# Spark学习

**一、Spark的优点：**

**速度**：在内存中运行程序比Hadoop的MapReduce要快100倍，在磁盘上的速度要快10倍

Spark 具有支持非循环数据流和内存计算的高级DAG执行引擎

**易操作：**可以用Java、Scala、Python、R语言来快速编写应用

**大致功能：**集合sql、streaming、和复杂的分析

Spark为一堆库（包括sql和dataframes），MLlib机器学习，GraphX和Spark Streaming提供功能，您可以在相同的应用程序中无缝的组合这些库

**可以在任何地方执行：**可以在Hadoop，Mesos，srandalone或者在云上都能跑

**可以访问不同的数据源：**包括HDFS、Cassandra、HBase和S3

SparkSQL是用于处理结构化数据的spark模块，可以将sql查询和spark程序无缝混合，Spark SQL允许使用SQL或熟悉的DataFrame API来查询Spark程序中的结构化数据

**统一数据访问：**以相同的方式连接到任何数据源。DataFrames和sql提供了访问各种数据源的常用方式，包括Hive，Avro，Parquet，ORC，JSON和JDBC。

**二、Spark SQL**

**写sql时，在进行逻辑解析时会把sql分成三个部分，project、DataSource、Filter模块，当生成执行部分时又称为Result模块**

**写完sql查询语句，sql的查询引擎首先把我们的查询语句进行解析，也就是Parse过程，解析的过程是把我们写的查询语句进行分割，把project、DataSource和Filter三个部分解析出来从而形成一个逻辑解析tree，结束后进入到Bind过程，在Bind过程中，其实就是把Parse过程后形成的逻辑解析tree和数据库的数据字典绑定，绑定后会形成一个执行tree，完成Bind过程后，数据库会根据执行计划的统计信息选择一个最优的执行计划，这个过程就是Optimize（优化）过程，最后就是执行Execute**

**sparksql流程：**

**DataSource从外部数据读入到SparkSQL中，然后SparkSQL进行处理逻辑，把经过处理后的结果写入外部存储（OutPut）**

**Spark SQL是用于处理结构化数据的Spark模块，可以将SQL查询和Spark程序无缝结合**

context = HiveContext(sc) results = context.sql(“SELECT \* FROM people”) name=results.map(lambda p : p.name)

**统一数据访问：**以相同的方式连接任何数据源

Context.jsonFile(“s3n://….”).registerTempTable(“json”) results=context.sql(“SELECT \* FROM people JOIN josn…”)

**集成Hive：**在现有仓库上运行SQL或HiveQL查询，Spark SQL支持HiveQL语法以及Hive SerDes和UDF，允许访问现有的Hive仓库

**标准连接：**通过JDBC或ODBC连接（JDBC是一种用于执行SQL语句的JAVA API，可以为多种关系数据库提供统一访问，也可以据此构建更高级的工具和接口；ODBC提供了一组对数据库访问的标准API，这些API利用SQL来完成大部分任务）

**性能与可扩展性：**Spark SQL包括基于成本的优化器，柱状存储和代码生成，使查询更快速。同时，它可以使用Spark引擎扩展到数千个节点和多小时查询，从而提供完整的中间查询容错能力

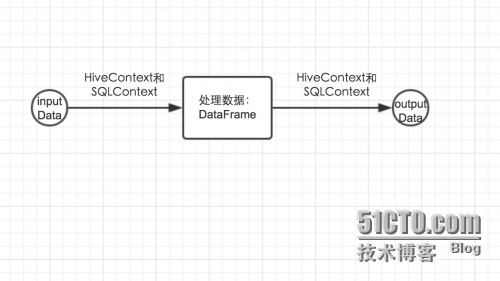
Sparksql怎么把数据读入？：SparkSQL采用的是DataFrame数据结构来组织读入到SparkSQL中的数据，DataFrame数据结构其实和数据库的表结构差不多，数据是按照行来进行存储，同时还有一个schema，就相当于数据库的表结构，记录这每一行数据属于哪个字段

