spark-Streaming 读取kafka 在zookeeper中手动维护offset

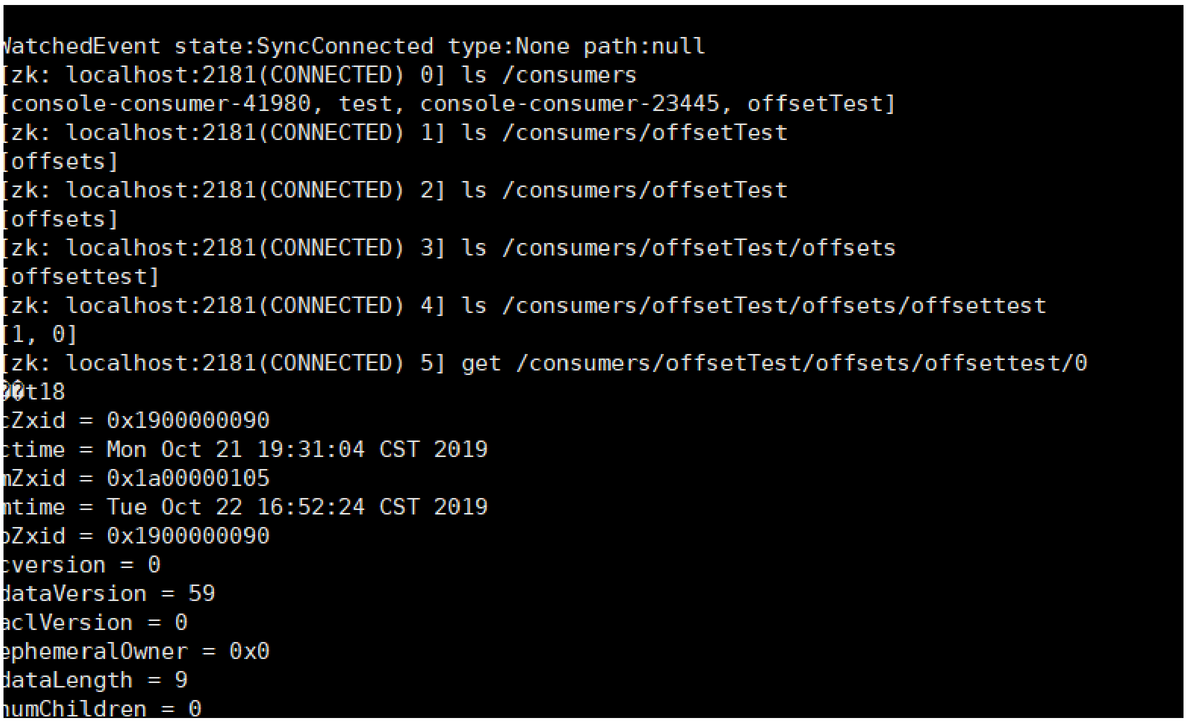
实验说明:

在kafka集群中创建以offsettest主题，启动生产者进程生产数据，利用在zookeeper维护一个以offsetTest命名的消费者组的offset偏移量

1. offset 在zookeeper中存储位置

首先启动zookeeper客户端

```
atguigu@hadoop101 zookeeper-3.4.10]$ bin/zkCli.sh
```



```
NatchedEvent state:SyncConnected type:None path:null
[zk: localhost:2181(CONNECTED) 0] ls /consumers
[console-consumer-41980, test, console-consumer-23445, offsetTest]
[zk: localhost:2181(CONNECTED) 1] ls /consumers/offsetTest
[offsets]
[zk: localhost:2181(CONNECTED) 2] ls /consumers/offsetTest
[offsets]
[zk: localhost:2181(CONNECTED) 3] ls /consumers/offsetTest/offsets
[offsettest]
[zk: localhost:2181(CONNECTED) 4] ls /consumers/offsetTest/offsets/offsettest
[1, 0]
[zk: localhost:2181(CONNECTED) 5] get /consumers/offsetTest/offsets/offsettest/0
00t18
:Zxid = 0x1900000090
:time = Mon Oct 21 19:31:04 CST 2019
nZxid = 0x1a00000105
ntime = Tue Oct 22 16:52:24 CST 2019
oZxid = 0x1900000090
:version = 0
lataVersion = 59
aclVersion = 0
ephemeralOwner = 0x0
lataLength = 9
numChildren = 0
```

进入客户端后输入命令

```
ls /consumers
```

consumer文件夹中是zookeeper中维护的消费者组。

注：当然我们第一次运行程序的时候，在zookeeper中是没有offsetTest消费者组的信息的。新版本已经不维护在zookeeper中，维护在broker集群中的__consumer_offset中，所以在zookeeper中没有消费者组的信息

假设我们已经运行了一此程序，在zookeeoer成功创建了offsetTest消费者组

在offsetTest消费者组维护的是offset，offset文件夹下则维护的是各个主题的文件夹，每个主题的文件夹下又是各个分区的信息，在每个分区的信息中第一行就是维护的偏移量。各个分区的信息如下

```
cZxid：节点创建时的zxid

ctime：节点创建时间

mZxid：节点最近一次更新时的zxid

