尚硅谷大数据技术之 Canal

(作者：尚硅谷研究院)

版本 3.0

# 第 1 章 Canal 入门

* 1. **什么是 Canal**

阿里巴巴 B2B 公司，因为业务的特性，卖家主要集中在国内，买家主要集中在国外，所以衍生出了同步杭州和美国异地机房的需求，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。

Canal 是用 Java 开发的基于数据库增量日志解析，提供增量数据订阅&消费的中间件。目前。Canal 主要支持了 MySQL 的 Binlog 解析，解析完成后才利用 Canal Client 来处理获得的相关数据。（数据库同步需要阿里的 Otter 中间件，基于 Canal）。

* 1. **MySQL 的Binlog**
     1. **什么是 Binlog**

MySQL 的二进制日志可以说 MySQL 最重要的日志了，它记录了所有的 DDL 和 DML(除了数据查询语句)语句，以事件形式记录，还包含语句所执行的消耗的时间，MySQL 的二进制日志是事务安全型的。

一般来说开启二进制日志大概会有 1%的性能损耗。二进制有两个最重要的使用场景:

其一：MySQL Replication 在Master 端开启Binlog，Master 把它的二进制日志传递给Slaves

来达到 Master-Slave 数据一致的目的。

其二：自然就是数据恢复了，通过使用 MySQL Binlog 工具来使恢复数据。

二进制日志包括两类文件：二进制日志索引文件（文件名后缀为.index）用于记录所有的二进制文件，二进制日志文件（文件名后缀为.00000\*）记录数据库所有的 DDL 和 DML(除了数据查询语句)语句事件。

* + 1. **Binlog 的分类**

MySQL Binlog 的格式有三种，分别是 STATEMENT,MIXED,ROW。在配置文件中可以选择配置 binlog\_format= statement|mixed|row。三种格式的区别：

* + - 1. statement：语句级，binlog 会记录每次一执行写操作的语句。相对 row 模式节省空间，但是可能产生不一致性，比如“update tt set create\_date=now()”，如果用 binlog 日志进行恢复，由于执行时间不同可能产生的数据就不同。

优点：节省空间。

缺点：有可能造成数据不一致。

* + - 1. row：行级， binlog 会记录每次操作后每行记录的变化。

优点：保持数据的绝对一致性。因为不管 sql 是什么，引用了什么函数，他只记录执行后的效果。

缺点：占用较大空间。

* + - 1. mixed：statement 的升级版，一定程度上解决了，因为一些情况而造成的statement模式不一致问题，默认还是 statement，在某些情况下譬如：当函数中包含 UUID() 时；包含 AUTO\_INCREMENT 字段的表被更新时；执行 INSERT DELAYED 语句时；用 UDF 时；会按照 ROW 的方式进行处理

优点：节省空间，同时兼顾了一定的一致性。

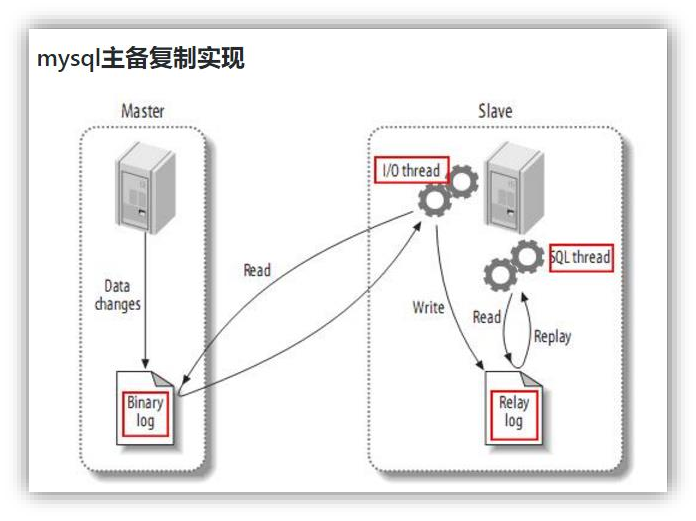
缺点：还有些极个别情况依旧会造成不一致，另外 statement 和 mixed 对于需要对

binlog 的监控的情况都不方便。

综合上面对比，Canal 想做监控分析，选择row 格式比较合适。

* 1. **Canal 的工作原理**
     1. **MySQL 主从复制过程**

1. Master 主库将改变记录，写到二进制日志(Binary Log)中；
2. Slave 从库向 MySQL Master 发送 dump 协议，将 Master 主库的 binary log events 拷贝到它的中继日志(relay log)；
3. Slave 从库读取并重做中继日志中的事件，将改变的数据同步到自己的数据库。



