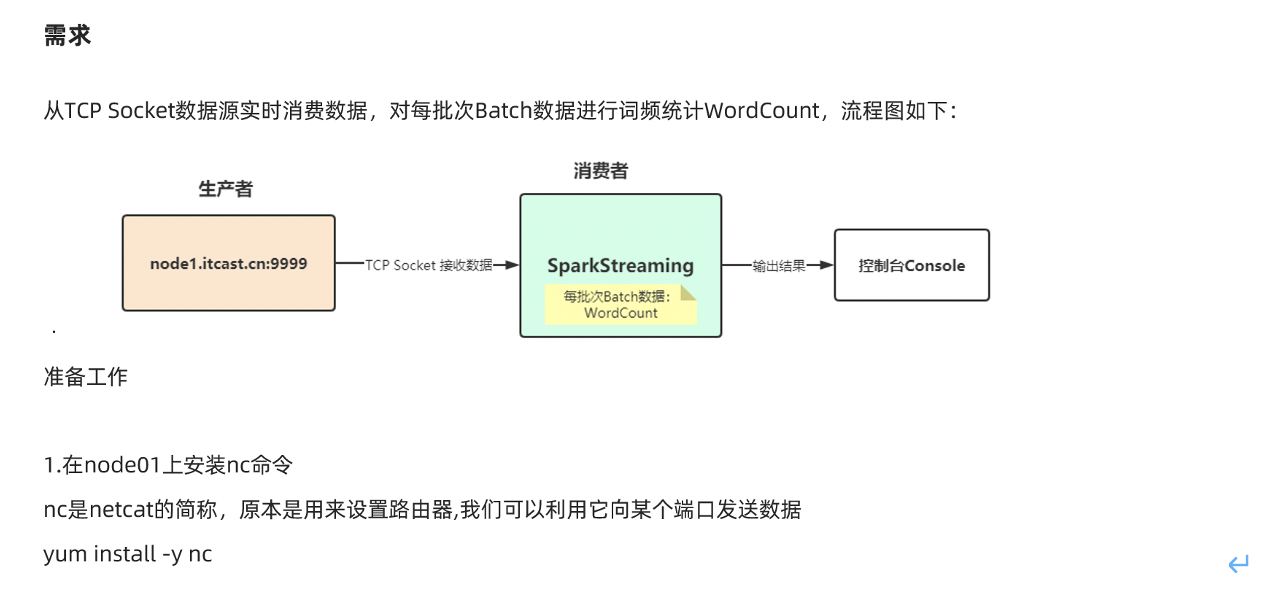
# SparkStreaming案例

## 案例1-WordCount

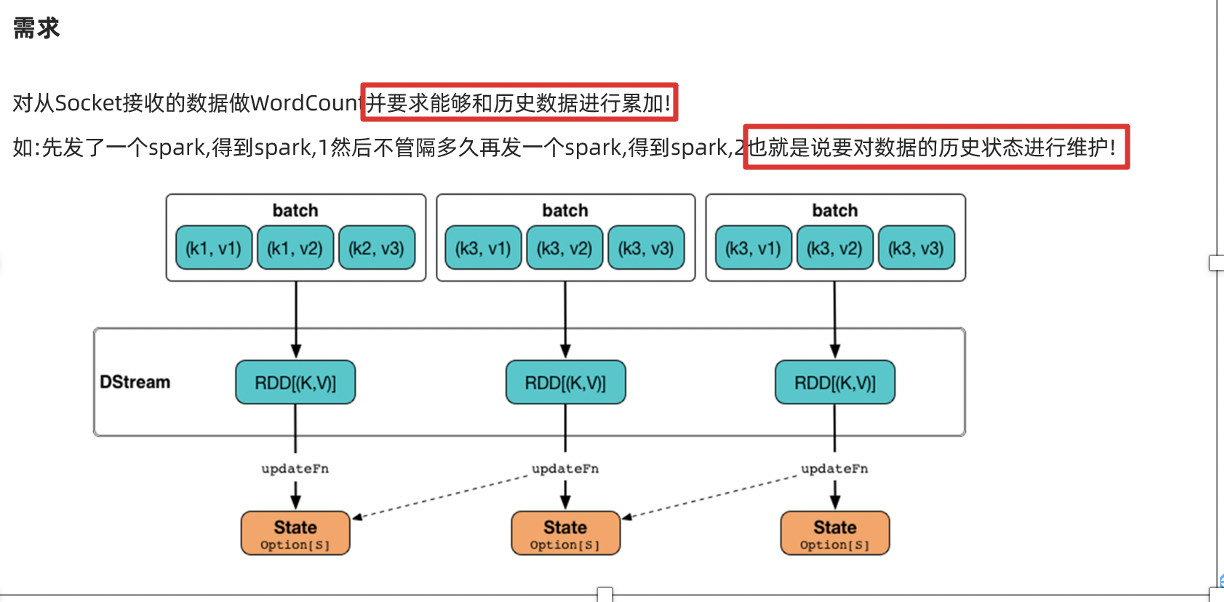


yum install -y nc

https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/streaming/NetworkWordCount.scala

package com.as.streaming  
  
import org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream}  
import org.apache.spark.{SparkConf, SparkContext, streaming}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 使用SparkStreaming接收node1:9999的数据并做WordCount  
 \*/  
object WordCount01 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc,Seconds(5))//每隔5s划分一个批次  
  
 //TODO 1.加载数据  
 val lines: ReceiverInputDStream[String] = ssc.socketTextStream("node1",9999)  
  
 //TODO 2.处理数据  
 val resultDS: DStream[(String, Int)] = lines.flatMap(\_.split(" "))  
 .map((\_, 1))  
 .reduceByKey(\_ + \_)  
  
 //TODO 3.输出结果  
 resultDS.print()  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination()//注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true)//优雅关闭  
 }  
}

## 案例2-状态管理

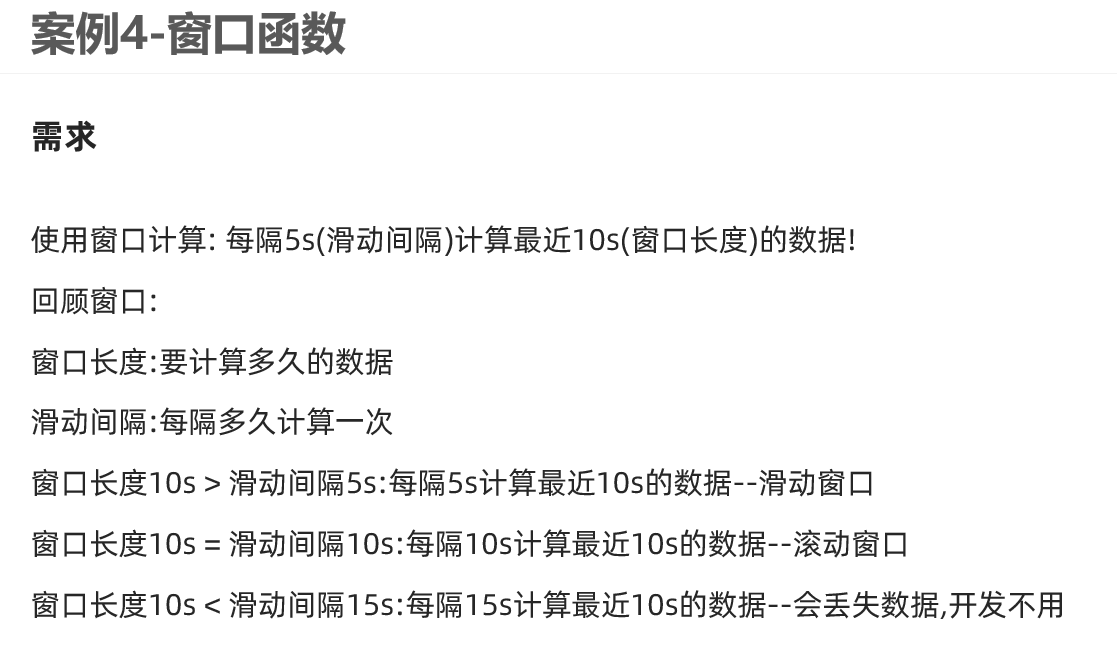


package com.as.streaming  
  
import org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 使用SparkStreaming接收node1:9999的数据并做WordCount+实现状态管理:  
 \* 如输入spark hadoop 得到(spark,1),(hadoop,1)  
 \* 再下一个批次在输入 spark spark,得到(spark,3)  
 \*/  
object WordCount02 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc, Seconds(5)) //每隔5s划分一个批次  
  
 //The checkpoint directory has not been set. Please set it by StreamingContext.checkpoint().  
 //注意:state存在checkpoint中  
 ssc.checkpoint("./ckp")  
  
 //TODO 1.加载数据  
 val lines: ReceiverInputDStream[String] = ssc.socketTextStream("node1", 9999)  
  
 //TODO 2.处理数据  
 //定义一个函数用来处理状态:把当前数据和历史状态进行累加  
 //currentValues:表示该key(如:spark)的当前批次的值,如:[1,1]  
 //historyValue:表示该key(如:spark)的历史值,第一次是0,后面就是之前的累加值如1  
 val updateFunc = (currentValues: Seq[Int], historyValue: Option[Int]) => {  
 if (currentValues.size > 0) {  
 val currentResult: Int = currentValues.sum + historyValue.getOrElse(0)  
 Some(currentResult)  
 } else {  
 historyValue  
 }  
 }  
  
 val resultDS: DStream[(String, Int)] = lines.flatMap(\_.split(" "))  
 .map((\_, 1))  
 //.reduceByKey(\_ + \_)  
 // updateFunc: (Seq[V], Option[S]) => Option[S]  
 .updateStateByKey(updateFunc)  
  
 //TODO 3.输出结果  
 resultDS.print()  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination() //注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true) //优雅关闭  
 }  
}

## 案例3-状态恢复-扩展

package com.as.streaming  
  
import org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 使用SparkStreaming接收node1:9999的数据并做WordCount+实现状态管理+状态恢复  
 \* 如输入spark hadoop 得到(spark,1),(hadoop,1)  
 \* 再下一个批次在输入 spark spark,得到(spark,3)  
 \*/  
object WordCount03 {  
 def creatingFunc():StreamingContext ={  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc, Seconds(5)) //每隔5s划分一个批次  
  
