**目录**

[虚拟机安装CentOS7 1](#_Toc26642566)

[远程连接虚拟机 5](#_Toc26642567)

[HDFS伪分布式搭建 7](#_Toc26642568)

[查看自带的openjdk 7](#_Toc26642569)

[卸载系统自带的openjdk 7](#_Toc26642570)

[各台虚拟机关闭防火墙 8](#_Toc26642571)

[各台机器关闭selinux（linux里面的安全策略，类似防火墙） 8](#_Toc26642572)

[各台机器更改主机名 9](#_Toc26642573)

[各台机器做主机名与IP地址的映射 9](#_Toc26642574)

[各台机器重启 9](#_Toc26642575)

[创建文件夹 9](#_Toc26642576)

[配置环境变量 10](#_Toc26642577)

[安装ssh 11](#_Toc26642578)

[修改core-site.xml 13](#_Toc26642579)

[修改hdfs-site.xml 14](#_Toc26642580)

[修改hadoop-env.sh 15](#_Toc26642581)

[修改mapred-site.xml 16](#_Toc26642582)

[修改yarn-site.xml 16](#_Toc26642583)

[修改slaves文件 17](#_Toc26642584)

[创建文件存放目录 17](#_Toc26642585)

[配置hadoop的环境变量 18](#_Toc26642586)

[集群启动 18](#_Toc26642587)

[脚本一键启动 18](#_Toc26642588)

[停止集群 19](#_Toc26642589)

[浏览器查看启动页面 19](#_Toc26642590)

[hdfs集群访问地址 19](#_Toc26642591)

[yarn集群访问地址 20](#_Toc26642592)

[jobhistory访问地址 20](#_Toc26642593)

[Hbase安装 20](#_Toc26642594)

[使用wget下载Hbase 20](#_Toc26642595)

[解压目录 20](#_Toc26642596)

[设置hbase环境变量 21](#_Toc26642597)

[配置 hbase-env.sh文件 21](#_Toc26642598)

[配置 hbase-site.xml 21](#_Toc26642599)

[启动hbase 22](#_Toc26642600)

[进入hbase shell 22](#_Toc26642601)

[进入hbase的web页面 23](#_Toc26642602)

[HBase shell 23](#_Toc26642603)

[表和列族操作 24](#_Toc26642604)

[创建表 24](#_Toc26642605)

[查看表名列表 24](#_Toc26642606)

[描述表结构 25](#_Toc26642607)

[修改表结构 25](#_Toc26642608)

[删除表 26](#_Toc26642609)

[数据更新 27](#_Toc26642610)

[计数器 28](#_Toc26642611)

[数据查询 29](#_Toc26642612)

[过滤查询 29](#_Toc26642613)

[行键过滤器 30](#_Toc26642614)

[列族和列过滤器 31](#_Toc26642615)

[值过滤器 32](#_Toc26642616)

[其他过滤器 32](#_Toc26642617)

[快照操作 33](#_Toc26642618)

[Java访问Hbase 33](#_Toc26642619)

[建立连接 34](#_Toc26642620)

[建立和删除表 34](#_Toc26642621)

[描述表结构 35](#_Toc26642622)

[数据更新 37](#_Toc26642623)

[数据查询 40](#_Toc26642624)

[删除行和列 41](#_Toc26642625)

[过滤器 43](#_Toc26642626)

[解决Java API不能远程访问HBase的问题 44](#_Toc26642627)

[Python 访问Hbase 45](#_Toc26642628)

[CentOS安装Thrift 45](#_Toc26642629)

[安装依赖 45](#_Toc26642630)

[安装boost包 45](#_Toc26642631)

[安装thrift 45](#_Toc26642632)

[验证是否安装成功 46](#_Toc26642633)

[打开HBase的Thrift服务 46](#_Toc26642634)

[客户端配置Python环境pip安装Thrift 46](#_Toc26642635)

[修改代码文件 46](#_Toc26642636)

[引用的类库 48](#_Toc26642637)

[建立连接 48](#_Toc26642638)

[列举所有表名 49](#_Toc26642639)

[表的建立 49](#_Toc26642640)

[表的禁用删除 49](#_Toc26642641)

[查看表结构 50](#_Toc26642642)

[插入数据 50](#_Toc26642643)

[检索数据 50](#_Toc26642644)

[删除数据 51](#_Toc26642645)

[安装Cassandra 51](#_Toc26642646)

[修改配置文件 cassandra.yaml 51](#_Toc26642647)

[a.修改cassandra集群的名字(默认是 Test Cluster) 52](#_Toc26642648)

[b.设置集群种子节点IP，如果多个用逗号分隔 52](#_Toc26642649)

[c.设置监听地址(本机的IP)，是为了其他节点能与节点进行通信(默认是 localhost)，每台机器填自己机器的IP 52](#_Toc26642650)

[d.开启 thrift rpc 服务(默认是 false) 52](#_Toc26642651)

[e.设置rpc的地址(默认是 localhost) 52](#_Toc26642652)

[f.设置数据文件所在路径(默认是 $CASSANDRA\_HOME/data/data) 52](#_Toc26642653)

[g.设置commitlog文件所在路径(默认是 $CASSANDRA\_HOME/data/commitlog) 53](#_Toc26642654)

[分发安装包 53](#_Toc26642655)

[修改 $CASSANDRA\_HOME/conf/cassandra.yaml 中的 listen\_address 和 rpc\_address 将其设置成自己的IP 53](#_Toc26642656)

[启动cassandra 53](#_Toc26642657)

[CentOS 6.9下将python2.6.6升级为Python2.7.13 54](#_Toc26642658)

[cqlsh基本用法 57](#_Toc26642659)

[键空间管理 58](#_Toc26642660)

[创建键空间 58](#_Toc26642661)

[删除键空间 58](#_Toc26642662)

[查看键空间列表 58](#_Toc26642663)

[修改键空间属性 58](#_Toc26642664)

[系统键空间 59](#_Toc26642665)

[数据表管理 59](#_Toc26642666)

[建立数据表 59](#_Toc26642667)

[设置复合型主键 60](#_Toc26642668)

[修改表结构 60](#_Toc26642669)

[删除数据并重建表 60](#_Toc26642670)

[用户自定义类型 60](#_Toc26642671)

[CQL数据查询 60](#_Toc26642672)

[条件查询 61](#_Toc26642673)

[切片查询 62](#_Toc26642674)

[索引机制 63](#_Toc26642675)

[使用标量函数 63](#_Toc26642676)

[数据更新 63](#_Toc26642677)

[插入更新删除 63](#_Toc26642678)

[数据插入 63](#_Toc26642679)

[数据更新 64](#_Toc26642680)

[数据删除 65](#_Toc26642681)

[json格式插入数据 65](#_Toc26642682)

[读写一致性 65](#_Toc26642683)

[查看一致性设置 65](#_Toc26642684)

[if 轻量级事物 65](#_Toc26642685)

[集合列操作 65](#_Toc26642686)

[list类型更新删除 66](#_Toc26642687)

[set类型更新和删除 66](#_Toc26642688)

[nodetool工具 67](#_Toc26642689)

[查看集群状态 67](#_Toc26642690)

[查看compation信息 67](#_Toc26642691)

[JAVA访问Cassandra 67](#_Toc26642692)

[修改pom.xml 67](#_Toc26642693)

[Python访问Cassandra 71](#_Toc26642694)

[MongoDB安装 75](#_Toc26642695)

[yum安装 75](#_Toc26642696)

[解压安装 76](#_Toc26642697)

[配置文件 77](#_Toc26642698)

[shell 77](#_Toc26642699)

[数据库和集合操作 77](#_Toc26642700)

[数据库操作 77](#_Toc26642701)

[集合操作 79](#_Toc26642702)

[基本增删改查操作 79](#_Toc26642703)

[文档更新 80](#_Toc26642704)

[聚合和管道 80](#_Toc26642705)

[索引操作 81](#_Toc26642706)

[python访问MongoDB 82](#_Toc26642707)

[安装 82](#_Toc26642708)

[导包 83](#_Toc26642709)

[建立连接 83](#_Toc26642710)

[切换数据库 83](#_Toc26642711)

[切换集合 83](#_Toc26642712)

[定义JSON文档 83](#_Toc26642713)

[插入记录 83](#_Toc26642714)

[查看数据 84](#_Toc26642715)

[更新数据 84](#_Toc26642716)

[$push $each插入多个元素 84](#_Toc26642717)

[累加方式更新数字类型元素 84](#_Toc26642718)

[$pop删除数据 84](#_Toc26642719)

[更新数据 85](#_Toc26642720)

[查看数据 85](#_Toc26642721)

[排序 限制 跳过 85](#_Toc26642722)

[控制显示的列 85](#_Toc26642723)

[聚合查询 86](#_Toc26642724)

[比较运算符 86](#_Toc26642725)

[地理索引查询 86](#_Toc26642726)

[Gridfs操作 87](#_Toc26642727)

[删除数据 87](#_Toc26642728)

[删除集合 87](#_Toc26642729)

[Neo4j安装 88](#_Toc26642730)

[下载Neo4j 88](#_Toc26642731)

[安装 88](#_Toc26642732)

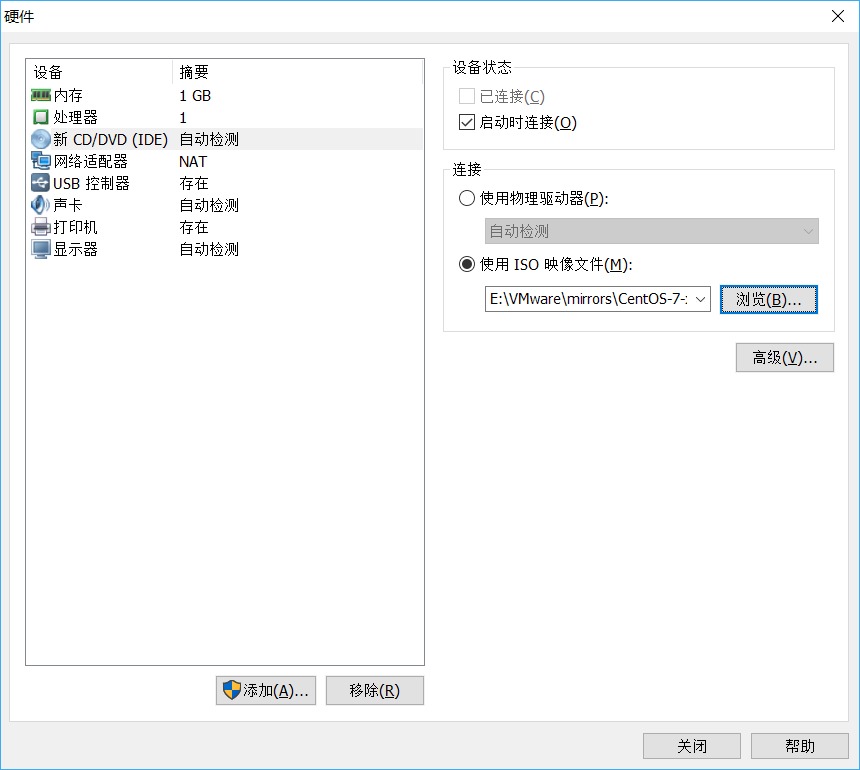
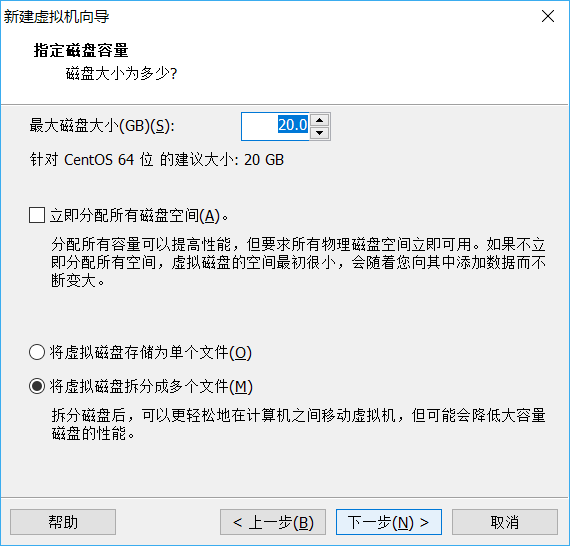
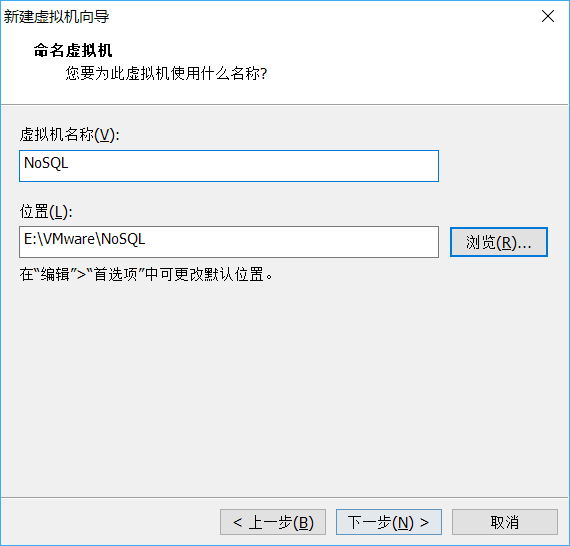
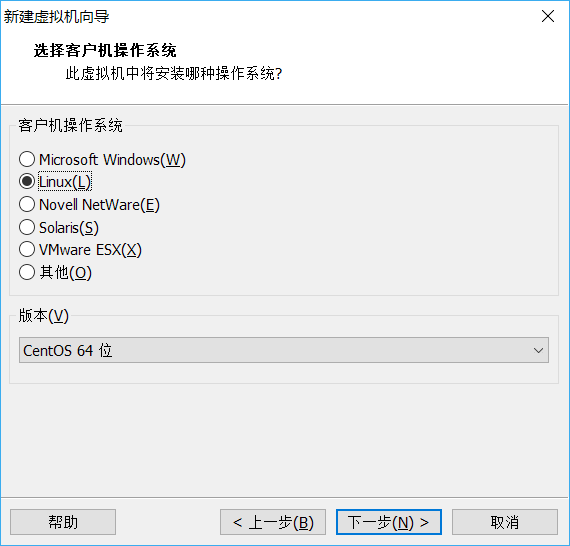
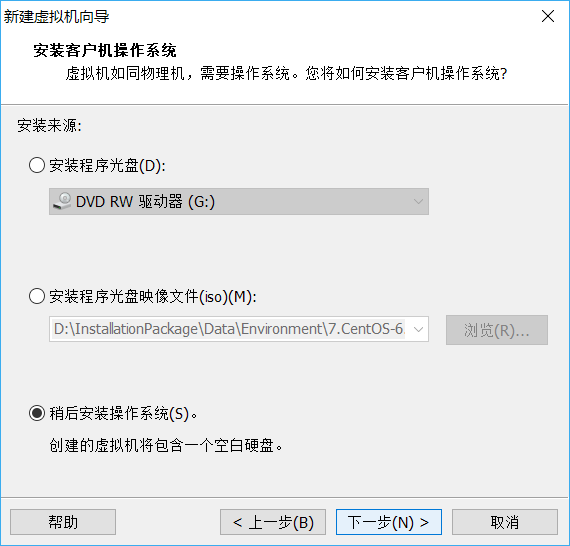
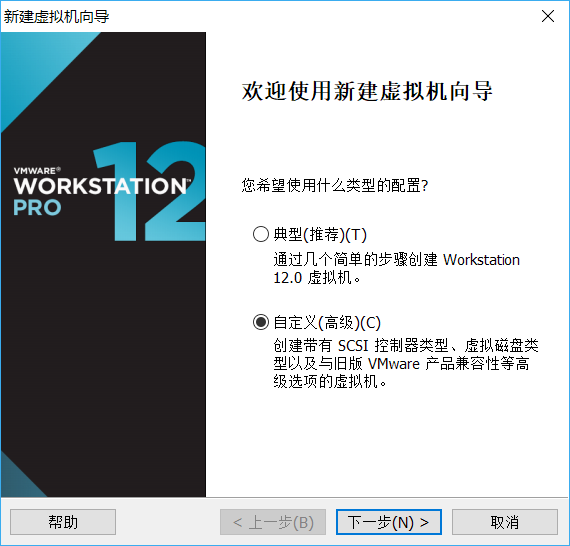
[启动neo4j服务 89](#_Toc26642733)

[访问web 89](#_Toc26642734)

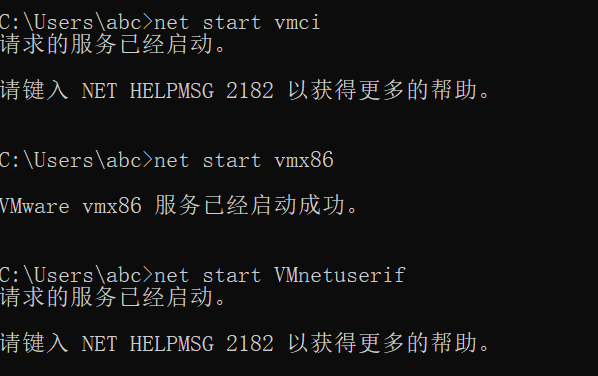
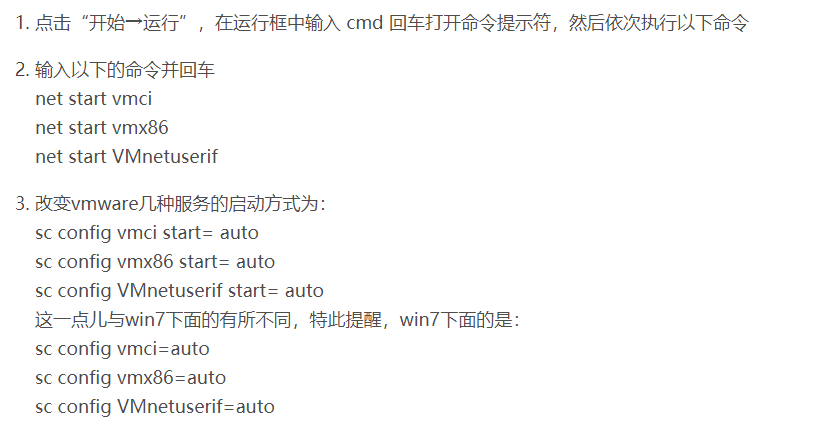
[Python访问Neo4j 89](#_Toc26642735)

