Spark2.1.1

1 使用mvn开发scala

<http://blog.csdn.net/kwu_ganymede/article/details/51832427>

cloudera文档：

<https://www.cloudera.com/documentation/enterprise/5-8-x/topics/spark.html>

翻译：

<http://www.cnblogs.com/BYRans/p/5292763.html#432>

# 一、Spark Programming Guide

## 1 Overview

1 每一个spark程序都是由driver进程（运行用户的main函数）和集群上并行执行的操作组成。

2 RDD是处理元素的集合，是一个逻辑概念。它是由分布在集群各个节点上的partition组成。Partition是spark并行的基础，每一个partiton对应一个task。

A RDD的生成可以从hadoop等外部文件系统； driver中scala collection也可以转换成RDD

B RDD 可以缓存在内存中，便于快速复用结果

C RDD 容错，失败恢复机制

D RDD 不可变

## 2 Linking with Spark

Spark 依赖scala，二者版本要对应。Spark2.1.1默认使用scala2.11.

如果访问hdfs，还需要依赖hadoop-client

## 3 Initializing Spark

## 4 Resilient Distributed Datasets (RDDs)

两种数据来源创造RDD

1 driver程序中，并行化scala容器:  parallelizing an existing collection

2 读取外部存储系统：hdfs 、hbase、es

### 1 Parallelized Collections

在driver进程中，使用sc.parallelize() 把数据转换成分布式的rdd。

非常重要的是：控制分片的数量，一个partition对应一个task，最好是核心数的2-3倍。

### 2 External Datasets

问题：都支持哪些数据源？用哪些api来读取数据源？怎么保存数据？

1 数据源：只要数据源是基于hdfs的，都支持

数据格式：text file、SequenceFile和其他的hadoop InputFormat

2 注意的地方：

1 如果是本地文件系统，则必须所有的节点都能访问这个路径。或者使用网络挂载的文件系统

2 所有读取文件的方法，包含textFile，支持传入目录，压缩文件，正则表达式匹配文件

3 textFile这个api，有第2个可选参数，控制partition的个数。默认情况下，一个block对应一个paritition（hdfs中128M为一个block），可以设置一个比默认的partition更大的数值，但是不能设置小的数值。

除了textFile，还有

1 以文件为单位，适用于小文件： SparkContext.wholeTextFiles

2 hadoop k/v序列化后的[SequenceFiles](http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/mapred/SequenceFileInputFormat.html) ： SparkContext’s sequenceFile[K, V]

3 For other Hadoop InputFormats, you can use the SparkContext.hadoopRDD method,

RDD Operations

保存，写到文件系统中去：

RDD.saveAsObjectFile and SparkContext.objectFile 以简单序列化的方式保存到文件。但序列化方式不如avro，最好自己定制输出。

### 3 RDD Operations

对Rdd有两种类型的操作：转换和行动，及各自的特点。

transformation: 从一种格式的RDD到另一种格式的RDD，如 map，reduceByKey  
action：一定是把对数据集的计算结果返回给driver，结果不再是rdd。 如 reduce

注意：reduce是action，但reduceByKey是transformation

特点：所有的transformation都是lazy的，只有action才会开始计算。

这样的好处： 返回给driver的结果会很小，而不是一个大的数据集合。

#### 1 Basics

理解这个示例的执行流程：

1 reduce的时候，每个节点会做一个本地的reduction，然后把结果发送到driver，再进行最终的聚合。

2 因此 lineLengths 这个rdd肯定不会被缓存，在reduce的时候已经使用了。如果需要复用，需要手动persist。

#### 2 Passing Functions to Spark

把一个执行函数从driver程序传递到集群中去执行：

1 使用匿名函数： map(\_ + \_)

2 使用单例（object）中的静态方法： Object中定义方法

3 定义一个类，类中定义函数和成员：这种是否可行？？？？

当使用类中的成员的时候，最好使用局部变量，如果使用对象调用，会使用对象的整个引用，耗费内存。

#### 3 Understanding closures

##### Local vs. cluster modes

如：foreach函数执行使用的是闭包，里面和外面是两个世界。里面是cluster并发执行的，而变量定义在外面driver中。这样执行的时候，每个Executor会拷贝一份数据，但已经不是外面数据本身了。

