toc: true title: 《从0到1学习Flink》—— 如何自定义 Data Source？ date: 2018-10-30 tags:

- Flink
- 大数据
- 流式计算

前言

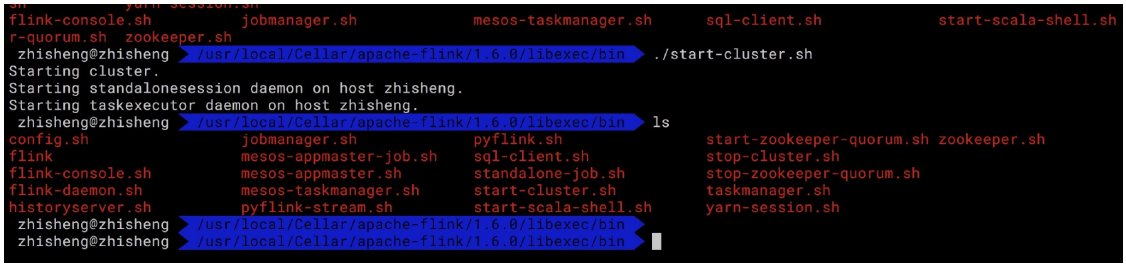
在 《从0到1学习Flink》—— Data Source 介绍 文章中，我给大家介绍了 Flink Data Source 以及简短的介绍了一下自定义 Data Source，这篇文章更详细的介绍下，并写一个 demo 出来让大家理解。

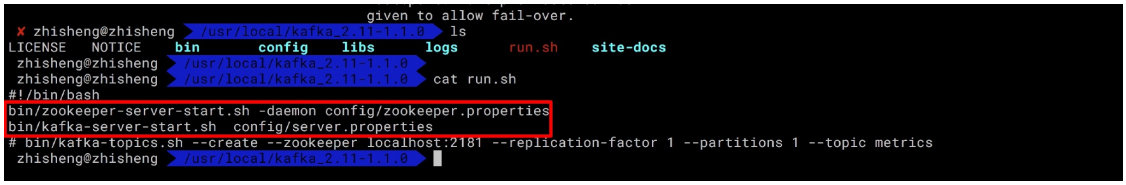
Flink Kafka source

准备工作

我们先来看下 Flink 从 Kafka topic 中获取数据的 demo，首先你需要安装好了 FLink 和 Kafka 。

运行启动 Flink、Zookepeer、Kafka，





好了，都启动了！

maven 依赖

```
<!--flink java-->
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-java</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-streaming-java_${scala.binary.version}
</artifactId>
    <version>${flink.version}</version>
    <scope>provided</scope>
</dependency>
<!--日志-->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.7</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
    <scope>runtime</scope>
</dependency>
<!--flink kafka connector-->
<dependency>
    <groupId>org.apache.flink</groupId>
    <artifactId>flink-connector-kafka-0.11_${scala.binary.version}
</artifactId>
    <version>${flink.version}</version>
</dependency>
<!--alibaba fastjson-->
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>fastjson</artifactId>
    <version>1.2.51</version>
</dependency>
```

测试发送数据到 kafka topic

实体类，Metric.java

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

import java.util.Map;

/**
 * Desc:
 * weixi: zhisheng_tian
 * blog: http://www.54tianzhisheng.cn/
 */
public class Metric {
    public String name;
    public long timestamp;
    public Map<String, Object> fields;
    public Map<String, String> tags;

    public Metric() {
    }

    public Metric(String name, long timestamp, Map<String, Object>
fields, Map<String, String> tags) {
        this.name = name;
        this.timestamp = timestamp;
        this.fields = fields;
        this.tags = tags;
    }