# HBase

## 虚拟机安装CentOS7

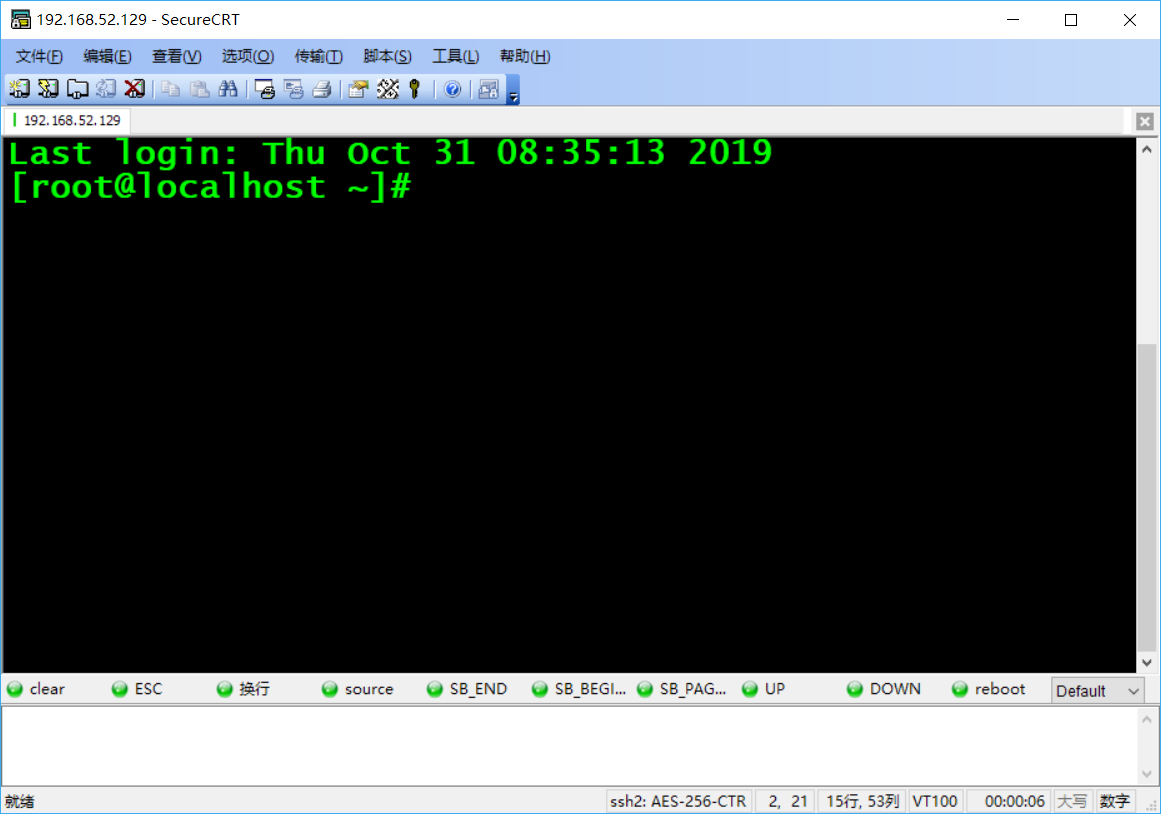
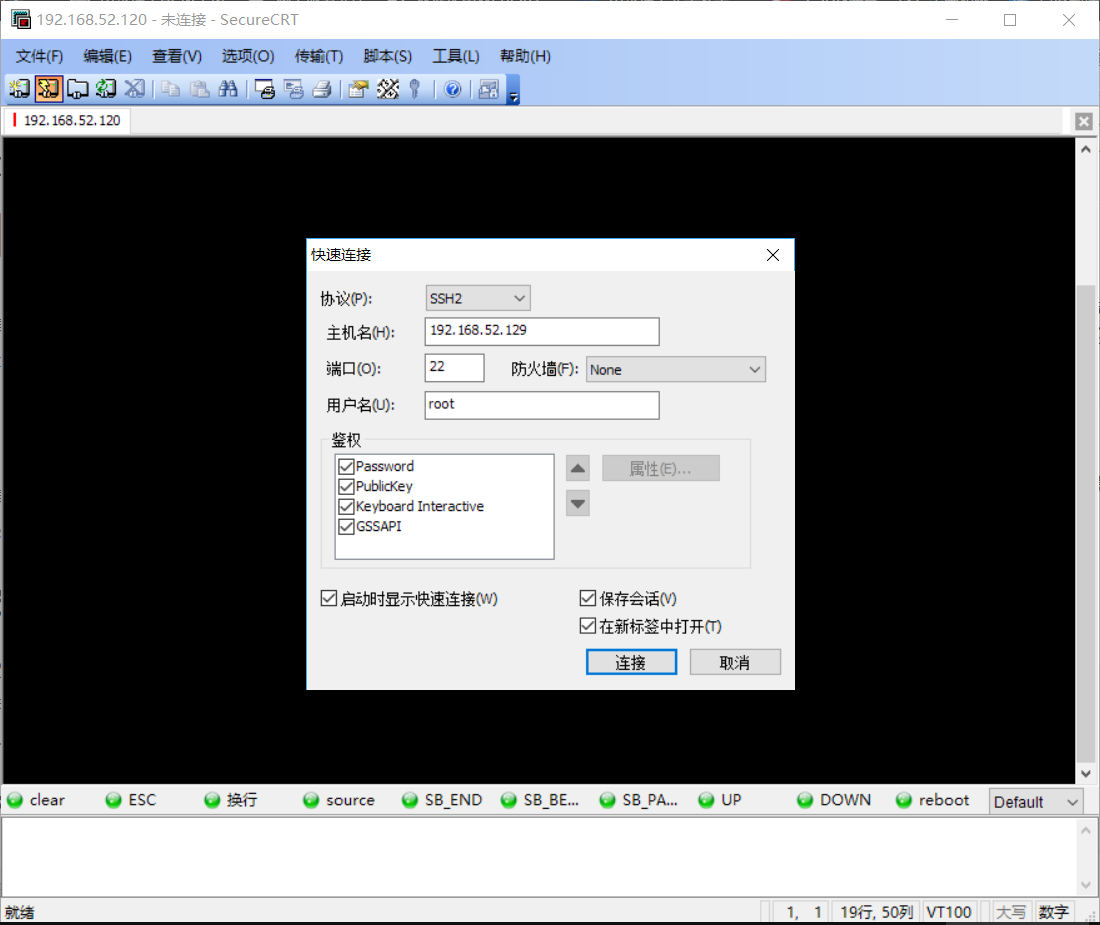
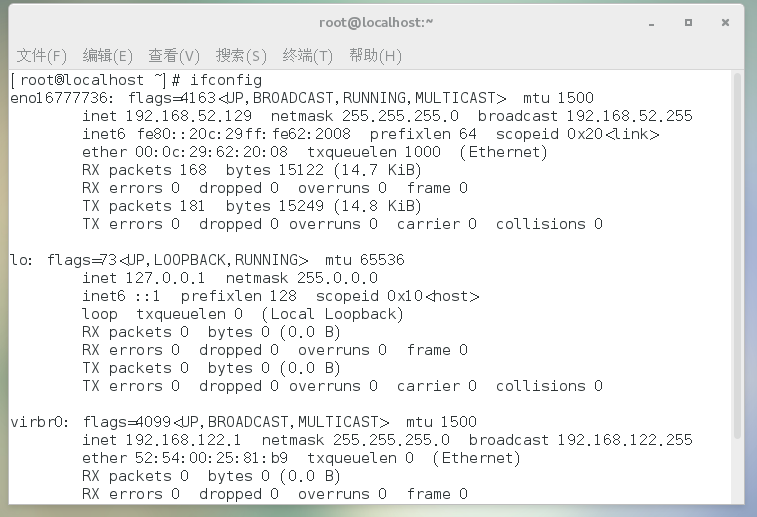






# 远程连接虚拟机

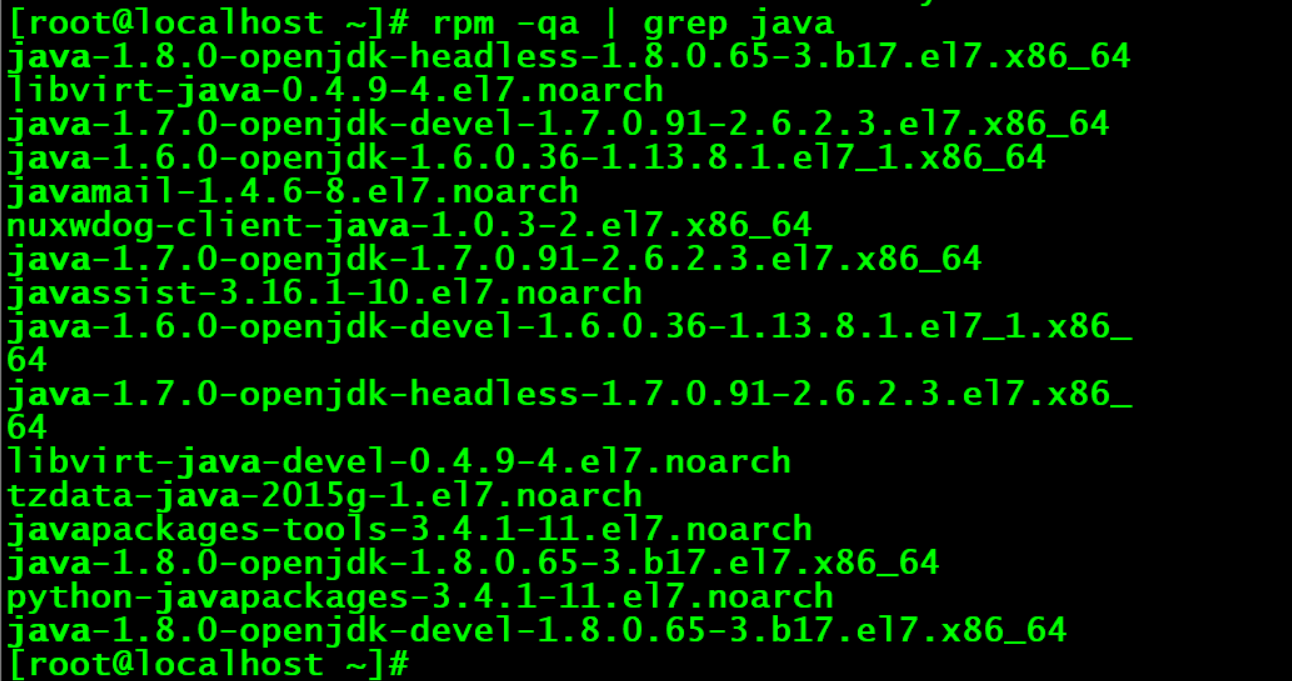
ifconfig



# HDFS伪分布式搭建

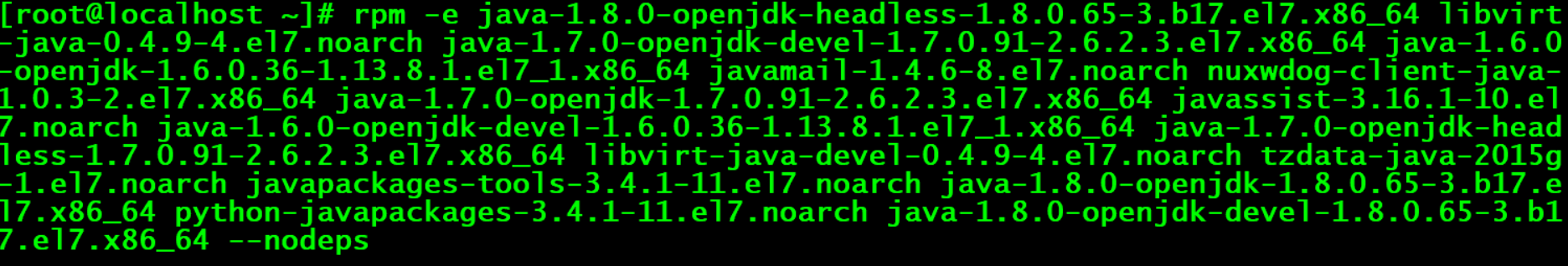
## 查看自带的openjdk

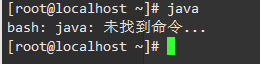
rpm -qa | grep java



## 卸载系统自带的openjdk

rpm -e java-1.8.0-openjdk-headless-1.8.0.65-3.b17.el7.x86\_64 libvirt-java-0.4.9-4.el7.noarch java-1.7.0-openjdk-devel-1.7.0.91-2.6.2.3.el7.x86\_64 java-1.6.0-openjdk-1.6.0.36-1.13.8.1.el7\_1.x86\_64 javamail-1.4.6-8.el7.noarch nuxwdog-client-java-1.0.3-2.el7.x86\_64 java-1.7.0-openjdk-1.7.0.91-2.6.2.3.el7.x86\_64 javassist-3.16.1-10.el7.noarch java-1.6.0-openjdk-devel-1.6.0.36-1.13.8.1.el7\_1.x86\_64 java-1.7.0-openjdk-headless-1.7.0.91-2.6.2.3.el7.x86\_64 libvirt-java-devel-0.4.9-4.el7.noarch tzdata-java-2015g-1.el7.noarch javapackages-tools-3.4.1-11.el7.noarch java-1.8.0-openjdk-1.8.0.65-3.b17.el7.x86\_64 python-javapackages-3.4.1-11.el7.noarch java-1.8.0-openjdk-devel-1.8.0.65-3.b17.el7.x86\_64 --nodeps



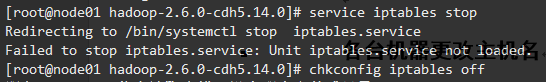


## 各台虚拟机关闭防火墙

各台机器执行以下命令（root用户来执行）

service iptables stop

chkconfig iptables off



1:查看防火状态

systemctl status firewalld

service iptables status

2:暂时关闭防火墙

systemctl stop firewalld

service iptables stop

3:永久关闭防火墙

systemctl disable firewalld

chkconfig iptables off

4:重启防火墙

systemctl enable firewalld

service iptables restart

5:永久关闭后重启

chkconfig iptables on

## 各台机器关闭selinux（linux里面的安全策略，类似防火墙）

vim /etc/selinux/config

## 各台机器更改主机名

vim /etc/sysconfig/network



NETWORKING=yes

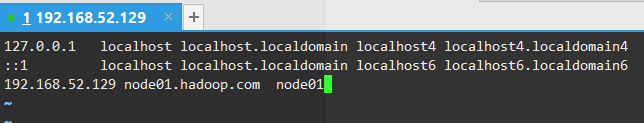
HOSTNAME=node01.hadoop.com



## 各台机器做主机名与IP地址的映射

vim /etc/hosts

192.168.52.129 node01.hadoop.com node01



## 各台机器重启

reboot -h now

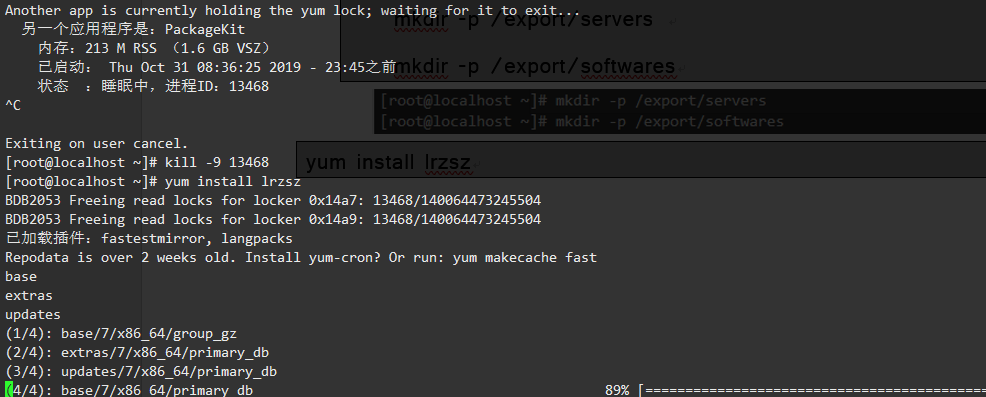
## 创建文件夹

mkdir -p /export/servers

mkdir -p /export/softwares

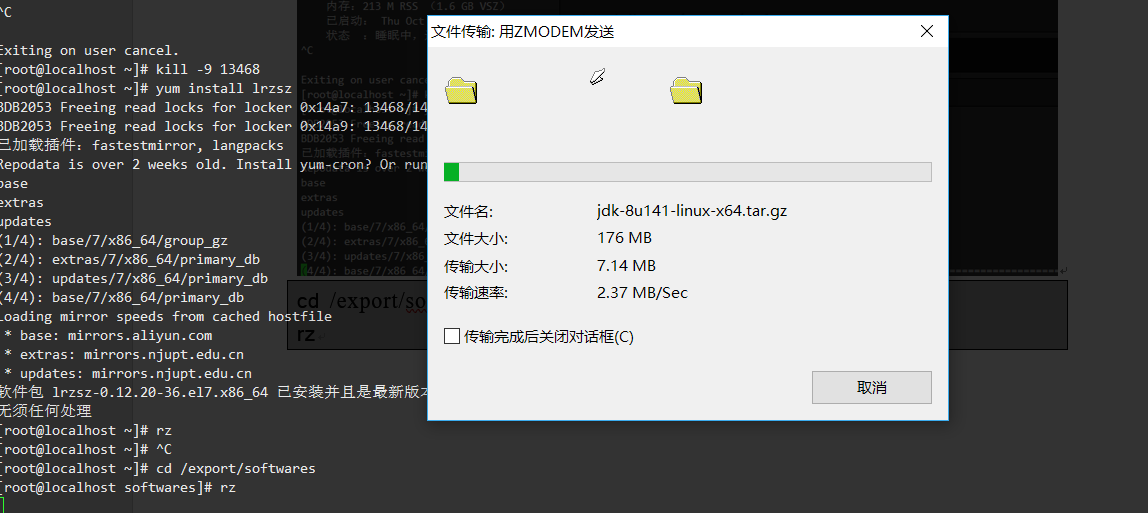


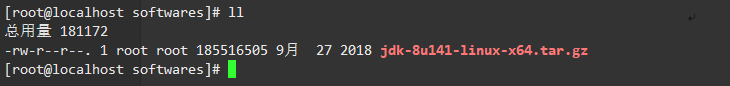
yum install lrzsz



cd /export/softwares

rz





tar -zxvf jdk-8u141-linux-x64.tar.gz -C ../servers/



## 配置环境变量

vim /etc/profile

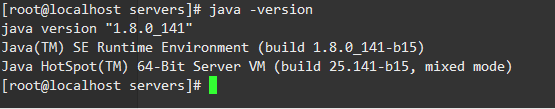
export JAVA\_HOME=/export/servers/jdk1.8.0\_141

export PATH=:$JAVA\_HOME/bin:$PATH



source /etc/profile



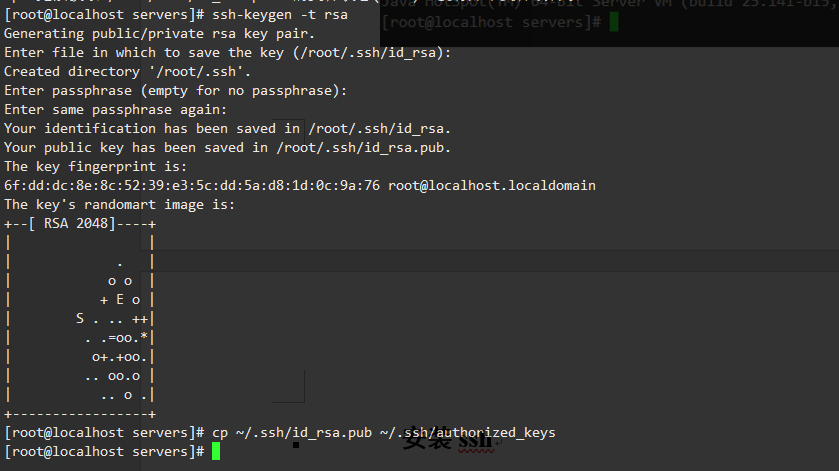


## 安装ssh

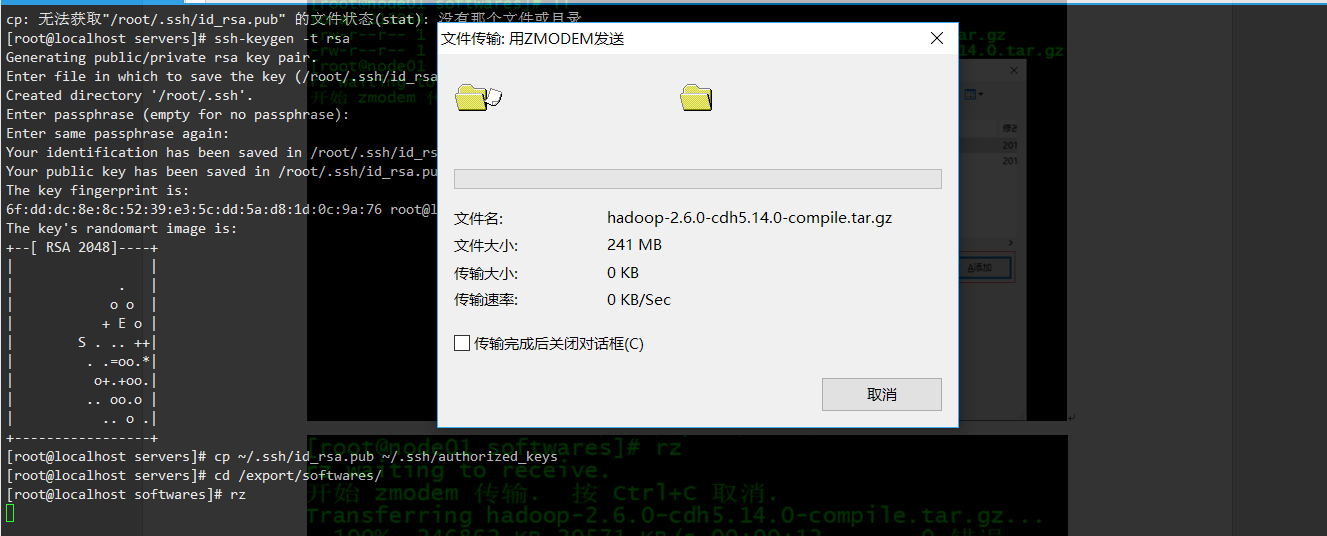
sudo yum install ssh

ssh-keygen -t rsa

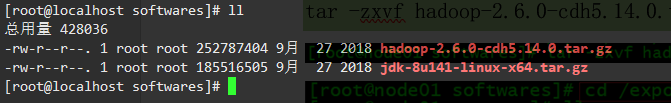
cp ~/.ssh/id\_rsa.pub ~/.ssh/authorized\_keys



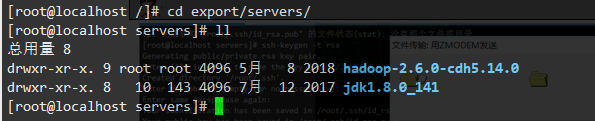
cd /export/softwares/  
rz



mv hadoop-2.6.0-cdh5.14.0-compile.tar.gz hadoop-2.6.0-cdh5.14.0.tar.gz



tar -zxvf hadoop-2.6.0-cdh5.14.0.tar.gz -C ../servers/



cd /export/servers/hadoop-2.6.0-cdh5.14.0

bin/hadoop checknative



## 修改core-site.xml

cd /export/servers/hadoop-2.6.0-cdh5.14.0/etc/hadoop

vim core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://node01:8020</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/tempDatas</value>

</property>

<property>

<name>io.file.buffer.size</name>

<value>4096</value>

</property>

<property>

<name>fs.trash.interval</name>

<value>10080</value>

</property>

</configuration>

## 修改hdfs-site.xml

cd /export/servers/hadoop-2.6.0-cdh5.14.0/etc/hadoop

vim hdfs-site.xml

<configuration>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>node01:50090</value>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>node01:50070</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/namenodeDatas</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/datanodeDatas</value>

