Spark SQL讲义

|  |  |  |
| --- | --- | --- |
|  | 文件标识： |  |
| 当前版本： | 1.0 |
| 作 者： | 马卫花 |
| 完成日期： | 2022-1-22 |

**目 录**

[第1章 SparkSQL 概述 1](#_Toc19394)

[1.1 SparkSQL 是什么 1](#_Toc2584)

[1.2 SparkSQL 特点 1](#_Toc7683)

[1.3 DataFrame 是什么 2](#_Toc18986)

[第2章 灾害性天气预报数据分析 4](#_Toc20463)

[2.1 项目背景 4](#_Toc6694)

[2.2 分析任务 4](#_Toc31326)

[2.2.1 架构设计 4](#_Toc26826)

[2.2.2 设计思路 5](#_Toc19773)

[2.2.3 数据集 6](#_Toc14827)

[2.2.4 统计分析 6](#_Toc27198)

# 第1章 SparkSQL 概述

## 1.1 SparkSQL 是什么





Spark SQL 是 Spark 用于结构化数据(structured data)处理的 Spark 模块。

## 1.2 SparkSQL 特点

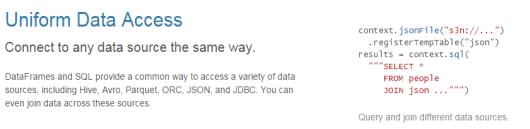
**1.2.1** 易整合

无缝的整合了SQL查询和Spark编程



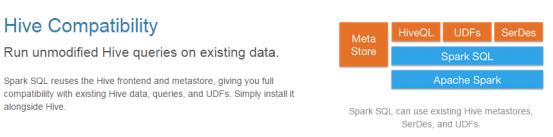
**1.2.2 统一的数据访问**

使用相同的方式连接不同的数据源



**1.2.3 兼容 Hive**

在已有的仓库上直接运行 SQL 或者 HiveQL



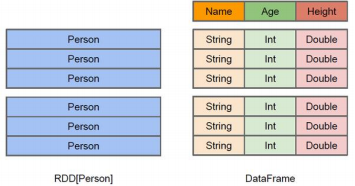
**1.2.4 标准数据连接**

通过 JDBC 或者 ODBC 来连接



## 1.3 DataFrame 是什么

在 Spark DataFrame 是一种以 RDD为基础的分布式数据集，类似于传统数据库中 的二维表格。DataFrame与 RDD的主要区别在于，前者带有 schema元信息，即 DataFrame 所表示的二维表数据集的每一列都带有名称和类型。这使得 Spark SQL 得以洞察更多的结构信息，从而对藏于 DataFrame背后的数据源以及作用于 DataFrame之上的变换进行了针对性的优化，最终达到大幅提升运行时效率的目标。反观 RDD，由于无从得知所存数据元素的具体内部结构，Spark Core 只能在 stage 层面进行简单、通用的流水线优化。



上图直观地体现了 DataFrame 和 RDD 的区别。

左侧的 RDD[Person]虽然以 Person 为类型参数，但 Spark 框架本身不了解 Person 类的内 部结构。而右侧的 DataFrame 却提供了详细的结构信息，使得 Spark SQL 可以清楚地知道该数据集中包含哪些列， 每列的名称和类型各是什么。

重点学习如何使用 Spark SQL 所提供的 DataFrame 和 DataSet 模型进行编程，本次实训主要介绍DataFrame实现数据分析。

Spark Core 中，如果想要执行应用程序， 需要首先构建上下文环境对象 SparkContext ，Spark SQL 其实可以理解为对 Spark Core 的一种封装， 不仅仅在模型上进行了封装， 上下文环境对象也进行了封装。在 Spark SQL 中 SparkSession 是创建 DataFrame 和执行 SQL 的入口， 创建 DataFrame 有三种方式： 通过 Spark 的数据源进行创建； 从一个存在的 RDD 进行转换； 还可以从 Hive

