# 数据库平滑扩容

## 学习目标

目标1：理解传统扩容实现方案

目标2：理解平滑扩容双写方案

目标3：掌握数据库2N扩容方案

目标4：实现数据库双主同步

目标5：掌握ShardingJDBC路由以及动态扩容技术

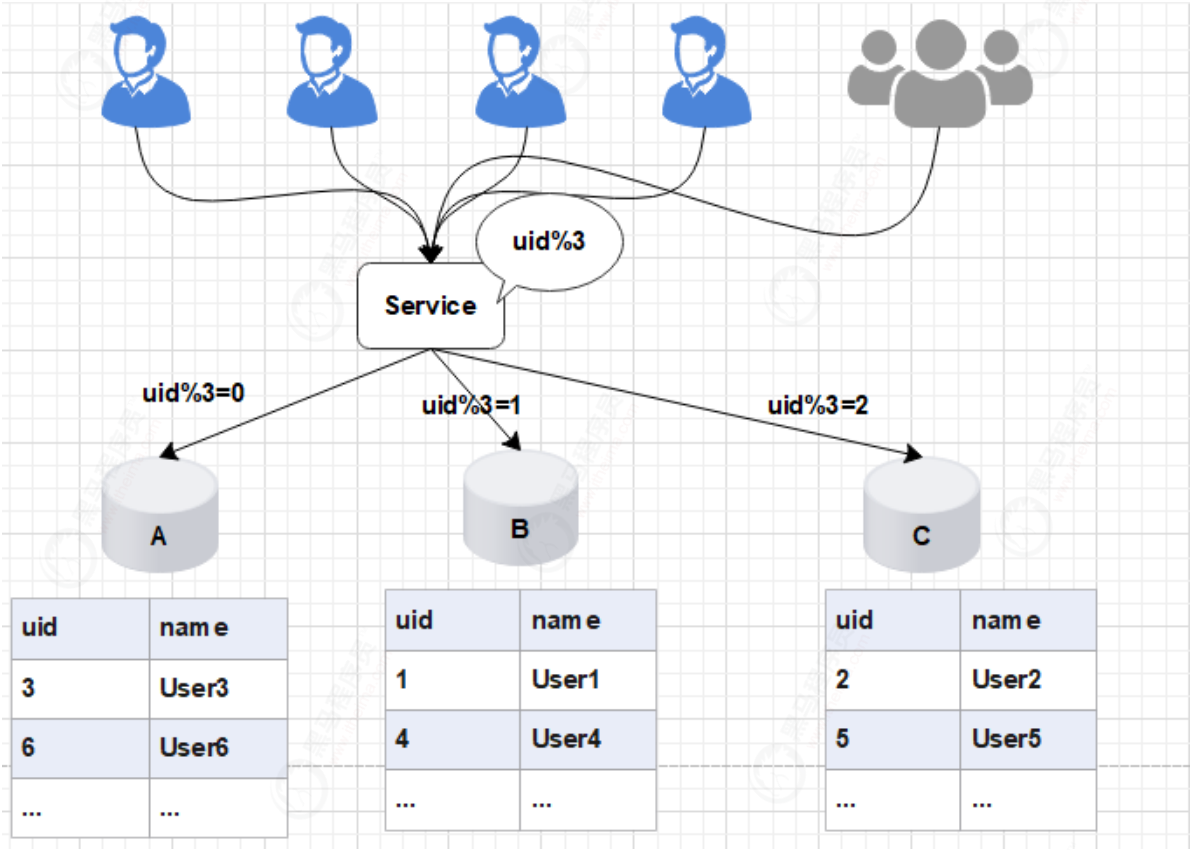
目标6：掌握KeepAlived+MariaDB数据库高可用方案

## 1. 扩容方案剖析

### 1.1 扩容问题

在项目初期，我们部署了三个数据库A、B、C，此时数据库的规模可以满足我们的业务需求。为了将数据做到平均分配，我们在Service服务层使用uid%3进行取模分片，从而将数据平均分配到三个数据库中。

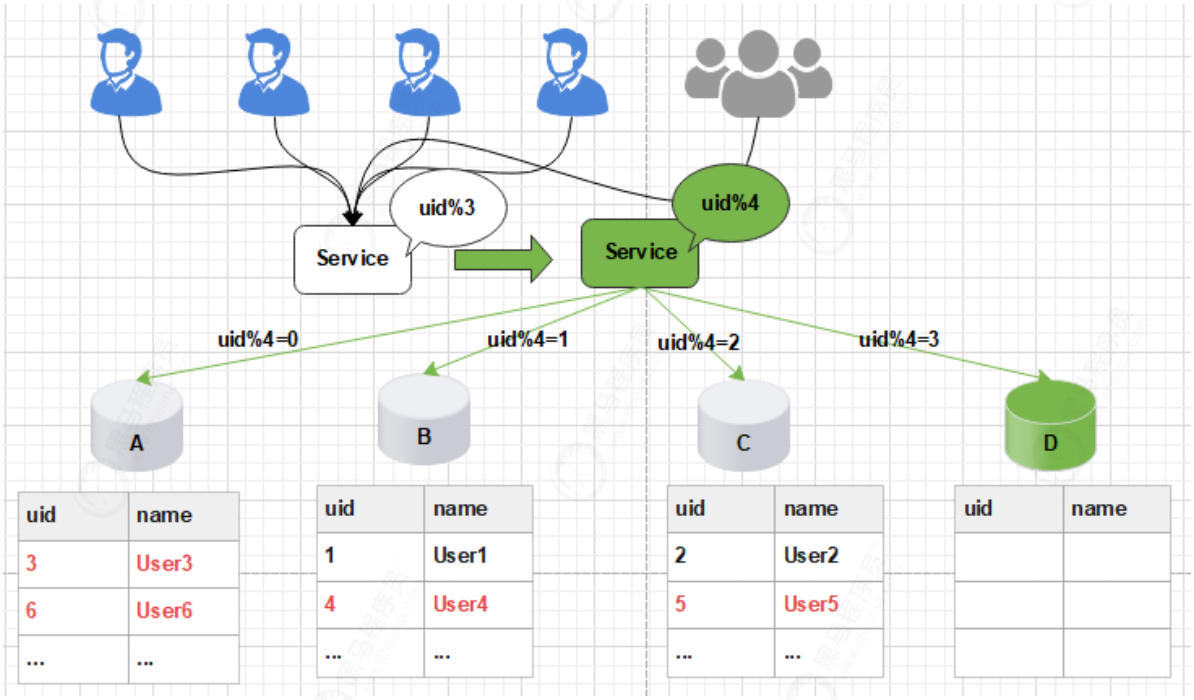
如图所示：




后期随着用户量的增加，用户产生的数据信息被源源不断的添加到数据库中，最终达到数据库的最佳存储容量。如果此时继续向数据库中新增数据，会导致数据库的CRUD等基本操作变慢，进而影响整个服务的响应速度。

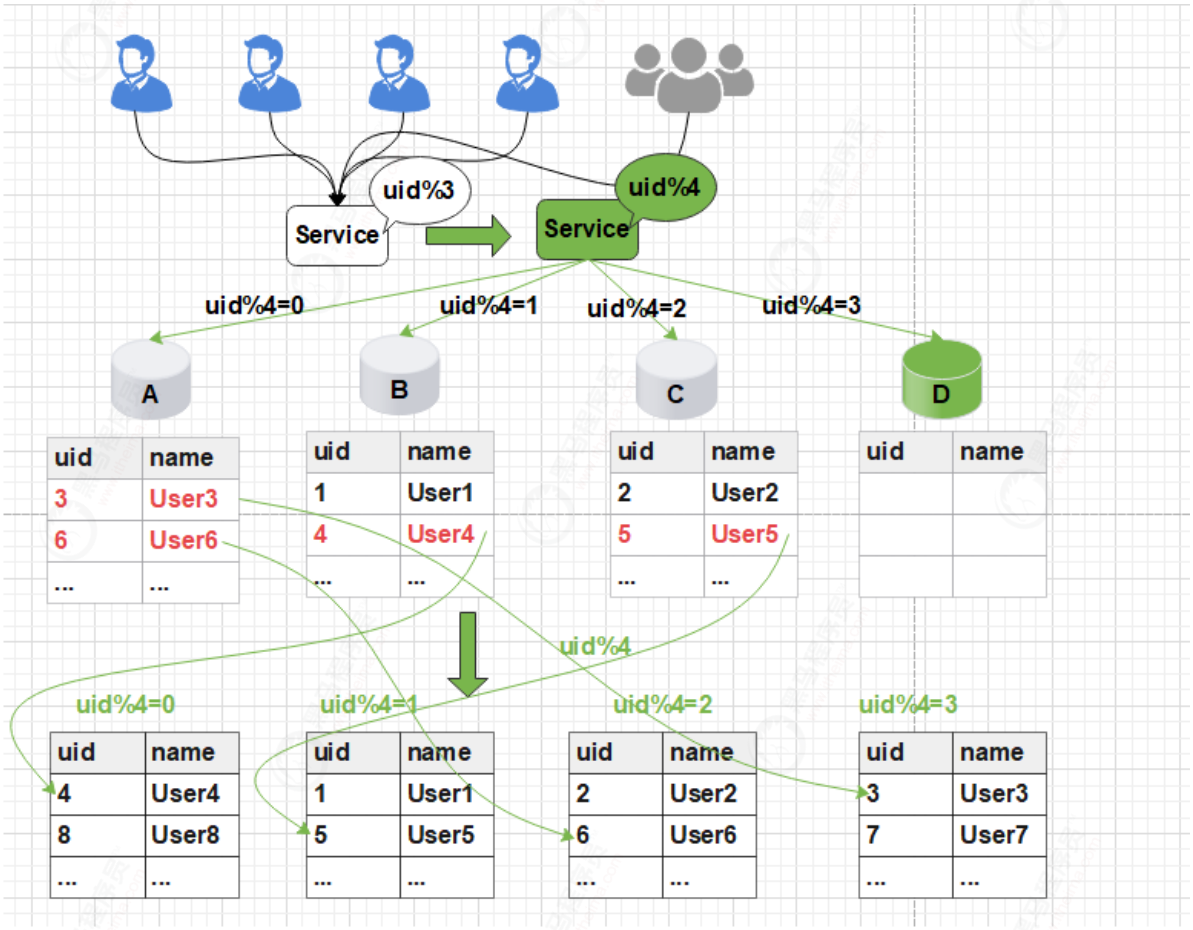
这时，我们需要增加新的节点，对数据库进行水平扩容，那么加入新的数据库D后，数据库的规模由原来的3个变为4个。

