**Window Operations（窗口操作）**

Spark还提供了窗口的计算，它允许你使用一个滑动窗口应用在数据变换中。下图说明了该滑动窗口。



如图所示，每个时间窗口在一个个DStream中划过，每个DSteam中的RDD进入Window中进行合并，操作时生成为窗口化DSteam的RDD。在上图中，该操作被应用在过去的3个时间单位的数据，和划过了2个时间单位。这说明任何窗口操作都需要指定2个参数。

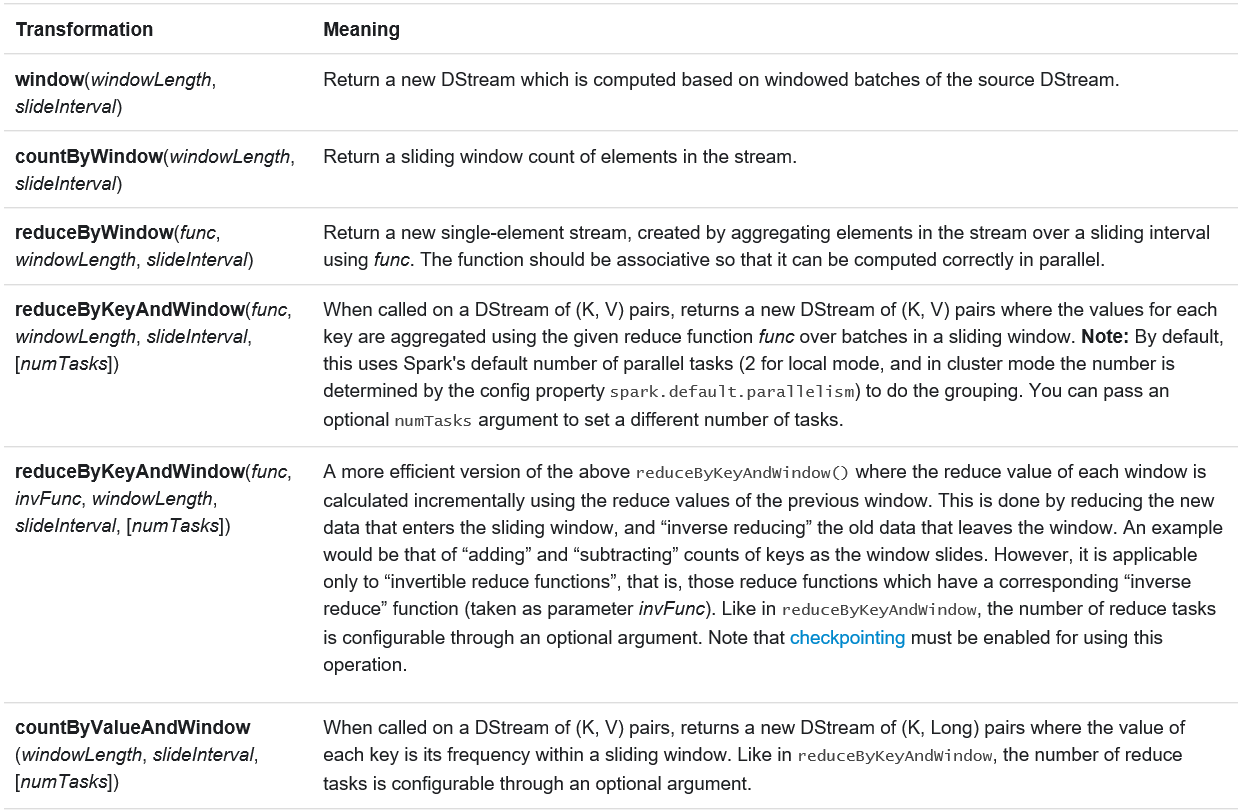
window length（窗口长度）：窗口的持续时间（上图为3个时间单位）

sliding interval （滑动间隔）- 窗口操作的时间间隔（上图为2个时间单位）。

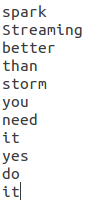
上面的2个参数的大小，必须是接受产生一个DStream时间的倍数

让我们用一个例子来说明窗口操作。比如说，你想用以前的WordCount的例子，来计算最近30s的数据的中的单词数，10S接受为一个DStream。为此，我们要用reduceByKey操作来计算最近30s数据中每一个DSteam中关于（word，1）的pair操作。它可以用reduceByKeyAndWindow操作来实现。

