备，从外部进行远程管理
  - 内网监控视频，从外部取流

## 软件管理

- 内容概述
  - 软件运行环境
  - 软件包基础
  - rpm包管理
  - yum和dnf管理
  - 定制yum仓库
  - 编译安装
  - Ubuntu软件管理

## 软件运行和编译

- 软件相关概念
  - ABI: Application Binary Interface (应用程序的二进制接口)
  - Windows与Linux不兼容
    - ELF (Executable and Linkable Format) Linux
    - PE (Portable Executable) Windows
    - **解释：在不同的系统中，二进制的格式是不同的，比如同一段二进制数据，可能含义不同，有的代指指令，有的指代数值。**

查看软件的文件信息（包含二进制格式）

```
# file /bin/hostname
```

- API: Application Programming Interface (应用程序的开发接口)
  - API可以在各种不同的操作系统上实现给应用程序提供完全相同的接口，由于在Linux和Windows上的API是不同的，导致软件在两个操作系统上无法兼容，为了实现兼容，出现了POSIX
- POSIX: Portable Operating System Interface
  - 可移植操作系统接口，定义了操作系统应该为应用程序提供的接口标准，是IEEE为要在各种UNIX操作系统上运行的软件而定义的一系列API标准的总称。Linux和Windows都要实现基本的POSIX标准，程序就在源代码级别可移植了。
- 开发语言
  - 系统级开发语言：汇编语言；C；C++
  - 应用级开发语言：java；Python；go；PHP；Perl；delphi；basic；ruby；bash
- C语言程序的实现过程

- 整体过程：
  - C程序源代码 -> 预处理 -> 编译 -> 汇编 -> 链接
- 预处理(pre-processing)
  - 将所有的#define删除，并且展开所有的宏定义
  - 处理所有的条件预编译指令，比如#define #elif #else #endif等
  - 处理#include预编译指令，将被包含的文件插入到预编译指令的位置。
  - 删除所有的注释// /\* \*/
  - 添加行号和文件标识，以便编译时产生调试用的行号及编译错误警告行号
  - 保留所有的#pragma编译指令，因为编译器需要使用它们
  - 预处理之后，形成如：hello.i的（文本）文件
- 编译(Compiling)
  - 编译过程就是把预处理完的文件进行一系列的词法分析，语法分析，语义分析及优化后，最后生成相应的汇编代码
  - 编译后，形成如：hello.s的汇编程序的文本文件
- 汇编(Assembling)
  - 汇编器是将汇编代码转变成机器可以执行的命令，每一个汇编语句几乎都对应一条机器指令。汇编相对于编译过程比较简单，根据汇编指令和机器指令的对照表——翻译即可。
  - 汇编之后，形成如：hello.o的二进制文件
- 链接(Linking)
  - 通过调用链接器id来链接程序运行需要的一大堆目标文件，以及所依赖的其它库文件，最后生成可执行文件

范例：gcc编译过程

```
# 分步骤编译运行
gcc -E hello.c -o hello.i    # 对hello.c文件进行预处理，生成hello.i文件
gcc -S hello.i -o hello.s    # 对预处理文件进行编译，生成了汇编文件
gcc -C hello.s -o hello.o    # 对汇编文件进行编译，生成了目标文件
gcc hello.o -o hello        # 对目标文件进行链接，生成可执行文件

# 一步实现编译过程
gcc hello.c -o hello    # 直接编译链接成可执行目标文件
```

- 软件模块的静态和动态链接
  - 定义：链接主要作用是把各个模块之间相互引用的部分处理好，使得各个模块之间能够正确的衔接，分为静态和动态链接
  - 静态链接
    - 将程序和依赖文件打包成一个文件
  - 动态链接
    - 每个依赖文件是一个个独立文件，程序按需调用这些依赖文件

查询文件所依赖的所有依赖库

```
# ldd <要查询的文件>
```

- 操作系统执行命令加载动态库的过程

- 加载共享库：当你运行一个动态链接的程序（如你的 hello）时，操作系统（如 Linux）负责加载程序所需的共享库。这是通过动态链接器（如 Linux 上的 ld-linux.so）来完成的，它是操作系统的一部分，负责在程序启动时解析出程序需要的共享库，并将它们加载到内存中。
- 查找共享库：操作系统按照特定的规则查找这些共享库。这通常包括一些预设的目录（如 /lib, /usr/lib）和环境变量（如 LD\_LIBRARY\_PATH）指定的目录。
- 符号解析：在库被加载之后，动态链接器还需要解决符号引用的问题，即确定程序中的外部函数或变量调用对应到库中具体哪些函数或变量的地址。这个过程称为符号解析（Symbol Resolution）。

- 动态链接器的运行时机

- 在程序运行时，通过操作系统的动态链接器（dynamic linker）或动态加载器（dynamic loader）来加载和链接共享库（动态库）的。
- 大致过程如下
  - 加载共享库到内存：当程序启动时，动态链接器根据程序的需求，将所需的共享库加载到内存中。如果库已经由其他程序加载到内存，操作系统会允许这些程序共享同一内存中的库副本，以节省资源。
  - 符号解析和重定位：接着，动态链接器对程序进行“符号解析”（symbol resolution）和“重定位”（relocation）。符号解析是指将程序中的外部引用（如函数调用）与其在共享库中定义的实际地址匹配起来的过程。重定位是调整这些引用，确保它们指向正确的内存地址。
  - 执行：一旦所有必需的库都被加载并且所有的符号都被正确解析和链接，程序就可以执行了。这意味着，程序中对共享库函数的调用会跳转到这些函数在内存中的实际位置执行

- 静态链接和动态链接的区别

- 静态链接

- 在这种链接方式中，编译器会把所有需要的库函数的代码复制到最终的可执行文件中。这意味着这些代码被物理地嵌入到了可执行文件里。静态链接的优点是可执行文件包含了所有它需要的代码，因此它不依赖于系统上的外部库文件；缺点是这可能会导致最终的可执行文件非常大。

- 动态链接

- 与静态链接不同，动态链接不会把库代码嵌入到可执行文件中。相反，它在可执行文件中保留对库文件的引用，这些库文件在程序运行时被加载（通常是操作系统完成的）。这样，多个程序可以共享同一个库文件的单个副本，节省空间。

- 动态链接器与动态库详解

- 动态链接器：动态链接器（dynamic linker）或动态加载器（dynamic loader）是负责在运行时将程序所需的动态库（shared libraries）加载到内存中，并解析程序对这些库中函数和变量的引用的系统程序。它确保了动态链接的可执行文件能够在运行时找到并使用它们所依赖的共享库文件。

- 动态链接器本身不是共享库，而是操作系统提供的一个特殊程序，通常是系统的一部分。在Linux系统中，动态链接器通常是ld-linux.so（对于32位系统是ld-linux.so.2，对于64位系统是ld-linux-x86-64.so.2等等）。这个程序在可执行文件开始执行时由操作系统自动调用，负责：

- 加载共享库：根据可执行文件的动态链接信息，动态链接器将所需的共享库加载到内存中。
- 符号解析：动态链接器解析程序中未定义的符号（如函数和变量的引用），并将这些符号绑定到加载到内存中的共享库提供的相应符号上。
- 重定位：调整代码和数据的地址引用，确保程序中的引用指向正确的内存地址。

- 动态库：

- 是包含代码和数据的文件，这些代码和数据可以被多个程序同时使用。与静态库不同，动态库在程序运行时被加载到内存中，而不是在编译时被链接到每个程序中。动态库在Linux系统中通常有.so（shared object）扩展名，在Windows中是.dll（dynamic-link library）文件。

- 动态库与可执行文件的对比**

- 相似之处

- 格式相似：动态库和可执行文件在很多操作系统中都是遵循相同格式的，例如，在Linux和Unix-like系统中，它们通常都是ELF（Executable and Linkable Format）格式。这意味着操作系统的加载器可以使用相同或类似的机制来读取和处理这些文件。
  - 都可以包含代码和数据：无论是动态库还是可执行文件，它们都可以包含代码段和数据段。

- 不同之处

- 用途不同：可执行文件包含了运行一个完整程序所需的所有指令和资源，而动态库包含的是可以被多个程序共享使用的代码和数据。动态库的主要目的是重用代码，减少程序的内存占用和磁盘空间使用。
  - 启动方式不同：可执行文件可以被操作系统直接加载和执行。动态库则不能直接执行，它们是被其他程序调用时动态加载到内存中的。当一个程序启动时，如果它依赖于某些动态库，操作系统的动态链接器会负责加载这些库。
  - 符号解析和重定位：动态库在被加载时，需要通过动态链接器进行符号解析和地址重定位。这一步骤是必需的，因为动态库中的函数和数据在不同程序中可能被加载到不同的内存地址。

- 动态库的实现**

- 假设我们有一个加法函数，我们希望将其编译为动态库。首先，创建一个名为add.c的源文件：

```
// add.c
int add(int a, int b) {
    return a + b;
}
```

- 编译动态库：使用GCC（GNU Compiler Collection）编译这个文件为动态库。打开终端或命令行界面，然后使用以下命令：

```

# 对于linux或Unix系统
gcc -fPIC -o libadd.so add.c

# 选项的含义
-fPIC          # 表示生成位置无关代码（Position Independent Code），这对于
                动态库是必需的，因为它允许代码在内存中被任意位置加载
-o libadd.so   # 指定输出文件的名称。在Linux系统中，动态库通常以 .so（共享对
                象）扩展名结尾。

# windows系统编译动态库
gcc -fPIC -o add.dll add.c
# 在Windows上，动态库通常以 .dll（Dynamic Link Library）扩展名结尾。

```

### ■ 使用动态库

- 生成动态库后，你可以在其他C程序中使用它。为了使用这个动态库，你需要在使用它的程序中声明add函数的原型，并在链接时指定动态库的位置。这里不详细展开如何调用动态库中的函数，因为这涉及到动态链接库的加载和符号解析，通常需要使用动态加载机制（如dlopen和dlsym在Unix-like系统中，LoadLibrary和GetProcAddress在Windows中）或在编译时链接动态库。

## 软件包和包管理器

- 软件包介绍：开源软件最初只提供了打包的源码文件，用户必须自己编译每个想在GNU/LINUX上运行的软件。用户急需系统能提供一种更加便利的方法来管理这些软件，当Debian诞生时，这样一个管理工具dpkg也就应运而生，可用来管理deb后缀的“包”文件。

从而著名的“package”概念第一次出现在GNU/Linux系统中，稍后Red Hat才开发自己的rpm包管理系统

- 软件包位置：
  - 一般在光盘的Appstream和BaseOS中
  - 查看光盘内文件：先挂载光盘再查看

```
mount /dev/sr0 /mnt
```

- 软件包中的文件分类
  - 二进制文件
  - 库文件
  - 配置文件
  - 帮助文件

范例：利用cpio工具查看包文件列表

```
# rpm2cpio 包文件 | cpio -itv    预览包内文件
# rpm2cpio 包文件 | cpio -id "*.conf" 释放包内文件
```

在Rocky中，一般cpio和rpm2cpio都是安装好的

- 程序包管理器

- 功能：将编译好的应用程序的各组成文件打包一个或几个程序包文件，利用包管理器可以方便快捷地实现程序包的安装，卸载，查询，升级，校验等管理操作。
- 主流的程序包管理器
  - redhat:rpm文件，**rpm包管理器**
  - debian:deb文件，**dpkg包管理器**
- 包命名
  - rpm包命名方式：
    - name-VERSION-release.arch.rpm
    - 即：[名称]-[版本].[发布].[架构].rpm
    - 名称 (Name): 这是软件/包的名称。例如，对于 httpd (Apache HTTP 服务器) 的 RPM，名称就是 httpd
    - 版本 (Version): 这表示软件的主版本号。例如，如果 Apache HTTP 服务器的版本是 2.4.6，则版本号就是 2.4.6。
    - 发布 (Release): 这表示软件的发行版或修订版。当软件包维护者对软件进行修改，例如修复 bug 或进行优化，但没有更改主版本号时，就会更改这个数字。例如，对于第一次打包的 2.4.6 版本的 Apache，发布版本可能是 1。如果该包稍后被重新打包（例如，由于一个修复），则发布版本可能会增加到 2。
    - 架构 (Architecture): 这描述了软件编译的目标硬件架构。常见的值包括：
      - i386, i686 - 用于 32 位 Intel/AMD 处理器
      - x86\_64 - 用于 64 位 Intel/AMD 处理器
      - noarch - 表示软件是架构无关的，例如纯脚本软件包
      - 还有其他的，例如 arm, ppc 等，表示其他硬件架构
    - .rpm: 这是文件的扩展名，表明这是一个 RPM 文件。
- 包的分类
  - Application-VERSION-ARCH.rpm: 主包
  - Application-devel-VERSION-ARCH.rpm: 开发子包
  - Application-utils-VERSION-ARCH.rpm: 其他子包
  - Application-libs-VERSION-ARCH.rpm: 其他子包
- 包的依赖
  - 软件包之间可能存在依赖关系，甚至循环依赖，即：A包依赖B包，B包依赖C包，C包依赖A包。安装软件包时，会因为缺少依赖的包，而导致安装包的失败。
  - 解决依赖包管理工具：
    - yum:rpm包管理器的前端工具
    - dnf: Fedora 18+ rpm包管理器前端管理工具，CentOS 8版本代替yum
    - apt: deb包管理器前端工具
    - zypper: suse上的rpm包管理工具
- 程序包管理器相关文件
  - 包文件组成（每个包独有）
    - 包内文件
    - 元素据，如：包的名称，版本，依赖性，描述等

- 可能会有包安装或卸载时运行的脚本
- 数据库（公共）：/var/lib/rpm
  - 程序包名称及版本
  - 依赖关系
  - 功能说明
  - 包安装后生成的各文件路径及校验码信息
- 第三方组织提供
  - Fedora-EPEL: Extra Packages for Enterprise Linux

```
https://fedoraproject.org/wiki/EPEL  
https://mirrors.aliyun.com/epel/  
https://mirrors.cloud.tencent.com/epel/
```

- SCL:Software Collections, 提供较高版本的第三方软件

```
https://wiki.centos.org/SpecialInterestGroup/SCLo
```

- Community Enterprise Linux Repository:支持最新的内核和硬件相关包

```
http://www.elrepo.org
```

- 软件项目官方站点

```
http://yum.mariadb.org/10.4/centos8-amd64/rpms/  
http://repo.mysql.com/yum/mysql-8.0-community/el/8/x86_64/
```

- 搜索引擎

```
http://pkgs.org  
http://rpmfind.net  
http://rpm.pbone.net  
https://sourceforge.net/
```

- 自己制作

- 将源码文件，利用工具，如：rpmbuild,fpm等工具制作成rpm包文件。

## rpm包管理器

- CentOS系统上使用rpm命令管理程序包
- 功能：安装，卸载，升级，查询，校验，数据库维护
  - 问题：一般来说，rpm管理器更多的是使用查询功能，而很少使用安装功能，因为rpm包管理器不支持包的依赖关系

```
使用rpm包管理器 - 安装
```

```
# rpm -i <rpm包名>  
# rpm -ivh <rpm包名> -- 能够看到安装过程
```

查询安装是否成功

```
# rpm -q <软件名> -- 这里不是包名，仅仅是开头的软件名  
# rpm -q 后面必须接完整的软件名，很不好用，建议用qa在过滤  
# rpm -qa | grep '软件名'  
-- 使用qa查询所有安装过的包，通过grep筛选确认是否安装成功
```

rpm管理器 - 卸载

```
# rpm -e <软件名>  
-- 还是因为依赖关系问题，所以一般不使用rpm管理器进行安装，卸载
```

重点：rpm管理器 - 查询

```
# rpm -ql <软件名>  
-- 查询这个软件包中包含的文件列表
```

```
# rpm -q --scripts <软件名>  
-- 查看这个软件包中的脚本文件
```

```
# rpm -qi <软件名>  
-- 查看软件包的详细信息
```

```
# rpm -qf <软件名>  
-- 可以查询到磁盘文件来源于哪个包
```

```
# rpm -qc <软件名>  
-- 只查看软件包里的配置文件
```

```
# rpm -qd <软件名>  
-- 只查看软件包里的文档
```

```
# rpm -qa --last  
-- 查看最近安装的所有包
```

## ● 包校验

```
# rpm -K <软件名>  
-- 查看软件的rpm包是否合法  
  
# 在查询合法性之前，必须导入所需公钥  
# 导入公钥(添加第三方软件仓库时尤其有用)  
rpm --import /etc/pki/rpm-gpg/RPM-XXXXXXXXXXXX
```

```
# 查看已导入的公钥  
rpm -qa "gpg-pubkey"  
  
# 查看公钥  
rpm -qi gpg-pubkey-XXXXXX
```

```
# rpm -V <软件名>  
-- 查看软件在安装后，是否被改过  
  
# rpm -Va  
-- 系统所有包，安装后，被改过的都列出来
```

## 实验：误删除rpm命令（/usr/bin/rpm），如何修复

```
# 进入救援模式  
# 使用救援模式的rpm包进行安装，安装至/mnt/sysroot  
rpm -ivh /run/install/repo/BaseOS/Packages/r/rpm-4.xxxxxx --root=/mnt/sysroot --force  
  
# 检查是否安装成功  
ls /mnt/sysroot/usr/bin/rpm  
  
# 安装成功后重启
```

## 数据库维护

- rpm包安装时生成的信息都放在rpm数据库中

```
/var/lib/rpm
```

- 可以重置数据库

```
rpm {--initdb|--rebuilddb}  
  
# 选项详解  
--initdb      # 初始化，如果事先不存在数据库，则新建之，否则不执行任何操作  
--rebuilddb   # 重建已安装的包头的数据库索引目录
```

## 包更新日志

```
rpm -q --changelog <软件名>  
其实是一个doc软件包中的一个文件  
rpm -qd <软件名> 如果该软件有更新日志，则doc文件中含有changelog文件
```

## yum和dnf

- 作用：CentOS使用yum,dnf解决rpm的包依赖关系
- YUM：Yellowdog Update Modifier，rpm的前端程序，可解决软件包相关依赖性，可在多个库之间定位软件包，up2date的替代工具，CentOS8用dnf代替了yum，不过保留了和yun的兼容性，配置也是通用的
- yum和rpm的关系：
  - yum依赖于rpm，安装时间接调用rpm，如果rpm被破坏，yum也会失效
- yum/dnf 工作原理
  - yum/dnf是基于C/S模式
    - yum 服务器存放rpm包和相关包的元数据库
    - yum 客户端访问yum服务器进行安装或查询等
  - yum 实现过程：先在yum服务器上创建yum repository（仓库），在仓库中事先存储了众多rpm包（一般放在Packages目录下），以及包的相关元数据文件（放置在特定目录repodata下），当yum客户端利用yum/dnf工具进行安装时，会自动下载repodata中的元数据，查询元数据是否存在相关的包及依赖关系，并再次访问yum服务器，自动从仓库中找到相关包下载并安装。

- yum客户端配置

- yum客户端配置文件

```
# /etc/yum.conf      为所有仓库提供公共配置  
# /etc/yum.repos.d/*.repo:    为每个仓库提供配置文件
```

每个仓库对应一个配置文件

- yum公共配置 (yum.conf)

```
gpgcheck=1          # 安装包前要做包的合法和完整性校验  
installonly_limit=3    # 同时可以安装3个包，最小值为2，如设0或1，为不限制  
clean_requirements_on_remove=True   # 删除包时，是否将不再使用的包删除  
best=True           # 升级时，自动安装最新版，即使缺少包依赖  
skip_if_unavailable=False     # 跳过不可用
```

- repo仓库配置文件内容

```
[repositoryID]  
name=Some name for this repository  
baseurl=url://path/to/repository  
enabled={1|0}    # 默认为1，0表示禁用该仓库  
gpgcheck={1|0}    # 默认1，检查包是否合法，0表示不检查
```

gpgcheck必须配合gpgkey使用，否则就设置0，不检查

gpgkey=URL # 通过key的路径，来实现gpgcheck的检查

# 注意：yum仓库指向路径一定必须是repodata目录所在目录

-----  
相关变量

```
# 下面的变量是yum自身带的变量，Linux脚本不支持  
yum的repo配置文件中可用的变量  
$releasever: 当前os的发行版的主版本号，如：8, 7, 6  
$arch: CPU架构，如：aarch64,i586,i686,x86_64  
$basearch: 系统基础平台：i386,x86_64  
$contentdir: 表示目录，比如：centos-8,centos-7  
$YUM0-$YUM9: 自定义变量
```

- yum相关指令

```
# yum repolist    查看所有的yum源客户端仓库  
  
# yum repolist -v    查看yum仓库的详细信息  
  
# yum repolist [all | enabled | disabled]  
  
# yum repolist --repoid=xxx -v  # 显示指定源
```

```
# yum install <软件名>      下载软件

# yum remove <软件名>      卸载软件

# yum list <软件名>      查询软件
支持模糊查询，通配符，比如: msm* 表示msm开头的软件
如果查询到的yum源前面有@，说明已经安装，反之，没安装
结果包含已安装的包，和源里的可用包

# yum list --installed 查询已安装过的包

# yum list updates 显示所有本地可更新的包

# yum list --available --showduplicates <软件名>
列出该软件在当前源下所有版本的包

# yum provides <软件名路径>  查询硬盘上没安装的软件来自于哪个包

# yum -y install --downloadonly --downloaddir=路径 <软件名>
下载某个软件的包到指定目录，但是不安装

# yum -y install /data/httpd/*.rpm
使用yum下载目录下的所有rpm包

# yum info <软件名>
查询软件说明，相当于rpm -qi

# yum search [string1]
搜索和关键词相关的包

# dnf repoquery -l <软件包>
查看未安装的软件包，安装后会在硬盘生成哪些文件

# dnf clean all
清理软件包元信息缓存
# 作用场景
这个命令通常在遇到与包数据库相关的问题或者当系统长时间没有更新且想要确保软件列表完全更新时使用。清理后，第一次运行 DNF 命令可能会比较慢，因为需要重新下载元数据缓存

# yum history
查看安装的历史

# yum history info <history的编号>
根据编号查看历史下载的具体细节

# yum updatenfno
查看互联网上新版本的软件信息
# yum updateinfo info 查看详细信息
# yum updateinfo info bugfix | newpackage|security

# yum grouplist
查看包组

# yum groupinstall <包组名>
安装包组
```

```
# yum groupremove <包组名>
卸载包组
```

- yum安装失败的原因
  - yum 配置格式有错
  - yum 元数据过旧，清理缓存
  - yum 源出问题，或网络有问题
- 包组
  - 作用："包组"（Package Group）是一个方便的概念，它允许你将相关的软件包归纳为一个组进行管理。这样，用户可以通过安装单个包组来批量安装一组具有相似功能或相关依赖的软件包，而不需要逐一安装每个包。
  - 包组的特点：
    - 便捷性：包组使得安装具有相似用途或相关依赖的一系列软件包更加方便快捷。
    - 管理简化：通过包组，系统管理员可以更容易地管理系统安装的软件，尤其是在进行初始系统设置或批量部署时。
    - 可定制：某些发行版允许用户在安装过程中选择特定的包组进行安装，从而实现更加定制化的安装体验。
    - 示例：

```
sudo dnf groupinstall "Development Tools"
```

## 实现私用yum仓库

步骤一：在yum server下搭建web服务，保证其他机器能通过web服务访问本机

步骤二：在yum server机上搭建yum仓服务（Packages repodata）

步骤三：在client机上将yum的repos源指向yum server机

## 服务端配置

```
# 安装web服务
yum install -y httpd

# 关闭防火墙
systemctl disabled --now firewalld.service

# 开启web服务
systemctl enable --now httpd.service

# 将阿里云的extras源的相关数据下载到本地，给客户端使用
yum reposync --repoid=nju-extras --download-metadata -p /var/www/html

# 将本地光盘中的内容CP到web目录中，给客户端使用
mkdir /cdrom
mount /dev/sr0 /cdrom
cp -r /cdrom/Baseos/* /var/www/html/Baseos
```

## yum仓库工具

```
# CentOS 8 dnf 工具集成  
dnf reposync --repoid=REPOID --download-metadata -p /path  
  
# CentOS 7 以前版本, reposync工具来自于yum-utils包  
reposync --repoid=REPOID --download-metadata -p /path
```

## 创建YUM仓库工具

- 可以根据目录中的rpm生成repodata元数据

```
createrepo [option] <directory_to_index>  
  
# 常用选项  
-v # 显示详细的操作信息  
-o|--outputdir # 指定生成的仓库元数据的输出目录  
-d|--database # 生成sqlite数据库文件, 可以加快包管理器的处理速度, 并支持一些高级  
查询功能  
--update # 更新现有仓库元数据  
--excludes <pattern> # 排除指定规则的文件  
--includepkgs # 指定规则的包创建元数据, 与excludes相反  
--compress-type <类型> # 指定压缩类型, 这会影响仓库元数据文件的压缩方式  
--workers <数量> # 指定生成元数据时使用的进程数, 可以加快元数据生成速度
```

- 示例

```
# 创建仓库元数据  
createrepo /path/to/repository  
# 创建后会在指定目录下, 生成一个repodata目录  
  
# 更新仓库  
createrepo --update /path/to/repository  
  
# 使用多进程创建仓库  
createrepo --workers 4 /path/to/repository  
  
# 在/etc/yum.repo.d/下, 配置新创建的仓库*.repo文件  
  
# 使用yum repolist --repoid=myself_test -v 测试仓库是否配置成功  
  
# 如果出现进程占用的情况, 使用yum clean all, 可以清除缓存, 解决问题
```

## yum Troubleshooting

```
# yum的配置文件格式或路径有问题  
解决*.repo文件格式  
  
# yum cache  
yum clean all  
  
# 重新建立缓存  
yun makecache
```

```
# 元数据缓存地址
```

```
/var/cache/dnf
```

```
/var/cache/yum
```

```
# 网络不通
```

```
网卡配置
```

## DNF介绍

- DNF，是新一代RPM软件包管理器。DNF软件包采用python编写，yum程序在安装的过程中，如果被终止，下次在执行将无法解决依赖，DNF可解决此问题

```
# 配置文件
```

```
/etc/dnf/dnf.conf
```

```
# 仓库文件
```

```
/etc/yum.repos.d/*.repo
```

```
# 日志
```

```
/var/log/dnf.rpm.log
```

```
/var/log/dnf.log
```

```
# 使用帮助
```

```
man dnf
```

## Ubuntu软件管理

- Debian 软件包通常为预编译的二进制格式的扩展名".deb",类似于rpm文件，因此安装快速，无需编译软件。包文件包括特定功能或软件所必须的文件、元数据和指令
  - dpkg: package manager for Debian, 类似于rpm, dpkg是基于Debian的系统的包管理器。可以安装，删除和构建软件包，但无法自动下载和安装软件包或其依赖性
  - apt: Advanced Packaging Tool, 功能强大的软件管理工具，甚至可升级整个Ubuntu的整个系统，基于客户/服务器 (c/s)

rhel系列与 debian系列包管理对比

操作系统	包文件后缀	本地包管理器	网络包管理器	网络包管理工作模式	配置文件
rhel	rpm	rpm	yum/dnf	C/S	/etc/yum.repos.d/*.repo
debian	deb	dpkg	apt	C/S	/etc/apt/sources.list

```
# 判断软件是否已安装
```

```
dpkg -V <软件包>
```

```
# $?返回0，则安装成功
```

```
# 安装
```

```
dpkg -i xxxxxxxx.deb
```

```
# 列出安装详情
```

```
dpkg -l xxxxxxxx.deb
```

```
# 可以接通配符
```

```
dpkg -l <通配符>
```

```

# 卸载
dpkg -r <软件名>

# 根据条件列出已安装的包
dpkg --get-selections v*
# 列出以v开头的包名的包

# 显示所有可以安装的包
dpkg -p

# 用于查询某个特定文件属于哪个已安装的软件包,dpkg -S 命令只能查询到已安装软件包的信息。
dpkg -S "XXXX"

# 显示包的详细信息
dpkg -s "XXX"

# 列出已安装包的所有文件
dpkg -L 应用程序名

# 列出包内所有文件
dpkg -L "XXXXX.deb"

```

## dpkg -l 显示解读

```

Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-await/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
|| / Name          Version           Architecture Description
+++-+-----+-----+-----+-----+
=====
un  v4l2loopback-dkms      <none>            <none>      (no description
available)
un  v4l2loopback-modules    <none>            <none>      (no description
available)
ii  vim              2:8.2.3995-1ubuntu2.16  amd64      Vi IMproved -
enhanced vi editor
un  vim-athena        <none>            <none>      (no description
available)
ii  vim-common        2:8.2.3995-1ubuntu2.16  all        Vi IMproved -
Common files
un  vim-doc           <none>            <none>      (no description
available)
un  vim-gtk3          <none>            <none>      (no description
available)
un  vim-lua            <none>            <none>      (no description
available)
ii  vim-nox           2:8.2.3995-1ubuntu2.16  amd64      Vi IMproved -
enhanced vi editor - with scripting languages support
un  vim-perl           <none>            <none>      (no description
available)
un  vim-python3        <none>            <none>      (no description
available)
un  vim-ruby           <none>            <none>      (no description
available)

```

ii	vim-runtime	2:8.2.3995-1ubuntu2.16	all	Vi IMproved -
Runtime files available)		<none>	<none>	(no description
un vim-scripts available)		<none>	<none>	(no description
un vim-tcl available)		<none>	<none>	(no description
ii vim-tiny	2:8.2.3995-1ubuntu2.16	amd64	Vi IMproved -	
enhanced vi editor - compact version				
un virtualbox-guest-modules available)		<none>	<none>	(no description
un vsearch available)		<none>	<none>	(no description

### # 共7个字段

Desired Status Err Name Version Architecture Description

#### # Desired 期望状态

u	Unknown	# 没有安装过
i	Install	# 请求安装
r	Remove	# 请求卸载
p	Purge	# 请求卸载并清理
h	Hold	# 保持

#### # Status 当前状态

n	Not	# 软件没有安装
i	Inst	# 安装完成并完成配置
c	Conf-files	# 已卸载，但还有保留配置文件
u	Unpacked	# 已解压缩，但没有配置
f	half-conf	# 配置时出错
h	Half-inst	# 安装时出错
w	trig-await	# 触发器等待
t	Trig-pend	# 触发器是未决状态

#### # Err 错误状态

空	# 正常情况下为空
h	# 被锁定，有其他包对此依赖，无法升级
r	# 被损坏，需要重装才能使用
x	# 损坏且被锁定

#### # NAME 包名

#### # Version 版本

#### # Architecture 平台架构

#### # Description 包的描述信息

#### # 前三列常见组合

ii	# 安装成功
pn	# 安装后卸载
un	# 没有安装过
iu	# 安装了但没有配置
rc	# 已卸载，但还有配置文件

## apt命令用法

```
# 列出所有包  
apt list  
  
# 列出所有已安装的包  
apt list --installed  
  
# 列出所有可升级的包  
apt list --upgradeable  
  
# 指定包名，使用通配符进行模糊查询  
apt list *sql*  
  
#-----  
  
# 默认在包名和描述信息中搜索，支持正则  
apt search  
  
apt search --names-only <软件名>  
  
# 查询包的具体信息  
apt info <软件名>  
  
# 显示所有版本
```

## apt包索引配置文件

```
# 配置系统默认编辑器的文件  
# Rocky/CentOS是通过环境变量$EDITOR来进行控制  
  
# select-editor命令可以用来配置系统默认编辑器文件
```

- apt配置文件位置

```
/etc/apt/sources.list  
/etc/apt/sources.list.d/  
# 该文件更新完后，要apt update进行更新软件包元数据  
  
# 在ubuntu中有两个重要目录，分别是dists和pool  
# dists目录中存放的是该仓库中的元数据  
# pool目录中存放的是具体的包文件
```

```
deb URL section1 section2  
  
# 字段说明  
deb      # 固定开头，表示是二进制包的仓库，如果deb-src开头，则表示是源码库  
URL      # 库所在的地址，可以是网络地址，也可以是本地镜像地址  
section1 # Ubuntu版本的代号，可在os-release查看(VERSION CODENAME)  
          # section1           主仓  
          # section1-backports   后备仓，该仓中软件当前版本不一定支持
```

```
# section1-security      修复仓，主要用来打补丁，有重大漏洞需要在当前版本修复
时，会放在此仓
# sections-updates     非安全性更新仓，不影响系统安全性的子版本迭代放在此仓
# section1-proposed    预更新仓，可理解为新软件的测试放在此仓
section2   # 软件分类
# main完全自由软件
# restricted不完全自由的软件
# universe社区支持的软件
# multiverse非自由软件
```

## 查看apt的安装历史

```
cat /var/log/dpkg.log
```

## 查看yum的安装历史

```
yum history
```

## snap 工具

- 默认snap应用格式包，专为物联网设备，嵌入式平台设计的迷你ubuntu

## 程序包编译

- C、C++的源码编译：使用make项目管理器

configure脚本 --> Makefile.in --> Makefile

- 相关开发工具
  - autoconf: 生成configure脚本
  - automake: 生成Makefile.in

- java的源码编译：使用maven
- 源码编译的好处
  - 可以实现软件功能的私人定制
  - 可以控制安装路径

## 编译安装准备

```
# CentOS
yum install gcc make gcc-c++ glibc glibc-devel pcre pcre-devel openssl openssl-devel systemd-devel zlib-devel vim lrzsz tree tmux lsof tcpdump wget net-tools iotop bc bzip2 zip unzip nfs-utils man-pages

# Ubuntu
sudo apt update
sudo apt install gcc make g++ libc6-dev libpcre3 libpcre3-dev libssl-dev libsystemd-dev zlib1g-dev vim lrzsz tree tmux lsof tcpdump wget net-tools iotop bc bzip2 zip unzip nfs-common manpages
```

- C语言源代码编译安装过程

利用编译工具，通常只需要三个大的步骤

- ./configure
  - 通过选项传递参数，指定安装路径、启用特性等；执行时会参考用户的指定以及 Makefile.in文件生成Makefile
  - 检查依赖到的外部环境，如依赖的软件包
- make 根据Makefile文件，会检查依赖的环境，进行构建应用程序
- make install 复制文件到相应路径

- 编译安装实战案例（tree安装）：

1. 到官网使用wget download\_url，下载新版tree的源码到当前目录并解压

```
# wget download_url  
# tar xvf tree-1.8.0.tgz
```

2. 进入解压后的目录，阅读README和INSTALL

```
# cd tree-1.8.0/  
# less README  
# less INSTALL
```

3. 编译准备，修改自定义安装地址，或指定功能

```
# vim Makefile
```

由于tree文件源码比较简单，因此不需要自己生成Makefile文件，解压目录下直接有

```
# prefix = /apps/tree
```

指定安装到这个目录下

4. 编译

```
# make
```

5 安装

```
# make install
```

6. 修改PATH变量

6-1:方法一 使用软链接

```
# echo $PATH  
# ln -s 软件安装路径 环境变量路径
```

6-2: 方法二 修改配置文件

```
# vim /etc/profile.d/tree.sh  
添加PATH=/apps/tree/bin:$PATH  
# ./etc/profile.d/tree.sh 使文件生效
```

7. 清除缓存

```
# hash -r
```

## 实验：编译安装nginx

```
# 下载nginx源码包
wget http://nginx.org/download/nginx-1.23.0.tar.gz

# 解压
tar -xf nginx-1.23.0.tar.gz

# 指定安装目录，同时开启http_ssl_module
./configure --prefix=/tmp/nginx --with-http_ssl_module

# 如果中间失败，提示缺少依赖项，通过yum search 查找软件包名称，然后安装，安装后重新执行
./configure ...

# 经过补全依赖项后，成功编译，生成Makefile文件，执行make
make

# 使用make install 将文件按makefile规则转移到对应目录下
make install

# 安装成功后，修改nginx文件，修改/tmp/nginx/conf/nginx.conf将user xxx; 改为 user root

# 然后关闭防火墙
systemctl stop ufw      # Ubuntu
systemctl stop firewalld # CentOS|Rocky

# 将/tmp/nginx/sbin/nginx 创建软链接到PATH路径下
ln -s /tmp/nginx/sbin/nginx /usr/local/bin/nginx

# 启动nginx
nginx

# 服务开启后测试，成功~~~~
```

## Ubuntu软件包管理工具

- 概述

```
dpkg 相当于rpm，不支持包的依赖关系的解决
apt 相当于yum，支持包的依赖解决
```

- dpkg 包管理器

```
帮助参看: man dpkg
```

```
安装包，不支持包的依赖
```

```
# dpkg -i package.deb
```

```
删除包（包括配置文件）
```

```
# dpkg -P package
```

```
列出当前已安装的包，类似rpm -qa
```

```
# dpkg -l package
```

列出该包中所包含的所有文件 类似`rpm -q1`

```
# dpkg -L package
```

查看包的具体信息

```
# dpkg -s package
```

查询软件来自于哪个包

```
# dpkg -S software_name
```

- `apt`

安装软件包

```
# apt install
```

卸载移除软件包及其配置文件

```
# apt purge
```

刷新存储库索引

```
# apt update
```

升级所有可升级的软件包

```
# apt upgrade
```

搜索应用程序

```
# apt search
```

- Ubuntu 配置文件

ubuntu软件源的下载地址， 默认外网

```
/etc/apt/sources.list
```

更改国内资源的方式：直接`copy`,

# Service and Process Management

## GRUB

- 什么是grub: GRUB (Grand Unified Bootloader) 是一个多操作系统引导程序。它是许多Unix和Linux系统上的默认引导加载器。GRUB的主要功能是允许用户在计算机启动时选择多个操作系统或不同的内核配置。
- grub 配置文件
  - /etc/default/grub 设置基本的grub配置
  - /etc/grub.d 设置更详细的引导设置
  - /boot/grub2/grub.cfg
  - grub2-mkconfig -o /boot/grub2/grub.cfg

```
/etc/default/grub 默认配置文件参数
```

```
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=1G-4G:192M,4G-64G:256M,64G-:512M
resume=/dev/mapper/r1_192-swap rd.lvm.lv=r1_192/root rd.lvm.lv=r1_192/swap
net.ifnames=0"
(net.ifnames=0 是修改网卡名称eth0的参数设置)
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
```

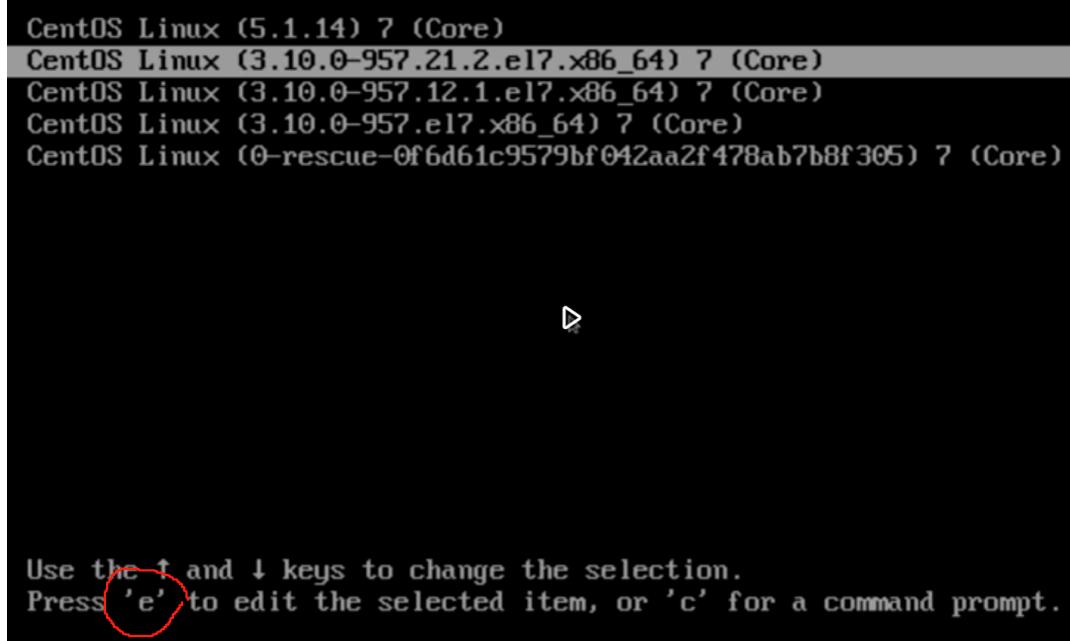
```
-----
```

查看默认引导内核的版本号

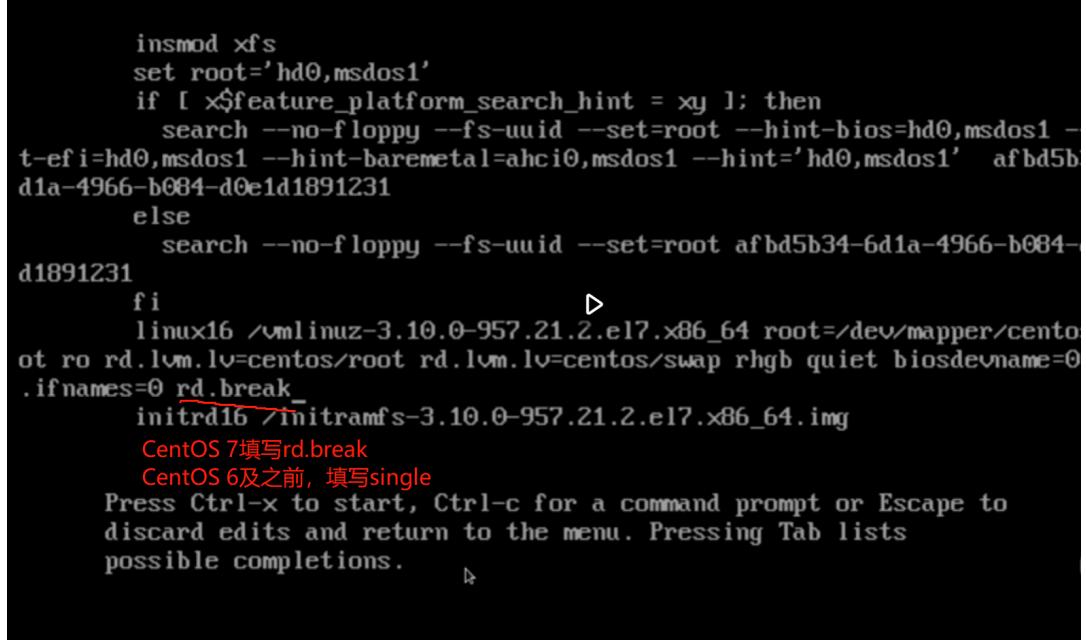
```
# grub2-editenv list
```

- 忘记root密码, 如何重置

- 步骤一：在启动选择项时，选择需要引导的内核，然后按e进入选择编辑项



- 步骤二：找到内核参数，在后面填写rd.break，然后按Ctrl-x进行启动



- 步骤三：启动之后，目前的根目录是挂载在内存中虚拟的文件系统上，应该挂载到硬盘的根目录上，使用mount

- mount -o remount,rw /sysroot
- chroot /sysroot
- 步骤四：进入硬盘的根目录后，就可以重置passwd了使用：echo <密码> | passwd --stdin root

## 进程管理

### 进程的概念与进程查看

- 进程的概念：进程是运行中的程序，从程序开始运行到终止的整个生命周期是可管理的
  - 例如C程序的启动是从main函数开始
    - int main(int argc,char\*argv[])
  - 终止的方式并不唯一，分为正常终止和异常终止

- 正常终止也分为从main返回，调用exit等方式
- 异常终止分为abort、接收信号等
- 进程的查看
  - 查看命令
    - ps(process status)

查看所有进程的状态

```
# ps -e
```

查看所有进程状态的详细信息

```
# ps -ef
```

示例:

UID	PID	PPID	C	S TTY	TIME	CMD
root	1	0	0	Aug27 ?	00:00:02	
/usr/lib/systemd/systemd		--switched-root	--system	--deserialize 31		
root	2	0	0	Aug27 ?	00:00:00	[kthreadd]
root	3	2	0	Aug27 ?	00:00:00	[rcu_gp]
root	4	2	0	Aug27 ?	00:00:00	[rcu_par_gp]
root	5	2	0	Aug27 ?	00:00:00	[slub_flushwq]
root	6	2	0	Aug27 ?	00:00:00	[netns]
root	8	2	0	Aug27 ?	00:00:00	[kworker/0:0H-events_highpri]
root	10	2	0	Aug27 ?	00:00:00	[kworker/0:1H-events_highpri]

参数详解:

UID: 有效用户id

PID: 进程id

PPID: 父进程id

查看线程情况

```
# ps -elf
```

```
```
- pstree
- top 更全面的看到进程的运行状态
```

```

top的详解

示例:

```
top - 10:58:20 up 19:40,  3 users,  load average: 0.00, 0.00, 0.00
Tasks: 224 total,   1 running, 223 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 1750.8 total,    697.8 free,     531.6 used,    688.5 buff/cache
MiB Swap: 2072.0 total,   2072.0 free,       0.0 used.  1219.2 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
17	root	20	0	0	0	0	I	0.3	0.0	0:01.94	rcu_preempt
836	root	20	0	241000	11256	7496	S	0.3	0.6	1:08.54	vmtoolsd
1	root	20	0	107588	17084	10600	S	0.0	1.0	0:02.61	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.03	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp

参数详解：

**up 19:40:** 意味着系统已经运行了19小时40分钟。

**3 users:** 表示当前系统有两个用户正在登录

**Load average:** 平均负载，后面跟3个值，分别表示1分钟，5分钟，15分钟的负载情况，1表示满负荷，用来查看系统的繁忙程度

**Load Average** 是如何计算的？

**Load average**的值反映了在特定时间段内的平均就绪进程数。“就绪”意味着进程要么正在运行，要么在等待CPU时间来运行。这也包括等待I/O的进程（例如，等待硬盘、网络等）

如何解释 **Load Average**？

解释这些数字最简单的方法是与你的CPU核心数进行比较。

如果你的系统只有一个CPU核心，那么**load average**为1意味着你的系统完全饱和。

如果你有4个CPU核心，那么数字4.00意味着所有核心都被完全利用。

一些基本的解读规则：

如果1分钟的负载（第一个数字）远高于5或15分钟的负载，系统的负载可能正在增加。

如果1分钟的负载（第一个数字）远低于5或15分钟的负载，系统的负载可能正在减少。

负载高于你的CPU核心数可能意味着系统上有竞争或者某种瓶颈。

这个数字是好是坏？

如果**load average**低于核心数，通常意味着系统有剩余的计算能力。

如果**load average**和核心数大致相等，系统可能达到了它的最大吞吐量。

如果**load average**远大于核心数，这可能意味着系统过载。

**Tasks:224 total :**表示共有224个进程在运行

**%Cpu(s):** 后面的数据表示Cpu资源使用和分配的百分比；(s) 表示多核cpu的使用平均值，按数字1，可以分开显示每个cpu的使用情况

**MiB Mem:** 表示内存的使用分配情况

MiB Swap: 表示swap虚拟内存的使用情况

默认每3s更新依次数据，按s键可以自定义输入更新的频率时间

...

- 结论
  - 进程也是树形结构
  - 进程和权限有着密不可分的关系

## 进程的控制命令

- 前置知识:

- 程序: 由程序员编写的一组稳定的指令, 存在磁盘中, 他可能会也可能不会成为作业
- 作业: 从一个程序被选中执行, 到其运行结束并执行下一程序这个过程中, 该程序被成为作业
  - 作业的几种状态:
    - 被选中后, 驻留在磁盘上等待被调入内存
    - 在内存中, 等待CPU执行
    - 驻留在内存或磁盘中, 等待IO事件
    - 在CPU中执行 (进程)
  - 进程: 运行中的程序, 在运行中等待被CPU调用, 或正在被CPU执行的作业, 只要作业装入内存就成为一个进程
  - 三者之间的关系:
    - 作业是一种特殊状态的程序, 即被选中的程序
    - 进程是一种特殊状态的作业, 即进入内存的作业

- 进程的优先级调整

- nice 范围从-20到19, 值越小优先级越高, 抢占资源越多

更改进程执行的优先级

```
# nice -n 数值 ./程序名称
```

- renice重新设置优先级

更改正在运行的进程执行的优先级

```
# renice -n 数值 ./程序名称
```

- 进程的作业控制

- jobs
- &符号

将进程调整为后台运行, 这样前台的终端仍然能执行其他指令

```
# ./a.sh &
```

将后台的进程调回前台运行

```
# jobs      先查看后台运行的进程  
# fg 数值  这个数值是jobs查询后，进程前边的显示数值
```

将前台进程调回后台，同时停止

组合键: **ctrl + z**

重新启动停止的进程，并在后台运行（在前台运行是fg）

```
# bg 数值
```

## 进程的通信方式——信号

- 信号是进程间通信方式之一，典型用法：终端用户输入终端命令，通过信号机制停止一个程序的运行
- 使用信号的常用快捷键和命令

查看信号

```
# kill -l
```

其中：**SIGINT** 通知前台进程组终止进程 **ctrl + c**

**SIGKILL**立即结束程序，不能被阻塞和处理 **kill -9 pid**

## screen和系统日志

- screen

◦ 作用：

- 会话持久性：即使你断开SSH连接或关闭终端，screen会话中运行的程序仍然可以继续运行。这对于长时间运行的任务或持久服务非常有用。
- 多窗口管理：你可以在一个screen会话内创建多个窗口，并在它们之间轻松切换。
- 会话分享与附加：你可以随时从不同的终端附加到一个正在运行的screen会话。这对于多用户协作或远程故障排查非常有用。
- 

启动一个新的会话

```
# screen
```

列出当前所有的screen会话

```
# screen -ls
```

重新连接到一个已经存在的screen会话。

```
# screen -r [session_id]
```

创建一个新会话，并给它命名

```
# screen -S [session_name]
```

从当前会话中断（但不停止会话）

```
Ctrl a + d
```

- 系统日志

系统常规日志

```
# ls /var/log/messages
```

内核启动信息

```
# ls /var/log/dmesg
```

系统安全日志

```
# ls /var/log/secure
```

**journalctl** - 日志查看工具

```
# journalctl -u sshd --no-pager
```

**-u:** 查看指定服务的日志

**--no-pager:** 默认使用分页模式查看日志，使用**no-pager**之后，可以将内容重定向到文件或使用管道，进行文本处理

## 服务管理工具**systemctl**

- 服务（提供常见功能的守护进程）集中管理工具

- service CentOS 6及以前

- systemctl CentOS 7之后

**systemctl** 常见操作

```
# systemctl status service_name  
  
# systemctl start service_name  
  
# systemctl stop service_name  
  
# systemctl restart service_name  
  
# systemctl reload service_name  
  
# systemctl enable service_name //开机自动启动  
  
# systemctl disable service_name
```

服务级别

配置文件: /lib/systemd/system/\*.target

```
runlevel[0-6].target -> 不同的状态
```

查看当前服务的级别

```
# systemctl get-default
```

修改当前服务级别

```
# systemctl set-default multi-user.target
```

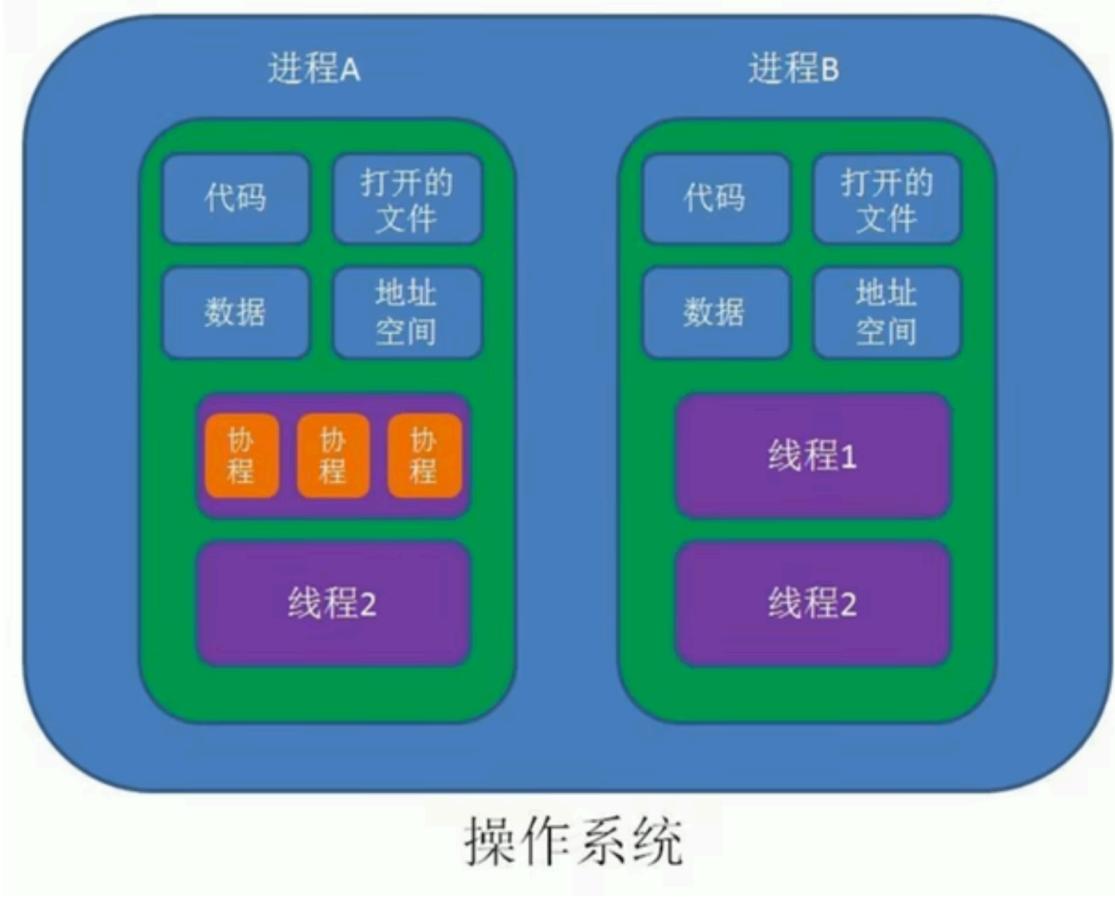
## 进程，系统性能和计划任务

### 进程和内存管理

- 内核功能：进程管理、内存管理、文件系统、网络功能、驱动程序、安全功能等
- 什么是进程：
  - Process：运行中的程序的一个副本，是被载入内存的一个指令的集合，是资源分配的单位
    - 进程ID(Process ID, PID)号码被用来标记各个进程(系统自动分配)
    - UID、GID、和SELinux语境决定对文件系统的存取和访问权限
    - 通常从执行进程的用户来继承
    - 存在生命周期
  - 进程创建
    - init：第一个进程（从CentOS 7之后，为systemd）
    - 进程：都是由其父进程创建，fork()，父子关系，COW(copy on write)
      - 一般来说，进程是由父进程调用fork()函数创建的，创建的时候基于COW机制
      - 前期，子进程和父进程共有一个内存空间，当数据发生变化的时候，子进程拷贝父进程的信息，生成子进程会给予进程分配新的内存空间
      - COW详解：Copy-On-Write 是一种计算机程序优化策略，它最大限度地延迟数据的复制，直到真正需要时才进行。当 fork() 被调用时，父进程的页表（内存页的索引）会被复制到子进程，但是物理内存页并不复制。取而代之的是，父进程和子进程都使用相同的物理页，这些页都标记为只读。

只有当父进程或子进程试图修改这些只读页中的某个值时，操作系统才会为写操作创建一个新的物理页（实际的复制操作），然后更新相应的页表条目以指向新的物理页。这就是为什么它被称为 Copy-On-Write：只有在真正需要写入时，数据才被复制。

- 进程，线程和协程



- 进程 (Process)

- 定义：进程是计算机中的程序关于其运行时的状态的一个实例。它是系统进行资源分配和调度的基本单位。每个进程都有自己的私有地址空间、代码、开放的文件、挂起的信号等。
- 隔离：每个进程在其自己的地址空间中运行，这意味着一个进程不能直接访问另一个进程的变量和数据结构。如果一个进程需要另一个进程的数据，它们必须通过进程间通信（IPC）机制，例如管道、信号、套接字等。

- 线程 (Thread)

- 定义：线程是进程内部的执行单位。一个进程可以拥有多个线程，这些线程共享进程的地址空间和资源，但每个线程都有自己的程序计数器、寄存器集和栈。
- 多线程：在多线程应用中，进程中的所有线程可以同时访问进程级的资源如堆。这允许更快的数据交换，但也增加了同步的复杂性，因为线程需要避免争用条件和其他并发相关的问题。

- 协程 (Coroutine)

- 定义：协程是一种轻量级的“线程”，但它的调度完全由用户控制，而不是由操作系统内核控制。协程为并发操作提供了方便，但没有真正的并行执行线程或进程那样的开销。
- 优势：由于协程的调度不涉及内核模式和用户模式之间的切换，因此上下文切换通常比线程更快、更轻便。协程也能有效地管理大量的并发任务，因为它们在内存使用和开销上比线程更为轻量。

- 关系：

- 进程与线程：线程是进程的子集。每个进程至少有一个线程（通常称为主线程），但可以有更多线程。线程共享父进程的内存空间。
- 线程与协程：协程可以被认为是“轻量级线程”。与线程不同，所有协程都运行在一个线程之内，并且由该线程的代码明确地调度。

- 进程与协程：协程为实现进程内部的并发提供了一种手段，但与进程相比，它们共享相同的地址空间。
- 区别：
  - 进程和线程的执行顺序是由操作系统决定和控制的
  - 协程的执行则是由程序员可以去控制的

#### 线程的查看

```
# pstree -p      其中花括号形式的就是线程，没有花括号则说明是单线程
```

通过文件查看线程信息

```
# cat /proc/PID/status | grep -i threads
```

通过进程**PID**，查找该进程文件所在路径

```
# ll /proc/PID/exe
```

直接查找指定程序的进程**PID**

```
# pidof 程序名
```

- 进程结构（后续补全，暂略）
- 进程相关概念
  - Page Frame：页框，给进程分配的内存最小单位

#### 查看page的大小

```
# getconf -a | grep -i size
```

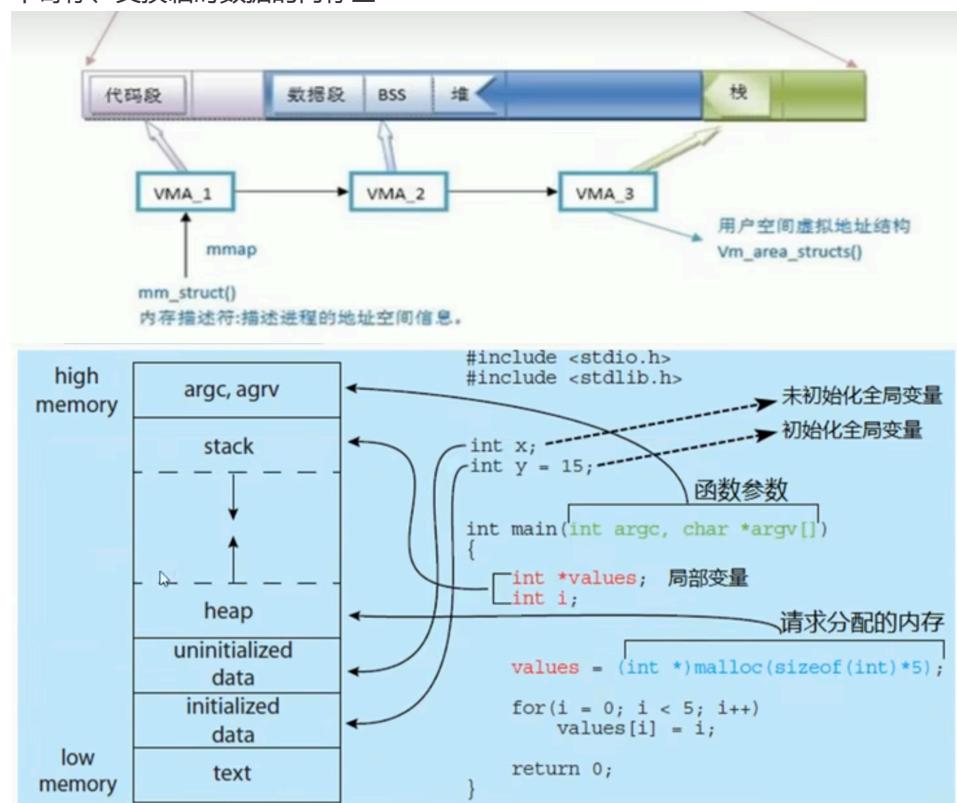
#### ○ 物理地址空间和虚拟地址空间

- MMU：Memory Management Unit 负责虚拟地址转换为物理地址（MMU是CPU的一个硬件组件）
- 程序在访问一个内存地址指向的内存时，CPU不是直接把这个地址送到内存总线上，而是送到MMU，然后把这个内存地址映射到实际物理内存地址上，然后通过总线再去访问内存，程序操作的地方称为虚拟内存地址
- TLB：Translation Lookaside Buffer 翻译后备缓冲器，用于保存虚拟地址和物理地址映射关系的缓存

#### ○ 用户和内核空间

- 例如：32位的操作系统，最大内存空间是4G，其中0-3G是用户空间（进程虚拟内存空间），3-4G是内核空间
- 内存的结构
  - VMA\_1：代码段
  - VMA\_2：数据段，BSS，堆
  - VMA\_3：栈
  - 每个进程都包括5种不同的数据段

- 代码段：用来存放可执行文件的操作指令，也就是说它是可执行程序在内存中的镜像。代码段需要防止在运行时被非法修改，所以只准许读取操作，而不允许写入(修改)操作--它不是可写的
- 数据段：用来存放可执行文件中已初始化全局变量，换句话说就是存放程序静态分配的变量和全局变量
- BSS段：包含了程序中未初始化的全局变量，在内存中bss段全部置零
- 堆(heap)：存放数组和对象，堆适用于存放进程运行中被动态分配的内存段，它的大小并不固定，可动态扩张或缩减。当进程调用malloc等函数分配内存时，新分配的内存就被动态添加到堆上；当利用free等函数释放内存时，被释放的内存从堆中被剔除
- 栈：栈是用户存放程序临时创建的局部变量，也就是说我们函数括弧{}中定义的变量如：函数体中local声明的变量。除此之外，在函数被调用时，其参数也会被压入发起调用的进程栈中，并且待到调用结束后，函数的返回值也会被存放在栈中。由于栈的后进先出的特点，所以特别方便用来保存/恢复调用现场。可以把堆栈看成一个寄存、交换临时数据的内存区



## 进程使用内存问题

### 内存泄漏：Memory Leak

- 指程序中用malloc或new申请了一块内存，但是没有用free或delete将内存释放，导致这块内存一直处于占用状态

### 内存溢出：Memory Overflow

- 指程序申请了10M的空间，但是在这个空间写入10M以上字节的数据，就是溢出

## 内存不足：OOM

- OOM即Out Of Memory, "内存用完了", 的情况在java程序中比较常见。系统会选出一个进程将之杀死，在日志messages中看到类似下面的提示

```
Jul 10 10:20:30 kernel: out of memory: kill process 9527(java) score 88 or sacrifice child
```

- 当JVM因为没有足够的内存来为对象分配空间并且垃圾回收器也已经没有空间可回收时，就会抛出这个error，因为这个问题已经严重到不足以被应用处理。
- 原因：
  - 给应用分配内存太少：比如虚拟机本身可使用的内存（一般通过启动时的VM参数指定）太少。
  - 应用用的太多，并且用完没释放，浪费了。此时就会造成内存泄漏或者内存溢出
- 使用的解决办法：
  - 限制java进程的max heap，并且降低java程序的worker数量，从而降低内存使用
  - 给系统增加swap空间
- 设置内核参数（不推荐），不允许内存申请过量：

```
echo 2 > /proc/sys/vm/overcommit_memory
echo 2 > /proc/sys/vm/overcommit_ratio
echo 2 > /proc/sys/vm/panic_on_oom
```

- 说明：
  - Linux默认是允许memory overcommit的，只要你来申请内存我就给你，寄希望于进程实际上用不到那么多内存，但万一用到那么多呢？Linux设计了一个**OOM killer机制挑选一个进程出来杀死，以腾出部分内存**，如果还不够就继续。也可**通过设置内核参数vm.panic\_on\_oom使得发生OOM时自动重启系统**。这都是有风险的机制，重启可能造成业务中断，杀死进程也有可能导致业务中断。所以Linux2.6以后允许通过内核参数vm.overcommit\_memory禁止memory\_overcommit.
- vm.panic\_on\_oom 决定系统出现oom的时候，要做的操作。接受的三种取值如下：

```
0 - 默认值，当出现oom的时候，触发oom killer
# 直接杀进程
1 - 程序在有cpuset, memory policy, memcg的约束情况下的OOM，可以考虑不panic，而是启动OOM killer。其它情况触发 kernel panic，即系统直接重启
# 根据情况杀进程或者重启
2 - 当出现oom，直接触发kernel panic，即系统直接重启
# 直接重启
```

- vm.overcommit\_memory接受三种取值：

0 - Heuristic overcommit handling. 这是缺省值，它允许overcommit，但过于冒失的overcommit会被拒绝，比如malloc一次性申请的内存大小就超过了系统总内存。Heuristic的意思是“试探式的”，内核利用某种算法猜测你的内存申请是否合理，它认为不合理就会拒绝overcommit。

# kernel设有一个阈值，申请的内存总数超过这个阈值就算overcommit，在/proc/meminfo中可以看到这个阈值的大小

1 - Always overcommit, 允许overcommit，对内存申请来者不拒。内核执行无内存过量使用处理。使用这个设置会增大内存超载的可能性，但是也可以增强大量使用内存任务的性能

2 - Don't overcommit, 禁止overcommit。内存拒绝等于或大于总可用swap大小以及overcommit\_ratio指定的物理RAM比例的内存请求。如果您希望减小内存过度使用的风险，这个设置就是最好的

- CommitLimit就是overcommit的阈值，申请的内存总数超过CommitLimit的话就算是overcommit。此值通过内核参数vm.overcommit\_ratio或vm.overcommit\_kbytes间接设置的，公式如下：

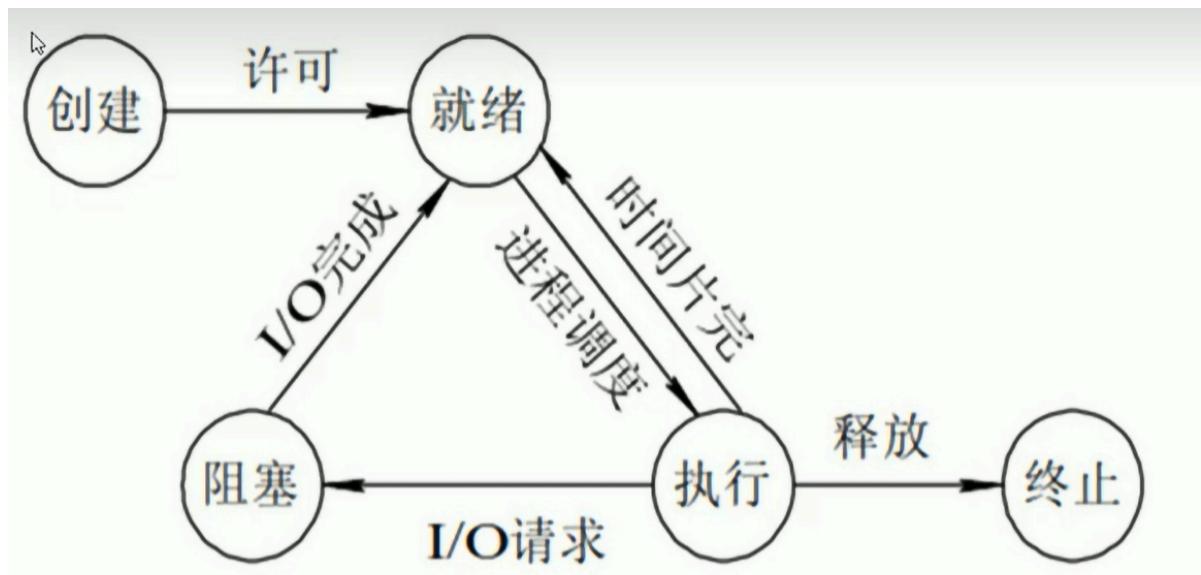
```
CommitLimit = (Physical RAM * vm.overcommit_ratio / 100) + Swap
```

- vm.overcommit\_ratio是内核参数，缺省值是50，表示物理内存的50%。如果你不想使用比率，也可以直接指定内存的字节数大小，通过另一个内核参数vm.overcommit\_kbytes即可；
- 如果使用了huge pages，那么需要从物理内存中减去，公式变成：

```
CommitLimit = ([total RAM] - [total huge TLB RAM]) * vm.overcommit_ratio / 100 + swap
```

## 进程的状态切换

### 进程的基本状态



- 创建状态：进程在创建时需要申请一个空白PCB (process control block)，向其中填写控制和管理进程的信息，完成资源分配。如果创建工作无法完成。比如资源无法满足，就无法被调度运行，把此时进程所处状态称为创建状态。
- 就绪状态：进程已经准备好，已分配到所需资源，只要分配到CPU就能够立即运行

- 执行状态：进程处于就绪状态被调度后，进程进入执行状态
- 阻塞状态：正在执行的进程由于某些事件（I/O请求，申请缓存区失败）而暂时无法运行，进程受到阻塞。在满足请求时进入就绪状态等待系统调用
- 终止状态：进程结束，或出现错误，或被系统终止，进入终止状态。无法再执行。

## 状态之间转换六种情况

- 运行--->就绪：
  - 主要是进程占用CPU的时间过长，而系统分配给该进程占用CPU的时间是有限的；
  - 在采用抢先式优先级调度算法的系统中，当有更高优先级的进程要进行时，该进程就被迫让出CPU，该进程便由执行状态变为就绪状态
- 就绪--->运行：
  - 运行的进程的时间片用完，调度就转到就绪队列中选择合适的进程分配CPU
- 运行--->阻塞：
  - 正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如发生了I/O请求
- 阻塞--->就绪：
  - 进程所等待的事件已经发生，就进入就绪队列
- 以下两种状态不可能发生：
  - 阻塞--->运行：即使阻塞进程分配CPU，也无法执行，**操作系统在进行调度时不会从阻塞队列进行挑选**，而是从就绪队列中选取
  - 就绪--->阻塞：就绪态根本就没有执行，谈不上进入阻塞态

## 进程更多状态：

- 运行态：running
- 就绪态：ready
- 睡眠态：分为两种，可中断：interruptable，不可中断：uninterruptable
- 停止态：stopped，暂停于内存，但不会被调度，除非手动启动
- 僵死态：zombie，结束进程，父进程结束前，子进程不关闭
  - 既不占有CPU资源，也仅占用极少的内存
  - 进程列表中存在，重启时会从进程列表中清除
  - 死的进程，无法再次杀死
  - 实现

```

echo $BASHPID
# 1436
bash
echo $BASHPID
# 1809
echo $PPID
# 1436

kill -19 1436 # 将父进程变为停止态 stat为T
kill -15 1809 # 给子进程发送后一个15信号，15信号为正常结束关闭进程
# 关闭进程后，回收该进程所有资源

```

```
# 正常关闭进程后，该进程应该在进程列表中清除，但是由于父进程为停止态，无法回收子进程的尸体，  
因此子进程变为了僵尸态
```

```
kill -18 1436 # 激活父进程  
# 由于父进程被激活，因此子进程的被回收，子进程从进程列表清除  
  
# 强杀父进程的时候，可以将子进程也杀死
```

**ps aux** 中的**stat**可以查看状态

STAT: 进程状态

R:	running
S:	interruptable sleeping (大部分进程处于睡眠态)
D:	uninterruptable sleeping
T:	stopped
Z:	zombie
+	前台进程
l:	多线程进程
L:	内存分页并带锁
N:	低优先级进程
<:	高优先级进程
s:	session leader, 会话发起者
I:	idle kernel thread, CentOS 8 新特性

## LRU算法

- LRU: Least Recently Used 近期最少使用算法，释放内存
- 计算机组成中详解

## 进程间通信

### IPC:Inter Process Communication

- 同一主机

pipe	# 管道
socket	# 套接字文件
Memory-mapped file	# 文件映射，将文件中的一段数据映射到物理内存，多个进程共享这片内存
signal	# 信号
Lock	# 对资源上锁，如果资源已被某进程锁住，则其他进程想修改甚至读取这些资源，都将被阻塞，直到锁被打开
semaphore	# 信号量，一种计数器

- 不同主机: socket=IP和端口号

RPC remote procedure call(远程过程调用)  
MQ 消息队列，生产者和消费者，如: Kafka, RabbitMQ, ActiveMQ

- 创建管道文件(单工，单向传输，只能一对一)

```

mkfifo /data/test fifo
cat > /data/test fifo
# 在另一个终端
cat /data/test fifo

```

## 进程优先级

- 优先级范围 (0-139)
  - (0 - 99) 实时进程, 内核中操作系统相关进程使用
  - (100 - 139) 非实时进程, 用户进程使用
- CentOS 优先级



- 进程优先级
  - realtime优先级: 99 - 0, 值最大, 优先级最高
  - nice值: -20到19, 对应系统优先级100 - 139
- 进程优先级的执行过程 (非Linux)

0 - 139个优先级, 有140个优先级队列, 数字越小, 优先级越高

假设当前只有100, 101, 102, 3个队列中有进程排队

100:p1, p2, p3

101:p4, p5

102:p6, p7

由于p1在100优先级队列, 优先级高, 因此, cpu先给p1分配时间片, 优先执行p1,

如果在时间片消耗之后, p1仍未执行完毕, 则剩余工作放入100优先级的就绪队列

然后处于100优先级的运行队列中的p2被分配CPU时间片, 运行, 假设p2运行完毕

100优先级队列中的p3被分配时间片, 运行, 此时100优先级队列中的进程数为空

假设p3在时间片消耗完, 也没有执行完毕, 剩余进程进入100优先级的就绪队列, 此时就绪队列有p1和p3,

由于100优先级的运行队列中的进程为空, 因此, 100优先级的就绪队列变为运行队列, 再执行p1, p3

如此反复，当100优先级队列中的p1,p3彻底执行完毕后，运行优先级101的队列

- Linux内核使用的是完全公平调度器（CFS），其目标是确保CPU时间在所有进程间公平分享，而不是严格基于优先级调度。

Linux内核中的完全公平调度器（Completely Fair Scheduler, CFS）是自Linux 2.6.23版本开始引入的默认CPU调度器，它基于公平调度算法设计，旨在为运行在系统上的每个进程提供尽可能公平的CPU时间分配。CFS的核心思想是基于红黑树（一种自平衡二叉查找树）来动态管理和调度进程，从而实现公平性和高效性。下面是CFS实现的一些详细说明，以及nice值（优先级）是如何影响进程调度的。

#### CFS的工作原理

红黑树：CFS使用一棵红黑树来维护所有可运行的进程（即处于就绪状态，等待被调度到CPU上执行的进程）。红黑树的每个节点代表一个进程，按照进程的虚拟运行时间（vruntime）排序。vruntime是一个进程获得CPU时间的衡量，意图是反映每个进程使用CPU资源的量。

选择下一个进程：当需要选择下一个要运行的进程时，CFS会选择红黑树最左侧的节点，即vruntime最小的进程，因为这代表了它相对于其他进程获得的CPU时间最少。

时间片：CFS不固定分配时间片大小，而是根据系统负载和进程数动态调整。理论上，时间片的长度是与系统中就绪进程数量的倒数成比例的，意味着更多的就绪进程会导致更短的时间片，从而每个进程能更频繁地被调度。

睡眠和唤醒：为了保证公平性，当进程从睡眠状态唤醒时，CFS会给予一定的时间补偿，确保长时间睡眠的进程在唤醒后能较快地获得CPU时间。

#### nice值的影响

在Linux中，nice值是一个介于-20到19的整数，用于调整进程的优先级。默认值为0，较低的nice值（负值）表示较高的优先级，而较高的nice值（正值）表示较低的优先级。

在CFS中，nice值通过调整进程的vruntime来间接影响进程的调度优先级：

负nice值（优先级高）：进程的vruntime增加得更慢，使得该进程更容易被调度。

正nice值（优先级低）：进程的vruntime增加得更快，导致该进程相对较难获得CPU时间。

这样，尽管CFS的目标是确保所有进程公平地分享CPU时间，nice值仍然允许系统管理员或用户调整特定进程的调度偏好，以反映出更高或更低的优先级需求。

## 进程分类

- 守护进程：守护进程：daemon，在系统引导过程中启动的进程，和终端无关进程
- 前台进程：跟终端相关，通过终端启动的进程
- 按进程资源的使用分类
  - CPU-Bound：CPU密集型，非交互
  - IO-Bound：IO密集型，交互

## I/O调度算法

- 操作系统版本不同，I/O调度算法不同

```
cat /sys/block/sda/queue/scheduler
```

# 进程管理和性能相关工具

## 进程树 pstree

- `pstree`

```
pstree [option] [PID | USER]
```

- 常用选项：

```
-p      # 显示PID  
-u      # 显示用户切换  
-H pid # 高亮指定进程及其前辈进程  
-h      # 高亮显示当前进程及其前辈进程
```

## 进程信息 ps

- `ps` 即process state，可以显示进程当前状态的快照，默认显示当前终端中的进程，Linux系统各进程的相关信息均保存在`/proc/PID`目录下的各文件中

- `ps`格式

```
ps [OPTION...]
```

- 支持三种选项

- UNIX选项：如：`-A`, `-e`
- BSD选项：如：`a`
- GNU选项：如：`--help`

- 常用选项

```
a          # 选项包括所有终端中的进程  
x          # 选项包括不链接终端的进程  
u          # 选项显示进程所有者信息  
f          # 选项显示进程树，相当于--forest  
k|--sort 属性 # 对属性排序，属性前加-表示倒序  
o 属性...   # 显示定制信息：pid, cmd, %cpu, %mem...  
-U         # 显示指定RUID或用户的进程  
-u         # 显示指定EUID或用户的进程  
-f         # 显示完整格式程序信息  
-c         # 指定命令，多个命令用逗号分隔
```

- 常用选项组合

```
ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.1	191184	3836	?	Ss	Mar14	2:36	/usr/lib/systemd/system --switched-root --system --deserialize 22
root	2	0.0	0.0	0	0	?	S	Mar14	0:00	[kthreadd]
root	4	0.0	0.0	0	0	?	S<	Mar14	0:00	[kworker/0:0H]
root	6	0.0	0.0	0	0	?	S	Mar14	0:15	[ksoftirqd/0]

root	7	0.0	0.0	0	0	?	S	Mar14	0:08	[migration/0]
root	8	0.0	0.0	0	0	?	S	Mar14	0:00	[rcu_bh]
root	9	0.0	0.0	0	0	?	S	Mar14	6:41	[rcu_sched]
root	10	0.0	0.0	0	0	?	S<	Mar14	0:00	[lru-add-drain]
root	11	0.0	0.0	0	0	?	S	Mar14	0:05	[watchdog/0]
root	12	0.0	0.0	0	0	?	S	Mar14	0:04	[watchdog/1]
root	13	0.0	0.0	0	0	?	S	Mar14	0:08	[migration/1]
root	14	0.0	0.0	0	0	?	S	Mar14	0:14	[ksoftirqd/1]
root	16	0.0	0.0	0	0	?	S<	Mar14	0:00	[kworker/1:0H]
root	18	0.0	0.0	0	0	?	S	Mar14	0:00	[kdevtmpfs]

# USER: 进程发起者  
# PID: 进程PID  
# %CPU: 占用CPU百分比  
# %MEM: 占用内存百分比  
# VSZ: 操作系统承诺的虚拟内存数量  
# RSS: 实际使用的内存大小  
# TTY: 终端  
# STAT: 状态信息  
# START: 什么时间启动的  
# TIME: CPU时间片累加值  
# COMMAND: 命令

# 排序显示命令  
**ps axo pid,cmd,%cpu,%mem k -%cpu**

字段	说明
USER	进程属主
PID	进程ID
PPID	父进程ID
%CPU	CPU占用率
%MEM	内存占用率
VSZ	Virtual memory SiZe, 虚拟内存集, 线性内存, 虚似内存
RSS	ReSident Size, 常驻内存集, 实际占用物理内存
TTY	终端
STAT	进程状态
START	进程开始时间
TIME	累计分配给进程的cpu时长
COMMAND	对应的程序及参数
UID	进程属主
C	cpu利用率, 取整
STIME	进程开始时间
CMD	对应的程序及参数
ni	nice值
pri	priority 优先级
rtprio	实时优先级
psr	processor CPU编号

- 面试题：找到未知进程的执行程序文件路径

```
ls -l /proc/1272/exe
# 通过查看proc/PID/exe的软链接指向
```

## 查看进程信息prtstat

- 可以显示进程信息，来自于psmisc包
- 格式：

```
prtstat [options] PID ...
```

- 常用选项

```
-r # 显示格式更容易观看
```

## 设置和调整进程优先级

- nice和renice只能调整非实时优先级
- nice，翻译为友善度，这个术语源自于它决定了你对待系统的其他用户的友善度。友善度越高，越谦虚，优先级越低。友善度越低，说明你已经不打算让步了，则优先级越高
- 详解实时优先级和非实时优先级的区别

### 非实时优先级

基于时间共享：非实时进程是基于时间共享（**time-sharing**）策略调度的，意味着这些进程按照一定的公平原则轮流使用CPU资源。操作系统调度器会尽量平等地分配CPU时间给每个进程，但也允许优先级调整以改变进程获取CPU时间的频率。

**nice值：**在Linux中，非实时进程的优先级可以通过**nice**值来调整。**nice**值的范围通常是-20（最高优先级）到19（最低优先级）。默认情况下，进程的**nice**值为0。通过调整**nice**值，用户和系统管理员可以影响进程的调度优先级。

**适用场景：**适用于大多数常规应用，如用户程序和系统后台服务。这些进程不需要严格的时间限制，可以接受在CPU调度上的延迟。

### 实时优先级

基于实时调度：实时进程根据实时调度策略运行，这意味着它们被赋予了更高的执行优先级，并且在被调度时，它们能够快速响应。实时进程旨在最小化响应时间，确保在指定的时间限制内完成任务。

**调度策略：**Linux支持多种实时调度策略，如**SCHED\_FIFO**（先入先出）、**SCHED\_RR**（轮转轮询）和**SCHED\_DEADLINE**（截止时间调度）。**SCHED\_FIFO**和**SCHED\_RR**进程有一个与之相关的实时优先级，范围通常是1到99，其中99代表最高优先级。

**实时优先级的设置：**实时优先级通常由系统管理员设置，需要特定的权限。这是为了防止实时进程过多地占用CPU时间，影响系统的稳定性。

**适用场景：**适用于需要快速确定性响应的应用，如音视频处理、工业控制和其他需要严格时间控制的应用。这些进程通常运行在高优先级，以确保它们能够及时完成任务。

### 区别总结

**调度策略：**实时优先级进程基于实时调度策略，而非实时优先级进程基于时间共享策略。

**优先级范围：**实时进程优先级通常高于非实时进程，且优先级范围不同。

**响应时间：**实时进程设计以保证最小响应时间，非实时进程则更侧重于公平性和资源共享。

**使用场景：**实时进程用于对响应时间有严格要求的场景，而非实时进程用于一般的计算任务。

正确配置和使用实时和非实时进程优先级是高效系统管理的关键部分，需要根据具体的应用场景和需求来调整。

- nice使用格式

```
nice -n -10 ping 127.0.0.1
# 将ping命令以-10的友善度执行进程

# -n 后面接具体数值（-20~19），指定命令的友善度
```

- 更改nice值

```
renice 命令

# 可以调整正在执行中的进程的优先级
renice -n -20 PID
```

- pri与rt prio和nice的关系与对比

- pri的范围是139 - 0，数值越高，优先级越高
- rt prio(实时优先级 (系统优先级)) 范围是99 - 0

- rtpri的99对应pri的139，数值越高，优先级越高
- nice的范围是-20到19
  - 对应pri的40 - 0， nice值越低，优先级越高

## 实现进程绑定指定CPU—— taskset

- taskset
- 格式：

```
taskset [options] mask command [argument...]
taskset [options] -p [mask] pid
```

- mask是CPU亲和性掩码，用于指定进程可以运行的CPU核心。掩码是一个十六进制数，每一位代表一个CPU核心，最低位代表CPU0。位值为1表示进程可以在该CPU上运行，为0表示不可以。
- command [argument...]是要启动的新进程及其参数。
- -p选项用于操作已经运行的进程，后面跟着的是进程ID（PID）。
- 示例：

```
# 假设有一个名为myapp的应用程序，你希望它只在第二个CPU（CPU1）上运行：
taskset 0x2 myapp
```

```
# 如果myapp已经在运行，其PID为1234，将其迁移到CPU1上：
taskset -p 0x2 1234
```

```
# 查询myapp的CPU亲和性设置
taskset -p 1234
```

- 详解cpu亲和性掩码

```
# 示例：pid 1231725's current affinity mask: 3
```

在**taskset**使用的亲和性掩码中，每一位二进制数代表一个CPU核心，从右到左分别代表CPU0、CPU1、CPU2等。亲和性掩码是一个十六进制数，转换为二进制后，每个1表示进程可以在对应的CPU上运行，0表示不可以。

亲和性掩码"3"转换为二进制是"11"。这意味着：

最右边的位（第一位，值为1）表示进程可以在CPU0上运行。

紧接着的第二位（值也为1）表示进程可以在CPU1上运行。

因此，亲和性掩码"3"表示该进程可以在CPU0和CPU1上运行，而不是单独在CPU2上。如果要设置进程仅在CPU2上运行，亲和性掩码应该是"4"（二进制为"100"），这样第三位为1，表示进程可以在CPU2上运行。

## 搜索进程 —— pgrep

- 按条件搜索进程
  - ps选项 | grep 'pattern' 灵活
  - pgrep 按预定义的模式（过滤）
  - /sbin/pidof 按确切的程序名称查看pid
- pgrep命令格式

```
pgrep [option] pattern
```

- 常用选项

```
-u uid: effective user, 生效者  
-U uid: real user, 真正发起运行命令者  
-t terminal: 与指定终端相关的进程  
-l: 显示进程名  
-a: 显示完整格式的进程名  
-P pid: 显示指定进程的子进程
```

- pgrep示例

```
pgrep -lu wang  
# 注意l放在u的前面, u后面接wang

pgrep -au wang

# 显示指定进程的子进程
pgrep -aP 2303

# 显示指定终端的进程
pgrep -at pts/2
```

## 查看进程编号 (PID)

- pidof
- 格式:

```
pidof <command>

# 示例:
pidof ping # 1987

# 查看脚本文件
pidof -x <脚本名称>

# 示例
pidof -x ping.sh
```

## 负载查询 uptime

- /proc/uptime 包括两个值, 单位s
  - 系统启动时长
  - 空闲进程的总时长 (按总的CPU核数计算)

```
cat /proc/uptime
# 13073.55 21437.56

# 系统空闲时间计算 21437.56 / 13073.5 * 2
```

- uptime和w显示以下内容
  - 当前时间
  - 系统已启动的时间
  - 当前上线人数
  - 系统平均负载 (1、5、15分钟的平均负载，一般不会超过1，超过5时建议警报)
- 系统平均负载：指在特定时间间隔内运行队列中的平均进程数，通常每个CPU内核的当前活动进程数不大于3，那么系统性能良好，如果每个CPU内核的任务数大于5，那么此主机的性能由严重问题
- uptime的显示和w的第一行一样

## 显示CPU相关统计 mpstat

- 来自于sysstat包
- 范例：

```
[root@localhost ~]# mpstat
Linux 4.18.0-513.5.1.el8_9.x86_64 (localhost.localdomain)        04/08/2024
_x86_64_          (2 CPU)

07:13:31 PM   CPU   %usr   %nice   %sys %iowait   %irq   %soft   %steal   %guest
%gnice   %idle
07:13:31 PM   all    5.68     0.06    7.18     0.31    1.05    0.34     0.00     0.00
      0.00   85.38

# 表示一秒执行一次
mpstat 1

# 表示一秒钟执行一次，收集6次退出
mpstat 1 6
```

- 各字段含义

**CPU:** 显示处理器编号。all 表示所有CPU的平均值。

**%usr:** 显示执行用户空间进程的时间百分比。用户空间进程是指那些不需要内核模式特权的进程。

**%nice:** 显示执行优先级较低的用户进程（被"nice"命令调整过优先级）的CPU时间百分比。

**%sys:** 显示在系统（内核）空间执行进程的时间百分比。系统空间进程是指那些需要内核模式特权的进程。

**%iowait:** 显示CPU等待输入输出操作完成时间的百分比。高的*%iowait*值表示磁盘IO可能是性能瓶颈。

**%irq:** 显示处理硬件中断请求时间的百分比。

**%soft:** 显示处理软件中断时间的百分比。软件中断通常由系统内部事件触发，而不是硬件中断。

**%steal:** 在虚拟化环境中，显示等待虚拟CPU的时间百分比，因为其他虚拟机占用了物理CPU时间。

**%guest:** 显示运行虚拟处理器的时间百分比。

**%gnice:** 显示运行带有nice优先级的虚拟处理器的时间百分比。

%idle: 显示CPU空闲时间的百分比，不包括等待I/O操作的时间。

## 查看进程实时状态 top (常用)

```
top - 19:25:04 up 16 min,  1 user,  load average: 0.00, 0.02, 0.04
Tasks: 151 total,   1 running, 150 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.3 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.3 hi,  0.2 si,  0.0 st
MiB Mem : 1734.3 total, 1145.5 free,   237.6 used,   351.2 buff/cache
MiB Swap: 2056.0 total, 2056.0 free,      0.0 used. 1340.4 avail Mem

PID USER      PR  NI      VIRT      RES      SHR S %CPU %MEM     TIME+ COMMAND
923 root      20   0  352920  10804    9188 S  0.3  0.6  0:03.37 vmtoolsd
1016 root      20   0  483004  31488   15372 S  0.3  1.8  0:05.64 tuned
1769 root      20   0  126468   5508    4248 S  0.3  0.3  0:00.20 sshd
2228 root      20   0   54364   4256    3556 R  0.3  0.2  0:00.06 top
1 root       20   0  175432  13900    8512 S  0.0  0.8  0:07.06 systemd
2 root       20   0      0      0      0 S  0.0  0.0  0:00.05 kthreadd
3 root      0 -20      0      0      0 I  0.0  0.0  0:00.00 rcu_gp
4 root      0 -20      0      0      0 I  0.0  0.0  0:00.00 rcu_par_gp
5 root      0 -20      0      0      0 I  0.0  0.0  0:00.00
slub_flushwq
7 root      0 -20      0      0      0 I  0.0  0.0  0:00.00
kworker/0:0H-events_highpri
```

```

      10 root      0 -20      0      0      0 I    0.0   0.0   0:00.00
mm_percpu_wq

      11 root      20  0      0      0      0 S    0.0   0.0   0:00.00
rcu_tasks_rude_

      12 root      20  0      0      0      0 S    0.0   0.0   0:00.00
rcu_tasks_trace

      13 root      20  0      0      0      0 S    0.0   0.0   0:00.08
ksoftirqd/0

```

- 信息详解
  - 第一行: uptime的命令结果, 显示负载情况
  - 第二行: 显示进程的统计结果
  - 第三行: CPU相关数据统计
  - 第四行: 内存相关数据统计
  - 第五行: Swap相关数据统计
- 快捷键:
  - **M** 是按内存利用率排序
  - **P** 是按CPU利用率排序
  - **T** 是按累计时间片Time+排序
- 首部信息显示
  - **1** 显示隐藏uptime信息
  - **t** 改变和显示隐藏tasks及cpu信息
  - **1** cpu分别显示
  - **m** memory信息
  - **q** 退出命令
  - **k** 终止指定进程
  - **s** 修改刷新时间间隔
  - **w** 保存文件

## 页面炫酷版top ---htop

- Ubuntu中可以直接下载使用

## 内存空间free

```
free -h
```

## 进程对应的内存映射pmap

- 格式

```
pmap PID  
  
# 显示进程中的内存映射  
# 相当于cat /proc/PID/maps
```

## 查看程序运行时的系统调用 strace

- 示例

```
strace ls
```

## 显示程序运行时的库调用 (C语言库) ltrace

```
ltrace ls
```

## 显示虚拟内存信息 vmstat

```
root@ubuntu2204:/proc/1025$vmstat  
procs      -----memory-----  --swap--  -----io---- -system--  -----cpu-----  
r b    swpd    free    buff    cache    si    so    bi    bo    in    cs us sy id wa st  
0 0      0 2853204 27964 687760    0    0    15     9    62   119  0  0 99  0  0
```

- 这里的si, so, bi, bo, 都是以内存为参照物
  - 比如：从硬盘中读取数据到内存，此时是内存进，硬盘出，所以是bi增长
- system字段
  - in: interrupts 中断速率，包括时钟
  - cs: context switch 进程切换速率
- 示例：

```
# 内存信息汇总  
vmstat -s  
  
# 2秒钟显示1次，显示5次  
vmstat 2 5
```

## 统计CPU和设备IO信息iostat

- 此工具由sysstat包提供
- 范例：

```
root@ubuntu2204:/proc/1025$iostat  
Linux 5.15.0-101-generic (ubuntu2204.mystical.org)        04/08/2024        _x86_64_  
(2 CPU)  
  
avg-cpu: %user    %nice %system %iowait  %steal    %idle  
          0.31     0.10     0.48     0.02     0.00    99.09
```

Device		tps	kB_read/s	kB_wrtn/s	kB_dscd/s	kB_read
kb_wrtn	kb_dscd					
dm-0		1.13	24.58	20.33	0.00	515289
426148	0					
dm-1		0.03	0.65	0.00	0.00	13684
0	0					
loop0		0.01	0.11	0.00	0.00	2212
0	0					
loop1		0.00	0.05	0.00	0.00	1056
0	0					
loop2		0.04	1.52	0.00	0.00	31827
0	0					
loop3		0.00	0.02	0.00	0.00	344
0	0					
loop4		0.00	0.02	0.00	0.00	345
0	0					
loop5		0.00	0.05	0.00	0.00	1099
0	0					
loop6		0.00	0.00	0.00	0.00	10
0	0					
sda		0.77	25.10	20.33	0.00	526190
426288	0					
sdb		0.06	1.32	0.10	0.00	27634
2052	0					
sdc		0.03	0.37	0.00	0.00	7816
0	0					
sr0		0.02	0.60	0.00	0.00	12554
0	0					

```
# 2秒钟显示1次，显示5次
iostat 2 5
```

## 系统资源统计 dstat

- dstat用于替代vmstat, iostat
- 由pcp-system-tools包提供

## 监视磁盘I/O iotop

- 可以看出具体导致I/O异常的进程或命令

## 显示网络带宽使用情况 iftop

- 常用选项

-n	# 以ip形式显示主机
-F	# 仅显示ipv4流量
-P	# 显示流量端口号

## 查看网络实时吞吐量nload

```
# 使用方向键切换网卡
```

## 远程综合监控glances

- 可以在一台设备上，远程监控另一台设备的情况
- 示例

```
# 在两台机器上同时安装glances  
# 其中需要监控的设备设置为服务端 glances -s  
# 另一个监控设备为客户端 glances -c <服务端ip>
```

## 查看进程打开文件 lsof

```
lsof [option]  
  
# 常用选项  
-i          # 后面接 ":端口号"，查看监听该端口的进程  
-p          # 列出指定进程打开的文件  
-c cmd      # 列出指定进程打开的文件
```

- 范例：

```
#查看当前哪个进程正在使用此文件  
[root@ubuntu ~]# lsof /var/log/messages  
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF       NODE NAME  
rsyslogd 1279 root    5w   REG  253,0  1882516 26487520 /var/log/messages  
  
#查看指定终端启动的进程  
[root@ubuntu ~]# lsof /dev/pts/1
```

## 信号发送 kill

- kill：内部命令，可用来向进程发送控制信号，以实现对进程管理，每个信号对应一个数字，信号名称以SIG开头，不区分大小写
- 显示当前系统可用信号

```
kill -1  
trap -1
```

