**spark的安装和使用**

spark的依赖环境比较多，需要Java JDK、hadoop的支持。我们就分步骤依次介绍各个依赖的安装和配置。spark2.1.0运行在Java 7+, Python 2.6+/3.4+ , R3.1+平台下，如果是使用scala语言，需要 Scala2.11.x版本，hadoop最好安装2.6以上版本。 由于spark本身是用scala实现的，所以建议使用scala，本文中的示例也大多是scala语言。当然spark也可以很好地支持java\python\R语言。   
spark的使用有这么几类：spark shell交互，spark SQL和DataFrames，spark streaming, 独立应用程序。   
*注意，spark的使用部分，不特殊说明，都是以hadoop用户登录操作的。*

**1.安装Java环境**（以下步骤跳过，在对应的文档中已安装）

我的centos6.5安装系统的时候选择了安装openJDK的环境，所以可以直接使用。但这里还是列出jdk的安装步骤供大家参考。java环境可以使用Oracle的jdk或者openjdk. 下面的步骤是openjdk的安装示范。

**a.首先检查是否安装了jdk, 和版本是否符合要求。**

java –version

若安装了java环境，但是版本太低，则先卸载原版本，再安装新版本。   
卸载可参考以下步骤

yum -y remove java-1.7.0-openjdk\*

yum -y remove tzdata-java.noarch

**b.若未安装或已卸载，安装新版本**   
查看可用版本

yum -y list java\*

以安装1.7版本为例

yum -y install java-1.7.0-openjdk\*

**c.配置环境变量**

vi /etc/profile

在文件的最后添加

JAVA\_HOME=/usr/lib/jvm/java-1.7.0-openjdk

PATH=$JAVA\_HOME/bin

CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar

export JAVA\_HOME

export PATH

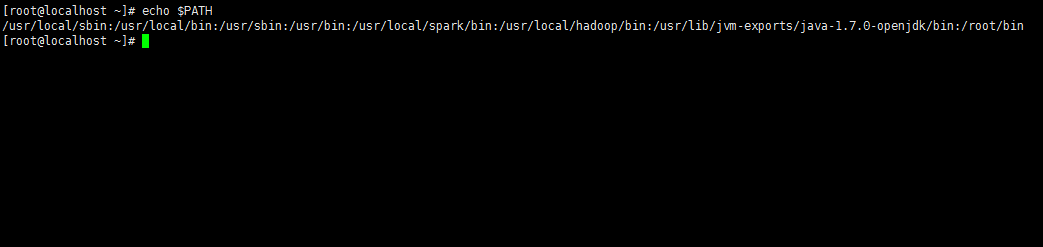
export CLASSPATH

其中JAVA\_HOME是你的java安装路径。其中PATH这个参数是以冒号：来分割不同的项的，后面我们hadoop和spark的环境变量配置也要修改这个参数。   
保存退出后，*还需要执行*

