# EventTime 和 Watermarks

针对数据乱序需求的案例分析，需要使用 eventtime 和 watermark 来解决

# 解释：

watermarks 的生成方式有两种
1：With Periodic Watermarks：周期性的触发 watermark 的生成和发送
2：With Periodic Watermarks：基于某些事件触发 watermark 的生成和发送

第一种方式比较常用，所以在这里我们使用第一种方式进行分析。

参考官网文档中 With Periodic Watermarks 的使用方法

```
/**
 * This generator generates watermarks assuming that elements arrive out of order,
 * but only to a certain degree. The latest elements for a certain timestamp t will arrive
 * at most n milliseconds after the earliest elements for timestamp t.
 */
public class BoundedOutOfOrdernessGenerator implements AssignerWithPeriodicWatermarks<MyEvent> {

    private final long maxOutOfOrderness = 3500; // 3.5 seconds

    private long currentMaxTimestamp;

    @Override
    public long extractTimestamp(MyEvent element, long previousElementTimestamp) {
        long timestamp = element.getCreationTime();
        currentMaxTimestamp = Math.max(timestamp, currentMaxTimestamp);
        return timestamp;
    }

    @Override
    public Watermark getCurrentWatermark() {
        // return the watermark as current highest timestamp minus the out-of-orderness bound
        return new Watermark(currentMaxTimestamp - maxOutOfOrderness);
    }
}
```

代码中的 extractTimestamp 方法是从数据本身中提取 EventTime
getCurrentWatermar 方法是获取当前水位线，利用 currentMaxTimestamp - maxOutOfOrderness
这里的 maxOutOfOrderness 表示是允许数据的最大乱序时间

所以在这里我们使用的话也实现接口 AssignerWithPeriodicWatermarks。

```
final StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

DataStream<MyEvent> stream = env.readFile(
        myFormat, myFilePath, FileProcessingMode.PROCESS_CONTINUOUSLY, 100,
        FilePathFilter.createDefaultFilter(), typeInfo);

DataStream<MyEvent> withTimestampsAndWatermarks = stream
        .filter( event -> event.severity() == WARNING )
        .assignTimestampsAndWatermarks(new MyTimestampsAndWatermarks());

withTimestampsAndWatermarks
        .keyBy( (event) -> event.getGroup() )
        .timeWindow(Time.seconds(10))
        .reduce( (a, b) -> a.add(b) )
        .addSink(...);
```

# 1：实现 watermark 相关代码

## 1.1：程序说明

从 socket 模拟接收数据，然后使用 map 进行处理，后面再调用 assignTimestampsAndWatermarks 方法抽取 timestamp 并生成 watermark。最后再调用 window 打印信息来验证 window 被触发的时机。

## 1.2：代码如下

```
package xuwei.tech.streaming.streamApiDemo;

import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.tuple.Tuple;
import org.apache.flink.api.java.tuple.Tuple2;
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.AssignerWithPeriodicWatermarks;
import org.apache.flink.streaming.api.functions.windowing.WindowFunction;
import org.apache.flink.streaming.api.watermark.Watermark;
import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;

import javax.annotation.Nullable;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
```

```java
/**
 *
 * Watermark 案例
 *
 * Created by xuwei.tech.
 */
public class StreamingWindowWatermark {

    public static void main(String[] args) throws Exception {
        //定义socket的端口号
        int port = 9000;
        //获取运行环境
        StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

        //设置使用eventtime，默认是使用processtime
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

        //设置并行度为1,默认并行度是当前机器的cpu数量
        env.setParallelism(1);

        //连接socket获取输入的数据
        DataStream<String> text = env.socketTextStream( hostname: "hadoop100", port,  delimiter: "\n");

        //解析输入的数据
        DataStream<Tuple2<String, Long>> inputMap = text.map(new MapFunction<String, Tuple2<String, Long>>() {
            @Override
            public Tuple2<String, Long> map(String value) throws Exception {
                String[] arr = value.split( regex: ",");
                return new Tuple2<>(arr[0], Long.parseLong(arr[1]));
            }
        });
```

```java
//抽取timestamp和生成watermark
DataStream<Tuple2<String, Long>> waterMarkStream = inputMap.assignTimestampsAndWatermarks(new AssignerWithPeriodicWatermarks<Tuple2<String, Long>>() {

    Long currentMaxTimestamp = 0L;
    final Long maxOutOfOrderness = 10000L;// 最大允许的乱序时间是10s

    SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss.SSS");
    /**
     * 定义生成watermark的逻辑
     * 默认100ms被调用一次
     */
    @Nullable
    @Override
    public Watermark getCurrentWatermark() {
        return new Watermark( timestamp: currentMaxTimestamp - maxOutOfOrderness);
    }

    //定义如何提取timestamp
    @Override
    public long extractTimestamp(Tuple2<String, Long> element, long previousElementTimestamp) {
        long timestamp = element.f1;
        currentMaxTimestamp = Math.max(timestamp, currentMaxTimestamp);
        System.out.println("key:"+element.f0+",eventtime:["+element.f1+"|"+sdf.format(element.f1)+"],currentMaxTimestamp:["+currentMaxTimestamp+"|"+
                sdf.format(currentMaxTimestamp)+"],watermark:["+getCurrentWatermark().getTimestamp()+"|"+sdf.format(getCurrentWatermark().getTimestamp())+"]");
        return timestamp;
    }
});
```

```java
//分组，聚合
DataStream<String> window = waterMarkStream.keyBy( ...fields: 0)
        .window(TumblingEventTimeWindows.of(Time.seconds(3)))//按照消息的EventTime分配窗口，和调用TimeWindow效果一样
        .apply(new WindowFunction<Tuple2<String, Long>, String, Tuple, TimeWindow>() {
            /**
             * 对window内的数据进行排序，保证数据的顺序
             * @param tuple
             * @param window
             * @param input
             * @param out
             * @throws Exception
             */
            @Override
            public void apply(Tuple tuple, TimeWindow window, Iterable<Tuple2<String, Long>> input, Collector<String> out) throws Exception {
                String key = tuple.toString();
                List<Long> arrarList = new ArrayList<Long>();
                Iterator<Tuple2<String, Long>> it = input.iterator();
                while (it.hasNext()) {
                    Tuple2<String, Long> next = it.next();
                    arrarList.add(next.f1);
                }
                Collections.sort(arrarList);
                SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss.SSS");
                String result = key + "," + arrarList.size() + "," + sdf.format(arrarList.get(0)) + "," + sdf.format(arrarList.get(arrarList.size() - 1))
                        + "," + sdf.format(window.getStart()) + "," + sdf.format(window.getEnd());
                out.collect(result);
            }
        });
//测试-把结果打印到控制台即可
window.print();

//注意，因为flink是懒加载的，所以必须调用execute方法，上面的代码才会执行
env.execute( jobName: "eventtime-watermark");
```

