下载spark

```
wget https://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
```

下载python环境变量 anaconda3

```
wget https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86_64.sh
```

将下载的spark解压和anaconda3(直接运行)安装

```
#spark解压
#解压
tar -zxf /export/software/spark-3.1.2-bin-hadoop3.2.tgz -C /export/server/
#配置软连接
ln -s /export/server/spark-3.1.2-bin-hadoop3.2 /export/server/spark
#anaconda3安装
bash /export/software/Anaconda3-2021.05-Linux-x86_64.sh
```

配置环境变量-hadoop的全部变量,profile是对全局不包括其他客户,.bashrc是对其他用户,或者可以解决远程无权限问题

/etc/profile和/root/.bashrc 两个文件都配置一下

```
#JAVA_HOME
JAVA_HOME=/export/server/jdk1.8.0_241
CLASSPATH=.:$JAVA_HOME/lib
PATH=$JAVA_HOME/bin:$PATH
export JAVA_HOME CLASSPATH PATH
#HADOOP_HOME
export HADOOP_HOME=/export/server/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
alias beeline="/export/server/apache-hive/bin/beeline -u jdbc:hive2://node1:10000 -n root -p 123456"
export SPARK_HOME=/export/server/spark
export PATH=$PATH:$SPARK_HOME/bin
export ANACONDA_HOME=/root/anaconda3
export PATH=$ANACONDA_HOME/bin:$PATH:/root
```

生效文件

```
source /etc/profile
source /root/.bashrc
```

查看python是否生效

```
#直接输入python如果是3.8则成功了
python
```

安装pyspark

```
pip install pyspark
```

在pycharm运行-- 配置连接省略了 -- 直接在pycharm运行

```
from pyspark import SparkConf,SparkContext
import os
os.environ['SPARK_HOME']='/export/server/spark'
PYSPARK_PYTHON='/root/anaconda3/bin/python'
os.environ['PYSPARK_PYTHON']=PYSPARK_PYTHON
os.environ['PYSPARK_DRIVER_PYTHON']=PYSPARK_PYTHON

if __name__ == '__main__':
    conf= SparkConf().setAppName('wordcount').setMaster('local[*]')
    sc=SparkContext(conf=conf)
    rdd1=sc.textFile('file:///export/pyworkspace_sz28/pyspark_sz28/pyspark-sparkbase_3.1.2/data/words.txt')
    rdd2=rdd1.flatMap(lambda line:line.split(' '))
    rdd3=rdd2.map(lambda word: (word ,1))
    rdd4=rdd3.reduceByKey(lambda x,y:x+y)
    list1=rdd4.collect()
    for x in list1 : print(x)
```

读取HDFS输入输出文件-- 可以直接在pycharm运行

```
from pyspark import SparkConf, SparkContext
import os
import sys

# 这里选择本地pyspark环境执行spark代码
os.environ['SPARK_HONE'] = '/export/server/spark'
PYSPARK_PYTHON = '/root/anaconda3/bin/python'
#当存在多个版本时,不指定很可能会导致出错
os.environ['PYSPARK_PYTHON'] = PYSPARK_PYTHON
os.environ['PYSPARK_DRIVER_PYTHON'] = PYSPARK_PYTHON

```

```
if __name__ == '__main__':
    conf = SparkConf().setAppName('wordcount').setMaster('local[*]')
    sc = SparkContext(conf=conf)
    rdd1 = sc.textFile('hdfs://node1:8020/pydata/words.txt')
    rdd2 = rdd1.flatMap(lambda line: line.split(' '))
    rdd3 = rdd2.map(lambda word: (word, 1))
    rdd4 = rdd3.reduceByKey(lambda x, y: x + y)
    rdd4.saveAsTextFile('hdfs://node1:8020/pydata/output2.txt')
```

上传文件

```
hdfs dfs -mkdir /pydata
hdfs dfs -put /export/pyworkspace/pyspark_sz28/pyspark-sparkbase_3.1.2/data/words.txt /pydata
```

