\- -

\- - - - - - Linux 基础

电脑启动过程

服务器优化方案

硬盘存储方案

1 > iSCSI 网络磁盘

服务器的存储空间分配给客户机使用，客户机就可以像使用本地硬盘一样使用 iscsi 磁盘.

Multipath 多路径

主要功能

冗余：主备模式，高可用

改进性能：主主模式，负载均衡

2 > RAID（廉价冗余磁盘阵列）

可以把硬盘整合成一个大磁盘，还可以在大磁盘上再分区，放数据还有一个大功能，多块盘放在一起可以有冗余（备份）

RAID 整合方式有很多，常用的：0 1 5 10

RAID 0，可以是一块盘和 N 个盘组合

其优点读写快，是 RAID 中最好的

缺点：没有冗余，一块坏了数据就全没有了

RAID 1，只能 2 块盘，盘的大小可以不一样，以小的为准

10G+10G 只有 10G，另一个做备份。它有 100%的冗余，缺点：浪费资源，成本高

RAID 5 ，3 块盘，容量计算 10*（n-1）,损失一块盘

特点，读写性能一般，读还好一点，写不好

冗余从好到坏：RAID1 RAID10 RAID 5 RAID0

性能从好到坏：RAID0 RAID10 RAID5 RAID1

成本从低到高：RAID0 RAID5 RAID1 RAID10

3 > Ceph

分布式文件系统具有高扩展、高可用、高性能的特点，可以提供对象存储、块存储、文件系统存储

Ceph 可以提供 PB 级别的存储空间 (PB--->TB--->GB) 软件定义存储 (Software Defined Storage）

常用分布式文件系统：Lustre、Hadoop、FastDFS、Ceph、GlusterFS

ceph 组件

OSDs 存储设备

Monitros 集群监控组建

MDSs　　存放文件系统的元数据（对象存储和块存储不需要该组件）
Client　　ceph 客户端

硬盘分区方式:
MBR/msdos,1~4 个主分区,最大支持容量为 2.2TB 的磁盘
GPT：128 个主分区， 最大支持容量为 18EB 的磁盘
--fdisk part
LVM 逻辑卷
作用： 1.整合分散的空间
2.逻辑卷空间可以扩大
新建逻辑卷：将众多的物理卷(pv)组成卷组(vg)，再从卷组中划分逻辑卷(lv)

swap 交换分区

以空闲分区当做交换空间，缓解真实物理内存的压力

服务协议及端口

http :超文本传输协议 80
FTP ：文件传输协议 21
https ：安全的超文本传输协议 443
DNS :域名解析协议 53
telnet :远程管理协议 23
smtp ：邮件协议，用户发邮件协议 25
pop3 ：邮件协议，用户收邮件协议 110
tftp ：简单文件传输协议 69
ssh ：远程登陆 22

DNS 服务

递归解析：指 DNS 服务器与其他 DNS 服务器交互，最终将解析结果带回来的过程
迭代解析：指 DNS 服务器与其他 DNS 服务器交互，最终将告知下一个 DNS 服务
缓存 DNS
1.全局转发，所有的 DNS 解析请求全部转发给公网 DNS
2.根域迭代，所有的 DNS 解析请求全部发给根域 DNS 服务器

```
options {
        directory       "/var/named";
        forwarders {    172.40.1.10;    };  #转发给 172.40.1.10
};
```

DNS 分离解析
– 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)
– 根据客户端的不同，解析同一个域名，得到的解析结果不同
– 目的：为客户端提供网络最近的服务器资源

```
view "lan" {
    match-clients {   192.168.4.207; } ;
    zone "sina.com" IN {
    ..... sina.com.lan;
    };
    192.168.4.100
};
view "other" {
     match-clients { any; } ;
     zone "sina.com"   IN   {
     .....   sina.com.other;
     };
};
```

DHCP 服务

自动分配入网参数

```
#vim etc/dhcp/dhcpd.conf
subnet   192.168.4.0   netmask 255.255.255.0 {
  range   192.168.4.100    192.168.4.200;
  option domain-name-servers 192.168.4.7;
  option routers 192.168.4.254;
  default-lease-time 600;
  max-lease-time 7200;
}
```

VPN(虚拟专用网络)

1、GRE 方式---------实现点到点的隧道通信（缺少加密机制）

2、PPTP（Point to Point Tunneling Protocol）------支持身份验证、数据加密、地址分配

3、L2TP+IPSec 方式-------支持身份认证、加密、分配地址

日志管理

常见的日志文件

```
/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

- -

网络服务

TCP/IP 的七层模型

应用层 (Application)：网络服务与最终用户的一个接口。

协议有：HTTP FTP TFTP SMTP SNMP DNS TELNET HTTPS POP3 DHCP

表示层（Presentation Layer）：数据的表示、安全、压缩。（在五层模型里面已经合并到了应用层）格式有，JPEG、ASCll、DECOIC、加密格式等

会话层（Session Layer）：建立、管理、终止会话。（在五层模型里面已经合并到了应用层）

对应主机进程，指本地主机与远程主机正在进行的会话

传输层 (Transport)：定义传输数据的协议端口号，以及流控和差错校验。　对应设备 - - > 防火墙

协议有：TCP UDP，数据包一旦离开网卡即进入网络传输层

TCP 的三次握手与四次断开

访问控制列表（ACL)

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

网络层 (Network)：进行逻辑地址寻址，实现不同网络之间的路径选择。　对应设备 - - > 路由器

协议有：ICMP IGMP IP（IPV4 IPV6） ARP RARP

路由器：跨越从源主机到目标主机的一个互联网络来转发数据包的过程

路由表：

1．直连路由：配置 IP 地址，端口 UP 状态，形成直连路由。

2．非直连路由：需要静态路由或动态路由，将网段添加到路由表中。

1 >静态路由,手工配置的，是单向的，因此需要在两个网络之间的边缘路由器上需要双方对指，否则就会造成流量有去无回，缺乏灵活性，适用于小型网络。

2 >缺省路由,在没有找到任何匹配的具体路由条目的情况下才使用的路由，适用于只有一个出口的末节网络（比如企业的网关出口），优先级最低，可以做为其他路由的补充。

3 >动态路由，由路由器通过路由协议自动设置, 减少了管理任务

OSPF 协议(开放式最短路径优先）

三层交换技术

使用三层交换技术实现 VLAN 间通信

NAT 网络地址转换

通过将内部网络的私有 IP 地址翻译成全球唯一的公网 IP 地址，使内部网络可以连接到互联网等外部网络上。

优点：

节省公有合法 IP 地址
处理地址重叠
安全性
NAT 的缺点
延迟增大
配置和维护的复杂性

NAT 实现方式
1）静态转换
IP 地址的对应关系是一对一，而且是不变的，借助静态转换，能实现外部网络对内部网络中某些特设定服务器的访问。并且没有节约公用 IP，只隐藏了主机的真实地址。
2）端口多路复用（PAT）
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 IP。

STP - Spanning Tree Protocol(生成树协议)
逻辑上断开环路，防止广播风暴的产生，当线路故障，阻塞接口被激活，恢复通信，起备份线路的作用．

热备份路由选择协议（HSRP）
作用：Cisco 私有协议 ，确保了当网络边缘设备或接入链路出现故障时，用户通信能迅速并透明地恢复，以此为 IP 网络提供冗余性。通过使用同一个虚拟 IP 地址和虚拟 MAC 地址，LAN 网段上的两台或者多台路由器可以作为一台虚拟路由器对外提供服务。HSRP 使组内的 cisco 路由器能互相监视对方的运行状态。（Cisco 私有协议）

数据链路层（Link）：建立逻辑连接、进行硬件地址寻址、差错校验等功能。 对应设备 - - > 交换机
（由底层网络定义协议）将比特组合成字节进而组合成帧，用 MAC 地址访问介质，错误发现但不能纠正
交换机是用来连接局域网的主要设备， 交换机分割冲突域，实现全双工通信
工作原理：学习，广播，转发，更新
交换机以太网接口双工模式
单工：两个数据站之间只能沿单一方向传输数据
半双工：两个数据站之间可以双向数据传输，但不能同时进行
全双工：两个数据站之间可双向且同时进行数据传输
冲突与广播域
广播域指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

虚拟局域网 VLAN
优势：广播控制、安全性、带宽利用、延迟
trunk 中继链接作用 :实现交换机之间的单一链路传递多个 vlan 的信息 ,可以承载多个 vlan
EthernetChannel（以太网通道）多条线路负载均衡，带宽提高容错，当一条线路失效时，其他线路通信，不会丢包
物理层（Physical Layer）：是计算机网络 OSI 模型中最低的一层． 对应设备 - - > 网卡
物理层规定:为传输数据所需要的物理链路创建、维持、拆除而提供具有机械的，电子的，功能的和规范的特性
为设备之间的数据通信提供传输媒体及互连设备，为数据传输提供可靠的环境，就是“信号和介质”
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
MAC 地址长度 48 位(6 个字节)，前 24 位代表厂商，后 24 位代表网卡编号
- -
web 服务
http，tomcat， nginx 对比
http 配置
配置文件/etc/httpd/conf/httpd.conf
– Listen: 监听地址:端口(80)
– ServerName: 本站点注册的 DNS 名称(空缺)
– DocumentRoot: 网页根目录(/var/www/html)
– DirectoryIndex: 起始页/首页文件名(index.html)
虚拟 Web 主机：
1.基于域名的虚拟 Web
2.基于端口的虚拟 Web
3.基于 IP 地址的虚拟 Web
<VirtualHost IP 地址:端口>
ServerName 此站点的 DNS 名称
DocumentRoot 此站点的网页根目录
</VirtualHost>
tomcat 配置
配置文件/usr/local/tomcat/conf/server.xml
部署虚拟主机
<Host name="www.aa.com" appBase="aa" unpackWARS="true" autoDeploy="true">
</Host>
Name:虚拟主机名称
appbase:定义基础目录，基础目录下默认项目 ROOT

<Context path="" docBase="base" reloadable="true"/>
DocBase 指定页面存储位置，默认 ROOT
Path 指定访问 URL 默认 index.html

<Context path="/test" docBase="/var/www/html/" />
跳转

配置 SSL 加密网站
1、创建私钥和证书
keytool -genkeypair -alias tomcat -keyalg RSA -keystore /usr/local/tomcat/keystore
//jdk 软件包安装生成工具
//提示输入密码为:123456, 交互界面信息输入完成输入是或者 y。
//-genkeypair 生成密钥对
//-alias tomcat 密钥别名
//-keyalg RSA 定义密钥算法为 RSA 算法
//-keystore 定义密钥文件存储在:/usr/local/tomcat/keystore
//y 或是 确认输入信息正确
2、修改配置文件 server.xml(取消注释添加密钥地址及密码即可)
<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" keystroePass="123456"
/>
//打开注释添加加密文件配置信息
虚拟主机设置独立日志文件
<Host name="www.bb.com" appBase="bb" unpackWARS="true" autoDeploy="true">
<Context path="" docBase="base" />
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
prefix=" bb_access" suffix=".txt"
pattern="%h %l %u %t "%r" %s %b" />
</Host>
nginx 配置
配置文件/usr/local/nginx/conf/nginx.conf
... ...
http{