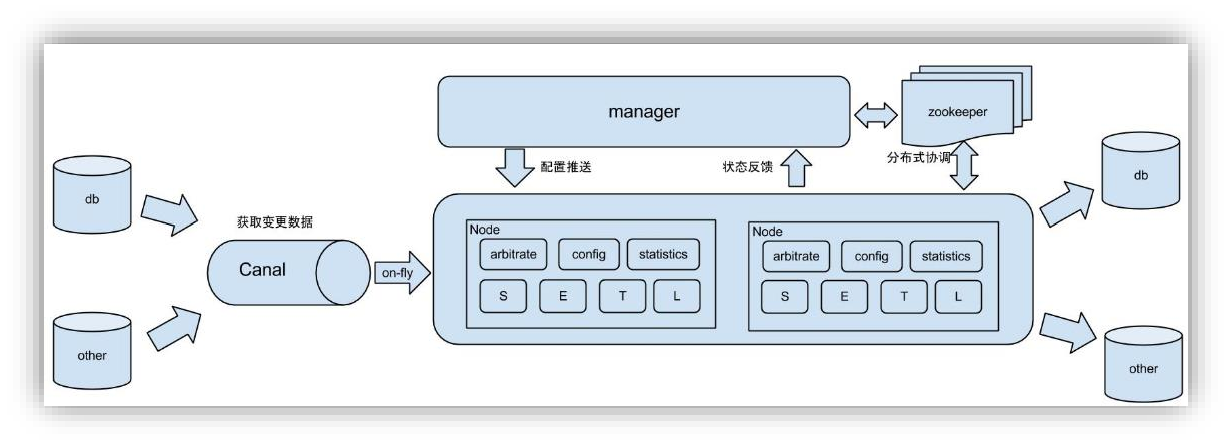
* + 1. **Canal 的工作原理**

很简单，就是把自己伪装成 Slave，假装从 Master 复制数据。

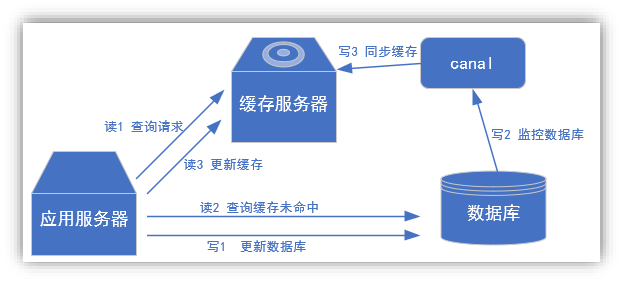
* 1. **使用场景**

1）原始场景： 阿里 Otter 中间件的一部分

Otter 是阿里用于进行异地数据库之间的同步框架，Canal 是其中一部分。



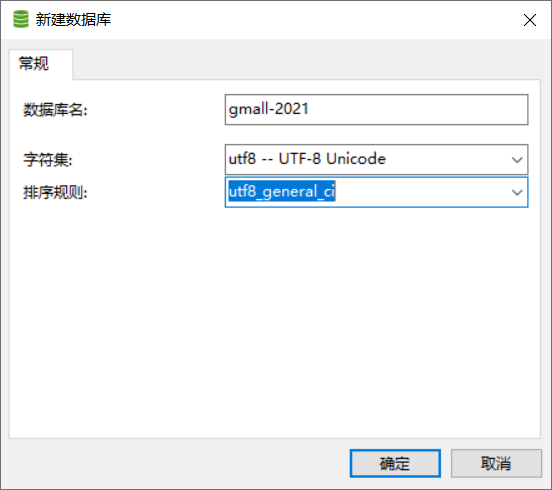
2）常见场景 1：更新缓存



3）常见场景 2：抓取业务表的新增变化数据，用于制作实时统计（我们就是这种场景）

# 第 2 章 MySQL 的准备

* 1. **创建数据库**



* 1. **创建数据表**

CREATE TABLE user\_info(

`id` VARCHAR(255),

`name` VARCHAR(255),

`sex` VARCHAR(255)

);

