《Spark编程基础（Python版）》

教材官网： http://dblab.xmu.edu.cn/post/spark-python/

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第7章 Structured Streaming

（PPT版本号：2019年春季学期）

扫一扫访问教材官网

林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：**http://www.cs.xmu.edu.cn/linziyu**

课程教材

Spark入门教程(Python版)
http://dblab.xmu.edu.cn/blog/1709-2/

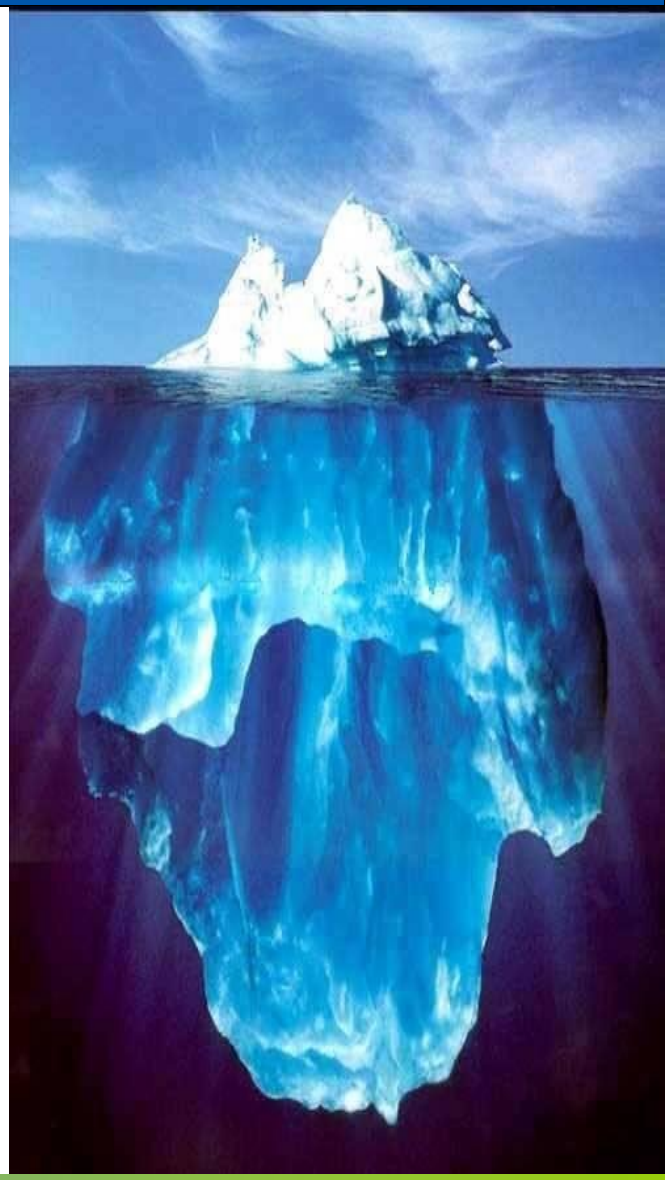
纸质教材预期在2019年夏天上市销售

厦门大学林子雨

扫一扫访问在线教程

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

提纲



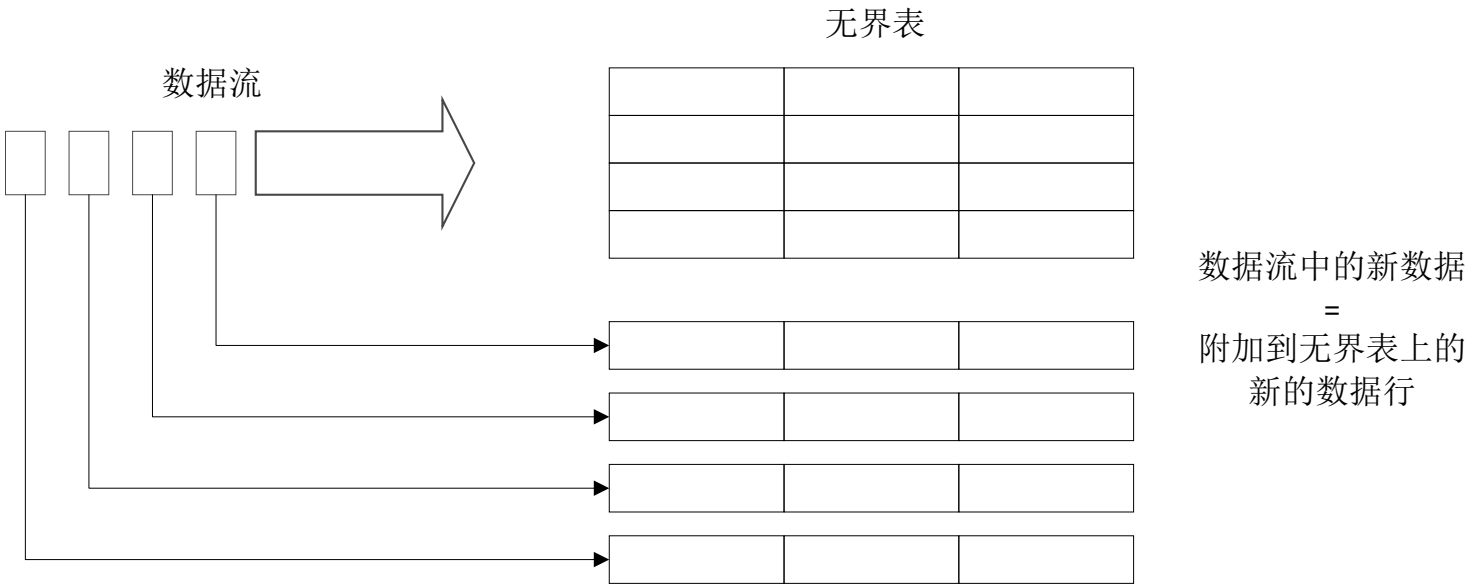
7.1 概述

7.1.1 基本概念
7.1.2 两种处理模型
7.1.3 Structured Streaming和Spark SQL、Spark Streaming关系

7.1.1 基本概念

•Structured Streaming的关键思想是将实时数据流视为一张正在不断添加数据的表

