

介绍:

canal是阿里巴巴旗下的一款开源项目，纯Java开发。基于数据库增量日志解析，提供增量数据订阅&消费，目前主要支持了MySQL（也支持mariaDB）。

背景

早期，阿里巴巴B2B公司因为存在杭州和美国双机房部署，存在跨机房同步的业务需求。不过早期的数据库同步业务，主要是基于trigger的方式获取增量变更，不过从2010年开始，阿里系公司开始逐步的尝试基于数据库的日志解析，获取增量变更进行同步，由此衍生出了增量订阅&消费的业务，从此开启了一段新纪元。ps. 目前内部使用的同步，已经支持mysql5.x和oracle部分版本的日志解析

基于日志增量订阅&消费支持的业务：

数据库镜像

数据库实时备份

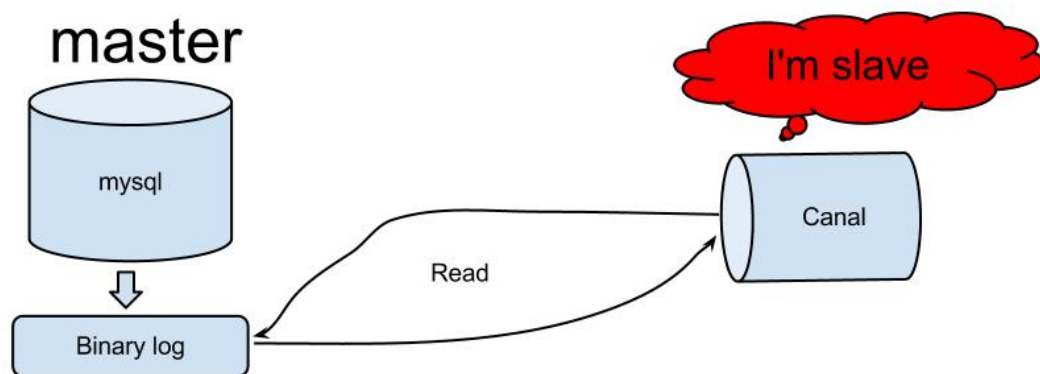
多级索引（卖家和买家各自分库索引）

search build

业务cache刷新

价格变化等重要业务消息

工作原理:

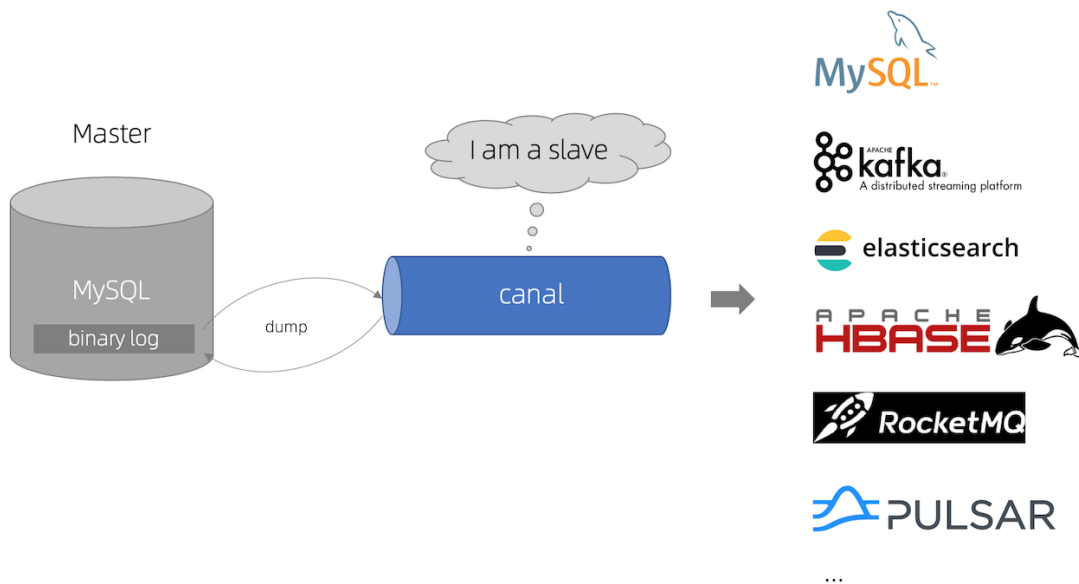


原理相对比较简单：

canal模拟mysql slave的交互协议，伪装自己为mysql slave，向mysql master发送dump协议

mysql master收到dump请求，开始推送binary log给slave(也就是canal)

canal解析binary log对象(原始为byte流)



