

• 导入依赖

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>sz.base.flink</groupId>
8      <artifactId>flink-base-sz28</artifactId>
9      <packaging>pom</packaging>
10     <version>1.0-SNAPSHOT</version>
11     <modules>
12         <module>day01_wordcount</module>
13     </modules>
14
15
16     <repositories>
17         <repository>
18             <releases>
19                 <enabled>true</enabled>
20                 <updatePolicy>never</updatePolicy>
21                 <checksumPolicy>fail</checksumPolicy>
22             </releases>
23             <snapshots>
24                 <enabled>false</enabled>
25                 <updatePolicy>always</updatePolicy>
26                 <checksumPolicy>warn</checksumPolicy>
27             </snapshots>
28             <id>cdh.repo</id>
29             <name>Cloudera Repositories</name>
30             <url>https://repository.cloudera.com/artifactory/cloudera-repos</url>
31             <layout>default</layout>
32         </repository>
33
34         <repository>
35             <id>spring</id>
36             <url>https://repo.spring.io/plugins-release/</url>
```

```
37         <releases>
38             <updatePolicy>always</updatePolicy>
39         </releases>
40     </repository>
41 </repositories>
42
43 <properties>
44     <flink.version>1.13.1</flink.version>
45     <java.version>1.8</java.version>
46     <scala.binary.version>2.11</scala.binary.version>
47     <hadoop.version>3.3.0</hadoop.version>
48     <hbase.version>2.0.0</hbase.version>
49     <zkclient.version>0.8</zkclient.version>
50     <hive.version>2.1.1</hive.version>
51     <mysql.version>5.1.48</mysql.version>
52     <log4j.version>1.7.32</log4j.version>
53     <logback.version>1.2.6</logback.version>
54 </properties>
55
56
57 <dependencies>
58     <!-- Apache Flink dependencies -->
59     <!-- These dependencies are provided, because they should not be packaged into
the JAR file. -->
60     <!--<dependency>
61         <groupId>org.apache.flink</groupId>
62         <artifactId>flink-java</artifactId>
63         <version>${flink.version}</version>
64         &lt;!--&dash;<scope>provided</scope>&dash;&gt;
65     </dependency>-->
66     <!-- https://mvnrepository.com/artifact/junit/junit -->
67     <dependency>
68         <groupId>junit</groupId>
69         <artifactId>junit</artifactId>
70         <version>4.13.2</version>
71         <scope>test</scope>
72     </dependency>
73     <dependency>
74         <groupId>org.apache.flink</groupId>
75         <artifactId>flink-streaming-scala_${scala.binary.version}</artifactId>
```

```
76         <version>${flink.version}</version>
77         <!--<scope>compile</scope>-->
78     </dependency>
79
80     <dependency>
81         <groupId>org.apache.flink</groupId>
82         <artifactId>flink-runtime-web_${scala.binary.version}</artifactId>
83         <version>${flink.version}</version>
84     </dependency>
85
86     <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-clients -->
87     <dependency>
88         <groupId>org.apache.flink</groupId>
89         <artifactId>flink-clients_${scala.binary.version}</artifactId>
90         <version>${flink.version}</version>
91     </dependency>
92
93     <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-table -->
94     <dependency>
95         <groupId>org.apache.flink</groupId>
96         <artifactId>flink-table-planner_2.11</artifactId>
97         <version>${flink.version}</version>
98     </dependency>
99
100    <dependency>
101        <groupId>org.apache.flink</groupId>
102        <artifactId>flink-table-planner-blink_2.11</artifactId>
103        <version>${flink.version}</version>
104    </dependency>
105
106    <dependency>
107        <groupId>org.apache.flink</groupId>
108        <artifactId>flink-table-runtime-blink_2.11</artifactId>
109        <version>${flink.version}</version>
110    </dependency>
111
112    <dependency>
113        <groupId>org.apache.flink</groupId>
114        <artifactId>flink-table-api-scala-bridge_2.11</artifactId>
```

```
115         <version>${flink.version}</version>
116     </dependency>
117
118
119     <dependency>
120         <groupId>org.apache.flink</groupId>
121         <artifactId>flink-java</artifactId>
122         <version>${flink.version}</version>
123     </dependency>
124     <!--kafka-->
125     <dependency>
126         <groupId>org.apache.flink</groupId>
127         <artifactId>flink-connector-kafka_2.11</artifactId>
128         <version>${flink.version}</version>
129     </dependency>
130     <dependency>
131         <groupId>org.apache.flink</groupId>
132         <artifactId>flink-sql-connector-kafka_2.11</artifactId>
133         <version>${flink.version}</version>
134     </dependency>
135
136     <dependency>
137         <groupId>org.apache.flink</groupId>
138         <artifactId>flink-queryable-state-runtime_2.11</artifactId>
139         <version>${flink.version}</version>
140     </dependency>
141
142     <!-- https://mvnrepository.com/artifact/org.apache.flink/flink-statebackend-
rocksdb -->
143     <dependency>
144         <groupId>org.apache.flink</groupId>
145         <artifactId>flink-statebackend-rocksdb_2.11</artifactId>
146         <version>${flink.version}</version>
147     </dependency>
148
149     <dependency>
150         <groupId>org.apache.kafka</groupId>
151         <artifactId>kafka-clients</artifactId>
152         <version>1.0.0</version>
153
```

```
154         </dependency>
155         <!--es6-->
156         <dependency>
157             <groupId>org.apache.flink</groupId>
158             <artifactId>flink-connector-elasticsearch6_${scala.binary.version}</artifactId>
159             <version>${flink.version}</version>
160         </dependency>
161
162         <!--flink-jdbc-->
163         <dependency>
164             <groupId>org.apache.flink</groupId>
165             <artifactId>flink-connector-jdbc_2.11</artifactId>
166             <version>${flink.version}</version>
167         </dependency>
168
169         <!--flink-hbase-->
170
171         <dependency>
172             <groupId>org.apache.flink</groupId>
173             <artifactId>flink-connector-hbase-2.2_2.11</artifactId>
174             <version>${flink.version}</version>
175             <exclusions>
176                 <exclusion>
177                     <artifactId>slf4j-api</artifactId>
178                     <groupId>org.slf4j</groupId>
179                 </exclusion>
180             </exclusions>
181         </dependency>
182
183
184         <!--hadoop-->
185         <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common -->
186         <dependency>
187             <groupId>org.apache.hadoop</groupId>
188             <artifactId>hadoop-common</artifactId>
189             <version>${hadoop.version}</version>
190             <exclusions>
191                 <exclusion>
192                     <artifactId>slf4j-api</artifactId>
```

```
193         <groupId>org.slf4j</groupId>
194     </exclusion>
195     <exclusion>
196         <artifactId>slf4j-log4j12</artifactId>
197         <groupId>org.slf4j</groupId>
198     </exclusion>
199 </exclusions>
200 </dependency>
201 <dependency>
202     <groupId>org.apache.flink</groupId>
203     <artifactId>flink-hadoop-compatibility_2.11</artifactId>
204     <version>${flink.version}</version>
205 </dependency>
206 <dependency>
207     <groupId>org.apache.hadoop</groupId>
208     <artifactId>hadoop-client</artifactId>
209     <version>${hadoop.version}</version>
210     <exclusions>
211         <exclusion>
212             <artifactId>slf4j-api</artifactId>
213             <groupId>org.slf4j</groupId>
214         </exclusion>
215     </exclusions>
216 </dependency>
217
218 <dependency>
219     <groupId>org.apache.flink</groupId>
220     <artifactId>flink-shaded-hadoop-2-uber</artifactId>
221     <version>2.7.5-10.0</version>
222     <exclusions>
223         <exclusion>
224             <artifactId>slf4j-log4j12</artifactId>
225             <groupId>org.slf4j</groupId>
226         </exclusion>
227     </exclusions>
228 </dependency>
229
230 <dependency>
231     <groupId>org.apache.flink</groupId>
232     <artifactId>flink-csv</artifactId>
```

```
233         <version>${flink.version}</version>
234     </dependency>
235
236     <!--flink-hbase-->
237     <dependency>
238         <groupId>org.apache.flink</groupId>
239         <artifactId>flink-json</artifactId>
240         <version>${flink.version}</version>
241         <!--<scope>test</scope>-->
242     </dependency>
243
244     <dependency>
245         <groupId>org.apache.flink</groupId>
246         <artifactId>flink-runtime_${scala.binary.version}</artifactId>
247         <version>${flink.version}</version>
248         <!--<scope>test</scope>-->
249     </dependency>
250
251     <!-- log4j日志 start-->
252     <dependency>
253         <groupId>ch.qos.logback</groupId>
254         <artifactId>logback-core</artifactId>
255         <version>${logback.version}</version>
256     </dependency>
257     <dependency>
258         <groupId>ch.qos.logback</groupId>
259         <artifactId>logback-classic</artifactId>
260         <version>${logback.version}</version>
261     </dependency>
262     <dependency>
263         <groupId>org.slf4j</groupId>
264         <artifactId>log4j-over-slf4j</artifactId>
265         <version>${log4j.version}</version>
266         <exclusions>
267             <exclusion>
268                 <artifactId>slf4j-api</artifactId>
269                 <groupId>org.slf4j</groupId>
270             </exclusion>
271         </exclusions>
272     </dependency>
```

```
273
274     <!-- json -->
275     <dependency>
276         <groupId>com.alibaba</groupId>
277         <artifactId>fastjson</artifactId>
278         <version>1.2.5</version>
279     </dependency>
280
281     <!-- On hive -->
282     <!-- Flink Dependency -->
283
284     <dependency>
285         <groupId>org.apache.flink</groupId>
286         <artifactId>flink-connector-hive_${scala.binary.version}</artifactId>
287         <version>${flink.version}</version>
288     </dependency>
289     <!-- Hive Dependency -->
290     <dependency>
291         <groupId>org.apache.hive</groupId>
292         <artifactId>hive-exec</artifactId>
293         <version>${hive.version}</version>
294         <exclusions>
295             <exclusion>
296                 <artifactId>log4j-slf4j-impl</artifactId>
297                 <groupId>org.apache.logging.log4j</groupId>
298             </exclusion>
299         </exclusions>
300     </dependency>
301
302     <dependency>
303         <groupId>org.projectlombok</groupId>
304         <artifactId>lombok</artifactId>
305         <version>1.18.2</version>
306         <scope>provided</scope>
307     </dependency>
308
309     <!-- mysql 连接驱动 -->
310     <dependency>
311         <groupId>mysql</groupId>
```



```
312         <artifactId>mysql-connector-java</artifactId>
313         <version>${mysql.version}</version>
314     </dependency>
315     <dependency>
316         <groupId>junit</groupId>
317         <artifactId>junit</artifactId>
318         <version>RELEASE</version>
319         <scope>compile</scope>
320     </dependency>
321
322     <!-- redis客户端 -->
323     <dependency>
324         <groupId>redis.clients</groupId>
325         <artifactId>jedis</artifactId>
326         <version>2.9.0</version>
327     </dependency>
328     <!-- flink 连接 redis 的 connector -->
329     <dependency>
330         <groupId>org.apache.bahir</groupId>
331         <artifactId>flink-connector-redis_2.11</artifactId>
332         <version>1.0</version>
333     </dependency>
334 </dependencies>
335
336 <build>
337     <!-- 默认加载此目录，作为source目录-->
338     <sourceDirectory>src/main/java</sourceDirectory>
339     <plugins>
340         <!-- java编译插件 -->
341         <plugin>
342             <groupId>org.apache.maven.plugins</groupId>
343             <artifactId>maven-compiler-plugin</artifactId>
344             <version>3.6.1</version>
345             <configuration>
346                 <source>${java.version}</source>
347                 <target>${java.version}</target>
348                 <encoding>UTF-8</encoding>
349             </configuration>
350         </plugin>
351         <!-- scala编译插件 -->
```

```
352     <plugin>
353         <groupId>net.alchim31.maven</groupId>
354         <artifactId>scala-maven-plugin</artifactId>
355         <version>4.0.2</version>
356         <executions>
357             <execution>
358                 <id>compile-scala</id>
359                 <phase>compile</phase>
360                 <goals>
361                     <goal>add-source</goal>
362                     <goal>compile</goal>
363                 </goals>
364             </execution>
365         </executions>
366     </plugin>
367     <!-- 打jar包插件(会包含所有依赖) -->
368     <plugin>
369         <groupId>org.apache.maven.plugins</groupId>
370         <artifactId>maven-assembly-plugin</artifactId>
371         <version>2.6</version>
372         <configuration>
373             <descriptorRefs>
374                 <descriptorRef>jar-with-dependencies</descriptorRef>
375             </descriptorRefs>
376             <archive>
377                 <manifest>
378                     <!-- 可以设置jar包的入口类(可选) -->
379     <mainClass>cn.itcast.flink.checkpoint.SavepointDemo</mainClass>
380                 </manifest>
381             </archive>
382         </configuration>
383         <executions>
384             <execution>
385                 <id>make-assembly</id>
386                 <phase>package</phase>
387                 <goals>
388                     <goal>single</goal>
389                 </goals>
390             </execution>
```

```
391         </executions>
392     </plugin>
393 </plugins>
394 </build>
395 </project>
```

• ExecutionEnvironment

```
1  package sz.base.flink.wordcount;
2
3
4  import org.apache.flink.api.common.functions.FlatMapFunction;
5  import org.apache.flink.api.common.functions.MapFunction;
6  import org.apache.flink.api.java.ExecutionEnvironment;
7  import org.apache.flink.api.java.operators.AggregateOperator;
8  import org.apache.flink.api.java.operators.DataSource;
9  import org.apache.flink.api.java.operators.FlatMapOperator;
10 import org.apache.flink.api.java.operators.MapOperator;
11 import org.apache.flink.api.java.tuple.Tuple2;
12 import org.apache.flink.util.Collector;
13
14 /**
15  * 编写flink程序，读取文件中的字符串，并以空格进行单词拆分打印
16  * 1. 获取批的执行环境
17  * 2. 读取文件数据
18  * 3. 将读取到文件数据进行拆分 hello,world,flink,hadoop,hello
19  * 4. 将拆分的单词转换成(hello,1)(world,)
20  * 5. 需要根据单词进行分组
21  * 6. 根据组内进行统计求和
22  * 7. 将结果打印输出到控制台
23  */
24 public class WordcountBatchDemo {
25     public static void main(String[] args) throws Exception {
26
27         //      * 1. 获取批的执行环境
28         ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();
29         //      * 2. 读取文件数据
30         DataSource<String> source = env.readTextFile("input/words.txt");
31         //      * 3. 将读取到文件数据进行拆分 hello,world,flink,hadoop,hello
32         source.flatMap(new FlatMapFunction<String, String>() {
```

```

33         @Override
34         public void flatMap(String s, Collector<String> collector) throws Exception
35     {
36         String[] words = s.split(",");
37         for (String word : words) {
38             collector.collect(word);
39         }
40         //          * 4.将拆分的单词转换成(hello,1)(world,)
41     }).map(new MapFunction<String, Tuple2<String, Integer>>() {
42         @Override
43         public Tuple2<String, Integer> map(String s) throws Exception {
44             return Tuple2.of(s, 1);
45         }
46         //          * 5.需要根据单词进行分组
47     //          * 6.根据组内进行统计求和
48     }).groupBy(0).sum(1).print();
49     //          * 7.将结果打印输出到控制台
50     env.execute();
51
52 }
53 }
54

```

• StreamExecutionEnvironment

```

1  package sz.base.flink.wordcount;
2
3  import org.apache.flink.api.common.functions.FlatMapFunction;
4  import org.apache.flink.api.common.functions.MapFunction;
5  import org.apache.flink.api.java.functions.KeySelector;
6  import org.apache.flink.api.java.tuple.Tuple2;
7  import org.apache.flink.streaming.api.datastream.DataStreamSource;
8  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
9  import org.apache.flink.util.Collector;
10
11 public class WordcountStreamDemo {
12     public static void main(String[] args) throws Exception {
13         //1.创建流执行环境

```

```

14      StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
15      //2.读取socket数据源
16      DataStreamSource<String> node1 = env.socketTextStream("192.168.88.161", 9999);
17      //3.对读取进来的数据进行flatMap拆分
18      node1.flatMap(new FlatMapFunction<String, String>() {
19          @Override
20          public void flatMap(String s, Collector<String> collector) throws Exception
{
21              String[] words = s.split(" ");
22              for (String word : words) {
23                  collector.collect(word);
24              }
25          }
26      }).map(new MapFunction<String, Tuple2<String, Integer>>() {
27          @Override
28          public Tuple2<String, Integer> map(String s) throws Exception {
29              return Tuple2.of(s, 1);
30          }
31      }).keyBy(new KeySelector<Tuple2<String, Integer>, String>() {
32          @Override
33          public String getKey(Tuple2<String, Integer> stringIntegerTuple2) throws
Exception {
34              return stringIntegerTuple2.f0;
35          }
36      }).sum(1).print();
37      env.execute();
38  }
39 }
40

```

• StreamLambda

```

1  package sz.base.flink.wordcount;
2
3  import org.apache.flink.api.common.typeinfo.Types;
4  import org.apache.flink.api.java.tuple.Tuple2;
5  import org.apache.flink.streaming.api.datastream.DataStreamSource;
6  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7  import org.apache.flink.util.Collector;
8

```

```

9
10 import java.util.Arrays;
11
12 public class WordCountLambdaDemo {
13     public static void main(String[] args) throws Exception {
14         StreamExecutionEnvironment env =
15         StreamExecutionEnvironment.getExecutionEnvironment();
16         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
17         node1.flatMap((String var1, Collector<String> var2) ->
18             Arrays.stream(var1.split(" ")).forEach(var2::collect)
19             ).returns(Types.STRING).map(t -> Tuple2.of(t,
20             1)).returns(Types.TUPLE(Types.STRING, Types.INT)).keyBy(k -> k.f0).sum(1).print();
21         env.execute();
22     }
23 }
24

```

• source集合

```

1 package sz.base.flink.source;
2
3 import org.apache.flink.streaming.api.datastream.DataStreamSource;
4 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
5
6 import java.util.ArrayList;
7 import java.util.List;
8
9 public class ElementListDemo2 {
10     public static void main(String[] args) throws Exception {
11         // * 1.创建流执行环境
12         StreamExecutionEnvironment env =
13         StreamExecutionEnvironment.getExecutionEnvironment();
14         env.setParallelism(1);
15         /* 2.创建本地的数据流
16         //      DataStreamSource<Integer> source = env.fromElements(1, 2, 3, 4, 5, 6, 7, 8,
17         //      9,10);
18         //从collection中获取集合
19         List<Student> studentList = new ArrayList<>();
20         studentList.add(new Student("zhangsan", 22));
21         studentList.add(new Student("lisi", 24));
22         DataStreamSource<Student> source = env.fromCollection(studentList);
23

```

