云计算简介：

云计算：一群提供服务的计算机的整体

↓提供服务（通信网络）

云服务：用户按需消费、使用

云计算课程=系统运维+管理云主机的技能

懒人原则：不碰到问题就不考虑解决

高效、简单原则

云计算运维做什么：

7*24 小时快速响应

故障处理

备份恢复

系统安全

软件部署

监控报警

架构调优

统计分析

脚本开发（自动化）

发展方向：

资深运维工程师

运维平台研发工程师

数据库工程师

运维总监

架构师

系统运维工程师：基础设施部署、应用环境部署、与开发协作更新应用版本、性能监控、容量规划、备份、响应处理运维故障、优化系统性能、改善自身运维方法流程

常用的开源软件：

操作系统：centos,ubantu

网站：nginx,php-fpm,tomcat

关系型数据库：mysql

nosql：redis,mongodb

代理：lvs,keepalived,haproxy

版本控制：gitlab

监控：zabbix

批量管理：ansible

打包工具：jenkins

日志分析：ELK(ElasticSearch+Logstash+Kibana)

脚本语言：shell,python

故障解决思路：

1.根据故障现象判断故障产生的影响，确定优先级

2.尽快恢复服务

a) 服务出错、假死：重启服务

b) 版本问题：保存日志待查，回滚版本

c) 资源不足：暂时关闭次要服务、紧急扩容

3.定位故障原因：

配置、版本是否变更

查看日志

4.后续完善响应监控

5.制定应急方案

运维的分类

系统运维：管理服务器、基础服务搭建、负载均衡、高可用、CDN、系统优化

桌面运维：管理客户端、企业桌面维护、办公设备维护、网络维护

网络运维：管理网络设备、服务器上架、硬件设备故障监控、设备保修

应用运维：参与产品设计、应用部署上线、版本更新、应用故障处理

运维研发：运维平台开发、基础设施开发（中间件、维护软件）、自动化运维

数据库运维：数据库设计、优化、监控、备份、灾备

运维安全：制定安全制度、安全培训、风险评估、漏洞检测

服务器：能够为其他计算机提供服务的更高级的电脑（更加稳定、高效）

分类：机架式、塔式、机柜式、刀片式

选择服务器的指标：

1.需求决定一切，着眼未来

2.不要贪图便宜，电费远比你想象的贵

3.机房中的空间是稀缺资源，机架式比塔式性价比更高

4.管理方便：比如螺丝少、支持远程控制

5.硬件通用性好，尤其是驱动

6.厂商支持

7.可替代产品是否广泛

8.在一定规模范围内，云主机的性价比更高

服务器尺寸：1U≈4.445 厘米

机柜容量：22U、37U、42U（主流）

网络设备机柜：600 或 800mm

服务器机柜：900 或 1000mm

选型参数：

支持的 CPU 颗数、CPU 核心数

内存插槽数（最少 24 个）、最大支持内存容量

硬盘接口（服务器：SAS，家用机：SATA）、盘架个数、最大支持硬盘容量

RAID 卡支持级别、RAID 卡电池

电源数量、整机功率

典型服务模式（C/S：Client/Server 架构）：

服务器：提供资源或某种功能

客户机：使用资源、功能

架构方式：

1.SMP 服务器：主要特征是共享。系统中所有的资源（例如：CPU、内存、I/O 等）都是共享的。扩展能力有限，最受限制的是内存，当 CPU 数量增多时，会导致内存紊乱，降低 CPU 效率。效率最高的 CPU 数量是 2～4 个

2.NUMA 服务器：主要特征是具有多个 CPU 模块，每个 CPU 模块由多个 CPU 组成，并且具有独立的内存、I/O 等。当需要运行大内存应用时，需要向其他模块借调内存

CPU 选择：

1．主要指标有缓存、主频、核心数量，服务器的 CPU 一般都是 XEON

2．CPU 物理核心数量越多，并发行越强，不包括超线程。超线程在 CPU 密集型计算，高负载的时候没有任何效果，甚至还会拖慢系统速度

3．CPU 主频越高单次计算速度就越快

4．CPU 缓存大的性能好

5．不同主频的内存不要混插

6．多通道的一定要按组接入，多 CPU 的按组对称分布，一般主板内存槽同组都会标示相同的颜色

7．RAID 卡很重要，带高速缓存的好，带备用电池单元的好，硬 RAID 卡（自带数据存储和专用 CPU）比软 RAID 卡好

硬盘选择：

1.硬盘转速

2.做 RAID 不能混用

3.SSD 在随机 I/O 读写上性能非常好

4.在格式化 SSD 分区的时候要注意对齐，影响 I/O 速度

5.在内核队列上选择 NOOP 效果比较好

功耗：如果选配的最大功耗大于电源模块的最大输出就需要考虑购买多个电源模块。电源不考虑冗余

基于智能平台管理接口（IPMI:Intelligent Platform Management Interface）协议的服务器带外管理工具

让用户通过网络来监控服务器上各种硬件组件的健康状况，如 CPU 电压、风扇转速、系统温度、功耗等,开放的免费标准

DELL 的管理命令：IPMITOOL

DELL 工具名称：iDRAC

参数：-U 登陆用户名

-P 登陆密码

-H 远程主机地址

-power 电源管理模块

-sel 硬件日志管理

-sol 串口重定向

网络设备选型：

交换机：端口数量、端口速率、背板带宽、POE 供电（有源以太网）

路由器：内存、flash、协议支持、模块扩展

TCP/IP：

TCP/IP 协议简介,准则 规则

主机与主机之间通信的三个要素:

IP 地址(IP address)

子网掩码(subnet mask)

IP 路由(IP router)

IP 地址(IP address) :

作用:用来标识一个节点的网络地址(节点:计算机)

IPv4 地址： 32 个二进制,点分隔 4 个部分,十进制表示

IPv6 地址： 128 个二进制,冒号分隔 8 个部分,16 进制表示

```
[root@server0 ~]# nmcli connection modify 'System eth0' ipv6.method manual
ipv6.addresses 2003:ac18::305/64 connection.autoconnect  yes
[root@server0 ~]# nmcli connection up  'System eth0'
[root@server0 ~]# ifconfig
[root@server0 ~]# ping6 2003:ac18::305
```

IP 地址的分类：

用于一般计算机网络：

A 类:1 ~ 126	网络位+主机位+主+主	A 类地址, 255.0.0.0
B 类:128 ~ 191	网+网+主+主	B 类地址, 255.255.0.0
C 类:192 ~ 223	网+网+网+主	C 类地址, 255.255.255.0

127.0.0.1 :表示特殊的 ip 地址（永远代表自己）

网络位：标识网络

主机位：标识主机

子网掩码： 标识 ip 地址的网络位与主机位，利用数字 1 表示网络位，0 表示主机位

例如：

IP 地址：192.168.1.1

子网掩码：255.255.255.0 或 192.168.1.1/24（/24 代表有 24 个网络位）

组播及科研专用：

D 类:224 ~ 239 组播

E 类:240 ~ 254 科研

公有地址（公网地址）：需要申请

私有地址：预留给企业的私有网络使用，无需申请，应用于局域网

A 类：10.0.0.0～10.255.255.255

B 类：172.16.0.0～172.31.255.255

C 类：192.168.0.0～192.168.255.255

网关：从一个网络连接到另一个网络的“关口”（应用于不同的网络之间的通信）

Windows 下查看 IP 和测试通信

查看 IP 地址的命令：

ipconfig

ipconfig /all ：可查看更详细的信息，例如 DNS 服务器信息

测试通信：一次 ping 指的双向的，有去有回

ping IP 地址

PS：如果请求超时，则需要关闭对方计算机的防火墙

计算机网络：

硬件：通过线缆将网络设备和计算机连接起来

软件：操作系统、应用软件、应用程序通过通信线路互连

功能：实现资源共享、信息传递、增加可靠性、提高系统处理能力

网络发展：

60 年代：分组交换

70-80 年代：TCP/IP

90 年代：WEB 技术

网络标准：ISO(国际标准化组织)、IEEE（电气和电子工程师学会）

网络分类：

广域网（Wide-Area Network）：连接远距离的计算机网络：internet

局域网（Local-Area Network）：连接较短距离内的计算机：企业网、校园网

设备生产厂商：思科、华为

网络拓扑结构：

点对点（WAN）： 两台设备之间有一条单独的连接，专用的用于广域网中的电路连接的路由器

星型（LAN）： 有一个中心节点的路由器，易于实现、易于网络扩展、易于故障排查、中心节点压力大、中心节点成本高

网状（LAN）： 一个节点与其他多个节点相连，提高冗余性和容错性，可靠性高，组网成本高

协议分层：为了降低网络设计的复杂性，将协议进行了分层设计

OSI 参考模型：7 层框架

应用层（数据单位：APDU）：网络服务与最终用户的一个接口

表示层（PPDU）：数据的表现形式，如加密、压缩

会话层（SPDU）：建立、管理、中止会话，例如断点续传

传输层（TPDU、数据段）：定义传输数据的协议端口号，以及流控和差错校验，实现了程序与程序的互连，可靠或不可靠的传输

网络层（报文、数据包）：进行逻辑地址寻址，实现不同网络之间的通信，定义了 IP 地址，为数据传输选择最佳路径，路由器工作在网络层

数据链路层（数据帧）：建立逻辑连接、进行硬件地址寻址、差错校验等功能、通过 MAC 地址实现数据的通信，帧包装、帧传输、帧同步。交换机工作在数据链路层

物理层（比特流）：建立、维护、断开物理连接，定义了接口及介质，实现了比特流的传输

TCP/IP 参考模型：5 层模型

应用层：HTTP、FTP、TFTP、DNS、SMTP、SNMP：计算机
传输层：TCP、UDP：防火墙
网络层：IP、ICMP、IGMP、ARP、RARP：路由器
数据链路层：交换机
物理层：网卡

套接字：
具有“通讯端点”概念的计算机网络数据结构
面向连接的套接字：主要协议是 TCP，套接字类型为 SOCK_STREAM
无连接的套接字：主要协议是 UDP，套接字类型为 SOCK_DGRAM

物理层：
以太网接口：
RJ：描述公用电信网络接口。RJ-45：网线；RJ-11：电话线
光纤接口：用以稳定的但并不是永久的连接两根或多根光纤的无源组件。FC，ST，SC，LC，MT-RJ

双绞线：由两根绝缘铜导线相互缠绕。
分类：UTP：非屏蔽双绞线； STP：屏蔽双绞线
cat5：百兆网络
cat6：千兆网络，传输频率为 200MHz
cat7：万兆网络，传输频率为 600MHz
连接：T568A：白绿（发送+）、绿色（发送-）、白橙（接收+）、蓝色、白蓝、橙色（接收-）、白棕、棕色
T568B：白橙、橙色、白绿、蓝色、白蓝、绿色、白棕、棕色
直连线：用于连接不同设备
交叉线：用于连接相同设备
控制线（全反线）：用于首次配置
中继器：放大信号，延长网络传输距离，只有一个输入端口和一个输出端口

思科交换机的命令行模式：
用户模式：登陆交换机
特权模式：一般用于查看配置信息
全局配置模式：对整个设备进行配置
接口模式：详细配置某个接口

命令：
？：查看当前模式的所有命令
no + 命令:撤销该命令的操作
ex(it):返回上一个模式
end(ctrl+z):返回特权模式

用户模式：
en(able)：进入特权模式

特权模式：

conf(igure) t(erminal)：进入全局配置模式

write：保存配置

rel(oad)：重新加载开机配置文件

copy r(unning-config) s(tartup-config)：将当前配置保存为开机配置

er(ase) (startup0config)：恢复出厂设置

全局配置模式：

h(ostname)：修改交换机名

ena(ble) p(assword)：设置交换机的明文密码

ena(ble) s(ecret):设置交换机的密文密码

line c(onsole) 0:进入控制台

no ip domain-lookup:禁用 DNS 查询

in(terfacce) f(astethernet) 0/端口号：进入某个接口

in(terfacce) r(ange) f(astethernet) 0/端口号范围 ：统一配置某个范围内的接口

控制台模式：

exec(-timeout) 分 秒：设置自动退出时间，0 分 0 秒为关闭自动退出

logging (synchronous)：设置信息自动同步

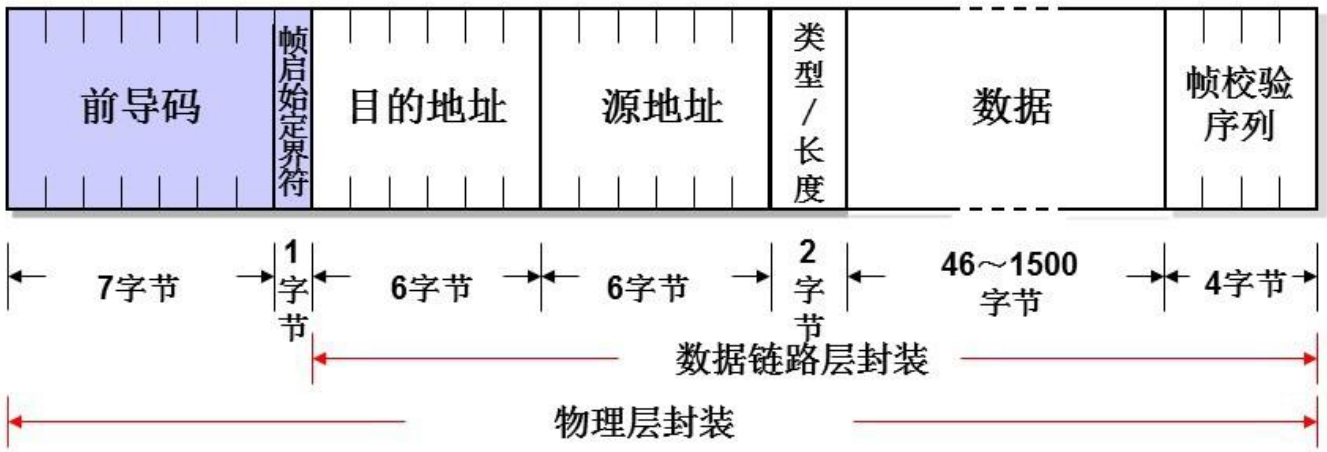
数据链路层：

以太网 MAC 地址：用来识别一个以太网上的某个单独的设备或一组设备，由 48 位二进制构成。

组成：24 位供应商标示+24 位网卡唯一编号

MAC 地址的第 8 位为 0 时表示该 MAC 地址为单播地址；为 1 时表示组播地址（这种地址通常都是虚拟的逻辑地址）。48 位都为 1 表示广播地址。

802.3 以太网帧格式：



类型：网络层协议类型

数据：如果长度不足 46 字节，则在后面用 0 补全

帧校验序列：采用 MD5 等校验方法

数据链路层封装大小：64 字节～1518 字节

以太网交换机：

交换机能够根据以太网帧中目标地址智能的转发数据，在数据链路层工作。交换机的每个端口访问另一个端口时，都有一条专有的线路，分割冲突域，实现全双工通信

交换机的工作原理：

初始状态--> 根据源 MAC 地址学习--> 向除源端口外的所有端口广播未知数据帧--> 接收方回应--> 交换机实现单播通信（转发）

更新：MAC 地址表老化时间 300 秒、交换机对应端口的 MAC 地址发生变化时

1 个接口可以对应多个 mac 地址

查看 MAC 地址表：Switch# sh(ow) mac-(address-table)

双工模式：

单工：只有一个信道，两个数据站之间只能沿单一方向传输数据 ，如：寻呼机、收音机

半双工：只有一个信道，两个数据站之间可以双向数据传输，但不能同时进行，如：对讲机

全双工：双信道，两个数据站之间可双向且同时进行数据传输

广播域：接收同样广播消息的节点的集合。交换机分割冲突域，但是不分割广播域；即交换机的所有端口属于同一个广播域

链路(VLAN)的作用：广播控制、安全性、带宽利用、控制延迟

查看 VLAN 表：Switch# sh(ow) VL(AN)

进入 VLAN 接口：Switch(config)#interface vlan 编号网关

给该接口配置 IP，方便以后远程管理：Switch(config-if)# ip address IP 地址 子网掩码

如果进入的是 VLAN1 接口的话，需要手工开启：Switch(config-if)#no shutdown

新建/进入 VLAN:Switch(config)# vlan 编号 #编号范围 0～4095，可用编号 1～4094

修改 VLAN 名：Switch(config-vlan)# na(me) 新的名字

进入端口：Switch(config)#interface fastethernet 0/1

将端口接入链路：Switch(config-if)# sw(itchport) a(ccess) (vlan) 编号

trunk 中继链路：实现不同交换机上相同 VLAN 之间的通信，采用 IEEE802.1Q，在数据帧的源地址和类型之间插入 VLAN 标示，

配置交换机接口为 trunk 模式：Switch(config-if)# sw(itchport) m(ode) t(runk)

从 trunk 中删除 VLAN：Switch(config-if)# sw(itchport) t(runk) a(llowed) (vlan) r(emove)+编号

从 trunk 中添加 VLAN：Switch(config-if)# sw(itchport) t(runk) a(llowed) (vlan) a(dd)+编号

查看 VLAN 配置：Switch# sh(ow) in(terfaces) g(igabitethernet) 0/n (switchport)

以太通道：为交换机提供了端口捆绑技术，允许交换机之间多条线路负载均衡，提高带宽，增强容错

两台交换机都要执行捆绑：

```
Switch(config)# in(terfacec) r(ange) g(igabitethernet) 0/1-2 （或 0/1，g 0/2）
Switch(config-if-range)# channel-group 以太通道组号 (mode) on
Switch(config)# in(terfacec) p(ort-channel) 以太通道组号
Switch(config-if)# sw(itchport) m(ode) t(runk)
Switch# show etherchannel summary                    #查看捆绑信息
```

网络层：提供点对点的连接

路由：跨越从源主机到目标主机的互联网络转发数据包的过程
路由器：连接不同网络的设备，将数据包从一个网络发送到另一个网络；路由器只关心网络的状态，决定最佳路由
路由器主要工作：
识别数据包的目标 IP 地址
识别数据包的源 IP 地址
在路由表中发现可能的路径
选择路由表中到达目标最好的路径
维护和检查路由信息
配置直连路由(connected)：
Router(config-if)# no shutdown　　#路由器端口默认关闭，需要手工开启
Router(config-if)# ip address IP 地址　　#通给路由器配 IP 地址，主机位常用 254
非直连路由：分为静态路由和动态路由
静态路由：由管理员手工配置，为单向条目，适合小型网络；通信双方的边缘路由器都需要指定，否则会导致数据包有去无回
路由器在内网方向配置静态路由的汇总路由，外网方向配置默认路由
配置静态路由(static)：Router(config)# ip route 目标网络 子网掩码 下一跳地址
配置浮动路由：Router(config)# ip route 目标网络 子网掩码 下一跳地址 路由优先级
#路由优先级管理距离，数字越小优先级越高
默认路由（缺省路由）：
一种特殊的静态路由，对于末梢网络的主机来说，也被称为“默认网关”，一般配置在局域网的出口路由器上
目标网络为 0.0.0.0/0.0.0.0，可匹配任何目标地址
只有当从路由表找不到任何明确匹配的路由条目时，才会使用缺省路由
三层交换技术：实现 VLAN 间通信；一次路由，多次交换；效率高于路由器+交换机的组合
思科快速转发：转发信息库（FIB：路由表的映射）、邻接关系表（MAC 地址）
三层交换机的配置：
1.确定哪些 VLAN 需要配置网关
2.如果三层交换机上没有该 VLAN，则需要手动创建
3.为每个 VLAN 创建相关的 SVI（Switch virtual Interface 交换机虚拟接口）
4.给每个 SVI 配置 IP 地址
5.启用 SVI 端口（仅限 VLAN 1）
6.启用三层交换机的 IP 路由功能
7.配置三层交换机的静态或动态路由
开启三层交换机三层功能：Switch(config)# ip routing
三层交换机需要跟其他三层交换机或者路由器通信时，需要将该连接端口改为路由端口：Switch(config-if)# no switchport
开启三层交换机虚拟 trunk 功能：Switch(config-if)# switchport trunk encapsulation dot1q #指定思科交换机 trunk 包标准
Switch(config-if)# switchport mode trunk
Switch(config)# interface vlan 编号　　#进入虚拟端口配置 IP
动态路由：基于某种路由协议实现；减少了管理任务；占用了网络带宽；根据网络拓扑或流量变化，由路由器通过路由协议自动配置；适合大型网络
路由协议分类：

按应用范围分类：

内部网关协议（IGP：Interior Gateway Protocol）:在一个自制系统（AS）内部的路由协议，内部网关路由协议有：RIP、IGRP、EIGRP、IS-IS、OSPF

外部网关协议（EGP：Exterior Gateway Protocol）：自制系统之间的路由协议，外部网关路由协议有：BGP

按执行算法分类：

距离矢量路由协议：根据从源网络到目标网络所经过的路由器的个数选择路由，适用于中小型网络：RIP、IGRP

链路状态路由协议：综合考虑从源网络到目标网络的各条路径的情况选择路由：IS-IS、OSPF

开放式最短路径优先协议（OSPF:Open Shortest Path First）：

在每台设备之间建立邻居列表，完善链路状态数据库，根据最短路径计算出路由表

OSPF 区域：为了适应大型的网络，在 AS 内划分多个区域，每个 OSPF 路由器只维护所在区域的完善链路状态信息，其中 Area0 为骨干区域，负责区域间路由信息传播

Router ID：OSPF 区域内唯一标识路由器的 IP 地址，也就是区域内的主路由器

Router ID 选取规则：选取路由器 loopback 接口上数值最高的 IP 地址，或者在物理端口选取 IP 地址最高的，或者使用 router-id 指定

配置 OSPF：

启动 OSPF 路由进程：Router(config)# router ospf 编号

指定 OSPF 协议运行的接口和所在区域：Router(config-router)# network IP 地址 反转子网掩码 area 编号　　#反转子网掩码也称为通配符掩码

在出口路由器上配置默认路由后，配置默认出口：Router(config-router)# default-information originate

在路由器上配置 DHCP：

预留静态 IP 地址：Router(config)# ip dhcp excluded-address 地址下限 地址上限

定义 IP 地址池：Router(config)# ip dhcp pool 名称

动态分配 IP 地址段：Router(dhcp-config)# network 地址网段　子网掩码

设定网关地址：Router(dhcp-config)# default-router 默认网关

配置 DNS 地址：Router(dhcp-config)# dns-server DNS 服务器

ACL 访问控制列表：

对数据包的控制：读取第三层、第四层包头信息，根据预先定义好的规则对包进行过滤

对接口应用的方向的控制：

出：已经过路由器的处理，正离开路由器接口的数据包

入：已到达路由器接口的数据包，将被路由器处理

基本 ACL（ACL 列表号：1～99）：针对源地址进行限制

扩展 ACL（ACL 列表号：100～199）：针对源地址、目标地址、端口、协议进行限制

命名访问控制列表：允许在标准和扩展访问控制列表中使用名称代替列表号，即列表号可以自定义名称

配置基本 ACL：

Router(config)# access-list 列表号 deny/permit host IP 地址　　#deny 表示拒绝，permit 表示通过，host 表示匹配主机

Router(config)# access-list 列表号 deny/permit　IP 地址　通配符掩码 #通配符掩码与 IP 地址一一对应，如果为 1 则表明这一位不需要进行匹配，如果为 0 则需

要严格匹配

配置扩展 ACL：

Router(config)# access-list 列表号 deny/permit ip 源地址 通配符掩码 目标地址 通配符掩码

Router(config)# access-list 列表号 deny/permit tcp 源地址 通配符掩码 目标地址 通配符掩码 eq 段口号

配置兜底 ACL：Router(config)# access-list 列表号 deny/permit ip any any #any 匹配所有

进入端口激活 ACL：Router(config-if)# ip access-group 列表号 in/out #in 表示入方向，out 表示出方向

查看访问控制列表：Router(config)# Show access-lists

NAT 网络地址转换(Network Address Translation)：将内部网络的私有 IP 地址转换成公网 IP 地址，使内部网络可以连接到外部网络上

优点：节省公有合法 IP 地址、处理地址重叠、增强内部网络安全性

缺点：延迟增大、配置和维护的复杂性、不支持某些应用（可以通过静态 NAT 映射避免）

实现方式：

静态转换(Static Translation)：多用于服务器

端口多路复用(PAT：Port Address Translation)：多用于办公室环境

静态 NAT 配置步骤：接口 IP 地址配置---->决定需要转换的主机地址---->决定采用什么公有地址---->在内部和外部端口上启用 NAT

配置静态转换：IP 映射：Router(config)# ip nat inside source static 内网 IP 公网 IP

端口映射：Router(config)# ip nat inside source static 协议名 内网 IP 端口号 公网 IP 端口号

启用静态转换：连接内网接口：Router(config-if)# ip nat inside

连接外网接口：Router(config-if)# ip nat outside

端口多路复用：通过改变外出数据包的源 IP 地址和源端口并进行端口转换，内部网络的所有主机均可共享一个合法 IP 地址实现互联网的访问，节约大量 IP

PAT 配置步骤：配置外部接口 IP 地址---->使用标准 ACL 确定哪些内部主机能访问外网---->配置多路复用---->在内外网接口中启用 NAT

配置多路复用（需要先配置一个 ACL）：Router(config)# ip nat inside source list ACL 编号 interface 路由器外部端口 overload

清除 NAT 转换表中的所有 PAT 缓存条目：Router# clear ip nat translation *

开启跟踪 NAT：Router# debug ip nat

关闭跟踪 NAT：Router# undebug all

ARP 地址解析协议(Address Resolution Protocol)：根据 IP 地址获取 MAC 地址的 TCP/IP 协议

原理：将包含目标 IP 地址的 ARP 请求广播到网络上的所有主机，广播的 MAC 地址为 FF-FF-FF-FF-FF-FF，并接收目标主机单播回复的 MAC 地址

收到回复后将该 IP 地址和物理地址存入本机 ARP 缓存中并保留一段时间

查看 arp 缓存表：arp -a

ICMP 控制报文协议(Internet Control Message Protocol)：TCP/IP 协议族的一个子协议，通过 IP 数据报在 IP 主机、路由器之间传递控制消息

控制信息包括：目的地不可达、TTL 超时、信息请求、信息应答、地址请求、地址应答

ping 包命令：

LINUX:-c 次数 ； -s 包大小

WINDOWS：-n 层似乎 ； -l 包大小 ； -t 持续测试网络

网络拓扑结构：核心层、汇聚层、接入层

核心层：介入互联网

汇聚层：VLan 间通讯，将主机的数据发送给核心层

接入层：用于连接主机（服务器）

TTL 生命周期：每传输过一个路由器，生命周期减 1，减到 0 时，数据包丢弃。为了防止一个数据包在网络中无限的循环下去。

广播风暴的产生：由于交换机的工作原理，当网络中存在物理环路时，广播风暴就会产生，最终会导致网络资源耗尽，交换机死机

生成树协议(STP：Spanning Tree Protocol)：逻辑上断开环路，防止数据链路层产生广播风暴，当线路故障，阻塞接口被激活，恢复通信，起备份线路作用

网桥 ID（交换机 ID）：网桥 ID 是唯一的，交换机之间选择 BID 值最小的交换机作为网络中的根网桥。16 位网桥优先级+48 位网桥 MAC 地址，其中，网桥优先级取值范围：0～65535，默认值 32768

阻塞接口：网桥 ID 最大的交换机上连接网桥 ID 第二大的交换机的接口

查看设备版本信息：Switch(config)# show version

启动生成树命令：Switch(config)# spanning-tree vlan VLAN 编号

查看生成树信息：Switch(config)# show spanning-tree vlan 编号

#若不指定 VLAN，则显示所有 VLAN 的生成树信息

配置速端口：Switch(config-if)# spanning-tree portfast

指定（主/次）根网桥：

Switch(config)# spanning-tree vlan VLAN 编号 priority 网桥优先级

#网桥优先级必须是 4096 的整数倍

Switch(config)# spanning-tree vlan VLAN 编号 root primary/secondary

#primary:主根网桥，secondary 次根网桥

增强的每 VLAN 生成树(PVST+：Per VLAN Spanning Tree Plus)：思科私有协议；配置网络中比较稳定的交换机为根网桥；实现交换机负载均衡

配置步骤：进入组接口---->若为三层交换机，则在接口设置 802.1q 封装---->设置中继链路---->设置汇聚层的交换机为一些 VLAN 的主根，另一些 VLAN 的次根---->在接入层交换机中查看生成树

HSRP 热备份路由协议 (Hot Standby Router Protocol):思科私有协议，在内网配置

成员：活跃路由器、备份路由器、虚拟路由器、其他路由器

配置 HSRP：Router(config-if)# standby 编号 ip 虚拟路由器 IP #配置完后，Standby:备份状态；Active：活跃状态；LIsten：监听状态（其他路由）

修改 HSRP 优先级：Router(config-if)# standby 编号 priority 优先级 #优先级取值范围：0~255，默认为 100，数字越大优先级越高

启动 HSRP 占先权：Router(config-if)# standby 编号 preempt

查看当前 HSRP 状态：Router# show standby brief

端口跟踪：Router(config-if)# standby 编号 track 需要跟踪的端口号 (降低优先级数值)　　#默认降 10，也可以自行决定降低优先级的数值

VRRP 虚拟路由器冗余协议(Virtual Router Redundancy Protocol)：类似 HSRP 的公有协议

NSF 不间断转发(None Stop Forwarding)：在路由器控制层面故障的过程中，数据转发不间断地正常执行。
GR 优雅重启(Graceful Restart)：实现 NSF 必须支持的功能

子网划分：无类地址（不属于 ABCDE 五类）

借位：从主机位借。所需要的网段=2^n，n 为需要借的位数

步骤：将 IP 和子网掩码都换算成二进制，将主机位都写成 0，得出网络 ID，算出该网段的可用 IP 范围

例如：192.168.0.125/26，属于 192.168.0.64 网段，该网段可用范围 192.168.0.65~126 广播地址 192.168.0.127，子网掩码 255.255.255.192

传输层：提供端到端的连接

协议：TCP 传输控制协议(Transmission Control Protocol):可靠的、面向连接的协议、传输效率低

UDP 用户数据报协议(User Datagram Protocol)：不可靠的、无连接的服务、传输效率高

TCP 的封装格式:

注：确认号=序列号+1

TCP 的连接（三次握手）：A 发送 SYN，请求建立连接-->B 回复 SYN、ACK-->A 发送 ACK

TCP 的断开（四次挥手）：A 发送 FIN，请求断开连接-->B 发送 ACK-->B 发送 FIN，请求断开连接-->A 发送 ACK

SYN：表示 SYN 报文，建立连接时将这个值设为 1

ACK(Acknowledgment Number)：ACK=1 表示确认，ACK=0 表示确认无效

FIN：表示没有数据需要发送了，FIN=1 表示断开连接请求

客户端独有的状态：(1)SYN_SENT (2)FIN_WAIT1 (3)FIN_WAIT2 (4)CLOSING (5)TIME_WAIT

服务器独有的状态：(1)LISTEN (2)SYN_RCVD (3)CLOSE_WAIT (4)LAST_ACK

共有的状态：(1)CLOSED (2)ESTABLISHED

TCP 连接断开的 10 种状态+1 种特殊状态：

LISTEN ：侦听 TCP 端口的连接请求；
SYN-SENT ：在发送连接请求后等待匹配的连接请求；
SYN-RECEIVED ：在收到和发送一个连接请求后等待对连接请求的确认；
ESTABLISHED ：代表一个打开的连接，数据可以传送给用户；
FIN-WAIT-1 ：等待远程 TCP 的连接中断请求，或先前的连接中断请求的确认；
FIN-WAIT-2 ：从远程 TCP 等待连接中断请求；
CLOSE-WAIT ：等待从本地用户发来的连接中断请求;
LAST-ACK ：等待原来发向远程 TCP 的连接中断请求的确认；
TIME-WAIT ：等待足够的时间以确保远程 TCP 接收到连接中断请求的确认；
CLOSED ：没有任何连接状态；
CLOSING ：服务器和客户端同时发起关闭时的特殊状态

TCP 常用端口号：
21：FTP
22：SSH
23：Telnet
25：SMTP
53：DNS
80：HTTP
110：POP3

UDP 的封装格式：

UDP 常用端口号：
53：DNS
69：TFTP
123：NTP

UDP 没有流控机制
UDP 的差错校验：需要上层协议提供差错控制

UNIX 诞生：1970.1.1
Linux 之父：Linus Torwalds(发布内核)
内核作用：负责调配所有的硬件
Linux 版本号：主版本.次版本.修订号
Linux 命令
用来实现某一类功能的指令或程序
命令的执行依赖于解释器(默认的解释器程序：/bin/bash)
命令 <------> 解释器（Shell） <------> 内核 <------> 硬件
Linux 命令的分类：
1.内部命令：属于解释器的

2.外部命令：解释器之外的其他程序

一套公开发布的完整 Linux 系统= Linux 内核 + 应用程序(主要是 GNU)

硬盘：

物理硬盘 ---> 分区规划 ----> 格式化 ----> 读/写文档

格式化:赋予空间文件系统: 定义数据在空间中如何存储（排列的规则），不同的文件系统，其存储方式不一样

Windows 文件系统：

FAT 旧版的格式

NTFS 新版的格式

Linux 文件系统：

RHEL6 默认 EXT4：第四代扩展文件系统，适合小文件

RHEL7 默认 XFS：高级日志文件系统，适合大文件

WAP：交换空间(虚拟内存)缓解物理真实内存的压力

格式化基本作用：

定义向磁盘介质上存储文档的方法和数据结构,以及读取文档的规则

不同类型的文件系统,其存储/读取方式不一样

格式化操作就是建立新的文件系统

最顶层为根目录(/)： 所有的数据都在此目录下（Linux 系统的起点）

/dev （Device）: 设备相关的文件

/etc :系统管理配置文件

/bin (Binary) :存放着最经常使用的命令

/sbin (Super User Binary) :超级权限的命令

/boot :启动 Linux 时使用的一些核心文件

/opt ： 主机额外安装软件存放的目录

/lib ： 系统最基本的动态连接共享库

/media：U 盘，光盘挂载目录

/mnt ： 临时挂载其他文件

/proc：内存的映射

/tmp :存放临时文件

/var: 日志文件

分区：

划分硬盘/dev/hda /dev/hdb /dev/hdc /dev/hdd

划分分区/dev/sda1 /dev/sda2 /dev/sda3 /dev/sda4

其中：hd,表示 IDE 设备

sd,表示 SCSI 设备

vd,表示虚拟机设备

例如：

/dev/sda7 表示：SCSI 设备,第一块硬盘,第七个分区

目录分配分区大小：

/boot：一般 500M

swap：虚拟内存，服务器的内存通常很大，分配内存大小的 25%的空间

/home：如果在服务器上开发较多可以单独挂载一个分区

/var：日志文件存放目录，日志文件很多的情况下可以单独挂载一个分区

/：根分区，剩余空间挂载位置

在 Virtual Machine Manager 上安装 RHEL7 系统

创建新虚拟机

本地安装介质（未来批量安装会使用网络引导）

点击浏览，载入 ISO 映象

设置虚拟机内存和 CPU 数（针对多核 CPU） PS：虚拟机和真机共享内存，即：虚拟机运行后，会全部占用预设的内存

设置虚拟机允许使用的最大硬盘空间和虚拟机名称，选择虚拟网络：隔离网络，仅使用内部和主机路由。原因：虚拟机设置 IP 不会影响真机网络

进入安装界面，从 iso 镜像安装时选择"install red hat enterprise linux *.*"。其他渠道安装选择"Test this media & ...."

此时，如果想让鼠标回到真机：Ctrl+Alt

语言可选英语（最上面）或者中文（倒数第二个）

软件选择：带 GUI 的服务器，即有图形界面的 Linux

安装位置：可选自动配置分区，也可手动分区

开始安装，同时设置 root 的密码，由于 RHEL7 对 root 的密码有复杂性要求，所以需要设置复杂密码或按 2 次“完成”设置

root：Linux 管理员用户

RHEL7 基本操作

虚拟控制台切换 (Ctrl + Alt + Fn 组合键)

Ctrl + Alt + F1:图形桌面，右键 --> 打开终端：伪字符终端

Ctrl + Alt + F2～6:字符控制台

命令连接虚拟机：virsh

list	列出所有运行中的虚拟机
list --all	列出所有虚拟机
start	开启虚拟机
console	连接到虚拟机的控制台，退出按 ctrl+]
shutdown	关闭虚拟机

虚拟机配置文件：/etc/libvirt/qemu/networks/

提示符： [用户名@计算机的主机名 当前所在目录]#或$

管理员的表示符号： #

普通用户的表示符号： $

～：用户的家目录

/root:管理员的家目录

/home/普通用户的用户名：普通用户的家目录

/home:存放所有普通用户的家目录

快捷键

tab：快速补全命令或者目录，对选项无效，续按两次 Tab 可查看所有可匹配项，实现此功能需要安装 bash-completion 软件包

```
[root@A ~]# ls  /et(tab)/sysco(tab)/netw(tab)-(tab)/ifc(tab)-e(tab)
[root@A ~]# ls  /etc/sysconfig/network-scripts/ifcfg-eth0
```

Ctrl + c ：结束正在运行的命令

Esc+. ：粘贴上一个命令的参数

Ctrl + l：复位

reset：复位并清屏

Ctrl + u：清空至行首

Ctrl + w：往回删除一个单词(以空格为界)

注意事项：

1.命令严格区分大小写

2.错误提示 bash: 命令名 : 未找到命令... 原因：命令输入有误或该命令(软件)没有安装

TUI(Text-based User Interface)：文本用户界面

GUI(Graphical User Interface)：图形用户界面

CLI(command-line interface)：命令行界面

命令行的一般格式：命令名 [选项] [参数 1] [参数 2]...

查看系统默认显示编码：locale

设置提示为中文：LANG=zh_CN.UTF-8

命令执行优先级：函数-->别名-->内部命令-->外部命令

函数：function 函数名() {...}

别名：alias

内部命令：系统内核自带的功能，不需要执行系统中的文件

外部命令：由$PATH 定义的目录下的文件提供的命令

pwd ： Print Working Directory

用途:查看当前工作目录

cd ： Change Directory

用途:切换工作目录

格式:cd [目标文件夹位置]

默认参数：~，表示当前用户家目录

cd ~user：表示切换到用户 user 的家目录

```
[root@A lisi]# cd ~root          #切换到 root 用户家目录
[root@A ~]#
```

ls ： List

格式:ls [选项] [目录或文件名]

常用命令选项：

短选项：-l:以长格式显示（显示目录内容的详细信息）

-d:显示目录本身(而不是内容)的属性,与 l 连用

-h:提供易读的容量单位(K、M 等),与 l 连用

-A:列出所有文件，包括以 ".." 开头的隐含文件

-R:递归列出所有子目录

-t:按时间信息排序

可由多个短选项组合为复合选项： -ld,-lh,....

长选项：--help

```
[root@localhost ~]# cd  /        #切换到根目录
[root@localhost /]# pwd          #查看当前所在目录
[root@localhost /]# ls           #显示当前目录内容
    bin   dev  home  lib    media  opt   root  sbin  sys  usr
    boot  etc  lib64  mnt    proc  run   srv   tmp  var
[root@localhost /]# cd /boot     #切换到/boot 目录
[root@A tom]#  ls -l  /root      #显示目录内容子文档的详细属性
[root@A tom]#  ls -ld  /root     #显示目录本身的详细属性
[root@A tom]#  ls -lh  /boot     #显示目录内容的详细属性加上易读的大小单位,如 KB，MB（b 不显示）
[root@A tom]#  ls -A   /root     #显示 root 下所有文档，包括名称以"."开头的隐藏文档
```

ls --color=auto 显示的内容颜色：

蓝色：目录

黑色：文件

绿色：可执行的程序

青色：快捷方式

红色：压缩包

查看文件或目录描述：

```
[root@fzr ~]# stat anaconda-ks.cfg
      文件："anaconda-ks.cfg"
      大小：1861        块：8          IO 块：4096    普通文件
    设备：802h/2050d    Inode：537347650   硬链接：1
    权限：(0600/-rw-------)  Uid：(    0/    root)   Gid：(    0/    root)
    最近访问：2018-04-29 23:46:23.642642750 +0800
    最近更改：2018-04-29 23:10:49.825102457 +0800
    最近改动：2018-04-29 23:10:49.825102457 +0800
    创建时间：-
```

绝对路径和相对路径：

1.以 / 开始的绝对路径

```
[root@localhost /]# cd /etc/pki
[root@localhost pki]# pwd
    /etc/pki
[root@localhost pki]# ls
```

```
CA          consumer     java    nss-legacy  product-default  rsyslog
ca-trust    entitlement  nssdb   product     rpm-gpg          tls
[root@localhost pki]# cd /etc/pki/CA
[root@localhost CA]#
```

2.以当前路径为参照的相对路径

```
[root@localhost CA]# cd /etc/pki/
[root@localhost pki]# pwd
    /etc/pki
[root@localhost pki]# cd CA
[root@localhost CA]# pwd
    /etc/pki/CA
[root@localhost CA]# cd ..      返回到上一层目录（后退）
[root@localhost pki]# pwd
    /etc/pki
```

du 命令：查看文件或目录的大小

参数：-s 统计目录的总容量

-h 人性化显示容量(带单位)

```
[root@desktop0 ~]#  du -sh /boot/
```

cat ： 连接文件并在标准输出上输出内容

格式：cat 选项 文件路径

常用选项：

-n ： 给所有输出行编号

-b ： 给非空输出行编号

```
[root@localhost /]# cat  /etc/redhat-release    #储存本机系统的具体版本信息
[root@localhost /]# cat  /etc/passwd            #查看/etc/passwd 文件的内容
[root@localhost /]# cat  /etc/fstab             #查看/etc/fstab 文件的内容
[root@localhost /]# cat -n /etc/passwd          #显示文件内容时加上行号
```

tr：将标准输入的字符进行替换，并将结果标准输出

格式：tr [选项]... SET1 SET2

示例：

```
[root@fzr ~]# tr 'a-z' 'A-Z'    #将小写转化为大写
```

uname ：显示输出系统信息

```
[root@localhost /]# uname -r        #显示操作系统发行版本(内核版本)
    3.10.0-327.el7.x86_64
[root@localhost /]# uname -p        #显示主机处理器(CPU)类型
    x86_64
[root@server0 ~]# uname -n          #显示主机名
    server0.example.com
```

lscpu ：列出 CPU 处理器信息

```
[root@room9pc01 ~]# lscpu
```

列出内存信息

```
[root@room9pc01 ~]# cat /proc/meminfo
```

查看帮助的方法,按 q 退出

命令 -h

命令 --help

man 命令 #对 man 帮助是可以/搜索

hostname:列出当前系统的主机名,临时修改主机名称

```
[root@svr7 桌面]#  hostname          #等同于 uname -n
[root@svr7 桌面]#  hostname  A       #修改主机名为 A
```

注：需要重新打开终端，才能看到提示符的变化

ifconfig:列出已激活的网卡连接信息，默认第一张网卡的名为 eth0

```
[root@svr7 ~]# ifconfig
[root@A ~]# ifconfig eth0
[root@A ~]# ifconfig eth0 192.168.1.1  #临时设置 IP 地址为 192.168.1.1
```

ping 命令：

格式：ping 参数 ip

不会自动停止，需要按 Ctrl+c 强行停止。 一次 ping 指的双向的，有去有回

参数：-c 次数 ;-i 间隔秒数 ; -W 超时等待时间

```
[root@A ~]# ping 192.168.1.1
```

ip：查看网卡

```
[root@client ~]# ip address ls
[root@client ~]# ip address ls dev eth3
[root@proxy ~]# ip address add dev eth1 192.168.2.5/24
```

poweroff 立刻关机

shutdown 关机

格式：-h 关机时间，不带时间表示马上关机

-c 取消-h 设置的关机

reboot:重启

ether-wake 网络开机

参数:-i 网卡名

```
[root@room9pc01 ~]# ether-wake -i p8p1 d8:9e:f3:09:b0:06
```

mkdir:创建文件夹(目录)

mkdir ： Make Directory

格式:mkdir [-p] [/路径/]目录名 1 [/路径/]目录名 2.. #-p ：创建多层的目录时，当父目录没有时创建父目录

--verbose 打印出新建立的每一个目录名

```
[root@svr7 ~]# mkdir  nsd01              #在家目录下创建文件夹 nsd01
[root@svr7 ~]# mkdir  /boot/nsd02        #在 boot 目录下创建文件夹 nsd02
[root@svr7 ~]# mkdir  /root/haha  /opt/test      #在 root 目录下创建文件夹 haha，在 opt 目录下创建文件夹 test
[root@server0 /]# mkdir -p /opt/aa/bb/cc/dd      #同时创建多层目录
[root@server0 /]# ls -R  /opt/aa         #递归显示：目录本身的内容，以及所有子目录的内容依次展开
```

touch：新建空文件

格式： touch [/路径/]文件名 1 /路径/]文件名 2...

```
[root@svr7 ~]# touch /opt/1.txt
[root@svr7 ~]# ls /opt
```

less:分屏阅读工具,查看大文件

格式:less [路径/]文件名...

```
[root@A ~]# less /etc/passwd
```

在页面底部的空白处：输入：/查找的内容，全文查找到的内容会用黑底标明，按 q 退出

head、tail :查看文件头尾内容

格式： head -n 文件名 #查看前 n 行的文件内容，默认 n=5

tail -n 文件名 #查看尾 n 行的文件内容，默认 n=5

```
[root@A ~]# head -3 /etc/passwd #显示 passwd 前 3 行的内容
[root@A ~]# tail -2 /etc/passwd #显示 passwd 后 2 行的内容
```

grep:输出包含指定字符串的行

格式:grep [选项]... ’查找条件’ 目标文件

-A n：显示包括查找条件所在行后 n 行的内容

-B n：显示包括查找条件所在行前 n 行的内容

-C n：显示包括查找条件所在行前后各 n 行的内容

-q ：取消输出

-i ：搜索时忽略大小写

-v ：取反匹配 显示文本文件内容中不包含指定字符串的行，常和‘^$’连用，表示非空行

^word：以字符串 word 开头

word$：以字符串 word 结尾

^$：空行

```
[root@A ~]# grep root  /etc/passwd  #在 passwd 中查找 root
[root@A ~]# grep bash  /etc/passwd  #在 passwd 中查找 bash
[root@A ~]# grep UUID  /etc/fstab   #在 fstab 中查找 UUID,区分大小写
[root@A ~]# grep man   /etc/man_db.conf      #在 man_db.conf 中查找 man
```

```
[root@server0 ~]# grep -i 'ROOT' /etc/passwd     #忽略大小写
[root@server0 ~]# grep -v 'root' /etc/passwd     #不包含 root 的行
[root@server0 ~]# grep '^root'  /etc/passwd #以 root 开头的行
[root@server0 ~]# grep 'root$'  /etc/passwd #以 root 结尾的行
[root@server0 /]# grep -v  '^$'  /etc/default/useradd   #表示出非空行
[root@server0 /]# grep -v '^#' /etc/login.defs | grep -v '^$' > /opt/1.txt
#配置文件/etc/login.defs 的有效信息（去除注释和行），并写入/opt/1.txt
```

history:历史命令

```
[root@desktop0 ~]# history      #查看历史
 上下键盘              #调用历史
 !sys         #调用最近的以 sys 开头的命令
 !数字        #通过历史命令的编号调用历史命令
[root@desktop0 ~]# history -c        #清空历史命令
[root@desktop0 ~]# > .bash_history  #清空历史记录文件
[root@desktop0 ~]# vim +46 /etc/profile      #直接定位第 46 行
   HISTSIZE=1000                     #默认历史记录是最多 1000 条
```

ln 命令：给文件或目录创建快捷方式，链接

1. 软链接

   格式：ln -s 源文件 软连接

   软链接不占用空间，但是源文件删除，链接失败

2. 硬链接

   格式：ln 源文件 硬连接

   硬链接不占用空间，源文件可以删除，链接依然能用

注意：做链接一定要用绝对路径

```
[root@desktop0 ~]# ln -s /root/123.txt /abc.txt #给/root/123.txt 创建了一个
软链接在/abc.txt
[root@desktop0 ~]# ln  /root/123.txt /cba.txt   #给/root/123.txt 创建了一个
硬链接在/cba.txt
[root@desktop0 ~]# cat /abc.txt                 #可以打开
[root@desktop0 ~]# cat /cba.txt                 #可以打开
[root@desktop0 ~]# rm -rf /root/123.txt         #删除源文件
[root@desktop0 ~]# cat /abc.txt                 #报错，找不到文件
```

访问光盘的内容：

Windows：

光盘--------->光驱设备------> CD 驱动器（访问点）

Linux：

光盘文件--------->光驱设备---------> 目录，手工配置访问点

1. 图形将光盘放入光驱设备：在虚拟机“显示虚拟硬件详情”中，找到 CD 光驱，点击连接，加载镜像

2. 查看光驱设备：

```
[root@A ~]# ls -l  /dev/cdrom
```

lrwxrwxrwx. 1 root root 3 2月 1 10:34 /dev/cdrom -> sr0

PS：sr0 ： SCSI

hda ： IDE

3.可以访问光驱设备的图形界面,也可以通过访问点访问光驱

mount 命令

格式:mount 设备路径 挂载点目录

[root@A ~]# mkdir /dvd #先创建一个挂载点目录

[root@A ~]# mount /dev/cdrom /dvd #将/dev/cdrom 设备的访问点设置为/dvd。设置后，将隐藏 dvd 目录下原有文件，仅显示光盘中文件

mount: /dev/sr0 写保护，将以只读方式挂载

[root@A ~]# ls /dvd #可以正常访问光盘里的内容，在 Linux 中访问设备资源内容，必须通过目录作为访问点进行访问

umount 命令：卸载挂载

格式:umount 挂载点目录

[root@A ~]# umount /dvd

[root@A ~]# ls /dvd

注意事项：

1.进行挂载时，挂载点目录必须要存在

2.进行挂载时，挂载点目录不建议是根目录下已经存在的目录

3.卸载时，确认当前没有任何人在访问点目录内，包括自己

挂载相关问题：

[root@A dvd]# umount /dvd

错误提示：umount: /dvd：目标忙。

(有些情况下通过 lsof(8) 或 fuser(1) 可以找到有关使用该设备的进程的有用信息)

原因：当前用户所在目录为 dvd，正在访问需要卸载的目录，所以执行失败

[root@A ~]# mount -a

错误提示：无法挂载新的镜像

原因：该镜像已挂载其他地方

解决方法：手动执行 umount -a 后重新挂载

alias：命令的别名（简化复杂命令的执行）

格式：alias 别名='真正执行的命令'

unalias：删除别名

格式：unalias 别名/[-a] #-a 表示删除所有别名

[root@A ~]# alias hn='hostname' #定义别名

[root@A ~]# hn

[root@A ~]# alias myls='ls -lh' #定义别名

[root@A ~]# myls /root

```
[root@A ~]# alias                #显示当前所有生效的别名
[root@A ~]# unalias  hn          #删除别名 hn
[root@A ~]# hn                   #命令没有找到
[root@A ~]# unalias  -a          #删除所有别名
```

永久指定别名：对单个用户：/家目录/用户名/.bashrc 或 /root/.bashrc
所有用户： /etc/bashrc

通配符：匹配不确定的文档名称，不等同于正则

*:任意多个任意字符

?:单个字符

[a-z,0-9]:连续范围中的一个字符,有且只有一个

{a,b,c}:多组不同的字符串,全匹配

示例：

```
[root@A ~]# ls /dev/tty*         #匹配/dev 目录下以 tty 开头的，后面有任意多个字符的文档
[root@A ~]# ls /etc/*.conf       #匹配/etc 目录下以.conf 结尾的文档
[root@A ~]# ls /etc/*tab         #匹配/etc 目录下以 tab 结尾的文档
[root@A ~]# ls /dev/tty?         #匹配/dev 目录下以 tty 开头的，且后面只有一个字符的文档
[root@A ~]# ls /dev/tty??        #匹配/dev 目录下以 tty 开头的，且后面只有两个字符的文档
[root@A ~]# ls /dev/tty[1-8]
[root@A ~]# ls /dev/tty[3-6]
[root@A ~]# ls /dev/tty{1,3,7,9,18}
[root@A ~]# ls /dev/tty{S0,S1}
[root@A ~]# ls /dev/tty{2[0-9],30}  #显示/dev/目录下 tty20 至 tty30
```

vim：文本编辑器

模式：

命令模式

输入模式（插入模式）

末行模式

格式：vim /目录/文件名

若目标文件不存在,则新建空文件并编辑

若目标文件已存在,则打开此文件并编辑

若目标目录不存在，打开文本编辑器，但无法保存

模式切换：

```
                按 i 或 a 进入输入模式
        命----------------------------------->输入模式
        令   按 Esc 键返回到命令模式
        模----------------------------------->末行模式
        式   按键盘":"进入末行模式
```

命令模式：

q!　　强制不保存并退出
o　　进入输入模式并另起一行
yy　　复制一行
3yy　　复制 3 行
p　　粘贴（到光标的下一行）
P　　粘贴（到光标的上一行）
h,j,k,l(左,下,上,右)
gg　　移动光标到文件头
G　　移动光标到文件尾
4G　　移动光标到第 4 行
x　　删除光标当前的一个字符
u　　撤销一步
^　　光标跳到行首
$　　光标跳到行尾
dd　　剪切一行
d$　　剪切到行尾
d^　　剪切到行首
ZZ　　保存退出
C　　删除光标之后整行，并且进入输入模式
查找关键词：　搜索（/word）切换结果（n 向后、N 向前）
ctrl+v --> G 或者↓ --> I --> # --> 两次 esc ：批量注释

末行模式

:4　　移动光标到第 4 行
:w　　保存(不退出)
:wq　　保存退出
:q!　　不保存退出
:r /etc/passwd　　读取其他文件
:w /root/tmp.txt　另存为/root/tmp.txt
:s/旧/新/　　旧内容替换为新内容，仅替换当前行的第 1 个内容
:s/旧/新/g　　替换当前行的所有旧内容为新内容
:%s/旧/新/　　替换所有行的第 1 个
:%s/旧/新/g　　替换所有行的所有旧内容
:3,5s/旧/新/g　替换 3 到 5 行的所有内容
:set nu　　显示行号
:set nonu　　不显示行号
:3,5d　　删除第 3 行到第 5 行

vim 配置文件：当前用户家目录下的.vimrc

set：设定
fenc：默认解码
fencs：默认解码列表
completeopt=longest,menu：智能补全
ai：自动缩进
ts：制表符(tabstop)缩进的空格数
et：将 tab 键转化为空格

sw：缩进代码时的缩进量(shiftwidth)

字符编码：

encoding：内部使用的字符编码方式，包括 Vim 的缓冲区、菜单文本、消息文本等

fileencoding：当前编辑的文件的字符编码方式，保存时的字符编码方式

fileencodings：fileencoding 的顺序列表，启动时会按照它所列出的字符编码方式逐一探测即将打开的文件的字符编码方式，并且将 fileencoding 设置为最终探测到的字符编码方式。

termencoding：工作的终端的字符编码方式

wc 统计文件中的行数、单词数、字节数

-l ：只输出行数

-w ：只输出单词数

-c ：只输出字节数

rm (Remove)

用途：删除文档

格式：rm [选项]... 文件或目录...

常用命令选项:

-rf:递归删除(含目录)、强制删除,一起使用表示删除该目录及该目录下所有文件

```
[root@A ~]# rm  -rf  /opt/* #删除/opt/下所有子文件
[root@A ~]# ls /opt/
[root@A ~]# touch /opt/1.txt
[root@A ~]# mkdir /opt/test01
[root@A ~]# ls -R /opt/
[root@A ~]# rm  -rf  /opt/  #删除包括/opt 在内的所有文件
[root@A ~]# ls  /opt/       #报错，opt 不存在
```

mv （Move）

用途：移动/改名

格式: mv 原文件路径 目标路径

重命名：路径不变的改名，等同于 F2

```
[root@A ~]# touch  /opt/1.txt
[root@A ~]# mkdir  /opt/nsd02
[root@A ~]# ls /opt/
    1.txt  nsd02
[root@A ~]# mv /opt/1.txt  /opt/nsd02
[root@A ~]# ls /opt/
    nsd02
[root@A ~]# ls /opt/nsd02
    1.txt
[root@A ~]# ls /opt/
[root@A ~]# mv /opt/nsd02  /opt/student
[root@A ~]# ls /opt/
```

cp （Copy）

格式:cp [选项]原文件 目标路径

常用命令选项:

-r:递归；复制目录时必须有此选项，如果没有，则报错，且略过目录

```
[root@A ~]# ls /opt/
[root@A ~]# cp /etc/redhat-release  /opt/
[root@A ~]# ls /opt/
[root@A ~]# cp -r /home/  /opt/
[root@A ~]# ls /opt/
[root@A ~]# cp -r /etc/fstab /boot/ /opt/        #当 cp 出现两个以上的参数时，永远会将最后一个参数作为目标目录，其它作为源文件
[root@A ~]# ls /opt/
[root@A ~]# ls /boot/vm*
[root@A ~]# cp /boot/vm* /opt/                   #cp 支持通配符
[root@A ~]# ls /opt/
[root@A ~]# cd /etc/sysconfig/network-scripts/
[root@A network-scripts]# cp  /etc/passwd        #cp 命令与"."连用,表示复制到当前目录下
[root@A opt]# rm -rf /opt/*
[root@A opt]# cp /etc/passwd  /opt/1.txt         #cp 命令可以重新命名，目标路径为新设文档的名称，将 passwd 以 1.txt 为名复制到/opt 下
[root@A opt]# ls /opt/
[root@A opt]# cat /opt/1.txt
[root@A opt]# cp /etc/vsftpd/vsftpd.conf{,.bak} #备份文件，逗号前为空代表源文件
```

ssh 远程管理：

格式：ssh 参数 对方用户名@服务器 IP 地址

选项：

-X(大写)：在远程管理时，可以在本机运行对端的图形程序

-p：指定端口号

注：如果不指定对方主机用户名，默认与本机用户同一用户名；不指定端口号默认 22 端口

```
[root@room9pc01 ~]# ssh  root@172.25.0.11     #远程连接 172.25.0.11
[root@room9pc01 ~]# ssh  root@172.25.0.10     #远程连接 172.25.0.10
[root@server0 ~]# exit   #退出远程管理，或者 ctrl+d
```

SSH 工具：XShell、secureCRT、putty

外网连接内网：堡垒主机（跳板机）、VPN（Virtual Private Network 虚拟专用网络）

堡垒主机：可以审计，发现用户的不正常行为

VPN：虚拟出来的企业内部专线。通过特殊的加密的通讯协议在连接在 internet 上的位于不同地方的两个或多个企业网络主机

XShell：Windows 界面远程连接 Linux，图形界面

XShell 上使用 sftp 上传下载文件：

help：帮助

bye：结束连接

put：上传文件

get：下载文件

lcd：选择本地路径

其他命令与 shell 相似

XShell 上使用 ZModem 上传下载文件:小于 200M 的非空文件,需要在服务器上安装 lrsz 包

上传：将文件拖入 XShell 窗口

下载：sz 文件名

XShel 上使用 socks5 隧道

SSH 访问控制：

常用的防护措施：用户限制、IP 限制、使用密钥登陆、防火墙设置...

修改配置文件：

```
[root@proxy ~]# vim /etc/ssh/sshd_config
    17:Port 33333                    #修改为非标准端口
    19:ListenAddress 192.168.4.5     #监听远程的本机 IP
    37:LoginGraceTime 2m             #最长等待密码时间
    38:PermitRootLogin no            #禁止 root 登陆
    40:MaxAuthTries 6                #每次连接最多认证次数
    47:AuthorizedKeysFile  .ssh/authorized_keys #密钥文件
    65:PasswordAuthentication no     #不允许密码认证
    115:seDNS no                     #不解析客户机
    加一行:protocol 2                #启用 SSH V2 协议，可以到
/etc/ssh/ssh_config 第 42 行复制
```

设置黑白名单。仅拒绝时，其余的都允许；仅允许时，其余的都拒绝：

```
DenyUsers 用户 1  用户 2@192.168.4.0/24  ...
AllowUsers 用户 1  *@192.168.4.100 ...
DenyGroups 组 1  组 2 ...
AllowGroups 组 1  组 2 ...
```

测试：

```
[root@room9pc01 ~]# ssh -p 33333 lisi@192.168.4.5
[root@room9pc01 ~]# ssh 192.168.4.5
[root@room9pc01 ~]# ssh -p 33333 192.168.4.5
[root@room9pc01 ~]# ssh -p 33333 lisi@192.168.2.5
```

客户端连接：

```
27:GSSAPIAuthentication no           #关闭 GSS 认证，提高连接速度
35:StrictHostKeyChecking no          #关闭密钥检查
```

Ctrl+Shift+t:在原有终端的基础上开启一个新的终端

软件包管理

使用 rpm 命令管理软件

RPM Package Manager,RPM 包管理器

格式：

```
rpm -q    软件名                          #检测程序是否安装
rpm -ivh 软件名-版本信息.rpm              #安装软件包，并显示安装过程,--nodeps:不验证软件包依赖
rpm -e    软件名                          #卸载软件
[root@server0 ~]# rpm  -q  firefox  #检测 firefox 程序是否安装
[root@server0 ~]# rpm  -q  haha     #检测 haha 程序是否安装
[root@server0 ~]# rpm  -q  bash     #检测 bash 解释器是否安装
```

安装软件包：

1.让光盘内容出现在系统中：首先关闭虚拟机 server，添加光驱设备

2.开机，查看是否具备光驱设备

```
[root@server0 ~]# ls /dev/cdrom
```

3.挂在光驱设备

```
[root@server0 ~]# mkdir /dvd
[root@server0 ~]# mount /dev/cdrom  /dvd
mount: /dev/sr0 写保护，将以只读方式挂载
[root@server0 ~]# ls /dvd
[root@server0 ~]# ls /dvd/Packages/vsftpd-3.0.2-22.el7.x86_64.rpm       #从光盘内容中找到安装包
[root@server0 ~]# ls /dvd/Packages/
```

4.安装及卸载软件包

```
[root@server0 ~]# rpm -q vsftpd     #查询软件是否安装
[root@server0 ~]# rpm -ivh  /dvd/Packages/vsftpd-3.0.2-22.el7.x86_64.rpm
[root@server0 ~]# rpm -q vsftpd     #查询是否安装成功
[root@server0 ~]# rpm -e vsftpd     #卸载软件
[root@server0 ~]# rpm -q vsftpd     #查询是否卸载成功
```

导入红帽签名信息(了解)

```
[root@server0 ~]# rpm --import /dvd/RPM-GPG-KEY-redhat-release
[root@server0 ~]# rpm -q vsftpd
[root@server0 ~]# rpm -ivh /dvd/Packages/vsftpd-3.0.2-22.el7.x86_64.rpm #此时安装将不再提示没有签名
```

错误：依赖检测失败：

```
bind = 32:9.9.4-50.el7 被 bind-chroot-32:9.9.4-50.el7.x86_64 需要
```

Yum 软件包仓库：自动解决依赖关系，安装软件

服务端 ：提供服务，为客户端解决依赖关系安装软件包

客户端 ：发出请求享受服务

客户端操作：

文件路径及格式(以.repo 结尾)：/etc/yum.repos.d/*.repo

错误的客户端文件会影响其余正确的客户端文件

baseurl 的提供方式：

互联网：http://

FTP：ftp://

本地：file://

[root@server0 ~]# rm -rf /etc/yum.repos.d/* #清空目录下所有 repo 文件，防止有错误的文件

[root@server0 ~]# vim /etc/yum.repos.d/nsd.repo #创建并打开 nsd.repo

```
[rhel7]                    #仓库的标识，不能有空格
name=rhel 7.0              #仓库的描述信息
baseurl=http://classroom.example.com/content/rhel7.0/x86_64/dvd/
enabled=1                  #本文件是否生效，等于 1 为生效，等于 0 为不生效
gpgcheck=1                 #是否检测红帽签名，等于 1 为检测，等于 0 为不检测
gpgkey=http://classroom.example.com/content/rhel7.0/x86_64/dvd/RPM-GPG-KEY-redhat-release  #指定签名文件
```

简便创建 repo 文件：

1.yum-config-manager --add 网址

2.echo gpgcheck=0 >>/etc/yum.repos.d/网址.repo

[root@server0 ~]# yum repolist #列出仓库信息，检测是否能发现 Yum 服务端

Yum(Yellow dog Updater, Modified)：

用于红帽系列系统的 Shell 前端软件包管理器，可以自动处理依赖性关系，并且一次安装所有依赖的软件包

命令：

yum install ：安装

yum remove ：卸载

[root@server0 ~]# yum -y install httpd sssd gcc #安装 httpd 软件、sssd 软件、gcc 软件，并且所有安装中的提示都选择 yes

[root@server0 ~]# rpm -ql httpd #查看 httpd 软件安装部署清单

示例：搭建本地 Yum 仓库

搭建 yum 源：将 CentOS 7.4 光盘内容挂载在系统中

[root@room9pc01 ~]# mkdir /dvd

[root@room9pc01 ~]# mount /iso/CentOS-7-x86_64-DVD-1708.iso /dvd #将 iso 文件的挂载点设置为 dvd

[root@room9pc01 ~]# ls /dvd

配置客户端文件：

[root@room9pc01 ~]# rm -rf /etc/yum.repos.d/*

[root@room9pc01 ~]# vim /etc/yum.repos.d/nsd.repo

```
[CentOS]
name=CentOS 7.4
baseurl=file:///dvd
enabled=1
gpgcheck=0
```

```
[root@room9pc01 ~]# yum repolist
```

测试：

```
[root@room9pc01 ~]# yum -y install xeyes    #安装一个图形的软件
[root@room9pc01 ~]# xeyes                   #运行图形的软件
```

包组：包含一个功能的所有软件的集合

多媒体包组提供声音及音频等支持

```
[root@room9pc01 ~]# yum groups list hidden              #列出所有的包组
[root@room9pc01 ~]# yum -y groups install  多媒体       #安装包组 多媒体
```

重定向命令输出： 将前面命令的输入结果当作文本文件的内容，写入到文件中

> ：覆盖重定向

>> ：追加重定向

echo 与重定向连用可直接写入内容

```
[root@server0 ~]# head -2 /etc/passwd  >  /opt/1.txt
[root@server0 ~]# cat /opt/1.txt
[root@server0 ~]# hostname  >  /opt/1.txt
[root@server0 ~]# cat /opt/1.txt
[root@server0 ~]# hostname  >>  /opt/1.txt
[root@server0 ~]# cat /opt/1.txt
[root@server0 ~]# echo 123456
[root@server0 ~]# echo 123456 >> /opt/1.txt
[root@server0 ~]# cat /opt/1.txt
```

wget:命令行下载工具

wget http://或 ftp://

```
[root@server0 ~]# wget
http://classroom.example.com/content/rhel7.0/x86_64/errata/Packages/kernel-3.10.0-123.1.2.el7.x86_64.rpm
[root@server0 ~]# ls        #查看是否下载成功
[root@server0 ~]# uname -r  #查看内核更新前版本
[root@server0 ~]# rpm -ivh kernel-3.10.0-123.1.2.el7.x86_64.rpm
[root@server0 ~]# reboot    #重启后显示新的版本号
[root@server0 ~]# uname -r
```

网络参数配置：

设置永久主机名，修改配置文件/etc/hostname

```
[root@server0 ~]# vim /etc/hostname    或  echo student.tedu.cn  >  /etc/hostname
[root@server0 ~]# cat /etc/hostname    或  hostname
[root@server0 ~]# exit                          #退出远程登陆
[root@room9pc01 ~]# ssh -X root@172.25.0.11     #重新登陆即可查看新的主机名
```

配置临时 ip 地址和子网掩码：

```
[root@proxy ~]# ip address add 201.1.2.5/24 dev eth3
```

[root@proxy ~]# ifconfig eth3 201.1.2.5/24

配置永久 ip 地址、子网掩码、网关地址：

方法 1：[root@A ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth1 #网卡配置文件

BOOTPROTO=none #获取 IP 方式，none 为手动，dhcp 为自动

NAME=eth1 #网卡名字

#UUID=bf9f81ad-849e-45bc-b0c9-3a43978d0e01 #复制其他网卡的，防止 UUID 冲突，所以注释

DEVICE=eth1 #设备名

ONBOOT=yes #是否开机自启动

IPADDR=192.168.2.5 #ip 地址

PREFIX=24 #子网掩码

方法 2：nmcli 连接管理（network manager commond line）

1.查看识别的网卡名称

[root@A ~]# nmcli connection show

2.配置 ip 地址 子网掩码 网关地址

[root@A ~]# nmcli connection modify 'System eth0' ipv4.method manual 或 auto

连接网络 修改 修改网卡名字 修改 ipv4 的方法 手工配置或自动配置

ipv4.addresses '172.25.0.100/24 172.25.0.254' connection.autoconnect yes

配置 ipv4 地址 ip 地址/子网掩码 网关地址 每次开机自动启用

[root@A ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0

3.激活配置

[root@A ~]# nmcli connection up 'System eth0'

配置永久 DNS 服务器地址：将网站域名解析成对应的 IP 地址

修改文件：/etc/resolv.conf

[root@server0 ~]# echo nameserver 172.25.254.254 > /etc/resolv.conf

nmcli 配置：

[root@server0 ~]# nmcli connection modify 'System eth0' ipv4.dns 172.25.254.254

测试 DNS 解析：

[root@server0 ~]# nslookup server0.example.com

Server: 172.25.254.254 #DNS 服务器地址

Address: 172.25.254.254#53

Name: server0.example.com #对方网址

Address: 172.25.0.11 #对方 IP

dhclient：DHCP 客户端

参数：

-I 指定网卡

-r 释放 ip 地址

释放地址：dhclient -I 网卡 -r

获取地址：dhclient -I 网卡 -v

示例：

```
[root@room9pc01 ~]# dhclient -I eht0 -r
[root@room9pc01 ~]# dhclient -I eht0 -v
```

find：查找文档位置

格式：find [目录] [条件 1] [条件 2]...

常用条件表示：

-type 类型 (f：文本文件、d：目录、l：快捷方式)

-name "文档名称" （可以使用通配符）

-iname 根据名称查找,忽略大小写

-size +或-文件大小(k、M、G)

-user 用户名

-group 组名

-maxdepth n 最大查找 n-1 层子目录

-mtime +或-n n 为文件修改时间（天）

```
[root@student ~]# find /boot/  -type l       #查找 boot 目录下为快捷方式
[root@student ~]# find /boot/  -type d       #查找 boot 目录下为目录
[root@student ~]# find /boot/  -type f       #查找 boot 目录下为文本文件
[root@student ~]# find  /etc  -name  "passwd"
[root@student ~]# find  /etc  -name  "*.conf"
[root@student ~]# find  /etc  -name  "*tab"
[root@server0 ~]# find /etc/  -iname  "PASSWD"          #实际匹配 /etc/passwd
[root@student ~]# find  /root/  -name  "nsd*"           #查找 root 下以 nsd 开头的文档
[root@student ~]# find  /root/  -name  "nsd*"  -type f      #查找 root 下以 nsd 开头的文件
[root@student ~]# find  /root/  -name  "nsd*"  -type d      #查找 root 下以 nsd 开头的目录
[root@student ~]# find /  -user student  -type f            #查找根目录下归属用户 student 的文件
[root@server0 ~]# find /home/ -group student            #查找/home 下所属组时 student 的文档
[root@student ~]# find /boot  -size +10M            #查找/boot 下大于 10M 的文档
```

[root@student ~]# find /boot -size -10M #查找/boot 下小于 10M 的文档

[root@server0 ~]# find /etc/ -maxdepth 1 -name "*.conf" #查找/etc 下以.conf 结尾的文档，不查找/etc 的子目录

[root@server0 ~]# find /var/log/ -mtime +100 #查找最后修改时间是 100 天以前的文档

[root@server0 ~]# find /var/log/ -mtime -10 #查找最后修改时间是 10 天以内的文档

-exec ：处理 find 查找的结果

格式：find .. .. -exec 处理命令 {} \;

以 {} 代替每一个结果,逐个处理,遇 \; 结束

[root@student ~]# find /boot -size +10M -exec cp {} /opt \;

[root@student ~]# find /root -name "nsd*" -type d -exec ls -lh \;

用户帐号的作用：1.可以登陆系统

2.实现访问控制（不同的用户具备不同的权限）

组帐号的作用：方便对用户帐号管理 权限方面

唯一标识：UID：UserID 用户编号

GID：GroupID 组编号

管理员（root）的 UID 必定为 0

组帐号的分类： 基本组(私有组)

附加组（公共组、从属组）

Linux 规定： 一个用户至少属于一个组（基本组）

用户管理：

用户基本信息存放在 /etc/passwd （相当于系统的户口本）

[root@server0 ~]# head -1 /etc/passwd

root: x :0 : 0 : root : /root :/bin/bash

用户名:密码占位符号:UID:基本的 GID:用户描述信息:家目录位置:默认的解释器

用户添加：useradd

格式：useradd [选项] 用户名

注意：由于未设密码，所以登陆界面无法看到

常用选项：-u 用户 id、 -d 家目录路径、 -s 指定登陆的解释器程序 、-G 附加组 、-r 创建为系统用户 、 -N 不创建同名的组 、 -M 不创建用户的家目录

[root@server0 ~]# useradd nsd01 #创建用户 nsd01

[root@server0 ~]# useradd -u 1200 nsd04 #创建用户 nsd04，同时设置用户 ID 为 1200

[root@server0 ~]# useradd -d /opt/nsd05 nsd05 #创建用户 nsd05，同时设置其家目录为/opt/nsd05

[root@server0 ~]# cat /etc/passwd

[root@server0 ~]# useradd -s /sbin/nologin nsd06 #创建用户 nsd06,同时设置解释器，该解释器表示用户不能登陆操作系统，常用于禁止用户登陆

[root@server0 ~]# id haha #如果用户不存在，显示未查找到该用户

[root@server0 ~]# id nsd06 #如果用户存在，则查看用户基本信息

组添加：groupadd

格式：groupadd 组名

[root@server0 ~]# groupadd tarena #创建 tarena 组

[root@server0 ~]# useradd -G tarena nsd07 #创建用户 nsd07,并指定附加组

[root@server0 ~]# id nsd07

设置/修改密码：passwd

格式：passwd [用户名]

[root@server0 ~]# passwd nsd01

更改用户 nsd01 的密码 。

新的 密码：

无效的密码： 密码是一个回文 #root 修改密码时，可以强制略过密码复杂性要求

重新输入新的 密码：

passwd：所有的身份验证令牌已经成功更新。#root 用户修改其他用户密码时，无需验证原密码，其他用户只能修改自己的密码，且需要输入原密码

PS：/etc/shadow 存放用户密码信息（加密）

切换帐号：su (Substitute User)：快速切换为指定的用户

普通用户执行此命令需要验证目标用户的口令

root 执行时无需验证口令

所有操作可在/var/log/secure 中查看

命令格式：su - 用户 -c “命令”

- 表示切换用户同时切换环境变量

不指明用户时，默认为 root

-c 表示用指定用户执行命令，但不切换用户

[root@proxy ~]# su - lisi

[lisi@proxy ~]$ su -

[lisi@proxy ~]$ exit #返回 su 之前的用户

[root@proxy ~]# su - lisi -c "mkdir ~/test"

[root@proxy ~]# ls -ld /home/lisi/test/

drwxrwxr-x 2 lisi lisi 6 5月 14 14:21 /home/lisi/test/

sudo(Super or another Do)：超级执行

配置文件：/etc/sudoers +92

#用户名或组名 能执行的主机=(目标用户身份，省略代表 root) 允许执行的命令，NOPASSWD 表示不验证用户密码

root ALL=(ALL) ALL

lisi ALL= /usr/sbin/user*,!/usr/sbin/user* * root,/usr/bin/passwd,!/usr/bin/passwd root

%wheel ALL=(root)

NOPASSWD:/bin/*,!/bin/passwd * root,!/bin/passwd ""

```
[lisi@proxy ~]$ sudo useradd zhangsan
[lisi@proxy ~]$ sudo passwd zhangsan
[lisi@proxy ~]$ sudo userdel zhangsan
```

查看当前用户权限：[lisi@proxy ~]$ sudo -l

为 sudo 建立专用日志，用命令编辑/etc/sudoers：

```
[root@proxy ~]# visudo
Defaults logfile="/var/log/sudo"
```

在配置文件设置别名：提高易读性、可重复使用、简化配置、更有条理

13:Host_Alias 主机别名=主机 1,主机 2...

20:User_Alias 用户别名=用户 1,用户 2...

30:Cmnd_Alias 命令集别名=命令 1,命令 2...

92:用户别名 主机别名=命令集别名

管道操作： 将前面的命令输出结果，交由后面命令，做为后面命令的参数再处理一遍

符号：|

```
[root@server0 ~]# ifconfig | head -2                    #取网络信息的前2行
[root@server0 ~]# head -12  /etc/passwd  |   tail -5     #passwd 文件的第 8-12 行
[root@server0 ~]# cat -n /etc/passwd | head -12  | tail -5  #passwd 文件的第 8-12 行，并保留原编号
```

非交互式设置密码

格式：echo '密码' | passwd --stdin 用户名

参数--stdin : 非交互式，取消交互

```
[root@server0 ~]# echo 123  |  passwd --stdin nsd01
[root@server0 ~]# echo redhat  |  passwd --stdin nsd01
```

修改用户属性：usermod (user modify)

格式：usermod [选项]... 用户名

常用命令选项：与 useradd 相同，-G 附加组：无论原先有多少个附加组，都重置为新设的附加组

```
[root@server0 ~]# useradd tom
[root@server0 ~]# id tom
[root@server0 ~]# usermod -u 1300 -d /opt/tom -s /sbin/nologin tom
[root@server0 ~]# id tom
```

删除用户:userdel(user delete)

格式 userdel [-r] 用户名 #参数-r 表示同时删除该用户/home 下的用户文件夹

```
[root@server0 ~]#  userdel -r nsd01
[root@server0 ~]#  id nsd01
[root@server0 ~]# useradd  harry        #创建用户 harry
```

```
[root@server0 ~]# cd ~harry              #切换到 harry 用户家目录
[root@server0 harry]# userdel  tom       #删除用户 tom
[root@server0 harry]# cd ~tom            #用户已被删除，cd 失败
```

组帐号的管理

组基本信息存放在 /etc/group 文件

```
[root@server0 ~]# head -1 /etc/group
root:    x       :  0    :
```

组名:组的密码占位符号:本组的 ID：本组的成员列表

添加组：groupadd

格式：groupadd [-g] 组名 #参数-g 可以指定 GID

```
[root@server0 ~]# groupadd  stugrp  #创建组
[root@server0 ~]# grep 'stugrp'  /etc/group
```

添加/删除组成员：gpasswd

添加：gpasswd -a 用户名 组名

删除：gpasswd -d 用户名 组名

```
[root@server0 ~]# gpasswd -a kenji stugrp
[root@server0 ~]# grep 'stugrp' /etc/group
```

删除组：groupdel

格式：groupdel 组名

压缩:

Linux 独有的压缩工具：gzip、bzip2 、xz

```
[root@server0 opt]# gzip   a.txt    #将 a.txt 压缩为 a.txt.gz，源文件消失
[root@server0 opt]# bzip2  b.txt    #将 b.txt 压缩为 b.txt.bz2，源文件消失
[root@server0 opt]# xz   c.txt      #将 c.txt 压缩为 c.txt.xz，源文件消失
```

压缩方式对应扩展名：

gzip：.gz

bzip2：.bz2

xz：.xz

归档压缩工具:

tar 集成备份工具:

格式：tar [选项] tar 包的名字 被归档的文件 1 被归档的文件 2 被归档的文件 3.....

常见选项：

-c:创建归档

-x:释放归档

-z、-j、-J:调用.gz、.bz2、.xz 格式的工具进行处理

-P:保持归档内文件的绝对路径

-C:指定释放路径

-t:显示归档中的文件清单

-f:指定归档文件名称

注意：所有的操作都有-f 选项，且 f 必须放在所有选项的最后

```
[root@server0 opt]# tar -cPf  file01.tar /etc/passwd  /home/  /boot/
    #将 etc/passwd，/home/，/boot/三个文档进行归档
[root@server0 opt]# tar -zcPf /root/test01.tar.gz  /etc/passwd  /home/
/boot/  #归档同时压缩需要写对扩展名
[root@server0 opt]# tar -jcPf /root/test02.tar.bz2  /etc/passwd  /home/
/boot/
[root@server0 opt]# tar -JcPf /root/test03.tar.xz  /etc/passwd  /home/
/boot/
[root@server0 opt]# tar -tf /root/test03.tar.xz                  #不解压
查看其中内容
[root@server0 opt]# tar -xf /root/test01.tar.gz -C /mnt/         #指定解
压缩的位置为/mnt,该目录必须存在
[root@server0 opt]# tar -xf /root/test01.tar.gz                  #不指定
位置则解压到当前目录
```

zip 压缩(window 和 linux 通用)：

格式：zip （-r） 压缩文件 源文件

unzip 压缩文件 （-d 解压目录）

```
[root@desktop0 ~]# zip -r  my.zip  /var/log #把/var/log 目录给压缩，压缩到当
前目录下 my.zip
[root@desktop0 ~]# unzip my.zip              #把 my.zip 解压到当前目录
[root@desktop0 ~]# unzip my.zip -d /tmp      #把 my.zip 解压到/tmp
```

date：日期与时间

格式：

修改日期与时间：date -s "年-月-日 时:分:秒 "，如果时分秒不写则用 0 补全，最早不能早于 1970-1-1

显示当前时间：date

显示日期：date +%F

显示年：date +%Y

显示月：date +%m

显示日：date +%d

显示时间（无秒)：date +%R

显示时间（有秒）：date +%T

显示小时：date +%H

显示分：date +%M

显示秒：date +%S

```
[root@server0 opt]# date -s "2018-2-5  16:06"
[root@server0 opt]# date
2018 年  02 月  05 日  星期一  16:06:36  CST
[root@server0 opt]# date +'%Y-%m-%d %H:%M'
```

NTP 网络时间协议:Network Time Protocol

NTP 服务器为客户机提供标准时间

NTP 客户机需要与 NTP 服务器保持沟通

客户端：连接服务端同步

1.安装 chrony 客户端软件

```
[root@server0 opt]# yum -y install chrony    #系统默认安装此软件
[root@server0 opt]# rpm -q chrony
```

2.配置 chrony 客户端软件

```
[root@user ~]# cat /etc/chrony.conf
    server 0.rhel.pool.ntp.org iburst
    server 1.rhel.pool.ntp.org iburst
    server 2.rhel.pool.ntp.org iburst
    server 201.1.2.5 iburst                    #指定 ntp 时间同步服务器
```

3.重启 chrony 客户端软件服务

```
[root@user ~]# systemctl  restart  chronyd.service    #重启服务，等待若干秒后同步成功
[root@user ~]# systemctl  enable  chronyd.service     #设置开机自启服务
```

4.验证

```
[root@user ~]# date
[root@user ~]# timedatectl
```

服务端：采用的是分层设计，总层数限制在 15 层以内，防止误差过大

1.装包：`[root@vpn ~]# yum -y install chrony`

2.修改服务端配置文件：

```
[root@vpn ~]# vim /etc/chrony.conf
    server 0.rhel.pool.ntp.org iburst    #iburst 表示尽快同步
    allow 201.1.2.0/24                   #允许的 IP 或网段，系统默认全拒绝，只开放允许的主机
    deny 201.1.2.10/24                   #允许的范围内拒绝的 IP 或网段
    local stratum 10                     #设置 NTP 的当前层数
```

3.起服务：`[root@vpn ~]# systemctl restart chronyd.service`

计划任务

用途:按照设置的时间间隔为用户反复执行某一项固定的系统任务

系统服务:crond

日志文件:/var/log/cron

设置计划任务：crontab

```
格式:crontab  -e  [-u 用户名]        #可指定执行用户名，不指定则默认当前用户
        时间                         执行任务
    分   时   日   月   周          任务命令行(绝对路径)
```

*: 匹配范围内任意时间

,: 分隔多个不连续的时间点

-: 指定连续时间范围

/n: 指定时间频率,每 n 时间执行一次

示例：每分钟记录当前的系统时间，写入/opt/time.txt

```
[root@server0 /]# crontab  -e
 */2   *   *   *   *   date  >>  /opt/time.txt
[root@server0 /]# cat /opt/time.txt
```

权限管理：基本权限、附加权限、acl 访问控制权限

基本权限：

读取 r ： 允许查看内容，read

写入 w ： 允许修改内容，write

可执行 x: 允许运行和切换，execute

目录的 r 权限: 能够 ls 浏览此目录内容

目录的 w 权限: 能够执行 rm/mv/cp/mkdir/touch/等更改目录内容的操作(对子目录可以)

目录的 x 权限: 能够 cd 切换到此目录

文件的 r 权限: cat，less，head，tail

文件的 w 权限: vim 且能够保存

文件的 x 权限: Shell 脚本

附加权限（特殊权限）：

Set UID:

附加在属主的 x 位上

属主的权限标识会变为 s

通常应用于可执行文件,Set UID 可以让使用者具有文件属主的身份及部分权限

功能：传递所有者身份

```
[root@server0 /]# chmod u+s /public
```

Set GID:

格式：chmod g+s 目录名

附加在属组的 x 位上，属组的权限标识会变为 s，如果属组没有 x 权限，则变为 S

适用于目录,Set GID 可以使目录下新增的文档自动继承父目录的属组

功能：继承父目录所属组身份

```
[root@server0 /]# chmod g+s /nsd07
```

Sticky Bit:粘滞位，也称为 t 标签

附加在其他人的 x 位上

如果有 x 权限，其他人的权限标识会变为 t；如果没有 x 权限，则变成 T

适用于开放 w 权限的目录,可以阻止用户滥用 w 写入权限

功能：只能修改删除自己创建的文档，禁止操作别人的文档

```
[root@server0 ~]# chmod u=rwx,g=rwx,o=rwx,o+t /public
[root@server0 ~]# ls -ld /public
drwxrwxrwt. 2 root root 6 2月  12 09:35 /public
```

权限适用对象(归属)

所有者 u ： 拥有此文件/目录的用户,user

所属组 g ： 拥有此文件/目录的组,group

其他用户 o ： 除所有者、所属组以外的用户,other

[root@room9pc01 ~]# ls -ld
d r-x r-x --- . 29
root root 4096 2 月 6 16:33
-：文本文件 所属用户权限 所属组权限 其他用户权限 是否存在 acl 硬连接数 所属用户 所属组 大小，单位 b 最后修改时间
d：目录
l：快捷方式（链接文件）
b：块存储
c：字符模块

设置权限 chmod（change modify）
格式：chmod [-R] 归属关系+-=权限类别 文档路径 #-R：递归设置
使用数字修改权限：
普通权限：r=4,w=2,x=1
格式：chmod nnn 文件或目录
chmod 777 文件或目录-->结果：rwxrwxrwx
chmod 000 文件或目录-->结果：--------
chmod 77 文件或目录-->结果：-rwxrwx
[root@server0 ~]# chmod u-w /nsd01 #关闭所属用户写入权限
[root@server0 ~]# chmod g+w /nsd01 #开启所属组写入权限
[root@server0 ~]# chmod o+w /nsd01 #开启其他用户写入权限
[root@server0 ~]# chmod o= /nsd01 #关闭其他用户所有权限
[root@server0 ~]# chmod u=rwx,g=rx,o=rx /nsd01 #同时设置
附加权限：suid=4,sgid=2,sticky=1
格式：chmod nnnn 文件或目录
注：第 1 个 n 是特殊权限
chmod 7777 文件-->结果：rwsrwsrwt
chmod 1777 文件-->结果：rwxrwxrwt
chmod 6777 文件-->结果：rwsrwsrwx
当提示：权限不足(Permission denied)，排错方案
1.依次查看用户对于该文档所属的身份，acl>所有者>所属组>其他人，匹配到即停止查看
2.查看相应身份的权限

使用 chown 命令
chown [-R] 属主 文档...
chown [-R] :属组 文档...
chown [-R] 属主:属组 文档...
[root@server0 /]# chown harry:stugrp /nsd03 #设置/nsd03 目录所有者为 harry，所属组为 stugrp
[root@server0 /]# chown zhangsan /nsd03 #设置/nsd03 目录所有者为 zhangsan，所属组不变
[root@server0 /]# chown :root /nsd03 #设置/nsd03 目录所属组为 root，所有者不变
示例：利用 root 用户新建/nsd05 目录，并进一步完成下列操作

1.将属主设为 gelin01，属组设为 tarena 组

```
[root@server0 /]# useradd gelin01
[root@server0 /]# groupadd tarena
[root@server0 /]# chown gelin01:tarena  /nsd05
```

2.使用户 gelin01 对此目录具有 rwx 权限，其他人对此目录无任何权限

```
[root@server0 /]# chmod o=--- /nsd05
```

3.将用户 gelin02 加入 tarena，使其能进入、查看此目录

```
[root@server0 /]# gpasswd -a  gelin02  tarena
```

4.将 gelin01 加入 tarena 组，将 nsd05 目录的权限设为 rw-r-x---，测试 gelin01 用户能否进入此目录（不能进入）

```
[root@server0 /]# chmod u=rw,g=rx /nsd05
```

acl 访问控制列表：能够对个别用户、个别组设置独立的权限

setfacl：设置 acl 访问控制列表

getfacl：查看 acl 访问控制列表

格式：查看指定文档的 acl 访问 ACL 策略：getfacl 文档...

新增该指定文档用户访问 ACL 策略：setfacl -m u:用户名:权限类别 文档...

新增指定文档组访问 ACL 策略：setfacl -m g:组名:权限类别 文档...