 //The checkpoint directory has not been set. Please set it by StreamingContext.checkpoint().  
 //注意:state存在checkpoint中  
 ssc.checkpoint("./ckp")  
  
 //TODO 1.加载数据  
 val lines: ReceiverInputDStream[String] = ssc.socketTextStream("node1", 9999)  
  
 //TODO 2.处理数据  
 //定义一个函数用来处理状态:把当前数据和历史状态进行累加  
 //currentValues:表示该key(如:spark)的当前批次的值,如:[1,1]  
 //historyValue:表示该key(如:spark)的历史值,第一次是0,后面就是之前的累加值如1  
 val updateFunc = (currentValues: Seq[Int], historyValue: Option[Int]) => {  
 if (currentValues.size > 0) {  
 val currentResult: Int = currentValues.sum + historyValue.getOrElse(0)  
 Some(currentResult)  
 } else {  
 historyValue  
 }  
 }  
  
 val resultDS: DStream[(String, Int)] = lines.flatMap(\_.split(" "))  
 .map((\_, 1))  
 //.reduceByKey(\_ + \_)  
 // updateFunc: (Seq[V], Option[S]) => Option[S]  
 .updateStateByKey(updateFunc)  
  
 //TODO 3.输出结果  
 resultDS.print()  
  
 ssc  
 }  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val ssc: StreamingContext = StreamingContext.getOrCreate("./ckp", creatingFunc \_)  
 ssc.sparkContext.setLogLevel("WARN")  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination() //注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true) //优雅关闭  
 }  
}

## 案例4-窗口计算

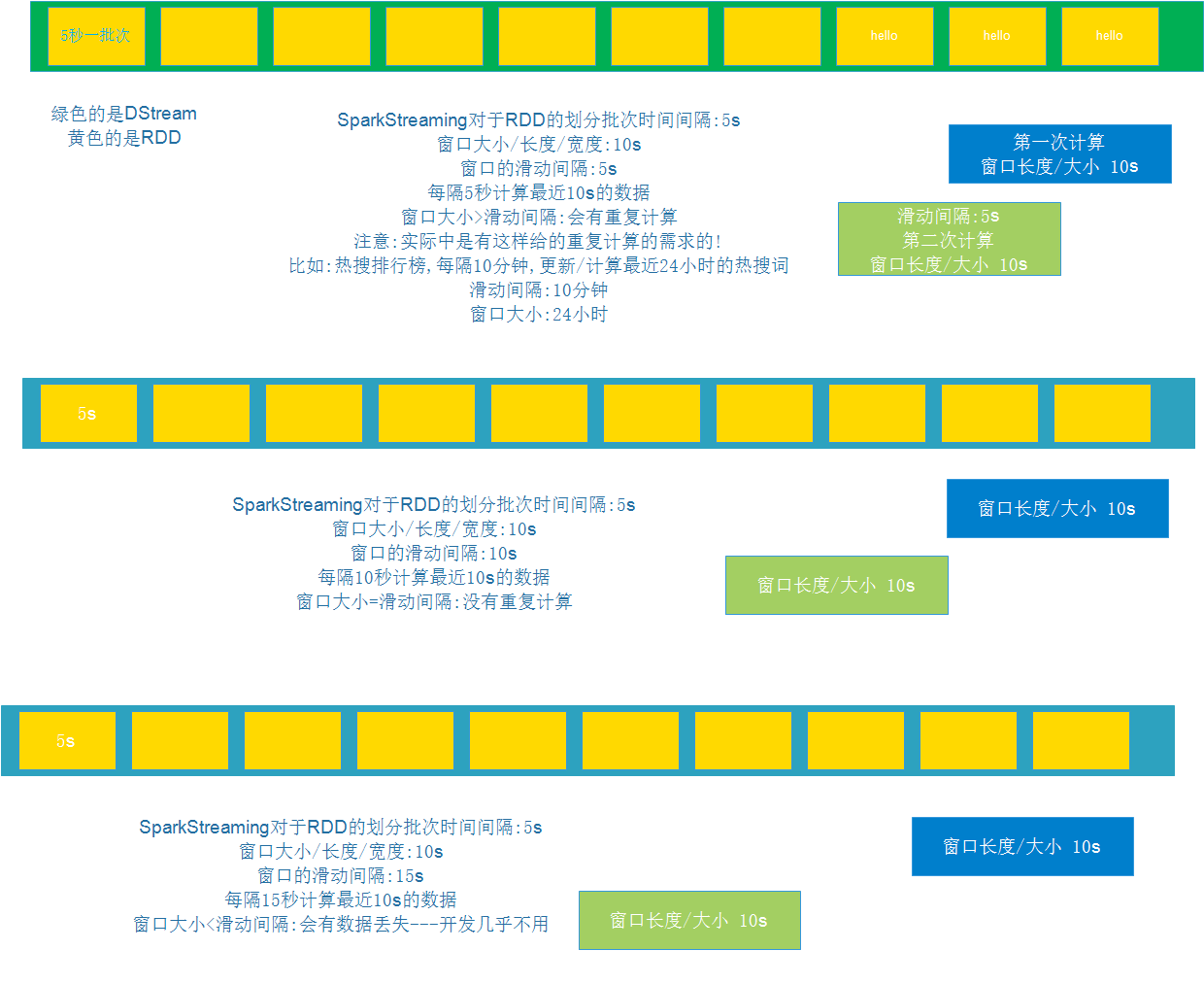


如实际开发中:

每隔1min计算最近24小时的热搜排行榜

每隔10s计算最近10分钟的广告点击量

每隔1h计算最近7天的热搜



package com.as.streaming  
  
import org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 使用SparkStreaming接收node1:9999的数据并做WordCount+窗口计算  
 \* 每隔5s计算最近10s的数据  
 \*/  
object WordCount04 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc,Seconds(5))//每隔5s划分一个批次  
  
 //TODO 1.加载数据  
 val lines: ReceiverInputDStream[String] = ssc.socketTextStream("node1",9999)  
  
 //TODO 2.处理数据  
 val resultDS: DStream[(String, Int)] = lines.flatMap(\_.split(" "))  
 .map((\_, 1))  
 //.reduceByKey(\_ + \_)  
 // windowDuration :窗口长度/窗口大小,表示要计算最近多长时间的数据  
 // slideDuration : 滑动间隔,表示每隔多长时间计算一次  
 // 注意:windowDuration和slideDuration必须是batchDuration的倍数  
 // 每隔5s(滑动间隔)计算最近10s(窗口长度/窗口大小)的数据  
 //reduceByKeyAndWindow(聚合函数,windowDuration,slideDuration)  
 //.reduceByKeyAndWindow(\_+\_,Seconds(10),Seconds(5))  
 .reduceByKeyAndWindow((a:Int,b:Int)=>a+b,Seconds(10),Seconds(5))  
 //实际开发中需要我们掌握的是如何根据需求设置windowDuration和slideDuration  
 //如:  
 //每隔10分钟(滑动间隔slideDuration)更新最近24小时(窗口长度windowDuration)的广告点击数量  
 // .reduceByKeyAndWindow((a:Int,b:Int)=>a+b,Minutes(60\*24),Minutes(10))  
  
 //TODO 3.输出结果  
 resultDS.print()  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination()//注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true)//优雅关闭  
 }  
}

## 案例5-topN



package com.as.streaming  
  
import org.apache.spark.rdd.RDD  
import org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 使用SparkStreaming接收node1:9999的数据并做WordCount+窗口计算  
 \* 模拟百度热搜排行榜每隔10s计算最近20s的热搜词  
 \*/  
object WordCount05 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc,Seconds(5))//每隔5s划分一个批次  
  
 //TODO 1.加载数据  
 val lines: ReceiverInputDStream[String] = ssc.socketTextStream("node1",9999)  
  
 //TODO 2.处理数据  
 val resultDS: DStream[(String, Int)] = lines.flatMap(\_.split(" "))  
 .map((\_, 1))  
 //模拟百度热搜排行榜每隔10s计算最近20s的热搜词Top3  
 //windowDuration: Duration,  
 //slideDuration: Duration  
 .reduceByKeyAndWindow((a: Int, b: Int) => a + b, Seconds(20), Seconds(10))  
 //注意DStream没有提供直接排序的方法,所以需要直接对底层的RDD操作  
 //DStream的transform方法表示对DStream底层的RDD进行操作并返回结果  
 val sortedResultDS: DStream[(String, Int)] = resultDS.transform(rdd => {  
 val sortRDD: RDD[(String, Int)] = rdd.sortBy(\_.\_2, false)  
 val top3: Array[(String, Int)] = sortRDD.take(3)  
 println("=======top3=====")  
 top3.foreach(println)  
 println("=======top3=====")  
 sortRDD  
 })  
  
 //TODO 3.输出结果  
 sortedResultDS.print()  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination()//注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true)//优雅关闭  
 }  
}  
/\*  
31省新增本土确诊23例:河北20例 31省新增本土确诊23例:河北20例 31省新增本土确诊23例:河北20例 31省新增本土确诊23例:河北20例  
特朗普签令禁止与8款中国应用交易 特朗普签令禁止与8款中国应用交易  
纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉  
多家航空公司发布进出京退改票方案 多家航空公司发布进出京退改票方案 多家航空公司发布进出京退改票方案  
石家庄中小学幼儿园暂停线下教学  
 \*/

## 案例6-自定义输出



package com.as.streaming  
  
import java.sql.{Connection, DriverManager, PreparedStatement, Timestamp}  
  
import org.apache.spark.rdd.RDD  
import org.apache.spark.streaming.dstream.{DStream, ReceiverInputDStream}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 使用SparkStreaming接收node1:9999的数据并做WordCount+窗口计算  
 \* 模拟百度热搜排行榜每隔10s计算最近20s的热搜词  
 \* 最后使用自定义输出将结果输出到控制台/HDFS/MySQL  
 \*/  
object WordCount06 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc,Seconds(5))//每隔5s划分一个批次  
  