```
    … …
    server{
    … …
        location{
        ……
        }
    }
}
```

1>用户认证

```
# htpasswd   /usr/local/nginx/pass    jerry          //追加用户，不使用-c 选项
server {
      … …
          auth_basic "Input Password:";                          //认证提示符
          auth_basic_user_file "/usr/local/nginx/pass";        //认证密码文件
          location / {
              … …
              }
       }
```

2>基于域名的虚拟主机

```
server {
      listen    80;                              //端口
      server_name    www.b.com;      //域名
    location / {
            … …
    }
}
```

3>基于端口的虚拟主机

```
          listen    80;                              //端口
```

4>基于 IP 的虚拟主机

```
          listen    192.168.4.254：80;                              //端口
```

5>基于加密网站的虚拟主机（源码安装使用--with-http_ssl_module）

```
#生成私钥与证书
cd /usr/local/nginx/conf
openssl genrsa > cert.key       //生成私钥
openssl refq -new -x509 -key cert.ket > cert.pem       //生成证书
#修改配置文件
vim    /usr/local/nginx/conf/nginx.conf     //原文件被注释，取消注释即可
    #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;
    #       }
    #}
```

6>配置支持 php 动态页面

```
 location    ~    \.php$    {
                root                 html;
                fastcgi_pass     127.0.0.1:9000;       #将请求转发给本机 9000 端口 ,PHP 解
释器
                fastcgi_index    index.php;
                #fastcgi_param                                          SCRIPT_FILENAME
$document_root$fastcgi_script_name;
                include    fastcgi.conf;          //模版错误，正确包含在 fastcgi.conf
           }
```

7>地址重写

rewrite 旧地址 新地址 [选项]

last 不再读其他 rewrite

break 不再读其他语句，结束请求

redirect 临时重定向

permanent 永久重定向

```
server {
              listen          80;
              server_name    localhost;
   location / {
```

```
    root     html;
  index    index.html index.htm;
  }
  if ($http_user_agent ~* firefox) {          //识别客户端 firefox 浏览器
  rewrite ^(.*)$         /firefox/$1;          //firefox 浏览器用户访问 firefox 目录下内容
  }
  }
```

8 >Nginx 反向代理--集群

```
http {
.. ..
upstream webserver {
            ip_hash;
           server 192.168.2.100:80 weight=1 max_fails=2 fail_timeout=10;
           server 192.168.2.200:80 weight=2 max_fails=2 fail_timeout=10;
           }
.. ..
server {
           listen        80;
           server_name    localhost;
           location / {
              proxy_pass http://webserver;
           }
}
//weight 设置服务器权重值,默认值为 1
//max_fails 设置最大失败次数
//fail_timeout 设置失败超时时间，单位为秒
// ip_hash  根据客户端 ip 分配固定的后端服务器
```

9>TCP/UDP 调度器

```
--with-stream  开启 ngx_stream_core_module 模块
配置在 http 外
stream {
                upstream backend {
                    server 192.168.2.100:22;                  //后端 SSH 服务器的 IP 和端口
                    server 192.168.2.200:22;
                }
                server {
                     listen 12345;                            //Nginx 监听的端口
                     proxy_connect_timeout 1s;
```

```
                proxy_timeout 3s;
                proxy_pass backend;
            }
}
```

连接测试 ssh 192.168.4.5 -p 12345

10>Nginx 优化

1>优化并发量

```
# ab -n 2000 -c 2000 http://192.168.4.5/          //模拟多并发
# vim /usr/local/nginx/conf/nginx.conf
.. ..
worker_processes    2;                  //与 CPU 核心数量一致
events {
worker_connections 65535;            //每个 worker 最大并发连接数
use epoll;
}
.. ..
```

修改配置文件后仍然无法高并发需要修改 linux 内核参数

```
 # ulimit -a                                  //查看所有属性值
 # ulimit -Hn 100000                          //设置硬限制（临时规则）
 # ulimit -Sn 100000                          //设置软限制（临时规则）
 # vim /etc/security/limits.conf              //永久
        .. ..
    *                soft    nofile              100000
    *                hard    nofile              100000
  //该配置文件分 4 列，分别如下：
  //用户或组    硬限制或软限制    需要限制的项目    限制的值
```

2>解决客户端访问头部信息过长的问题

```
# vim /usr/local/nginx/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;          //大请求包头部信息的缓存个数与容量
.. ..
}
```

3>客户端浏览器缓存数据

Firefox 地址栏内输入 about:cache 将显示 Firefox 浏览器的缓存信息

```
# vim /usr/local/nginx/conf/nginx.conf
server {
       ...
location ~* \.(jpg|jpeg|gif|png|css|js|ico|xml)$ {       //浏览器缓存图片等信息
expires              30d;                    //定义客户端缓存时间为 30 天
}
....
}
```

4>自定义报错页面

