

## 前言

在前面文章 [Flink 常用的 Source 和 Sink Connectors 介绍](#) 中我讲解了 Flink 中的 Data Source 和 Data Sink，然后介绍了 Flink 中自带的一些 Source 和 Sink 的 Connector，接着我们还有几篇实战会讲解了如何从 Kafka 处理数据写入到 Kafka、MySQL、ElasticSearch 等，当然 Flink 还有一些其他的 Connector，我们这里就不一一介绍了，大家如果感兴趣的话可以去官网查看一下，如果对其代码实现比较感兴趣的话，也可以去看看其源码的实现。我们这篇文章来讲解一下如何自定义 Source 和 Sink Connector？这样我们后面再遇到什么样的需求都难不倒我们了。

## 如何自定义 Source Connector?

上面就是 Flink 自带的 Kafka source，那么接下来就模仿着写一个从 MySQL 中读取数据的 Source。

首先 pom.xml 中添加 **MySQL 依赖**：

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

**数据库建表**如下：

```
1 DROP TABLE IF EXISTS `student`;
2 CREATE TABLE `student` (
3     `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
4     `name` varchar(25) COLLATE utf8_bin DEFAULT NULL,
5     `password` varchar(25) COLLATE utf8_bin DEFAULT NULL,
6     `age` int(10) DEFAULT NULL,
7     PRIMARY KEY (`id`)
8 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

**插入数据**：

```
1 INSERT INTO `student` VALUES ('1', 'zhisheng01', '123456', '18'), ('2',
2     'zhisheng02', '123', '17'), ('3', 'zhisheng03', '1234', '18'), ('4',
3     'zhisheng04', '12345', '16');
4 COMMIT;
```

**新建实体类**：Student.java

```
1 package com.zhisheng.flink.model;
2
3 /**
4  * blog: http://www.54tianzhisheng.cn/
5  */
6 @Data
7 @AllArgsConstructor
```

```

8  @NoArgsConstructor
9  public class Student {
10     public int id;
11     public String name;
12     public String password;
13     public int age;
14 }

```

**新建 Source 类** SourceFromMySQL.java, 该类继承 RichSourceFunction, 实现里面的 open、close、run、cancel 方法:

```

1  package com.zhisheng.flink.source;
2
3  import com.zhisheng.flink.model.Student;
4  import org.apache.flink.configuration.Configuration;
5  import org.apache.flink.streaming.api.functions.source.RichSourceFunction;
6
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9  import java.sql.PreparedStatement;
10 import java.sql.ResultSet;
11
12
13 /**
14  * blog: http://www.54tianzhisheng.cn/
15  */
16 public class SourceFromMySQL extends RichSourceFunction<Student> {
17
18     PreparedStatement ps;
19     private Connection connection;
20
21     /**
22      * 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。
23      *
24      * @param parameters
25      * @throws Exception
26      */
27     @Override
28     public void open(Configuration parameters) throws Exception {
29         super.open(parameters);
30         connection = getConnection();
31         String sql = "select * from Student;";
32         ps = this.connection.prepareStatement(sql);
33     }
34
35     /**
36      * 程序执行完毕就可以进行, 关闭连接和释放资源的动作了
37      *
38      * @throws Exception
39      */
40     @Override
41     public void close() throws Exception {
42         super.close();
43         if (connection != null) { //关闭连接和释放资源
44             connection.close();
45         }
46         if (ps != null) {
47             ps.close();

```

```

48     }
49 }
50
51 /**
52  * DataStream 调用一次 run() 方法用来获取数据
53  *
54  * @param ctx
55  * @throws Exception
56  */
57 @Override
58 public void run(SourceContext<Student> ctx) throws Exception {
59     ResultSet resultSet = ps.executeQuery();
60     while (resultSet.next()) {
61         Student student = new Student(
62             resultSet.getInt("id"),
63             resultSet.getString("name").trim(),
64             resultSet.getString("password").trim(),
65             resultSet.getInt("age"));
66         ctx.collect(student);
67     }
68 }
69
70 @Override
71 public void cancel() {
72 }
73
74 private static Connection getConnection() {
75     Connection con = null;
76     try {
77         Class.forName("com.mysql.jdbc.Driver");
78         con =
79 DriverManager.getConnection("jdbc:mysql://localhost:3306/test?
80 useUnicode=true&characterEncoding=UTF-8", "root", "root123456");
81     } catch (Exception e) {
82         System.out.println("-----mysql get connection has
83 exception , msg = "+ e.getMessage());
84     }
85     return con;
86 }

```

## Flink 程序:

