数据集成第19组实验报告

201250125 刘承杰

201250123 刘晓旭

201250152 赵凝晖

201250213 徐晨

1. 团队分工

201250125 刘承杰 数据库表部分

201250123 刘晓旭 环境搭建和数据分析

201250152 赵凝晖 流式数据部分

201250213 徐晨 可视化部分

1. 环境搭建
   1. 数据库表处理环境

数据库表的处理在虚拟机内进行，虚拟机参数如下：

|  |  |
| --- | --- |
| CPU | Intel Core i7-10875h@5.0GHZ\*12 |
| MEM | DDR4 3200MHZ 16GB |
| SSD | PCIE4.0 100GB |
| SYS | Ubuntu20.04 |

第一步是安装jdk8环境，选择的版本是jdk8u361-linux-x64。

将JDK解压至/usr/local/java，并在/etc/profile中添加如下几行：

export JAVA\_HOME=/usr/local/java

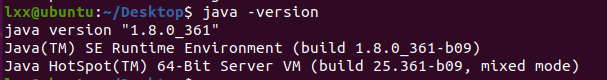
export JRE\_HOME=/usr/local/java/jre

PATH=$PATH:$JAVA\_HOME/bin:$JRE\_HOME/bin

然后添加软链接，重启后验证版本：

ln -s /usr/local/java/bin/java /usr/bin/java

java -version



第二步是安装hadoop环境，选择的版本是hadoop-2.7.4。

将hadoop解压至/usr/local/Hadoop，并在/etc/profile中添加如下几行：

export HADOOP\_HOME=/usr/local/hadoop

PATH=$PATH:$HADOOP\_HOME/bin

在path/to/hadoop/etc/hadoop/{hadoop-env.sh,yarn-env.sh}中添加：

export JAVA\_HOME=/usr/local/java

在path/to/hadoop/etc/hadoop/core-site.xml中添加以下代码：

然后添加临时文件夹目录。

<configuration><property> <name>fs.default.name</name> <value>hdfs://localhost:9111</value> <description>HDFS的URI，文件系统://namenode标识:端口号</description> </property> <property> <name>hadoop.tmp.dir</name> <value>/usr/local/hadoop/tmp</value> <description>namenode上本地的hadoop临时文件夹</description> </property> </configuration>

在path/to/hadoop/etc/hadoop/hdfs-site.xml中添加以下代码：

然后添加存储位置目录，但是保留一级，防止出现node启动失败问题。

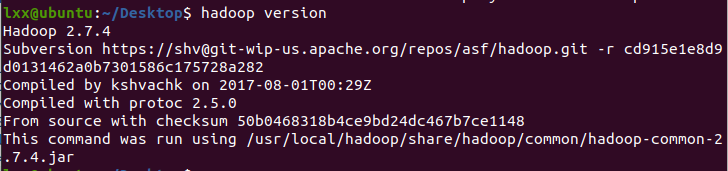
<configuration><property> <name>dfs.name.dir</name> <value>/usr/local/hadoop/hdfs/name</value> <description>namenode上存储hdfs名字空间元数据 </description> </property> <property> <name>dfs.data.dir</name> <value>/usr/local/hadoop/hdfs/data</value> <description>datanode上数据块的物理存储位置</description> </property> <property> <name>dfs.replication</name> <value>1</value> <description>副本个数，配置默认是3,应小于datanode机器数量</description> </property> </configuration>

而后，对mapred-site.xml和yarn-site的设置与教程中一致，但是在配置yarn-site.xml的时候取消了对resourcemanager.webapp.address的设置。

<configuration><property><name>mapreduce.framework.name</name> <value>yarn</value> </property> </configuration>

<configuration><property><name>yarn.nodemanager.aux-services</name> <value>mapreduce\_shuffle</value></property></configuration>

然后重启并验证版本。



第三步是安装spark环境，选择的版本是spark-2.3.3-bin-without-hadoop。

在解压后的conf/slave里添加一行：

localhost

在解压后的conf/spark-env.sh里添加一行：

export JAVA\_HOME=/usr/local/java

export HADOOP\_HOME=/usr/local/hadoop

export HADOOP\_CONF\_DIR=/usr/local/hadoop/etc/hadoop

export SPARK\_MASTER\_IP=192.168.1.29

export SPARK\_DIST\_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)

第四步是安装clickhouse，按照官方文档即可。

第五步是启动所有安装的进程。并在终端输入jps命令查看效果。

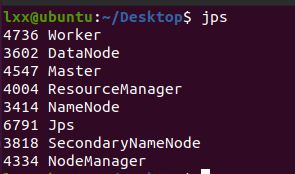
/usr/local/hadoop/sbin/start-dfs.sh

/usr/local/hadoop/sbin/start-yarn.sh

/usr/local/spark/sbin/start-all.sh

sudo service clickhouse-server start

clickhouse-client



第六步是安装maven环境。选择的版本是maven3.9.1。

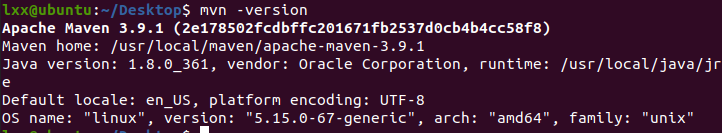
将其解压到/usr/local/maven，并在/etc/profile中添加如下内容：

export MAVEN\_HOME=/usr/local/maven/apache-maven-3.9.1

export CLASSPATH=${MAVEN\_HOME}/lib:$CLASSPATH

export PATH=${MAVEN\_HOME}/bin:$PATH

重启检验：



* 1. 流式数据获取环境

流式数据的获取在宿舍的台式机内进行，台式机参数如下：

|  |  |
| --- | --- |
| CPU | Intel Core i9-13900@5.1GHZ\*32 |
| MEM | DDR4 3200MHZ 32GB\*2 |
| SSD | PCIE4.0 2TB |
| SYS | Ubuntu22.04 |

台式机构建了idea IDE环境与jdk8环境（内嵌于IDE）。

在idea项目中又添加了Kafka与Slf4j依赖。

* 1. 流式数据处理环境

数据库表的处理在虚拟机内进行，虚拟机参数如下：

|  |  |
| --- | --- |
| CPU | Intel Core i7-10875h@5.0GHZ\*12 |
| MEM | DDR4 3200MHZ 16GB |
| SSD | PCIE4.0 100GB |
| SYS | Ubuntu20.04 |