- 常用信号

```
1. SIGHUP # 无须关闭进程而让其重读配置文件  
2. SIGINT # 中止正在运行的进程，相当于Ctrl+C  
3. SIGQUIT # 相当于Ctrl+\  
9. SIGKILL # 强制杀死正在运行的进程  
15. SIGTERM # 终止正在运行的进程（kill命令默认信号）  
18. SIGCONT # 继续运行（可激活停止态进程）（或者让前台进程进入后台运行）  
19. SIGSTOP # 后台休眠（使进程强制进入T停止态）无法被忽略  
20. SIGTSTP # 相当于Ctrl+Z，一个可以被进程捕获和忽略的停止信号
```

- 对指定进程发送信号

```
kill -1
```

## killall

- 来自于psmisc

```
killall [信号] 命令名称
```

# 特殊信号 0

该信号不会真的发送一个信号，但是会进行错误检查，检查进程的健康性  
观察echo \$? 如果是0，说明没问题

# kill -0 不能检查出僵尸进程，对于僵尸进程的检测，\$?的值也为0

## 能够多个命令联动使用

```
lsof (查看进程) ---> pstree(查看父进程) ---> kill 信号控制进程
```

## 作业管理

### Linux的作业控制

- 前台作业：通过终端启动，且启动后一直占据终端
- 后台作业：可通过终端启动，但启动后即转入后台运行（释放终端）
- 让作业运行于后台
  - 运行中的作业：Ctrl+Z（进入后台，并进入停止态）
  - 尚未启动的作业：COMMAND &
- 使用kill信号，让前台进程进入后台并保持运行态

```
kill -18 %1
```

# 百分号后面是作业编号，可通过jobs查看

- fg

- 把后台指令恢复到前台运行

```
fg <作业编号>
```

- 关闭终端保证进程不死的两种方法

- nohup
  - 会将标准输出输入到当前目录的一个文件中hup.out
  - 可以通过nohup &> /dev/null 解决
- 会话管理
  - screen
  - tmux

## 并行运行

- 方法1

```
cat all.sh
f1.sh&
f2.sh&
f3.sh&
wait
```

- 方法2

```
(ping 127.1&);(ping 127.2&);(ping 127.3&)
```

- 方法3

```
ping 127.1& ping 127.2& ping 127.3&
```

- 方法4

```
# 多组命令实现并行访问
{ ping -c3 127.1; ping 127.2; }& ;{ ping -c3 127.3; ping 127.4; }&
```

## 计划任务

### 一次性任务

#### at工具

- 指定时间点，执行一次性任务
- at 工具
  - 由包at提供
  - 依赖与atd服务，需要启动才能实现at任务
  - at队列存放在/var/spool/at目录中
- 使用前需要确认atd.service处于running状态

```
systemctl status atd.service
```

- at命令

```
at [option] TIME      # ctrl + D 结束并完成任务设置

# 常用选项
-1          # 列出未执行的一次性任务，等价于atq命令
-c <编号>   # 查看指定编号的计划任务的具体内容
-d <编号>   # 删除指定编号的计划任务，等价于atrm + 编号
-f file     # at -f a.txt 15:20 等同于at 15:20 < a.txt
-m          # 当任务完成后，即使没有标准输出，也会给用户发邮件

# 非交互方式实现计划任务
echo hello | at 14:30

echo wall hello | at 14:30 # wall 命令是广播，会在所有终端屏幕上出现
```

- at 时间格式

```
HH:MM           # 若时刻以过，则明天的此时执行任务

HH:MM  YYYY-MM-DD # 规定某年某月 某天的特殊时刻执行该任务，不支持到秒，最小到分钟

now+#{minutes, hours, days, OR weeks}
```

- 注意：

- 作业执行命令的结果中的标准输出和错误以执行任务的用户的身份，发邮件通知给root
- 默认CentOS8最小化安装没有安装邮件服务，需自行安装

```
dnf install postfix -y
systemctl enable --now postfix

# 安装mail
sudo apt install mailutils
```

- 创建的at任务，文件在 /var/spool/at 目录下
- 执行任务时，PATH变量和当前定义任务的用户身份一致
- /etc/at.{allow, deny} 控制用户是否能执行at任务
  - 白名单：/etc/at.allow，默认不存在，只有该文件中的用户才能执行at命令
  - 黑名单：默认存在，拒绝文件中的用户执行at命令
  - 如果两个文件都不存在，则只有root能够执行at命令
  - 白名单的优先级高于黑名单

## batch

- 系统自行选择空闲时间去执行此处指定的任务

# 周期性计划任务cron

- 周期性计划任务cron相关的程序包
  - cronie: 主程序包, 提供crond守护进程及相关辅助工具
  - crontabs: 包含CentOS提供系统维护任务
  - cronie-anacron: cronie的补充程序, 用于监控cronie的任务执行状态, 如: cronie中的任务在过去该运行的时间点未能正常运行, 则anacron会随后启动一次任务
- cron依赖于crond服务, 确保crond守护处于运行状态

## CentOS中的cron

- 有/etc/cron.deny, 可以给cron设置黑名单 (Ubuntu默认没有该文件)
- run-parts
  - 立即执行目录中所有脚本, 要求目录内脚本有执行权限 (Ubuntu默认没有该文件)
- cron任务分为
  - 系统cron任务: 系统维护作业, /etc/crontab (总配置文件) /etc/cron.d/目录下, 自行创建(子配置文件)
  - 用户cron任务: 保存在 /var/spool/cron/USERNAME, 利用crontab命令管理
- 计划任务日志: /var/log/cron
- cron配置文件编辑

```
# Example of job definition:  
# .----- minute (0 - 59)  
# | .----- hour (0 - 23)  
# | | .----- day of month (1 - 31)  
# | | | .---- month (1 - 12) OR jan,feb,mar,apr ...  
# | | | | .--- day of week (0 - 6) (Sunday=0 or 7) OR  
sun,mon,tue,wed,thu,fri,sat  
# | | | | |  
# * * * * * user-name command to be executed  
17 * * * * root    cd / && run-parts --report /etc/cron.hourly  
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report  
/etc/cron.daily )  
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report  
/etc/cron.weekly )  
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report  
/etc/cron.monthly )  
  
# 支持时间格式  
1. 逗号分隔  
2. 范围, eg: 1-10  
3. 频率, eg: /5
```

## 普通用户创建计划任务

- 使用crontab命令

```
crontab -e 创建计划任务
```

```
# 常用选项
-1          # 列出所有任务
-e          # 编辑任务
-r          # 移除所有任务
-i          # 同-r一同使用，已交互模式移除指定任务
-u user    # 仅root可运行，指定用户管理cron任务
```

- 练习作业

- 每周的工作日1:30,将/etc/备份至/backup目录中，保存的文件名格式为"etcbak-yyyy-mm-dd-HH.tar.xz",其中日期是前一天的时间
- 每两小时取出当前系统/proc/meminfo文件中以S或M开头的信息追加至/tmp/meminfo.txt文件中
- 工作日时间，每10分钟执行一次磁盘空间检查，一旦发现任何分区利用率高于80%，就执行wall警报

## wall命令

- 作用：wall (write all) 是一个在Linux和Unix系统中用于向所有当前登录用户的终端发送消息的命令。这个命令对系统管理员特别有用，比如在重启服务器之前，管理员可以使用wall发送一条广播消息给所有用户，通知他们即将发生的重启。

```
echo message | wall
wall message
wall < filename
```

## Systemd定时器

在现代Linux系统中，`systemd timers` 来管理和执行定时任务，取代了传统的 `cron` 作业。`systemd` 定时器提供了更加灵活和强大的功能，能够更好地与 `systemd` 的服务管理机制集成

### 基本概念

`systemd` 定时器(`systemd timer`)是`systemd`中的一种单元类型，类似于服务单元(`service unit`)。定时器单元通常有两个文件

- 定时器单元文件(`.timer`)：定义定时规则(类似于 `cron` 表达式)
- 服务单元文件(`.service`)：定义由定时器触发的实际任务

## 创建和使用 `systemd` 定时器

示例：创建一个简单的定时任务

1. 创建服务单元文件

```
# 定义实际要执行的任务
vim /etc/systemd/system/mytask.service

# 内容如下
[Unit]
Description=My custom task          # Description:描述这个服务的用途

[Service]
ExecStart=/usr/bin/my-script.sh    # ExecStart:指定要执行的命令或脚本
```

## 2. 创建定时器单元文件

```
# 定义了任务的执行时间
vim /etc/systemd/system/mytask.timer

[Unit]
Description=Run mytask.service daily

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timer.target
```

## OnCalendar基本语法

```
OnCalendar=<时间表达式>
```

### 1. 时间表达式基本格式

```
DayOfweek Year-Month-Day Hour:Minute:Second
```

- DayOfweek (周几) :可选，表示星期几(如: "Mon"、"Tue")，可以是全称或缩写。可以使用通配符 \*  
• Year-Month-Day (年-月-日) : 年、月、日均可使用通配符 \*  
• Hour:Minute:Second (时:分:秒) : 小时、分钟、秒，均可使用通配符 \*

### 2. 常见的时间表达式

- 每天定时任务

```
# 每天午夜
OnCalendar=daily
OnCalendar=*-*-* 00:00:00      # 和daily等价

# 每天的特定时间(如每天上午6:30)
OnCalendar=*-*-* 06:30:00
```

- 每周定时任务

```
# 每周一凌晨  
OnCalendar=weekly  
OnCalendar=Mon *-*-* 00:00:00 # 和weekly等价
```

```
# 每周一上午7:00  
OnCalendar=Mon *-*-* 07:00:00
```

- 每月定时任务

```
# 每月的1号午夜  
OnCalendar=monthly  
OnCalendar=*-*-*01 00:00:00
```

```
# 每月的15号中午12:00  
OnCalendar=*-*-*15 12:0:0
```

- 每年定时任务

```
# 每年1月1日午夜  
OnCalendar=yearly  
OnCalendar=*-01-01 00:00:00  
  
# 每年7月4日上午 10::00  
OnCalendar=*-07-04 10:00:00
```

- 每小时定时任务

```
# 每小时的第0分  
OnCalendar=hourly  
OnCalendar=*-*-* *:00:00  
  
# 每小时的第15分钟  
OnCalendar=*-*-* *:15:00
```

### 3. 特殊事件格式

`systemd` 定义了一些特殊的时间格式，简化常用的定时任务设置

- `minutely`：每分钟执行一次，相当于 `*-*-* *::*:00`
- `hourly`：每小时执行一次，相当于 `*-*-* *:00:00`
- `daily`：每天执行一次，相当于 `--* 00:00:00`。
- `weekly`：每周执行一次，相当于 `Mon --* 00:00:00`。
- `monthly`：每月执行一次，相当于 `--01 00:00:00`。
- `yearly` 或 `annually`：每年执行一次，相当于 `*-01-01 00:00:00`。

### 4. 组合时间表达式

`OnCalendar` 允许你组合多个时间表达式

- 每天上午6:00和晚上18:00

```
OnCalendar=*-*-* 06:00:00, *-*-* 18:00:00
```

- 每月的1号和15号中午12:00

```
onCalendar=*-*-*-* 12:00:00, *-*-*-* 12:00:00
```

## 5. 精确时间范围

通过指定时间范围来控制定时任务的运行时间

- 每个月第一个星期一,上午9:00

```
onCalendar=Mon *-*-*-* 09:00:00
```

- 每年的1月和7月的第一个星期一上午9:00

```
onCalendar=Mon *-*-*-* 09:00:00
```

## 6. 随机延迟

可以结合 `RandomizedDelaySec` 参数, 设置一个随机的延迟时间, 以防止多个任务同时触发

```
[Timer]
onCalendar=hourly
RandomizedDelaySec=15m # 任务在每小时第0分执行, 但有最多15分钟的随机延迟
```

## 实例解析(logrotate)

- logrotate.service

```
[Unit]
Description=Rotate log files
Documentation=man:logrotate(8) man:logrotate.conf(5)
RequiresMountsFor=/var/log
ConditionACPower=true

[Service]
Type=oneshot
ExecStart=/usr/sbin/logrotate /etc/logrotate.conf

# performance options
Nice=19 # 优先级范围-20~19, Nice=19表示这个服务运行时会有最低的CPU优先级, 从而减少系统其他进程的影响
IOSchedulingClass=best-effort
IOSchedulingPriority=7

# hardening options (安全强化选项)
# details: https://www.freedesktop.org/software/systemd/man/systemd.exec.html
# no ProtectHome for userdir logs
# no PrivateNetwork for mail delivery
# no NoNewPrivileges for third party rotate scripts
# no RestrictSUIDSGID for creating setgid directories
LockPersonality=true
MemoryDenyWriteExecute=true
PrivateDevices=true
PrivateTmp=true
ProtectClock=true
```

```
ProtectControlGroups=true  
ProtectHostname=true  
ProtectKernelLogs=true  
ProtectKernelModules=true  
ProtectKernelTunables=true  
ProtectSystem=full  
RestrictNamespaces=true  
RestrictRealtime=true
```

- logrotate.timer

```
[Unit]  
Description=Daily rotation of log files  
Documentation=man:logrotate(8) man:logrotate.conf(5)  
  
[Timer]  
OnCalendar=daily  
AccuracySec=1h  
Persistent=true  
  
[Install]  
WantedBy=timers.target
```

## Linux启动流程和内核管理

### CentOS6 启动流程

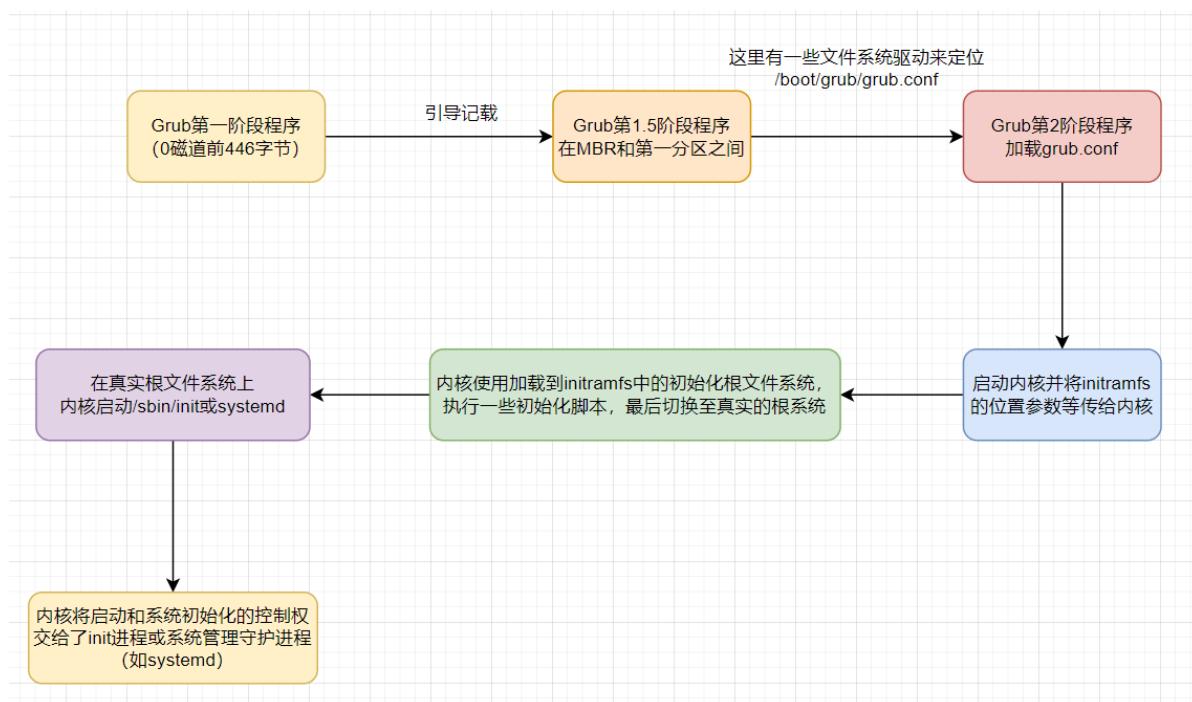
#### 硬件启动POST (Power-On-Self-Test)

- BIOS加电自检
  - BIOS是固化在主板ROM上的程序，主板上两个芯片，一个ROM，一个RAM，RAM上存储着BIOS程序的修改配置，由主板上的一颗纽扣电池供电
    - 如果RAM上没电，则BIOS里面的设置项会恢复成出厂设置。

#### GRUB启动阶段

- stage 1: MBR前446字节 (grub程序的一部分) 这部分代码负载引导bootloader
- stage 1.5: 在MBR后续的扇区中 (**1扇区到2047扇区，也就是存在于MBR和第一个分区之间的空间**)，Stage 1.5 包含了一些文件系统的驱动，这允许它能够读取位于不同文件系统上的 /boot 分区，启动其中的配置文件，并进入stage2
  - 在CentOS6中，它会启动配置文件 /boot/grub/grub.conf，
  - 在CentOS7以上版本，它会启动配置文件 /boot/grub2/grub.cfg
  - 在Ubuntu中，启动配置文件是 /boot/grub/grub.cfg
- stage2: 通过配置文件中的内容，依次启动内核，和initramfs...img，
  - 内核启动
    - 内核被加载后，控制权从GRUB转移到内核。内核首先被解压（如果需要），然后开始执行。内核初始化硬件设备、驱动程序，并设置内存管理等系统基本结构
    - **内核如何处理initramfs**

- 内核启动后，它被配置为使用 initramfs 作为其初始根文件系统。这不是由 Stage 1.5 中的文件系统驱动来处理的；而是内核自身，在被 GRUB 加载时，已经被告知 initramfs 的内存位置(**GRUB 会将 initramfs 的加载位置作为启动参数传递给内核。**)。
- 内核具有必要的代码来“理解”和挂载 initramfs，并开始执行 initramfs 中的 /init 脚本或程序，以进行早期的系统设置和驱动加载。
- 使用initramfs（此时控制权在内核手中）：
  - 内核随后使用加载到内存中的initramfs作为初始的根文件系统。initramfs中的初始化脚本开始执行，负责一系列启动前任务（例如检测硬件设备、加载必要的模块等，以确保内核能够访问真实的根文件系统。）。
- 切换到真实根文件系统：
  - 细节：一旦initramfs中的脚本准备好了真实的根文件系统（比如通过挂载文件系统），系统会进行“切换根”操作（switch\_root或pivot\_root），从initramfs的临时根文件系统切换到真实的根文件系统上。这一步涉及到卸载旧的根文件系统（initramfs）并切换到新的根文件系统。
- 启动init或systemd
  - 在真实的根文件系统中，内核会启动/sbin/init程序（这可能是一个指向如systemd或Upstart的符号链接），这是系统的第一个用户空间程序。



- grub配置文件CentOS6

```
[root@c6 ~]# ll /etc/grub.conf
lrwxrwxrwx. 1 root root 22 Aug 29 05:50 /etc/grub.conf -> ../../boot/grub/grub.conf
[root@c6 ~]# cat /boot/grub/grub.conf
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE: You have a /boot partition. This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/mapper/vg_c6-lv_root
#          initrd /initrd-[generic-]version.img
```

```
#boot=/dev/sda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title CentOS (2.6.32-71.el6.x86_64)
root (hd0,0)
kernel /vmlinuz-2.6.32-71.el6.x86_64 ro root=/dev/mapper/vg_c6-lv_root
rd_LVM_LV=vg_c6/lv_root rd_LVM_LV=vg_c6/lv_swap rd_NO_LUKS rd_NO_MD rd_NO_DM
LANG=en_US.UTF-8 SYSFONT=tatarcyrheb-sun16 KEYBOARDTYPE=pc KEYTABLE=us
crashkernel=auto rhgb quiet
initrd /initramfs-2.6.32-71.el6.x86_64.img
```

- CentOS的grub安装
- 非交互式安装

```
grub-install --root-directory=DIR /dev/DISK
```

## grub安全

- 破解CentOS6中的root口令

1. 编辑grub菜单(选定要编辑的title, 而后使用a 或 e 命令)
2. 在选定的kernel后附加参数 1 (s|s|single) , 此参数是用来进入单用户模式
3. 在kernel所在行, 敲 b 键, 以此配置启动, 即可无密码以root身份进入系统

- 设置grub密码, 防止破解系统root密码

```
grub-md5-crypt
grub-crypt

# 执行上述命令, 生成口令后, 按格式粘贴到grub配置文件中
default=0
timeout=5
password --md5 $1$B0do8$z5jKhxah4sInKxPGQRQWE0
title Centos6
root (hd0,0)
kernel /vmlinuz-2.6.32-71.el6.x86_64 ro root=/dev/mapper/vg_c6-lv_root
...
```

## 生成背景图片

```
convert -resize 640*480 -colors 14 winner.png splash.xpm
# 生成splash.xpm.gz
gzip splash.xpm
mv splash.xpm.gz /boot/grub
```

# 加载内核

## kernel自身初始化过程

- 探测可识别到的所有硬件设备
- 加载硬件驱动程序（借助于ramdisk (initramfs) 加载驱动）
- 以只读方式挂载根目录
- 运行用户空间的第一个应用程序：/sbin/init

## 内核特点

- 支持模块化：.ko(内核对象)，如：文件系统，硬件驱动，网络协议等
- 支持内核模块的动态装载和卸载

## 内核的组成

- 核心文件：/boot/vmlinuz-VERSION-release
  - ramdisk:辅助的伪根系统，加载相应的硬件启动
- 模块文件：/lib/modules/VERSION-release

## init初始化

- init进程初始化文件
  - init程序来自于upstart的包（仅CentOS6用）
  - CentOS6: /etc/inittab

```
# inittab is only used by upstart for the default runlevel.
#
# ADDING OTHER CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
#
# System initialization is started by /etc/init/rcS.conf
#
# Individual runlevels are started by /etc/init/rc.conf
#
# Ctrl-Alt-Delete is handled by /etc/init/control-alt-delete.conf
#
# Terminal gettys are handled by /etc/init/tty.conf and /etc/init/serial.conf,
# with configuration in /etc/sysconfig/init.
#
# For information on how to write upstart event handlers, or how
# upstart works, see init(5), init(8), and initctl(8).
#
# Default runlevel. The runlevels used are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:  # 定义开机启动模式
```

- 如果不小心设置为默认模式6，无限重启，可以卡在操作系统选择界面，使用-a，修改grub，在最后输入3，进入多用户模式，将初始化文件改掉
- CentOS6中的所有init相关配置文件
  - /etc/init/control-alt-delete.conf (定义了同时按这3个按键会重启)

```
# control-alt-delete - emergency keypress handling
#
# This task is run whenever the Control-Alt-Delete key combination is
# pressed. Usually used to shut down the machine.
#
# Do not edit this file directly. If you want to change the behaviour,
# please create a file control-alt-delete.override and put your changes
# there.

start on control-alt-delete

exec /sbin/shutdown -r now "Control-Alt-Delete pressed"
# 将这行注释掉，可以让这三个按键不再有重启的效果
```

- /etc/init/rc.conf

```
start on runlevel [0123456]

stop on runlevel [!$RUNLEVEL]

task

export RUNLEVEL
console output
exec /etc/rc.d/rc $RUNLEVEL
# 这段脚本是执行/etc/rc.d/rc，这个shell脚本
```

- /etc/rc.d/rc

```
for i in /etc/rc$runlevel.d/k* ; do

    # Check if the subsystem is already up.
    subsys=${i#/etc/rc$runlevel.d/K??}
    [ -f /var/lock/subsys/$subsys -o -f /var/lock/subsys/$subsys.init ] || continue
    check_runlevel "$i" || continue

    # Bring the subsystem down.
    [ -n "$UPSTART" ] && initctl emit --quiet stopping JOB=$subsys
    $i stop
    [ -n "$UPSTART" ] && initctl emit --quiet stopped JOB=$subsys
done

# Now run the START scripts.
for i in /etc/rc$runlevel.d/s* ; do

    # Check if the subsystem is already up.
    subsys=${i#/etc/rc$runlevel.d/S??}
```

```

[ -f /var/lock/subsys/$subsys ] && continue
[ -f /var/lock/subsys/$subsys.init ] && continue
check_runlevel "$i" || continue

# If we're in confirmation mode, get user confirmation
if [ "$do_confirm" = "yes" ]; then
    confirm $subsys
    rc=$?
    if [ "$rc" = "1" ]; then
        continue
    elif [ "$rc" = "2" ]; then
        do_confirm="no"
    fi
fi

update_boot_stage "$subsys"
# Bring the subsystem up.
[ -n "$UPSTART" ] && initctl emit --quiet starting JOB=$subsys
if [ "$subsys" = "halt" -o "$subsys" = "reboot" ]; then
    export LC_ALL=C
    exec $i start
fi
$i start
[ -n "$UPSTART" ] && initctl emit --quiet started JOB=$subsys
done
[ "$do_confirm" = "yes" ] && rm -f /var/run/confirm
exit 0

```

- 该脚本的作用是遍历初始化配置目录，将所有K开头的开机时关闭服务，所有S开头的开机时开启服务
- /etc/rc\$runlevel.d/目录下的所有文件都是指向/etc/init.d/目录下的服务可执行文件的软连接，**通过软链接名称的格式化，实现对多个服务的统一处理**
- chkconfig命令管理服务 (CentOS6)

### chkconfig

```

# 查看所有服务的开机启动
chkconfig --list
chkconfig --list <服务名> # 查看指定服务的模式

# 更改具体服务的开机启动项
chkconfig --level [0-6]+ <服务名> <off|on> # 不指定level，默认2, 3, 4, 5

# 实战示例：在Centos6上关闭防火墙
service iptables stop; chkconfig iptables off

chkconfig --del <服务名> # 本质上是从/etc/rc$runlevel.d/目录下的软连接删掉

```

## CentOS6中的服务脚本的书写格式

```

#!/bin/sh
#
# atd Starts/stop the "at" daemon
#

```

```

# chkconfig: 345 95 5
# description: Runs commands scheduled by the "at" command at the time \
#     specified when "at" was run, and runs batch commands when the load \
#     average is low enough.

### BEGIN INIT INFO
# Provides: atd at batch
# Required-Start: $local_fs
# Required-Stop: $local_fs
# Default-Start: 345
# Default-Stop: 95
# Short-Description: Starts/stop the "at" daemon
# Description:      Runs commands scheduled by the "at" command at the time
#     specified when "at" was run, and runs batch commands when the load
#     average is low enough.
### END INIT INFO

# Source function library.
. /etc/rc.d/init.d/functions

exec=/usr/sbin/atd
prog="atd"
config=/etc/sysconfig/atd

[ -e /etc/sysconfig/$prog ] && . /etc/sysconfig/$prog

lockfile=/var/lock/subsys/$prog

start() {
    [ -x $exec ] || exit 5
    [ -f $config ] || exit 6
    echo -n $"Starting $prog: "
    daemon $exec $OPTS
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
}

stop() {
    echo -n $"Stopping $prog: "
    if [ -n "`pidfileofproc $exec`" ] ; then
        killproc $exec
        RETVAL=3
    else
        failure $"Stopping $prog"
    fi
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
}

restart() {
    stop
    start
}

```

```

reload() {
    restart
}

force_reload() {
    restart
}

rh_status() {
    # run checks to determine if the service is running or use generic status
    status $prog
}

rh_status_q() {
    rh_status >/dev/null 2>&1
}

case "$1" in
    start)
        rh_status_q && exit 0
        $1
        ;;
    stop)
        rh_status_q || exit 0
        $1
        ;;
    restart)
        $1
        ;;
    reload)
        rh_status_q || exit 7
        $1
        ;;
    force-reload)
        force_reload
        ;;
    status)
        rh_status
        ;;
    condrestart|try-restart)
        rh_status_q || exit 0
        restart
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart|condrestart|try-
restart|reload|force-reload}"
        exit 2
esac
exit $?

```

- 重点:

- # chkconfig: 345 95 5
- 表示默认3, 4, 5模式下开机启动, S95, K5

## init总结

- 内核显示执行 /sbin/init
- /init 这个进程加载之后读取/etc/inittab这个文件
  - 用来指定默认启动级别
- 假设默认级别是3，那么就会进入/etc/rc3.d(rc\$runlevel.d)这个模式下，将S, K开头的服务，以一定的次序依次执行或关闭
- /etc/init目录下的是所有的初始化配置脚本
- /etc/rc.d/rc.sysinit 里面是所有服务加载前的初始化的脚本（由 /etc/init/rcs.conf 调用该脚本）

1. 设置主机名
2. 设置欢迎信息
3. 激活udev和selinux
4. 挂载/etc/fstab文件中定义的文件系统
5. 检测根文件系统，并以读写方式重新挂载跟文件系统
6. 设置系统时钟
7. 激活swap设备
8. 根据/etc/sysctl.conf文件设置内核参数
9. 激活lvm及software raid设备
10. 加载额外设备的驱动程序
11. 清理操作

- /etc/rc.d/rc 控制服务脚本呢的开机自动运行
  - 脚本内容是调用etc/rc\$runlevel.d/目录下的软连接
- etc/rc\$runlevel.d/ 目录下有一个特殊的文件 s99local->.../rc.local, 这个文件只有2-5有

```
#!/bin/sh
#
# 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 /var/lock/subsys/local
```

- 和普通服务格式不同，它的作用是在rc.local里的所有可执行文件都会开机启动
  - 该文件下的可执行文件，都能被service执行
  - centos7以后的版本也可以用这个文件进行开机启动，不过默认没有执行权限，需要自行添加

## 独立和非独立服务管理

- 一个服务，管理一系列不常使用的服务，负责监听，当某个服务被需要触发时，有该服务负责激活对应的服务，这种仿佛值班人的服务就是代理服务
  - 在CentOS6上，该服务叫做超级守护服务（默认没有安装，需要自行下载）

```
yum info xinetd
```

- 依赖超级守护服务的服务就叫做非独立服务

```
xinetd based services:  
    chargen-dgram: off  
    chargen-stream: off  
    daytime-dgram: off  
    daytime-stream: off  
    discard-dgram: off  
    discard-stream: off  
    echo-dgram: off  
    echo-stream: off  
    rsync: off  
    tcpmux-server: off  
    time-dgram: off  
    time-stream: off
```

- xinetd管理的子服务都在/etc/xinetd.d的目录下
- 更改自服务状态

```
# 通过命令改:  
chkconfig telnet off  
  
# 通过修改配置文件改  
vim /etc/xinetd.d/<文件名> # 修改disabled的值
```

## 实验：实现私人定义Linux

### 分区并创建文件系统

```
# 添加一个硬盘，然后分两个必要的分区  
fdisk /dev/sdb  
  
# /dev/sdb1对应 /boot /dev/sdb2对应 根/  
mkfs.ext4 /dev/sdb1  
mkfs.ext4 /dev/sdb2
```

### 挂载boot

```
mkdir /mnt/boot  
mount /dev/sdb1 /mnt/boot
```

### 安装grub

```
# 重点：/mnt的下级目录，必须有boot目录  
# 因为默认在名字为boot的下级目录中创建grub子文件目录  
grub-install --root-directory=/mnt /dev/sdb
```

### 准备内核和initramfs文件

```
cp /boot/vmlinuz-2.6.32-642.el6.x86_64 /mnt/boot/  
cp /boot/initramfs-2.6.32-642.el6.x86_64.img /mnt/boot
```

## 建立grub.conf

```
vim /mnt/boot/grub/grub.conf

# 内容如下
default=0
timeout=6
title mysticallinux
root (hd0,0)
# 禁用selinux, root=/dev/sda2是因为装好系统后, 将硬盘插到别的设备, 作为第一个硬盘的名称会默认为sda
kernel /vmlinuz-2.6.32-642.el6.x86_64 root=/dev/sda2 selinux=0 init=/bin/bash
initrd /nitramfs-2.6.32-642.el6.x86_64.img
# -----
```

## 准备根下面相关程序和库

```
mkdir /mnt/sysroot
mount /dev/sdb2 /mnt/sysroot
mkdir -pv
/mnt/sysroot{etc,lib,lib64,bin,sbin,tmp,var,usr,sys,proc,opt,home,root,boot,dev,m
nt,media}
复制bash等命令和相关库文件, 如:
bash,ifconfig,insmod,ping,mount,ls,cat,lsblk,blkid,free,df,tree,ps,pstree,mkdir,v
im,ip等
```

## 复制相关命令需要写个脚本, 脚本要求如下

- 编写脚本/root/bin/copycmd.sh
  - 提示用户输入一个可执行命令名称
  - 获取此命令所依赖到的所有库文件列表
  - 复制命令至某目标目录（例如/mnt/sysroot）下的对应路径下
    - /bin/bash --> /mnt/sysroot/bin/bash
    - /usr/bin/passwd --> /mnt/sysroot/usr/bin/passwd
  - 复制此命令依赖到所有库文件至目标目录下的对应路径下:
    - 如: /lib64/ld-linux-x86-64.so.2 --> /mnt/sysroot/lib64/ld-linux-x86-64.so.2
  - 每次复制完成一个命令后, 不要退出, 而是提示用户键入新的要复制的命令, 并重复完成上述功能, 直到用户输入quit退出
- 脚本如下

```
#!/bin/bash

ch_root="/mnt/sysroot"
[! -d $ch_root ] && mkdir $ch_root

# 拷贝命令
bincopy() {
    if which $1 &> /dev/null; then
        local cmd_path=`which --skip-alias $1`
```

```

local bin_dir=`dirname $cmd_path`
[ -d ${ch_root}${bin_dir} ] || mkdir -p ${ch_root}${bin_dir}
[ -f ${ch_root}${cmd_path} ] || cp ${cmd_paht} ${ch_root}${bin_dir}
return 0
else
    echo "Command not found."
    return 1
fi
}

# 拷贝依赖
libcopy() {
...
}

```

## 拷贝网卡驱动

```

# 查询网卡驱动名称
ethtool -i eth0

# 查看网卡驱动路径
modinfo -n e1000

# 将网卡驱动拷贝到指定的路径下
cp `modinfo -n e1000` /mnt/sysroot/lib

```

**建议手动在/etc目录下创建fstab文件，手动指定挂载目录**

**尝试切根，查看效果**

```
chroot /mnt/sysroot(切根)
```

然后在自建的根下进行测试

**将硬盘文件拿下来，然后新建虚拟机测试**

```
进入自建的操作系统后，加载网卡驱动
insmod /lib/c1000.ko
```

## /proc目录和内核参数管理

- /proc目录：内核把自己内部状态信息及统计信息，以及可配置参数通过proc伪文件系统加入输出
  - 帮助：man proc
- 内核参数：
  - 只读：只用于输出信息
  - 可写：可接受用户指定“新值”来实现对内核某功能或特性的配置
- /proc/sys设置
  - sysctl命令用于查看或设定此目录中诸多参数

```
sysctl -w path.to.parameter=VALUE

# 示例:
echo 1 > /proc/sys/net/ipv4/ip_forward

# 等价于
sysctl -w net.ipv4.ip_forward=1
```

- 默认配置文件(永久保存内核参数): /etc/sysctl.conf及以下文件

```
# 总配置文件
/etc/sysctl.conf

# 示例:
echo "net.ipv4.ip_forward=1" >> /etc/sysctl.d/test.conf

# 使其立即生效
# 如果参数加载总配置文件/etc/sysctl.conf中则不用加文件地址
# 如果参数添加到其他配置文件，则需sysctl -p后面跟绝对路径信息
sysctl -p /etc/sysctl.d/test.conf

# 其他:
/run/sysctl.d/*.conf

# 推荐
/etc/sysctl.d/*.conf
/usr/local/lib/sysctl.d/*.conf
/usr/lib/sysctl.d/*.conf
/lib/sysctl.d/*.conf
/etc/sysctl.conf
```

- sysctl命令

```
# 列出各种正在生效的内核参数
sysctl -a
```

- 常用内核参数

```
net.ipv4.ip_forward
net.ipv4.icmp_echo_ignore_all
# 允许应用程序可以监听本地不存在的IP
net.ipv4.ip_nonlocal_bind
vm.drop_cache
# 文件系统中文件最多能打开的个数
fs.file-max=1020000
```

- fs.file-max的应用场景
  - 用户访问网站，每个用户都会使该网站上打开一个socket文件，即文件描述符，如果有10w人同时访问网站，则此时服务器同时打开文件个数会是10w+，因此此时就需要优化该参数，使其可以承接这些流量
- 关于nonlocal\_bind内核参数的应用场景：

- 用户要访问一个大型互联网的IP，是一个VIP即虚拟IP，这个IP，并不是任何该互联网公司的设备IP
- 假设这个互联网公司，有两台设备，以主备的形式构建在该公司互联网的入口，用来接受用户请求，并将其发往后方
- 设备A作为主设备，日常用来接收用户请求，此时VIP和端口号绑定在设备A上，当设备A出现故障，设备B自动接任该工作
- 设备B上也应该绑定VIP和指定端口号，但是在A没有出现故障的时候，B并不工作你，但是也需要绑定VIP，防止意外，因此此时，VIP对于B来说就属于本地不存在的IP
- 内核TCP参数优化

```
net.ipv4.tcp_fin_timeout = 2
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.ip_local_port_range = 2000 65000
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_max_tw_buckets = 36000
net.ipv4.route.gc_timeout = 100
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_max_orphans = 16384
net.core.somaxconn = 16384
net.core.netdev_max_backlog = 16384
```

## /sys目录——伪文件系统（偏硬件）

### 内核模块管理和编译

- 单内核体系设计：但充分借鉴了微内核设计体系的优点，为内核引入模块化机制
- 内核组成部分：
  - kernel：内核核心，一般为bzimage，通常在/boot目录下，名称为vmlinuz-VERSION-RELEASE
  - kernel object：内核对象，一般放置于/lib/modules/VERSION-RELEASE(一些不是必须的内核功能，模块，就放在该目录下)
  - 辅助文件：ramdisk
    - initrd
    - initramfs

### 查看各驱动文件路径

```
modinfo -n ext4      # ext4文件系统的驱动路径
ethtool -i eth0      # 查看网卡驱动名称
modinfo -n e1000      # 网卡驱动路径
```

## 内核版本

- 运行中的内核:

- uname命令:

## 内核模块管理和编译

### 内核模块命令

- lsmod命令

- 显示由核心已经装载的内核模块
  - 注意: 没有装载到内核中, 不代表没有, 可以通过modinfo + 模块名查看内核中是否有这个模块
  - 类似于usb-storage这种硬件驱动, 通常在需要的时候会被自动加载
  - 显示的内容来自于: /proc/modules文件

```
# 四列信息  
模块名称  
大小  
被使用的次数  
被谁所依赖, 即依赖关系 (后面的依赖前面的)
```

- modinfo命令

- 功能: 管理内核模块
  - 配置文件: /etc/modprobe.conf, /etc/modprobe.d/\*.conf

```
-n          # 只显示模块文件路径  
-p          # 显示模块参数  
-a          # 作者  
-d          # 描述
```

```
lsmod | grep usb  
btusb           61440  0  
btrtl          24576  1 btusb  
btbcm          20480  1 btusb  
btintel        36864  1 btusb  
btmtk          16384  1 btusb  
bluetooth      724992  6 btrtl,btmtk,btintel,btbcm,btusb
```

- vmare默认无法识别usb3.0, 需要在虚拟机设置中, 将USB控制器, 手动更改USB兼容性, 使其能够识别USB3.0
  - 可以通过 /var/log/messages 查看内核日志

```
tail -f /var/log/message
```

- modprobe命令

- 功能：加载或卸载内核模块

```
modprobe -r usb_storage # 手动卸载内核(usb)模块
# 手动加载模块，比如加载usb_storage
modprobe usb_storage
```

- rmmod命令

- 功能：卸载模块

```
rmmod usb_storage
```

- 统计系统中所有的内核模块

```
find /lib/modules -name "*.ko*" | wc -l
```

## 内核编译与管理

- /boot/config-4.18.0-513.5.1.el8\_9.x86\_64
  - 定义了当前内核的配置信息

定义了哪些模块集成在内核中，后面是y  
`CONFIG_ARCH_STACKWALK=y`  
 也定义了哪些模块放在lib/modules/下，后面是m  
`CONFIG_TEXTSEARCH_FSM=m`  
 后面是is not set表示不启用该模块，也不加载该模块  
`CONFIG_DMA_API_DEBUG is not set`

## 开发环境相关包

```
gcc make ncurses-devel flex bison openssl-devel elfutils-libelf-devel
```

## 内核编译安装实现

- 下载源码
- 准备文件配置文件/boot/config `uname -r`
- make menuconfig: 配置内核选项

# 类似于常见编译中的config命令  
 通常config命令的作用  
 1. 检测依赖关系，是否缺失依赖的包  
 2. 确定安装路径  
 3. 选在加载和不加载的功能或模块  
 体现在生成的makefile文件中

# 内核的编译  
 1. 不需要指定路径  
 2. 选择的参数使用menuconfig  
`make menuconfig`  
 3. `make [-j#]` 或者用以下两步实现  
`make -j # bzImage` 编译内核  
`make -j # modules` 编译模块  
 # 通常直接使用`make -j #`(#表示并行编译的数字)

4. 安装模块: `make modules_install`
5. 安装内核相关模块: `make install`
  - 5.1 安装bzimage为`/boot/vmlinuz-VERSION-RELEASE`
  - 5.2 生成`initramfs`文件
  - 5.3 编辑`grub`的配置文件

## 准备过程

```
dnf update

dnf groupinstall "Development Tools"

dnf install ncurses-devel bison flex elfutils-libelf-devel openssl-devel
```

## 编译过程

```
# 安装依赖
yum install -y gcc make ncurses-devel flex bison openssl-devel elfutils-libelf-
devel

# 去官方下载最新内核源码
https://www.kernel.org/

# 导入linux系统中，然后解压到/usr/src
tar xf linux-6.8.9.tar.xz -C /usr/src

cd /usr/src

ln -sv linux-6.8.9 linux

cd /usr/src/linux

cp /boot/config-$(uname -r) ./config

# 注意：要更改CONFIG_SYSTEM_TRUSTED_KEYS="certs/rhel.pem"
# 默认会验证这个key，但是我们没有这个key，因此要删掉它
vim .config
#CONFIG_SYSTEM_TRUSTED_KEYS="certs/rhel.pem"
CONFIG_SYSTEM_TRUSTED_KEYS=""
# 注释掉MODULE_SIG
CONFIG_MODULE_SIG=y

# 生成一个菜单，可以在里面选择需要加载的模块和一些有趣的操作
make menuconfig

make -j 2

make modules_install

make install

reboot
```

## 内核编译说明

## 1. 配置内核选项

支持更新模式进行配置: make help

1. make config: 基于命令行以遍历的方式配置内核中可配置的每个选项
2. make menuconfig: 基于curses的文本窗口界面
3. 其他 (略)  
    支持“全新配置”模式进行配置
4. make defconfig: 基于内核为目标平台提供的默认配置进行配置
5. make allyesconfig: 所有选项均为yes
6. make allnoconfig: 所有选项均为no

## 编译内核

- 全编译:

```
make [-j #]
```

- 编译内核的一部分功能
  - 只编译某子目录中的相关代码

```
cd /usr/src/linux; make dir/
```

- 只编译一个特定的模块

```
cd /usr/src/linux; make dir/file.ko
```

- 交叉编译内核
  - 编译的目标平台与当前平台不相同

```
make ARCH=arch_name
```

- 重新编译需要事先清理操作

```
make clean # 清理大多数编译生成的文件，但会保留config文件等  
make mrproper # 清理所有编译生成的文件，config及某些备份文件  
make distclean # mrproper、清理pathches以及编辑器备份文件
```

## 卸载内核

- 删除/usr/src/linux/目录下不需要的内核源码
- 删除/lib/modules/目录下不需要的内核库文件
- 删除/boot目录下启动的内核和内核映像文件
- 更改grub的配置文件，删除不需要的内核启动列表

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

- CentOS 8 还需要删除/boot/loader/entries/XXXXX新内核版

# systemd

## systemd特性

- systemd：从CentOS7版本之后开始用systemd实现init进程，系统启动和服务器守护进程管理器，负责在系统启动运行时，激活系统资源，服务器进程和其他进程
- systemd新特性
  - 系统引导时，实现服务并行启动（并不是完全并行，有依赖关系的会有前后顺序）
  - 按需启动守护进程
  - 自动化的服务依赖关系管理
    - 如果需要启动一个服务，但这个服务需要依赖其他的服务才能启动，在systemd的管理下，会将它以及它所依赖的所有服务一并启动
  - 同时采用socket式与D-Bus总线式激活服务
  - socket与服务程序分离
    - socket代表ip端口号
    - servers服务代表一个进程
    - 可以实现一个效果，就是由systemd进行监听socket,如果有人访问该端口，才会激活这个服务，而不像之前那样，一旦一个端口处于监听状态，就必须在上面启动一个服务
  - 向后兼容sysv, init脚本
  - 使用systemctl命令管理，systemctl命令固定不变，不可扩展，非由systemd启动的服务，systemd无法与之通信和控制
  - 系统状态快照

## systemd核心概念：unit

- 查看unit类型

```
systemctl -t help

#service unit, 文件扩展名为.service, 用于定义系统服务
# Target unit, 文件扩展名为.target, 用于模拟实现运行级别
# Device unit, 用于定义内核识别的设备
# Mode unit, 定义文件系统挂载点
# Socket unit, 定义进程间通信用的socket文件, 也可在系统启动时, 延迟驱动服务, 实现按需分配
# Snapshot, 管理系统快照
# Swap 用于标识swap设备
# Automount unit: automessage, 文件系统的自动挂载点
# Path unit : .path, 用于定义文件系统中的一个文件, 或目录使用, 常用于文件系统变化时, 延迟激活服务, 入伍微信, spool目录
```

- unit的配置文件

```
/usr/lib/systemd/system; # 每个服务最主要的启动脚本设置, 类似于之前的/etc/init.d
/lib/systemd/system      # ubuntu的对应目录
/run/systemd/system;    # 系统执行过程中所产生的服务脚本, 比上面目录优先运行(内存中)
/etc/systemd/system;     # 管理员建立的执行脚本, 比上面目录优先运行(通常是命令自动生成的, 不
需要手动进行修改), 类比Centos6的rc#.d目录
```

## systemctl管理系统服务service unit

- 命令: `systemctl COMAMND name.service`

```
# 启动, 相当于service name start
systemctl start name.service

# 停止, 相当于service name stop
systemctl stop name.service

# 重启, 相当于service name restart
systemctl restart name.service

# 查看状态, 相当于service name status
systemctl status name.service

# 禁止自动和手动启动
# 本质上是创建一个软链接, 指向/dev/null
# ln -sv /dev/null /etc/systemd/system/xxx.service
systemctl mask name.service

# 取消禁止
systemctl unmask name.service

# 查看某服务当前激活与否的状态
systemctl is-active name.service

# 查看下会开机, 是否自启
systemctl is-enabled name.service

# 查看所有已激活的服务
systemctl list-units --type|-t service

# 查看所有服务
systemctl list-units --type service --all|-a

# 设置某服务开机自启, 相当于chkconfig name on
systemctl enable [--now] name.service

# 设置某服务开机禁止启动, 相当于chkconfig name off
systemctl disable [--now] name.service

# 查看所有服务的开机自启状态
systemctl list-units --type service

# 重新加载systemctl的状态
systemctl daemon-reload

# 列出失败的服务
systemctl --failed --type=service

# 查看服务的依赖关系
systemctl list-dependencies name.service

# 杀掉进程
```

```
systemctl kill unitname
```

## service unit文件格式

```
/etc/systemd/system      # 系统管理员和用户使用  
/usr/lib/systemd/system   # 发行版打包者使用
```

## service文件

功能：让systemd接管该服务

service文件格式

```
#由三个语句块组成，分别是[Unit][Service][Install]  
# [Unit] 里面是关于这个服务单元的描述信息与依赖关系  
# 修改service信息后，需要执行systemctl daemon=reload才能使其生效  
[Unit]  
# 描述文本  
Description=Job spooling tools  
# After表明，该服务必须在下列服务启动后，才能执行，即写明了该服务的依赖关系  
After=syslog.target systemd-user-sessions.service  
  
[Service]  
# EnvironmentFile指明了声明变量的文件，指明后可以直接调用  
EnvironmentFile=/etc/sysconfig/atd  
# ExecStart这个才是服务启动时，运行的指令  
ExecStart=/usr/sbin/atd -f $OPTS  
# 表示是否支持信号  
IgnoreSIGPIPE=no  
  
[Install]  
WantedBy=multi-user.target
```

service unit file文件通常由三部分组成

- 
- [Service]: 与特定类型相关的专用选项，此处为Service类型
- [Install]: 定义由"systemctl enable"以及"systemctl disable"命令在实现服务启用或禁用时用到的一些选项

Unit段的常用选项

- Description: 描述信息
- After: 定义Unit的启动次序，表示当前Unit应该晚于哪些Unit启动，其功能与Before相反
- Requires: 依赖到的其他units，强依赖，被依赖的units无法激活时，当前unit也无法激活
- Wants: 依赖到的其它units，弱依赖
- Conflicts: 定义units间的冲突关系

Service段的常用选项

- Type: 定义影响ExecStart及相关参数的功能的unit进程启动类型
  - simple: 默认值，这个daemon主要由ExecStart接的指令串来启动，启动后常驻于内存中

- forking: 由ExecStart启动的程序透过spawns延伸出其他子程序来作为此daemon的主要服务，原生父进程在启动结束后就停止
- oneshot: 与simple类似，不过这个程序在工作完毕后就结束了，不会常驻在内存中
- dbus: 与simple类似，但这个daemon必须在取得一个D-Bus的名称之后，才会继续运作，因此通常也要同时设定BusName=才行
- notify: 在启动完成后会发送一个通知信息，要需要配合NotifyAccess来让Systemd接收信息
- idle: 与simple类似，要执行这个daemon必须要所有的工作都顺利进行完毕后才会执行。这类的daemon通常是开机到最后才执行即可的服务
- EnvironmentFile: 环境配置文件
- ExecStart: 指明启动unit要运行命令或脚本的绝对路径
- ExecStartPre: ExecStart前运行
- ExecStartpost: ExecStart后运行
- ExecStop: 指明停止unit要运行的命令或脚本
- Restart: 当设定Restart=1时，则当次daemon服务以外终止后，会再次自动启动此服务
- PrivateTmp: 设定为yes时，会在生成/tmp/systemd-UUID-NAME.service.XXXtmp/目录

Install段的常用选项

- Alias: 别名，可使用systemctl commandAlias.service

socket文件格式

```
[Unit]
Description=Telnet Server Activation Socket
Documentation=man:telnetd(8)

[Socket]
ListenStream=23
Accept=true

[Install]
WantedBy=sockets.target
```

## 写service文件的技巧

- 借鉴rpm包安装的服务（在其他服务器，使用yum安装，看下安装包生成的service文件），然后看一下别人的写法，然后根据自己的需求进行调整

## 实现Ubuntu开机启动

- 创建可执行文件：/etc/rc.local
- /etc/rc.local内的脚本程序，会实现开机直接运行
- 注意：实现rc.local能够开机运行的服务：/lib/systemd/system/rc.local.service

## 运行级别

- target units: 相当于CentOS6之前的runlevel， unit配置文件：.target

```
ls /usr/lib/systemd/system/*.target
```

- 查看不同级别target的下会加载的服务

```
systemctl list-dependencies graphical.target
```

- 级别切换

```
systemctl isolate name.target
```

- 设置开机进入的默认target

```
systemctl set-default name.target
```

- 查看当前的默认target

```
systemctl get-default
```

- 切换至紧急救援模式

```
systemctl rescue
```

- 禁用ctrl+alt+delete (重启快捷键)

```
systemctl mask ctrl_alt_del.target
```

# 使其生效

```
init q      # 等价于systemctl deamon reload
```

## CentOS7之后版本的引导顺序

- 完整过程

1. UEFI或BIOS初始化，进行POST开机自检
2. 选择启动设备
3. 引导装载程序，CentOS7是grub2加载装载程序的配置文件：  
`/etc/grub.d`  
`/etc/default/grub`  
`/boot/grub2/grub.cfg`
4. 加载initramfs驱动模块
5. 加载内核选项
6. 内核初始化，CentOS7使用systemd代替init
7. 执行initrd.target所有单元，包括挂载`/etc/fatab`
8. 从initramfs根文件系统切换到磁盘根目录
9. systemd执行默认target配置，配置文件`/etc/systemd/default.target`
10. systemd执行sysinit.target初始化系统及basic.target准备操作系统
11. systemd启动multi-user.target下的本机与服务器程序
12. systemd执行multi-user.target下的`/etc/rc.d/rc.local`
13. systemd执行multi-user.target下的getty.target及登陆服务
14. systemd执行graphical需要的服务

- 通过systemd-analyze工具可以了解启动详情

```
systemd-analyze plot > boot.html
```

# 或者

```
systemd-analyze blame
```

- 通过设置内核参数进入救援模式

启动时，到启动菜单，按e键，找到linux开头的行，后添加systemd.unit=rescue.target

## 破解CentOS7, 8的root密码

# 方法1

在grub界面，按e进入编辑模式

光标移动linux开始的行，添加内核参数rd.break

按ctrl-x启动

```
mount -o remount , rw /sysroot # 因为当前的操作系统是只读，无法修改文件，因此需要重新挂载  
chroot /sysroot  
passwd root
```

# 如果selinux启用，需添加下面的命令

```
touch /.autorelabel
```

# 方法2

在linux开始的行，改为rw init=/sysroot/bin/sh

按ctrl-x启动

```
chroot /sysroot  
passwd root
```

## 实现grub2安全

添加grub密码

```
grub2-setpassword
```

# 添加密码后会在/boot/grub2目录下，添加user.cfg文件，里面是密码

```
echo "" > user.cfg # 可以清空密码
```

## 修复grub2

- 主要配置文件：/boot/grub2/grub.cfg
- 修复配置文件：grub2-mkconfig > /boot/grub2/grub.cfg
- 修复grub

```
grub2-install /dev/sda # BIOS环境  
grub2-install # UEFI环境
```

- 设置默认启动内核

```
# 以下命令是修改 /boot/grub2/grubenv实现
# 查看内核信息
ls /boot/loader/entries/
# 可以将需要设置的默认内核信息，添加替换至/boot/grub2/grubenv的saved_entry=后面，将它后面的
替换掉
grub2-set-default 0
# 或者
vim /etc/default/grub
GRUB_DEFAULT=0
```

- 实战案例1

```
# 破坏前446字节的引导信息
dd if=/dev/zero of=/dev/sda bs=1 count=446
# 光盘进入救援模式
grub2-install --root-directory=/mnt/sysimage /dev/sda
```

- 实战案例2

```
# 删除/boot/grub2/*所有内容，进行修复
光盘进入救援模式
chroot /mnt/sysimage
grub2-install /dev/sda
grub2-mkconfig -o /boot/grub2/grub.cfg
```

- 实战案例3

```
# 清空/boot下文件，进行修复
光盘进入救援模式
# 特别说明，CentOS8必须先grub，再安装kernel，否则安装kernel-core时会提示grub出错
chroot /mnt/sysimage
mount /dev/sr0 /mnt
grub2-install /dev/sda

2. 安装kernel
rpm -ivh <安装包路径> --force

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

4. 退出重启
exit
```

# 服务

## 域名系统DNS

### 名称解析介绍和DNS

- 实现此服务的方法
  - 本地名称解析配置文件: hosts
    - Linux: /etc/hosts
    - windows: %WINDIR%/system32/drivers/etc/hosts
      - windows中使用 `set %WINDIR%` 查看变量数值

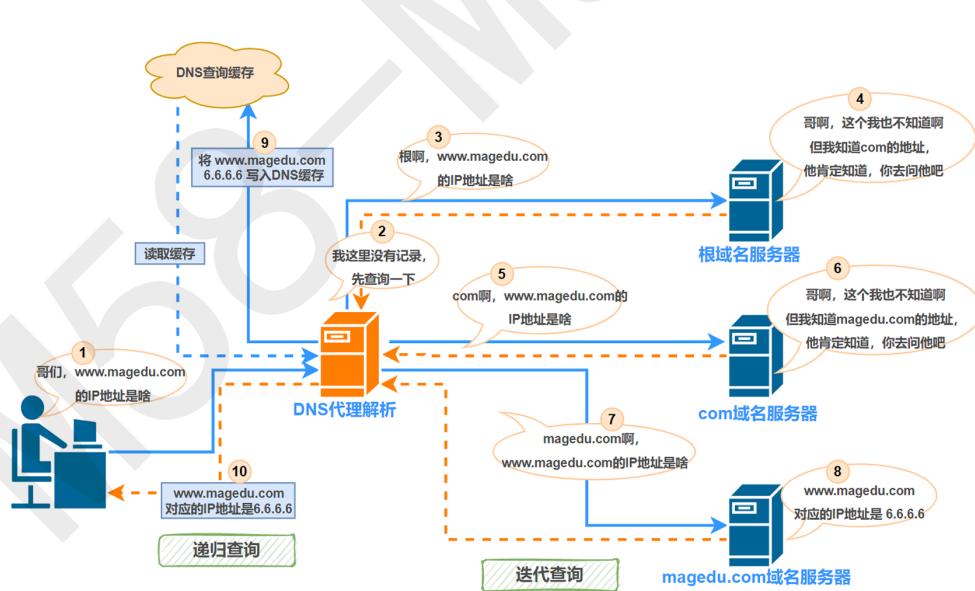
```
122.10.117.2 www.magedu.org  
93.46.8.89 www.google.com
```

- DNS: Domain Name System 域名系统应用层协议, 是互联网的一项服务, 它作为将域名和IP地址相互映射的一个分布式数据库, 能够使人更方便的访问互联网
  - 基于C/S架构, **服务端: 53/udp, 53/tcp**
- BIND: Bekerley Internet Name Domain, 由ISC ([www.isc.org](http://www.isc.org)) 提供的DNS软件实现
  - DHCP也是ISC负责维护管理的
- **DNS重要思想: 分布式管理**

### DNS域名结构

- FQDN: 全程域名 = 主机名 + 域名
  - 域名 = 子域名 + 父域名
- 域名结构
  - 根域 . 13个
  - 一级域名: Top Level Domain: com,edu,gov,org,io...
    - 三类: 组织域, 国家域, 反向域
  - 二级域名: magedu.com
  - 三级域名: study.magedu.com
  - 最多可达到127级域名
- ICANN(The Internet Corporation for Assigned Names and Numbers) 互联网名称与数字地址分配机构, 负责在全球范围内对互联网通用顶级域名以及国家和地区顶级域名系统的管理, 以及根服务器系统的管理

## DNS服务工作原理



## DNS查询类型

- **递归查询:**
  - 是指DNS服务器在收到用户发起的请求时，必须向用户返回一个准确的查询结果。如果DNS服务器本地没有存储与之对应的信息，则该服务器需要询问其他服务器，并将返回的查询结果提供给用户。
  - 一般客户机和本地DNS服务器之间属于递归查询，即当客户机向DNS服务器发出请求后，若DNS服务器本身不能解析，则会向另外的DNS服务器发出查询请求，得到最终的肯定或否定的结果后转交给客户机。
  - 此查询的源和目标保持不变，为了查询结果只需要发起一次查询。
  - 客户端向LocalDNS发起域名查询-->localDNS不知道域名对应的IP-->但它知道谁知道->他代为帮客户端去查找-->最后再返回最终结果。
- **迭代查询:**
  - 是指DNS服务器在收到用户发起的请求时，并不直接回复查询结果，而是告诉另一台DNS服务器的地址，用户再向这台DNS服务器提交请求，这样依次反复，直到返回查询结果。
  - 一般情况下(有例外)本地的DNS服务器向其它DNS服务器的查询属于迭代查询，如：若对方不能返回权威的结果，则它会向下一个DNS服务器(参考前一个DNS服务器返回的结果)再次发起进行查询，直到返回查询的结果为止。此查询的源不变，但查询的目标不断变化，为查询结果一般需要发起多次查询。

## 解析类型

- FQDN ---> IP 正向解析
- IP ----> FQDN 反向解析
- 注意：正反向解析是两个不同的名称空间，是两颗不同的树

## 完整的查询请求经过的流程

```
client -->
hosts文件 -->
Client DNS Service Local Cache -->
DNS Server(Recursion 递归) -->
DNS Service Cache --> iteration(迭代) -->
根 -->
顶级域名DNS -->
二级域名...
```

## DNS缓存

- Windows系统中显示DNS缓存

```
ipconfig/displaydns # 正斜杠, 这里没写错
```

- Windows系统中清除DNS缓存

```
ipconfig/flushdns
```

- CentOS系统中查看DNS缓存

nscd (Name Service Cache Daemon) 是一个用于缓存各种名称服务请求的守护进程，比如用户和组信息、主机名、服务等

```
# 下载nscd
yum install -y nscd

# 启动nscd
nscd

# 查看DNS缓存
nscd -g

hosts cache:

yes cache is enabled
yes cache is persistent
yes cache is shared
211 suggested size
# 缓存数据池的总大小（以字节为单位）。这个数据池用于存储缓存的主机名解析结果。
216064 total data pool size
# 当前已使用的数据池大小（以字节为单位）。表示目前没有缓存的数据被存储。
0 used data pool size
# 正面条目的生存时间（TTL，单位为秒）。表示成功解析的主机名缓存 3600 秒（1 小时）后过期。
3600 seconds time to live for positive entries
# 负面条目的生存时间（TTL，单位为秒）。表示未能解析的主机名缓存 20 秒后过期
20 seconds time to live for negative entries
# 正面条目的缓存命中次数。表示从缓存中成功获取到解析结果的次数。
0 cache hits on positive entries
# 负面条目的缓存命中次数。表示从缓存中成功获取到未能解析结果的次数。
0 cache hits on negative entries
1 cache misses on positive entries
```

```
1 cache misses on negative entries
# 缓存命中率。表示命中次数在总查询次数中的比例，目前为 0%。
0% cache hit rate
# 当前缓存的条目数量。表示当前缓存中没有任何条目
0 current number of cached values
2 maximum number of cached values
0 maximum chain length searched
0 number of delays on rdlock
0 number of delays on wrlock
0 memory allocations failed
yes check /etc/hosts for changes

# 清除缓存
nscd -i hosts
```

- Ubuntu系统中查看DNS缓存

```
# systemd-resolved的配置文件
vim /etc/systemd/resolved.conf

systemctl restart systemd-resolved

# 查看DNS缓存
resolvectl statistics

# 清空DNS缓存
resolvectl reset-statistics
```

- 使用systemd-resolved服务需要提前准备的事情

- 将之前的 /etc/resolve.conf 删掉

```
rm /etc/resolve.conf
```

- 建立链接路径正确的软链接文件

```
ln -sf /run/systemd/resolve/stub-resolv.conf /etc/resolv.conf
```

- 确保 [Resolve] 部分包含以下内容，并且没有被注释

```
[Resolve]
DNS=114.114.114.114 180.76.76.76
DNSStubListener=yes
```

- 重启服务

```
systemctl restart systemd-resolved
```

# DNS软件bind

- DNS服务器软件: bind, powerdns (基于LAMP) , unbound, coredns

## BIND相关程序包

```
yum list all bind*
```

- bind: 服务器
- bind-libs: 相关库
- bind-utils: 客户端
- bind-chroot: 安全包, 将dns相关文件放至 /var/named/chroot
- 范例: 安装bind软件

```
yum install -y bind
```

## BIND包相关文件

- BIND主程序: /usr/bin/named
- rndc使用953端口
- 本地配置的DNS服务器文件 /etc/resolv.conf
- 当安装并启动named的时候, 此时并没有任何DNS配置, 上面自带关于根域的记录, 此时该DNS服务器叫做只缓存服务器, 初次访问的所有域名都通过访问根, 依次访问后几级域名, 来实现DNS域名转换
- cat /var/named/named.ca 存放了13个根记录 (Ubuntu中是 /usr/share/dns/root.hints )

```
# /var/named/named.ca
;; ANSWER SECTION:
.          518400  IN      NS      a.root-servers.net.
.          518400  IN      NS      b.root-servers.net.
.          518400  IN      NS      c.root-servers.net.
.          518400  IN      NS      d.root-servers.net.
.          518400  IN      NS      e.root-servers.net.
.          518400  IN      NS      f.root-servers.net.
.          518400  IN      NS      g.root-servers.net.
.          518400  IN      NS      h.root-servers.net.
.          518400  IN      NS      i.root-servers.net.
.          518400  IN      NS      j.root-servers.net.
.          518400  IN      NS      k.root-servers.net.
.          518400  IN      NS      l.root-servers.net.
.          518400  IN      NS      m.root-servers.net.

;; ADDITIONAL SECTION:
a.root-servers.net.   518400  IN      A       198.41.0.4
b.root-servers.net.   518400  IN      A       199.9.14.201
c.root-servers.net.   518400  IN      A       192.33.4.12
d.root-servers.net.   518400  IN      A       199.7.91.13
e.root-servers.net.   518400  IN      A       192.203.230.10
f.root-servers.net.   518400  IN      A       192.5.5.241
g.root-servers.net.   518400  IN      A       192.112.36.4
```

h.root-servers.net.	518400	IN	A	198.97.190.53
i.root-servers.net.	518400	IN	A	192.36.148.17
j.root-servers.net.	518400	IN	A	192.58.128.30
k.root-servers.net.	518400	IN	A	193.0.14.129
l.root-servers.net.	518400	IN	A	199.7.83.42
m.root-servers.net.	518400	IN	A	202.12.27.33
a.root-servers.net.	518400	IN	AAAA	2001:503:ba3e::2:30
b.root-servers.net.	518400	IN	AAAA	2001:500:200::b
c.root-servers.net.	518400	IN	AAAA	2001:500:2::c
d.root-servers.net.	518400	IN	AAAA	2001:500:2d::d
e.root-servers.net.	518400	IN	AAAA	2001:500:a8::e
f.root-servers.net.	518400	IN	AAAA	2001:500:2f::f
g.root-servers.net.	518400	IN	AAAA	2001:500:12::d0d
h.root-servers.net.	518400	IN	AAAA	2001:500:1::53
i.root-servers.net.	518400	IN	AAAA	2001:7fe::53
j.root-servers.net.	518400	IN	AAAA	2001:503:c27::2:30
k.root-servers.net.	518400	IN	AAAA	2001:7fd::1
l.root-servers.net.	518400	IN	AAAA	2001:500:9f::42
m.root-servers.net.	518400	IN	AAAA	2001:dc3::35

- 每个根域名后面是一堆服务器，这里的IP是VIP虚拟IP
- 通过 `ss -ntulp` 可以查看所有端口的信息

```
[root@localhost /etc/sysconfig/network-scripts] $ ss -tulpn
Netid          State          Recv-Q
Send-Q          Local Address:Port
   Peer Address:Port      Process

udp            UNCONN         0
                           127.0.0.1:53
                           0.0.0.0:*
   users:
(("named",pid=3286,fd=512))
udp            UNCONN         0
                           [::]:53
                           ::/*:*
   users:
(("named",pid=3286,fd=513))
tcp            LISTEN          0
10             127.0.0.1:53
                           0.0.0.0:*
   users:
(("named",pid=3286,fd=21))
tcp            LISTEN          0
128            0.0.0.0:22
                           0.0.0.0:*
   users:
(("sshd",pid=1006,fd=3))
tcp            LISTEN          0
128            127.0.0.1:953
                           0.0.0.0:*
   users:
(("named",pid=3286,fd=23))
tcp            LISTEN          0
10             [::]:53
                           ::/*:*
   users:
(("named",pid=3286,fd=22))
tcp            LISTEN          0
128            [::]:22
                           ::/*:*
   users:
(("sshd",pid=1006,fd=4))
tcp            LISTEN          0
128            [::]:953
                           ::/*:*
   users:
(("named",pid=3286,fd=24))
```

- 发现， udp和tcp的53端口是监听在127.0.0.1这个端口上，也就是说，只能本机进行访问，无法对外提供服务，
- 更改named上监听个端口需要修改配置文件(典型C语言风格的配置文件)
  - /etc/named.conf

```
options {
    # 更改这里
    // listen-on port 53 { 127.0.0.1; };
    listen-on port 53 { 127.0.0.1; 10.0.0.171;};
    listen-on-v6 port 53 { ::1; };
    directory      "/var/named";
    dump-file      "/var/named/data/cache_dump.db";
    statistics-file "/var/named/data/named_stats.txt";
    memstatistics-file "/var/named/data/named_mem_stats.txt";
```

```

    secroots-file    "/var/named/data/named.secroots";
    recursing-file  "/var/named/data/named.recursing";
    allow-query      { localhost; };

    /*
     - If you are building an AUTHORITATIVE DNS server, do NOT enable
     recursion.
     - If you are building a RECURSIVE (caching) DNS server, you need to
     enable
         recursion.
     - If your recursive DNS server has a public IP address, you MUST enable
     access
         control to limit queries to your legitimate users. Failing to do so
     will
         cause your server to become part of large scale DNS amplification
         attacks. Implementing BCP38 within your network would greatly
         reduce such attack surface
    */
    recursion yes;

    dnssec-enable yes;
    dnssec-validation yes;

    managed-keys-directory "/var/named/dynamic";

    pid-file "/run/named/named.pid";
    session-keyfile "/run/named/session.key";

    /* https://fedoraproject.org/wiki/Changes/CryptoPolicy */
    include "/etc/crypto-policies/back-ends/bind.config";
};

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";

```

- 修改/etc/named.conf之后，使用named-checkconf进行该文件的语法检查

```

# 将/etc/named.conf中
listen-on port 53 { 127.0.0.1; };
# 改为
listen-on port 53 { localhost; };
# localhost是一个关键字在named中，表示本地所有的地址
named-checkconf

```

```

# 重启
systemctl reload named
rndc reload # 等价

# 问题排查期间使用的命令
ss -ntupl

tcpdump -i <网卡名> prot 端口

# 更改下列字段，使其可以接受外界的传输
allow-query { localhost; 10.0.0.0/24; }; # 这个位置可以写网段，表示这个网段内的所有设备
都可以通过该DNS服务器实现域名查询
allow-query { any; }; # any表示这台DNS是公共服务器，任何主机都可以通过它查询IP

# 更简单的方法
将allow-query和listen-on 全部注释掉

DNS默认允许任何设备从主服务器拉取信息
# 严重安全问题补丁
# 只允许从服务器进行区域传输(主服务器配置)
allow-transfer { 从服务器IP; };

# 不允许其他主机进行区域传输
allow-transfer { none; }; (从服务器配置)

```

- 使用dig抓取信息的方法

```
dig -t axfr magedu.org @10.0.0.8
```

## DNS服务器类型

- 主DNS服务器
  - 管理和维护所负责解析的域内解析库的服务器
- 从DNS服务器
  - 从主服务器或从服务器复制（区域传输）解析库副本
    - 序列号：解析库版本号，主服务器解析库变化时，其序列递增
    - 刷新时间间隔：从服务器从主服务器请求同步解析的时间间隔
    - 重试时间间隔：从服务器请求同步失败时，再次尝试时间间隔
    - 过期时间：从服务器联系不上主服务器时，多久会停止服务
    - 通知机制：主服务器解析库发生变化时，会主动通知从服务器
- 缓存DNS服务器（转发器）

## 区域传输

- 完全传输：传输整个解析库
- 增量传输：传递解析库变化的那部分内容

## 主DNS服务器实现

- 要创建维护一个magedu.org的域，就要创建一个解析magedu.org的域的数据库
  - 该数据库单独存放，这个数据库文件里面记录所有magedu.org为后缀的域以及IP的对应关系
  - 一个区域数据库对应一个域
  - 该区域数据库由一行一行的区域记录组合而成
- 区域解析库：由众多RR组成
  - 各种资源记录： RR (Resource Record)
  - 记录类型： A, AAAA, PTR, SOA, NS, CNAME, MX
  - SOA: Start Of Authority (权威)，起始授权记录：一个区域解析库有且仅有一个SOA记录，必须位于解析库的第一条记录
  - A: Internet Addresses: 作用： FQDN -> IP
  - AAAA: FQDN -> IPv6
  - PTR: IP -> FQDN
  - NS: NAME Service，专用于标明当前区域的DNS服务
  - CNAME: Canonical Name，别名记录
  - MX: 邮件交换器
  - TXT: 对域名进行标识和说明的一种方式
- 资源记录定义的格式

```
name [TTL] IN rr_type value
```

- 注意：
  - TTL可以从全局继承
  - 使用@符号可用于引用当前区域的名字
- SOA

```
$TTL 86400
# 一个小时做一次拉取操作，如果失败了10分钟后重试，当从服务器和主服务器无法同步，一天之后，从主将
无法对外提供服务，最后一个参数：错误结果缓存时间
@ IN SOA ns1.magedu.org admin.magedu.org (123, 1h, 10m, 1d, 12h)
ns1.magedu.org IN A 10.0.0.8

#: ns1.magedu.org: 表示当前维护magedu.org这个域的服务器的名称（我是谁）

#: 后面需要补一个
ns1.magedu.org IN A 10.0.0.33 # 我在哪

# admin.magedu.org: 域名服务器的管理员邮箱，指代admin@magedu.org

# 123: 该数据库版本名称
# 1h: 每隔一个小时从服务器从主服务器拉取数据，（是否更新的判断是版本号）
# 10m: 如果同步失败，则每10分钟重新尝试一次
# 1d: 如果1天以上都无法实现同步，则从服务器数据失效
# 12h: 错误访问记录的缓存有效期，即如果用户查找一个不存在数据库中的域名，则定一个缓存有效期，12
个小时内，用户如果继续访问，则直接返回该记录不存在。（否定答案的TTL值）
```

- 主服务器和从服务器之间一般有两种操作：一种是推push，一种是拉pull
  - 主服务器数据变了，主动把数据推送给从服务器
  - 从服务器定时从主服务器 拉取数据，查看是否更新
- NS：通过NS记录，判断当前数据库中，有哪些主从服务器
  - 对NS记录而言，任何一个ns记录后面的服务器名称，都应该在后续有一个A记录

```
@ IN NS ns1      # 主服务器
@ IN NS ns2      # 从服务器
ns1 IN NS ip_addr
ns2 IN NS ip_addr
```

- 存放DNS数据库的路径 /var/named
- DNS数据库命名规范：所管理的域名+zone
  - 示例：magedu.org.zone
  - 新建DNS数据库的时候，可以复制之前的自带的文件，`cp -p 源文件`，将其权限等一同复制

## DNS客户端测试工具

- host命令

```
host 域名
# 测试该域名是否能够解析成功
```

- dig命令

```
dig 域名
dig 域名 @DNS服务器IP    # @表示直接向该DNS服务器进行查询，而不是向默认服务器查询
# 相比host，其信息显示更全
```

- nslookup

```
nslookup 域名
```

- 使用CNAME实现流量分发(负载均衡)

```
@ IN SOA ns1 admin.magedu.org (123, ...)
@ NS ns1
ns1 A 10.0.0.8

www    CNAME    websrv
websrv A      10.0.0.6
websrv A      10.0.0.7
# 此时用户访问www，流量会被均摊到6,7两台服务器上，从而实现服务器性能的提升
```

## 区域数据库写完后，通过配置文件让named服务加载区域数据库

- 全局配置文件: `/etc/named.conf` (不推荐)

```
# 格式:  
zone "." IN {  
    type hint|master|slave;  
    file "named.ca";  
};
```

- `/etc/named.rfc1912.zones` (推荐), 将数据库信息加载至该文件

```
# 格式:  
zone "." IN {  
    type hint|master|slave;  
    file "named.ca";  
};  
  
# hint指示该服务器启动时应如何寻找根DNS服务器  
# hint类型的作用是在DNS查询过程开始时，提供一个初始的查询点  
  
zone "magedu.org" IN {  
    type master;  
    file "magedu.org.zone";  
    # 路径不用写，默认找/var/named，就是这么智能  
};
```

- 配好之后检查

```
# 配置文件语法检查  
named-checkconf  
  
# 区域数据库语法检查  
named-checkzone 域名 区域数据库路径
```

- 检查成功后，重新加载

```
systemctl reload named  
rndc reload
```

## 解析邮件服务器

```
@ MX 10 mailsrv  
@ MX 20 mailsrv2  
mailsrv A 10.0.0.6  
mailsrv B 10.0.0.7
```

- 查询当前域中，谁是邮件服务器

```
dig -t mx magedu.org
```

## 泛解析案例

```
# 使用 *
# 使用 @

# $GENERATE 1-254 HOST$ IN A 1.2.3.$ 相当于
HOST1 IN A 1.2.3.1e
HOST2 IN A 1.2.3.2
...
HOST254 IN A 1.2.3.254
```

## 实现反向解析区域

- 应用场景：邮件服务
- 反向区域：即将IP反向解析为FQDN
- 区域名称：网络地址反写in.addr.arpa

```
172.16.100. --> 100.16.172.in-addr.arpa.
```

- 定义区域

```
zone "ZONE_NAME" IN {
    type {master | slave | forward};
    file "网络地址.zone"; # eg:10.0.0.zone
};
```

- 定义区域解析库文件
  - 注意：不需要MX，以PTR记录为主
- 区域数据库(10.0.0.zone)

```
$TTL 1D

@ IN SOA ns1.magedu.org. admin.magedu.org. (1 12H 10M 3D 1D )

NS ns1.magedu.org.

7 PTR websrv.magedu.org.
123 PTR mailsrv.magedu.com.
```

- 测试反向解析

```
dig -t ptr 7.0.0.10.in-addr.arpa
# 等价于
dig -x 10.0.0.7
```

## 从服务器的实现

- 实现过程
  - 应该为一台独立的名称服务器
  - 主服务器的区域解析库中必须有一条NS记录，指向从服务器
  - 从服务器只需要定义区域，而不需提供解析库文件；解析库文件应该放置在域/var/named/slaves目录中
  - 主服务器得到允许从服务器做区域传输
  - 主服务器时间应该同步，通过ntp进行
  - bind程序的版本应该保持一致；否则，应该从高，主低
- 定义从区域

```
zone "ZONE_NAME" IN {  
    type slave;  
    masters { MASTER_IP; };  
    file "slaves/ZONE_NAME.zone" ;  
}
```

- 主服务器增加NS记录

```
@ NS ns2  
ns2 A 10.0.0.18 #从服务器IP
```

## 安全加固

- 默认任何主机都可以从主服务器同步区域数据库，这有很大的安全风险
- 解决方法

```
# 主从服务器添加配置信息  
allow-transfer {};  
# 主服务器  
allow-transfer { 从服务器IP; };  
# 从服务器  
allow-transfer { none; };
```

## 实现子域

```
magedu.org 10.0.0.8 主DNS服务器
```

```
# 用另一台设备来负责子域DNS服务  
sh.magedu.org 10.0.0.28
```

- 在10.0.0.28上实现

```
hostnamectl set-hostname shanghai  
  
# 在父的区域数据库中，增加子域记录  
sh NS ns3  
ns3 A 10.0.0.28
```

```

# 在10.0.0.28配置
/etc/named.conf

注释: listen-on port 53...
注释: allow-query { localhost; };

allow-transfer { none; };

# 添加数据库引用
zone "sh.magedu.org" {
    type master;
    file "sh.magedu.org.zone"
};

# 在/var/named/创建sh.magedu.org.zone
$TTL 1D
@ IN SOA ns1 admin.magedu.org. (23, 1D, 1H, 1W, 3H)
@ NS ns1
ns1 A 10.0.0.28
www CNAME websrv
websrv A 10.0.0.66

```

- 易错点：注意子域数据库的权限问题

## 实现DNS转发（缓存）服务器

- DNS转发：利用DNS转发，可以将用户的DNS请求，转发至指定的DNS服务，而非默认的根DNS服务器，并将指定服务器查询的返回结果进行缓存，提高效率
- 注意：
  - 被转发的服务器需要能够为请求者做递归，否则转发请求不予进行
  - 在全局配置中，关闭dnssec功能

```

dnssec-enable no;
dnssec-validation no;

```

## 全局转发

- 对非本机所负责解析区域的请求，全转发给指定的服务器
- 在全局配置块中实现：

```

options {
    forward first| only;
    forwarders { ip; };
}

# first 表示如果转发给的DNS服务器上没有需要的记录，则自己去互联网查询
# only 表示如果转发给的DNS服务器上没有需要的记录，直接返回失败，自己本身不去查询

```

## 特定区域转发

- 仅转发特定的区域请求，比全局转发优先级高

```
zone "ZONE_NAME" IN {  
    type forward;  
    forward first | only;  
    forwarders { ip; };  
}
```

## 智能DNS

- 目的：让网民访问网站的速度更快

## GSLB

- GSLB：Global Server Load Balance 全局负载均衡；GSLB是对服务器和链路进行综合判断来决定哪个地点的服务器来提供服务，实现异地服务器群服务质量的保证；GSLB主要的目的是在整个网路范围内将用户的请求定向到最近的节点（或者区域）；
- GSLB分为基于DNS实现，基于重定向实现，基于路由协议实现，其中最通用的是基于DNS解析方式
- 范例：查询VIP（唯品会）使用网宿的CDN服务

## CDN (Content Delivery Network) 内容分发网络

- CDN服务商
  - 服务商：阿里，腾讯，蓝汛，网宿，帝联等
  - 智能DNS：dnsppod, dns.la

## 智能DNS相关技术

- bind中ACL：把一个或多个地址归并为一个集合，并通过一个统一的名称调用
- 注意：只能先定义，后使用；因此一般定义在配置文件中，处于options的前面
- 格式

```
acl acl_name {  
    ip;  
    net/prelen;  
};
```

- 范例：

```
acl beijingnet{  
    172.16.0.0/16;  
    10.10.10.10;  
    ...  
};
```

- bind有四个内置的acl
  - none：没有一个主机

- andy: 都是主机
- localhost: 本机
- localnet: 本机的IP同掩码运算后得到的网络地址
- 访问控制的指令
  - allow-query{}: 允许查询的主机: 白名单
  - allow-transfer{}: 允许区域传送的主机; 白名单
  - allow-recursion{}: 允许递归的主机, 建议全局使用
  - allow-update{}: 允许更新区域数据库中的内容

## view视图

- View: 视图, 将ACL和区域数据库实现对应关系, 以实现只能DNS
  - 一个bind服务器可以定义多个view, 每个view中可定义一个或多个zone
  - 每个view用来匹配一组客户端
  - 多个view内可能需要对同一个区域进行解析, 但使用不同的区域解析文件
- 注意:
  - 一旦启用了view, 所有的zone都只能定义在view中
  - 仅在允许递归请求的客户端所在view中定义根区域
  - 客户端请求到达时, 是自上而下检查每个view所服务的客户端列表
- view格式

```

view VIEW_NAME {
    match-clients { beijingnet; };
    zone "magedu.org" {
        type master;
        file "magedu.org.zone.bj";
    };
    include "/etc/named.rfc1912.zones";
};

view VIEW_NAME {
    match-clients { shanghaiet; };
    zone "magedu.org" {
        type master;
        file "magedu.org.zone.sh";
    };
};

```

## 实战案例：利用view实现智能DNS

- 实验目的: 搭建DNS主存服务架构, 实现DNS服务冗余
- 环境要求:

需要五台主机

DNS主服务器和web服务器1: 192.168.8.8/24 172.16.0.8/16

web服务器2: 192.168.8.7/24

web服务器3: 172.16.0.7/16

DNS客户端1: 192.168.8.6/24

DNS客户端2: 172.16.0.6/16

- 前提准备

关闭selinux

关闭防火墙

时间同步

## 实现步骤

- DNS服务器的网卡配置

```
# 配置两个IP地址
# eth0: 192.168.8.8/24
# eth1: 172.16.0.8/16
```

- 主DNS服务器配置文件实现view

```
yum install bind -y

vim /etc/named.conf
# 在文件最前面加下面行
acl beijingnet {
    192.168.8.0/24
};

acl shanghaiet{
    172.16.0.0/16
};

acl othernet {
    any;
};

# 注释掉下面两行
//listen-on-port 53...
// allow-query...

# 创建view
view beijingview {
    match-clients { beijingnet; };
    include "/etc/named.rfc1912.zones.bj";
};

view shanghaiview {
    match-clients { shanghaiet; };
    include "/etc/named.rfc1912.zones.sh";
};

view otherview {
```

```
matchclients { othernet; };
include "/etc/named.rfc1912.zones.other"
}
include "/etc/named.root.key"
```

- 实现区域配置文件

```
vim /etc/named.rfc1912.zones.bj
zone "." IN {
    type hint;
    file "named.ca";
}

zone "magedu.org" {
    type master;
    file "magedu.org.zones.bj";
};

vim /etc/named.rfc1912.zones.sh
zone "." IN {
    type hint;
    file "named.ca";
}

zone "magedu.org" {
    type master;
    file "magedu.org.zones.sh";
};

vim /etc/named.rfc1912.zones.other
zone "." IN {
    type hint;
    file "named.ca";
}

zone "magedu.org" {
    type master;
    file "magedu.org.zones.other";
};

chgrp named /etc/named.rfc1912.zones.bj
chgrp named /etc/named.rfc1912.zones.sh
chgrp named /etc/named.rfc1912.zones.other
```

- 创建区域数据库文件

```
vim /var/named/magedu.org.zones.bj
$TTL 1D
@ IN SOA master admin.magedu.org (...)
    NS master
master A 192.168.8.8
websrv A 192.168.8.7
www      CNAME websrv
```

```

vim /var/named/magedu.org.zone.sh
$TTL 1D
@ IN SOA master admin.magedu.org (...)

    NS master
master A 192.168.8.8
websrv A 172.16.0.7
www      CNAME websrv

vim /var/named/magedu.org.zone.other
$TTL 1D
@ IN SOA master admin.magedu.org (...)

    NS master
master A 192.168.8.8
websrv A 127.0.0.1
www      CNAME websrv

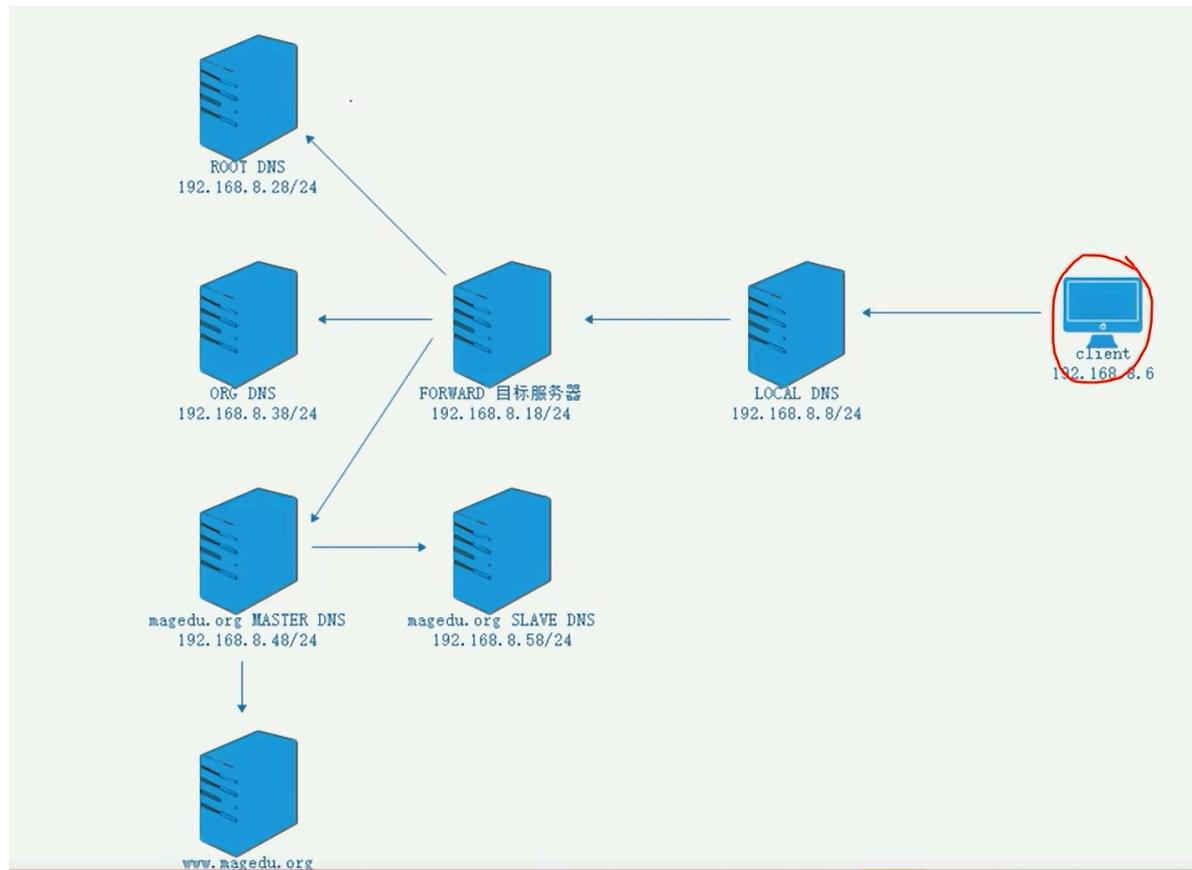
chgrp named /var/named/magedu.org.zone.bj
chgrp named /var/named/magedu.org.zone.sh
chgrp named /var/named/magedu.org.zone.other

systemctl start named    # 第一次启动
rndc reload              # 不是第一次启动服务

```

- 实现位于不同区域的三个WEB服务器：略
- 客户端测试

## 综合实验



准备8台机器

```
10.0.0.201    client
10.0.0.202    localdns
10.0.0.203    forwarddns
10.0.0.204    rootdns
10.0.0.204    orgdns
10.0.0.206    magedumaster
10.0.0.207    mageduslave
10.0.0.208    webserver
```

在复杂架构中，我们进行配置，要保证每次配置都能够得到验证，防止全部配置后，出现问题无法排错，因此，这里我们从下往上搭建，先搭建主从复制

#### 客户单client配置

```
# 更改网卡配置，将DNS指向10.0.0.202
# Ubuntu将resolve和网卡DNS对齐
rm -rf /etc/resolv.conf
ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf

# 保证所有主机都能ping通
```

#### 配置Webserver

```
# 下载nginx
apt install -y nginx
systemctl enable --now nginx
echo www.magedu.org > /var/www/html/index.html
```

#### 搭建主MasterDNS服务器

```
# 服务器: 10.0.0.206

vim /etc/bind/name.conf.default-zones

# 添加
zone "magedu.org" {
    type master;
    file "/etc/bind/db.magedu.org.zone"; # 写绝对路径
};

vim /etc/bind/db.magedu.org.zone

$TTL 1D

@    IN    SOA    master    admin.magedu.org. (1 1D 1H 1W 3H)

                NS    master
                NS    slave

master    A    10.0.0.206
slave     A    10.0.0.207
www      A    10.0.0.208

# 授权
```

```
chown bind:bind db.magedu.org.zone

# 检查语法命令
named-checkzone magedu.org /etc/bind/db.magedu.org.zone
# 检查bind配置
named-checkconf
```

## 实现从节点DNS服务器

```
vim /etc/bind/named.conf.default-zones

zone "magedu.org" {
    type slave;
    masters {10.0.0.206;};
    file "/etc/bind/slaves/magedu.org.zone"; #建议绝对路径
};

mkdir /etc/bind/slaves
chown -R bind:bind /etc/bind/slaves

# 给权限
vim /etc/apparmor.d/usr.sbin.named

# 添加
/etc/bind/slaves/** rw,
/etc/bind/slaves/ rw,

# 重启
systemctl restart named
```

## 配置magdu的父域

```
# 服务器: 10.0.0.205, orgdns
apt install -y bind9

vim /etc/bind/named.conf.default-zones

zone "org." {
    type master;
    file "/etc/bind/db.org";
}

vim db.org

$TTL 86400

@ IN SOA master admin.magedu.org. ( 123 1D 1H 1W 3D )

NS master

magedu NS mageduns1
magedu NS mageduns2

master A 10.0.0.205
```

```
mageduns1 A 10.0.0.206
```

```
mageduns2 A 10.0.0.207
```

```
# 重启
```

```
rndc reload
```

## 实现根域的主DNS服务器

```
# 服务器: 10.0.0.204 rootdns
# 修改配置
vim named.conf.default-zones

zone "." {
    type master;
    file "/etc/bind/root.zone";
};

vim root.zone

$TTL 1D

@ IN SOA master admin.magedu.org. ( 1 1D 1H 1W 3D )

        NS master
org NS orgns

master A 10.0.0.204
orgns A 10.0.0.205
```

## 实现转发目标的DNS服务器

```
# 转发DNS: 10.0.0.203 forwarddns

apt install -y bind9

vim /usr/share/dns/root.hints
.          3600000      NS      A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.   3600000      A      10.0.0.204

# 重启服务
systemctl restart named

# ubuntu, 关闭安全加密
dnssec-validation no;

# 如果是rocky, 将安全加密关闭
dnssec-enable no;
dnssec-validation no;

# 测试
nslookup www.magedu.org 10.0.0.203
```

## 配置本地localDNS服务器

```
# 服务器: 10.0.0.202 localdns

# 设置转发, 并取消安全加固
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow multiple
    // ports to talk. See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses replacing
    // the all-0's placeholder.

    forward first;
    forwarders { 10.0.0.203; };

    //=====
    // If BIND logs error messages about the root key being expired,
    // you will need to update your keys. See https://www.isc.org/bind-
    //=====

    dnssec-validation no;

    listen-on-v6 { any; };

};
```

### 在客户端测试

```
# 在客户端10.0.0.201测试
[root@client ~] $ curl www.magedu.org
www.magedu.org

# 成功
```

# Ubuntu中的systemd-resolved服务

- 在ubuntu系统中，虽然在网卡中配置了DNS服务器的IP地址，但在使用相关命令进行DNS解析时，默认的DNS服务器使用的是127.0.0.53，而并不是我们在网卡上配置的DNS服务器地址

State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port	Process
LISTEN	0	10	10.0.0.108:53	0.0.0.0:*	users(({"name": "pid-991", "fd": 25}))
LISTEN	0	10	10.0.0.108:53	0.0.0.0:*	users(({"name": "pid-991", "fd": 25}))
LISTEN	0	128	0.0.0.0:22	0.0.0.0:*	users(({"csdn": "pid-991", fd: 3}))
LISTEN	0	4996	0.0.0.0:111	0.0.0.0:*	users(({"rphbind": "pid-76", fd: 4}, {"system": "pid-1", fd: 36}))
LISTEN	0	5	127.0.0.1:953	0.0.0.0:*	users(({"named": "pid-991", fd: 45}))
LISTEN	0	5	127.0.0.1:953	0.0.0.0:*	users(({"named": "pid-991", fd: 45}))
LISTEN	0	10	127.0.0.1:153	0.0.0.0:*	users(({"named": "pid-991", fd: 42}))
LISTEN	0	10	127.0.0.1:153	0.0.0.0:*	users(({"named": "pid-991", fd: 42}))
LISTEN	0	10	127.0.0.1:153	0.0.0.0:*	users(({"named": "pid-991", fd: 42}))
LISTEN	0	5	127.0.0.1:161289	0.0.0.0:*	users(({"elance": "pid-892", fd: 4}))
LISTEN	0	2996	127.0.0.1:161289	0.0.0.0:*	users(({"elance": "pid-892", fd: 4}))
LISTEN	0	10	[::]:153	[::]:*	users(({"named": "pid-991", fd: 3}))
LISTEN	0	10	[::]:153	[::]:*	users(({"named": "pid-991", fd: 3}))
LISTEN	0	128	[::]:22	[::]:*	users(({"csdn": "pid-991", fd: 4}))
LISTEN	0	2996	[::]:111	[::]:*	users(({"rphbind": "pid-76", fd: 4}, {"system": "pid-1", fd: 36}))
LISTEN	0	10	[Fe80::250:5eff:fe2a:10ff%eth0]:53	[::]:*	users(({"named": "pid-991", fd: 42}))
LISTEN	0	10	[Fe80::250:5eff:fe2a:10ff%eth0]:53	[::]:*	users(({"named": "pid-991", fd: 42}))
LISTEN	0	5	[::]:953	[::]:*	users(({"named": "pid-991", fd: 45}))
LISTEN	0	5	[::]:953	[::]:*	users(({"named": "pid-991", fd: 45}))

