SMTP POP3 IMAP 邮件解析 MySQL 分布式存储

目录

SMTP POP3 IMAP 邮件解析 MySQL 分布式存储

1. 程序介绍

Input：

两种输入方式：

(1) python Mail_Process.py `pcapng 文件主目录` null # 解析当天 pcapng 文件

(2) python Mail_Process.py `pcapng 文件主目录` 20220128 # 解析指定时间 pcapng 文件

Output:

输出为图片信息和 MySQL 数据库信息：

(1) image 文件夹：以原始 pcapng 名称存放解析图片信息。如下图所示：

(2) 解析信息 MySQL 存取：imap_database，smtp_database，pop3_database。如下图所示：

id	insert_time	timestamp	sport	dport	source_ip	target_ip	len	ttl	host_name	mail_from	mail_to
801	2022-01-29	2021-12-23 10:2	51843	143	172.26.1.22	58.251.106.181	64	40	DESKTOP-VNDAI	18640660987@	lifanchao19977
803	2022-01-29	2022-01-21 16:0	56906	143	172.26.1.22	123.126.97.78	64	49	DESKTOP-VNDAI	18640660987@	lifanchao19977

x_mail	mail_subject	mail_priority	message_id	mail_content	file_path	image_sign	parse_sign
Foxmail 7.2.23.11	123	3	2021122310273	111	E:\home\one\Dat	0	1
Foxmail 7.2.23.11	111	3	2022012116072	123	E:\home\one\Dat	1	1

id:编号

insert_time:插入时间

timestamp:邮件时间戳

sport:客户端端口号

dport:服务器端口号

source_ip:源目标 ip 地址

target_ip:目标 ip 地址

len:ip 包的总长度比特，以字节为单位

ttl:生存时间

host_name:主机名

mail_from:邮件发送方

mail_to:邮件接收方

x_mail:邮件客户端

mail_subject:邮件主题
mail_priority:邮件设置等级
message_id:邮件 id
mail_content:邮件内容(内容经过 base64 解析)
fail_path:pcapng 存储位置
image_sign:pcapng 文件 image 标签（0：无图片信息，1 有图片信息）
Parse_sign:pcapng 文件处理成功标签（0：处理失败，1 插入成功）

2. 数据库中间件

2.1 数据库中间件

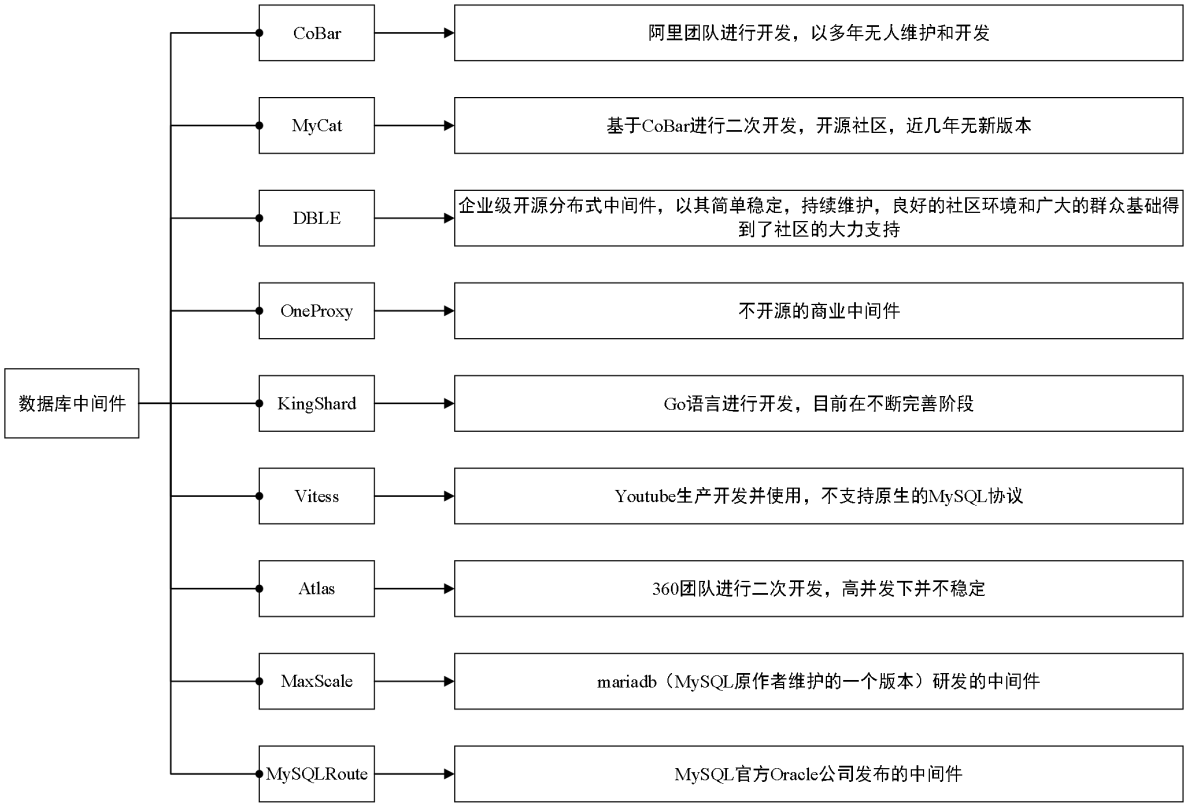
中间件：是一类连接软件组件和应用的计算机软件，以便于软件各部件之间的沟通。例子：Tomcat，web 中间件。

