项目技巧及工具

引言

在项目中遇到一些问题，有的需要找以前的资料，有的需要上网查，有的需要自己去思考，于是就之前的问题整理起来，即使问题总结，又是项目技巧，以后可以直接查

一、技巧篇

1. DStream转DateFrame

使用场景：

```
val sparkSession = SparkSession.builder()
    .appName("xxx")
    .getOrCreate()
val res = provinceAdCount.foreachRDD{rdd =>
    //隐式转换
    import sparkSession.implicits._

    //DStream对应的列名，省略里自定义caseclass或SchemaStruct
    val DataFrame = rdd.toDF("date", "province","adId", "num")

    //注册表
    DataFrame.createOrReplaceTempView("table")

    //使用窗口函数row_number()求top3
    val wordCountsDataFrame =
        sparkSession.sql(
        """
            |select date, province, adId, num, rank
            |from(
            | select date, province, adId, num,
            | row_number() over (partition by date, province order by num
desc) as rank
            | from table)
            |where rank < 4
        """.stripMargin)
}
```

结果：

```
//+----------+--------+----+---+----+
//|      date|province|adId|num|rank|
//+----------+--------+----+---+----+
//|2018-12-18|   Hubei|   4|  3|   1|
```

```
//|2018-12-18|   Hubei|   2|  2|   2|
//|2018-12-18|   Hubei|   3|  1|   3|
//|2018-12-18| Jiangsu|   4|  3|   1|
//+----------+--------+----+---+----+
```

2. 两种应用于updateByKey的自定义func

适用场景：当然是用updateByKey的时候了

```
方法一：(key ->(v))一个key对应一个需要累加的value
val fun = (it: (Iterator[((String, String, String, String), Seq[Int],
Option[Int])] )) => {
    it.map(x => {
      ((x._1._1, x._1._2, x._1._3, x._1._4), x._2.sum + x._3.getOrElse(0))
    })
  }
方法一：(key ->(v,v))一个key对应两个需要累加的value
val provinceSum = (it: (Iterator[((String, String), Seq[(Int, Int)],
Option[(Int, Int)])])) => {
    it.map(x => {
      val suc = x._2.unzip._1.sum + x._3.getOrElse((0, 0))._1
      val num = x._2.unzip._2.sum + x._3.getOrElse((0, 0))._2
      ((x._1._1, x._1._2), (suc, num))
    })
  }
```

3. 一气呵成的组内key-value累加排序求topN分离重组

适用场景：拿到一个这样的数据RDD[(String, List[(String, Int),...])]，如何将其进行组内kv累加，然后存入数据库

```
val res = RDD.reduceByKey((list1, list2) => (list1 ::: list2)
  .groupBy(_._1)
  .mapValues(_.foldLeft[Int](0)(_ + _._2))
  .toList)
//组内排序求topN
res.map(row => {
  val sortList = row._2.sortBy(_._2).reverse.take(10)
  (row._1, sortList)
}).flatMap(tp => {
  tp._2.map(id => (tp._1, id._1, id._2))
})
//存入数据库看Util篇
```

4.批量写入数据库

适用场景：写好的util中，存入数据库的那代码里面，是将一个列表批量写入那么遇到的问题是我在执行RDD的foreach时之前定义好的List能存储，怎么做?

- RDD

```
val list = util.ArrayList[object]()
rdd.foreachPartition(partition=>{
    partition.foreach(row=>{
        val test = new Test
        ...
        list.add(test)
    })
    val TestDao = DaoFactory.getTestDao
    TestDao.insertBatch(list)
})
```

- DStream

```
rdd.foreachRDD(partition=>{
    val list = util.ArrayList[object]()
    partition.foreachPartition(row=>{
        val test = new Test
        ...
        list.add(test)
    })
    val TestDao = DaoFactory.getTestDao
    TestDao.insertBatch(list)
})
```

5.breakable用法

适用场景：在scala的while循环中直接不存在像java中那样的break和continue，在写逻辑结构时遇到了，怎么办

- 解决方法：breakable

```
var i = 0
import scala.util.control.Breaks._
breakable {//相当于声明
  while (i < 100) {
    if (i == 20)
      break//类似于java中break
    i += 1
  }
}
println(i)
```