完整代码如下：

```
package xuwei.tech.streaming.streamApiDemo;


import org.apache.flink.api.common.functions.MapFunction;
import org.apache.flink.api.java.tuple.Tuple;
import org.apache.flink.api.java.tuple.Tuple2;
```

```
import org.apache.flink.streaming.api.TimeCharacteristic;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.api.functions.AssignerWithPeriodicWatermarks;
import org.apache.flink.streaming.api.functions.windowing.WindowFunction;
import org.apache.flink.streaming.api.watermark.Watermark;
import org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;
import org.apache.flink.streaming.api.windowing.time.Time;
import org.apache.flink.streaming.api.windowing.windows.TimeWindow;
import org.apache.flink.util.Collector;

import javax.annotation.Nullable;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;


/**
 *
 * Watermark 案例
 *
 * Created by xuwei.tech.
 */
public class StreamingWindowWatermark {

    public static void main(String[] args) throws Exception {
        //定义 socket 的端口号
        int port = 9000;
        //获取运行环境
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();

        //设置使用 eventtime，默认是使用 processtime
        env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);

        //设置并行度为 1,默认并行度是当前机器的 cpu 数量
        env.setParallelism(1);

        //连接 socket 获取输入的数据
        DataStream<String> text = env.socketTextStream("hadoop100", port, "\n");

        //解析输入的数据
```

```
        DataStream<Tuple2<String, Long>> inputMap = text.map(new MapFunction<String, Tuple2<String, Long>>() {
            @Override
            public Tuple2<String, Long> map(String value) throws Exception {
                String[] arr = value.split(",");
                return new Tuple2<>(arr[0], Long.parseLong(arr[1]));
            }
        });

        //抽取 timestamp 和生成 watermark
        DataStream<Tuple2<String, Long>> waterMarkStream = inputMap.assignTimestampsAndWatermarks(new AssignerWithPeriodicWatermarks<Tuple2<String, Long>>() {

            Long currentMaxTimestamp = 0L;
            final Long maxOutOfOrderness = 10000L;// 最大允许的乱序时间是 10s

            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
            /**
             * 定义生成 watermark 的逻辑
             * 默认 100ms 被调用一次
             */
            @Nullable
            @Override
            public Watermark getCurrentWatermark() {
                return new Watermark(currentMaxTimestamp - maxOutOfOrderness);
            }

            //定义如何提取 timestamp
            @Override
            public long extractTimestamp(Tuple2<String, Long> element, long previousElementTimestamp) {
                long timestamp = element.f1;
                currentMaxTimestamp = Math.max(timestamp, currentMaxTimestamp);

System.out.println("key:"+element.f0+",eventtime:["+element.f1+"|"+sdf.format(element.f1)+"], currentMaxTimestamp:["+currentMaxTimestamp+"|"+

sdf.format(currentMaxTimestamp)+"],watermark:["+getCurrentWatermark().getTimestamp()+"|"+sdf.format(getCurrentWatermark().getTimestamp())+"]");
                return timestamp;
            }
        });
```

```
        //分组，聚合
        DataStream<String> window = waterMarkStream.keyBy(0)
                .window(TumblingEventTimeWindows.of(Time.seconds(3)))// 按照消息的EventTime 分配窗口，和调用 TimeWindow 效果一样
                .apply(new WindowFunction<Tuple2<String, Long>, String, Tuple, TimeWindow>() {
                    /**
                     * 对 window 内的数据进行排序，保证数据的顺序
                     * @param tuple
                     * @param window
                     * @param input
                     * @param out
                     * @throws Exception
                     */
                    @Override
                    public void apply(Tuple tuple, TimeWindow window, Iterable<Tuple2<String, Long>> input, Collector<String> out) throws Exception {
                        String key = tuple.toString();
                        List<Long> arrarList = new ArrayList<Long>();
                        Iterator<Tuple2<String, Long>> it = input.iterator();
                        while (it.hasNext()) {
                            Tuple2<String, Long> next = it.next();
                            arrarList.add(next.f1);
                        }
                        Collections.sort(arrarList);
                        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SSS");
                        String result = key + "," + arrarList.size() + "," + sdf.format(arrarList.get(0)) + "," + sdf.format(arrarList.get(arrarList.size() - 1))
                                + "," + sdf.format(window.getStart()) + "," + sdf.format(window.getEnd());
                        out.collect(result);
                    }
                });
        //测试-把结果打印到控制台即可
        window.print();

        //注意：因为 flink 是懒加载的，所以必须调用 execute 方法，上面的代码才会执行
        env.execute("eventtime-watermark");

    }
```

```
}
```

## 1.3：程序详解

(1) 接收 socket 数据
(2) 将每行数据按照逗号分隔，每行数据调用 map 转换成 tuple<String,Long>类型。其中 tuple 中的第一个元素代表具体的数据，第二个元素代表数据的 eventtime
(3) 抽取 timestamp，生成 watermar，允许的最大乱序时间是 10s，并打印（key,eventtime,currentMaxTimestamp,watermark）等信息
(4) 分组聚合，window 窗口大小为 3 秒，输出（key，窗口内元素个数，窗口内最早元素的时间，窗口内最晚元素的时间，窗口自身开始时间，窗口自身结束时间）

# 2：通过数据跟踪 watermark 的时间

在这里重点查看 watermark 和 timestamp 的时间，通过数据的输出来确定 window 的触发时机

首先我们开启 socket，输入第一条数据

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
```

输出的内容如下：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
```

为了查看方便，我们把输入内容汇总到表格中

| Key | Event Time | CurrentMaxTimeStamp | WaterMark |
|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 |

此时，wartermark 的时间，已经落后于 currentMaxTimestamp10 秒了。我们继续输入

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
```

此时，输入内容如下：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
key:0001, eventtime:[1538359886000|2018-10-01 10:11:26.000], currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000], watermark:[1538359876000|2018-10-01 10:11:16.000]
```

我们再次汇总，如下表：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark |
|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 |

继续输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
```

输出内容如下：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
key:0001, eventtime:[1538359886000|2018-10-01 10:11:26.000], currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000], watermark:[1538359876000|2018-10-01 10:11:16.000]
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359892000|2018-10-01 10:11:32.000], watermark:[1538359882000|2018-10-01 10:11:22.000]
```

汇总如下：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark |
|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 |
| 0001 | 1538359892000 | 1538359892000 | 1538359882000 |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:22.000 |

到这里，window 仍然没有被触发，此时 watermark 的时间已经等于了第一条数据的 Event Time 了。那么 window 到底什么时候被触发呢？我们再次输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
0001,1538359893000
```

输出内容如下：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
key:0001, eventtime:[1538359886000|2018-10-01 10:11:26.000], currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000], watermark:[1538359876000|2018-10-01 10:11:16.000]
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359892000|2018-10-01 10:11:32.000], watermark:[1538359882000|2018-10-01 10:11:22.000]
key:0001, eventtime:[1538359893000|2018-10-01 10:11:33.000], currentMaxTimestamp:[1538359893000|2018-10-01 10:11:33.000], watermark:[1538359883000|2018-10-01 10:11:23.000]
```

