

## 前言

之前其实在《从0到1学习Flink》—— 如何自定义 Data Sink ? 文章中其实已经写了点将数据写入到 MySQL，但是一些配置化的东西当时是写死的，不能够通用，最近知识星球里有朋友叫我：写个从 kafka 中读取数据，经过 Flink 做个预聚合，然后创建数据库连接池将数据批量写入到 mysql 的例子。



于是才有了这篇文章，更多提问和想要我写的文章可以在知识星球里像我提问，我会根据提问及时回答和尽可能作出文章的修改。

## 准备

你需要将这两个依赖添加到 pom.xml 中

```
1 <dependency>
2   <groupId>mysql</groupId>
3   <artifactId>mysql-connector-java</artifactId>
4   <version>5.1.34</version>
5 </dependency>
```

## 读取 kafka 数据

这里我依旧用的以前的 student 类，自己本地起了 kafka 然后造一些测试数据，这里我们测试发送一条数据则 sleep 10s，意味着往 kafka 中一分钟发 6 条数据。

```
1 package com.zhisheng.connectors.mysql.utils;
2
3 import com.zhisheng.common.utils.GsonUtil;
4 import com.zhisheng.connectors.mysql.model.Student;
5 import org.apache.kafka.clients.producer.KafkaProducer;
6 import org.apache.kafka.clients.producer.ProducerRecord;
7
8 import java.util.Properties;
```

```

9
10 /**
11  * Desc: 往kafka中写数据,可以使用这个main函数进行测试
12  * Created by zhisheng on 2019-02-17
13  * Blog: http://www.54tianzhisheng.cn/tags/Flink/
14  */
15 public class KafkaUtil {
16     public static final String broker_list = "localhost:9092";
17     public static final String topic = "student"; //kafka topic 需要和
    flink 程序用同一个 topic
18
19     public static void writeToKafka() throws InterruptedException {
20         Properties props = new Properties();
21         props.put("bootstrap.servers", broker_list);
22         props.put("key.serializer",
23             "org.apache.kafka.common.serialization.StringSerializer");
24         props.put("value.serializer",
25             "org.apache.kafka.common.serialization.StringSerializer");
26         KafkaProducer producer = new KafkaProducer<String, String>
27             (props);
28
29         for (int i = 1; i <= 100; i++) {
30             Student student = new Student(i, "zhisheng" + i, "password"
31                 + i, 18 + i);
32             ProducerRecord record = new ProducerRecord<String, String>
33                 (topic, null, null, GsonUtil.toJson(student));
34             producer.send(record);
35             System.out.println("发送数据: " + GsonUtil.toJson(student));
36             Thread.sleep(10 * 1000); //发送一条数据 sleep 10s, 相当于 1 分
37             钟 6 条
38         }
39         producer.flush();
40     }
41
42     public static void main(String[] args) throws InterruptedException
43     {
44         writeToKafka();
45     }
46 }

```

从 kafka 中读取数据, 然后序列化成 student 对象。

```

1  final StreamExecutionEnvironment env =
    StreamExecutionEnvironment.getExecutionEnvironment();
2  Properties props = new Properties();
3  props.put("bootstrap.servers", "localhost:9092");
4  props.put("zookeeper.connect", "localhost:2181");
5  props.put("group.id", "metric-group");
6  props.put("key.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer");
7  props.put("value.deserializer",
    "org.apache.kafka.common.serialization.StringDeserializer");
8  props.put("auto.offset.reset", "latest");
9
10 SingleOutputStreamOperator<Student> student = env.addSource(new
    FlinkKafkaConsumer011<>(
11     "student",    //这个 kafka topic 需要和上面的工具类的 topic 一致
12     new SimpleStringSchema(),
13     props)).setParallelism(1)
14     .map(string -> GsonUtil.fromJson(string, Student.class)); //, 解
    析字符串成 student 对象
15

```

因为 RichSinkFunction 中如果 sink 一条数据到 mysql 中就会调用 invoke 方法一次，所以如果要想实现批量写的话，我们最好在 sink 之前就把数据聚合一下。那这里我们开个一分钟的窗口去聚合 Student 数据。

```

1  student.timeWindowAll(Time.minutes(1)).apply(new
    AllWindowFunction<Student, List<Student>, TimeWindow>() {
2      @Override
3      public void apply(TimeWindow window, Iterable<Student> values,
        Collector<List<Student>> out) throws Exception {
4          ArrayList<Student> students = Lists.newArrayList(values);
5          if (students.size() > 0) {
6              System.out.println("1 分钟内收集到 student 的数据条数是: " +
            students.size());
7              out.collect(students);
8          }
9      }
10 });