第一步是安装jdk环境，与2.1中一致。

第二步是安装kafka环境，选择的版本是kafka\_2.11-2.1.0。

解压后，进入config下的zookeeper.properties，并添加以下内容：

dataDir=/usr/local/kafka/zookeeper/data

dataLogDir=/usr/local/kafka/zookeeper/logs

clientPort=2181

initLimit=20

syncLimit=10

tickTime=2000

然后进入config下的server.properties，并添加以下内容：

broker.id=0

listeners=PLAINTEXT://localhost:9092

num.network.threads=3

num.io.threads=8

socket.send.buffer.bytes=102400

socket.receive.buffer.bytes=102400

socket.request.max.bytes=104857600

log.dirs=/usr/local/kafka/kafka-logs

num.partitions=1

num.recovery.threads.per.data.dir=1

offsets.topic.replication.factor=1

transaction.state.log.replication.factor=1

transaction.state.log.min.isr=1

log.retention.hours=168

log.segment.bytes=1073741824

log.retention.check.interval.ms=300000

zookeeper.connect=localhost:2181

zookeeper.connection.timeout.ms=6000

group.initial.rebalance.delay.ms=0

第三步是安装flink环境，选择的版本是flink1.13.5-bin-scala\_2.11。

解压后，设置权限：

sudo mv ./flink-\*/ ./flink

sudo chown -R hadoop:hadoop ./flink

在/etc/profile中添加环境变量：

export FLINK\_HOME=/usr/local/flink

PATH=$FLINK\_HOME/bin:$PATH

第四步是安装maven，同2.1部分。

第五步是启动全部的依赖：

nohup bin/zookeeper-server-start.sh config/zookeeper.properties &

nohup bin/kafka-server-start.sh config/server.properties &

./bin/kafka-topics.sh --create --zookeeper 127.0.0.1:2181 --replication-factor 1 --partitions 1 --topic testtopic

./bin/kafka-topics.sh  --zookeeper 127.0.0.1:2181 --list

./bin/kafka-console-producer.sh --broker-list 127.0.0.1:9092 --topic testtopic

./bin/kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic testtopic --from-beginning

* 1. 可视化部分环境

在2.3部分的基础上进行。可视化部分使用的是tabix，一种集成到网页端的clickhouse管理工具。

第一步是安装nginx。

sudo apt-get install nginx

sudo service nginx start

第二步是进入nginx的配置文件/etc/nginx/nginx.conf，并修改为：

worker\_processes  1;

events {

    worker\_connections  1024;

}

http {

    include       mime.types;

    default\_type  application/octet-stream;

    sendfile        on;

server {

    listen 80;

    server\_name ui.tabix.io;

    charset        utf-8;

    root /home/lxx/Desktop/tabix/build;

    location / {

        if (!-f $request\_filename) {

            rewrite ^(.\*)$ /index.html last;

        }

        index  index.html index.htm;

    }

}

}

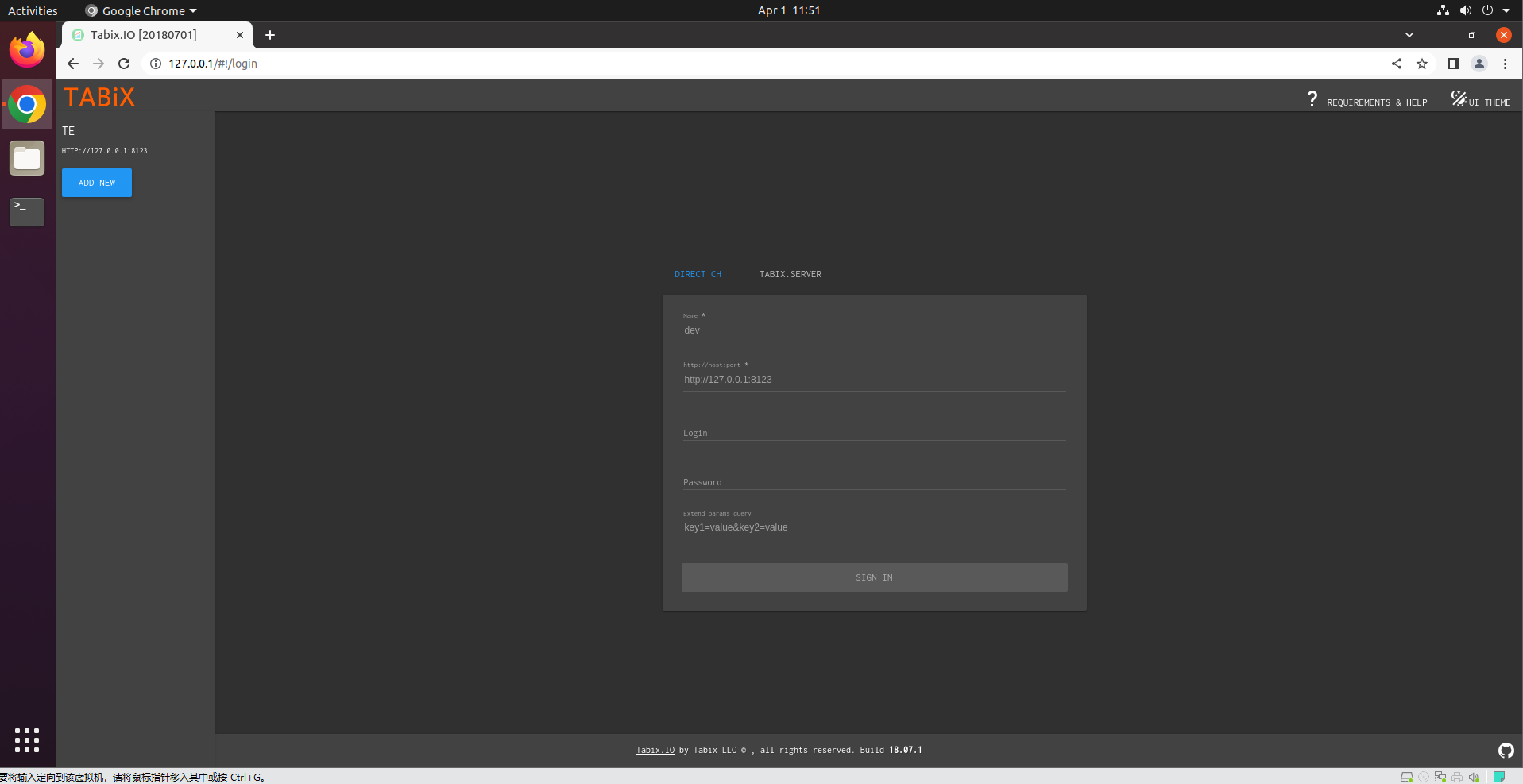
第三步，下载tabix并解压至上述目录：（我采用了外部拖拽的方式 wget也可以的）

第四步，重启nginx服务。

systemctl restart nginx

第五步，下载基于chromium的浏览器并安装，如google chrome。

最后在浏览器键入127.0.0.1即可。



1. 数据获取
   1. 数据库表数据获取（从远端服务器）

数据的获取方式大致与助教给的方法相同。

val conf = new SparkConf()

          .setAppName(this.getClass.getSimpleName)

          .setMaster("local[\*]")

        val session = SparkSession.builder()

          .config(conf)

          .getOrCreate()

        val reader = session.read.format("jdbc")

          .option("url", "jdbc:hive2://172.29.4.17:10000/default")

          .option("user", "student")

          .option("password", "nju2023")

          .option("driver", "org.apache.hive.jdbc.HiveDriver")

        val registerHiveDqlDialect = new RegisterHiveSqlDialect()

        registerHiveDqlDialect.register()

        val tblNameDsts = List("pri\_cust\_contact\_info")

        for (tblNameDst <- tblNameDsts) {

            var df = reader.option("dbtable", tblNameDst).load()

            val columnNames = df.columns.toList.map(name => name.substring(tblNameDst.length + 1)).toArray

            df = df.toDF(columnNames: \_\*)

            df.show(5, false)

            var newDf = df.groupBy("uid")

        }

* 1. 流式数据数据获取（从远端服务器）

我们参与了第一次和第二次的数据推送，获得了全部数据——第一次约600~700万条，第二次约2600~2700万条。

以下是获取流数据的代码：

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

        FileWriter fw = null;