</property>

<property>

<name>dfs.namenode.edits.dir</name>

<value>file:///export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/dfs/nn/edits</value>

</property>

<property>

<name>dfs.namenode.checkpoint.dir</name>

<value>file:///export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/dfs/snn/name</value>

</property>

<property>

<name>dfs.namenode.checkpoint.edits.dir</name>

<value>file:///export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/dfs/nn/snn/edits</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.permissions</name>

<value>false</value>

</property>

<property>

<name>dfs.blocksize</name>

<value>134217728</value>

</property>

</configuration>

## 修改hadoop-env.sh

cd /export/servers/hadoop-2.6.0-cdh5.14.0/etc/hadoop

vim hadoop-env.sh

export JAVA\_HOME=/export/servers/jdk1.8.0\_141

## 修改mapred-site.xml

cd /export/servers/hadoop-2.6.0-cdh5.14.0/etc/hadoop

vim mapred-site.xml

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<property>

<name>mapreduce.job.ubertask.enable</name>

<value>true</value>

</property>

<property>

<name>mapreduce.jobhistory.address</name>

<value>node01:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>node01:19888</value>

</property>

</configuration>

## 修改yarn-site.xml

cd /export/servers/hadoop-2.6.0-cdh5.14.0/etc/hadoop

vim yarn-site.xml

<configuration>

<property>

<name>yarn.resourcemanager.hostname</name>

<value>node01</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<name>yarn.log-aggregation.retain-seconds</name>

<value>604800</value>

</property>

</configuration>

## 修改slaves文件

cd /export/servers/hadoop-2.6.0-cdh5.14.0/etc/hadoop

vim slaves

## 创建文件存放目录

mkdir -p /export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/tempDatas

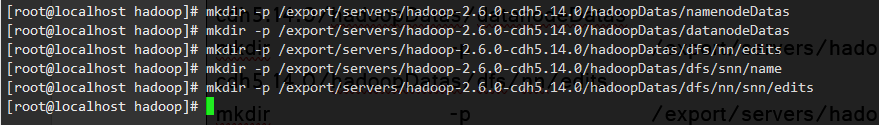
mkdir -p /export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/namenodeDatas

mkdir -p /export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/datanodeDatas

mkdir -p /export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/dfs/nn/edits

mkdir -p /export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/dfs/snn/name

mkdir -p /export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/dfs/nn/snn/edits

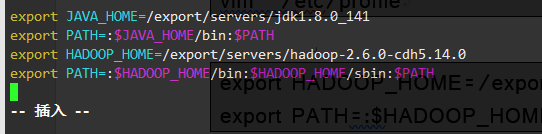


## 配置hadoop的环境变量

vim /etc/profile

export HADOOP\_HOME=/export/servers/hadoop-2.6.0-cdh5.14.0

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



source /etc/profile

## 集群启动

要启动 Hadoop 集群，需要启动 HDFS 和 YARN 两个集群。

注意：首次启动HDFS时，必须对其进行格式化操作。本质上是一些清理和准备工作，因为此时的 HDFS 在物理上还是不存在的。

cd /export/servers/hadoop-2.6.0-cdh5.14.0/

bin/hdfs namenode -format

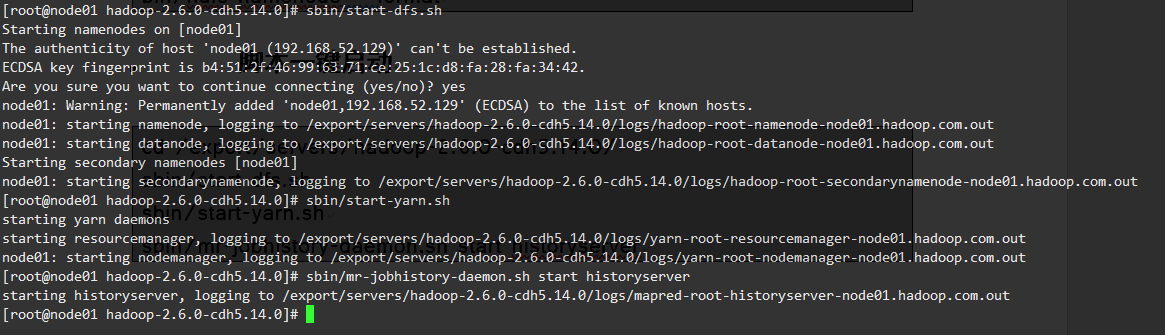
## 脚本一键启动

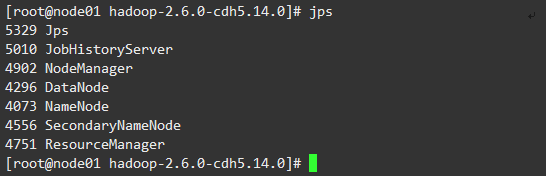
cd /export/servers/hadoop-2.6.0-cdh5.14.0/

sbin/start-dfs.sh

sbin/start-yarn.sh

sbin/mr-jobhistory-daemon.sh start historyserver





## 停止集群

sbin/stop-dfs.sh

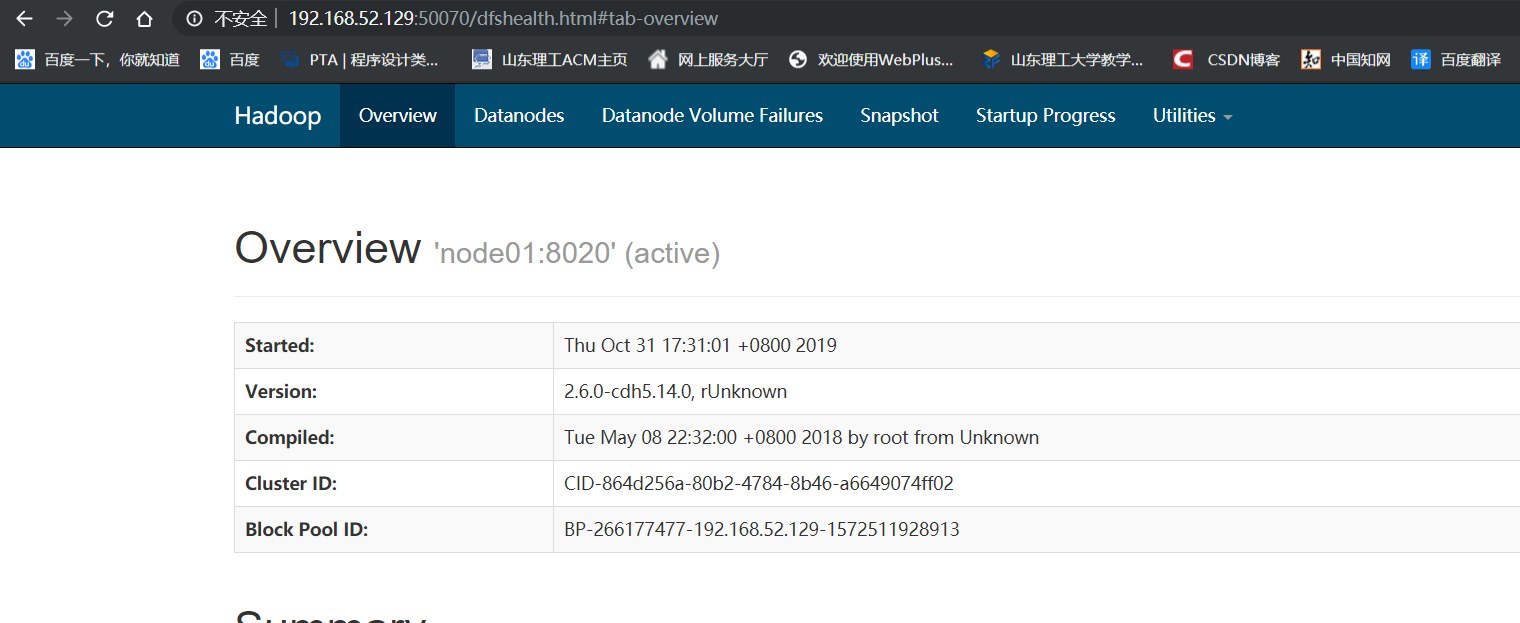
sbin/stop-yarn.sh

sbin/mr-jobhistory-daemon.sh stop historyserver

## 浏览器查看启动页面

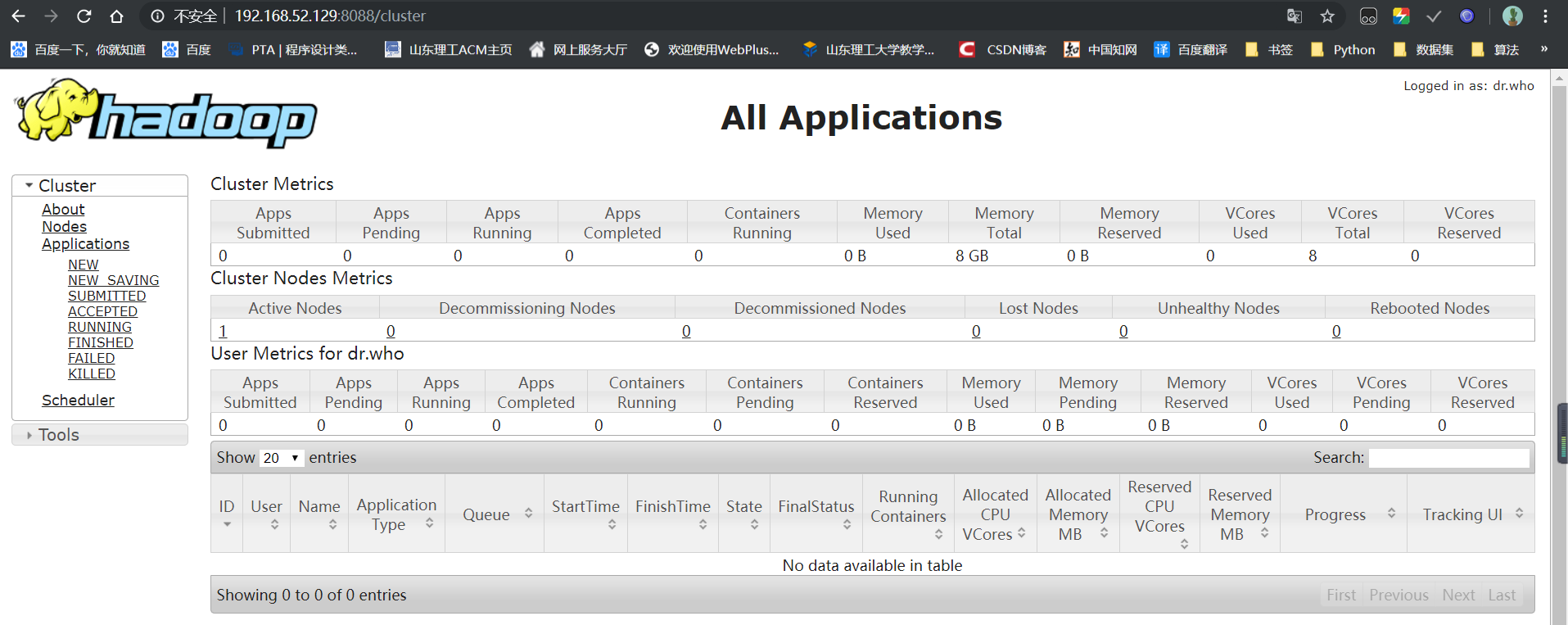
### hdfs集群访问地址

<http://192.168.52.129:50070/dfshealth.html#tab-overview>



### yarn集群访问地址

<http://192.168.52.129:8088/cluster>



### jobhistory访问地址

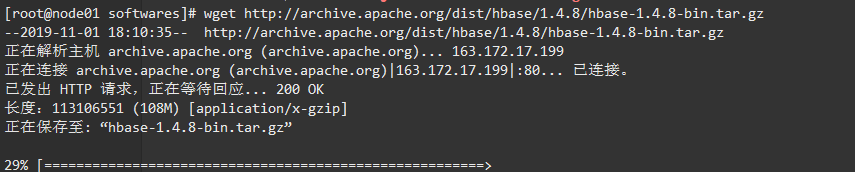
<http://192.168.52.129:19888/jobhistory>



# Hbase安装

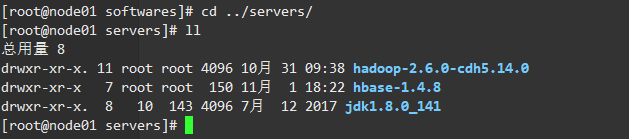
## 使用wget下载Hbase

wget http://archive.apache.org/dist/hbase/1.4.8/hbase-1.4.8-bin.tar.gz



## 解压目录

tar -zxvf hbase-1.4.8-bin.tar.gz -C ../servers/



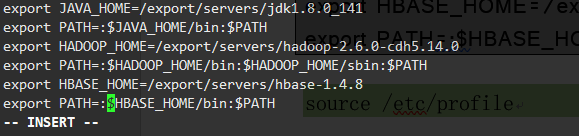
mkdir /export/servers/hbase-1.4.8/zk\_data

## 设置hbase环境变量

vi /etc/profile

export HBASE\_HOME=/export/servers/hbase-1.4.8

export PATH=:$HBASE\_HOME/bin:$PATH



source /etc/profile

## 配置 hbase-env.sh文件

vi /export/servers/hbase-1.4.8/conf/hbase-env.sh

export JAVA\_HOME=/export/servers/jdk1.8.0\_141/

export HBASE\_MANAGES\_ZK=true #此配置信息，设置由hbase自己管理zookeeper，不需要单独的zookeeper

export HBASE\_PID\_DIR=/export/servers/hbase-1.4.8/pids

## 配置 hbase-site.xml

vi /export/servers/hbase-1.4.8/conf/hbase-site.xml



<configuration>

<property>

<name>hbase.rootdir</name>

<value>hdfs://node01:8020/hbase</value>

</property>

<property>

<name>hbase.cluster.distributed</name>

<value>true</value>

</property>

<property>

<name>hbase.zookeeper.quorum</name>

<value>node01</value>

</property>

<property>

<name>hbase.zookeeper.property.dataDir</name>

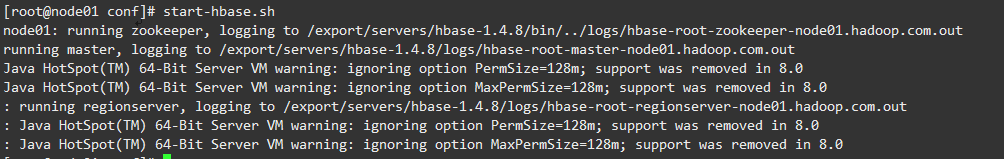
<value>/export/servers/hbase-1.4.8/zk\_data</value>

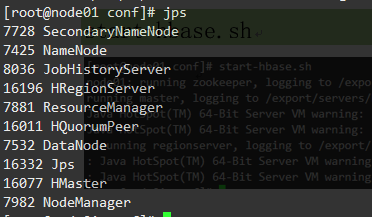
</property>

</configuration>

## 启动hbase

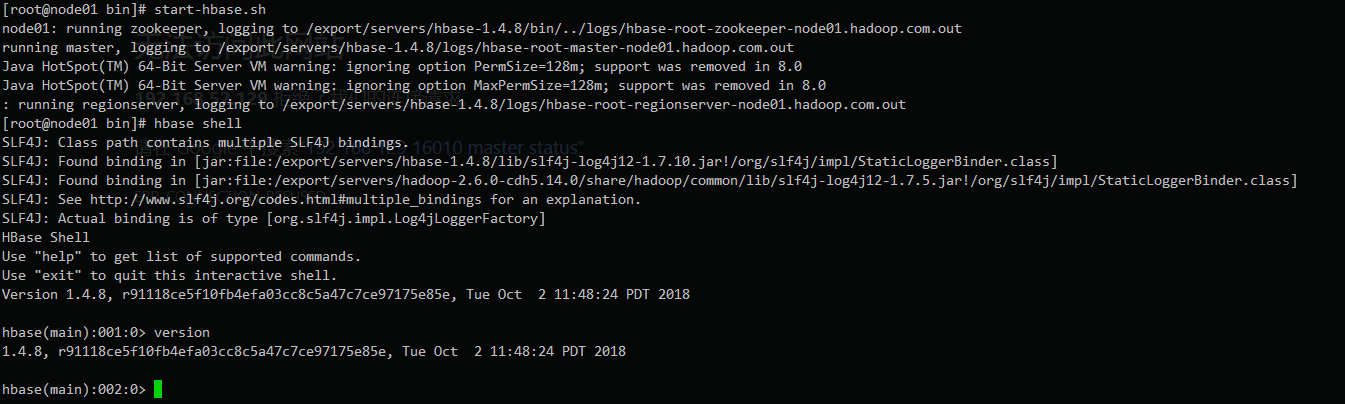
start-hbase.sh





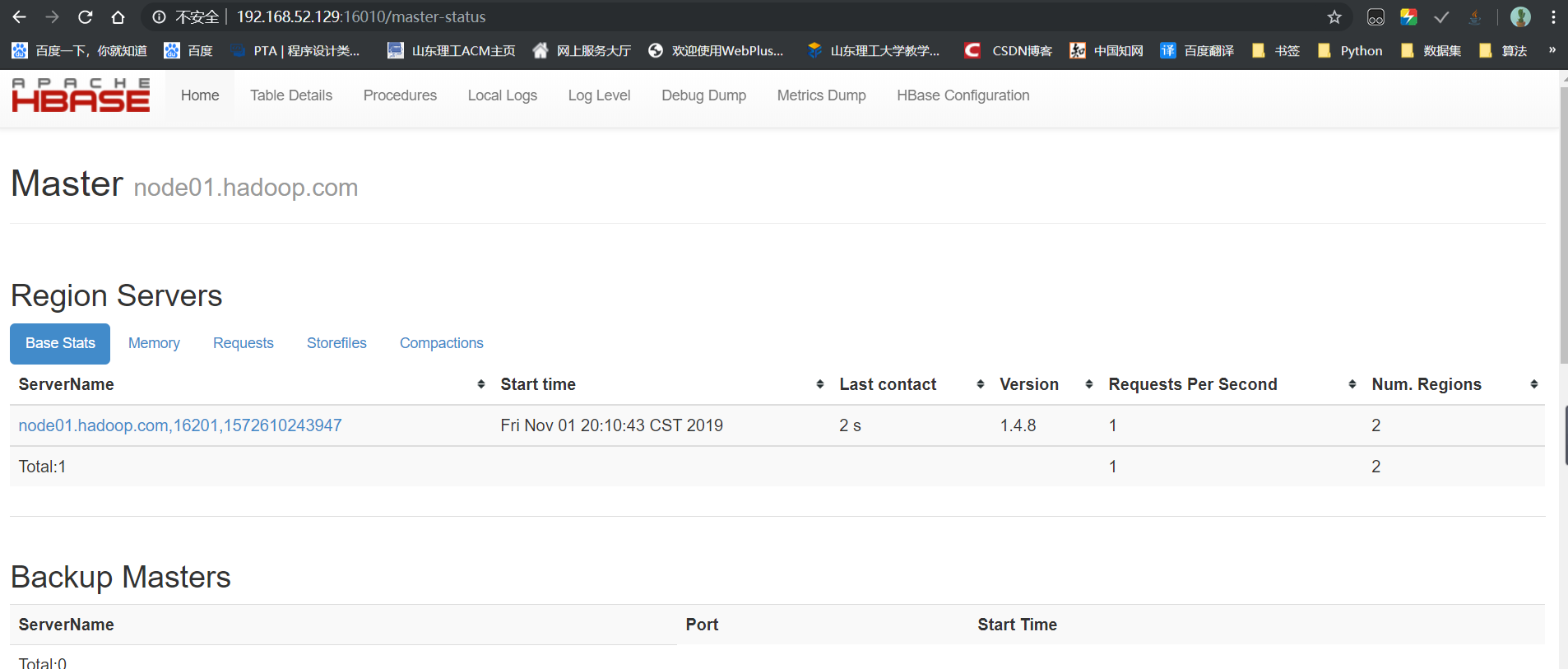
## 进入hbase shell

hbase shell

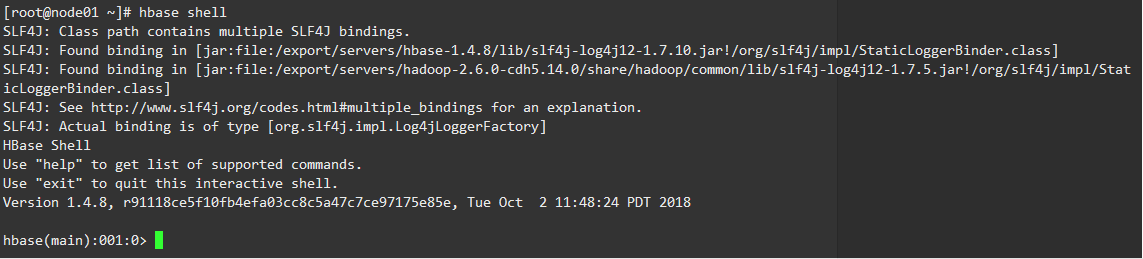


## 进入hbase的web页面

<http://192.168.52.129:16010/>



# HBase shell



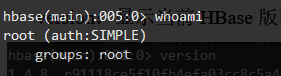
version：显示当前HBase版本号



status：显示各个主节点状态



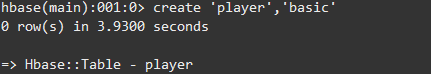
whoami：显示当前用户名



# 表和列族操作

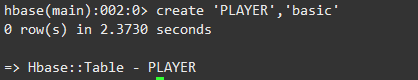
## 创建表

create 'player','basic'



