# 考试题：SparkCore和SparkSQL

总分100分制，答题时间200分钟

最后得分：

## 单选题（10题，每题2分，共20分）

1、以下程序在第几行有错误（A）？

1 class Person{

2 protected val salary = 100.00

3 protected[this] var address = "BeiJing"

4 }

5 class Employee extends Person{

6 private val name = "Garry"

7 def makeAddressDifference(emp:Employee): Unit ={

8 println("my address=> " + address +", your address=> " + emp.address)

9 }

10}

A) 8 B) 6 C) 3 D) 2

2、Spark Job默认的调度模式（A）？

1. FIFO B) FAIR C) 无 D) 运行时指定

3、下面哪个不是RDD的特点(C )？

A) 可分区 B) 可序列化 C) 可修改 D) 可持久化

4、以下程序的输出内容为（A）？

object SingletonOps {

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

val s1 = Singleton

val s2 = Singleton

s1.num = 6

s1.show

s2.show

}

}

object Singleton {

var num:Int = 5

def show = println("num: " + num)

}

A) 6和6 B)6和5 C)5和5 D)5和6

5、关于累加器，下面哪个是错误的 （D）？

A) 支持加法 B) 支持数值类型 C) 可并行 D) 不支持自定义类型

6、Stage 的 Task 的数量由什么决定 （A）？

A) partition B) Job C) Stage D) TaskScheduler

7、下面哪个操作肯定是宽依赖 （C）？

A) map B) coalesce C) reducByKey D) union

8、 SparkCore默认的存储级别 （A）？

A) MEMORY\_ONLY B) MEMORY\_ONLY\_SER

C) MEMORY\_AND\_DISK D) MEMORY\_AND\_DISK\_SER

9、DataFrame和RDD最大的区别（B）？

A) 科学统计支持 B) 多了schema C) 存储方式不一样 D) 外部数据源支持

10、spark.deploy.recoveryMode 不支持那种 （C）？

A) ZooKeeper B) FileSystem C) Hadoop

## 填空（4题，每空1分，共15分）

1、JVM运行时区域分为\_\_堆\_\_\_、\_\_方法区\_\_\_\_\_、\_\_程序计数器\_\_\_\_\_和栈四个区域，其中栈又可以分为\_\_虚拟机栈\_\_\_\_\_和\_\_本地方法栈\_\_\_\_\_。

2、Spark JOB的依赖分为\_\_\_窄依赖\_\_\_\_、\_\_宽依赖\_\_\_\_\_两种，其中Stage划分基于\_\_\_宽依赖\_\_\_\_阶段？

3、JVM堆区域可以分为\_\_\_新生\_\_\_\_代、\_\_永久\_\_\_\_\_代、\_\_老年\_\_\_\_\_代。

4、SparkSQL数据集DataFrame或者Dataset在落地过程中可选SaveMode有哪四种？\_Append\_\_\_\_\_\_、\_\_\_Ignore\_\_\_\_、\_ErrorIfExists\_\_\_\_\_\_、\_OverWrite\_\_\_\_\_\_。

## 编程（5题，共65分）

1. 通过sparksql加载如下文件，文件内容见附件：

读数据进行计算，计算每个科目的平均值，然后将结果存储到mysql数据库，数据库和表自定义，要求源代码以及执行结果截图。(10分)

**package** com.desheng.bigdata.exam  
  
**import** java.util.Properties  
  
**import** org.apache.spark.sql.SparkSession  
  
**object** ExamTest1 {  
 **def** main(args: Array[String]): Unit = {  
 **val** spark = SparkSession.*builder*()  
 .appName(**"ExamTest1"**)  
 .master(**"local[\*]"**)  
 .getOrCreate()  
  