- Ubuntu中的systemd-resolved服务为本地应用程序提供了网络名字解析服务，系统通过它对外进行dns请求

```
root@ubuntu2204:/etc$ ll resolv.conf
lrwxrwxrwx 1 root root 39 Aug 10 2023 resolv.conf ->
./run/systemd/resolve/stub-resolv.conf
# ubuntu中, resolv.conf是一个软连接, 指向/run/systemd/resolve/stub-resolv.conf
# 修改这个软链接的指向到/run/systemd/resolve/resolv.conf, 而不是systemd-reload

ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf

# 修改后, 则会和网卡上配置的DNS对齐
```

- 设置全局DNS

```
vim /etc/systemd/resolved.conf
...
DNS=223.5.5.5
```

## bind安装和配置

- 配置文件 /etc/bind/named.conf.default-zones

```
root@ubuntu2204:/etc$ cat /etc/bind/named.conf.default-zones
// prime the server with knowledge of the root servers
zone "." {
    type hint;
    file "/usr/share/dns/root.hints"; # 记录了根域的信息
};

// be authoritative for the localhost forward and reverse zones, and for
// broadcast zones as per RFC 1912

zone "localhost" {
    type master;
    file "/etc/bind/db.local";
};

zone "127.in-addr.arpa" {
    type master;
    file "/etc/bind/db.127";
};

zone "0.in-addr.arpa" {
    type master;
    file "/etc/bind/db.0";
};

zone "255.in-addr.arpa" {
    type master;
    file "/etc/bind/db.255";
};
```

- 主配置文件 /etc/bind/named.conf

```
// This is the primary configuration file for the BIND DNS server named.  
//  
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the  
// structure of BIND configuration files in Debian, *BEFORE* you customize  
// this configuration file.  
//  
// If you are just adding zones, please do that in /etc/bind/named.conf.local  
  
include "/etc/bind/named.conf.options"; # bind配置项  
include "/etc/bind/named.conf.local"; # 基本不用  
include "/etc/bind/named.conf.default-zones"; # 中间配置文件，该文件中定义了域名和具体解析规则文件的对应关系
```

- bind配置项文件 `/etc/bind/named.conf.options`

```
options {  
    # 此配置表示DNS服务只监听了本机127.0.0.1的53端口，如果对外提供DNS服务，可以将此行注释或改为any  
    listen-on port 53 { 127.0.0.1; };  
  
    # 监听IPV6的53端口，配置方法同上  
    listen-on-v6 port 53 { ::1; };  
  
    # 此配置表示仅本机客户以使用DNS服务的解析查询功能，如果对外提供DNS服务，可以将此行注释或者值改为any  
    allow-query { localhost; };  
  
    # 是否启用加密验证，在使用转发的时候，将此项改为no  
    dnssec-validation auto;  
  
    # 转发服务器  
    forwards { 10.0.0.207; };  
  
    forward first;  
}
```

## 加密安全

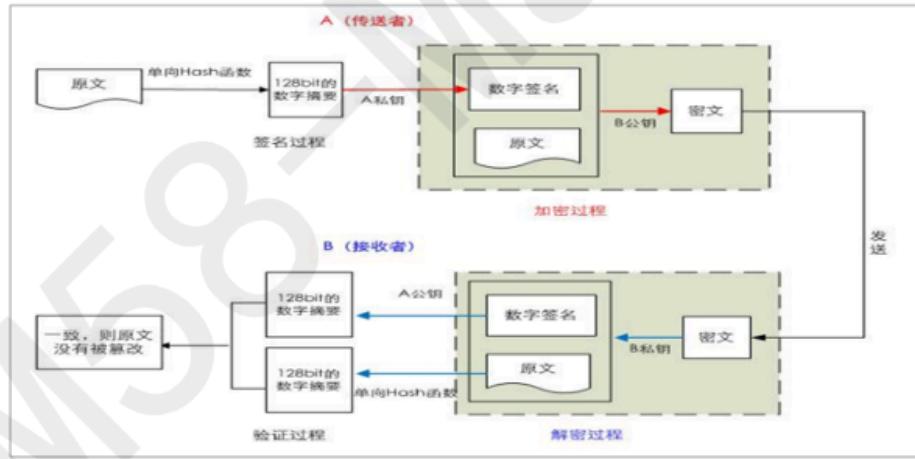
### 重点1：综合加密和签名（面试题）

- 签名的本质：即A使用私钥加密，B通过A的公钥去验证A的身份，同时里面的数据通过hash值保证数据的完整性

#### 1.7.4.3 综合加密和签名

即实现数据加密，又可以保证数据来源的可靠性、数据的完整性和一致性

##### 方法1: $Pb\{Sk[hash(data)]+data\}$



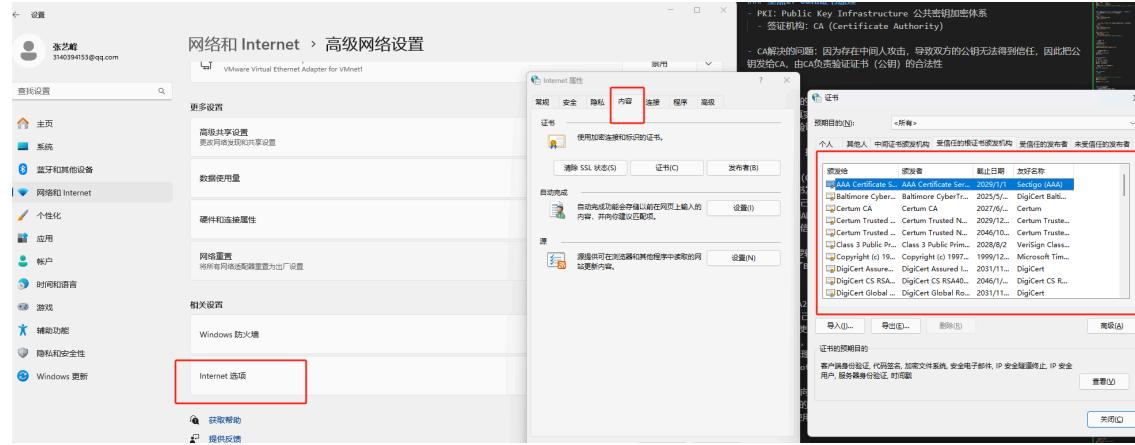
##### 方法2: 对称key $\{Sk[hash(data)]+data\}+Pb(\text{对称key})$



## 重点2：Ca和证书原理

- PKI: Public Key Infrastructure 公共密钥加密体系
  - 签证机构: CA (Certificate Authority)
- CA解决的问题: 因为存在中间人攻击，导致双方的公钥无法得到信任，因此把公钥发给CA，由CA负责验证证书(公钥)的合法性
- 过程:
  - A把自己的公钥发送给CA，(CA也有自己的公钥和私钥)，CA负责验证收到的公钥是否真实可靠，验证A的身份，
  - 如果CA验证A的身份没有问题，CA就会用自己的私钥进行签名，然后将其生成证书

- 签名：把A的公钥用CA的私钥进行签名，同时添加ca的信息，有效期等其他信息
- 签名 (CA私钥加密的A的公钥) + 其他信息，构成证书
- CA将证书发给A
- A得到自己的证书后，就可以通过网路将证书发送给B，此时B就得到了A的证书
  - B的到A的证书，目的就是安全可靠的得到A的公钥
- (前提B信任CA) B的到A的证书后，使用CA的公钥对证书进行解密，得到A的公钥
- 同样的逻辑，B向CA申请证书，然后将证书发给A，这样A就通过CA公钥解密证书，得到了B的公钥
- CA1 和 CA2两个子CA机构要向root CA即根CA机构，验证自己的身份
  - CA1将自己的公钥发给RootCA, RootCA也有自己的私钥和公钥
  - RootCA使用自己的私钥加密CA1的公钥，加上根CA的信息+有效期等信息，生成CA1的证书，发给CA1
  - CA2，同理，从RootCA那里得到CA2的证书（上面有RootCA私钥加密的CA2公钥的信息+RootCA信息+有效期等信息）
- CA1的用户向CA2的用户进行加密通讯，就需要互相得到对方的公钥，即CA2下的用户需要CA1的公钥，CA1下的用户需要CA2的公钥，此时需要向RootCA申请对方的证书，然后使用RootCA的公钥进行解密，得到对方的公钥
- 所有电脑上默认保存着根CA的公钥

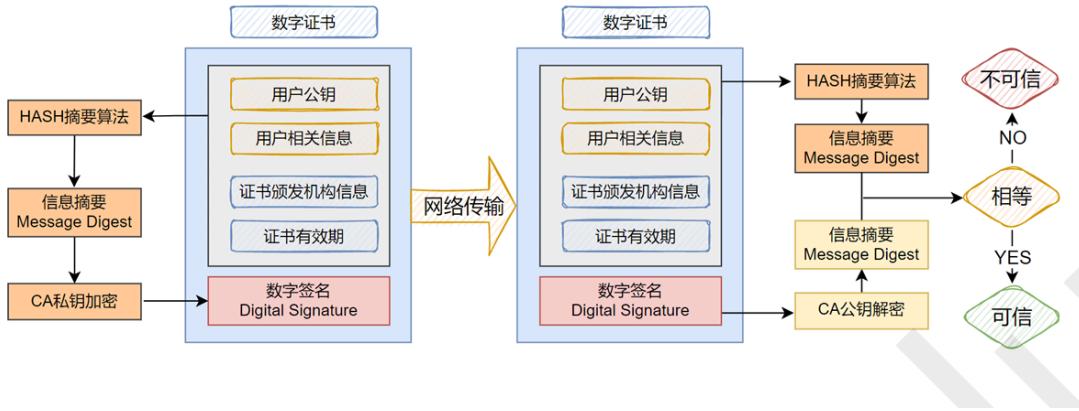


- 根CA自己给自己办发证书（自签名的证书）

`srootca(Prootca) + Prootca + expire + other`

- 为保证安全，根CA一般不联网
- CRL分发点上面记录着被吊销的证书

- 补充



- 数字证书上的信息

- 信息摘要：
  - 用户公钥
  - 用户相关信息
  - 证书颁发机构信息
  - 证书有效期等
- 数字签名（将上面的所有数据算出一个hash算法，算一个hash值，然后使用CA私钥加密）
  - 该hash值无法被中间人篡改，中间人只能篡改摘要信息，但是中间人更改了摘要信息，就会和数字签名中的信息不匹配，从而判定不可信
- 证书：里面有PK，可读不可改，并且不可伪造

### 重点3：https简化原理

- https单向认证，访问者要验证网站，而网站不能验证访问者
- 整个过程

1. 客户端发起HTTPS请求

用户在浏览器里面输入一个https网址，然后连接到服务器的443端口

2. 服务端的配置

采用HTTPS协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己办发的证书要客户端验证通过，才能继续访问，而使用受信任的公司申请的证书则不会弹出提示页面，这套证书起始就是一对公钥和私钥

3. 传送服务器的证书给客户端

证书里其实是公钥，并且还包含很多的信息，比如证书的颁发机构，过期时间等

4. 客户端解析验证服务器证书

这部分工作是由客户端的TLS来完成，首先会验证公钥是否有效，比如：颁发机构，过期时间等，如果发现异常，则会弹出一个提示，提示证书有问题，如果证书没问题，就会生成一个随机值，然后用证书的公钥对该随机值进行非对称加密

5. 客户端加密信息传送服务器

这部分传送使用证书加密后的随机值，目的就是让服务器得到这个随机值，以后客户端和服务器的通信就可以通过这个随机值进行对称加密

- 总结:
  - 就是客户端向服务端发起请求
  - 服务端响应客户端并将证书发给客户端
  - 客户端验证证书，如果验证没有问题，则产生一个随机值，并用公钥将其加密，并返回给服务端
  - 服务端使用私钥将其解密，得到这个随机值，后续的通信，就可以用这个随机值作为密钥，进行对称加密
  - 对称加密的好处就是简单高效，效率高，性能好

## base64编码原理

- 一个ascii字符8bit
- base64是将其进行6bit分割，然后查询base64的字符表，进行重组，如果最后不足6bit，每2bit用一个=补充

## 重点4： openssl命令

### openssl单向哈希加密（摘要算法）

- 查看所有摘要命令

```
openssl list -digest-commands
```

- 使用md5计算文件摘要值

```
openssl md5 fstab

openssl dgst -md5 fstab

md5sum fstab

openssl dgst fstab 默认sha256
```

- openssl passwd 生成用户密码

### openssl命令

- 查看openssl的版本

```
openssl version
```

### openssl对称加密

- 工具：`openssl enc, gpg`
- 算法：`3des, aes, blowfish, twofish`
- enc命令帮助：`man enc`

```
# 加密
openssl enc -e -des3 -a -salt -in testfile -out testfile.cipher
# -e 加密
# -des3 对称加密算法，表示使用三重des加密（加密三次）
# -a 使用base64输出可见字符
# -salt 加盐（随机生成，默认）
# - in 后面加要处理的文件
# -out 将加密后的信息输出到指定文件

# 解密
openssl enc -d -des3 -a -salt -in testfile.cipher -out testfile
```

## openssl命令单向加密

- 工具: openssl dgst
- 算法: md5sum, sha1sum, sha224sum, sha256sum

```
openssl dgst -md5 /PATH/FILENAME
md5sum /PATH/FILE
```

## openssl生成用户密码

```
openssl passwd -1 -salt SALT(最多8位)
openssl passwd -6 -salt SALT value
```

## openssl 生成随机数

- 随机数生成器: 伪随机数字, 块终中断生成随机数
- 调用: /dev/random; /dev/urandom

```
openssl rand -base64 8
# 8 表示8个字节

openssl rand -base64 10 | head -c10
```

## openssl命令实现PKI

- 公钥加密
  - 算法: RSA, ELGama
  - 工具: gpg, openssl rsautl
- 生成私钥

```
openssl genrsa -out /PATH/TO/FILENAME [-des3 ] NUM_BITS # (默认2048位)
genrsa 既能实现加密也能实现数字签名, gendsa只支持数字签名, genrsa更通用
```

#示例

```
# 生成对称密钥加密的私钥
(umask 077; openssl genrsa -out test.key -des3 2048) # genrsa就是用rsa算法生成的私钥
# 生成一个 2048 位的 RSA 私钥, 并使用 3DES 对该私钥进行加密

# 将加密对称密钥key解密
openssl rsa -in test.key -out test2.key

# 从私钥中提取出公钥
openssl rsa -in PRIVATEKEYFILE -pubout -out PUBLICKEYFILE
```

## 建立私有CA实现证书申请颁发

- 建立私有CA
  - OpenCA: 开源软件
  - openssl: openssl和openssl-libs两个包
- 证书申请及签署步骤
  - 生成私钥
  - 生成申请请求
  - RA核验
  - CA签署
  - 获取证书
- openssl的配置文件

```
# rocky
# openssl和openssl-libs
/etc/pki/tls/openssl.cnf

# ubuntu
# openssl和libssl-dev
/etc/ssl/openssl.cnf
```

- 三种策略

- match匹配、optional可选、supplied提供
- match: 要求申请填写的信息跟CA设置信息必须一致
- optional: 可有可无, 跟CA设置信息可以不一致
- supplied: 必须填写这项申请信息

```
cat /etc/pki/tls/openssl.cnf

...
# 一个设备可以创建多个CA, 下面的是默认CA
[ ca ]
default_ca = CA_default      # the default ca section
```

```

[ ca_default ]
dir = /etc/pki/CA          # 所有创建CA相关文件都放在这里，如果该目录不存在，则需要手工创建
certs = $dir/certs          # 存放发布的证书
crl_dir = $dir/crl          # 证书的吊销列表
database = $dir/index.txt   # 需要手工创建

new_certs_dir = $dir/newcerts      # 默认颁发的新证书放在这里
certificate = $dir/cacert.pem     # 存放CA的证书（RootCA，自签名）
serial = $dir/serial            # 证书的当前序号，下一个颁发证书的编号
crlnumber = $dir/crlnumber       # 吊销的编号

crl = $dir/crl.pem           # 吊销列表
private_key = $dir/private/cakey.pem # 自己的私钥
RANDFILE = $dir/private/.rand

x509_extensions = usr_cert

name_opt = ca_default
cert_opt = ca_default

default_days = 365             # 默认有效期
default_crl_days = 30
default_md = sha256
preserve = no

policy = policy_match          # 证书颁发策略

[ policy_match ]               # match意味着，CA信息和客户端向CA申请的信息必须一致
countryName = match
stateOrProvinceName = match
OrganizationName = match
OrganizationalUnitName = optional
commonName = supplied          # 通常是网站域名
emailaddress = optional

[ policy_match ]
countryName = optional
stateOrProvinceName = optional
OrganizationName = optional
OrganizationalUnitName = optional
commonName = supplied          # 通常是网站域名
emailaddress = optional

```

## 创建私有CA

- 创建CA所需的文件

```

# 生成证书索引数据库文件
touch /etc/pki/CA/index.txt

# 指定第一个颁发证书的系列号
echo 01 > /etc/pki/CA/serial

```

- 生成CA私钥

```
cd /etc/pki/CA/  
  
(umask 066; openssl genrsa -out private/cakey.pem 2048) # 文件名要和配置文件对上
```

- 生成CA自签名证书

```
openssl req -new -x509 -key /etc/pki/CA/private/cakey.pem -days 3650 -out  
/etc/pki/CA/cacert.pem
```

# -new: 生成新证书签署请求  
# -x509: 专用于CA生成自签证书  
# -key: 生成请求时用到的私钥文件  
# -days: 证书有效期  
# -out: 证书的保存路径

- 国家代码: <https://country-code.cl/> (中国CN)
- 以文本形式查看证书内容

```
openssl x509 -in /etc/pki/CA/cacert.pem -noout -text
```

## 申请证书并颁发证书

- 为需要使用证书的主机生成私钥

```
(umask 066; openssl genrsa -out /data/test.key 2048)
```

- 为需要使用证书的主机生成证书申请文件

```
openssl req -new -key /data/test.key -out /data/test.csr
```

使用标准输入重定向来非交互式生成申请证书

```
openssl req -new -key /data/test.key -out /data/test.csr << EOF  
CN  
beijing  
beijing  
magedu  
it  
...  
EOF
```

- 在CA签署证书并将证书颁发给请求者

```
openssl ca -in /tmp/test.csr -out /etc/pki/CA/certs/test.crt -days 100
```

# 注意: 默认要求国家, 省, 公司名称三项必须和CA一致

## 查看/验证证书

验证指定编号证书的有效性

```
openssl ca -status SERIAL
```

## 最后将证书相关文件发送到用户端使用

- 默认不允许同一个用户申请多个证书
- 可以通过修改 /etc/pki/CA/index.txt.attr

```
# 修改配置选项
```

```
unique_subject = no      #原本是yes，仅允许唯一的目标，改为no则可以允许，同一用户申请多个证书
```

## 吊销证书

- 在客户端获取要吊销的证书的serial

```
openssl x509 -in /PATH/FROM/CERT_FILE -noout -serial -subject
```

- 在CA上，根据客户提交的serial与subject信息，对比验证是否与index.txt文件中的信息一致，吊销证书

```
openssl ca -revoke /etc/pki/CA/newcerts/SERIAL.pem
```

- 指定第一个吊销证书的编号，注意：第一次更新证书吊销列表前，才需要执行

```
echo 01 > /etc/pki/CA/crlnumber  
# /etc/pki/CA/crlnumber,手动创建，是吊销证书的编号文件
```

- 更新证书吊销列表

```
openssl ca -gencrl -out /etc/pki/CA/crl.pem
```

```
# 将此文件发送到windows，将后缀改为crl.pem.crl
```

- 查看crl文件

```
openssl crl -in /etc/pki/CA/crl.pem -noout -text
```

## 作业：将申请证书的过程脚本自动化

### Ubuntu建立私有CA

#### 安装包与配置文件

```
apt install libssl-dev
```

```
# 配置文件路径
```

```
cat /etc/ssl/openssl.cnf
```

## 创建私有CA

- 创建相关配置文件

```
mkdir -pv /etc/pki/CA/{certs,crl,newcerts,private}
```

- 生成CA私钥

```
cd /etc/pki/CA  
openssl genrsa -out private/cakey.pem 2048
```

- 生成CA自签名证书

```
openssl req -new -x509 -key /etc/pki/CA/private/cakey.pem -days 1000 -out  
/etc/pki/CA/cacert.pem
```

- 查看证书

```
openssl x509 -in /etc/pki/CA/cacert.pem -noout -text
```

## 时间同步服务

- 多主机协作工作时，各个主机的时间同步很重要，时间不一致会造成很多重要应用的故障，如：加密协议，日志，集群等。
- 利用NTP（Network Time Protocol）协议使各个网络中的各个计算机时间达到同步
- 目前NTP协议属于运维基础架构中必备的基本服务之一。

## CentOS7中实现时间同步

```
# 可用定时任务实现周期性对钟  
crontab -e  
  
*/1 * * * * /usr/sbin/ntpdate ntp.aliyun.com #每个一分钟，进行一次时间同步
```

## chrony介绍

- 实现NTP协议的自由软件
- 可使系统时钟与NTP服务器，参考时钟（例如GPS接收器）以及使用手表和键盘的手动输入进行同步
- 还可以作为NTPv4（RFC 5905）服务器和对等体运行，为网络中的计算机提供时间服务
- 设计用于在各种条件下良好运行，包括间歇性和高度拥挤的网络连接，温度变化（计算机时钟对温度敏感），以及不能连续运行或在虚拟机上运行的系统。
- CentOS8中以后得源中，只有chrony，且默认已经安装

## chrony的优势

- 更快的同步只需要数分钟而非数小时时间，从而最大程度减少了时间和频率误差，对于并非全天24小时运行的虚拟计算机而言非常有用，能够更好地响应时钟频率的快速变化，对于具备了不稳定时钟的虚拟机或导致时钟频率发生变化的节能技术而言非常有用
- 在初始同步后，它不会停止时钟，以防对需要系统时间保持单调的应用程序造成影响
- 在应对临时非对称延迟时（例如，在大规模下载造成链接饱和时）提供了更好的稳定性
- 无需对服务器进行定期轮询，因此具备间歇性网络连接的系统仍然可以快速同步时钟
- 官方文档

```
https://chrony.tuxfamily.org/
```

# 官方网站

```
https://.chrony.tuxfamily.org/documentation.html/
```

# 官方文档

## ntp和chrony的区别

- ntp：渐进式，一点点追，直到最终同步
- chrony：一开始，直接同步，后续如果出现时间不同步，则采用渐进式方式同步

## chrony文件组成

- 软件包

```
rpm -q chrony  
chrony-4.1-1.e18.x86_64
```

```
# Ubuntu2204要自己下载  
apt install -y chrony
```

- 可执行程序

```
# 命令行用户工具，用于监控性能并进行多样化的配置。他可以在chrony实例控制的计算机上工作，也可在一台不同的远程计算机工作
```

```
11 /usr/bin/chronyc
```

```
# 后台运行的守护进程，用于调整内核中运行的系统时钟和时间服务器同步。它确定计算机增减时间的比率，并对此进行补偿
```

```
11 /usr/bin/chronyd
```

- 服务unit文件

```
systemctl status chronyd.service
```

- 配置文件

```
root@ubuntu2204:~$tree /etc/chrony
/etc/chrony
├── chrony.conf      # 主配置文件
├── chrony.keys
├── conf.d
│   └── README
└── sources.d
    └── README

2 directories, 4 files
```

- 监听端口

```
服务端: 123/udp      # 其他机器通过此端口连接本机, 将本机当做ntp服务器
客户端: 323/udp      # 本机通过此端口同步时间
```

## 配置文件说明

<https://chrony.tuxfamily.org/doc/3.5/chrony.conf.html>

man chrony.conf

## 主配置文件内容与解读

```
# This will use (up to):
# - 4 sources from ntp.ubuntu.com which some are ipv6 enabled
# - 2 sources from 2.ubuntu.pool.ntp.org which is ipv6 enabled as well
# - 1 source from [01].ubuntu.pool.ntp.org each (ipv4 only atm)
# This means by default, up to 6 dual-stack and up to 2 additional IPv4-only
# sources will be used.
# At the same time it retains some protection against one of the entries being
# down (compare to just using one of the lines). See (LP: #1754358) for the
# discussion.
#
# About using servers from the NTP Pool Project in general see (LP: #104525).
# Approved by Ubuntu Technical Board on 2011-02-08.
# See http://www.pool.ntp.org/join.html for more information.
pool ntp.ubuntu.com      iburst maxsources 4      # 指同步的服务器在哪
pool 0.ubuntu.pool.ntp.org iburst maxsources 1
pool 1.ubuntu.pool.ntp.org iburst maxsources 1
pool 2.ubuntu.pool.ntp.org iburst maxsources 2

# maxsources 4 详解
# 作用:
# 指定最大源数: maxsources 定义了从指定的池 (例如 ntp.ubuntu.com) 中最多可以使用的时间源
# (NTP 服务器) 数量。
# 控制同步的时间源: 在 NTP 中, 一个池 (pool) 可能包含许多 NTP 服务器, 例如 pool.ntp.org 通常
# 包含全球范围内的多个服务器。maxsources 选项可以限制从这个池中选择的服务器数量。

# 为什么需要 maxsources?
# 减少资源消耗:
# 如果不限制 maxsources, 系统可能会尝试与池中的许多服务器同步, 增加网络负载和资源消耗。
# 限制服务器数量有助于平衡网络流量和同步精度。
```

```
# 提高同步效率:  
# NTP 通常需要与多个服务器比较时间，选择最稳定的时间源。选择 3-5 个时间源通常已足够提供高精度的  
同步。  
# 避免冗余:  
# 对于单个客户端，使用过多的时间源不会显著提高同步精度，但会增加不必要的查询和流量。  
  
# 默认行为：如果没有指定 maxsources，系统会根据默认策略决定使用多少个服务器：  
# 大多数情况下，默认值是 10。  
# 这意味着系统可能会尝试与池中的多达 10 个服务器建立连接。  
  
# Use time sources from DHCP.  
sourcedir /run/chrony-dhcp  
  
# Use NTP sources found in /etc/chrony/sources.d.  
sourcedir /etc/chrony/sources.d  
  
# This directive specify the location of the file containing ID/key pairs for  
# NTP authentication.  
keyfile /etc/chrony/chrony.keys  
  
# This directive specify the file into which chronyd will store the rate  
# information.  
driftfile /var/lib/chrony/chrony.drift  
  
# Save NTS keys and cookies.  
ntsdumpdir /var/lib/chrony  
  
# Uncomment the following line to turn logging on.  
#log tracking measurements statistics  
  
# Log files location.  
logdir /var/log/chrony  
  
# Stop bad estimates upsetting machine clock.  
maxupdateskew 100.0  
  
# This directive enables kernel synchronisation (every 11 minutes) of the  
# real-time clock. Note that it can't be used along with the 'rtcsync' directive.  
rtcsync  
  
# Step the system clock instead of slewing it if the adjustment is larger than  
# one second, but only in the first three clock updates.  
makestep 1 3  
  
# Get TAI-UTC offset and leap seconds from the system tz database.  
# This directive must be commented out when using time sources serving  
# leap-smeared time.  
leapsectz right/UTC
```

- 常用字段

**server:** 时钟服务器地址，加burst选项表示服务可用时，一次发送八个数据包，包间隔通常为2秒，可加快初始同步速度（客户端配置）

**pool:** 语法和指令与server字段相似，不同之处在于其指定的NTP服务可以解析多个地址

**allow:** 指定可以使用本机服务的设备，格式可以是IP，子网，网段（服务端配置）

**deny:** 指定不可以使用本机服务的设备，格式可以是IP，子网，网段

**local stratum 10:** 即使server指令中时间服务器不可用，也允许将本地时间作为标准时间授时给其他客户端（服务端配置）

## Chrony客户端工具

- chronyc 可以运行在交互式和非交互式两种方式(c 表示client)
- 交互式客户端有以下常用子命令

```
chronyc
# tracking指令
chronyc> tracking
Reference ID      : CA760151 (time.neu.edu.cn)          # 当前同步的NTP服务器ID
和IP地址
Stratum           : 2                                # 层次，跳数
Ref time (UTC)   : Mon Sep 02 14:47:42 2024        # 源最后一次获取到的UTC
时间
System time      : 0.001897034 seconds slow of NTP time    # 当前系统时间与NTP服务
时间的偏移量
Last offset      : -0.000424999 seconds
RMS offset       : 0.002490461 seconds
Frequency        : 50.877 ppm fast
Residual freq   : -0.401 ppm
Skew              : 5.828 ppm
Root delay       : 0.047850452 seconds
Root dispersion  : 0.001142370 seconds
Update interval  : 64.7 seconds                      # 最近两次时钟更新的间隔
Leap status       : Normal

# 扩展:NTP的分层结构
# 1. Stratum 0 (参考时钟)：
# 这是最顶层的时间源，通常是非常精确的硬件时钟（如 GPS 接收器、原子钟或高精度的参考时钟）。
# 它不能直接通过网络访问

# 2. Stratum 1 (直接连接的服务器)：
# 这些服务器直接连接到 Stratum 0 的参考时钟设备，并通过它们提供时间。
# 它们是网络上最可靠的时间服务器。

# 3. Stratum 2 (客户端服务器)：
# stratum 2 的服务器从 Stratum 1 的服务器获取时间。
# 它们也可以向 Stratum 3 的客户端提供时间。
# 您的系统是一个 Stratum 2 设备。

# 4. Stratum N (层级 N 的设备)：
# 每向下同步一级，层级数字加一。例如，Stratum 3 从 Stratum 2 获取时间，Stratum 4 从
Stratum 3 获取时间，以此类推。
```

```

# 列出当前可用的ntp服务器
chronyc> activity
200 OK
8 sources online
# 之所以是8个sources online是因为在配置文件中
# pool ntp.ubuntu.com          iburst maxsources 4
# pool 0.ubuntu.pool.ntp.org iburst maxsources 1
# pool 1.ubuntu.pool.ntp.org iburst maxsources 1
# pool 2.ubuntu.pool.ntp.org iburst maxsources 2
# 后面的数字加起来: 4+1+1+2的最终结果是8
0 sources offline
0 sources doing burst (return to online)
0 sources doing burst (return to offline)
0 sources with unknown address

# 列出所有的源地址
chronyc> sourcestats
Name/IP Address          NP   NR   Span   Frequency   Freq Skew   offset   Std Dev
=====
prod-ntp-4.ntp1.ps5.cano> 13    5    721    -2.317    17.508   -47ms  3863us
prod-ntp-3.ntp4.ps5.cano>  6    4    326    -22.111   178.062   -52ms  4569us
prod-ntp-5.ntp4.ps5.cano> 18    9    916    -0.166    18.186   -47ms  5897us
alphyn.canonical.com      17   10    911    -6.786    6.430   -8228us 2128us
ntp1.flashdance.cx        17   13    918    -3.355    3.658   -12ms  1028us
time.neu.edu.cn            18   7    913    -0.464    2.900   +2857us 915us
time.neu.edu.cn            9    6    520    +1.657    7.417   +3094us 636us
119.28.183.184           18   10    916    -1.687    5.594   -13ms  1589us

```

MS Name/IP address	Stratum	Poll	Reach	LastRx	Last sample
^- prod-ntp-3.ntp1.ps5.cano>	2	9	377	280	-45ms[-45ms] +/- 120ms
^- prod-ntp-5.ntp1.ps5.cano>	2	9	377	473	-36ms[-36ms] +/- 110ms
^- prod-ntp-4.ntp1.ps5.cano>	2	9	377	470	-40ms[-40ms] +/- 116ms
^- alphyn.canonical.com	2	9	277	343	-18ms[-17ms] +/- 140ms
^+ 111.230.189.174	2	9	277	214	-5747us[-5747us] +/- 48ms
^* time.neu.edu.cn	1	9	377	215	+11ms[+12ms] +/- 25ms
^- ntp1.flashdance.cx	2	8	377	281	-3051us[-2802us] +/- 78ms
^+ 139.199.215.251	2	7	377	19	-6593us[-6593us] +/- 65ms

# M NTP源, ^表示服务器, =表示二级时钟, # 表示本地时钟  
# S NTP源状态, \*此源已同步, +可接收的源, -合并算法排除的可接受的源, ? 没连接的源, x认为该源有错, ~不确定的源  
# Name / IP address NTP服务器主机名后或ip地址或refid值  
# Stratum 层次, 跳数, 1.表示本地时钟, 2.表示通过第一层级的服务器实现同步, 依次类推  
# Poll NTP源的轮询频率, 以秒为单位, 值为基数2的对数, 6表示64秒进行一次同步  
# REACH 略  
# LASTRX 多久前从源收到最后一次数据, 默认单位是秒  
# Last sample 上次同步时NTP服务器与本地时间的偏移量, 调整后偏移量与实际测量中的误差值, +表示正偏移, 本地块

- 修改配置文件

```
vim /etc/chrony/chrony.conf
```

```
# 注释pool行
```

```
# 添加下列行
```

```
server ntp.aliyun.com iburst (为高可用性, 通常至少写两条)
```

```
# 详解iburst和burst
```

```
# 启用 iburst 的效果
```

```
# 1. 没有 iburst 时:
```

```
# 系统在启动后会每隔 64 秒发送一次时间同步请求, 直到成功。
```

```
# 如果网络条件不佳, 初次同步可能会很慢。
```

```
# 2. 启用 iburst 时:
```

```
# 系统在启动时会立即向 NTP 服务器快速发送 4 个同步请求包。
```

```
# 如果服务器响应正常, 初次同步可以在几秒钟内完成。
```

```
# iburst
```

```
# 仅在初次尝试同步时生效。
```

```
# 目的是加快初次时间同步
```

```
# 不会对服务器造成过多压力。
```

```
# burst
```

```
# 在每次同步时都发送多次请求
```

```
# 适用于高精度时间同步, 但对服务器压力较大。
```

```
#
```

```
# 通常推荐使用 iburst, 而非 burst。
```

```
# 重启服务
```

```
sudo systemctl restart chronyd.service
```

- chrony是渐进式同步, 如果差距过大, 想立即同步完成, 则可以重启服务

```
date +%F-%T
```

```
sudo systemctl restart chronyd.service
```

## 常用公共NTP服务

ntp pool	cn.pool.ntp.org, 0-3.cn.pool.ntp.org
阿里云公共NTP	ntp.aliyun.com, ntp1-7.aliyun.com
time.pool.aliyun.com(windows)	
腾讯公共NTP	time1-5.cloud.tencent.com
北京邮电大学NTP	s1a.time.edu.cn
清华大学NTP	s1b.time.edu.cn
北京大学NTP	s1c.time.edu.cn
国家授时中心服务器	210.72.145.44
美国标准技术院	time.nist.gov

## 使用timedatectl开关chrony.service

```
timedatectl set-ntp 0  
timedatectl set-ntp 1
```

## 实现私有时间服务

- 在同一网络内，如果有多个需要进行时间同步的服务器，则我们可以在内网自建NTP Server，这样可以节约访问外网的网路资源；另一方面，如果外网不可用，则至少可以保证，内网的NTP服务还是可用的
- 服务端配置

```
vim /etc/chrony/chrony.conf  
  
allow 10.0.0.0/24          # 允许10.0.0.0网段的主机将本机作为时间同步服务器  
local stratum 10           # 允许本机在不能与外网同步的情况下，还能提供服务  
  
# 重启服务  
systemctl restart chrony.service  
  
# 关闭防火墙  
systemctl stop firewalld.service
```

- chronyd
  - 服务端监听使用123端口
  - 客户端访问使用323端口
- 客户端配置

```
# 添加server，生产环境下至少两台，保证高可用  
vim /etc/chrony.conf  
server 10.0.0.206 iburst    # 之前配置的服务端NTP的IP  
  
重启服务  
systemctl restart chrony
```

## 日志服务管理

### rsyslog系统日志服务

- RSYSLOG is the rocket-fast system for log processing
- rsyslog是CentOS6以后的版本中使用的日志管理程序，是一个默认安装的服务，并且默认开机启动
- rsyslog特性
  - 支持输出日志到各种数据库，如MySQL, PostgreSQL, MongoDB ElashticSearch，实现使用第三方服务对日志进行存储和分析；
  - 精细的输出格式控制以及对日志内容的强大过滤能力，可以实现过滤日志信息中的指定部分；
  - 通过RELP+TCP实现数据的可靠传输
  - 支持数据的加密和压缩传输等

# ELK日志服务

- ELK是三个开源软件的缩写，分别表示：Elasticsearch, Logstash, Kibana，它们都是开源软件，在后续发展的过程中增加了一个FileBeat，这几款软件通常在一起配合使用，各司其职。ELK主要用于部署在企业架构中，收集多台设备上多个服务的日志信息，并将其统一整合提供给用户。我们后面的课程中会有专门的章节讲解ELK服务
- rsyslog日志服务与ELK日志服务的区别
  - rsyslog主要用于单机日志管理，ELK主要用于分布式集群环境中的日志管理。

## rsyslog日志管理

### 系统日志相关概念

- facility: 设施，从功能或程序上对日志进行归类
  - 在一台主机上会同时运行多个服务和软件，每个服务或软件都有可能会产生大量日志，如果每个服务或软件产生的日志都独立存放管理，那文件数量就太多了，如果能放到一个文件中，似乎也不是很合适，所以rsyslog将日志进行了分类，相同的类型的日志放一个文件，这样便于管理

```
# syslog内置分类
LOG_AUTH          # auth 安全和认证相关的日志
LOG_AUTHPRIV      # authpriv 安全和认证相关的日志，私有
LOG_CRON          # cron 系统定时任务，crontab与at产生的相关日志
LOG_DAEMON        # daemon 各守护进程产生的日志
LOG_FTP            # ftp, ftpd 守护进程产生的日志
LOG_KERN           # 内核产生的日志
LOG_LOCAL_0 -- LOG_LOCAL_7 # local 0-7 自定义分类
LOG_LPR             # LOG_LPR lpr 打印服务日志
LOG_MAIL           # mail 邮件服务日志
LOG_NEWS           # 网络新闻服务产生的日志
LOG_SYSLOG         # 服务自己的日志
LOG_USER           # 用户登记
LOG_UUCP           # uucp子系统的日志信息
*                  # 通配符，代表所有分类
```

- priority: 优先级别，从高到低排序
  - rsyslog在记录日志的时候，将各种产生日志的事件和行为进行了优先级的排序，使用者可以根据不同环境和需求，设置不同的级别来记录日志，这样可以保证，记录下来的内容都是我们想要的。

```
# syslog 内置优先级分类，从高到低，如果在记录日志时，设置了优先级，则只会记录设定的优先级和高于  
设定优先级的日志  
LOG_EMERG          # emerg/panic 紧急，致命错误  
LOG_ALERT           # alert 告警，当前状态必须立即进行纠正  
LOG_CRIT            # crit 关键状态的警告，如果硬件故障  
LOG_ERR              # err/error 其他错误  
LOG_WARNING          # warning/warn 警告级别的信息  
LOG_NOTICE           # notice 通知级别的信息  
LOG_INFO             # info 通告级别的信息  
LOG_DEBUG             # debug 调试程序时的信息  
*                   # 所有级别的日志  
none                # 不需要任何日志
```

## rsyslog服务组成

```
# 查看所有rsyslog包中的文件  
dpkg -L rsyslog  
  
/usr/sbin/rsyslogd      # 主程序  
/lib/systemd/system/rsyslog.service    # 服务脚本  
/etc/rsyslog.conf        # 主配置文件  
/etc/rsyslog.d/*.conf    # 配置文件目录中的配置文件  
/usr/lib/x86_64-linux-gnu/rsyslog/*.so   # 库文件
```

## rsyslog 配置文件

```
/etc/rsyslog.conf  
  
ls /etc/rsyslog.d  
# 20-ufw.conf 21-cloudinit.conf 50-default.conf
```

- 配置文件解析

```
# 在主配置文件中，将配置分为三个部分，分别是模块（MODULES），全局配置（GLOBAL DIRECTIVES），  
日志记录规则配置（RULES）  
/etc/rsyslog.conf
```

```
# ubuntu系统中，默认rule规则是单独放在一个文件中  
# Default logging rules can be found in /etc/rsyslog.d/50-default.conf  
/etc/rsyslog.d/50-default.conf
```

```
# rsyslog在安装是有很多支持模块，但默认不是所有模块都开启，写在此处即可
```

---

```
# 第一部分：MODULES模块  
不常用，略
```

---

```
# 第二部分：GLOBAL DIRECTIVES 全局配置  
  
# 默认日志模版  
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
```

```

# 默认开启重复过滤
$RepeatedMsgReduction on

# 关于重复过滤的示例
[root@ubuntu2204 ~]#logger "hello"
[root@ubuntu2204 ~]#logger "hello"
[root@ubuntu2204 ~]#logger "hello"
[root@ubuntu2204 ~]#logger "hello"
[root@ubuntu2204 ~]#logger "hello-m8"

# syslog上的记录如下
Aug 18 22:36:42 ubuntu2204 root: hello
# 因为开启了重复过滤，因此仅记录了次数
Aug 18 22:37:22 ubuntu2204 root: message repeated 3 times: [ hello]
Aug 18 22:37:30 ubuntu2204 root: hello-m8

# 创建日志文件的默认权限和属主属组
$FileOwner      syslog
$FileGroup      adm
$FileCreateMode 0640
$DirCreateMode  0755
$Umask          0022
$PrivDropToUser syslog
$PrivDropToGroup syslog

# 工作目录，默认目录为空
$workDirectory /var/spool/rsyslog

# 独立配置文件引用目录
#IncludeConfig /etc/rsyslog.d/*. conf

```

- rule 配置规则
  - 每一行rule由两列组成，分别是选择器和处理动作，选择器将过滤后的日志交由处理动作处理
  - 选择器可以同时有多个，用分号;分隔，处理动作也可以同时有多个，用&分隔，处理动作中可以指定模版，不同的模版会生成不同的日志内容，模版可以自定义
  - 总结：
    - 选择器selector: 分号;分隔
    - 处理动作action: and符 & 分隔
    - 设施和优先级: 逗号,分隔
  - 选择器可以有以下几种定义方式
    - 用分类和优先级来过滤，同一条rule中，分类和优先级都可以有多个，用逗号分隔
    - 基于日志内容中的指定字段过滤
    - 基于表达式构建脚本来过滤

```

# 方式1:
# selector - 分类和优先级
mail.err        /var/log/mail.err

# 就是mail分类中，优先级err或以上的所有日志记录，输出给/var/log/mail.err

```

```
# facility.priority[...] action
# =priority      # 仅处理指定级别日志，即该级别往上的日志不再记录
```

方式2：：

```
# selector - 基于日志内容指定字段过滤
# 字段, 比较表达式, 要比较的值
:property, [!]compare-operation, "value" action
# 示例
:msg,contains,"[UFW" /var/log/ufw.log
```

## property: 预定义属性

**msg**

含义：完整的日志消息文本

示例：将"error"的日志消息吸入"error.log"文件

```
:msg, contains, "error" /var/log/error.log
```

**hostname**

含义：产生日志的主机名

示例：将来主机名为"webserver"的日志消息写入"webserver.log"文件

```
:hostname, isEqual, "webserver" /var/log/webserver.log
```

**fromhost-ip**

含义：日志消息来源的IP地址

示例：将来自特定IP地址的日志消息写入"network.log"文件

```
:fromhost-ip, isEqual, "192.168.1.1" /var/log/network.log
```

**syslogfacility**

含义：Syslog设施码，拥有标识日志消息的来源(例如：内核，邮件系统等)

示例：将所有邮件相关的日志消息写入"mail.log"文件

```
:syslogfacility-text, isEqual, "mail" /var/log/mail.log
```

**syslogseverity**

含义：Syslog严重性级别(severity)，表示日志消息的重要性级别

将严重性为"alert"的日志消息写入"alert.log"文件

```
:syslogseverity-text, isEqual, "alert" /var/log/alert.log
```

**programname**

含义：生成日志消息的程序名

示例：将由"sshd"生成的所有日志消息写入"sshd.log"文件

```
:programname, isEqual, "sshd" /var/log/sshd.log
```

**pri-text**

含义：日志消息的优先级，以文本形式表示(结合facility和severity)

示例：将优先级为"kern.crit"的日志消息写入"critical.log"文件

```
:pri-text, isEqual, "kern.crit" /var/log/critical.log
```

**timestamp**

含义：日志消息的时间戳。

示例：将时间戳包含特定日期的日志消息写入`dated\_logs.log`文件。

```
:timestamp, contains, "2024-08-16" /var/log/dated_logs.log
```

**app-name**

含义：应用名称，一般表示生成日志的应用程序的名称

示例：将特定应用程序的日志写入"app.log"文件

```
:app-name, isEqual, "myapp" /var/log/app.log
```

#### syslogtag

描述：包含了日志条目的应用程序名和进程号（可选）。

用途：用于根据应用程序将日志分流到不同文件。

#### # 比较表达式

# contains	字段是否包含value的内容
# contains_i	字段是否包含value的内容，不区分大小写
# isEqual	字段的内容是否与value相等
# startswith	是否以value开头
# startswith_i	是否以value开头，不区分大小写
# regex	基本正则
# ereregex	扩展正则
# isEmpty	是否为空，不需要后面的value

#### # selector - 基于rainerscript设置更复杂的日志过滤

<https://www.rsyslog.com/doc/v8-stable/rainerscript/index.html>

# 不常用，有点复杂，后续有空再看

### ◦ 处理动作有以下几种

- 输出到日志文件或某个特定设备
- 保存到数据库
- 发送到指定用户，该用户必须已登录，可以同时指定多个用户，用逗号分隔
- 传送到远程主机
- 通过管道传送给其他命令
- 丢弃，不处理 /dev/null

#### # action - 输出到日志文件或设备

```
/path/file      # 将日志内容写到指定文件  
-/path/file     # 将日志内容写到指定文件，异步写入，指日志先放入缓冲区，待指定时间或指定大小到达后，再落盘
```

# 异步写入的优缺点：

## 这种方式可以提高性能，尤其是在处理大量日志时，因为它减少了磁盘 I/O 的频率。但也存在风险，如果系统崩溃或断电，缓冲区中未写入磁盘的日志可能会丢失。

```
/dev/null       # 将日志内容输出到指定设备
```

#### # action - 保存到数据库，保存到数据库要开启相应模块

```
module (load="ommysql")  
. . . action(type="ommysql"  server="10.0.0.210" db="rsyslog" uid="rsyslogger"  
pwd="123456")
```

#### # action - 发给指定用户

```
root           # 将日志内容发送给用户root  
root, tom     # 将日志内容发送给用户root和tom  
*             # 将日志内容发送给所有已登录用户
```

#### # action - 发送到远程主机

```
@192.168.2.123      # 使用UDP协议到远程主机，默认端口514  
@@log.magedu.com:256 # 使用TCP协议发送到远程主机256端口，默认端口514  
  
# action - 不处理  
stop                # 不处理
```

## rsyslog 日志内容和模版

- 日志内容由template决定，如果没有显式指定，默认使用RSYSLOG\_TraditionalFileFormat，其具体内容如下：

```
# vim /etc/rsyslog.conf  
# 在rsyslog.conf这行指定了默认的模版  
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
```

```
template(name="RSYSLOG_TraditionalFileFormat" type="string"  
       string="%TIMESTAMP% %HOSTNAME% %syslogtag%%msg:::sp-if-no-1st-sp%%msg:::drop-  
       last-1f%\n")
```

- RSYSLOG\_TraditionalFileFormat是一个预定义模版，它是rsyslog内置的一部分，并且通常不会以独立文件的形式存在于文件系统中。

## 自定义模版

- 在rsyslog.conf或其他rsyslog配置文件中使用template指令

```
template(name="MyCustomFormat" type="string" string="%timegenerated% %HOSTNAME%  
%syslogtag%msg%)
```

- rsyslog中有13个内置的模版，我们可以在配置文件中直接使用，其名称如下，具体定义的内容需要查询相关文档

```
RSYSLOG_TraditionalFileFormat  
RSYSLOG_FileFormat  
RSYSLOG_TraditionalForwardFormat  
RSYSLOG_SysklogdFileFormat  
RSYSLOG_ForwardFormat  
RSYSLOG_SyslogProtocol23Format  
RSYSLOG_DebugFormat  
RSYSLOG_WallFmt  
RSYSLOG_StdUserMsgFmt  
RSYSLOG_StdPgSQLFmt  
RSYSLOG_spoofadr  
RSYSLOG_StdJSONFmt
```

- 相关文档

```
https://www.rsyslog.com/doc/v8-stable/configuration/templates.html
```

- 除了使用内置模版外，我们还可以自定义模版
- 自定义模版可以直接写在配置文件中

- 生成的日志内容模版决定，而模版是由rsyslog中的相关属性组成。这些属性在生成日志内容时会被替换成具体内容。所谓属性是指rsyslog中的一些特殊关键字，在模版语法中，使用%属性名%来表示一个字段。
- rsyslog常用属性

```
https://www.rsyslog.com/doc/v8-stable/configuration/properties.html
https://www.rsyslog.com/doc/v8-stable/configuration/property\_replacer.html
```

## 补充内容：:omusrmmsg

```
*. emerg :omusrmmsg.*
# 详解:omusrmmsg:*
# 这是一个输出模块(output module),"omusrmmsg"是rsyslog中的一个模块，用于将日志消息发送给所有登录到系统的用户
# omusrmmsg的作用是通过用户终端发送消息。每当一个紧急事件发生时，所有当前登录的用户都会在他们的终端上看到这条消息
```

## bttmp, lastlog, wtmp命令

日志文件	相关命令	备注
/var/log/btmp	lastb	当前系统上，用户的失败尝试登录相关的日志{bad logins}，二进制格式
/var/log/lastlog	lastlog	每一个用户最近一次的登录信息，二进制格式
/var/log/wtmp	last	当前系统上，用户正常登录系统的相关日志信息(who was logged in)，二进制格式

## dmesg命令

- dmesg命令用来查看主机硬件相关信息

## logger命令

- logger命令可以手动生成相关日志

```
logger [options] [message]

#常用选项
-p | --priority          # 指定优先级
-f | --file               # 从文件中读取日志内容
-t | --tag                # 指定日志tag
-n | --server             # 指定远程主机IP或主机名
-P | --port               # 指定远程主机端口
-T | --tcp                # 指定使用TCP协议传输
-d | --udp                # 指定使用UDP协议传输
```

## 在本机自定义服务日志

```
# ssh服务日志默认是归属于AUTH分类，默认级别是INFO
[root@mystical /etc/rsyslog.d] $cat /etc/ssh/sshd_config | grep -i log
# Logging
#SyslogFacility AUTH
#LogLevel INFO
```

```

#LoginGraceTime 2m
#PermitRootLogin prohibit-password
PermitRootLogin yes
# the setting of "PermitRootLogin without-password".
#PrintLastLog yes

```

将sshd\_config中的默认分类改为Local6

```
SyslogFacility Local6
```

然后在/etc/rsyslog.d/50-default.conf中添加配置项

```
Local6.*      /var/log/sshd.log
```

然后重启rsyslog和sshd服务

即可观察到/var/log/目录下，创建了一个sshd.log文件，并且之后的sshd.log的信息都会被记录在那里

## 使用网络转发日志(日志转储)

- 可以使用rsyslog服务中的远程转发功能，通过TCP或UDP协议将当前主机的日志转发到日志服务器上，进行集中存储，方便统一管理
- 主机清单

主机IP	操作系统	角色
10.0.0.180	Ubuntu	log-server
10.0.0.151	CentOS	client1
10.0.0.150	Ubuntu	client2
10.0.0.181	Ubuntu	mysql-server

- 实验第一步：打开log-server的UDP，TCP模块，使其可以在服务器上监听tcp/udp514端口的数据

```

vim /etc/rsyslog.conf
# 启动相关模块，去掉UDP,TCP模块的注释
# provides UDP syslog reception
module(load="imudp")
input(type="imtcp" port="514")

# 添加hosts解析(可选)
cat /etc/hosts
10.0.0.150 client-1
10.0.0.208 client-2

systemctl restart rsyslog

[root@log-server ~] $ss -tulpn|grep 514
udp    UNCONN 0      0          0.0.0.0:514      0.0.0.0:*      users:
(("rsyslogd",pid=1544,fd=5))
udp    UNCONN 0      0          [::]:514        [::]:*      users:
(("rsyslogd",pid=1544,fd=6))
tcp    LISTEN 0      25         0.0.0.0:514      0.0.0.0:*      users:
(("rsyslogd",pid=1544,fd=7))
tcp    LISTEN 0      25         [::]:514        [::]:*      users:
(("rsyslogd",pid=1544,fd=8))

```

```
# 重启服务器后，发现tcp和udp的514端口都处于监听状态，至此log-server配置成功
```

- 实验第二步，在客户端client1添加配置文件

```
# 配置远程转发，转发到10.0.0.151的udp协议，协议端口号514，可以省略  
vim /etc/rsyslog.d/net.conf  
*.info @10.0.0.151:514  
  
# 重启服务  
systemctl restart rsyslog
```

- 实验第三步，在客户端client2添加配置文件

```
# 配置远程转发，转发到10.0.0.151的udp协议，协议端口号514，可以省略  
vim /etc/rsyslog.d/50-default.conf  
*.info @10.0.0.150:514  
  
# 重启服务  
systemctl restart rsyslog
```

## 使用MySQL数据库转储日志

```
# Log-server上安装rsyslog的mysql包  
apt install rsyslog-mysql  
  
# 查看rsyslog-mysql中的配置文件  
dpkg -L rsyslog-mysql  
  
[root@mysql-server ~] $dpkg -L rsyslog-mysql  
/.  
/usr  
/usr/lib  
/usr/lib/x86_64-linux-gnu  
/usr/lib/x86_64-linux-gnu/rsyslog  
/usr/lib/x86_64-linux-gnu/rsyslog/ommysql.so          # 库文件  
/usr/share  
/usr/share/dbconfig-common  
/usr/share/dbconfig-common/data  
/usr/share/dbconfig-common/data/rsyslog-mysql  
/usr/share/dbconfig-common/data/rsyslog-mysql/install  
/usr/share/dbconfig-common/data/rsyslog-mysql/install/mysql    # mysql数据表文件  
/usr/share/doc  
/usr/share/doc/rsyslog-mysql  
/usr/share/doc/rsyslog-mysql/copyright  
/usr/share/rsyslog-mysql  
/usr/share/rsyslog-mysql/rsyslog-mysql.conf.template      # rsyslog配置文件模版  
/usr/share/doc/rsyslog-mysql/NEWS.Debian.gz  
/usr/share/doc/rsyslog-mysql/changelog.Debian.gz  
  
# 使用ss p-tnlp 能看到，当前mysql只监听本机的端口，因此需要修改配置，使其可以监听外部信息  
State      Recv-Q      Send-Q          Local Address:Port          Peer Address:Port  
Process
```

```

LISTEN      0        151          127.0.0.1:3306          0.0.0.0:*
users:(("mysqld",pid=2732,fd=23))
LISTEN      0        70           127.0.0.1:33060         0.0.0.0:*
users:(("mysqld",pid=2732,fd=21))
LISTEN      0        4096         127.0.0.53%lo:53        0.0.0.0:*
users:(("systemd-resolve",pid=846,fd=14))
LISTEN      0        128           0.0.0.0:22          0.0.0.0:*
users:(("sshd",pid=907,fd=3))
LISTEN      0        128

vim /etc/mysql/mysql.conf.d/mysqld.cnf

#bind-address      = 127.0.0.1
#mysqlx-bind-address = 127.0.0.1

# 将这两行注释掉，然后重启mysql服务
systemctl restart mysql

```

- 配置mysql服务器

```

# 在数据库服务器上建表
scp /usr/share/dbconfig-common/data/rsyslog-mysql/install/mysql
10.0.0.180:/root/rsyslog.sql

# 创建数据库
create database rsyslog

use rsyslog

# 导入SQL文件建表
source /root/rsyslog.sql

# 创建用户并授权
create user 'rsysloger'@'10.0.0.%' identified by '123456'

grant all on rsyslog.* to 'rsysloger'@'10.0.0.%';

flush privileges

```

```

# log-server服务器上配置mysql转发
cp /usr/share/rsyslog-mysql/rsyslog-mysql.conf.template /etc/rsyslog.d/mysql.conf

# 修改配置文件
### Configuration file for rsyslog-mysql
### Changes are preserved

module (load="ommysql")
*.* action(type="ommysql" server="10.0.0.181" db="rsyslog" uid="rsysloger"
pwd="123456")

# 修改好之后，重启服务
systemctl restart rsyslog.service

# 此时就已经功能了，mysql的数据表里应该已经有文件了

```

- 在此基础上，我们可以再实现一个web服务，将Mysql中的日志数据在页面上显示出来

```
# 推荐的开源程序
https://logalyzer.adiscon.com/
```

## 服务日志管理工具 journalctl

- 在systemd为1号进程的系统版本中，systemd提供了一个集中的方式来处理所有来自进程，应用程序等的操作系统日志，所有这些日志事件都由systemd的journald守护进程来处理。journald守护进程收集所有来自Linux操作系统的各种日志，并将其作为二进制数据存储在文件中。
- 以二进制数据集中记录事件，系统问题的好处有很多，例如：由于系统日志是以二进制而不是文本形式存储的，你可以以文本，JSON对象等多种方式进行转译，以满足各种需求。另外，由于日志是按顺序存储的，通过对日志的日期、时间操作，超级容易追踪到单个事件。

```
# 配置文件
cat /etc/systemd/journald.conf

# 日志数据存储目录，非文本文件爱，无法用文本工具打开
tree /var/log/journal

[root@mysql-server ~] $tree /var/log/journal/
/var/log/journal/
└── 5ec4403ae6c74697a9271555b40e7218
    ├── system@00061654cce473d-809b9b3f672383b3.journal~
    ├── system@000616559666195d-d6a5ad2ce7506598.journal~
    ├── system@0006165e94f681f1-2d9cf664135397d6.journal~
    ├── system@42e8e241b15b44069b69af41c6593bed-0000000000000001-
    0006165596616b0f.journal
        ├── system@99eebfa8a9d149f79bd52ced9b525211-0000000000000001-
    0006165e94f14e0f.journal
            ├── system@c61b181bab8b499da5a7dcc700764ee8-0000000000000001-
    00061654cce5c199.journal
            ├── system@c61b181bab8b499da5a7dcc700764ee8-0000000000000007ff-
    00061654cef2cab1.journal
                ├── system@c61b181bab8b499da5a7dcc700764ee8-0000000000000847-
    0006165510ade064.journal
                    ├── system.journal
                    └── user-1000@16f1cad5641647d7ae71127174715e96-0000000000000863-
    0006164bab4e4b02.journal

    1 directory, 10 files
```

- journalctl命令

```
journalctl [options...] [MATCHES...]

# 常规选项
--system          # 显示系统服务和内核相关日志
--user            # 显示来自当前用户的服务日志
-M|--machine=CONTAINER # 显示本地容器中的日志
-S|--since=DATE   # 显示从指定时间之后的日志, YYYY-MM-DD
HH:MM:SS|yesterday|today
-U|--until=DATE   # 显示从指定时间之前的日志
```

```

-b | --boot[-ID]          # 查看详细的启动日志
--list-boots              # 列出详细的系统启动日志
-u | --unit=UNIT           # 根据单元查看日志
-g | --grep=PATTERN        # 根据正则表达式过滤
-f | --follow              # 实时查看日志
-r | --reverse             # 反转输出，最新的在最上面
--header                  # 查看日志文件元数据
--list-catalog            # 查看日志分类
-k                        # 查看内核日志

## 查看启动日志列表
journeyctl --list-boots

## 查看本次启动日志详细信息
journetcctl --boot 0

## 查看nginx服务日志
journalctl -u nginx.service

## 查看ssh服务日志
journalctl -u ssh.service

## 查询日志占用了多少磁盘空间
journalctl --disk-usage

```

## Logrotate 日志转储（文档压缩分割）

### Logrotate介绍

- 在Linux系统中，能够帮助使用者定位问题的有效手段之一就是查日志
- 如果一个服务或一个程序的日志，一直只写一个文件，则会导致该日志文件越来越大，无论是查看还是搜索内容、备份等，都会特别不方便，而且如果服务器数量较多，日志文件大小增长比较快，也会很容易出发告警
- 为了解决这种情况，我们客户使用日志转储服务，对服务日志进行分割，按照一定的规则将日志保存在不同的文件中，这样更便于管理和归档
- logrotate程序是一个日志文件管理工具，用来把旧的日志文件删除，并创建新的日志文件，称为日志转储或滚动。可以根据日志文件的大小，也可以根据其天数来转储，这个过程一般通过cron程序来执行
- 各Linux发行版默认安装logrotate包

dpkg -l logrotate包

- 相关文件

<code>/etc/cron.daily/logrotate</code>	# 定时任务脚本，放在 <code>cron.daily</code> 目录中，默认系统会每天执行一次
<code>/etc/logrotate.conf</code>	# 主配置文件，定义日志转储策略
<code>/etc/logrotate.d/</code>	# 配置文件目录，定义日志转储策略
<code>/usr/sbin/logrotate</code>	# 主程序
<code>/var/lib/logrotate/status</code>	# logrotate服务的日志文件

- 工作原理：系统计划任务每天执行一次脚本文件，在脚本中再执行/usr/sbin/logrotate /etc/logrotate.conf，即调用logrotate程序再配合定义好的转储规则对日志文件进行转储

```
[root@mystical /etc/cron.daily] $cat /etc/cron.daily/logrotate
#!/bin/sh
# skip in favour of systemd timer
if [ -d /run/systemd/system ]; then
    exit 0
fi

# this cronjob persists removals (but not purges)
if [ ! -x /usr/sbin/logrotate ]; then
    exit 0
fi

# 整个脚本本质上就执行这一句
/usr/sbin/logrotate /etc/logrotate.conf
EXITVALUE=$?
if [ $EXITVALUE != 0 ]; then
    /usr/bin/logger -t logrotate "ALERT exited abnormally with [$EXITVALUE]"
fi
exit $EXITVALUE
```

## Logrotate配置

- 在配置文件中定义转储规则，配置文件中的主要配置项

```
[root@mystical /etc/cron.daily] $cat /etc/logrotate.conf
# see "man logrotate" for details

# global options do not affect preceding include directives

# rotate log files weekly
weekly                                # 默认每周一次转储

# use the adm group by default, since this is the owning group
# of /var/log/syslog.
su root adm                            # 默认使用adm组

# keep 4 weeks worth of backlogs
rotate 4                               # 默认保留最近四周的文件（4个文件）

# create new (empty) log files after rotating old ones
create                                # 转储完成后生成新的空文件

# use date as a suffix of the rotated file
#dateext                               # 默认不适用日期后缀

# uncomment this if you want your log files compressed
#compress                             # 默认不启用文件压缩

# packages drop log rotation information into this directory
include /etc/logrotate.d      # 包含的子目录
```

```
# system-specific logs may also be configured here.
```

- 每个服务单独的配置文件，如果在单独配置文件中没有配置项，则使用主配置中的配置或默认配置

```
[root@mystical /etc/cron.daily] $ls /etc/logrotate.d
alternatives apport apt bootlog btmp dpkg rsyslog ubuntu Advantage-tools
ufw unattended-upgrades wtmp

[root@mystical /etc/cron.daily] $cat /etc/logrotate.d/rsyslog
/var/log/syslog
/var/log/mail.info
/var/log/mail.warn
/var/log/mail.err
/var/log/mail.log
/var/log/daemon.log
/var/log/kern.log
/var/log/auth.log
/var/log/user.log
/var/log/lpr.log
/var/log/cron.log
/var/log/debug
/var/log/messages
{
    rotate 4
    weekly
    missingok
    notifempty
    compress
    delaycompress
    sharedscripts
    postrotate
        /usr/lib/rsyslog/rsyslog-rotate
    endscript
}
```

## Logrotate自定义规则实现

- logrotate命令

```
logrotate [option...] <configfile>

# 常用选项
-f|--force          # 强制执行
-m|--mail=command   # 指定执行邮件发送的命令
-v|--verbose         # 显示详细信息
```

- 转储规则-常用配置项

```
compress           # 通过gzip压缩转储以后的日志
nocompress         # 不压缩
create mode owner group # 转储文件，使用指定的权限，所有者，所属组创建新的日志文件
su user group      # 指定转储的用户和组
delaycompress      # 和compress一起使用时，转储的日志文件到下一次转储时才压缩
```

```

error address          # 转储时的错误信息发送到指定的Email地址
ifempty               # 即使是空文件也转储，此为默认选项
notifempty            # 如果空文件的话，不转储
mail address          # 把转储的日志文件发送到指定的E-mail地址
nomail                # 转储时不发送日志文件
olddir directory     # 转储后的日志文件放入指定目录，必须和当前日志文件在同一个文件系统
noolddir               # 转储后的日志文件和当前日志文件放在同一个目录下
prerotate/endscript  # 在转储以前需要执行的命令，这两个关键字必须单独成行，日志文件路径
是第一个参数
postrotate/endscript # 在转储以后需要执行的命令，这两个关键字必须单独成行，日志文件路径
是第一个参数
daily                 # 指定转储周期是每天
weekly                # 指定转储周期是每周
monthly               # 指定转储周期是每月
rotate count          # 指定日志文件删除之前转储的次数，0指没有备份，5指保留5个备份
size size              # 当日志文件到达指定大小时才转储，默认单位是字节bytes，也可以指定
kb或mb
sharedscript           # 对每个转储日志运行prerotate和postrotate脚本，此为默认值
missingok              # 如果日志不存在，不提示错误，继续处理下一个
nomissingok            # 如果日志不存在，提示存在，此为默认值

```

- 自定义实现

```

# 创建测试日志文件
dd if=/dev/zero of=/var/log/test1.log bs=2M count=1
dd if=/dev/zero of=/var/log/test2.log bs=2M count=1

# 定义转储规则
cat /etc/logrotate.d/test{1,2}
/var/log/test1.log {
    daily
    rotate 5
    su root root
    compress
    delaycompress
    missingok
    size 1M
    notifempty
    create 0640 syslog adm
    postrotate
        echo `date +%F_%T` >> /tmp/test1.log
    endscript
}

/var/log/test2.log {
    daily
    rotate 5
    su root root
    dateext
    compress
    delaycompress
    missingok
    size 1M
    notifempty
    create 644 root root
}
```

```

postrotate
    echo `date +%F_%T` >> /tmp/test1.log
endscript
}

# 手动执行转储
logrotate /etc/logrotate.d/test1

# 查看日志, 生成新的空文件, 权限, 属主属组都符合预设
[root@log-server ~]# ls -lh /var/log/test1*
-rw-r----- 1 syslog adm      0 Jul  5 22:58 /var/log/test1.log
-rw-r--r-- 1 root   root  2.0M Jul  5 21:05 /var/log/test1.log.1

...

```

## 网络文件共享

### 常见存储类型

- 直连式存储: Direct-Attached Storage(DAS)
  - 可以理解为本地文件系统。这种设备直接连接到主机总线上，通常是SCSI或FC连接，计算机将其识别为一个块设备，例如常见的硬盘，U盘等，也可以连到一个磁盘阵列，里面有很多块磁盘，DAS只能连接一台服务器，其他服务器无法共享该存储
- 存储区域网络: Storage Area Network (SAN)
  - 存储区域网络，这个是通过光纤通道或以太网交换机连接存储阵列和服务器主机，最后成为一个专用的存储网络。SAN经过多年历史的发展，已经相当成熟，成为业界的事实标准（但各个厂商的光纤交换技术不完全相同，其服务器和SAN存储有兼容性的要求）。SAN提供了一种与现有LAN连接的简易方法，并且通过同一物理通道支持广泛使用的SCSI和IP协议。SAN不受现今主流的基于SCSI存储结构的布局限制。特别重要的是，随着存储容量的爆炸增长，SAN允许企业独立地增加它们的存储容量。**SAN结构允许任何服务器连接到任何存储阵列，这样不管数据放置在哪里，服务器都可以存取所需的数据。**因为采用了光纤接口，SAN还具有更高的带宽。如今的SAN解决方案通常会采用以下两种方法，光纤通道以及SCSI或者基于IP的SAN，也就是FC SAN和IP SAN。光纤信道是SAN解决方案中大家最熟悉类型。但是，最近一段时间以来，基于SCSI的SAN解决方案开始大量出现在市场上，与光纤通道技术比较而言，这种技术具有良好的性能，而且价格低廉。一般通过在主机上安装HBA(Host Bus Adapter主机总线适配器)卡，再通过光纤连接到光纤交换机。在连接至SAN存储上。
- 网络附加存储: Network-Attached Storage(NAS)
  - NAS存储也通常被称为附加存储，顾名思义，就是存储设备通过标准的网路拓扑结构（例如以太网）添加到一群计算机上。与DAS以及SAN不同，**NAS是文件级的存储方法**。采用NAS较多的功能是用来进行文件共享。而且随着云计算的发展，一些NAS厂商也推出了云存储功能，大大方便了企业和个人用户的使用。NAS产品是真正即插即用的产品。
- SAN和NAS的区别

**NAS:** 文件系统在NAS设备上创建和管理，通过文件共享协议远程访问。适合文件共享和数据备份。

**SAN:** 提供块存储，文件系统由连接的服务器在块设备上创建和管理。适合性能密集型应用如数据库。

结论：

因此，NAS设备上的文件系统由**NAS**管理，而客户端通过高级协议进行文件级访问，无法改变文件系统的底层结构。而在**SAN**中，客户端（即连接的服务器）对其所接收的存储块有完全的控制权，包括决定如何格式化和管理这些块存储资源。这种差异定义了两种技术在应用场景中的最佳适用性。

# 常见的存储方式

- 块存储
  - 块存储 (Block Storage Service) 设备不能被操作系统直接访问，需要创建分区或者逻辑卷，再创建文件系统 (EXT3, EXT4, NTFS等)，然后再经过挂载，才能使用。
  - 对于主机而言，其并不能识别出存储设备是真正的物理设备还是二次划分的逻辑设备 (RAID 等)，块存储不仅仅是直接使用物理设备，间接使用物理设备的也叫块设备。比如虚拟机上创建的虚拟磁盘。
  - 块存储很多时候是本地存储，读写速度快，但不利于扩展，数据不能被共享
- 文件存储
  - 文件存储(File Storage Service)可以分为本地文件存储和网络文件存储。文件存储最明显的特征是支持POSIX的文件访问接口。例如：open, read, write, seek, close等 (这几个是常用的文件操作API接口)，与块存储不同，主机不需要再对文件存储进行格式化和创建文件系统，文件管理功能已由文件存储自行管理。
  - 文件存储的特点是便于扩展和共享，但是读写速度较慢
- 对象存储
  - 对象存储 (Object Storage Service) 也叫基于对象的存储，是一种解决和处理离散单元的方法，可提供基于分布式系统之上的对象形式的数据存储服务，对象存储和我们经常接触到的块和文件系统等存储形式不同，它提供RESTful API数据读写接口以及丰富的SDK接口，并且常以网络服务的形式提供数据的访问。
  - 对象存储不支持随机读写，只能进行全读和全写的操作（无法直接在存储上进行修改，只能以文件为单位进行获取和删除），如果需要修改数据，需要在本地编辑完成后整文件上传。

## NFS工作原理

- 网络文件系统：Network File System(NFS), 是由SUN公司研制的UNIX表示层协议，配合客户端和服务端软件，可以本地挂载远程的存储空间，让本地用户和程序直接访问本地目录从而实现数据和存储空间的共享

## NFS内部工作机制

有一个网络服务端(服务提供者)，还有一个网络客户端(服务消费者)，网络服务端会被网络中的其他用户访问，但是网络中的用户并不知道到底是谁来提供服务，因为提供服务的程序很多，用户也很多，那么用户如何找到需要的服务，此时有一个注册中心的概念

注册中心，就是服务端在启动服务的时候，会主动的把自己的服务信息注册到注册中心，注册好之后，用户如何要使用这个服务，就会咨询查看注册中心，进而获取服务提供者的地址，进而相互之间进行服务的提供，访问

NFS需要采取这种方式的原因：这种共享服务有一个特点，它提供的端口是不固定，而且很多，因为NFS要保证多个用户同时访问，必然会产生一些数据的冲突，正常来讲，一个文件在磁盘上如果同时改，就会造成文件的破坏，为了防止这种现象，因此在NFS服务器上有很多机制，来保证文件的安全性，比如锁机制，数据一致性的安全机制，因此，它会用到很多不同的端口，而且这些不同的端口还是动态端口，每次启动的时候端口不固定。所以用户要访问NFS服务，就必须通过注册中心的方式把服务端的地址和端口找到，进而通过注册中心提供的地址，来进行进一步访问。

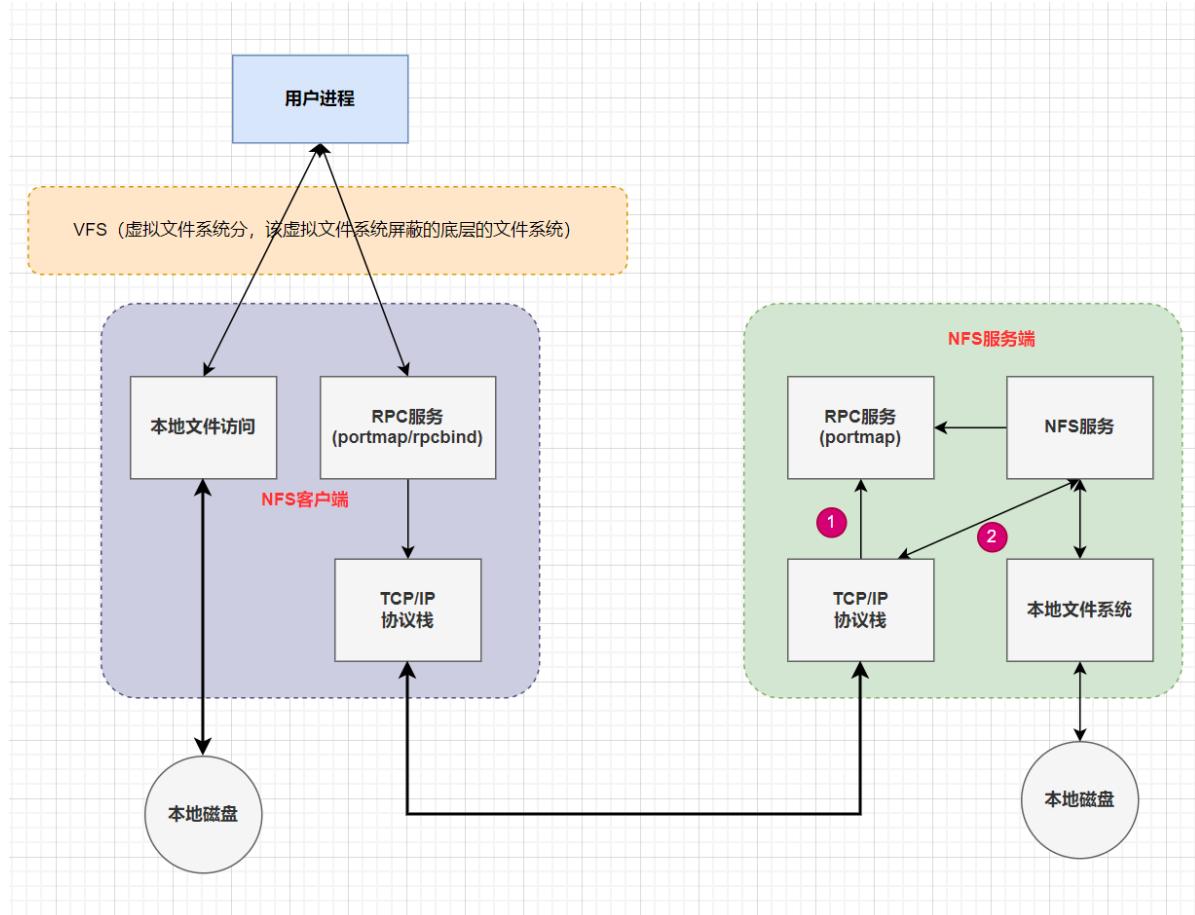
- NFS中的远程访问是基于RPC (Remote Procedure Call Protocol, 远程过程调用) 来实现的

- RPC采用C/S模式，客户机请求程序调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送应答信息，然后等待下一个调用信息，最后，客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行

RPC（远程过程调用）白话版：当用户请求服务器端，需要执行一个操作的时候，他会把这些调用的需求通过网络发给远程的服务器端，服务器端收到以后，会根据客户端发送过来的请求执行对应的操作，把这个操作处理完毕后，在将结果发回给客户端，客户端利用这个结果进一步进行其他操作

(远程过程调用，就相当于你打电话让朋友帮忙做点事，朋友做完事情，你得到这个结果后，继续做你的事情)。

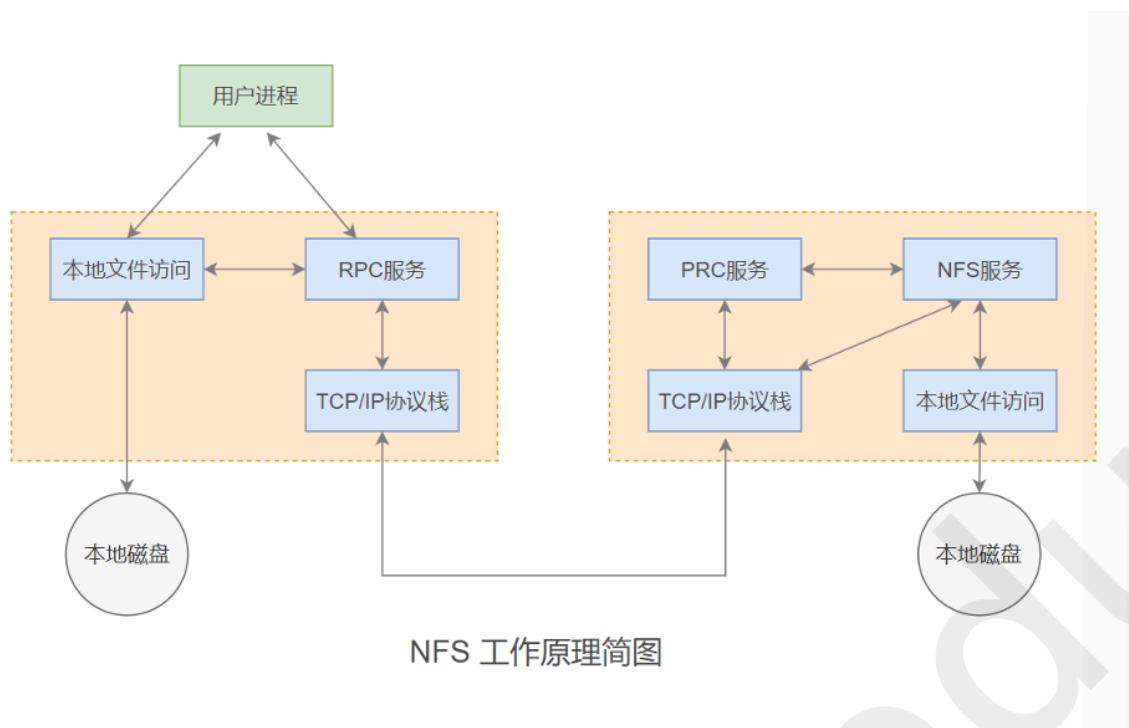
## NFS工作原理图



- VFS (虚拟文件系统)
  - VFS是操作系统中抽象的文件系统接口。无论文件是存储在本地磁盘上，还是通过网络文件系统(如NFS)进行访问，用户进程都是通过VFS来进行统一的文件操作
  - 对于用户进程来说，访问本地文件和远程文件的方式是一样的，这得益于VFS的抽象作用。VFS屏蔽了底层文件系统的实现细节，使得用户进程能够透明的访问远程文件
  - 在NFS的场景下，VFS会决定将用户进程的文件访问请求交给本地文件系统处理(对于本地磁盘)，还是通过NFS客户端来处理远程文件访问
- NFS客户端部分
  - 当VFS识别到某个文件请求是针对远程的NFS服务器时，它会将请求交给NFS客户端
  - NFS客户端会首先尝试通过本地缓存来处理请求，若缓存中不存在该文件，则会通过RPC(远程过程调用)服务来与远程的NFS服务器进行通信
  - RPC服务依赖于Portmap/rpcbind服务来管理和定位服务器上的相应的服务端口
  - NFS客户端最终通过TCP/IP协议栈向NFS服务器发送文件访问请求

- NFS服务器部分

- NFS服务器首先通过RPC服务来接受客户端的文件访问请求，该服务用于定位并分配适合的端口，以便处理来自客户端的请求。
- RPC服务将请求转发给NFS服务，NFS服务负责与服务器端的文件系统进行交互，处理相应的文件读写操作。
- NFS服务从本地文件系统中读取或写入数据，数据存储在服务器的本地磁盘中。
- 一旦数据处理完成，NFS服务通过TCP/IP协议栈将结果返回给客户端



## NFS安装和配置

- NFS基于C/S模式实现，所以有客户端软件和服务端软件

```
# 在rocky中安装，包含了客户端工具和服务端工具
yum install nfs-utils

# 在Ubuntu中安装服务端包
apt install nfs-kernel-server

# 在Ubuntu中安装客户端包
apt install nfs-common
```

服务端依赖包，注册中心，nfs服务中有大量组件，部分组件端口不固定，需要依赖rpcbind发布端口

```
[root@mystical ~] $ss -tulnp|grep rpc
udp    UNCONN  0      0          0.0.0.0:36446      0.0.0.0:*      users:
  ("rpc.mountd",pid=2747,fd=12)
udp    UNCONN  0      0          0.0.0.0:47765      0.0.0.0:*      users:
  ("rpc.statd",pid=2737,fd=8)
udp    UNCONN  0      0        127.0.0.1:793      0.0.0.0:*      users:
  ("rpc.statd",pid=2737,fd=5)
udp    UNCONN  0      0          0.0.0.0:42960      0.0.0.0:*      users:
  ("rpc.mountd",pid=2747,fd=8)
udp    UNCONN  0      0          0.0.0.0:44015      0.0.0.0:*      users:
  ("rpc.mountd",pid=2747,fd=4)
```

```

    udp  UNCONN 0      0          0.0.0.0:111      0.0.0.0:*      users:
    (("rpcbind",pid=2140,fd=5),("systemd",pid=1,fd=95))
    udp  UNCONN 0      0          [::]:37619      [::]:*      users:
    (("rpc.mountd",pid=2747,fd=6))
    udp  UNCONN 0      0          [::]:54212      [::]:*      users:
    (("rpc.mountd",pid=2747,fd=10))
    udp  UNCONN 0      0          [::]:38997      [::]:*      users:
    (("rpc.statd",pid=2737,fd=10))
    udp  UNCONN 0      0          [::]:111      [::]:*      users:
    (("rpcbind",pid=2140,fd=7),("systemd",pid=1,fd=97))
    udp  UNCONN 0      0          [::]:51616      [::]:*      users:
    (("rpc.mountd",pid=2747,fd=14))
    tcp  LISTEN 0     4096      0.0.0.0:35435      0.0.0.0:*      users:
    (("rpc.mountd",pid=2747,fd=5))
    tcp  LISTEN 0     4096      0.0.0.0:56571      0.0.0.0:*      users:
    (("rpc.mountd",pid=2747,fd=9))
    tcp  LISTEN 0     4096      0.0.0.0:54383      0.0.0.0:*      users:
    (("rpc.mountd",pid=2747,fd=13))
    tcp  LISTEN 0     4096      0.0.0.0:111      0.0.0.0:*      users:
    (("rpcbind",pid=2140,fd=4),("systemd",pid=1,fd=94))
    tcp  LISTEN 0     4096      0.0.0.0:35073      0.0.0.0:*      users:
    (("rpc.statd",pid=2737,fd=9))
    tcp  LISTEN 0     4096      [::]:55033      [::]:*      users:
    (("rpc.mountd",pid=2747,fd=7))
    tcp  LISTEN 0     4096      [::]:37573      [::]:*      users:
    (("rpc.mountd",pid=2747,fd=15))
    tcp  LISTEN 0     4096      [::]:55811      [::]:*      users:
    (("rpc.statd",pid=2737,fd=11))
    tcp  LISTEN 0     4096      [::]:111      [::]:*      users:
    (("rpcbind",pid=2140,fd=6),("systemd",pid=1,fd=96))
    tcp  LISTEN 0     4096      [::]:46513      [::]:*      users:
    (("rpc.mountd",pid=2747,fd=11))

```

# 这里面rpcbind的固定监听111端口，告知客户端mountd的开放端口

## 详解RPC协议

- 功能:
  - RPC是一种协议，它允许一个程序请求另一个程序在远程系统上执行代码，而不需要开发者具体处理远程交互的细节。NFS使用RPC来实现其网络功能，允许客户端访问远程服务器上的文件系统，就好像它们是本地挂载的一样。
- rpcbind 的功能:
  - rpcbind 是 RPC 系统的一个组件，它在RPC服务和RPC客户端之间起到了桥梁的作用。具体来说，RPC服务在启动时，会在 rpcbind 上注册它们的一些信息，包括:
    - 服务的标识符（通常是一个程序号和版本号）
    - 它们所监听的端口：当一个RPC客户端（例如NFS客户端）需要与服务（如NFS服务器）通信时，它首先会联系 rpcbind 来查询需要的服务当前监听在哪个端口上。得知端口号后，客户端就可以直接与服务器端的相应服务建立连接。 **rpcbind作用于服务端**
- 服务端主要进程

```
/usr/sbin/rpc.mountd      # 挂载和卸载nfs文件系统，包括权限管理  
/usr/sbin/rpc.nfsd        # 主要的nfs进程，管理用户端是否可登录  
/usr/sbin/rpc.statd       # 非必要，检查文件一致性，可修复文件
```

## NFS远程共享实现

- 配置文件

```
/etc/exports  
/etc/exports.d/*.exports  # 该目录默认不存在，需要自行创建
```

- 配置文件的写法

```
# /etc/exports: the access control list for filesystems which may be exported  
#                   to NFS clients. See exports(5).  
#
```

```
# Example for NFSv2 and NFSv3:
```

```
# /srv/homes      hostname1(rw,sync,no_subtree_check)
```

```
hostname2(ro,sync,no_subtree_check)  
#
```

```
# Example for NFSv4:
```

```
# /srv/nfs4      gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
```

```
# /srv/nfs4/homes gss/krb5i(rw,sync,no_subtree_check)  
#
```

```
# 每一行都是一个共享配置，两列
```

```
# NFS服务器上要被共享的目录路径      可以共享此目录的客户端主机（配置项...）  可以共享此目录的客  
户端主机（配置项...）
```

```
# 客户端主机写法
```

```
# anonymous, *表示不限制，即任何主机都能访问
```

```
# 单个主机可以写具体的IPV4, FQDN, 主机名
```

```
# 网段 10.0.0.0/24 10.0.0.0/255.255.255.0 这两种格式都行
```

```
# 主机名，主机名可以用通配符表示多台主机
```

```
# 网域或主机组 @group_name
```

```
# gss/krb5i 这种写法来限制对使用rpcsec_gss安全性的客户端的访问
```

```
# 配置项写法
```

```
# 默认(ro,sync,root_squash,no_all_squash)
```

```
# 主要选项
```

```
# ro          只读
```

```
# rw          读写
```

```
# root_squash  远程客户端主机上的root用户映射成NFS主机上的UID=65534的用户
```

```
# no_root_squash  远程客户端root用户映射成NFS服务器的root
```

```
# all_squash    所有远程客户端主机用户都映射成NFS主机上的UID=65534的用户，此项覆盖  
no_root_squash
```

```
# anonuid      指定用户映射成特定的用户的UID
```

```
# anongid      指定用户映射成用户的GID
```

```
# sync         同步落盘，在请求数据时立即写入NFS的磁盘上，性能低，安全性高
```

```
# aync         异步落盘，数据先暂存于缓冲区，再一次性落盘，性能高，安全性低
```

```
# insecure    允许非授权访问
```

```
# subtree_check  如果共享子目录，强制NFS检查父目录
```

```
# no_subtree_check 不检查父目录权限  
# wdelay 多个用户写时等待同时写  
# no_wdelay 多个用户写时不等待，立即写，当使用async时，无需此设置  
# hide 不共享NFS服务中的子目录，在NFSv4中无效  
# secure NFS服务通过1024以下的安全TCP/IP端口通讯  
# insecure NFS服务通过1024以上的安全TCP/IP端口通讯
```

- 修改好配置文件后，重载配置

```
# 重载配置  
exportfs -r  
  
# 查看配置  
exportfs -v
```

- 远程挂载实现

```
# 在服务器端10.0.0.150，配置文件中添加  
/data/dir      *  
  
# 指定一台主机授权  
# 多台机器，空格分隔  
/data/dir  10.0.0.151(rw) 10.0.0.165  
  
# 保存后，重载配置  
exportfs -r  
  
# 在客户端10.0.0.151远程挂载  
mount 10.0.0.150:/data/dir1 ./test  
  
# 挂载成功后，查看远程共享的文件，发现该目录下文件只读  
# 在服务器配置文件中添加rw属性  
vim /etc/exports  
/data/dir1  *(rw)  
  
# 重载  
exportfs -r  
  
# 之后在客户端创建文件，不会在提示只读，而是提示权限不足  
  
# 此时发现服务端被挂载的目录，权限是755，只有属主有w权限，其他和组都没有，因此更改权限  
chmod 777 /data/dir1  
  
# 再次尝试在客户端创建文件，成功  
# 发现文件在服务端的属组属主是 nogroup和nobody
```

- 永久挂载

```
vim /etc/fstab  
# 记得添加_netdev，标识是网络设备  
# 如果设备不在线，有此标识，不会卡在开机阶段  
10.0.0.208:/data/dir1  /var/www/html  nfs  defaults,_netdev  0  0
```

## NFS常用工具

- rpcinfo
  - rpcinfo可以访问指定的RPC服务器，显示其响应信息，从而查询出该服务器上注册的RPC服务
  - 如果不指定主机，则默认当前主机

```
# 使用简短模式  
rpcinfo -s  
  
# 查看指定主机  
rpcinfo -s 10.0.0.28
```

- exportfs
  - exportfs命令用于管理本机NFS文件系统，默认配置文件/etc/exports

```
# 重新挂载  
exportfs -r  
  
# 卸载  
exportfs -u  
  
# 显示挂载信息  
exportfs -v
```

- showmount
  - showmount可以查看远程主机的共享设置

```
showmount [option...] [host]  
  
# 常用选项  
-a          # 显示已连接的客户端  
-e          # 显示指定NFS服务器上的配置列表  
  
# 实例  
showmount -e 10.0.0.208
```

- mount.nfs
  - 客户端NFS服务挂载命令，将远程NFS服务器上共享出来的目录挂载到本机，可以直接写成mount

```
mount.nfs remotetarget dir [option]  
  
# 常用选项  
-r          # 只读挂载  
-w          # 读写挂载  
-v          # 显示详细信息  
-o          # 指定挂载参数  
  
# 常用挂载参数  
fg          # 前台挂载，默认  
bg          # 后台挂载
```

```
hard      # 持续请求, 如果挂不上, 一直尝试
soft      # 非持续挂载
intr      # 是否可以强制中断, 配合hard, 有此参数, 可以使用ctrl + c强行停止挂载
_netdev   # 无网络服务时, 不挂载NFS资源
vers=n|nfsvers=n # 指定nfs协议版本号, 如果nfs服务端不支持此版本, 则无法挂载, 默认4.2
```

## NFS共享实现

- 在客户端root用户创建的文件, 在服务端的UID, GID默认65534, 即nobody,nogroup
- 更改这种映射关系

```
# 添加选项no_root_squash (不推荐)
客户端是root权限, 服务端不映射, 也是root权限
```

```
# 只有root映射, 普通用户不映射, 继承UID和GID
```

```
# 使用all_squash 所有的都映射, 不管客户端什么用户添加文件, 在服务端统一都是nobody和nogroup
```

- 统一用户映射

```
#NFS 服务端配置
# 创建一个www的组
groupadd -g 12345 www

# 创建一个新用户
useradd -u 12345 -g 12345 -s /sbin/nologin www

# 指定映射用户, 对root和普通用户都生效
/data/www *(rw, all_squash, anonuid=12345, anongid=12345)

# 重载服务
exportfs -r
```

- 统一客户端主机用户和权限的方法
  - 保证客户端主机使用相同的用户 (相同的用户ID) 对共享文件进行读写
  - 在NFS服务器上统一映射
  - 使用LDAP服务进行集中的账号管理

## 文件实时同步

- 在生产环境中, 某些场景下, 要将数据或文件进行实时同步, 保证数据更新后其他节点能立即获得最新的数据
- 数据同步的两种方式
  - PULL: 使用定时任务的方式, 配合同步命令或脚本等, 从指定服务器上将数据同步到本地, 一般是周期性定时同步
  - PUSH: 如果当前机器上的数据或文件发生了更新, 立即推送到指定的节点上, 可以做到实时同步

# 实时同步的实现方式

- inotify + rsync
- sersync

## inotify + rsync实现文件实时同步

- 实现原理
  - 利用内核中的inotify监控指定目录，当目录中的文件或数据发生变化时，立即调用rsync服务将数据推送到远程主机上

## inotify服务

- inotify是一个内核用于通知用户空间程序文件系统变化的机制，在监听到文件系统发生变化后，会向相应的应用程序发送事件，比如：文件增加，修改，删除等事件发生后可以立即让用户空间知道

```
https://github.com/rvicilas/inotify-tools #项目地址
```

- inotify是内核中的功能，在2.6.13版本的内核开始都默认存在

```
# ubuntu和CentOS都类似，仅仅是内核版本号不同
grep -i inotify /boot/config-5.15.0.76-generic
CONFIG_INOTIFY_USER=y
```

- inotify内核参数

```
[root@mystical /data/dir1] $ls -l /proc/sys/fs/inotify
total 0
-rw-r--r-- 1 root root 0 Apr 20 07:36 max_queued_events
-rw-r--r-- 1 root root 0 Apr 20 07:36 max_user_instances
-rw-r--r-- 1 root root 0 Apr 20 07:36 max_user_watches

# max_queued_events 事件队列最大长度
# 值太小会出现Event Queue Overflow 错误，默认16384，生成环境建议调大，比如327679
# 含义：一个队列中，可以同时拥有的事件数

# max_usr_instance 每个用户创建inotify实例的最大值，默认128
# 一个用户可以创建128个实例，一个实例可以监控多个目录或文件，这个受内核参数max_user_watches控制

# max_user_watches
# 每个实例可以监视文件的总数量
```

- 内核参数如果需要修改，可以配置文件

```
cat /etc/sysctl.conf

fs.inotify.max_queued_events=66666
fs.inotify.max_user_instances=256
fs.inotify.max_user_watches=1000000

# 激活设置的内核参数
sysctl -p
```

- 安装软件
- Inotify是内核中的功能模块，只能通过调用API接口的形式使用其功能，我们可以通过相关软件对其进行操作，能实现内核中inotify调用的软件主要有以下几个

```
inotify-tools, sersync, lrsyncd
```

- 下载安装inotify-tools

```
apt install inotify-tools

# 主要工具
/usr/bin/fsnotifywait      # fsnotifywait监控工具，是inotifywait的新版本
/usr/bin/fsnotifywatch     # fsnotify统计工具
/usr/bin/inotifywait        # 实时监控指定目录的所有事件，在被监控的文件或目录上等待特定事件发生(open,close,write...)
/usr/bin/inotifywatch       # 收集被监控文件系统使用的统计数据，指文件系统事件发生的次数统计
```