* 1. **修改配置文件开启Binlog**

[atguigu@hadoop102 module]$ sudo vim /etc/my.cnf server-id=1

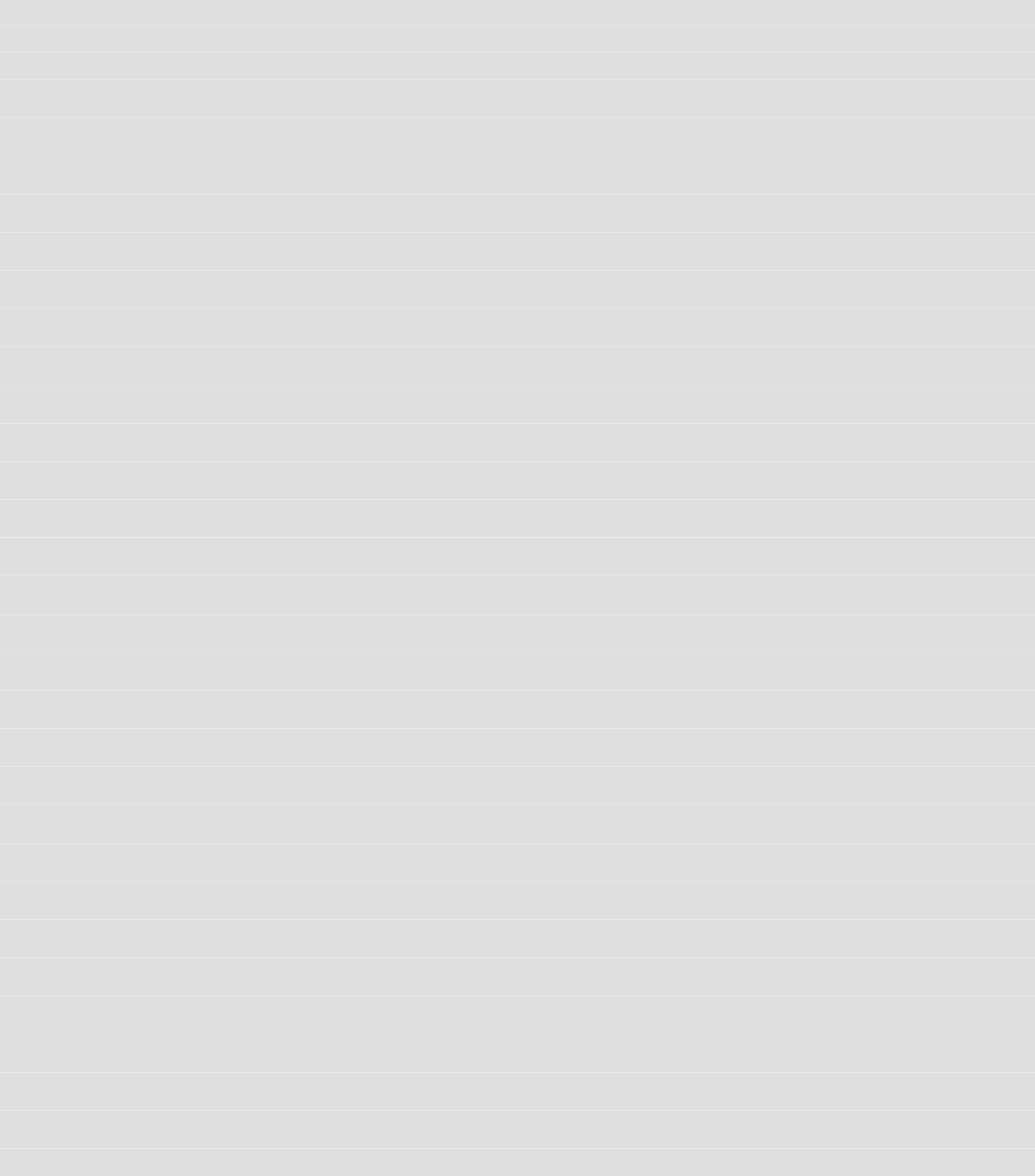
log-bin=mysql-bin binlog\_format=row binlog-do-db=gmall-2021

