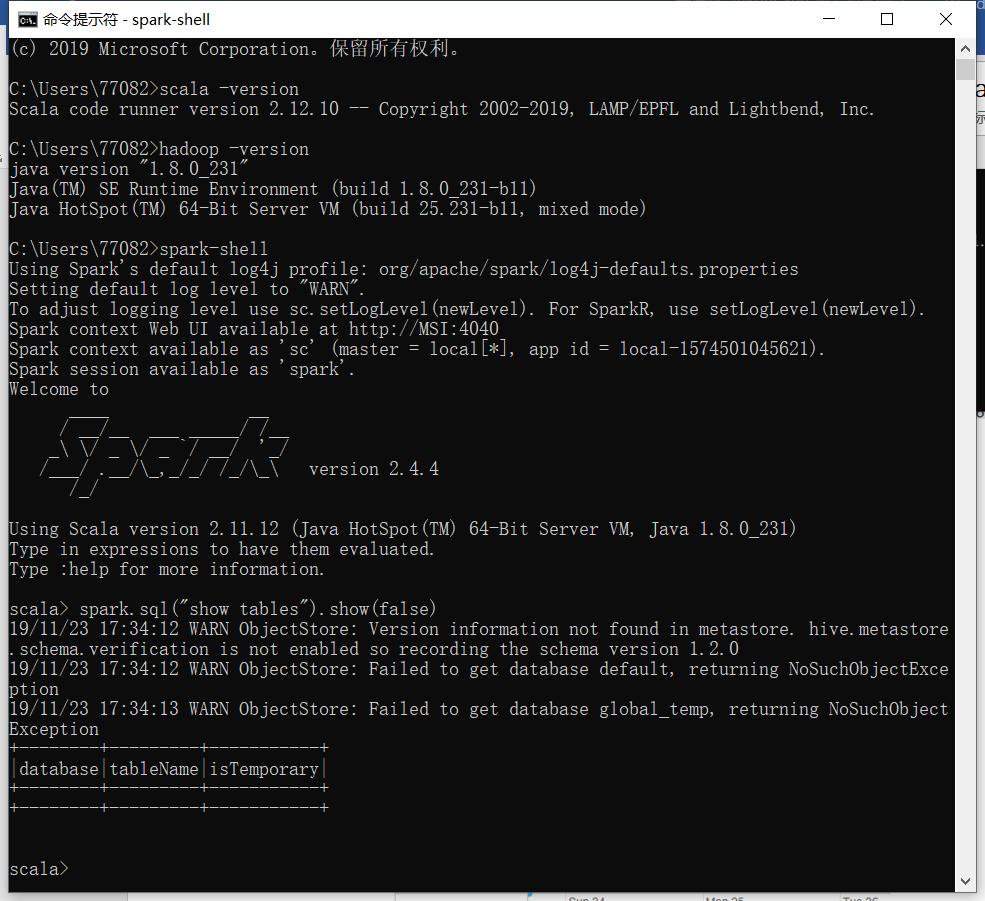
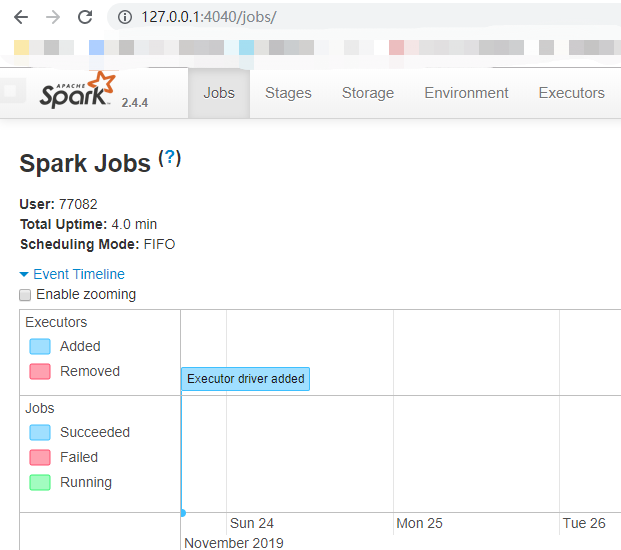
1. 首先根据网上教程搭建win10 + scala + spark + hadoop环境



登录网站<http://127.0.0.1:4040/jobs/>



环境搭建成功。

1. 配置IDEA环境
2. PageRank 是执行多次连接的一个迭代算法。 算法会维护两个数据集（在spark中为RDD）：

links：由 (pageID, linkList) 的元素组成，包含每个页面的相邻页面的列表。其中的一个元素例如:(A,[B,C,D]) 代表A中含指向B C D的链接

ranks：由 (pageID,PR) 元素组成，包含每个页面的当前排序值。它按如下步 骤进行计算。其中的一个元素例如（A,0.7）代表

(1) 将每个页面的排序值初始化为 1.0。

(2) 在每次迭代中，对页面 p，向其每个相邻页面（有直接链接的页面）发送一个值为PR(p)/L(p) 的贡献值。

(3) 将每个页面的排序值设为 0.15 + 0.85 \* contributionsReceived。

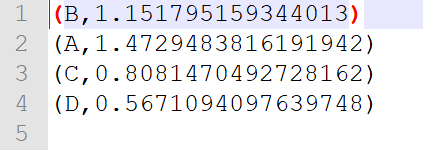
最后两步会重复几个循环，在此过程中，算法会逐渐收敛于每个页面的实际 PageRank 值。通常需要大约 10 轮迭代。

1. 实现源码如下：

算法思路如上所示。

import org.apache.spark.{HashPartitioner, SparkConf,SparkContext}  
object PageRank {  
 def main(args:Array[String]):Unit = {  
 val conf = new SparkConf().setAppName("PageRank").setMaster("local")  
 val sc = new SparkContext(conf)  
 var links = sc.parallelize(*List*(  
 ("A",*List*("B","C","D")),  
 ("B",*List*("A")),  
 ("C",*List*("A","B")),  
 ("D",*List*("B","C"))  
 ))  
 // 初始化 ranks中每个页面的PR值为1.0  
 var ranks = links.mapValues(v => 1.0)  
 for(i <- 0 until 10){  
 val contributions = links.join(ranks).flatMap{  
 case(pageID,(links,rank)) => links.map(link => (link , rank / links.size))  
 }  
 ranks = contributions.reduceByKey((x,y) => x+y).mapValues(v => 0.15 \* 1.0 + 0.85 \*v)  
 }  
 ranks.collect().foreach(*println*)  
 ranks.saveAsTextFile("result")  
 }  
}

1. 运行结果：保存在result文件夹下的part-000000文件



IDEA环境下的输出为:

