1：1、用Flume实时采集一个目录（自定）下文件数据，写入Kafka；

```
#a1 是agent名称
a1.sources = s1
a1.sinks = k1
a1.channels = c1

# Describe/configure the source
a1.sources.s1.type = spooldir
a1.sources.s1.spoolDir = /var/log/hadoop-hdfs
a1.sources.s1.batchSize = 200

# Describe the sink
a1.sinks.k1.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k1.kafka.topic = mytopic
a1.sinks.k1.kafka.bootstrap.servers = localhost:9092
a1.sinks.k1.kafka.flumeBatchSize = 20

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 1000

# 组合
a1.sources.s1.channels = c1
a1.sinks.k1.channel = c1
```

```
##########################################################################
```

2、用Spark Steaming消费Kafka，进行WordCount计算。

```
package com.ex

import org.apache.spark.SparkConf
import org.apache.spark.streaming.kafka.KafkaUtils
import org.apache.spark.streaming.{Seconds, StreamingContext}

object kafkaWordCount {
  def main(args: Array[String]): Unit = {
    if (args.length < 4) {
            System.err.println("Usage: KafkaWordCountProducer
            <metadataBrokerList> <topic> " +
              "<messagesPerSec> <wordsPerMessage>")
            System.exit(1)
          }

    // val Array(brokers, topic, messagesPerSec, wordsPerMessage) = args

    val zkQuorum = "slave1:2181"
    val group = "g1"
    val topics = "myTopic"
    val numThreads = 2

    val sparkConf = new
    SparkConf().setAppName("KafkaWordCount").setMaster("local[2]")
      .set("spark.testing.memory", "571859200")

    val ssc = new StreamingContext(sparkConf, Seconds(2))
    ssc.checkpoint("hdfs://master:8020/user/root/checkpoint/wordCount")
    //设置有状态的检查点，存储总数值

    val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group,
```

```
    topicMap).map(_._2)

    val words = lines.flatMap(_.split(" "))
    val wordCounts = words.map(x => (x, 1))    //.reduceByKey(_ + _)
    //每个批次进行局部汇总

    val addFunc = (currValues: Seq[Int], prevValueState: Option[Int]) => {

      //通过Spark内部的reduceByKey按key归约，然后这里传入某key当前批次的Seq,
      再计算每个key的总和
      val currentCount = currValues.sum
      // 已累加的值
      val previousCount = prevValueState.getOrElse(0)
      // 返回累加后的结果，是一个Option[Int]类型
      Some(currentCount + previousCount)
    }

    wordCounts.updateStateByKey[Int](addFunc).print()
    //对pairRDD里的每个key的values进行addFunc处理
    ssc.start()
    ssc.awaitTermination()
  }
}
```