删除指定文档 ACL 策略：setfacl -x u:用户名 文档...

删除指定文档所有的 ACL 策略：setfacl -b 文档...

参数：-R：递归设置

```
[root@server0 /]# setfacl -m u:zhangsan:rwx  /nsd11 #设置用户的 ACL 权限
[root@server0 /]# getfacl /nsd11                    #查看该目录 ACL 权限

[root@server0 /]# setfacl -x u:natasha  /nsd11      #删除指定的 ACL 权限
[root@server0 /]# setfacl -b /nsd11                 #删除所有的 ACL 权限
```

LDAP 认证，轻量级目录访问协议（Lightweight Directory Access Protoco）

由服务器来集中存储用户信息并向客户端提供的信息,存储方式，类似于 DNS 分层结构

提供的信息包括:用户名、密码

为一组客户机集中提供可登录的用户账号

本地用户：用户名：/etc/passwd 密码：/etc/shadow

网络用户：用户名、密码信息存储在 LDAP 服务端

客户端设置：

1.安装一个客户端 sssd 软件,与 LDAP 服务端沟通

```
[root@server0 /]# yum  -y  install  sssd
```

2.安装 authconfig-gtk 图形的工具，配置 sssd 软件

```
[root@server0 /]# yum  -y  install  authconfig-gtk
```

3.运行 authconfig-gtk 图形的工具

```
[root@server0 /]# authconfig-gtk
```

选择 LDAP

LDAP 搜索基础：dc=example,dc=com #指定服务端域名

LDAP 服务器：classroom.example.com #指定服务端主机名

勾选 TLS 加密

使用证书加密： http://classroom.example.com/pub/example-ca.crt

选择 LDAP 密码

4.重启客户端服务 sssd 服务，设置开机自启动

```
[root@server0 /]# systemctl restart sssd     #重启 sssd 服务
[root@server0 /]# systemctl enable sssd      #设置开机自启动
```

5.验证

```
[root@server0 ~]# grep 'ldapuser0' /etc/passwd      #/etc/passwd 中未找到用户 ldapuser0
[root@server0 ~]# id ldapuser0                      #可以找到用户 ldapuser0
```

家目录漫游，NFS（Network File System）网络文件系统

由 NFS 服务器将指定的文件夹共享给客户机

客户机将此共享目录 mount 到本地目录,访问此共享资源就像访问本地目录一样方便，类似于 EXT4、XFS 等类型,只不过资源在网上

客户端设置：

1.查看共享 classroom.example.com

```
[root@server0 ~]#  showmount  -e  classroom.example.com
Export list for classroom.example.com:
/home/guests  172.25.0.0/255.255.0.0
```

家目录所在路径：/home/guests，允许登陆的 IP 地址为：172.25.*.*/16

2.家目录漫游

访问共享内容,将服务端的共享文件夹数据,挂载到本地/mnt 以本地的/mnt 作为访问点

```
[root@server0 ~]# mount classroom.example.com:/home/guests   /home/guests
[root@server0 ~]# su - ldapuser0
```

使用 autofs 设置家目录漫游

```
[root@server0 ~]# yum install autofs -y     #安装 autofs
[root@server0 ~]# vim /etc/auto.master
    参照原本追加一行：/home/guests   /etc/auto.guests
[root@server0 ~]# vim /etc/auto.guests
    参照/etc/auto.misc 内容写入：* -rw classroom.example.com:/home/guests/&
        #‘&’代表前一个‘*’的内容
[root@server0 /]# systemctl restart autofs
[root@server0 ~]# su - ldapuser0
```

tmux 分离会话

快捷键：按 ctrl+b 进入快捷键模式

- d：将软件放入后台
- c：创建窗口
- w：查看会话列表
- p：前一个会话
- n：后一个会话
- %：左右分屏
- “：上下分屏
- x：退出分屏

&：退出

```
[root@proxy ~]# yum -y install tmux
[root@proxy ~]# tmux                          #开启软件
[root@proxy ~]# tmux kill-session -t 0        #杀死指定会话
```

硬盘分区管理

基础知识：每个扇区 512 字节

文件系统的作用：数据在空间中排列规则

硬盘的使用：识别硬盘 --> 分区规划 --> 格式化 --> 挂载使用

使用磁盘空间需要的命令：

1. 查看识别的磁盘 ： lsblk
2. 划分分区 ： fdisk
3. 刷新分区 ： partprobe
4. 格式化分区 ： mkfs.ext4 、mkfs.xfs 、blkid 查看格式化结果
5. 挂载使用 ： mount 、 df -h
6. 实现开机自动挂载 ： /etc/fstab

具体步骤：

1.识别硬盘

```
[root@server0 ~]# lsblk          #查看本机识别的硬盘
```

2.分区规划

MBR/msdos 分区模式（主引导记录分区方案）：最大支持 2.2TB

GPT 分区模式：最大支持 18EB（1EB=1024PB,1PB=1024TB0）分区的类型：主分区、扩展分区、逻辑分区

MBR 最多只能有 4 个主分区：1~4 个主分区,或者 3 个主分区+1 个扩展分区(可再分为 n 个逻辑分区)

扩展分区是一个容器，不能格式化

```
[root@server0 /]# fdisk /dev/vdb     #打开 fdisk 软件
   n 创建新的分区----->回车----->回车---->回车----->在 last 结束时 +2G
   p 查看分区表
   d 删除分区
   w 保存并退出
[root@server0 ~]# lsblk              #查看分区结果，大小，挂载点

[root@server0 ~]# ls  /dev/vdb[1-2]
```

3.分区格式化

EXT4 格式化：mkfs.ext4 分区设备路径

XFS 格式化：mkfs.xfs 分区设备路径

FAT 格式化：mkfs.vfat 分区设备路径

```
[root@server0 ~]# mkfs.ext4  /dev/vdb1
[root@server0 ~]# mkfs.xfs   /dev/vdb2
[root@server0 ~]# blkid  /dev/vdb1  #查看分区文件系统类型
[root@server0 ~]# blkid  /dev/vdb2
```

4.挂载使用

```
[root@server0 ~]# mkdir /part1
[root@server0 ~]# mount /dev/vdb1  /part1
[root@server0 ~]# mount /dev/vdb2  /part2
mount: 挂载点 /part2 不存在
[root@server0 ~]# mkdir /part2
[root@server0 ~]# mount /dev/vdb2  /part2
[root@server0 ~]# df -h   #查看正在挂载的设备信息
```

5.综合分区，实现开机自动挂载

注意：创建的第四个分区为拓展分区，将所有剩下的空间都划给第四个分区，以便后期再划分逻辑分区

配置文件 vim /etc/fstab

格式：

设备路径	挂载点	文件系统类型	参数	备份标记	检测顺序
/dev/vdb1	/part1	ext4	defaults	0	0
/dev/vdb2	/part2	xfs	defaults	0	0

6.验证：

```
[root@server0 ~]# df  -h
[root@server0 ~]# mount  -a
[root@server0 ~]# df  -h
```

mount -a 检测/etc/fstab 开机自动挂载配置文件格式是否正确；如果正确，则对当前没有挂载的设备进行挂载

LVM 逻辑卷

动态管理存储空间的方式

优点：

1.整合分散的空间

2.逻辑卷空间可以扩大

过程：将众多的物理卷，组成卷组，在从卷组中划分逻辑卷

创建逻辑卷：

1.创建卷组：

命令格式：vgcreate 卷组名字 分区路径

```
[root@server0 ~]# vgcreate  systemvg  /dev/vdc[1-2]   #同时创建物理卷和卷组，支持通配符
[root@server0 ~]# pvs                          #查看物理卷信息
[root@server0 ~]# vgs                          #查看卷组信息
```

2.从卷组中划分逻辑卷

命令格式： lvcreate -L 大小 -n 逻辑卷名字 卷组名字

```
[root@server0 ~]# lvcreate -L 16G -n mylv systemvg    #创建一个 16G 的逻辑卷
[root@server0 ~]# lvcreate -l 16 -n mylv systemvg     #创建一个大小为 16 个 PE 的逻辑卷，
[root@server0 ~]# lvs
```

3.使用逻辑卷

```
[root@server0 ~]# mkfs.ext4 /dev/systemvg/mylv        #用 EXT4 格式进行格
```

式化

```
[root@server0 ~]# mkdir /test
[root@server0 ~]# vim /etc/fstab                #设置自动挂载
 /dev/systemvg/mylv   /test   ext4   defaults  0   0
[root@server0 ~]# df -h
[root@server0 ~]# mount -a                      #激活自动挂载
[root@server0 ~]# df -h
```

修改 PE 的方式（基本用不到）：(PE：卷组划分空间的单位)

```
[root@server0 ~]# vgcreate  -s 16M systemvg  /dev/vdc[1-2]      #若需要更大的 PE，需要在创建卷组时设置，且设置的 PE 值必须是 2 的幂次
[root@server0 ~]# vgchange -s  512KB  卷组名     #默认为 4M，只能改成更小的 PE，最小 512B
[root@server0 ~]# vgdisplay  systemvg           #显示卷组详细信息查看 PE 大小
```

扩展逻辑卷：

1.判断卷组是否有足够的剩余空间

如果不够：扩展卷组

```
[root@server0 ~]# vgextend   systemvg  /dev/vdc3        #加入新的物理卷拓展卷组空间
[root@server0 ~]# vgs
```

2.扩展逻辑卷空间的大小

```
[root@server0 ~]# lvs
[root@server0 ~]# vgs
[root@server0 ~]# lvextend -L 18G /dev/systemvg/mylv     #将逻辑卷拓展到 18G
[root@server0 ~]# lvextend -L +2G /dev/systemvg/mylv     将逻辑卷在原基础上拓展 2G
[root@server0 ~]# vgs
[root@server0 ~]# lvs
```

3.扩展文件系统的大小

```
   ext4 :  resize2fs
   xfs :  xfs_growfs
[root@server0 ~]# df -h
[root@server0 ~]# resize2fs /dev/systemvg/mylv          #刷新文件系统
[root@server0 ~]# df -h
```

逻辑卷缩减步骤（不建议使用，有一定概率会损坏数据）：

1.缩减文件系统

2.缩减空间

删除逻辑卷

流程：删除逻辑卷（lv）---->删除卷组（vg）---->删除物理卷（pv）

lvremove vgremove pvremove

重定向输出:

```
>  :  只收集前面命令的正确输出，将其写入文本文件
2> :  只收集前面命令的错误输出，将其写入文本文件
```

&> ： 收集前面命令的正确与错误输出，将其写入文本文件

[root@server0 ~]# cat /opt/1.txt /etc > /opt/a.txt

[root@server0 ~]# cat /opt/1.txt /etc 2> /opt/a.txt

[root@server0 ~]# cat /opt/1.txt /etc &> /dev/null #/dev/null ： 黑洞设备，专门收集无用的输出信息

用户密码安全：chage 工具(change age)

常用选项：

-d 0 下次登陆时强制要求修改密码，将/etc/shadow 的第三列的数字设为0

-l 查看密码有效期相关信息

-E 年-月-日 指定帐号失效日期，值在/etc/shadow 的第八列

-E -1 取消指定日期失效，删除/etc/shadow 第八列的值

```
[root@proxy ~]# chage -d 0 lisi
[root@proxy ~]# chage -l lisi
[root@proxy ~]# chage -E 2018-6-1 lisi
[root@proxy ~]# chage -E  -1 lisi
```

修改配置文件，只对新建用户生效：[root@proxy ~]# vim /etc/login.defs +25

PASS_MAX_DAYS 两次修改密码的最长间隔时间，默认为 99999，值在/etc/shadow 的第五列

PASS_MIN_DAYS 两次修改密码的最短间隔时间，0 表示无限制，值在/etc/shadow 的第四列

PASS_MIN_LEN 密码最短长度

PASS_WARN_AGE 密码过期前发出警告的天数，值在/etc/shadow 的第六列

用户帐号安全：passwd 命令

常用选项： -l 锁定帐号，等同于在/etc/shadow 中 lisi 的密码前加!!

-u 解锁帐号，等同于将/etc/shadow 中 lisi 的密码前的!!删除，或者修改 lisi 密码

-S 查看帐号状态

```
[root@proxy ~]# passwd -l lisi
[root@proxy ~]# passwd -S lisi
[root@proxy ~]# passwd -u lisi
```

伪装登陆提示信息：修改配置文件

```
[root@proxy ~]# vim /etc/issue          #修改本地登陆提示
[root@proxy ~]# vim /etc/issue.net      #修改远程登陆提示，默认不显示
[root@proxy ~]# vim /etc/ssh/sshd_config +123
    Banner /etc/issue.net
```

锁定文件：chattr,lsattr

控制方式：+，-，=

属性：

i 不可改变，用于重要文件，防止误删除

a 仅可追加，用于日志文件

```
[root@proxy ~]# chattr +i host
[root@proxy ~]# lsattr host
[root@proxy ~]# chattr +a host
[root@proxy ~]# chattr -ia host
[root@proxy ~]# chattr =i host
[root@proxy ~]# chattr =ai host
[root@proxy ~]# chattr = host
```

Linux 系统安全保护：Security-Enhanced Linux

美国 NSA 国家安全局主导开发,一套增强 Linux 系统安全的强制访问控制体系

集成到 Linux 内核(2.6 及以上)中运行

RHEL7 基于 SELinux 体系针对用户、进程、目录和文件提供了预设的保护策略,以及管理工具

SELinux 控制每个软件能否读写某些资料

SELinux 有安全上下文件

配置文件：/etc/selinux/config

模式控制：

7:SELINUX=enforcing\permissive\disabled

策略集：

targeted:针对网络程序定义规则,例如 http,ftp...

minimum:只针对某几个程序定义规则

mls:尽可能对市面上的程序定义规则,没定义的不让启动

12:SELINUXTYPE=targeted\minimum\mls

设置布尔值：

在 targeted 策略的布尔值设置中，默认禁止某些服务，如 SMB 的读写，FTP 的匿名写入和完全访问

```
[root@proxy ~]# setsebool ftpd_anon_write on
[root@proxy ~]# setsebool ftpd_full_access on
```

查看属性：ls -Z

属性构成： 用户：角色：访问类型：选项

```
[root@proxy ~]# ls -lZ /var/ftp/
```

指定访问类型：-t

```
[root@proxy ~]# chcon -tR public_content_t /var/ftp/b                #仅修改访问类型
[root@proxy ~]# restorecon /var/ftp/a.pub                            #仅重置访问类型
[root@proxy ~]# chcon --reference=/var/ftp/a.pub /var/ftp/love.gpg   #同步所有属性
```

操作规律：

移动：原有的访问类型不变

复制、新建：自动继承目标目录的访问类型

图形界面查看 SELinux 问题：需要装包 setroubleshoot-server

```
[root@server0 ~]# sealert -b
```

SELinux 的运行模式：

enforcing(强制)<----->permissive(宽松) 可以不重启系统切换

↓ 切换需要重启系统

disabled(彻底禁用)

切换运行模式：

临时切换:setenforce 1|0 #enforcing | permissive

[root@server0 ~]# getenforce #查看当前 SELinux 运行模式

[root@server0 ~]# setenforce 0 #修改当前 SELinux 运行模式为 permissive

[root@server0 ~]# getenforce

例如：httpd 这个软件，只能读写标签是 httpd_sys_content_t 的文档

[root@server0 ~]# ls -Z #查看安全标签，每个文件或目录都有一个标签，这个标签控制了哪些程序能读什么

[root@server0 ~]# echo "nb" > nb.html

[root@server0 ~]# ls -Z nb.html

[root@server0 ~]# mv nb.html /var/www/html/ #mv 移动文件，标签会被保留，cp 拷贝文件，标签不会保留

[root@server0 ~]# firefox http://server0.example.com/nb.html #因为 SELinux 的标签不对，所有无法访问

[root@server0 ~]# sealert -b #图形界面查看 SELinux 问题：需要装包 setroubleshoot-server

[root@server0 ~]# chcon -R -t httpd_sys_content_t /var/www/html/nb.html

[root@server0 ~]# firefox http://server0.example.com/nb.html #修改标签后，再次访问就可以成功

搭建基本的 FTP（文件传输协议）

1.安装服务端软件：vsftpd

2.配置文件：/etc/vsftpd/vsftpd.conf

12:anonymous_enable=YES #开启匿名访问

19:write_enable=YES #开启可写入

29:anon_upload_enable=YES #允许上传文件

33:anon_mkdir_write_enable=YES #允许上传目录

3.启动 vsftpd 服务，设置开机自起

4.默认共享路径：/var/ftp

5.安装客户端软件：ftp

6.连接服务端：ftp 192.168.4.5

cd 调整 ftp 的目录

lcd 调整本机的目录

put 上传

防火墙策略管理

作用：隔离

分类：硬件防火墙、软件防火墙

Linux 软件防火墙：

系统服务:firewalld

管理工具:firewall-cmd（命令）、firewall-config（图形）

命令 ： firewall-cmd 选项 参数

常用选项：

设置为永久配置：--permanent

要设置的区域：--zone=

查看区域规则：--list-all

新增端口转发：--add-forward-port=

删除端口转发：--remove-forward-port=

增加端口规则：--add-port=

删除端口规则：--remove-port=

新增服务：--add-service=

删除服务：--remove-service=

新增 IP：--add-source=

删除 IP：--remove-source=

增加网卡规则：--add-interface=

移除网卡规则：--remove-interface=

获取默认区域：--get-default-zone

设置默认区域：--set-default-zone=

重新加载防火墙：--reload

根据所在的网络场所区分,预设保护规则集：

public:仅允许访问本机的 sshd 等少数几个服务

trusted:允许任何访问

block:阻塞任何来访请求

drop:丢弃任何来访的数据包

防火墙判定进入哪一个区域的规则：

1.查看客户端请求数据包中，源 IP 地址，查看自己所有的区域中哪一个区域有该源 IP 地址的策略，则进入哪一个区域。规则匹配：匹配即停止

2.如果不匹配规则 1，则进入默认区域

宽松方式 1（黑名单）：默认区域为 trusted，将想要拒绝的源 IP 地址放入到 block 或 drop

严格方式 2（白名单）：默认区域为 block 或 drop，将想要允许的源 IP 地址放入到 trusted

[root@server0 ~]# firewall-cmd --get-default-zone #查看默认区域

[root@server0 ~]# firewall-cmd --zone=public --list-all #查看 public 区域的规则

[root@server0 ~]# firewall-cmd --set-default-zone=public #修改默认区域

[root@server0 ~]# firewall-cmd --zone=public --list-all #查看 public 区域规则

[root@server0 ~]# firewall-cmd --zone=public --add-service=ftp #为区域 public 中添加 ftp 服务

[root@desktop0 ~]# firefox ftp://172.25.0.11 #可以访问

[root@desktop0 ~]# firefox 172.25.0.11 #不可以访问

[root@server0 ~]# firewall-cmd --zone=public --add-service=http #为区域中添加 http 服务

[root@desktop0 ~]# firefox 172.25.0.11 #可以访问

[root@server0 ~]# firewall-cmd --permanent --zone=public --add-service=http

[root@desktop0 ~]# firewall-cmd --reload #重新加载防火墙配置，使永久配置生效

[root@desktop0 ~]# firewall-cmd --permanent --zone=block --add-source=172.25.0.10 #禁 IP

端口：数字（编号）：标识进程或程序

数据包包含：源 IP 地址 目标 ip 地址 数据 目标端口号

http ： 默认端口 80 ，加密端口 443

FTP ： 默认端口 21

端口转发

[root@desktop0 ~]# firewall-cmd --add-forward-port=port=5423:proto=tcp:toport=80 #设置端口为 5423，且使用 TCP 协议的数据传入 80 端口

防火墙：保护、隔离

firewalld 的底层调用包过滤防火墙 iptables

iptables 基本策略:

1.安装：[root@proxy ~]# yum -y install iptables-services

2.框架：

4 个表（区分大小写）：filter 表（数据过滤表）、nat 表（地址转换表）、raw 表（状态跟踪表）、mangle 表（包标记表）

5 个链（区分大小写）：INPUT 链（入站规则）、OUTPUT 链（出站规则）、FORWARD 链（转发规则）、PREROUTING 链（路由前规则）、POSTROUTING 链（路由后规则）

3.规则链内匹配规则：顺序比对，匹配即停止，LOG 除外，若无匹配项，则按默认策略处理

4.命令：iptables -t 表名 选项 链名 条件 -j 目标操作

如果不指定表，默认为 filter 表

如果不指定链，默认为表的所有链，通常用于查看规则时

除设置默认策略外，必须指定匹配条件

大部分选项、链名、目标操作用使用大写字母，其余都使用小写

常用选项：

类别	选项	用法
通用匹配	协议匹配	-p 协议名
	地址匹配	-s 源地址、-d 目标地址
	接口匹配	-i 收数据的网卡、-o 发数据的网卡
隐含匹配	端口匹配	--sport 源端口、--dport 目标端口
	ICMP类型匹配	--icmp-type ICMP类型

类别	选项	用途
添加规则	-A	在链的末尾追加一条规则
	-I	在链的开头（或指定序号）插入一条规则
查看规则	-L	列出所有的规则条目
	-n	以数字形式显示地址、端口等信息
	--line-numbers	查看规则时，显示规则的序号
删除规则	-D	删除链内指定序号（或内容）的一条规则
	-F	清空所有的规则
默认策略	-P	为指定的链设置默认规则

目标操作：

ACCEPT：允许通过/放行

DROP：直接丢弃，不给出任何回应

REJECT：拒绝通过，必要时会给出提示

LOG：记录日志，然后传给下一条规则

常用条件：

5.查看规则：

```
[root@proxy ~]# iptables -L                 #查看数据过滤表所有链规则
[root@proxy ~]# iptables -L INPUT           #查看数据过滤表的入站规则
    Chain PREROUTING (policy ACCEPT)        #policy 表示默认规则为允许通过
    target  prot        opt     source      destination
    规则    协议类型            源地址      目的地址
[root@proxy ~]# iptables -t nat -L          #查看地址转换表所有链规则
```

6.创建规则：

```
[root@proxy ~]# iptables -t filter -A INPUT -p tcp -j ACCEPT#在 filter 表中的 INPUT 链的末尾，写入允许 TCP
[root@proxy ~]# iptables -I INPUT -p udp -j ACCEPT          #在 filter 表中的 INPUT 链的开头，写入允许 UDP
[root@proxy ~]# iptables -I INPUT 2 -p icmp -j ACCEPT       #在 filter 表中的 INPUT 链的第 2 行，写入允许 ICMP
```

7.删除规则：

```
[root@proxy ~]# iptables -D INPUT 2     #删除 filter 表中的 INPUT 链的第 2 行
[root@proxy ~]# iptables -nL            #查看规则，以数字显示地址和端口
```

[root@proxy ~]# iptables -F　　#删除 filter 表中所有链的规则

[root@proxy ~]# iptables -F -t nat　　#删除 nat 表中所有链的规则

8.设置默认规则：

[root@proxy ~]# iptables -P INPUT DROP　#设置 filter 表中的 INPUT 链的默认规则为丢弃，若无其他规则会导致远程断开

9.保存规则：

配置文件：/etc/sysconfig/iptables-config

[root@proxy ~]# iptables-save

iptables 网络防火墙：

1.开启服务器路由转发：

临时设置：[root@proxy ~]# echo 1 > /proc/sys/net/ipv4/ip_forward

永久设置：[root@proxy ~]# echo net.ipv4.ip_forward=1 >> /etc/sysctl.conf

2.创建规则：

[root@proxy ~]# iptables -A INPUT -p tcp --dport 80 -j REJECT　　#禁止所有主机访问 80 端口

[root@proxy ~]# iptables -A INPUT -s 192.168.4.100 -j REJECT　　#禁止特定主机访问本机

[root@proxy ~]# iptables -A INPUT -s 192.168.4.100 -p tcp --dprot 80 -j REJECT　　#结合上面两个

[root@proxy ~]# iptables -A INPUT -s 192.168.4.100 -p tcp --dport 80 -i eth0 -j REJECT　#增加指定网卡

[root@proxy ~]# iptables -A INPUT -s 192.168.4.0/24 -p tcp --dport 22 -i eth0 -j REJECT　#禁用特定网段远程

3.设置转发规则,阻止特定主机访问防火墙后的网页：

[root@proxy ~]# iptables -A FORWARD -s 192.168.4.100 -p tcp --dport 80 -j DROP

4.禁止 ping：

禁止所有 ping 包：[root@proxy ~]# iptables -A INPUT -p icmp -j DROP

仅禁止其他主机 ping 本机：[root@proxy ~]# iptables -I INPUT -p icmp --icmp-type echo-request -j DROP

注：可以通过 iptables -p icmp --help 查询。echo-request 表示发出的 ping 包，echo-reply 表示返回 ping 包

5.filter 总结：

INPUT：数据包的目的 IP 地址是防火墙，这个数据包是防火墙接收的

OUTPUT：数据包的源 IP 地址是防火墙，这个数据包是防火墙发送的

FORWARD：数据包的源地址和目的地址都不是防火墙，防火墙会检查这个链里的规则，这个规则可以过滤 nat 的数据包

iptables 扩展策略：

语法：iptables 选项 链名称 -m 扩展模块 --具体扩展条件 -j 动作

1.根据 MAC 地址过滤：

匹配 MAC 地址：-m mac --mac-source XX:XX:XX:XX:XX:XX

[root@proxy ~]# iptables -A INPUT -m mac --mac-source 52:54:00:4e:66:ea -p tcp --dport 22 -j DROP

2.设置多端口过滤：

匹配多端口：-m multiport --sports 源端口 /--dports 目标端口

[root@proxy ~]# iptables -A INPUT -p tcp -m multiport --dport 20:22,25,80,110,143 -j DROP

3.根据 IP 范围设置规则：

设定 IP 范围：-m iprange --src-range/--dst-range ip[-ip]

[root@proxy ~]# iptables -A INPUT -m iprange --src-range 192.168.4.100-192.168.4.200 -p tcp --dport 22 -j DROP

iptables 进行源地址转换 SNAT(Source Network Address Translation)：需要将规则写入 POSTROUTING

将 192.168.4.0 网段的主机通过地址转换变成 192.168.2.5：

[root@proxy ~]# iptables -t nat -A POSTROUTING -s 192.168.4.0/24 -j SNAT --to-source 192.168.2.5

仅当 192.168.4.0 网段的主机访问外网的网页时，才将地址转换为 192.168.2.5：

[root@proxy ~]# iptables -t nat -A POSTROUTING -s 192.168.4.0/24 -p tcp --dport 80 -j SNAT --to-source 192.168.2.5

将 192.168.4.0 网段的主机智能转化为动态 IP，masquerade 的意思是伪装：

[root@proxy ~]# iptables -t nat -A POSTROUTING -s 192.168.4.0/24 -j MASQUERADE

traceroute URL :可以查看访问该网址经过的路由

配置聚合连接\网卡绑定\链路聚合：

模式：

热备份(activebackup)：连接冗余(活跃状态、备份状态)

轮询式(roundrobin)：流量负载均衡

配置：eth1：team0-slave + eth2：team0-slave =>虚拟网卡:team0:master

注意：旧版本使用 bond

制作网卡绑定

1.制作虚拟网卡 team0 #"runner": {"name": "activebackup"} 参考 man teamd.conf 全文查找/example 按 n 跳转匹配项

nmcli connection add type team autoconnect yes con-name team0 ifname team0 config '{"runner": {"name": "activebackup"}}'

连接 添加 类型 聚合 每次开机自动启用 配置文件命名 显示的名字为 内部成员工作模式 热备份

2.为 team0 添加成员

nmcli connection add type team-slave con-name team0-1 ifname eth1 master team0

连接 添加 类型 聚合的内部成 内部成员命名 网卡原名 设置主设备

nmcli connection add type team-slave con-name team0-2 ifname eth2 master team0

3.配置 team0 的 ip 地址与激活

nmcli connection modify team0 ipv4.method manual ipv4.addresses 192.168.1.1/24 connection.autoconnect yes #配置网卡IP
nmcli connection up team0 #激活team0网卡
nmcli connection up team0-1 #激活team0-1成员
nmcli connection up team0-2 #激活team0-2成员

制作网卡错误的解决办法：删除后重新执行以上步骤

```
[root@server0 ~]# nmcli connection delete team0
[root@server0 ~]# nmcli connection delete team0-1
[root@server0 ~]# nmcli connection delete team0-2
```

专用于team测试查看的命令

```
[root@server0 ~]# teamdctl team0 state     #查看team0信息
[root@server0 ~]# ifconfig eth1 down       #禁用eth1网卡
[root@server0 ~]# teamdctl team0 state     #查看team0信息
[root@server0 ~]# ifconfig eth1 up         #开启eth1网卡
```

team配置文件存放路径：/etc/sysconfig/network-scripts/ifcfg-team0

配置SMB共享（跨平台的共享：Windows 与 Linux）
Samba 软件项目
用途:为客户机提供共享使用的文件夹
协议:SMB(TCP 139)、CIFS(TCP 445)
所需软件包:samba
系统服务:smb

客户端访问服务端资源：
1.服务本身访问控制
2.目录的本地权限
3.SELinux访问控制
4.防火墙策略控制

服务端：设置只读共享
1.安装软件包:samba

```
[root@server0 ~]# yum -y install samba
```

2.建立Samba共享帐号

```
[root@server0 ~]# useradd harry
[root@server0 ~]# useradd kenji
[root@server0 ~]# useradd chihiro
[root@server0 ~]# pdbedit -a harry      #将本地用户harry设置为Samba共享帐号
[root@server0 ~]# pdbedit -a kenji
[root@server0 ~]# pdbedit -a chihiro
```

```
[root@server0 ~]# pdbedit -L              #显示本地都有那些 Samba 共享帐号
```

3. 修改配置文件设置 Samba 共享

```
[root@server0 ~]# mkdir  /common
[root@server0 ~]# echo  123  >  /common/1.txt
[root@server0 ~]# ls  /common/
[root@server0 ~]# vim /etc/samba/smb.conf
    workgroup = STAFF                     #设置工作组
  [common]                                #设置共享名
    path = /common                        #设置共享的实际路径
```

4.重起 smb 服务，设置开机自起

```
[root@server0 ~]# systemctl restart smb #重起服务
[root@server0 ~]# systemctl enable smb  #设置开机自起
```

5. SELinux 设置布尔值（功能的开关）

```
[root@server0 ~]# getsebool  -a  |  grep  samba          #查看所有布尔值
[root@server0 ~]# setsebool -P samba_export_all_ro on    #修改布尔值的只读开
```

关，-P 表示写入内核，实现永久设置

6.设置防火墙

```
[root@server0 ~]# firewall-cmd  --set-default-zone=trusted
```

服务端：设置可读写共享

7.修改配置文件，声明 chihiro 可写

```
[root@server0 ~]# vim /etc/samba/smb.conf
      [devops]
        path = /devops
        write list = chihiro              #设置 chihiro 可以读写
```

8.设置本地权限

```
[root@server0 ~]# setfacl -m  u:chihiro:rwx  /devops/
[root@server0 ~]# getfacl  /devops/
```

9.修改 SELinux 功能开关

```
[root@server0 ~]# getsebool -a | grep samba
[root@server0 ~]# setsebool samba_export_all_rw on
```

10.重启服务

```
[root@server0 ~]#  systemctl restart smb
```

客户端操作：只读和读写都是一样操作

1.设置防火墙

```
[root@desktop0 ~]# firewall-cmd  --set-default-zone=trusted
```

2.安装软件包:samba-client

```
[root@desktop0 ~]# yum -y install samba-client
```

3.查看对端的共享名

```
[root@desktop0 ~]# smbclient  -L  172.25.0.11
Enter root's password:          #按回车跳过
```

4. 访问对端的共享

```
[root@desktop0 ~]# smbclient  -U  harry  //172.25.0.11/common
Enter harry's password:
Domain=[STAFF] OS=[Unix] Server=[Samba 4.1.1]
smb: \>
```

客户端使用挂载点访问

1.创建挂载点

```
[root@desktop0 ~]# mkdir  /mnt/nsd
```

2.所需软件包:cifs-utils

```
[root@desktop0 ~]# yum  -y install  cifs-utils
[root@desktop0 ~]# mount -o user=harry,pass=123  //172.25.0.11/common
/mnt/nsd/
[root@desktop0 ~]# vim  /etc/fstab
 //172.25.0.11/common   /mnt/nsd   cifs
defaults,user=harry,pass=123,_netdev   0  0 #_netdev : 标识本设备为网络设备（先启动网络服务具备 ip 地址等网络参数后，再进行挂载本设备）
                                        cifs：samba 共享设备的网络文件系统
[root@desktop0 ~]# mount  -a       #检测/etc/fstab 是否书写正确
[root@desktop0 ~]# df -h
```

多用户（multiuser）的 samba 共享，专为普通用户设计，BUG 多

SMB 客户端的 multiuser 挂载技术

管理员只需要作一次挂载，管理员无法访问挂载点，普通用户不切换新的用户也无法访问

客户端在访问挂载点时,若需要不同权限,可以临时切换为新的共享用户(无需重新挂载)

实现方式

1)挂载 SMB 共享时启用 multiuser 支持

2)使用 cifscreds 临时切换身份

操作：修改开机自动挂载配置文件，添加参数

multiuser,提供对客户端多个用户身份的区分支持

sec=ntlmssp,提供 NT 局域网管理安全支持

```
[root@desktop0 ~]# vim /etc/fstab
//172.25.0.11/devops /mnt/dev cifs
```

```
defaults,user=kenji,pass=123,_netdev,multiuser,sec=ntlmssp 0 0
        [root@desktop0 ~]# mount -a
        [root@desktop0 ~]# df -h
        [root@desktop0 ~]# su - student
        [student@desktop0 ~]$ cifscreds add -u chihiro 172.25.0.11  #提交新的身
份
        [student@desktop0 ~]$ ls /mnt/dev/
        [student@desktop0 ~]$ touch /mnt/dev/haha.txt
        [student@desktop0 ~]$ exit
```

配置 NFS 共享（Linux 内部）：Network File System（网络文件系统）

用途:为客户机提供共享使用的文件夹

协议:NFS(TCP/UDP 2049)、RPC(TCP/UDP 111)

所需软件包: nfs-utils

系统服务: nfs-server

使用 DRBD+Heartbeat+NFS 实现高可用文件共享存储

服务端：

1.所需软件包: nfs-utils

```
[root@server0 ~]# rpm -q nfs-utils
```

2.修改 /etc/exports

文件夹路径 客户机地址(权限) 客户机地址(权限) .. ..

```
[root@server0 ~]# vim /etc/exports
/public    *(ro)                    #设置所有用户只读/public
[root@server0 ~]# mkdir  /public
[root@server0 ~]# systemctl restart  nfs-server
[root@server0 ~]# systemctl enable  nfs-server
```

客户端：参照家目录挂载

```
[root@desktop0 ~]# mkdir /mnt/nfs
[root@desktop0 ~]# showmount -e 172.25.0.11     #查看有哪些 nfs 共享
[root@desktop0 ~]# vim /etc/fstab               #实现开机自动挂载
172.25.0.11:/public  /mnt/nfs   nfs   defaults,_netdev  0  0
[root@desktop0 ~]# mount -a
[root@desktop0 ~]# df -h
```

存储：根据不同的应用环境通过采用合理、安全、有效的方式将数据保存到某些介质上并能保证有效访问

目标：

数据临时或长期驻留的物理媒介

保证数据完整、安全存放的方式或行为

技术分类：

SCSI(Small Computer System Interface)小型计算机系统接口：

作为输入/输出接口

主要用于硬盘、光盘、磁带机等设备

DAS(Direct Attached Storage)直连式存储：

将存储设备通过 scsi 接口或光纤通道直接连接到计算机上

不能实现数据与其他主机的共享

占用服务器操作系统资源,如 CPU、IO 等

数据量越大,性能越差

NAS(Network Attached Storage)网络附属存储：

专用的数据存储服务器,本质上就是共享文件夹

以数据为中心,将存储设备与服务器彻底分离,集中管理数据

设防带宽、提高性能、降低总拥有成本、保护投资

通过 TCP/IP 协议访问数据

SAN(Storage Area Network)存储区域网络：

通过光纤交换机、光纤路由器、光纤集线器等设备将磁盘阵列、磁带等存储设备与相关服务器连接起来,形成高速专网网络

组成部分：

连接：路由器、光纤交换机

接口：scsi、fc

通信协议：IP、scsi

FC(Fibre Channel)光纤通道：

适用于前兆数据传输的、成熟而安全解决方案

提供更高的数据传输速率、更远的传输距离、更多的设备连接支持以及更稳定的性能、更简易的安装

主要组件：光纤、HBA(Host Bus Adapter)主机总线适配器、FC 交换机

拓扑结构：点到点(point to point)、仲裁循环(arbitrated loop)、交换结构(switched fabric)

iSCSI 共享（磁盘共享)internet scsi

概念:

backstore：后端真正的存储设备(实物)

target:共享名(虚拟)

lun:绑定、关联存储设备

target 的共享名有要求[要符合 iqn 规范]:iqn.年-月.反转域名.字符串:子字符串

如:iqn.2018-02.com.example:data

配置 iscsi：

1.在 server0 上准备一个磁盘分区(vdb1)3G

2.安装软件包，修改配置

```
[root@server0 ~]#  yum -y install targetcli
[root@server0 ~]#  targetcli
/>  backstores/block create back_store /dev/vdb1    #把刚刚分的分区加入后端存储
/>  iscsi/ create iqn.2018-02.com.example:data  #创建一个 iscsi 共享（共享名称)
```

注意：创建 iscsi 共享名时如果提示：WWN not valid as: iqn, naa, eui，则说明名称不符合规则

/> iscsi/iqn.2018-02.com.example:data/tpg1/luns create /backstores/block/back_store #把共享名和后端的设备通过lun关联在一起

/> iscsi/iqn.2018-02.com.example:data/tpg1/acls create iqn.2018-02.com.example:desktop0 #创建了一个访问的口令,以后仅知道口令的客户端才可以访问共享

/> iscsi/iqn.2018-02.com.example:data/tpg1/portals create 172.25.0.11 #以后客户端访问本机的172.25.0.11的3260端口就可以访问到共享

/> saveconfig

/> exit

[root@server0 ~]# firewall-cmd --set-default-zone=trusted

3.设置target开机自启动

4.客户端访问共享

[root@server0 ~]# yum -y install iscsi-initiator-utils #在实验环境可以跳过，生产环境需要确认

[root@server0 ~]# vim /etc/iscsi/initiatorname.iscsi

InitiatorName=iqn.2018-02.com.example:desktop0 #注意在这里给客户端配置访问口令，一定要与服务器的ACL一致

[root@server0 ~]# man iscsiadm #搜索/example

[root@server0 ~]# iscsiadm --mode discoverydb --type sendtargets --portal 172.25.0.11 --discover #发现对方服务器上的共享

[root@server0 ~]# iscsiadm --mode node --targetname iqn.2018-02.com.example:data --portal 172.25.0.11:3260 --login #挂载服务器上的共享

PS：如果挂载提示authentication，表示口令不对

5.重启iscsid并设置开机自启动

[root@server0 ~]# systemctl restart iscsid

iSCSI(Internet Small Computer System Interface)互联网小型计算机系统接口：

IETF制定的标准,将SCSI数据块映射为以太网数据包

一种给予IP Storage理论的新型存储技术

将存储行业广泛应用的SCSI接口技术与IP网络结合

可以在IP网络上创建san

由IBM下属的两大研发机构加利福尼亚AImaden和以色列Haifa研究中心共同开发的

优点：

基于IP协议技术的标准

允许网络在TCP/IP上传输SCSI命令

IP SAN相对FC SAN投资更低

解决了传输效率、存储容量、兼容性、开发性、安全性等方面的问题

传输无限制

客户端：

iscsi initiator：软件,成本低,性能低

iscsi HBA：硬件,性能好,成本高

存储设备：iscsi Target

由于ext4和xfs是单节点文件系统,如果多个节点同时挂载,就会损坏文件系统,使得数据丢失

配置iscsi用于存放数据库文件：挂载iscsi步骤忽略

```
[root@vh02 ~]# yum -y install mariadb-server
[root@vh02 ~]# mkfs.xfs /dev/sda1
[root@vh02 ~]# mount /dev/sda1 /var/lib/mysql/
[root@vh02 ~]# chown mysql: /var/lib/mysql/
[root@vh02 ~]# systemctl start mariadb.service
```

Udev：动态管理设备文件的方法

例如：主机连接U盘出现新的文件，移除U盘时，文件消失

作用：从内核收到添加、移除硬件时，udev会分析：/sys目录下信息和/etc/udev/rules.d/目录中的规则

Udev可以为设备改名,或执行操作：

在/etc/udev/rules.d/目录中创建规则文件

接入新设备,如果满足规则,则会按管理员的规则应用

udev的主配置文件是：/etc/udev/udev.conf

规则文件位置及格式：etc/udev/rules.d/数字-名字.rules

规则的作用：管理员指定硬件满足什么条件时,执行什么操作

规则格式：关键字+操作符+匹配,注意：所有的关键字在一个规则中只能使用一次

操作符：

==表示匹配

!=表示不匹配

+=表示额外增加

udev的主配置文件是：/etc/udev/udev.conf

示例：

1.找到硬盘在/sys目录中的说明路径：

```
[root@vh03 ~]# udevadm info --query=path --name=/dev/sda1
    /devices/platform/host2/session1/target2:0:0/2:0:0:0/block/sda/sda1
```

2.根据上一步查询到的路径,查询硬盘的硬件信息：

```
[root@vh03 ~]# udevadm info --query=all --attribute-walk --path=/devices/platform/host2/session1/target2:0:0/2:0:0:0/block/sda/sda1
```

3.在硬件信息中找到该类设备独有的内容：

```
SUBSYSTEMS=="scsi"
```

4.编写规则文件：

ACTION：行为,add表示接入时

KERNEL：没有人为干预时,内核起的名

SYMLINK：创建快捷方式

常用替代变量：

%k：内核识别出来的设备名

%n：设备的内核编号

%p：设备路径

```
[root@vh03 ~]# vim /etc/udev/rules.d/90-iscsi.rules
```

```
ACTION=="add",KERNELS=="sd[a-z]*",SUBSYSTEMS=="scsi",SYMLINK+="
iscsi%n"
```

5.重新加载 iscsi

```
[root@vh03 ~]# ls -l /dev/iscsi*
lrwxrwxrwx. 1 root root 3 6月  14 16:12 /dev/iscsi -> sda
lrwxrwxrwx. 1 root root 4 6月  14 16:12 /dev/iscsi1 -> sda1
```

文件系统类型：

本地文件系统,硬盘：EXT4,SWAP,XFS,NTFS,FAT

伪文件系统,内存空间：/proc,/sys

网络文件系统,网络存储：NFS

NFS：

NFS 共享协议:Linux 最基本的文件共享机制,1980 年由 SUN 公司开发,依赖于 RPC(远程过程调用)映射机制

优势：存取位于远程磁盘中的文档数据,就好像访问本地文件一样

/etc/exports 配置解析：

共享目录 允许访问的客户机地址(参数,参数...)

客户机地址：

单个 IP：192.168.4.3

网段：192.168.4.*或 192.168.4.0/24

所有主机：*

单个域：*.example.com

单个主机名：server.example.com

参数：

rw：读写权限,同时需要配置共享目录的其他人读写权限

ro：只读权限

sync：同步写入

async：异步写入

no_root_squash：保留客户端的 root 权限

all_squash：客户端权限都姜维 nfsnobody

Multipath 多路径：

描述：

服务器到某存储设备有多条路径时,每条路径都会识别为一个单独的设备

多路径允许将服务器节点和储存阵列间的多个 I/O 路径配置为一个单一设备

这些路径包含独立电缆,交换机和控制器的实体 SAN 连接

多路径集合了多个 I/O 路径,并生成由这些集合路径组成的新设备

主要功能：冗余,负载均衡

配置多路径：

1.在 192.168.4.0 网段和 192.168.2.0 网段分别加载一次 iscsi：

```
[root@vh03 ~]# iscsiadm --mode discoverydb --type sendtargets --portal
192.168.4.1 --discover
[root@vh03 ~]# iscsiadm --mode discoverydb --type sendtargets --portal
192.168.2.1 --discover
```

```
[root@vh03 ~]# iscsiadm --mode node --targetname
iqn.2018-06.cn.tedu.nsd1802 --portal 192.168.4.1:3260 --login
[root@vh03 ~]# iscsiadm --mode node --targetname
iqn.2018-06.cn.tedu.nsd1802 --portal 192.168.2.1:3260 --login
```

2.安装多路径软件包：

```
[root@vh03 ~]# yum -y install device-mapper-multipath
```

3.创建配置文件：

```
[root@vh03 ~]# mpathconf --user_friendly_names n
```

4.查看 WWID：

每个多路径设备都有一个 WWID(全球识别符),全球唯一,无法更改

默认情况下,WWID 就是多路径设备的名称

由于 sda 和 sdb 实际上是同一个设备,所以 WWID 相同

```
[root@vh03 ~]# /lib/udev/scsi_id --whitelisted --device=/dev/sda
    36001405d3760b15edee46ccb96a059b9
[root@vh03 ~]# /lib/udev/scsi_id --whitelisted --device=/dev/sdb
    36001405d3760b15edee46ccb96a059b9
```

5.修改配置文件：

```
[root@vh03 ~]# vim /etc/multipath.conf
#参考 60-74 行设置别名
    multipaths{
        multipath{
            wwid "36001405d3760b15edee46ccb96a059b9"
            alias mpath}}
```

6.启动服务：

```
[root@vh03 ~]# systemctl start multipathd.service
```

7.查看：

```
[root@vh03 ~]# multipath -ll
    #查看多路径
    |-+- policy='service-time 0' prio=1 status=active
    | `- 5:0:0:0 sda 8:0  active ready running
    `-+- policy='service-time 0' prio=1 status=enabled
      `- 6:0:0:0 sdb 8:16 active ready running
[root@vh03 ~]# multipath -rr
    #重新加载多路径信息
[root@vh03 ~]# ls /dev/mapper/
#查看多路径设备
    36001405d3760b15edee46ccb96a059b9
    36001405d3760b15edee46ccb96a059b9p1
```

网站服务器：

LINUX、UNIX:

php、python：apache、nginx、tengine

java：tomcat、jboss

WINDOWS：IIS

Web 的工作原理：

服务器：

存储 Web 上的内容信息，提供管理环境

响应浏览器的请求，执行服务器端程序

安全功能

浏览器：

提交请求

作为 HTML 解释器和内嵌脚本程序执行器

用图形化的方式显示 HTML 文档

Web 的相关技术：

服务器：

PHP (Hypertext Preprocessor)

JSP (Java Server Page)

ASP (Active Server Page)

Python Web

客户端：

HTML

CSS

JavaScript

HTML(HyperText Markup Language)：超文本标记语言

标记语法：用于描述功能的符号称为“标记”，标记在使用时必须使用尖括号括起来。如：<h1>

封闭类型标记：

双标记，必须成对出现

格式：<标记>内容</标记>

非封闭类型标记：

空标记或者单标记，

格式：<标记 /> 或 <标记>

元素：

每一对尖括号包围的部分

<body></body>包围的部分就叫做 body 元素

元素就像是小标签，用于标识网页文档的不同部分

元素可以包含文本内容和其他元素,也可以是空的

元素之间可以相互嵌套，形成更为复杂的语法

在嵌套元素时需要注意标记的嵌套顺序

属性：用来修饰元素

属性的声明必须位于开始标记里

一个元素的属性可能不止一个，多个属性之间用空格隔开

多个属性之间不区分先后顺序

每个属性都有值，用等号连接，属性的值包含在引号中

标准属性：通用属性

id
title
class
style

注释：
只在编辑文档情况下可见，在浏览器展示页面时并不会显示
格式：<!-- 注释的文本内容 -->
注意：注释不可以嵌套在其他注释中，注释不可以位于嵌在<>中

结构：
声明：
在文档的起始用 DOCTYPE 声明指定版本和风格
页面：<html>元素
<head>：页面的头部内容
为页面定义全局信息
紧跟在起始标签 <html> 之后
所有其他头元素的容器，如：title、meta、script、style、link
<title>：标题元素
<title></title> 用于为文档定义标题
标题元素的内容出现在浏览器顶部
没有属性
必须出现在 <head> 元素中
一个文档只能有一个标题元素
<meta>：元数据元素
<meta>用于定义网页的基本信息
非封闭类型标记
常用属性有:content、http-equiv
<body>：页面的主体内容
页面的主体内容
可以包含除了 html、head 外所有元素

特殊字符：
< <
> >
  空格
© ©

空格折叠：多个空格或制表符压缩成单个空格，即只显示一个空格， 不受此规则约束

文本样式：文本进行修饰
加粗
倾斜
<u>下划线</u>
<s>删除线</s>
^{上标}
_{下标}

块级元素：块级元素会独占一行,即元素前后都会自动换行

标题元素：<hn>,n=1, 2, 3, 4, 5, 6

让文字以醒目的方式显示，用于 body 中标题

段落元素：<p>

添加一段额外的垂直空白距离，作为段落间距

常用属性:align

换行元素：

手工换行

非封闭类型标记

块分区元素：

为元素分组，常用于页面布局

行内元素：不会换行,可以和其他行内元素位于同一行

行内分区元素:

为元素分组，常用于设置同一行文字内的不同格式

链接：

URL(Uniform Resource Locator)：统一资源定位器，用来标识网络中的任何资源

绝对路径：从最高级目录下开始的完整的路径，无论当前路径是什么，使用绝对路径总是能找到要链接的文件

完整路径：协议、主机名、目录路径、文件名

相对路径：相对于当前文件的位置。它包括目录名或指向一个可以从当前目录出发找到该文件的路径

图像：<img>

非封闭类型标记

必须属性：

src：指向图像的 URL

常用属性：

alt：图像无法显示时，显示的文字

width：图像的长

height：图像的高

链接元素：<a>

href：链接 URL

#：返回页面顶部的空链接

javascript:...：链接到 JavaScript

mailto:...@...：电子邮件链接

target：

_blank：新窗口打开

_self：本窗口打开

name：锚点名称

锚点是文档中某行的一个记号，用于链接到文档中的某个位置

如果文本/图像与锚点在同一页面，可以直接引用

如果文本/图像与锚点在不同页面，需要指明 URL

创建表格：

定义表格：使用成对的<table></table>标记

width：设置表格宽度

height：设置表格高度

align:设置表格对齐方式
left：左对齐
center：中间对齐
right：右对齐
border:设置表格边框宽度
cellpadding:设置内边距(单元格边框与内容之间的距离)
cellspacing:设置外边距(单元格之间的距离)
bgcolor:设置表格背景颜色
创建表行：使用成对的<tr></tr>标记
align：设置水平对齐方式(left|center|right)
valign：设置垂直对齐方式(top|middle|bottom)
创建单元格：使用成对的<td></td>标记
align：设置水平对齐方式(left|center|right)
valign：设置垂直对齐方式(top|middle|bottom)
width：设置宽度
height：设置高度
colspan：设置单元格跨列，水平方向延伸单元格，值为正整数
rowspan：设置单元格跨行，垂直方向延伸单元格，值为正整数

列表：

将具有相似特征或者具有先后顺序的几行文字进行对齐排列
有序列表：
用于列出页面上有特定次序的一些项目
type 值：项目前的序号
1：数字(默认)
a：小写字母
A：大写字母
i：小写罗马数字
I：大写罗马数字
start：起始编号
列表项：
具体的列表内容
无序列表：
用于列出页面上没有特定次序的一些项目
type 值：项目前的符号
disc：实心圆(默认)
circle：空心圆
square：实心矩形
列表项：
列表嵌套：可以创建多层列表，常用于创建文档大纲、导航菜单

表单：

用于显示、收集信息，并提交信息到服务器
基本部分：实现数据交互的可见的界面元素、提交后的表单处理
表单元素<form></form>：
action:表单被提交时发生的动作,通常包含服务方脚本的 URL(比如 JSP、PHP)

method：表单数据提交的方式，值为 get 或 post
enctype：表单数据进行编码的方式
name：表单名称

控件：

<input>：收集用户信息
type 指定类型
text：文本框
submit：提交表单
password：密码框
button：按钮
checkbox：多选框
checked：设置默认被选中
name：设置名称，用于分组，同组多选框的名称必须相同
radio：单选框
checked：设置默认被选中
name：设置名称，用于分组，同组单选框的名称必须相同
reset：重置
hidden：隐藏域
file：文件选择框
value：值
type="button","reset","submit"：定义按钮上的显示的文本
type="text","password","hidden"：定义输入字段的初始值
type="checkbox","radio","image"：定义与输入相关联的值
name：设定名称；在 JavaScript 中引用元素；在表单提交时引用表单数据。
disabled：禁用控件
maxlength：限制输入的字符数
readonly：设置控件只读
placeholder：占位符
<textarea></textarea>：多行文本输入框
name：名称
cols：列数
rows：行数
readonly：设置只读
<select></select>：下拉菜单
name：选项框名称
size：1 表示下拉菜单；n(n>1)表示 n 行的滚动列表
multiple：通过鼠标滑动进行多选，会屏蔽下拉菜单
option：菜单选项
value：选项的值
selected：预选中
<label></label>：将文本与控件联系在一起
for：指定控件的 id

CSS(Cascading Style Sheets)：层叠样式表，级联样式表，简称样式表

实现了将内容与表现分离，提高代码的可重用性和可维护性
HTML 是页面的内容，CSS 是页面的表现
使用方式：
内联方式：样式定义在单个的 HTML 元素中
使用 style 属性的值的方式表示
属性和属性值之间用:连接
多对属性之间用;隔开
内部样式表：样式定义在 HTML 页的头元素中
在文档的<head>元素内添加<style>元素
在<style>元素中添加样式规则，可以定义多个样式规则
规则有两个部分：
选择器：决定哪些元素使用这些规则
样式声明：一对大括号,包含一个或者多个属性/值对
外部样式表：将样式定义在一个外部的 CSS 文件中(.css 文件)，由 HTML 页面引用样式表文件
创建一个单独的样式表文件用来保存样式规则，文件后缀为.css，该文件中只能包含样式规则
在需要使用该样式表文件的<head>元素内添加<link>元素,链接需要的外部样式表文件
样式表特征：
继承性：大多数 CSS 的样式规则可以被继承
层叠性：可以定义多个样式；不冲突时,多个样式表中的样式可层叠为一个
优先级：样式定义冲突时，按照不同样式规则的优先级来应用样式
通用选择器：星号(*)，可以与任何元素匹配，常用于设置一些默认样式
元素选择器：html 文档的元素，如 h1,body
类选择器：标签.类名。
如果不带标签，则表示所有标签
如果标签后带有空格，则表示该标签下的子标签使用该类也遵循规则
所有能够附带class属性的元素将class属性的值设置为样式类名即可使用此样式
class 属性的值中可以包含一个词列表，各个词之间用空格分隔，每个词都是一个类选择器
id 选择器：标签#id
仅作用于指定 id 的属性的文本
不带标签表示所有标签
群组选择器：以逗号隔开的选择器列表，将一些相同的规则作用于多个元素
伪类选择器：选择器:伪类选择器
用于向某些选择器添加特殊的效果
链接伪类：
link：适用于尚未访问的链接
visited：适用于访问过的链接
动态伪类：用于呈现用户操作
hover：适用于鼠标悬停在 HTML 元素时
active：适用于 HTML 元素被激活时

focus：适用于 HTML 元素获取焦点时

尺寸单位：

%：百分比

in：英寸

cm：厘米

mm：毫米

pt：磅(1pt 等于 1/72 英寸)

px：像素(计算机屏幕上的一个点)

em：1em 等于当前的字体尺寸，2em 等于当前字体尺寸的两倍，继承父级元素的字体大小

rem：与 em 类似，但是相对于根元素设置字体尺寸的倍数

颜色单位：

rgb(x,x,x):RGB 值

rgb(x%,x%,x%):RGB 百分比值

#rrggbb:十六进制数

#rgb:简写的十六进制数

表示颜色的英文单词

尺寸属性：

宽度属性：

width

max-width

min-width

高度属性：

height

max-height

min-height

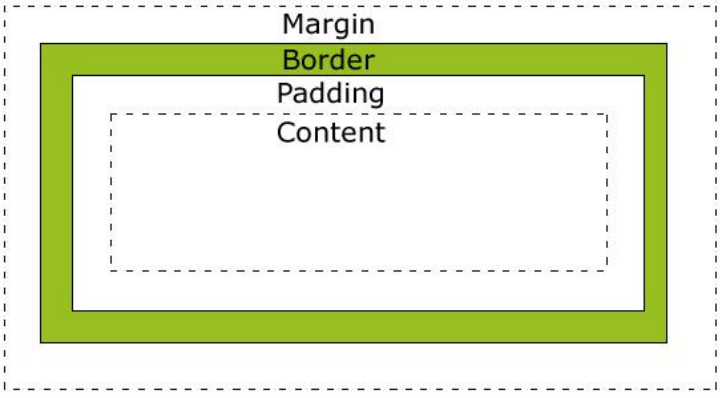
框模型(Box Model)：

Margin(外边距)：元素边框周围的空白区域，外边距是透明的。值可以为负

Border(边框)：围绕在内边距和内容外的边框。

Padding(内边距)：内容区域和边框之间的空间，内边距是透明的。值不能为负

Content(内容)：盒子的内容，显示文本和图像。



边框属性：

border：简写属性，用于把针对四个边的属性设置在一个声明。

border-style：用于设置元素所有边框的样式，或者单独地为各边设置边框样

式。

1 个值：四面边框

2 个值：上下边框，左右边框

3 个值：上边框，左右边框，下边框

4 个值：上边框开始顺时针：上，右，下，左

border-width：简写属性，用于为元素的所有边框设置宽度，或者单独地为各边边框设置宽度。

border-color：简写属性，设置元素的所有边框中可见部分的颜色，或为 4 个边分别设置颜色。

border-bottom：简写属性，用于把下边框的所有属性设置到一个声明中。

border-bottom-color：设置元素的下边框的颜色。

border-bottom-style：设置元素的下边框的样式。

border-bottom-width：设置元素的下边框的宽度。

border-left：简写属性，用于把左边框的所有属性设置到一个声明中。

border-left-color：设置元素的左边框的颜色。

border-left-style：设置元素的左边框的样式。

border-left-width：设置元素的左边框的宽度。

border-right：简写属性，用于把右边框的所有属性设置到一个声明中。

border-right-color：设置元素的右边框的颜色。

border-right-style：设置元素的右边框的样式。

border-right-width：设置元素的右边框的宽度。

border-top：简写属性，用于把上边框的所有属性设置到一个声明中。

border-top-color：设置元素的上边框的颜色。

border-top-style：设置元素的上边框的样式。

border-top-width：设置元素的上边框的宽度。

背景：

background-color：设置背景色，可取值为 transparent 透明

background-image：设置背景图片，默认值是 none，URL 指明图片地址

background-repeat：设置图片重复方式。不指定该设置时，值默认为 repeat

background-size：图片占用页面的大小。100%表示铺满页面

字体：

font：简写属性，按顺序设置 5 个属性

font-style font-variant font-weight font-size font-family

font-family：指定字体

font-size：字体大小

font-weight：字体加粗。bold 加粗

font-style：字体样式。italic 斜体

font-variant：字体变化。all-small-caps 转化为小型大写字母

文本格式：

color：文本颜色

text-align：文本排列

text-decoration：文字修饰

line-height：行高

Bootstrap：

简介、直观、强大的前端开发框架

官网：http://www.bootcss.com/

可下载压缩包，也可以直接引用网络上的样式库

由于需要 HTML 元素和 CSS 属性，所以需要设置 HTML5

Bootstrap 的所有 JavaScript 插件都依赖 jQuery，因此 jQuery 必须在 Bootstrap 之前引入。在 bower.json 文件中列出了 Bootstrap 所支持的 jQuery 版本。

移动设备优先：确保适当的绘制和触屏缩放，需要在<head>中添加元数据元素

<meta name="viewport" content="width=device-width,initial-scale=1">

目录说明：

css 目录：用于存放 Bootstrap 框架使用的样式文件

bootstrap.css 文件：Bootstrap 框架的样式文件，未经压缩

bootstrap.min.css 文件：Bootstrap 框架的样式压缩文件

bootstrap-theme.css 文件：Bootstrap 框架的主题文件

fonts 目录：用于存放 Bootstrap 框架使用的字体文件

js 目录：用于存放 Bootstrap 框架使用的核心 javascript 文件

bootstrap.js 文件：Bootstrap 框架的核心 javascript 文件，未经压缩

bootstrap.min.js 文件：Bootstrap 框架的核心 javascript 压缩文件

解决兼容性文件：

html5shiv.min.js：解决 IE8 浏览器支持 HTML5 的新元素

respond.min.js：解决 IE8 浏览器支持 CSSMediaQuery

全局设置：

body 元素颜色默认为 black

所有链接设置了默认颜色，当处于 hover 状态时添加下划线且变色

布局容器：

.container 类用于固定宽度并支持响应式布局的容器

.container-fluid 类用于占据全部视窗的容器

按钮：

6 种颜色：

btn-primary：首选项

btn-danger：危险

btn-success：成功

btn-warning：警告

btn-info：一般信息

btn-default：默认

4 种尺寸：

btn-lg：大

btn-default：默认

btn-sm：小

btn-xs：极小

文本格式：

对齐：

text-left：左对齐

text-center：中间对齐

text-right：右对齐
text-justify：两端对齐
text-nowrap：不换行

字母大小写转换：

text-lowercase：转化为小写
text-uppercase：转化为大写
text-capitalize：首字母大写

颜色：text-success、text-warning、text-danger、text-info、text-primary

列表组：

：.list-group
：.list-group-item
代替<ul>元素：通过使用锚标签代替列表项，向列表组添加链接

表单：<form>

垂直表单：默认
内联表单：.form-inline。向左对齐的，标签是并排的，将表单控件放在一行显示
<label>设置.sr-only 将被隐藏
水平表单：.form-horizontal

表格：

table：基本格式
table-bordered：带边框
table-hover：悬停鼠标实现背景效果
<tr>标签行背景样式，<th>和<td>标签设置单元格背景：success，warning，info，danger

字体图标：glyphicon

不能和其他组件混合使用
只对内容为空的元素起作用

标签页：选项卡功能

网站访问的参数：

PV(Page View)访问量，页面浏览量或点击量，在一定统计周期内用户每打开或刷新一个页面就记录 1 次

UV(Unique Visitor)独立访客，统计 1 天内(00:00-24:00)访问某站点的用户数，以 cookie 为依据

IP(Internet Protocol)独立 IP 数，是指 1 天内多少个不同的 IP 浏览了页面，多个局域网 IP 在同一个路由器内上网，只计 1 个

HIT：点击量，页面中的某个元素被用户点击打开的次数

httpd 和 nginx 的对比：

Apache 2.X 支持插入式并行处理模块，称为多路处理模块(MPM)：/etc/httpd/conf.modules.d/00-mpm.conf

Prefork MPM：进程模式，每个请求相互独立
Worker MPM：进程线程混合模式，能处理比进程模式更大的访问量
Event MPM：事件触发模式，负载能力最高，但是 httpd 不能很好支持

nginx：

select：类似进程模式

poll：类似进程线程混合模式

epoll：类似事件触发模式，能很好的支持

HTTP 事务模型：每个请求仅能对应一个响应

常见模型：

HTTP close：

客户端向服务端发送请求，服务端响应后即断开连接. 多次请求延迟较大

keep-alive：

依次连接可以传输多个请求，客户端需要知道传输内容的长度，以避免无限期的等待

pipelining：

keep-alive 的基础上，不用等待前面的请求得到响应，即发送后续请求，适用于大量图片的页面

搭建基本的 Web（http：超文本传输协议）

1.安装服务端软件 httpd （作者：Apache:开源软件基金会）

[root@server0 ~]# yum install -y httpd

2.启动 httpd 服务，设置为开机自起的服务

[root@server0 ~]# systemctl restart httpd

[root@server0 ~]# systemctl enable httpd

3.书写自己的主页

默认网页文件路径：/var/www/html

默认网页文件名字 : index.html

默认书写网页内容的语言： html

[root@server0 ~]# vim /var/www/html/index.html

<marquee><font color=red><h1> NSD1801 #只需要测试功能即可

滚动 字体颜色 标题字体 内容

web 服务器：通过 web 软件将网页目录共享出去

网站基于 B/S 设计（浏览器/服务器）

在服务器上默认共享目录：/var/www/html

httpd 的配置文件：

/etc/httpd/conf/httpd.conf 主配置文件，基本不修改

/etc/httpd/conf.d/ 配置文件目录

查看主配置文件：

[root@server0 ~]# cat /etc/httpd/conf/httpd.conf

42:Listen 80 #服务器监听的端口

95:#ServerName www.example.com:80 #服务器的域名是什么，默认注释

119:DocumentRoot "/var/www/html" #网站的根路径在哪里，也就是页面存放的位置

144:Options Indexes FollowSymLinks #在没有 index.html 的情况下查看所有目录下的文件

164:DirectoryIndex index.html #默认的首页叫什么名字

353:IncludeOptional conf.d/*.conf #加载 conf.d 目录下的所有配置文件

虚拟主机：一台服务器，安装一个 httpd，做多个网站，防止资源浪费
虚拟主机类型：
基于 IP 地址（要求服务器有多个 IP）
基于域名（要求买多个域名）
基于端口

默认测试域名：server0.example.com，www0.example.com，webapp0.example.com
[root@server0 ~]# vim /etc/httpd/conf.d/server.conf #创建新的配置文件
<VirtualHost *:80>
ServerName server0.example.com
DocumentRoot /var/www/html
</VirtualHost>
[root@server0 ~]# vim /etc/httpd/conf.d/www.conf
<VirtualHost *:80>
ServerName www0.example.com
DocumentRoot /var/www/dachui
</VirtualHost>
[root@server0 ~]# mkdir /var/www/dachui
[root@server0 ~]# vim /var/www/dachui/index.html
[root@server0 ~]# cd /var/www/html/
[root@server0 ~]# vim index.html
hello world
[root@server0 ~]# vim test.html
nb is nb
[root@server0 ~]# mkdir haha
[root@server0 ~]# echo "xixi" > haha/index.html
[root@server0 ~]# chcon -R -t httpd_sys_content_t /var/www/html/nb.html #设置 SELinux 标签
[root@server0 ~]# systemctl restart httpd

在客户端上操作:
[root@server0 ~]#curl http://server0.example.com #打开的是之前实验创建的第 1 个页面
[root@server0 ~]#curl http://www0.example.com #打开的是刚刚创建的第 2 个页面
[root@server0 ~]#curl http://server0.example.com/test.html
[root@server0 ~]#curl http://server0.example.com/haha/

网站的 ACL 访问控制
语法格式：
<Directory 目录的绝对路径>
Require all denied|granted 或 Require ip IP 或网段地址 .. ..
</Directory>

[root@server0 ~]#vim /etc/httpd/conf.d/server.conf　　#在原配置基础上追加写入

<Directory "/var/www/html/jpg">

Require ip 172.25.254.250 172.25.0.0/24　　#仅允许 172.25.0.*和 172.25.254.250 访问 jpg 目录

</Directory>

[root@server0 ~]#systemctl restart httpd

静态页面：html、mp3、flv、jpg、gif 等都是静态

动态页面：httpd 这个软件本身不能解释，需要被服务器的解释器解释的页面，如：java,php,python,shell

动态页面：

[root@server0 ~]#yum -y install mod_wsgi　　#mod_wsgi 支持 httpd 与 python 交互，当用户要访问一个 python 脚本时，httpd 可以把页面脚本给 python 执行一遍，然后把执行的结果给用户

#客户端请求<----浏览器

---->httpd<----mod_wsgi----->python

[root@server0 ~]# mkdir /var/www/webapp/

[root@server0 ~]# cd /var/www/webapp/

[root@server0 webapp]# wget http://classroom/pub/materials/webinfo.wsgi

[root@server0 webapp]# vim /etc/httpd/conf.d/webapp.conf

Listen 8909　　#开启一个端口

<VirtualHost *:8909> //使用前面开启的端口

ServerName　webapp0.example.com

DocumentRoot /var/www/webapp

WSGIScriptAlias /　/var/www/webapp/webinfo.wsgi #当访问网站的根时，等同于访问/var/www/webapp/webinfo.wsgi 脚本文件

</VirtualHost>

[root@server0 ~]# systemctl restart httpd　　#重启失败，根据提示执行 journalctl -xn，里面的 SELinux 的命令提示

[root@server0 ~]# semanage port -a -t http_port_t -p tcp 8909　#-a 是添加一台规则，-t 是指定 http 能使用什么端口，-p 允许 httpd 使用 tcp 的 8090 端口

HTTPS:网站加密

HTTP 是明文协议，网络中传输的任何数据都是明文,包括用户和密码，如果有人抓包，所有数据都可以获得

加密算法：

对称算法(AES,DES）：加密和解密是一把钥匙，适合单机加密

非对称算法(RSA,DSA)：加密用公钥，解密用私钥

信息摘要(md5,sha512,sha256)：数据完整性校验(检查数据是否被人修改过)、数据

秒传

服务器操作：

[root@server0 ~]# yum -y install mod_ssl　　#安装该软件包让 httpd 支持加密网站

这三个密钥原则上放哪都可以，但是有默认存放位置：

公钥(证书)：*.crt ： /etc/pki/tls/certs/

根证书(上面的私钥和公钥是谁提供的)：*.crt ： /etc/pki/tls/certs/

私钥：*.key ： /etc/pki/tls/private/

[root@server0 ~]# vim /etc/httpd/conf.d/ssl.conf

<VirtualHost *:443>　　#默认加密端口是 443

59: ServerName server0.example.com

60: DocumentRoot /var/www/html/

100:SSLCertificateFile /etc/pki/tls/certs/server0.crt　　#使用哪个密钥加密数据（公钥）

107:SSLCertificateKeyFile /etc/pki/tls/private/server0.key #使用哪个密钥解密数据（私钥）

122:SSLCACertificateFile /etc/pki/tls/certs/example-ca.crt #使用哪个跟证书

</VirtualHost>

[root@server0 ~]#systemctl restart httpd

客户端验证：

[root@desktop0 ~]# firefox http://server0.example.com　　#非加密，因为用的是 http

[root@desktop0 ~]# firefox https://server0.example.com　　#加密访问，提示不安全,第一次访问需要点击“高级”，“添加例外”

HTTP 请求头部：包含很多有关客户端环境和请求正文的有用信息

方法：get

URI：前面是网址，?后面接参数

版本：HTTP1.1

HTTP 响应头部：

版本：HTTP1.1

状态码：200

HTTP 支持 PHP：

安装 PHP 软件包：[root@web ~]# yum -y install php

安装完成后，会生成以下几个文件用于 HTTP 启动 PHP:

/etc/httpd/conf.d/php.conf

/etc/httpd/conf.modules.d/10-php.conf

/usr/lib64/httpd/modules/libphp5.so

重启 HTTP 服务即可：[root@web ~]# systemctl start httpd.service

HTTP 子进程重启：[root@fzr ~]# apachectl graceful

正向代理：缓存服务器

反向代理：代理服务器接受连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给请求连接的客户端

nginx：轻量级的 http 服务器，高性能的 http 和反向代理服务器，也是 imap、pop3、smtp

代理服务器

参数：启动服务不加参数

-s stop 关闭服务

-s reload 重新加载配置文件

-V 查看软件信息

nginx 安装：

```
[root@client ~]# cat nginx.sh
    #!/bin/bash
    n=$(yum repolist | awk '/repolist/{print $2}' |sed 's/,//g')
    [ $n -le 0 ] && echo 'yum 不可用' && exit
    yum -y install gcc openssl-devel pcre-devel >/dev/null
    useradd -s /sbin/nologin nginx
    tar -xf ~/nginx-1.10.3.tar.gz
    cd ~/nginx-1.10.3/
    ./configure --user=nginx --group=nginx --with-http_ssl_module >/dev/null
    make >/dev/null
    make install >/dev/null
    /usr/local/nginx/sbin/nginx
    /usr/local/nginx/sbin/nginx -V
[root@proxy ~]# ln -s /usr/local/nginx/sbin/nginx  /sbin/
[root@proxy ~]# netstat  -anptu  |  grep nginx
```

nginx 升级：

```
[root@proxy ~]# tar -xf nginx-1.12.2.tar.gz
[root@proxy ~]# cd nginx-1.12.2/
[root@proxy nginx-1.12.2]# ./configure --user=nginx --group=nginx
--with-http_ssl_module >/dev/null
[root@proxy nginx-1.12.2]# make >/dev/null
[root@proxy nginx-1.12.2]# mv /usr/local/nginx/sbin/nginx
/usr/local/nginx/sbin/nginx.old
[root@proxy nginx-1.12.2]# cp objs/nginx  /usr/local/nginx/sbin/
[root@proxy nginx-1.12.2]# make upgrade
```

nginx 加密：

```
    [root@proxy conf]# vim nginx.conf
        server{
        ...
        auth_basic "hello";
        auth_basic_user_file "/usr/local/nginx/pass";
        ...
        }
    [root@proxy nginx]# yum -y install  httpd-tools
    [root@proxy nginx]# htpasswd -c /usr/local/nginx/pass nb    #-c 为创建文件
```

```
[root@proxy nginx]# cat /usr/local/nginx/pass
[root@proxy nginx]# nginx -s reload                          #重新加载配置文件生效
```

Nginx 虚拟主机：

修改 Nginx 配置文件，添加 server 容器实现虚拟主机功能；对于需要进行用户认证的虚拟主机添加 auth 认证语句。

虚拟主机一般可用分为：基于域名、基于 IP 和基于端口的虚拟主机

```
server {
    listen       8000;                                   #修改端口
    listen       somename:8080;                          #修改端口及 IP
    server_name  somename  alias  another.alias;         #修改域名，定义别名
    charset utf-8;                                       #设置支持中文
    location / {
        root   html;
        index  index.html index.htm;
    }
}
```

Nginx 加密：

```
[root@client ~]# tcpdump                                 #抓包软件
[root@proxy conf]# openssl genrsa > cert.key             #生成私钥
[root@proxy conf]# openssl req -new -x509 -key cert.key > cert.pem  #生成公钥（证书）
[root@proxy conf]# vim nginx.conf                        #修改配置文件
    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;
        }
    }
```

Nginx 安全：默认会安装一部分模块。最小化安装永远是对的方案

开启默认不安装的模块：--with
禁用默认安装的模块：--without
1.查看所有模块：[root@proxy nginx-1.12.2]# ./configure --help
2.选择适合的模块：
pcre：开启正则表达式支持
http_autoindex_module：自动索引模块
ssi_module：SSI 脚本
http_ssl_module：ssl 支持 http 模块
http_v2_module：http_v2 协议
http_realip_module：获取客户端真实 ip
http_stub_status_module：查看服务器工作状态
http_charset_module：编码模块，例如 utf-8
http_gzip_module：压缩模块
http_auth_basic_module：用户认证
http_rewrite_module：地址重写模块
http_proxy_module：代理服务器模块
mail_pop3_module：邮件服务
mail_imap_module：邮件服务
mail_smtp_module：邮件服务
http_upstream_ip_hash_module：哈希轮询
[root@proxy nginx-1.12.2]# ./configure --without-http_autoindex_module #禁止访问时文件索引列表
[root@proxy nginx-1.12.2]# make
[root@proxy nginx-1.12.2]# make install
3.隐藏版本号：也可以在下一步中直接删除 CRLF 参数
[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf
http {
server_tokens off;
...}
4.隐藏软件信息：需要重新编译安装
[root@proxy nginx-1.12.2]# vim src/http/ngx_http_header_filter_module.c +49
static u_char ngx_http_server_string[] = "Server: Jacob" CRLF;
#简洁信息
static u_char ngx_http_server_full_string[] = "Server: Jacob" CRLF;
#完整信息
static u_char ngx_http_server_build_string[] = "Server: Jacob" CRLF;#构造信息
5.重写 40x.html 和 50x.html，屏蔽错误提示中的 nginx 信息
6.限制并发量：
DDOS 攻击者会发送大量的并发连接，占用服务器资源，导致正常用户处于等待或无法访问服务器的状态。
Nginx 默认安装模块中提供了一个 ngx_http_limit_req_module 模块，可以有效降低 DDOS 攻击的风险。

语法格式如下：limit_req_zone 关键字 zone=名字:容量 rate=接收请求速度。

$binary_remote_addr：系统内置变量，表示远程连接地址。

```
[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf
    http {
    ...
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;
            server {
            ...
            limit_req zone=one burst=5;
            ...
    }
```

将客户端 IP 信息存储在名称为 one 的内存中，内存空间 10M，能存储 8 万个主机连接状态，每秒仅接受 1 个请求，多余的放入漏斗，漏斗中的请求超过 5 个后则拒绝接下来的所有请求

7.拒绝非法的请求：

HTTP 协议中定义了很多方法，可以让用户连接服务器，获得需要的资源。但实际应用中一般仅用到 GET 和 POST：

GET：请求指定的页面信息，并返回实体主体

HEAD：类似于 GET，但是只返回报头

POST：提交数据并进行处理

DELETE：请求服务器删除指定页面

PUT：向服务器特定位置上传资料

```
[root@proxy ~]# curl -i -X GET  http://192.168.4.5      #-i 显示头部信息，-X 指定请求方法
[root@proxy ~]# curl -i -X HEAD http://192.168.4.5
[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf
    server {
    ...
    if ($request_method !~ ^(GET|POST)$ ) {return 444;} #当访问类型不是 GET 和 POST 时，返回 444 错误
    }
```

8.防止缓存溢出：

当客户端连接服务器时，服务器会启用各种缓存，用来存放连接的状态信息。

如果攻击者发送大量的连接请求，而服务器不对缓存做限制的话，内存数据就有可能溢出。

```
[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf
    http {
    client_body_buffer_size 8K;          #主体缓存空间
    client_max_body_size 16k;            #主体最大缓存
    client_header_buffer_size 1k;        #默认请求包头信息的缓存
    large_client_header_buffers 4 4k;    #大请求包头信息的缓存个数，每个缓存的容量
    ...}
```

LNMP 主流企业网站平台，由 Linux+Nginx+Mysql/Mariadb+PHP/Python 构成

安装：

```
[root@proxy ~]# yum -y install mariadb mariadb-server mariadb-devel php
php-mysql
[root@proxy lnmp_soft]# rpm -ivh php-fpm-5.4.16-42.el7.x86_64.rpm
```

也可以进入存有 rpm 的目录后直接用 yum 安装：

```
[root@proxy lnmp_soft]# yum -y install php-fpm-5.4.16-42.el7.x86_64.rpm
```

端口：

```
nginx：80               netstat -utnlp | grep :80
php：9000               netstat -utnlp | grep :9000
mariadb：3306           netstat -utnlp | grep :3306
```

启动服务：

```
[root@proxy ~]# systemctl restart php-fpm
[root@proxy ~]# systemctl restart mariadb
```

修改配置文件：重启服务生效

```
[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf
    location ~  \.php$ {                        #location 匹配用户的地
址栏，支持正则。未定义的地址匹配根的规则
        root           html;
        fastcgi_pass   127.0.0.1:9000;          #转发目标
        fastcgi_index  index.php;
        include        fastcgi_params;          #修改为 fastcgi.conf
    }
[root@proxy ~]# vim /etc/php-fpm.d/www.conf #本次实验不修改
    [www]
    listen = 127.0.0.1:9000                 #PHP 端口号
    pm.max_children = 50                    #最大进程数
    pm.start_servers = 5                    #最小进程数
    pm.min_spare_servers = 5                #最少需要空闲的进程数
    pm.max_spare_servers = 35               #最大允许几个空闲进程
```

注意：如果安装 php，则需要修改/usr/local/php/etc/php-fpm.d/www.conf，将 listen 的值/var/run/php-fpm.sock 修改为 127.0.0.1:9000

排错：

```
[root@proxy nginx]# vim /var/log/php-fpm/error.log          #PHP 错误
[root@proxy nginx]# vim /usr/local/nginx/logs/error.log     #nginx 错误
```

测试 php:

```
[root@proxy ~]# vim /usr/local/nginx/html/test1.php
    <?php
    $i="This is a test Page";
    echo $i;
    ?>
```

测试 PHP 与 MySQL 互通:

```
[root@proxy ~]# vim lnmp_soft/php_scripts/mysql.php
    <?php
```

```
$mysqli = new mysqli('localhost','root','','mysql');
if (mysqli_connect_errno()){
    die('Unable to connect!'). mysqli_connect_error();
}
$sql = "select * from user";
$result = $mysqli->query($sql);
while($row = $result->fetch_array()){
    printf("Host:%s",$row[0]);
    printf("</br>");
    printf("Name:%s",$row[1]);
    printf("</br>");
}
?>
```

nginx 地址重写：

格式：rewrite 旧地址 新地址 [选项]

选项：

last 匹配即停止读其他 rewrite

break 匹配后停止读下面的所有配置文件

redirect 临时重定向，给搜索引擎、爬虫看的

permament 永久重定向，给搜索引擎、爬虫看的

```
[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf
```

网页重定向：

```
location / {
...
rewrite /a.html /b.html;
}
```

跳转地址栏：

```
location / {
...
rewrite /a.html /b.html redirect;
}
```

网址重定向：

```
server {
...
rewrite ^/  http://www.baidu.com/;
}
```

跳转到相同的子页面：常用功能

```
server {
...
rewrite ^/(.*)  http://www.baidu.com/$1;
}
```

根据不同的浏览器跳转不同的网页：主要针对手机端和电脑端

需要准备两套不同的网页，放在两个不同的目录下

```
        server {
        ...
        if ($http_user_agent ~* firefox) {rewrite /(.*) /firefox/$1;}
        }
```

nginx 反向代理：采用轮询的方式调用后端 Web 服务器实现负载均衡、热备份

weight 设置后台服务器的权重，默认权重为 1；

max_fails 设置后台服务器的最大失败次数；

fail_timeout 设置后台服务器的失败后暂停服务的时间，单位为秒

down 设置后台服务器关闭

backup 设置备份服务器

```
[root@proxy ~]# vim /usr/local/nginx/conf/nginx.conf
    http {
    ...
    upstream webs {
        ip_hash;                        #配置调度算法，设置相同客户端访问相
    同 Web 服务器
        server 192.168.2.100:80;        #第一台后台服务器
        server 192.168.2.200:80 weight=2 max_fails=2 fail_timeout=10 down;
            }
    ...
    server {
            listen   192.168.4.5:80;
            server_name  www.e.com;
                location / {
                proxy_pass http://webs;
                }
            }
    ...
    }
```

Nginx 的 TCP/UDP 调度器：安装--with-stream（4 层代理模块），开启 ngx_stream_core_module 模块

proxy_timeout 代理超时时间

proxy_connect_timeout 获取后台服务超时时间

安装模块：升级或者重装都可以

```
[root@proxy nginx-1.12.2]# ./configure --user=nginx --group=nginx
--with-http_ssl_module --with-stream
[root@proxy nginx-1.12.2]# make
[root@proxy nginx-1.12.2]# make install
[root@proxy nginx]# vim conf/nginx.conf
    events {...}
    stream {
```

```
        upstream backend {
        server 192.168.2.100:22;          #后端服务器 IP 和端口号
        server 192.168.2.200:22;
        }
        server {
            listen 12345;                 #监听端口
            proxy_connect_timeout 1s;     #获取后台服务超时时间
            proxy_timeout 3s;             #代理超时时间，建议不写，用默认时间
            proxy_pass backend;
                 }
    }
    http{...}
```

PS:如果服务是多端口的，则需要建立多次端口转发

优化 nginx 的并发：

1.修改 nginx 配置文件

```
[root@proxy nginx]# vim conf/nginx.conf
    worker_processes  1;                  #线程数，最大匹配 CPU 核心数，通常设置为 auto
    worker_rlimit_nofile 16384;           #设置文件描述符，默认 1024
    events {
        use epoll;                        #使用 epoll 模式
        worker_connections  1024;}        #每线程并发量，最大设置 65535，匹配最大端口号
```

2.修改 linux 内核参数

```
[root@proxy ~]# ulimit -a                 #查看所有属性值
[root@proxy ~]# ulimit -Hn 100000         #设置硬限制,非 root 用户允许修改的软限制上限
[root@proxy ~]# ulimit -Sn 100000         #设置软限制,非 root 用户可以自行修改
```

3.优化 nginx 数据包头缓存

```
[root@proxy nginx]# vim conf/nginx.conf
    http{
    client_header_buffer_size  1k;            #默认请求包头信息的缓存
    large_client_header_buffers  4 4k;        #大请求包头信息的缓存个数，每个缓存的容量
    ...
    }
```

4.定义静态页面的缓存时间

```
[root@proxy nginx]# vim conf/nginx.conf
    server {
    ...
    location ~* \.(jpg|jpeg|gif|png|ico|xml)$ {      #定义需要缓存的文件格式
    expires        30d;                               #定义缓存天数
    }
    ...}
```

5.对网页进行压缩

```
[root@proxy nginx]# vim conf/nginx.conf
    http{
    ...
    gzip on;                          #开启压缩
    gzip_min_length 1000;             #小文件不压缩，单位 kb
    gzip_comp_level 4;                #压缩比率，数字越大，压缩效果越好，处理器压力越大
    gzip_types text/plain text/css; #需要压缩的文件类型，扩展名对应类型可通过 conf/mime.types 查询
    ...}
```

6.自定义报错页面

```
[root@proxy nginx]# vim conf/nginx.conf
    http{
    ...
     error_page  403  404  414      /404.html;  #将模板注释去除即可
    ...}
```

7.nginx 动静分离

将静态页面与动态页面、图片资源、声音视频资源放在不同的服务器上

常见的 http 状态码：

200 一切正常
301 永久重定向
302 临时重定向
304 缓存重定向
400 请求语法错误
401 用户名或密码错误
403 禁止访问，IP 被拒绝
404 文件不存在
414 请求 URL 头部过长
500 服务器内部错误
502 网关或代理服务器返回了非法的路由

查看服务器状态：--with-http_stub_status_module 模块

1.安装模块

2.

```
[root@proxy nginx]# vim conf/nginx.conf
    location /status{
    stub_status on;              #开启模块
    allow 192.168.4.100;         #允许 IP
    deny all;}
```

3.

```
[root@client ~]# curl 192.168.4.5/status
    Active connections: 1
    server accepts handled requests
     1 1 1
```

Reading: 0 Writing: 1 Waiting: 0

参数含义：

Active connections：当前活动的连接数量。

Accepts：已经接受客户端的连接总数量。

Handled：已经处理客户端的连接总数量（一般与 accepts 一致，除非服务器限制了连接数量）。

Requests：客户端发送的请求数量。

Reading：当前服务器正在读取客户端请求头的数量。

Writing：当前服务器正在写响应信息的数量。

Waiting：当前多少客户端在等待服务器的响应。

PHP 的 Session 信息

Session：存储在服务器端，保存用户名、密码等信息。

Cookies：由服务端下发给客户端，内容包括：SessionID、帐号名、过期时间、路径、域。

默认存放路径： /var/lib/php/session/

修改配置文件，将所有后端服务器的 Session 统一放在数据库服务器上

[root@web1 ~]# vim /etc/php-fpm.d/www.conf

php_value[session.save_handler] = memcache

php_value[session.save_path] = "tcp://192.168.2.5:11211"

JAVA：一种跨平台的、面向对象的程序设计语言，具有通用性、高效性、平台移植性和安全性

java 体系：SE（标准版）、EE（企业版）、ME（移动版）...