        File f = new File("dataCollection2.txt");

        fw = new FileWriter(f,true);

        PrintWriter pw = new PrintWriter(fw);

        FileWriter fw2 = null;

        File f2 = new File("dataSpecified2.txt");

        fw2 = new FileWriter(f2,true);

        PrintWriter pw2= new PrintWriter(fw2);

        Properties props = new Properties();

        props.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "172.29.4.17:9092");

        props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        props.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        // GROUP\_ID请使用学号，不同组应该使用不同的GROUP。

        // 原因参考：https://blog.csdn.net/daiyutage/article/details/70599433

        props.put(ConsumerConfig.GROUP\_ID\_CONFIG, "201250123");

        // 原因参考：https://blog.csdn.net/matrix\_google/article/details/88658234

        props.put(ConsumerConfig.AUTO\_OFFSET\_RESET\_CONFIG,"earliest");

        props.put(CommonClientConfigs.SECURITY\_PROTOCOL\_CONFIG, "SASL\_PLAINTEXT");

        props.put(SaslConfigs.SASL\_MECHANISM, "PLAIN");

        props.put(SaslConfigs.SASL\_JAAS\_CONFIG,

                "org.apache.kafka.common.security.plain.PlainLoginModule required username=\"student\" password=\"nju2023\";");

        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);

        consumer.subscribe(Collections.singletonList("transaction"));

        // 会从最新数据开始消费

        while (true) {

            ConsumerRecords<String, String> records = consumer.poll(100);

            for (ConsumerRecord<String, String> record : records) {

                // 获取消息数据

                System.out.println(record.value());

                // 存储消息数据

                pw.println(record.value());

                pw.flush();

                fw.flush();

                // 获取消息头

                Header groupIdHeader = record.headers().lastHeader("groupId");

                if (Objects.nonNull(groupIdHeader)) {

                    byte[] groupId = groupIdHeader.value();

                    // 此处yourGroupId替换成你们组的组号

                    if(Arrays.equals("19".getBytes(), groupId)){

                        // 额外记录这条数据

                        pw2.println(record.value());

                        pw2.flush();

                        fw2.flush();

                    }

                }

            }

        }

    }

以下是添加的依赖：

<dependencies>

        <dependency>

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

            <artifactId>kafka\_2.13</artifactId>

            <version>3.1.0</version>

        </dependency>

        <dependency>

            <groupId>org.slf4j</groupId>

            <artifactId>slf4j-simple</artifactId>

            <version>1.7.25</version>

            <scope>compile</scope>

        </dependency>

    </dependencies>

以下是第一次的数据属性：



以下是第二次的数据属性：



接下来是特化数据（助教之前声明针对小组特别推送的数据）

以下是第一次的特化数据（只展示一条，详见链接：<https://lxx0921.obs.cn-north-4.myhuaweicloud.com/%E5%A4%A7%E4%B8%89%E4%B8%8B/DC/dataSpecified.txt>）

{"eventBody":{"tran\_date":"20210208","batch\_no":"\"\"","uid":"320926195502278599","is\_secu\_card":"1","acct\_no":"6230661372674355060","belong\_org":"320982034","ent\_acct":"4558560207925467863054","cust\_name":"贺叫抛","ent\_name":"盐城市大丰区新丰镇引水村村民委员会","eng\_cert\_no":"3209820342013152","tran\_amt":"8532.00","trna\_channel":"WYDF","tran\_log\_no":"320982052K040003571","etl\_dt":"2021-02-09"},"eventType":"gzdf","eventDate":"2021-02-09"}

以下是第二次的特化数据（只展示一条，详见链接：<https://lxx0921.obs.cn-north-4.myhuaweicloud.com/%E5%A4%A7%E4%B8%89%E4%B8%8B/DC/dataSpecified2.txt>）

{"eventBody":{"dr\_amt":"0.00","sys\_date":"20210119","curr\_type":"CNY","oppo\_acct\_no":"3209823401109000837221","dscrp\_code":"1948","remark":"大丰新丰支行","tran\_time":"105944993","channel\_flag":"POS","bal":"11307.49","uid":"320926196905010276","card\_no":"6224521393674944524","remark\_1":"","oppo\_cust\_name":"","tran\_teller\_no":"320982034P04","cust\_name":"莫休龄","cr\_amt":"490.00","tran\_card\_no":"6224521393674944524","etl\_dt":"2021-01-19","oppo\_bank\_no":"","open\_org":"320982034","tran\_date":"20210119","tran\_code":"8002","tran\_type":"转帐","agt\_cust\_name":"","agt\_cert\_no":"","det\_n":"18991","acct\_no":"4558561527549467328872","src\_dt":"20210119","tran\_amt":"490.00","tran\_log\_no":"320982034P040000006","agt\_cert\_type":"X"},"eventType":"sa","eventDate":"2021-01-19"}

* 1. 本地流式数据获取

本地流式数据的获取采用了从文件中读取行并将其转换为csv格式的方法。这里以producer的一段代码举例：

public class kProducer {

    public static void main(String[] args) {

        Properties props = new Properties();

        //kafka 集群，broker-list

        props.put("bootstrap.servers", "127.0.0.1:9092");

        props.put("acks", "0");

        //重试次数

        props.put("retries", 10);

        //批次大小

        props.put("batch.size", 16384);

        //等待时间

        props.put("linger.ms", 1);

        //RecordAccumulator 缓冲区大小

        props.put("buffer.memory", 33554432);

        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        try (Producer<String, String> producer = new KafkaProducer<>(props)) {

            BufferedReader reader;

            reader = new BufferedReader(new FileReader("/home/lxx/Desktop/dataCollection2-1.txt"));

            String line = reader.readLine();

            int i=0;

            while (line != null) {

                String eve\_type\_date = line.substring(line.indexOf("eventType"));

                int a = eve\_type\_date.indexOf("\"", 10);

                int b = eve\_type\_date.indexOf("\"", 12);

                String eve\_type = eve\_type\_date.substring(a + 1, b);

                i=i+1;

                System.out.println(i);

                if (eve\_type.equals("sa")) {

                    int uid\_pos=line.indexOf("uid");

                    int card\_no\_pos=line.indexOf("card\_no");

                    int det\_n\_pos=line.indexOf("det\_n");

                    int curr\_type\_pos=line.indexOf("curr\_type");

                    int tran\_teller\_no\_pos=line.indexOf("tran\_teller\_no");

                    int cr\_amt\_pos=line.indexOf("cr\_amt");

                    int tran\_amt\_pos=line.indexOf("tran\_amt");

                    int tran\_card\_no\_pos=line.indexOf("tran\_card\_no");

                    int tran\_type\_pos=line.indexOf("tran\_type");

                    int tran\_log\_no\_pos=line.indexOf("tran\_log\_no");

                    int dr\_amt\_pos=line.indexOf("dr\_amt");

                    int open\_org\_pos=line.indexOf("open\_org");

                    int dscrp\_code\_pos=line.indexOf("dscrp\_code");

                    int remark\_pos=line.indexOf("remark");

                    int tran\_time\_pos=line.indexOf("tran\_time");

                    int tran\_date\_pos=line.indexOf("tran\_date");

                    int sys\_date\_pos=line.indexOf("sys\_date");