- 解决方法：continue

```
var i = 0
import scala.util.control.Breaks._
while (i < 100) {
    breakable {//相当于声明
      if (i == 10)
        break//类似于java中continue
    }
}
println(i)
```

6. scala中的引用数据类型

适用场景:在一次for循环中用过，就是一下代码中

```
//定义一个总的Map(date,Map(hour,count))
val dateHourCountMap = new
mutable.HashMap[String,mutable.HashMap[String,Long]]()
for(i <- 0 to 2){
    for(j<- 0 to 3){
        val date = i.toString
        val hour = j.toString
        val count = i+j*10
        //获取i天的hourCountMap
        var hourCountMap = dateHourCountMap.get(date).getOrElse(null)
        if(hourCountMap==null){
            //真正定义类型的地方
            hourCountMap = new mutable.HashMap[String,Long]()
            //并存入
            dateHourCountMap.put(date,hourCountMap)
        }
        //此处添加的是dateHourCountMap中的value
        hourCountMap.put(hour,count)
    }
}
//结果
Map(2 -> Map(2 -> 22, 1 -> 12, 0 -> 2, 3 -> 32),
    1 -> Map(2 -> 21, 1 -> 11, 0 -> 1, 3 -> 31),
    0 -> Map(2 -> 20, 1 -> 10, 0 -> 0, 3 -> 30))
```

如果让我写，虽然结果对，但是有点蠢，引用数据类型

```
val dateHourCountMap = new
mutable.HashMap[String,mutable.HashMap[String,Long]]()
for(i <- 0 to 2){
    for(j<- 0 to 3){
        val date = i.toString
        val hour = j.toString
        val count = i+j*10
        //获取i天的hourCountMap
```

```
        var hourCountMap = dateHourCountMap.get(date).getOrElse(null)
        if(hourCountMap==null){
            hourCountMap = new mutable.HashMap[String,Long]()
            hourCountMap.put(hour, count)
            dateHourCountMap.put(date,hourCountMap)
        }else {
            hourCountMap.put(hour,count)
            dateHourCountMap.put(date,hourCountMap)
        }
    }
}
//结果
Map(2 -> Map(2 -> 22, 1 -> 12, 0 -> 2, 3 -> 32),
    1 -> Map(2 -> 21, 1 -> 11, 0 -> 1, 3 -> 31),
    0 -> Map(2 -> 20, 1 -> 10, 0 -> 0, 3 -> 30))
```

```
Map和Set
有引用
var x: mutable.Map[String, Int] = mutable.HashMap[String, Int]()
var y: mutable.Map[String, Int] = x
y.put("xx", 2)
println(x,"hashCode值: ",x.hashCode())
println(y,"hashCode值: ",y.hashCode())
//结果
(Map(xx -> 2),hashCode值: ,-463003677)
(Map(xx -> 2),hashCode值: ,-463003677)


var x = mutable.HashSet[Int]()
var y= x
y.add(1)
println(x,"hashCode值: ",x.hashCode())
println(y,"hashCode值: ",y.hashCode())
//结果
(Set(1),hashCode值: ,-1075495872)
(Set(1),hashCode值: ,-1075495872)
```

7. sql中的if、case等操作

适用场景：sparkSql用于查询

- Case：两种格式，简单case函数和case搜索函数

```
--简单函数
case sex where '1' then '男' where '2' the '女' else 'other' end
--搜索函数
case where sex = '1' the '男' where sex = '2' the '女' else 'other' end
--注意：使用判断为null时，使用is null
```

- IF：在mysql中if()函数的用法类似于java中的三目表达式，其用处也比较多

  语法：IF(expr1,expr2,expr3)，如果expr1的值为true，则返回expr2的值，如果expr1的值为false, 则返回expr3的值。

```
select name,if(sex=0,'女','男') as sex from student;
```

- IFNULL:

  语法：IFNULL(expr1,expr2) 如果expr1不为NULL,则返回值是expr1；否则返回expr2。
- IF ELSE: 不常用

8. 自定义filter过滤函数的注意点

适用场景：因为这个return，把代码改了好几遍

```
RDD.filter(sex=>{
    val flag = sex._1
    if(flag.equals("男"))
        false//没有return
    else
        true
})
//或
RDD.filter(sex=>funx(sex))
def funx(sex){
    val flag = sex._1
    if(flag.equals("男"))
        false
    return true
}
```

9.自定义累加器

适用场景：要对RDD中所以元素的某些值进行累加求和，并且原始的累加器不能满足条件时，就使用自定义累加器

```
class SessionAccumulator extends AccumulatorV2[String,String]{

  var result = (0,0,0)

  override def isZero: Boolean = {
    true
  }

  override def copy(): AccumulatorV2[String, String] = {
    val copyAccumulator = new SessionAccumulator()
    copyAccumulator.result = this.result
    copyAccumulator
  }
```

```
  /**
    * 重置（初始化）
    */
  override def reset(): Unit = {
    this.result = (0,0,0)
  }

  /**
    * task调用，实现task在executor上对累加器进行累加
    * @param v
    */
  override def add(v: String): Unit = {
    val v1 = result
    val v2 = v

    if (StringUtils.isNotEmpty(v1) && StringUtils.isNotEmpty(v2)) {
      // 提取v1中跟v2对应的字段，对这个字段进行累加
      (v1.1+v2.1,...)
    }
    result
  }

  /**
    * 多个分区进行合并
    */
  override def merge(other: AccumulatorV2[String, String]): Unit = other match
{
    case map:SessionAccumulator =>
    {
      val res = other.value
      var v1 = res
      var v2 = result
      result = (v1.1+v2.1,...)

    }
    case _ =>
      throw  new UnsupportedOperationException(s"SessionAccumulator error")
  }

  override def value: String = result
}
```

10. 反射实现DF

适用场景：老版本（好像是2.11之前）字段属性多于22时，如果实现反射生成DataFrame?

case class最多可以放22个参数，否则会出现运行错误，case class和tuple继承了product类，所以我们可以创建product的子类来代替case class。

实现:1、继承product 2、重写productElement方法，重新定义属性字段 override def productElement(n: Int): Any = elemAll(n) 3、重写productArity方法，自定义长度 override def productArity: Int = elemAll.length 4、重写canEqual方法，保证不重复 override def canEqual(that:Any):Boolean=elemAll.contains(that)

11. scalikejdbc

使用场景：应用于scala对数据库的存储，操作简单

pom.xml引入scalikejdbc的依赖

```
<!-- scalikejdbc_2.11 -->
<dependency>
    <groupId>org.scalikejdbc</groupId>
    <artifactId>scalikejdbc_2.11</artifactId>
    <version>2.5.0</version>
</dependency>
```

application.conf配置文件

```
db.default.driver="com.mysql.jdbc.Driver"
db.default.url="jdbc:mysql://localhost/cmccAnalysisSystem?
useUnicode=true&characterEncoding=utf8"
db.default.user="root"
db.default.password="123"
```

用一个小例子来说明一下怎么用：

将一条学生成绩更新到数据库中，如果数据库中存在此学生id就在score基础上累加，否则就插入这条数据

```
//配置，直接写，自动去application.conf找
DBs.setup()
//list是往数据存的一个列表
val iterable = list.iterator()
while (iterable.hasNext) {
    val it = iterable.next()
    //以学生id为条件查询成绩
    val s: Seq[List[String]] = DB.readOnly { implicit session =>
        //sql语句
        SQL("select score from stu where sid = ?")
        //填充?
        .bind(it.sid)
        //以列表的形式返回，也可以是对象，不过需要完整的对象属性，或者有相应的构造方法
        .map(rs => List(rs.string("score"))).list().apply()
    }
    val num = s.size
    if (num == 0) {
        //数据库中没有
        //插入操作
```

```
            DB.autoCommit { implicit session =>
                SQL("insert into stu(sid,score) values(?,?)")
                .bind(it.sid, it.score).update().apply()
            }
        } else {
            //数据库中有
            //更新
            DB.autoCommit { implicit session =>
                SQL("update stu set score=score+? where sid = ?")
                .bind(it.sid ,it.score).update().apply()
            }
        }
    }
```