```
21         //接受一个data
22         DataSourceSource<Integer> source1 = env.fromElements(1, 2, 3, 4, 5, 6, 7, 8, 9,
0, 10);
23         //接受一个范围
24         DataSourceSource<Long> source2 = env.fromSequence(1, 100);
25 //      * 3.打印输出数据流
26         source.printToErr();
27 //      * 4.执行流环境
28         env.execute();
29     }
30
31     public static class Student {
32         private String name;
33         private int age;
34
35         public Student(String name, int age) {
36             this.name = name;
37             this.age = age;
38         }
39
40         public String getName() {
41             return name;
42         }
43
44         public void setName(String name) {
45             this.name = name;
46         }
47
48         public int getAge() {
49             return age;
50         }
51
52         @Override
53         public String toString() {
54             return "当前学生: "+ this.name+" 的年龄为: "+this.age;
55         }
56
57         public void setAge(int age) {
58             this.age = age;
```

```
59
60
61
62
63
64
65
66
```

```
    }
}
```

• source读取文件

```
1  package sz.base.flink.file;
2
3  import org.apache.flink.api.java.io.TextInputFormat;
4  import org.apache.flink.streaming.api.datastream.DataStreamSource;
5  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6  import org.apache.flink.streaming.api.functions.source.FileProcessingMode;
7
8  /**
9   * 读取文件中的数据
10  * 1. 读取一个普通的文本文件（once），静态文件
11  * 2. 读取一个文本文件（once），readFile，静态文件
12  * 3. 读取一个文本文件，每5s钟重新加载一下文件读取，读取流文本数据
13  * 开发步骤：
14  * 1. 创建流执行环境
15  * 2. 设置并行度等参数
16  * 3. 读取文本文件的数据源
17  * 4. 打印输出文本文件的内容
18  * 5. 执行流环境
19  */
20  public class TextFile {
21      public static void main(String[] args) throws Exception {
22          //      * 1. 创建流执行环境
23          StreamExecutionEnvironment env =
24              StreamExecutionEnvironment.getExecutionEnvironment();
25          //      * 2. 设置并行度等参数
26          env.setParallelism(1);
27          //      * 3. 读取文本文件的数据源
28          // 用于批处理任务，数据时静态的，有界的
29          DataStreamSource<String> source = env.readTextFile("input/abc.csv");
```



```

29         //读取一个文本文件，readFile静态文件
30         //      DataSource<String> source = env.readFile(new TextInputFormat(null),
31         "input/abc.csv");
32         //每隔5s读取一次,会把所有文件重新读取一次，只有文件内容变化才会重新打印输出一次，
33         DataSource<String> source = env.readFile(new TextInputFormat(null),
34         "input/abc.csv", FileProcessingMode.PROCESS_CONTINUOUSLY, 5000L);
35         //只读取一次
36         DataSource<String> source1 = env.readFile(new TextInputFormat(null),
37         "input/abc.csv", FileProcessingMode.PROCESS_ONCE, 5000L);
38         //      * 4.打印输出文本文件的内容
39         source.printToErr();
40         //      * 5.执行流环境
41         env.execute("读取文本文件的 job");
42     }
43 }
44

```

ParallelSourceFunction 接口案例

• 自定义source

```

1  package sz.base.flink.cs;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7  import org.apache.flink.streaming.api.functions.source.SourceFunction;
8
9  import java.text.SimpleDateFormat;
10 import java.util.Random;
11 import java.util.UUID;
12
13 /**
14  * 每1s钟随机生成一条订单信息（订单ID、用户ID、订单金额、时间戳）
15  * 开发步骤：
16  * 1.创建流执行环境
17  * 2.实现一个SourceFunction类，重写run方法，实现每一秒钟打印一条数据
18  * 3.生成订单，封装一个Order的类
19  * 4.将这个自定义生成的订单数据流打印输出

```

```

20  * 5.执行流环境
21  */
22  public class CustomOrder1 {
23      public static void main(String[] args) throws Exception {
24          //      * 1.创建流执行环境
25              StreamExecutionEnvironment env =
                StreamExecutionEnvironment.getExecutionEnvironment();
26              env.setParallelism(1);
27          //      * 2.实现一个SourceFunction类，重写run方法，实现每一秒钟打印一条数据
28              env.addSource(new CustomSource()).printToErr();
29          //      * 3.生成订单，封装一个Order的类
30          //      * 4.将这个自定义生成的订单数据流打印输出
31          //      * 5.执行流环境
32              env.execute();
33      }
34      public static class CustomSource implements SourceFunction<Order>{
35          //定义一个标记
36          private boolean isRuning=true;
37          Random rm=new Random();
38          //定义一个格式化工具
39          SimpleDateFormat sdf= new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.sss");
40          /**
41           * 生成自定义数据源的业务逻辑
42           * @param sourceContext 收集器
43           * @throws Exception
44           */
45          @Override
46          public void run(SourceContext<Order> sourceContext) throws Exception {
47              //每1s钟随机生成一条订单信息（订单ID、用户ID、订单金额、时间戳）
48              while (isRuning){
49                  Order order=new Order(
50                      UUID.randomUUID().toString(),
51                      rm.nextInt(3),
52                      rm.nextInt(101),
53                      System.currentTimeMillis(),
54                      sdf.format(System.currentTimeMillis()))
55                  );
56                  sourceContext.collect(order);
57                  Thread.sleep(1000);
58              }

```

```

59         }
60
61         /**
62          * 用户取消生成自定义数据源的方式
63          */
64         @Override
65         public void cancel() {
66             isRuning=false;
67         }
68     }
69
70
71     @Data
72     @AllArgsConstructor
73     @NoArgsConstructor
74     public static class Order{
75         //随机生成订单ID (UUID)
76         private String oid;
77         //随机生成用户ID (0-2)
78         private int uid;
79         //随机生成订单金额 (0-100)
80         private int money;
81         //时间戳为当前系统时间
82         private long timestamp;
83         //当前时间
84         private String datetime;
85
86     }
87 }
88
89

```

RichParallelSourceFunction案例

• 读取MySQL

- Rich 是富函数继承了 AbstractRichFunciton，实现了
- 生命周期的 open 和 close 方法
 - a. open 方法，用于实现当前生成的初始化条件

b. close 方法，用于生成数据结束的收尾工作

c. getRuntimeContext 方法，用于获取当前的程序的上下文对象（参数、环境变量、状态、累加器等）

```
1 package sz.base.flink.cs;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import org.apache.flink.configuration.Configuration;
6 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7 import org.apache.flink.streaming.api.functions.source.RichParallelSourceFunction;
8
9 import java.sql.Connection;
10 import java.sql.DriverManager;
11 import java.sql.ResultSet;
12 import java.sql.Statement;
13
14 /**
15  * 读取数据库中的数据表数据并打印输出
16  * 开发步骤：
17  * 1. 获取流执行环境
18  * 2. 设置参数设置并行度
19  * 3. 读取自定义数据源RichSourceFunction
20  * RichSourceFunction与SourceFunction多了个AbstractRichFunction抽象类，多了两个方法：
21  * 称为生命周期方法
22  * open ()：开启链接
23  * close ()：关闭链接
24  * 4. 自定义一个类User用于接收返回的数据
25  * 5. 打印输出User对象
26  * 6. 执行流环境
27  */
28 public class UserSource {
29     public static void main(String[] args) throws Exception {
30         // 1.
31         StreamExecutionEnvironment env =
32             StreamExecutionEnvironment.getExecutionEnvironment();
33         env.setParallelism(1);
34         env.addSource(new RichParallelSourceFunction<User>() {
35             private Connection conn=null;
36             private Statement statement=null;
37             private boolean isRunning=true;
```

```

37      /**
38       * 生命周期的开始，再次方法中注意用于定义初始化的操作
39       * 初始化的动作就是连接数据库，就连接一次
40       * @param parameters
41       * @throws Exception
42       */
43      @Override
44      public void open(Configuration parameters) throws Exception {
45          //1.设置连接MySQL驱动
46          Class.forName("com.mysql.jdbc.Driver");
47          //2.获取MySQL的连接
48          conn=DriverManager.getConnection(
49              "jdbc:mysql://localhost:3306/likou?userSSL=false",
50              "root",
51              "root"
52          );
53          statement = conn.createStatement();
54      }
55
56      /**
57       * 具体业务逻辑实现的地方
58       * 读取数据表中的数据并将其赋值给user对象
59       * @param sourceContext
60       * @throws Exception
61       */
62      @Override
63      public void run(SourceContext<User> sourceContext) throws Exception {
64          while (isRunning) {
65              //定义SQL查询语句
66              String sql="select id,username,password,name from user";
67              //执行SQL查询
68              ResultSet resultSet = statement.executeQuery(sql);
69              // 将其封装到User中
70              while (resultSet.next()){
71                  int id = resultSet.getInt("id");
72                  String username = resultSet.getString("username");
73                  String password = resultSet.getString("password");
74                  String name = resultSet.getString("name");
75                  sourceContext.collect(new User(
76                      id,username,password,name

```

```

77         ));
78     }
79     Thread.sleep(10*1000L);
80     //每10s循环读取一次MySQL中的数据
81 }
82 }
83 /**
84  * 如果执行被取消，不再生成User对象
85  */
86 @Override
87 public void cancel() {
88     isRunning=false;
89 }
90
91 /**
92  * 生命周期的结束，再次方法中定义收尾的操作
93  * 关闭数据库连接，状态的连接，statement的连接
94  * @throws Exception
95  */
96 @Override
97 public void close() throws Exception {
98     if(!statement.isClosed())statement.close();
99     if(!conn.isClosed())conn.close();
100 }
101 }).printToErr();
102 env.execute();
103
104 }
105
106 @Data
107 @AllArgsConstructor
108 public static class User{
109     private int id;
110     private String username;
111     private String password;
112     private String name;
113 }
114
115 }
116

```

• 合流算子

• union

```
1 package sz.base.flink.transfromation;
2
3 import org.apache.flink.streaming.api.datastream.DataStream;
4 import org.apache.flink.streaming.api.datastream.DataStreamSource;
5 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6
7 /**
8  * 使用union实现
9  * 将以下数据进行取并集操作
10  * 数据集1
11  * "hadoop", "hive", "flume"
12  * 数据集2
13  * "hadoop", "hive", "spark"
14  *
15  * 注意:
16  * 1: 合并后的数据不会自动去重
17  * 2: 要求数据类型必须一致
18  */
19 public class UnionDemo {
20     public static void main(String[] args) throws Exception {
21         /**
22          * 实现步骤:
23          * 1) 初始化flink的流处理的运行环境
24          * 2) 加载/创建数据源
25          * 3) 处理数据
26          * 4) 打印输出
27          * 5) 递交执行作业
28          */
29         StreamExecutionEnvironment env =
30             StreamExecutionEnvironment.getExecutionEnvironment();
31
32         DataStreamSource<String> ds1 = env.fromElements("hadoop", "hive", "flume");
33         DataStreamSource<String> ds2 = env.fromElements("hadoop", "hive", "spark");
34         DataStream<String> result = ds1.union(ds2);
35         result.printToErr();
36     }
37 }
```

```
35
36     env.execute();
37 }
38 }
```

• connector

```
1 package sz.base.flink.transfromation;
2
3 import org.apache.flink.streaming.api.datastream.ConnectedStreams;
4 import org.apache.flink.streaming.api.datastream.DataStreamSource;
5 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
6 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7 import org.apache.flink.streaming.api.functions.co.CoMapFunction;
8
9 /**
10  *
11  */
12 public class ConnectDemo {
13     public static void main(String[] args) throws Exception {
14         StreamExecutionEnvironment env =
15             StreamExecutionEnvironment.getExecutionEnvironment();
16         //生成两个数据流
17         env.setParallelism(1);
18         DataStreamSource<Integer> source = env.fromElements(1, 2, 3, 4, 5, 6, 7);
19         DataStreamSource<String> source1 = env.fromElements("9", "10", "11", "12",
20             "13");
21         ConnectedStreams<Integer, String> connect = source.connect(source1);
22         SingleOutputStreamOperator<String> map = connect.map(new CoMapFunction<Integer,
23             String, String>() {
24             @Override
25             public String map1(Integer integer) throws Exception {
26                 return integer.toString();
27             }
28
29             @Override
30             public String map2(String s) throws Exception {
31                 return s;
32             }
33         });
34         map.printToErr();
35     }
36 }
```



```
32         env.execute();
33     }
34 }
35
```

• 分流

```
1  package sz.base.flink.transfromation;
2
3  import org.apache.flink.api.common.typeinfo.Types;
4  import org.apache.flink.streaming.api.datastream.DataStreamSource;
5  import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
6  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7  import org.apache.flink.streaming.api.functions.ProcessFunction;
8  import org.apache.flink.util.Collector;
9  import org.apache.flink.util.OutputTag;
10
11 /**
12  * 将1~100之间的数据进行一个根据奇数和偶数进行分流操作
13  * 侧输出流：单独的输出管道
14  */
15 public class SplitStreamDemo {
16     public static void main(String[] args) throws Exception {
17         //1.创建流执行环境
18         StreamExecutionEnvironment env =
19             StreamExecutionEnvironment.getExecutionEnvironment();
19         //2.设置并行度
20         env.setParallelism(1);
21         //3.生成数据源1-100序列
22         DataStreamSource<Long> source = env.fromSequence(1, 100);
23         //定义侧输出流保存偶数和奇数
24         OutputTag<Long> odd = new OutputTag<>("odd", Types.LONG);
25         OutputTag<Long> even = new OutputTag<>("even", Types.LONG);
26         //4.1将偶数放到一个侧输出流中
27         //4.2将奇数放到一个侧输出流中
28         SingleOutputStreamOperator<Long> process = source.process(new
29             ProcessFunction<Long, Long>() {
30                 @Override
31                 public void processElement(Long aLong, Context context, Collector<Long>
32                     collector) throws Exception {
33                     if (aLong % 2 == 0) {
```

```

32         context.output(even, aLong);
33     } else {
34         context.output(odd, aLong);
35     }
36 }
37 });
38 //5.打印偶数或技术
39 process.print("主干中的值");
40 process.getSideOutput(odd).print("奇数");
41 process.getSideOutput(even).print("偶数");
42 //6.执行流环境
43 env.execute("拆分数据流");
44
45 }
46 }
47

```

● 物理分区

● 5.重分区

```

1  package sz.base.flink.transfromation;
2
3  import org.apache.flink.api.common.functions.FilterFunction;
4  import org.apache.flink.api.common.functions.RichMapFunction;
5  import org.apache.flink.api.java.tuple.Tuple2;
6  import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
7  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
8
9  /**
10   * 随机生成100个数字，过滤出来90个数字，封装taskid和数字，统计每个taskid
11   * 最终的数字的个数是否都均匀，没有rebalance重分布，每个分区taskid对应的数字个数不均匀（倾斜）
12   * 使用rebalance之后各个数字均匀
13   * 开发步骤：
14   * 0.获取流执行环境并设置并行度为3
15   * 1.过滤出来90和数字[通过rebalance进行均衡]
16   * 2.得到一个转换[taskid,1]
17   * 3.taskid进行分组操作
18   * 4.求和 - 每个cpu index处理的数字个数
19   * 5.打印输出每个cpu index处理的个数

```

```

20  * 6.执行流环境
21  */
22  public class RebalanceDemo {
23      public static void main(String[] args) throws Exception {
24          StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
25          // * 0.获取流执行环境并设置并行度为3
26          //90条数据，分给3个线程去处理，每个线程处理30条数据
27          env.setParallelism(3);
28          // * 1.过滤出来90和数字[通过rebalance进行均衡]
29          SingleOutputStreamOperator<Long> source = env.fromSequence(1, 100).filter(new
FilterFunction<Long>() {
30              @Override
31              public boolean filter(Long aLong) throws Exception {
32                  return aLong > 10;
33              }
34          });
35          // * 2.得到一个转换[taskid,1],,rebalance把数据重分布，平均分给每个cpu
36          source.rebalance().map(new RichMapFunction<Long, Tuple2<Integer,Integer>>() {
37              @Override
38              public Tuple2<Integer, Integer> map(Long aLong) throws Exception {
39                  //获取上下文对象，在获取子任务的id
40                  int taskid = getRuntimeContext().getIndexOfWorkThisSubtask();
41                  return Tuple2.of(taskid,1);
42              }
43          })
44          // * 3.taskid进行分组操作
45          .keyBy(t->t.f0)
46          // * 4.求和 - 每个cpu index处理的数字个数
47          .sum(1)
48          // * 5.打印输出每个cpu index处理的个数
49          .printToErr();
50          // * 6.执行流环境
51          env.execute();
52      }
53
54  }
55

```

7.自定义分区

```
1 package sz.base.flink.transfromation;
2
3 import org.apache.flink.api.common.functions.Partitioner;
4 import org.apache.flink.streaming.api.datastream.DataStream;
5 import org.apache.flink.streaming.api.datastream.DataStreamSource;
6 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7
8 public class CustomPartition {
9     public static void main(String[] args) throws Exception {
10         StreamExecutionEnvironment env =
11             StreamExecutionEnvironment.getExecutionEnvironment();
12         env.setParallelism(3);
13         DataStreamSource<String> source = env.fromElements("flink", "sqark", "123",
14             "flink", "flink", "flink", "flink", "flink", "flink", "hadoop");
15         //自定义分区，将flink单独放到一个分区中，hadoop单独放到另一个分区中，其他放到另一个分
16         区中
17         DataStream<String> dataStream = source.partitionCustom(new MyPartition(), key ->
18             key);
19         dataStream.print();
20         env.execute();
21     }
22     public static class MyPartition implements Partitioner<String>{
23         @Override
24         public int partition(String s, int i) {
25             if(s.equals("flink")){
26                 return 0;
27             }else if(s.equals("hadoop")){
28                 return 1;
29             }else {
30                 return 2;
31             }
32         }
33     }
34 }
```

DataSink数据输出

• 2.file和csv -- 方法弃用