- inotifywait命令

```
inotifywait [option...] file1 [file2]...

# 常用选项
@<file>                      # 排除指定文件
--exclude <pattern>            # 不监控指定的文件或目录，正则匹配
--excludei <pattern>           # 不监控指定文件或目录，正则，不区分大小写
--include <pattern>            # 仅监控指定的文件或目录，正则匹配
--includei <pattern>           # 仅监控指定的文件或目录，正则，不区分大小写
-m|--monitor                    # 始终保持监听，直到超时
-d|--daemon                     # 以守护进程方式执行，配合-o使用
-r|--recursive                  # 对目录递归监控，即监控目录内所有内容
--fromfile <file>              # 从文件中读取要监控的内容
-o|--outfile <file>            # 将执行结果输出到指定文件
-q|--quiet                       # 简短输出，仅输出事件信息
-qq                             # 禁用输出
--format <fmt>                 # 指定输出格式
--timefmt <fmt>                 # 指定时间输出格式
-e|--event <event1>             # 只监听指定事件，默认监听所有事件

# 范例
--timefmt "%Y-%m-%d %H:%M:%S"

# --format选项可用字段
%T      # 输出时间格式中定义的时间格式信息，通过--timefmt option语法格式指定语法信息
%w      # 事件出现时，监控文件或目录的名称信息，相当于dirname
```

```

%f      # 事件出现时, 将显示监控目录下出发事件的文件或目录信息, 否则为空, 相当于basename
%e      # 显示发生的事件信息, 不同的事件默认用逗号分隔
%xe    # 显示发生的事件信息, 不同的事件指定用x进行分割

# 范例
--format "%T %w%f event: %;e"

# --timefmt 选项可用字段
%Y      # 年份信息, 包含世纪信息
%y      # 年份信息, 不包含世纪信息
%m      # 显示月份信息, 范围01-12
%d      # 每月的第几天, 范围是01-31
%H      # 小时信息, 使用24小时制, 范围00-23
%M      # 分钟, 范围00-59
%S      # 秒, 范围0-60

# -e 选项可以指定的事件
access      # 文件或目录内容被读取
modify      # 内容发生变化
attrib      # 属性发生变化
close_write  # 写入模式打开后关闭
close_nowrite # 只读打开后关闭
close        # 关闭事件, 无关模式
open         # 打开事件, 无关模式
moved_to    # 文件或目录被移动到监控的目录中
moved_from  # 文件或目录从监控的目录中被移动
create       # 创建文件或目录
delete      # 文件或目录被删除
unmount     # 取消挂载

```

- 实例

```

# 监控单一目录
inotifywait /data

# 在另一个会话中, 查看该目录
ls /data

# 监控端显示 (默认只监控一步)
[root@mystical /data] $inotifywait /data/
Setting up watches.
watches established.
/data/ OPEN, ISDIR

# 持续监控
inotifywait -m /data

# 持续递归监控
inotifywait -mr /data

```

# rsync服务

- rsync常用于作为linux系统下的数据镜像备份工具，实现远程同步，支持本地复制，或者与其他SSH，rsync主机同步数据，支持增量备份，配合任务计划，rsync能实现定时或间隔同步，配合inotify或sersync，可以实现触发式的实时数据同步

```
http://rsync.samba.org/ # 官方网站
```

- 软件安装

```
apt install rsync
```

- 命令格式

```
# Local
rsync [option]... SRC [SRC]... DEST # 本地文件同步

# Access via remote shell
rsync [option]... [user@]HOST:SRC[DEST] #PULL 拉，将远程主机上的文件拉到本地
rsync [option]... SRC [SRC]... [USER@]HOST:DEST #PUSH 推，将本地文件推送到远程主机上

# Access via rsync daemon
Pull:
rsync [OPTION...] [USER@]HOST::SRC... [DEST]
rsync [OPTION...] rsync://[USER@]HOST[:PORT]/SRC... [DEST]

Push:
rsync [OPTION...] SRC... [USER@]HOST::DEST
rsync [OPTION...] SRC... rsync://[USER@]HOST[:PORT]/DEST
```

- rsync有三种工作方式
  - 本地文件系统上实现同步，命令行语法格式为上述local段的格式
  - 本地主机使用远程shell和远程主机通信。命令行语法格式为上述"Access via rsync shell"段的格式，使用的用户名和密码是rsync服务中配置的用户名和密码
  - 本地主机通过网路套接字连接远程主机上的rsync daemon。命令行语法格式为上述"Access via rsync daemon"段的格式，使用的用户名和密码是rsync服务中配置的用户名和密码
- 前两者的本质是通过本地或远程shell，而第三种方式则是让运行rsyncd服务，使其监听在一个端口上，等待客户端的连接，此种方式使用的rsync的专有协议
- 常用选项

```
--verbose|-v      # 显示详细信息，最多可以-vvvv
--quiet|-q        # 不输出错误信息
--archive | -a    # 归档模式，保留文件属性
-r                # 递归
```

# 实现rsync daemon服务

- 主机清单
  - rsync-server: 10.0.0.208
  - rsync-client: 10.0.0.206

```
# 服务端以守护进程运行
# 在备份服务器启动rsync进程
# 本质是rsync --daemon
systemctl enable --now rsync.service

# 启动失败，因为/etc/rsyncd.conf配置文件不存在

# 创建空配置文件
touch /etc/rsyncd.conf

# 启动服务
systemctl start rsync.service

# 可以使用ss查看到rsync监听873端口
ss -tnlp | grep rsync

# 客户端链接测试
rsync rsync://10.0.0.208
```

```
# 修改服务端配置，指定共享目录
cat /etc/rsyncd.conf
[dir1]
path=/data/dir1/
read only=no

# 重启服务
systemctl restart rsync.service

# 此时客户端在执行测试就能看到共享目录的自定义名称
rsync rsync://10.0.0.208
dir1
# 或者后面跟两个冒号也能表示rsync协议
rsync 10.0.0.208:::

# 此时传输文件到远程会失败
root@localhost ~/test] $rsync /etc/fstab root@10.0.0.150::dir1
rsync: [receiver] mkstemp "/.fstab.cr007k" (in dir1) failed: Permission denied
(13)
rsync error: some files/attrs were not transferred (see previous errors) (code
23) at main.c(1189) [sender=3.1.3]
# 失败原因：此处的root是指rsync服务的用户，当前服务端并没有配置此信息，默认会被映射成nobody
# 解决方法1：因为最终被映射到服务端的用户名都是nobody，因此可以通过给nobody权限来解决

setfacl -m u:nobody:rwx /data/dir1

# 客户端重新执行文件传输
root@localhost ~/test] $rsync /etc/fstab root@10.0.0.150::dir1
```

```
# 成功，服务端可以看到，传过来的文件属主属组是nobody:nogroup
[root@mystical /data] $ll dir1
total 24
drwxrwxr-x+ 2 root    root    4096 Apr 20 11:04 .
drwxr-xr-x  4 root    root    4096 Apr 20 10:50 ..
-rw-r--r--  1 root    root     19 Apr 20 10:02 abc.txt
-rw-r--r--  1 nobody  nogroup 655 Apr 20 11:04 fstab

# 解决方法2（推荐）
# 在配置文件上给rsync指定账号密码，日志文件，并禁止匿名链接
[root@mystical /data] $cat /etc/rsyncd.conf
uid=root
gid=root
max connection=0
log file=/var/log/rsyncd.log
pid file=/var/run/rsyncd.pid
lock file=/var/run/rsyncd.lock

[dir1]
path=/data/dir1/
comment=rsync dir1
read only=no
auth users=rsyncer
secrets file=/etc/rsyncd.pwd

# 密码文件
[root@mystical /data] $cat /etc/rsyncd.pwd
rsyncer:123456

# 重启服务
systemctl restart rsync.service

# 此时在客户端尝试发送文件还会报错

● rsync.service - fast remote file copy program daemon
   Loaded: loaded (/lib/systemd/system/rsync.service; disabled; vendor preset: enabled)
     Active: active (running) since Sat 2024-04-20 11:13:38 UTC; 48min ago
       Docs: man:rsync(1)
              man:rsyncd.conf(5)
     Main PID: 18089 (rsync)
        Tasks: 1 (limit: 2178)
      Memory: 824.0K
         CPU: 18ms
        CGroup: /system.slice/rsync.service
                  └─18089 /usr/bin/rsync --daemon --no-detach

Apr 20 11:13:38 mystical rsyncd[18089]: Unknown Parameter encountered: "max connection"
Apr 20 11:13:38 mystical rsyncd[18089]: IGNORING unknown parameter "max connection"
Apr 20 11:13:38 mystical rsyncd[18089]: rsyncd version 3.2.7 starting, listening on port 873
```

```
Apr 20 12:01:26 mystical rsyncd[18111]: unknown Parameter encountered: "max
connection"
Apr 20 12:01:26 mystical rsyncd[18111]: IGNORING unknown parameter "max
connection"
Apr 20 12:01:26 mystical rsyncd[18111]: name lookup failed for 10.0.0.151: Name
or service not known
Apr 20 12:01:26 mystical rsyncd[18111]: connect from UNKNOWN (10.0.0.151)
Apr 20 12:01:26 mystical rsyncd[18111]: rsync allowed access on module dir1 from
UNKNOWN (10.0.0.151)
# 这里重点: secrets file must not be other-accessible (see strict modes option)
Apr 20 12:01:35 mystical rsyncd[18111]: secrets file must not be other-accessible
(see strict modes option)
Apr 20 12:01:35 mystical rsyncd[18111]: auth failed on module dir1 from UNKNOWN
(10.0.0.151) for rsyncer: ignoring secrets file
```

# 错误原因，密码文件不能被所有人可读，因此需要将密码文件权限设置为600

```
chmod 600 /etc/rsyncd.pwd
```

# 重启服务

```
systemctl restart rsync
```

# 重新在客户端测试，即可成功

```
$rsync /etc/fstab rsyncer@10.0.0.150::dir1
passwd:123456
```

# 将客户端密码保存在单独文件，实现非交互连接

```
echo "123456" > /etc/rsyncd.pwd
chmod 600 /etc/rsyncd.pwd
```

# 客户端测试，保证服务端与客户端同步

# --delete选项，此次同步完成后，删除服务端不在此次同步的文件

# -a保留属性

# -v显示过程

# -z压缩传输

```
rsync -avz --delete --password-file=/etc/rsyncd.pwd /data/www
rsyncer@10.0.0.208::dir1
```

- rsync服务常用配置项

```
cat /usr/share/doc/rsync/examples/rsyncd.conf

# GLOBAL OPTIONS          # 全局配置部分

uid=root                  # 可指定uid, gid
gid=root
motd file=/etc/motd      # motd文件路径
log file=/var/log/rsyncd   # 日志文件
pid file=/var/run/rsyncd.pid # PID文件
syslog facility=daemon    # rsyslog 日志服务中的日志归类

[ftp]                      # 客户端显示的名称

comment=public archive     # 客户端列出该共享目录时的说明信息
```

```

path= /var/www/pub          # 服务器具体目录
use chroot=yes              # 更改映射目录的根目录
max connection=10           # 最大连接数，同时有多少客户端可以连接
lock file=/var/lock/rsyncd   # 锁文件
read only=yes                # 默认客户只读访问
list = yes                   # 客户端是否可以列出服务端内容
exclude=                     # 指定排除项，在此处指定的内容在客户端无法显示
exclude from=                # 从指定文件中读取排除项
include=                     # 指定显示项
include from=                # 同上
auth users=                  # 客户端用户名，启动此项表示禁用匿名登录
secrets file=/etc/rsyncd.secrets # 客户端用户名和密码对应文件
hosts allow=                 # 客户端白名单
hosts deny=                  # 客户端黑名单

```

## inotify + rsync 实现数据实时同步

```

客户端同步脚本
cat ./www_rsync.sh

#!/bin/bash

SRC='/data/www/'
DEST='rsyncer@10.0.0.105::dir1'
dpkg -v inotify-tools &> /dev/null || apt install inotify-tools -y
dpkg -v rsync &> /dev/null || apt install rsync -y

# 不间断监控指定目录中的特定事件，当目录中有事件发生变化时，调用rsync命令进行同步
inotifywait --mrq --exclude=".*\.\swp" --timefmt '%Y-%m-%d %H:%M:%S' --format '%T
%w %f' -e create, delete, moved_to, close_write, attrib ${SRC} | while read DATE
TIME DIR FILE;do
    FILEPATH=${DIR}${FILE}
    rsync -az --delete --password-file=./www_rsync.pwd $SRC $DEST && echo "At
${TIME} on ${DATE}, file ${FILEPATH} was backup via rsync" >> ./www_rsync.log
done

```

## sersync实现数据实时同步

- sersync类似于inotify，同样用于监控，但它克服了inotify的缺点，Inofity最大的不足是会产生重复事件，或者同一目录下多个文件的操作会产生多个事件
  - 例如：当监控目录中有5个文件时，删除目录时会产生6个监控事件，从而导致重复调用rsync命令。另外比如：vim文件时，inotify会监控到临时文件的事件，但这些事件相对于rsync来说是不应该被监控的
- sersync特点
  - sersync是使用C++编写的，而且对Linux系统文件系统产生的临时文件或重复的文件操作系统过滤，所以在结合sersync同步的时候，节省了运行时耗和网络资源。因此更快
  - sersync配置很简单，其中提供了静态编译好的二进制文件和XML配置文件，直接使用即可
  - sersync使用多线程进行同步，尤其在同步较大文件时，能够保证多个服务实时保持同步状态
  - sersync有错误处理机制，通过失败队列对出错文件重新同步，如果仍旧失败，则按设定时长对同步失败文件重新同步

- serynsc不仅可以实现实时同步，另外还自带crontab功能，只需在xml配置文件中开启，即也可以按要求隔一段时间整体同步一次，而不需再额外配置crontab功能
- sersync可以二次开发
- 项目地址

```
https://code.google.com/archive/sersync/ # 项目地址
```

```
# 下载
wget https://storage.googleapis.com/google-code-archive-
downloads/v2/code.google.com/sersync/sersync2.5.4_64bit_binary_stable_final.tar.g
z

# 解压
tar xf sersync2.5.4_64bit_binary_stable_final.tar.gz

[root@mystical ~] $ls
sersync2.5.4_64bit_binary_stable_final.tar.gz  snap  tmux-onedark-theme
[root@mystical ~] $tar xf sersync2.5.4_64bit_binary_stable_final.tar.gz
[root@mystical ~] $ls
GNU-Linux-x86  sersync2.5.4_64bit_binary_stable_final.tar.gz  snap  tmux-onedark-
theme
[root@mystical ~] $cd GNU-Linux-x86/
[root@mystical ~/GNU-Linux-x86] $ls
confxml.xml  sersync2

# 移动至环境变量
mv GNU-Linux-x86/ /usr/local/sersyncpa

# 确认安装rsync客户端工具
rpm -q rsync &> /dev/null || apt -y install rsync

# 备份sersync配置文件
cp /usr/local/sersync/confxml.xml{,.bak}

# 修改sersync配置文件
vim /usr/local/sersync/confxml.xml

<?xml version="1.0" encoding="ISO-8859-1"?>
<head version="2.5">
    <host hostip="localhost" port="8008"></host>
    <debug start="false"/>          # 是否开启调试模式
    <fileSystem xfs="false"/>
    <filter start="false">          # 不开启文件过滤功能，当为true时，以下类型的文件将不同
步
    <exclude expression="(.*)\svn"></exclude>
    <exclude expression="(.*)\gz"></exclude>
    <exclude expression="^info/*"></exclude>
    <exclude expression="^static/*"></exclude>
</filter>
    <inotify>          # 监控事件
    <delete start="true"/>
    <createFolder start="true"/>
    <createFile start="false"/>
    <closewrite start="true"/>
```

```

<moveFrom start="true"/>
<moveTo start="true"/>
<attrib start="false"/> # 修改为true, 文件属性变化也会同步
<modify start="false"/>
</inotify>

<sersync> # rsync命令的配置段
<localpath watch="/opt/tongbu"> # 需要同步的源目录或文件, 建议同步目录
    <remote ip="127.0.0.1" name="tongbu1"/> # 指定备份服务器地址和rsync daemon
    的模块名, 如果下面开起来ssh start, 此时name为远程shell方式运行时的目标目录
        <!--<remote ip="192.168.8.39" name="tongbu"/>-->
        <!--<remote ip="192.168.8.40" name="tongbu"/>-->
</localpath>
<rsync> # 指定rsync选项
    <commonParams params="-artuz"/>
    <auth start="false" users="root" passwordfile="/etc/rsync.pas"/> # 修改为
    true, 定备份服务器的rsync配置的用户和密码文件
        <userDefinedPort start="false" port="874"/><!-- port=874 --> # 指定rsync的
    非端口号
        <timeout start="false" time="100"/><!-- timeout=100 -->
        <ssh start="false"/> # 默认使用rsync daemon运行rsync命令, true为使用远程shell
    模式
</rsync>
<failLog path="/tmp/rsync_fail_log.sh" timeToExecute="60"/><!--default every
60mins execute once--> # 错误重传及日志文件路径
<crontab start="false" schedule="600"><!--600mins--> # 不开启crontab
    <crontabfilter start="false"> # 不开启crontab定时传输的筛选功能
        <exclude expression="*.php"/></exclude>
        <exclude expression="info/*"/></exclude>
    </crontabfilter>
</crontab>
<plugin start="false" name="command"/>
</sersync>

#####
# 后续不需要修改 #####
<plugin name="command">
    <param prefix="/bin/sh" suffix="" ignoreError="true"/> <!--prefix
    /opt/tongbu/mmm.sh suffix-->
    <filter start="false">
        <include expression="(.*)\.\php"/>
        <include expression="(.*)\.\sh"/>
    </filter>
</plugin>

<plugin name="socket">
<localpath watch="/opt/tongbu">
    <deshost ip="192.168.138.20" port="8009"/>
</localpath>
</plugin>
<plugin name="refreshCDN">
<localpath watch="/data0/htdocs/cms.xoyo.com/site/">
    <cdninfo domainname="ccms.chinacache.com" port="80" username="xxxx"
    passwd="xxxx"/>
        <sendurl base="http://pic.xoyo.com/cms"/>
        <regexurl regex="false" match="cms.xoyo.com/site([/a-zA-Z0-
    9]*)\.xoyo.com/images"/>

```

```

    </localpath>
    </plugin>
</head>

# 创建连接rsynd服务器的用户密码文件，并必须修改权限
echo magedu > /etc/rsync.pas
chmod 600 /etc/rsync.pas

# 以后台方式执行同步
sersync2 -dro /usr/local/sersync/confxml.xml

```

## 基于远程shell实现sersync

```

# 不需要配置rsync daemon，只需要配置基于key验证的ssh即可
ssh-keygen
ssh-copy-id backup-server

# 下载sersync，并拷贝至相应的目录，设置PATH变量同5.5.2

# 修改sersync配置文件
# 重点修改
# 第一步修改
<ssh start="true"/> # 修改此行为true，使用远程shell方式的rsync连接方式，无需在目标主机上配置启动rsync daemon服务

# 第二步修改
<remote ip="10.0.0.18" name="/data/backup"/>

# 不启用rsync认证
<auth start="false" usrs="root" passwordfile="/etc/rsync.pas">

```

## 排查方法

```

cat /tmp/rsync_fail_log.sh

#errno -1
cd /data/dir1 && rsync -artuz -R --delete ./ --include="a.txt" --exclude=*
rsyncer@10.0.0.231::dir1 --password-file=/etc/rsyncd.pwd >/dev/null 2>&1

#errno -1
cd /data/dir1 && rsync -artuz -R "./aa/bb" rsyncer@10.0.0.231::dir1 --password-
file=/etc/rsyncd.pwd >/dev/null 2>&1

#errno -1
cd /data/dir1 && rsync -artuz -R "./aa" rsyncer@10.0.0.231::dir1 --password-
file=/etc/rsyncd.pwd >/dev/null 2>&1

# 将报错语句的重定向去掉，查看报错
cd /data/dir1 && rsync -artuz -R "./aa" rsyncer@10.0.0.231::dir1 --password-
file=/etc/rsyncd.pwd

```

```
ERROR: password file must not be other-accessible
rsync error: syntax or usage error (code 1) at authenticate.c(197) [sender=3.2.3]

# 报错显示密码文件不能被其他人访问，但是当前可以所以报错，说明是权限问题
# 更改密码文件权限

chmod 600 /etc/rsyncd.pwd
```

## Web服务

### Web访问响应模型 (Web I/O)

- 单进程I/O模型：启动一个进程处理用户请求，而且一次只处理一个，多个请求被串行
- 多进程I/O模型：并行启动多个进程，每个进程响应一个链接请求
- 复用I/O模型：启动一个进程，同时响应N个请求
- 复用的多进程I/O模型：启动M个进程。每个进程响应N个连接请求，同时接受M\*N个请求

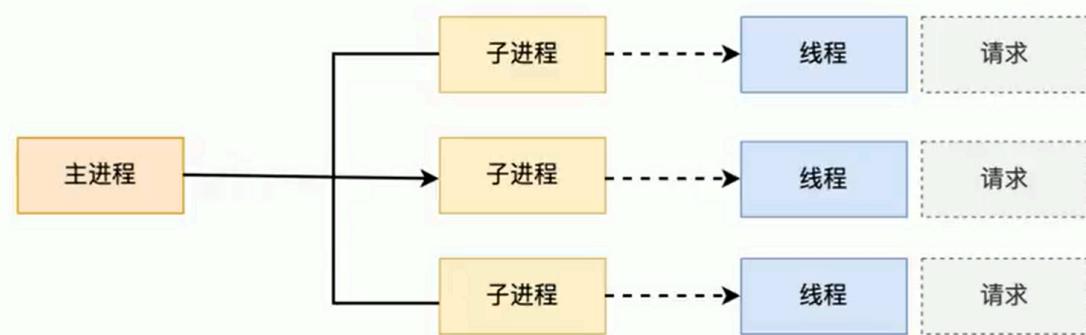
### Apache经典的Web服务端

### Apache三种工作模式（面试题）

- MPM: multi-processing modules 有三种工作模式

#### Apache Prefork模型

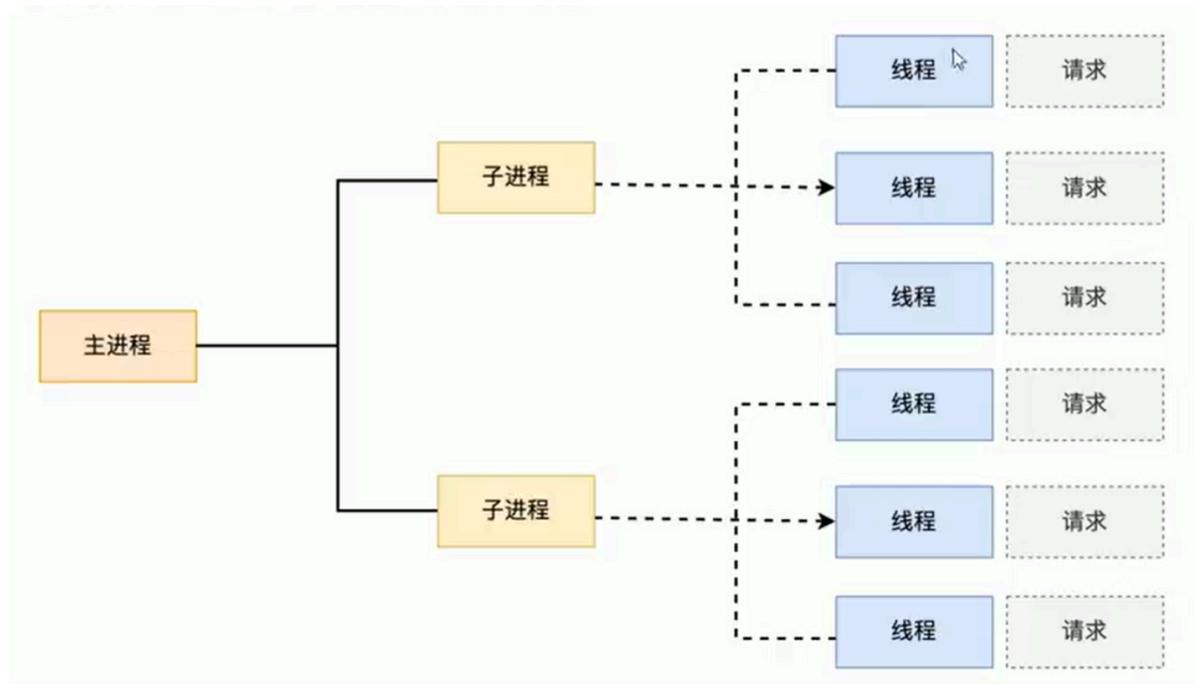
- 预派生模式，有一个主控制进程，然后生成多个子进程，每个子进程有一个独立的线程响应用户请求，相对比较占用内存，但是比较稳定，可以设置最大和最小进程数，是最古老的一种模式，也是最稳定的模式，适用于访问量不大的场景
- 优点：稳定
- 缺点：每个用户请求需要对应开启一个进程，占用资源较多，并发性差，并不适用于高并发场景



#### Apache worker模型

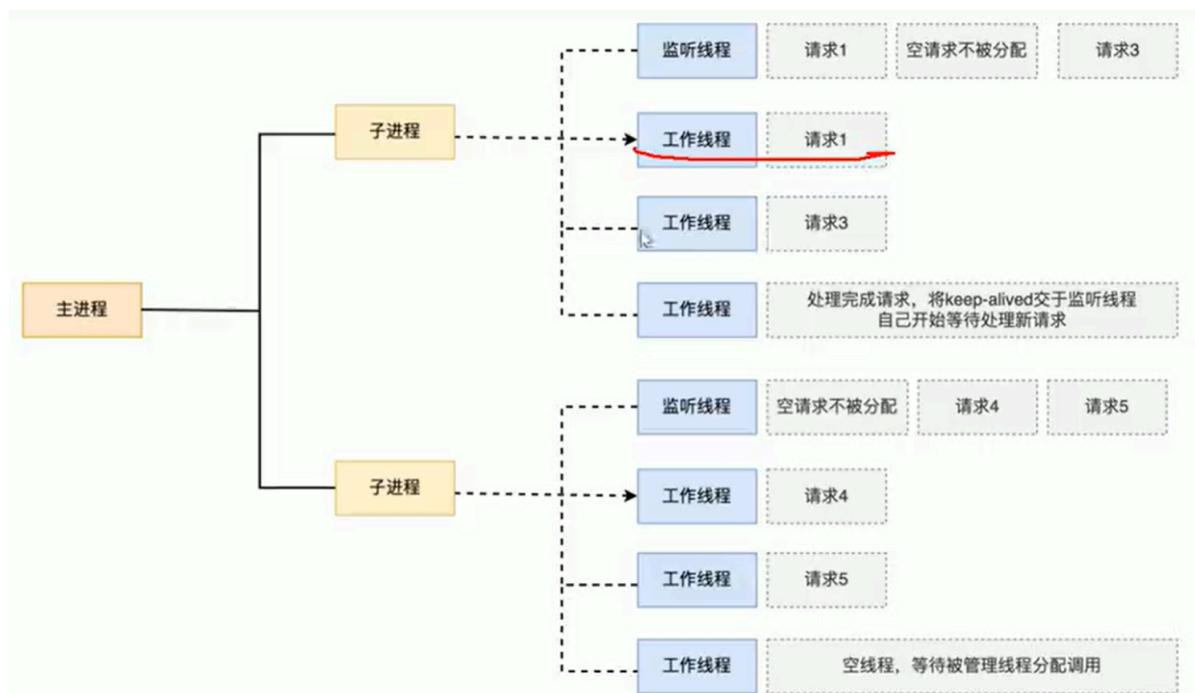
- 一种多进程和多线程混合的模型，有一个控制进程，启动多个子进程，每个子进程里面包含固定的线程，使用线程来处理请求，当线程不够使用的时候会再启动一个新的子进程，然后在进程里面再启动线程处理请求，由于其使用了线程处理请求，因此可以承受更高的并发
- 优点：相比prefork占用的资源较少，可以同时处理更多的请求

- 缺点：使用keepalive的长连接方式，某个线程会一直被占据，即便没有传输数据，也需要一直等到超时才会被释放，如果过多的线程，被这样占据，也会导致再高并发场景下的无服务线程可用（该问题在prefork模式下，同样会发生）



## Apache event模型

- Apache中最新的模式，2012年发布的apache2.x系列正式支持event模型，属于事件驱动模型，每个进程响应多个请求，在现在版本里的已经是稳定可用的模型。它和work模型很像，最大的区别在于，它解决了keepalive场景下，长期被占用的线程的资源浪费问题（某些线程因为被keepalive，空挂在那里等待，中间几乎没有请求过来，甚至等到超时），event MPM中，会有一个专门的线程来管理这些keepalive类型的线程，当有真实请求过来的时候，请求传递给服务线程，执行完毕后，有允许它释放，这样增强了高并发场景下的请求处理能力。
- 优点：单线程响应多请求，占据更少的内存，高并发下表现更优秀，会有一个专门的线程来管理keep-alive类型的线程，当有真实请求过来的时候，将请求传递给服务线程，执行完毕后，有允许它释放。
- 缺点：没有线程安全控制



# Apache2的安装和使用

## Apache介绍

- Apache通常可以指代两个相关但不同的概念，分别是Apache软件基金会和Apache HTTP Server
- Apache2的特性
  - 高度模块化: core + modules
  - DSO: Dynamic Shared Object 动态加载/卸载
  - MPM: multi-processing module 多路处理模块
- apache2功能
  - 虚拟主机: IP, Port, FQDN[完全限定域名 (Fully Qualified Domain Name) ]
  - CGI: Common Gateway Interface 通用网关接口
  - 反向代理
  - 负载均衡
  - 路径别名
  - 丰富的用户认证机制: basic, digest
  - 支持第三方模块
- apache2-2.4新特性
  - MPM支持运行为DSO机制，以模块形式按需加载
  - eventMPM生产环境可用
  - 异步读写机制
  - 支持每模块及每目录的单独日志级别定义
  - 每请求相关的专用配置
  - 增强版的表达式分析式
  - 毫秒级持久连接时长定义
  - 基于FQDN的虚拟主机不需要NameVirtualHost指令
  - 新指令
  - 支持用户自定义变量
  - 更低的内存消耗

## 安装Apache2

- Ubuntu安装

```
apt install -y apache2
```

- Rocky安装

```
yum install -y httpd
```

# Apache2配置和使用

## Apache2的配置说明

- Apache2的配置主要分为三部分，分别是全局配置，虚拟主机配置，模块配置，这些配置分散在不同的目录和文件中，在主配置文件中以文件包含的形式引用它们
- 配置文件中每一行包含一个配置指令，并指定值，配置指令不区分大小写，但值区分大小写
- 以#开头的行是注释行，注释不能出现在指令的后边，空白行和指令前的空白字符将被忽略，因此可以采用缩进以保持配置层次的清晰

```
# 配置文件语法检查  
apachectl configtest  
apachectl -t
```

- 参考文档

```
http://httpd.apache.org/docs/2.4/ # 文档  
https://httpd.apache.org/docs/2.4/zh-cn/mod/quickerferenct.html # 指令索引
```

- 配置文件

```
# Ubuntu  
[root@mystical ~] $ ll /etc/apache2/  
total 88  
drwxr-xr-x  8 root root  4096 Apr 21 06:50 ./  
drwxr-xr-x 97 root root  4096 Apr 21 06:50 ../  
-rw-r--r--  1 root root  7224 Apr 10 17:45 apache2.conf          # 主配置文件  
drwxr-xr-x  2 root root  4096 Apr 21 06:50 conf-available/      # 子配置文件目录  
drwxr-xr-x  2 root root  4096 Apr 21 06:50 conf-enabled/        # 生效的自配置文件，  
链接到conf-available中  
-rw-r--r--  1 root root  1782 Dec  4 18:58 envvars             # 全局环境变量配置文  
件  
-rw-r--r--  1 root root 31063 Dec  4 18:58 magic                # 配合  
mod_mime_magic模块判断MIME类型的配置文件  
drwxr-xr-x  2 root root 12288 Apr 21 06:50 mods-available/      # 可用的模块配置文件  
drwxr-xr-x  2 root root  4096 Apr 21 06:50 mods-enabled/        # 生效的模块配置文件  
-rw-r--r--  1 root root   320 Dec  4 18:58 ports.conf           # 默认端口配置文件  
drwxr-xr-x  2 root root  4096 Apr 21 06:50 sites-available/      # 可用的虚拟主机配置  
文件  
drwxr-xr-x  2 root root  4096 Apr 21 06:50 sites-enabled/        # 生效的虚拟主机配置  
文件，连接到 sites-available 中  
  
# 主配置文件中的包含  
cat /etc/apache2/apache2.conf | grep "^Include"  
IncludeOptional mods-enabled/*.load  
IncludeOptional mods-enabled/*.conf  
Include ports.conf  
IncludeOptional conf-enabled/*.conf  
IncludeOptional sites-enabled/*.conf  
  
# 全局配置  
  
# Rocky
```

```
[root@localhost ~] $tree /etc/httpd/
/etc/httpd/
├── conf
│   ├── httpd.conf
│   └── magic
├── conf.d
│   ├── autoindex.conf
│   ├── README
│   ├── userdir.conf
│   └── welcome.conf
└── conf.modules.d
    ├── 00-base.conf
    ├── 00-dav.conf
    ├── 00-lua.conf
    ├── 00-mpm.conf
    ├── 00-optional.conf
    ├── 00-proxy.conf
    ├── 00-systemd.conf
    ├── 01-cgi.conf
    ├── 10-h2.conf
    ├── 10-proxy_h2.conf
    └── README
├── logs -> ../../var/log/httpd
├── modules -> ../../usr/lib64/httpd/modules
├── run -> /run/httpd
└── state -> ../../var/lib/httpd
```

- Ubuntu的配置文件引用规则

所有的配置文件在**conf-available**目录下

但是想使其生效，需要在**conf-enabled**目录下创建软连接，引用**conf-available**目录下的配置文件

因为主配置文件**apache2.conf**里面包含的是**conf-enabled**

```
# 模块mods同理
```

- 注意

```
# 刚下载好的apache2，检测配置文件语法会报错，原因是没有在配置文件添加"ServerName ubuntu"
[root@mystical ~] $apachectl -t
AH00558: apache2: Could not reliably determine the server's fully qualified
domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress
this message
Syntax OK

# 添加后即成功
[root@mystical /etc/apache2] $echo "ServerName ubuntu" >> apache2.conf
```

## 常用全局配置

- 主配置文件中的指令对整个服务器都生效，如果只想改变某一部分的配置，则可以把指令嵌入到 `<Directory>`、`<directory>`、`<Files>`、`<FilesMatch>`、`<Location>`、`<LocationMatch>` 配置段中，这样就可以限制指令的作用域为文件系统中的某些位置或特定的URL，这些配置段还可以进行嵌套，已进行更精细的配置
- Apache2还具备同时支持多个站点的能力，称为虚拟主机，`<virtualHost>` 配置段中的指令仅对该段中的特定站点（虚拟主机）有效
- 虽然大多数指令可以包含在任意的配置段中，但是某些指令仅在某些特定的范围内才有意义，一个指令可以被作用于哪些配置段中，取决于该指令的作用域
- 作用域：表示该指令出现在配置文件中的什么位置才是合法的

```
server config          # 该指令全局生效，不能写在<virtualHost>, <Directory>,
.htAccess中

virtual host           # 可以写在<virtualHost>中

directory              # 可以写在<Directory>, <Location>, <Files>, <Proxy>中

.htaccess               # 可以写在目录及其子目录的.htaccess文件中，但可能会因overrides的设
置而不起作用
```

- 常用配置项（作用域列说明：S代表server config, V代表virtual host, D代表directory, H代表.htaccess）
  - 略
- 主配置文件

```
[root@mystical /etc/apache2] $cat /etc/apache2/apache2.conf|grep -Ev "^#|^$"
DefaultRuntimeDir ${APACHE_RUN_DIR}          # 默认运行目录
PidFile ${APACHE_PID_FILE}                  # PID文件目录
Timeout 300                                # 超时时长默认300s
KeepAlive On                                 # 默认开启keepalive
MaxKeepAliveRequests 100                     # keepalive保持的最大连接数
KeepAliveTimeout 5                          # keepalive 超时时长5s
User ${APACHE_RUN_USER}                      # 进程属主
Group ${APACHE_RUN_GROUP}                    # 进程属组
HostnameLookups off                         # 错误日志路径
ErrorLog ${APACHE_LOG_DIR}/error.log        # 日志级别
LogLevel warn                               # 文件包含 先加载模块，再开启模块才能使
用该模块
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
Include ports.conf
<Directory />                            # 目录权限设置
    Options FollowSymLinks
    AllowOverride None
    Require all denied
</Directory>
<Directory /usr/share>                   # 目录权限设置
    AllowOverride None
    Require all granted
</Directory>
```

```

<Directory /var/www/>                                # 目录权限设置-默认网站配置, 默认
DocumentRoot 是 /var/www/html, 从此处继承
    # Indexes: 如果目录下文件没有默认打开的文件, 路径上又没有指明, 则列出目录索引
    # 如果没有Indexes选项, 则不会列出目录索引 (危险操作)
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>
AccessFileName .htaccess
<FilesMatch "^\\.ht">
    Require all denied
</FilesMatch>
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %o \"%{Referer}i\" \"%{User-Agent}i\""
vhost_combined
LogFormat "%h %l %u %t \"%r\" %>s %o \"%{Referer}i\" \"%{User-Agent}i\" combined
LogFormat "%h %l %u %t \"%r\" %>s %o" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
IncludeOptional conf-enabled/*.conf
IncludeOptional sites-enabled/*.conf
ServerName ubuntu                                     # 主机名

```

## 默认虚拟主机配置文件

```

[root@mystical /etc/apache2] $cat sites-enabled/000-default.conf |grep -Ev "#|^$"
<virtualHost *:80>                               # 监听本机所有IP:80端口
    ServerAdmin webmaster@localhost               # 管理员邮箱
    DocumentRoot /var/www/html                   # 网站根目录
    ErrorLog ${APACHE_LOG_DIR}/error.log          # 错误日志
    CustomLog ${APACHE_LOG_DIR}/access.log combined # 访问日志路径和格式
</virtualHost>

```

## 全局默认页面配置文件

```

[root@mystical /etc/apache2] $cat /etc/apache2/mods-enabled/dir.conf |grep -Ev "#|^$"
<IfModule mod_dir.c>
    # 表示目录下默认打开文件的优先级
    DirectoryIndex index.html index.cgi index.pl index.php index.xhtml
    index.htm
</IfModule>

# 删除默认目录下所有文件, 在浏览器中查看, 能看到列出来当前目录
rm -f /var/www/html/*

```

## Listen 配置监听IP和端口

- Listen选项用来配置Httpd服务监听的IP地址和端口, 可以写多条

```

# 默认监听本机所有IP和80端口
cat /etc/apache2/ports.conf
Listen 80
# 可以指定多个端口, 也可以指定IP和端口
Listen 8080

```

```
Listen 192.168.2.1:80
```

```
<IfModule ssl_module>
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

## 配置默认内容目录

- (该实验需要先学习访问资源控制)

```
# 在全局配置文件apache2.conf的最后添加
DocumentRoot /data/www/

# 然后修改sites-enabled/000-default.conf
注释掉DocumentRoot /var/www/html

# 保存后重载
mkdir -pv /data/www ; systemctl reload apache2

# 此时会发现，返回403，原因是apache对新创建的/data/www没有权限，需要给该目录创建权限，才能让
apache访问

# 即在访问资源控制的章节会学习如何解决权限问题
```

## 配置数据压缩

- 启用apache2的数据压缩功能能减少网络IO的数据量，提高服务器的网络吞吐能力
- 默认已开启数据压缩

```
Request Headers 客户端的请求头中，会有
Accept-Encoding: gzip, deflate 字段（客户端要求服务端压缩）
```

```
# 需要模块支持，默认已加载
apache2ctl -M | grep deflate
[root@mystical /etc/apache2] $ apache2ctl -M |grep deflate

deflate_module (shared)
```

- 如果服务端没有压缩的时候，在响应头中会有

```
content-length，即会有数据字符长度
```

```
# 压缩后就看不到长度了
```

- 客户端请求压缩---->服务端压缩过程

```
curl http://10.0.0.150/syslog.html
```

浏览器：

1. 服务端收到请求，读出来，客户端支持压缩
2. 请求的资源是html，根据配置文件（MIME类型），html类型资源支持压缩  
# 所以就压缩了

- 查看或设置支持压缩的资源类型

- 打开 Apache 配置文件：位置可能会根据安装和操作系统的不同而异。通常的文件路径可能是 /etc/httpd/httpd.conf、/etc/apache2/apache2.conf 或 /etc/apache2/sites-available/000-default.conf 等。
- 查找 mod\_deflate 的配置：在配置文件中查找 AddOutputFilterByType DEFLATE。这行命令后面会列出支持压缩的 MIME 类型。

```
[root@mystical /etc/apache2/mods-enabled] $ cat deflate.conf
<IfModule mod_deflate.c>
    <IfModule mod_filter.c>
        AddOutputFilterByType DEFLATE text/html text/plain text/xml
        text/css text/javascript
        AddOutputFilterByType DEFLATE application/x-javascript
        application/javascript application/ecmascript
        AddOutputFilterByType DEFLATE application/rss+xml
        AddOutputFilterByType DEFLATE application/wasm
        AddOutputFilterByType DEFLATE application/xml
    </IfModule>
</IfModule>
# 这些行指定了哪些资源类型（基于 MIME 类型）将被 mod_deflate 压缩
# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

- 修改或添加配置：如果需要添加更多类型的压缩支持，可以按照上面的格式添加新的行。确保在进行任何修改后重启 Apache 服务
- 使用 curl 在请求包中指定压缩测试

```
curl 10.0.0.150/syslog.html -vI --compressed
```

## ServerTokens 配置响应头中的 Server 字段信息

- ServerTokens 选项用来配置 HTTP 响应头中的 Server 字段信息，其取值如下

FULL	Server:Apache/2.4.2(Unix) PHP/4.2.2MyMod/1.2
OS	Server:Apache/2.4.2(Unix)
Minimal	Server:Apache/2.4.2
Minor	Server:Apache/2.4
Major	Server:Apache/2
Prod	Server:Apache # 生产环境下使用

- Server 信息所在配置文件

```
cat /etc/apache2/conf-enabled/security.conf

# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of: Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
#ServerTokens Minimal
ServerTokens OS
#ServerTokens Full
```

## 配置持久连接

- 持久连接是指：建立连接后，每个资源获取完后不会立即断开连接，而是保持一段时间后断开，默认是开启了持久连接
- 配合持久连接的有一个连接限制选项，单位为秒，apache2.4之后支持毫秒级设置
- 对并发量大的服务器，持久连接会使一些请求得不到响应

```
# 默认设置
cat /etc/apache2/apache2.conf | grep -vE "^#" | grep -i "KeepAlive"

[root@mystical /etc/apache2] $cat apache2.conf|grep -i "Keepalive"
# KeepAlive: whether or not to allow persistent connections (more than
KeepAlive On          # 默认开启了持久连接
# MaxKeepAliveRequests: The maximum number of requests to allow
MaxKeepAliveRequests 100   # 在一个持久连接内，最多可以立即处理100次请求
# KeepAliveTimeout: Number of seconds to wait for the next request from the
KeepAliveTimeout 5        # 持久连接保持时间
```

- 关于长连接的测试使用telnet

```
[root@localhost ~] $telnet 10.0.0.180 80
Trying 10.0.0.180...
Connected to 10.0.0.180.
Escape character is '^>'.
GET /a.html HTTP/1.1
host: 1.2.3.4

HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 12:20:57 GMT
Server: Apache/2.4.52 (Ubuntu)
Last-Modified: Sun, 21 Apr 2024 12:19:41 GMT
ETag: "5-6169a55e1030f"
Accept-Ranges: bytes
Content-Length: 5
Content-Type: text/html

aaaa
GET /b.html HTTP/1.1
host: 1.2.3.4

HTTP/1.1 200 OK
```

```
Date: Sun, 21 Apr 2024 12:21:17 GMT
Server: Apache/2.4.52 (Ubuntu)
Last-Modified: Sun, 21 Apr 2024 12:19:47 GMT
ETag: "5-6169a5638c64a"
Accept-Ranges: bytes
Content-Length: 5
Content-Type: text/html

bbbb
Connection closed by foreign host. # 15s后才断开连接
```

```
# 如果把keepalive参数改为off
[root@localhost ~] $telnet 10.0.0.180 80
Trying 10.0.0.180...
Connected to 10.0.0.180.
Escape character is '^].
GET /a.html HTTP/1.1
host: 1.2.3.4
```

```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2024 12:23:04 GMT
Server: Apache/2.4.52 (Ubuntu)
Last-Modified: Sun, 21 Apr 2024 12:19:41 GMT
ETag: "5-6169a55e1030f"
Accept-Ranges: bytes
Content-Length: 5
Connection: close
Content-Type: text/html
```

```
aaaa
Connection closed by foreign host. # 完成一次请求响应后直接断开
```

## 定义路径别名

- Alias指令用于在URL和文件系统之间实现映射，使不在DocumentRoot目录下的内容也能成为项目的一部分

```
Alias [URL-path] file-path|directory-path

# 以URL-path开头的路径在访问时将被映射到后面的文件系统路径，URL-path区分大小写
# 作用域: Server config, virtual host, directory

# 此指令配合Alias，不使用URL-path参数时使用，默认值OFF，作用域同上
AliasPreservePath OFF|NO

#ON      完整映射，每个资源都单独存在
#OFF     整体映射，被命中的资源都映射同一个资源
```

- 实例

```
# 在主配置文件apache2.conf中，最后面添加
Alias "/css/" "/var/www/html/source/css/"

# 之后测试
[root@mystical /etc/apache2] $curl 10.0.0.180/css/test.css
hello this test
```

- 如果alias定义的目录在DocumentRoot之外，则需要单独授权，否则不可用
  - 别名中定义的URL-path如果不是以/结尾，则后面的file-path也不能以/结尾

```
Alias "/test" "/alias/test"
```

## 访问资源控制(权限管理)

```
# 修改主配置文件apache2.conf
<Directory /data/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
    # all granted 表示都可以访问
    # all denied 表示都不能访问

</Directory>

DocumentRoot /data/www/

# 在ports.conf里添加一个监听端口
Listen 8080

# 在sites-enabled/000-default.conf中添加端口对应的页面
<virtualHost *:8080>
    ServerAdmin webmaster@localhost
    DocumentRoot /data/www
</virtualHost>

# 重启后测试
systemctl reload apache2
[root@mystical /etc/apache2] $curl 127.1:8080
<h1>Hello world</h1>
# 测试成功
```

- 常用配置指令，要写在限定的路径或URL规则下

```
https://httpd.apache.org/docs/2.4/mod/core.html#options
https://httpd.apache.org/docs/2.4/mod/core.html#allowoverride
https://httpd.apache.org/docs/2.4/zh-cn/mod/mod\_authz\_core.html#require
```

- Options

```

Options [+|-]option [[+|-]option]
# 指定目录选项，可以同时有多个值，作用域server config, virtual host, directory,
.htaccess
# Indexes 显示目录内容列表（类似阿里镜像源页面）
# Includes 允许使用`mod_include`模块提供的服务器端包含功能
# FollowSymlinks 可以跟随软连接，默认值
# ExecCGI 允许使用`mod_cgi`模块执行CGI脚本
# Multiviews 允许使用`mod_negotiation`模块提供内容协商的多重视图
# None 不支持任何特性

# apache2允许一个目录被配置多次options，如果一个目录被设置了多条options，则生效最下面的那条，其他options中指定的值不会被合并
# 如果在值前面使用了+，则一定被合并，使用-，则一定被去除，使用-Include*，不论前面如何设置，都会完全禁用服务器端包含

```

- AllowOverride

```

AllowOverride All|None|directive-type [directive-type]
# 作用域 directory
# 允许在.htaccess文件中指定的指令类型，文件名不一定是.htaccess，由AccessFileName选项指定
# 仅在不包含正则表达式的<directory>配置段才是有效的
# 在<Localtion>, <DirectoryMatch>, <Files>配置段中都是无效的
# ALL 所有指令都可以在.htaccess文件中生效
# None 不使用.htaccess文件中的指令
# AllowOverride AuthConfig 只能在.htaccess配置AuthConfig相关指令
# AllowOverride Options=Followsmlinks Indexes 可以在.htaccess文件中配置Followsmlinks
Indexes

```

- Require

```

Require [not] entity-name [entity-name]
# 作用域 directory, .htaccess
# 用于指定哪些用户允许访问哪些资源
# 某些情况下需要配合AuthName, AuthType, AuthUserFile, AuthGroupFile指令一起使用（认证模块）
# 上述认证模块，后面有详解
# Require all granted          所有用户都可以访问
# Require all denied           所有用户都不能访问
# Require method http-method [http-method]    特定请求方法可以访问
# Require user userid         特定用户可以访问
# Require ip 10 172.20 192.168.2    指定IP可以访问
# Require forward-dns dynamic.example.org    指定主机名或域名可以访问

```

- 例外事项

- AllowOverride指令只能出现在<directory>段中
- Option指令不能用于<Files>和<FilesMatch>段
- options中的FollowSymLinks和SymLinksIfOwnerMatch只能出现在<directory>段或者.htaccess文件中
- 基于文件系统路径的访问控制

```
# 基于目录-路径匹配
```

```

<Directory "/path">
    #...
</Directory>

# 基于目录-扩展正则匹配
<Directory ~ "regex">
    #...
</Directory>

# 基于目录-扩展正则匹配
<DirectoryMatch "regex">
    #...
</DirectoryMatch>

# 基于文件-文件路径匹配
<Files "/path/file">
    #...
</Files>

# 基于文件-通配符
<Files "/path/*file*">
    #...
</Files>

# 基于文件-扩展正则匹配
<Files ~ "regex">
    #...
</Files>

#基于正则表示-扩展正则
<FilesMatch "regex">
    #...
<FilesMatch>

# 范例
<FilesMatch "^\\.ht">
    Require all denied
<FilesMatch>

# 基于正则，文件后缀名控制
<FilesMatch ".+.(gif|jpe?g|png)$">
    options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</FilesMatch>

```

- 基于URL路径的访问控制

```

# 基于URL路径
<Location "/URL">
    #...
</Location>

# 基于URL路径-扩展正则
<Location ~ "regex">
    #...
</Location>

```

- 控制特定IP访问

```

# RequireAll      允许所有客户端，拒绝指定主机，拒绝优先，作用域directory, .htaccess
# RequireAny      拒绝所有客户端，允许指定主机，允许优先，作用域同上
<Directory "/data/www/dira/">
    <RequireAny>
        Require all denied          # 拒绝所有客户端
        Require ip 10.0.0.207       # 允许10.0.0.207,可以写成网段
    </RequireAny>
</Directory>

```

## 日志配置

- apache2中的日志设置包括错误日志和访问日志两类

```

[root@mystical /etc/apache2] $cat apache2.conf|grep -i log|grep -Ev "^#"
ErrorLog ${APACHE_LOG_DIR}/error.log
LogLevel warn
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %o \"%{Referer}i\" \"%{User-Agent}i\""
vhost_combined
LogFormat "%h %l %u %t \"%r\" %>s %o \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %o" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# User-agent    # 用户代理，浏览器头
# Referer       # 前一个页面，表示是从哪一个页面跳转过来

# 没有被虚拟主机配置命中的访问的访问日志
[root@mystical /etc/apache2] $cat conf-enabled/other-vhosts-access-log.conf |grep -Ev "^$|^#"
CustomLog ${APACHE_LOG_DIR}/other_vhosts_access.log vhost_combined

# 默认虚拟主机日志设置
[root@mystical /etc/apache2] $cat sites-enabled/000-default.conf |grep -i log
    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # It is also possible to configure the loglevel for particular
    #LogLevel info ssl:warn
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

# 默认日志
/var/log/apache2/access.log
/var/log/apache2/error.log

```

```
/var/log/apache2/other_vhosts_access.log
```

- 自定义访问日志格式并使用
  - 增加日志格式
  - 在配置文件中引用自定义格式

```
# 在/etc/apache2/apache2.conf 中增加日志格式定义
LogFormat "%h %l %u test-format %{User-agent}i" test-format

# 修改 /etc/apache2/conf-enabled/other-vhosts-access-log.conf
CustomLog ${APACHE_LOG_DIR}/other_vhosts_access.log test-format

# 重启服务测试
```

## 状态页配置

```
# 状态页需要相关模块的支持
apachectl -M | grep status
status_module (shared)

# 默认已开启状态页配置
[root@mystical /etc/apache2] $cat mods-enabled/status.load
LoadModule status_module /usr/lib/apache2/modules/mod_status.so

[root@mystical /etc/apache2] $cat mods-enabled/status.conf| grep -Ev "^.*/#|^$"
<IfModule mod_status.c>
    <Location /server-status>          # 可以自定义路径
        SetHandler server-status         # 由此处决定上一行中定义的地址是查看状态
        Require local                   # 默认只能本地访问，修改此处为all
granted,在浏览器测试
    </Location>
    ExtendedStatus On                # 详细显示
    <IfModule mod_proxy.c>
        ProxyStatus On
    </IfModule>
</IfModule>

访问10.0.0.180/server-status
```

## 虚拟主机配置

- apache2服务支持在一台物理服务器上配置多个网站，我们可以通过虚拟主机配置来实现该功能
- 在apache2服务中配置了虚拟主机，还需要在DNS解析中添加域名解析，将域名指向apache2服务器所在的IP地址
- DNS解析负责将域名解析到apache2服务器上，虚拟主机配置负责区分不同的网站，将不同的域名与内容分别对应起来
- 多虚拟主机的实现方式
  - 基于端口实现：用不同的端口标识不同的虚拟主机
  - 基于IP实现：用不同的IP地址标识不同的虚拟主机
  - 基于域名实现：用不同的域名标识不同的虚拟主机

- 虚拟主机配置文件 /etc/apache2/sites-enabled/000-default.conf
- 配置基于端口区分的虚拟主机

```
cat /etc/apache2/sites-enabled/vhost.conf # 自行创建
Listen 81
Listen 82
Listen 83

<virtualHost *:81>

    ServerAdmin webmaster@localhost
    DocumentRoot /data/www/web1/
    <Directory /data/www/web1/>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</virtualHost>

<virtualHost *:82>

    ServerAdmin webmaster@localhost
    DocumentRoot /data/www/web2/
    <Directory /data/www/web2/>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</virtualHost>

<virtualHost *:83>

    ServerAdmin webmaster@localhost
    DocumentRoot /data/www/web3/
    <Directory /data/www/web3/>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</virtualHost>
```

- 配置基于IP区分的虚拟主机

```
# 在网卡添加多IP
ip a a 10.0.0.81/24 dev eth0
```

```
ip a a 10.0.0.82/24 dev eth0
ip a a 10.0.0.83/24 dev eth0

# 修改hosts文件，绑定域名和IP
10.0.0.81 www.a.com
10.0.0.82 www.a.com
10.0.0.83 www.a.com

# 修改配置文件，以IP地址来区分
cat /etc/apache2/sites-enabled/vhost.conf

<virtualHost 10.0.0.81:80>

    ServerAdmin webmaster@localhost
    DocumentRoot /data/www/web1/
    <Directory /data/www/web1/>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</virtualHost>
```

...同上

- 配置基于域名区分的虚拟主机

```
cat /etc/apache2/sites-enabled/vhost.conf

<virtualHost *:80>

    ServerAdmin webmaster@localhost
    DocumentRoot /data/www/web1/
    ServerName www.a.com
    <Directory /data/www/web1/>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</virtualHost>
```

...其它同上

## 错误页面配置

- 在HTTP协议中，响应报文都有相应的状态码，其中200表示正确返回，其它状态码都表示错误，可以在Apache2中设置根据不同的状态码设置不同的错误页面，也可以在虚拟主机的配置中为每个网站设置不同的错误页面
- 设置全局错误页面-字符串

```
ErrorDocument 403 "<h1>sorry! </h1> permission denied"
ErrorDocument 404 "<h1>sorry! your page not found</h1>"
```

- 设置全局错误页面-重定向

```
cat apache2.conf
```

```
Alias "/error/" "/var/www/html/error/" # 也许很重要，目前实验测试，不加不生效，待解决
ErrorDocument 403 "error/403.html"
ErrorDocument 404 http://10.0.0.208/404.html
```

- 在虚拟主机配置中设置

```
<virtualHost *:80>

    ServerAdmin webmaster@localhost
    DocumentRoot /data/www/web1/
    ServerName www.a.com

    ErrorDocument 403 "error/403.html"
    ErrorDocument 404 http://10.0.0.208/404.html
    ProxyErrorOverride on

    <Directory /data/www/web1/>
        Options FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</virtualHost>
```

## 访问验证配置

```
[root@mystical /usr/share/apache2] $htpasswd -c /usr/share/apache2/apache2-pwd
tom
New password:
Re-type new password:
Adding password for user tom

<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    #Require all granted
    AuthType Basic
    AuthName "Pls Input your name and pwd"
    AuthBasicProvider file
    AuthUserFile "/usr/share/apache2/apache2-pwd"
    Require user tom
</Directory>
```

## 模块配置和工作模式配置

- 列出所有模块

```
apachectl -M
```

- 列出所有核心模块

```
apachectl -l
```

## Apache2实现Https访问

- ssl功能需要相关模块支持， 默认没有加载此模块

```
# 加载模块
cd /etc/apache2/mod2-enabled
ln -sv ../mods-available/ssl.conf ./ssl.conf
ln -sv ../mods-available/ssl.load ./ssl.load
ln -sv ../mods-available/socache_shmcb.load ./socache_shmcb.load

# 添加虚拟主机配置
cd /etc/apache2/sites-enabled/
ln -sv ../sites-available/default-ssl.conf ./default-ssl.conf

# 配置文件内容
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        # SSL证书文件
        SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
        # 私钥文件
        SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
        # SSL证书链文件，上级CA证书
        #SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt
        # 顶级CA证书文件，与上面证书链文件二选一
        #SSLCACertificateFile /etc/apache2/ssl.crt/ca-bundle.crt
        <FilesMatch "\.(cgi|shtml|phtml|php)$">
            SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
            SSLOptions +StdEnvVars
        </Directory>
    </VirtualHost>
</IfModule>
```

## 自签证书实现多域名HTTPS

```
# 安装软件，比openssl使用简单
apt install easy-rsa -y

cd /usr/share/easy-rsa/

# 初始化证书目录
./easyrsa init-pki

# 查看生成的证书目录
[root@mystical /usr/share/easy-rsa] $tree pki
pki
├── openssl-easyrsa.cnf
├── private
├── reqs
└── safessl-easyrsa.cnf

2 directories, 2 files

# 生成CA机构证书，不使用密码
./easyrsa build-ca nopass

# 生成a.com的私钥和证书申请文件
./easyrsa gen-req a.com.server nopass

# 生成b.com的私钥和证书申请文件
./easyrsa gen-req b.com.server nopass

# 生成c.com的私钥和证书申请文件
./easyrsa gen-req c.com.server nopass

# 签发证书
./easyrsa sign-req server a.com.server

./easyrsa sign-req server b.com.server

./easyrsa sign-req server c.com.server

# 配置虚拟主机
cat /etc/apache2/sites-enabled/www.a.com.conf
```

# Nginx 高性能的Web服务端

## 影响用户体验的因素

### # 客户端

客户端硬件配置  
客户端网络速率  
客户端与服务端距离

### # 服务端

服务端网络速率  
服务端硬件设备  
服务端架构设计  
服务端应用程序工作模式  
服务端并发数量  
服务端应文件大小及数量**buffer cache**  
服务端I/O压力

## 服务端I/O流程

- 磁盘I/O
- 网络I/O
  - 网络通信就是网络协议栈到用户空间进程的IO就是网络IO
- 网络IO过程

客户端通过浏览器与服务端建立连接，发送请求，服务端接收请求

服务器通过网卡接收数据的请求，网卡收到请求后，使用DMA技术（直接内存访问），将收到的数据Copy到内存空间（内核缓冲区），该过程不需要CPU参与

内核收到数据后，Copy数据到web服务器的进程（用户空间），Nginx收到请求

此时，web服务进程无法直接从硬盘读取数据，因为用户空间没有能力直接访问硬件

因此web服务进程，发请求给内核，内核收到请求后（内核拥有特权指令），内核可以将硬盘上的数据Copy到内核缓冲区，然后再copy一份到web服务进程（用户空间）

web服务进程拿到数据后封装响应报文头，然后copy一份发给内核空间

内核空间封装IP，tcp头等，然后从网卡发给客户端

## I/O模型 (Unix网络编程中分为5类)

- I/O模型相关概念
  - 同步和异步（消息通知机制）：
    - 同步： synchronous, 被调用者并不提供事件的处理结果相关的通知信息，需要调用者主动询问事情是否处理完成
    - 异步： asynchronous, 被调用者通过状态，通知或回调机制主动通知调用者被调用者的运行状态（回调函数）
  - 阻塞和非阻塞

- 阻塞: blocking, 指IO操作需要彻底完成后才返回到用户空间, 调用结果返回之前, 调用者被挂起, 干不了别的事情
  - 非阻塞: nonblocking, 指IO操作被调用后, 立即返回给用户一个状态值, 而无需等待IO操作完成, 在最终的调用结果返回之前, 调用者不会被挂起, 可以去做别的事情。
- 网路IO模型
    - 阻塞型, 非阻塞型, 复用型, 信号驱动型, 异步
  - 阻塞型I/O模型 (blocking IO)
    - 阻塞IO模型是最简单的IO模型, 用户线程在内核进行IO操作时被阻塞
    - 用户线程通过系统调用read发起IO读操作, 由用户空间转到内核空间, 内核等到数据包到达后, 然后将数据拷贝到用户空间, 完成read操作, 应用程序需要等待read将数据读取到buffer后, 才继续处理接受的数据, 整个IO请求的过程中, 用户线程是被阻塞的, 这导致用户在发起IO请求时, 不能做任何事情, 对CPU的资源利用率不高
    - 优点: 程序简单, 在阻塞等待数据期间, 进程/线程挂起, 基本不会占用CPU资源
    - 缺点: 每个链接需要独立的进程/线程单独处理, 当并发请求量大时, 为了维护程序, 内存和线程切换开销大, apache的prefork使用的就是这种模式

用户在客户端使用浏览器访问web服务器, 向web服务器发起请求, 通过网络到达web服务器的网卡, 网卡收到后

阶段1: 这个请求数据先交给内核, 内核有个缓冲区

阶段2: 内核将该请求数据复制一份, 交给应用程序 (比如Nginx的缓冲区)  
(这里从内核空间, 到用户空间)

# 这里数据从客户端到服务端网卡, 涉及到网络IO通讯过程

# Nginx运行起来后, 会等待请求数据被发送到内核中 (阶段1)

当应用程序尝试从网络中接受数据的时候, 通过系统调用 (`recvfrom`), 让内核替该应用程序监听, 网络中是否有数据到达内核 (从网络中传输, 到内核收到数据需要一段时间)

当内核拿到数据, 将数据拷贝到用户空间

数据拷贝到用户空间后, 会返回一个成功的结果, 并告知应用程序数据已经准备好了

## 在整个过程中, 应用程序做了一个事, 即发送了一个请求给内核, 让内核去准备数据  
## 该应用程序发送了该请求给内核后, 会一直监听内核状态, 看内核是否有数据被准备好  
## 这个期间, 应用程序处于阻塞状态, 无法做其他任务, 只能等待数据返回

- 更加准确的说法 - 关于系统调用read()

**1.1** 当web服务器需要读取来自客户端的数据时，它会使用**read()**系统调用从与客户端通信的套接字文件描述符中读取数据

**1.2** 在调用**read()**时，web服务会提供文件描述符，一个缓冲区的地址（通常在用户空间中分配）以及要读取的最大字节数

**1.3** **read()**系统调用导致执行从用户模式切换到内核模式，内核接管控制，检查文件文件描述符，查看与之关联的资源，并尝试从该资源（如网络连接的接受缓冲区）中读取指定数量的数据

**2.1** 如果数据已经在内核的接受缓冲区中可用，内核会将这些数据从内核空间的缓冲区拷贝到**read()**调用中指定的用户空间缓冲区

**2.2** 一旦数据被拷贝，内核返回从资源中实际读取的字节数，并将控制返回给用户空间的web服务程序

**3.1** 如果没有可用数据，根据文件描述符的配置（阻塞或非阻塞），**read()**可能会阻塞调用进程，知道有数据可读，或者立即返回一个错误（通常是**EAGAIN**或**EWOULD\_BLOCK**），表明没有数据可读

**4.1** 完成数据传输后，web服务继续在用户空间执行，使用从**read()**调用中接受到的数据进行处理，如解析HTTP请求

- 从用户模式切换到内核模式的实现

- 系统调用指令

- 在X86架构上，可以使用**int 0x80** 或 **syscall** 指令来触发系统调用
    - 这些指令会将CPU从用户模式切换到内核模式，并跳转到操作系统预设的一个入口点（通常是一个固定的内存地址）

- 寄存器使用：

- 系统调用号（定义调用哪个系统函数，如**read**,**write**等）和参数通常通过特定的寄存器传递。例如，在x86架构上，**eax** 寄存器用于传递系统调用号，其它寄存器（如：**ebx**, **ecx**, **edx** 等）用于传递参数
    - 这要求系统调用之前必须正确设置这些寄存器

- 中断处理

- 当**syscall** 或 **int 0x80** 指令执行时，CPU会自动保存当前的程序计数器和其他状态信息到系统栈上，并从一个预设的地址加载新的程序计数器，这通常指向内核的一个中断服务例程。
    - 内核的中断服务例程会根据传入的系统调用号分派到对应的内核函数

- 示例汇编代码（X86架构，使用**int 0x80**）

```
; eax = 系统调用号
; ebx, ecx, ... = 系统调用参数
mov eax, 4      ; 系统调用号为4 (sys_write)
mov ebx, 1      ; 文件描述符 1 (stdout)
mov ecx, msg    ; 消息的地址
mov edx, len    ; 消息长度
int 0x80        ; 出发中断，继续系统调用

; 消息数据
msg db 'Hello, world!', 0xA ; 消息文本加换行符
len equ $ - msg             ; 消息长度

; 从内核模式返回用户模式
; 完成系统调用后，内核会将控制权返回给用户程序，这通常涉及到恢复用户的CPU寄存器状态和程序计数器，然后执行iret指令（在X86上）来从中断返回
```

## • 非阻塞IO模型 (nonblocking IO)

- 用户线程发起IO请求时，立即返回，但并未读取到任何数据，用户线程需要不断的发起IO请求，直到数据到达后，才能真正读取到数据，继续执行。即“轮询”机制
- 存在两个问题
  - 如果大量文件描述符都要等，那么就得一个一个的read，这会带来大量的Context Switch (read是系统调用，每调用一次就得在用户态和内核态间切换)，造成大量资源消耗（非阻塞：程序向内核状态发送IO请求后等待内核响应，如果内核处理请求和IO操作不能立即返回IO结果，进程将不再等待，而是继续处理其他请求，但是仍然需要进程隔一段时间就查看内核IO是否完成）

这里，数据从客户端到服务端内核这个阶段不发生阻塞，但是当检测到服务端内核收到数据后，从内核复制数据到用户空间buff这个过程中仍发生阻塞表面上第一阶段不阻塞，实际上由于要不断轮询切换上下文，导致资源开销反而变大

## • I/O多路复用型 (I/O multiplexing)

- 上面的模型中，每一个文件描述符对应的IO是由一个线程监控和处理，即如果一个IO流进来，就开启一个进程处理这个IO流。那么假设现在有一百万个IO流进来，那就需要开启一百万个进程一一对应处理这些IO流（这就是传统意义上的多进程并发处理）。一百万个进程，CPU占有率会很高，所以提出了I/O多路复用这个模型，一个线程，通过记录IO流的状态来同时管理多个IO，可以提高服务器的吞吐能力
- 多路复用IO指一个线程可以同时（实际是交替实现，即并发完成）监控和处理多个文件描述符对应各自的IO，即复用同一个线程
- 一个线程之所以能实现同时处理多个IO，是因为这个线程调用了内核中的SELECT，POLL或EPOLL等系统调用，从而实现多路复用IO
- I/O常见实现：Nginx支持在多种不同的操作系统实现不同的事件驱动模型，但是其在不同的操作系统甚至不同的系统版本上的实现方式不尽相同，主要有以下几种方式

### 1. select

select库是在linux和windows平台都基本支持的事件驱动模型库，并且在接口的定义也基本相同，只是部分参数的含义略有差异，最大并发限制1024，是最早的事件驱动模型，I/O多路复用这个概念被提出来以后，select于1983年第一个在BSD里面实现

select 在处理多个多路复用的IO的时候，采用的是遍历机制，所以效率很低，是反复循环遍历，所以IO的线路越多，遍历处理的时间越长

select 底层基于数组实现

最大连接数 1024(x86)； 2048(x64)

select fd拷贝：每次调用select都需要把fd集合从用户拷贝到内核态

### 2. POLL:

在Linux的基本驱动模型，Windows不支持此驱动模型，是select的升级版，取消了最大并发限制，在编译Nginx的时候可以使用--with-poll\_module和--without-poll\_module这两个指定是否编译select库。poll出现在1997年

poll 也是遍历机制

poll 底层是基于链表实现

最大连接数：无上限

`poll` fd拷贝：每次调用`select`都需要把fd集合从用户拷贝到内核态

### 3. EPOLL

`epoll`库是Nginx服务器支持的最高性能的事件驱动库之一，是公认的非常优秀的事件启动模型，它和`select`和`poll`有很大区别，`epoll`是`poll`的升级版，但是与`poll`有很大区别，在2002年Davide Libenzi 实现了`epoll`

`epoll`的处理方式是创建一个待处理的事件列表，然后把这个列表发给内核，返回的时候，再去轮询这个表，以判断事件是否发生，`epoll`支持一个进程打开的最大事件描述符的上限是系统可以打开的文件的最大数，同时`epoll`库的I/O效率不随描述符数目增加而线性下降，因为它只会对内核上报的活跃的描述符进行操作

`epoll` 操作方式是基于回调函数实现

`epoll` 底层基于哈希表实现

`epoll` 调用`epoll_ctl`时拷贝进内核并保持，之后每次`epoll_wait`不拷贝

### • 信号驱动式IO模型 (signal-drivenIO)

- 信号驱动I/O的意思就是进程现在不用傻等着，也不用轮询，而是让内核在数据就绪时，发送信号通知进程。
- 调用的步骤是：通过系统调用`sigaction`，并注册一个信号处理的回调函数，该调用会立即返回，然后主程序可以继续向下执行，当有I/O操作准备就绪，即内核数据就绪时，内核会为该进程产生一个SIGIO信号，并回调注册的信号回调函数，这样就可以在信号回调函数中系统调用`recvfrom`获取数据，将用户进程所需数据从内核空间拷贝到用户空间
- 此模型的优势在于等待数据报到达期间进程不被阻塞，用户主程序可以继续执行，只要等待来自信号处理函数的通知
- 在信号驱动式I/O模型中，应用程序使用套接字接口进行信号驱动I/O，并安装一个信号处理函数，进程继续运行不阻塞

依然分两个阶段

在用户进程调用系统调用通知内核的时候不阻塞，也不轮询，而是继续向下执行

第二阶段，内核在收到数据，会主动发信号给用户进程，此时用户进程收到信号，会将数据内核空间拷贝到用户空间，这个阶段仍会发生阻塞

### • 异步IO模型 (asynchronous IO)

- 异步I/O与信号驱动I/O最大的区别在于，信号驱动式内核通知用户进程何时开始I/O操作，而异步I/O是由内核通知用户进程I/O操作何时完成，两者由本质区别。

实现了两个阶段都没有阻塞发生，数据接受，到数据拷贝都交给内核处理，

## I/O 选择

- 完全跨平台：可以使用`select`, `poll`, 但是性能较差
- 针对不同操作系统自行选择支持的最优技术，比如：Linux选用`epoll`, Mac选择`kqueue`, Windows选择`IOCP`, 提高I/O处理的性能
- 最大并发连接数和内存有直接关系

## 单个进程能够监视的文件描述符的最大数量，是被内核限制

- 解决方法：

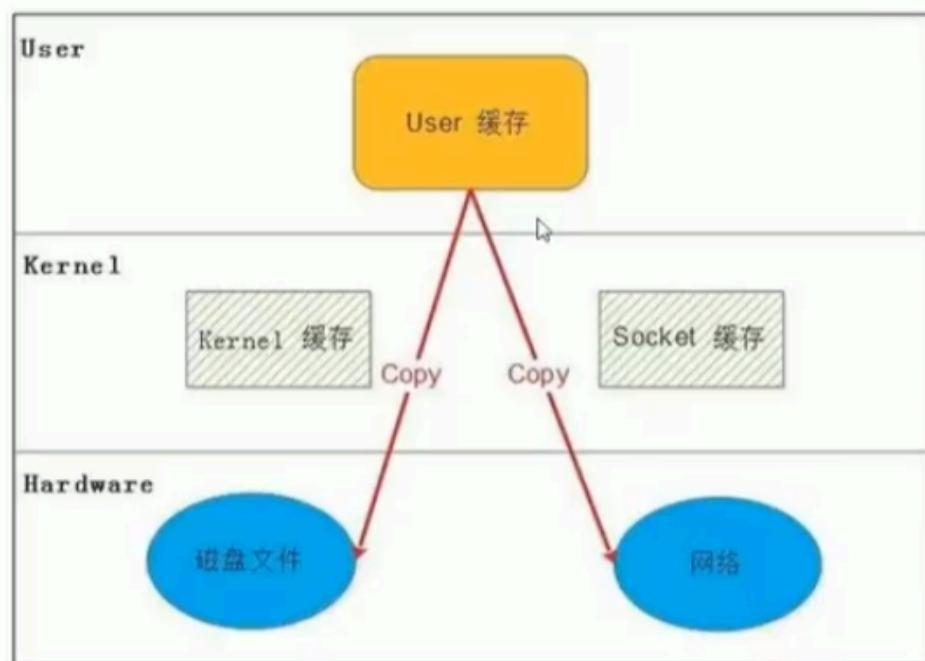
```
grep -R FD_SETSIZE linux-5.8/*
```

```
#define __FD_SETSIZE 1024 # 单个进程能够监视的文件描述符的最大数量
// 更改内核中的源代码后，重新编译，才能使用
// 不难，有时间搞一下（问题是改了之后，性能不一定变好，牵一发而动全身。；）
```

## 零拷贝

### 零拷贝介绍

- 传统Linux中I/O的问题
  - 传统的Linux系统的标准I/O接口（read, write）是基于数据拷贝的，也就是数据都是copy\_to\_user或者copy\_from\_user，这样做的好处是，通过中间缓存的机制，减少磁盘I/O的操作，但是坏处也很明显，大量数据的拷贝，用户态和内核态的频繁切换，会消耗大量的CPU资源，严重影响数据传输的性能，统计表明，在Linux协议中，数据包在内核态和用户态之间的拷贝所用的时间甚至占到了数据包整个处理流程时间的57.1%



## # 传统I/O的过程解析

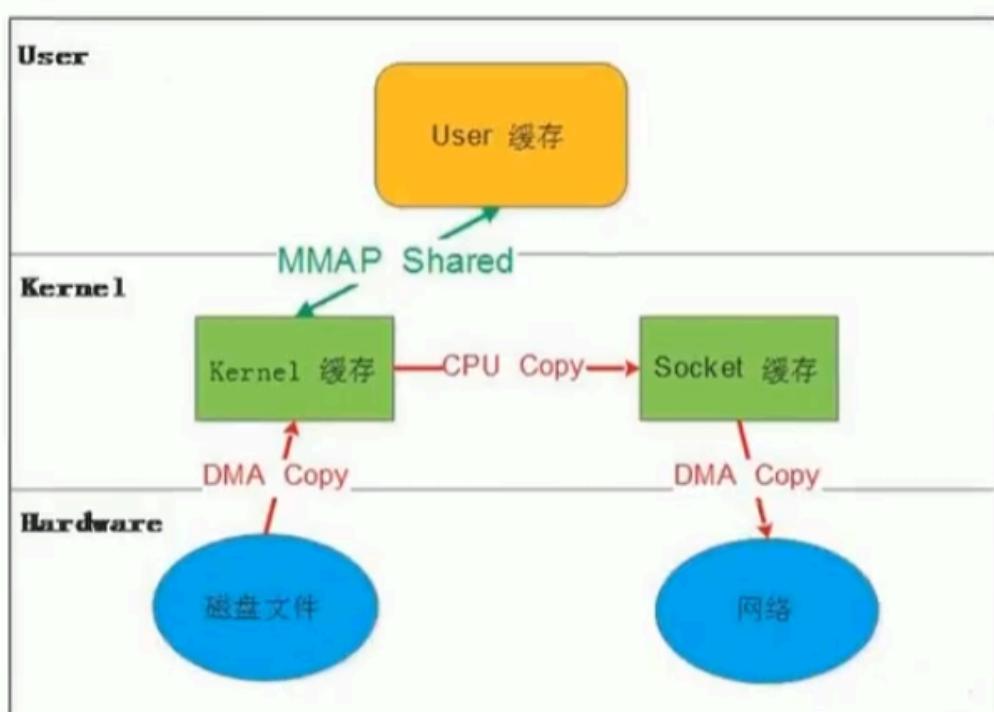
用户发送请求，通过网路到达服务器  
从网络将数据复制到内核空间的socket缓冲区  
通过socket缓冲区复制到用户空间（比如nginx）  
这里比如用户发送的是get请求，申请一个html文件  
此时需要将文件从硬盘拷贝到和用户空间的Nginx进行报文封装  
这个过程需要应用向内核发送请求，让内核从硬盘读取html文件，然后复制到用户空间  
用户空间的Nginx无法直接访问硬盘  
因此只能借助内核，通过系统调用告诉内核获取硬盘文件  
内核获取用户空间的系统调用后，发送一个DMA指令（直接内存访问），将磁盘上的文件复制到内核的缓冲区，然后再把数据从内核空间的缓冲区复制到用户空间的缓冲区  
此时Nginx拿到文件，封装http的响应报文头，然后发送到内核的socket缓冲区，然后再发送到网络中（期间还要加TCP报文头，IP报文头，MAC地址等报文...）  
这个操作涉及到频繁的内核空间和用户空间的切换

## • 什么是零拷贝

- 零拷贝就是上述问题的一个解决方案，通过尽量避免拷贝操作，来缓解CPU压力，零拷贝并没有真正做到“0”拷贝，它更多的是一种思想，很多零拷贝技术都是基于这个思想去做的优化

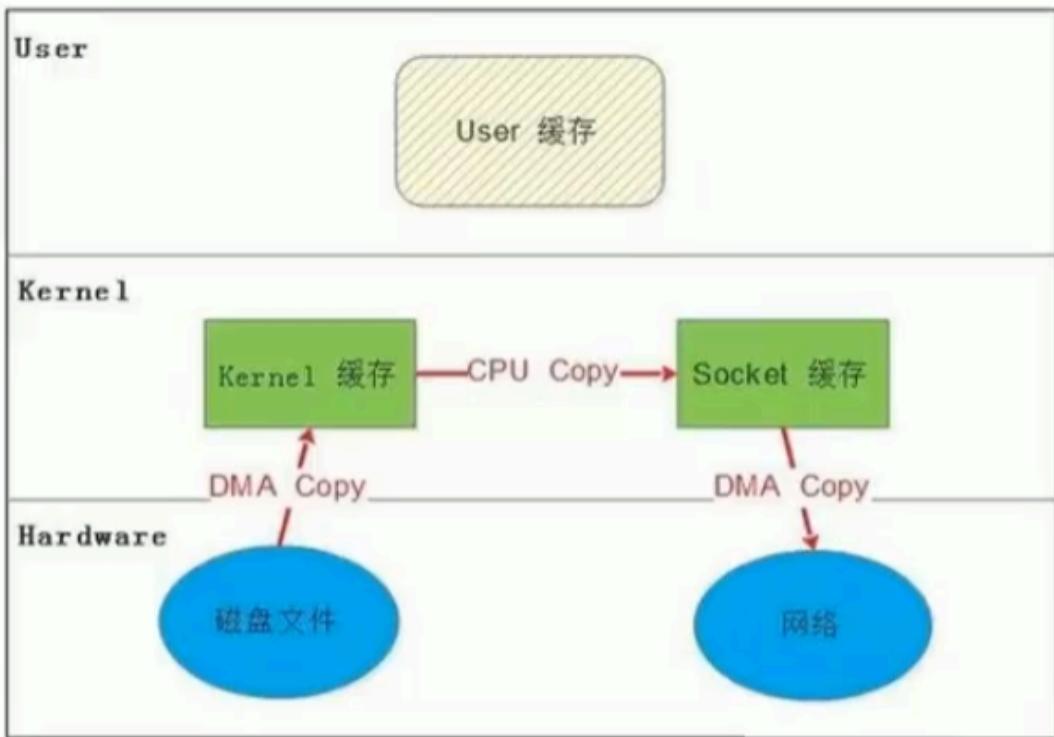
## 零拷贝相关技术

### • MMAP技术

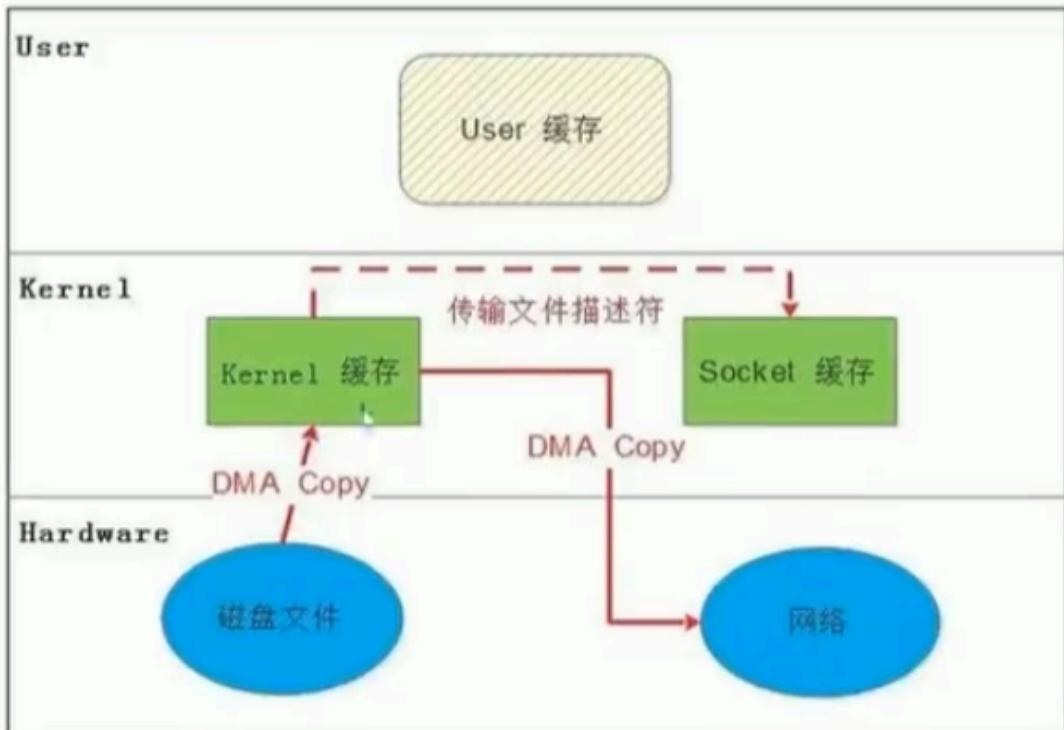


- mmap()系统调用使得进程之间通过映射同一个普通文件实现共享内存。普通文件被映射到进程地址空间后，进程可以向访问普通内存一样对文件进行访问。
- mmap是一种内存映射文件的方法，即将一个文件或者其他对象映射到进程的地址空间，实现文件磁盘地址和进程虚拟地址空间中一段虚拟地址的一一对应关系。
- 实现这样的映射关系后，进程就可以采用指针的方式读写操作这一段内存，而系统会自动会写脏页面到对应的文件磁盘中，即完成了对文件的操作而不必再调用内存映射减少数据在用户空间和内核空间之间的拷贝操作，适合大量数据传输
- Nginx也支持mmap技术，这个也是它性能好的原因之一

- SENDFILE技术 (Nginx也支持该技术)



- DMA辅助的SENDFILE (需要专用硬件支持)



## Nginx介绍

### Nginx功能介绍

- 静态的Web资源服务器html, 图片, js, CSS, txt等静态资源
- http/https协议的反向代理
- 结合FastCGI/uWSGI/SCGI等协议反向代理动态资源请求
- tcp/udp协议的请求转发 (反向代理)

- imap4/pop3协议的反向代理

## 基础特性

- 模块化设计，较好的扩展性
- 高可靠性
- 支持热部署，不停机更新配置文件
- 低内存消耗：10000个keep-alive连接模式下的非活动连接，仅需2.5M内存
- event-driven.aio.mmap, sendfile

## Nginx架构和工作模型

- Nginx中，Master进程和Worker进程分别承担不同的角色，共同协作在以提供高性能的服务
- Master进程(主进程)
  - 启动过程：当你启动Nginx时，Master进程首先启动，它会读取配置文件，初始化全局资源，并创建指定数量的Work进程 (**Master进程创建Work进程**)
  - 创建Work进程（子进程）：Master进程会创建Worker进程来处理实际的客户端请求，每个Worker进程都是一个独立的进程，它们之间相互独立，互不影响
  - 负责处理多个Worker进程间的竞争关系
  - 管理Worker进程：**Master进程负责监控Worker进程的状态。**如果某个Worker进程异常退出（比如发生错误），Master进程会重新启动一个新的Worker进程，以确保服务的可用性。
- Worker进程
  - 处理客户端请求，Work进程是实际处理客户请求的进程。每个Worker进程都是单线程的，但通过多进程的方式，Nginx能够同时处理多个请求，实现高并发
  - 事件驱动和异步处理：Worker进程采用事件驱动的异步模型。它使用事件循环和非阻塞I/O操作，允许在不同的连接之间高效切换，已处理大量并发请求
  - 负载均衡：如果配置了负载均衡，多个Worker进程可以分担负载，均衡地分配请求，提高系统的整体性能

## Nginx启动和HTTP连接建立

- Nginx启动时，Master进程，加载配置文件
- Master进程，初始化监听socket
- Master进程，fork出多个Worker进程
- Worker进程，竞争新的连接，获胜方通过三次握手，连接Socket连接，并处理请求

## Nginx模块介绍

- Nginx有多种模块
  - 核心模块：是Nginx服务器正常运行必不可少的模块，提供错误日志记录，配置文件解析，事件驱动机制，进程管理等核心功能
  - 标准HTTP模块：提供HTTP协议解析相关的功能，比如：端口配置，网页编码设置，HTTP响应头设置等
  - 可选HTTP模块：主要用于扩展标准的HTTP功能，让Nginx能处理一些特殊的服务，比如：Flash多媒体传输，解析GenIP请求，网络传输压缩，安全协议SSL支持等

- 邮件服务模块：主要用于支持Nginx的邮件服务，包括POP3协议，IMAP协议和SMTP协议的支持
- Stream服务模块：实现反向代理功能，包括TCP协议代理
- 第三方模块：是为了扩展Nginx服务器应用，完成开发者自定义功能，比如：Json支持，Lua支持等

#### # 模块分类

核心模块: `core module`

标准模块:

```
HTTP模块: ngx_http_*
    HTTP Core modules      # 默认功能
    HTTP Optional modules # 需编译时指定
Mail模块: ngx_mail_*
Stream 模块: ngx_stream_* (四层代理模块)
```

第三方模块

## Nginx编译安装

```
# 安装依赖包
# Ubuntu
apt update && apt -y install gcc make libpcre3 libpcre3-dev openssl libssl-dev
zlib1g-dev

# 红帽系统
yum -y install gcc pcre-devel openssl-devel zlib-devel
yum -y install gcc make gcc-c++ libtool pcre pcre-devel zlib zlib-devel openssl
openssl-devel perl-ExtUtils-Embed

# 创建用户和组
useradd -s /sbin/nologin nginx

cd /usr/local/src/
wget http://nginx.org/download/nginx-1.18.0.tar.gz

tar xf nginx-1.18.0.tar.gz

cd nginx-1.18.0/

./configure --prefix=/apps/nginx \
--user=nginx \
--group=nginx \
--with-http_ssl_module \
--with-http_v2_module \
--with-http_realip_module \
--with-http_stub_status_module \
--with-http_gzip_static_module \
--with-pcre \
--with-stream \
--with-stream_ssl_module \
--with-stream_realip_module
```

```
make && make install  
  
chown -R nginx.nginx /apps/nginx
```

## 创建Nginx自启动文件

```
# 复制同一版本的nginx的yum安装生成的service文件  
vim /lib/systemd/system/nginx.service  
  
# nginx.service内容  
[Unit]  
Description=nginx -high preformance web server  
Documentation=http://nginx.org/en/docs/  
After=network-online.target remote-fs.target nss-lookup.target  
Wants=network-online.target  
  
[Service]  
Type=forking  
# 指定pid文件的目录， 默认在log目录下， 可选配置  
PIDFile=/apps/nginx/run/nginx.pid  
# Nginx主程序 -c Nginx配置文件路径  
ExecStart=/apps/nginx/sbin/nginx -c /apps/nginx/conf/nginx.conf  
ExecReload=/bin/kill -s HUP $MAINPID  
ExecStop=/bin/kill -s TERM $MAINPID  
LimitNOFILE=100000  
  
[Install]  
WantedBy=multi-user.target  
  
# 创建pid文件存放的目录  
mkdir /apps/nginx/run/  
  
# 修改配置文件  
vim /apps/nginx/conf/nginx.conf  
pid /apps/nginx/run/nginx.pid;
```

## 验证Nginx自启动文件

```
systemctl daemon-reload  
systemctl enable --now nginx
```

## Nginx自动编译脚本

- 老王的视频：【马哥教育】2024年最新Nginx教程入门精讲 ---- P7 Nginx包安装和编译安装 37:00

### 问题1

- 为什么Nginx的master是root用户运行，而work是普通账号运行
  - master进程会对外监听80端口，接收用户请求，然后交给子进程worker处理
  - 80端口是特权端口，只有root账号有权限使用，普通用户无法监听1023以内的端口
  - nc命令，实现指定端口监听

```
# 监听1234
nc -l 1234 # 假设10.0.0.108开启监听

nc 10.0.0.108 # 连接该主机，实现主机间通信

su - mystical
nc -l 1023 # 报错，1023以内，特权端口，只有root可以监听，普通用户不能监听
```

## Nginx命令常见用法

```
# 可以发送信号
nginx -s signal

# 常见信号signal
stop          # 退出，强退
quit          # 优雅退出
reopen
reload        # 重新加载配置文件，会加载新的进程，平滑关闭旧的worker进程

# 以前台方式运行
nginx -g 'daemon off;'
```

## 平滑升级流程

- 平滑升级4个阶段
  - 只有旧版nginx的master和worker进程
  - 旧版和新版nginx的master和worker进程并存，由旧版nginx接收处理用户的新请求
  - 旧版和新版nginx的master和worker进程并存，由新版nginx接收处理用户的新请求
  - 只有新版nginx的master和worker进程
- 不停机更新Nginx二进制文件

```
# 服务端创建一个大文件
dd if=/dev/zero of=/apps/nginx/html/test.img bs=1M count=10

# 客户端下载该文件，保证有进程在工作
wget --limit-rate=1k http://10.0.0.200/test.img

# 编译新版本，得到新版本的二进制文件
wget http://nginx.org/download/nginx-1.24.0.tar.gz
tar xf nginx-1.24.0.tar.gz
cd nginx-1.24.0/

# 参考老版本的编译选项，去编译新版本
# 查看老版本
nginx -V
./configure --prefix=/apps/nginx --user=nginx --group=nginx ...
make # 只执行make，不执行make install
# 在objs/nginx -v

# 把之前的nginx命令备份
cp /apps/nginx/sbin/nginx opt/nginx.old
```

```

# 把新版的nginx复制过去，覆盖旧版的文件，需要-f选项，强制覆盖，否则会提示Text file busy

# 启动新版本
kill -USR2 `cat /apps/nginx/logs/nginx.pid` #-USER2，并来实现USR2

# 此时如果旧版worker进程有用户的旧的请求，会一直等待处理完后才会关闭，即平滑关闭
kill -WINCH `cat /apps/nginx/logs/nginx.pid.oldbin` 

# 如果有新请求，则由新版本提供服务

# 经过一段时间测试，如果新版本没问题，最后发送QUIT信号，退出老的master，完成全部升级过程
kill -QUIT `cat /apps/nginx/logs/nginx.pid.oldbin` 

#####
##### 回滚 #####
#如果升级的新版本发现问题，需要回滚，可以发送HUP信号，重新拉起旧版本的worker
kill -HUP `cat /apps/nginx/logs/nginx.pid.oldbin` 

# 最后关闭新版的master和worker，如果不执行上面的HUP信号，此步QUIT信号也可以重新拉起旧版本的
woker进程
kill -QUIT `cat /apps/nginx/logs/nginx.pid` 
# 恢复旧的文件
mv /opt/nginx.old /apps/nginx/sbin

```

## Nginx核心模块

### 配置文件说明

- Nginx的配置文件的组成部分：
  - 主配置文件：nginx.conf
  - 子配置文件：include conf.d/\*.conf
  - fastcgi/php), uwsgi/python), scgi等协议相关配置文件
  - mime.types：支持的mime类型， MIME多用途互联网邮件扩展类型， MIME消息包含文本， 图像，音频，视频以及其他应用程序专用的数据，是设定某种扩展名的文件用一种应用程序来打开的方式类型，当该扩展名文件被访问时，浏览器会自动使用指定应用程序来打开。多用于指定一些客户自定义的文件名，以及一些媒体文件打开方式
  - nginx.conf.default等\*.default等文件用于如果之前的文件改错了，可以用这个文件尝试恢复，起到一个备份的效果
- 主配置文件结构：四部分

```

# 全局配置段
main block: 主配置段，即全局配置段，对http, mail都有效

# 事件驱动相关的配置
event {
  ...
}

#http/https 协议相关配置段
http {
  ...
}

```

```
# 默认配置文件不包括下面两个块
#mail相关配置段
mail {
    ...
}

# stream 服务器相关配置段
stream {
    ...
}
```

## 主块main

```
# 全局配置段，对全局生效，主要设置nginx的启动用户/组，启动的工作进程数量，工作模式，Nginx的PID
# 路径，日志路径等
user      nginx nginx      # worker进程

# 将user注释后，属组属主使用编译时，指定的属组属主

worker_processes  1;      #启动工作进程数数量，建议设为CPU核数，EPOLL模型，多路复用，一个进程可以处理多个IO请求
# 本质上是该进程调用系统调用中的epoll()同时对多个文件描述符进行操作
# auto选项，自动根据CPU核数，使用进程数（生产中一般使用auto）

# pid文件保存路径
pid      /apps/nginx/logs/nginx.pid;

# worker进程优先级
worker_priority 0  # 相当于CPU的NICE优先级，且范围和Nice相同，都是-20到+19

# 所有worker进程能打开的文件数量上限
# 一个连接对一个文件描述符，文件数量上限即连接上限
# 通常配合lvs反向代理使用，比如该nginx所在服务器最多承载5w的连接，则可以限流5w，多余的浏览由lvs
# 将其分流给其他服务器处理
# 该数据受内核参数`ulimit -n`数量限制，来需要该外面的其他设置
# 如果是nginx直接启动，更改/etc/security/limits.conf
# 更改limits.conf
# 添加 * hard nofile 10000
# * soft nofile 10000
# root hard nofile 10000
# root soft nofile 10000

# 如果是systemctl启动，则更改service文件中的参数
# LimitNOFILE=100000
worker_rlimit_nofile 65536

# 查看nginx的连接数量限制（硬限制）
pidof nginx      # 查找nginx的PID

cat /proc/`pid`/limits  # 即可显示具体限制数据
```

- 重新加载nginx配置文件

```
nginx -s reload
```

- 通过亲源机制，把CPU和进程进行绑定（面试题：nginx优化手段之一）

```
# 将Nginx工作进程绑定到指定的CPU核心，默认Nginx是不进行绑定的，绑定并不是意味着当前Nginx进程独占一核CPU，但是可以保证
# 此进程不会运行在其他核心，这就减少了nginx的工作进程在不同cpu核心上的来回跳转，可以有效提升
nginx服务器的性能

# 不同CPU之间跳转带来的问题：CPU中缓存数据的失效

worker_cpu_affinity 00000001 00000010 00000100 00001000 | auto ;
#CPU MASK: 00000001 0号CPU
#           00000010 1号CPU

# 示例
worker_processes 4;
worker_cpu_affinity 00000010 00001000 00100000 1000000
ps axo pid,cmd,psr (显示CPU编号) |grep nginx

# 特殊示例
worker_cpu_affinity 0101 1010 一个进程可以同时绑定多个CPU # 实例中表示，两个进程，可绑定
2个CPU上

# auto 自动，随机结亲缘
worker_processes auto;
worker_cpu_affinity 不支持 auto，需要手动配置绑定掩码。

# 通过脚本生成 worker_cpu_affinity 配置，实现细粒度的 CPU 绑定。
CORES=$(nproc) # 获取 CPU 核心数
for ((i=0; i<$CORES; i++)); do
    MASK=$((printf "%0${CORES}d" 0 | sed "s/0/1/${i}") )
    echo -n "$MASK "
done
```

