Spark

# Scala

## 函数式编程概念

### 无副作用

无状态修改, 值得修改更新都在函数内部, 不会影响到函数外部, 即只更新局部变量, 不更新全局变量和成员变量

### 引用透明性

对于相同的输入,总是的导向相同的输出

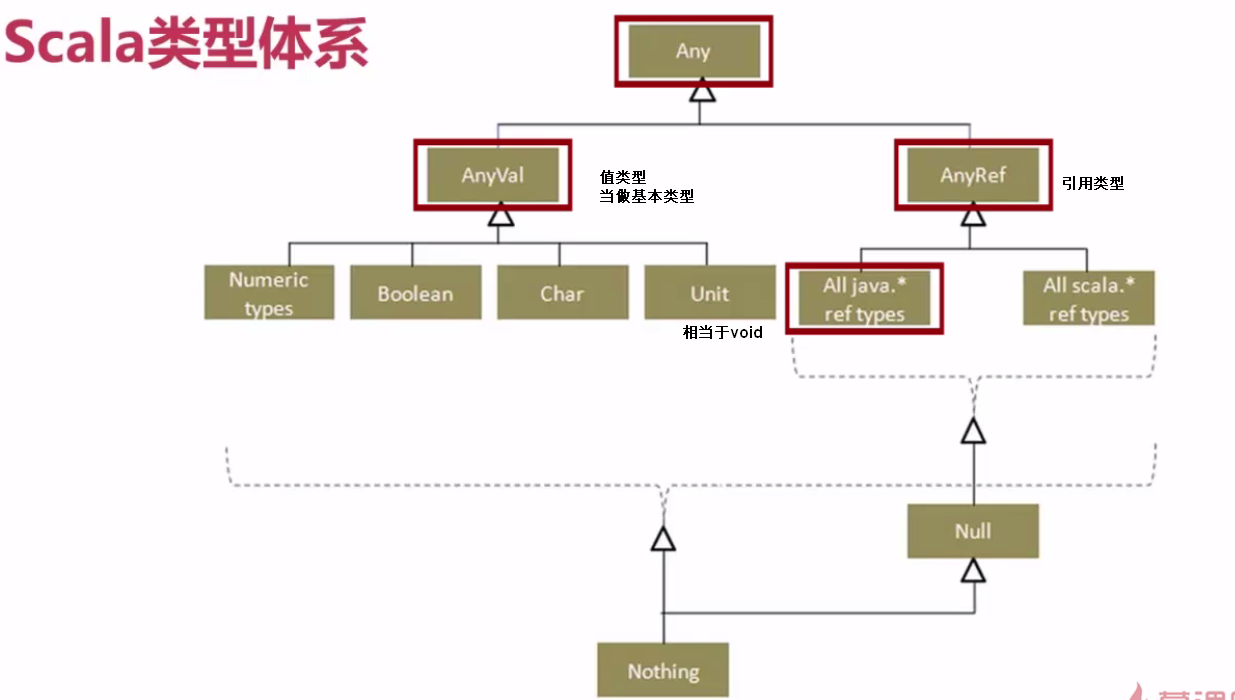
## val VS var VS lazy val

val : 变量值不可修改,一旦分配不能重新指向别的值

var : 分配后可以指向类型相同的值

lazy val : 延迟求值, 第一次使用到时才求值

## scala类型体系



unit 表示 void, 一般表明该函数是有副作用的

若抛出异常, 出现错误, 一般返回nothing类型

## 求值策略

### call by value

默认求值策略, 对函数实参求值, 且仅求值一次

如入参参数为 3+4 , 则先计算出7替换3+4

### call by name

函数实参每次在函数体内被用到时都会求值

如果函数形参以 => 开头, 则使用call by name

如入参为3+4, 则在使用到这个3+4的结果值时, 才计算3+4的结果

|  |
| --- |
| def foo(x:Int) = x //call by value  def foo(x: => Int) = x //call by name |