•可以把流计算等同于在一个静态表上的批处理查询，Spark会在不断添加数据的无界输入表上运行计算，并进行增量查询



7.1.1 基本概念

•在无界表上对输入的查询将生成结果表，系统每隔一定的周期会触发对无界表的计算并更新结果表

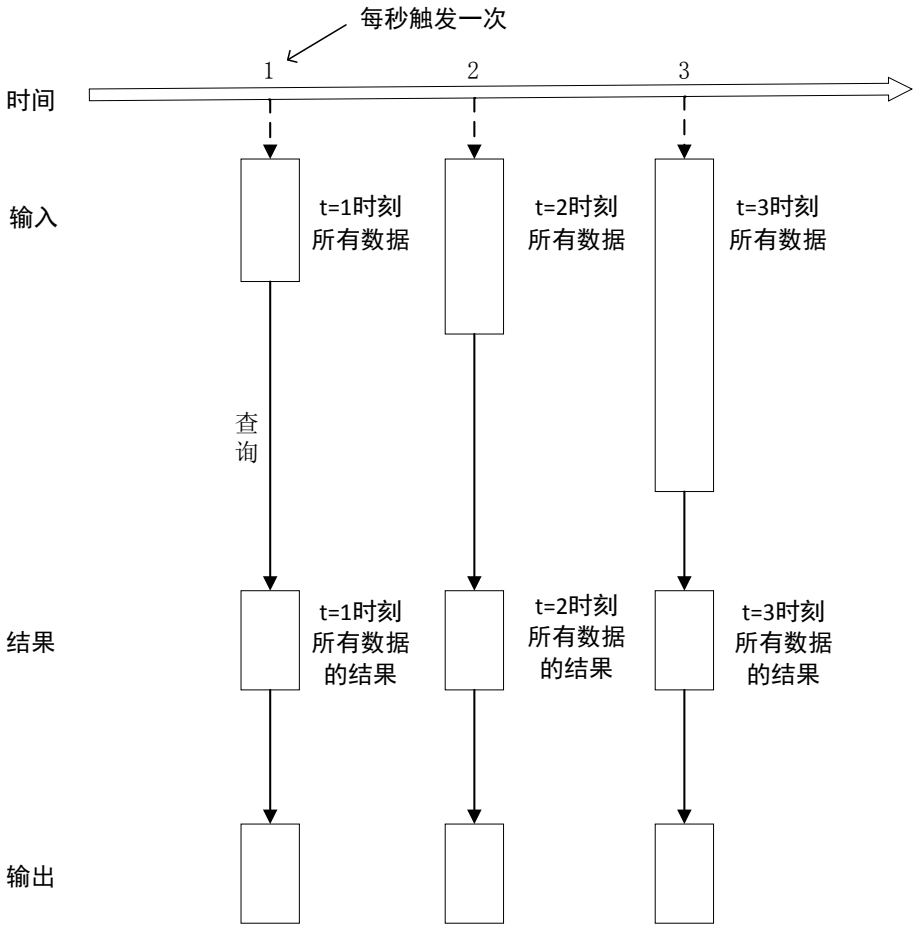
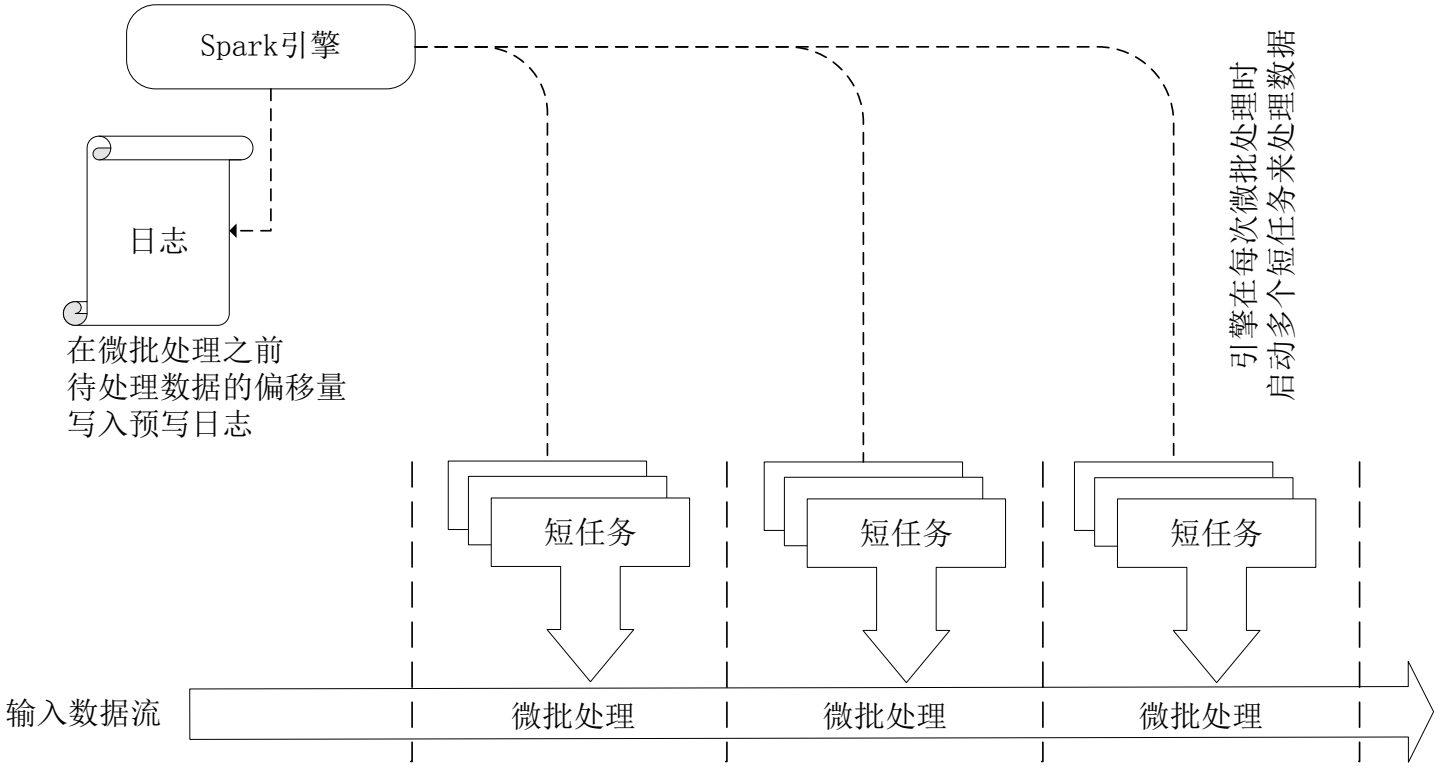


图 Structured Streaming编程模型

7.1.2 两种处理模型

（1）微批处理

- Structured Streaming默认使用微批处理执行模型，这意味着Spark流计算引擎会定期检查流数据源，并对自上一批次结束后到达的新数据执行批量查询
- 数据到达和得到处理并输出结果之间的延时超过100毫秒

