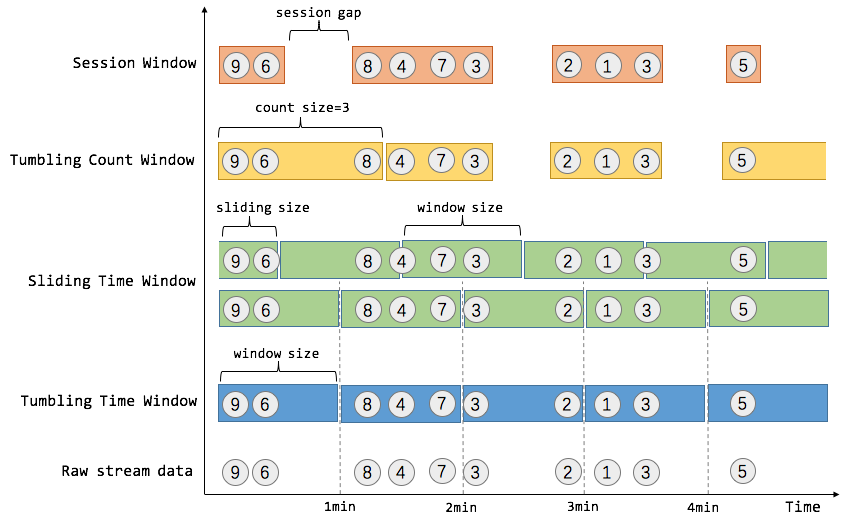
### 什么是window机制

在流处理应用中，数据是连续不断的，因此我们不可能等到所有数据都到了才开始处理。当然我们可以每来一个消息就处理一次，但是有时我们需要做一些聚合类的处理，例如：在过去的1分钟内有多少用户点击了我们的网页。在这种情况下，我们必须定义一个窗口，用来收集最近一分钟内的数据，并对这个窗口内的数据进行计算。

窗口可以是时间驱动的（Time Window，例如：每30秒钟），也可以是数据驱动的（Count Window，例如：每一百个元素）。

窗口可以是时间驱动的（Time Window，例如：每30秒钟），也可以是数据驱动的（Count Window，例如：每一百个元素）。一种经典的窗口分类可以分成：滚动窗口（Tumbling Window，无重叠），滑动窗口（Sliding Window，有重叠），和会话窗口（Session Window，活动间隙）。

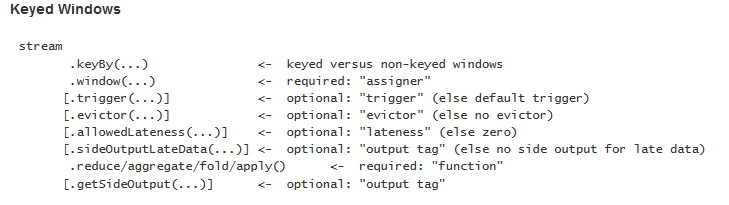


2.

每个窗口算子中，包含了Windows Assigner、Windows Trigger（窗口触发器）、Evictor（数据剔除器）、Lateness（时延设定）。OutPutTag（输出标签）以及Windows Function等。其中Windows Assigner和Windows Function是所有窗口算子必须指定的的。

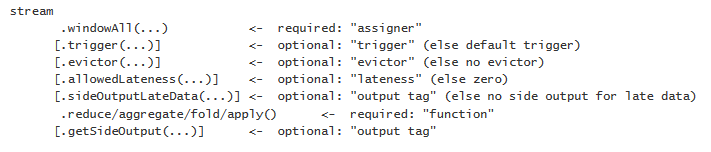
2.1keyed windows

keyed windows会将数据分区，最后计算出每个key（同一分区）的结果数据，例如：同一用户在某一段时间内的访问频次。



2.2non-keyed

non-keyed则需要调用windowsAll来制定window assigner，所有数据都会在窗口算子中路由到一个Task中进行计算，并得到全局统计结果。



### Countwindow

**package** com.sqq;  
  
**import** com.sq.wc1;  
**import** org.apache.flink.api.common.functions.FlatMapFunction;  
**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.windowing.time.Time;  
**import** org.apache.flink.util.Collector;  
  
**public class** ww1 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setStreamTimeCharacteristic(TimeCharacteristic.***ProcessingTime***);  
  
