Binlog日志:

Diff 文档名 文档名 :对比

启用 binlog 日志

1)修改配置文件,并重启服务。

]# vim /etc/my.cnf

[mysqld]

server\_id=1 //指定 server\_id

log-bin=/mylog/db50 //指定日志目录及名称

]# mkdir /mylog //创建目录

]# chmod mysql /mylog //修改所有者

经典案例(NSD RDBMS1 DAY04)

]# systemctl restart mysqld.service //重启服务

]# mysql -uroot -p”密码”

Mysql> show master status;

Mysql> flush logs //刷新日志(刷新几次就有几个日志)

Mysql>purge master logs to “日志名称”

使用binlog日志恢复数据

]# Vim/etc/my.cnf

Binlog\_format=”mixed” //定义日志格式

server\_id=1

log-bin=/mylog/db50 //定义日至名

]# systemctl restart mysqld //起服务

2)确认 binlog 日志文件

新启用 binlog 后,每次启动 MySQl 服务都会新生成一份日志文件:

]# ls /var/lib/mysql/mysql-bin.\*

/var/lib/mysql/mysql-bin.000001 /var/lib/mysql/mysql-bin.index

]# cat /var/lib/mysql/mysql-bin.index

重启 MySQL 服务程序,或者执行 SQL 操作“FLUSH LOGS;”,会生成一份新的日志:

]# ls /var/lib/mysql/mysql-bin.\*

/var/lib/mysql/mysql-bin.000001 /var/lib/mysql/mysql-bin.index

13/var/lib/mysql/mysql-bin.000002

]# cat /var/lib/mysql/mysql-bin.index

./mysql-bin.000001

./mysql-bin.000002

步骤二:利用 binlog 日志重做数据库操作

1)执行数据库表添加操作

创建 db1·库 tb1 表,表结构自定义:

mysql> CREATE DATABASE db1;

Query OK, 1 row affected (0.05 sec)

mysql> USE db1;

Database changed

mysql> CREATE TABLE tb1(

-> id int(4) NOT NULL,name varchar(24)

-> );

Query OK, 0 rows affected (0.28 sec)

插入 3 条表记录:

mysql> INSERT INTO tb1 VALUES

-> (1,'Jack'),

-> (2,'Kenthy'),

-> (3,'Bob');

Query OK, 3 rows affected (0.12 sec)

Records: 3 Duplicates: 0 Warnings: 0

确认插入的表记录数据:

mysql> SELECT \* FROM tb1;

+----+--------+

| id | name |

+----+--------+

| 1 | Jack |

| 2 | Kenthy |

| 3 | Bob

|

+----+--------+

3 rows in set (0.00 sec)

2)删除前一步添加的 3 条表记录

执行删除所有表记录操作:

mysql> DELETE FROM tb1;

Query OK, 3 rows affected (0.09 sec)

确认删除结果:

mysql> SELECT \* FROM tb1;

Empty set (0.00 sec)

步骤三:通过 binlog 日志恢复表记录务

1)查看 mysql-bin.000002 日志内容

]# mysqlbinlog /var/lib/mysql/mysql-bin.000002

]# mysqlbinlog \

--start-datetime="2017-04-12 12:06:55" \

--stop-datetime="2017-04-12 12:07:23" \

/var/lib/mysql/mysql-bin.000002 | mysql -u root -p

Enter password: //验证口令

3)确认恢复结果

mysql> SELECT \* FROM db1.tb1;

+----+--------+

| id | name |

+----+--------+

| 1 | Jack |

| 2 | Kenthy |

| 3 | Bob

|

+----+--------+

3 rows in set (0.00 sec)

物理备份缺点:

跨平台性差

备份四件长,冗余备份,浪费存储空间

Mysqldump备份缺点:

效率低,备份和还原速度慢,锁表

备份过程中,数据插入和更新操作被阻塞

第五天

Percona在线热备:

备份过程中不锁库表(只锁行)

主含两个组件

操作:

在真机/linux-soft/03/mysql 下

libev-4.15-1.el6.rf.x86\_64.rpm

percona-xtrabackup-24-2.4.7-1.el7.x86\_64.rpm

--Host 主机名

--User 用户名

--Port 端口号

--Password 密码

--Databases 数据库名

--On-timestamp 不用日期命名备份文件存储的子目录名

选项:

Redo-only 合并日志

--apply-log 准备恢复数据

--copy-back 拷贝数据

Incremental 目录名 增量备份时指定上一次备份数据存储的目录名

--incremental-basedir=目录名 准备恢复数据时,指定增量备份数据存储的目录名

--export 导出表信息

Import 导入表空间

命令格式:

完全备份

innobackupex --user 用户名 --password 密码 备份目录名 --no-timestamp

Innobackupex --apply-log 目录名 //准备恢复数据

Innobackupex --copy-back 目录名 //恢复数据(拷贝回数据库目录下)

完全恢复

Systemctl stop mysqld

Rm -rf /var/lib/mysql/\*

Innobackupex ---apply-log 目录名

Innobackupex --copy-back 目录名

Chown -R mysql:mysql /var/lib/mysql

Systemctl restart mysqld

#################################

恢复备份中的单张表:

删除表空间

Alter table 库名.表名 discard tablespace;

导出表信息

Innobackupex --apply-log --export 数据库完全备份目录

拷贝表信息文件

Cp 数据完全备份目录/数据库名目录/表名.{ibd,cfg.exp} 数据库目录/库名目录

修改所有者/组

Chown mysql:mysql 目录名/库名

导入表空间

Alter table 库名.表名 import tablespace;

Rm -rf /var/lib/mysql/库名/表名.cfg

Rm -rf /var/lib/mysql/库名/表名.exp

查看

Select \* from 库名.表名

删除表信息文件

Rm -rf 数据库目录/库名/表名.{cfg,exp}

增量备份与增量恢复

增量备份:

完全备份

Innobackupex --user root --password 密码 备份文件名 -no-timestamp

增量备份

Innobackupex --user root --password 密码 --incremental /new1dir

--incremental-basedir=/fullbak --no-timestamp

增量备份:

Innobackupex --user root --password 密码 --incremental /new2dir

--incremental-basedir=/fullbak --no-timestamp

增量恢复:

Scp 文件 目标主机地址

关闭mysqld服务

Rm -rf /var/lib/mysql/\*

完全恢复

Innobackupex --apply-log --redo-only /备份文件名

恢复增量

Innobackupex --apply-log --redo-only /备份文件名

--incremental-dir=/new1dir

/备份文件/xtrabackup\_checkpoints

主从同步

Slave\_IO:复制master主机 binlog日志文件里的SQL命令

到本机的relay-log文件里 (master端)

SLave\_SQL:执行本机relay-log文件里的SQL语句,

实现与Master数据一致(Slave端)

构建思路:

配置主库

启用binlog日志 授权用户 查看binlog日志信息

配置从服务器

确保主服务器数据一致

这只server\_id, 指定主库信息 启动Slave程序

以51为主

修改配置文件,启用binlog日志

Vim/etc/my.cnf

[mysql]

Log-bin=日志名

Server\_id=服务器id值

Systemctl restart mysqld //起服务

进入数据库:

用户授权:只给同步的权限

Grant replication Slave on \*.\* to repluser@”%” identified by “密码”

Replication slave //同步权限

show full processlist; //查看当前有哪些主机连接我的数据库

Mysql>Show master status\G; //查看日志信息

从服务器信息

修改配置文件

指定server\_id=id值,不允许与主库的id值相同

Vim /etc/my.cnf

[mysql]

Server\_id=52

Systemctl restart mysqld

在主库上备份数据,备份文件拷贝给从库

从库使用备份文件恢复数据

再查看

mysqldump -uroot -p123456 --master-data system > /root/xx.sql

--master-data:记录一下当前备份数据对应的日志信息

scp xx.sql root@”对方IP地址”:/root

到从服务器上:(要先把库创建出来)

Mysql -uroot -p密码 数据库名 < /root/xx.sql

查看数据是否一致

mysql -uroot -p123456 -e "select \* from 库名.表名”

指定主服务器信息:

grep MASTER xx.sql //在命令行查看偏移量

Mysql>change master to

>master\_host=”主服务器的ip”

>master\_user=”主服务器授权的用户”

>master\_password=”授权用户的密码”

>master\_log\_file=”主库的日志文件”

>master\_log\_pos=”日志的偏移量”

Mysql>start slave //启动Slave进程

Mysql>show slave status\G; //查看状态信息

Master\_Host: 192.168.4.51 //主库的IP

Slave\_IO\_Running: Yes

Slave\_SQL\_Running: Yes

IO非yes:看last\_IO\_ERORR:

SQL非yes看last\_SQL\_ERORR:

(报错:解决方法(暴力解决)

Cd /var/lib/mysql

Rm -rf master.info relay-log.info

Rm -rf \*-relay -bin\*

重启服务:systemctl restart mysqld

再重新定义

Start slave

Show slave status\G; )

(必须两个都是yes)

(master信息会自动保存到/var/lib/mysql/master.info文件

若要更改主库信息时,应先执行stop Slave:修改后在执行start Slave)

相关文件:

Master.info //主库信息

Relay-log.info //中继日志信息

主机名-relay-bin.xxxxxx //中继日志

主机名-relay-bin.index //索引文件

在50客户端上测试主从配置

主服务器51添加客户端连接的用户

grant all on 库.表 to “用户”@"%" identified by "密码";

在50上:连接

在库里创建表测试,添加内容

在到服务端查看

主从同步结构

基本:

主 <------从

一主一从

扩展

一主多从:从<----主---->从

|

从

链式复制:主<----从<----从

互为主从:主<------>主(用于高可用)

配置从服务器

Vim /etc/my.cnf

[mysql]

Server\_id=”id值”

再重启服务

mysql> change master to master\_host=" 主服务器IP",

-> master\_user="用户名",

-> master\_password="用户密码",

-> master\_log\_file="日志名",

-> master\_log\_pos=偏移量;

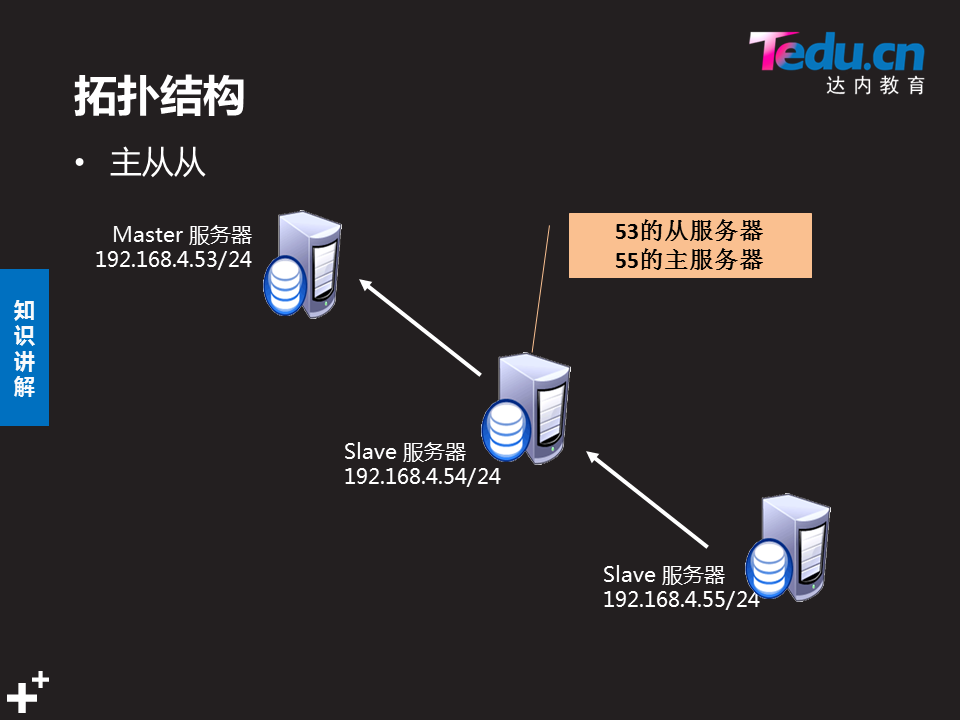
Query OK, 0 rows affected, 2 warnings (0.50 sec)

mysql> start slave; //启动Slave进程

Query OK, 0 rows affected (0.06 sec)

mysql> show slave status\G; //检查状态

配置主从从结构



启用binlog日志

Vim /etc/my.cnf

[mysql]