Table 进行查询返回。本章主要介绍从数据源中创建。

* 查看 Spark 支持创建文件的数据源格式：csv、format、jdbc、json、load option、options、orc、 parquet、 schema table、 text 、textFile
* 将数据源中的comments.csv文件上传至HDFS
* 读取csv文件创建DataFrame

|  |
| --- |
| scala> val df = spark.read.format("csv").option("header","true").csv("/music/comments.csv")  df: org.apache.spark.sql.DataFrame = [commentId: string, content: string ... 3 more fields]  scala> df.show  +----------+-------------------------------------+----------+-------------------+----------+  | commentId| content|likedCount| time| userId|  +----------+-------------------------------------+----------+-------------------+----------+  |5345691262| 13倍速进入新时代| 0|2021-06-30 14:26:03|3225402678|  |5345739784| 哈哈哈哈哈哈哈| 0|2021-06-30 14:15:33|1473278275|  |5345750746| 帕梅拉| 0|2021-06-30 14:15:26| 53353985|  |5345616605| 黄老板跟小贝同框| 0|2021-06-30 13:33:32| 98508243| |

* 对 DataFrame 创建一个临时表

scala> df.createOrReplaceTempView("commnets")

* 通过 SQL 语句实现查询全表

scala> val sqlDF = spark.sql("select \* from comments")

* 查看结果

sqlDF.show

* 统计不同月份的评论数量

scala> val sqlDF = spark.sql("select month(time) month ,count(\*) from commnets group by month(time)")

## 1.4 IDEA开发Spark SQL

1）新建Maven项目

在IDEA中新建maven项目，然后添加Spark SQL依赖，关键代码如下所示：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>org.example</groupId>  <artifactId>sparksqlexample</artifactId>  <version>1.0-SNAPSHOT</version>  <name>sparksqlexample</name>  <url>http://www.example.com</url> <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <spark.version>3.0.0</spark.version>  <scala.version>2.12</scala.version>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-core\_2.12</artifactId>  <version>${spark.version}</version>  </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-yarn\_2.12</artifactId>  <version>${spark.version}</version>  </dependency>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-sql\_2.12</artifactId>  <version>${spark.version}</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.20</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>2.10.1</version>  </dependency>  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.10</version>  </dependency>  </dependencies>  </project> |

1. 读取csv文件实现数据分析

读取CSV文件实现数据分析的关键代码如下所示：

|  |
| --- |
| import org.apache.spark.SparkConf  import org.apache.spark.sql.{DataFrame, Dataset, Row, SaveMode, SparkSession}  /\*\*  \* 知识点:  \* SparkSQL:  \* 1)加载数据 (csv,jdbc,json,txt....)  \* spark.read.  \* format(" "): 加载的数据类型 csv,jdbc,json  \* option(".."): //如果读取jdbc数据源，设置jdbc相应的参数driver,url,user,password,dbtable  \* load("") :// 如果是csv,json文件，设定加载文件的路径  \*  \* Spark sql读取csv文件  \* src\\input\\comments.csv文件  \*  \* 虚拟机运行命令:  \* ./bin/spark-submit --class SparkSqlCSVExample --master local sparksqlexample-1.0-SNAPSHOT.jar  \*  \*/  object SparkSqlCSVExample {  def main(args: Array[String]): Unit = {  //1.创建Spark环境配置对象  val conf = new SparkConf().setAppName("SparkSqlCSVExample").setMaster("local")  //2.创建SparkSession对象  val spark:SparkSession = SparkSession.builder().config(conf).getOrCreate()  import spark.implicits.\_  var commentData: DataFrame = spark.read.format("csv")  .option("header", true)  .option("multiLine", true)  .load("\\input\\comments.csv")  //3.注册临时表  commentData.createOrReplaceTempView("tbl\_comment")  //4.查询操作  val sqlresult : DataFrame=  spark.sql("select substring(time,12,2) as commenthour,count(\*) as commenttotal " +  " from tbl\_comment " +  " group by substring(time,12,2) " +  " order by substring(time,12,2) desc")  sqlresult.show()  //5.将分析结果保存到数据表中  // eg:存入weatherdb数据库中  sqlresult.write  .format("jdbc")  .option("url","jdbc:mysql://192.168.198.101:3306/weatherdb")  .option("driver","com.mysql.cj.jdbc.Driver")  .option("user","root")  .option("password","123456")  .option("dbtable","tbl\_comment\_process\_result")  .mode(SaveMode.Append)  .save()  }  } |