 DataStream<Tuple2<String, Integer>> dataStream = env  
 .socketTextStream(args[0], Integer.*parseInt*(args[1]))  
 .flatMap(**new** wc1.Sp())  
 .keyBy(0)  
 .countWindow(10)  
 .sum(1);  
  
 dataStream.print();  
  
 env.execute(**"Window WordCount"**);  
 }  
  
 **public static class** Sp **implements** FlatMapFunction<String, Tuple2<String, Integer>> {  
 @Override  
 **public void** flatMap(String sentence, Collector<Tuple2<String, Integer>> out) **throws** Exception {  
 **for** (String word : sentence.split(**" "**)) {  
 out.collect(**new** Tuple2<String, Integer>(word, 1));  
 }  
 }  
 }  
  
}

### Timewindow

**package** com.sqq;  
  
**import** com.sq.wc1;  
**import** org.apache.flink.api.common.functions.FlatMapFunction;  
**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.windowing.time.Time;  
**import** org.apache.flink.util.Collector;  
  
**public class** ww2 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setStreamTimeCharacteristic(TimeCharacteristic.***ProcessingTime***);  
  
 DataStream<Tuple2<String, Integer>> dataStream = env  
 .socketTextStream(args[0], Integer.*parseInt*(args[1]))  
 .flatMap(**new** wc1.Sp())  
 .keyBy(0)  
 .timeWindow(Time.*seconds*(15))  
 .sum(1);  
  
 dataStream.print();  
  
 env.execute(**"Window WordCount"**);  
 }  
  
 **public static class** Sp **implements** FlatMapFunction<String, Tuple2<String, Integer>> {  
 @Override  
 **public void** flatMap(String sentence, Collector<Tuple2<String, Integer>> out) **throws** Exception {  
 **for** (String word : sentence.split(**" "**)) {  
 out.collect(**new** Tuple2<String, Integer>(word, 1));  
 }  
 }  
 }  
  
}

### 2.3 滑动窗口

.timeWindow(Time.*seconds*(150),Time.*seconds*(20))

窗口20s，计算150s内的数据

### 3.1 滚动窗口

滚动窗口（Tumbling Windows）：根据固定时间或大小进行切分，窗口与窗口之间元素互不重叠。DataStream Api中提供了基于Event Time和process Time两种时间类型的滚动窗口，对应的Assigner分别为TumblingEventTimeWindows，TumblingProcessingTimeWindows。

.keyBy(0).window(TumblingEventTimeWindows.*of*(Time.*seconds*(10L)))

开发：

**package** com.sqq1;  
  
  
**import** org.apache.flink.api.common.functions.FlatMapFunction;  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.api.java.tuple.Tuple3;  
**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.timestamps.BoundedOutOfOrdernessTimestampExtractor;  
**import** org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;  
**import** org.apache.flink.streaming.api.windowing.time.Time;  
**import** org.apache.flink.util.Collector;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** w1 {  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setStreamTimeCharacteristic(TimeCharacteristic.***EventTime***);  
  
 DataStream<Tuple3<String,Integer,Long>> dataStreamSource = env.socketTextStream(**"192.168.8.201"**,9999)  
 .map(**new** MapFunction<String, Tuple3<String, Integer, Long>>() {  
 @Override  
 **public** Tuple3<String, Integer, Long> map(String s) **throws** Exception {  
 **return new** Tuple3<String, Integer, Long>  
 (s.split(**" "**)[0], Integer.*parseInt*(s.split(**" "**)[1]),Long.*parseLong*(s.split(**" "**)[2]));  
 }  
 });  
 DataStream<Tuple3<String,Integer,Long>> result = dataStreamSource.assignTimestampsAndWatermarks(  
 **new** BoundedOutOfOrdernessTimestampExtractor<Tuple3<String, Integer, Long>>(Time.*seconds*(10)) {  
 @Override  
 **public long** extractTimestamp(Tuple3<String, Integer, Long> element) {  
 **return** element.**f2**;  
 }  
 }  
 ).keyBy(0).window(TumblingEventTimeWindows.*of*(Time.*seconds*(10L)))  
 .sum(1);  
 result.print();  
 env.execute(**"assign"**);  
 }  
}

nc窗口

[hadoop@h201 ~]$ nc -l 9999

sq 100 2000

sq 100 2001

sq 100 200000

结果：

scala> (sq,200,2000)

(sq,100,200000)

TumblingProcessingTimeWindows类似。我们还可以通过timeWindow（）的方式定义Window Assigner，timeWindow是Flink对时间窗口做的一层封装

基于

### 3.2滑动窗口

滑动窗口（Sliding Windows）：滑动窗口只是在滚动窗口的基础上增加了窗口滑动时间（Slide Time），允许窗口数据发生重叠。简言之，当窗口的size固定之后，窗口会根据给定的Slide Time向前滑动，即窗口之间的数据重叠大小是根据window size和 Slide size来决定的。也就是说有可能会出现窗口不连续，数据可能不在任何一个窗口内，当slide size和windows size相等时，滑动窗口就降级为滚动窗口了。

keyBy(0).window(SlidingEventTimeWindows.*of*(Time.*seconds*(5L),Time.*seconds*(100L)))