删除：

```
DB.autoCommit { implicit session =>
    SQL("delete from people where name = ?")
    .bind("张三").update().apply()
}
```

事务：以下语句执行一定会报错，因为1/0会报异常，不使用事务，则第一条会存入数据库，但使用了事务，两条都不会写入数据库，由事务的特性(ACID)决定了，要么全部完成要么全部回滚。

```
DB.localTx { implicit session =>
    SQL("insert into provincenumtop10(date,province,num,ratio)
values(?,?,?,?)")
    .bind("2017-04-12", "山东", 1234, 0.5).update().apply()
    val r = 1 / 0
    SQL("insert into provincenumtop10(date,province,num,ratio)
values(?,?,?,?)")
    .bind("2017-04-12", "山东", 1234, 0.5).update().apply()
}
```

12. replace into的几种用法

二、存取方式篇

1. flume对接kafka

```
#任务：将/root/log/cmcc.json中的数据每秒十条传入kafka

#1.定时的写入flume的监控日志/root/log/cmcc.log
    #编写脚本，1秒读入10条
    for line in `cat /root/log/cmcc.json`
    do
```

```
      `echo $line >> /root/log/cmcc.log`
      sleep 0.1s
    done

#2.编写flume脚本
        agent.sources = s1

        agent.channels = c1

        agent.sinks = k1
        agent.sources.s1.type=exec

        agent.sources.s1.command=tail -F /root/log/cmcc.log

        agent.sources.s1.channels=c1

        agent.channels.c1.type=memory

        agent.channels.c1.capacity=10000

        agent.channels.c1.transactionCapacity=100
        #设置一个kafka接收器
        agent.sinks.k1.type= org.apache.flume.sink.kafka.KafkaSink
        #设置kafka的broker地址和端口号(所有的)
        agent.sinks.k1.brokerList=hadoop01:9092,hadoop02:9092,hadoop03:9092
        #设置kafka的topic
        agent.sinks.k1.topic=cmcc2
        #设置一个序列化方式
        agent.sinks.k1.serializer.class=kafka.serializer.StringEncoder
        #组装
        agent.sinks.k1.channel=c1

#3.启动kafka
    #单机zk启动
        nohup bin/zookeeper-server-start.sh config/zookeeper.properties &
    #启动kafka:
        nohup bin/kafka-server-start.sh  config/server.properties &
    #查看kafka的topic列表:
        bin/kafka-topics.sh --list --zookeeper localhost:2181
    #查看topic中的数据:
        bin/kafka-console-consumer.sh --zookeeper localhost:2181 --from-
beginning --topic cmcc


#4.执行flume脚本
    bin/flume-ng agent -c conf -f conf/flume_kafka.sh -n agent -
Dflume.root.logger=INFO,console
```

2. Hbase存取

```
//application.conf文件
hbase.zookeeper.host="192.168.138.101:2181,192.168.138.102:2181,192.168.138.103:2181"
hbase.table.name="dmp_02"

//获取配置信息
val load = ConfigFactory.load()
val hbaseTableName = load.getString("hbase.table.name")
val configuration = sc.hadoopConfiguration
configuration.set("hbase.zookeeper.quorum",load.getString("hbase.zookeeper.host"))

//建立连接
val hbConn = ConnectionFactory.createConnection(configuration)
//创建Adain对象
val hbadmin = hbConn.getAdmin
if (!hbadmin.tableExists(TableName.valueOf(hbaseTableName))) {
    println("HBASE Table Name Create")
    val tableDescriptor = new
HTableDescriptor(TableName.valueOf(hbaseTableName))
    //创建列簇
    val columnDescriptor = new HColumnDescriptor("tags")
    //将列簇加入表中
    tableDescriptor.addFamily(columnDescriptor)
    hbadmin.crea    teTable(tableDescriptor)
    hbadmin.close()
    hbConn.close()
}
//创建一个jobconf任务
val jobConf = new JobConf(configuration)
//指定key的输出类型
jobConf.setOutputFormat(classOf[TableOutputFormat])
//指定输出到那个表
jobConf.set(TableOutputFormat.OUTPUT_TABLE, hbaseTableName)
//RDD中的数据往Hbase中储存
RDD(val put = new Put(Bytes.toBytes(userid))
    put.addImmutable(
        Bytes.toBytes("tags"), Bytes.toBytes(s"$day"),Bytes.toBytes(tags))
    (new ImmutableBytesWritable(), put)
   ).saveAsHadoopDataset(jobConf)
```

