# 安装JDK

卸载openjdk

先卸载本机的jdk，安装完Centos6.5的BaseServer版本会默认安装OpenJDK，首先需要删除OpenJDK。

先执行java，如果有安装就会提示，

rpm –qa | grep java

搜索java安装，会搜到两个openjdk，依次删除。如果没有搜索到，跳过此步。

rpm -e --nodeps java-1.7.1-openjdk-1.7.0.45-2.4.3.3.el6.x86\_64

rpm -e --nodeps java-1.6.0-openjdk-1.6.0.0-1.66.1.13.0.el6.x86\_64

6.9.2 安装

yum -y install glibc.i686 #jdk依赖glibc

mkdir /usr/local/src/java #按习惯用户自己安装的软件存放到/usr/local/src目录下

rz 上传jdk tar包 #利用xshell的rz命令上传文件

tar -xvf jdk-7u51-linux-x64.tar.gz #解压压缩包

6.9.3 配置环境变量

1）vi /etc/profile

2）在尾行添加

#set java environment

JAVA\_HOME=/usr/local/src/java/jdk1.7.0\_51

JAVA\_BIN=/usr/local/src/java/jdk1.7.0\_51/bin

PATH=$JAVA\_HOME/bin:$PATH

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

export JAVA\_HOME JAVA\_BIN PATH CLASSPATH

保存退出

3）source /etc/profile 使更改的配置立即生效

4）java -version 查看JDK版本信息。如显示1.7.0证明成功。

# 关闭防火墙

service iptables stop

# rpm包解压命令

前提是linux中必须有yum,

解压: rpm -ivh 包名

查询所有rpm包: rpm –qa (| grep mysql想找的包)

删除: rpm -e 包名 (- - nodeps强制删除)

# Mysql授权语句

mysql>GRANT ALL PRIVILEGES ON \*.\* TO ‘root’@’%’ ID ENTIFIED BY ‘123’ WITH GRANT OPTION;

mysql>FLUSH PRIVILEGES;

# Spark的shell操作

sbin/start-master.sh –h 192.168.1.106 #启动master

sbin/stop-master.sh #停止master

sbin/start-slave.sh spark://192.168.1.106:7077 #指向master的访问地址

#sbin/start-slave.sh spark://192.168.1.106:7077 –h spark1 指定IP地址或者主机名访问，连接HDFS按主机名会刚好些，IP可能出错。

sbin/stop-slave.sh #停止服务

有时第一次连接时不成功，重新连接次就好了。

同样也会有日志输出，刚开始你不熟悉时，最好看一下日志，看下启动时有没有错误，如果你配置有错，或者连接地址有错，或者防火墙没开，它都有可能连接不到master。所以必须先看下日志，养成一个好习惯。

tail –f 日志文件路径

启动shell:

bin/spark-shell --master=spark://192.168.1.106:7077 #master的IP地址

连接关系型数据库转换为DF对象。

上传mysql-connector-java-5.1.28-bin.jar包

**启动参数方式：**

bin/spark-shell --master=local[4] --driver-class-path=mysql-connector-java-5.1.28-bin.jar

1. mysql服务器要能远程连接
2. mysql要授权远程访问权限

mysql –uroot –proot

mysql>\

mysql>flush privileges;

执行代码

val prop = new java.util.Properties

prop.put("user","root")

prop.put("password","root")

val df=sqlContext.read.jdbc("jdbc:mysql://ip:3306/jtdb","tb\_user",prop)

df: org.apache.spark.sql.DataFrame = [id: bigint, username: string, password: string, phone: string, email: string, created: timestamp, updated: timestamp]

用在当关系型数据库需要groupby等，由于数据量太大它做不了或者太慢。就可以交给spark来完成。

**配置文件方式：**

SPARK\_LOCAL\_IP=192.168.1.106

SPARK\_CLASSPATH=mysql-connector-java-5.1.28-bin.jar

bin/spark-shell --master=local[4]

val jdbcdf = sqlContext.load("jdbc",Map("url"->"jdbc:mysql://192.168.80.1:3306/jtdb?user=root&password=root","dbtable"->"tb\_user"))

jdbcdf.show

导入mysql-connector-java-5.1.34.jar jar包后,进入jar所在包路径:

例如:[root@spark1 spark-1.5.2-bin-hadoop2.6]

bin/spark-shell --master=spark://192.168.80.91:7077 --driver-class-path=mysql-connector-java-5.1.34.jar(必须在当前目录下启动) (--master==后面一般写成local,写成spark://192.168.80.91:7077容易出错)

# Spark的partition和RDD关系:

可以这样理解partition是RDD存储数据的地方，实际是个逻辑单位，真正要取数据时，它就调用BlockManage去读，它是以数据块的方式来读。比如一次读取32k还是64k。它不是一条一条读，一条一条读肯定性能低。它读时首先是看本地还是远程，如果是本地就直接读这个文件了，如果是远程，它就是发起一次socket请求，创建一个socket链接。然后发起一次远程调用，告诉远程的读取程序，读取哪些数据。读到的内容再通过socket传过来。

# RDD中的依赖关系:

Transformations类型的命令都没有触发job，它们这几步都在创建依赖关系，每一步都在生成新的RDD, 每一个RDD不是独立的，它必须依赖上一个RDD。或者称为父RDD对象。它们之间都有父子关系。spark就会应用一种叫流水线的技术，它特别的节省内存，并且也让性能得到提升。每个RDD有几个分区个数由父RDD决定。

上面的过程只是绘制了一个美好的蓝图，并未执行，那什么时候会真正执行运算呢？那就是执行一个actions类型命令时才会运算。

res12.collect

触发一次完整的计算，根据分区个数生成任务，根据2个分区生成了两个任务（Task）。spark就把它封装成任务对象，底层通过akka框架发给每个worker。Task对象中就包括了所有的RDD信息。Task最初在Driver上（SparkShell）。真正执行不是在SparkShell，而是在worker上执行。

怎么发？就是akka框架，先把它序列化为二进制的，再把它发给worker，worker收到二进制的，反序列化，就是Task。RDD都在Task里面，它们之间的关系就知道了。

具体计算是怎么计算的呢？它采用的是倒推的方式进行最后的计算。它先看最后一个RDD，res12去执行时，遇到变量x，它就去读res7，res7遇到变量x，就去读取res1，res1遇到textFile就按事先分区好的去读取前5行。（文件总共8行，前5行分给第一个分区，后3行分给第二个分区。）每次读一行，因为res7的x，每次只要一个。

每个分区的数据在一个worker上运行

新RDD的分区的元素来自旧RDD的某个分区，但又不全来旧的RDD的这个分区。只要有一个部分依赖就发生了shuffle。如果完全依赖才未发生shuffle。

完全依赖的好处是，那它们的task任务就可以放在一台服务器上执行。

# Spark与Hadoop之间的联系:

1,有两个worker，一个本地一个远程，spark都不把任务交给远程的worker直接本地执行了。它发现spark4数据和datanode在同一台服务器上。它就优先使用本地的。这样无需跨网络，性能就高。这是spark中一个相当优化的措施。这个不是spark自己实现的，它还是调用hadoop的api来实现判断是否是同一台服务器。

总结：在启动slave时多加-h参数指定主机名，就可以实现NODE\_LOCAL设置。

2,当文件需要在多个worker上用到，这时候最好采用hdfs这种分布式文件系统，而不是用本地文件, 如果worker和hadoop的dataNode是同一台机器，那么这样性能最好，称之为数据的本地化【\*】