

计算机科学与技术学院 大数据分析实践 课程实验报告

实验题目：SPARK 实践		学号：202100202121
日期：2023. 12. 7	班级：数据 21	姓名：李芷墨
Email：1621737438@qq. com		
<p>实验目的：</p> <p>本实验旨在介绍学习者如何配置和运行 Apache Spark，以及如何使用 Spark 进行简单的数据处理和分析。实验将涵盖以下内容：</p> <ol style="list-style-type: none">1.安装和配置 Apache Spark。2.运行一个简单的 Spark 应用程序，以理解 Spark 的基本概念。3.使用 Spark 进行数据处理和分析。		
<p>实验软件和硬件环境：</p> <div></div>		
<p>实验原理和方法：</p> <p>配置好实验环境</p> <ol style="list-style-type: none">1.下载 Apache Spark 并解压缩它到您的计算机。2.设置 SPARK_HOME 和 JAVA_HOME 环境变量，以指向 Spark 和 Java 的安装路径。3.配置 Spark 集群，包括主节点和工作节点。4.启动 Spark 集群，并确保它正常工作。		

下载数据集: sales.csv

<https://pan.baidu.com/s/18eqffdkDTP8clbZpE8G1>

Ow?pwd=xg2p 提取码: xg2p

二、实验步骤：（不要求罗列完整源代码）BERT 实践

进入 spark 的 bin 目录下（/usr/local/spark-3.2.1/bin）

1. 创建一个新的 Spark 应用程序，使用 Python 编写。
2. 在应用程序中初始化 SparkSession 对象，它是 Spark 应用程序的入口点。
3. 加载示例数据集到 Spark 中。
4. 对数据进行一些简单的转换操作，如筛选、映射或聚合。
5. 运行 Spark 应用程序并查看结果。

Python 实现部分代码

```
import org.apache.spark.sql.*;

/**
/**
 * java版本的DataFrame的常用操作。
 * 通常将DataFrame 存储在一张临时表中，后面通过执行sql的方式执行操作。
 *
 * @author yangshaojun
 * #date 2019/4/3 9:03
 * @version 1.0
 */
public class DataFrameNormalOperation {
    public static void main(String[] args) {

        SparkSession spark = SparkSession
            .builder()
            .appName("DataFrameNormalOperation")
            .master("local")
            .getOrCreate();

        // 等同于 SQLContext对象 sc.read().json("../.json")以及 sc.read().load("../.json)
        Dataset<Row> rowDataset = spark.read().json("data/sparksql/people.json");

        // 1、打印DataFrame数据信息
        rowDataset.show();
        // 2、打印schema信息
        rowDataset.printSchema();
        // 3、打印某列信息
        rowDataset.select("name").show();//等同于rowDataset.select(rowDataset.col("name"))
        // 4、查询多列信息；并且对列使用表达式操作（df.col(columnName))
        rowDataset.select(rowDataset.col("name"), rowDataset.col("age").plus(1)).show();
```

对 sales.csv 处理:

1、日销量最多

修改核心代码

```
# 日最多
spark.sql('select Store, Customers+0 as M from global_temp.people where DayofWeek >=1 and DayofWeek <=5 orderby M desc').show()
```

没有对店铺进行 groupby, 导致店铺重复出现

```
+-----+-----+
|Store| Max.C|
+-----+-----+
| 817|7388.0|
| 262|5494.0|
| 262|5458.0|
1| 262|5387.0|
| 262|5297.0|
| 262|5192.0|
| 262|5152.0|
| 262|5145.0|
| 262|5132.0|
| 262|5112.0|
| 262|5106.0|
| 262|5090.0|
| 262|5069.0|
| 262|5063.0|
| 262|5034.0|
| 262|5028.0|
| 262|5024.0|
| 262|5014.0|
| 262|5013.0|
| 262|4989.0|
+-----+-----+
only showing top 20 rows
```