 **val** df = spark.read.csv(**"file:///E:/data/spark/sql/student.csv"**)  
 .toDF(**"sid"**, **"name"**, **"age"**, **"gender"**, **"course"**, **"score"**)  
  
 df.createTempView(**"student"**)  
 **val** sql =  
 **"""  
 |select  
 | course,  
 | round(avg(score), 2) avg\_score  
 |from student  
 |group by course  
 """**.stripMargin  
 **var** ret = spark.sql(sql)  
 *println*(**"--------------------"**)  
 **import** spark.implicits.\_  
 **val** ds = df.map(row => {  
 **val** sid = row.getString(0).toInt  
 **val** name = row.getString(1)  
 **val** age = row.getString(2).toInt  
 **val** gender = row.getString(3)  
 **val** course = row.getString(4)  
 **val** score = row.getString(5).toDouble  
 *Student*(sid, name, age, gender, course, score)  
 })  
 ret = ds.groupBy(**"course"**).avg(**"score"**)  
  
 **val** url = **"jdbc:mysql://localhost:3306/sz-1902"  
 val** table = **"student"  
 val** properties = **new** Properties()  
 properties.put(**"user"**, **"szxg"**)  
 properties.put(**"password"**, **"sorry"**)  
 ret.write.jdbc(url, table, properties)  
 spark.stop()  
 }  
}  
  
**case class** Student(sid:Int, name:String, age:Int, gender:String, course:String, score:Double)

2.编写SparkSQL自定义UDF完成对字符串函数substr(str, pos[, len])的模拟。（10分）

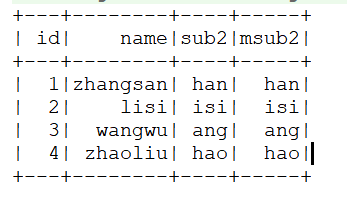
test\_sql

|  |  |  |
| --- | --- | --- |
| id | int | 用户id |
| name | string | 用户名称 |
| age | int | 用户年龄 |

数据见附件：

可以基于用户名称进行统计。

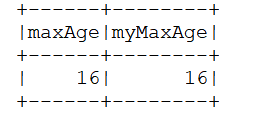
**object** ExamTest2 {  
 **def** main(args: Array[String]): Unit = {  
 **val** spark = SparkSession.*builder*()  
 .appName(**"ExamTest2"**)  
 .master(**"local[\*]"**)  
 .getOrCreate()  
  
 spark.udf.register[String, String, Int, Int](**"mySubstr"**, *mySubstr*)  
  
 **val** df = spark.read.json(**"file:///C:/Users/thinkpad/Desktop/test-sql.json"**)  
  
 df.createTempView(**"person"**)  
 **val** sql =  
 **"""  
 |select  
 | id,  
 | name,  
 | substr(name, 2, 3) sub2,  
 | mySubstr(name, 2, 3) msub2  
 |from person  
 """**.stripMargin  
 spark.sql(sql).show  
 spark.stop()  
 }  
  
 **def** mySubstr(str:String, pos:Int, len:Int = 0): String = {  
 **if**(pos >= 0 && pos < str.length - 1) {  
 **if** (len == 0) {  
 str.substring(pos - 1)  
 } **else** {  
 str.substring(pos - 1, pos + len - 1)  
 }  
 } **else** {  
 **null** }  
 }  
}



3.举例编写SparkSQL自定义UADF完成对max函数的模拟。（15分）

说明：表结构和数据通上题，可以基于age进行处理

**object** ExamTest3 {  
 **def** main(args: Array[String]): Unit = {  
 **val** spark = SparkSession.*builder*()  
 .appName(**"ExamTest3"**)  
 .master(**"local[\*]"**)  
 .getOrCreate()  
  
 spark.udf.register(**"myMax"**, **new** MyMax)  
  
 **val** df = spark.read.json(**"file:///C:/Users/thinkpad/Desktop/test-sql.json"**)  
  
 df.createTempView(**"person"**)  
 **val** sql =  
 **"""  
 |select  
 | max(age) maxAge,  
 | myMax(age) myMaxAge  
 |from person  
 """**.stripMargin  
 spark.sql(sql).show  
 spark.stop()  
 }  
}  
  
