kafka传数据到Flink存储到mysql之Flink使用SQL语句聚合数据流（设置时间窗口，EventTime）...

网上没什么资料，就分享下:)

简单模式：kafka传数据到Flink存储到mysql 可以参考网站：

利用Flink stream从kafka中写数据到mysql

maven依赖情况：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.xxr</groupId>
    <artifactId>flink</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <flink.version>1.4.1</flink.version>
    </properties>
    <build>
        <pluginManagement>
            <plugins>
<!-- 设置java版本为1.8 -->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                        <encoding>UTF-8</encoding>
                        <compilerArgs>
                            <arg>-extdirs</arg>
```

```
                            <arg>${project.basedir}/src/lib</arg>
                        </compilerArgs>
                    </configuration>
                </plugin>
                <!-- maven-assembly方式打包成jar -->
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-assembly-plugin</artifactId>
                    <version>2.5.5</version>
                    <configuration>
                        <archive>
                            <manifest>

<mainClass>com.xxr.flink.stream_sql</mainClass>
                            </manifest>
                        </archive>
                        <descriptorRefs>
                            <descriptorRef>jar-with-
dependencies</descriptorRef>
                        </descriptorRefs>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
            <version>1.8.0-beta1</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.8.0-beta1</version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
```

```
            <version>1.2.17</version>
        </dependency>
        <dependency>
            <groupId>org.scala-lang</groupId>
            <artifactId>scala-compiler</artifactId>
            <version>2.11.1</version>
        </dependency>
        <dependency>
            <groupId>org.scala-lang.modules</groupId>
            <artifactId>scala-xml_2.11</artifactId>
            <version>1.0.2</version>
        </dependency>

        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-java</artifactId>
            <version>${flink.version}</version>
        </dependency>

        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-java_2.11</artifactId>
            <version>${flink.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-streaming-scala_2.11</artifactId>
            <version>${flink.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-clients_2.11</artifactId>
            <version>${flink.version}</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-core
-->
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-core</artifactId>
            <version>${flink.version}</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-
runtime -->
        <dependency>
```

```
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-runtime_2.11</artifactId>
            <version>${flink.version}</version>
        </dependency>

        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-wikiedits_2.11</artifactId>
            <version>${flink.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kafka-0.8_2.11</artifactId>
            <version>${flink.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-table_2.11</artifactId>
            <version>${flink.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-jdbc</artifactId>
            <version>${flink.version}</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.39</version>
        </dependency>
    </dependencies>
</project>
```

配置文件及sql语句，时间窗口是1分钟：

```
public class JDBCTestBase {
    //每过一分钟计算一分钟内的num最大值，以rowtime作为时间基准
    public static final String SQL_MAX = "SELECT  MAX(num)
,TUMBLE_END(rowtime, INTERVAL '1' minute) as wEnd FROM wiki_table  group by
TUMBLE(rowtime, interval '1' minute)";
    public static final String SQL_AVG = "SELECT  AVG(num)
,TUMBLE_END(rowtime, INTERVAL '1' minute) as wEnd FROM wiki_table  group by
TUMBLE(rowtime, interval '1' minute)";
```

```
    public static final String SQL_MIN = "SELECT  MIN(num)
,TUMBLE_END(rowtime, INTERVAL '1' minute) as wEnd FROM wiki_table  group by
TUMBLE(rowtime, interval '1' minute)";
    public static final String kafka_group = "test-consumer-group";
    public static final String kafka_zookper = "localhost:2181";
    public static final String kafka_hosts = "localhost:9092";
    public static final String kafka_topic = "wiki-result";
    public static final String DRIVER_CLASS = "com.mysql.jdbc.Driver";
    public static final String DB_URL = "jdbc:mysql://localhost:3306/db?
user=user&password=password";
}
```

MySQL建表：