注意：binlog-do-db 根据自己的情况进行修改，指定具体要同步的数据库，如果不配置则表示所有数据库均开启Binlog

* 1. **重启 MySQL 使配置生效**

sudo systemctl restart mysqld

到/var/lib/mysql 目录下查看初始文件大小 154

[atguigu@hadoop102 lib]$ pwd

/var/lib

[atguigu@hadoop102 lib]$ sudo ls -l mysql

总用量 474152

-rw-r-----. 1 mysql mysql 56 8 月 7 2020 auto.cnf

drwxr-x---. 2 mysql mysql 4096 9 月 25 2020 azkaban

-rw-------. 1 mysql mysql 1680 8 月 7 2020 ca-key.pem

-rw-r--r--. 1 mysql mysql 1112 8 月 7 2020 ca.pem

drwxr-x--- 2 mysql mysql 4096 8 月 18 16:56 cdc\_test

-rw-r--r--. 1 mysql mysql 1112 8 月 7 2020 client-cert.pem

-rw-------. 1 mysql mysql 1676 8 月 7 2020 client-key.pem

drwxr-x---. 2 mysql mysql 4096 9 月 25 2020 gmall\_report

-rw-r----- 1 mysql mysql 1085 12 月 1 09:12 ib\_buffer\_pool

-rw-r-----. 1 mysql mysql 79691776 12 月 13 08:45 ibdata1

-rw-r-----. 1 mysql mysql 50331648 12 月 13 08:45 ib\_logfile0

-rw-r-----. 1 mysql mysql 50331648 12 月 13 08:45 ib\_logfile1

-rw-r----- 1 mysql mysql 12582912 12 月 13 08:45 ibtmp1

drwxr-x--- 2 mysql mysql 4096 9 月 22 15:30 maxwell

drwxr-x---. 2 mysql mysql 4096 8 月 12 2020 metastore

drwxr-x---. 2 mysql mysql 4096 9 月 22 15:43 mysql

-rw-r-----. 1 mysql mysql 154 12 月 13 08:45 mysql-bin.000001

-rw-r----- 1 mysql mysql 19 12 月 13 08:45 mysql-bin.index

srwxrwxrwx 1 mysql mysql 0 12 月 13 08:45 mysql.sock

-rw------- 1 mysql mysql 5 12 月 13 08:45 mysql.sock.lock

drwxr-x---. 2 mysql mysql 4096 8 月 7 2020 performance\_schema

-rw-------. 1 mysql mysql 1680 8 月 7 2020 private\_key.pem

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -rw-r--r--. | 1 | mysql | mysql | 452 8 月 7 2020 public\_key.pem |
| -rw-r--r--. | 1 | mysql | mysql | 1112 8 月 7 2020 server-cert.pem |
| -rw --. | 1 | mysql | mysql | 1680 8 月 7 2020 server-key.pem |
| drwxr-x---. | 2 | mysql | mysql | 12288 8 月 7 2020 sys |
| drwxr-x--- 2 mysql mysql 4096 2 月 2 2021 test  [atguigu@hadoop102 lib]$ | | | | |

* 1. **测试 Binlog 是否开启**

1. 插入数据

INSERT INTO user\_info VALUES('1001','zhangsan','male');

1. 再次到/var/lib/mysql 目录下，查看index 文件的大小

[atguigu@hadoop102 lib]$ sudo ls -l mysql

总用量 474152

-rw-r-----. 1 mysql mysql 56 8 月 7 2020 auto.cnf

drwxr-x---. 2 mysql mysql 4096 9 月 25 2020 azkaban

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -rw --. | 1 | mysql | mysql | 1680 8 月 7 2020 ca-key.pem |
| -rw-r--r--. | 1 | mysql | mysql | 1112 8 月 7 2020 ca.pem |
| drwxr-x--- | 2 | mysql | mysql | 4096 8 月 18 16:56 cdc\_test |
| -rw-r--r--. | 1 | mysql | mysql | 1112 8 月 7 2020 client-cert.pem |
| -rw --. | 1 | mysql | mysql | 1676 8 月 7 2020 client-key.pem |
| drwxr-x---. | 2 | mysql | mysql | 4096 9 月 25 2020 gmall\_report |
| -rw-r----- | 1 | mysql | mysql | 1085 12 月 1 09:12 ib\_buffer\_pool |
| -rw-r-----. | 1 | mysql | mysql | 79691776 12 月 13 08:45 ibdata1 |
| -rw-r-----. | 1 | mysql | mysql | 50331648 12 月 13 08:45 ib\_logfile0 |
| -rw-r-----. | 1 | mysql | mysql | 50331648 12 月 13 08:45 ib\_logfile1 |
| -rw-r----- | 1 | mysql | mysql | 12582912 12 月 13 08:45 ibtmp1 |
| drwxr-x--- | 2 | mysql | mysql | 4096 9 月 22 15:30 maxwell |
| drwxr-x---. | 2 | mysql | mysql | 4096 8 月 12 2020 metastore |
| drwxr-x---. | 2 | mysql | mysql | 4096 9 月 22 15:43 mysql |
| -rw-r-----. | 1 | mysql | mysql | 452 12 月 13 08:45 mysql-bin.000001 |
| -rw-r----- | 1 | mysql | mysql | 19 12 月 13 08:45 mysql-bin.index |
| srwxrwxrwx | 1 | mysql | mysql | 0 12 月 13 08:45 mysql.sock |
| -rw------- | 1 | mysql | mysql | 5 12 月 13 08:45 mysql.sock.lock |
| drwxr-x---. | 2 | mysql | mysql | 4096 8 月 7 2020 performance\_schema |
| -rw --. | 1 | mysql | mysql | 1680 8 月 7 2020 private\_key.pem |
| -rw-r--r--. | 1 | mysql | mysql | 452 8 月 7 2020 public\_key.pem |
| -rw-r--r--. | 1 | mysql | mysql | 1112 8 月 7 2020 server-cert.pem |
| -rw --. | 1 | mysql | mysql | 1680 8 月 7 2020 server-key.pem |
| drwxr-x---. | 2 | mysql | mysql | 12288 8 月 7 2020 sys |
| drwxr-x--- 2 mysql mysql 4096 2 月 2 2021 test  [atguigu@hadoop102 lib]$ | | | | |