                    int tran\_code\_pos=line.indexOf("tran\_code");

                    int remark\_1\_pos=line.indexOf("remark\_1");

                    int oppo\_cust\_name\_pos=line.indexOf("oppo\_cust\_name");

                    int cust\_name\_pos=line.indexOf("cust\_name",oppo\_cust\_name\_pos+20);

                    int agt\_cert\_type\_pos=line.indexOf("agt\_cert\_type");

                    int agt\_cert\_no\_pos=line.indexOf("agt\_cert\_no");

                    int agt\_cust\_name\_pos=line.indexOf("agt\_cust\_name");

                    int channel\_flag\_pos=line.indexOf("channel\_flag");

                    int bal\_pos=line.indexOf("bal",channel\_flag\_pos);

                    int oppo\_acct\_no\_pos=line.indexOf("oppo\_acct\_no");

                    int acct\_no\_pos=line.indexOf("acct\_no",oppo\_acct\_no\_pos+20);

                    int oppo\_bank\_no\_pos=line.indexOf("oppo\_bank\_no");

                    int src\_dt\_pos=line.indexOf("src\_dt");

                    int etl\_dt\_pos=line.indexOf("etl\_dt");

                    int eve\_pos=line.indexOf("eventType");

                    String dr\_amt=line.substring(dr\_amt\_pos+9,sys\_date\_pos-3);

                    String sys\_date=line.substring(sys\_date\_pos+11,curr\_type\_pos-3);

                    String curr\_type=line.substring(curr\_type\_pos+12,oppo\_acct\_no\_pos-3);

                    String oppo\_acct\_no=line.substring(oppo\_acct\_no\_pos+15,dscrp\_code\_pos-3);

                    String dscrp\_node=line.substring(dscrp\_code\_pos+13,remark\_pos-3);

                    String remark=line.substring(remark\_pos+9,tran\_time\_pos-3);

                    String tran\_time=line.substring(tran\_time\_pos+12,channel\_flag\_pos-3);

                    String channel\_flag=line.substring(channel\_flag\_pos+15,bal\_pos-3);

                    String bal=line.substring(bal\_pos+6,uid\_pos-3);

                    String uid=line.substring(uid\_pos+6,card\_no\_pos-3);

                    String card\_no=line.substring(card\_no\_pos+10,remark\_1\_pos-3);

                    String remark\_1=line.substring(remark\_1\_pos+11,oppo\_cust\_name\_pos-3);

                    String oppo\_cust\_name=line.substring(oppo\_cust\_name\_pos+17,tran\_teller\_no\_pos-3);

                    String tran\_teller\_no=line.substring(tran\_teller\_no\_pos+17,cust\_name\_pos-3);

                    String cust\_name=line.substring(cust\_name\_pos+12,cr\_amt\_pos-3);

                    String cr\_amt=line.substring(cr\_amt\_pos+9,tran\_card\_no\_pos-3);

                    String tran\_card\_no=line.substring(tran\_card\_no\_pos+15,etl\_dt\_pos-3);

                    String etl\_dt=line.substring(etl\_dt\_pos+9,oppo\_bank\_no\_pos-3);

                    String oppo\_bank\_no=line.substring(oppo\_bank\_no\_pos+15,open\_org\_pos-3);

                    String open\_org=line.substring(open\_org\_pos+11,tran\_date\_pos-3);

                    String tran\_date=line.substring(tran\_date\_pos+12,tran\_code\_pos-3);

                    String tran\_code=line.substring(tran\_code\_pos+12,tran\_type\_pos-3);

                    String tran\_type=line.substring(tran\_type\_pos+12,agt\_cust\_name\_pos-3);

                    String agt\_cust\_name=line.substring(agt\_cust\_name\_pos+16,agt\_cert\_no\_pos-3);

                    String agt\_cert\_no=line.substring(agt\_cert\_no\_pos+14,det\_n\_pos-3);

                    String det\_n=line.substring(det\_n\_pos+8,acct\_no\_pos-3);

                    String acct\_no=line.substring(acct\_no\_pos+10,src\_dt\_pos-3);

                    String src\_dt=line.substring(src\_dt\_pos+9,tran\_amt\_pos-3);

                    String tran\_amt=line.substring(tran\_amt\_pos+11,tran\_log\_no\_pos-3);

                    String tran\_log\_no=line.substring(tran\_log\_no\_pos+14,agt\_cert\_type\_pos-3);

                    String agt\_cert\_type=line.substring(agt\_cert\_type\_pos+16,eve\_pos-4);

                    String res="("+"'"+uid+"'"+","+"'"+card\_no+"'"+","+"'"+cust\_name+"'"+","+"'"+acct\_no+"'"+","+det\_n+","+"'"+curr\_type+"'"+","+"'"+tran\_teller\_no+"'"+","+cr\_amt+","+bal+","+tran\_amt+","+"'"+tran\_card\_no+"'"+","+"'"+tran\_type+"'"+","+"'"+tran\_log\_no+"'"+","+dr\_amt+","+"'"+open\_org+"'"+","+"'"+dscrp\_node+"'"+","+"'"+remark+"'"+","+"'"+tran\_time+"'"+","+"'"+tran\_date+"'"+","+"'"+sys\_date+"'"+","+"'"+tran\_code+"'"+","+"'"+remark\_1+"'"+","+"'"+oppo\_cust\_name+"'"+","+"'"+agt\_cert\_type+"'"+","+"'"+agt\_cert\_no+"'"+","+"'"+agt\_cust\_name+"'"+","+"'"+channel\_flag+"'"+","+"'"+oppo\_acct\_no+"'"+","+"'"+oppo\_bank\_no+"'"+","+"'"+src\_dt+"'"+","+"'"+etl\_dt+"'"+")";

                    ProducerRecord<String, String> record = new ProducerRecord<>("testtopic", res);

                    producer.send(record);

//                    System.out.println(res);

                } else if (eve\_type.equals("shop")) { ……

                line = reader.readLine();

            }

            reader.close();

        } catch (IOException e) {

            throw new RuntimeException(e);

        }

    }

    public static void removeFirstLine(String fileName) throws IOException {

        RandomAccessFile raf = new RandomAccessFile(fileName, "rw");

        //Initial write position

        long writePosition = raf.getFilePointer();

        raf.readLine();

        // Shift the next lines upwards.

        long readPosition = raf.getFilePointer();

        byte[] buff = new byte[1024];

        int n;

        while (-1 != (n = raf.read(buff))) {

            raf.seek(writePosition);

            raf.write(buff, 0, n);

            readPosition += n;

            writePosition += n;

            raf.seek(readPosition);

        }

        raf.setLength(writePosition);

        raf.close();

    }

}

用最繁琐但也是速度最快的方法高效地将数据转换成clickhouse的插入语方式并发送，以500万条的测试数据为例，共用时1分30秒，速度达到约每秒6万条。

用这样的办法错误率也很低，还是以500万条的测试数据为例，错误率约为3/5000000。

1. 数据转换
   1. 数据库表部分

首先是小作业的数据转换示例。小作业要求将con\_type为TEL OTH MOB的contact合并在contact\_phone字段，将其他类型合并在contact\_address字段。

 for (tblNameDst <- tblNameDsts) {

            var df = reader.option("dbtable", tblNameDst).load()

            val columnNames = df.columns.toList.map(name => name.substring(tblNameDst.length + 1)).toArray

            df = df.toDF(columnNames: \_\*)

            df.show(5, false)

            var newDf = df.groupBy("uid")

