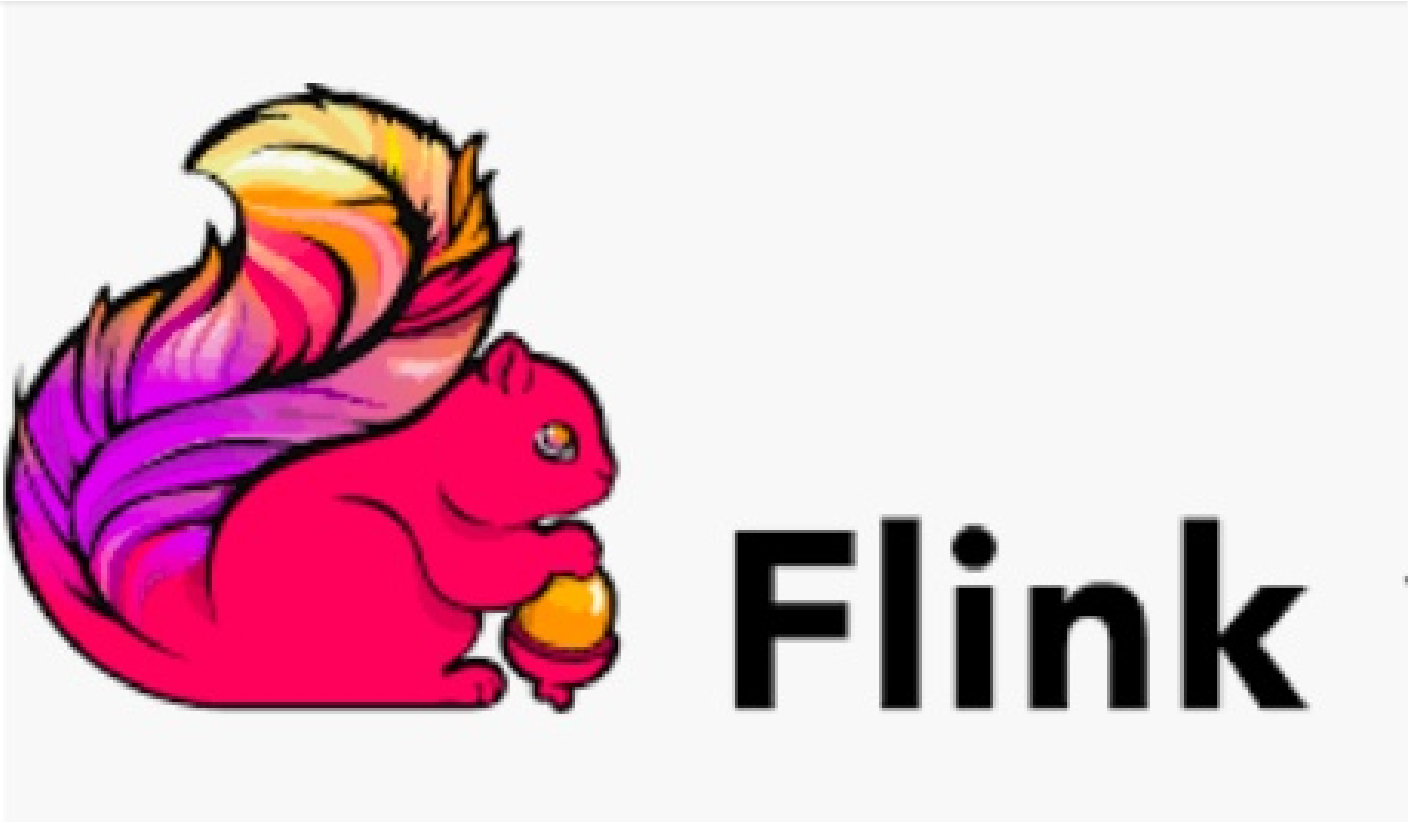
知乎 首发于 大数据技术与应用实战



Flink教程-聊聊 flink 1.11 中新的水印策略

zhangjun
公众号（大数据技术与应用实战），分享一些大数据实战案例。

背景

在flink 1.11之前的版本中，提供了两种生成水印（Watermark）的策略，分别是AssignerWithPunctuatedWatermarks和AssignerWithPeriodicWatermarks，这两个接口都继承自TimestampAssigner接口。