* 1. **赋权限**

在 MySQL 中执行

mysql> set global validate\_password\_length=4; mysql> set global validate\_password\_policy=0;

mysql> GRANT SELECT, REPLICATION SLAVE, REPLICATION CLIENT ON \*.\* TO

'canal'@'%' IDENTIFIED BY 'canal' ;

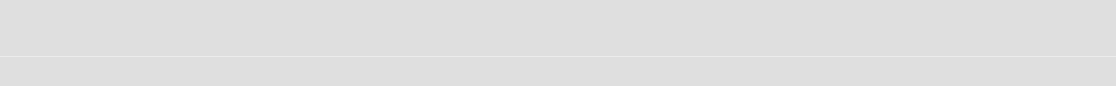
**第 3 章 Canal 的下载和安装**

* 1. **下载并解压 Jar 包**

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

我们直接/2.资料下的 canal.deployer-1.1.2.tar.gz 拷贝到/opt/sortware 目录下，然后解压到/opt/module/canal 包下

注意：canal 解压后是分散的，我们在指定解压目录的时候需要将canal 指定上



[atguigu@hadoop102 software]$ mkdir /opt/module/canal

[atguigu@hadoop102

/opt/module/canal

software]$

tar -zxvf canal.deployer-1.1.2.tar.gz -C

* 1. **修改 canal.properties 的配置**

[atguigu@hadoop102 conf]$ pwd

/opt/module/canal/conf

[atguigu@hadoop102 conf]$ vim canal.properties #################################################

######### common argument ############# #################################################

canal.id = 1 canal.ip = canal.port = 11111

canal.metrics.pull.port = 11112 canal.zkServers =

# flush data to zk canal.zookeeper.flush.period = 1000 canal.withoutNetty = false

# tcp, kafka, RocketMQ canal.serverMode = tcp

# flush meta cursor/parse position to file

说明：这个文件是 canal 的基本通用配置，canal 端口号默认就是 11111，修改 canal 的输出 model，默认 tcp，改为输出到kafka

多实例配置如果创建多个实例，通过前面 canal 架构，我们可以知道，一个 canal 服务中可以有多个instance，conf/下的每一个 example 即是一个实例，每个实例下面都有独立的配置文件。默认只有一个实例 example，如果需要多个实例处理不同的 MySQL 数据的话，直接拷贝出多个 example，并对其重新命名，命名和配置文件中指定的名称一致，然后修改 canal.properties 中的 canal.destinations=实例 1，实例 2，实例 3。

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

######### destinations ############# #################################################

canal.destinations = example

* 1. **修改instance.properties**

我们这里只读取一个 MySQL 数据，所以只有一个实例，这个实例的配置文件在

conf/example 目录下

[atguigu@hadoop102 example]$ pwd

/opt/module/canal/conf/example

[atguigu@hadoop102 example]$ vim instance.properties

* + 1. 配置 MySQL 服务器地址

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

## mysql serverId , v1.0.26+ will autoGen canal.instance.mysql.slaveId=20

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

# position info canal.instance.master.address=hadoop102:3306

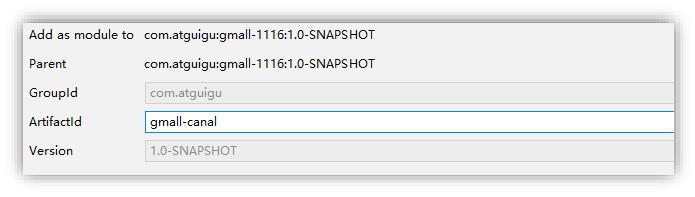
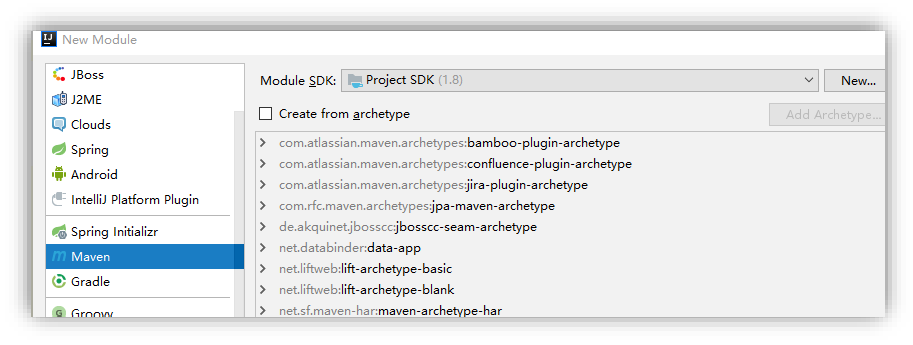
* + 1. 配置连接 MySQL 的用户名和密码，默认就是我们前面授权的canal

