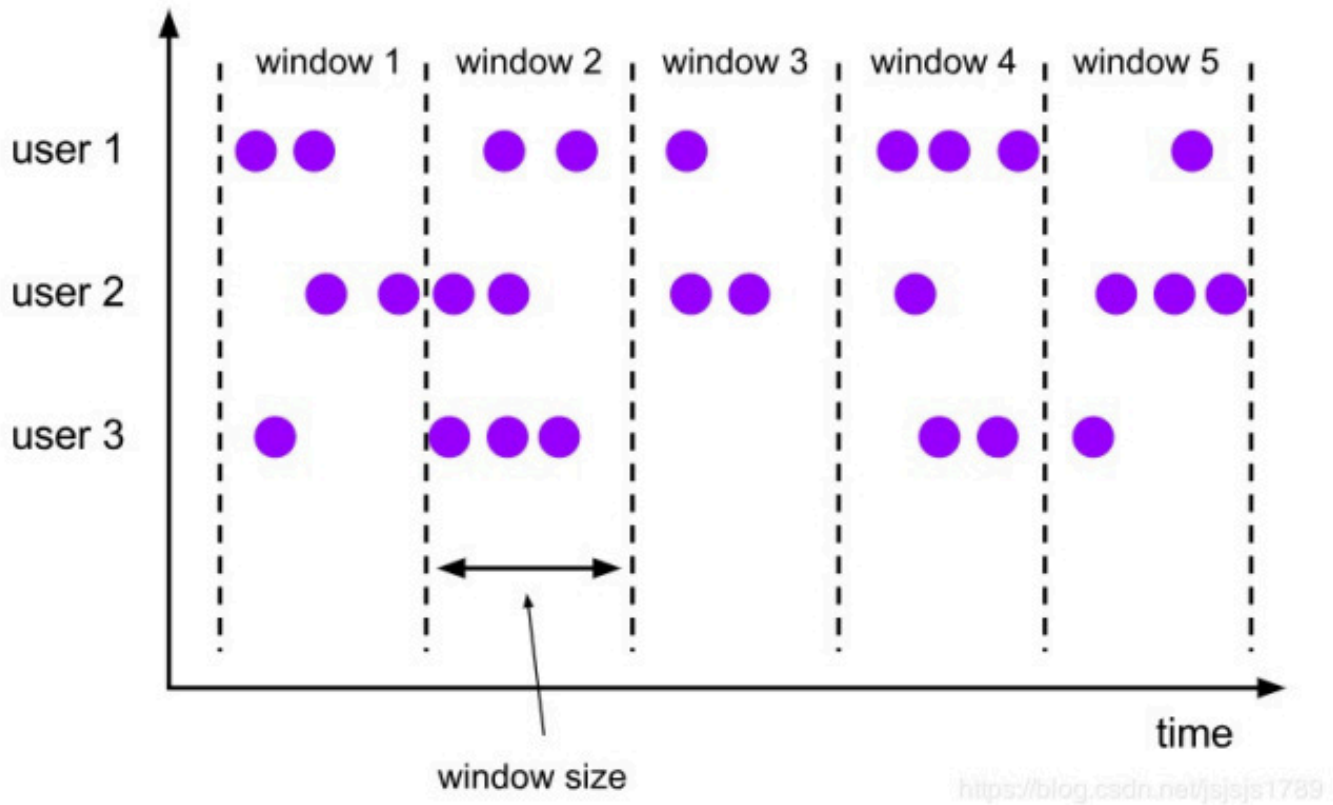
Flink 源码分析之写给大忙人看的 Flink Window 原理

发布于: 2020 年 06 月 13 日