数据库中间件：连接 Java、Python 等应用程序和数据库。

2.2 数据库中间件优点

(1) 应用端与数据库紧耦合。
(2) 高访问量高并发对数据库的压力。
(3) 读写请求数据不一致。

2.3 数据库中间件对比



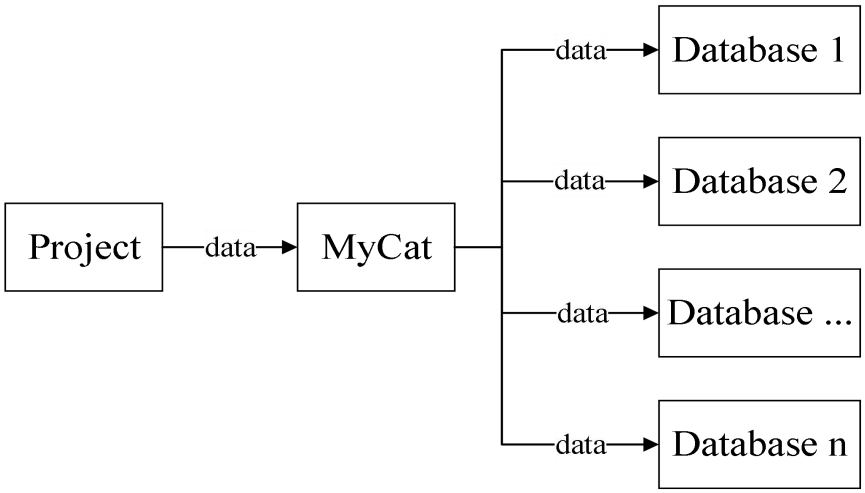
3. MyCat

3.1 MyCat 介绍

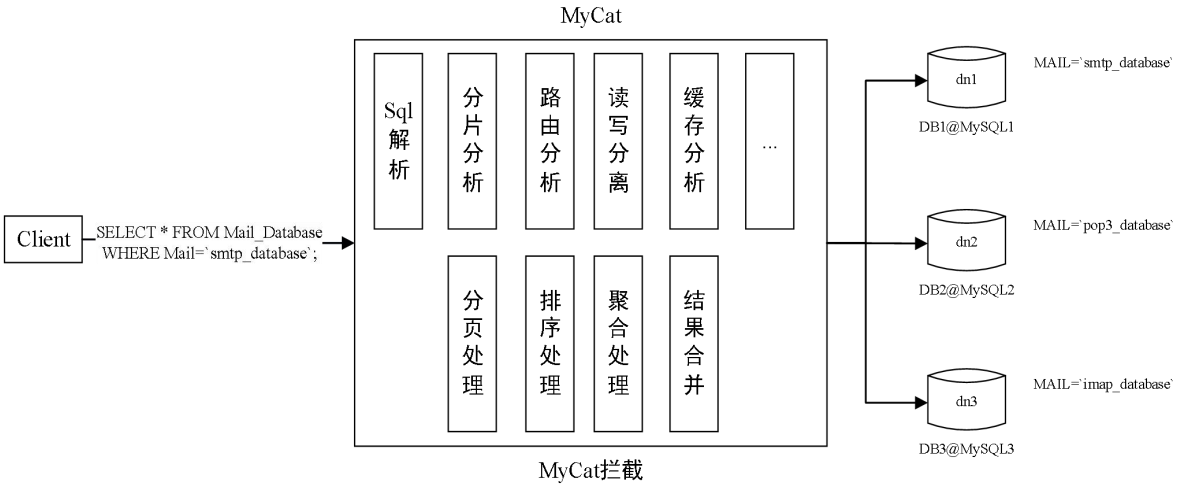
MyCat 是目前比较流行的基于 Java 语言编写的数据库中间件，是一个实现了 MySql 协议的服务器，其核心功能是分库分表；配合数据库的主从模式可以实现读写分离；具有优秀的第三方工具。