              .agg(

                  concat\_ws(",", collect\_list(when(col("con\_type") === "TEL" || col("con\_type") === "OTH" || col("con\_type") === "MOB", col("contact")).otherwise(null))).as("contact\_phone"),

                  concat\_ws(",", collect\_list(when(col("con\_type") =!= "TEL" && col("con\_type") =!= "OTH" && col("con\_type") =!= "MOB", col("contact")).otherwise(null))).as("contact\_address")

              )

        }

对于其它九张表，我们采取了按表特性进行不同处理的方式。

首先是第一张表djk\_info。采用去重空、清除bal列、按名字姓氏笔画排序的方法。

val transformedDf\_djk\_info = df

.filter(col("acct\_no").isNotNull)

.dropDuplicates()

.drop("bal")

.sort(col("cust\_name").asc)

djkfq\_info。采用去重空、删除mge\_org和recom\_no、和按姓氏笔划排序的方法。

val transformedDf\_djkfq\_info = df

                 .filter(col("acct\_no").isNotNull)

.dropDuplicates()

.drop("mge\_org")

.drop("recom\_no")

.sort(col("cust\_name").asc)

credit\_info。采用去重空、根据信用数据判断是否有信用评级的方法。

val transformedDf\_credit\_info = df

                 .filter(col("credit\_level").isNotNull)

.dropDuplicates()

.withColumn("is\_credited", when(col("credit\_level") === -1, "N").otherwise("Y"))

asset\_acct\_info。采用去重空、去列（期限、利率、自动转存标志、到期日期）。

val transformedDf\_asset\_acct\_info = df

.filter(col("cust\_no").isNotNull)

.dropDuplicates()

.drop("term")

.drop("rate")

.drop("auto\_dp\_flg")

.drop("matu\_date")

asset\_info/base\_info/liab\_acct\_info/liab\_info。去重空。

val transformedDf\_asset\_info = df

                        .filter(col("cust\_no").isNotNull)

                        .filter(col("cust\_name").isNotNull)

                        .dropDuplicates()

        val transformedDf\_base\_info = df

                        .filter(col("cert\_type").isNotNull)

.filter(col("cert\_type") =!= "未说明")

                        .filter(col("cust\_no").isNotNull)

                        .dropDuplicates()

        val transformedDf\_liab\_acct\_info = df

                        .filter(col("belong\_org").isNotNull)

                        .filter(col("exam\_org").isNotNull)

                        .dropDuplicates()

        val transformedDf\_liab\_info = df

                        .filter(col("all\_bal").isNotNull)

                        .filter(col("bad\_bal").isNotNull)

                        .dropDuplicates()

star\_info。去重空，判断是否已有星级评级。

val transformedDf\_star\_info = df

                     .filter(col("star\_level").isNotNull)

.withColumn("is\_starred", when(col("star\_level") === -1, "N").otherwise("Y"))

.dropDuplicates()

* 1. 流式数据部分

流式数据采用的方法相对简单，只是将脏数据进行丢弃，这里的脏数据指的是不符合原本规范的数据。这里指出一条：

第25784577条：

{"eventBody":{"reg\_org":"","receipt\_no":"0970920211021646355","pay\_cyc":"X","intr\_type":"1","pay\_freq":"07","owed\_int\_out":"0.00","bal":"0.00","owed\_int\_in":"0.00","operator":"I078309","extend\_times":"0","cust\_no":"133455113269","buss\_rate":"","vouch\_type":"信用/免担保","fine\_intr\_int":"0.00","pay\_back\_acct":"6230661372679355164","etl\_dt":"2022-03-08","actu\_buss\_rate":"5.6600000000","putout\_acct":"6230661372679355164","fine\_pr\_int":"0.00","mgr\_no":"I078309","pay\_times":"0","loan\_use":"","dlay\_days":"0","actu\_matu\_date":"20211221","occur\_date":"","buss\_type":"2060000005659","mge\_org":"320982062","acct\_no":"3209820624010000024624","dlay\_amt":"0.00","matu\_date":"20211221","norm\_bal":"0.00","dull\_amt":"0.0000000000","bad\_debt\_amt":"0.0000000000","loan\_channel":"手机银行","curr\_type":"CNY","uid":"320982199006221728","operate\_org":"320982062","buss\_amt":"10000.00","putout\_date":"20211021","cust\_name":"乔赴酿","due\_intr\_days":"0","pay\_type":"正常收回","ten\_class":"QL0101","contract\_no":"BC097202108040002123","pay\_acct":"","src\_dt":"20211128","subject\_no":"13011105","loan\_cust\_no":"CM2018020200000338","intr\_cyc":"M1","register":""},"eventType":"duebill","eventDate":"2022-03-08"}

这里的bad\_debt\_amt顺序与之前的流式数据不一致，采用丢弃处理。其余的数据写入即可。

1. 数据存储
   1. 数据库表部分

以小作业为例，其它表的方法大同小异，不做赘述。

val prop = new Properties()

            prop.setProperty("user", "default")

            prop.setProperty("password", "root")

            prop.setProperty("driver", "com.clickhouse.jdbc.ClickHouseDriver")

          newDf.coalesce(1)

              .write

              .mode(SaveMode.Overwrite)

              .option("batchsize", "2000")

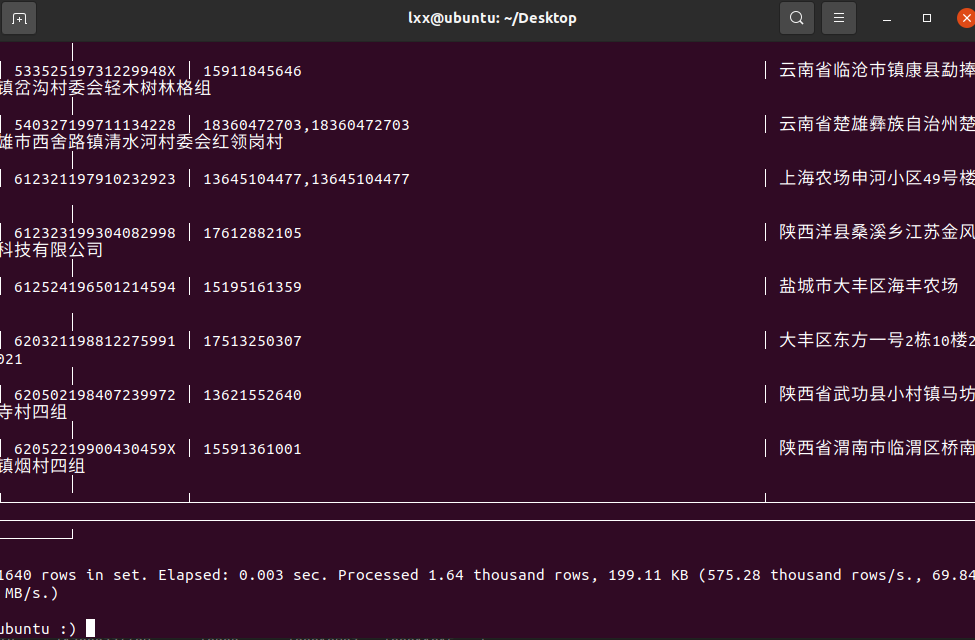
              .option("isolationLevel", "NONE")