开发

**package** com.sqq1;  
  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.api.java.tuple.Tuple3;  
**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.timestamps.BoundedOutOfOrdernessTimestampExtractor;  
**import** org.apache.flink.streaming.api.windowing.assigners.SlidingEventTimeWindows;  
**import** org.apache.flink.streaming.api.windowing.assigners.TumblingEventTimeWindows;  
**import** org.apache.flink.streaming.api.windowing.time.Time;  
  
**public class** w2{  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setStreamTimeCharacteristic(TimeCharacteristic.***EventTime***);  
  
 DataStream<Tuple3<String,Integer,Long>> dataStreamSource = env.socketTextStream(**"192.168.8.201"**,9999)  
 .map(**new** MapFunction<String, Tuple3<String, Integer, Long>>() {  
 @Override  
 **public** Tuple3<String, Integer, Long> map(String s) **throws** Exception {  
 **return new** Tuple3<String, Integer, Long>  
 (s.split(**" "**)[0], Integer.*parseInt*(s.split(**" "**)[1]),Long.*parseLong*(s.split(**" "**)[2]));  
 }  
 });  
 DataStream<Tuple3<String,Integer,Long>> result = dataStreamSource.assignTimestampsAndWatermarks(  
 **new** BoundedOutOfOrdernessTimestampExtractor<Tuple3<String, Integer, Long>>(Time.*seconds*(10)) {  
 @Override  
 **public long** extractTimestamp(Tuple3<String, Integer, Long> element) {  
 **return** element.**f2**;  
 }  
 }  
 ).keyBy(0).window(SlidingEventTimeWindows.*of*(Time.*seconds*(5L),Time.*seconds*(100L)))  
 .sum(1);  
 result.print();  
 env.execute(**"assign"**);  
 }  
}

**想要使用process Time只需要将例子中SlidingEventTimeWindows改为SlidingProcessingTimeWindows即可。**

### 3.3 会话窗口

会话窗口（Session Windows）：本质还是TimeWindow，将某段时间内活跃度比较高的数据聚合成一个窗口进行计算，窗口触发的条件是Session Gap，Session Gap规定了不活跃数据的时间上限。会话窗口适用于非连续型数据处理或者周期性产生数据的场景。

**.window(EventTimeSessionWindows.*withGap*(Time.*seconds*(20)))**

**会话中20s不活跃就任务是一个窗口**

开发

**package** com.sqq1;  
  
**import** org.apache.flink.api.common.functions.MapFunction;  
**import** org.apache.flink.api.java.tuple.Tuple3;  
**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.timestamps.BoundedOutOfOrdernessTimestampExtractor;  
**import** org.apache.flink.streaming.api.windowing.assigners.EventTimeSessionWindows;  
**import** org.apache.flink.streaming.api.windowing.time.Time;  
  
**public class** w3{  
 **public static void** main(String[] args) **throws** Exception {  
 StreamExecutionEnvironment env = StreamExecutionEnvironment.*getExecutionEnvironment*();  
 env.setStreamTimeCharacteristic(TimeCharacteristic.***EventTime***);  
  
 DataStream<Tuple3<String,Integer,Long>> dataStreamSource = env.socketTextStream(**"192.168.8.201"**,9999)  
 .map(**new** MapFunction<String, Tuple3<String, Integer, Long>>() {  
 @Override  
 **public** Tuple3<String, Integer, Long> map(String s) **throws** Exception {  
 **return new** Tuple3<String, Integer, Long>  
 (s.split(**" "**)[0], Integer.*parseInt*(s.split(**" "**)[1]),Long.*parseLong*(s.split(**" "**)[2]));  
 }  
 });  
 DataStream<Tuple3<String,Integer,Long>> result = dataStreamSource.assignTimestampsAndWatermarks(  
 **new** BoundedOutOfOrdernessTimestampExtractor<Tuple3<String, Integer, Long>>(Time.*seconds*(10)) {  
 @Override  
 **public long** extractTimestamp(Tuple3<String, Integer, Long> element) {  
 **return** element.**f2**;  
 }  
 }  
 ).keyBy(0).window(EventTimeSessionWindows.*withGap*(Time.*seconds*(20)))  
 .sum(1);  
 result.print();  
 env.execute(**"assign"**);  
 }  
}