**Spark sql的使用**

Spark SQL是用于结构化数据处理的Spark模块。与基本的Spark RDD API不同，Spark SQL提供的接口为Spark提供了有关正在执行的数据和计算结构的更多信息。在内部，Spark SQL使用这些额外的信息来执行额外的优化

Spark SQL的一个用途是执行SQL查询。Spark SQL也可用于从现有的Hive安装中读取数据。当另一种编程语言运行SQL时，结果将作为Dataset/DataFrame返回，还可以使用命令行或者JDBC/ODBC与SQL接口进行交互

当用sparksql来解决需求时，经历了三个步骤：读入数据 -> 对数据进行处理 -> 写入最后结果，这三个步骤用的主要类其实就是三个：读入数据和写入最后结果用到两个类HiceContext和SQLContext，对数据进行处理用到的是DataFrame类，此类是把数据从外部读入到内存后，数据在内存中存储的基本数据结构



**读数据操作：**

在解决需求时，首先是读入数据，把数据读入到内存中去，读数据SQLContext提供了两个方法

1. 、对数据源文件进行操作

Val sql = new SQLContext(sc)

Val peopleInfo = sql.read.json(“文件路径”) //其中peopleInfo返回的结果是：org.apache.spark.sql.DataFrame类型的

（**执行这些命令的中间过程：首先sparksql先找到文件，以解析json的形式进行解析，同时通过json的key形成schema**）

**DataFrame**是sparksql处理大数据的基本并且是核心的数据结构，是来存储sparksql把数据读入到内存中，**数据在内存中进行存储的基本数结构**。它采用的存储是类似于数据库的表的形式进行存储的。DataFrame按照数据一行一行的进行数据存储，**数据表的字段和字段类型都是存放在schema中**，可以调用schema来查看它存储的是什么。

PeopleInfo.schema //返回的结果是org.apache.spark.sql.type.StructType类型的

read函数还提供了其他读入数据的接口：

|  |  |
| --- | --- |
| 函数 | 用途 |
| json(path:String) | 读取json文件用此方法 |
| table(tableName:String) | 读取数据库中的表 |
| jdbc(url: String,table: String,predicates:Array[String],connectionProperties:Properties) | 通过jdbc读取数据库中的表 |
| orc(path:String) | 读取以orc格式进行存储的文件 |
| parquet(path:String) | 读取以parquet格式进行存储的文件 |
| schema(schema:StructType) | 这个是一个优化，当我们读入数据的时候指定了其schema，底层就不会再次解析schema从而进行了优化，一般不需要这样的优化，不进行此优化，时间效率还是可以接受 |

1. 、选择部分字段。

这个方法主要是处理从数据表中选择部分字段，而不是选择表中的所有字段。这种方式是直接写sql的查询语句。把上述json格式的数据保存为数据库中表的格式。这种方式只能处理数据库表的数据

Val peopleInfo = Sql.sql(“””select id ,name,age from peopleInfo”””.stringMargin) //其中的StringMargin方法是来解析我们写的sql语句的。返回的结果是和read读取返回的结果是一样的 结果类型是org.apache.spark.sql.DataFrame 需要注意的是其返回的schema中的字段的顺序和我们查询的顺序是不一致的

**写入数据操作**

一般情况下，需要写入的数据是一个DataFrame类型的，如果不是DataFrame类型的，我们需要将其转换为DataFrame类型。把没有schema的数据，构建schema类型

1. 、通过类构建schema

val sql = new SQLContext(sc) //创建一个SQLContext对象

import sql.implicits.\_ //这里的import不是导入一个jar包，这个sql是上面定义的sql

val people = sc.textFile(“people.txt”) //采用spark的类型读入数据，因为如果使用//SQLContext进行读入，会自动创建schema

case class People(id:Int,name:String,age:Int) // 定义了一个类

val peopleInfo = people.map(lines => lines.split(“,”))

.map(p => People(p(0).toInt,p(1),p(2).toInt)).toDF //这样的一个toDF就创建了一个DataFrame，如果没有导入sql.implicits.\_,那么这个toDF方法是不可以用的

