4.21 RocketMQ-Streams作业

1. 基于代码实例以及数据源，打印指定数据

数据源：

```
{"InFlow":"1","ProjectName":"ProjectName-0","LogStore":"LogStore-
0","OutFlow":"0"}
{"InFlow":"2","ProjectName":"ProjectName-1","LogStore":"LogStore-
1","OutFlow":"1"}
{"InFlow":"3","ProjectName":"ProjectName-2","LogStore":"LogStore-
2","OutFlow":"2"}
{"InFlow":"4","ProjectName":"ProjectName-0","LogStore":"LogStore-
0","OutFlow":"3"}
{"InFlow":"5","ProjectName":"ProjectName-1","LogStore":"LogStore-
1","OutFlow":"4"}
{"InFlow":"6","ProjectName":"ProjectName-2","LogStore":"LogStore-
2","OutFlow":"5"}
{"InFlow":"7","ProjectName":"ProjectName-0","LogStore":"LogStore-
0","OutFlow":"6"}
{"InFlow":"8","ProjectName":"ProjectName-1","LogStore":"LogStore-
1","OutFlow":"7"}
{"InFlow":"9","ProjectName":"ProjectName-2","LogStore":"LogStore-
2","OutFlow":"8"}
{"InFlow":"10","ProjectName":"ProjectName-0","LogStore":"LogStore-
0","OutFlow":"9"}
```

逻辑：

先读取数据源，然后转换json数据类型，最后过滤所需要的数据。

程序代码：

```
public class FileSourceExample {
    public static void main(String[] args) {
```

```
        DataStreamSource source = StreamBuilder.dataStream("namespace",
"pipeline");
        source.fromFile("data.txt", false)
                .flatMap(new FlatMapFunction<String, String>() {
                    @Override
                    public List<String> flatMap(String message) throws Exception
{
                        List<String> result = new ArrayList<>();
                        Data data = JSON.parseObject(message, Data.class);
                        if (Integer.parseInt(data.getInFlow()) > 4 &&
data.getLogStore().equals("LogStore-1")) {
                            result.add(data.getProjectName());
                        }
                        return result;
                    }
                })
                .toPrint(1)
                .start();
    }
}
```

2. 思考题：基于事件时间，如何处理乱序数据?

参考流计算处理框架Flink，

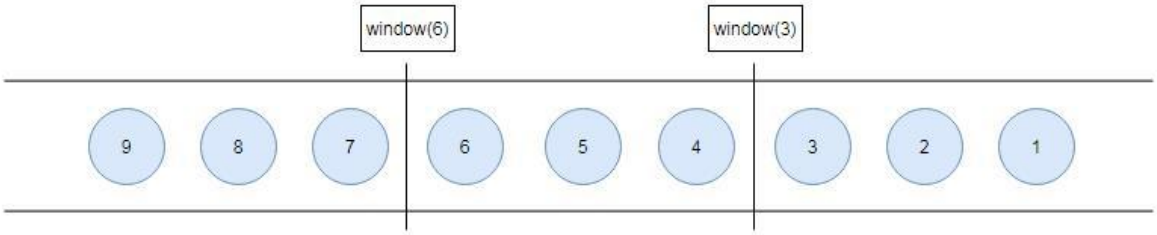
1. 首先可以引入watermark机制，

- Watermarks：可以把他理解为一个水位线，等于evevtTime - delay(比如规定为20分钟)，一旦Watermarks大于了某个window的end_time，就会触发此window的计算，Watermarks就是用来触发window计算的。

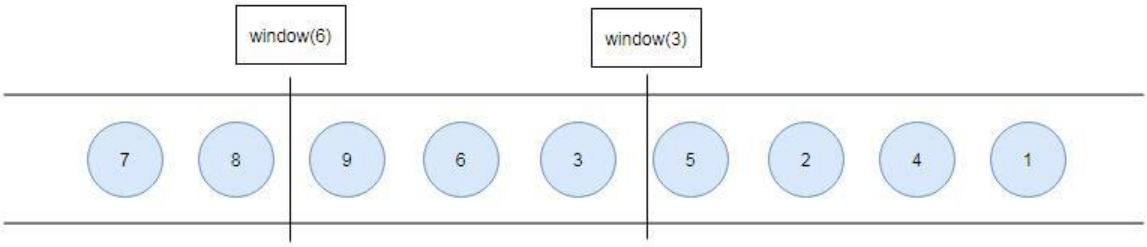
推迟窗口触发的时间，实现方式：通过当前窗口中最大的eventTime-延迟时间所得到的Watermark与窗口原始触发时间进行对比，当Watermark大于窗口原始触发时间时则触发窗口执行！！！我们知道，流处理从事件产生，到流经source，再到operator，中间是有一个过程和时间的，虽然大部分情况下，流到operator的数据都是按照事件产生的时间顺序来的，但是也不排除由于网络、分布式等原因，导致乱序的产生，所谓乱序，就是指Flink接收到的事件的先后顺序不是严格按照事件的Event Time顺序排列的。

那么此时出现一个问题，一旦出现乱序，如果只根据eventTime决定window的运行，我们不能明确数据是否全部到位，但又不能无限期的等下去，此时必须要有个机制来保证一个特定的时间后，必须触发window去进行计算了，这个特别的机制，就是Watermark。

Watermark是一种衡量Event Time进展的机制。 Watermark是用于处理乱序事件的，而正确的处理乱序事件，通常用Watermark机制结合window来实现。 数据流中的Watermark用于表示timestamp小于Watermark的数据，都已经到达了，因此，window的执行也是由Watermark触发的。 Watermark可以理解成一个延迟触发机制，我们可以设置Watermark的延时时长t，每次系统会校验已经到达的数据中最大的maxEventTime，然后认定eventTime小于maxEventTime - t的所有数据都已经到达，如果有窗口的停止时间等于maxEventTime – t，那么这个窗口被触发执行。 有序流的Watermarker如下图所示：（Watermark设置为0）



乱序流的Watermarker如下图所示：（Watermark设置为2）



当Flink接收到数据时，会按照一定的规则去生成Watermark，这条Watermark就等于当前所有到达数据中的maxEventTime - 延迟时长，也就是说，Watermark是由数据携带的，一旦数据携带的Watermark比当前未触发的窗口的停止时间要晚，那么就会触发相应窗口的执行。由于Watermark是由数据携带的，因此，如果运行过程中无法获取新的数据，那么没有被触发的窗口将永远都不被触发。

上图中，我们设置的**允许最大延迟到达时间为2s**，所以时间戳为5s的事件对应的Watermark是3s，时间戳为9s的事件的Watermark是7s，如果我们的窗口1是1s-3s，窗口2是4s-6s，那么时间戳为5s的事件到达时的Watermarker恰好触发窗口1，时间戳为9s的事件到达时的Watermark触发窗口2。

Watermark 就是触发前一窗口的“关窗时间”，一旦触发关门那么以当前时刻为准在窗口范围内的所有所有数据都会收入窗中。只要没有达到水位那么不管现实中的时间推进了多久都不会触发关窗。

2. watermark没有等到的数据可以使用`allowLateness`方法进行单独处理，每到一条新的迟到数据就再出发计算一次

3. `allowLateness`之后仍没有等到的乱序数据就把其加入测输出流，最后可以单独处理或者选择合并到主流。