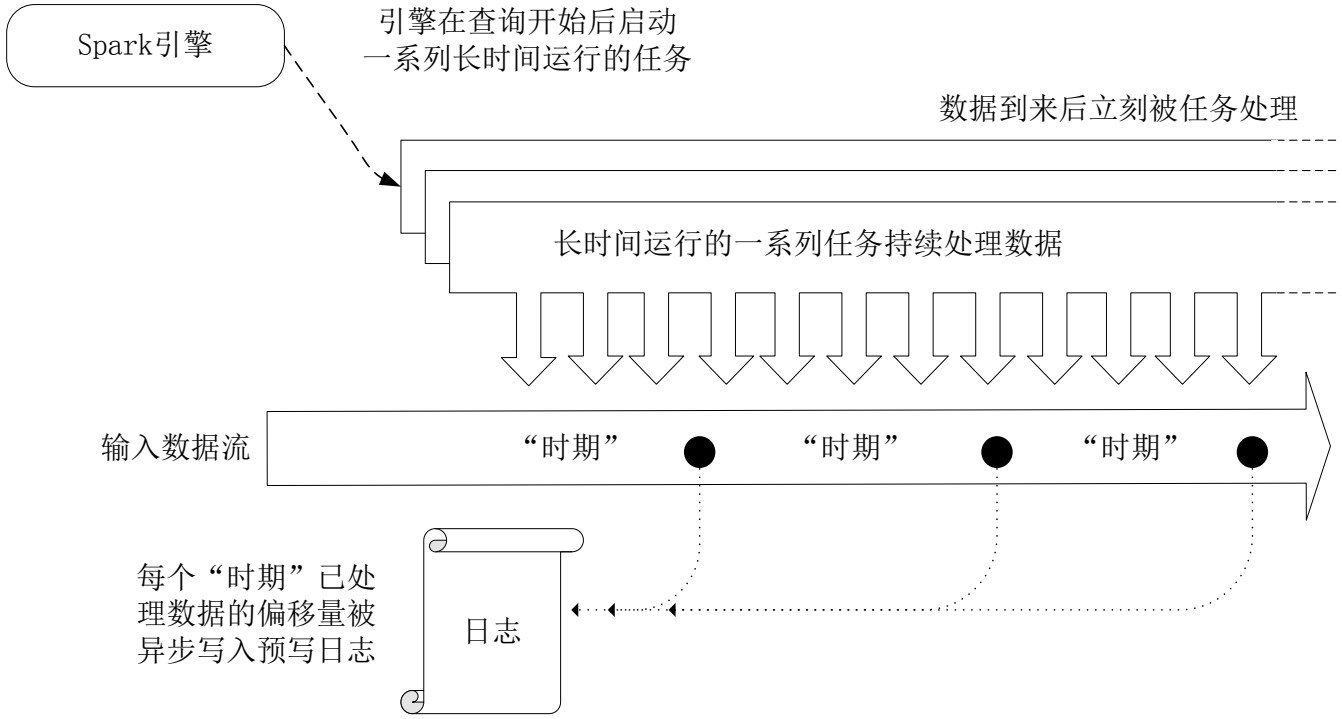


7.1.2 两种处理模型

（2）持续处理

•Spark从2.3.0版本开始引入了持续处理的试验性功能，可以实现流计算的毫秒级延迟

•在持续处理模式下，Spark不再根据触发器来周期性启动任务，而是启动一系列的连续读取、处理和写入结果的长时间运行的任务



7.1.3 Structured Streaming和Spark SQL、Spark Streaming关系

•Structured Streaming处理的数据跟Spark Streaming一样，也是源源不断的数据流，区别在于，Spark Streaming采用的数据抽象是DStream（本质上就是一系列RDD），而Structured Streaming采用的数据抽象是DataFrame。

•Structured Streaming可以使用Spark SQL的DataFrame/Dataset来处理数据流。虽然Spark SQL也是采用DataFrame作为数据抽象，但是，Spark SQL只能处理静态的数据，而Structured Streaming可以处理结构化的数据流。这样，Structured Streaming就将Spark SQL和Spark Streaming二者的特性结合了起来。

7.1.3 Structured Streaming和Spark SQL、Spark Streaming关系

•Structured Streaming可以对DataFrame/Dataset应用前面章节提到的各种操作，包括select、where、groupBy、map、filter、flatMap等。

•Spark Streaming只能实现秒级的实时响应，而Structured Streaming由于采用了全新的设计方式，采用微批处理模型时可以实现100毫秒级别的实时响应，采用持续处理模型时可以支持毫秒级的实时响应。

7.2 编写Structured Streaming程序的基本步骤

编写Structured Streaming程序的基本步骤包括：

- 导入pyspark模块
- 创建SparkSession对象
- 创建输入数据源
- 定义流计算过程
- 启动流计算并输出结果

7.2 编写Structured Streaming程序的基本步骤

实例任务：一个包含很多行英文语句的数据流源源不断到达，Structured Streaming程序对每行英文语句进行拆分，并统计每个单词出现的频率

7.2 编写Structured Streaming程序的基本步骤

1.步骤1：导入pyspark模块

导入PySpark模块，代码如下：

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

由于程序中需要用到拆分字符串和展开数组内的所有单词的功能，所以引用了来自pyspark.sql.functions里面的split和explode函数。

2.步骤2：创建SparkSession对象

创建一个SparkSession对象，代码如下：

```
if __name__ == "__main__":
    spark = SparkSession \
        .builder \
        .appName("StructuredNetworkWordCount") \
        .getOrCreate()

    spark.sparkContext.setLogLevel('WARN')
```