source /etc/profile

该文件才可以生效。检查环境变量是否配置生效

echo $PATH



**2.安装hadoop**（以下步骤跳过，在对应的文档中已安装）

如果你安装 CentOS 的时候不是用的 “hadoop” 用户，那么需要增加一个名为 hadoop 的用户。

**a.以root身份登录，添加”hadoop”用户**

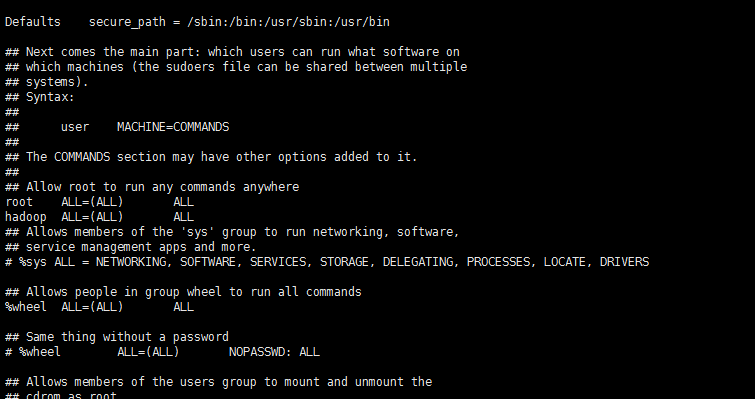
useradd -m hadoop -s /bin/bash

密码输入两次，笔者使用”hadoop”作为密码，比较好记忆。这样，一个用户名为hadoop, 密码也是hadoop的用户就添加好了。

**b.可为 hadoop 用户增加管理员权限，方便部署，避免一些对新手来说比较棘手的权限问题**

visudo

找到 root ALL=(ALL) ALL 这行（应该在第98行，可以先按一下键盘上的 ESC 键，然后输入 :98 (按一下冒号，接着输入98，再按回车键)，可以直接跳到第98行 ），然后在这行下面增加一行内容：hadoop ALL=(ALL) ALL （当中的间隔为tab），如下图所示：

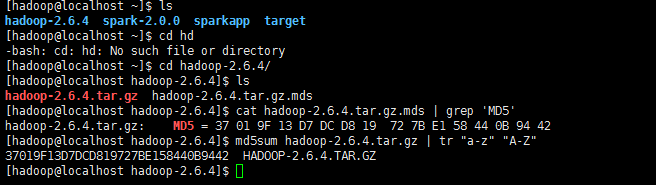


**c.centos默认安装ssh. 如果你的操作系统中没有ssh, 可以自行安装，最后的参考资料中有ssh的安装和配置。**

**d.安装hadoop**   
去官网下载hadoop的安装包，下载时请下载 hadoop-2.x.y.tar.gz 这个格式的文件，这是编译好的，另一个包含 src 的则是 Hadoop 源代码，需要进行编译才可使用。   
下载时强烈建议也下载 hadoop-2.x.y.tar.gz.mds 这个文件，该文件包含了检验值可用于检查 hadoop-2.x.y.tar.gz 的完整性，否则若文件发生了损坏或下载不完整，Hadoop 将无法正常运行。   
校验一下下载文件是否完整

cat hadoop-2.6.4.tar.gz.mds | grep 'MD5' # 列出md5检验值

md5sum hadoop-2.6.4.tar.gz | tr "a-z" "A-Z"

  
我们选择将 Hadoop 安装至 /usr/local/ 中：

sudo tar -zxf hadoop-2.6.4.tar.gz -C /usr/local # 解压到/usr/local中

cd /usr/local/

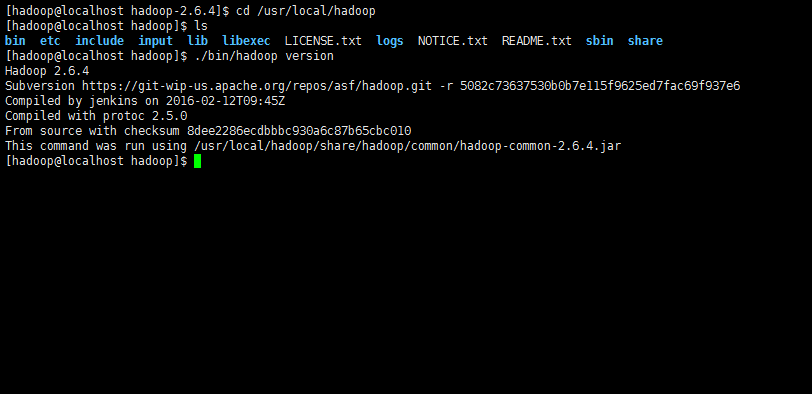
sudo mv ./hadoop-2.6.0/ ./hadoop # 将文件夹名改为hadoop

sudo chown -R hadoop:hadoop ./hadoop

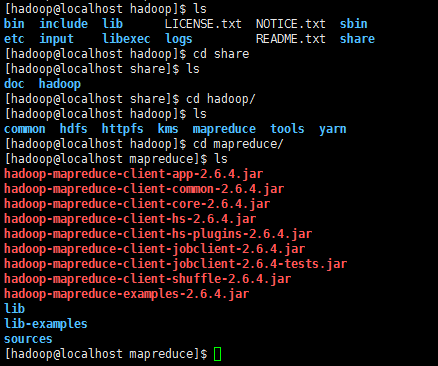
Hadoop 解压后即可使用。输入如下命令来检查 Hadoop 是否可用，成功则会显示 Hadoop 版本信息：

cd /usr/local/hadoop

./bin/hadoop version



Hadoop 默认模式为非分布式模式，无需进行其他配置即可运行。非分布式即单 Java 进程，方便进行调试。   
现在我们可以执行例子来感受下 Hadoop 的运行。Hadoop 附带了丰富的例子（运行 ./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.6.4.jar 可以看到所有例子），包括 wordcount、terasort、join、grep 等。