汇总如下：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark |
|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 |
| 0001 | 1538359892000 | 1538359892000 | 1538359882000 |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:22.000 |

| | | | |
|---|---|---|---|
| 0001 | 1538359893000 | 1538359893000 | 1538359883000 |
| | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:23.000 |

window 仍然没有触发，此时，我们的数据已经发到 2018-10-01 10:11:33.000 了，根据 eventtime 来算，最早的数据已经过去了 11 秒了，window 还没有开始计算，那到底什么时候会触发 window 呢？

我们再次增加 1 秒，输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
0001,1538359893000
0001,1538359894000
```

输出：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
key:0001, eventtime:[1538359886000|2018-10-01 10:11:26.000], currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000], watermark:[1538359876000|2018-10-01 10:11:16.000]
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359892000|2018-10-01 10:11:32.000], watermark:[1538359882000|2018-10-01 10:11:22.000]
key:0001, eventtime:[1538359893000|2018-10-01 10:11:33.000], currentMaxTimestamp:[1538359893000|2018-10-01 10:11:33.000], watermark:[1538359883000|2018-10-01 10:11:23.000]
key:0001, eventtime:[1538359894000|2018-10-01 10:11:34.000], currentMaxTimestamp:[1538359894000|2018-10-01 10:11:34.000], watermark:[1538359884000|2018-10-01 10:11:24.000]
(0001),1,2018-10-01 10:11:22.000,2018-10-01 10:11:22.000,2018-10-01 10:11:21.000,2018-10-01 10:11:24.000
```

汇总如下：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 | | |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 | | |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 | | |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 | | |
| 0001 | 1538359892000 | 1538359892000 | 1538359882000 | | |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:22.000 | | |
| 0001 | 1538359893000 | 1538359893000 | 1538359883000 | | |
| | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:23.000 | | |
| 0001 | 1538359894000 | 1538359894000 | 1538359884000 | | |
| | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:24.000 | [10:11:21.000 | 10:11:24.000) |

到这里，我们做一个说明：

window 的触发机制，是先按照自然时间将 window 划分，如果 window 大小是 3 秒，那么 1 分钟内会把 window 划分为如下的形式【左闭右开】：

| |
|---|
| [00:00:00,00:00:03) |

[00:00:03,00:00:06)
[00:00:06,00:00:09)
[00:00:09,00:00:12)
[00:00:12,00:00:15)
[00:00:15,00:00:18)
[00:00:18,00:00:21)
[00:00:21,00:00:24)
[00:00:24,00:00:27)
[00:00:27,00:00:30)
[00:00:30,00:00:33)
[00:00:33,00:00:36)
[00:00:36,00:00:39)
[00:00:39,00:00:42)
[00:00:42,00:00:45)
[00:00:45,00:00:48)
[00:00:48,00:00:51)
[00:00:51,00:00:54)
[00:00:54,00:00:57)
[00:00:57,00:01:00)
...

window 的设定无关数据本身，而是系统定义好了的。

输入的数据中，根据自身的 Event Time，将数据划分到不同的 window 中，如果 window 中有数据，则当 watermark 时间>=Event Time 时，就符合了 window 触发的条件了，最终决定 window 触发，还是由数据本身的 Event Time 所属的 window 中的 window_end_time 决定。

上面的测试中，最后一条数据到达后，其水位线已经升至 10:11:24 秒，正好是最早的一条记录所在 window 的 window_end_time，所以 window 就被触发了。

为了验证 window 的触发机制，我们继续输入数据:

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
0001,1538359893000
0001,1538359894000
0001,1538359896000
```

输出：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
key:0001, eventtime:[1538359886000|2018-10-01 10:11:26.000], currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000], watermark:[1538359876000|2018-10-01 10:11:16.000]
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359892000|2018-10-01 10:11:32.000], watermark:[1538359882000|2018-10-01 10:11:22.000]
key:0001, eventtime:[1538359893000|2018-10-01 10:11:33.000], currentMaxTimestamp:[1538359893000|2018-10-01 10:11:33.000], watermark:[1538359883000|2018-10-01 10:11:23.000]
key:0001, eventtime:[1538359894000|2018-10-01 10:11:34.000], currentMaxTimestamp:[1538359894000|2018-10-01 10:11:34.000], watermark:[1538359884000|2018-10-01 10:11:24.000]
(0001),1,2018-10-01 10:11:22.000,2018-10-01 10:11:22.000,2018-10-01 10:11:21.000,2018-10-01 10:11:24.000
key:0001, eventtime:[1538359896000|2018-10-01 10:11:36.000], currentMaxTimestamp:[1538359896000|2018-10-01 10:11:36.000], watermark:[1538359886000|2018-10-01 10:11:26.000]
```

汇总如下：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 | | |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 | | |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 | | |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 | | |
| 0001 | 1538359892000 | 1538359892000 | 1538359882000 | | |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:22.000 | | |
| 0001 | 1538359893000 | 1538359893000 | 1538359883000 | | |
| | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:23.000 | | |
| 0001 | 1538359894000 | 1538359894000 | 1538359884000 | | |
| | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:24.000 | [10:11:21.000 | 10:11:24.000) |
| 0001 | 1538359896000 | 1538359896000 | 1538359886000 | | |
| | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:26.000 | | |

此时，watermark 时间虽然已经达到了第二条数据的时间，但是由于其没有达到第二条数据所在 window 的结束时间，所以 window 并没有被触发。那么，第二条数据所在的 window 时间是:

| [00:00:24,00:00:27) |
|---|

也就是说，我们必须输入一个 10:11:27 秒的数据，第二条数据所在的 window 才会被触发。我们继续输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
0001,1538359893000
0001,1538359894000
0001,1538359896000
0001,1538359897000
```

输出：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
key:0001, eventtime:[1538359886000|2018-10-01 10:11:26.000], currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000], watermark:[1538359876000|2018-10-01 10:11:16.000]
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359892000|2018-10-01 10:11:32.000], watermark:[1538359882000|2018-10-01 10:11:22.000]
key:0001, eventtime:[1538359893000|2018-10-01 10:11:33.000], currentMaxTimestamp:[1538359893000|2018-10-01 10:11:33.000], watermark:[1538359883000|2018-10-01 10:11:23.000]
key:0001, eventtime:[1538359894000|2018-10-01 10:11:34.000], currentMaxTimestamp:[1538359894000|2018-10-01 10:11:34.000], watermark:[1538359884000|2018-10-01 10:11:24.000]
(0001),1,2018-10-01 10:11:22.000,2018-10-01 10:11:22.000,2018-10-01 10:11:21.000,2018-10-01 10:11:24.000
key:0001, eventtime:[1538359896000|2018-10-01 10:11:36.000], currentMaxTimestamp:[1538359896000|2018-10-01 10:11:36.000], watermark:[1538359886000|2018-10-01 10:11:26.000]
key:0001, eventtime:[1538359897000|2018-10-01 10:11:37.000], currentMaxTimestamp:[1538359897000|2018-10-01 10:11:37.000], watermark:[1538359887000|2018-10-01 10:11:27.000]
(0001),1,2018-10-01 10:11:26.000,2018-10-01 10:11:26.000,2018-10-01 10:11:24.000,2018-10-01 10:11:27.000
```