一些常见的窗口操作如下。所有这些操作都需要两个参数--- window length（窗口长度）和sliding interval（滑动间隔）。



-------------------------实验数据--------------------------------



（每秒在其中随机抽取一个，作为Socket端的输入）

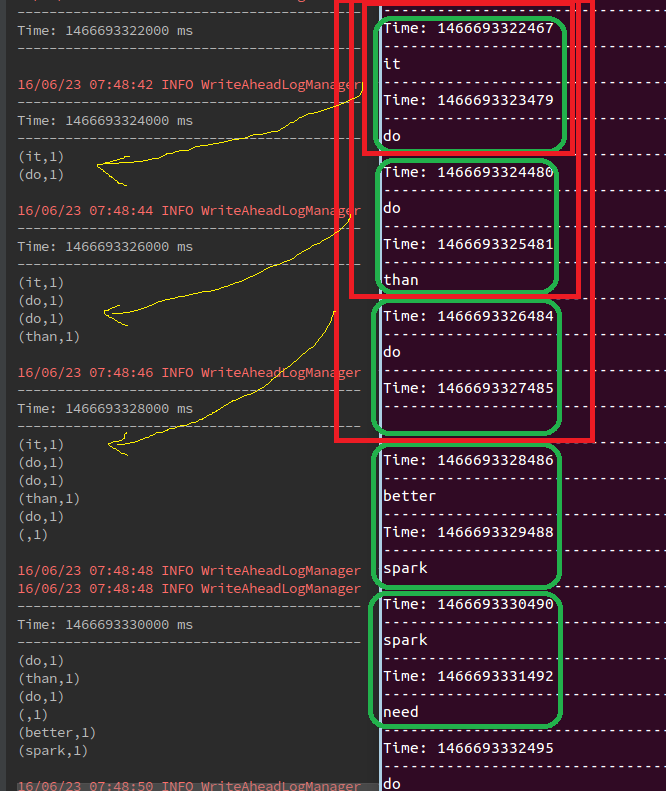
-----------------------window操作---------------------------------

*//输入:窗口长度（隐：输入的滑动窗口长度为形成Dstream的时间）*

*//输出：返回一个DStream,這个DStream包含這个滑动窗口下的全部元素***def** window(windowDuration: Duration): DStream[T] = window(windowDuration, **this**.slideDuration)  
  
*//输入:窗口长度和滑动窗口长度*

*//输出：返回一个DStream,這个DStream包含這个滑动窗口下的全部元素***def** window(windowDuration: Duration, slideDuration: Duration): DStream[T] = ssc.withScope {  
 **new** WindowedDStream(**this**, windowDuration, slideDuration)  
}

实验（实验代码见附录实验1）



--------------------**reduceByKeyAndWindow**操作--------------------------------

*/\*\*通过对每个滑动过来的窗口应用一个reduceByKey的操作，返回一个DSream，有点像*

*\** `*DStream.reduceByKey()*,但是只是這个函数只是应用在*滑动过来*的窗口，hash分区是采用spark集群

*\** 默认的分区树

*\** ***@param reduceFunc 从左到右的****reduce 函数*

*\** ***@param windowDuration 窗口时间***

*\** ***滑动窗口默认是1个batch interval***

***\* 分区数是是RDD默认（depend on spark集群core）***

*\*/***def** reduceByKeyAndWindow(  
 reduceFunc: (V, V) => V,  
 windowDuration: Duration  
 ): DStream[(K, V)] = ssc.withScope {  
 reduceByKeyAndWindow(reduceFunc, windowDuration, self.slideDuration, defaultPartitioner())  
}  
  
*/\*\*通过对每个滑动过来的窗口应用一个reduceByKey的操作，返回一个DSream，有点像*

*\** `*DStream.reduceByKey()*,但是只是這个函数只是应用在*滑动过来*的窗口，hash分区是采用spark集群

*\** 默认的分区树

*\** ***@param reduceFunc 从左到右的****reduce 函数*

*\** ***@param windowDuration 窗口时间***

*\** ***@param slideDuration 滑动时间***

*\*/***def** reduceByKeyAndWindow(  
 reduceFunc: (V, V) => V,  
 windowDuration: Duration,  
 slideDuration: Duration  
 ): DStream[(K, V)] = ssc.withScope {  
 reduceByKeyAndWindow(reduceFunc, windowDuration, slideDuration, defaultPartitioner())  
}  
  
