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**：

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.34</version>
</dependency>
```

数据库建表如下：

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

插入数据：

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

新建实体类：Student.java

```
package com.zhisheng.flink.model;

/**
 * blog: http://www.54tianzhisheng.cn/
 */
@Data
@AllArgsConstructor
```

```
@NoArgsConstructor
public class Student {
    public int id;
    public String name;
    public String password;
    public int age;
}
```

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

```
 package com.zhisheng.flink.source;

import com.zhisheng.flink.model.Student;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.source.RichSourceFunction;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;


/**
 * blog: http://www.54tianzhisheng.cn/
 */
public class SourceFromMySQL extends RichSourceFunction<Student> {

    PreparedStatement ps;
    private Connection connection;

    /**
     * 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。
     *
     * @param parameters
     * @throws Exception
     */
    @Override
    public void open(Configuration parameters) throws Exception {
        super.open(parameters);
        connection = getConnection();
        String sql = "select * from Student;";
        ps = this.connection.prepareStatement(sql);
    }

    /**
     * 程序执行完毕就可以进行，关闭连接和释放资源的动作了
     *
     * @throws Exception
     */
    @Override
    public void close() throws Exception {
        super.close();
        if (connection != null) { //关闭连接和释放资源
            connection.close();
        }
        if (ps != null) {
            ps.close();
```

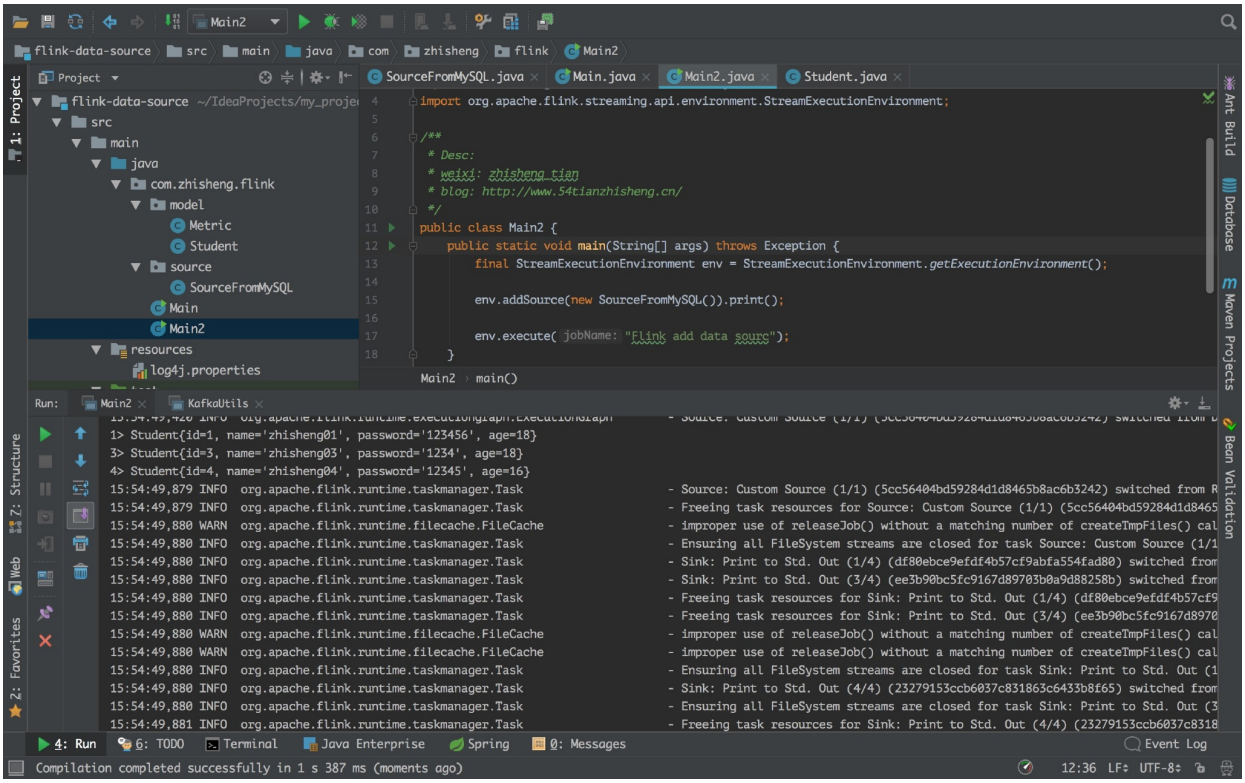
```
        }
    }