汇总如下：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 | | |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 | | |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 | | |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 | | |
| 0001 | 1538359892000 | 1538359892000 | 1538359882000 | | |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:22.000 | | |
| 0001 | 1538359893000 | 1538359893000 | 1538359883000 | | |
| | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:23.000 | | |
| 0001 | 1538359894000 | 1538359894000 | 1538359884000 | | |
| | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:24.000 | [10:11:21.000 | 10:11:24.000) |
| 0001 | 1538359896000 | 1538359896000 | 1538359886000 | | |
| | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:26.000 | | |
| 0001 | 1538359897000 | 1538359897000 | 1538359887000 | | |
| | 2018-10-01 10:11:37.000 | 2018-10-01 10:11:37.000 | 2018-10-01 10:11:27.000 | [10:11:24.000 | 10:11:27.000) |

此时，我们已经看到，window 的触发要符合以下几个条件：

1、watermark 时间 >= window_end_time
2、在[window_start_time,window_end_time)区间中有数据存在，注意是左闭右开的区间

同时满足了以上 2 个条件，window 才会触发。

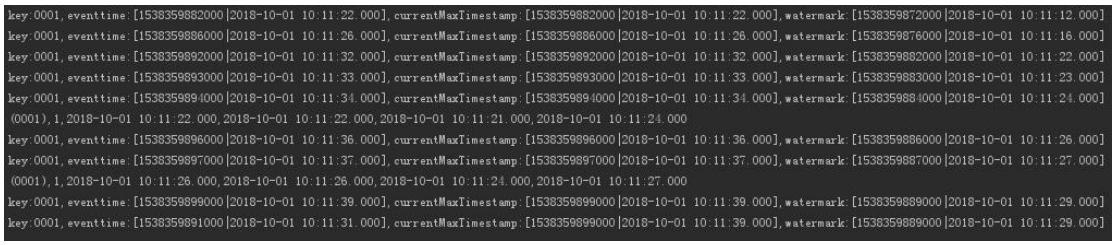
# 3：watermark+window 处理乱序数据

我们上面的测试，数据都是按照时间顺序递增的，现在，我们输入一些乱序的（late）数据，看看 watermark 结合 window 机制，是如何处理乱序的。

输入两行数据：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
0001,1538359893000
0001,1538359894000
0001,1538359896000
0001,1538359897000
0001,1538359899000
0001,1538359891000
```

输出：




汇总如下：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 | | |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 | | |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 | | |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 | | |
| 0001 | 1538359892000 | 1538359892000 | 1538359882000 | | |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:22.000 | | |
| 0001 | 1538359893000 | 1538359893000 | 1538359883000 | | |
| | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:23.000 | | |
| 0001 | 1538359894000 | 1538359894000 | 1538359884000 | | |
| | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:24.000 | [10:11:21.000 | 10:11:24.000) |
| 0001 | 1538359896000 | 1538359896000 | 1538359886000 | | |
| | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:26.000 | | |
| 0001 | 1538359897000 | 1538359897000 | 1538359887000 | | |
| | 2018-10-01 10:11:37.000 | 2018-10-01 10:11:37.000 | 2018-10-01 10:11:27.000 | [10:11:24.000 | 10:11:27.000) |
| 0001 | 1538359899000 | 1538359899000 | 1538359889000 | | |
| | 2018-10-01 10:11:39.000 | 2018-10-01 10:11:39.000 | 2018-10-01 10:11:29.000 | | |

| | | | | | |
|---|---|---|---|---|---|
| 0001 | 1538359891000 | 1538359899000 | 1538359889000 | | |
| | 2018-10-01 10:11:31.000 | 2018-10-01 10:11:39.000 | 2018-10-01 10:11:29.000 | | |

可以看到，虽然我们输入了一个 10:11:31 的数据，但是 currentMaxTimestamp 和 watermark 都没变。此时，按照我们上面提到的公式：

1、watermark 时间 >= window_end_time
2、在[window_start_time,window_end_time)中有数据存在

watermark 时间（10:11:29） < window_end_time（10:11:33），因此不能触发 window。

那如果我们再次输入一条 10:11:43 的数据，此时 watermark 时间会升高到 10:11:33，这时的 window 一定就会触发了，我们试一试：

输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
0001,1538359893000
0001,1538359894000
0001,1538359896000
0001,1538359897000
0001,1538359899000
0001,1538359891000
0001,1538359903000
```

输出：

```
key:0001, eventtime:[1538359882000|2018-10-01 10:11:22.000], currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000], watermark:[1538359872000|2018-10-01 10:11:12.000]
key:0001, eventtime:[1538359886000|2018-10-01 10:11:26.000], currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000], watermark:[1538359876000|2018-10-01 10:11:16.000]
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359892000|2018-10-01 10:11:32.000], watermark:[1538359882000|2018-10-01 10:11:22.000]
key:0001, eventtime:[1538359893000|2018-10-01 10:11:33.000], currentMaxTimestamp:[1538359893000|2018-10-01 10:11:33.000], watermark:[1538359883000|2018-10-01 10:11:23.000]
key:0001, eventtime:[1538359894000|2018-10-01 10:11:34.000], currentMaxTimestamp:[1538359894000|2018-10-01 10:11:34.000], watermark:[1538359884000|2018-10-01 10:11:24.000]
(0001),1,2018-10-01 10:11:22.000,2018-10-01 10:11:22.000,2018-10-01 10:11:21.000,2018-10-01 10:11:24.000
key:0001, eventtime:[1538359896000|2018-10-01 10:11:36.000], currentMaxTimestamp:[1538359896000|2018-10-01 10:11:36.000], watermark:[1538359886000|2018-10-01 10:11:26.000]
key:0001, eventtime:[1538359897000|2018-10-01 10:11:37.000], currentMaxTimestamp:[1538359897000|2018-10-01 10:11:37.000], watermark:[1538359887000|2018-10-01 10:11:27.000]
(0001),1,2018-10-01 10:11:26.000,2018-10-01 10:11:26.000,2018-10-01 10:11:24.000,2018-10-01 10:11:27.000
key:0001, eventtime:[1538359899000|2018-10-01 10:11:39.000], currentMaxTimestamp:[1538359899000|2018-10-01 10:11:39.000], watermark:[1538359889000|2018-10-01 10:11:29.000]
key:0001, eventtime:[1538359891000|2018-10-01 10:11:31.000], currentMaxTimestamp:[1538359899000|2018-10-01 10:11:39.000], watermark:[1538359889000|2018-10-01 10:11:29.000]
key:0001, eventtime:[1538359903000|2018-10-01 10:11:43.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),2,2018-10-01 10:11:31.000,2018-10-01 10:11:32.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
```