 //TODO 1.加载数据  
 val lines: ReceiverInputDStream[String] = ssc.socketTextStream("node1",9999)  
  
 //TODO 2.处理数据  
 val resultDS: DStream[(String, Int)] = lines.flatMap(\_.split(" "))  
 .map((\_, 1))  
 //模拟百度热搜排行榜每隔10s计算最近20s的热搜词Top3  
 //windowDuration: Duration,  
 //slideDuration: Duration  
 .reduceByKeyAndWindow((a: Int, b: Int) => a + b, Seconds(20), Seconds(10))  
 //注意DStream没有提供直接排序的方法,所以需要直接对底层的RDD操作  
 //DStream的transform方法表示对DStream底层的RDD进行操作并返回结果  
 val sortedResultDS: DStream[(String, Int)] = resultDS.transform(rdd => {  
 val sortRDD: RDD[(String, Int)] = rdd.sortBy(\_.\_2, false)  
 val top3: Array[(String, Int)] = sortRDD.take(3)  
 println("=======top3=====")  
 top3.foreach(println)  
 println("=======top3=====")  
 sortRDD  
 })  
  
 //TODO 3.输出结果  
 sortedResultDS.print()//默认的输出  
 //自定义输出  
 sortedResultDS.foreachRDD((rdd,time)=>{  
 val milliseconds: Long = time.milliseconds  
 println("------自定义输出---------")  
 println("batchtime:"+milliseconds)  
 println("------自定义输出---------")  
 //最后使用自定义输出将结果输出到控制台/HDFS/MySQL  
 //输出到控制台  
 rdd.foreach(println)  
 //输出到HDFS  
 rdd.coalesce(1).saveAsTextFile("data/output/result-"+milliseconds)  
 //输出到MySQL  
 /\*  
CREATE TABLE `t\_hotwords` (  
 `time` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  
 `word` varchar(255) NOT NULL,  
 `count` int(11) DEFAULT NULL,  
 PRIMARY KEY (`time`,`word`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
 \*/  
 rdd.foreachPartition(iter=>{  
 //开启连接  
 val conn: Connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/bigdata?characterEncoding=UTF-8","root","root")  
 val sql:String = "INSERT INTO `t\_hotwords` (`time`, `word`, `count`) VALUES (?, ?, ?);"  
 val ps: PreparedStatement = conn.prepareStatement(sql)  
 iter.foreach(t=>{  
 val word: String = t.\_1  
 val count: Int = t.\_2  
 ps.setTimestamp(1,new Timestamp(milliseconds) )  
 ps.setString(2,word)  
 ps.setInt(3,count)  
 ps.addBatch()  
 })  
 ps.executeBatch()  
 //关闭连接  
 if (conn != null) conn.close()  
 if (ps != null) ps.close()  
 })  
 })  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination()//注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true)//优雅关闭  
 }  
}  
/\*  
31省新增本土确诊23例:河北20例 31省新增本土确诊23例:河北20例 31省新增本土确诊23例:河北20例 31省新增本土确诊23例:河北20例  
特朗普签令禁止与8款中国应用交易 特朗普签令禁止与8款中国应用交易  
纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉 纸张价格上涨直逼猪肉  
多家航空公司发布进出京退改票方案 多家航空公司发布进出京退改票方案 多家航空公司发布进出京退改票方案  
石家庄中小学幼儿园暂停线下教学  
 \*/

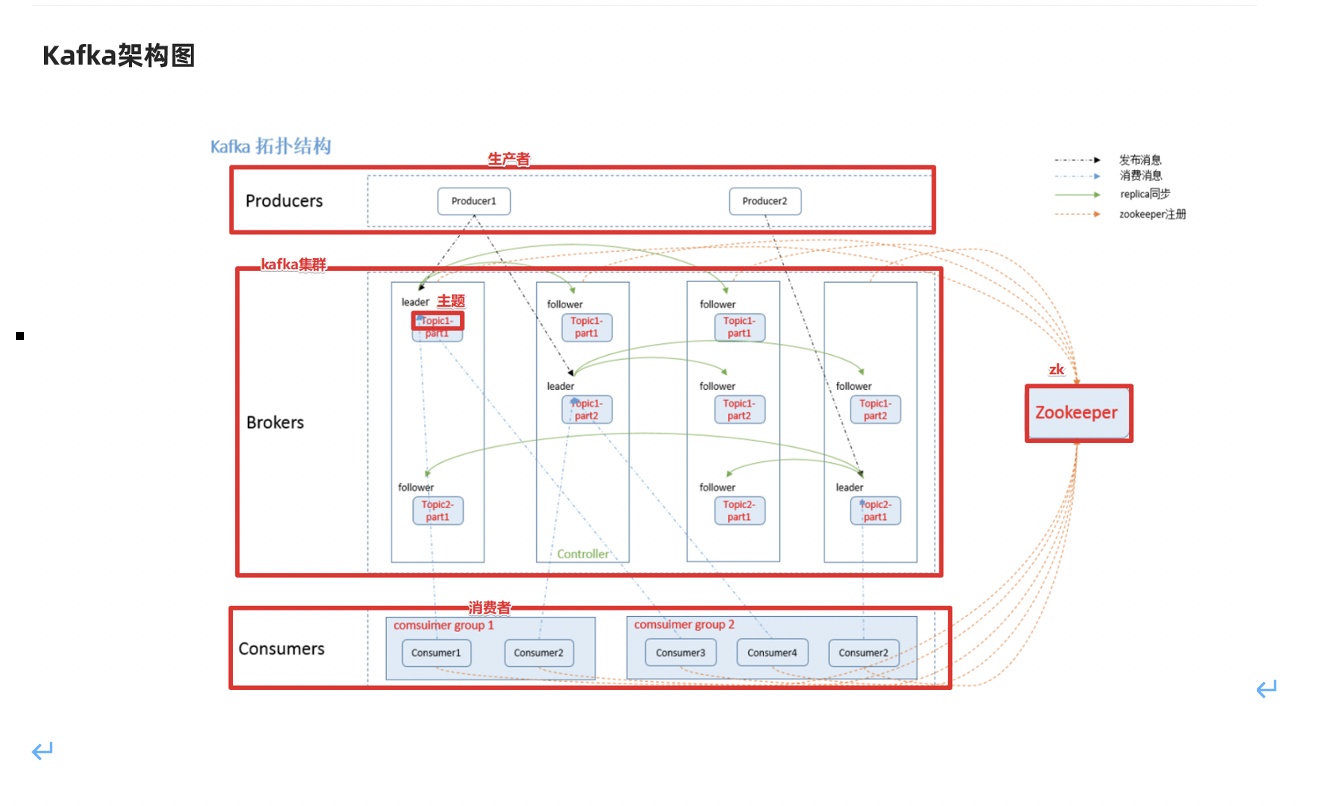
# SparkStreaming整合Kafka

## 前置说明

### SparkStreaming+Kafka流程

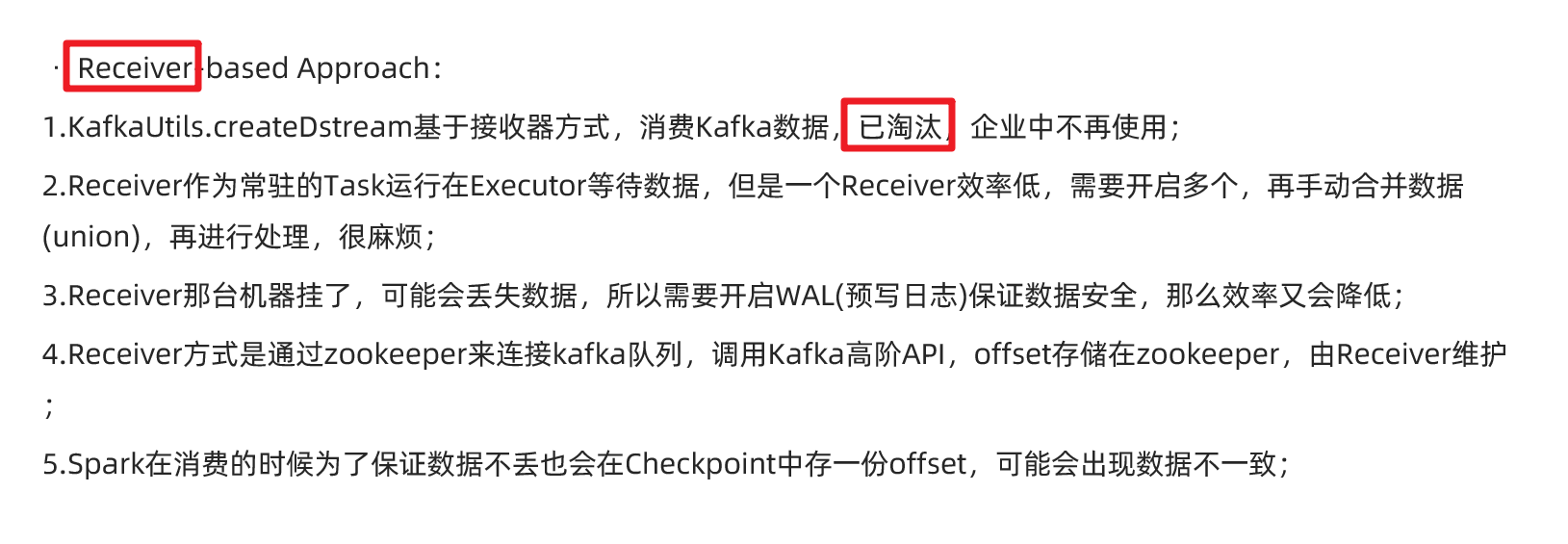
流式数据 ---> (Flume)---->Kafka--->SparkStreaming/StructStreaming/Flink--->Redis/HBase/HDFS

