## Spark操作Mysql

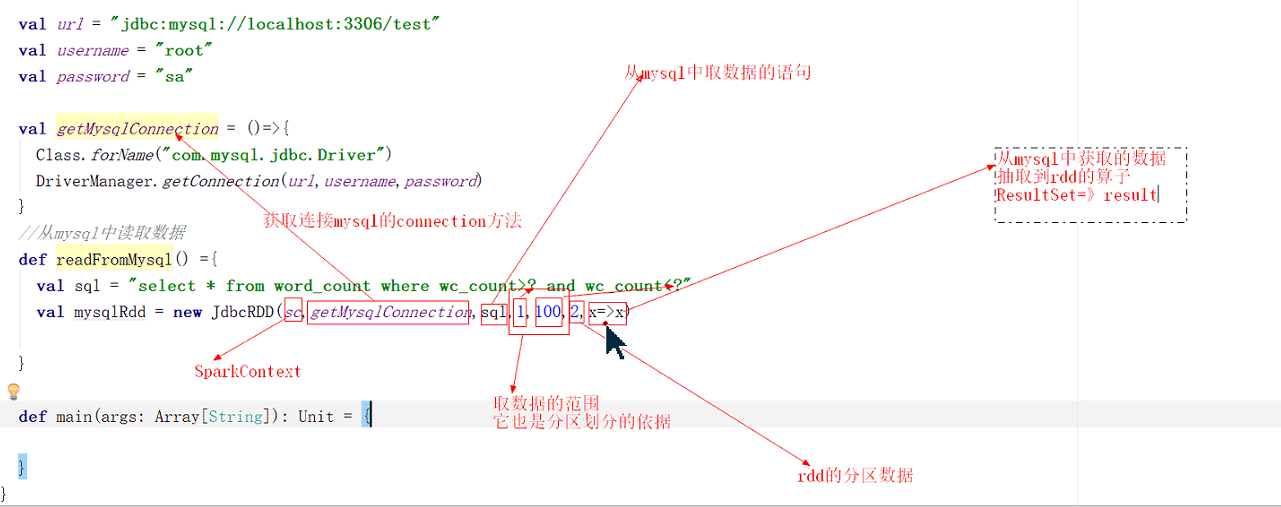
JdbcRdd通过实例化jdbcRDD对象来完成对关系型数据库中的数据作为spark数据源的一种方式

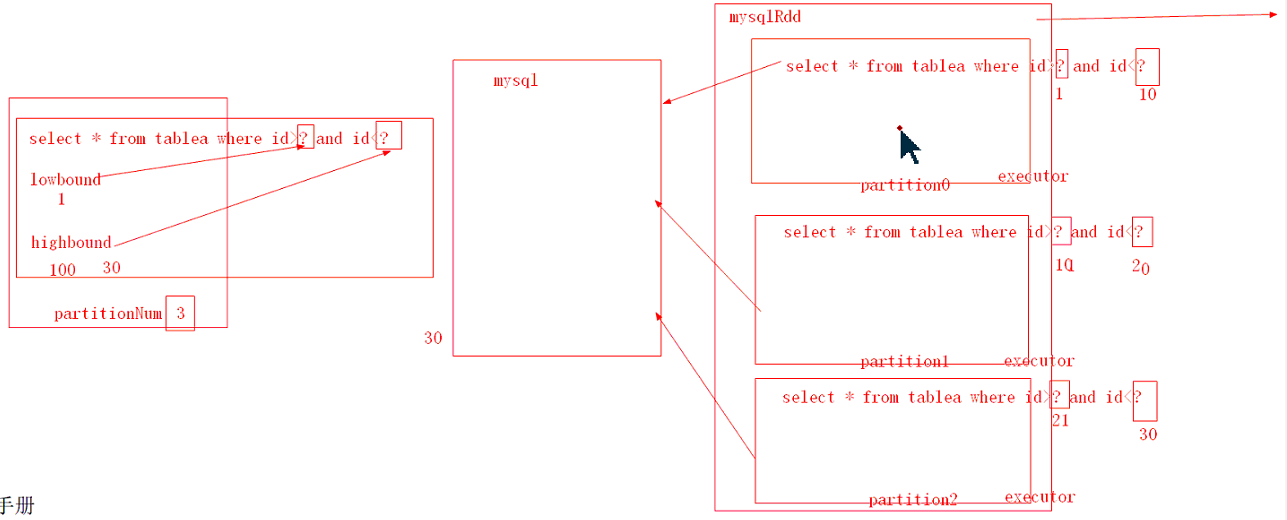
实例化的过程需要:SparkContext、关系型数据库的连接、sql(必须包含2个问号，lowbound、highbound分别和sql中的2个问号对应)、分区数、结果转换函数：r=>r 从关系型数据库里获取的数据是resultSet，哪里读取进来的rdd中希望它是什么类型就用这个函数来完成转换