2、日店铺最多

```
# 日店铺最多
spark.sql('select Store, max(Customers+0) as M from global_temp.people where DayofWeek >=1 \
and DayofWeek <=5 group by Store orderby M desc').show()
```

```

+-----+-----+
|Store| Max.C|
+-----+-----+
| 817|7388.0|
| 262|5494.0|
|1114|4911.0|
| 586|4762.0|
| 733|4645.0|
| 251|4635.0|
| 769|4582.0|
| 756|4180.0|
| 562|4099.0|
| 336|3961.0|
| 335|3929.0|
|1097|3804.0|
| 259|3648.0|
| 320|3507.0|
| 580|3360.0|
| 467|3351.0|
| 513|3302.0|
| 827|3241.0|
| 983|3200.0|
| 470|3178.0|
+-----+-----+
only showing top 20 stores

```

Customers+0 的目的是为了确保在执行聚合函数 max 时，Customers 字段的值是以数值方式进行比较，而不是以字符串形式进行比较。

3、均店铺最多

```

# 日店铺最多
datasets<Row> = spark.sql('select Store, avg(Customers+0) as Afrom \
                             global_temp.people where DayofWeek >=1 and DayofWeek <=5 group by Store orderby A desc')

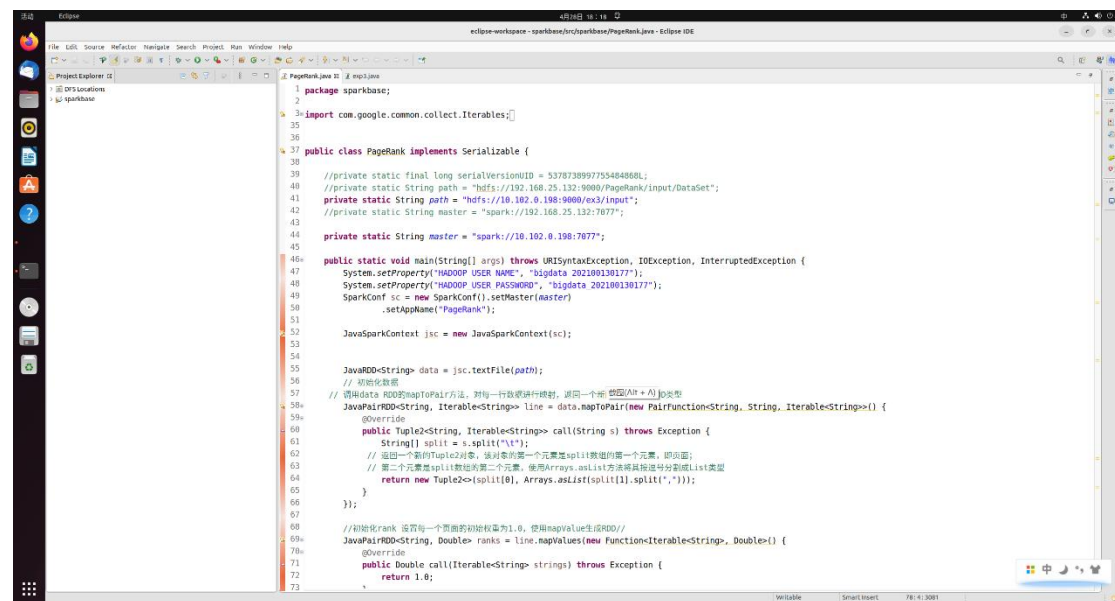
```

```
+-----+-----+
|Store| Avg.C|
+-----+-----+
| 733|3403.5|
| 262|3402.0|
| 562|3105.1|
1 | 769|3081.1|
| 1114|2664.1|
| 817|2605.5|
| 1097|2420.9|
| 335|2385.3|
| 259|2347.1|
| 251|2026.5|
| 586|1945.3|
| 756|1940.5|
| 423|1886.1|
| 383|1826.2|
| 682|1758.8|
| 513|1744.7|
| 320|1717.1|
| 948|1687.0|
| 676|1644.6|
| 336|1623.2|
+-----+-----+
only showing top 20 stores
```