JDK(Java Development Kit)：Sun 针对 Java 开发者推出的 Java 语言的软件开发工具包，包括运行环境、工具、类库

JRE(Java Runtime Environment)：JDK 的子集，Java 运行环境包，包括虚拟机、类库。没有开发工具

常见的 JAVA 扩展 WEB 服务器功能的组件容器：

websphere：IBM

weblogic：Oracle

tomcat：Apache

Jboss：Redhat

安装：[root@web1 ~]# yum -y install java-1.8.0-openjdk java-1.8.0-openjdk-headless

查看版本：[root@web1 ~]# java -version

tomcat：

1.免安装：

[root@web1 lnmp_soft]# tar -xf apache-tomcat-8.0.30.tar.gz

[root@web1 lnmp_soft]# mv apache-tomcat-8.0.30 /usr/local/tomcat

目录概述：

bin/ 主程序目录

lib/ 库文件目录

logs/ 日志目录

temp/　　临时目录
work/　　自动编译目录 jsp 代码转换 servlet
conf/　　配置文件目录
webapps/　默认页面目录

2.启动服务：[root@web1 ~]# /usr/local/tomcat/bin/startup.sh

3.验证端口：[root@web1 tomcat]# ss -ntulp | grep java

需要 3 个端口，8080，8005，8009，如果 8005 端口启动非常慢，可使用 urandom 替换 random

[root@web1 ~]# mv /dev/random /dev/random.bak
[root@web1 ~]# ln -s /dev/urandom /dev/random

4.查看默认配置：

```
[root@web1 tomcat]# cat conf/server.xml
    <Server ...>
        ...
        <Service ...>
        ...
            <Connector port="8080" ... />                    #负责建立
        http 连接
            <Connector port="8443" ... />                    #负责建立安
        全 http 连接
            <Connector port="8009" ... />                    #负责和其他
        http 服务器连接
            <Engine ... defaultHost="localhost">
            #defaultHost 决定直接访问 IP 时的默认网页
                <Host name="localhost"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">
                </Host>
                ...
            </Engine>
        </Service>
    </Server>
```

5.修改配置文件：

<Host>标签：

name　　域名
appBase 主目录
path　　定义访问路径，默认为/
docBase 文件夹名，默认为 ROOT
reloadable　出错后是否重加载

```
[root@web1 tomcat]# vim conf/server.xml
    <Host name="www.a.com"  appBase="aa" unpackWARs="true"
autoDeploy="true">
        <Context path="" docBase="base" reloadable="true"/>
        <Context path="/test" docBase="/var/www/html/" />
        <Context path="/b" docBase="../bb/base" />          #类似重定向
```

```
        </Host>
```

6.创建日志：

```
    directory   日志目录
    prefix      日志文件名
    suffix      日志扩展名
    [root@web1 tomcat]# vim conf/server.xml
        <Host ...>
        ...
        <Valve className="org.apache.catalina.valves.AccessLogValve"
    directory="logs"
        prefix="localhost_access_log" suffix=".txt" pattern="%h %l %u %t
    &quot;%r&quot; %s %b" />
        </Host>
    [root@web1 tomcat]# ls logs/
```

7.重启服务：[root@web1 ~]# /usr/local/tomcat/bin/shutdown.sh

[root@web1 ~]# /usr/local/tomcat/bin/startup.sh

8.在 tomcat 上发布 java 程序：通过 Maven 打包成 war 包，然后拷贝到 tomcat 的页面目录下，重启 tomcat 即可

配置加密的 tomcat：

1.生成密钥：[root@web1 tomcat]# keytool -genkeypair -alias tomcat -keyalg RSA -keystore /usr/local/tomcat/keystore

keytool	JDP 的密钥工具
-genkeypair	生成密钥对
-alias	定义别名，可选
-keyalg	定义密钥算法
-keystore	定义存储路径

2.修改配置文件，默认注释，需要解除注释。对 service 下的所有网页都生效：

```
    [root@web1 tomcat]# vim conf/server.xml
        <Connector port="8443"
protocol="org.apache.coyote.http11.Http11NioProtocol"
                   maxThreads="150" SSLEnabled="true" scheme="https"
secure="true"
                   clientAuth="false" sslProtocol="TLS"
                   keystoreFile="/usr/local/tomcat/keystore"
                keystorePass="123456"/>
```

port	端口
maxThreads	最大线程数
SSLEnabled	是否开启 SSL
scheme	网页类型
secure	是否开启加密
keystoreFile	密码库文件路径
keystorePass	密钥库的密钥口令

3.重启服务

Tomcat 安全：

1.隐藏版本信息：

```
[root@web1 ~]# yum -y install java-1.8.0-openjdk-devel
[root@web1 ~]# cd /usr/local/tomcat/bin/
[root@web1 lib]# jar -xf catalina.jar
[root@web1 lib]# vim org/apache/catalina/util/ServerInfo.properties +16
    server.info=hello world
    server.number=2018
[root@web1 tomcat]# vim conf/server.xml +69
    <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
redirectPort="8443" server="hello"/>
[root@web1 ~]# curl http://192.168.2.100:8080/xx    #验证网页
[root@web1 ~]# curl -I http://192.168.2.100:8080/xx #验证头部
```

2.使用普通用户启动

```
[root@web1 ~]# useradd tomcat -s /sbin/nologin
[root@web1 ~]# chown -R tomcat:tomcat /usr/local/tomcat/
[root@web1 ~]# /usr/local/tomcat/bin/shutdown.sh
[root@web1 ~]# su - tomcat -c "/usr/local/tomcat/bin/startup.sh"
[root@web1 ~]# vim /etc/rc.local                          #设置开机自启动
    su - tomcat -c "/usr/local/tomcat/bin/startup.sh"
```

3.删除默认的测试页面：[root@web1 tomcat]# rm -rf webapps/*

Tomcat 优化：

配置配置文件连接数：

设置 server.xml 的<Connector>标签：

minProcessors：最小空闲连接线程数，用于提高系统处理性能，默认值为 10。

maxProcessors：最大连接线程数，并发处理的最大请求数，默认值为 75。

acceptCount：指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理。默认值为 100。

enableLookups：是否反查域名，取值为 true 或 false。为了提高处理能力，应设置为 false。

connectionTimeout：网络连接超时，单位：毫秒。设置为 0 表示永不超时。通常可设置为 20000～30000 毫秒。

maxThreads：Tomcat 使用线程来处理接收的每个请求。这个值表示 Tomcat 可创建的最大的线程数。

maxConnections：最大并发连接数。

minSpareThreads：启动时创建的线程数。

maxSpareThreads：一旦创建的线程超过这个值，Tomcat 就会关闭不再需要的 socket 线程。

配置内存大小：默认最大内存为 128MB

修改 bin/catalina.bat 中的 set CATALINA_OPTS=-Xms64m -Xmx128m。Xms 指最小内存，Xmx 指最大内存

tomcat 内存控制：

修改 bin/catalina.sh 的 JAVA_OPTS 参数：

-Xms：初始堆内存大小

-Xmx：最大堆内存大，一般设置-Xms 与-Xmx 一样大小，根据应用类型和物理内存

大小来决定二者的大小

-Xmn(-XX:NewSize)：堆内存中年轻代的大小

-XX:PermSize：持久代内存的初始大小

-XX:MaxPermSize：持久代内存的最大值

Java 内存溢出(OOM)：

参考网址：https://www.jianshu.com/p/2fdee831ed03

Java 应用程序在启动时会指定所需要的内存大小，它被分割成两个不同的区域：Heap space(堆空间)和 Permgen space(持久代)。分别通过参数-Xmx 和-XX:MaxPermSize 设置。堆空间又被划分成两个不同的区域：新生代(Young)和老年代(Tenured)。新生代又被划分为 3 个区域：Eden、From Survivor、To Survivor。持久代主要存储的是每个类的信息，比如：类加载器引用、运行时常量池(所有常量、字段引用、方法引用、属性)、字段(Field)数据、方法(Method)数据、方法代码、方法字节码。JDK1.8 以上使用元空间(Metaspace)代替持久代，注意：部分持久代中的内容移动到了普通堆里面。

1.java.lang.OutOfMemoryError:Java heap space

原因：流量/数据量峰值：应用程序在设计之初均有用户量和数据量的限制，某一时刻，当用户数量或数据量突然达到一个峰值，并且这个峰值已经超过了设计之初预期的阈值，那么以前正常的功能将会停止，并触发 java.lang.OutOfMemoryError: Java heap space 异常。

内存泄漏：Java 的内存自动管理机制依赖于 GC 定期查找未使用对象并删除它们。特定的编程错误会导致你的应用程序不停的消耗更多的内存,每次使用有内存泄漏风险的功能就会留下一些 GC 无法识别的且已经不再使用的对象到堆空间中，而这些未使用的对象一直留在堆空间中随着时间的推移，泄漏的对象会消耗所有的堆空间，最终触发 java.lang.OutOfMemoryError: Java heap space 错误。

解决方法：增大-Xmx、修改程序

2.java.lang.OutOfMemoryError:GC overhead limit exceeded

GC：垃圾回收器，负责垃圾回收的模块集合

原因：超过 98%的时间用来做 GC 却回收了不到 2%的内存。具体表现为应用几乎耗尽所有可用内存，并且 GC 多次均未能清理干净。

解决方法：修改程序

3.java.lang.OutOfMemoryError:Permgen space

原因：

太多的类或者太大的类被加载到 permanent generation(持久代)，导致持久代所在区域的内存被耗尽。

许多第三方库以及糟糕的资源处理方式在每次重新部署过程中,应用程序所有的类的先前版本将仍然驻留在 Permgen 区中。结果就是，每次部署都将生成几十甚至几百 M 的垃圾。导致 Permgen 区的内存一直增加直到出现 Permgen space 错误。

解决方法：

初始化时的报错：应用程序需要更多的空间来加载所有的类到 PermGen 区域，所以我们只需要增加它的大小。增大-XX：MaxPermSize 参数

Redeploy 时的报错：

首先，找出引用在哪里被持有；其次，给你的 web 应用程序添加一个关闭的 hook，或者在应用程序卸载后移除引用。

如果是你自己代码的问题请及时修改，如果是第三方库，请试着搜索一下

是否存在"关闭"接口，如果没有给开发者提交一个 bug 或者 issue 吧。

解决运行时报错：首先你需要检查是否允许 GC 从 PermGen 卸载类，JVM 的标准配置相当保守，只要类一创建，即使已经没有实例引用它们，其仍将保留在内存中，特别是当应用程序需要动态创建大量的类但其生命周期并不长时，在启动脚本中添加以下配置参数来实现允许卸载类：-XX:+CMSClassUnloadingEnabled。如果无法解决，利用工具寻找应该卸载却没被卸载的类加载器，然后对该类加载器加载的类进行排查，找到可疑对象，分析使用或者生成这些类的代码，查找产生问题的根源并解决它。

4.java.lang.OutOfMemoryError:Metaspace

原因：太多的类或太大的类被加载到元空间。

解决方法：增加-XX：MaxMetaspaceSize、删除-XX：MaxMetaspaceSize 参数(默认没有限制)

5.java.lang.OutOfMemoryError:Unable to create new native thread

原因：Java 应用程序已达到其可以启动线程数量的极限了。最大线程数依赖于物理配置和系统内核

解决方法：增大 ulimit、程序已经出现了很严重的编程错误

6、java.lang.OutOfMemoryError:Out of swap space

原因：

交换空间即将耗尽，并且由于缺少可用的物理内存和交换空间。

操作系统配置的交换空间不足。

系统上的另一个进程消耗所 s 有内存资源。

本地内存泄漏导致应用程序失败

注：该错误消息中包含分配失败的大小(以字节为单位)和请求失败的原因。

解决方法：增加交换空间、增大物理内存、优化应用程序以减少其内存占用

7.java.lang.OutOfMemoryError:Requested array size exceeds VM limit

原因：应用程序试图分配大于 Java 可以分配的最大数组。该错误由 native code 抛出。不同的系统内核和 JDK 版本都会影响最大数组的值。

解决方法：减少数组的大小，或者将数组分成更小的数据块，然后分批处理数据。

注：Java 数组是由 int 索引的，因此，数组不能超过 2^31-1 个元素。

8、Out of memory:Kill process or sacrifice child

当内核检测到系统内存不足时，内存杀手(Out of memory killer)被激活，然后选择一个占用内存最多的进程被杀死。

解决方法：增大物理内存

Varnish 代理服务器：一款高性能且开源的反向代理服务器，具有性能高、速度快、管理方便等优点

1.安装,需要依赖包(环境已预装):gcc,readline-devel,ncurses-devel,pcre-devel :

```
[root@proxy lnmp_soft]# yum -y install
python-docutils-0.11-0.2.20130715svn7687.el7.noarch.rpm
[root@proxy lnmp_soft]# useradd -s /sbin/nologin varnish
[root@proxy lnmp_soft]# tar -xf varnish-5.2.1.tar.gz
[root@proxy lnmp_soft]# cd varnish-5.2.1/
[root@proxy varnish-5.2.1]# ./configure > /dev/null
[root@proxy varnish-5.2.1]# make > /dev/null
```

[root@proxy varnish-5.2.1]# make install > /dev/null

2.复制配置文件模板

[root@proxy varnish-5.2.1]# cp etc/example.vcl /usr/local/etc/default.vcl

3.修改配置

```
[root@proxy ~]# vim  /usr/local/etc/default.vcl
    backend default {
        .host = "192.168.2.100";
        .port = "80";
    }
```

4.启动服务：[root@proxy ~]# varnishd -f /usr/local/etc/default.vcl

可以加参数：

使用文件作为缓存，输入文件名和大小，文件不需提前创建：-s file,/var/lib/varnish_storage.bin,1G

使用内存作为缓存，设置大小：-s malloc,200M

5.查看日志：

```
[root@proxy ~]# varnishlog          #更详细的访问日志（实时）
[root@proxy ~]# varnishncsa         #访问日志（实时）
```

6.手动更新缓存，默认 2 分钟左右自动更新

```
[root@proxy ~]# varnishadm
    ban req.url ~ .*
```

加密，解密：

加密的目的：确保数据的机密性、保护信息的完整性

常见的加密算法：对称加密、非对称加密、哈希散列（用户信息摘要，不可逆）

MD5 完整性校验：md5sum

GPG(GNU Privacy Guard)工具：数据加密、数字签名

参数：

```
-c      仅使用对称加密，需要输入两次密码，源文件不消失
-e      加密
-d      解密，默认将结果输出在屏幕，可使用重定向保存，加密文件不消失
-a      以 ASCII 码输出，增加可读性。只给电脑看时可不加。
-k      查看公钥
-K      查看私钥
--gen-key   生成一对密钥
--export        导出公钥
--import        导入公钥
--delete-secret-keys        删除私钥
--delete-key                删除公钥
```

1.安装：[root@proxy ~]# yum -y install gnupg2

2.对称加密：

```
[root@proxy ~]# gpg -c host
[root@proxy ~]# gpg -d host.gpg > 1
```

3.非对称加密，按提示操作：[root@proxy ~]# gpg --gen-key

若生成密钥时，因为随机数不够导致生成太慢，可执行：

[root@proxy ~]# yum -y install rngd

[root@proxy ~]# rngd -r /dev/urandom

4.查看公钥：[root@proxy ~]# gpg --list-keys

5.删除公钥：

若是本机生成的公钥，需要先删除私钥：[root@web1 ~]# gpg --delete-secret-keys test1

删除公钥：[root@web1 ~]# gpg --delete-key test1

6.导出公钥：

若有多个公钥，需要指定公钥的uid；只有一个公钥时，可省略

[root@proxy ~]# gpg --export -a test1 > a.pub

7.传递公钥，可通过scp，ftp，http等发布：

[root@proxy ~]# scp a.pub 192.168.2.100:~

8.客户机导入公钥：[root@web1 ~]# gpg --import a.pub

9.客户机使用公钥加密：[root@web1 ~]# gpg -e -r test1 love

10.传回服务端

11.服务端解密：[root@proxy ~]# gpg -d love.gpg

软件签名与验证：软件官方以私钥对软件包执行数字签名，用户下载软件包和官方公钥，用公钥验证软件包签名。

-b　　分离式签名,默认使用第一个私钥对,如果需要使用其他私钥,可用--default-key指定私钥

--verify　　验证签名

1.创建测试文件：[root@client ~]# tar -zcf log.tar.gz /var/log/

2.创建分离式数字签名：[root@client ~]# gpg -b log.tar.gz

3.将文件、签名、公钥传给用户：[root@client ~]# scp log.tar.gz log.tar.gz.sig a.pub 192.168.4.5:~

4.用户导入公钥后验证（需要将签名放在文件前）：[root@proxy ~]# gpg --verify log.tar.gz.sig log.tar.gz

AIDE(Advanced Intrusion Detection Environment)：入侵检测系统

1.装包：[root@proxy ~]# yum -y install aide

2.修改配置：

[root@proxy ~]# vim /etc/aide.conf

3:@@define DBDIR /var/lib/aide　　#数据库目录

4:@@define LOGDIR /var/log/aide　　#日志目录

7:database=file:@@{DBDIR}/aide.db.gz　　#比对数据库文件名

12:database_out=file:@@{DBDIR}/aide.db.new.gz　　#校验结果输出的数据库名

20:report_url=file:@@{LOGDIR}/aide.log　#日志文件

26~52:可以检测的项目

#u:　　user

#g:　　group

#s:　　size

#md5:　md5 checksum

```
#sha1:  sha1 checksum
54~93:设置检测规则，不同的规则包括不同的项目组合
FIPSR = p+i+n+u+g+s+m+c+acl+selinux+xattrs+sha256
ALLXTRAHASHES = sha1+rmd160+sha256+sha512+tiger
NORMAL = sha256
99~312:设置对什么目录进行什么校验
/boot/   CONTENT_EX
/bin/    CONTENT_EX
/sbin/   CONTENT_EX
/lib/    CONTENT_EX
/lib64/  CONTENT_EX
/root/   CONTENT_EX
!/usr/tmp/                    # !表示不检验的目录
```

3.入侵前对数据进行校验，生成初始化数据库：

```
[root@proxy ~]# aide --init
```

4.备份数据库：如拷贝到U盘或其他目录或其他服务器

```
/var/lib/aide/aide.db.new.gz
```

5.入侵后检测：

```
[root@proxy ~]# mv /var/lib/aide/aide.db.new.gz /var/lib/aide/aide.db.gz
[root@proxy ~]# aide --check
```

扫描：以获取一些公开的、非公开信息为目的。检测潜在的风险，查找可攻击的目标，收集设备、主机、系统、软件信息，发现可利用的漏洞

扫描方式：主动探测(scan)、被动监听嗅探(sniff)、抓包(capture)

常用的分析工具：扫描器（NMAP）、协议分析（tcpdump\WireShark）

NMAP:支持ping扫描、多端口扫描、TCP/IP指纹校验,需要能连接到监控的主机或经过的路由器

装包：[root@proxy ~]# yum -y install nmap

扫描目标主机开启的服务：

常用的扫描类型：

```
-sS TCP SYN半开扫描，加快速度
-sT TCP全开扫描，默认选项
-sU UDP扫描
-sP ICMP扫描
-A  对目标系统全面分析
-O  检测系统版本
-sV 检测软件版本
[root@proxy ~]# nmap 192.168.4.100
[root@proxy ~]# nmap 192.168.4.100-200          #扫描一段主机
[root@proxy ~]# nmap 192.168.4.0/24
[root@proxy ~]# nmap 192.168.4.0/24 -p 3306     #-p表示仅扫描具体端口号
[root@proxy ~]# nmap 192.168.4.0/24 -p 22,3306
```

[root@proxy ~]# nmap -sU 192.168.4.100

[root@client ~]# nmap -A 192.168.4.5 #能看到使用端口的软件及版本

[root@client ~]# nmap -n -sP 192.168.4.5 # -n 表示不解析 DNS

tcpdump 抓包：提取 tcp 数据包的命令行工具

格式：tcpdump 选项 过滤条件

常见的选项：

-i 指定监控的网络接口

-A 转换为 ACSII 码

-w 将数据包信息保存到指定文件

-r 从指定文件读取数据包信息

常用的过滤条件：

类型：host、net、port、portrange

方向：src（源地址）、dst（目的地址）

协议：tcp、udp、ip、wlan、arp

多个条件组合：and、or、not

1.装包：[root@proxy ~]# yum -y install tcpdump

2.抓包：

[root@client ~]# tcpdump -A host 192.168.4.5

[root@proxy ~]# tcpdump -A src host 192.168.4.10

[root@proxy ~]# tcpdump -A dst host 192.168.4.10

[root@client ~]# tcpdump -i eth0 host 192.168.4.5

[root@client ~]# tcpdump -A host 192.168.4.5 and tcp port 21

[root@client ~]# tcpdump -A host 192.168.4.5 and tcp port 21 -w ftp.cap

[root@client ~]# tcpdump -r ftp.cap | egrep '(USER|PASS)'

tcpdump 可以抓到 nginx 的 basic 加密:使用 base64 编码(源码和编码一一对应)

[root@fzr ~]# echo a | base64

YQo=

[root@fzr ~]# echo YQo= | base64 -d

a

WireShark 协议分析器：网络协议分析软件，图形界面

1.装包：[root@proxy ~]# yum -y install wireshark wireshark-gnome

2.运行：[root@proxy ~]# wireshark

审计：基于事先配置的规则生成日志，记录可能发生在系统上的事件（正常或非正常行为的事件）。

审计不会为系统提供额外的安全保护,但会发现并记录违反安全策略的人及其对应的行为。

审计能够记录的日志内容：

1. 日期与事件以及事件的结果

2. 触发事件的用户

3. 所有认证机制的使用都可以被记录，如 ssh 等

4. 对关键数据文件的修改行为等都可以被记录

作用：监控文件访问、系统调用、记录用户运行的命令、监控网络访问行为

audit 审计系统：

1.装包：[root@proxy ~]# yum -y install audit

2.起服务：

[root@proxy ~]# systemctl start auditd

[root@proxy ~]# service auditd restart #该软件随内核启动,不允许使用 systemctl 关闭或重启

[root@proxy ~]# service auditd stop

3.配置临时审计规则:

语法格式：auditctl -w 文件或目录 -p 权限 -k 关键字

权限包括：r（读）、w（写）、x（执行）、a（属性改变）

关键词为搜索日志提供便捷

[root@proxy ~]# auditctl -w /etc/passwd -p wa -k passwd_change #监控文件

[root@proxy ~]# auditctl -w /etc/selinux -p wa -k selinux_change

#监控目录

[root@proxy ~]# auditctl -w /usr/sbin/fdisk -p x -k disk_partition

#监控程序

[root@proxy ~]# auditctl -s #查看状态

[root@proxy ~]# auditctl -l #查看规则

[root@proxy ~]# auditctl -D #删除所有规则

4.查看日志：

[root@proxy ~]# tail -f /var/log/audit/audit.log

type=SYSCALL msg=audit(1526608101.215:173): arch=c000003e syscall=82 success=yes exit=0 a0=7ffe0c32f970 a1=557e88873ce0 a2=7ffe0c32f8e0 a3=9 items=5 ppid=1362 pid=1903 auid=0 uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts1 ses=2 comm="useradd" exe="/usr/sbin/useradd" subj=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 key="passwd_change"

type=CWD msg=audit(1526608101.215:173): cwd="/root"

type=PATH msg=audit(1526608101.215:173): item=0 name="/etc/" inode=33554497 dev=fd:00 mode=040755 ouid=0 ogid=0 rdev=00:00 obj=system_u:object_r:etc_t:s0 objtype=PARENT

type 日志类型

SYSCALL 记录系统命令详情

CWD 用来记录当前工作目录

PATH 用来记录路径

msg 为 1970-1-1 至今的秒数,等同于 date +%s

arch=c000003e 代表 64 位（使用 16 进制表示）

success=yes/no 表示事件是否成功

a0-a3 是程序调用时前 4 个参数，16 进制编码

ppid 父进程 ID
pid 进程 ID
auid 是审核用户的 id，可以追踪 su 前的账户
uid，gid 表示用户与组
tty 表示执行命令的终端,tty 表示本地连接,pts 表示远程连接
comm="useradd" 表示用户在命令行执行的指令
exe="/usr/sbin/useradd"表示实际程序的路径
subj 表示 selinux 规则
key="passwd_change"表示触发日志的策略关键字 key
cwd="/root"表示执行时的$PWD
ouid(owner's user id) 表示对象所有者 id
ogid(owner's groupid) 表示对象所有者 id

5.通过工具搜索：[root@proxy ~]# ausearch -k passwd_change -i　　#-i 表示人性化显示,-f 指定日志文件(查看旧的日志)

6.永久配置，修改配置文件：

[root@proxy ~]# vim /etc/audit/rules.d/audit.rules
-w /usr/sbin/fdisk -p x -k disk_partition

监控：

目的：实时报告系统状态，提前发现系统的问题。

监控的资源：共有数据（HTTP、FTP 等）和私有数据（CPU、内存、进程数等）。

监控软件：

系统自带的命令：ps ， uptime ， free ,swapon -s ,df -h ,ifconfig ,ss ,ping ,traceroute ,iostat

script：记录操作记录，并保存为文档，格式：script [文件]

Cacti 监控系统：基于 SNMP 协议的监控软件，强大的绘图能力

Nagios 监控系统：基于 Agent 监控，强大的状态检查与报警机制，插件多，可以自己写监控脚本

Zabbix 监控系统：基于多种监控机制，支持分布式监控

Jumpserver：开源的堡垒机，网址：http://www.jumpserver.org/

Zabbix:高度集成的监控解决方案，实现企业级的开源分布式监控。通过 Client/Server 模式采集监控数据，通过 B/S 模式实现 WEB 管理

监控服务器:通过 SNMP 或 Agent 采集数据,数据可以写入 MySQL 等数据库中,使用 LNMP 实现 WEB 前端管理

被监控主机：主机需要安装 Agent，网络设备一般支持 SNMP

监控服务器:

1.装包：Zabbix 监控管理控制台需要通过 Web 页面展示出来，并且还需要使用 MySQL 来存储数据

[root@server ~]# yum -y install gcc pcre-devel openssl-devel php php-mysql mariadb-server mariadb-devel

[root@server lnmp_soft]# yum -y install php-fpm-5.4.16-42.el7.x86_64.rpm

[root@server nginx-1.12.2]# ./configure --user=nginx --group=nginx --with-http_ssl_module

```
[root@server nginx-1.12.2]# make && make install
```

2.修改 Nginx 配置文件：支持 PHP 动态网站，因为有大量 PHP 脚本需要执行，因此还需要开启各种 fastcgi 缓存，加速 PHP 脚本的执行速度

```
[root@server ~]# vim /usr/local/nginx/conf/nginx.conf
    http {
    ...
    fastcgi_buffers 8 16k;          #缓存 php 生成的页面内容，8 个 16k
    fastcgi_buffer_size 32k;        #缓存 php 生成的头部信息
    fastcgi_connect_timeout 300;    #连接 PHP 的超时时间
    fastcgi_send_timeout 300;       #发送请求超时时间
    fastcgi_read_timeout 300;       #读取请求超时时间
    ...
        server {
            location ~ \.php$ {配置}
    }}
```

3.启动 Nginx、PHP-FPM、MariaDB

4.源码安装 Zabbix Server:

```
--enable-server #安装部署 zabbix 服务器端软件
--enable-agent  #安装部署 zabbix 被监控端软件
--enable-proxy  #安装部署 zabbix 代理相关软件
--with-mysql    #配置 mysql_config 路径，/usr/bin/mysql_config 由 mariadb-devel 提供
--with-net-snmp #允许 zabbix 通过 snmp 协议监控其他设备
--with-libcurl  #安装相关 curl 库文件,这样 zabbix 就可以通过 curl 连接 http 等服务，测试被监控主机服务的状态
[root@server lnmp_soft]# yum -y install net-snmp-devel libcurl-devel libevent-devel-2.0.21-4.el7.x86_64.rpm
[root@server zabbix-3.4.4]# ./configure --enable-server --enable-proxy --enable-agent --with-mysql=/usr/bin/mysql_config --with-net-snmp --with-libcurl
[root@server zabbix-3.4.4]# make && make install
```

5.创建数据库

```
MariaDB [(none)]> create database zabbix character set utf8;    #创建数据库，支持中文字符集
MariaDB [(none)]> grant all on zabbix.* to zabbix@'localhost' identified by 'zabbix';
[root@server zabbix-3.4.4]# cd database/mysql/                  #zabbix 源码包目录下，有提前准备好的测试数据
[root@server mysql]# mysql zabbix < schema.sql                  #按顺序导入
[root@server mysql]# mysql zabbix < images.sql
[root@server mysql]# mysql zabbix < data.sql
```

6.上线 Zabbix 的 Web 页面：

```
[root@server zabbix-3.4.4]# cd frontends/php/
```

```
[root@server php]# cp -r * /usr/local/nginx/html/
[root@server ~]# chmod -R 777 /usr/local/nginx/html/*
```

7.修改 zabbix_server 的配置文件

```
[root@server ~]# vim /usr/local/etc/zabbix_server.conf
38:LogFile=/tmp/zabbix_server.log       #日志文件
85:DBHost=localhost                     #解除注释，表示数据库在本地
95:DBName=zabbix                        #数据库名称
111:DBUser=zabbix                       #数据库账户
119:DBPassword=zabbix                   #数据库密码
```

8.启动 zabbix_server：如果因为配置文件不对，导致服务无法启动时，不要重复执行 zabbix_server，一定要先关闭服务后，再重新启动一次。

```
[root@server ~]# useradd -s /sbin/nologin -r -N zabbix  #不创建用户无法启动
[root@server ~]# zabbix_server
[root@server ~]# killall zabbix_server
[root@server ~]# ss -antup | grep zabbix_server
```

9.修改 zabbix_agent 的配置文件：

```
[root@server ~]# vim /usr/local/etc/zabbix_agentd.conf
30:LogFile=/tmp/zabbix_agentd.log       #日志文件
93:Server=127.0.0.1,192.168.2.5         #允许通过被动模式监控本机的 IP
134:ServerActive=127.0.0.1,192.168.2.5  #允许通过主动模式监控本机的 IP
145:Hostname=server                     #设置本机主机名
280:UnsafeUserParameters=1              #是否允许自定义监控项，0 表示不允许，1 表示允许
```

10.启动 zabbix_agentd：

```
[root@server ~]# zabbix_agentd
[root@server ~]# killall zabbix_agentd
[root@server ~]# ss -ntulp |grep zabbix_agentd
```

11.浏览器访问 zabbix 服务器的 Web 页面，根据错误提示安装依赖包，修改配置文件：

```
[root@server lnmp_soft]# yum -y install  php-gd php-xml php-bcmath-5.4.16-42.el7.x86_64.rpm php-mbstring-5.4.16-42.el7.x86_64.rpm
[root@server ~]# vim /etc/php.ini
384:max_execution_time = 300        #最大执行时间
394:max_input_time = 300            #接收数据的时间限制
672:post_max_size = 16M             #post 最大容量
878:date.timezone = Asia/Shanghai   #时区
[root@server ~]# systemctl restart php-fpm.service
```

12.刷新浏览器，根据提示初始化页面并登陆

被监控主机：

1.装包：

```
[root@web1 ~]# useradd -s /sbin/nologin -r -N zabbix
[root@web1 ~]# yum -y install gcc pcre-devel
[root@web1 zabbix-3.4.4]# ./configure --enable-agent
[root@web1 zabbix-3.4.4]# make && make install
```

2.修改 zabbix_agent 的配置文件：

[root@web1 ~]# vim /usr/local/etc/zabbix_agentd.conf

69:EnableRemoteCommands=1　　#监控异常后，是否允许服务器通过远程执行命令。0 表示不允许，1 表示允许

93:Server=127.0.0.1,192.168.2.5

134:ServerActive=127.0.0.1,192.168.2.5

145:Hostname=web1

280:UnsafeUserParameters=1

3.拷贝源码包目录下的启动脚本，方便管理服务：

[root@web1 zabbix-3.4.4]# cp misc/init.d/fedora/core/zabbix_agentd /etc/init.d/

[root@web1 ~]# service zabbix_agentd start

[root@web1 ~]# service zabbix_agentd status

[root@web1 ~]# service zabbix_agentd stop

[root@web1 ~]# service zabbix_agentd restart

Web 端配置：

1.添加监控主机：

通过 Configuration（配置）-->Hosts（主机）-->Create Host（创建主机）添加被监控 Linux 主机

添加被监控主机时，需要根据提示输入被监控 Linux 主机的主机名称、主机组、IP 地址等参数

2.为被监控主机添加监控模板

通过监控模板来对监控对象实施具体的监控功能，对于 Linux 服务器的监控，Zabbix 已经内置了相关的模板 Template OS Linux

3.查看监控数据

Monitoring(监控中)—> Latest data(最新数据)，正过滤器中填写过滤条件，根据监控组和监控主机选择需要查看的监控数据

4.如果检测不到数据：

a)　先检查连通性：[root@server ~]# zabbix_get -s 192.168.2.100 -k system.cpu.load

b)　检查客户端配置文件

自定义监控项：

1.修改配置：自定义的 key 文件一般存储在/usr/local/etc/zabbix_agentd.conf.d/目录

[root@web1 ~]# vim /usr/local/etc/zabbix_agentd.conf +264

Include=/usr/local/etc/zabbix_agentd.conf.d/

2.创建自定义监控项：UserParameter=自定义键值名称,命令或脚本

[root@web1 zabbix_agentd.conf.d]# vim test

UserParameter=test,wc -l /etc/passwd | awk '{print $1}'　　#区分大小写

3.重起服务并在监控端测试：

[root@server ~]# zabbix_get -s 192.168.2.100 -k test

自定义监控模板：

1.在监控控制台通过 Configuration(配置)-->Template(模板)-->Create template(创

建模板)，填写模板名称，新建模板群组

2.点击模板后面的 Application（应用集），打开创建应用的页面，点击 Create application（创建应用集）按钮，设置应用名称

3.点击模板后面的 item（监控项），选择 Create items（创建监控项）创建项目，给项目设置名称及对应的自定义键值，选择应用集

4.点击模板后面的 Graph（图形），设置图形基于的监控项，便于后期通过图形的方式展示监控数据

5.找到被监控主机 Configuration（配置）-->Hosts（主机），打开监控主机设置页面，在 Template（模板）页面中选择新建的模板群组，添加新建的模板

6.点击 Monitoring（监控中）-->Graphs（图形），根据需要选择条件，查看监控图形

Zabbix 的报警机制：

自定义的监控项默认不会自动报警，首页也不会提示错误，需要配置触发器和报警动作

触发器（trigger）：当触发条件发生后，会导致一个触发事件，触发事件会执行预定的动作

动作（action）：触发器被触发后的行为

1.通过 Configuration（配置）--> Templates（模板），点击模板后面的 triggers（触发器）

2.创建触发器

a) 手工创建需要定义表达式，触发器表达式（Expression）是触发异常的条件，触发器表达式格式如下：

{<server>:<key>.<function>(<parameter>)}<operator><constant>

{主机：key.函数(参数)}<表达式>常数

大多数函数使用秒作为参数，可以使用#来表示其他含义

avg, count, last, min and max 等函数支持额外的第二个参数 time_shift（时间偏移量），这个参数允许从过去一段时间内引用数据。

函数内容	描述
sum(600)	600秒内所有值的总和
sum(#5)	最后5个值的总和
last(20)	最后20秒的值
last(#5)	倒数第5个值
avg(1h,1d)	一天前的1小时的平均值

b) 自动配置触发器：设置触发器名称，add 添加表达式，选择监控项，设置变化时间和常数，选择触发器报警级别。

3.通过 Administration（管理）-->Media Type（报警媒体类型）-->选择 Email（邮件），设置邮件服务器信息，设置邮件服务器及邮件账户信息

4.在 Administration（管理）-->Users（用户）中点击 admin 账户

a) 选择 Media（报警媒介）菜单-->点击 Add(添加)报警媒介,点击 Add（添加）后，填写报警类型，收件人，时间等信息

5.通过 Configuration（配置）-->Actions（动作）-->Create action（创建动作）

a) 在动作选项下填写 Action 动作的名称,配置触发器被触发时会执行的 Action 动作

b) 在操作选项下添加出现问题后的操作

c) 在恢复操作选项下添加问题恢复后的操作

Zabbix 自动发现：

概念：当 Zabbix 需要监控的设备越来越多，使用自动发现功能，自动添加被监控主机，实现自动批量添加一组监控主机功能

功能：自动发现、添加主机，自动添加主机到组；自动连接模板到主机，自动创建监控项目与图形等。

1.通过 Configuration（配置）-->Discovery（自动发现）-->Create discovery rule（创建发现规则）

a) 填写自动发现的 IP 范围（逗号，横线都可以），自动发现的时间间隔，配置检查类型和端口范围

2.通过 Configuration（配置）--> Actions（动作）-->Actions Event source(事件源)选择：Discovery（自动发现）-->Create action（创建动作）

a) 在动作选项下添加动作名称，添加触发动作的条件：自动发现规则、自动发现状态、服务类型

b) 在操作选项下添加操作，添加主机、添加到主机群组、链接到模板

3.在/etc/hosts 下设置 ip 对应的主机名可自动显示主机名

Zabbix 主动监控：

被动监控：Server 向 Agent 发起连接，发送监控 key，Agent 接受请求，响应监控数据。

主动监控：Agent 向 Server 发起连接，Agent 请求需要检测的监控项目列表，Server 响应 Agent 发送一个 items 列表，Agent 确认收到监控列表后开始周期性收集数据。

优势：Server 不用每次需要数据都连接 Agent，Agent 会自己收集数据并处理数据，Server 仅需要保存数据即可，减轻 Server 压力

1.修改 Agent 端的配置文件：

```
[root@web2 ~]# vim /usr/local/etc/zabbix_agentd.conf
93:#Server=127.0.0.1,192.168.2.5          #被动模式 Server 端 IP，注释该行
118:StartAgents=0                         #被动模式启动的进程数，0 表示关闭被动模式
134：ServerActive=192.168.2.5             #主动模式 Server 端 IP，删除本机 IP
151:Hostname=web2                         #本机主机名，要和 Server 端的监控主机名一致
183:RefreshActiveChecks=120               #检测间隔
```

2.重起 zabbix_agentd 服务，不占用端口

3.创建主动监控的监控模板：

a) 通过 Configuration（配置）-->Templates（模板）-->选择 Template OS Linux--> 全克隆，克隆出一个新的模板，名称为：Template OS Linux ServerActive

b) 修改监控项目为主动监控模式：选择克隆的模板，点击 Items（监控项）-->点击全选，点击批量更新，将类型修改为：Zabbix Agent（Active 主动模式）

c) 部分监控项目不支持主动模式，无法被批量更新，关闭这些项目

d) 将自动发现规则全部修改为主动式

4.手动添加监控主机：IP 地址设置为 0.0.0.0，端口设置为 0，链接新建的模板。其余和被动模式相同

创建 zabbix 拓扑图：绘制拓扑图可以快速了解服务器架构

1.通过 Monitoring（监控中）-->Maps（拓扑图），编辑默认的拓扑图，也可以新建一个

2.作图：

Icon（图标）：添加新的设备后可以修改图标

Shape（形状）

Link（连线）：先选择两个图标，再选择连线

创建 zabbix 聚合图形：在一个页面显示多个数据图表，方便了解多组数据

1.通过 Monitoring（监控中）-->Screens（聚合图形）-->Create screen(创建聚合图形)即可创建聚合图形

2.添加监控图形：选择刚刚创建的聚合图形，点击构造函数（constructor），点击 Change(更改)，设置每行每列需要显示的数据图表

Zabbix 自定义监控项：

Nginx 监控：

1.status 监控模块：--with-http_stub_status_module

2.自定义监控项：

```
[root@web1 ~]# vim /usr/local/etc/zabbix_agentd.conf.d/nginx.status
    UserParameter=nginx.status[*],/usr/local/bin/nginx_status.sh $1
```

3.自定义监控脚本：

```
[root@web1 ~]# vim /usr/local/bin/nginx_status.sh
    #!/bin/bash
    case $1 in
    active)
    curl -s http://192.168.2.100/status |awk '/Active/{print $NF}';;
    accepts)
    curl -s http://192.168.2.100/status |awk 'NR==3{print $2 }';;
    esac
[root@web1 ~]# chmod +x /usr/local/bin/nginx_status.sh
```

4.重起服务测试：

```
[root@web1 ~]# service zabbix_agentd restart
[root@server ~]# zabbix_get -s 192.168.2.100 -k 'nginx.status[accepts]'
```

5.在 web 中创建监控项：

点击 Configuration（配置）-->Hosts(主机)，点击主机后面的 items（监控项），点击 Create item（创建监控项），修改项目参数

监控网络连接状态：

1.自定义监控项：

```
[root@web1 ~]# vim /usr/local/etc/zabbix_agentd.conf.d/net.status
    UserParameter=net.status[*],/usr/local/bin/net_status.sh $1
```

2.自定义监控脚本：

```
[root@web1 ~]# vim /usr/local/bin/net_status.sh
    #!/bin/bash
    case $1 in
    TIME-WAIT)
    ss -antp |awk '/TIME-WAIT/{x++} END{print x}';;
    ESTAB)
    ss -antp |awk '/ESTAB/{x++} END{print x}';;
    esac
```

3.+x，重起，测试

4.在 web 中创建监控项

打补丁：

diff：逐行比较，如果修改第一个文件可以得到第二个文件

diff 选项 第一个文件 第二个文件

常用选项：

-u 输出统一内容的头部信息，让计算机知道哪些文件需要修改

-r 递归对比目录中的所有资源

-a 将所有文件视为文本（包括二进制）

-N 如何从无文件或空文件修改为第二个文件

[root@proxy ~]# diff -u /tmp/1.sh /tmp/2.sh > /tmp/test.patch

patch：打补丁

格式：patch -p 数字 #数字指定删除补丁文件中多少层路径前缀

-p 0 表示不去前缀，保持整个路径不变

-p 1 表示去掉一层前缀

安装：[root@proxy ~]# yum -y install patch

1.环境准备：

```
[root@proxy tmp]# mkdir test{1,2}
[root@proxy tmp]# echo "hello" > test1/test.sh
[root@proxy tmp]# cp /bin/find test1/
[root@proxy tmp]# echo "hello world" > test2/test.sh
[root@proxy tmp]# echo "test" > test2/temp.txt
[root@proxy tmp]# cp /bin/find test2/
[root@proxy tmp]# echo 1 >> test2/find
```

2.制作补丁文件：

```
[root@proxy tmp]# diff -u test1 test2/              #仅比对了文本文件
[root@proxy tmp]# diff -uN test1 test2/          #增加了比对空文件
[root@proxy tmp]# diff -uNa test1 test2/            #增加了比对程序
[root@proxy tmp]# diff -uNar test1 test2/ > test.patch  #增加了对子目录的对比，并生成补丁文件
```

3.进入需要打补丁的目录，打补丁：

```
[root@proxy tmp]# cd test1/
[root@proxy test1]# patch -p1 < ../test.patch       #去掉第一层前缀 test/
```

4.打补丁后还原：

```
[root@proxy test1]# patch -RE < ../test.patch       #-R 还原,-E 如果还原后的文件为空文件，则删除
```

数据库概念：

DB(DataBase)数据库：依照某种数据模型进行组织并存放到存储器的数据集合

DBMS(DataBase Management System)数据库管理系统：用来操纵和管理数据库的大型服务软件

RDBMS(Relation DataBase Management System)关系型数据库管理系统

DBS(DataBase System)数据库系统：即 DB+DBMS ：指带有数据库并整合了数据库管理软件

的计算机系统
NoSQL(Not Only SQL)：非关系型数据库

关系型数据库：
采用实体-关系模型(E-R 模型：Entity-Relationship Model)
数据按照预先设置的组织结构,存储在物理存储介质上，数据之间可以做关联操作
软件：
Oracle Database：甲骨文公司，商业软件，闭源软件
MySQL：甲骨文公司，开源软件，应用于所有平台
MariaDB：开源软件，应用于所有平台
SQL Server：微软公司，商业软件，闭源软件，应用于 Windows
Access：微软公司，商业软件，闭源软件，应用于 Windows
DB2：IBM 公司，商业软件，闭源软件，应用于所有平台
Sybase：Sybase 公司，开源软件
PostgreSQL：加州大学伯克利分校，开源软件
非关系型数据库：
不需要预先定义数据存储结构,表的每条记录都可以有不同的类型和结构
软件：Redis,MongoDB,Memcached

RHEL7 内置的数据库是 MariaDB
[root@server0 ~]# yum -y install mariadb-server mairadb #前者是服务器，后者是客户端，使用客户端软件连接服务器做增，删，改，查等操作
[root@server0 ~]# systemctl start mariadb #启动服务
[root@server0 ~]# mysql #默认 mariadb 数据库仅有一个账户 root，且不设置密码
MariaDB [(none)]> show databases; #查看有哪些数据库(数据库就是目录)
MariaDB [(none)]> use mysql; #进入数据库(mysql)
MariaDB [mysql]> show tables; #查看有哪些数据表
MariaDB [mysql]> select * from user; #查看数据表里的数据
MariaDB [(none)]> create database b; #创建一个新的数据库，名称为 b
MariaDB [(none)]> drop database b; #删除某个数据库
MariaDB [(none)]> create database nb;
MariaDB [(none)]> use nb;
MariaDB [(nb)]> create table info(id int,name varchar(10),password varchar(20)); #创建数据表，创建的数据库都在/var/lib/mysql 目录
#创建一个表,表名是 info，第 1 列是 id，类型是数字，第 2 列是 name，类型是字符串,10 位，第 3 列是 password，类型是字符串，20 位
MariaDB [(nb)]> insert into info values(1,'tom','123456'); #往 info 这个表里查入数据，id 号,名字，密码
MariaDB [(nb)]> insert into info values(3,'jerry','123456'),(4,'lucy','123456');
MariaDB [(nb)]> select name,password from info; #仅查看表中的用户和密码信息[不显示 id 信息]
MariaDB [(nb)]> select * from info where password='123456' #查看 info

表中 password 是 123456 的所有数据

MariaDB [(nb)]> update info set password='123' where name='lily'; #仅当用户名是 lily 时才更新她的密码为 123

MariaDB [(nb)]> select * from info where password='123456' and id=1 #列出 info 表中的数据，密码为 123456 并且 id 等于 1

MariaDB [(nb)]> select * from info where password='123456' or id=1 #列出 info 表中的数据，密码为 123456 或者 id 等于 1

MariaDB [nb]> drop table info; #删除数据表 info

MariaDB [(nb)]> describe info; #查看表结构

MariaDB [(none)]> grant all on nb.* to tom@'localhost' identified by '123456';

#创建 tom 用户，只能本地登陆，若需要远程登陆，@后需要写上 IP 地址，拥有对 nb 数据库下的所有表的所有权限，也可以写具体的指令[select,insert]，identified by 设置密码

MariaDB [mysql]> exit #退出数据库系统

[root@server0 ~]# mysqladmin -uroot password '123456' #给 mariadb 数据库的管理员 root 修改密码,密码为 123456

[root@server0 ~]# mysql #直接登陆数据库会失败

[root@server0 ~]# mysql -u root -p123456 #使用用户与密码登陆

[root@server0 ~]# mysql -u tom -p123456

数据库的备份与还原

[root@server0 ~]# mysqldump -uroot -p123456 nb > my.bak #把名称为 nb 的数据库备份，备份到 my.bak 文件

[root@server0 ~]# mysql -uroot -p123456 nb < my.bak #使用 my.bak 这个备份文件，还原名称为 nb 的数据库，需要数据库中有 nb 这个数据库

多表查询

MariaDB [(nb)]> SELECT count(*) FROM base,location WHERE base.name="Barbara" AND location.city="Sunnyvale" AND base.id=location.id;

mysql 和 oracle 的区别：

Oracle 是大型数据库，支持大并发，大访问量；Mysql 是中小型数据库

Mysql 是开源的，Oracle 价格非常高。

Mysql 只需要 152M，Oracle 需要 3G 左右

使用的时候 Oracle 占用特别大的内存空间和其他资源。

操作上的区别：

①自动增长类型的处理：

MYSQL 有自动增长的数据类型，插入记录时不用操作此字段，会自动获得数据值

ORACLE 没有自动增长的数据类型，需要建立一个自动增长的序列号，插入记录时要把序列号的下一个值赋于此字段。

②单引号的处理：

MYSQL 里可以用双引号包起字符串

ORACLE 里只可以用单引号包起字符串。

③分页的 SQL 语句的处理：

MYSQL 处理分页的 SQL 语句比较简单，用 LIMIT 开始位置,记录个数

ORACLE 处理分页的 SQL 语句就比较繁琐了。每个结果集只有一个 ROWNUM 字段标明它的位置,并且只能用 ROWNUM<100,不能用 ROWNUM>80

④长字符串的处理：

ORACLE 使用 INSERT 和 UPDATE 时最大可操作的字符串长度是 4000 个单字节,如果要插入更长的字符串,字段考虑用 CLOB 类型,方法借用 ORACLE 里自带的 DBMS_LOB 程序包。

ORACLE 插入修改记录前一定要做进行非空和长度判断，不能为空的字段值和超出长度字段值都应该提出警告。

⑤空字符的处理：

MYSQL 的非空字段允许为空字符串

ORACLE 里定义了非空字段就不容许为空字符串。

⑥字符串的模糊比较：

MYSQL 里用字段名 like'%字符串%'

ORACLE 里也可以用字段名 like'%字符串%'但这种方法不能使用索引,速度不快。

⑦日期字段的处理：

MYSQL 日期字段分 DATE 和 TIME 两种

ORACLE 日期字段只有 DATE，包含年月日时分秒信息，使用当前数据库的系统时间为 SYSDATE，精确到秒

MySQL：

主要特点：

适用于中小规模、关系型数据库系统

支持 Linux/Unix、Windows 等多种操作系统

使用 C 和 C++编写，可移植性强

通过 API 支持 Python/Java/Perl/PHP 等语言

典型应用环境：

LAMP 平台，与 Apache HTTP Server 组合

LNMP 平台，与 Nginx 组合

相关参数：

/etc/my.cnf	主配置文件
/var/lib/mysql	数据库目录
默认端口号	3306
进程名	mysqld
传输协议	TCP
进程所有者	mysql
进程所属组	mysql

数据库对应/var/lib/mysql 目录：

库----文件夹

表----文件

记录--行

字段--列

操作指令类型：

MySQL 指令：环境切换、看状态、退出等控制

SQL 指令：数据库定义/查询/操纵/授权语句

DDL 数据定义语言：create、alter、drop

DML 数据操作语言：insert、update、delete

DCL 数据控制语言：grant、revoke

DTL 数据事务语言：commit、rollback、savepoint

基本注意事项

1. 操作指令不区分大小写（密码、变量值除外）
2. 每条 SQL 指令以 ; 结束或分隔
3. 不支持 Tab 键自动补齐
4. ctrl+c 可废弃当前编写错的操作指令

MySQL8 的新功能：

1．在读/写工作负载、IO 密集型工作负载、以及高竞争(热点竞争问题)工作负载都有很大提高。

2．对 NoSQL 存储功能的支持得到了更大的改进。该项功能消除了对独立的 NoSQL 文档数据库的需求，而 MySQL 文档存储也为 schema-less 模式的 JSON 文档提供了多文档事务支持和完整的 ACID 合规性

3．新增了窗口函数(Window Functions)的概念，它可以用来实现若干新的查询方式。例如：将多行查询结果的结果放回多行当中。

4．索引可以被“隐藏”和“显示”。当对索引进行隐藏时，它不会被查询优化器所使用。我们可以使用这个特性用于性能调试：先隐藏一个索引，如果数据库性能有所下降，说明这个索引是有用的;如果数据库性能看不出变化，说明这个索引是多余的，可以考虑删掉。

5．为索引提供按降序方式进行排序的支持，在这种索引中的值也会按降序的方式进行排序。

6．通用表表达式(Common Table Expressions:CTE)：在复杂的查询中使用嵌入式表时使用,使得查询语句更清晰。

7．使用 utf8mb4 作为 MySQL 的默认字符集。

8．大幅改进了对 JSON 的支持，添加了基于路径查询从 JSON 字段中抽取数据的 JSON_EXTRACT()函数，以及用于将数据分别组合到 JSON 数组和对象中的 JSON_ARRAYAGG()和 JSON_OBJECTAGG()聚合函数。

9．InnoDB 支持表 DDL 的原子性，不至于出现 DDL 时部分成功的问题，此外还支持 crash-safe 特性，元数据存储在单个事务数据字典中。

10．InnoDB 集群为数据库提供集成的原生 HA 解决方案。

11．对 OpenSSL 的改进、新的默认身份验证、SQL 角色、密码强度、授权。

安装 MySQL：

```
mysql-community-client          #客户端应用程序
mysql-community-common          #数据库和客户端库共享文件
mysql-community-devel           #客户端应用程序的库和头文件
mysql-community-embedded        #嵌入式函数库
mysql-community-embedded-compat #嵌入式兼容函数库
mysql-community-embedded-devel  #头文件和库文件作为 Mysql 的嵌入式库文件
mysql-community-libs            #MySQL 数据库客户端应用程序的共享库
```

```
mysql-community-libs-compat      #客户端应用程序的共享兼容库
mysql-community-server           #服务端软件
[root@host50 ~]# yum -y install mysql-community-*
```

首次启动时，正常进行数据初始化并创建随机密码：

```
[root@host50 ~]# systemctl start mysqld.service
[root@host50 ~]# ps -e | grep mysql
[root@host50 ~]# ss -antup | grep mysql
```

首次启动时，初始化为空密码且不启动安全策略：

```
[root@host1 ~]# mysqld --initialize-insecure --user=mysql
[root@host1 ~]# systemctl start mysqld
```

首次登陆 MySQL：默认的数据库管理账号 root，仅允许从 localhost 访问，登录密码在安装时随机生成，存储在/var/log/mysqld.log

```
[root@host50 ~]# grep password /var/log/mysqld.log
[root@host50 ~]# mysql -h localhost -u root -p
```

修改 root 密码：

```
mysql> set global validate_password_policy=0;        #只验证长度
mysql> set global validate_password_length=6;        #修改密码长度,默认值是8个字符，最小值 4 个字符
mysql> alter user user() identified by "123456";     #修改密码
[root@host50 ~]# vim /etc/my.cnf
    validate_password_policy=0
    validate_password_length=6
```

不进入 mysql 修改 root 密码：需要输入原密码

```
[root@host50 ~]# mysqladmin -u root -p password "654321"
```

恢复 root 密码：

```
[root@host50 ~]# vim /etc/my.cnf          #重起 mydql
    #validate_password_policy=0           #注释对授权表的定义
    #validate_password_length=6
    skip-grant-tables                     #跳过授权表
mysql> update mysql.user set authentication_string=password("123456") where user="root" and host="localhost";
mysql> flush privileges;                  #任何修改授权库 mysql 的操作，刷新后才会生效
[root@host50 ~]# vim /etc/my.cnf          #重起 mydql
    #skip-grant-tables
```

连接数据库：

```
Mysql -h 服务器  -u 用户名  -p 密码  数据库名
quit 或 exit 或 ctrl+d 退出
```

重新初始化 mysql：

```
[root@host53 ~]# systemctl stop mysqld.service          #关闭服务
[root@host53 ~]# rm -rf /var/lib/mysql/*                #删除数据库目录
[root@host53 ~]# vim /etc/my.cnf                        #还原配置文件，将添加的内容注释
[root@host53 ~]# mysql_secure_installation              #启动还原脚本
```

```
[root@host53 ~]# systemctl start mysqld                  #启动服务
[root@host53 ~]# vim /etc/my.cnf                         #去掉注释
[root@host53 ~]# systemctl restart mysqld                #重起服务
[root@host53 ~]# grep password /var/log/mysqld.log       #查看密码
[root@host53 ~]# mysql -u root -p                        #登陆
mysql> set password=password("123456");                  #修改密码
```

库管理命令：

显示已有的库：show databases;

显示当前所在的库：select database();

创建新库：create database 库名;

切换库：use 库名;

显示已有的表：show tables;

删除库：drop database 库名;

库和表的命名规则：

可以使用数字、字母、下划线，但不能纯数字

区分大小写，具有唯一性

不可使用指令关键字、特殊字符

表管理命令：

新建表：create table 库名.表名(

字段名 1 字段类型(宽度) 约束条件,

...

字段名 N 字段类型(宽度) 约束条件);

查看表结构：desc 表名;

查看表记录：select 字段 1,...字段 N from 表名 where 条件表达式;

插入表记录：insert into 表名 values ("值 1",...,"值 N"),("值 1",...,"值 N");

插入部分字段：insert into 表名(字段 M,字段 N) values ("值 M","值 N"),("值 M","值 N");

修改表记录：update 表名 set 字段 1=值 1,字段 2=值 2 where 条件表达式;

删除表记录：delete from 表名 where 条件表达式;

查看创表信息：show create table 表名;

删除表：drop table 表名;

注意：如果要让数据表支持中文，需要在建表时，在末尾加 default charset=utf8，或者在主配置文件中添加 character_set_server=utf8

增加表记录注意事项：

1. 字段值要与字段类型相匹配
2. 对于字符类型的字段,要用双引号或单引号括起来
3. 依次给所有字段赋值时,字段名可以省略
4. 只给一部分字段赋值时,必须明确写出对应的字段名称
5. 记录在表的末尾

查询表记录注意事项：

1. 可用*匹配所有字段
2. 指定表名时,可采用库名.表名的形式
3. 可使用 where 条件表达式缩小查询范围

4. 不加 where 表示匹配所有

更新表记录注意事项：

1. 字段值要与字段类型相匹配
2. 对于字符类型的字段,要用双或单引号括起来
3. 若不使用 WHERE 限定条件,会更新所有记录
4. 限定条件时,只更新匹配条件的记录

删除表记录注意事项：

1. 若不使用 WHERE 限定条件,会删除所有的表记录
2. 限定条件时,仅删除符合条件的记录

案例：创建一个学生信息表

```
mysql> create table 学生(学号 char(10),姓名 char(20),性别 char(4),手机号 int,地址 char(50))default charset=utf8;
mysql> insert into 学生 value ("nsd131201","张三","男","13012345678","北京");
mysql> select * from 学生;
mysql> desc 学生;
```

常见的信息种类：

数值型：体重、身高、成绩、工资

字符型：姓名、工作单位、通信住址

枚举型：性别

日期时间型：出生日期、注册时间

数值类型：

类型	大小	范围	名称
TINYINT	1 字节	2^8	微小整数
SMALLINT	2 字节	2^{16}	小整数
MEDIUMINT	3 字节	2^{24}	中整数
INT	4 字节	2^{32}	大整数
BIGINT	8 字节	2^{64}	极大整数
FLOAT(M,D)	4 字节		单精度浮点数
DOUBLE(M,D)	8 字节		双精度浮点数

DECIMAL(M,D)M 为有效位数,D 为小数位数，M 应大于 D，占用 M+2 字节

注意：

使用 UNSIGNED 修饰类型时，对应的字段只保存正数；

数值不够指定宽度时，在左边填空格补位；使用 ZEROFILL 修饰类型时，用 0 代替空格补位，同时附带 UNSIGNED 属性

宽度仅代表最小显示宽度，数值的范围由类型决定

数值超出范围时，报错

字符类型：

char(字符数)：定长，最大长度 255 字符，不够指定字符数时在右边用空格补齐

varchar(字符数)：变长，最大长度 65532 字符，按数据实际大小由 CPU 分配存储空间

text 大文本类型，字符数大于 65535 存储时使用，排序和比较时，对 TEXT 不区分大小写

blob：大文本类型，字符数大于 65535 存储时使用，排序和比较时，对 BLOB 值区分大小写

日期时间类型：

DATETIME：日期+时间，8 个字节，范围：1000-01-01 00:00:00.000000～9999-12-31 23:59:59.999999

TIMESTAMP：日期+时间，4 个字节，范围：1970-01-01 00:00:00.000000～2038-01-19 03:14:07.999999

DATE：日期，4 个字节，范围：0001-01-01～9999-12-31

YEAR：年份，1 个字节，范围：1901～2155

TIME：时间，3 个字节，格式：HH:MM:SS

YEAR 年份的处理：默认用 4 位数字表示，当只用 2 位数字赋值时，1～69 视为 2001～2069，70～99 视为 1970～1999，0 视为 0000

日期时间未赋值处理：TIMESTAMP 字段赋值为当前系统时间，DATETIME 字段赋值为 NULL

时间函数：使用 SELECT 指令可直接调用,输出函数结果

now()	获取系统当前日期和时间
year()	执行时动态获得系统日期时间
sleep(N)	休眠 N 秒
curdate()	获取当前的系统日期
curtime()	获取当前的系统时刻
month()	获取指定时间中的月份
date()	获取指定时间中的日期
time()	获取指定时间中的时刻

案例：创建个人信息表：

```
mysql> create table t4(name char(15), age tinyint(2) unsigned, birth date ,time time, year year(4), party timestamp);
mysql> insert into t4 value ("jack",20,19900101,083000,1990,20180623203028);
mysql> insert into t4 value ("tom",5,19950601,time(now()),year(now()),now());
```

枚举类型

ENUM：从给定值集合中选择单个值，格式：enum(值 1, 值 2, ... 值 N)

SET：从给定值集合中选择一个或多个值，格式：set(值 1, 值 2, ... 值 N)

约束条件：

NOT NULL	不允许为空。null 表示空，"null"表示字段
Key	索引类型，键值
Default	设置默认值,缺省为 NULL

案例：创建一个班级信息表：

```
mysql> create table t7(name char(15) not null , age tinyint(2) unsigned default 18, class char(7) default "nsd1802", sex enum("男","女"))default charset utf8;
mysql> insert into t7(name,sex) values ("克斯","男");
mysql> insert into t7(name) values ("如来");
mysql> insert into t7(name,sex) values ("","女");
mysql> insert into t7(name,age) values ("马三立",null);
mysql> select * from t7;
+----------+-----+---------+-----+
| name     | age | class   | sex |
+----------+-----+---------+-----+
| 克斯     |  18 | nsd1802 | 男  |
```

```
| 如来      |   18 | nsd1802 | NULL |
|           |   18 | nsd1802 | 女   |
| 马三立    | NULL | nsd1802 | NULL |
+----------+------+---------+------+
```

修改表结构：

语法结构:ALTER TABLE 表名 执行动作

执行动作：

Add 添加字段

Modify 修改字段类型

Change 修改字段名

Drop 删除字段

Rename 修改表名

添加新字段：

ALTER TABLE 表名 ADD 字段名 类型(宽度) 约束条件 指定位置;

指定位置：after 字段名 ，first

删除字段：同时删除字段下所有数据

ALTER TABLE 表名 drop 字段名;

修改字段名：类型和约束条件可原样复制，也可设置新的

ALTER TABLE 表名 change 源字段名 新字段名 类型(宽度) 约束条件;

修改字段类型：不能与已存储的数据类型冲突

ALTER TABLE 表名 modify 字段名 类型(宽度) 约束条件 指定位置;

修改表名：同时修改/var/lib/mysql 目录下的文件名

ALTER TABLE 表名 Rename 新表名;

案例：修改已有的表：

```
mysql> alter table t2 rename test2;
mysql> show tables;
[root@host50 test]# ls
mysql> alter table test2 add QQ varchar(11) default "10000" after 地址;
mysql> alter table test2 add email varchar(11) first ,add sex enum("b","g") ;
mysql> alter table test2 change 电话 手机号 bigint(20);
mysql> alter table test2 change 姓名 名字 char(10)  not null;
mysql> alter table test2 drop email;
mysql> alter table test2 modify 爱好 set ("eat","buy","game","sleep") default "eat,sleep";
```

SQL 语句主要分类：

DDL(Data Definition Language):数据定义语言，定义对数据库对象(库、表、列、索引)的操作。

CREATE、DROP、ALTER、RENAME、TRUNCATE

DML(Data Manipulation Language): 数据操作语言，定义对数据库记录的操作。

INSERT、DELETE、UPDATE、SELECT 等

DCL(Data Control Language): 数据控制语言，定义对数据库、表、字段、用户的访问权限和安全级别。

GRANT、REVOKE

Transaction Control:事务控制

COMMIT、ROLLBACK、SAVEPOINT

delete、truncate、drop 三种删除的区别：

delete：属于 DML 语句，删除数据，保留表结构，需要提交事务，可以回滚，如果数据量大，很慢。

truncate：属于 DDL 语句，删除所有数据，保留表结构，自动提交事务，不可以回滚，一次全部删除所有数据，速度相对较快。

Drop：属于 DDL 语句，删除数据和表结构,不需要提交事务，删除速度最快。

索引：对记录集的多个字段进行排序的方法

类型：Btree(默认) 、 B+tree 、 hash

B 树：索引和实际数据是分开的，非聚集索引

B+树：数据结构中存储的都是实际的数据，聚集索引

优点：通过创建唯一性索引,可以保证数据库表中每一行数据的唯一性，可以加快数据的检索速度

缺点：当对表中的数据进行增加、删除和修改的时候,索引也要动态的维护,降低了数据的维护速度。索引占用物理空间

注意：

避免索引过多，会影响写性能

给筛选效果低的字段加索引，几乎无效，如性别、状态标志等

每条查询执行时，只会使用一个索引，有需要时应该创建复合索引

复合索引使用时遵守“从左到右”原则，严禁左百分号

不要在索引字段上执行运算操作和使用函数

不建议使用索引的情况：

表记录太少时

经常插入、删除、修改的表

数据重复且分布平均的表字段

键值类型：

INDEX	普通索引
UNIQUE	唯一索引
FULLTEXT	全文索引
PRIMARY KEY	主键
FOREIGN KEY	外键

普通索引：

注意：

一个表中可以有多个 INDEX 字段，通常都是做查询条件的字段

字段的值允许有重复,且可以赋 NULL 值

INDEX 字段的 KEY 标志是 MUL

属于非聚簇索引，索引信息和数据分开存储。找到索引后，需要回行到数据中定位，返回数据

建表的时候指定索引字段：create table 表名（字段 1,字段 2,...,INDEX(字段 1),INDEX(字段 2),...);

在已有的表中设置 INDEX 字段：CREATE INDEX 索引名 ON 表名(字段名);

删除指定表的索引字段：DROP INDEX 索引名 ON 表名 ;

查看指定表的索引信息：show index from 表名\G;

```
mysql> create index 姓名 on test2(姓名);
mysql> show index from test2\G;
mysql> drop index 号码 on test2;
mysql> create table t8 (id tinyint(1),name char(10),phone
char(11),index(id),index(name));
```

主键(primary key)：

注意：

一个表中只能有一个主键字段。如果有多个字段都作为主键,称为复合主键,必须一起创建，一起删除。

对应的字段值不允许有重复,且不允许赋 NULL 值

主键字段的 KEY 标志是 PRI

通常与 AUTO_INCREMENT 连用，AUTO_INCREMENT 的作用是让字段的值自增长

通常把表中能够唯一标识记录的字段设置为主键字段

属于聚簇索引，索引信息下存储数据。找到索引后，直接返回数据，由于加载入内存时，索引带了数据，所以会消耗更多的内存

主键与唯一索引的区别：

主键的要求比唯一索引更严格：唯一索引可以为空，主键不能为空

建表的时候指定主键字段：

create table 表名（字段 1,字段 2,...,PRIMARY KEY(字段名));

create table 表名（字段 1 primary key,字段 2,...);

在已有的表中设置主键字段：该字段必须符合主键的要求，否则会报错

ALTER TABLE 表名 ADD PRIMARY KEY(字段名);

移除表中的主键字段：

ALTER TABLE 表名 DROP PRIMARY KEY;

复合主键：作为主键的多个字段的值不允许同时相同

create table 表名（字段 1,字段 2,...,PRIMARY KEY(字段 1,字段 2));

```
mysql> create table t9(id int(2) primary key,  name char(15), age tinyint(2));
mysql> create table t10(id int(2), name char(15), age tinyint(2),primary
key(id));
mysql> alter table t9 drop primary key;
mysql> alter table t9 add primary key(id);
mysql> alter table 女友 add primary key(姓名,生日);
mysql> alter table 女友 drop primary key;
mysql> alter table 女友 add id tinyint(1) primary key auto_increment first;
mysql> create table t11(ip char(15), port smallint, status enum("deny","allow"),
primary key(ip,port));
mysql> create table t12 (id int(1) auto_increment primary key, name char(15),
sex enum("man","woman"), age tinyint(1));
mysql> insert into t12(name,sex,age) value ("jack","man",15);
mysql> insert into t12 value (3,"tom","man",16);
mysql> insert into t12 value (null,"gre","man",19);
mysql> select * from t12;
+----+-----+------+-----+
| id | name | sex   | age  |
```

```
+----+------+-------+------+
|  1 | jack | man   |   15 |
|  3 | tom  | man   |   16 |
|  4 | gre  | man   |   19 |
+----+------+-------+------+
```

外键(foreign key)：让当前表字段的值从指定表中的指定字段的值中选择

使用外键的条件：

表的存储引擎必须是 innodb

两张表的字段类型必须一致

被参照字段必须是索引类型的一种，通常是主键

创建外键：

必须先创建被参考表：create table 表名(字段..)engine=innodb;

create table 表名(...,FOREIGN KEY(表 A 的字段名) References 表 B(被参考字段名) ON UPDATE CASCADE ON DELETE CASCADE)engine=innodb;

删除外键：

ALTER TABLE 表名 DROP FOREIGN KEY 约束名 ;

注意：约束名需要查询创表信息中的 CONSTRAINT 字段获得

```
mysql> create table jfb(id int(2) primary key auto_increment, name char(15), class char(7), pay enum("yes","no"))engine=innodb;
mysql> create table bjb(id int(2),sex enum("boy","girl"), foreign key(id) references jfb(id) on update cascade on delete cascade)engine=innodb;
mysql> show create table bjb;
mysql> alter table bjb drop foreign key bjb_ibfk_1;
mysql> alter table bjb add foreign key(id) references jfb(id) on update cascade on delete cascade;
```

存储引擎：

可通过 show engines;查询，不同的引擎有不同的功能和数据存储方式。5.5 以后的版本默认是 innodb

注意：Engine 列表示引擎名；Support 列值为 DEFAULT 表示默认引擎，YES 表示支持，NO 表示不支持

建表时指定存储引擎：create table 表名()engine=引擎名;

修改存储引擎：alter table 表名 engine=引擎名;

注意：在有数据存储后，再修改存储引擎有可能对数据造成影响

设置默认存储引擎：修改主配置文件

```
[root@host50 ~]# vim /etc/my.cnf
default-storage-engine=引擎名
```

示例：

```
mysql> show engines;
mysql> create table t15 (haha int);
mysql> alter table t15 engine=myisam;
mysql> alter table t15 engine=memory;
[root@host50 ~]# vim /etc/my.cnf
    default-storage-engine=myisam
```

存储引擎特点：

Myisam：支持表级锁。

对应 3 个存储文件：

.frm　表结构

.MYD　数据

.MYI　索引

InnoDB：支持行级锁。支持事务、事务回滚、外键

对应 2 个存储文件：

.frm　表结构

.ibd　数据和索引

名词解释：

事务：对数据库服务的访问过程，包括：连接数据库服务器、操作数据、断开连接。是最小的逻辑工作单元

事务回滚：在事务执行过程中，任一步操作失败，都会恢复之前的所有操作

事务特性 (ACID)：

原子性(Atomic)：事务的整个操作是一个整体,不可分割,要么全部成功,要么全部失败，不可能停滞在中间环节。事务执行过程中出错，会回滚到事务开始前的状态。

一致性(Consistency)：事务操作过程中,表中的记录不会改变。

隔离性(Isolation)：事务操作是相互隔离不受影响的。同一时间，只允许一个事务请求同一数据，不同的事务之间彼此没有任何干扰。

持久性(Durability)：数据一旦提交,不可改变,永久改变表数据,不能回滚。

查看事务自动提交状态：mysql> show variables like "autocommit";

关闭自动提交：mysql> set autocommit=off;

注意：关闭自动提交为临时操作，仅对当前终端的当前数据库进程有效，退出数据库，切换终端，重起数据库服务都会失效

数据回滚:mysql> rollback;

提交事务：mysql> commit;

支持事务的表在主目录下有对应的事务日志文件记录执行过程中的操作，用于回滚：ib_logfile0,ib_logfile1,ibdata1

脏读：事务 A 读取了事务 B 更新的数据，然后 B 回滚操作，那么 A 读取到的数据是脏数据

不可重复读：事务 A 多次读取同一数据，事务 B 在事务 A 多次读取的过程中，对数据作了修改，导致事务 A 多次读取同一数据时，结果不一致。

幻读：事务 A 读取与搜索条件相匹配的若干行,事务 B 以插入或删除行等方式来修改事务 A 的结果集，导致事务 A 返回了以前不存在的记录。

锁：解决并发访问冲突问题

读锁（共享锁）：select

写锁（排他锁、互斥锁）:insert,update,delete

行级锁：只给当前被访问中的行加锁,可以解决不可重复读。适用于写操作多的表，增大并发量

表级锁：只要该表中的某一行被访问，则对整张表加锁,可以解决幻读。适用于查询操作多的表，节约锁操作的系统资源

悲观锁：总是假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。

乐观锁：总是假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

不能解决脏读问题

inndob 事务的隔离级别:	脏独	不可重复读	幻读
读未提交（read-uncommitted）	是	是	是
不可重复读（read-committed）	否	是	是
可重复读（repeatable-read）	否	否	是
串行化（serializable）	否	否	否

事务日志的执行过程:

事务在修改时，要先记 undo，在记 undo 之前要记 undo 的 redo，然后修改数据，再记数据页修改的 redo

Redo 一定要比数据页先持久化到磁盘。当事务需要回滚时，因为有 undo，可以把数据页回滚到前镜像的状态

崩溃恢复时，如果 redolog 中事务没有对应的 commit 记录，那么需要用 undo 把该事务的修改回滚到事务开始之前

如果有 commit 记录，就用 redo 前滚到该事务完成时并提交掉

设置搜索路径：

查看路径：mysql> show variables like "secure_file_priv";

默认路径：/var/lib/mysql-files/

修改路径：在主配置文件中添加一行

```
[root@host50 ~]# vim /etc/my.cnf
    secure_file_priv="/mydata"      #mysql 必须对该目录有 7 权限
```

注意：若不在原目录下创建，注意 SELinux 标签 mysqld_db_t。也可以关闭 SELinux

数据导入：把系统文件的内容存储到数据库服务器的表里

语法：load data infile "目录名/文件名" into table 表名 fields terminated by "分隔符" lines terminated by "换行符";

注意：文件路径使用绝对路径；字段分隔符要与文件内的一致，默认为空格和制表符，换行符默认为"\n"；导入数据的表字段类型要与文件字段匹配

案例：导入 passwd

```
[root@host50 ~]# mkdir /mydata
[root@host50 ~]# chown mysql /mydata
[root@host50 ~]# cp /etc/passwd /mydata/
mysql> create table user1 (name char(30),password char(1),uid smallint(2),gid smallint(2),comment char(100),homedir char(100),shell char(50),index(name));
mysql> alter table user1 modify uid smallint(2) unsigned;
mysql> alter table user1 modify gid smallint(2) unsigned;
mysql> load data infile "/mydata/passwd" into table db1.user1 fields terminated by ":" lines terminated by "\n";
mysql> alter table user1 add id int(2) primary key auto_increment first;
mysql> select * from user1;
```

数据导出：把表里的内容存储到系统文件

语法：select 查询语句 into outfile "目录名/文件名" fields terminated by "分隔符" lines terminated by "换行符";

注意：导出的内容由 SQL 查询语句决定，文件路径使用绝对路径，分隔符默认为制表符，换行符默认为"\n"

案例：导出到 user

```
mysql> select * from user1 into outfile "/mydata/user1.txt";
mysql> select * from user1 into outfile "/mydata/user2.txt" fields terminated by ":";
mysql> select name,password,uid,gid,comment,homedir,shell from user1 into outfile "/mydata/user" fields terminated by ":";
```

基本查询条件：适用于查询、更新、删除

数值比较：

字段类型必须是数值类型

用法：where 字段名 符号 数字

符号：

=	等于
>	大于
>=	大于或等于
<	小于
<=	小于或等于
!=	不等于

示例：

```
mysql> select * from user1 where id<10;
mysql> update user1 set password="8",homedir="/home" where id=10;
mysql> delete from user1 where uid>3000;
mysql> select id,name from user1 where uid=gid;
```

字符比较：

用法：where 字段名 符号

符号：

="字符串"	等于某字符串
!="字符串"	不等于某字符串
IS NULL	匹配空
IS NOT NULL	匹配非空

示例：

```
mysql> select name from user1 where shell="/bin/bash";
mysql> select name from user1 where shell!="/bin/bash";
mysql> update user1 set uid=null where id=2;
mysql> select name from user1 where uid is null;
mysql> update user1 set name="" where name="ftp";       #删除了ftp，但是非空
mysql> select name from user1 where name is not null;
```

逻辑比较：

用法：多个判断条件时使用

符号：

OR	逻辑或
AND	逻辑与，优先级高于 OR
!或 NOT	逻辑非
( )	提高优先级

示例：

```
mysql> select id,name from user1 where name="lisi" or id=1 and
name="root";
+----+------+
| id | name |
+----+------+
| 40 | lisi |
|  1 | root |
+----+------+
mysql> select id,name from user1 where (name="lisi" or id=1) and
name="root";
+----+------+
| id | name |
+----+------+
|  1 | root |
+----+------+
```

范围内匹配：

用法：匹配范围内的任意一个值即可

符号：

In（值列表）	在值列表里
Not in（值列表）	不在值列表里
Between 数字 1 and 数字 2	在数字 1 到数字 2 之间

示例：

```
mysql> select id,name from user1 where id between 10 and 25;
mysql> select id,name from user1 where id in (1,3,5,8,15,65);
mysql> select id,name from user1 where shell not in
("/bin/bash","/sbin/nologin");
```

去重显示：

用法：

```
select distinct 字段名  from ...
```

示例：

```
mysql> select distinct shell from user1;
mysql> select distinct shell from user1 where id>10;
```

高级查询条件：

模糊匹配：like

用法：WHERE 字段名 LIKE "通配字串"

通配符：

_	匹配单个字符
%	匹配 0~N 个字符

示例：

```
mysql> select * from user1 where name like '__';
mysql> select * from user1 where name like 'r%t';
```

正则匹配：

用法：WHERE 字段名 REGEXP "正则表达式"

示例：

```
mysql> select * from user1 where name regexp '^r.*t$';
```

四则运算：

字段类型必须是数值类型

适用于 select,upate

符号：

+	加法
-	减法
*	乘法
/	除法
%	取余数(求模)

示例：

```
mysql> update user1 set uid=uid+1 where uid between 10 and 20;
mysql> select name,year(now())-uid age from user1 where name="lisi";
    +------+------+
    | name | age  |
    +------+------+
    | lisi | 1018 |
    +------+------+
mysql> select name,(uid+gid)/2 ave from user1 where id<3;
    +------+--------+Delete、truncate、drop 的区别：
    delete：属于 DML 语句，删除数据，保留表结构，需要 commit，可以回滚，如果数据量大，很慢。
    truncate：属于 DDL 语句，删除所有数据，保留表结构，自动 commit，不可以回滚，一次全部删除所有数据，速度相对较快。
    Drop：属于 DDL 语句，删除数据和表结构,不需要 commit，删除速度最快。
    | name | ave    |
    +------+--------+
    | root | 0.0000 |
    | bin  |   NULL |
    +------+--------+
```

聚集函数：

字段类型必须是数值类型

适用于 select

内置数据统计函数：

avg(字段名)	求平均值
sum(字段名)	求和
min(字段名)	统计最小值
max(字段名)	统计最大值
count(字段名)	统计个数

示例：

```
mysql> select avg(uid) from user1;
mysql> select sum(uid) from user1 where name like '%a%';
mysql> select count(name) from user1 where shell="/bin/bash";
mysql> select min(uid) from user1;
```

```
mysql> select max(gid) from user1;
```

操作查询结果：

查询结果排序：

用法：ORDER BY 字段名 asc/desc

注意：默认为 asc 升序，若需要降序，则需要加 desc

示例：

```
mysql> select id,name from user1 where uid>30 order by id;
mysql> select id,name from user1 where id<20 order by name desc;
```

查询结果分组：

用法：group by 字段名

注意：select 的字段名必须和 group by 的字段名一致，且只能选择一个字段分组

查询结果过滤：

用法：HAVING 条件表达式

示例：

```
mysql> select shell from user1 group by shell having shell="/bin/bash";
mysql> select name,shell from user1 where id>30 having name in ("lisi","mysql");
```

限制查询结果显示行数：

用法：

```
LIMIT N      显示查询结果前 N 条记录
LIMIT N,M    显示从第 N 行开始 M 行的查询记录，行号从 0 开始
```

示例：

```
mysql> select id,name from user1 where uid<20 limit 4;
mysql> select id,name from user1 where uid<20 limit 0,9;
mysql> select id,name from user1 order by uid desc limit 5;
mysql> (select id,name from user1 order by uid desc limit 5) order by id;
```

复制表：

将源表复制为新表：键值不会被复制，该功能也可以作为备份

```
create table 新表 select * from 源表
```

将指定的查询结果复制为新表：

```
create table 新表 select 查询语句;
```

复制源表的结构到新表：

```
create table 新表 select * from 源表 where false;
```

示例：

```
mysql> create table user4 select * from user1;
mysql> create table t1 select name,uid,homedir from user1 limit 3;
mysql> create table test.test1 select * from db1.user1;
```

多表查询：连接查询

作用；

将 2 个或 2 个以上的表按某个条件连接起来,从中选取需要的数据

当多个表中存在相同意义的字段(字段名可以不同)时,可以通过该字段连接多个表

格式：

select 字段名列表 from 表a,表b;　　#查询结果为笛卡尔积集，显示查询结果的总条目数=表a的条目数*表b的条目数

select 字段名列表 from 表a,表b where 条件;

示例：

```
mysql> select * from t1,t2;
mysql> select * from t1,t2 where t1.uid=t2.uid;
mysql> select t1.name,t2.name from t1,t2 where t1.uid=t2.uid;
mysql> select t1.name,t2.name from t1,t2 where t1.uid=t2.uid and t1.name=t2.name;
```

嵌套查询：where 子查询，把内层查询结果作为外层查询的查询条件

格式：

select 字段名列表 from 表 A where 条件 (select 字段名列表 from 表A);

select 字段名列表 from 表 A where 条件 (select 字段名列表 from 表A where 条件);

示例：

```
mysql> select name from user1 where name in (select name from t1 where shell="/bin/bash");
mysql> select name from test1 where name in (select name from db1.t1);
mysql> select name from test1 where name not in (select name from db1.t1);
```

左连接查询：

条件成立时，以左表为主显示相同的记录。左表存在，右表不存在时，右表用空值匹配

用法：select 字段名列表 from 左表 LEFT JOIN 右表 ON 条件表达式;

右连接查询：

条件成立时，以右表为主显示相同的记录。右表存在，左表不存在时，左表用空值匹配

用法：select 字段名列表 from 左表 RIGHT JOIN 右表 ON 条件表达式;

示例：

```
mysql> select * from t1 left join t2 on t1.uid=t2.uid;
+--------+------+---------+--------+------+--------------+
| name   | uid  | homedir | name   | uid  | shell        |
+--------+------+---------+--------+------+--------------+
| root   |    0 | /root   | root   |    0 | /bin/bash    |
| daemon |    2 | /sbin   | daemon |    2 | /sbin/nologin |
+--------+------+---------+--------+------+--------------+
mysql> select t1.* from t1 left join t2 on t1.uid=t2.uid and t1.name=t2.name;
+--------+------+---------+
| name   | uid  | homedir |
+--------+------+---------+
| root   |    0 | /root   |
| daemon |    2 | /sbin   |
| NULL   | NULL | NULL    |
| NULL   | NULL | NULL    |
+--------+------+---------+
mysql> select t1.name,t2.name from t1 right join t2 on t1.uid=t2.uid and
```

```
t1.name=t2.name;
+--------+------+---------+--------+------+--------------+
| name   | uid  | homedir | name   | uid  | shell        |
+--------+------+---------+--------+------+--------------+
| root   |    0 | /root   | root   |    0 | /bin/bash    |
| daemon |    2 | /sbin   | daemon |    2 | /sbin/nologin |
+--------+------+---------+--------+------+--------------+
```

查询方式总结：

分组 order by
排序 group by
限制条目数 limit
查询结果过滤 having
单表查询 select ... from ... where ...
嵌套查询 select ... from ... Where...(select ...)
多表查询 select ... from 表1,...表n where ...
左连接查询 left ... join ... on
右连接查询 right ... join ... on

常见的 MySQL 管理工具：

mysql 命令行，跨平台，MySQL 官方 bundle 包自带
MySQL-Workbench 图形，跨平台，MySQL 官方提供
MySQL-Front 图形，Windows，开源,轻量级客户端软件
phpMyAdmin 浏览器，跨平台，开源,需 LAMP 平台
Navicat 图形，Windows，专业、功能强大,商业版

PhpMyAdmin：

1.安装 httpd、php、php-mysql 及相关包
2.启动 httpd 服务程序
3.解压 phpMyAdmin 包,复制到网站目录下
4.复制主配置文件：cp config.sample.inc.php config.inc.php
5.配置 config.inc.php，指定 mysql 主机位置

```
[root@host50 phpadmin]# vim config.inc.php
17:$cfg['blowfish_secret'] = '1';                    #指定绝密短语
31:$cfg['Servers'][$i]['host'] = 'localhost';    #指定数据库所在主机
```

6.在数据库中创建授权用户，网页上使用该用户登陆使用

授权：在数据库服务器添加可以连接的用户，默认只有数据库管理员 root 在本机才能登陆

用法：GRANT 权限列表 ON 库名.表名 TO 用户名@"客户端地址" IDENTIFIED BY "密码" (WITH GRANT OPTION);

注意：

加上 WITH GRANT OPTION 表示对该表有授权权限，同时要授予对 mysql 库的插入权限(用于 grant 和 revoke)

当库名.表名为*.*时,匹配所有库的所有表

授权设置放在 mysql 库的 user 表中

权限列表：

all：授予所有权限，包括建库权限，所以该库可以不用事先创建

Select,insert,...：授予插入、查询权限

Select,insert,update(字段 1,字段 2...)：授予插入、查询权限，指定字段的修改权限

Usage：无权限

客户端地址：

Localhost：本机

% :匹配所有主机

192.168.1.% :匹配指定的一个网段

192.168.1.1 :匹配指定 IP 地址的单个主机

%.tarena.com :匹配一个 DNS 区域（了解）

svr1.tarena.com :匹配指定域名的单个主机（了解）

示例：

```
mysql> grant all on *.* to root@"192.168.4.%" identified by "123456" with grant option;
mysql> grant all on test.* to test@"%" identified by "123456" with grant option;
mysql> grant insert on mysql.* to test@"%";
mysql> grant select,insert,update on test.* to test@"%" identified by "123456";
mysql> grant select,insert,update(name,uid) on test.test1 to lisi@"192.168.4.%" identified by "123456";
mysql> grant usage on *.* to sb@"192.168.4.%" identified by "123456";
```

授权库 mysql 的表信息：

User 表：存储授权用户的访问权限

Db 表：存储授权用户对数据库的访问权限

tables_priv 表：存储授权用户对表的访问权限

columns_priv 表：存储授权用户对字段的访问权限

查看授权：

查看当前用户：select user();

查看数据库所在主机名：select @@hostname;

查看当前用户授权表：show grants;

Root 查看其他用户授权：

```
select host,user from mysql.user;
show grants for 用户名@"客户端地址";
```

Root 查看库授权：

```
select host,db,user from mysql.db;
select * from mysql.db where db="test"\G;
select * from mysql.db where user="test"\G;
```

root 查看表的授权:

```
select * from mysql.tables_priv;
select * from mysql.tables_priv where Table_name="test1"\G;
select * from mysql.tables_priv where user="lisi"\G;
```

root 查看字段授权：

select * from mysql.columns_priv;

授权目录：

命令	权限
usage	登录权限
SELECT	查询表记录
INSERT	插入表记录
UPDATE	更新表记录
DELETE	删除表记录
CREATE	创建库、表
DROP	删除库、表
RELOAD	重新载入授权，用于执行 flush {tables | logs | privileges}
SHUTDOWN	允许关闭 mysql 服务，使用 mysqladmin shutdown 来关闭 mysql
PROCESS	允许查看用户登录数据库服务器的进程(show processlist;)
FILE	导入、导出数据
REFERENCES	创建外键
INDEX	创建索引
ALTER	修改表结构
SHOW DATABASES	查看库
SUPER	关闭属于任何用户的线程
CREATE TEMPORARY TABLES	允许在 create table 语句中使用 TEMPORARY 关键字
LOCK TABLES	允许使用 LOCK TABLES 语句
EXECUTE	执行存在的 Functions,Procedures
REPLICATION SLAVE	从主服务器读取二进制日志
REPLICATION CLIENT	允许在主/从数据库服务器上使用 show status 命令
CREATE VIEW	创建视图
SHOW VIEW	查看视图
CREATE ROUTINE	创建存储过程
ALTER ROUTINE	修改存储过程
CREATE USER	创建用户
EVENT	有操作事件的权限
TRIGGER	有操作触发器的权限
CREATE TABLESPACE	有创建表空间的权限

撤销授权：

用法：revoke 权限列表 on 库名.表名 from 用户名@"客户端地址";

注意：

库名.表名必须与授权时对应

可单独撤销一部分授权

权限列表为 all 时，可一次性清空所有授权，无论之前是否授权

示例：

```
mysql> revoke grant option on *.* from root@"192.168.4.%";
mysql> revoke select,insert,update on test.* from test@"%";
mysql> revoke select,insert,update(uid,name) on test.test1 from lisi@"%";
mysql> revoke all on db1.user1 from lisi@"%";
mysql> revoke insert on *.* from root@"192.168.4.%";
```

mysql> revoke all on *.* from root@"192.168.4.%";

修改已授权用户的键值也可以修改权限：改后需要刷新

mysql> update mysql.user set Select_priv="y" where user="root" and host="192.168.4.%";

mysql> update mysql.tables_priv set Table_priv="select,update" where user="lisi";

mysql> flush privileges;

删除授权用户：自动删除该用户下所有授权

用法：drop user 用户名@"客户端地址";

示例：mysql> drop user test@"%";

重设用户密码：

授权用户修改自己的密码：SET PASSWORD=PASSWORD("新密码");

管理员重置授权用户的密码：SET PASSWORD FOR 用户名@"客户端地址"=PASSWORD("新密码");

示例:

mysql> set password=password("654321");

mysql> set password for root@localhost=password("123456");

将一个用户的权限作为模板授权给其他用户:

GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION

查询授权模板功能是否开启:mysql> show variables like '%proxy%';

临时配置:

mysql> set global check_proxy_users=on;

mysql> set global mysql_native_password_proxy_users=on;

永久配置:

vim /etc/my.cnf

[mysqld]

check_proxy_users=on

mysql_native_password_proxy_users=on

创建授权用户:mysql> grant all on game.* to mysqladmin identified by '123456';

以 mysqladmin 用户为模板创建用户:

一步创建:mysql> grant proxy on mysqladmin to will identified by '123456';

分步创建:

创建用户:mysql> create user tom identified by '123456';

按模板授权:mysql> grant proxy on mysqladmin to tom;

数据备份：

数据备份方式：

物理备份：只适用于 Myisam 引擎。使用 cp,tar 等命令

逻辑备份：备份时，根据已有的库表记录，生成对应的 sql 命令，保存命令到指定的文件。恢复时，执行这些命令还原数据

示例：

[root@host50 ~]# cp -r /var/lib/mysql/mysql ~/mysql.bak

[root@host50 ~]# tar -czf mysql.bak.gz mysql.bak/

[root@host50 ~]# scp -r mysql.bak 192.168.4.51:/var/lib/mysql/mysql

[root@host51 ~]# chown -R mysql: /var/lib/mysql/mysql

```
[root@host51 ~]# systemctl restart mysqld
```