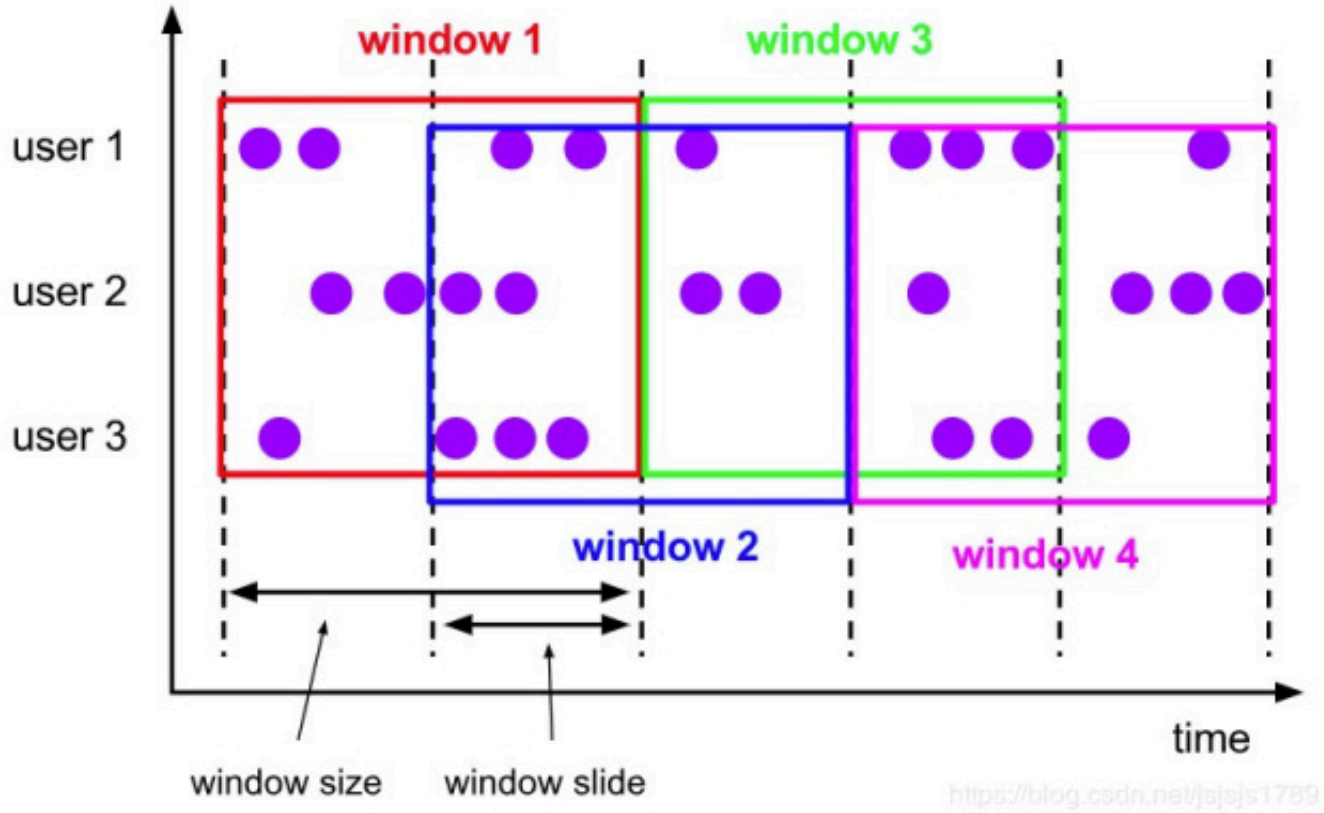
Window 可以说是 Flink 中必不可少的 operator 之一，在很多场合都有很非凡的表现。今天呢，我们就一起来看一下 window 是如何实现的。