    /**
     * DataStream 调用一次 run() 方法用来获取数据
     *
     * @param ctx
     * @throws Exception
     */
    @Override
    public void run(SourceContext<Student> ctx) throws Exception {
        ResultSet resultSet = ps.executeQuery();
        while (resultSet.next()) {
            Student student = new Student(
                    resultSet.getInt("id"),
                    resultSet.getString("name").trim(),
                    resultSet.getString("password").trim(),
                    resultSet.getInt("age"));
            ctx.collect(student);
        }
    }

    @Override
    public void cancel() {
    }

    private static Connection getConnection() {
        Connection con = null;
            try {
                Class.forName("com.mysql.jdbc.Driver");
                con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=UTF-8", "root", "root123456");
            } catch (Exception e) {
                System.out.println("----------mysql get connection has
exception , msg = "+ e.getMessage());
            }
        return con;
    }
}
```

Flink 程序：

```
package com.zhisheng.flink;

import com.zhisheng.flink.source.SourceFromMySQL;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

/**
 * blog: http://www.54tianzhisheng.cn/
 */
public class Main2 {
    public static void main(String[] args) throws Exception {
        final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();

        env.addSource(new SourceFromMySQL()).print();
```

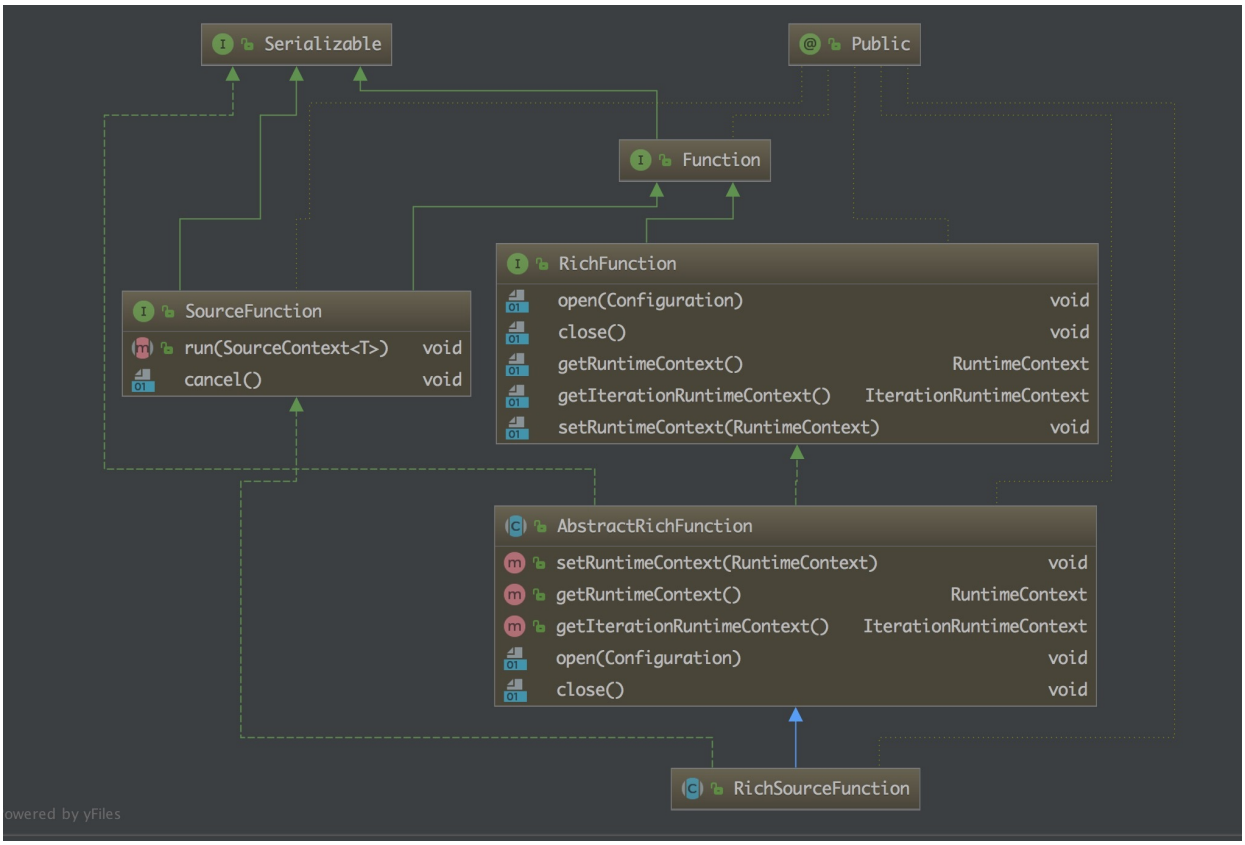
```
14
15              env.execute("Flink add data sourc");
16          }
17      }
```

运行 Flink 程序，控制台日志中可以看见打印的 student 信息。

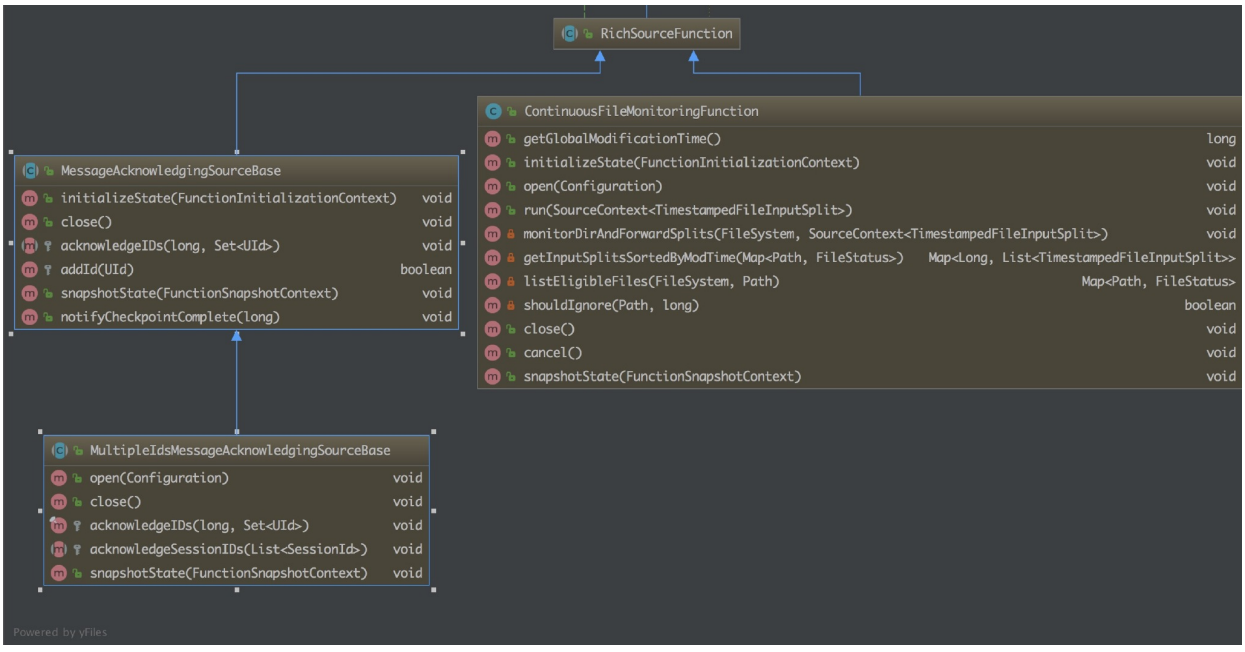


RichSourceFunction 讲解

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