```
CREATE TABLE wiki (
  Id int(11) NOT NULL AUTO_INCREMENT,
  avg double DEFAULT NULL,
  time timestamp NULL DEFAULT NULL,
  PRIMARY KEY (Id)
)
```

发送数据到kafka,用的是flink example的wikiproducer：

Monitoring the Wikipedia Edit Stream

```
import org.apache.flink.api.java.functions.KeySelector;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.datastream.KeyedStream;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer08;
import org.apache.flink.streaming.connectors.wikiedits.WikipediaEditEvent;
import org.apache.flink.streaming.connectors.wikiedits.WikipediaEditsSource;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.api.common.functions.FoldFunction;
import org.apache.flink.api.common.functions.MapFunction;
```

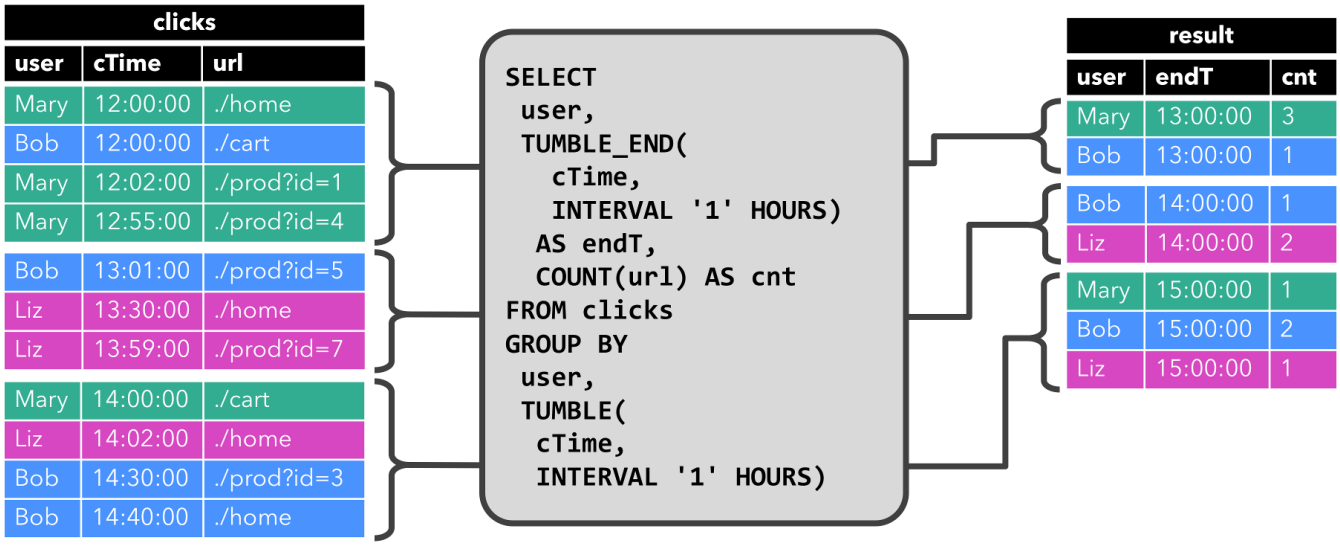
```
public class WikipediaAnalysis {
    public static void main(String[] args) throws Exception {

        StreamExecutionEnvironment see =
StreamExecutionEnvironment.getExecutionEnvironment();

        DataStream<WikipediaEditEvent> edits = see.addSource(new
WikipediaEditsSource());

        KeyedStream<WikipediaEditEvent, String> keyedEdits = edits
          .keyBy(new KeySelector<WikipediaEditEvent, String>() {
            @Override
            public String getKey(WikipediaEditEvent event) {
              return event.getUser();
            }
          });