  
在此我们选择运行 grep 例子，我们将 input 文件夹中的所有文件作为输入，筛选当中符合正则表达式 dfs[a-z.]+ 的单词并统计出现的次数，最后输出结果到 output 文件夹中

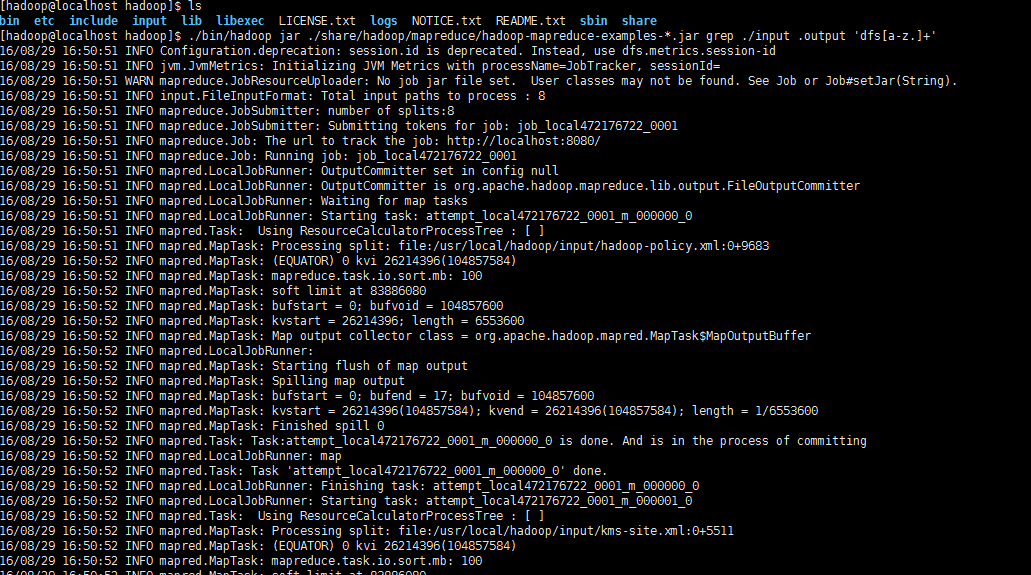
cd /usr/local/hadoop

mkdir ./input

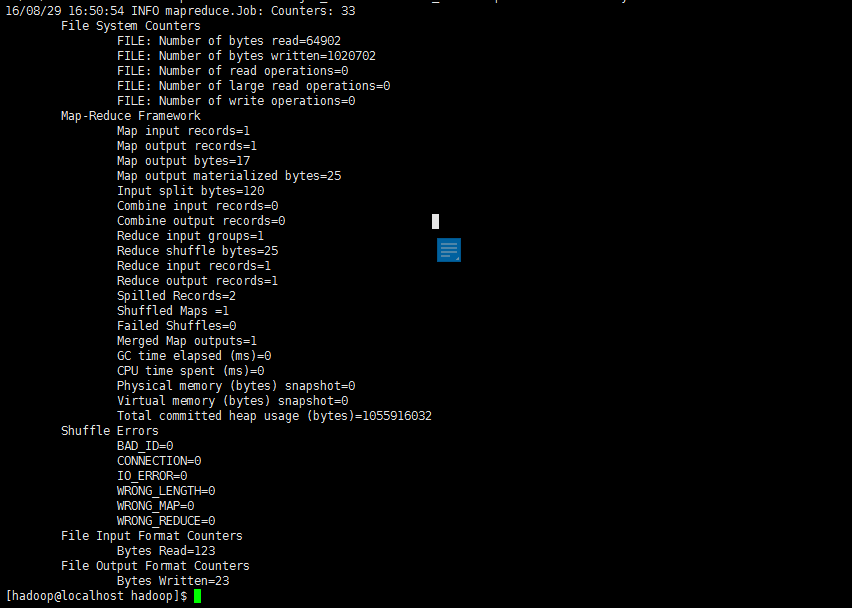
cp ./etc/hadoop/\*.xml ./input # 将配置文件作为输入文件