3．步骤3：创建输入数据源

创建一个输入数据源，从“监听在本机（localhost）的9999端口上的服务”那里接收文本数据，具体语句如下：

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

4.步骤4：定义流计算过程

有了输入数据源以后，接着需要定义相关的查询语句，具体如下：

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

7.2 编写Structured Streaming程序的基本步骤

5.步骤5：启动流计算并输出结果

定义完查询语句后，下面就可以开始真正执行流计算，具体语句如下：

```
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .trigger(processingTime="8 seconds") \
    .start()

query.awaitTermination()
```

7.2 编写Structured Streaming程序的基本步骤

把代码写入文件StructuredNetworkWordCount.py
在执行StructuredNetworkWordCount.py之前，需要启动HDFS。
启动HDFS的命令如下：

```
$ cd /usr/local/hadoop
$ sbin/start-dfs.sh
```

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：

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

7.2 编写Structured Streaming程序的基本步骤

•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，nc程序会把这些数据发送给StructuredNetworkWordCount.py程序进行处理，比如输入如下数据：

```
apache spark
apache hadoop
```

则在“流计算终端”窗口内会输出类似以下的结果信息：

```
-------------------------------------------
Batch: 0
-------------------------------------------
+------+-----+
|  word|count|
+------+-----+
|apache|    1|
| spark|    1|
+------+-----+
```

```
-------------------------------------------
Batch: 1
-------------------------------------------
+------+-----+
|  word|count|
+------+-----+
|apache|    2|
| spark|    1|
|hadoop|    1|
+------+-----+
```

7.3 输入源

7.3.1 File源
7.3.2 Kafka源
7.3.3 Socket源
7.3.4 Rate源

7.3.1 File源

•File源（或称为“文件源”）以文件流的形式读取某个目录中的文件，支持的文件格式为csv、json、orc、parquet、text等。

•需要注意的是，文件放置到给定目录的操作应当是原子性的，即不能长时间在给定目录内打开文件写入内容，而是应当采取大部分操作系统都支持的、通过写入到临时文件后移动文件到给定目录的方式来完成。

7.3.1 File源

一个实例

这里以一个JSON格式文件的处理来演示File源的使用方法，主要包括以下两个步骤：

- 创建程序生成JSON格式的File源测试数据
- 创建程序对数据进行统计

7.3.1 File源

（1）创建程序生成JSON格式的File源测试数据

为了演示JSON格式文件的处理，这里随机生成一些JSON格式的文件来进行测试。代码文件spark_ss_filesource_generate.py内容如下：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# 导入需要用到的模块
import os
import shutil
import random
import time
```

7.3.1 File源

```
TEST_DATA_TEMP_DIR = '/tmp/'
TEST_DATA_DIR = '/tmp/testdata/'

ACTION_DEF = ['login', 'logout', 'purchase']
DISTRICT_DEF = ['fujian', 'beijing', 'shanghai', 'guangzhou']
JSON_LINE_PATTERN = '{{"eventTime": {}, "action": "{}", "district":
"{}"}}\n‘

# 测试的环境搭建，判断文件夹是否存在，如果存在则删除旧数据，并建
立文件夹
def test_setUp():
    if os.path.exists(TEST_DATA_DIR):
        shutil.rmtree(TEST_DATA_DIR, ignore_errors=True)
    os.mkdir(TEST_DATA_DIR)
```

7.3.1 File源

```
# 测试环境的恢复，对文件夹进行清理
def test_tearDown():
    if os.path.exists(TEST_DATA_DIR):
        shutil.rmtree(TEST_DATA_DIR, ignore_errors=True)


# 生成测试文件
def write_and_move(filename, data):
    with open(TEST_DATA_TEMP_DIR + filename,
              "wt", encoding="utf-8") as f:
        f.write(data)

    shutil.move(TEST_DATA_TEMP_DIR + filename,
                TEST_DATA_DIR + filename)
```

7.3.1 File源