# username/password

canal.instance.dbUsername=canal canal.instance.dbPassword=canal canal.instance.connectionCharset = UTF-8 canal.instance.defaultDatabaseName =test # enable druid Decrypt database password canal.instance.enableDruid=false

# 第 4 章 实时监控测试

* 1. **TCP 模式测试**
     1. **创建 gmall-canal 项目**



* + 1. **在 gmall-canal 模块中配置 pom.xml**

<dependencies>

<dependency>

<groupId>com.alibaba.otter</groupId>

<artifactId>canal.client</artifactId>

<version>1.1.2</version>

</dependency>

<dependency>

<groupId>org.apache.kafka</groupId>

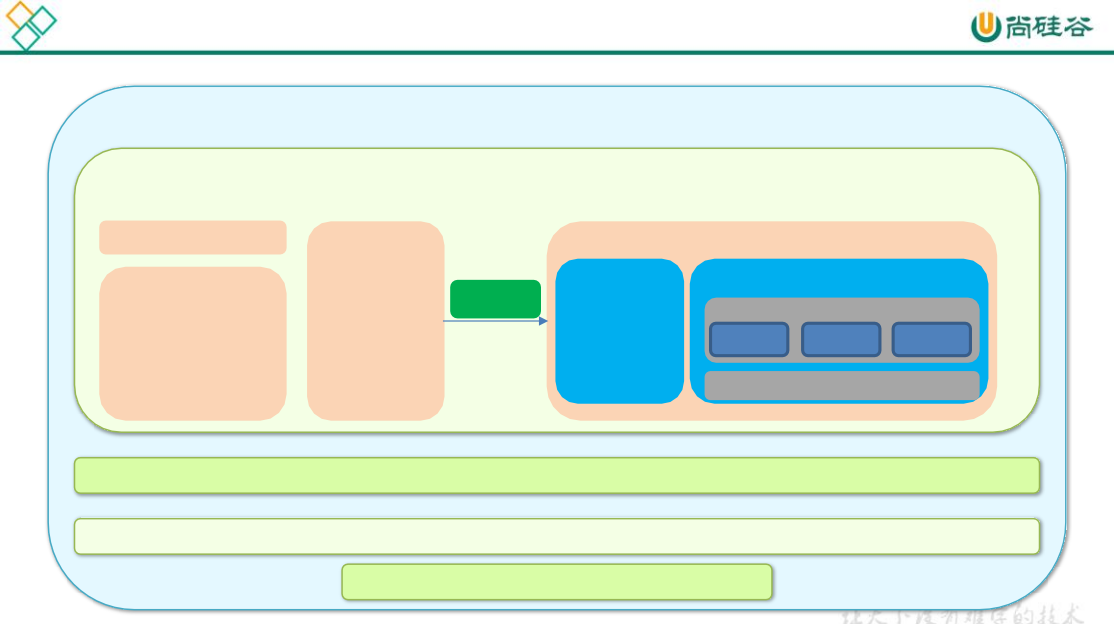
<artifactId>kafka-clients</artifactId>

<version>2.4.1</version>

</dependency>

</dependencies>

* + 1. **通用监视类 –CanalClient**
       1. Canal 封装的数据结构



Canal数据结构

**Message：一次canal从日志中抓取的信息,一个message可以包含多个sql执行的结果**

**Entry：对应一个sql命令，一个sql可能会对多行记录造成影响**

TableName

StoreValue

RowChange

**RowDataList**

EntryType

**EevntType**

序列化的 反序列化

不可直接使用

TRANSACTIONBEGIN

ROWDATA TRANSACTIONEND HEARTBEAT GTIDLOG

INSERT

UPDATE DELETE CREATE

… …

**… …**

**Entry**

**Entry**

……

**RowData**

column column column

* + - 1. 在 gmall-canal 模块下创建 com.atguigu.app 包，并在包下创建 CanalClien（t java 代码）

代码如下：

import com.alibaba.fastjson.JSONObject;

import com.alibaba.otter.canal.client.CanalConnector; import com.alibaba.otter.canal.client.CanalConnectors; import com.alibaba.otter.canal.protocol.CanalEntry; import com.alibaba.otter.canal.protocol.Message; import com.atguigu.constants.GmallConstants;

import com.atguigu.utils.MyKafkaSender; import com.google.protobuf.ByteString;

import com.google.protobuf.InvalidProtocolBufferException;

import java.net.InetSocketAddress; import java.util.List;

import java.util.Random;

public class CanalClient {

public static void main(String[] args) throws InvalidProtocolBufferException {

//1.获取 canal 连接对象

CanalConnector canalConnector = CanalConnectors.newSingleConnector(new InetSocketAddress("hadoop102", 11111), "example", "", "");

while (true) {

//2.获取连接 canalConnector.connect();

//3.指定要监控的数据库 canalConnector.subscribe("gmall.\*");

//4.获取 Message

Message message = canalConnector.get(100);

