7.4 基本输入源

7.4.1 文件流

7.4.2 套接字流

7.4.3 RDD队列流

7.4.1 文件流



1.在pyspark中创建文件流

```
$ cd /usr/local/spark/mycode
$ mkdir streaming
$ cd streaming
$ mkdir logfile
$ cd logfile
```

7.4.1 文件流



进入pyspark创建文件流。
请另外打开一个终端窗口，启动进入pyspark

```
>>> from pyspark import SparkContext
>>> from pyspark.streaming import StreamingContext
>>> ssc = StreamingContext(sc, 10)
>>> lines = ssc. ₩
... textFileStream('file:///usr/local/spark/mycode/streaming/logfile')
>>> words = lines.flatMap(lambda line: line.split(' '))
>>> wordCounts = words.map(lambda x : (x,1)).reduceByKey(la
mbda a,b:a+b)
>>> wordCounts.pprint()
>>> ssc.start()
>>> ssc.awaitTermination()
```

7.4.1 文件流



上面在pyspark中执行的程序，一旦你输入ssc.start()以后，程序就开始自动进入循环监听状态，屏幕上会显示一堆的信息，如下：

```
-------------------------------------------
Time: 2023-09-17 10:56:20
-------------------------------------------

-------------------------------------------
Time: 2023-09-17 10:56:30
-------------------------------------------

-------------------------------------------
Time: 2023-09-17 10:56:40
-------------------------------------------
```

7.4.1 文件流



在“/usr/local/spark/mycode/streaming/logfile”目录下新建一个log.txt文件，就可以在监听窗口中显示词频统计结果

```
-------------------------------------------
Time: 2023-09-17 10:56:50
-------------------------------------------
('hadoop', 1)
('love', 2)
('spark', 1)
('I', 2)
```

7.4.1 文件流



2. 采用独立应用程序方式创建文件流

```
$ cd /usr/local/spark/mycode
$ cd streaming
$ cd logfile
$ vim FileStreaming.py
```

7.4.1 文件流



用vim编辑器新建一个FileStreaming.py代码文件，请在里面输入以下代码：

```
# coding:utf8

from pyspark import SparkContext, SparkConf
from pyspark.streaming import StreamingContext

if __name__ == '__main__':
    conf = SparkConf()
    conf.setAppName('TestDStream')
    conf.setMaster('local[2]')
    sc = SparkContext(conf = conf)
    ssc = StreamingContext(sc, 10)
    lines = ssc.textFileStream('file:///usr/local/spark/mycode/streaming/logfile')
    words = lines.flatMap(lambda line: line.split(' '))
    wordCounts = words.map(lambda x : (x,1)).reduceByKey(lambda a,b:a+b)
    wordCounts.pprint()
    ssc.start()
    ssc.awaitTermination()
```