```
if __name__ == "__main__":
    test_setUp()

    for i in range(1000):
        filename = 'e-mall-{}.json'.format(i)

        content = ''
        rndcount = list(range(100))
        random.shuffle(rndcount)
        for _ in rndcount:
            content += JSON_LINE_PATTERN.format(
                str(int(time.time())),
                random.choice(ACTION_DEF),
                random.choice(DISTRICT_DEF))
        write_and_move(filename, content)

        time.sleep(1)

    test_tearDown()
```

7.3.1 File源

这段程序首先建立测试环境，清空测试数据所在的目录，接着使用for循环一千次来生成一千个文件，文件名为“e-mall-数字.json”，

文件内容是不超过100行的随机JSON行，行的格式是类似如下：
{"eventTime": 1546939167, "action": "logout", "district": "fujian"}\n

7.3.1 File源

（2）创建程序对数据进行统计

spark_ss_filesource.py”，其代码内容如下：

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# 导入需要用到的模块
import os
import shutil
from pprint import pprint

from pyspark.sql import SparkSession
from pyspark.sql.functions import window, asc
from pyspark.sql.types import StructType, StructField
from pyspark.sql.types import TimestampType, StringType
# 定义JSON文件的路径常量
TEST_DATA_DIR_SPARK = 'file:///tmp/testdata/'
```

7.3.1 File源

```
if __name__ == "__main__":
    # 定义模式，为时间戳类型的eventTime、字符串类型的操作和省份组成
    schema = StructType([
        StructField("eventTime", TimestampType(), True),
        StructField("action", StringType(), True),
        StructField("district", StringType(), True)])

    spark = SparkSession \
        .builder \
        .appName("StructuredEMallPurchaseCount") \
        .getOrCreate()

    spark.sparkContext.setLogLevel('WARN')
```

7.3.1 File源

```
lines = spark \
    .readStream \
    .format("json") \
    .schema(schema) \
    .option("maxFilesPerTrigger", 100) \
    .load(TEST_DATA_DIR_SPARK)

# 定义窗口
windowDuration = '1 minutes'

windowedCounts = lines \
    .filter("action = 'purchase'") \
    .groupBy('district', window('eventTime', windowDuration)) \
    .count() \
    .sort(asc('window'))
```

7.3.1 File源

```
query = windowedCounts \
        .writeStream \
        .outputMode("complete") \
        .format("console") \
        .option('truncate', 'false') \
        .trigger(processingTime="10 seconds") \
        .start()

    query.awaitTermination()
```

7.3.1 File源

（3）测试运行程序

程序运行过程需要访问HDFS，因此，需要启动HDFS，命令如下：

```
$ cd /usr/local/hadoop
$ sbin/start-dfs.sh
```

新建一个终端，执行如下命令生成测试数据：

```
$ cd /usr/local/spark/mycode/structuredstreaming/file
$ python3 spark_ss_filesource_generate.py
```

新建一个终端，执行如下命令运行数据统计程序：

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

7.3.1 File源

运行程序以后，可以看到类似如下的输出结果：

```
-------------------------------------------
Batch: 0
-------------------------------------------
+---------+-----------------------------------------+-----+
|district |window                                   |count|
+---------+-----------------------------------------+-----+
|guangzhou|[2019-01-08 17:19:00, 2019-01-08 17:20:00]|283  |
|shanghai |[2019-01-08 17:19:00, 2019-01-08 17:20:00]|251  |
|fujian   |[2019-01-08 17:19:00, 2019-01-08 17:20:00]|258  |
|beijing  |[2019-01-08 17:19:00, 2019-01-08 17:20:00]|258  |
|guangzhou|[2019-01-08 17:20:00, 2019-01-08 17:21:00]|492  |
|beijing  |[2019-01-08 17:20:00, 2019-01-08 17:21:00]|499  |
|fujian   |[2019-01-08 17:20:00, 2019-01-08 17:21:00]|513  |
|shanghai |[2019-01-08 17:20:00, 2019-01-08 17:21:00]|503  |
|guangzhou|[2019-01-08 17:21:00, 2019-01-08 17:22:00]|71   |
|fujian   |[2019-01-08 17:21:00, 2019-01-08 17:22:00]|74   |
|shanghai |[2019-01-08 17:21:00, 2019-01-08 17:22:00]|66   |
|beijing  |[2019-01-08 17:21:00, 2019-01-08 17:22:00]|52   |
+---------+-----------------------------------------+-----+
```

7.3.2 Kafka源

Kafka源是流处理最理想的输入源，因为它可以保证实时和容错。

实例演示

在这个实例中，使用生产者程序每0.1秒生成一个包含2个字母的单词，并写入Kafka的名称为“wordcount-topic”的主题（Topic）内。Spark的消费者程序通过订阅wordcount-topic，会源源不断收到单词，并且每隔8秒钟对收到的单词进行一次词频统计，把统计结果输出到Kafka的主题wordcount-result-topic内，同时，通过2个监控程序检查Spark处理的输入和输出结果。

7.3.2 Kafka源

1.启动Kafka

在Linux系统中新建一个终端（记作“Zookeeper终端”），输入下面命令启动Zookeeper服务：

```
$ cd /usr/local/kafka
$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

不要关闭这个终端窗口，一旦关闭，Zookeeper服务就停止了。另外打开第二个终端（记作“Kafka终端”），然后输入下面命令启动Kafka服务：