./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-\*.jar grep ./input ./output 'dfs[a-z.]+'

cat ./output/\*



执行信息会很多，最后结果如下图所示：

  
*注意，Hadoop 默认不会覆盖结果文件，因此再次运行上面实例会提示出错，需要先将 ./output 删除。*

**3.spark的安装**

**a.官网下载安装包**   


注意第二项要选择和自己hadoop版本相匹配的spark版本，然后在第3项点击下载。若无图形界面，可用windows系统下载完成后传送到centos中。

**b.安装spark**

[hadoop@master ~]$ cd /tmp/pkg/

[hadoop@master pkg]$ rz

[hadoop@master pkg]$ tar -zxvf spark-2.1.0-bin-hadoop2.6.tgz -C /opt/bigdata/

[hadoop@master pkg]$ cd /opt/bigdata/

[hadoop@master bigdata]$ mv spark-2.1.0-bin-hadoop2.6/ spark

**c.配置spark**   
安装后，需要在 ./conf/spark-env.sh 中修改 Spark 的 Classpath，执行如下命令拷贝一个配置文件：

[hadoop@master bigdata]$ cp ./conf/spark-env.sh.template ./conf/spark-env.sh

编辑 ./conf/spark-env.sh（vim ./conf/spark-env.sh） ，在最后面加上：

#export SPARK\_DIST\_CLASSPATH=$(/opt/bigdata/hadoop/etc/hadoop classpath)

export SCALA\_HOME=/opt/scala

export JAVA\_HOME=/opt/jdk1.8.0\_151

export SPARK\_MASTER\_IP=master

export SPARK\_MASTER\_WEBUI\_PORT=8085 #web页面端口

export SPARK\_WORKER\_MEMORY=1g

export HADOOP\_CONF\_DIR=/opt/bigdata/hadoop/etc/hadoop

将 slaves.template 拷贝到 slaves，编辑其内容为：

Master

slave01

slave02

**d.添加环境变量**   
安装后，需要在 ./conf/spark-env.sh 中修改 Spark 的 Classpath，执行如下命令拷贝一个配置文件：

[hadoop@master spark]$ sudo vi /etc/profile

#添加如下

export SPARK\_HOME=/opt/bigdata/spark

export PATH=$SPARK\_HOME/bin:$SPARK\_HOME/sbin:$PATH

[hadoop@master spark]$ . /etc/profile

**e.分发到其他结点**   
分发到slave1和slave2

[hadoop@master bigdata]$ scp -r spark/ hadoop@slave1:/opt/bigdata/

[hadoop@master bigdata]$ scp -r spark/ hadoop@slave2:/opt/bigdata/

**f.启动**   
分别启动master和slave

[hadoop@master bigdata]$ start-all.sh

或

[hadoop@master bigdata]$ stop-all.sh

**4.spark的简单使用**

在 ./examples/src/main 目录下有一些 Spark 的示例程序，有 Scala、Java、Python、R 等语言的版本。我们可以先运行一个示例程序 SparkPi（即计算 π 的近似值），执行如下命令：

[hadoop@master sbin]$ cd /opt/bigdata/spark/bin/

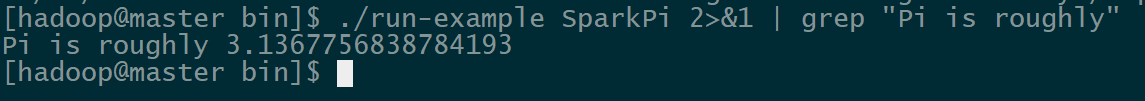
[hadoop@master bin]$ ./run-example SparkPi

执行时会输出非常多的运行信息，输出结果不容易找到，可以通过 grep 命令进行过滤（命令中的 2>&1 可以将所有的信息都输出到 stdout 中，否则由于输出日志的性质，还是会输出到屏幕中）：

[hadoop@master sbin]$ cd /opt/bigdata/spark/bin/

[hadoop@master bin]$ ./run-example SparkPi 2>&1 | grep "Pi is roughly"