安装canal:

首先，下载canal安装包

<https://github.com/alibaba/canal/releases>

9. 修复charset编码不存在 #1662

Assets 5

canal.adapter-1.1.3.tar.gz

canal.deployer-1.1.3.tar.gz

canal.example-1.1.3.tar.gz

Source code (zip)

Source code (tar.gz)

安装步骤:

创建一个canal文件夹

解压:

```
tar -zxvf canal.deployer-1.0.24.tar.gz
```

```

drwxr-xr-x 6 root root 87 3月 16 13:44 canal
drwxr-xr-x 6 mysql mysql 4096 3月 10 20:00 data
-rw-r--r-- 1 root root 64 2月 21 15:37 lib.pl
drwxr-xr-x 5 mongo mongo 101 2月 21 15:41 mongodb
drwxr-xr-x 10 root root 222 3月 10 19:43 mysql
drwxr-xr-x 13 root root 245 2月 21 15:39 nginx
drw----- 16 root root 4096 3月 10 19:43 panel
drwxr-xr-x 2 root root 6 2月 21 15:35 php
drwxr-xr-x 2 root root 131 3月 4 14:22 rabbitmq
drwx----- 7 redis redis 4096 3月 15 16:15 redis
drwxr-xr-x 3 root root 77 3月 4 14:20 rocketmq
drwxr-xr-x 2 root root 24 2月 21 15:33 stop
drwxr-xr-x 9 www www 202 2月 21 15:52 tomcat
drwxr-xr-x 3 root root 81 2月 25 21:38 zk
[root@192-168-65-232 server]# cd canal/
[root@192-168-65-232 canal]# ll
总用量 49784
drwxr-xr-x 2 root root 76 3月 16 13:44 bin
-rw-r--r-- 1 root root 50971636 3月 16 13:44 canal.deployer-1.1.3.tar.gz
drwxr-xr-x 5 root root 93 3月 16 13:44 conf
drwxr-xr-x 2 root root 4096 3月 16 13:44 lib
drwxrwxrwx 2 root root 6 4月 4 2019 logs
[root@192-168-65-232 canal]#

```

修改canal配置文件

vi conf/example/instance.properties

```

# enable gtid use true/false
canal.instance.gtidon=false

# position info
canal.instance.master.address=192.168.65.232:3306
canal.instance.master.journal.name=
canal.instance.master.position=
canal.instance.master.timestamp=
canal.instance.master.gtid=

# rds oss binlog
canal.instance.rds.accesskey=
canal.instance.rds.secretkey=
canal.instance.rds.instanceId=

# table meta tsdb info
canal.instance.tsdb.enable=true
#canal.instance.tsdb.url=jdbc:mysql://127.0.0.1:3306/canal_tsdb
#canal.instance.tsdb.dbUsername=canal
#canal.instance.tsdb.dbPassword=canal

#canal.instance.standby.address =
#canal.instance.standby.journal.name =
#canal.instance.standby.position =
#canal.instance.standby.timestamp =
#canal.instance.standby.gtid=

# username/password
canal.instance.dbUsername=root
canal.instance.dbPassword=root
canal.instance.connectionCharset = UTF-8
# enable druid Decrypt database password
canal.instance.enableDruid=false
#canal.instance.pwdPublicKey=MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBALK4BUxdL1tRRE5/zXpVEVPUGunvscYFtE

# table regex
canal.instance.filter.regex=.*\\..*
# table black regex
canal.instance.filter.black.regex=