Log-bin=日志名

Server\_id=id值

重启服务

Systemctl restart mysqld

进入数据库授权

Grant replication slave on \*.\* to 用户@”%” identified by ” 密码”;

show master status; //查看

到54上

修改配置

Vim /etc/my.cnf

Server\_id=id值

Log-bin=”日志名”

Log\_slave\_updates //允许级联复制

起服务

Systemctl restart mysqld

授权

Grant replication slave on \*.\* to 用户名@”%” identified by “ 密码”

(作为主服务器)

作为从服务器:

修改配置

Vim /etc/my.cnf

Server\_id=id值

起服务

Systemctl restart mysqld

mysql> change master to master\_host=" 主服务器IP",

-> master\_user="用户名",

-> master\_password="用户密码",

-> master\_log\_file="日志名",

-> master\_log\_pos=偏移量;

Query OK, 0 rows affected, 2 warnings (0.50 sec)

mysql> start slave; //启动Slave进程

Query OK, 0 rows affected (0.06 sec)

mysql> show slave status\G; //检查状态

在主服务器53上建立新库,表,记录测试

在从服务器查看

复制模式:

异步模式:主库执行完后,立即将结果给客户端,不关心从库是否已接收

全同步模式:当主库执行完后,所有库都执行了该事务后才把结果给客户端

半同步模式:主库在执行完后,等待至少一个库接收到并写到relaylog总才将结果给客户端

半同步复制

查看是否允许加载动态模块

Mysql>show variables like “have\_dynamic\_loading”

若不允许:

Set have\_dynamic\_loading yes

安装模块

Install plugin rpl\_semi\_sync\_master SONAME “semicync\_master.so”; //主服务器上执行

Install plugin rpl\_semi\_sync\_slave SENAME “semisync\_slave.so”; //从服务器

Mysql>select plugin\_name,plugin\_status from information\_schema.plugins where plugin\_name like '%semi%'; //查看是否安装

开启模块:

mysql> set global rpl\_semi\_sync\_master\_enabled=1; //主服务器上

mysql> set global rpl\_semi\_sync\_slave\_enabled=1;

//从服务器上

show variables like "rpl\_semi\_sync\_%\_enabled"; //查看模块状态

+------------------------------+-------+

| Variable\_name | Value |

+------------------------------+-------+

| rpl\_semi\_sync\_master\_enabled | ON |

| rpl\_semi\_sync\_slave\_enabled | ON |

+------------------------------+-------+

永久启用半同步复制

修改主配置文件

[mysql]plugin-load=rpl\_semi\_sync\_maste=semisync\_master.so

rpl\_semi\_sync\_master\_enabled=1

////主服务器

plugin-load=rpl\_semi\_sync\_maste=semisync\_slave.so

rpl\_semi\_sync\_slave\_enabled=1

////从服务器

读写分离



部署maxscale服务

代理软件:

部署maxscale服务

修改配置: /etc/maxscale.cnf

[maxscale]

threads=1 (auto :自动)//根据CPU来创建线程的数量

[server]

Type=server //定义数据库数据库服务器主机名

Address=192.168.4.10 //master主机IP地址

Port=3306

Protocol=MYSQLBackend

[server2]

Type=server //定义数据库数据库服务器主机名

Address=192.168.4.20 //slave主机IP地址

Port=3306

Protocol=MYSQLBackend

[MySQL Monitor] //定义要监视的数据库信息

type=monitor

module=mysqlmon

servers=server1,server2 ///主从数据库的主机名

user= ///用户名

passwd=123456 ///密码

monitor\_interval=10000

[Read-Write Service] //定义要分离的数据库节点

type=service

router=readwritesplit

servers=server1,server2 //主从数据库的主机名

user= //路由用户名(登录时看配置的用户是否存在)

passwd=密码

max\_slave\_connections=100%

在主,从数据库服务器创建授权用户

Mysql>grant replication slave,replication client on \*.\* to ‘自己定义的用户名’@”%” identified by “密码”; //创建监控用户

Mysql>grant select on mysql.\* to ‘自己定义的用户名’@”%” identified by “密码”; //创建路由用户

启动服务,查看端口,停止服务

Maxscale -f /etc/maxscale.cnf

Ss -antulp | grep maxscale

Kill -9

日志文件路径:/var/log/maxscale/maxscale.log

在代理服务器本机访问:

Maxadmin -uadmin -pmariadb -P端口

~]# maxadmin -uadmin -pmariadb -P4016

MaxScale> list servers //查看监控到的服务器状态

Servers.

-------------------+-----------------+-------+-------------+-----------

Server | Address | Port | Connections | Status

-------------------+-----------------+-------+-------------+-----------

server1| 192.168.4.51 | 3306 | 0 | Master, Running

server2 | 192.168.4.52 | 3306 | 0 | Slave, Running

-------------------+-----------------+-------+-------------+-----------

测试配置:

mysql> create database hh;

mysql> use hh;

mysql> create table hh(name char(20), xx char(30));

mysql> insert into hh values('hh','dd'),('xx','lo');

客户端连接读写分离服务器

mysql -h代理服务器的IP -u创建的用户 -p密码

建库

再把库的所有权授权给登录用户(在登录show databases 验证)

验证读写分离的功能:

再从服务器本机表里插入记录(从)

到分离服务器上查询(从), 插入记录(主)

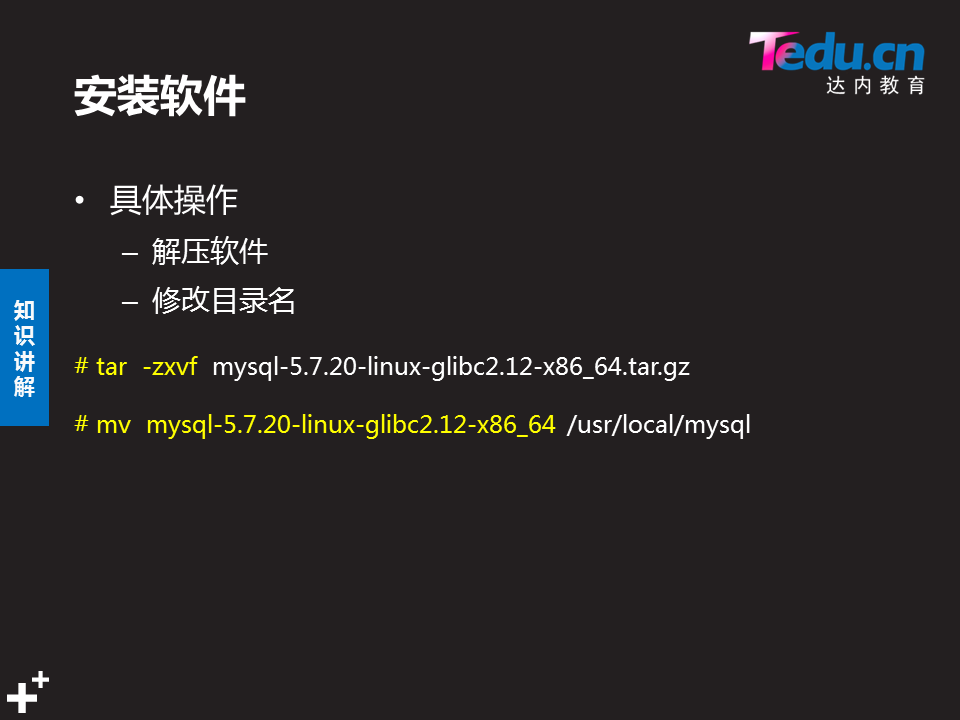
多实例

在一台主机上运行多个数据库服务

有点:节约运维成本

提高硬件利用

配置多实例:



主配置文件/etc/my.cnf

\*每个实列要有独立的数据库目录和监听端口

\*每个实列要有独立的实列名称和sock文件

mv /etc/my.cnf xx //移走

]# vim /etc/my.cnf (手写)

[mysqld\_multi] //启用多实例

mysqld=/usr/local/mysql/bin/mysql\_safe //指定进程文件路径

mysqladmin=/usr/local/mysql/bin/mysqladmin //指定管理命令路径

user=root //指定进程用户

[mysqld x] //实列进程号(有几个服务就写几个编号)

port=自定义 //端口号

datadir=/xx //数据库目录,手动创建

socket=/xx/xx.sock //指定sock文件的路径和名称

pid-file=/xx/xx.pid //进程pid号文件位置

log-error=/xx/xx.err //错误日志位置

/usr/local/mysql/bin/mysqld\_multi start //启动服务

/usr/local/mysql/bin/mysqld\_multi --user=root --password=密码 stop 实列编号 //停止服务

通过本机连接用初始密码进入数据库修改密码

数据库分片,部署mycat服务

分库分表:

(确保所有mysql处于初始化状态)

将放在一台数据库服务器中的数据,按照特定的方式进行拆分

分散在多台数据库服务器中,以达到分散单台服务负载的效果

垂直分割:

将单个按照不同的字段类别分别存储到不同的数据库

水平分割:

按照表中指定的字段的分片规则,按行分散存储在多个数据库中

软件:

Mycat:

本分基于java的分布式数据库中间件,为高并发的环境分布式存储提供解决方案

优点:

适合数据大量的写入存储需求

支持mysqk oracle sqlserver mongodb等

提供数据读写分离

基于数据分片服务

开源软件

分片规则:

枚举法 固定分片 范围约定 求模法 日期列分区发 通配取模 SACII求模通配 编程指定 字符串拆分hash解析

一致性hash



yum -y install java-1.8.0-openjdk //安装

which java //查看是否安装

Bin //命令

catlet //扩展功能

conf //配置文件

lib //jar包

logs //启用日志和运行日志

Wrapper.log //服务启动日志

Mycat.log //SQL脚本执行后的报错内容

重要配置文件:

Server.xml //连接账号及逻辑

Schema.xml //配置数据分片

Rule.xml //分片规则

\*.TXT //函数调用文件

配置服务

/usr/local/mycat/conf/server.xml(文件路径)

<user name="用户名">

<property name="password">密码</property>

用户名密码

<property name="schemas">库名</property>

逻辑库

</user>

<user>

<property name="readOnly">库名</property>

只读权限

</user>

定义分片的表:

<Schema>.......</schema> //定义分片的信息

Name //逻辑表

Datanode //指定数据节点名

Rule //指定使用的分片规则

Type=global //数据不分片存储

<datahost>.....</datahost> //定义主机名的IP地址

mycat]#sed -i "56,77d" conf/schema.xml

mycat]# sed -i "16,18d" conf/schema.xml

mycat]# sed -i "36,39d" conf/schema.xml

mycat]# vim conf/schema.xml

修改主机名和主机地址:

<dataHost name="主机名" maxCon="1000" minCon="10" balance="0"writeType="0" dbType="mysql" dbDriver="native" switchType="1" slaveThreshold="100">

<heartbeat>select user()</heartbeat>

<writeHost host="hostM2" url="主机IP地址:3306" user="root"

password="123456">

</writeHost>

password="123456" />

</dataHost>

创建库: 在数据库服务器上创建配置文件的库

添加一个授权用户

启动服务:

/usr/local/mycat/bin/mycat {console | start | stop | restart | dump}

Ss -antulp | grep :8066(默认端口)

连接测试:

配置文件的三台主机(用户)连接分片服务器

mysql -P8066 -p分片服务器地址 -u用户 -p密码

分片规则:(在schema.xml,配置问价中,每一个分片都有自己的表结构)

Sharding-by-intfile 俗称:枚举法

/函数的规则 //usr/local/mycat/conf/rule.xml

<tableRule name="sharding-by-intfile">

<rule> <columns>sharding\_id</columns>

<algorithm>hash-int</algorithm>

</rule>

</tableRule>

修改规则:/usr/local/mycat/conf/partition-hash-int.txt

10000=0 //数据存储到dn1库里

10010=1 //数据存储到dn2库里

10020=2 //数据存储在dn3库里

使用客户端连接分片服务器创建一个表(符合配置文件表名)