如图所示：



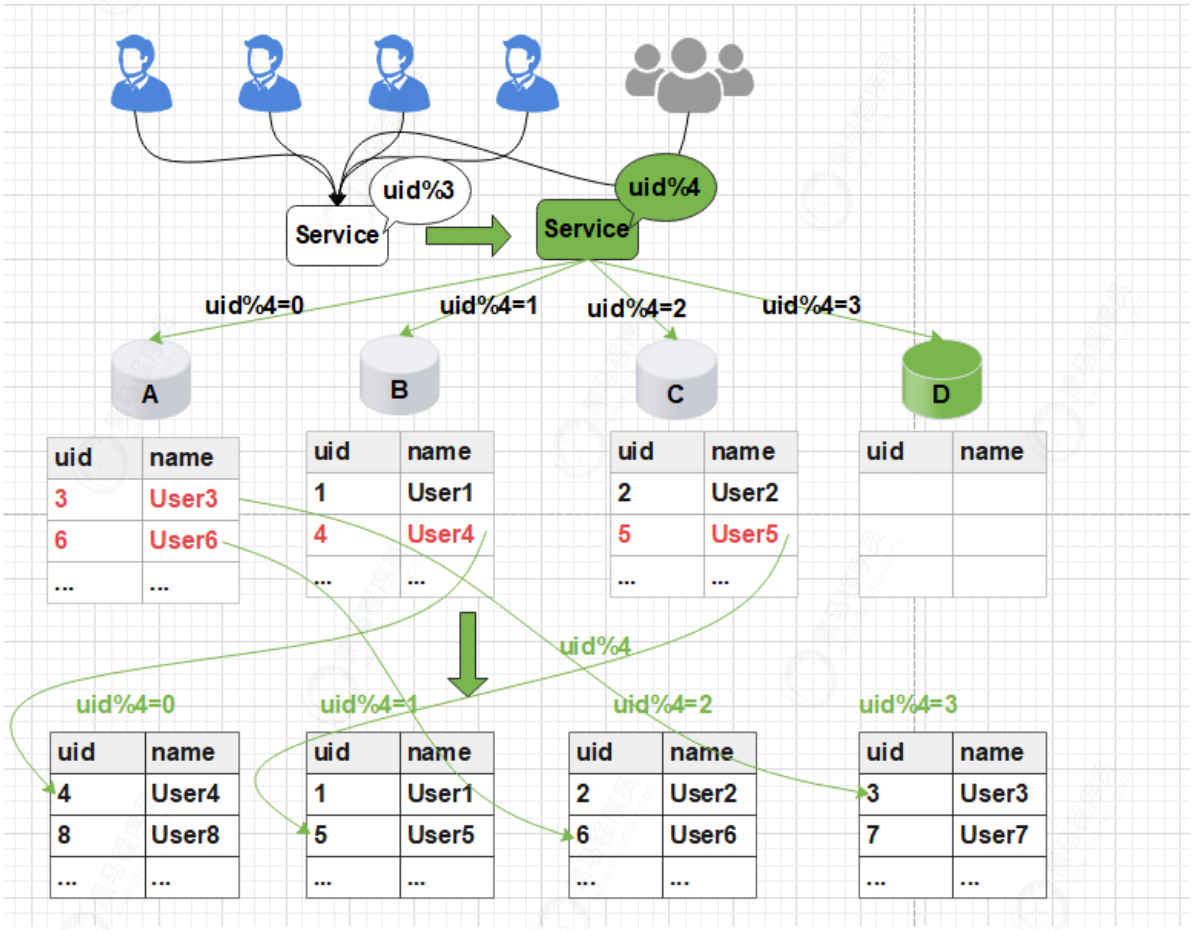

此时由于分片规则发生了变化（uid%3 变为uid%4），导致大部分的数据，无法命中原有的数据，需要重新进行分配，要做大量的数据迁移处理。

> 比如之前uid如果是uid=3取模3%3=0， 是分配在A库上，新加入D库后， uid=3取模3%4=3，分配在D库上；



新增一个节点， 大概会有90%的数据需要迁移， 这样会面临大量的数据压力，并且对服务造成极大的不稳定性。

## 1.2 停机方案




1. 发布公告

   为了进行数据的重新拆分，在停止服务之前，我们需要提前通知用户，比如：我们的服务会在yyyy-MM-dd进行升级，给您带来的不便敬请谅解。

2. 停止服务

   关闭Service

3. 离线数据迁移（拆分，重新分配数据）

   将旧库中的数据按照Service层的算法，将数据拆分，重新分配数据

4. 数据校验

   开发定制一个程序对旧库和新库中的数据进行校验，比对

5. 更改配置

   修改Service层的配置算法，也就是将原来的uid%3变为uid%4

6. 恢复服务

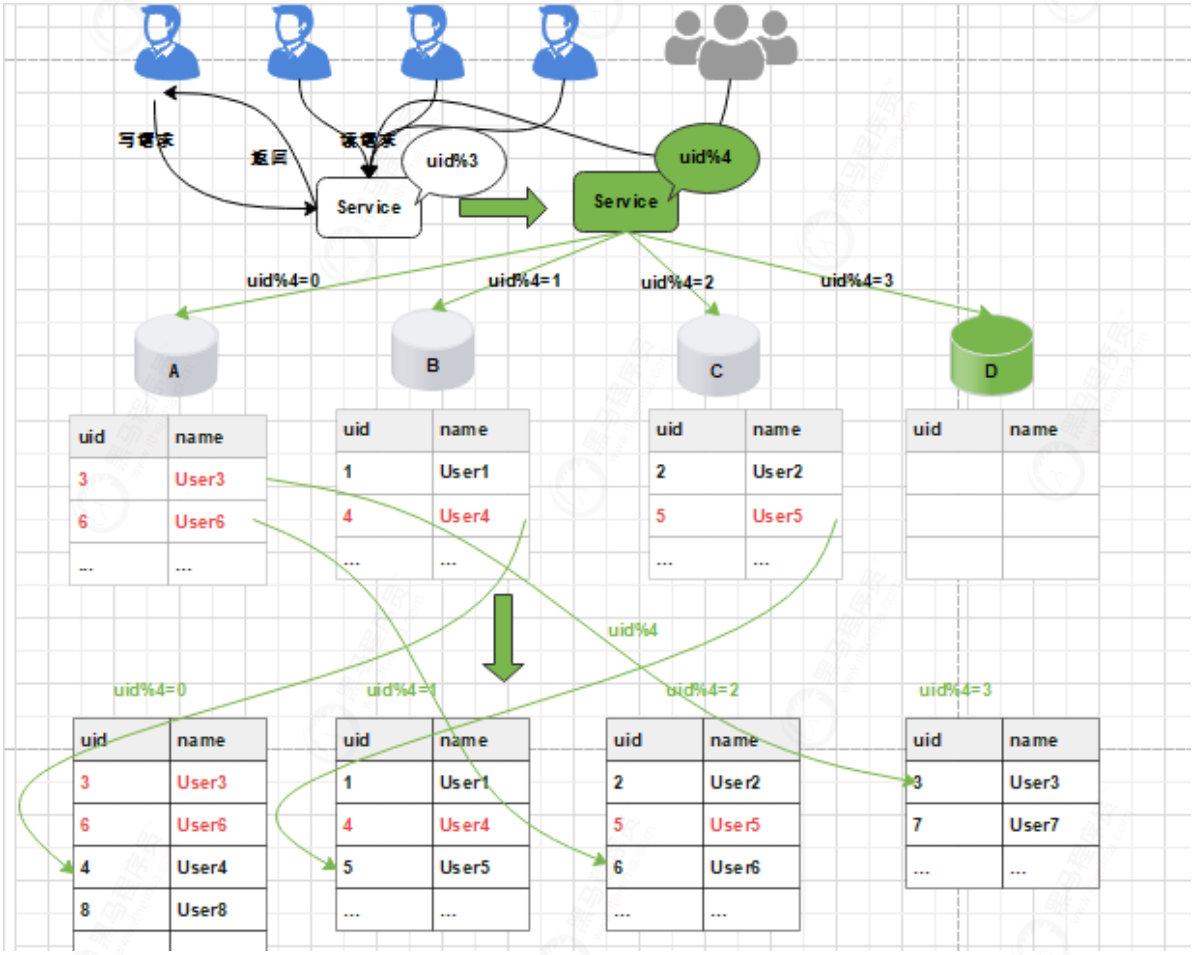
   重启Service服务

7. 回滚预案

   针对上述的每个步骤都要有数据回滚预案，一旦某个环节（如：数据迁移，恢复服务等）执行失败，立刻进行回滚，重新再来

停止服务之后， 能够保证迁移工作的正常进行， 但是服务停止，伤害用户体验， 并造成了时间压力，必须在指定的时间内完成迁移。

## 1.3 停写方案




1. 支持读写分离

   数据库支持读写分离，在扩容之前，每个数据库都提供了读写功能，数据重新分配的过程中，将每个数据库设置为只读状态，关闭写的功能

2. 升级公告

   为了进行数据的重新拆分，在停写之前，我们需要提前通知用户，比如：我们的服务会在yyyy-MM-dd进行升级，给您带来的不便敬请谅解。

3. 中断写操作，隔离写数据源（或拦截返回统一提示）

   在Service层对所有的写请求进行拦截，统一返回提示信息，如：服务正在升级中，只对外提供读服务

4. 数据同步处理

   将旧库中的数据按照Service层的算法，将数据重新分配，迁移（复制数据）

5. 数据校验

   开发定制一个程序对旧库中的数据进行备份，使用备份的数据和重新分配后的数据进行校验，比对

6. 更改配置

   通过配置中心，修改Service层的配置算法，也就是将原来的uid%3变为uid%4，这个过程不需要重启服务

7. 恢复写操作

   设置数据库恢复读写功能，去除Service层的拦截提示

8. 数据清理

使用delete语句对冗余数据进行删除

9. 回滚预案

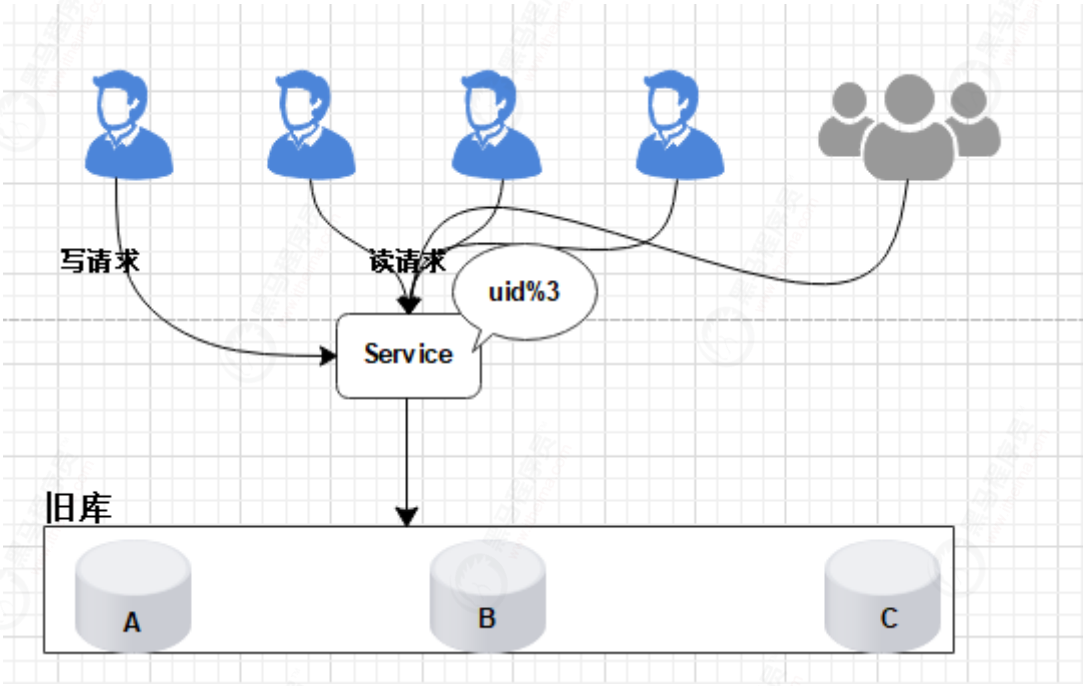
   针对上述的每个步骤都要有数据回滚预案，一旦某个环节（如：数据迁移等）执行失败，立刻进行回滚，重新再来

缺点：在数据的复制过程需要消耗大量的时间，停写时间太长，数据需要先复制，再清理冗余数据

## 1.4 日志方案

核心是通过日志进行数据库的同步迁移， 主要操作步骤如下：

1. 数据迁移之前， 业务应用访问旧的数据库节点。




2. 日志记录

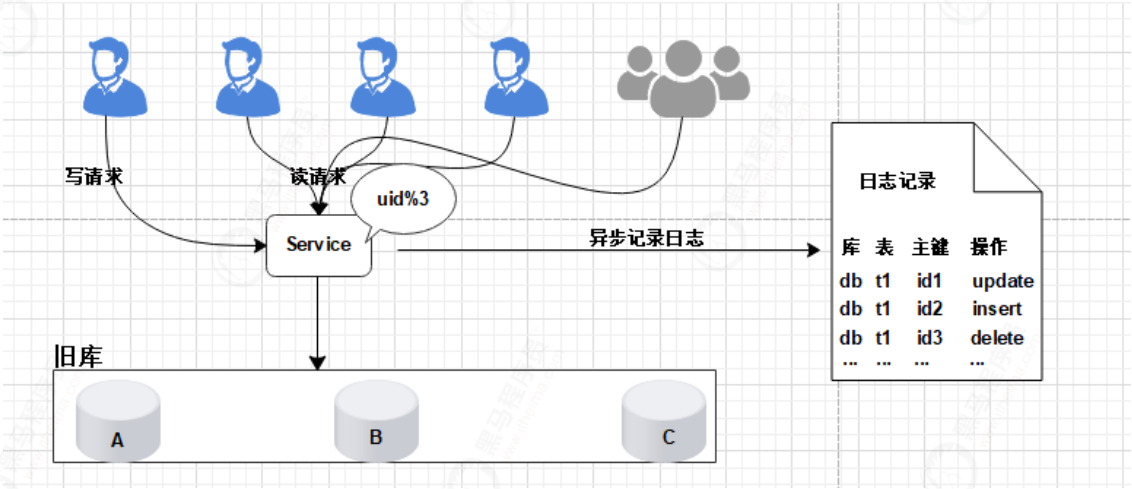
   在升级之前， 记录“对旧数据库上的数据修改”的日志（这里修改包括增、删、改），这个日志不需要记录详细的数据信息，主要记录：