    @Override
    public String toString() {
        return "Metric{" +
                "name='" + name + '\'' +
                ", timestamp='" + timestamp + '\'' +
                ", fields=" + fields +
                ", tags=" + tags +
                '}';
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public long getTimestamp() {
        return timestamp;
    }
```

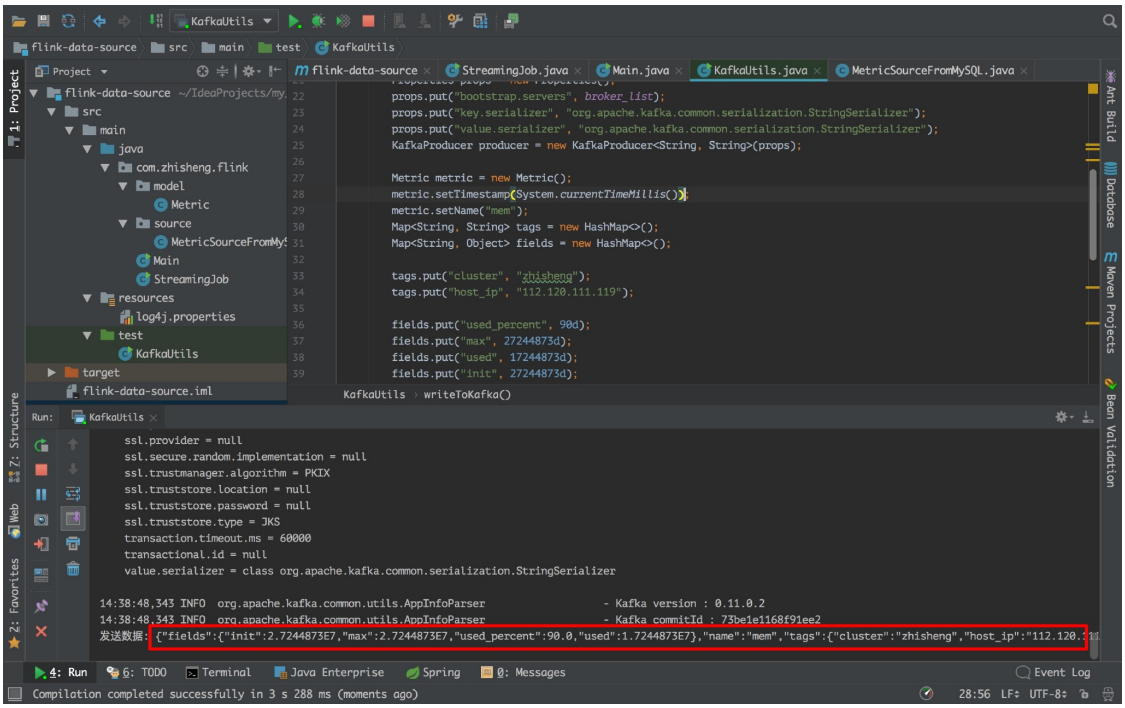
```
    public void setTimestamp(long timestamp) {
        this.timestamp = timestamp;
    }

    public Map<String, Object> getFields() {
        return fields;
    }

    public void setFields(Map<String, Object> fields) {
        this.fields = fields;
    }

    public Map<String, String> getTags() {
        return tags;
    }

    public void setTags(Map<String, String> tags) {
        this.tags = tags;
    }
}
```

往 kafka 中写数据工具类：KafkaUtils.java

```
import com.alibaba.fastjson.JSON;
import com.zhisheng.flink.model.Metric;
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
 * weixin: zhisheng_tian
 * blog: http://www.54tianzhisheng.cn/
 */
public class KafkaUtils {
    public static final String broker_list = "localhost:9092";
    public static final String topic = "metric";  // kafka topic,
Flink 程序中需要和这个统一

    public static void writeToKafka() throws InterruptedException {
        Properties props = new Properties();
        props.put("bootstrap.servers", broker_list);
        props.put("key.serializer"
```

```
        props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer"); //key 序
列化
        props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer"); //value
序列化
        KafkaProducer producer = new KafkaProducer<String, String>
(props);

        Metric metric = new Metric();
        metric.setTimestamp(System.currentTimeMillis());
        metric.setName("mem");
        Map<String, String> tags = new HashMap<>();
        Map<String, Object> fields = new HashMap<>();

        tags.put("cluster", "zhisheng");
        tags.put("host_ip", "101.147.022.106");

        fields.put("used_percent", 90d);
        fields.put("max", 27244873d);
        fields.put("used", 17244873d);
        fields.put("init", 27244873d);

        metric.setTags(tags);
        metric.setFields(fields);