1. 、直接构造schema

直接构造schema，需要把我们的数据类型进行转化成Row类型，不然会报错

Val sql = new SQLContext(sc) //创建一个SQLContext对象

Val people = sc.textFile(“people.txt”).map(lines => lines.split(“,”)) //读取txt文件中的内容，//并且按照逗号分隔

Val peopleRow = sc.map(p => Row(p(0),p(1),p(2))) //把RDD转化为RDD（Row）类型

Val schema = StructType(StructFile(“id”,IntergerType,true)::

StructFile(“name”,StringType,true)::

StructFile(“age”,IntergerType,true)::Nil)

Val peopleInfo = sql.createDataFrame(peopleRow,schema) //peopleRow的每一行的数据

//类型一定要和schema的一致，否则会报错，说类型无法匹配，同时peopleRow

//的每一行的长度也要和schema一致，否则也会报错

构造schema用到了两个类StructType和StructFile，其中StructFile类的三个参数分别是（字段名称，类型，数据是否存在可以用null填充）

**直接构造有很大的制约性，必须保证Row中的数据类型和我们定义的schema类型一致，长度也一样**

**完整的写数据操作**

Val sql = new SQLContext(sc)

Val people = sc.textFile(“people.txt”).map(lines => lines.split(“,”))

Val peopleRow = sc.map(p => Row(p(0),p(1),p(2)))

Val schema = StructType(StructFile(“id”,IntergerType,true)::

StructFile(“name”,StringType,true)::

StructFile(“age”,IntergerType,true)::Nil)

Val peopleInfo = sql.createDataFrame(peopleRow,schema)

peopleInfo.registerTempTable(“tempTable”) //只有有了这个注册的表tempTable，才可以

//sql.sql(“” ””)进行查询，这个是在内存中注册//一个临时表用户查询

sql.sql.sql("""

|insert overwrite table tagetTable

|select

| id,

| name,

| age

|from tempTable

""".stripMargin) //这样就把数据写入到了数据库目标表tagetTable中

**对schema的操作**

Val sql = new SQLContext(sc)

Val people = sql.read.json(“people.txt”) //people是一个DataFrame类型的对象

//数据读进来，然后查看下schema

people.schema

//返回的类型是org.apache.spark.sql.types.StructType

//返回的内容是：//StructType(StructField(age,LongType,true),

//StructField(id,LongType,true),

//StructField(name,StringType,true))

//用数组的形式分schema

People.dtypes //返回的类型是：Array[(String,String)]

//返回的结果是：Array((age,LongType),(id,LongType),(name,StringType))

//返回schema中的字段

People.columns //返回的类型是：Array[String]

//返回的结果是：Array(age,id,name)

//以tree的形式打印输出schema

People.printSchema //返回的类型是：tree状结果

//返回的结果是： |-- age: long (nullable = true)

// |-- id: long (nullable = true)

// |-- name: string (nullable = true)

下面是一些sql中的语句：

|  |  |
| --- | --- |
| 方法(sql使我们定义的sql = new SQLContext(sc)) df是一个DataFrame对象 | 实例说明 |
| sql.read.table(tableName) | 读取一张表的数据 |
| df.where(),          df.filter() | 过滤条件，相当于sql的where部分;  用法：选择出年龄字段中年龄大于20的字段。  返回值类型：DataFrame    df.where("age >= 20"),df.filter("age >= 20") |
| df.limit() | 限制输出的行数，对应于sql的limit  用法：限制输出一百行  返回值类型：DataFrame    df.limit(100) |
| df.join() | 链接操作，相当于sql的join  对于join操作，下面会单独进行介绍 |
| df.groupBy() | 聚合操作，相当于sql的groupBy  用法：对于某几行进行聚合  返回值类型：DataFrame    df.groupBy("id") |
| df.agg() | 求聚合用的相关函数，下面会详细介绍 |
| df.intersect(other:DataFrame) | 求两个DataFrame的交集 |
| df.except(other:DataFrame) | 求在df中而不在other中的行 |
| df.withColumn(colName:String,col:Column) | 增加一列 |