数据备份策略：

完全备份：备份所有数据：一台服务器上的所有库、一个库的所有表、一张表的所有记录。使用 mysqldump 命令

差异备份：备份自上一次完全备份之后有变化的数据

增量备份：备份自上一次备份之后有变化的数据

数据备份考虑因素：

时间：数据库不繁忙的时候

备份策略：完全备份+差异备份/增量备份

备份频率：例如周二完全备份，其余 6 天差异备份/增量备份

存储空间：采用逻辑卷模式，备份文件数据库本机一份，独立存储设备一份

备份名：有标示的文件名

完全备份+差异备份：

优点：恢复时，只需要执行完全备份和最新的差异备份即可，丢失非最新备份都不影响数据恢复

缺点：重复备份

完全备份+增量备份：

优点：只备份新增的数据，不会重复备份

缺点：丢失任一备份都会影响数据恢复

生产环境下执行备份数据的手段：计划任务+备份脚本，部署 mysql 主从同步结构实现数据自动备份

数据完全备份：mysqldump

用法：mysqldump -u 用户名 -p 源数据库名 > 目录名/文件名.sql

注意：备份的用户必须对该库有备份权限。通常使用 root 用户备份

优点：可以跨版本备份恢复

缺点：耗时长，锁库

备份时记录偏移量：--master-data＝值

当值为 1 时，记录日志名和偏移量且不注释，默认值

当值为 2 时，记录日志名和偏移量但会注释

数据库名的表示方式：

所有库：-A 或--all-databases

一个库的所有表：数据库名

一张表的所有记录：库名 表名

备份多个库的所有表：-B 数据库名 数据库名

示例：

```
[root@host50 ~]# mysqldump -u root -p -A > all.sql
[root@host50 ~]# mysqldump -u root -p mysql user > mysql.user.sql
[root@host1 ~]# mysqldump -A --master-data > all.sql
```

数据恢复：mysql

用法：

```
mysql -u 用户名 -p 目标数据库名 < 目录名/文件名.sql
mysql> source 目录名/文件名.sql
```

注意：

由于备份文件中没有建库命令(所有库备份除外)，所以，若库被删除，需要先

建空库,再执行 mysql 导入

由于所有库的备份指定了库名，所以恢复所有库的备份时，可以不指定目标数据库名

使用所有库的备份可以恢复单个库或单个表:通过查看备份文件,找到该库、表的创表信息及写入信息,重新执行一遍

在数据库中执行 source 命令需要跟绝对路径，除所有库备份外都需要进入库目录下

示例：

```
mysql> source ~/all.sql
[root@host50 ~]# mysql -u root -p < all.sql
[root@host50 ~]# mysql -u root -p mysql < mysql.user.sql
```

设置自动备份：

```
[root@host50 ~]# cat bakdb.sh
#!/bin/bash
mysqldump -u root -p 123456 -A > ~/data/data-$(date +%F).sql
[root@host50 ~]# chmod +x bakdb.sh
[root@host50 ~]# crontab -e
0 4 * * 1 ~/bakdb.sh &> /dev/null
```

实时增量备份,恢复：

启用 mysql 服务的 binlog 日志文件实现实时增量备份

binlog 日志:mysql 服务日志的一种,记录用户连接后执行的除查询(select,show,desc)之外的 sql 命令

启用 binlog 日志：

server_id：自定义服务编号，范围：1～255

log-bin=目录/文件名：指定日志存放位置，默认为/var/lib/mysql/$HOSTNAME-bin，不写目录代表/var/lib/mysql，mysql 必须对该目录有写入权限

max_binlog_size：日志最大容量，默认 500m，超过后自动生成下一个日志文件

binlog_format：设置记录格式

```
[root@host50 ~]# vim /etc/my.cnf
server_id=50
log_bin=log/test
max_binlog_size=500m        #该行可省略
binlog_format=mixed
```

记录格式：

statement：每一条修改数据的 sql 命令都会记录在 binlog 日志中。

row：不记录 sql 语句上下文相关信息,仅保存哪条记录被修改。

mixed：以上两种格式的混合使用。

查看 binlog 日志：

```
[root@host50 ~]# mysqlbinlog /var/lib/mysql/log/test.000001
[root@host50 ~]# mysqlbinlog /var/lib/mysql/host50-bin.000001
```

日志相关文件：记录已有日志文件名

```
[root@host50 ~]# cat /var/lib/mysql/log/test.index
```

日志记录分段：

偏移量：pos

时间点：time

示例：

```
# at 2579            #起始偏移量
#180529 15:22:14 server id 50  end_log_pos 2644     #起始时间 服务编号 结束偏移量
```

执行日志文件记录的 sql 命令恢复数据：

思路：使用 mysqlbinlog 提取历史 SQL 操作，通过管道交给 mysql 命令执行

用法：mysqlbinlog 选项 文件 | mysql -u root -p123456

选项：

```
--start-position=起始偏移量
--stop-position=结束偏移量
--start-datetime="yyyy-mm-dd hh:mm:ss"
--stop-datetime="yyyy-mm-dd hh:mm:ss"
```

注意：结束位置需要包含 commit，表示命令执行

示例：

```
[root@host50 ~]# mysqlbinlog --start-position=4 --stop-position=2965 /var/lib/mysql/log/test.000001 | mysql -u root -p
[root@host50 ~]# mysqlbinlog --start-datetime="2018-05-29 14:50:58" --stop-datetime="2018-05-29 15:24:34" /var/lib/mysql/log/test.000001 | mysql -u root -p
```

手动新建日志：

1.重起 mysqld 服务

2.mysqldump 备份时加--flush-logs 选项

3.mysql> flush logs;

4.[root@host50 ~]# mysql -u root -p -e "flush logs"

示例：

```
[root@host50 ~]# mysqldump -uroot -p123456 --flush-logs test > ~/data/test.sql #备份时新建日志
mysql> create table t2 (id int);
mysql> insert into t2 value (12);
mysql> flush logs;
            #将创建表和插入数据写入一张日志中，并新建日志
mysql> drop table t2;

[root@host50 ~]# mysqlbinlog /var/lib/mysql/log/test.000005 | mysql -u root -p #恢复删除的内容
```

清理 binlog 日志：

删除早于指定编号的 binlog 日志：purge master logs to "binlog 文件名";

删除所有 binlog 日志,重建新日志：reset master;

示例：

```
mysql> purge master logs to "test.000004";
mysql> reset master;
mysql> show master status;          #显示当前的 binlog 日志及偏移量
```

重做日志(binlog)和二进制日志(redo log)的区别:

涉及存储引擎不一样：

binlog 记录的是所有存储引擎的操作记录

redo log 只记录 innodb 存储引擎的日志

记录内容不一样：

binlog 记录的是关于一个事务的具体操作内容,为逻辑日志

redo log 记录的是每个页更改的物理情况

写的时间不一样：

binlog 文件仅在事务提交前进行提交，即只写磁盘一次

而在事务进行过程中，却不断有重做日志条目被写入到 redo log 中

XtraBackup：

特点：强大的在线热备份工具，备份过程中不锁库表,适合生产环境，由专业组织 Percona 提供

组件：xtrabackup，innobackupex

innobackupex 备份过程：

xtrabackup 在备份 InnoDB 相关数据时，是有 2 种线程的，1 种是 redo 拷贝线程，负责拷贝 redo 文件，1 种是 ibd 拷贝线程，负责拷贝 ibd 文件；redo 拷贝线程只有一个，在 ibd 拷贝线程之前启动，在 ibd 线程结束后结束。xtrabackup 进程开始执行后，先启动 redo 拷贝线程，从最新的 checkpoint 点开始顺序拷贝 redo 日志；然后再启动 ibd 数据拷贝线程，在 xtrabackup 拷贝 ibd 过程中，innobackupex 进程一直处于等待状态（等待文件被创建）。

xtrabackup 拷贝完成 idb 后，通知 innobackupex（通过创建文件），同时自己进入等待（redo 线程仍然继续拷贝）；

innobackupex 收到 xtrabackup 通知后,执行 FLUSH TABLES WITH READ LOCK(FTWRL)，取得一致性位点，然后开始备份非 InnoDB 文件（包括 frm、MYD、MYI、CSV、opt、par 等）。拷贝非 InnoDB 文件过程中，因为数据库处于全局只读状态，如果在业务的主库备份的话，要特别小心，非 InnoDB 表（主要是 MyISAM）比较多的话整库只读时间就会比较长，这个影响一定要评估到。

当 innobackupex 拷贝完所有非 InnoDB 表文件后，通知 xtrabackup(通过删文件)，同时自己进入等待（等待另一个文件被创建）；

xtrabackup 收到 innobackupex 备份完非 InnoDB 通知后，就停止 redo 拷贝线程，然后通知 innobackupexredolog 拷贝完成（通过创建文件）；

innobackupex 收到 redo 备份完成通知后，就开始解锁，执行 UNLOCK TABLES；

最后 innobackupex 和 xtrabackup 进程各自完成收尾工作，如资源的释放、写备份元数据信息等，innobackupex 等待 xtrabackup 子进程结束后退出。

在上面描述的文件拷贝，都是备份进程直接通过操作系统读取数据文件的，只在执行 SQL 命令时和数据库有交互，基本不影响数据库的运行，在备份非 InnoDB 时会有一段时间只读（如果没有 MyISAM 表的话，只读时间在几秒左右），在备份 InnoDB 数据文件时，对数据库完全没有影响，是真正的热备。

InnoDB 和非 InnoDB 文件的备份都是通过拷贝文件来做的，但是实现的方式不同，前者是以 page 为粒度做的(xtrabackup)，后者是 cp 或者 tar 命令(innobackupex)，xtrabackup 在读取每个 page 时会校验 checksum 值，保证数据块是一致的，而

innobackupex 在 cp MyISAM 文件时已经做了 flush（FTWRL），磁盘上的文件也是完整的，所以最终备份集里的数据文件都是写入完整的。

增量备份只能对 InnoDB 做增量，InnoDB 每个 page 有个 LSN 号，LSN 是全局递增的，page 被更改时会记录当前的 LSN 号，page 中的 LSN 越大，说明当前 page 越新（最近被更新）。每次备份会记录当前备份到的 LSN（xtrabackup_checkpoints 文件中），增量备份就是只拷贝 LSN 大于上次备份的 page，比上次备份小的跳过，每个 ibd 文件最终备份出来的是增量 delta 文件。

安装 XtraBackup：

```
[root@host50 ~]# yum -y install
percona-xtrabackup-24-2.4.7-1.el7.x86_64.rpm libev-4.15-1.el6.rf.x86_64.rpm
```

基本选项：

```
--host          主机名，本机可不输
--user          用户名
--port          端口号，默认 3306 端口可不输
--password      密码
--databases     数据库名，不指定则默认所有库
--no-timestamp  不用日期作为子目录名存放在备份目录下
--apply-log     准备还原(回滚日志)
--copy-back     恢复数据
--redo-only     日志合并
--incremental 目录名              增量备份存储路径,该目录需要为空或不存在
--incremental--basedir=目录名     增量备份时,指定上一次备份数据存储的目录名
--incremental-dir=目录名          增量备份时,指定增量备份数据存储的目录名
--export        导出表信息
databases 的数据库名：不指定该选项则备份所有库
    单个库：--databases="库名"
    多个库：--databases="库名 1  库名 2"
    单个表：--databases="库名.表名"
```

完全备份、恢复：

1.完全备份：只留下 db5 库，系统库必须备份

```
[root@host50 ~]# innobackupex --user root --password 123456
--databases="mysql performance_schema sys db5" /db5 --no-timestamp
```

2.模拟数据丢失：

```
[root@host50 ~]# systemctl stop mysqld
[root@host50 ~]# rm -rf /var/lib/mysql/*
```

3.回滚日志：

```
[root@host50 ~]# cat /db5/xtrabackup_checkpoints
    backup_type = full-backuped     #状态为完全备份
    from_lsn = 0                    #起始日志序列号
    to_lsn = 18547155               #终止日志序列号
[root@host50 ~]# innobackupex --user root --password 123456
--databases="mysql performance_schema sys db5" --apply-log /db5
[root@host50 ~]# cat /db5/xtrabackup_checkpoints
```

```
        backup_type = full-prepared      #状态为完全准备就绪
        from_lsn = 0
        to_lsn = 18547155
```

4.恢复数据：

```
[root@host50 ~]# innobackupex --user root --password 123456
--databases="mysql performance_schema sys db5" --copy-back /db5
[root@host50 ~]# chown -R mysql: /var/lib/mysql
```

5.重起服务：

```
[root@host50 ~]# systemctl start mysqld
```

增量备份、恢复：

1.完全备份

2.第一次增量备份：

```
[root@host50 ~]# innobackupex --user root --password 123456
--incremental /new1 --incremental-basedir=/db5 --no-timestamp
[root@host50 ~]# cat /new1/xtrabackup_checkpoints
    backup_type = incremental
    from_lsn = 18547155
    to_lsn = 18556025
```

3.第二次增量备份：

```
[root@host50 ~]# innobackupex --user root --password 123456
--incremental /new2 --incremental-basedir=/new1 --no-timestamp
[root@host50 ~]# cat /new2/xtrabackup_checkpoints
    backup_type = incremental
    from_lsn = 18556025
    to_lsn = 18556357
```

4.数据丢失：

```
[root@host50 ~]# systemctl stop mysqld
[root@host50 ~]# rm -rf /var/lib/mysql/*
```

5.回滚日志：

```
[root@host50 ~]# innobackupex --user root --password 123456
--apply-log /db5
[root@host50 ~]# innobackupex --user root --password 123456
--apply-log --redo-only /db5
[root@host50 ~]# innobackupex --user root --password 123456
--apply-log --redo-only /db5 --incremental-dir=/new1
[root@host50 ~]# innobackupex --user root --password 123456
--apply-log --redo-only /db5 --incremental-dir=/new2
[root@host50 ~]# cat /db5/xtrabackup_checkpoints
    backup_type = log-applied
    from_lsn = 0
    to_lsn = 18556357
```

6.恢复数据：

```
[root@host50 ~]# innobackupex --user root --password 123456
--copy-back /db5
```

```
[root@host50 ~]# chown -R mysql: /var/lib/mysql/
[root@host50 ~]# systemctl restart mysqld
[root@host50 ~]# rm -rf /new1 /new2
```

在完全备份文件中恢复单个表：

删除表空间：alter table 表名 discard tablespace;

导入表空间：alter table 表名 import tablespace;

1.完全备份单个库：

```
[root@host50 ~]# innobackupex --user root --password 123456
--databases="db5" /db5 --no-timestamp
```

2.清空表内容，或这删除表后重建：

```
mysql> delete from t2;
或
mysql> drop table t2;
mysql> create table t2(pay int);
```

3.删除表空间：

```
mysql> alter table t2 discard tablespace;
```

4.导出表信息，并复制：

```
[root@host50 ~]# innobackupex --user root --password 123456
--databases="db5" --apply-log --export /db5
[root@host50 ~]# cp /db5/db5/t2.* /var/lib/mysql/db5/          #不
替换.frm 文件
[root@host50 ~]# chown mysql: /var/lib/mysql/db5/t2.*
```

5.导入表空间:

```
mysql> alter table t2 import tablespace;
```

6.删除 t2.exp t2.cfg

mysql 主从同步：存储数据的一种结构模式

主库(Master)：被客户端访问的数据库服务器

从库(Slave)：连接主库服务器，并自动同步主库上的所有数据

MySQL 主从同步结构模式：

单向复制：主 --> 从

链式复制:主 --> 从 --> 从

双向复制（互为主从）:主 <--> 从

放射式复制:从 <-- 主 --> 从

从库的同步进程：

Slave_IO ： 复制 master 主机 binlog 日志文件里的 SQL 语句到本机的 relay-log 文件里。

Slave_SQL：执行本机 relay-log 文件里的 SQL 语句，重现 Master 的数据操作。

主库的配置步骤：

1.启用 binlog 日志

2.给从库同步数据的用户授权：

3.查看正在使用的 binlog 日志文件信息：

从库的配置步骤：

1.指定本机 server_id

2.root 指定主数据库服务器的信息

3.启动 slave 进程

配置主从结构：单向复制、双向复制、放射式复制都是这种结构

1.配置前，要保证主从的数据相同。

2.启用主库的 binlog 日志：

```
[root@host52 ~]# vim /etc/my.cnf
    server_id=52
    log_bin=master
    binlog_format="mixed"
```

3.主库授权：

```
mysql> grant replication slave on *.* to t1@"%" identified by "123456";
```

4.查看主库日志文件状态：

```
mysql> show master status;
+--------------+----------+--------------+------------------+-------------------+
| File         | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+--------------+----------+--------------+------------------+-------------------+
| master.000001 |      154 |              |                  |                   |
+--------------+----------+--------------+------------------+-------------------+
```

5.从库测试网络连通性及主库的授权用户：

```
[root@host53 ~]# mysql -h 192.168.4.52 -u t1 -p
```

6.从库指定本机 server_id：

```
[root@host53 ~]# vim /etc/my.cnf
    server_id=53
```

7.从库使用 root 指定主数据库服务器的信息：

```
master_host：主库 IP
master_user：连接主库用户
master_password：密码
master_log_file：当前 binlog 日志名
master_log_pos：当前偏移量
mysql> change master to
master_host="192.168.4.52",master_user="t1",master_password="123456",master_log_file="master.000001",master_log_pos=154;
```

8.从库启动 slave 进程：要更改 Master 信息时,应先停止该进程

```
mysql> start slave;
mysql> show slave status\G;
*************************** 1. row ***************************
       Slave_IO_State: Waiting for master to send event
          Master_Host: 192.168.4.52
          Master_User: t1
          Master_Port: 3306
```

```
              Connect_Retry: 60
            Master_Log_File: master.000001
        Read_Master_Log_Pos: 154
             Relay_Log_File: host53-relay-bin.000003
              Relay_Log_Pos: 317
      Relay_Master_Log_File: master.000001
           Slave_IO_Running: Yes          #IO 进程运行情况
          Slave_SQL_Running: Yes          #SQL 进程运行情况
                ......
            Last_IO_Errno: 0              #IO 错误数量
            Last_IO_Error:                #IO 错误描述
           Last_SQL_Errno: 0              #SQL 错误数量
           Last_SQL_Error:                #SQL 错误描述
```

查看主库用户登陆：可以看出，IO 信息是由主库主动发送给从库的，进程为 Binlog_Dump

```
mysql> show processlist;
+----+------+------------------+------+-------------+------+---------
-------------------------------------------------------+------------------+
| Id | User | Host             | db   | Command     | Time | State
| Info             |
+----+------+------------------+------+-------------+------+---------
-------------------------------------------------------+------------------+
|  3 | t1   | 192.168.4.54:50382 | NULL | Binlog Dump |  533 | Master has
sent all binlog to slave; waiting for more updates | NULL             |
|  4 | t1   | 192.168.4.53:53708 | NULL | Binlog Dump |  530 | Master has
sent all binlog to slave; waiting for more updates | NULL             |
|  5 | root | localhost          | NULL | Query       |    0 | starting
| show processlist |
+----+------+------------------+------+-------------+------+---------
-------------------------------------------------------+------------------+
```

从库文件：

连接主服务器信息：master.info

中继日志文件：host53-relay-bin.*

中继日志索引：host53-relay-bin.index

记录中继日志信息：relay-log.info

关闭同步：

临时关闭：mysql> stop slave;

永久关闭：

```
[root@host53 mysql]# rm -f host53-relay-bin.* master.info relay-log.info
[root@host53 mysql]# systemctl restart mysqld
```

主库配置选项：影响所有从库

binlog_do_db=库名	设置 Master 记日志的库
binlog_ignore_db=库名	设置 Master 不记日志的库

从库配置选项：只影响本机

log_slave_updates	记录从库更新,允许链式复制（级联复制），适用于主

从从结构

relay_log=文件名　　指定中继日志文件名

replicate_do_db=库名,库名　　指定允许复制的库，其余不复制

replicate_ignore_db=库名　　指定不允许复制的库，其余库允许

注意：无论是主库还是从库，ignore-db 与 do-db 只能选一种，都不选时允许所有库

配置主从从结构：

1.第一对主从结构不变

2.设置第二对主从的主库：

```
[root@host53 ~]# vim /etc/my.cnf
    log_bin
    binlog_format="mixed"
    log_slave_updates
```

3.参照第一对主从结构设置

复制模式：

异步复制(Asynchronous replication)：主库在执行完客户端提交的事务后会立即将结果返给客户端,并不关心从库是否已经接收并处理。

全同步复制(Fully synchronous replication)：当主库执行完一个事务,所有的从库都执行了该事务才返回给客户端。

半同步复制(Semisynchronous replication)：主库在执行完客户端提交的事务后，等待至少一个从库接收到并写到中继日志中才返回结果给客户端。

复制模式配置：

1.查看是否允许动态加载模块：默认允许

```
mysql> show variables like "have_dynamic_loading";
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| have_dynamic_loading | YES   |
+----------------------+-------+
```

2.临时加载模块：

a) 主库配置：`mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME "semisync_master.so";`

b) 从库配置：`mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME "semisync_slave.so";`

c) 查看配置：`mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME LIKE "%semi%";`

```
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| rpl_semi_sync_slave  | ACTIVE        |
| rpl_semi_sync_master | ACTIVE        |
+----------------------+---------------+
```

3.临时启用半同步复制：

a) 主库启用：`mysql> SET GLOBAL rpl_semi_sync_master_enabled=1;`

b) 从库启用：`mysql> SET GLOBAL rpl_semi_sync_slave_enabled=1;`

c) 查看启用：mysql> show variables like "rpl_semi_sync_%_enabled";

```
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| rpl_semi_sync_master_enabled  | ON    |
| rpl_semi_sync_slave_enabled   | ON    |
+-------------------------------+-------+
```

4.修改配置文件设置永久启用：

```
[root@host53 ~]# vim /etc/my.cnf
    plugin-load=rpl_semi_sync_master=semisync_master.so
    plugin-load=rpl_semi_sync_slave=semisync_slave.so
    rpl_semi_sync_master_enabled=1
    rpl_semi_sync_slave_enabled=1
```

注意：在有的高可用架构下，通常所有库都需要同时开启主从库配置，以便在切换后能继续使用半同步复制

MySQL 读写分离：

概念：把客户端查询数据的请求和写入数据的请求分发给不同的数据库服务器处理

原理：

多台 MySQL 服务器，分别提供读、写服务,均衡流量，通过主从同步保持数据一致性

mysql 中间件提供代理服务，实现数据读写分离：

mysql-proxy,mycat,maxscale,MHA

由 MySQL 代理面向客户端，设置策略，收到写请求时,交给主库处理，收到读请求时,交给从库处理

配置读写分离：

1.配置主从结构

2.安装 maxscale：从 https://downloads.mariadb.com/MaxScale/ 下载 rpm 包安装

3.配置 maxscale 主配置文件：

```
[root@host53 ~]# vim /etc/maxscale.cnf
    [maxscale]
    threads=auto                    #自动获取 CPU 核数
    [server1]                       #第一台服务器名称
    type=server
    address=192.168.4.52            #第一台服务器 IP
    port=3306
    protocol=MySQLBackend
    [server2]                       #第二台服务器名
    type=server
    address=192.168.4.54            #第二台服务器 IP
    port=3306
    protocol=MySQLBackend
    [MySQL Monitor]
    servers=server1,server2         #设置监控的服务器
    user=test                       #用户名
```

```
passwd=123456                #密码
#[Read-Only Service]        #关闭只读
[Read-Write Service]        #开启读写分离路由
servers=server1,server2     #读写分离的服务器
user=maxscale               #当客户端访问时，该用户向后端服务器验证
客户端的用户名和密码
passwd=123456
[MaxAdmin Service]          #读写分离管理服务
type=service
router=cli
#[Read-Only Listener]       #关闭只读监听
[Read-Write Listener]       #设置读写分离监听
port=4006                   #若代理上不启动 mysql，可修改为 3306
[MaxAdmin Listener]         #管理服务监听
port=4019
```

4.在后端服务器创建监控用户：

```
mysql> grant replication slave, replication client on *.* to test@"%"
identified by "123456";
```

5.在后端服务器创建路由用户：

```
mysql> grant select on mysql.* to maxscale@'%' identified by "123456";
```

6.测试监控用户和路由用户连接和权限：

```
[root@host53 ~]# mysql -h 192.168.4.52 -u test -p123456 -e "show grants"
[root@host53 ~]# mysql -h 192.168.4.54 -u test -p123456 -e "show grants"
[root@host53 ~]# mysql -h 192.168.4.54 -u maxscale -p123456 -e "show grants"
[root@host53 ~]# mysql -h 192.168.4.52 -u maxscale -p123456 -e "show grants"
```

7.启动 maxscale：

```
[root@host53 ~]# maxscale -f /etc/maxscale.cnf
```

8.查看进程，关闭进程：

```
[root@host53 ~]# ss -antup | grep maxscale
[root@host53 ~]# pkill maxscale
```

9.查看后端服务器状态：

```
[root@host53 ~]# maxadmin -u admin -pmariadb -P4019
MaxScale> list servers
Servers.
-----------------+----------------+-------+-------------+-----------
---------
Server           | Address        | Port  | Connections | Status
-----------------+----------------+-------+-------------+-----------
---------
server1          | 192.168.4.52   |  3306 |           0 | Master,
Running
server2          | 192.168.4.54   |  3306 |           0 | Slave,
Running
-----------------+----------------+-------+-------------+-----------
```

10.创建访问数据用户并登陆：

mysql> grant all on *.* to tom@"%" identified by "123456";

[root@host55 ~]#mysql -h192.168.4.53 -utom -p123456 -P4006 #若代理的配置上设置了 3306，可不用-P 参数

MySQL 常见监控项：

操作系统：CPU，内存，IO，网卡

MySQL：运行状况，连接数，qps，tps，慢查询数，打开表数，当前脏页数，锁情况，主从同步状态

MySQL 性能调优：

服务体系结构：8 个功能组件

1.管理工具

2.连接池

3.sql 接口

4.分析器：检查命令，目标的表是否正确

5.优化器

6.查询缓存：存储曾经查找过的记录，该空间时从物理内存中划分出来的

7.存储引擎：myisam,innodb...

8.文件系统：/var/lib/mysql 所在硬盘分区的格式化方式

影响数据库服务器性能的因素：

硬件配置低、老化：查看硬件资源使用率，升级硬件

查看 CPU：uptime

查看内存：free -m

查看硬盘 IO：top 的第三行 wa 参数

网络速度：带宽

数据库版本：升级版本

数据库参数

通过 explain+sql 命令的方式可以查看该命令消耗的资源

type:本次查询表联接类型，从这里可以看到本次查询大概的效率

key:最终选择的索引，如果没有索引的话，本次查询效率通常很差

key_len:本次查询用于结果过滤的索引实际长度

rows:预计需要扫描的记录数，预计需要扫描的记录数越小越好

extra:额外附加信息，主要确认是否出现 Using filesort、Using temporary 类似情况

修改数据库参数设置的方法：

临时：set 参数=值

永久：

vim /etc/my.cnf

[mysqld]

参数=值

并发及连接控制：

最大允许并发连接数：max_connections

mysql> set global max_connections=1000;

值的计算方法：

查看历史最大连接数：mysql> show global status like "max_used_connections";

重置状态值：mysql> flush status;

公式：max_used_connections/max_connections=85%

等待建立连接的超时时间（只在登录时有效，默认 10 秒）：connect_timeout

```
mysql> set global connect_timeout=8;
```

不活动的连接超时时间（默认 28800 秒，即 8 小时）：wait_timeout

```
mysql> set wait_timeout=3600;
```

缓存参数控制：

缓存中预开启的线程数量：thread_cache_size

```
mysql> set global thread_cache_size=2;
```

缓存允许所有线程同时打开表的数量：table_open_cache

```
mysql> set global table_open_cache=800;
```

用于 MyISAM 引擎的关键索引缓存大小（默认 8M）：key_buffer_size

```
mysql> set global key_buffer_size=20000000;
```

为排序读取表的线程分配缓存空间（默认 256K）：sort_buffer_size

```
mysql> set sort_buffer_size=512000;
```

为顺序读取表的线程分配缓存空间（默认 128K）：read_buffer_size

```
mysql> set read_buffer_size=200000;
```

SQL 查询优化：

```
mysql> show variables like "query_cache%";
+------------------------------+---------+
| Variable_name                | Value   |
+------------------------------+---------+
| query_cache_limit            | 1048576 |
| query_cache_min_res_unit     | 4096    |
| query_cache_size             | 1048576 |
| query_cache_type             | OFF     |
| query_cache_wlock_invalidate | OFF     |
+------------------------------+---------+
```

query_cache_size：查询缓存空间大小

query_cache_min_res_unit：查询缓存空间最小单元

query_cache_limit：查询缓存允许存储的单次结果的大小上限

query_cache_type：查询缓存是否开始

query_cache_wlock_invalidate：控制查询缓存的读锁是否无效，对 myisam 存储引擎的表有效，控制脏读

```
mysql> show global status like "qcache%";
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Qcache_free_blocks    | 1       |
| Qcache_free_memory    | 1031832 |
| Qcache_hits           | 0       |
| Qcache_inserts        | 0       |
```

```
| Qcache_lowmem_prunes    | 0       |
| Qcache_not_cached       | 3       |
| Qcache_queries_in_cache | 0       |
| Qcache_total_blocks     | 1       |
+-------------------------+---------+
```

Qcache_hits：记录在查询缓存里查找到数据的次数

Qcache_inserts：记录数据库服务器接受查询请求的次数

Qcache_lowmem_prunes：记录清除查询缓存空间里数据的次数

MySQL 日志类型：/etc/my.cnf

数据库主目录：datadir=/var/lib/mysql

数据库访问接口：socket=/var/lib/mysql/mysql.sock

错误日志：log-error=/var/log/mysqld.log

PID 记录日志：pid-file=/var/run/mysqld/mysqld.pid

查询日志：可用系统默认日志/var/log/messages 代替

启用查询日志：general-log

查询日志文件：general-log-file=目录/文件

慢查询日志：记录耗时较长或不使用索引的查询操作，通常数据库高负载都是由慢查询导致的

启用慢查询日志：slow-query-log

慢查询日志文件：slow-query-log-file

超过指定秒数的 sql 命令才被记录（默认 10 秒）：long-query-time

记录未使用索引的查询：log-queries-not-using-indexes

存储数据的网络结构部署有传输瓶颈：由于网络结构的原因，集群中某台服务器压力过大，调整网络架构

MySQL 高负载解决方案：

1.查看整体负载：sar -d 1

2.查看磁盘 IO：sar -q 1

3.重启服务，重启主机

4.通过 show processlist 或 mysqladmin pr|grep -v Sleep 查看当前正在执行的 sql，查看状态：

Sending data：sql 正在从表中查询数据，如果查询条件没有适当的索引，则会导致 sql 执行时间过长

Copying to tmp table on disk：通常是由于临时结果集太大，超过了数据库规定的临时内存大小，需要拷贝临时结果集到磁盘上，这个时候需要对 sql 语句进行优化

Sorting result 和 Using filesort：sql 执行排序操作消耗较多 cpu，需要添加适当的索引来消除排序，或者缩小排序的结果集

5.是否受到了攻击

6.增加节点

MySQL 多实例：在一台物理主机上运行多个数据库服务

优点：节约运维成本、提高硬件利用率

1.安装支持多实例服务的软件包：

```
[root@host50 ~]# tar -xf mysql-5.7.20-linux-glibc2.12-x86_64.tar.gz
[root@host50 ~]# mv mysql-5.7.20-linux-glibc2.12-x86_64 /usr/local/mysql
```

```
[root@host50 ~]# PATH=/usr/local/mysql/bin:$PATH
```

2.修改主配置文件：每个实例要有独立的数据库目录，监听端口号，实例名称和独立的 sock 文件

```
[root@host50 ~]# vim /etc/my.cnf
    [mysqld_multi]                                  #启用多实例
    mysqld=/usr/local/mysql/bin/mysqld_safe         #指定进程文件的路径
    mysqladmin=/usr/local/mysql/bin/mysqladmin      #指定管理命令路径
    user=root                                       #指定调用进程的用户
    [mysqld1]                                       #实例进程名称
mysqld 后跟实例编号
    port= 3307                                      #端口号
    datadir=/dir3307                                #数据库目录
    socket=/dir3307/mysql3307.sock                  #指定 sock 文件的路
径和名称
    pid-file=/dir3307/mysqld3307.pid                #进程 pid 号文件
    log-error=/dir3307/mysqld3307.log               #错误日志
    [mysqld2]
    port= 3308
    datadir=/dir3308
    socket=/dir3308/mysql3308.sock
    pid-file=/dir3308/mysqld3308.pid
    log-error=/dir3308/mysqld3308.log
```

3.根据配置文件做相应设置：

```
[root@host50 ~]# mkdir /dir330{7,8}
创建 mysql 用户
```

4.初始化授权库：

格式：mysqld --user=mysql --basedir=软件安装目录 --datadir=数据库目录 --initialize

```
[root@host50 ~]# mysqld --user=mysql --basedir=/usr/local/mysql
--datadir=/dir3307 --initialize
    2018-06-02T03:22:14.441931Z 1 [Note] A temporary password is generated
for root@localhost: tx+47F1/-guf
```

5.启动服务：

```
[root@host50 /]# mysqld_multi start 1
```

6.访问测试：

```
[root@host50 /]# mysql -u root -p -S /dir3307/mysql3307.sock
[root@host50 /]# mysql -u root -p -h 127.0.0.1 -P3307
mysql> set password=password("123456");
```

7.停止服务：

```
[root@host50 /]# mysqld_multi --user=root --password=123456 stop 1
[root@host50 ~]# kill $(cat /dir3307/mysqld3307.pid)
```

高性能计算(High-Performance Computing)：致力于开发超级计算机，研究并行算法和开发相关软件。主要用于大规模科学问题、存储和处理海量数据

HPC 环境：
用户管理软件：
LDAP
NIS
存储管理软件：
NFS
LUSTRE
GPFS
调度管理软件：
Torque
Openpbs
Pbspro
Slurm
Lsf
NIS 的安装：
服务端：ypserv ypbind
客户端：ypbind
NIS 的配置：
服务端：

```
[root@proxy yp]# cat /etc/sysconfig/network     #这个文件只有一行
    NISDOMAIN=hpc
[root@proxy yp]# cat /etc/yp.conf               #这个文件只有一行
    domain hpc server 192.168.4.5
[root@proxy yp]# vim /etc/nsswitch.conf +33     #每行加 nis
    passwd:     files nis sss
    shadow:     files nis sss
    group:      files nis sss
[root@proxy yp]# systemctl restart yppasswdd.service
[root@proxy yp]# systemctl restart ypserv.service
[root@proxy yp]# make                           #同步 nis 数据库
```

客户端：

```
[root@client yp]# vim /etc/sysconfig/network    #三个配置文件同服务端
[root@client yp]# vim  /etc/yp.conf
[root@client yp]# vim /etc/nsswitch.conf +33
[root@client yp]# systemctl restart ypbind.service
```

MHA(Master High Availability)高可用集群：

由日本 DeNA 公司的 youshimaton 开发的 MySQL 高可用性环境下故障切换和主从提升的高可用软件

在 MySQL 数据库发生故障后，MHA 能在 30 秒之内自动完成数据库的故障切换操作，在进行故障切换的过程中，MHA 能在最大程度上保证数据的一致性，以达到真正意义上的高可用

下载地址：https://github.com/yoshinorim/mha4mysql-manager/wiki/Downloads

MHA 的组成：

MHA Manager(管理节点)：可以单独部署在一台独立的机器上，也可以部署在一台 slave 节点上

MHA Node(数据节点)：运行在每台 MySQL 服务器上

MHA 的工作过程：

MHA Manager 会定时探测集群中的 master 节点，当 master 出现故障时，它可以自动将最新更新的 slave 提升为新的 master，然后将所有其他的 slave 重新指向新的 master

1.从宕机崩溃的 master 保存二进制日志事件(binlog events)

2.识别含有最新更新的 slave

3.应用最新的 slave 的中继日志(relay log)，将差异的部分更新到其他的 slave

4.应用从 master 保存的二进制日志事件更新未发送给从库中继日志的操作

5.将含有最新更新的 slave 提升为新的 master

6.使其他的 slave 连接新的 master

搭建 MHA 集群：

51：主库，开半同步复制

52：从库（备用主库），开半同步复制

53：从库（备用主库），开半同步复制

54：从库，不做备用主库所以不用开半同步复制

55：从库，不做备用主库所以不用开半同步复制

56：管理主机

1.在数据库主机上创建主从结构：repluser@"%"主从同步用户， root@"%"管理员用户

2.所有主机上安装 mha-node：

```
[root@fzr ~]# for i in {51..56};do pssh -iH 192.168.4.$i "yum -y install perl-DBD-MySQL perl-DBI";done
[root@fzr ~]# for i in {51..56};do pssh -iH 192.168.4.$i " yum -y install mha4mysql-node.noarch";done
```

3.管理主机上安装 perl 依赖包和 mha-manager,并拷贝安装包下 bin 目录的命令：

```
[root@fzr ~]# scp mha4mysql-manager-0.56.tar.gz 192.168.4.56:~
[root@host56 ~]# yum -y install perl-Config-Tiny perl-Parallel-ForkManage perl-Log-Dispatch perl-MIME-Lite perl-ExtUtils-Embed perl-Time-HiRes perl-CPAN
[root@host56 mha4mysql-manager-0.56]# perl Makefile.PL
    [Core Features]
    - DBI                   ...loaded. (1.627)
    - DBD::mysql            ...loaded. (4.023)
    - Time::HiRes           ...loaded. (1.9725)
    - Config::Tiny          ...loaded. (2.14)
    - Log::Dispatch         ...loaded. (2.41)
    - Parallel::ForkManager ...loaded. (1.18)
    - MHA::NodeConst        ...loaded. (0.56)
[root@host56 mha4mysql-manager-0.56]# make
[root@host56 mha4mysql-manager-0.56]# make install
[root@host56 ~]# cp ~/mha4mysql-manager-0.56/bin/* /usr/local/bin
```

masterha_check_ssh：检查 MHA 的 SSH 配置状况

masterha_check_rep：检查 MySQL 复制状况

masterha_manger：启动 MHA

masterha_check_status：检测 MHA 运行状态

masterha_master_monitor：检测 master 是否宕机

4.配置所有数据库主机之间密钥登陆，管理主机能密钥登陆所有数据库主机：

```
[root@fzr ~]# ssh-keygen -N '' -f ~/.ssh/id_rsa
[root@fzr ~]# for i in {51..56};do ssh-copy-id 192.168.4.$i;scp
~/.ssh/id_rsa 192.168.4.$i:~/.ssh/; done
```

5.关闭定时清理日志文件：在主库中关闭即可，从库会同步

```
[root@host51 ~]# mysql -u root -p123456 -e 'set global relay_log_purge=off'
```

6.修改管理主机主配置文件：

```
[root@host56 ~]# mkdir /etc/mha
[root@host56 ~]# cp /root/mha4mysql-manager-0.56/samples/conf/app1.cnf
/etc/mha        #复制模板
[root@host56 ~]# vim /etc/mha/app1.cnf
    [server default]
    manager_workdir=/etc/mha                                    #工
作目录
    manager_log=/etc/mha/manager.log                            #日
志文件
    #master_ip_failover_script=/usr/local/bin/master_ip_failover    #宕
机后自动切换脚本，测试成功后,启动服务前取消注释
    ssh_user=root
    ssh_port=22
    repl_user=repluser                                          #主
从同步用户名
    repl_password=123456
    user=root                                                   #连
接数据库服务器用户名
    password=123456

    [server1]
    hostname=192.168.4.51
    port=3306

    [server2]
    hostname=192.168.4.52
    port=3306
    candidate_master=1                                          #设
置为候选 master

    [server3]
    hostname=192.168.4.53
    port=3306
```

```
candidate_master=1

[server4]
hostname=192.168.4.54
port=3306
no_master=1                                                    #不
竞选 master

[server5]
hostname=192.168.4.55
port=3306
no_master=1
```

7.测试集群密钥对认证配置：

```
[root@host56 ~]# masterha_check_ssh --conf=/etc/mha/app1.cnf
```

8.复制宕机后自动切换脚本：

```
[root@host56 ~]# cp
/root/mha4mysql-manager-0.56/samples/scripts/master_ip_failover
/usr/local/bin/master_ip_failover
[root@host56 ~]# chmod +x /usr/local/bin/master_ip_failover
[root@host56 ~]# vim /usr/local/bin/master_ip_failover          #放在
my();后面
    my $vip = '192.168.4.100/24';
    my $key = "1";
    my $ssh_start_vip = "/sbin/ifconfig eth0:$key $vip";
    my $ssh_stop_vip = "/sbin/ifconfig eth0:$key down";
```

9.测试主从同步状态：

```
[root@host56 ~]# masterha_check_repl --conf=/etc/mha/app1.cnf
```

10.启动 MHA 监控：

```
--remove_dead_master_conf    在 app1.cnf 文件里删除宕机的主库的信息
--ignore_last_failover       忽略.health 文件
[root@host56 ~]# masterha_manager --conf=/etc/mha/app1.cnf
--remove_dead_master_conf --ignore_last_failover
[root@host56 ~]# masterha_check_status --conf=/etc/mha/app1.cnf
    app1 (pid:4693) is running(0:PING_OK), master:192.168.4.51
```

11.测试集群：

将 51 的服务关闭后，假设 52 是含有最新更新的 slave。MHA 将主库交给 52，删除配置文件中 51 的信息，同时设置其他从库（53，54，55）的主库为 52

当 51 修复后，此时的 51 和集群中的主机没有任何关系。将 52 的数据通过备份还原覆盖 51 的数据后，设置 51 的主库为 52

添加配置文件中 51 的信息，关闭 MHA 后重新启动

12.停止服务：

```
[root@host56 ~]# masterha_stop --conf=/etc/mha/app1.cnf
```

mysql 视图：

介绍：一个虚拟表，结构与真实表相似，包含一系列带有名称的列和行数据。其中的数

据来自定义视图时所引用的表，引用视图时动态生成

优点：

简单：用户完全不需要了解视图中的数据是如何得到的，视图中的数据对用户来说是已经过滤好的符合条件的结果集

安全：用户只能看到视图中的数据

数据独立：可以屏蔽表结构变化对用户的影响

限制：

不能在视图上创建索引，不能使用子查询

包含聚集函数、去重显示、分组结构、过滤结果、常量视图、连接查询的视图不能被更新

视图的基本使用：

创建视图：create view 视图名 as select 语句

PS：视图的字段名默认跟原表相同，视图名后加字段名称可以重命名字段名，注意个数必须相同

查看表状态：

查看当前库下的所有表的状态，可以区分视图表和基本表：show table status;

通过创表信息可以查看视图的信息：show create table 表名;

操作视图表：与操作基本表的命令相同。对视图操作即是对基本操作,反之亦然

查询记录：Select

插入记录：Insert into

更新记录：Update

删除记录：Delete from

删除视图：drop view 视图名;

示例：

```
mysql> create view v1 as select * from user;
mysql> create view v2 as select shell from user where shell="/bin/bash";
mysql> create view v3 (id,user) as select uid,name from user;
mysql> show table status\G;
mysql> show create table v2;
mysql> insert into v1(name,uid) value ("zhangs","88");          #增加一条记录，v1 和 user 都能查到
mysql> insert into v2 value("/sbin/nologin");                   #增加一条记录，由于 v2 只能查到/bin/bash，所以只能在 user 中查到
mysql> update v2 set shell="null";                              #修改 v2 中的记录，所以修改后再次查询 v2 为空，user 中的记录被修改
mysql> drop view v1;
```

视图的进阶使用：

设置字段别名：由于视图中的字段名不可以重复

格式：Create as select 表别名.源字段名 as 字段别名 from 源表名 表别名 left join 源表名 表别名 on 条件;

示例：

```
mysql> create view v2 as select a.name as aname,b.name as bname,a.uid as auid,b.uid as buid from a left join b on a.uid=b.uid;
mysql> create view v1 as select a1.name as aname,b1.name as bname from
```

a a1 left join b b1 on a1.uid=b1.uid;

创建或替换视图：创建时,若视图不存在则创建，若视图已存在,会替换已有的视图

格式：Create or replace view 视图名 as select 查询

示例：mysql> create or replace view v2 as select * from user;

定义处理视图的方式：

格式：Create ALGORITHM=选项 view 视图名 as select 查询

选项：

MERAGE：替换方式，视图名直接被视图公式替换，把视图公式合并到了 select 中

TEMPTABLE：具体化方式，先得到视图的执行结果,该结果形成一个中间结果暂存到内存中, 外面的 select 语句就调用了这些中间结果

UNDEFINED：未定义方式，默认使用替换方式

检查选项：当视图是根据另一个视图定义时，视图的更新、删除、插入需要满足范围限制

with local check option：仅受当前视图的限制

with (CASCADED) check option：同时满足当前视图和另一个视图的限制

示例：

mysql> create or replace view v1 as select name,uid from test where uid between 5 and 25;

mysql> create or replace view v2 as select * from v1 where uid <=15 with check option;

mysql> create or replace view v3 as select * from v1 where uid <=15 with local check option;

mysql> update v2 set uid=4 where name="sync"; #失败，因为不能同时满足 v1 的限制

mysql> update v3 set uid=4 where name="sync"; #成功，只需要满足 v3 的限制，但是改完后，不满足 v1 表的条件，v3 表就看不到了

MySQL 存储过程：MySQL 中的脚本，保存的一系列 sql 命令的集合，可以使用变量、条件判断、流程控制等

优点：提高性能、减轻网络负担、可以防止对表的直接访问、避免重复的 sql 操作

注意：存储过程必须在库里创建，并保存在该库下

创建存储过程：

```
delimiter //                #delimiter 关键字声明当前段分隔符，MySQL 默认以";"为分隔符
create procedure 存储过程名称()
begin
功能代码;
...
end
//
delimiter ;
```

查看存储过程：

```
mysql> show procedure status\G;
mysql> select db,name,type from mysql.proc;
```

调用存储过程：Call 存储过程名();

注意：存储过程没有参数时，()可以省略。有参数时,在调用存储过程时,必须传递参数。

删除存储过程：drop procedure 存储过程名;

如果存在则删除：drop procedure if exists 存储过程名;

示例：

```
mysql> delimiter //
mysql> create procedure say()
    -> begin
    -> select * from v3;            #不加库名只能在当前库执行
    -> end
    -> //
mysql> delimiter ;
mysql> select * from mysql.proc where name="say"\G;
mysql> call say;
mysql> drop procedure say;
```

存储过程参数类型：

格式：类型 1 参数名 1 数据类型 1,类型 2 参数名 2 数据类型 2

类型：参数在存储过程中调用时不需要加@符号

in：输入参数，默认类型。给存储过程传值，可以直接传值，也可以通过变量传值，在存储过程中该参数的值不允许修改。

out：输出参数，必须使用变量，该值可在存储过程内部被改变,并存储在变量中返回结果。

inout：输入/输出参数，必须使用变量，调用时获得该参数的值，并且可在存储过程中被改变，存储在变量中返回结果。

示例：

```
mysql> delimiter //
mysql> create procedure say2(username char(10))
    -> begin
    -> select count(name) from db1.user where name=username;
    -> end
    -> //
mysql> delimiter ;
mysql> call say2("lisi");
```

存储过程变量类型：

系统变量：使用 set 命令定义

会话变量：修改只影响到当前的会话

全局变量：修改会影响到整个服务器

用户变量：在客户端连接到数据库服务的整个过程中都是有效的。当前连接断开后所有用户变量失效。

定义用户变量：set @变量名=值;

输出用户变量：select @变量名;

局部变量：仅用于 begin/end 语句块中,语句块执行完毕后,变量失效。调用时可以不加@

定义局部变量：declare 变量名 数据类型;

定义局部变量：set 变量名=值;

输出局部变量：select 变量名;

示例 1：

```
mysql> show session variables\G;
mysql> show global variables\G;
mysql> set @x=99;
```

示例 2：

```
mysql> delimiter //
mysql> create procedure say3()
    -> begin
    -> declare x int default 9;
    -> declare y char(10);
    -> set y="tom";
    -> select x;
    -> select y;
    -> end
    -> //
mysql> delimiter ;
mysql> call say3;
    +------+
    | x    |
    +------+
    |    9 |
    +------+
    +------+
    | y    |
    +------+
    | tom  |
    +------+
mysql> select @x;
    +------+
    | @x   |
    +------+
    |   99 |
    +------+
```

示例 3：

```
mysql> delimiter //
mysql> create procedure say9()
    ->  begin
    ->  declare x int(2);
    ->  declare name char(10);
    ->  select x;
    ->  set x=0;
    ->  select x;
```

```
    ->  set name="bob";
    ->  select name;
    ->  end
    ->  //
mysql> delimiter ;
mysql> call say9;
```

示例 4：

```
mysql> delimiter //
mysql> create procedure say11(in username char(10),out usernum
int(2),inout shellname char(20))
    -> begin
    -> select username;
    -> select usernum;
    -> select shellname;
    -> end
    -> //
mysql> delimiter ;
mysql> set @x="root";
mysql> call say11(@x,@a,@b);                #a 和 b 可以不赋值，此处作用
为占位
```

示例 5：

```
mysql> delimiter //
mysql> create procedure say14(inout shellname char(20))
    -> begin
    -> select name,shell from db1.user where shell=shellname;
    -> set shellname="shutdown";            #重新定义
    -> end
    -> //
mysql> delimiter ;
mysql> set @name="/bin/false";
mysql> call say14(@name);
    +----------+------------+
    | name     | shell      |
    +----------+------------+
    | mysql    | /bin/false |
    | maxscale | /bin/false |
    +----------+------------+
mysql> select @name;
    +----------+
    | @name    |
    +----------+
    | shutdown |
    +----------+
```

示例 6：

```
mysql> delimiter //
mysql> create procedure say(in bash char(20),in nologin char(25),out x
int,out y int)
    -> begin
    -> declare z int;
    -> set z=0;
    -> select count(name) into x from db1.user where shell=bash;
    -> select count(name) into y from db1.user where shell=nologin;
    -> set z=x+y;
    -> select z;
    -> end
    -> //
mysql> delimiter ;
mysql> call say("/bin/false","/sbin/nologin",@g,@h);
    +------+
    | z    |
    +------+
    |   37 |
    +------+
```

示例 7：

```
mysql> set @x=5;
    +------+
    | @x   |
    +------+
    |    5 |
    +------+
mysql> delimiter //
mysql> create procedure say(inout x int)
    -> begin
    -> select count(name) into x from user where uid<10;
    -> end
    -> //
mysql> delimiter ;
mysql> call say(@x);
mysql> select @x;
    +------+
    | @x   |
    +------+
    |    9 |
    +------+
```

算数运算符号：

+：加法运算

-：减法运算

*：乘法运算

/：除法运算

DIV：整除运算

%：取模

条件判断：参照查询条件

数值比较、逻辑比较、范围、空、非空、模糊、正则

流程控制：

顺序结构：

单分支:

```
if 条件 then
功能代码
end if;
```

双分支:

```
if 条件 then
功能代码 1
else
功能代码 2
end if;
```

多分支:

```
if 条件 then
功能代码 1
elseif 条件 then
功能代码 2
else
功能代码 3
end if;
```

示例 1：

```
mysql> drop procedure if exists say;
mysql> delimiter //
mysql> create procedure say(in x int(1))
    -> begin
    -> if x <= 10 then
    -> select * from db1.user where id <=x;
    -> end if;
    -> end
    -> //
mysql> delimiter ;
mysql> call say(1);
mysql> call say(10);
mysql> call say(15);        #超过范围，显示空
```

示例 2：

```
mysql> delimiter //
mysql> create procedure say(in x int(1))
    -> begin
    -> if x is null then
```

```
    -> set x=1;
    -> select * from db1.user where id=x;
    -> else
    -> select * from db1.user where id<= x;
    -> end if;
    -> end
    -> //
mysql> delimiter ;
mysql> call say(null);
mysql> call say(8);
```

循环结构：

条件式循环：反复测试条件，只要成立就执行命令

语法：

```
while 条件判断 do
循环体
end while;
```

示例：

```
mysql> delimiter //
mysql> create procedure say()
    -> begin
    -> declare i int;
    -> set i=1;
    -> while i<=5 do
    -> select name,uid from db1.user where uid=i;
    -> set i=i+1;
    -> end while;
    -> end
    -> //
mysql> delimiter ;
```

无限循环：

语法：

```
loop
循环体
end loop ;
```

示例：

```
mysql> delimiter //
mysql> create procedure say()
    -> begin
    -> declare i int;
    -> set i=1;
    -> loop
    -> select i;
    -> set i=i+1;
    -> end loop;
```

```
    -> end
    -> //
mysql> delimiter ;
```

until 条件判断,不成立时结束循环：

语法：

```
repeat
循环体
until 条件判断
end repeat ;
```

示例：

```
mysql> delimiter //
mysql> create procedure say()
    -> begin
    -> declare i int;
    -> set i=1;
    -> repeat
    -> select i;
    -> set i=i+1;
    -> until i=6
    -> end repeat;
    -> end
    -> //
mysql> delimiter ;
```

控制语句：

LEAVE 标签名：跳出循环，类似 break

ITERATE 标签名：开始下一次循环，类似 continue

示例：

```
mysql> delimiter //
mysql> create procedure say()
    -> begin
    -> declare i int;
    -> set i=0;
    -> loab1:loop
    -> set i=i+1;
    -> if i=3 then
    -> iterate loab1;
    -> end if;
    -> if i>10 then
    -> leave loab1;
    -> end if;
    -> select i;
    -> end loop;
    -> end
    -> //
```

```
mysql> delimiter ;
```

分库分表：

概念：将存放在一个数据库(主机)中的数据,分散存放到多个数据库(主机)中，以达到分散单台设备负载的效果

数据的切分根据其切分规则的类型,分为 2 种切分模式:垂直分割(纵向)和水平分割(横向)

垂直分割：

把单一的表,拆分成多个表,并分散到不同的数据库上。

一个数据库由多个表构成,每个表对应不同的业务,可以按照业务对表进行分类,将其分布到不同的数据库上,实现专库专用,让不同的库分担不同的业务。

水平分割：

把向表中写入的记录分散到多个库中。

按照数据行切分,将表中的某些行存储到指定的数据库中。

数据库安全：

1.运行 mysql 安全脚本：为 root 设置密码，禁止 root 从远程其他主机登陆数据库，删除测试性数据库 test：

```
[root@proxy ~]# mysql_secure_installation
```

2.手动修改密码：

```
[root@proxy ~]# mysqladmin -u root -p123456 password 'mysql'
MariaDB [(none)]> set password for root@'localhost'=password('redhat');
```

3.清空设置密码的记录：

```
[root@proxy ~]# rm -f .mysql_history .bash_history
```

4.选用 5.6 及以后的 mysql 版本

5.备份数据库：

```
[root@proxy ~]# mysqldump -u root -p mysql > mysql.sql          #备份 mysql 数据库
[root@proxy ~]# mysqldump -u root -p mysql user > user.sql      #备份 mysql 中的 user 表
[root@proxy ~]# mysqldump -u root -p --all-databases > all.sql  #备份所有数据库
```

6.还原数据库：

```
[root@proxy ~]# mysql -u root -p mysql < mysql.sql              #还原数据库
[root@proxy ~]# mysql -u root -p mysql < user.sql               #还原数据表
[root@proxy ~]# mysql -u root -p < all.sql                      #还原所有数据库
```

7.创建一个可以远程登陆的普通用户：

```
MariaDB [(none)]> grant all on *.* to tom@'%' identified by '123';
```

远程登陆测试：`[root@client ~]# mysql -u tom -p -h 192.168.4.5`

8.远程传输过程中的数据安全：

使用 SSH 登陆到服务器，再本地登陆数据库

mysql 支持 SSL 加密，确保网络传输过程中，数据是加密的

Mycat：

软件介绍：基于 Java 的分布式数据库系统中间层,为高并发下的分布式提供解决方案

功能介绍：

支持 JDBC 形式连接

支持 MySQL、Oracle、Sqlserver、Mongodb 等

提供数据读写分离服务

可以实现数据库服务器的高可用

提供数据分片服务

基于阿里巴巴 Cobar 进行研发的开源软件

适合数据大量写入数据的存储需求

分片规则：

1 枚举法：sharding-by-intfile

2 固定分片：rule1

3 范围约定：auto-sharding-long

4 求模法：mod-long

5 日期列分区法：sharding-by-date

6 通配取模：sharding-by-pattern

7 ASCII 码求模通配：sharding-by-prefixpattern

8 编程指定：sharding-by-substring

9 字符串拆分哈希解析：sharding-by-stringhash

10 一致性哈希：sharding-by-murmur

工作过程：

当 Mycat 收到一个 SQL 时，会先解析这个 SQL 查找涉及到的表,然后看此表的定义

如果有分片规则,则获取到 SQL 里分片字段的值,并匹配分片函数,得到该 SQL 对应的分片列表

然后将 SQL 发往这些分片去执行,最后收集和处理所有分片返回的结果数据,并输出到客户端

配置 mycat：

1.安装：

```
[root@host56 ~]# yum -y install java-1.8.0-openjdk
[root@host56 ~]# tar -xf
Mycat-server-1.6-RELEASE-20161028204710-linux.tar.gz
[root@host56 ~]# mv mycat/ /usr/local/
```

2.目录结构说明：

```
[root@host56 ~]# ls /usr/local/mycat/
bin  catlet  conf  lib  logs  version.txt
```

bin：mycat 命令

catlet：扩展功能

conf：配置文件

server.xml：设置连接 mycat 服务的账号、密码等

schema.xml：配置 mycat 使用的真实数据库和表

rule.xml：定义 mycat 分片规则

lib：mycat 使用的 jar 包(mycat 是 java 开发的)

log：mycat 启动日志和运行日志

wrapper.log：mycat 服务启动日志 ,启动有问题可以看这个日志的内容

mycat.log：记录 sql 脚本执行后的具体报错内容

3.修改配置文件：

配置标签说明：

<user>...</user>:定义连接 mycat 服务时使用的用户和密码及逻辑库的名字

<schema>...</schema>：定义逻辑数据库，前台仅显示逻辑数据库，后台分库分表

<table name="表名" primaryKey="关键字" dataNode="数据节点" rule="分片规则"/>：定义分片的表

<dataNode name="分片使用的库名" dataHost="物理库主机名" database="物理库数据库名" />：指定数据节点

<datahost name="物理库主机名 4" maxCon="最大连接数" minCon="最小连接数">...</datahost>：指定数据库服务器及连接数据库时的参数

balance 定义负载均衡类型：

0:不开启读写分离机制,所有读操作都发送到当前可用的 writeHost 上。

1：全部的 readHost 与 stand：by：writeHost 参与 select 语句的负载均衡

2：所有读操作都随机的在 writeHost、readhost 上分发。

3：所有读请求随机的分发到 wiriterHost 对应的 readhost 执行，writerHost 不负担读压力

WriteType 定义写操作：

0：所有写操作都发送到可用的 writeHost 上。

1：所有写操作都随机的发送到 readHost。

2：所有写操作都随机发送到 writeHost 和 readhost 上。

switchType 定义切换的模式：

-1：表示不自动切换

1：默认值，表示自动切换

2：基于 MySQL 主从同步的状态决定是否切换,心跳语句为 show slavestatus

3：基于 MySQLgalarycluster 的切换机制(适合集群)，心跳语句为 show status like 'wsrep%'

<writeHost host="标识" url="ip：端口号" user="用户名" password="密码">...</writeHost>:指定数据库服务器的 IP 地址及连接数据库时使用的授权用户名及密码

```
[root@host56 mycat]# vim conf/server.xml                    #可以不修改，使用默认配置，但是逻辑库名必须与 schema.xml 中的一致
    <user name="test">                                      #设置连接 mycat 的用户名
            <property name="password">123456</property>     #设置连接 mycat 的密码
            <property name="schemas">testdb</property>      #设置虚拟的逻辑库名
```

```
        </user>
        <user name="user">
                <property name="password">123456</property>
                <property name="schemas">testdb</property>
                <property name="readOnly">true</property>   #设置只读
        </user>
    [root@host56 mycat]# vim conf/schema.xml
        <mycat:schema xmlns:mycat="http://org.opencloudb/">
            <schema name="testdb" checkSQLschema="false" sqlMaxLimit="100">
                    <table name="travelrecord" dataNode="dn1,dn2"
    rule="auto-sharding-long" />
                    <table name="company" primaryKey="ID" type="global"
    dataNode="dn1,dn2" />
                    <table name="employee" primaryKey="ID"
                dataNode="dn1,dn2" rule="sharding-by-intfile" />
                    ...
            </schema>
            <dataNode name="dn1" dataHost="host54" database="db1" />
            <dataNode name="dn2" dataHost="host55" database="db2" />
            <dataHost name="host54" maxCon="1000" minCon="10" balance="0"
    writeType="0" dbType="mysql" dbDriver="native" switchType="1"
    slaveThreshold="100">
                    <heartbeat>select user()</heartbeat>
                    <writeHost host="hostM1" url="192.168.4.54:3306"
    user="root" password="123456">
                    </writeHost>
            </dataHost>
            <dataHost name="host55" maxCon="1000" minCon="10" balance="0"
    writeType="0" dbType="mysql" dbDriver="native" switchType="1"
    slaveThreshold="100">
                    <heartbeat>select user()</heartbeat>
                    <writeHost host="host2" url="192.168.4.55:3306"
    user="root" password="123456">
                    </writeHost>
            </dataHost>
        </mycat:schema>
```

4.修改数据库服务器配置文件：设置表名不区分字母大小写

```
    [root@fzr ~]# pssh -iH "192.168.4.54 192.168.4.55" "echo
'lower_case_table_names=1' >> /etc/my.cnf"
    [root@fzr ~]# pssh -iH "192.168.4.54 192.168.4.55" "systemctl restart
mysqld"
```

5.启动服务：

```
    [root@host56 ~]# mysql -h 192.168.4.54 -u root -p123456 -e "create database
db1"
```

```
[root@host56 ~]# mysql -h 192.168.4.55 -u root -p123456 -e "create database db2"
[root@host56 ~]# ln -s /usr/local/mycat/bin/mycat /bin
[root@host56 ~]# mycat start
[root@host56 ~]# mycat status
[root@host56 ~]# ss -antup | grep 8066
```

6.客户端测试：

```
[root@fzr ~]# mysql -h192.168.4.56 -P8066 -utest -p123456
mysql> show databases;          #数据库名在 server.xml 定义
    +----------+
    | DATABASE |
    +----------+
    | testdb   |
    +----------+
mysql> show tables;             #所有的表都在 schema.xml 的 table 字段定义，但是只有表名，表实际上不存在
    +-----------------+
    | Tables in testdb |
    +-----------------+
    | company          |
    | customer         |
    | customer_addr    |
    | employee         |
    | goods            |
    | hotnews          |
    | orders           |
    | order_items      |
    | travelrecord     |
    +-----------------+
mysql> create table employee (id tinyint(2) primary key auto_increment,sharding_id enum("1","10"),name char(10));
mysql> insert into employee(id,sharding_id,name) value (1,1,"zhangsan");
mysql> insert into employee(id,sharding_id,name) value (2,10,"lisi");
```

注：创表是需要参考定义，例如 employee 表:需要设置主键字段 id,需要设置哈希字段 sharding_id,且该字段取值受 partition-hash-int.txt 限制

```
[root@host56 mycat]# vim conf/rule.xml
    <table name="employee" primaryKey="ID" dataNode="dn1,dn2" rule="sharding-by-intfile" />

[root@host56 ~]# vim /usr/local/mycat/conf/schema.xml
    <tableRule name="sharding-by-intfile">
        <rule>
            <columns>sharding_id</columns>
            <algorithm>hash-int</algorithm>
```

```
        </rule>
    </tableRule>
    <function name="hash-int"
class="org.opencloudb.route.function.PartitionByFileMap">
        <property name="mapFile">partition-hash-int.txt</property>
    </function>

[root@host56 mycat]# cat conf/partition-hash-int.txt
    1=0
    10=1
```

传统数据库(SQL)的问题：随着数据量的增大、访问的集中，就会出现关系型数据库的负担加重、数据库响应恶化、网站显示延迟等重大影响。
新型数据库(Not Only SQL:NOSQL)：缓存数据库，键值对应(key=value)：memcached、redis、mongodb

Redis(Remode DIctionary Server)：远程字典服务器

介绍:使用 C 语言编写的,遵守 BSD 的开源软件,是一款高性能的(Key/Values)分布式内存数据库,并支持数据持久化的 NoSQL 数据库服务软件

特点：支持数据持久化，可以把内存里数据保存到硬盘中，不仅支持 key/values 类型的数据，同时还支持 list、hash、set、zset 类型，支持主从模式数据备份

1.装包

```
[root@host51 ~]# yum -y install gcc gcc-c++
[root@host51 ~]# tar -xf redis-4.0.9.tar.gz
[root@host51 ~]# cd redis-4.0.9/
[root@host51 redis-4.0.9]# make
[root@host51 redis-4.0.9]# make install
```

2.初始化配置

```
[root@host51 redis-4.0.9]# utils/install_server.sh
    Port           : 6379                          #端口号
    Config file    : /etc/redis/6379.conf          #主配置文件
    Log file       : /var/log/redis_6379.log       #日志文件
    Data dir       : /var/lib/redis/6379           #主目录
    Executable     : /usr/local/bin/redis-server   #启动程序
    Cli Executable : /usr/local/bin/redis-cli      #连接数据库命令
```

3.启动服务：初始化时会自动启动服务

```
[root@host51 ~]# service redis_6379 stop
[root@host51 ~]# service redis_6379 status
[root@host51 ~]# service redis_6379 start
```

4.测试：

```
[root@host51 ~]# ps -C redis
[root@host51 ~]# netstat -antup | grep redis
[root@host51 ~]# redis-cli
[root@host51 ~]# ls /var/lib/redis/6379/dump.rdb    #将数据库中的数据保
```

存到硬盘的文件

常用操作指令：

```
Set 键名 值          #存储变量
get 键名             #获取变量
Select 数据库编号    #切换库,编号范围 0-15,默认在 0 库
Keys *               #打印所有变量
Keys a?              #打印符合正则的变量
Exits 键名           #测试变量是否存在,返回 1 表示存在,返回 0 表示不存在
ttl 键名             #查看生存时间,非负数表示剩余秒数,-1 表示永不过期,-2 表示已过期
type 键名            #查看类型
move 键名 数据库编号#移动变量,1 表示成功,0 表示失败
expire 键名 秒       #设置有效时间
del 键名             #删除变量,1 表示成功,0 表示失败
flushall             #删除所有变量,非所在库的变量也会删除
save                 #保存变量
shutdown             #关闭服务,等同于 service redis_6379 stop
```

示例：

```
127.0.0.1:6379> ping
PONG
127.0.0.1:6379> select 5
OK
127.0.0.1:6379[5]> set v1 1
OK
127.0.0.1:6379[5]> set v2 2
OK
127.0.0.1:6379[5]> set av 3
OK
127.0.0.1:6379[5]> keys *
1) "av"
2) "v2"
3) "v1"
127.0.0.1:6379[5]> keys a?
1) "av"
127.0.0.1:6379[5]> ttl v1
(integer) -1
127.0.0.1:6379[5]> EXPIRE v1 4
(integer) 1
127.0.0.1:6379[5]> ttl v1
(integer) 0
127.0.0.1:6379[5]> ttl v1
(integer) -2
127.0.0.1:6379[5]> EXISTS v1
(integer) 0
```

```
127.0.0.1:6379[5]> EXISTS v2
(integer) 1
127.0.0.1:6379[5]> TYPE v2
string
127.0.0.1:6379[5]> set v1 5
OK
127.0.0.1:6379[5]> move v1 0
(integer) 1
127.0.0.1:6379[5]> keys *
1) "av"
2) "v2"
127.0.0.1:6379[5]> select 0
OK
127.0.0.1:6379> keys *
1) "v1"
127.0.0.1:6379> del v1
(integer) 1
127.0.0.1:6379> get v1
(nil)
127.0.0.1:6379> FLUSHALL
OK
127.0.0.1:6379[5]> save
OK
127.0.0.1:6379[5]> shutdown
```

配置文件选项说明：/etc/redis/6379.conf

常用选项：

```
70:bind 127.0.0.1                    #IP 地址
93:port 6379                         #端口号
102:tcp-backlog 511                  #TCP 最大连接数,包括已连接和正在连接的,
超过的进入等待列表
114:timeout 0                        #连接超时时间,0 表示不超时
131:tcp-keepalive 300                #长连接时间
137:daemonize yes                    #以守护进程方式运行
159:pidfile /var/run/redis_6379.pid #存储 pid 文件
172:logfile /var/log/redis_6379.log #指定日志文件
187:databases 16                     #数据库个数
264:dir /var/li                  127.0.0.1:323               *:*
users:(("chronyd",pid=497,fd=1))
udp   UNCONN     0      0       ::1:323                      :::*
users:(("chronyd",pid=497,fd=2))
tcp   LISTEN     0      128     *:22                         *:*
users:(("sshd",pid=736,fd=3))
tcp   LISTEN     0      100     127.0.0.1:25                 *:*
users:(("master",pid=950,fd=13))
```

```
tcp    ESTAB      0      0        192.168.1.15:22
192.168.1.254:60110      users:(("sshd",pid=7846,fd=3))
tcp    LISTEN     0      80       :::3306                    :::*
users:(("mysqld",pid=8169,fd=31))
b/redis/6379             #数据库主目录
533:# maxclients 10000              #并发连接数量,默认注释
```

内存管理：

```
12-17：说明了内存的单位换算
    12 # 1k => 1000 bytes
    13 # 1kb => 1024 bytes
    14 # 1m => 1000000 bytes
    15 # 1mb => 1024*1024 bytes
    16 # 1g => 1000000000 bytes
    17 # 1gb => 1024*1024*1024 bytes
560：# maxmemory <bytes>           #设置内存空间,<bytes>表示最大内存
565-572：定义内存清除策略：
    565 # volatile-lru             #在设置了过期的key里删除最少使用的key,
使用lru算法
    566 # allkeys-lru             #删除最少使用的key,使用lru算法
    567 # volatile-lfu            #同上,使用lfu算法
    568 # allkeys-lfu             #同上,使用lfu算法
    569 # volatile-random         #在设置了过期的key里随机删除
    570 # allkeys-random          #随机删除key
    571 # volatile-ttl            #删除最近过期的key
    572 # noeviction              #不删除,写满时报错
591：# maxmemory-policy noeviction  #定义使用的策略
602：# maxmemory-samples 5          #选取模板数据的个数,针对lru和ttl策略
```

设置密码：

```
501：# requirepass foobared        #默认注释,注释后不需要密码
```

示例：

```
[root@host51 ~]# vim /etc/redis/6379.conf
    bind 127.0.0.1 192.168.4.51
    requirepass 123456
    maxmemory 1gb
    maxmemory-policy volatile-ttl
[root@host51 ~]# redis-cli -h 192.168.4.51          #-h 主机名
192.168.4.51:6379> auth 123456
[root@host51 ~]# redis-cli -a 123456                #-a 密码
127.0.0.1:6379>
[root@host51 ~]# redis-cli -a 123456 shutdown
[root@host51 ~]# vim /etc/init.d/redis_6379         #修改启动关闭脚本
    8：REDISPORT="6379"                             #如果改了配置文件的端口
号,就需要修改端口号
    43：$CLIEXEC -p $REDISPORT -a 123456 shutdown   #若设置了密码,需要添加
```

-a 参数,若禁用了本地连接,还需要添加-h 参数

LNMP+Redis：

1.安装启动 nginx,MySQL,php

2.安装启动 redis

3.安装 php 扩展支持 Redis：

```
[root@host53 ~]# yum -y install php-devel php-mysql
[root@host53 ~]# wget
ftp://192.168.4.254/soft/redis/php-redis-2.2.4.tar.gz
[root@host53 ~]# tar -xf php-redis-2.2.4.tar.gz
[root@host53 ~]# cd phpredis-2.2.4/
[root@host53 phpredis-2.2.4]# phpize
[root@host53 phpredis-2.2.4]# ./configure
--with-php-config=/usr/bin/php-config
[root@host53 phpredis-2.2.4]# make && make install
    Installing shared extensions:     /usr/lib64/php/modules/
```

4.修改 PHP 配置文件 php.ini,加载模块：

```
[root@host53 ~]# vim /etc/php.ini
     728：extension_dir = "/usr/lib64/php/modules/"
     729：extension = "redis.so"
[root@host53 ~]# php -m | grep -i redis
```

5.测试,此时 redis 未设置密码

```
[root@host53 ~]# vim /usr/local/nginx/html/redis.php
    <?php
    $redis = new redis();
    $redis->connect('127.0.0.1',6379);
    $redis->set('x','666666');
    echo $redis->get('x');
    ?>
```

redis 集群的方案：

redis-cluster 集群：官方提供的集群搭建方案，适用于数据量大的架构

redis+zookeeper 集群

redis+keepalive 集群

redis+sentinel 集群

redis-cluster 集群：

1.装包,配置文件:

```
#!/bin/bash
for i in {51..56}
do
scp redis-4.0.9.tar.gz 192.168.4.$i:~
pssh -iH "192.168.4.$i" "yum -y install gcc gcc-c++"
pssh -iH "192.168.4.$i" "tar -xf redis-4.0.9.tar.gz"
expect << EOF
```

```
spawn ssh 192.168.4.$i
expect "#" {send "cd ~/redis-4.0.9\n"}
expect "#" {send "make && make install\n"}
expect "#" {send "~/redis-4.0.9/utils/install_server.sh\n"}
expect "6379" {send "\r\n"}
expect "/etc/redis/6379.conf" {send "\r\n"}
expect "/var/log/redis_6379.log" {send "\r\n"}
expect "/var/lib/redis/6379" {send "\r\n"}
expect "/usr/local/bin/redis-server" {send "\r\n"}
expect "ENTER" {send "\r\n"}
expect "#" {send "\r"}
EOF
pssh -iH 192.168.4.$i "service redis_6379 stop"
pssh -iH 192.168.4.$i "wget ftp://192.168.4.254/pub/redis.sh"
pssh -iH 192.168.4.$i "bash ~/redis.sh"
pssh -iH 192.168.4.$i "service redis_6379 start"
pssh -iH 192.168.4.$i "service redis_6379 status"
pssh -iH 192.168.4.$i "ss -antup | grep redis"
done
```

redis.sh:

```
#!/bin/bash
i=$(ifconfig eth0 | awk '/inet /{print $2}' | awk -F. '{print $4}')
sed -i "70c bind 192.168.4.$i" /etc/redis/6379.conf;
sed -i "501c #requirepass 123456" /etc/redis/6379.conf
sed -i "93c port 63$i" /etc/redis/6379.conf
sed -i "815s/#.//" /etc/redis/6379.conf
#开启集群
sed -i "823c cluster-config-file nodes.conf" /etc/redis/6379.conf
#设置集群的配置文件名称,不使用默认
sed -i "829c cluster-node-timeout 5000" /etc/redis/6379.conf
#请求超时 5 秒
sed -i "8c REDISPORT=63$i" /etc/init.d/redis_6379
sed -i "43c /usr/local/bin/redis-cli -p 63$i -h 192.168.4.$i shutdown"
/etc/init.d/redis_6379
```

2.创建集群:集群至少需要 3 个主库

```
[root@host51 ~]# yum -y install ruby rubygems ruby-devel
[root@host51 ~]# wget
ftp://192.168.4.254/soft/redis-cluster/redis-3.2.1.gem
[root@host51 ~]# gem install redis-3.2.1.gem
[root@host51 ~]# cd redis-4.0.9/src/
[root@host51 src]# ./redis-trib.rb create --replicas 1 192.168.4.51:6351
192.168.4.52:6352 192.168.4.53:6353 192.168.4.54:6354 192.168.4.55:6355
192.168.4.56:6356
```

```
>>> Creating cluster
>>> Performing hash slots allocation on 6 nodes...
Using 3 masters:
192.168.4.51:6351
192.168.4.52:6352
192.168.4.53:6353
Adding replica 192.168.4.55:6355 to 192.168.4.51:6351
Adding replica 192.168.4.56:6356 to 192.168.4.52:6352
Adding replica 192.168.4.54:6354 to 192.168.4.53:6353
M: 本机 ID 192.168.4.51:6351 slots:0-5460 (5461 slots) master
M: 本机 ID 192.168.4.52:6352 slots:5461-10922 (5462 slots) master
M: 本机 ID 192.168.4.53:6353 slots:9678,10923-16383 (5462 slots)
master
S: 本机 ID 192.168.4.54:6354 replicates 主库 ID
S: 本机 ID 192.168.4.55:6355 replicates 主库 ID
S: 本机 ID 192.168.4.56:6356 replicates 主库 ID
```

注:--replicas n:表示自动为每一个 master 节点分配 n 个 slave 节点,执行前需要清空所有 redis 内所有数据:FLUSHALL,save

哈希池:在写入 key 时,使用 CRC16 算法计算,将计算结果和 16384 求模,得到的结果写入该值所在的主库中

3.查看集群:

```
redis-cli -c -h IP 地址 -p 集群端口          #原端口+10000 为集群端口,-c 表示进入集群端口
    cluster nodes   #查看本机信息
    cluster info    #查看集群信息
[root@fzr ~]# pssh -ih hosts "cat /var/lib/redis/6379/nodes.conf"
[root@host51 ~]# redis-cli -h 192.168.4.51 -p 6351
192.168.4.51:16351> cluster nodes       #能看到整个集群的所有主机信息
192.168.4.51:16351> CLUSTER info        #cluster_state 状态为 ok
```

4.删除集群(创建集群报错时使用):清空/var/lib/redis/6379/下所有集群配置信息,重启服务

```
[root@fzr ~]# pssh -ih hosts "rm -rf /var/lib/redis/6379/*.conf"
[root@fzr ~]# pssh -ih hosts "service redis_6379 stop"
[root@fzr ~]# pssh -ih hosts "service redis_6379 start"
```

5.测试集群:

```
[root@host50 ~]# redis-cli -h 192.168.4.56 -p 6356 -c
192.168.4.56:6356> keys *
    "name"
192.168.4.56:6356> get name
    -> Redirected to slot [5798] located at 192.168.4.52:6352
    "tom"
192.168.4.52:6352> set name lucy
    OK
192.168.4.52:6352> SHUTDOWN
```

```
192.168.4.56:6356> CLUSTER NODES                    #当某一台主库宕机后,
其他主库会选取该主库的一个从库成为新的主库
    56 主机 ID 192.168.4.56:6356@16356 myself,master - 0 1528439562307 7
connected 5461-10922
    52 主机 ID 192.168.4.52:6352@16352 master,fail - 1528439530823
1528439529920 2 disconnected
[root@host52 ~]# service redis_6379 start
192.168.4.56:6356> CLUSTER NODES                    #当宕机修复后,自动
成为新主库的从库
    56 主机 ID 192.168.4.56:6356@16356 myself,master - 0 1528439751000 7
connected 5461-10922
    52 主机 ID 192.168.4.52:6352@16352 slave 56 主机 ID 0 1528439751000 7
connected
[root@host50 ~]# redis-cli -h 192.168.4.51 -p 6351 -c
192.168.4.51:6351> KEYS *
 "name2"
192.168.4.51:6351> get x
-> Redirected to slot [16287] located at 192.168.4.53:6353
    "1"
192.168.4.53:6353> get y
    "2"
192.168.4.53:6353> get z
-> Redirected to slot [8157] located at 192.168.4.56:6356
    "3"
192.168.4.56:6356> quit
```

管理 redis 集群:redis-trib.rb 脚本:

格式:Redis-trib.rb 选项 参数 指定集群

注:以下示例中出现的 192.168.4.51:6351 均为指定集群,可以指定该集群中任意一台节点

常用选项:

add-node 添加新节点

check 对节点主机做检查

reshard 对节点主机重新分片

add-node --slave 添加从节点主机.需要指定主节点需要加--master-id 参数;如果不指定主节点的话,会把新节点随机添加为从节点最少的主节点的从节点

del-node 删除节点主机

1.添加新主节点到集群中:

```
[root@host51 src]# ./redis-trib.rb add-node 192.168.4.57:6357
192.168.4.51:6351
    [OK] New node added correctly.
```

2.查看分配的哈希池:

```
[root@host51 src]# ./redis-trib.rb check 192.168.4.51:6351

    M: 62f11afc937671be216d6fdd6baabce82dc532e8 192.168.4.57:6357
```

```
            #查出 57 节点的 ID
           slots: (0 slots) master
           0 additional replica(s)
        [OK] All 16384 slots covered.
```

3.手动对集群的哈希池重新分配:

```
    [root@host51 src]# ./redis-trib.rb reshard 192.168.4.51:6351

        How many slots do you want to move (from 1 to 16384)? 4096
            #平均分配给 4 个节点,16384/4=4096
        What is the receiving node ID?
    62f11afc937671be216d6fdd6baabce82dc532e8    #输入 57 节点的 ID
        Source node #1:all
            #从所有其他主节点获取哈希池
        Do you want to proceed with the proposed reshard plan (yes/no)? yes
            #确认信息
```

4.添加从节点:

```
    [root@host51 ~]# ~/redis-4.0.9/src/redis-trib.rb add-node --slave
--master-id 62f11afc937671be216d6fdd6baabce82dc532e8 192.168.4.58:6358
192.168.4.51:6351
    [root@host51 ~]# ~/redis-4.0.9/src/redis-trib.rb add-node --slave
192.168.4.58:6358 192.168.4.51:6351
```

5.移除从节点:

```
    [root@host51 ~]# ~/redis-4.0.9/src/redis-trib.rb del-node
192.168.4.51:6351 e735086e556ba114784ad4b32e8d3db9b18dab3c
        >>> SHUTDOWN the node.
            #移除后自动关闭改节点
```

6.移除主节点:

```
    [root@host51 ~]# ~/redis-4.0.9/src/redis-trib.rb reshard
192.168.4.51:6351
        How many slots do you want to move (from 1 to 16384)? 4096
            #转移的槽数,57 节点上有 4096 个槽,都移除
        What is the receiving node ID?
    88553754b3082f3fefeb93cddb2256f6c9cbf594    #接收节点 ID
        Source node #1:62f11afc937671be216d6fdd6baabce82dc532e8
            #源节点 ID
        Source node #2:done
            #done 表示输入结束
        Do you want to proceed with the proposed reshard plan (yes/no)? yes
            #确认信息
    [root@host51 ~]# ~/redis-4.0.9/src/redis-trib.rb del-node
    192.168.4.51:6351 62f11afc937671be216d6fdd6baabce82dc532e8
        >>> SHUTDOWN the node.
```

Redis 主从复制:

结构模式:

一主一从

一主多从

主从从

工作原理:

1. Slave 向 Master 发送 sync 命令
2. Master 启动后台存盘进程,同时收集所有修改数据命令
3. Master 执行完后台存盘进程后,传送整个数据文件到 slave。
4. Slave 接收数据文件后,将其存盘并加载到内存中完成首次完全同步
5. 有新数据产生时,master 继续将新的所以收集到的修改命令依次传给 slave,完成同步。

缺点:

网络繁忙,会产生数据同步延时问题

系统繁忙,会产生数据同步延时问题

查看主从配置信息:info replication

redis 服务运行后,默认都是 master 服务器

```
192.168.4.51:6351> info replication
    # Replication
    role:master
    connected_slaves:0
    master_replid:f385547248f7fe5ff5226fb35fd6654687980702
    master_replid2:0000000000000000000000000000000000000000
    master_repl_offset:0
    second_repl_offset:-1
    repl_backlog_active:0
    repl_backlog_size:1048576
    repl_backlog_first_byte_offset:0
    repl_backlog_histlen:0
```

在从库上指定主库:SLAVEOF host port

```
192.168.4.52:6352> SLAVEOF 192.168.4.51 6351
192.168.4.52:6352> info replication
    # Replication
    role:slave
    master_host:192.168.4.51
    master_port:6351
    master_link_status:up
    master_last_io_seconds_ago:1
    master_sync_in_progress:0
    slave_repl_offset:14
    slave_priority:100
    slave_read_only:1
    connected_slaves:0
    master_replid:f385547248f7fe5ff5226fb35fd6654687980702
    master_replid2:0000000000000000000000000000000000000000
    master_repl_offset:14
```

```
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:14
192.168.4.51:6351> info replication
# Replication
role:master
connected_slaves:1
slave0:ip=192.168.4.52,port=6352,state=online,offset=28,lag=0
master_replid:f385547248f7fe5ff5226fb35fd6654687980702
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:28
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1
repl_backlog_histlen:28
```

从库变回主库:已同步的数据不会消失

重启 redis 服务:主从配置是临时配置,重启后失效

SLAVEOF no one

```
192.168.4.52:6352> SLAVEOF no one
```

哨兵模式 sentinel:主库宕机后,从库自动升级为主库,适用于主从从结构

1.设置 52 是 51 的从库,是 53 的主库:

```
192.168.4.52:6352> info replication
# Replication
role:slave
master_host:192.168.4.51
master_port:6351
master_link_status:up
master_last_io_seconds_ago:2
master_sync_in_progress:0
slave_repl_offset:1597
slave_priority:100
slave_read_only:1
connected_slaves:1
slave0:ip=192.168.4.53,port=6353,state=online,offset=1597,lag=0
master_replid:f385547248f7fe5ff5226fb35fd6654687980702
master_replid2:0000000000000000000000000000000000000000
master_repl_offset:1597
second_repl_offset:-1
repl_backlog_active:1
repl_backlog_size:1048576
repl_backlog_first_byte_offset:1598
```

repl_backlog_histlen:0

2.在任一主机编辑 sentinel.conf 文件:

格式:

port 26379

dir "目录"

logfile "文件名"

sentinel monitor 主库组名(自定义) 主库的 IP 地址 主库 redis 使用的端口 票数

sentinel auth-pass 主库组名 密码

注:大于等于设定票数的才会被升级为主库

[root@host51 ~]# vim /etc/sentinel.conf

sentinel monitor host51 192.168.4.51 6351 1

3.启动哨兵模式:

[root@host51 ~]# redis-sentinel /etc/sentinel.conf &

或 redis-server /etc/sentinel.conf --sentinel &

4.测试:51 主库宕机后,52 检测到后,自动升级为新的主库:

192.168.4.52:6352> info replication

Replication

role:master

connected_slaves:1

slave0:ip=192.168.4.53,port=6353,state=online,offset=8679,lag=0

master_replid:90baa869b90febd75a4110a58184a52c94f56e2e

master_replid2:f385547248f7fe5ff5226fb35fd6654687980702

master_repl_offset:8679

second_repl_offset:8381

repl_backlog_active:1

repl_backlog_size:1048576

repl_backlog_first_byte_offset:1598

repl_backlog_histlen:7082

5.51 恢复后,52 检测到后,将 51 自动加为从库:

192.168.4.52:6352> info replication

Replication

role:master

connected_slaves:2

slave0:ip=192.168.4.53,port=6353,state=online,offset=14196,lag=1

slave1:ip=192.168.4.51,port=6351,state=online,offset=14196,lag=1

master_replid:90baa869b90febd75a4110a58184a52c94f56e2e

master_replid2:f385547248f7fe5ff5226fb35fd6654687980702

master_repl_offset:14334

second_repl_offset:8381

repl_backlog_active:1

repl_backlog_size:1048576

repl_backlog_first_byte_offset:1598

repl_backlog_histlen:12737

5.删除哨兵模式:

```
[root@host52 ~]# jobs -l
    [1]+  2635 运行中               redis-sentinel /etc/sentinel.conf &
[root@host52 ~]# kill -9 2635
[root@host52 ~]# rm -f /etc/sentinel.conf
[root@host51 ~]# vim /etc/redis/6379.conf       #删除最后一行,哨兵模式添加的
    #slaveof 192.168.4.52 6352
[root@fzr ~]# for i in {51..53};do pssh -iH "192.168.4.$i" "service redis_6379 restart";done
```

带验证的主从复制:

1.配置 master:设置连接密码,重启服务,修改启动脚本

2.配置 slave:指定主库 ip,设置连接密码,重启服务

```
[root@host53 ~]# sed -n '282p;289p' /etc/redis/6379.conf
    # slaveof <masterip> <masterport>
    # masterauth <master-password>
[root@host53 ~]# sed -i "282c slaveof 192.168.4.52 6352" /etc/redis/6379.conf
[root@host53 ~]# sed -i "289c masterauth 123456" /etc/redis/6379.conf
```

Redis 持久化:

RDB(Reids DataBase):在指定时间间隔内,将内存中的数据集快照写入硬盘。恢复时,将快照文件直接读到内存里。

持久化时,Redis 服务会创建一个子进程来进行持久化,先将数据写入到一个临时文件中,待持久化过程都结束了,再用这个临时文件替换上次持久化好的文件

整个持久化过程中,主进程不做任何 IO 操作

优点:

在系统不忙时,拥有极高的性能。

如果要进程大规模数据恢复,且对数据完整行要求不是非常高,使用 RDB 更高效。

缺点:意外宕机时,最后一次持久化后的数据会丢失;系统繁忙时,会消耗大量计算机资源

相关配置参数:/etc/redis/6379.conf

```
217:#save ""              #禁用 RDB
254:dbfilename dump.rdb #指定文件名
219:save 900 1            #900 秒内达到 1 次修改,则存盘
220:save 300 10           #300 秒内达到 10 次修改,则存盘
221:save 60 10000         #60 秒内达到 10000 次修改,则存盘
236:stop-writes-on-bgsave-error yes #bgsave 出错,则停止存盘操作
242:rdbcompression yes  #对存盘数据进行压缩
251:rdbchecksum yes     #在存储快照后,使用 crc16 算法对数据校验
```