### Spark读mysql数据库

|  |
| --- |
| **object** MysqlTest {  **val** *conf* = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"MysqlTest"**)  **val** *sc* = SparkContext.getOrCreate(conf)   **val** *url* = **"jdbc:mysql://localhost:3306/xs"  val** *username* = **"root"  val** *password* = **"root"   def** getMysqlConnection = () => {  Class.*forName*(**"com.mysql.jdbc.Driver"**)  DriverManager.*getConnection*(*url*,*username*,*password*)  }   *// 从mysql中读取数据* **def** readFromMysql() = {  **val** sql =**"select \* from t\_user where id > ? and id < ?"** *// sc ：SparkContext,getMysqlConnection:connection,sql:sql语句,1:第一个?,100第二个?; 2:分区数;最后一个参数:从，mysql中获取的数据抽取到rdd的算子，resultSet* **val** mysqlRdd = **new** JdbcRDD(*sc*,*getMysqlConnection*,sql,0,100,2,x => x)  mysqlRdd.foreach(x => {  *println*(**s"$**{x.getInt(**"id"**)}**, $**{x.getString(**"username"**)}**, $**{x.getString(**"password"**)}**"**)  })  }   **def** main(args: Array[String]): Unit = {  *readFromMysql*()  } } |

### jdbcRdd参数的意义





### spark写入数据到mysql

|  |
| --- |
| **val** *conf* = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"MysqlTest"**)  **val** *sc* = SparkContext.getOrCreate(conf)   **val** *url* = **"jdbc:mysql://localhost:3306/xs"  val** *username* = **"root"  val** *password* = **"root"   def** getMysqlConnection = () => {  Class.*forName*(**"com.mysql.jdbc.Driver"**)  DriverManager.*getConnection*(*url*,*username*,*password*)  }  *// 写入mysql* **def** writeToMysql() = {  *// 统计每个用户在每个ip的行为次数，结果存放在mysql中user\_ip\_times* **val** rdd = *sc*.textFile(**"/bd14/user-logs-large.txt"**)  **val** result = rdd.map(x => {  **val** info = x.split(**"\\t"**)  ((info(0),info(2)),1)  }).reduceByKey(\_ + \_)  *// result.foreach(println)  // 把结果保存到mysql中,使用foreachPartition，减少mysql的连接次数，提高性能* result.foreachPartition(x => {  *// 批量写入* **val** connection = *getMysqlConnection*()  **val** sql = **"insert into user\_ip\_times (user\_name,ip,times)values (?,?,?)"  val** prepareStatement = connection.prepareStatement(sql)  x.foreach(record => {  *// 写入到mysql* prepareStatement.setString(1,record.\_1.\_1)  prepareStatement.setString(2,record.\_1.\_2)  prepareStatement.setInt(3,record.\_2)  prepareStatement.addBatch()  })  prepareStatement.executeBatch()  })  } |