```

修改mysql配置:

vi /etc/my.cnf

```

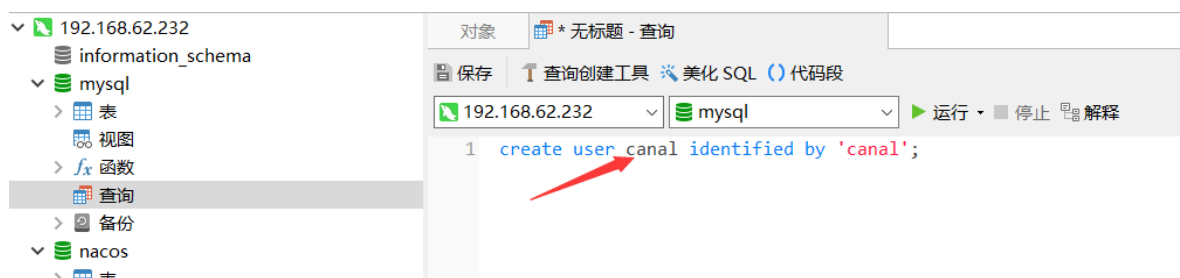
1 log-bin=mysql-bin          #添加这一行就ok
2 binlog-format=ROW          #选择row模式

```

```
3 server-id=1      #配置mysql replaction需要定义，不能和canal的slaveId重复
4 binlog-do-db=micromall
```

执行mysql 创建canal用户:

```
1 create user canal identified by 'canal';
2
3 GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'canal'@'%';
4
5 FLUSH PRIVILEGES;
```



查看是否授权成功:

```
1 select * from user where user='canal'
```

canal的工作原理:

canal 模拟 MySQL slave 的交互协议，伪装自己为 MySQL slave，向 MySQL master 发送 dump 协议

MySQL master 收到 dump 请求，开始推送 binary log 给 slave (即 canal)

canal 解析 binary log 对象(原始为 byte 流)

启动canal:

```
1 cd bin
2 ./startup.sh
```

测试代码:

mysql 主从权限》启动配置》binlog》消息中间件

依赖:

```
1 <dependency>
2   <groupId>com.alibaba.otter</groupId>
3   <artifactId>canal.client</artifactId>
4   <version>1.1.0</version>
5 </dependency>
```

```
1 package com.tuling.tulingmall.client;
2
3 import java.net.InetSocketAddress;
```

```
4 import java.util.List;
5 import com.alibaba.otter.canal.client.CanalConnectors;
6 import com.alibaba.otter.canal.client.CanalConnector;
7 import com.alibaba.otter.canal.common.utils.AddressUtils;
8 import com.alibaba.otter.canal.protocol.Message;
9 import com.alibaba.otter.canal.protocol.CanalEntry.Column;
10 import com.alibaba.otter.canal.protocol.CanalEntry.Entry;
11 import com.alibaba.otter.canal.protocol.CanalEntry.EntryType;
12 import com.alibaba.otter.canal.protocol.CanalEntry.EventType;
13 import com.alibaba.otter.canal.protocol.CanalEntry.RowChange;
14 import com.alibaba.otter.canal.protocol.CanalEntry.RowData;
15
16 /**
17  * @author : 图灵学院
18  * @date : Created in 2020/3/3
19  * @version: V1.0
20  * @slogan: 天下风云出我辈，一入代码岁月催
21  * @description:
22  */
23 public class SimpleCanalClientExample {
24
25
26     public static void main(String args[]) {
27         // 创建链接
28         CanalConnector connector = CanalConnectors.newSingleConnector(new InetSocketAddress("127.0.0.1",
29             11111), "example", "", "");
30         int batchSize = 1000;
31         int emptyCount = 0;
32         try {
33             connector.connect();
34             connector.subscribe(".*\\..*");
35             connector.rollback();
36             int totalEmptyCount = 120;
37             while (emptyCount < totalEmptyCount) {
38                 Message message = connector.getWithoutAck(batchSize); // 获取指定数量的数据
39                 long batchId = message.getId();
40                 int size = message.getEntries().size();
41                 if (batchId == -1 || size == 0) {
42                     emptyCount++;
43                     System.out.println("empty count : " + emptyCount);
44                 }
45                 Thread.sleep(1000);
```

```

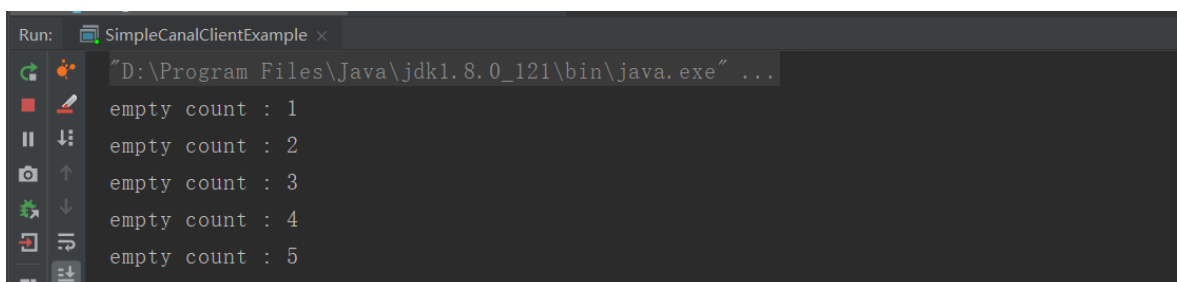
46 } catch (InterruptedException e) {
47 }
48 } else {
49     emptyCount = 0;
50     // System.out.printf("message[batchId=%s,size=%s] \n", batchId, size);
51     printEntry(message.getEntries());
52 }
53
54 connector.ack(batchId); // 提交确认
55 // connector.rollback(batchId); // 处理失败，回滚数据
56 }
57
58 System.out.println("empty too many times, exit");
59 } finally {
60     connector.disconnect();
61 }
62 }
63
64 private static void printEntry(List<Entry> entrys) {
65     for (Entry entry : entrys) {
66         if (entry.getEntryType() == EntryType.TRANSACTIONBEGIN ||
67             entry.getEntryType() == EntryType.TRANSACTIONEND) {
68             continue;
69         }
70         RowChange rowChage = null;
71         try {
72             rowChage = RowChange.parseFrom(entry.getStoreValue());
73         } catch (Exception e) {
74             throw new RuntimeException("ERROR ## parser of eromanga-event has an error ,
75             data:" + entry.toString(),
76             e);
77         }
78         EventType eventType = rowChage.getEventType();
79         System.out.println(String.format("=====> binlog[%s:%s] , name[%s,%s] , eventType : %s",
80             entry.getHeader().getLogfileName(), entry.getHeader().getLogfileOffset(),
81             entry.getHeader().getSchemaName(), entry.getHeader().getTableName(),
82             eventType));
83
84         for (RowData rowData : rowChage.getRowDatasList()) {
85             if (eventType == EventType.DELETE) {

```

```

86     printColumn(rowData.getBeforeColumnsList());
87 } else if (eventType == EventType.INSERT) {
88     printColumn(rowData.getAfterColumnsList());
89 } else {
90     System.out.println("-----> before");
91     printColumn(rowData.getBeforeColumnsList());
92     System.out.println("-----> after");
93     printColumn(rowData.getAfterColumnsList());
94 }
95 }
96 }
97 }
98
99 private static void printColumn(List<Column> columns) {
100     for (Column column : columns) {
101         System.out.println(column.getName() + " : " + column.getValue() + " update
102         =" + column.getUpdated());
103     }
104 }

```



```

Run: SimpleCanalClientExample x
"D:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...
empty count : 1
empty count : 2
empty count : 3
empty count : 4
empty count : 5

```

接入消息中间件:

修改instance 配置文件 **vi conf/example/instance.properties**

```

1 # mq config
2 canal.mq.topic=example
3 # dynamic topic route by schema or table regex
4 #canal.mq.dynamicTopic=mytest1.user,mytest2\\.*,.*\\..*
5 canal.mq.partition=0
6 # hash partition config
7 #canal.mq.partitionsNum=3
8 #canal.mq.partitionHash=test.table:id^name,.*\\..*