过滤后的运行结果如下图所示，可以得到 π 的 近似值 ：



如果是Python 版本的 SparkPi， 则需要通过 spark-submit 运行：

[hadoop@master bin]$ ./spark-submit ../examples/src/main/python/pi.py

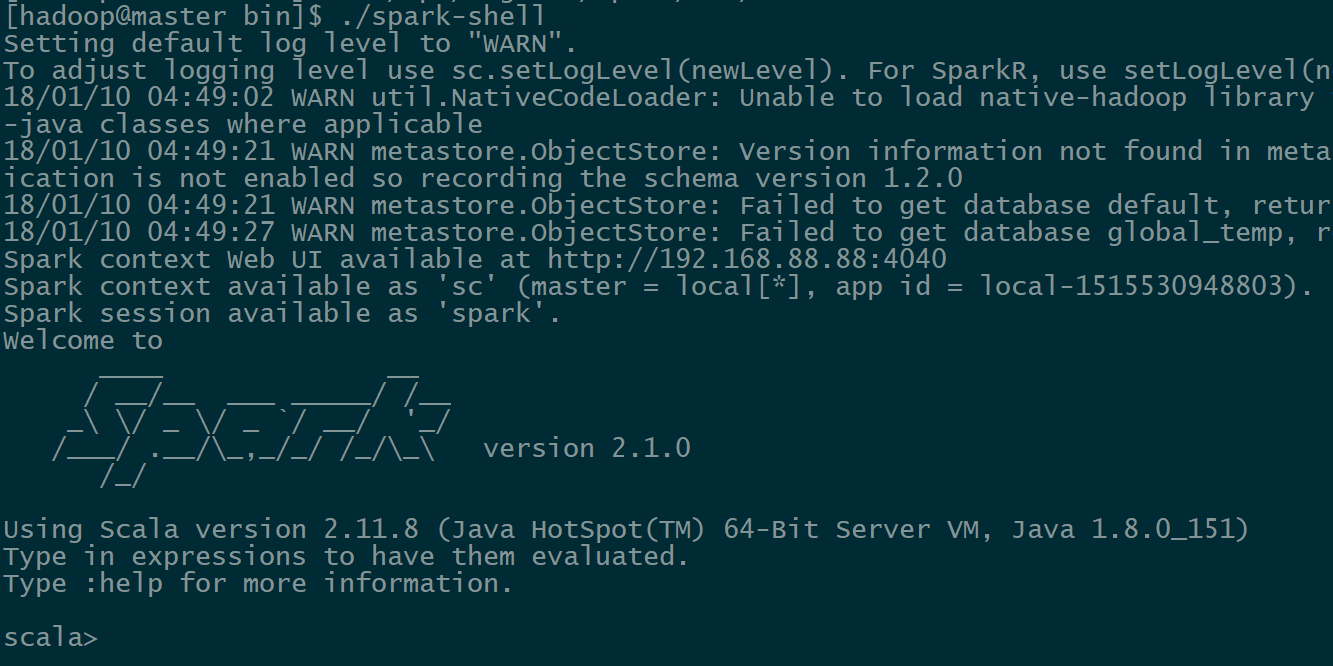
**5.spark的交互模式**

**a.启动spark shell**   
Spark shell 提供了简单的方式来学习 API，也提供了交互的方式来分析数据。Spark Shell 支持 Scala 和 Python，本文中选择使用 Scala 来进行介绍。

Scala 是一门现代的多范式编程语言，志在以简练、优雅及类型安全的方式来表达常用编程模式。它平滑地集成了面向对象和函数语言的特性。Scala 运行于 Java 平台（JVM，Java 虚拟机），并兼容现有的 Java 程序。

Scala 是 Spark 的主要编程语言，如果仅仅是写 Spark 应用，并非一定要用 Scala，用 Java、Python 都是可以的。使用 Scala 的优势是开发效率更高，代码更精简，并且可以通过 Spark Shell 进行交互式实时查询，方便排查问题。

[hadoop@master bin]$ ./spark-shell



**b.spark shell使用小例子**   
Spark 的主要抽象是分布式的元素集合（distributed collection of items），称为RDD（Resilient Distributed Dataset，弹性分布式数据集），它可被分发到集群各个节点上，进行并行操作。RDDs 可以通过 Hadoop InputFormats 创建（如 HDFS），或者从其他 RDDs 转化而来。