# Sprak

## Transformation

如map, filter, groupBy, join

lazy operation, 从其他RDDs建立RDD, 建立DAG(有向无环图)

## Action

执行一个DAG返回一个结果给driver或者将结果写入存储

### collect

遍历整个RDD, 向driver program返回RDD内容, 需要单机能够容纳下所有数据, 大数据时, 使用saveAsTextFile() action

### foreach

计算RDD中的每个元素, 但不返回到本地

## RDD

### 血统关系图

spark维护着RDDs之间的依赖关系和创建关系, 称为血统关系图, spark使用血统关系图来计算每个RDD的需求和恢复丢失的数据

### 延迟计算

spark对RDDs的计算时在第一次使用action操作时

### 持久化

默认每次在RDDs上进行action操作时(即真正执行了计算),spark都重新计算RDDs, 使用RDD.persist()可以保存RDD的计算结果, 直接复用, 而不用再次进行计算

unpersist()从缓存中移除RDD

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 级别 | 空间占用 | cpu消耗 | 在内存中 | 在硬盘上 | 备注 |
| MEMORY\_ONLY | high | low | y | n |  |
| MEMORY\_ONLY\_SER | low | high | y | n | 序列化 |
| DISK\_ONLY | low | high | n | y |  |
| MEMORY\_AND\_DISK | high | medium | some | some | 内存不足就放到硬盘 |
| MEMORY\_AND\_DISK\_SER | low | high | some | some | 内存不足就放到硬盘,内存中数据是序列化的 |

### combineByKey()

参数: createCombiner, mergeValue, mergeCombiners, partitions

最常用的基于key的聚合函数, 返回的类型可以与输入的类型不一样

遍历partition中的元素, 若为新元素, 则使用createCombiner(), 若partition已存在key, 则使用mergeValue函数

合并所有partition结果, 使用mergeCombiners()

eg: 求a,b均值

1.定义

|  |
| --- |
| val scores = sc.parallelize(Array(("a",88.0),("a",90.0),("a",89.0),("b",88.0),("b",89.0),("b",90.0))) |
| (a,90.0)  (a,89.0)  (b,88.0)  (a,88.0)  (b,89.0)  (b,90.0) |

2.分组求和

|  |
| --- |
| val score2 = scores.combineByKey(score=>(1,score),(c1:(Int,Double),newScore)=>(c1.\_1+1,c1.\_2+newScore),(c1:(Int,Double),c2:(Int,Double))=>(c1.\_1+c2.\_1,c1.\_2+c2.\_2)) |

createCombiner

score=>(1,score), 记录下分数及出现的次数, 如(1, 88.0)(1, 90.0)

mergeValue :

(c1:(Int,Double),newScore)=>(c1.\_1+1,c1.\_2+newScore)

根据key值, c1:(Int,Double)表示统计历史记录之和, newScore表示新遍历元素中的value值

c1:(Int,Double)应该和score=>(1,score)有对应关系

mergeCombiners:

(c1:(Int,Double),c2:(Int,Double))=>(c1.\_1+c2.\_1,c1.\_2+c2.\_2)

c1,c2代表各个分区中统计的值, 对各分区汇总

结果值

|  |
| --- |
| (b,(3,267.0))  (a,(3,267.0)) |

3.计算均值

|  |
| --- |
| val average = score2.map{case(name,(num,score))=>(name,score/num)} |

case(name,(num,score)) : 格式对应 (a,(3,267.0))

## 架构

### driver

driver进程， 提交spark程序的结点，不一定是master，启动后发送请求到master结点注册，driver之后可以获得executor信息，之后提交task到exetuor中

### master

进程，负责资源的调度分配，集群监控等

### worker

进程

* 在内存中储存RDD的某个/某些partition
* 启动其他进程（executor）处理RDD partition计算处理

### executor

进程，由worker启动，和worker在同一节点，executor启动多个task线程处理任务，executor启动后会向driver进行反注册，方便driver提交任务