3）读取数据库中的数据实现数据分析

读取数据库中的数据实现数据分析的关键代码如下所示：

|  |
| --- |
| def main(args: Array[String]): Unit = {  //1.创建Spark环境配置对象  val conf = new SparkConf().setAppName("SparkSqlWriteDataExample").setMaster("local")  //2.创建SparkSession对象  val spark:SparkSession = SparkSession.builder().config(conf).getOrCreate()  import spark.implicits.\_  var testtableDf : DataFrame =spark.read.format("jdbc")  .option("url","jdbc:mysql://192.168.198.128:3306/weatherdb")  .option("driver","com.mysql.cj.jdbc.Driver")  .option("user","root")  .option("password","Root123")  .option("dbtable","testtable")  .load()  testtableDf.createOrReplaceTempView("testtable")  val result = spark.sql(  """  |select count(\*),name from testtable group by name  |""".stripMargin)  result.show()  } |

**第2章 灾害性天气预报数据分析**

## 2.1 项目背景

天气变化是人们重点研究的自然对象，从古至今记载的气象资料是一个非常庞大的数据库，随着互联网的发展，每时每刻都形成庞大的气象数据资料，及时的、实时的记录、存储、分析计算得出结果就是现代气象每天都要解决的问题。

得益于信息科技的发展，超级计算机大数据中心的成立、5G通信技术的进步以及计算机应用技术的发展，应用大数据技术对这些实时上传的海量数据资料进行采集、传输、存储、计算，进而分析和预测未来天气变成可能。

在本案例中要解决的问题就是要研究如何应用大数据技术通过对记录数据资料获取、清洗、存储、分析、可视化展示智能预测未来天气变化的实际问题。

## 2.2 分析任务

世界各国的天气气温数据源是从2008年至2012年气温，中国的气温数据源分析的是2010年至2012年的气温。本案例分析任务以及可视化显示效果预览如下所示：

### 2.2.1 架构设计

本案例主要由三大部分组成，分别是基于spark进行数据分析、基于Echarts进行数据可视化、基于机器学习进行数据预测。该案例的系统架构图如下所示：

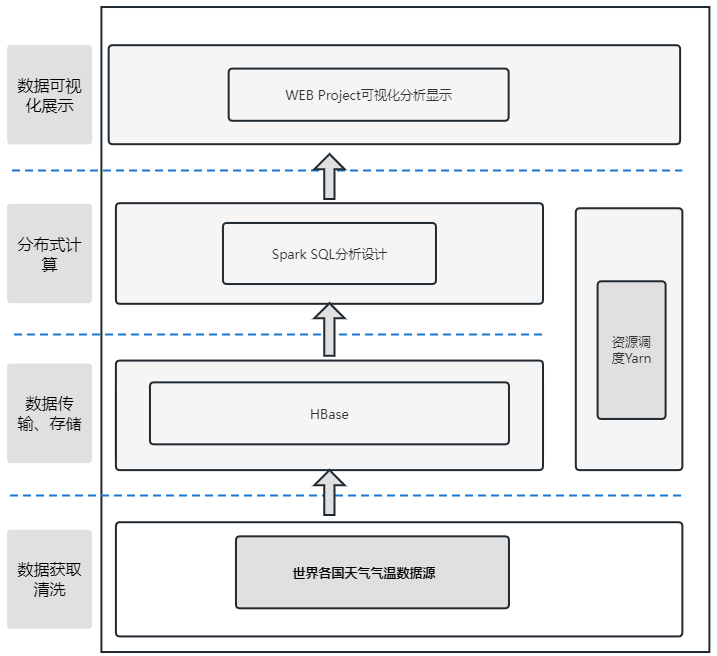


图2-1 世界气温数据分析系统架构图

### 2.2.2 设计思路

数据多维度分析，不同维度的分析也许只是展现了某种想象、某种情况的一点，有时某个现象的推测分析需要综合多个维度的分析结果，所以数据的分析统计要从全局考虑，然后再落实到某一个维度分析，细致到某天、某地区的分析。

每年天气预报的预报具有季节性，做好天气预报的分析就需要定期展示、对比以往天气进行分析。

* 同比：同比就是今年第n月与去年第n月比。同比发展速度主要是为了实现同一季节性的温度对比，用以说明本季度气温与往年季度的气温相比增加或减少的比例。
* 环比：环比就是报告期间与前一统计时间段比较。