window 分类

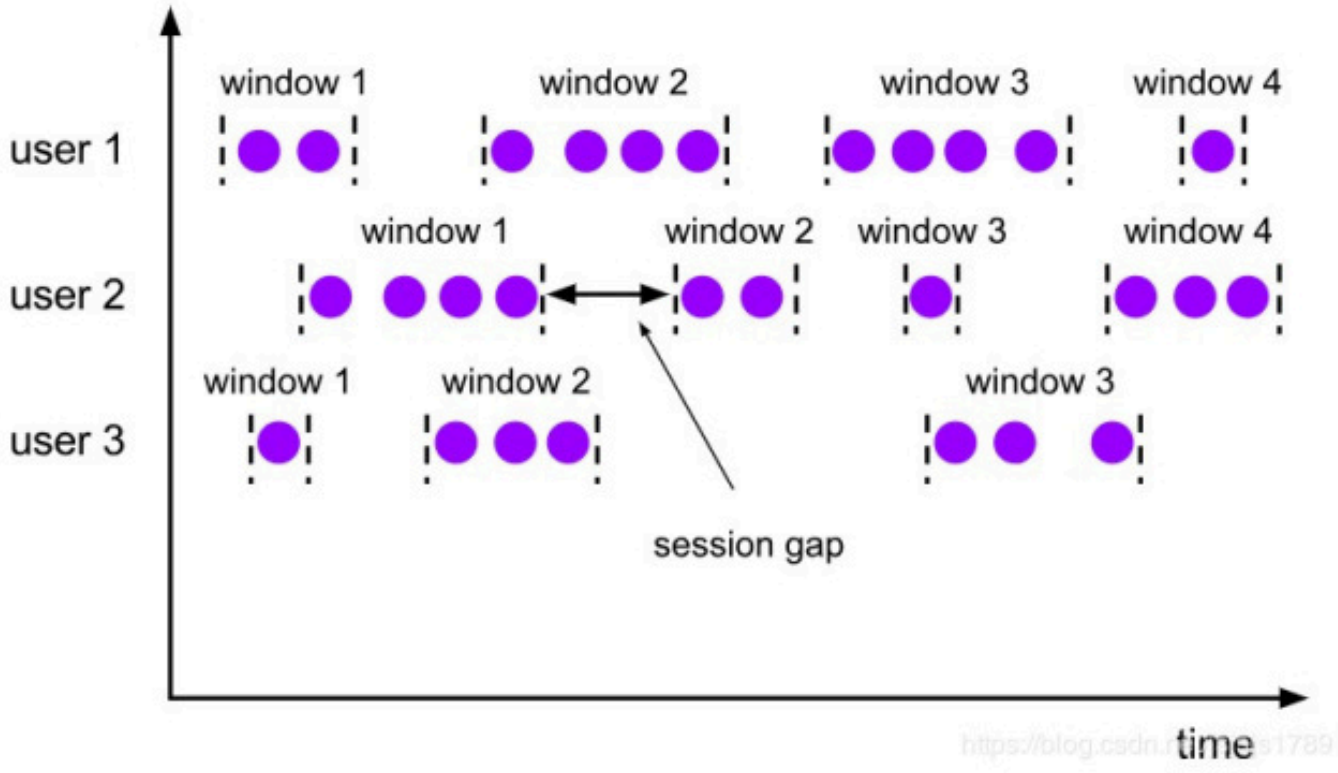
Tumbling Window



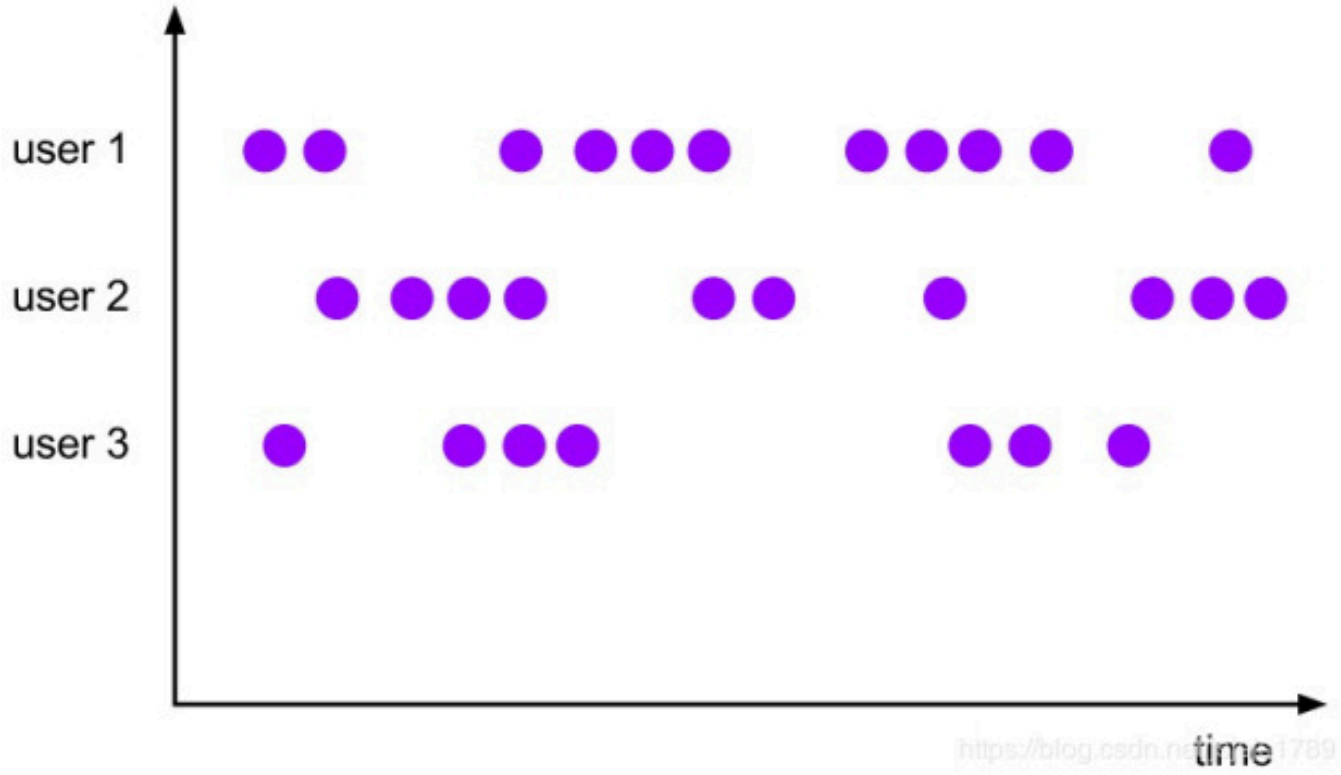
Sliding Window



Session Window



Global Window



window operator

evictor

evictor 主要用于做一些数据的自定义操作，可以在执行用户代码之前，也可以在执行用户代码之后，更详细的描述可以参考 org.apache.flink.streaming.api.windowing.evictors.Evictor 的 evicBefore 和 evicAfter 两个方法。

trigger

trigger 用来判断一个窗口是否需要被触发，每个 WindowAssigner 都自带一个默认的 trigger，如果默认的 trigger 不能满足你的需求，则可以自定义一个类，继承自 Trigger 即可，我们详细描述下 Trigger 的接口以及含义：