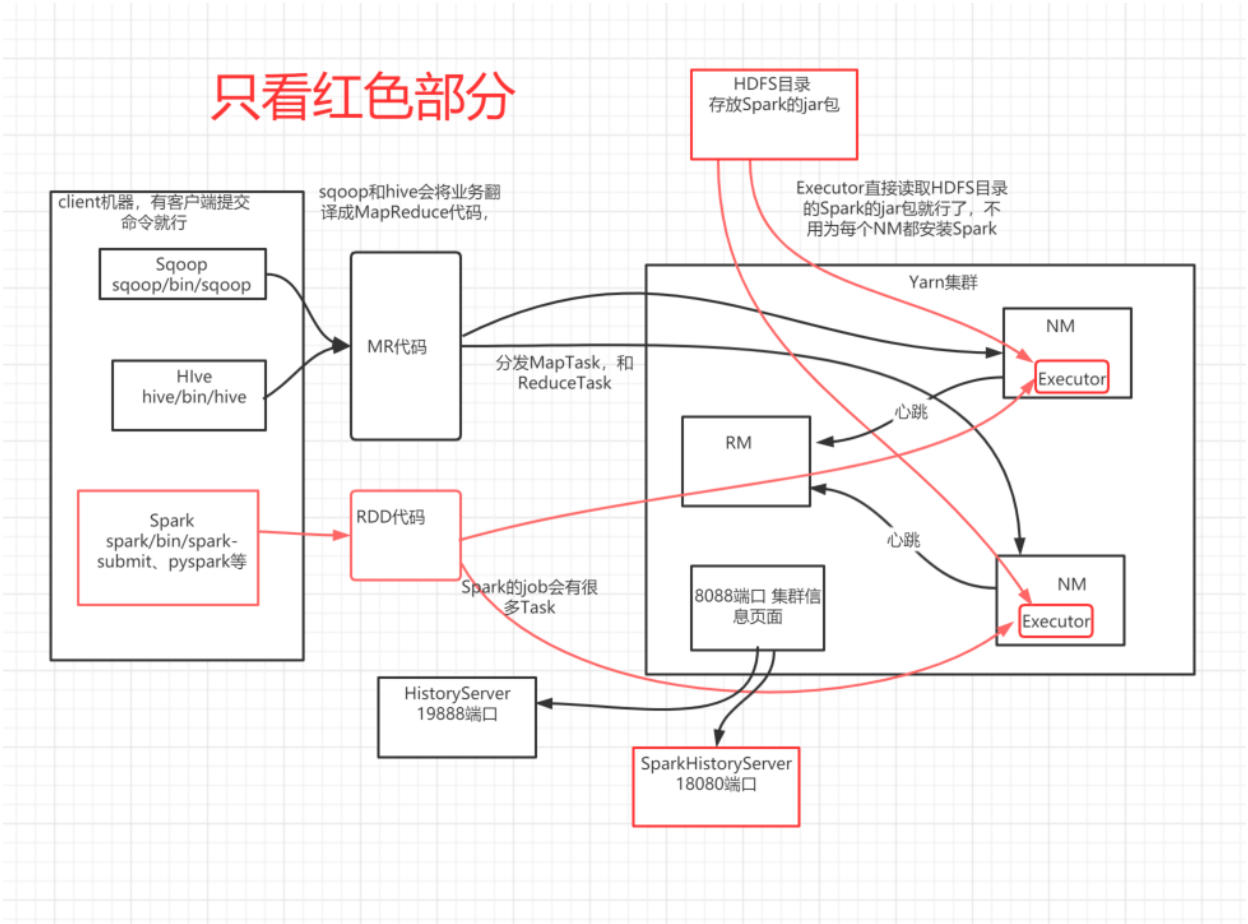
使用spark-submit提交运行

```
import sys

from pyspark import SparkConf,SparkContext
import os
os.environ['SPARK_HOME']='/export/server/spark'
PYSPARK_PYTHON='/root/anaconda3/bin/python'
os.environ['PYSPARK_PYTHON']=PYSPARK_PYTHON
os.environ['PYSPARK_DRIVER_PYTHON']=PYSPARK_PYTHON

if __name__ == '__main__':
    #1.首先创建sparkcontext上下文环境,
    conf= SparkConf().setAppName('wordcount').setMaster('local[*]')
    sc=SparkContext(conf=conf)
    #2.从外部文件数据源读取数据
    rdd1=sc.textFile(sys.argv[1])
    #3.执行flatmap执行扁平化操作
    rdd2=rdd1.flatMap(lambda line:line.split(' '))
    #4.执行map转化操作,得到(word,1)
    rdd3=rdd2.map(lambda word: (word ,1))
    #5.reduceByKey将相同的key的value数据累加操作
    rdd4=rdd3.reduceByKey(lambda x,y:x+y)
    #6.将结果打印
    list1=rdd4.collect()
    for x in list1 : print(x)
```