              .option("numPartitions", "1")

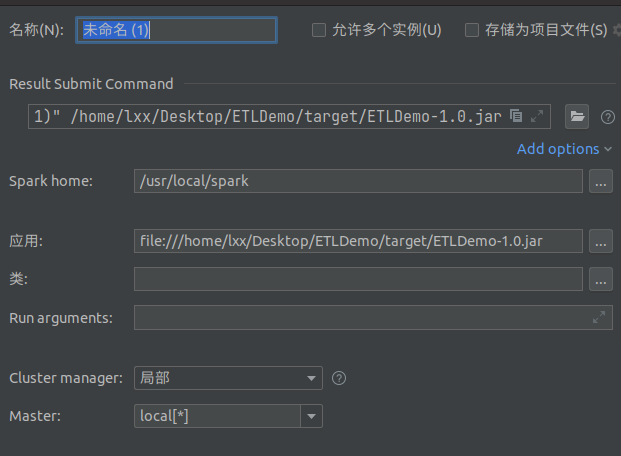
              .option("createTableOptions", "ENGINE=MergeTree() ORDER BY uid")

              .jdbc("jdbc:clickhouse://127.0.0.1:8123", "contact\_info", prop)

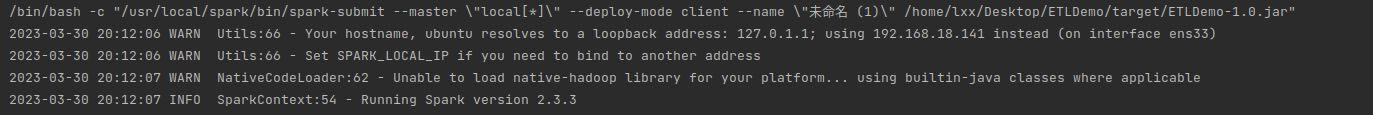
以下是select \* from contact\_info的结果：

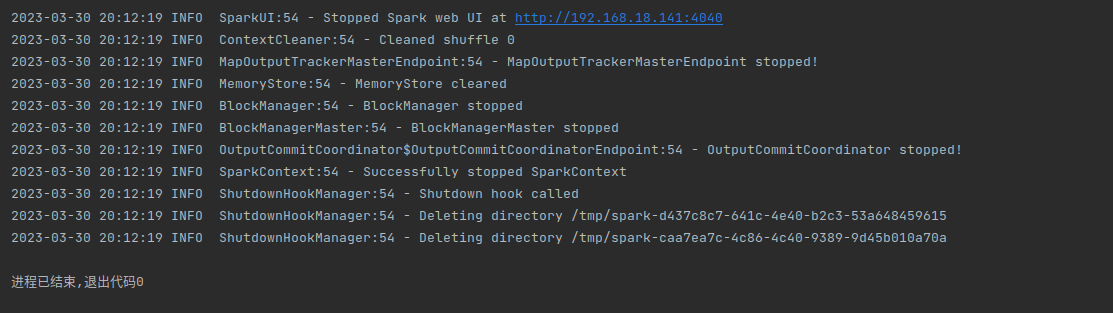


对于spark-submit提交jar包，以下是运行配置：



以下是运行结果：





需要额外注意，当建立另外九张表时，需要设定nullable，因为clickhouse在写入时默认不能“set null to non-null column”。详细的建表语句详见作业提交附件的hive-clickhouse.sql。

* 1. 流式数据部分

流式数据的存储分为以下几个部分：

首先要对原数据进行分割，数据太大了，出现问题不容易查错。附加文件中的cast文件夹提供了分割方法：

import java.io.\*;

public class KCast {

    public static void main(String[] args) {

        splitFileByLine("D:\\windows\\desktop\\流式数据\\dataCollection2.txt","D:\\windows\\desktop\\流式数据\\",5000000);

    }

    /\*\*

     \* 按行分割文件

     \* @param sourceFilePath 为源文件路径

     \* @param targetDirectoryPath 文件分割后存放的目标目录

     \* @param rows 为多少行一个文件

     \*/

    public static int splitFileByLine(String sourceFilePath, String targetDirectoryPath, int rows) {

        String sourceFileName = sourceFilePath.substring(sourceFilePath.lastIndexOf(File.separator) + 1, sourceFilePath.lastIndexOf("."));//源文件名

        String splitFileName = targetDirectoryPath + File.separator + sourceFileName + "-%s.txt";//切割后的文件名

        File targetDirectory = new File(targetDirectoryPath);

        if (!targetDirectory.exists()) {

            targetDirectory.mkdirs();

        }

        PrintWriter pw = null;//字符输出流

        String tempLine;

        int lineNum = 0;//本次行数累计 , 达到rows开辟新文件

        int splitFileIndex = 1;//当前文件索引

        try (BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(sourceFilePath)))) {

            pw = new PrintWriter(String.format(splitFileName, splitFileIndex));

            while ((tempLine = br.readLine()) != null) {

                if (lineNum > 0 && lineNum % rows == 0) {//需要换新文件

                    pw.flush();

                    pw.close();

                    pw = new PrintWriter(String.format(splitFileName , ++splitFileIndex));

                }

                pw.write(tempLine + "\n");

                lineNum++;

            }

            return splitFileIndex;

        } catch (Exception e) {

            e.printStackTrace();

            return -1;

        }finally {

            if (null != pw) {

                pw.flush();

                pw.close();

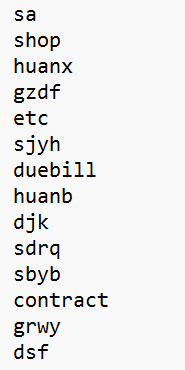
            }

        }

    }

}

然后是对数据库表的整定。数据库表总共有以下十四种：



其中，原来建表语句中contract项的is\_oth\_vouch不存在，需要删除重新建表。更改后的建表文件在附加文件中的real.sql。

最后是将数据写入clickhouse。代码如下：

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

        //make environment

        final StreamExecutionEnvironment env = StreamExecutionEnvironment.createLocalEnvironment();

        Map<String, String> globalParameters = new HashMap<>();

        // ClickHouse cluster properties

        globalParameters.put(ClickHouseClusterSettings.CLICKHOUSE\_HOSTS, "http://localhost:8123/");

        globalParameters.put(ClickHouseClusterSettings.CLICKHOUSE\_USER, "default");

        globalParameters.put(ClickHouseClusterSettings.CLICKHOUSE\_PASSWORD, "root");

        // sink common

        globalParameters.put(ClickHouseSinkConst.TIMEOUT\_SEC, "1");

        globalParameters.put(ClickHouseSinkConst.FAILED\_RECORDS\_PATH, "d:/");

        globalParameters.put(ClickHouseSinkConst.NUM\_WRITERS, "2");

        globalParameters.put(ClickHouseSinkConst.NUM\_RETRIES, "2");

        globalParameters.put(ClickHouseSinkConst.QUEUE\_MAX\_CAPACITY, "2");

        globalParameters.put(ClickHouseSinkConst.IGNORING\_CLICKHOUSE\_SENDING\_EXCEPTION\_ENABLED, "false");

        // set global parameters

        ParameterTool parameters = ParameterTool.fromMap(globalParameters);

        env.getConfig().setGlobalJobParameters(parameters);

        env.setParallelism(1);