用户想使用不同的水印生成方式，则需要实现不同的接口，但是这样引发了一个问题，对于想给水印添加一些通用的、公共的功能则变得复杂，因为我们需要给这两个接口都同时添加新的功能，这样还造成了代码的重复。

知乎 首发于 大数据技术与应用实战

新的水印生成接口

当我们构建了一个DataStream之后，使用assignTimestampsAndWatermarks方法来构造水印，新的接口需要传入一个WatermarkStrategy对象。

```
DataStream#assignTimestampsAndWatermarks(WatermarkStrategy<T>)
```

WatermarkStrategy 这个接口是做什么的呢？这里面提供了很多静态的方法和带有缺省实现的方法，只有一个方法是非default和没有缺省实现的，就是下面的这个方法。

```
/**
 * Instantiates a WatermarkGenerator that generates watermarks according to this stra
 */
@Override
WatermarkGenerator<T> createWatermarkGenerator(WatermarkGeneratorSupplier.Context con
```

所以默认情况下，我们只需要实现这个方法就行了，这个方法主要是返回一个WatermarkGenerator，我们在进入这里边看看。

```
@Public
public interface WatermarkGenerator<T> {

 /**
  * Called for every event, allows the watermark generator to examine and remember the
  * event timestamps, or to emit a watermark based on the event itself.
  */
 void onEvent(T event, long eventTimestamp, WatermarkOutput output);

 /**
  * Called periodically, and might emit a new watermark, or not.
  *
  * <p>The interval in which this method is called and Watermarks are generated
  * depends on {@link ExecutionConfig#getAutoWatermarkInterval()}.
  */
 void onPeriodicEmit(WatermarkOutput output);
}
```

- onEvent： 每个元素都会调用这个方法，如果我们想依赖每个元素生成一个水印，然后发射到下游(可选，就是看是否用output来收集水印)，我们可以实现这个方法.
- onPeriodicEmit : 如果数据量比较大的时候，我们每条数据都生成一个水印的话，会影响性能，所以这里还有一个周期性生成水印的方法。这个水印的生成周期可以这样设置：env.getConfig().setAutoWatermarkInterval(5000L);

我们自己实现一个简单的周期性的发射水印的例子：

在这个onEvent方法里，我们从每个元素里抽取了一个时间字段，但是我们并没有生成水印发射给下游，而是自己保存了在一个变量里，在onPeriodicEmit方法里，使用最大的日志时间减去我们想要的延迟时间作为水印发射给下游。

```
DataStream<Tuple2<String,Long>> withTimestampsAndWatermarks = dataStream.assignTimes
  new WatermarkStrategy<Tuple2<String,Long>>(){
   @Override
   public WatermarkGenerator<Tuple2<String,Long>> createWatermarkGenerator(
     WatermarkGeneratorSupplier.Context context){
    return new WatermarkGenerator<Tuple2<String,Long>>(){
     private long maxTimestamp;
     private long delay = 3000;
     @Override
     public void onEvent(
       Tuple2<String,Long> event,
       long eventTimestamp,
       WatermarkOutput output){
      maxTimestamp = Math.max(maxTimestamp, event.f1);
     }
     @Override
     public void onPeriodicEmit(WatermarkOutput output){
      output.emitWatermark(new Watermark(maxTimestamp - delay));
     }
    };
   }
  });
```

知乎 首发于 大数据技术与应用实战

内置水印生成策略

为了方便开发，flink提供了一些内置的水印生成方法供我们使用。

固定延迟生成水印

通过静态方法forBoundedOutOfOrderness提供,入参接收一个Duration类型的时间间隔，也就是我们可以接受的最大的延迟时间.使用这种延迟策略的时候需要我们对数据的延迟时间有一个大概的预估判断。

```
WatermarkStrategy#forBoundedOutOfOrderness(Duration maxOutOfOrderness)
```