        DataStream<Tuple3<String, Long,Long>> result = keyedEdits
          .timeWindow(Time.seconds(10))
          .fold(new Tuple3<>("", 0L,0L), new
FoldFunction<WikipediaEditEvent, Tuple3<String, Long,Long>>() {
            @Override
            public Tuple3<String, Long,Long> fold(Tuple3<String, Long,Long>
acc, WikipediaEditEvent event) {
              acc.f0 = event.getUser().trim();
              acc.f1 += event.getByteDiff();
              acc.f2 = System.currentTimeMillis();
              return acc;
            }
          });

        result
        .map(new MapFunction<Tuple3<String,Long,Long>, String>() {
            @Override
            public String map(Tuple3<String, Long,Long> tuple) {
                return tuple.toString();
            }
        })
        .addSink(new FlinkKafkaProducer08<>("localhost:9092", "wiki-result",
new SimpleStringSchema()));
        result.print();
        see.execute();
      }
}
```

重写RichSinkFunction，用于写入到mysql中：

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Timestamp;

import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.streaming.api.functions.sink.RichSinkFunction;


import kafka.common.Config;

public class WikiSQLSink extends RichSinkFunction<Tuple3<String,Long, Long>>
{

    private static final long serialVersionUID = 1L;
    private Connection connection;
    private PreparedStatement preparedStatement;
    String drivername = JDBCTestBase.DRIVER_CLASS;
    String dburl = JDBCTestBase.DB_URL;

    @Override
    public void invoke(Tuple3<String,Long, Long> value) throws Exception {
        Class.forName(drivername);
        connection = DriverManager.getConnection(dburl);
        String sql = "INSERT into wiki(name,avg,time) values(?,?,?)";
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString(1, value.f0);
        preparedStatement.setLong(2, value.f1);
        preparedStatement.setLong(3, value.f2);
        //preparedStatement.setTimestamp(4, new
Timestamp(System.currentTimeMillis()));
        preparedStatement.executeUpdate();
        if (preparedStatement != null) {
            preparedStatement.close();
        }
        if (connection != null) {
            connection.close();
        }
```

```
    }

}
```

用Flink中流计算类，用的是EventTime，用sql语句对数据进行聚合，写入数据到mysql中去，sql的语法用的是是另一个开源框架Apache Cassandra：



