Spark 2.0介绍：Spark SQL中的Time Window使用

《Spark 2.0技术预览：更容易、更快速、更智能》文章中简单地介绍了Spark 2.0带来的新技术等。Spark 2.0是Apache Spark的下一个主要版本。此版本在架构抽象、API以及平台的类库方面带来了很大的变化，为该框架明年的发展奠定了方向，所以了解Spark 2.0的一些特性对我们能够使用它有着非常重要的作用。本博客将对Spark 2.0进行一序列的介绍（参见Spark 2.0分类），欢迎关注。

Spark SQL中Window API

Spark SQL中的window API是从1.4版本开始引入的，以便支持更智能的分组功能。这个功能对于那些有SQL背景的人来说非常有用；但是在Spark 1.x中，window API一大缺点就是无法使用时间来创建窗口。时间在诸如金融、电信等领域有着非常重要的角色，基于时间来理解数据变得至关重要。

不过值得高兴的是，在Spark 2.0中，window API内置也支持time windows！Spark SQL中的time windows和Spark Streaming中的time windows非常类似。在这篇文章中，我将介绍如何在Spark SQL中使用time windows。

时间序列数据

在我们介绍如何使用time window之前，我们先来准备一份时间序列数据。本文将使用Apple

公司从1980年到2016年期间的股票交易信息。如下（完整的数据点击这里获取）：

```
Date,Open,High,Low,Close,Volume,Adj Close
2016-7-11,96.75,97.650002,96.730003,96.980003,23298900,96.980003
2016-7-8,96.489998,96.889999,96.050003,96.68,28855800,96.68
2016-7-7,95.699997,96.5,95.620003,95.940002,24280900,95.940002
2016-7-6,94.599998,95.660004,94.370003,95.529999,30770700,95.529999
2016-7-5,95.389999,95.400002,94.459999,95.040001,27257000,95.040001
2016-7-1,95.489998,96.470001,95.330002,95.889999,25872300,95.889999
2016-6-30,94.440002,95.769997,94.300003,95.599998,35836400,95.599998
2016-6-29,93.970001,94.550003,93.629997,94.400002,36531000,94.400002
2016-6-28,92.900002,93.660004,92.139999,93.589996,40444900,93.589996
2016-6-27,93,93.050003,91.5,92.040001,45489600,92.040001
2016-6-24,92.910004,94.660004,92.650002,93.400002,75311400,93.400002
2016-6-23,95.940002,96.290001,95.25,96.099998,32240200,96.099998
2016-6-22,96.25,96.889999,95.349998,95.550003,28971100,95.550003
2016-6-21,94.940002,96.349998,94.68,95.910004,35229500,95.910004
2016-6-20,96,96.57,95.029999,95.099998,33942300,95.099998
2016-6-17,96.620003,96.650002,95.300003,95.330002,60595000,95.330002
2016-6-16,96.449997,97.75,96.07,97.550003,31236300,97.550003
2016-6-15,97.82,98.410004,97.029999,97.139999,29445200,97.139999
2016-6-14,97.32,98.480003,96.75,97.459999,31931900,97.459999
2016-6-13,98.690002,99.120003,97.099998,97.339996,38020500,97.339996
2016-6-10,98.529999,99.349998,98.480003,98.830002,31712900,98.830002
2016-6-9,98.5,99.989998,98.459999,99.650002,26601400,99.650002
2016-6-8,99.019997,99.559998,98.68,98.940002,20848100,98.940002
2016-6-7,99.25,99.870003,98.959999,99.029999,22409500,99.029999
2016-6-6,97.989998,101.889999,97.550003,98.629997,23292500,98.629997
2016-6-3,97.790001,98.269997,97.449997,97.919998,28062900,97.919998
2016-6-2,97.599998,97.839996,96.629997,97.720001,40004100,97.720001
2016-6-1,99.019997,99.540001,98.330002,98.459999,29113400,98.459999
2016-5-31,99.599998,100.400002,98.82,99.860001,42084800,99.860001
```

