SL01: 8.4 输出操作

8.4.1 启动流计算(Initiate streaming Computation)

8.4.2 输出模式(output mode)

8.4.3 输出接收器(output sink)

SL02: 8.4.1 启动流计算



DataFrame/Dataset的.writeStream()方法将会返回DataStreamWriter接口，接口通过.start()真正启动流计算，并将DataFrame/Dataset写入到外部的输出接收器，DataStreamWriter接口有以下几个主要函数：

（1）format：接收器类型。
（2）outputMode：输出模式，指定写入接收器的内容，可以是Append模式、Complete模式或Update模式。
（3）queryName：查询的名称，可选，用于标识查询的唯一名称。
（4）trigger：触发间隔，可选，设定触发间隔，如果未指定，则系统将在上一次处理完成后立即检查新数据的可用性。如果由于先前的处理尚未完成导致超过触发间隔，则系统将在处理完成后立即触发新的查询。

SL03: 8.4.1 启动流计算



不同的接收器类型存在通用选项（option），如checkpointLocation：检查点位置选项，某些接收器可以保证端到端的容错的能力，这需要提前指定将检查点信息保存到文件系统内的位置，位置可以是与HDFS兼容的容错文件系统中的目录。

以下是一个示例代码：

```
query = wordcount \
    .writeStream \
    .outputMode("update") \
    .format("console") \
    .option('truncate', 'false')\
    .trigger(processingTime="8 seconds") \
    .start()
```

SL04: 8.4.2 输出模式



输出模式用于指定写入接收器的内容，主要有以下几种：

- Append模式：只有结果表中自上次触发间隔后增加的新行，才会被写入外部存储器。这种模式一般适用于“不希望更改结果表中现有行的内容”的使用场景。
- Complete模式：已更新的完整的结果表可被写入外部存储器。
- Update模式：只有自上次触发间隔后结果表中发生更新的行，才会被写入外部存储器。这种模式与Complete模式相比，输出较少，如果结果表的部分行没有更新，则不会输出任何内容。当查询不包括聚合时，这个模式等同于Append模式。

SL05: 8.4.2 输出模式



不同的流计算查询类型支持不同的输出模式，二者之间的兼容性如表8-1所示。

表 8-1　　流计算查询类型与输出模式之间的兼容性

查询类型		支持的输出模式	备注
聚合查询	在事件时间字段上使用水印的聚合	Append Complete Update	Append 模式使用水印（在 8.6.3 节介绍）来清理旧的聚合状态
	其他聚合	Complete Update	
连接查询		Append	
其他查询		Append Update	不支持 Complete 模式，因为无法将所有未分组数据保存在结果表内

SL06: 8.4.2 输出模式



以8.2节的示例代码为例，如果要使用Append输出模式，则必须设定水印或者不使用聚合查询，比如，将原先使用groupBy聚合的代码：

```
wordCounts = words.groupBy("word").count()
```

修改为：

```
wordCounts = words.filter("word == 'Hello'")
```

SL07: 8.4.3 输出接收器(output sink)



系统内置的输出接收器包括

- File接收器、
- Kafka接收器、
- Foreach接收器、
- Console接收器、
- Memory接收器等，

其中，Console接收器和Memory接收器仅用于调试用途。有些接收器由于无法保证输出的持久性，导致其不是容错的。

SL08: 8.4.3 输出接收器(output sink)



表 8-2 Spark 内置的输出接收器的详细信息

接收器	支持的输出模式	选项	容错
File 接收器	Append	path: 输出目录的路径，必须指定； retention:输出文件的存活时间。过老的文件会被从元数据日志内排除，使得后续查询不会处理这些老文件。值是类似“12h”、“7d”等等的时间字符串。默认不启用	是。数据只会被处理一次
Kafka 接收器	Append Complete Update	选项较多，具体可查看 Kafka 对接指南	是。数据至少被处理一次
Foreach 接收器	Append Complete Update	无	如果是 Foreach 接收器，则是容错的，其中数据至少被处理一次，如果是 ForeachBatch 接收器，则依赖于具体的实现
Console 接收器	Append Complete Update	numRows：每次触发后打印多少行，默认为 20； truncate：如果行太长是否截断，默认为“是”	否
Memory 接收器	Append Complete	无	否。在 Complete 输出模式下，重启查询会重建全表