   （1）修改的库；

   （2）修改的表；

   （3）修改的唯一主键；

   （4）修改操作类型。



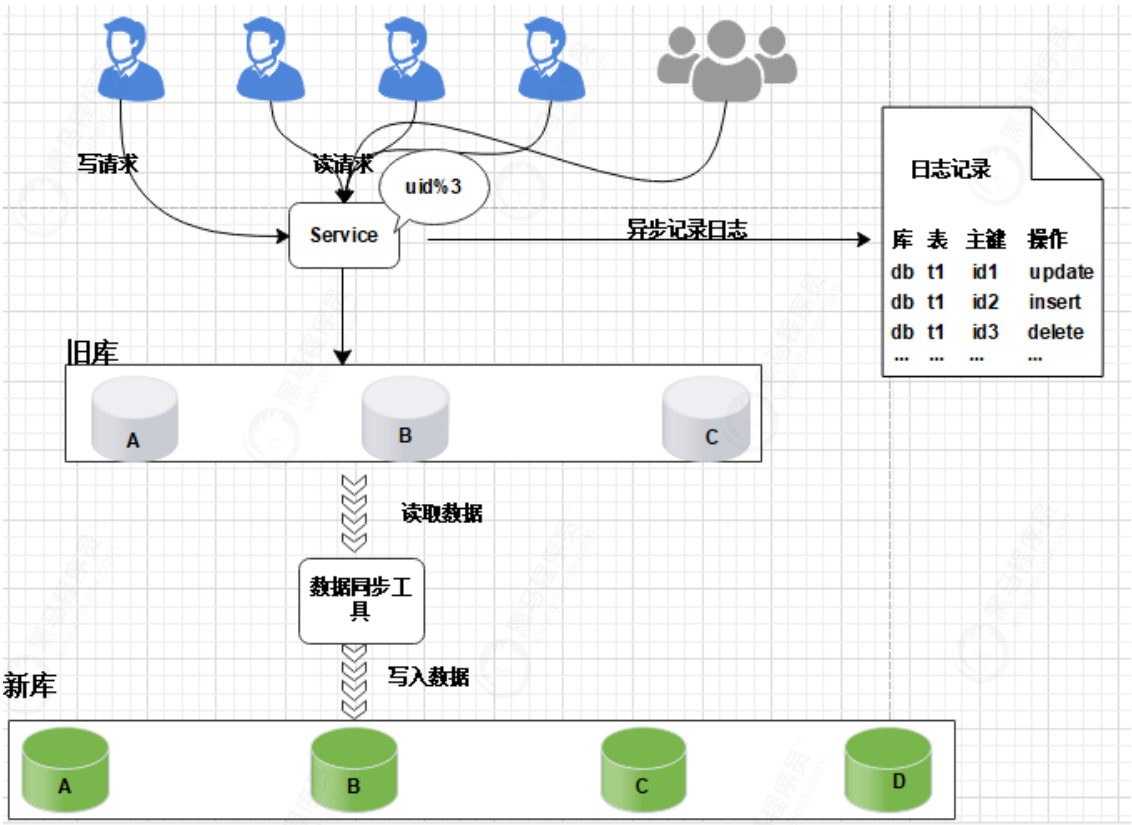
日志记录不用关注新增了哪些信息，修改的数据格式，只需要记录以上数据信息，这样日志格式是固定的， 这样能保证方案的通用性。

服务升级日志记录功能风险较小：

> 写和修改接口是少数， 改动点少；
>
> 升级只是增加了一些日志，采用异步方式实现， 对业务功能没有太多影响。

3. 数据迁移：

研发定制数据迁移工具， 作用是把旧库中的数据迁移至新库中。

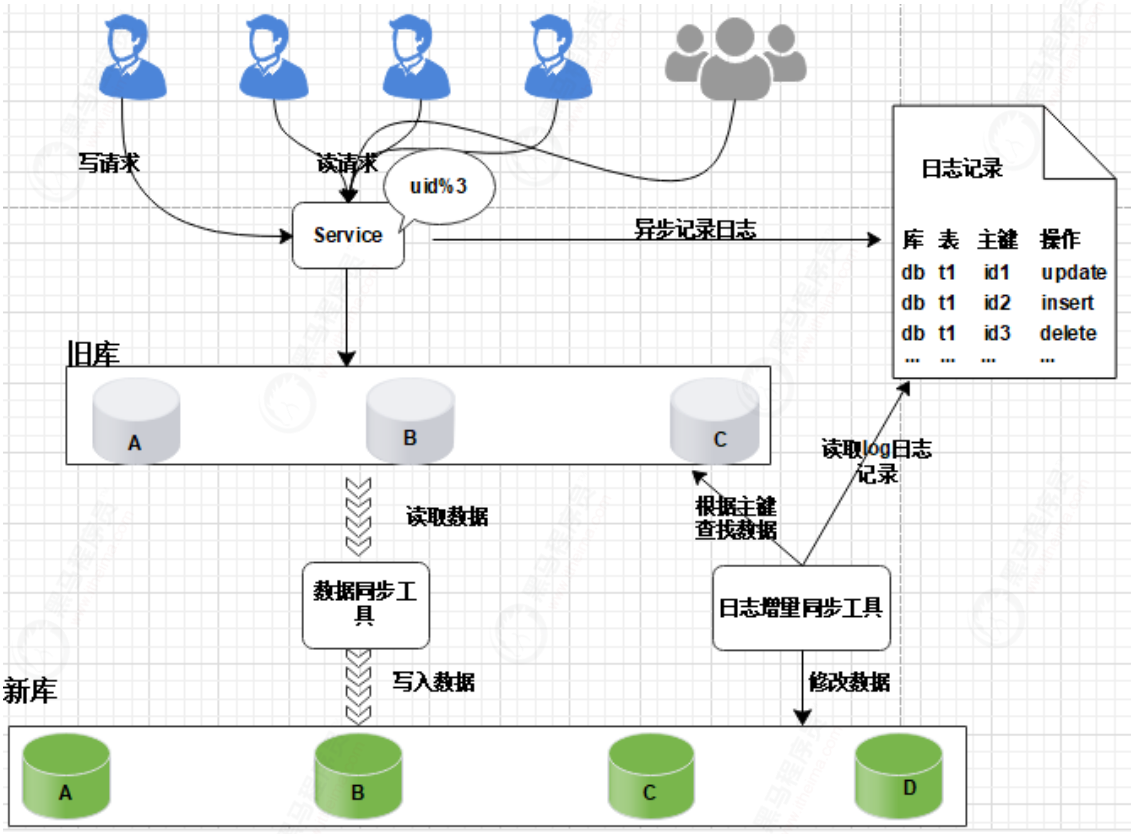



> 整个过程仍然采用旧库进行对外服务。
>
> 数据同步工具实现复杂度不高。
>
> 只对旧库进行读取操作， 如果同步出现问题， 都可以对新库进行回滚操作。
>
> 可以限速或分批迁移执行， 不会有时间压力。

数据迁移完成之后， 并不能切换至新库提供服务。

因为旧库依然对线上提供服务， 库中的数据随时会发生变化， 但这些变化的数据并没有同步到新库中， 旧库和新库数据不一致， 所以不能直接进行切换， 需要将数据同步完整。

4. 日志增量迁移




研发一个日志迁移工具，把上面迁移数据过程中的差异数据追平，处理步骤：

> 读取log日志，获取具体是哪个库、表和主键发生了变化修改；
>
> 把旧库中的主键记录读取出来
>
> 根据主键ID，把新库中的记录替换掉

这样可以最大程度的保障数据的一致性。风险分析：

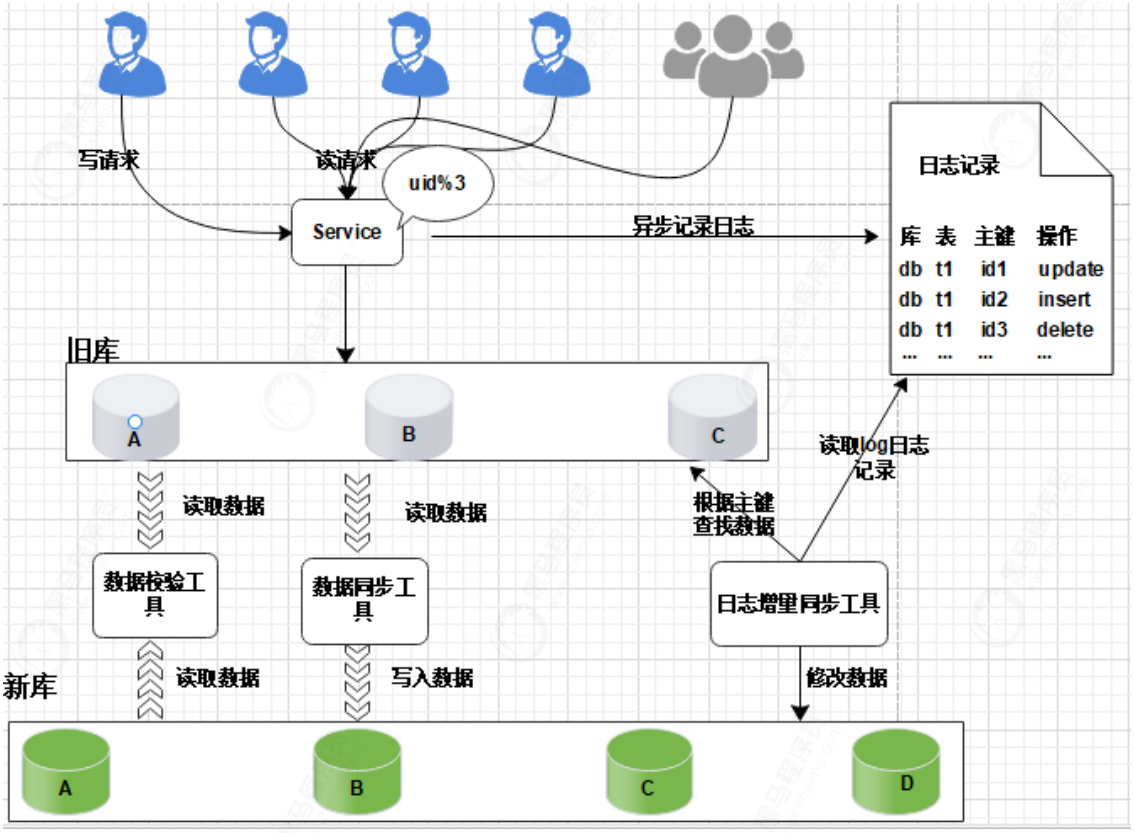
> 整个过程， 仍然是旧库对线上提供服务；
>
> 日志迁移工具实现的复杂度较低；
>
> 任何时间发现问题， 可以重新再来，有充分的容错空间；
>
> 可以限速重放处理日志， 处理过程不会因为对线上影响造成时间压力。

但是， 日志增量同步完成之后， 还不能切换到新的数据库。

因为日志增量同步过程中，旧库中可能有数据发生变化， 导致数据不一致，所以需要进一步读取日志， 追平数据记录； 日志增量同步过程随时可能会产生新的数据， 新库与旧库的数据追平也会是一个无限逼近的过程。