```

## 写入数据库

这里使用 DBCP 连接池连接数据库 mysql，pom.xml 中添加依赖：

```
1 <dependency>
2   <groupId>org.apache.commons</groupId>
3   <artifactId>commons-dbcp2</artifactId>
4   <version>2.1.1</version>
5 </dependency>
```

如果你想使用其他的数据库连接池请加入对应的依赖。

这里将数据写入到 MySQL 中，依旧是和之前文章一样继承 RichSinkFunction 类，重写里面的方法：

```
1 package com.zhisheng.connectors.mysql.sinks;
2
3 import com.zhisheng.connectors.mysql.model.Student;
4 import org.apache.commons.dbcp2.BasicDataSource;
5 import org.apache.flink.configuration.Configuration;
6 import org.apache.flink.streaming.api.functions.sink.RichSinkFunction;
7
8 import javax.sql.DataSource;
9 import java.sql.Connection;
10 import java.sql.DriverManager;
11 import java.sql.PreparedStatement;
12 import java.util.List;
13
14 /**
15  * Desc: 数据批量 sink 数据到 mysql
16  * Created by zhisheng_tian on 2019-02-17
17  * Blog: http://www.54tianzhisheng.cn/tags/Flink/
18  */
19 public class SinkToMySQL extends RichSinkFunction<List<Student>> {
20     PreparedStatement ps;
21     BasicDataSource dataSource;
22     private Connection connection;
23
24     /**
25      * open() 方法中建立连接，这样不用每次 invoke 的时候都要建立连接和释放连接
26      *
27      * @param parameters
28      * @throws Exception
29      */
30     @Override
31     public void open(Configuration parameters) throws Exception {
32         super.open(parameters);
33         dataSource = new BasicDataSource();
```

```
34         connection = getConnection(dataSource);
35         String sql = "insert into Student(id, name, password, age)
values(?, ?, ?, ?)";
36         ps = this.connection.prepareStatement(sql);
37     }
38
39     @Override
40     public void close() throws Exception {
41         super.close();
42         //关闭连接和释放资源
43         if (connection != null) {
44             connection.close();
45         }
46         if (ps != null) {
47             ps.close();
48         }
49     }
50
51     /**
52      * 每条数据的插入都要调用一次 invoke() 方法
53      *
54      * @param value
55      * @param context
56      * @throws Exception
57      */
58     @Override
59     public void invoke(List<Student> value, Context context) throws
Exception {
60         //遍历数据集合
61         for (Student student : value) {
62             ps.setInt(1, student.getId());
63             ps.setString(2, student.getName());
64             ps.setString(3, student.getPassword());
65             ps.setInt(4, student.getAge());
66             ps.addBatch();
67         }
68         int[] count = ps.executeBatch();//批量后执行
69         System.out.println("成功了插入了" + count.length + "行数据");
70     }
71
72
73     private static Connection getConnection(BasicDataSource dataSource)
{
74         dataSource.setDriverClassName("com.mysql.jdbc.Driver");
75         //注意, 替换成自己本地的 mysql 数据库地址和用户名、密码
```

```

76     dataSource.setUrl("jdbc:mysql://localhost:3306/test");
77     dataSource.setUsername("root");
78     dataSource.setPassword("root123456");
79     //设置连接池的一些参数
80     dataSource.setInitialSize(10);
81     dataSource.setMaxTotal(50);
82     dataSource.setMinIdle(2);
83
84     Connection con = null;
85     try {
86         con = dataSource.getConnection();
87         System.out.println("创建连接池: " + con);
88     } catch (Exception e) {
89         System.out.println("-----mysql get connection has
exception , msg = " + e.getMessage());
90     }
91     return con;
92 }
93 }
94

```

## 核心类 Main

核心程序如下:

```

1  public class Main {
2      public static void main(String[] args) throws Exception{
3          final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
4          Properties props = new Properties();
5          props.put("bootstrap.servers", "localhost:9092");
6          props.put("zookeeper.connect", "localhost:2181");
7          props.put("group.id", "metric-group");
8          props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
9          props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
10         props.put("auto.offset.reset", "latest");
11
12         SingleOutputStreamOperator<Student> student = env.addSource(new
FlinkKafkaConsumer011<>(
13             "student",    //这个 kafka topic 需要和上面的工具类的 topic
一致
14             new SimpleStringSchema(),