### task

线程，由executor启动

# spark集群运行

## scala

### hdfs上传文件

|  |
| --- |
| hadoop fs –put fromfile tofile  eg:  hadoop fs –put spark.txt /spark.txt |

### 编写程序

### 导出jar包

export导出jar包

将jar包复制到spark服务器中

### 编写执行脚本

xxx.sh

|  |
| --- |
| /usr/local/spark/bin/spark-submit \  --class com.feng.spark.spark1.StructuredNetworkWordCount \  --master spark://spark1:7077 \  --num-executors 1 \  --driver-memory 500m \  --executor-memory 500m \  --executor-cores 1 \  /usr/local/test\_data/spark1-0.0.1-SNAPSHOT-jar-with-dependencies.jar \ |

如果不加

|  |
| --- |
| --master spark://spark1:7077 \ |

则默认是本地模式，单进程多线程执行

修改执行脚本权限

|  |
| --- |
| chmod 777 xxx.sh |

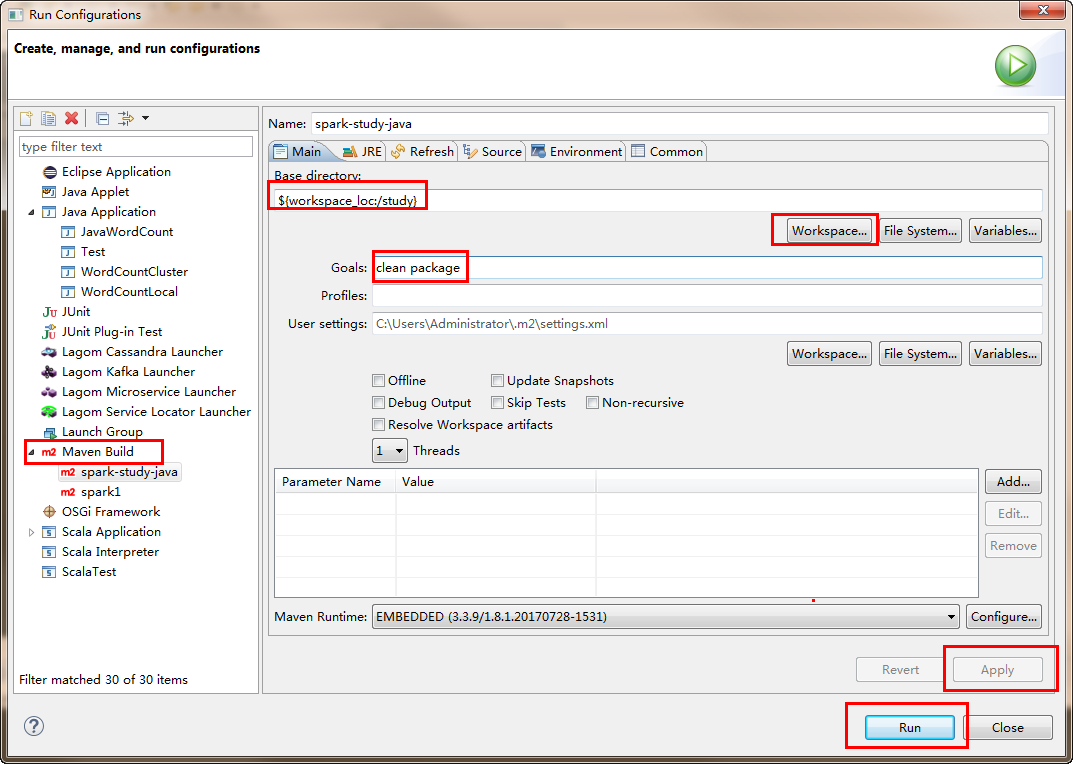
### 执行

|  |
| --- |
| ./xxx.sh |

## java

如scala，将导出jar包改为maven build

run as -> run configurations ->



再复制全有依赖的jar包到spark服务器