该模块主要由Scala语言利用Spark、Hbase技术设计实现。

Hbase，高可靠、高性能、面向列、可伸缩的分布式数据库，用来代替HDFS，以满足大规模数据实时处理应用的需求。

Spark提供了一个全面、统一的框架用于管理各种有着不同性质（文本数据、图表数据等）的数据集和数据源（批量数据或实时的流数据）的大数据处理的需求。

### 2.2.3 数据集

本案例的数据源主要有countryTemperatures.csv和stateTemperatures.csv两个文件。

countryTemperatures.csv文件的数据字段如下表所示：

|  |  |
| --- | --- |
| 字段 | 说明 |
| id | 编号 |
| dt | 日期 |
| AverageTemperature | 平均气温 |
| TemperatureConfidence | 气温置信度 |
| Country | 国家 |

stateTemperatures.csv文件的数据字段如下表所示：

|  |  |
| --- | --- |
| 字段 | 说明 |
| id | 编号 |
| dt | 日期 |
| AverageTemperature | 平均气温 |
| TemperatureConfidence | 气温置信度 |
| State | 州 |
| Country | 国家 |

### 2.2.4 统计分析

天气预报大数据分析主要应用于气象部门以及需要关注天气预测的相关部门。本案例主要是获取国外和国内历年的气温数据，采用大数据分析技术实现从多个不同维度的分析，并预测一段时间内的气温变化情况为相关部门提供一定的数据参考依据。

灾害性天气预报数据分析案例主要实现：2008-2012年全球的平均地表温度统计分析、统计中国每年每季度的气温平均值统计分析、统计中国不同省份不同季度的平均气温等多个不同维度的分析。

灾害性天气预报数据分析的实现采用IDEA作为开发环境，maven作为项目构建和管理工具，规范化项目的管理。

**1） 在IDEA中创建maven项目**

weatherprocess模块主要采用Spark SQL实现灾害性天气预报的数据分析，分析的结果写入MySQL数据库。在weatherproject/pom.xml文件中添加Spark Core、Spark SQL、等相关依赖包，代码如下所示。

代码2-1 添加核心依赖包

|  |
| --- |
| <properties>  <spark.version>3.0.0</spark.version>  <scala.version>2.12</scala.version>  </properties>  <dependencies>  <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-core\_${scala.version}</artifactId>  <version>${spark.version}</version>  </dependency>    <dependency>  <groupId>org.apache.spark</groupId>  <artifactId>spark-sql\_${scala.version}</artifactId>  <version>${spark.version}</version>  </dependency>  </dependencies> |

**2） 加载数据**

将全球气温数据上传至HDFS。

**3） 封装工具类**

灾害性天气预报数据分析的不同维度分析的总体设计思路是读取HDFS中的数据，根据需求实现不同维度的分析，然后将分析的结果写入MySQL数据库。

在项目中建立DBTools工具类实现数据源的读取以及分析结果的保存，关键代码如下所示。

代码2-2 DBTools工具类代码清单

|  |
| --- |
| package com.software.util  import org.apache.hadoop.conf.Configuration  import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}  /\*\*  \* 读取HBase数据源的工具类  \*/  object DBTools {  /\*\*  \* 功能：获取Session对象  \* @param appname  \* @param master  \*/  def getSession(appname:String,master:String) :SparkSession= {  val session: SparkSession =  SparkSession.builder().master(master).appName(appname).getOrCreate()  session  }  /\*\*  \* 分析结果写入MySQL数据库  \* @param tableName  \* @param result  \*/  def WriteMySql(tableName:String,result:DataFrame): Unit ={  result.write  .format("jdbc")  .option("url","jdbc:mysql://127.0.0.1:3306/weatherdb")  .option("driver","com.mysql.jdbc.Driver")  .option("user","root")  .option("password","123456")  .option("dbtable",tableName)  .mode(SaveMode.Overwrite)  .save()  }  } |