```

```

15         props)).setParallelism(1)
16         .map(string -> GsonUtil.fromJson(string,
Student.class)); //
17         student.timeWindowAll(Time.minutes(1)).apply(new
AllWindowFunction<Student, List<Student>, TimeWindow>() {
18             @Override
19             public void apply(TimeWindow window, Iterable<Student>
values, Collector<List<Student>> out) throws Exception {
20                 ArrayList<Student> students =
Lists.newArrayList(values);
21                 if (students.size() > 0) {
22                     System.out.println("1 分钟内收集到 student 的数据条数
是: " + students.size());
23                     out.collect(students);
24                 }
25             }
26         }).addSink(new SinkToMySQL());
27
28         env.execute("flink learning connectors kafka");
29     }
30 }
31

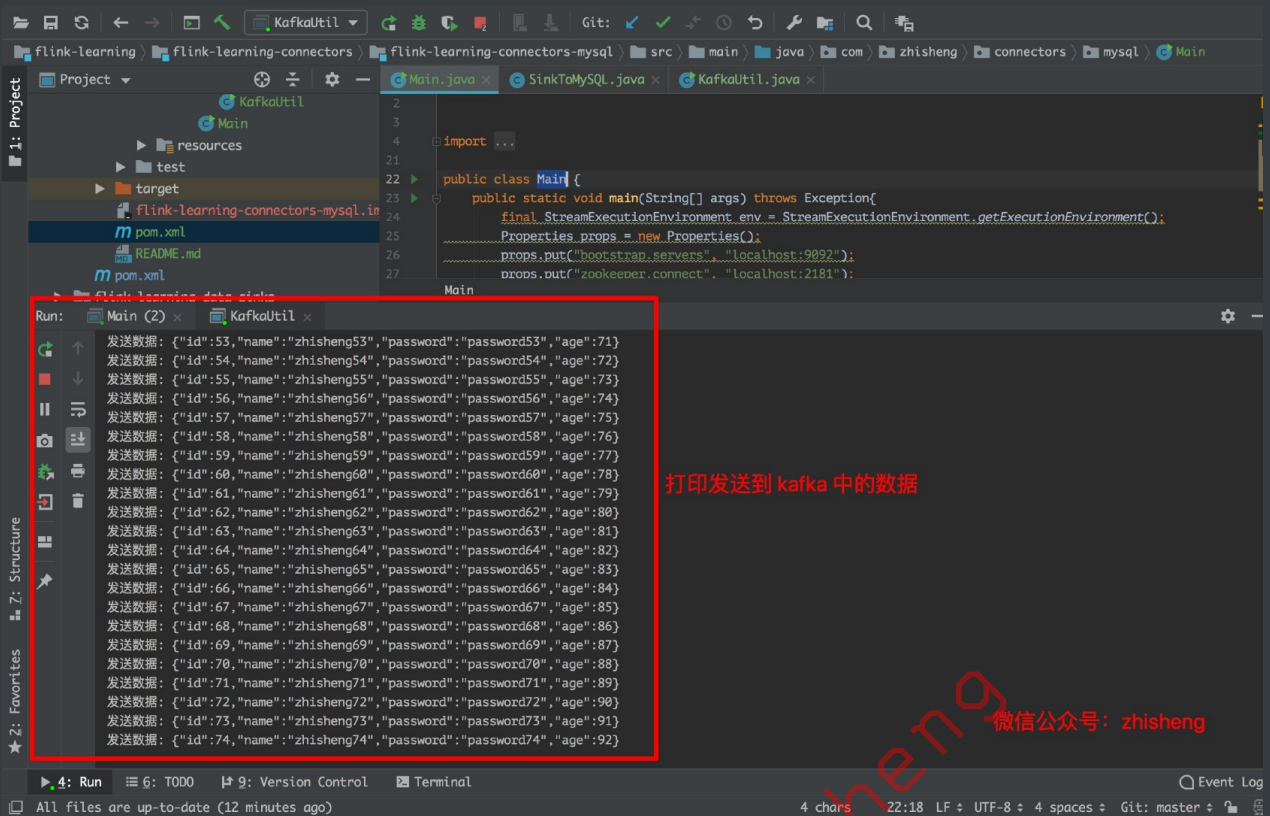
```