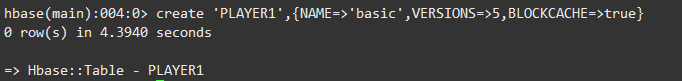
大小写参数敏感

create 'PLAYER','basic'



建表时指定列族的参数

create 'PLAYER1',{NAME=>'basic',VERSIONS=>5,BLOCKCACHE=>true}

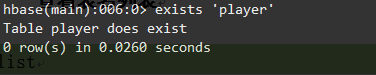


## 查看表名列表

list

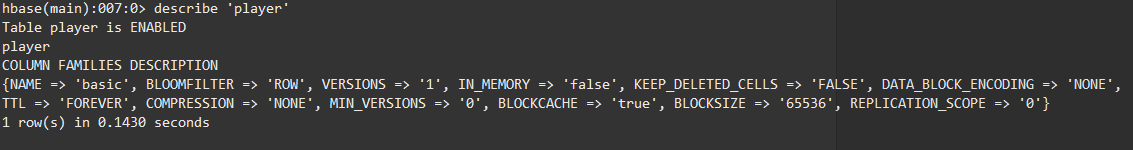


exists 'player'

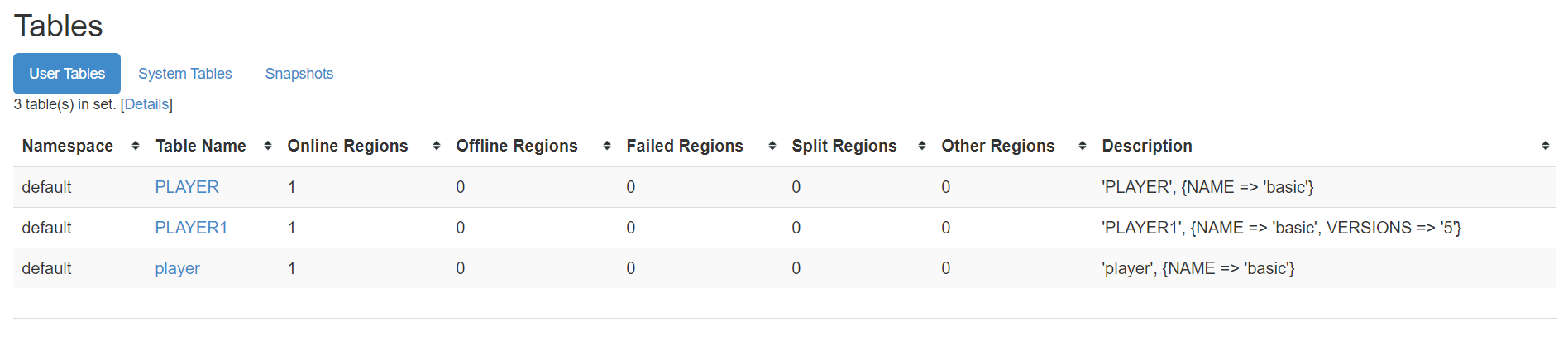


## 描述表结构

describe 'player'



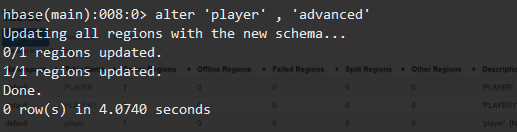
Hbase Web界面



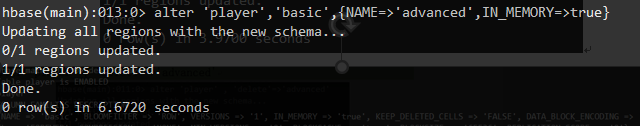
## 修改表结构

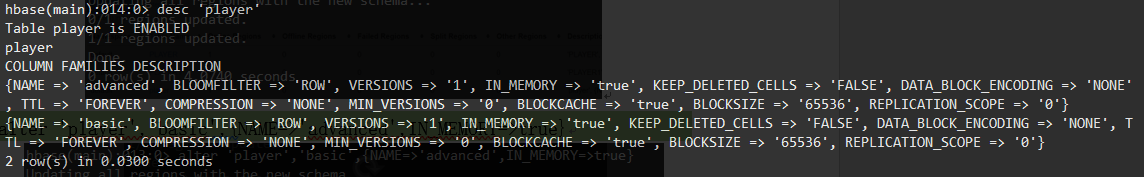
增加列族

alter 'player','advanced'



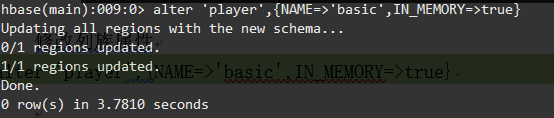
alter 'player','basic',{NAME=>'advanced',IN\_MEMORY=>true}

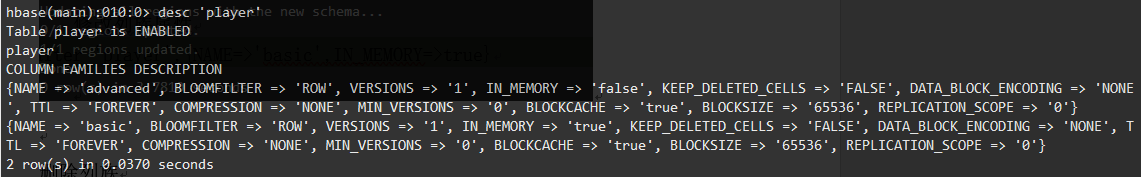




修改列族属性

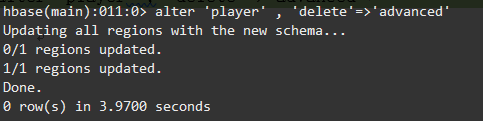
alter 'player',{NAME=>'basic',IN\_MEMORY=>true}

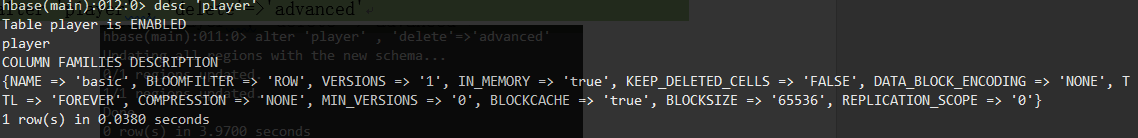




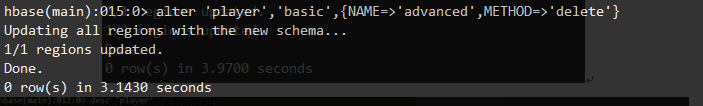
删除列族

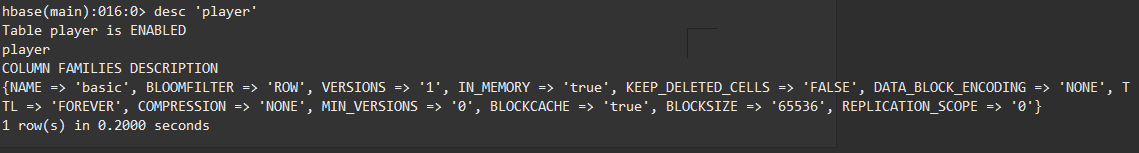
alter 'player' , 'delete'=>'advanced'





alter 'player','basic',{NAME=>'advanced',METHOD=>'delete'}



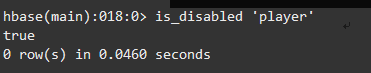


## 删除表

disable 'player'



is\_disabled 'player'



drop 'player'



数据清空

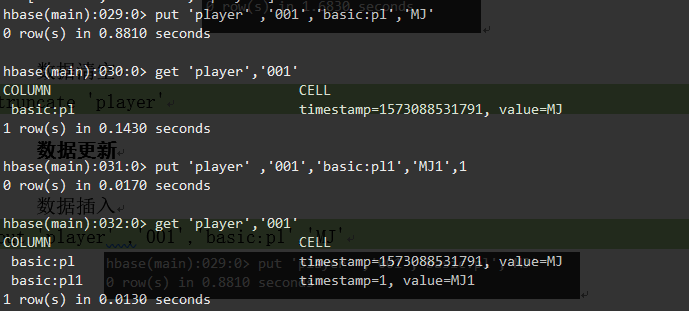
truncate 'player'

## 数据更新

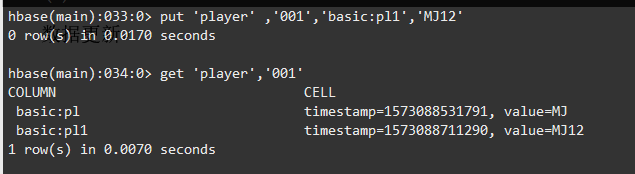
数据插入

put 'player' ,'001','basic:pl','MJ'



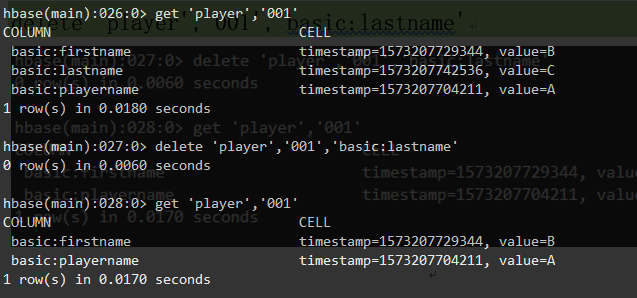


数据更新



数据删除

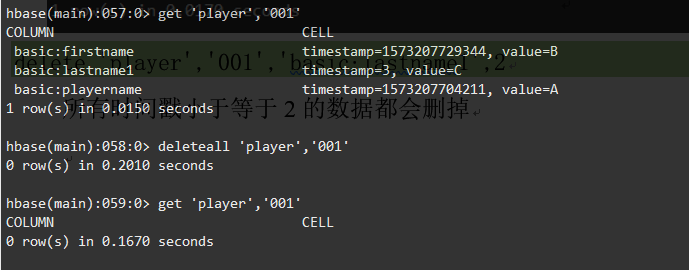
delete 'player','001','basic:lastname'



delete 'player','001','basic:lastname1',2

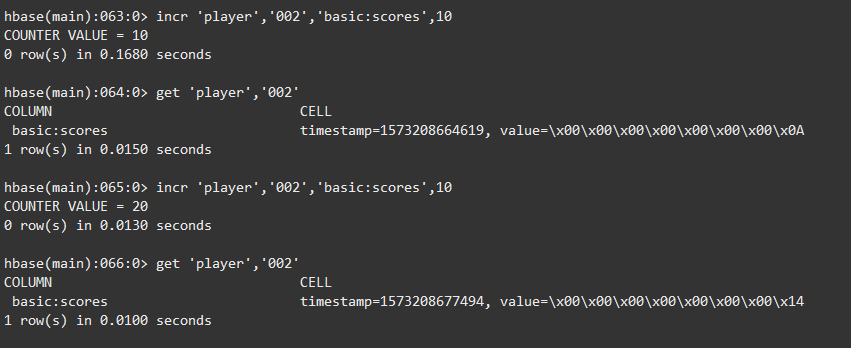
所有时间戳小于等于2的数据都会删掉

deleteall 'player','001'



## 计数器

incr 'player','002','basic:scores',10

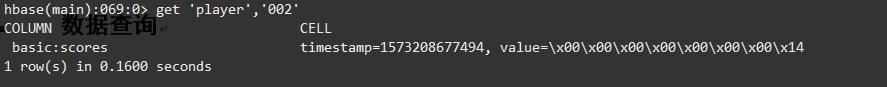


get\_counter 'player' ,'002','basic:scores'



## 数据查询

get 'player','002'



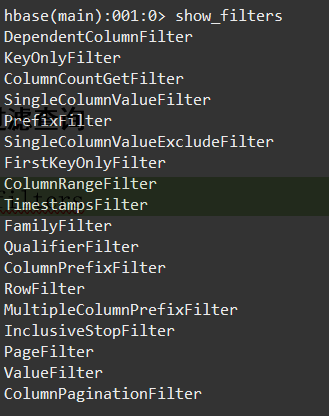
数据扫描

scan 'player'



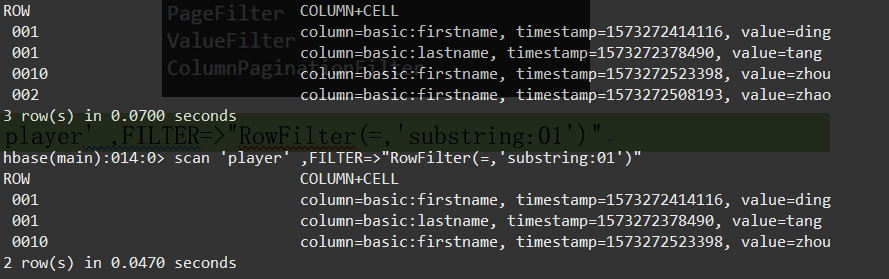
## 过滤查询

show\_filters

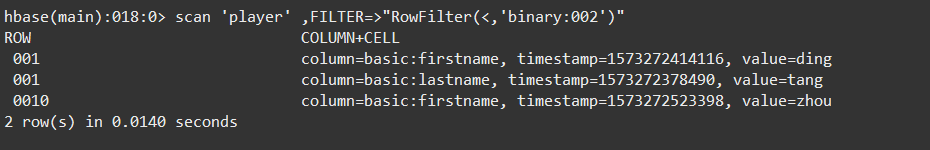


### 行键过滤器

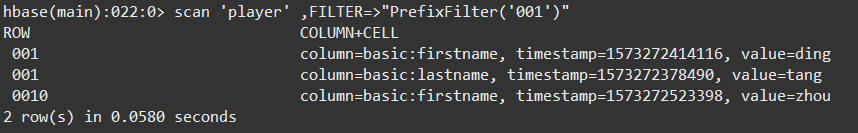
scan 'player' ,FILTER=>"RowFilter(=,'substring:01')"



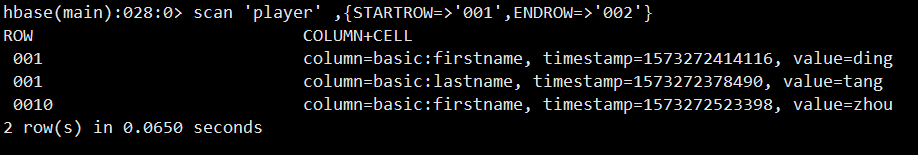
scan 'player' ,FILTER=>"RowFilter(<,'binary:002')"



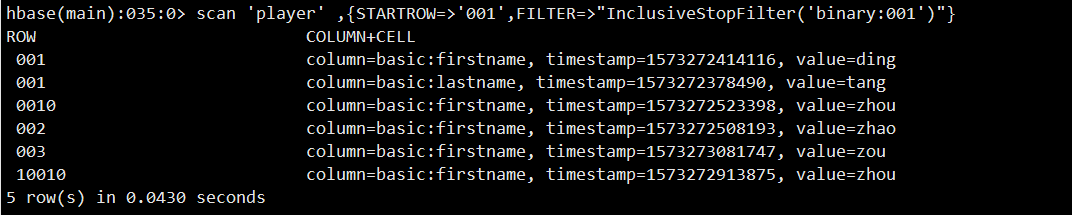
scan 'player' ,FILTER=>"PrefixFilter('001')"



scan 'player' ,{STARTROW=>'001',ENDROW=>'002'}

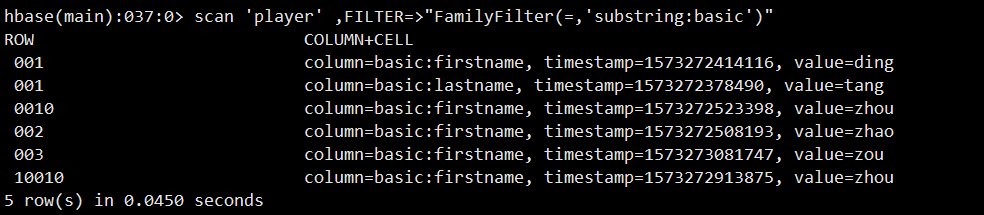


scan 'player' ,{STARTROW=>'001',FILTER=>"InclusiveStopFilter('binary:001')"}

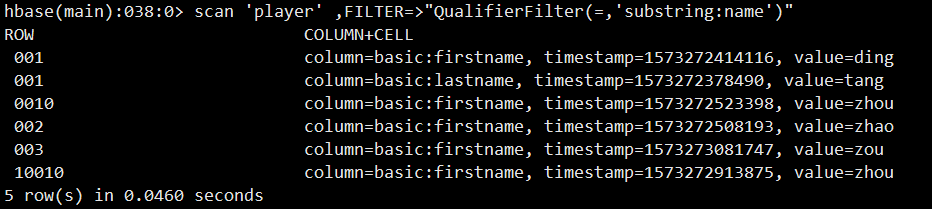


### 列族和列过滤器

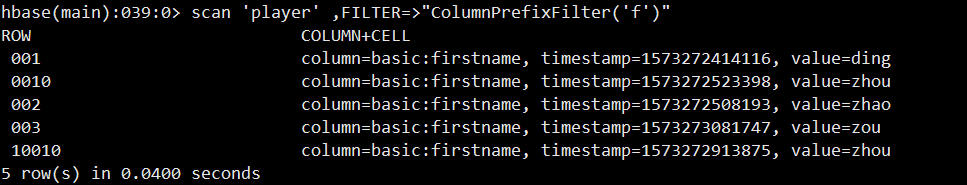
scan 'player' ,FILTER=>"FamilyFilter(=,'substring:basic')"



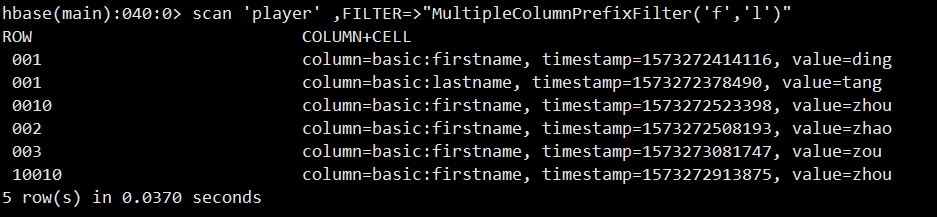
scan 'player' ,FILTER=>"QualifierFilter(=,'substring:name')"



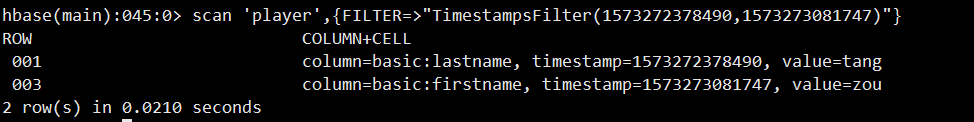
scan 'player' ,FILTER=>"ColumnPrefixFilter('f')"



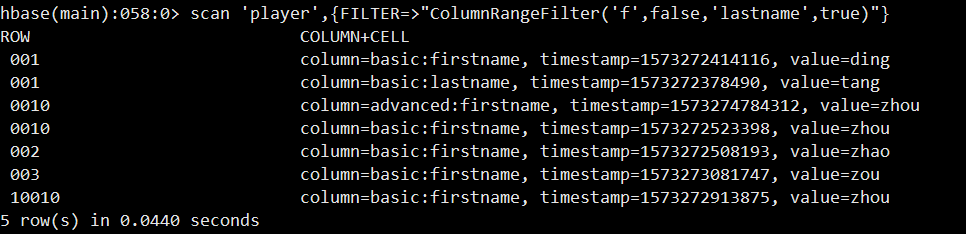
scan 'player' ,FILTER=>"MultipleColumnPrefixFilter('f','l')"