onElement() 每次往 window 增加一个元素的时候都会触发

onEventTime() 当 event-time timer 被触发的时候会调用

onProcessingTime() 当 processing-time timer 被触发的时候会调用

onMerge() 对两个 trigger 的 state 进行 merge 操作

clear() window 销毁的时候被调用

上面的接口中前三个会返回一个 TriggerResult，TriggerResult 有如下几种可能的选择：

CONTINUE 不做任何事情

FIRE 触发 window

PURGE 清空整个 window 的元素并销毁窗口

FIRE*AND*PURGE 触发窗口，然后销毁窗口

window code

```
1
2  package org.apache.flink.streaming.connectors.kafka;
3
4  import org.apache.flink.api.common.serialization.SimpleStringSchema;
5  import org.apache.flink.api.java.functions.KeySelector;
6  import org.apache.flink.contrib.streaming.state.RocksDBStateBackend;
7  import org.apache.flink.runtime.state.StateBackend;
8  import org.apache.flink.streaming.api.CheckpointingMode;
9  import org.apache.flink.streaming.api.TimeCharacteristic;
10 import org.apache.flink.streaming.api.environment.CheckpointConfig;
11 import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
12 import org.apache.flink.streaming.api.windowing.time.Time;
13 import org.slf4j.LoggerFactory;
14
15 import java.util.Properties;
16
17 /**
18  * @author shengjk1
19  * @date 2019/9/4
20  */
21 public class Main {
22     protected final static org.slf4j.Logger logger = LoggerFactory.getLogger(Main.class);
23
24     public static void main(String[] args) throws Exception {
25         final StreamExecutionEnvironment env =
   StreamExecutionEnvironment.getExecutionEnvironment();
26
27         env.setStreamTimeCharacteristic(TimeCharacteristic.ProcessingTime);
28
29         env.enableCheckpointing(60000, CheckpointingMode.EXACTLY_ONCE);
30         env.getCheckpointConfig().setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE);
31         env.getCheckpointConfig().setMinPauseBetweenCheckpoints(5000);
32         env.getCheckpointConfig().setCheckpointTimeout(60000);
33         env.getCheckpointConfig().setMaxConcurrentCheckpoints(5);
```

```
34
env.getCheckpointConfig().enableExternalizedCheckpoints(CheckpointConfig.ExternalizedCheckpointCleanup.RETAIN_ON_CANCELLATION);
35          env.getCheckpointConfig().setFailOnCheckpointingErrors(false);
36
37          env.setParallelism(1);
38
39          StateBackend backend =
40              new RocksDBStateBackend("file:////Users/iss/sourceCode/spark/flink/flink-connectors/flink-connector-kafka/src/test/java/org/apache/flink/streaming/connectors/kafka/checkpoints", true);
41          env.setStateBackend(backend);
42          Properties properties = new Properties();
43          properties.setProperty("bootstrap.servers", "bigdata-dev-mq:9092");
44          properties.setProperty("group.id", "test");
45
properties.setProperty(FlinkKafkaConsumerBase.KEY_PARTITION_DISCOVERY_INTERVAL_MILLIS, "1000");
46
47
48          FlinkKafkaConsumer<String> consumer = new FlinkKafkaConsumer<>("test", new SimpleStringSchema(), properties);
49          consumer.setStartFromEarliest();
50
51          env.addSource(consumer).uid("orderAndRegisterUserIdSource")
52                  .rebalance()
53                  .keyBy(new KeySelector<String, String>() {
54                      @Override
55                      public String getKey(String value) throws Exception {
56                          return value;
57                      }
58                  })
59                  .timeWindow(Time.seconds(2))
60                  .trigger(new CountAndTimeTrigger(2L)
61                  .process(new ProcessWindowFunctionImp()).uid("process");
62
63
64          // execute program
65          env.execute("realTimeDataWareHouse");
66      }
67  }
```