SL09: 8.4.3 输出接收器(output sink)



以File接收器为例，这里把8.2节的实例修改为使用File接收器，
修改后的代码文件为StructuredNetworkWordCountFileSink.py

```
#!/usr/bin/env python3

from pyspark.sql import SparkSession
from pyspark.sql.functions import split
from pyspark.sql.functions import explode
from pyspark.sql.functions import length
```

SL10: 8.4.3 输出接收器



```
if __name__ == "__main__":
    spark = SparkSession ₩
        .builder ₩
        .appName("StructuredNetworkWordCountFileSink") ₩
        .getOrCreate()

    spark.sparkContext.setLogLevel('WARN')


    lines = spark ₩
        .readStream ₩
        .format("socket") ₩
        .option("host", "localhost") ₩
        .option("port", 9999) ₩
        .load()
```

SL11: 8.4.3 输出接收器



```
 words = lines.select(
    explode(
        split(lines.value, " ")
    ).alias("word")
)

all_length_5_words = words.filter(length("word") == 5)

query = all_length_5_words ₩
    .writeStream ₩
    .outputMode("append") ₩
    .format("parquet") ₩
    .option("path", "file:///tmp/filesink") ₩
    .option("checkpointLocation", "file:///tmp/file-sink-cp") ₩
    .trigger(processingTime="8 seconds") ₩
    .start()
query.awaitTermination()
```

SL12: 8.4.3 输出接收器



在Linux系统中新建一个终端（记作“数据源终端”），输入如下命令：

```
$ nc -lk 9999
```

再新建一个终端（记作“流计算终端”），执行如下命令执行StructuredNetworkWordCountFileSink.py：

```
$ cd /usr/local/spark/mycode/structuredstreaming
$ /usr/local/spark/bin/spark-submit StructuredNetworkWordCountFileSink.py
```

为了模拟文本数据流，可以在数据源终端内用键盘不断敲入一行行英文语句，并且让其中部分英语单词长度等于5

SL13: 8.4.3 输出接收器



由于程序执行后不会在终端输出信息，这时可新建一个终端，执行如下命令查看File接收器保存的位置：

```
$ cd /tmp/filesink
$ ls
```

可以看到以parquet格式保存的类似如下的文件列表：

```
Terminal File Edit View Search Terminal Help
(base) hadoop@hadoop01:/tmp/filesink$ ls
part-00000-3bd6cc95-fe5d-4b41-97c6-e9383cfa1873-c000.snappy.parquet
part-00000-549c64f4-c776-49c3-a913-18f2153a0eb1-c000.snappy.parquet
part-00000-941c83ca-13cc-450c-93ac-c7b6651d50de-c000.snappy.parquet
part-00000-d9fa0dae-443d-4066-a6ba-0eff040bab31-c000.snappy.parquet
_spark_metadata
(base) hadoop@hadoop01:/tmp/filesink$
```

可以使用strings命令查看文件内的字符串，具体如下：

```
(base) hadoop@hadoop01:/tmp/filesink$ strings part-00000-3bd6cc95-fe5
d-4b41-97c6-e9383cfa1873-c000.snappy.parquet
```

SL14: 8.4.3 输出接收器(output sink)



```
Terminal File Edit View Search Terminal Help
part-00000-941c83ca-13cc-450c-93ac-c7b6651d50de-c000.snappy.parquet
part-00000-d9fa0dae-443d-4066-a6ba-0eff040bab31-c000.snappy.parquet
 spark_metadata
(base) hadoop@hadoop01:/tmp/filesink$ strings part-00000-3bd6cc95-fe5
d-4b41-97c6-e9383cfa1873-c000.snappy.parquet
PAR1
hello
every
every
hello
spark_schema
word%
word
hello
every
org.apache.spark.version
3.4.0
)org.apache.spark.sql.parquet.row.metadata
[{"type":"struct","fields":[{"name":"word","type":"string","nullable"
:false,"metadata":{}}]}
Jparquet-mr version 1.12.3 (build f8dced182c4c1fbdec6ccb3185537b5a01e
6ed6b)
PAR1
(base) hadoop@hadoop01:/tmp/filesink$
```