汇总如下：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359882000 | 1538359882000 | 1538359872000 | | |
| | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:22.000 | 2018-10-01 10:11:12.000 | | |
| 0001 | 1538359886000 | 1538359886000 | 1538359876000 | | |
| | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:26.000 | 2018-10-01 10:11:16.000 | | |
| 0001 | 1538359892000 | 1538359892000 | 1538359882000 | | |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:22.000 | | |

| | | | | | |
|---|---|---|---|---|---|
| 0001 | 1538359893000 | 1538359893000 | 1538359883000 | | |
| | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:33.000 | 2018-10-01 10:11:23.000 | | |
| 0001 | 1538359894000 | 1538359894000 | 1538359884000 | | |
| | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:34.000 | 2018-10-01 10:11:24.000 | [10:11:21.000 | 10:11:24.000) |
| 0001 | 1538359896000 | 1538359896000 | 1538359886000 | | |
| | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:36.000 | 2018-10-01 10:11:26.000 | | |
| 0001 | 1538359897000 | 1538359897000 | 1538359887000 | | |
| | 2018-10-01 10:11:37.000 | 2018-10-01 10:11:37.000 | 2018-10-01 10:11:27.000 | [10:11:24.000 | 10:11:27.000) |
| 0001 | 1538359899000 | 1538359899000 | 1538359889000 | | |
| | 2018-10-01 10:11:39.000 | 2018-10-01 10:11:39.000 | 2018-10-01 10:11:29.000 | | |
| 0001 | 1538359891000 | 1538359899000 | 1538359889000 | | |
| | 2018-10-01 10:11:31.000 | 2018-10-01 10:11:39.000 | 2018-10-01 10:11:29.000 | | |
| 0001 | 1538359903000 | 1538359903000 | 1538359893000 | | |
| | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:33.000 | [10:11:30.000 | 10:11:33.000) |

这里，我们可以看到，窗口中有 2 个数据，10:11:31 和 10:11:32，但是没有 10:11:33 的数据，原因是窗口是一个前闭后开的区间，10:11:33 的数据是属于[10:11:33,10:11:36)的窗口的。

上边的结果，已经表明，对于 out-of-order 的数据，Flink 可以通过 watermark 机制结合 window 的操作，来处理一定范围内的乱序数据。那么对于“迟到(late element)”太多的数据，Flink 是怎么处理的呢？

# 4：late element(延迟数据)的处理

延迟数据三种处理方案

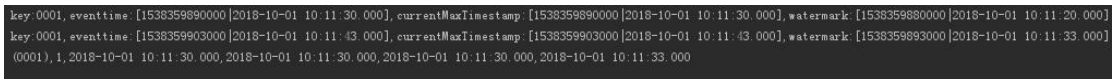
## 4.1：丢弃(默认)

我们输入一个乱序很多的（其实只要 Event Time < watermark 时间）数据来测试下：

输入：【输入两条内容】

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
```

输出：




汇总如下：

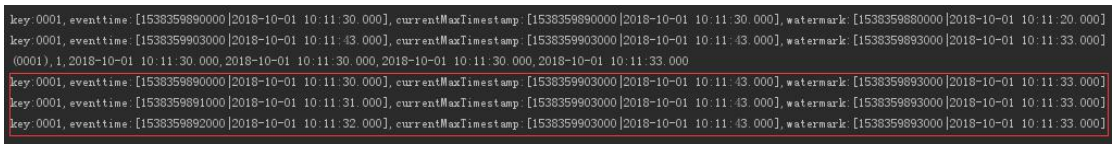
| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359890000<br>2018-10-01 10:11:30.000 | 1538359890000<br>2018-10-01 10:11:30.000 | 1538359880000<br>2018-10-01 10:11:20.000 | | |
| 0001 | 1538359903000<br>2018-10-01 10:11:43.000 | 1538359903000<br>2018-10-01 10:11:43.000 | 1538359893000<br>2018-10-01 10:11:33.000 | [10:11:30.000 | 10:11:33.000) |

注意：此时 watermark 是 2018-10-01 10:11:33.000

下面我们再输入几个 eventtime 小于 watermark 的时间
输入：【输入了三行内容】

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359890000
0001,1538359891000
0001,1538359892000
```

输出：




注意：此时并没有触发 window。因为输入的数据所在的窗口已经执行过了，flink 默认对这些迟到的数据的处理方案就是丢弃。

# 4.2：allowedLateness 指定允许数据延迟的时间

在某些情况下，我们希望对迟到的数据再提供一个宽容的时间。
Flink 提供了 allowedLateness 方法可以实现对迟到的数据设置一个延迟时间，在指定延迟时间内到达的数据还是可以触发 window 执行的。

修改代码：

```
//分组，聚合
DataStream<String> window = waterMarkStream.keyBy( ...fields: 0)
        .window(TumblingEventTimeWindows.of(Time.seconds(3)))//按照消息的EventTime分配窗口，和调用
        .allowedLateness(Time.seconds(2))//允许数据迟到2秒
        .apply(new WindowFunction<Tuple2<String, Long>, String, Tuple, TimeWindow>() {
            /**
             * 对window内的数据进行排序，保证数据的顺序
             * @param tuple
             * @param window
```

下面我们来验证一下：

输入：【输入两行内容】

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
```

输出：

```
key:0001, eventtime:[1538359890000|2018-10-01 10:11:30.000], currentMaxTimestamp:[1538359890000|2018-10-01 10:11:30.000], watermark:[1538359880000|2018-10-01 10:11:20.000]
key:0001, eventtime:[1538359903000|2018-10-01 10:11:43.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),1,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
```

正常触发 window，没什么问题。

汇总：

| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359890000 | 1538359890000 | 1538359880000 | | |
| | 2018-10-01 10:11:30.000 | 2018-10-01 10:11:30.000 | 2018-10-01 10:11:20.000 | | |
| 0001 | 1538359903000 | 1538359903000 | 1538359893000 | | |
| | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |

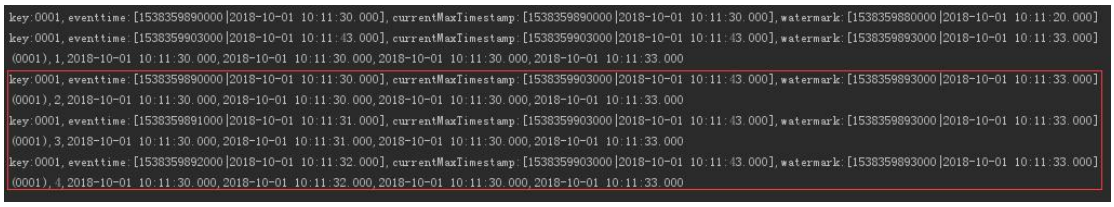
此时 watermark 是 2018-10-01 10:11:33.000

那么现在我们输入几条 eventtime<watermark 的数据验证一下效果

输入：【输入三行内容】

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359890000
0001,1538359891000
0001,1538359892000
```