3.生产和消费kafka(0.8版本和0.10)

- 模拟生产者往kafka中写数据

```
package com.gowhere.cmccAnalysisSystem.producter

import java.util
```

```
import com.gowhere.cmccAnalysisSystem.util.GetPropKey
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig,
ProducerRecord}
import scala.io.Source

object CmccProducter {
  def main(args: Array[String]): Unit = {

    val fileName = "F:/bigdata/cmcc/cmcc.json"

    val fileData = Source.fromFile(fileName)

    val (brokers, topic) = (GetPropKey.brokers, GetPropKey.topic)
    // Zookeeper connection properties
    val props = new util.HashMap[String, Object]()
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, brokers)
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
      "org.apache.kafka.common.serialization.StringSerializer")

    val producer = new KafkaProducer[String, String](props)

    for(line <- fileData.getLines)
    {
      val message = line
      producer.send(new ProducerRecord[String, String](topic, message))
      println(line)
      Thread.sleep(50)
    }
    fileData.close()
  }
}
```

- 从kafka中消费，并用zookeeper维护行偏移量（kafka-0.8）

  我写到博客里了，⬇⬇⬇⬇⬇

  https://blog.csdn.net/jklcl/article/details/85217660
- 使用scallikeJdbc维护offset（单机kafka-0.10）

```
val conf = new SparkConf().setAppName("xxx").setMaster("local[2]")
val sc = new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(5))

//配置信息
//earliest为从头开始
//latest为从最新的开始
val kafkaParams = Map[String, Object](
    "bootstrap.servers" -> "hadoop01:9092",
```

```
    "key.deserializer" -> classOf[StringDeserializer],
    "value.deserializer" -> classOf[StringDeserializer],
    "group.id" -> "g007",
    "auto.offset.reset" -> "earliest", //latest"
    "enable.auto.commit" -> (true: java.lang.Boolean)
)
//topic列表
val topics = Array("cmcctest")
//自定义的ScalalikeJdbcUtil类
//实现offset的获取
val topicPartitionOffset = ScalalikeJdbcUtil.offsetCheck(topics)

//获取数据
val stream: InputDStream[ConsumerRecord[String, String]] =
KafkaUtils.createDirectStream[String, String](
    ssc, PreferConsistent, Subscribe[String, String](topics, kafkaParams,
topicPartitionOffset))


//维护
stream.foreachRDD { rdd =>
    val offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges
    rdd.foreachPartition { iter =>
        val o = offsetRanges(TaskContext.get.partitionId)
        ScalalikeJdbcUtil.updateOffset(o)
    }
}

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

ScalalikeJdbcUtil类

```
/**
  * 获取offset列表
  * @param topics
  * @return
  */
def offsetCheck(topics: Array[String]): mutable.Map[TopicPartition, Long]
={
    DBs.setup()
    val res = DB.readOnly { implicit session =>
        val offsets = mutable.HashMap[TopicPartition, Long]()
        for(topic <- topics){
            SQL("select partition, offset from offset_check where
topic=?")
            .bind(topic)
            .map(rs =>{
                val partition = rs.string("partition").toInt
                val offset = rs.string("offset").toLong
```

```
                offsets.put(new TopicPartition(topic, partition), offset)
            }).list().apply()
        }
        offsets
    }
    res
}

/**
  * 维护offset
  * @param it
  */
def updateOffset(it: OffsetRange): Unit ={
    DBs.setup()
    DB.autoCommit { implicit session =>
        SQL("replace into offset_check set topic=?, partition=?,
offset=?")
        .bind(it.topic, it.partition, it.untilOffset).update().apply()
    }
}
```

4. DF对mysql存值