MyCat 发展到目前的版本，已经不是一个单纯的 MySQL 代理了，它的后端可以支持 MySQL、SQL Server、Oracle、DB2、PostgreSQL 等主流数据库，也支持 MongoDB 这种新型 NoSQL 方式的存储，未来还会支持更多类型的存储。而在最终用户看来，无论是那种存储方式，在 MyCat 里，都是一个传统的数据库表，支持标准的 SQL 语句进行数据的操作，这样一来，对前端业务系统来说，可以大幅降低开发难度，提升开发速度。

3.2 MyCat 使用框架



3.3 MyCat 原理



mycat 的原理中最重要的一个动作是“拦截”，它拦截了用户发送过来的 SQL 语句，首先对 SQL 语句做了一些特定的分析：如分片分析、路由分析、读写分离分析、缓存分析等，然后将此 SQL 发送后端的真实数据库，并将返回的结果做适当的处理，最终再返回给用户。

上述图片里，Mail_Database 表被分为了三个分片 datanode(简称 dn)，这三个分片是分布在三台 MySQL Server 上（Datahost），即 datanode=database@datahost，因此可以用一台到 N 台服务器来分片，根据分片规则（sharding rule）对数据进行判断存储在哪台节点。

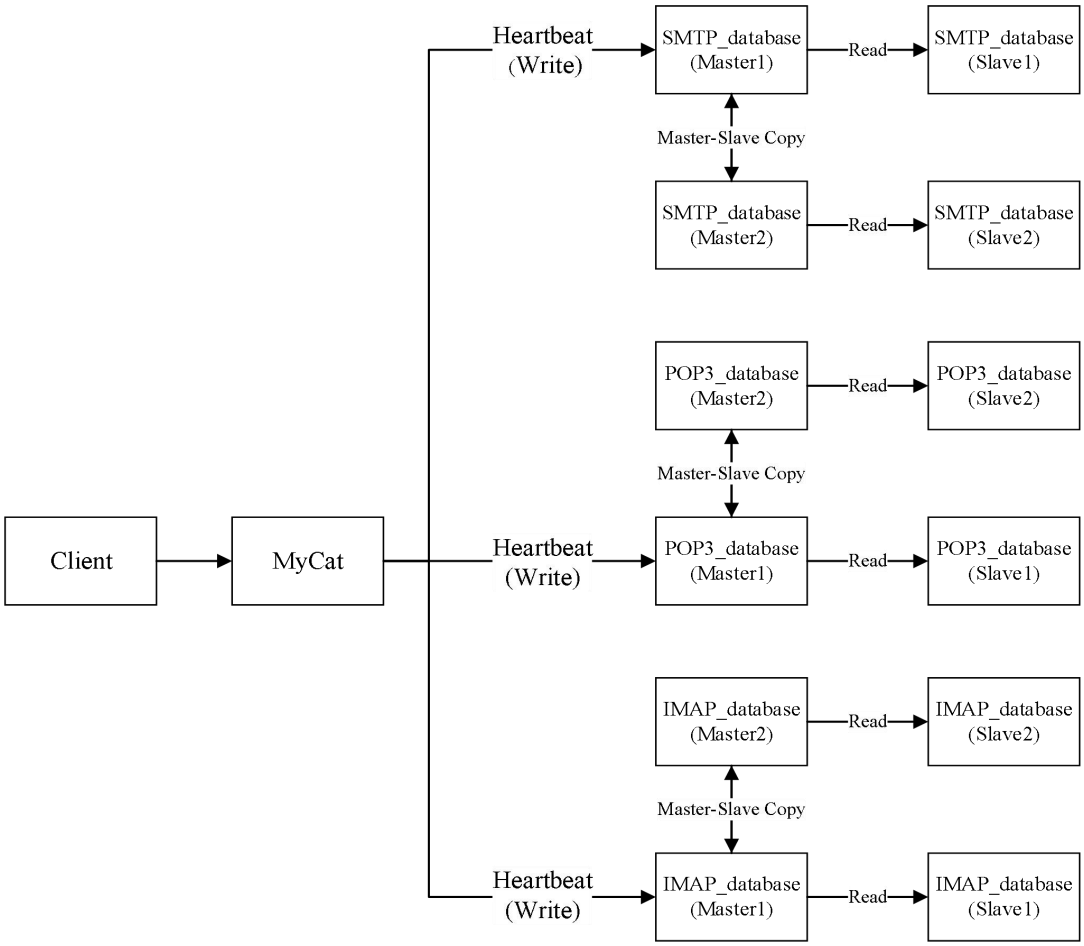
当 mycat 收到一个 SQL 时，会先解析这个 SQL,查找涉及到的表，然后看此表的定义，如果有分片规则，则获取到 SQL 里分片字段的值，并分配分片函数，得到该 SQL 对应的分片列表，然后将 SQL 发往这些分片去执行，最后收集和理所有分片返回的结果数据，并输出到客户端，以 SELECT * FROM Mail_Database WHERE Mail=`smtp_database`语句为例，查到 MAIL=`smtp_database`，按照分片函数，smtp_database 返回 dn1，于是 sql 就发给了 mysql1，去取 db1 上的查询结果，并返回给用户。