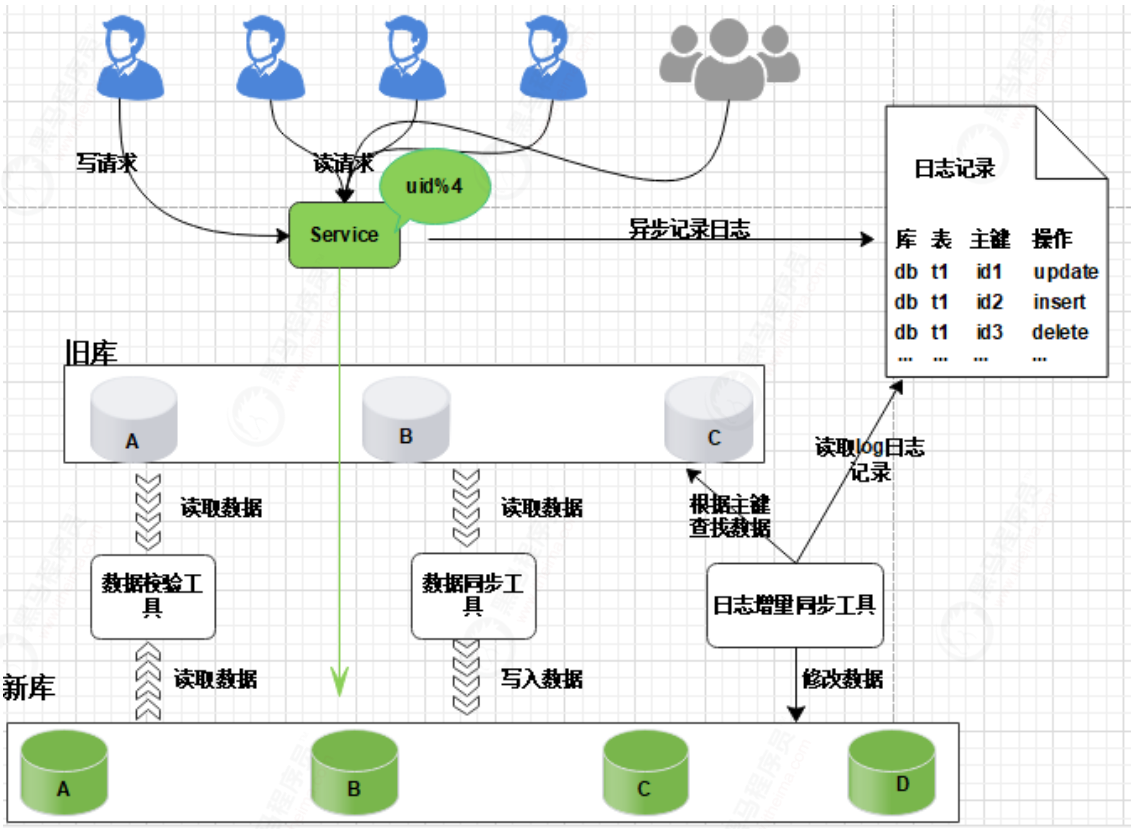
5. 数据校验

   准备好数据校验工具，将旧库和新库中的数据进行比对，直到数据完全一致。




6. 切换新库

数据比对完成之后， 将流量转移切换至新库， 至此新库提供服务， 完成迁移。

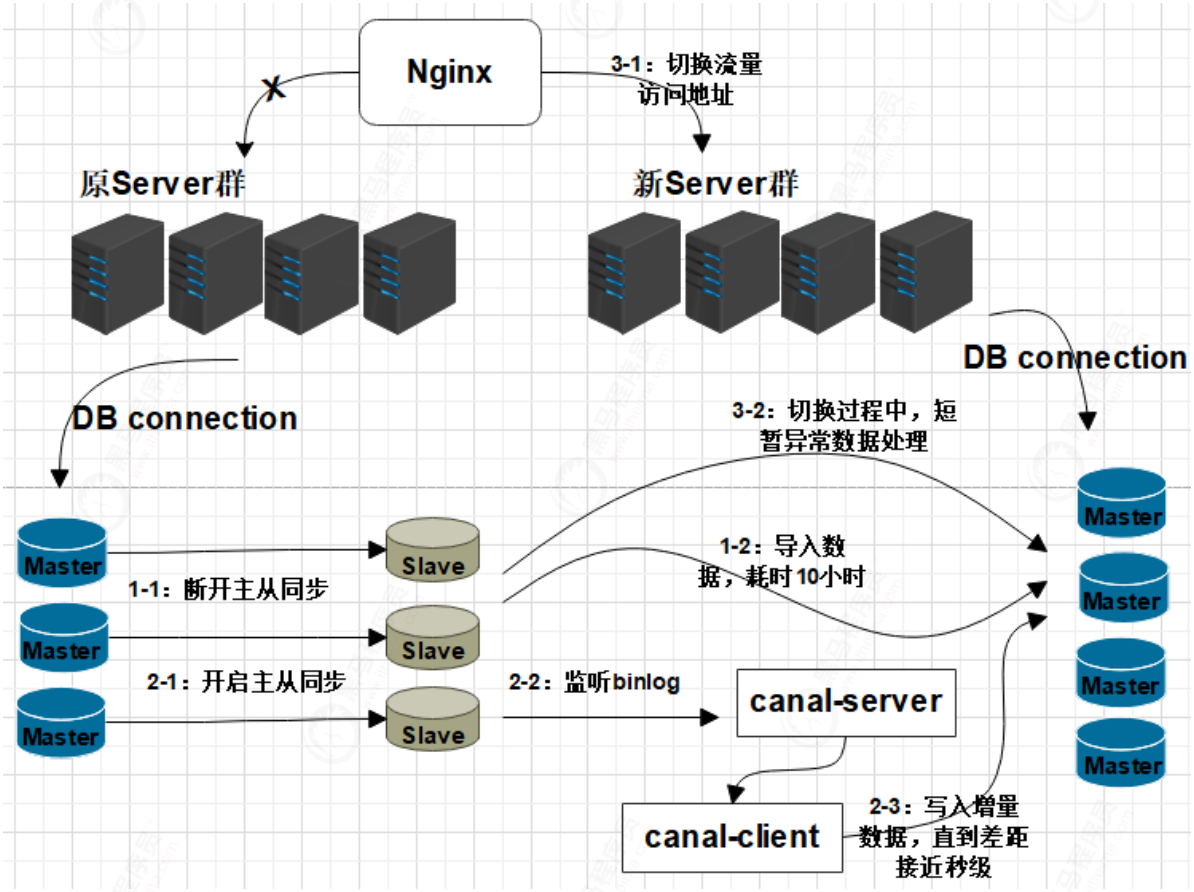



但是在极限情况下， 即便通过上面的数据校验处理， 也有可能出现99.99%数据一致， 不能保障完全一致，这个时候可以在旧库做一个readonly只读功能， 或者将流量屏蔽降级，等待日志增量同步工具完全追平后， 再进行新库的切换。

至此，完成日志方案的迁移扩容处理， 整个过程能够持续对线上提供服务， 只会短暂的影响服务的可用性。

这种方案的弊端，是操作繁琐，需要适配多个同步处理工具，成本较高， 需要制定个性化业务的同步处理， 不具备普遍性，耗费的时间周期也较长。

## 1.5 双写方案（中小型数据）



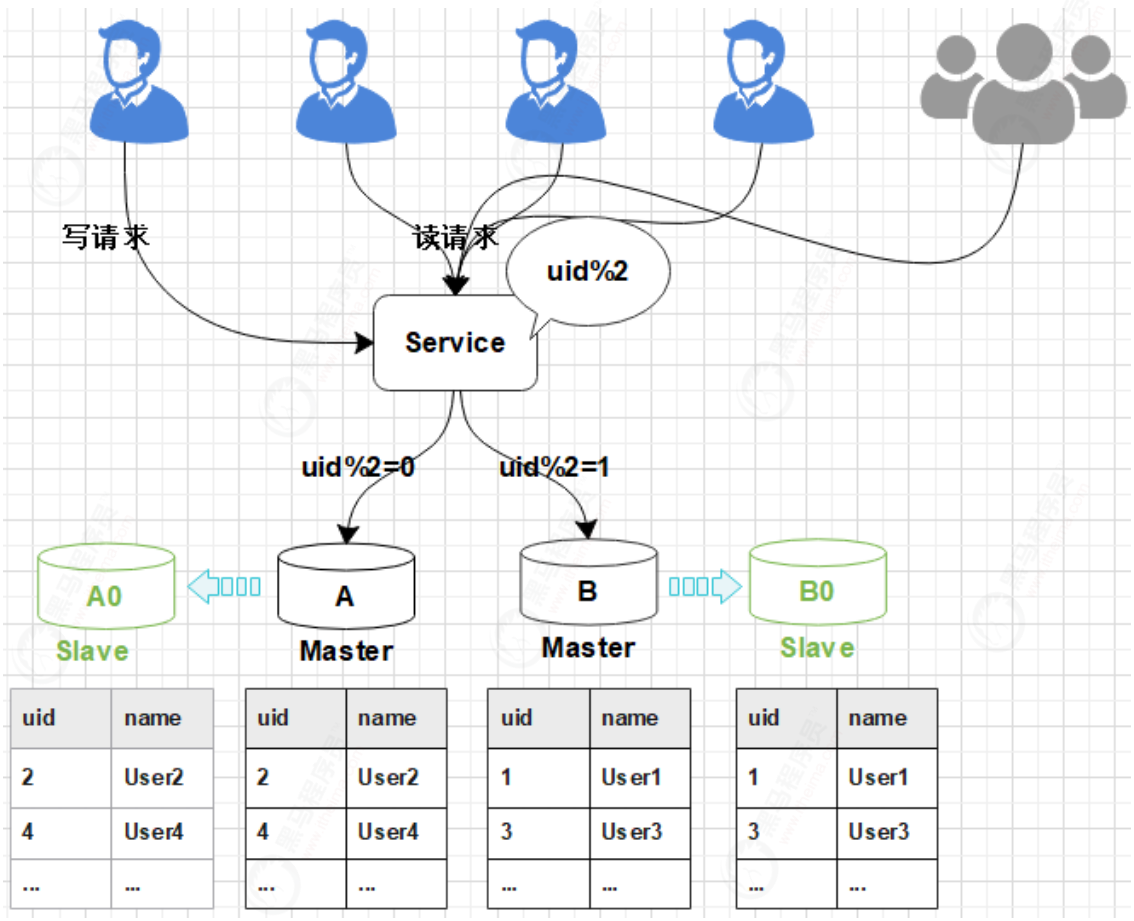

双写方案可通过canal或mq做实现。

1. 增加新库，按照现有节点， 增加对应的数量。
2. 数据迁移：避免增量影响， 先断开主从，再导入（耗时较长）， 同步完成并做校验
3. 增量同步：开启Canal同步服务， 监听从节点数据库， 再开启主从同步，从节点收到数据后会通过Canal服务， 传递至新的DB节点。
4. 切换新库：通过Nginx，切换访问流量至新的服务。
5. 修复切换异常数据：在切换过程中， 如果出现，Canal未同步，但已切换至新库的请求（比如下单，修改了资金， 但还未同步 ）， 可以通过定制程序， 读取检测异常日志，做自动修复或人工处理。

   针对此种情况， 最好是在凌晨用户量小的时候， 或专门停止外网访问，进行切换，减少异常数据的产生。
6. 数据校验：为保障数据的完全一致， 有必要的数据的数量完整性做校验。