- 压力测试

```
# ab命令
# ab命令参数
# -n: 总请求数； -c: 模拟的并发数 -k: 以持久连接模式测试
# Ubuntu在apache2-utils
ab -c 1000 -n 10000 http://10.0.0.200/
# 表示一次发1000个请求，总共发10000次

# 在10.0.0.200使用watch命令观察
watch -n0.5 'ps axo pid,cmd,psr | grep nginx'
```

## 面试重点：nginx优化手段

- CPU结亲缘
- 指定worker进程数，使其一个CPU核心上执行一个进程
- 优先级 (worker\_priority)
  - 优先级数值越小，优先级越高，CPU越优先调度资源给该进程
- 增加work\_connection的数值，默认512，这个是nginx单个work进程能够处理的连接数

- 开启防惊群设置 accept\_mutex on 默认是off,会产生惊群问题

## 面试重点：top命令

- top命令的作用：动态跟踪执行命令

```
# 21:28:51(当前时间) up 105 days, 9:42(设备已经运行的时间), 4个用户, 负载均衡在1分钟, 5分钟, 15分钟的时间内, CPU进程数
top - 21:28:51 up 105 days, 9:42, 4 users, load average: 0.00, 0.00, 0.00

# 总共165个进程, 9个在运行, 156个睡眠中, 0个停止, 0个僵尸
Tasks 165 total, 9 running, 156 sleeping, 0 stopped, 0 zombie

# 用户空间cpu使用率32.7, 内核空间cpu使用率66.0, 优先级0.2, id(cpu空闲时间百分比), wa(cpu等待io完成的时间百分比), hi(硬中断时间), si(软中断时间), st(偷取时间, 即虚拟机占用CPU时间片)
%CPU(s): 32.7 us, 66.0 sy, 0.2 ni, 0.0 id, 0.0 wa, 1.0 hi, 0.2 si, 0.0 st

# 内存信息
MiB Mem: 797.1 total, 238.9 free, 215.7 used, 342.5 buff/cache

# Swap信息
MiB Swap: 2048.0 total, 2048.0 free, 0.0 used, 431.6 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

# PR: 优先级 (PR优先级0, 相当于NICE优先级20)
# NI: Nice优先级 (Nice优先级范围: -20到+19)
# VIRT: 虚拟内存: 承诺你用的内存
# RES: 实际分配的内存
# SHR: 共享的内存
# S: 运行状态
# %CPU: CPU利用率
# %MEM: 内存利用率
# TIME+: 累计CPU时间片长度
```

- top: 默认按照CPU利用率排序进行显示

```
# 快捷键
M      # 按内存利用率排序
P      # 按CPU利用率排序
T      # 按积累占据CPU时长TIME+进行排序
```

- 首部信息展示

```
uptime信息: l命令
task命令及CPU信息: t命令 (花里胡哨, 用处不大)
CPU分别显示: 1(数字)
memory信息: m命令
退出命令: q
修改刷新时间间隔: s
终止指定进程: k
保存文件: w
```

- top选项

```
-d      # 指定刷新时间间隔, 默认3秒
-b      # 全部显示所有进程
-n      # 刷新多少次后退出
-H      # 线程模式
```

## event模块

```
worker_connection 65535 # 设置单个工作进程的最大并发连接数,默认512,建议加大,否则,只能接受最大指定连接数,超出则会丢弃请求,不会处理。
user epoll          # 使用epoll事件驱动,默认epoll
accept mutex on     # 惊群,推荐on, on为同一时刻一个请求轮流由worker进程处理,而防治被同时唤醒所有worker
# 默认为off,新请求会唤醒所有的worker进程
multi_accept on    # on时Nginx服务器的每个工作进程可以同时接受多个新的网络连接,此指令默认为off,即默认为一个工作进程只能一次接受一个新的网络连接,打开后可以同时接受多个,建议设置on,充分利用多路复用的理念
```

## http模块

- http协议相关的配置结构

```
http {
  ...
  ... # server的公共配置
  server { # 每个server用于定义一个虚拟主机,第一个server为默认虚拟服务器
    ...
  }
  server {
    ...
    server_name # 虚拟主机名
    root        # 主目录
    alias       # 路径别名
    location [OPERATOR] URL { # 指定URL的特性
      ...
      if CONDITION {
        ...
      }
    }
  }
}
```

- http协议配置说明

```
http {
  include      mime.types; # 导入支持的文件类型,是相对于/apps/nginx/conf的目录
  default_type application/octet-stream; # 除mime.types中文件类型外,设置其他文件默认类型,访问其他类型时会提示下载不匹配的类型文件

  #log_format main '$remote_addr - $remote_user [$time_local] "$request" '
  #                                '$status $body_bytes_sent "$http_referer" '
  #                                '"$http_user_agent" "$http_x_forwarded_for"';
```

```
#access_log  logs/access.log  main;

sendfile      on;  # 一种零拷贝的技术
#tcp_nopush   on; # 开启sendfile的情况下，合并请求后统一发给客户端，必须开启
sendfile
#tcp_nodelay  off; # 开启keepalive模式下的连接是否启用TCP_NODELAY选项，为off时，  
延迟0.2秒，默认on时，不延迟发送，立即发送用户响应报文。
# tcp_nopush 和 tcp_nodelay 两者互斥，不能同时生效

#keepalive_timeout  0;
keepalive_timeout  65;

#gzip  on;

server {
    listen      80;
    server_name localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;

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

    #error_page  404          /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root  html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass    http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root          html;
    #    fastcgi_pass  127.0.0.1:9000;
    #    fastcgi_index index.php;
    #    fastcgi_param SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
}
```

```

#location ~ /\.ht {
#    deny all;
#}

# another virtual host using mix of IP-, name-, and port-based configuration
#
#server {
#    listen      8000;
#    listen      somename:8080;
#    server_name somename alias another.alias;

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

# HTTPS server
#
#server {
#    listen      443 ssl;
#    server_name localhost;

#    ssl_certificate      cert.pem;
#    ssl_certificate_key  cert.key;

#    ssl_session_cache    shared:SSL:1m;
#    ssl_session_timeout  5m;

#    ssl_ciphers  HIGH:!aNULL:!MD5;
#    ssl_prefer_server_ciphers  on;

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

}

```

## 指定响应报文server首部

```

server_tokens on | off | build | string;

# 默认值是on
# 如果要隐藏Nginx版本，使用off
# 写在http模块内，server模块外

```

- 修改Server头部信息

```
vim /usr/local/src/nginx-1.18.0/src/core/nginx.h
#define NGINX_VERSION      "1.68.9"
#define "fenginx" NGINX_VERSION

vim nginx-1.18.0/src/http/ngx_http_header_filter_module.c
static u_char ngx_http_server_string[] = "Server: feng-nginx" CRLF;
```

## 指定字符集

- 指定字符集utf-8,防止网页显示乱码

```
# 是否在响应报文中的Content-Type显示指定的字符集, 默认off不显示
charset charset | off;
charset utf-8;
```

## 常见核心模块配置

### 新建PC Web站点

```
# 定义自配置文件路径
mkdir /apps/nginx/conf/conf.d

vim /apps/nginx/conf/nginx.conf
http {
    .....
    include /apps/nginx/conf/conf.d/*.conf
    # 在配置文件的最后添加此行, 注意不要放在最前面, 会导致前面的命令无法生效
}

# 创建PC网站配置
cat /apps/nginx/conf.d/pc.conf
server {
    listen 80;
    server_name www.wang.org;
    location / {
        root /data/nginx/html/pc
    }
}

# 实际配置两行就行
server {
    server_name www.feng.org;
    root /data/nginx/html/pc
}

mkdir -p /data/nginx/html/pc
echo "pc web" > /data/nginx/html/pc/index.html
```

## 扩展：curl用法详解

```
curl [url | IP]

# 常用参数
-I      # 只显示响应头(head)信息
-v      # 显示详细内容
-H      # Extra header to include in the request when sending HTTP to a server.
You may specify any number of extra headers. Note that if you should add a custom
header that has the same name as one of the internal ones curl would use, you
externally set header will be used instead of the internal one.
# 示例:
curl -H "User-Agent: yes-please/2000" https://example.com
curl -H "Host:" https://example .com
```

## root和alias

- root和alias的区别
  - root: 自定地址+.../
  - alias: 直接访问自定地址/...

```
server {
    server_name www.feng.org;
    # 指定网站家目录
    location / {
        root /data/nginx/html/pc;
    }
    location /about {
        # 访问www.feng.org/about
        # 实际访问路径是/opt/pc/index.html
        alias /opt/pc;
        # 访问www.feng.org/about
        # 实际访问路径是/opt/pc/about/index.html
        #root /opt/pc;
    }
}
```

## location指令

- 针对特定的uri (url)

```

# 语法规则
location [ = | ~ | ~* | ^~ ] uri {...}

= # 用于标准uri前，表示请求字符串和uri精确匹配，大小敏感
^~ # 用于标准uri前，表示包含正则表达式，并且匹配以指定正则开头，对uri的最左边部分做匹配检查，不区分大小写
~ # 用于标准uri前，表示包含正则，区分大小写
~* # 用于标准uri前，表示包含正则，不区分大小写
不带符号 # 匹配起始于此uri的所有uri
\ # 用于标准uri前，表示包含正则表达式并且转义字符，可以将.*?等转译为普通符号
# 匹配优先级：
=, ^~, ~/~*, 不带符号

```

- Location @name 这样的location不用于常规请求处理，而是用于请求重定向

```

# cat /etc/nginx/conf.d/www.wang.org.conf

server {
    listen 80;
    server_name www.wang.org;
    root /data/www;
    location / {
        index index.html;
    }
    # 如果出现异常，则重新定向到@error_404这个location上
    error_page 404 @error_404;
    location @error_404 {
        default_type text/html;
        charset utf8;
        return 200 'your page is not found!';
    }
}

```

## 核心模块（面试常问：nginx有哪些模块）

### 访问控制模块（Module ngx\_http\_access\_module）

- 常用命令
  - allow
  - deny
- 作用域（Context）

```
http, server, location, limit_except
```

- 语法格式

Syntax: allow address| CIDR | unix; |all

Syntax: deny address| CIDR | unix; |all

# return 会覆盖deny的作用，deny无法禁止return的返回值  
# 同时多个allow和deny一起使用时，按上面先生效

## 账户认证功能

- 由ngx\_http\_auth\_basic\_module模块提供此功能

```
#CentOS安装包  
yum -y install httpd-tools  
#Ubuntu安装包  
apt -y install apache2-utils
```

- Directives

```
auth_basic      "string" | off ;  
auth_basic_user_file  conf/htpasswd;  
  
# 示例:  
location / {  
    #显示在响应头里  
    auth_basic      "string" | off;  
    auth_basic_user_file  conf/htpasswd;  
}  
  
# 创建htpasswd  
# 第一次使用-c, 创建密码文件  
htpasswd -bc ./httpuser tom 123456  
# 第二次不加-c, 否则新创建文件覆盖原文件  
htpasswd -b ./httpuser jerry 123456  
  
# 使用curl命令访问  
curl -u user2:123456 www.xxx.org/admin
```

## 自定义错误页面

- 自定义错误页面，同时也可以用指定的响应状态码进行响应，可用位置：http,server,location,if in location
- 语言格式

```
Syntax: error_page code ... [=response]] uri;  
Default: -  
Context: http, server, location, if in location
```

- 示例

```
error_page 500 502 503 504 /error.html;  
location = /error.html {  
    root /data/nginx/html;  
}  
# 重启Nginx访问不存在页面测试
```

## 自定义错误日志

- 可以自定义错误日志

```
Syntax: error_log file [level]
Default:
error_log logs/error.log error;
Context: main, http, mail, stream, server, location
Level: debug, info, notice, warn, error, crit, alert, emerg

# 关闭错误日志
error_log /dev/null;
access_log /apps/nginx/logs/wang.org_access.log;

# 定义错误日志
error_log /apps/nginx/logs/wang.org_error.log warn
location = /error.html {
    root /data/nginx/html;
}
```

## 日志切割

```
# 将原log日志备份
mv access.log bak.log

# 执行reopen
nginx -s reopen
# 下列命令等同于nginx -s reopen
kill -USR1 $(cat /usr/local/openresty/nginx/logs/nginx.pid)

# 可以把这两个命令写出bash脚本，放在crontab中，每周执行一次
```

## 检测文件是否存在

- try\_files会按顺序检查文件是否存在，返回第一个找到的文件或文件夹（结尾加斜线表示为文件夹），如果所有文件或文件夹都找不到，会进行一个内部重置定向到最后一个参数。只有最后一个参数可以引起一个内部重定向，之前的参数只设置内部URI的指向。最后一个参数是回退URI且必须存在，否则会出现内部500错误。

```
Syntax: try_files file...uri;
try_files file... = code;
Default: -
Context: server, location
```

- 范例

```
# Ubuntu包安装默认配置
location / {
    try_files $uri $uri / =404;
}
```

- 范例：如果不存在页面，就转到default.html页面

```
location / {  
    try_files $uri $uri.html $uri/index.html /about/default.html;  
}
```

- 完整示例

```
location / {  
    root /data/nginx/html/pc;  
    index index.html;  
    try_files $uri $uri.html $uri/index.html /about/default.html;  
}
```

## 检测文件是否存在 (补充)

- 语法格式

```
# 语法格式1  
Syntax: try_files file... uri;  
# 语法格式2  
Syntax: try_files file... =code;  
# 如果自定义的状态码，则会显示在返回数据的状态码中
```

## 长连接配置

```
Syntax: keepalive_requests number;  
# 示例  
keepalive_requests 1000;  
  
Syntax: keepalive_time time;  
#示例  
keepalive_time 1h;  
  
Syntax: keepalive_timeout timeout [header_timeout];  
# 这里head_timeout在响应报文里会有，纯诈骗  
# 示例  
keepalive_timeout 75s;  
  
keepalive_time time; 限制对一个请求处理的最长时间，到时间后续的再有新的请求会断开连接，默认1h;  
keepalive_timeout timeout [header_timeout]: 设置保持空闲的连接超时时长，0表示禁止连接，  
默认为75s，通常配置在http字段作为站点全局配置  
keepalive_requests number: 在一次长连接上所允许请求的资源的最大数量， 默认为1000次
```

## 作为下载服务器配置

- ngx\_http\_autoindex\_module模块处理以斜杠字符“/”结尾的请求，并生成目录列表，可以做为下载服务器配置使用
- 类似镜像源的网站

```

Syntax: autoindex on | off;

# 其他参数
autoindex on|off      # 自动文件索引功能, 默认为off
autoindex_exact_size on | off # 计算文件确切大小(单位byte), off是显示大概大小(单位K, M), 默认on
autoindex_localtime on | off # 显示本机时间而非GMT时间, 默认off
charset charset|off # 指定字符编码, 默认为off, 中文会乱码, 指定为utf8
autoindex|format html | xml | json | jsonp # 显示索引的页面文件风格, 默认html
limit_rate rate;      # 限制响应客户端传输速率(除GET和HEAD以外的所有方法), 单位B/s, 即Bytes/second, 默认值0, 表示无限制, 此指令由ngx_http_core_module提供
set $limit_rate 4k; # 也可以通过变量限速, 单位B/s, 同时设置, 此项优先级高

```

## 做为上传服务器

- 以下指令控制上传数据

```

client_max_body_size 1m; # 设置允许客户端上传单个文件的最大值, 默认值为1m, 上传文件超过此值会出413错误
client_body_buffer_size size; # 用于接收每个客户请求报文的body部分的缓冲区大小; 默认16k; 超出此大小时, 其将暂存到磁盘上的由client_body_temp_path指令定义的位置
client_body_temp_path path [level1 level2...]; # 设定存储客户端请求报文的body部分的临时存储路径及子目录结构和数量, 目录名为16进制的数字, 使用hash之后的值从后往前截取1位, 2位, 2位做为目录名

1级目录占1位16进制, 即 $2^4=16$ 个目录 0-f
2级目录占2位16进制, 即 $2^8=256$ 个目录 00-ff
3级目录占2位16进制, 即 $2^8=256$ 个目录 00-ff

```

## Nginx限流限速

- 目前 Nginx 中主要的三种限速操作分别是:
  - 限制请求数 (request) ,
  - 限制连接数 (connection) ,
  - 限制响应速度 (rate) ,
- 对应在 Nginx 中的模块指令分别是
  - limit\_req,
  - limit\_conn,
  - limit\_rate
- 限制单一连接下载速度

```

limit_rate rate; #对单个客户端连接限速, 默认单位为字节, 其它单位需要显式指定, 表示每秒的下载速度

limit_rate_after size; #在传输了多少数据之后开始限速, 默认值0, 表示一开始就限速

```

- 限制客户端请求数

```

# 定义一个限速规则, 供limit_req指令调用
limit_req_zone key zone=name:size rate=rate [sync];

```

```
key      # 定义用于限速的关键词，表示以什么为依据限速

key的常用值
$binary_remote_addr
$remote_addr  # 以文本形式存储IP地址，推荐$binary_remote_addr，因为它更节省内存
$request_uri #这个键基于完整的请求 URI 来限制请求。这允许对特定页面或路由的请求进行更精细的控制。
# 适用于保护特定高流量或敏感的页面，例如登录页面或搜索功能

$uri # 与$request_uri类似，但是值不包含查询字符，因此针对不带参数的 URL 进行限制非常有用。
# 适用于对相同页面的请求进行限制，无论查询参数如何变化。

组合键: $binary_remote_addr$request_uri
# 可以针对同一 IP 地址对不同的 URI 进行细分限制。

zone=name:size

name    # 表示规则名称
size    # 表示为限制区域分配的内存大小

# 限制区域内存的作用：
# 这个内存区域用于存储所有被监控的 IP 地址（或其他定义的 key）的状态信息，包括：
    ## 当前每个 key 的请求计数。
    ## 请求的时间戳等信息。
# 这些信息用于确定是否允许当前请求继续处理还是应该延迟或拒绝以维持设定的请求速率

# 限制区域内存存在的意义：
# 由于 Nginx 需要跟踪可能的数千甚至数百万个并发连接的状态，因此需要在内存中快速查找和更新每个连接的状态。将这些信息存储在共享内存中使得多个 worker 进程可以高效地访问这些信息，而不需要频繁的磁盘 I/O 操作，这对性能至关重要。

# 缓存在共享内存空间中的数据的生命周期：
## 请求处理完毕：当一个请求被处理完毕后，与该请求相关的特定数据（如请求体、响应等）会被释放，但用于 limit_req 的状态信息（即 IP 地址的请求频率数据）通常会保留在内存中。
## 状态信息更新：每当相同的 IP 地址再次发起请求时，其对应的状态信息会被更新。如果该 IP 在一段时间内没有新的请求，其状态信息可能仍然保留在内存中，直到内存区域需要为新的活跃 IP 释放空间。

# 实际应用
# 对于高流量的网站，应合理配置共享内存大小以确保足够的空间存储所需的状态信息。如果设置得过小，可能会导致频繁的内存清理，影响性能；设置过大，则可能浪费资源。

rate=rate

# 举例：rate=1r/s
# 每个 IP 地址被允许平均每秒产生一个请求

# 调用定义好的限制规则
limit_req zone=name [burst=number] [nodelay | delay=number]

zone=name # 表示调用哪个limit_req_zone
[burst=number] # 表示超过rate之后，还能接收多少请求，这些请求会被放到队列慢慢处理
[nodelay | delay=number]
# nodelay表示当启用 nodelay 时，只要 burst 队列中有空间，超出 rate 速率的请求将立即被处理，而不进行人为延迟。这意味着如果服务器有处理能力（即系统资源允许），这些请求可以立即得到响应，而不用等待。
```

```
# 也就是说开启了nodeLAY，服务资源充足的情况下，数据不会排在队列等待慢慢被处理，队列的数值仅仅是请求是否被拒绝的一个指标

# delay=number表示有多少个请求直接处理，剩下的加入队列慢慢处理

limit_req_status code; # 因为limit_req配置被拒绝处理的请求的返回状态码，默认503

limit_req_log_level info|notice|warn|error
# 定义因limit_req配置被拒绝处理的请求的日志级别，默认error

limit_req_dry_run on|off
# 这个指令允许你观察限流规则如何影响请求处理，而不会实际地阻止这些请求。这对于调试和优化限流配置非常有用，特别是在生产环境中，你可能不想直接启用新规则而导致潜在的服务中断。

# on: 启用干运行模式。在这种模式下，即使请求超出了设定的速率限制，也不会真正地拒绝这些请求。相反，Nginx会记录哪些请求本应被限制，但实际上允许它们正常处理。这允许你观察并调整你的速率限制设置，而不影响用户体验。
```

- 限制客户端并发连接数

```
# 定义一个限速连接，供limit_conn指令调用
limit_conn_zone key zone=name:size;
```

## Nginx中的变量

- Nginx变量可以在配置文件中使用，用作判断或定义日志格式的源等场景，Nginxbian
- Nginx内置变量
  - Nginx内置噶数据是Nginx自行ID定义的
- 常用内置变量
- 用户自定义变量
- 在Nginx中，除了内置变量外，我们还可以使用set指令来自定义变量

```
# 格式:
set $variable value;
```

## Nginx状态页

```
stub_status; # 添加此指令后可开启Nginx状态页，作用域server, location

# 示例
location /status {
    stub_status;
}

# 返回内容
Active connections: 1
server accepts handled requests
 3 3 3
Reading: 0 Writing: 1 Waiting: 0

# Active connection: 当前处于活动状态的客户端连接数，=reading + writing + waiting
```

```
# accepts 总数, 自Nginx启动到当前已接收的客户端请求总数  
# handled 总数, 自Nginx启动到当前已处理完成的连接数, 通常等于accepts, 如果有失败链接, 则要去掉失败连接  
# requests 总数, 自Nginx启动到当前客户端已发送的请求总数  
  
# Reading状态, 当前正在读在请求报文的连接数, 值越大, 说明排队现象越严重, 性能不足  
# Writing状态 正在向客户发送响应报文的过程中的连接数, 值越大, 说明访问量越大  
# waiting 已建立连接, 但没有数据传输的空闲连接数, 需开启keepalive
```

## Nginx第三方模块使用

- 示例

```
https://github.com/vozlt/nginx-module-vts # 第三方流量监控软件  
https://github.com/openresty/echo-nginx-module #echo模块, 可以直接输出
```

- 使用过程

```
# 安装编译工具链  
apt update && apt install -y make gcc libpcre3 libpcre3-dev openssl libssl-dev  
zlibg-dev  
  
# 创建运行用户  
useradd -r -s /usr/sbin/nologin.nginx  
  
# 下载最新版源码并解压, 并上传到Linux服务器  
wget https://nginx.org/download/nginx-1.22.1.tar.gz
```

## 自定义变量

- 常用内置变量

## 自定义访问日志

定义日志格式常用的变量

\$remote_addr	#客户端的IP地址
\$remote_user	#使用 HTTP 基本身份验证时的远程用户
\$time_local	#服务器本地时间
\$request	#客户端请求的 HTTP 方法、URI 和协议
\$status	#服务器响应的状态码
\$body_bytes_sent	#发送给客户端的字节数, 不包括响应头的大小
\$http_referer	#客户端跳转前的来源页面
\$http_user_agent	#客户端的用户代理字符串
\$http_x_forwarded_for	#请求的主机头字段
\$host	#服务器名称
\$server_name	#通过代理服务器传递的客户端真实 IP 地址
\$request_time	#请求处理时间
\$upstream_status	#上游服务器的响应状态码
\$time_iso8601	#ISO 8601 格式的本地时间
\$upstream_response_time	#从上游服务器接收响应的时间
\$request_id	#用于唯一标识请求的 ID

```
$ssl_protocol          #使用的 SSL 协议  
$ssl_cipher            #使用的 SSL 加密算法  
  
# 自定义访问日志格式  
log_format basic '$string';  
  
access_log path [format [buffer=size] [gzip[=level]] [flush=time]  
[if=condition]];
```

## Nginx压缩功能

- 语法格式

```
gzip on | off  #启用或禁用压缩功能， 默认off  
gzip_buffers number size; # nginx在压缩时要想服务器申请的缓存空间个数和大小
```

## favicon 图标配置

```
# 给出文件favicon  
location = /favicon.ico {  
    root /var/www/html/www.m99-magedu.com/static;  
    expires 7d; # 首次请求缓存后，7天内不能发请求，：  
}
```

## Nginx实现Https

- Nginx中的Https功能需要ngx\_http\_ssl\_module模块支持，使用Yum/apt安装的Nginx中已经包含了该模块功能。如果是自行编译的Nignx，需要在编译的时候指定相关的编译项

```
# 常用选项  
ssl on|off  
  
ssl_certificate file; # 当前虚拟主机的证书文件，通常是PEM格式（两个证书合二为一）前面是虚拟  
# 主机证书，后面是CA机构证书  
ssl_certificate_key file; # 当前虚拟主机的私钥文件路径  
  
ssl_session_cache off|none [builtin[:size]][shared:name:size]  
# 配置SSL缓存，作用域http, server  
# off 禁用SSL/TLS会话缓存  
# none 通知客户端可以重用会话，但并没有缓存相关数据，默认值  
# builtin[:size] 使用openssl内建缓存，可指定大小，每个worker进程独享  
# 【share:name:size】 使用共享缓存，每个worker共享该缓存中的数据可以指定name和大小  
  
ssl_session_timeout time; # 配置SSL/TLS会话缓存的超时时间，默认值为5m  
# 使用SSL/TLS会话缓存有助于减少服务器的加密和解密负担，提高HTTPS连接的响应速度，启用缓存需要  
ssl_session_cache,ssl_session_timeout 两个选项一起使用
```

#SSL/TLS会话缓存存储的是SSL/TLS握手过程中生成的会话数据。在SSL/TLS握手过程中，服务器和客户端会交换一系列数据，其中包括协商的密钥、加密算法、会话标识符等信息。这些信息用于确保安全通信，并在建立连接后用于加密和解密通信的数据  
#SSL/TLS会话缓存中存储的主要数据包括 会话标识符(Session Identifier),主密钥(Master Secret),加密算法和参数

#通过存储这些会话数据，SSL/TLS会话缓存允许服务器在处理新连接时，如果客户端提供了先前使用过的会话标识符，就可以重用这些数据，避免重新执行完整的SSL/TLS握手。这样可以大幅度减少握手过程中的计算和通信开销，提高性能和响应速度。同时，它还有助于减少服务器的负担，因为不需要重新生成新的密钥和协商参数

## Nginx中配置防盗链

- 语法格式

```
valid_referers none | blocked | server_names | string... ;  
  
# none: 请求头中没有referers则 $invalid_referer,值为空  
# static: 请求头中,有referer字段,但其值不合法(不是http或https协议),则值为空  
# server_name: 具体主机名,支持正则表达式,匹配成功,则值为空
```

- 示例

```
server {  
    listen 80;  
    server_name www.abc-234.com;  
    root /var/www/html/www.abc-123.com;  
    valid_referers none blocked server_names *.test.com ~\..baidu\. ~\..bing\.  
~\..so\.;  
    if ($invalid_referer) {  
        return 403 "Forbidden Access"  
    }  
}
```

## Nginx中的Rewrite

- 在Nginx中，rewrite指令用于重写URI，允许Nginx修改客户端请求的URI，基于此，可用该指令实现URI重定向，修改请求参数，改变请求含义，改变URL结构等
- 该指令来自于ngx\_http\_rewrite\_module 模块
- 主要作用：允许管理员通过配置文件来修改客户端的请求URI，从而实现重新URI，重定向请求，更改请求参数等操作
- 相关指令说明

```
break; # 跳出server, 后续的nginx_http_rewrite_module模块的其他命令都不生效, 只中断这个模块,  
其他模块正常执行  
last; # 跳出当前location, 继续从头开始匹配  
  
if (condition) {  
    ...  
}  
# 仅能做单次判断, 不支持if-else多分支  
# if ($var) {} 仅0和空字符串为false  
# 支持运算符  
# =      比较变量和字符串是否相等, 非赋值  
# !=     比较变量和字符串是否不等  
# ~      区分大小写, 是否正则匹配  
# ~*    不区分大小写  
# !~*   不区分大小写, 是否不匹配
```

```

# -f|-!-f 判断文件是否存在 | 不存在
# -d|-!-d 判断目录是否存在 | 不存在
# -x|-!-x 判断是否可执行 | 不可执行
# -e|-!-e 判断文件（包括文件，目录，软连接）是否存在 | 不存在

return code [text]
return code url;
return url    # 不写code, 默认302
              # return后的指令不再执行

rewrite regex replacement [flag];
# flag: last|break|redirect(客户端)|permanent(同redirect)
# redirect (302) 和 permanent (301) 几乎一样, 一个301, 一个302

rewrite_log on|off   # 是否记录nginx_http_rewrite_module模块产生的日志到error_log中, 默认off

set $variable value;  设置变量, 给变量赋值

```

- 示例

```

location /scheme {
    if ($scheme = http) {
        return 200 "http"
    }
    if ($scheme = https) {
        return 200 "https"
    }
}

location /file {
    if (!-e $request_filename) {
        return 200 "$request_file file not exists";
    }
}

```

- 利用set实现指定限速

```

set $slow 1;
location = /test.img {
    if ($slow) {
        limit_rate 10k;
    }
}

```

- return实现浏览器判断

```

location /return {
    if ($http_user_agent ~* curl|wget|ApacheBench) {
        return 403 "agent error";
    }
    return 200 "success";
}

```

- rewrite 指令测试

```
server {
    listen 80;
    server_name www.feng.org;
    root /var/www/html;

    location /{
        rewrite /1.html /2.html;
        rewrite /2.html /3.html;
    }

    location /2.html {
        rewrite /2.html /a.html;
    }

    location /3.html {
        rewrite /3.html /b.html;
    }
}
```

## Nginx反向代理

### 实现http协议反向代理

- 相关指令和参数

```
proxy_pass URL;      # 转发的后端服务器地址，可以写主机名，域名，IP地址。也可以额外指定端口

proxy_hide_header field;  # 显示指定不回传的响应头字段

proxy_pass_header field;  # 显示指定要回传给客户端的响应头字段

proxy_pass_request_body on|off  # 是否向后端服务器发送客户端http请求的body，默认on

proxy_connect_timeout time;      # Nginx与后盾服务器建立连接超时时长，默认60s，超时会向客户返回504

proxy_read_timeout time;          # Nginx等待后端服务器返回数据的超时时长，默认60sd，超时返回504

proxy_send_timeout time;          # Nginx向后端服务器返回数据的超时时长

proxy_set_body value;            # 重新定义传给后端服务器的请求正文，可以包含文本，变量等

proxy_set_header field value;    # 更改或添加请求头字段并发送到后端服务器

proxy_http_version 1.0|1.1;      # 设置向后端服务器发送请求时，http协议版本，默认1.0

proxy_ignore_client_abort on|off # 客户端中断连接，Nginx是否继续执行与后端的连接，默认off，客户端中断，Nginx也中断与服务端的连接，on表示客户端中断，Nginx还会继续处理与服务端连接

proxy_headers_hash_buket_size size; # 当配置了proxy_hide_header和proxy_set_header的时候，用于设置nginx保存HTTP报文头hash表大小，默认64
```

```
proxy_headers_hash_max_size size; # 上一个参数的上限, 默认值512
```

## 基本配置

- 转发到指定IP

```
# 转发到指定ip, http默认80端口
server {
    ...
    location / {
        proxy_pass http://10.0.0.180;
    }
}
```

## 小实验：搭建一个具备缓存功能的反向代理

```
# 将上游提供服务的web服务器, 设置为仅本机进程访问8080端口
# 有上游服务器
server {
    listen 127.0.0.1:8080;
    server_name www.web.org;
}
```

## 篡改代理转发给后端服务器的头部信息

```
# 可更改或添加客户端的请求头部信息内容并转发至后端服务器, 比如在后端服务器想要获取客户端的真实IP
的时候, 就要更改每一个报文的头部
proxy_set_header

# 示例
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header [首部字段] [值]

# 示例
server {
    listen 80;
    server_name www.wang.org;
    location / {
        proxy_pass http://10.0.0.18/;
        proxy_set_header Host $http_host; # 转发主机头至后端服务器
        proxy_connect_timeout 10s;
    }
}
```

## 添加响应报文的头部信息

- nginx基于模块ngx\_http\_headers\_module可以实现对后端服务器响应给客户端的报文中添加指定的响应首部字段

```

Syntax add_header name value [always];

# 添加响应报文自定义首部
add_header name value [always];

# 示例
add_header X-Via $server_addr;    # 当前nginx主机的IP
add_header X-Cache $upstream_cache_status   # 缓存命中HIT, 未命中MISS
add_header X-Accel $server_name; 客户访问的FQDN

```

## 动静分离实验

- 实验环境准备
  - Client ----- 10.0.0.151
  - Proxy ----- 10.0.0.180
  - API Server ----- 10.0.0.181
  - Static ----- 10.0.0.150
- 错误情况

```

# 在代理服务器10.0.0.180配置
location /static {
    proxy_pass http://10.0.0.150;
    proxy_set_header Host "static.feng.org"
}

# 后端服务器10.0.0.150配置
server {
    server_name static.feng.org;
    root /data/html;
}

# 错误现象：客户端会301重定向，并向static.feng.org/static/发起访问
# 由于客户端没有在hosts添加static.feng.org域名的解析，因此会报错

```

- 原因：

数据包的传输分为以下几个阶段

阶段1：客户端将发送请求给代理服务器180

阶段2：代理服务器180根据配置将数据转发给150，并将数据头部信息的HOST改为static.feng.org

阶段3：后端服务器收到代理转发的包，根据上面的地址进行拼接：static.feng.org/static，此时出现一个问题：static被判断为目录，因此服务器在后面自动补齐斜线，并发送一个301的响应报文，要求客户端重定向到补齐斜线的新路径：static.feng.org/static/

最终：客户端浏览器收到这个响应报文，根据要求重定向，向static.feng.org/static/发送请求，但是由于没有static.feng.org的域名解析，因此报错

- 解决方法：
  - 方法1：在180代理服务器上，将/补齐

```
# 更改为
location /static/ {
    proxy_pass http://10.0.0.150;
    proxy_set_header Host "static.feng.org"
}
```

- 方法2：在150的配置文件中添加try\_files，使其自动查找路径下文件，而不是告诉浏览器重新发请求

```
# 在150的配置中添加
try_files $uri/index.html /index.html;
```

- 180代理服务器配置

```
location /api/ {
    proxy_pass http://10.0.0.181;
    proxy_set_header Host "api.feng.org";
}

location /static { # 在150配置了try_files进行兜底，防止后台发起重定向
    proxy_pass http://10.0.0.150;
    proxy_set_header Host "static.feng.org";
}
```

## 代理服务器实现数据缓存

- 前置条件：各服务器时间和时区先统一，方便测试

```
# Proxy Server 配置

# 定义缓存
proxy_cache_path /tmp/proxycache levels=1:2 keys_zone=proxycache:20m inactive=60s
max_size=1g;

# path /tmp/proxycache # 表示缓存数据的存放路径，要保证nginx，有写权限
# levels 表示缓存数据目录层级，16进制表示，levels=1:2 表示第一级有16个目录，0-f，第一级中每个目录下有16*16个子目录00-ff
# keys_zone=name:size
    ## name 表示缓存名称，先定义后使用 size 表示该zone 空间大小
# inactive: 表示缓存数据生命周期，默认值10分钟

server {
    listen 80;
    server_set_header Host "statci.feng.org";

    location /static {
        proxy_pass http://10.0.0.150;
        proxy_set_header Host "static.feng.org";

        proxy_cache proxycache;          # 表示使用缓存
        proxy_cache_key $request_uri;    # 使用什么表示资源，示例中使用url
        proxy_cache_valid 200 302 301 90s # 以下状态码可以使用缓存
        proxy_cache_valid any 2m;       # 此处一定要写，否则缓存不生效
    }
}
```

```
        add_header X-Cache $upstream_cache_status # 添加响应头，可以用来查看是否命中缓存
    }

# 重载，生成缓存目录
nginx -s reload
```

- `inactive time` 和 `proxy_cache_valid` 的区别

```
# 控制缓存项在多长时间内没有被访问就会被删除
inactive time
# 控制基于 HTTP 响应状态码的缓存有效期。多少秒内不会向后端请求资源，而是使用缓存资源
proxy_cache_valid 状态码|any time
```

## 实现客户端IP地址透传

- 修改代理服务器配置，透传真实客户端IP

```
server {
    listen 80;
    server_name www.feng.org;

    location / {
        proxy_pass http://10.0.0.150;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

- 后端服务器配置

```
location / {
    return 200 ${remote_addr}---${http_x_real_ip}----${http_x_forwarded_for};
}
```

## 实现http协议反向代理的负载均衡

- 在实现Nginx反向代理的基础上，可以基于`ngx_http_upstream_module`模块实现后端服务器的分组，权重分配，状态监测，调度算法等高级功能
- 相关指令

```
# 定义一个后端服务器组
upstream name { server address [parameters] } # 定义一个后端服务器组，可以包含一台或多台服务器； 定义好后在proxy_pass指令中引用，作用域http

server address [parameters] # 在upstream中定义一个具体的后端服务器，作用于upstream; address指定后端服务器，可以是IP，主机名，UNIX_Socket，可以加端口号
```

- 负载均衡的实现

```
# upstream 作用域是http
# 将多台服务器设置为一个服务器组
upstream group1 {
    server 10.0.0.150;
    server 10.0.0.153;
}

upstream group2 {
    server 10.0.0.153:8080;
    server 10.0.0.176:8080;
}

# 将数据分流到整个服务器组，然后根据组内策略分配流量
# 反向代理服务器配置
server {

    listen 80;
    server_name www.m99-magedu.com;
    default_type text/html;

    location / {
        proxy_pass http://group1;
        proxy_set_header host $http_host;
    }
}

server {
    listen 80;
    server_name www.m99_magedu.net;
    default_type text/html;
    location / {
        proxy_pass http://group2;
        proxy_set_header host $http_host;
    }
}

# 150服务器配置
server {
    listen 80;
    root /var/www/html/www.m99-magedu.com;
    server_name www.m99-magedu.com;
}

# 153服务器配置
server {
    listen 80;
    root /var/www/html/www.m99-magedu.com;
    server_name www.m99-magedu.com;
}

vim www.m99-magedu.net.conf
server {
    listen 8080;
    root /var/www/html/www.m99-magedu.net;
    server_name www.m99-magedu.net;
```

```
}
```

# 在不设置任何策略时，默认多台机器轮流接受请求

```
# 176服务器配置
vim www.m99-magedu.net.conf
server {
    listen 8080;
    root /var/www/html/www.m99-magedu.net;
    server_name www.m99-magedu.net;
}
```

- 设置权重

```
# group1组的两台服务器的调度比例为3:1
# 可以根据weight的配置，控制流量比
upstream group1 {
    server 10.0.0.210 weight=3;
    server 10.0.0.159;
}
```

- 限制最大活动连接数

```
# 10.0.0.210 同时只能维持两个活动链接
upstream group1 {
    server 10.0.0.150 max_conns=2;
    server 10.0.0.153;
}

# 实验
# 150服务器的配置
server {
    listen 80;
    root /var/www/html/www.m99-magedu.com;
    server_name www.m99-magedu.com;
    # 限制响应速度
    limit_rate 10k;
}

# 在服务器目录下生成一个待下载文件，后续实验用
dd if=/dev/zero of=/var/www/html/www.m99-magedu.com/test.img bs=1M count=500

# 153服务器配置（同153）
server {
    listen 80;
    root /var/www/html/www.m99-magedu.com;
    server_name www.m99-magedu.com;
    # 限制响应速度
    limit_rate 10k;
}

# 创建4个窗口，都执行wget www.m99-magedu.com/test.img
# 保证153和150各有两个连接在占用
# 由于150的后面配置了max_conns=2，所以后续的所有访问都不会再交给150
```

```
# 后续的所有流量全部分给153处理
```

```
# 可以通过ss -tnl命令看到当前连接
```

- 后端服务器健康性检查
  - Nginx的upstream指令对于后端服务器的健康性检查是被动检查，其步骤如下
    - 如果不可用，就调度到别的服务器，并记录1次
    - 当到达指定次数，则规定时间内，不会在向该服务器调度请求
    - 直到超过规定时间，再次向该服务器调度请求，如果依然不可用，则继续等待一个规定时间段，直至该服务器恢复
  - 重点两个参数
    - 判定服务器不可用的指定次数（默认1次）
    - 确定服务器不可用后，规定时间探测一次的指定时间（默认10s）

```
# 相关指令和参数
```

```
max_fails=number # 指定次数  
fail_timeout=time # 制定时间
```

- 设置备用服务器

```
# 设置backup，当10.0.0.210和10.0.0.159都不可用时，请求会被调度到10.0.0.213
```

```
upstream group1 {  
    server 10.0.0.210;  
    server 10.0.0.159;  
    server 10.0.0.213 backup; # 在组内其他服务器没死光前，213不会被使用  
}
```

- 设置后端服务器平滑下线

```
修改Proxy Server配置，将10.0.0.210下线
```

```
upstream group1 {  
    server 10.0.0.210 down;  
    server 10.0.0.159;  
}  
  
#重新生效  
nginx -s reload # 不能使用systemctl restart nginx
```

```
# 等待一段时间，等所有的连接都被调度到159后，则可以肆无忌惮的对210进行操作
```

## 负载均衡调度算法(面试重点)

- 源IP地址hash

- ip\_hash算法只使用ipv4的前24位做hash运算，如果客户端前24位一致，则会被调度到同一台后端服务器

```
# Proxy_Server 配置
upstream group1 {
    ip_hash;          # 只要加上ip_hash即可
    server 10.0.0.210;
    server 10.0.0.159;
}
```

- 使用自行指定的key做hash调度

```
# 三台server权重一样，调度算法hash($remote_addr)%3,值为0,1,2根据不同的值调度到不同server
upstream group1 {
    hash $remote_addr;
    server 10.0.0.210;
    server 10.0.0.159;
    server 10.0.0.213;
}

# 三台server权重不一样，调度算法hash($remote_addr)%(1+2+3),值为0,1,2,3,4,5
# 0 调度到210
# 1, 2调度到159
# 3,4,5调度到213
upstream group1 {
    hash $remote_addr;
    server 10.0.0.210 weight=1;
    server 10.0.0.159 weight=2;
    server 10.0.0.213 weight=3;
}

# 也可以根据request_uri进行调度，不同客户端访问同一个资源会被调度到同一个后端服务器上
upstream group1 {
    hash $request_uri
    server 10.0.0.210;
    server 10.0.0.159;
    server 10.0.0.213;
}
```

- 最少连接调度算法

```
# 根据哪台服务器上的连接数少，就调度到哪里，如果多台服务器连接数一致，则这几台服务器按照轮询的方式进行调度
upstream group1 {
    least_conn;
    server 10.0.0.210;
    server 10.0.0.159;
}
```

- 一致性hash

```
# 问题：如果组内的服务器数量变化，则ip访问的服务器也会发生变化
```

```
# 示例
```

```
upstream group1 {  
    hash $remote_addr;  
    server 10.0.0.210;  
    server 10.0.0.159;  
    server 10.0.0.213;  
}
```

```
# 此时如果增加了一台服务器
```

```
调度算法从hash%3，变成了hash%4，则此时会有同hash值的访问，变更了访问的组内服务器
```

- 一致性hash定义

- 一致性hash是一种用于分布式系统中数据分片和负载均衡的算法，其中的“hash环”是该算法的核心概念之一
- 在一致性hash中，所有可能得数据节点或服务器被映射到一个虚拟的环上，这个环的范围通常是一个固定的hash空间，比如0到 $2^{32}-1$ ,每个数据节点或服务器被映射到环上的一个点，通过对对其进行哈希计算得到，这个哈希值的范围也是在0到 $2^{32}-1$ 之间
- 在这个环上，数据被分散到最接近它的节点。当有新的数据要存储时，首先通过hash计算得到该数据的哈希值，然后在环上找到离这个hash值最近的节点，将数据存储在这个节点上，同样，当要查询数据时，也就是通过hash计算得到数据的hash，然后找到最近的节点进行查询
- 由于hash环是一个环形结构，节点的添加和删除对整体的影响相对较小。当添加或删除节点时，只有相邻的节点受到影响，而其他节点保持不变。这使得一致性hash算法在分布式系统中能够提供较好的负载均衡性能，同时减小了数据迁移的开销
- 总的来说，一致性hash中的hash环是通过hash计算将数据节点映射到环上，以实际数据分片和负载均衡的分布式算法

```
upstream group1 {  
    hash $remote_addr consistent;  
    server 10.0.0.210;  
    server 10.0.0.159;  
    server 10.0.0.213;  
}
```

## 综合练习

- 实现http自动重定向至https，并将客户端https请求通过负载均衡的方式反向代理到后端的多台http服务器上
- 略

## Nginx的四层代理和负载

### 实现TCP协议的反向代理

- 实验配置

- Client ----- 10.0.0.208
- Proxy Server ----- 10.0.0.181
- Mysql Server ----- 10.0.0.150

- Redis Server ----- 10.0.0.153

```
# Proxy配置,写在http模块外
stream {
    server {
        listen 3306;
        proxy_pass 10.0.0.150:3306;
    }
    server {
        listen 6379;
        proxy_pass 10.0.0.153:6379;
    }
}
```

## 实现TCP协议的负载均衡

```
stream {
    upstream mysql {
        server 10.0.0.210:3306;
        server 10.0.0.159:3306;
    }
    upstream redis {
        server 10.0.0.210:6379;
        server 10.0.0.159:6379;
    }

    server {
        listen 3306;
        proxy_pass mysql;
    }

    server {
        listen 6379;
        proxy_pass redis;
    }
}
```

## 实现FastCGI代理

- 需要使用的模块

```
ngx_http_fastcgi_module.html
```

- 常用配置项

```
fastcgi_index name; # 后端FastCGI服务器默认资源, 默认为空

fastcgi_pass address; # 指定后端FastCGI服务器地址, 可以写IP:Port, 也可以指定socket文件

fastcgi_param parameter value [if_not_empty];
# 设置传递给FastCGI服务器的参数值, 可以是文本, 变量或组合, 可用于将Nginx的内置变量赋值给自定义key
```

- PHP环境准备

```
# 安装php-fpm
apt install php8.1-fpm

# php的三个配置目录
cli      # php命令行的配置
fpm      # 前端通过apache或nginx和php进行网络交互
mods-available

# 重点: php提供了一个fpm接口, apache或nginx通过php提供的fpm接口, 进行交互

# 修改/etc/php/8.1/fpm/pool.d/www.conf这个配置文件
# 使用 IP:port 监听, Nginx 和 php-fpm 通信走网络 FastCGI 接口
vim /etc/php/8.1/fpm/pool.d/www.conf
;listen = /run/php/php8.1-fpm.sock    # 注释掉这行
listen = 127.0.0.1:9000               # 使用端口进行数据交互

#取消下面几行注释
[root@ubuntu ~]# vim /etc/php/8.1/fpm/pool.d/www.conf
;pm.status_path = /status    # 将行注释, 防止和nginx和status页面混淆
pm.status_path = /php-status
pm.status_listen = 127.0.0.1:9001    # 9001监视状态, 9000走业务

# 重载配置文件
systemctl reload php8.1-fpm.service
```

- 配置Nginx转发到FastCGI

```
server{
    listen 80;
    server_name www.m99-magedu.com;
    root /var/www/html/www.m99-magedu.com;

    # /etc/nginx/fastcgi.conf 和 /etc/nginx/fastcgi_params 都是用于配置 FastCGI 的默认参数的文件
    # 这两个文件中都用 fastcgi_param 指令定义了很多要传给后端 FastCGI 接口的参数

    #反向代理 php
    location ~ \.php$ {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        include fastcgi.conf;
    }

    # php-fpm status, 路径要与 /etc/php/8.1/fpm/pool.d/www.conf 中的
    # pm.status_path = /php-status 一致
    location /php-status {
        fastcgi_pass 127.0.0.1:9001;
        include fastcgi.conf;
    }

    # php-fpm 心跳检测
    # /etc/php/8.1/fpm/pool.d/www.conf在这个文件中
    # 将ping.path = /ping ; ping.response = pong 这两行注释取消
```

```

location = /ping{
    fastcgi_pass 127.0.0.1:9000;
    include fastcgi.conf;
}

#nginx status
location /status {
    stub_status;
}

}

#在浏览器中测试
http://www.m99-magedu.com
http://www.m99-magedu.com/index.php
http://www.m99-magedu.com/test.php
http://www.m99-magedu.com/ping
http://www.m99-magedu.com/status
http://www.m99-magedu.com/status?json
http://www.m99-magedu.com/status?ful

```

## 企业级WEB应用服务器TOMCAT

### Java动态网页技术

#### Servlet

- Servlet是Java编程语言的一种服务端程序，主要用于扩展Web服务器的功能。Servlet运行在Web服务器中，接收并响应来自客户端（通常是Web浏览器）的请求
- 如果要用Servlet构建Web应用，则需要将Html标签嵌套在Java的逻辑代码中，而Html代码有要频繁迭代，相应的，需要重新编译，这种开发模式非常不方便

#### JSP

- JSP (Java Server Pages) 是一种用于创建动态Web页面的Java技术，与Servlet一样，JSP也是在服务器端执行的，但其主要目标是简化Web页面的开发，允许开发者在Html页面中嵌入Java代码
- 以下是一些关键的JSP特性和与Servlet的区别
  - 页面描述语言：JSP使用一种称为JSP标记的页面描述语言，允许开发者在HTML页面中嵌入Java代码片段。这些标记以<% %>为界，使得在页面中嵌入动态内容变得更加容易。
  - 易读性：JSP的语法更接近HTML，因此对于前端开发者来说更易读写，开发者可以在HTML页面中嵌入Java代码，而无需像Servlet那样在Java代码中嵌套HTML字符串
  - 简化开发：JSP旨在简化动态WEB页面的开发，使得将Java代码嵌入HTML更加直观，相对于Servlet，JSP提供了更高层次的对象
  - 隔离逻辑和展示：Servlet通常在Java代码中生成HTML，而JSP允许在HTML中插入Java代码，这有助于更好地隔离逻辑和展示
  - 自定义标签库：JSP允许开发者定义和使用自定义标签库，以便将特定的逻辑模块化，提高代码的可重用性
  - 独立性：Servlet是一个独立的Java类，而JSP页面实际上在首次访问时被Servlet容器翻译成Java代码并编译成Servlet。因此，从本质上说，Servlet和JSP最终都是以Servlet的形式在服务器上执行

- 虽然Servlet和JSP在某些方面有区别，但他们通常一起使用，Servlet用于处理业务逻辑，而JSP用于呈现用户界面。在实际的WEB应用中，Servlet和JSP经常一起协同工作，已达到更好的代码组织和可维护性。

## MVC

- MVC (Model-View-Controller) 是一种设计模式，使得每个组件可以独立地变化而不影响其他组件。这种分离有助于提高代码的可维护性，可扩展性和复用性。常见的应用场景包括Web开发框架，图形用户界面应用程序等
- 模型 (Model)：模型表示应用程序的数据和业务逻辑。它负责处理数据的存储，检索和更新，同时包含应用程序的业务规则。模型通常不包含有关用户界面或展示方式的信息，而是专注于处理数据
- 视图 (View)；视图负责用户界面的呈现和展示，它接收来自模型的数据，并将其以用户友好的方式显示。视图不负责处理数据。而只关注展示和用户交互
- 控制器 (Control)：控制器充当模型和视图之间的协调者。它接收用户的输入（通常来自用户界面），然后根据输入更新模型或视图。控制器负责处理用户请求，更新模型的状态以及选择适当的视图进行展示

## REST

- REST是一种基于网络的软件架构风格，代表"Representational State Transfer"（表示层状态转移）。它是由Roy Fielding在他的博士论文中提出的，并成为构建分布式系统和网络应用程序的一种设计原则

## JDK和JRE

jRE(Java Runtime Environment)和JDK (Java Development Kit) 是Java平台中两个不同但相关的组件，他们之间有以下关系：

- JDK包含JRE：JDK是Java开发工具包，它是Java开发的完整包。JDK包括了JRE，以及用于Java开发的一系列工具和库，比如编译器 (javac)、调试器 (jdb)、文档生成器 (javadoc) 等。因此，可以说JRE是JDK的一个子集
- JRE用于运行Java程序：JRE是Java运行时环境，提供了在计算机上运行Java应用程序所需的所有组件。它包含Java虚拟机 (JVM)、Java类库、Java命令行工具和其他支持文件。当用户仅需要运行Java程序时，安装JRE就足够了
- JDK用于开发和运行Java程序：JDK不仅包含了JRE，还包含了用于开发Java应用程序的工具和库。开发人员使用JDK中的工具编写、编译和调试Java代码、JDK是面向Java开发人员的完整工具包、而JRE主要面向终端用户，用于执行Java应用程序。

总而言之，JDK是适用于Java开发的完整套件，包含JRE和其他开发工具，而JRE则是仅包含运行时环境的一部分，如果你只是希望运行Java程序，安装JRE就是足够了。如果你要进行Java应用程序的开发，那么你需要安装JDK。

## Oracle JDK

```
# Ubuntu中安装  
dpkg -i jdk-11.0.22_linux_x64_bin.deb  
  
# Rocky中安装  
rpm -ivh jdk-11.0.22_linux-x64_bin.rpm
```

## OpenJDK

```
# Ubuntu安装  
apt update; apt list openjdk*  
apt install openjdk-11-jdk -y  
  
# rokcy安装  
yum list java*openjdk*  
yum install java-11-openjdk -y
```

## Tomcat基本使用

### Tomcat介绍

Tomcat(全程为Apache—Tomcat)是一个开源的、轻量级的应用服务器，由Apache软件基金会开发和维护，它实现了Java—Servlet、JavaServerPages(JSP)和Java Expression Language (EL) 等Java EE规范。并提供了一个运行这些Web应用程序的环境。在使用Tomcat时，将javaWeb应用程序（包括Servlet、JSP等文件）部署到Tomcat服务器中，然后通过HTTP协议访问这些应用程序，Tomcat提供了一个简单而强大的方式来托管和运行JavaWeb应用程序。

Tomcat具有以下一些特点

- Servlet容器：Tomcat是一个Servlet容器，负责处理和执行Java Servlet。Servlet是一种用Java编写的服务端程序。用于处理Web请求和生成动态Web内容；
- JSP支持：Tomcat还支持JSP，一种将Java代码嵌入HTML中的技术，用于创建动态WEB页面
- 轻量级：Tomcat是轻量级应用服务器，易于安装和配置。它专注于基本的Servlet和JSP支持
- 开源
- 连接器支持：Tomcat支持多种连接器，可以与不同的Web服务器进行集成。最常见的是与Apache Http服务器一起使用的AJP连接器
- 管理工具
- 安全性
- 跨平台性

### jar包和war包

- Jar包可以直接在JVM上运行
- War包则必须部署在tomcat上

### Tomcat安装

- Ubuntu中安装

```
# Ubuntu  
apt install tomcat9*  
  
systemctl status tomcat9.service  
  
# tomcat默认使用8080端口  
  
# 浏览器上使用IP:8080查看
```

```
# ubuntu中的包安装版tomcat存在Bug（无法将资源配置到catalina.base目录之外）
# 因此使用手动下载官方软件包，然后编译安装

wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.87/bin/apache-tomcat-
9.0.87.tar.gz

tar xf apache-tomcat-9.0.87.tar.gz -C /usr/local/

cd /usr/local

ln -sv apache-tomcat-9.0.87/ tomcat

# 添加环境变量
ln -sv /usr/local/tomcat/bin/* /usr/local/bin/

# 启动
catalina.sh start

# 访问http://IP:8080/查看

# 用脚本停止
shutdown.sh

# 添加用户并修改文件属主属组
useradd -r -s /sbin/nologin tomcat
chown -R tomcat.tomcat /usr/local/tomcat/

# 服务脚本
cat /lib/systemd/system/tomcat.service
[Unit]
Description=Tomcat
After=syslog.target.network.target

[Service]
Type=forking
Environment=Java_HOME=/usr/lib/jvm/jdk-11-oracle-x64/
ExecStart=/usr/local/tomcat/bin/startup.sh
ExecStop=/usr/local/tomcat/bin/shutdown.sh
PrivateTmp=true
User=tomcat
Group=tomcat

[Install]
WantedBy=multi-user.target

# 加载到daemon
systemctl daemon-reload

# 查看tomcat状态
systemctl status tomcat.service
```

- Rocky中安装

```
 wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.87/bin/apache-tomcat-9.0.87.tar.gz

 tar xf apache-tomcat-9.0.87.tar.gz -C /usr/local/

 ln -sv /usr/local/apache-tomcat-9.0.86/ /usr/local/tomcat

 # 主要使用的脚本
 ls /usr/local/tomcat/bin/

 # 加环境变量
 cat /etc/profile.d/tomcat.sh

#!/bin/bash
PATH=/usr/local/tomcat/bin:$PATH

# 执行该脚本
source /etc/profile.d/tomcat.sh

# 创建用户
useradd -r -s /sbin/nologin tomcat
chown -R tomcat.tomcat /usr/local/tomcat/

# 服务脚本
cat /lib/systemd/system/tomcat.service
[Unit]
Description=Tomcat
After=syslog.target network.target

[Service]
Type=forking
Environment=JAVA_HOME=/usr/bin/jvm/jdk-11-oracle-x64/
ExecStart=/usr/local/tomcat/bin/startup.sh
ExecStop=/usr/local/tomcat/bin/shutdown.sh
PrivateTmp=true
User=tomcat
Group=tomcat

[Install]
WantedBy=multi-user.target

# 加载服务脚本
systemctl daemon-reload

# 查看状态
systemctl status tomcat.service

# 先用脚本停止，再用服务管理启动
shutdown.sh

systemctl start tomcat.service

# 加开机启动项
systemctl enable tomcat.service
```

## Tomcat组成

```
ls -l /usr/local/tomcat/
bin                         # 管理脚本文件目录
BUILDING.txt
conf                        # 配置文件目录
CONTRIBUTING.md
lib                          # 库文件目录
LICENSE
logs                         # 日志目录
NOTICE
README.md
RUNNING.txt
temp                         # 临时文件目录
webapps                     # 应用程序目录    相当于/var/www/html
work                         # Jsp编译后的结果文件, 建议提前预热访问, 升级应用后, 删除此目录数
据才能更新
```

## tomcat和catalina的关系

Tomcat的核心分为三个部分

- Web容器：用于处理静态页面
- JSP容器：把jsp翻译成一般的Servlet
- catalina：是一个Servlet容器，用于处理Servlet

## tomcat组件

每一个组件都有一个Java类实现，在配置文件中通常用className来指定

## Tomcat常用配置项

```
tree /usr/local/tomcat/conf/
Catalina --localhost      # 每个项目的单独配置文件存放目录
catalina.policy           # java程序对各目录的权限配置
catalina.properties       # tomcat环境变量配置以及jvm调优相关参数
context.xml                # web项目默认的上下文配置, 包括数据库连接池, JNDI资源, 会话管理
等, 每个项目也可以单独设置
jaspic-provider.xml       # jaspic配置
jaspic-providers.xsd      # 上述文件中的标签取值约束
logging.properties         # tomcat的日志配置, 定义日志级别, 格式等
server.xml                 # 主配置文件, 用于配置整个Tomcat服务器的全局设置, 包括连接器, 虚拟
主机, 引擎等, 定义了Tomcat基本结构和行为
tomcat-users.xml          # 用户配置文件, 用于配置Tomcat的用户, 角色和其对应的权限
tomcat-users.xsd          # 上述文件中的标签取值约束
web.xml                   f NM./.''
L; '
\
```

## /usr/local/tomcat/conf/web.xml

/usr/local/tomcat/conf/web.xml文件是Apache Tomcat中用于配置Web应用程序的部署描述符(Deployment Description)文件，采用XML格式。在这个文件中，可以配置许多与WEB应用程序相关的设置。

- 常见的配置及其含义

```
<!-- mime类型-->
<mime-mapping>
  ...
</mime-mapping>
  ...
<mime-mapping>
  <extension>zirz</extension>
  <mime-type>application/vnd.zul</mime-type>
</mime-mapping>
<mime-mapping>
  <extension>zmm</extension>
  <mime-type>application/vnd.handheld-entertainment+xml</mime-type>
</mime-mapping>

<!-- 默认主页-->
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>

<!-- 错误页面，后续详解-->
<error-page>
</error-page>
```

## /usr/local/tomcat/conf/server.xml

/usr/local/tomcat/conf/server.xml文件是Apache Tomcat中的主要配置文件之一，用于配置Tomcat服务器的全局设置，连接器(Connectors)，虚拟主机(Virtual Hosts)等。以下是一些server.xml文件中可能包含的主要配置项及其含义的简要解释

server.xml的组件结构

- Server: 服务器，Tomcat运行的进程实例，一个Server中可以有多个service，但通常就一个
- Service: 服务，用来组织Engine和Connector的对应关系，一个Service中只有一个Engine
- Connector: 连接器，负责客户端的HTTP、HTTPS、AJP等协议连接，一个Connector只属于某一个Engine
- Engine: 引擎，用来响应并处理用户请求，一个Engine上可以绑定多个Connector
- Host: 虚拟主机，可以实现多虚拟主机，例如使用不同的主机头区分
- Context: 应用的上下文，配置特定url路径映射和目录的映射关系：url=>directory
- 总结：
  - 一个Service中只有一个Engine，
  - 一个Engine可以有多个Connector，

- 但是一个Connector只属于一个Engine

## server.xml文件解析

```

<!-- 最外层 -->
<!-- Server其他参数
  className 指定实现此组件的java类, 默认org.apache.catalina.core.StandardServer
  address 监听端口绑定的地址, 不指定默认为 localhost
-->
<!-- 因此可以通过telnet在本地连接8005, 通过SHUTDOWN关闭tomcat
  telnet 127.0.0.1 8005
  > 输入SHUTDOWN
  > tomcat即可被关闭
-->
<Server port="8005" shutdown="SHUTDOWN">

  <!-- 监听器 -->
  <Listener className="" />
  <!-- 配置全局命名资源 -->
  <GlobalNamingResources>
    <Resource key="value" key="value"... />
  </GlobalNamingResources>
  ...

  <!--
    元素定义了Tomcat中的一个服务（实例），一个tomcat服务器可以开多个实例，不同实例可以监听不同的
    端口，使用不同的配置和部署不同的web应用程序，每个实例被定义成一个Service，一个Service可以包
    含多个连接器和多个虚拟主机
  -->

  <!--
    className 指定实现的Java类, 默认org.apache.catalina.core.StandardService
    name 指定实例名称
  -->
  <Service className="" name="">
    <!--
      <Executor> 用于配置Service的共享线程池
      className 指定实现的Java类, 默认
      org.apache.catalina.core.StandardThreadExecutor
        name 指定线程池名称, 其他组件通过名称引用线程池
        threadPriority 线程优先级, 默认值为5
        daemon 线程是否以daemon的方式运行, 默认为true
        namePrefix 执行器创建每个线程时的名称前缀, 最终线程的名称为
        namePrefix+threadNumber
        maxThreads 线程池激活的最大线程数量, 默认为200
        minSpareThreads 线程池中最少空闲的线程数量, 默认为25
        maxIdleTime 空闲线程被销毁的保留时长, 默认60000ms, 即空闲线程要保留一分钟才销
        销
        maxQueueSize 可执行任务的最大列数, 达到队列上限时的连接请求将被拒绝
        prestartminSpareThreads 在启动executor时是否立即创建minSpareThreads个线程,
        默认false, 即需要时才创建
    -->
    <Executor className="" name="" threadPriority="" daemon="" namePrefix=""
    maxThreads="" minSpareThreads="" maxIdleTime="" maxQueueSize=""
    prestartminSpareThreads="" />

```

```

<!--
<Connector ... />
    用于配置Tomcat的连接器，负责处理客户端与Tomcat之间的通信，一个service中可以有多个
    connector，每个connector定义了一个端口和协议的组合，允许Tomcat在不同的端口上监听不同的网络协
    议或配置不同的连接器属性

        port          指定监听端口
        address       指定监听IP，默认本机所有IP
        protocol      协议版本
        connectionTimeout 超时时长
        redirectPort 安全连接重定向的端口
        executor      指定共享线程池
        acceptCount   等待队列的最大长度，通常在tomcat所有处理线程均处于繁忙状态
        时，新发来的请求将被放置于等待队列中
        maxConnections 允许建立的最大连接数，当maxConnections小于acceptCount
        时，超出maxConnections的连接请求将被接收，但不会与之建立连接
        keepAliveTimeout 长连接状态的超时时间。超出该值时，长连接将关闭
        enableLookups  是否通过DNS查询获取客户端的主机名，默认为true，应设置为
        false可减少资源消耗
        compression    是否压缩数据。默认为off。on 表示只压缩text文本，force表示
        压缩所有内容
        useSendfile   是否启用sendfile的功能，默认为true，启用该属性将会禁止
        compression属性
        URLEncoding   用于指定编码URI的字符编码，默认UTF-8

-->
<Connector port="" protocol="" conectionTimeout="" redirectPort=""
maxParameterCount="" />

<!--
<Engine...> </Engine>
    engine是service组件中用来分析协议的引擎机器，它从一个或多个connector上接收请求，  

    并将请求交给对应的虚拟主机进行处理，最后返回完整的响应数据给connector，通过connector将响应数据  

    返回给客户端
    engine元素必须嵌套在每个service中，且engine必须在其所需要关联的connector之后，这  

    样在engine前面的connector都可以被此engine关联，而在engine后面的connector则被忽略。
    一个service只允许有一个engine，其内可以嵌套一个或多个Host作为虚拟主机，且至少一个  

    host要和engine中的默认虚拟主机名称对应。除了host，还可以嵌套realm和valve组件

        className     指定实现的Java类，默认org.apache.catalina.core.StandardEngine
        defaultHost   默认虚拟主机，在Engine中定义的多个虚拟主机的主机名称中至少有一个跟
        defaultHost定义的主机名称相同（相当于nginx上的default_server）
        name          指定组件名称
        jvmRoute      在启用session粘性时指定使用哪种负载均衡的标识符。所有的tomcat
        server实例中该标识符必须唯一，它会追加在session标识符的尾部，因此能让前端代理总是将特定的
        session转发至同一个tomcat实例上。

-->
<Engine className="" defaultHost="" name="" jvmRoute="">

<!--
    在Engine内部配置了一个安全领域（Realm）用于身份验证和授权，该配置对Engine下的
    所有Host生效

        className     指定实现的java类，不同的类表示组件功能不同，属性不同
        常用类：

```

	<b>JAASRealm</b>	基于java Authentication and Authorization Service
实现用户认证		
	<b>JDBCRealm</b>	通过JDBC访问某关系型数据库表，实现用户认证
	<b>JNDIRealm</b>	基于JNDI使用目录服务实现用户信息的获取
	<b>MemoryRealm</b>	查找tomcat-user.xml文件实现用户信息的获取
	<b>UserDatabaseRealm</b>	基于UserDatabase文件（通常是tomcat-user.xml）
实现用户认证		
-->		
	<b>&lt;Releam</b> <code>className=""</code>	
	<b>&lt;/Releam&gt;</b>	
	<b>&lt;!--&gt;</b>	
	<b>&lt;Host&gt;&lt;/Host&gt;</b>	
	host容器用来定义虚拟主机，engine从connector接收到请求进行分析后，会将相关的属性参数传递给对应的（筛选方式是从请求首部的host字段和虚拟主机名进行匹配）虚拟host进行处理。如果没有合适的虚拟主机，则传递给默认虚拟主机。因此每个容器中必须至少定义一个虚拟主机，且必须有一个虚拟主机和engine容器中定义的默认虚拟主机名称相同，其下级标签包括： Alias, Cluster, Listener, valve, Realm, Context	
		常用参数
	<b>className</b>	指定实现的java类，默认
	<b>org.apache.catalina.core.StandardHost</b>	
	<b>name</b>	指定主机名，忽略大小写，可以使用通配符，匹配优先级低
	<b>appBase</b>	此项目的程序目录（网站家目录）
	<b>xmlBase</b>	此虚拟主机上的context xml目录
	<b>autoDeploy</b>	Tomcat处于运行状态时放置于appBase目录中的应用程序文件是否自动进行部署，默认true
	<b>unpackWARs</b>	自动解压缩war包
	<b>workDir</b>	该虚拟主机的工作目录，每个webapp都有自己的临时IO目录，默认该工作目录为\$CATALINA_BASE/work
-->		
	<b>&lt;Host&gt;</b>	
	<b>&lt;!--&gt;</b>	
	定义一个特定功能的组件，可以有多条，按照定义顺序生效	
	<b>className</b>	指定实现的Java类，不同的class组件功能不同，属性不同
		常用功能：
	<b>AccessLogValve</b>	访问日志
	<b>ExtendedAccessValve</b>	扩展功能的访问日志
	<b>RequestDumperValve</b>	请求转储
	<b>RemoteAddrValve</b>	基于远程地址的控制访问(类似Nginx的allow和deny)
	<b>RemoteHostValve</b>	基于远程主机名的访问控制
	.....	
-->		
	<b>&lt;valve&gt;</b>	
	<b>&lt;/valve&gt;</b>	
	<b>&lt;valve&gt;</b>	
	<b>&lt;/valve&gt;</b>	
	.....	
	<b>&lt;/Host&gt;</b>	

```

<!--
<Context></Context>
配置WEBAPP的上下文, tomcat对请求uri与context中定义的path进行最大匹配前缀的
规则进行挑选, 从中选出使用哪个Context来处理该HTTP请求, 必须要有一个context的path为空字符串,
用于处理无法被现有规则命中的URI
类似于location / {}

常用参数
className          指定实现的java类, 默认值
org.apache.catalina.core.StandardContext
cookie             默认为true, 表示启用cookie来标识session
path               定义生效的URL路径规则, 如果配置是独立文件, 此属性可以为
空, tomcat可以用文件名来推出path
...
-->
<Context>

</Context>

</Engine>

</Service>
</Server>

```

## Tomcat工作原理

1. 客户端向服务器发请求
2. 由连接器接受这个请求到指定端口
  - 连接器就是server.xml中的
  - 通常连接器有两种
    - HTTP, 默认8080
    - AJP, 默认8009
3. 连接器接收数据后, 交给引擎处理
4. 将访问的资源解析到虚拟主机Host
5. 通过Host的name进行匹配
  1. 匹配后, 通过context获取请求, 并通过mapping table找到-----匹配【·
    - 对应的servlet
  2. servlet (server applet) :是以servlet编码方式写的一个tomcat的服务器端
  3. servlet是在web.xml 里定义的
6. 由servlet去封装request和response
7. 封装好后将结果返回引擎, 再从引擎返回到连接区, 在返回给客户端

# 各组件的作用及关系

## 1. Server

- **顶层组件:** `<Server>` 是 Tomcat 配置文件的顶层组件，用来定义整个 Tomcat 服务器实例的配置。
  - 作用
    - 代表整个服务器的运行实例。
    - 包含一个或多个 `<Service>`。
    - 负责监听服务器生命周期（启动和关闭）事件。
    - 通过 `port` 属性配置一个端口（通常是 `8005`），用于监听关闭命令。
  - 关键属性
    - `port`: 监听关闭命令的端口号。
    - `shutdown`: 接收关闭命令的字符串。

```
xmlCopy code<Server port="8005" shutdown="SHUTDOWN">
...
</Server>
```

## 2. Service

- **中间组件:** `<Service>` 是 Server 的子组件，一个 Server 可以包含一个或多个 Service。
  - 作用
    - 表示一个具体的服务实例。
    - 负责将 `Connector` 和 `Engine` 组合在一起工作。
    - 允许在同一个服务器中运行多个服务（比如一个服务处理 HTTP，另一个处理 HTTPS）。
  - 关键属性
    - `name`: Service 的名称。

```
xmlCopy code<Service name="catalina">
...
</Service>
```

## 3. Engine

- **核心组件:** `<Engine>` 是 Service 的子组件，一个 Service 必须包含一个 Engine。
  - 作用
    - 表示整个服务的核心引擎，负责将来自 Connector 的请求分发到正确的 Web 应用（Context）。

- 每个 Engine 必须配置一个默认的 `Host`，以便处理请求时无法匹配具体虚拟主机的情况。

- **关键属性**

:

- `name`：Engine 的名称。
- `defaultHost`：默认虚拟主机的名称，用于处理无法匹配到其他 Host 的请求。

```
xmlCopy code<Engine name="Catalina" defaultHost="localhost">
    ...
</Engine>
```

## 4. Connector

- **通信组件**：`<Connector>` 是 Service 的子组件，用于定义服务的通信方式。

- **作用**

:

- 负责监听特定的协议端口（如 HTTP、HTTPS、AJP 等），将收到的请求传递给 Engine。
- 一个 Service 可以包含多个 Connector，以支持不同协议或端口。

- **关键属性**

:

- `port`：监听的端口号。
- `protocol`：通信协议（如 HTTP/1.1、AJP/1.3）。
- `URIEncoding`：编码方式（如 UTF-8）。

```
xml
```

```
Copy code
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
redirectPort="8443" />
```

## 组件之间的关系

1. `Server → Service`

- 一个 Server 可以包含多个 Service，每个 Service 是一个独立的服务实例。

2. `Service → Engine`

- 每个 Service 必须包含一个 Engine，Engine 是整个服务的核心。

3. `Service → Connector`

- 每个 Service 可以包含多个 Connector，负责监听不同协议或端口的请求。

4. `Engine → Host → Context`

- Engine 负责请求分发，它包含一个或多个虚拟主机（Host）。

- 每个 Host 代表一个虚拟主机（比如不同的域名）。

- 每个 Host 包含多个 Web 应用 (Context) , 每个 Context 对应一个部署的应用。
- 

## 工作流程

### 1. 启动阶段:

- Server 启动后, 初始化所有 Service。
- 每个 Service 初始化对应的 Engine 和 Connector。

### 2. 请求处理阶段:

- Connector 监听特定端口, 接收来自客户端的请求。
- Connector 将请求交给 Engine。
- Engine 根据请求的域名 (Host Header) 选择对应的 Host。
- Host 根据请求的 URI, 将请求分发到对应的 Context (Web 应用)。
- Context 将请求交给具体的 Servlet 或资源处理。

## 示例配置

```
xmlCopy code<Server port="8005" shutdown="SHUTDOWN">
    <!-- 定义一个服务 -->
    <Service name="Catalina">

        <!-- 定义一个 HTTP 协议的连接器 -->
        <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
redirectPort="8443" />

        <!-- 定义一个 AJP 协议的连接器 -->
        <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

        <!-- 定义服务的核心引擎 -->
        <Engine name="Catalina" defaultHost="localhost">

            <!-- 定义一个虚拟主机 -->
            <Host name="localhost" appBase="webapps" unpackWARs="true"
autoDeploy="true">

                <!-- 定义一个 web 应用 -->
                <Context path="" docBase="ROOT" />
                <Context path="/app1" docBase="app1" />
            </Host>

        </Engine>
    </Service>
</Server>
```

## 总结

- **Server** 是顶层容器，负责管理多个 Service 的运行。
- **Service** 将 Connector 和 Engine 绑定在一起，管理整个服务的通信和请求分发。
- **Engine** 是 Service 的核心组件，负责请求的分发。
- **Connector** 负责网络通信，将请求转发给 Engine。
- 各组件紧密协作，构成了 Tomcat 的请求处理体系。

## 日志配置

自定义日志内容配置的转义字符说明

```
https://tomcat.apache.org/tomcat-9.0-doc/config/valve.html
```

默认Web访问日志

```
<Host name="localhost" appBase="webapps"
      unpackWARs="true" autoDeploy="true">

    <valve className="org.apache.catalina.valves.AccessLogValve"
          directory="logs"
          prefix="localhost_access_log" suffix=".txt"
          pattern="%h %l %u \"%r\" %s %b" />

</Host>
```

上述配置解析

```
className="org.apache.catalina.valves.AccessLogValve"      # 日志类
directory="logs"   # 日志文件目录
prefix="localhost_access_log"                            # 日志文件前缀
suffix=".txt"   # 日志文件后缀
pattern="%h %l %u \"%r\" %s %b"                      # 日志文件内容格式
```

更改为json格式的日志配置

```
<valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
      prefix="localhost_access_log" suffix=".txt" pattern="
      {"clientip":"%h","ClientUser":"%l","
      ;authenticated:"%u","AccessTime":"%t","
      method:"%r","status:"%s","SendBytes":"
      ;","%b","Query?"
      string:"%q","partner:":"%
      {Referer}i","AgentVersion":"%{User-Agent}i"}" />
```

```
# 重新加载配置文件
systemctl restart tomcat.service

# 使用jq查看
tail -n 1 /usr/local/tomcat/logs/localhost_access_log.2024-0323.txt | jq
```

# 应用部署

Tomcat部署应用有三种方式：

- 以文件或war包直接部署
  - 将应用文件夹或war包文件直接复制到tomcat的webapps目录下，这样tomcat启动的时候会将webapps目录下的文件夹或war文件的内容当做应用部署
- 编辑/usr/local/tomcat/conf/server.xml配置文件，添加Context标签
  - 在tomcat的server.xml配置文件中的Host节点下增加Context子节点，配置域名，文件路径
- 在/usr/local/tomcat/conf/[Engine]/[Host]目录下创建独立配置
  - 在该目录下新建xml文件，文件名为应用名，然后在配置文件中添加Context子节点，配置域名，文件路径等

## 注意事项

- 访问10.0.0.150/doc

```
<Host name="localhost" appBase="webapps"
      unpackWARs="true" autoDeploy="true">
# 虽然配置文件上是指定在webapps
# 如果访问10.0.0.150:8080/test.html,实际访问路径是webapps/ROOT/test.html
# 如果访问10.0.0.150:8080/docs[examples, host-manager, manager]
# 实际访问路径是webapps/docs[examples, host-manager, manager]
# 如果ROOT目录下有同名的docs等，外面的优先级更高
# 只有直接在10.0.0.150:8080/file，只有在/后直接加文件名，则会从ROOT目录下找文件
# 如果/后是目录名，则该目录与ROOT同级
```

访问10.0.0.150:8080/docs,返回403的原因

```
# 该目录下有一个context.xml文件
cd /usr/local/apache-tomcat-9.0.88/webapps/docs/META-INF

vim context.xml

<Context antiResourceLocking="false" >
    <Valve className="org.apache.catalina.valves.RemoteAddrValve"
          allow="127\\.\\d+\\.\\d+\\.\\d+|::1|0:0:0:0:0:0:1" />
</Context>

# 这个内容限制了该页面只有本机127网段的ip才能访问，把它注释掉即可
```

访问jsp等动态文件，会在/usr/local/apache-tomcat-9.0.88/work目录下生成servlet代码

```
[root@mystical /usr/local/apache-tomcat-9.0.88] $ tree work/
work/
└── Catalina
    └── localhost
        ├── docs
        ├── examples
        ├── host-manager
        ├── manager
        └── org
```

```

    |   └── apache
    |       └── jsp
    |           └── WEB_002dINF
    |               └── jsp
    |                   ├── _403_jsp.class
    |                   └── _403_jsp.java
    |
    └── ROOT
        └── org
            └── apache
                └── jsp
                    ├── index_jsp.class
                    └── index_jsp.java

```

- 所以如果修改jsp文件后，再次在浏览器访问，如果没有生效，可以尝试将worker下的缓存文件清除，再重新访问，即可生效

## 默认端口和默认主页

```

# 全局默认端口配置，可以修改端口，或增加connector
vim conf/server.xml

<Connector port="8080" protocol="HTTP/1.1"          # 此处改成80无效
    connectionTimeout="20000"                         # 因为tomcat服务器以tomcat身份运行
    redirectPort="8443"                                # 无法使用特权端口
    maxParameterCount="1000"
/>

<Connector port="8000" protocol="HTTP/1.1"          # 增加一个connector，监听8000，重启
    services
    connectionTimeout="20000"
    redirectPort="8443"
    maxParameterCount="1000"
/>

systemctl restart tomcat.servicecd .

```

```

# /conf/web.xml
# 默认全局配置
# 可以在这里更改或添加默认主页
i<>

```

## 在单独项目目录设置默认主页

```

tree webapps/test/
test
├── index.html
├── index.txt
└── WEB-INF          # 创建WEB-INF
    └── web.xml      # 输出web.xml

# 注意xml文件的格式，注意清除隐藏字符
cat web.xml
<?xml version="1.0" encoding="UTF-8"?>

```

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" version="4.0" metadata-
  complete="true">

  <welcome-file-list>
    <welcome-file>index.txt</welcome-file>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
```

## 部署独立域名的WEB服务

```
# 创建文件目录
mkdir -pv /data/webapps/ROOT

# 向文件中添加文件
cp index.{html,jsp} /data/webapps/ROOT/

# 修改权限：只要给jsp文件w权限即可

# 新增域名配置
<Host name="java.fenge.org" appBase="/data/webapps" unpackWARS="true"
  autoDeploy="true">
</Host>

# 重启服务，在客户端配置域名解析，并测试
systemctl restart tomcat.service
```

## 以war格式压缩包部署

```
# 注意权限问题，war里的文件属主属组要是tomcat
# 创建文件

cd /data/webapps/ROOT

# 将ROOT下的文件打成war包
cp jar -cvf ../../test.war *

# 此时test.war和webapps平级目录
# 将test.war移动进webapps
# 自动解包部署
[root@mystical /data/webapps] $ls
ROOT test test.war

[root@mystical /data/webapps] $tree test
test
├── index.html
├── index.jsp
└── META-INF
    ├── MANIFEST.MF
    └── war-tracker
```

## 实验：部署jpress

### 默认应用管理

```
# 在META-INF目录下的context.xml里将10网段加进去
<valve className="org.apache.catalina.valves.RemoteAddrValve"
      allow="127.\d+.\d+.\d+|::1|0:0:0:0:0:0:1|10.\d+.\d+.\d+" />

# 然后在conf/tomcat-users.xml里将用户密码的设置添加进去
<role rolename="manager-gui"/>
<user username="tomcat" password="123456" roles="manager-gui"/>

# role 说明
# manager-gui
# manager-status
# manager-script
# manager-jmx
以HTML 显示 status 页面
仅显示 status 页面
以文本显示 status 页面内容
以 xml 显示页面内容

# 重启服务
systemctl restart tomcat:

# host-manager
<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<user username="tomcat" password="123456" roles="manager-gui,admin-gui"/>
```

## Tomcat的多实例部署

本质是将主程序目录和数据目录分离，使用一个主程序控制多个数据目录，每个数据目录可以是不同的项目

### 目录结构

```
# 主程序目录只有两个目录，一个是主程序bin目录，一个是函数库lib目录
[root@ubuntu2204 ~]#ls /usr/local/tomcat
bin  BUILDING.txt  CONTRIBUTING.md  lib  LICENSE  NOTICE  README.md  RELEASE-
NOTES  RUNNING.txt

# 实例数据目录
[root@ubuntu2204 ~]#ls /www/tomcat1
conf  logs  temp  webapps  work
[root@ubuntu2204 ~]#ls /www/tomcat2
conf  logs  temp  webapps  work

# 使用脚本控制多实例
# 开启多实例
bash start_tomcat.sh /www/tomcat1
bash start_tomcat.sh /www/tomcat2

# 注意修改tomcat数据目录中server.xml中的端口，防止端口冲突
# 比如tomcat1使用默认的8080端口访问（server标签），使用8085（connector标签）控制
```

```
# tomcat2使用默认的8081端口访问，使用8086控制
```

## 控制脚本内容

```
# 确定主程序目录
export CATALINA_HOME=/usr/local/tomcat/
# 确定数据目录地址
export CATALINA_BASE=${1%/}

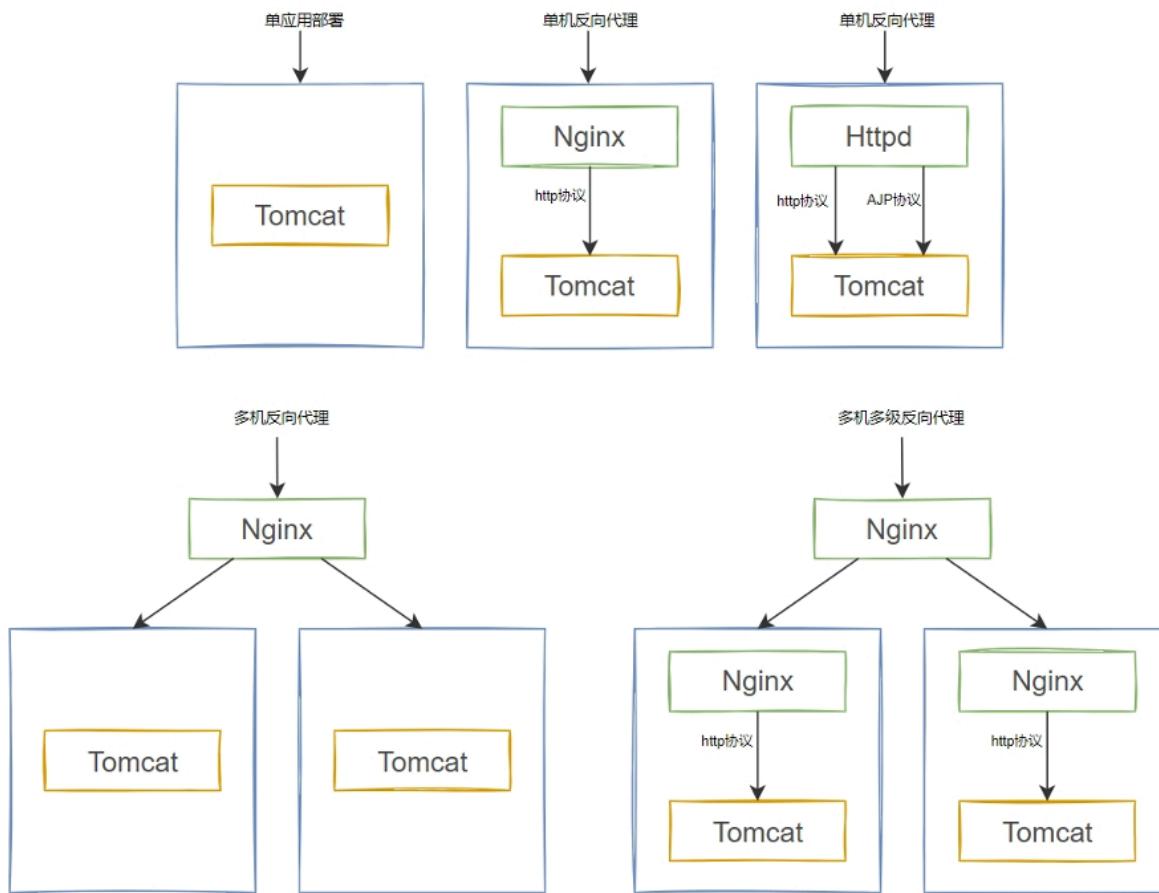
TOMCAT_ID=`ps aux | grep "java"|grep "Dcatalina.base=$CATALINA_BASE "|grep -v "grep"|awk '{ print $2 }'` 

if [ -n "$TOMCAT_ID" ] ; then
echo "tomcat(${TOMCAT_ID}) still running now . please shutdown it first";
exit 2;
fi

# 使用startup.sh开启tomcat，将startup改成shutdown即可实现关闭指定tomcat实例
TOMCAT_START_LOG=`$CATALINA_HOME/bin/startup.sh` 

if [ "$?" = "0" ]; then
    echo "$0 $1 start successed"
else
    echo "$0 ${1%/} start faild"
    echo $TOMCAT_START_LOG
fi
```

# 反向代理中部署Tomcat



## Nginx单机反向代理实现

在同一台服务器上部署Nginx和Tomcat，Nginx监听80端口，将请求转发至后端Tomcat的8080端口进行处理

```
# tomcat配置
cat conf/server.xml

<Host name="java.fenge.org" appBase="/data/webapps" unpackWARs="true"
      autoDeploy="true">
</Host>

systemctl restart tomcat.service

# nginx配置
cat /etc/nginx/conf.d/java.fenge.org.conf

server {
    listen 80;
    server_name java.fenge.org;

    location / {
        proxy_pass http://10.0.0.150:8080;
        proxy_set_header host $http_host;
    }
}
```

```
# 在nginx中配置动静分离

server {
    listen 80;
    server_name java.fenge.org;
    root /data/nginx/html/jhtml;

    location ~* \.jsp$ {
        proxy_pass http://10.0.0.150:8080;
        proxy_set_header host $http_host;
    }
}
```

## Nginx代理多台Tomcat实现

### 实验准备

- Nginx ----- 10.0.0.180
- Tomcat ----- 10.0.0.150
- Tomcat ----- 10.0.0.151

```
# tomcat部署，两台配置一样

<Host name="java.fenge.org" appBase="/data/webapps" unpackWARs="true"
autoDeploy="true">
</Host>

# Tomcat 10.0.0.151 配置
upstream group11 {
    server 10.0.0.151:8080;
    server 10.0.0.150:8080;
}

server {
    listen 80;
    server_name java.fenge.org;
    root /data/nginx/html/jhtml;

    location ~* \.jsp$ {
        proxy_pass http://group11;
        proxy_set_header host $http_host;
    }
}
```

## 实现会话保持

```
# 实现客户端会话绑定
[root@ubuntu ~]# cat /etc/nginx/conf.d/java.m99-magedu.com.conf
upstream tomcat {
    ip_hash;
    hash $remoute_addr;
    hash $cookie_JSESSION consistent;  # 三选一

    server 10.0.0.208:8080;
    server 10.0.0.210:8080;
}
```

## 配置https监听转发到后端http

```
#先使用easy-rsa生成证书文件
# Nginx上的配置
upstream group1 {
    server 10.0.0.118:8080;
    server 10.0.0.128:8080;
}

server {
    server_name java.feng.org;
    root /data/html/java;
    return 302 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name java.feng.org;
    root /data/html/java;

    ssl_certificate /usr/share/easy-rsa/pki/java.feng.org.pem;
    ssl_certificate_key /usr/share/easy-rsa/pki/private/java.feng.org.key;
    ssl_session_cache shared:sslcache:20m;
    ssl_session_timeout 10m;

    location ~* \.jsp$ {
        proxy_pass http://group1;
        proxy_set_header host $http_host;
    }
}
```

## 实现真实客户端IP地址透传

```
# 在Nginx服务器上添加X-Forwarded-For
location ~* \.jsp\$ {
    proxy_pass http://group1;
    proxy_set_header host $http_host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}

# 在tomcat的server.xml上更改日志，查看效果
<valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
       prefix="java.feng.org_access_log" suffix=".txt"
       pattern="%h %l %u %t "%r" %s %b %{x-forwarded-for}i" />
```

## Tomcat Session 复制集群

在上述Nginx代理多机Tomcat的架构中，我们在Nginx代理节点通过调度算法实现会话绑定，将来自同一客户端的请求调度到相同的后端服务器上，在这种情况下，如果后端Tomcat服务不可用，Nginx在检测后会将请求调度到可用的节点，则原来的Session数据还是会丢失。

我们可以使用Tomcat中的session复制功能，实现在多台Tomcat服务器上复制Session的功能，这种配置下，任何一台Tomcat服务器都会有全量的session数据。

```
<!-- 在 server.xml 中指定的 Host 标签内添加下列内容 -->
<cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
channelSendOptions="8">
    <Manager className="org.apache.catalina.ha.session.DeltaManager"
expireSessionsOnShutdown="false" notifyListenersOnReplication="true"/>

    <!-- Channel定义了用于节点间通信的通道 -->
    <Channel className="org.apache.catalina.tribes.group.GroupChannel">
        <!-- 定义集群相关配置，集群中每个节点此处配置相同，才表示是同一个集群，此处配置用来做心跳检测 -->
        <Membership className="org.apache.catalina.tribes.membership.McastService"
address="228.0.0.4" port="45564" frequency="500" dropTime="3000"/>
        <!-- 定义集群节点接收消息的配置 -->
        <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
address="auto" port="4000" autoBind="100" selectorTimeout="5000"
maxThreads="6"/>

        <Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
            <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
        </Sender>
        <Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
        <Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatchInterceptor"/>
    </Channel>
    <valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter="" />
    <valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>
```

```

<Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
tempDir="/tmp/war-temp/" deployDir="/tmp/war-deploy/" watchDir="/tmp/war-listen/"
watchEnabled="false"/>
  <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>
</cluster>

```

- 在项目的web.xml中添加<distributable/>

```
cp -r webapps/ROOT/WEB-INF/ /data/webapps/ROOT/
```

```

cat /data/webapps/ROOT/WEB-INF/web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0" metadata-complete="true">
<display-name>Welcome to Tomcat</display-name>
<description>
  Welcome to Tomcat
</description>
<distributable /> # 添加此行
</web-app>

```

```
# 保证权限
chown -R tomcat.tomcat /data/html/ROOT/WEB-INF/
```

```
systemctl restart tomcat.service
```

```
# 成功
# 中途失败可能
1. 目录权限
2. 配置文件
```

```

# 集群日志总结:
Replication member added:
[org.apache.catalina.tribes.membership.MemberImpl[tcp://{10, 0, 0, 128}:4000,{10,
0, 0, 128},4000, alive=14089, securePort=-1, UDP Port=-1, id={73 9 60 -82 -42 28
78 -98 -114 -73 -89 127 17 117 -42 106 }, payload={}, command={}, domain={}]]
# 这段表面集群成员添加成功，表明一个集群成员（IP 地址为 10.0.0.128 端口 4000）已经成功被添加到集群中

```

```
Done sleeping, membership established, start level:[4]
# 集群建立完成
```

```
Deploying web application directory [/data/html/ROOT]
# 部署web应用，部署到指定的/data/html/ROOT目录
```

```

Manager []: In reply to the 'Get all session data' message sent at [5/5/24, 7:19
PM], a 'No matching context manager' message was received after [118] ms.
# 会话管理警告，得到了所有的数据，但是指出在集群中的某个节点上缺少有效的会话管理器配置。
# 这种情况下，基本可以确定是web.xml所在上下文的问题，要吗是这个文件本身配置写错了，要不就是上下文的权限有问题

```

# Memcached 实现Session共享

## Memcached基础

Memcached(Memory Cache Daemon)是一种开源的，高性能的分布式内存对象缓存系统，主要用于减轻数据库负载，提高动态Web应用的性能。它最初由Brad Fitzpatrick开发，现在由一个开源社区维护。

### 以下是Memcached的一些关键特性

1. 分布式缓存：Memcached可以在多台服务器上运行，形成一个分布式缓存系统，这意味着它可以横向扩展以处理大量的请求和数据存储需求。
2. 内存存储：Memcached将数据存储在内存中，这使得它能够提供快速的读写操作。内存存储也是其高性能的主要原因之一。
3. 简单的键值存储：Memcached是一个键值对存储系统，每个键对应一个值，这使得它非常适合存储简单的数据结构，如：对象，字符串等。
4. 缓存有效期：可以为存储在Memcached中的数据设置生存时间，一旦超过该时间，数据被自动删除。这有助于确保缓存中的数据不会过期，同时减少了需要手动清理缓存的工作。
5. 高性能：由于数据存储在内存中，Memcached提供了非常快速的读写操作。这使得它成为处理高并发，大规模应用的理想选择。
6. 分布式哈希表：Memcached使用一致性哈希算法来确定数据存储在哪个节点，这确保了在添加和删除节点时，数据的迁移量被最小化，同时保持了负载均衡。
7. 支持多种编程语言