```

1 package com.zhisheng.flink;
2
3 import com.zhisheng.flink.source.SourceFromMySQL;
4 import
5 org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6
7 /**
8  * blog: http://www.54tianzhisheng.cn/
9  */
10 public class Main2 {
11     public static void main(String[] args) throws Exception {
12         final StreamExecutionEnvironment env =
13 StreamExecutionEnvironment.getExecutionEnvironment();
14
15         env.addSource(new SourceFromMySQL()).print();
16     }
17 }

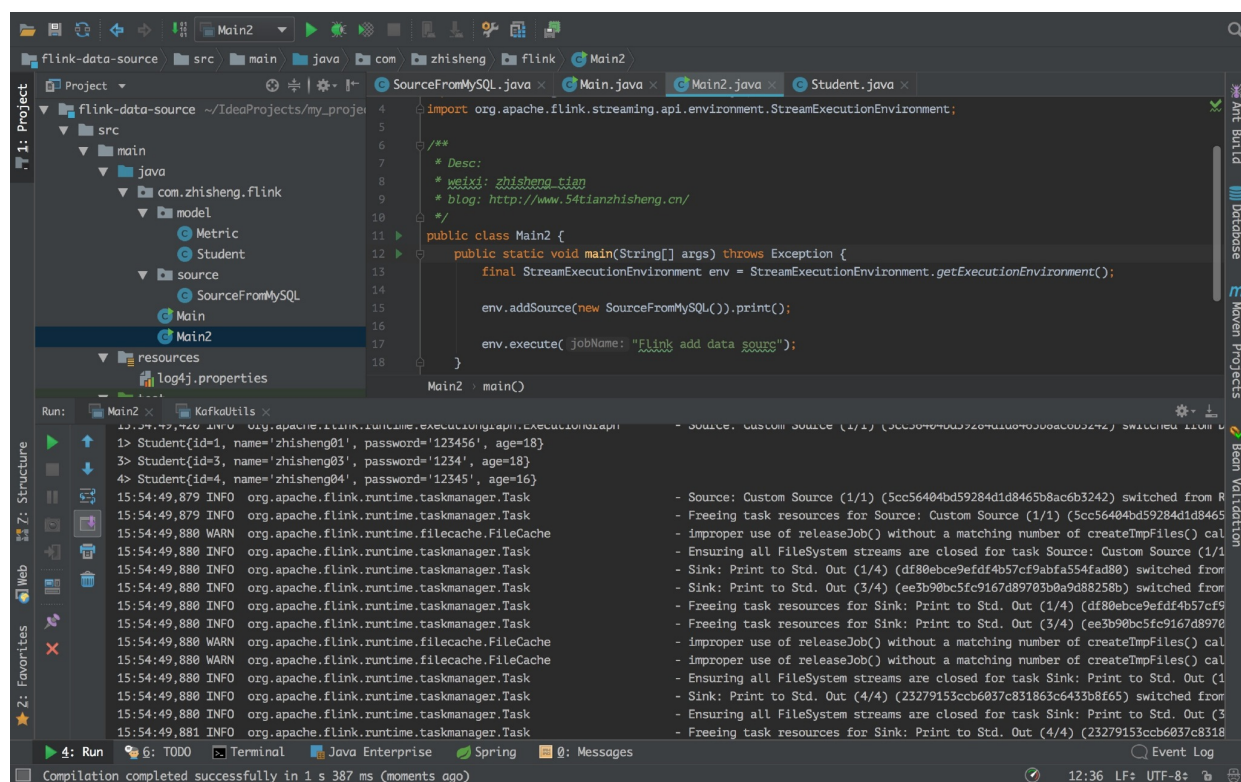
```

```

14 |
15 |     env.execute("Flink add data sourc");
16 | }
17 | }

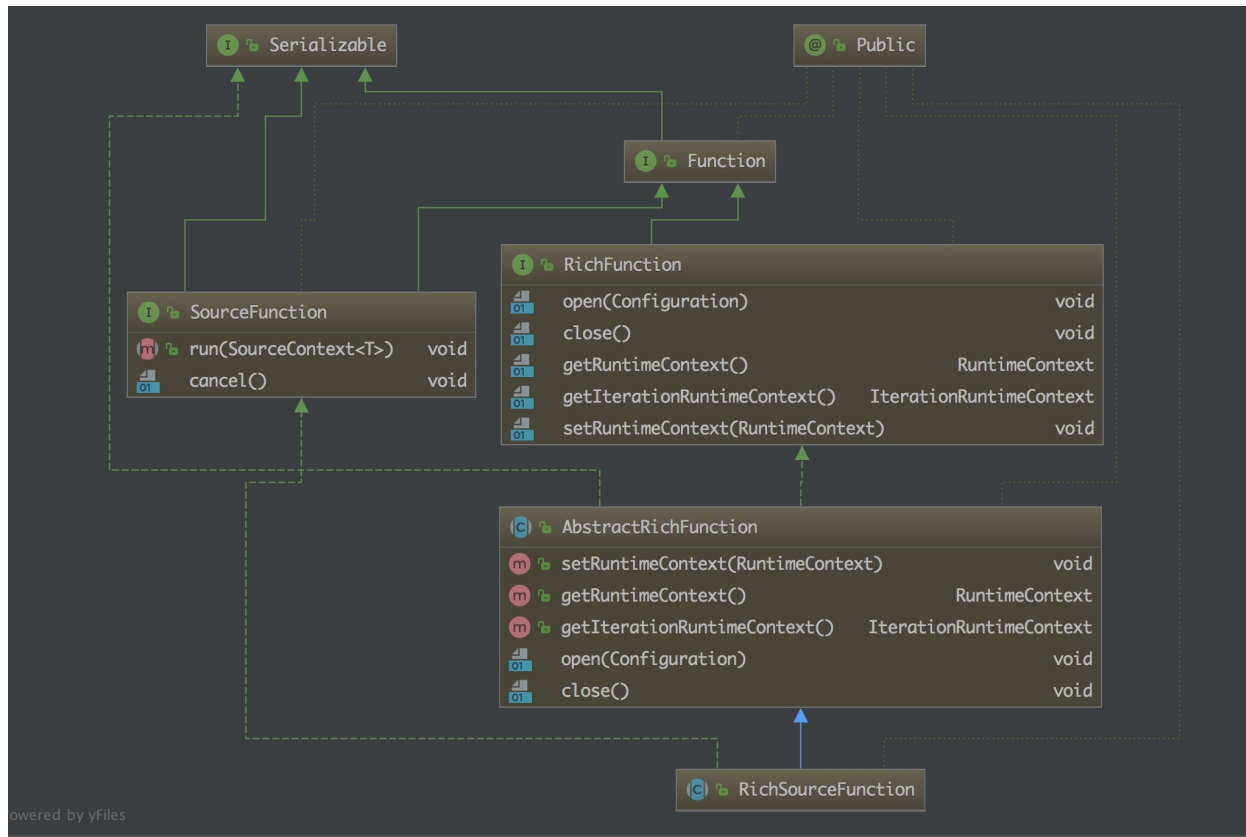
```

运行 Flink 程序，控制台日志中可以看见打印的 student 信息。

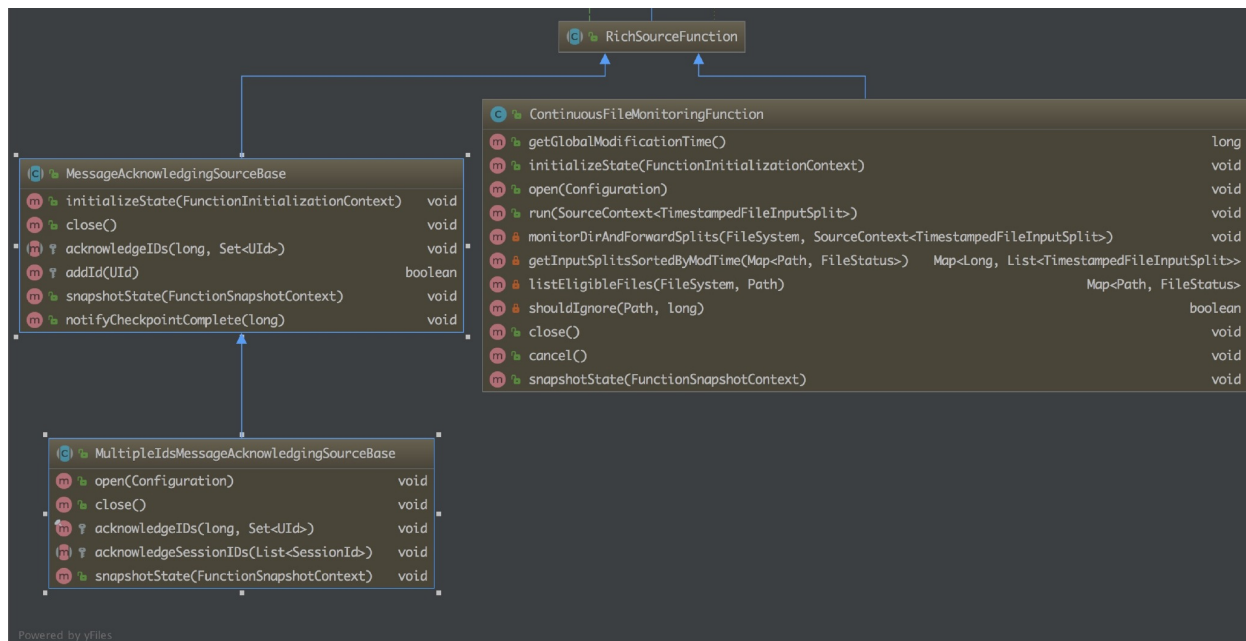