分别在其他数据库中登录数据库中查看相应的库里的表结构

在库中创建表:

mysql> create table employee(

-> ID int,

-> sharding\_id int,

-> name char(20),

-> tel char(20));

在插入:

mysql> insert into employee(sharding\_id,name,tel) values('10000',' ',' ');

mysql> insert into employee(sharding\_id,name,tel) values('10020',' ',' ');

mysql> insert into employee(sharding\_id,name,tel) values('10010',' ',' ');

在分别到数据库中查看这个表格;

Mod-long:求模法

mycat]#vim conf/partition-range-mod.txt

mycat]# vim conf/schema.xml

mycat]# vim conf/rule.xml

<table name="hotnews" primaryKey="ID" autoIncrement="true" dataNode="dn1,dn2,dn3"

rule="mod-long" />

删除配置文件中的主键和自增长(求模算法会起冲突)

新建表:

mysql> create table hotnews(

-> id int,

-> title char(60),

-> comment char(200),

-> worker char(20));

插入内容(必须带字段)

insert into hotnews(id,title,comment,worker) values(),

再到各个数据库去查看结果

<table name="company" primaryKey="ID" type="global" dataNode="dn1,dn2,dn3" />

不做分片处理

部署MHA集群

简介:

由日本DeNA公司开发

实现MySQL高可用的解决方案

能在0-30秒完成故障切换

故障切换过程中最大保正数据一致性

MHA管理节点:

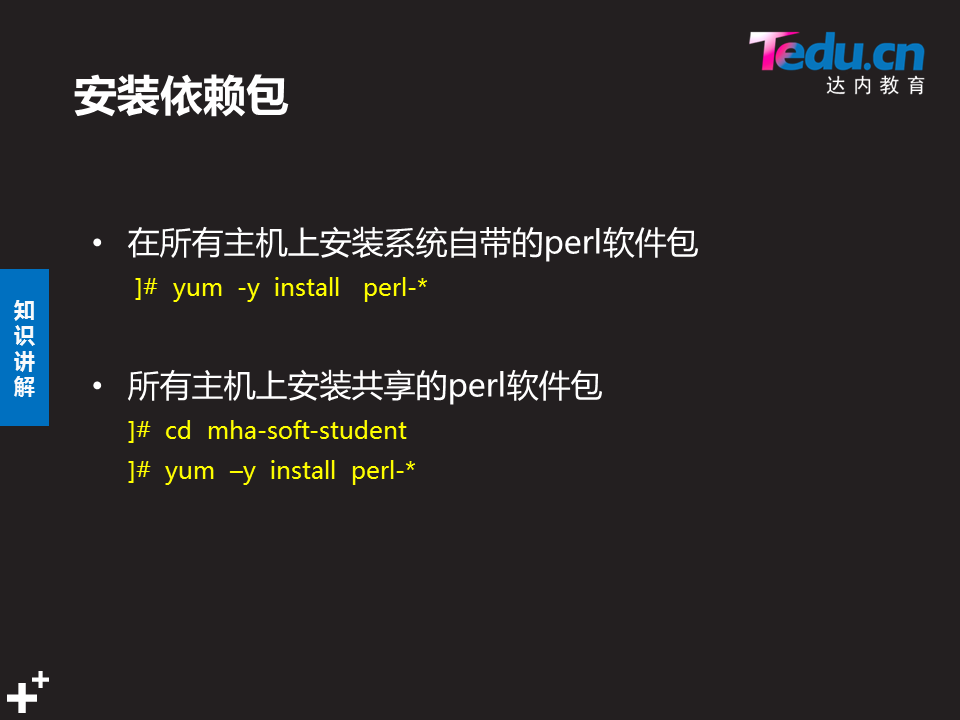
管理所有数据库服务器,也可以单独部署在一台独立机器上,

也可以部署在某台数据库服务器上

数据节点:

存储数据的MYSQL服务器

运行在每台MYSQL服务器上



配置ssh密匙认证登录(服务器每一台都要做密匙)

配置一主多从结构

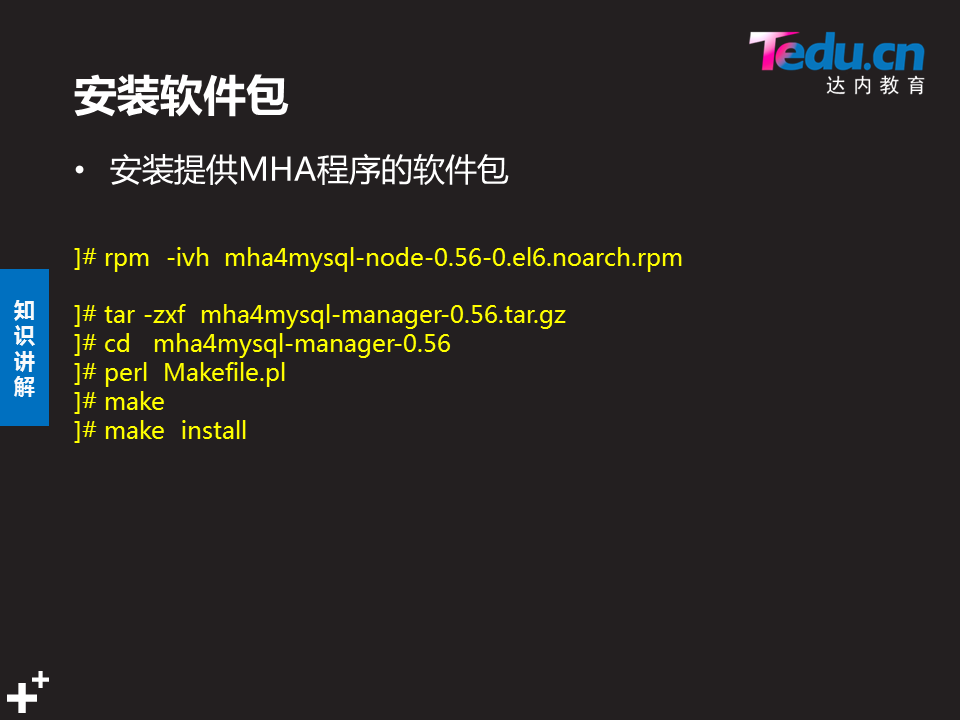
主库:

修改配置文件 , 起服务 , 授权用户 , 查看Slave状态信息(show master status)

从库:

修改配置文件 , 起服务 , 做change master同步结构 , 启动线程(startSlave) , 查看(show Slave status)

配置管理节点



谁的括号是空白就装谁

配置管理节点,配置故障切换节点

ls /usr/local/bin/masterha\_\* //查看管理集群的命令

masterha\_check\_repl //检查MySQL复制状况

masterha\_check\_ssh //检查MHA的ssh配置状况

masterha\_manager //启动MHA

masterha\_check\_status //检测MHA状态信息

masterha\_stop //停止MHA服务

配置主配置文件:

由于没有文件路径.所以需要自己创建

mkdir /etc/mha

模板可以参考拷贝包的:mha4mysql-manager-0.56/samples/conf/

cp mha4mysql-manager-0.56/samples/conf/app1.cnf /etc/mha/

编辑配置文件: vim /etc/mha/app1.cnf

[server default] //管理服务默认配置

manager\_workdir=/etc/mha/app1 //工作目录

manager\_log=/etc/mha/app1/manager.log //日志文件

master\_ip\_failover\_script=/etc/mha/master\_ip\_failover

故障切换脚本

ssh\_user=root //访问ssh服务的用户

ssh\_port=22 //端口号

repl\_user=repluser //同步数据授权用户

repl\_password=123456 //密码

user=root //监控用户

password=123456 //密码

[server1] //指定第几台服务器

hostname=192.168.4.51 //指定服务器的IP地址

port=3306 //端口号

candidate\_master=1 //竞选主库

(有几台服务器就写几个[server])

安装软件自带:mha4mysql-manager-0.56/samples/scripts/

创建故障切换脚本

~]# cp master\_ip\_failover /etc/mha/

~]# vim /etc/mha/master\_ip\_failover

my $vip = '192.168.4.100/24'; # Virtual IP

my $key = "1";

my $ssh\_start\_vip = "/sbin/ifconfig eth0:$key $vip";

my $ssh\_stop\_vip = "/sbin/ifconfig eth0:$key down";

安装软件包:

cd mha4mysql-manager-0.56/

yum -y install mha4mysql-node-0.56-0.el6.noarch.rpm

(在所有的数据库服务器上安装)

在数据库服务器上授权配置文件用户(主除外):

grant replication slave on \*.\* to repluser@"%" identified by "123456"; //数据同步用户

添加监控用户root:

grant all on \*.\* to root@"%" identified by "123456";

//监控用户

添加虚拟IP(不写永久)

~]#ifconfig eth0:1 192.168.4.100

~]# ifconfig eth0:1

eth0:1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.4.100 netmask 255.255.255.0 broadcast 192.168.4.255 ether 52:54:00:d4:82:a5 txqueuelen 1000

修改配置文件(所有数据库服务器都要做):

log-bin="xx"

server\_id=52

plugin-load = "rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so" //添加模块

rpl\_semi\_sync\_master\_enabled=1 //启用模块

rpl\_semi\_sync\_slave\_enabled=1 //启用模块

relay\_log\_purge=0 //禁止自动删除中继日志文件

检查IO和SQL线程是否是yes状态

启动MHA\_Manager:

masterha\_manager

--remove\_dead\_master //若主服务器坏了,则自动移除

--ignore\_last\_failover //忽略xxx.health文件

masterha\_manager --conf=/etc/mha/app1.cnf --remove\_dead\_master --ignore\_last\_failover //执行检测

masterha\_check\_status --conf=/etc/mha/app1.cnf

查看状态,当前的数据库服务器IP是多少

PXC

基于MySQL高可用集群解决方案

免费开源

由两部分组成(同步,多住复制插件)

官网:http://galeracluster.com

特点:

数据强一至性,无同步延迟

没有主从切换操作,不用虚IP

支持innodb存储引擎

多线程复制

支持节点自动加入,无需手动拷贝数据

服务端口:

3306 //数据库服务端口

4444 //sst端口

4567 //集群成员通信端口

4568 //ist端口

Sst //全量同步

Ist //增量同步

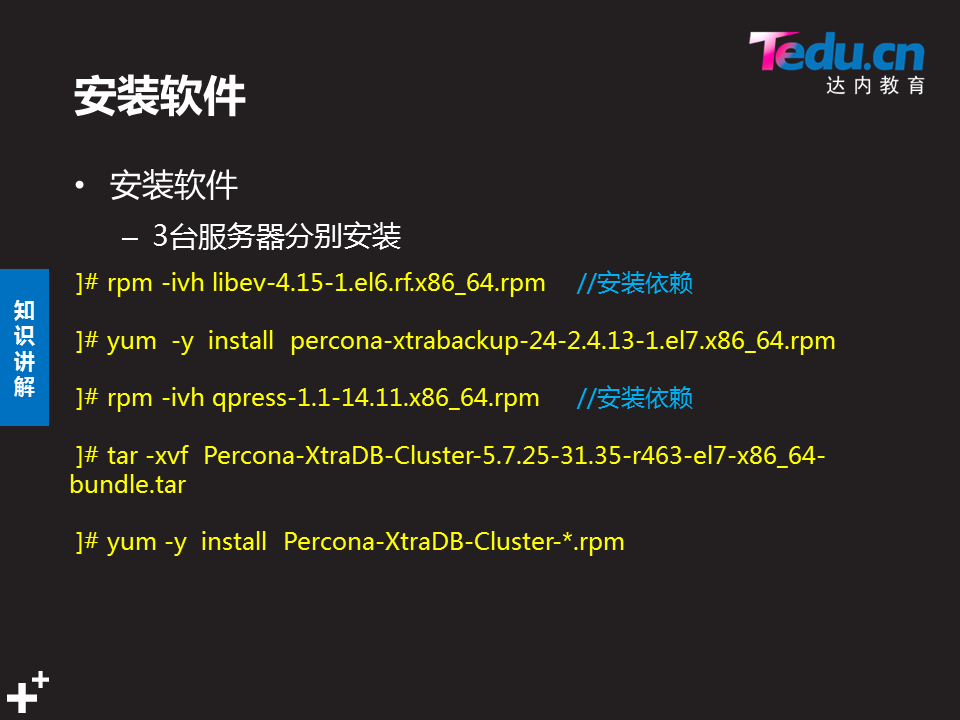
部署PXC

安装软件:

percona-release-0.1-4.noarch.rpm //在线热备 percona-xtrabackup-24-2.4.13-1.el7.x86\_64.rpm //递归压缩

qpress-1.1-14.11.x86\_64.rpm //集群服务

安装软件:



配置服务:

/etc/percona-xtradb-cluster.conf.d/ //所在目录

Mysqld.cnf //数据库服务运行参数配置

mysqld\_safe.cnf //Percona server 5.7 配置文件

Wsrep.cnf //PXC集群配置文件

修改(mysqld.cnf):

[mysqld](三台)

server-id=[三台配至] //server\_id不允许重复

datadir=/var/lib/mysql //数据库目录

socket=/var/lib/mysql/mysql.sock //socket文件

log-error=/var/log/mysqld.log //日志文件

pid-file=/var/run/mysqld/mysqld.pid //pid文件

log-bin //启用binlog日志

log\_slave\_updates //启用链式复制

expire\_logs\_days=7 //日志文件保留天数

(mysqld\_safe.cnf配置文件(不需要修改))

[mysqld\_safe]

pid-file = /var/run/mysqld/mysqld.pid //pid文件位置名称

socket = /var/lib/mysql/mysql.sock //socket文件位置名称

nice = 0

(PXC集群配置文件wsrep.cnf)

wsrep\_node\_address=ip地址 //本机IP地址

wsrep\_cluster\_name=名称 //集群名称(3台相同)

wsrep\_node\_name=主机名 //本机主机名

wsrep\_sst\_auth="用户名:123456" //sst数据同步授权用户

启动集群服务:systemctl start mysql@bootstrap

grep password /var/log/mysqld.log //查看密码

mysql -uroot -p密码 //登录数据库

mysql> alter user root@"localhost" identified by "123456" //修改登录密码

grant reload,lock tables,replication client,process on \*.\* to 用户@"localhost" identified by "密码"

//添加配置文件的授权用户

在其他机器上启动mysqld服务

Systemctl restart mysqld

查看端口

Ss -antulp | grep :3306 和:4567

在主机上查看状态:show status like "%wsrep%";

wsrep\_incoming\_addresses: | 192.168.4.72:3306,192.168.4.73:3306,192.168.4.71:3306

wsrep\_cluster\_size 3 //集群的服务台数

wsrep\_cluster\_status Primary //服务器状态

wsrep\_connected ON //链接状态

wsrep\_ready ON //集群状态

随意授权一个用户,随意任意一台机器在客户端登录,建库,建表

每台机器都能查看到

存储引擎

Mysql服务软件自带的功能程序,处理表的能力

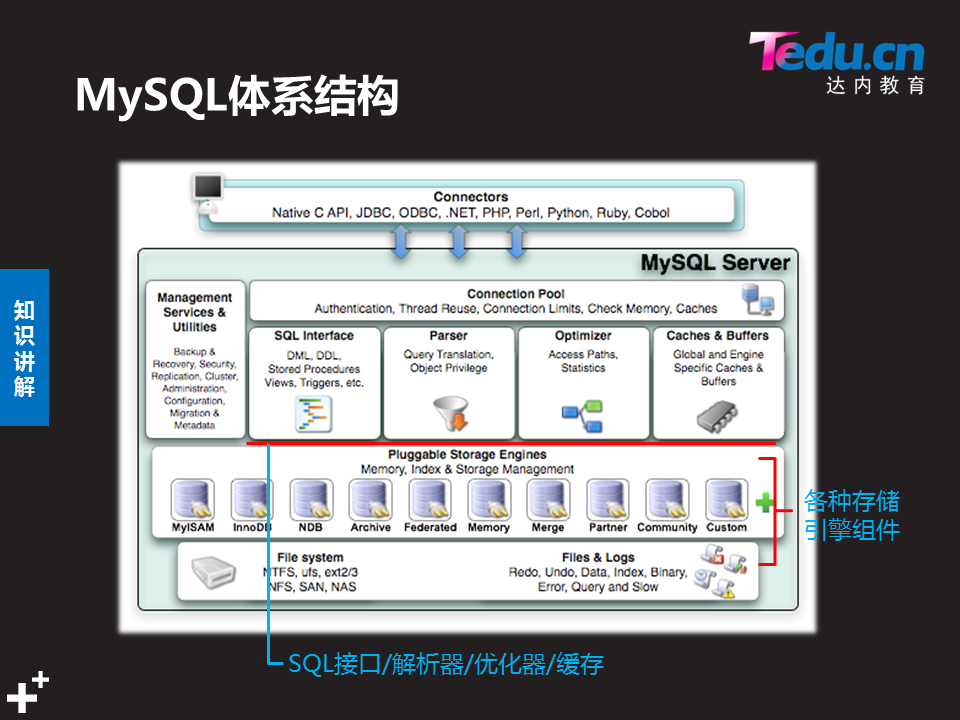
不同的存储引擎有不同的功能和数据存储方式

MySQL5.0/5.1 ----->MYISAM

MYSQL 5.5/5.6 ----->INNODB

show engines; //列出可用的存储引擎类型

Mysql体系结构:



连接池

Sql连接口,分析,优化,缓存(查询)

工具

文件

配置存储引擎

建表时手动指定,未指定是默认存储引擎

在扩号外添加 “) engine=引擎”

修改引擎:

vim /etc/my.cnf

[mysqld]

default-storage-engine=存储引擎名称

在到数据库查看引擎状态:show engines;

特点

支持表级锁

不支持事务,事务回滚,外键

表文件:

表名.frM //表结构

表名.MYI //索引

表名.MYD //数据

表名.ibd

事务日志文件

Ibddata1

Id\_logfile0

Id\_logfile1

Mysql锁机制:

表级锁:对整张表加锁

行级锁:仅对某一行分别加锁

锁类型:

读锁(共享锁):支持并发读

写锁(互斥锁,排他锁):上锁期间其他线程不能读写表

Show status like “table\_lock%”; //查看当前锁状态

Table\_locks\_waited | 0 //当前等待的用户有多少

事务特性:

Atomic:原子性

是一个整体要么失败,要么成功

Consistency:一致性

事务操作的前后,表记录没有变化

Isolation:隔离性

操作时互相隔离不收影响

Durability:持久性

永久改变数据

相关命令

show variables like "autocommit" //查看提交状态

Set autocommit=off //关闭自动提交

Rollback //数据回滚

Commit //提交数据

Redis

主流的RDBMS关系型软件: MySQL ,mariadb ,oracle ,db2 ,sql server

redis没有表结构,非关系型数据库

redis服务

高性能的分布式内存储数据库

支持数据持久化(定期存储到硬盘)

支持多种数据类型string ,list ,hash

支持master-Slave模式数据备份

支持中文网站

安装软件:

Yum -y install gcc

Tar -xf redis-4.0.8.tar.gz //源码包

Make && make install

初始配置:

./utlis/install\_server.sh //初始化

6379 //端口号

/etc/redis/6379.conf //主配置文件

/var/log/redis\_6379.log //日志文件

/var/lib/redis/6379 /数据库目录

/usr/local/bin/redis-server //服务启动程序

/usr/local/bin/redis-cli //命令行连接命令

/etc/init.d/redis\_6379 stop //停止服务

/etc/init.d/redis\_6379 start //启动服务

Ps -C redis-server //查看进程

连接服务:

常用命令:

Set key 名 key 值 //存储一个key值

Mset key 名列表 //存储多个key值

Get key名 //获取

Mget 获取多个

Select 数据库编号0-15 //切换库

Keys \* //显示所有

Keys a? //显示指定

Existi key名 //测试key名是否存在

Ttl key名 //显示key生存时间

Type key名 //查看key类型

Move key 库编号 //移动到key指定的库

Expire key 名 数字 //设置key有效时间

Del key名 //删除指定的key值

Flushall //删除内存里所有的key值

Flushdb //删除所在库的所有key值

Save //保存所在库所有key值

Shutdown //停止服务

文件配置分类:  
NETWORK //网络

GENERAL //常规

SNAPSHOTTING //快照

REPLICATION //复制

SECURITY //安全

CLIENTS //客户端

NEMORY MANAGEMENT //内存管理

数据单位:

1k => 1000 bytes

1kb => 1024 bytes

1m => 1000000 bytes

1mb => 1024\*1024 bytes

1g => 1000000000 bytes

1gb => 1024\*1024\*1024 bytes

常用配置:

93 Port 6379 //端口

70 bind 192.168.1.100 10.0.0.1 //IP地址

137 daemonize yes //守护进程方式运行

187 databases 16 //数据库个数

172 logfile /var/log/redis\_6379.log //日志文件(启动和报错)

533 # maxclients 10000 //并发连接数

264 dir /var/lib/redis/6379 //数据库目录

内存管理:

565 # volatile-lru //最近最少使用

566 # allkeys-lru //删除最少使用的key值

567 # volatile-lfu //从所有配置了过期时间的key中清除使用频率最少的key

568 # allkeys-lfu //清除所有的key使用频率最少的key

569 # volatile-random //在设置了TTL的key里随机移除

570 # allkeys-random //随机移除key值

571 # volatile-ttl -> //移除最近过期的key

572 # noeviction //随机移除key值

Maxmemory //最大内存

591 # maxmemory-policy //定义使用策略

602 # maxmemory-samples 选取key模板的个数(lru和ttl策略)

LNMP和redis平台

部署LNMP:

Yum -y install php-devel php-fpm openssl-devel gcc

tar -xf lnmp\_soft.tar.gz

tar -xf nginx-1.12.2.tar.gz -C /root

cd nginx-1.12.2/

./configure --with-http\_ssl\_module --user=nginx --group=nginx

Make && make install

vim /usr/local/nginx/conf/nginx.conf

部署动态页面

起服务systemctl restart php-fpm(nginx)

配置支持redis

cd lnmp\_soft/

tar -xf redis-3.0.6.tar.gz -C /root

cd redis-4.0.8/

make && make install

./utils/install\_server.sh //初始化

安装php扩展(依赖关系)

需要php php-devel软件包

rpm -q autoconf automake //检查

tar -xf php-redis-2.2.4.tar.gz

cd phpredis-2.2.4/

Phpize //生成配置文件php-config和configure命令

./configure --with-php-config=/usr/bin/php-config

make && make install

/usr/lib64/php/modules/ //保存地址

vim /etc/php.ini

728 extension\_dir = "/usr/lib64/php/modules/";

//模块目录

730 extension = "redis.so"; //模块名

systemctl restart php-fpm //起服务

php -m | grep redis //查看是否支持redis

测试配置php连接redis

/linux-soft/03/redis/lnmp

cp linkredis.php /usr/local/nginx/html/save.php

vim //usr/local/nginx/html/save.php

<?php

$redis = new redis();

$redis->connect('192.168.4.50',6350);

$redis->auth('123456');

$redis->set('redistest','opreation');

?>

(存的脚本)

<?php

$redis = new redis();

$redis->connect('192.168.4.50',6350);

$redis->auth('123456');

echo $redis->get('redistest');

?>

(取的脚本)

<?php

$redis = new redis();

$redis->connect('192.168.4.50',6350);

$redis->auth('123456')

$redis->set('six','hash');

echo $redis->get('six');

?>

(存取一体脚本)

创建集群(除了客户端和最好一台其余的都要执行)