### 以下是Memcached的限制

1. 由于基于内存存储，存储容量受到物理内存的限制。
2. 不具备持久性存储能力，数据一般在服务重启后会丢失。

## Memcached和Redis的比较

相同点：

- 内存存储
- 键值存储
- 数据结构支持

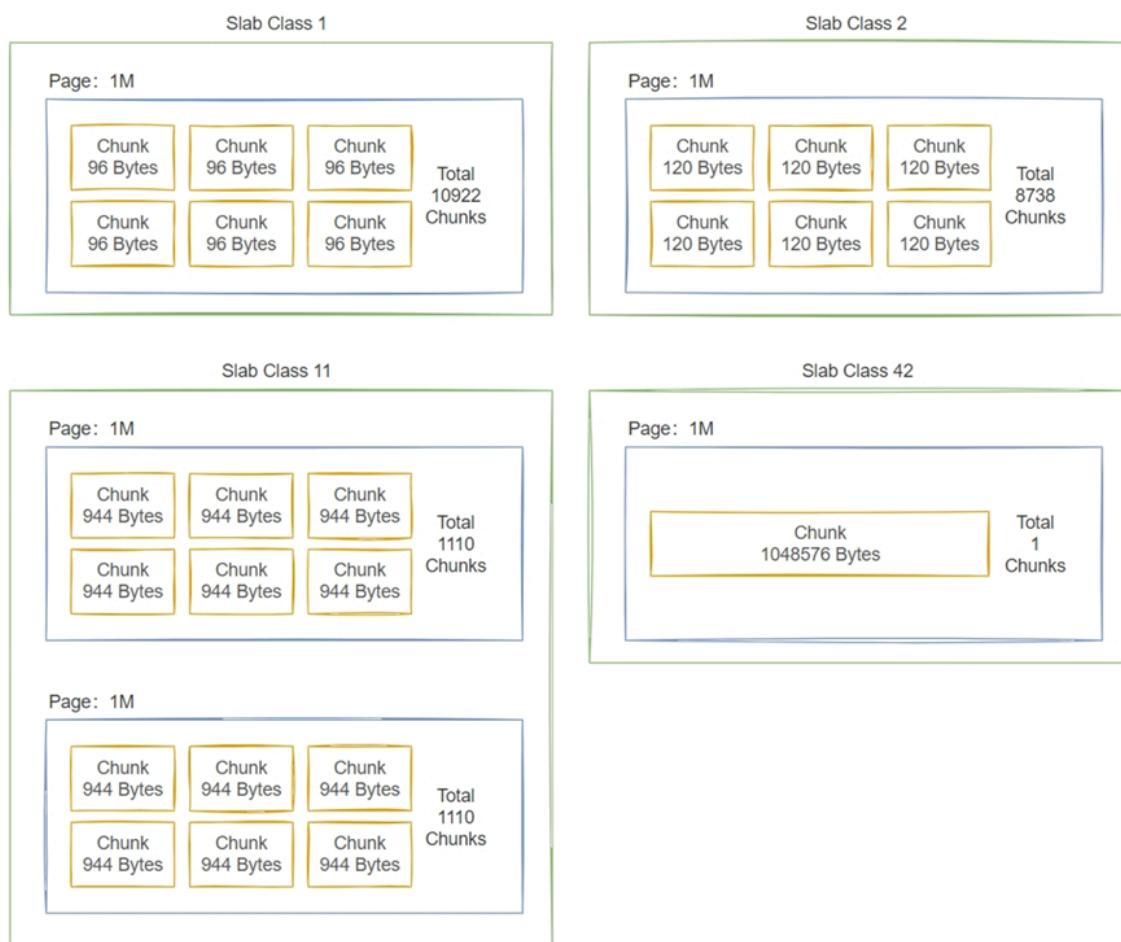
不同点：

- 数据持久化
- 功能：Redis提供了许多附加功能，例如发布/订阅、事务、Lua脚本等，使得它不仅仅是一个缓存系统，还可以用作消息队列，计数器等。而Memcached更专注于作为分布式缓存的角色，功能相对简单。
- 数据类型处理：Redis能处理更复杂的数据结构
- 性能：由于Redis提供了更多的功能和数据类型支持，因此在某方面可能略逊于Memcached。然而，具体性能取决于使用场景，数据访问模式以及部署配置等因素。

## Memcached 工作机制

Memcached采用Slab Allocator机制来分配、管理内存

- Page：分配给Slab的内存空间，默认为1MB，分配后就得到一个Slab，Slab分配之后内存安装固定字节大小等分成chunk
- Chunk：用于缓存记录k/v值的内存空间，Memcached会根据数据大小选择存到哪一个chunk中，假设 chunk 有 128bytes、64bytes等多种，数据只有 100bytes 存储在 128bytes 中，存在少许浪费。chunk 增长因子 default: 1.25，Chunk 最大就是一个 Page 大小，所以 Memcached 无法存储单个 K/V 超过 1M 的数据
- Slab Class：Slab 按照 Chunk 的大小分组，就组成不同的 Slab Class，第一个 Chunk 大小为 96B 的 Slab 为 Class1，Chunk 120B 为 Class 2，如果有 100bytes 要存，那么 Memcached 会选择下图中Slab Class 2 存储，因为它是120 bytes 的 Chunk，Slab 之间的差异可以使用 Growth Factor 控制，默认 1.25



- 可控参数：Page大小（默认1M），chunk的成长因子（默认1.25）

懒过期

memcached不会监视数据是否过期，而是在取数据时才看是否过期，如果过期，把数据有效期表示为0，并不清楚该数据，以后可以覆盖该位置存储其他数据

LRU（最近最少使用算法）

集群

Memcached集群，成为基于客户端的分布式集群，即由客户端实现集群功能，即Memcached本身不支持集群，Memcached集群内部并不相互通信，一切都需要客户端连接到Memcached服务器后自行组织这些节点，并确定数据存储的点

## 为什么Memcached是基于客户端的分布式集群

1. 无中心协调：Memcached服务器本身不维护关于其他服务器状态或数据存放位置的信息，它们独立运行，不需要彼此通信来处理缓存数据
2. 客户端决策：数据在哪里存储（哪台Memcached服务器）由客户端决定。客户端使用一致性hash等算法选择数据应该存储在哪个服务器上，这种方法减少了单点故障的风险，因为没有一个中央节点是关键的
3. 简单性和性能：这种模式使Memcached非常简单和高效。服务器仅仅响应客户端的请求，所有智能决策都由客户端完成。这减轻了服务器的计算负担，使系统能更快地响应。

## 安装和启动

```
# 下载安装memcached
apt update; apt install memcached -y

# 查看memcached的状态
systemctl status memcached.service

# 启动脚本和配置文件
[root@ubuntu ~]# systemctl cat memcached.service | grep Exec
ExecStart=/usr/share/memcached/scripts/systemd-memcached-wrapper
/etc/memcached.conf

# 注释掉-1，使其对外监听所有设备
#-1 127.0.0.1

[root@mystical ~] $ss -tpln
State      Recv-Q      Send-Q      Local Address:Port          Peer Address:Port      Process
LISTEN      0            128           0.0.0.0:22              0.0.0.0:*               users:
(("sshd",pid=916,fd=3))
LISTEN      0            1024          0.0.0.0:11211           0.0.0.0:*               users:
(("memcached",pid=3124,fd=22))

# 使用nc可以连接memcached进行测试

nc 10.0.0.181 11211
>> stats      # 输入stats
STAT pid 3124
STAT uptime 126
STAT time 1714996697
STAT version 1.6.14
STAT libevent 2.1.12-stable
STAT pointer_size 64
STAT rusage_user 0.024244
STAT rusage_system 0.036366
STAT max_connections 1024
STAT curr_connections 2
STAT total_connections 3
STAT rejected_connections 0
STAT connection_structures 3
...
# 指定增长因子，每个chunk2倍大小增长
```

```
memcached -p 11211 -f 2 -u memcache -vv
# -f <num> 指定增长因子
# -p 指定端口
# -u 指定运行用户
# -P 指定pid文件路径
```

## Memcached常用操作命令

在memcached中，根据功能不同可以将命令划分为存储、检索、删除、增加/减少、统计、清空等六个部分

```
# 命令格式
<command name> <key> <flag> <exptime> <bytes> [noreply]\r\n<data block>\r\n

# <command name>          具体命令
# <key>                   任意字符串，用于存取数据的标识
# <flag>                  任意一个无符号整型数学，该字段对服务器不透明，客户端可将其作为位域来存储数据特定信息
# <exptime>                缓存有效期，以秒为单位
# <bytes>                 此次写入数据占用的空间大小
# [noreply]                可选，表示服务器不用返回任何数据

# 常用命令
# stats                    显示当前memcached服务状态
# set                      设置一个key/value
# add                      key不存在时，新增一个key/value
# replace                  替换一个已存在的key中的内容
# append                   追加数据到key的尾部
# prepend                  插入数据到key的头部
# get                      获取key的flag及value
# delete                  删除一个key
# incr                     对key做自增1，key保存的必须是整数
# decr                     对key做自减1，key保存的必须是整数
# flush_all                清空所有数据

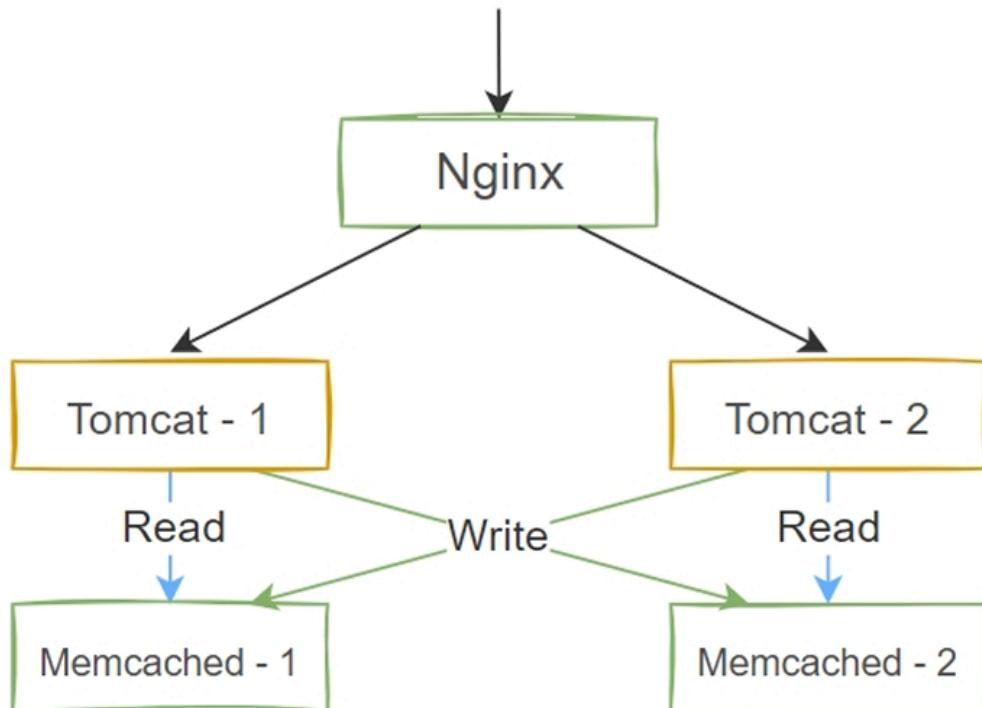
# memcached的使用示例
[root@ubuntu ~]# telnet 10.0.0.206 11211
Trying 10.0.0.206...
Connected to 10.0.0.206.
Escape character is '^].
set test 1 30 3          # 设置一个 key/value , key=test, flag=1, exp=30S,
bytes=3
abc                      # 此处只能存3字节，不能多也不能少
STORED
get test                  # 获取数据
VALUE test 1 3
abc
END
get test                  # 30S 后数据消失
END
```

## MSM介绍和安装

MSM(msmcached session manager) 提供将Tomcat的session保持到memcached或Redis的程序，可以实现高可用（该项目看在github上）

### 基于Sticky模式实现Session共享集群

sticky模式即前端tomcat和后端memcached有关联（粘性）关系



```
# nginx节点测试 Memcached远程访问

# 安装python包管理器pip
apt install -y python3-pip

# 安装python客户端工具
pip install python-memcached

# 写一个memcache测试脚本
vim test.py

#!/usr/bin/python3

import memcache
client = memcache.Client(['10.0.0.150:11211', '10.0.0.151:11211'], debug=True)

for i in client.get_stats('items'):
    print(i)

print('-' * 35)

for i in client.get_stats('cachedump 5 0'):
    print(i)

# 执行python脚本
```

```
[root@mystical ~] $python3 test.py
('10.0.0.150:11211 (1)', {})
('10.0.0.151:11211 (1)', {})

-----
('10.0.0.150:11211 (1)', {})
('10.0.0.151:11211 (1)', {})

# 显示无任何数据

# 配置（150的session写到151的memcached； 151的session写到150的memcached）
# tomcat节点配置session共享

# 10.0.0.150

# 下载相关jar包
cd /usr/local/tomcat/lib

wget https://repo1.maven.org/maven2/org/ow2/asm/asm/5.2/asm-5.2.jar
wget https://repo1.maven.org/maven2/com/esotericsoftware/kryo/3.0.3/kryo-
3.0.3.jar
wget https://repo1.maven.org/maven2/de/javakaffee/kryo-serializers/0.45/kryo-
serializers-0.45.jar
wget https://repo1.maven.org/maven2/de/javakaffee/msm/memcached-session-
manager/2.3.2/memcached-session-manager-2.3.2.jar
wget https://repo1.maven.org/maven2/de/javakaffee/msm/memcached-session-manager-
tc9/2.3.2/memcached-session-manager-tc9-2.3.2.jar
wget https://repo1.maven.org/maven2/com/esotericsoftware/minlog/1.3.1/minlog-
1.3.1.jar
wget https://repo1.maven.org/maven2/de/javakaffee/msm/msm-kryo-
serializer/2.3.2/msm-kryo-serializer-2.3.2.jar
wget https://repo1.maven.org/maven2/org/objenesis/objenesis/2.6/objenesis-2.6.jar
wget https://repo1.maven.org/maven2/com/esotericsoftware/reflectasm/1.11.9/reflectasm-
1.11.9.jar
wget https://repo1.maven.org/maven2/net/spy/spymemcached/2.12.3/spymemcached-
2.12.3.jar

# 修改配置
cat /usr/local/tomcat/conf/context.xml

# 添加下列内容
# 10.0.0.150
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="m1:10.0.0.150:11211,m2:10.0.0.151:11211" failoverNodes="m1"
requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFacto
ry"/>

# 10.0.0.151
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="m1:10.0.0.150:11211,m2:10.0.0.151:11211" failoverNodes="m2"
requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFacto
ry"/>

# 150和151都重启tomcat服务
```

```
systemctl restart tomcat
```

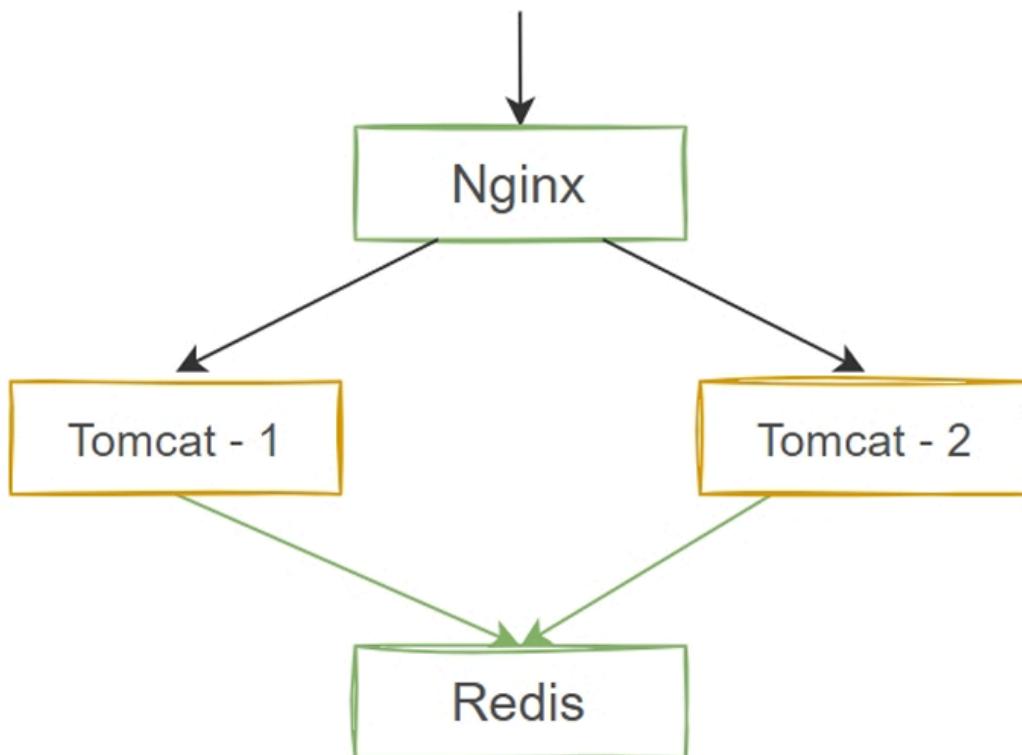
## 基于非Sticky模式实现Session共享集群

```
# 将之前的数据改为
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="m1:10.0.0.150:11211,m2:10.0.0.151:11211"
sticky="false"
sessionBackupAsync="false"
lockingMode="uriPattern:/path1|/path2"
requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFacto
ry" />
```

## Redis实现Session共享

### 基于MSM实现Session共享



```
# 安装redis，并开发端口
apt install -y redis-server

# /etc/redis/redis.conf 里面的bind 127.0.0.1:6379注释掉

# 重启redis

# 在/usr/local/tomcat/lib/下载jar包
wget https://repo1.maven.org/maven2/redis/clients/jedis/3.0.0/jedis-3.0.0.jar

# 在context.xml中添加
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
```

```
memcachedNodes="redis://10.0.0.181"
sticky="false"
sessionBackupAsync="false"
lockingMode="uriPattern:/path1|/path2"
requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"

transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFacto
ry" />
```

## Tomcat性能优化

### CG垃圾收集器

当需要分配的内存空间不再使用的时候，JVM将调用垃圾回收机制来回收内存空间

在堆内存中如果创建的对象不再使用,仍占用着内存,此时即为垃圾.需要即使进行垃圾回收,从而释放内存空间给其它对象使用

其实不同的开发语言都有垃圾回收问题,C,C++需要程序员人为回收,造成开发难度大,容易出错等问题,但执行效率高,而JAVA和Python中不需要程序员进行人为的回收垃圾,而由JVM或相关程序自动回收垃圾,减轻程序员的开发难度,但可能会造成执行效率低下

堆内存里面经常创建、销毁对象，内存也是被使用、被释放。如果不妥善处理，一个使用频繁的进程，可能会出现虽然有足够的内存容量，但是无法分配出可用内存空间，因为没有连续成片的内存了，内存全是碎片化的空间

所以需要有合适的垃圾回收机制,确保正常释放不再使用的内存空间,还需要保证内存空间尽可能的保持一定的连续

### Garbage垃圾确定方法

- 引用计数：每一个堆内对象上都与一个私有引用计数器，记录着被引用的次数，引用计数清零，该对象所占用堆内存就可以被回收。循环引用的对象都无法将引用计数器清零，就无法清楚。
- 根搜索(可达)算法：Root Searching

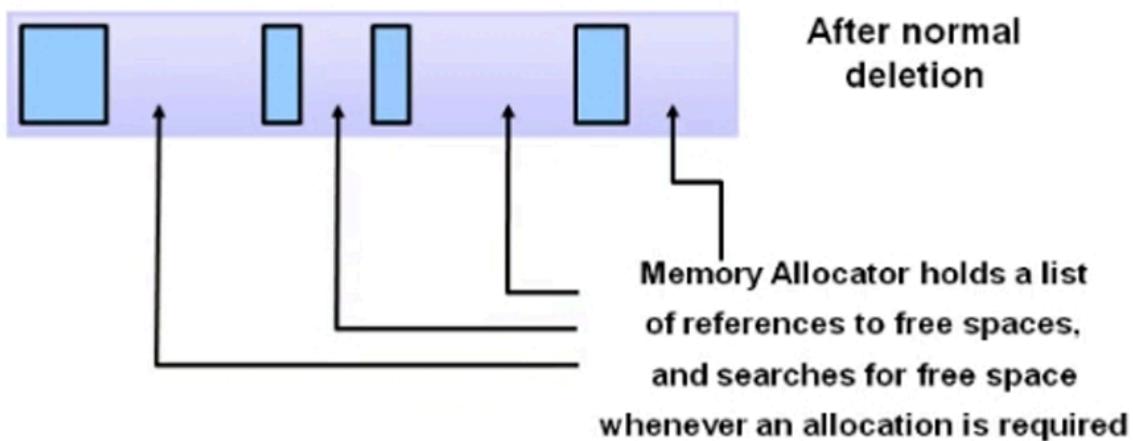
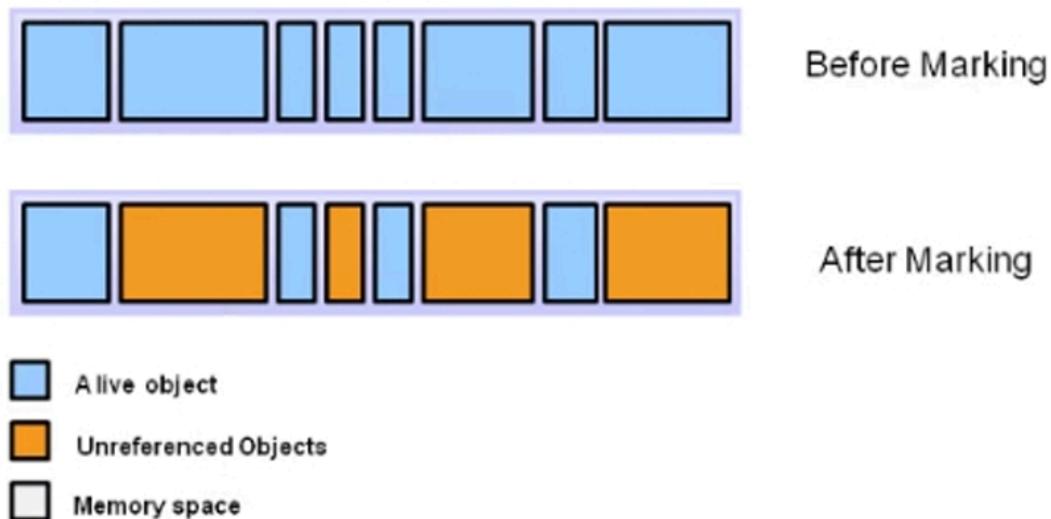
## 垃圾回收基本算法

### 标记-清除Mark-Sweep

分垃圾标记阶段和内存释放两个阶段

- 标记阶段，找到所有可访问对象打个标记，清理阶段，遍历整个堆
- 对未标记对象（即不再使用的对象）逐一进行清理

特点：算法简单，不会浪费内存空间，效率较高，但会形成内存碎片

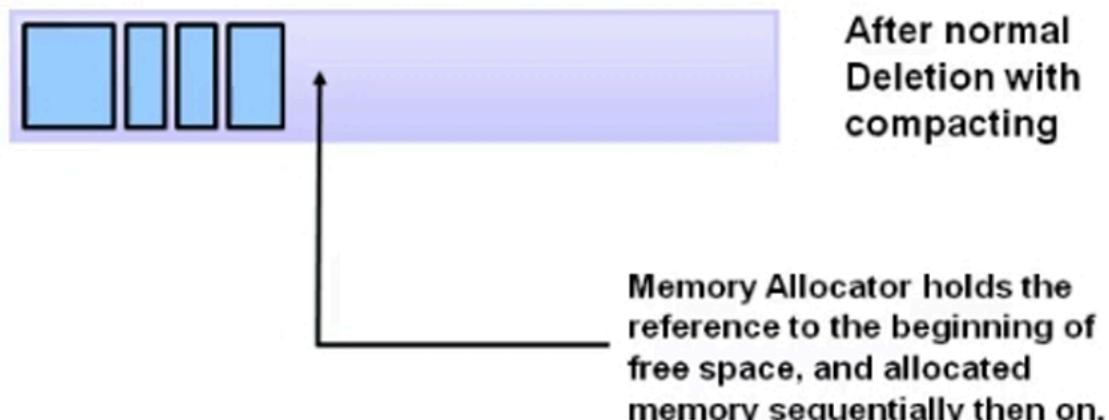


### 标记-压缩（压实）Mark-Compact

分垃圾标记阶段

- 标记阶段，找到所有可访问对象打个标记
- 内存清理阶段时，整理时将对象内存一端移动，整理后存活对象连续的集中在内存一端

特点：整理后的内存空间是连续的，有大段的连续内存可分配，没有内存碎片，缺点是内存整理过程有消耗，效率相对低下



### 复制Copying

先将可用内存分为大小相同两块区域A和B，每次只用其中一块，比如A。当A用完后，则将A中存活的对象复制到B。复制到B的时候连续的使用内存，最后将A一次性清除干净

特点：好处是没有碎片，复制过程中保证对象使用连续空间，且一次性清除所有垃圾，所以即使对象很多，收回效率也很高，缺点是比较浪费内存，只能使用原来一般内存，因为内存对半划分

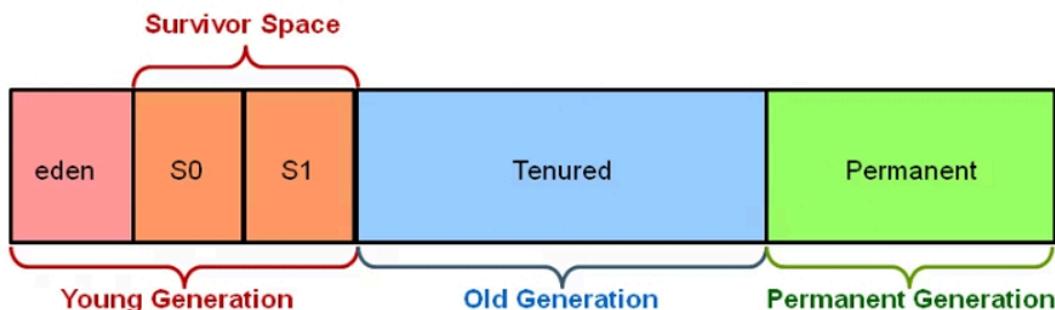
### 三种算法的特点与比较

- 效率：复制算法>标记清除算法> 标记压缩算法
- 内存整齐度：复制算法=标记压缩算法> 标记清除算法
- 内存利用率：标记压缩算法=标记清除算法>复制算法

## 分代堆内存CG策略

堆内存分代: 将heap内存空间分为三个不同类别，年轻代，老年代，持久代

**Hotspot Heap Structure**



### 年轻代Young: Young Generation

伊甸园eden: 只有一个，刚刚创建的对象

幸存(存活区)Survivor Space: 有2个幸存区，一个是from区，一个是to区。大小相等，地位相同、可互换

### 老年代Tenured: Old Generation

长时间存活对象

### 永久代Tenured: Old Generation

JDK1.7之前使用，即Method Area方法区，保存 JVM 自身的类和方法，存储 JAVA 运行时的环境信息，JDK1.8 后改名为 MetaSpace，此空间不存在垃圾回收，关闭JVM会释放此区域内存，此空间物理上不属于heap内存，但逻辑上存在于heap内存

### 默认空间大小比例：

默认JVM试图分配最大内存的总内存的1/4, 初始化默认总内存为总内存的1/64, 年青代中heap的1/3, 老年代占2/3



规律：一般情况99%的对象都是临时对象

# Tomcat监控与调优

## status

status监控步骤

### 1. 配置用户及角色

```
#更改访问权限
#/tomcat/webapps/manager/META-INF/context.xml
#/tomcat/webapps/host-manager/META-INF/context.xml
# 在META-INF目录下的context.xml里将10网段加进去
<valve className="org.apache.catalina.valves.RemoteAddrValve"
      allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1|10\.\d+\.\d+\.\d+" />

# 然后在conf/tomcat-users.xml里将用户密码的设置添加进去
<role rolename="manager-gui"/>
<user username="tomcat" password="123456" roles="manager-gui"/>

# role 说明
# manager-gui
# manager-status
# manager-script
# manager-jmx
以HTML 显示 status 页面
仅显示 status 页面
以文本显示 status 页面内容
以 xml 显示页面内容

# 重启服务
systemctl restart tomcat:

# host-manager
<role rolename="manager-gui"/>
<role rolename="admin-gui"/>
<user username="tomcat" password="123456" roles="manager-gui,admin-gui"/>
```

- 这里角色role相当于权限的概念
  - admin-gui角色
  - manager-gui角色

设置角色后就可以进入监控页

The screenshot shows the Apache Tomcat Manager Status page. At the top, there's a navigation bar with tabs like '管理器' (Manager), '应用程序列表' (Application List), 'HTML管理器帮助' (HTML Manager Help), '管理者帮助' (Manager Help), and '完整的服务器状态' (Full Server Status). Below the navigation is a section titled '服务器信息' (Server Information) with a table showing Tomcat version (Apache Tomcat/9.0.89), JVM version (11.0.23+7-LTS-222), JVM provider (Oracle Corporation), OS name (Linux), OS version (5.15.0-107-generic), OS architecture (amd64), host name (ubuntu2204.wang.org), and IP address (205.178.189.129). The main content area is titled 'JVM' and shows memory usage details. Below that is a section titled '"http-nio-8080"' displaying network statistics.

## probe探针

1. 配置用户及角色
2. 配置好角色 (manager-gui角色) 后重启服务器
3. 进入监控页面

```
# 在GitHub上下载probe.war部署在服务器上
# PsiProbe 的 web.xml 文件可能在 WEB-INF 目录下。查找是否有 <security-constraint> 配置
强制 HTTPS。,如果有, 将其更改成NONE
如果存在 <transport-guarantee>CONFIDENTIAL</transport-guarantee>, 需要修改为 NONE 或者删除相关配置。
```

修改 server.xml

删除或设置 `redirectPort="0"`

## application

## 查看资源请求情况详情

Version 4.1.4 running on ubuntu2204, UP for 0 days 2 hours 1 minutes

Installed applications

请求										session		sessiontimeout		调用的JSP文件						
Application statistics			REQ.		SESS.		S.ATTR		C.ATTR		SESS.TIMEOUT		JSP		JBDC USAGE		CLSTRED.?		SER.?	
/	running	Welcome to Tomcat	0	0	0	0	0	9	30	30	15	15	no	yes	no	yes	no	yes		
/examples	running	Servlet and JSP Examples	0	0	0	0	0	10	30	30	15	15	no	yes	no	yes	no	yes		
/host-manager	running	Tomcat Host Manager Application	0	0	0	0	0	10	30	30	15	15	no	yes	no	yes	no	yes		
/manager	running	Tomcat Manager Application	0	0	0	0	0	10	30	30	15	15	no	yes	no	yes	no	yes		
/docs	running	Tomcat Documentation	0	0	0	0	0	9	30	30	15	15	no	yes	no	yes	no	yes		
/probe	running	PSI Probe for Apache Tomcat v4.1.4	948	62	246	16	0	9	30	30	15	15	no	yes	no	yes	no	yes		

## 通过request插看

Version 4.1.4 running on ubuntu2204, UP for 0 days 1 hours 57 minutes

Servlets 资源路径										请求数 (可以看作点击率)							
APP	NAME	AVAIL	STARTUP	LOAD TIME	REQ	PROC TIME	ERR	MIN TIME	MAX TIME	MULT THRD							
/probe	probe	yes	0	235	822	0:00:29.569	0	2	1464	yes							
/probe	ptprobe.ProbeServlet	yes	1	9	50	0:00:00.582	0	1	87	yes							
/probe	default	org.apache.catalina.servlets.DefaultServlet	yes	3	1	0:00:06.598	1	6598	6598	yes							
/	jsp	org.apache.jasper.servlet.JspServlet	yes	1	0	0:00:00.000	0	0	0	yes							
/	default	org.apache.catalina.servlets.DefaultServlet	yes	3	1	0:00:00.000	0	0	0	yes							
/examples	jsp	org.apache.jasper.servlet.JspServlet	yes	3	1	0:00:00.000	0	0	0	yes							
/examples	RequestInfoExample	yes	no	0	0	0:00:00.000	0	0	0	yes							
/examples	RequestInfoExample	yes	no	0	0	0:00:00.000	0	0	0	yes							
/examples	async3	async.Async3	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	numberwriter	nonblocking.NumberWriter	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	async2	async.Async2	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	async1	async.Async1	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	jsp	org.apache.jasper.servlet.JspServlet	yes	3	3	0:00:00.000	0	0	0	yes							
/examples	async0	async.Async0	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	bytecounter	nonblocking.ByteCounter	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	responsetrailer	trailers.ResponseTrailers	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	ServletToJsp	ServletToJsp	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	RequestParamExample	RequestParamExample	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	RequestHeaderExample	RequestHeaderExample	yes	no	0	0:00:00.000	0	0	0	yes							
/examples	default	org.apache.catalina.servlets.DefaultServlet	yes	1	11	0:00:00.000	0	0	0	yes							
/examples	CookieExample	CookieExample	yes	no	0	0:00:00.000	0	0	0	yes							

## Session

Toggle Expire Session search

SESSIONS 点击具体的资源的session的数值即可进入，查看具体的session信息，人员，地域，访问时间，IP等

96 items found, displaying 1 to 50. [First/Prev] 1, 2 [Next/Last]

SESSION ID	LAST IP	IDLE TIME	AGE	EXPIRY TIME	OBJECT COUNT	SER.
□ F2739804723E8EBC5CAEA88CF034C16-m2	10.0.0.1	0:05:21.742	0:05:21.745	Wed Dec 11 23:54:09 CST 2024	3	yes
□ 73F38BCD94E9E5F429FC9919E6B8EF6-m2	10.0.0.1	0:05:32.462	0:05:32.464	Wed Dec 11 23:53:58 CST 2024	3	yes
□ FCB892E975780C28E3A8551080117230-m2	10.0.0.1	0:08:35.046	0:08:35.055	Wed Dec 11 23:50:55 CST 2024	3	yes
□ 363A19FDDEBA736258BEF5C2D3ED3EEB-m2	10.0.0.1	0:09:26.322	0:09:26.336	Wed Dec 11 23:50:04 CST 2024	3	yes
□ FFF1397BB8861502479E6819164C13AD4-m2	10.0.0.1	0:07:54.744	0:07:54.755	Wed Dec 11 23:51:36 CST 2024	3	yes
□ 9395685CD03F7E669667E1B45AAEAF-m2	10.0.0.1	0:05:33.167	0:05:33.174	Wed Dec 11 23:53:57 CST 2024	3	yes
□ 58927E451CD67DC88F0DA69951AD3D2-m2	10.0.0.1	0:00:23.858	0:00:23.867	Wed Dec 11 23:59:06 CST 2024	3	yes
□ 3AE17A99CB0868B7D918F4B7D81C724-m2	10.0.0.1	0:05:33.106	0:05:33.109	Wed Dec 11 23:53:57:CST 2024	3	yes
□ 3B0CFO00EE5C534DE6C9D1706A6E9601B-m2	10.0.0.1	0:07:00.856	0:07:00.876	Wed Dec 11 23:52:29 CST 2024	3	yes
□ F6F6A0E12E6DFCE6A6E8A6E6F30ED3AC-m2	10.0.0.1	0:07:25.878	0:07:25.882	Wed Dec 11 23:52:04 CST 2024	3	yes
□ C092746D94915B60D83D791303A0D9B55-m2	10.0.0.1	0:06:30.867	0:06:30.879	Wed Dec 11 23:52:59 CST 2024	3	yes
□ 003C5B228A0CEDB6D4CE02795D3064C8-m2	10.0.0.1	0:06:00.992	0:06:00.996	Wed Dec 11 23:53:29 CST 2024	3	yes
□ 337AD093DB921B5C2A6291EFPB861460-m2	10.0.0.1	0:05:46.783	0:05:48.825	Wed Dec 11 23:53:42 CST 2024	3	yes
□ 235384B4244671FB41B2AA72BF70076-m2	10.0.0.1	0:02:24.923	0:02:24.935	Wed Dec 11 23:57:05 CST 2024	3	yes
□ 34F080D4AB13E6A7332E88B57532A37-m2	10.0.0.1	0:09:26.318	0:09:26.342	Wed Dec 11 23:50:04 CST 2024	3	yes
□ C6B5ABC3AFB6568C5B2E8B6D66669E0-m2	10.0.0.1	0:06:24.849	0:06:24.863	Wed Dec 11 23:53:05 CST 2024	3	yes
□ 7FF2FC4F4109138933E90390B89C19C-m2	10.0.0.1	0:05:45.471	0:05:34.580	Wed Dec 11 23:53:56 CST 2024	3	yes
□ 195BFDCDBFB0DBAE15A3019D47B92250-m2	10.0.0.1	0:08:09.844	0:08:09.868	Wed Dec 11 23:51:20 CST 2024	3	yes
□ 19ED022D126AFCDOCA25A069C9F86-m2	10.0.0.1	0:08:24.986	0:08:24.991	Wed Dec 11 23:51:05 CST 2024	3	yes
□ FA02B4B1581A1778970BCC5518C8E67-m2	10.0.0.1	0:04:42.991	0:04:25.009	Wed Dec 11 23:55:05 CST 2024	3	yes
□ C1482975A4BD11B7407EBAD43DBF4B80-m2	10.0.0.1	0:08:30.019	0:08:30.028	Wed Dec 11 23:51:00 CST 2024	3	yes
□ 217F83E75498EB2C8E96233487403C0-m2	10.0.0.1	0:05:34.527	0:05:34.527	Wed Dec 11 23:53:56 CST 2024	3	yes
□ 65883A9942B7A11807BAB863EDE05A55-m2	10.0.0.1	0:05:27.836	0:05:27.838	Wed Dec 11 23:54:02 CST 2024	3	yes
□ 1106309098700A19F3046F2F6F7D75-m2	10.0.0.1	0:05:55.767	0:05:55.769	Wed Dec 11 23:53:35 CST 2024	3	yes
□ E3E7D6E7813E8B53C281913E2D2A5-m2	10.0.0.1	0:05:32.412	0:05:32.416	Wed Dec 11 23:53:56 CST 2024	3	yes
□ E05BF2227B36E6C41BE60544994D98-m2	10.0.0.1	0:08:50.175	0:08:50.191	Wed Dec 11 23:50:40 CST 2024	3	yes
□ 9BF34346FBC81341BF990087715E151D-m2	10.0.0.1	0:05:39.394	0:05:39.394	Wed Dec 11 23:53:51 CST 2024	3	yes
□ 1C7E55036CF514DF4EBC6A34F87930F6-m2	10.0.0.1	0:10:55.882	0:10:55.891	Wed Dec 11 23:48:34 CST 2024	3	yes
□ EC97E8C49F8AC31F7E23C19C19B8A3-m2	10.0.0.1	0:07:42.851	0:07:42.871	Wed Dec 11 23:51:47 CST 2024	3	yes
□ F51F40C05E1D19586B806128C9F9582B-m2	10.0.0.1	0:05:32.393	0:05:32.411	Wed Dec 11 23:53:58 CST 2024	3	yes
□ BB87AE1A70A70A807CAA86C728320385-m2	10.0.0.1	0:05:21.759	0:05:21.759	Wed Dec 11 23:54:09 CST 2024	3	yes

## 线程池数据 (连接数)

- 根据下列指标即可看出线程池是否够用，是否需要增加线程池的线程数量

Version 4.1.4 running on ubuntu2204.wang.org. UP for 0 days 2 hours 20 minutes

Threads Pools

ID	NAME	EXEC. POINT	STATE	IN_NATIVE	SUSP.	WC	BC
1	main	java.net.PlainSocketImpl.socketAccept ( native code )	RUNNABLE	true	false	1	0
12	AsyncFileHandlerWriter-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	29	0
15	logback-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	108	0
17	Probe_Quartz-1	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16219	270
18	Probe_Quartz-2	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16223	271
19	Probe_Quartz-3	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16218	288
2	Reference Handler	java.lang.ref.Reference.waitForReferencePendingList ( native code )	RUNNABLE	false	false	0	4
20	Probe_Quartz-4	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16212	283
21	Probe_Quartz-5	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16206	271
22	ProbeScheduler_QuartzScheduler Thread	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	874	638
23	Log4j2-TF-5-Scheduled-1	jdk.internal.misc.Unsafe.park ( native code )	TIMED_WAITING	false	false	270	0
24	mysql-cj-abandoned-connection-clean-up	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	1619	2
25	HikariPool-1 housekeeper	jdk.internal.misc.Unsafe.park ( native code )	TIMED_WAITING	false	false	273	0
27	TokenManager	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	55	1
28	ForkJoinPool.commonPool-worker-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	90	0
29	jboot-scheduler-thread-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	107	0
3	Finalizer	java.lang.Object.wait ( native code )	WAITING	false	false	67	66
30	jboot-scheduler-thread-2	jdk.internal.misc.Unsafe.park ( native code )	TIMED_WAITING	false	false	117	0
31	jboot-scheduler-thread-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	136	0
32	jboot-scheduler-thread-4	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	98	0
33	jboot-scheduler-thread-5	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	82	0
34	jboot-scheduler-thread-6	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	117	0
35	jboot-scheduler-thread-7	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	216	0
36	jboot-scheduler-thread-8	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	362	0
37	ContainerBackgroundProcessor[StandardEngine[Catalina]]	java.lang.Thread.sleep ( native code )	TIMED_WAITING	false	false	809	0
38	http-nio-8082-exec-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	69
39	http-nio-8082-exec-2	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	20
40	http-nio-8082-exec-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	63
41	http-nio-8082-exec-4	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	74	55
42	http-nio-8082-exec-5	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	75	3
43	http-nio-8082-exec-6	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	76	6
44	http-nio-8082-exec-7	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	74	28
45	http-nio-8082-exec-8	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	76	1
46	http-nio-8082-exec-9	sun.management.ThreadImpl.getThreadInfo1 ( native code )	RUNNABLE	false	false	76	7
47	http-nio-8082-exec-10	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	66
48	http-nio-8082-Poller	sun.nio.ch.PollSet.poll ( native code )	RUNNABLE	true	false	0	181
49	http-nio-8082-Acceptor	sun.nio.ch.ServerSocketChannelImpl.accept0 ( native code )	RUNNABLE	true	false	0	0
50	http-nio-8082-AsyncTimeout	java.lang.Thread.sleep ( native code )	TIMED_WAITING	false	false	8083	0
52	Java2D Disposer	java.lang.Object.wait ( native code )	WAITING	false	false	15	14
54	logback-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	47	0
55	jboot-scheduler-thread-9	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	91	0

△ 不安全 10.0.202.8082/manager/status



服务器状态

管理器		HTML 管理器帮助		管理者帮助		完整的服务	
<b>服务器信息</b>							
Tomcat 版本	JVM 版本	JVM 提供商	OS 名称	操作系统版本	操作系統架构	主机名	IP 地址
Apache Tomcat/8.5.100	11.0.23+7-LTS-222	Oracle Corporation	Linux	5.15.0-107-generic	amd64	ubuntu2204.wang.org	205.178.189.12

DVM

剩余内存: 78.71 MB 峰值内存: 150.00 MB 最大内存 484.00 MB

内存池	类型	初始化	总共	最大值	已用
G1 Eden Space	Heap memory	8.00 MiB	61.00 MiB	-0.00 MiB	4.00 MiB
G1 Old Gen	Heap memory	24.00 MiB	85.00 MiB	484.00 MiB	61.87 MiB (12%)
G1 Survivor Space	Heap memory	0.00 MiB	4.00 MiB	-0.00 MiB	3.67 MiB
CodeHeap 'non-nmethods'	Non-heap memory	2.43 MiB	2.43 MiB	5.55 MiB	1.33 MiB (23%)
CodeHeap 'non-profiled nmethods'	Non-heap memory	2.43 MiB	8.93 MiB	117.22 MiB	8.89 MiB (7%)
CodeHeap 'profiled nmethods'	Non-heap memory	2.43 MiB	33.93 MiB	117.21 MiB	33.87 MiB (28%)
Compressed Class Space	Non-heap memory	0.00 MiB	21.29 MiB	1024.00 MiB	20.55 MiB (2%)
Metaspace	Non-heap memory	0.00 MiB	149.18 MiB	-0.00 MiB	146.56 MiB

"http-nio-8082"

最大线程: 200 当前线程数: 10 当前线程繁忙: 1 存活套接字总数: 1  
最大处理时间: 1340 ms 处理时间: 11.036 s 请求数: 589 错误数: 18 收到字节: 0.05 MiB 发送字节: 4.18 MiB

阶段	时间	发送字节:	接收字节	客户端 (转发)	客户端 (实际)	虚拟主机	请求
R	?	?	?	?	?	?	
R	?	?	?	?	?	?	
R	?	?	?	?	?	?	
R	?	?	?	?	?	?	
R	?	?	?	?	?	?	
R	?	?	?	?	?	?	
S	201 ms	0 KiB	0 KiB	10.0.0.1	10.0.0.1	10.0.0.202	GET /manager/status HTTP/1.1

P: 解析和准备request S: 服务 F: 结束 R: 就绪 K: 存活

## 内存数据

Version 4.1.4 running on ubuntu2204, UP for 0 days 2 hours 19 minutes

[这里可以查看系统资源](#)

System information

MEMORY UTILIZATION

CURRENT MEMORY USAGE IS 16% [Advise Garbage Collection](#)  
FREE: 40.86 MB TOTAL: 118.00 MB MAX: 482.00 MB

OS INFORMATION

JVM: Java(TM) SE Runtime Environment 11.0.23+7-LTS-222 Java HotSpot(TM) 64-Bit Server VM  
OS: Linux (unknown) amd64 5.15.0-125-generic  
PROCESSORS: 2  
CURRENT TIME: Wed Dec 11 23:56:23 CST 2024  
WORKING DIR: /

CONTAINER INFORMATION

CONTAINER: Apache Tomcat/9.0.89  
CATALINA\_BASE: /usr/local/apache-tomcat-9.0.89  
CATALINA\_HOME: /usr/local/apache-tomcat-9.0.89  
APPLICATION BASE: /usr/local/apache-tomcat-9.0.89/webapps  
CONFIGURATION BASE: /usr/local/apache-tomcat-9.0.89/conf/localhost

SYSTEM

Overview Memory utilization System properties  
OS Information OS/Hardware Info (Slow)  
Wrapper control Trust Store

点击这里可以查看内存变化

10.0.0.202:8082/probe/memory.htm

应用 Learn VIM while p... Google https://twitter.co... 仪表板 数据结构基础 | Cou... Computer Science... JS 官方文档 openai/openai-co... Python零基础高新区 Shell-官方查询帮助... 力扣 (LeetCode)

Version 4.1.4 running on ubuntu2204.wang.org, UP for 0 days 2 hours 29 minutes

CURRENT MEMORY USAGE

NAME	USAGE SCORE	PLOT	USED	COMMITTED	MAXIMUM	INITIAL	GROUP
Compressed Class Space	20.58 MB	21.30 MB	1.00 GB	0 B	NON_HEAP		
Metaspace	147.33 MB	149.94 MB	-1 B	0 B	NON_HEAP		
CodeHeap 'non-nmethods'	1.33 MB	2.44 MB	5.56 MB	2.44 MB	NON_HEAP		
CodeHeap 'profiled nmethods'	33.65 MB	34.63 MB	117.22 MB	2.44 MB	NON_HEAP		
G1 Eden Space	5.00 MB	59.00 MB	-1 B	8.00 MB	HEAP		
G1 Old Gen	62.46 MB	89.00 MB	484.00 MB	24.00 MB	HEAP		
G1 Survivor Space	1.95 MB	2.00 MB	-1 B	0 B	HEAP		
CodeHeap 'non-profiled nmethods'	9.09 MB	9.25 MB	117.22 MB	2.44 MB	NON_HEAP		
Total	281.39 MB	367.55 MB	1.71 GB	39.31 MB	TOTAL		

MEMORY USAGE HISTORY

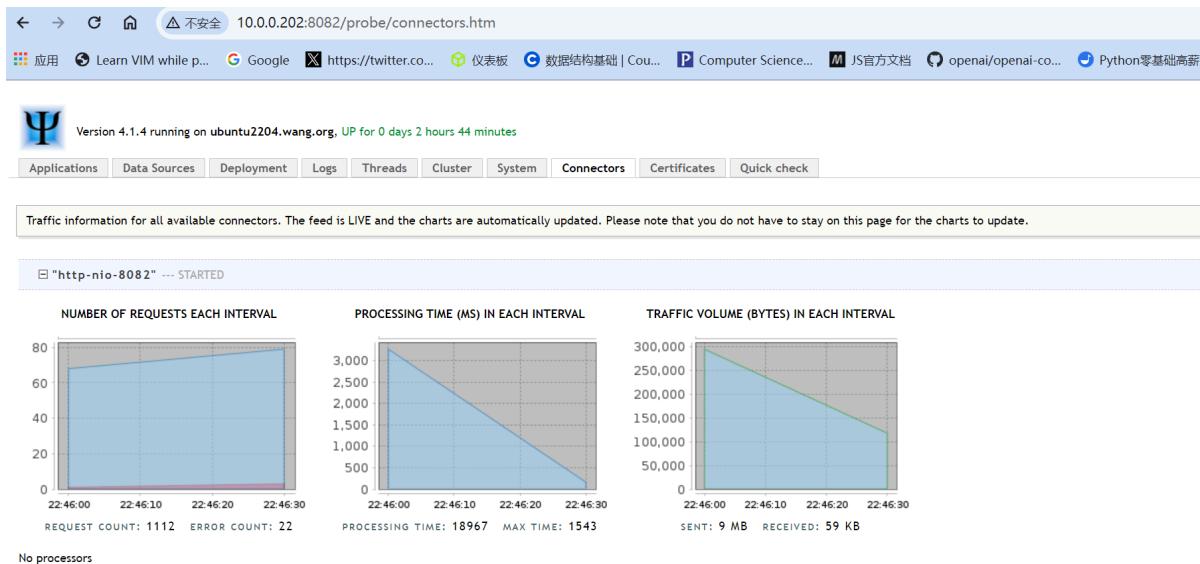
根据图表，内存一定是一上一下成锯齿状波动的，如果是一条横线就一定是出现了问题了

因为存在垃圾回收机制，所以

- 锯齿状图形：表示垃圾回收机制正常工作，定期回收内存。
- 水平状态：可能指示垃圾回收未能有效释放内存，需关注内存管理
- 优化和监控：通过调整 JVM 参数、分析堆转储、持续监控和优化代码来改进垃圾回收和内存使用。

整体的优化过程基本是围绕着通过调整 JVM 参数，去进行垃圾回收行为的优化，和通过分析堆转储找到内存泄漏和高内存使用的原因

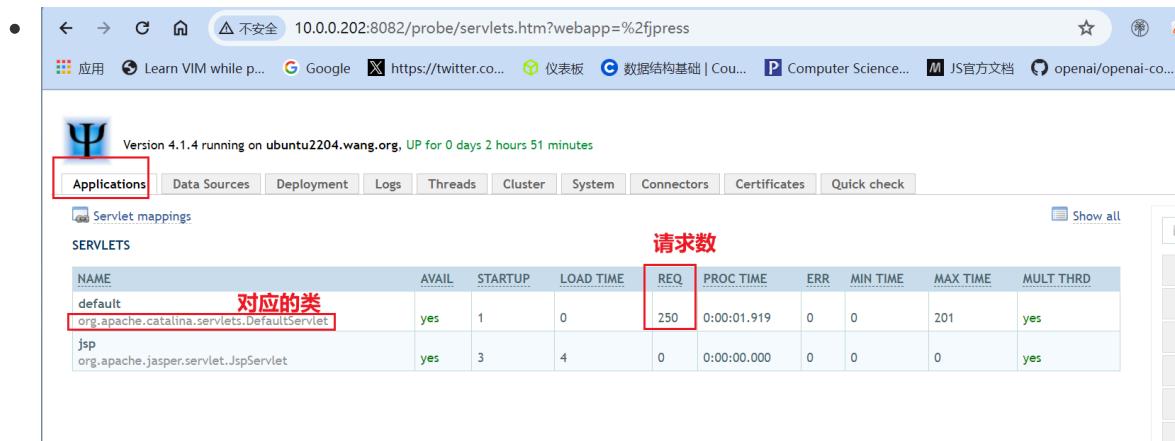
连接器



重点关注类似点击率（每隔多少时间有多少请求数）和吞吐量

## probe重点关注数据总结

- 请求数，分析是按个站点的请求内容比较多，如果出问题的话是哪个类引起的，这些都可以从下图看出，虽然运维看的不是特别明确但是开发是看的懂的，哪个类在消耗，开发自身应该很清楚



- 通过线程池数据分析线程数，看线程够不够用，如果线程用满了就即使去增大，防止出现排队，出现排队超时

Version 4.1.4 running on ubuntu2204.wang.org, Up for 0 days 2 hours 20 minutes

Applications | Data Sources | Deployment | Logs | **Threads** | Cluster | System | Connectors | Certificates | Quick check | What are those abbreviations?

Threads Pools

ID	NAME	EXEC. POINT	STATE	IN.NATIVE	SUSP.	WC	BC
1	main	java.net.PlainSocketImpl.socketAccept ( native code )	RUNNABLE	true	false	1	0
12	AsyncFileHandlerWriter-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	29	0
15	logback-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	106	0
17	Probe_Quartz-1	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16219	270
18	Probe_Quartz-2	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16223	271
19	Probe_Quartz-3	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16218	288
2	Reference Handler	java.lang.ref.Reference\$WaitForReferencePendingList ( native code )	RUNNABLE	false	false	0	4
20	Probe_Quartz-4	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16212	283
21	Probe_Quartz-5	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	16206	271
22	ProbeScheduler_Quartz5SchedulerThread	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	874	638
23	Log4j2-TF-5-Scheduled-1	jdk.internal.misc.Unsafe.park ( native code )	TIMED_WAITING	false	false	270	0
24	mysql-cj-abandoned-connection-cleanup	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	1619	2
25	HikariPool-1-housekeeper	jdk.internal.misc.Unsafe.park ( native code )	TIMED_WAITING	false	false	273	0
27	TokenManager	java.lang.Object.wait ( native code )	TIMED_WAITING	false	false	55	1
28	ForkJoinPool.commonPool-worker-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	90	0
29	jboot-scheduler-thread-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	107	0
3	Finalizer	java.lang.Object.wait ( native code )	WAITING	false	false	67	66
30	jboot-scheduler-thread-2	jdk.internal.misc.Unsafe.park ( native code )	TIMED_WAITING	false	false	117	0
31	jboot-scheduler-thread-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	136	0
32	jboot-scheduler-thread-4	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	98	0
33	jboot-scheduler-thread-5	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	82	0
34	jboot-scheduler-thread-6	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	117	0
35	jboot-scheduler-thread-7	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	216	0
36	jboot-scheduler-thread-8	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	362	0
37	ContainerBackgroundProcessor[[StandardEngine[Catalina]]]	java.lang.Thread.sleep ( native code )	TIMED_WAITING	false	false	809	0
38	http-nio-8082-exec-1	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	69
39	http-nio-8082-exec-2	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	20
40	http-nio-8082-exec-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	63
41	http-nio-8082-exec-4	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	74	55
42	http-nio-8082-exec-5	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	75	3
43	http-nio-8082-exec-6	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	76	6
44	http-nio-8082-exec-7	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	74	28
45	http-nio-8082-exec-8	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	76	1
46	http-nio-8082-exec-9	sun.management.ThreadImpl.getThreadInfo ()	RUNNABLE	false	false	76	7
47	http-nio-8082-exec-10	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	73	66
48	http-nio-8082-Poller	sun.nio.ch.Pollable.poll ( native code )	RUNNABLE	true	false	0	181
49	http-nio-8082-Acceptor	sun.nio.ch.ServerSocketChannelImpl.accept0 ( native code )	RUNNABLE	true	false	0	0
50	http-nio-8082-syncTimeout	java.lang.Thread.sleep ( native code )	TIMED_WAITING	false	false	8083	0
52	J2D Driposer	java.lang.Object.wait ( native code )	WAITING	false	false	15	14
54	logback-3	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	47	0
55	jboot-scheduler-thread-9	jdk.internal.misc.Unsafe.park ( native code )	WAITING	false	false	91	0

http线程池，当前一共10个  
只有一个在使用RUNNABLE  
如果都在使用，就表示繁忙，  
此时可能需要着增加连接数

- 观察内存中内存分布的情况（青年态，老年态）

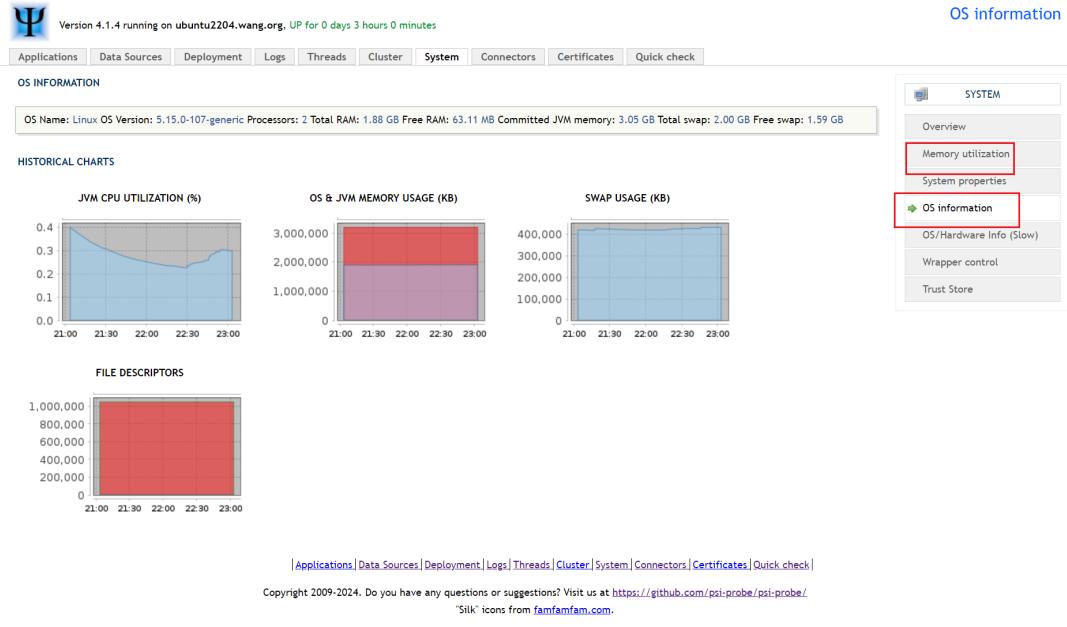
- (可以在catalina.sh的文件中指定内存的大小)，后期根据实际情况去调整内存的分布,防止内存溢出和产生频发回收机制(比如年轻态过小，就会频繁出发回收，而频繁的GC会消耗系统资源)

```
#!/bin/sh
JAVA_OPTS="-Xms500m -Xmx1024m -XX:SurvivorRatio=3 -XX:NewSize=250m -XX:MetaspaceSize=300m"
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements. See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License. You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

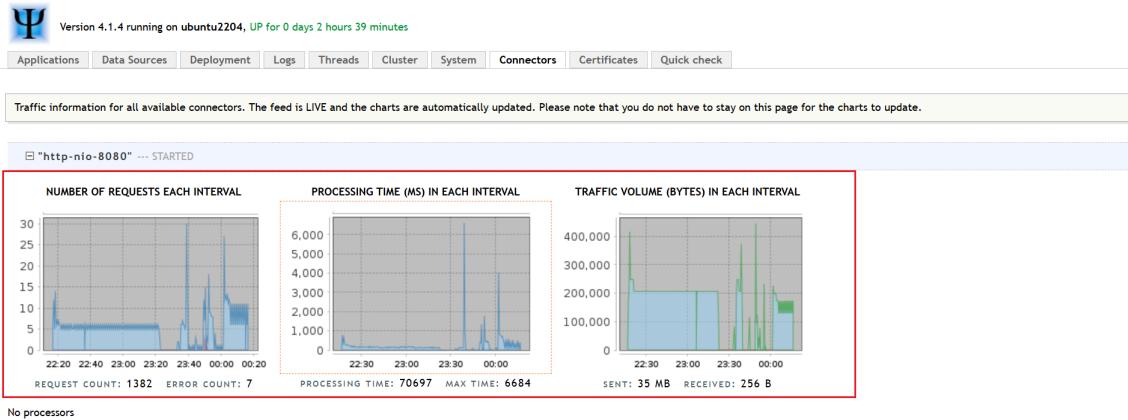
# -----
# Control Script for the CATALINA Server
#
# Environment Variable Prerequisites
#
# Do not set the variables in this script. Instead put them into a script
# setenv.sh in CATALINA_BASE/bin to keep your customizations separate.
#
# CATALINA_HOME May point at your Catalina "build" directory.
#
# CATALINA_BASE (Optional) Base directory for resolving dynamic portions
# of a Catalina installation. If not present, resolves to
# the same directory that CATALINA_HOME points to.
#
# CATALINA_OUT (Optional) Full path to a file where stdout and stderr
# will be redirected.
#
```

- 通过看系统资源的情况分析，比如由于内存过小，导致频繁出发GC，导致CPU使用率增高
- JMP CPU UTILIZATION过高，就会导致CPU负载变高

- FILE DESCRIPTIONS 变高，又或者发生频繁的读取文件，频繁导致磁盘IO变高



- 观察 (http) 连接器里的点击率和吞吐量



## JVM调优

JVM当前有三家公司开发分别是

- SUN (ORACLE)
- BEA
- IBM

## JVM工作原理

```
public class MyMath {
    public static final int INIT_DATA = 666;
    public static User user = new User();

    public int compute() { // 一个方法对应一块栈帧内存区域
        int a = 1;
        int b = 2;
        int c = (a + b) * 10;
        return c;
    }
}
```

```

public static void main(String[] args) {
    MyMath math = new MyMath();
    int result = math.compute();
    System.out.println("计算结果为: " + result); // 输出结果
}

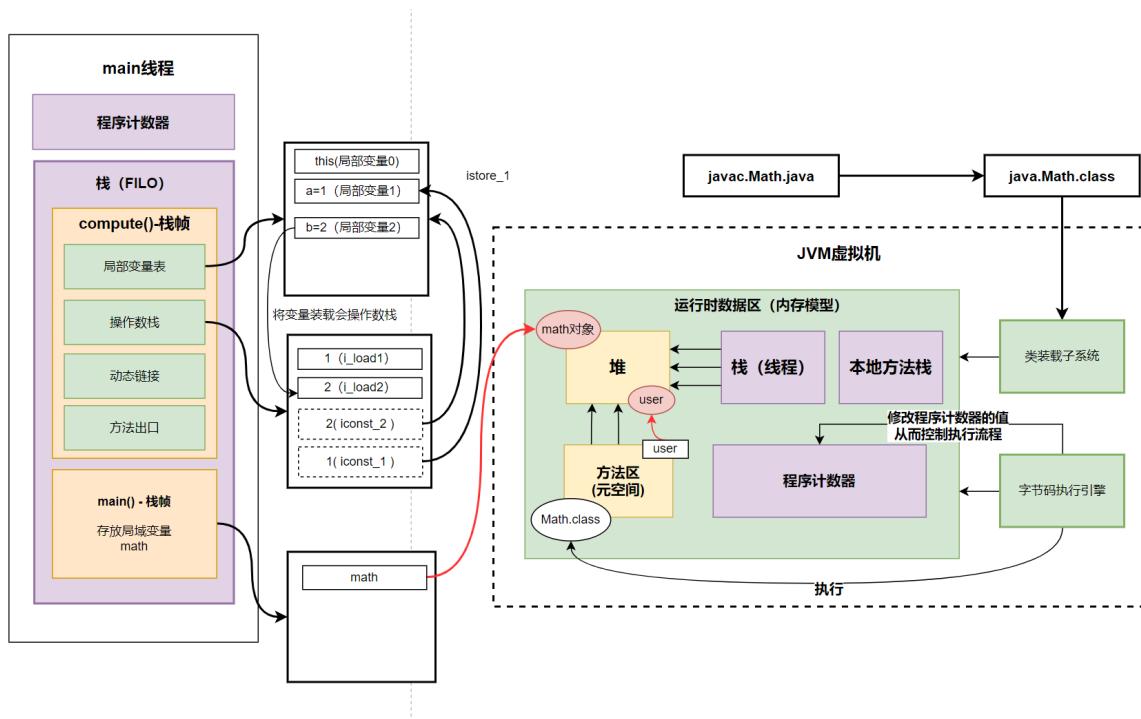
class User {
    private String name;

    public User() {
        this.name = "Default User";
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```



## java程序编译执行流程

- 第一步：通过javac指令将Math.java变成字节码文件，在通过java命令去运行
- 第二步：执行java命令后，字节码文件加载到类装载子系统，通过类装载子系统将字节码加载到JVM的第二块组成部分（内存区域或运行时数据区）
- 第三步：通过JVM第三个组成部分：字节码执行引擎来执行内存区域里的代码

我们学的堆，栈，元空间等概念都仅仅指的是内存区域组成部分的一小块内容  
我们的JVM调优，主要调的就是内存区域

## JVM内存区域详解

- 堆 (Heap)
  - 通常new的实例都在堆中
- 栈 (Java Virtual Machine Stacks)
  - 也叫线程栈
  - 当我们的**main方法**开始运行，就会启用一个线程运行，而在线程运行过程中，局部变量需要有内存去存放，而这些局部变量都是放在线程栈
  - 只要有一个线程开始运行，JVM就会给这个线程分配一个**专属的**栈内存区域，用来放局部变量
  - **栈帧**
    - 当我们的线程开始运行，JVM就会给这个线程分配一块自己专属的线程栈
    - 而局部变量的作用范围通常只在方法内部有效，比如上述代码中的变量a, b, c，只在方法compute执行的过程中有效，
    - 而math只在main方法执行的过程中生效，方法一结束，这个局部变量肯定就没有了
      - 实现过程：
        - 只要我们的线程开始执行方法，马上会给这个方法在我们的线程栈内存区域分配一个块方法对应的内存区域，用来存放方法内部的局部变量
        - 当调用compute方法的时候，也会给这个方法分配一块自己内部的内存区域，我们把方法对应的这个内存区域就叫**栈帧**
  - **数据结构的栈和这个栈空间的关系是什么**
    - 栈的特点是FILO(First IN Last Out)
    - 我们的线程栈这块内存区放栈帧的这个数据结构，就是用的数据结构里的栈
    - 根据上述代码，我们的程序在运行过程中，他的内存分配如下
      - 先是线程开始运行，分配一大块内存区域（线程栈）
      - 然后开始执行main方法，然后分配一块main方法的栈帧内存区域，然后执行compute方法，compute方法又会被分配一块栈帧内存区域
      - compute方法一旦运行完，它对应的栈帧局部内存空间全部都会被释放掉，也就是所谓的出栈
      - 然后在回到main方法继续执行，main方法执行完，一样的也会把main方法对应的栈帧内存空间销毁掉，说白了就是出栈
      - 这也就是为什么要用栈这种数据结构来存储我们的方法，因为他跟我们程序的嵌套调用的先后顺序完全一致。

```
// 通过javap -c MyMath > MyMath.txt, 得到java字节码的反汇编
// 对上述代码生成的字节码进行反汇编
// 可以通过对照oracle上面的JVM指令码手册，进行阅读
Compiled from "MyMath.java"
public class MyMath {
    public static final int INIT_DATA;

    public static User user;

    public MyMath();
    Code:
        0: aload_0
```

```

1: invokespecial #1                         // Method java/lang/Object."<init>":
Ov
4: return

public int compute();
Code:
0: iconst_1                                // iconst_1 将int类型常量1压入操作数栈, 操
作数栈内部也是栈结构来存储数据
1: istore_1                                 // 将int类型的值存入局部变量1, 这里也就是将
操作栈中的常量1出栈, 然后给局部变量a开辟一个内存空间, 然后将常量1存入局部变量a的内存空间中
2: iconst_2                                 // iconst_2 将int类型常量2压入操作数栈
3: istore_2
4: iload_1                                  // iload 从局部变量中装载int类型值1
5: iload_2
6: iadd                                     // iadd 从操作数栈中, 弹出两个值, 做加法, 得
到的结果, 重新压回操作数栈
7: bipush        10                         // bipush 将一个8位带符号整数(10)压入栈
9: imul
10: istore_3
11: iload_3
12: ireturn

public static void main(java.lang.String[]);
Code:
0: new           #2                         // class MyMath
3: dup
4: invokespecial #3                         // Method "<init>":()V
7: astore_1
8: aload_1

```

## • 局部变量表与操作数栈

- 在我们的栈帧中, 有一个表结构来存放我们的局部变量, 也就是**局部变量表**
- 整个栈帧中除了有**局部变量表**, **操作数栈**, **动态链接**, **方法出口**
- **局部变量表**就是存放局部变量的数据存储结构
- **操作数栈**临时存放局部变量值的地方
- **程序计数器**每个线程从程序计数器的内存空间中, 取一部分作为自己的程序计数器内存空间, 内部存放的数据是记录我们的代码执行到什么位置 (方法区的内存地址)
  - 程序计数器的意义: 为什么JVM的程序设计人员要设计它: 当线程时间片用完, 产生线程间切换的时候, 该程序计数器记录线程运行的位置, 在后续切换回来的时候使用
  - Math.class在java运行的时候, 要将其加载到方法区 (也就是元空间), 加载到元空间后, 由字节码执行引擎执行
  - 程序计数器上记录了代码执行的位置, 而这个程序计数器的值由字节码执行引擎处理
- **动态链接**: 涉及C++底层源码, java程序是由C++写的,
  - 大概意思: 我们的函数名, 方法名, 看做一个符号, 比如compute就是一个符号, 动态链接实质上就是程序在运行过程中, 根据这个符号找到这个方法所在的方法区内的内存地址, 根据这些内存地址作为入口, 找到这个方法的代码进行执行
  - 所以动态链接放的就是我们的方法对应的在方法区内的解析之后的入口的内存地址, 根据这个内存地址可以找到compute方法的具体代码

- **方法出入口**: 记录compute代码执行完后, 返回到main方法的那个位置
- 每个栈帧都有上述的这四个部分, 即: 局部变量表, 操作数栈, 动态链接, 方法出入口
- **main方法内的局部变量表**: 有些不同的地方: 就是Math.math接受的值是一个对象( new Math() ), 而这个对象是被分配在堆里的, 而我们的局部变量比如: a=1是被分配在局部变量表中
- 那么局部变量表中的变量和堆中的math对象有什么区别: 也就是局部变量表中的数值是存放的常量, 而对象则是存放的堆中存放该对象数据的地址, 即指针
- **方法区 (元空间)** : 方法区内放常量, 静态变量, 类元信息,
  - **常量**: `public static final int initData = 666;`
  - **静态变量**: `public static User user = new User();`, 而这个变量的值是一个变量, 因此也是一个指针指向堆内的user对象, 这也就是方法区和堆之中的关系
  - **本地方法栈**: 本地方法就是native方法, 底层是C++写的, 这部分native方法就是从本地方法栈分配的内存空间
    - 和线程栈比较类似, 线程栈是java语言实现的方法, 它的内存空间是从栈空间分配的
    - 对于本地方法 (native方法) , 底层是C++/C语言实现的, 而这些方法的内存空间从本地方法中分配

## 堆

- 对象new出来时放在Eden区, 当我们不断new对象, 将Eden区撑满, 会触发minor GC, minor GC本质上是字节码执行引擎开启的一个线程, 专门用来做垃圾对象回收,
- **GC底层垃圾回收的过程**:
  - **可达性分析算法**
    - 将"GC Roots"对象作为起点, 从这些节点开始向下搜索引用的对象, 找到的对象都标记为非垃圾对象, 其余未标记的对象都是垃圾对象,
    - 比如: 可以以math变量作为GC ROOT, 看math变量有没有成员变量, 如果有引用成员变量, 继续向下找, 直到找到最后一个对象, 它没有任何引用的成员变量, 整个链上的所有变量被打一个标记, 记为非垃圾, 而所有的非垃圾会复制到survivor上去, 所有没有GC ROOT指向的对象, 也就是垃圾对象, 直接销毁掉
    - 一个对象在经历过GC后, 他的分代年龄会加1, 假设程序一直在运行, 下一次, Eden区又放满了, 会再次出发minor GC, minor GC实质上回收的是整个年轻代, 它不仅会回收Eden区, 还会回收Survivor区的对象, 如果第二次回收, survivor区域中还有对象存活, 它会被复制到另一块survivor区域, 即s1区, 如果eden区还有对象存活, 也会复制到s1区, 剩下的eden去和s0区里的所有对象直接清空, 效率非常高
    - 对象的组成: 对象中除了实例数据 (比如: 成员变量外) , 还有对象头 (Object Header) , 包含对象的一些附属信息, 比如: 锁状态, 数组长度 (只有数组对象才有) , 分代年龄 (用4bit存储)
    - 每经过一次minor GC, 分代年龄都会加1, 当分代年龄变为15后, 直接移到老年代 (这里的15不是绝对的, 不同的垃圾回收器的值不同)
    - 在web应用中, 向数据库连接池, 对象缓存池, 静态变量这些最终都会移到老年代
    - **GC Roots根节点**: 线程栈的本地变量, 静态变量, 本地方法栈的变量等等

## 总结: 三种对象从青年代移动到老年代的原因

- 对象的年龄超过阈值（默认15，具体看JVM实现）
  - 当对象在 Survivor 区的年龄超过了阈值，JVM 会将对象移动到老年代
  - 过程解释
    - 在年轻代中，内存划分为 Eden、Survivor from (S0) 和 Survivor to (S1) 三部分。
    - 新创建的对象首先分配在 Eden 区
    - 当 Eden 区内存满时，Minor GC 触发
    - 存活的对象将被移入 Survivor S0 区。
    - 在接下来的 GC 中，存活的对象会从 S0 → S1，它们的“年龄”会增加 1。
    - 当对象的年龄达到某个阈值（默认为15），JVM 就会将该对象移入老年代。
  - 年龄阈值
    - JVM 中的对象年龄由 MaxTenuringThreshold 决定，默认值是15。
    - 也就是说，如果一个对象的年龄达到 15（在 S0/S1 之间存活了 15 次 GC），这个对象就会被移到老年代
  - 修改阈值

`-XX:MaxTenuringThreshold=10`

# 将对象的“最大生存代数”调整为 10。也就是说，当对象在 Survivor 区经历了 10 次 GC 后，它就会被移入老年代。

- 对象的年龄超过阈值
  - 大对象（通常是大数组或大字符串）会直接分配到老年代，而不会经历年轻代的过程。
  - 为什么要直接进入老年代？
    - 如果一个大对象（比如一个大数组或大字符串）被分配到 Eden 区，可能会很快填满 Eden 区，频繁触发 GC。这种频繁的 GC 会导致性能问题。为了解决这个问题，JVM 提供了一种机制，大对象直接进入老年代。
  - 如何定义大对象的大小？
    - JVM 使用 `PretenureSizeThreshold` 参数来决定大对象的最小大小。
    - 只要对象的大小大于 `PretenureSizeThreshold`，它就会被直接分配到老年代。
  - 如何设置大对象的阈值？

# 表示大于 1MB 的对象将直接分配到老年代。  
`-XX:PretenureSizeThreshold=1M`

- 特殊情况：Survivor 区满了的情况
  - 如果 Survivor 区满了，存活的对象也会被直接送入老年代。
  - 为什么会发生？
    - 在 GC 过程中，如果 Survivor 区中的空间不足以存放所有 Eden 和 Survivor 中存活的对象，那么一部分对象会直接被分配到老年代。
    - 这有点像应急转移的机制。
  - 这种情况在高并发场景中常见，比如网络高并发请求，瞬时大量对象创建导致 Survivor 区不够用。

## 堆满时，老年代如何分配？

- 当老年代也满了时，JVM 将触发 Full GC。
- Full GC 触发后，如果无法回收老年代中的对象，JVM 会抛出：

```
java.lang.OutOfMemoryError: Java heap space
```

## JVM调优工具

### 使用jvisualvm监控内存

jvisualvm一款图形化的内存监控工具，在jdk-8u361之前的版本中是内置的组件，但在之后的JDK版本中已经取消了该组件，要单独下载并配置

```
# 安装依赖
apt install libxrender1 libxrender1 libxtst6 libxi6 fontconfig -y

wget https://github.com/oracle/visualvm/releases/download/2.1.8/visualvm_218.zip

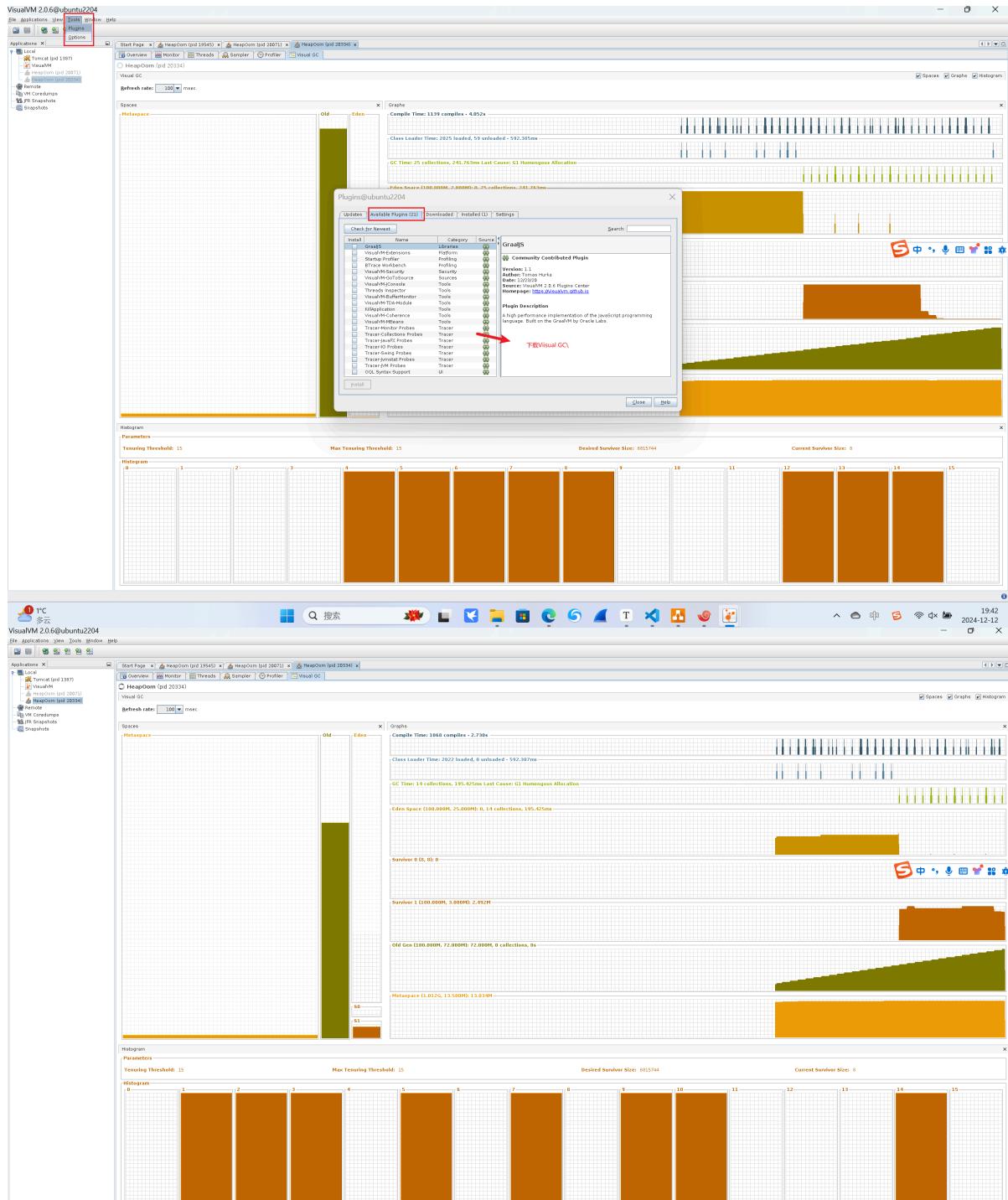
unzip visualvm_218.zip

# 在windows中开启Xmanager - Passive
export DISPLAY=10.0.0.1:0.0

# 执行，在windows中能看到GUI界面，在GUI中点击Tools菜单，Plugins，然后安装VisualGC插件
```

### 测试程序

```
// HeapOom.java
import java.util.ArrayList;
import java.util.List;
public class HeapOom {
    public static void main(String[] args) {
        List<byte[]> list = new ArrayList<byte[]>();
        int i = 0;
        boolean flag = true;
        while(flag) {
            try {
                i++;
                list.add(new byte[1024 * 1024]); // 每次增加一个1M大小的数据对象
                Thread.sleep(1000);
            } catch (Throwable e) {
                e.printStackTrace();
                flag = false;
                System.out.println("count=" + i); // 记录运行次数
            }
        }
    }
}
```



老年代放满后，会触发**Full GC**，它会回收整个堆的内存空间，但是由于上述代码的对象一直在被列表引用，Full GC无法回收，因此会触发OOM

```
java.lang.OutOfMemoryError: Java heap space
```

## 阿里巴巴内部JVM调优工具Arthas详解

启动Demo

```
curl -o https://arthas.aliyun.com/arthas-boot.jar
java -jar arthas-demo.jar
```

### Arthas使用场景

得益于Arthas强大且丰富的功能，让Arthas能做的事情超乎想象。下面仅仅例举几项常见的使用情况，更多的使用场景可以在熟悉了Arthas之后自行探索

- 是否有一个全局视角来查看系统的运行状况？
- 为什么CPU又升高了，到底是哪里占用了CPU？
- 运行的多线程有死锁吗？有阻塞吗？
- 程序运行耗时很长，是哪里耗时比较长呢？如何监测呢？
- 这个类从哪个jar包加载的？为什么会报各种类相关的Exception？
- 我改的代码为什么没有执行到？难道是我没commit?分支搞错了？
- 遇到问题无法在线上debug，难道只能通过加日志再重新发布吗？
- 有什么办法可以监控到JVM的实时运行状态？

```
[root@ubuntu2204 ~]$ java -jar arthas-boot.jar
[INFO] JAVA_HOME: /usr/lib/jvm/java-11-oracle-x64
[INFO] arthas-boot version: 4.0.4
[INFO] Found existing java process, please choose one and input the serial number of the process, eg : 1. Then hit ENTER.
* [1]: 19208 org.netbeans.Main
* [2]: 1387 org.apache.catalina.startup.Bootstrap
1
[INFO] arthas home: /root/.arthas/lib/4.0.4/arthas
[INFO] Try to attach process 19208
Picked up JAVA_TOOL_OPTIONS:
[INFO] Attach process 19208 success.
[INFO] arthas-client connect 127.0.0.1 3658
arthas
```

wiki https://arthas.aliyun.com/doc  
tutorials https://arthas.aliyun.com/doc/arthas-tutorials.html  
version 4.0.4  
main\_class  
pid 19208  
time 2024-12-12 20:18:10.234

[arthas@19208]\$

## 执行dashboard

# 把当前的监控进程里面的所有的线程执行的情况，全部打出来，包括内存的分配情况  
dashboard

ID	NAME	GROUP	PRIORITY	STATE	%CPU	DELTA_TIME	TIME	INTERRUPT	DAEMON
-1	C1 CompilerThread0	-	-1	-	0.67	0.033	0:14.245	false	true
-1	C2 CompilerThread0	-	-1	-	0.5	0.025	1:8.077	false	true
1180	Timer-for-arthas-dashboard-17ce	system	5	RUNNABLE	0.45	0.022	0:0.125	false	true
-1	VM Periodic Task Thread	-	-1	-	0.21	0.010	0:17.526	false	true
1178	arthas-NettyHttpTelnetBootstrap	system	5	RUNNABLE	0.19	0.009	0:0.344	false	true
18	AWT-XAWT	system	6	RUNNABLE	0.16	0.007	0:54.998	false	true
35	Timer-0	system	1	TIMED_WAIT	0.04	0.001	0:4.529	false	true
-1	G1 Young RemSet Sampling	-	-1	-	0.03	0.001	0:4.259	false	true
-1	VM Thread	-	-1	-	0.02	0.000	0:2.068	false	true
-1	GC Thread#1	-	-1	-	0.01	0.000	0:1.175	false	true
1182	Inactive RequestProcessor threa	system	1	TIMED_WAIT	0.01	0.000	0:0.001	false	true
-1	GC Thread#0	-	-1	-	0.01	0.000	0:1.193	false	true
Memory									
heap	82M	124M	768M	10.80%	gc.g1_young_generation.count	40			
gl_eden_space	41M	61M	-1	67.21%	gc.g1_young_generation.time(ms)	704			
gl_old_gen	36M	58M	768M	4.81%	gc.g1_old_generation.count	1			
gl_survivor_space	5M	5M	-1	100.00%	gc.g1_old_generation.time(ms)	402			
nonheap	123M	133M	-1	92.34%					
codeheap_`non-nmethods'	1M	2M	5M	29.52%					
metaspace	74M	80M	-1	92.68%					
codeheap_`profiled_nmethod	28M	30M	117M	24.45%					
compressed_class_space	9M	11M	1024M	0.94%					
codeheap_`non-profiled_nme	8M	9M	117M	7.42%					
thods'									
Runtime									
os.name	Linux								
os.version	5.15.0-125-generic								
java.version	11.0.25								
java.home	/usr/lib/jvm/java-11-openjdk-amd64								
system.load.average	0.13								
processors	2								
timestamp/uptime	Thu Dec 12 20:21:13 CST 2024/9243s								

## thread <线程号>

# 定位这个线程占用CPU执行的具体的代码  
thread 18

更多的用法：查看Arthas官网的更多详细高阶用法

## JVM虚拟机调优的目的

尽可能的减少GC，不论是FULL GC还是MINOR GC，减少GC，特别是FULL GC的目的是尽量减少STW (Stop The World)

STW一旦发生，在客户端的影响就是，用户会发现卡顿

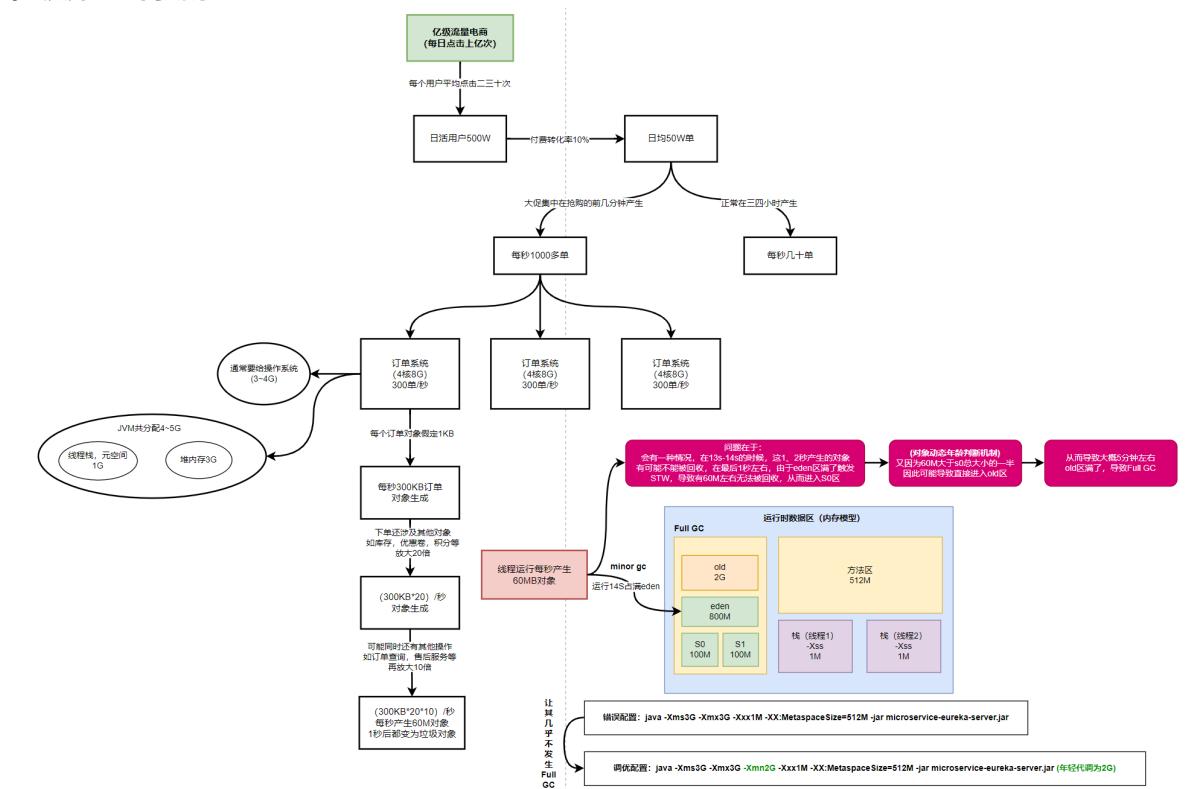
### JVM的开发人员为什么要设计SWT机制

使用反证法，假设JVM在做GC的过程中，不STW，没有STW机制，后台线程在做GC的过程中，同时用户线程也继续在执行，会发生什么效果

- 如果用户线程不暂停，程序一直在执行，则对象的状态会一直变，可能GC在或变量检查时，还不是垃圾对象，但是GC可能还没检测完，而程序线程运行外，导致之前的部分非垃圾对象变为垃圾对象，此时怎么办？GC重新遍历检测吗？
- 所以为了防止这个情况发生，就需要SWT机制，在GC触发时，暂停线程运行，直到GC结束
- 不同的垃圾收集器的SWT会有区别

## 通过案例讲解JVM优化

### 亿级流量电商场景



## 堆和栈的区别

程序执行需要加载到内存，内存中分堆和栈的目的是将数据类型进行区分，将不同的对象放到不同的内存中，用不同的数据结构去组织，已达到高效的目的

栈是运行单位，所有的运行对象，都是在栈中，当程序运行时，JVM会为每个线程分配一个栈大小。每个线程栈是不通用的，因为每个任务都有一个独立的线程执行。

堆是存储单位，所以有需要使用的数据都在堆中，堆是可以共享的。也就是说，堆是处理数据的地方，栈是用来处理逻辑的地方。

之所以区分堆和栈，这样的好处是可以将业务逻辑和数据进行分离，同时也可以提高数据的共享程度。

堆中有垃圾回收机制GC，因此要继续对堆内存进行划分

- 年轻代 (Young Generation)
- 年老代 (Old Generation)
- 持久代 (Permanent Generation)
  - JDK8之后，不再叫持久代，而叫做元空间(metaspaces)
  - 持久代在JVM的内存中，而元空间在本地内存中

## JVM的三代（内存）

所谓的jVM的调优，本质上就是设置内存的使用

## GC回收方法

- GC回收的是堆内的内存，主要对3代进行回收

Java中GC回收机制有两种，Scavenge GC和Full GC

- Scavenge GC回收机主要是对年轻代进行回收，或者说年轻代主要使用ScavengeGC来回收内存，年轻代分为一个eden区和两个Survivor区
- Scavenge GC首先是对eden区的对象进行回收，如果回收结束后还是存活对象，我们会将他放入Survivor区，由于eden区的对象变化很频繁，所以在年轻代中经常使用Scanvange进行回收
- Full GC是指对整个堆内存的对象进行回收，对年轻代，老年代和持久代都会进行回收，但是堆Java8的版本已经没有持久代，被metaspaces所替代。由于Full GC回收的内容比较多，所以一般不可能频繁触发Full GC回收
  - 年老代被写满
  - 持久代被写满
  - system.gc()被调用
  - 上一次GC后，heap分配的策略发生变化

## GC的算法

引用计数、标记清除算法

- 对每个对象进行计数，如果对象引用一次，那么计数+1，如果这个引用释放那减1，如果计数为0，那么表示这个对象要消除
- 标记清除算法会不断扫描GC Roots，将存活的对象标记出来，等到不用的时候再清除

复制算法

将内存分为两份，每次只用一份，通过来回复制来解决碎片的问题

## 垃圾收集的算法

通常GC回收的算法有三种，串行收集器，并行收集器，并发收集器

# JVM的参数设置

- 实际上是设置每个代的大小

## 修改JVM参数的方法

修改JVM的方法通常有三种

- 使用eclipse进行设置
- 使用java小程序来设置
- 修改配置疑问还能来设置JVM
  - 使用bin/catalina.sh;
  - 修改bin/startup.sh

修改JVM的语法有三种

```
#第一种语法
set CATALINA_OPTS=-Xmx512m -Xms512m -Xmn64m -Xss2m

# 第二种语法
set JAVA_OPTS=-Xmx512 -Xms512 -Xmn64m -Xss2m

# 第三种
JAVA_OPTS="-Xms512 -Xmx1024m -Xmn512m"

# 堆相关参数
-Xms:          # 初始堆大小
-Xmx:          # 最大堆大小
# 初始堆大小一般会设置与最大堆大小一致，对于32位操作系统来说，-Xmx设置为1.5G-2.5G即可，如果是
64位则一般不做限制
-XX:NewSize=n;    # 设置年轻代大小
-XX:MaxNewSize   # 设置新生代最大空间大小
-XX:NewRatio=n   # 设置年轻代和老年代的比值，如为3，代表年轻代与老年代的比例是1:3
-XX:SurvivorRatio=n; # 如果值为3:2，表示Eden:Survive=3:2
-XX:MaxPermSize:n # 设置持久代大小
-Xss: #设置每个线程的堆大小
```

metaspace参数设置

```
-XX:MetaspaceSize    # 初始空间大小，达到该值就会触发垃圾回收机制进行类卸载，同时GC会对该值进
行调整，如果释放了大量的空间，就适当降低该值，如果释放了很少的空间，那么不超过MaxMetaspaceSize
时，适当提高该值
-XX:MaxMetaspaceSize # 最大空间，默认没有上线
-XX:MinMetaSpaceFreeRatio # 在GC之后，最小的Metaspace剩余的空间容量的百分比，减少为分配
空间所导致的垃圾收集
-XX:MaxMetaspaceFreeRatio # 在GC之后，最大的Metaspace剩余空间容量的百分比，减少为释放空
间所导致的垃圾收集
```

并行收集器

```
-XX:ParallelGCThreads=n # 设置并行收集器时使用的CPU数，并行收集线程数
-XX:MaxGCPauseMillis=n # 设置并行收集最大暂停时间
-XX:GCTimeRatio=n      # 设置垃圾回收时间占程序运行时间的百分比，公式为1/(1+n)
```

## 并发收集器

```
-XX:+CMSIncrementalMode #设置为增量模式，适用于单CPU  
-XX:ParallelGCThreads=n # 设置并发收集器年轻代收集方式为并行收集时，使用的CPU数，
```

## 连接器

Http连接器

AJP连接器

### 问题：jpress的验证码无法识别

问题出现的原因：Java 图形处理功能在 Linux 系统上找不到有效的字体配置时会发生。

解决方案

```
# 1. 安装基础字体和字体配置包  
# ubuntu  
sudo apt-get update  
sudo apt-get install -y fontconfig ttf-mscorefonts-installer  
# CentOS  
sudo yum install -y fontconfig dejavu-sans-fonts  
# 重建字体缓存  
sudo fc-cache -fv  
  
# 2. 检查并配置字体路径(这个的可能性比较大)  
# 将这个变量赋值写入.bashrc或者catalina.sh  
export JAVA_FONTS=/usr/share/fonts  
  
# 3. 验证字体配置  
# 查看这些文件配置是否完整，尤其是/etc/fonts/font.conf  
ls /etc/fonts  
ls /etc/fonts/conf.d/  
  
# 如果依然解决不了  
# 安装或更新 JDK  
# ubuntu  
sudo apt-get install -y openjdk-11-jdk  
# centos  
sudo yum install -y java-11-openjdk-devel  
  
# 设置默认 Java 版本  
# 使用 update-alternatives 设置默认的 Java 版本：  
sudo update-alternatives --config java  
  
# 使用脚本验证是否成功
```

图形验证脚本TestFont.java

```
import java.awt.*;  
import java.awt.image.BufferedImage;  
import javax.imageio.ImageIO;  
import java.io.File;  
import java.io.IOException;
```