我们从 ./README 文件新建一个 RDD，代码如下（本文出现的 Spark 交互式命令代码中，与位于同一行的注释内容为该命令的说明，命令之后的注释内容表示交互式输出结果）：

val textFile = sc.textFile("file:///opt/bigdata/spark/README.md")

代码中通过 “file://” 前缀指定读取本地文件。Spark shell 默认是读取 HDFS 中的文件，需要先上传文件到 HDFS 中，否则会报错。   
RDDs 支持两种类型的操作

actions: 在数据集上运行计算后返回值   
transformations: 转换, 从现有数据集创建一个新的数据集   
下面我们就来演示 count() 和 first() 操作：

textFile.count() // RDD 中的 item 数量，对于文本文件，就是总行数

textFile.first() //RDD 中的第一个 item，对于文本文件，就是第一行内容

接着演示 transformation，通过 filter transformation 来返回一个新的 RDD，代码如下：

val linesWithSpark = textFile.filter(line => line.contains("Spark")) // 筛选出包含 Spark 的行

linesWithSpark.count() // 统计行数

action 和 transformation 可以用链式操作的方式结合使用，使代码更为简洁：

textFile.filter(line => line.contains("Spark")).count() // 统计包含 Spark 的行数

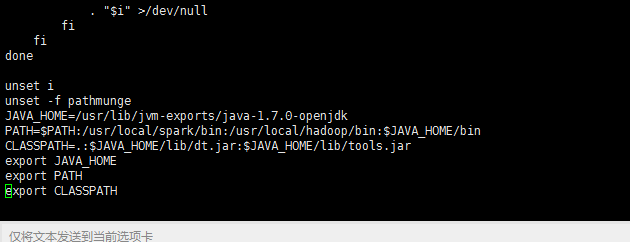
**6.spark执行独立程序(以下内容未作测试与整理)**

**a.配置spark和hadoop环境变量**

cd ~

vi /etc/profile

找到PATH参数，在最后添加spark和hadoop的环境变量，具体到bin即可。注意每条之间使用冒号隔开。如下图



修改完成，保存退出，执行

source /ect/profile

文件生效。同样可以使用echo $PATH 来查看环境变量是否添加成功。

**b.安装sbt**   
SBT(Simple Build Tool)即简单构造工具，它是用scala写的，具有强大的依赖管理功能，所有任务的创建都支持Scala，可连续执行命令。可以在工程的上下文里启动REPL。   
一般来说，使用 Scala 编写的程序需要使用 sbt 进行编译打包，相应的，Java 程序使用 Maven 编译打包，而 Python 程序通过 spark-submit 直接提交。但是scala也可以使用maven来打包，不过配置起来较为复杂。这里就不再赘述了。   
到官网下载安装包(<http://www.scala-sbt.org/>)。安装到/usr/local/sbt文件夹中

sudo mkdir /usr/local/sbt

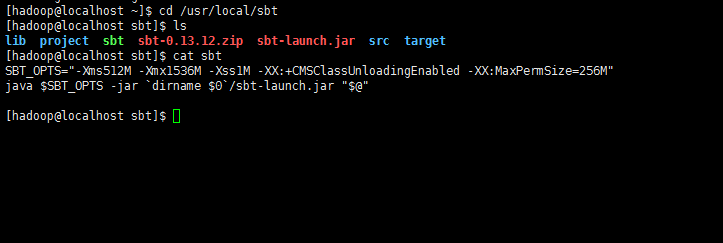
sudo chown -R hadoop /usr/local/sbt # 此处的 hadoop 为你的用户名

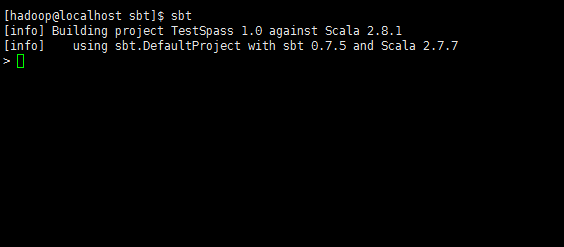
cd /usr/local/sbt

接着在 /usr/local/sbt 中创建 sbt 脚本（vim ./sbt），添加如下内容：

SBT\_OPTS="-Xms512M -Xmx1536M -Xss1M -XX:+CMSClassUnloadingEnabled -XX:MaxPermSize=256M"

java $SBT\_OPTS -jar `dirname $0`/sbt-launch.jar "$@"