Sed-i"70p;93p;815p;823p;829p;501p"/etc/redis/6379.conf

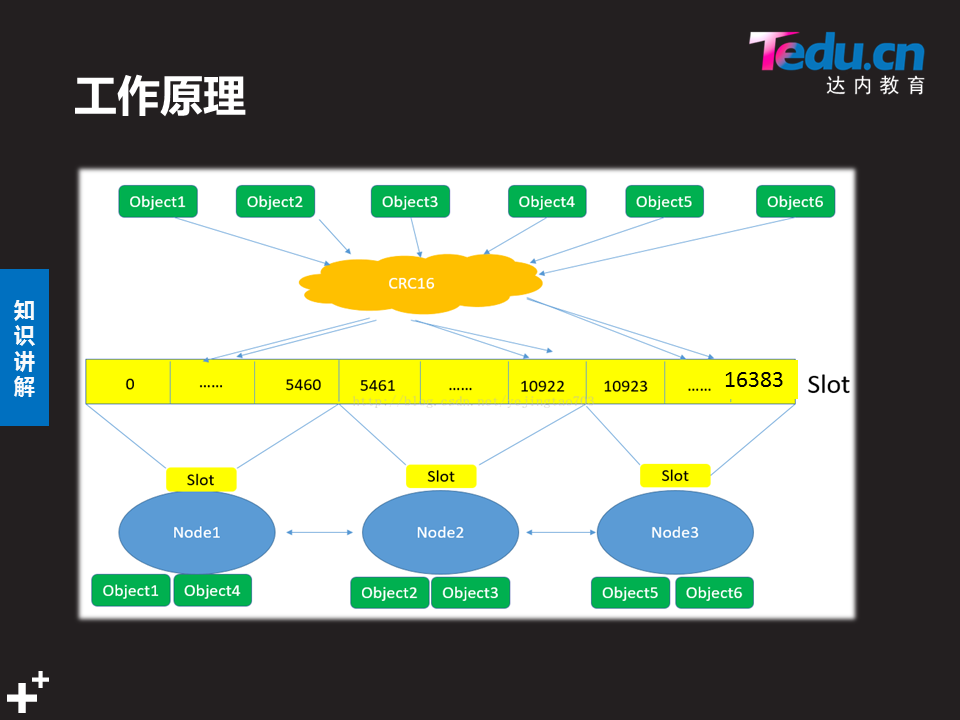
vim /etc/redis/6379.conf

rm -rf /var/lib/redis/6379/

vim +43 /etc/init.d//redis\_6379

ls /var/lib/redis/6379/

/etc/init.d/redis\_6379 start



集群里hash槽的个数有16384个

16384/主机数 平均分配槽

存在哪一个服务器上

部署管理主机

Redis提供一个ruby脚本(创建)

rpm -q rubygems //查看是否有ruby环境

yum -y install rubygems //安装

scp /linux-soft/03/redis/redis-3.2.1.gem root@主机地址:/root //插件,确保能通信

gem install redis-3.2.1.gem //安装

cd redis-4.0.8/src/

ls \*.rb

mkdir /root/bin //创建命令检索目录

cp \*.rb /root/bin/ //创建管理集群脚本

chmod +x /root/bin/redis-trib.rb

bin/redis-trib.rb help //查看命令帮助

创建集群环境:

redis-trib.rb create --replicas 1 主机名:端口号

提示(can't connect to node 主机名:端口)

修改配置文件的bind回环地址改成主机地址就行

查看集群信息:

在管理主机查看 redis-trib.rb info 主机IP:端口

在服务主机查看:

redis-cli -h 主机IP -p 端口

6379> cluster info

cluster\_state:ok

......

cluster\_current\_epoch:6

cluster\_my\_epoch:3

6379> cluster nodes //查看集群节点信息

访问集群:

redis-cli -c -h IP地址 -p 端口

-c :集群模式

6379> set school hh //存储数据

6379> get hh //取数据时会显示存储在那一台机器并自动跳转

-> Redirected to slot [12077] located at 192.168.4.53:6379

集群存储原理:

存取时都会变量名和crc16算法做hash计算 ,计算后会得到一个数字,和16384取余 ,根据余数的值找对应的redis服务器存储数据

测试高可用:

关闭一台redis服务器,对应的从库会自动升成主库

重新启动关掉的redis服务器,会自动接替成为从库

并自动同步宕机期间产生的新数据

添加新的集群:

做新的两台redis服务

添加时不指定主机角色,默认新主机被选为master

添加的master主机,需要手动分配hash值

redis-trib.rb add-node 新的主机IP:端口 主库IP

>>> Adding node IP to cluster IP

[OK] New node added correctly //添加成功

添加hash槽个数:

How many slots do you want to move (from 1 to 16384)?

接受hash槽的ID:

What is the receiving node ID?

移除hash槽的主机ID

all

迁移hash槽时,数据也会跟着移除

添加从服务器(谁的从少做谁的,从一样时就随意选择)

redis-trib.rb add-node --slave 要添加的主机IP ip,端口

(此处使用的默认)

redis-trib.rb info IP管理主机

redis-trib.rb chack IP //具体显示主从信息

平均分配hash槽:

redis-trib.rb rebalance 主机:端口

移除Slave服务器

因为没有hash槽,所以可以直接移除

redis-trib.rb del-node 主机IP:端口 ID值

SHUTDOWN the node //停止移除主机的redis服务

移除master服务器

首先要释放占用的hash槽

redis-trib.rb reshard 主机IP:端口

What is the receiving node ID? 要接受hash值的ID值

Source node #1:移除谁的hash值的的ID值

Source node #2:done

最后在移除主机:

redis-trib.rb del-node 主机IP:端口 ID值

添加移除的主机:

启动要添加的主机的redis服务

进入主机的redis数据库

redis-cli -h 192.168.4.53 -p 6379

192.168.4.53:6379> CLUSTER RESET //重置数据库信息

在到管理主机添加

redis-trib.rb add-node 主机IP:端口 任意一台主机

在添加hash槽

redis-trib.rb reshard 要添加hash槽的主机IP:端口

移出hash槽个数

接受hash槽主机ID

移出hash槽主机的ID

添加是(all) 移出是(done)

主从复制

主服务器接受客户端访问

从服务器自动同步主服务器数据到本机

工作原理:

配置主从复制

配置从库:

Info replication //查看复制信息

\首先进入配置文件修改bindIP,否则会在主服务器上无法查看(所有服务器)

配置从服务器

在从服务器上进入redis数据库

Slaveof 主库IP 端口

永久配置:

/etc/redis/6379.cnf

282 slaveof <masterip> <masterport>

主从连接成功时,数据会自动同步;

将从服务器恢复为主服务器

redis-cli -h 主机IP

slaveof no one //恢复主服务器

永久配置:

/etc/redis/6379.conf

注释掉,起服务就行

带验证的主从复制

配置master,设置连接密码

在重启服务

vim +501 /etc/redis/6379.conf

requirepass 密码

/etc/init.d/redis\_6379 start //起服务

哨兵服务

监视master服务器,发现宕机后将Slave服务器升级为master服务器

配置服务:

cd redis-4.0.8/

cp sentinel.conf /etc/

15 bind 0.0.0.0 //哨兵服务地址

69 sentinel monitor 主机名 IP地址 数量 //监视主服务器

71 sentinel auth-pass 主机名 密码 //主服务器密码

redis-sentinel /etc/sentinel.conf //启动哨兵服务

再开一个终端验证:宕机时,会自动把从服务器升为主服务器(启动主服务后自动识别等待时间有点长)

/etc/init.d/redis\_6379 stop

redis-cli -h 192.168.4.51

ss -antulp | grep redis

/etc/init.d/redis\_6379 start

ss -antulp | grep redis

redis-cli -h 192.168.4.51

>info replication

持久化

RDB(redis database)

数据持久化方式之一

数据持久化默认方式之一

按照指定的时间将内存中的数据集快照写入硬盘

快照 (snapshot)

恢复时将快照文件直接读入内存

配置服务:

vim **/**etc**/**redis**/6379.**conf

254 dbfilename "dump.rdb" //定义RDB的文件名

219 save 900 1 //15分钟有1个key改变

220 save 300 10 //5分钟有10个key改变

221 save 60 10000 //1分钟有10000个key改变

备份数据:

/etc/init.d/redis\_6379 stop //先暂停服务

rm -rf /var/lib/redis/6379/\* //删除数据库目录的所有文件

/etc/init.d/redis\_6379 start //启动服务

redis-cli -h IP地址 //进入redis库写入内容

cp /var/lib/redis/6379/dump.rdb /mnt/ //复制数据库下的文件

scp /mnt/dump.rdb [root@192.168.4.52:/root](mailto:root@192.168.4.52:/root) //发给另一台主机

在另一台服务器上

关闭服务,删除原有的数据库目录,把新的目录文件放到下面,在起服务,就完成了数据备份.

优点:高性能的持久化实现,比较适合大规模数据恢复,且对数据完整性不是很高的场合

缺点:意外宕机时,丢失最后一次持久化的所有数据

AOF

追加的方式记录写操作的文件

记录redis服务所有操作

不断的将新的写操作,追加到文件的末尾

默认没有启用,用cat可以查看

:6379> config set appendonly yes //启用AOF

:6379> CONFIG REWRITE //写进配置文件

备份数据:

]# ls /var/lib/redis/6379/

appendonly.aof dump.rdb

~]# cd /var/lib/redis/6379/

]# cp appendonly.aof /mnt //复制到其他位置

恢复数据:

/etc/init.d/redis\_6379 stop

mv /mnt/appendonly.aof /var/lib/redis/6379/ //拷贝备份文件到数据库目录

/etc/init.d/redis\_6379 start //重启redis服务

优化配置

appendonly yes //开启AOF服务,默认是no

appendfilename "appendonly.aof" //定义文件名称

AOF文件记录写操作的方式

# appendfsync always //实时记录,并完成磁盘同步

appendfsync everysec //每一秒记录.并完成磁盘同步

# appendfsync no //写入AOF,不执行磁盘同步

优化配置:

auto-aof-rewrite-percentage 100 //再次重写,增长百分比

auto-aof-rewrite-min-size 64mb //首次重写触发值

64mb第一次是64\*2,第二次在64\*2的基础上在\*2

修复AOF文件

~]# redis-check-aof --fix appendonly.aof //把文件恢复到最后的正确操作

优点:可以灵活的设置持久化方式,出现宕机时,尽可能丢失1秒的数据

缺点:持久化文件的体积通常会大于RDB方式

执行fsync策略时的速度可能比RDB方式慢

数据类型

String字符串:

Set key value [EX seconds] [PX milliseconds] [NX|XX]

:6379>set x 8 //x值为8

下次在赋值时,就会覆盖原来的值

Set格式是字符模式

6379> set tell 123456

6379> SETRANGE tell 1 789 //从字符的第一为开始改,为789

(integer) 6

192.168.4.51:6379> get tell

"178956"

6379> SETRANGE tell 5 xxx //多出几位则增加几位

(integer) 8

6379> STRLEN key //统计字符串的长度

6379> APPEND key //给已有的变量追加值

1G=1024M

1M=1024k

1K=1024B

1字节=8位(0或1)

6379> SETBIT key(值) offset(设置第几位) value(值为1或0)

6379> DECR key //自减

DECRbykey 10 //自定义减的个数这里是10

6379> GETRANGE key 5 3 //返回字串中的字串值(-为倒数),大数在后

List是一个字符队列

先进后出

一个key可以有多个值

6379> LPUSH key 1 2 3 //值为3 2 1

6379> LRANGE key 0-1 //从0开始读到结束

0 2 //从0开始,读到2为止

0 -2 //从开始读到倒数第二为为止

6379> LPOP key //删除值里面的开头

6379> LLEN key //查看key有多少值

6379> LINDEX key 5 //输出key中的第几个值

6379> LSET key 5 xx //修改key值中的第几个值改为什么

6379> RPUSH key ss //将ss插入到值中的末尾

6379> LPOP key //删除末尾的值(若在key后加上值,那么就删除指定的字段)

Hash表

是一个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

讲一个对象存储为hash类型,教于每个字段都存储成string类型更能节省内存

6379> HSET xx baidu 54654 //存储在xx表中的baidu里,可以存储多个值

6379> HGET xx baidu //输出xx表中的baidu里的值

HMSET xx jd 54646 xh 4683 //同时给hash表中的多个xx赋值

6379> HMGET xx jd baidu xh //输出hash表中多个xx的值

6379> HKEYS xx //输出hash表中所有的field名称

6379> HGETALL xx //输出hash表中所有field名称和值

6379> HVALS xx //输出hash表中的所有field值

6379> HDEL xx jd //删除hash表中的field的值(不存在则忽略)

监控与服务安全

目地:

报告系统运行状况,提前发现问题

资源类别:

公开数据

Web,FTP,SSH,数据库等应用服务,TCP或UDP端口

私有数据:  
CPU,内存.磁盘,网卡流量等使用信息,用户,进程等运行信息

常用的自动化监控系统:

CACTI:基于snmp协议的监控软甲,强大的绘图能力

Nagios:基于Agent监控,强大的状态检查与报警机制,

插件很多,自己写监控脚本潜入到Nagios非常方便

Zabbix:基于多种监控机制,支持分布式监控

Free:内存

Traceroute:路由信息,到目地所经过的路由信息

Iostat:硬盘每秒写的字节数和读的字节数

Zabbix

优点:是一个高度集成的监控解决方案

可以实现企业级的开源分布式监控

通过C/S(client server)模式采集监控数据

通过B/S(浏览器方式查看)模式实现web管理

支持snmp,简单的网络协议

默认端口10050

监控服务器:

可以通过cnmp或agent采集数据并写入数据库中

使用LNMP实现前段的管理

被监控主机需要安装agent服务,常见的网络一般支持snmp.