4. MyCat 配置(操作系统:Ubuntu 18.04)

4.1 配置文件说明

(1) schema.xml：定义逻辑库，表、分片节点等内容。
(2) rule.xml：定义分片规则。
(3) server.xml：定义用户以及系统相关变量，如端口等。

4.2 MySQL 分库分表框架



程序端连接 MyCat，实现双主从复制和读写分离。首先对 Master1 进行心跳检测(Heartbeat)，检测是否连接成功，如果连接失败立马切换到 Master2，Master1 和 Slave1、Master2 和 Slave2、Master1 和 Master2 会建立索引日志，保证数据同步成功。其次 Master1、Master2 设置为写主机，Slave1、Slave2 设置为读主机，实现读写分离。

4.3 用户名密码修改

修改配置文件 server.xml，将用户名设置为 root，密码设置为 123456，表名设置为 mycat。

```
<user name="root">
        <property name="password">123456</property>
        <property name="schemas">mycat</property>
```

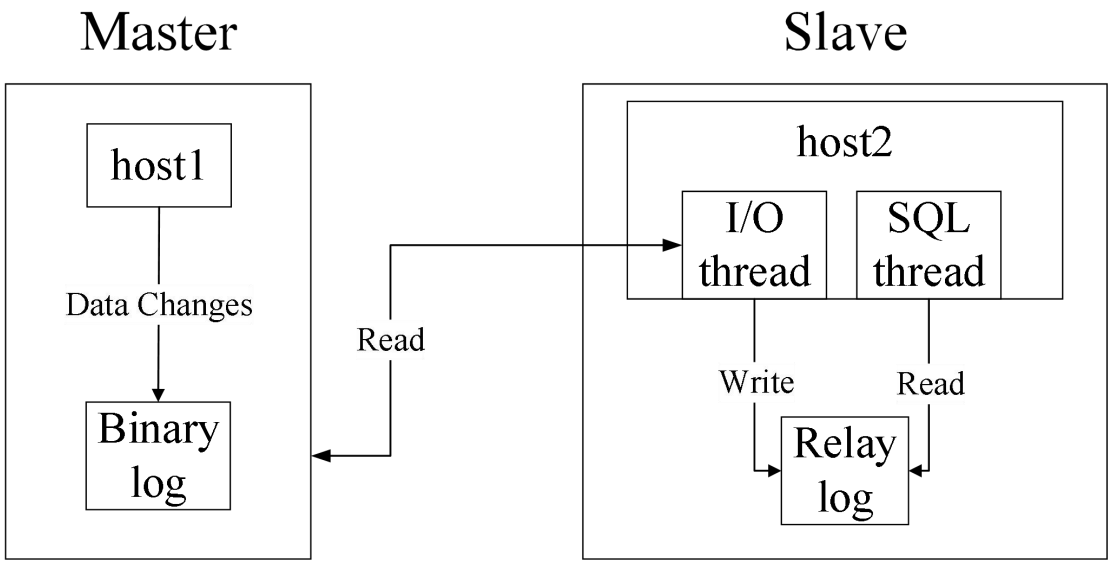
MyCat 启动：

./mycat console 控制台启动

MySQL 测试连接 MyCat：

mysql -u root -p 123456 -h ip 地址 -P 8066

4.4 双主从复制



(1) master 将改变记录到二进制日志(Binary log)

