第七次作业

目录

[Spark RDD 1](#_Toc10818378)

[技术方法 1](#_Toc10818379)

[过滤特殊字符 1](#_Toc10818380)

[WordCount 2](#_Toc10818381)

[结果 3](#_Toc10818382)

[心得 3](#_Toc10818383)

[代码 3](#_Toc10818384)

[文件过滤 3](#_Toc10818385)

[WordCount 4](#_Toc10818386)

[Spark SQL 4](#_Toc10818387)

[技术方法 5](#_Toc10818388)

[文件过滤 5](#_Toc10818389)

[导入文件到SQL 7](#_Toc10818390)

[SQL查询 7](#_Toc10818391)

[结果 8](#_Toc10818392)

[心得 8](#_Toc10818393)

[代码 9](#_Toc10818394)

[文件过滤 9](#_Toc10818395)

[导入csv 11](#_Toc10818396)

[查询 12](#_Toc10818397)

# Spark RDD

Spark RDD：对给出的莎士比亚文集Shakespere.txt进行wordcount（注：文件中包含特殊字符，请先进行过滤操作仅留下英文字符）

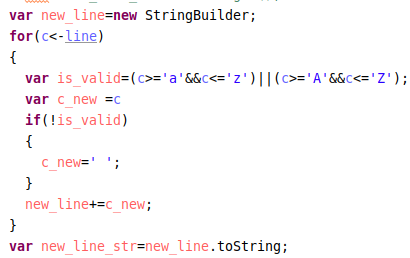
## 技术方法

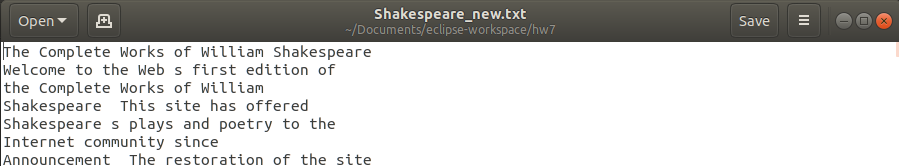
环境：eclipse

语言：scala

### 过滤特殊字符

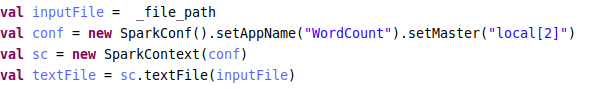
每个字符，如果不是英文字母，则替换为空格。转换后的文件保存于Shakespeare\_new.txt





### WordCount

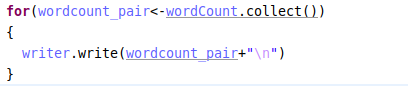
将文件读取到RDD数据库中



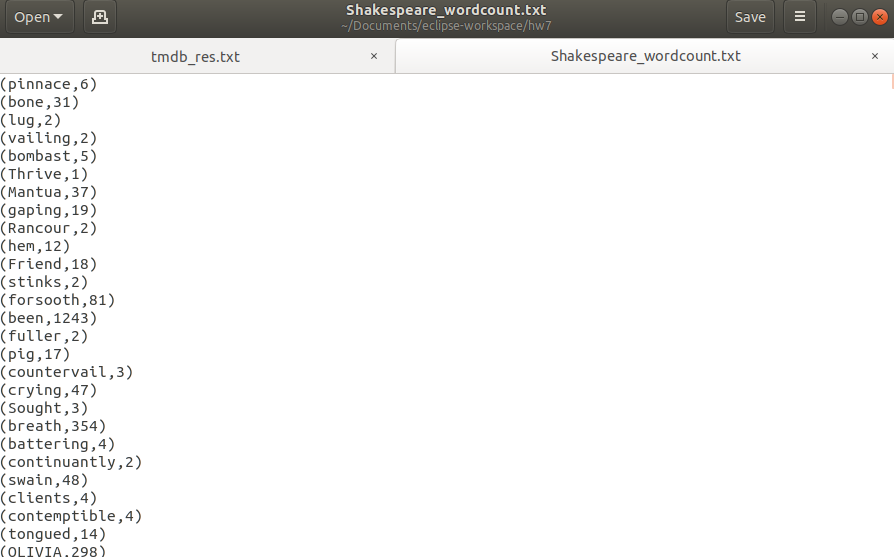
此时RDD中的每个元素为文件的一行，对RDD进行flatmap操作，每个元素即每行根据空格split成新的元素即每个单词，然后把每个单词map成(word,1)的键值对，然后根据key进行reduce，把相同key的键值对的计数相加。