```
# vim /usr/local/nginx/conf/nginx.conf
 .. ..
 error_page     404    /40x.html;    //自定义错误页面
 .. ..
```

常见 http 状态码

200	一切正常
301	永久重定向
302	临时重定向
401	用户名或密码错误
403	禁止访问（客户端 ip 地址被拒绝）
404	文件不存在
414	请求 URL 头部过长
500	服务器内部错误
502	Bad Gateway

5>查看服务器状态信息

使用模块 --with-http_stub_status_module

```
# cat /usr/local/nginx/conf/nginx.conf
   … …
   location /status {
                    stub_status on;
                }
   … …
```

查看

```
#curl    http://192.168.4.5/status
```

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。

6>开启 gzip 压缩功能，提高数据传输效率

```
  # cat /usr/local/nginx/conf/nginx.conf
    http {
    .. ..
    gzip on;                                    //开启压缩
    gzip_min_length 1000;                       //小文件不压缩
    gzip_comp_level 4;                          //压缩比率
    gzip_types text/plain text/css application/json application/x-javascript text/xml application/xml application/xml+rss text/javascript;
                                                //对特定文件压缩，类型参考 mime.types
    .. ..
}
```

7>服务器内存缓存

```
    http {
            open_file_cache       max=2000    inactive=20s;
            open_file_cache_valid       60s;
            open_file_cache_min_uses 5;
            open_file_cache_errors      off;
    //设置服务器最大缓存 2000 个文件句柄，关闭 20 秒内无请求的文件句柄
    //文件句柄的有效时间是 60 秒，60 秒后过期
    //只有访问次数超过 5 次会被缓存
    }
```

netstat 命令可以查看系统中启动的端口信息，该命令常用选项如下：
-a 显示所有端口的信息
-n 以数字格式显示端口号
-t 显示 TCP 连接的端口
-u 显示 UDP 连接的端口
-l 显示服务正在监听的端口信息，如 httpd 启动后，会一直监听 80 端口
-p 显示监听端口的服务名称是什么（也就是程序名称）

Squid、Varinsh 和 Nginx 有什么区别，工作中你怎么选择？
Squid、Varinsh 和 Nginx 都是代理服务器

什么是代理服务器：
能当替用户去访问公网，并且能把访问到的数据缓存到服务器本地，等用户下次再访问相同的资源的时候，代理服务器直接从本地回应给用户，当本地没有的时候，我代替你去访问公网，我接收你的请求，我先在我自已的本地缓存找，如果我本地缓存有，我直接从我本地的缓存里回复你如果我在我本地没有找到你要访问的缓存的数据 ,那么代理服务器就会代替你去访问公网

区别：
1）Nginx 本来是反向代理/web 服务器，用了插件可以做做这个副业但是本身不支持特性挺多，只能缓存静态文件
2）从这些功能上。varnish 和 squid 是专业的 cache 服务，而 nginx 这些是第三方模块完成
3）varnish 本身的技术上优势要高于 squid，它采用了可视化页面缓存技术在内存的利用上，Varnish 比 Squid 具有优势，性能要比 Squid 高。还有强大的通过 Varnish 管理端口，可以使用正则表达式快速、批量地清除部分缓存它是内存缓存，速度一流，但是内存缓存也限制了其容量，缓存页面和图片一般是挺好的
4）squid 的优势在于完整的庞大的 cache 技术资料，和很多的应用生产环境

工作中选择：
要做 cache 服务的话 ,我们肯定是要选择专业的 cache 服务 ,优先选择 squid 或者 varnish

Tomcat 和 Resin 有什么区别，工作中你怎么选择？
区别：Tomcat 用户数多，可参考文档多，Resin 用户数少，可考虑文档少
最主要区别则是 Tomcat 是标准的 java 容器，不过性能方面比 resin 的要差一些
但稳定性和 java 程序的兼容性，应该是比 resin 的要好

工作中选择：现在大公司都是用 resin，追求性能；而中小型公司都是用 Tomcat，追求稳定和程序的兼容

什么是中间件？什么是 jdk？
中间件介绍：

中间件是一种独立的系统软件或服务程序 ,分布式应用软件借助这种软件在不同的技术之间共享资源
中间件位于客户机/ 服务器的操作系统之上，管理计算机资源和网络通讯
是连接两个独立应用程序或独立系统的软件。相连接的系统，即使它们具有不同的接口

但通过中间件相互之间仍能交换信息。执行中间件的一个关键途径是信息传递
通过中间件，应用程序可以工作于多平台或 OS 环境。

jdk：jdk 是 Java 的开发工具包
它是一种用于构建在 Java 平台上发布的应用程序、applet 和组件的开发环境

Tomcat8005、8009、8080 三个端口的含义？
8005==》 关闭时使用
8009==》 为 AJP 端口，即容器使用，如 Apache 能通过 AJP 协议访问 Tomcat 的 8009 端口
8080==》 一般应用使用

什么叫 CDN？
- 即内容分发网络
- 其目的是通过在现有的 Internet 中增加一层新的网络架构，将网站的内容发布到最接近用户的网络边缘，使用户可就近取得所需的内容，提高用户访问网站的速度

什么叫网站灰度发布？
灰度发布是指在黑与白之间，能够平滑过渡的一种发布方式
AB test 就是一种灰度发布方式，让一部用户继续用 A，一部分用户开始用 B
如果用户对 B 没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到 B 上面 来
灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度

简述 DNS 进行域名解析的过程？
用户要访问 www.baidu.com，会先找本机的 host 文件，再找本地设置的 DNS 服务器，如果也没有的话，就去网络中找根服务器，根服务器反馈结果，说只能提供一级域名服务器.cn，就去找一级域名服务器，一级域名服务器说只能提供二级域名服务器.com.cn,就去找二级域名服务器，二级域服务器只能提供三级域名服务器.baidu.com.cn，就去找三级域名服务器，三级域名服务器正好有这个网站 www.baidu.com，然后发给请求的服务器，保存一份之后，再发给客户端

= - =

集群

一组通过高速网络互联的计算组。将很多服务器集中起来，提供同一种服务，在客户端看起来就像只有一个服务器。可以在付出较低成本的情况下获得在性能、可靠性、灵活性方面的相对较高的收益。

分类：高可用集群 负载均衡集群 高性能计算集群