本质：是不是在同一个jvm中运行。

##### Printing elements of an RDD

上面是变量闭包，这里是函数闭包。如print

rdd.foreach(println) or rdd.map(println)

rdd会在每个Executor上执行，因此输出到executor自身的stdout上，driver并看不到。这就要求我们在Executor上的变量和动作，必须是无状态的。

解决方法：

1 rdd.collect().foreach(println) ： 把所有的rdd收集到driver中，再打印。

存在占用大量内存的问题。

2  rdd.take(100).foreach(println)： 打印一部分数据，而不是全部。

#### Working with Key-Value Pairs

1 对kv结构的数据操作是最常用的：如根据k对数据做分组(grouping)和聚合(aggregating)操作。

2 自动对scala的Tuple2类类型。Key value pair operation自动可用。

对kv数据类型操作，k的hashcode和equals方法很重要，一定要匹配。

1 grouping需要依赖k的hashCode

2 aggregating 需要依赖k的hashCode和equals方法

#### Transformations

#### Actions

1 注意foreach函数

2 countByKey是action

#### Shuffle operations

Shuffle对数据重新进行分组(分partition)，涉及到数据的跨网络、跨节点、跨executor、跨partition的数据拷贝，最耗费性能的操作。

##### Background

虽然shuffle之后，在每个partition上新创建的数据集是确定的，但是数据集里面的数据顺序是不确定的。

如果想让数据集有序:

1 mapPartitions +  .sorted : 对每一个parttion进行操作。

2 repartitionAndSortWithinPartitions ： 在reparation的时候，同时进行排序

3 sortBy ： 返回一个全局有序的RDD

可以引起shuffle操作的api：主要有三类

1 用来repartiton的： repartiton， [coalesce](http://spark.apache.org/docs/latest/programming-guide.html#CoalesceLink)

2 对kv的 **ByKey** operations : groupByKey, reduceByKey (countByKey 不是)

3 join操作：[cogroup](http://spark.apache.org/docs/latest/programming-guide.html#CogroupLink)、join

##### Performance Impact

##### 1 非常昂贵的操作：

CPU(对象序列化和反序列化)、内存、IO（网络IO和磁盘IO）

##### 2 内部的执行细节：

map中会把数据放内存，直到内存不足。每一个目标partition的数据会进行排序，并写到一个单独的文件。Reduce端去读取对应相应分区的文件。

##### 3 Shuffle为什么会消耗大量的堆内存

因为在转换数据前后，会使用基于内存的数据结构来组织数据。需要特别说明的是，reduceByKey和aggregateByKey在map阶段创造这些数据结构， ByKey操作在reduce阶段创建数据结构。当内存不足时，会把数据溢出到磁盘，导致额外的磁盘IO和GC的提高。

##### 4 shuffle产生的中间文件怎么处理

shuffle会产生大量的中间文件，只有在生成的rdd不再使用并被GC回收后，才会删除这些文件。

为什么？rdd出错，不用再根据血统关系重新计算这个rdd，相当于磁盘冗余。

磁盘目录配置：spark.local.dir

### 4 RDD Persistence

 cache() 和persist()

1什么时候要缓存RDD。

用于**复用**RDD，不再重新计算。

2 有哪些策略？

存储位置：memory, disk, offheap

对象方式：序列化和原始数据格式

存储份数：1 和 2（冗余）

3 在shuffle时，会自动persist一部分中间数据，避免因为一个节点的失败而重新计算所有的输入

4 怎样选择合适的策略？

1 如果内存足够，则使用MEMORY\_ONLY(这是默认的)，内存不足，则使用 MEMORY\_ONLY\_SER

2 不要序列化到磁盘，除非经过计算过滤了大量数据。因为序列化到磁盘，再从磁盘读的效率同重新计算的效率一样。

### Removing Data

1 系统会使用LRU策略删除缓存数据。

2 可以使用 RDD.unpersist() 显示删除。

## 5 Shared Variables

### 1 Broadcast Variables

1 为什么使用广播变量?