其中的 CountAndTimeTrigger 可参考 Flink 自定义触发器实现带超时时间的 countAndTimeTrigger

window 原理剖析

首先，当此程序开始消费消息时(可参考 一文搞定 Flink 消费消息的全流程) 进入 WindowOperator processElement 方法

```
1
2  // window operator 的 processElement
3      public void processElement(StreamRecord<IN> element) throws Exception {
```

```
4       final Collection<W> elementWindows = windowAssigner.assignWindows(
5           element.getValue(), element.getTimestamp(), windowAssignerContext);
6
7       //if element is handled by none of assigned elementWindows
8       boolean isSkippedElement = true;
9
10      final K key = this.<K>getKeyedStateBackend().getCurrentKey();
11
12      if (windowAssigner instanceof MergingWindowAssigner) {
13          MergingWindowSet<W> mergingWindows = getMergingWindowSet();
14
15          for (W window: elementWindows) {
16
17              // adding the new window might result in a merge, in that case the
actualWindow
18              // is the merged window and we work with that. If we don't merge then
19              // actualWindow == window
20              W actualWindow = mergingWindows.addWindow(window, new
MergingWindowSet.MergeFunction<W>() {
21                  @Override
22                  public void merge(W mergeResult,
23                          Collection<W> mergedWindows, W stateWindowResult,
24                          Collection<W> mergedStateWindows) throws Exception {
25
26                      if ((windowAssigner.isEventTime() && mergeResult.maxTimestamp() +
allowedLateness <= internalTimerService.currentWatermark())) {
27                          throw new UnsupportedOperationException("The end timestamp of an
" +
28                                  "event-time window cannot become earlier than the
current watermark " +
29                                  "by merging. Current watermark: " +
internalTimerService.currentWatermark() +
30                                  " window: " + mergeResult);
31                      } else if (!windowAssigner.isEventTime() &&
mergeResult.maxTimestamp() <= internalTimerService.currentProcessingTime()) {
32                          throw new UnsupportedOperationException("The end timestamp of a
" +
33                                  "processing-time window cannot become earlier than the
current processing time " +
34                                  "by merging. Current processing time: " +
internalTimerService.currentProcessingTime() +
35                                  " window: " + mergeResult);
36                      }
37
38                      triggerContext.key = key;
39                      triggerContext.window = mergeResult;
40
41                      triggerContext.onMerge(mergedWindows);
42
43                      for (W m: mergedWindows) {
44                          triggerContext.window = m;
45                          triggerContext.clear();
46                          deleteCleanupTimer(m);
47                      }
48
49                      // merge the merged state windows into the newly resulting state
```

```
window
                    windowMergingState.mergeNamespaces(stateWindowResult, mergedStateWindows);
                }
            });

            // drop if the window is already late
            if (isWindowLate(actualWindow)) {
                mergingWindows.retireWindow(actualWindow);
                continue;
            }
            isSkippedElement = false;

            W stateWindow = mergingWindows.getStateWindow(actualWindow);
            if (stateWindow == null) {
                throw new IllegalStateException("Window " + window + " is not in in-flight window set.");
            }

            windowState.setCurrentNamespace(stateWindow);
            windowState.add(element.getValue());

            triggerContext.key = key;
            triggerContext.window = actualWindow;

            TriggerResult triggerResult = triggerContext.onElement(element);

            if (triggerResult.isFire()) {
                // RockdbListState  RocksDBReducingState
                ACC contents = windowState.get();
                if (contents == null) {
                    continue;
                }
                emitWindowContents(actualWindow, contents);
            }

            if (triggerResult.isPurge()) {
                windowState.clear();
            }
            registerCleanupTimer(actualWindow);
        }

        // need to make sure to update the merging state in state
        mergingWindows.persist();
    } else {
        for (W window: elementWindows) {

            // drop if the window is already late
            if (isWindowLate(window)) {
                continue;
            }
            isSkippedElement = false;

            windowState.setCurrentNamespace(window);
            //数据过来之后会先存入 windowState 直至 window fire
            windowState.add(element.getValue());
```

```
103
104                 triggerContext.key = key;
105                 triggerContext.window = window;
106
107                 //调用用户定义的 onElement 代码
108                 TriggerResult triggerResult = triggerContext.onElement(element);
109                 //当触发窗口时，从 windowState 中获取数据，在本样例中 windowState 为
    RocksDBListState
110                 if (triggerResult.isFire()) {
111                     //RocksDBListState RocksDBReducingState
112                     //
113                     ACC contents = windowState.get();
114                     if (contents == null) {
115                         continue;
116                     }
117                     //当窗口触发时，会将 window 中数据发送到下游,调用用户的 process 方法。
118                     emitWindowContents(window, contents);
119                 }
120
121                 if (triggerResult.isPurge()) {
122                     windowState.clear();
123                 }
124                 // 注册 timer，其实就是定时调度任务。底层通过
    ScheduledThreadPoolExecutor.schedule(...)来实现的
125                 // 每个窗口中的每个 key 会有且仅有一个 timer( 判断方式的一部分是通过 map 来实现的)
126                 registerCleanupTimer(window);
127             }
128         }
```