```

public class TestFont {
    public static void main(String[] args) {
        int width = 150, height = 50;
        BufferedImage bufferedImage = new BufferedImage(width, height,
        BufferedImage.TYPE_INT_RGB);

        Graphics g = bufferedImage.getGraphics();
        g.setColor(Color.WHITE);
        g.fillRect(0, 0, width, height);
        g.setColor(Color.BLACK);
        g.setFont(new Font("Arial", Font.BOLD, 40));
        g.drawString("Test", 20, 40);

        try {
            ImageIO.write(bufferedImage, "png", new File("testfont.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

测试

```

javac TestFont.java
java TestFont

```

## 扩展知识：Session和Cookie的区别与关系

Cookie

- 存储位置：Cookie是存储在客户端(浏览器)中的一小段数据
- 生命周期：
  - 会话(Session)Cookie：会在浏览器关闭时自动删除
  - 持久性(persistent)Cookie：可以在设定的过期日期之前一直存在
- 传输方式：每次客户端向服务端发送请求时，浏览器会自动将所有相关的Cookie包含在请求头中

Session

- 存储位置：Session数据存储在服务端，通常在服务器的内存或数据库中
- 生命周期：由服务器控制，通常在一段时间内（会话超时）或用户登出后失效
- 传输方式：Session通过SessionID关联，每次请求SessionID通过Cookie或URL参数传到服务器

自我总结：

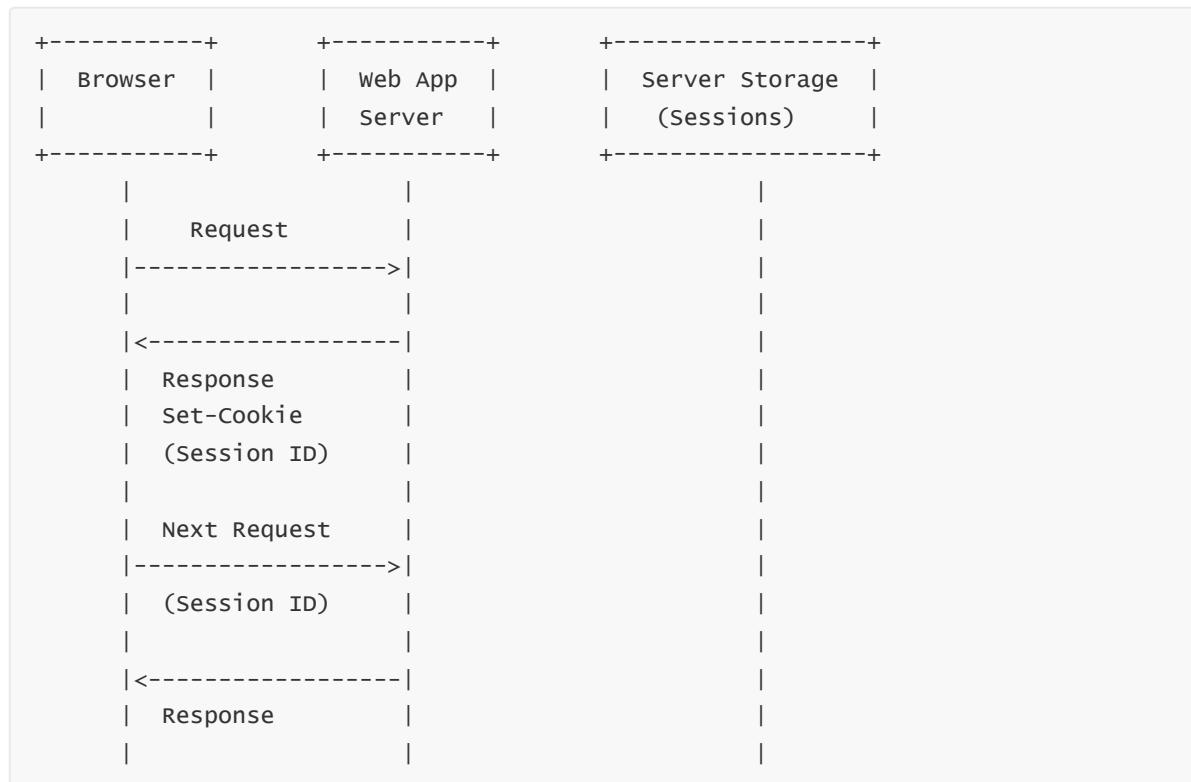
Session通过Cookie传递SessionID来作为从服务器查询数据的索引，从而维持客户端的状态

工作原理与流程

- 用户第一次访问Web应用时，服务器为该用户创建一个Session，并生成一个唯一的SessionID
- 服务器将这个SessionID发给客户端，并通过HTTP响应头的 `set-cookie` 指令将其存储在客户端的Cookie中

- 在后续的请求中，浏览器会自动将存储在Cookie中的SessionID发送给服务器
- 服务器接收到SessionID后，通过它查找相应的Session数据，从而识别用户和他们的会话状态

## 图示



## 关于Session的安全

- HttpOnly
  - 可以防止JavaScript访问SessionID
- Secure
  - 设置Cookie的Secure属性，可以确保Cookie只在https连接中发送
- SameSite
  - 使用SameSite属性防止跨站请求伪造(CSRF)攻击

设置安全Cookie

```
Set-Cookie: JSESSIONID=ABC123; Path=/; HttpOnly; SameSite=Strict
```

## Session存储

- 内存存储
- 数据库存储
- 分布式缓存(Redis)

# Java程序编译

## 编译流程

1. 获取java程序源码
2. 进入源码目录，运行mvn命令，mvn会根据项目中的pom.xml文件和编译参数编译源码，得到.war或.jar结尾的目标文件
3. 部署运行

## 使用Maven编译Java项目

### Maven介绍

Maven是一个项目管理工具，可以对java项目进行构建，解决打包依赖等

Maven基于项目对象模型(POM project object model)实现项目管理，POM即一段描述信息（配置）可以用来管理项目的构建，因此每个maven项目都有一个pom.xml文件

POM(project Object Model,项目对象模型)是Maven工程的基本工作单元，是一个XML文件，包含了项目的基本信息，用于描述项目如何构建，声明项目依赖等，在执行任务或目标时，Maven会在当前目录中查找POM文件，通过读取pom文件获取所需的配置信息，然后执行目标。本质上，每个Maven项目都包含一个POM文件，该文件定义每个项目的对应的相关信息，Maven使用POM来决定不同的构建动作，及制品的生成等

pom.xml文件中可以指定以下配置：

- 项目依赖
- 插件
- 执行目标
- 项目构建 profile
- 项目版本
- 项目开发者列表，相关邮件列表信息
- 用<packing>指定项目打包形式，jar或者war

### Maven的仓库分类

中央仓库 (Central Repository)

Maven官方服务器里面存放了绝大多数市面上流行的jar包，允许用户注册后，上传自己的项目到官方仓库服务器

```
https://mvnrepository.com/  
https://repo.maven.apache.org
```

镜像仓库 (Mirror Repository)

对于国内来说访问国外的中央Maven仓库会特别慢，镜像仓库就是中央仓库的备份（镜像）

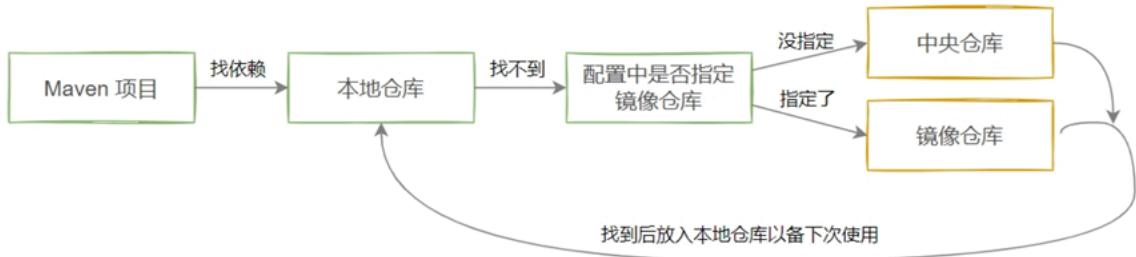
国内开发者使用阿里云镜像或者华为云镜像，这样可以大大提升中央仓库下载资源的速度，但它仍然是一个远程库

本地仓库(Local Repository)

使用当前执行maven构建的本机的仓库文件夹作为本地仓库

本地仓库指本机的一份拷贝，用来缓存从远程仓库下载的包，也包含尚未发布的临时构件

### Maven仓库使用流程



1. 首先，在本地repo搜索指定的依赖，而后是Maven的repo服务
2. 若配置了远程repo服务，Maven还是搜索该repo
3. Maven默认使用其官方存储库，在国内的下载速度较慢，因而建议使用国内的Maven镜像仓库

通常编译打包java项目会依赖很多的资源即包jar文件才能完成，所以需要通过Maven的资源坐标路径查找到相关包并下载使用

#### Local Repository 本地仓库说明

- Maven 的本地仓库存储了 maven 需要的插件和你项目中需要用到的依赖 jar 包，当 maven 使用到上述内容的时候，如果本地仓库中已经存在了，则直接使用本地仓库中内容，如果本地仓库中不存在，则需要从中央仓库下载到本地然后才可以继续使用，因此 maven 的本地仓库会存储很多的文件
- Maven的默认配置的本地仓库路径： \${HOME}/.m2/repository
- 可以在maven的配置文件 settings.xml 文 中指定本地仓库路径

```

<!-- localRepository
| The path to the local repository maven will use to store artifacts.
|
| Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->

```

## Maven安装和配置

### 包仓库安装

```

# 安装maven前必须安装java环境
apt install maven -y

# 修改默认仓库为国内源，在mirrors标签内添加阿里云配置
vim /etc/maven/setting.xml

<mirror>
    <id>aliyunmaven</id>
    <mirrorof>*</mirrorof>
    <name>阿里云公共仓库</name>
    <url>https://maven.aliyun.com/repository/public</url>
</mirror>

```

### 二进制安装

```

wget https://mirror.tuna.tsinghua.edu.cn/apache/maven/maven-
3/3.9.6/binaries/apache-maven-3.9.6-bin.zip

unzip apache-maven-3.9.6-bin.zip -d /usr/local

ln -s /usr/local/apache-maven-3.9.6/ /usr/local/maven

[root@ubuntu ~]# ls /usr/local/maven
bin  boot  conf  lib  LICENSE  NOTICE  README.txt
[root@ubuntu ~]# ls /usr/local/maven/bin/
m2.conf  mvn  mvn.cmd  mvnDebug  mvnDebug.cmd  mvnjar

```

## 主要配置文件

- 在maven中主要的配置文件有三个，分别是全局的setting.xml,用户的setting.xml,和每个项目单独的配置文件pox.xml
- 全局settings.xml是maven的全局配置文件，一般位于\$(maven.home)/conf/setting.xml,即maven文件目录下的conf中
- 用户setting是maven的用户配置文件，一般位于\$(user.home)/.m2/setting.xml，即每位用户都有一份配置文件
- pom.xml文件是项目配置文件，一般位于项目根目录后子目录下
- 配置优先级从高到低： pom.xml > 本地setting > 全局settings
- 如果这些文件同时存在，在应用配置时，会合并它们的内容，如果有重复的配置，优先级高的配置会覆盖优先级低的

```

/etc/maven/settings.xml          #包安装
/usr/local/maven/conf/settings.xml #源码安装

#localRepository 配置本地仓库目录，默认${user.home}/.m2/repository
<!-- localRepository
| The path to the local repository maven will use to store artifacts.
|
| Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->

#interactiveMode 是否需要和用户交换获得输入，默认 true
<!-- interactiveMode
| This will determine whether maven prompts you when it needs input. If set to
false, | maven will use a sensible default value, perhaps based on some other
setting, for
| the parameter in question.
|
| Default: true
<interactiveMode>true</interactiveMode>
-->

#offline 是否可以在没有网络的情况下进行编译，默认 false
<!-- offline
| Determines whether maven should attempt to connect to the network when
executing a build.
| This will have an effect on artifact downloads, artifact deployment, and

```

```
others.

|
| Default: false
<offline>false</offline>
-->

# server 配置私有仓库地址, 包括用户名, 密码等
<servers>
    <!-- server
        | Specifies the authentication information to use when connecting to a
        particular server, identified by
            | a unique name within the system (referred to by the 'id' attribute
            below).
        |
        | NOTE: You should either specify username/password OR
        privateKey/passphrase, since these pairings are
            | used together.
        |
    <server>
        <id>deploymentRepo</id>
        <username>repouser</username>
        <password>repopwd</password>
    </server>-->
    <!-- Another sample, using keys to authenticate.
    <server>
        <id>siteServer</id>
        <privateKey>/path/to/private/key</privateKey>
        <passphrase>optional; leave empty if not used.</passphrase>
    </server>-->
</servers>

# mirror 配置依赖仓库地址, 可以有多个仓库配置
<mirrors>
    <!-- mirror
        | Specifies a repository mirror site to use instead of a given repository.
The repository that
        | this mirror serves has an ID that matches the mirrorOf element of this
mirror. IDs are used
        |
    for inheritance and direct lookup purposes, and must be unique across the
set of mirrors.
    |
    <mirror>
        <id>mirrorId</id>
        <mirrorOf>repositoryId</mirrorOf>
        <name>Human Readable Name for this Mirror.</name>
        <url>http://my.repository.com/repo/path</url>
    </mirror>-->
    <mirror>
        <id>maven-default-http-blocker</id>
        <mirrorOf>external:http:</mirrorOf>
        <name>Pseudo repository to mirror external repositories initially using
HTTP.</name>
        <url>http://0.0.0.0/</url>
        <blocked>true</blocked>
    </mirror>
```

```
</mirrors>
```

- 阿里的镜像仓库

#打开 Maven 的配置文件(windows机器一般在maven安装目录的conf/settings.xml), 在<mirrors></mirrors>标签中添加 mirror 子节点:

```
<mirror>
    <id>aliyunmaven</id>
    <mirrorOf>*</mirrorOf>
    <name>阿里云公共仓库</name>
    <url>https://maven.aliyun.com/repository/public</url>
</mirror>
```

## Maven命令和编程流程

```
mvn [options] [<goal(s)>]
```

# 常用构建子命令

```
clean          # 清理之前已构建和编译的产物
compile        # 将源码编译成class字节码文件
test           # 行项目的单元测试
package         # 将项目打包成war包或jar包
install         # 将项目打包并安装到本地Maven仓库, 以便其他项目使用
clean install   # 先清理, 再编译安装
```

# 常用命令

```
mvn clean install package                  # 清理, 打包, 并安装
mvn clean install package -Dmaven.test.skip=true  # 清理, 打包, 并安装, 跳过单元
测试部分
mvn -T 4 clean install package -Dmaven.test.skip=true # 4线程编译
mvn -f mall-gateway clean package dockerfile:build      # 构建docker镜像, 要在
pom.xml中配置
```

## 实验示例

```
#安装git, git 用来clone 代码, 后面会讲
[root@ubuntu ~]# apt install git -y
#clone 代码到本地
[root@ubuntu ~]# git clone https://gitee.com/lbtooth/spring-boot-helloworld.git

# 进入项目目录, 因为需要读取pom.xml文件
cd spring-boot-helloworld

mvn clean package -Dmaven.test.skip=true

#多出一个target 目录,
[root@ubuntu spring-boot-helloworld]# ls
deploy Dockerfile Jenkinsfile LICENSE pom.xml README.md sonar
project.properties src target

#spring-boot-helloworld-0.9.0-SNAPSHOT.jar 就是构建好的目标
[root@ubuntu spring-boot-helloworld]# ls target/
classes generated-sources maven-archiver maven-status spring-boot-helloworld
```

```
0.9.0-SNAPSHOT.jar  spring-boot-helloworld-0.9.0-SNAPSHOT.jar.original

#依赖包都下载到本地了
[root@ubuntu ~]# ls .m2
repository
[root@ubuntu ~]# ls .m2/repository/
aopalliance  backport-util-concurrent  ch  classworlds  com  commons-codec
commons-io  io  javax  junit  net  org
[root@ubuntu ~]# du -sh .m2
52M .m2

# 执行
#运行, 根据输出内容, 可知PID为13735, 占用了 8080端口, 如果需要修改端口, 可加 --server.port=8888
[root@ubuntu spring-boot-helloworld]# cd
[root@ubuntu ~]# java -jar /data/spring-boot-helloworld-0.9.0-SNAPSHOT.jar

# 执行单元测试
mvn test

# 清理之前编译的文件
mvn clean
```

## 使用Nexus构建私有仓 (重点)

### Nexus介绍

Nexus是一个非常流行的开源软件项目，主要用于软件包管理和存储，它最初是由Sonatype开发的，旨在帮助开发人员和组织管理他们的软件构建，依赖关系和存储库，Nexus主要有两个版本：Nexus Repository 和 Nexus Firewall

### Nexus Repository Manager

Nexus Repository Manager 是一个用于管理软件构建和依赖关系的存储库管理器。它支持多种包类型，包括Maven、npm、Docker、RubyGems、Pypi等。以下是Nexus Repository Manager的主要功能和用途

- 包管理：开发人员可以使用Nexus Repository Manager来存储，管理和共享各种软件包和构建组件
- 代理远程存储库：Nexus可以代理远程存储库，使您能够在本地缓存外部依赖项，提高构建速度和稳定性
- 安全和权限：Nexus具有丰富的安全功能，可以管理用户访问权限，设置安全策略，保护组织的代码库免受潜在的安全漏洞
- 镜像和同步：Nexus支持镜像和同步功能，让您能够创建多个存储实例，并确保它们之间的数据同步

### Nexus安装和配置Maven仓库

#### 实验准备

- 设备1
  - IP: 10.0.0.150
  - 部署服务: java, Maven

- 备注：将Maven中的仓库配置指向自建的私有仓库
- 设备2
  - IP: 10.0.0.182 (重点)
  - 部署服务: Nexus Server
  - 内存需要4G以上，实现Maven的仓库，apt仓库，yum仓库

安装Nexus，并配置Maven仓库

```

# 安装jdk-11
dpkg -i jdk-11.0.23_linux-x64_bin.deb

wget https://download.sonatype.com/nexus/3/nexus-3.67.1-01-javal1-unix.tar.gz

tar xf nexus-3.67.1-01-javal1-unix.tar.gz -c /usr/local/

ln -sv /usr/local/nexus-3.67.1-01/ /usr/local/nexus

# 主配置文件，指定监听的端口号和IP
## DO NOT EDIT - CUSTOMIZATIONS BELONG IN $data-dir/etc/nexus.properties
##
# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-args=${jetty.etc}/jetty.xml,${jetty.etc}/jetty-http.xml,${jetty.etc}/jetty-requestlog.xml
nexus-context-path=/

# Nexus section
nexus-edition=nexus-pro-edition
nexus-features=\nexus-pro-feature

nexus.hazelcast.discovery.isEnabled=true

# 主要文件
[root@mystical /usr/local/nexus] $tree bin
bin
├── contrib
│   ├── karaf-service.sh
│   ├── karaf-service-template.conf
│   ├── karaf-service-template.init
│   ├── karaf-service-template.init-debian
│   ├── karaf-service-template.init-redhat
│   ├── karaf-service-template.solaris-smf
│   ├── karaf-service-template.systemd
│   ├── karaf-service-template.systemd-instances
│   ├── karaf-service-win.exe
│   └── karaf-service-win.xml
├── nexus          # 可执行二进制脚本文件，主程序
├── nexus.rc       # 配置运行时用户身份
└── nexus.vmoptions # 配置服务启动时的Java选项

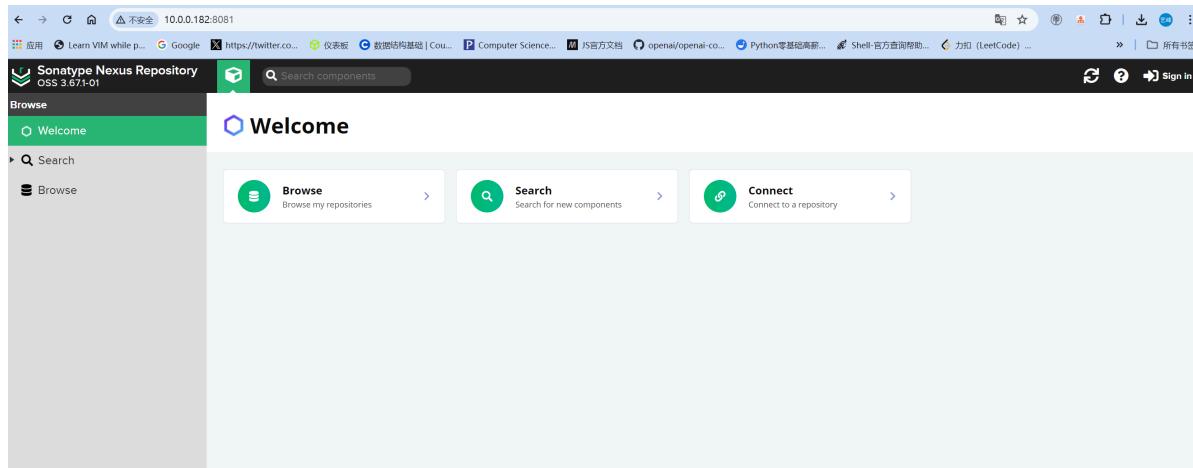
# 可以使用nexus.rc 指定执行程序的身份
run_as_user="root"

```

```
# 将二进制执行文件加入全局变量
ln -sv /usr/local/nexus/bin/nexus /usr/bin/

# 启用nexus

# 在浏览器访问10.0.0.182:8081
```

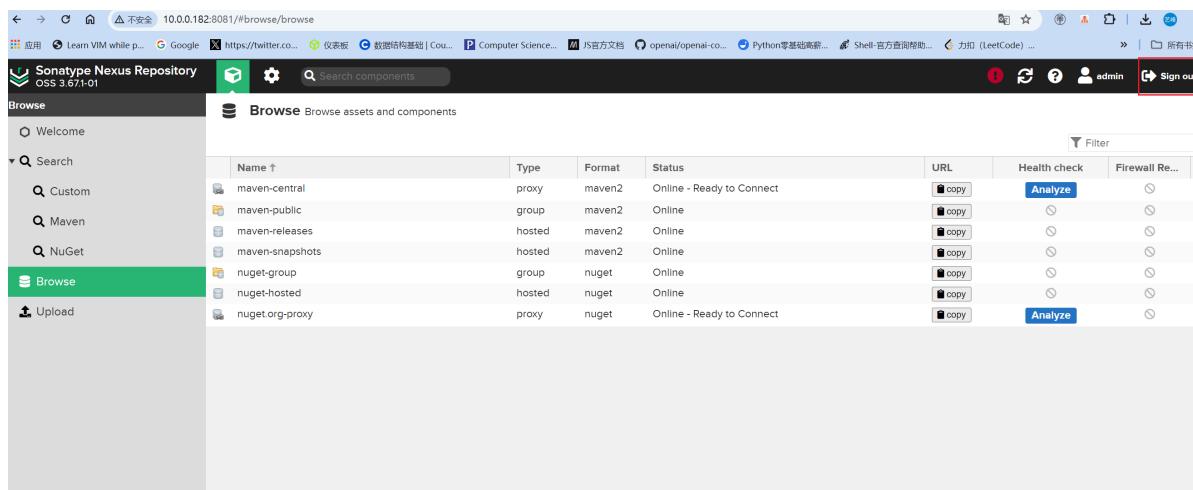


```
# 配置服务脚本，使用systemctl来进行服务管理
# 参考: https://help.sonatype.com/en/installation-and-upgrades.html
```

```
cat lib/systemd/system/nexus.service
[Unit]
Description=nexus service
After=network.target

[Service]
Type=forking
LimitNOFILE=65536
ExecStart=/usr/local/nexus/bin/nexus start
ExecStop=/usr/local/nexus/bin/nexus stop
User=root
#User=nexusvim v
Restart=on-abort

[Install]
WantedBy=multi-user.target
```



首次登录，用户名为admin; 密码在/usr/local/sonatype-work/nexus3/admin.password文件中

首次登录成功后，可以修改新的密码

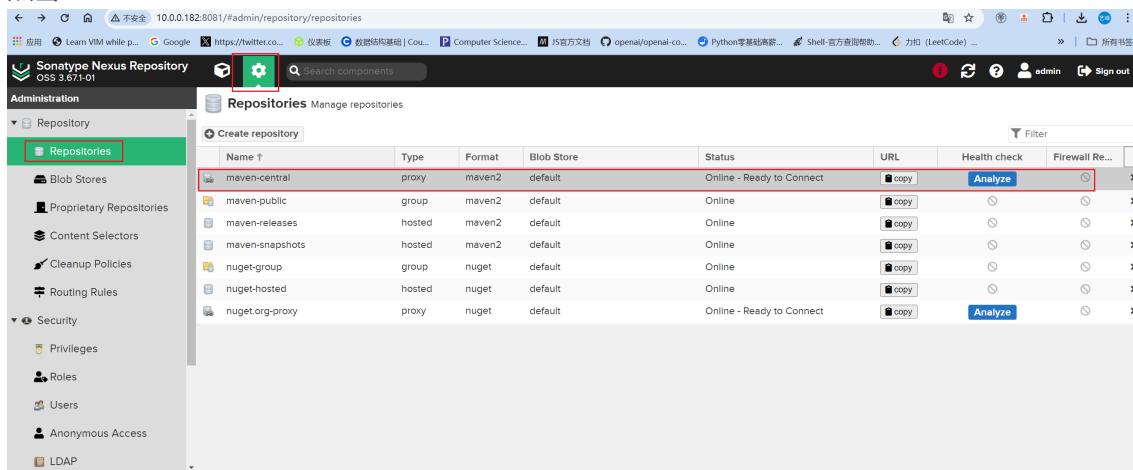
# 首次登录成功后修改密码，并选择支持匿名访问，做成公开库

```
# proxy 代理仓，如果nexus服务器上没有，则先去maven的官方仓库下载回来  
# hosted 本地仓，如果nexus服务器上没有，不会去公网下载  
# group 多个仓库集合，及支持maven仓库，yum和apt不能合并  
#           https://maven.aliyun.com 也有集合
```

# 配置maven-central的公网地址以为国内阿里云

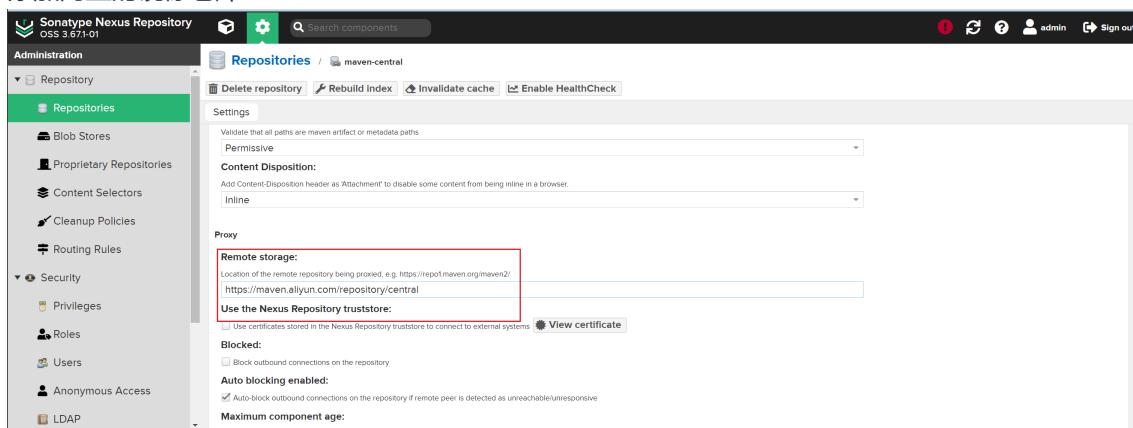
<https://maven.aliyun.com/repository/central>

- 点击



Name	Type	Format	Blob Store	Status	URL	Health check	Firewall Re...
maven-central	proxy	maven2	default	Online - Ready to Connect	<input type="button" value="copy"/>	Analyze	<input type="button" value=""/>
maven-public	group	maven2	default	Online	<input type="button" value="copy"/>	<input type="button" value=""/>	<input type="button" value=""/>
maven-releases	hosted	maven2	default	Online	<input type="button" value="copy"/>	<input type="button" value=""/>	<input type="button" value=""/>
maven-snapshots	hosted	maven2	default	Online	<input type="button" value="copy"/>	<input type="button" value=""/>	<input type="button" value=""/>
nuget-group	group	nuget	default	Online	<input type="button" value="copy"/>	<input type="button" value=""/>	<input type="button" value=""/>
nuget-hosted	hosted	nuget	default	Online	<input type="button" value="copy"/>	<input type="button" value=""/>	<input type="button" value=""/>
nuget.org-proxy	proxy	nuget	default	Online - Ready to Connect	<input type="button" value="copy"/>	Analyze	<input type="button" value=""/>

- 添加阿里的镜像仓库



Settings

Validate that all paths are maven artifact or metadata paths

Content Disposition:

Proxy

Remote storage:

Use the Nexus Repository truststore:

Use certificates stored in the Nexus Repository truststore to connect to external systems  View certificate

Blocked:

Auto blocking enabled:  Auto-block outbound connections on the repository if remote peer is detected as unreachable/unresponsive

Maximum component age:

在Maven主机上测试自建仓库(10.0.0.150)

```
cat /usr/local/maven/conf/settings.xml
```

```
<mirror>  
  <id>nexus-maven</id>  
  <mirrorOf>*</mirrorOf>  
  <name>自建仓库</name>  
  <url>http://10.0.0.182:8081/repository/maven-central/</url>  
</mirror>
```

```
#在nexus 服务器上，对应的包文件放在此目录中  
[root@ubuntu ~]# du -sh /usr/local/sonatype-work/nexus3/blobs/default/  
56M /usr/local/sonatype-work/nexus3/blobs/default/  
#但并不是可读的友好格式，这是基于安全方面的考虑，保有在 web 页面可见  
[root@ubuntu ~]# tree /usr/local/sonatype-work/nexus3/blobs/default/
```

## Nexus配置Apt仓库

1. 创建目录/data/blobs
2. 进入nexus配置界面，在Blob Stores页面点击Create Blob Store，创建一个存储，Type选择File，Name填Ubuntu2204，Path填/data/blobs/ubuntu2204，点击保存
3. 在Repositories页面点击Create repository,选择apt(proxy)，在新页面中Name填Ubuntu2204，Distribution填jammy,Remote storage 填<https://mirror.aliyun.com/ubuntu/>, Blob store 选择ubuntu2204, 点击保存
4. 在Browse页面选择刚创建的ubuntu2204，点击copy，复制原地址，然后修改一台ubuntu主机的/etc/apt/sources.list,将源指向自建的nextus服务器中的原地址
5. 测试

## Linux 防火墙

### netfilter基础

netfilter/iptables项目由Rusty Russes 创建于1998年，并于1999年建立了Netfilter Core Team,并在此后负责维护此项目，同时也于2000年3月合并进了Linux 2.3.x版本的Linux内核。

从Linux 2.4.x版本的内核开始，netfilter就正式成为了新一代的Linux防火墙机制，是Linux内核的一个子系统。Netfilter采用模块化设计，具有良好的可扩充性，提供扩展网络服务的结构化底层框架。Netfilter与IP协议栈是无缝契合，并允许对数据包进行过滤，地址转换，处理等操作。

Netfilter/iptables有三部分组成，分别是

- Netfilter框架
- iptables（内核空间）----- 根据规则过滤
- iptables命令行工具（用户空间。）----- 定义规则

内核中Netfilter相关的模块

```
[root@ubuntu ~]# grep -i netfilter /boot/config-5.15.0-73-generic  
CONFIG_NETFILTER=y  
CONFIG_NETFILTER_ADVANCED=y  
CONFIG_BRIDGE_NETFILTER=m  
# Core Netfilter Configuration  
CONFIG_NETFILTER_INGRESS=y  
CONFIG_NETFILTER_NETLINK=m  
CONFIG_NETFILTER_FAMILY_BRIDGE=y  
CONFIG_NETFILTER_FAMILY_ARP=y  
CONFIG_NETFILTER_NETLINK_HOOK=m  
CONFIG_NETFILTER_NETLINK_ACCT=m  
.....
```

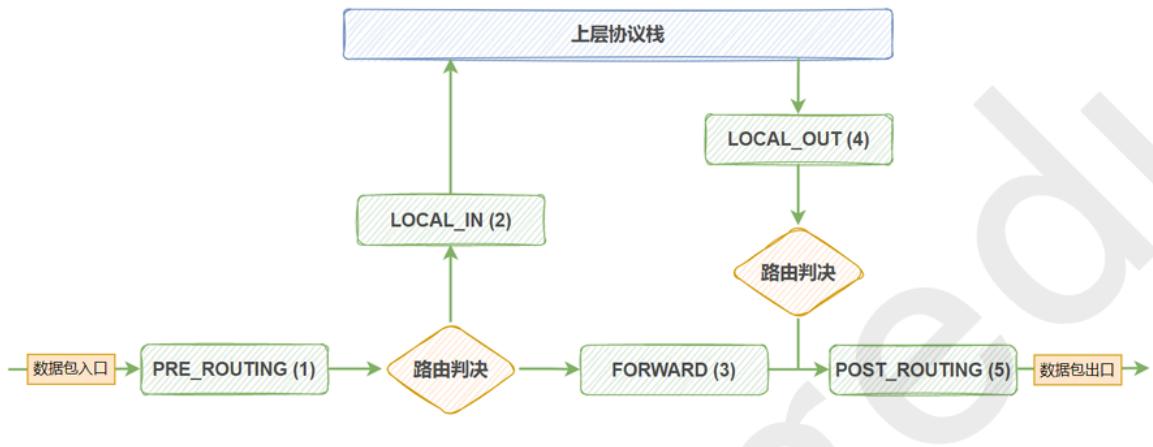
# 防火墙工具

## iptables (默认安装)

由软件包iptables提供的命令行工具，工作在用户空间，用来编写规则，写好的规则被送往Netfilter，告诉内核如何去处理信息包

## Netfilter工作原理

Netfilter在内核中某些位置放置了五个钩子函数 (hook function)，分别是INPUT, OUTPUT, FORWARD, PREROUTING, POSTROUTING，管理员可以通过配置工具配置规则，让这五个函数按照自定义的规则进行工作，达到防火墙的功能。



### 特别说明

从Linux kernel 4.2版以后，Netfilter在prerouting前加了一个ingress钩子函数。可以使用这个新的入口挂钩来过滤来自第二层的流量，这个新挂钩比预路由要早，基本上是tc命令(流量控制工具)的替代品

### 三种报文流行

- 流入本机
  - PREROUTING -----> INPUT -----> 用户空间进程
- 流出本机
  - 用户空间进程 -----> OUTPUT -----> POSTROUTING
- 从本机转发
  - PREROUTING -----> FORWARD -----> POSTROUTING

## iptables的组成

Netfilter在内核中设置了五个钩子函数，管理员在用户空间，通过程序设置过滤规则，内核中的钩子函数根据预设的规则进行工作，达到对流经的数据包进行过滤，拒绝，转发等功能

Iptables中有五个链(chain)对应内核中的五个钩子函数。除此之外，iptables中还有五个表(table)，一起来配合工作

### iptables中的链

- 类型：
  - 内置链：对应内核中的每个钩子函数  
(INPUT,OUTPUT,FORWARD,PREROUTING,POSTROUTING)

- 自定义链：用于对内置链进行扩展或补充，可实现更灵活的规则组织管理机制，只有HOOK钩子调用自定义链时，才会生效。

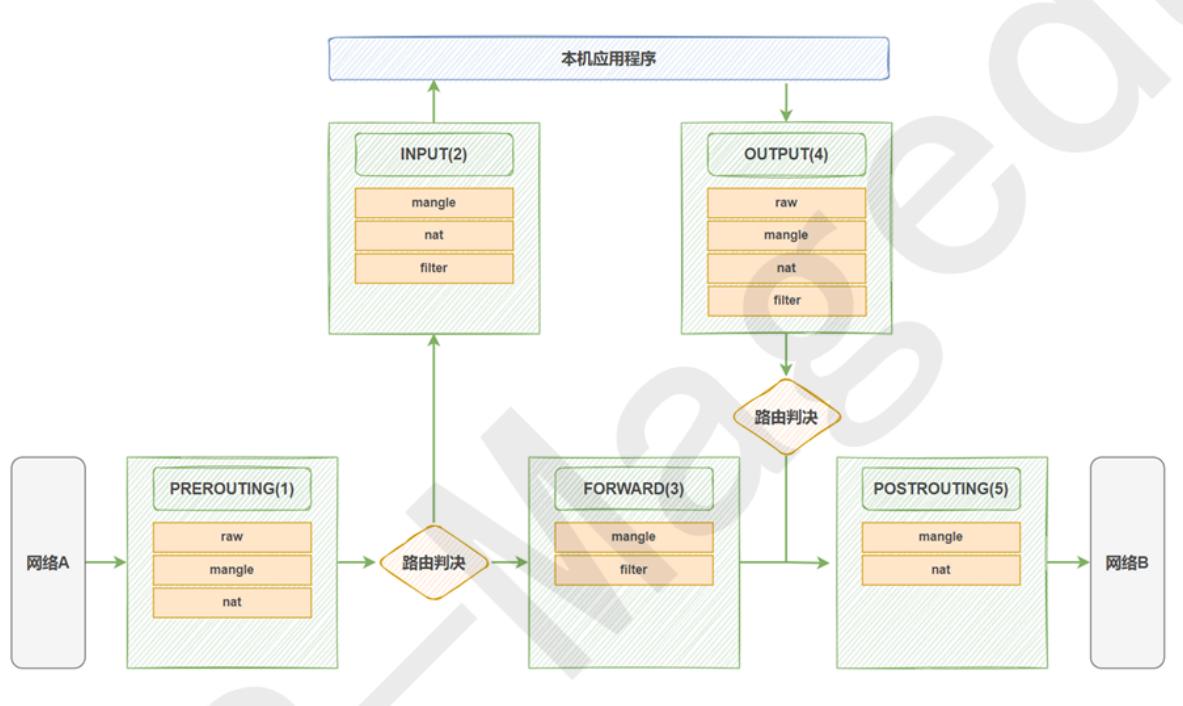
## Iptables中的表

- filter：过滤规则表，根据预定义的规则过滤符合条件的数据包，默认表
- nat：network address translation 地址转换规则表
- mangle：修改数据标记位规则表
- raw：关闭启用的连接跟踪机制，加快封包穿越防火墙速度
- security：用于强制访问控制，由Linux安全模块（SELinux）实现，很少使用

## 表的优先级

security ---> raw ---> mangle ---> nat ---> filter

## 链和表的对应关系（不是每个链都支持五个表）



# iptables

## 前置工作

在Ubuntu中，在用iptables命令设置防火墙之前，先关闭自带的ufw服务，因为ufw服务中也有一套自定义的规则，如果在开启ufw服务的情况下再用iptables设置或删除规则，容易发生冲突

```
systemctl is-active ufw
```

在redhat中要先停用firewalld服务

```
systemctl is-active firewalld
```

注意：如果不是最小化安装，在禁用firewalld之后，再次重启还是能看到防火墙规则，这是由KVM启动

卸载kvm功能

```
yum remove qemu-kvm
```

## iptables命令组成

### 完整格式实例

```
iptables -t filter -A INPUT -s 192.168.0.1 -j DROP

# 指定表
-t | --table table #ram|mangle|nat|filter,默认filter

# 操作链
-N | --new-chain chain          # 添加自定义新链
-X | --delete-chain [chain]     # 删除自定义链
-P | --policy chain target     # 设置默认策略,对filter表中的链而言,其默认策略有ACCEPT|DROP
-E | --rename-chain old-chain new-chain # 重命名自定义链,引用计数不为0的自定义链不能被重命名
-L | --list [chain]             # 列出链上的所有规则
-F | --flush [chain]            # 清空链上的所有规则,默认所有链
-Z | --zero [chain [rulenumber]] # 置0,清空计数器,默认操作所有链上的所有规则

# 操作具体规则
-A | --append chain rule-specification # 往链表上追加规则
-I | --insert chain [rulenumber]         # 往链上插入规则,可以指定变好,默认插入到最前面
-C | --check chain rule-specification   # 检查链上的规则是否正确
-D | --delete chain rulenumber          # 根据编号删除链上的规则
-R | --replace chain rulenumber new_rule # 根据链上的规则编号,使用新的规则替换原有规则

# 其他选项
--line-numbers                      # 显示每条规则编号
```

## 黑名单与白名单

```
# 创造白名单
# 指定ip访问
iptables -A INPUT -s 10.0.0.1 -j ACCEPT
# 不指定匹配条件,则全部拒绝
iptables -t filter -A INPUT -j REJECT
```

## 指定数据流入设备

```
iptables -t filter -I INPUT -i lo -j ACCEPT
```

## 规则替换

```
inputable -R INPUT 1 -s 127.1 -j ACCEPT
```

## 清空统计数据

```
iptables -Z INPUT 1
```

## 默认规则

```
iptables -P FORWARD DROP
```

注意：iptables规则只有root才能操作

## iptables匹配条件

iptables中的匹配条件分为基本匹配和扩展匹配两种，当一条iptables规则中有多个匹配条件时，默认是“与”，即要同时满足

### 基本匹配条件

```
[!] -s | --source address[/mask][...]      # 匹配源IP地址或网段  
[!] -d | --destination address[/mask][...]   # 匹配目标IP地址或网段  
[!] -p | --protocol protocol                 # 匹配具体协议，可以使用协议号或协议名，具体见/etc/protocol  
[!] -i | --in-interface name                  # 匹配报文流入接口，只用于数据流入  
[!] -o | --out-interface name                 # 匹配报文流出接口，只用于数据流出
```

### 匹配条件-扩展匹配

扩展匹配条件需要加载扩展模块才能使用

查看扩展模块

```
[root@ubuntu ~]# ls -l /usr/lib/x86_64-linux-gnu/xtables/*so | head-rw-r--r-- 1  
root root 14784 Mar 24 2022 /usr/lib/x86_64-linux  
gnu/xtables/libarpt_mangle.so-rw-r--r-- 1 root root 14776 Mar 24 2022  
/usr/lib/x86_64-linux  
gnu/xtables/libebt_802_3.so-rw-r--r-- 1 root root 14776 Mar 24 2022  
/usr/lib/x86_64-linux  
gnu/xtables/libebt_among.so-rw-r--r-- 1 root root 14784 Mar 24 2022  
/usr/lib/x86_64-linux  
gnu/xtables/libebt_arpreply.so-rw-r--r-- 1 root root 14776 Mar 24 2022  
/usr/lib/x86_64-linux  
gnu/xtables/libebt_arp.so-rw-r--r-- 1 root root 14784 Mar 24 2022  
/usr/lib/x86_64-linux  
gnu/xtables/libebt_dnat.so  
.....  
.....
```

扩展匹配条件分为隐式扩展和显示扩展两种

隐式扩展

Iptables在使用-p选项指明了特定的协议时，无需再用-m选项指明扩展模块的扩展机制，不需要手动加载模块，不需要用-m显示指定的扩展，即隐式扩展

tcp,udp,icmp这三个协议是可以用-m指定的模块，但同时，也可以在基本匹配里面用-p来指定这几个协议

范例

```
iptables -t filter -A INPUT -s 10.0.0.150 -d 10.0.0.110 -p tcp --dport 21:23 -j REJECT
```

根据TCP标志位过滤

```
iptables -t filter -A INPUT -s 10.0.0.150 -d 10.0.0.110 -p tcp --tcp-flags SYN,ACK,FIN,RST SYN -j REJECT # 检查SYN,ACK,FIN,RST四个标志位，其中SYN值为1，其它值为0，表示第一次握手  
# --tcp-flags SYN,ACK,FIN,RST SYN,ACK # 第二次握手  
# [!] --syn 第一次握手简单写法
```

udp协议扩展选项

```
[!]--source-port|--sport port[:port] # 匹配报文的源端口或端口范围  
[!]--destination-port|--dport port [:port] # 匹配报文的目标端口或端口范围
```

icmp协议的扩展选项

```
[!] --icmp-type {type[/code] | typename}  
0/0|echo-reply # icmp应答  
8/0| echo-request # icmp请求  
  
# 范例：拒绝ICMP的请求包  
iptable -A INPUT -s 10.0.0.150 -p icmp --icmp-type 8 -j REJECT]
```

## 扩展：基于协议的扩展选项总结

```
# 1. tcp协议扩展选项  
-p tcp --sport port[:port] # port:port表范围，示例：21:23，表示21,22,23  
-p tcp --dport port[:port]  
-p tcp -m multiport --sports|--dports num,num... # 同时指定离散的端口  
## 示例  
iptables -A INPUT -p tcp -m multiport --dports 21,23 -j REJECT  
  
-p tcp --tcp-flags SYN,ACK,FIN,RST [SYN,ACK,FIN,RST] # 如果相应标志位是1，则在后面写标志位名称  
## 示例  
iptables -t filter -A INPUT -s 10.0.0.150 -d 10.0.0.110 -p tcp --tcp-flags SYN,ACK,FIN,RST SYN -j REJECT # --tcp-flags SYN,ACK,FIN,RST SYN表示第一次握手
```

## 显示扩展及其相关模块

显示扩展即必须使用-m选项指明要调用的扩展模块名称，需要手动加载扩展模块

格式

```
-m matchname [per-match-options]
```

## multiport扩展

以离散方式定义多个端口匹配，最多指定15个端口

```
[!] --source-ports | --sports port[,port|,port]...
[!] --destination-ports|--dports port [,port|,port]...
[!] --ports port[,port|,port:port]...
```

示例

```
ip tables -A INPUT -s 10.0.0.150 -p tcp -m multiport --dports 22,80 -j REJECT
```

## iprange扩展

指明连续的（但一般不是整个网络）ip地址范围

```
[!] --src-range from[-to]      # 源地址范围
[!] --dst-range from[-to]      # 目标IP地址范围
```

范例

```
拒绝10.0.0.9-10.0.0.99 这91个IP
iptables -A INPUT -m iprange --src-range 10.0.0.9-10.0.0.99 -j DROP
```

## 修改MAC地址

mac模块可以指明源MAC地址，适用于：PREOUTING, FORWARD, INPUT chains

```
# 仅有mac地址是00:0c:29:f3:44:9a 的主机才能访问
]# iptables -t filter -A INPUT -d 10.0.0.110 -m mac --mac-source
00:0c:29:f3:44:9a -j ACCEPT
```

## string扩展

对报文中的应用层数据做字符串模式匹配测试

```
--algo {bm|kmp}          #过滤算法 Boyer-Moore|Knuth-Pratt-Morris, 在Rocky8.6中bm算法配合nginx有问题
--from offset             #开始偏移量，一般用来跳过包头，只检查具体内容，这样能提高效率

--to offset                #结束偏移量
[!] --string pattern      #具体要匹配的字符串
[!] --hex-string pattern  #具体要匹配的字符串，16进制格式
```

范例

```
#设置出口规则，在返回的数据包中，跳过前62字节的报文头，如果内容中出现 google，则拒绝返回
iptables -t filter -A OUTPUT -m string --algo kmp --from 62 --string "google" -j
REJECT
```

## time扩展

根据将报文到达的时间与指定的时间范围进行匹配，此扩展中默认使用UTC时间，所以使用时，要根据当前时区进行计算。

此扩展在CentOS8中有问题

```
--datestart YYYY[-MM[-DD[Thh[:mm:ss]]]]]      # 开始生效时间
--datestop YYYY[-MM[-DD[Thh[:mm:ss]]]]]       # 结束生效时间
--timestart hh:mm[:ss]                          # 开始生效时间
--timestamp hh:mm[:ss]                          # 结束生效时间
[!] --monthdays day[,day...]                  # 指定几号, 1-31
[!] --weekdays day[,day...]                   # 指定星期几, 可以写
Mon|Tue|Wed|Thu|Fri|Sat|Sun, 也可以写1|2|3|4|5|6|7
--contiguous
--kerneltz                                     # 内核时间
```

范例

```
iptables -t filter -A INPUT -m time --timestart 01:00 --timestamp
14:00 -s 10.0.0.150 -j REJECT
```

## connlimit扩展

根据每客户端IP做并发连接数数量匹配，可以防止DOS攻击

```
--connlimit-upto N                      # 连接的数量小于等于N时匹配
--connlimit-above N                      # 连接的数量大于N时匹配
```

限制并发连接

```
# 每台机器的最大并发数量不能超过10个
iptables -t filter -A INPUT -m connlimit --connlimit-above 10 -j REJECT
```

## limit扩展

基于收发报文的速率做匹配，令牌桶过滤器。

connlimit扩展是限制单个客户单对服务器的并发连接数，limit扩展是限制服务器上所有的连接数

```
--limit-burst number                    # 前多个包不限制
--limit [/second/minute/hour/day]      # 在一个时间区内能接收的连接数
```

范例

```
# 添加icmp方向规则，前10个不处理，后面每分钟放行20个
iptables -t filter -A INPUT -p icmp --icmp-type 8 -m limit --limit 20/minute --
limit-burst 10 -j ACCEPT
```

```
# 兜底规则
iptables -t filter -A INPUT -p icmp -j REJECT
```

## state扩展

state扩展，可以根据“连接追踪机制”去检查连接的状态，比较消耗资源

```
[!] --state state
```

state扩展状态

- 状态类型

- NEW (常用)：新发出请求；连接追踪信息库中不存在此连接的相关信息条目，因此，将其识别为第一次发出的请求
- ESTABLISHED (常用)：NEW状态之后，连接追踪信息库中为其建立的条目失效之前期间内所进行的通信状态
- RELATED：新发起的但与已有连接相关联的连接，如：ftp协议中的数据连接与命令之间的关系
- INVALID：无效的连接，如flag标记不正确
- UNTRACKED：未进行追踪的连接，如：raw表中关闭追踪

查看是否启用了连接追踪机制

```
#查看是否启用  
lsmod|grep nf_conntrack  
  
# 启用连接跟踪机制后，会自动生成此文件  
cat /proc/net/nf_conntrack
```

动态加载，只要系统中某处要使用到连接追踪机制，连接追踪机制相关模块会自动加载

示例：拒绝新连接

```
# 先建立连接，再设置防火墙规则  
64 bytes from 10.0.0.150: icmp_seq=383 ttl=64 time=0.477 ms  
64 bytes from 10.0.0.150: icmp_seq=384 ttl=64 time=0.900 ms  
64 bytes from 10.0.0.150: icmp_seq=385 ttl=64 time=1.03 ms  
  
# 已连接的客户端可以继续通信  
iptables -t filter -A INPUT -m state --state ESTABLISHED -j ACCEPT  
  
# 新的客户端拒绝连接  
iptables -t filter -A INPUT -m state --state NEW -j REJECT
```

再次查看模块加载和追踪文件

```
lsmod | grep nf_conntrack
nf_conntrack      172032  1  xt_conntrack
nf_defrag_ipv6    20480   1  nf_conntrack
nf_defrag_ipv4    20480   1  nf_conntrack
libcrc32c        16384   3  nf_conntrack, nf_tables, xfs

# 启用连接跟踪机制之后，连接状态时记录在文件中的
cat /proc/net/nf_conntrack
ipv4 2 icmp 1 29 src=10.0.0.206 dst=10.0.0.150 type=8 code=0 id = 1
src=10.0.0.150 dst=10.0.0.206 type=0 code=0 id=1 mark=0 zone=0 use=2
...
...
```

建立新链接测试，无法连接

```
#nf_conntrack文件中能记录多少内容，取决于配置
cat /proc/sys/net/netfilter/nf_conntrack_max
65536

# 将此参数改小，表示记录连接状态的文件，只能保存一条记录
echo 1 > /proc/sys/net/netfilter/nf_conntrack_max
cat /proc/sys/net/netfilter/nf_conntrack_max
1

# 查看系统日志，连接数超过了可跟踪记录的最大值，报文被抛弃
tail -3 /var/log/messages
Nov 16 08:50:22 rocky kernel: nf_conntrack: nf_conntrack: table full, dropping
packet
```

相关内核参数

```
# 连接追踪功能所能够容纳的最大连接数量，修改此处，下一项也会改变
/proc/sys/net/netfilter/nf_conntrack_max

# 连接追踪功能所能够容纳的最大连接数量，修改此处，上一项也会改变
/proc/sys/net/nf_conntrack_max

# 已追踪的连接数量
/proc/sys/net/netfilter/nf_conntrack_count

# 不同协议超时时长，单位为秒，超时后会踢出已监控的队列
/proc/sys/net/netfilter/*_timeout_*
```

面试题：有一台服务器无法建立新的连接，查看系统日志有如下提示，请问是什么问题，如何解决

```
Nov 16 08:50:22 rocky kernel: nf_conntrack: nf_conntrack: table full, dropping
packet
Nov 16 08:50:22 rocky kernel: nf_conntrack: nf_conntrack: table full, dropping
packet
Nov 16 08:50:22 rocky kernel: nf_conntrack: nf_conntrack: table full, dropping
packet
```

- 答：加大连接限制，降低nf\_conntrack\_timeout时间

```
vim /etc/sysctl.conf
net.nf_conntrack_max = 393216
net.netfilter.nf_conntrack_max = 393216
```

## target

target

- ACCEPT
  - 允许通过，命中此规则后，不再对比当前链的其他规则，进入下一个链
- DROP:
  - 丢弃，命中此规则后，该链接直接中断，发送方收不到任何回应
- REJECT:
  - 拒绝，命中此规则后，该链接直接中断，发送方能收到拒绝回应
- RETURN:
  - 返回，在当前链中不再继续匹配，返回到主链，一般用于自定义规则链中
- LOG:
  - 记录日志，将被命中的数据记录在日志中
- SNAT:
  - 源IP地址转换
- DNAT:
  - 目标IP地址转换
- REDIRECT
  - 本机端口转换
- MASQUERADE
  - 源IP地址动态转换
- 自定义

## 记录日志LOG

```
--log-level level      # 日志级别
--log-prefix prefix    # 日志前缀，用于区别不同的日志，最多29个字符
```

```
iptables -t filter -A INPUT -s 10.0.0.150 -j LOG --log-prefix "test-"

# 测试
ping 10.0.0.206 -c2

# 查看206主机日志
tail -2 /var/log/syslog
```

## iptables 规则保持

使用iptables命令定义的规则，手动删除之前，其生效期限为kernel存活期限，当系统重启之后，定义的规则会消失

### 持久化保存规则

```
# 设置一条防火墙规则  
iptables -t filter -A INPUT -s 10.0.0.1 -j ACCEPT  
  
# 在标准输出中显示  
iptables-save  
  
# 导出到文件  
iptables-save > iptables.rule
```

记载保存的规则

```
# 将文件中保持的防火墙规则，导入防火墙  
iptable-restore < ./iptables.rule
```

开机自动加载规则

```
vim /etc/rc.d/rc.local  
  
# 增加如下行  
iptables-restore > /root/iptables.rule  
  
# 保存后给文件增加执行权限，CentOS8中默认没有可执行权限  
chmod a+x /etc/rc.d/rc.local  
  
# Ubuntu中没有该文件，要自行创建  
vim /etc/rc.local  
#!/bin/bash  
  
iptables-restore > /root/iptables.rule  
  
chmod a+x /etc/rc.localr
```

方法2：在redhat系列中使用iptables-services实现

```
yum install -y iptables-services

systemctl status iptables

# 开启iptables服务
systemctl start iptables

# 开启服务后，会产生一系列规则，其规则保存在文件/etc/sysconfig/iptables中

# 将自定义的规则保存到/etc/sysconfig/iptables中，再将iptables服务设置成开机启动即可
Iptables-save > /etc/sysconfig/iptables

systemctl enable --now iptables
```

## 网路防火墙

### NAT网络转换原理

NAT: (Network Address Translation) 网络地址转换

局域网中的主机都是分配的私有IP地址，这些IP地址在互联网上是不可达的，局域网中的主机，在与互联网通讯时，要经过网络地址转换，去到互联网时，变成公网IP地址对外发送数据。服务器返回数据时，也是返回到这个公网地址，再经由网络地址转换返回给局域网中的主机。

一个局域网中的主机，想要访问互联网，在出口处，应该有一个公网可达的IP地址，应该能将局域网中的IP地址通过NAT转换成公网IP

NAT的分类

- SNAT (Source NAT)
  - 源地址转换，修改请求报文中的源IP地址
- DNAT (Destination NAT)
  - 目标地址转换，修改响应报文中的目标IP地址
- PNAT (Port Nat)
  - 端口转换，IP地址和端口都进行转换

### SNAT 实现源IP地址转换

SNAT: Source NAT: 源地址转换，基于nat表，工作在POSTROUTING链上。

具体是指将经过当前主机转发的请求报文的源IP地址转换成根据防火墙规则指定的IP地址

格式

```
iptables -t nat -A POSTROUTING -s LocalNet ! -d localNet -j SNAT --to-source ExtIP [--random]

# --to-source    转换成指定IP，或指定范围内的IP，端口可选
```

示例

```
iptables -t nat -A POSTROUTING -s 10.0.0.108 ! -d 10.0.0.108 -j SNAT --to-source
192.168.10.151
```

## MASQUERADE 实现源IP地址转换

如果我们内网的出口设备上有固定IP，则直接指定--to-sourceIP，没有任何问题。但是如果是使用拨号上网，出口网络设备上的IP地址会发生变化，这种情况下，我们的出口IP不能写出固定的

这种场景下，我们需要使用MASQUERADE进行地址转换，MASQUERADE可以从主机网卡上自动获取IP地址作为出口IP地址

格式

```
iptables -t nat -A POSTROUTING -s LocalNET ! -d locatNet -j MASQUERADE [--to-  
ports port]      # MASQUEADE可以从主机网卡上自动获取
```

## DNAT 实现目标与IP地址转换

在内网换环境中，使用私有IP地址的设备需要与互联网进行时，需要借助出口设备将原内部IP地址转换成公网可达的IP地址可以再次隐藏

DNAT：Destination NAT，目标地址转换，基于nat表，工作在PREROUTING.

```
iptales -t nat -A PREROUTING -d 192.168.10.123 -p tcp --dport 80 -j DNAT --to-  
destination 192.168.10.150:80
```

DNAT 只能将一条请求规则重定向到一台后端主机，无法实现负载均衡

## 扩展：为什么SNAT在postrouting上更改源地址，而DNAT在prerouting上更改目标地址

- SNAT更改源地址不会影响路由决策，而DNAT更改目标地址的IP会影响路由决策，因此要在PREROUTING链上进行更改

## REDIRECT 实现本机端口转换

REDIRECT：重定向，通过定义规则，将收到的数据包转发至同一主机的不同端口  
REDIRECT 功能无需开启内核ip\_forward转发

格式

```
iptables -t nat -A PREROUTING -d ExtIP -p tcp|udp -dport PORT -j REDIRECT --to-  
ports PORT
```

## iptables中的自定义链

iptables中除了系统自带的五个链之外，还可以自定义链，来实现讲规则进行分组，重复调用的目的

自定义链添加规则之后，要作为系统链的target与之关联，才能起到作用

```
# 添加自定义链
iptables -t filter -N web_chain

# 修改自定义链的名称
iptables -t filter -E web_chain WEB_CHAIN

# 向自定义链添加规则
iptables -t filter -A WEB_CHAIN -p tcp -m multiport --dports 80,443 -j ACCEPT

# 将自定义链挂到系统链上
iptables -t filter -I INPUT 2 -j WEB_CHAIN
```

在使用单独的自定义链的情况下，如果需要修改，则只需要修改链上的具体规则，而不用修改自定义链与系统链的关联关系

```
# 单独修改自定义链的规则
iptables -t filter -R WEB_CHAIN 1 -p tcp -m multiport --dports 80,443,8080 -j
ACCEPT

# 删除自定义链
iptables -t filter -X WEB_CHAIN

# 如果自定义链中还有规则，无法删除
```

## VPN服务OpenVPN

VPN技术网关通过对数据包的加密和数据包目标地址的转换实现远程访问。VPN可通过服务器、硬件、软件等多种方式实现

### OpenVPN实现

OpenVPN是Linux下开源的VPN应用，它提供了良好的性能和友好的用户GUI

OpenVPN是一个基于OpenSSL库的应用层VPN实现。和传统VPN相比，它的优点是简单易用。

OpenVPN提供了多种身份验证方式，用以确认参与连接双方的身份，包括

- 共享私钥
  - 最简单，但只能用于建立点对点的VPDN
- 基于PKI的第三方证书
  - 提供了最完善的功能，但是需要额外的精力去维护一个PKI证书体系
- 用户密码组合
  - 可以省略客户端证书，但是仍有一份服务器证书需要被用作加密

配置说明：

在阿里云购买三台主机，设置为同一个内网网段，模拟公司内网，其中一台添加一个公网IP，在该机器上部署OpenVPN服务，以便让不在公司内网的远程主机可以连进公司内网的目地

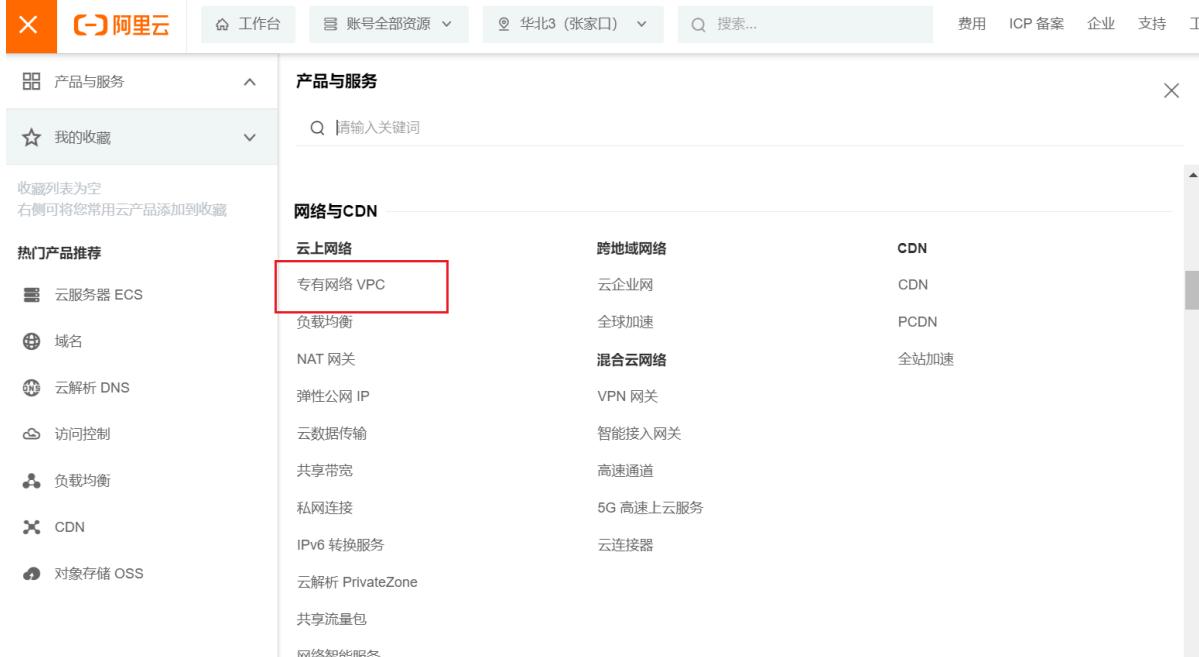
- 主机：
  - 192.168.0.101 --- 模拟马哥北京校区，配置OpenVPN Client，通过VPN网络以内网校色链接阿里云主机

- 172.30.0.66 --- 阿里云主机，配置OpenVPN Server，需要公网IP
- 172.30.0.100 --- 阿里云主机
- 172.30.0.200 --- 阿里云主机

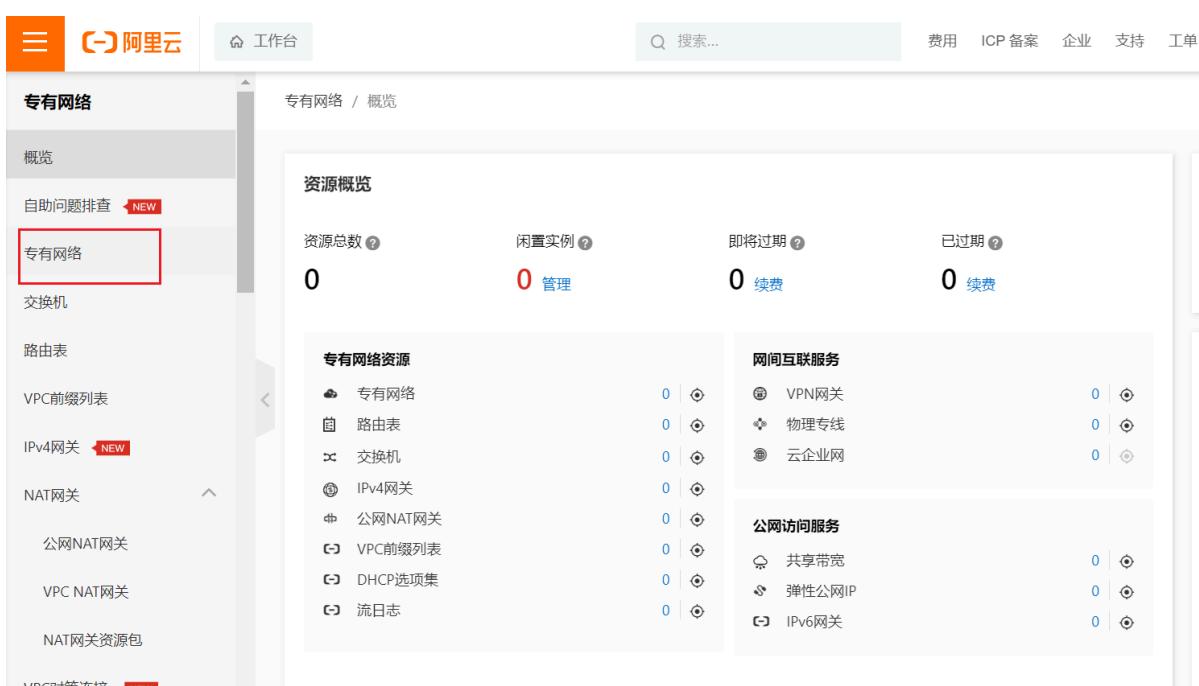
## 部署流程

### 准备阿里云主机，其中一台配置公网IP

- 在阿里云部署专有网络VPC



The screenshot shows the Alibaba Cloud Product Catalog interface. In the center, under the 'Network & CDN' section, the 'Exclusive Network VPC' option is highlighted with a red box. The left sidebar lists various services like ECS, DNS, and OSS.

The screenshot shows the Alibaba Cloud VPC console. The left sidebar has 'Exclusive Network' selected and highlighted with a red box. The main area displays resource statistics and a list of network resources.

阿里云

工作台 账号全部资源 华北3 (张家口) 搜索...

费用 ICP 备案 企业 支持 工单 简体

自助问题排查 NEW 什么是...

**专有网络**

- 概览
- 自助问题排查 NEW
- 专有网络**
- 交换机
- 路由表
- VPC前缀列表
- IPv4网关 NEW
- NAT网关
- 公网NAT网关
- VPC NAT网关
- NAT网关资源包
- VPC对等连接 NEW

**专有网络**

专有网络VPC (Virtual Private Cloud) 是用户基于阿里云创建的自定义私有网络，不同的专有网络之间二层逻辑隔离，用户可以在自己创建的专有网络内创建和管理云产品实例，比如ECS、负载均衡、RDS等。

**创建专有网络**

**核心优势**

**安全** 每个VPC都有一个独立的隧道号，一个隧道号对应着一个虚拟化网络。专有网络之间通过隧道ID进行隔离。

**可控** 您可以通过安全组规则、访问控制白名单等方式灵活地控制访问专有网络内云资源的出入流量。

(记得选择服务器所在地域，创建专有网络)

阿里云

工作台 搜索...

费用 ICP 备案 企业 支持 工单 简体

**专有网络**

- 概览
- 自助问题排查 NEW
- 专有网络**
- 交换机
- 路由表
- VPC前缀列表
- IPv4网关 NEW
- NAT网关
- 公网NAT网关
- VPC NAT网关
- NAT网关资源包
- VPC对等连接 NEW

**\* 专有网络**

\* 地域 华北3 (张家口)

名称 M58-VPC 随便写名字

\* IPv4网段 建议您使用RFC私网地址作为专有网络的网段 网段配置建议  
10.0.0.0/8 使用 172.16.0.0/12 使用 192.168.0.0/16 使用

172.30.0.0/16 填写专用网络网段

IPv6网段 不分配

描述

阿里云

工作台 搜索...

费用 ICP 备案 企业 支持 工单 简体 张艺

**专有网络**

- 概览
- 自助问题排查 NEW
- 专有网络**
- 交换机
- 路由表
- VPC前缀列表
- IPv4网关 NEW
- NAT网关
- 公网NAT网关
- VPC NAT网关
- NAT网关资源包
- VPC对等连接 NEW
- 终端节点
- 终端节点服务
- 高可用虚拟IP

**交换机**

新建专有网络/华北3 (张家口) 张家口 可用区A

交换机 172.30.0.0/24
-------------------

张北 可用区B

交换机 172.30.1.0/24
-------------------

**专有网络架构预览**

至少配置2个不同可用区的交换机 实现跨机房的高可用

阿里云

工作台 账号全部资源 华北3 (张家口) 搜索... 费用 ICP 备案 企业 支持 工单 简体 帮助文档 张艺峰6461 主题

云服务器 ECS 实例

实例

创建实例

自动识别 选择实例属性项搜索/输入关键字识别搜索 搜索 标签筛选 不分组 高级搜索 按钮

实例 ID / 名称 状态 标签 操作系统 监控 可用区 配置 操作

当前选择地域: 华北3 (张家口) 没有查询到符合条件的记录  
您可以尝试以下操作:  
· 查看当前选择的地域是否符合预期  
· 通过 全局搜索 查看全局资源

创建完VPC后, 创建ECS实例

我的常用 实例与镜像 实例 镜像 网络与安全 安全组 弹性网卡 密钥对 存储与快照 云盘 快照

云服务器 ECS 快速购买 自定义购买

欢迎您, 免费试用云服务器ECS, 个人用户可免费试用3个月, 每月200元免费额度 [立即试用](#)

付费类型: 包年包月 先付费后使用, 价格优惠 按量付费 先使用后付费, 按需开通

按量付费 实例不支持备案服务

地域: 华北3 (张家口)

使用须知: 实例创建之后地域将无法更改, 不同地域的实例之间内网互不通; 距离实例所在地域越近, 对实例访问速度越快

网络及可用区: M58-VPC / vpc-8vbmmsf9mmvm26iyodh1w 华北3 可用区 A | SW-1 / vsw-8vb97ervcrdlwlnzju6ty | 网段: 172.30.0.0/24

如何选择可用区: 创建专享网络 创建交换机

指定主网卡私网IP地址

批量创建实例时, 指定的私有IP地址为起始IP地址。

配置概要: 保存为启动模板

配置概要: 付费类型 按量付费 地域 华北3 (张家口) 可用区 华北3 可用区 A 网络类型 专有网络 带宽 M58-VPC / vpc-8vbmmsf9mmvm26iyodh1w 交换机 SW-1 / vsw-8vb97ervcrdlwlnzju6ty 实例规格 经济型 e / ecs.e-c1m1.large (2 vCPU 2 GB) 热备 Ubuntu 22.04.6 (64位/多个镜像)

购买实例数量: 3

最小购买数量: 3

使用时限: 设置自动释放服务时间 配置费用: ¥ 0.34761 /时

查看明细 [《云服务器 ECS 服务条款》](#)

确认下单

云服务器 ECS 安全组

安全组

注意: 根据《中华人民共和国电信条例》及《电信业务经营许可管理办法》, 开展递归解析服务需要具备相应的增值电信业务经营许可证, [查看详情](#)

创建安全组 智能匹配 搜索框 标签筛选 自定义安全组

安全组ID/名称 标签 网络 IP占用数/配额 安全组类型 创建时间 描述 操作

当前选择地域: 华北3 (张家口) 没有查询到符合条件的资源  
您可以尝试以下操作:  
· 点击 [创建安全组](#)  
· 通过 [全局搜索](#) 查看全局资源

购买前记得回ECS, 创建安全组

我的常用 实例与镜像 实例 镜像 网络与安全 安全组 弹性网卡 密钥对 存储与快照 云盘 快照 部署与弹性 弹性伸缩 节省计划

阿里云

工作台 账号全部资源 华北3 (张家口) 搜索... 费用 ICP 备案 企业 支持 工单 简体 张艺峰646130 主账号

### 云服务器 ECS

入方向 出方向 手动添加 快速添加

授权策略	优先级	协议类型	端口范围	授权对象	描述	操作
允许	1	自定义 TCP	* 目的: (HTTP (80))	* 源: 0.0.0.0/0	0.0.0.0/0将允许或拒绝所有IP的访问, 设置时请务必谨慎	<a href="#">复制</a> <a href="#">删除</a>
允许	1	自定义 TCP	* 目的: (HTTPS (443))	* 源: 0.0.0.0/0	0.0.0.0/0将允许或拒绝所有IP的访问, 设置时请务必谨慎	<a href="#">复制</a> <a href="#">删除</a>
允许	1	自定义 TCP	* 目的: (SSH (22))	* 源: 0.0.0.0/0	0.0.0.0/0将允许或拒绝所有IP的访问, 设置时请务必谨慎	<a href="#">复制</a> <a href="#">删除</a>
允许	1	自定义 TCP	* 目的: (RDP (3389))	* 源: 0.0.0.0/0	0.0.0.0/0将允许或拒绝所有IP的访问, 设置时请务必谨慎	<a href="#">复制</a> <a href="#">删除</a>
允许	1	全部 ICMP(IPv4)	* 目的: (-1/-1)	* 源: 0.0.0.0/0	0.0.0.0/0将允许或拒绝所有IP的访问, 设置时请务必谨慎	<a href="#">复制</a> <a href="#">删除</a>

[创建安全组](#) 规则预览 取消创建 将上述端口放行后 (这是个白名单), 点击创建,

返回购买界面，刷新后选择配置，操作系统，充钱购买即可

阿里云

工作台 账号全部资源 华北3 (张家口) 搜索... 费用 ICP 备案 企业 支持 工单 简体 张艺峰646130 主账号

### 云服务器 ECS / 实例

实例

创建实例 ...

自动识别 选择实例属性项搜索/输入关键字识别搜索 搜索 标签筛选 不分组

实例 ID: i-8vb4und6ewgr6wj8flm4, i-8vb4und6ewgr6wj8flm5, i-8vb4und6ewgr6wj8flm6

实例 ID / 名称	状态	标签	操作系统	监控	可用区	配置	IP 地址	操作
i-8vb4und6ewgr6wj8flm4 launch-advisor-20240528	运行中				华北3 (张家口) A	2核(vCPU) 2 GiB 0 Mbps ecs.e-c1m1.large	172.30.0.121 (私有)	<a href="#">远程连接</a> <a href="#">资源变配</a> <a href="#">停止</a>
i-8vb4und6ewgr6wj8flm6 launch-advisor-20240528	运行中				华北3 (张家口) A	2核(vCPU) 2 GiB 0 Mbps ecs.e-c1m1.large	172.30.0.120 (私有)	<a href="#">远程连接</a> <a href="#">资源变配</a> <a href="#">停止</a>
i-8vb4und6ewgr6wj8flm5 launch-advisor-20240528	运行中				华北3 (张家口) A	2核(vCPU) 2 GiB 0 Mbps ecs.e-c1m1.large	172.30.0.122 (私有)	<a href="#">远程连接</a> <a href="#">资源变配</a> <a href="#">停止</a>

阿里云

工作台 账号全部资源 华北3 (张家口) 搜索... 费用 ICP 备案 企业 支持 工单 简体 张艺峰646130 主账号

### 云服务器 ECS / 实例

实例

停止实例 ②

您所选的 3个实例 将执行停止操作, 您是否确定操作?

i-8vb4und6ewgr6wj8flm4 / launch-advisor-20240528  
i-8vb4und6ewgr6wj8flm6 / launch-advisor-20240528  
i-8vb4und6ewgr6wj8flm5 / launch-advisor-20240528

停止  强制停止

停止模式

普通停机模式  
停止实例后保留实例的资源并继续收费  
如果您停止实例是为了更换操作系统、重新初始化云盘、更改实例规格、修改私网 IP 等操作, 建议您选择该模式, 以避免启动失败

节省停机模式 (原停机不收费)  
计算资源(CPU和内存)、镜像License费用、固定公网IP的固定带宽模式暂停计费  
系统盘、数据盘、弹性公网 IP 的固定带宽模式持续计费

**注意项:** 由于计算资源被回收, 启动时可能因库存不足导致启动失败, 请您稍后重试或更换实例规格。如果实例停止前绑定了弹性公网 IP 地址, 重启后 IP 地址不变; 否则, 固定公网 IP 地址可能会变, 但私网 IP 地址不变。

确定 取消 共 3 条 < 1 > 20 条/页

阿里云

工作台 账号全部资源 华北3 (张家口)

搜索... 费用 ICP 备案 企业 支持 工单 帮助 简体

云服务器 ECS 实例

实例

创建实例 ...

搜索框：自动识别 选择实例属性项搜索/输入关键字识别 搜索

实例 ID : i-8vb4und6ewgr6wj8flm4-i-8vb4und6ewgr6wj8flm5-i-8v...

操作栏：高级搜索 标签筛选 不

实例 ID / 名称	状态	标签	操作系统	监控	可用区	配置	IP 地址	操作
i-8vb4und6ewgr6wj8flm4 launch-advisor-20240528	已停止 普通停机模式							
i-8vb4und6ewgr6wj8flm6 launch-advisor-20240528	已停止 普通停机模式							
i-8vb4und6ewgr6wj8flm5 launch-advisor-20240528	已停止 普通停机模式							

右侧悬浮菜单：ip 快速查找您想要的功能 网络和安全组 云盘与镜像  
高级公有IP 挂载云盘  
更换弹性公网IP 添加快照包日志云盘  
绑定弹性公网IP 重新初始化云盘  
修改私有IP  
管理辅助私网IP  
管理IPv6

**修改私有IP**

阿里云

工作台 账号全部资源 华北3 (张家口)

搜索... 费用 ICP 备案 企业 支持 工单 帮助 简体 张艺超

云服务器 ECS 实例

实例

修改私有IP

实例 i-8vb4und6ewgr6wj8flm4 可用区 cn-zhangjiakou-a

现私有IP 172.30.0.121 交换机 vsw-8vb97ervcrdlwlnzju6ty(172.30.0.24)

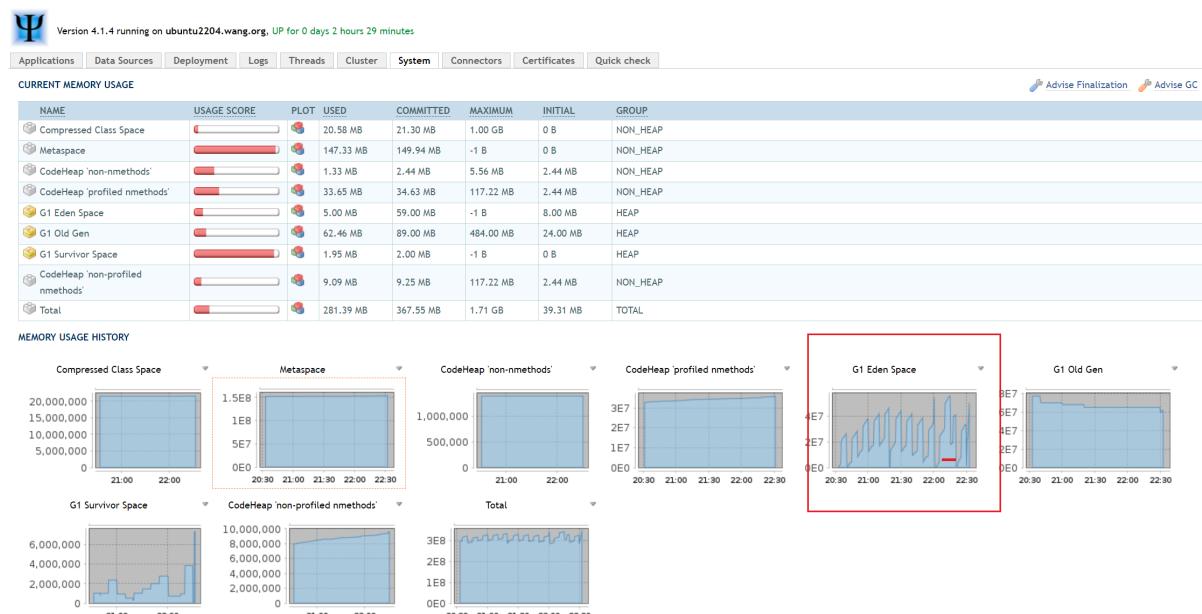
私有IP 172.30.0.66 确定 关闭

指定的私有IP必须为交换机网段中的未被占用的私有IP,如果不指定将自动为云服务器实例分配一个空闲的私有IP。

IP 地址	操作
172.30.0.121 (私有)	资源变配 启动   更换操作系统
172.30.0.120 (私有)	资源变配 启动   更换操作系统
172.30.0.122 (私有)	资源变配 启动   更换操作系统

10.0.0.202:8082/probe/memory.htm

应用 Learn VIM while p... Google https://twitter.co... 仪表板 数据结构基础 | Cou... Computer Science... JS 官方文档 openai/openai-co... Python 零基础编程 Shell 官方查询帮助... 力扣 (LeetCode)



将三台实例依次改为172.30.0.66, 172.30.0.100, 172.30.0.200

## 购买弹性公网IP

The screenshot shows the 'Create Elastic Public IP' page. On the left, there's a sidebar with network-related options like '专有网络', '交换机', '路由表', etc. The main area has a title '弹性公网IP' with a brief description: '独立的公网IP资源，可以绑定到阿里云专有网络VPC类型的ECS、NAT网关、私网负载均衡SLB上，并可以动态解绑，实现公网IP和ECS、NAT网关、SLB的解耦，满足灵活管理的要求。'. Below this is a red-bordered button labeled '创建弹性公网IP'. To the right of the button are '申请特定IP' and '刷新' buttons. On the far right, there's a vertical toolbar with icons for copy, edit, API, etc.

**Elastic Public IP Creation Page:**

- Left sidebar: 专有网络, 交换机, 路由表, VPC前缀列表, IPv4网关, NAT网关, 公网NAT网关, VPC NAT网关, NAT网关资源包, VPC对等连接, 终端节点, 终端节点服务, 高可用虚拟IP, DHCP选项集.
- Top right: 张艺峰646130, 简体, Google Translate.
- Main Content:
  - Title: 弹性公网IP
  - Description: 独立的公网IP资源，可以绑定到阿里云专有网络VPC类型的ECS、NAT网关、私网负载均衡SLB上，并可以动态解绑，实现公网IP和ECS、NAT网关、SLB的解耦，满足灵活管理的要求。
  - Buttons: 创建弹性公网IP (highlighted), 申请特定IP, 刷新.
  - Right toolbar: 复制, 编辑, API, 购买, 打印, 设置, 导出.

**Elastic Public IP Management Page:**

- Left sidebar: 专有网络, 交换机, 路由表, VPC前缀列表, IPv4网关, NAT网关, 公网NAT网关, VPC NAT网关, NAT网关资源包, VPC对等连接.
- Top right: 张艺峰646130, 简体, Google Translate.
- Main Content:
  - Title: 弹性公网IP
  - Info: “省钱神器”云数据传输已上线，开通后部分地域可以节省25%+公网流量费，开通请到CDN控制台。
  - Table: 列出已购买的弹性公网IP，包括实例ID/名称 (eip-8vbwzdc9jnsbxp83zrihy6), 安全防护, IP地址 (47.92.252.12), IP地址池ID, 监控, 诊断, 带宽 (200 Mbps), 操作 (发起诊断, 未加入, 绑定资源).
  - Bottom: 每页显示 20 条, 共 1 条, 上一页, 下一页, 导出, API, 购买, 打印, 设置.

## 购买后，回到ECS实例页面绑定IP

The screenshot shows the 'ECS Instances' management page. The left sidebar includes '我的常用' (My常用) under '实例与镜像', which is highlighted. The main area lists three stopped instances. A context menu is open over the first instance, showing options like '快速查找您想要的功能' (Quickly find the functions you want), '绑定弹性公网IP' (Bind Elastic Public IP), and other maintenance options. A search bar at the top is also visible.

**ECS Instance Management Page:**

- Left sidebar: 云服务器 ECS, 实例, 实例与镜像 (highlighted), 实例, 镜像, 网络与安全, 存储与快照, 部署与弹性, 节省计划.
- Top right: 张艺峰646130, 简体, Google Translate.
- Main Content:
  - Title: 云服务器 ECS / 实例
  - Table: 列出三台已停止的ECS实例 (i-8vb4und6ewgr6wj8flm4, i-8vb4und6ewgr6wj8flm6, i-8vb4und6ewgr6wj8flm5), 包括状态 (已停止), 标签, 操作系统, 监控, 可用区, 配置, IP地址, 操作 (启动, ...).
  - Context menu (over first instance):
    - 快速查找您想要的功能
    - 绑定弹性公网IP (highlighted)
    - 修改安全组
    - 实例快照回滚
    - 修改密钥对
    - 云盘与镜像
    - 更换云盘
    - 添加快照
    - 创建自定义镜像
    - 费用
    - 购买相同配置
    - 按量付费转包年包月
    - 为按量付费购买预留实例券
    - 查询消费明细
    - 运维与诊断
    - 修改实例维护属性
    - 实例问题排查历史
    - 查看实例回滚记录
  - Bottom: 启动, ..., 更多, 分页: 共 3 条, 1, 20 条/页.

阿里云

工作台 账号全部资源 华北3 (张家口)

搜索... 费用 ICP 备案 企业 支持 工单 高级搜索 简体 张艺群6461 主页

### 云服务器 ECS / 实例

**实例**

实例 ID / 名称	状态	标签	操作系统	监控	可用区	配置	IP 地址	操作
i-8vb4und6ewgr6wj8flm4 launch-advisor-20240528	已停止 普通停机模式							
i-8vb4und6ewgr6wj8flm6 launch-advisor-20240528	已停止 普通停机模式							
i-8vb4und6ewgr6wj8flm5 launch-advisor-20240528	已停止 普通停机模式							

请输入功能名称关键词

**快速查找您想要的功能**

- 网络和安全组
- 修改安全组
- 保存为自动模板
- 绑定密钥对
- 绑定辅助弹性网卡
- 云盘与镜像
- 挂载云盘
- 添加包年包月云盘
- 创建自定义镜像
- 费用
- 更改公网IP
- 公网IP转为弹性公网IP
- 绑定弹性公网IP
- 修改私有IP
- 更换交换机
- 管理辅助私网IP
- 更换操作系统
- 运维与诊断
- 修改实例维护属性
- 实例问题排查
- 实例问题排查历史
- 开始实例健康检查

共 3 条 < 1 > 20 条/页

阿里云

工作台 账号全部资源 华北3 (张家口)

搜索... 费用 ICP 备案 企业 支持 工单 高级搜索 简体 张艺群

### 云服务器 ECS / 实例

**实例**

**绑定弹性公网IP**

ECS 实例: launch-advisor-20240528

弹性公网 IP:  **确定** **关闭**

选择刚刚购买的ip, 点击确定

IP 地址	操作
172.30.0.66 (私有)	资源变配   启动   更换操作系统   ...
172.30.0.100 (私有)	资源变配   启动   更换操作系统   ...
172.30.0.200 (私有)	资源变配   启动   更换操作系统   ...

绑定后, 将停止的3台实例全部启动, 然后登录控制台

阿里云

工作台 账号全部资源 华北3 (张家口)

搜索... 费用 ICP 备案 企业 支持 工单 高级搜索 简体 张艺群64613 主页

### 云服务器 ECS / 实例

**实例**

实例 ID / 名称	状态	标签	操作系统	监控	可用区	配置	IP 地址	操作
i-8vb4und6ewgr6wj8flm4 launch-advisor-20240528	运行中				华北3 (张家口) A	2核(vCPU) 2 GiB 200 Mbps ecs.e-c1m1.large	47.92.252.12 (弹性) 172.30.0.66 (私有)	<b>远程连接</b> 资源变配   停止   ...
i-8vb4und6ewgr6wj8flm6 launch-advisor-20240528	运行中				华北3 (张家口) A	2核(vCPU) 2 GiB 0 Mbps ecs.e-c1m1.large	172.30.0.100 (私有)	<b>远程连接</b> 资源变配   停止   ...
i-8vb4und6ewgr6wj8flm5 launch-advisor-20240528	运行中				华北3 (张家口) A	2核(vCPU) 2 GiB 0 Mbps ecs.e-c1m1.large	172.30.0.200 (私有)	<b>远程连接</b> 资源变配   停止   ...

点击登录 ---> 输入密码，登录后台

```

文件 编辑 视图 实例 会话 功能 帮助 数据库
1. root@launch-advisor:~# ping 172.30.0.100
PING 172.30.0.100 (172.30.0.100) 56(84) bytes of data.
64 bytes from 172.30.0.100: icmp_seq=1 ttl=64 time=0.57 ms
64 bytes from 172.30.0.100: icmp_seq=2 ttl=64 time=0.351 ms
64 bytes from 172.30.0.100: icmp_seq=3 ttl=64 time=0.300 ms
...
-- 172.30.0.100 ping statistics --
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.300/0.402/0.557/0.111 ms
root@i28vb4und6ewgr6wj8flm4:~# ping 172.30.0.200
PING 172.30.0.200 (172.30.0.200) 56(84) bytes of data.
64 bytes from 172.30.0.200: icmp_seq=1 ttl=64 time=0.382 ms
64 bytes from 172.30.0.200: icmp_seq=2 ttl=64 time=0.282 ms
64 bytes from 172.30.0.200: icmp_seq=3 ttl=64 time=0.221 ms
...
-- 172.30.0.200 ping statistics --
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.202/0.268/0.382/0.080 ms
root@i28vb4und6ewgr6wj8flm4:~#

```

## 在阿里云安全组中放行1194端口

授权策略	优先级	协议类型	端口范围	授权对象	描述	操作
允许	1	自定义 TCP	1194	0.0.0.0/0		保存   预览   删除
允许	1	全部 ICMP(Pv4)	源: -1/-1 目的: -1/-1	源: 0.0.0.0/0 目的:		编辑   复制   删除
允许	1	自定义 TCP	目的: 3389/3389	源: 0.0.0.0/0		编辑   复制   删除
允许	1	自定义 TCP	目的: 22/22	源: 0.0.0.0/0		编辑   复制   删除
允许	1	自定义 TCP	目的: 443/443	源: 0.0.0.0/0		编辑   复制   删除

## 在有公网IP的阿里云主机上部署OpenVPN Server

### 在Magedu内网主机上部署OpenVPN Client

#### 测试OpenVPN网络

#### 配置OpenVPN证书

#### 安装相关软件

```
# ssh远程连接绑定公网ip的实例
# 安装OpenVPN，安装证书管理工具
apt update; apt install openvpn easy-rsa

root@vpn-server:~# dpkg -l openvpn
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-await/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name          Version         Architecture Description
++-----+
=====
ii  openvpn       2.5.9-0ubuntu0.22.04.2 amd64           virtual private network
daemon

root@vpn-server:~# dpkg -l easy-rsa
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/half-conf/Half-inst/trig-await/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name          Version         Architecture Description
++-----+
ii  easy-rsa      3.0.8-1ubuntu1 all            Simple shell based CA utility

root@vpn-server:~# tree /etc/openvpn
/etc/openvpn
├── client
├── server
└── update-resolv-conf

2 directories, 1 file
```

#### 配置CA证书

##### 准备证书环境

```
# 准备证书环境，让证书和openvpn配置处于一个目录，方便迁移
cp -r /usr/share/easy-rsa/ /etc/openvpn/

cd /etc/openvpn/easy-rsa

root@vpn-server:/etc/openvpn/easy-rsa# ls
easyrsa  openssl-easyrsa.cnf  vars.example  x509-types

mv vars.example vars
```

## 初始化证书目录

```
root@vpn-server:/etc/openvpn/easy-rsa# ./easyrsa init-pki

Note: using Easy-RSA configuration from: /etc/openvpn/easy-rsa/vars

init-pki complete; you may now create a CA or requests.
Your newly created PKI dir is: /etc/openvpn/easy-rsa/pki

# 查看目录
root@vpn-server:/etc/openvpn/easy-rsa# tree pki/
pki/
├── openssl-easyrsa.cnf
├── private
└── reqs
    └── safessl-easyrsa.cnf

2 directories, 2 files
```

自己给自己签发一个证书，使其称为CA机构

使用者发送证书申请文件给CA机构  
CA机构签发证书，返还给申请者  
申请者在自己的服务器上部署证书

## 生成CA机构证书

```
# 生成CA机构证书，不使用密码
./easyrsa build-ca nopass

Note: using Easy-RSA configuration from: /etc/openvpn/easy-rsa/vars
Using SSL: openssl OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Common Name (eg: your user, host, or server name) [Easy-RSA CA]: magedu      # 填写
CA机构名称

CA creation complete and you may now import and sign cert requests.
Your new CA certificate file for publishing is at:
/etc/openvpn/easy-rsa/pki/ca.crt

# 可以查看ca证书
./easyrsa show-ca
```

# 配置OpenVPN服务器证书

生成证书申请文件

审核申请文件，颁发机构证书

```
# 输入命令./easyrsa sign-req server server
root@vpn-server:/etc/openvpn/easy-rsa# ./easyrsa sign-req server server

Note: using Easy-RSA configuration from: /etc/openvpn/easy-rsa/vars
Using SSL: openssl OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)

You are about to sign the following certificate.
Please check over the details shown below for accuracy. Note that this request
has not been cryptographically verified. Please be sure it came from a trusted
source or that you have verified the request checksum with the sender.

Request subject, to be signed as a server certificate for 825 days:

subject=
commonName = server
```

```
Type the word 'yes' to continue, or any other input to abort.
Confirm request details: yes      # 这里填写yes
Using configuration from /etc/openvpn/easy-rsa/pki/easy-rsa-
3631.EFaQZY/tmp.kElqkB
801B1799217F0000:error:0700006C:configuration file routines:NCONF_get_string:no
value:../crypto/conf/conf_lib.c:315:group=<NULL> name=unique_subject
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName          :ASN.1 12:'server'
Certificate is to be certified until Aug 31 14:35:22 2026 GMT (825 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /etc/openvpn/easy-rsa/pki/issued/server.crt
```

## 创建Diffie-Hellman密钥

Diffie-Hellman密钥交换方法是迪菲(Whitefield Diffie)和赫尔曼(Martin Hellman)在1976年公布的一种密钥交换算法，他是一种建立密钥的方法，而不是加密方法，所以密钥必须和其他一种加密算法结合使用。这种密钥交换技术的目的在于使两个用户安全地交换一个密钥，用此密钥作为对称密钥来加密后续的报文传输。

## 配置OpenVPN客户端证书

```
# 修改配置文件，将证书有效期时长改为180天
vim vars

#set_var EASYRSA_CERT_EXPIRE      825
set_var EASYRSA_CERT_EXPIRE      180

# 生成客户证书申请文件
root@vpn-server:/etc/openvpn/easy-rsa# ./easyrsa gen-req tom nopass

Note: using Easy-RSA configuration from: /etc/openvpn/easy-rsa/vars
Using SSL: openssl OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
```

You are about to be asked to enter information that will be incorporated into your certificate request.

what you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value.

If you enter '.', the field will be left blank.

-----

Common Name (eg: your user, host, or server name) [tom]:

Keypair and certificate request completed. Your files are:

req: /etc/openvpn/easy-rsa/pki/reqs/tom.req

key: /etc/openvpn/easy-rsa/pki/private/tom.key

# 给tom签署证书

```
root@vpn-server:/etc/openvpn/easy-rsa# ./easyrsa sign-req client tom
```

Using SSL: openssl OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2)

Please check over the details shown below for accuracy.

Please check over the details shown below for accuracy. Note that this request has not been cryptographically verified. Please be sure it came from a trusted source or that you have verified the request checksum with the sender.