        //set flink-kafka properties

        Properties properties = new Properties();

        properties.setProperty(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

        properties.setProperty(ConsumerConfig.GROUP\_ID\_CONFIG, "test-group");

        //set flink-kafka consumer && fetch data

        FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>("testtopic", new SimpleStringSchema(), properties);

        DataStream<String> inputStream = env.addSource(consumer);

        inputStream.print();

        String[] chart\_array\_info={"dm.dm\_v\_tr\_sa\_mx","dm.v\_tr\_shop\_mx","dm.dm\_v\_tr\_huanx\_mx","dm.dm\_v\_tr\_gzdf\_mx","dm.dm\_v\_tr\_etc\_mx","dm.dm\_v\_tr\_sjyh\_mx","dm.dm\_v\_tr\_duebill\_mx","dm.dm\_v\_tr\_huanb\_mx","dm.dm\_v\_tr\_djk\_mx","dm.dm\_v\_tr\_sdrq\_mx","dm.dm\_v\_tr\_sbyb\_mx","dm.dm\_v\_tr\_contract\_mx","dm.dm\_v\_tr\_grwy\_mx","dm.dm\_v\_tr\_dsf\_mx"};

        Properties props0 = new Properties();

        props0.put(ClickHouseSinkConst.TARGET\_TABLE\_NAME, chart\_array\_info[0]);

        props0.put(ClickHouseSinkConst.MAX\_BUFFER\_SIZE, "10000");

        ClickHouseSink sink = new ClickHouseSink(props0);

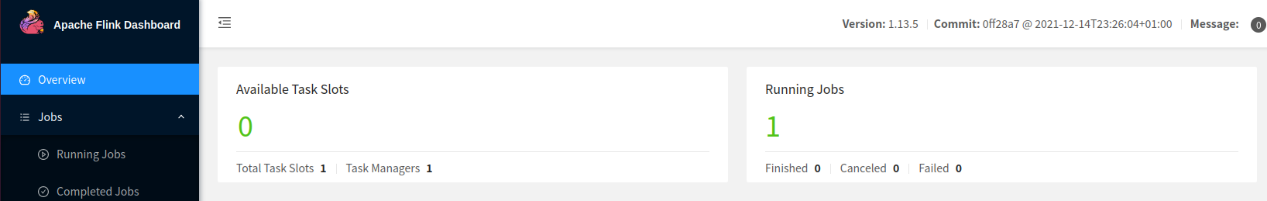
        inputStream.addSink(sink);

        //make environment execute

        env.execute("Flink Kafka Consumer");

