**Unit01—Spark基础**

1. Spark的基础运行模式：
2. Standalone，Spark自己集成的集群模式
3. 单机版，比如使用spark-shell 后面不加—master 就是单机版，也比如idea自己执行的情况
4. 基于yarn的，这个没有配置过
5. 基于Spark官方的metos没用过
6. Bin文件目录的下的使用
7. 安装完后可以使用.**/run-example org.apache.spark.examples.SparkPi 来检测是否可以跑成功。**
8. **Spark-shell 里他是已经给你创建好了一个SparkContext 所以直接使用sc.就可以如果还想自己创建可以导包 org.apache.spark.\_。 使用集群后面跟 –master**
9. **关于sc. Parallelize ,官方解释如下，简单说就是并行化集合，注意这里很关键的可以设置分区：**

并行集合是通过调用创建的 SparkContext 的 并行化 方法对现有的在你的驱动程序(一个Scala集合 Seq )。 集合的元素复制到形成一个分布式数据集,可以并行操作。 例如,下面是如何创建一个并行收集的数字1 - 5:

**val** data **=** **Array**(1, 2, 3, 4, 5)

**val** distData **=** sc.parallelize(data)

一旦创建,分布式数据集( distData )可以并行操作。 例如,我们可能会调用 distData。 减少((a,b)= > a + b) 添加元素的数组。 我们描述分布式数据集上的操作。

为并行集合是一个重要的参数的数量 分区减少数据集。 火花将运行一个任务为每个分区的集群。 通常需要2 - 4分区为集群中的每个CPU。 通常,火花试图设置分区的数量自动根据您的集群。 然而,你也可以手动设置它通过它作为第二个参数 并行化 (如。 sc.parallelize(数据,10) )。 注意:一些地方在代码中使用术语片(分区)的同义词来维持向后兼容性。

1. **关于RDD，官方给出的定义是弹性分布式数据集:**
2. **官方给的概念，**引发围绕的概念 弹性分布式数据集(抽样),这是一种容错的元素的集合可以在并行操作。 有两种方法可以创建抽样: 并行现有的收藏在你的驱动程序,或引用在外部存储系统的数据集,例如 共享的文件系统,HDFS,HBase或任何数据源提供Hadoop InputFormat。（ There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing(引用) a dataset in an external(外部的) storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.）
3. **RDD分两种，一种是lazy 懒加载，就是比如 定义好并不执行，必须调用collect才会执行，另外一种是直接执行。**
4. **最后做了一个统计超市一些信息的，小例子,注意里面用到了很多基础的东西，需要好好看：**

**数据.csv 的数据如下：**

|  |
| --- |
| John,iPhone Cover,9.99 John,Headphones,5.49 Jack,iPhone Cover,9.99 Jill,Samsung Galaxy Cover,8.95 Bob,iPad Cover,5.49 |

源码如下：

|  |
| --- |
| **//这里一定注意，我没有使用SparkConf来初始化，这里第一个参数是设置Master的,这里的集群化设置直接RUN不行必须打成JAR包，如果要本地跑设置成 local[4](4是线程数)**  **val** sc= **new** SparkContext("spark://hadoop-server03:7077","fuck") //读取csv数据，并且按行切分“，”，然后分成三个 用户 商品 单价 **val** data= sc.textFile("hdfs://hadoop-server03:9000/test.csv").map(line=>line.split(",")).map(purchaseRecord=>(purchaseRecord(0),purchaseRecord(1),purchaseRecord(2))) //计算总体销售数量 **val** numPurchase = data.count();  //计算有多少顾客，按姓名区别不能重复 **val** uniqueUsers = data.map{ **case** (user, product, price) => user }.distinct().count() //计算销售额 **val** totalRevenue = data.map{**case** (user, product, price)=> price.toDouble}.sum() //计算卖出最多的商品，这里需要注意的SortBy 加了一个-，因为不加是按从小到大的，加了后会从大到小 **val** productsByPopularity = data.map{ **case** (user, product, price)=> (product,1)}.reduceByKey(\_+\_).collect().sortBy(-\_.\_2) //直接取出最大的受欢迎的商品 **val** mostPopular = productsByPopularity(0) *println*("Total purchases: " + numPurchase) *println*("Unique users: " + uniqueUsers) *println*("Total revenue: " + totalRevenue) productsByPopularity.map(x=>*println*(x)) *println*("Most popular product: %s with %d purchases".  format(mostPopular.\_1, mostPopular.\_2)) |

**测试下git….**