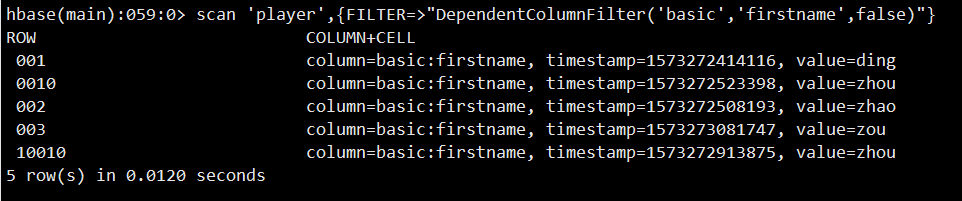
scan 'player',{FILTER=>"TimestampsFilter(1573272378490,1573273081747)"}



scan 'player',{FILTER=>"ColumnRangeFilter('f',false,'lastname',true)"}

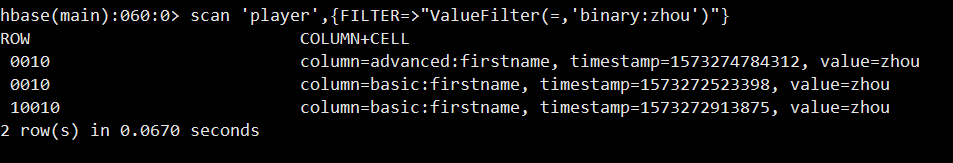


scan 'player',{FILTER=>"DependentColumnFilter('basic','firstname',false)"}

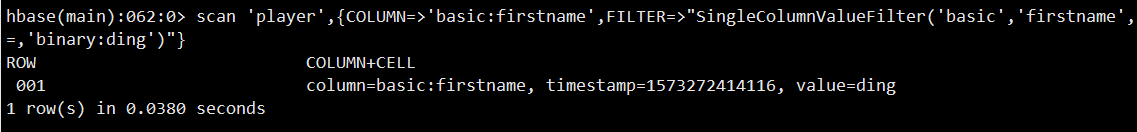


### 值过滤器

scan 'player',{FILTER=>"ValueFilter(=,'binary:zhou')"}

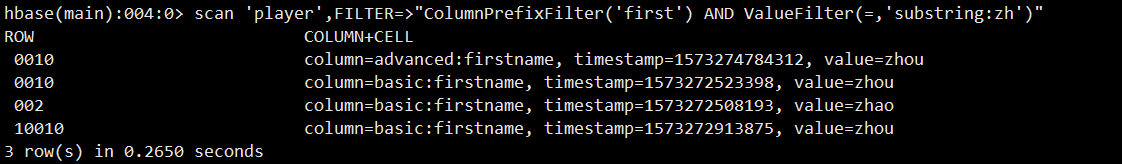


scan 'player',{COLUMN=>'basic:firstname',FILTER=>"SingleColumnValueFilter('basic','firstname',=,'binary:ding')"}



## 其他过滤器

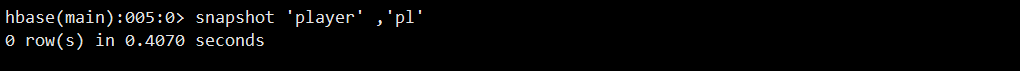
scan 'player',FILTER=>"ColumnPrefixFilter('first') AND ValueFilter(=,'substring:zh')"



## 快照操作

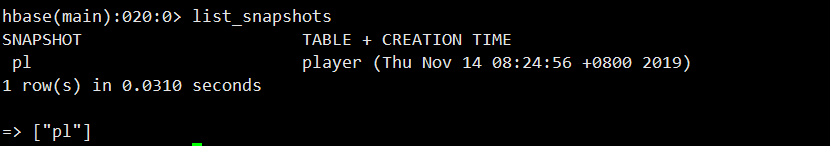
建立快照

snapshot 'player' ,'pl'



显示快照列表

list\_snapshots



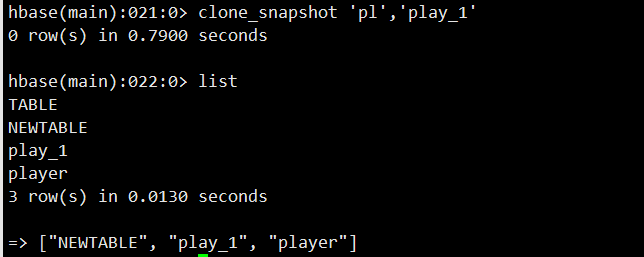
删除快照

delete\_snapshot 'pl'



通过快照生成新表

clone\_snapshot 'pl','play\_1'



# Java访问Hbase

pom.xml

<dependency>

<groupId>org.apache.hbase</groupId>

<artifactId>hbase-client</artifactId>

<version>1.4.8</version>

</dependency>

## 建立连接

public static Configuration conf;

public static Connection connection;

public void getconncet(){

conf= HBaseConfiguration.create();

conf.set("hbase.zookeeper.quorum","node");

conf.set("hbase.zookeeper.property.clientPort","2181");

conf.set("zookeeper.znode.parent","/hbase");

conf.set("hbase.master", "node:16000");

try{

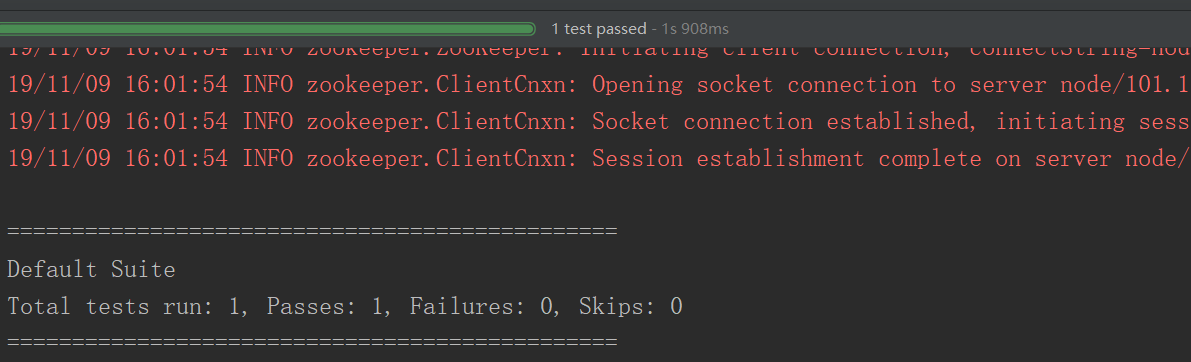
connection=ConnectionFactory.createConnection(conf);

} catch (IOException e) {

e.printStackTrace();

}

}



## 建立和删除表

public void createtable() throws IOException {

getconncet();

TableName tableName =TableName.valueOf("NEWTABLE");

Admin admin = connection.getAdmin();

if (admin.tableExists(tableName)){

admin.disableTable(tableName);

admin.deleteTable(tableName);

System.out.println(tableName.toString()+"is exists, delete it..............");

}

HTableDescriptor descriptor = new HTableDescriptor(tableName);

HColumnDescriptor columnDescriptor = new HColumnDescriptor("cf1");

columnDescriptor.setBloomFilterType(BloomType.ROWCOL);

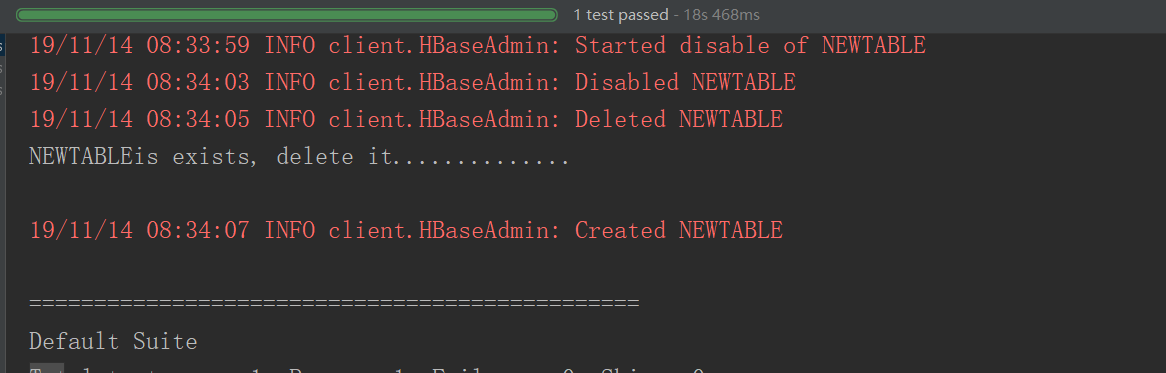
descriptor.addFamily(columnDescriptor);

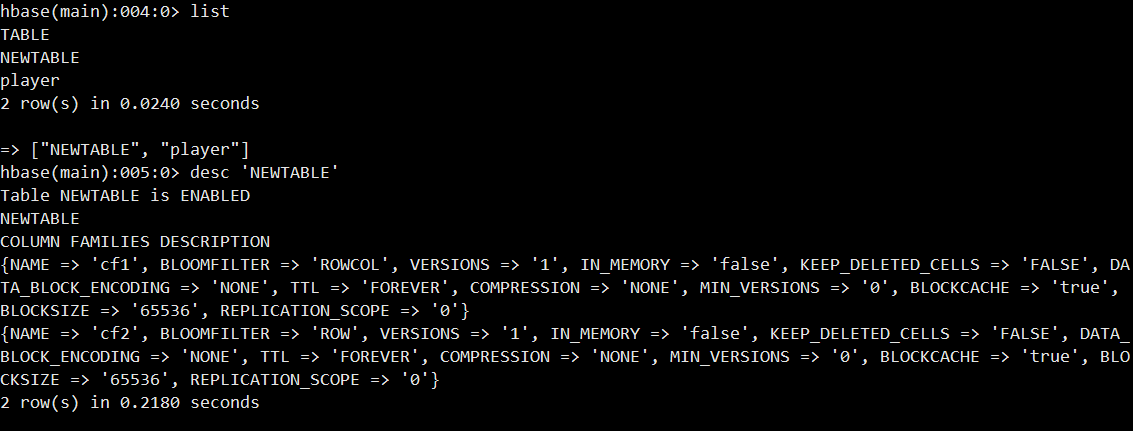
descriptor.addFamily(new HColumnDescriptor("cf2"));

admin.createTable(descriptor);

admin.close();

}





## 描述表结构

public void desctable() throws IOException {

getconncet();

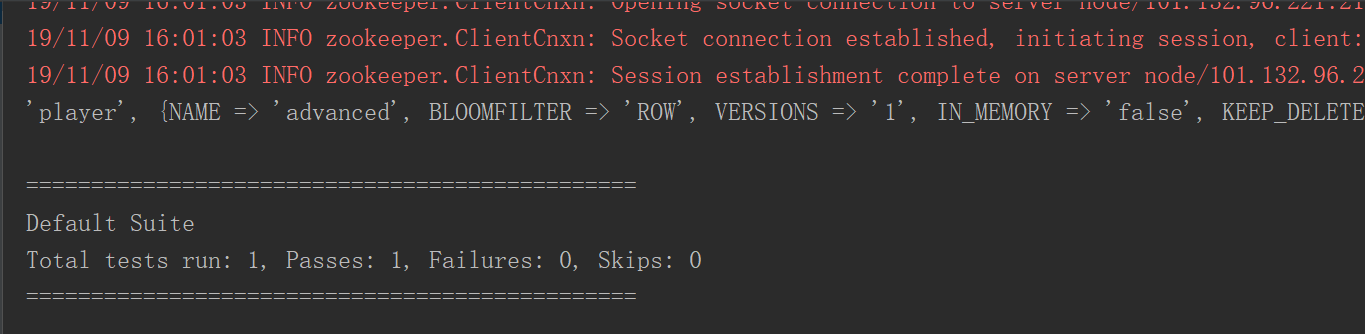
TableName tableName =TableName.valueOf("player");

Admin admin = connection.getAdmin();

HTableDescriptor descriptor=admin.getTableDescriptor(tableName);

System.out.println(descriptor.toString());

}



public void desctable() throws IOException {

getconncet();

TableName tableName =TableName.valueOf("NEWTABLE");

Admin admin = connection.getAdmin();

HTableDescriptor descriptor=admin.getTableDescriptor(tableName);

System.out.println(descriptor.toString());

System.out.println("table information:......................");

System.out.println("getNameAsString:"+descriptor.getNameAsString());

System.out.println("getMaxFileSize:"+descriptor.getMaxFileSize());

System.out.println("getMemStoreFlushSize:"+descriptor.getMemStoreFlushSize());

System.out.println("getRegionSplitPolicyClassName:"+descriptor.getRegionSplitPolicyClassName());

System.out.println("getRegionSplitPolicyClassName:"+descriptor.getRegionSplitPolicyClassName());

Collection<HColumnDescriptor>families=descriptor.getFamilies();

System.out.println("Column family infomation...................");

for (HColumnDescriptor result:families){

System.out.println("getNameAsString:"+result.getNameAsString());

System.out.println("getBloomFilterType:"+result.getBloomFilterType());

System.out.println("getBlocksize:"+result.getBlocksize());

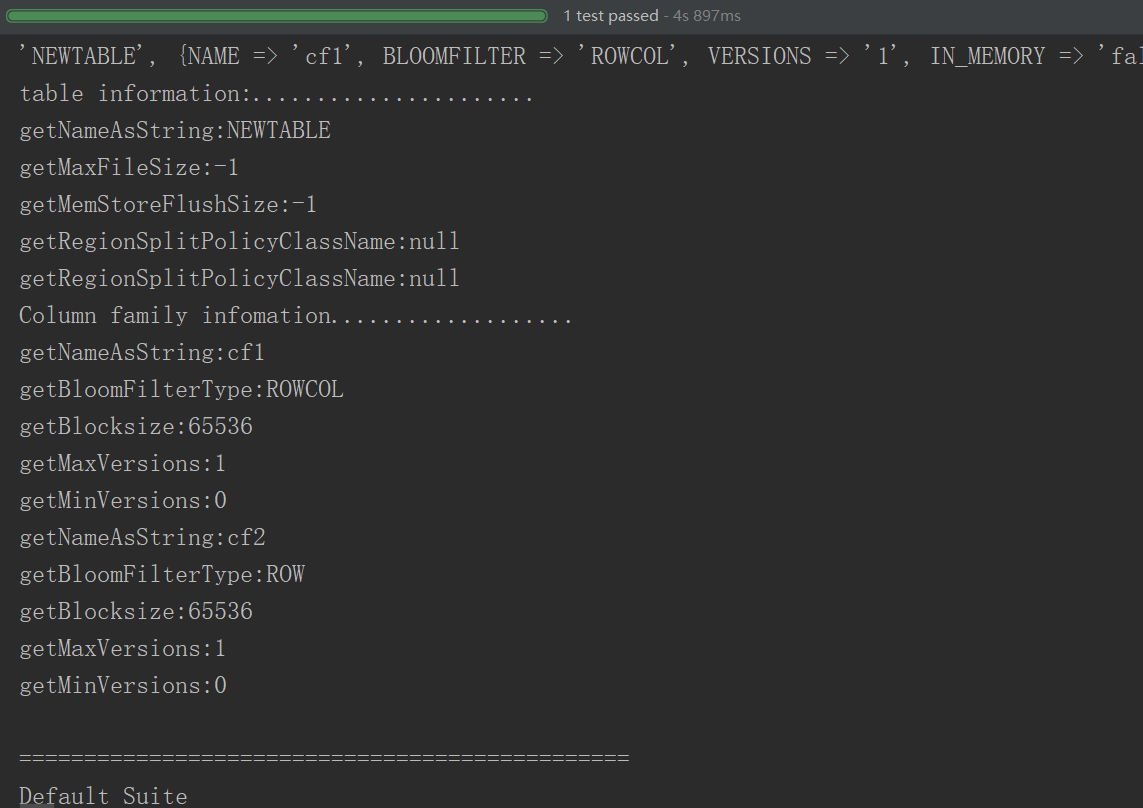
System.out.println("getMaxVersions:"+result.getMaxVersions());

System.out.println("getMinVersions:"+result.getMinVersions());

}

admin.close();

}



## 数据更新

public void addData() throws IOException{

getconncet();

HTable table=(HTable)connection.getTable(TableName.valueOf("NEWTABLE"));

table.setWriteBufferSize(6\*1024\*1024);

table.setAutoFlushTo(false);

Put put=new Put(Bytes.toBytes("row1"));

put.setDurability(Durability.SKIP\_WAL);

put.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col0"),Bytes.toBytes("value0"));

put.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col1"),Bytes.toBytes("value1"));

put.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col2"),Bytes.toBytes("value2"));

put.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col3"),Bytes.toBytes("value3"));

table.put(put);

table.flushCommits();

Put put2=new Put("row2".getBytes());

put2.setDurability(Durability.SKIP\_WAL);

put2.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col0"),Bytes.toBytes("value4"));

put2.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col4"),Bytes.toBytes("value5"));

put2.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col3"),Bytes.toBytes("value6"));

put2.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col5"),Bytes.toBytes("value7"));

table.put(put2);

table.flushCommits();

Put put3=new Put("row3".getBytes());

put3.setDurability(Durability.SKIP\_WAL);

put3.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col0"),Bytes.toBytes("value4"));

put3.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col6"),Bytes.toBytes("value8"));

put3.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col3"),Bytes.toBytes("value9"));

put3.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col7"),Bytes.toBytes("value10"));

Put put4=new Put("row4".getBytes());

put4.setDurability(Durability.SKIP\_WAL);

put4.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col0"),Bytes.toBytes("value11"));

put4.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col8"),Bytes.toBytes("value8"));

put4.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col3"),Bytes.toBytes("value9"));

put4.addColumn(Bytes.toBytes("cf2"),Bytes.toBytes("col9"),Bytes.toBytes("value12"));

List<Put>putList=new ArrayList<Put>();

putList.add(put3);

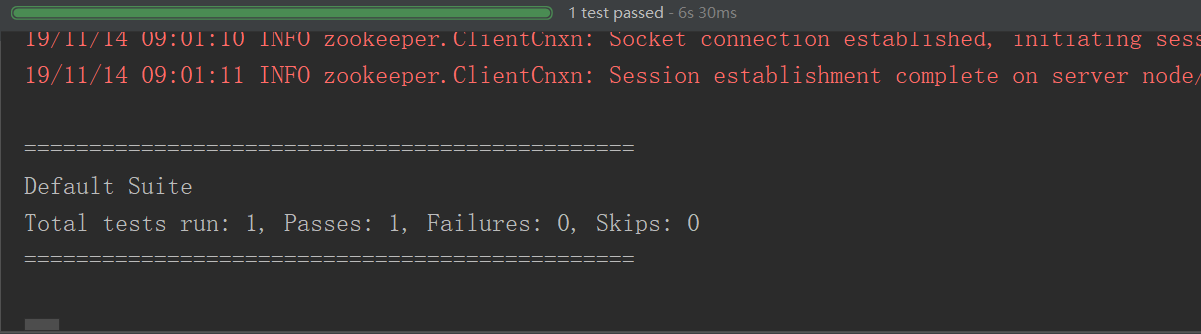
putList.add(put4);

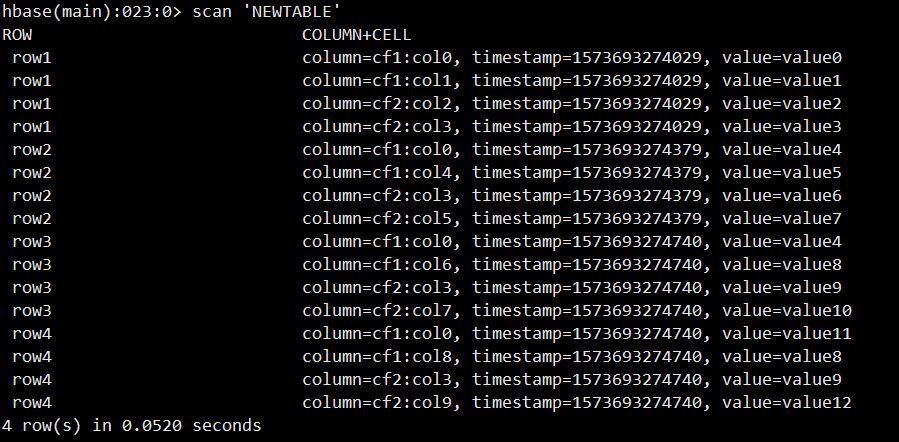
table.put(putList);

table.flushCommits();

table.close();

}





## 数据查询

get方法

private void getData() throws IOException{

getconncet();

Table table =connection.getTable(TableName.valueOf("NEWTABLE"));

Get get=new Get(Bytes.toBytes("row1"));

Result result=table.get(get);

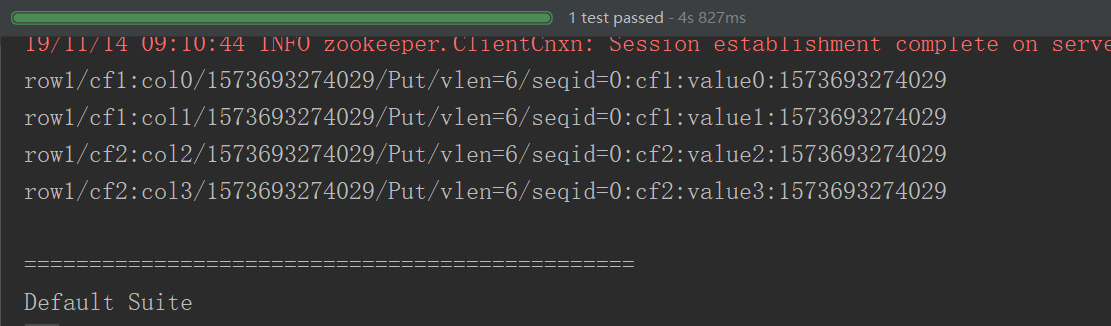
for (Cell cell:result.rawCells()){

System.out.println(new String(CellUtil.getCellKeyAsString(cell))+":"+new String(CellUtil.cloneFamily(cell))+":"+new String(CellUtil.cloneValue(cell))+":"+cell.getTimestamp());

}

table.close();

}



scan方法

private void ScanData()throws IOException{

getconncet();

Table table =connection.getTable(TableName.valueOf("NEWTABLE"));

Scan scan=new Scan();

ResultScanner results=table.getScanner(scan);

for (Result result:results){

for (Cell cell:result.rawCells()){

System.out.println(new String(CellUtil.getCellKeyAsString(cell))+":"+new String(CellUtil.cloneFamily(cell))+":"+new String(CellUtil.cloneValue(cell))+":"+cell.getTimestamp());