7.4.1 文件流



可以在PyCharm中运行该程序，或者也可以通过spark-submit命令提交运行程序：

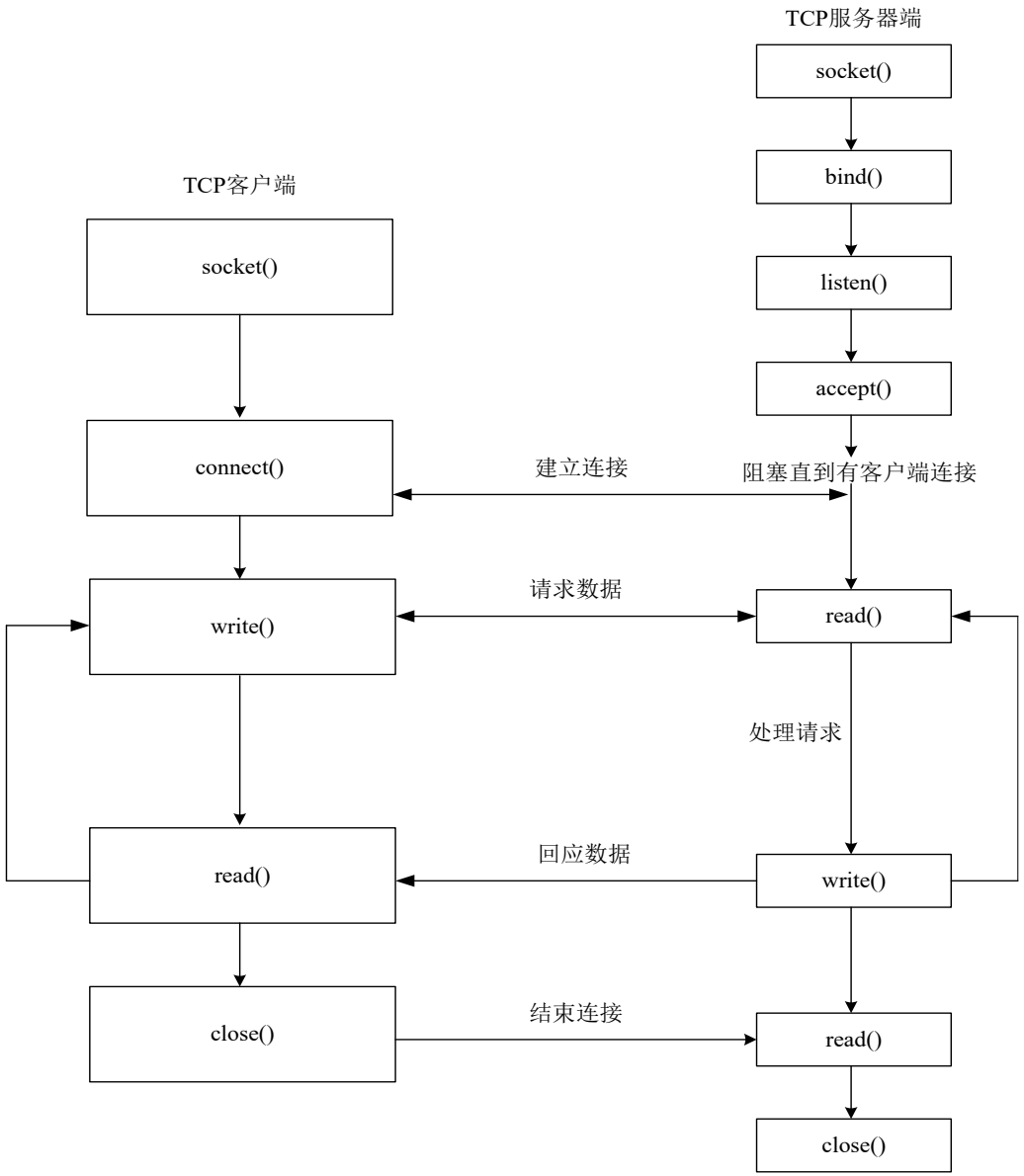
```
$ cd /usr/local/spark/mycode/streaming/logfile/
$ /usr/local/spark/bin/spark-submit FileStreaming.py
```

7.4.2 套接字流



- Spark Streaming可以通过Socket端口监听并接收数据，然后进行相应处理

1.Socket工作原理



7.4.2 套接字流



2.使用套接字流作为数据源

```
$ cd /usr/local/spark/mycode
$ mkdir streaming #如果已经存在该目录，则不用创建
$ cd streaming
$ mkdir socket
$ cd socket
$ vim NetworkWordCount.py
```

7.4.2 套接字流



请在NetworkWordCount.py文件中输入如下内容：

```
# coding:utf8

from __future__ import print_function
import sys
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: NetworkWordCount.py <hostname> <port>", file=sys.stderr)
        exit(-1)
    sc = SparkContext(appName="PythonStreamingNetworkWordCount")
    ssc = StreamingContext(sc, 1)
    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
    counts = lines.flatMap(lambda line: line.split(" ")) ₩
                  .map(lambda word: (word, 1)) ₩
                  .reduceByKey(lambda a, b: a+b)
    counts.pprint()
    ssc.start()
    ssc.awaitTermination()
```