关于 window 消息顺序性问题，可以参考 一文搞懂 Flink window 元素的顺序问题

当注册的 timer 到期之后开始调用 onProcessingTime

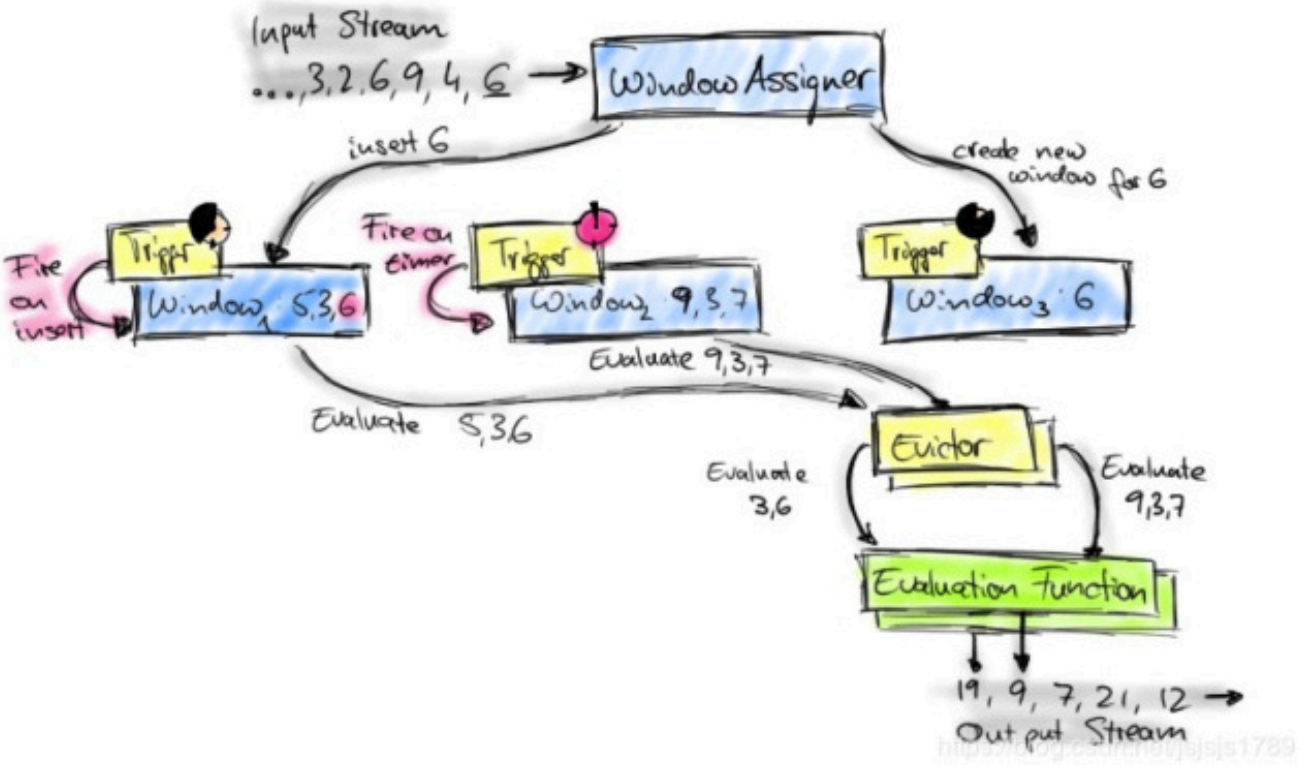
```
1
2 // 这个是通过 timer 来调用的,
3     // processElement 的时候 registerCleanupTimer(window) 会创建相应的 timer
4     public void onProcessingTime(InternalTimer<K, W> timer) throws Exception {
5         triggerContext.key = timer.getKey();
6         triggerContext.window = timer.getNamespace();
7
8         MergingWindowSet<W> mergingWindows;
9
10        if (windowAssigner instanceof MergingWindowAssigner) {
11            mergingWindows = getMergingWindowSet();
12            W stateWindow = mergingWindows.getStateWindow(triggerContext.window);
13            if (stateWindow == null) {
14                // Timer firing for non-existent window, this can only happen if a
15                // trigger did not clean up timers. We have already cleared the merging
16                // window and therefore the Trigger state, however, so nothing to do.
17                return;
18            } else {
19                windowState.setCurrentNamespace(stateWindow);
20            }
21        } else {
22            windowState.setCurrentNamespace(triggerContext.window);
23            mergingWindows = null;
24        }
```

```
25
26          TriggerResult triggerResult = triggerContext.onProcessingTime(timer.getTimestamp());
27
28          if (triggerResult.isFire()) {
29              ACC contents = windowState.get();
30              if (contents != null) {
31                  emitWindowContents(triggerContext.window, contents);
32              }
33          }
34
35          if (triggerResult.isPurge()) {
36              windowState.clear();
37          }
38
39          if (!windowAssigner.isEventTime() && isCleanupTime(triggerContext.window,
    timer.getTimestamp())) {
40              // 会清空所有的 state
41              // 先 windowState.clear() 调用用户定义的 clear 方法，然后再清除 windowContext 内部的状
    态：
42              // 仅仅是通过 onProcessingTime or onEventTime method fire window 才可能会触发
    clearAllState 操作
43              // 否则会可以理解为还是一个窗口虽然 fire 了。
44              // 先增量增量的 fire 然后再全量的 fire ( onProcessingTime and  onEventTime  导致的
    fire ，未指定 purge)
45              clearAllState(triggerContext.window, windowState, mergingWindows);
46          }
47
48          if (mergingWindows != null) {
49              // need to make sure to update the merging state in state
50              mergingWindows.persist();
51          }
52      }
```

需要注意的是 window 跟 key 有关

总结

整个 window 流程



发布于: 2020 年 06 月 13 日　　阅读数: 38