此时RDD中的即为wordcount，对此时的RDD进行collect()遍历其中元素，将wordcount结果存入文件。



### 结果



## 心得

eclipse的scala library container应改为2.11版本以与scala匹配

scala依赖较多，可用maven配置依赖，并用阿里云的仓库替换中央仓库

textfile为RDD数据库，无法直接遍历，需要先用collect()转换才可遍历其中元素

## 代码

### 文件过滤

**def** file\_filter(prev\_path:String,new\_path:String):Unit=

{

**val** prev\_file=Source.fromFile(prev\_path)

**val** writer = **new** PrintWriter(**new** File(new\_path))

**for**(line <- prev\_file.getLines)

{

// val line\_char\_num=line.length();

**var** new\_line=**new** StringBuilder;

**for**(c<-line)

{

**var** is\_valid=(c>='a'&&c<='z')||(c>='A'&&c<='Z');

**var** c\_new =c

**if**(!is\_valid)

{

c\_new=' ';

}

new\_line+=c\_new;

}

**var** new\_line\_str=new\_line.toString;

writer.write(new\_line\_str+"\n")

}

prev\_file.close

writer.close();

}

### WordCount

**def** word\_count(\_file\_path:String):Unit=

{

// val inputFile = "file:///home/lwx/Documents/big\_data/hw7/Shakespeare.txt"

**val** inputFile = \_file\_path

**val** conf = **new** SparkConf().setAppName("WordCount").setMaster("local[2]")

**val** sc = **new** SparkContext(conf)

**val** textFile = sc.textFile(inputFile)

**val** wordCount = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey((a, b) => a + b)

// wordCount.foreach(println)

**val** writer = **new** PrintWriter(**new** File("Shakespeare\_wordcount.txt"))

**for**(wordcount\_pair<-wordCount.collect())

{

writer.write(wordcount\_pair+"\n")

}

writer.close()

}

# Spark SQL

Spark SQL：重现你第二次作业中在tmdb数据上实现的查询功能

在电影数据集 tmdb-5000-movie-dataset 上完成至少一个任务，比如计算出每个公司的高分电影（得分6.5以上）总收入。

## 技术方法

环境：eclipse

语言：scala

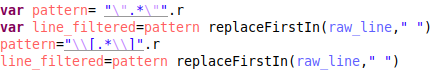
### 文件过滤

由于原csv文件中有较多的特殊字符和不必要的信息，所以首先将原文件过滤。

用正则表达式对每一行过滤出公司名字，电影评分和电影收入

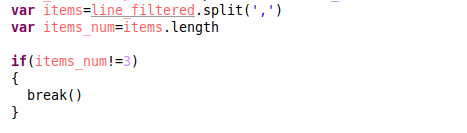
为了防止一些行有乱码，加上try-catch捕捉异常

由于公司名称是由引号里的中括号里的多个大括号组成的复杂结构，所以先匹配引号及其中的内容并替换为空格，只剩下关于评分和收入的信息。考虑到有些行的中括号外没有引号，再匹配中括号并将其替换为空格。

