Spark

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>

# 一、Spark Programming Guide

## 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？

3 怎样选择合适的策略？

1 如果内存足够，则使用MEMORY\_ONLY，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。

# Shared Variables

## 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。

## Accumulators

# 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….)

# 四、Spark Streaming

原理：

数据流

-》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。

# 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)

## Input DStreams and Receivers

#### Window Operations

## 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

2 driver的错误恢复

#### How to configure Checkpointing

## Input DStreams and Receivers

 built-in streaming sources 分为两种

1 Basic sources： file systems, and socket connections

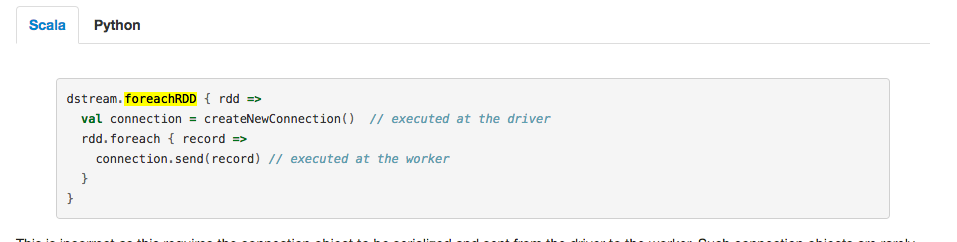
2 Advanced sources： 读取消息中间件，需要依赖额外包

### Advanced Sources

## 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>

# Performance Tuning

## Reducing the Batch Processing Times

# 五、Tuning Spark

## 1 Data Serialization

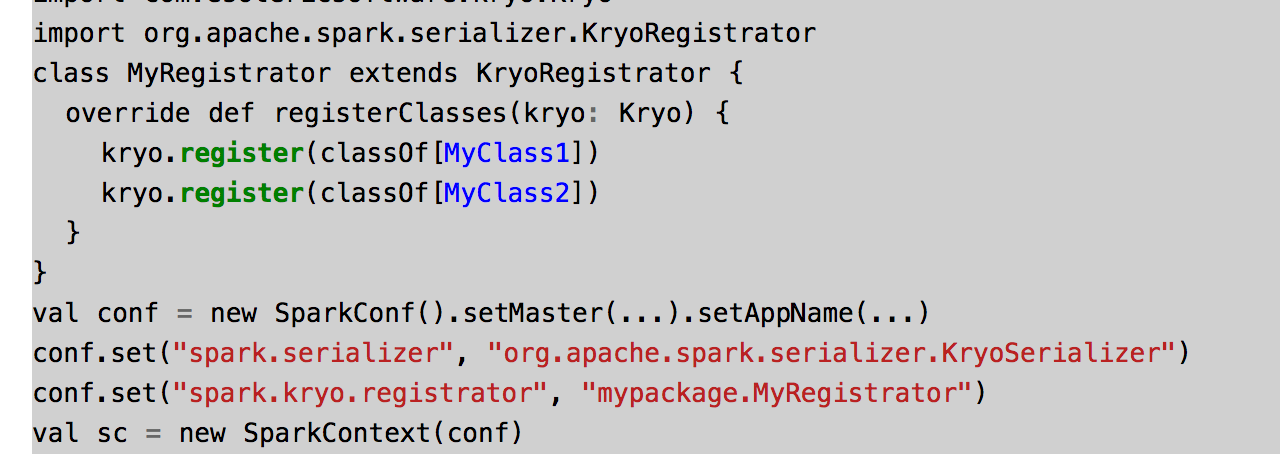
分布式，肯定涉及到数据的网络传输，则必须调优对象的序列化方式。

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" \l "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