```
1 package sz.base.flink.sink;
2
3 import org.apache.flink.core.fs.FileSystem;
4 import org.apache.flink.streaming.api.datastream.DataStreamSource;
5 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6
7
8 /**
9  * 写出数据到文件系统
10  */
11 public class SinkFileDemo {
12     public static void main(String[] args) throws Exception {
13         //1.创建流执行环境
14         StreamExecutionEnvironment env =
15             StreamExecutionEnvironment.getExecutionEnvironment();
16         //2.设置并行度等参数
17         env.setParallelism(1);
18         //3.生成数据源
19         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
20         //4.将数据写入到文件系统中,重复写入,并行写入文件形成一个文件夹放入数据
21         node1.writeAsText(
22             "data/output",
23             FileSystem.WriteMode.OVERWRITE
24         ).setParallelism(2);
25         //写出格式为csv, 指定行分隔符和列分隔符
26         /* node1.writeAsCsv(
27             "data/output",
28             FileSystem.WriteMode.OVERWRITE,
29             "\n", "\001"
30         );*/
31         env.execute();
32     }
33 }
```

connector连接器

MySQL写出

```
1 package sz.base.flink.connector;
2
3 import org.apache.flink.api.common.typeinfo.Types;
4 import org.apache.flink.api.java.tuple.Tuple2;
5 import org.apache.flink.connector.jdbc.*;
6 import org.apache.flink.streaming.api.datastream.DataStreamSource;
7 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
8 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
9 import org.apache.flink.util.Collector;
10
11 import java.sql.PreparedStatement;
12 import java.sql.SQLException;
13
14
15 /**
16  * 通过connector连接器，将单词统计的值保存到MySQL数据表中
17  * 开发步骤：
18  * 1. 导入包connector连接到jdbc的jar包
19  * 2. 创建数据库，并创建表
20  * 3. 创建流执行环境
21  * 4. 获取socket数据源
22  * 5. 将获取到文本映射成【word, 1】
23  * 6. keyBy分组和sum统计
24  * 7. 将结果.addSink（使用connector连接器）
25  * 8. 执行流环境
26  * 9. 查看结果
27  */
28 public class SinkMySQLDemo {
29     public static void main(String[] args) throws Exception {
30         // * 3. 创建流执行环境
31         StreamExecutionEnvironment env =
32             StreamExecutionEnvironment.getExecutionEnvironment();
33         // 设置并行度
34         env.setParallelism(1);
```

```

34      // * 4.获取socket数据源
35      DataSourceSource<String> node1 = env.socketTextStream("node1", 9999);
36      // * 2.创建数据库, 并创建表
37      // * 1.导入包connector连接到jdbc的jar包
38      SingleOutputStreamOperator<Tuple2<String, Integer>> sum = node1.flatMap((String
word, Collector<Tuple2<String, Integer>> out) -> {
39          String[] words = word.split(" ");
40          for (String s : words) {
41              // * 5.将获取到文本映射成【word, 1】
42              out.collect(Tuple2.of(s, 1));
43          }
44          // * 6.keyBy分组和sum统计
45      }).returns(Types.TUPLE(Types.STRING, Types.INT)).keyBy(t -> t.f0).sum(1);
46      // * 7.将结果.addSink (使用connector连接器)
47      sum.addSink(JdbcSink.sink(
48          "insert into t_wordcount(word,counts) values (?,?) on duplicate key
update counts=?",
49          new JdbcStatementBuilder<Tuple2<String, Integer>>() {
50              @Override
51              public void accept(PreparedStatement ps, Tuple2<String, Integer>
str) throws SQLException {
52                  ps.setString(1, str.f0);
53                  ps.setInt(2, str.f1);
54                  ps.setInt(3, str.f1);
55              }
56          },
57          new JdbcExecutionOptions.Builder()
58              //5s自动将数据插入到数据表中
59              .withBatchIntervalMs(5000)
60              //每个批次插入的数量
61              .withBatchSize(1)
62              //最大重试的次数
63              .withMaxRetries(3)
64              .build(),
65          new JdbcConnectionOptions.JdbcConnectionOptionsBuilder()
66              .withDriverName("com.mysql.jdbc.Driver")
67              .withUrl("jdbc:mysql://localhost:3306/likou?useSSL=false")
68              .withUsername("root")
69              .withPassword("root")
70              .build()
71      ));

```

```
72         // * 8.执行流环境
73         env.execute();
74         // * 9.查看结果
75     }
76 }
77
```

kafka

消费数据

```
1  package sz.base.flink.connector;
2
3  import org.apache.flink.api.common.serialization.SimpleStringSchema;
4  import org.apache.flink.streaming.api.datastream.DataStreamSource;
5  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6  import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
7  import org.apache.kafka.clients.consumer.ConsumerConfig;
8
9  import java.util.Properties;
10
11  import static
    org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumerBase.KEY_PARTITION_DISCOVER
    Y_INTERVAL_MILLIS;
12
13  /**
14   * 读取kafka集群中的数据通过connector
15   * 开发步骤:
16   * 1.获取流执行环境
17   * 2.设置并行度
18   * 3.Flink消费kafka的数据参数
19   * 3.1配置参数
20   * 3.2实例化FlinkKafkaConsumer 对象
21   * 4.设置消费kafka的重启位置，如果当前程序崩溃了，从哪里接着消费
22   * 5.将offset提交给flink的checkpoint来管理
23   * 6.将生成的FlinkKafkaConsumer对象添加到source中
24   * 7.打印输出数据
25   * 8.执行流环境
26   */
27  public class FlinkKafkaReader {
```



```
28     public static void main(String[] args) throws Exception {
29         // * 1.获取流执行环境
30         StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
31         // * 2.设置并行度
32         env.setParallelism(1);
33         //开启checkpoint检查点
34         env.enableCheckpointing(1000);
35         // * 3.Flink消费kafka的数据参数
36         // * 3.1配置参数
37         Properties properties = new Properties();
38         //kafka主机和端口(固定的)
39         properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
"node1:9092,node2:9092,node3:9092");
40         //kafka groupId
41         properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, "__consumer_src_");
42         //kafka autocommit 将offset是否自动提交到kafka consumer 中保存
43         properties.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
44         //自动分区发现
45         properties.setProperty(KEY_PARTITION_DISCOVERY_INTERVAL_MILLIS, 6 * 1000 + "");
46         // * 3.2实例化FlinkKafkaConsumer 对象
47         FlinkKafkaConsumer<String> srctopic = new FlinkKafkaConsumer<>(
48             "srctopic", new SimpleStringSchema(),
49             properties
50         );
51         // * 4.设置消费kafka的重启位置, 如果当前程序崩溃了, 从哪里接着消费
52         //从头
53         srctopic.setStartFromEarliest();
54         // * 5.将offset提交给flink的checkpoint来管理
55         srctopic.setCommitOffsetsOnCheckpoints(true);
56         // * 6.将生成的FlinkKafkaConsumer对象添加到source中
57         DataStreamSource<String> source = env.addSource(srctopic);
58         // * 7.打印输出数据
59         source.printToErr();
60         // * 8.执行流环境
61         env.execute();
62     }
63 }
64
```

生产数据到kafka

```
1 package sz.base.flink.connector;
2
3 import org.apache.flink.api.common.serialization.SimpleStringSchema;
4 import org.apache.flink.streaming.api.datastream.DataStreamSource;
5 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
7 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;
8 import org.apache.kafka.clients.consumer.ConsumerConfig;
9 import org.apache.kafka.clients.producer.ProducerConfig;
10
11 import java.util.Properties;
12
13 import static
    org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumerBase.KEY_PARTITION_DISCOVERY_INTERVAL_MILLIS;
14
15 /**
16  * Flink向kafka中写入数据
17  * 开发步骤:
18  * 1. 获取流执行环境
19  * 2. 设置并行度
20  * 3. Flink消费kafka的数据参数
21  * 3.1 配置参数
22  * 3.2 实例化FlinkKafkaConsumer 对象
23  * 4. 设置消费kafka的重启位置, 如果当前程序崩溃了, 从哪里接着消费
24  * 5. 将offset提交给flink的checkpoint来管理
25  * 6. 将生成的FlinkKafkaConsumer对象添加到source中
26  * 7. 打印输出数据
27  * 8. 执行流环境
28  */
29 public class FlinkKafkaWriter {
30     public static void main(String[] args) throws Exception {
31         // * 1. 获取流执行环境
32         StreamExecutionEnvironment env =
            StreamExecutionEnvironment.getExecutionEnvironment();
33         // * 2. 设置并行度
34         env.setParallelism(1);
```

```

35      //开启checkpoint检查点
36      env.enableCheckpointing(1000);
37      //从socket 数据源生成数据
38      //将数据直接写入到kafka
39      DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
40      // * 3.Flink消费kafka的数据参数
41      // * 3.1配置参数
42      Properties properties = new Properties();
43      //kafka主机和端口(固定的)
44      properties.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
45      "node1:9092,node2:9092,node3:9092");
46      // * 3.2实例化FlinkKafkaConsumer 对象
47      FlinkKafkaProducer<String> srctopic = new FlinkKafkaProducer<>(
48          "srctopic", new SimpleStringSchema(),properties
49          );
50      node1.addSink(srctopic);
51      // * 7.打印输出数据
52      node1.printToErr();
53      // * 8.执行流环境
54      env.execute();
55  }
56

```

window窗口

滑动窗口和滚动窗口

```

1  package sz.base.flink.window;
2
3  import org.apache.flink.api.java.tuple.Tuple2;
4  import org.apache.flink.streaming.api.datastream.DataStreamSource;
5  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6  import org.apache.flink.streaming.api.functions.source.SourceFunction;
7  import org.apache.flink.streaming.api.windowing.assigners.SlidingEventTimeWindows;
8  import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows;
9  import org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows;
10 import org.apache.flink.streaming.api.windowing.time.Time;
11

```

```

12 import java.util.ArrayList;
13 import java.util.Arrays;
14 import java.util.List;
15 import java.util.Random;
16
17 /**
18  * 统计全窗口和分流窗口，窗口大小是5s，计算单词的对应的数字之和
19  * 输入的数据[apache,10][hadoop,2][flink,3][spark,5][hadoop,2]
20  * 输出的数据
21  * 窗口大小: 5s window
22  * 输出数据 keyBy,聚合: [hadoop,10][apache,20][spark,25]...
23  * 全窗口,不分流直接求和[hadoop,55]
24  * 开发步骤:
25  * 1.获取流执行环境
26  * 2.获取随机数据源 GenerateRandomNumEverySecond
27  * 3.全窗口滚动处理时间窗口为5s，统计sum求和
28  * 4.根据单词进行分区，滚动处理时间窗口为5s，统计sum求和
29  * 5.打印输出
30  * 6.执行流环境
31  * <p>
32  * window api 格式
33  * 数据流.keyBy(分组字段) -- 分流操作
34  * .window(窗口的类型) --时间窗口（滚动时间、滑动时间、会话时间）计数窗口
35  * .windowAll(窗口的类型) -- 窗口 没有用keyBy用windowAll,用了keyBy用window
36  * .trigger(触发的时间) -- 触发,默认触发方式
37  * .allowedLateness(允许延迟的时间) --3min 在3min中之内来的数据依然会被计算
38  * .sideOutputLateData(侧输出流) output tag -->超过3min, 将这些数据流保存的位置 Side
    OutputTag 可以取出来
39  * .聚合函数() reduce / aggregate / fold / apply() / process()全量主要 -- 聚合函数 ①增量
    (来一条处理一条) ②全量 (窗口内数据聚合)
40 */
41 public class TimeWindowDemo01_1 {
42     public static void main(String[] args) throws Exception {
43         //1.获取流执行环境
44         StreamExecutionEnvironment env =
            StreamExecutionEnvironment.getExecutionEnvironment();
45         //2.设置并行度
46         env.setParallelism(1);
47         //3.获取数据源
48         DataStreamSource<Tuple2<String, Integer>> source = env.addSource(new
            GenerateRandomNumEverySecond());

```

```

49         //4.窗口的划分和聚合计算
50         //4.1windowAll 全窗口, 5s 内所有的元素聚合, 不分类keyBy
51         /*source.windowAll(TumblingProcessingTimeWindows.of(Time.seconds(5))) //会统计每
52         5s的value累加, 不管key是什么会输出第一个进来的key值
53
54         .sum(1).printToErr();*/
55         //4.2keyBy ... window 先分组再开窗
56         source.keyBy(k->k.f0)
57
58         //5.对开窗的数据求和,窗口分类: ①时间窗口: 滚动时间、滑动时间、会话时间 ② 技术窗口: 滚
59         动计数、滑动计数窗口
60
61         //窗口API: 窗口名称 + 时间
62
63         /*.window(TumblingProcessingTimeWindows.of(Time.seconds(5))).sum(1).printToErr();*/
64         //偏移量,用来偏移(Time.days(1), Time.hours(-8))时间, 一般是中国时间如左例
65
66         //
67         .window(TumblingProcessingTimeWindows.of(Time.seconds(5),Time.seconds(2))).sum(1).printT
68         oErr();
69
70         //计算,每2s (滑动时间) 计算5s (窗口)
71
72         .window(SlidingProcessingTimeWindows.of(Time.seconds(5),Time.milliseconds(2100))).sum(1)
73         .printToErr();
74
75         //偏移量,用来偏移Time.hours(12), Time.hours(1), Time.hours(-8))时间, 一般是中国时间
76         如左例
77
78         //.window(SlidingProcessingTimeWindows.of(Time.seconds(5),Time.seconds(2)),Time.seconds(
79         1)).sum(1).printToErr();
80
81         //6.打印输出结果
82
83         //7.执行流环境
84
85         env.execute();
86     }
87
88     /**
89      * 实现一个SourceFunction, 每一秒好创建一个Tuple2
90      */
91
92     public static class GenerateRandomNumEverySecond implements
93     SourceFunction<Tuple2<String, Integer>> {
94
95         boolean isRunning = true;
96
97         //随机数
98
99         final Random rm = new Random();
100
101         //定义一个数组/集合
102
103         List<String> keys = Arrays.asList("hadoop", "spark", "flink", "hadoop", "hive");
104
105
106         /**
107          * 核心业务逻辑, 每秒生成二元元组[hadoop/spark,随机数字]

```

```

81      *
82      * @param sourceContext
83      * @throws Exception
84      */
85      @Override
86      public void run(SourceContext<Tuple2<String, Integer>> sourceContext) throws
Exception {
87
88          while (isRunning) {
89              //获取列表中的随机的key值
90              String key = keys.get(rm.nextInt(keys.size()));
91              //获取一个value= [0 ~ 50]之间的值
92              int value = rm.nextInt(10);
93              //返回Tuple2
94              Tuple2<String, Integer> of = Tuple2.of("spark", value);
95              sourceContext.collect(of);
96              System.out.println(of);
97              //要求每1s打印一条数据
98              Thread.sleep(1000);
99
100         }
101     }
102
103     @Override
104     public void cancel() {
105         isRunning = false;
106     }
107 }
108 }
109

```

会话窗口

```

1  package sz.base.flink.window;
2
3  import org.apache.flink.api.java.tuple.Tuple2;
4  import org.apache.flink.streaming.api.datastream.DataStreamSource;
5  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6  import org.apache.flink.streaming.api.functions.source.SourceFunction;

```

```

7 import org.apache.flink.streaming.api.windowing.assigners.ProcessingTimeSessionWindows;
8 import org.apache.flink.streaming.api.windowing.time.Time;
9
10 import java.text.SimpleDateFormat;
11 import java.util.Arrays;
12 import java.util.Date;
13 import java.util.List;
14 import java.util.Random;
15
16 /**
17  * 会话窗口
18  * 会话窗口 -
19  * 需求1: 定义一个会话时间窗口, 5sgap, 统计全量windowall (Non-key) 数据之和
20  * 需求2: 定义一个会话时间窗口, 5s gap , 统计按照key分组后的每个组数据内的数字和
21  */
22 public class SessionWindowDemo02_2 {
23     public static void main(String[] args) throws Exception {
24         //获取流执行环境
25         StreamExecutionEnvironment env =
26             StreamExecutionEnvironment.getExecutionEnvironment();
27         //设置并行度
28         env.setParallelism(1);
29         //添加自定义数据源
30         DataStreamSource<Tuple2<String, Integer>> source = env.addSource(new
31             GenerateRandomNumRandomSecond());
32         //1.全窗口windowAll, non-keyed window 设置处理时间session窗口, 间隔5s求和
33         /*source.windowAll(ProcessingTimeSessionWindows.withGap(Time.seconds(5))).sum(1).printTo
34             Err();*/
35         //2.根据单词设置处理时间session窗口, 间隔5s, 求和
36         source.keyBy(k-
37             >k.f0).window(ProcessingTimeSessionWindows.withGap(Time.seconds(5))).sum(1).printToErr()
38             ;
39         //执行
40         env.execute();
41     }
42
43     /**
44      * 自定义Source
45      * 每隔随机时间(1~7秒之间)产生一个的k,v k是hadoop spark flink 其中某一个, v是随机数字
46      */
47 }

```

```

42     public static class GenerateRandomNumRandomSecond implements
SourceFunction<Tuple2<String, Integer>> {
43         private SimpleDateFormat sdf =new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
44         private boolean isRun = true;
45         private final Random random = new Random();
46         private final List<String> keyList = Arrays.asList("hadoop", "spark", "flink");
47         @Override
48         public void run(SourceContext<Tuple2<String, Integer>> ctx) throws Exception {
49             while (this.isRun) {
50                 String key = keyList.get(random.nextInt(3));
51                 Tuple2<String, Integer> value = Tuple2.of(key, random.nextInt(9));
52                 ctx.collect(value);
53                 long sleepTime = 5000L;
54                 while (sleepTime == 5000L) {
55                     sleepTime = random.nextInt(7) * 1000L;
56                 }
57                 System.out.println(sdf.format(new Date()) + " :---will sleep " +
sleepTime + " ms---: " + value);
58                 Thread.sleep(sleepTime);
59             }
60         }
61
62         @Override
63         public void cancel() {
64             this.isRun = false;
65         }
66     }
67 }
68

```

计数窗口

```

1  package sz.base.flink.window;
2
3  import org.apache.flink.api.java.tuple.Tuple2;
4  import org.apache.flink.streaming.api.datastream.DataStreamSource;
5  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
6  import org.apache.flink.streaming.api.functions.source.SourceFunction;
7
8  import java.text.SimpleDateFormat;

```