注意这里的最后一行的 dirname 0,它是被倒引号括起来的，不是单引号。被倒引号括起来的东西表示要执行的命令。*dirname*0 只能用在脚本中，在命令行中是无效的，它的意思是去当前脚本所在位置的路径。   
  
很明显。*这里你要检查/usr/local/sbt 夹下sbt-launch.jar这个文件是否存在。*因为笔者和网上众多教程都提到了，因为网络的原因，sbt下载的时候，这个依赖包有可能缺失。如果没有，请自行下载(<http://pan.baidu.com/s/1gfHO7Ub>)



**c.构建scala工程目录**   
sbt打包scala是有固定工程目录结构的。

cd ~

mkdir ./sparkapp # 创建应用程序根目录

mkdir -p ./sparkapp/src/main/scala # 创建所需的文件夹结构

**d.编写独立程序**   
这里我们借用官网上的一个小的demo.

cd ~

cd sparkapp/src/main/scala

vi SimpleApp.scala

文件内容为程序主体

/\* SimpleApp.scala \*/

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object SimpleApp {

def main(args: Array[String]) {

val logFile = "file:///usr/local/spark/README.md" // Should be some file on your system

val conf = new SparkConf().setAppName("Simple Application")

val sc = new SparkContext(conf)

val logData = sc.textFile(logFile, 2).cache()

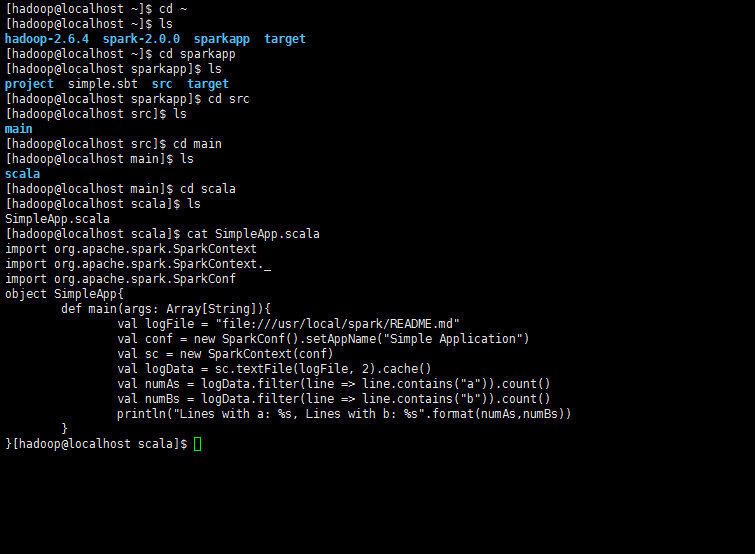
val numAs = logData.filter(line => line.contains("a")).count()

val numBs = logData.filter(line => line.contains("b")).count()

println("Lines with a: %s, Lines with b: %s".format(numAs, numBs))