## Spark操作Hbase

### Spark写数据到hbase

|  |
| --- |
| **val** *conf* = **new** SparkConf().setMaster(**"local[\*]"**).setAppName(**"HbaseTest"**)  **val** *sc* = SparkContext.*getOrCreate*(*conf*)   *// 将每个用户在每个ip上的行为次数，保存在hbase中，user\_ip\_times,rowkey 用户名，列族i  // 列成员ip，列:times* **def** writeToHbase() = {  **val** configuration = HBaseConfiguration.*create*(*sc*.hadoopConfiguration)  *// 设置保存到hbase的配置信息* configuration.set(TableOutputFormat.*OUTPUT\_TABLE*,**"user\_ip\_times"**)  **val** job = Job.*getInstance*(configuration)  job.setOutputKeyClass(*classOf*[ImmutableBytesWritable])  job.setOutputValueClass(*classOf*[Put])  job.setOutputFormatClass(*classOf*[TableOutputFormat[ImmutableBytesWritable]])  *// setOutputPath，因版本问题，需要设置，没有什么实际意义* FileOutputFormat.*setOutputPath*(job,**new** Path(**"/hb"**))   *// 计算* **val** rdd = *sc*.textFile(**"/bd14/user-logs-large.txt"**)  **val** result = rdd.map(x => {  **val** info = x.split(**"\\t"**)  ((info(0),info(2)),1)  }).reduceByKey(\_ + \_) *// result.foreach(println)   // 转换成能往hbase中保存的数据* **val** hbaseResult = result.map(x => {  **val** put = **new** Put(Bytes.*toBytes*(x.\_1.\_1))  put.addColumn(Bytes.*toBytes*(**"i"**),Bytes.*toBytes*(x.\_1.\_2),Bytes.*toBytes*(x.\_2.toString))  (**new** ImmutableBytesWritable,put)  })  *// 写入hbase中* hbaseResult.saveAsNewAPIHadoopDataset(job.getConfiguration)  } |

### Spark读取Hbase

|  |
| --- |
| *// spark从hbase中读取数据加载成RDD* **def** readFromHbase() = {  **val** configuration = HBaseConfiguration.*create*(*sc*.hadoopConfiguration)  *// 在configuration上配置读取hbase所需的参数* configuration.set(TableInputFormat.*INPUT\_TABLE*, **"user\_ip\_times"**)  *// val job = Job.getInstance(configuration)  // job.setInputFormatClass()* **val** hbaseRdd = *sc*.newAPIHadoopRDD(configuration, *classOf*[TableInputFormat], *classOf*[ImmutableBytesWritable], *classOf*[Result])  **val** resultRdd = hbaseRdd.map(x => {  **val** result = x.\_2  **val** user = Bytes.*toString*(result.getRow)  **val** cellScanner = result.cellScanner()  **val** values = **new** ArrayBuffer[String]()  **while** (cellScanner.advance()) {  **val** cell = cellScanner.current()  **val** ip = Bytes.*toString*(CellUtil.*cloneQualifier*(cell))  **val** times = Bytes.*toString*(CellUtil.*cloneValue*(cell))  values += **s"$**ip **$**times**"** }  (user, values.toList)  })  resultRdd.foreach(*println*) } |

格式化输出结果

|  |
| --- |
| *// 对hbaseRdd进行转换，使其格式和hbase scan的效果相同* **def** readFromHbase1() = {  **val** configuration = HBaseConfiguration.*create*(*sc*.hadoopConfiguration)  *// 在configuration上配置读取hbase所需的参数* configuration.set(TableInputFormat.*INPUT\_TABLE*, **"user\_ip\_times"**)  *// val job = Job.getInstance(configuration)  // job.setInputFormatClass()* **val** hbaseRdd = *sc*.newAPIHadoopRDD(configuration, *classOf*[TableInputFormat], *classOf*[ImmutableBytesWritable], *classOf*[Result])  **val** resultRdd = hbaseRdd.flatMap(x => {  **val** result = x.\_2  **val** user = Bytes.*toString*(result.getRow)  **val** cellScanner = result.cellScanner()  **val** values = **new** ArrayBuffer[String]()  **val** list = *List* **while** (cellScanner.advance()) {  **val** cell = cellScanner.current()  **val** ip = Bytes.*toString*(CellUtil.*cloneQualifier*(cell))  **val** times = Bytes.*toString*(CellUtil.*cloneValue*(cell))  **val** ts = cell.getTimestamp  list :: *List*(ip,times)  values += **s"$**{user} **ip:$**{ip} **times:$**{times} **timestamp:$**{ts}**"** }  values  })  resultRdd.foreach(*println*) } |