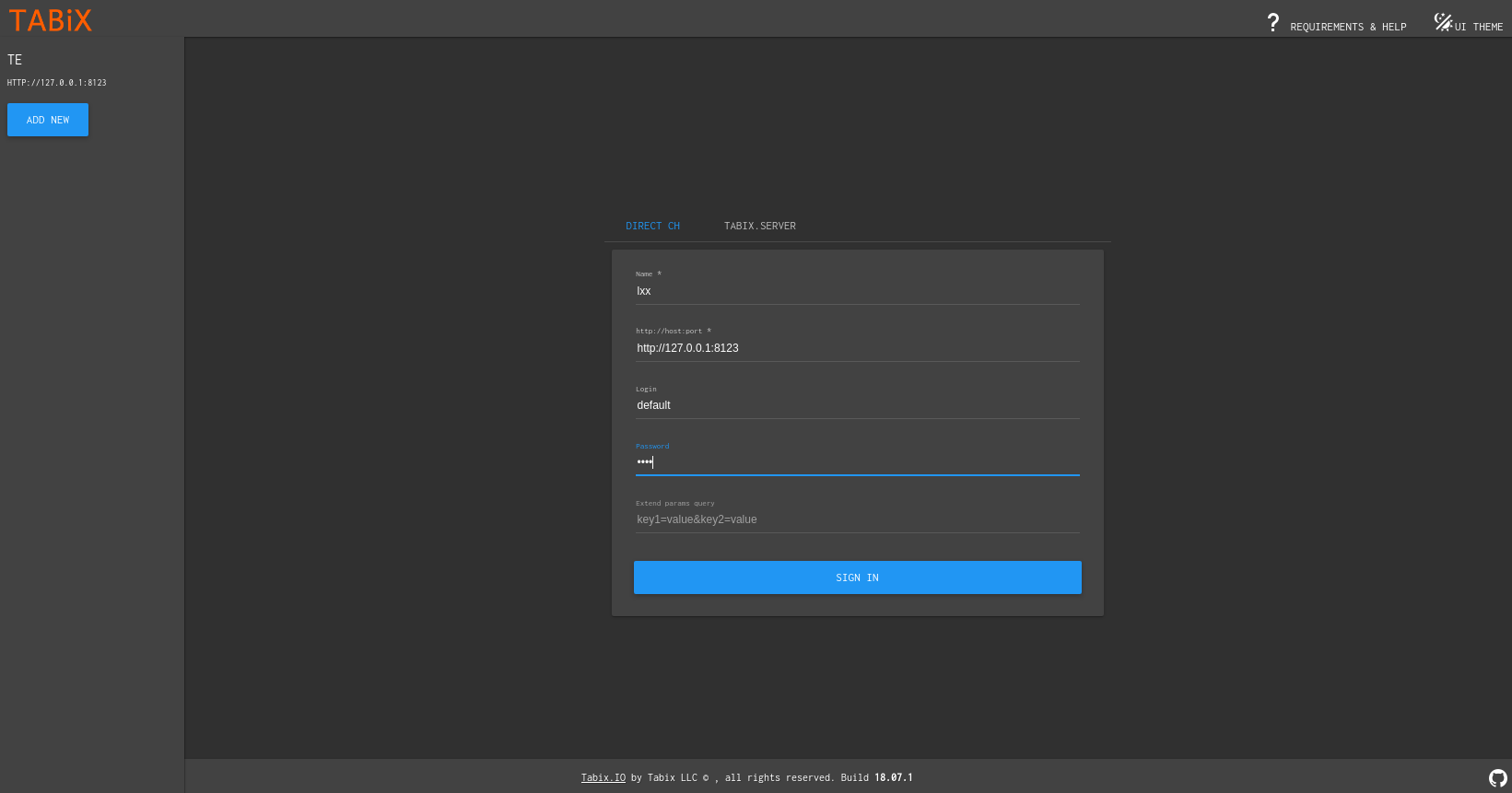
    }

最后是使用WEB UI提交。

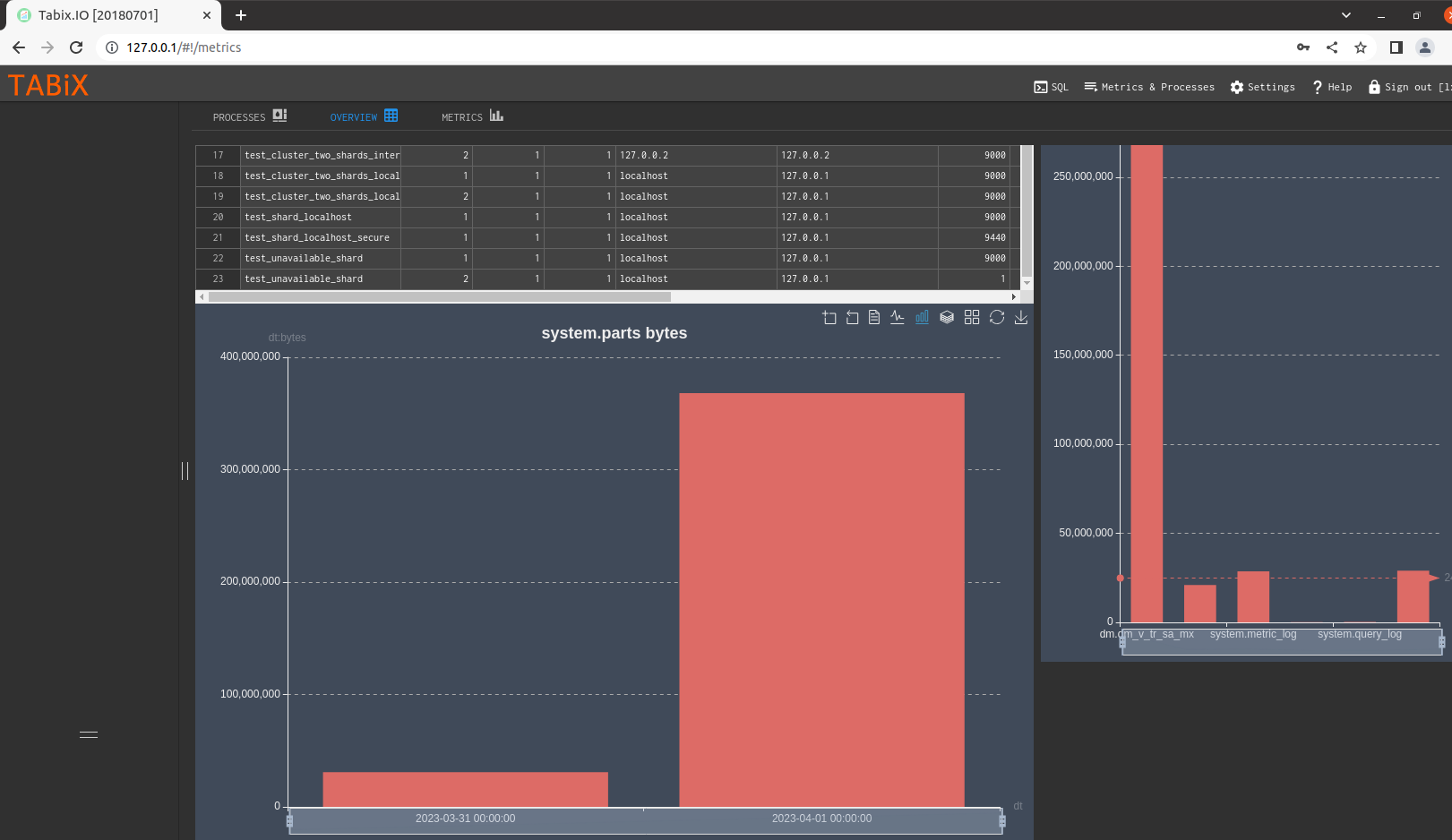


1. 可视化

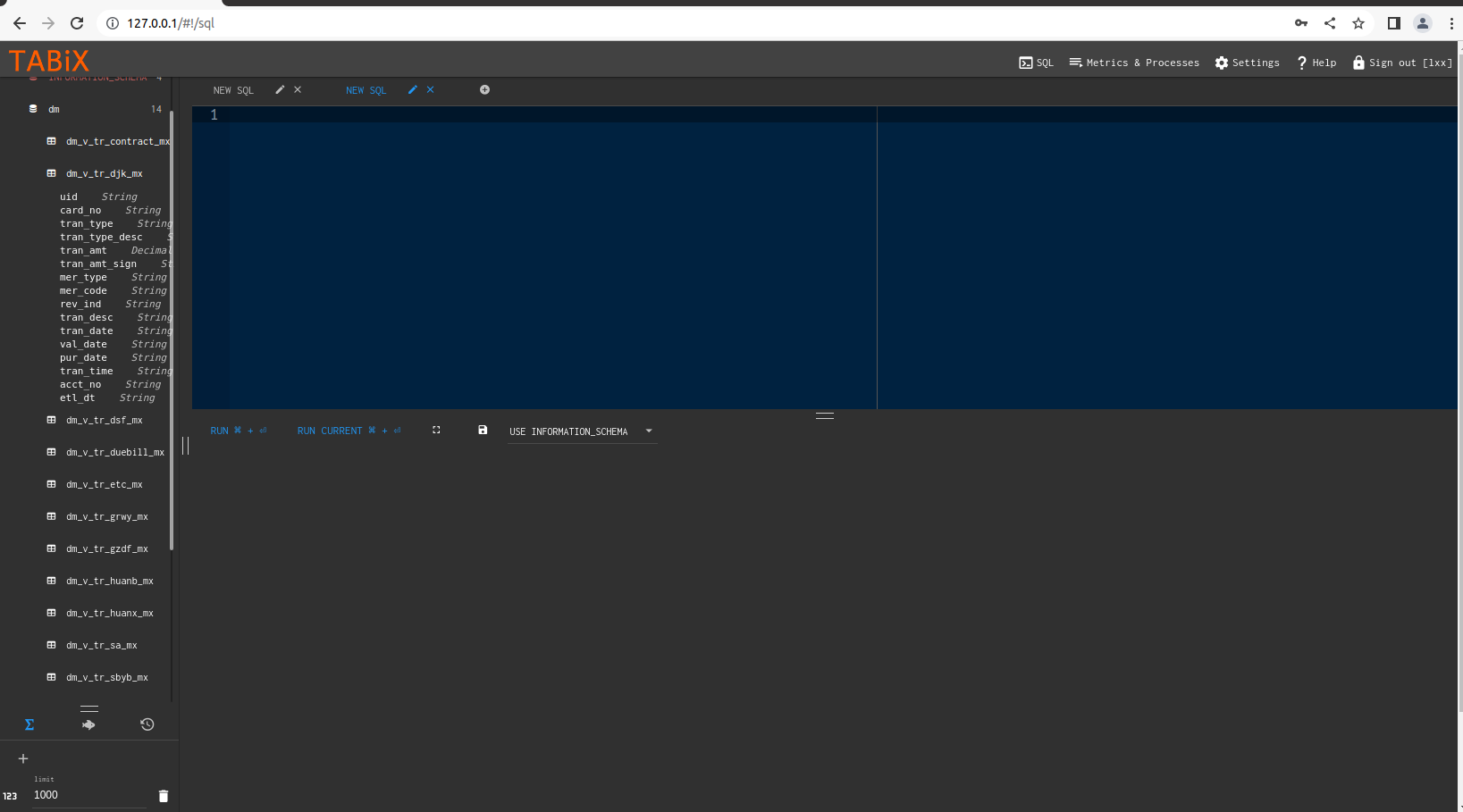
通过tabix，我们可以从网页端访问数据库。



可以查看数据集的大小。



可以通过在tabix的sql框中输入sql语句，实现对消费数额大小的查询。



1. 附加文件

hive部分：项目，hive2.scala，hive2-2.scala，hive-clickhouse.sql，pom.xml

流式数据部分：Cast，生产者producer，消费者consumer，建表语句real.sql