手动立刻存盘:

```
192.168.4.54:6354> save
192.168.4.54:6354> bgsave
```

备份:cp /var/lib/redis/6379/dump.rdb /var/lib/redis/6379/dump.rdb.bak

还原:cp /var/lib/redis/6379/dump.rdb.bak /var/lib/redis/6379/dump.rdb

AOF(Append Only File):只追加操作的文件,记录所有写操作,不断的将新的写操作,追加到文件的末尾。

优点:可以灵活设置记录方式,通常宕机时,只丢失最后 1 秒的数据

缺点:AOF 文件的体积通常会大于 RDB 件的体积,执行 fsync 策略时的速度可能会比 RDB 慢

相关配置参数:/etc/redis/6379.conf

```
[root@host54 ~]# sed -n "673p;677p" /etc/redis/6379.conf
    appendonly no                          #AOF 功能是否开启
    appendfilename "appendonly.aof"        #存储文件名
[root@host54 ~]# sed -i "673s/no/yes/" /etc/redis/6379.conf
```

AOF 写操作的三种方式:

```
    702:# appendfsync always          #有新的写操作立即记录,性能差,完整性好
    703:appendfsync everysec          #每秒记录一次,宕机时会丢失最后 1 秒的数据
    704:# appendfsync no              #从不记录
```

AOF 日志重写:redis 会记录上次重写时 AOF 文件的大小,默认当 aof 文件是上次 rewrite 后大小的 100%且文件大于 64MB 时触发

```
    744:auto-aof-rewrite-percentage 100
    745:auto-aof-rewrite-min-size 64mb
```

修复 AOF 文件:把文件恢复到最后一次的正确操作,用于将无关的内容写入 appendonly.aof 后的修复

```
redis-check-aof --fix /var/lib/redis/6379/appendonly.aof
```

备份:cp /var/lib/redis/6379/appendonly.aof /var/lib/redis/6379/appendonly.aof.bak

还原:cp /var/lib/redis/6379/appendonly.aof.bak /var/lib/redis/6379/appendonly.aof

数据类型:String 字符串,List 列表,Hash 表

String 字符串:

字符串操作:

```
set key value [ex seconds] [px milliseconds] [nx|xx]:设置键值和过期时间
    ex 秒数:多少秒后过期
    px 毫秒数:多少毫秒后过期
    nx:只有 key 不存在,才执行操作,防止误覆盖
    xx:只有 key 已存在,才执行操作,相当于更新
strlen key:统计字串长度
setrange key offset value:从偏移量 offset 开始复写 key 的一部分值,偏移量从 0 开始
append key value:字符存在则追加,不存在则创建 key 及 value,返回值为 key 的长度
```

示例:

```
192.168.4.54:6354> set x 100 ex 5        #设置 x=100,5 秒后过期
192.168.4.54:6354> set x 100 px 3000     #设置 x=100,3 秒后过期
192.168.4.54:6354> set x 100 nx          #x 已过期,设置 x=100
192.168.4.54:6354> set x 200 xx          #更新 x=200
```

```
192.168.4.54:6354> set phone 12345678901 ex 100 nx
192.168.4.54:6354> STRLEN phone
    (integer) 11
192.168.4.54:6354> SETRANGE phone 3 ****
192.168.4.54:6354> get phone
    "123****8901"
192.168.4.54:6354> APPEND name bob
    (integer) 3
192.168.4.54:6354> get name
    "bob"
192.168.4.54:6354> APPEND name tom
    (integer) 6
192.168.4.54:6354> get name
    "bobtom"
```

setbit key offset value:对 key 所存储字串,设置或重写特定偏移量上的位,偏移量从 0 开始

key 不存在,则创建新 key

offset 为 0~2^{32}之间的数

Value 值可以为 1 或 0

bitcount key:统计字串中被设置为 1 的比特位数量

可以用于记录网站用户的登录数据,一年的数据只需要 45 字节储存

示例:

```
192.168.4.54:6354> SETBIT x 0 1      #设置第 0 位为 1
192.168.4.54:6354> SETBIT x 1 1      #设置第 1 位为 1
192.168.4.54:6354> SETBIT x 2 0      #设置第 2 位为 1
192.168.4.54:6354> BITCOUNT x        #统计 1 的个数
    (integer) 2
192.168.4.54:6354> SETBIT name 3 1
192.168.4.54:6354> get name
    "robtom"
```

decr key:将 key 中的值减 1,若 key 不存在则先初始化为 0,再减 1

decrby key decrement:将 key 中的值,减去 decrement

incr key:将 key 的值加 1,若 key 不存在则先初始化为 0,再加 1

incrby key increment:将 key 的值,加上 increment

incrbyfloat key increment:为 key 中所储存的值加上浮点数增量 increment

示例:

```
192.168.4.54:6354> set x 100
192.168.4.54:6354> DECR x
    (integer) 99
192.168.4.54:6354> DECR y
    (integer) -1
192.168.4.54:6354> DECRBY x 10
    (integer) 89
192.168.4.54:6354> incr x
```

```
(integer) 90
192.168.4.54:6354> INCRBY x 10
(integer) 100
192.168.4.54:6354> INCRBYFLOAT y 5.5
"4.5"
```

get key:返回 key 所存储的字符串值,如果 key 不存在则返回特殊值 nil,如果 key 的值不是字串,则返回错误

getrange key start end:返回字串值中的子字串,截取范围为 start 和 end,偏移量从 0 开始。负数偏移量表示从末尾开始数,-1 表示最后一个字符

mset key1 value1 key2 value2 ...:一次设置多个 key 及值,空格分隔

mget key1 key2 ...:一次获取一个或多个 key 的值,空格分隔

示例:

```
192.168.4.54:6354> GETRANGE name -3 -1
"tom"
192.168.4.54:6354> GETRANGE name 2 -1
"btom"
192.168.4.54:6354> GETRANGE name -4 5
"btom"
192.168.4.54:6354> mset v1 1 v2 2 v3 3
192.168.4.54:6354> mget v1 v2 v3
1) "1"
2) "2"
3) "3"
```

Hash 表:

特点:一个 string 类型的 field 和 value 的映射表,一个 key 可对应多个 field,一个 field 对应一个 value

有点:将一个对象存储为 hash 类型,较于每个字段都存储成 string 类型更能节省内存

hset key field value:将 hash 表中 field 值设置为 value

hget key filed:获取 hash 表中 field 的值

hmset key field value [field value...]:同时给 hash 表中的多个 field 赋值

hmget key field [field...]:返回 hash 表中多个 field 的值

hkeys key:返回 hash 表中所有 field 名称

hgetall key:返回 hash 表中所有 field 和值

hvals key:返回 hash 表中所有 field 的值

hdel key field [field...]:删除 hash 表中多个 field 的值,不存在则忽略

示例:

```
192.168.4.54:6354> hset book name ywzd
192.168.4.54:6354> HSET book zuozhe dmy
192.168.4.54:6354> HSet book fxrq 20170101
192.168.4.54:6354> HSET book age 30
192.168.4.54:6354> HKEYS book
1) "name"
2) "zuozhe"
3) "fxrq"
```

```
4) "age"
192.168.4.54:6354> hget book name
"ywzd"
192.168.4.54:6354> HMSET book pay 20 page 600
192.168.4.54:6354> HMGET book pay page
1) "20"
2) "600"
192.168.4.54:6354> HGETALL book
 1) "name"
 2) "ywzd"
 3) "zuozhe"
 4) "dmy"
 5) "fxrq"
 6) "20170101"
 7) "age"
 8) "30"
 9) "pay"
10) "20"
11) "page"
12) "600"
192.168.4.54:6354> HVALS book
1) "ywzd"
2) "dmy"
3) "20170101"
4) "30"
5) "20"
6) "600"
192.168.4.54:6354> HDEL book fxrq age
```

List 列表:一个字符队列,先进后出,最后输入的数字在列表的开头,一个 key 可以有多个值

lpush key value [value...]:将一个或多个值 value 插入到列表 key 的表头,若 Key 不存在,则创建 key

lrange key start stop:从 start 位置读取 key 的值到 stop

lpop key:移除列表头元素并返回值,key 不存在则返回 nil

llen key:返回列表 key 的长度

lindex key index:返回列表中第 index 个值

lset key index value:将 key 中 index 位置的值修改为 value

rpush key value [value...]:将 value 插入到 key 的末尾

rpop key:删除列表末尾的值并返回值,key 不存在则返回 nil

示例:

```
192.168.4.54:6354> LPUSH num a b c d
192.168.4.54:6354> LRANGE num 0 2
1) "d"
2) "c"
3) "b"
```

```
192.168.4.54:6354> LPOP num
"d"
192.168.4.54:6354> LLEN num
(integer) 3
192.168.4.54:6354> LINDEX num 2
"a"
192.168.4.54:6354> LINDEX num 0
"c"
192.168.4.54:6354> LINDEX num -2
"b"
192.168.4.54:6354> LSET num 2 e
192.168.4.54:6354> LSET num -2 f
192.168.4.54:6354> RPUSH num z x y
(integer) 6
192.168.4.54:6354> RPOP num
"y"
```

通用操作:

del key [key...]:删除一个或多个 key
exists key:测试一个 key 是否存在
expire key seconds:设置 key 的生存周期
persist key:设置 key 永不过期
ttl key:查看 key 的生存周期
keys 正则表达式:找符合匹配条件的 key,特殊符号用\屏蔽
move key db_id:将当前数据库的 key 移动到 db_id 数据库中
rename key newkey:给 key 改名为 newkey,newkey 已存在时,则覆盖其值
renamenx key newkey:仅当 newkey 不存在时,才将 key 改名为 newkey
sort key:对 key 进行升序排序
sort key desc:进行降序排序
type key:返回 key 的数据类型

MongoDB:

介绍:

介于关系数据库和非关系数据库之间的产品

一个基于分布式文件存储的数据库。由 C++语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

将数据存储为一个文档,数据结构由键值对组成。字段值可以包含其他文档,数组及文档数组。

特点:

安装简单
面向文档存储,操作比较简单容易
支持丰富的查询表达
可以设置任何属性的索引
支持主流编程语言 RUBY|PYTHON|JAVA|PHP|C++
支持副本集,分片

搭建 MongoDB 服务器:

1.装包

```
[root@host51 ~]# wget
ftp://192.168.4.254/soft/mongodb/mongodb-linux-x86_64-rhel70-3.6.3.tgz
[root@host51 ~]# tar -xf mongodb-linux-x86_64-rhel70-3.6.3.tgz
[root@host51 ~]# mkdir /usr/local/mongodb
[root@host51 ~]# cp -r ~/mongodb-linux-x86_64-rhel70-3.6.3/bin
/usr/local/mongodb/
[root@host51 ~]# ln -s /usr/local/mongodb/bin/* /bin/
[root@host51 ~]# cd /usr/local/mongodb
[root@host51 mongodb]# mkdir -p etc log data/db
```

2.手动创建配置文件

```
[root@host51 mongodb]# vim etc/mongodb.conf
    logpath=/usr/local/mongodb/log/mongodb.log#日志文件
    logappend=true#以追加方式记录日志
    dbpath=/usr/local/mongodb/data/db#数据库目录
    fork=true#守护进程方式运行
```

3.启动服务

```
[root@host51 ~]# mongod -f /usr/local/mongodb/etc/mongodb.conf
[root@host51 ~]# ps -C mongod
[root@host51 ~]# ss -antup  | grep 27017
```

4.连接服务:

```
[root@host51 ~]# mongo
    MongoDB shell version v3.6.3
    connecting to: mongodb://127.0.0.1:27017
> show dbs
    admin   0.000GB
    config  0.000GB
    local   0.000GB
> exit
```

MongoDB 命令:命令严格区分大小写

库管理命令:

```
show dbs                                    #查看已有的库
db                                          #显示当前所在的库
use 库名                                    #切换库,若库不存在的话
```

自动延时创建库

```
show tables                                 #查看库下已有的集合,也
```

可以用 show collections

```
db.dropDatabase()                           #删除当前所在的库
```

库名需满足以下条件:

不能是空字符串

不能含有空格、.、$、/、\和空字符

使用小写

最多 64 字节

集合管理

db.集合名.drop()　　#删除集合

db.集合名.save({"键1":"值1","键2":"值2",...}) #添加文档,集合不存在时,创建集合并添加文档

集合名需满足以下条件:

不能是空字符串

不能含有空字符

不能以"system."开头

不能含有保留字符

文档管理:

db.集合名.find()　　#查看所有记录

db.集合名.count()　　#统计记录数

db.集合名.insert({"键":"值"})　　#插入一条文档

db.集合名.find({条件})　　#按条件查找

db.集合名.findOne()　　#返回一条文档

db.集合名.remove({})　　#删除所有文档

db.集合名.remove({条件})　　#删除与条件匹配的所有文档

示例:

```
> use t1                          #t1 库不存在,创建 t1 库并进入
    switched to db t1
> db                              #查看所在库
    t1
> db.test.save({name:"lucy",sex:"girl",age:"20"})    #添加 test 文档,并写入数据
    WriteResult({ "nInserted" : 1 })
> db.test.count()                 #查看记录数
    1
> db.test.insert({name:"tom"})  #插入一条文档
    WriteResult({ "nInserted" : 1 })
> db.test.count()
    2
> db.test.find()                  #查看记录
    { "_id" : ObjectId("5b1f301479a5f8fbf1c9446f"), "name" : "lucy", "sex" : "girl", "age" : "20" }
    { "_id" : ObjectId("5b1f305779a5f8fbf1c94470"), "name" : "tom" }
> db.test.find(name:"lucy")      #查看名字为 lucy 的记录
    2018-06-12T10:31:32.503+0800 E QUERY    [thread1] SyntaxError: missing ) after argument list @(shell):1:17
> db.test.remove({name:"tom"})  #删除名字为 tom 的记录
    WriteResult({ "nRemoved" : 1 })
> db.test.remove({})              #删除所有记录
    WriteResult({ "nRemoved" : 1 })
> show tables                     #查看库下的文档
```

```
        test
    > db.test.drop()                    #删除 test 文档
        true
    > show dbs                          #查看所有数据库
        admin   0.000GB
        config  0.000GB
        local   0.000GB
        t1      0.000GB
    > db.dropDatabase()                 #删除当前所在的库:t1
        { "dropped" : "t1", "ok" : 1 }
    > show dbs
        admin   0.000GB
        config  0.000GB
        local   0.000GB
```

基本数据类型:

字符串(string):{键:"值"}

布尔(bool):{键:true}

空(null):{键:null}

数值:shell 默认使用 64 位浮点型数值:

{键:3.14}

NumberInt(4 字节整数):{键:NumberInt(3)}

NumberLong(8 字节整数):{键:NumberLong(3)}

数组(array):数据列表或数据集可以表示为数组:{键:["值 1","值 2",...]}

代码:查询和文档中可以包括任何 JavaScript 代码:

{键:function(){/*代码*/}}

日期:日期被存储为自新纪元依赖经过的毫秒数,不存储时区:

{键:new Date()}

对象:对象 id 是一个 12 字节的字符串,是文档的唯一标识,不指定会自动生成:

{键:ObjectId()}

正则表达式:使用正则表达式作为限定条件:

{键:/正则表达式/}

内嵌:文档可以嵌套其他文档,被嵌套的文档作为值来处理:

{键:{键:"值",...}}

示例:

```
> db.t1.save({name:"张三"})
> db.t1.save({x:true})
> db.t1.save({y:null})
> db.t1.save({z:3.14})
> db.t1.save({z2:NumberLong(3)})
> db.t1.save({z1:NumberInt(3.14)})
> db.t1.save({zz:["a","b","c"]})
> db.t1.save({php:function(){/*<?php echo "hello"?>*/}})
> db.t1.save({d:new Date()})
```

```
> db.t1.save({o:ObjectId(),a:/^a/})
> db.t1.save({tarena:{address:"beijing",tel:"88888888"}})
> db.t1.find()
    { "_id" : ObjectId("5b1f41c579a5f8fbf1c94471"), "name" : "张三" }
    { "_id" : ObjectId("5b1f41d679a5f8fbf1c94472"), "x" : true }
    { "_id" : ObjectId("5b1f41df79a5f8fbf1c94473"), "y" : null }
    { "_id" : ObjectId("5b1f41f179a5f8fbf1c94474"), "z" : 3.14 }
    { "_id" : ObjectId("5b1f421979a5f8fbf1c94476"), "z2" :
NumberLong(3) }
    { "_id" : ObjectId("5b1f422079a5f8fbf1c94477"), "z1" : 3 }
    { "_id" : ObjectId("5b1f428679a5f8fbf1c94478"), "zz" : [ "a", "b",
"c" ] }
    { "_id" : ObjectId("5b1f42c079a5f8fbf1c94479"), "php" : { "code" :
"function (){/*<?php echo \"hello\"?>*/}" } }
    { "_id" : ObjectId("5b1f42ce79a5f8fbf1c9447a"), "d" :
ISODate("2018-06-12T03:49:34.561Z") }
    { "_id" : ObjectId("5b1f432279a5f8fbf1c9447c"), "o" :
ObjectId("5b1f432279a5f8fbf1c9447b"), "a" : /^a/ }
    { "_id" : ObjectId("5b1f43c179a5f8fbf1c9447d"), "tarena" :
{ "address" : "beijing", "tel" : "88888888" } }
```

修改 mongodb 的 IP 地址和端口号:

```
[root@host51 ~]# tail -2 /usr/local/mongodb/etc/mongodb.conf
    bind_ip=192.168.4.51
    port=27051
```

重启 mongodb 服务后,连接需要带 IP 和端口号

```
[root@host51 ~]# mongo --host 192.168.4.51:27051
[root@host51 ~]# mongo --host 192.168.4.51 --port 27051
```

mongodb 数据导入导出:

导入:mongoimport

--host　IP 地址,默认 IP 地址 127.0.0.1 时可不输

--port　端口,默认端口 27017 时可不输

-d　　库名

-c　　集合名

-f　　指定指定每个字段的键,键的数量超过字段的数量后忽略后面的键,至少需要指定 2 个键,csv 格式用

--type　指定文件格式,支持 json,csv 和 tsv,默认为 json

--drop　删除原有数据后导入新数据

--headerline　　将导入的第一行当做字段名列表,适用于 csv 和 tsv

注:导入数据时库和集合不存在时,会创建库和集合后导入数据,若存在则以追加的方式导入数据到集合里

ID 字段为唯一标识,若导入数据与已有的 ID 重复,则导入失败,可修改或删除导入数据的_id 字段

导出:mongoexport

--host　IP 地址,默认 IP 地址 127.0.0.1 时可不输

```
    --port   端口,默认端口 27017 时可不输
    -d       库名
    -c       集合名
    -f       指定字段名列表,csv 格式用
    -q       指定条件
    --type   指定文件格式,支持 json 或 csv,默认为 json
示例:
    [root@host51 ~]# mongoexport --host 192.168.4.51:27051 -d test -c t1> /tmp/ex1.json
    [root@host51 ~]# mongoexport --host 192.168.4.51:27051 -d test -c t1 -q '{name:"张三"}' > /tmp/ex2.json
    [root@host51 ~]# mongoexport --host 192.168.4.51:27051 -d test -c t1 -f name,x --type=csv > /tmp/ex3.csv
    [root@host51 ~]# cat /tmp/ex2.json                                  #删除导出数据的_id 字段
        {"name":"张三"}
    [root@host51 ~]# cat /tmp/ex3.csv
        name,x
        张三,
        ,true
        ,
        ...                                                             #匹配不到的都跟最后一行一样
    [root@host51 ~]# mongoimport --host 192.168.4.51:27051 -d test -c t2 /tmp/ex1.json
    [root@host51 ~]# mongoimport --host 192.168.4.51:27051 -d test -c t2 /tmp/ex2.json
    [root@host51 ~]# mongoimport --host 192.168.4.51:27051 -d test -c t2 --type=csv --headerline /tmp/ex3.csv
    [root@host51 ~]# mongoimport --host 192.168.4.51:27051 -d test -c t2 --type=csv -f name,x,xx  /tmp/ex3.csv
    > db.t2.find()
        { "_id" : ObjectId("5b1f796872c6b0ce7c91a19f"), "name" : "name", "x" : "x" }
        { "_id" : ObjectId("5b1f796872c6b0ce7c91a1a0"), "name" : "张三", "x" : "" }
        { "_id" : ObjectId("5b1f796872c6b0ce7c91a1a1"), "name" : "", "x" : "true" }
        { "_id" : ObjectId("5b1f796872c6b0ce7c91a1a2"), "name" : "", "x" : "" }
        ...                                                             #无数据的匹配空值,跟最后一行一样
    [root@host51 ~]# sed "s/:/,/g" /etc/passwd > /tmp/pass.csv
    [root@host51 ~]# mongoimport --host 192.168.4.51:27051 -d test -c user --type=csv -f name,pass,uid,gid,home,comm,shell /tmp/pass.csv
```

mongodb 数据备份恢复:

数据备份:mongodump

指定库:-d 数据库名　　#默认所有库

指定集合:-c 集合名　　#默认所有集合

指定目录:-o 目录　　#目录无需事先创建,默认当前目录的 dump 目录下

示例:

```
[root@host51 ~]# mongodump --host 192.168.4.51:27051
[root@host51 ~]# mongodump --host 192.168.4.51:27051 -d test
[root@host51 ~]# mongodump --host 192.168.4.51:27051 -d test -c user
[root@host51 ~]# mongodump --host 192.168.4.51:27051 -d test -c user
-o /tmp/
```

数据恢复:mongorestore

恢复到指定库:-d 数据库名　　#若该库不存在,则自动创建

指定恢复的集合:-c 集合名　　#默认所有集合,若指定集合,则必须指定 bson 文件

示例:

```
[root@host51 ~]# mongorestore --host 192.168.4.51:27051 -d game
dump/test/
[root@host51 ~]# mongorestore --host 192.168.4.51:27051 -d game -c haha
/tmp/test/user.bson
```

MongoDB 副本集:

介绍:具有访问节点的高可用性,并提供了数据冗余备份,并可以保证数据的安全性,允许从硬件故障和服务中断中恢复数据

原理:至少需要两个节点。其中一个是主节点,负责处理客户端请求,其余的都是从节点,负责复制主节点上的数据。

搭配方式:一主一从、一主多从

工作过程:主节点记录在其上所有操作的 oplog,从节点定期轮询主节点获取这些操作,然后对自己的数据副本执行这些操作,从而保证从节点的数据与主节点一致。

Master-Slave 主从复制:已淘汰

Replica Sets 复制集:故障自动切换和自动修复成员节点,各个 DB 之间数据完全一致,降低了维护成本,使用 replicaset 故障切换完全自动。

配置 Replica Sets 副本集:

1.修改配置文件,指定主机所在的副本集名称,同一个副本集成员使用相同的副本集名称

```
[root@fzr ~]# for i in {51..53};do pssh -iH 192.168.4.$i "echo
replSet=rs1 >> /usr/local/mongodb/etc/mongodb.conf"; done
[root@host51 ~]# tail -1 /usr/local/mongodb/etc/mongodb.conf
    replSet=rs1
```

2.启动服务

3.在任一副本集成员主机上连接 mongod 服务,配置节点信息:

格式:config={_id:"rs1",members:[{_id:0,host:"IP 地址:端口"},{_id:1,host:"IP 地址:端口"},{_id:2,host:"IP 地址:端口"}]};

```
[root@host51 ~]# mongo --host 192.168.4.51:27051
> config={_id:"rs1",members:[
```

```
... {_id:0,host:"192.168.4.51:27051"},
... {_id:1,host:"192.168.4.52:27052"},
... {_id:2,host:"192.168.4.53:27053"}]}
```

4.初始化 Replica Sets 环境

```
> rs.initiate(config)
```

5.查看副本集信息

查看状态信息:

```
rs1:SECONDARY> rs.status()
    "set" : "rs1",                          #副本集名称
        ...
        "myState" : 1,                      #副本集状态
        ...
    "members" : [
        {
            "_id" : 0,                      #ID
            "name" : "192.168.4.51:27051",
            "health" : 1,                   #是否正常,1 表示正常
            "state" : 1,                    #状态,1 表示主库,2 表示从库
            "stateStr" : "PRIMARY",         #标识,PRIMARY 表示主
库,SECONDARY 表示从库
        ...}
        {
            "_id" : 1,
            "name" : "192.168.4.52:27052",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
        ...}
        {
                "_id" : 2,
                "name" : "192.168.4.53:27053",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
        ...}
```

查看主从库信息:

```
rs1:PRIMARY> rs.isMaster()
    "hosts" : [                             #主机
            "192.168.4.51:27051",
            "192.168.4.52:27052",
            "192.168.4.53:27053"
        ],
        "setName" : "rs1",                  #副本集名称
        "setVersion" : 1,                   #副本集版本
```

```
        "ismaster" : true,                  #主库标识,true 表示是,false
表示否
        "secondary" : false,                #从库标识
        "primary" : "192.168.4.51:27051",   #主库地址
        "me" : "192.168.4.51:27051",        #本机是谁
    ...
```

查看配置:

```
rs1:PRIMARY> rs.conf()
```

6.验证

在从库上设置允许从库查看数据:rs1:SECONDARY> db.getMongo().setSlaveOk()

同步数据验证:在主库上写入信息,在从库上查看

```
rs1:PRIMARY> db.t1.save({name:"bob"})
rs1:SECONDARY> db.t1.find()
    { "_id" : ObjectId("5b207f7266acc281bb419bad"), "name" : "bob" }
```

自动切换主库验证:bully 算法(欺负算法)

当主库宕机后,剩余的从库开始执行 bully 算法,由编号最大的从库接管主库

当宕机被修复后,先同步所有数据,然后重新开始选举,如果恰好是编号最大的,则重新接管主库

```
[root@host51 ~]# kill 2070
rs1:SECONDARY> rs.isMaster()
    {
        "hosts" : [
            "192.168.4.51:27051",
            "192.168.4.52:27052",
            "192.168.4.53:27053"
        ],
        "setName" : "rs1",
        "setVersion" : 1,
        "ismaster" : true,
        "secondary" : false,
        "primary" : "192.168.4.52:27052",
        "me" : "192.168.4.52:27052",
    ...}
[root@host51 ~]# mongod -f /usr/local/mongodb/etc/mongodb.conf
rs1:SECONDARY> rs.isMaster()
    {
        "hosts" : [
            "192.168.4.51:27051",
            "192.168.4.52:27052",
            "192.168.4.53:27053"
        ],
        "setName" : "rs1",
        "setVersion" : 1,
        "ismaster" : false,
```

```
        "secondary" : true,
        "primary" : "192.168.4.52:27052",
        "me" : "192.168.4.51:27051",
    ...}
```

MongoDB 文档管理:

插入:

1.save 方式

格式:db.集合名.save({key:“值”,key:”值”})

集合不存在时创建集合,后插入记录

_id 字段值已存在时修改文档字段值

_id 字段值不已存在时插入文档

2.insert 方式

格式:db.集合名.insert({key:“值”,key:”值”})

集合不存在时创建集合,后插入记录

_id 字段值已存在时放弃插入

_id 字段值不已存在时插入文档

3.插入多条记录

格式:db.集合名.insertMany([{key:"值",key:"值"},{key:"值",key:"值"}])

示例:

```
rs1:PRIMARY> db.t1.insert({_id:1,name:"zhangsan"})
rs1:PRIMARY> db.t1.insert({_id:2,name:"lisi"})
rs1:PRIMARY> db.t1.save({_id:2,name:"wangwu"})
rs1:PRIMARY> db.t1.find()
{ "_id" : 1, "name" : "zhangsan" }
{ "_id" : 2, "name" : "wangwu" }
rs1:PRIMARY>
db.t1.insertMany([{_id:3,name:"lucy"},{_id:4,name:"tom",age:20},{_id:5,name:"jack"}])
```

查询:

显示所有行:db.集合名.find()

显示第 1 行:db.集合名.findOne()

指定查询条件并指定显示的字段:db.集合名.find({条件},{定义显示的字段})

条件:

指定值:{key:"值",key:"值"}

范围比较:

在列表里:{key:{$in:[n1,n2...]}}

不在列表里:{key:{$nin:[n1,n2...]}}

或:{$or:[{key1:值 1},{key2:值 2}]}

正则匹配:{key:/正则表达式/}

数值比较:{key:{比较符:n}}

比较符:$lt,$lte,$gt,$gte,$ne

匹配空或没有的字段:{key:null}

定义显示的字段:0:不显示,1:显示。不定义时,显示所有字段,有定义时,默认

显示_id,其他不显示
显示前几行:db.集合名.find().limit(n)
跳过前几行:db.集合名.find().skip(n)
排序:db.集合名.find().sort({key:n})　　#n=1 表示升序,n=0 表示降序
　　优先级:sort>skip>limit
示例:

```
　　rs1:PRIMARY> db.user.findOne()
　　rs1:PRIMARY> db.user.find({},{name:1,uid:1})
　　rs1:PRIMARY> db.user.find({},{_id:0,name:1,uid:1})
　　rs1:PRIMARY> db.user.find({},{_id:0})
　　rs1:PRIMARY> db.user.find({},{_id:0,name:1,uid:1}).limit(5)
　　rs1:PRIMARY>
db.user.find({},{_id:0,name:1,uid:1}).skip(10).limit(3)
　　rs1:PRIMARY> db.user.find({},{_id:0,name:1,uid:1}).sort({uid:1})
　　rs1:PRIMARY>
db.user.find({},{_id:0,name:1,uid:1}).sort({uid:1}).skip(2).limit(5)
　　{ "name" : "daemon", "uid" : 2 }
　　{ "name" : "adm", "uid" : 3 }
　　{ "name" : "lp", "uid" : 4 }
　　{ "name" : "sync", "uid" : 5 }
　　{ "name" : "shutdown", "uid" : 6 }
　　rs1:PRIMARY> db.user.find({ "name" : "daemon", "uid" : 2 }
　　{ "name" : "adm", "uid" : 3 }
　　{ "name" : "lp", "uid" : 4 }
　　{ "name" : "sync", "uid" : 5 }
　　{ "name" : "shutdown", "uid" :
6 }{},{_id:0,name:1,uid:1}).limit(5).sort({uid:-1})
　　{ "name" : "nfsnobody", "uid" : 65534 }
　　{ "name" : "lisi", "uid" : 1000 }
　　{ "name" : "polkitd", "uid" : 999 }
　　{ "name" : "libstoragemgmt", "uid" : 998 }
　　{ "name" : "colord", "uid" : 997 }
　　rs1:PRIMARY> db.user.find({shell:"/bin/bash"},{_id:0,name:1,uid:1})
　　{ "name" : "root", "uid" : 0 }
　　{ "name" : "lisi", "uid" : 1000 }
　　rs1:PRIMARY>
db.user.find({name:{$in:["root","mysql","haha"]}},{_id:0,name:1,uid:1})
　　{ "name" : "root", "uid" : 0 }
　　{ "name" : "mysql", "uid" : 27 }
　　rs1:PRIMARY>
db.user.find({name:{$nin:["root","mysql","haha"]}},{_id:0,name:1,uid:1}
)
　　rs1:PRIMARY>
db.user.find({$or:[{name:"root"},{uid:5}]},{_id:0,name:1,uid:1})
```

```
        { "name" : "root", "uid" : 0 }
        { "name" : "sync", "uid" : 5 }
    rs1:PRIMARY> db.user.find({name:/^r/},{_id:0,name:1,uid:1})
    rs1:PRIMARY> db.user.find({uid:{$lte:5}}, {_id:0,name :1,uid:1})
        { "name" : "root", "uid" : 0 }
        { "name" : "bin", "uid" : 1 }
        { "name" : "daemon", "uid" : 2 }
        { "name" : "adm", "uid" : 3 }
        { "name" : "lp", "uid" : 4 }
        { "name" : "sync", "uid" : 5 }
    rs1:PRIMARY>
    db.user.find({uid:{$gte:10,$lte:40}},{_id:0,name:1,uid:1})
        { "name" : "operator", "uid" : 11 }
        { "name" : "games", "uid" : 12 }
        { "name" : "ftp", "uid" : 14 }
        { "name" : "rpc", "uid" : 32 }
        { "name" : "rpcuser", "uid" : 29 }
        { "name" : "ntp", "uid" : 38 }
        { "name" : "mysql", "uid" : 27 }
    rs1:PRIMARY>
    db.user.find({haha:null},{_id:0,name:1,uid:1}).limit(3)
        { "name" : "root", "uid" : 0 }
        { "name" : "bin", "uid" : 1 }
        { "name" : "daemon", "uid" : 2 }
```

更新:

单行更新:db.集合名.update({条件},{修改的字段})

注:修改成功后,会把未修改的字段都删除了,只留下了修改字段,且只修改匹配条件的第 1 行

$set:条件匹配时,修改指定字段的值

db.user.update({条件},{$set:{key:values}})

$unset:删除与条件匹配文档的字段,value 对命令不产生影响,但是要输

db.集合名.update({条件},{$unset:{key:values}})

多行更新:

格式:db.user.update({条件},{$set:{修改的字段}},false,true)

示例:

```
    rs1:PRIMARY> db.user.update({name:"adm"},{$set:{pass:"123"}})
    rs1:PRIMARY> db.user.find({name:/^r/},{_id:0,name:1,pass:1})
        { "name" : "root", "pass" : "123" }
        { "name" : "rpc", "pass" : "x" }
        { "name" : "rtkit", "pass" : "x" }
        { "name" : "radvd", "pass" : "x" }
        { "name" : "rpcuser", "pass" : "x" }
    rs1:PRIMARY> db.user.update({name:/^r/},{$unset:{pass:"x"}})
    rs1:PRIMARY> db.user.find({name:/^r/},{_id:0,name:1,pass:1})
```

```
        { "name" : "root" }
        { "name" : "rpc", "pass" : "x" }
        { "name" : "rtkit", "pass" : "x" }
        { "name" : "radvd", "pass" : "x" }
        { "name" : "rpcuser", "pass" : "x" }
    rs1:PRIMARY>
db.user.update({pass:"x"},{$set:{pass:"A"}},false,true)
    rs1:PRIMARY> db.user.find({},{_id:0,name:1,pass:1})
```

自加减:条件匹配时,字段值自加或自减,n 为正整数表示自加,n 为负整数表示自减,只修改匹配条件的第 1 行

格式:db.集合名.update({条件},{$inc:{字段名:n}})

向数组中添加新元素:db.集合名.update({条件},{$push:{数组名:"值"}})

避免数组重复赋值:db.集合名.update({条件},{$addToSet:{数组名:"值"}})

从数组中删除一个元素:db.集合名.update({条件},{$pop:{数组名:n}})

n=1 时删除数组尾部元素,n=-1 时删除数组头部元素

删除数组中指定元素:db.集合名.update({条件},{$pull:{数组名:"值"}})

只对数组有效,删除匹配的所有值

示例:

```
    rs1:PRIMARY> db.user.update({name:/^r/},{$inc:{uid:3}})
    rs1:PRIMARY> db.user.find({name:/^r/},{_id:0,name:1,uid:1})
        { "name" : "root", "uid" : 3 }
        { "name" : "rpc", "uid" : 32 }
        { "name" : "rtkit", "uid" : 172 }
        { "name" : "radvd", "uid" : 75 }
        { "name" : "rpcuser", "uid" : 29 }
    rs1:PRIMARY> db.user.update({name:/^r/},{$inc:{uid:-3}},false,true)
    rs1:PRIMARY> db.user.find({name:/^r/},{_id:0,name:1,uid:1})
        { "name" : "root", "uid" : 0 }
        { "name" : "rpc", "uid" : 29 }
        { "name" : "rtkit", "uid" : 169 }
        { "name" : "radvd", "uid" : 72 }
        { "name" : "rpcuser", "uid" : 26 }
    rs1:PRIMARY> db.user.update({name:"root"},{$push:{age:5}})
    rs1:PRIMARY> db.user.find({name:"root"},{_id:0})
        { "name" : "root", "uid" : 3, "pid" : 0, "common" : "root", "home" :
    "/root", "shell" : "/bin/bash", "age" : [ 5 ] }
    rs1:PRIMARY> db.user.update({name:"root"},{$addToSet:{age:6}})
    rs1:PRIMARY> db.user.find({name:"root"},{_id:0})
        { "name" : "root", "uid" : 3, "pid" : 0, "common" : "root", "home" :
    "/root", "shell" : "/bin/bash", "age" : [ 5, 6 ] }
    rs1:PRIMARY> db.user.update({name:"root"},{$pop:{age:1}})
    rs1:PRIMARY> db.user.find({name:"root"},{_id:0})
        { "name" : "root", "uid" : 3, "pid" : 0, "common" : "root", "home" :
    "/root", "shell" : "/bin/bash", "age" : [ 5 ] }
```

```
rs1:PRIMARY> db.user.update({name:"root"},{$pop:{age:-1}})
rs1:PRIMARY> db.user.find({name:"root"},{_id:0})
    { "name" : "root", "uid" : 3, "pid" : 0, "common" : "root", "home" :
"/root", "shell" : "/bin/bash", "age" : [ ] }
rs1:PRIMARY> db.user.update({name:"root"},{$pull:{age:6}})
```

删除:

删除集合:db.集合名.drop()

删除文档:db.集合名.remove({条件})

不加条件则匹配所有条件,即删除所有文档

示例:

```
rs1:PRIMARY> db.user.remove({uid:{$lte:10}})
rs1:PRIMARY> db.user.remove({shell:"/bin/bash"})
rs1:PRIMARY> db.user.remove({})
rs1:PRIMARY> db.user.drop()
```

memcached 数据库:

简介:集中缓存数据库查询结果,减少数据库访问次数。存储的数据在断电后清除,保存不重要的或可以恢复的数据。

1.安装:[root@proxy ~]# yum -y install memcached

2.修改配置文件:

```
[root@proxy ~]# cat /etc/sysconfig/memcached
PORT="11211"                #端口号
USER="memcached"
MAXCONN="1024"          #最大并发量
CACHESIZE="64"          #内存大小,单位 M
OPTIONS=""
```

3.启动:[root@proxy ~]# systemctl start memcached #TCP\UDP 的 11211 端口都提供服务

4.使用 telnet 测试(选做)

```
[root@proxy ~]# yum -y install telnet
[root@proxy ~]# telnet 192.168.4.5 11211
    set key 0 180 3         #设置一个变量,压缩级别为 0,存储 180 秒,键为 3 个字符
    abc                     #输入 3 个字符
    STORED                  #系统提示成功,不提示代表失败
    get key                 #获取变量
    VALUE key 0 3           #输出结果
    abc
    END
    add key 0 10 2          #新建一个变量,若已存在则报错
    aa
    STORED
    replace key 0 180 3     #替换一个变量,若不存在则报错
    abc
```

```
STORED
append key 0 20 2        #追加值，存储时间实际无效
11
STORED
delete key               #删除一个变量
DELETED
stats                    #查看状态
....
flush_all                #清空所有变量
OK
quit                     #退出登陆
Connection closed by foreign host.[root@localhost ~]# cd
/var/lib/php/session/
```

5.使用 php 脚本测试 memcache

```
[root@proxy ~]# yum -y install php-pecl-memcache
[root@proxy ~]# systemctl restart php-fpm.service
```

使用 subversion 实现版本控制

subversion 是一个自由、开源的版本控制系统；允许将数据恢复到早期版本；检查修改历史；允许多人协作修改并跟踪；端口号：3690

subversion 架构：

客户端：命令行、图形界面

通信方式：本地、svn 服务器、web

版本库：版本控制的核心，记录每一次改变

1.安装(客户端和服务端装同一个)：[root@web1 ~]# yum -y install subversion

2.创建版本库：

```
[root@web1 ~]# mkdir /var/svn/
[root@web1 ~]# svnadmin create /var/svn/project
```

3.导入本地数据并初始化：[root@web1 ~]# svn import . file:///var/svn/project/ -m "Init Data"

4.修改配置文件:

```
[root@web1 ~]# ls /var/svn/project/conf/
    authz  passwd  svnserve.conf
[root@web1 ~]# vim /var/svn/project/conf/svnserve.conf      #去除注释及之后的空格
    19:anon-access = none                    #拒绝匿名
    20:auth-access = write                   #有效用户可写
    27:password-db = passwd              #设置密码文件
    34:authz-db = authz                  #设置 ACL
[root@web1 ~]# vim /var/svn/project/conf/passwd
    [users]
    harry = pass                     #用户名 = 密码
    sally = pass
[root@web1 ~]# vim /var/svn/project/conf/authz
```

```
[/]
harry = rw
sally= rw
* =                                    #设置其他人不能访问
```

5.启动服务：

```
[root@web1 ~]# svnserve -d  -r /var/svn/project      #-d 表示后台执行，-r 指定仓库路径
[root@web1 ~]# svnserve -d                           #表示共享所有仓库，客户端访问时需要加仓库的绝对路径
[root@web1 ~]# netstat -nutlp | grep svnserve
```

6.客户端访问：

username：用户名，第一次登陆后可选自动保存

password：密码，第一次登陆后可选自动保存

co：下载

code：新建 code 目录，并将代码下载到此目录

```
[root@web2 temp]# svn --username harry --password pass co svn://192.168.2.100/ code
```

7.客户端同步：

```
[root@web2 temp]# cd code/
[root@web2 code]# svn ci -m "shuoming"          #ci 表示上传,-m 设置修改原因
[root@web2 code]# svn update                    #将服务器上的数据同步到本地
[root@web2 code]# svn info svn://192.168.2.100  #查看版本库信息
[root@web2 code]# svn log svn://192.168.2.100   #查看版本库日志
[root@web2 code]# svn mkdir test                #增加目录并加入版本控制
[root@web2 code]# echo 1> test/test.sh
[root@web2 code]# svn add test/test.sh          #将新增的文件加入版本控制
[root@web2 code]# mkdir 11
[root@web2 code]# svn add 11/                   #将新增的目录加入版本控制
[root@web2 code]# svn rm 11/                    #删除目录并加入版本控制
[root@web2 code]# echo haha > test/test.sh
[root@web2 code]# svn diff                      #查看本地所有文件和服务端的差异
[root@web2 code]# svn diff test/test.sh         #查看指定文件和服务端的差异
[root@web2 code]# svn cat svn://192.168.2.100/test/test.sh  #查看服务端的内容
[root@web2 code]# svn revert test/test.sh       #还原文件
[root@web2 code]# rm 1.sh
[root@web2 code]# svn update                    #将服务端的内容同步到本地
```

[root@web2 code]# echo 1111 > 1.sh
[root@web2 code]# svn ci -m "x"
[root@web2 code]# svn merge -r7:6 1.sh #将文件从版本 7 恢复到版本 6

8.服务端备份、还原版本库：

[root@web1 ~]# svnadmin dump /var/svn/project > project.bak #服务端备份
[root@web1 ~]# svnadmin create /var/svn/pj1
[root@web1 ~]# svnadmin load /var/svn/pj1 < project.bak #服务端还原

9.多人协同工作:

a) 修改不同文件：分别提交后，更新版本号
b) 修改相同文件的不同行（未增删行）：后提交的需要先同步再提交
c) 修改相同文件的相同行：

[root@web2 code]# svn ci -m "s"
正在发送 1.sh
传输文件数据.svn: E160028: 提交失败(细节如下):
svn: E160028: 文件 “/1.sh” 已经过时
[root@web2 code]# svn update
正在升级 '.':
在 “/temp/code/1.sh” 中发现冲突。
选择: (p) 推迟, (df) 显示全部差异, (e) 编辑,
(mc) 我的版本, (tc) 他人的版本,
(s) 显示全部选项:

通常选 p 保留冲突，商议后：

[root@web2 code]# rm -f 1.sh.r14 1.sh.r15 1.sh.mine #删除 3 个临时文件
[root@web2 code]# svn ci -m "s" #将商议的结果上传

10.客户端是 windows 的情况：安装 TortoiseSVN，图形界面

集群：一组通过高速网络互联的服务器组,提供同一种服务,并以单一系统的模式加以管理

优势：付出较低成本的情况下获得性能,可靠性,灵活性方面的较高收益
核心技术：任务调度
目的：
提高性能：计算密集型应用
降低成本：比一台超级计算机便宜
可扩展性：增加集群节点
增强可靠性：多个节点完成相同功能
分类：
高性能计算机集群 HPC：通过集群模式并行的应用程序,提高运算速度
负载均衡集群 LB(Load Balance)：平摊客户端的访问
高可用集群 HA(High Available)：避免单点故障
服务器分类：
应用服务器：web 服务器,存程序源代码
数据库服务器：mysql

文件服务器：存储静态文件

高可用性集群：keepalived，RHCS(RedHat Cluster Suit)红帽集群套件

集群调度硬件：F5

集群调度软件：nginx,lvs,HAproxy

Nginx：

优点：

工作在第 7 层应用层,可以针对 http 做分流策略

正则表达式强大

安装、配置、测试简单,通过日志可以解决多数问题

并发量可以达到几万次

Nginx 还可以作为 Web 服务器使用

缺点：

仅支持 http、https、mail 协议,应用面小

监控检查仅通过端口,无法使用 url 检查

负载能力差

LVS：

优点：

负载能力强,工作在第 4 层传输层,对内存、CPU 消耗低

配置性低,没有太多可配置性,减少人为错误

应用面广,几乎可以为所有应用提供负载均衡

缺点：

不支持正则表达式,不能实现动静分离

如果网站架构庞大,LVS-DR 配置比较繁琐

本身没有对后端服务器的健康监测机制

HAProxy：

优点：

支持 session、cookie 功能

可以通过 url 进行健康检查

效率、负载均衡速度介于 Nginx 和 LVS 之间

HAProxy 支持 TCP,可以对 MySQL 进行负载均衡

调度算法丰富

缺点：

正则弱于 Nginx

日志依赖于 syslogd,不支持 apache 日志

网络规模较小时，网站所有页面的日访问量(PV:Page View)只有几百万时，可以使用 Nginx 和 HAProxy

当规模扩大，达到上千万 PV 时，可以使用 LVS

当规模继续扩大时，可以购买硬件 F5，但 F5 价格昂贵，几十万上百万

LVS(Linux Virtual Server)Linux 虚拟服务器：

优势：

使用集群技术和 Linux 操作系统实现一个高性能、高可用的服务器

可伸缩性(Scalability)

可靠性(Reliability)

可管理性(Manageability)

三层结构：

负载调度器(load balancer)：整个集群对外面的前端机,由多台负载调度器构成

服务器池(server pool)：真正执行客户请求的服务

共享存储(shared storage)：提供一个共享的存储区

术语：

调度服务器(Director Server)：将负载分发到真实服务器

真实服务器(Real Server)：提供应用服务的服务器

虚拟 IP(Virtual IP)：公布给用户访问的虚拟 IP

真实 IP(Real IP)：集群节点上的 IP

调度器连接节点服务器的 IP(Director IP)

工作模式：

NAT：通过网络地址转换实现的虚拟服务器,大并发访问时,调度器的性能会成为瓶颈

DR：直接使用路由技术实现虚拟服务器,节点服务器需要配置 VIP,直接由服务器返回数据

TUN：通过隧道方式实现虚拟服务器,很少用

Full NAT：大规模网络使用,解决的是 LVS 和 RS 跨 VLAN 的问题

调度算法：

轮询(Round Robin)：将客户端请求平均分发到服务器

加权轮询(Weighted Round Robin)：根据服务器权重值进行轮询调度

最少连接(Least Connections)：选择连接最少的服务器

加权最少连接(Weighted Least Connections)：根据服务器的权重,选择连接数最少的服务器

源地址散列(Source Hashing)：将请求的目标 IP 地址作为散列键,从静态分配的散列表找出对应的服务器

基于局部性的最少连接

带复制的基于局部性最少连接

目标地址散列

最短的期望的延迟

最少队列调度

部署论坛数据库集群：

1.在 4.1 上安装 mariadb,并授权：

```
MariaDB [(none)]> grant all on *.* to root@"%" identified by "123456" with grant option;
```

2.在 4.2 和 4.3 上安装网页：

```
[root@fzr ~]# for i in 2 3;do pssh -iH "192.168.4.$i" "yum -y install httpd php php-mysql";done
[root@fzr ~]# for i in 2 3;do pssh -iH "192.168.4.$i" "systemctl start httpd";done
```

3.在 4.2 上安装论坛：

```
[root@fzr ~]# scp Discuz_X3.3_SC_UTF8.zip 192.168.4.2:~
[root@fzr ~]# pssh -iH "192.168.4.2" "unzip Discuz_X3.3_SC_UTF8.zip"
```

```
[root@fzr ~]# pssh -iH "192.168.4.2" "cp -r ~/upload/ /var/www/html/bbs"
[root@fzr ~]# pssh -iH "192.168.4.2" "chown -R apache: /var/www/html/bbs"
```

4.进入网页安装论坛：

http://192.168.4.2/bbs

5.拷贝 4.2 的论坛到 4.3：由于使用相同的数据库,所以两边数据相同

```
[root@vh02 ~]# scp -r /var/www/html/bbs 192.168.4.3:/var/www/html/bbs
[root@vh03 ~]# chown -R apache: /var/www/html/bbs
```

部署 LVS-NAT 集群：

1.配置 4.4 为网关,开启路由转发,设置外网地址 201.1.1.4/24

```
[root@vh04 ~]# sysctl -a | grep ip_forward      #查看路由是否开启,默认已开启
[root@vh04 ~]# echo "net.ipv4.ip_forward = 1">> /etc/sysctl.conf
[root@localhost ~]# sysctl -p                   #读取 sysctl.conf 文件的内容
    net.ipv4.ip_forward = 1
```

2.安装 LVS：

```
[root@vh04 ~]# yum -y install ipvsadm
```

3.创建虚拟服务器,使用轮询算法：

格式：ipvsadm -A|E -t|u service-address [-s scheduler]

参数：

-A 添加虚拟服务

-E 编辑虚拟服务

-t TCP 服务

-u UDP 服务

-s 指定调度算法：rr、wrr、lc、wlc、lblc、lblcr、dh、sh、sed、nq

```
[root@vh04 ~]# ipvsadm -A -t 201.1.1.4:80 -s rr
```

4.将服务器添加到虚拟服务器中

格式：ipvsadm -a|e -t|u service-address -r server-address [options]

参数：

-a 添加真实服务器

-e 修改真实服务器

-r 指定真实的服务器地址

-w 为节点服务器设置权重,默认为 1

-m 伪装,NAT

-g 路由模式,DR,默认模式

```
[root@vh04 ~]# ipvsadm -a -t 201.1.1.4:80 -r 192.168.4.2 -m -w2
[root@vh04 ~]# ipvsadm -a -t 201.1.1.4:80 -r 192.168.4.3 -m
```

5.查看规则,并在 web 中验证：

格式：ipvsadm -l [options]

参数：

-n 以数字形式输出端口

```
[root@vh04 ~]# ipvsadm -ln
  TCP  201.1.1.4:80 rr
    -> 192.168.4.2:80               Masq    2      0          0
```

```
  -> 192.168.4.3:80               Masq    1      0          0
```

6.修改调度算法：

```
[root@vh04 ~]# ipvsadm -E -t 201.1.1.4:80 -s wrr
```

7.删除：

格式：

```
ipvsadm -D -t|u service-address
ipvsadm -d -t|u service-address -r server-address
```

参数：

-D 删除虚拟服务

-d 删除真实服务器

```
[root@vh04 ~]# ipvsadm -d -t 201.1.1.4:80 -r 192.168.4.2
[root@vh04 ~]# ipvsadm -D -t 201.1.1.4:80
```

部署 LVS-DR 集群

拓扑：LVS 调度器只有一个 IP 地址,和真实服务器在同一个网络

注意：

客户机直接访问 VIP,所以需要在调度器和 web 端都配置 vip。

为了地址不冲突，需要把 VIP 配置在调度器的 eth0 上，把 VIP 配置在 web 服务器的 lo 上

1.在调度器上配置 vip：

```
[root@vh04 ~]# cp /etc/sysconfig/network-scripts/ifcfg-eth0{,:0}
[root@vh04 ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth0:0
    TYPE=Ethernet
    BOOTPROTO=none
    NAME=eth0:0
    DEVICE=eth0:0
    ONBOOT=yes
    IPADDR=192.168.4.100
    PREFIX=24
[root@vh04 ~]# ifup eth0:0
```

2.在 web 服务器上配置 vip：

子网掩码设置为 255.255.255.255 是为了防止与调度器 IP 冲突

```
[root@fzr ~]# cat > ~/ifcfg-lo:0 << EOF
    DEVICE=lo:0
    IPADDR=192.168.4.100
    NETMASK=255.255.255.255
    NETWORK=192.168.4.100
    BROADCAST=192.168.4.100
    ONBOOT=yes
    NAME=lo:0
    EOF
[root@fzr ~]# for i in 2 3
    do
    scp ~/ifcfg-lo:0 192.168.4.$i:/etc/sysconfig/network-scripts/
    pssh -iH "192.168.4.$i" "ifup lo:0"
```

```
done
```

3.在 web 服务器上修改内核参数：

```
[root@fzr ~]# sysctl -a | grep arp_ign
[root@fzr ~]# sysctl -a | grep arp_ann
[root@fzr ~]# for i in 2 3
do
pssh -iH 192.168.4.$i "echo net.ipv4.conf.all.arp_ignore =
1 >>/etc/sysctl.conf"
pssh -iH 192.168.4.$i "echo net.ipv4.conf.lo.arp_ignore =
1 >>/etc/sysctl.conf"
pssh -iH 192.168.4.$i "echo net.ipv4.conf.all.arp_announce =
2 >>/etc/sysctl.conf"
pssh -iH 192.168.4.$i "echo net.ipv4.conf.lo.arp_announce =
2 >>/etc/sysctl.conf"
done
```

4.在调度服务器上配置规则：

```
[root@vh04 ~]# ipvsadm -A -t 192.168.4.100:80 -s wlc
[root@vh04 ~]# ipvsadm -a -t 192.168.4.100:80 -r 192.168.4.2
[root@vh04 ~]# ipvsadm -a -t 192.168.4.100:80 -r 192.168.4.3
```

5.启动服务,查看

```
[root@vh04 ~]# systemctl restart ipvsadm.service
```

如果报错,手动执行：

```
[root@vh04 ~]# ipvsadm-save -n > /etc/sysconfig/ipvsadm
```

HAProxy：

介绍：

免费、快速、可靠

适用于负载特别大的 web 站点,通常又需要会话保持或七层处理

提供高可用性、负载均衡以及基于 TCP 和 http 应用的代理

负载均衡器性能的指标：

session rate 会话率：每秒产生的会话数

session concurrency 并发会话数：服务器处理会话的时间越长，并发会话数越多

data rate 数据速率：以 Mbps 衡量

工作模式：

http：客户端请求被深度分析后再发往服务器，适用于 web

tcp：客户端与服务器之间建立会话，不检查第七层

health：健康状态检查，不使用

配置参数：

命令行：优先级最高

配置文件：

global：全局设置进程级别参数

defaults：设置默认参数

frontend：接受客户端侦听套接字集

backend：转发链接的服务器集

listen：把 frontend 和 backend 结合到一起的完整声明

HAProxy 配置：

1.安装：[root@vh04 ~]# yum -y install haproxy

2.修改配置文件：

[root@vh04 ~]# vim /etc/haproxy/haproxy.cfg

```
global
    #    local2.*     /var/log/haproxy.log  #默认日志设置
    log         127.0.0.1 local2             #记录所有日志信息
    chroot      /var/lib/haproxy             #程序根目录
    pidfile     /var/run/haproxy.pid         #pid 存放路径
    maxconn     4000                         #最大连接数
    user        haproxy
    group       haproxy
    daemon                                   #创建守护进程
    stats socket /var/lib/haproxy/stats      #状态套接字
defaults
    mode                    http             #默认工作模式为 http
    log                     global           #日志类型为全局日志
    option                  httplog          #http 日志格式
    option                  dontlognull      #不记录健康状态检查日志
    option http-server-close                 #每次请求完毕后主动关闭
http 通道
    option forwardfor  except 127.0.0.0/8    #后端服务器可以从 http
Header 中获得客户端 ip
    option                  redispatch       #服务器挂掉后强制定向到
其他健康服务器
    retries                 3                #3 次连接失败就认为服务
不可用
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s              #连接超时
    timeout client          1m               #客户端连接超时
    timeout server          1m               #服务器连接超时
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000             #最大连接数
listen stats
    bind    0.0.0.0:1080                     #监听端口
    stats   refresh 30s                      #统计页面自动刷新时间
    stats   uri /test                        #统计页面目录
    stats   realm   hello world              #统计页面密码框上提示文
本
    stats   auth    admin:123                #统计页面登录用户名和密
码
```

```
    stats   hide-version                    #隐藏统计页面上 HAProxy
的版本信息
listen websrv-rewrite 0.0.0.0:80
    cookie SERVERID rewrite
    balance roundrobin                     #算法设置为轮询，
leastconn 为最小连接
    server web1 192.168.4.2:80 cookie applinst1 check inter 2000 rise
2 fall 5
    server web2 192.168.4.3:80 cookie applinst2 check inter 2000 rise
2 fall 5
        #inter 检查时间，单位毫秒；rise 连续 n 次检测成功认为该机活跃；
        fall 失败 n 次认为该机不活跃
```

3.启动服务并查看

```
[root@vh04 ~]# systemctl start haproxy.service
[root@vh04 ~]# systemctl status haproxy.service
[root@vh04 ~]# ss -antup | grep 80
```

访问：http://192.168.4.4/

访问监控页面：http://192.168.4.4:1080/test

监控页面参数说明：

- Queue：队列，越小越好
  - Cur：当前队列
  - Max：最大队列
  - Limit：限制的最大队列数
- Session rate：会话速率
- Session：会话数
- Bytes：字节数
- Denied：拒绝
  - Req：请求
  - Resp：响应
- Errors：错误
- Warnings：警告
- Servers：服务
  - Status：状态
  - Wight：权重
  - Act：活跃

Keepalived 热备：

介绍：

实现了高可用集群

最初是为 LVS 设计的,专门监控各服务器节点的状态

后来加入了 VRRP(虚拟路由冗余协议)功能,防止单点故障

运行原理：

Keepalived 检测每个服务器节点状态

服务器节点异常或工作出现故障，Keepalived 将故障节点从集群系统中剔除

故障节点恢复后，Keepalived 再将其加入到集群系统中

Keepalived 高可用配置:

1.安装：

```
[root@fzr ~]# pssh -iH "192.168.4.2 192.168.4.3" "yum -y install keepalived.x86_64"
```

2.查看修改配置文件：

```
[root@vh02 ~]# vim /etc/keepalived/keepalived.conf
    global_defs {
       notification_email {                    #设置报警收件人邮箱
            ...
            }
       notification_email_from root@localhost  #设置发件人邮箱
       smtp_server 127.0.0.1                   #指定邮件服务器
       smtp_connect_timeout 30
       router_id LVS_DEVEL                     #路由器 id 号
       vrrp_skip_check_adv_addr
       #vrrp_strict                            #自动添加防火墙规则，注释后关闭
       vrrp_garp_interval 0
       vrrp_gna_interval 0
    }
    vrrp_instance VI_1 {
        state MASTER                           #服务器类型，MASTER 表示主服务器，BACKUP 表示从服务器
        interface eth0                         #网络接口
        virtual_router_id 2                    #虚拟路由器编号
        priority 200                           #路由器优先级
        advert_int 1                           #心跳消息发送频率,单位秒
        authentication {                       #共享用户名和密码
            auth_type PASS
            auth_pass 1111
        }
        virtual_ipaddress {192.168.4.200}      #设置 vip
    }
    删除后续配置，用于配置其他服务
```

3.在 3 号机配置从服务器：

```
[root@vh02 ~]# scp /etc/keepalived/keepalived.conf 192.168.4.3:/etc/keepalived/keepalived.conf
[root@vh03 ~]# vim /etc/keepalived/keepalived.conf  #修改 3 行
    vrrp_instance VI_1 {
        state BACKUP
        virtual_router_id 3
        priority 100}
```

4.启动服务：

[root@fzr ~]# pssh -iH "192.168.4.2 192.168.4.3" "systemctl start keepalived.service"

[root@fzr ~]# pssh -iH "192.168.4.2 192.168.4.3" "ip address show eth0"

访问：http://192.168.4.200/

Keepalived+LVS：

1.在4号机和5号机安装ipvsadm和keepalived

2.确定4号机和5号机上都没有lvs规则，因为规则将由Keepalived自动配置

3.确定4号机和5号机上都没有vip规则，因为将由Keepalived决定vip归属

4.将2号机和3号机配置成web服务器，按照DR的配置方式在lo上配置vip，修改内核参数

5.修改Keepalived配置文件:

[root@vh04 ~]# vim /etc/keepalived/keepalived.conf

```
global_defs {
   notification_email {
        root@localhost
        }
   notification_email_from root@localhost
   smtp_server 127.0.0.1
   smtp_connect_timeout 30
   router_id LVS_DEVEL
   vrrp_skip_check_adv_addr
   #vrrp_strict
   vrrp_garp_interval 0
   vrrp_gna_interval 0
}

vrrp_instance VI_1 {
    state MASTER
    interface eth0
    virtual_router_id 4
    priority 200
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    virtual_ipaddress {192.168.4.100}
}

virtual_server 192.168.4.100 80{     #设置vip,监听80端口
    delay_loop 6
    lb_algo rr                       #设置LVS调度算法
```

```
        lb_kind DR                          #设置 lvs 模式，区分大小写
        persistence_timeout 50              #同一客户端在设定的时间内调度给
同一台服务器，单位秒
        protocol TCP

        real_server 192.168.4.2 80 {
            weight 1                        #设置权重
            TCP_CHECK {                 #检查 TCP 连接,区分大小写
                connect_timeout 3           #连接超时次数
                nb_get_retry 3              #重试次数
                delay_before_retry 3}}  #两次重试间间隔时间,单位秒
        real_server 192.168.4.3 80 {
            weight 1
            TCP_CHECK {
                connect_timeout 3
                nb_get_retry 3
                delay_before_retry 3}}
    }
```

6.配置从属服务器：

```
[root@vh04 ~]# scp /etc/keepalived/keepalived.conf
192.168.4.5:/etc/keepalived/keepalived.conf
[root@vh05 ~]# vim /etc/keepalived/keepalived.conf
    state BACKUP
    virtual_router_id 5
    priority 100
```

7.启动服务：

```
[root@fzr ~]# pssh -iH "192.168.4.4 192.168.4.5" "systemctl restart
keepalived"
```

块存储、文件存储、对象存储的区别：

块存储:将裸磁盘空间整个映射给主机使用的。操作系统需要对挂载的裸硬盘进行分区、格式化后，才能使用，与平常主机内置的硬盘无差异。

典型设备：磁盘阵列、硬盘

优点：

1.通过 Raid 与 LVM 等手段对数据提供了保护；

2.可以将多块廉价的硬盘组合起来，组合为一个大容量的逻辑盘对外提供服务，提高了容量；

3.写入数据时，由于是多块磁盘组合出来的逻辑盘，可以并行写入，提升了读写效率；

4.通常块存储采用 SAN 架构组网，由于传输速度以及封装协议的原因，使得传输速度和读写效率得到提升

缺点：

1.采用 SAN 架构组网时，需要额外为主机购买光纤通道卡，还要购买光纤交换机，造价成本高；

2.主机之间数据无法共享，只能提供给一台服务器使用；

3.不利于不同操作系统主机间的数据共享：因为操作系统使用不同的文件系统，格式化后，不同的文件系统间的数据是共享不了的。

文件存储：可以实现文件的共享。主机可以直接对文件存储进行文件的上传和下载，文件管理功能完成对文件存储的格式化。

典型设备：FTP、NFS 服务器

优点：

1.造价低：不需要专门的机器和网络；

2.方便文件共享。

缺点：

1.读写速率低，传输速率慢：以太网，上传下载速度较慢；所有读写都要 1 台服务器里面的硬盘承受，速率比磁盘阵列慢了许多。

对象存储：综合块存储读写快，文件存储可共享的优点

典型设备：分布式存储服务器

对象存储将一个文件拆分为元数据（例如该文件的大小、修改时间、存储路径等）以及内容（数据）。

控制元数据节点叫元数据服务器（服务器+对象存储管理软件），里面主要负责存储对象的属性（主要是对象的数据被打散存放到了那几台分布式服务器中的信息）而其他负责存储数据的分布式服务器叫做 OSD，主要负责存储文件的数据部分。当用户访问对象，会先访问元数据服务器，元数据服务器只负责反馈对象存储在哪里 OSD。另一方面，对象存储软件是有专门的文件系统的，所以 OSD 对外又相当于文件服务器，那么就不存在共享方面的困难了，也解决了文件共享方面的问题。

分布式文件系统(Distributed File System)：

介绍：

文件系统管理的物理存储资源通过计算机网络与节点相连

基于客户机/服务器模式

常用软件：Lustre、Hadoop、FastDFS、Ceph、GlusterFS

Ceph：

介绍：

高可扩展性、高可用性、高性能

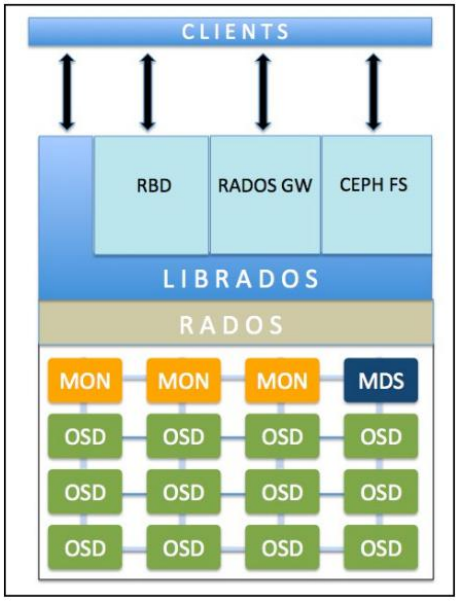
可以提供对象存储、块存储、文件系统存储

但是文件系统存储不是很成熟，很少用

可以提供 PB 级别的存储空间

软件定义存储(Software Defined Storage)

组件：



OSD：对象存储设备，唯一真正进行数据存储的组件，通常一个 OSD 进程绑定一块物理硬盘

PG(Placement Group)归置组：是一致性哈希中的虚拟节点，维护了一部分数据并且是数据迁移和改变的最小单位

MON：Monitor 集群监控组件，跟踪整个集群的健康状态，为每个 Ceph 组件维护一个映射表，进程数为＞1 的奇数

MDS：元数据服务器，为文件系统存储提供元数据，其他存储不需要。元数据用于描述数据

RADOS：可靠自主的分布式对象存储，是 ceph 存储的基础，保证各种数据都是对象形式，保持 ceph 的一致性

RBD：为客户端提供块存储接口

RADOS GW：为客户端提供对象存储接口

CEPH FS：为客户端提供文件系统存储接口

Client：ceph 客户端

对象：

对象是系统中数据存储的基本单位，一个对象实际上就是文件的数据和一组属性信息(Meta Data)的组合

每个对象都有一个唯一的对象标识(OID)，还有数据，元数据和其他属性

对象存储：

键值存储，通过接口指令，例如：GET,PUT,DEL 等，向存储服务器上传下载数据

对象存储中所有数据都被认为是一个对象,所以,任何数据都可以存入对象存储服务器

RGW(Rados Gateway)是 Ceph 对象存储网关,用于向客户端应用呈现存储界面,提供 RESTful API 访问接口

部署 ceph 集群：

1.准备工作：YUM 源、hosts 配置域名解析、ssh 密钥、NTP 时间同步，存储磁盘

```
[root@fzr ~]# tail -1 /etc/fstab
    /root/rhcs2.0-rhosp9-20161113-x86_64.iso /var/ftp/ceph iso9660
loop,ro 0 0
[root@fzr ~]# cat init.sh
    #!/bin/bash
```

```
for i in 1 2 3 4 10
do
expect << EOF
spawn ssh-copy-id 192.168.4.$i
expect "password" {send "123456\n"}
expect "#" {send "\n"}
EOF
expect << EOF
spawn ssh 192.168.4.$i
expect "#" {send "cd /etc/yum.repos.d/\n"}
expect "#" {send "wget ftp://192.168.4.254/rhel.repo\n"}
expect "#" {send "\n"}
EOF
scp /var/ftp/ceph.repo 192.168.4.$i: /etc/yum.repos.d/
pssh -iH 192.168.4.$i "yum repolist"
pssh -iH 192.168.4.$i "sed -i '35c StrictHostKeyChecking no'
/etc/ssh/ssh_config"
scp .ssh/id_rsa 192.168.4.$i:~/.ssh/
done
[root@fzr ~]# cat addhost.sh
#!/bin/bash
cat >> /etc/hosts << EOF
192.168.4.1 node1
192.168.4.2 node2
192.168.4.3 node3
192.168.4.4 node4
192.168.4.10 client
EOF
[root@fzr ~]# bash addhost.sh
[root@fzr ~]# for i in 1 2 3 4 10;do scp addhost.sh 192.168.4.$i:~ ; pssh
-iH 192.168.4.$i "bash addhost.sh" ; done
[root@fzr ~]# pssh -ih hosts "sed -i '3c server 192.168.4.10 iburst'
/etc/chrony.conf "
[root@client ~]# vim /etc/chrony.conf
[root@fzr ~]# pssh -ih hosts "systemctl restart chronyd.service"
```

2.将 node1 作为部署主机，安装软件：

该软件可以用于同步配置文件，远程安装软件

scp /etc/ceph/{ceph.client.admin.keyring,ceph.conf}等同于 ceph-deploy admin

yum -y install 等同于 ceph-deploy install

```
[root@node1 ~]# yum -y install ceph-deploy
```

3.创建目录，用于存放密钥与配置文件

```
[root@node1 ~]# mkdir ceph-test
[root@node1 ~]# cd ceph-test
```

4.新建 ceph 集群配置，所有节点都为 MON

```
[root@node1 ceph-test]# ceph-deploy new node1 node2 node3
```

5.给所有节点安装 ceph 软件包

```
[root@node1 ceph-test]# ceph-deploy install node1 node2 node3
```

6.初始化所有节点的 mon 服务

```
[root@node1 ceph-test]# ceph-deploy mon create-initial
```

7.给所有节点的 vdb 磁盘分区，用来做存储服务器的日志盘

```
[root@fzr ~]# cat disk.sh
    echo "n


    +150G

    n




    w
    y
    " | gdisk /dev/vdb
[root@fzr ~]# for i in 1 2 3 ;do scp disk.sh 192.168.4.$i:~ ;pssh -iH
192.168.4.$i "bash ~/disk.sh";done
[root@fzr ~]# for i in 1 2 3;do pssh -iH 192.168.4.$i "chown ceph: /dev/vdb?";
done
```

8.初始化清空磁盘数据

```
[root@node1 ceph-test]# for i in 1 2 3;do ceph-deploy disk zap node${i}:vdc
node${i}:vdd; done
```

9.创建 OSD 存储空间：一个存储设备对应一个日志设备，日志通常需要 SSD，不需要很大

```
[root@node1 ceph-test]# for i in 1 2 3;do ceph-deploy osd create
node$i:vdc:/dev/vdb1 node$i:vdd:/dev/vdb2;done
```

10.查看集群状态：三台主机的状态是一致的

```
[root@node1 ~]# ceph -s
    health HEALTH_OK
```

11.配置 udev，使得磁盘属主属组在 reboot 后，仍然是 ceph

```
[root@node1 ~]# vim /etc/udev/rules.d/90-mydisk.rules
    ACTION=="add", KERNEL=="vdb[12]", OWNER="ceph", GROUP="ceph"
```

12.排错：

如果创建 OSD 存储空间时，提示 run gatherkeys，重新导入密钥：

```
[root@node1 ceph-test]# ceph-deploy gatherkeys node1 node2 node3
```

如果集群状态为 HEALTH_ERR，重启服务：

```
[root@node1 ceph-test]# systemctl restart ceph\*.service ceph\*.target
```

如果集群状态为 HEALTH_WARN，查看错误提示：

如果是时间同步问题，则检查时间同步或者重启主机

如果是 monmap 的问题，检查是否所有主机都在集群中

如果是 osdmap 的问题，检查是否有 miss 的硬盘

CephX：是整个 Ceph 系统的用户名和密码

默认用户 client.admin 中 client 是 cephx 认证模式的 type，admin 是 ID。在 Ceph 配置中用这种写法，客户端挂载 CephFS 的时候，用 ID 就行了，不用写 TYPE。

主配置文件中：cephx 就是 ceph 的认证方式_x000B_ 集群认证方式：auth_cluster_required = cephx _x000B_ 服务端认证方式：auth_service_required = cephx_x000B_ 客户端认证方式:auth_client_required = cephx

更多查看：https://blog.csdn.net/hxpjava1/article/details/80164158

Ceph 块存储：RADOS 块设备(RADOS block device:RBD)

介绍：

RBD 驱动已经集成在了 Linux 内核中

RBD 提供了企业功能,如快照、COW 克隆

RBD 支持内存缓存

1.查看存储池：Ceph 默认有一个存储池. 名为 rbd

```
[root@node1 ~]# ceph osd lspools
```

2.创建镜像：给客户端提供块设备

```
[root@node1 ~]# rbd create image-test --image-feature layering --size 10G
    #在默认池中创建
[root@node1 ~]# rbd create rbd/image --image-feature layering --size 10G
    #指定在 rbd 池中创建
```

3.查看镜像列表

```
[root@node1 ~]# rbd list
```

4.查看单个镜像详情

```
[root@node1 ~]# rbd info image
```

5.调整镜像大小：

实际工作中不允许缩小容量，所以强制缩小时需要加--allow-shrink 选项

缩小容量：[root@node1 ~]# rbd resize --size 8G image --allow-shrink

扩容容量：[root@node1 ~]# rbd resize --size 1T image

6.客户端装包

```
[root@client ~]# yum -y install ceph-common
```

7.客户端获取服务端的配置文件和连接密钥，也可以服务端传过去

```
[root@client ~]# scp 192.168.4.1:/etc/ceph/ceph.conf /etc/ceph/
[root@client ~]# scp 192.168.4.1:/etc/ceph/ceph.client.admin.keyring
/etc/ceph/
```

8.客户端映射镜像到本地磁盘

```
[root@client ~]# rbd map image
    /dev/rbd0
[root@client ~]# lsblk
    rbd0          251:0    0    1T  0 disk
[root@client ~]# rbd showmapped
    0  rbd  image      -    /dev/rbd0
```

9.格式化并挂载

```
[root@client ~]# mkfs.xfs /dev/rbd0
[root@client ~]# mount /dev/rbd0 /mnt
[root@client ~]# df -h
    /dev/rbd0              1.0T   34M  1.0T    1% /mnt
```

Ceph 镜像快照：快照使用 COW 技术,对大数据的快照速度很快

1.服务端创建镜像快照

```
[root@node1 ~]# rbd snap create image --snap image-snap-test
```

2.服务端查看镜像快照

```
[root@node1 ~]# rbd snap ls image
```

3.客户端模拟删除

```
[root@client mnt]# rm -f *
```

4.客户端解除挂载

```
[root@client ~]# umount /mnt/
```

5.服务端还原快照

```
[root@node1 ~]# rbd snap rollback image --snap image-snap-test
```

6.客户端重新挂载

```
[root@client ~]# mount /dev/rbd0 /mnt/
```

通过快照创建镜像

1.给镜像设置保护，被保护的快照无法删除

```
[root@client ~]# rbd snap protect image --snap image-snap-test
```

2.使用快照克隆一个新的镜像，镜像名为：image-clone

```
[root@client ~]# rbd clone image --snap image-snap-test image-clone
--image-feature layering
```

3.查看新镜像信息，得到父镜像

```
[root@client ~]# rbd info image-clone
    parent: rbd/image@image-snap-test
```

4.如果希望克隆镜像可以独立工作,就需要将父快照中的数据,全部拷贝一份,但非常耗时

PS：在独立克隆镜像和删除任务进行过程中，有时候会提示：Transport endpoint is not connected 服务不可用，是因为虚拟机性能的问题

```
[root@client ~]# rbd flatten image-clone
```

删除映射：

1.查看挂载

```
[root@client ~]# lsblk
    rbd0          251:0    0    1T  0 disk /mnt
```

2.删除挂载

```
[root@client ~]# umount /mnt
```

3.查看映射

```
[root@client ~]# rbd showmapped
    0  rbd  image      -    /dev/rbd0
```

4.取消映射

```
[root@client ~]# rbd unmap image
```

删除快照、镜像：

1.查看镜像信息

```
[root@client ~]# rbd snap ls image
    4 image-snap-test 1024 GB
```

2.如果快照有保护则取消保护，若还有子镜像需要先删除子镜像

```
[root@client ~]# rbd rm image-clone
[root@client ~]# rbd snap unprotect image --snap image-snap-test
```

3.删除快照

```
[root@client ~]# rbd snap rm image --snap image-snap-test
```

4.删除镜像

```
[root@client ~]# rbd unmap image
[root@client ~]# rbd rm image
```

安装 KVM 虚拟机，使用 Ceph 提供磁盘存储

1.配置真实机为客户端：装包，拷贝配置文件和连接密钥

2.为 KVM 创建一个名为 vm1 的镜像

```
[root@fzr ~]# rbd create vm1 --image-feature layering --size 1T
```

3.查看镜像

```
[root@fzr ~]# rbd info vm1
[root@fzr ~]# qemu-img info rbd:rbd/vm1
```

4.新建虚拟机 rhel7.4，虚拟机创建完成后，会自动启动，强制关机

5.KVM 虚拟机使用 ceph 时，需要认证，创建通行证：

Ceph 默认开启用户认证,客户端需要账户才可以访问

默认账户名称为 client.admin，key＝账户的密钥，记载在 ceph.client.admin.keyring 中

可以使用 ceph auth 添加新账户

```
[root@fzr ~]# cat /etc/ceph/ceph.client.admin.keyring
    [client.admin]
        key = AQBJCCpbmOXNJBAAo4tk+hmWBn2vgGRo+CW50g==
```

6.创建临时文件，并使用该文件生成通行证信息

```
[root@fzr ~]# vim /tmp/secret.xml
    <secret ephemeral='no'  private='no'>
        <usage type='ceph'>
            <name>client.admin  secret</name>
        </usage>
    </secret>
[root@fzr ~]# virsh secret-define --file /tmp/secret.xml
    生成 secret ff0129f7-cdcd-4667-bf84-c2958859f55e
[root@fzr ~]# virsh secret-list
    ff0129f7-cdcd-4667-bf84-c2958859f55e  ceph client.admin secret
```

7.将 KVM 的通行证和 Ceph 的账户进行关联

secret 指定 secret 的 UUID

base64 指定 ceph 的 client.admin 账户的密码

```
[root@fzr ~]# virsh secret-set-value --secret
ff0129f7-cdcd-4667-bf84-c2958859f55e --base64
AQBJCCpbmOXNJBAAo4tk+hmWBn2vgGRo+CW50g==
```

8.导出 rhel7.4 的 xml 文件并修改

```
[root@fzr ~]# virsh dumpxml rhel7.4 > /tmp/vm.xml
[root@fzr ~]# vim /tmp/vm.xml
    <disk type='network' device='disk'>
        <driver name='qemu' type='raw'/>
        <auth username='admin'>
        <secret type='ceph'
    uuid='ff0129f7-cdcd-4667-bf84-c2958859f55e'/>
        </auth>
        <source protocol='rbd' name='/rbd/vm1'>
            <host name='192.168.4.1' port='6789'/>
        </source>
        <target dev='vda' bus='virtio'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x07'
    function='0x0'/>
    </disk>
```

修改硬盘类型为网络

添加用户字段,添加用户名为 admin 密码类型为 ceph,以及关联好的通行证的 uuid

修改 source 字段，该字段提供源路径，制定了 ceph 主机、端口、镜像池和镜像

target 不变，说明了获取源后，用于虚拟机的 vda 盘

9.删除虚拟机 rhel7.4 并使用修改过的配置文件重新生成虚拟机

```
[root@fzr ~]# virsh define /tmp/vm.xml
```

使用 ceph FS 提供分布式文件系统

1.新增 4 号机为 MDS(元数据服务器)，装包

```
[root@node1 ~]# ceph-deploy install --mds node4
```

2.在管理主机上配置 4 号机的 MDS 并同步配置文件和 key

```
[root@node1 ceph-test]# ceph-deploy admin node4
[root@node1 ceph-test]# ceph-deploy mds create node4
```

3.创建存储池：需要至少 2 个池，一个池用于存储数据，一个池用于存储元数据

创建 2 个存储池，对应 128 个 PG

PG 有多个副本，分布在多个 OSD，其中一个为主，其余为辅，当其中一个 PG 损坏后，会在其他 OSD 中再生成一个 PG

```
[root@node1 ~]# ceph osd pool create data_test 128
[root@node1 ~]# ceph osd pool create metadata_test 128
```

4.创建文件系统：

注：只能创建 1 个文件系统. 先写存储元数据池，再写存储数据池

```
[root@node1 ~]# ceph fs new test metadata_test data_test
    new fs with metadata pool 2 and data pool 1
```

5.查看文件系统

```
[root@node1 ~]# ceph fs ls
    name: test, metadata pool: metadata_test, data pools: [data_test ]
```

6.查看 MDS 状态

```
[root@node1 ~]# ceph mds stat
    1 up:standby
```

7.客户端挂载

注意：

文件系统类型为 ceph

挂载节点为管理节点的 IP

用户名和密钥从/etc/ceph/ceph.client.admin.keyring 获取

```
[root@client ~]# mount -t ceph 192.168.4.1:6789:/ /mnt -o
name=admin,secret=AQBJCCpbmOXNJBAAo4tk+hmWBn2vgGRo+CW50g==
[root@client ~]# cat /etc/fstab
192.168.4.1:6789:/      /mnt    ceph
name=admin,secret=AQBJCCpbmOXNJBAAo4tk+hmWBn2vgGRo+CW50g==  0   0
[root@client ~]# df -h
192.168.4.1:6789:/     1.8T  8.9G  1.8T    1% /mnt
```

Ceph 对象存储：

1.新增 5 号机，用于 RGW，装包

```
[root@node1 ~]# ceph-deploy install --rgw node5
```

2.同步配置文件和密钥：

```
[root@node1 ceph-test]# ceph-deploy admin node5
```

3.启动 RGW 服务

```
[root@node1 ceph-test]# ceph-deploy rgw create node5
```

4.在 5 号机查看结果

```
[root@node5 ~]# ps aux | grep radosgw
[root@node5 ~]# systemctl status ceph-radosgw@*
```

5.由于 RGW 内置了一个 web 服务，默认服务端口为 7480，修改为 80 端口更方便使用

```
[root@node5 ~]# cat >> /etc/ceph/ceph.conf << EOF
[client.rgw.node5]
host=node5
rgw_frontends="civetweb port=80"
EOF
```

6.重启服务

```
[root@node5 ~]# systemctl restart ceph-radosgw@*
```

7.客户端测试

```
[root@fzr ~]# curl 192.168.4.5
```

使用亚马逊提供的命令行客户端 s3

1.创建新的账户：

--display-name 指定别名

```
[root@node1 ~]# radosgw-admin user create --uid="test"
--display-name="first"
[root@node1 ~]# radosgw-admin user info --uid=test
"keys": [
    {
        "user": "test",
        "access_key": "F5ZTFS72L8VD5GK7WZO1",
        "secret_key": "c8my1bC1N9nRKveHXXDPIPMwJ382Jw4Zh2lwr37e"
    }
```

```
    ],
```

新建账户的文件存储在/var/lib/ceph

PS：删除账户

```
[root@node1 ~]# radosgw-admin user rm --uid="test"
```

2.装包

```
[root@fzr ~]# scp s3cmd-2.0.1-1.el7.noarch.rpm 192.168.4.10:~
[root@client ~]# yum -y install s3cmd-2.0.1-1.el7.noarch.rpm
```

3.初始化配置：

```
[root@client ~]# s3cmd --configure
    Access Key: F5ZTFS72L8VD5GK7WZO1
    Secret Key: c8my1bC1N9nRKveHXXDPIPMwJ382Jw4Zh2lwr37e
    Default Region [US]:回车
    S3 Endpoint [s3.amazonaws.com]: 192.168.4.5:80
    azonaws.com]: %(bucket)s.192.168.4.5:80
    Encryption password:回车
    Path to GPG program [/usr/bin/gpg]:回车
    Use HTTPS protocol [Yes]: no
    HTTP Proxy server name:回车
    New settings:
      Access Key: F5ZTFS72L8VD5GK7WZO1
      Secret Key: c8my1bC1N9nRKveHXXDPIPMwJ382Jw4Zh2lwr37e
      Default Region: US
      S3 Endpoint: 192.168.4.5:80
      DNS-style bucket+hostname:port template for accessing a
bucket: %(bucket)s.192.168.4.5:80
      Encryption password:
      Path to GPG program: /usr/bin/gpg
      Use HTTPS protocol: False
      HTTP Proxy server name:
      HTTP Proxy server port: 0
    Test access with supplied credentials? [Y/n] y
    Save settings? [y/N] y
        Configuration saved to '/root/.s3cfg'          #配置文件保存地
    址
```

4.测试：

创建存储数据的存储器(make bucket)：[root@client ~]# s3cmd mb s3://test_haha

查看目录：[root@client ~]# s3cmd ls

上传功能：[root@client ~]# s3cmd put /etc/passwd s3://test_haha

查看目录内容：[root@client ~]# s3cmd ls s3://test_haha

下载功能：[root@client ~]# s3cmd get s3://test_haha/passwd /tmp

删除功能：[root@client ~]# s3cmd del s3://test_haha/passwd

FastDFS:

开源的轻量级分布式文件系统,主要解决了海量数据存储问题,特别适合以中小文件(建议范围：4KB~500MB)为载体的在线服务。

组件：

跟踪服务器(tracker server)

存储服务器(storage server)

客户端(client)

存储服务器:

以组或卷(group/volume)为单位组织，一个组内包含多台存储服务器，数据互为备份

注：存储空间以组内容量最小的存储为准，所以建议同组内的存储服务器尽量配置相同，以免造成存储空间的浪费。

功能：

应用隔离：将不同应用数据存到不同的组中

负载均衡：将应用分配到不同的组中

副本数定制：同组内存储服务器的数量即为副本数

缺点：

组的容量受单机存储容量的限制

当组内有机器坏掉时，数据恢复只能依赖组内其他机器，恢复时间长

跟踪服务器：

FastDFS 的协调者

负责管理所有的 storage server 和 group

每个 storage 在启动后会连接 Tracker，告知自己所属的 group 等信息，并保持周期性的心跳

tracker 根据 storage 的心跳信息，建立 group-->[storage server list]的映射表

Tracker 需要管理的元信息很少，会全部存储在内存中

tracker 上的元信息都是由 storage 汇报的信息生成的，本身不需要持久化任何数据

多台 tracker 可组成集群,集群中每个 tracker 之间是完全对等的,所有的 tracker 都接受 stroage 的心跳信息和读写请求

上传文件：tracker.conf

1. 客户端在上传文件时任意选择一个 trakcer。当 tracker 接收到上传文件的请求时，会为该文件分配一个可以存储该文件的 group

选择 group 的规则：store_lookup

0：Round robin：所有的 group 间轮询

1：Specified group：指定某一个确定的 group

2:Load balance：剩余存储空间多多 group 优先

2. 当选定 group 后，tracker 会在 group 内选择一个 storage server 给客户端

选择 storage 的规则：store_server

0:Round robin：在 group 内的所有 storage 间轮询

1:First server ordered by ip：按 ip 排序

2:First server ordered by priority：按优先级排序(优先级在 storage 上配置)

3. 分配好 storage server 后，客户端将向 storage 发送写文件请求，storage 将会

为文件分配一个数据存储目录

分配数据存储目录规则：store_path

0:Round robin：多个存储目录间轮询

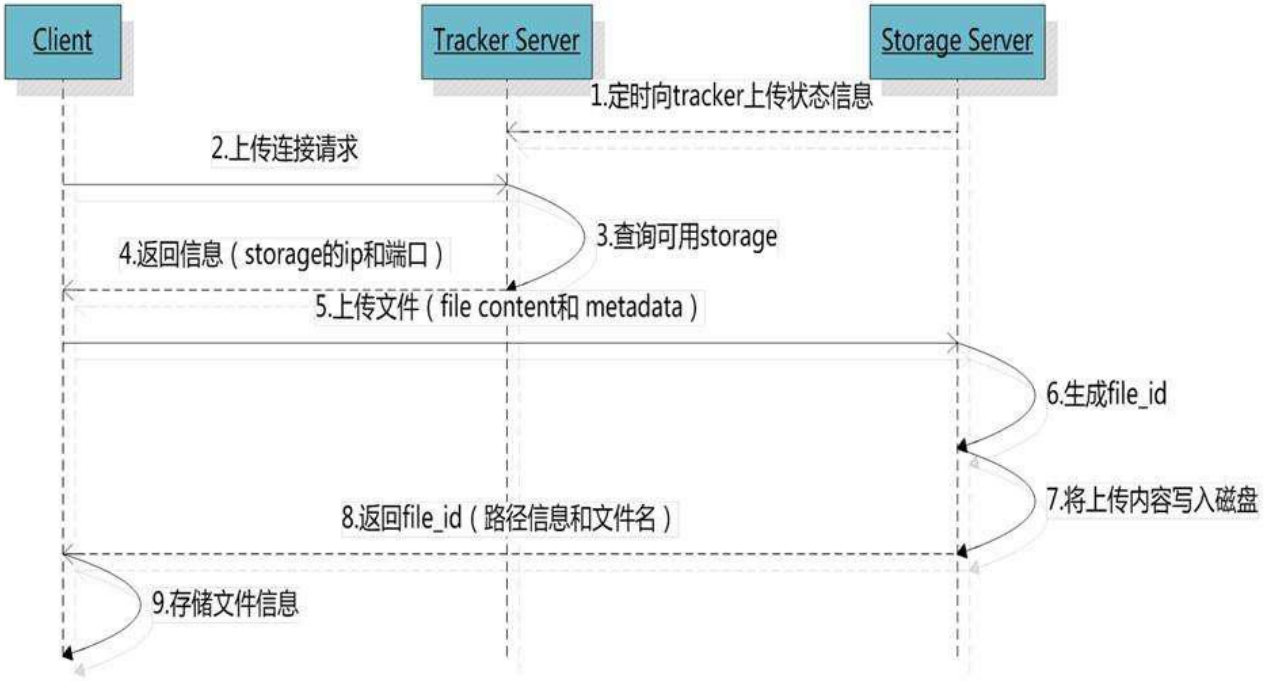
2:剩余存储空间最多的优先

4. 选定存储目录后，将文件路由到一个二级目录文件，存储到该目录下的某个子目录后，即认为该文件存储成功，会为该文件生成一个文件名

文件名命名规则：group、存储目录、两级子目录、fileid、文件后缀名(由客户端指定，主要用于区分文件类型)拼接而成

文件同步：

客户端将文件写入 group 内任一个 storage server 即认为写文件成功，之后会由后台



线程将文件同步至同 group 内其他的 storage server

每个 storage 写文件后，同时会写一份 binlog，binlog 里不包含文件数据，只包含文件名等元信息，这份 binlog 用于后台同步，storage 会记录向 group 内其他 storage 同步的进度，以便重启后能继续上次的进度继续同步，进度以时间戳的方式进行记录。storage 的同步进度会作为元数据的一部分汇报到 tracker 上，tracke 在选择读 storage 的时候会参考同步进度。