```

9 #####

修改canal 配置文件vi /usr/local/canal/conf/canal.properties

```
1 #####
2 ##### MQ #####
3 #####
4 # kafka/rocketmq 集群配置:
5 192.168.1.117:9092,192.168.1.118:9092,192.168.1.119:9092
6 canal.mq.servers = 192.168.1.150:9876
7 canal.mq.retries = 0
8 canal.mq.batchSize = 16384
9 canal.mq.maxRequestSize = 1048576
10 canal.mq.lingerMs = 1
11 canal.mq.bufferMemory = 33554432
12 #消息生产组名
13 canal.mq.producerGroup = Canal-Producer
14 # Canal的batch size, 默认50K, 由于kafka最大消息体限制请勿超过1M(900K以下)
15 canal.mq.canalBatchSize = 30
16 # Canal get数据的超时时间, 单位: 毫秒, 空为不限超时
17 canal.mq.canalGetTimeout = 100
18 # 是否为flat json格式对象
19 canal.mq.flatMessage = true
20 canal.mq.compressionType = none
21 canal.mq.acks = all
22 # use transaction for kafka flatMessage batch produce
23 canal.mq.transaction = false
24 #canal.mq.properties. =
```

配置文件:



canal.properties
4.92KB



instance.properties
1.79KB

参数说明:

参数名	参数说明	默认值
canal.mq.servers	kafka为bootstrap.servers rocketMQ中为nameserver列表	127.0.0.1:6667
canal.mq.retries	发送失败重试次数	0
canal.mq.batchSize	kafka为 <code>ProducerConfig.BATCH_SIZE_CONFIG</code> rocketMQ无意义	16384
canal.mq.maxRequestSize	kafka为 <code>ProducerConfig.MAX_REQUEST_SIZE_CONFIG</code> rocketMQ无意义	1048576
canal.mq.lingerMs	kafka为 <code>ProducerConfig.LINGER_MS_CONFIG</code> , 如果是flatMessage格式建议将该值调大, 如: 200 rocketMQ无意义	1
canal.mq.bufferMemory	kafka为 <code>ProducerConfig.BUFFER_MEMORY_CONFIG</code> rocketMQ无意义	33554432
canal.mq.producerGroup	kafka无意义 rocketMQ为ProducerGroup名	Canal-Producer
canal.mq.canalBatchSize	获取canal数据的批次大小	50
canal.mq.canalGetTimeout	获取canal数据的超时时间	100
canal.mq.flatMessage	是否为json格式 如果设置为false,对应MQ收到的消息为protobuf格式 需要通过CanalMessageDeserializer进行解码	true
canal.mq.transaction	kafka消息投递是否使用事务, 主要针对flatMessage的异步发送和动态多topic消息投递进行事务控制来保持和canal binlog position的一致性, flatMessage模式下建议开启(需要kafka版本支持)。如果设置为false, flatMessage消息将会采用逐条同步的方式投递, 可能会产生消息丢失或者重复投递 rocketMQ无意义	false
—	—	—
canal.mq.topic	mq里的topic名	无
canal.mq.dynamicTopic	mq里的动态topic规则, 1.1.3版本支持	无
canal.mq.partition	单队列模式的分区下标,	1
canal.mq.partitionsNum	散列模式的分区数	无
canal.mq.partitionHash	散列规则定义 库名.表名: 唯一主键, 比如mytest.person: id 1.1.3版本支持新语法, 见下文	无

canal内部原理：



- server代表一个canal运行实例，对应于一个jvm
- instance对应于一个数据队列（1个server对应1..n个instance）

instance模块：

- eventParser (数据源接入，模拟slave协议和master进行交互，协议解析)
- eventSink (Parser和Store链接器，进行数据过滤，加工，分发的任务)