 */\*\*通过对每个滑动过来的窗口应用一个reduceByKey的操作，返回一个DSream，有点像*

*\** `*DStream.reduceByKey()*,但是只是這个函数只是应用在*滑动过来*的窗口，hash分区是采用spark集群

*\** 默认的分区树 *\** ***@param reduceFunc 从左到右的****reduce 函数*

*\** ***@param windowDuration 窗口时间***

*\** ***@param slideDuration 滑动时间***

*\** ***@param numPartitions*** *每个RDD的分区数.  
 \*/***def** reduceByKeyAndWindow(  
 reduceFunc: (V, V) => V,  
 windowDuration: Duration,  
 slideDuration: Duration,  
 numPartitions: Int  
 ): DStream[(K, V)] = ssc.withScope {  
 reduceByKeyAndWindow(reduceFunc, windowDuration, slideDuration,  
 defaultPartitioner(numPartitions))  
}  
  
*/\*\*  
 /\*\*通过对每个滑动过来的窗口应用一个reduceByKey的操作，返回一个DSream，有点像*

*\* `DStream.reduceByKey(),但是只是這个函数只是应用在滑动过来的窗口，hash分区是采用spark集群*

*\* 默认的分区树*

*\* @param reduceFunc 从左到右的reduce 函数*

*\* @param windowDuration 窗口时间*

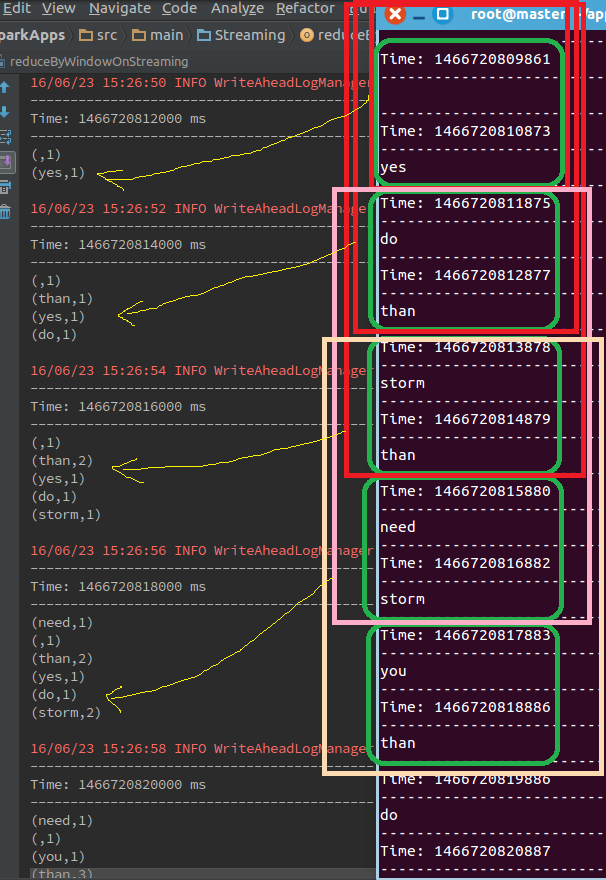
*\* @param slideDuration 滑动时间*

*\* @param numPartitions 每个RDD的分区数.  
 \** ***@param partitioner 设置每个partition的分区数***

*\*/***def** reduceByKeyAndWindow(  
 reduceFunc: (V, V) => V,  
 windowDuration: Duration,  
 slideDuration: Duration,  
 partitioner: Partitioner  
 ): DStream[(K, V)] = ssc.withScope {  
 self.reduceByKey(reduceFunc, partitioner)  
 .window(windowDuration, slideDuration)  
 .reduceByKey(reduceFunc, partitioner)  
}  
  
*/\*\*  
 \*通过对每个滑动过来的窗口应用一个reduceByKey的操作.同时对old RDDs进行了****invReduceFunc操作*** *\* hash分区是采用spark集群，默认的分区树  
 \** ***@param reduceFunc****从左到右的reduce 函数  
 \** ***@param invReduceFunc*** *inverse reduce function; such that for all y, invertible x:  
 \** `*invReduceFunc(reduceFunc(x, y), x) = y*`  
 *\** ***@param windowDuration****窗口时间*

*\** ***@param slideDuration*** *滑动时间  
 \** ***@param filterFunc 来赛选一定条件的 key-value 对的*** *\*/***def** reduceByKeyAndWindow(  
 reduceFunc: (V, V) => V,  
 invReduceFunc: (V, V) => V,  
 windowDuration: Duration,  
 slideDuration: Duration = self.slideDuration,  
 numPartitions: Int = ssc.*sc*.defaultParallelism,  
 filterFunc: ((K, V)) => Boolean = **null** ): DStream[(K, V)] = ssc.withScope {  
 reduceByKeyAndWindow(  
 reduceFunc, invReduceFunc, windowDuration,  
 slideDuration, defaultPartitioner(numPartitions), filterFunc  
 )  
}  
  