        ProducerRecord record = new ProducerRecord<String, String>
(topic, null, null, JSON.toJSONString(metric));
        producer.send(record);
        System.out.println("发送数据: " + JSON.toJSONString(metric));

        producer.flush();
    }

    public static void main(String[] args) throws
InterruptedException {
        while (true) {
            Thread.sleep(300);
            writeToKafka();
        }
    }
}
```

运行：



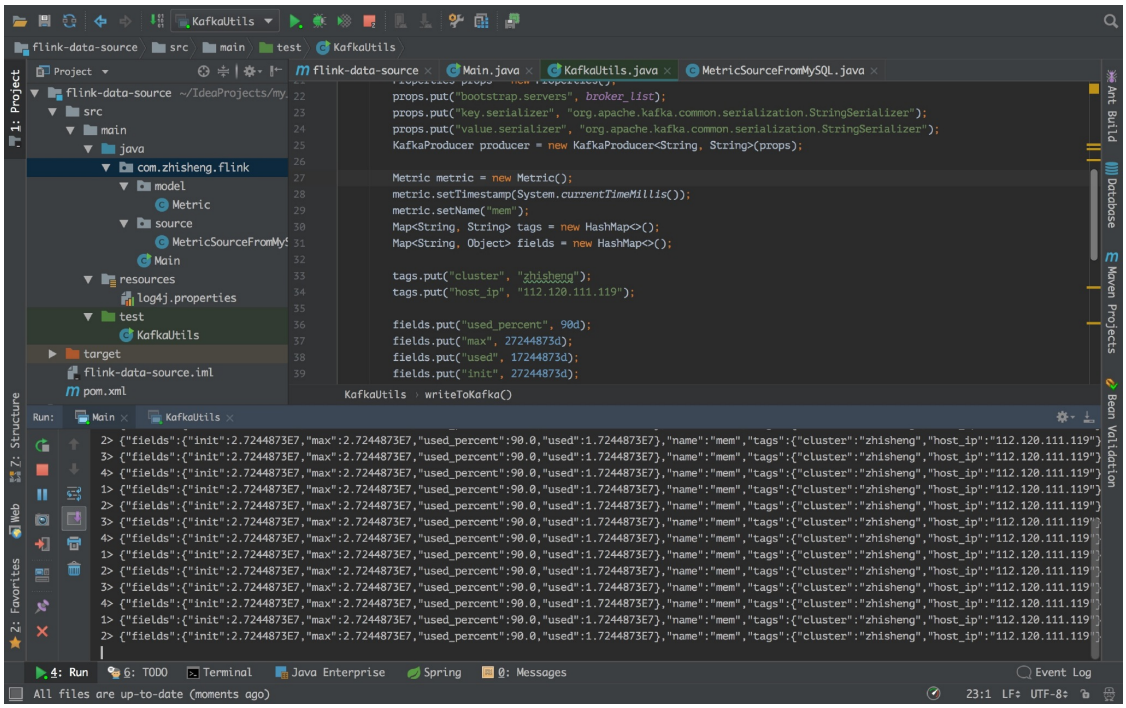
如果出现如上图标记的，即代表能够不断的往 kafka 发送数据的。

Flink 程序

Main.java

```
package com.zhisheng.flink;


import
org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStreamSource;
import
org.apache.flink.streaming.api.environment.StreamExecutionEnvironme
nt;
import
org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer011;

import java.util.Properties;

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

        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("zookeeper.connect", "localhost:2181");
        props.put("group.id", "metric-group");
        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");  //key
反序列化
        props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
        props.put("auto.offset.reset", "latest"); //value 反序列化

        DataStreamSource<String> dataStreamSource =
env.addSource(new FlinkKafkaConsumer011<>(
                "metric",  //kafka topic
                new SimpleStringSchema(),  // String 序列化
                props)).setParallelism(1);

        dataStreamSource.print(); //把从 kafka 读取到的数据打印在控制台

        env.execute("Flink add data source");
    }
}
```

运行起来：



看到没程序，Flink 程序控制台能够源源不断的打印数据呢。

自定义 Source

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：