输出：




在这里看到每条数据都触发了 window 执行。

汇总：

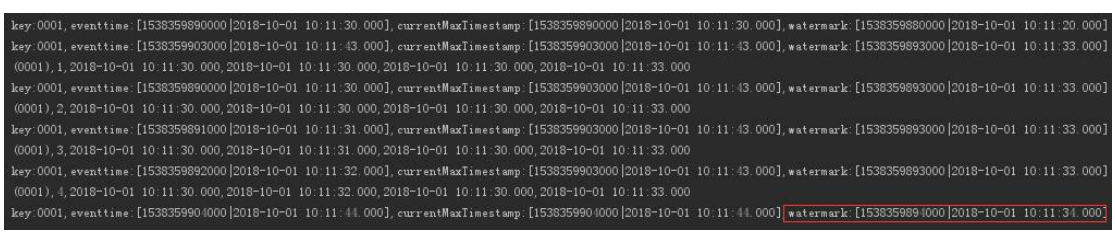
| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359890000 | 1538359890000 | 1538359880000 | | |
| | 2018-10-01 10:11:30.000 | 2018-10-01 10:11:30.000 | 2018-10-01 10:11:20.000 | | |
| 0001 | 1538359903000 | 1538359903000 | 1538359893000 | | |
| | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |
| 0001 | 1538359890000 | 1538359903000 | 1538359893000 | | |
| | 2018-10-01 10:11:30.000 | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |
| 0001 | 1538359891000 | 1538359903000 | 1538359893000 | | |
| | 2018-10-01 10:11:31.000 | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |
| 0001 | 1538359892000 | 1538359903000 | 1538359893000 | | |
| | 2018-10-01 10:11:32.000 | 2018-10-01 10:11:43.000 | 2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |

我们再输入一条数据，把 water 调整到 10:11:34

输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359890000
0001,1538359891000
0001,1538359892000
0001,1538359904000
```

输出：



汇总如下：

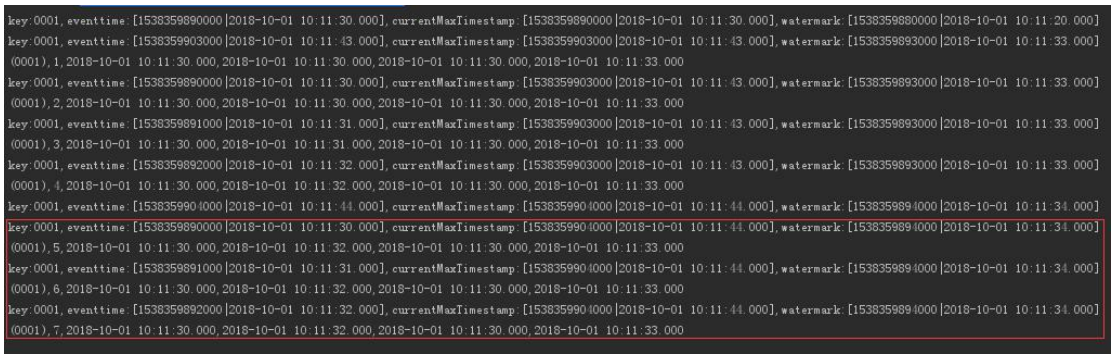
| Key | Event Time | CurrentMaxTimeStamp | WaterMark | window_start_time | window_end_time |
|---|---|---|---|---|---|
| 0001 | 1538359890000<br>2018-10-01 10:11:30.000 | 1538359890000<br>2018-10-01 10:11:30.000 | 1538359880000<br>2018-10-01 10:11:20.000 | | |
| 0001 | 1538359903000<br>2018-10-01 10:11:43.000 | 1538359903000<br>2018-10-01 10:11:43.000 | 1538359893000<br>2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |
| 0001 | 1538359890000<br>2018-10-01 10:11:30.000 | 1538359903000<br>2018-10-01 10:11:43.000 | 1538359893000<br>2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |
| 0001 | 1538359891000<br>2018-10-01 10:11:31.000 | 1538359903000<br>2018-10-01 10:11:43.000 | 1538359893000<br>2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |
| 0001 | 1538359892000<br>2018-10-01 10:11:32.000 | 1538359903000<br>2018-10-01 10:11:43.000 | 1538359893000<br>2018-10-01 10:11:33.000 | [10:11:33.000 | 10:11:33.000) |
| 0001 | 1538359904000<br>2018-10-01 10:11:44.000 | 1538359904000<br>2018-10-01 10:11:44.000 | 1538359894000<br>2018-10-01 10:11:34.000 | | |

此时，把 water 上升到了 10:11:34，我们再输入几条 eventtime<watermark 的数据验证一下效果

输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359890000
0001,1538359891000
0001,1538359892000
0001,1538359904000
0001,1538359890000
0001,1538359891000
0001,1538359892000
```

输出：




发现输入的三行数据都触发了 window 的执行。

我们再输入一条数据，把 water 调整到 10:11:35

输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359890000
0001,1538359891000
0001,1538359892000
0001,1538359904000
0001,1538359890000
0001,1538359891000
0001,1538359892000
0001,1538359905000
```

输出：