在Overview中，“it ships a copy of each variable used in the function to each task”,driver

中定义的变量，每个Task中执行的时候都会拷贝过来，如果变量占用的内存空间很大，每个task都会有，会有很大的冗余。

因为：我们可以 广播变量，在每个节点上缓存一份，而在这个节点上的所有的task都能共享。

2 广播变量不能被修改

3 在同一个stage的tasks，公共的数据会自动被以序列化的形式广播。因此，我们可以显式广播跨stage的变量。

### 2 Accumulators

Executor向Driver发出added的命令，但是不能读取Accumulators的内容，只要driver能够读取。（我觉得是用Actor的消息通信写的）

### 1 两种定义方式

1 spark 自己实现的：Long、Double

2 可以实现的自己的类型：继承[AccumulatorV2](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.util.AccumulatorV2)，重写方法

a、 reset

b、add

c、merge

### 2 只能在Actions中使用

只能在Actions中使用，不能在转换中使用。

为什么？

转换的task失败，会重新启动，这样没有任何意义。而Action只有成功后才会统计；而失败后，不会更新值。

# 二、SparkSQL

# Overview

1 同spark的区别是： 处理有结构的数据，并做了优化

2 执行引擎用的同一个，但是外面使用的语言可以多个。

# Datasets and DataFrames

Dataset概念：RDD

DataFrame 概念：数据有个结构

在scala中：相当于 Dataset[Row]

在java中：相当于 Dataset<Row>

# Getting Started

## Starting Point: SparkSession

先调用静态成员函数builder，生成一个Builder；

在Builder中配置各种属性

调用 getOrCreate方法，返回一个SparkSession对象。

## Creating DataFrames

DataFrame来源：

1 RDD转换而来，引入隐式转换的声明

2 外部文件：读取文件的api都在DataFrameReader中

1 json文件: read.json()

2 text文件:

3 csv文件

## Untyped Dataset Operations

所有的操作都在DataSet中，DataFrame是DataSet[Row]的别名

1 select: select()

2 where: filter()

3 group: groupBy()

4 count(\*) : count

**Running SQL Queries Programmatically**

使用程序中+sql语句的方式，就像查询一个表。需要把df转换成一个临时的视图。

1 createOrReplaceTempView(“”) //创建一个临时表

2 spark.sql("SELECT \* FROM people") //执行sql语句

## Global Temporary View

临时视图是session相关的，以session为生命周期。

要想跨session共享，并直至程序结束，可以创建一个全局的view。

Api：

df.createGlobalTempView("people") //创建一个全局的临时表

## Creating Datasets

DataSet还有一套类似DataFrame的api。

Dataset和Rdd的区别：

1 序列化方式不一样

Rdd是用传统的序列化方式（kryo或其他）将其转换为字节

Dataset虽然也是转换成为字节，但是是有结构的，encoders是动态生成的，支持一系列对数据的操作。

## Interoperating with RDDs

将rdd转换为Dataset有两种方式:

1 使用反射的方式推断schema

定义一个case class

2 程序中动态生成schema

缺点：复杂

优点：不用提前建立，等知道数据后，根据数据来动态生成schema

# Data Sources

### Manually Specifying Options

写入文件：指明写入的格式和名称

write.format("parquet").save("namesAndAges.parquet")

### Save Modes

写并不是原子的也没使用锁，是线程不安全的。

指明写的模式

### Saving to Persistent Tables

可以直接写到hive中，并生成元数据文件。

## Parquet Files

### Loading Data Programmatically

### Partition Discovery

## JSON Datasets

## JDBC To Other Databases

1 直接通过jdbc读取数据库中的内容

# Performance Tuning

## Caching Data In Memory

像rdd一样，把表缓存到内存中。会根据配置自动压缩。

Api:

spark.cacheTable("tableName") or dataFrame.cache()

也可以从内存中删除这个表

## Other Configuration Options

还要其他配置用于调优。

# Distributed SQL Engine

# 其他

# 1 RDD、DataFrame和DataSet的区别

<http://www.jianshu.com/p/c0181667daa0>

<https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html>

RDD与DataFrame区别：

1 RDD: 中的数据是不可变的。操作后对对象的复制

DataFrame 中的内容是可变的，对原对象进行操作。

2 框架能知道DataFrame的结构，但是不止RDD中每个元素的。

DataFrame和DataSet的区别和联系

1

什么时候使用RDD？

1 使用low-level的api：转换和行为

2 数据没有结构化，即没有特定的列名字

3 想使用rdd的API而不是领域专用语言

4 不使用schema，使用列名访问数据

5 放弃性能优化

RDD被废弃了吗？

没有，当使用底层api事。 三者之间可以无缝切换，DataFrame和DataSet是在RDD之上的。

## DataFrames

1 也是不可变的数据，但是有结构。就像一张表。

因此，可以使用领域专用语言（sql）

2 DataFrame和DataSet api进行了合并。两种api： 有类型的和无类型的。

## Datasets

1 Datasets 是强类型的，

## Benefits of Dataset APIs

2

<https://databricks.com/blog/2015/07/15/introducing-window-functions-in-spark-sql.html>

<http://stackoverflow.com/questions/33655467/get-topn-of-all-groups-after-group-by-using-spark-dataframe>

源码

<https://my.oschina.net/corleone/blog?sort=time&p=3&temp=1490693518474>

# 三、API

<http://www.jianshu.com/p/b3caf825a4b9>

<http://blog.csdn.net/u012102306/article/details/51762294>

## 1、flatMapValues

1首先是对Pari(k,v)的操作-》是对v的操作

2 v原来是一个值，经过操作后，v变成一个数组

3 k只有一个，flat操作相当于 k join (v1, v2, v3...)

4 最终是：（k,v1），（k,v2）,(k,v3)….(k2,v….)

## 2、repartition 和 coalesce

<http://www.cnblogs.com/fillPv/p/5392186.html>

## 3 countByKey

为什么不进行shuffle

## 4 sortBy

怎么返回一个全局有序的RDD？

# 四、shuffle

<https://www.iteblog.com/archives/1672.html>

# 五、Spark Streaming

# Overview

工作原理：

input stream ->>Spark Streaming

-》按照时间间隔划分批次

-》每批次转换为一个RDD

-》spark engine对这个rdd进行处理。

重要的抽象概念:

 Internally, a DStream is represented as a sequence of [RDDs](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.rdd.RDD)