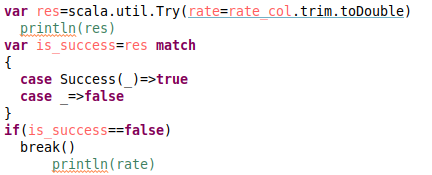


按逗号split，如果不能分成三部分，则该行格式不正确，忽略该行，继续处理下一行。

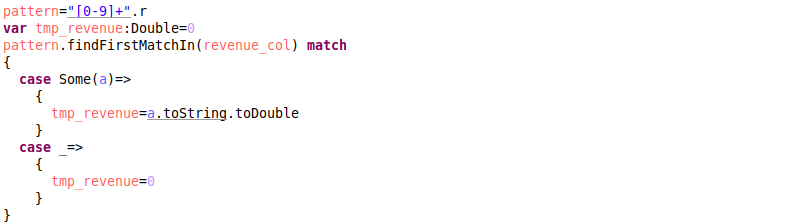
因为scala没有continue和break，所以这里的语句都放在breakable块中，所以break相当于continue



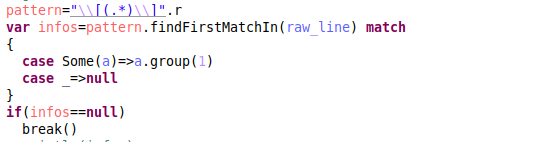
选取rate并试图将其转换为double类型。为防止格式不对导致程序中断，先调用scala的try语句试图转换，如果不成功则忽略该行，继续处理下一行。