## Spark程序发布运行模式

Hadoop发布运行使用hadoop jar

发布程序需要使用spark-submit指令

-- Class应用程序主函数的入口

-- Master 发布运行模式

-- jars指定应用程序运行的依赖jar包的位置，一般情况下我们可以在应用程序中使用sc.addJar方法指定hdfs上的jar加载应用程序的依赖jar

### Local

本地模式运行，local[n],local[\*]

1. 将应用程序打包上传，注意在设置sparkConf是不设置setMaster，不设置sc.addJar

|  |
| --- |
| **package** top.xiesem.bd14 **import** org.apache.spark.{SparkConf, SparkContext} **object** WordCount {  **def** main(args: Array[String]): Unit = { **val** conf = **new** SparkConf().setAppName(**"WordCount App"**)*//.setMaster("spark://master:7077")* **val** sc = **new** SparkContext(conf)**val** fileRdd = sc.textFile(**"/readme.md"**)  *// 数据处理开始* **val** wordRDD = fileRdd.flatMap(line => line.split(**"\\s"**))  **val** result = wordRDD.map(x => (x, 1)).reduceByKey(\_ + \_)result.foreach(*println*)  sc.stop()} } |

1. spark-submit --class top.xiesem.bd14.WordCount(全路径) --master local sparkhwbd14.jar(jar包位置)