高可用：在服务出现故障时，集群系统可以自动将服务从故障节点切换到另一个备用节点，从而提供不间断性服务，保证了业务的持续运行。常用的高可用开源软件包括 keepalived，hertbeat。

keepalived 的工作原理：

1、先介绍 VRRP

keepalived 高可用对之间通过 VRRP 通信的。VRRP，全称 Virtual Router Redundancy Protocal，虚拟路由冗余协议，VRRP 的出现是为了解决静态路由的单点故障。VRRP 用 IP 多播的方式实现高可用对之间的通信，工作时主节点发包，备节点收包，当备节点接收不到主节点发的数据时，启动接管程序，接管主节点的资源。被节点可以有多个，通过优先级竞选。

2、再介绍 Keepalived 工作原理

keepalived 高可用对之间通过 VRRP 进行通信的，VRRP 是通过竞选机制来确定主备的，主的优先级高于备，因此工作时主会优先获得所有的资源，备节点处于等待状态，当主挂了的时候，备节点就会接管主节点的资源，然后顶替主节点对外提供服务。

在 keepalived 服务对之间，只有作为主的服务器会一直发送 VRRP 广播包，告诉备它还活着，此时备不会抢占主，当主不可用时，即备监听不到主发送的广播包时，就会启动相关服务接管资源，保证业务的连续性。

负载均衡 :通常有两台或两台以上的服务器组成，分为前段负载调度，和后端节点两个部分。调度器获取客户端的请求，并通过自身定义的负载分担策略，将请求平均分配到后端各个服务节点，每个节点都承担一定的访问请求压力。

lvs 负载均衡的模式

三种部署方式：VS/NAT（nat 模式） VS/DR(路由模式) VS/TUN（隧道模式）。

一、NAT 模式（VS-NAT）

原理：就是把客户端发来的数据包的 IP 头的目的地址，在负载均衡器上换成其中一台 RS 的 IP 地址并发至此 RS 来处理,RS 处理完后把数据交给负载均衡器,负载均衡器再把数据包原 IP 地址改为自己的 IP 将目的地址改为客户端 IP 地址即可期间,无论是进来的流量,还是出去的流量,都必须经过负载均衡器

优点：集群中的物理服务器可以使用任何支持 TCP/IP 操作系统，只有负载均衡器需要一个合法的 IP 地址

缺点：扩展性有限。当服务器节点（普通 PC 服务器）增长过多时,负载均衡器将成为整个系统的瓶颈因为所有的请求包和应答包的流向都经过负载均衡器。当服务器节点过多时大量的数据包都交汇在负载均衡器那，速度就会变慢！

二、IP 隧道模式（VS-TUN）

原理：首先要知道，互联网上的大多 Internet 服务的请求包很短小，而应答包通常很大那么隧道模式就是，把客户端发来的数据包，封装一个新的 IP 头标记(仅目的 IP)发给 RS。RS 收到后,先把数据包的头解开,还原数据包,处理后,直接返回给客户端,不需要再经过负载均衡器。注意,由于 RS 需要对负载均衡器发过来的数据包进行还原,所以说必须支持 IPTUNNEL 协议，所以,在 RS 的内核中,必须编译支持 IPTUNNEL 这个选项。
优点：负载均衡器只负责将请求包分发给后端节点服务器，而 RS 将应答包直接发给用户所以，减少了负载均衡器的大量数据流动，负载均衡器不再是系统的瓶颈，就能处理很巨大的请求量这种方式，一台负载均衡器能够为很多 RS 进行分发。而且跑在公网上就能进行不同地域的分发。
缺点：隧道模式的 RS 节点需要合法 IP，这种方式需要所有的服务器支持”IP Tunneling”(IP Encapsulation)协议，服务器可能只局限在部分 Linux 系统上
三、直接路由模式（VS-DR）
原理 :负载均衡器和 RS 都使用同一个 IP 对外服务但只有 DR 对 ARP 请求进行响应所有 RS 对本身这个 IP 的 ARP 请求保持静默也就是说,网关会把对这个服务 IP 的请求全部定向给 DR 而 DR 收到数据包后根据调度算法,找出对应的 RS,把目的 MAC 地址改为 RS 的 MAC(因为 IP 一致）并将请求分发给这台 RS 这时 RS 收到这个数据包,处理完成之后，由于 IP 一致，可以直接将数据返给客户则等于直接从客户端收到这个数据包无异,处理后直接返回给客户端由于负载均衡器要对二层包头进行改换,所以负载均衡器和 RS 之间必须在一个广播域也可以简单的理解为在同一台交换机上
优点：和 TUN（隧道模式）一样，负载均衡器也只是分发请求，应答包通过单独的路由方法返回给客户端与 VS-TUN 相比，VS-DR 这种实现方式不需要隧道结构，因此可以使用大多数操作系统做为物理服务器。
缺点：（不能说缺点，只能说是不足）要求负载均衡器的网卡必须与物理网卡在一个物理段上。
LVS、Nginx、HAproxy 有什么区别？工作中你怎么选择？
LVS： 是基于四层的转发
HAproxy： 是基于四层和七层的转发，是专业的代理服务器
Nginx： 是 WEB 服务器，缓存服务器，又是反向代理服务器，可以做七层的转发

区别： LVS 由于是基于四层的转发所以只能做端口的转发
而基于 URL 的、基于目录的这种转发 LVS 就做不了

工作选择：
HAproxy 和 Nginx 由于可以做七层的转发，所以 URL 和目录的转发都可以做
在很大并发量的时候我们就要选择 LVS，像中小型公司的话并发量没那么大
选择 HAproxy 或者 Nginx 足已，由于 HAproxy 由是专业的代理服务器
配置简单，所以中小型企业推荐使用 HAproxy

lvs/nginx/haproxy 优缺点

nginx 分析

优点：

工作在 7 层，可以针对 http 做分流策～

正则表达式比 HAProxy 强大

安装、配置、测试间大，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

LVS 分析

优点

负载能力强，工作在 4 层，对内存、CPU 消耗低

配置性低，没有大多可配置性，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

HAProxy 分析

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，高于 nginx、低于 LVS

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

Nginx 的优点是：

1、工作在网络的 7 层之上，可以针对 http 应用做一些分流的策略，比如针对域名、目录结构它的正则规则比 HAProxy 更为强大和灵活，这也是它目前广泛流行的主要原因之一 Nginx 单凭这点可利用的场合就远多于 LVS 了。

2、Nginx 对网络稳定性的依赖非常小，理论上能 ping 通就就能进行负载功能，这个也是

它的优势之一相反 LVS 对网络稳定性依赖比较大，这点本人深有体会；
3、Nginx 安装和配置比较简单，测试起来比较方便，它基本能把错误用日志打印出来 LVS 的配置、测试就要花比较长的时间了，LVS 对网络依赖比较大。
4、可以承担高负载压力且稳定，在硬件不差的情况下一般能支撑几万次的并发量，负载度比 LVS 相对小些。
5、Nginx 可以通过端口检测到服务器内部的故障，比如根据服务器处理网页返回的状态码、超时等等，并且会把返回错误的请求重新提交到另一个节点，不过其中缺点就是不支持 url 来检测。比如用户正在上传一个文件，而处理该上传的节点刚好在上传过程中出现故障，Nginx 会把上传切到另一台服务器重新处理，而 LVS 就直接断掉了如果是上传一个很大的文件或者很重要的文件的话，用户可能会因此而不满。
6、Nginx 不仅仅是一款优秀的负载均衡器/反向代理软件，它同时也是功能强大的 Web 应用服务器 LNMP 也是近几年非常流行的 web 架构，在高流量的环境中稳定性也很好。
7、Nginx 现在作为 Web 反向加速缓存越来越成熟了，速度比传统的 Squid 服务器更快，可考虑用其作为反向代理加速器