(2) Slave 访问 Master 将 Master 的 Binary log 记录拷贝到 Slave 的中继日志(Relay log)

(3) Slave 的 SQL thread 线程执行 Relay log 的事件，将改变执行一遍,同步到 Slave 的数据库中

编号	角色	IP 地址	机器名
1	Master1	192.168.81.215	Host1
2	Slave1	192.168.81.216	Host2
3	Master2	192.168.81.217	Host3
4	Slave2	192.168.81.218	Host4

修改配置文件：sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf

(1) Master1 配置

```
# 主服务器唯一 ID
server-id=1
#启用二进制日志
log-bin=mysql-bin
# 设置不要复制的数据库(可设置多个)
binlog-ignore-db=mysql
binlog-ignore-db=information_schema
# 设置需要复制的数据库
binlog-do-db=需要复制的主数据库名字
# 设置 logbin 格式
binlog_format=STATEMENT
# 在作为从数据库的时候，有写入操作也要更新二进制日志文件
```

log-slave-updates

```
server-id=1
log-bin=mysql-bin
binlog-ignore-db=mysql
binlog-ignore-db=information_schema
binlog-do-db=imap_database
binlog-format=STATEMENT

log-slave-updates
```

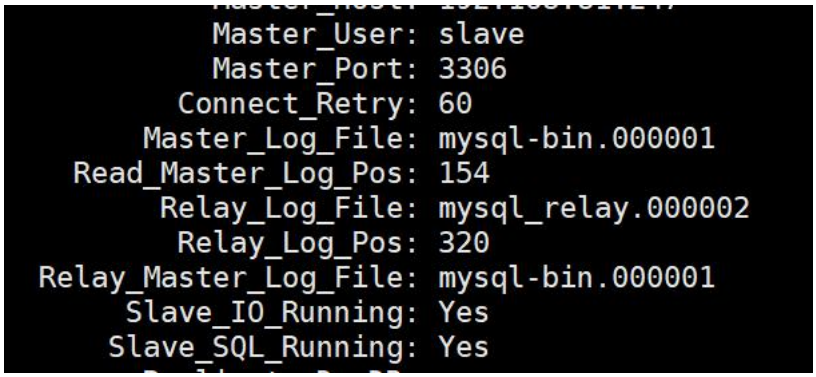
(2) Slave1 配置
#从服务器唯一 ID
server-id=2
#启用中继日志
relay-log=mysql-relay

```
server_id=2
relay_log=mysql_relay
```

(3) Master2、Slave2 同理。修改 id 分别为 3、4
(4) 双主机、双从机重启 mysql 服务
(5) 两台主机上建立帐户并授权 slave
#在主机 Master1、Master2 MySQL 里执行授权命令
GRANT REPLICATION SLAVE ON *.* TO 'slave'@'%' IDENTIFIED BY '123456';
#查询 Master1、Master2 的状态，分别记录下 File 和 Position 的值
show master status;

```
mysql> show master status;
+------------------+----------+--------------+--------------------------+-------------------+
| File             | Position | Binlog_Do_DB  | Binlog_Ignore_DB         | Executed_Gtid_Set |
+------------------+----------+--------------+--------------------------+-------------------+
| mysql-bin.000001 |      154 | imap_database | mysql,information_schema |                   |
+------------------+----------+--------------+--------------------------+-------------------+
1 row in set (0.00 sec)
```

(6) 在从机上配置需要复制的主机
Slava1 复制 Master1，Slava2 复制 Master2，Master1 复制 Master2
#复制主机的命令
CHANGE MASTER TO MASTER_HOST='主机的 IP 地址',
MASTER_USER='slave',
MASTER_PASSWORD='123456',
MASTER_LOG_FILE='mysql-bin.具体数字',MASTER_LOG_POS=具体值;
#启动两台从服务器复制功能
start slave;
#查看从服务器状态
show slave status\G;