**4）统计2008-2012年全球的平均地表温度**

统计2008-2012年全球的平均地表温度的实现，在weatherprocess模块中创建GlobalTemperatureProcess伴生对象，在GlobalTemperatureProcess伴生对象中定义方法实现Spark读取HDFS上的数据，关键代码如下所示：

代码2-3 统计分析全球的平均地表温度数据分析的代码清单

|  |
| --- |
| package com.software.process  import java.text.SimpleDateFormat  import java.util.Date  import com.software.process.RegionalTemperatureProcess.regexpReplace  import com.software.util.DBTools  import org.apache.log4j.{Level, Logger}  import org.apache.spark.rdd.RDD  import org.apache.spark.sql.functions.expr  import org.apache.spark.sql.{DataFrame, SparkSession}  //全球气温的统计分析  object GlobalTemperatureProcess extends Serializable {  Logger.getLogger("org").setLevel(Level.ERROR)  def main(args: Array[String]): Unit = {  //编码阶段创建的SparkSession对象  val sparkSession: SparkSession =  SparkSession.builder().master("local[\*]")  .appName("GlobalTemperatureProcess")  .getOrCreate()  //部署运行阶段创建的SparkSession对象  /\*val sparkSession: SparkSession =  SparkSession.builder()  .appName("GlobalTemperatureProcess")  .getOrCreate()\*/  //2.加载csv文件（数据源）  import sparkSession.implicits.\_  import org.apache.spark.sql.functions.\_  val inputWebfile = "data/countryTemperatures.csv"  var weatherData: DataFrame = sparkSession.read.format("csv")  .option("header", true)  .option("multiLine", true)  .load(inputWebfile)  //正则表达式的替换  weatherData=regexpReplace(weatherData,"dt","/","-")  weatherData.show(5)  //3.注册临时表  weatherData.createOrReplaceTempView("tbl\_globalCountryTemperatures")  //3.1统计2008-2012年全球的平均地表温度  var result :DataFrame=  sparkSession.sql(  """  |select year(dt) year ,round(avg(AverageTemperature),2) average,Country  |from tbl\_globalCountryTemperatures  |where AverageTemperature !='null' and dt!=""  |and year(dt) between 2008 and 2012  |group by Country,year(dt)  |order by average desc  | """.stripMargin)  result.show(false)  DBTools.WriteMySql("country\_2008\_2012\_temperatures",result)  //3.2 统计中国每年每季度的平均气温（1836-2012年）  val avgresult :DataFrame=  sparkSession.sql(  """  |SELECT year(dt) year,round(AVG(averageTemperature),3) AS averageTemperature,CEIL(MONTH(dt)/3) AS season  |FROM tbl\_globalCountryTemperatures  |where AverageTemperature != '' and dt!=""  |AND country='China'  |and year(dt)>=1990  |GROUP BY year(dt),season  |ORDER BY year(dt),season  | """.stripMargin)  avgresult.show(false)  DBTools.WriteMySql("china\_season",avgresult)  }  //正则替换  def regexpReplace(df:DataFrame,columnName:String,regexp:String,newValue:Any):DataFrame={  val exprString:String = "regexp\_replace("+columnName+",'"+regexp+"','"+newValue+"')"  df.withColumn(columnName,expr(exprString).alias(columnName))  }  } |

**5）统计国内不同地区不同月份的平均气温（1990-2012年）**

统计中国不同地区不用月份的平均气温，获取数据源并转换为DataFrame对象的方法参考代码5-4。实现1990~2012年国内每年每季度的平均气温关键代码如下所示。