在linux客户端运行

```
spark-submit --master local[*] /export/pyworkspace/pyspark_sz28/pyspark-sparkbase_3.1.2/main/wordcount_spark-submit.py /export/pyworkspace/pyspark_sz28/pyspark-sparkbase_3.1.2/data/words.txt

```

sparkOnYarn安装配置



每个节点都需要python3环境,所有每台机器都装anaconda3,可以不装pyspark

```
scp /export/software/Anaconda3-2021.05-Linux-x86_64.sh root@node2:$PWD
scp /export/software/Anaconda3-2021.05-Linux-x86_64.sh root@node3:$PWD
```

复制hive的/bin/hive-size.xml到spark的同目录下--metastore地址

```
cp /export/server/hive/conf/hive-size.xml /export/server/spark/conf/
```

创建spark.env.sh

```
cd /export/server/spark/conf
cp spark-env.sh.template spark-env.sh
```

在spark.env.sh添加

```
JAVA_HOME=/export/server/jdk1.8.0_241
## HADOOP软件配置文件目录，读取HDFS上文件和运行YARN集群
HADOOP_CONF_DIR=/export/server/hadoop/etc/hadoop
YARN_CONF_DIR=/export/server/hadoop/etc/hadoop
SPARK_HISTORY_OPTS="-Dspark.history.fs.logDirectory=hdfs://node1:8020/sparklog/ -
Dspark.history.fs.cleaner.enabled=true"
```

整合历史服务器并关闭资源检查

使用虚拟机运行服务,默认会检查,如果内存不足无法运行

在$HADOOP_HOME/etc/hadoop/yarn-site.xml,添加:

```
    <!-- 关闭yarn内存检查 -->
    <property>
        <name>yarn.nodemanager.pmem-check-enabled</name>
        <value>false</value>
    </property>
    <property>
        <name>yarn.nodemanager.vmem-check-enabled</name>
        <value>false</value>
    </property>
```

然后分发到其他节点

```
scp /export/server/hadoop/etc/hadoop/yarn-site.xml
node2:///export/server/hadoop/etc/hadoop/
scp /export/server/hadoop/etc/hadoop/yarn-site.xml
node3:///export/server/hadoop/etc/hadoop/
```

配置spark历史服务器spark

```
#手动创建HDFS目录
hdfs dfs -mkdir /sparklog
cd /export/server/spark/conf
cp spark-defaults.conf.template spark-defaults.conf
```

创建spark.defaults.conf

```
cd /export/server/spark/conf
cp spark.defaults.conf.template spark.defaults.conf
#在spark-defaults.conf添加内容
spark.eventLog.enabled                  true
```

```
spark.eventLog.dir                      hdfs://node1:8020/sparklog/
spark.eventLog.compress                 true
spark.yarn.historyServer.address        node1:18080
spark.yarn.jars  hdfs://node1:8020/spark/jars/*
```

设置日志级别

```
cp log4j.properties.template log4j.properties
#修改log4j.properties,将INFO改成WARN 大概在19行
```

配置依赖spark jar包

```
## hdfs上创建存储spark相关jar包目录
hadoop fs -mkdir -p /spark/jars/
## 上传$SPARK_HOME/jars所有jar包
hadoop fs -put /export/server/spark/jars/* /spark/jars/
```