```

```
mtime：节点最近一次更新的时间

cversion：子节点数据更新次数

dataVersion：本节点数据更新次数

aclVersion：节点ACL(授权信息)的更新次数

ephemeralOwner：如果该节点为临时节点,ephemeralOwner值表示与该节点绑定的session id. 如果
该节点不是临时节点,ephemeralOwner值为0

dataLength：节点数据长度，本例中为hello world的长度

numChildren：子节点个数
```

因此得到在zookeeper中最终的offset偏移量地址

```
/consumers/offsetTest/offsets/offsettest/各个分区：[0,1,2,....]
/消费者/各个消费者组文件夹/偏移量文件夹/各个topic文件夹
```

2. 实战代码

```
package com.bupt.sparkStreaming

import kafka.common.TopicAndPartition
import kafka.message.MessageAndMetadata
import kafka.serializer.StringDecoder
import kafka.utils.{ZKGroupTopicDirs, ZkUtils}
import org.I0Itec.zkclient.ZkClient
import org.apache.kafka.clients.consumer.ConsumerConfig
import org.apache.log4j.{Level, Logger}
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.dstream.InputDStream
import org.apache.spark.streaming.kafka.{HasOffsetRanges, KafkaUtils}

object KafkaDirect_ZK_Offset {
  def main(args: Array[String]): Unit = {
    Logger.getLogger("org.apache.spark").setLevel(Level.OFF)
    val conf: SparkConf = new
SparkConf().setAppName("KafkaDirect_ZK_Offset").setMaster("local[*]")
    val ssc: StreamingContext = new StreamingContext(conf, Seconds(5))
    val groupId = "offsetTest"
    val brokers = "hadoop101:9092"
    /**
      * kafka参数列表
      */
    val kafkaParams = Map[String, String](
      ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG -> brokers,
      ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG ->
"org.apache.kafka.common.serialization.StringDeserializer",
      ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG ->
"org.apache.kafka.common.serialization.StringDeserializer",
      ConsumerConfig.GROUP_ID_CONFIG -> groupId
    )

    val topic = "offsettest"
    // val topics = ArrayList(topic)


    val zKGroupTopicDirs: ZKGroupTopicDirs = new ZKGroupTopicDirs(groupId,
topic)
    /**
      * 生成的目录结构
      * /customer/offsetTest/offsets/offsettest
      */

    val offsetDir: String = zKGroupTopicDirs.consumerOffsetDir
    //zk字符串连接组
    // val zkGroups = "hadoop101:2181,hadoop102:2181"
    val zkGroups = "192.168.1.102:2181"
    println(offsetDir)
    //创建一个zkClient连接
    val zkClient: ZkClient = new ZkClient(zkGroups)
    //子节点的数量
    val childrenCount: Int = zkClient.countChildren(offsetDir)

    //子节点的数量>0就说明非第一次
    var stream: InputDStream[(String, String)] = null
    if (childrenCount > 0) {
      println("已经启动过")
      //用来存储我们已经读取到的偏移量
      var fromOffset = Map[TopicAndPartition, Long]()
      (0 until childrenCount).foreach(partitionId => {
        val offset = zkClient.readData[String](offsetDir + s"/$partitionId")
        fromOffset += (new TopicAndPartition(topic, partitionId) ->
offset.toLong)
      })
      val mess = (mmd: MessageAndMetadata[String, String])=>
(mmd.key(),mmd.message()) //这个会将 kafka 的消息进行 transform，最终 kafka的数据都会
变成 (topic_name, message) 这样的 tuple
      stream =
KafkaUtils.createDirectStream[String,String,StringDecoder,StringDecoder,
(String,String)](ssc, kafkaParams,fromOffset,mess)
/*      for (tp <- fromOffset) {
        println(s"topAndPartition: ${tp._1.toString}  offset: ${tp._2}")
      }*/
    }
    else {
      println("第一次启动")
      stream = KafkaUtils.createDirectStream[String, String, StringDecoder,
StringDecoder](ssc, kafkaParams, Set(topic))

    }
    val flatMapStream = stream.flatMap(t => t._2.split(" "))
    val mapWithStateDStream = flatMapStream.map((_,1))
    var result = mapWithStateDStream.reduceByKey(_+_)
   // result.print()
    stream.foreachRDD(
      rdd => {
        //转换rdd为Array[OffsetRange]
        val offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges

        //计算逻辑

//        rdd.foreach(println)
//        rdd.partitions.foreach(println)
//        rdd.values.foreach(println)
        //自己存储数据，自己管理
        for (o <- offsetRanges) {
          //写入到zookeeper,第二个参数为是否启动安全
          println(s"topicAndPartition: ${o.topicAndPartition()} top: ${o.topic}
patition: ${o.partition} utiloffset: ${o.untilOffset} fromoffset:
${o.fromOffset}")
          val path = offsetDir +"/"+o.partition
       //   println(path)
          ZkUtils.updatePersistentPath(zkClient,path,o.untilOffset.toString)
        }
      }

    )
    ssc.start()
    ssc.awaitTermination()
  }
}
```

pom

```
 <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming_2.10</artifactId>
            <version>2.1.0</version>
        </dependency>

        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-streaming-kafka-0-8_2.10</artifactId>
            <version>2.2.0</version>
        </dependency>
        <!--<dependency>-->
            <!--<groupId>org.apache.spark</groupId>-->
            <!--<artifactId>spark-streaming-kafka_2.10</artifactId>-->
            <!--<version>1.0.0</version>-->
        <!--</dependency>-->
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.11.0.0</version>
        </dependency>
```