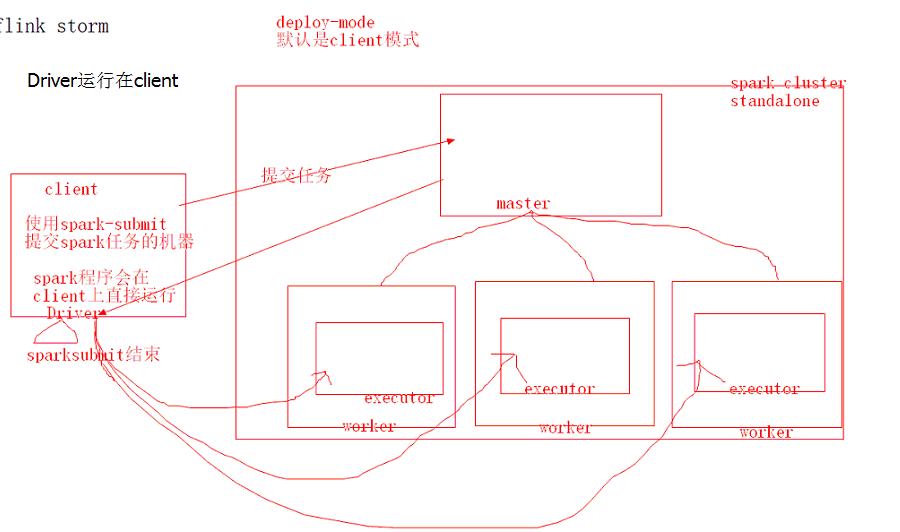
### Standlone

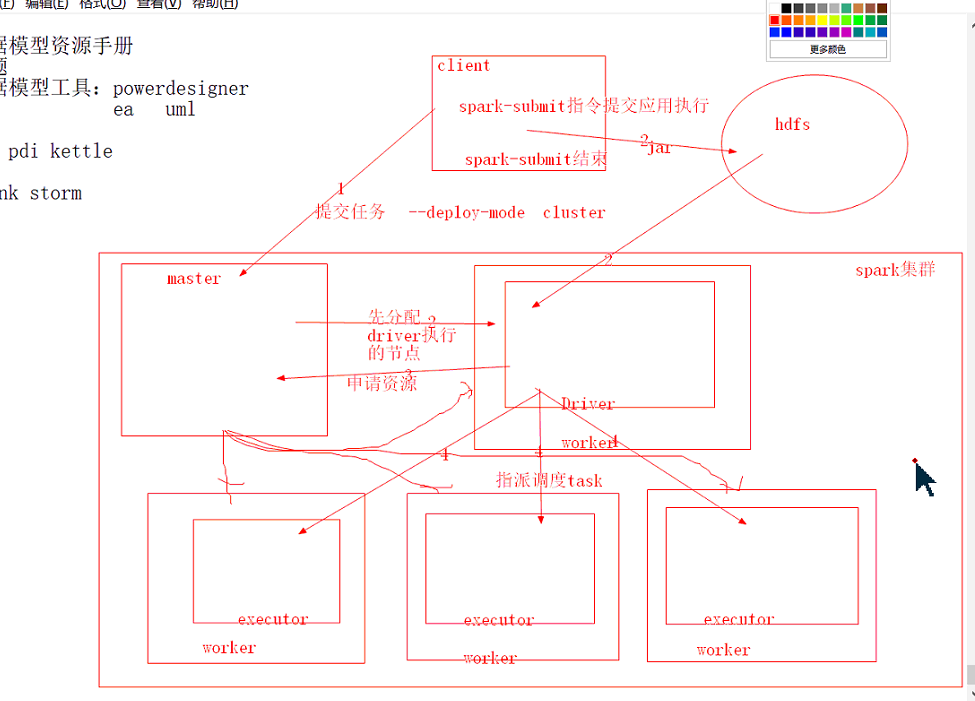
--deploy-model client/cluster

spark-submit --master spark://master:7077 --class top.xiesem.bd14.WordCount sparkhwbd14.jar

Client模式是运行在client上，哪里spark-submit的，哪里就是client

Cluster模式是driver程序运行在集群中的某个工作节点上，客户端哪里只要提交任务





### Yarn

spark-submit --master yarn --deploy-mode cluster --class top.xiesem.bd14.WordCount sparkhwbd14.jar

spark-submit --master yarn --deploy-mode client --class top.xiesem.bd14.WordCount sparkhwbd14.jar

Mesos

Deploy-mode部署模式，处理local之外，yarn、Standlone、mesos都有两种部署模式，Cluster、client

--conf 运行环境参数，比如运行的内存设置，executor的数量

## java编写spark

1.构建maven项目

2.添加spark依赖

3.构建JavaSparkContext

4.使用JavaSparkContext构建JavaRDD

5.对JavaRDD调用transformation和action 操作数据（大量的内部类写法实例化算子）

6.计算结束，停用sparkcontext：sc.stop()

|  |
| --- |
| **package** top.xiesen.bd14;  **import** java.util.Arrays;  **import** java.util.Iterator;  **import** org.apache.spark.SparkConf;  **import** org.apache.spark.api.java.JavaPairRDD;  **import** org.apache.spark.api.java.JavaRDD;  **import** org.apache.spark.api.java.JavaSparkContext;  **import** org.apache.spark.api.java.function.FlatMapFunction;  **import** org.apache.spark.api.java.function.Function2;  **import** org.apache.spark.api.java.function.PairFunction;  **import** scala.Tuple2;  **public** **class** WorldCount {  **public** **static** **void** main(String[] args) {  SparkConf conf = **new** SparkConf().setMaster("local[\*]").setAppName("eclipseWorldCount");  JavaSparkContext sc = **new** JavaSparkContext(conf);  // 构建rdd  JavaRDD<String> jRdd = sc.textFile("/readme.md");  // 把一行转换成单词  JavaRDD<String> woRdd = jRdd.flatMap(**new** FlatMapFunction<String,String>() {  **public** Iterator<String> call(String t) **throws** Exception {  **return** Arrays.*asList*(t.split("\\s")).iterator();  }  });  // 把单词的rdd转换成kv的rdd  JavaPairRDD<String,Integer> javaPairRDD = woRdd.mapToPair(**new** PairFunction<String, String, Integer>() {  @Override  **public** Tuple2<String, Integer> call(String t) **throws** Exception {  **return** **new** Tuple2<String, Integer>(t, 1);  }  });  JavaPairRDD<String, Integer> result = javaPairRDD.reduceByKey(**new** Function2<Integer, Integer, Integer>() {    @Override  **public** Integer call(Integer v1, Integer v2) **throws** Exception {  **return** v1 + v2;  }  });  result.saveAsTextFile("/spark/java-spark-output");  }  } |

## Spark处理数据倾斜问题