部署LNMP



修改配置文件:改动态配置文件就行,最后在起服务

~]# systemctl restart mariadb

~]# systemctl restart php-fpm

写一个测试页面,php脚本

部署Zabbix

~]# yum -y install net-snmp-devel curl-devel libevent-devel

//安装Zabbix源码包

tar -xf zabbix-3.4.4.tar.gz //解压Zabbix tar包

cd zabbix-3.4.4/

./configure --enable -server --enable -proxy --enable -agent

//客户端监控

--with-mysql=/usr/bin/mysql\_config //支持nginx程序

--with-net-snmp --with-libcurl

//Zabbix安装模块

--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

--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，

|  |  |
| --- | --- |
| 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 |
| 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,就可以通过curl链接http等服务 |

可以通过curl连接http等服务，测试被监控主机服务的状态

make install //源码安装

~]# ls /usr/local/etc/ //查看配置文件路径

~]# ls /usr/local/bin/ //查看命令路径

~]# ls /usr/local/sbin/ //服务的启动程序

创建数据库与数据库账户

~]# mysql

> create database zabbix character set utf8;

> grant all on zabbix.\* to zabbix@"localhost" identified by "123456"

cd /zabbix/database/mysql/

mysql -uzabbix -p123456 zabbix < schema.sql

mysql -uzabbix -p123456 zabbix < images.sql

mysql -uzabbix -p123456 zabbix < data.sql

//发送数据到建的库(切记,发送顺序要严格按照以上步骤)

部署zabbix页面:

zabbix-3.4.4/

cd zabbix-3.4.4/frontends/php/

cp -a \* /usr/local/nginx/html/

ls /usr/local/nginx/html/

~]# chmod -R 777 /usr/local/nginx/html/\* //赋予权限

到浏览器访问192.168.2.5/index.php



没有解决依赖会报错

解决依赖:

/usr/local/nginx/conf/nginx.conf 在http下复制添加:

fastcgi\_buffers **8** 16k**;**          //缓存php生成的页面内容，8个16k

fastcgi\_buffer\_size 32k**;**           //缓存php生产的头部信息

fastcgi\_connect\_timeout **300;**         //连接PHP的超时时间

fastcgi\_send\_timeout **300;**           //发送请求的超时时间

fastcgi\_read\_timeout **300;**            //读取请求的超时时间

安装依赖软件:

yum -y install php-gd php-xml php-ldap php-bcmath php-mbstring.x86\_64

修改php配置文件:

vim /etc/php.ini

878 date.timezone = Asia/shanghai //设置时区

384 max\_execution\_time = 300 //最大执行时间,秒

post\_max\_size = 16M //post数据最大容量

max\_input\_time = 600 //服务器接受数据的时间限制

在起服务:

/usr/local/nginx/sbin/nginx

systemctl restart php-fpm

最后在浏览器访问192.168.2.5/index.PHP

就不会报错

修改服务端的配置文件:/usr/local/etc/zabbix\_agentd.conf

85 DBHost=localhost //数据库主机(只允许本机登录)

95 DBName=zabbix //设置数据库名称

111 DBUser=zabbix //设置数据库账户

119 DBPassword=123456 //设置数据库密码

38 LogFile=/tmp/zabbix\_server.log //设置日志

useradd zabbix //必须要想创建用户

zabbix\_agentd

ss -antulp | grep zabbix

配置客户端

安装zabbix软件,提供zabbix\_agentd服务

配置主配置文件

//usr/local/etc/zabbix\_agentd.conf //配置文件路径 客户端配置文件

93 Server=127.0.0.1,192.168.2.5 //允许访问服务地址列表

134 ServerActive=192.168.2.5:10051 //监控服务器IP地址

30 LogFile=/tmp/zabbix\_agentd.log //日志文件

启动服务:

useradd zabbix //必须要先创建zabbix用户

]# zabbix\_

zabbix\_agentd zabbix\_get zabbix\_sender

]# zabbix\_agentd //启动服务

管理员登录服务器

添加监控主机

选择监控模板

查看监控信息

自定义监控

修改被监控端的配置文件:

280 UnsafeUserParameters=1 //是否允许自定义key

264 Include=/usr/local/etc/zabbix\_agentd.conf.d/ //加载配置文件

创建自定义key

cd /usr/local/etc/zabbix\_agentd.conf.d/

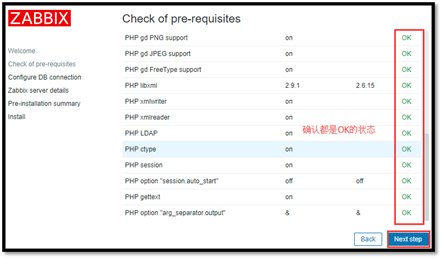
vim \*.conf

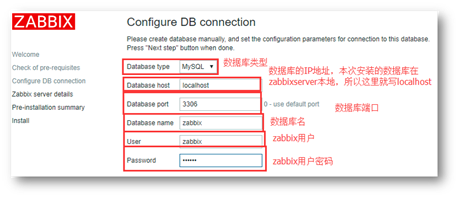
killall -9 zabbix\_agentd

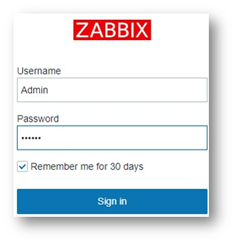
zabbix\_agentd

zabbix\_get -s 127.0.0.1 -k count.line.passwd //检测key是否生效

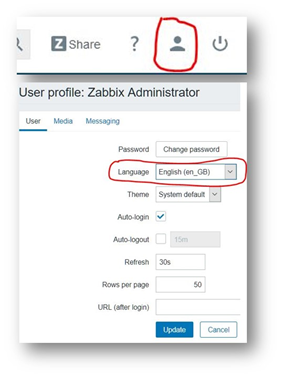
图形操作”



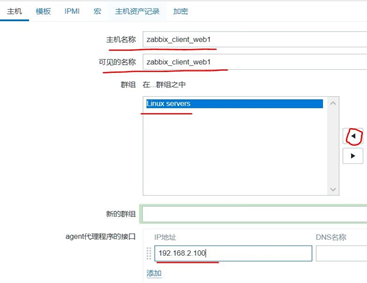




在登陆页面，使用用户(admin)和密码(zabbix)登陆，登陆后设置语言环境为中文

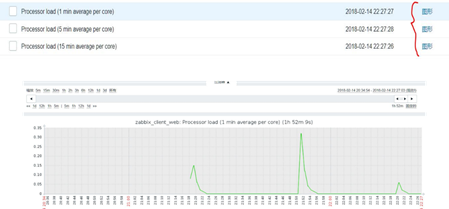




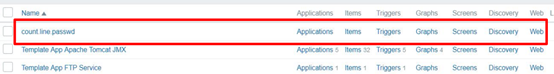










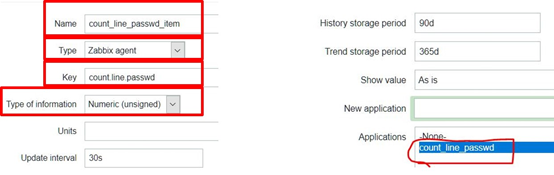


IMG_256

创建应用集

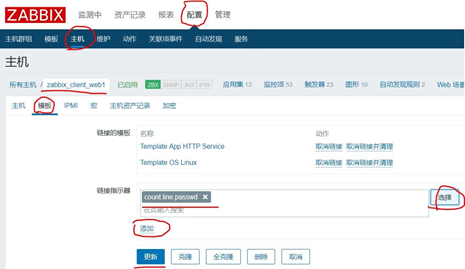






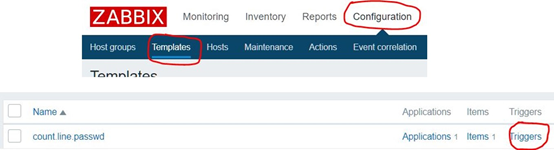






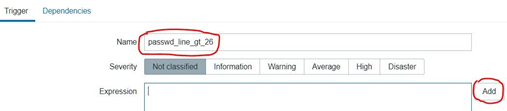


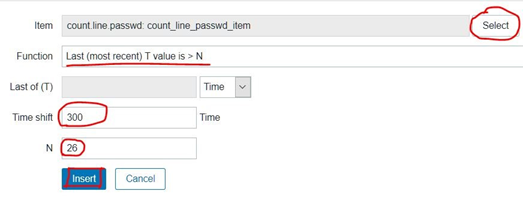
配置触发器:



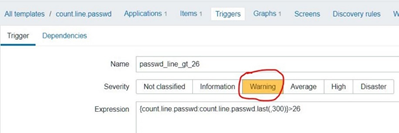
触发器表达式:

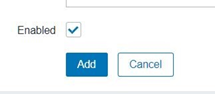






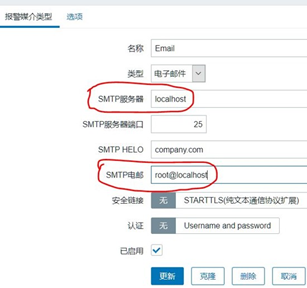
警报级别:





设置邮件:





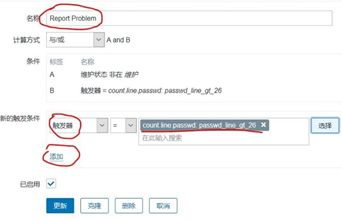




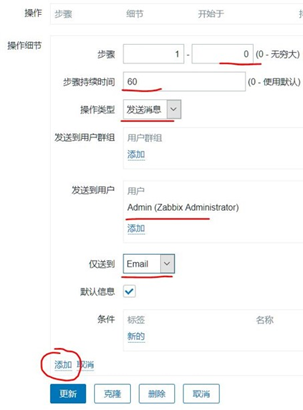


创建Action动作











测试验证

自动发现:



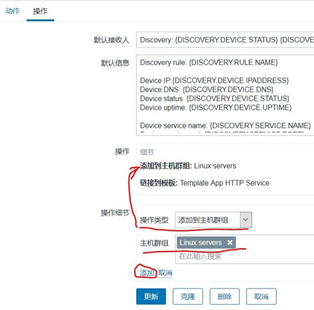




创建动作:







常见新的虚拟机(安装http服务,起服务)

验证结果

创建新的虚拟机主动监控

yum -y install gcc pcre-devel

tar -xf zabbix-3.4.4.tar.gz

cd zabbix-3.4.4/

./configure --enable-agent

make install

vim /usr/local/etc/zabbix\_agentd.conf

93 #Server=127.0.0.1,192.168.2.5 //注释掉

118 StartAgents=0 //禁止被动监控

134 ServerActive=192.168.2.5:10050 //监控服务器IP127.0.0.1要取消

145 Hostname=Zabbixserver //告诉监控服务器是谁发的数据信息(一定要和zabbix服务器配置的监控主机名称一致)

183 RefreshActiveChecks=120 //默认120秒检测一次

useradd zabbix

zabbix\_agentd

ps -C zabbix\_agentd //查看zabbix进程

在客户端编写监控脚本

Linux安全防护

用户账户安全:

设置账号有效期

使用chage工具

-d 0 强制修改密码

-E \*-\*-\*,指定失效日期

-l 查看用户的密码时间

账号的锁定和解锁:

使用passwd命令 (-l锁定 -u解锁 -S看状态)