## 运行项目

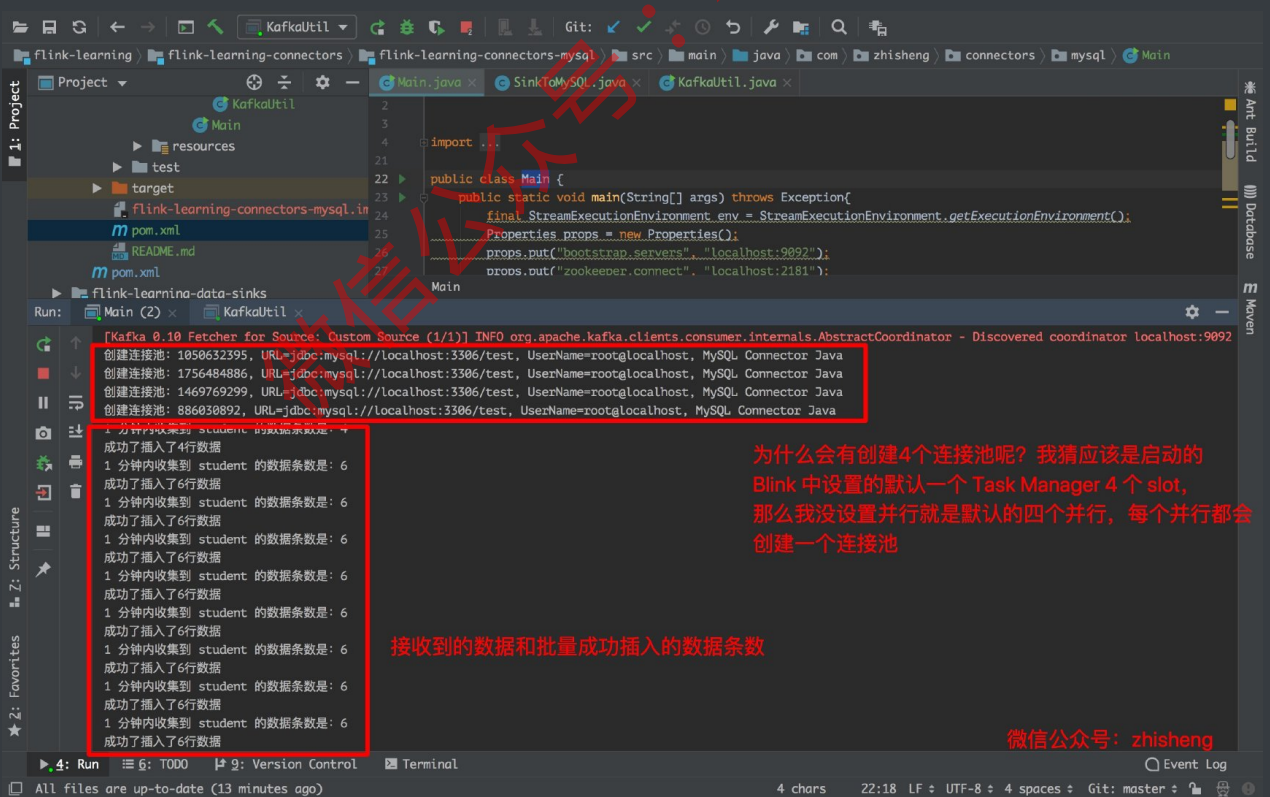
运行 Main 类后再运行 KafkaUtils.java 类!

下图是往 Kafka 中发送的数据:





下图是运行 Main 类的日志，会创建 4 个连接池是因为默认的 4 个并行度，你如果在 addSink 这个算子设置并行度为 1 的话就会创建一个连接池：



下图是批量插入数据库的结果：



Structure Content Relations Triggers Table Info Query				
				test/student
Search: id =				
id	name	password	age	
60	zhisheng60	password60	78	
61	zhisheng61	password61	79	
62	zhisheng62	password62	80	
63	zhisheng63	password63	81	
64	zhisheng64	password64	82	
65	zhisheng65	password65	83	
66	zhisheng66	password66	84	
67	zhisheng67	password67	85	
68	zhisheng68	password68	86	
69	zhisheng69	password69	87	
70	zhisheng70	password70	88	
71	zhisheng71	password71	89	
72	zhisheng72	password72	90	
73	zhisheng73	password73	91	
74	zhisheng74	password74	92	
75	zhisheng75	password75	93	
76	zhisheng76	password76	94	
77	zhisheng77	password77	95	
78	zhisheng78	password78	96	
79	zhisheng79	password79	97	
80	zhisheng80	password80	98	
81	zhisheng81	password81	99	
82	zhisheng82	password82	100	
83	zhisheng83	password83	101	
84	zhisheng84	password84	102	
85	zhisheng85	password85	103	
86	zhisheng86	password86	104	
87	zhisheng87	password87	105	
88	zhisheng88	password88	106	
89	zhisheng89	password89	107	
90	zhisheng90	password90	108	
91	zhisheng91	password91	109	
92	zhisheng92	password92	110	
93	zhisheng93	password93	111	
94	zhisheng94	password94	112	
95	zhisheng95	password95	113	
96	zhisheng96	password96	114	
97	zhisheng97	password97	115	
98	zhisheng98	password98	116	
99	zhisheng99	password99	117	
100	zhisheng100	password100	118	

数据库成功插入100条数据

你可以看下每次都是几条数据  
几条数据的增加，证明我们的  
批量写起作用了

微信公众号: zhisheng

## 总结

本文从知识星球一位朋友的疑问来写的，应该都满足了他的条件（批量/数据库连接池/写入mysql），的确网上很多的例子都是简单的 demo 形式，都是单条数据就创建数据库连接插入 MySQL，如果要写的数量很大的话，会对 MySQL 的写有很大的压力。这也是我之前在《从0到1学习Flink》—— Flink 写入数据到 ElasticSearch 中，数据写 ES 强调过的，如果要提高性能必定要批量的写。就拿我们现在的这篇文章来说，如果数据量大的话，聚合一分钟数据达万条，那么这样批量写会比来一条写一条性能提高不知道有多少。

本文原创地址是: <http://www.54tianzhisheng.cn/2019/01/09/Flink-MySQL-sink/> , 未经允许禁止转载。

## 关注我

微信公众号: **zhisheng**

另外我自己整理了些 Flink 的学习资料, 目前已经全部放到微信公众号了。你可以加我的微信: **zhisheng\_tian**, 然后回复关键字: **Flink** 即可无条件获取到。



更多私密资料请加入知识星球!



## Flink 精进学习

星主: zhisheng

知识星球

微信扫描预览星球详情



### Github 代码仓库

<https://github.com/zhisheng17/flink-learning/>

以后这个项目的所有代码都将放在这个仓库里，包含了学习 flink 的一些 demo 和博客。

本文的项目代码在 <https://github.com/zhisheng17/flink-learning/tree/master/flink-learning-connectors/flink-learning-connectors-mysql>

### 相关文章

- 1、[《从0到1学习Flink》—— Apache Flink 介绍](#)
- 2、[《从0到1学习Flink》—— Mac 上搭建 Flink 1.6.0 环境并构建运行简单程序入门](#)
- 3、[《从0到1学习Flink》—— Flink 配置文件详解](#)
- 4、[《从0到1学习Flink》—— Data Source 介绍](#)
- 5、[《从0到1学习Flink》—— 如何自定义 Data Source ?](#)
- 6、[《从0到1学习Flink》—— Data Sink 介绍](#)
- 7、[《从0到1学习Flink》—— 如何自定义 Data Sink ?](#)
- 8、[《从0到1学习Flink》—— Flink Data transformation\(转换\)](#)
- 9、[《从0到1学习Flink》—— 介绍Flink中的Stream Windows](#)

- 10、[《从0到1学习Flink》—— Flink 中的几种 Time 详解](#)
- 11、[《从0到1学习Flink》—— Flink 写入数据到 ElasticSearch](#)
- 12、[《从0到1学习Flink》—— Flink 项目如何运行？](#)
- 13、[《从0到1学习Flink》—— Flink 写入数据到 Kafka](#)
- 14、[《从0到1学习Flink》—— Flink JobManager 高可用性配置](#)
- 15、[《从0到1学习Flink》—— Flink parallelism 和 Slot 介绍](#)
- 16、[《从0到1学习Flink》—— Flink 读取 Kafka 数据批量写入到 MySQL](#)

微信公众号：zhisheng9