启动

启动服务:HDFS、YARN、MRHistoryServer和Spark HistoryServer

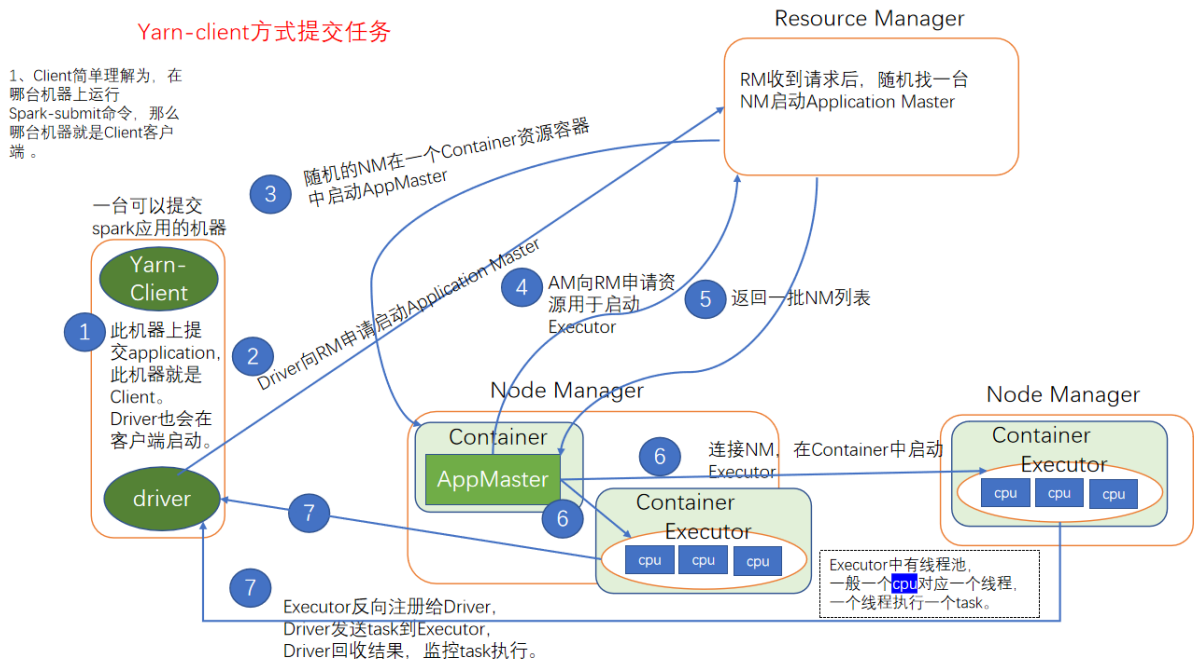
```
## 启动HDFS和YARN服务，在node1执行命令
start-dfs.sh
start-yarn.sh
或
start-all.sh
注意：在onyarn模式下不需要启动spark/bin/start-all.sh（jps查看一下看到worker和master）
## 启动MRHistoryServer服务，在node1执行命令
mr-jobhistory-daemon.sh start historyserver
## 启动Spark HistoryServer服务，，在node1执行命令
/export/server/spark/sbin/start-history-server.sh

```

测试

client模式

```
SPARK_HOME=/export/server/spark
${SPARK_HOME}/bin/spark-submit \
--master yarn \
--deploy-mode client \
--driver-memory 512m \
--executor-memory 512m \
--executor-cores 1 \
```

```
--num-executors 2 \
--queue default \
--conf "spark.pyspark.driver.python=/root/anaconda3/bin/python3" \
--conf "spark.pyspark.python=/root/anaconda3/bin/python3" \
${SPARK_HOME}/examples/src/main/python/pi.py \
10
```