```
$ cd /usr/local/kafka
$ bin/kafka-server-start.sh config/server.properties
```

7.3.2 Kafka源

不要关闭这个终端窗口，一旦关闭，Kafka服务就停止了。
再新开一个终端（记作“监控输入终端”），执行如下命令监控Kafka收到的文本：

```
$ cd /usr/local/kafka
$ bin/kafka-console-consumer.sh \
> --bootstrap-server localhost:9092 --topic wordcount-topic
```

再新开一个终端（记作“监控输出终端”），执行如下命令监控输出的结果文本：

```
$ cd /usr/local/kafka
$ bin/kafka-console-consumer.sh \
> --bootstrap-server localhost:9092 --topic wordcount-result-topic
```

7.3.2 Kafka源

2.编写生产者（Producer）程序

代码文件spark_ss_kafka_producer.py内容如下：

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

import string
import random
import time

from kafka import KafkaProduce
```

7.3.2 Kafka源

```
if __name__ == "__main__":
    producer = KafkaProducer(bootstrap_servers=['localhost:9092'])

    while True:
        s2 = (random.choice(string.ascii_lowercase) for _ in range(2))
        word = ''.join(s2)
        value = bytearray(word, 'utf-8')

        producer.send('wordcount-topic', value=value) \
            .get(timeout=10)

        time.sleep(0.1)
```

7.3.2 Kafka源

如果还没有安装Python3的Kafka支持，需要按照如下操作进行安装：
（1）首先确认有没有安装pip3，如果没有，使用如下命令安装：

```
$ sudo apt-get install pip3
```

（2）安装kafka-python模块，命令如下：

```
$ sudo pip3 install kafka-python
```

然后在终端中执行如下命令运行生产者程序：

```
$ cd /usr/local/spark/mycode/structuredstreaming/kafka/
$ python3 spark_ss_kafka_producer.py
```

生产者程序执行以后，在“监控输入终端”的窗口内就可以看到持续输出包含2个字母的单词

7.3.2 Kafka源

3.编写消费者（Consumer）程序

代码文件spark_ss_kafka_consumer.py内容如下：

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

from pyspark.sql import SparkSession


if __name__ == "__main__":
    spark = SparkSession \
        .builder \
        .appName("StructuredKafkaWordCount") \
        .getOrCreate()

    spark.sparkContext.setLogLevel('WARN‘)
```

7.3.2 Kafka源

```
lines = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", 'wordcount-topic') \
    .load() \
    .selectExpr("CAST(value AS STRING)")

wordCounts = lines.groupBy("value").count()
```

7.3.2 Kafka源

```
    query = wordCounts \
        .selectExpr("CAST(value AS STRING) as key",
"CONCAT(CAST(value AS STRING), ':', CAST(count AS STRING)) as
value") \
        .writeStream \
        .outputMode("complete") \
        .format("kafka") \
        .option("kafka.bootstrap.servers", "localhost:9092") \
        .option("topic", "wordcount-result-topic") \
        .option("checkpointLocation", "file:///tmp/kafka-sink-cp") \
        .trigger(processingTime="8 seconds") \
        .start()

    query.awaitTermination()
```

7.3.2 Kafka源

在终端中执行如下命令运行消费者程序：

```
$ cd /usr/local/spark/mycode/structuredstreaming/kafka/
$ /usr/local/spark/bin/spark-submit \
> --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.0 \
> spark_ss_kafka_consumer.py
```

消费者程序运行起来以后，可以在“监控输出终端”看到类似如下的输出结果：

```
sq:3
bl:6
lo:8
…
```

7.3.3Socket源

Socket源从一个本地或远程主机的某个端口服务上读取数据，数据的编码为UTF8。因为Socket源使用内存保存读取到的所有数据，并且远端服务不能保证数据在出错后可以使用检查点或者指定当前已处理的偏移量来重放数据，所以，它无法提供端到端的容错保障。Socket源一般仅用于测试或学习用途。

Socket源的实例可以参考7.2节的StructuredNetworkWordCount.py。

7.3.4Rate源

Rate源可每秒生成特定个数的数据行，每个数据行包括时间戳和值字段。时间戳是消息发送的时间，值是从开始到当前消息发送的总个数，从0开始。Rate源一般用来作为调试或性能基准测试。

代码文件spark_ss_rate.py内容如下：

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

from pyspark.sql import SparkSession


if __name__ == "__main__":
    spark = SparkSession \
        .builder \
        .appName("TestRateStreamSource") \
        .getOrCreate()

    spark.sparkContext.setLogLevel('WARN‘)
```