- eventStore (数据存储)
- metaManager (增量订阅&消费信息管理器)

```
1 mysql> show variables like 'binlog_format';
2 +-----+-----+
3 | Variable_name | Value |
4 +-----+-----+
5 | binlog_format | ROW |
6 +-----+-----+
7 1 row in set (0.00 sec)
```

解析开始：

com.alibaba.otter.canal.parse.inbound.AbstractEventParser#start

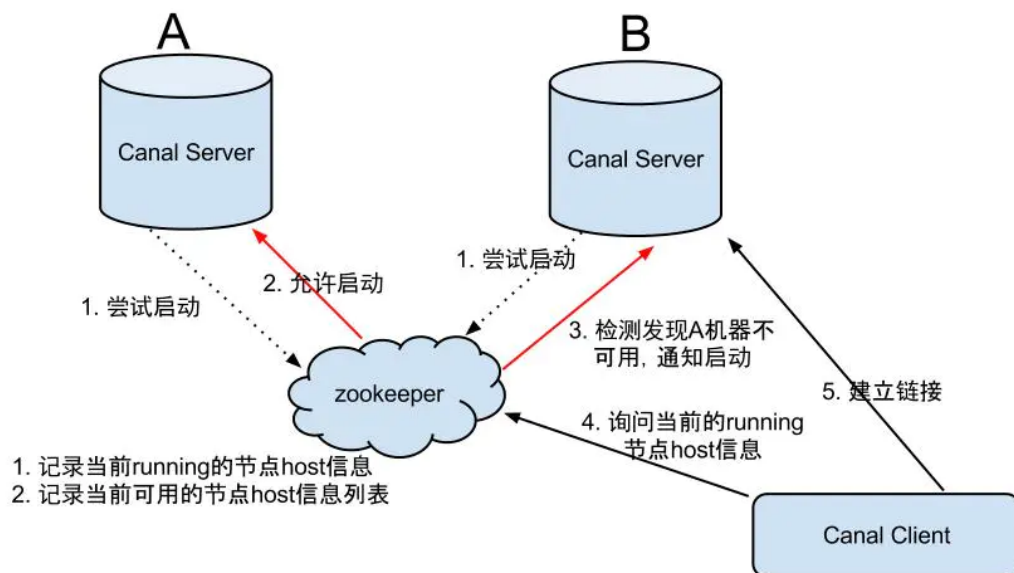
集群高可用：

canal的ha分为两部分，canal server和canal client分别有对应的ha实现

canal server: 为了减少对mysql dump的请求，不同server上的instance要求同一时间只能有一个处于running，其他的处于standby状态。

canal client: 为了保证有序性，一份instance同一时间只能由一个canal client进行get/ack/rollback操作，否则客户端接收无法保证有序。

整个HA机制的控制主要是依赖了zookeeper的几个特性，watcher和EPHEMERAL节点(和session生命周期绑定)。



大致步骤:

1. canal server要启动某个canal instance时都先向zookeeper进行一次尝试启动判断 (实现: 创建EPHEMERAL节点, 谁创建成功就允许谁启动)
2. 创建zookeeper节点成功后, 对应的canal server就启动对应的canal instance, 没有创建成功的canal instance就会处于standby状态
3. 一旦zookeeper发现canal server A创建的节点消失后, 立即通知其他的canal server再次进行步骤1的操作, 重新选出一个canal server启动instance.
4. canal client每次进行connect时, 会首先向zookeeper询问当前是谁启动了canal instance, 然后和其建立链接, 一旦链接不可用, 会重新尝试connect.

Canal Client的方式和canal server方式类似, 也是利用zookeeper的抢占EPHEMERAL节点的方式进行控制.

文档: canal安装与使用和原理详解.note

链接: [http://note.youdao.com/noteshare?](http://note.youdao.com/noteshare?id=dbb19e16255ee256bcd6a13f022d7f98&sub=21FD7C556576403EB97B6F512DCDF56B)

id=dbb19e16255ee256bcd6a13f022d7f98&sub=21FD7C556576403EB97B6F512DCDF56B