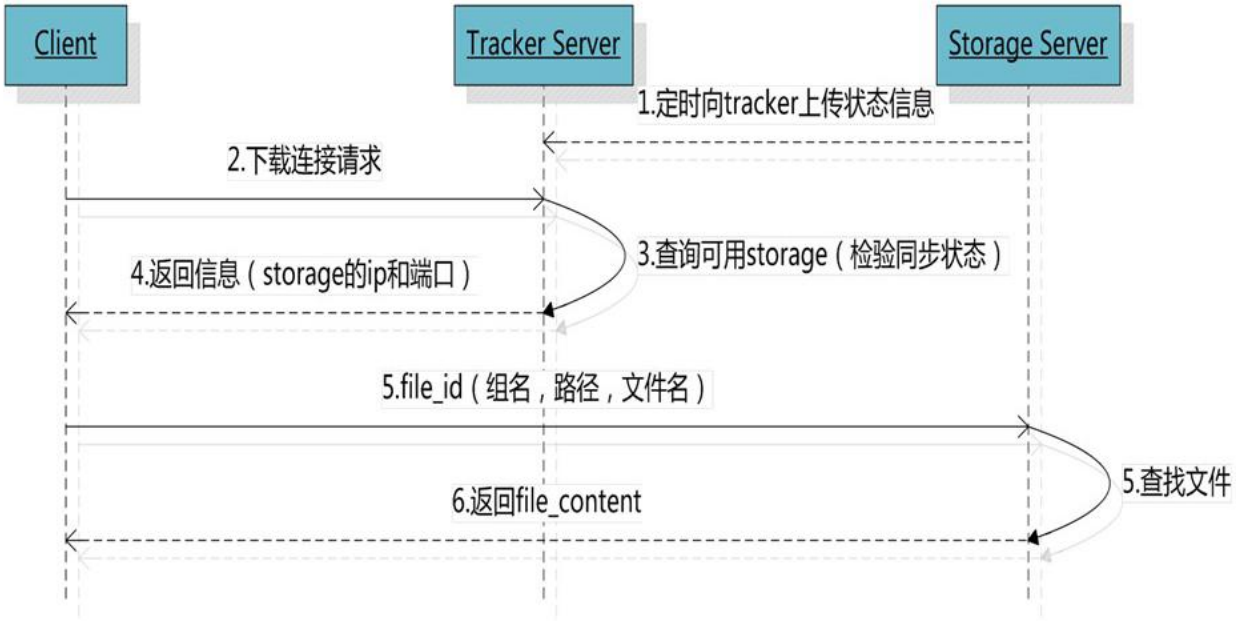
下载文件：

1. tracker 发送 download 请求给某个 tracker，必须带上文件名信息
2. tracke 从文件名中解析出文件的 group、大小、创建时间等信息
3. 为该请求选择一个 storage 用来服务读请求。

注：由于 group 内的文件同步在后台异步进行的，所以有可能出现在读的时候，文件还没有同步到某些 storage server 上，为了尽量避免访问到没完成的同步，tracker 按照一定规则选择 group 内可读的 storage。

规则：

1.源头 storage 只要存活着，肯定包含这个文件，源头的地址被编码在文件名



中。

2. 文件创建时间戳==storage 被同步到的时间戳且(当前时间-文件创建时间戳) > 文件同步最大时间(如 5 分钟)。文件创建后，认为经过最大同步时间后，肯定已经同步到其他 storage 了。

3. 文件创建时间戳 < storage 被同步到的时间戳。同步时间戳之前的文件确定已经同步了

4.(当前时间-文件创建时间戳) > 同步延迟阀值（如一天）。超过同步延迟阈值时间，认为文件肯定已经同步了。

搭建 FastDFS：

下载网址：

FastDFS：https://github.com/happyfish100/fastdfs/tree/master

libfastcommon：https://github.com/happyfish100/libfastcommon

注：目录下都有一个 INSTALL，是安装说明

1. 安装 libfastcommon：包含了 FastDFS 运行所需要的一些基础库

```
[root@client ~]# yum -y install gcc gcc-c++ libevent-devel git
[root@client ~]# git clone
https://github.com/happyfish100/libfastcommon.git
[root@client ~]# cd libfastcommon
[root@client libfastcommon]# git checkout V1.0.38
[root@client libfastcommon]# ./make.sh
[root@client libfastcommon]# ./make.sh install
```

2. 安装 FastDFS

```
[root@client ~]# git clone https://github.com/happyfish100/fastdfs.git
[root@client ~]# cd fastdfs/
[root@client fastdfs]# git checkout V5.11
[root@client fastdfs]# ./make.sh
[root@client fastdfs]# ./make.sh install
```

3. 修改跟踪服务器配置文件

```
[root@client ~]# cp /etc/fdfs/tracker.conf.sample /etc/fdfs/tracker.conf
[root@client ~]# vim /etc/fdfs/tracker.conf
    11：port=22122
    22：base_path=/fastdfs
    49：store_lookup=2
```

54：store_group=group2

60：store_server=0

65：store_path=0

260：http.server_port=80

4. 启动跟踪服务

```
[root@client ~]# mkdir /fastdfs
[root@client ~]# fdfs_trackerd /etc/fdfs/tracker.conf start
```

或

```
[root@client ~]# service fdfs_trackerd start
```

注：

早期版本的 libfastcommon 将 libfastcommon.so 放在/lib，后来移到了/lib64。可以通过 ldd /usr/bin/fdfs_trackerd 查看。

如果安装过早期版本，只需将/local 下的动态库文件删除，指向位置就会自动更新

```
[root@client ~]# rm -rf /usr/local/lib/libfastcommon.so.1
```

5. /fastdfs 目录说明

data：存数据

logs：存日志

6. 修改存储服务器配置文件

```
[root@client ~]# cp /etc/fdfs/storage.conf.sample /etc/fdfs/storage.conf
[root@client ~]# vim /etc/fdfs/storage.conf
```

11：group_name=group1　　#定义组名，默认为 group1

41：base_path=/fastdfs

109：store_path0=/fastdfs

118：tracker_server=172.16.186.96:22122

284：http.server_port=8888

注：

单机测试 IP 为 172.16.186.96

如果有多个挂载磁盘则定义多个 store_path，例如 110 行

有多个跟踪服务器则配置多行 tracker，一行只能指定一个跟踪服务器

tracker_server 不允许指定为 127.0.0.1

7. 启动存储服务

```
[root@client ~]# fdfs_storaged /etc/fdfs/storage.conf start
```

或

```
[root@client ~]# service fdfs_storaged start
```

8. 使用 FastDFS 自带 client 测试

```
[root@client ~]# cp /etc/fdfs/client.conf.sample /etc/fdfs/client.conf
[root@client ~]# vim /etc/fdfs/client.conf
```

10:base_path=/fastdfs

14:tracker_server=172.16.186.96:22122

```
[root@client ~]# fdfs_test /etc/fdfs/client.conf upload test.txt
```

group_name=group1,

remote_filename=M00/00/00/rBC6YFvbxKSAfRNaAAAABelEJbQ726.txt

example file url:

http://172.16.186.96/group1/M00/00/00/rBC6YFvbxKSAfRNaAAAABelEJbQ726.txt

example file url: http://172.16.186.96/group1/M00/00/00/rBC6YFvbxKSAfRNaAAAABelEJbQ726_big.txt

FastDFS +nginx：需要完全前面 7 步

下载地址：https://github.com/happyfish100/fastdfs-nginx-module

1. 下载

```
[root@client ~]# git clone https://github.com/happyfish100/fastdfs-nginx-module.git
[root@client ~]# cd fastdfs-nginx-module/
[root@client fastdfs-nginx-module]# git checkout V1.20
```

2. 修改配置文件

```
[root@client fastdfs-nginx-module]# cd src/
[root@client src]# vim mod_fastdfs.conf
    10：base_path=/fastdfs
    40：tracker_server=172.16.186.96:22122
    47：group_name=group1              #同 storage.conf
    53：url_have_group_name = true  #URL 中是否包含 group 名
    62：store_path0=/fastdfs           #指定文件存储路径，同 storage.conf
[root@client src]# cp mod_fastdfs.conf /etc/fdfs/
```

3. 移动目录，Nginx 需要 src 目录下其他几个文件

```
[root@client ~]# cp -r fastdfs-nginx-module /usr/local/
[root@client ~]# vim /usr/local/fastdfs-nginx-module/src/config
    6：ngx_module_incs="/usr/include/fastdfs /usr/include/fastcommon"
    15：CORE_INCS="$CORE_INCS /usr/include/fastdfs /usr/include/fastcommon"
```

4. 安装 Nginx

```
[root@client ~]# mkdir -p /temp/nginx/{client,proxy,fastcgi,uwsgi,scgi}
[root@client ~]# yum -y install pcre-devel zlib-devel
[root@client nginx-1.15.5]# ./configure --user=nginx --group=nginx
--add-module=/usr/local/fastdfs-nginx-module/src
--http-client-body-temp-path=/temp/nginx/client
--http-proxy-temp-path=/temp/nginx/proxy
--http-fastcgi-temp-path=/temp/nginx/fastcgi
--http-uwsgi-temp-path=/temp/nginx/uwsgi
--http-scgi-temp-path=/temp/nginx/scgi
[root@client nginx-1.15.5]# make && make install
```

5. 拷贝 FastDFS 的配置文件到/etc/fdfs

```
[root@client ~]# cp ~/fastdfs/conf/http.conf ~/fastdfs/conf/mime.types /etc/fdfs/
```

6. 修改 Nginx 的配置文件

注：group1 为组名。M00 对应 storage.conf 的 store_path0 字段

```
[root@client ~]# vim /usr/local/nginx/conf/nginx.conf
```

```
http {
    server {
        server_name  172.16.186.96;
        ...
        location /group1/M00/ {
                root /fastdfs/data/;
                ngx_fastdfs_module;
        }
        ...
        }}
```

7. 启动 Nginx

```
[root@client ~]# /usr/local/nginx/sbin/nginx
```

Git：

一个开源的分布式版本控制系统，用于敏捷高效地处理任何项目。

采用了分布式版本库的方式，不需要服务器端软件支持

工作区：就是你在电脑里能看到的目录

暂存区：一般存放在.git/index 中,所以我们把暂存区也叫作索引(index)

版本库：工作区有一个隐藏目录.git,这个不算工作区,而是 Git 的版本库

gitlib：用于实现 git 功能的开发库

github：一个基于 git 实现的在线代码仓库，有网站界面

gitlab：一个基于 git 实现的在线代码仓库软件，用于企业搭建 github 私服

git 相关安装包:

git：基本功能，命令行模式

git-daemon：使用 git://xxx/xxx.git 的方式访问 git

gitweb：使用 web 方式访问 git

GitHub：

1.在网站上创建仓库 https://github.com，用户名为 a1441668968，仓库名为 test

2.装包：[root@room9pc01 ~]# yum -y install git

3.下载代码：[root@room9pc01 ~]# git clone https://github.com/a1441668968/test

4.上传代码需要配置个人设置

```
[root@fzr ~]# git config --global user.name a1441668968
[root@fzr ~]# git config --global user.email 1441668968@qq.com
[root@fzr ~]# git config --list
```

5.初始化版本库

```
[root@fzr test]# git init .
```

6.设置编写代码说明的编辑器是 vim

```
[root@fzr ~]# git config --global core.editor vim
```

7.查看状态

```
[root@fzr test]# git status
```

8.增加文件到暂存区：

```
[root@room9pc01 test]# echo 1 > test.txt
[root@room9pc01 test]# git add .
```

9.提交文件到版本库

[root@room9pc01 test]# git commit -m "xx"

10. 上传，提示输入用户名和密码

[root@room9pc01 test]# git push

11. 查看版本库：

[root@fzr test]# git log

12. 查看版本库中的文件

[root@fzr test]# git ls-files

13. 查看文件在工作目录与暂存区的差别：git diff 目录

14. 恢复版本：版本库编号为 f67779ddde5acb949c77f4b41f2dc3a956456bc1

[root@fzr test]# git checkout f67779

15. 更新本地数据：git pull = git fetch + git merge，先取下更新再合并

[root@fzr test]# git pull

本地 gitlab 服务器：

1. 安装相关软件包：

container-selinux
container-storage-setup
docker
docker-client
docker-common
oci-register-machine
oci-systemd-hook
oci-umount
skopeo-containers

2. 导入中文版 gitlab 容器镜像：

[root@host1 ~]# docker load < gitlab_zh.tar

或

[root@host1 ~]# docker pull registry.cn-hangzhou.aliyuncs.com/acs-sample/gitlab-ce-cn:8.8.5

[root@host1 ~]# docker tag registry.cn-hangzhou.aliyuncs.com/acs-sample/gitlab-ce-cn:8.8.5 gitlab_zh

3. 由于该虚拟机只为了加载容器，为方便上传，将虚拟机远程端口改为 2222，将容器的远程端口映射为 22

[root@host1 ~]# docker run -h gitlab --name gitlab -p 443:443 -p 80:80 -p 22:22 --restart always -v /srv/gitlab/config:/etc/gitlab -v /srv/gitlab/logs:/var/log/gitlab -v /srv/gitlab/data -d gitlab_zh

4. 网页访问：第一次登录时需要给 root 设置密码，至少 8 位数，建议设置本地域名解析

http://gitlab/或 http://192.168.1.126

5. 依次创建群组、项目和用户，将用户加入到项目中并设置角色

6. 生成 URL：http://gitlab/devops/test.git，页面下方会列出相关操作代码

7. 新用户登陆后，将公钥复制到 gitlab 的 SSH 密钥中后，才可以使用 ssh 上传代码

8. 上传：ssh 路径：git@gitlab:devops/test.git

[root@host2 myproject]# git init .

[root@host2 myproject]# git add .

```
[root@host2 myproject]# git commit -m "add"
[root@host2 myproject]# git remote rename origin old-origin
[root@host2 myproject]# git remote add origin git@gitlab:devops/test.git
[root@host2 myproject]# git push -u origin --all
```

9.tag 标签：用于记录达到一个重要的阶段后，特别重要的快照

打标签：git tag 标签名

查看标签：git tag

网页上查看：http://192.168.1.126/root/myproject/tags

切换到标签：git checkout 标签名

切换到 master 最新的标签：git checkout master

上传标签，建议使用 ssh 路径：git push -u origin --tags

Git 分支：

新建并切换分支：git checkout -b 分支名

新建分支：git branch 分支名

切换分支：git checkout 分支名

合并分支：git merge 被合并分支名

删除分支：git branch -d 分支名

合并分支时产生冲突，查看冲突：git status

制作 RPM 包：把源码安装的结果打包

1.安装软件：[root@web1 ~]# yum -y install rpm-build

2.生成 rpmbuild 目录：[root@web1 ~]# rpmbuild -ba xxxx　　#xxxx 随便输

3.将源码软件复制到 SOURCES 目录：[root@web1 ~]# cp lnmp_soft/nginx-1.12.2.tar.gz rpmbuild/SOURCES/

4.创建 spec 文件：[root@web1 ~]# vim rpmbuild/SPECS/nginx.spec　　#自动创建模板

```
Name:nginx
Version:1.12.2
Release:    1%{?dist}
Summary:    1
License:    GPL
Source0:nginx-1.12.2.tar.gz
%description
%prep
%setup -q
%build
./configure --user=nginx --group=nginx --with-stream
make %{?_smp_mflags}
%install
make install DESTDIR=%{buildroot}
%files
%doc
/usr/local/nginx/*
```

%changelog

5.创建 RPM 包：[root@web1 ~]# rpmbuild -ba rpmbuild/SPECS/nginx.spec

6.查看 RPM 包：[root@web1 ~]# ls /root/rpmbuild/RPMS/x86_64/

nginx-1.12.2-1.el7.x86_64.rpm nginx-debuginfo-1.12.2-1.el7.x86_64.rpm

7.发布：[root@web1 ~]# scp rpmbuild/RPMS/x86_64/nginx-1.12.2-1.el7.x86_64.rpm 192.168.2.200:~

虚拟专用网络 VPN(Virtual Private Network)：

在公网上建立专用私有网络，进行加密通讯，多用于出差人员连接公司服务器，以及各地区子公司建立连接

主流的 VPN 技术：GRE，PTP，L2TP+IPSec，SSTP

使用系统内置的 GRE：缺少加密机制，仅用于 Linux 之间，临时配置，应用于两个固定的公网 IP 之间，如为分布在各地的公司服务器建立专线

1.查看 GRE（选做）：

[root@user ~]# modprobe ip_gre #加载模块

[root@user ~]# lsmod | grep ip_gre #查看是否加载成功

[root@user ~]# modinfo ip_gre #查看模块信息,模块放在/lib/modules/目录下

[root@user ~]# rmmod ip_gre #删除模块

2.在客户端上创建隧道：

[root@user ~]# ip tunnel add tun0 mode gre remote 201.1.2.5 local 201.1.2.10

#ip 相关命令可通过 ip help 查看,tunne 为隧道相关命令；tun0 为自定义隧道名；mode 设置隧道模式；remote 设置对方 IP；local 表示本机 IP

[root@user ~]# ip link set tun0 up #启动隧道网卡

[root@user ~]# ip link show #查看网卡

[root@user ~]# ip address add dev tun0 10.10.10.10/24 peer 10.10.10.5/24

#peer 前设置隧道的本机 IP；peer 后设置对方 IP，影响 destination 字段，但实际无用，可不写,这些 IP 用于隧道内通信

PS：删除隧道：[root@user ~]# ip tunnel del tun0

3.在服务端创建对应的隧道：

[root@vpn ~]# ip tunnel add tun0 mode gre remote 201.1.2.10 local 201.1.2.5

[root@vpn ~]# ip link set tun0 up

[root@vpn ~]# ip address add dev tun0 10.10.10.5/24 peer 10.10.10.10/24

[root@vpn ~]# ip address show

4.服务端开启路由转发，默认已开启：

[root@vpn ~]# echo 1 > /proc/sys/net/ipv4/ip_forward

5.如果网络不通，可能是路由导致的，在客户端加一条路由：

[root@user ~]# route add -net 192.168.2.0/24 gw 10.10.10.5

[root@user ~]# route add -net 0.0.0.0/0 gw 10.10.10.5 加默认路由也可以

[root@user ~]# route -n

点对点隧道协议 PPTP(Point to Point Tunneling Protocol)：

概述:使用账号密码进行身份验证，用户名和密码使用 MPPE 加密，不对数据进行加密，

支持 Windows 访问，默认端口号 1723

注意：链路控制协议 LCP(Link Control Protocol)有时候会阻止访问，配置 /etc/sysconfig/iptables-config，增加模块

1.安装：

```
[root@vpn vpn]# yum -y install pptpd
[root@vpn vpn]# rpm -qc pptpd                    #-q 查询; -c 配置文件
    /etc/ppp/options.pptpd
    /etc/pptpd.conf
    /etc/sysconfig/pptpd
```

2.修改配置文件：参照第 102-106 行

```
[root@vpn ~]# vim /etc/pptpd.conf
    ...
    localip 201.1.2.5                            #服务器本地 IP
    remoteip 192.168.3.1-100,192.168.3.245       #分配给客户端的 IP 池,连续的用-表示，不连续的用逗号隔开
[root@vpn ~]# vim /etc/ppp/options.pptpd
    40:require-mppe-128                          #默认不修改
    66:ms-dns 8.8.8.8                            #默认注释，修改 DNS 服务器地址
[root@vpn ~]# vim /etc/ppp/chap-secrets          #设置账户
    client          用户名
    server          服务器标示
    secret          密码
    IP addresses    允许的客户端 IP
    # client        server  secret                  IP addresses
    jacob           *       123456                  *
```

3.启动服务：[root@vpn ~]# systemctl restart pptpd.service

4.在 Windows 上设置 VPN

5.翻墙需要在代理服务器上设置防火墙转发规则,表示将内网的地址转化为本机的地址:

```
[root@vpn ~]# iptables -t nat -A POSTROUTING -s 192.168.3.0/24 -j SNAT --to-source 201.1.2.5
```

补充：[root@fzr ~]# sed -n 6p /etc/sysconfig/iptables-config

```
IPTABLES_MODULES="nf_nat_pptp"
```

L2TP+IPSec:

L2TP(Layer Two Tunneling Protocol)第二层隧道协议：建立主机间的 VPN 隧道、压缩、验证

IPSec(Internet Protocol Security)因特网协议安全性：提供数据加密、数据校验、访问控制

IPSec:

1.安装：[[root@10 ~]# yum -y install libreswan

2.查看主配置文件，不修改：

```
[root@10 ~]# cat /etc/ipsec.conf                 #密钥配置主文件,
```

virtual_private 指定 IP 地址的分配范围,%v4 代表 ipv4,%v6 代表 ipv6

```
        virtual_private=%v4:10.0.0.0/8,%v4:192.168.0.0/16,%v4:172.16.0.0/12,
    %v4:25.0.0.0/8,%v4:100.64.0.0/10,%v6:fd00::/8,%v6:fe80::/10
        include /etc/ipsec.d/*.conf
    [root@10 ipsec.d]# cat /etc/ipsec.secrets        #预定义共享密钥主文件
        include /etc/ipsec.d/*.secrets
```

3.修改配置文件：

```
    [root@10 vpn]# vim /etc/ipsec.d/myipsec.conf
        conn IDC-PSK-NAT
            rightsubnet=vhost:%priv                          #允许建立的 VPN 虚拟
    网络
            also=IDC-PSK-noNAT

        conn IDC-PSK-noNAT
            authby=secret                                    #加密认证
                ike=3des-sha1;modp1024                       #算法
                phase2alg=aes256-sha1;modp2048               #算法
            pfs=no
            auto=add
            keyingtries=3
            rekey=no
            ikelifetime=8h
            keylife=3h
            type=transport
            left=201.1.2.200                                 #服务端的外网 IP
            leftprotoport=17/1701                            #指定服务端端口号
    1701
            right=%any                                       #允许的客户端，%any
    表示匹配所有
            rightprotoport=17/%any                           #允许的客户端端口号
    [root@10 ipsec.d]# vim mypass.secrets
    #   VPN 服务器 IP    可连接 IP    预共享密钥(pre-shared key)  密钥
        201.1.2.200     %any:       PSK                          "randpass"
```

4.启动 IPSec：

```
    [root@10 ~]# systemctl start ipsec
    [root@10 ~]# netstat -ntulp | grep pluto
```

XL2TP:

1.装包：[root@10 vpn]# yum -y install xl2tpd

2.修改配置：

```
    [root@10 ~]# vim /etc/xl2tpd/xl2tpd.conf
        32:ip range = 192.168.3.128-192.168.3.254        #隧道 IP 地址池
        33:local ip = 201.1.2.200                        #VPN 服务器的 IP 地址
    [root@10 ~]# vim /etc/ppp/options.xl2tpd
        3:ms-dns  8.8.8.8
```

```
    10:#crtscts                                    #注释后表示不允许证书
    16:#lock                                       #注释后表示关闭某种加密
    21:require-mschap-v2                           #添加一行,强制要求认证
[root@10 ~]# vim /etc/ppp/chap-secrets
    # client        server  secret                 IP addresses
    jacob           *       123456                  *
```

3.启动服务：

```
[root@10 ~]# systemctl start xl2tpd
[root@10 ~]# netstat  -ntulp |grep xl2tpd
```

如果 Windows 连接 VPN 提示 789 错误的 BUG：

1.在 cmd 中输入"regedit"，进入注册表编辑器

2.找到注册表子项：

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Rasman\Parameters

3."新建"->"DWORD 值"->名称为："ProhibitIpSec"

4.将值设置为 1

5.重启 Windows

PSSH：提供了一套并发远程连接功能工具，使用 python 编写，需要 python 环境才能使用

pssh 的常用选项：

```
-A      使用密码远程其他主机（默认使用密钥）
-i      将输出显示在屏幕
-H      设置需要连接的主机
-h      设置主机列表文件
-p      设置并发数量
-t      设置超时时间
-o 目录 设置标准输出信息保存的目录
-e 目录 设置错误输出信息保存的目录
-x      传递参数给 ssh
-r      递归拷贝
```

1.安装：[root@proxy lnmp_soft]# yum -y install pssh-2.3.1-5.el7.noarch.rpm

安装完成后会提供 5 个命令：

```
/usr/bin/pnuke
/usr/bin/prsync
/usr/bin/pscp.pssh
/usr/bin/pslurp
/usr/bin/pssh
```

2.使用密码远程多台主机：

```
[root@proxy ~]# pssh -iAH "192.168.4.100 192.168.2.100 192.168.2.200" -x '-o StrictHostKeyChecking=no' date
```

3.实际环境，使用密钥远程：

```
[root@proxy ~]# cat /etc/hosts
```

```
192.168.4.5 proxy
192.168.4.100 client
192.168.2.100 web1
192.168.2.200 web2
[root@proxy ~]# cat host
client
web1
web2
[root@proxy ~]# ssh-copy-id client
[root@proxy ~]# ssh-copy-id web1
[root@proxy ~]# ssh-copy-id web2
[root@proxy ~]# pssh -ih 'host' -o /tmp/ date   #用密钥远程并将标准输出写入到/tmp 下
```

4.批量拷贝文件到其他主机：

```
[root@proxy ~]# pscp.pssh -h host lnmp_soft/nginx.conf ~
[root@proxy ~]# pscp.pssh -h host -r lnmp_soft/varnish-5.2.1/ ~ #递归拷贝目录
```

5.批量下载文件到本机：

```
[root@proxy ~]# pslurp -h host -r /etc /tmp/                #将远程主机的 etc 下载到本机当前目录下的对应主机名下的 tmp
[root@proxy ~]# ls client/ web1/ web2/
client/:
tmp

web1/:
tmp

web2/:
tmp
[root@proxy ~]# pslurp -r -h host -L /media /boot/ /tmp/    #将远程主机的 boot 载到本地的 media 下的对应主机名下的 tmp
[root@proxy ~]# ls /media/client/ /media/web1/ /media/web2/
/media/client/:
tmp

/media/web1/:
tmp

/media/web2/:
tmp
```

6.批量杀死其他主机进程：

```
[root@proxy ~]# pssh -h host sleep 50 &          #运行一个进程
[root@proxy ~]# pnuke -h host sleep              #杀死所有带有 sleep 的进程
```

```
[root@proxy ~]# pssh -h host pkill sleep        #杀死所有带有 sleep 的进程
```

邮件系统：postfix

```
[root@server0 ~]# vim /etc/postfix/main.cf
    76:myhostname = server0.example.com                                    #主机名
    83:mydomain = example.com                                              #域名的后缀，邮箱的后缀
    99:myorigin = $mydomain                                                #发件人
    116:inet_interfaces = all                                              #允许所有人给自己发邮件，默认仅接收本机发的邮件
    164:mydestination = $myhostname, localhost.$mydomain, localhost        #接收的邮件域名
    264:mynetworks = [::1]/128, 127.0.0.0/8                                #信任的网络
    313:relayhost = [gateway.my.domain]                                    #后端邮件服务器
[root@server0 ~]# systemctl restart postfix
[root@server0 ~]# mail -s 标题    收件人
        内容
        内容
        .                    #独立的点回车表示邮件内容结束
[root@server0 ~]# echo "内容" | mail -s 标题    收件人
[root@server0 ~]# mail -s 标题    收件人 < /etc/passwd
[root@server0 ~]# mail -s 标题    收件人 << EOF
    内容
    EOF
[root@server0 ~]# mail              #收取邮件，q 退出
```

分区工具 parted
fdisk 分区只能分 4 个主分区，每个分区最大只能是 2T
parted 可以划分最多 128 个主分区,分区可以大于 2T

parted 分区首先要选择分区的类型：msdos,gpt 模式（msdos 就是 fdisk 使用的模式）

```
[root@server0 ~]# parted /dev/vdb mklabel gpt       #选择分区模式，可以是msdos 或者 gpt
[root@server0 ~]# parted /dev/vdb mkpart primary 0 2G   #对/dev/vdb 分区(mkpart),分主分区(primary)，从硬盘的开始到 2G 之间的容量分一个分区
[root@server0 ~]# parted /dev/vdb mkpart primary 2G 5G
[root@server0 ~]# parted /dev/vdb print         #查看分区结果
[root@server0 ~]# mkswap /dev/vdb2              #创建交换分区，也就是虚拟内存
```

[root@server0 ~]# vim /etc/fstab
/dev/vdb2 swap swap defaults 0 0
[root@server0 ~]# swapon -a

交换分区命令：
格式：swapon［参数］虚拟分区名
参数：-a 激活所有交换分区
-s 查看交换分区

自定义yum服务器
yum源（yum服务器）：
需要有RPM软件包
数据库
repodata/ 仓库档案数据
filelists.xml.gz #软件包的文件安装清单
primary.xml.gz #软件包的基本/主要信息
other.xml.gz #软件包的其他信息
repomd.xml #提供.xml.gz下载和校验信息
1.加载光盘直接做
[root@desktop0 ~]# mkdir /var/www/html/rhel
[root@desktop0 ~]# vim /etc/fstab
/iso/rhel-server-7.4-x86_64-dvd.iso /var/www/html/rhel/ iso9660 defaults,loop 0 0
[root@desktop0 ~]# mount -a
备注：在光盘里有Packages，里面都是RPM包；repodata目录，里面有相关数据库
[root@desktop0 ~]# systemctl restart httpd

2.网上下载，通常只有rpm软件
[root@desktop0 ~]# unzip other.zip -d /var/www/html/ #将软件解压到文件夹下
[root@desktop0 ~]# createrepo /var/www/html/other #给/var/www/html/other目录下RPM软件自动生成数据库

源码包安装软件
优点：可以实现自定义安装，可以只安装需要的模块
1.下载源码包
[root@room9pc01 ~]# wget ftp://172.40.50.118/course/SERVICE/inotify-tools-3.13.tar.gz
[root@room9pc01 ~]# scp inotify-tools-3.13.tar.gz 172.25.0.11:/root/ #把文件的inotify-tools-3.13.tar.gz拷贝到172.25.0.11电脑的/root/目录

2.安装软件
[root@server0 ~]# tar -xf inotify-tools-3.13.tar.gz
[root@server0 ~]# cd inotify-tools-3.13/ #计算机的系统软件一般都是用

c 语言写的

[root@server0 inotify]# ./configure #检查你的计算机环境

[root@server0 inotify]# yum -y install gcc #gcc 是 linux 里面的一个 C 语言的解释器

[root@server0 inotify]# ./configure (--prefix=/路径) #检查环境，并设置安装路径，如果没有指定 prefix，则默认一般在/usr/local/

[root@server0 inotify]# make #编译，用 gcc 解释器把源码转换为二进制

[root@server0 inotify]# make install #把编译好的二进制程序安装到你的计算机

inotify-tools 这个软件可以监控你的计算机目录

[root@server0 ~]# inotifywait -mrq /root/ #监控 root 目录的所有操作

systemctl 命令

格式：systemctl 选项 参数

选项：

-t service #列出启动的服务

-t service --all #列出所有的服务,包括没启动的

stop #关闭服务

disable #取消开机自启动

enable #开机自启

start #立刻启动服务

restart #重启服务

status #查看某个服务是否启动

DNS 服务器的功能：

正向解析:根据注册的域名查找其对应的 IP 地址

反向解析:根据 IP 地址查找对应的注册域名(不常用)

DNS（域名解析）服务器：解析域名--->将域名解析为 IP 地址

例如：客户端访问 Web 网站：www.qq.com ----> DNS ----> 腾讯的 Web 服务器

Full Qualified Domain Name(FQDN),完全合格主机名，以点结尾

站点名.域名后缀

站点名...二级域.一级域

域名分级：

根域 .

一级域 .com .cn .us .tw .kr .hk

二级域 .com.cn .net.cn .org.cn .edu.cn

三级域 .tedu.com.cn .haha.com.cn .xixi.com.cn

常见的顶级（一级域名）

国家/地区域: .cn、.us、.kr、.hk、.tw ...

组织域: .com、.net、.edu、.org、.gov、.mil ...

BIND 域名服务

BIND(Berkeley Internet Name Daemon：伯克利大学 Internet 域名服务)

官方站点:https://www.isc.org/

安装包名称：bind：域名服务包 ；bind-chroot：提供虚拟根支持,笼环境

系统服务:named

默认端口:TCP/UDP 53

运行时的虚拟根环境:/var/named/chroot/ #运行时，所有的操作都在虚拟根下

主配置文件: /etc/named.conf #设置本机负责解析的域名是什么

地址库文件: /var/named/ #主机名与 ip 地址的对应关系

搭建基本的 DNS 服务

服务端：

1.安装软件包

```
[root@svr7 ~]# yum -y  install   bind    bind-chroot
```

2.修改配置文件

```
[root@svr7 ~]# vim /etc/named.conf
 options {
    //listen-on port 53 { 127.0.0.1; };     #默认监听 127.0.0.1 的 53 端口,该行删除或注释
    directory   "/var/named";     #指定地址库文件存放路径
    //allow-query     { localhost; };   #允许访问的主机，默认为本机，该行删除或注释
  };
 zone   "tedu.cn"  IN {            #指定本机负责解析的域名
    type  master;                  #指定本机为权威主 DNS 服务器
    file  "tedu.cn.zone";          #指定地址库文件为 tedu.cn.zone
  };
[root@svr7 ~]# named-checkconf      #检查配置是否正确
```

3.建立地址库文件 tedu.cn.zone

DNS 资源记录：

SOA：起始授权

NS：名称服务器

A：将名称解析为 ip 地址

PTR：将 ip 地址解析为名称

MX：邮件交换器

CHAME：设置别名

```
[root@svr7 ~]# cd /var/named/
[root@svr7 named]# cp -p named.localhost tedu.cn.zone  #权限属性不变拷贝
[root@svr7 named]# ls -l tedu.cn.zone
-rw-r----- 1 root named 152 6月  21 2007 tedu.cn.zone  #必须保证属组是named
[root@svr7 named]# vim  tedu.cn.zone
$TTL 1D                          #DNS 解析的有效时间 1 天
@   IN  SOA ....                 #@表示本域名，这段不修改
```

```
     tedu.cn.   NS   svr7.tedu.cn.      #声明 tedu.cn.域名的 DNS 服务器为
svr7.tedu.cn.
     svr7       A    192.168.4.7       #指定 svr7.tedu.cn.的 ip 地址为 192.168.4.7
     www        A    1.1.1.1
     ftp        A    2.2.2.2
```

4.重起 named，设置开机自启动

```
    [root@svr7 named]# systemctl restart named
    [root@svr7 named]# systemctl enable named
```

5.客户端访问测试：

```
    [root@pc207 ~]# echo nameserver 192.168.4.7 > /etc/resolv.conf
    [root@pc207 ~]# nslookup www.tedu.cn
```

多区域 DNS 服务（多写几个 zone）

配置文件/etc/named.conf 新增如下内容：

```
    zone "qq.com" IN {
        type master;
        file "qq.com.zone";
    };
    [root@svr7 named]# vim  qq.com.zone
     qq.com.   NS   svr7
     svr7      A    192.168.4.7
     www     A    3.3.3.3
     ftp       A    4.4.4.4
```

基于 DNS 域名的负载均衡

```
    [root@svr7 named]# vim qq.com.zone
    qq.com.   NS   svr7
    svr7      A    192.168.4.7
    www       A    192.168.4.11    #DNS 服务器会随机从 3 个 IP 中选择一个提供，后面的
域名可以省略
                A    192.168.4.12
                A    192.168.4.13
```

泛域名解析：以*匹配所有

```
    [root@svr7 named]# vim qq.com.zone
     qq.com.   NS   svr7
     svr7      A    192.168.4.7
     www       A    192.168.4.11
     *         A    1.2.3.4
```

有规律的泛域名解析

函数： $GENERATE 生成连续范围的数字

```
    [root@svr7 named]# vim qq.com.zone
      qq.com.   NS   svr7
```

```
svr7      A    192.168.4.7
www       A    192.168.4.11
ftp       A    4.4.4.4
*         A    1.2.3.4
$GENERATE  1-50  web$   A    192.168.10.$
```

DNS 的子域授权

父域 ：www.tedu.cn ，由 svr7 服务器 192.168.4.7 解析

子域 ：www.bj.tedu.cn ，由 pc207 服务器 192.168.4.207 解析

虚拟机 A：子域授权

```
[root@svr7 named]# vim  tedu.cn.zone   #指定子域的 DNS 服务器
tedu.cn.     NS  svr7
bj.tedu.cn.  NS  pc207.bj
svr7         A   192.168.4.7
pc207.bj     A   192.168.4.207
[root@svr7 named]# systemctl restart named
```

虚拟机 B：搭建子域 DNS 负责解析 bj.tedu.cn 域名

修改配置文件/etc/named.conf

```
options {
   directory   "/var/named";
 };
zone   "bj.tedu.cn"   {
   type  master;
   file  "bj.tedu.cn.zone";
 };
[root@pc207 named]# vim   bj.tedu.cn.zone
 bj.tedu.cn.  NS  pc207
 pc207    A   192.168.4.207
 www          A   11.12.13.14
```

测试：

```
[root@svr7 named]# nslookup  www.bj.tedu.cn   192.168.4.7
Server:     192.168.4.7
Address:          192.168.4.7#53

Non-authoritative answer:    #非权威解答
Name:   www.bj.tedu.cn
Address: 11.12.13.14
```

递归查询： 首选 DNS 服务器到相应其他 DNS 服务器上询问，将最终结果带回（客户端与首选 DNS 服务器交互）

迭代查询： 每一个 DNS 服务器告知首选 DNS 服务器下一个 DNS 服务器地址（首选 DNS 服务器与其他 DNS 服务器交互）

递归查询是默认开启的

通过设置/etc/named.conf 开关

recursion yes/no;

缓存 DNS

作用：加速解析过程，让客户端最快得到结果

主要适用环境：互联网出口带宽较低的企业局域网；ISP 服务商的公共 DNS 服务器

解析记录来源：

全局转发:将请求转发给指定的公共 DNS(其他缓存 DNS),请求递归服务

根域迭代:依次向根、一级、二级.....域的 DNS 服务器迭代

配置：在/etc/named.conf 的 options 下增加 forwarders { 176.19.0.26; };

#添加转发器，176.19.0.26 是公共 DNS 的 IP 地址

Split 分离解析：

当收到客户机的 DNS 查询请求的时候能够区分客户机的来源地址

为不同类别的客户机提供不同的解析结果(IP 地址)

当不同类别的客户机请求解析同一个域名时，得到的解析结果不同

适用场景：

访问压力大的网站，购买 CDN 服务商提供的内容分发服务：

在全国各地、不同网络内部部署大量镜像服务节点

让客户端访问网络中最近/最快的服务器

方法：BIND 的 view 视图

根据源地址集合将客户机分类：

不同客户机获得不同结果

客户机分类得当（所有的客户端都要找到对应的分类：兜底）

由上到下匹配，匹配即停止

所有的 zone 都必须在 view 字段里面

格式：

```
view "nsd" {
  match-clients { 192.168.4.207; }   #匹配客户端的地址
  zone "tedu.cn" {
    ..... 地址库 1;
   };
};

 view "abc" {
  match-clients { any; }            #any 兜底
  zone "tedu.cn" {
   ..... 地址库 2;
  };
};
```

配置 ACL 队列库：

acl myip { IP1;IP2; ... };

修改配置文件:

[root@svr7 /]# vim /etc/named.conf

acl myip { 192.168.4.207; 192.168.4.10; 192.168.4.1; 192.168.4.123; };

view "nsd" {

```
     match-clients   {   myip  };
     zone  "tedu.cn"  {
        type  master;
        file  "tedu.cn.nsd";
        };
      };
    view  "abc"  {
     match-clients  {   any;  };
     zone  "tedu.cn"  {
        type  master;
        file  "tedu.cn.abc";
        };
      };
```

配置地址库文件

```
[root@svr7 /]# tail  -3  /var/named/tedu.cn.nsd
tedu.cn.     NS  svr7
svr7         A   192.168.4.7
www         A   192.168.4.100

[root@svr7 /]# tail  -3  /var/named/tedu.cn.abc
tedu.cn.     NS  svr7
svr7         A   192.168.4.7
www          A   1.2.3.4
```

RAID 磁盘阵列（Redundant Arrays of Inexpensive Disks：廉价冗余磁盘阵列）

通过硬件/软件技术,将多个较小/低速的磁盘整合成一个大磁盘

阵列的价值:提升 I/O 效率（速度）、硬件级别的数据冗余(备份)

附加优点：拥有更大容量的磁盘、比直接购买大容量/高速磁盘性价比更高

RAID 实现方式：硬 RAID：由 RAID 控制卡管理阵列；软 RAID；由操作系统来管理阵列

不同 RAID 级别的功能、特性各不相同:

RAID 0,条带模式

同一个文档分散存放在不同磁盘

并行写入以提高效率

至少 2 块磁盘

RAID 1,镜像模式

一个文档复制成多份,分别写入不同磁盘

多份拷贝提高可靠性,效率无提升

至少 2 块磁盘

RAID 0+1/RAID 1+0

整合 RAID 0、RAID 1 的优势

并行存取提高效率、镜像写入提高可靠性

至少 4 块磁盘

RAID5,高性价比模式

相当于 RAID0 和 RAID1 的折中方案

需要至少一块磁盘的容量来存放校验数据

至少 3 块磁盘

RAID6,高性价比/可靠模式

相当于扩展的 RAID5 阵列,提供 2 份独立校验方案

需要至少两块磁盘的容量来存放校验数据

至少 4 块磁盘

进程管理

程序：静态的代码，占用硬盘

进程：动态的代码，占用 cpu、内存

分类：父进程、子进程（树型结构）

进程的标识：PID

pstree(Processes Tree)：进程树

格式:pstree [选项] [PID 或用户名]

常用命令选项

-a:显示完整的命令行

-p:列出对应 PID 编号

PS:systemd 是所有进程的父进程 ，PID=1

```
[root@svr7 /]# pstree
[root@svr7 /]# pstree  lisi
[root@svr7 /]# pstree  -p  lisi
[root@svr7 /]# pstree  -ap  lisi
```

ps(Processes Snapshot)：进程快照

参数：

aux：列出正在运行的所有进程 ：用户 进程 ID CPU 占比 内存占比 虚拟内存固定内存 终端 状态 起始时间 CPU 时间 程序指令

-elf：列出正在运行的所有进程，其中 PPID 是该进程的父进程的 PID

-eo：自定义查看所有进程的项目

```
[root@fzr ~]# ps -eo ppid,pid,stat,cmd | grep Z #查看僵尸进程
[root@fzr ~]# kill -SIGCONT ppid                    #让僵尸进程的父进程杀死僵尸
[root@fzr ~]# ps -eo %mem,ucmd|awk '{A[$2]+=$1}END{for(a in A)if(A[a]>0)printf("%.2f\t%s\n",A[a],a)}'   #查看程序占用内存
[root@fzr ~]# ps -eo %cpu,ucmd|awk '{A[$2]+=$1}END{for(a in A)if(A[a]>0)printf("%.2f\t%s\n",A[a],a)}'   #查看程序占用 CPU
```

top 交互式工具

格式:top [-d 刷新秒数]

[root@svr7 ~]# top -d 1

load average（CPU 处理器的平均负载）：最近 1 分钟平均、最近 5 分钟平均、最近 15 分钟平均

按 1 切换多核模式，id 表示 CPU 空闲率

按 P 按 CPU 占有比例降序排列

按 M 按内存占有比例降序排列

按 k 输入 PID 可杀死该进程

iotop：查看每个命令的硬盘 IO 使用情况

iostat：监控磁盘 IO 状态

安装包名：sysstat

命令：iostat 选项 时间间隔 次数

pgrep （Process Grep）：检索进程

格式：pgrep [选项]... 查询条件

选项：-l 输出进程名

-U 检索指定用户进程,与 l 连用

-t 检索指定终端的进程,终端可通过 who 命令查看

[root@svr7 ~]# pgrep cron

[root@svr7 ~]# pgrep -l cron

[root@svr7 ~]# pgrep -lU lisi

[root@svr7 ~]# pgrep . | wc -l #查看当前主机运行的进程数

free 查看空余的内存

参数：

-m 以 M 为单位

-g 以 G 为单位

显示列说明;

total：总内存

used：被使用的

free：空闲的

shared：共享内存

buff/cache：被用于缓存的

iptraf：监控网络

安装包名：iptraf-ng

进入文本界面：iptraf-ng

lsof(list opened files)：列举系统中已经被打开的文件

输出字段说明：

1、COMMAND：默认以 9 个字符长度显示的命令名称。可使用+c 参数指定显示的宽度，若+c 后跟的参数为零，则显示命令的全名(-c)

2、PID：进程的 ID 号(-p)

3、TID：如果是一个线程，显示线程 ID

4、USER：命令的执行 UID 或系统中登陆的用户名称。(-u)

5、FD：是文件描述符 File Descriptor number(-d)

6、TYPE：类型

7、DEVICE：使用 character special、block special 表示的设备号

8、SIZE/OFF：文件的大小，如果不能用大小表示的，会留空。

9、NODE：本地文件的 inode id ，挂载点和文件的全路径（链接会被解析为实际路径），或者连接双方的地址和端口、状态等

示例：

```
[root@fzr ~]# lsof urfile /      #显示根文件被哪些程序调用
[root@fzr ~]# lsof -c systemd    #显示 systemd 进程使用了哪些文件
[root@fzr ~]# lsof -p 3867       #显示 pid=3867 的进程使用了哪些文件
[root@fzr ~]# lsof -i :22        #显示 22 端口被哪些 ip 使用
[root@fzr ~]# lsof  +d  /root    #显示 root 下被打开的文件
[root@fzr ~]# lsof  +D /root     #递归显示 root 下被打开的文件
[root@fzr ~]# lsof -u root       #显示 root 用户打开的文件
[root@fzr ~]# lsof -c '/^bash/i' -a -d cwd  /root   #查看/root 目录正在被哪些进程调用
[root@fzr ~]# lsof|grep deleted #显示哪些文件被删除后没有释放空间
```

fuser：报告进程使用的文件和网络套接字

格式：fuser 选项 参数

选项：

-a：显示命令行中指定的所有文件；

-k：杀死访问指定文件的所有进程；

-i：杀死进程前需要用户进行确认；

-l：列出所有已知信号名；

-m：指定一个被加载的文件系统或一个被加载的块设备；

-n：选择不同的名称空间；

-u：在每个进程后显示所属的用户名。

参数：文件名或者 TCP、UDP 端口号

进程的前后台调度

后台启动： 在命令行末尾添加“&”符号,不占用当前终端

fg (forground) 命令 ：将后台任务恢复到前台运行

bg (background) 命令 ：激活后台被挂起的任务

jobs:查看后台进程，-l 显示 PID

```
[root@svr7 ~]# sleep 800 &       #将程序放入后台运行
[root@svr7 ~]# jobs              #查看后台进程
[root@svr7 ~]# sleep 700         按 Ctrl + z  暂停放入后台
[root@svr7 ~]# jobs              #查看后台进程
[1]-  运行中                 sleep 800 &
[2]+  已停止                 sleep 700
[root@svr7 ~]# bg 2              #将后台编号为 2 的进程继续运行
```

```
[root@svr7 ~]# jobs
[1]-  运行中                 sleep 800 &
[2]+  运行中                 sleep 700 &
[root@svr7 ~]# fg 2              #将后台编号为2的进程恢复到前台
sleep 700
```

杀死进程

```
kill    [-9]  PID...
killall [-9]  进程名...
pkill    关键字...
killall [-9] -u  用户名        #杀死该用户开启的所有进程，-9表示强制踢出用户
[root@svr7 ~]# killall sleep
```

进程：系统进行资源分配和调度的基本单位，是操作系统结构的基础

进程的状态：

- -D 无法中断的休眠状态，无法被强制回收内存终止
- -R 运行中的进程
- -S 休眠的进程
- -T 停止或被追踪
- -t 跟踪调试状态
- -X 死掉的进程
- -Z 僵尸进程

父进程派生子进程的时候子进程会继承：环境、堆栈、内存、打开的文件、进程组号、终端

进程间的通讯方式：管道、信号、报文、共享内存、信号量、套接字

一般来说，括号和管道都会产生子进程

进程替换与命令替换：

命令替换：把一个命令的结果赋予给一个变量

进程替换：把一个进程的输出回馈给另一个进程

多进程:

fork:分岔.父进程创建一个子进程,并将自身资源拷贝一份,命令在子进程中运行时,具有和父进程完全一样的运行环境

子进程的生命周期:

父进程fork出子进程并挂起

子进程运行完毕后,释放大部分资源并通知父进程,这个时候,子进程变成僵尸进程

父进程获知子进程结束,子进程所有资源释放,僵尸进程结束

注意:

僵尸进程没有任何可执行代码,也不能被调度

如果系统中存在过多的僵尸进程,将因为没有可用的进程号而导致系统不能产生新的进程

杀死僵尸进程的父进程或重启系统来消除僵尸进程

示例:定位僵尸进程的父进程

```
[root@room9pc01 ~]# ps -eo ppid,pid,stat,cmd | grep Z
```

线程:所有的线程运行在同一个进程中,共享相同的运行环境

状态:开始,顺序执行和结束

线程的运行可能被抢占(中断),或暂时的被挂起(睡眠),让其它的线程运行,这叫做让步

一个进程中的各个线程之间共享同一片数据空间,所以线程之间可以比进程之间更方便地共享数据、相互通讯

线程一般都是并发执行

多线程:

线程的并行和数据共享的机制使得多个任务可以相互合作

全局解释器锁（Global Interpreter Lock）是计算机程序设计语言解释器用于同步线程的工具，使得任何时刻仅有一个线程在执行

由于线程是基于进程的,所以,所有的线程都由一个 CPU 核心执行

日志管理

功能：系统和程序的“日记本”

记录系统、程序运行中发生的各种事件

通过查看日志,了解及排除故障

主要日志文件：

/var/log/	日志目录
/var/log/messages	记录内核消息、各种服务的公共消息
/var/log/dmesg	记录系统启动过程的各种消息
/var/log/cron	记录与 cron 计划任务相关的消息
/var/log/maillog	记录邮件收发相关的消息
/var/log/secure	记录与访问限制相关的安全消息
/var/log/btmp	查看登陆失败的记录，可用 lastb 命令读取
/var/log/wtmp	查看登陆和退出的记录，可用 last 命令读取
/var/run/utmp	查看当前登陆的用户，可用 who 命令读取
/var/log/lastlog	查看所有用户最后登陆的时间，可用 lastlog 读取

日志进程：rsyslogd

日志的配置文件：/etc/rsyslog.conf

```
*.info;mail.none;authpriv.none;cron.none                /var/log/messages
#info 级别及以上的日志记录；所有邮件不记录；所有认证信息不记录；所有计划任务不记录
authpriv.*      /var/log/secure  #记录所有成功认证和失败认证的信息
mail.*       -/var/log/maillog   #-表示实时同步信息到硬盘
cron.*     /var/log/cron         #计划任务执行日志
*.emerg       :omusrmsg:*        #系统崩溃后，通知所有登陆中的用户
```

分析：

tailf ：实时跟踪日志消息

last -f：读取日志文件

users、who、w ：查看已登录的用户信息,详细度不同

last、lastb ： 查看最近登录成功/失败的用户信息

```
[root@svr7 ~]# users
```

[root@svr7 ~]# who
[root@svr7 ~]# who | wc -l　　#查看有几个用户登陆
[root@svr7 ~]# w
[root@svr7 ~]# last -2　　#查看最近 2 次成功登陆
[root@svr7 ~]# lastb -2　　#查看最近 2 次失败登陆

logger 命令：默认把日志记录到/var/log/messages

常用参数：　-i 同时记录进程的 IPD
　　-p 指定级别和类 ixing
　　-t 指定进程名

[root@client ~]# logger -i -t dhcpd "hello"
[root@client ~]# logger -p local3.notice "haha"

ac 命令：报告用户连接的时间

常用参数：　-d 按天统计
　　-p 统计总数

logrotate：日志转存，以计划任务方式执行

配置文件：/etc/logrotate.conf
存放位置：rpm -qc logrotate

远程日志服务器：

UDP 方式:

1.修改日志服务器的配置文件：

[root@client ~]# vim /etc/rsyslog.conf +15
　　$ModLoad imudp
　　$UDPServerRun 514

2.服务端重启服务生效

[root@client ~]# systemctl restart rsyslog.service
[root@client ~]# ss -antup | grep 514

3.修改日志客户端的配置文件：

[root@proxy ~]# vim /etc/rsyslog.conf
+1:*.info @192.168.4.100

4.客户端重启服务生效

TCP 方式:

1.修改日志服务器的配置文件：

[root@client ~]# vim /etc/rsyslog.conf +19
　　$ModLoad imtcp
　　$InputTCPServerRun 514

2.服务端重启服务生效

3.修改日志客户端的配置文件：

[root@proxy ~]# vim /etc/rsyslog.conf
+1:*.info @@192.168.4.100

4.客户端重启服务生效

RELP 方式:(Reliable Event Logging Protocol)基于 TCP 封装的可靠日志消息传输协议

1.装包：[root@client ~]# yum -y install rsyslog-relp
　　[root@client ~]# ssh 192.168.4.5 "yum -y install rsyslog-relp"

2.修改配置文件：

[root@client ~]# man rsyslog.conf

复制到服务端：/etc/rsyslog.conf

$ModLoad imrelp

$InputRELPServerRun 2514

复制到客户端：/etc/rsyslog.conf

$ModLoad omrelp

. :omrelp:192.168.4.100:2514

使用 journalctl 工具

提取由 systemd-journal 服务搜集的日志（主要包括内核/系统日志、服务日志）

常见用法

journalctl | grep 关键词

常用参数

-x 扩展信息

-e 显示更多

--no-pager 取消分页

-u 服务名

-p 优先级。优先级高于或等于的都会列出

-n 消息条数

--since="yyyy-mm-dd HH:MM:SS" --until="yyyy-mm-dd HH:MM:SS" 按时间搜索

[root@svr7 ~]# journalctl -u httpd

[root@svr7 ~]# journalctl -u httpd -p 4 -n 10 | grep 8909

Linux 内核定义的事件紧急程度

分为 0~7 共 8 种优先级别， 其数值越小,表示对应事件越紧急/重要

0 EMERG（紧急）： 会导致主机系统不可用的情况

1 ALERT（警告）： 必须马上采取措施解决的问题

2 CRIT（严重）： 比较严重的情况

3 ERR（错误）： 运行出现错误

4 WARNING（提醒）： 可能会影响系统功能的事件

5 NOTICE（注意） ： 不会影响系统但值得注意

6 INFO（信息）： 一般信息

7 DEBUG（调试）： 程序或系统调试信息等

管理运行级别（运行模式）

RHEL5、RHEL6 切换运行级别的命令：init

参数：0 ： 关机

1 ： 单用户模式（破解密码、修复系统）

2 ： 字符模式（不支持网络）

3 ： 字符模式（支持网络）

4 ： 无定义

5 ： 图形模式

6 ： 重起

RHEL7 运行模式

multi-user.target 字符模式（支持网络）

graphical.target 图形模式

临时切换运行模式

```
[root@svr7 /]# systemctl isolate graphical.target   #临时直接切换到图形
[root@svr7 /]# systemctl isolate multi-user.target  #临时直接切换到字符
```

永久改变默认的运行模式

```
[root@svr7 /]# systemctl get-default                    #查看当前默认的运行模式
[root@svr7 /]# systemctl set-default graphical.target   #修改默认的运行模式
```

DHCP 自动分配网络参数服务

Dynamic Host Configuration Protocol : 动态主机配置协议,由 IETF(Internet 网络工程师任务小组)组织制定,用来简化主机地址分配管理

主要分配以下入网参数

IP 地址/子网掩码/广播地址

默认网关地址、DNS 服务器地址

DHCP 工作过程以广播的方式进行，先到先得

DHCP 地址分配的四次会话

DISCOVERY 客户机请求 --> OFFER 服务器提供--> REQUEST 客户机选择--> ACK 服务器确认

服务端基本概念

租期:允许客户机租用 IP 地址的时间期限,单位为秒

作用域:分配给客户机的 IP 地址所在的网段

地址池:用来动态分配的 IP 地址的范围

搭建基本 DHCP 服务

1.安装 dhcp 包

```
[root@svr7 ~]# yum -y install dhcp
[root@svr7 ~]# rpm -q dhcp
```

2.修改配置文件 /etc/dhcp/dhcpd.conf

网段：代表的一群 ip 地址的集合

网络位不变，主机位都用 0 表示

```
[root@svr7 ~]# vim /etc/dhcp/dhcpd.conf
末行模式      :r  /usr/share/doc/dhcp*/dhcpd.conf.example
 subnet  192.168.4.0  netmask  255.255.255.0  {  #指定分配的网段
```

```
    range 192.168.4.10   192.168.4.100;              #指定的分配具体的ip地址范围
    option domain-name-servers   8.8.8.8;            #指定 dns 地址
    option routers   192.168.4.254;                  #指定网关地址
}
```

4.重起 dhcpd 服务

5.确认监听状态

netstat 的常用选项：

-a 所有连接

-n 以数字方式显示地址和端口

-p 列出对应的进程和 PID

-t 列出 TCP 类型的连接

-u 列出 UDP 类型的连接

```
[root@room9pc01 ~]# netstat -anptu | grep :67                #DHCP 服务具有
```

排他性，如果有两个服务存在，需要结束其中一个

```
udp        0      0 0.0.0.0:67              0.0.0.0:*
8934/dhcpd
udp        0      0 0.0.0.0:67              0.0.0.0:*
1819/dnsmasq
```

6.使用客户机确认结果

```
[root@srv7 ~]# dhclient -d eth0                      #检查获取的 IP 地址是否
```

在预先指定的地址池中

网络装机的优势

规模化:同时装配多台主机

自动化:装系统、配置各种服务

远程实现:不需要光盘、U 盘等物理安装介质

PXE 网络:Pre-boot eXecution Environment

预启动执行环境,在操作系统之前运行

可用于远程安装

工作模式

PXE client 集成在网卡的启动芯片中

当计算机引导时,从网卡芯片中把 PXE client 调入内存执行,获取 PXE server 配置、显示菜单,根据用户选择将远程引导程序下载到本机运行

服务组件

DHCP 服务：分配 IP 地址、定位引导程序

TFTP 服务：提供引导程序下载

HTTP 服务(或 FTP/NFS)：提供 yum 安装源

客户机应具备的条件：

网卡芯片必须支持 PXE 协议

主板支持从网卡启动

搭建 PXE 网络装机服务器
1.dhcp：ip 地址、next-server、filename
2.tftp：pxelinux.0、pxelinux.cfg/default、vesamenu.c32、splash.png、vmlinuz、initrd.img、ks=http://192.168.4.7/ks.cfg
3.httpd：http://192.168.4.7/rhel7

一、配置 dhcp 服务 pxe 的设置
1.修改配置文件/etc/dhcp/dhcpd.conf
```
subnet 192.168.4.0 netmask 255.255.255.0 {
  range 192.168.4.10 192.168.4.100;
  option domain-name-servers 8.8.8.8;
  option routers 192.168.4.254;
  next-server   192.168.4.7;      #指定下一个服务器地址
  filename   "pxelinux.0";        #pxelinux.0：网卡引导文件（安装说明书）二进制的文件
}
```

2.重起 dhcpd 服务
```
[root@svr7 ~]# systemctl restart dhcpd
```

二、搭建 tftp 服务
tftp:简单文件传输协议,默认端口:69
默认共享的路径：/var/lib/tftpboot
1.安装 tftp-server
```
[root@svr7 ~]# yum -y install tftp-server
```

2.启动 tftp 服务
```
[root@svr7 ~]# systemctl restart tftp
[root@svr7 ~]# systemctl enable tftp
```

3.部署 pxelinux.0 引导文件
```
[root@svr7 ~]# yum provides  */pxelinux.0   #查询什么软件包产生 pxelinux.0
[root@svr7 ~]# yum -y install  syslinux
[root@svr7 ~]# rpm -ql syslinux | grep pxelinux.0  #查询软件包安装清单
[root@svr7 ~]# cp /usr/share/syslinux/pxelinux.0 /var/lib/tftpboot/
```

4.部署默认菜单文件
```
[root@svr7 ~]# mkdir /var/lib/tftpboot/pxelinux.cfg
[root@svr7 ~]# cp /dvd/isolinux/isolinux.cfg /var/lib/tftpboot/pxelinux.cfg/default
[root@svr7 ~]# chmod u+w /var/lib/tftpboot/pxelinux.cfg/default     #方便接下来编辑，不需要强制保存
```

5.部署 图形的模块 与 背景图片

```
[root@svr7 ~]# cp  /dvd/isolinux/vesamenu.c32  /dvd/isolinux/splash.png /var/lib/tftpboot/      #拷贝图形模块和背景图片
```

6.部署启动内核与驱动程序

```
[root@svr7 ~]# cp /dvd/isolinux/vmlinuz /dvd/isolinux/initrd.img /var/lib/tftpboot/  #拷贝驱动内核和初始化文件
[root@svr7 ~]# ls /var/lib/tftpboot/        #确认文件,其中 splash.png 为可选拷贝
 initrd.img  pxelinux.cfg  vesamenu.c32
 pxelinux.0  splash.png    vmlinuz
```

7.修改菜单文件

```
[root@svr7 ~]# vim  /var/lib/tftpboot/pxelinux.cfg/default
default vesamenu.c32    #默认加载图形的模块
timeout 600             #读秒时间为 60 秒
........
menu background splash.png      #指定背景图片
menu title NSD1801 PXE Server ! #标题信息
......
label linux
  menu label Install ^1.RHEL7   #显示菜单的内容
  kernel vmlinuz                #加载内核
  append initrd=initrd.img      #加载初始化环境

label local
    menu label Boot from ^2.local drive
    menu default                #设置为默认选项
    localboot 0xffff            #从硬盘启动
```

备注：如果需要引导安装多个系统，可在/var/lib/tftpboot/下再建一个子目录 ，将 initrd.img、vmlinuz 这 2 个文件拷贝到该目录下，并在 default 中添加相对路径

三、搭建 httpd 服务

1.安装 httpd 软件包

2.重起 httpd 服务，设置开机自启动

3.利用 httpd 服务共享光盘所有内容

```
[root@svr7 ~]# mkdir /var/www/html/rhel7
[root@svr7 ~]# mount /dev/cdrom  /var/www/html/rhel7
```

四、配置无人值守安装，应答文件的生成

1.安装图形的工具，system-config-kickstart

```
[root@svr7 ~]# yum -y install system-config-kickstart
```

2.检查 yum 仓库的标识是否为[development]
3.运行

```
[root@svr7 ~]# system-config-kickstart  #配置完后，默认保存为/root/ks.cfg
[root@svr7 ~]# vim /root/ks.cfg         #事后想要修改可直接改配置文件
```

五、利用 httpd 服务共享 ks.cfg 应答文件

```
[root@svr7 ~]# cp /root/ks.cfg   /var/www/html/
```

六、修改默认菜单文件,指定 ks.cfg 应答文件

```
[root@svr7 ~]# vim /var/lib/tftpboot/pxelinux.cfg/default
  在最后一行 append initrd=initrd.img 追加 ks=http://192.168.4.7/ks.cfg
```

Dnsmasq+网络 Yum 源搭建 PXE：
dnsmasq：一个小巧且方便地用于配置 DNS 和 DHCP 的工具，适用于小型网络。提供 DNS 缓存和 DHCP 服务功能，自带 PXE 服务器
装包：[root@pxe ~]# yum -y install dnsmasq
检查语法：

```
[root@pxe ~]# dnsmasq --test
```

该软件提供 DNS+DHCP+TFTP 服务：

```
[root@pxe ~]# vim /etc/dnsmasq.conf
    10:port=53                                          #指定 DNS 端口，设置为 0
表示关闭 DNS
    46:resolv-file=/etc/resolv.conf                     #上级 DNS 服务器
    79:#address=/www.baidu.com/192.168.1.254            #自定义域名解析，测试用
    111:listen-address=192.168.1.15                     #定义 dnsmasq 监听的地
址，指向本机的 IP 地址
    124:bind-interfaces                                 #开启 DNS
    157:dhcp-range=192.168.1.50,192.168.1.150,12h       #dhcp 池，不指定子网掩码
则根据 IP 地址自动生成
    334:dhcp-option=option:router,192.168.1.254         #开启默认路由
    442:dhcp-boot=pxelinux.0                            #PXE 引导
    499:enable-tftp                                     #开启 tftp
    502:tftp-root=/var/ftpd                             #tftp 根路径
    669:conf-dir=/etc/dnsmasq.d,.rpmnew,.rpmsave,.rpmorig   #子配置文件目录
[root@pxe ~]# systemctl restart dnsmasq.service
[root@pxe ~]# mkdir /var/ftpd
[root@pxe ~]# cp /usr/share/syslinux/pxelinux.0 /var/ftpd
[root@pxe ~]# mkdir /var/ftpd/pxelinux.cfg
[root@pxe pxelinux.cfg]# wget ftp://192.168.1.254/centos7/isolinux/isolinux.cfg
-O default
[root@pxe ftpd]# wget ftp://192.168.1.254/centos7/isolinux/vesamenu.c32
[root@pxe ftpd]# wget ftp://192.168.1.254/centos7/isolinux/initrd.img
[root@pxe ftpd]# wget ftp://192.168.1.254/centos7/isolinux/vmlinuz
```

```
[root@pxe ftpd]# vim pxelinux.cfg/default
    label linux
      menu label ^Install CentOS 7
      menu default
      kernel vmlinuz
      append initrd=initrd.img ks=ftp://192.168.1.254/ks.cfg
ftp://192.168.1.254/ks.cfg
    #platform=x86, AMD64, or Intel EM64T
    #version=DEVEL
    # Install OS instead of upgrade
    install
    # Keyboard layouts
    keyboard 'us'
    # Root password
    rootpw --iscrypted
$6$3LhZ1SHMfXNmHv7u$Sa3LY8QW/AoqPNxtz3Llr0P0P8vkYejSBp/Ont/.FyGcCUtUUBbbj5DL1MG.
BfBNRfP4zXccJeKMKbxCJw58./
    # System timezone
    timezone Asia/Shanghai
    # Use network installation
    url --url="ftp://192.168.1.254/centos7"
    # System language
    lang en_US.UTF-8
    # Firewall configuration
    firewall --disabled
    # System authorization information
    auth  --useshadow  --passalgo=sha512
    # Use text mode install
    text
    # Installation logging level
    logging --level=warning
    # Run the Setup Agent on first boot
    firstboot --disable
    # SELinux configuration
    selinux --disabled
    # Do not configure the X Window System
    skipx
    # Network information
    network  --bootproto=dhcp --device=eth0 --ipv6=auto --activate
    network  --hostname=localhost

    # Reboot after installation
    reboot
    # System bootloader configuration
```

```
bootloader --location=mbr
# Clear the Master Boot Record
zerombr
# Partition clearing information
clearpart --all --initlabel
# Disk partitioning information
part /boot --asprimary --fstype=xfs --size=512
part /     --asprimary --fstype=xfs --size=1 --grow

%packages --nobase
@Core --nodefaults
-iwl3160-firmware
-iwl6000g2b-firmware
-iwl2030-firmware
-iwl7265-firmware
-iwl1000-firmware
-iwl4965-firmware
-iwl2000-firmware
-iwl3945-firmware
-alsa-tools-firmware
-aic94xx-firmware
-iwl135-firmware
-iwl7260-firmware
-iwl6050-firmware
-iwl6000g2a-firmware
-iwl5000-firmware
-ivtv-firmware
-iwl100-firmware
-iwl5150-firmware
-iwl105-firmware
-iwl6000-firmware
-alsa-firmware
-postfix
-audit
-tuned
chrony
psmisc
net-tools
screen
vim-enhanced
tcpdump
lrzsz
ltrace
strace
```

```
    traceroute
    whois
    bind-utils
    tree
    mlocate
    rsync
    lsof
    lftp
    patch
    diffutils
    cpio
    time
    nmap
    socat
    man-pages
    -rpm-build
    %end

    %pre
    %end

    %post --interpreter=/bin/bash
    rm -f /etc/yum.repos.d/*.repo
    cat >/etc/yum.repos.d/local.repo <<'EOF'
    [local_repo]
    name=CentOS-$releasever - Base
    baseurl=ftp://192.168.1.254/centos7
    enabled=1
    gpgcheck=1
    gpgkey=ftp://192.168.1.254/centos7/RPM-GPG-KEY-CentOS-7
    EOF
    yum erase -y NetworkManager NetworkManager-libnm kexec-tools
firewalld-filesystem polkit
    sed 's,^CRONDARGS=.*,&"-m off",' -i /etc/sysconfig/crond
    sed 's,^\(OPTIONS=\).*,\1"-4",'  -i /etc/sysconfig/chronyd
    sed 's,^server .*,&\ncmdallow 127.0.0.1,' -i /etc/chrony.conf
    sed 's,^#\(terminfo xterm \x27is.*\),\1\nterm xterm,' -i /etc/screenrc
    echo -e "::1\t\tlocalhost localhost.localdomain localhost6
localhost6.localdomain6" >/etc/hosts
    echo -e "127.0.0.1\tlocalhost localhost.localdomain localhost4
localhost4.localdomain4" >>/etc/hosts
    echo -e 'export TZ='Asia/Shanghai'
PYTHONSTARTUP="/usr/lib64/python2.7/pystartup.py"
TMOUT=7200' >/etc/profile.d/environ.sh
```

```
    echo -e "blacklist acpi_pad\nblacklist
power_meter" >/etc/modprobe.d/blacklist.conf
    cat >/usr/lib64/python2.7/pystartup.py <<'EOF'
    #!/usr/bin/python
    # -*- coding:utf_8 -*-
    #from __future__ import print_function
    from rlcompleter import readline
    readline.parse_and_bind("tab: Complete")
    EOF
    cat >/etc/sysctl.d/70-system.conf <<'EOF'
    net.ipv4.ip_forward = 1
    net.ipv4.ip_default_ttl = 255
    net.ipv6.conf.all.disable_ipv6 = 1
    net.ipv6.conf.default.disable_ipv6 = 1
    net.ipv6.conf.lo.disable_ipv6 = 0
    net.ipv4.conf.default.rp_filter = 1
    net.ipv4.conf.all.arp_ignore = 1
    net.ipv4.conf.all.arp_announce = 2
    kernel.sysrq = 16
    vm.swappiness = 0
    EOF
    # config vimrc
    cat >>/etc/vimrc<<'EOF'
    set wrapscan
    set noautoindent
    set showmatch
    set binary
    set noswapfile
    set ignorecase          " Do case insensitive matching
    set foldmethod=syntax
    set foldlevel=100
    filetype plugin off
    EOF
    sed -i '35c StrictHostKeyChecking no' /etc/ssh/ssh_config
    cat >/etc/sysconfig/network-scripts/ifcfg-eth0 <<'EOF'
    # Generated by dracut initrd
    DEVICE="eth0"
    ONBOOT="yes"
    IPV6INIT="no"
    IPV4_FAILURE_FATAL="no"
    NM_CONTROLLED="no"
    TYPE="Ethernet"
    BOOTPROTO="dhcp"
    EOF
```

%end

KS 文件说明：

keyboard：指定键盘，通常是 us

lang en_US.UTF-8：系统默认使用语言，通常使用英文

timezone：指定时区

rootpw：设置 root 密码，使用密文

auth --enableshadow --passalgo=sha512：密码加密方式

cdrom 或 url：安装源

network --bootproto=dhcp --device=eth0 --ipv6=auto --activate：指定网卡

network --hostname=localhost.localdomain：指定主机名

part：配置分区挂载目录、文件系统、硬盘、分区大小等

%packages...%end：软件包，-开头的表示不安装

iwl...firmware 各种无线网卡驱动

alsa-firmware 声卡驱动

audit 审计系统

Postfix 邮件

psmisc 进程树命令

Screen 分屏

Vim-enhanced 文本编辑

Tcpdump 扫描

Traceroute 查看路由路径

Socat 在两个数据流之间建立通道，可以用于反向渗透防火墙

%post...%end：脚本，用于初始化系统

rsync 同步服务

同步与复制的差异：

复制:完全拷贝源到目标

同步:增量拷贝,只传输变化过的数据

命令：

rsync [选项...] 源目录 目标目录 #将整个源目录本身同步到目标目录

rsync [选项...] 源目录/ 目标目录 #只同步目录下的数据到目标目录

选项：

-n:测试同步过程,不做实际修改

--delete:删除目标文件夹内多余的文档

-a:归档模式,相当于-rlptgoD

-v:显示详细操作信息

-z:传输过程中启用压缩/解压

[root@svr7 ~]# rsync -avz /dir1/ /nsd/ #增量同步

[root@svr7 ~]# rsync -avz --delete /dir1/ /nsd/ #同步的同时删除多余的文档

rsync+SSH 同步

下行: rsync [参数] user@host（IP）:远程目录 本地目录

上行: rsync [参数] 本地目录 user@host（IP）:远程目录

```
[root@svr7 ~]# rsync -avz --delete /dir1/  root@192.168.4.207:/opt/
```

实时同步

SSH 无密码的验证,部署公私钥

1.生成公私钥

```
[root@svr7 ~]# ssh-keygen                    #生成密钥
[root@svr7 ~]# ssh-keygen -N '' -f ~/.ssh/id_rsa    #-N 表示密钥密码,-f 表示存放位置
[root@svr7 ~]# ls /root/.ssh/                #查看密钥
```

2.传递公钥

```
[root@svr7 ~]# ssh-copy-id root@192.168.4.207
```

3.验证

```
[root@svr7 ~]# ssh root@192.168.4.207
```

二、安装 inotify-tools，监控目录内容的变化

inotifywait [选项] 目标文件夹

常用命令选项：

-m：持续监控(捕获一个事件后不退出)

-r：递归监控、包括子目录及文件

-q：减少屏幕输出信息

-qq：屏幕不输出信息

-e：指定监视的事件类别。例如：modify,move,create,delete,attrib.....

三、目录内容变化，立即同步

while 循环

格式：

```
while [条件]
do
      循环重复执行的语句
done
```

```
[root@svr7 /]# vim  /root/rsync.sh
 #!/bin/bash
 while inotifywait -rqq /dir1/
 do
  rsync -az --delete /dir1/  root@192.168.4.207:/opt
 done
```

安装 cobbler

1.解压 cobbler.zip 包，cobbler 主程序、工具包等

```
[root@room9pc01 /]# unzip /root/桌面/Cobbler/cobbler.zip -d /
[root@room9pc01 /]# ls  /cobbler
```

2.挂载/iso/CentOS-7-x86_64-DVD-1708.iso

```
[root@room9pc01 ~]# mount /iso/CentOS-7-x86_64-DVD-1708.iso /dvd/
[root@room9pc01 ~]# ls  /dvd/
```

3.书写 Yum 客户端文件

```
[root@room9pc01 ~]# vim /etc/yum.repos.d/nsd.repo
[development]
name=dvd
baseurl=file:///dvd
enabled=1
gpgcheck=0
[root@room9pc01 ~]# yum repolist
```

4.为真机永久挂载/iso/CentOS-7-x86_64-DVD-1708.iso

```
[root@room9pc01 ~]# vim /etc/fstab
```

5.安装 cobbler 主程序、工具包等

```
[root@room9pc01 /]# yum -y install /cobbler/*.rpm
```

cobbler 网络装机部署

1.安装软件：

```
cobbler          #cobbler 主程序包
cobbler-web      #cobbler 的 web 服务包
pykickstart      #cobbler 检查 kickstart 语法错误
httpd            #Apache web 服务
dhcp             #Dhcp 服务
tftp-server      #tftp 服务
```

2.配置 cobbler

```
[root@svr7 /]# vim /etc/cobbler/settings
修改：
next_server:  192.168.4.254 #设置下一个服务器 IP 地址
server: 192.168.4.254       #设置 cobbler 服务器 IP
manage_dhcp: 1              #打开 cobbler 管理 dhcp 服务功能
pxe_just_once: 1            #安装目录增加 local，防止客户端重复安装操作系统
```

3.配置 cobbler 的 dhcp

```
[root@svr7 /]# vim /etc/cobbler/dhcp.template   #修改 DHCP 配置
:%s /192.168.1/192.168.4/g
[root@svr7 /]# vim /etc/sysconfig/dhcpd
增加一行：DHCPDARGS=private1    #表示 dhcp 将只在 private1 网络接口上提供 DHCP 服务
```

4.绝对路径解压 cobbler_boot.tar.gz   #众多的引导文件

[root@room9pc01 /]# tar -xPf /root/桌面/Cobbler/cobbler_boot.tar.gz

5.启动相关服务

```
[root@svr7 /]# systemctl restart cobblerd
[root@svr7 /]# systemctl enable cobblerd
[root@svr7 /]# systemctl restart httpd
[root@svr7 /]# systemctl enable httpd
[root@svr7 /]# systemctl restart tftp
[root@svr7 /]# systemctl enable tftp
[root@svr7 /]# systemctl restart rsyncd
[root@svr7 /]# systemctl enable rsyncd
```

6.同步刷新 cobbler 配置

```
[root@svr7 /]# cobbler sync
[root@svr7 /]# firefox https://192.168.4.254/cobbler_web
```

7.导入安装镜像数据：cobbler import --path=挂载点 --name=导入系统命名

```
[root@Cobbler ~]# mount /dev/cdrom  /dvd
[root@room9pc01 /]# cobbler list        #查看有哪些系统
[root@room9pc01 /]# cobbler import --path=/dvd  --name=CentOS7  #挂载centos
[root@room9pc01 /]# cobbler import --path=/rhel7  --name=RedHat7    #挂载redhat
```

PS：cobbler 导入的镜像放在：/var/www/cobbler/ks_mirror

```
[root@room9pc01 /]# cobbler profile report   #查看 cobbler 导入信息
```

默认 kickstart 文件存放位置：/var/lib/cobbler/kickstarts/

```
[root@cobbler ~]# cobbler list
```

8.修改 kickstart 文件：

```
[root@cobbler ~]# cobbler profile edit --name=CentOS7-x86_64 --kickstart=/var/lib/cobbler/kickstarts/CentOS-7.3-x86_64.cfg
[root@cobbler ~]# cobbler profile report
```

netstat 显示网络

参数：

- -a 显示所有
- -n 以数字形式显示
- -t TCP
- -u UDP
- -p 显示进程名

ss 等同于 netstat

```
[root@fzr ~]# ss -ant|awk '$1~/ESTAB/{A[gensub(":.*","","",$NF)]++}END{for(a in A)print a"\t"A[a]}'  #查看连接的主机
```

tac 从下到上显示文件

格式：tac [选项] [文件]

选项：-s 字符串 ：使用指定字符串代替换行作为分隔标志

dmidecode ：查看硬件设备信息

格式：dmidecode 参数 设备类型代码

常用参数：-t

设备类型代码：

0 BIOS
1 System
2 Base Board
3 Chassis
4 Processor
5 Memory Controller
6 Memory Module
7 Cache
8 Port Connector
9 System Slots
10 On Board Devices
11 OEM Strings
12 System Configuration Options
13 BIOS Language
14 Group Associations
15 System Event Log
16 Physical Memory Array
17 Memory Device
18 32-bit Memory Error
19 Memory Array Mapped Address
20 Memory Device Mapped Address
21 Built-in Pointing Device
22 Portable Battery
23 System Reset
24 Hardware Security
25 System Power Controls
26 Voltage Probe
27 Cooling Device
28 Temperature Probe
29 Electrical Current Probe
30 Out-of-band Remote Access
31 Boot Integrity Services
32 System Boot
33 64-bit Memory Error
34 Management Device

35 Management Device Component
36 Management Device Threshold Data
37 Memory Channel
38 IPMI Device
39 Power Supply

示例：

[root@room9pc01 ~]# dmidecode | grep -i power
[root@fzr ~]# dmidecode -t 1

watch 动态查看

格式：watch -n 秒数 命令

[root@apache ~]# watch -n 1 uptime

网站压力测试工具：

ab：

格式：ab 参数 网址

参数：

-c：同时连接数
-n：访问总量

示例：[root@apache ~]# ab -c 40 -n 400 192.168.1.15/testmysql.php

Wrk：

源码包：需要 gcc 环境，make 后即可使用

格式：wrk 参数 <url>

参数：

-t 线程数
-c 连接数
--timeout 超时时间
-d 持续时间

示例：[root@nginx wrk]# ./wrk -c 20 -t 20 -d 1s http://192.168.1.18

WebBench：

源码包安装：make && make install

格式：webbench 参数 URL

参数：

-c 并发数
-t 秒数

示例：[root@apache ~]# webbench -c 20 -t 5 http://192.168.1.15/

修改启动文件以限制服务：

存放目录：/usr/lib/systemd/system/

适用于所有使用 systemctl 启动的文件

更多参数可以查看 man systemd.exec

对 cpu 负载进行限制：

LimitCPU：对 cpu 时间进行限制，单位是秒，超过时间后杀死程序。注意：CPU 时间与自然时间不同

例如：针对一些死循环进程：LimitCPU=10

对最大进程数进行限制：

例如：针对 httpd：LimitNOFILE=32

系统资源限制：

配置文件：/etc/security/limits.conf

#<domain>	<type>	<item>	<value>
# 对象	类型：硬、软	条目	值
*	soft	nofile	100000
*	hard	nofile	100000

命令行：ulimit 参数

参数：-a 查看所有规则

条目：

corefile：对正在运行的进程保存内存镜像，常用于排错，默认值为 0，表示禁止

data：限制进程的数据使用的内存大小

补充：进程的内存空间结构：

Text 文本:存储可执行的代码

数据:Data 初始化数据(如变量),BSS 初始化 0 数据(初始化数据为 0)

Heap(堆)通过内存的 malloc 方法,动态内存分配(一般放数据)

Stack(栈)(存储局部变量,函数等)

stack：限制进程使用堆栈段，即内存

fsize:限制文件的最大大小

参数：-f blocks

注：每个 blocks=512B

memlock:锁定内存地址最大空间

用途：需要将有些程序的数据锁定在内存,以提供更高的响应速度

参数：-l

nofile：可以打开的最大文件数量，包括文件描述符

参数：-n

rss：限制进程所使用的最大物理内存

参数：-m

as：限制进程所使用的虚拟内存

参数：-v

cpu:限制进程能使用的 CPU 时间

nproc：最大进程数量(仅对普通用户有效)

参数：-u

priority：限制优先级的取值范围(仅对普通用户有效)

nice：限制 nice 程序本身的取值范围(仅对普通用户有效)

未启动的程序在启动时，通过 nice 命令调整进程的优先级。数字越小，优先级越高，默认取值范围-20～19

示例：nice -n -10 sleep 5&

已启动的程序通过 renice 调整

示例：renice -n -5 4146

maxlogins：单个用户最大登陆次数

maxsyslogins：最大普通用户同时登录数

umask 设置限制新建文件权限的掩码。

常用参数：-p　　：显示 umask 的值

-S　：显示最终创建的文件的权限

新值　　：设置 umask 的值

u= ,g= ,o=　:设置 umask 的值

登陆系统默认执行：

/etc/profile

/$USER/bash_profile

/$USER/.bashrc

/etc/bashrc

xargs 给其他命令传递参数的一个过滤器，也是组合多个命令的一个工具。默认参数是 echo

参数：

-0 ：　将\x00 作为定界符

-d ：定界符设置

-n 个数：以此传递 n 个参数

-I 替换符：使用替换符将变量替换

-i ：变量替换，默认使用{}

-a ：从文件读取数据，而不是标准输入

-l ：逐行读取

恢复系统：

1.使用打包的方式备份系统：

```
[root@host5 ~]# mkdir /backup
[root@host5 ~]# cd /backup
[root@host5 backup]# tar -cvf back.tar --exclude=/proc/* --exclude=/tmp/* --exclude=/sys/* --exclude=/backup/* --exclude=/run/* /*
[root@host5 backup]# scp back.tar 192.168.4.100:~
```

2.模拟删除系统：

```
[root@host5 ~]# dd if=/dev/zero of=/dev/vda bs=1M count=1
```

3.还原系统：

方法：

使用 U 盘引导系统还原

使用光盘的救援模式花园

使用 PXE 服务器还原

搭建 PXE 服务器:

```
[root@host100 ~]# yum -y install dhcp
[root@host100 ~]# vim /etc/dhcp/dhcpd.conf
[root@host100 ~]# systemctl start dhcpd
[root@host100 ~]# yum -y install tftp-server
[root@host100 ~]# systemctl start tftp
```

4.使用 pxe 启动，进入救援模式，重新格式化硬盘，将备份的 tar 包拷贝回并解压在/a 下

选择：troubleshooting
选择：rescue a Red Hat Enterprise Linux system
选择：1 continue 进入救援模式
fdisk /dev/vda
/dev/vdb1：/boot
/dev/vdb2：swap
/dev/vdb3：/
mkfs.xfs /dev/vdb1
mkswap /dev/vdb2
mkfs.xfs /dev/vdb3
mkdir /a
mount /dev/vdb3 /a
mount /dev/vdb1 /a/boot
cd /a
scp 192.168.4.100:~/back.tar .
tar -xf back.tar

5.查看并修改启动文件：

vim /boot/grub2/grub.cfg

启动文件：

linux16 /vmlinuz-0-rescue-b938ec788cdd4b629b825fd71c2cc706
initrd16 /initramfs-0-rescue-b938ec788cdd4b629b825fd71c2cc706.img

修改：

root=/dev/vdb3

6.修改开机挂载文件：

vim /etc/fstab
/dev/vdb3 / xfs defaults 0 0
/dev/vdb1 /boot xfs defaults 0 0
/dev/vdb2 swap swap defaults 0 0

7.将/a 变成根分区

chroot /a

8.安装 grub 到 MBR:

grub2-install /dev/vda

9.修改 ssh 文件的属组为 ssh_keys

```
[root@fzr ssh]# ls -l *key
-rw-r-----. 1 root ssh_keys  227 4月  29 23:45 ssh_host_ecdsa_key
-rw-r-----. 1 root ssh_keys  387 4月  29 23:45 ssh_host_ed25519_key
-rw-r-----. 1 root ssh_keys 1679 4月  29 23:45 ssh_host_rsa_key
```

shell ：在 Linux 内核与用户之间的解释器程序；负责向内核翻译及传达用户、程序指令
bash ：具体的解释器，通常指/bin/bash

脚本：可以运行文本文件，可以实现某种功能；提前设计可执行语句,用来完成特定任务的文件；将命令都写入到文本文件，然后赋予文本文件执行权限(命令的堆积)
shell 的使用方式：非交互式、需要提前设计、智能化难度大、批量执行、效率高、方便在后台悄悄运行
切换 shell 环境：通过 usermod、useradd 更改登陆 shell；手工执行目标 shell 程序
bash 的基本特性：命令行环境回顾、快捷键、tab 补齐、历史命令、别名、标准输入输出、重定向、管道

一个规范的 Shell 脚本构成包括：
脚本声明（需要的解释器、作者信息等）
注释信息（步骤、思路、用途、变量含义等）
可执行语句（操作代码）

简单脚本技巧：使用 | 管道操作：将前一条命令的标准输出交给后一条命令处理

Shell 脚本的执行方式：
作为“命令字”：指定脚本文件的路径，前提是有 x 权限。运行时开启子进程。
[root@server0 ~]# chmod +x /root/hello.sh #给 hello.sh 统一添加 x 权限
作为“参数”：使用 bash、sh、source 来加载脚本文件。bash、sh 运行时开启子进程；source 运行时不开启子进程。
[root@server0 ~]# bash /root/hello.sh

变量：会变化的量，以不变的名称，存储可以变化的值（容器）,方便以固定名称重复使用某个值，提高对任务需求、运行环境变化的适应能力
变量定义：变量名=变量值
查看变量：通过$变量名 可输出变量值；使用 env 可查看所有环境变量；使用 set 可查看所有变量（包括 env 能看到的环境变量）
引用变量:$变量名 或 ${变量名}
撤销变量：使用 unset 命令
注意事项：
若指定的变量名已存在,相当于为此变量重新赋值
等号两边不要有空格
变量名只能由字母、数字、下划线组成,区分大小写
变量名不能以数字开头,不要使用关键字和特殊字符

shell 环境变量：$USER、$UID、$PWD、$PWD、$HOME、$HOSTNAME、$SHELL、$RANDOM、$PS1
位置变量：在执行脚本时写在脚本后面的命令行参数
预定义变量：
$0 ：脚本的名称
$1 ：第一个参数，位置变量
$2 ：第二个参数，位置变量
$* ：所有位置变量的值
$# ：位置变量的个数

$$ ：当前进程的进程号

$? ：返回上一个命令的状态码;0 表示正常,其他值异常,shell 脚本编写时用 exit n 控制输出的值

双引号”“的应用：界定一个完整字符串。
单引号’’的应用：界定一个完整的字符串，并且可以实现屏蔽特殊符号的功能。
反撇号``或$()的应用：可以将命令执行的标准输出作为字符串存储，因此称为命令替换。
${}：将变量和后面的输入区分开
$[] ：数学运算

read ：产生交互的方式，将用户从键盘上的输入，赋予一个变量来储存

格式：read -p ’内容’ 变量 #-p ：屏幕输出提示内容

```
[root@server0 ~]# vim /root/user.sh
    #!/bin/bash
    read  -p  '请输入您要创建的用户名:'    abc
    useradd  $abc  &> /dev/null
    echo  $abc 用户创建成功
    echo 123  |  passwd  --stdin  $abc  &> /dev/null
    echo  $abc 用户密码设置成功
```

回显功能：

关闭：stty -echo

恢复：stty echo

```
[root@fzr ~]# cat test.sh
    #!/bin/bash
    #version v0.2
    read -p "用户名" i
    [ -z $i ] && echo error && exit
    stty -echo
    read -p "密码" j
    stty echo
    [ -z $j ] && echo error && exit
    useradd  $i   2>>~/cuowu
    echo  $j  | passwd  --stdin  $i  >/dev/null
```

使用 export 发布全局变量：默认情况下，自定义的变量为局部变量，只在当前 Shell 环境中有效，而在子 Shell 环境中无法直接使用。

用法：export 变量

```
[root@localhost ~]# export z=1          #直接发布新变量
[root@localhost ~]# export a                #发布已定义的变量
```

整数运算运算：

expr 整数运算

格式：expr 整数 1 运算符 整数 2

注意：乘法操作应采用*或’*’转义，避免被作为 Shell 通配符；参与运算的整数值与运算操作符之间以空格分开。

$[]表达式：乘法无需转义，运算符两侧可以无空格；引用变量可省略$；计算结果替换表达式本身，可结合 echo 命令输出。

let 命令：乘法无需转义，运算符两侧不能有空格，引用变量可省略$，对变量值做运算后保存新的值，不显示结果，可以结合 echo 命令来查看

小数运算工具：

bc 交互式运算：先执行 bc 命令进入交互环境，然后再输入需要计算的表达式，quit 退出交互计算器，不支持引用变量。

bc 非交互式运算：将需要运算的表达式通过管道操作交给 bc 运算，支持引用变量。小数位的长度可采用 scale=N 限制，也受参与运算的数值的小数位影响

条件测试：

使用“test 表达式”或者[表达式]都可以，表达式两边及中间至少要留一个空格。

条件测试操作本身不显示出任何信息，可以在测试后查看变量$?的值来做出判断，或者结合&&、||等逻辑操作显示出结果。

A && B ：当 A 成功时，执行 B； 当 A 失败时，B 为无效命令

A || B ：当 A 失败时，执行 B；当 A 成功时，B 为无效命令

应用：[判断]&& 命令 ；[判断]|| 命令 ；[判断]|| 命令 && 命令 .....