结论分析与体会：

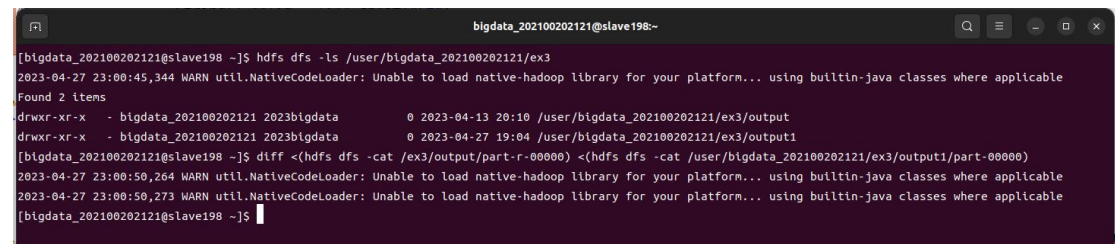
大二下大数据分析与实践就接触过 Spark，当时使用的是 java，故上手较快。

下附 word_count java 实现



```
1 package sparkbase;
2
3 import com.google.common.collect.Iterables;
4
5 public class PageRank implements Serializable {
6
7     //private static final long serialVersionUID = 537873899775484868L;
8     //private static String path = "hdfs://192.168.25.132:9000/PageRank/input/DataSet";
9     private static String path = "hdfs://10.102.0.198:9000/ex3/input";
10    //private static String master = "spark://192.168.25.132:7077";
11
12    private static String master = "spark://10.102.0.198:7077";
13
14    public static void main(String[] args) throws URISyntaxException, IOException, InterruptedException {
15        System.setProperty("HADOOP_USER_NAME", "bigdata_202100202121");
16        System.setProperty("HADOOP_USER_PASSWORD", "bigdata_202100202121");
17        SparkConf sc = new SparkConf().setMaster(master)
18            .setAppName("PageRank");
19
20        JavaSparkContext jsc = new JavaSparkContext(sc);
21
22        JavaRDD<String> data = jsc.textFile(path);
23        // 读取数据
24        // 调用data RDD的mapToPair方法，对每一行数据进行遍历，返回一个键-值对 (key + value) 类型
25        JavaPairRDD<String, Iterable<String>> line = data.mapToPair(new PairFunction<String, String, Iterable<String>>() {
26            @Override
27            public Tuple2<String, Iterable<String>> call(String s) throws Exception {
28                String[] split = s.split("\t");
29                // 返回一个新的Tuple2对象，该对象的第一个元素是split数组的第一个元素，即单词
30                // 第二个元素是split数组的第二个元素，使用Arrays.asList方法将数组转换为List类型
31                return new Tuple2<>(split[0], Arrays.asList(split[1].split(",")));
32            }
33        });
34
35        // 初始rank 设置每个页面的初始值为1.0，使用mapValues生成RDD
36        JavaPairRDD<String, Double> ranks = line.mapValues(new Function<Iterable<String>, Double>() {
37            @Override
38            public Double call(Iterable<String> strings) throws Exception {
39                return 1.0;
40            }
41        });
```

集群提交 diff 比较，无误。



```
bigdata_202100202121@slave198:~
[bigdata_202100202121@slave198 ~]$ hdfs dfs -ls /user/bigdata_202100202121/ex3
2023-04-27 23:00:45,344 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
drwxr-xr-x - bigdata_202100202121 2023bigdata 0 2023-04-13 20:10 /user/bigdata_202100202121/ex3/output
drwxr-xr-x - bigdata_202100202121 2023bigdata 0 2023-04-27 19:04 /user/bigdata_202100202121/ex3/output1
[bigdata_202100202121@slave198 ~]$ diff <(hdfs dfs -cat /ex3/output/part-r-000000) <(hdfs dfs -cat /user/bigdata_202100202121/ex3/output1/part-000000)
2023-04-27 23:00:50,264 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2023-04-27 23:00:50,273 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[bigdata_202100202121@slave198 ~]$
```