}

}

}



## 删除行和列

删除列族和列

private void removecol()throws IOException{

getconncet();

Admin admin=connection.getAdmin();

HTableDescriptor descriptor=admin.getTableDescriptor(TableName.valueOf("NEWTABLE"));

TableName tableName =TableName.valueOf("NEWTABLE");

descriptor.removeFamily(Bytes.toBytes("col0"));

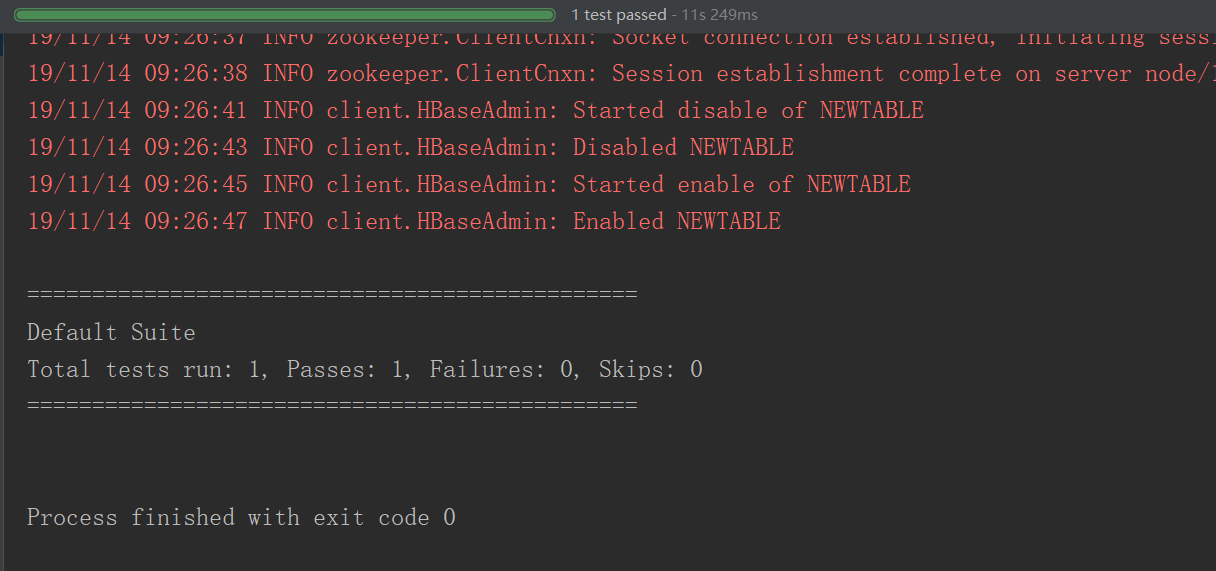
admin.disableTable(tableName);

admin.deleteColumn(tableName, Bytes.toBytes("cf2"));

admin.enableTable(tableName);

admin.close();

}



删除行和键值对

public void deleteRow()throws IOException{

getconncet();

HTable table=null;

try {

table=(HTable)connection.getTable(TableName.valueOf("NEWTABLE"));

Delete delete1=new Delete(Bytes.toBytes("row1"));

Delete delete2=new Delete(Bytes.toBytes("row2"));

Delete delete3=new Delete(Bytes.toBytes("row3"));

delete2.addFamily(Bytes.toBytes("cf1"));

delete3.addColumn(Bytes.toBytes("cf1"),Bytes.toBytes("col6"));

table.delete(delete1);

table.delete(delete2);

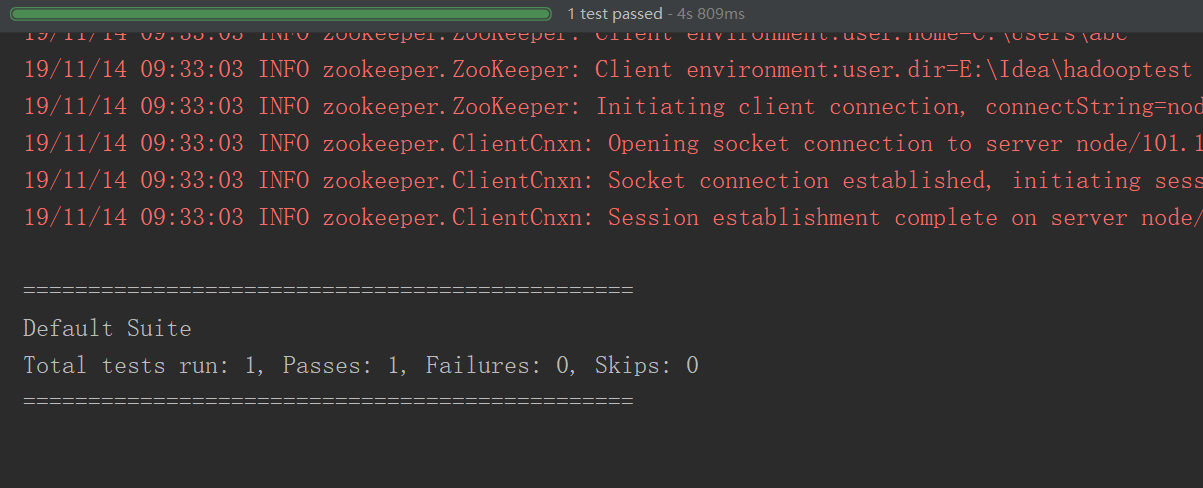
table.delete(delete3);

table.close();

}catch (Exception e){

}

}



## 过滤器

public void filter()throws IOException{

getconncet();

TableName tableName =TableName.valueOf("NEWTABLE");

Table table=connection.getTable(tableName);

Scan scan=new Scan();

FilterList filterList=new FilterList(FilterList.Operator.MUST\_PASS\_ALL);

filterList.addFilter(new RowFilter(CompareFilter.CompareOp.LESS,new BinaryComparator(Bytes.toBytes("row3"))));

filterList.addFilter(new KeyOnlyFilter());

scan.setFilter(filterList);

ResultScanner results=table.getScanner(scan);

for (Result result:results){

for (Cell cell:result.rawCells()){

System.out.println(new String(CellUtil.getCellKeyAsString(cell))+":"+new String(CellUtil.cloneFamily(cell))+":"+new String(CellUtil.cloneValue(cell))+":"+cell.getTimestamp());

}

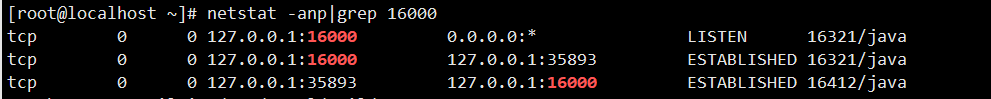
}

}

## 解决Java API不能远程访问HBase的问题

查看发现HBase绑定的是本地IP：127.0.0.1，这当然访问不了

netstat -anp|grep 16000



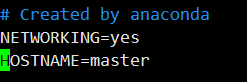
配置Linux的hostname

vim /etc/sysconfig/network

NETWORKING=yes

HOSTNAME=master

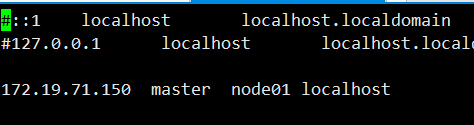
这里配置的hostname要Linux重启才生效，为了不重启就生效，我们可以执行：hostname master命令，暂时设置hostname



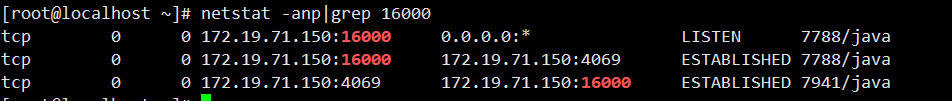
配置Linux的hosts，映射ip的hostname的关系

vi /etc/hosts

172.19.71.150 master



netstat -anp|grep 16000



配置访问windows的hosts

路径为：C:\Windows\System32\drivers\etc\hosts

172.19.71.150 master

配置完这三项Java API就可以远程访问HBase了，切记最后配置windows的hosts也是必须的

# Python 访问Hbase

## CentOS安装Thrift

### 安装依赖

yum -y install automake libtool flex bison pkgconfig gcc-c++ boost-devel libevent-devel zlib-devel python-devel ruby-devel openssl-devel

### 安装boost包

cd /export/softwares/

wget <http://sourceforge.net/projects/boost/files/boost/1.53.0/boost_1_53_0.tar.gz>

tar xvf boost\_1\_53\_0.tar.gz

cd boost\_1\_53\_0

./bootstrap.sh

./b2 install

### 安装thrift

升级bison:

wget http://ftp.gnu.org/gnu/bison/bison-2.5.1.tar.gz

tar xvf bison-2.5.1.tar.gz

cd bison-2.5.1

./configure

make

make install

wget <http://mirrors.hust.edu.cn/apache/thrift/0.9.3/thrift-0.9.3.tar.gz>

tar xzvf thrift-0.9.3.tar.gz

cd thrift-0.9.3

./configure

make

make install

### 验证是否安装成功

thrift –version

## 打开HBase的Thrift服务

hbase-daemon.sh start thrift

hbase-daemon.sh start thrift2

## 客户端配置Python环境pip安装Thrift

conda create --name nosql python=3.7

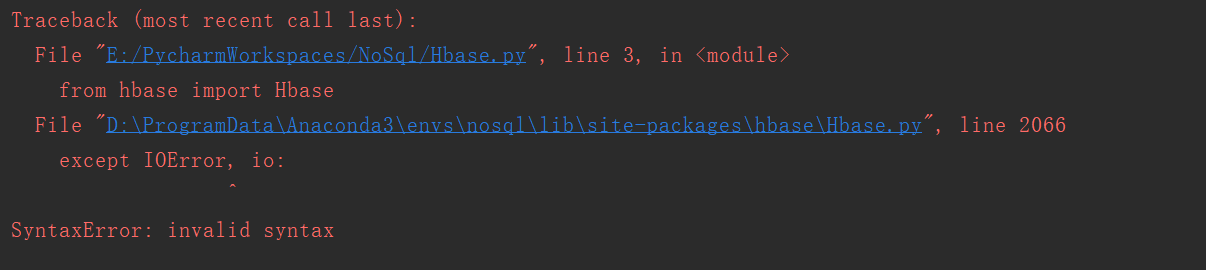
conda activate nosql

conda install thrift

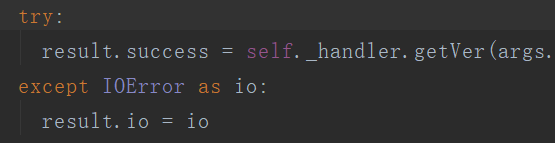
pip install hbase-thrift

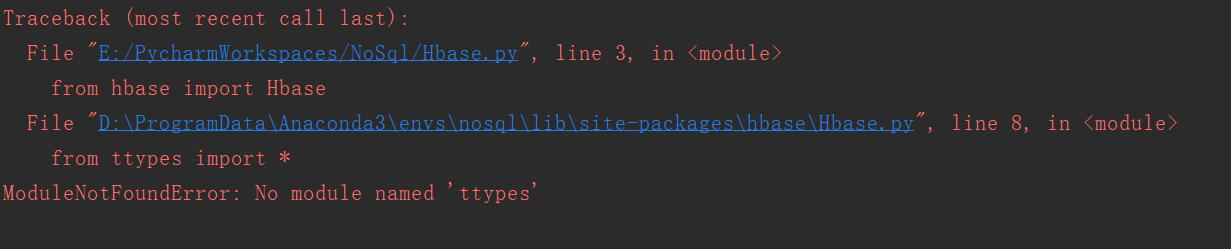
## 修改代码文件

将Python2风格代码改为Python3

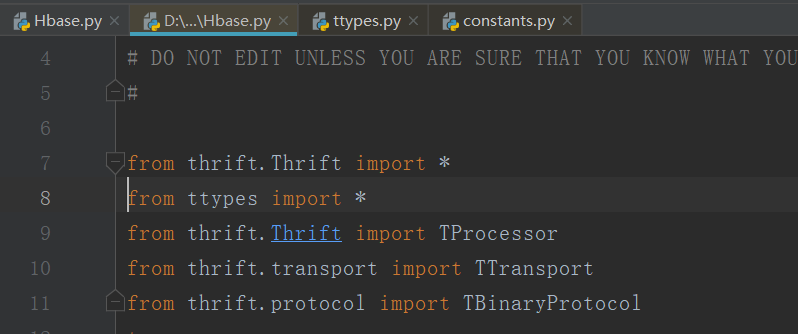
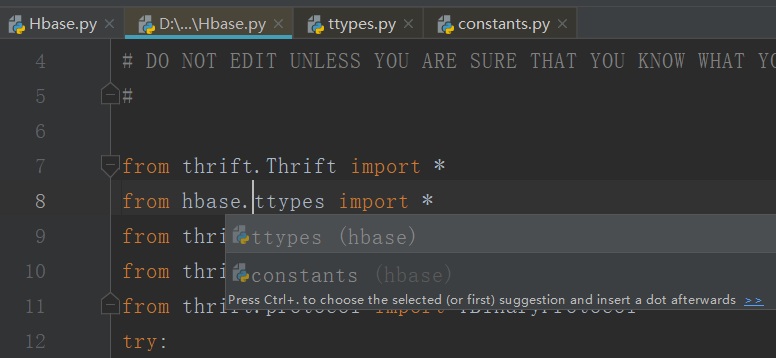


改为



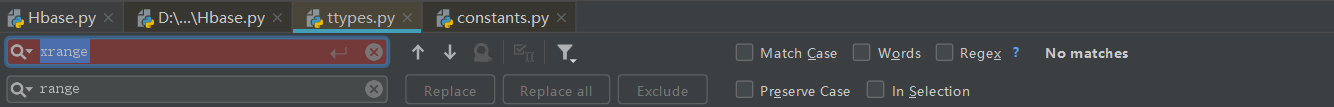


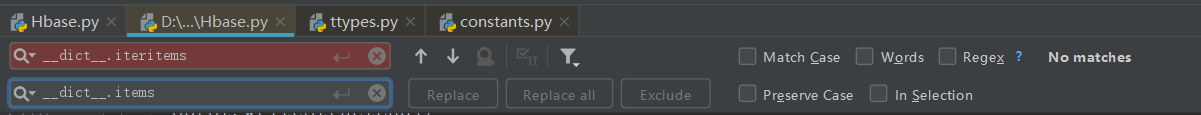
改为

替换处理









## 引用的类库

from thrift.transport import TSocket

from thrift.protocol import TBinaryProtocol

from hbase.Hbase import \*

from hbase.ttypes import \*

## 建立连接

transport = TSocket.TSocket("192.168.52.100",9090)

protocol = TBinaryProtocol.TBinaryProtocol(transport)

client=Client(protocol)

try:

transport.open()

except Exception as e:

print(e)

exit()

# 关闭连接

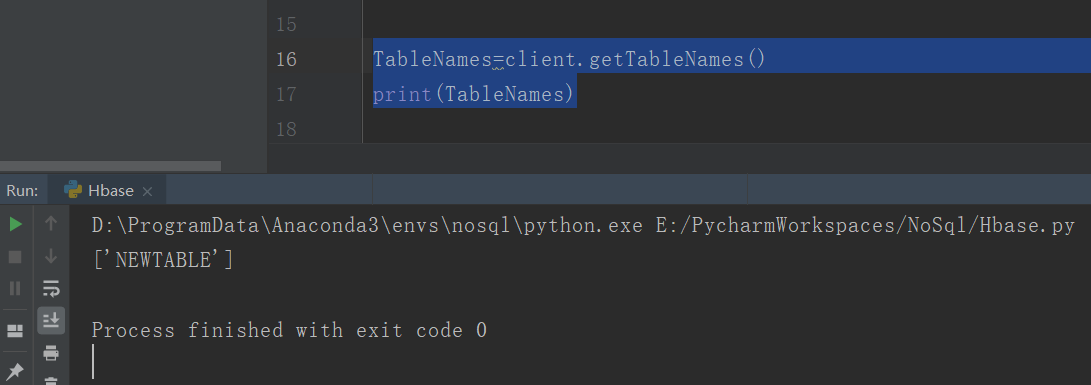
transport.close()



## 列举所有表名

TableNames=client.getTableNames()

print(TableNames)



## 表的建立

content1=ColumnDescriptor(name='cf1',maxVersions=1)

content2=ColumnDescriptor(name='cf2',)

client.createTable('testtable',[content1,content2])



## 表的禁用删除

try:

client.disableTable('testtable')

client.deleteTable('testtable')

except:

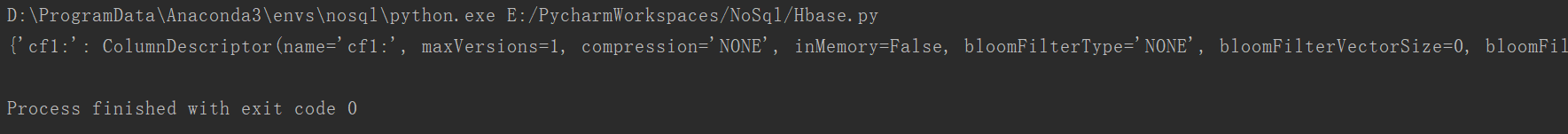
pass



## 查看表结构

ColumnDescriptors=client.getColumnDescriptors('testtable')

print(ColumnDescriptors)



## 插入数据

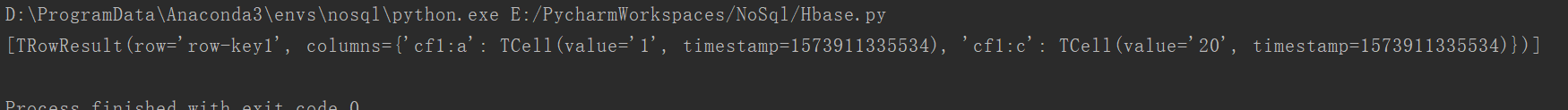
mutations=[Mutation(column='cf1:a',value='1'),Mutation(column='cf1:c',value='20')]

client.mutateRow('testtable','row-key1',mutations)

## 检索数据

result=client.getRow('testtable','row-key1')

print(result)



for r in result:

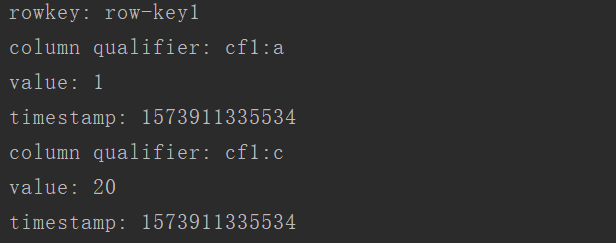
print('rowkey:',r.row)

for c in r.columns.keys():

print("column qualifier:",c)

print('value:',r.columns[c].value)

print('timestamp:', r.columns[c].timestamp)



## 删除数据

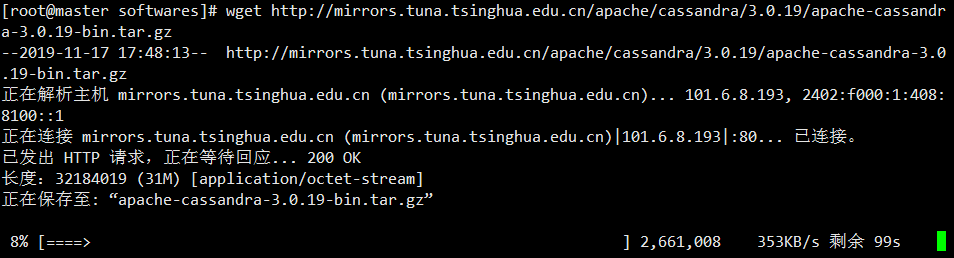
client.deleteAllRow('testtable',row='row-key1')

client.deleteAllRowTs('testtable','row-key1',1573912214382)

## 安装Cassandra

cd /export/softwares/

wget <http://mirrors.tuna.tsinghua.edu.cn/apache/cassandra/3.0.19/apache-cassandra-3.0.19-bin.tar.gz>



tar zxvf apache-cassandra-3.0.19-bin.tar.gz -C ../servers

vi /etc/profile

export CASSANDRA\_HOME=/export/servers/apache-cassandra-3.0.19

export PATH=:$CASSANDRA\_HOME/bin:$PATH

source /etc/profile

## 修改配置文件 cassandra.yaml

#进入$CASSANDRA\_HOME/conf配置文件所在的目录

cd $CASSANDRA\_HOME/conf

### a.修改cassandra集群的名字(默认是 Test Cluster)

# The name of the cluster. This is mainly used to prevent machines in

# one logical cluster from joining another.

cluster\_name: 'Test Cluster'

### b.设置集群种子节点IP，如果多个用逗号分隔

# seeds is actually a comma-delimited list of addresses.

# Ex: "<ip1>,<ip2>,<ip3>"

- seeds: "192.168.52.100,192.168.52.110,192.168.52.120"