```sql
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：

```sql
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

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

/**
 * Desc:
 * weixi: zhisheng_tian
 * blog: http://www.54tianzhisheng.cn/
 */
public class Student {
    public int id;
    public String name;
    public String password;
    public int age;

    public Student() {
    }

    public Student(int id, String name, String password, int age) {
        this.id = id;
        this.name = name;
        this.password = password;
        this.age = age;
    }
```

```
    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", password='" + password + '\'' +
                ", age=" + age +
                '}';
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

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

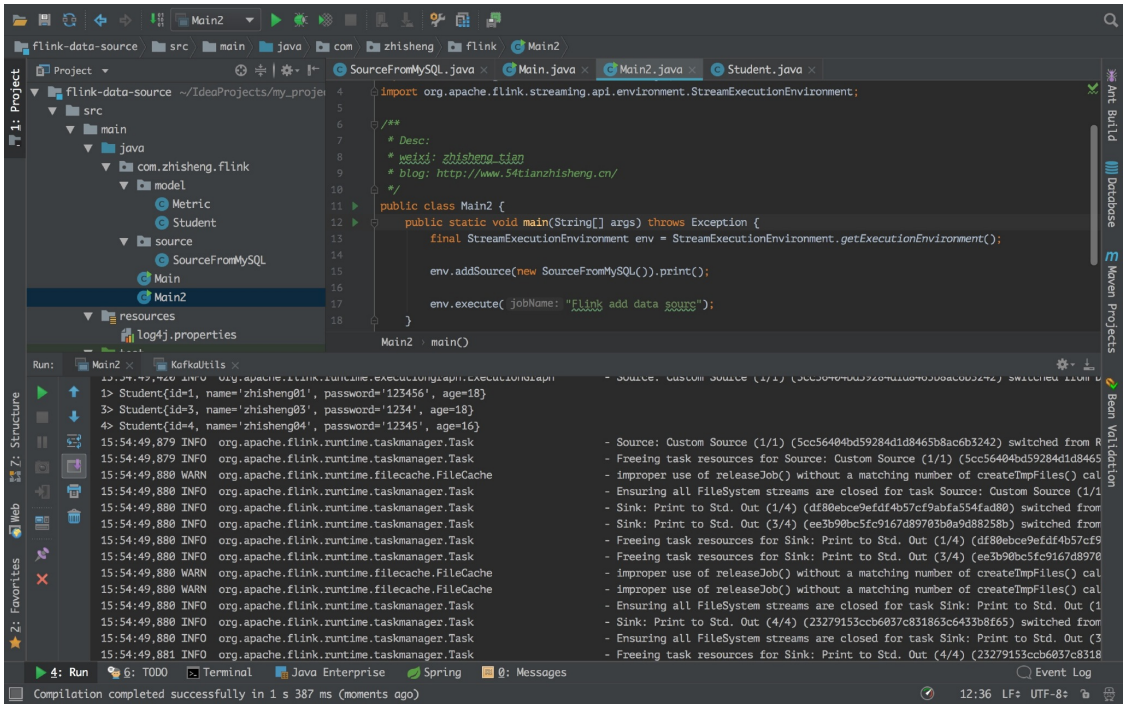
```java
package com.zhisheng.flink.source;


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;


/**
* Desc:
* weixi: zhisheng_tian
* 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：