|  |  |
| --- | --- |
| df.withColumnRenamed(exName,newName) | 对某一列的名字进行重新命名 |
| df.map(),  df.flatMap,  df.mapPartitions(),  df.foreach()  df.foreachPartition()  df.collect()  df.collectAsList()  df.repartition()  df.distinct()  df.count() | 这些方法都是spark的RDD的基本操作，其中在DataFrame类中也封装了这些方法，需要注意的是这些方法的返回值是RDD类型的，不是DataFrame类型的，在这些方法的使用上，一定要记清楚返回值类型，不然就容易出现错误 |
| df.select() | 选取某几列元素，这个方法相当于sql的select的功能  用法：返回选择的某几列数据  返回值类型：DataFrame    df.select("id","name") |

**Spark-sql的join操作：**

1. 内连接，等值连接，会把连接的列合并成一个列

Val sql = new SQLContext(sc)

Val pInfo = sql.read.json(“people.txt”)

Val pSalar = sql.read.json(“salary.txt”)

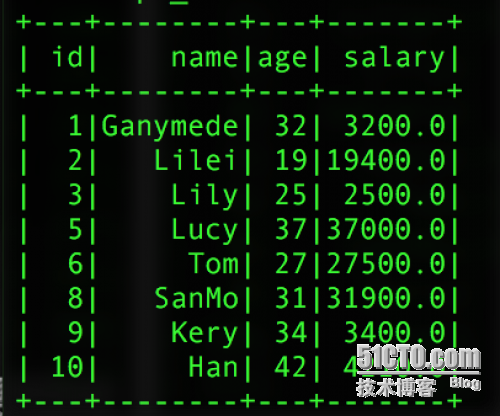
Val info\_salary = pInfo.join(pSalar,”id”) //单个字段进行内连接

Val info\_salary1 = pInfo.join(pSalar,Seq(“id”,”name”)) //多个字段进行连接

**单个字段进行连接，如下图所示：**



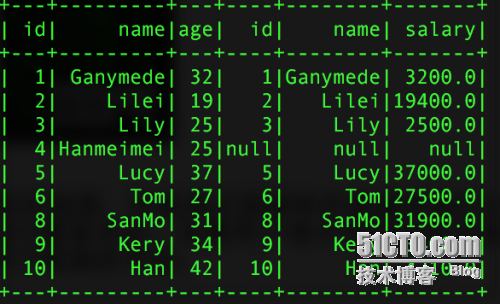
**多个字段进行连接，如下图所示：**



1. join还支持左连接和右连接，但是其左连接和右连接和我们sql的连接的意思是一样的，在连接的时候不能对字段进行重新命名

//单字段连接

Val left = pInfo.join(pSalar,pInfo(“id”) === pSalar(“id”),”left\_outer”)



//多字段连接

Val left2 = pInfo.join(pSalar,pInfo(“id”) === pSalar(“id”) and pInfo(“name”) === pSalar(“name”),”left\_outer”)



**SparkSQL的agg操作**

其中sparkSQL的agg是sparkSQL聚合操作的一种表达式，当我们调用agg时，其一般情况下都是和groupBy（）一起使用的

//读取表格中的数据

Val pSalar = new SQLContext(sc).read.json(“salary.txt”)

//将某一列取出来，进行agg操作，然后进行满足条件的聚合操作

Val group = pSalar.groupBy(“name”).agg(“salary” -> “avg”)

Val group2 = pSalar.groupBy(“id”,”name”).agg(“salary” -> “avg”)

Val group3 = pSalar.groupBy(“name”).agg(Map(“id” -> “avg”,”salary” -> “max”))

**SparkSQ;的na操作**

sparkSQL的na方法，返回的是一个DataFrameFuctions对象，此类主要是对DataFrame中值为null的行操作，只提供三个方法，drop（）删除行，fill（）填充行，replace（）代替行的操作。