### c.设置监听地址(本机的IP)，是为了其他节点能与节点进行通信(默认是 localhost)，每台机器填自己机器的IP

# Setting listen\_address to 0.0.0.0 is always wrong.

listen\_address: 192.168.52.100

### d.开启 thrift rpc 服务(默认是 false)

# Whether to start the thrift rpc server.

start\_rpc: true

### e.设置rpc的地址(默认是 localhost)

# For security reasons, you should not expose this port to the internet. Firewall it if needed.

rpc\_address: 192.168.52.100

### f.设置数据文件所在路径(默认是 $CASSANDRA\_HOME/data/data)

# If not set, the default directory is $CASSANDRA\_HOME/data/data.

data\_file\_directories:

- /data1/cassandradata/data

- /data2/cassandradata/data

- /data3/cassandradata/data

- /data4/cassandradata/data

- /data5/cassandradata/data

### g.设置commitlog文件所在路径(默认是 $CASSANDRA\_HOME/data/commitlog)

# If not set, the default directory is $CASSANDRA\_HOME/data/commitlog.

commitlog\_directory: /data6/cassandradata/commitlog

问：为什么要设置 data\_file\_directories 和 commitlog\_directory？

答：因为这两个文件很大，分散集群中磁盘I/O压力，前者是cassandra实际数据存放的目录，后者是数据写入commitlog的文件目录

## 分发安装包

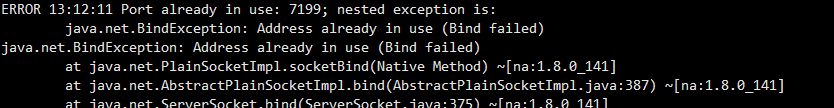
scp -r apache-cassandra-3.0.19/ node02:$PWD

scp -r apache-cassandra-3.0.19/ node03:$PWD

## 修改 $CASSANDRA\_HOME/conf/cassandra.yaml 中的 listen\_address 和 rpc\_address 将其设置成自己的IP

## 启动cassandra

cassandra -r



netstat -tunlp |grep 7199

cassandra -p cassandra.pid

pkill -F cassandra.pid

nodetool status



## CentOS 6.9下将python2.6.6升级为Python2.7.13

查看当前系统中的 Python 版本

python --version

返回 Python 2.6.6 为正常。

检查 CentOS 版本

cat /etc/redhat-release

返回 CentOS release 6.9 (Final) 为正常。

安装所有的开发工具包

yum groupinstall -y "Development tools"

安装其它的必需包

yum install -y zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel

下载、编译和安装 Python 2.7.13

wget https://www.python.org/ftp/python/2.7.13/Python-2.7.13.tgz

tar zxf Python-2.7.13.tgz

cd Python-2.7.13

./configure

make && make install

默认 Python 2.7.13 会安装在 /usr/local/bin 目录下。

ll -tr /usr/local/bin/python\*

/usr/local/bin/python2.7

/usr/local/bin/python2.7-config

/usr/local/bin/python -> python2

/usr/local/bin/python2 -> python2.7

/usr/local/bin/python2-config -> python2.7-config

/usr/local/bin/python-config -> python2-config

而系统自带的 Python 是在 /usr/bin 目录下。

ll -tr /usr/bin/python\*

/usr/bin/python2.6-config

/usr/bin/python2.6

/usr/bin/python

/usr/bin/python2 -> python

/usr/bin/python-config -> python2.6-config

更新系统默认 Python 版本

先把系统默认的旧版 Python 重命名。

mv /usr/bin/python /usr/bin/python.old

再删除系统默认的 python-config 软链接。

rm -f /usr/bin/python-config

最后创建新版本的 Python 软链接。

ln -s /usr/local/bin/python /usr/bin/python

ln -s /usr/local/bin/python-config /usr/bin/python-config

ln -s /usr/local/include/python2.7/ /usr/include/python2.7

以上步骤做完以后，目录 /usr/bin 下的 Python 应该是

ll -tr /usr/bin/python\*

/usr/bin/python2.6-config

/usr/bin/python2.6

/usr/bin/python.old

/usr/bin/python2 -> python

/usr/bin/python -> /usr/local/bin/python

/usr/bin/python-config -> /usr/local/bin/python-config

查看新的 Python 版本

python --version

返回 Python 2.7.13 为正常。

以下步骤还是有必要的

为新版 Python 安装 setuptools

wget https://bootstrap.pypa.io/ez\_setup.py -O - | python

setuptools 正确安装完成后，easy\_install 命令就会被安装在 /usr/local/bin 目录下了。

wget https://pypi.python.org/packages/source/d/distribute/distribute-0.6.10.tar.gz

tar xf distribute-0.6.10.tar.gz

cd distribute-0.6.10

python2.7 setup.py install

wget http://curl.haxx.se/ca/cacert.pem

mv cacert.pem ca-bundle.crt

cp ca-bundle.crt /etc/pki/tls/certs/

为新版 Python 安装 pip

wget https://bootstrap.pypa.io/get-pip.py

python get-pip.py

至此，新版 Python 即算安装完毕了。

注意：这可能会导致以前安装过的 Python 程序运行不了或者无法重启之类的（比如著名的 Shadowsocks Python 版）。原因是旧版的 pkg\_resources 位于 /usr/lib/python2.6/site-packages 下。而新版的则是在 /usr/local/lib/python2.7/site-packages 下。

所以，也许你需要重新安装一下程序。

再次注意：升级 Python 可能会导致 yum 命令不可用。解决方法如下：

编辑 /usr/bin/yum 文件，将开头第一行的

#!/usr/bin/python

改为

#!/usr/bin/python2.6

但是，这种改法，万一哪天你 yum update 了一下，yum 被升级了后，又变回老样子了。

记住旧版本 Python 2.6.6 的重要路径如下所示，在运行 yum 命令的时候，会提示你哪个 module 不存在，不存在的我们就去旧版本的路径下找，一定能找到的。找到后，复制到新版本 Python 的路径 /usr/local/lib/python2.7/site-packages/ 下即可。

/usr/lib/python2.6/site-packages/

/usr/lib64/python2.6/site-packages/

我的复制过程是这样的：

cp -r /usr/lib/python2.6/site-packages/yum /usr/local/lib/python2.7/site-packages/

cp -r /usr/lib/python2.6/site-packages/rpmUtils /usr/local/lib/python2.7/site-packages/

cp -r /usr/lib/python2.6/site-packages/iniparse /usr/local/lib/python2.7/site-packages/

cp -r /usr/lib/python2.6/site-packages/urlgrabber /usr/local/lib/python2.7/site-packages/

cp -r /usr/lib64/python2.6/site-packages/rpm /usr/local/lib/python2.7/site-packages/

cp -r /usr/lib64/python2.6/site-packages/curl /usr/local/lib/python2.7/site-packages/

cp -p /usr/lib64/python2.6/site-packages/pycurl.so /usr/local/lib/python2.7/site-packages/

cp -p /usr/lib64/python2.6/site-packages/\_sqlitecache.so /usr/local/lib/python2.7/site-packages/

cp -p /usr/lib64/python2.6/site-packages/sqlitecachec.py /usr/local/lib/python2.7/site-packages/

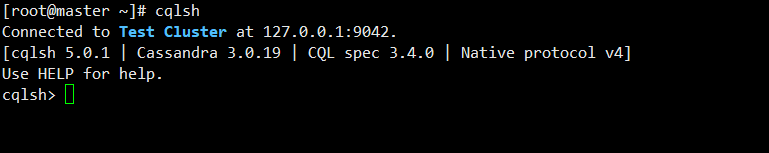
cp -p /usr/lib64/python2.6/site-packages/sqlitecachec.pyc /usr/local/lib/python2.7/site-packages/

cp -p /usr/lib64/python2.6/site-packages/sqlitecachec.pyo /usr/local/lib/python2.7/site-packages/

## cqlsh基本用法

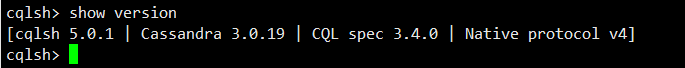
进入shell

cqlsh



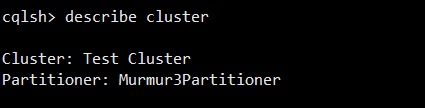
查看版本信息

show version



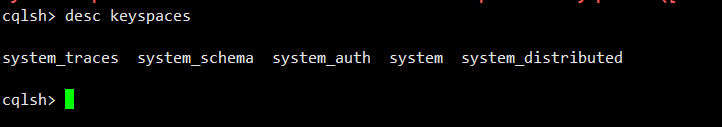
描述集群信息

describe cluster



查看空间列表

desc keyspaces



## 键空间管理

### 创建键空间

简单复制策略(SimpleStrategy)

create keyspace ks1 with replication={'class':'SimpleStrategy','replication\_factor':'1'};

网络拓扑复制策略(NetworkTopologyStrategy)

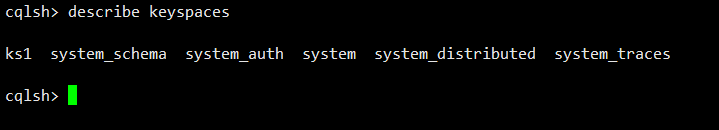
create keyspace ks1 with replication={'class':'NetworkTopologyStrategy','dc1':3,'dc2':2 } AND DURABLE\_WRITES=false;

### 删除键空间

drop keyspace ks1

### 查看键空间列表

describe keyspaces

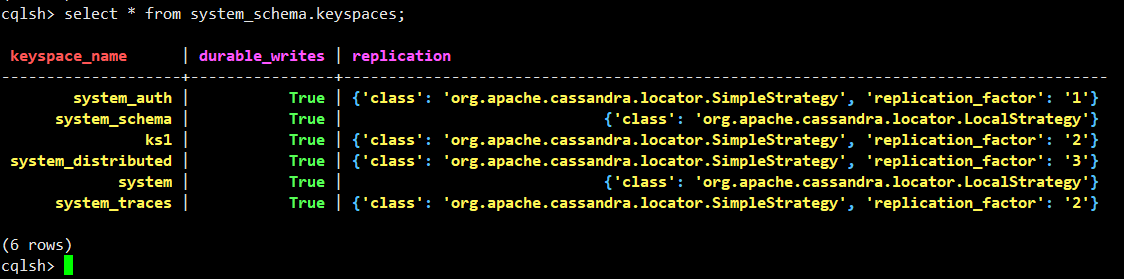


### 修改键空间属性

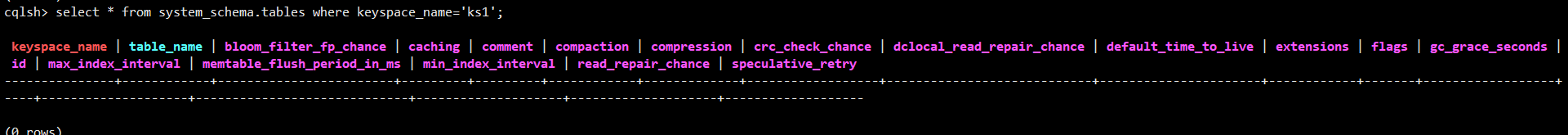
alter keyspace ks1 with replication ={'class':'SimpleStrategy','replication\_factor':'2'};

### 系统键空间

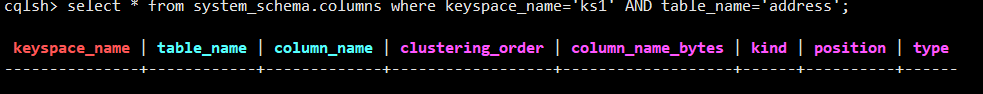
select \* from system\_schema.keyspaces;



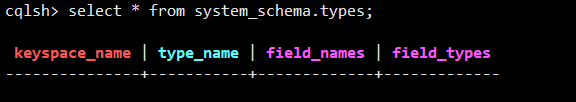
select \* from system\_schema.tables where keyspace\_name='ks1';



select \* from system\_schema.columns where keyspace\_name='ks1' AND table\_name='address';



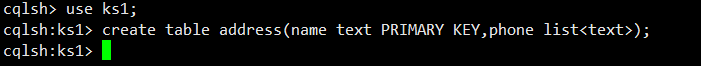
select \* from system\_schema.types;



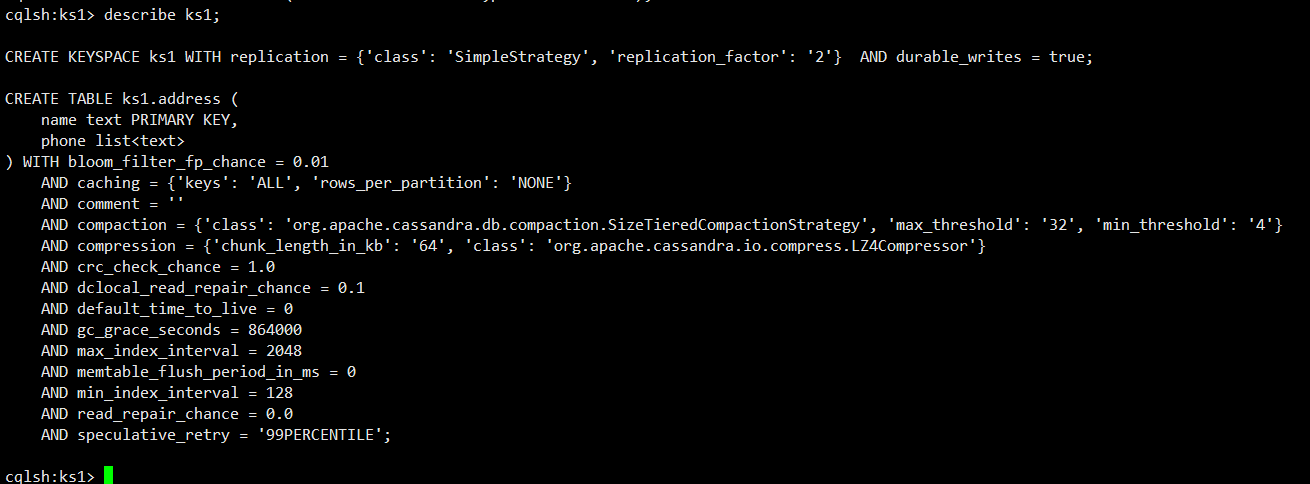
## 数据表管理

### 建立数据表

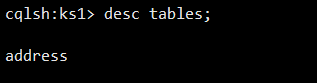
create table address(name text PRIMARY KEY,phone list<text>);



describe ks1;



desc tables;



### 设置复合型主键

create table address2(name text,phone list<text> , primary key(name));

### 修改表结构

alter table address add age int;

alter table address with bloom\_filter\_fp\_chance=0.01;

### 删除数据并重建表

truncate address;

## 用户自定义类型

create type scores(subject text,score int);

drop type scores;

## CQL数据查询

create table ks1.testtable1(

col1 text,

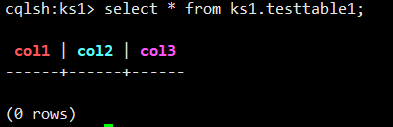
col2 int,

col3 tuple<text,text>,

PRIMARY KEY (col1,col2)

);

select \* from ks1.testtable1;



### 条件查询

create table test(

key int,

col1 int,

col2 int,

col3 int,

col4 int,

primary key((key),col1,col2,col3,col4)

);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,1,1,1);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,1,1,2);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,1,1,3);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,2,2,1);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,2,2,2);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,2,2,3);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,2,2,1);

insert into ks1.test(key,col1,col2,col3,col4) values(100,2,1,2,2);

insert into ks1.test(key,col1,col2,col3,col4) values(100,2,1,2,3);

insert into ks1.test(key,col1,col2,col3,col4) values(100,2,1,1,1);

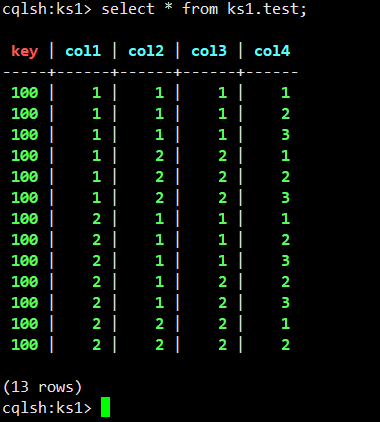
insert into ks1.test(key,col1,col2,col3,col4) values(100,2,1,1,2);

insert into ks1.test(key,col1,col2,col3,col4) values(100,2,1,1,3);

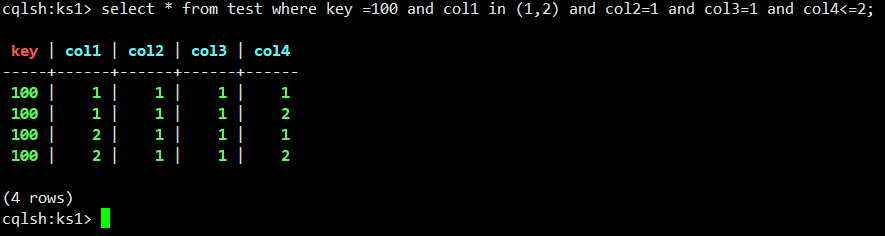
insert into ks1.test(key,col1,col2,col3,col4) values(100,2,2,2,1);

insert into ks1.test(key,col1,col2,col3,col4) values(100,2,2,2,2);

insert into ks1.test(key,col1,col2,col3,col4) values(100,1,2,2,3);

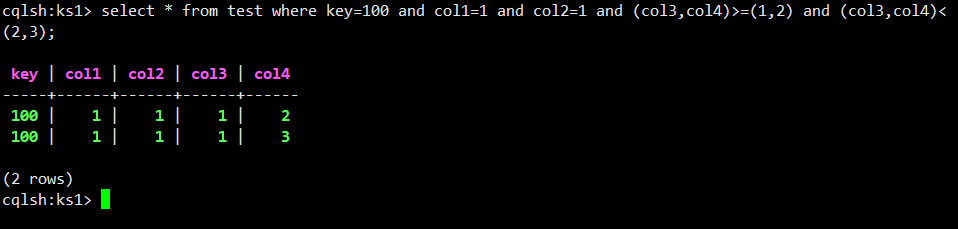


select \* from test where key =100 and col1 in (1,2) and col2=1 and col3=1 and col4<=2;



### 切片查询

select \* from test where key=100 and col1=1 and col2=1 and (col3,col4)>=(1,2) and (col3,col4)<(2,3);



## 索引机制

建立索引

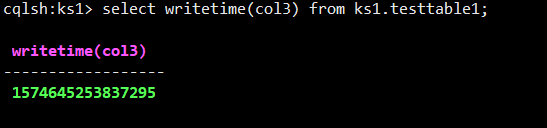
create index indexofaddress on address(age) ;

删除索引

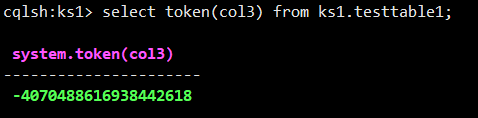
drop index indexofaddress;

## 使用标量函数

select writetime(col3) from ks1.testtable1;



select token(col3) from ks1.testtable1;



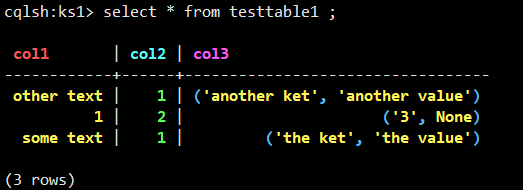
# 数据更新

## 插入更新删除

### 数据插入

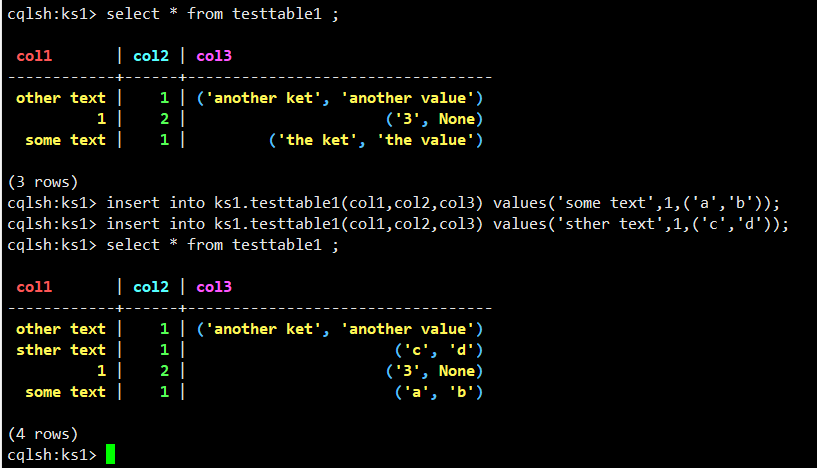
insert into ks1.testtable1(col1,col2,col3) values('some text',1,('the ket','the value'));

insert into ks1.testtable1(col1,col2,col3) values('other text',1,('another ket','another value'));