字符串比较：

== 比较两个字符串是否相同

!= 比较两个字符串是否不相同

-z 检查变量的值是否未设置（空值）

! 取反

整数值的比较：

-gt ： 大于

-ge ： 大于等于

-eq ： 等于

-ne ： 不等于

-lt ： 小于

-le ： 小于等于

检查文件状态：

-e：判断文档是否存在

-d：判断目录是否存在

-f：判断文件是否存在

-r：判断是否存在并且当前用户有读取权限

-w：判断是否存在并且当前用户有写入权限

-x：判断是否存在并且当前用户有执行权限

多个条件/操作的逻辑组合：

&&：逻辑与

[[条件 A] && [条件 B]] 给定条件必须都成立，整个判断结果才为真

||：逻辑或

[[条件A]] || [条件B]]　只要其中一个条件成立，则整个判断结果为真

```
[root@fzr ~]# lscpu | grep GenuineIntel >/dev/null && echo 英特尔 || echo AMD
```

if选择结构：

if单分支的语法组成：

```
if  条件测试
    then  命令序列
fi
```

if双分支的语法组成：

```
if  条件测试
    then  命令序列1
    else  命令序列2
fi
```

if多分支的语法组成：

```
if  条件测试1
    then  命令序列1
elif  条件测试2
    then  命令序列2
 ...
    else  命令序列n
fi
```

```
[root@fzr ~]# cat gradediv.sh
#!/bin/bash
read -p 分数 score
if [ $score -ge 90 ]
then echo   神功绝世
elif [ $score -ge 80 ]
then echo 登峰造极
elif [ $score -ge 70 ]
then echo 炉火纯青
elif [ $score -ge 60 ]
then echo 略有小成
else echo 初学乍练
fi
```

```
[root@fzr ~]# cat ping.sh
#!/bin/bash
if [ ! $1 ]
then read -p IP ip
else ip=$1
fi
```

```
if [ ! $ip ]
then echo IP 呢?
exit 2
fi
ping -c 4 $ip > /dev/null
if [ $? -eq 0 ]
 then echo $ip 通
 else echo $ip 不通
fi
```

for 循环处理:

```
for  变量名  in  值列表          #值列表用空格，tab 分隔
do
  命令序列
done
```

值列表可用如下方式表示:

```
$(cat 1.txt)        #文件可用空格，tab，换行分隔
{1..n}          #循环 n 次
$(seq 5)        #seq n 列出 1~n 的数；seq n m 列出 n~m 的数；seq n step m 列出 n~m，以 step 为步长的数
```

```
[root@fzr ~]# vim for.sh
#!/bin/bash
num=$RANDOM
for i in {1..10}
do
read -p "输入" cai
if [ $num -eq $cai ]
then
echo 恭喜
exit 1
elif [ $num -gt $cai ]
then
echo 小了
else echo 大了
fi
done
echo 没有次数了 $num
```

```
[root@server ~]# vim /root/batchusers
#!/bin/bash
if [ $# -eq 0 ] ; then
echo "Usage: /root/batchusers <userfile>"
```

```
exit 1
fi
if [ ! -f $1 ] ; then
echo "Input file not found"
exit 2
fi
for name in $(cat $1)
do
useradd -s /bin/false $name
```

```
[root@fzr ~]# cat ping.sh
#!/bin/bash
for ip in $(seq 254)
do
ping -i0.1 -W1 -c 2 "176.19.4.$ip" > /dev/null
if [ $? -eq 0 ]
 then echo 176.19.4.$ip 通
 else echo 176.19.4.$ip 不通
fi
done
```

```
[root@fzr ~]# cat 99.sh
#!/bin/bash
for i in $(seq 9)
do
 for j in $(seq $i)
 do
  echo -n $j*$i=$[i*j]
  echo -n " "
 done
 echo
done
```

while 循环：

```
while [条件]
do
      循环重复执行的语句
done
```

```
[root@fzr ~]# cat cai.sh
#!/bin/bash
num=$RANDOM
i=0
j=1
while [ $i -lt 10 ]
```

```
    do
    read -p "输入" cai
    if [ ! $cai ]
    then echo 你浪费了$j 次
    let i++ j++
    continue
    fi
    if [ $num -eq $cai ]
    then
    echo 恭喜
    exit 1
    elif [ $num -gt $cai ]
    then
    echo 小了
    else echo 大了
    fi
    let i++
    done
    echo 没有次数了 $num

[root@fzr ~]# cat cai2.sh
    #!/bin/bash
    num=$RANDOM
    i=1
    j=1
    while :
    do
    read -p "输入" cai
    if [ ! $cai ]
    then echo 你浪费了$j 次,第$i 次猜错了
    let i++ j++
    continue
    fi
    if [ $num -eq $cai ]
    then
    echo 恭喜，你在第$i 次猜对了
    exit
    elif [ $num -gt $cai ]
    then
    echo 小了
    else echo 大了
    fi
    echo 这是你第$i 次猜错了
    let i++
```

```
done
```

case 分支：针对指定的变量预先设置一些可能的取值，判断该变量的实际取值是否与预设的某一个值相匹配，如果匹配上了，就执行相应的一组操作，如果没有匹配，就执行预先设置的默认操作或退出

case 分支结构：

```
case  变量值  in
模式 1)
    命令序列 1 ;;
模式 2)
    命令序列 2 ;;
    ..
*)
    默认命令序列
esac
```

```
[root@fzr ~]# cat saocaozuo.sh
#!/bin/bash
case $1 in
-n)
touch $2 ;;
-v)
vim $2;;
-r)
rm -rf $2;;
-c)
cat $2;;
*)
echo "$0 (-n|-v|-r|-c) file"
esac
```

```
[root@fzr ~]# cat caiquan.sh
#!/bin/bash
echo "猜拳开始："
echo "石头"
echo "剪刀"
echo "布"
read -p "请输入你猜的" cai
case $cai in
    石头)
    a=0;;
    剪刀)
    a=1;;
    布)
```

```
        a=2;;
        *)
        echo 你想干吗？;;
    esac
    b=$[$RANDOM%3]
    let c=$a-$b
    case $c in
        -1)
        echo 恭喜你赢了;;
        2)
        echo 恭喜你赢了;;
        0)
        echo 平局;;
        *)
        echo 你输了;;
    esac
```

函数：将一些需重复使用的操作，定义为公共的语句块。通过使用函数，可以使脚本代码更加简洁，增强易读性，提高 Shell 脚本的执行效率

格式 1：

```
function  函数名 {
    命令序列
    .. ..
}
```

格式 2：

```
函数名() {
    命令序列
    .. ..
}
```

echo -e 扩展输出：

m:0X 为样式，3X 为字体颜色，4X 为背景颜色

```
[root@fzr ~]# echo -e "\033[31;2;46mHello World\033[0m"
```

#0m：执行完后恢复默认

H:指定横纵坐标

```
[root@fzr ~]# echo -e "\033[12;15Hhello world\033[15H"
```

#15H:执行完后光标跳到第 15 行的下一行

```
[root@fzr ~]# function cecho
> {
> echo -e "\033[$1m$2\033[0m"
> }
[root@fzr ~]# cecho 31 ok
```

```
[root@fzr ~]# cecho "4;31" "hello word"
[root@fzr ~]# cecho "4;31;47" error
```

脚本中断退出功能：

continue 结束本次循环，进入下一次循环

break 可以结束整个循环

exit 结束整个脚本

子串截取的三种用法：

1.${变量名:起始位置:长度}　　#截取字符串时，起始位置编号是从 0 开始的，可以省略，默认为 0

2.expr substr "$变量名" 起始位置 长度　　#截取字符串时，起始位置编号从 1 开始

3.echo $变量名 | cut -b 起始位置-结束位置　　#截取字符串时，起始位置编号从 1 开始。起始位置、结束位置都可以省略

```
[root@fzr ~]# echo ${#phone}          #查看字符串长度
[root@fzr ~]# echo ${phone:4}         #从位置 4 开始截取到最后
[root@fzr ~]# echo ${phone::6}        #从开头开始截取 6 位
[root@fzr ~]# echo ${phone:9:6}       #从位值 9 截取到最后，超过长度的部分不显示
[root@fzr ~]# expr substr "$phone" 1 6    #从第 1 位截取到第 6 位
[root@fzr ~]# echo $phone | cut -b 2-     #从第 2 位截取到最后
[root@fzr ~]# echo $phone | cut -b 2-6    #截取第 2-6 位
[root@fzr ~]# echo $phone | cut -b -6     #从开头截取到第 6 位
[root@fzr ~]# echo $phone | cut -b 2,4,6  #截取第 2，4，6 位
```

子串替换的两种用法：

1.只替换第一个匹配结果：${变量名/old/new}

2.替换全部匹配结果：${变量名//old/new}

```
[root@fzr ~]# echo ${phone/88/**}
[root@fzr ~]# echo ${phone//88/**}
```

字符串掐头去尾：

1.从左向右，最短匹配删除：${变量名#*关键词}

2.从左向右，最长匹配删除：${变量名##*关键词}

3.从右向左，最短匹配删除：${变量名%关键词*}

4.从右向左，最长匹配删除：${变量名%%关键词*}

```
[root@fzr ~]# A=$(head -1 /etc/passwd)
[root@fzr ~]# echo ${A#*:}
[root@fzr ~]# echo ${A##*:}
[root@fzr ~]# echo ${A%%:*}
[root@fzr ~]# echo ${A%:*}
```

字符串初值：

${var:-word}:若变量 var 已存在且非空，则返回 $var 的值；否则返回字串“word”，原变量 var 的值不受影响。

```
[root@fzr ~]# xx=11
[root@fzr ~]# echo ${xx:-123}
[root@fzr ~]# echo ${yy:-123}
[root@fzr ~]# cat user.sh
    #!/bin/bash
    #version v0.2
    read -p "用户名" user
    [ -z $user ] && echo error && exit 1
    stty -echo
    read -p "密码" pass
    stty echo
    pass=${pass:-123}
    useradd  $user    2>>~/cuowu
    echo  $pass  | passwd  --stdin  $user  >/dev/null

[root@fzr ~]# cat jiafa
    #!/bin/bash
    a=$1
    b=$2c=$3
    if [ ! $1 ]
    then
    read -p "请输入第一个数" a
    read -p "请输入第二个数" b
    read -p "请输入第三个数" c
    fi
    [ -z $a ]&& [ -z $c ]&&echo 至少输入 2 个数字&&exit 1
    b=${b:-100}
    sum=0
    for i in $(seq $a $b $c)
    do
    let sum+=$i
    done
    echo $sum
```

shell 调试：发现引发脚本错误的原因以及在脚本源代码中定位发生错误的行。
方法：分析输出的错误信息，在脚本中加入调试语句，进入调试模式查看运行过程
基本调试技术：echo
例如：

```
#!/bin/bash
id=(1 2 3 4 5 6 7 8 9)
for ((i=0;i<${#id[@]};i++))
do
if [ $[id[i]%2] ]
```

```
    then
    unset id[i]
    fi
    done
    echo ${id[@]}
```

执行后发现，脚本执行到一半就停止了
使用 echo 后发现，由于删除了奇数导致 id 长度变短，导致没有达到预期效果
解决方法：将长度固定即可

多层管道会导致中间过程没有输出，很难定位，这时候可以在管道中加入 tee 解决
sort 对子段排序
uniq -c 统计子段个数

```
[root@fzr ~]#  ss -ant | awk '{print $1}' | uniq -c
      1 State
      5 LISTEN
      2 ESTAB
      4 LISTEN
[root@fzr ~]#  ss -ant | awk '{print $1}' | sort
    ESTAB
    ESTAB
    LISTEN
    LISTEN
    LISTEN
    LISTEN
    LISTEN
    LISTEN
    LISTEN
    LISTEN
    LISTEN
    State
```

系统有一个内置变量 LINENO，可以查看是哪一行脚本执行
可以预先指定一些调试开关，也就是调试钩子，通常使用 DEBUG，但是会牺牲脚本的性能
定义时：function DEBUG(){ true && $@; }
使用时：DEBUG echo ${i} ${id[@]}
不调试时，将 true 改成 failse 即可

查看系统信号：kill -l
SIGINT：即 ctrl+c
SIGKILL：kill 命令
shell 脚本在执行时，会产生 4 个伪信号，可以通过“trap 命令 信号”捕获：
exit：整个脚本执行完毕时
例如：

```
    #!/bin/bash
```

```
function test(){
echo "test is run,$?"
}
function foo(){
echo foo
}
foo
trap "test" exit
echo end
exit 1
```

PS：脚本在退出时，执行 test 函数，如果在该函数中加入$0，则该脚本无法关闭。

return：函数退出，必须有 trace 状态

err：当一条命令返回非零状态时

例如：

```
#!/bin/bash
function test(){
echo "test,$?"
}
function foo(){
echo foo
return 1
}
trap 'test' err
foo
echo end
```

方便捕获执行不成功的命令

debug：脚本每一条命令执行前

例如：

```
#!/bin/bash
trap 'echo "line:$LINENO,a=$a,b=$b,c=$c"' debug
a=1
b=2
c=3
exit
```

使用系统信号：

例如：

```
#!/bin/bash
trap 'echo 打死也不停' int
while :
do
echo ok
sleep 1
Done
```

可以捕获的系统信号：

SIGHUP	终止进程	终端线路挂断
SIGINT	终止进程	中断进程，ctrl+c
SIGQUIT	建立 CORE 文件	终止进程，并且生成 core 文件
SIGILL	建立 CORE 文件	非法指令
SIGTRAP	建立 CORE 文件	跟踪自陷
SIGBUS	建立 CORE 文件	总线错误
SIGSEGV	建立 CORE 文件	段非法错误
SIGFPE	建立 CORE 文件	浮点异常
SIGIOT	建立 CORE 文件	执行 I/O 自陷
SIGPIPE	终止进程	向一个没有读进程的管道写数据
SIGALARM	终止进程	计时器到时
SIGTERM	终止进程	软件终止信号
SIGTSTP	停止进程	终端来的停止信号
SIGCONT	忽略信号	继续执行一个停止的进程
SIGURG	忽略信号	I/O 紧急信号
SIGIO	忽略信号	描述符上可以进行 I/O
SIGCHLD	忽略信号	当子进程停止或退出时通知父进程
SIGTTOU	停止进程	后台进程写终端
SIGTTIN	停止进程	后台进程读终端
SIGXGPU	终止进程	CPU 时限超时
SIGXFSZ	终止进程	文件长度过长
SIGWINCH	忽略信号	窗口大小发生变化
SIGPROF	终止进程	统计分布图用计时器到时
SIGUSR1	终止进程	用户定义信号 1
SIGUSR2	终止进程	用户定义信号 2
SIGVTALRM	终止进程	虚拟计时器到时

无法捕获的系统信号：该信号直接发给内核

9：SIGKILL 杀死

19：SIGSTOP 停止

1-31 的系统信号是非实时信号

32-64 的系统信号是实时信号，支持排队

mknod 创建块文件、字符文件、FIFO 文件

选项：

b	创建(有缓冲的)区块特殊文件
C	创建(没有缓冲的)字符特殊文件
p	创建先进先出(FIFO)特殊文件

```
[root@room9pc01 ~]# mknod test p
```

调试：shopt -o

参数：-s 开启

-u 关闭

```
[root@fzr ~]# shopt -o -s functrace
```

Bash 调试

-x 跟踪脚本，每经过了一次命令后的结果用+开头列出；直接观察执行中的输出、报错信息

-n 检查语法

-c

提示符变量：

主提示符：$PS1

第二级提示符：$PS2

第三级提示符：$PS3

第四级提示符：$PS4

expect:

为交互式过程（比如 FTP、SSH 等登录过程）自动输送预先准备的文本或指令，无需人工干预。触发的依据是预期会出现的特征提示文本。

```
[root@fzr ~]# cat yuancheng.sh
    #!/bin/bash
    expect << EOF
    spawn ssh -o StrictHostKeyChecking=no 192.168.4.139          #创建交互式进程，远程时不检测密钥
    expect "password" {send "123456\n"}                          #自动发送密码
    expect "#" {send "rm -rf /etc/yum.repos.d/*\n"}              #发送命令
    expect "#" {send "cd /etc/yum.repos.d/\n"}
    expect "#" {send "wget http://192.168.4.138/rhel.repo\n"}
    expect "#" {send "yum repolist\n"}
    expect "#" {send "yum -y install httpd\n"}
    expect "#" {send "systemctl restart httpd.service\n"}
    expect "#" {send "systemctl enable httpd.service\n"}
    expect "#" {send "exit\n"}                                   #最后一行默认不执行
    EOF
[root@room9pc01 ~]# vim /etc/ssh/ssh_config +35                  #修改配置文件取消验证密码
    StrictHostKeyChecking no
```

正则表达式：

基本正则：所有软件和命令都支持

^ 行首

$ 行尾

[] 集合，取集合中任意单个字符

[a-z]

[a-Z]

[a-zA-Z0-9]

[^] 对集合取反

. 任意单个字符

```
*       前一个字符出现 0 次或多次
.*      任意字符出现 0 次或多次
\{n,m\} 前一个字符出现 n~m 次
\{n,\}  前一个字符出现 n 次以上
\{n\}   前一个字符出现 n 次
\( \)   保留
```

扩展正则：部分软件和命令不支持，grep 不识别扩展正则，需要改用 egrep 命令

```
{n,m}   前一个字符出现 n~m 次
{n,}    前一个字符出现 n 次以上
{n}     前一个字符出现 n 次
()      将多个字符组合为整体；保留
?       前面的字符出现 0 次或 1 次
+       前面的字符至少出现 1 次
|       或者
\b      单词边界
```

```
[root@fzr ~]# grep the regular_express.txt          #过滤下载文件中包含 the 关键字
[root@fzr ~]# grep -v the regular_express.txt       #过滤下载文件中不包含 the 关键字
[root@fzr ~]# grep -i the regular_express.txt       #过滤下载文件中不论大小写 the 关键字
[root@fzr ~]# grep t[ae]ste* regular_express.txt    #过滤 test 或 taste 这两个单字
[root@fzr ~]# grep oo regular_express.txt           #过滤有 oo 的字节
[root@fzr ~]# grep [^g]oo regular_express.txt       #过滤 oo 前面没有 g 的
[root@fzr ~]# grep [^a-z]oo regular_express.txt     #过滤 oo 前面没有小写字
[root@fzr ~]# grep [0-9] regular_express.txt        #过滤有数字的那一行
[root@fzr ~]# grep ^the regular_express.txt         #过滤以 the 开头的
[root@fzr ~]# grep ^[a-z] regular_express.txt       #过滤以小写字母开头的
[root@fzr ~]# grep ^[^a-Z] regular_express.txt      #过滤开头不是英文字母

[root@fzr ~]# grep "\.$" regular_express.txt        #过滤行尾结束为小数点.那一行
[root@fzr ~]# grep ^$ regular_express.txt           #过滤空白行
[root@fzr ~]# grep g..d regular_express.txt         #过滤出 g??d 的字串
[root@fzr ~]# grep ooo* regular_express.txt         #过滤至少两个 o 以上的字串
[root@fzr ~]# grep goo*g regular_express.txt        #过滤 g 开头和 g 结尾但是两个 g 之间仅存在至少一个 o
[root@fzr ~]# grep [0-9] regular_express.txt        #过滤任意数字的行
[root@fzr ~]# grep oo regular_express.txt           #过滤两个 o 的字串
[root@fzr ~]# grep 'go\{2,\}g' regular_express.txt  #过滤 g 后面接 2 个以上 o 的
```

[root@fzr ~]# grep "go\{2,5\}g" regular_express.txt #过滤 g 后面接 2 到 5 个 o,然后在接一个 g 的字串

sed 文本处理工具

用法 1：前置命令 | sed [选项] '条件指令'

用法 2：sed [选项] '条件指令' 文件

说明：

条件可以是行号或者正则

没有条件时，默认为所有条件

指令可以是增、删、改、查等指令

默认 sed 会将所有输出的内容都打印出来

默认 sed 只是通过内存临时修改文件，源文件无影响

常用选项：

-n 屏蔽默认输出

-r 让 sed 支持扩展正则

-i 让 sed 直接修改源文件

常用条件：行号，/正则表达式/

常用指令：

p(print)打印 ；

d(delete) 删除；

s(substitution) 替换关键字；

a(append) 指定行之后追加；

i(insert) 指定行之前插入；

c 替换行；

r(read) 读入；

w(write) 另存为

剪切复制粘贴

模式空间：

存放当前处理的行，将处理结果输出

若当前行不符合处理条件，则原样输出

处理完当前行再读入下一行处理

保持空间：类似于剪贴板，默认存放一个换行符\n

指令：

-H 模式空间 追加到 保持空间

-h 模式空间 覆盖到 保持空间

-G 保持空间 追加到 模式空间

-g 保持空间 覆盖到 模式空间

[root@fzr ~]# sed "3p" /etc/passwd -n #输出第 3 行

[root@fzr ~]# sed "3,6p" /etc/passwd -n #输出第 3-6 行

[root@fzr ~]# sed "3p;6p" /etc/passwd -n #输出第 3 行和第 6 行

[root@fzr ~]# sed "3,+10p" /etc/passwd -n #输出第 3 行以及接下来 10 行

[root@fzr ~]# sed "1～10p" /etc/passwd -n #输出从第 1 行开始，步长为 10 的行

[root@fzr ~]# sed -i "1,2d" test.txt #删除文件的第 1，2 行
[root@fzr ~]# sed -n '/root/p' /etc/passwd #输出包含 root 的行，正则写在 / /之间
[root@fzr ~]# sed -n '/bash$/p' /etc/passwd #输出以 bash 结尾的行
[root@fzr ~]# sed -n '$=' /etc/passwd #输出文件行数
[root@fzr ~]# sed '/user/d' a.txt #删除包含 user 的行
[root@fzr ~]# sed '/root/!d' a.txt #删除不包含 root 的行
[root@fzr ~]# sed '/^$/d' a.txt #删除所有空行
[root@fzr ~]# sed 's/2017/2018/p' sucai.txt -n #将每行第一个 2017 替换为 2018
[root@fzr ~]# sed 's/2017/2018/2p' sucai.txt -n #将每行第 2 个 2017 替换为 2018
[root@fzr ~]# sed '2s/2017/2018/gp' sucai.txt -n#将第二行的所有 2017 都替换为 2018
[root@fzr ~]# sed 's/2017//p' sucai.txt -n #将每行的第一个 2017 替换为空
[root@fzr ~]# sed "1,3s/2017/2018/3p" sucai.txt -n #将第 1-3 行的第三个 2017 替换为 2018
[root@fzr ~]# sed s#'bin/bash'#'sbin/sh'#g /etc/passwd #替换操作的分隔/可改用其他任何字符，如#，便于修改文件路径
[root@fzr ~]# sed s,'bin/bash','sbin/sh',g /etc/passwd
[root@fzr ~]# sed -nr '1s/^(.)(.*)(.)$/\3\2\1/p' /etc/passwd#将第一行的首尾字符互换
[root@fzr ~]# sed "s/[0-9]//g;s/^#+//" /etc/passwd #将文件中的所有数字和行首注释删除
[root@fzr ~]# sed -r "s/([0-9])/([\1])/g" /etc/passwd #替换所有数字加个框
[root@fzr ~]# sed -n "/^#anon/s/^#//p" /etc/vsftpd/vsftpd.conf #删除配置文件中 anon 前的注释
[root@fzr ~]# sed "2a a" sucai.txt #在第 2 行下面，追加写入 a
[root@fzr ~]# sed "2i a" sucai.txt #在第 2 行上面，追加写入 a
[root@fzr ~]# sed "2c a" sucai.txt #将第 2 行修改为 a
[root@fzr ~]# sed "r /etc/hostname" sucai.txt #在 sucai.txt 的每行的下面写入 hostname 中的内容
[root@fzr ~]# sed "w /etc/hostname" sucai.txt #将 sucai.txt 的内容存入 hostname 中
[root@fzr ~]# sed "2w /etc/hostname" sucai.txt #将 sucai.txt 的第二行的内容存入 hostname 中
[root@fzr ~]# sed "2H;4g" sucai.txt #将 sucai.txt 的第二行覆盖到第四行后面，并在前面加入一个空行
[root@fzr ~]# sed "2h;2d;4G" sucai.txt #将 sucai.txt 的第二行剪切到第四行后面

awk 工具：

基于模式匹配检查输入文本，逐行处理并输出

通常用在 shell 脚本中，获取指定的数据

单独用时，可对文本数据做统计

格式：

用法一：awk [选项] '[条件]{指令}' 文件

用法二：awk [选项] 'BEGIN{指令} {指令} END{指令}' 文件

选项和条件都是可选项，指令必须写在{}中间

选项：

-F 指定分隔符 #未指定分隔符，则默认将空格、制表符等作为分隔符

条件：

/正则表达式/

$n~/正则表达式/ #第 n 列正则匹配

$n!~/正则表达式/ #第 n 列正则不匹配

数值/字符串比较：比较符号：==(等于) !=（不等于） >（大于）>=（大于等于） <（小于） <=（小于等于）

指令：

print 是最常用的编辑指令。

若有多条编辑指令，可用分号分隔。

$n 文本的第 n 列，支持仅打印某一列

NR 文件当前行的行号

NF 文件当前行的列数

BEGIN{ } 行前处理：读取文件内容前执行，指令执行 1 次，可以没有操作文件

{ } 逐行处理：读取文件过程中执行，指令执行 n 次

END{ } 行后处理：读取文件结束后执行，指令执行 1 次

if 分支结构：

if(判断){指令}

if(判断){指令}else{指令}

if(判断){指令}else if(判断){指令}...else{指令}

数组：

定义数组的格式：数组名=(值 1 值 2 值 3);数组名[下标]=元素值

调用数组的格式：数组名[下标]

遍历数组的用法：for(变量 in 数组名){print 数组名[变量]}

```
[root@fzr ~]# awk '{print $2,$1}' sucai.txt      #先打印第二列，再打印第一列
[root@fzr ~]# df -h | awk '{print $1,$4}'        #列出各硬盘的可用空间
[root@fzr ~]# awk -F : '{print $1,"解释器",$7}' /etc/passwd   #列出用户名及对应的解释器
[root@fzr ~]# awk -F [:/] '{print $11}' /etc/passwd #以冒号和斜线做为分隔符
[root@fzr ~]# awk '{print NR}' sucai.txt         #打印每行的行号
[root@fzr ~]# awk '{print NF}' sucai.txt         #打印每行的列数
[root@fzr ~]# awk '{print $NF}' sucai.txt        #打印每行的最后一列
[root@fzr ~]# awk '/2018/{print $3}' sucai.txt  #打印含有 2018 的那行的第三列
[root@fzr ~]# awk '/Failed/{print $11}' /var/log/secure      #找出所有登陆失败的 IP
```

```
[root@fzr ~]# free | awk '/Mem/{print $4}'        #查看剩余内存
[root@fzr ~]# ifconfig | awk '/inet /{print $2}'#查看 IP 地址
[root@fzr ~]# uptime | awk '{print $NF}'          #查看 15 分钟平均 CPU 占用
[root@fzr ~]# df -h |awk '/\/$/{print $4}'        #查看根分区可用空间
[root@fzr ~]# awk 'BEGIN{x=0} /bash$/{x++} END{print x}' /etc/passwd#统计使用 bash 登陆的用户个数
[root@fzr ~]# awk -F: 'BEGIN{print "用户 UID HOME"} {print $1,$3,$6} END{print "已处理",NR}' /etc/passwd | column -t
        #第一行为列表标题，中间打印用户的名称、UID、家目录信息，最后一行提示一共已处理文本的总行数。column -t 为格式化输出
[root@fzr ~]# awk -F: '$1~/root/' /etc/passwd    #查看用户名中包含 root 的用户
[root@fzr ~]# awk -F: '$1=="root"' /etc/passwd   #查看 root 用户的信息
[root@fzr ~]# awk -F: '$3<1000{print $1}' /etc/passwd        #查看所有 UID 小于 1000 的用户
[root@fzr ~]# awk -F: '$3>=20 && $3<=100 {print $1}' /etc/passwd     #查看 UID 在 20-100 之间的用户
[root@fzr ~]# awk -F: '$3==0 || $3==1000 {print $1}' /etc/passwd     #查看 UID 是 0 和 UID 是 1000 的用户
[root@fzr ~]# seq 300 | awk '$1%3==0 && $1~/3/' #列出 300 以内能被 3 整除且数字中包含 3 的数字
[root@fzr ~]# awk -F: '{if($3<1000){x++}else{y++}} END{print x,y}' /etc/passwd
#查看系统用户和自建用户分别有多少
[root@fzr ~]# awk 'BEGIN{x[0]=4;x[5]=22;x[7]=44;for(i in x){print x[i]}}'
    #查看数组内的值
[root@fzr ~]# awk '{ip[$1]++} END{for(i in ip) {print i,ip[i]}}' /var/log/httpd/access_log    #统计次数
```

DOS(deny of service)攻击:

```
ab -c 1000 -n 100000 http://192.168.4.138/          #模拟 1000 人访问网站 10 万次
```

脚本实现一键部署 Nginx 软件（Web 服务器）：

1.脚本自动判断 yum 是否可用

2.脚本自动安装相关软件的依赖包

3.一键源码安装 Nginx 软件

```
[root@fzr ~]# cat nginx.sh
    #!/bin/bash
    n=$(yum repolist | awk '/repolist/{print $2}' |sed 's/,//g')
    [ $n -le 0 ] && echo 'yum 不可用' && exit
    yum -y install gcc openssl-devel pcre-devel
    tar -xf ~/nginx-1.10.3.tar.gz
    cd ~/nginx-1.10.3/
    ./configure >/dev/null
```

```
make >/dev/null
make install >/dev/null
/usr/local/nginx/sbin/nginx
/usr/local/nginx/sbin/nginx -V
```

设置简单命令开关查看 Nginx

```
[root@fzr bin]# cat nginx
#!/bin/bash
case $1 in
start)
/usr/local/nginx/sbin/nginx &>/dev/null
[ $? -ne 0 ] && echo 请检查状态;;
stop)
/usr/local/nginx/sbin/nginx    -s stop;;
status)
netstat -ntulp |grep nginx >/dev/null
[ $? -eq 0 ] && echo 已启动 || echo 未启动;;
*)
echo 参数错误 start,stop,status;;
esac
```

监控计算机各种参数：

```
[root@fzr ~]# cat jiankong.sh
#!/bin/bash
uptime | awk '{print "CPU 平均负载 1 5 15:",$8,$9,$10}' | sed "s,//g"
ifconfig eth0 | awk '/packets/{print $1,"的流量包",$3}'
free | awk '/Mem/{print "剩余内存",$4}'
df -h  | awk '/\/$/{print "剩余磁盘容量",$4}'
echo -n 用户数
cat /etc/passwd |wc -l
echo -n 登陆用户数
who |wc -l
echo -n 当前计算机启动的进程数量为
ps aux |wc -l
echo -n 当前计算机已安装的软件数量
rpm -qa  |wc -l
```

将密码或者用户名输错 3 次以上的 IP 加入黑名单：

```
[root@fzr ~]# cat secure.sh
#!/bin/bash
ip1=$(awk '/Invalid user/{print $10}' /var/log/secure | awk
'{ip[$1]++}END{for(i in ip){print ip[i],i}}' | awk '$1>3{print $2}')
firewall-cmd  --zone=block --add-source=$ip1 2>/dev/null
ip2=$(awk '/Failed password/&&$9!~/invalid/{print $11}' /var/log/secure
| awk '{ip[$1]++}END{for(i in ip){print ip[i],i}}' | awk '$1>3{print $2}')
firewall-cmd  --zone=block --add-source=$ip2 2>/dev/null
```

给复制添加进度条：

```
[root@fzr ~]# cat jindu.sh
    #!/bin/bash
    jindu(){
    while :
    do
        echo -ne '\033[42m \033[0m'
        sleep 0.3
    done
    }
    jindu &
    cp -r  $1  $2
    kill $!
[root@fzr ~]# alias cp="bash ./jindu.sh"
```

通过网页直接测试服务器运行状态

```
[root@fzr cgi-bin]# cat test.html
    #!/bin/bash
    echo "Context-type:text/html"
    echo ""
    ip=$(ifconfig eth0 | awk '/inet /{print $2}')
    men=$(free | awk '/Mem/{print $4}')
    cpu=$(uptime | awk '{print $NF}')
    echo "IP$ip"
    echo "内存$men"
    echo "CPU$cpu"
[root@fzr ~]# chmod +x /var/www/cgi-bin/test.html
[root@fzr ~]# curl 127.0.0.1/cgi-bin/test.html
```

查看目前有几个 IP 连接本机

```
[root@fzr ~]# netstat -atun | awk 'NR>2{print $5}' | awk
'$1!~/0.0.0.0:*/&&$1!~/:::*/' | awk -F: '{print $1}' | awk '{ip[$1]++}END{for(i in
ip){print ip[i],i}}'
```

查看 13：30-20：30 之间访问网页的数量

```
[root@fzr ~]# awk '{print $4}' /var/log/httpd/access_log |awk -F:
'$2":"$3>="13:30" && $2":"$3<="20:30"'|wc -l
```

查看访问的 IP 及次数

```
[root@fzr ~]# awk '{print $4,$1}' /var/log/httpd/access_log |awk
'$2!~/127.0.0.1/{print $2}'|awk '{ip[$1]++}END{for(i in ip){print ip[i],i}}'
```

文件描述符：0 ， 1 ， 2

0: stdin (standard in)

1: stdout (standard out)

2: stderr (standard error)

存放位置：/proc/self/fd/

查看当前终端的进程号：echo $$

当前终端的文件描述符存放位置：/proc/4853/fd/

关闭文件描述符：

n<&-　　关闭输入文件描述符号 n （n=0,1,2）

n>&-　　关闭输出文件描述符号 n

重定向：

>　标准输出，默认值为 1

<　标准输入，默认值为 0

追加重定向：

>>：追加输出

<<：设置结束标志

单行重定向：每次执行完后重定向都会被重置

将正确输出写入文件，将错误输出以正确输出的形式写入文件：命令 >文件 2<&1

将错误输出以正确输出的形式显示在屏幕上，并将正确输出写入文件：命令 2<&1 >文件

清空文件：> 文件

由输入文件将参数传递给命令，得出的结果写入输出文件：命令 < 输入文件 > 输出文件

只允许追加，不允许覆盖：set -o noclobber

恢复上一个：set +o noclobber

关闭终端：exec 0<&-

管道后端产生了子进程，子进程继承了父进程的文件描述符

exec 重定向文件描述符：

如：3<&1 ，将标准输出重定向给 3

wall：向所有终端发送信息

格式：wall 信息内容

虚拟化技术：

介绍：

虚拟化(Virtualization)是一种资源管理技术，是将计算机的各种实体资源，予以抽象、转换后呈现出来

虚拟化主要厂商及产品：

VMware：VMware Workstation，vSphere(vcenter+ESX)

Microsoft：VirtualPC

Citrix：Xen

Oracle：VirtualBox

开源软件：KVM(Kernel-based Virtual Machine)集成在 Linux 的各个主要发行版本内核中

主要软件组：

虚拟化平台(Virtualization Platform)

虚拟化主机(Virtualization Host)

虚拟化客户端(Virtualization Client)

虚拟化服务必备软件：

qemu-kvm：为虚拟机提供底层支持

libvirt-client：提供 virsh 软件

libvirt-daemon：libvirt 服务进程

libvirt-daemon-driver-qemu：virsh 连接 qemu 的驱动

虚拟化服务可选软件：

virt-install：系统安装工具

virt-manager：图形管理工具

virt-p2v：物理机迁移 kvm 工具

virt-v2v：其他虚拟机迁移 kvm 工具

虚拟机的磁盘镜像文件格式：RAW、QCOW2

COW(Copy On Write)：写时复制

映射原始盘的数据内容，当修改数据时，在修改之前自动将数据存入前端盘，不修改原始盘

virsh 命令工具：

格式：virsh 控制指令 [虚拟机名称] [参数]

常用控制指令：

nodeinfo：查看 KVM 节点(服务器)信息

list [--all]：列出虚拟机，--all 会列出未启动的

net-list [--all]：列出虚拟网络

net-info 网络名：查看网络信息

net-define：根据 xml 文件定义虚拟网络

net-undefine：删除网络，同时删除 xml 配置

net-create：创建虚拟网络

net-destroy：停止虚拟网络

dominfo 虚拟机名称：查看指定虚拟机的信息

domblkinfo 虚拟机名称：查看虚拟机块设备信息

domfsinfo 虚拟机名称：查看虚拟机硬件

domiflist 虚拟机名称：查看虚拟机网卡信息

start 虚拟机名称：开启指定的虚拟机

reboot 虚拟机名称：重启指定的虚拟机

shutdown 虚拟机名称：关闭指定的虚拟机

destroy 虚拟机名称：强制关闭指定的虚拟机

autostart [--disable] 虚拟机名称：将指定的虚拟机开启或取消开机自动运行

dumpxml 虚拟机名：导出 xml 配置文件

edit 虚拟机名：对虚拟机的配置进行编辑

define XML 文件：根据 xml 文件定义虚拟机

undefine 虚拟机名：取消定义虚拟机

-c URI：远程连接虚拟机

示例：

```
[root@fzr ~]# virsh net-define --file /etc/libvirt/qemu/networks/vbr.xml
[root@fzr ~]# virsh -c qemu+ssh://root@176.19.4.58/system
```

环境准备：

1.还原真机环境：

```
[root@fzr ~]# rm -f /etc/libvirt/qemu/networks/autostart/*
[root@fzr ~]# echo "net.ipv4.ip_forward = 1">> /etc/sysctl.conf
```

```
[root@fzr ~]# yum remove firewalld python-firewall firewalld-filesystem
[root@fzr ~]# systemctl disable NetworkManager
[root@fzr ~]# sed -i "7c SELINUX=disabled" /etc/selinux/config
[root@fzr ~]# reboot
```

2.创建新的虚拟网络配置文件

```
[root@fzr ~]# vim /etc/libvirt/qemu/networks/vbr.xml
    <network>                       #定义网络
        <name>vbr</name>            #定义网络名
        <bridge name="vbr"/>        #定义网桥名
        <forward mode="nat"/>       #定义转发模式为NAT
        <ip address="192.168.1.254" netmask="255.255.255.0">    #定义本
    地虚拟网卡地址
            <dhcp>                  #定义地址池
            <range start="192.168.1.100" end="192.168.1.200"/>
            </dhcp>
        </ip>
    </network>
```

3.加载虚拟网络配置文件并设置自启动：

```
[root@fzr ~]# virsh net-define /etc/libvirt/qemu/networks/vbr.xml
[root@fzr ~]# virsh net-start vbr
[root@fzr ~]# virsh net-autostart vbr
```

设置完成后，会自动在 vbr.xml 中生成该网卡的 uuid 和 Mac 地址

4.创建虚拟机硬盘

```
[root@fzr ~]# cd /var/lib/libvirt/images/
[root@fzr images]# qemu-img create -f qcow2 node.qcow2 200G
```

5.安装系统，可以使用 ftp，http 等提供光盘镜像，选择最小安装

6.安装基础软件：

```
[root@localhost ~]# yum -y install net-tools
[root@localhost ~]# yum -y install vim-enhanced
[root@localhost ~]# yum -y install bash-completion
```

7.禁用空路由：

```
[root@localhost ~]# echo nozeroconf="yes" >> /etc/sysconfig/network
[root@localhost ~]# ip route show
```

8.设置可以通过 console 连接虚拟机

```
[root@localhost ~]# vim /etc/sysconfig/grub
    给 GRUB_CMDLINE_LINUX 参数添加：
    console=tty0 console=ttyS0,115200n8
[root@localhost ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

9.修改网卡文件

```
[root@localhost ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth0
    TYPE=Ethernet
    BOOTPROTO=dhcp
    IPV4_FAILURE_FATAL=no
    IPV6INIT=no
```

```
DEVICE=eth0
ONBOOT=yes
```

10. 关机后拷贝虚拟机的配置文件，并删除虚拟机：

```
[root@fzr ~]# cp /etc/libvirt/qemu/centos.xml ~
[root@fzr ~]# virsh undefine centos
```

11. 修改配置文件

```
[root@fzr ~]# vim centos.xml
```

xml 的重要参数：虚拟机名称、内存、CPU 数量、磁盘、网卡

可以使用 sed 命令删除 uuid、address 和 mac address 开头的行

声卡、显卡、USB 等服务器用不到的设备也可以删

```
<domain type='kvm'>
  <name>centos7</name>
  <memory unit='KiB'>2048000</memory>
  <currentMemory unit='KiB'>2048000</currentMemory>
  <vcpu placement='static'>2</vcpu>
  <os>
    <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
  </features>
  <cpu mode='custom' match='exact' check='partial'>
    <model fallback='allow'>Skylake-Client</model>
  </cpu>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <pm>
    <suspend-to-mem enabled='no'/>
    <suspend-to-disk enabled='no'/>
  </pm>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/var/lib/libvirt/images/node.qcow2'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <disk type='file' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <target dev='hda' bus='ide'/>
      <readonly/>
```

```
    </disk>
    <controller type='pci' index='0' model='pci-root'/>
    <controller type='ide' index='0'>
    </controller>
    <controller type='virtio-serial' index='0'>
    </controller>
    <interface type='network'>
      <source network='vbr'/>
      <model type='virtio'/>
    </interface>
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <channel type='unix'>
      <target type='virtio' name='org.qemu.guest_agent.0'/>
    </channel>
    <channel type='spicevmc'>
      <target type='virtio' name='com.redhat.spice.0'/>
    </channel>
    <input type='tablet' bus='usb'>
    </input>
    <input type='mouse' bus='ps2'/>
    <input type='keyboard' bus='ps2'/>
    <graphics type='spice' autoport='yes'>
      <listen type='address'/>
      <image compression='off'/>
    </graphics>
    <memballoon model='virtio'>
    </memballoon>
  </devices>
</domain>
```

12. 以原先虚拟机的磁盘为模板创建磁盘文件

```
[root@fzr ~]# cd /var/lib/libvirt/images/
[root@fzr images]# qemu-img create -b node.qcow2 -f qcow2 node1.img
```

PS：创建新的磁盘文件：qemu-img create -f 格式 磁盘路径 磁盘大小

13. 将模板配置文件拷贝到虚拟机目录下，修改

```
[root@fzr ~]# cp centos.xml /etc/libvirt/qemu/node1.xml
[root@fzr qemu]# vim node1.xml
```

设置虚拟机名：<name>node1</name>

设置磁盘文件：<source file='/var/lib/libvirt/images/node1.img'/>

14. 导入虚拟机、启动并测试

```
[root@fzr qemu]# virsh define node1.xml
[root@fzr qemu]# virsh start node1
[root@fzr qemu]# virsh console node1
```

15.创建自动创建脚本

将模板 xml 的第 2 行和第 21 行稍作修改保存为 test.xml：

```
<name>test</name>
<source file='/var/lib/libvirt/images/test.img'/>
```

```
[root@fzr ~]# vim clone.sh
#!/bin/bash
read -p "需要创建的台数" a
for i in $(seq ${a})
do
cat test.xml > node.xml
sed -i "s/test/node${i}/" node.xml
qemu-img create -b /var/lib/libvirt/images/node.qcow2 -f qcow2
/var/lib/libvirt/images/node${i}.img
virsh define node.xml
virsh start node${i}
done
```

guestmount 工具：离线挂载 raw、qcow2 格式虚拟机磁盘，可以在虚拟机关机的情况下，直接修改磁盘中的文档

格式：guestmount -a 虚拟机磁盘路径 -i 挂载点

注意：如果修改后端盘，则所有前端盘都需要重建

软件包：libguestfs-tools

```
[root@fzr ~]# guestmount -a /var/lib/libvirt/images/node.qcow2 -i /mnt
```

云计算：

介绍：

基于互联网的相关服务的增加、使用和交付模式

提供可用的、便捷的、按需的网络访问，进入可配置的计算资源共享池（网络,服务器,存储,应用软件,服务）

资源能够被快速提供，只需投入很少的管理工作，或与服务供应商进行很少的交互

通过互联网来提供动态、易扩展且通常是虚拟化的资源

分类：

IaaS(Infrastructure as a Service)：基础设施即服务

提供给消费者的服务是对所有计算基础设施的利用,包括处理 CPU、内存、存储、网络和其它基本的计算资源,用户能够部署和运行任意软件,包括操作系统和应用程序

消费者不管理或控制任何云计算基础设施，但能控制操作系统的选择、存储空间、部署的应用

用法:公有云、私有云的和混合云

PaaS(Platform-as-a-Service)：平台即服务

不仅仅是单纯的基础平台,而且包括针对该平台的技术支持服务,甚至针对该

平台而进行的应用系统开发、优化等服务

PaaS 平台就是指云环境中的应用基础设施服务,也可以说是中间件即服务

SaaS(Software-as-a-Service)：软件即服务

一种通过 Internet 提供软件的模式，厂商将应用软件统一部署在自己的服务器上，客户可以根据自己实际需求,通过互联网向厂商定购所需的应用软件服务

用户不用再购买软件，而改用向提供商租用基于 Web 的软件，来管理企业经营活动，且无需对软件进行维护

服务提供商会全权管理和维护软件，软件厂商在向客户提供互联网应用的同时，也提供软件的离线操作和本地数据存储,，让用户随时随地都可以使用其定购的软件和服务

Openstack：

介绍：

由 NASA(美国国家航空航天局)和 Rackspace 合作研发并发起的项目

一套 IaaS 解决方案

开源的云计算管理平台

以 Apache 许可证为授权

主要组件：

Horizon：

用于管理 Openstack 各种服务的、基于 web 的管理接口

通过图形界面实现创建用户、管理网络、启动实例等操作

Keystone：

为其他服务提供认证和授权的集中身份管理服务也提供了集中的目录服务

支持多种身份认证模式,如密码认证、令牌认证、以及 AWS(Amazon Web Server)登陆

为用户和其他服务提供了 SSO 认证服务

Neutron：

一种软件定义网络服务

用于创建网络、子网、路由器、管理浮动 IP 地址

可以实现虚拟交换机、虚拟路由器

可用于在项目中创建 VPN

Cinder：

为虚拟机管理存储卷的服务

为运行在 Nova 中的实例提供永久的块存储

可以通过快照进行数据备份

经常应用在实例存储环境中,如果数据库文件

Nova：

管理主机部署在虚拟机的计算节点，用于管理虚拟机资源

Nova 是一个分布式的服务,能够与 Keystone 交互实现认证,与 Glance 交互实现镜像管理

Nova 被设计成在标准硬件上能够进行水平扩展

启动实例时,如果有需要则下载镜像

Glance：

扮演虚拟机镜像注册的角色

允许用户直接存储拷贝服务器镜像

这些镜像可以用于新建虚拟机的模板

部署 Openstack：

1. 部署安装环境：yum 源、bind 域名解析、NTP 时间服务

```
[root@fzr ~]# vim /etc/named.conf
    options {
    listen-on port 53 { 192.168.1.254; };
    ...
    allow-query     { any; };
    }
    forwarders { 176.19.0.26; };
    dnssec-enable no;
    dnssec-validation no;
[root@fzr ~]# systemctl restart named
[root@fzr ~]# vim /etc/chrony.conf
    allow 192.168.0.0/16
    local stratum 10
[root@fzr ~]# systemctl restart chronyd
[root@fzr ~]# chronyc sources -v                #查看时间同步是否成功
[root@fzr ~]# cp /etc/libvirt/qemu/networks/vbr{,1}.xml
[root@fzr ~]# vim /etc/libvirt/qemu/networks/vbr1.xml
[root@fzr ~]# virsh net-define /etc/libvirt/qemu/networks/vbr1.xml
[root@fzr ~]# virsh net-start vbr1
[root@fzr ~]# virsh net-autostart vbr1
```

yum 源需要开启 gpg 验证：

```
    [root@nova ~]# wget ftp://192.168.1.254/centos7/RPM-GPG-KEY-CentOS-7
    [root@nova ~]# rpm --import RPM-GPG-KEY-CentOS-7
```

或者直接在 repo 文件中指定：

```
    [root@openstack ~]# vim /etc/yum.repos.d/centos.repo
        ...
        gpgcheck=1
        gpgkey=ftp://192.168.1.254/centos7/RPM-GPG-KEY-CentOS-7
```

2. 配置两张网卡，eth0 为公共网络，eth1 为隧道接口

3. 关闭 NetworkManager 服务，禁用 SELINUX，卸载 firewalld

```
[root@openstack ~]# rpm -qa | grep -P "firewall|network"
```

puppet-firewall：packstack 依赖包

dracut-network：系统默认安装包

libvirt-daemon-driver-network：libvirt 依赖包

4. 如果需要扩分区需要安装 cloud-utils-growpart：

示例：growpart /dev/sda 1   #重置 sda 第一个分区的空间

注意：该命令需要在英文环境下运行

5. 在/etc/hosts 添加域名解析

```
192.168.1.10 openstack
192.168.1.11 nova
```

6. 配置 Openstack 卷组：

Openstack 为虚拟机提供的云硬盘,本质上是本地的逻辑卷

创建名为 cinder-volumes 的卷组，名称不能改

```
[root@openstack ~]# yum -y install lvm2
[root@openstack ~]# vgcreate cinder-volumes /dev/vdb
```

7. 安装额外软件包：

yum 并不能解决所有依赖包，所以需要额外安装

```
[root@openstack ~]# yum -y install qemu-kvm libvirt-client libvirt-daemon libvirt-daemon-driver-qemu python-setuptools
```

8. 安装 packstack：

```
[root@openstack ~]# yum -y install openstack-packstack
```

9. 配置应答文件：该机不能有 ssh 私钥

--gen-answer-file 自动创建应答文件模板，包含组件的配置

```
[root@openstack ~]# packstack --gen-answer-file answer.ini
[root@openstack ~]# vim answer.txt
11:CONFIG_DEFAULT_PASSWORD=123456          #默认密码
42:CONFIG_SWIFT_INSTALL=n                  #外接存储，y 表示有，n 表示无
75:CONFIG_NTP_SERVERS=192.168.1.254        #时间服务器
98:CONFIG_COMPUTE_HOSTS=192.168.1.10,192.168.1.11    #指定计算节点主机
102:CONFIG_NETWORK_HOSTS=192.168.1.10,192.168.1.11#指定网络节点
554:CONFIG_CINDER_VOLUMES_CREATE=n         #创建卷组，y 表示一键部署时创建，n 表示不创建，需要手工创建
561:CONFIG_CINDER_VOLUMES_SIZE=20G         #卷组大小，需小于硬盘空间
840:CONFIG_NEUTRON_ML2_TYPE_DRIVERS=flat,vxlan       #为 2 块网卡分别配置网络模型
876:CONFIG_NEUTRON_ML2_VXLAN_GROUP=239.1.1.5         #用于通讯的组播地址
910:CONFIG_NEUTRON_OVS_BRIDGE_MAPPINGS=physnet1:br-ex    #将物理网桥映射为逻辑设备
921:CONFIG_NEUTRON_OVS_BRIDGE_IFACES=br-ex:eth0 #将逻辑设备绑定在 eth0 上
936:CONFIG_NEUTRON_OVS_TUNNEL_IF=eth1    #隧道网络
1179:CONFIG_PROVISION_DEMO=n               #demo 测试，y 表示开启，n 表示关闭
```

10. 一键部署 Openstack：

如果 controller 提示 memory 错误，检查 openstack 内存至少为 8G，nova 内存至少为 5G

如果 controller 提示/usr/sbin/ntpdate 错误，检查时间同步

如果 network 提示 openvswitch 错误，检查虚拟机 CPU 是否支持 SSSE3 指令集

根据 computer 错误提示，在 nova 上补齐软件包 libvirt-client

--answer-file 指定应答文件

```
[root@openstack ~]# packstack --answer-file answer.ini
```

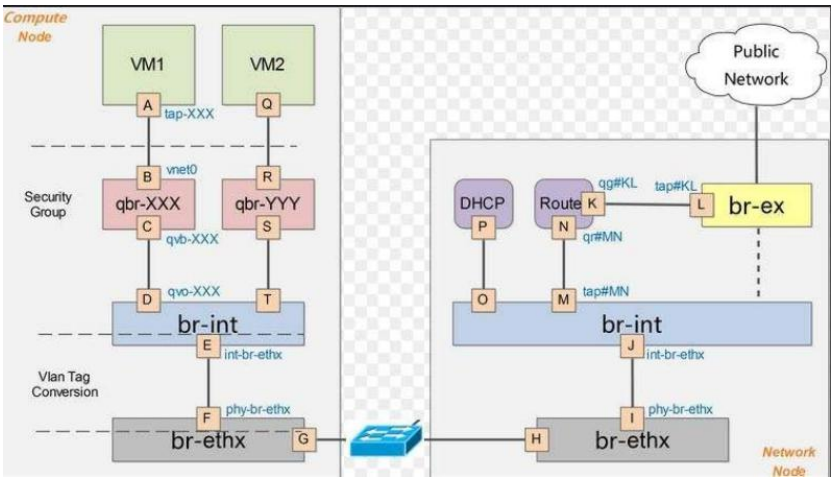
11. 网络拓扑

```
[root@nova ~]# ifconfig
```

br-ex:与实体交换机进行转发的设备

vxlan_sys_4789:网络系统通讯设备，负责封装数据，也称为 br-int

br-ethx：物理网卡



12.配置外部 OVS(Open VSwitch)网桥，通常 Openstack 会自动部署

配置 br-ex 为外部 OVS 网桥：

```
[root@openstack ~]# cat /etc/sysconfig/network-scripts/ifcfg-br-ex
    ONBOOT=yes
    NM_CONTROLLED="no"
    IPADDR=192.168.1.10
    PREFIX=24
    GATEWAY=192.168.1.254
    DEFROUTE=yes
    DEVICE=br-ex
    NAME=br-ex
    DEVICETYPE=ovs
    OVSBOOTPROTO="static"
    TYPE=OVSBridge
```

配置 eth0 为外部 OVS 网桥的端口：

```
[root@openstack ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth0
    DEVICE=eth0
    NAME=eth0
    DEVICETYPE=ovs
    TYPE=OVSPort
    OVS_BRIDGE=br-ex
    ONBOOT=yes
    BOOTPROTO=none
[root@openstack ~]# ovs-vsctl show
```

Openstack 服务的 Web 控制：

Horizon 是一个用以管理、控制 OpenStack 服务的 Web 控制面板，也称称为 Dashboard 仪表盘

基于 python 的 django web 框架开发

可以管理实例、镜像、创建密匙对,对实例添加卷、操作 Swift 容器等

在控制面板中使用终端(console)或 VNC 直接访问实例

所有的数据都放在数据库中

如果误删 Horizon 的 admin 用户，可以通过一系列复杂的操作恢复

功能与特点：

实例管理:创建、终止实例，查看终端日志,VNC 连接,添加卷等

访问不安全管理:创建安全群组，管理密匙对，设置浮劢 IP 等

偏好设定:对虚拟硬件模板可以进行不同偏好设定

镜像管理:编辑或删除镜像

用户管理:创建用户等

卷管理:创建卷和快照

对象存储处理:创建、删除容器和对象

项目:一组隔离的资源和对象。由一组关联的用户进行管理

项目中可以有多个用户，项目中的用户可以在该项目创建、管理虚拟资源

具有 admin 角色的用户可以创建项目

项目相关信息保存到数据库，默认为 MariaDB 中

每个版本都有 bug，本版本 bug 是由 Apache 配置文件导致的：

```
[root@openstack ~]# vim /etc/httpd/conf.d/15-horizon_vhost.conf
    在<VirtualHost>标签中，加入：WSGIApplicationGroup %{GLOBAL}
[root@openstack ~]# apachectl graceful  #重新载入配置文件
```

查看登录用户名和密码，IP：

```
[root@openstack ~]# cat keystonerc_admin
    export OS_USERNAME=admin
    export OS_PASSWORD=bf3ca311f434453b
    export OS_AUTH_URL=http://192.168.2.10:5000/v2.0
```

命令行进入接口：不建议使用

```
[root@openstack ~]# source keystonerc_admin
```

新建项目和用户：身份管理-->项目或用户

云主机类型：资源模板

概念：

定义了一台云主机可以使用的资源，如内存、磁盘容量、CPU 核心数等

Openstack 提供了几个默认的云主机类型

管理员可以自定义云主机类型

参数：

名称：云主机类型名称

ID：云主机类型 ID，可以自定义，选择 atuo 会自动生成一个 UUID

VCPUs：虚拟 CPU 数目

内存(MB)：

根磁盘(GB)：外围磁盘大小。如果希望使用本地磁盘,设置为 0

临时磁盘(GB)：第二个外围磁盘

swap 磁盘(MB)：交换磁盘大小

创建云主机：

1.使用管理员登录

2.选择管理员-->云主机类型-->创建云主机类型

3.输入参数：

名称：nsd1802

ID：auto

VCPU 数量：2
内存 (MB)：512
根磁盘(GB)：10
临时磁盘(GB)：0
Swap 磁盘(MB)：0

镜像：

概念：

虚拟磁盘文件，磁盘文件中应该已经安装了可启动的操作系统
镜像管理功能由 Glance 服务提供
镜像可以由用户上传,也可以通过红帽官方站点下载

Glance 磁盘格式：

raw:非结构化磁盘镜像格式
vhd:VMware、Xen、Microsoft、VirtualBox 等均支持的通用磁盘格式
vmdk:另一个通用的磁盘格式
vdi:VirtualBox 虚拟机和 QEMU 支持磁盘格式
iso:光盘数据内容的归档格式
qcow2:QEMU 支持的磁盘格式。空间自动扩展,并支持写时复制

镜像服务：

提供了服务器镜像的拷贝、快照功能，可以作为模板快速建立、启动服务器
上传镜像时,容器格式必须指定
容器格式指示磁盘文件格式是否包含了虚拟机元数据
镜像服务维护了镜像的一致性

镜像容器格式：混合云的基础

bare:镜像中没有容器或元数据封装
ovf:一种开源的文件规范,描述了一个开源、安全、有效、可拓展的便携式虚拟打包以及软件分布格式
ova:OVA 归档文件
aki:亚马逊内核镜像
ami:亚马逊主机镜像

通过命令行另存镜像为本地文件：openstack image save --file 保存路径 镜像名

创建镜像：

1.使用管理员登录
2.选择管理员-->镜像-->创建镜像
3.输入参数：
镜像名称：nsd1802
文件：通过浏览导入
镜像格式：QCOW2

网络：

工作原理：

实例被分配到子网中,以实现网络连通性
每个项目可以有一到多个子网
OpenStack 网络服务是缺省的网络选项,Nova 网络服务作为备用
管理员能够配置丰富的网络,将其他 Openstack 服务连接到这些网络的接口上
每个项目都能拥有多个私有网络,各个项目的私有网络互相不受干扰

类型：

项目网络：项目拥有的网络由 Neutron 提供。网络间采用 VLAN 隔离

外部网络：访问虚拟机实例的流量,通过外部网络进入。实例需要配置浮动 IP 地址

提供商网络：将实例连接到现有网络,实现虚拟机实例与外部系统共享同一个二层网络

创建网络：

1.使用管理员登录

2.选择管理员-->网络-->创建网络

3.输入参数：

名称：public

项目：nsd1802

供应商网络类型：flat

物理网络：physnet1

管理状态：UP

开启共享和外部网络

4.使用创建的项目用户登录

5.在 public 网络下增加子网：

子网名称：wan

网络地址：192.168.1.0/24

网关 IP：192.168.1.254

6.创建网络：

网络名称：lan

管理状态：UP

子网名称：lan

网络地址：192.168.100.0/24

网关 IP ：192.168.100.254

分配地址池：192.168.100.50,192.168.100.100

DNS 服务器：192.168.1.254

7.新建路由：

路由名称：r1

管理状态：UP

外部网络：public

8.在路由内新增接口：

子网：选择 lan

IP 地址：192.168.100.254

9.创建云主机：

项目-->云主机数量-->创建云主机

云主机名称：host1

源：

选择源：镜像

创建新卷：否

选择已有的镜像，点击+

云主机类型：

选择已有的类型，点击+

网络：

若有外网 IP，则选择 public

通常选择内网 IP，即 lan

其他按需选择

10.点击云主机-->控制台：控制云主机

安全组：

作用：

用于控制对虚拟机实例的访问

在高层定义了授权

每个项目都可以定义自己的安全组

项目成员可以编辑默认的安全规则,也可以添加新的安全规则

所有的项目都有一个默认的 default 安全组

规则：定义了如何处理网络访问

创建规则

项目-->计算-->访问和安全-->添加规则

创建安全组：名称：policy

添加规则：配置规则、方向、打开端口、端口

在云主机数量的动作中编辑安全组

浮动 IP 地址：

作用：

用于从外界访问虚拟机

只能从现有浮动 IP 地址池中分配

虚拟机实例起动后,可以关联一个浮动 IP 地址

虚拟机实例也可以解除 IP 地址绑定

创建浮动 IP：

项目-->计算-->云主机数量-->绑定浮动 IP

安装计算节点：

1.在配置文件的 CONFIG_COMPUTE_HOST 和 CONFIG_NETWORK_HOSTS 添加主机

2.在管理节点上重新执行安装命令

3.使用管理员登录 web，在管理员-->主机聚合下查看

注意：不会覆盖已安装的服务，只有被改动的部分选项需要重新配置

云主机热迁移：

云主机可以在不同的节点间热迁移，需要 qemu-kvm-rhev 包，有些版本包名叫 qemu-kvm-ev

如果迁移失败，先检查两个节点间能否互相使用主机名 ping 通

查看错误日志排错：/var/log/nava/nova-compute.log

Docker 容器

概述：完整的一套容器管理系统，提供了一组命令，让用户更加方便直接使用，不需要过多关心底层内核技术

容器技术已经成为应用程序封装和交付的核心技术

3 大技术核心：由内核提供

CGroups(Control Groups)：限制、隔离进程组所使用的资源管理，文件位置：/sys/fs/cgroup。

可以约束资源使用：内存、文件系统缓存、CPU 利用率、磁盘分组，审计，挂载进程

NameSpace：修改进程视图

UTS(-uts)：UTS 命名空间用于设置主机名和对该命名空间中正在运行的进程可见的域。

IPC(-ipc)：IPC 命名空间提供命名的共享内存段，信号量和消息队列的分离。

rootfs：根文件系统

SELinux：安全

由于是在物理机上实施隔离，启动一个容器就像启动一个进程一样快速

优点：

使用共享的公共库和程序

对比虚拟化技术更加简洁高效

缺点：

隔离性差，所有容器内的进程都运行在物理机

由于共用 Linux 内核，安全性有缺陷

SELinux 难以驾驭

监控容器和容器排错是难点

系统准备：64 位操作系统、RHEL7 以上的版本、关闭防火墙

关闭防火墙的原因：docker 会在 iptables 内写入规则，防火墙会使规则混乱，导致部分 docker 无法使用

安装 Docker：

软件包：docker-engine 和 docker-engine-selinux

```
[root@fzr ~]# pssh -ih hosts "yum -y install
docker-engine-1.12.1-1.el7.centos.x86_64.rpm
docker-engine-selinux-1.12.1-1.el7.centos.noarch.rpm"
[root@fzr ~]# pssh -ih hosts "systemctl start docker.service"
[root@fzr ~]# pssh -ih hosts "systemctl enable docker.service
[root@fzr ~]# pssh -ih hosts ifconfig
```

Docker 镜像：

容器是基于镜像启动的

镜像是启动容器的核心

镜像采用分层设计

使用快照的 COW 技术,确保底层数据不丢失

Docker 镜像常用命令：

```
docker images   #查看镜像列表：仓库名称、标签、ID、创建时间、大小
docker history  #查看镜像制作时的历史命令
docker inspect  #查看镜像底层信息：环境变量、存储卷、标签
docker pull     #下载镜像
```

格式：docker pull [-a] NAME[:TAG]

参数-a 表示所有镜像

```
docker push     #上传镜像，需要先自定义仓库
```

格式：docker push NAME[:TAG]

```
docker rmi      #删除本地镜像，需要先删除容器
docker save     #镜像备份，另存为 tar 包
```

格式：docker save 名称:标签 > 名称.tar

docker load #镜像还原，用 tar 包导入

格式：docker load 名称 < 名称.tar

docker search #搜索镜像

格式：docker search 镜像名

docker tag #给镜像建立软链接

Docker 容器常用命令：

docker run #运行容器

格式：docker run [选项] 镜像[:标签] [命令]

默认命令为 bash，进入容器的解释器，如果命令不是解释器，则在容器内执行该命令后退出容器

-i 交互

-t 终端

-d 放入后台

-p 物理机端口:容器端口 将容器的端口和物理机端口绑定

-m 设定启动分配的内存

-h 定义容器的主机名

-v 文件映射

--name 给容器命名

docker ps #查看容器列表，需要另外开启终端

格式：docker ps [选项]

-a 查看所有

-q 仅显示容器 id

-l 显示标签

-s 显示总空间

docker stop #关闭容器

docker start #启动容器

docker restart #重启容器

docker attach #进入容器，退出时容器会关闭

进入容器后，接管 systemd 进程

docker exec #进入容器，退出时不会关闭容器

相当于使用终端连入容器

docker inspect #查看容器底层信息

docker top #查看容器进程列表

docker rm #删除容器

标签：latest 默认标签，在不指定标签时，默认使用该标签

注意：

1.容器在退出后，容易就关闭了，如果需要重新进入，则需要重新启动容器

2.由于有些启动容器的默认命令是非交互式，所以不加启动命令-it 的话，容器会在启动后自动关闭

3.启动命令-it 适合交互式容器，-d 适合非交互式容器，不确定时可以使用-itd

4.id 是每个容器的唯一标识，ps 查看时，只显示 id 的前 12 位

5.操作容器时，都需要指定容器 id，可以只输入 id 的开头一部分，只要没有重复即可

示例：

```
[root@host1 ~]# docker load < ubuntu.tar
[root@host1 ~]# docker save nginx > n.tar
[root@host1 ~]# docker images
[root@host1 ~]# docker search rhel7          #需要联网
[root@host1 ~]# docker pull rhel7            #需要联网
[root@host1 ~]# docker run -it centos
[root@host1 ~]# docker run -it redis bash    #正常启动
[root@host1 ~]# docker run redis bash        #启动完成后自动退出
[root@host1 ~]# docker inspect centos
    "State": {
    "Pid": 5827,
    },
    "Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Cmd": [
    "/bin/sh",
    "-c",
    "#(nop) ",
    "CMD [\"/bin/bash\"]"        #启动后进去 bash 解释器
    ],
[root@host1 ~]# docker inspect nginx
    "Cmd": [
    "nginx",
    "-g",
    "daemon off;"                #启动后非交互，所以启动后就挂起了
    ],
[root@host1 ~]# docker tag centos test
[root@host1 ~]# docker tag centos haha:aa
[root@host1 ~]# docker rmi test
[root@host1 ~]# docker rmi haha:aa  #删除必须带标签
[root@host1 ~]# docker run -itd centos
[root@host1 ~]# docker ps
    CONTAINER ID        IMAGE       COMMAND     CREATED         STATUS ...
    3146f39e45c7        centos      "/bin/bash"     13 seconds ago      Up
9 seconds...
[root@host1 ~]# docker ps -a        #查看所有容器 id
[root@host1 ~]# docker start e59d030450b7
[root@host1 ~]# docker rm $(docker ps -aq)
[root@host1 ~]# docker run -d nginx #服务类容器在后台启动即可
    94cf6886ae538d28f0e9dc8982ca363daff615b9efb12f07dc3001826646e61a
[root@host1 ~]# docker stop 94cf6886
[root@host1 ~]# docker ps -a
    CONTAINER ID        IMAGE       COMMAND             CREATED     STATUS
```

```
94cf6886ae53        nginx       "nginx -g 'daemon off"  2 minutes ago
Exited (0) 36 seconds ago
3146f39e45c7        centos      "/bin/bash"             8 minutes ago
Exited (137) 4 seconds ago
[root@host1 ~]# docker run -p 80:80 -d nginx
f6f767799c25db6e1af50da4d5af05596c3b81244220fd4f32eedfd6968e8c02
[root@host1 ~]# docker exec -it f6f76 bash
```

自定义镜像：

创建镜像：docker commit 容器 ID 镜像名:标签

1.启动容器并进入：

```
[root@host1 ~]# docker run -itd centos
    0fcf0188f86fba57b522ea8bfe1216251184e19f982a9766f77df091165eba57
[root@host1 ~]# docker exec -it 0fcf bash
```

2.配置 yum 并安装软件

```
[root@0fcf0188f86f /]# rm -f /etc/yum.repos.d/CentOS-*
[root@0fcf0188f86f /]# vi /etc/yum.repos.d/centos.repo
[root@0fcf0188f86f /]# yum -y install bash-completion vim-enhanced
```

3.配置完成后，停止容器，创建新的镜像

```
[root@host1 ~]# docker stop 0fc
[root@host1 ~]# docker commit 0fc new:test
```

4.测试新镜像

```
[root@host1 ~]# docker run -it new:test bash
```

使用 Dockerfile 自动创建镜像

语法格式：

FROM:基础镜像

MAINTAINER:镜像创建者信息

EXPOSE:开放的端口号

ENV:设置环境变量

ADD:添加文件到镜像

RUN:制作镜像时执行的命令,可以有多个

WORKDIR:定义容器默认工作目录

CMD:容器启动时执行的命令,只能有一条

启动 Dockerfile：docker build -t 镜像名称:标签 配置文件目录

注意：

由于正常系统的上帝进程为 system，而容器的上帝进程为 bash

所以，/usr/sbin/sshd 和/usr/sbin/httpd 可以在正常系统中启动程序，但不能在容器中启动

/usr/sbin/sshd -D 表示将进程放在前台，也就是 bash 下，所以容器中可以用这种方式启动进程

/usr/sbin/httpd -DFOREGROUND 用于启动 httpd，nginx 需要在配置文件中写入 daemon off;

环境变量可不设置，设置后可以防止某些报错

遵循原则：数据与程序分离；层次不要太多

示例 1：

```
[root@host1 ~]# mkdir new
[root@host1 ~]# cd new
[root@host1 new]# vim Dockerfile
    from centos
    run rm -f /etc/yum.repos.d/CentOS-*
    add centos.repo /etc/yum.repos.d/
    run yum -y install bash-completion vim-enhanced net-tools
    cmd ["bash"]
[root@host1 new]# cp /etc/yum.repos.d/centos.repo .
[root@host1 new]# docker build -t test .
```

示例 2：

```
[root@host1 new]# vim Dockerfile
    from test
    run yum -y install openssh-server
    run echo 123456 | passwd --stdin root
    env EnvironmentFile=/etc/sysconfig/sshd
    run /usr/sbin/sshd-keygen
    expose 22
    cmd ["/usr/sbin/sshd","-D"]
[root@host1 new]# docker build -t test: .ssh
[root@host1 ~]# docker run -p 80:80 -d test:ssh
```

示例 3：

```
[root@host1 new]# vim Dockerfile
    from test:ssh
    run yum -y install httpd
    env EnvironmentFile=/etc/sysconfig/httpd
    workdir /var/www/html
    run echo hello > index.html
    expose 80 22
    add start.sh /etc/init.d/run.sh
    cmd ["/etc/init.d/run.sh"]
[root@host1 new]# vim start.sh
    #!/bin/bash
    /usr/sbin/sshd -D&
    /usr/sbin/httpd -DFOREGROUND/srv/gitlab
[root@host1 mkssh]# docker build -t test:all
[root@host1 ~]# docker run -p 80:80 -p 1122:22 -d test:all
```

自定义镜像仓库：

registry 私有仓库：共享镜像的一台服务器

1.在 1.10 上创建配置文件，使用 http 连接仓库：默认使用 HTTPS 连接仓库，需要证书

```
[root@host1 ~]# vim /etc/docker/daemon.json
    {"insecure-registries":["192.168.1.10:5000"]}
[root@host1 ~]# systemctl restart docker
```

2.在 1.10 启动仓库镜像：

```
[root@host1 ~]# docker run -p 5000:5000 -d registry
```

3.给镜像打上标签后上传

```
[root@host1 ~]# docker tag test 192.168.1.10:5000/haha
[root@host1 ~]# docker tag new:test 192.168.1.10:5000/abc:aaa
[root@host1 ~]# docker push 192.168.1.10:5000/haha
[root@host1 ~]# docker push 192.168.1.10:5000/abc:aaa
```

4.在 1.11 测试镜像仓库

```
[root@host2 ~]# vim /etc/docker/daemon.json
    {"insecure-registries":["192.168.1.10:5000"]}
[root@host2 ~]# systemctl restart docker
[root@host2 ~]# docker run -it 192.168.1.10:5000/haha
```

5.查看仓库

镜像：`[root@host2 ~]# curl 192.168.1.10:5000/v2/_catalog`

```
    {"repositories":["abc","haha"]}
```

标签：`[root@host2 ~]# curl 192.168.1.10:5000/v2/haha/tags/list`

指定镜像仓库：以阿里云为例

```
[root@host1 ~]# vim /etc/docker/daemon.json
{
  "registry-mirrors": ["https://522gf3xr.mirror.aliyuncs.com"]
}
```

持久化存储：

存储卷：

docker 容器不保持任何数据

重要数据使用外部卷存储

容器可以挂载真实机文件(目录和文本)或共享存储

如果容器文件地址和真实文件地址相同，可不写容器地址

主机卷映射：-v 真实机文件:容器文件

```
[root@host1 ~]# mkdir /web
[root@host1 ~]# echo hello > /web/index.html
[root@host1 ~]# docker run -p 80:80 -v /web:/var/www/html -d test:httpd
```

共享存储：真实机挂载后，-v 挂载目录:容器目录

```
[root@host2 ~]# mkdir /nfsfile
[root@host2 ~]# echo 1.11> /nfsfile/index.html
[root@host2 ~]# vim /etc/exports
    /nfsfile ro(192.168.1.10)
[root@host2 ~]# systemctl restart nfs-server
[root@host1 ~]# showmount -e 192.168.1.11
    /nfsfile 192.168.1.10
[root@host1 ~]# mkdir /nfs
[root@host1 ~]# vim /etc/fstab
     192.168.1.11:/nfsfile /nfs nfs _netdev 0 0
[root@host1 ~]# mount -a
[root@host1 ~]# docker run -p 80:80 -v /nfs:/var/www/html -d test:httpd
```

Docker 网络架构：

查看 Docker 创建的网络模型：

```
[root@host2 ~]# docker network list
    NETWORK ID          NAME                DRIVER              SCOPE
    5525a282f506        bridge              bridge              local
    e4cf64e525aa        host                host                local
    0a6bddcab80a        none                null                local
```

查看网桥详细信息：

```
[root@host1 ~]# docker network inspect bridge
[root@host1 ~]# docker network inspect 5525
```

使用 Docker 创建网桥：docker network create --help

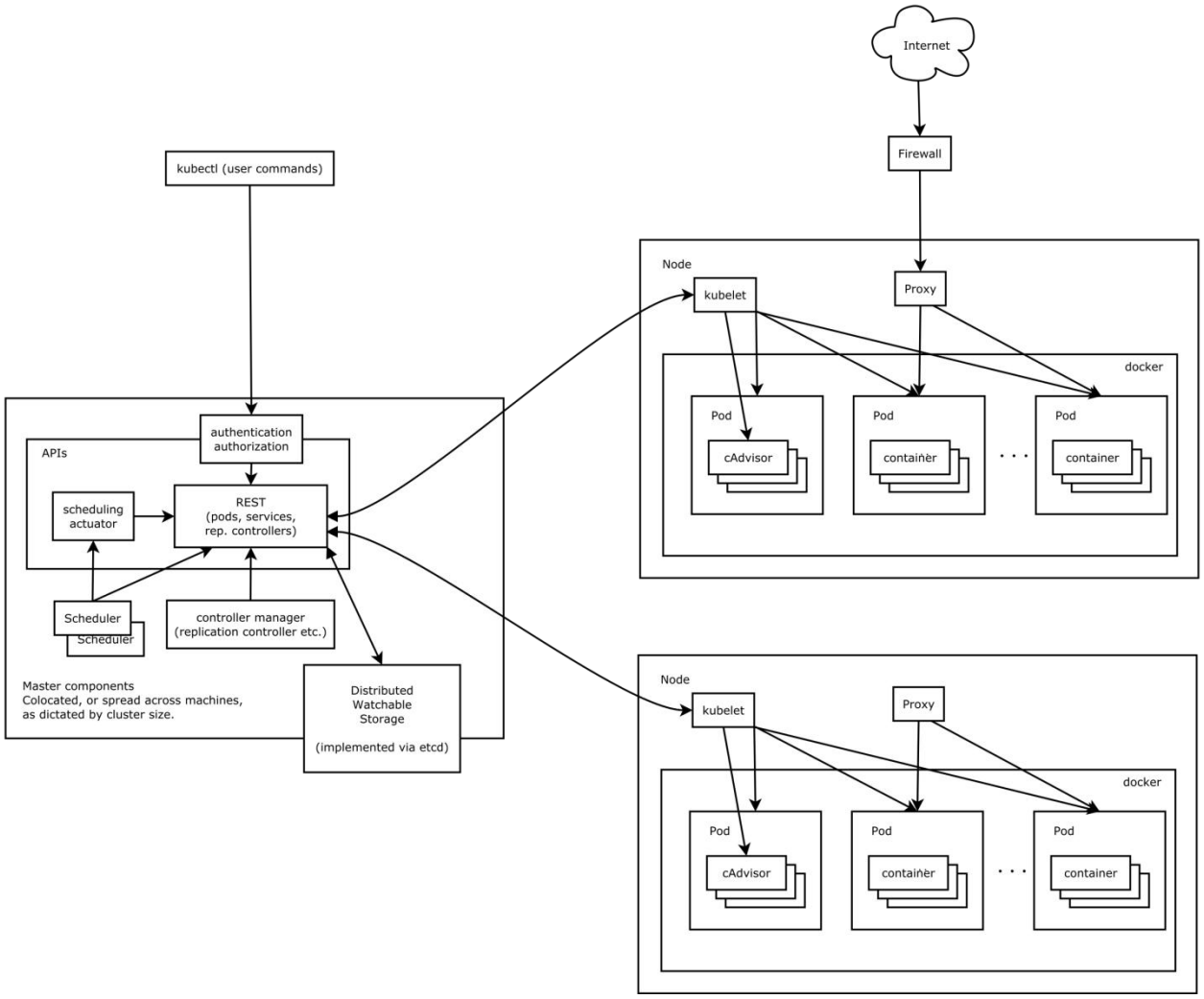
```
[root@host1 ~]# docker network create --subnet 192.168.100.0/24 -d bridge
-o com.docker.network.bridge.name=docker1 docker1
```

使用自定义网桥：

```
[root@host1 ~]# docker run -it --network=docker1 new:test
```

k8s：Kubernetes

架构图：



核心组件：

etcd：保存了整个集群的状态

apiserver：提供了资源操作的唯一入口，并提供认证、授权、访问控制、API 注册和发现等机制

controller manager：负责维护集群的状态，比如故障检测、自动扩展、滚动更新等

scheduler：负责资源的调度，按照预定的调度策略将 Pod 调度到相应的机器上

kubelet：负责维护容器的生命周期，同时也负责 Volume（CVI）和网络（CNI）的管理

Container runtime：负责镜像管理以及 Pod 和容器的真正运行（CRI）

kube-proxy：负责为 Service 提供 cluster 内部的服务发现和负载均衡

POD：k8s 调度的最小单位

RC(Replication Controller)：POD 应答控件

RS(Replica Set)：RC 集合

Deployment：启动 RS

SVC(Service)：核心的资源对象

Kubeadmin：集成启动命令

流程:外部通过 UI 或者命令行链接 API,由 API 提交给 master,再由 master 将需求转发给 node 执行

分层架构：

核心层：Kubernetes 最核心的功能，对外提供 API 构建高层的应用，对内提供插件式应用执行环境

应用层：部署（无状态应用、有状态应用、批处理任务、集群应用等）和路由（服务发现、DNS 解析等）

管理层：系统度量（如基础设施、容器和网络的度量），自动化（如自动扩展、动态 Provision 等）以及策略管理（RBAC、Quota、PSP、NetworkPolicy 等）

接口层：kubectl 命令行工具、客户端 SDK 以及集群联邦

生态系统：在接口层之上的庞大容器集群管理调度的生态系统

Kubernetes 外部：日志、监控、配置管理、CI、CD、Workflow、FaaS、OTS 应用、ChatOps 等

Kubernetes 内部：CRI、CNI、CVI、镜像仓库、Cloud Provider、集群自身的配置和管理等

ETCD：

开源的分布式键值对存储工具。在每个 coreos 节点上面运行的 etcd，共同组建了 coreos 集群的共享数据总线。

保证 coreos 集群的稳定，可靠。

当集群网络出现动荡，或者当前 master 节点出现异常时，etcd 可以优雅的进行 master 节点的选举工作，同时恢复集群中损失的数据。

分布在各个 coreos 节点中的 app，都可以自由的访问到 etcd 中的数据。

功能：

简单可靠，API 丰富(支持 http，jason)

支持客户端通过 SSL 认证，保证安全性

每个实例可以支持每秒 1000 次写操作

基于 RAFT 协议完成分布式操作

通过 http 轮询，监听网络变化

FLEET：

管理 coreos 和部署 app 的工具。

Fleet 可以把整个 coreos 集群当做一台节点来处理。将应用都封装成轻量级的服务，这些服务很容易在集群中进行管理和部署。

功能：

在当前 coreos 集群中随机部署 docker container

在集群中跨主机进行服务分发

负责维护集群中的服务实例，当服务实例异常时，重新进行任务调度来恢复服务

发现集群中的各个节点

自动 SSH 到其它节点来执行 job

最低配置：2CPU+2G 内存

环境准备：

1. 关闭虚拟内存：swapoff -a
2. 查看/etc/fstab 是否有虚拟内存的配置
3. 在内核中配置网桥，要求 iptables 对 bridge 的数据进行处理

   net.bridge.bridge-nf-call-iptables：二层的网桥在转发包时也会被 iptables 的 FORWARD 规则过滤

   net.bridge.bridge-nf-call-ip6tables：同上，IPv6 版本

   vm.swappiness：配置虚拟内存

   ```
   [root@client ~]# vim /etc/sysctl.d/k8s.conf
       net.bridge.bridge-nf-call-ip6tables = 1
       net.bridge.bridge-nf-call-iptables = 1
       net.ipv4.ip_forward = 1
       vm.swappiness=0
   [root@client ~]# sysctl --system
   ```

4. 配置 docker 和 k8s 的 yum 源

   ```
   [root@client ~]# cat > /etc/yum.repos.d/kubernetes.repo << EOF
   [kubernetes]
   name=Kubernetes
   baseurl=http://mirrors.aliyun.com/kubernetes/yum/repos/kubernetes-el7-x86_64/
   enabled=1
   gpgcheck=1
   gpgkey=http://mirrors.aliyun.com/kubernetes/yum/doc/yum-key.gpg

   http://mirrors.aliyun.com/kubernetes/yum/doc/rpm-package-key.gpg
   EOF
   [root@client ~]# wget http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
   [root@client ~]# mv docker-ce.repo /etc/yum.repos.d/
   ```

5. 安装 docker，kubeadm，kubelet，kubectl。当前版本为 1.12.2

   ```
   [root@client ~]# yum install -y docker-ce kubelet kubeadm kubectl
   ```

kubernetes-cni

```
[root@client ~]# systemctl start docker
[root@client ~]# systemctl start kubelet
```

6. kubernetes 集群不允许开启 swap，所以我们需要忽略这个错误

```
[root@client ~]# vim /etc/sysconfig/kubelet
    KUBELET_EXTRA_ARGS="--fail-swap-on=false"
```

7. 使用 yaml 自动配置 kubernetes

apiVersion 可通过查看：

https://github.com/kubernetes/kubernetes/tree/release-1.12/cmd/kubeadm/app/apis/kubeadm

```
[root@client ~]# vim kubeadm.yaml
    apiVersion: kubeadm.k8s.io/v1alpha2
    kind: MasterConfiguration
    controllerManagerExtraArgs:
      horizontal-pod-autoscaler-use-rest-clients: "true"
      horizontal-pod-autoscaler-sync-period: "10s"
      node-monitor-grace-period: "10s"
    apiServerExtraArgs:
      runtime-config: "api/all=true"
    kubernetesVersion: "v1.12.2"
```

8. 根据错误提示确定镜像和版本

```
[root@client ~]# kubeadm init --config kubeadm.yaml
[preflight] Some fatal errors occurred:
    [ERROR ImagePull]: failed to pull image k8s.gcr.io/kube-apiserver:v1.12.2: ...
    [ERROR ImagePull]: failed to pull image k8s.gcr.io/kube-controller-manager:v1.12.2: ...
    [ERROR ImagePull]: failed to pull image k8s.gcr.io/kube-scheduler:v1.12.2: ...
    [ERROR ImagePull]: failed to pull image k8s.gcr.io/kube-proxy:v1.12.2: ...
    [ERROR ImagePull]: failed to pull image k8s.gcr.io/etcd:3.2.24: ...
    [ERROR ImagePull]: failed to pull image k8s.gcr.io/coredns:1.2.2: ...
    [ERROR ImagePull]: failed to pull image k8s.gcr.io/pause:3.1: ...
```

9. kubernetes 的镜像托管在 google 云上，无法直接下载。从 https://github.com/anjia0532/gcr.io_mirror 拉取镜像

```
[root@client ~]# vim pull.sh
    #!/bin/bash
    images=(kube-proxy:v1.12.2 kube-scheduler:v1.12.2
    kube-controller-manager:v1.12.2 kube-apiserver:v1.12.2 etcd:3.2.24
    coredns:1.2.2 pause:3.1 )
    for imageName in ${images[@]}
```

```
do
docker pull anjia0532/google-containers.$imageName
docker tag anjia0532/google-containers.$imageName
k8s.gcr.io/$imageName
docker rmi anjia0532/google-containers.$imageName
done
```

10. 重新执行 yaml，完成 Kubernetes Master 的部署，这个过程需要几分钟

```
[root@client ~]# kubeadm init --config kubeadm.yaml
Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a
regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
...
kubeadm join 172.16.186.96:6443 --token 8qxmb3.yvrj42ib39a6h1at
--discovery-token-ca-cert-hash
sha256:fe33ff9b62cf7cb6adac85f6bb9247d4bc4e7078c3f1d306 5afa78d65313
90f5
```

11. 配置 kubectl 与 apiserver 的认证

```
[root@client ~]#  mkdir -p $HOME/.kube
[root@client ~]#  sudo cp -i /etc/kubernetes/admin.conf
$HOME/.kube/config
[root@client ~]#  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

12. 检查健康状态

```
[root@client ~]# kubectl get cs
```

13. 查看节点状态

```
[root@client ~]# kubectl get nodes
```

14. 部署网络插件 Weave 并验证

```
[root@client ~]# kubectl apply -f https://git.io/weave-kube-1.6
[root@client ~]# kubectl get pods -n kube-system
NAME                              READY   STATUS    RESTARTS
AGE
coredns-576cbf47c7-kwwk8          1/1     Running   0
42m
coredns-576cbf47c7-p62rv          1/1     Running   0
42m
etcd-client                       1/1     Running   0
41m
kube-apiserver-client             1/1     Running   0
41m
kube-controller-manager-client    1/1     Running   0
```

```
41m
    kube-proxy-pvsg7                  1/1     Running   0
42m
    kube-scheduler-client             1/1     Running   0
41m
    weave-net-jwqsk                   2/2     Running   0
78s
```

15. 使用 master 调度 pod。注：由于是单机测试，所以没有 client

```
[root@client ~]# kubectl taint nodes --all
node-role.kubernetes.io/master-
    node/client untainted
```

16. 安装可视化插件镜像

```
[root@client ~]# wget
https://raw.githubusercontent.com/kubernetes/dashboard/master/src/deploy/recommended/kubernetes-dashboard.yaml
[root@client ~]# vim kubernetes-dashboard.yaml
    ...
    kind: Deployment
    apiVersion: apps/v1beta2
    ...
    spec:
      ...
      template:
        ...
        spec:
          containers:
            image: k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.0
    ...
    #修改最后几行，可以直接使用 token 认证进入
    kind: Service
    apiVersion: v1
    metadata:
      labels:
        k8s-app: kubernetes-dashboard
      name: kubernetes-dashboard
      namespace: kube-system
    spec:
      type: NodePort          #添加 Service 的 type 为 NodePort
      ports:
        - port: 443
          targetPort: 8443
          nodePort: 30001    #添加映射到 docker 的端口,k8s 只支持
30000-32767 之间的端口
      selector:
```

```
            k8s-app: kubernetes-dashboard
    [root@client ~]# docker pull
anjia0532/google-containers.kubernetes-dashboard-amd64:v1.10.0
    [root@client ~]# docker tag
anjia0532/google-containers.kubernetes-dashboard-amd64:v1.10.0
k8s.gcr.io/kubernetes-dashboard-amd64:v1.10.0
    [root@client ~]# docker rmi
anjia0532/google-containers.kubernetes-dashboard-amd64:v1.10.0
    [root@client ~]# kubectl apply -f kubernetes-dashboard.yaml
```

17. 验证 Dashboard 是否正常启动

```
    [root@client ~]# kubectl get pods -n kube-system
        ...
        weave-net-jwqsk                          2/2     Running   0
    64m
        ...
```

18. 安装容器存储插件镜像

```
    [root@client ~]# kubectl apply -f
https://raw.githubusercontent.com/rook/rook/master/cluster/examples/kub
ernetes/ceph/operator.yaml
    [root@client ~]# kubectl apply -f
https://raw.githubusercontent.com/rook/rook/master/cluster/examples/kub
ernetes/ceph/cluster.yaml
```

19. 查看容器存储插件安装情况。

    注：由于通过 deployment 启动 pod，拉镜像需要时间，在第 18 步完成后，需要一段时间后才能查看到结果

```
    [root@client ~]# kubectl get pods -n rook-ceph-system
    [root@client ~]# kubectl get pods -n rook-ceph
```

20. 开启服务

```
    [root@client ~]# nohup  kubectl proxy --address='0.0.0.0'
--accept-hosts='^*$'  --disable-filter=true &
```

21. 获取 token 命令

```
    [root@client ~]# kubectl -n kube-system describe $(kubectl -n
kube-system get secret -n kube-system -o name | grep namespace) | grep token
```

22. 网页访问 dashboard

```
    [root@client ~]# curl 127.0.0.1:30001
```

查看命令：

查看全部节点：kubectl get pods --all-namespaces

查看 pods：kubectl describe pod -n kube-system

查看具体问题：kubectl describe pod kubernetes-dashboard-77fd78f978-m5stn -n kube-system 或 kubectl describe pod kubernetes-dashboard -n kube-system

云计算：按使用量付费的模式，提供可用的、便捷的、按需的网络访问，网络、服务器、存储、应用软件、服务等资源能够被快速提供

组件：

SLB 负载均衡：流量入口
ECS 应用服务器：云主机
RDS 数据库
OSS 对象存储

ansible

一款 IT 自动化和 DevOps 软件,已被 Redhat 收购。

糅合了很多老运维工具的优点，实现了批量操作系统配置、批量程序的部署、批量运行命令等功能

自动化部署 APP、自动化管理配置项、自动化的持续交付、自动化的(AWS)云服务管理

模块化设计,调用特定的模块来完成特定任务

基于 pythone 语言实现 paramiko、PyYAML、jinja2

模块支持 JSON 等标准输出格式,可采用任何编程语言重写

优点：

仅需 ssh 和 Python 即可使用
无客户端
功能强大，模块丰富，上手容易
基于 python 开发，容易做二次开发
使用公司多,社区活跃

特性：

部署简单
主从模式工作
支持自定义模块
支持 playbook
易于使用
支持多层部署
支持异构 IT 环境

管理主机：

要求 Python2.6 以上
paramiko
PyYAML
Jinja2
httplib2
six

被托管主机：

Ansible 默认通过 SSH 协议管理机器
被管理主机要开启 ssh 服务,允许 ansible 主机登陆
在托管节点上也需要安装 Python2.5 或以上的版本
如果托管节点上开启了 SElinux,需要安装 libselinux-python

1.安装 ansible：

先搭好 yum 源：网络上的扩展源 extras

也可以直接下载

http://mirror.centos.org/centos/7.5.1804/extras/x86_64/Packages/ansible-2.4.2.0-2.el7.noarch.rpm

[root@ansible~]#yum -y install ansible

PS:支持源码安装：

https://github.com/ansible/ansible/tree/stable-2.6

python setup.py build

python setup.py install

2.配置文件：

优先级顺序：

首先检测 ANSIBLE_CONFIG 变量定义的配置文件

其次检查当前目录下的 ansible.cfg 文件

再次检查当前用户家目录下~/ansible.cfg 文件

最后检查/etc/ansible/ansible.cfg 文件

[root@ansible ~]# vim /etc/ansible/ansible.cfg

14:inventory = /etc/ansible/hosts #定义托管主机地址配置文件

3.配置分组文件：域名解析可以在管理主机使用/etc/hosts，也可以搭建 dns

格式：

[组名称]主机名

主机名称、ip 地址、登录用户名、密码、端口等信息

参数说明：

ansible_ssh_host：要连接的进程主机名与设定的主机的别名不同的情况下,设置主机名

ansible_ssh_port：ssh 端口号，如果不是默认的端口号，设置端口号

ansible_ssh_user：ssh 用户名

ansible_ssh_pass：ssh 密码(这种方式并不安全，强烈建议使用--ask-pass 或 SSH 密钥)

ansible_sudo_pass：sudo 密码(建议使用--ask-sudo-pass)

ansible_sudo_exe(new in version 1.8)：sudo 命令路径(适用于 1.8 及以上版本)

ansible_shell_type：目标系统的 shell 类型；默认情况下,命令的执行使用'sh'语法,可设置为'csh'或'fish'.

ansible_python_interpreter：目标主机的 python 路径；适用于系统中有多个 python,或者命令路径不是"/usr/bin/python"

[root@ansible ~]# vim /etc/ansible/hosts

```
[web]
    web1
    web2
[db]
    db[1:2]
[db:vars]                          #db 组的全组参数
    ansible_ssh_user="root"
    ansible_ssh_pass="123456"
[other]
    cache ansible_ssh_user="root" ansible_ssh_pass="123456"
[app:children]                     #引用组，下面跟组名
    web
    db
```

4.测试连通性：

```
[root@ansible ~]# ansible cache -m ping
[root@ansible ~]# ansible app -m ping
[root@ansible ~]# ansible all -m ping
```

5.自定义配置文件和分组文件：

```
[root@ansible ~]# mkdir test
[root@ansible ~]# cd test
[root@ansible test]# vim ansible.cfg
    [defaults]
    inventory=host123
[root@ansible test]# vim host123
    [test1]
        192.168.1.2
        db1
    [test2]
        192.168.1.3
    [test3]
        db2
    [test4]
        cache
    [test:children]
        test2
        test3
```

6.动态主机：

通过外部脚本获取主机列表，并按照 ansible 所要求的格式返回给 ansilbe 命令

JSON(JavaScript Object Notation)：一种基于文本，独立于语言的轻量级数据交换格式

注意：

主机部分必须是列表格式的

主机列表键值行，"hosts"可以省略，但如果使用，必须是"hosts"

```
[root@ansible ~]# vim test.py
    #!/usr/bin/python
    import json
    hostlist = {}
    hostlist["web"] = ["192.168.1.2", "web2"]
    hostlist["db"] = {"hosts" : ["192.168.1.4","db2"]}
    hostlist["cache"] = {
    "hosts" : ["192.168.1.6"],
    "vars" : {
    "ansible_ssh_user":"root","ansible_ssh_pass":"123456"}}
    print(json.dumps(hostlist))
```

ansible 命令；

格式：ansible 主机或分组 参数

参数：

-M 指定模块路径
-m 使用模块,默认 command 模块
-a 模块的参数
-i 指定主配置文件路径,或可执行脚本
-k 使用交互式登录密码
-e 定义变量
-v 详细信息
-vvvv 开启 debug 模式

列出所有主机：ansible all --list-hosts

批量检测主机：ansible all -m ping

批量执行命令：

非交互式：ansible all -m command -a "命令"

交互式：ansible all -m command -a "命令" -k

批量部署公钥：先生成密钥

exclusive：如果原密钥已经存在，true 表示覆盖

manage_dir：如果.ssh 目录不存在，true 表示创建

$(</root/.ssh/authorized_keys)：读取该文件中的内容

ansible all -m authorized_key -a "user=root exclusive=true manage_dir=true key='$(</root/.ssh/id_rsa.pub)'" -k

ansible 模块：

官方模块地址：

https://docs.ansible.com/ansible/latest/user_guide/modules.html

模块执行流程：

1. 将模块文件读入内存，然后添加传递给模块的参数，最后将模块中所需要的类添加到内存，由 zipfile 压缩后，再由 base64 进行编码，写入到模板文件内

2. 通过默认的连接方式(一般是 ssh)，ansible 连接到远程主机，创建临时目录，并关闭连接

3. 打开另外一个 ssh 连接,将模板文件以 sftp 方式传送到刚刚创建的临时目录中,写完后关闭连接

4. 打开一个 ssh 连接将任务对象赋予可执行权限，执行成功后关闭连接

5. 最后，ansible 将再打开一个新连接来执行模块，并删除临时目录及其所有内容

6. 模块的结果是从标准输出 stdout 中获取 json 格式的字符串。ansible 将解析和处理此字符串

ansible-doc：模块的手册

ansible-doc -l：列出所有模块

ansible-doc 命令：查看该命令帮助

统一格式：ansible 主机或组 -m 模块名 -a "命令或参数"

ping 模块：测试 ssh 连通性，没有参数

格式：ansible 主机或组 -m ping

command 模块：默认模块，进程执行命令

注意：不支持重定向，管道，set，&等，原因是该模块在 ssh 执行时不启动 shell

shell 模块：用法基本和 command 一样

通过/bin/sh 执行命令

raw 模块：用法和 shell 模块一样

注意：比 shell 模块少了 chdir（指定目录）、creates（创建）、removes（删除）参数

script 模块：批量执行本地脚本，可以使 shell,python,php...