代码2-4 统计国内每年每季度的代码清单

|  |
| --- |
| package com.software.process  import com.software.util.DBTools  import org.apache.log4j.{Level, Logger}  import org.apache.spark.SparkConf  import org.apache.spark.rdd.RDD  import org.apache.spark.sql.functions.expr  import org.apache.spark.sql.{DataFrame, SparkSession}  object RegionalTemperatureProcess {  Logger.getLogger("org").setLevel(Level.ERROR)  def main(args: Array[String]): Unit = {  //1.创建SparkSession对象（启动Spark）  val sparkSession: SparkSession =  SparkSession.builder().master("local[\*]")  .appName("RegionalTemperatureProcess")  .getOrCreate()  /\*val sparkSession: SparkSession =  SparkSession.builder()  .appName("RegionalTemperatureProcess")  .getOrCreate()\*/  //2.加载csv文件（数据源）  import sparkSession.implicits.\_  import org.apache.spark.sql.functions.\_  val inputWebfile = "data/stateTemperatures.csv"  var weatherData: DataFrame = sparkSession.read.format("csv")  .option("header", true)  .option("multiLine", true)  .load(inputWebfile)  //正则表达式的替换  weatherData=regexpReplace(weatherData,"dt","/","-")  weatherData.show(5)  //3.注册临时表  weatherData.createOrReplaceTempView("tbl\_globalLandTemperaturesByState")  //4.统计中国不同省份每个月的气温  val result :DataFrame=  sparkSession.sql(  """  |SELECT state,substring(dt,0,4) year,  |sum(CasE month(dt) WHEN 1 THEN ROUND(AverageTemperature,3) ELSE 0 END) as January,  |sum(CasE month(dt) WHEN 2 THEN ROUND(AverageTemperature,3) ELSE 0 END) as February,  |sum(CasE month(dt) WHEN 3 THEN ROUND(AverageTemperature,3) ELSE 0 END) as March,  |sum(CasE month(dt) WHEN 4 THEN ROUND(AverageTemperature,3) ELSE 0 END) as April,  |sum(CasE month(dt) WHEN 5 THEN ROUND(AverageTemperature,3) ELSE 0 END) as May,  |sum(CasE month(dt) WHEN 6 THEN ROUND(AverageTemperature,3) ELSE 0 END) as June,  |sum(CasE month(dt) WHEN 7 THEN ROUND(AverageTemperature,3) ELSE 0 END) as July,  |sum(CasE month(dt) WHEN 8 THEN ROUND(AverageTemperature,3) ELSE 0 END) as August,  |sum(CasE month(dt) WHEN 9 THEN ROUND(AverageTemperature,3) ELSE 0 END) as September,  |sum(CasE month(dt) WHEN 10 THEN ROUND(AverageTemperature,3) ELSE 0 END) as October,  |sum(CasE month(dt) WHEN 11 THEN ROUND(AverageTemperature,3) ELSE 0 END) as November,  |sum(CasE month(dt) WHEN 12 THEN ROUND(AverageTemperature,3) ELSE 0 END) as December  |FROM tbl\_globalLandTemperaturesByState  |where Country='China'  |group by state,substring(dt,0,4)  |order by substring(dt,0,4)  |""".stripMargin)  result.show(false)  DBTools.WriteMySql("china\_province\_temperatures",result)  //5.统计中国不同地区每年每季度的平均气温  val chinaresult :DataFrame=  sparkSession.sql(  """  |SELECT state ,YEAR(dt) AS year,round(AVG(averageTemperature),3) AS AverageSeasonTemperature,CEIL(MONTH(dt)/3) AS season  |FROM tbl\_globalLandTemperaturesByState  |where AverageTemperature != ''  |AND country='China'  |GROUP BY state,year,season  |ORDER BY year desc  | """.stripMargin)  DBTools.WriteMySql("china\_season\_compare\_temperature",chinaresult)  }  //正则替换  def regexpReplace(df:DataFrame,columnName:String,regexp:String,newValue:Any):DataFrame={  val exprString:String = "regexp\_replace("+columnName+",'"+regexp+"','"+newValue+"')"  df.withColumn(columnName,expr(exprString).alias(columnName))  }  } |

**6）统计国内不同地区不同季度的平均气温（1990-2012年）**

统计中国不同地区不用季度的平均气温，实的关键代码如下所示。

代码2-5 统计国内不同省份不同季度温度数据分析的代码清单

|  |
| --- |
| //5.统计中国不同地区每年每季度的平均气温  val chinaresult :DataFrame=  sparkSession.sql(  """  |SELECT state ,YEAR(dt) AS year,round(AVG(averageTemperature),3) AS AverageSeasonTemperature,CEIL(MONTH(dt)/3) AS season  |FROM tbl\_globalLandTemperaturesByState  |where AverageTemperature != ''  |AND country='China'  |GROUP BY state,year,season  |ORDER BY year desc  | """.stripMargin)  DBTools.WriteMySql("china\_season\_compare\_temperature",chinaresult) |