Slave_IO_Running: Yes
Slave_SQL_Running: Yes
显示配置成功

4.5 基于主从复制实现读写分离

修改 Mycat 的配置文件 schema.xml
以 smtp 数据库为例，imap 和 pop3 同理。
(1)修改<dataHost>的 balance 属性，通过此属性配置读写分离的类型.
负载均衡类型，目前的取值有 4 种：
①balance="0", 不开启读写分离机制，所有读操作都发送到当前可用的 writeHost 上。
②balance="1"，全部的 readHost 与 stand by writeHost 参与 select 语句的负载均衡，简单的说，当双主双从模式(M1->S1，M2->S2，并且 M1 与 M2 互为主备)，正常情况下，M2,S1,S2 都参与 select 语句的负载均衡。
③balance="2"，所有读操作都随机的在 writeHost、readhost 上分发。
④balance="3"，所有读请求随机的分发到 readhost 执行，writerHost 不负担读压力。
(2) 设置 Master 写主机 ip 和 Slave 读主机 ip，balance="1"，实现读写分离。

```
<dataHost name="host1" maxCon="1000" minCon="10" balance="1"
                writeType="0" dbType="mysql" dbDriver="native" switchType="1"  slaveThreshold="100">
        <heartbeat>select user()</heartbeat>
        <writeHost host="hostM1" url="192.168.81.215:3306" user="root" password="123456">
                <readHost host="hostS1" url="192.168.81.216:3306" user="root" password="123456" />
        </writeHost>
        <writeHost host="hostM2" url="192.168.81.217:3306" user="root" password="123456">
                <readHost host="hostS2" url="192.168.81.218:3306" user="root" password="123456" />
        </writeHost>
</dataHost>
```

#writeType="0": 所有写操作发送到配置的第一个 writeHost，第一个挂了切到还生存的第二个
#writeType="1"，所有写操作都随机的发送到配置的 writeHost，1.5 以后废弃不推荐使用。
#writeHost，写主机 ip
#readHost，读主机 ip
#switchType="1": 1 默认值，自动切换。
#　　-1 表示不自动切换
#　　2 基于 MySQL 主从同步的状态决定是否切换。
Master1、Master2 互做备机，负责写的主机宕机，备机切换负责写操作，保证数据库读写分离高可用性。

4.6 日期分片

此规则为按天分片。设定时间格式、范围。以 imap_2022_01_27-imap_2022_29 为例。

（1）修改 schema.xml 配置文件

<table name="imap_2022_01" primaryKey="id" dataNode='dn1' subTables= "imap_2022_01_$27-29" rule="sharding-by-date" />

rule 配置规则按照日期进行分片

（2）修改 rule.xml 配置文件

```
<tableRule name="sharding-by-date">
        <rule>
                <columns>insert_time</columns>
                <algorithm>partbyday</algorithm>
        </rule>
</tableRule>
```

```
<function name="partbyday"
                class="io.mycat.route.function.PartitionByDate">
        <property name="dateFormat">yyyy-MM-dd</property>
        <!-- property name="sNaturalDay">0</property -->
        <property name="sBeginDate">2022-01-27</property>
        <!-- property name="sEndDate">2014-01-31</propert -->
        <property name="sPartionDay">1</property>
</function>
```

columns：分片字段，algorithm：分片函数

dateFormat：日期格式

sBeginDate：开始日期

sEndDate：结束日期,则代表数据达到了这个日期的分片后循环从开始分片插入，可以不进行设置

sPartionDay：分区天数，即默认从开始日期算起，分隔 x 天一个分区

4.7 全局序列

在实现分库分表的情况下，数据库自增主键已无法保证自增主键的全局唯一。为此，Mycat 提供了全局 sequence，并且提供了包含本地配置和数据库配置等多种实现方式。

(1) 本地文件

此方式 Mycat 将 sequence 配置到文件中，当使用到 sequence 中的配置后，Mycat 会更新 classpath 中的 sequence_conf.properties 文件中 sequence 当前的值。

① 优点：本地加载，读取速度较快

② 缺点：抗风险能力差，Mycat 所在主机宕机后，无法读取本地文件。

(2) 时间戳方式

全局序列 ID= 64 位二进制 (42(毫秒)+5(机器 ID)+5(业务编码)+12(重复累加) 换算成十进制为 18 位数的 long 类型，每毫秒可以并发 12 位二进制的累加。

① 优点：配置简单

② 缺点：18 位 ID 过长

(3) 数据库方式：使用最多

实现方式：

在数据库中建立一张表 MYCAT_SEQUENCE ，用于存放序列 sequence 名称(name)，sequence 当前值(current_value)，步长(increment int 类型每次读取多少个 sequence。

原理步骤：

①初次获取 sequence 时，根据传入的序列名称，从数据库表中获取 current_value,increment 到 MyCat 中，并将数据库中的 current_value 更新为 current_value + increment。

②MyCat 将读取到的 current_value + increment 作为本次的序列，在下次使用时，将序列的值 + 1，当使用 increment 次后，重复第一个步骤。