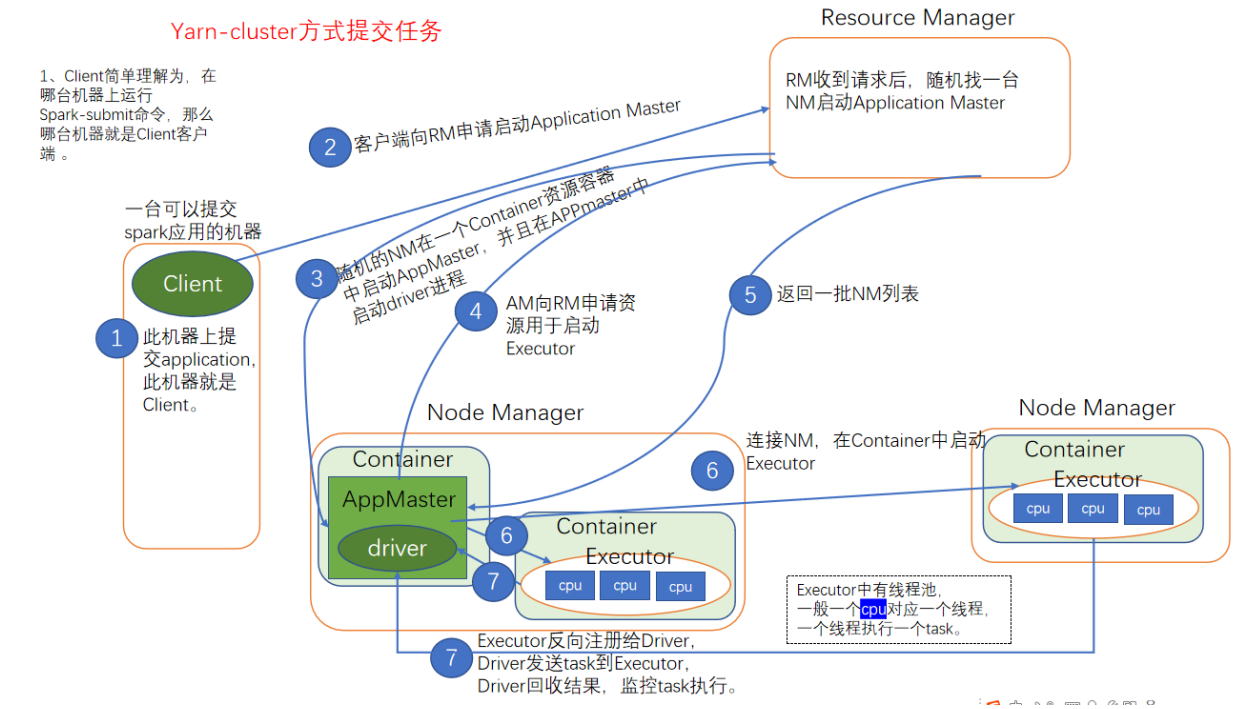


cluster模式

```
${SPARK_HOME}/bin/spark-submit \
--master yarn \
--deploy-mode cluster \
--driver-memory 512m \
--executor-memory 512m \
--executor-cores 1 \
--num-executors 2 \
--queue default \
--conf "spark.pyspark.driver.python=/root/anaconda3/bin/python3" \
--conf "spark.pyspark.python=/root/anaconda3/bin/python3" \
${SPARK_HOME}/examples/src/main/python/pi.py \
10
```

```
spark-submit \
--master yarn \
--deploy-mode cluster \
```

```
--driver-memory 512m \
--executor-memory 512m \
--executor-cores 1 \
--num-executors 2 \
--queue default \
--class bigdata.spark.sql.Spark06_SparkSQL \
root/spark-1.0-SNAPSHOT.jar

```



参数说明

```
--master  spark使用什么资源管理器运行,比如yarn
--deploy-mode 模式:client和cluster
--name 给一个程序起一个名字[可选]
--class 包名.包类 [可选]
--driver-core 给driver申请几个核
--driver-memory 申请driver进程的内存
--executor-memory 申请每个executor内存
--executor-core 申请每个executor几个核
--num-executor 一共申请几个executor
--queue 用来隔离CPU和内存资源,每个队列中都包含指定容量的CPU和内存
```

启动spark端口

```
nohup /export/server/hive/bin/hive --service metastore 2>&1 > /tmp/hive-metastore.log &
/export/server/spark/sbin/start-thriftserver.sh \
```

```
--hiveconf hive.server2.thrift.port=10001 \
--hiveconf hive.server2.thrift.bind.host=node1 \
--master local[*]
```

spark和hive乱码（查询）

```
-- 下面修改是在MySQL中修改，因为MySQL记录维护着元数据

use hive3;

-- （1）修改表字段注解和表注解

alter table COLUMNS_V2 modify column COMMENT varchar(256) character set utf8;
alter table TABLE_PARAMS modify column PARAM_VALUE varchar(4000) character set utf8;

-- （2）修改分区字段注解

alter table PARTITION_PARAMS modify column PARAM_VALUE varchar(4000) character set utf8
;
alter table PARTITION_KEYS modify column PKEY_COMMENT varchar(4000) character set utf8;

# （3）修改索引注解

alter table INDEX_PARAMS modify column PARAM_VALUE varchar(4000) character set utf8;

-- 下面这个在Hive的配置文件中修改

# <!-- 存储元数据mysql相关配置 -->
# <property>
#  <name>javax.jdo.option.ConnectionURL</name>
#  <value>jdbc:mysql://node1:3306/hive3?
createDatabaseIfNotExist=true&amp;useSSL=false&amp;useUnicode=true&amp;characterEncoding
=UTF-8</value>
# </property>
#
# 在IDEA/pycharm中修改
# tools（工具）-部署-advanced（配置）-把GDK改成utf-8
```