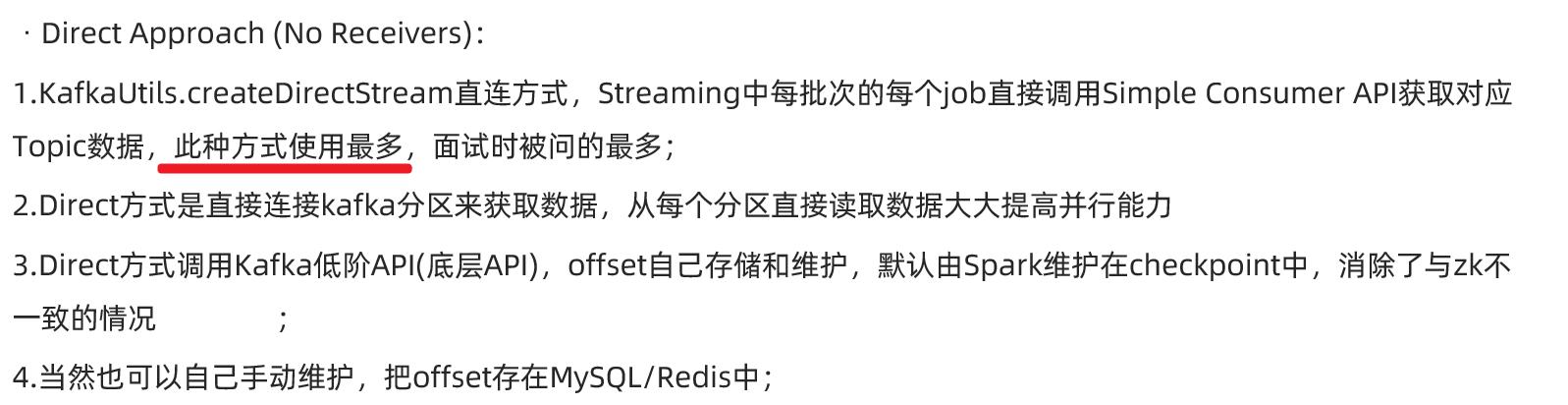
### Kafka

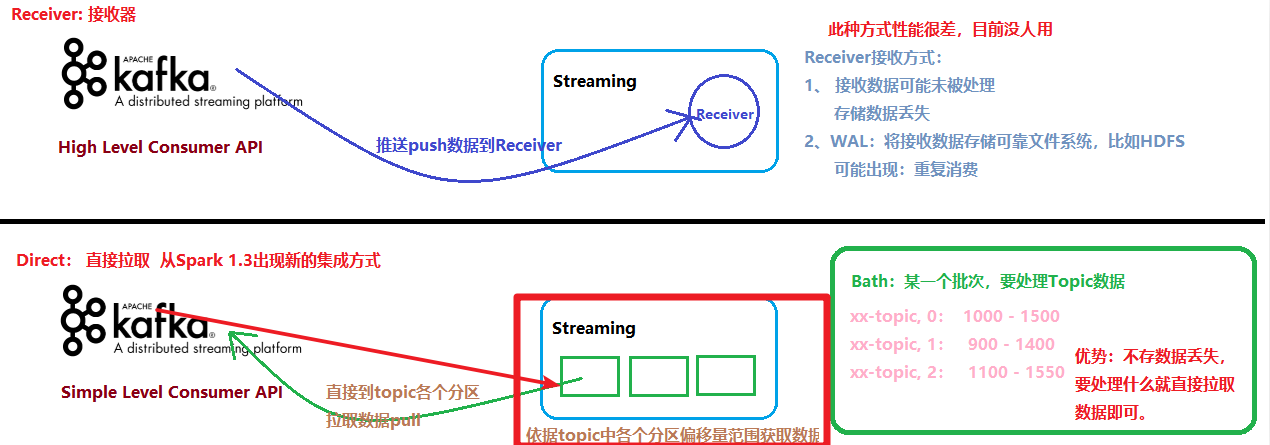


#启动kafka  
/export/server/kafka/bin/kafka-server-start.sh -daemon /export/server/kafka/config/server.properties   
  
#停止kafka  
/export/server/kafka/bin/kafka-server-stop.sh   
  
#查看topic信息  
/export/server/kafka/bin/kafka-topics.sh --list --zookeeper node1:2181  
   
#创建topic  
/export/server/kafka/bin/kafka-topics.sh --create --zookeeper node1:2181 --replication-factor 1 --partitions 3 --topic test  
   
#查看某个topic信息  
/export/server/kafka/bin/kafka-topics.sh --describe --zookeeper node1:2181 --topic test  
   
#删除topic  
/export/server/kafka/bin/kafka-topics.sh --zookeeper node1:2181 --delete --topic test  
   
#启动生产者--控制台的生产者--一般用于测试  
/export/server/kafka/bin/kafka-console-producer.sh --broker-list node1:9092 --topic spark\_kafka  
  
   
# 启动消费者--控制台的消费者  
/export/server/kafka/bin/kafka-console-consumer.sh --bootstrap-server node1:9092 --topic spark\_kafka --from-beginning

### SparkStreaming连接Kafka两种方式

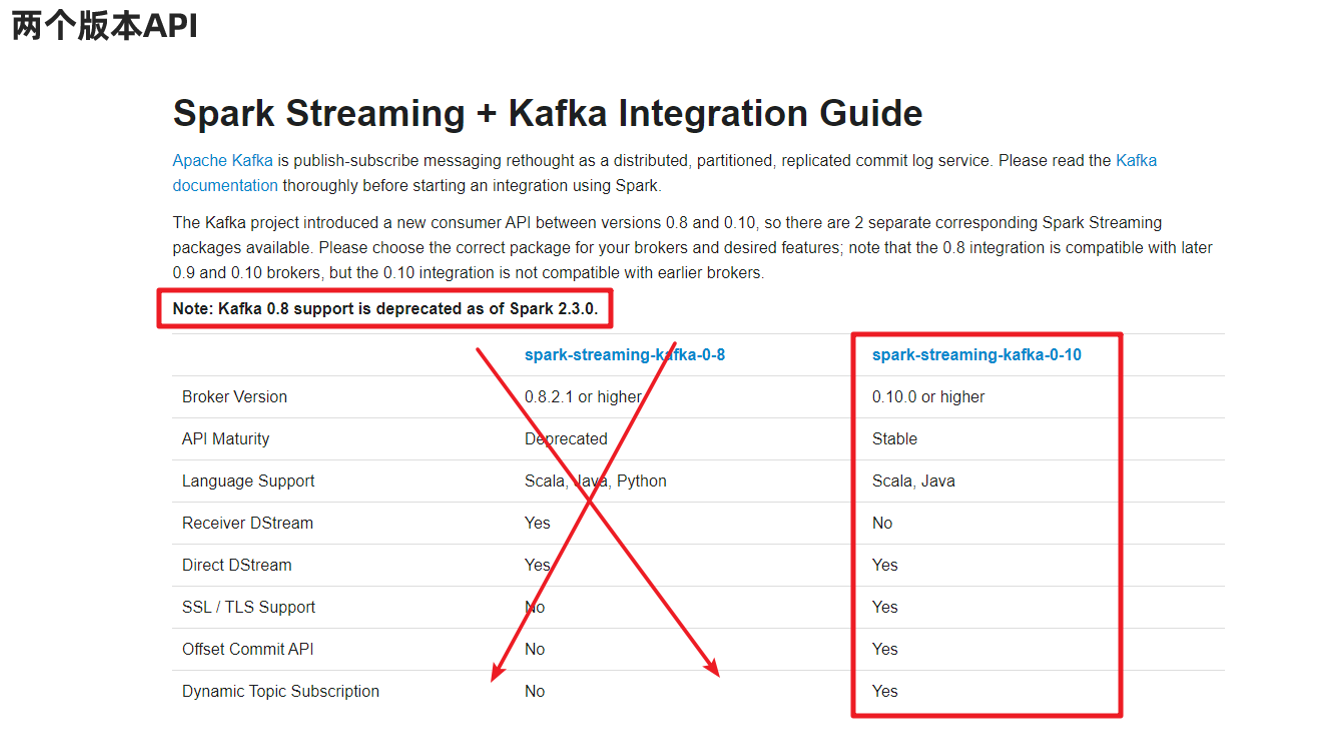






### 两种API





### 总结

在学习和开发中都是直接使用spark-Streaming-kafka-0.10版本API中的Direct模式来连接Kafka!

## 代码演示-1-自动提交偏移量

http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html

自动提交偏移量到默认主题和Checkpoint中

package com.as.streaming  
  
import org.apache.kafka.clients.consumer.ConsumerRecord  
import org.apache.kafka.common.serialization.StringDeserializer  
import org.apache.spark.streaming.dstream.{DStream, InputDStream}  
import org.apache.spark.streaming.kafka010.{ConsumerStrategies, KafkaUtils, LocationStrategies}  
import org.apache.spark.{SparkConf, SparkContext}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示使用spark-streaming-kafka-0-10\_2.12中的Direct模式连接Kafka消费数据  
 \*/  
object SparkStreaming\_Kafka\_Demo01 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc,Seconds(5))//每隔5s划分一个批次  
 ssc.checkpoint("./ckp")  
  
 //TODO 1.加载数据-从Kafka  
 val kafkaParams = Map[String, Object](  
 "bootstrap.servers" -> "node1:9092",//kafka集群地址  
 "key.deserializer" -> classOf[StringDeserializer],//key的反序列化规则  
 "value.deserializer" -> classOf[StringDeserializer],//value的反序列化规则  
 "group.id" -> "sparkdemo",//消费者组名称  
 //earliest:表示如果有offset记录从offset记录开始消费,如果没有从最早的消息开始消费  
 //latest:表示如果有offset记录从offset记录开始消费,如果没有从最后/最新的消息开始消费  
 //none:表示如果有offset记录从offset记录开始消费,如果没有就报错  
 "auto.offset.reset" -> "latest",  
 "auto.commit.interval.ms"->"1000",//自动提交的时间间隔  
 "enable.auto.commit" -> (true: java.lang.Boolean)//是否自动提交  
 )  
 val topics = Array("spark\_kafka")//要订阅的主题  
 //使用工具类从Kafka中消费消息  
 val kafkaDS: InputDStream[ConsumerRecord[String, String]] = KafkaUtils.createDirectStream[String, String](  
 ssc,  
 LocationStrategies.PreferConsistent, //位置策略,使用源码中推荐的  
 ConsumerStrategies.Subscribe[String, String](topics, kafkaParams) //消费策略,使用源码中推荐的  
 )  
  