```
key:0001, eventtime:[1538359890000|2018-10-01 10:11:30.000], currentMaxTimestamp:[1538359890000|2018-10-01 10:11:30.000], watermark:[1538359880000|2018-10-01 10:11:20.000]
key:0001, eventtime:[1538359903000|2018-10-01 10:11:43.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),1,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359890000|2018-10-01 10:11:30.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),2,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359891000|2018-10-01 10:11:31.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),3,2018-10-01 10:11:30.000,2018-10-01 10:11:31.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),4,2018-10-01 10:11:30.000,2018-10-01 10:11:32.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359904000|2018-10-01 10:11:44.000], currentMaxTimestamp:[1538359904000|2018-10-01 10:11:44.000], watermark:[1538359894000|2018-10-01 10:11:34.000]
key:0001, eventtime:[1538359890000|2018-10-01 10:11:30.000], currentMaxTimestamp:[1538359904000|2018-10-01 10:11:44.000], watermark:[1538359894000|2018-10-01 10:11:34.000]
(0001),5,2018-10-01 10:11:30.000,2018-10-01 10:11:32.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359891000|2018-10-01 10:11:31.000], currentMaxTimestamp:[1538359904000|2018-10-01 10:11:44.000], watermark:[1538359894000|2018-10-01 10:11:34.000]
(0001),6,2018-10-01 10:11:30.000,2018-10-01 10:11:32.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359904000|2018-10-01 10:11:44.000], watermark:[1538359894000|2018-10-01 10:11:34.000]
(0001),7,2018-10-01 10:11:30.000,2018-10-01 10:11:32.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359905000|2018-10-01 10:11:45.000], currentMaxTimestamp:[1538359905000|2018-10-01 10:11:45.000], watermark:[1538359895000|2018-10-01 10:11:35.000]
```

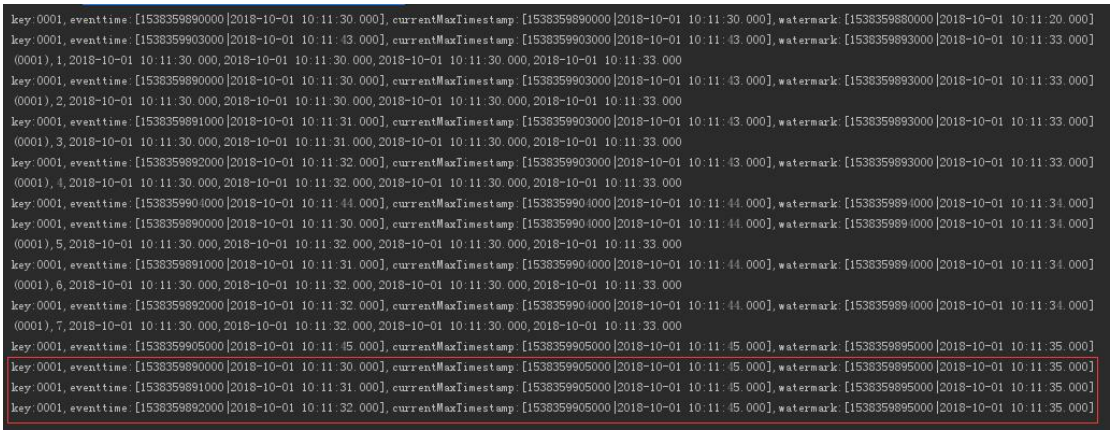
此时，watermark 上升到了 10:11:35

我们再输入几条 eventtime<watermark 的数据验证一下效果

输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359890000
0001,1538359891000
0001,1538359892000
0001,1538359904000
0001,1538359890000
0001,1538359891000
0001,1538359892000
0001,1538359905000
0001,1538359890000
0001,1538359891000
0001,1538359892000
```

输出：




发现这几条数据都没有触发 window。

分析：
当 watemark 等于 10:11:33 的时候，正好是 window_end_time，所以会触发[10:11:30~10:11:33)的 window 执行。
当窗口执行过后，我们输入[10:11:30~10:11:33) window 内的数据会发现 window 是可以被触发的。
当 watemark 提升到 10:11:34 的时候，我们输入[10:11:30~10:11:33)window 内的数据会发现 window 也是可以被触发的。
当 watemark 提升到 10:11:35 的时候，我们输入[10:11:30~10:11:33)window 内的数据会发现 window 不会被触发了。

由于我们在前面设置了 allowedLateness(Time.seconds(2))，可以允许延迟在 2s 内的数据继续触发 window 执行。

所以当 watermark 是 10:11:34 的时候可以触发 window，但是 10:11:35 的时候就不行了。

总结：
对于此窗口而言，允许 2 秒的迟到数据，即第一次触发是在 watermark >=window_end_time 时
第二次（或多次）触发的条件是 watermark < window_end_time + allowedLateness 时间内，这个窗口有 late 数据到达时。

解释：
当 watermark 等于 10:11:34 的时候，我们输入 eventtime 为 10:11:30、10:11:31、10:11:32 的数据的时候，是可以触发的，因为这些数据的 window_end_time 都是 10:11:33，也就是 10:11:34<10:11:33+2 为 true。

但是当 watermark 等于 10:11:35 的时候，我们再输入 eventtime 为 10:11:30、10:11:31、10:11:32 的数据的时候，这些数据的 window_end_time 都是 10:11:33，此时，10:11:35<10:11:33+2 为 false 了。所以最终这些数据迟到的时间太久了，就不会再触发 window 执行了。

# 4.3：sideOutputLateData 收集迟到的数据

通过 sideOutputLateData 可以把迟到的数据统一收集，统一存储，方便后期排查问题。

需要先调整代码：

```
//保存被丢弃的数据
OutputTag<Tuple2<String, Long>> outputTag = new OutputTag<Tuple2<String, Long>>( id: "late-data"){};
//注意，由于getSideOutput方法是SingleOutputStreamOperator子类中的特有方法，所以这里的类型，不能使用它的父类dataStream。
SingleOutputStreamOperator<String> window = waterMarkStream.keyBy( ...fields: 0)
        .window(TumblingEventTimeWindows.of(Time.seconds(3)))//按照消息的EventTime分配窗口，和调用TimeWindow效果一样
        //.allowedLateness(Time.seconds(2))//允许数据迟到2秒
        .sideOutputLateData(outputTag)
        .apply(new WindowFunction<Tuple2<String, Long>, String, Tuple, TimeWindow>() {
            /**
             * 对window内的数据进行排序，保证数据的顺序
             * @param tuple
             * @param window
             * @param input
             * @param out
             * @throws Exception
             */
            @Override
            public void apply(Tuple tuple, TimeWindow window, Iterable<Tuple2<String, Long>> input, Collector<String
                String key = tuple.toString();
                List<Long> arrarList = new ArrayList<Long>();
                Iterator<Tuple2<String, Long>> it = input.iterator();
                while (it.hasNext()) {
                    Tuple2<String, Long> next = it.next();
                    arrarList.add(next.f1);
                }
                Collections.sort(arrarList);
                SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss.SSS");
                String result = key + "," + arrarList.size() + "," + sdf.format(arrarList.get(0)) + "," + sdf.format
                        + "," + sdf.format(window.getStart()) + "," + sdf.format(window.getEnd());
                out.collect(result);
            }
        });
//把迟到的数据暂时打印到控制台，实际中可以保存到其他存储介质中
DataStream<Tuple2<String, Long>> sideOutput = window.getSideOutput(outputTag);
sideOutput.print();
//测试-把结果打印到控制台即可
window.print();

//注意：因为flink是懒加载的，所以必须调用execute方法，上面的代码才会执行
env.execute( jobName: "eventtime-watermark");
```

我们来输入一些数据验证一下

输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
```

输出：

```
key:0001, eventtime:[1538359890000|2018-10-01 10:11:30.000], currentMaxTimestamp:[1538359890000|2018-10-01 10:11:30.000], watermark:[1538359880000|2018-10-01 10:11:20.000]
key:0001, eventtime:[1538359903000|2018-10-01 10:11:43.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),1,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
```

此时，window 被触发执行了，此时 watermark 是 10:11:33

下面我们再输入几个 eventtime 小于 watermark 的数据测试一下

输入：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359890000
0001,1538359891000
0001,1538359892000
```