}

}



**e.添加配置文件**

cd ~

cd sparkapp/

vi simple.bat

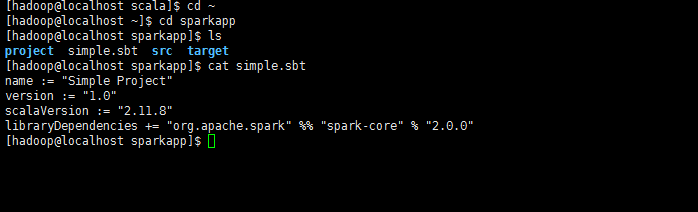
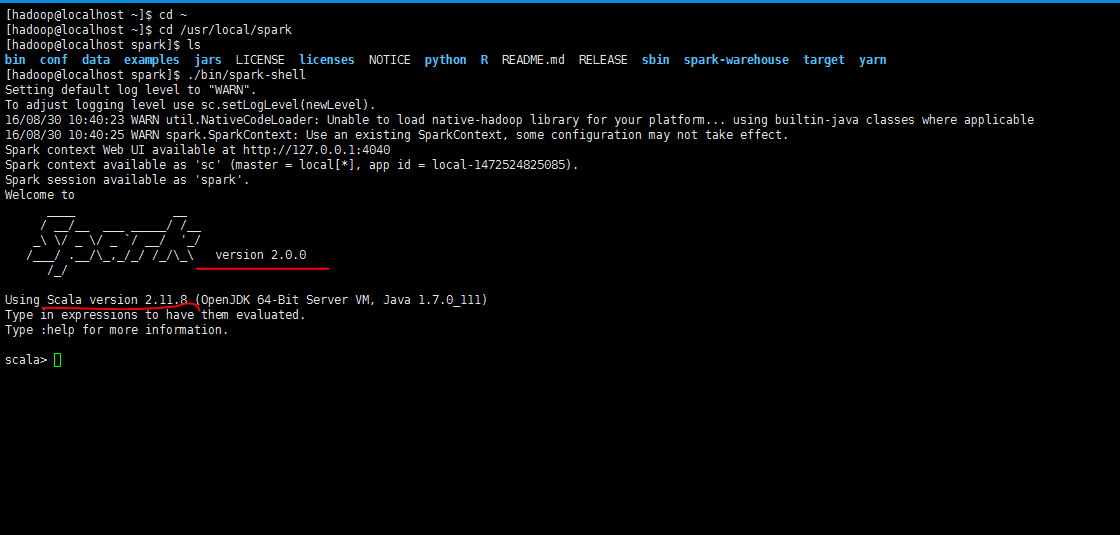
文件添加下面内容

name := "Simple Project"

version := "1.0"

scalaVersion := "2.11.8"

libraryDependencies += "org.apache.spark" %% "spark-core" % "2.0.0"

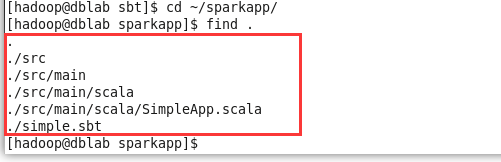
  
\*注意，这里的scalaVersion和spark-core后面的版本号都要换成你自己的。   
这两个版本号，在启动spark的时候有显示。如下图\*   


至此为止，检查一下工程目录结构

cd ~

cd sparkapp

find .



**f.使用sbt打包scala程序**

cd ~

cd sparkapp/

sbt package

第一次打包时间很长，需要下载各种依赖包，所以请耐心等待。生成的jar包位置在~/sparkapp/target/scala-2.11/simple-project\_2.11-1.0.jar

**g.提交编译后的程序**

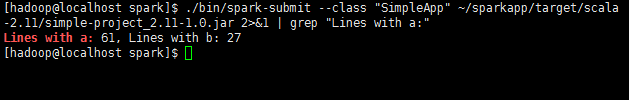
cd ~

cd /usr/local/spark

./bin/spark-submit --class "SimpleApp" ~/sparkapp/target/scala-2.11/simple-project\_2.11-1.0.jar

输出信息较多，可使用grep过滤结果

.bin/spark-submit --class "SimpleApp" ~/sparkapp/target/scala-2.11/simple-project\_2.11-1.0.jar 2>&1 | grep "Lines with a:"



到此为止，本文就结束了，关于文章中没有介绍的spark SQL和DataFrames，大家有兴趣的可以到下面列出的参考文献中查找。

列出参考文献供大家参考：   
1.<http://spark.apache.org/docs/latest/building-spark.html>   
2.<http://spark.apache.org/docs/latest/quick-start.html>   
3.<http://spark.apache.org/docs/latest/programming-guide.html>   
4.<http://www.importnew.com/4311.html>   
5.<http://www.scala-sbt.org/>   
6.<http://blog.csdn.net/czmchen/article/details/41047187>   
7.<http://blog.csdn.net/zwhfyy/article/details/8349788>   
8.<http://jingyan.baidu.com/article/948f59242c231fd80ff5f9ec.html>   
9.<http://dblab.xmu.edu.cn/blog/install-hadoop/>   
10.<http://dblab.xmu.edu.cn/blog/spark-quick-start-guide/>   
11.<http://dblab.xmu.edu.cn/blog/install-hadoop-in-centos/>