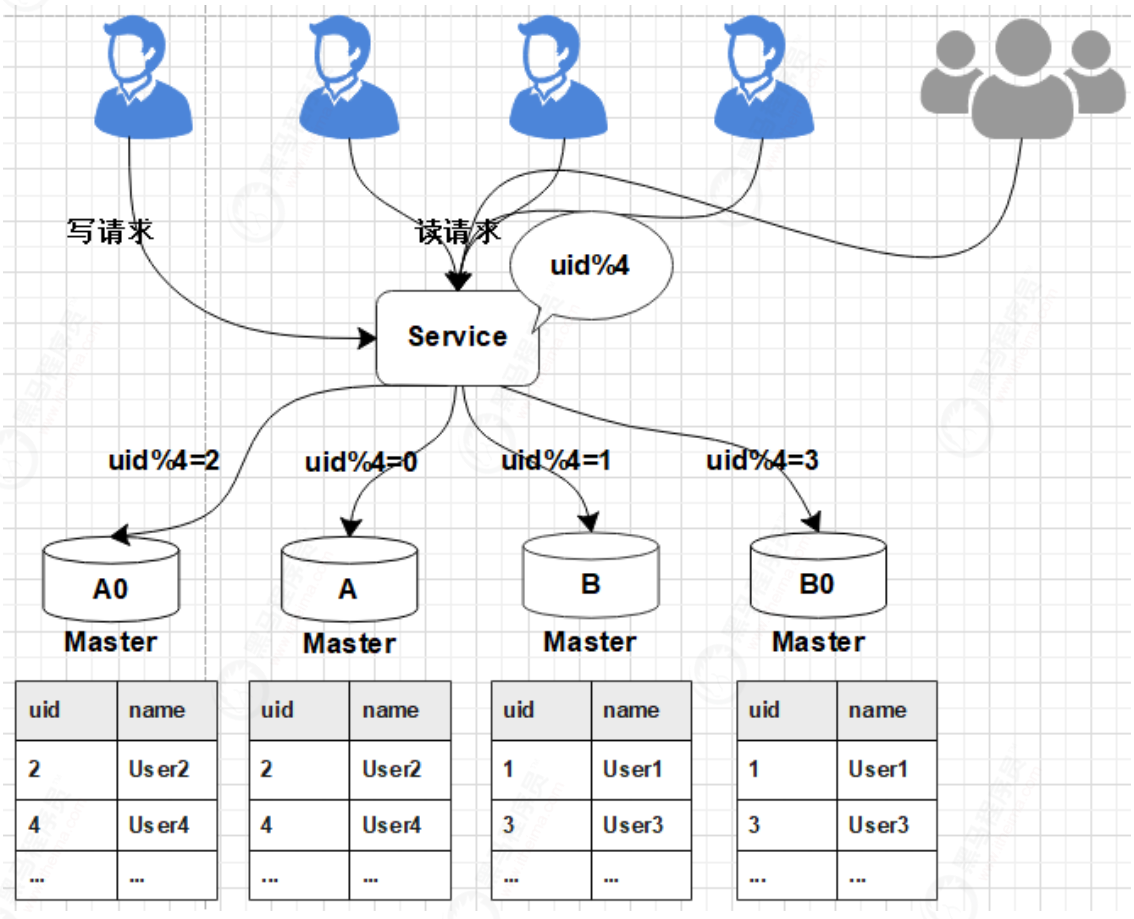
## 1.6 平滑2N方案（大数据量）

1. 线上数据库，为了保障其高可用，一般每台主库会配置一台从库，主库负责读写，从库负责读取。下图所示，A，B是主库，A0和B0是从库。

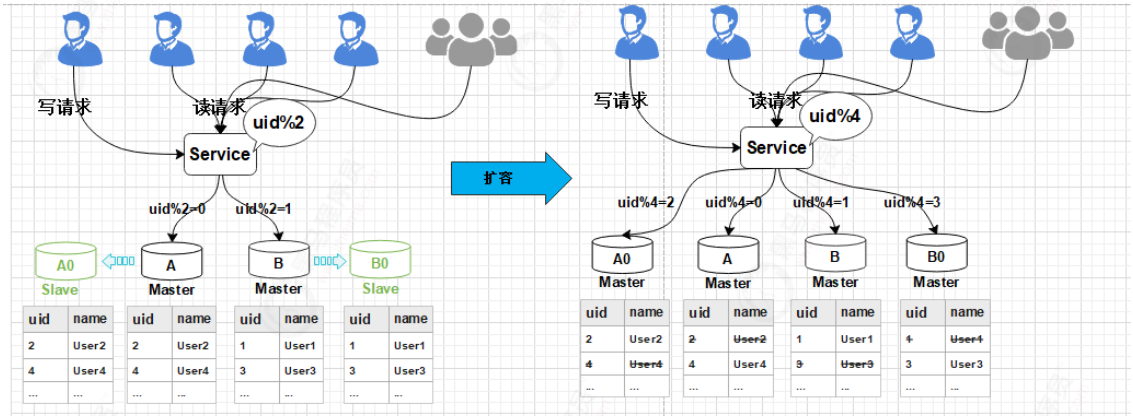



2. 当需要扩容的时候，我们把A0和B0升级为新的主库节点，如此由2个分库变为4个分库。同时在上层的分片配置，做好映射，规则如下:

> 把uid%4=0和uid%4=2的数据分别分配到A和A0主库中
>
> 把uid%4=1和uid%4=3的数据分配到B和B0主库中



3. 因为A和A0库的数据相同，B和B0数据相同，此时无需做数据迁移。只需调整变更一下分片配置即可，通过配置中心更新，不需要重启。

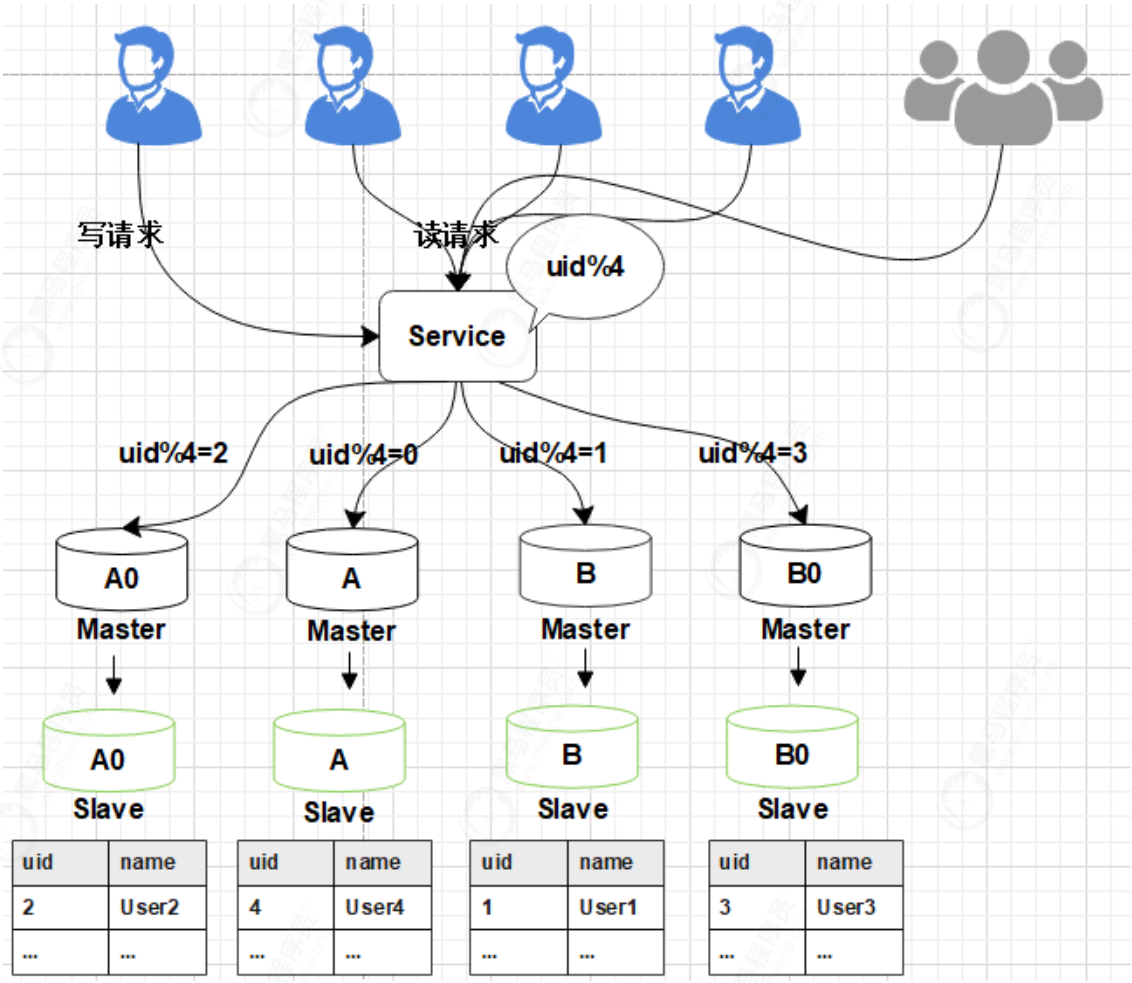



由于之前uid%2的数据是分配在2个库里面，扩容之后需要分布到4个库中，但由于旧数据仍存在（uid%4=0的节点,还有一半uid%4=2的数据），所以需要对冗余数据做一次清理。

这个清理，并不会影响线上数据的一致性，可以随时随地进行。

4. 处理完成之，为保证数据的高可用，以及将来下一步的扩容需求。

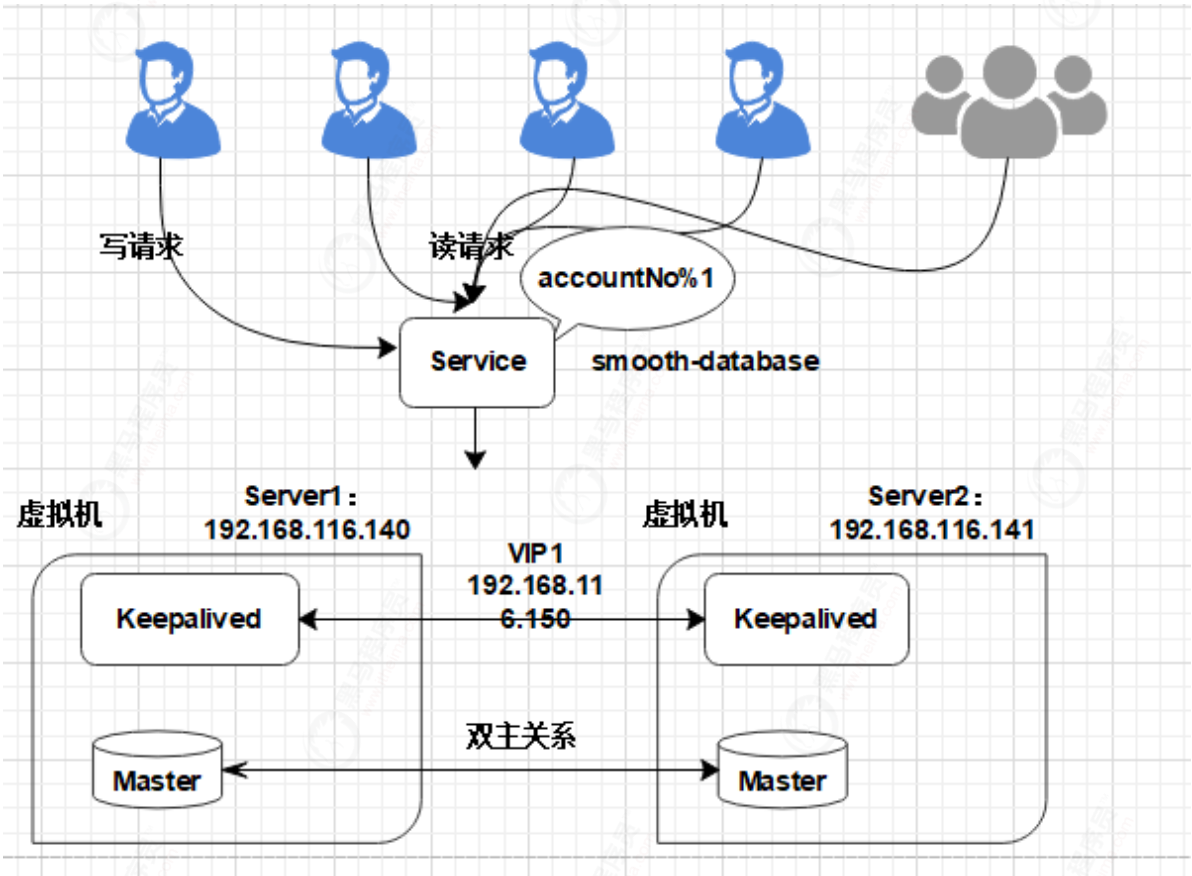
可以为现有的主库再次分配一个从库。




## 2. 平滑2N扩容方案实践

## 2.1 实现应用服务级别的动态扩容

扩容前部署架构：




虚拟路由器

| 虚拟机 | state | virtual_router_id | priority | virtual_ipaddress | real_server | notify_down |
|---|---|---|---|---|---|---|
| Server1 | BACKUP | 111 | 100 | 192.168.116.150 | 192.168.116.140 | mariadb.sh |
| Server2 | BACKUP | 111 | 98 | 192.168.116.150 | 92.168.116.141 | mariadb.sh |