## RichSourceFunction 讲解

从上面自定义的 Source 可以看到我们继承的就是这个 RichSourceFunction 类，那么来了解一下：



一个抽象类，继承自 AbstractRichFunction。为实现一个 Rich SourceFunction 提供基础能力。该类的子类有三个，两个是抽象类，在此基础上提供了更具体的实现，另一个是 ContinuousFileMonitoringFunction。



- MessageAcknowledgingSourceBase：它针对的是数据源是消息队列的场景并且提供了基于 ID 的应答机制。
- MultipleIdsMessageAcknowledgingSourceBase：在 MessageAcknowledgingSourceBase 的基础上针对 ID 应答机制进行了更为细分的处理，支持两种 ID 应答模型：session id 和 unique message id。
- ContinuousFileMonitoringFunction：这是单个（非并行）监视任务，它接受 FileInputFormat，

并且根据 FileProcessingMode 和 FilePathFilter，它负责监视用户提供的路径；决定应该进一步读取和处理哪些文件；创建与这些文件对应的 FileInputSplit 拆分，将它们分配给下游任务以进行进一步处理。

## 如何自定义 Sink Connector?

下面将写一个 demo 教大家将从 Kafka Source 的数据 Sink 到 MySQL 中去。

### 工具类

工具类往 kafka topic student 发送数据