```

9  import java.util.Arrays;
10 import java.util.Date;
11 import java.util.List;
12 import java.util.Random;
13
14 /**
15  * 随机生成数据[hadoop/flink,随机生成数字20/30], 使用计数窗口进行计算
16  * 需求1: 通过滚动计数 window all non-keyed
17  * 需求2: 通过滑动计数 keyBy window
18  */
19 public class CountWindowDemo03_3 {
20     public static void main(String[] args) throws Exception {
21         //计算每5个计算non-keyed 窗口内的数据
22         //1.创建流环境
23         StreamExecutionEnvironment env =
24 StreamExecutionEnvironment.getExecutionEnvironment();
25         //设置参数
26         env.setParallelism(1);
27         //获取数据源
28         DataStreamSource<Tuple2<String, Integer>> source = env.addSource(new
29 GenerateRandomNumRandomSecond());
30         //窗口计算
31         /*source.countWindowAll(5).sum(1).printToErr();*/
32         //先分组在count window
33         /*source.keyBy(t->t.f0).countWindow(5).sum(1).printToErr();*/
34         //先分组在滑动 count window 每2条计算前5条 如果分组则统计同一组内
35         source.keyBy(t->t.f0).countWindow(5,2).sum(1).printToErr();
36         env.execute();
37     }
38     /**
39     自定义Source
40     每隔随机时间(1~7秒之间)产生一个的k,v k是hadoop spark flink 其中某一个, v是随机数字
41     */
42     public static class GenerateRandomNumRandomSecond implements
43 SourceFunction<Tuple2<String, Integer>> {
44         private SimpleDateFormat sdf =new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
45         private boolean isRun = true;
46         private final Random random = new Random();
47         private final List<String> keyList = Arrays.asList("hadoop", "spark", "flink");
48         @Override
49         public void run(SourceContext<Tuple2<String, Integer>> ctx) throws Exception {

```

```

47         while (this.isRun) {
48             String key = keyList.get(random.nextInt(3));
49             Tuple2<String, Integer> value = Tuple2.of(key, random.nextInt(9));
50             ctx.collect(value);
51             System.out.println("-----: " + value);
52             Thread.sleep(1000);
53         }
54     }
55
56     @Override
57     public void cancel() {
58         this.isRun = false;
59     }
60 }
61 }
62

```

计数窗口2

```

1  package sz.base.flink.window;
2
3  import org.apache.flink.api.java.tuple.Tuple3;
4  import org.apache.flink.streaming.api.datastream.DataStreamSource;
5  import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
6  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
7  import org.apache.flink.streaming.api.functions.windowing.ProcessWindowFunction;
8  import org.apache.flink.streaming.api.windowing.windows.GlobalWindow;
9  import org.apache.flink.util.Collector;
10
11 import java.util.Arrays;
12
13 /**
14  * 每5位同学统计一下这五位同学的平均成绩
15  * 输入的数据["class1","姓名",100L]
16  * 输出的数据： 每5个同学计算出平均分
17  * .聚合函数() reduce / aggregate / fold / apply() / process()全量主要 -- 聚合函数 ①增量
   (来一条处理一条) ②全量 (窗口内数据聚合)
18  */
19 public class WindowCountProcessDemo_4 {

```

```

20     public static void main(String[] args) throws Exception {
21         //获取流环境
22         StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
23         //设置并行度
24         env.setParallelism(2);
25         //获取数据源
26         DataStreamSource<Tuple3<String, String, Long>> source =
env.fromCollection(Arrays.asList(ENGLISH));
27         //分班开窗，每5个同学分到一个组内，全量的计算，计算平均分
28         SingleOutputStreamOperator<Double> process = source.shuffle().keyBy(k -> k.f0)
29             .countWindow(5)
30             //全窗口计算 方法传入的参数
31             //T: 输入流的类型
32             // R: 输出的类型
33             //K : 分组的字段类型
34             //W: 开窗的类型 ① 时间窗口类型 ② 计数窗口类型
35             .process(new ProcessWindowFunction<Tuple3<String, String, Long>, Double,
String, GlobalWindow>() {
36                 /**
37                  * 核心处理逻辑，5个同学的平均分
38                  * @param s 分组的字段，班级
39                  * @param context 上下文对象
40                  * @param iterable 就是这个计数窗口内的所有数据
41                  * @param collector
42                  * @throws Exception
43                  */
44                 @Override
45                 public void process(String s, Context context,
Iterable<Tuple3<String, String, Long>> iterable, Collector<Double> collector) throws
Exception {
46                     //定义一个变量用于接收所有同学的成绩
47                     Long sum = 0L;
48                     for (Tuple3<String, String, Long> stu : iterable) {
49                         sum += stu.f2;
50                     }
51                     //计算平均值
52                     double result = sum / 5.00D;
53                     //收集数据
54                     collector.collect(result);
55                 }

```

```

56         });
57         //得到窗口的结果
58         //打印输出结果
59         process.print();
60         //执行流环境
61         env.execute();
62     }
63
64     public static final Tuple3<String, String, Long>[] ENGLISH = new Tuple3[]{
65         Tuple3.of("class1", "张三", 100L),
66         Tuple3.of("class2", "小七", 59L),
67         Tuple3.of("class1", "李四", 78L),
68         Tuple3.of("class1", "小七", 59L),
69         Tuple3.of("class2", "李四", 78L),
70         Tuple3.of("class2", "王五", 99L),
71         Tuple3.of("class1", "王五", 99L),
72         Tuple3.of("class1", "赵六", 81L),
73         Tuple3.of("class2", "赵六", 81L),
74         Tuple3.of("class2", "张三", 100L),
75     };
76 }
77

```

水印机制

水印机制

```

1  package sz.base.flink.watermark;
2
3  import org.apache.flink.api.common.eventtime.WatermarkStrategy;
4  import org.apache.flink.api.common.functions.FlatMapFunction;
5  import org.apache.flink.api.common.functions.MapFunction;
6  import org.apache.flink.api.common.serialization.SimpleStringSchema;
7  import org.apache.flink.api.java.functions.KeySelector;
8  import org.apache.flink.api.java.tuple.Tuple2;
9  import org.apache.flink.streaming.api.datastream.DataStreamSource;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
12 import org.apache.flink.util.Collector;

```

```

13 import org.apache.kafka.clients.consumer.ConsumerConfig;
14
15 import java.time.Duration;
16 import java.util.Properties;
17
18 /**
19  * 实现数据源上的水印机制，source端设置水印机制
20  * 开发步骤：
21  * 1.获取流环境
22  * 2.读取kafka source ， 初始化了FlinkKafkaConsumer
23  * 3.source 设置水印机制
24  * 4.添加数据源
25  * 5.wordcount
26  * 6.打印输出
27  * 7.执行流环境
28  */
29 public class KafkaSourceWatemarkDemo_5 {
30     public static void main(String[] args) throws Exception {
31         //1.获取流环境
32         StreamExecutionEnvironment env =
33 StreamExecutionEnvironment.getExecutionEnvironment();
34         //设置参数
35         env.setParallelism(1);
36         env.enableCheckpointing(2000);
37         //2.读取kafka
38         Properties properties = new Properties();
39         properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "node1:9092,node2:9092,node3:9092");
40         properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, "__consumer_src_");
41         FlinkKafkaConsumer<String> srctopic = new FlinkKafkaConsumer<>(
42             "srctopic",
43             new SimpleStringSchema(),
44             properties
45         );
46         //分配提交checkpoint 维护offset
47         srctopic.setCommitOffsetsOnCheckpoints(true);
48         //如果失败了，从哪里继续来读取,从组内
49         srctopic.setStartFromGroupOffsets();
50         //3.设置水印
51         //作用就是，在source端就将乱序的数据排个序

```

```

51     srctopic.assignTimestampsAndWatermarks(
52         //选择乱序时间，等待30s
53         WatermarkStrategy.forBoundedOutOfOrderness(Duration.ofSeconds(30))
54     );
55     //4.添加数据源
56     DataStreamSource<String> source = env.addSource(srctopic);
57     //
58     source.flatMap(new FlatMapFunction<String, String>() {
59         @Override
60         public void flatMap(String s, Collector<String> collector) throws Exception
61     {
62         String[] words = s.split(" ");
63         for (String word : words) {
64             collector.collect(word);
65         }
66     }).map(new MapFunction<String, Tuple2<String, Integer>>() {
67         @Override
68         public Tuple2<String, Integer> map(String s) throws Exception {
69             return Tuple2.of(s, 1);
70         }
71     }).keyBy(new KeySelector<Tuple2<String, Integer>, String>() {
72         @Override
73         public String getKey(Tuple2<String, Integer> stringIntegerTuple2) throws
74         Exception {
75             return stringIntegerTuple2.f0;
76         }
77     }).sum(1).print();
78     //执行流环境
79     env.execute();
80 }
81

```

assignTimestampsAndWatermarks

forBoundedOutOfOrderness

- 分配水印机制，单调乱序的水印，会有最大的延迟时间 (最新用法)，乱序实现

```
1 package sz.base.flink.watermark;
```

```
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import org.apache.commons.collections.IteratorUtils;
6 import org.apache.flink.api.common.eventtime.WatermarkStrategy;
7 import org.apache.flink.api.common.functions.MapFunction;
8 import org.apache.flink.streaming.api.datastream.DataStreamSource;
9 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11 import org.apache.flink.streaming.api.functions.windowing.ProcessWindowFunction;
12 import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
13 import org.apache.flink.streaming.api.windowing.time.Time;
14 import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
15 import org.apache.flink.util.Collector;
16
17 import java.time.Duration;
18
19 /**
20  * 在单调递增的水印，在非source上添加水印
21  * 1.定义类 WaterSensor String id; Long ts; Integer vc;
22  * 2.创建流执行环境
23  * 3.获取socket文本数据
24  * 4.将字符串数据切分成 WaterSensor 对象数据
25  * 5.分配水印机制，单调递增
26  * 6.分配后的数据根据id进行分组
27  * 7.设置滚动事件时间窗口，时间为10秒
28  * 8.对开窗数据进行process
29  */
30 public class WaterSensorDemo_6 {
31     public static void main(String[] args) throws Exception {
32         //1.定义类 WaterSensor String id; Long ts; Integer vc;
33         //2.创建流执行环境
34         StreamExecutionEnvironment env =
35             StreamExecutionEnvironment.getExecutionEnvironment();
36         env.setParallelism(1);
37         //3.获取socket文本数据
38         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
39         //4.将字符串数据切分成 WaterSensor 对象数据
40         SingleOutputStreamOperator<WaterSensor> operator = node1.map(new
41             MapFunction<String, WaterSensor>() {
```

```

40         @Override
41         public WaterSensor map(String s) throws Exception {
42             String[] split = s.split(",");
43             return new WaterSensor(
44                 split[0],
45                 Long.parseLong(split[1]),
46                 Integer.parseInt(split[2])
47             );
48         }
49     });
50     //5.分配水印机制,单调递增,默认为0s 分配水印机制,单调乱序的水印,会有最大的延迟时间
    (最新用法)
51     SingleOutputStreamOperator<WaterSensor> waterSensorSingleOutputStreamOperator =
    operator.assignTimestampsAndWatermarks(WatermarkStrategy.
    <WaterSensor>forBoundedOutOfOrderness(Duration.ofSeconds(3)).withTimestampAssigner((elem
    ent, recordTimestamp)->element.getTs()*1000));
52     //6.分配后的数据根据id进行分组
53     SingleOutputStreamOperator<String> process =
    waterSensorSingleOutputStreamOperator.keyBy(waterSensor -> waterSensor.id)
54     //7.设置滚动事件时间窗口,时间为10秒
55     .window(TumblingEventTimeWindows.of(Time.seconds(10)))
56     //8.对开窗数据进行process
57     .process(new ProcessWindowFunction<WaterSensor, String, String,
    TimeWindow>() {
58         @Override
59         public void process(String id, Context context,
    Iterable<WaterSensor> iterable, Collector<String> collector) throws Exception {
60             String str = "id=" + id + "\n" + "数据应为: " + iterable +
    "\n" + "数据条数: " + IteratorUtils.toList(iterable.iterator()).size() + "\n" + "窗口开
    始时间: "
61             + context.window().getStart() + ",窗口结束时间: " +
    context.window().getEnd() + "\n" + "-----";
62             collector.collect(str);
63         }
64     });
65     //9.打印输出
66     process.print();
67     //10.执行流环境
68     env.execute();
69 }
70
71 @Data
72 @AllArgsConstructor

```



```

73     public static class WaterSensor{
74         private String id;
75         private Long ts;
76         private Integer vc;
77
78     }
79 }
80

```

for Monotonous Timestamps

- 单调递增,默认为0s

```

1  package sz.base.flink.watermark;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import org.apache.commons.collections.IteratorUtils;
6  import org.apache.flink.api.common.eventtime.WatermarkStrategy;
7  import org.apache.flink.api.common.functions.MapFunction;
8  import org.apache.flink.streaming.api.datastream.DataStreamSource;
9  import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11 import org.apache.flink.streaming.api.functions.windowing.ProcessWindowFunction;
12 import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
13 import org.apache.flink.streaming.api.windowing.time.Time;
14 import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
15 import org.apache.flink.util.Collector;
16
17 import java.time.Duration;
18
19 public class WaterSensorOutOfOrdemessDemo_7 {
20     public static void main(String[] args) throws Exception {
21         //1.定义类 WaterSensor  String id; Long ts; Integer vc;
22         //2.创建流执行环境
23         StreamExecutionEnvironment env =
24             StreamExecutionEnvironment.getExecutionEnvironment();
25         env.setParallelism(1);
26         //3.获取socket文本数据
27         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);

```

```

27         //4.将字符串数据切分成 WaterSensor 对象数据
28         SingleOutputStreamOperator<WaterSensor> operator = node1.map(new
MapFunction<String, WaterSensor>() {
29             @Override
30             public WaterSensor map(String s) throws Exception {
31                 String[] split = s.split(",");
32                 return new WaterSensor(
33                     split[0],
34                     Long.parseLong(split[1]),
35                     Integer.parseInt(split[2])
36                 );
37             }
38         });
39         //5.分配水印机制，单调递增，默认为0s
40         SingleOutputStreamOperator<WaterSensor> waterSensorSingleOutputStreamOperator =
operator.assignTimestampsAndWatermarks(WatermarkStrategy.
<WaterSensor>forMonotonousTimestamps().withTimestampAssigner((element, recordTimestamp)
-> element.getTs() * 1000));
41         //6.分配后的数据根据id进行分组
42         SingleOutputStreamOperator<String> process =
waterSensorSingleOutputStreamOperator.keyBy(waterSensor -> waterSensor.id)
43         //7.设置滚动事件时间窗口，时间为10秒
44         .window(TumblingEventTimeWindows.of(Time.seconds(10)))
45         //8.对开窗数据进行process
46         .process(new ProcessWindowFunction<WaterSensor, String, String,
TimeWindow>() {
47             @Override
48             public void process(String id, Context context,
Iterable<WaterSensor> iterable, Collector<String> collector) throws Exception {
49                 String str = "id=" + id + "\n" + "数据应为: " + iterable +
"\n" + "数据条数: " + IteratorUtils.toList(iterable.iterator()).size() + "\n" + "窗口开
始时间: "
50                 + context.window().getStart() + ",窗口结束时间: " +
context.window().getEnd() + "\n" + "-----";
51                 collector.collect(str);
52             }
53         });
54         //9.打印输出
55         process.print();
56         //10.执行流环境
57         env.execute();
58     }
59

```

```

60     @Data
61     @AllArgsConstructor
62     public static class WaterSensor {
63         private String id;
64         private Long ts;
65         private Integer vc;
66
67     }
68 }
69
70
71

```

允许迟到时间allowedLateness

```

1  package sz.base.flink.allowlateness;
2
3  import org.apache.commons.collections.IteratorUtils;
4  import org.apache.flink.api.common.eventtime.WatermarkStrategy;
5  import org.apache.flink.api.common.functions.MapFunction;
6  import org.apache.flink.api.java.tuple.Tuple2;
7  import org.apache.flink.streaming.api.datastream.DataStreamSource;
8  import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
9  import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
10 import org.apache.flink.streaming.api.functions.windowing.ProcessWindowFunction;
11 import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
12 import org.apache.flink.streaming.api.windowing.time.Time;
13 import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
14 import org.apache.flink.util.Collector;
15
16 import java.time.Duration;
17
18 /**
19  * 需求- 根据socket输入的个数
20  * 来统计一下在指定时间窗口内一共有多少个元素
21  * 输入的格式: hello,1
22  * 输出的格式: 根据时间窗口, 得到一个个数 10 , 窗口的开始时间和结束时间
23  */
24 public class AllowLatenssDemo_8 {

```

```

25     public static void main(String[] args) throws Exception {
26         //获取流执行环境
27         StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
28         env.setParallelism(1);
29         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
30         SingleOutputStreamOperator<Tuple2<String, Integer>> operator = node1.map(new
MapFunction<String, Tuple2<String, Integer>>() {
31             @Override
32             public Tuple2<String, Integer> map(String s) throws Exception {
33                 String[] lines = s.split(",");
34                 return Tuple2.of(lines[0], Integer.parseInt(lines[1]));
35             }
36         });
37         //分配水位线，最大延迟3s
38         SingleOutputStreamOperator<String> process =
operator.assignTimestampsAndWatermarks(WatermarkStrategy.<Tuple2<String,
Integer>>forBoundedOutOfOrderness(Duration.ofSeconds(3)).withTimestampAssigner((element,
recordTimestamp) -> element.f1 * 1000))
39             .keyBy(t -> t.f0)
40             .window(TumblingEventTimeWindows.of(Time.seconds(5)))
41             //允许最大严重乱序时间为2s,触发计算窗口之后的延迟
42             .allowedLateness(Time.seconds(2))
43             //process处理，对窗口数据中的元素进行统计，生成[单词,出现次数]，并将窗口开始时间
和结束时间打印到控制台
44             .process(new ProcessWindowFunction<Tuple2<String, Integer>, String,
String, TimeWindow>() {
45                 @Override
46                 public void process(String s, Context context,
Iterable<Tuple2<String, Integer>> iterable, Collector<String> collector) throws
Exception {
47                     int size = IteratorUtils.toList(iterable.iterator()).size();
48                     collector.collect(s + ":" + size);
49                     System.out.println("当前窗口的start: " +
context.window().getStart() + " 窗口结束时间: " + context.window().getEnd());
50                 }
51
52             });
53         process.print();
54         env.execute();
55     }
56 }
57

```

侧输出流sideallowedLateness