③MyCat 负责维护序列表 MYCAT_SEQUENCE，当用到序列时，往数据库表中插入一条记录即可。如果某次读取完的序列还没有用完数据库就崩了，那么剩余的没有使用的序列将会被废弃掉。

操作步骤：

①建库序列脚本

```
#在 dn1 上创建全局序列表
CREATE TABLE MYCAT_SEQUENCE (NAME VARCHAR(50) NOT NULL,current_value INT NOT
NULL,increment INT NOT NULL DEFAULT 100, PRIMARY KEY(NAME)) ENGINE=INNODB;

#创建全局序列所需函数
DELIMITER $$
CREATE FUNCTION mycat_seq_currval(seq_name VARCHAR(50)) RETURNS VARCHAR(64)
DETERMINISTIC
BEGIN
DECLARE retval VARCHAR(64);
SET retval="-999999999,null";
SELECT CONCAT(CAST(current_value AS CHAR),",",CAST(increment AS CHAR)) INTO retval FROM
MYCAT_SEQUENCE WHERE NAME = seq_name;
RETURN retval;
END $$
DELIMITER ;

DELIMITER $$
CREATE FUNCTION mycat_seq_setval(seq_name VARCHAR(50),VALUE INTEGER) RETURNS
VARCHAR(64)
DETERMINISTIC
BEGIN
```

```
UPDATE MYCAT_SEQUENCE
SET current_value = VALUE
WHERE NAME = seq_name;
RETURN mycat_seq_currval(seq_name);
END $$
DELIMITER ;

DELIMITER $$
CREATE FUNCTION mycat_seq_nextval(seq_name VARCHAR(50)) RETURNS VARCHAR(64)
DETERMINISTIC
BEGIN
UPDATE MYCAT_SEQUENCE
SET current_value = current_value + increment WHERE NAME = seq_name;
RETURN mycat_seq_currval(seq_name);
END $$
DELIMITER ;

#初始化序列表记录
INSERT INTO MYCAT_SEQUENCE(NAME,current_value,increment) VALUES ('ORDERS', -99,100);
```

设置初始为 1，步长取值为 100，即每次分配 100 号段。

②修改 Mycat 配置

#修改 sequence_db_conf.properties

ORDERS 这个序列在 dn1 这个节点上，具体 dn1 节点在 host 配置，请参考 schema.xml，根据实际配置情况进行配置即可。

```
#sequence stored in datanode
GLOBAL=dn1
COMPANY=dn1
CUSTOMER=dn1
ORDERS=dn1
```

③修改 server.xml

#全局序列类型：0-本地文件，1-数据库方式，2-时间戳方式。此处修改成 1。

```
        <property name="sequenceHandlerType">1</property>
<!--<property name="sequnceHandlerPattern">(?:(\s*next\s+value\s+fo
INSERT INTO `travelrecord` (`id`,user_id) VALUES ('next value for M
```

④MySQL 事件

create event if not exists update_event ON SCHEDULE EVERY 1 DAY STARTS '2022-01-27 00:00:00' on completion not preserve comment '更新current_value' do update MYCAT_SEQUENCE set current_value=-99;

每天定时任务将全局序列置 0

4.8 MyCat 定时脚本

在步骤 7，更新完毕全局序列必须要重新启动 MyCat 才可以生效。

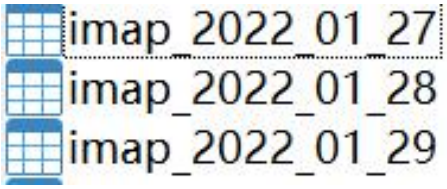
```
# 每天定时开始执行脚本。判断主程序是否在运行，主程序在运行则等待主程序运行完毕在执行，主程序不在执行则开始运行脚本重启 MyCat。
def execCmd(cmd):
    r = os.popen(cmd)
    text = r.read()
    r.close()
    return text
def mycat_restart():
    MycatRestartCmd = "/usr/local/mycat/bin/mycat restart"
    execCmd(MycatRestartCmd)
    print("ok")

if __name__ == '__main__':
    while True:
        time.sleep(2)
        #最后一个|是导向 grep 过滤掉 grep 进程：因为 grep 查看程序名也是进程，会混到查询信息里
        programIsRunningCmd = "ps -ef|grep demo1.py|grep -v grep"
        programIsRunningCmdAns = execCmd(programIsRunningCmd)
        ansLine = programIsRunningCmdAns.split('\n')
       # 判断如果返回行数>2 则说明 python 脚本程序已经在运行，打印提示信息结束程序，否则运行脚本代码 doSomething()
        if len(ansLine) > 2:
            print("programName have been Running")
        else:
            mycat_restart()
            Break
```