```
1  import com.alibaba.fastjson.JSON;
2  import com.zhisheng.flink.model.Metric;
3  import com.zhisheng.flink.model.Student;
4  import org.apache.kafka.clients.producer.KafkaProducer;
5  import org.apache.kafka.clients.producer.ProducerRecord;
6
7  import java.util.HashMap;
8  import java.util.Map;
9  import java.util.Properties;
10
11  /**
12   * 往kafka中写数据
13   * 可以使用这个main函数进行测试一下
14   * blog: http://www.54tianzhisheng.cn/
15   */
16  public class KafkaUtils2 {
17      public static final String broker_list = "localhost:9092";
18      public static final String topic = "student"; //kafka topic 需要和
19      //flink 程序用同一个 topic
20
21      public static void writeToKafka() throws InterruptedException {
22          Properties props = new Properties();
23          props.put("bootstrap.servers", broker_list);
24          props.put("key.serializer",
25              "org.apache.kafka.common.serialization.StringSerializer");
26          props.put("value.serializer",
27              "org.apache.kafka.common.serialization.StringSerializer");
28          KafkaProducer producer = new KafkaProducer<String, String>(props);
29
30          for (int i = 1; i <= 100; i++) {
31              Student student = new Student(i, "zhisheng" + i, "password" +
32                  i, 18 + i);
33              ProducerRecord record = new ProducerRecord<String, String>
34                  (topic, null, null, JSON.toJSONString(student));
35              producer.send(record);
36              System.out.println("发送数据: " + JSON.toJSONString(student));
37          }
38          producer.flush();
39      }
40  }
```

```

36     public static void main(String[] args) throws InterruptedException {
37         writeToKafka();
38     }
39 }

```

## SinkToMySQL

该类就是 Sink Function，继承了 RichSinkFunction，然后重写了里面的方法。在 invoke 方法中将数据插入到 MySQL 中。