输出：

```
key:0001, eventtime:[1538359890000|2018-10-01 10:11:30.000], currentMaxTimestamp:[1538359890000|2018-10-01 10:11:30.000], watermark:[1538359880000|2018-10-01 10:11:20.000]
key:0001, eventtime:[1538359903000|2018-10-01 10:11:43.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001),1,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
key:0001, eventtime:[1538359890000|2018-10-01 10:11:30.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001,1538359890000)
key:0001, eventtime:[1538359891000|2018-10-01 10:11:31.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001,1538359891000)
key:0001, eventtime:[1538359892000|2018-10-01 10:11:32.000], currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000], watermark:[1538359893000|2018-10-01 10:11:33.000]
(0001,1538359892000)
```

此时，针对这几条迟到的数据，都通过 sideOutputLateData 保存到了 outputTag 中。

# 5：在多并行度下的 watermark 应用

前面代码中设置了并行度为 1
env.setParallelism(1);

如果这里不设置的话，代码在运行的时候会默认读取本机 CPU 数量设置并行度。

把代码的并行度代码注释掉

```
//设置并行度为1,默认并行度是当前机器的cpu数量
//env.setParallelism(1);
```

然后在输出内容前面加上线程 id

```
//定义如何提取timestamp
@Override
public long extractTimestamp(Tuple2<String, Long> element, long previousElementTimestamp) {
    long timestamp = element.f1;
    currentMaxTimestamp = Math.max(timestamp, currentMaxTimestamp);
    long id = Thread.currentThread().getId();
    System.out.println("currentThreadId:"+id+",key:"+element.f0+",eventtime:["+element.f1+"|"+sdf.format(ele
            sdf.format(currentMaxTimestamp)+"],watermark:["+getCurrentWatermark().getTimestamp()+"|"+sdf.for
    return timestamp;
}
```

会出现如下数据：

输入如下几行内容：

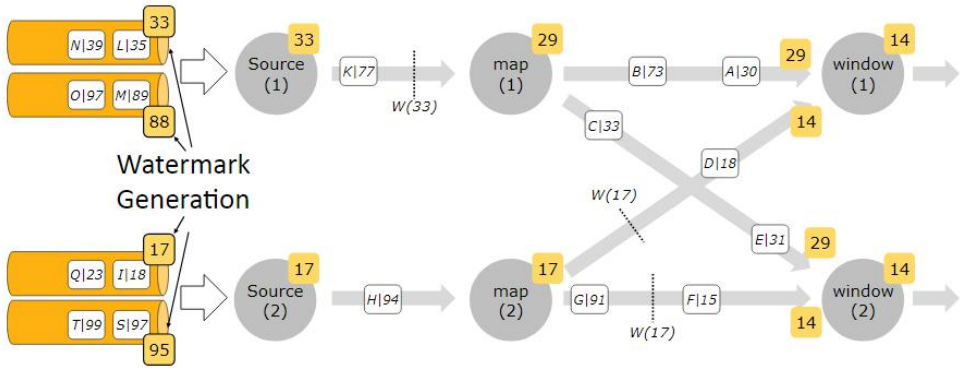
```
[root@hadoop100 soft]# nc -l 9000
0001,1538359882000
0001,1538359886000
0001,1538359892000
0001,1538359893000
0001,1538359894000
0001,1538359896000
0001,1538359897000
```

输出：

```
currentThreadId:64,key:0001,eventtime:[1538359882000|2018-10-01 10:11:22.000],currentMaxTimestamp:[1538359882000|2018-10-01 10:11:22.000],watermark:[1538359872000|2018-10-01 10:11:12.000]
currentThreadId:51,key:0001,eventtime:[1538359886000|2018-10-01 10:11:26.000],currentMaxTimestamp:[1538359886000|2018-10-01 10:11:26.000],watermark:[1538359876000|2018-10-01 10:11:16.000]
currentThreadId:50,key:0001,eventtime:[1538359892000|2018-10-01 10:11:32.000],currentMaxTimestamp:[1538359892000|2018-10-01 10:11:32.000],watermark:[1538359882000|2018-10-01 10:11:22.000]
currentThreadId:49,key:0001,eventtime:[1538359893000|2018-10-01 10:11:33.000],currentMaxTimestamp:[1538359893000|2018-10-01 10:11:33.000],watermark:[1538359883000|2018-10-01 10:11:23.000]
currentThreadId:46,key:0001,eventtime:[1538359894000|2018-10-01 10:11:34.000],currentMaxTimestamp:[1538359894000|2018-10-01 10:11:34.000],watermark:[1538359884000|2018-10-01 10:11:24.000]
currentThreadId:44,key:0001,eventtime:[1538359896000|2018-10-01 10:11:36.000],currentMaxTimestamp:[1538359896000|2018-10-01 10:11:36.000],watermark:[1538359886000|2018-10-01 10:11:26.000]
currentThreadId:48,key:0001,eventtime:[1538359897000|2018-10-01 10:11:37.000],currentMaxTimestamp:[1538359897000|2018-10-01 10:11:37.000],watermark:[1538359887000|2018-10-01 10:11:27.000]
```

会发现 window 没有被触发。
因为此时，这 7 条数据都是被不同的线程处理的。每个线程都有一个 watermark。

因为在多并行度的情况下，watermark 对齐会取所有 channel 最小的 watermark
但是我们现在默认有 8 个并行度，这 7 条数据都被不同的线程所处理，到现在还没获取到最小的 watermark，所以 window 无法被触发执行。




下面我们来验证一下，把代码中的并行度调整为 2.

```
//设置并行度为1,默认并行度是当前机器的cpu数量
env.setParallelism(2);
```

输入如下内容：

```
[root@hadoop100 soft]# nc -l 9000
0001,1538359890000
0001,1538359903000
0001,1538359908000
```

输出：

```
currentThreadId:44,key:0001,eventtime:[1538359890000|2018-10-01 10:11:30.000],currentMaxTimestamp:[1538359890000|2018-10-01 10:11:30.000],watermark:[1538359880000|2018-10-01 10:11:20.000]
currentThreadId:42,key:0001,eventtime:[1538359903000|2018-10-01 10:11:43.000],currentMaxTimestamp:[1538359903000|2018-10-01 10:11:43.000],watermark:[1538359893000|2018-10-01 10:11:33.000]
currentThreadId:44,key:0001,eventtime:[1538359908000|2018-10-01 10:11:48.000],currentMaxTimestamp:[1538359908000|2018-10-01 10:11:48.000],watermark:[1538359898000|2018-10-01 10:11:38.000]
1> (0001),1,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:30.000,2018-10-01 10:11:33.000
```

此时会发现，当第三条数据输入完以后，[10:11:30,10:11:33)这个 window 被触发了。
前两条数据输入之后，获取到的最小 watermark 是 10:11:20，这个时候对应的 window 中没有数据。
第三条数据输入之后，获取到的最小 watermark 是 10:11:33，这个时候对应的窗口就是[10:11:30,10:11:33)。所以就触发了。