```
import com.alibaba.fastjson.JSON;
import com.zhisheng.flink.model.Metric;
import com.zhisheng.flink.model.Student;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

/**
 * 往kafka中写数据
 * 可以使用这个main函数进行测试一下
 * blog: http://www.54tianzhisheng.cn/
 */
public class KafkaUtils2 {
    public static final String broker_list = "localhost:9092";
    public static final String topic = "student";  //kafka topic 需要和 flink 程序用同一个 topic

    public static void writeToKafka() throws InterruptedException {
        Properties props = new Properties();
        props.put("bootstrap.servers", broker_list);
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        KafkaProducer producer = new KafkaProducer<String, String>(props);

        for (int i = 1; i <= 100; i++) {
            Student student = new Student(i, "zhisheng" + i, "password" + i, 18 + i);
            ProducerRecord record = new ProducerRecord<String, String>(topic, null, null, JSON.toJSONString(student));
            producer.send(record);
            System.out.println("发送数据: " + JSON.toJSONString(student));
        }
        producer.flush();
    }

```

```
    public static void main(String[] args) throws InterruptedException {
        writeToKafka();
    }
}
```

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。

```
package com.zhisheng.flink.sink;

import com.zhisheng.flink.model.Student;
import org.apache.flink.configuration.Configuration;
import org.apache.flink.streaming.api.functions.sink.RichSinkFunction;

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

/**
 * blog: http://www.54tianzhisheng.cn/
 */
public class SinkToMySQL extends RichSinkFunction<Student> {
    PreparedStatement ps;
    private Connection connection;

    /**
     * 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
     *
     * @param parameters
     * @throws Exception
     */
    @Override
    public void open(Configuration parameters) throws Exception {
        super.open(parameters);
        connection = getConnection();
        String sql = "insert into Student(id, name, password, age) values(?, ?, ?, ?);";
        ps = this.connection.prepareStatement(sql);
    }