 //TODO 2.处理消息  
 val infoDS: DStream[String] = kafkaDS.map(record => {  
 val topic: String = record.topic()  
 val partition: Int = record.partition()  
 val offset: Long = record.offset()  
 val key: String = record.key()  
 val value: String = record.value()  
 val info: String = s"""topic:${topic}, partition:${partition}, offset:${offset}, key:${key}, value:${value}"""  
 info  
 })  
  
 //TODO 3.输出结果  
 infoDS.print()  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination()//注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true)//优雅关闭  
 }  
}  
//测试:  
//1.准备kafka  
// /export/server/kafka/bin/kafka-topics.sh --list --zookeeper node1:2181  
// /export/server/kafka/bin/kafka-topics.sh --create --zookeeper node1:2181 --replication-factor 1 --partitions 3 --topic spark\_kafka  
// /export/server/kafka/bin/kafka-console-producer.sh --broker-list node1:9092 --topic spark\_kafka  
//2.启动程序  
//3.发送数据  
//4.观察结果

## 代码演示-2-手动提交

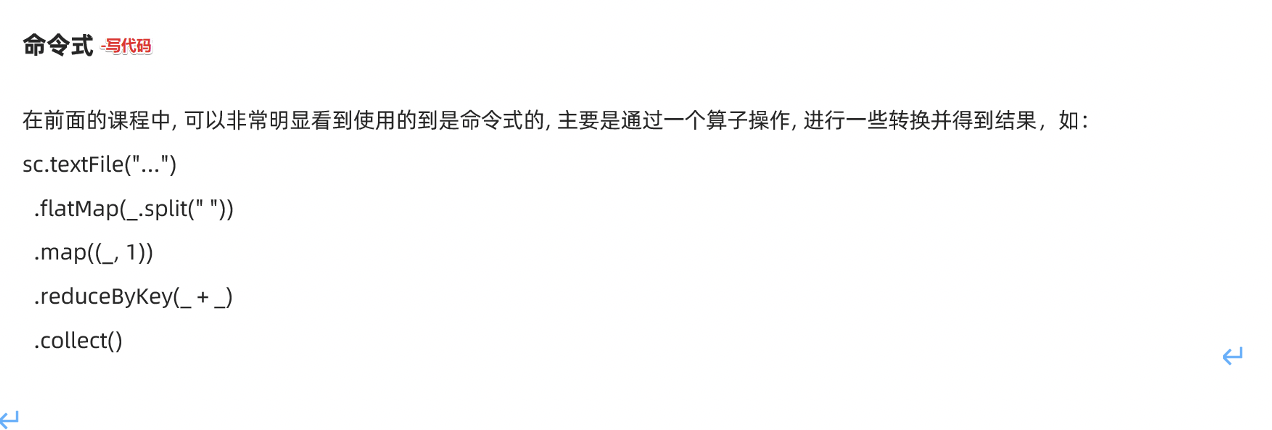
package com.as.streaming  
  
import org.apache.kafka.clients.consumer.ConsumerRecord  
import org.apache.kafka.common.serialization.StringDeserializer  
import org.apache.spark.streaming.dstream.{DStream, InputDStream}  
import org.apache.spark.streaming.kafka010.{CanCommitOffsets, ConsumerStrategies, HasOffsetRanges, KafkaUtils, LocationStrategies, OffsetRange}  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import org.apache.spark.{SparkConf, SparkContext}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示使用spark-streaming-kafka-0-10\_2.12中的Direct模式连接Kafka消费数据+手动提交offset  
 \*/  
object SparkStreaming\_Kafka\_Demo02 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc, Seconds(5)) //每隔5s划分一个批次  
 ssc.checkpoint("./ckp")  
  
 //TODO 1.加载数据-从Kafka  
 val kafkaParams = Map[String, Object](  
 "bootstrap.servers" -> "node1:9092", //kafka集群地址  
 "key.deserializer" -> classOf[StringDeserializer], //key的反序列化规则  
 "value.deserializer" -> classOf[StringDeserializer], //value的反序列化规则  
 "group.id" -> "sparkdemo", //消费者组名称  
 //earliest:表示如果有offset记录从offset记录开始消费,如果没有从最早的消息开始消费  
 //latest:表示如果有offset记录从offset记录开始消费,如果没有从最后/最新的消息开始消费  
 //none:表示如果有offset记录从offset记录开始消费,如果没有就报错  
 "auto.offset.reset" -> "latest",  
 //"auto.commit.interval.ms"->"1000",//自动提交的时间间隔  
 "enable.auto.commit" -> (false: java.lang.Boolean) //是否自动提交  
 )  
 val topics = Array("spark\_kafka") //要订阅的主题  
 //使用工具类从Kafka中消费消息  
 val kafkaDS: InputDStream[ConsumerRecord[String, String]] = KafkaUtils.createDirectStream[String, String](  
 ssc,  
 LocationStrategies.PreferConsistent, //位置策略,使用源码中推荐的  
 ConsumerStrategies.Subscribe[String, String](topics, kafkaParams) //消费策略,使用源码中推荐的  
 )  
  
 //TODO 2.处理消息  
 //注意提交的时机:应该是消费完一小批就该提交一次offset,而在DStream一小批的体现是RDD  
 kafkaDS.foreachRDD(rdd => {  
 if(!rdd.isEmpty()){  
 //消费  
 rdd.foreach(record => {  
 val topic: String = record.topic()  
 val partition: Int = record.partition()  
 val offset: Long = record.offset()  
 val key: String = record.key()  
 val value: String = record.value()  
 val info: String = s"""topic:${topic}, partition:${partition}, offset:${offset}, key:${key}, value:${value}"""  
 println("消费到的消息的详细信息为: "+info)  
 })  
 //获取rdd中offset相关的信息:offsetRanges里面就包含了该批次各个分区的offset信息  
 val offsetRanges: Array[OffsetRange] = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  
 //提交  
 kafkaDS.asInstanceOf[CanCommitOffsets].commitAsync(offsetRanges)  
 println("当前批次的数据已消费并手动提交")  
 }  
 })  
  
 //TODO 3.输出结果  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination() //注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true) //优雅关闭  
 }  
}  
  
//测试:  
//1.准备kafka  
// /export/server/kafka/bin/kafka-topics.sh --list --zookeeper node1:2181  
// /export/server/kafka/bin/kafka-topics.sh --create --zookeeper node1:2181 --replication-factor 1 --partitions 3 --topic spark\_kafka  
// /export/server/kafka/bin/kafka-console-producer.sh --broker-list node1:9092 --topic spark\_kafka  
//2.启动程序  
//3.发送数据  
//4.观察结果

## 代码演示-3-手动提交到MySQL-扩展

package com.as.streaming  
  
import java.sql.{DriverManager, ResultSet}  
  
import org.apache.kafka.clients.consumer.ConsumerRecord  
import org.apache.kafka.common.TopicPartition  
import org.apache.kafka.common.serialization.StringDeserializer  
import org.apache.spark.streaming.dstream.InputDStream  
import org.apache.spark.streaming.kafka010.\_  
import org.apache.spark.streaming.{Seconds, StreamingContext}  
import org.apache.spark.{SparkConf, SparkContext}  
  
import scala.collection.mutable  
  
/\*\*  
 \* Author roy  
 \* Desc 演示使用spark-streaming-kafka-0-10\_2.12中的Direct模式连接Kafka消费数据+手动提交offset到MySQL  
 \*/  
object SparkStreaming\_Kafka\_Demo03 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val conf: SparkConf = new SparkConf().setAppName("spark").setMaster("local[\*]")  
 val sc: SparkContext = new SparkContext(conf)  
 sc.setLogLevel("WARN")  
 //the time interval at which streaming data will be divided into batches  
 val ssc: StreamingContext = new StreamingContext(sc, Seconds(5)) //每隔5s划分一个批次  
 ssc.checkpoint("./ckp")  
  
 //TODO 1.加载数据-从Kafka  
 val kafkaParams = Map[String, Object](  
 "bootstrap.servers" -> "node1:9092", //kafka集群地址  
 "key.deserializer" -> classOf[StringDeserializer], //key的反序列化规则  
 "value.deserializer" -> classOf[StringDeserializer], //value的反序列化规则  
 "group.id" -> "sparkdemo", //消费者组名称  
 //earliest:表示如果有offset记录从offset记录开始消费,如果没有从最早的消息开始消费  
 //latest:表示如果有offset记录从offset记录开始消费,如果没有从最后/最新的消息开始消费  
 //none:表示如果有offset记录从offset记录开始消费,如果没有就报错  
 "auto.offset.reset" -> "latest",  
 //"auto.commit.interval.ms"->"1000",//自动提交的时间间隔  
 "enable.auto.commit" -> (false: java.lang.Boolean) //是否自动提交  
 )  
 val topics = Array("spark\_kafka") //要订阅的主题  
  
 //Map[主题分区, offset]  
 val offsetsMap: mutable.Map[TopicPartition, Long] = OffsetUtil.getOffsetMap("sparkdemo","spark\_kafka")  
 val kafkaDS: InputDStream[ConsumerRecord[String, String]] = if(offsetsMap.size > 0){  
 println("MySQL中存储了该消费者组消费该主题的偏移量记录,接下来从记录处开始消费")  
 //使用工具类从Kafka中消费消息  
 KafkaUtils.createDirectStream[String, String](  
 ssc,  
 LocationStrategies.PreferConsistent, //位置策略,使用源码中推荐的  
 ConsumerStrategies.Subscribe[String, String](topics, kafkaParams,offsetsMap) //消费策略,使用源码中推荐的  
 )  
 }else{  
 println("MySQL中没有存储该消费者组消费该主题的偏移量记录,接下来从latest开始消费")  
 //使用工具类从Kafka中消费消息  
 KafkaUtils.createDirectStream[String, String](  
 ssc,  
 LocationStrategies.PreferConsistent, //位置策略,使用源码中推荐的  
 ConsumerStrategies.Subscribe[String, String](topics, kafkaParams) //消费策略,使用源码中推荐的  
 )  
 }  
  