```
1 package sz.base.flink.allowlateness;
2
3 import org.apache.commons.collections.IteratorUtils;
4 import org.apache.flink.api.common.eventtime.WatermarkStrategy;
5 import org.apache.flink.api.common.functions.MapFunction;
6 import org.apache.flink.api.common.typeinfo.Types;
7 import org.apache.flink.api.java.tuple.Tuple2;
8 import org.apache.flink.streaming.api.datastream.DataStreamSource;
9 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11 import org.apache.flink.streaming.api.functions.windowing.ProcessWindowFunction;
12 import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
13 import org.apache.flink.streaming.api.windowing.time.Time;
14 import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
15 import org.apache.flink.util.Collector;
16 import org.apache.flink.util.OutputTag;
17
18 import java.time.Duration;
19
20 /**
21  * 需求- 根据socket输入的个数
22  * 来统计一下在指定时间窗口内一共有多少个元素
23  * 输入的格式: hello,1
24  * 输出的格式: 根据时间窗口, 得到一个个数 10 , 窗口的开始时间和结束时间
25  */
26 public class AllowLatenssSideDemo_9 {
27     public static void main(String[] args) throws Exception {
28         //获取流执行环境
29         StreamExecutionEnvironment env =
30             StreamExecutionEnvironment.getExecutionEnvironment();
31         env.setParallelism(1);
32         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
33         SingleOutputStreamOperator<Tuple2<String, Integer>> operator = node1.map(new
34             MapFunction<String, Tuple2<String, Integer>>() {
35                 @Override
36                 public Tuple2<String, Integer> map(String s) throws Exception {
37                     String[] lines = s.split(",");
```

```

36         return Tuple2.of(lines[0], Integer.parseInt(lines[1]));
37     }
38 });
39 OutputTag<Tuple2<String, Integer>> seriousLade = new OutputTag<>(
40     "seriousLadeDate",
41     Types.TUPLE(Types.STRING, Types.INT)
42 );
43 //分配水位线, 最大延迟3s
44 SingleOutputStreamOperator<String> process =
operator.assignTimestampsAndWatermarks(WatermarkStrategy.<Tuple2<String,
Integer>>forBoundedOutOfOrderness(Duration.ofSeconds(3)).withTimestampAssigner((element,
recordTimestamp) -> element.f1 * 1000))
45     .keyBy(t -> t.f0)
46     .window(TumblingEventTimeWindows.of(Time.seconds(5)))
47     //允许最大严重乱序时间为2s, 触发计算窗口之后的延迟
48     .allowedLateness(Time.seconds(2))
49     //超过最大严重乱序时间保存到哪里, 保存到侧输出流
50     .sideOutputLateData(seriousLade)
51     //process处理, 对窗口数据中的元素进行统计, 生成[单词, 出现次数], 并将窗口开始时间
和结束时间打印到控制台
52     .process(new ProcessWindowFunction<Tuple2<String, Integer>, String,
String, TimeWindow>() {
53         @Override
54         public void process(String s, Context context,
Iterable<Tuple2<String, Integer>> iterable, Collector<String> collector) throws
Exception {
55             int size = IteratorUtils.toList(iterable.iterator()).size();
56             collector.collect(s + ":" + size);
57             System.out.println("当前窗口的start: " +
context.window().getStart() + " 窗口结束时间: " + context.window().getEnd());
58         }
59     });
60 });
61 process.print();
62 process.getSideOutput(seriousLade).print();
63 env.execute();
64 }
65 }
66

```

state状态

```
1 package sz.base.flink.keyedstate;
2
3 import org.apache.flink.api.common.functions.FlatMapFunction;
4 import org.apache.flink.api.common.functions.RichReduceFunction;
5 import org.apache.flink.api.common.state.ValueState;
6 import org.apache.flink.api.common.state.ValueStateDescriptor;
7 import org.apache.flink.api.common.typeinfo.Types;
8 import org.apache.flink.api.java.tuple.Tuple2;
9 import org.apache.flink.configuration.Configuration;
10 import org.apache.flink.streaming.api.datastream.DataStreamSource;
11 import org.apache.flink.streaming.api.datastream.KeyedStream;
12 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
13 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
14 import org.apache.flink.util.Collector;
15
16 /**
17  * 单词统计
18  * 主要用于flink自带的state管理的算子 sum，本身是支持state状态
19  * 自定义一个 state -> ValueState 保存每次聚合的中间结果 来实现单词次数的聚合
20  * state的数据结构分类：
21  * 1.ValueState: 存储单值
22  * 2.ListState: 存储的是值的集合
23  * 3.MapState: 存储的是key-value 键值对
24  * 4.BroadcastState : 广播状态，使用在 broadcast（广播状态），用于广播变量中
25  * 5.ReduceState: 聚合状态
26  * <p>
27  * 分析：
28  * 1.做一个常规的wordcount，sum去看单词统计的结果
29  * 2.自定义适用ValueState 保存每次聚合的中间结果 来实现单词次数的聚合
30  * <p>
31  * 开发步骤：
32  * 1.创建流环境
33  * 2.设置并行度参数
34  * 3.构建socket 数据源
35  * 4.每行输入数据的单词的拆分
36  * 5.根据 word 进行分组/分流
37  * 6.先用 sum 进行求和（自带的算子带状态）
38  * 7.使用 reduce 聚合（没有状态）的算子，如何去做
39  * 8.打印输出的结果
```

```

40  * 9.执行流环境
41  */
42
43
44 public class WordcountKeyedState_1 {
45     public static void main(String[] args) throws Exception {
46         // * 1.创建流环境
47         StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
48         // * 2.设置并行度参数
49         env.setParallelism(1);
50         // * 3.构建socket 数据源
51         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
52         // * 4.每行输入数据的单词的拆分
53         SingleOutputStreamOperator<Tuple2<String, Integer>> flatMap = node1.flatMap(new
FlatMapFunction<String, Tuple2<String, Integer>>() {
54             @Override
55             public void flatMap(String s, Collector<Tuple2<String, Integer>> collector)
throws Exception {
56                 String[] lines = s.split(",");
57                 for (String line : lines) {
58                     collector.collect(Tuple2.of(line, 1));
59                 }
60             }
61         });
62         // * 5.根据 word 进行分组/分流
63         KeyedStream<Tuple2<String, Integer>, String> keyedStream = flatMap.keyBy(t ->
t.f0);
64         // * 6.先用 sum 进行求和（自带的算子带状态）
65         // SingleOutputStreamOperator<Tuple2<String, Integer>> result =
keyedStream.sum(1);
66         // * 7.使用 reduce 聚合（没有状态）的算子，如何去做
67         SingleOutputStreamOperator<Tuple2<String, Integer>> result =
keyedStream.reduce(new RichReduceFunction<Tuple2<String, Integer>>() {
68             //定义状态的描述器,中间结果状态
69             ValueState<Tuple2<String, Integer>> reduceState = null;
70
71             /**
72              * 初始化工作
73              * 获取ValueState，用于保存或读取中间结果state的值（中间结果值）
74              * @param parameters

```



```

75         * @throws Exception
76     */
77     @Override
78     public void open(Configuration parameters) throws Exception {
79         //从上下文变量获取
80         reduceState = getRuntimeContext().getState(new
ValueStateDescriptor<Tuple2<String, Integer>>("reduceState", Types.TUPLE(Types.STRING,
Types.INT)));
81
82     }
83
84     /**
85     * 做值的累加，获取ValueState中的值和当前的值进行累加，保存到状态中
86     * @param stringIntegerTuple2
87     * @param t1
88     * @return
89     * @throws Exception
90     */
91     @Override
92     public Tuple2<String, Integer> reduce(Tuple2<String, Integer>
stringIntegerTuple2, Tuple2<String, Integer> t1) throws Exception {
93         Tuple2<String, Integer> value = reduceState.value();
94         //第一次存的时候，没有值，需要对状态赋值
95         if (value == null) {
96             value = stringIntegerTuple2;
97         }
98         //更新中间结果
99         reduceState.update(Tuple2.of(value.f0, value.f1 + t1.f1));
100         return Tuple2.of(value.f0, value.f1 + t1.f1);
101     }
102
103
104     @Override
105     public void close() throws Exception {
106         System.out.println(reduceState.value().f0 + "--" +
reduceState.value().f1);
107     }
108     });
109     // * 8.打印输出的结果
110     result.print();
111     // * 9.执行流环境

```

```
112         env.execute();
113     }
114 }
115
```

operator state

```
1  package sz.base.flink.ouperator;
2
3  import org.apache.flink.api.common.restartstrategy.RestartStrategies;
4  import org.apache.flink.api.common.state.ListState;
5  import org.apache.flink.api.common.state.ListStateDescriptor;
6  import org.apache.flink.api.common.typeinfo.Types;
7  import org.apache.flink.runtime.state.FunctionInitializationContext;
8  import org.apache.flink.runtime.state.FunctionSnapshotContext;
9  import org.apache.flink.streaming.api.checkpoint.CheckpointedFunction;
10 import org.apache.flink.streaming.api.datastream.DataStreamSource;
11 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
12 import org.apache.flink.streaming.api.functions.source.SourceFunction;
13
14 import java.awt.peer.CheckboxMenuItemPeer;
15
16 /**
17  * 需求：模拟一个消费kafka，将读取每一条数据都记录一个offset+=1，将offset 偏移量保存到
18  * operator state 中，每生成 5 条数据，模拟个bug， 让程序自动重启，接着上次保存的最新的
19  * offset 接着执行。
20  * Operator state 实现接口：
21  * 1.ListCheckpointed
22  * 2.CheckpointedFunction
23  * 开发步骤：
24  */
25 public class OperatorStateDemo_2 {
26     public static void main(String[] args) throws Exception {
27         //1.创建流执行环境，设置并行度
28         StreamExecutionEnvironment env =
29             StreamExecutionEnvironment.getExecutionEnvironment();
30         env.setParallelism(1);
31         //2.启用checkpoint ，每1s 做一次 checkpoint
32         env.enableCheckpointing(1000);
33         //3.设置重启策略，程序挂掉，自动重新启动起来
```

```

32     env.setRestartStrategy(RestartStrategies.fixedDelayRestart(3, 3000));
33     //4.添加自定义数据源, 将模拟的 offset 返回
34     DataSource<Long> source = env.addSource(new MySource());
35     //5.map操作, 将Long 类型转换成字符串类型并打印输出
36     source.map(t->t.toString()).print();
37     //6.执行流环境
38     env.execute();
39 }
40
41 private static class MySource implements SourceFunction<Long>, CheckpointedFunction
42 {
43     //自定义source 实现 SourceFunction<Integer>和 CheckpointedFunction
44     //1.定义变量
45     //1.1定义标记用于循环生成标记
46     boolean isRunning = true;
47     //1.2定义 currentCounter 用于保存当前的计数值
48     Long currentCounter = 0L;
49     //1.3定义ListState
50     ListState<Long> offsetState = null;
51     //1.4定义ListState 描述
52
53     //2.重写SnapshotState 方法, 清除状态和将最新的累加值添加到状态中 (给当前状态做一个快照)
54     @Override
55     public void snapshotState(FunctionSnapshotContext functionSnapshotContext)
56     throws Exception {
57         //将上一次的 state 状态清空,然后最新的添加进去(checkpoint记录的(累加器)偏移量,中间状态),只保存一个最新的中间状态
58         offsetState.clear();
59         offsetState.add(currentCounter);
60     }
61
62     //3.重写 initializeState方法, 获取状态并遍历Iterable, 将其赋值给 currentCounter
63     @Override
64     public void initializeState(FunctionInitializationContext context) throws
65     Exception {
66         //3.1将历史存储(checkpoint中)的状态遍历出来 赋值给currentCounter变量
67         offsetState = context.getOperatorStateStore().getListState(
68             new ListStateDescriptor<Long>(
69                 "offsetState", Types.LONG

```

```

68         )
69
70     );
71     //将从 operator state 中读取最新 offset 赋值给 currentCounter
72     Iterable<Long> longs = offsetState.get();
73     //将 List 中最新的 offset 赋值给 currentCounter
74     for (Long aLong : longs) {
75         currentCounter = aLong;
76     }
77 }
78
79 //4.重写run方法，每秒循环收集累加的counter，每5个生成一个异常
80 @Override
81 public void run(SourceContext<Long> sourceContext) throws Exception {
82     //4.1 持续循环
83     while (isRunning) {
84         //4.2 对currentCounter 累加 读一条+1，相当于偏移量
85         currentCounter++;
86         //4.3 收集当前累加的值
87         sourceContext.collect(currentCounter);
88         //4.4休眠1s
89         Thread.sleep(1000);
90         //4.5 如果是5的倍数就模拟输出异常
91         if (currentCounter % 5 == 0) {
92             throw new RuntimeException("出错了，出bug了");
93         }
94     }
95 }
96
97 //5.重写cancel方法
98 @Override
99 public void cancel() {
100     isRunning = false;
101 }
102 }
103 }
104

```

状态有效期TTL

```

1 package sz.base.flink.keyedstate;
2
3
4 import org.apache.commons.io.FileUtils;
5 import org.apache.flink.api.common.functions.FlatMapFunction;
6 import org.apache.flink.api.common.functions.RichFlatMapFunction;
7 import org.apache.flink.api.common.functions.RichReduceFunction;
8 import org.apache.flink.api.common.state.StateTtlConfig;
9 import org.apache.flink.api.common.state.ValueState;
10 import org.apache.flink.api.common.state.ValueStateDescriptor;
11 import org.apache.flink.api.common.time.Time;
12 import org.apache.flink.api.common.typeinfo.Types;
13 import org.apache.flink.api.java.tuple.Tuple2;
14 import org.apache.flink.configuration.Configuration;
15 import org.apache.flink.streaming.api.datastream.DataStreamSource;
16 import org.apache.flink.streaming.api.datastream.KeyedStream;
17 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
18 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
19 import org.apache.flink.streaming.api.functions.source.SourceFunction;
20 import org.apache.flink.util.Collector;
21
22 import java.io.File;
23 import java.util.List;
24
25 /**
26  * 需求- 读取文件数据，每3s读取一行，然后将读取到的字符串按照空格拆分计算每个单词出现的次数
27  * 使用到state ttl，将TTL 设置为 6s，查看统计的结果
28  * 输入：
29  * 文件中的所有单词，每行1~n单词，每3s 读取一行
30  * 输出：
31  * hello,1
32  * hello,2
33  * spark,2
34  * flink,3
35  * hello,1
36  */
37 public class WordcountTTLState_3 {
38     public static void main(String[] args) throws Exception {
39         // * 1.创建流环境

```

```

40      StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
41      // * 2.设置并行度参数
42      env.setParallelism(1);
43      // * 3.读取文件中的数据，每3s中读取一行
44      DataSource<String> node1 = env.addSource(new SourceFunction<String>() {
45          boolean isRunning = true;
46
47          @Override
48          public void run(SourceContext<String> sourceContext) throws Exception {
49              while (isRunning) {
50                  //直接通过文件的工具类去读取文件
51                  List<String> lines = FileUtils.readLines(
52                      new File("input/words.txt"), "utf-8"
53                  );
54                  //遍历每行的数据，并打印输出，输出每 3s 输出一行
55                  for (String line : lines) {
56                      sourceContext.collect(line);
57                      //休眠3s
58                      Thread.sleep(3000);
59                  }
60              }
61          }
62
63          @Override
64          public void cancel() {
65              isRunning = false;
66          }
67      });
68      // * 4.每行输入数据的单词的拆分
69      SingleOutputStreamOperator<Tuple2<String, Integer>> flatMap = node1.flatMap(new
FlatMapFunction<String, Tuple2<String, Integer>>() {
70          @Override
71          public void flatMap(String s, Collector<Tuple2<String, Integer>> collector)
throws Exception {
72              String[] lines = s.split(",");
73              for (String line : lines) {
74                  collector.collect(Tuple2.of(line, 1));
75              }
76          }
77      });

```

```

78         // * 5.根据 word 进行分组/分流
79         KeyedStream<Tuple2<String, Integer>, String> keyedStream = flatMap.keyBy(t ->
t.f0);
80         // * 6.先用 sum 进行求和（自带的算子带状态）
81         //         SingleOutputStreamOperator<Tuple2<String, Integer>> result =
keyedStream.sum(1);
82         // * 7.使用
83         SingleOutputStreamOperator<Tuple2<String, Integer>> result =
keyedStream.flatMap(new RichFlatMapFunction<Tuple2<String, Integer>, Tuple2<String,
Integer>>() {
84             @Override
85             public void flatMap(Tuple2<String, Integer> stringIntegerTuple2,
Collector<Tuple2<String, Integer>> collector) throws Exception {
86                 Tuple2<String, Integer> value = reduceState.value();
87                 //第一次存的时候，没有值，需要对状态赋值
88                 if (value == null) {
89                     value = stringIntegerTuple2;
90                     collector.collect(value);
91                     reduceState.update(value);
92                 }else {
93                     //更新中间结果
94                     Tuple2<String, Integer> of = Tuple2.of(value.f0, value.f1 + value.f1);
95                     reduceState.update(of);
96                     collector.collect(of);
97                 }}
98
99         //定义状态的描述器,中间结果状态
100         ValueState<Tuple2<String, Integer>> reduceState = null;
101
102         /**
103          * 初始化工作
104          * 获取ValueState，用于保存或读取中间结果state的值（中间结果值）
105          * @param parameters
106          * @throws Exception
107          */
108         @Override
109         public void open(Configuration parameters) throws Exception {
110
111             //定义状态的描述器
112             //设置生命周期ttl

```

```

113         StateTtlConfig builder =
StateTtlConfig.newBuilder(Time.seconds(6)).setUpdateType(StateTtlConfig.UpdateType.OnReadAndWrite).setStateVisibility(StateTtlConfig.StateVisibility.NeverReturnExpired).build();
114         ValueStateDescriptor<Tuple2<String, Integer>> reduceState = new
ValueStateDescriptor<>("reduceState", Types.TUPLE(Types.STRING, Types.INT));
115         reduceState.enableTimeToLive(builder);
116         //从上下文变量获取
117         this.reduceState = getRuntimeContext().getState(reduceState);
118
119     }
120
121
122     @Override
123     public void close() throws Exception {
124         //System.out.println(reduceState.value().f0 + "--" +
reduceState.value().f1);
125     }
126 });
127 // * 8.打印输出的结果
128 result.print();
129 // * 9.执行流环境
130 env.execute();
131 }
132 }
133

```

broadcaststate

```

1 package sz.base.flink.broadcast;
2
3 import org.apache.flink.api.common.state.BroadcastState;
4 import org.apache.flink.api.common.state.MapStateDescriptor;
5 import org.apache.flink.api.common.state.ReadOnlyBroadcastState;
6 import org.apache.flink.api.common.typeinfo.Types;
7 import org.apache.flink.api.java.tuple.Tuple2;
8 import org.apache.flink.streaming.api.datastream.BroadcastStream;
9 import org.apache.flink.streaming.api.datastream.DataStreamSource;
10 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
11 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;

```