### 2.1.1 MariaDB服务安装

1. 切换阿里云镜像服务（YUM安装过慢可以切换）

```
yum -y install wget
mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.bak
wget -O /etc/yum.repos.d/CentOS-Base.repo
http://mirrors.aliyun.com/repo/Centos-7.repo
wget -P /etc/yum.repos.d/ http://mirrors.aliyun.com/repo/epel-7.repo
yum clean all
yum makecache
```

2. 配置YUM源

```
vi /etc/yum.repos.d/mariadb-10.2.repo
```

增加以下内容：

```
[mariadb]
name = MariaDB
baseurl = https://mirrors.ustc.edu.cn/mariadb/yum/10.2/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
```

3. 执行安装

```
yum -y install mariadb mariadb-server MariaDB-client  MariaDB-common
```

4. 如果之前已经安装， 需要先删除（如果之前没有安装， 可以忽略此步骤）

   - 停止Mariadb服务

     ```
     [root@localhost yum.repos.d]# ps -ef | grep mysql
     root       1954      1  0 Oct04 ?        00:05:43 /usr/sbin/mysqld --
     wsrep-new-cluster --user=root
     root      89521  81403  0 07:40 pts/0    00:00:00 grep --color=auto
     mysql
     [root@localhost yum.repos.d]# kill 1954
     ```

   - 卸载Mariadb服务

     ```
     yum -y remove Maria*
     ```

   - 删除数据与配置：

     ```
     rm -rf /var/lib/mysql/*
     rm -rf /etc/my.cnf.d/
     rm -rf /etc/my.cnf
     ```

5. 启动MariaDB后，执行安全配置向导命令，可根据安全配置向导提高数据库的安全性

```
systemctl  start mariadb
mysql_secure_installation
```

6. 开启用户远程连接权限

   将连接用户root开启远程连接权限；

```
mysql -uroot -p654321
```

   进入MySQL服务， 执行以下操作：

```
use mysql;
delete from user;
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '654321' WITH GRANT
OPTION;
FLUSH PRIVILEGES;
```

### 2.1.2 MariaDB双主同步

1. 在Server1增加配置：

   在/etc/my.cnf中添加以下配置：

```
[mysqld]
server-id  = 1
log-bin=mysql-bin
relay-log = mysql-relay-bin
replicate-wild-ignore-table=mysql.%
replicate-wild-ignore-table=information_schema.%
log-slave-updates=on
slave-skip-errors=all
auto-increment-offset=1
auto-increment-increment=2
binlog_format=mixed
expire_logs_days=10
```

   注意， Server1自增为奇数位：

   > auto-increment-offset=1 主键自增基数, 从1开始。
   >
   > auto-increment-increment=2 主键自增偏移量，每次为2。

2. 在Server2增加配置：

   修改/etc/my.cnf:

```
[mysqld]
server-id = 2
log-bin=mysql-bin
relay-log = mysql-relay-bin
replicate-wild-ignore-table=mysql.%
replicate-wild-ignore-table=information_schema.%
log-slave-updates=on
slave-skip-errors=all
auto-increment-offset=2
auto-increment-increment=2
binlog_format=mixed
expire_logs_days=10
```

   Server2自增为偶数位：

   > auto-increment-offset=2 主键自增基数, 从2开始。
   >
   > auto-increment-increment=2 主键自增偏移量，每次为2。

   配置修改完成后， 重启数据库。

3. 同步授权配置

   在Server1创建replica用于主从同步的用户：

```
MariaDB [(none)]> grant replication slave, replication client on *.* to
'replica'@'%' identified by 'replica';
mysql> flush privileges;
```

查询日志文件与偏移量，开启同步时需使用：

```
MariaDB [(none)]> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |      663 |              |                  |
+------------------+----------+--------------+------------------+
```

同样， 在Server2创建replica用于主从同步的用户：

```
MariaDB [(none)]> grant replication slave, replication client on *.* to
'replica'@'%' identified by 'replica';
mysql> flush privileges;
```

查询日志文件与偏移量：

```
MariaDB [(none)]> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |      663 |              |                  |
+------------------+----------+--------------+------------------+
```

4. 配置主从同步信息

在Server1中执行：

```
MariaDB [(none)]> change master to
master_host='192.168.116.141',master_user='replica',
master_password='replica', master_port=3306, master_log_file='mysql-
bin.000018', master_log_pos=374, master_connect_retry=30;
```

在Server2中执行：

```
MariaDB [(none)]> change master to
master_host='192.168.116.140',master_user='replica',
master_password='replica', master_port=3306, master_log_file='mysql-
bin.000027', master_log_pos=374, master_connect_retry=30;
```

5. 开启双主同步

在Server1和Server2中分别执行：

```
MariaDB [(none)]> start slave;
Query OK, 0 rows affected (0.00 sec)
```

在Server1查询同步信息：

```
MariaDB [(none)]>  show slave status\G;
```

```
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.10.20.126
                  Master_User: replica
                  Master_Port: 3306
                Connect_Retry: 30
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 663
               Relay_Log_File: mysql-relay-bin.000002
                Relay_Log_Pos: 555
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
...
```

在Server2查询同步信息：

```
MariaDB [(none)]>  show slave status\G;
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.10.20.125
                  Master_User: replica
                  Master_Port: 3306
                Connect_Retry: 30
              Master_Log_File: mysql-bin.000001
          Read_Master_Log_Pos: 663
               Relay_Log_File: mysql-relay-bin.000002
                Relay_Log_Pos: 555
        Relay_Master_Log_File: mysql-bin.000001
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
...
```

Slave_IO_Running和Slave_SQL_Running 都是Yes，说明双主同步配置成功。

### 2.1.3 KeepAlived安装与高可用配置

1. 在Server1与Server2两台节点安装keepalived：

```
yum -y install keepalived
```

2. 关闭防火墙

```
systemctl stop firewalld
systemctl disable firewalld
```

3. 设置主机名称：

Server1节点：

```
hostnamectl set-hostname vip1
```

Server2节点：

```
hostnamectl set-hostname vip2
```

4. Server1节点配置

/etc/keepalived/keepalived.conf：

```
global_defs {
   router_id vip1            # 机器标识，和主机名保持一致
}
vrrp_instance VI_1 {             #vrrp实例定义
    state BACKUP                #lvs的状态模式，MASTER代表主， BACKUP代表备份节点
    interface ens33               #绑定对外访问的网卡
    virtual_router_id 111         #虚拟路由标示，同一个vrrp实例采用唯一标示
    priority 100                #优先级，100代表最大优先级， 数字越大优先级越高
    advert_int 1               #master与backup节点同步检查的时间间隔，单位是秒
    authentication {            #设置验证信息
        auth_type PASS          #有PASS和AH两种
        auth_pass 6666          #验证密码，BACKUP密码须相同
    }
    virtual_ipaddress {          #KeepAlived虚拟的IP地址
        192.168.116.150
    }
}
virtual_server 192.168.116.150 3306 {        #配置虚拟服务器IP与访问端口
    delay_loop 6                  #健康检查时间
    lb_algo rr                    #负载均衡调度算法， rr代表轮询
    lb_kind DR                    #负载均衡转发规则
    persistence_timeout 0         #会话保持时间，这里要做测试， 所以设为0， 实际可根
据session有效时间配置
    protocol TCP                #转发协议类型，支持TCP和UDP
    real_server 192.168.116.140 3306 {     #配置服务器节点VIP1
    notify_down /usr/local/shell/mariadb.sh #当服务挂掉时， 会执行此脚本，结束
keepalived进程
    weight 1                #设置权重，越大权重越高
    TCP_CHECK {               #r状态监测设置
       connect_timeout 10        #超时配置， 单位秒
       retry 3              #重试次数
       delay_before_retry 3         #重试间隔
       connect_port 3306          #连接端口， 和上面保持一致
       }
    }

}
```

创建关闭脚本mariadb.sh

/usr/local/shell/mariadb.sh：

```
pkill keepalived
```

加入执行权限：

```
chmod a+x mariadb.sh
```

5. Server2节点配置：

```
global_defs {
   router_id vip2           # 机器标识，和主机名保持一致
}
vrrp_instance VI_1 {            #vrrp实例定义
    state BACKUP               #lvs的状态模式，MASTER代表主， BACKUP代表备份节点
    interface ens33               #绑定对外访问的网卡
    virtual_router_id 111         #虚拟路由标示，同一个vrrp实例采用唯一标示
    priority 98                #优先级，100代表最大优先级， 数字越大优先级越高
    advert_int 1               #master与backup节点同步检查的时间间隔，单位是秒
    authentication {           #设置验证信息
        auth_type PASS         #有PASS和AH两种
        auth_pass 6666         #验证密码，BACKUP密码须相同
    }
    virtual_ipaddress {         #KeepAlived虚拟的IP地址
        192.168.116.150
    }
}
virtual_server 192.168.116.150 3306 {       #配置虚拟服务器IP与访问端口
    delay_loop 6                  #健康检查时间
    lb_algo rr                    #负载均衡调度算法， rr代表轮询，可以关闭
    lb_kind DR                    #负载均衡转发规则，可以关闭
    persistence_timeout 0         #会话保持时间，这里要做测试， 所以设为0， 实际可根
据session有效时间配置
    protocol TCP                #转发协议类型，支持TCP和UDP
    real_server 192.168.116.141 3306{    #配置服务器节点VIP2
    notify_down /usr/local/shell/mariadb.sh #当服务挂掉时， 会执行此脚本，结束
keepalived进程
    weight 1                #设置权重，越大权重越高
    TCP_CHECK {               #r状态监测设置
       connect_timeout 10        #超时配置， 单位秒
       retry 3              #重试次数
       delay_before_retry 3         #重试间隔
       connect_port 3306          #连接端口， 和上面保持一致
       }
    }

}
```

和Server1的差异项：

```
router_id vip2   # 机器标识，和主机名保持一致
priority 98                #优先级，100代表最大优先级， 数字越大优先级越高
real_server 10.10.20.126 3306  #配置服务器节点VIP2
```

注意， 两台节点都设为BACKUP

```
virtual_router_id 111         #同一个vrrp实例采用唯一标示
state BACKUP
```

如果不想重启后， 争夺备用节点的VIP， 可以设置此项

```
nopreempt #不主动抢占资源
```

6. 验证高可用

   停止主节点MariaDB服务， 验证是否自动切换。

### 2.1.4 搭建应用服务工程

1. ShardingJDBC的介绍

   是ShardingSphere 下的一个产品

   定位为轻量级 Java 框架，在 Java 的 JDBC 层提供的额外服务。 它使用客户端直连数据库，以 jar 包形式提供服务，无需额外部署和依赖，可理解为增强版的 JDBC 驱动，完全兼容 JDBC 和各种 ORM 框架。

   - 适用于任何基于 JDBC 的 ORM 框架，如：JPA, Hibernate, Mybatis, Spring JDBC Template 或直接使用 JDBC。
   - 支持任何第三方的数据库连接池，如：DBCP, C3P0, BoneCP, Druid, HikariCP 等。
   - 支持任意实现 JDBC 规范的数据库，目前支持 MySQL，Oracle，SQLServer，PostgreSQL 以及任何遵循 SQL92 标准的数据库