配置文件/etc/login.defs (grep -v '#' /etc/login.defs )不看加#号的行

对新建的用户有效

主要控制属性:

25 PASS\_MAX\_DAYS:最大有效天数

26 PASS\_MIN\_DAYS :最小有效天数

27 PASS\_WARN\_AGE:密码过期警告

伪装登录提示:

配置文件:

vim /etc/issue

issue issue.net

]$ vim /etc/issue.net

本地 远程

默认提示内核,系统版本等信息

\S(系统版本)

Kernel \r(内核版本) on an \m

文件系统控制:

RHEl7

~]# systemctl disable 服务程序 //设置不开机自启

Chkconfig 服务名 off //RHEl6,设置开机不自启

Chkconfig 服务名 on //RHEl6,设置开机自启

锁定/解锁保护文件

EXT3,EXT4文件属性控制

Chattr,lsattr

+, -, = 控制方式

属性i:不可变

属性a:只能追加,不能修改

删除属性:(添加是减号改为加号就行)

chattr -a /etc/hosts

~]# chattr -i /etc/hosts

Chattr:

A:(Atime),告诉系统不要修改这个文件的最后访问时间

S:(sync),一旦应用程序对这个文件进行了写操作,立即把修改的加过写到磁盘

b:不更新文件或目录的最后存取时间

Lsattr:

a:显示所有文件和目录

d:显示目录名,不是内容

R:递归

V:(小):显示文件或目录名

V:(大):显示版本信息

用户切换与提权身份

Su:

不加- ,依然是上一个用户的环境,只是名字变了

加- , 求换环境,名字变成切换的用户

-c :~]# su - -c "命令" 用户 //切换到哪一个用户下执行-C里面的命令

用户提权:

设置系统用户执行root用户的权限

安全日志:/var/log/secure,普通用户执行的su操作

Jan 26 04:44:37 localhost login: DIALUP AT ttyS0 BY root

时间 主机 执行了什么操作

Sudo提升执行权限:

配置提权主配置文件:/etc/sudoers 保存必须用强制保存退出 (wq!)

格式1:

用户名 主机名列表=命令列表

格式2:

%用户组名 主机名列表=命令列表

查看提权命令:  
sudo -l

执行提权命令:把用户添加到wheel组  
sudo 提权命令 (比如 sudo yum -y install 软件包)

所以为了以防万一:

99 #%wheel ALL=(ALL) ALL //要把这个组注释掉

用户名 主机=/usr/bin/vim etc/my.cnf , /etc/hosts (执行操作的文件)

用户名 主机=/usr/bin/vim etc/my.cnf , !/etc/hosts /禁止执行/etc/hosts,这个文件(!)取反

修改全局配置,启用日志:

Defaults logfile="/var/log/sudo.log"

要提权后才能有这个日志,(电脑创建)

比如,登录一个用户创建一个用户

再去看这个文件就有了

Sudo别名设置

用户别名 别名名称=用户

主机别名 别名名称=主机

命令别名 别名名称=命令

User\_alias

Host\_alias

Cmnd\_alias

Ssh基本防护

存在安全隐患

密码嗅探,键盘记录,暴力枚举账号,猜解密码

防范措施:

用户限制,黑白名单,更新验证方式,防火墙

配置操作

Ssh的主配置文件:

/etc/ssh/sshd\_config

17 Port 2222 //修改非标准端口

19 ListenAddress 0.0.0.0 //允许哪一个地址登录

38 PermitRootLogin yes //默认同意root用户登录

115 UseDNS no //不解析客户端地址

37 LoginGraceTime 2m //登录限时

40 MaxAuthTries 3 //默认连接次数

黑白名单配置:(向配置文件内加入(仅允许一部分用户从指定的地址登录))

DenyUsers user1 user2

AllowUser user@host user2@host

Denygroups group1 group2

Allowgroups group1 group2

Ssh密匙对验证

验证方式控制

口令验证:检查登录用户的口令是否一致

密钥验证

检查客户端私钥与服务器上的公钥是否匹配

配置主机可以无密码登录,可以用root用户

修改配置文件禁止口令登录

47 AuthorizedKeysFile .ssh/authorized\_keys //公钥库(存放授权客户机的公钥文本)

65 PasswordAuthentication yes //检查密钥是否一对

~]# ssh-keygen

关闭PasswordAuthentication yes 改成no

~]# ssh-copy-id root@目地IP地址

Selinux防护:

配置文件://etc/sysconfig/selinux

因为环境原因:

touch /.autorelabel //要创建一个文件

什么是selinus

集成到Linux内核(2.6以上)

扩展安全的功能模块

策略集

SELINUXTYPE=targeted //保护最常见/关键的网络服务.其他不限制

Mls //全部保护

Minimum //最小保护(介于上两个之间)

~]# sestatus //显示当前的selinux的状态

SELinux status: enabled //是否启用

Loaded policy name: targeted //启用模式

Current mode: enforcing //启用状态

查看上下文 (-Z(大写))

ls -ldZ /etc //查看文件本身的安全上下文

~]# ps aux -Z | grep httpd //查看服务的安全上下文

调整布尔值

~]# getsebool -a //查看布尔值

~]# setsebool -P //永久修改布尔值

Lcd //进入客户端的目录(没有的l的话就进入了服务器的目录)

Put //上传

anon\_upload\_enable=YES //允许匿名用户上传

加密,入侵检测,抓包

加密:

数据对称加密与解密 (同一个密钥)

非对称加密与解密 (公钥和私钥)

数子签名与验证签名

保护信息的完整度

常见的加密算法:

对称加密:

DES (data encryption standard)

AES (advanced encryption standard)

非对称加密:

RSA (rivest shamirh adleman)

DSA (digital signature algorithm)

Hash散列技术,用于信息摘要

MD5 ,SHA

原文本不变化,每次校验的值不变,长度不变(反之则变)

输入的文本不同,生成的摘要也不同

~]# sha

sha1sum sha224sum sha256sum sha384sum sha512sum

~]# md5sum

Gpg加密解密工具:(gpnupg)

<https://www.gnupg.org/> //软件网址

~]# which gnupg2 //查找命令在哪一个文件

~]# rpm -qf /usr/ //查看命令是有哪一个包产生的

~]# gpg --version

gpg (GnuPG) 2.0.22

Home: ~/.gnupg

支持的算法：

公钥：RSA, ?, ?, ELG, DSA

对称加密：IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256,

TWOFISH, CAMELLIA128, CAMELLIA192, CAMELLIA256

散列：MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224

压缩：不压缩, ZIP, ZLIB, BZIP2

基本用法:

对称加密/解密

-c, --symmetric 仅使用对称加密

-d, --decrypt 解密数据(默认)

新建两个用户:

启用加密用户

~]# ssh -X xx@localhost //远程连接用户.本机登录

另一台电脑一样

创建一个文件夹测试:

~]$ vim a.txt

~]$ gpg -c a.txt

gpg: 已创建目录‘/home/hh/.gnupg’

gpg: 新的配置文件‘/home/hh/.gnupg/gpg.conf’已建立

gpg: 警告：在‘/home/hh/.gnupg/gpg.conf’里的选项于此次运行期间未被使用

gpg: 钥匙环‘/home/hh/.gnupg/pubring.gpg’已建立

以上说明加密完成

~]$ file b.txt.gpg //查看加密后的文件

~]$ mv b.txt.gpg /tmp //放到公共目录下

再切换到另一个用户查看公共目录下的文件

~]$ gpg -d /tmp/b.txt.gpg > a.txt //解密文件并放到家目录下的文件里

非对称加密:

公钥加密,私钥文件解密

接收方:

生成秘钥对

~]# gpg --gen-key

Gpg -a --export 文件名 ＞　／文件名／　　//把加密的内容导出到开放文件中

gpg --import /文件名　　　　//　再导入公钥

发送方：

Gpg -e -r 文件名　　　　　　　　　//加密

放到公共目录下／ｔｍｐ／

到查看端去

Gpg -d /文件名／　　　＞　文件名　　　　　　　//解密并导入到某个文件里

数字签名：

私钥文件签名，公钥文件验证签名

工作中签名的应用

建立签名文件：建立好后会生成一．Sig结尾的文件

分离式签名

Gpg　-ｂ /文件名

验证软件包签名：（--verify）

~]# gpg --import 文件名　　　　／／导入官方公钥

~]# gpg --verify 文件名 //验证签名

Aide入侵检测

是入侵检测，只能检测，不能防止（yum源就能装）

记录在机器上做了什么操作

~]# yum -y install aide

~]# vim /etc/aide.conf　　　　　　　／／主配置文件

@@define DBDIR /var/lib/aide　　　　　　　　　　　　／／数据库目录

@@define LOGDIR /var/log/aide　　　　　　　　　　　　／／日志文件

database\_out=file:@@{DBDIR}/aide.db.new.gz　　　　　　／／数据库文件名

希望检查的项目：

#p: permissions

#i: inode:

#n: number of links

#u: user

#g: group

#s: size

#b: block count

#m: mtime

#a: atime

#c: ctime

#S: check for growing size

对那些目录进行什么检验：

99 /boot/ CONTENT\_EX

100 /bin/ CONTENT\_EX

101 /sbin/ CONTENT\_EX

102 /lib/ CONTENT\_EX

103 /lib64/ CONTENT\_EX

104 /opt/ CONTENT

加！就是不检测指定的目录

112 !/usr/src/

113 !/usr/tmp/

自定义检测：

~]# sed -i "99,312s/^/#/" /etc/aide.conf　　　　　　　／／注释掉

~]# vim +98 /etc/aide.conf　　　／／自定义检测

99 /root/ FIPSR

初始化检查

~]# aide --init

AIDE, version 0.15.1

AIDE database at /var/lib/aide/aide.db.new.gz initialized.

备份数据库：

Cp /var/lib/aide/aide.db.new.gz /文件名／

随意创建一个文件

执行入侵检测：

cd /var/lib/aide/

mv aide.db.new.gz aide.db.gz　　　（复制是必须要修改文件名不要ｎｅｗ，否则会失败）

aide --check

扫描与抓包

为什么要扫描：

加测潜在的风险，查找可攻击的目标，收集设备／主机／系统等，发现可利用的安全漏洞

扫苗方式和工具：

典型的扫描方式

Scam 主动检测

Sniff 被动监听／嗅探

Capture 数据包捕获（抓包）

扫描器：NMAP

协议分析:tcpdump wireshark

NMAP 是一款强大的网络检测利器工具

支持多种探测技术　（ping 技术　　　多端口扫描　　　TCP/IP指纹协议）

Nmap {扫描类型}　｛选项｝　＜扫描目标＞

常见的类型：

-ＳS TCP SYN 扫描　（半开）

-sT TCP 连接扫描（全开）

-sU UDP扫描

-sP ICMP扫描

-A 目标系统全面分析

检查那些主机开启了那些端口

~]# nmap　－ｓＵ　主机名　　　　　　　／／指定-sU 扫描UDP

检查那些主机开启了FTP,SSH服务段

nmap -p 21-22（服务端口号） 192.168.4.0/24（ＩＰ地址，全网段）

检查目标目标主机的存活状态（是否ｐｉｎｇ通）

~]# nmap -n -sP 192.168.4.0/24

检查操作系统指纹

~]# nmap -A 192.168.4.11

网络抓包工具

Tcpdump抓包命令

基本用法tcpdump [选项]　［过滤条件］

选项：

-i　：指定监控网络借口

-A :转换为ACSLL码　，以方便阅读

-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

-c　：定义抓包个数

过滤条件：

类型：host ,net port portrange

方向: src dst

协议: tcp udp ip wlan arp

多条件组合: and or not

抓到的包放在文件里，以．cap结尾

tcpdump -c 3 -w a.cap　（抓3个包放到a.cap文件里）

Icmp 协议　　ping

~]# tcpdump -i eth0 ip host 目标ＩＰ　　　（抓取ＩＰ　ｐｉｎｇ的包）

Wireshark图形环境：

系统审计，服务安全

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

不会为系统提供保护，只能发现并记录违反策略的人和行为

审计能记录的日志内容：

日期与事件，事件结果

触发事件的用户

所有机制的使用都可以被记录　　ssh链接等

对关键数据文件的修改行为

监控文件的访问

监控系统的调用

记录用户运行的命令

审计可以监控网络访问行为

Ausearch工具

需要安装audit包

主配置文件：/etc/audit/auditd.conf

yum -y install audit

~]# systemctl start auditd　　　　　//启动服务（start）

设置开机自启

/var/log/audit/audit.log　　　　　　//日志文件路径（审计规则）

自定义审计规则（临时）：

audispd auditctl（查看审计规则） auditd

~]# auditctl -s　　　　　　//查看状态

~]# auditctl -ｌ ／／查看规则

~]# auditctl -Ｄ　　　　　//删除所有规则

－ｗ 审计对象

-p　　　审计权限（权限可以是r,w,x,a）

-k　　　审计目录

~]# auditctl -w /etc/passwd -p rwxa -k passwd\_change //定义/etc/passwd/这个目录的审计规则，并输出到passwd\_change文件里