```

1  package com.zhisheng.flink.sink;
2
3  import com.zhisheng.flink.model.Student;
4  import org.apache.flink.configuration.Configuration;
5  import org.apache.flink.streaming.api.functions.sink.RichSinkFunction;
6
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9  import java.sql.PreparedStatement;
10
11  /**
12   * blog: http://www.54tianzhisheng.cn/
13   */
14  public class SinkToMySQL extends RichSinkFunction<Student> {
15      PreparedStatement ps;
16      private Connection connection;
17
18      /**
19       * 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
20       *
21       * @param parameters
22       * @throws Exception
23       */
24      @Override
25      public void open(Configuration parameters) throws Exception {
26          super.open(parameters);
27          connection = getConnection();
28          String sql = "insert into Student(id, name, password, age)
values(?, ?, ?, ?)";
29          ps = this.connection.prepareStatement(sql);
30      }
31
32      @Override
33      public void close() throws Exception {
34          super.close();
35          //关闭连接和释放资源
36          if (connection != null) {
37              connection.close();
38          }
39          if (ps != null) {
40              ps.close();
41          }
42      }
43
44      /**
45       * 每条数据的插入都要调用一次 invoke() 方法

```

```

46     *
47     * @param value
48     * @param context
49     * @throws Exception
50     */
51     @Override
52     public void invoke(Student value, Context context) throws Exception {
53         //组装数据, 执行插入操作
54         ps.setInt(1, value.getId());
55         ps.setString(2, value.getName());
56         ps.setString(3, value.getPassword());
57         ps.setInt(4, value.getAge());
58         ps.executeUpdate();
59     }
60
61     private static Connection getConnection() {
62         Connection con = null;
63         try {
64             Class.forName("com.mysql.jdbc.Driver");
65             con =
66             DriverManager.getConnection("jdbc:mysql://localhost:3306/test?
67             useUnicode=true&characterEncoding=UTF-8", "root", "root123456");
68         } catch (Exception e) {
69             System.out.println("-----mysql get connection has
70             exception , msg = "+ e.getMessage());
71         }
72     }
73 }

```

## Flink 程序

这里的 source 是从 kafka 读取数据的, 然后 Flink 从 Kafka 读取到数据 (JSON) 后用阿里 fastjson 来解析成 student 对象, 然后在 addSink 中使用我们创建的 SinkToMySQL, 这样就可以把数据存储到 MySQL 了。

```

1 package com.zhisheng.flink;
2
3 import com.alibaba.fastjson.JSON;
4 import com.zhisheng.flink.model.Student;
5 import com.zhisheng.flink.sink.SinkToMySQL;
6 import org.apache.flink.api.common.serialization.SimpleStringSchema;
7 import org.apache.flink.streaming.api.datastream.DataStreamSource;
8 import
9 org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
10 import
11 org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
12 import org.apache.flink.streaming.api.functions.sink.PrintSinkFunction;
13 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer011;
14 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011;
15
16 import java.util.Properties;
17
18 /**
19  * blog: http://www.54tianzhisheng.cn/
20  */

```



```

19 public class Main3 {
20     public static void main(String[] args) throws Exception {
21         final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
22
23         Properties props = new Properties();
24         props.put("bootstrap.servers", "localhost:9092");
25         props.put("zookeeper.connect", "localhost:2181");
26         props.put("group.id", "metric-group");
27         props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
28         props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
29         props.put("auto.offset.reset", "latest");
30
31         SingleOutputStreamOperator<Student> student = env.addSource(new
FlinkKafkaConsumer011<>(
32             "student",    //这个 kafka topic 需要和上面的工具类的 topic 一致
33             new SimpleStringSchema(),
34             props)).setParallelism(1)
35             .map(string -> JSON.parseObject(string, Student.class));
//Fastjson 解析字符串成 student 对象
36
37         student.addSink(new SinkToMySQL()); //数据 sink 到 mysql
38
39         env.execute("Flink add sink");
40     }
41 }

```

## 结果

运行 Flink 程序，然后再运行 KafkaUtils2.java 工具类，这样就可以了。

如果数据插入成功了，那么查看下我们的数据库：

对象	student @test (localhost)			
id	name	password	age	
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	

select \* from `test`.`student` limit 0,1000

100 条记录在第 1 页

数据库中已经插入了 100 条我们从 Kafka 发送的数据了。证明我们的 SinkToMySQL 起作用了。

## 总结

本文先通过一个 Flink Kafka Source Connector 教大家如何使用连接器，然后通过自定义一个 Source Connector 从 MySQL 中读取数据，接着分析自定义 Source Connector 的方法，然后通过自定义一个 Sink Connector 将 Kafka 数据写入到 MySQL 中。

## Github 代码仓库

<https://github.com/zhisheng17/flink-learning/tree/master/flink-learning-data-sources>

<https://github.com/zhisheng17/flink-learning/tree/master/flink-learning-data-sinks>