Request subject, to be signed as a client certificate for 180 days:

**subject=**

```
commonName          = tom

Type the word 'yes' to continue, or any other input to abort.
Confirm request details: yes
Using configuration from /etc/openvpn/easy-rsa/pki/easy-rsa-
3883.Zs8s4f/tmp.8HQ0jy
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName          :ASN.1 12:'tom'
Certificate is to be certified until Nov 24 14:46:26 2024 GMT (180 days)

Write out database with 1 new entries
Data Base Updated

Certificate created at: /etc/openvpn/easy-rsa/pki/issued/tom.crt
```

重复上述步骤，给jerry生成证书

```
./easyrsa gen-req jerry nopass

./easyrsa sign-req client jerry
```

## 整理目录

将openVPN服务端相关证书文件复制到server目录中

```
cp /etc/openvpn/easy-rsa/pki/ca.crt /etc/openvpn/server/
cp /etc/openvpn/easy-rsa/pki/issued/server.crt /etc/openvpn/server/
cp /etc/openvpn/easy-rsa/pki/private/server.key /etc/openvpn/server/
cp /etc/openvpn/easy-rsa/pki/dh.pem /etc/openvpn/server/
```

为客户端用户创建目录，并将相关文件放到指定目录

```
mkdir /etc/openvpn/client/{tom,jerry}

cp /etc/openvpn/easy-rsa/pki/ca.crt /etc/openvpn/client/tom/
cp /etc/openvpn/easy-rsa/pki/private/tom.key /etc/openvpn/client/tom/
cp /etc/openvpn/easy-rsa/pki/issued/tom.crt /etc/openvpn/client/tom/

cp /etc/openvpn/easy-rsa/pki/ca.crt /etc/openvpn/client/jerry/
cp /etc/openvpn/easy-rsa/pki/private/jerry.key /etc/openvpn/client/jerry/
cp /etc/openvpn/easy-rsa/pki/issued/jerry.crt /etc/openvpn/client/jerry/
```

## OpenVPN服务端配置

### 服务端配置文件说明

配置文件默认不存在，需要手动创建

```
cp /usr/share/doc/openvpn/examples/sample-config-files/server.conf /etc/openvpn/
```

## 设置服务端配置文件

```
mv server.conf server.conf.bak

# 新建配置文件
vim server.conf

port 1194
proto tcp
dev tun
ca /etc/openvpn/server/ca.crt
cert /etc/openvpn/server/server.crt
key /etc/openvpn/server/server.key
dh /etc/openvpn/server/dh.pem
server 10.8.0.0 255.255.255.0
push "route 172.30.0.0 255.255.255.0"
keepalive 10 120
cipher AES-256-CBC
compress lz4-v2
push "compress lz4-v2"
max-clients 2048
user root
group root
status /var/log/openvpn/openvpn-status.log
log-append /var/log/openvpn/openvpn.log
verb 3
mute 20
```

## 设置server服务脚本文件

```
systemctl enable --now openvpn.service

root@vpn-server:/etc/openvpn# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
qlen 1000
    link/ether 00:16:3e:0b:1e:c4 brd ff:ff:ff:ff:ff:ff
    altname enp0s5
    altname ens5
    inet 172.30.0.66/24 metric 100 brd 172.30.0.255 scope global dynamic eth0
        valid_lft 315354941sec preferred_lft 315354941sec
    inet6 fe80::216:3eff:fe0b:1ec4/64 scope link
        valid_lft forever preferred_lft forever
# 重启后生成了tun0网卡
3: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN group default qlen 500
    link/none
    inet 10.8.0.1 peer 10.8.0.2/32 scope global tun0
```

```
valid_lft forever preferred_lft forever
inet6 fe80::4b72:d43a:f28f:d8d6/64 scope link stable-privacy
valid_lft forever preferred_lft forever
```

## OpenVPN客户端配置

### 为不同用户设置配置文件

```
vim client/tom/client.ovpn
client
dev tun
proto tcp
remote 47.92.252.12 1194
resolv-retry infinite
nobind
#persist-key
#persist-tun
ca ca.crt
cert tom.crt
key tom.key
remote-cert-tls server
#tls-auth ta.key 1
cipher AES-256-CBC
verb 3
compress lz4-v2
```

## 部署OpenVPN客户端

部署Windows客户端

下载

```
https://openvpn.net/community-downloads/
```

将客户端配置下载到windows中，将相关文件放到 C:/Program Files/OpenVPN/config 目录下

```
cd client/
tar zcvf tom.tar.gz tom/
sz tom.tar.gz
```

将文件解压到 C:/Program Files/OpenVPN/config 目录下后，打开OpenVPN GUI软件，连接即可  
连接成功后，会有一个10.8.0.1的网卡，可以ping 172.30.0.66

## 配置后端主机转发

OpenVPN服务端开启ip\_forward转发

```
echo net.ipv4.ip_forward = 1 >> /etc/sysctl.conf
sysctl -p
```

在OpenVPN主机上设置SNAT转发，将从10.8.0.0/24网段主机请求的IP转换成本机(172.30.0.66)

```
iptables -t nat -A POSTROUTING -s 10.8.0.0/24 ! -d 10.8.0.0/24 -j MASQUERADE
```

## OpenVPN管理

OpenVPN的管理功能主要关于安全加强及客户端的证书管理，用户密码验证等

### 启用安全加强功能

服务端配置

```
openvpn --genkey --secret /etc/openvpn/server/ta.key

# 修改服务端配置文件
vim /etc/openvpn/server.conf
tls-auth /etc/openvpn/server/ta.key 0 # 增强此行，服务端后面要跟数字0，客户端后面数字是1

# 重启服务器，客户端会断开连接，小图标会变成黄色
```

客户端配置

```
# 将key文件下载到本地，转到C:\Program Files\OpenVPN\config目录下
sz /etc/openVPN/server/ta.key

# 修改客户端配置文件 C:\Program Files\OpenVPN\client.ovpn
# 新增一行
tls-auth ta.key 1

# 客户端重新连接即可
```

## LVS

### LVS与集群简介

#### 集群Cluster

Cluster: 集群，为解决某个特定问题将多台计算机组合起来形成的单个系统

Cluster分为三种类型

- LB: Load Balancing: 负载均衡，多个主机组成，每个主机只承担一部分访问请求
- HA: High Availability, 高可用，避免SPOF(single point of failure)
  - MTBF: Mean Time Between Failure 平均无故障时间
  - MTTR: Mean Time Restoration (repair)平均恢复前时间，故障时间

- SLA: 服务等级协议 (简称: SLA, 全程: Service level agreement) 。是指在一定开销下为保障服务的性能和可用性, 服务提供商与用户间定义的一种双方认可的协议。通常这个开销是驱动提供服务质量的主要因素。在常规的领域中, 总是设定3个9, 4个9, N个9等来进行表示, 当没有达到这种水平的时候, 就会有一些惩罚性措施, 而运维最主要的目标就是达成这种服务水平。
- 停机时间有分两种, 一种是计划内停机时间, 一种是计划外停机时间, 而运维则主要关注计划外停机时间
- HPC: High-performance computing。高性能

## 分布式系统

分布式常见应用

- 分布式用用-服务按功能拆分, 使用微服务
- 分布式静态资源-静态资源放在不同的存储集群上
- 分布式数据和存储-使用key-value缓存系统
- 分布式计算-对特殊业务使用分布式计算, 比如Hadoop集群

分布式存储: Ceph(升级版本NFS), GlusterFSS, FasstDFS, MogileFS

分布式计算: hadoop, Spark

## LB Cluster 负载均衡集群

### 按实现方式划分

- 硬件
  - F5 Big-IP
  - Citrix Netscaler
  - A10
- 软件
  - lvs: Linux Virtual Server, 阿里云四层SLB(Server Load Balance)使用
  - nginx: 支持七层调度, 阿里云七层SLB使用Tengine
  - haproxy: 支持七层调度
  - ats: Apache Traffic Server, yahoo捐助给apache
  - perlbal: Perl编写
  - pound

### 基于工作的协议层次划分

- 传输层(通用): DNAT和DPROT
  - LVS
  - Nginx:stream
  - haproxy: mode tcp
- 应用层(专用): 针对特定协议, 常称为proxy server
  - http: nginx; httpd; haproxy(mode http)
  - fastcgi: nginx, httpd, ...

- mysql: mysql-proxy, mycat...

## 负载均衡的会话保持

- session sticky: 同一用户调度固定服务器
  - Source IP: LVS sh算法
  - Cookie
- Session replication: 每台服务器拥有全部session
  - session multicast cluster
- session server: 专门的session服务器
  - Redis, Memcached

## HA 高可用集群实现

keepalived: vrrp协议

Ais: 应用接口规范

- heartbeat
- cman+gmanage(RHCS)
- coresync+pacemaker

## Linux Virtual Server

### LVS介绍

LVS: Linux Virtual Server, 负载均衡调度器, 章文嵩 (花名 正明) , 阿里的四层SLB(Server Load Balance)是基于LVS+keepalived实现

LVS是全球最流行的四层负载均衡开源软件, 由章文嵩博士 (当前阿里云产品技术负责人) 在1998年5月创立, 可以实现Linux平台的负载均衡

整个SLB系统由3部分组成: 四层负载均衡, 七层负载均衡和控制系统

- 四层负载均衡, 采用开源软件LVS(linux virtual server), 并根据云计算需求对其进行了定制化, 该技术已经在阿里巴巴内部业务全面上线应用2年多
- 七层负载均衡, 采用开源软件Tengine; 该技术已经在阿里巴巴内部业务全面上线应用3年多
- 控制系统, 用于配置和监控负载均衡系统

### LVS工作原理

V根据请求报文的目标和目标协议及端口将其调度转发至某RS, 根据调度算法来挑选RS。LVS是内核级功能, 工作在INPUT链的位置, 将发往INPUT的流量进行“处理”

查看内核是否收录lvs

```
grep -i ipvs /boot/config-5.15.0-52-generic

# 查看相关模块
[root@mystical ~] $modinfo ip_vs
filename:      /lib/modules/5.15.0-107-
generic/kernel/net/netfilter/ipvs/ip_vs.ko
license:       GPL
srcversion:    3B1741352B12452381D737A
```

```

depends:      nf_conntrack,nf_defrag_ipv6,libcrc32c
retpoline:    Y
intree:       Y
name:        ip_vs
vermagic:    5.15.0-107-generic SMP mod_unload modversions
sig_id:      PKCS#7
signer:      Build time autogenerated kernel key
sig_key:     66:36:BA:75:E1:C2:01:89:6B:68:62:8D:3F:0A:A5:21:B5:B3:F5:C9
sig_hashalgo: sha512
signature:   B0:89:B9:44:A4:D1:F5:FF:56:60:4B:69:17:E1:E8:33:91:84:F7:59:
             9F:77:BE:6A:B8:A0:4A:26:C3:F6:20:AD:68:FF:D0:A6:8B:B2:D5:EC:
             14:F3:B1:82:1F:00:F7:31:5B:A6:64:0C:3A:E9:1C:29:52:C3:B0:61:
             FF:02:DC:79:36:06:81:E3:60:0A:6A:E9:B0:33:96:78:50:71:74:5C:
             21:7F:0B:AD:31:5D:56:57:CD:24:52:17:8A:A5:98:01:02:70:B4:59:
             6B:1F:6A:3E:7E:69:FF:19:46:B8:52:89:12:96:3A:23:A7:C7:09:27:
             10:92:3F:54:6D:9E:14:79:A5:4B:5D:EF:D9:09:74:E4:7A:61:FD:CC:
             AF:67:CC:EE:AB:A6:B7:E1:C2:87:17:B9:48:4F:EE:40:1F:34:24:F8:
             02:16:CD:65:A3:79:76:04:4A:8B:23:AE:F6:E3:8F:76:A8:90:13:E9:
             BE:08:F2:99:6D:26:C3:9C:AF:D7:CD:26:BD:BA:E8:C2:41:FD:70:42:
             44:4F:45:AE:8C:D8:8F:A7:1C:8A:78:F7:28:D5:B2:4D:82:83:97:52:
             5D:13:81:E6:2D:B2:93:78:BD:17:33:B3:63:13:28:3F:71:E3:AD:82:
             CA:DF:21:52:42:B6:42:28:CF:49:B5:C3:2E:56:BB:6F:C8:43:39:57:
             46:25:37:CE:81:57:FA:75:44:AD:13:0D:6F:86:E9:22:26:DD:CE:07:
             D2:5A:0F:1B:54:40:62:48:14:98:9A:28:93:D4:D9:16:F0:B5:CB:CB:
             F4:56:BC:D1:F9:6F:A1:94:84:0C:FF:44:22:79:82:A6:1F:02:DA:DE:
             A7:F1:16:27:6C:67:9B:18:33:31:20:A1:98:35:3F:D6:68:88:5A:61:
             F5:F4:E7:8C:B8:9A:26:FD:5C:C5:F6:6B:ED:29:0E:51:84:06:0D:02:
             DA:9A:38:BE:4A:AE:00:74:92:27:22:D8:FB:EF:D3:DD:37:C1:97:B1:
             B9:D2:4A:A9:5C:E1:07:90:D9:E9:C2:3F:2F:72:2B:AF:C0:44:5A:70:
             6B:1A:72:B9:C0:A7:99:1A:B4:56:5D:16:AD:C3:B8:A6:C2:39:77:E7:
             22:47:78:F6:D0:D4:4D:DB:59:42:4F:F1:DA:1B:85:CE:34:6D:A0:EB:
             9E:7B:85:60:81:4C:DE:F1:81:4B:7B:75:73:24:80:53:A7:4D:9C:D1:
             57:95:AA:08:8D:3A:53:6E:BF:6D:38:49:E6:63:86:CE:A4:F7:88:1E:
             6F:EC:39:D5:F5:46:75:66:1D:FF:64:DD:1F:E6:19:CC:83:F0:92:08:
             90:AA:99:6C:96:76:00:A2:3F:5F:D5:58
parm:      conn_tab_bits:Set connections' hash size (int)

```

```
# 查看是否加载使用该模块
lsmod | grep ip
```

lvs是内核中加载的功能，但是要使用该功能，需要下载对应的管理工具，而不可能直接操作内核

lvs的管理工具就是 ipvsadm

## LVS集群类型中的术语

VS: Virtual Server, Director Server(DS), Dirpatcher(调度器), Load Balancer

- 外界用户访问的时候访问的是调度器 (lvs) ,但是调度器并不是真的服务器，上面并不承载服务，而仅仅负责负载转发的作用。所以叫vs，即虚拟服务器

RS: Real Server(lvs), upstream server(nginx), backend server(haproxy)

CIP: Client IP

VIP: Virtual Server IP VS外网的IP

DIP: Director IP VS内网的IP

RIP: Real server IP

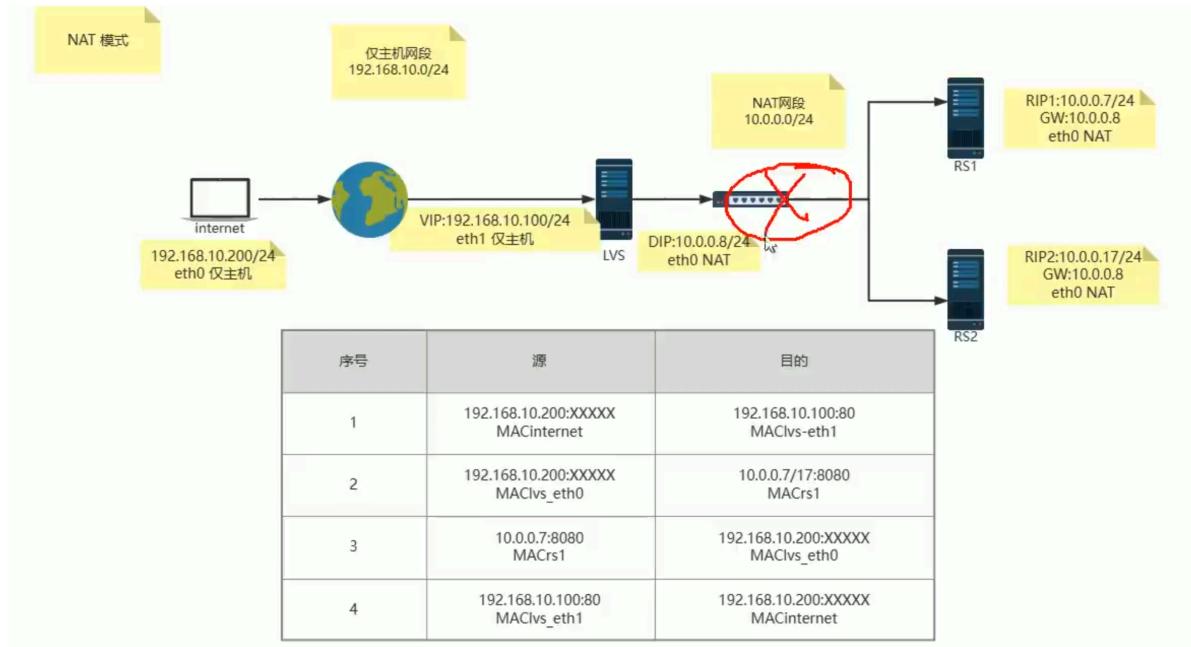
访问流程: CIP <--> VIP == DIP <--> RIP

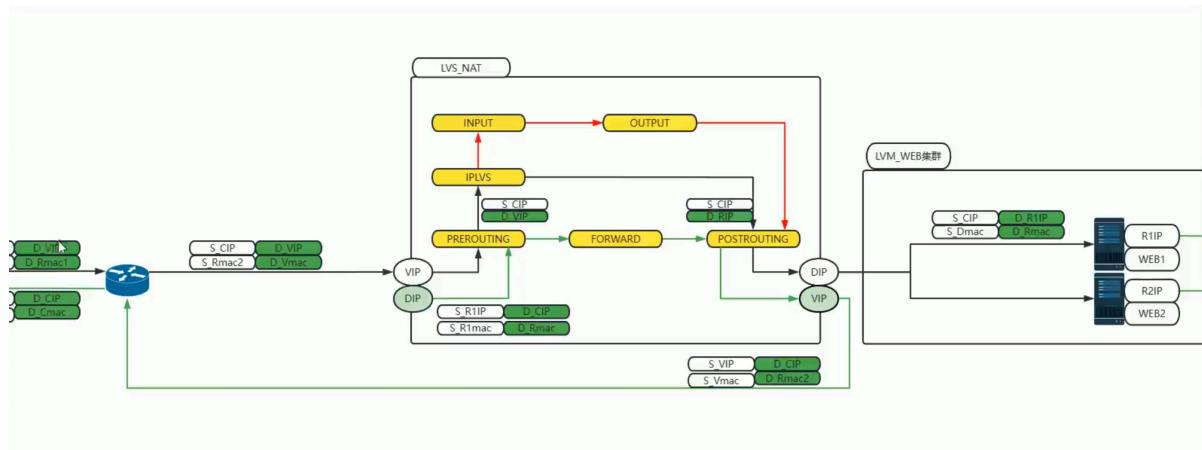
## LVS工作模式和相关命令

### LVS集群的工作模式

- lvs-nat
  - 修改请求报文的目标IP，多目标IP的DNAT
  - 就是用户请求，传输到lvs，lvs将目标地址替换为后端服务器地址，并传输过去，源地址不替换
  - 目标服务器收到后，原路返回，原路返回后再替换回去，也因此，目标服务器的网关必须是lvs保证数据包返回用户的时候，要经过lvs，将之前修改过的目标地址替换回来。
  - 这里一共两步操作
    - lvs上的ip\_forward,网卡之间的转发要开
    - 后端服务器的网关要指向lvs
  - nat模式，lvs和后端服务器的中间可以路由，但会影响性能
- lvs-dr
- 操纵封装新的MAC地址
- lvs-tun
- 在源请求IP报文之外新加一个IP首部
- lvs-fullnat
- 修改请求报文的源和目标IP，默认内核不支持

### LVS的NAT模式





### 数据包的传输过程

- 用户发请求 ---> 发送到LVS服务器
  - 数据包从网卡先经过PREROUTING
  - PREROUTING收到数据包后，查询路由表，看数据包中的地址是否是本机地址，如果是，就将数据包发给INPUT，由INPUT交由本地进程处理，如果不是就转到FORWARD链上
  - 但是INPUT后面并没有进程处理，所有的服务都是后端服务器提供了LVS本身不提供服务，所以不能让数据包进入INPUT链
  - 因此LVS必须做一件事，就是就截胡，在PREROUTING和INPUT中间加一个IPLVS，截住数据包
  - 然后将截取的数据包的目标地址，更改为后端的服务器地址和端口
  - 改完数据包地址后，从POSTROUTING发出去，传递给后端服务器
  - 后端服务器收到数据后，返回数据包，发给LVS的PREROUTING，然后由于返回数据包的目标地址是CIP，并不是LVS的vip,因此，不会向上走，而是直接查路由表，走 FORWARD 到 POSTROUTING (因为数据包在返回的时候，走的是PREROUTING->FORWARD->POSTROUTING的路径，所以lvs上必须开启IP\_forward的转发功能)
  - 又因为进来的时候，LVS更改了数据包的目的地址，改为了DIP的地址，现在要改回去，改回LVS自己的VIP地址，然后又LVS的源IP身份将数据包发回目标地址CIP

lvs-nat: 本质是多目标IP的DNAT，通过将请求报文中的目标地址和目标端口修改为某挑出的RS的RIP和PORT实现转发

- RIP和DIP应在同一IP网络，且应使用私网地址；RS的网关要指向DIP
- 请求报文和响应报文都必须经由DIRECTOR转发，Director容易称为系统瓶颈
- 支持端口映射，可修改请求报文的目标PORT
- VS必须是Linux系统，RS可以是任意OS系统
- LVS的DR模式

LVS-DR: Direct Routing, 直接路由，LVS默认模式，应用最广泛，通过为请求报文重新封装一个MAC头部进行转发，源MAC是DIP所在的接口的MAC，目标MAC是某挑选出的RS的RIP所在接口的MAC地址；源IP/PORT，以及目标IP/PORT均保持不变

前置问题：为什么多个设备配置一个IP会产生地址冲突

原因：如果多个服务器地址都是一样的，在arp协议中，通过arp广播，尝试将IP转换为MAC地址，结果一发广播，所有机器都响应arp，返回自己的地址，都返回自己的地址，但是客户端只能接受一个，就会产生懵逼效应

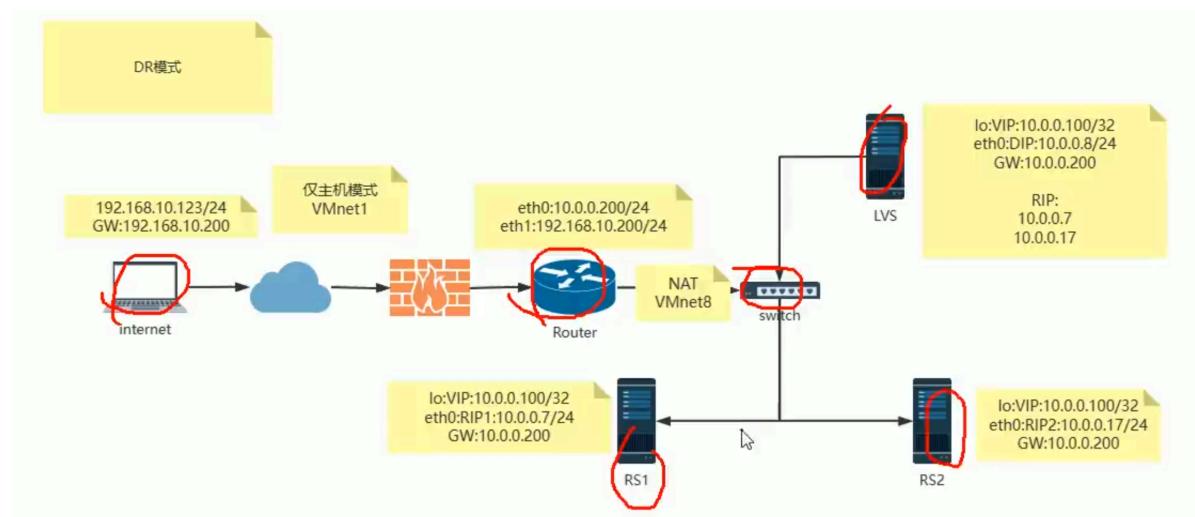
解决方案：

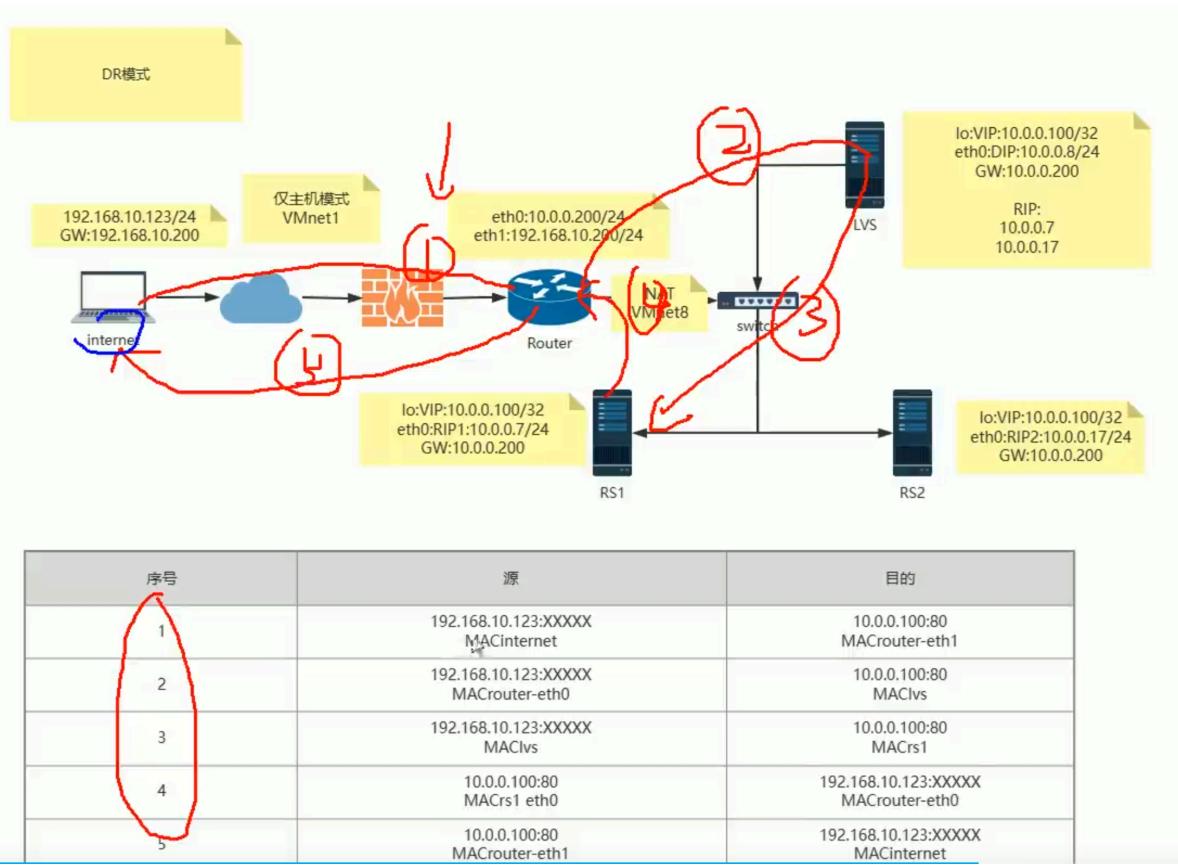
- 在这个数据包的传输过程中，会发现，RS上的VIP仅仅用作返回数据包作为源地址，与请求包的目的地址保持一致，并不负责接收数据包，只有LVS上的VIP负责接收数据包
- 因此，只要让RS上的VIP只封装数据包作为响应包的源地址，而对请求数据包即不接收也不回应，所有的请求数据包都交由LVS上的VIP去处理即可
- 通过更改两个和网卡相关的内核参数，来让该网卡既不接收数据包也不发送数据包

扩展知识：

- 免费ARP
  - 当网卡要生效启动的时候，它会向外发免费ARP，主动向外发送我自己拥有的IP地址，并让周边机器更新我的IP地址和我MAC地址的对应关系（发很多次）
  - 通过免费ARP避免有人和它产生地址冲突

DR模型是基于数据链路层工作的





在这个数据链路中只更改MAC地址，通过MAC地址，实现端到端也就是LVS到RS之间的数据传输，也因此，无法实现端口转换，如果客户端发送的请求包是要求访问80端口，则后端RS只能用80端口接收，不能改成别的端口

- 由于DR只更改数据链路层，而不更改网路层，因此它的性能相比于NAT模型更好
- 且这里LVS和RS之间不能用路由器，只能用交换机（即LVS和RS之间不能隔网段，必须在一个网段内），（因为在lvs上将10.0.0.7的ip解析成RS1的mac地址）lvs到rs1的数据传输靠ARP的实现机制是广播机制，而路由器隔离广播

DR模型中LVS服务器上不需要开启ip\_forward，因为整个数据传输链路中，根本不经过forward这个链，直接从PREROUTING到POSTROUTING

总结DR模式的特点

1. Director和各RS都配置有VIP
2. 确保前端路由器将目标IP为VIP的请求报文发往Director
  - 在前端网关做静态绑定VIP和Director的MAC地址
  - 在RS上使用arpTables工具

```
arpTables -A IN -d $VIP -j DROP
arpTables -A OUT -s $VIP -j mangle --mangle-ip-s $RIP
```

- 在RS上修改arp参数以限制arp通告及应答级别

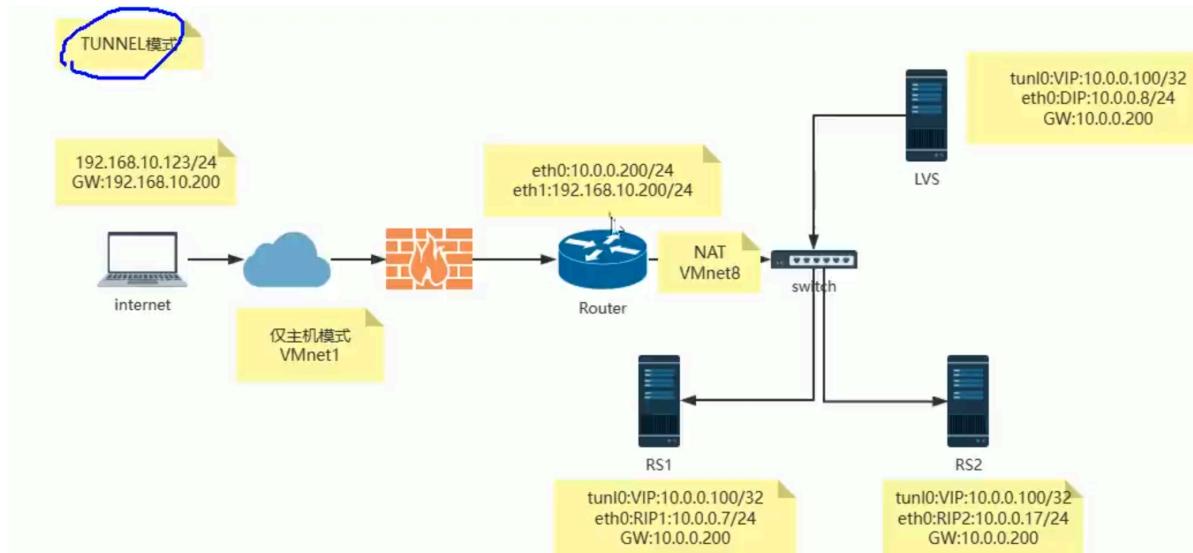
```
# 不主动不负责不拒绝
```

```
/proc/sys/net/ipv4/conf/all/arp_ignore
/proc/sys/net/ipv4/conf/all/arp_announce
```

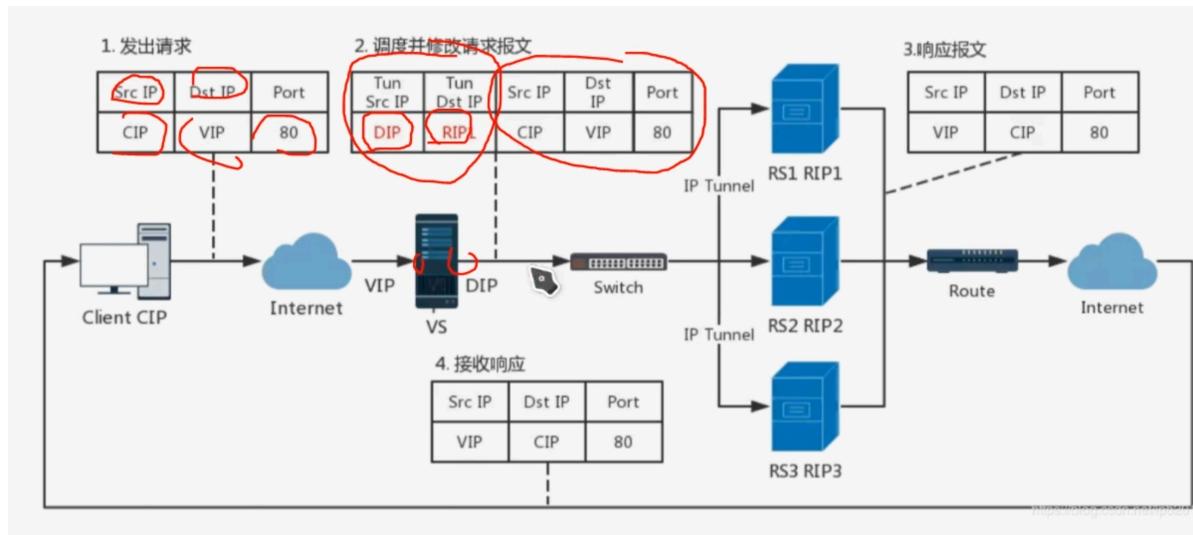
3. RS的RIP可以使用私网地址，也可以是公网地址；RIP和DIP在同一IP网络；RIP的网关不能指向DIP，以确保响应报文不会经由Director

4. RS和Director要在同一物理网络
5. 请求报文要经由Director, 但响应报文不经由Director, 而由RS直接发往Client
6. 不支持端口映射 (端口不能修改)
7. 无需开启ip\_forward
8. RS可使用大多数OS系统

## LVS的TUN模式



序号	源	目的
1	192.168.10.123:XXXXX MACinternet	10.0.0.100:80 MACrouter-eth1
2	192.168.10.123:XXXXX MACrouter-eth0	10.0.0.100:80 MAClvs
3	10.0.0.8---192.168.10.123:XXXXX MAClvs	10.0.0.7---10.0.0.100:80 MACrs1
4	10.0.0.100:80	192.168.10.123:XXXXX

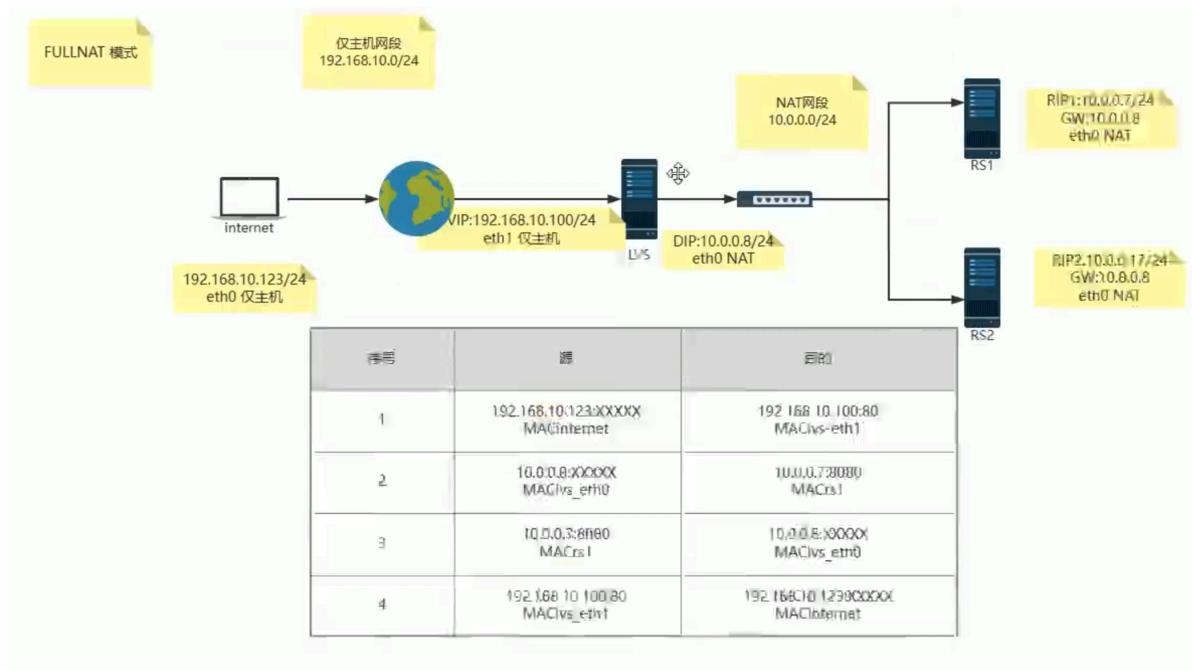


三层，添加一个新的IP头，支持LVS和RS并不在同一网段，不支持端口修改（端口在第四层）

## TUN模式实现

```
# 确认ipip模块已加载  
modprobe ipip  
  
# 在LVS上执行（配置IP隧道）  
# 添加虚拟服务器  
sudo ipvsadm -A -t 192.168.0.100:80 -s rr  
  
# 添加 Real Server  
sudo ipvsadm -a -t 192.168.0.100:80 -r 192.168.1.101:80 -i  
sudo ipvsadm -a -t 192.168.0.100:80 -r 192.168.1.102:80 -i  
  
# 在每个RS服务器上配置  
# 启用 IPIP 隧道  
sudo ip addr add 192.168.0.100/32 dev tun0  
sudo ip link set tun0 up  
  
# 配置 ARP 行为  
vim /etc/sysctl.conf  
net.ipv4.conf.all.arp_ignore = 1  
net.ipv4.conf.all.arp_announce = 2  
  
# 使其生效  
sysctl -p
```

## FULL NET模式



相比于NAT模式的优点：确保了数据一定回到LVS，经由LVS将之前由DNAT更改的目的地址变回来

如果是NAT模式的话，是有可能直接从网关出去，而不经由LVS，从而导致之前更改的目的地址无法变回来，而通讯失败

### 3.1.5 LVS工作模式总结和比较

	NAT	TUN	DR
Real Server	any	Tunneling	Non-arp device
Real server network I	private	LAN/WAN	LAN
Real server number	low (10~20)	High (100)	High (100)
Real server gateway	load balancer	own router	Own router
优点	端口转换	WAN	性能最好
缺点	性能瓶颈	要求支持隧道,不支持端口转换	不支持跨网段和端口转换

lvs-nat与lvs-fullnat:

- 请求和响应报文都经由Director
- lvs-nat: RIP的网关要指向DIP
- lvs-fullnat: RIP和DIP未必在同一IP网络, 但要能通信

lvs-dr与lvs-tun:

- 请求报文要经由Director, 但响应报文由RS直接发往Client
- lvs-dr: 通过封装新的MAC首部实现, 通过MAC网络转发
- lvs-tun: 通过在原IP报文外封装新IP头实现转发, 支持远距离通信

## 浓缩总结

**Nat** 多目标的DNAT, 四层,  
支持端口修改,  
请求报文和响应报文都要经过LVS

**DR** 二层, 只修改MAC, 不支持端口修改, 性能好, LVS负载要小,  
LVS和RS在同一网段,  
请求报文经过LVS, 响应报文不经过LVS

**TUNNEL** 三层, 添加一个新的IP头  
支持LVS和RS不在同一网段, 支持端口修改  
请求报文经过LVS, 响应报文不经过LVS

**FULLNAT** 多目标的SNAT+DNAT, 四层,  
支持端口修改  
请求报文和响应报文都要经过LVS

## LVS调度算法(共10种)

ipvs scheduler: 根据其调度时是否考虑各RS当前的负载状态, 分为两种: 静态方法和动态方法

静态方法和动态方法的区别是: 是否关注后端服务器的状态

### 静态方法 (面试)

仅根据算法本身进行调度

1. 轮询算法RR: roundrobin, 轮询, 较常用
2. 加权轮询算法WRR: Weight RR, 较常用
3. 源地址哈希算法SH: source Hashing, 实现session sticky, 源IP地址hash; 将来自于同一个IP地址的请求始终发往第一次挑中的RS, 从而实现会话绑定 (哈希: 单向不可逆)
4. 目标地址哈希算法DH: Destination Hashing; 目标地址哈希, 第一个轮询调度至RS, 后续将发往同一个目标地址的请求始终转发至第一次挑中的RS, 典型使用场景是正向代理缓存场景中的负载均衡: 如Web缓存

## 动态方法（面试）

主要根据RS当前的负载状态及调度算法进行调度 Overhead=value较小的RS将被调度

1. 最小连接算法LC: Least connection适用于长连接（没有考虑到后面服务器性能差异）

```
Overhead=activeconns*256+inactiveconns
```

2. 加权最少连接算法WLC: Weight Least connection，默认调度方法，较常见

```
Overhead=(activeconns*256+inactiveconns)/weight
```

3. 最短期望延迟算法SED: Shortest Expection Delay, 初始连接高权重优先，只检查活动连接，不考虑非活动连接

```
Overhead=(activeconns+1) * 256 / weight
```

4. 最少队列算法NQ: Never Queue, 第一轮均匀分配，后续SED
5. 基于局部的最少连接算法LBLC: Locality-Based LC, 动态的DH算法，使用场景：根据负载状态实现正向代理，实现Web Cache等
6. 带复制的基于局部的最少连接算法LBLCR: LBLC with Replication, 带复制功能的LBLC的，解决LBLC负载不均衡问题，从负载重的复制到负载轻的RS，实现WEB cache等

## LVS相关软件

### 程序包：ipvsadm

Unit File: ipvsadm service

主程序: /usr/sbin/ipvsadm

规则保存工具: /usr/sbin/ipvsadm.save

规则重载工具: /usr/sbin/ipvsadm.restore

配置文件: /etc/sysconfig/ipvsadm.config

ipvs调度规则文件: /etc/sysconfig/ipvsadm

### 安装ipvsadm

```
apt install ipvsadm
```

总共需要掌握三种用法

- 创建集群(-A)
- 在现有的集群中增加后端服务器(-a)
- 查看当前的集群状态(-L)

## 创建集群

```
# 创建集群, -t默认tcp
root@ubuntu2204:~$ipvsadm -A -t 10.0.0.108:80 -s rr
root@ubuntu2204:~$ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
    -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP   10.0.0.108:80 rr
```

## 添加服务器到集群

```
# -m默认net模式
root@ubuntu2204:~$ipvsadm -a -t 10.0.0.108:80 -r 10.0.0.128:80 -m
root@ubuntu2204:~$ipvsadm -a -t 10.0.0.108:80 -r 10.0.0.138:80 -m

# 查看集群中的状态
root@ubuntu2204:~$ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
    -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP   10.0.0.108:80 rr
    -> 10.0.0.128:80               Masq    1      0      0
    -> 10.0.0.138:80               Masq    1      0      0

# 在内存中查看配置规则
cat /proc/net/ip_vs
```

## 删除集群或里面的服务器

```
# 删除指定设备
ipvsadm -d -t 10.0.0.108:80 -r 10.0.0.128:80

# 删除整个集群
ipvsadm -D -t 10.0.0.108:

# 清楚所有的集群
ipvsadm -C
```

## 存储规则和导入规则

```
[root@mystical ~] $ipvsadm-save -n
-A -t 10.0.0.102:80 -s rr
-a -t 10.0.0.102:80 -r 10.0.0.107:80 -m -w 1
-a -t 10.0.0.102:80 -r 10.0.0.181:80 -m -w 1
# 将规则存储到指定文件
[root@mystical ~] $ipvsadm-save -n > ipvs.rules
[root@mystical ~] $ls
ipvs.rules  snap tmux-onedark-theme
# 清空所有集群
[root@mystical ~] $ipvsadm -C
[root@mystical ~] $ipvsadm -Ln
```

```
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
    -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
# 将导出的规则, 重新导入ipvsadm
[root@mystical ~] $ipvsadm-restore < ipvs.rules
[root@mystical ~] $ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
    -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP 10.0.0.102:80 rr
    -> 10.0.0.107:80               Masq    1      0      0
    -> 10.0.0.181:80               Masq    1      0      0
```

## 修改集群规则

```
# 修改集群规则
ipvsadm -E -t192.168.10.200:30 -s wrr

# 修改集群中的设备属性, 比如权重, 使用-e
ipvsadm -e -t 192.168.10.200:80 -r 10.0.0.201:8080 -w 3 -m
```

## Ubuntu系统保存规则和开机加载规则

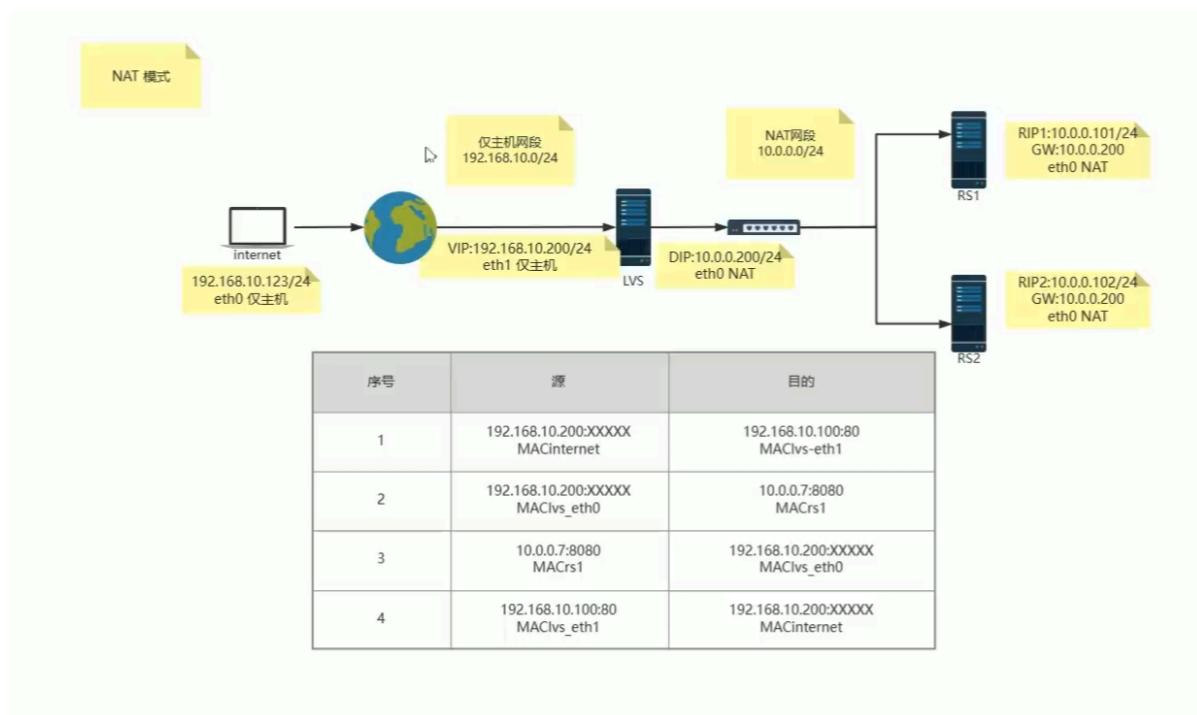
```
# 方法1:
service ipvsadm save

cat /etc/ipvsadm.rules

# 方法2:
ipvsadm-save -n > /etc/ipvsadm.rules

# 开机自启
vim /etc/default/ipvsadm
AUTO="true"
```

# LVS实验演示



## LVS(DR模型)实现

- Client: 192.168.0.6/24
- Router
  - eth0: 10.0.0.200/24
  - eth1: 192.168.0.200/24
- LVS
  - VIP: 10.0.0.100/24
  - DIP: 10.0.0.8/24
  - GW: 10.0.0.200
- RS1
  - VIP: 10.0.0.100/24
  - RIP1: 10.0.0.7/24
  - GW: 10.0.0.200
- RS2
  - VIP: 10.0.0.100/24
  - RIP2: 10.0.0.17/24
  - GW: 10.0.0.200

LVS-DR模式单网段案例

```

# 先配置内核参数，再修改网卡，防止冲突
# RS1和RS2调整内核参数
# 将all设置完之后，再设置具体网卡，配置才能生效，仅更改具体网卡配置，无效
echo 1 > /proc/sys/net/ipv4/conf/all/arp_ignore
echo 2 > /proc/sys/net/ipv4/conf/all/arp_announce
echo 1 > /proc/sys/net/ipv4/conf/lo/arp_ignore
echo 2 > /proc/sys/net/ipv4/conf/lo/arp_announce

# LVS,RS1,RS2上添加vip到回环网卡上
ifconfig lo:1 10.0.0.100/32

```

## LVS服务器

```

# 安装ipvsadm
dnf -y install ipvsadm

# 配置规则 -g对应DR模型
ipvsadm -A -t 10.0.0.100:80 -s rr
ipvsadm -a -t 10.0.0.100:80 -r 10.0.0.7:80 -g
ipvsadm -a -t 10.0.0.100:80 -r 10.0.0.17:80 -g

```

- 注意事项
  - LVS上的网关必须指向路由10.0.0.200
  - 虽然LVS的网关不参与通信过程，但是如果网关，则网卡不会处理IP源地址和目标地址不在同一网段的数据包

## LVS(DR模型)多网段案例

VIP和DIP不在同一网段，原理类似

## 防火墙标记

FWM: FireWall Mark

MARK target 可用于给特定的报文打标记

--set-mark value

其中：value可为0xffff格式，表示十六进制数字

借助于防火墙标记来分类报文，而后基于标记定义集群服务；可将多个不同的应用使用同一个集群服务进行调度

实现方法：

在Director主机打标记

```

iptables -t mangle -A PREROUTING -d $vip -p $proto -m multiport --dports $port1,
$port,... -j MARK --set-mark NUMBER

# 示例
iptables -t mangle -A PREROUTING -d 172.160.0.100 -p tcp -m multiport --dports
80, 443 -j MARK --set-mark 10

```

此时80和443算作同一个集群内，遵循轮询算法，连续访问不会同时打到同一台服务器内

# LVS持久连接

session绑定：对共享同一组RS的多个集群服务，需要统一进行绑定，lvs sh算法无法实现持久连接(lvs persistence)模版：实现无论使用任何调度算法，在一段时间内(默认360s)，能够实现将来自同一个地址的请求始终发往同一个RS

```
ipvsadm -A|E -t|u|f service-address [-s scheduler] [-p [timeout]]  
# -p 后面不跟时间，默认360s  
  
# 示例  
# -E表修改，-f 10表防火墙标记10 -p表持久时间， 默认360s  
ipvsadm -E -f 10 -p
```

持久连接实现方式

- 每端口持久(PPC)：每个端口定义为一个集群服务，每集群服务单独调度
- 每防火墙标记持久(PFWMC)：基于防火墙标记定义集群服务；可实现将多个端口上的应用统一调度，即所谓的port Affinity
- 每客户端持久(PCC)：基于0端口(表示所有服务)定义集群服务，即将客户端对所有应用的请求都调度至后端主机，必须定义为持久模式

# 高可用集群KEEPALIVED

## Keepalived介绍和安装

### VRRP协议

VRRP: (Virtual Router Redundancy Protocol)虚拟路由冗余协议

VRRP是一种网络协议，用于提高网络的可靠性和容错性。它允许多个路由器作为一个虚拟路由器组(VRID)工作，实现冗余和故障转移

VRRP协议特性：

- 冗余路由器组：VRRP允许管理员将多个路由器配置为一个虚拟路由器组。其中一个路由器被选举为主(Master)，而其他路由器则成为备用(Backup)；
- 虚拟路由器标识(VRID)：每个VRRP路由器都有一个唯一的VRID，用于区分不同的组。在同一个网络中，每个组都有一个唯一的VRID；
- 主备切换：主路由器负载处理传入数据流量。如果主路由器失效，备用路由器会接管主路由器的工作，以确保网络的联通性。这种切换是透明的，对网络中的终端设备是无感知的
- 优先级：每个VRRP路由器都有一个预先配置的优先级。优先级最高的路由器将成为主路由器。如果主路由器失效，具有最高优先级的备用路由器将接管。
- 健康检查：VRRP路由器会定期发送通告消息来确认它们的健康状态。如果其他路由器在一定时间内没有收到主路由器的通告消息，它们将假定主路由器已经失效，并启用备用路由器；

## Keepalived介绍

Keepalived是基于VRRP协议的软件实现，是高可用集群的通用无状态应用解决方案

Keepalived是一个开源的软件，用于在Linux系统上实现高可用性(High Availability, HA)。它通过监视服务器和服务的可用性，并在主服务器失效时自动将工作负载切换到备用服务器，从而确保系统的持续可用性。

Keepalived是许多基于Linux的高可用性解决方案中的重要组成部分，常用于负载均衡，Web服务器，邮件服务器等关键服务的高可用性部署中

Keepalived的主要功能包括：

- 虚拟IP(Virtual IP,VIP)管理：Keepalived可以管理一个虚拟IP地址，这个IP地址是用户访问服务的入口。当主服务器发生故障时Keepalived会将这个VIP快速漂移到备用服务器上，实现服务的无缝切换
- 健康检查
  - Keepalived可以定期检查服务器和服务的健康状态。通过配置健康检查机制，它可以监视服务器的负载，内存，CPU使用率等指标，以及特定服务(如HTTP，SMTP等)的可用性。一旦检测到主服务器故障，它会触发故障转移机制
- 故障转移
- 双机热备(Active-Backup)模式
- 配置简单
  - 可以通过编辑一个文本配置文件来定义虚拟IP，健康检测参数和故障转移策略等

## Keepalived架构

核心组件(Core components): 该框架定义了一些通用和全局的库

- VRRP栈(VRRP Stack): 实现VIP的消息通告（多播）
- 检查器(Checkers): 检查负责对真实服务器的健康检查
- 系统调用(System call): 实现vrrp协议状态转换时调用脚本的功能
- SMTP: 邮件组件，用于发送邮件通知
- IPVS封装器 (IPVS warpper) : 用于生成IPVS规则
- Netlink反射器 (Netlink Reflector) :Keepalived 的网络接口
- 看门狗(Watch Dog): 监控模块，用于监控Keepalived的运行情况

监控组件(Contorl Plane): 提供Keepalived.conf的解析器，完成Keepalived配置

调度器-I/O多路复用器(Scheduler-I/O Multiplexer): 针对网络目的而优化的自己的线程抽象

内存管理(Memory Management): 为某些通用的内存管理功能（例如分配，重新分配，发布等）提供访问权限

## Keepalived安装

- 编译安装

```
#更新仓库索引，安装编译工具，依赖库等
```

```
apt update && apt -y install make gcc ipvsadm build-essential pkg-config automake  
autoconf libipset-dev libnl-3-dev libnl-genl-3-dev libssl-dev libxtables-dev
```

```

libip4tc-dev libip6tc-dev bmagic-dev libsnmp-dev libglib2.0-dev ibpcrc2-dev

# 下载源码，并编译安装
wget https://keepalived.org/software/keepalived-2.2.8.tar.gz
tar xf keepalived-2.2.8.tar.gz -C /usr/local/src/
cd /usr/local/src/keepalived-2.2.8
./configure --prefix=/usr/local/keepalived
make && make install
cp keepalived/keepalived.service /lib/systemd/system
# 导入配置文件
mkdir /etc/keepalived
cp /usr/local/keepalived/etc/keepalived/keepalived.conf.sample
/etc/keepalived/keepalived.conf
# 注意：重启服务器前，将配置中的网卡名和实际网卡名统一，建议都是eth0，否则会报错导致服务无法启动
# 重启keepalived
systemctl start keepalived.service
# 修改/etc/keepalived/keepalived.conf
将vrrp_stric6注释掉
添加include /etc/keepalived/conf.d/*.conf
mkdir /etc/keepalived/conf.d

```

## 安装后查看配置文件

```

vim /usr/local/keepalived/etc/sysconfig/keepalived

# Options for keepalived. See `keepalived --help' output and keepalived(8) and
# keepalived.conf(5) man pages for a list of all options. Here are the most
# common ones :
#
# --vrrp          -P    Only run with VRRP subsystem.
# --check         -C    Only run with Health-checker subsystem.
# --dont-release-vrrp -V    Dont remove VRRP VIPs & VROUTES on daemon stop.
# --dont-release-ipvs -I    Dont remove IPVS topology on daemon stop.
# --dump-conf      -d    Dump the configuration data.
# --log-detail     -D    Detailed log messages.
# --log-facility    -S    0-7 Set local syslog facility (default=LOG_DAEMON)
#
KEEPALIVED_OPTIONS="-D"

# 编辑过程中生成的Service文件
vim /usr/local/src/keepalived-2.2.8/keepalived.service

[Unit]
Description=LVS and VRRP High Availability Monitor
After=network-online.target syslog.target
Wants=network-online.target
Documentation=man:keepalived(8)
Documentation=man:keepalived.conf(5)
Documentation=man:genhash(1)
Documentation=https://keepalived.org

[Service]
Type=notify
PIDFile=/run/keepalived.pid

```

```

KillMode=process
# 读配置文件，当前之前有一个选项
# KEEPALIVED_OPTIONS="-D"
EnvironmentFile=-/usr/local/keepalived/etc/sysconfig/keepalived
ExecStart=/usr/local/keepalived/sbin/keepalived --dont-fork $KEEPALIVED_OPTIONS
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target

# 将keepalived.service拷贝到/lib/systemd/system
# systemctl daemon-reload

```

当前Keepalived的配置文件keepalived.conf内容如下

```

! Configuration File for keepalived

global_defs {
    # 邮件通知模块，keepalived发现故障，发邮件通知
    # 收件人邮箱
    notification_email {
        acassen@firewall.loc
        failover@firewall.loc
        sysadmin@firewall.loc
    }
    # 发件人邮箱
    notification_email_from Alexandre.cassen@firewall.loc
    smtp_server 192.168.200.1
    smtp_connect_timeout 30
    -----
    # 下面是keepalived配置
    router_id LVS_DEVEL
    vrrp_skip_check_adv_addr
    # 严格模式，建议注释掉
    #vrrp_strict
    vrrp_garp_interval 0
    vrrp_gna_interval 0
    # 组播地址，默认项，同一个keepalived集群中的所有节点此地址相同，如果配置了单播，则此项无效
    vrrp_mcast_group4 224.0.0.18

    # 和vrrp_strct同时开启，会添加防火墙规则
    vrrp_iptables
}

include /etc/keepalived/conf.d/*.conf

#虚拟路由器配置
vrrp_instance VI_1 {
    state MASTER
    # 可以和VIP不是同一设备，此处设备用于心跳数据
    interface eth0
    # 虚拟路由ID，范围0-255
    virtual_router_id 51
    # 优先级，优先级高低决定VIP的位置
    priority 100
}
```

```
# vrrp通告的时间间隔, 默认1s, 同一集群内多节点, 此值必须相同
advert_int 1
# 集群内的节点验证, ID相同, 密码相同才算同一集群
authentication {
    auth_type PASS
    auth_pass 1111
}
# 此IP不可用, 因为Keepalived内嵌了防火墙策略
# 可以使用nft查看
# nft list rulest, 详情见下面
# vip
virtual_ipaddress {
    192.168.200.16
    192.168.200.17
    192.168.200.18
}
# 单播模式
unicast_src_ip 192.168.10.110
# 接受心跳数据的其它机器IP, 可以多条
unicast_peer {
    192.168.10.120
    192.168.10.130
}
# 当前主机状态变成master节点时执行脚本
notify_master "/path/notify.sh"
# 当主机状态变成backup节点时执行脚本
notify_backup "/path/notify.sh"
# 当前主机状态变成fault(失败)节点时执行脚本
notify_fault "/path/notify.sh"
# 通用配置, 该项可以替代前3项
notify "/path/notify.sh"
# 当前主机停止VRRP时触发该脚本
notify_stop "/path/notify.sh"
}

# LVS配置
        # vip和port
# 这里的vip必须在上面vrrp实例的vip中出现
virtual_server 192.168.200.100 443 {
    # 检查后端服务器的时间间隔, 默认60s
    delay_loop 6
    # 集群调度算法
    lb_algo rr
    # 集群类型
    lb_kind NAT
    persistence_timeout 50
    protocol TCP
    # 所有后端RS节点不可用时, 备用节点
    sorry_server 192.168.123 443

    # 后端RS节点的IP和Port, 一个LVS集群可以有多个后端RS配置
    real_server 192.168.201.100 443 {
        # 节点上线通知
        notify_up "/path/notify.sh"
        # 节点下线通知
        notify_down "/path/notify.sh"
    }
}
```

```
# RS节点权重
weight 1

# RS节点的健康状态检查，支持四层和七层检查
#
HTTP_GET|SSL_GET|TCP_CHECK|SMTP_CHECK|DNS_CHECK|PING_CHECK|FILE_CHECK|UDP_CHECK
SSL_GET {
    url {
        # 检测路径
        path /
        # genhash 命令来获取检查路径的hash值是否匹配
        digest ff20ad2481f97b1754ef3e12ecd3a9cc
    }
    url {
        path /mrtg/
        # url返回码，和digest二选一
        status_code 200
    }
    # 健康检查超时时长，超过此时长没有返回，就认为后端服务器不可用，默认5s
    connect_timeout 3
    # 重试次数
    retry 3
    # 重试之前的延迟
    delay_before_retry 3

    # 向RS的指定IP发起检查
    connect_ip 192.168.201.100
    # 向RS的指定端口发起检查
    connect_port 8080
    # 向RS
    bindto 192.168.10.8
}

# 使用应用层协议检查后端RS服务器
HTTP_GET|SSL_GET {

}

virtual_server 10.10.10.2 1358 {
    delay_loop 6
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    sorry_server 192.168.200.200 1358

    real_server 192.168.200.2 1358 {
        weight 1
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            url {
        }
```

```
        path /testurl2/test.jsp
        digest 640205b7b0fc66c1ea91c463fac6334d
    }
    url {
        path /testurl3/test.jsp
        digest 640205b7b0fc66c1ea91c463fac6334d
    }
    connect_timeout 3
    retry 3
    delay_before_retry 3
}
}

real_server 192.168.200.3 1358 {
    weight 1
    HTTP_GET {
        url {
            path /testurl/test.jsp
            digest 640205b7b0fc66c1ea91c463fac6334c
        }
        url {
            path /testurl2/test.jsp
            digest 640205b7b0fc66c1ea91c463fac6334c
        }
        connect_timeout 3
        retry 3
        delay_before_retry 3
    }
}
}

virtual_server 10.10.10.3 1358 {
    delay_loop 3
    lb_algo rr
    lb_kind NAT
    persistence_timeout 50
    protocol TCP

    real_server 192.168.200.4 1358 {
        weight 1
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            url {
                path /testurl2/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            url {
                path /testurl3/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            connect_timeout 3
            retry 3
            delay_before_retry 3
        }
    }
}
```

```

        }

    real_server 192.168.200.5 1358 {
        weight 1
        HTTP_GET {
            url {
                path /testurl/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            url {
                path /testurl2/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            url {
                path /testurl3/test.jsp
                digest 640205b7b0fc66c1ea91c463fac6334d
            }
            connect_timeout 3
            retry 3
            delay_before_retry 3
        }
    }
}

```

查看Keepalived内嵌的防火墙策略

```

nft list ruleset
# 如果有防火墙策略，直接清空
nft flush ruleset

```

主要配置项总结

```

# 主要配置项
global_def {}                      # 全局配置

vrrp_instance NAME                  # VRRP实例配置，虚拟路由器配置，可以有多条配置，每个业务一个，名字唯一即可

include /path/file                  # 文件包含

virtual_server 192.168.200.100 443 {}      # LVS 配置
virtual_server 10.10.10.2 1358 {}          # LVS 配置
virtual_server 10.10.10.3 1358 {}          # LVS 配置

```

## Keepalived 实现LVS高可用

### 环境说明

前置条件

- 各节点操作系统一致
- 时间统一

- 各节点网卡修改即传统修改笔命名形式
- 各节点关闭防火墙

## Keepalived 实现VIP高可用

### 架构

- Keepalived-1(master)
  - NAT:10.0.0.5 (VIP)10.0.0.123
  - 仅主机: 192.168.10.15
- Keepalived-2(slave)
  - NAT:10.0.0.22
  - 仅主机: 192.168.10.18
- Web-Server-1
  - NAT:10.0.0.200/24
  - ol:10.0.0.123/32(VIP)----此IP仅用于收包
- Web-Server-2
  - NAT:10.0.0.219/24
  - ol: 10.0.0.123/32(VIP)----此IP仅用于收包

### 配置Master节点

```
#网络配置
[root@ubuntu ~]# cat /etc/netplan/eth0.yaml
network:
  ethernets:
    eth0:
      addresses: [10.0.0.5/24]
      routes: [{to: default, via: 10.0.0.2}]
      nameservers:
        addresses: [10.0.0.2,223.5.5.5]
      version: 2
[root@ubuntu ~]# cat /etc/netplan/eth1.yaml
network:
  ethernets:
    eth1:
      addresses: [192.168.10.15/24]
      version: 2

[root@ubuntu ~]# cat /etc/default/grub
GRUB_CMDLINE_LINUX=" net.ifnames=0"

[root@ubuntu ~]# grub-mkconfig -o /boot/grub/grub.cfg; reboot

#确认网卡
[root@ubuntu ~]# ip a

# 配置Master节点
cp /etc/keepalived/keepalived.conf.sample /etc/keepalived/keepalived.conf
```

```

cat /etc/keepalived/keepalived.conf

global_defs {
    # 当前节点ID k216
    router_id k216
    # 广播地址
    vrrp_mcast_group4 226.6.6.6
    # 多播组地址是一种特殊的 IP 地址，位于 224.0.0.0 到 239.255.255.255 的范围内，用于将数据包发送给订阅了该多播组的所有主机。226.6.6.6 是一个多播地址，意味着任何发送到该地址的数据包都会被发送给所有订阅该多播组的节点。
    # 多播相关的关键代码通常涉及 IGMP (Internet Group Management Protocol) 和 MLD (Multicast Listener Discovery) 协议的实现。
}

vrrp_instance VI_1 {
    state Master
    interface eth1
    virtual_router_id 66
    priority 100
    advert_int 3
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        # VIP是10.0.0.123/24配置在eth0上
        10.0.0.123/24 dev eth0
    }
}
}

重启服务
systemctl restart keepalived.service

# 在windows中测试
ping 10.0.0.123

# 抓VRRP包，当前主机已经在向广播IP发送数据了
tcpdump vrrp -nn -i eth1

```

## 配置K2, Backup节点

```

#网络配置
[root@ubuntu ~]# cat /etc/netplan/eth0.yaml
network:
  ethernets:
    eth0:
      addresses: [10.0.0.217/24]
      routes: [{to: default, via: 10.0.0.2}]
      nameservers:
        addresses: [10.0.0.2,223.5.5.5]
  version: 2
[root@ubuntu ~]# cat /etc/netplan/eth1.yaml
network:
  ethernets:
    eth1:

```

```

    addresses: [192.168.10.18/24]
    version: 2

#配置Slave 节点
[root@ubuntu ~]# cp /etc/keepalived/keepalived.conf.sample
/etc/keepalived/keepalived.conf
[root@ubuntu ~]# cat /etc/keepalived/keepalived.conf
! Configuration File for keepalived
global_defs {
    router_id k217
    # router_id的功能:
    # router_id 是为每台运行 Keepalived 的服务器指定一个唯一的标识符。这是一个字符串，通常用于标识具体的路由器或服务器，但它与 VRRP 协议本身无关，也不会影响 VRRP 的运行
    # router_id的唯一性
    # 每台服务器（即 Master 和 Backup）都应该有一个不同的 router_id，以便在日志记录、调试、和管理时，可以清楚地区分不同的服务器。它只是一个用于识别的标签，不会参与 VRRP 的选举或状态决策。
    vrrp_mcast_group4 226.6.6.6
}
vrrp_instance VI_1 {
    state BACKUP
    interface eth1
    virtual_router_id 66
    # virtual_router_id的功能:
    # virtual_router_id 是 VRRP 实例中的一个重要参数，它唯一标识一个 VRRP 实例。这意味着在同一个网络中，每个 VRRP 实例都必须有唯一的 virtual_router_id，以避免与其他 VRRP 实例冲突
    priority 80
    advert_int 3
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        10.0.0.123/24 dev eth0
    }
}
# 重启服务
[root@ubuntu ~]# systemctl restart keepalived.service

```

## 为不同的项目设置独立的配置文件

```

mkdir /etc/keepalived/conf.d

# 在配置文件中引用
cat /etc/keepalived/keepalived.conf

! Configuration File for keepalived
global_defs {
    router_id k217
    vrrp_mcast_group4 226.6.6.6
}
include /etc/keepalived/conf.d/*.conf    #引用目录

# 每个项目一个配置文件
vrrp_instance www.m99-magedu.com {
    state BACKUP

```

```
interface eth1
virtual_router_id 66
priority 80
advert_int 3
authentication {
    auth_type PASS
    auth_pass 123456
}
virtual_ipaddress {
10.0.0.123/24 dev eth0
}
}
```

## 脑裂现象分析

"脑裂"通常指的是分布式系统中的一种故障状态。这种状态发生在集群中的节点之间失去了相互通信的能力，导致集群中的不同部分无法相互通信或协调。脑裂可能是由于网络故障、硬件故障、软件错误或配置问题等原因引起的，脑裂现象可能会导致集群中的不同部分采取冲突的操作，这可能会导致数据不一致或系统的行为不可预测。

总结：在 keepalived 集群中，同一时间，VIP 存在于一个以上的节点上，就是脑裂

```
# arping 判断同一时间有多少个vip
apt install -y arping

arping 10.0.0.123
ARPING 10.0.0.123
60 bytes from 00:50:56:33:97:17 (10.0.0.123): index=0 time=348.893 usec
60 bytes from 00:50:56:33:97:17 (10.0.0.123): index=1 time=362.429 usec
60 bytes from 00:50:56:33:97:17 (10.0.0.123): index=2 time=330.120 usec
60 bytes from 00:50:56:33:97:17 (10.0.0.123): index=3 time=251.661 usec
# 显示所有对应vip的mac地址都一致，表示没问题:w
```

## 设置keepalived的独立日志

```
# 默认日志的位置
tail -f /var/log/syslog

# keepalived中和log相关的选项
keepalived -h

-1, --log-console Log message to local console
-D, --log-detail Detailed log messages
-S, --log-facility=([0-7]|local[0-7]|user|daemon) Set syslog facility to
LOG_LOCAL[0-7], user or daemon(default)
-G, --no-syslog Don't log via syslog

# 默认日志的配置文件
cat /etc/default/keepalived
# Options to pass to keepalived

# DAEMON_ARGS are appended to the keepalived command-line
DAEMON_ARGS=""
```

```

# 修改为
DAEMON_ARGS="-s 6"

# 自定义日志
cat /etc/rsyslog.d/50-default.conf
local6.* /var/log/keepalived.log # 添加此行

# 重启服务
systemctl restart rsyslog
systemctl restart keepalived

```

## 抢占模式和非抢占模式

在上述配置中，Master节点不可用之后，VIP会漂移到Backup节点，当Master节点再次上线后，VIP还是会再次回到MASTER节点，这就是默认的抢占模式(preempt)，如果不希望发生再次漂移，可以设为非抢占模式(nopreempt)

还可以将集群中的节点的优先级设置为相同，也可以实现非抢占的效果  
非抢占实现了地址的尽可能稳定

```

# master节点配置
cat /etc/keepalived/conf.d/www.m99-magedu.com.conf
vrrp_instance www.m99-magedu.com {
    state BACKUP          # MASTER和SLAVE都配置成BACKUP
    interface eth1
    virtual_router_id 66
    priority 100          # 但MASTER的优先级更高
    advert_int 3
    nopreempt             # 显示指定非抢占模式
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        10.0.0.123/24 dev eth0
    }
}

# slave节点配置
无变化，同之前配置相同

```

## 抢占延时模式

抢占延时模式(preempt\_delay)：优先级高的主机恢复后，不会立即收回VIP，而是延时一段时间（默认300S）再收回VIP，但如果优先级低的主机不可用后，会立即获得VIP，而不会延时，此模式中不支持vrrp\_strict

```

# MASTER节点配置
cat /etc/keepalived/conf.d/www.m99-magedu.com.conf
vrrp_instance www.m99-magedu.com {
    state BACKUP          # MASTER和SLAVE都配置成BACKUP
    interface eth1
    virtual_router_id 66
    priority 100          # 但MASTER的优先级更高

```

```

advert_int 3
preempt_delay 15      # 延时15s, 默认300s
authentication {
    auth_type PASS
    auth_pass 123456
}
virtual_ipaddress {
    10.0.0.123/24 dev eth0
}
}

# slave节点配置
无变化, 同之前配置相同

```

## 配置单播通讯

keepalived中默认是多播通讯, 这样会将数据包发送到同一子网中所有设备上(交换机不能隔离多播和广播), 如果该设备上配置了多播IP, 就会对收到的数据报文进行处理, 这样会消耗更多的网络资源, 我们可以将其配置以单播点对点通信, 减少网络资源消耗, 在有些公有云环境中, 如果不支持多播网络, 也可以用单播实现, 此模式中不支持vrrp\_strict

```

# master主机配置
cat /etc/keepalived/keepalived.conf

global_defs {
    router_id k216
    vrrp_mcast_group4 226.6.6.6    # 改成单播后不会向广播地址发送数据包
}

include /etc/keepalived/conf.d/*.conf

cat /etc/keepalived/conf.d/www.m99-magedu.com.conf

vrrp_instance www.m99-magedu.com {
    state MASTER
    interface eth1
    virtual_router_id 66
    priority 100
    advert_int 3
    unicast_src_ip 192.169.10.8    # 发送数据IP
    unicast_peer {                 # 接受数据IP, 可以写多条
        192.168.10.18
    }
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        10.0.0.123/24 dev eth0
    }
}

# 重启服务
systemctl restart keepalived

```

```

# slave主机配置
cat /etc/keepalived/conf.d/www.m99-magedu.com.conf

vrrp_instance www.m99-magedu.com {
    state BACKUP
    interface eth1
    virtual_router_id 66
    priority 80
    advert_int 3
    unicast_src_ip 192.169.10.18      # 发送数据IP
    unicast_peer {                   # 接受数据IP, 可以写多条
        192.168.10.8
    }
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        10.0.0.123/24 dev eth0
    }
}

# 重启服务
systemctl restart keepalived

```

## 实现状态切换的邮件通知

```

# 发送邮件脚本, MASTER节点和SLAVE节点上都要部署
cat notify.sh

#!/bin/bash

source /etc/os-release

SENDMAIL="3140394153@qq.com"
SENPWD="XXXXXX"
SEDSMTP="smtp.qq.com"
RECEIVEMAIL="ji.wan@magedu.com"

function installSendEmail() {
    if [ $ID = "ubuntu" ];then
        dpkg -l sendemail &>/dev/null || apt install sendemail libnet-ssleay-perl
        libio-socket-ssl-perl -y
    elif [[ $ID =~ rhel|centos|rocky ]];then
        rpm -q sendemail &>/dev/null || yum install sendemail -y
    else
        echo "your os not support"
        exit
    fi
}

function sendEmail() {
    if [[ $1 =~ ^(master|backup|fault)$ ]];then
        title="$(hostname) to be $1, vip floating"
        content="$(date +'%F %T'): vrrp transition, $(hostname) changed to be $1"
    fi
}

```

```

sendemail -f "$SENDMAIL" -t "$RECEIVEMAIL" -u "$title" -m "$content" -s
"$SENDSMTP" -o message-charset=utf-8 -o tls=yes -xu "$SENDMAIL" -xp "$SENPWD"

echo "$title :: $content" >> /var/log/keepalived_notify.log

if [ $? -eq 0 ];then
    echo "send mail success"
else
    echo "send mail success"
fi
else
    echo "Usage: $(basename $0) {master|backup|fault}"
    exit 1
fi
}

installSendEmail
sendEmail $1

# 加执行权限
chmod +x notify.sh

# 修改配置文件
cat /etc/keepalived/conf.d/www.m99-magedu.com.conf
vrrp_instance www.m99-magedu.com {
    state MASTER
    interface eth1
    virtual_router_id 66
    priority 100
    advert_int 3
    unicast_src_ip 192.168.10.8
    unicast_peer {
        192.168.10.18
    }

    authentication {
        auth_type PASS
        auth_pass 123456
    }

    virtual_ipaddress {
        10.0.0.123/24 dev eth0
    }

    # MASTER节点和SLAVE节点上都需要配置
    notify_master "/root/notify.sh master"      # 当前节点变成MASTER时执行
    notify_backup "/root/notify.sh backup"       # 当前节点变成SLAVE节点时执行
    notify_fault "/root/notify.sh fault"         # 当前节点切换失败时执行
}

# 两个节点配置完成后重启服务
systemctl restart keepalived.service

```

# Keepalived实现LVS高可用

lvs用DR模型实现

```
# 10.0.0.218 后端RS主机配
apt update && apt install -y nginx

cat /etc/nginx/sites-enabled/www.m99-magedu.com.conf
server {
    listen 80;
    server_name www.m99-magedu.com;
    root /var/www/html/www.m99-magedu.com;
}

cat /var/www/html/ping.html
pong

cat /var/www/html/www.m99-magedu.com/index.html
www.m99-magedu.com -- web-1 -- index.html

systemctl restart nginx.service

# 配置VIP地址，禁用VIP的ARP请求和响应
ip a a 10.0.0.123/32 dev lo
echo 1 > /proc/sys/net/ipv4/conf/all/arp_ignore
echo 1 > /proc/sys/net/ipv4/conf/lo/arp_ignore
echo 1 > /proc/sys/net/ipv4/conf/all/arp_announce
echo 1 > /proc/sys/net/ipv4/conf/lo/arp_announce
```

10.0.0.219 后端RS主机配置

同218一样

```
# 10.0.0.216 keepalived节点配置
cat /etc/keepalived/conf.d/www.m99-magedu.com.conf
vrrp_instance www.m99-magedu.com {
    state MASTE
    interface eth1
    virtual_router_id 66
    priority 100
    advert_int 1
    unicast_src_ip 192.168.10.8
    unicast_peer {
        192.168.19.18
    }
    authentication {
        auth_type PASS
        auth_pass 123456
    }
    virtual_ipaddress {
        10.0.0.123/24 dev eth0
    }
}

virtual_server 10.0.0.123 80 {
    delay_loop 3          # 3s检查一次后端服务器
```

```
lb_algo rr          # 轮训算法
lb_kind DR          # DR模型，不支持端口映射
# persistence_timeout 50
protocol TCP         # TCP协议

real_server 10.0.0.218.80 {
    weight 1           # 权重1
    HTTP_GET {
        url {
            path /ping.html
            status_code 200
        }
        connect_timeout 1      # 检测超时时长
        retry 3               # 重试次数
        delay_before_retry-1   # 重试前等待时长
    }
}
real_server 10.0.0.219 80 {
    weight 1
    HTTP_GET {
        url {
            path /ping.html
            state_code 200
        }
        connect_timeout 1
        retry 3
        delay_before_retry 1
    }
}

# 成功重启其服务
systemctl restart keepalived
```

# KVM

KVM官网

[www.Linux-kvm.org](http://www.Linux-kvm.org)

## Hypervisor类型

### Hypervisor介绍

- Hypervisor是一种运行在基础物理服务器和操作系统之间的中间软件层，其可以允许多个操作系统和应用共享底层的内存、CPU、磁盘等物理硬件，也可叫做VMM(virtual machine monitor)，即虚拟机监视器

- Hypervisor 是所有虚拟化技术的核心，非中断地支持多工作负载迁移的能力是 Hypervisor 的基本功能，当服务器启动并执行 Hypervisor 时，它会给每一台虚拟机分配适当的内存、CPU、网络和磁盘，并加载所有虚拟机的客户操作系统

### # 理解非中断地支持多工作负载迁移的能力

非中断工作负载迁移 是 Hypervisor 的基本功能之一，特别是在企业环境中，它对于确保高可用性和负载均衡非常重要。这种能力通常称为 实时迁移 (Live Migration)。

#### 2.1 实时迁移的作用

实时迁移 允许将运行中的虚拟机从一台物理主机移动到另一台，而不会中断服务或用户活动。它有助于：

负载均衡：在物理主机之间重新分配虚拟机，以优化资源利用率。

维护和升级：在不中断虚拟机运行的情况下进行硬件维护或软件升级。

高可用性：在出现硬件故障时，将虚拟机迁移到其他健康的主机上。

#### 2.2 实时迁移的实现原理

实时迁移通常通过以下步骤实现：

预复制阶段：将虚拟机的内存状态从源主机复制到目标主机。源主机上的内存写操作继续进行，任何变化都会被跟踪。

增量复制阶段：复制内存的增量变化，直到源主机和目标主机的内存差异变得很小。

最终同步阶段：暂停虚拟机，将最后的内存变化和 CPU 状态复制到目标主机，并在目标主机上恢复虚拟机的运行。

完成迁移：在目标主机上运行虚拟机，源主机释放资源。

这种方法可以最小化迁移过程中对虚拟机运行状态的影响。

- Hypervisor 允许多种操作系统在相同的物理系统中运行
- Hypervisor 控制硬件并向来宾操作系统提供访问底层硬件的途径
- Hypervisor 向来宾操作系统提供虚拟化的硬件

## Hypervisor 分类

### 类型1：裸金属型虚拟化

### 类型2：宿主型虚拟化

## KVM 的作用

KVM 并不是将操作系统转换为 VMM，而是通过在 Linux 内核中嵌入虚拟化功能，将 Linux 内核变成一个功能强大的 Type 1 Hypervisor。KVM 依靠硬件虚拟化支持实现高效的虚拟化，并与 QEMU 配合提供完整的虚拟化解决方案。它在性能、兼容性和灵活性方面有很大优势，被广泛应用于服务器和云计算环境中。

## KVM 架构和部署

KVM 是基于虚拟化扩展 (Intel VT 或者 AMD-V) 的 X86 硬件的开源的 Linux 原生的全虚拟化解决方案。KVM 中，虚拟机被实现为常规的 Linux 进程，由标准 Linux 调度程序进行调度；虚拟机的每个虚拟 CPU 被实现为一个常规的 Linux 进程。这使得 KVM 能够使用 Linux 内核的已有功能。

但是，KVM 本身不执行任何硬件模拟，需要客户空间程序通过 /dev/kvm 接口设置一个客户机虚拟服务器的地址空间，向它提供模拟的 I/O，并将它的视频显示映射回宿主的显示屏。目前这个应用程序使用 QEMU

- KVM Module:
  - 作为 Linux 内核模块，为虚拟机提供基本的虚拟化功能。
- QEMU:
  - 在用户空间运行，为虚拟机提供设备模拟、I/O 操作等高级功能。
- Hardware:
  - 提供硬件支持的虚拟化功能，如 Intel VT-x 或 AMD-V。

## 硬件虚拟化扩展

硬件虚拟化扩展是 CPU 提供的一组特性，用于支持和优化虚拟化。它们旨在减少虚拟化软件（如 Hypervisor）和物理硬件之间的开销，从而提高虚拟机的性能和资源利用效率。

- 主要的硬件虚拟化扩展
  - Intel VT-x 和 AMD-V
  - Intel VT-d 和 AMD-Vi

## KVM部署

### 扩展：如何查询内核特性，一个功能在内核有没有

```
# 内核编译特性可以查看 /boot/config-5.15.0-107-generic
# 里面记录了所有的功能特性
root@ubuntu2204:~$ cat /boot/config-5.15.0-101-generic | head -n20
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86 5.15.0-101-generic Kernel Configuration
#
CONFIG_CC_VERSION_TEXT="gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0"
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=110400
CONFIG_CLANG_VERSION=0
CONFIG_AS_IS_GNU=y
CONFIG_AS_VERSION=23800
CONFIG_LD_IS_BFD=y
CONFIG_LD_VERSION=23800
CONFIG_LLD_VERSION=0
CONFIG_CC_CAN_LINK=y
CONFIG_CC_CAN_LINK_STATIC=y
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_CC_HAS_ASM_GOTO_OUTPUT=y
CONFIG_CC_HAS_ASM_GOTO_TIED_OUTPUT=y
CONFIG_CC_HAS_ASM_INLINE=y
CONFIG_CC_HAS_NO_PROFILE_FN_ATTR=y
...
ONFIG_NOTIFIER_ERROR_INJECTION=m
CONFIG_PM_NOTIFIER_ERROR_INJECT=m
# CONFIG_NETDEV_NOTIFIER_ERROR_INJECT is not set
CONFIG_FUNCTION_ERROR_INJECTION=y
# CONFIG_FAULT_INJECTION is not set
CONFIG_ARCH_HAS_KCOV=y
CONFIG_CC_HAS_SANCOV_TRACE_PC=y
# CONFIG_KCOV is not set
```

```

CONFIG_RUNTIME_TESTING_MENU=y
# CONFIG_LKDTM is not set
# CONFIG_TEST_MIN_HEAP is not set
# CONFIG_TEST_DIV64 is not set
# CONFIG_KPROBES_SANITY_TEST is not set
# CONFIG_BACKTRACE_SELF_TEST is not set
# CONFIG_RBTREE_TEST is not set
# CONFIG_REED_SOLOMON_TEST is not set
# CONFIG_INTERVAL_TREE_TEST is not set

# y表示功能以集成到内核中
# m表示有该功能，并以模块方式提供(按需加载)
# not set表示内核中无此模块

root@ubuntu2204:~$ls /lib/modules/5.15.0-101-generic/
build      kernel      modules.alias.bin  modules.builtin.alias.bin
modules.builtin.modinfo  modules.dep.bin   modules.order    modules.symbols
vdsd
initrd    modules.alias  modules.builtin   modules.builtin.bin      modules.dep
               modules.devname  modules.softdep  modules.symbols.bin

```

## 在虚拟机上CPU开启虚拟化

出现CPU开启了虚拟化，但是在VMware上开启虚拟化，出现平台不支持的提示，解决方案

1、修改系统设置：控制面板–程序–启用或关闭windows功能

关闭以下：

Hyper-V

windows沙盒

windows虚拟机监控程序平台 容器

适用于Linux的windows的子系统

虚拟机平台

2. 修改系统设置： windows 安全中心 – 设备安全性 – 内核隔离 – 内核隔离详细信息–内存完整性（设置为关闭状态）  
 2、修改系统设置： windows 安全中心 – 设备安全性 – 内核隔离 – 内核隔离详细信息–内存完整性（设置为关闭状态）

3. 打开cmd

运行命令：bcdedit /set hypervisorlaunchtype off

4. 重启

开启虚拟化后，查看KVM模块在内核加载情况

```

rep -i kvm /boot/config-5.15.0-112-generic
CONFIG_KVM_GUEST=y
CONFIG_HAVE_KVM=y
CONFIG_HAVE_KVM_IRQCHIP=y
CONFIG_HAVE_KVM_IRQFD=y
CONFIG_HAVE_KVM_IRQ_ROUTING=y
CONFIG_HAVE_KVM_EVENTFD=y
CONFIG_KVM_MMIO=y
CONFIG_KVM_ASYNC_PF=y
CONFIG_HAVE_KVM_MSI=y
CONFIG_HAVE_KVM_CPU_RELAX_INTERCEPT=y

```

```
CONFIG_KVM_VFIO=y
CONFIG_KVM_GENERIC_DIRTYLOG_READ_PROTECT=y
CONFIG_KVM_COMPAT=y
CONFIG_HAVE_KVM_IRQ_BYPASS=y
CONFIG_HAVE_KVM_NO_POLL=y
CONFIG_KVM_XFER_TO_GUEST_WORK=y
CONFIG_HAVE_KVM_PM_NOTIFIER=y
CONFIG_KVM=m # 表示KVM是模块
CONFIG_KVM_WERROR=y
CONFIG_KVM_INTEL=m
CONFIG_X86_SGX_KVM=y
CONFIG_KVM_AMD=m
CONFIG_KVM_AMD_SEV=y
CONFIG_KVM_XEN=y
# CONFIG_KVM_MMU_AUDIT is not set
CONFIG_PTP_1588_CLOCK_KVM=m
CONFIG_DRM_I915_GVT_KVMGT=m

root@ubuntu2204:~$modinfo kvm
filename:      /lib/modules/5.15.0-102-generic/kernel/arch/x86/kvm/kvm.ko
license:       GPL
author:        Qumranet
srcversion:    11E73BF9F209D9AB655CE26
depends:
retpoline:     Y
intree:        Y
name:          kvm
vermagic:      5.15.0-102-generic SMP mod_unload modversions
sig_id:        PKCS#7
signer:        Build time autogenerated kernel key
sig_key:       2E:23:16:41:BE:E2:75:E4:E8:36:37:AE:9D:F0:67:4D:5A:8B:52:0F
sig_hashalgo:  sha512
signature:    B9:56:8C:A1:E1:7B:A2:3B:CD:4D:F7:19:9D:12:5B:34:84:5D:89:02:
              B2:DA:26:A3:A5:A1:1D:C8:9B:D3:8B:B8:4A:BB:B3:AE:10:4E:09:05:
              AF:EF:7E:A0:65:AA:26:C1:E2:0D:71:0E:41:8F:B9:46:DC:E3:11:B8:
              20:74:3E:A9:5C:F5:D5:3C:BD:28:C5:C9:47:95:65:10:7B:9E:27:E2:
              08:84:5B:73:6A:C5:40:FD:84:59:B7:16:D5:90:E1:BC:05:46:7B:1F:
              C3:07:1F:EC:F7:C1:96:39:E9:A0:FA:E7:86:7A:70:CA:F4:90:9C:D2:
              F5:2F:13:18:0E:CF:9D:69:7F:2F:3A:11:04:07:6D:6A:11:BF:14:12:
              D3:84:E8:88:55:10:5A:80:3C:A7:BC:94:70:AC:01:9D:C9:A8:B1:C3:
              BC:C3:45:21:6C:DA:F0:49:06:1B:59:72:2B:63:EF:7D:B6:44:72:E4:
              6B:2E:F8:3D:BE:75:D5:1E:0F:75:BF:4D:53:F6:30:BB:C9:E5:C5:3A:
              8F:F9:E6:38:9A:C8:6D:EA:DC:D1:2D:3A:AB:68:CB:19:85:A5:A5:F0:
              8E:30:1F:B4:1B:55:B3:F6:AF:52:BC:AC:15:C5:A1:1F:AB:2F:D1:40:
              A2:35:10:55:CB:77:F0:B2:AC:86:7F:2B:7E:31:ED:DA:B2:75:8B:D4:
              56:09:71:C9:3B:C8:B0:BD:75:5B:07:FD:B0:4E:30:C4:0F:A4:32:82:
              D7:0F:F9:76:D0:01:94:DA:DD:DD:D2:01:19:F2:E6:6C:D7:26:A6:A9:
              DF:93:95:EB:20:7D:A9:FA:DE:68:43:C2:C6:1C:65:EB:BB:D2:FA:A5:
              09:59:48:55:88:97:7A:A5:B4:E5:4F:4E:74:BF:B9:D9:5C:23:2F:D5:
              D7:B4:75:05:D8:A7:04:35:D3:39:12:9B:91:71:36:1D:96:56:CF:82:
              DA:14:45:09:EF:3E:3C:DB:7D:4F:0E:5D:E1:49:20:E4:0D:27:7B:21:
              3A:83:57:F3:2D:97:55:BB:15:A8:D7:94:0C:C1:38:C7:56:28:A1:83:
              4B:82:14:47:EB:6D:F1:73:6F:BA:FC:55:1F:6F:CB:9B:0D:22:3A:21:
              8D:71:E8:30:6C:92:E1:75:C5:EB:33:EC:4D:EC:F6:BE:6C:2D:27:5B:
              FF:59:6D:25:E4:AC:2C:FA:39:2B:15:4D:1E:BA:85:B5:33:70:C7:4F:
              A4:60:03:12:66:C0:8B:2C:20:F1:E7:C9:CB:20:C7:8E:F4:B6:4C:F6:
```

```

71:F8:D6:87:4B:6B:ED:01:B8:A6:16:36:B5:2C:D6:33:1F:14:D6:67:
16:52:4B:93:A5:34:17:41:46:10:79:67

parm: tdp_mmu:bool
parm: mmio_caching:bool
parm: nx_huge_pages:bool
parm: nx_huge_pages_recovery_ratio:uint
parm: flush_on_reuse:bool
parm: ignore_msrs:bool
parm: report_ignored_msrs:bool
parm: min_timer_period_us:uint
parm: kvmclock_periodic_sync:bool
parm: tsc_tolerance_ppm:uint
parm: lapic_timer_advance_ns:int
parm: vector_hashing:bool
parm: enable_vmmware_backdoor:bool
parm: force_emulation_prefix:bool
parm: pi_inject_timer:bint
parm: mitigate_smt_rsb:bool
parm: halt_poll_ns:uint
parm: halt_poll_ns_grow:uint
parm: halt_poll_ns_grow_start:uint
parm: halt_poll_ns_shrink:uint

```

# lsmod用来查看已经加载到内核的模块

```
lsmod|grep -i kvm
```

```

root@ubuntu2204:~$lsmod |grep -i kvm
kvm_intel           372736  0
kvm                 1032192  1 kvm_intel

```

```

root@ubuntu2204:~$lscpu|grep -i vmx
Flags:          fpu vme de pse tsc msr pae mce cx8 apic sep
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb
rdtscp lm constant_tsc arch_perfmon rep_good nop1 xtopology tsc_reliable
nonstop_tsc cpuid tsc_known_freq pni pclmulqdq vmx ssse3 fma cx16 sse4_1 sse4_2
x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor
lahf_lm abm 3dnowprefetch ssbd ibrs ibpb stibp ibrs_enhanced tpr_shadow vnmi ept
vpid ept_ad fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid rdseed adx smap
clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves avx_vnni arat umip pkru
ospke gfni vaes vpclmulqdq rdpid movdir64b fsrm md_clear serialize
flush_l1d arch_capabilities
vulnerability Itlb multihit:           KVM: Mitigation: VMX disabled

```

## KVM体系结构

- KVM: 初始化CPU硬件, 打开虚拟机模式, 负责CPU, 内存, 中断控制器, 时钟, 由内核模块kvm\_xxx.ko实现, 工作于hypervisor, 设备/dev/kvm是一个字符设备, 在用户空间可通过ioctl()系统调用完成VM创建, 启动, 为VM分配内存, 读写VCPU的寄存器, 向VCPU注入中断, 时钟等管理功能
- QEMU进程: 工作于用户空间, 主要用于实现模拟I/O设备, 如显卡, 网卡, 硬盘等, qemu-kvm进程; 工作于用户空间, 用于实现一个虚拟机实例
- Libvirt: 提供同一的API, 守护进程libvirtd和相关工具, 如: virsh, virt-manager等

# 安装KVM相关包

## Ubuntu安装KVM

```
apt -y install cpu-checker

# 如果cpu不支持，会提示
kvm-ok
INFO: your CPU does not support KVM extensions

# 如果支持，会提示
root@ubuntu2204:~$kvm-ok
INFO: /dev/kvm exists
KVM acceleration can be used
```

```
# Ubuntu
apt update
apt -y install qemu-kvm virt-manager libvirt-daemon-system

# Centos
yum -y install qemu-kvm libvirt virt-manager virt-install virt-viewer
systemctl start --now libvirtd
yum -y install cockpit cockpit-machines
```

## Ubuntu安装cockpit工具

```
apt update && apt -y install cockpit cockpit-machines

# 安装成功后，会开放9090端口，会在web页面展示虚拟机管理页面https://10.0.0.200:9090
# 注意必须是https
```

## 图形化工具virt-manager

### Ubuntu使用virt-manager

```
export DISPLAY=10.0.0.1:0.0
virt-manager
# 必须在windows上安装x-manager，该软件是付费软件，可免费使用一个月
```

## 创建镜像目录

```
mkdir /data/isos -p
cd /data/isos
rz [镜像]
```