~]# auditctl -l　　　//查看

-w /etc/passwd -p rwxa -k passwd\_change

-w /etc/selinux -p rwxa -k /selin\_change

定义永久的规则：

~]# vim /etc/audit/rules.d/audit.rules　　　//永久规则路径

-w /etc/passwd -p rwxa -k passwd\_change

-w /etc/selinux -p rwxa -k /selin\_change

定义的临时规则粘贴复制到路径内

查看日志：

~]# ausearch -k passwd\_change -i　　//ｋ定义的审计日志的名称　-ｉ　交互式操作

type=EXECVE msg=audit(2019年07月30日 09:46:21.846:25) : argc=3 a0=fdisk a1=-lu a2=/dev/vda

type=SYSCALL msg=audit(2019年07月30日 09:46:21.846:25) : arch=x86\_64 syscall=execve success=yes exit=0 a0=0x113f1b0 a1=0x11496b0 a2=0x1051020 a3=0x7ffcf595e0e0 items=2 ppid=1231 pid=3204 auid=root uid=root gid=root euid=root suid=root fsuid=root egid=root sgid=root fsgid=root tty=pts0 ses=7 comm=fdisk exe=/usr/sbin/fdisk key=hh\_change

argc=3 a0=fdisk a1=-lu a2=/dev/vda　　　　//命令的参数

msg=　　　　　　　　　　　//执行命令的时间

success=yes　　　　　　　//命令是否执行成功

uid=root gid=root　　　　//用户和组

tty=pts0　　　　　　　　　//在哪一个终端执行的命令

comm=fdisk　　　　　　　　//执行的什么命令

exe=/usr/sbin/fdisk　　　　//命令的路径

~]# aureport　　　　//把日志生成报表的形式输出

网站服务：

Nginx服务安全：

源码安装nginx

server {

autoindex on;

启用索引模块

./configure --without-http\_autoindex\_module --without-http\_ssi\_module　//禁用自动索引模块

make install

grep -n autoindex /usr/local/nginx/conf/nginx.conf

sed -i '37s/^/#/' /usr/local/nginx/conf/nginx.conf

grep -n autoindex /usr/local/nginx/conf/nginx.conf

/usr/local/nginx/sbin/nginx -s stop

/usr/local/nginx/sbin/nginx

因为配置文件启用了所以启动服务会报错，所以要加注释

修改源码软件版本：

cd nginx-1.12.2/

cd src/http

vim ngx\_http\_header\_filter\_module.c

49 static u\_char ngx\_http\_server\_string[] = "Server: hh" CRLF;

50 static u\_char ngx\_http\_server\_full\_string[] = "Server: hh" NGINX\_VER CRLF;

51 static u\_char ngx\_http\_server\_build\_string[] = "Server: hh" NGINX\_VER\_BUILD CRLF;

Server后面随意改

./configure　--without-http\_autoindex\_module --without-http\_ssi\_module

make install

/usr/local/nginx/sbin/nginx -s stop

/usr/local/nginx/sbin/nginx

/usr/local/nginx/sbin/nginx -V

限制并发访问量（限制单台客户端的并发链接数量）：

limit\_req\_zone $binary\_remote\_addr zone=one:20m rate=2r/s;

为默认模块，可以减低DDos攻击风险

在配置文件http下添加：

limit\_req\_zone $binary\_remote\_addr zone=one:20m（分出20兆的内存） rate=2r/s;

２秒处理一个，只处理20个请求

server {

limit\_req zone=one burst=10;

在客户端访问

~]$ ab -c 1000 -n 10000 <http://192.168.4.10/>

一兆可以存储８千个ip信息　20兆可以存储16万个ip信息

拒绝非法请求：

防止buffer溢出

防止客户端请求数据溢出，降低Ｄos攻击风险

Client\_body\_buffer\_size 1K;

Client\_header\_buffer\_size 1k;

Client\_max\_body\_size 1k;

Large\_client\_header\_buffers 4 4k;

进程运行时的详细参数信息；

数据库服务的安全预加固

Yum -y install mariadb

~]# systemctl restart mariadb

刚启动时在本机登录没有密码（包括用户）

~]# mysql\_secure\_installation //执行初始化安全脚本

密码安全：

登录状态下：

update mysql.user set password=password("147258")

-> where

-> host="localhost";

更新user表中的password行密码，当主机是localhost时

Set password for root@”localhost” =password(“密码”)

命令行：mysqladmin -uroot -p密码 password “修改的密码”

但是在5.6以后版本中会出现明文密码

Cat .bash\_history　：　日志文件就能看到

5.6以后的版本修复了

Tomcat服务安全

Yum -y install java\*

优化配置：

cd lib/

jar -xf catalina.jar

/usr/local/tamcat/lib/org/apache/catalina／utlil/serverinfo.lass　　　//配置文件路径

16 server.info=hh^M

17 server.number=0.0.0.0^M

修改配置文件，随意写

69 <Connector port="8080" protocol="HTTP/1.1"

70 connectionTimeout="20000"

71 redirectPort="8443" server="nginx" />

Server自己添加，随意写

/usr/local/tamcat/bin/shutdown.sh

ss -antulp | grep 8080

/usr/local/tamcat/bin/startup.sh

ss -antulp | grep 8080

firefox http://192.168.4.10:8080/

121 curl -I http://192.168.4.10:8080/a1.html

测试

降权运行：

useradd tomcat

chown -R tomcat /usr/local/tamcat/

su -c /usr/local/tamcat/bin/startup.sh tomcat　　//把root降级运行这个服务

~]# rm -rf /usr/local/tamcat/webapps/\*

自己定义页面内容

安全打补丁

比较差异，有不同的则更新不同的部分

编辑两个内容不同的两个ｓｈｅｌｌ脚本，

使用diff进行比较

选项

－ｕ 输出同意内容的头部信息（打补丁使用）

-r 递归对比目录中的所有资源（包括目录）

-a 所有文件视为文本（包括二进值）

-N　无文件视为空文件

@@ -1,2 +1,3 @@

#!/bin/bash

echo "hello world"

+echo "xx"

　最后要添加的行会有＋号，不同的行

对比目录中的文件：

~]# diff -r hh xx

只在 hh 存在：ls

只在 hh 存在：opreation

打补丁

Patch -P数字（删除重复的个数）　＜

Iptables防火墙，filter表控制

一道保护性的安全隔离

RHEL7默认使用firewalld作为防火墙

但firewalld底层还是调用包过滤防火墙iptables　　（网络层，ip协议）

防火墙服务：

~]# systemctl status firewalld

yum -y install iptables-services.x86\_64

systemctl restart iptables.service

systemctl enable iptables.service

Iptables的组成，四张表，五条链

表是iptables,服务分类＞

Rew : 对ip包做状态跟踪

Mangle :对到达防火墙的ip包打标记

Nat:地址转换，端口转换

Filter:过滤表

链是匹配ip包传输的方向

Imput　进入防火墙主机的包

Output 从防火墙出去的包

Forward 从防火墙经过的包

Postrouting 路由后处理

Prerouting 路由前处理

Iptables [-t 表名]　选项　［链名］　条件　［－j　目标操作］

选项

-Ａ :在链末尾追加一条规则

-I:在链的开头添加，插入一条规则

-L :列出所有的规则条目

-n :以数字的形式查看规则，

--line-numbers ：查看规则时，显示规则的序号

-D :删除链内指定的序号

-F　：清空所有的规则

-P : 为指定的链设置默认规则

-p:协议名

目标操作：

ACCEPT:允许通过／放行

DROP：直接丢弃，不给出任何的回应

REJECT：拒绝通过，必要时会给提示

LOG：记录日志，然后传送给下一条规则

自定义防火墙

主机型：写防火墙规则保护本机

网络型：写防火墙规则限制数据包的进出

匹配的条件：

通用匹配：

-p 协议

-s　　　　原地址

-d　　　　目标地址

-i　　　　收数据的网卡

-o　　　　发数据的网卡

隐含匹配

-sport 源端口

--dport 目标端口

ICMP类型匹配：

--icmp-type

~]# iptables -t filter -A IMPUT -p tcp --dport -j ACCEPT

只允许ssh链接，ping也不行

网络防火墙:

~]# iptables-save > /etc/sysconfig/iptables //保存防火墙规则到某个文件中

~]# sysctl -a //列出所有的内核参数

~]# sysctl -a | grep ipv4.ip\_forward //查看是否开启内核的路由转发功能

若没有:

~]# echo "net.ipv4.ip\_forward = 1" >> /etc/sysctl.conf

~]# echo 1 > /proc/sys/net/ipv4/ip\_forward

两种方法,随意

网关:

~]# route -n //查看网关

~]# route add(del) default gw 192.168.2.11 //添加网关(删除)

主机

yum -y install httpd

systemctl start httpd

echo "123456" > /var/www/html/a.html

在第三台机器上:

~]# curl <http://192.168.4.10/a.html>

会连接不上,没有防火墙规则

创建防火墙规则:

~]# iptables -t filter -A FORWARD -p tcp --dport 80 -j ACCEPT

请求的服务

~]# iptables -t filter -A FORWARD -p tcp --sport 80 -j ACCEPT

回应的服务

定义ssh服务

~]# iptables -t filter -A FORWARD -p tcp --dport 22 -j ACCEPT

~]# iptables -t filter -A FORWARD -p tcp --sport 22 -j ACCEPT

~]# iptables-save > /etc/sysconfig/iptables

扩展匹配:

前提:有对应的防火墙模块

用法

-m 扩展模块 --扩展模块 条件值

Mac地址匹配:

Mac --mac-source mac地址列表

多端口匹配:

Multiport --sport 源端口列表

Multiport --dport 目标端口列表

IP范围匹配:

Iprange --src-range ip1-ip2

Iprange --dst-range ip1-ip2

以上四个端口合成两个:

请求端口,

~]# iptables -t filter -A FORWARD -p tcp -m multiport --dport 22,80 -j ACCEPT

回应目标端口,

~]# iptables -t filter -A FORWARD -p tcp -m multiport --sport 22,80 -j ACCEPT

开放ping协议端口:

~]# iptables -t filter -A FORWARD -p ICMP -j ACCEPT

给防火墙设置起始IP和结束IP:

~]# iptables -t filter -I FORWARD 3 -p tcp -m iprange --src-range 192.168.2.100-192.168.2.200 -j REJECT

删除防火墙规则里面的FORWARD 行,第三行

~]# iptables -t filter -D FORWARD 3

禁止掉一个主机的mac地址,ping不通:

~]# arp -n //查看mac地址

~]# iptables -t filter -I FORWARD 3 -p tcp -m --mac-source mac地址 -j DROP

NAT转换:(适用于路由器,固定的IP)

NAT表做数据包中的原地址转换,事先所有主机共享一个公网IP地址上网

修改数据包的原地址,仅用于net表的POSTROUTING链

~]# iptables -t nat -nL //查询nat表中的链路

配置防火墙策略:

~]# iptables -t nat -A POSTROUTING -s 192.168.4.0/24(局域网网段地址) -p tcp(外网借口)--dport 80 -j SNAT() --to-source 192.168.2.100(外网借口的IP地址)

公网地址是服务商分配动态IP时:

定义防火墙的规则:

~]# iptables -t nat -A POSTROUTING -s 192.168.4.0/24(局域网网段地址) -o eth1 -j

MASQUERADE --to-source 192.168.2.100(外网借口的IP地址)