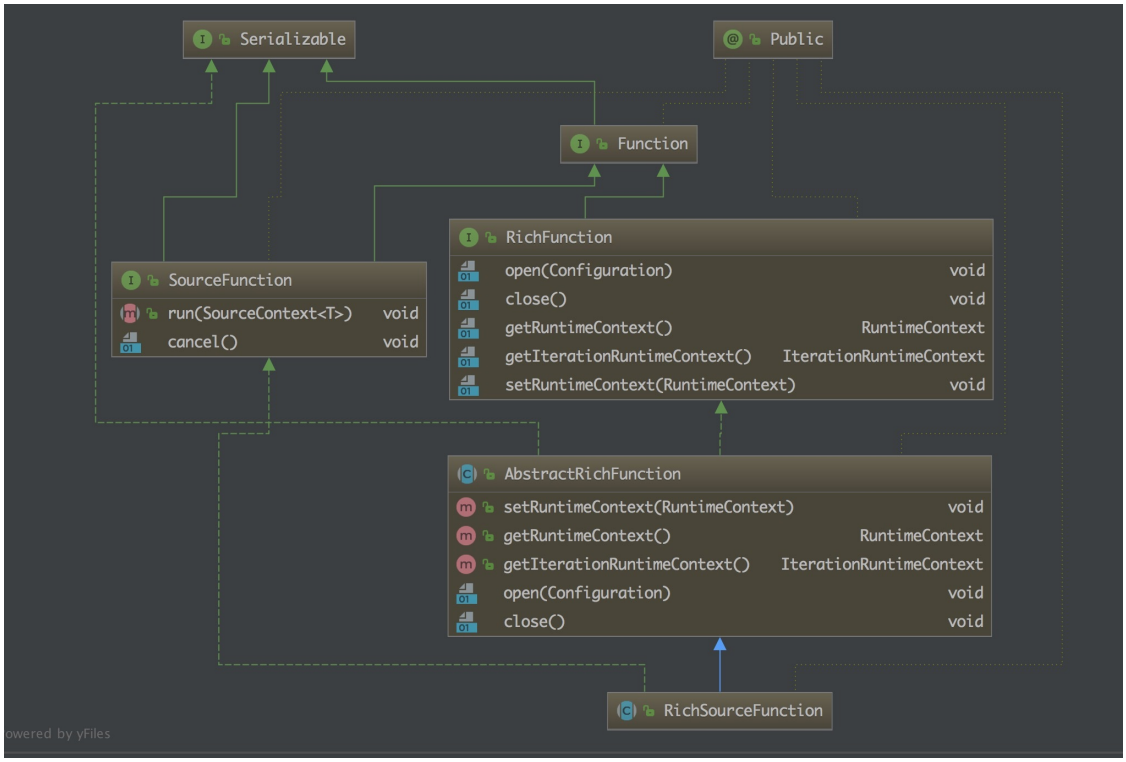
```
package com.zhisheng.flink;

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

/**
 * Desc:
 * weixi: zhisheng_tian
 * 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();

        env.execute("Flink add data sourc");
    }
}
```

运行 Flink 程序，控制台日志中可以看见打印的 student 信息。

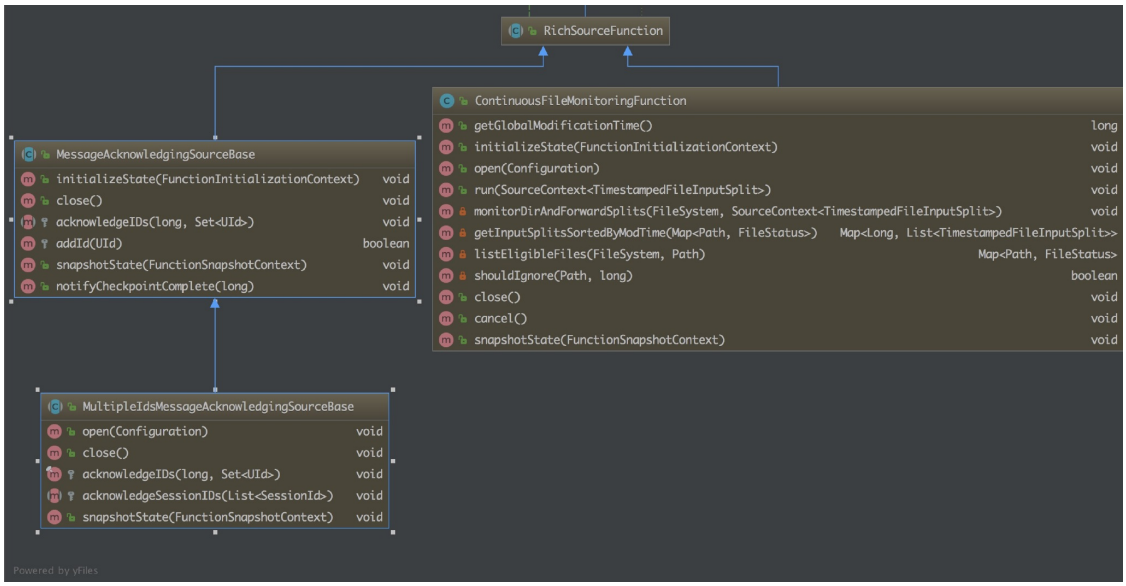


RichSourceFunction

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。

最后

本文主要讲了下 Flink 使用 Kafka Source 的使用，并提供了一个 demo 教大家如何自定义 Source，从 MySQL 中读取数据，当然你也可以从其他地方读取，实现自己的数据源 source。可能平时工作会比这个更复杂，需要大家灵活应对！

关注我

转载请务必注明原创地址为：http://www.54tianzhisheng.cn/2018/10/30/flink-create-source/

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。

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，包含了自己学习 flink 的一些 demo 和博客

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