  
 //TODO 2.处理消息  
 //注意提交的时机:应该是消费完一小批就该提交一次offset,而在DStream一小批的体现是RDD  
 kafkaDS.foreachRDD(rdd => {  
 if(!rdd.isEmpty()){  
 //消费  
 rdd.foreach(record => {  
 val topic: String = record.topic()  
 val partition: Int = record.partition()  
 val offset: Long = record.offset()  
 val key: String = record.key()  
 val value: String = record.value()  
 val info: String = s"""topic:${topic}, partition:${partition}, offset:${offset}, key:${key}, value:${value}"""  
 println("消费到的消息的详细信息为: "+info)  
 })  
 //获取rdd中offset相关的信息:offsetRanges里面就包含了该批次各个分区的offset信息  
 val offsetRanges: Array[OffsetRange] = rdd.asInstanceOf[HasOffsetRanges].offsetRanges  
 //提交  
 //kafkaDS.asInstanceOf[CanCommitOffsets].commitAsync(offsetRanges)  
 //提交到MySQL  
 OffsetUtil.saveOffsetRanges("sparkdemo",offsetRanges)  
 println("当前批次的数据已消费并手动提交到MySQL")  
 }  
 })  
  
 //TODO 3.输出结果  
  
 //TODO 4.启动并等待结束  
 ssc.start()  
 ssc.awaitTermination() //注意:流式应用程序启动之后需要一直运行等待手动停止/等待数据到来  
  
 //TODO 5.关闭资源  
 ssc.stop(stopSparkContext = true, stopGracefully = true) //优雅关闭  
 }  
 /\*  
 手动维护offset的工具类  
 首先在MySQL创建如下表  
 CREATE TABLE `t\_offset` (  
 `topic` varchar(255) NOT NULL,  
 `partition` int(11) NOT NULL,  
 `groupid` varchar(255) NOT NULL,  
 `offset` bigint(20) DEFAULT NULL,  
 PRIMARY KEY (`topic`,`partition`,`groupid`)  
 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
 \*/  
 object OffsetUtil {  
 //1.将偏移量保存到数据库  
 def saveOffsetRanges(groupid: String, offsetRange: Array[OffsetRange]) = {  
 val connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/bigdata?characterEncoding=UTF-8", "root", "root")  
 //replace into表示之前有就替换,没有就插入  
 val ps = connection.prepareStatement("replace into t\_offset (`topic`, `partition`, `groupid`, `offset`) values(?,?,?,?)")  
 for (o <- offsetRange) {  
 ps.setString(1, o.topic)  
 ps.setInt(2, o.partition)  
 ps.setString(3, groupid)  
 ps.setLong(4, o.untilOffset)  
 ps.executeUpdate()  
 }  
 ps.close()  
 connection.close()  
 }  
  
 //2.从数据库读取偏移量Map(主题分区,offset)  
 def getOffsetMap(groupid: String, topic: String) = {  
 val connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/bigdata?characterEncoding=UTF-8", "root", "root")  
 val ps = connection.prepareStatement("select \* from t\_offset where groupid=? and topic=?")  
 ps.setString(1, groupid)  
 ps.setString(2, topic)  
 val rs: ResultSet = ps.executeQuery()  
 //Map(主题分区,offset)  
 val offsetMap: mutable.Map[TopicPartition, Long] = mutable.Map[TopicPartition, Long]()  
 while (rs.next()) {  
 offsetMap += new TopicPartition(rs.getString("topic"), rs.getInt("partition")) -> rs.getLong("offset")  
 }  
 rs.close()  
 ps.close()  
 connection.close()  
 offsetMap  
 }  
 }  
  
}  
  
//测试:  
//1.准备kafka  
// /export/server/kafka/bin/kafka-topics.sh --list --zookeeper node1:2181  
// /export/server/kafka/bin/kafka-topics.sh --create --zookeeper node1:2181 --replication-factor 1 --partitions 3 --topic spark\_kafka  
// /export/server/kafka/bin/kafka-console-producer.sh --broker-list node1:9092 --topic spark\_kafka  
//2.启动程序  
//3.发送数据  
//4.观察结果

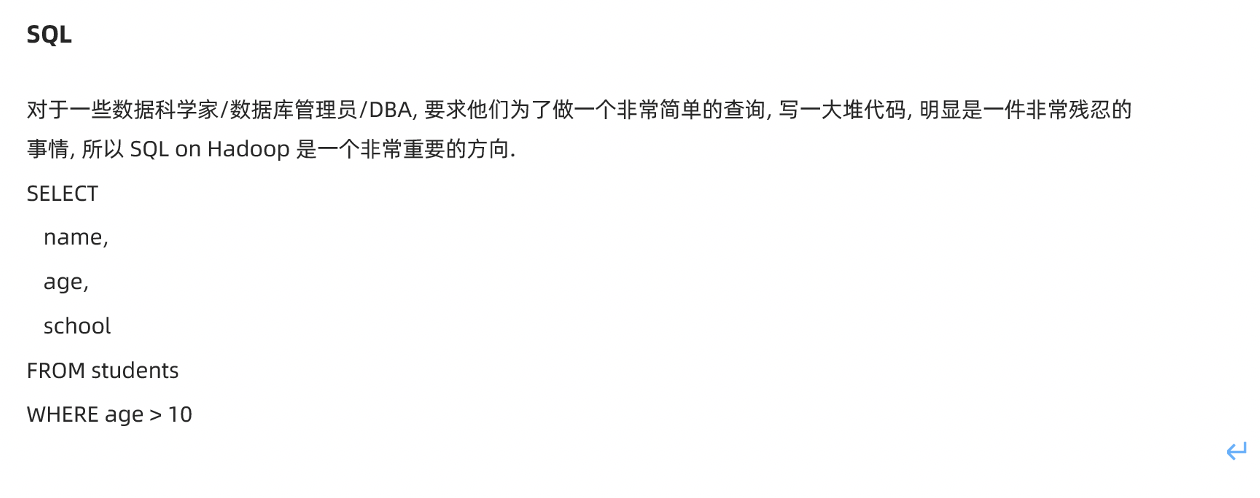
# SparkSQL概述

## 数据分析方式



缺点: 有一定的学习成本/入门门槛

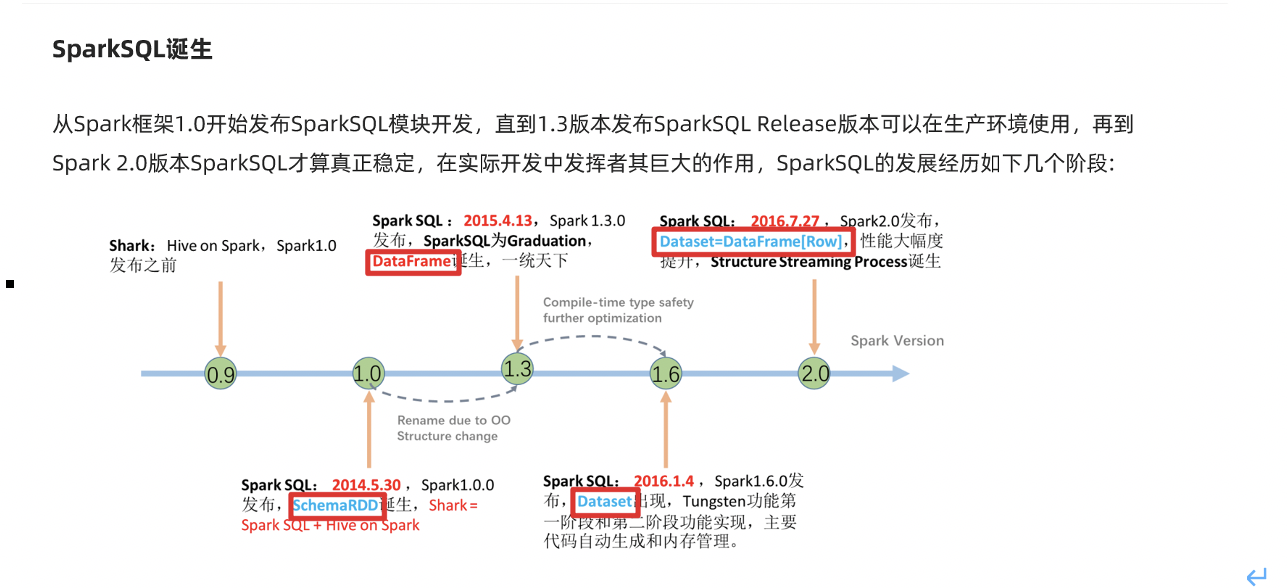
优点: 灵活!可以使用底层的API完成很复杂的业务



优点:入门门槛低,只要会英文单词/简单语法规则就可以写

缺点:只能做一些简单的业务,复杂业务实现起来较困难

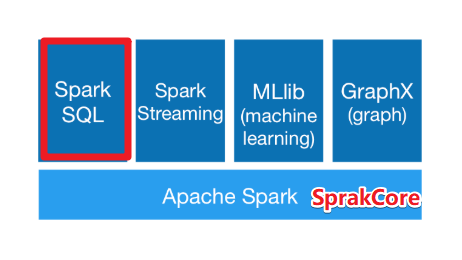
## SparkSQL发展历史



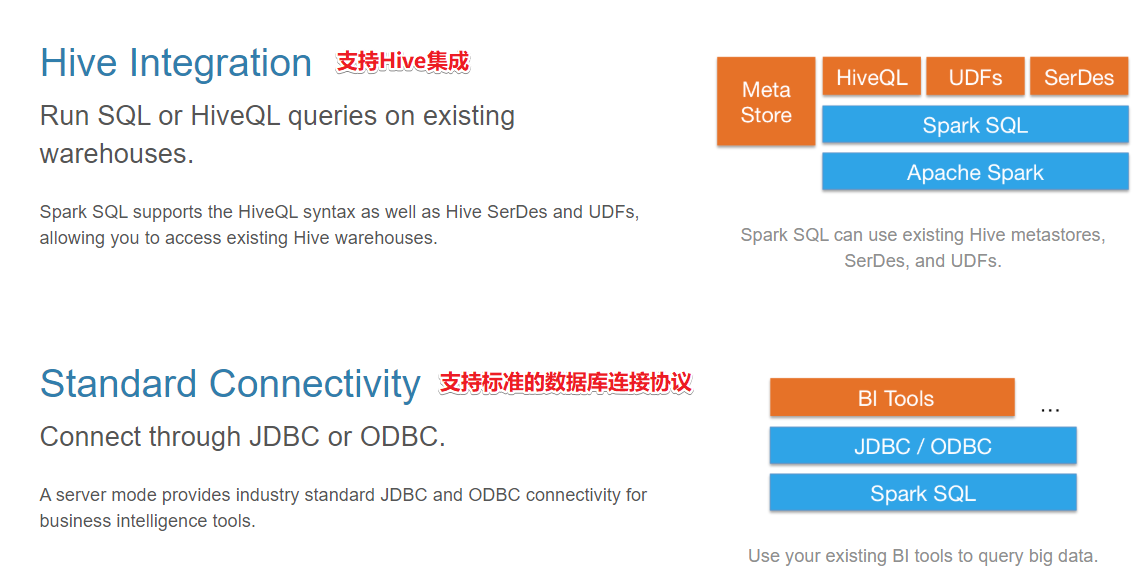


## SparkSQL官方介绍









说明:

* 结构化数据--支持
* 有固定的结构和约束Schema(字段名称/类型)
* 
* 半结构化数据--支持较为严格的半结构化数据
* 有不是固定的结构和约束
* [  
  {  
   "name": "jack",  
   "tel": "1388888888",  
  },  
  {  
   "name": "jack",  
   "tel": 13888888888,  
   "age":18  
  },  
  {  
   "name": "jack",  
   "tel": "1388888888",  
   "age": "18"  
  }  
  ]
* 非结构数据--需要处理之后变为结构化/半结构化才支持

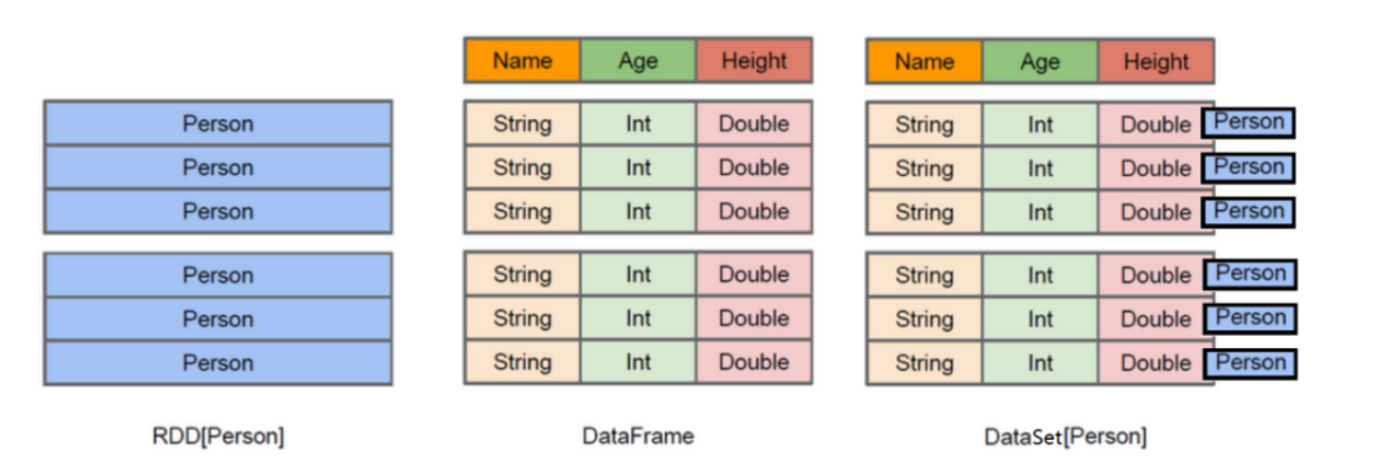
如视频/图片/音频...

# SparkSQL数据抽象

SparkCore的数据抽象:RDD

SparkStreaming的数据抽象:DStream,底层是RDD

SparkSQL的数据抽象:DataFrame和DataSet,底层是RDD



## DataFrame

DataFrame = RDD - 泛型 + Schema约束(指定了字段名和类型) + SQL操作 + 优化

DataFrame 就是在RDD的基础之上做了进一步的封装,支持SQL操作!

DataFrame 就是一个分布式表!

## DataSet

DataSet = DataFrame + 泛型

DataSet = RDD + Schema约束(指定了字段名和类型) + SQL操作 + 优化

DataSet 就是在RDD的基础之上做了进一步的封装,支持SQL操作!

DataSet 就是一个分布式表!

# SparkSQL实战

## 实战1-加载数据成为分布式表

package com.as.sql  
  
import org.apache.spark.SparkContext  
import org.apache.spark.sql.{DataFrame, Dataset, SparkSession}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示SparkSQL初体验  
 \*/  
object Demo01 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val spark: SparkSession = SparkSession.builder().appName("sparksql").master("local[\*]").getOrCreate()  
 val sc: SparkContext = spark.sparkContext  
 sc.setLogLevel("WARN")  
  
 //TODO 1.加载数据  
 val df1: DataFrame = spark.read.text("data/input/text")  
 val df2: DataFrame = spark.read.json("data/input/json")  
 val df3: DataFrame = spark.read.csv("data/input/csv")  
  
 //TODO 2.处理数据  
  
 //TODO 3.输出结果  
 df1.printSchema()  
 df2.printSchema()  
 df3.printSchema()  
 df1.show()  
 df2.show()  
 df3.show()  
  
 //TODO 4.关闭资源  
 spark.stop()  
 }  
}

## 案例2-将RDD转为DataFrame

### 使用样例类

package com.as.sql  
  
import org.apache.spark  
import org.apache.spark.SparkContext  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.{DataFrame, SparkSession}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示SparkSQL-RDD2DataFrame  
 \*/  
object Demo02\_RDD2DataFrame1 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val spark: SparkSession = SparkSession.builder().appName("sparksql").master("local[\*]").getOrCreate()  
 val sc: SparkContext = spark.sparkContext  
 sc.setLogLevel("WARN")  
  
 //TODO 1.加载数据  
 val lines: RDD[String] = sc.textFile("data/input/person.txt")  
  
 //TODO 2.处理数据  
 val personRDD: RDD[Person] = lines.map(line => {  
 val arr: Array[String] = line.split(" ")  
 Person(arr(0).toInt, arr(1), arr(2).toInt)  
 })  
  
 //RDD-->DF  
 import spark.implicits.\_  
 val personDF: DataFrame = personRDD.toDF()  
  
 //TODO 3.输出结果  
 personDF.printSchema()  
 personDF.show()  
  
 //TODO 4.关闭资源  
 spark.stop()  
 }  
 case class Person(id:Int,name:String,age:Int)  
}

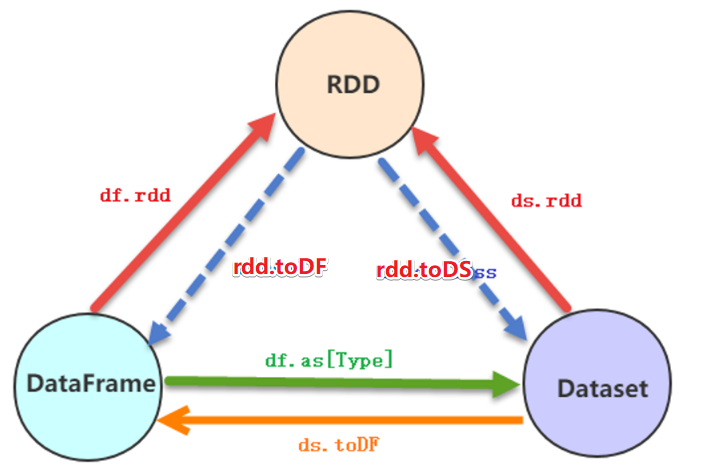
### 指定类型+列名

package com.as.sql  
  
import com.as.sql.Demo02\_RDD2DataFrame1.Person  
import org.apache.spark.SparkContext  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.{DataFrame, SparkSession}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示SparkSQL-RDD2DataFrame-指定类型和列名  
 \*/  
object Demo02\_RDD2DataFrame2 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val spark: SparkSession = SparkSession.builder().appName("sparksql").master("local[\*]").getOrCreate()  
 val sc: SparkContext = spark.sparkContext  
 sc.setLogLevel("WARN")  
  
 //TODO 1.加载数据  
 val lines: RDD[String] = sc.textFile("data/input/person.txt")  
  
 //TODO 2.处理数据  
 val tupleRDD: RDD[(Int, String, Int)] = lines.map(line => {  
 val arr: Array[String] = line.split(" ")  
 (arr(0).toInt, arr(1), arr(2).toInt)  
 })  
  
 //RDD-->DF  
 import spark.implicits.\_  
 val personDF: DataFrame = tupleRDD.toDF("id","name","age")  
  
 //TODO 3.输出结果  
 personDF.printSchema()  
 personDF.show()  
  
 //TODO 4.关闭资源  
 spark.stop()  
 }  
  
}

### 自定义Schema

package com.as.sql  
  
import org.apache.spark.SparkContext  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}  
import org.apache.spark.sql.{DataFrame, Row, SparkSession}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示SparkSQL-RDD2DataFrame-自定义Schema  
 \*/  
object Demo02\_RDD2DataFrame3 {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val spark: SparkSession = SparkSession.builder().appName("sparksql").master("local[\*]").getOrCreate()  
 val sc: SparkContext = spark.sparkContext  
 sc.setLogLevel("WARN")  
  
 //TODO 1.加载数据  
 val lines: RDD[String] = sc.textFile("data/input/person.txt")  
  
 //TODO 2.处理数据  
 val rowRDD: RDD[Row] = lines.map(line => {  
 val arr: Array[String] = line.split(" ")  
 Row(arr(0).toInt, arr(1), arr(2).toInt)  
 })  
  