# A Quick Example

1、在生成 StreamingContext对象的时候，传入interval

2、启动方式，不同于离线处理的action，而是streaming.start

3、一直阻塞driver进程

# Basic Concepts

## Initializing StreamingContext

**val** ssc **=** **new** **StreamingContext**(conf, **Seconds**(1))

1 在context中，有个batch interval 参数，流入的数据根据多长时间划分为一个批次，即把这批的数据转换成一个RDD。

2 可以使用一个已经存在sparkContext 来构造一个StreamingContext

3 sparkContext和streamingContext的区别和联系：

## Discretized Streams (DStreams)

1 a DStream is represented by a continuous series of RDDs

2 Each RDD in a DStream contains data from a certain interval

3 我们对流的处理本质上对**这一时间间隔内的rdd进行操作**

## Input DStreams and Receivers

**一个Input DStream 对应一个Receiver，每一个Receiver对应一个cpu和一个线程**，因此Receiver消耗cpu资源，不管是本地环境还是线上环境，设置的cpu个数一定要大于Receiver个数。

有两种source：

1 BasicSources

a 读取网络文本

b 读取文件目录： spark会监控目录，只能新增文件，不能追加文件。

2 Advanced Sources

读取消息中间件，需要依赖额外包

### Receiver Reliability

Receiver的可靠性：区别在于当spark读取到消息后，是否发确认消息，来保证遇到任何失败后，数据不会丢失。

## Transformations on DStreams

#### UpdateStateByKey Operation

统计从开始到现在的状态

#### Transform Operation

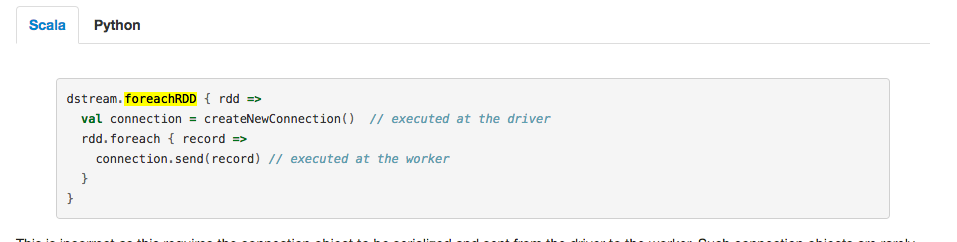
#### Window Operations

#### Join Operations

## Output Operations on DStreams

### Design Patterns for using foreachRDD

**Rdd是跨节点的；partition是在一个机器上。**

****

**疑问：如果connection在rdd中生成，是否会成功？**

把spark中的数据写到外部的系统中。

如果里面调用driver初始化的变量，必须能序列化，并跨机器传输。但jdbc连接是不能的。

foreachRDD同foreachPartiton结合起来使用。

注意要点：

1 默认，一个时间只能执行一个输出动作。如果有多个，会按照程序中的顺序执行。

### 通过连接池，rdd写入mysql中：

参考：<http://www.cnblogs.com/xlturing/p/spark.html>

## Caching / Persistence

1 同RDD一样，DStreams 也能显式persist和cache。

2 对于窗口函数(ByWindow)和状态更新函数，源码中默认就是true

3 同RDD 不一样，DStream默认持久化级别是 MEMORY\_ONLY\_SER

4 Reciver如果是接收网络传输的数据，持久化会冗余到两个节点（两份）

## Checkpointing

为什么需要checkpoint？

Streaming应用程序需要24\*7不中断执行，如果不是因为程序逻辑问题，而是系统错误，jvm崩溃导致的错误怎么办？

因此需要checkpoint必要的信息来容错，从而能做到**失败恢复（满血复活）** 。

两种类型的数据需要checkpoint：

*1 Metadata checkpointing*

把定义streaming的计算信息保存到存储中。用于driver节点的失败恢复。

配置信息：

Dstream操作信息：

未完成的批次信息

2 *Data checkpointing*

中间生成的Rdd，切断rdd的依赖链。

#### When to enable Checkpointing

1 使用状态转换的函数：跨批的

2 driver的错误恢复

#### How to configure Checkpointing

1 分为代码层面和部署层面

Driver错误，代码层面如何实现容错?