**class** MyMax **extends** UserDefinedAggregateFunction {  
 **override def** inputSchema: StructType = *StructType*(  
 *StructField*(**"age"**, DataTypes.*IntegerType*, **false**) :: *Nil* )  
  
 **override def** bufferSchema: StructType = *StructType*(  
 *StructField*(**"age"**, DataTypes.*IntegerType*, **false**) :: *Nil* )  
  
 **override def** dataType: DataType = DataTypes.*IntegerType* **override def** deterministic: Boolean = **true  
 override def** initialize(buffer: MutableAggregationBuffer): Unit = {  
 buffer.update(0, Integer.*MIN\_VALUE*)  
 }  
  
 **override def** update(buffer: MutableAggregationBuffer, input: Row): Unit = {  
 **val** max = buffer.getInt(0)  
 **if**(max < input.getInt(0)) {  
 buffer.update(0, input.getInt(0))  
 }  
 }  
  
 **override def** merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {  
 **val** max = buffer1.getInt(0)  
 **if**(max < buffer2.getInt(0)) {  
 buffer1.update(0, buffer2.getInt(0))  
 }  
 }  
  
 **override def** evaluate(buffer: Row): Int = buffer.getInt(0)  
}



1. 使用SparkSQL完成如下统计：

|  |  |  |
| --- | --- | --- |
| id | int | Article文章id |
| author | string | Article文章作者 |
| content | string | 文章内容(英文，单词用空格分割) |
| url | string | 文章链接地址 |

求：统计所有文章内容中每一个单词出现的次数，并按照每个单词出现次数降序排序的HQL(15分)。

数据见附件：

select

tmp.word,

count(tmp.word) as counts

from (select

explode(split(content, "\\s+")) word

from test-wc) tmp

group by tmp.word

order by counts desc

1. 部门表dept

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **字段涵义** |
| id | int | 部门编号 |
| name | string | 部门名称 |

员工表employee

|  |  |  |
| --- | --- | --- |
| **字段名称** | **类型** | **字段涵义** |
| id | int | 员工号id |
| dept\_id | int | 员工所在部门编号 |
| name | string | 员工姓名 |
| age | int | 员工年龄 |
| sex | int | 员工性别，0为女，1为男 |
| salary | float | 员工薪资 |

求：使用SparkSQL，根据部门表dept和员工表employee，求出按照部门分组求出每个部门薪资最高的3名员工的基本信息，包括id, name, age, sex(0显示为0，1显示为男)，薪资，所在部门，薪资排名rank的HQL(15分)。

数据见附件（说明，数据列之间以制表符“\t”为分隔符）



select

tmp.\*

from (

select

e.id,

e.name,

d.name dept\_name,

e.age,

if(e.sex == 1, '男', '女') gender,

e.salary,

row\_number() over(partition by e.dept\_id order by e.salary desc) rank

from dept d

inner join employee e on d.id = e.dept\_id

) tmp

where tmp.rank < 4

## 附加题（总分除外，如果正确附加20分）

使用代码解决spark-sql在计算wordcount过程中出现的数据倾斜问题。

object \_03SparkSQLDataskewOps {

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

val conf = new SparkConf()

.setAppName("SparkSQLDataskew")

.setMaster("local[\*]")

val spark = SparkSession.builder()

.config(conf)

.getOrCreate()

val list = List(

"zhang zhang wen wen wen wen yue yue",

"gong yi can can can can can can can can can can",

"chang bao peng can can can can can can"

)

import spark.implicits.\_

val lines = spark.createDataset(list).toDF("line")

lines.createTempView("test")

var sql =

"""

|select

| substr(t2.prefix\_word, instr(t2.prefix\_word, '\_') + 1) word,

| sum(t2.counts) as countz

|from (

| select

| concat\_ws("\_", cast(floor(rand() \* 2) as string), t1.word) prefix\_word,

| count(1) as counts

| from (

| select

| explode(split(line, '\\s+')) word

| from test

| ) t1

| group by prefix\_word

|) t2

|group by word

""".stripMargin

spark.sql(sql).show()

spark.stop()

}

}