8、Nginx 可作为中层反向代理使用，这一层面 Nginx 基本上无对手，唯一可以对比 Nginx 的就只有 lighttpd 了不过 lighttpd 目前还没有做到 Nginx 完全的功能，配置也不那么清晰易读，社区资料也远远没 Nginx 活跃
9、Nginx 也可作为静态网页和图片服务器，这方面的性能也无对手。还有 Nginx 社区非常活跃，第三方模块也很多

Nginx 的缺点是：
1、Nginx 仅能支持 http、https 和 Email 协议，这样就在适用范围上面小些，这个是它的缺点
2、对后端服务器的健康检查，只支持通过端口来检测，不支持通过 url 来检测　　不支持 Session 的直接保持，但能通过 ip_hash 来解决

LVS：使用 Linux 内核集群实现一个高性能、高可用的负载均衡服务器它具有很好的可伸缩性（Scalability)、可靠性（Reliability)和可管理性（Manageability)

LVS 的优点是：
1、抗负载能力强、是工作在网络 4 层之上仅作分发之用，没有流量的产生这个特点也决定了它在负载均衡软件里的性能最强的，对内存和 cpu 资源消耗比较低

2、配置性比较低，这是一个缺点也是一个优点，因为没有可太多配置的东西所以并不需要太多接触，大大减少了人为出错的几率
3、工作稳定，因为其本身抗负载能力很强，自身有完整的双机热备方案如 LVS+Keepalived，不过我们在项目实施中用得最多的还是 LVS/DR+Keepalived
4、无流量，LVS 只分发请求，而流量并不从它本身出去，这点保证了均衡器 IO 的性能不

会收到大流量的影响。
5、应用范围较广，因为 LVS 工作在 4 层，所以它几乎可对所有应用做负载均衡，包括 http、数据库、在线聊天室等

LVS 的缺点是：
1、软件本身不支持正则表达式处理，不能做动静分离而现在许多网站在这方面都有较强的需求，这个是 Nginx/HAProxy+Keepalived 的优势所在
2、如果是网站应用比较庞大的话，LVS/DR+Keepalived 实施起来就比较复杂了特别后面有 Windows Server 的机器的话，如果实施及配置还有维护过程就比较复杂了相对而言，Nginx/HAProxy+Keepalived 就简单多了。

HAProxy 的特点是：
1、HAProxy 也是支持虚拟主机的。
2、HAProxy 的优点能够补充 Nginx 的一些缺点，比如支持 Session 的保持，Cookie 的引导同时支持通过获取指定的 url 来检测后端服务器的状态
3、HAProxy 跟 LVS 类似，本身就只是一款负载均衡软件单纯从效率上来讲 HAProxy 会比 Nginx 有更出色的负载均衡速度，在并发处理上也是优于 Nginx 的
4、HAProxy 支持 TCP 协议的负载均衡转发，可以对 MySQL 读进行负载均衡对后端的 MySQL 节点进行检测和负载均衡，大家可以用 LVS+Keepalived 对 MySQL 主从做负载均衡
5、HAProxy 负载均衡策略非常多，HAProxy 的负载均衡算法现在具体有如下 8 种：
①roundrobin，表示简单的轮询，这个不多说，这个是负载均衡基本都具备的；
② static-rr，表示根据权重，建议关注；
③leastconn，表示最少连接者先处理，建议关注；
④ source，表示根据请求源 IP，这个跟 Nginx 的 IP_hash 机制类似我们用其作为解决 session 问题的一种方法，建议关注；
⑤ri，表示根据请求的 URI；
⑥rl_param ,表示根据请求的 URI 参数’ balance url_param’ requires an URL parameter name；
⑦hdr(name)，表示根据 HTTP 请求头来锁定每一次 HTTP 请求；
⑧rdp-cookie(name)，表示根据据 cookie(name)来锁定并哈希每一次 TCP 请求。
- = -

linux 安全

基本防护

1．账户安全

chage -E 20180920 zhangsan （change age）

passwd -l zhangsan

修改 tty 登录提示信息，隐藏系统版本修改/etc/issue、/etc/issue.net（远程）

2.文件安全

Chattr -i /tmp/aaa.txt (change attribute)

sudo 使用

#sudo [-u 目标用户] 特权命令

配置 sudo 授权

visudo 或 vim /etc/sudoers

用户名或%组名 目标身份（省去时表示 root） 执行权限可以使用通配符*、！符号取反

softadm ALL=(ALL) /usr/bin/systemctl //授权 softadm 用户以 root 身份执行 systemctl 命令

SSH 访问控制

SSH 基本防护

配置安全策略

#vim /etc/ssh/sshd_config

.. ..

Protocol 2 //去掉 SSH 协议 V1

PermitRootLogin no //禁止 root 用户登录

PermitEmptyPasswords no //禁止密码为空的用户登录

UseDNS no //不解析客户机地址,反向解析

LoginGraceTime 1m //登录限时

MaxAuthTries 3 //每连接最多认证次数

PasswordAuthentication no //禁止密码登陆 ,可密钥登陆

.. ..

AllowUsers zengye john useradm@192.168.4.0/24 //定义账户白名单

##DenyUsers USER1 USER2 //定义账户黑名单

##DenyGroups GROUP1 GROUP2 //定义组黑名单

##AllowGroups GROUP1 GROUP2 //定义组白名单

SSH 密钥验证登陆

ssh-keygen //生成密钥 id_rsa、id_rsa.pub

ssh-copy-id root@192.168.4.5 //部署密钥

tail -2 /root/.ssh/authorized_keys //确认部署结果

selinux 策略

配置文件 /etc/selinux/config

SELINUX=permissive，enforcing，disabled

SELINUXTYPE=targeted，minimum，mls

安全上下文（Security Context）

ls -Z 文件名

常见访问类型：

bin_t	二进制执行文件
etc_t	系统配置文件
fsadm_exec_t	文件系统管理
admin_home_t	管理员账户的宿主目录
user_home_t	普通用户的宿主目录
httpd_sys_content_t	http 网站内容

修改安全上下文

chcon [选项]... 环境 文件...

或: chcon [选项]... [-u 用户] [-r 角色] [-l 范围] [-t 类型] 文件...

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

selinux 布尔值

getsebool / setsebool （0|1）

-a 列出所有布尔值

-P 永久更改，重启后仍然有效

加密与解密

对称加密:加密/解密用同一个密钥（DES、AES）

非对称加密:加密/解密用不同的密钥（RSA、DSA）

GPG 签名

AIDE 入侵检测系统

检查数据文件的权限、时间、大小、哈希值等、检验数据的完整性。需要在数据破坏之前，对数据完 成初始化校验，生成校验数据库文件，检查时进行对比检验文件。

扫描与抓包

#tcpdump -A host 192.168.4.100 and tcp port 21

audit 监控文件

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，但她会发现并记录违反安全策略的人及其对应的行为。