股票数据一共有六列，但是这里我们仅关心Date和Close两列，它们分别代表股票交易时间和当天收盘的价格。

将时间序列数据导入到DataFrame中

我们有了样本数据之后，需要将它导入到DataFrame中以便下面的计算。所有的time window API需要一个类型为timestamp的列。我们可以使用spark-csv工具包来解析上面的Apple股票数据（csv格式），这个工具可以自动推断时间类型的数据并自动创建好模式。代码如下：

```
scala> val stocksDF = spark.read.option("header","true").option("inferSchema","true").csv("file:
///user/iteblog/applestock.csv")
stocksDF: org.apache.spark.sql.DataFrame = [Date: timestamp, Open: double ... 5 more fields]
```


计算2016年Apple股票周平均收盘价格

现在我们已经有了初始化好的数据，所以我们可以进行一些基于时间的窗口分析。在本例中我们将计算2016年Apple公司每周股票的收盘价格平均值。下面将一步一步进行介绍。

步骤一：找出2016年的股票交易数据

因为我们仅仅需要2016年的交易数据，所以我们可以对原始数据进行过滤，代码如下：

```
/**
 * User: 过往记忆
 * Date: 2016年07月12日
 * Time: 下午23:45
 * bolg: https://www.iteblog.com
 * 本文地址：https://www.iteblog.com/archives/1705
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */

scala> val stocks2016 = stocksDF.filter("year(Date)==2016")
stocks2016: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Date: timestamp, Open
: double ... 5 more fields]
```

上面代码片段我们使用了内置的year函数来提取出日期中的年。

步骤二：计算平均值

现在我们需要对每个星期创建一个窗口，这种类型的窗口通常被称为tumbling window，代码片段如下：

```
scala> val tumblingWindowDS = stocks2016.groupBy(window(stocks2016.col("Date"),"1 week")
).agg(avg("Close").as("weekly_average"))
tumblingWindowDS: org.apache.spark.sql.DataFrame = [window: struct<start: timestamp, end:
 timestamp>, weekly_average: double]
```

上面代码中展示了如何使用 time window API。window一般在group by语句中使用。window方法的第一个参数指定了时间所在的列；第二个参数指定了窗口的持续时间(duration)，它的单位可以是seconds、 minutes、 hours、 days或者weeks。创建好窗口之后，我们可以计算平均值。

步骤三：打印window的值

我们可以打印出window中的值，我们先定义好打印的公共函数，代码片段如下：

```
/**
 * User: 过往记忆
 * Date: 2016年07月12日
 * Time: 下午23:45
 * bolg: https://www.iteblog.com
 * 本文地址：https://www.iteblog.com/archives/1705
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */

def printWindow(windowDF:DataFrame, aggCol:String) ={
   windowDF.sort("window.start").
   select("window.start","window.end",s"$aggCol").
   show(truncate = false)
}
```