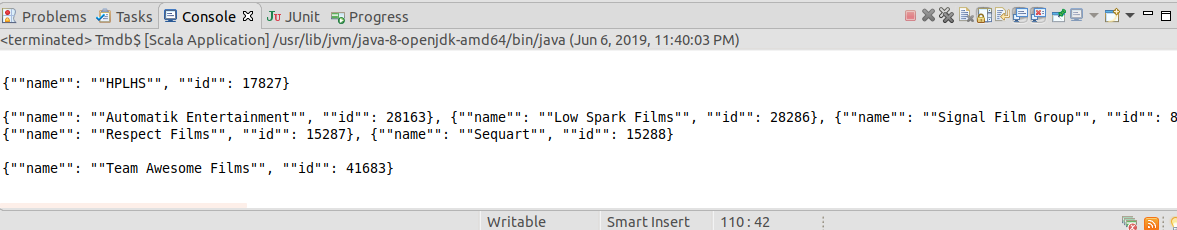


选取revenue，并用正则表达式匹配出数字部分以避免其他字符，如果匹配不到数字部分，则格式不对，继续处理下一行。

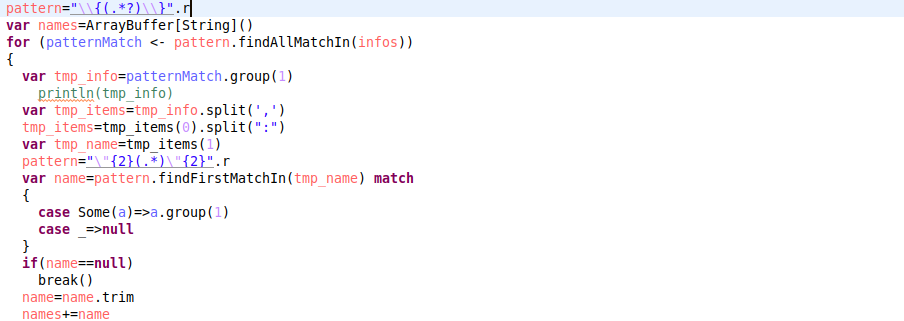


在原始的line中匹配中括号中的内容，包括公司名称的信息

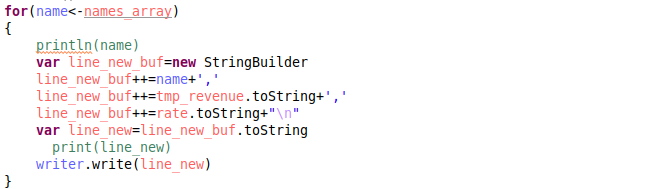




公司名称的信息包括多个公司，匹配每个花括号中的内容即为每个公司的信息。在根据逗号和冒号split即可得到每个公司的名字。



将每个公司名和这行的revenue，rate作为一行存到新的csv中



### 导入文件到SQL

建立spark session，导入implicits以进行隐式转换到dataframe

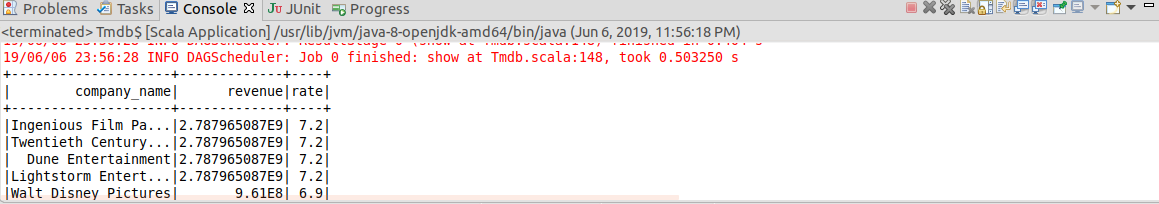


在函数外声明Film每个域及其数据类型



先将csv文件读到rdd中，再根据逗号split并根据Film类的每个域隐式类型转换为dataframe



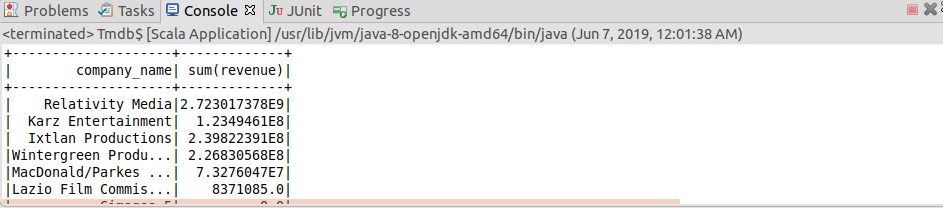


建立临时视图以进行后续sql操作

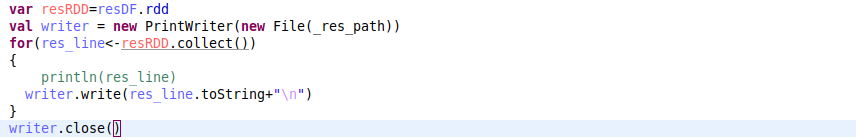


### SQL查询

建立spark session并在视图film中查询电影评分大于6.5的每个公司的总收入



得到的结果为dataframe，将dataframe转换为rdd并按行存储到结果文件中



### 结果

每个公司的高分电影（得分6.5以上）总收入



## 心得

过滤文件时，需要考虑文件中的乱码，加入try-catch。但是仅仅捕获错误会使程序停止，需要判断这行的格式是否正确，如果不正确应继续转换下一行。

scala是函数式语言，没有continue和break，所以在for循环中的每个循环放进一个breakable语句，如果需要continue就在breakable中break即可直接进入下一个循环

有些行的格式不对，需要判断每行的元素个数是否正确，revenue是否可以转换成double，是否有公司名称等，如果不符合则继续处理下一行

**import** spark.implicits.\_需要在spark session建立后导入

RDD由行组成，而dataframe类似于数据库，可用临时视图进行sql操作,两者可相互转换

dataframe直接存csv时只能存一列，可以转换为rdd后将每行存储到文件中

## 代码

**Scala**

### 文件过滤

**def** csv\_filter(\_prev\_csv\_path:String,\_new\_csv\_path:String):Unit=

{

**try**

{

**val** prev\_file=Source.fromFile(\_prev\_csv\_path)

**val** writer = **new** PrintWriter(**new** File(\_new\_csv\_path))

**for**(raw\_line <- prev\_file.getLines)

{

breakable

{

**var** pattern= "\".\*\"".r

**var** line\_filtered=pattern replaceFirstIn(raw\_line," ")

pattern="\\[.\*\\]".r

line\_filtered=pattern replaceFirstIn(raw\_line," ")