5. Table 结构

5.1 Table 创建

IMAP_database 、SMTP_database、POP3_database 按照 database_year_month_day 建表。

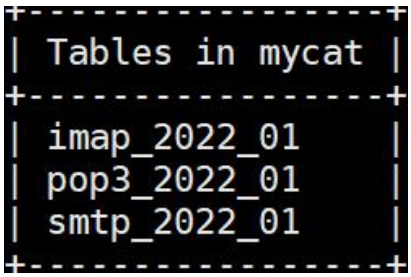


5.2 MyCat 查询

按 database_year_month 进行查表。

USE mycat;

SELECT * FROM IMAP_2022_01;



5.3 Table 设计

```
CREATE TABLE `imap_2022_01_27` (
  `id` int(32) NOT NULL,
  `insert_time` date DEFAULT NULL,
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `sport` int(11) DEFAULT NULL,
  `dport` int(11) DEFAULT NULL,
  `source_ip` char(30) DEFAULT NULL,
  `target_ip` char(30) DEFAULT NULL,
  `len` int(11) DEFAULT NULL,
  `ttl` int(11) DEFAULT NULL,
  `host_name` varchar(50) DEFAULT NULL,
  `mail_from` varchar(50) DEFAULT NULL,
  `mail_to` varchar(50) DEFAULT NULL,
  `x_mail` varchar(50) DEFAULT NULL,
  `mail_subject` varchar(1000) DEFAULT NULL,
  `mail_priority` int(11) DEFAULT NULL,
  `message_id` varchar(50) DEFAULT NULL,
  `mail_content` varchar(10000) DEFAULT NULL,
```

```
  `file_path` varchar(200) DEFAULT NULL,
  `image_sign` int(11) DEFAULT NULL,
  `parse_sign` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

id	insert_time	timestamp	sport	dport	source_ip	target_ip	len	ttl	host_name	mail_from	mail_to
801	2022-01-29	2021-12-23 10:2	51843	143	172.26.1.22	58.251.106.181	64	40	DESKTOP-VNDAI	18640660987@	lifanchao19977
803	2022-01-29	2022-01-21 16:0	56906	143	172.26.1.22	123.126.97.78	64	49	DESKTOP-VNDAI	18640660987@	lifanchao19977

x_mail	mail_subject	mail_priority	message_id	mail_content	file_path	image_sign	parse_sign
Foxmail 7.2.23.11	123	3	20211223102733	111	E:\home\one\Dat	0	1
Foxmail 7.2.23.11	111	3	20220121160726	123	E:\home\one\Dat	1	1

image_sign:图片标签（0：无图片信息，1 有图片信息）
parse_sign:处理成功标签（0：处理失败，1 插入成功）

6. MyCat-Web 监控组件

6.1 MyCat-Web 简介

Mycat-web 是 Mycat 可视化运维的管理和监控平台，弥补了 Mycat 在监控上的空白。帮 Mycat 分担统计任务和配置管理任务。Mycat-web 引入了 ZooKeeper 作为配置中心，可以管理多个节点。

6.2 MyCat-Web 安装

(1) zookeeper

MyCat-eye 运行过程中需要依赖 zookeeper，因此需要先安装 zookeeper。下载 zookeeper，然后解压，在 conf/目录下找到 zoo-sample.cfg，将其复制为 zoo.cfg。

zookeeper 启动

./zkServer.sh start

ZooKeeper 服务端口为 2181，查看服务是否已经启动

netstat -ant | grep 2181

(2) MyCat-web

进入 mycat-web 的目录下运行启动命令

./start.sh & # 以后台形式进行启动

Mycat-web 服务端口为 8082，查看服务已经启动

netstat -ant | grep 8082

(3) 地址访问

http://ip:8082/mycat/

6.3 MyCat-Web 配置

(1) 注册中心配置 ZooKeeper 地址，配置后刷新页面，可见：

(2) 新增 Mycat 监控

管理端口：9066；服务端口：8066；ip 地址、用户名、密码需自己根据实际情况配置。

6.4 Mycat 性能监控指标

在 Mycat-web 上可以进行 Mycat 性能监控，例如：内存分享、流量分析、连接分析、活动线程分析等等。