然后我们打印出tumblingWindowDS中的值：

```
printWindow(tumblingWindowDS,"weekly_average")
+--------------------+--------------------+------------------+
|start               |end                 |weekly_average    |
+--------------------+--------------------+------------------+
|2015-12-31 08:00:00.0|2016-01-07 08:00:00.0|101.30249774999999|
|2016-01-07 08:00:00.0|2016-01-14 08:00:00.0|98.47199859999999 |
|2016-01-14 08:00:00.0|2016-01-21 08:00:00.0|96.72000125000001 |
|2016-01-21 08:00:00.0|2016-01-28 08:00:00.0|97.6719984        |
|2016-01-28 08:00:00.0|2016-02-04 08:00:00.0|96.239999         |
|2016-02-04 08:00:00.0|2016-02-11 08:00:00.0|94.39799819999999 |
|2016-02-11 08:00:00.0|2016-02-18 08:00:00.0|96.2525005        |
|2016-02-18 08:00:00.0|2016-02-25 08:00:00.0|96.09400000000001 |
|2016-02-25 08:00:00.0|2016-03-03 08:00:00.0|99.276001         |
|2016-03-03 08:00:00.0|2016-03-10 08:00:00.0|101.64000100000001|
|2016-03-10 08:00:00.0|2016-03-17 08:00:00.0|104.226001        |
|2016-03-17 08:00:00.0|2016-03-24 08:00:00.0|106.0699996       |
|2016-03-24 08:00:00.0|2016-03-31 08:00:00.0|107.8549995       |
|2016-03-31 08:00:00.0|2016-04-07 08:00:00.0|110.08399979999999|
|2016-04-07 08:00:00.0|2016-04-14 08:00:00.0|110.4520004       |
|2016-04-14 08:00:00.0|2016-04-21 08:00:00.0|107.46800060000001|
|2016-04-21 08:00:00.0|2016-04-28 08:00:00.0|101.5520004       |
|2016-04-28 08:00:00.0|2016-05-05 08:00:00.0|93.9979994        |
|2016-05-05 08:00:00.0|2016-05-12 08:00:00.0|92.35599959999999 |
|2016-05-12 08:00:00.0|2016-05-19 08:00:00.0|93.3299974        |
+--------------------+--------------------+------------------+
only showing top 20 rows
```

上面的输出按照window.start进行了排序，这个字段标记了窗口的开始时间。上面的输出你可能已经看到了第一行的开始时间是2015-12-31，结束时间是2016-01-07。但是你从原始数据可以得到：2016年Apple公司的股票交易信息是从2016-01-04开始的；原因是2016-01-01是元旦，而2016-01-02和2016-01-03正好是周末，期间没有股票交易。

我们可以手动指定窗口的开始时间来解决这个问题。

带有开始时间的Time window

在前面的示例中，我们使用的是tumbling window。为了能够指定开始时间，我们需要使用sliding window（滑动窗口）。到目前为止，没有相关API来创建带有开始时间的tumbling window，但是我们可以通过将窗口时间(window duration)和滑动时间(slide

duration)设置成一样来创建带有开始时间的tumbling window。代码如下：

```
/**
 * User: 过往记忆
 * Date: 2016年07月12日
 * Time: 下午23:45
 * bolg: https://www.iteblog.com
 * 本文地址：https://www.iteblog.com/archives/1705
 * 过往记忆博客，专注于hadoop、hive、spark、shark、flume的技术博客，大量的干货
 * 过往记忆博客微信公共帐号：iteblog_hadoop
 */

val iteblogWindowWithStartTime = stocks2016.groupBy(window(stocks2016.col("Date"),"1 week","1 week", "4 days")).agg(avg("Close").as("weekly_average"))
```

上面的示例中，4 days参数就是开始时间的偏移量；前两个参数分别代表窗口时间和滑动时间，我们打印出这个窗口的内容：

```
printWindow(iteblogWindowWithStartTime,"weekly_average")
+--------------------+--------------------+------------------+
|start               |end                 |weekly_average    |
+--------------------+--------------------+------------------+
|2015-12-28 08:00:00.0|2016-01-04 08:00:00.0|105.349998        |
|2016-01-04 08:00:00.0|2016-01-11 08:00:00.0|99.0699982        |
|2016-01-11 08:00:00.0|2016-01-18 08:00:00.0|98.49999799999999 |
|2016-01-18 08:00:00.0|2016-01-25 08:00:00.0|98.1220016        |
|2016-01-25 08:00:00.0|2016-02-01 08:00:00.0|96.2539976        |
|2016-02-01 08:00:00.0|2016-02-08 08:00:00.0|95.29199960000001 |
|2016-02-08 08:00:00.0|2016-02-15 08:00:00.0|94.2374975        |
|2016-02-15 08:00:00.0|2016-02-22 08:00:00.0|96.7880004        |
|2016-02-22 08:00:00.0|2016-02-29 08:00:00.0|96.23000160000001 |
|2016-02-29 08:00:00.0|2016-03-07 08:00:00.0|101.53200079999999|
|2016-03-07 08:00:00.0|2016-03-14 08:00:00.0|101.6199998       |
|2016-03-14 08:00:00.0|2016-03-21 08:00:00.0|105.63600160000001|
|2016-03-21 08:00:00.0|2016-03-28 08:00:00.0|105.92749950000001|
|2016-03-28 08:00:00.0|2016-04-04 08:00:00.0|109.46799940000001|
|2016-04-04 08:00:00.0|2016-04-11 08:00:00.0|109.39799980000001|
|2016-04-11 08:00:00.0|2016-04-18 08:00:00.0|110.3820004       |
|2016-04-18 08:00:00.0|2016-04-25 08:00:00.0|106.15400079999999|
|2016-04-25 08:00:00.0|2016-05-02 08:00:00.0|96.8759994        |
|2016-05-02 08:00:00.0|2016-05-09 08:00:00.0|93.6240004        |
|2016-05-09 08:00:00.0|2016-05-16 08:00:00.0|92.13399799999999 |
+--------------------+--------------------+------------------+
```