Iptables 基本管理

IPTABLES 是与最新的 3.5 版本 Linux 内核集成的 IP 信息包过滤系统。如果 Linux 系统

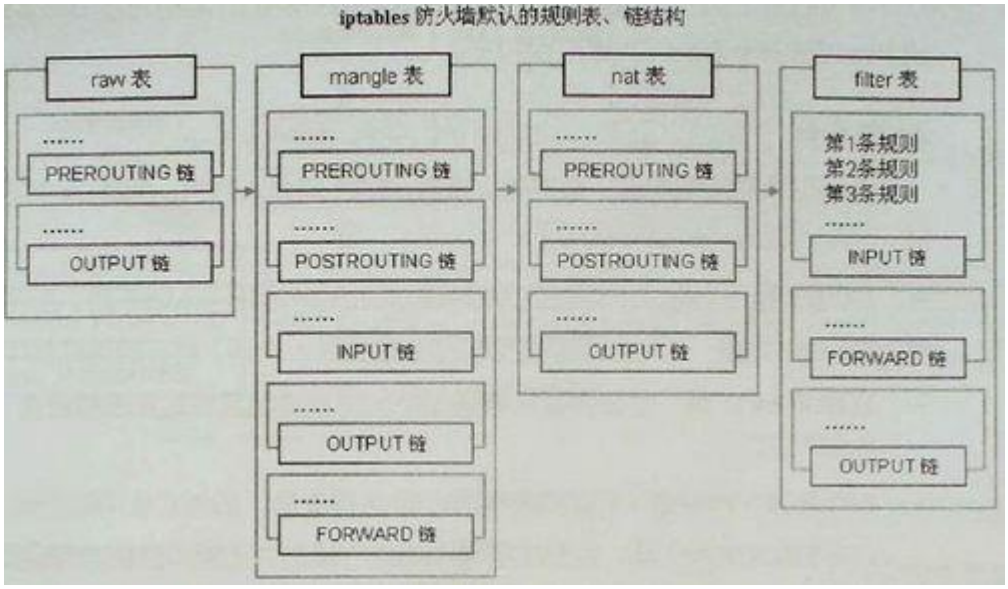
连接到因特网或LAN、服务器或连接LAN和因特网的代理服务器，则该系统有利于在 Linux 系统上更好地控制 IP 信息包过滤和防火墙配置。

yum -y install iptables-services

systemctl start iptables.service

//redhat7 默认使用 firewalld 防火墙，但底层仍然调用 iptables

//规则链内顺序比对，匹配即停止原则。若无任何匹配，则按默认策略处理



iptables 框架

1）iptables 的 4 个表（区分大小写）：

iptables 默认有 4 个表，nat 表（地址转换表）、filter 表（数据过滤表）、raw 表（状态跟踪表）、mangle 表（包标记表）。

2）iptables 的 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（路有前规则）

POSTROUTING 链（路由后规则）

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

常用选项：

添加规则 -A 追加一条防火墙规则至链的末尾位置

-I 插入一条防火墙规则至链的开头，命令追加数字插入到指定位置
查看规则 -L 查看 iptables 所有规则
-n 以数字形式显示地址、端口等信息
--line-numbers 查看规则时，显示规则的行号
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
默认规则 -P 为指定的链设置默认规则

语法结构：
#iptables [-t 表名] 选项 [链名] [条件] [-j 目标操作]
注意事项与规律：
//可以不指定表，默认为 filter 表
//可以不指定链，默认为对应表的所有链
//除非设置默认策略，否则必须指定匹配条件
//选项/链名/目标操作用大写字母，其余都小写
iptables -t filter -A INPUT -p tcp -j ACCEPT//追加规则至 filter 表中的 INPUT 链的末尾
iptables -I INPUT 2 -p icmp -j ACCEPT //插入规则 filter 表 INPUT 链第 2 行
//icmp 为 ping 协议
iptables -nL INPUT //仅查看 INPUT 链的规则
iptables -L INPUT --line-numbers //查看规则，显示行号
iptables -D INPUT 3 //删除 filter 表中 INPUT 链的第 3 条规则
iptables -F //清空 filter 表中所有链的防火墙规则
iptables -t filter -P INPUT DROP //设置默认规则，默认为 ACCEPT
iptables 过滤条件
通用匹配 协议匹配 -p 协议名称
地址匹配 -s 源地址、-d 目标地址
接口匹配 -i 接受数据的网卡、-o 发送数据的网卡
隐含匹配 端口匹配 --sport 源端口号、--dport 目标端口号
ICMP 类型匹配 --icmp-type ICMP 类型
//需要取反时，使用！

linux 路由转发功能
临时：echo (0|1) > /proc/sys/net/ipv4/ip_forward
永久：echo ‘net.ipv4.ip_forward=1’ >> /etc/sysctl.conf

iptables -A INPUT -p icmp --icmp-type echo-request -j DROP
//仅禁止入站的 ping 请求，不拒绝入站的 ping 回应包

//iptables -p icmp --help 查看帮助信息
iptables 扩展规则
iptables 选项 链名称 -m 扩展模块 --具体扩展条件 -j 动作 //-m 开启扩展功能
iptables -A INPUT -p tcp --dport 22\
> -m mac --mac-source 52:54:00:00:00:0b -j DROP
//拒绝 52:54:00:00:00:0b 这台主机远程本机
iptables -A INPUT -p tcp \
> -m multiport --dports 20:22,25,80,110,143,16501:16800 -j ACCEPT
//一次性开启 20,21,22,25,80,110,143,16501 到 16800 所有的端口
iptables -A INPUT -p tcp --dport 22 \
> -m iprange --src-range 192.168.4.10-192.168.4.20 -j ACCEPT
//允许从 192.168.4.10-192.168.4.20 登录

实现 IP 地址的伪装（SNAT 源地址转换）
echo 1 > /proc/sys/net/ipv4/ip_forward //开启路由转发
iptables -t nat -A POSTROUTING -s 192.168.4.0/24 –p tcp \
> --dport 80 -j SNAT --to-source 192.168.2.5 //实现 SNAT 地址转换

防火墙安全策略
区域划分
– public:仅允许访问本机的 sshd、DHCP、ping 少数几个服务
– trusted:允许任何访问
– block:阻塞任何来访请求，明确拒绝
– drop:丢弃任何来访的数据包，节省资源
_*_
*_*_*_*
数据库
MySQL 的复制原理以及流程
3 个线程以及之间的关联；
（1）主：binlog 线程——记录下所有改变了数据库数据的语句，放进 master 上的 binlog 中；
（2）从：io 线程——在使用 start slave 之后，负责从 master 上拉取 binlog 内容，放进自己的 relay log 中；
（3）从：sql 执行线程——执行 relay log 中的语句；

Mysql 主从复制原理过程详细描述
1、在 Slave 服务器上执行 start slave 命令开启主从复制开关，开始进行主从同步。
2、此时，Slave 服务器 I/O 线程为通过 Master 上已经授权的复制用户权限请求连接 Master

服务器，并请求从指定 binlog 日志文件的指定位置（日志文件名和位置就是在配置主从复制服务时执行 change master 命令指定的）之后开始发送 binlog 日志内容。

3、Master 服务器接收到来自 Slave 服务器的 I/O 线程请求后，其上负责复制的 I/O 线程会根据 Slave 服务器的 I/O 线程请求的信息分批读取指定 binlog 日志文件指定位置之后的 binlog 日志信息 ,然后返回给 Slave 端的 I/O 线程。返回的信息中除了 binlog 日志内容外，还有在 Master 服务器端记录的新的 binlog 文件名称 ,以及在新的 binlog 中的下一个执行更新位置。

4、当 Slave 服务器的 I/O 线程获取到 Master 服务器上 I/O 线程发送的日志内容，日志文件及位置点后，会将 binlog 日志内容依次写到 Slave 端自身 Relay Log（中级日志）文件（MySQL-relay-bin.xxxx）的最末端，并将新的 binlog 文件名和位置记录到 master-info 文件中，以便下一次读取 Master 端新 binlog 日志时能够告诉 Master 服务器从新 binlog 日志的指定文件及位置开始请求新的 binlog 日志文件。