根据checkpoint目录重StreamingContext

除了代码层面实现，还可以在部署层面实现

2 checkpoint周期的设置

1 对于状态转换的函数，默认会checkpoint，周期为batch interval的倍数。

2 自己设置checkpoint周期，不能太小，会严重降低吞吐率；

不能太大，万一发生错误，需要根据血统关系重新计算，任务太多。

一般为batch interval的5-10倍。

## Accumulators, Broadcast Variables, and Checkpoints

累加器和广播变量如果使用一般的方法定义，失败重启时无法恢复的。

需要把他们定义在懒加载，单例对象中。

## Deploying Applications

# Performance Tuning

## Reducing the Batch Processing Times

# 五、Tuning Spark

## 1 Data Serialization

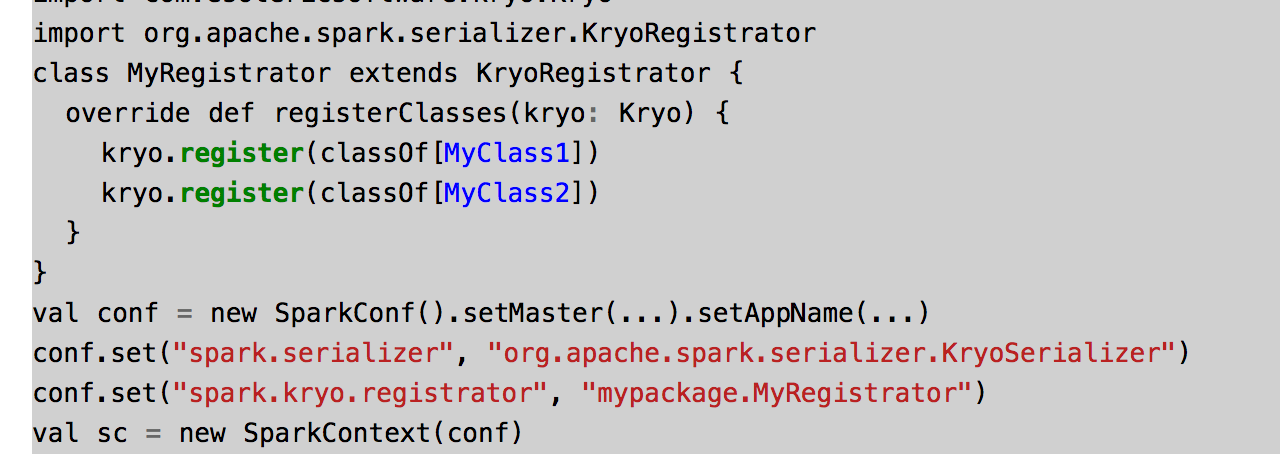
分布式，肯定涉及到数据的网络传输，则必须调优对象的序列化方式。

1 conf.set(**"spark.serializer"**,**"org.apache.spark.serializer.KryoSerializer"**)

2 将自定义的类进行注册

conf.registerKryoClasses(**Array**(classOf[MyClass1], classOf[MyClass2]))

或者使用这种注册方式：



## 2 Memory Tuning

spark将内存分为两种：

1 execution 执行内存： shuffles， sorts, joins, aggregations

2 storage 存储内存: cache persist

设计思想：

1 execution和storage使用一个统一的region。

2 当没有execution时，storage 能使用全部的内存

3 execution内存不足时， 可以驱逐storage内存，但直到一定的比例。

4 storage 不能驱逐execution。

参数配置：涉及到两个参数，一般情况下，默认值适用于大多程序。

1 spark.memory.fraction： M占用jvm heap内存的百分比。默认0.6.

2 spark.memory.storageFraction：

## Determining Memory Consumption

1 使用cache，查看日志

2 SizeEstimator’s estimate 方法

## Tuning Data Structures

原则：1 能使用基本类型就使用基本类型

2 能使用数组，不使用List等容器

3 如果需要容器中存储基本类型，可使用第三方库。

1 使用数组代替java的容器；使用基本类型替换引用。

可以使用第三库的容器：如 [fastutil](http://fastutil.di.unimi.it/) 。支持存储内容为 **基本类型**。

2 避免很多嵌套对象

3 使用枚举和数值 替换 keys

4 如果小于32G内存，设置jvm  -XX:+UseCompressedOops to make pointers be four bytes instead of eight。可在[spark-env.sh](http://spark.apache.org/docs/latest/configuration.html#environment-variables)中添加。