List<CanalEntry.Entry> entries = message.getEntries(); if (entries.size() <= 0) {

System.out.println("没有数据，休息一会"); try {

Thread.sleep(1000);

} catch (InterruptedException e) { e.printStackTrace();

}

} else {

for (CanalEntry.Entry entry : entries) {

//TODO 获取表名

String tableName = entry.getHeader().getTableName();

// TODO Entry 类型

CanalEntry.EntryType entryType = entry.getEntryType();

// TODO 判断 entryType 是否为 ROWDATA if

(CanalEntry.EntryType.ROWDATA.equals(entryType)) {

// TODO 序列化数据

ByteString storeValue = entry.getStoreValue();

// TODO 反序列化

CanalEntry.RowChange rowChange = CanalEntry.RowChange.parseFrom(storeValue);

//TODO 获取事件类型

CanalEntry.EventType eventType = rowChange.getEventType();

//TODO 获取具体的数据

List<CanalEntry.RowData> rowDatasList = rowChange.getRowDatasList();

//TODO 遍历并打印数据

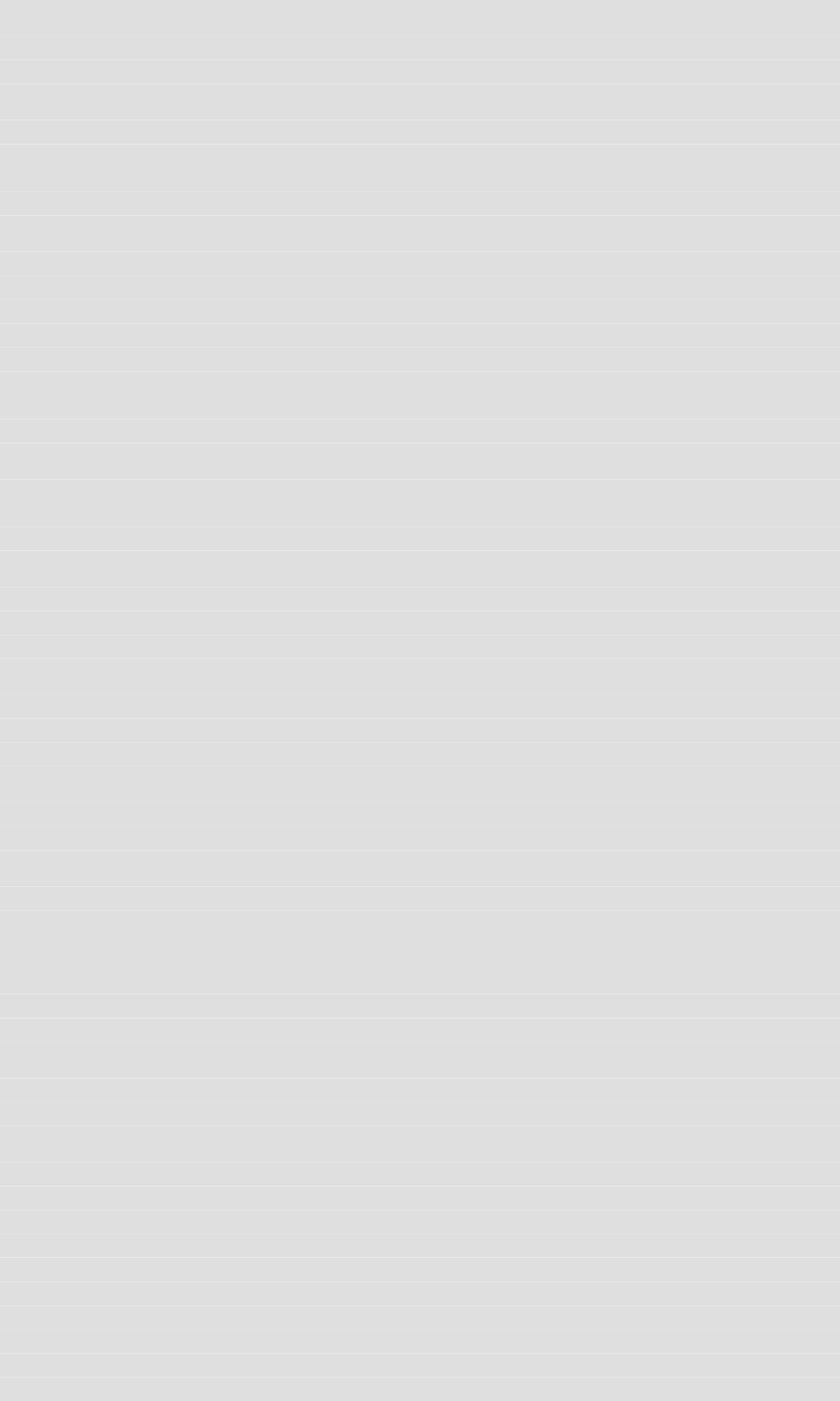
for (CanalEntry.RowData rowData : rowDatasList)