5、Slave 服务器端的 SQL 线程会实时检测本地 Relay Log 中 I/O 线程新增加的日志内容，然后及时地把 Relay Log 文件中内容解析成 SQL 语句 ,并在自身 Slave 服务器上按解析 SQL 语句的位置顺序执行应用这些 SQL 语句，并在 relay-log.info 中记录当前应用中继日志的文件名及位置点。　　经过上面的过程，就可以确保在 Master 端和 Slave 端执行了同样的 SQL 语句。当复制状态正常时，Master 端和 Slave 端的数据是完全一样的。

常用的 MYSQL 调优策略

1、硬件层相关优化

　　修改服务器 BIOS 设置

　　选择 Performance Per Watt Optimized(DAPC)模式，发挥 CPU 最大性能。

　　Memory Frequency（内存频率）选择 Maximum Performance（最佳性能）

　　内存设置菜单中，启用 Node Interleaving，避免 NUMA 问题

2、磁盘 I/O 相关

　　使用 SSD 硬盘

　　如果是磁盘阵列存储，建议阵列卡同时配备 CACHE 及 BBU 模块，可明显提升 IOPS。

　　raid 级别尽量选择 raid10，而不是 raid5.

3、文件系统层优化

　　使用 deadline/noop 这两种 I/O 调度器，千万别用 cfq

　　使用 xfs 文件系统，千万别用 ext3；ext4 勉强可用，但业务量很大的话，则一定要用 xfs；

　　文件系统 mount 参数中增加：noatime, nodiratime, nobarrier 几个选项（nobarrier 是 xfs 文件系统特有的）；

4、内核参数优化

　　修改 vm.swappiness 参数 ,降低 swap 使用率。RHEL7/centos7 以上则慎重设置为 0，可能发生 OOM

　　调整 vm.dirty_background_ratio、vm.dirty_ratio 内核参数，以确保能持续将脏数据

刷新到磁盘，避免瞬间 I/O 写。产生等待。

调整 net.ipv4.tcp_tw_recycle、net.ipv4.tcp_tw_reuse 都设置为 1 ,减少 TIME_WAIT，提高 TCP 效率。

5、Mysql 参数优化建议

建议设置 default-storage-engine=InnoDB，强烈建议不要再使用 MyISAM 引擎。

调整 innodb_buffer_pool_size 的大小，如果是单实例且绝大多数是 InnoDB 引擎表的话，可考虑设置为物理内存的 50% -70%左右。

设置 innodb_file_per_table = 1，使用独立表空间。

调整 innodb_data_file_path = ibdata1:1G:autoextend，不要用默认的 10M,在高并发场景下，性能会有很大提升。

设置 innodb_log_file_size=256M，设置 innodb_log_files_in_group=2，基本可以满足大多数应用场景。

调整 max_connection（最大连接数）、max_connection_error（最大错误数）设置，根据业务量大小进行设置。

另外，open_files_limit 、 innodb_open_files 、 table_open_cache 、table_definition_cache 可以设置大约为 max_connection 的 10 倍左右大小。

key_buffer_size 建议调小，32M 左右即可，另外建议关闭 query cache。

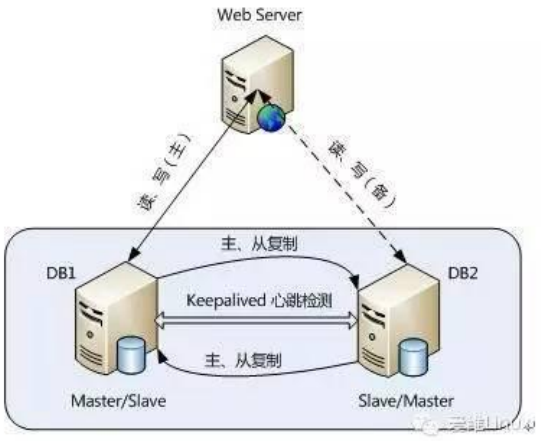
mp_table_size 和 max_heap_table_size 设置不要过大，另外 sort_buffer_size、join_buffer_size、read_buffer_size、read_rnd_buffer_size 等设置也不要过大。

MYSQL 常见的应用架构分享

1、主从复制解决方案

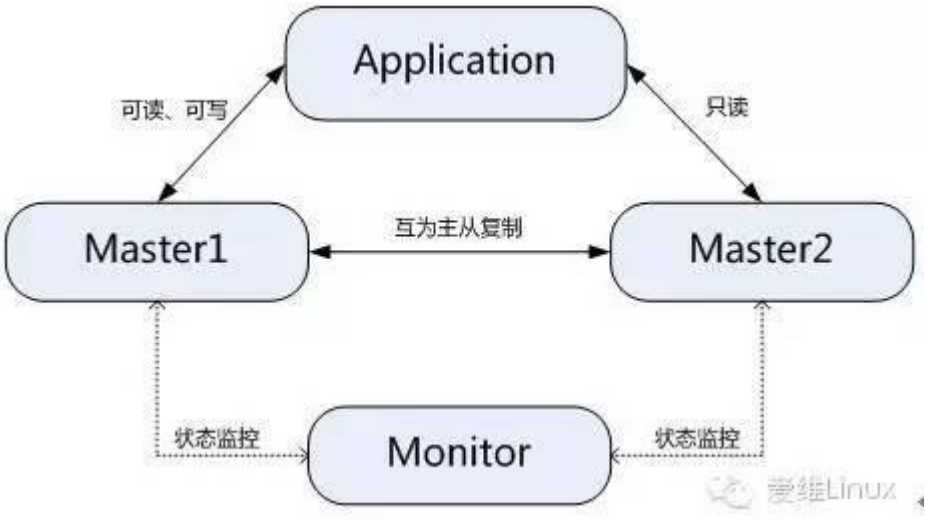
这是 MySQL 自身提供的一种高可用解决方案，数据同步方法采用的是 MySQL replication 技术。MySQL replication 就是从服务器到主服务器拉取二进制日志文件，然后再将日志文件解析成相应的 SQL 在从服务器上重新执行一遍主服务器的操作，通过这种方式保证数据的一致性。

为了达到更高的可用性，在实际的应用环境中，一般都是采用 MySQL replication 技术配合高可用集群软件 keepalived 来实现自动 failover，这种方式可以实现 95.000%的 SLA。



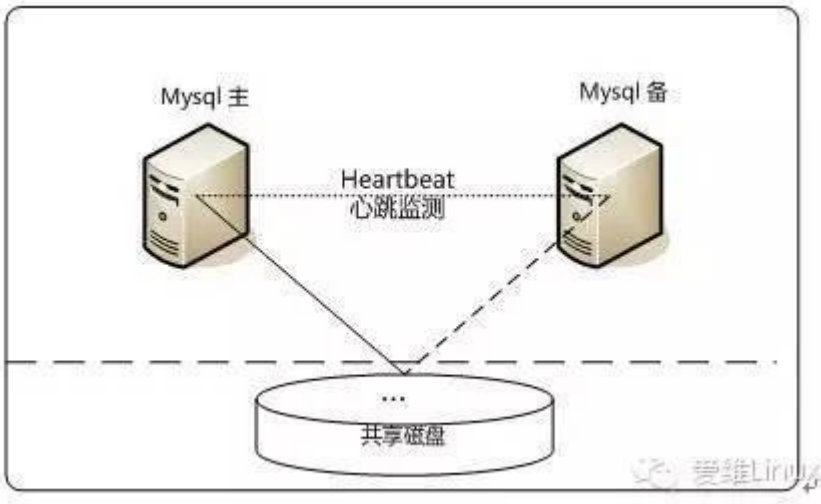
2、MMM/MHA 高可用解决方案

MMM 提供了 MySQL 主主复制配置的监控、故障转移和管理的一套可伸缩的脚本套件。在 MMM 高可用方案中，典型的应用是双主多从架构，通过 MySQL replication 技术可以实现两个服务器互为主从，且在任何时候只有一个节点可以被写入，避免了多点写入的数据冲突。同时，当可写的主节点故障时，MMM 套件可以立刻监控到，然后将服务自动切换到另一个主节点，继续提供服务，从而实现 MySQL 的高可用。



3、Heartbeat/SAN 高可用解决方案

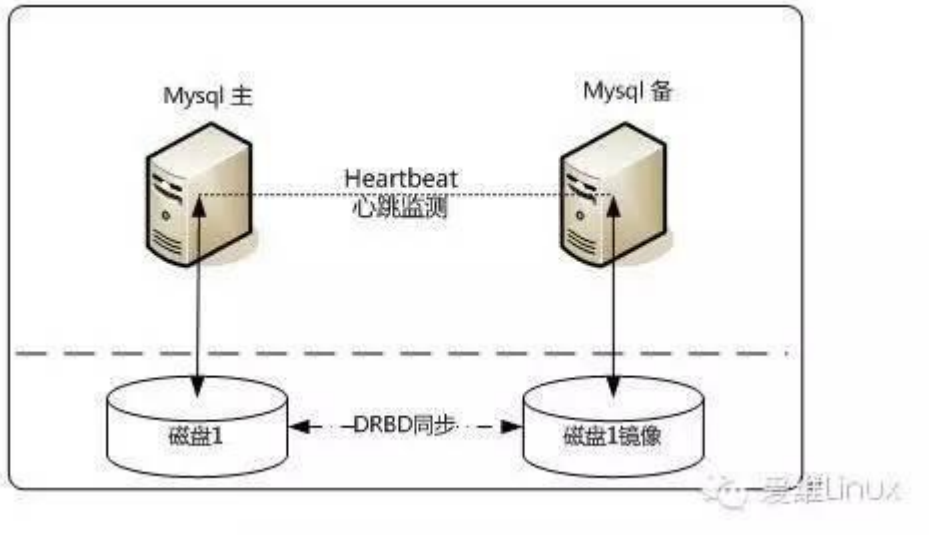
在这个方案中，处理 failover 的方式是高可用集群软件 Heartbeat，它监控和管理各个节点间连接的网络，并监控集群服务，当节点出现故障或者服务不可用时，自动在其他节点启动集群服务。在数据共享方面，通过 SAN（Storage Area Network）存储来共享数据，这种方案可以实现 99.990%的 SLA。



4、Heartbeat/DRBD 高可用解决方案

此方案处理 failover 的方式上依旧采用 Heartbeat，不同的是，在数据共享方面，采用了基于块级别的数据同步软件 DRBD 来实现。

DRBD 是一个用软件实现的、无共享的、服务器之间镜像块设备内容的存储复制解决方案。和 SAN 网络不同，它并不共享存储，而是通过服务器之间的网络复制数据。



MySQL 中 myisam 与 innodb 的区别，至少 5 点

(1)、问 5 点不同：

1>.InnoDB 支持事物，而 MyISAM 不支持事物

2>.InnoDB 支持行级锁，而 MyISAM 支持表级锁

3>.InnoDB 支持 MVCC，而 MyISAM 不支持

4>.InnoDB 支持外键，而 MyISAM 不支持

5>.InnoDB 不支持全文索引，而 MyISAM 支持。

(2)、innodb 引擎的 4 大特性：

插入缓冲（insert buffer)；

二次写(double write)；

自适应哈希索引(ahi)；

预读(read ahead)。

(3)、2 者 selectcount(*)哪个更快，为什么

myisam 更快，因为 myisam 内部维护了一个计数器，可以直接调取。

mysql 的 innodb 如何定位锁问题，mysql 如何减少主从复制延迟？

mysql 的 innodb 如何定位锁问题:

在使用 show engine innodb status 检查引擎状态时，发现了死锁问题

在 5.5 中，information_schema 库中增加了三个关于锁的表（MEMORY 引擎）

innodb_trx ## 当前运行的所有事务

innodb_locks ## 当前出现的锁

innodb_lock_waits ## 锁等待的对应关系

mysql 如何减少主从复制延迟:

如果延迟比较大，就先确认以下几个因素：
1. 从库硬件比主库差，导致复制延迟