only showing top 20 rows

从上面的结果可以看出，我们已经有了一个从2016-01-04的结果；不过结果中还有2015年的数据。原因是我们的开始时间是4
days，2016-01-04之前的一周数据也会被显示出，我们可以使用filter来过滤掉那行数据：

```
val filteredWindow = iteblogWindowWithStartTime.filter("year(window.start)=2016")
```

现在来看看输出的结果：

```
printWindow(filteredWindow,"weekly_average")
+--------------------+--------------------+------------------+
|start               |end                 |weekly_average    |
+--------------------+--------------------+------------------+
|2016-01-04 08:00:00.0|2016-01-11 08:00:00.0|99.0699982        |
|2016-01-11 08:00:00.0|2016-01-18 08:00:00.0|98.49999799999999 |
|2016-01-18 08:00:00.0|2016-01-25 08:00:00.0|98.1220016        |
|2016-01-25 08:00:00.0|2016-02-01 08:00:00.0|96.2539976        |
|2016-02-01 08:00:00.0|2016-02-08 08:00:00.0|95.29199960000001 |
|2016-02-08 08:00:00.0|2016-02-15 08:00:00.0|94.2374975        |
|2016-02-15 08:00:00.0|2016-02-22 08:00:00.0|96.7880004        |
|2016-02-22 08:00:00.0|2016-02-29 08:00:00.0|96.23000160000001 |
|2016-02-29 08:00:00.0|2016-03-07 08:00:00.0|101.53200079999999|
|2016-03-07 08:00:00.0|2016-03-14 08:00:00.0|101.6199998       |
|2016-03-14 08:00:00.0|2016-03-21 08:00:00.0|105.63600160000001|
|2016-03-21 08:00:00.0|2016-03-28 08:00:00.0|105.92749950000001|
|2016-03-28 08:00:00.0|2016-04-04 08:00:00.0|109.46799940000001|
|2016-04-04 08:00:00.0|2016-04-11 08:00:00.0|109.39799980000001|
|2016-04-11 08:00:00.0|2016-04-18 08:00:00.0|110.3820004       |
|2016-04-18 08:00:00.0|2016-04-25 08:00:00.0|106.15400079999999|
|2016-04-25 08:00:00.0|2016-05-02 08:00:00.0|96.8759994        |
|2016-05-02 08:00:00.0|2016-05-09 08:00:00.0|93.6240004        |
|2016-05-09 08:00:00.0|2016-05-16 08:00:00.0|92.13399799999999 |
|2016-05-16 08:00:00.0|2016-05-23 08:00:00.0|94.77999880000002 |
+--------------------+--------------------+------------------+
only showing top 20 rows
```

到目前为止，我们已经了解了如何在Spark中使用Window了。

数据文件下载