 //RDD-->DF  
 import spark.implicits.\_  
 /\*val schema: StructType = StructType(  
 StructField("id", IntegerType, false) ::  
 StructField("name", StringType, false) ::  
 StructField("age", IntegerType, false) :: Nil)\*/  
 val schema: StructType = StructType(List(  
 StructField("id", IntegerType, false),  
 StructField("name", StringType, false),  
 StructField("age", IntegerType, false)  
 ))  
  
 val personDF: DataFrame = spark.createDataFrame(rowRDD, schema)  
  
 //TODO 3.输出结果  
 personDF.printSchema()  
 personDF.show()  
  
 //TODO 4.关闭资源  
 spark.stop()  
 }  
  
}

## 案例3-RDD-DF-DS相互转换



package com.as.sql  
  
import org.apache.spark.SparkContext  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.{DataFrame, Dataset, Row, SparkSession}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示SparkSQL-RDD\_DF\_DS相互转换  
 \*/  
object Demo03\_RDD\_DF\_DS {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val spark: SparkSession = SparkSession.builder().appName("sparksql").master("local[\*]").getOrCreate()  
 val sc: SparkContext = spark.sparkContext  
 sc.setLogLevel("WARN")  
  
 //TODO 1.加载数据  
 val lines: RDD[String] = sc.textFile("data/input/person.txt")  
  
 //TODO 2.处理数据  
 val personRDD: RDD[Person] = lines.map(line => {  
 val arr: Array[String] = line.split(" ")  
 Person(arr(0).toInt, arr(1), arr(2).toInt)  
 })  
  
 //转换1:RDD-->DF  
 import spark.implicits.\_  
 val personDF: DataFrame = personRDD.toDF()  
 //转换2:RDD-->DS  
 val personDS: Dataset[Person] = personRDD.toDS()  
 //转换3:DF-->RDD,注意:DF没有泛型,转为RDD时使用的是Row  
 val rdd: RDD[Row] = personDF.rdd  
 //转换4:DS-->RDD  
 val rdd1: RDD[Person] = personDS.rdd  
 //转换5:DF-->DS  
 val ds: Dataset[Person] = personDF.as[Person]  
 //转换6:DS-->DF  
 val df: DataFrame = personDS.toDF()  
  
  
  
 //TODO 3.输出结果  
 personDF.printSchema()  
 personDF.show()  
 personDS.printSchema()  
 personDS.show()  
 rdd.foreach(println)  
 rdd1.foreach(println)  
  
 //TODO 4.关闭资源  
 spark.stop()  
 }  
 case class Person(id:Int,name:String,age:Int)  
}

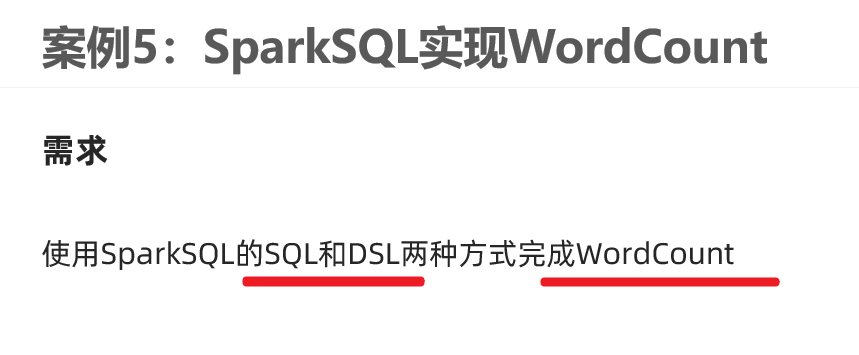
## 案例4-SparkSQL花式查询



需求:针对personDF中的数据使用SQL和DSL两种方式进行各种查询

package com.as.sql  
  
import org.apache.spark.SparkContext  
import org.apache.spark.rdd.RDD  
import org.apache.spark.sql.{DataFrame, SparkSession}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示SparkSQL-SQL和DSL两种方式实现各种查询  
 \*/  
object Demo04\_Query {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val spark: SparkSession = SparkSession.builder().appName("sparksql").master("local[\*]").getOrCreate()  
 val sc: SparkContext = spark.sparkContext  
 sc.setLogLevel("WARN")  
  
 //TODO 1.加载数据  
 val lines: RDD[String] = sc.textFile("data/input/person.txt")  
  
 //TODO 2.处理数据  
 val personRDD: RDD[Person] = lines.map(line => {  
 val arr: Array[String] = line.split(" ")  
 Person(arr(0).toInt, arr(1), arr(2).toInt)  
 })  
  
 //RDD-->DF  
 import spark.implicits.\_  
 val personDF: DataFrame = personRDD.toDF()  
 personDF.printSchema()  
 personDF.show()  
 /\*  
root  
 |-- id: integer (nullable = false)  
 |-- name: string (nullable = true)  
 |-- age: integer (nullable = false)  
  
+---+--------+---+  
| id| name|age|  
+---+--------+---+  
| 1|zhangsan| 20|  
| 2| lisi| 29|  
| 3| wangwu| 25|  
| 4| zhaoliu| 30|  
| 5| tianqi| 35|  
| 6| kobe| 40|  
+---+--------+---+  
 \*/  
  
 //TODO ===========SQL==============  
 //注册表名  
 //personDF.registerTempTable("")//过期的  
 //personDF.createOrReplaceGlobalTempView("")//创建全局的,夸SparkSession也可以用,但是生命周期太长!  
 personDF.createOrReplaceTempView("t\_person")//创建临时的,当前SparkSession也可以用  
  
 //=1.查看name字段的数据  
 spark.sql("select name from t\_person").show()  
 //=2.查看 name 和age字段数据  
 spark.sql("select name,age from t\_person").show()  
 //=3.查询所有的name和age，并将age+1  
 spark.sql("select name,age,age+1 from t\_person").show()  
 //=4.过滤age大于等于25的  
 spark.sql("select name,age from t\_person where age >= 25").show()  
 //=5.统计年龄大于30的人数  
 spark.sql("select count(\*) from t\_person where age > 30").show()  
 //=6.按年龄进行分组并统计相同年龄的人数  
 spark.sql("select age,count(\*) from t\_person group by age").show()  
 //=7.查询姓名=张三的  
 spark.sql("select name from t\_person where name = 'zhangsan'").show()  
  
 //TODO ===========DSL:面向对象的SQL==============  
 //=1.查看name字段的数据  
 //personDF.select(personDF.col("name"))  
 personDF.select("name").show()  
 //=2.查看 name 和age字段数据  
 personDF.select("name","age").show()  
 //=3.查询所有的name和age，并将age+1  
 //personDF.select("name","age","age+1").show()//错误的:cannot resolve '`age+1`' given input columns: [age, id, name];;  
 //注意$是把字符串转为了Column列对象  
 personDF.select($"name",$"age",$"age" + 1).show()  
 //注意'是把列名转为了Column列对象  
 personDF.select('name,'age,'age + 1).show()  
 //=4.过滤age大于等于25的  
 personDF.filter("age >= 25").show()  
 personDF.filter($"age" >= 25).show()  
 personDF.filter('age >= 25).show()  
 //=5.统计年龄大于30的人数  
 val count: Long = personDF.where('age > 30).count() //where底层filter  
 println("年龄大于30的人数为:"+count)  
 //=6.按年龄进行分组并统计相同年龄的人数  
 personDF.groupBy('age).count().show()  
 //=7.查询姓名=张三的  
 personDF.filter("name = 'zhangsan'").show()  
 personDF.filter($"name"==="zhangsan").show()  
 personDF.filter('name ==="zhangsan").show()  
 personDF.filter('name =!="zhangsan").show()  
  
 //TODO 3.输出结果  
 //TODO 4.关闭资源  
 spark.stop()  
 }  
 case class Person(id:Int,name:String,age:Int)  
}

## 案例5-WordCount



package com.as.sql  
  
import org.apache.spark.SparkContext  
import org.apache.spark.sql.{DataFrame, Dataset, SparkSession}  
  
/\*\*  
 \* Author roy  
 \* Desc 演示SparkSQL-SQL和DSL两种方式实现WordCount  
 \*/  
object Demo05\_WordCount {  
 def main(args: Array[String]): Unit = {  
 //TODO 0.准备环境  
 val spark: SparkSession = SparkSession.builder().appName("sparksql").master("local[\*]").getOrCreate()  
 val sc: SparkContext = spark.sparkContext  
 sc.setLogLevel("WARN")  
 import spark.implicits.\_  
  
  
 //TODO 1.加载数据  
 val df: DataFrame = spark.read.text("data/input/words.txt")  
 val ds: Dataset[String] = spark.read.textFile("data/input/words.txt")  
 df.printSchema()  
 df.show()  
 ds.printSchema()  
 ds.show()  
 /\*  
root  
 |-- value: string (nullable = true)  
  
+----------------+  
| value|  
+----------------+  
|hello me you her|  
| hello you her|  
| hello her|  
| hello|  
+----------------+  
 \*/  
 //TODO 2.处理数据  
 //df.flatMap(\_.split(" "))//注意:df没有泛型,不能直接使用split  
 val words: Dataset[String] = ds.flatMap(\_.split(" "))  
 words.printSchema()  
 words.show()  
 /\*  
 root  
 |-- value: string (nullable = true)  
  
+-----+  
|value|  
+-----+  
|hello|  
| me|  
| you|  
| her|  
|hello|  
| you|  
| her|  
|hello|  
| her|  
|hello|  
+-----+  
 \*/  
 //TODO ===SQL===  
 words.createOrReplaceTempView("t\_words")  
 val sql:String =  
 """  
 |select value,count(\*) as counts  
 |from t\_words  
 |group by value  
 |order by counts desc  
 |""".stripMargin  
 spark.sql(sql).show()  
  
 //TODO ===DSL===  
 words.groupBy('value)  
 .count()  
 .orderBy('count.desc)  
 .show()  
  
 //TODO 3.输出结果  
 //TODO 4.关闭资源  
 spark.stop()  
 }  
}