```

12 import org.apache.flink.streaming.api.functions.co.BroadcastProcessFunction;
13 import org.apache.flink.streaming.api.functions.source.SourceFunction;
14 import org.apache.flink.util.Collector;
15
16 import java.util.*;
17 import java.util.concurrent.TimeUnit;
18
19 /**
20  * 公司有10个广告位，其广告的内容（描述和图片）会经常变动（广告到期，更换广告等）
21  */
22 public class BroadcastDemo_4 {
23     public static void main(String[] args) throws Exception {
24         //1.获取流执行环境
25         StreamExecutionEnvironment env =
26             StreamExecutionEnvironment.getExecutionEnvironment();
27         //2.设置参数，设置检查点每5s中一次
28         env.setParallelism(1);
29         env.enableCheckpointing(5000);
30         //3构建数据源
31         //3.1构建数据源，并将数据转换成整数值类型
32         DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
33         //3.2构建自定义数据源用于获取广告位信息（导入数据源）
34         DataStreamSource<Map<Integer, Tuple2<String, String>>> mapDataStreamSource =
35             env.addSource(new MySourceForBroadcastFunction());
36         //4.将广告位信息广播出去，广播描述信息为
37         MapStateDescriptor<Integer, Tuple2<String, String>>
38             //定义MapStateDescriptor
39             MapStateDescriptor<Integer, Tuple2<String, String>> advertiseState = new
40             MapStateDescriptor<>(
41                 "advertiseState", Types.INT,
42                 Types.TUPLE(Types.STRING, Types.STRING)
43             );
44         BroadcastStream<Map<Integer, Tuple2<String, String>>> broadcast =
45             mapDataStreamSource.broadcast(advertiseState);
46         //5.将广告ID流connect上广播流
47         SingleOutputStreamOperator<Tuple2<String, String>> process =
48             node1.connect(broadcast)
49                 //6.对关联的数据进行拉宽操作process
50                 .process(new BroadcastProcessFunction<String, Map<Integer,
51                     Tuple2<String, String>>, Tuple2<String, String>>() {
52                     //6.1处理每个element
53                     @Override

```

```

47         public void processElement(String s, ReadOnlyContext
readOnlyContext, Collector<Tuple2<String, String>> collector) throws Exception {
48             //通过上下文获取广播状态,把广播出去的获取到
49             ReadOnlyBroadcastState<Integer, Tuple2<String, String>>
broadcastState = readOnlyContext.getBroadcastState(advertiseState);
50             //根据value获取配置信息
51             Tuple2<String, String> stringIntegerTuple2 =
broadcastState.get(Integer.parseInt(s));
52             //如果配置信息不为空就收集
53             if (stringIntegerTuple2 != null) {
54                 collector.collect(stringIntegerTuple2);
55             }
56         }
57
58         //6.2处理广播element
59         @Override
60         public void processBroadcastElement(Map<Integer, Tuple2<String,
String>> integerTuple2Map, Context context, Collector<Tuple2<String, String>> collector)
throws Exception {
61             //通过上下文获取广播状态
62             BroadcastState<Integer, Tuple2<String, String>> broadcastState =
context.getBroadcastState(advertiseState);
63             //清空
64             broadcastState.clear();
65             //保存最新的
66             broadcastState.putAll(integerTuple2Map);
67         }
68     });
69     process.print();
70     env.execute();
71
72
73 }
74
75     public static class MySourceForBroadcastFunction implements
SourceFunction<Map<Integer, Tuple2<String, String>>> {
76         private final Random random = new Random();
77         private final List<Tuple2<String, String>> ads = Arrays.asList(
78             Tuple2.of("baidu", "搜索引擎"),
79             Tuple2.of("google", "科技大牛"),
80             Tuple2.of("aws", "全球领先的云平台"),
81             Tuple2.of("aliyun", "全球领先的云平台"),

```

```

82         Tuple2.of("腾讯", "氪金使我变强"),
83         Tuple2.of("阿里巴巴", "电商龙头"),
84         Tuple2.of("字节跳动", "靠算法出名"),
85         Tuple2.of("美团", "黄色小公司"),
86         Tuple2.of("饿了么", "蓝色小公司"),
87         Tuple2.of("瑞幸咖啡", "就是好喝")
88     );
89     private boolean isRun = true;
90
91     @Override
92     public void run(SourceContext<Map<Integer, Tuple2<String, String>>> ctx) throws
Exception {
93         while (isRun) {
94             Map<Integer, Tuple2<String, String>> map = new HashMap<>();
95             int keyCounter = 0;
96             for (int i = 0; i < ads.size(); i++) {
97                 keyCounter++;
98                 map.put(keyCounter, ads.get(random.nextInt(ads.size())));
99             }
100             ctx.collect(map);
101
102             TimeUnit.SECONDS.sleep(5L);
103         }
104     }
105
106     @Override
107     public void cancel() {
108         this.isRun = false;
109     }
110 }
111 }
112

```

端对端仅一次语义

kafka-kafka

```

1 package sz.base.flink.exatylyonce;
2

```

```
3 import org.apache.flink.api.common.functions.FlatMapFunction;
4 import org.apache.flink.api.common.restartstrategy.RestartStrategies;
5 import org.apache.flink.api.common.serialization.SimpleStringSchema;
6 import org.apache.flink.api.common.time.Time;
7 import org.apache.flink.api.java.tuple.Tuple2;
8 import org.apache.flink.streaming.api.CheckpointingMode;
9 import org.apache.flink.streaming.api.datastream.DataStreamSource;
10 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
11 import org.apache.flink.streaming.api.environment.CheckpointConfig;
12 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
13 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaConsumer;
14 import org.apache.flink.streaming.connectors.kafka.FlinkKafkaProducer;
15 import org.apache.flink.streaming.connectors.kafka.KafkaSerializationSchema;
16 import org.apache.flink.util.Collector;
17 import org.apache.kafka.clients.consumer.ConsumerConfig;
18 import org.apache.kafka.clients.producer.ProducerConfig;
19 import org.apache.kafka.clients.producer.ProducerRecord;
20
21 import javax.annotation.Nullable;
22 import java.util.Properties;
23 import java.util.Random;
24
25 /**
26  * 需求：读取kafka中的数据 将数据通过实现一个wordcount 的逻辑 并将其写入到kafka中
27  * 要求：支持exactly-once 语义
28  * 开发步骤：
29  * 1. 获取流环境
30  * 2. 设置checkpoint 1s 状态后端到hdfs或本地file
31  * 设置checkpoint属性配置，支持仅一次、超时、并行、容忍、最小间隔、取消任务保存checkpoint
32  * 3. 设置重启策略3次，10s间隔
33  * 4. 配置kafka consumer 属性：服务器、消费组、重置从最新、自动发现分区
34  * 5. 设置consumer设置从最新的读取
35  * 6. 设置提交offset数据越
36  * 7. 切分单词并记1，遍历每个单词中，随机从0~4中给一个值，如果该值大于3就模拟异常bug，将bug收集
37  * 8. 对数据进行分组、聚合
38  * 9. 对最终word和count进行map映射成 word:::count
39  * 10 设置写到kafka的属性 服务器和事务超时时间5s
40  * 11. 创建FlinkKafkaProducer
41  * 12. 将producer 添加到sink，需要支持仅一次语义
42  * 13. 执行流环境
```

```

43  */
44  public class FlinkFromKafkaToKafka {
45      public static void main(String[] args) throws Exception {
46          System.setProperty("HADOOP_USER_NAME", "ROOT");
47          // * 1.获取流环境
48          StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
49          // * 2.设置checkpoint 1s 状态后端到hdfs或本地file
50          env.setParallelism(1);
51          env.enableCheckpointing(1000);
52          // * 设置checkpoint属性配置，支持仅一次、超时、并行、容忍、最小间隔、取消任务保存
checkpoint
53          CheckpointConfig conf = env.getCheckpointConfig();
54          conf.setCheckpointStorage("hdfs://node1:8020/flink-
checkpoints/FlinkFromKafkaToKafka");
55          conf.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
56          conf.setCheckpointTimeout(60000);
57          conf.setMaxConcurrentCheckpoints(1);
58          conf.setTolerableCheckpointFailureNumber(8);
59          conf.setMinPauseBetweenCheckpoints(500);
60
conf.enableExternalizedCheckpoints(CheckpointConfig.ExternalizedCheckpointCleanup.RETAIN
_ON_CANCELLATION);
61          // * 3.设置重启策略3次，10s间隔n
62          env.setRestartStrategy(RestartStrategies.fixedDelayRestart(3,
Time.seconds(10)));
63          // * 4.配置kafka consumer 属性：服务器、消费组、重置从最新、自动发现分区
64          Properties properties = new Properties();
65          properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
"node1:9092,node2:9092,node3:9092");
66          properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, "__consumer_src_topic_");
67          properties.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, true + "");
68          properties.setProperty(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, 10000 +
"");
69
properties.setProperty(FlinkKafkaConsumer.KEY_PARTITION_DISCOVERY_INTERVAL_MILLIS, 30 *
60 * 1000 + "");
70          //创建读取kafka的实例
71          FlinkKafkaConsumer<String> srctopic = new FlinkKafkaConsumer<>("srctopic", new
SimpleStringSchema(), properties);
72          // * 5.设置consumer设置从最新的读取
73          srctopic.setStartFromLatest();
74          // 设置提交offset到checkpoint

```

```

75         // * 6.设置提交offset数据源
76         srctopic.setCommitOffsetsOnCheckpoints(true);
77         // 添加kafka数据源
78         DataStreamSource<String> source = env.addSource(srctopic);
79         // * 7.切分单词并记1 ， 遍历每个单词中，随机从0~4中给一个值，如果该值大于3就模拟异常
bug，将bug收集
80         SingleOutputStreamOperator<String> result = source.flatMap(new
FlatMapFunction<String, Tuple2<String, Integer>>() {
81             Random rm = new Random();
82
83             @Override
84             public void flatMap(String s, Collector<Tuple2<String, Integer>> collector)
throws Exception {
85                 String[] words = s.split(" ");
86                 for (String word : words) {
87                     int random = rm.nextInt(5);
88                     //模拟一个bug，如果等于4 报错
89                     if (random == 4) {
90                         throw new Exception("程序除了一点点小bug ， 请检查!");
91                     }
92                     //输出这个tuple2
93                     collector.collect(Tuple2.of(word, 1));
94                 }
95             }
96         })
97         // * 8.对数据进行分组、聚合
98         .keyBy(k -> k.f0)
99         // * 9.对最终word和count进行map映射成 word::count
100        .map(t -> t.f0 + "::" + t.f1);
101        // * 10设置写到kafka的属性 服务器和事务超时时间5s
102        Properties properties1 = new Properties();
103        properties1.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
"node1:9092,node2:9092,node3:9092");
104        properties1.setProperty(ProducerConfig.TRANSACTION_TIMEOUT_CONFIG, 2000 + "");
105        properties1.setProperty(ProducerConfig.BATCH_SIZE_CONFIG, 5 + "");
106        FlinkKafkaProducer<String> producer = new FlinkKafkaProducer<String>(
107            "outtopic", new KafkaSerializationSchema<String>() {
108                @Override
109                public ProducerRecord<byte[], byte[]> serialize(String s, @Nullable Long
aLong) {
110                    return new ProducerRecord<>("outtopic", s.getBytes());

```

```

111         }
112     }, properties1, FlinkKafkaProducer.Semantic.AT_LEAST_ONCE
113 );
114 // * 11.创建FlinkKafkaProducer
115 // * 12.将producer 添加到sink ， 需要支持仅一次语义
116 result.addSink(producer);
117 // * 13.执行流环境
118 env.execute();
119 }
120 }
121

```

ProcessFunction

实现onTimer方法

```

1 package sz.base.flink.process;
2
3 import org.apache.commons.collections.IteratorUtils;
4 import org.apache.flink.api.common.functions.MapFunction;
5 import org.apache.flink.api.common.state.ListState;
6 import org.apache.flink.api.common.state.ListStateDescriptor;
7 import org.apache.flink.api.common.time.Time;
8 import org.apache.flink.api.common.typeinfo.Types;
9 import org.apache.flink.api.java.tuple.Tuple2;
10 import org.apache.flink.configuration.Configuration;
11 import org.apache.flink.streaming.api.TimeCharacteristic;
12 import org.apache.flink.streaming.api.datastream.DataStreamSource;
13 import org.apache.flink.streaming.api.datastream.KeyedStream;
14 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
15 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
16 import org.apache.flink.streaming.api.functions.KeyedProcessFunction;
17 import org.apache.flink.util.Collector;
18
19 import java.text.SimpleDateFormat;
20 import java.util.Iterator;
21
22 /**

```

```
23  * 需求：需要实时监控服务器机架的温度，如果一定时间内温度超过了一定阈值（100度），且后一次上报的
    温度超过了前一次上报的温度，需要触发告警（温度持续升高中）
24  */
25  public class ServerMonitor {
26      public static void main(String[] args) throws Exception {
27          //初始化流计算运行环境，制定并行度为1
28          StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
29          env.setParallelism(1);
30          //开启checkpoint
31          env.enableCheckpointing(1000);
32          //设置事件时间属性,现在版本已经默认是EventTime
33          // env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
34          //接入socket数据源，获取数据 1,100
35          DataStreamSource<String> node1 = env.socketTextStream("node1", 9999);
36          //将获取到的数据转换成tuple2<Integer,Integer>
37          KeyedStream<Tuple2<String, Integer>, String> keyedStream = node1.map(new
MapFunction<String, Tuple2<String, Integer>>() {
38              @Override
39              public Tuple2<String, Integer> map(String s) throws Exception {
40                  String[] lines = s.split(",");
41                  //生成tuple2 ,数据 1,100 2,101 3,103 4,104
42                  return Tuple2.of(lines[0], Integer.parseInt(lines[1]));
43              }
44          })
45          //根据f0进行分流
46          .keyBy(k -> k.f0);
47          SingleOutputStreamOperator<String> process = keyedStream.process(new
MyProcessFunction());
48          process.printToErr();
49          env.execute();
50          //实现如下方法
51          //2.
52
53      }
54
55      ;
56
57      //自定义processFunction对象，继承KeyedProcessFunction<Tuple, Tuple2<Integer,Integer>,
String>抽象类
```



```

58     public static class MyProcessFunction extends KeyedProcessFunction<String,
Tuple2<String, Integer>, String> {
59         //定义一个变量存储列表中最后一个值
60         Integer lastTemperature = 0;
61         ListState<Tuple2<String, Integer>> lastTemperatureState = null;
62
63         //初始化ListState<Tuple2<机架id, 机架温度>>保存上次温度
64         //1.open 获取ListState Tuple2<Integer,Integer>获取状态
65         //超过100度, 并且比上次温度高的数据保存到状态里 ListState
66         //状态的数据结构: ValueState ListState BroadcastState ReduceState MapState
67         @Override
68         public void open(Configuration parameters) throws Exception {
69             lastTemperatureState = getRuntimeContext().getListState(
70                 new ListStateDescriptor<Tuple2<String, Integer>>(
71                     "lastTemperatureState", Types.TUPLE(Types.STRING, Types.INT)
72                 )
73             );
74         }
75
76         /**
77          * 主要实现: 读取的每个机架的温度, 如果高于100, 并且比上次高就给个定时器然后报警, 核心
业务逻辑
78          *
79          * @param stringIntegerTuple2
80          * @param context
81          * @param collector
82          * @throws Exception
83          */
84         @Override
85         public void processElement(Tuple2<String, Integer> stringIntegerTuple2, Context
context, Collector<String> collector) throws Exception {
86             //定义一个时间格式化工具
87             SimpleDateFormat sdf =new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
88             //当前处理的时间
89             long currentTime = context.timerService().currentProcessingTime();
90
91             Iterable<Tuple2<String, Integer>> temperatures = lastTemperatureState.get();
92             for (Tuple2<String, Integer> temperature : temperatures) {
93                 lastTemperature = temperature.f1;
94             }

```

```

95         if (stringIntegerTuple2.f1 > 100 && (stringIntegerTuple2.f1 >
lastTemperature)) {
96             //将当前的温度先保存到状态,用于下次比较
97             lastTemperatureState.add(stringIntegerTuple2);
98             //获取当前处理时间,注册一个定时器,定时时间为5s
99             context.timerService().registerProcessingTimeTimer(currentTime+5000);
100             //返回字符串,打印一下当前的温度和当前的处理时间
101             collector.collect( String.format(+stringIntegerTuple2.f1+" 当前的处理时
间: "+sdf.format(currentTime)));
102         }else{
103             //当前温度 < 100 || 下次温度小于上次的温度
104             lastTemperatureState.clear();
105             //删除触发器
106             context.timerService().deleteEventTimeTimer(currentTime+5000);
107             //输出字符串
108             collector.collect("当前的告警触发器被解除!");
109         }
110     }
111     //定义触发警告定时器的时长和格式化为: yyyy-MM-dd HH:mm:ss.SSS
112
113     @Override
114     public void onTimer(long timestamp, OnTimerContext ctx, Collector<String> out)
throws Exception {
115         //获取状态中数据size,从状态中获取连续上涨温度有多少次
116         Iterator<Tuple2<String, Integer>> iterator =
lastTemperatureState.get().iterator();
117         int size = IteratorUtils.toList(iterator).size();
118         System.out.println("当前超过100度并累加温度升高的个数为: "+size);
119         if(size>1){
120             System.out.println("当前温度过高,高温报警,滴滴滴!");
121             out.collect("当前温度过高,高温报警,滴滴滴!");
122         }
123         //清空历史数据
124         lastTemperatureState.clear();
125     }
126 }
127
128 ;
129 }
130

```

双流JOIN

窗口实现

```
1 package sz.base.flink.join;
2
3 import com.alibaba.fastjson.JSON;
4 import lombok.Data;
5 import org.apache.flink.api.common.eventtime.*;
6 import org.apache.flink.api.common.functions.JoinFunction;
7 import org.apache.flink.configuration.Configuration;
8 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
9 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
10 import org.apache.flink.streaming.api.functions.source.RichSourceFunction;
11 import org.apache.flink.streaming.api.windowing.assigners.SlidingProcessingTimeWindows;
12 import org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows;
13 import org.apache.flink.streaming.api.windowing.time.Time;
14
15 import java.math.BigDecimal;
16 import java.util.ArrayList;
17 import java.util.List;
18 import java.util.Random;
19 import java.util.UUID;
20 import java.util.concurrent.TimeUnit;
21
22 /**
23  * 需求-使用两个指定source模拟数据，一个source是订单明细，一个source是商品数据，我们通过
24  * window join，将数据关联到一起
25  *
26  *
27  */
28 public class DoubleStreamJoin {
29     public static void main(String[] args) throws Exception {
30         //获取流执行环境
31         StreamExecutionEnvironment env =
32             StreamExecutionEnvironment.getExecutionEnvironment();
33         env.setParallelism(1);
34         //添加数据源，商品表和订单表
```