参数：脚本路径

copy 模块：复制文件到进程主机

命令选项：

src：要复制的文件的本地地址，如果路径是一个目录，则递归复制

dest 远程主机的绝对路径，当源文件是目录时，该路径也必须是目录

backup：覆盖复制时是否备份源文件，yes 表示备份，no 表示不备份

force：复制时是否覆盖，yes 表示覆盖，no 表示不覆盖

lineinfile 模块：行编辑替换模块

命令选项：

path=目的文件

regexp=正则表达式，指定行

line=替换后的结果

replace 模块：指定字符编辑替换模块

命令选项：

path=目的文件

regexp=正则表达式，扩展正则的保留

replace=\n 新字符

yum 模块：远程使用 yum 来管理软件包

config_file：yum 的配置文件

disable_gpg_check：关闭签名检查

disablerepo：不使用某个源

enablerepo：使用某个源

name：要进行操作的软件包的名字,也可以传递一个 url 或者一个本地的 rpm 包的路径

state：选项：installed 和 removed，默认为安装

service 模块：远程管理服务

name：服务名称

enabled：开机自启，yes 表示开机自启，no 表示开机不自启

sleep：在执行 restarted 时，在 stop 和 start 之间沉睡几秒钟

state：对当前服务执行 started,stopped,restarted,reloaded 等操作

setup 模块：获取主机信息

filter：过滤指定关键字

示例：

```
[root@ansible ~]# ansible-doc -l
[root@ansible ~]# ansible-doc ping
[root@ansible ~]# ansible web -m command -a "useradd zhang3"
[root@ansible ~]# ansible web -m shell -a "echo 123456 | passwd --stdin zhang3"
[root@ansible ~]# ansible web -m raw -a "chage -d 0 zhang3"
[root@ansible test]# ansible test1 -m shell -a "id zhang3 || useradd li4"
```

```
[root@ansible test]# vim user.sh
    #!/bin/bash
    id zhang3
    if [ $? != 0 ]
    then
    useradd li4
    echo 123456 | passwd --stdin li4
    chage -d 0 li4
    fi
[root@ansible test]# ansible test3 -m script -a "./user.sh"
[root@ansible ~]# ansible all -m copy -a "src=~/a dest=/tmp/b"
[root@ansible ~]# ansible web1 -m lineinfile -a 'path=/etc/passwd
regexp="^zhang3" line="zhang3:x:1111:1111::/home/zhang3:/bin/bash"'
[root@ansible ~]# ansible web1 -m replace -a 'path=/etc/hosts
regexp="^(127.0.0.1 ).*" replace="\1localhost"'
[root@ansible ~]# ansible web1 -m yum -a "name='httpd'"
[root@ansible ~]# ansible web1 -m service -a "name='httpd' enabled='yes'
state='started'"
[root@ansible ~]# ansible web1 -m setup -a 'filter=ansible_distribution'
```

ansible 七大工具：

ansible 命令：用于执行临时性的工作

ansible-doc：文档说明，针对每个模块都有详细的用法说明及应用案例介绍

ansible-console：为用户提供的一款交互式工具，用户可以在 ansible-console 虚拟出来的终端上像 Shell 一样使用 Ansible 内置的各种命令

ansible-galaxy：从 github 上下载管理 Roles 的一款工具

ansible-playbook：日常应用中使用频率最高的命令，其工作机制是通过读取预先编写好的 playbook 文件实现批量管理。按一定条件组成的 ansible 任务集

ansible-vault：主要用于配置文件加密，ansible-vault 可加密/解密这个配置文件

ansible-pull：通常在配置大批量机器的场景下会使用，让客户端来管理机上传下载，灵活性稍有欠缺，但效率几乎可以无限提升

开启管道连接，可以优化 ansible：

管道连接：与远程主机只有一个连接，命令通过数据流的方式发送执行

```
[root@ansible ~]# vim /etc/ansible/ansible.cfg
383:pipelining = True
```

json(JavaScript Object Notation)：JavaScript 对象表示法

基于文本，独立于语言的轻量级数据交换格式

分隔符：单引号、小括号、中括号、大括号、冒号和逗号

特性：纯文本、可读性、层级结构、通过 JavaScript 进行解析

语法规则：数据在键值对中、由逗号分隔，大括号保存对象，中括号保存数组，键值用冒号隔开

简单结构：

```
{ "讲师":
["牛犇", "丁丁", "静静","李欣"]
```

```
}
```

复合复杂结构：

```
{ "讲师":
[ {"牛犇":"小逗逼", "负责阶段":"1"},
{"丁丁":"老逗逼", "负责阶段":"2"},
{"静静":"漂亮姐", "负责阶段":"3"},
{"李欣":"老司机", "负责阶段":"4"}
]
}
```

YAML(YAML Ain't Markup Language)：可读性高,用来表达数据序列的格式

基础语法：

YAML 的结构通过空格来展示

数组："- "

键值对："： "

使用一个固定的缩进风格表示数据层级结构关系,一般每个缩进级别由两个以上空格组成

同一层级缩进必须对齐

表示注释

不能使用 tab

通过修改 vim 配置，方便书写 yaml 文件：

```
[root@ansible ~]# vim .vimrc
    autocmd filetype yaml setlocal sw=2 ts=2 et ai
```

键值表达式：

采用冒号分隔，冒号后面必须有一个空格

结构 1："庞丽静": "漂亮姐"

结构 2：

```
"庞丽静":
  "漂亮姐"
```

结构 3：

```
"讲师":
  "庞丽静": "漂亮姐"
```

结构 4：

```
"讲师":
  "庞丽静":
    "漂亮姐"
```

数组表达：

使用一个短横杠加一个空格

结构 1：

```
- "牛犇"
- "丁丁"
- "静静"
- "李欣"
```

结构 2：哈希数组复合

```
"讲师":
  - "牛犇"
  - "丁丁"
  - "静静"
  - "李欣"
```

结构 3：

```
"讲师":
  -
    "牛犇": "小逗比"
    "阶段": 1
  -
    "丁丁": "老逗比"
    "阶段": 2
  -
    "静静": "漂亮姐"
    "阶段":3
  -
   "李欣": "老司机"
   "阶段": 4
```

高级语法：

>：表示对应的值为多行字符，把\n 转换为空格

|：表示对应的值为多行字符

!：设置类型

!!：强制类型转换

*：锚点参考

<<：散列合并

&：锚点标记

jinja2：基于 python 的模板引擎,包含变量和表达式

模版基本语法：

表达式："{{...}}"：调用变量，计算，判断

控制语句："{%...%}"：

```
{% if...%}
{% elif...%}
  {% for...%}
    {{do...}}
  {% endfor %}
{% endif %}
```

注释："{#...#}",支持块注释

过滤器：前一个过滤器的输出会被作为后一个过滤器的输入

playbook：是 ansible 用于配置、部署和管理托管主机的剧本

playbook 语法格式：

playbook 由 YAML 语言编写,遵循 YAML 标准

在同一行中,#之后的内容为注释

同一个列表中的元素应该保持相同的缩进

playbook 由一个或多个 play 组成

play 中 hosts,variables,roles,tasks 等对象的表示方法都是以":"分隔键值

所有的 YAML 文件首行都是---，这是 YAML 格式的一部分,表明一个文件的开始

构成：

Target：定义将要执行 playbook 的远程主机组

Variable：定义 playbook 运行时需要使用的变量

Tasks：定义将要在远程主机上执行的任务列表

Handler：定义 task 执行完成以后需要调用的任务

执行结果：

使用 ansible-playbook 运行 playbook 文件,得到输出内容为 JSON 格式。并且由不同颜色组成,便于识别

绿色代表执行成功

***代表系统代表系统状态发生改变

红色代表执行失败

示例：

-f 并发进程数量，默认是 5，数量不建议超过 cpu 线程数的 2 倍

hosts：一个或多个组或主机的名称，以逗号分隔

remote_user：账户名

tasks：任务列表，只有在前一个任务在其所对应的所有主机上执行完后才执行下一个

任务就是一个个模块，可以通过 ansible-doc 模块名，查找例子

```
[root@ansible ~]# vim first.yml
    ---
    - hosts: all
      remote_user: root
      tasks:
        - ping:
[root@ansible ~]# ansible-playbook first.yml -f 5
[root@ansible ~]# vim httpd.yml
    ---
    - hosts: web2
      remote_user: root
      tasks:
        - name: install Apache
          yum:
            name: httpd
        - lineinfile:
            path: /etc/httpd/conf/httpd.conf
            regexp: "^Listen "
            line: 'Listen 8080'
        - lineinfile:
            path: /etc/httpd/conf/httpd.conf
```

```
            regexp: "#ServerName"
            line: 'servername localhost'
        - service:
            name: httpd
            state: started
            enabled: yes
        - shell: echo "hello world" > /var/www/html/index.html
```

PS：lineinfile 的 regexp 匹配不到时，会在最后一行插入 line 的内容，原因在于 backrefs 默认值是 no

playbook 高级语法：

变量：通过 vars 定义

user 模块的 password 使用变量过滤器 password_hash 写入

错误处理方法：

1.在命令后加：|| true

2.在命令下增加一行：ignore_errors: True

示例：

```
[root@ansible ~]# vim user.yml
---
- hosts: db1
  remote_user: root
  vars:
    username: wang5
  tasks:
    - user:
        name: "{{username}}"
        password: "{{'123456'|password_hash('sha512')}}"
      ignore_errors: True
    - shell: chage -d 0 "{{username}}"
```

handlers：当关注的资源发生变化时采取一定的操作

notify：在所有的剧本都结束后，调用 handlers

notify 调用的是 handler 的 name 定义的字符串，必须一致，否则不会被触发

多个 task 触发同一个 notify 的时候，同一个服务只会触发一次

notify 可以触发多个条件,handler 非常适合在生产环境中修改某一个配置文件后要重启若干服务的场景

示例：

```
[root@ansible ~]# vim httpd.yml
---
- hosts: web2
  remote_user: root
  tasks:
    - name: install Apache
      yum:
        name: httpd
    - lineinfile:
```

```
      path: /etc/httpd/conf/httpd.conf
      regexp: "^Listen "
      line: 'Listen 80'
      backrefs: yes
    notify:
      - reload httpd
  - shell: echo "hello world" > /var/www/html/index.html
 handlers:
  - name: reload httpd
    service:
      name: httpd
      state: started
      enabled: yes
```

when：判断

register：保存前一个命令的返回，不加参数表示返回状态，后面可以调用，ignore_errors 会使 register 的状态永远为真

```
[root@ansible ~]# vim monit.yml
---
- hosts: web
  remote_user: root
  tasks:
    - shell: uptime | awk '{printf("%.2f\n",$(NF-2))}'
      register: result
    - service: name=httpd state=stopped
      when: result.stdout|float > 0.7
```

with_items：标准循环，一个列表或字典，通过{{ item }}迭代获取每个值

with_nested：嵌套循环

tags：给指定的任务定义一个调用标识，ansible-playbook 调用时可以指定标签调用

-t 标签名

--tags=标签名

--skip-tags=标签名

--start-at-task=标签名

include：引用其他文件中的 play、task 或 handler

roles：引入一个项目的文件和目录

vars：变量层

tasks：任务层

handlers：触发条件

files：文件

template：模板

default：默认

debug：调试

在运行时输出更为详细的信息，帮助排错

检测语法：--syntax-check

测试运行：-C

显示收到影响到主机：--list-hosts

显示工作的任务：--list-tasks

显示将要运行的标签：--list-tags

示例 1：循环

```
[root@ansible ~]# vim item.yml
---
- hosts: cache
  remote_user: root
  vars:
    pwd: "123456"
  tasks:
    - user:
        name: "{{item.name}}"
        group: "{{item.group}}"
      with_items:
        - {name: "nb", group: "root"}
        - {name: "dd", group: "root"}
        - {name: "jj", group: "wheel"}
        - {name: "lx", group: "wheel"}
```

示例 2：嵌套循环

```
[root@ansible ~]# vim nest.yml
---
- hosts: cache
  remote_user: root
  vars:
    un: [a, b, c]
    id: [1, 2, 3]
  tasks:
    - name: add users
      shell: echo {{item}}
      with_nested:
        - "{{un}}"
        - "{{id}}"
```

示例 3：调试

```
[root@ansible ~]# vim debug.yml
---
- hosts: web
  remote_user: root
  tasks:
    - shell: uptime |awk '{printf("%.2f\n",$(NF-2))}' >&2
      register: result
    - service:
        name: httpd
        state: stopped
```

```
    when: result.stdout|float >0.7
  - name: Show debug info
    debug: var=result
```

ELK：

介绍：一整套解决方案，3 款软件首字母缩写

Elasticsearch:负责日志检索和储存

Logstash:负责日志的收集和分析、处理

Kibana:负责日志的可视化

功能：

分布式日志数据集中式查询和管理

系统监控,包含系统硬件和应用各个组件的监控

故障排查

安全信息和事件管理

报表功能

下载地址：https://www.elastic.co/downloads/

Elasticsearch：

介绍：基于 Lucene 的搜索服务器，提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful API 的 web 接口

特点：

实时分析

分布式实时文件存储，并将每一个字段都编入索引

文档导向，所有的对象全部是文档

高可用性，易扩展，支持集群(Cluster)、分片(Shards)和复制(Replicas)

接口友好，支持 JSON

概念：

Node：装有一个 ES 服务器的节点

Cluster：有多个 Node 组成的集群

Document：一个可被搜索的基础信息单元

Index：拥有相似特征的文档的集合

Type：一个索引中可以定义一种或多种类型

Filed：是 ES 的最小单位

Shards：索引的分片，每一个分片就是一个 Shard

Replicas：索引的拷贝

ES 与数据库对比：

数据库		ES	
数据库	database	index	索引
表	table	type	类型
行	rows	document	文档
列	column	field	字段
分库分表	schema	mapping	映射
关键字	index	每个都是关键字	
搜索	select	get http://	

更新	update	put http://

ES 集群安装：

1.环境准备：

/etc/hosts 设置域名解析和 hostname 设置主机名，要求集群之间能通过主机名 ping 通

elasticsearch5.0 以上版本默认需要剩余内存 1G 以上，可以修改 /etc/elasticsearch/jvm.options

2.安装 JDK 和 es 包：推荐使用 OpenJDK 1.8

```
[root@es01 ~]# yum -y install java-1.8.0-openjdk
[root@es01 ~]# yum -y install elasticsearch-2.3.4.rpm
```

3.修改配置文件：仅用于测试

```
[root@es01 ~]# vim /etc/elasticsearch/elasticsearch.yml
54:network.host: 192.168.1.1
```

4.起服务：

```
[root@es01 ~]# systemctl start elasticsearch
[root@es01 ~]# systemctl enable elasticsearch
```

5.测试：默认会启动 9200 和 9300 两个端口

```
[root@es01 ~]# ss -antup | grep java
[root@es01 ~]# curl http://192.168.1.1:9200/
```

6.配置集群：在其他主机上重复步骤 1，2

```
[root@es01 ~]# vim install.yml
---
- hosts: all
  remote_user: root
  tasks:
    - copy:
        src: ~/elasticsearch-2.3.4.rpm
        dest: ~/elasticsearch-2.3.4.rpm
    - copy:
        src: /etc/hosts
        dest: /etc/hosts
    - yum:
        name: java-1.8.0-openjdk
[root@fzr ~]# for i in {2..5};do
    pssh -iH "192.168.1.$i" "echo rs0$i > /etc/hostname"
    pssh -iH "192.168.1.$i" "yum -y install elasticsearch-2.3.4.rpm"
    pssh -iH "192.168.1.$i" "systemctl start elasticsearch"
    pssh -iH "192.168.1.$i" "systemctl enable elasticsearch"
    done
```

7.修改配置文件

```
[root@es01 ~]# vim /etc/elasticsearch/elasticsearch.yml
17:cluster.name: nsd1802              #集群名称
23:node.name: es01                 #当前节点标识
```

54:network.host: 192.168.1.1 #节点 ip

68:discovery.zen.ping.unicast.hosts: ["es01", "es02", "es03"] #申明集群的节点，起服务时必须先启动

```
[root@fzr ~]# for i in {2..5};do
    scp 192.168.1.1:/etc/elasticsearch/elasticsearch.yml 192.168.1.${i}:/etc/elasticsearch/elasticsearch.yml
    pssh -iH "192.168.1.$i" sed -i "23s/es01/es0${i}/" /etc/elasticsearch/elasticsearch.yml
    pssh -iH "192.168.1.$i" sed -i "54s/192.168.1.1/192.168.1.${i}/" /etc/elasticsearch/elasticsearch.yml
    done
[root@fzr ~]# pssh -ih hosts "systemctl restart elasticsearch"
```

8.验证集群：

```
http://192.168.1.1:9200/_cluster/health?pretty
  "cluster_name" : "nsd1802",        #定义的集群名
  "status" : "green",                #集群状态，绿色为正常，黄色表示有问题但不是很严重，红色表示严重故障
  "number_of_nodes" : 5,             #集群中节点的数量
  "number_of_data_nodes" : 5,
```

ES 插件：使用 json 格式

plugin 安装工具：

支持网络安装：http://，ftp://和本地安装：file://，默认从官网装插件

不加参数可查看帮助：/usr/share/elasticsearch/bin/plugin

参数：

```
install 安装
remove  卸载
list        查看已安装
```

示例：

```
[root@es01 ~]# /usr/share/elasticsearch/bin/plugin install ftp://192.168.1.254/elasticsearch-head-master.zip
[root@es01 ~]# /usr/share/elasticsearch/bin/plugin install ftp://192.168.1.254/bigdesk-master.zip
[root@es01 ~]# /usr/share/elasticsearch/bin/plugin install ftp://192.168.1.254/elasticsearch-kopf-master.zip
[root@es01 ~]# /usr/share/elasticsearch/bin/plugin list
```

访问插件：http://192.168.1.1:9200/_plugin/插件名

head 插件：进行索引(Index)和节点(Node)的操作，对集群的查询

索引：当集群有 n 台主机时，最大分片数为 n，最大副本数为 n-1

kopf 插件：对集群操作

bigdesk 插件：对集群监控

Rest API：

_cat：查询集群状态、节点信息

nodes：查询节点状态信息

indices：索引信息

v：显示详细信息
help：显示帮助
PUT　　增
DELETE　删
POST　　改
GET 查
注意:
1.创建索引和增加数据时,都是使用 json 格式
2.类型不需要创建，增加数据时，会自动创建
3.增加数据时，id 必须唯一，否则会覆盖旧数据
示例：

```
[root@fzr ~]# curl 192.168.1.1:9200/_cat/health
[root@fzr ~]# curl 192.168.1.1:9200/_cat/health?v
[root@fzr ~]# curl 192.168.1.1:9200/_cat/health?help
[root@fzr ~]# curl -X "PUT" "192.168.1.1:9200/test" -d "
    {
        "settinds":{
            "index":{
                "number_of_shards":5,
                "number_of_replicas":1
            }}}"
[root@fzr ~]# curl 192.168.1.1:9200/_cat/indices?v
[root@fzr ~]# curl 192.168.1.1:9200/_cat/nodes?v
[root@fzr ~]# curl -X "PUT" "192.168.1.1:9200/test/teacher/1" -d '
    {
    "title":"a",
    "name":{"first":"b","last":"c"},
    "age":30}'
[root@fzr ~]# curl -X "POST" "192.168.1.1:9200/test/teacher/1/_update" -d
'
    {
    "doc":
    {"age":20}}'
[root@fzr ~]# curl -X "GET" "192.168.1.1:9200/test/teacher/1"
[root@fzr ~]# curl -X "DELETE" "192.168.1.1:9200/test/teacher/1"
[root@fzr ~]# curl -X "DELETE" "192.168.1.1:9200/test"
[root@fzr ~]# curl -X "DELETE" "192.168.1.1:9200/*"
```

kibana：数据可视化平台工具
特点:
灵活的分析和可视化平台
实时总结和流数据的图表
为不同的用户显示直观的界面
即时分享和嵌入的仪表板
1.安装：[root@kibana ~]# yum -y install kibana-4.5.2-1.x86_64.rpm

2.修改配置文件：只修改第 15 行，将 elasticsearch 的 ip 填入

```
[root@kibana ~]# vim /opt/kibana/config/kibana.yml
2:server.port: 5601          #端口不允许修改
5:server.host: "0.0.0.0"
15:elasticsearch.url: "http://192.168.1.1:9200"
23:kibana.index: ".kibana"  #kibana 在 elasticsearch 插入库的默认名称
26:kibana.defaultAppId: "discover"       #kibana 的默认主页，未初始化时，会自动跳转 settings 设置页面
53:elasticsearch.pingTimeout: 1500
57:elasticsearch.requestTimeout: 30000
64:elasticsearch.startupTimeout: 5000
```

3.启动服务

```
[root@kibana ~]# systemctl start kibana
[root@kibana ~]# systemctl enable kibana
```

4.测试

```
[root@kibana ~]# ss -antup| grep 5601
http://192.168.1.6:5601/status
```

kibana 数据批量导入：

_bulk：使用 POST 方式，数据格式为 json，编码使用 data-binary

当数据中申明了 index 和 type 时，在导入时不需要指定：指定了也没用

```
[root@kibana ~]# head -1 shakespeare.json
{"index":{"_index":"shakespeare","_type":"act","_id":0}}
```

当数据中没有申明时，必须指定 index 和 type：

```
[root@kibana ~]# head -1 accounts.json
{"index":{"_id":"1"}}
```

导入数据：

```
[root@kibana ~]# curl -X "POST" 192.168.1.1:9200/_bulk --data-binary @shakespeare.json
[root@kibana ~]# curl -X "POST" 192.168.1.1:9200/test/a/_bulk --data-binary @accounts.json
```

kibana 数据查询：

人性化显示：pretty

```
[root@kibana ~]# curl -X GET 192.168.1.1:9200/test/a/1?pretty
```

批量查询：_mget

```
[root@kibana ~]# curl -X GET 192.168.1.1:9200/test/a/_mget?pretty -d '{
    "docs":[
    {"_id":1},
    {"_id":2}]}'
[root@kibana ~]# curl -X GET 192.168.1.1:9200/_mget?pretty -d '{
    "docs":[
    {"_index": "shakespeare",
    "_type:": "scene",
    "_id": 1
    },
```

```
            {"_index": "test",
            "_type:": "a",
            "_id": 1}]}'
```

map 映射：

创建索引的时候，可以预先定义字段的类型及相关属性。

作用：让索引建立得更加的细致和完善。

分类：

动态映射：自动根据数据进行相应的映射

静态映射：自定义字段映射数据类型

kibana 的页面使用：

1.创建索引，初始化：

Index name or pattern 使用默认的 logstash-*

Time-field name 选择@timestramp 作为索引

2.由于测试数据是 2015 年的，所以在右上角设置查看时间。absolute 表示自定义时间

3.visualize 作图，保存后可以在 Dashboard 查看

logstash：数据采集、加工处理以及传输的工具

功能：

所有类型的数据集中处理

不同模式和格式数据的正常化

自定义日志格式的迅速扩展

给自定义数据源添加插件

软件环境：

Logstash 依赖 java 环境

Logstash 没有默认的配置文件,需要手动配置

logstash 低版本安装在/opt/logstash 目录下

logstash5.0 以上版本安装在/usr/share/logstash 目录,且需要 1G 以上空闲内存，可在/etc/logstash/jvm.options 中修改

工作流程：数据源-->input{}-->filter{}-->output{}-->ES

logstash 语法：

类型：

布尔值类型	ssl_enable => true
字节类型	my_bytes => "1MiB" #1024 进制
	my_bytes => "1mb" #1000 进制
字符串类型	name => "jack"
	name => "It\"s a beautiful day" #支持转义字符
数值类型	port => 22
数组	match => ["数组 1","数组 2"]
哈希	match => {"field" => "value"}
编码解码	codec => "json"
路径	path => "文件名"
注释	#

条件判断：

等于	==
不等于	!=

小于 <
大于 >
小于等于 <=
大于等于 >=
匹配正则 =~
不匹配正则 !~
包含 in
不包含 not in
与 and
或 or
非与 nand
非或 xor
复合表达式 ()
取反符合 !()

插件：/opt/logstash/bin/logstash-plugin list

logstash-codec：编码类插件
logstash-filter：处理类插件
logstash-input：采集类插件
logstash-output：输出类插件

编写 logstash 文件

基本格式：

```
[root@logstash ~]# vim logstash.conf
    input{...}
    filter{...}
    output{...}
```

运行：[root@logstash ~]# /opt/logstash/bin/logstash -f logstash.conf

更多查看：https://www.elastic.co/guide/en/logstash/current/index.html

logstash 完美匹配 json 格式，无需处理即可标准输出

示例 1：codec 格式转化

```
input{stdin{codec=>"json"}}
filter{}
output{stdout{codec=>"rubydebug"}}
```

示例 2：file 模块

在读取文件时，logstas 会在当前用户的家目录下自动创建.sincedb 开头的文件，用于记录上次读取的位置

sincedb_path 可以指定存放的位置，start_position 可以指定开始读取的位置

```
input{
    file{
        path=>["/tmp/a.log","/tmp/b.log"]
        type => "test"
        sincedb_path=>"/tmp/since.db"
        start_position=>"beginning"}}
filter{}
output{stdout{codec=>"rubydebug"}}
```

示例 3：开启 tcp 和 udp 监听，启动后可以检查 8888 端口

```
input{
    tcp {
        port=>8888
        mode=>"server"       #默认 server
        type=>"tcptest"}
    udp {
        port=>8888
        type=>"udptest"}}
filter{}
output{stdout{codec=>"rubydebug"}}
```

测试：

```
[root@logstash ~]# vim aa
    function fasong(){
    exec 9<>/dev/$1/192.168.1.7/8888
    echo $2 >&9
    exec 9<&-
    }
[root@logstash ~]# . aa
[root@logstash ~]# fasong tcp test
```

示例 4：远端传输日志文件

```
input{
    syslog {
    port => 514}}
filter{}
output{stdout{codec=>"rubydebug"}}
```

测试：

```
[root@es01 ~]# vim /etc/rsyslog.conf
19:$ModLoad imtcp
20:$InputTCPServerRun 514
+1:*.info @@192.168.1.7:514
[root@es01 ~]# systemctl restart rsyslog.service
[root@es01 ~]# logger -i dhcpd "hello"
```

示例 5：grok 结构化数据

解析各种非结构化的日志数据插件

使用正则表达式把非结构化的数据结构化

分组匹配时，正则表达式需要根据具体数据结构编写

直接写正则：

```
input{
    file{
    path=>["/tmp/a.log"]
    sincedb_path=>"/dev/null"
    start_position=>"beginning"}}
filter{
```

```
        grok {
        match => { "message" => "(?<ip>[0-9.]+).*\[(?<time>.+).\+0800\]
    \"(?<method>[A-Z]+) (?<url>\S+).*\" (?<res>[0-9]+).*; (?<os>.*);"}}}
    output{stdout{codec=>"rubydebug"}}
```

调用正则宏：修改 match 行

正则宏路径：

低版本：

/opt/logstash/vendor/bundle/jruby/1.9/gems/logstash-patterns-core-2.0.5/patterns/grok-patterns

高版本：

/usr/share/logstash/vendor/bundle/jruby/2.3.0/gems/logstash-patterns-core-4.1.2/patterns/grok-patterns

```
match => { "message" => "%{COMMONAPACHELOG}"}
```

示例 6：将 json 格式的文件发送给 ES

```
input{
    file{
    path=>["/tmp/a.log"]
    sincedb_path=>"/var/lib/logstash/since.db"
    start_position=>"beginning"
    type=>"filelog"
    codec=>"json"}}
filter{}
output{
    if [type] == "filelog"{
    elasticsearch {
    hosts => ["192.168.1.1:9200"]
    index => "log"
    flush_size => 2000
    idle_flush_time => 10}}}
```

连通 web、Logstash 和 ES：

1.在 web 服务器上部署 filebeat，收集 http 日志

```
[root@es01 ~]# yum -y install filebeat-1.2.3-x86_64.rpm
[root@es01 ~]# grep -v "#" /etc/filebeat/filebeat.yml | grep -v ^$
    filebeat:
      prospectors:
        -
          paths:
            - /var/log/httpd/access_log
          input_type: log
          document_type: log
      registry_file: /var/lib/filebeat/registry
    output:
      logstash:
        hosts: ["192.168.1.7:5044"]
```

```
    shipper:
    logging:
      files:
[root@es01 ~]# systemctl restart filebeat.service
```

2.配置 Logstash

```
[root@logstash ~]# vim logstash.conf
    input{
        beats {
        port => 5044}}
    filter{
        grok {match => { "message" => "%{COMMONAPACHELOG}"}}}
    output{
        elasticsearch {
        hosts => ["192.168.1.1:9200"]
        index => "apache_log"
        flush_size => 2000
        idle_flush_time => 10}
        }
[root@logstash ~]# /opt/logstash/bin/logstash -f logstash.conf
```

大数据：

无法在一定时间范围内用常规软件工具进行捕捉、管理和处理的数据集合。大数据技术能从中快速获得有价值信息

大数据的 5V 特性：

Volume(大体量)：数百 TB 到 PB、甚至 EB 的规模。

Variety(多样性)：大数据包括各种格式和形态的数据。

Velocity(时效性)：很多大数据需要在一定的时间限度下得到及时处理。

Veracity(准确性)：处理的结果要保证一定的准确性。

Value(大价值)：大数据包含很多深度的价值,大数据分析挖掘和利用将带来巨大的商业价值。

Hadoop：

定义：

是一种分析和处理海量数据的软件平台

是一款开源软件,使用 JAVA 开发

可以提供一个分布式基础架构

特点：高可靠性、高扩展性、高效性、高容错性、低成本

Google 提出了：GFS，MapReduce，BigTable

GFS 是一个可扩展的分布式文件系统

GFS 用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，提供容错功能。

Mapreduce 是针对分布式并行计算的一套编程模型

Mapreduce 由 Map 和 reduce 组成，Map 是影射，把指令分发到多个 worker 上去，reduce 是规约，合并 worker 计算出来的结果

BigTable 存储结构化数据。

BigTable 是建立在 GFS,Scheduler,LockService 和 MapReduce 之上的。每个 Table 都是一个多维的稀疏图

Hadoop 基于 Google 的理论设计的开源 Java 软件

核心组件：

HDFS：分布式文件存储系统

MapReduce：分布式计算框架

Yarn：集群资源管理系统

常用组件：

HDFS

Mapreduce

Zookeeper：分布式协作服务

Hbase：分布式列存数据库

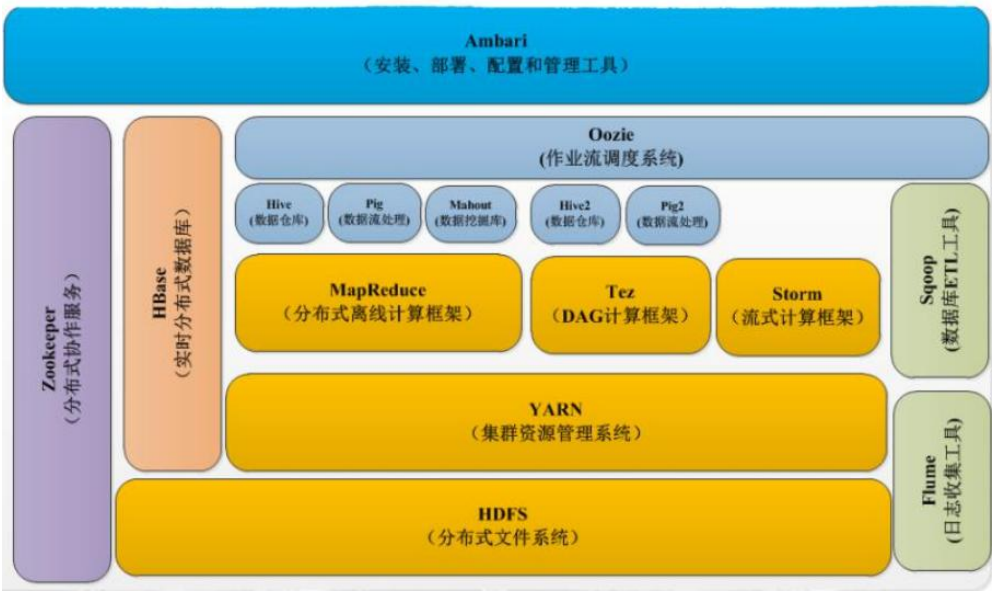
Hive：基于 Hadoop 的数据仓库

Sqoop：数据同步工具

Pig：基于 Hadoop 的数据流系统

Mahout：数据挖掘算法库

Flume：日志收集工具



HDFS：数据存储管理的基础。高度容错系统，用于在低成本的通用硬件上运行

NameNode：

Master 节点，管理 HDFS 的名称空间和数据块映射信息，配置副本策略，处理所有客户端请求。

SecondaryNameNode：

定期合并 fsimage 和 fsedits，推送给 NameNode

紧急情况下，可辅助恢复 NameNode，但并非 NameNode 的热备份。

DataNode：

数据存储节点，存储实际的数据

汇报存储信息给 NameNode。

Client：

切分文件

访问 HDFS
与 NameNode 交互，获取文件位置信息
与 DataNode 交互，读取和写入数据。

Block：
每块默认 64MB 大小
每块可以多个副本

MapReduce：

Job Tracker：
Master 节点，唯一
将任务分解成一系列任务，并分派给 Task Tracker
管理所有作业，同时负责监控、错误处理等

Task Tracker：
Slave 节点
运行 Map Task 和 Reduce Task
与 Job Tracker 交互，汇报任务状态。

Map Task：
解析每条数据记录，传递给用户编写的 map()并执行
将输出结果写入本地磁盘(如果为 map-only 作业，直接写入 HDFS)。

Reducer Task：
从 Map Task 的执行结果中，远程读取输入数据，对数据进行排序，将数据按照分组传递给用户编写的 reduce 函数执行。

Yarn：

ResourceManager：
处理客户端请求
启动、监控 ApplicationMaster
监控 NodeManager
资源分配不调度

NodeManager：
单个节点上的资源管理
处理来自 ResourceManager 的命令
处理来自 ApplicationMaster 的命令 Yarn 角色及概念

Container：
对任务运行行环境的抽象，封装了 CPU、内存等
多维资源以及环境变量、启动命令等任务运行相关的
信息资源分配不调度

ApplicationMaster：
数据切分
为应用程序申请资源,并分配给内部任务
任务监控不容错 Yarn 角色及概念

Client
用户与 YARN 交互的客户端程序
提交应用程序、监控应用程序状态，杀死应用程序等

分布式文件系统(Distributed File System)：

文件系统管理的物理存储资源可以是本地节点，也可以通过计算机网络与节点相连

对等特性允许一些系统扮演客户机和服务器的双重角色

特点：

有效解决数据的存储和管理难题

将固定于某个地点的某个文件系统,扩展到多地点多文件系统

每个节点可以分布在不同的地点，通过网络进行节点间的通信和数据传输

用户使用时就像使用本地文件系统一样管理和存储文件系统中的数据

Hadoop 部署：单机、伪分布式、完全分布式

Hadoop 单机模式部署：只需要配置好环境变量即可运行，一般用来学习和测试

1.安装 java 环境和配置主机名

```
[root@host01 ~]# yum -y install java-1.8.0-openjdk{,-devel}
[root@host01 ~]# cat /etc/hostname
    host01
[root@host01 ~]# cat /etc/hosts
    192.168.1.1 host01
```

2.下载解压即可使用

```
http://hadoop.apache.org/releases.html
[root@host01 ~]# tar -xf hadoop-3.0.3.tar.gz
[root@host01 ~]# mv hadoop-3.0.3 /usr/local/hadoop
```

3.修改配置文件

```
JAVA_HOME 通过 rpm -ql java-1.8.0-openjdk 查找
[root@host01 hadoop]# vim etc/hadoop/hadoop-env.sh
54:export
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.161-2.b14.el7.x86_64/jre
68:export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

4.测试

```
[root@host01 hadoop]# ./bin/hadoop version
[root@host01 hadoop]# ./bin/hadoop
jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-3.0.3.jar wordcount
test xx
```

Hadoop 伪分布式部署：所有角色安装在一台机器上，使用本地磁盘，一般用于测试

Xml 文件配置格式：

```
<configuration>
    <property>
        <name>关键字</name>
        <value>变量值</value>
        <description>描述</description>
    </property>
</configuration>
```

修改配置文件：

全局配置文件：etc/hadoop/core-site.xml

HDFS 配置文件：etc/hadoop/hdfs-site.xml

MapReduce 配置文件：etc/hadoop/mapred-site.xml

Yarn 配置文件：etc/hadoop/yarn-site.xml

Hadoop 完全分布式部署：HDFS+Yarn

1.安装 java 环境和配置主机名，解压软件包

```
[root@master ~]# cat /etc/hosts
    192.168.1.1 master
    192.168.1.2 node1
    192.168.1.3 node2
    192.168.1.4 node3
```

注意：

保证所有机器系统版本及 java 版本一致

保证所有主机 hadoop 路径一致，便于将配置文件复制到其他主机

保证所有主机的主机名和/etc/hosts 一致

保证 master 能用域名 ping 通所有主机

保证 node 能用域名 ping 通 master

保证所有主机间能密钥登录

2.指定数据节点：

```
[root@master hadoop]# vim etc/hadoop/workers
    node1
    node2
    node3
```

3.修改环境变量配置文件：

JAVA_HOME：java 的家目录

HADOOP_CONF_DIR：hadoop 的配置文件目录

```
[root@host01 hadoop]# vim etc/hadoop/hadoop-env.sh
54:export
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.161-2.b14.el7.x86_64/jre
68:export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop
```

4.修改主配置文件：

参考官方文档的 core-default.xml

fs.defaultFS：申明集群使用的文件系统

hadoop.tmp.dir：指定 hadoop 切片数据存储的根目录，通常该目录需要单独挂载一块硬盘

```
[root@master hadoop]# vim etc/hadoop/core-site.xml
    <configuration>
        <property>/var/hadoop
            <name>fs.defaultFS</name>
            <value>hdfs://master:9000</value>
        </property>
        <property>
            <name>hadoop.tmp.dir</name>
            <value>/var/hadoop</value>
        </property>
    </configuration>
```

5.修改主配置文件中申明的文件系统的配置文件

参考官方文档的 hdfs-default.xml

dfs.replication：指定切片的数量，默认是 3

dfs.namenode.http-address：指定 namenode 的地址和端口，默认端口 9870

dfs.namenode.secondary.http-address:指定 secondarynamenode 的地址和端口，默认端口 9868

```
[root@master hadoop]# vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
    <configuration>
        <property>
            <name>dfs.replication</name>
            <value>2</value>
        </property>
        <property>
            <name>dfs.namenode.http-address</name>
            <value>master:50070</value>
        </property>
        <property>
            <name>dfs.namenode.secondary.http-address</name>
            <value>master:50090</value>
        </property>
    </configuration>
```

6.将所有修改过的配置文件拷贝到其他节点上

7.在 namenode 上执行格式化操作

```
[root@master hadoop]# ./bin/hdfs namenode -format
```

8.如果启动失败

如果是 hdfs,修改 start-dfs.sh 和 stop-dfs.sh;如果是 yarn,修改 start-yarn.sh 和 stop-yarn.sh

```
[root@master hadoop]# vim sbin/start-dfs.sh #加 4 行，示例在 31 行
    HDFS_DATANODE_USER=root
    HADOOP_SECURE_DN_USER=hdfs
    HDFS_NAMENODE_USER=root
    HDFS_SECONDARYNAMENODE_USER=root
[root@master hadoop]# vim sbin/stop-dfs.sh  #同 start-dfs.sh
[root@master hadoop]# vim sbin/start-yarn.sh    #加 3 行
    YARN_RESOURCEMANAGER_USER=root
    HADOOP_SECURE_DN_USER=yarn
    YARN_NODEMANAGER_USER=root
[root@master hadoop]# vim sbin/stop-yarn.sh     #同 start-yarn.sh
```

9.启动 HDFS

```
[root@master hadoop]# ./sbin/start-dfs.sh
```

10.验证 HDFS

```
[root@fzr ~]# pssh -ih hosts jps
[root@master hadoop]# ./bin/hdfs dfsadmin -report
```

11.修改 yarn 配置文件：基于 mapreduce

参考官方文档的 mapred-default.xml 和 yarn-default.xml

mapreduce.framework.name：执行 MapReduce 的主机，参数可以是 local, classic 或 yarn

yarn.resourcemanager.hostname：配置 ResourceManager 的主机名

yarn.nodemanager.aux-services：

```
[root@master hadoop]# vim etc/hadoop/mapred-site.xml
    <configuration>
        <property>
            <name>mapreduce.framework.name</name>
            <value>yarn</value>
        </property>
    </configuration>
[root@master hadoop]# vim etc/hadoop/yarn-site.xml
    <configuration>
        <property>
            <name>yarn.resourcemanager.hostname</name>
            <value>master</value>
        </property>
        <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
        </property>
    </configuration>
```

12.拷贝步骤 11 修改的配置文件到其他节点上

13.启动 yarn

```
[root@master hadoop]# ./sbin/start-yarn.sh
```

14.测试

```
[root@master hadoop]# ./bin/yarn node -list
以下默认端口可通过查看官方文档
http://master:50070 #NameNode 端口
http://master:50090 #SecondaryNameNode 端口
http://master:8088      #ResourceManager 端口
http://node1:8042       #NodeManager 端口
http://node1:9864       #DataNode 端口
```

HDFS 的基本使用：

格式：./bin/hadoop fs 命令

常用命令：

- -ls 目录：查看集群目录
- -mkdir 目录：在集群中创建目录
- -rmdir 目录：在集群中删除目录
- -touchz 文件：创建文件
- -cat 文件：查看文件
- -rm 文件：删除文件
- -put 本地 目的路径：上传
- -get 目标路径 本地：下载，默认下载到当前路径

示例：

```
[root@master hadoop]# ./bin/hadoop fs -mkdir /test
```

```
[root@master hadoop]# ./bin/hadoop fs -ls /test
[root@master hadoop]# ./bin/hadoop fs -touchz /test/1
[root@master hadoop]# ./bin/hadoop fs -ls /test
[root@master hadoop]# ./bin/hadoop fs -rm /test/1
[root@master hadoop]# ./bin/hadoop fs -put *.txt /test
[root@master hadoop]# ./bin/hadoop fs -get /test/NOTICE.txt /tmp
```

NFS 网关：作为 client 和 HDFS 的中转

用途：

用户可以通过操作系统兼容的本地 NFSv3 客户端来阅览 HDFS 文件系统

用户可以从 HDFS 文件系统下载文档到本地文件系统

用户可以通过挂载点直接流化数据，支持文件附加

注意：

不支持随机写

在非安全模式,运行网关的用户是代理用户

在安全模式时，Kerberos keytab 中的用户是代理用户

配置说明：

nfs.aix.compatibility.mode.enabled：AIX 访问 NFS 网关

nfs.superuser：HDFS 超级用户是与 NameNode 进程本身具有相同标识的用户，可以执行任何操作

log4j.property 文件：调试和日志排错

log4j.logger.org.apache.hadoop.hdfs.nfs=DEBUG

log4j.logger.org.apache.hadoop.oncrpc=DEBUG

HDFS 节点管理：存储数据，所有节点变化会相对复杂

增加节点：

1.配置 hadoop 环境：主机名、域名解析、ssh 密钥、java 环境

```
[root@master ~]# vim /etc/hosts
    192.168.1.4 node3
```

2.解包安装 Hadoop

3.在 namenode 的 workers 增加该节点

```
[root@master ~]# vim /usr/local/hadoop/etc/hadoop/workers
+1:node4
```

4.将 namenode 的配置文件同步到新节点的目录下

```
[root@master ~]# rsync -avz --delete /usr/local/hadoop/
root@192.168.1.5:/usr/local/hadoop/
```

5.在新节点启动 Datanode

```
[root@node4 ~]# /usr/local/hadoop/bin/hdfs --daemon start datanode
```

管理节点：

同步带宽：67108864=64M

```
[root@master ~]# /usr/local/hadoop/bin/hdfs dfsadmin
-setBalancerBandwidth 67108864
```

同步数据：threshold 设置阀值，默认值为 5

```
[root@master ~]# /usr/local/hadoop/sbin/start-balancer.sh -threshold
5
```

查看集群状态：

```
[root@master ~]# /usr/local/hadoop/bin/hdfs dfsadmin -report
```

删除节点：

1. 修改配置文件

```
dfs.hosts.exclude：排除主机
[root@master hadoop]# vim etc/hadoop/hdfs-site.xml
    <property>
        <name>dfs.hosts.exclude</name>
        <value>/usr/local/hadoop/exclude</value>
    </property>
```

2. 创建/usr/local/hadoop/exclude 并写入要删除的主机

```
[root@master hadoop]# vim /usr/local/hadoop/exclude
    node4
```

3. 刷新节点：

```
[root@master hadoop]# ./bin/hdfs dfsadmin -refreshNodes
```

4. 等待节点状态变更

```
[root@master ~]# /usr/local/hadoop/bin/hdfs dfsadmin -report
Normal(正常)-->Decommission in progress(数据迁
移)-->Decommissioned(已下线)
```

5. 停止该节点的服务

```
[root@node4 ~]# /usr/local/hadoop/bin/hdfs --daemon stop datanode
```

6. 删除 workers 中的该节点

```
[root@node1 ~]# vim /usr/local/hadoop/etc/hadoop/workers
rsync 同步到其他节点
```

修复节点：重新配置 datanode，启动服务，保证主机名和 ip 与原节点相同，同步带宽和数据

Yarn 节点管理：只是计算节点，不存储数据，节点的数量只影响计算速度

增加节点：/usr/local/hadoop/bin/yarn --daemon start nodemanager

删除节点：/usr/local/hadoop/bin/yarn --daemon stop nodemanager

查看节点：/usr/local/hadoop/bin/yarn node -list

NameNode 高可用：官方提供两种解决方案

NFS 方案：DRBD 解决 NFS 高可用，Hadoop 把数据存储在共享存储里

QJM 方案：需要让每一个 datanode 知道活跃和备用 namenode 的位置，并把信息同时发给两个 namenode

配置 NFS 网关：

1. 在 NFS 网关上配置域名解析：

```
[root@nfsgw ~]# vim /etc/hosts
    192.168.1.1 master
    192.168.1.2 node1
    192.168.1.3 node2
    192.168.1.4 node3
```

2. 在 NFS 和 master 上添加代理用户，两台机的 uid 和 gid 必须一致

```
[root@nfsgw ~]# useradd -g 100 test      #指定组方便管理
[root@nfsgw ~]# id test
    uid=1000(test) gid=100(users) 组=100(users)
```

```
[root@master ~]# useradd -u 1000 -g 100 test
```

3.修改主配置文件：若集群已启动，则先停止

hadoop.proxyuser.用户名.hosts：指定代理主机的地址，*表示所有

hadoop.proxyuser.用户名.groups：指定代理用户的 gid，多个 gid 用,隔开，*表示所有

```
[root@master hadoop]# ./sbin/stop-all.sh
[root@master hadoop]# vim etc/hadoop/core-site.xml
    <property>
        <name>hadoop.proxyuser.test.groups</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.test.hosts</name>
        <value>*</value>
    </property>
```

4.将修改后的 core-site.xml 复制到其他主机

5.启动集群：

```
[root@master hadoop]# ./sbin/start-all.sh
```

6.在 NFS 上安装 java 依赖包

```
[root@nfsgw ~]# yum -y install java-1.8.0-openjdk-devel
```

7.将 master 上的 hadoop 同步给 NFS

```
[root@master ~]# rsync -avz --delete /usr/local/hadoop/
root@192.168.1.15:/usr/local/hadoop/
```

8.删除 NFS 上的 log 目录并重新创建，创建配置文件中定义的目录

```
[root@nfsgw ~]# rm -rf /usr/local/hadoop/logs
[root@nfsgw ~]# mkdir /usr/local/hadoop/logs
[root@nfsgw ~]# setfacl -m u:test:rwx /usr/local/hadoop/logs
[root@nfsgw ~]# mkdir /var/hadoop /var/nfstemp
[root@nfsgw ~]# chown test:users /var/nfstemp
```

9.编辑 NFS 的 hdfs 配置文件：

dfs.namenode.accesstime.precision：超时时间，默认为 3600000，如果挂载时需要设置 noatime，则使用默认值

nfs.dump.dir：文件转储目录配置，常用于临时存储无序的写操作，路由有延迟

nfs.port.monitoring.disabled：是否允许非特定端口的 NFS 访问

nfs.rtmax：NFS 传输支持的最大读取请求，单位字节，4194304=4M

nfs.wtmax：NFS 传输支持的最大写入请求，单位字节，1048576=1M

nfs.exports.allowed.hosts：设置挂载策略配置

值的格式：机器名 访问策略;机器名 访问策略...

机器名：单一的主机，Java 正则表达式或者 IPv4 地址

访问策略：rw 或 ro，默认是 ro

```
[root@nfsgw hadoop]# vim etc/hadoop/hdfs-site.xml
    <property>
        <name>nfs.exports.allowed.hosts</name>
        <value>* rw</value>
```

```
    </property>
    <property>
        <name>dfs.namenode.accesstime.precision</name>
        <value>3600000</value>
    </property>
    <property>
        <name>nfs.dump.dir</name>
        <value>/var/nfstemp/test</value>
    </property>
    <property>
        <name>nfs.rtmax</name>
        <value>4194304</value>
    </property>
    <property>
        <name>nfs.wtmax</name>
        <value>1048576</value>
    </property>
    <property>
        <name>nfs.port.monitoring.disabled</name>
        <value>false</value>
    </property>
```

10. 如果 portmap 启动失败，检查 rpcbind 和 nfs-utils

```
[root@nfsgw ~]# yum -y remove rpcbind nfs-utils
```

11. 使用 root 启动 portmap 服务，stop 关闭服务

```
[root@nfsgw hadoop]# ./bin/hdfs --daemon start portmap
```

12. 使用 test 启动 nfs3 服务，stop 关闭服务

```
[test@nfsgw hadoop]$ ./bin/hdfs --daemon start nfs3
```

13. 测试：分别使用 root 和 test 用户执行 jps 命令
14. 客户端挂载测试

    挂载参数说明：

    vers=3：协议版本号为 v3

    proto=tcp：目前仅使用 TCP 作为传输协议

    nolock：不支持 NLM，禁用

    noacl：禁用扩展权限

    noatime：禁用 access time 时间戳

    sync：同步客户端，尽可能避免重新排序写入

```
[root@client ~]# yum -y install nfs-utils
[root@client ~]# vim /etc/fstab
    192.168.1.15:/ /mnt nfs vers=3,proto=tcp,nolock,noacl,noatime,sync 0
0
```

zookeeper：

分布式的、开源的分布式应用程序协调服务

保证数据在集群间的事务性一致

角色：

Leader：接受所有 Follower 的提案请求并统一协调发起提案的投票，负责与所有的 Follower 进行内部的数据交换

Follower：直接为客户端服务并参与提案的投票，同时与 Leader 进行数据交换

Observer：直接为客户端服务但并不参与提案的投票，同时也与 Leader 进行数据交换

选举：

服务在启动的时候没有角色(LOOKING)

角色是通过选举产生的

选举产生一个 leader，剩下的全是 follower

选举 leader 原则：集群中得到(n/2)+1 台服务器投票

如果 leader 死亡，重新选举 leader

如果死亡的机器数量达到一半，集群挂起

如果无法得到足够的投票数量，就重新发起投票，如果参与投票的机器不足(n/2)+1，则集群停止工作

如果 follower 死亡过多，剩余机器不足(n/2)+1 与，则集群停止工作（observer 不计算在投票总设备数量里面）

可伸缩扩展性原理：

leader 负责所有写相关操作

follower 负责读操作，响应 leader 提议和接受 leader 通知

Observer 负责读操作和接受 leader 通知

工作原理：

客户端提交读请求，则由每台服务器的本地副本数据库直接响应

客户端提交写请求,转发给 Leader,由 Leader 根据该请求发起投票,所有 follower 进行投票，当票数过半时 Leader 会向所有的服务器发送一个通知消息。当 Client 所连接的服务器收到该消息时，会把该操作更新到内存中并对 Client 的写请求做出回应

提供的服务：

1.统一命名服务。有一组服务器向客户端提供某种服务，我们希望客户端每次请求服务端都可以找到服务端集群中某一台服务器,这样服务端就可以向客户端提供客户端所需的服务。对于这种场景，我们的程序中一定有一份这组服务器的列表，每次客户端请求时候，都是从这份列表里读取这份服务器列表。列表的高可用是分布式存储的，它是由存储这份列表的服务器共同管理的，如果存储列表里的某台服务器坏掉了，其他服务器马上可以替代坏掉的服务器，并且可以把坏掉的服务器从列表里删除掉，让故障服务器退出整个集群的运行，而这一切的操作又不会由故障的服务器来操作，而是集群里正常的服务器来完成。这是一种主动的分布式数据结构，能够在外部情况发生变化时候主动修改数据项状态的数据机构。Zookeeper 框架提供了这种服务。这种服务名字就是：统一命名服务，它和 javaEE 里的 JNDI 服务很像。

2.分布式锁服务。当分布式系统操作数据，例如：读取数据、分析数据、最后修改数据。在分布式系统里这些操作可能会分散到集群里不同的节点上，那么这时候就存在数据操作过程中一致性的问题，如果不一致，我们将会得到一个错误的运算结果，在单一进程的程序里，一致性的问题很好解决，但是到了分布式系统就比较困难，因为分布式系统里不同服务器的运算都是在独立的进程里,运算的中间结果和过程还要通过网络进行传递，那么想做到数据操作一致性要困难的多。Zookeeper 提供了一个锁服务解决了这样的问题，能让我们在做分布式数据运算时候，保证数据操作的一致性。

3.配置管理。在分布式系统里，我们会把一个服务应用分别部署到 n 台服务器上，

这些服务器的配置文件是相同的（例如：我设计的分布式网站框架里，服务端就有 4 台服务器，4 台服务器上的程序都是一样，配置文件都是一样），如果配置文件的配置选项发生变化，那么我们就得一个个去改这些配置文件，如果我们需要改的服务器比较少，这些操作还不是太麻烦，如果我们分布式的服务器特别多，比如某些大型互联网公司的 hadoop 集群有数千台服务器，那么更改配置选项就是一件麻烦而且危险的事情。这时候 zookeeper 就可以派上用场了，我们可以把 zookeeper 当成一个高可用的配置存储器，把这样的事情交给 zookeeper 进行管理，我们将集群的配置文件拷贝到 zookeeper 的文件系统的某个节点上，然后用 zookeeper 监控所有分布式系统里配置文件的状态，一旦发现有配置文件发生了变化，每台服务器都会收到 zookeeper 的通知，让每台服务器同步 zookeeper 里的配置文件，zookeeper 服务也会保证同步操作原子性，确保每个服务器的配置文件都能被正确的更新。

4.集群管理。为分布式系统提供故障修复的功能。在分布式的集群中，集群管理最麻烦的事情就是节点故障管理，比如硬件故障，软件故障，网络问题，有新的节点加入进来，也有老的节点退出集群。这个时候，集群中其他机器需要感知到这种变化，然后根据这种变化做出对应的决策。zookeeper 可以让集群选出一个健康的节点作为 master，master 节点会知道当前集群的每台服务器的运行状况，一旦某个节点发生故障，master 会把这个情况通知给集群其他服务器，从而重新分配不同节点的计算任务。Zookeeper 不仅可以发现故障，也会对有故障的服务器进行甄别，看故障服务器是什么样的故障，如果该故障可以修复，zookeeper 可以自动修复或者告诉系统管理员错误的原因让管理员迅速定位问题，修复节点的故障。

特点：

一个精简的文件系统。这点它和 hadoop 有点像，但是 zookeeper 这个文件系统是管理小文件的，而 hadoop 是管理超大文件的。

提供了丰富的“构件”，这些构件可以实现很多协调数据结构和协议的操作。例如：分布式队列、分布式锁以及一组同级节点的“领导者选举”算法。

高可用的，它本身的稳定性很好，分布式集群完全可以依赖 zookeeper 集群的管理，利用 zookeeper 避免分布式系统的单点故障的问题。

采用了松耦合的交互模式。这点在 zookeeper 提供分布式锁上表现最为明显，zookeeper 可以被用作一个约会机制，让参入的进程不在了解其他进程的（或网络）的情况下能够彼此发现并进行交互，参入的各方甚至不必同时存在，只要在 zookeeper 留下一条消息，在该进程结束后，另外一个进程还可以读取这条信息，从而解耦了各个节点之间的关系。

为集群提供了一个共享存储库，集群可以从这里集中读写共享的信息，避免了每个节点的共享操作编程，减轻了分布式系统的开发难度。

设计采用的是观察者的设计模式，zookeeper 主要是负责存储和管理大家关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper 就将负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式。

安装 zookeeper 集群：

1.下载地址：http://zookeeper.apache.org/releases.html

2.环境配置：openjdk 环境，ip，主机名，域名解析

3.停止 hadoop 集群：

```
[root@master hadoop]# sbin/stop-all.sh
```

4.在所有主机安装 zookeeper

```
[root@fzr ~]# for i in {1..5};do scp zookeeper-3.4.12.tar.gz
192.168.1.${i}:~ ; done
[root@fzr ~]# pssh -ih hosts "tar -xf zookeeper-3.4.12.tar.gz"
[root@fzr ~]# pssh -ih hosts mv zookeeper-3.4.12 /usr/local/zookeeper
```

5.修改配置文件

```
[root@fzr ~]# pssh -ih hosts cp /usr/local/zookeeper/conf/zoo_sample.cfg
/usr/local/zookeeper/conf/zoo.cfg
[root@master ~]# vim /usr/local/zookeeper/conf/zoo.cfg
    server.1=node1:2888:3888
    server.2=node2:2888:3888
    server.3=node3:2888:3888
    server.4=node4:2888:3888
    server.5=master:2888:3888:observer
[root@master ~]# for i in {2..5};do scp /usr/local/zookeeper/conf/zoo.cfg
192.168.1.${i}:/usr/local/zookeeper/conf/zoo.cfg ;done
```

6.创建目录/tmp/zookeeper，该目录在 zoo.cfg 的 dataDir 字段配置

```
[root@fzr ~]# pssh -ih hosts mkdir /tmp/zookeeper
```

7.在该目录下创建 myid 文件，myid 文件中 id 必须匹配 zoo.cfg 的 server.id 中的 id

```
[root@master ~]# echo 5 > /tmp/zookeeper/myid
[root@node1 ~]# echo 1 > /tmp/zookeeper/myid
[root@node2 ~]# echo 2 > /tmp/zookeeper/myid
[root@node3 ~]# echo 3 > /tmp/zookeeper/myid
[root@node4 ~]# echo 4 > /tmp/zookeeper/myid
```

8.在所有主机启动 zookeeper

```
[root@fzr ~]# pssh -ih hosts /usr/local/zookeeper/bin/zkServer.sh start
```

9.等待所有主机启动成功后查看角色

```
[root@fzr ~]# pssh -ih hosts "ss -antup | grep 2181"
[root@fzr ~]# pssh -ih hosts /usr/local/zookeeper/bin/zkServer.sh status
    Mode: leader/follower/observer
```

bin/zkCli.sh：ZooKeeper 简单客户端

启动参数：

-timeout：当前会话的超时时间，zookeper 依靠与客户端的心跳来判断会话是否有效，单位是毫秒。0 表示不超时

-r：只读模式，zookeeper 的只读模式指一个服务器与集群中过半机器失去连接以后，这个服务器就不再处理客户端的请求，但我们仍然希望该服务器可以提供读服务。

-server：zookeeper 服务器 ip 地址和端口号，默认为 localhost:2181

登陆后，会进入如下界面：

[zk: localhost:2181(CONNECTED) 0]，

IP：端口号(连接状态) 已执行的命令数

命令：

h：查看帮助

注：所有的 path 均指绝对路径，zk 不接受相对路径和.或..

ls path：查看指定路径下的节点，初始根节点下只有 zookeeper 一个节点，是 zk 默认创建的，用于存储节点的一些状态信息

stat path [watch]：查看指定路径节点的状态信息

[watch]：监听节点状态变化

所列信息说明：

czxid：节点被创建的事务 ID

ctime: 创建时间

mzxid: 最后一次被更新的事务 ID

mtime: 修改时间

pzxid：子节点列表最后一次被更新的事务 ID

cversion：子节点的版本号

dataversion：数据版本号

aclversion：权限版本号

ephemeralOwner：用于临时节点，代表临时节点的事务 ID，如果为持久节点则为 0

dataLength：节点存储的数据的长度

numChildren：当前节点的子节点个数

ls2 path [watch]:同时列出子节点和这些节点的状态信息，相当于 ls 和 stat 的组合

get path [watch]：获取指定路径节点的数据内容

[watch]：监听节点数据变化。如果其它客户端修改了该节点的数据，则会通知监听了该节点的所有客户端

set path "data" [version]：修改当前节点的数据内容

data：需要写入节点的数据

version：默认覆盖原有数据后版本号+1。如果指定版本，需要和当前节点的数据版本一致，防止覆盖其他节点写入的数据

create [-s|-e] path "data" acl：

-s：创建顺序节点，默认选项，创建同名的节点时，会给节点自动加上编号

-e：创建临时节点，在客户端结束与服务器的会话后自动消失

data：需要写入节点的数据

acl：节点权限，默认所有人都可以对该节点进行读写操作

setquota -n|-b val path：设置节点配额，对节点的数据和子节点的数量规定阀值，超出后在日志中记录警告信息

-n：限制子节点的数量

-b：限制节点的数据长度

val：限制的值

listquota path：查看路径节点的配额信息

delquota [-n|-b] path：删除节点配额

-n：删除子节点数量配额限制

-b：删除节点数据长度配额限制

delete path [version]：删除没有子节点的节点

version：节点版本号，版本号和服务器的版本号一致

rmr path：删除节点及其所有子节点

setAcl path acl:设置节点 ACL

acl：ACL 权限模式：

c(create)：创建子节点

d(delete)：删除子节点

r(read)：读取节点数据

w(write)：修改节点数据
a(admin)：设置子节点权限

getAcl path：获取节点 ACL

addauth scheme auth：给当前客户端添加授权信息

scheme：授权方式，一般为 digest(用户名/密码授权)，

auth：授权信息，格式：用户名:密码

history：查看历史命令

redo cmdno：执行历史命令

printwatches on|off：监听日志，默认打开，用于 watch

sync path：与 leader 同步数据

connect host:port：连接到其他 zk 服务器

close：关闭当前客户端连接

quit：退出 zkCli.sh

kafka：

由 LinkedIn 开发的一个分布式的消息系统，使用 Scala 编写，中间件

优点：解耦、冗余、提高扩展性、缓冲、保证顺序、灵活、削峰填谷、异步通信

角色：

producer:生产者,负责发布消息

consumer:消费者,负责读取处理消息

topic:消息的类别

Parition:分区，每个 Topic 包含一个或多个分区

Broker:Kafka 服务器集群，可以是一个或多个

Kafka 通过 Zookeeper 管理集群配置，选举 leader

kafka 集群安装：

1.准备一个 zookeeper 集群

2.下载，解压，安装 kafka

http://kafka.apache.org/downloads

```
[root@fzr ~]# pscp.pssh -h hosts hadoop/kafka_2.12-1.1.0.tgz ~
[root@fzr ~]# pssh -ih hosts tar -xf kafka_2.12-1.1.0.tgz
[root@fzr ~]# pssh -ih hosts mv kafka_2.12-1.1.0 /usr/local/kafka
```

3.修改 server.properties 配置文件：

broker.id 是每台服务器的 id，不能相同

zookeeper.connect：集群地址，申明一部分即可

```
[root@fzr ~]# for i in {1..5};do pssh -iH 192.168.1.$i sed -i "21s/0/$i/
/usr/local/kafka/config/server.properties";pssh -iH 192.168.1.$i sed -i
"123s/localhost:2181/node1:2181,node2:2181,node3:2181/
/usr/local/kafka/config/server.properties";done
```

4.启动服务：需要至少 2G 内存

```
[root@fzr ~]# pssh -ih hosts "/usr/local/kafka/bin/kafka-server-start.sh
-daemon /usr/local/kafka/config/server.properties"
```

5.验证

```
[root@fzr ~]# pssh -ih hosts "ss -antup | grep 9092"
[root@fzr ~]# pssh -ih hosts jps
```

kafka 使用：

创建主题：

[root@node1 kafka]# ./bin/kafka-topics.sh --create --partitions 2 --replication-factor 2 --zookeeper node1:2181 --topic test

查看已存在的主题：

[root@node1 kafka]# ./bin/kafka-topics.sh --list --zookeeper node1:2181

查看主题详细信息：

[root@node1 kafka]# ./bin/kafka-topics.sh --describe --zookeeper node1:2181 --totopic test

生产者发布消息：

[root@node1 kafka]# ./bin/kafka-console-producer.sh --broker-list master:9092,de1:9092 --topic test

消费者读取信息：

--from-beginning：从头开始读取信息

[root@node2 kafka]# ./bin/kafka-console-consumer.sh --bootstrap-server node2:92,node3:9092 --topic test

NameNode 高可用方案(QJM)：

组成：

Active NameNode：对外提供服务，响应集群中所有的客户端

Standby NameNode：仅同步 Active NameNode 的状态，保证在必要的时候提供一个快速的转移。

原理：

1.与称为 JNS(Journal Nodes)的互相独立的进程保持通信

2.当 Active Node 上更新了 namespace，它将记录修改日志发送给 JNS

3.Standby noes 将会从 JNS 中读取这些修改

4.当宕机发生时，Standby noes 确保从 JNS 中读取所有修改后提升为 Standby Node

5.DataNodes 上需要同时配置这两个 Namenode 的地址

6.任何时刻，只能有一个 Active NameNode

部署 QJM：

1.环境配置：hosts，密钥，hadoop，kafka，zookeeper

192.168.1.1 master

192.168.1.2 node1

192.168.1.3 node2

192.168.1.4 node3

192.168.1.5 node4

192.168.1.20 standby

2.删除之前实验的 hdfs 集群数据

[root@fzr ~]# pssh -ih hosts "rm -rf /var/hadoop/*"

3.配置 hadoop-env.sh：与 hadoop 的配置相同

export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.161-2.b14.el7.x86_64/jre

export HADOOP_CONF_DIR=/usr/local/hadoop/etc/hadoop

4.配置 core-site.xml

fs.defaultFS：跟组名，组名在 hdfs-site.xml 中定义

hadoop.tmp.dir：指定 hadoop 切片数据存储的根目录，通常该目录需要单独挂载一块硬盘

ha.zookeeper.quorum：zookeeper 服务地址列表

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://test</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/var/hadoop</value>
    </property>
    <property>
        <name>ha.zookeeper.quorum</name>
        <value>node1:2181,node2:2181,node3:2181</value>
    </property>
</configuration>
```

5.配置 hdfs-site.xml

参考官方文档的 NameNode HA With QJM

dfs.nameservices：定义主机组的名称

dfs.ha.namenode.组名：申明该组的类型是高可用，申明集群的两个 NameNode 的名称分别为 nn1,nn2

dfs.namenode.rpc-address：申明 namenode 的 rpc 通信地址，默认端口是 8020

dfs.namenode.http-address：申明 namenode 的 http 通信地址，默认端口是 9870

dfs.namenode.shared.edits.dir:指定 namenode 元数据存储在 journalnode 中的路径

dfs.journalnode.edits.dir：指定 journalnode 日志文件存储的路径

dfs.client.failover.proxy.provider：指定 HDFS 客户端连接 active namenode 的 java 类

dfs.ha.fencing.methods：配置隔离机制为 ssh

dfs.ha.fencing.ssh.private-key-files：指定秘钥的位置

dfs.ha.automatic-failover.enabled：自动故障转移，默认关闭

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
    <property>
        <name>dfs.nameservices</name>**********
        <value>test</value>
    </property>
    <property>
        <name>dfs.ha.namenodes.test</name>
        <value>nn1,nn2</value>
```

```
        </property>
        <property>
            <name>dfs.namenode.rpc-address.test.nn1</name>
            <value>master:8020</value>
        </property>
        <property>
            <name>dfs.namenode.rpc-address.test.nn2</name>
            <value>standby:8020</value>
        </property>
        <property>
            <name>dfs.namenode.http-address.test.nn1</name>
            <value>master:9870</value>
        </property>
        <property>
            <name>dfs.namenode.http-address.test.nn2</name>
            <value>standby:9870</value>
        </property>
        <property>
            <name>dfs.namenode.shared.edits.dir</name>
            <value>qjournal://node1:8485;node2:8485;node3:8485/test</value>
        </property>
        <property>
            <name>dfs.journalnode.edits.dir</name>
            <value>/var/hadoop/journal</value>
        </property>
        <property>
            <name>dfs.client.failover.proxy.provider.test</name>
            <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFail
        overProxyProvider</value>
        </property>
        <property>
            <name>dfs.ha.fencing.methods</name>
            <value>sshfence</value>
        </property>
        <property>
            <name>dfs.ha.fencing.ssh.private-key-files</name>
            <value>/root/.ssh/id_rsa</value>
        </property>
        <property>
            <name>dfs.ha.automatic-failover.enabled</name>
            <value>true</value>
        </property>
    </configuration>
```

6. 配置 mapred-site.xml：与 hadoop 配置一样，不修改

mapreduce.framework.name：执行 MapReduce 的主机

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

7.配置 yarn-site.xml：

参考官方文档的 ResourceManager HA 和 yarn-site.xml

yarn.nodemanager.aux-services：服务列表，使用逗号分隔

yarn.resourcemanager.ha.enabled：是否启用 RM 高可用，默认不启动

yarn.resourcemanager.ha.rm-ids：用于列出 RM 集合中的节点

yarn.resourcemanager.recovery.enabled：RM 启动后是否恢复状态，默认不启动

yarn.resourcemanager.store.class：指定持续存储的类

yarn.resourcemanager.zk-address：指定持续存储的 zookeeper 地址

yarn.resourcemanager.cluster-id：集群的名称

yarn.resourcemanager.hostname：RM 的主机名

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.resourcemanager.ha.enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.resourcemanager.ha.rm-ids</name>
        <value>rm1,rm2</value>
    </property>
    <property>
        <name>yarn.resourcemanager.recovery.enabled</name>
        <value>true</value>
    </property>
    <property>
        <name>yarn.resourcemanager.store.class</name>
        <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.Z
    KRMStateStore</value>
    </property>
    <property>
        <name>yarn.resourcemanager.zk-address</name>
        <value>node1:2181,node2:2181,node3:2181</value>
    </property>
    <property>
```

```
            <name>yarn.resourcemanager.cluster-id</name>
            <value>yarn-ha</value>
        </property>
        <property>
            <name>yarn.resourcemanager.hostname.rm1</name>
            <value>master</value>
        </property>
        <property>
            <name>yarn.resourcemanager.hostname.rm2</name>
            <value>standby</value>
        </property>
    </configuration>
```

8. 配置 workers：

```
node1
node2
node3
node4
```

9. 同步配置文件到其他主机：

```
[root@master ~]# for i in 2 3 4 5 20;do rsync -avz
/usr/local/hadoop/etc/hadoop/
root@192.168.1.${i}:/usr/local/hadoop/etc/hadoop/; done
```

10. 在 master 上初始化 zookeeper

```
[root@master hadoop]# ./bin/hdfs zkfc -formatZK
```

11. 在 node 子节点上启动 journal

```
[root@fzr ~]# for i in node{1..4};do pssh -iH $i /usr/local/hadoop/bin/hdfs
--daemon start journalnode;done
```

12. 格式化 master 节点

```
[root@master hadoop]# ./bin/hdfs namenode -format
```

13. 将 master 的/var/hadoop/目录下的内容同步到 standby

```
[root@master ~]# rsync -avz /var/hadoop/ root@192.168.1.20:/var/hadoop/
```

14. master 初始化 JNS

```
[root@master hadoop]# ./bin/hdfs namenode  -initializeSharedEdits
```

15. 在 node 子节点上停止 journalnode

```
[root@fzr ~]# for i in node{1..4};do pssh -iH $i /usr/local/hadoop/bin/hdfs
--daemon stop journalnode;done
```

16. master 上启动 hadoop

```
[root@master ~]# /usr/local/hadoop/sbin/start-all.sh
```

根据错误提示定义：

```
[root@master ~]# vim /usr/local/hadoop/sbin/start-dfs.sh
    HDFS_DATANODE_USER=root
    HADOOP_SECURE_DN_USER=hdfs
    HDFS_NAMENODE_USER=root
    HDFS_SECONDARYNAMENODE_USER=root
    HDFS_JOURNALNODE_USER=root
```

```
    HDFS_ZKFC_USER=root
/usr/local/hadoop/sbin/stop-dfs.sh  #同 start-dfs.sh
[root@standby ~]# vim /usr/local/hadoop/sbin/start-yarn.sh
    YARN_RESOURCEMANAGER_USER=root
    HADOOP_SECURE_DN_USER=yarn
    YARN_NODEMANAGER_USER=root
/usr/local/hadoop/sbin/stop-yarn.sh #同 start-yarn.sh
```

17. standby 上启动 Yarn 的 ResourceManager

```
[root@standby ~]# /usr/local/hadoop/bin/yarn --daemon start resourcemanager
```

18. 查看集群状态

```
[root@master hadoop]# ./bin/hdfs haadmin -getServiceState nn1
[root@master hadoop]# ./bin/hdfs haadmin -getServiceState nn2
[root@master hadoop]# ./bin/yarn rmadmin -getServiceState rm1
[root@master hadoop]# ./bin/yarn rmadmin -getServiceState rm2
[root@master hadoop]# ./bin/hdfs dfsadmin -report
[root@master hadoop]# ./bin/yarn node -list
```

19. 测试集群以及高可用

```
[root@master hadoop]# ./bin/hadoop fs -ls /
```

Jenkins：

由 java 编写的一款开源软件

用于 CI(持续集成)工作，监视重复工作的执行。例如软件工程的构建，计划任务下设置的工作

CI 工具通过自动构建和自动测试来验证结果。可以检测到当前程序代码的问题，迅速提供反馈

特点：

- 简单、可扩展、用户界面友好
- 支持各种 SCM(软件配置管理)工具：SVN、GIT、CVS 等
- 能够构建各种风格的项目
- 可以选择安装多种插件
- 跨平台：几乎可以支持所有的平台

下载：https://pkg.jenkins.io/redhat-stable/

下载 rpm 包，装包起服务，需要 java 环境

网页访问，默认端口为 8080

初始密码在/var/lib/jenkins/secrets/initialAdminPassword

CI/CD(Continuous Integration/Continuous Delivery or Deployment：持续集成/持续交付 or 持续部署)

Jenkins 分发服务器管理：

优化构建工程：

- 在 Jenkins 服务器上安装 apache，用于分发应用程序
- 打包成为一个文件，并为其生成 md5 值

配置分发服务器：

- 配置 http，设定下载目录为/var/www/html/deploy/packages

```
[root@host ~]# mkdir -pv /var/www/html/deploy/packages
[root@host ~]# chown -R jenkins: /var/www/html/deploy/
[root@host ~]# systemctl start httpd
```

新建工程：

General-->参数化构建过程-->设置名字：test；设置参数类型 branch or tag

-->源码管理-->Git-->Repository URL:从 gitlab 中复制地址;Branches to build: ${名字}；Additional Behaviours：项目名${名字}

-->构建-->执行 shell：

```
deploy_dir=/var/www/html/deploy/packages
cp -r myproject${test} $deploy_dir
rm -rf $deploy_dir/myproject${test}/.git
cd $deploy_dir
tar -czf myproject${test}.tar.gz myproject${test}
rm -rf myproject${test}
md5sum myproject${test}.tar.gz | awk '{print $1}' >
myproject${test}.tar.gz.md5
[ -f live_version ] && cat live_version > last_version
echo ${test} > live_version
```

-->应用-->保存

-->Build with Parameters-->选择标签-->构建

-->查看目录：ls /var/www/html/deploy/packages/

自动切换或回滚版本：

根据生成的 live_version 和 last_version 确定版本，下载后根据 md5 进行校验

编程发布不同的版本：

```
import requests
from urllib import request
import hashlib
import os
import tarfile


def get_web(url):
    r = requests.get(url=url)
    return r.text


def download(url, fname):
    html = request.urlopen(url=url)
    with open(fname, 'wb')as f:
        while True:
            data = html.read(1024)
            if not data:
                break
            f.write(data)
```

```
def check_md5(fname):
    m = hashlib.md5()
    with open(fname, 'rb')as f:
        while True:
            data = f.read(4096)
            if not data:
                break
            m.update(data)
    return m.hexdigest()


def deploy(soft):
    os.chdir('/var/www/html/deploy/packages/')
    tar = tarfile.open(soft, 'r:gz')
    tar.extractall()
    tar.close()
    src = soft
    dst = '/var/www/html/app'
    if os.path.exists(dst):
        os.unlink(dst)
    os.link(src, dst)


if __name__ == '__main__':
    ver =
get_web('http://192.168.1.161/deploy/packages/live_version').strip()
    soft_name = 'myproject%s.tar.gz' % ver
    soft_url = 'http://192.168.1.161/deploy/packages/' + soft_name
    soft_path = os.path.join('/var/www/html/packages/', soft_name)
    download(soft_url, soft_path)
    local_md5 = check_md5(soft_path)
    get_md5 = get_web(soft_url + '.md5').strip()
    if local_md5 == get_md5:
        deploy(soft_path)
```

Dubbo(开源分布式服务框架)：

高性能优秀的服务框架，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 Spring 框架无缝集成。

下载地址：

https://github.com/apache/incubator-dubbo

https://github.com/alibaba/dubbo

注：需要通过 Maven 编译

主要核心部件：

Remoting：网络通信框架，实现了 sync-over-async 和 request-response 消息机制

RPC：一个远程过程调用的抽象，支持负载均衡、容灾和集群功能

Registry：服务目录框架用于服务的注册和服务事件发布和订阅

工作原理：

Provider(服务提供者)：暴露服务方。

Consumer(服务消费者)：调用远程服务方。

Registry(服务注册中心)：服务注册与发现的中心目录服务。

Monitor(服务监控中心)：统计服务的调用次数和调用时间的日志服务。