我们实现一个延迟3秒的固定延迟水印，可以这样做：

```
DataStream dataStream = ..... ;
dataStream.assignTimestampsAndWatermarks(WatermarkStrategy.forBoundedOutOfOrderness(Du
```

他的底层使用的WatermarkGenerator接口的一个实现类BoundedOutOfOrdernessWatermarks。我们看下源码中的这两个方法，是不是和我们上面自己写的很像.

```
@Override
public void onEvent(T event, long eventTimestamp, WatermarkOutput output) {
 maxTimestamp = Math.max(maxTimestamp, eventTimestamp);
}

@Override
```

```
}
```

单调递增生成水印

通过静态方法forMonotonousTimestamps来提供.

```
WatermarkStrategy.forMonotonousTimestamps()
```

这个也就是相当于上述的延迟策略去掉了延迟时间，以event中的时间戳充当了水印。

在程序中可以这样使用：

```
DataStream dataStream = ..... ;
dataStream.assignTimestampsAndWatermarks(WatermarkStrategy.forMonotonousTimestamps());
```

它的底层实现是AscendingTimestampsWatermarks，其实它就是BoundedOutOfOrdernessWatermarks类的一个子类，没有了延迟时间，我们来看看具体源码的实现.

```
@Public
public class AscendingTimestampsWatermarks<T> extends BoundedOutOfOrdernessWatermarks<

 /**
  * Creates a new watermark generator with for ascending timestamps.
  */
 public AscendingTimestampsWatermarks() {
  super(Duration.ofMillis(0));
 }
}
```

event时间的获取

上述我们讲了flink自带的两种水印生成策略，但是对于我们使用eventtime语义的时候，我们想从我们的自己的数据中抽取eventtime，这个就需要TimestampAssigner了.

```
public interface TimestampAssigner<T> {

    ............

 long extractTimestamp(T element, long recordTimestamp);
}
```

使用的时候我们主要就是从我们自己的元素element中提取我们想要的eventtime。

使用flink自带的水印策略和eventtime抽取类，可以这样用：

```
DataStream dataStream = ..... ;
dataStream.assignTimestampsAndWatermarks(
    WatermarkStrategy
      .<Tuple2<String,Long>>forBoundedOutOfOrderness(Duration.ofSeconds(5))
      .withTimestampAssigner((event, timestamp)->event.f1));
```

处理空闲数据源

在某些情况下，由于数据产生的比较少，导致一段时间内没有数据产生，进而就没有水印的生成，导致下游依赖水印的一些操作就会出现问题，比如某一个算子的上游有多个算子，这种情况下，水印是取其上游两个算子的较小值，如果上游某一个算子因为缺少数据迟迟没有生成水印，就会出现eventtime倾斜问题，导致下游没法触发计算。

所以filnk通过WatermarkStrategy.withIdleness()方法允许用户在配置的时间内（即超时时间内）没有记录到达时将一个流标记为空闲。这样就意味着下游的数据不需要等待水印的到来。

当下次有水印生成并发射到下游的时候，这个数据流重新变成活跃状态。

通过下面的代码来实现对于空闲数据流的处理

```
WatermarkStrategy
        .<Tuple2<Long, String>>forBoundedOutOfOrderness(Duration.ofSeconds(20))
        .withIdleness(Duration.ofMinutes(1));
```

完整的代码请参考：

知乎 首发于 大数据技术与应用实战

更多精彩内容，欢迎关注我的公众号【大数据技术与应用实战】

发布于 2020-07-12

Flink 大数据 Java

文章被以下专栏收录

推荐阅读

2020 Flink 学习路线总结（持续更新）

程序员小陶 发表于大数据进击...

Flink 零基础实战教程：如何计算实时热门商品

Flink 中文社区

1 条评论

回家喽 2020-08-07

您好，关于空闲数据源的问题，假如一直没有数据流入，最后一个窗口该如何执行计算。比如我下游有个窗口EventTimeSessionWindows.withGap()，在没有数据流入时不会触发算子也就是窗口永远不会关闭了，但是ProcessingTimeSessionWindows.withGap()却不会这

赞