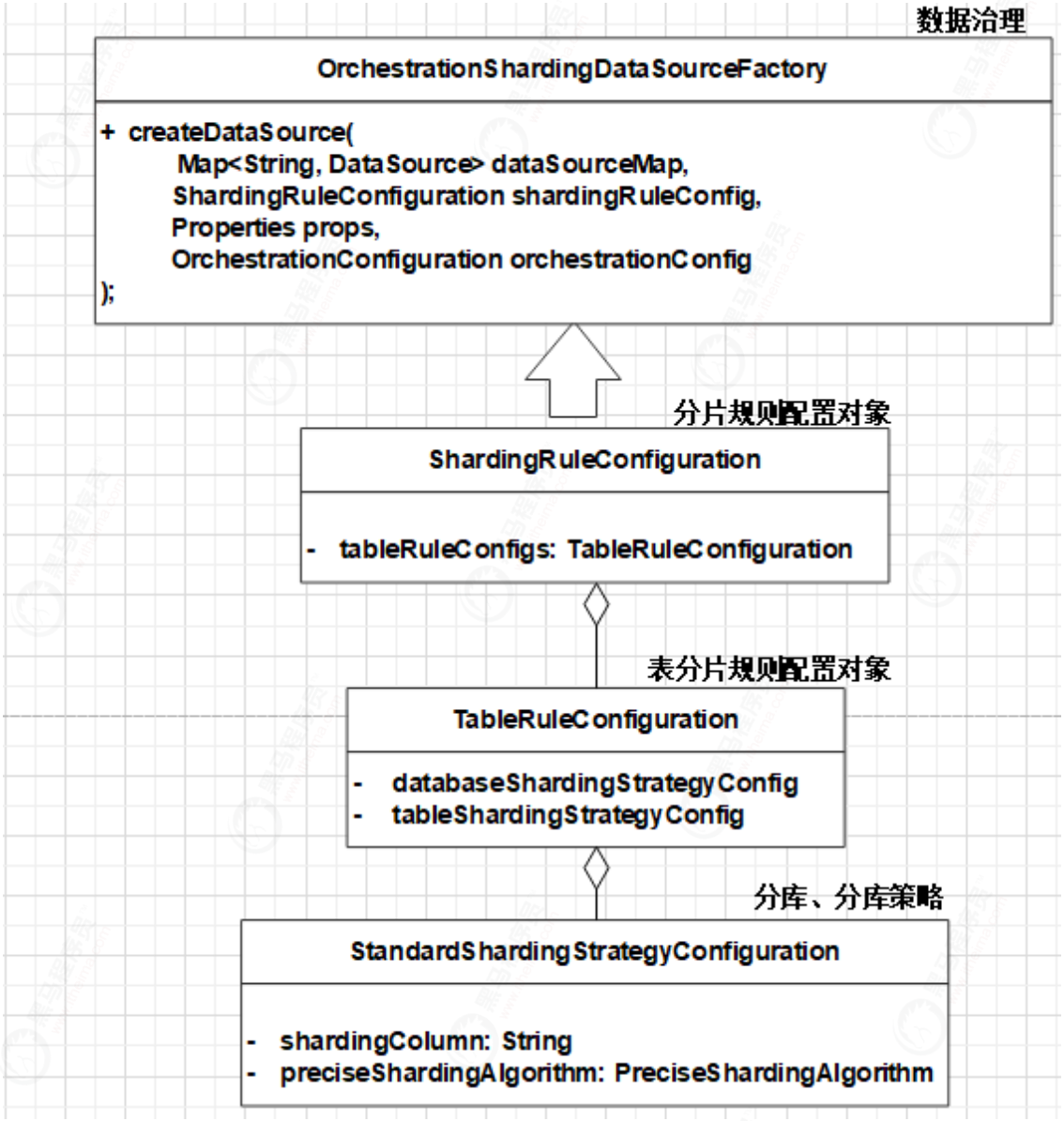
2. ShardingJDBC初始化流程

   1）配置ShardingRuleConfiguration对象

   2）配置表分片规则TableRuleConfiguration对象，设置分库、分表策略

   3）通过Factory对象将Rule对象与DataSource对象装配

   4）ShardingJDBC使用DataSource对象进行分库




3. ShardingJDBC集成配置

1）maven依赖

2）规则配置application.yml

3）创建DataSource

4. 验证应用服务动态扩容

1. 配置两个数据源，分别指向Server1和Server2
2. 分片只配置一个数据源
3. 动态增加另一个数据源

```
// 动态数据源配置实现扩容
Properties properties = loadPropertiesFile("datasource1.properties");
try {
    log.info("load datasource config url: " + properties.get("url"));
    DruidDataSource druidDataSource = (DruidDataSource)
DruidDataSourceFactory.createDataSource(properties);
    druidDataSource.setRemoveAbandoned(true);
    druidDataSource.setRemoveAbandonedTimeout(600);
    druidDataSource.setLogAbandoned(true);
```

```
    // 设置数据源错误重连时间
    druidDataSource.setTimeBetweenConnectErrorMillis(60000);
    druidDataSource.init();
    OrchestrationShardingDataSource dataSource =
SpringContextUtil.getBean("tradeSystemDataSource",
OrchestrationShardingDataSource.class);
    Map<String, DataSource> dataSourceMap =
dataSource.getDataSource().getDataSourceMap();
    dataSourceMap.put(DatasourceEnum.DATASOURCE_2.getValue(),
druidDataSource);

    Map<String, DataSourceConfiguration> dataSourceConfigMap = new
HashMap<String, DataSourceConfiguration>();
    for(String key : dataSourceMap.keySet()) {
        dataSourceConfigMap.put(key,
DataSourceConfiguration.getDataSourceConfiguration(dataSourceMap.get(key)));
    }
    String result =
SHARDING_RULE_TABLE_ORDER.replace(SHARDING_RULE_DATASOURCE, newRule);
    replaceActualDataNodes(result);
    SHARDING_RULE_DATASOURCE = newRule;

    dataSource.renew(new DataSourceChangedEvent(
        "/" + DruidSystemDataSourceConfiguration.DYNAMIC_SHARDING +
"/config/schema/logic_db/datasource",
        dataSourceConfigMap));
    return;


} catch (Exception e) {
    log.error(e.getMessage(), e);
}
```

5. 注意事项

Sharding JDBC, Mycat, Drds 等产品都是分布式数据库中间件, 相比直接的数据源操作, 会存在一些限制, Sharding JDBC在使用时, 要注意以下问题:

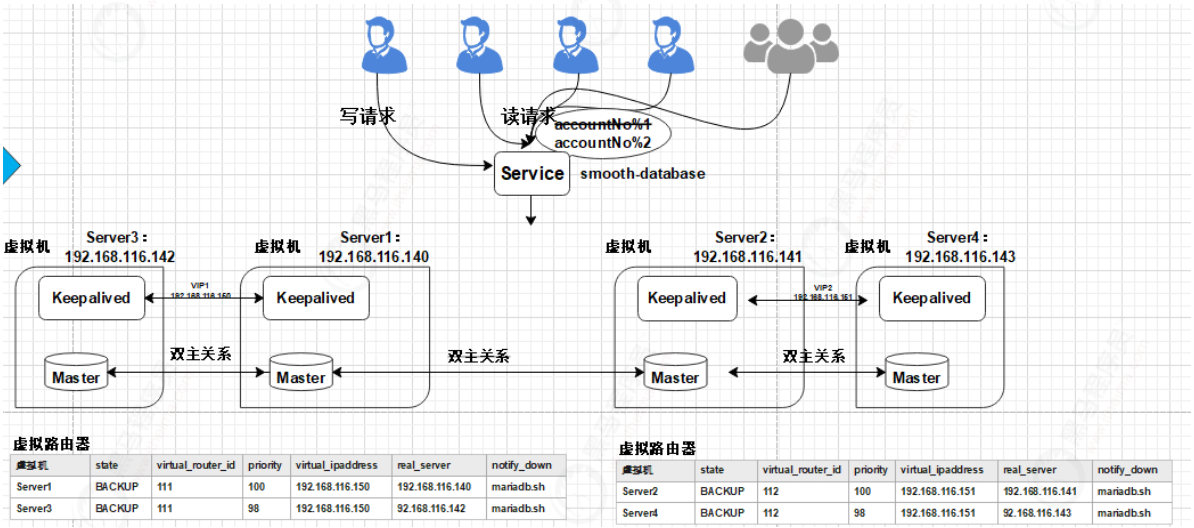
- 有限支持子查询
- 不支持HAVING
- 不支持OR，UNION 和 UNION ALL
- 不支持特殊INSERT
- 每条INSERT语句只能插入一条数据，不支持VALUES后有多行数据的语句
- 不支持DISTINCT聚合
- 不支持dual虚拟表查询
- 不支持SELECT LAST_INSERT_ID(), 不支持自增序列
- 不支持CASE WHEN

6. 课后作业

ShardingJDBC数据源的动态切换实现?

## 2.2 实现数据库的秒级平滑2N扩容

扩容部署架构：




| 虚拟机 | state | virtual_router_id | priority | virtual_ipaddress | real_server | notify_down |
|---|---|---|---|---|---|---|
| Server1 | BACKUP | 111 | 100 | 192.168.116.150 | 192.168.116.140 | mariadb.sh |
| Server3 | BACKUP | 111 | 98 | 192.168.116.150 | 92.168.116.142 | mariadb.sh |

| 虚拟机 | state | virtual_router_id | priority | virtual_ipaddress | real_server | notify_down |
|---|---|---|---|---|---|---|
| Server2 | BACKUP | 112 | 100 | 192.168.116.151 | 192.168.116.141 | mariadb.sh |
| Server4 | BACKUP | 112 | 98 | 192.168.116.151 | 92.168.116.143 | mariadb.sh |

### 2.2.1 新增数据库VIP

1. 在Server2节点， 增加VIP

   修改/etc/keepalived/keepalived.conf

```
global_defs {
   router_id vip2
}
vrrp_instance VI_1 {              #vrrp实例定义
    state BACKUP                 #lvs的状态模式，MASTER代表主， BACKUP代表备份节点
    interface ens33                #绑定对外访问的网卡
    virtual_router_id 112          #虚拟路由标示，同一个vrrp实例采用唯一标示
    priority 100                 #优先级，100代表最大优先级， 数字越大优先级越高
    advert_int 1                #master与backup节点同步检查的时间间隔，单位是秒
    authentication {             #设置验证信息
        auth_type PASS           #有PASS和AH两种
        auth_pass 6666           #验证密码，BACKUP密码须相同
    }
    virtual_ipaddress {           #KeepAlived虚拟的IP地址
        192.168.116.151
    }
}
virtual_server 192.168.116.151 3306 {        #配置虚拟服务器IP与访问端口
    delay_loop 6                  #健康检查时间
    persistence_timeout 0         #会话保持时间，这里要做测试， 所以设为0， 实际可根
据session有效时间配置
    protocol TCP                 #转发协议类型，支持TCP和UDP
    real_server 192.168.116.141 3306{     #配置服务器节点VIP1
    notify_down /usr/local/shell/mariadb.sh
    weight 1                   #设置权重，越大权重越高
    TCP_CHECK {                  #r状态监测设置
       connect_timeout 10         #超时配置， 单位秒
       retry 3                 #重试次数
       delay_before_retry 3          #重试间隔
       connect_port 3306           #连接端口， 和上面保持一致
       }
    }
```

```
}
```

注意配置项：

```
virtual_router_id 112          #虚拟路由标示，同一个vrrp实例采用唯一标示
priority 100                   #优先级，100代表最大优先级， 数字越大优先级越高
```

### 2.2.2 应用服务增加动态数据源

1. 修改应用服务配置， 增加新的数据源， 指向新设置的VIP： 192.168.116.151
2. 通过应用服务接口， 动态扩容调整

### 2.2.3 解除原双主同步

mysql -uroot -p654321

1. 进入Server1:

```
MariaDB [(none)]> stop slave;
```

2. 进入Server2:

```
MariaDB [(none)]> stop slave;
```

3. 通过应用服务接口验证数据是否解除同步

### 2.2.4 安装MariaDB扩容服务器

1. 新建两台虚拟机， 分别为Server3和Server4。
2. 在Server3和Server4两台节点上安装MariaDB服务

   参考2.1.1 MariaDB服务安装
3. 配置Server3与Server1，实现新的双主同步

    1. Server3节点， 修改/etc/my.cnf:

```
[mysqld]
server-id = 3
log-bin=mysql-bin
relay-log = mysql-relay-bin
replicate-wild-ignore-table=mysql.%
replicate-wild-ignore-table=information_schema.%
log-slave-updates=on
slave-skip-errors=all
auto-increment-offset=2
auto-increment-increment=2
binlog_format=mixed
expire_logs_days=10
```

2. 重启Server3数据库

```
service mariadb restart
```

3. 创建replica用于主从同步的用户：

```
MariaDB [(none)]> grant replication slave, replication client on *.* to
'replica'@'%' identified by 'replica';
mysql> flush privileges;
```

4. 在Server1节点，进行数据全量备份：

```
mysqldump -uroot -p654321 --routines --single_transaction --master-data=2 --
databases smooth > server1.sql
```

5. 查看并记录master status信息：

```
...
--
-- Position to start replication or point-in-time recovery from
--

-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000002',
MASTER_LOG_POS=17748;
...
```