```

34      SingleOutputStreamOperator<Goods> goodsDataStream = env.addSource(new
GoodsSource()).assignTimestampsAndWatermarks(new GoodsWatermark());
35      //订单项数据流
36      SingleOutputStreamOperator<OrderItem> orderDataStream = env.addSource(new
OrderItemSource()).assignTimestampsAndWatermarks(new OrderItemWatermark());
37      //分配水印机制
38      //进行双流join（滚动事件时间窗口）
39
orderDataStream.join(goodsDataStream).where(OrderItem::getGoodsId).equalTo(Goods::getGoodsId)
40      //滚动窗口
41      //      .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
42      //滑动窗口
43      .window(SlidingProcessingTimeWindows.of(Time.seconds(5), Time.seconds(1)))
44      .apply(new JoinFunction<OrderItem, Goods, FactOrderItem>() {
45          @Override
46          public FactOrderItem join(OrderItem orderItem, Goods goods) throws
Exception {
47              FactOrderItem factOrderItem = new FactOrderItem();
48              factOrderItem.setGoodsId(goods.goodsId);
49              factOrderItem.setGoodsName(goods.goodsName);
50              factOrderItem.setCount(new BigDecimal(orderItem.count));
51              factOrderItem.setTotalMoney(goods.getGoodsPrice().multiply(new
BigDecimal(orderItem.count)));
52              return factOrderItem;
53          }
54      })
55      //打印结果
56      .printToErr();
57      //执行流环境
58      env.execute();
59
60
61  }
62  //商品类
63  @Data
64  public static class Goods {
65      private String goodsId;
66      private String goodsName;
67      private BigDecimal goodsPrice;
68

```

```

69     public static List<Goods> GOODS_LIST;
70     public static Random r;
71
72     static {
73         r = new Random();
74         GOODS_LIST = new ArrayList<>();
75         GOODS_LIST.add(new Goods("1", "小米12", new BigDecimal(4890)));
76         GOODS_LIST.add(new Goods("2", "iphone12", new BigDecimal(12000)));
77         GOODS_LIST.add(new Goods("3", "MacBookPro", new BigDecimal(15000)));
78         GOODS_LIST.add(new Goods("4", "Thinkpad X1", new BigDecimal(9800)));
79         GOODS_LIST.add(new Goods("5", "MeiZu One", new BigDecimal(3200)));
80         GOODS_LIST.add(new Goods("6", "Mate 40", new BigDecimal(6500)));
81     }
82
83     public static Goods randomGoods() {
84         int rIndex = r.nextInt(GOODS_LIST.size());
85         return GOODS_LIST.get(rIndex);
86     }
87
88     public Goods() {
89     }
90
91     public Goods(String goodsId, String goodsName, BigDecimal goodsPrice) {
92         this.goodsId = goodsId;
93         this.goodsName = goodsName;
94         this.goodsPrice = goodsPrice;
95     }
96
97     @Override
98     public String toString() {
99         return JSON.toJSONString(this);
100     }
101 }
102
103 //订单明细类
104 @Data
105 public static class OrderItem {
106     private String itemId;
107     private String goodsId;
108     private Integer count;

```

```

109
110     @Override
111     public String toString() {
112         return JSON.toJSONString(this);
113     }
114 }
115
116 //关联结果
117 @Data
118 public static class FactOrderItem {
119     private String goodsId;
120     private String goodsName;
121     private BigDecimal count;
122     private BigDecimal totalMoney;
123     @Override
124     public String toString() {
125         return JSON.toJSONString(this);
126     }
127 }
128 //构建一个商品Stream源（这个好比就是维表）
129 public static class GoodsSource extends RichSourceFunction<Goods> {
130     private Boolean isCancel;
131     @Override
132     public void open(Configuration parameters) throws Exception {
133         isCancel = false;
134     }
135     @Override
136     public void run(SourceContext sourceContext) throws Exception {
137         while(!isCancel) {
138             Goods.GOODS_LIST.stream().forEach(goods ->
sourceContext.collect(goods));
139             TimeUnit.SECONDS.sleep(1);
140         }
141     }
142     @Override
143     public void cancel() {
144         isCancel = true;
145     }
146 }
147 //构建订单明细Stream源

```

```

148     public static class OrderItemSource extends RichSourceFunction<OrderItem> {
149         private Boolean isCancel;
150         private Random r;
151         @Override
152         public void open(Configuration parameters) throws Exception {
153             isCancel = false;
154             r = new Random();
155         }
156         @Override
157         public void run(SourceContext sourceContext) throws Exception {
158             while(!isCancel) {
159                 Goods goods = Goods.randomGoods();
160                 OrderItem orderItem = new OrderItem();
161                 orderItem.setGoodsId(goods.getGoodsId());
162                 orderItem.setCount(r.nextInt(10) + 1);
163                 orderItem.setItemId(UUID.randomUUID().toString());
164                 sourceContext.collect(orderItem);
165                 orderItem.setGoodsId("111");
166                 sourceContext.collect(orderItem);
167                 TimeUnit.SECONDS.sleep(1);
168             }
169         }
170
171         @Override
172         public void cancel() {
173             isCancel = true;
174         }
175     }
176     //构建水印分配器（此处为了简单），直接使用系统时间了
177     public static class GoodsWatermark implements WatermarkStrategy<Goods> {
178
179         @Override
180         public TimestampAssigner<Goods>
181         createTimestampAssigner(TimestampAssignerSupplier.Context context) {
182             return (element, recordTimestamp) -> System.currentTimeMillis();
183         }
184
185         @Override
186         public WatermarkGenerator<Goods>
187         createWatermarkGenerator(WatermarkGeneratorSupplier.Context context) {

```

```

186         return new WatermarkGenerator<Goods>() {
187             @Override
188             public void onEvent(Goods event, long eventTimestamp, WatermarkOutput
output) {
189                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
190             }
191
192             @Override
193             public void onPeriodicEmit(WatermarkOutput output) {
194                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
195             }
196         };
197     }
198 }
199
200 public static class OrderItemWatermark implements WatermarkStrategy<OrderItem> {
201     @Override
202     public TimestampAssigner<OrderItem>
createTimestampAssigner(TimestampAssignerSupplier.Context context) {
203         return (element, recordTimestamp) -> System.currentTimeMillis();
204     }
205     @Override
206     public WatermarkGenerator<OrderItem>
createWatermarkGenerator(WatermarkGeneratorSupplier.Context context) {
207         return new WatermarkGenerator<OrderItem>() {
208             @Override
209             public void onEvent(OrderItem event, long eventTimestamp,
WatermarkOutput output) {
210                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
211             }
212             @Override
213             public void onPeriodicEmit(WatermarkOutput output) {
214                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
215             }
216         };
217     }
218 }
219 }
220

```


Interval join

```
1 package sz.base.flink.join;
2
3 import com.alibaba.fastjson.JSON;
4 import lombok.Data;
5 import org.apache.flink.api.common.eventtime.*;
6 import org.apache.flink.configuration.Configuration;
7 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
8 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
9 import org.apache.flink.streaming.api.functions.co.ProcessJoinFunction;
10 import org.apache.flink.streaming.api.functions.source.RichSourceFunction;
11 import org.apache.flink.streaming.api.windowing.time.Time;
12 import org.apache.flink.util.Collector;
13
14 import java.math.BigDecimal;
15 import java.util.ArrayList;
16 import java.util.List;
17 import java.util.Random;
18 import java.util.UUID;
19 import java.util.concurrent.TimeUnit;
20
21 /**
22  * 双流join-其中一个流指定一个上界和下界，和另外一个数据进行关联操作
23  */
24 public class IntervalStreamJoin {
25     public static void main(String[] args) throws Exception {
26         //获取流执行环境
27         StreamExecutionEnvironment env =
28             StreamExecutionEnvironment.getExecutionEnvironment();
29         //设置并行度
30         env.setParallelism(1);
31         //构建两个数据流
32         SingleOutputStreamOperator<Goods> goodsDataStream = env.addSource(new
33             GoodsSource11()).assignTimestampsAndWatermarks(new GoodsWatermark());
34         SingleOutputStreamOperator<OrderItem> orderDataStream = env.addSource(new
35             OrderItemSource()).assignTimestampsAndWatermarks(new OrderItemWatermark());
36         //双流interval join，计算前两秒和下一秒
37         SingleOutputStreamOperator<FactOrderItem> process = orderDataStream.keyBy(t ->
38             t.getGoodsId()).intervalJoin(goodsDataStream.keyBy(k ->
```

```

k.getGoodsId()).between(Time.seconds(-2), Time.seconds(1))
35 //                不包含上界[-2,)
36                .upperBoundExclusive()
37                //不包含下界两个一起用则都不包含
38 //                .lowerBoundExclusive()
39                //进行process处理
40                .process(new ProcessJoinFunction<OrderItem, Goods, FactOrderItem>() {
41                    @Override
42                    public void processElement(OrderItem orderItem, Goods goods,
Context context, Collector<FactOrderItem> collector) throws Exception {
43                        FactOrderItem factOrderItem = new FactOrderItem();
44                        factOrderItem.setGoodsId(goods.getGoodsId());
45                        factOrderItem.setCount(new BigDecimal(orderItem.count));
46                        factOrderItem.setGoodsName(goods.goodsName);
47                        factOrderItem.setTotalMoney(goods.getGoodsPrice().multiply(new
BigDecimal(orderItem.getCount())));
48                        collector.collect(factOrderItem);
49                    }
50                });
51        process.printToErr();
52        env.execute();
53    }
54
55    //商品类
56    @Data
57    public static class Goods {
58        private String goodsId;
59        private String goodsName;
60        private BigDecimal goodsPrice;
61
62        public static List<Goods> GOODS_LIST;
63        public static Random r;
64
65        static {
66            r = new Random();
67            GOODS_LIST = new ArrayList<>();
68            GOODS_LIST.add(new Goods("1", "小米12", new BigDecimal(4890)));
69            GOODS_LIST.add(new Goods("2", "iphone12", new BigDecimal(12000)));
70            GOODS_LIST.add(new Goods("3", "MacBookPro", new BigDecimal(15000)));
71            GOODS_LIST.add(new Goods("4", "Thinkpad X1", new BigDecimal(9800)));

```

```
72         GOODS_LIST.add(new Goods("5", "MeiZu One", new BigDecimal(3200)));
73         GOODS_LIST.add(new Goods("6", "Mate 40", new BigDecimal(6500)));
74     }
75
76     public static Goods randomGoods() {
77         int rIndex = r.nextInt(GOODS_LIST.size());
78         return GOODS_LIST.get(rIndex);
79     }
80
81     public Goods() {
82     }
83
84     public Goods(String goodsId, String goodsName, BigDecimal goodsPrice) {
85         this.goodsId = goodsId;
86         this.goodsName = goodsName;
87         this.goodsPrice = goodsPrice;
88     }
89
90     @Override
91     public String toString() {
92         return JSON.toJSONString(this);
93     }
94 }
95
96 //订单明细类
97 @Data
98 public static class OrderItem {
99     private String itemId;
100    private String goodsId;
101    private Integer count;
102
103    @Override
104    public String toString() {
105        return JSON.toJSONString(this);
106    }
107 }
108
109 //关联结果
110 @Data
```

```
111     public static class FactOrderItem {
112         private String goodsId;
113         private String goodsName;
114         private BigDecimal count;
115         private BigDecimal totalMoney;
116
117         @Override
118         public String toString() {
119             return JSON.toJSONString(this);
120         }
121     }
122
123     //构建一个商品Stream源（这个好比就是维表）
124     public static class GoodsSource11 extends RichSourceFunction<Goods> {
125         private Boolean isCancel;
126
127         @Override
128         public void open(Configuration parameters) throws Exception {
129             isCancel = false;
130         }
131
132         @Override
133         public void run(SourceContext sourceContext) throws Exception {
134             while (!isCancel) {
135                 Goods.GOODS_LIST.stream().forEach(goods ->
sourceContext.collect(goods));
136                 TimeUnit.SECONDS.sleep(1);
137             }
138         }
139
140         @Override
141         public void cancel() {
142             isCancel = true;
143         }
144     }
145
146     //构建订单明细Stream源
147     public static class OrderItemSource extends RichSourceFunction<OrderItem> {
148         private Boolean isCancel;
149         private Random r;
```

```

150
151     @Override
152     public void open(Configuration parameters) throws Exception {
153         isCancel = false;
154         r = new Random();
155     }
156
157     @Override
158     public void run(SourceContext sourceContext) throws Exception {
159         while (!isCancel) {
160             Goods goods = Goods.randomGoods();
161             OrderItem orderItem = new OrderItem();
162             orderItem.setGoodsId(goods.getGoodsId());
163             orderItem.setCount(r.nextInt(10) + 1);
164             orderItem.setItemId(UUID.randomUUID().toString());
165             sourceContext.collect(orderItem);
166             orderItem.setGoodsId("111");
167             sourceContext.collect(orderItem);
168             TimeUnit.SECONDS.sleep(1);
169         }
170     }
171
172     @Override
173     public void cancel() {
174         isCancel = true;
175     }
176 }
177
178 //构建水印分配器（此处为了简单），直接使用系统时间了
179 public static class GoodsWatermark implements WatermarkStrategy<Goods> {
180
181     @Override
182     public TimestampAssigner<Goods>
183     createTimestampAssigner(TimestampAssignerSupplier.Context context) {
184         return (element, recordTimestamp) -> System.currentTimeMillis();
185     }
186
187     @Override
188     public WatermarkGenerator<Goods>
189     createWatermarkGenerator(WatermarkGeneratorSupplier.Context context) {

```

```

188         return new WatermarkGenerator<Goods>() {
189             @Override
190             public void onEvent(Goods event, long eventTimestamp, WatermarkOutput
output) {
191                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
192             }
193
194             @Override
195             public void onPeriodicEmit(WatermarkOutput output) {
196                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
197             }
198         };
199     }
200 }
201
202 public static class OrderItemWatermark implements WatermarkStrategy<OrderItem> {
203     @Override
204     public TimestampAssigner<OrderItem>
createTimestampAssigner(TimestampAssignerSupplier.Context context) {
205         return (element, recordTimestamp) -> System.currentTimeMillis();
206     }
207
208     @Override
209     public WatermarkGenerator<OrderItem>
createWatermarkGenerator(WatermarkGeneratorSupplier.Context context) {
210         return new WatermarkGenerator<OrderItem>() {
211             @Override
212             public void onEvent(OrderItem event, long eventTimestamp,
WatermarkOutput output) {
213                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
214             }
215
216             @Override
217             public void onPeriodicEmit(WatermarkOutput output) {
218                 output.emitWatermark(new Watermark(System.currentTimeMillis()));
219             }
220         };
221     }
222 }
223 }
224

```

FlinkTableAPI和FlinkSQL

FlinkSQL

```
1 package sz.base.flink.cases;
2
3
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7 import org.apache.flink.api.java.tuple.Tuple2;
8 import org.apache.flink.streaming.api.datastream.DataStream;
9 import org.apache.flink.streaming.api.datastream.DataStreamSource;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11 import org.apache.flink.table.api.DataTypes;
12 import org.apache.flink.table.api.EnvironmentSettings;
13 import org.apache.flink.table.api.Schema;
14 import org.apache.flink.table.api.Table;
15 import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;
16 import org.apache.flink.types.Row;
17
18
19 import java.beans.Expression;
20
21 import static org.apache.flink.table.api.Expressions.$;
22
23
24 /**
25  * 案例：单词统计的案例，使用FlinkTable & FlinkSQL
26  * 输出表的样式：
27  * Hello | 2
28  * word | 1
29  *
30  * sql 的实现样式
31  * select word,sum(1) as frequency from WC group by word ;
32  *
33  * 总结：
```

```

34  * 数据流: DataStream
35  * 虚拟表: "t_words"
36  * Table对象 result
37  * Table对象 -> DataStream -> 打印输出
38  */
39 public class WordCountDemo {
40     public static void main(String[] args) throws Exception {
41         //1.获取流环境
42         StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
43         //环境设置
44         EnvironmentSettings settings = EnvironmentSettings.newInstance()
45             .useBlinkPlanner().inStreamingMode().build();
46         //设置流表环境
47         StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
48         //2.source获取 单词信息
49         DataStreamSource<WC> input = env.fromElements(new WC("hello", 1),
50             new WC("world", 1),
51             new WC("flink", 3),
52             new WC("hadoop", 2),
53             new WC("hello", 2));
54         //3.创建视图wordcount,参数: 表名 数据流 字段...
55         // tEnv.createTemporaryView("t_words",input,$("word"),$("frequency"));
56         //参数: 表名, 数据流, schema(构造者模式)
57         tEnv.createTemporaryView("t_words",input, Schema.newBuilder().column("word",
DataTypes.STRING()).column("frequency",DataTypes.BIGINT()).build() );
58         //4.执行查询, 单词统计
59         Table result = tEnv.sqlQuery("select word,sum(frequency) as cnt from t_words " +
60             "group by word");
61         //5.输出结果retractStream获取数据流
62         //打印当前表的表结构
63         result.printSchema();
64         //将Table对象转换成 DataStream 在输出
65         DataStream<Tuple2<Boolean, Row>> dataStream = tEnv.toRetractStream(result,
Row.class);
66         //打印输出结果
67         dataStream.printToErr();
68         //执行流环境
69         env.execute();
70     }
71

```



```

72     @Data
73     @AllArgsConstructor
74     @NoArgsConstructor
75     public static class WC{
76         public String word;
77         public long frequency;
78     }
79 }
80

```

FlinkTableAPI

```

1  package sz.base.flink.cases;
2
3
4  import lombok.AllArgsConstructor;
5  import lombok.Data;
6  import lombok.NoArgsConstructor;
7  import org.apache.flink.api.java.tuple.Tuple2;
8  import org.apache.flink.streaming.api.datastream.DataStream;
9  import org.apache.flink.streaming.api.datastream.DataStreamSource;
10 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
11 import org.apache.flink.table.api.DataTypes;
12 import org.apache.flink.table.api.EnvironmentSettings;
13 import org.apache.flink.table.api.Schema;
14 import org.apache.flink.table.api.Table;
15 import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;
16 import org.apache.flink.types.Row;
17
18 import static org.apache.flink.table.api.Expressions.$;
19
20
21 /**
22  * 案例：单词统计的案例，使用FlinkTable
23  * 输出表的样式：
24  * Hello | 2
25  * word | 1
26  *
27  * sql 的实现样式

```

