大数据分析与应用

2020春

第2章 分布式时代的代码

- 分布式计算对编程范式的影响：函数式编程（functional programming）或称函数程序设计，又称泛函编程，是一种编程范型，它将电脑运算视为数学上的函数计算，并且避免使用程序状态以及易变对象。

  - 举例：计算向量平方和

```
public class AA {
 public static void main(String[] args) {
  int a[]={1,2,3,4,5};
  int sum=0;
  for(int i=0;i<a.length;i++){
   sum=sum+a[i]*a[i];
  }
  System.out.println("sum="+sum);
 }
}

运行结果
sum=42
```

  - 现有算法的基本假设：单个计算单元（CPU）；
  - for语句完成的任务：相同计算任务的重复；循环语句的任务可被拆分；

  - 两类最基本的循环任务：

1）在每个元素上分别完成的任务（map）

2）在上一次任务的结果上完成的任务（reduce）

第2章 分布式时代的代码——MapReduce

Google: MapReduce: Simplified Data Processing on Large Clusters

Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

```
>>> def f(x):
...     return x * x
...
>>> map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])
[1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> def add(x, y):
...     return x + y
...
>>> reduce(add, [1, 3, 5, 7, 9])
25
```

第2章 分布式时代的代码

Spark python编程实现：

```
data = [1,2,3,4,5]
distData = sc.parallelize(data)
Res =distData.map(lambda x:x*x).reduce(lambda a,b:a+b)

//distData.map(lambda x:x*x).collect()
```

- RDD（Resilient Distributed Dataset）叫做弹性分布式数据集，是Spark中最基本的数据抽象，它代表一个不可变、可分区、里面的元素可并行计算的集合。RDD具有数据流模型的特点：自动容错、位置感知性调度和可伸缩性。RDD允许用户在执行多个查询时显式地将工作集缓存在内存中，后续的查询能够重用工作集，这极大地提升了查询速度。
- Lambda表达式：匿名函数

第2章 分布式时代的代码

分支语句(if)的函数式编程

#选择奇数

```
def is_odd(n):
    return n % 2 == 1

filter(is_odd, [1, 2, 4, 5, 6, 9, 10, 15])

# 结果: [1, 5, 9, 15]
```

第2章 分布式时代的代码

编程泛型：

- 程序：按顺序排列的计算机指令；
- 结构化编程；
- 面向对象编程；
- 函数式编程；

函数式编程的优点

在函数式编程中，由于数据全部都是不可变的，所以没有并发编程的问题，是多线程安全的。可以有效降低程序运行中所产生的副作用，对于快速迭代的项目来说，函数式编程可以实现函数与函数之间的热切换而不用担心数据的问题，因为它是以函数作为最小单位的，只要函数与函数之间的关系正确即可保证结果的正确性。

函数式编程的表达方式更加符合人类日常生活中的语法，代码可读性更强。实现同样的功能函数式编程所需要的代码比面向对象编程要少很多，代码更加简洁明晰。函数式编程广泛运用于科学研究中，因为在科研中对于代码的工程化要求比较低，写起来更加简单，所以使用函数式编程开发的速度比用面向对象要高很多，如果是对开发速度要求较高但是对运行资源要求较低同时对速度要求较低的场景下使用函数式会更加高效。

函数式编程的缺点

由于所有的数据都是不可变的，所以所有的变量在程序运行期间都是一直存在的，非常占用运行资源。同时由于函数式的先天性设计导致性能一直不够。虽然现代的函数式编程语言使用了很多技巧比如惰性计算等来优化运行速度，但是始终无法与面向对象的程序相比，当然面向对象程序的速度也不够快。

函数式编程虽然已经诞生了很多年，但是至今为止在工程上想要大规模使用函数式编程仍然有很多待解决的问题，尤其是对于规模比较大的工程而言。如果对函数式编程的理解不够深刻就会导致跟面相对象一样晦涩难懂的局面。

第2章 分布式时代的代码

- 统计文档每个单词的个数

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

- Key-value 对

第2章 分布式时代的代码

- 使用随机算法计算Pi的值

```
from random import random
from operator import add
def f(_):
    x = random() * 2 - 1
    y = random() * 2 - 1
    return 1 if x ** 2 + y ** 2 <= 1 else 0

count = sc.parallelize(range(0, 10000)).map(f).reduce(add)
print("Pi is roughly %f" % (4.0 * count / 10000))
```

第2章 分布式时代的代码

不同语言的比较

```scala
//Scala
val textFile = sc.textFile("hdfs://...")
val counts = textFile.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

```python
#Python
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

```java
//JAVA
JavaRDD<String> textFile = sc.textFile("hdfs://...");
JavaPairRDD<String, Integer> counts = textFile
    .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
    .mapToPair(word -> new Tuple2<>(word, 1))
    .reduceByKey((a, b) -> a + b);
counts.saveAsTextFile("hdfs://...");
```