**var** items=line\_filtered.split(',')

**var** items\_num=items.length

**if**(items\_num!=3)

{

break()

}

**var** rate\_col=items(2)

**var** rate:Double=0

**var** res=scala.util.Try(rate=rate\_col.trim.toDouble)

// println(res)

**var** is\_success=res **match**

{

**case** Success(\_)=>**true**

**case** \_=>**false**

}

**if**(is\_success==**false**)

break()

// println(rate)

**var** revenue\_col=items(1);

pattern="[0-9]+".r

**var** tmp\_revenue:Double=0

pattern.findFirstMatchIn(revenue\_col) **match**

{

**case** Some(a)=>

{

tmp\_revenue=a.toString.toDouble

}

**case** \_=>

{

tmp\_revenue=0

}

}

// println(tmp\_revenue)

//get names

pattern="\\[(.\*)\\]".r

**var** infos=pattern.findFirstMatchIn(raw\_line) **match**

{

**case** Some(a)=>a.group(1)

**case** \_=>**null**

}

**if**(infos==**null**)

break()

// println(infos)

pattern="\\{(.\*?)\\}".r

**var** names=ArrayBuffer[String]()

**for** (patternMatch <- pattern.findAllMatchIn(infos))

{

**var** tmp\_info=patternMatch.group(1)

// println(tmp\_info)

**var** tmp\_items=tmp\_info.split(',')

tmp\_items=tmp\_items(0).split(":")

**var** tmp\_name=tmp\_items(1)

pattern="\"{2}(.\*)\"{2}".r

**var** name=pattern.findFirstMatchIn(tmp\_name) **match**

{

**case** Some(a)=>a.group(1)

**case** \_=>**null**

}

**if**(name==**null**)

break()

name=name.trim

names+=name

// println(name)

}

**var** names\_array=names.toArray

**if**(names\_array.length==0)

break()

**for**(name<-names\_array)

{

// println(name)

**var** line\_new\_buf=**new** StringBuilder

line\_new\_buf++=name+','

line\_new\_buf++=tmp\_revenue.toString+','

line\_new\_buf++=rate.toString+"\n"

**var** line\_new=line\_new\_buf.toString

// print(line\_new)

writer.write(line\_new)

}

}

}

prev\_file.close()

writer.close()

}

**catch**

{

**case** e:Exception => e.printStackTrace()

}

}

### 导入csv

**case** **class** Film(company\_name:String,revenue:Double,rate:Double)

**def** load\_csv(\_csv\_path:String):Unit=

{

**val** spark = SparkSession.builder().appName("load\_csv").master("local").getOrCreate()

**import** spark.implicits.\_

// val dataDF = spark.read.csv(\_csv\_path)

// dataDF.withColumnRenamed("\_c0","company\_name")

// dataDF.show()

**val** filmRDD=spark.sparkContext.textFile(\_csv\_path)

**var** filmDF=filmRDD.map(\_.split(",")).map(a => Film(a(0), a(1).trim.toDouble,a(2).trim.toDouble)).toDF()

filmDF.createOrReplaceTempView("film")

// filmDF.show()

}

### 查询

**def** tmdb\_statistics(\_res\_path:String):Unit=

{

**val** spark = SparkSession.builder().appName("load\_csv").master("local").getOrCreate()

**val** resDF=spark.sql("SELECT company\_name,SUM(revenue) FROM film WHERE rate>6.5 GROUP BY company\_name")

resDF.show()

// res.write.format("csv").save("res.csv")

// var dataFrameWithOnlyOneColumn = dataFrame.select(concat(res.columns).alias('data'))

// dataFrameWithOnlyOneColumn.coalesce(1).write.format("text").option("header", "false").mode("append").save("<path>")

**var** resRDD=resDF.rdd

**val** writer = **new** PrintWriter(**new** File(\_res\_path))

**for**(res\_line<-resRDD.collect())

{

// println(res\_line)

writer.write(res\_line.toString+"\n")

}

writer.close()

}