## Serialized RDD Storage

当缓存rdd的时候，如果内存吃紧，可以先对对象序列化再缓存到内存。（参考：rdd persist的策略）

## Garbage Collection Tuning

1 spark LRU 清理内存的策略， 要 查看所有的对象，并根据LRU策略清理内存。=》对象越多，清理越慢=》要减少对象的数量。

2 如果GC压力很大，可以设置缓存策略：rdd先序列化后再缓存，这样每个rdd partition就会序列化成一个字节数组。

**Measuring the Impact of GC**

先通过打印GC日志，查看jvm 垃圾收集的状态：

-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps

**Advanced GC Tuning**

**通过日志看minor gc和full gc的频率是不是在一个合适的比例：**

**1 如果minor gc频繁，**

**1扩大**Young generation ， 调整年轻代和年老代的比例。

2 增加总内存

2 如果在一个task中，full gc频繁，则内存不足。

1 增加总内存量

2 扩大年老代，调整年轻代和年老代的比例。

3 因为spark的executor内存一部分，调整spark.memory.fraction参数

4 使用G1垃圾收集器， -XX:+UseG1GC

# Other Considerations

## Level of Parallelism

1 最大限度的利用CPU并行处理任务

2 配置属性spark.default.parallelism ，更改默认最大的并行数

## Memory Usage of Reduce Tasks

如果因为shuffle导致的OOM，可以增加并行数，减少每个task的数据量。

## Broadcasting Large Variables

广播大的变量

在driver上定义的变量，会分发到每个task中，如果变量序列化后太大，会耗费内存资源和性能。可以将变量广播到每个节点上，每个节点上所有的task共享一份数据。

## Data Locality

数据本地行。数据和执行代码最优在一个JVM中。

# 六、spark数据入es

参考：

<http://blog.csdn.net/u010454030/article/details/71216054>

注意： 配置的端口是9200

# 七、SparkStreaming API

## 1 **updateStateByKey**

用法：从程序刚一开始执行，有流，然后到现在不断更新的累计状态

<http://www.jianshu.com/p/261636f397b8>

1 因为是随着批次不断更新，为了容错，肯定有checkpoint路径

2 函数参数的设计思想： 本批次数据，同一个key的序列，怎么计算？

本批计算结果和以前的状态，怎么计算？

# 八、好的网址

1 里面很多实战的例子

<http://blog.csdn.net/qq_21234493/article/details/51417548>

<https://kevinhjk.gitbooks.io/spark-src-study01/content/di_1_ke.html>

# 九、源码分析

## 1 入门：worldcount

<http://www.jianshu.com/p/f5b50e233acc>

任务的执行：

1 通过动作调用 runJob ：抽象出一个 执行函数和对对每一个分区结果的处理resultHandler。

2 调用dagScheduler.runJob来执行作业，返回一个JobWaiter，来等待任务结束。

3 给自己发送一个*JobSubmitted*的消息来执行任务

4 dagScheduler 进行handleJobSubmitted 处理：

a 先生成最后一个stage：ResultStage， 要生成这个Stage，需要知道父Stage才能构造。

b 于是调用getOrCreateParentStages

查找父stage，其实就是rdd根据它的依赖一直向上迭代，直到是shuffleDependency, 然后把shuffleDependcy封装成ShuffleMapStage

调用getOrCreateShuffleMapStage

## 2 spark 集群的Master、woker

1 <https://wongxingjun.github.io/2016/12/08/Spark-RPC%E8%A7%A3%E8%AF%BB/>

## 3 spark rpc

1 [http://www.jianshu.com/p/5 08ea1085068](http://www.jianshu.com/p/508ea1085068)

2 <https://github.com/ColZer/DigAndBuried/blob/master/spark/spark-network-netty.md>

3 http://www.codexiu.cn/spark/blog/15040/

RpcEnv、RpcEndpoint 及 RpcEndpointRef

RpcEnv类：是通信的Netty环境，相当于AKKA中的ActorSystem

1持有RpcEndpoint(如Master和Worker作为通讯的双方，向RpcEnv中注册)，

2支持对RpcEndpoint的查询：返回的是RpcEndpointRef

3 转换中心：通过 endpointRef()方法，把RpEendpoint 封装成RpcEndpointRef

RpcEndpoint类：相当于Netty的Handler或Akka的Actor

1 封装了对各种事件的处理和各种回调函数，如Woker和Master都继承RpcEndpoint。

## 4 spark storage

1 <http://www.jianshu.com/p/730eed6a98d2>

2 <http://www.jianshu.com/p/356db9726d04>

StorageManager 也是主从结构的，主在driver上，从在executor上。

其依赖应用程序存在。

## 5 executor模块

1 <http://www.jianshu.com/p/5dab83e94cac>

1 涉及的类：

1、StandaloneAppClient：

应用程序的客户端类，主要封装了一个ClientEndpoint对象，通过这个对象，同集群的master进行交互。

a 注册应用程序

运行应用程序，需要配置模式参数：是使用local、standalone、mesos、还是yarn。它们对应启动任务执行者进程。

进程是需要依赖执行应用程序的。因此在SparkContext中根据配置，对相应的成员变量进行初始化。

1 在SparkContext构造函数中，SparkContext.createTaskScheduler: 根据配置的master，创建TaskSchedulerImpl 实例 和 SchedulerBackend(如： StandaloneSchedulerBackend)