7.4.2 套接字流



- 新打开一个窗口作为nc窗口，启动nc程序：

```
$ nc -lk 9999
```

- 再新建一个终端（记作“流计算终端”），执行如下代码启动流计算：

```
$ cd /usr/local/spark/mycode/streaming/socket
$ /usr/local/spark/bin/spark-submit NetworkWordCount.py localhost 9999
```

- 可以在nc窗口中随意输入一些单词，监听窗口就会自动获得单词数据流信息，在监听窗口每隔1秒就会打印出词频统计信息，大概会在屏幕上出现类似如下的结果：

```
-------------------------------------------
Time: 2023-09-17 11:09:37
-------------------------------------------
('Spark', 1)
('love', 1)
('I', 1)
(spark,1)
```

7.4.2 套接字流



3. 使用Socket编程实现自定义数据源

- 下面我们再前进一步，把数据源头的产生方式修改一下，不要使用nc程序，而是采用自己编写的程序产生Socket数据源

```
$ cd /usr/local/spark/mycode/streaming/socket
$ vim DataSourceSocket.py
```

7.4.2 套接字流



```
# coding:utf8
import socket
# 生成socket对象
server = socket.socket()
# 绑定ip和端口
server.bind(('hadoop01', 9999))
# 监听绑定的端口
server.listen(1)
while 1:
    # 为了方便识别，打印一个"I'm waiting the connect..."
    print("I'm waiting the connect...")
    # 这里用两个值接收，因为连接上之后使用的是客户端发来请求的这个实例
    # 所以下面的传输要使用conn实例操作
    conn,addr = server.accept()
    # 打印连接成功
    print("Connect success! Connection is from %s " % addr[0])
    # 打印正在发送数据
    print('Sending data...')
    conn.send('I love hadoop I love spark hadoop is good spark is fast'.encode())
    conn.close()
    print('Connection is broken.')
```

7.4.2 套接字流



执行如下命令启动Socket服务端：

```
$ cd  /usr/local/spark/mycode/streaming/socket
$ /usr/local/spark/bin/spark-submit DataSourceSocket.py
```

启动客户端，即NetworkWordCount程序。新建一个终端（记作"流计算终端"），输入以下命令启动NetworkWordCount程序：

```
$ cd  /usr/local/spark/mycode/streaming/socket
$ /usr/local/spark/bin/spark-submit NetworkWordCount.py hadoop01 999
9
-------------------------------------------
Time: 2023-09-17 11:16:25
-------------------------------------------
('good', 1)
('hadoop', 2)
('is', 2)
('love', 2)
('spark', 2)
('I', 2)
('fast', 1)
```

7.4.3 RDD队列流



- 在调试Spark Streaming应用程序的时候，我们可以使用streamingContext.queueStream(queueOfRDD)创建基于RDD队列的DStream
- 新建一个RDDQueueStream.py代码文件，功能是：每隔1秒创建一个RDD，Streaming每隔2秒就对数据进行处理

7.4.3 RDD队列流



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

import time
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":
    sc = SparkContext(appName="PythonStreamingQueueStream")
    ssc = StreamingContext(sc, 2)
    #创建一个队列，通过该队列可以把RDD推给一个RDD队列流
    rddQueue = []
    for i in range(5):
        rddQueue += [ssc.sparkContext.parallelize([j for j in range(1, 1001)], 10)]
        time.sleep(1)
    #创建一个RDD队列流
    inputStream = ssc.queueStream(rddQueue)
    mappedStream = inputStream.map(lambda x: (x % 10, 1))
    reducedStream = mappedStream.reduceByKey(lambda a, b: a + b)
    reducedStream.pprint()
    ssc.start()
    ssc.stop(stopSparkContext=True, stopGraceFully=True)
```

7.4.3 RDD队列流



下面执行如下命令运行该程序：

```
$ cd  /usr/local/spark/mycode/streaming/rddqueue
$ /usr/local/spark/bin/spark-submit RDDQueueStream.py
```

```
-------------------------------------------
Time: 2023-09-17 15:41:24
-------------------------------------------
(0, 100)
(8, 100)
(2, 100)
(4, 100)
(6, 100)
(1, 100)
(3, 100)
(9, 100)
(5, 100)
(7, 100)
```