图片说明：

https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/table/streaming.html#time-attributes

```
package com.xxr.flink;

import java.sql.Timestamp;
import java.util.Date;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

import org.apache.commons.lang3.StringUtils;
import org.apache.flink.api.common.functions.FilterFunction;
import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.common.restartstrategy.RestartStrategies;
import org.apache.flink.api.common.typeinfo.TypeInformation;
import org.apache.flink.api.common.typeinfo.Types;
import org.apache.flink.api.java.io.jdbc.JDBCAppendTableSink;
import org.apache.flink.api.java.tuple.Tuple3;
import org.apache.flink.api.java.tuple.Tuple4;
import org.apache.flink.core.fs.FileSystem.WriteMode;
import org.apache.flink.streaming.api.TimeCharacteristic;
```

```
import org.apache.flink.streaming.api.datastream.DataStream;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.TimestampAssigner;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer08;
import org.apache.flink.streaming.util.serialization.SimpleStringSchema;
import org.apache.flink.table.api.Table;
import org.apache.flink.table.api.TableEnvironment;
import org.apache.flink.table.api.WindowedTable;
import org.apache.flink.table.api.java.StreamTableEnvironment;
//时间参数网址
//https://ci.apache.org/projects/flink/flink-docs-release-
1.4/dev/table/streaming.html#event-time
//Concepts & Common API
//https://ci.apache.org/projects/flink/flink-docs-release-
1.4/dev/table/common.html#register-a-table
//SQL语法
//https://ci.apache.org/projects/flink/flink-docs-release-
1.4/dev/table/sql.html
public class stream_sql {
    public static void main(String[] args) throws Exception {

        Properties pro = new Properties();
        pro.put("bootstrap.servers", JDBCTestBase.kafka_hosts);
        pro.put("zookeeper.connect", JDBCTestBase.kafka_zookper);
        pro.put("group.id", JDBCTestBase.kafka_group);
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        StreamTableEnvironment tableEnv =
TableEnvironment.getTableEnvironment(env);
        // env.getConfig().disableSysoutLogging(); //设置此可以屏蔽掉日记打印情况

env.getConfig().setRestartStrategy(RestartStrategies.fixedDelayRestart(4,
10000));
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
        env.enableCheckpointing(5000);

        DataStream<String> sourceStream = env
                    .addSource(new FlinkKafkaConsumer08<String>
(JDBCTestBase.kafka_topic, new SimpleStringSchema(), pro));

        DataStream<Tuple3<Long, String, Long>> sourceStreamTra =
sourceStream.filter(new FilterFunction<String>() {
            @Override
```

```
            public boolean filter(String value) throws Exception {
                return StringUtils.isNotBlank(value);
            }
        }).map(new MapFunction<String, Tuple3<Long, String, Long>>() {
            @Override
            public Tuple3<Long, String, Long> map(String value) throws
Exception {
                String temp = value.replaceAll("(\\(|\\))", "");
                String[] args = temp.split(",");
                try {
                    return new Tuple3<Long, String, Long>
(Long.valueOf(args[2]), args[0].trim(), Long.valueOf(args[1]));

                } catch (Exception e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                    return new Tuple3<Long, String, Long>
(System.currentTimeMillis(), args[0].trim(),0L);

                }
            }
        });
        //設置将哪个字段用于eventTime
        DataStream<Tuple3<Long, String, Long>> withTimestampsAndWatermarks =
sourceStreamTra
                .assignTimestampsAndWatermarks(new FirstTandW());
        //内置参数rowtime.rowtime就是EventTime protime是ProcessingTime
        tableEnv.registerDataStream("wiki_table",
withTimestampsAndWatermarks, "etime,name, num,rowtime.rowtime");
        withTimestampsAndWatermarks.print();
        // define sink for room data and execute query
        JDBCAppendTableSink sink =
JDBCAppendTableSink.builder().setDrivername(JDBCTestBase.DRIVER_CLASS)
                .setDBUrl(JDBCTestBase.DB_URL).setQuery("INSERT INTO wiki
(avg,time) VALUES (?,?)")
                .setParameterTypes(Types.LONG, Types.SQL_TIMESTAMP).build();
        //执行查询
        Table result = tableEnv.sqlQuery(JDBCTestBase.SQL_MIN);
        //写入csv
//        result.writeToSink(new CsvTableSink("D:/a.csv", // output path
//                "|", // optional: delimit files by '|'
//                1, // optional: write to a single file
//                WriteMode.OVERWRITE)); // optional: override existing
files
        //写入数据库
```

```
        result.writeToSink(sink);

        env.execute();


    }
}
```

重写AssignerWithPeriodicWatermarks设置watermark，处理时间是EventTime的话必须要有这个方法，ProcessingTime 可忽略

```
import org.apache.flink.api.java.tuple.Tuple3;
import
org.apache.flink.streaming.api.functions.AssignerWithPeriodicWatermarks;
import org.apache.flink.streaming.api.watermark.Watermark;

public class FirstTandW implements
AssignerWithPeriodicWatermarks<Tuple3<Long,String, Long>> {

    private final long maxOutOfOrderness = 3500; // 3.5 seconds

    private long currentMaxTimestamp;
    @Override
    public long extractTimestamp(Tuple3<Long,String, Long> element, long
previousElementTimestamp) {
        // TODO Auto-generated method stub
        long timestamp = element.f0;
        currentMaxTimestamp = Math.max(timestamp, currentMaxTimestamp);
        return timestamp;
    }

    @Override
    public Watermark getCurrentWatermark() {
        // TODO Auto-generated method stub
        return new Watermark(currentMaxTimestamp - maxOutOfOrderness);
    }

}
```

maven assembly打包成jar，放flink运行就行了，不会打包看我博客

基础知识

Flink 的Window 操作

Flink流计算编程--在WindowedStream中体会EventTime与ProcessingTime

Flink文档写的很好。。刚开始做没仔细看，坑不少

git：https://github.com/xxrznj/flink-kafka-sql