2 taskScheduler调用initialize方法，并调用start方法。start方法中会启动backend.start() 方法。

3 StandaloneSchedulerBackend继承自CoarseGrainedSchedulerBackend，先调用父类的start方法。

A 在start方法中，根据rpcEnv，创建DriverEndpoint，并在env中进行注册，然后赋值给成员driverEndpoint

B

## 6 cache和checkpoint

<https://vimsky.com/article/949.html>

## 7 SparkContext 源码

<http://www.jianshu.com/p/f7ffcceb415c>

## 8 DAGScheduler

1 <http://www.jianshu.com/p/3d2202105315>

2 <http://www.jianshu.com/p/e60e15a35b54>

3<http://coolplayer.net/2016/12/15/spark-%E6%BA%90%E7%A0%81%E5%89%96%E6%9E%90%E4%B9%8BDAG%E8%B0%83%E5%BA%A6/>

4 <http://www.jianshu.com/p/9323c83c7018>

## 9 RDD

1 <http://www.jianshu.com/p/54b3a4e786d9>

2 <http://www.jianshu.com/p/207607888767>

3 <http://www.jianshu.com/p/6b9e4001723d>

## 10 HashPartitioner和RangePartitoner

<https://www.iteblog.com/archives/1522.html>

## 4 spark shuffle

<https://wongxingjun.github.io/2016/05/27/Spark%E6%9E%B6%E6%9E%84%E4%B9%8Bshuffle/>

必读：原理

<https://github.com/JerryLead/SparkInternals/tree/master/markdown>

<https://github.com/jacksu/utils4s/tree/master/spark-knowledge/md>

<http://coolplayer.net/>

<https://ihainan.gitbooks.io/spark-source-code/content/section3/index.html>

<https://www.ibm.com/developerworks/cn/analytics/library/ba-cn-apache-spark-memory-management/index.html>

<http://www.jianshu.com/u/001d44710e2e>

<https://patrickhuang888.gitbooks.io/spark-with-scala/content/appendonlymap.html>

<http://www.jianshu.com/u/033a1d1f0c58>

必读：

<https://github.com/ColZer/DigAndBuried/tree/master/spark>

数据清理：

<http://zhou-yuefei.iteye.com/blog/2308596>

<http://developer.51cto.com/art/201605/511024.htm>

### 2 [RDD中WithScope是什么？](http://blog.csdn.net/legotime/article/details/51289351)

<http://blog.csdn.net/legotime/article/details/51289351>

3 Streaming 源码

必读：

<https://github.com/lw-lin/CoolplaySpark/tree/master/Spark%20Streaming%20%E6%BA%90%E7%A0%81%E8%A7%A3%E6%9E%90%E7%B3%BB%E5%88%97>

<http://www.jianshu.com/u/2bd9b48f6ea8>

<http://tech.youmi.net/2016/08/168828815.html>

<http://www.jianshu.com/p/4f2978040d0e>

<https://www.gitbook.com/@ihainan>

# 九、专题讲解

## 1 shuffle

1 <http://www.jianshu.com/p/4c5c2e535da5>

2 <https://github.com/JerryLead/SparkInternals/blob/master/markdown/4-shuffleDetails.md>

3 <http://smilekevinsovi.github.io/julyhou24/spark/2016/07/05/Spark-Shuffle.html>

<http://www.jianshu.com/p/c83bb237caa8>

<http://www.jianshu.com/p/d328c96aebfd>

<https://github.com/jacksu/utils4s/blob/master/spark-knowledge/md/spark%E5%86%85%E5%AD%98%E6%A6%82%E8%BF%B0.md>