{

List<CanalEntry.Column> beforeColumnsList =

rowData.getBeforeColumnsList();

JSONObject beforeData = new JSONObject(); for (CanalEntry.Column column :

beforeColumnsList) { column.getValue());

beforeData.put(column.getName(),

}

JSONObject afterData = new JSONObject(); List<CanalEntry.Column> afterColumnsList =

rowData.getAfterColumnsList();

for (CanalEntry.Column column :

afterColumnsList) {

afterData.put(column.getName(),

column.getValue());

}

System.out.println("TableName:" + tableName

+

",EventType:" + eventType + ",After:" + beforeData + ",After:" + afterData);

}

}

}

}

}

}

}

* 1. **Kafka 模式测试**

1. 修改 canal.properties 中 canal 的输出 model，默认 tcp，改为输出到kafka

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

######### common argument ############# #################################################

canal.id = 1 canal.ip = canal.port = 11111

canal.metrics.pull.port = 11112 canal.zkServers =

# flush data to zk canal.zookeeper.flush.period = 1000 canal.withoutNetty = false

# tcp, kafka, RocketMQ canal.serverMode = kafka

# flush meta cursor/parse position to file

1. 修改 Kafka 集群的地址

################################################## ######### MQ ############# ##################################################

canal.mq.servers = hadoop102:9092,hadoop103:9092,hadoop104:9092

1. 修改 instance.properties 输出到 Kafka 的主题以及分区数

# mq config canal.mq.topic=canal\_test canal.mq.partitionsNum=1

# hash partition config #canal.mq.partition=0

#canal.mq.partitionHash=mytest.person:id,mytest.role:id

注意：默认还是输出到指定 Kafka 主题的一个 kafka 分区，因为多个分区并行可能会打乱 binlog 的顺序， 如果要提高并行度， 首先设置 kafka 的分区数>1, 然后设置 canal.mq.partitionHash 属性

1. 启动Canal

[atguigu@hadoop102 example]$ cd /opt/module/canal/ [atguigu@hadoop102 canal]$ bin/startup.sh

1. 看到 CanalLauncher 你表示启动成功，同时会创建 canal\_test 主题

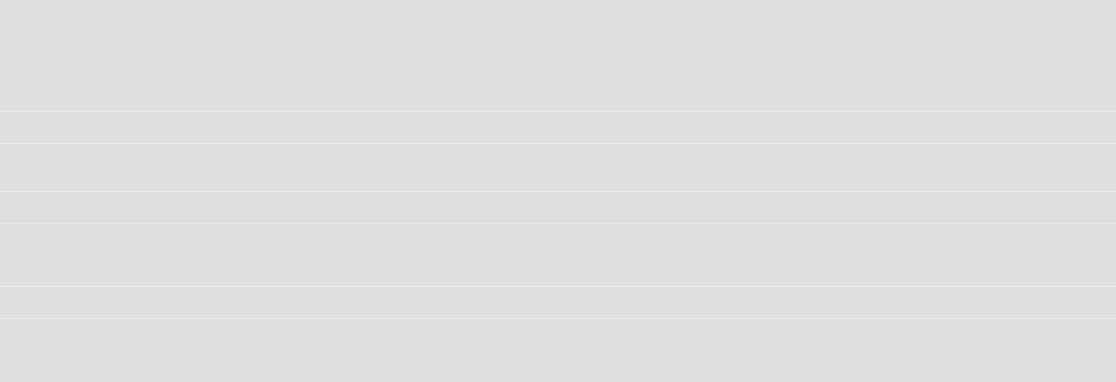
[atguigu@hadoop102 canal]$ jps 2269 Jps

2253 CanalLauncher [atguigu@hadoop102 canal]$

1. 启动 Kafka 消费客户端测试，查看消费情况

[atguigu@hadoop102 kafka]$ bin/kafka-console-consumer.sh --bootstrap-server hadoop102:9092 --topic canal\_test

1. 向 MySQL 中插入数据后查看消费者控制台



插入数据

INSERT

INTO

user\_info

VALUES('1001','zhangsan','male'),('1002','lisi','female');

Kafka 消费者控制台

{"data":[{"id":"1001","name":"zhangsan","sex":"male"},{"id":"1002 ","name":"lisi","sex":"female"}],"database":"gmall-

2021","es":1639360729000,"id":1,"isDdl":false,"mysqlType":{"id":"

varchar(255)","name":"varchar(255)","sex":"varchar(255)"},"old":n

ull,"sql":"","sqlType":{"id":12,"name":12,"sex":12},"table":"user

\_info","ts":1639361038454,"type":"INSERT"}