7.3.4Rate源

```
lines = spark \
    .readStream \
    .format("rate") \
    .option('rowsPerSecond', 5) \
    .load()

print(lines.schema)

query = lines \
    .writeStream \
    .outputMode("update") \
    .format("console") \
    .option('truncate', 'false') \
    .start()

query.awaitTermination()
```

7.3.4Rate源

在Linux终端中执行如下命令执行spark_ss_rate.py：

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

上述命令执行后，会得到类似如下的结果：

```
StructType(List(StructField(timestamp,TimestampType,true),Str
uctField(value,LongType,true)))
-------------------------------------------
Batch: 0
-------------------------------------------
+---------+-----+
|timestamp|value|
+---------+-----+
+---------+-----+
```

7.3.4Rate源

```
-------------------------------------------
Batch: 1
-------------------------------------------
+----------------------+-----+
|timestamp             |value|
+----------------------+-----+
|2018-10-01 15:42:38.595|0    |
|2018-10-01 15:42:38.795|1    |
|2018-10-01 15:42:38.995|2    |
|2018-10-01 15:42:39.195|3    |
|2018-10-01 15:42:39.395|4    |
+----------------------+-----+
```

7.4 输出操作

7.4.1 启动流计算
7.4.2 输出模式
7.4.3 输出接收器

7.4.1 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。

7.4.2 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。

7.4.3 输出接收器

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。

以File接收器为例，这里把7.2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
```

7.4.3 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()
```

7.4.3 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()
```

7.4.3 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

7.4.3 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：

```
part-00000-2bd184d2-e9b0-4110-9018-a7f2d14602a9-c000.snappy.parquet
part-00000-36eed4ab-b8c4-4421-adc6-76560699f6f5-c000.snappy.parquet
part-00000-dde601ad-1b49-4b78-a658-865e54d28fb7-c000.snappy.parquet
part-00001-eedddae2-fb96-4ce9-9000-566456cd5e8e-c000.snappy.parquet
_spark_metadata
```

可以使用strings命令查看文件内的字符串，具体如下：

```
$ strings part-00003-89584d0a-db83-467b-84d8-53d43baa4755-c000.snappy.parquet
```

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://dblab.xmu.edu.cn/post/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），现为厦门大学计算机科学系助理教授（讲师），曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度和2017年度厦门大学教学类奖教金获得者，荣获2017年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过500万字高价值的研究和教学资料，累计网络访问量超过500万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过100万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：《大数据技术原理与应用》教材

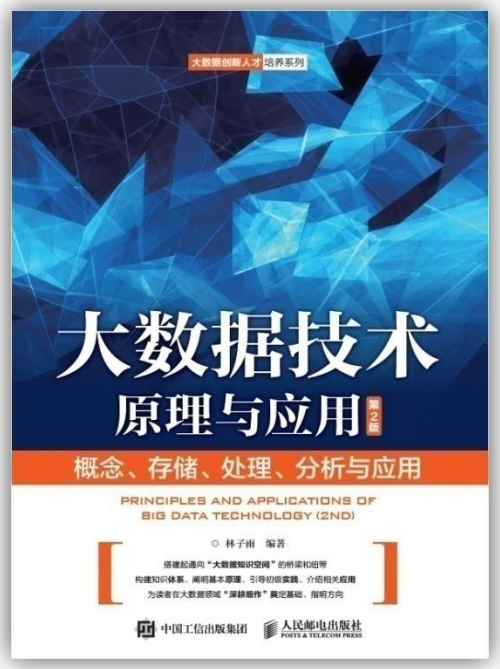
《大数据技术原理与应用——概念、存储、处理、分析与应用（第2版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社 ISBN:978-7-115-44330-4 定价：49.80元

全书共有15章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase和MapReduce等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata

扫一扫访问教材官网



附录D：《大数据基础编程、实验和案例教程》

本书是与《大数据技术原理与应用（第2版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

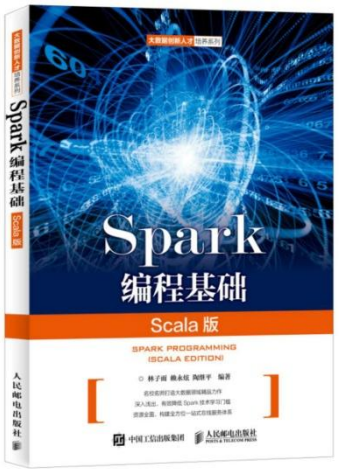
厦门大学林子雨编著

配套实验指导书

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，五套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

清华大学出版社 ISBN:978-7-302-47209-4 定价：59元

附录E：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录F：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

Department of Computer Science, Xiamen University, 2019