```

28  * select word,sum(1) as frequency from WC group by word ;
29  *
30  * 总结:
31  * 数据流: DataStream -> 虚拟表 path
32  * 虚拟表: "t_words" -> Table对象
33  * Table API -> result
34  * Table对象 -> DataStream -> 打印输出
35  */
36 public class WordCountTableDemo {
37     public static void main(String[] args) throws Exception {
38         //1.获取流环境
39         StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
40         //环境设置
41         EnvironmentSettings settings = EnvironmentSettings.newInstance()
42             .useBlinkPlanner().inStreamingMode().build();
43         //设置流表环境
44         StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
45         //2.source获取 单词信息
46         DataSource<WC> input = env.fromElements(new WC("hello", 1),
47             new WC("world", 1),
48             new WC("flink", 3),
49             new WC("hadoop", 2),
50             new WC("hello", 2));
51         //3.创建视图wordcount,参数: 表名 数据流 字段...
52         // tEnv.createTemporaryView("t_words",input,$("word"),$("frequency"));
53         //参数: 表名, 数据流, schema(构造者模式)
54         tEnv.createTemporaryView("t_words",input, Schema.newBuilder().column("word",
DataTypes.STRING()).column("frequency",DataTypes.BIGINT()).build() );
55         //4.执行查询, 单词统计
56         //虚拟表转换成Table对象
57         Table words = tEnv.from("t_words");
58         //使用Table API 实现wordcount
59         Table result = words.groupBy($("word")).select($("word"),
$("frequency").sum().as("cnt"));
60         //5.输出结果retractStream获取数据流
61         //打印当前表的表结构
62
63         result.printSchema();
64         //将Table对象转换成 DataStream 在输出

```

```

65     DataStream<Tuple2<Boolean, Row>> dataStream = tEnv.toRetractStream(result,
    Row.class);
66     //打印输出结果
67     dataStream.printToErr();
68     //执行流环境
69     env.execute();
70 }
71
72 @Data
73 @AllArgsConstructor
74 @NoArgsConstructor
75 public static class WC{
76     public String word;
77     public long frequency;
78 }
79 }
80

```

输出到表

- 输出到文件系统

```

1  package sz.base.flinkconnector;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6  import org.apache.flink.api.common.functions.MapFunction;
7  import org.apache.flink.api.common.restartstrategy.RestartStrategies;
8  import org.apache.flink.api.java.io.TextInputFormat;
9  import org.apache.flink.streaming.api.datastream.DataStream;
10 import org.apache.flink.streaming.api.datastream.DataStreamSource;
11 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
12 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
13 import org.apache.flink.streaming.api.functions.source.FileProcessingMode;
14 import org.apache.flink.table.api.EnvironmentSettings;
15 import org.apache.flink.table.api.Table;
16 import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;
17
18 import static org.apache.flink.table.api.Expressions.$;

```

```

19
20 /**
21  * 需求- 将过滤出来的地区为北京地区的数据写出到文件系统 FileSystem 中
22  */
23 public class OrderSinkFileSystem {
24     public static void main(String[] args) throws Exception {
25
26         //创建流执行环境设置并行度
27         StreamExecutionEnvironment env =
28             StreamExecutionEnvironment.getExecutionEnvironment();
29
30         env.setParallelism(1);
31         //设置环境设置为blink计划器并且是流模式
32         EnvironmentSettings settings =
33             EnvironmentSettings.newInstance().useBlinkPlanner().inStreamingMode().build();
34
35         //谁次checkpoint
36         env.enableCheckpointing(10000);
37         //设置重启策略
38         env.setRestartStrategy(RestartStrategies.fixedDelayRestart(3, 3000));
39         //从 order.csv 读取文件数据源 读取为流数据
40         DataStreamSource<String> source = env.readFile(new TextInputFormat(null),
41             "input/order.csv", FileProcessingMode.PROCESS_CONTINUOUSLY, 60 * 1000);
42         //将字符串map转换切分转换成 OrderInfo 对象
43         SingleOutputStreamOperator<OrderInfo> map = source.map(new MapFunction<String,
44             OrderInfo>() {
45
46                 @Override
47                 public OrderInfo map(String s) throws Exception {
48                     String[] arr = s.split(",");
49                     OrderInfo orderInfo = new OrderInfo(arr[0]
50                         , Long.parseLong(arr[1]),
51                         arr[2],
52                         Double.parseDouble(arr[3])
53                         , arr[4]);
54                     return orderInfo;
55                 }
56             });
57         //创建表环境
58         StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
59         //将数据流转换成 Table 数据流, from只是把路径转换成table
60         Table orderTable = tEnv.fromDataStream(map, $("uid"), $("tms"), $("category"),
61             $("price"), $("areaName"));
62         //1. Flink Table api 语法筛选filter出区域为北京的所有字段数据

```

```

55  /*      Table result = orderTable.where(
56          $("areaName").isEqual("北京")
57      ).select($("uid"), $("tms"), $("category"), $("price"), $("areaName"));*/
58  //2. 将table创建临时视图
59  tEnv.createTemporaryView("t_order", orderTable);
60
61  //编写SQL查询获取北京的所有字段信息
62  Table result2Table = tEnv.sqlQuery("select * from t_order where areaName='北京'
");
63  //1.1将结果Table转换成数据流
64  //      DataStream<Tuple2<Boolean, OrderInfo>> result1 = tEnv.toRetractStream(result,
OrderInfo.class);
65  //2.1
66  DataStream<OrderInfo> result1 = tEnv.toAppendStream(result2Table,
OrderInfo.class);
67  //打印输出到文件系统 FileSystem 中
68  String sql = "CREATE TABLE t_order_result (" +
69      " uid STRING," +
70      " tms bigint," +
71      " category STRING," +
72      " price double," +
73      " areaName STRING" +
74      ") WITH( " +
75      " 'connector'='filesystem'," +
76      " 'path'='file:///D:/order'," +
77      " 'format'='csv'," +
78      " 'sink.rolling-policy.rollover-interval'='1 min' " +
79      ")";
80  //执行流环境
81  tEnv.executeSql(sql);
82  result2Table.executeInsert("t_order_result");
83  env.execute();
84
85  }
86
87  @Data
88  @AllArgsConstructor
89  @NoArgsConstructor
90  public static class OrderInfo {
91      private String uid;
92      private Long tms;

```

```
93     private String category;
94     private Double price;
95     private String areaName;
96 }
97 }
98
```

输出到kafka

```
1  package sz.base.flinkconnector;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5  import lombok.NoArgsConstructor;
6  import org.apache.flink.api.common.functions.MapFunction;
7  import org.apache.flink.api.common.restartstrategy.RestartStrategies;
8  import org.apache.flink.api.java.io.TextInputFormat;
9  import org.apache.flink.streaming.api.datastream.DataStream;
10 import org.apache.flink.streaming.api.datastream.DataStreamSource;
11 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;
12 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
13 import org.apache.flink.streaming.api.functions.source.FileProcessingMode;
14 import org.apache.flink.table.api.EnvironmentSettings;
15 import org.apache.flink.table.api.Table;
16 import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;
17
18 import static org.apache.flink.table.api.Expressions.$;
19
20 /**
21  * 需求 - 筛选出地区为北京的数据并将其写入kafka
22  */
23 public class OrderSinkFKafka {
24     public static void main(String[] args) throws Exception {
25
26         //创建流执行环境设置并行度
27         StreamExecutionEnvironment env =
28             StreamExecutionEnvironment.getExecutionEnvironment();
29         env.setParallelism(1);
30         //设置环境设置为blink计划器并且是流模式

```

```

30     EnvironmentSettings settings =
EnvironmentSettings.newInstance().useBlinkPlanner().inStreamingMode().build();
31     //谁次checkpoint
32     env.enableCheckpointing(10000);
33     //设置重启策略
34     env.setRestartStrategy(RestartStrategies.fixedDelayRestart(3, 3000));
35     //从 order.csv 读取文件数据源 读取为流数据
36     DataStreamSource<String> source = env.readFile(new TextInputFormat(null),
"input/order.csv", FileProcessingMode.PROCESS_CONTINUOUSLY, 60 * 1000);
37     //将字符串map转换切分转换成 OrderInfo 对象
38     SingleOutputStreamOperator<OrderSinkFileSystem.OrderInfo> map = source.map(new
MapFunction<String, OrderSinkFileSystem.OrderInfo>() {
39         @Override
40         public OrderSinkFileSystem.OrderInfo map(String s) throws Exception {
41             String[] arr = s.split(",");
42             OrderSinkFileSystem.OrderInfo orderInfo = new
OrderSinkFileSystem.OrderInfo(arr[0]
43                 , Long.parseLong(arr[1]),
44                 arr[2],
45                 Double.parseDouble(arr[3])
46                 , arr[4]);
47             return orderInfo;
48         }
49     });
50     //创建表环境
51     StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
52     //将数据流转换成 Table 数据流转table, from只是把路径转换成table
53     Table orderTable = tEnv.fromDataStream(map, $("uid"), $("tms"), $("category"),
$("price"), $("areaName"));
54     //1. Flink Table api 语法筛选filter出区域为北京的所有字段数据
55     /*
56         $("areaName").isEqual("北京")
57     ).select($("uid"), $("tms"), $("category"), $("price"), $("areaName"));*/
58     //2. 将table创建临时视图
59     tEnv.createTemporaryView("t_order", orderTable);
60
61     //编写SQL查询获取北京的所有字段信息
62     Table result2Table = tEnv.sqlQuery("select * from t_order where areaName='北京'
");
63     //1.1将结果Table转换成数据流
64     //
DataStream<Tuple2<Boolean, OrderInfo>> result1 = tEnv.toRetractStream(result,
OrderInfo.class);

```

```

65         //2.1
66         DataStream<OrderSinkFileSystem.OrderInfo> result1 =
tEnv.toAppendStream(result2Table, OrderSinkFileSystem.OrderInfo.class);
67         //打印输出到文件系统 FileSystem 中
68         String sql = "CREATE TABLE t_order_result (" +
69             " uid STRING," +
70             " tms bigint," +
71             " category STRING," +
72             " price double," +
73             " areaName STRING" +
74             ") WITH( " +
75             " 'connector'='kafka'," +
76             " 'topic'='output'," +
77             " 'properties.bootstrap.servers'='node1:9092,node2:9092,node3:9092' ," +
78             " 'format'='json' ," +
79             " 'scan.topic.partition-discovery.interval'='30000' ," +
80             " 'sink.semantic'='at-least-once' ," +
81             " 'scan.startup.mode'='latest-offset' ," +
82             " 'properties.group.id'='__consumer_output_' " +
83             ")";
84         //执行流环境
85         tEnv.executeSql(sql);
86         result2Table.executeInsert("t_order_result");
87         env.execute();
88
89     }
90
91     @Data
92     @AllArgsConstructor
93     @NoArgsConstructor
94     public static class OrderInfo {
95         private String uid;
96         private Long tms;
97         private String category;
98         private Double price;
99         private String areaName;
100     }
101 }
102

```


kafka输出到MySQL

```
1 package sz.base.flinkconnector;
2
3
4 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
5 import org.apache.flink.table.api.EnvironmentSettings;
6 import org.apache.flink.table.api.Table;
7 import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;
8 import org.apache.flink.types.Row;
9
10 /**
11  * 需求 - 将kafka中的订单的数据写入到MySQL中
12  * 开发步骤:
13  * 1.在MySQL中先创建数据库和数据表
14  * 2.创建流表环境
15  * 3.读取kafka中的数据源
16  * 4.执行kafka数据源SQL
17  * 5.写入MySQL的SQL并执行
18  * 6.实现 insert into 目标表 select 字段列表 from 源表
19  * 7.执行流表环境
20  */
21 public class OrderKafkaMySQL {
22
23     public static void main(String[] args) throws Exception {
24         // * 1.在MySQL中先创建数据库和数据表
25         StreamExecutionEnvironment env =
26             StreamExecutionEnvironment.getExecutionEnvironment();
27         // * 2.创建流表环境
28         EnvironmentSettings settings =
29             EnvironmentSettings.newInstance().useBlinkPlanner().inStreamingMode().build();
30         StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
31         // * 3.读取kafka中的数据源
32         String kafkaTable = "CREATE TABLE KafkaTable (" +
33             " `uid` STRING," +
34             " `tms` BIGINT," +
35             " `category` STRING," +
36             " `price` DOUBLE," +
37             " `areaName` STRING" +
```

```

36         ") WITH (" +
37         " 'connector'='kafka'," +
38         " 'topic'='output'," +
39         " 'properties.bootstrap.servers'='node1:9092,node2:9092,node3:9092' ," +
40         " 'format'='json' ," +
41         " 'scan.topic.partition-discovery.interval'='30000' ," +
42         " 'sink.semantic'='at-least-once' ," +
43         " 'scan.startup.mode'='earliest-offset' ," +
44         " 'properties.group.id'='__consumer_output_' " +
45         ");";
46 // * 4执行kafka数据源SQL,虚拟表就会被创建
47 tEnv.executeSql(kafkaTable);
48 //读取kafka中的数据并打印输出
49 Table table1 = tEnv.from("KafkaTable");
50 tEnv.toAppendStream(table1, Row.class).printToErr();
51 // * 5.写入MySQL的SQL并执行
52 String mysqlTable = "CREATE TABLE MyUserTable (" +
53         " uid STRING," +
54         " tms BIGINT," +
55         " category STRING," +
56         " price DOUBLE," +
57         " areaName STRING," +
58         " PRIMARY KEY (uid) NOT ENFORCED" +
59         ") WITH (" +
60         " 'connector' = 'jdbc'," +
61         " 'url' = 'jdbc:mysql://localhost:3306/test?
62 useSSL=false&characterEncoding=utf-8'," +
63         " 'username'='root'," +
64         " 'password'='root'," +
65         " 'sink.buffer-flush.max-rows'='1'," +
66         " 'sink.buffer-flush.interval'='1s'," +
67         " 'table-name' = 'order_test'" +
68         ");";
69 //执行落地表
70 tEnv.executeSql(mysqlTable);
71 // * 6.实现 insert into 目标表 select 字段列表 from 源表
72 tEnv.executeSql("INSERT INTO order_test select uid,tms,category,price,areaName
73 from KafkaTable");
74 // * 7.执行流表环境
75 env.execute();

```

```
74     }  
75 }  
76
```

• 分配水印

```
1  package sz.base.flik.sql;  
2  
3  import lombok.AllArgsConstructor;  
4  import lombok.Data;  
5  import lombok.NoArgsConstructor;  
6  import org.apache.calcite.avatica.com.google.protobuf.SourceContext;  
7  import org.apache.flink.api.common.eventtime.WatermarkStrategy;  
8  import org.apache.flink.api.common.restartstrategy.RestartStrategies;  
9  import org.apache.flink.api.java.tuple.Tuple2;  
10 import org.apache.flink.streaming.api.datastream.DataStream;  
11 import org.apache.flink.streaming.api.datastream.SingleOutputStreamOperator;  
12 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;  
13 import org.apache.flink.streaming.api.functions.source.RichSourceFunction;  
14 import org.apache.flink.table.api.EnvironmentSettings;  
15 import org.apache.flink.table.api.Table;  
16 import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;  
17 import org.apache.flink.types.Row;  
18  
19 import java.time.Duration;  
20 import java.util.Random;  
21 import java.util.UUID;  
22 import java.util.concurrent.TimeUnit;  
23  
24 import static org.apache.flink.table.api.Expressions.$;  
25 import static org.apache.flink.table.api.Expressions.e;  
26  
27 /**  
28  * 需求    - 实现一个订单需求案例  
29  * 随机生成一个订单并将其转换，根据事件时间进行分组，求出订单的和  
30  */  
31 public class OrderTime {  
32     public static void main(String[] args) throws Exception {  
33         //1.创建流执行环境和流表环境
```

```

34      StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
35      EnvironmentSettings settings =
EnvironmentSettings.newInstance().useBlinkPlanner().inStreamingMode().build();
36      //开启Checkpoint
37      env.enableCheckpointing(1000);
38      env.setParallelism(1);
39      //开启重启策略
40      env.setRestartStrategy(RestartStrategies.fixedDelayRestart(3,3000));
41      StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);
42      //2.source 自定义order 每1s 中睡眠一次
43      //3.Transformation 分配时间戳水印2s
44      SingleOutputStreamOperator<Order> source = env.addSource(new
MySource()).assignTimestampsAndWatermarks(WatermarkStrategy.
<Order>forBoundedOutOfOrderness(Duration.ofSeconds(3)).withTimestampAssigner((k, t) ->
k.createTime()));
45      //4.注册表, 创建临时视图并分配rowtime
46      tEnv.createTemporaryView("t_order",source,
47          $("orderId"),
48          $("userId"),
49          $("money"),
50          $("createTime").rowtime()//标记当前的字段是事件时间
51      );
52      //获取流表对象
53      //注册临时表
54      //5.编写SQL, 根据userid 和createTime 滚动分组统计userid、订单总笔数、最大、最小金额
55      Table result = tEnv.sqlQuery("select userId,count(orderId)as cnt , max(money)
as maxMoney, min(money) as minMoney" +
56          " from t_order" +
57          " group by userId,tumble(createTime,interval '5' second)");//根据userid和
时间5s分组
58      );
59      //6.执行查询语句返回结果,查询执行计划
60      System.out.println(result.explain());
61      //7.sink toRetractStream → 将计算后的新的数据在DataStream原数据的基础上更新true或是
删除false
62      DataStream<Tuple2<Boolean, Row>> tuple2DataStream = tEnv.toRetractStream(result,
Row.class);
63      tuple2DataStream.printToErr();
64      env.execute();
65
66  }
67  private static class MySource extends RichSourceFunction<Order> {

```

```
68         //循环条件
69         volatile boolean isRunning = true;
70
71         @Override
72         public void run(SourceContext<Order> ctx) throws Exception {
73             Random rm = new Random();
74             while (isRunning) {
75                 Order order = new Order();
76                 order.setOrderId(UUID.randomUUID().toString());
77                 order.setUserId(rm.nextInt(3));
78                 order.setMoney(rm.nextInt(101));
79                 //水印机制，模拟延迟的数据，随机三秒
80                 order.setCreateTime(System.currentTimeMillis() - rm.nextInt(3) * 1000);
81                 ctx.collect(order);
82                 //一秒一条
83                 TimeUnit.SECONDS.sleep(1);
84             }
85         }
86
87         @Override
88         public void cancel() {
89             isRunning = false;
90         }
91     }
92
93     @Data
94     @AllArgsConstructor
95     @NoArgsConstructor
96     public static class Order {
97         private String orderId;
98         private Integer userId;
99         private Integer money;
100        private Long createTime;
101    }
102 }
103
```