2. 主从复制单线程，如果主库写并发太大，来不及传送到从库就会导致延迟。更高版本的 mysql 可以支持多线程复制
3. 慢 SQL 语句过多
4. 网络延迟
5. master 负载
主库读写压力大，导致复制延迟，架构的前端要加 buffer 及缓存层
6. slave 负载
一般的做法是，使用多台 slave 来分摊读请求，再从这些 slave 中取一台专用的服务器只作为备份用，不进行其他任何操作.另外， 2 个可以减少延迟的参数:
–slave-net-timeout=seconds 单位为秒 默认设置为 3600 秒

#参数含义：当 slave 从主数据库读取 log 数据失败后，等待多久重新建立连接并获取数据
–master-connect-retry=seconds 单位为秒 默认设置为 60 秒

#参数含义：当重新建立主从连接时，如果连接建立失败，间隔多久后重试
通常配置以上 2 个参数可以减少网络问题导致的主从数据同步延迟

MySQL 数据库主从同步延迟解决方案
最简单的减少 slave 同步延时的方案就是在架构上做优化，尽量让主库的 DDL 快速执行
还有就是主库是写，对数据安全性较高，比如 sync_binlog=1，innodb_flush_log_at_trx_commit= 1 之类的设置，而 slave 则不需要这么高的数据安全，完全可以讲 sync_binlog 设置为 0 或者关闭 binlog innodb_flushlog 也可以设置为 0 来提高 sql 的执行效率。另外就是使用比主库更好的硬件设备作为 slave

MYSQL 经典应用架构
数据库索引
MYSQL 几种日志类型
重做日志（redo log）
确保事务的持久性，防止在发生故障的时间点，尚有脏页未写入磁盘。在重启 MySQL 服务的时候，根据 redo log 进行重做，从而达到事务的持久性这一特性。
回滚日志（undo log）
保存了事务发生之前的数据的一个版本，可以用于回滚，同时可以提供多版本并发控制

下的读（MVCC），也即非锁定读。

二进制日志（binlog）

用于复制，在主从复制中，从库利用主库上的 binlog 进行重播，实现主从同步；用于数据库基于时间点的还原。

错误日志（errorlog）

慢查询日志（slow query log）

一般查询日志（general log）

中继日志（relay log）

innodb 的事务与日志的实现方式

(1)、有多少种日志：

错误日志：记录出错信息，也记录一些警告信息或者正确的信息。

查询日志：记录所有对数据库请求的信息，不论这些请求是否得到了正确的执行。

慢查询日志：设置一个阈值，将运行时间超过该值的所有 SQL 语句都记录到慢查询的日志文件中。

二进制日志：记录对数据库执行更改的所有操作。

中继日志。

事务日志。

(2)、事物的 4 种隔离级别

隔离级别

读未提交(RU)

读已提交(RC)

可重复读(RR)

串行

(3)、事务是如何通过日志来实现的，说得越深入越好。

事务日志是通过 redo 和 innodb 的存储引擎日志缓冲（Innodb log buffer）来实现的，当开始一个事务的时候，会记录该事务的 lsn(log sequence number)号; 当事务执行时，会往 InnoDB 存储引擎的日志。

的日志缓存里面插入事务日志；当事务提交时，必须将存储引擎的日志缓冲写入磁盘（通过 innodb_flush_log_at_trx_commit 来控制），也就是写数据前，需要先写日志。这种方式称为“预写日志方式”。

问了下 MySQL 数据库 cpu 飙升到 500%的话他怎么处理？

(1)、没有经验的，可以不问；

(2)、有经验的，问他们的处理思路。

列出所有进程　show processlist　观察所有进程　多秒没有状态变化的(干掉)

查看超时日志或者错误日志（做了几年开发,一般会是查询以及大批量的插入会导致 cpu 与 i/o 上涨,,,,当然不排除网络状态突然断了,,导致一个请求服务器只接受到一半，比如 where

子句或分页子句没有发送,,当然的一次被坑经历)

sql 优化

(1)、explain 出来的各种 item 的意义；

select_type

表示查询中每个 select 子句的类型

type

表示 MySQL 在表中找到所需行的方式，又称“访问类型”

possible_keys

指出 MySQL 能使用哪个索引在表中找到行，查询涉及到的字段上若存在索引，则该索引将被列出，但不一定被查询使用

key

显示 MySQL 在查询中实际使用的索引，若没有使用索引，显示为 NULL

key_len

表示索引中使用的字节数，可通过该列计算查询中使用的索引的长度

ref

表示上述表的连接匹配条件，即哪些列或常量被用于查找索引列上的值

Extra

包含不适合在其他列中显示但十分重要的额外信息。

(2)、profile 的意义以及使用场景；

查询到 SQL 会执行多少时间，并看出 CPU/Memory 使用量，执行过程中 Systemlock，Table lock 花多少时间等等。

备份计划，mysqldump 以及 xtranbackup 的实现原理

(1)、备份计划；

这里每个公司都不一样，您别说那种 1 小时 1 全备什么的就行

(2)、备份恢复时间；

这里跟机器，尤其是硬盘的速率有关系，以下列举几个仅供参考

20G 的 2 分钟（mysqldump）

80G 的 30 分钟(mysqldump)

111G 的 30 分钟（mysqldump)

288G 的 3 小时（xtra)

3T 的 4 小时（xtra)

逻辑导入时间一般是备份时间的 5 倍以上

(3)、xtrabackup 实现原理

在 InnoDB 内部会维护一个 redo 日志文件，我们也可以叫做事务日志文件。事务日志会存储每一个 InnoDB 表数据的记录修改。当 InnoDB 启动时，InnoDB 会检查数据文件和事务

日志，并执行两个步骤：它应用（前滚）已经提交的事务日志到数据文件，并将修改过但没有提交的数据进行回滚操作。

- -

DDoS 攻击和防御的 11 种方针

DDoS 攻击，在短时间内发起大量请求，耗尽服务器的资源，无法响应正常的访问，造成网站实质下线。

DDoS 防御的方法：

1、采用高性能的网络设备

2、尽量避免 NAT 的使用

3、充足的网络带宽保证

4、升级主机服务器硬件

5、把网站做成静态页面或者伪静态

6、增强操作系统的 TCP/IP 栈

7、安装专业抗 DDOS 防火墙

8、HTTP 请求的拦截

9、备份网站

10、部署 CDN

处理服务器遭受攻击的一般流程

一、一般思路

1.切断网络

2.查找攻击源

3.分析入侵原因和途径

4.备份用户数据

5.重新安装系统

6.修复程序或系统漏洞

7.恢复数据和连接网络

二、检查并锁定可疑用户

1.登录系统查看可疑用户

2.锁定可疑用户

3.通过 last 命令查看用户登录事件（/var/log/wtmp 文件）

三、查看系统日志（/var/log/messages、/var/log/secure、家目录.bash_history 等）

四、检查并关闭系统可疑进程

五、检查文件系统的完好性

- -

- -

如何优化 Linux 系统？

不用 root，添加普通用户，通过 sudo 授权管理
更改默认的远程连接 SSH 服务端口及禁止 root 用户远程连接
定时自动更新服务器时间
配置国内 yum 源
关闭 selinux 及 iptables（iptables 工作场景如果有外网 IP 一定要打开，高并发除外）
调整文件描述符的数量
精简开机启动服务（crond rsyslog network sshd）
内核参数优化（/etc/sysctl.conf）
更改字符集，支持中文，但建议还是用英文字符集，防止乱码
锁定关键系统文件
清空/etc/issue，去除系统及内核版本登录前的屏幕显示