*/\*\*  
\*通过对每个滑动过来的窗口应用一个reduceByKey的操作.同时对old RDDs进行了****invReduceFunc操作*** *\* hash分区是采用spark集群，默认的分区树  
 \** ***@param reduceFunc****从左到右的reduce 函数  
 \** ***@param invReduceFunc*** *inverse reduce function; such that for all y, invertible x:  
 \** `*invReduceFunc(reduceFunc(x, y), x) = y*`  
 *\** ***@param windowDuration****窗口时间*

*\** ***@param slideDuration*** *滑动时间  
 \** ***@param partitioner*** *每个RDD的分区数.  
 \** ***@param filterFunc 来赛选一定条件的 key-value 对的*** *\*/***def** reduceByKeyAndWindow(  
 reduceFunc: (V, V) => V,  
 invReduceFunc: (V, V) => V,  
 windowDuration: Duration,  
 slideDuration: Duration,  
 partitioner: Partitioner,  
 filterFunc: ((K, V)) => Boolean  
 ): DStream[(K, V)] = ssc.withScope {  
  
 **val** cleanedReduceFunc = ssc.*sc*.clean(reduceFunc)  
 **val** cleanedInvReduceFunc = ssc.*sc*.clean(invReduceFunc)  
 **val** cleanedFilterFunc = **if** (filterFunc != **null**) *Some*(ssc.*sc*.clean(filterFunc)) **else** None  
 **new** ReducedWindowedDStream[K, V](  
 self, cleanedReduceFunc, cleanedInvReduceFunc, cleanedFilterFunc,  
 windowDuration, slideDuration, partitioner  
 )  
}



這里出现了invReduceFunc函数這个函数有点特别，现在通过分析ReducedWindowedDStream這个类内部来进行说明：



------------------**reduceByWindow**操作---------------------------

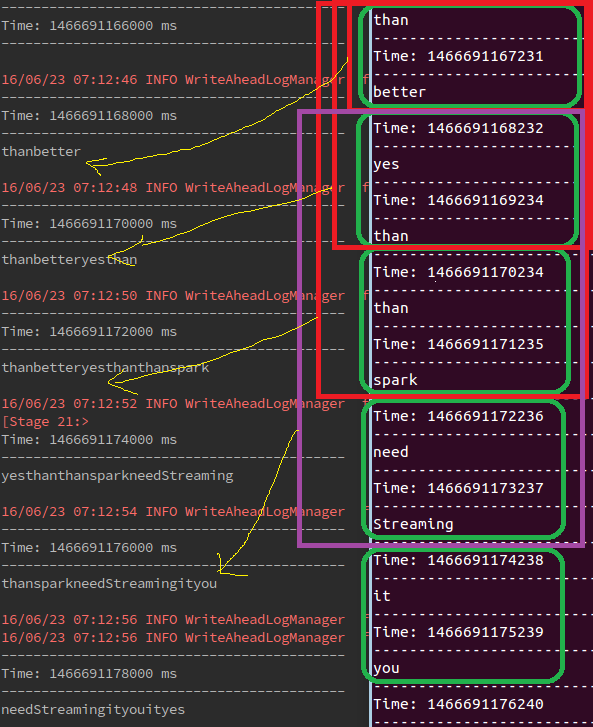
源码分析

*//输入：reduceFunc、窗口长度、滑动长度*

*//输出：（a,b）为从几个从左到右一次取得两个元素*

*//（，a,b）进入reduceFunc,***def** reduceByWindow(  
 reduceFunc: (T, T) => T,  
 windowDuration: Duration,  
 slideDuration: Duration  
 ): DStream[T] = ssc.withScope {  
 **this**.reduce(reduceFunc).window(windowDuration, slideDuration).reduce(reduceFunc)  
}  
*/\*\**

*\*输入****reduceFunc，invReduceFunc，****窗口长度、滑动长度  
 \*/***def** reduceByWindow(  
 reduceFunc: (T, T) => T,  
 invReduceFunc: (T, T) => T,  
 windowDuration: Duration,  
 slideDuration: Duration  
 ): DStream[T] = ssc.withScope {  
 **this**.map((1, \_))  
 .reduceByKeyAndWindow(reduceFunc, invReduceFunc, windowDuration, slideDuration, 1)  
 .map(\_.\_2)  
}