```
//application.conf文件
jdbc.url="jdbc:mysql://loaclhost:3306/dmp?characterEncoding=utf-8"
jdbc.tableName="tableName"
jdbc.user="root"
jdbc.password="123"

//存储到mysql中
val load = ConfigFactory.load()
val prop = new Properties()
props.setProperty("user", load.getString("jdbc.user"))
props.setProperty("password", load.getString("jdbc.password"))

res.write.mode(SaveMode.Append)
    .jdbc(load.getString("jdbc.url"), load.getString("jdbc.tableName"),props)
//追加写
SaveMode.Append
//覆盖写
SaveMode.OverWrite
```

5.Redis存取

```
jedis连接池
object JedisConnectionPool{
    //获取到配合对象
    val config = new JedisPoolConfig()
    //设置最大数
```

```
    config.setMaxTotal(20)
    //设置最大空闲连接数
    config.setMaxIdle(10)
    val pool = new JedisPool(config, "192.168.xxx.xxx", 6379, 1000)

    def getConnection():Jedis={
        pool.getResource
    }
}

//创建jedis对象，建立连接
val jedis = JedisConnectionPool.getConnection()
//存取kv对
jedis.set(key, value)
//获取kv对
jedis.get(key)
//关闭
jedis.close()
```

三、Util篇

1. application.conf

常用的：

```
jdbc.url="jdbc:mysql://localhost:3306/cmccAnalysisSystem?
useUnicode=true&characterEncoding=utf8"
jdbc.user="root"
jdbc.password="123"

kafka.topic="cmcckafka"
kafka.brokers="hadoop01:9092,hadoop02:9092,hadoop03:9092"
zookeeper.brokers="hadoop01:2181,hadoop02:2181,hadoop03:2181"

province={
100=北京,
200=广东,
210=上海,
220=天津)
```

2. GetConfValue

对上面的获取

```
val load = ConfigFactory.load()
val topic = load.getString("kafka.topic")
val brokers = load.getString("kafka.brokers")
val jdbcurl = load.getString("jdbc.url")
val jdbcuser = load.getString("jdbc.user")
val jdbcpassword = load.getString("jdbc.password")
val zkQuorum = load.getString("zookeeper.brokers")
//map形式的kv对
def getProvince() ={
   import scala.collection.JavaConversions._
   val provinceMap = load.getObject("province").unwrapped().toMap
   provinceMap
}
```

3. pom.xml整合

```
<!-- scalikejdbc_2.11 -->
<dependency>
    <groupId>org.scalikejdbc</groupId>
    <artifactId>scalikejdbc_2.11</artifactId>
    <version>2.5.0</version>
</dependency>
```

4.hadoop各种组件的命令

- Redis

> *后台启动：redis-server redis.conf*
> *关闭：redis-cli shutdown*
> *进入shell界面：redis-cli -h 127.0.0.1 -p 6379*
> *清空：1.select 0 2.DBSIZE 3.flushall*

- zookeeper

> *启动客户端：./zkCli.sh -server localhost:2181*
> *帮助命令：help*
> *获取节点信息：get /*
> *查看节点的状态信息：stat /*
> *设置节点的数据：set /usergrid hellUsergrid*
> *获取路径下的节点信息：ls /consumers 或 ls2 /consumers*

> *删除节点：delete /（跑路）*
>
> *退出客户端：quit*
>
> *节点信息如下*
>
> *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*
>
> *https://www.cnblogs.com/senlinyang/p/7833669.html*

- kafka

> *#单机zk启动 nohup bin/zookeeper-server-start.sh config/zookeeper.properties & #启动kafka： nohup bin/kafka-server-start.sh config/server.properties & #查看kafka的topic列表： bin/kafka-topics.sh --list --zookeeper localhost:2181 #查看topic中的数据： bin/kafka-console-consumer.sh --zookeeper localhost:2181 --from-beginning --topic cmcc #创建topic bin/kafka-topics.sh --create --zookeeper localhost:2191 --replication-factor 1 --partitions 1 --topic test11*
> *#写入topic中的数据： bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test*

未完待续，连载中.....