6. 将备份的server1.sql通过scp命令拷贝至Server3节点。

```
scp server1.sql root@192.168.116.142:/usr/local/
```

7. 将数据还原至Server3节点：

```
mysql -uroot -p654321 < /usr/local/server1.sql
```

8. 配置主从同步信息

根据上面的master status信息， 在Server3中执行：

```
MariaDB [(none)]> change master to
master_host='192.168.116.140',master_user='replica',
master_password='replica', master_port=3306, master_log_file='mysql-
bin.000002', master_log_pos=17748, master_connect_retry=30;
Query OK, 0 rows affected (0.01 sec)
```

9. 开启主从同步：

```
MariaDB [(none)]> start slave;
Query OK, 0 rows affected (0.00 sec)
```

如果出现问题， 复原主从同步信息：

```
MariaDB [(none)]> reset slave;
Query OK, 0 rows affected (0.01 sec)
```

10. 检查同步状态信息：

```
MariaDB [(none)]> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.10.20.125
                  Master_User: replica
                  Master_Port: 3306
                Connect_Retry: 30
              Master_Log_File: mysql-bin.000004
          Read_Master_Log_Pos: 11174
               Relay_Log_File: mysql-relay-bin.000002
                Relay_Log_Pos: 1746
        Relay_Master_Log_File: mysql-bin.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

11. 配置Server1与Server3节点的同步

查看Server3的日志信息：

```
MariaDB [(none)]> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |     4781 |              |                  |
+------------------+----------+--------------+------------------+
```

在Server1节点， 配置同步信息：

```
MariaDB [(none)]> reset slave;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> change master to
master_host='192.168.116.142',master_user='replica',
master_password='replica', master_port=3306, master_log_file='mysql-
bin.000001', master_log_pos=4781, master_connect_retry=30;

MariaDB [(none)]> start slave;
Query OK, 0 rows affected (0.00 sec)
```

4. 配置Server4与Server2的双主同步

    1. Server4节点， 修改/etc/my.cnf:

```
[mysqld]
server-id = 3
log-bin=mysql-bin
relay-log = mysql-relay-bin
replicate-wild-ignore-table=mysql.%
replicate-wild-ignore-table=information_schema.%
log-slave-updates=on
slave-skip-errors=all
auto-increment-offset=2
auto-increment-increment=2
binlog_format=mixed
expire_logs_days=10
```

2. 重启Server4数据库

```
service mariadb restart
```

3. 创建replica用于主从同步的用户：

```
MariaDB [(none)]> grant replication slave, replication client on *.* to
'replica'@'%' identified by 'replica';
mysql> flush privileges;
```

4. 在Server2节点，进行数据全量备份：

```
mysqldump -uroot -p654321 --routines --single_transaction --master-data=2 --
databases smooth > server2.sql
```

5. 查看并记录master status信息：

```
...
--
-- Position to start replication or point-in-time recovery from
--

-- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000003', MASTER_LOG_POS=4208;

...
```

6. 将备份的server2.sql通过scp命令拷贝至Server4节点。

```
scp server2.sql root@192.168.116.143:/usr/local/
```

7. 将数据还原至Server4节点：

```
mysql -uroot -p654321 < /usr/local/server2.sql
```

8. 配置主从同步信息

根据上面的master status信息， 在Server4中执行：

```
MariaDB [(none)]> change master to
master_host='192.168.116.141',master_user='replica',
master_password='replica', master_port=3306, master_log_file='mysql-
bin.000003', master_log_pos=4208, master_connect_retry=30;
Query OK, 0 rows affected (0.01 sec)
```

9. 开启主从同步：

```
MariaDB [(none)]> start slave;
Query OK, 0 rows affected (0.00 sec)
```

注意， 如果出现问题， 复原主从同步信息：

```
MariaDB [(none)]> reset slave;
Query OK, 0 rows affected (0.01 sec)
```

10. 检查同步状态信息：

```
MariaDB [(none)]> show slave status \G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: 10.10.20.125
                  Master_User: replica
                  Master_Port: 3306
                Connect_Retry: 30
              Master_Log_File: mysql-bin.000004
          Read_Master_Log_Pos: 11174
               Relay_Log_File: mysql-relay-bin.000002
                Relay_Log_Pos: 1746
        Relay_Master_Log_File: mysql-bin.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
```

11. 配置Server2与Server4节点的同步

查看Server4的日志信息：

```
MariaDB [(none)]> show master status;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |     3696 |              |                  |
+------------------+----------+--------------+------------------+
```

在Server2节点， 配置同步信息：

```
MariaDB [(none)]> reset slave;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> change master to
master_host='192.168.116.143',master_user='replica',
master_password='replica', master_port=3306, master_log_file='mysql-
bin.000001', master_log_pos=3696, master_connect_retry=30;

MariaDB [(none)]> start slave;
Query OK, 0 rows affected (0.00 sec)
```

### 2.2.5 增加KeepAlived服务实现高可用

1. 确保新增的Server3和Server4节点安装Keepalived服务。
2. 修改Server3节点配置

```
global_defs {
   router_id vip3          # 机器标识，一般设为hostname，故障发生时，邮件通知会使用
到。
}
vrrp_instance VI_1 {            #vrrp实例定义
    state BACKUP               #lvs的状态模式，MASTER代表主， BACKUP代表备份节点
    interface ens33              #绑定对外访问的网卡
    virtual_router_id 111        #虚拟路由标示，同一个vrrp实例采用唯一标示
    priority 98               #优先级，100代表最大优先级， 数字越大优先级越高
    advert_int 1              #master与backup节点同步检查的时间间隔，单位是秒
    authentication {           #设置验证信息
        auth_type PASS         #有PASS和AH两种
        auth_pass 6666         #验证密码，BACKUP密码须相同
    }
    virtual_ipaddress {         #KeepAlived虚拟的IP地址
        192.168.116.150
    }
}
virtual_server 192.168.116.150 3306 {       #配置虚拟服务器IP与访问端口
    delay_loop 6                 #健康检查时间
    persistence_timeout 0        #会话保持时间，这里要做测试， 所以设为0， 实际可根
据session有效时间配置
    protocol TCP               #转发协议类型，支持TCP和UDP
    real_server 192.168.116.142 3306{    #配置服务器节点VIP3
    notify_down /usr/local/shell/mariadb.sh
    weight 1                #设置权重，越大权重越高
    TCP_CHECK {               #r状态监测设置
       connect_timeout 10       #超时配置， 单位秒
       retry 3              #重试次数
       delay_before_retry 3        #重试间隔
       connect_port 3306         #连接端口， 和上面保持一致
       }
    }

}
```

注意里面IP配置正确， 修改完成后重启服务。

创建关闭脚本mariadb.sh

/usr/local/shell/mariadb.sh：

```
pkill keepalived
```

加入执行权限：

```
chmod a+x mariadb.sh
```

3. 修改Server4节点配置

```
global_defs {
   router_id vip4          # 机器标识，一般设为hostname，故障发生时，邮件通知会使用到。
}
vrrp_instance VI_1 {            #vrrp实例定义
    state BACKUP                #lvs的状态模式，MASTER代表主， BACKUP代表备份节点
    interface ens33               #绑定对外访问的网卡
    virtual_router_id 112         #虚拟路由标示，同一个vrrp实例采用唯一标示
    priority 98                 #优先级，100代表最大优先级， 数字越大优先级越高
    advert_int 1                #master与backup节点同步检查的时间间隔，单位是秒
    authentication {            #设置验证信息
        auth_type PASS          #有PASS和AH两种
        auth_pass 6666          #验证密码，BACKUP密码须相同
    }
    virtual_ipaddress {          #KeepAlived虚拟的IP地址
        192.168.116.151
    }
}
virtual_server 192.168.116.151 3306 {       #配置虚拟服务器IP与访问端口
    delay_loop 6                  #健康检查时间
    persistence_timeout 0         #会话保持时间，这里要做测试， 所以设为0， 实际可根据session有效时间配置
    protocol TCP                #转发协议类型，支持TCP和UDP
    real_server 192.168.116.143 3306{    #配置服务器节点VIP4
    notify_down /usr/local/shell/mariadb.sh
    weight 1                #设置权重，越大权重越高
    TCP_CHECK {               #r状态监测设置
       connect_timeout 10        #超时配置， 单位秒
       retry 3              #重试次数
       delay_before_retry 3         #重试间隔
       connect_port 3306          #连接端口， 和上面保持一致
       }
    }

}
```

创建关闭脚本mariadb.sh

/usr/local/shell/mariadb.sh：

```
pkill keepalived
```

给所有的用户组加入执行权限：

```
chmod a+x mariadb.sh
```

4. 修改完后重启Keepalived服务。

### 2.2.6 清理数据并验证

1. 通过应用服务动态扩容接口做调整和验证
2. 在Server1节点清理数据

   根据取模规则， 保留accountNo为偶数的数据

   ```
   delete from t_trade_order where accountNo % 2 != 0
   ```

3. 在Server2节点清理数据

   根据取模规则， 保留accountNo为奇数的数据

   ```
   delete from t_trade_order where accountNo % 2 != 1
   ```

# 3.keepalived高可用配置大全

## Server1和Server2双主关系

## Server1: keepalived.conf

vi /etc/keepalived/keepalived.conf

```
global_defs {
   router_id vip1
}
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 111
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 6666
    }
    virtual_ipaddress {
        192.168.116.150
    }
}
virtual_server 192.168.116.150 3306 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    persistence_timeout 0
    protocol TCP
    real_server 192.168.116.140 3306 {
    notify_down /usr/local/shell/mariadb.sh
    weight 1
    TCP_CHECK {
       connect_timeout 10
       retry 3
       delay_before_retry 3
       connect_port 3306
       }
    }
}
```

## Server2:keepalived.conf

vi /etc/keepalived/keepalived.conf

```
global_defs {
   router_id vip2
}
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 111
    priority 98
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 6666
    }
```

```
    virtual_ipaddress {
        192.168.116.150
    }
}
virtual_server 192.168.116.150 3306 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    persistence_timeout 0
    protocol TCP
    real_server 192.168.116.141 3306{
    notify_down /usr/local/shell/mariadb.sh
    weight 1
    TCP_CHECK {
       connect_timeout 10
       retry 3
       delay_before_retry 3
       connect_port 3306
       }
    }
}
```

## 新增数据库VIP

### Server2:keepalived.conf

vi /etc/keepalived/keepalived.conf

```
global_defs {
   router_id vip2
}
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 112
    priority 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 6666
    }
    virtual_ipaddress {
        192.168.116.151
    }
}
virtual_server 192.168.116.151 3306 {
    delay_loop 6
    persistence_timeout 0
    protocol TCP
    real_server 192.168.116.141 3306{
    notify_down /usr/local/shell/mariadb.sh
    weight 1
    TCP_CHECK {
       connect_timeout 10
       retry 3
       delay_before_retry 3
       connect_port 3306
```

```
        }
    }
}
```

## Server1和Server3双主关系

### Server3: keepalived.conf

vi /etc/keepalived/keepalived.conf

```
global_defs {
   router_id vip3
}
vrrp_instance VI_1 {
    state BACKUP
    interface ens33
    virtual_router_id 111
    priority 98
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 6666
    }
    virtual_ipaddress {
        192.168.116.150
    }
}
virtual_server 192.168.116.150 3306 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    persistence_timeout 0
    protocol TCP
    real_server 192.168.116.142 3306 {
    notify_down /usr/local/shell/mariadb.sh
    weight 1
    TCP_CHECK {
       connect_timeout 10
       retry 3
       delay_before_retry 3
       connect_port 3306
       }
    }
}
```

## Server2和Server4双主关系

### Server4: keepalived.conf

vi /etc/keepalived/keepalived.conf

```
global_defs {
   router_id vip4
}
vrrp_instance VI_1 {
    state BACKUP
```

```
    interface ens33
    virtual_router_id 112
    priority 98
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 6666
    }
    virtual_ipaddress {
        192.168.116.151
    }
}
virtual_server 192.168.116.151 3306 {
    delay_loop 6
    lb_algo rr
    lb_kind DR
    persistence_timeout 0
    protocol TCP
    real_server 192.168.116.143 3306{
    notify_down /usr/local/shell/mariadb.sh
    weight 1
    TCP_CHECK {
       connect_timeout 10
       retry 3
       delay_before_retry 3
       connect_port 3306
       }
    }
}
```