----------------countByWindow操作---------------------------------

源码分析

*/\*\**

*\* 输入 窗口长度和滑动长度，返回窗口内的元素数量  
 \** ***@param windowDuration 窗口长度***

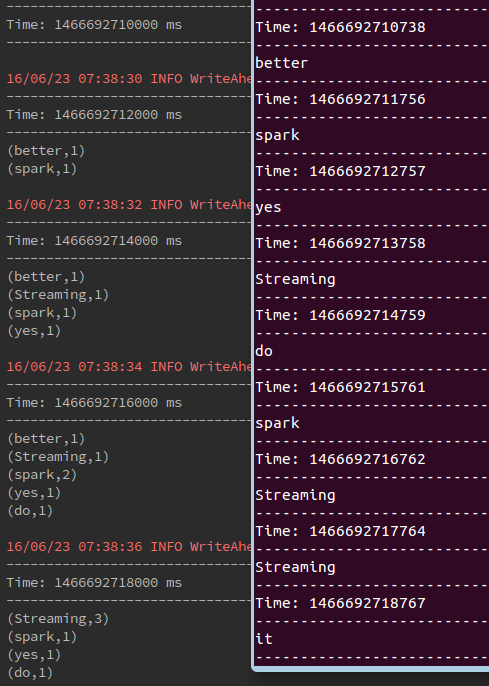
*\** ***@param slideDuration 滑动长度***

*\*/***def** countByWindow(  
 windowDuration: Duration,  
 slideDuration: Duration): DStream[Long] = ssc.withScope {  
 **this**.map(\_ => 1L).reduceByWindow(\_ + \_, \_ - \_, windowDuration, slideDuration)  
//窗口下的DStream进行map操作，把每个元素变为1之后进行reduceByWindow操作

}

-------------------------------------- countByValueAndWindow-------------

*/\*\*  
\*输入 窗口长度、滑动时间、RDD分区数（默认分区是等于并行度）  
 \** ***@param windowDuration*** *width of the window; must be a multiple of this DStream's  
 \* batching interval  
 \** ***@param slideDuration*** *sliding interval of the window (i.e., the interval after which  
 \* the new DStream will generate RDDs); must be a multiple of this  
 \* DStream's batching interval  
 \** ***@param numPartitions*** *number of partitions of each RDD in the new DStream.  
 \*/***def** countByValueAndWindow(  
 windowDuration: Duration,  
 slideDuration: Duration,  
 numPartitions: Int = ssc.*sc*.defaultParallelism)  
 (**implicit** ord: Ordering[T] = **null**)  
 : DStream[(T, Long)] = ssc.withScope {  
 **this**.map((\_, 1L)).reduceByKeyAndWindow(  
 (x: Long, y: Long) => x + y,  
 (x: Long, y: Long) => x - y,  
 windowDuration,  
 slideDuration,  
 numPartitions,  
 (x: (T, Long)) => x.\_2 != 0L  
 )  
}



附录

实验1

import org.apache.log4j.{Level, Logger}

import org.apache.spark.streaming.{Seconds, StreamingContex，t}

import org.apache.spark.{SparkConf, SparkContext}

/\*\*

\* Created by legotime on 6/1/16.

\*/

object windowOnStreaming {

def main(args: Array[String]) {

/\*\*

\* this is test of Streaming operations-----window

\*/

Logger.getLogger("org.apache.spark").setLevel(Level.ERROR)

Logger.getLogger("org.eclipse.jetty.Server").setLevel(Level.OFF)

val conf = new SparkConf().setAppName("the Window operation of SparK Streaming").setMaster("local[2]")

val sc = new SparkContext(conf)

val ssc = new StreamingContext(sc,Seconds(3))

//set the Checkpoint directory

ssc.checkpoint("/Res")

//get the socket Streaming data

val socketStreaming = ssc.socketTextStream("master",9999)

val data = socketStreaming.map(x =>(x,1))

//def window(windowDuration: Duration): DStream[T]

val getedData1 = data.window(Seconds(9))

println("windowDuration only : ")

getedData1.print()

//ssame as

// def window(windowDuration: Duration, slideDuration: Duration): DStream[T]

//val getedData2 = data.window(Seconds(9),Seconds(3))

//println("Duration and SlideDuration : ")

//getedData2.print()

ssc.start()

ssc.awaitTermination()

}

}