    @Override
    public void close() throws Exception {
        super.close();
        //关闭连接和释放资源
        if (connection != null) {
            connection.close();
        }
        if (ps != null) {
            ps.close();
        }
    }

    /**
     * 每条数据的插入都要调用一次 invoke() 方法
```

```
     *
     * @param value
     * @param context
     * @throws Exception
     */
    @Override
    public void invoke(Student value, Context context) throws Exception {
        //组装数据，执行插入操作
        ps.setInt(1, value.getId());
        ps.setString(2, value.getName());
        ps.setString(3, value.getPassword());
        ps.setInt(4, value.getAge());
        ps.executeUpdate();
    }

    private static Connection getConnection() {
        Connection con = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=UTF-8", "root", "root123456");
        } catch (Exception e) {
            System.out.println("-----------mysql get connection has
exception , msg = "+ e.getMessage());
        }
        return con;
    }
}
```

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。

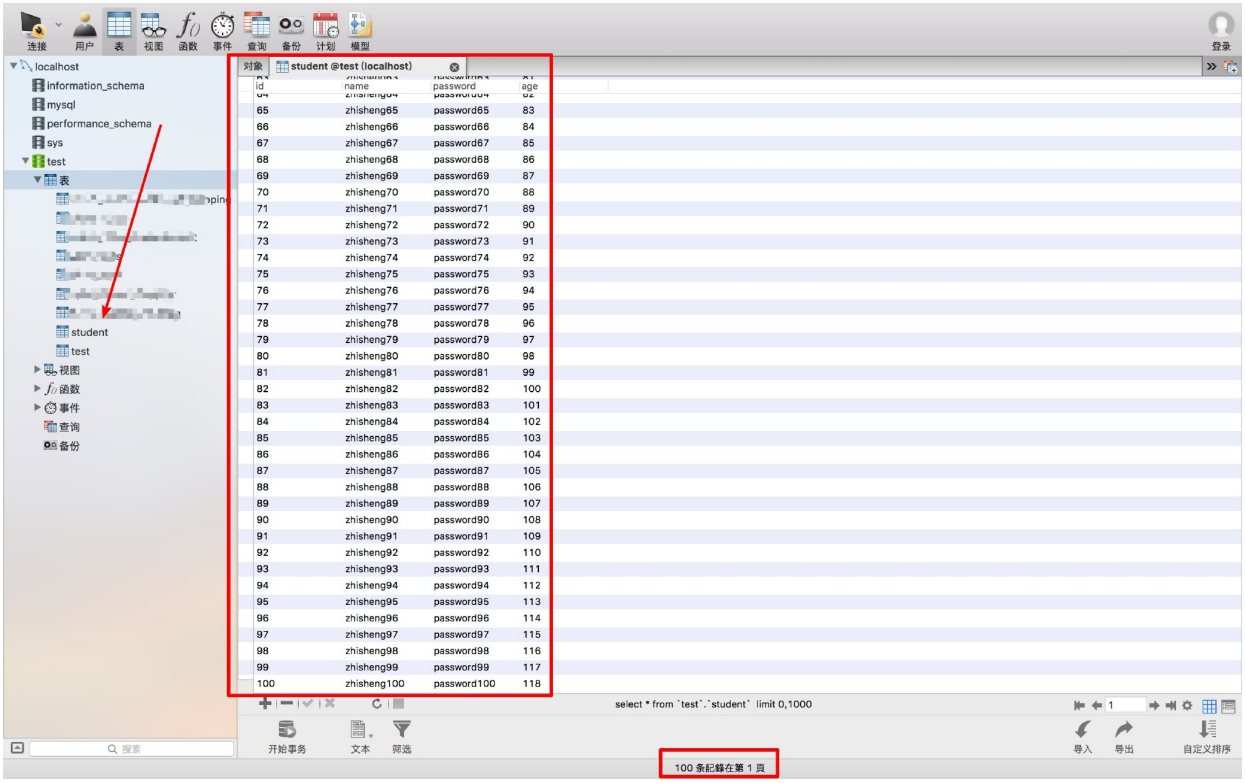
```
package com.zhisheng.flink;

import com.alibaba.fastjson.JSON;
import com.zhisheng.flink.model.Student;
import com.zhisheng.flink.sink.SinkToMySQL;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStreamSource;
import
org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.sink.PrintSinkFunction;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer011;
import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer011;

import java.util.Properties;

/**
 * blog: http://www.54tianzhisheng.cn/
 */
```

```
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：



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