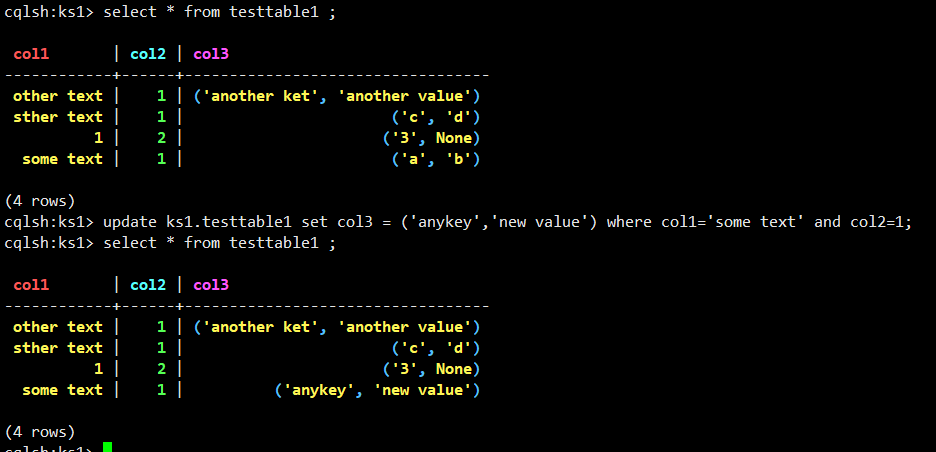
insert into ks1.testtable1(col1,col2,col3) values('some text',1,('a','b'));

insert into ks1.testtable1(col1,col2,col3) values('sther text',1,('c','d'));



### 数据更新

update ks1.testtable1 set col3 = ('anykey','new value') where col1='some text' and col2=1;



### 数据删除

DELETE col3 FROM ks1.testtable1 WHERE col1='some test'and col2=1;

DELETE from ks1.testtable1 where col1='other text' and col2=10;

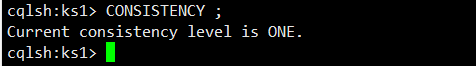
### json格式插入数据

insert into ks1.testtable1 JSON '{"col1":"json test","col2":1000}';

## 读写一致性

### 查看一致性设置

CONSISTENCY ;



### if 轻量级事物

insert into ks1.testtable1(col1,col2,col3) values('some test',1,('another key','another value')) if not exists;

## 集合列操作

create table testtable2(

col1 int PRIMARY KEY,

col2 list<text>,

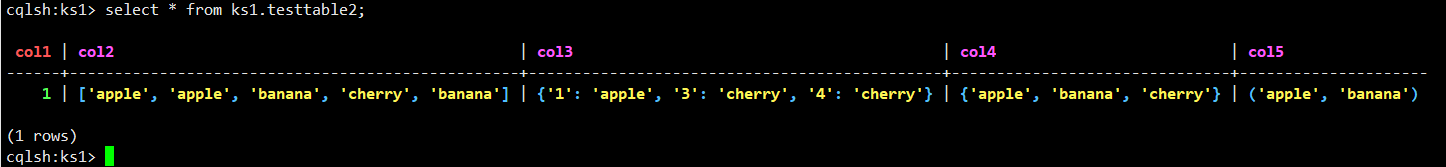
col3 map<text,text>,

col4 set<text>,

col5 frozen<tuple<text,text>>

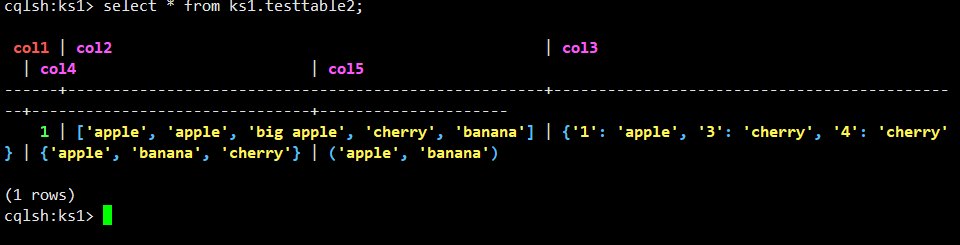
);

INSERT INTO ks1.testtable2(col1,col2,col3,col4,col5) VALUES (1,['apple','apple','banana','cherry','banana'],{'1':'apple','1':'banana','3':'cherry','4':'cherry'},{'apple','banana','cherry','apple'}, ('apple','banana'));



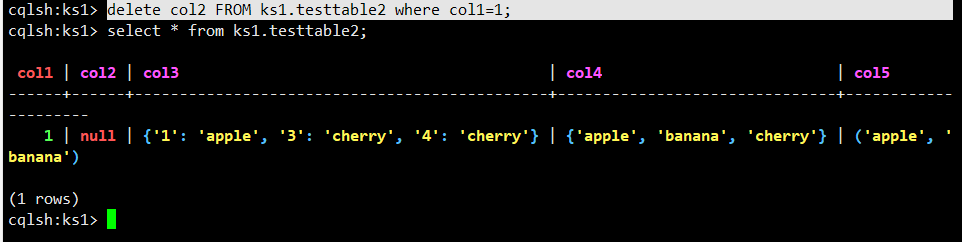
### list类型更新删除

update ks1.testtable2 set col2[2] = 'big apple' where col1 = 1 ;



delete col2[2] from ks1.testtable2 WHERE col1 =1;

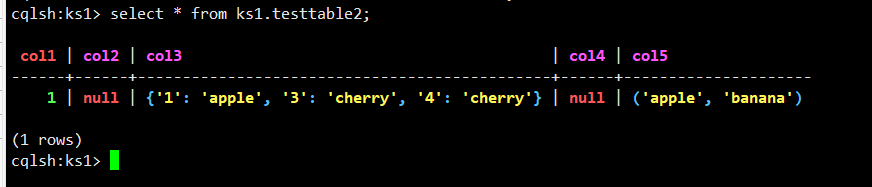
delete col2 FROM ks1.testtable2 where col1=1;



### set类型更新和删除

UPDATE ks1.testtable2 set col4=col4+{'big apple','small apple'} where col1=1;

DELETE col4 FROM ks1.testtable2 WHERE col1=1;



## nodetool工具

## 查看集群状态

nodetool version



nodetool ring



### 查看compation信息

nodetool compactionstats

# JAVA访问Cassandra

## 修改pom.xml

<!-- https://mvnrepository.com/artifact/com.datastax.cassandra/cassandra-driver-core -->

<dependency>

<groupId>com.datastax.cassandra</groupId>

<artifactId>cassandra-driver-core</artifactId>

<version>3.7.2</version>

</dependency>

package cassandra;

import com.datastax.driver.core.Cluster;

import com.datastax.driver.core.ColumnDefinitions.Definition;

import com.datastax.driver.core.ResultSet;

import com.datastax.driver.core.Row;

import com.datastax.driver.core.Session;

import org.testng.annotations.Test;

public class Cassandra {

public Cluster cluster;

public Session session;

public void connect(){

Cluster culster=Cluster.builder().withClusterName("Test Cluster").addContactPoint("192.168.52.129").build();

session=culster.connect();

}

/\*\*

\* 创建键空间

\*/

public void createKeyspace()

{

/\*\*单数据中心 复制策略 ：1\*\*/

String cql = "CREATE KEYSPACE if not exists mydb WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': '1'}";

session.execute(cql);

}

/\*\*

\* 创建表

\*/

public void createTable()

{

/\*\* a,b为复合主键 a：分区键，b：集群键\*\*/

String cql = "CREATE TABLE if not exists mydb.test (a text,b int,c text,d int,PRIMARY KEY (a, b))";

session.execute(cql);

}

/\*\*

\* 插入

\*/

public void insert()

{

String cql = "INSERT INTO mydb.test (a , b , c , d ) VALUES ( 'a2',4,'c2',6);";

session.execute(cql);

}

/\*\*

\* 修改

\*/

public void update()

{

// a,b是复合主键 所以条件都要带上，少一个都会报错，而且update不能修改主键的值，这应该和cassandra的存储方式有关

String cql = "UPDATE mydb.test SET d = 1234 WHERE a='aa' and b=2;";

// 也可以这样 cassandra插入的数据如果主键已经存在，其实就是更新操作

String cql2 = "INSERT INTO mydb.test (a,b,d) VALUES ( 'aa',2,1234);";

// cql 和 cql2 的执行效果其实是一样的

session.execute(cql);

}

/\*\*

\* 删除

\*/

public void delete()

{

// 删除一条记录里的单个字段 只能删除非主键，且要带上主键条件

String cql = "DELETE d FROM mydb.test WHERE a='aa' AND b=2;";

// 删除一张表里的一条或多条记录 条件里必须带上分区键

String cql2 = "DELETE FROM mydb.test WHERE a='aa';";

session.execute(cql);

session.execute(cql2);

}

/\*\*

\* 查询

\*/

public void query()

{

String cql = "SELECT \* FROM mydb.test;";

String cql2 = "SELECT a,b,c,d FROM mydb.test;";

ResultSet resultSet = session.execute(cql);

System.out.print("这里是字段名：");

for (Definition definition : resultSet.getColumnDefinitions())

{

System.out.print(definition.getName() + " ");

}

System.out.println();

System.out.println(String.format("%s\t%s\t%s\t%s\t\n%s", "a", "b", "c", "d",

"--------------------------------------------------------------------------"));

for (Row row : resultSet)

{

System.out.println(String.format("%s\t%d\t%s\t%d\t", row.getString("a"), row.getInt("b"),

row.getString("c"), row.getInt("d")));

}

}

@Test

public void Test(){

connect();

createKeyspace();

createTable();

insert();

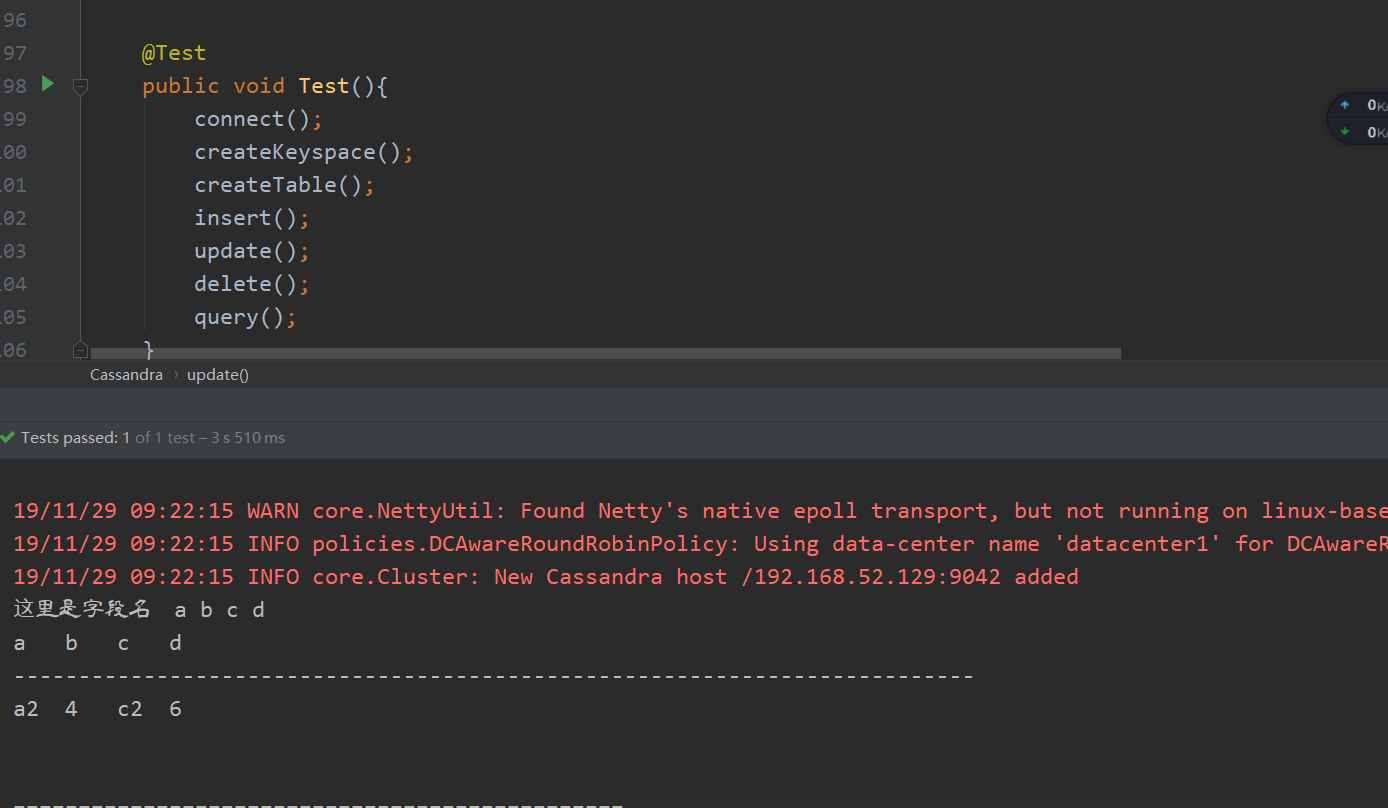
update();

delete();

query();

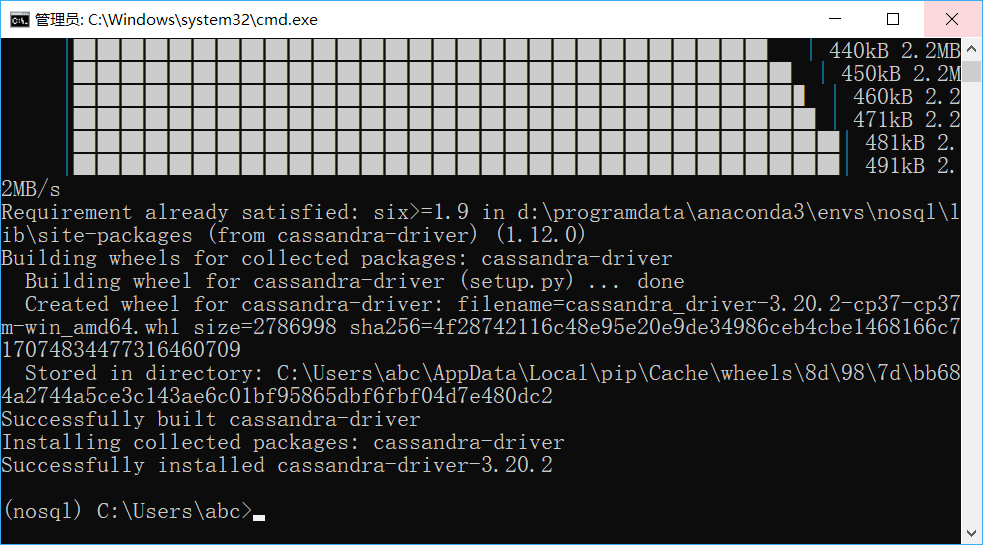
}

}



## Python访问Cassandra

pip install cassandra-driver



# encoding=UTF-8

from cassandra.cluster import Cluster

cluster = Cluster(['192.168.52.129'])

cluster.port=9042

session = cluster.connect()#创建连接

session\_keyspace = cluster.connect("ks1") #直接连接keyspace

'''

对Cassandra进行操作

'''

#1. 创建键空间

#1.1 使用SimpleStrategy策略创建键空间

'''

简单复制策略是指在一个数据中心的情况下使用简单的策略.该策略中,第一个副本被放置在所选择的节点上,剩下的节点采用Dynamo论文中的副本策略,不考虑机架位置

'''

#'replication':n用于指示副本数量

#session.execute("create keyspace test with replication={'class':'SimpleStrategy','replication\_factor':'1'};")

#1.2网络拓补复制策略,在该策略下,数据副本采用二级机架感知策略

#Durable\_writes默认为true,表示数据再写入时,先持久化在预写日志中,便于故障恢复.再网络拓补复制策略下,该选项可设置为false,但有数据丢失的风险

#session.execute("create keyspace ks3 with replication={'class':'NetworkTopologyStrategy','dc1':3,'dc2':2} and durable\_writes=false;")

#1.4 查看键空间列表

#在cqlsh里可使用 describe spaces;查询键空间列表,但execute不支持describe命令

print(cluster.metadata.keyspaces)

print("\n")

# 因为cassandra为键值对的存储方式,所以,可使用类似显示字典内容的方式输出键空间

for k,v in cluster.metadata.keyspaces.items():

print(k,v)

#1.5 修改键空间属性

session.execute("alter keyspace ks1 with replication={'class':'SimpleStrategy','replication\_factor':2}")

#1.7 系统键空间

'''

system\_schema表中存储当前所有的键空间和数据表的局部信息(schema信息)

'''

result\_keySpace=session.execute("select \* from system\_schema.keyspaces;")

for i in result\_keySpace:

print(i)

for j in i:

print(j)

'''

所有数据表的信息存储于系统表system\_shema.tables中

'''

result\_ks1\_tables=session.execute("select \* from system\_schema.tables where keyspace\_name = 'ks1';")

for i in result\_ks1\_tables:

print(i)

'''

所有数据表的列信息存储于system\_schema.columns中

'''

result\_ks1\_columns=session.execute("select \* from system\_schema.columns where keyspace\_name = 'ks1';")

for i in result\_ks1\_columns:

print(i)

'''

所有用户自定义数据类型都存储在system\_schema.types中

'''

result\_ks1\_udf = session.execute("select \* from system\_schema.types")

for i in result\_ks1\_udf:

print(i)

'''

创建表,在此之前需要保证是先使用目标键空间

'''

session.execute("use ks1;")

#2.1建表操作

# session.execute("create table py(name text primary key,phone list<text>);")

print([i for i in session.execute("select table\_name from system\_schema.tables where keyspace\_name='ks1';")])

# 几种特殊的数据类型：Map（键值对）、Set、list、Frozen（非具体类型，强调对被frozen限制的整体的操作，例如:frozen<tuple<text,text>>）

#2.2 删除表

session.execute("drop table address;")

#2.3 设置复合型主键

'''

可直接在单一主键后加primary key，也可单独设置。

默认情况下，复合型主键的第一个主键为分区键(列)，其他为分簇键(列)

分簇列可单独设定升序(ASC)与降序(DESC)

'''

session.execute("create table address(firstname text,lastname text,No int,phone list<text>,primary key((firstname,lastname),No)) with clustering order by(No DESC);")

#2.4 修改表结构

'''

alter 仅支持 add drop rename 操作

'''

session.execute("alter table address add age int;")

print([i for i in session.execute("select column\_name,type from system\_schema.columns where keyspace\_name='ks1' and table\_name='address';")])

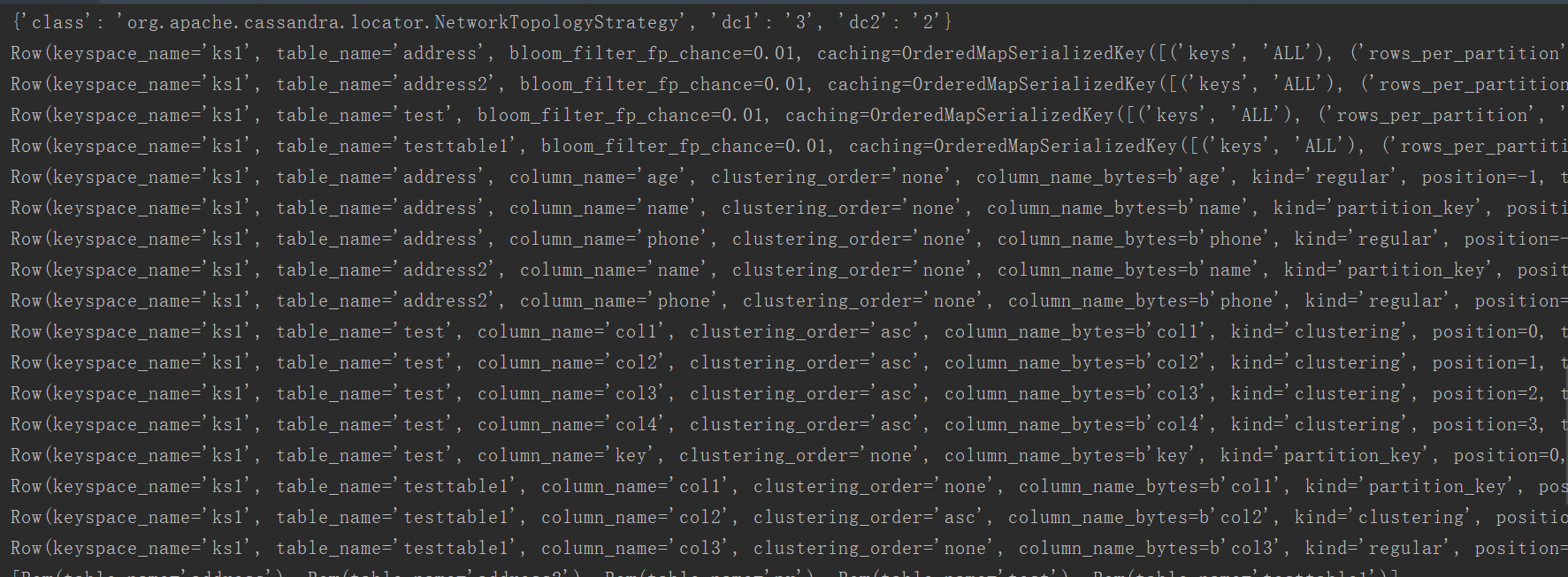
session.execute("alter table address drop age;")

print([i for i in session.execute("select column\_name,type from system\_schema.columns where keyspace\_name='ks1' and table\_name='address';")])

#间接查询，注意：若未设置索引，查询需添加ALLOW FILTER子句作为条件(性能无法预测)

print([i for i in session.execute("select \* from system\_schema.tables where table\_name='stu' allow filtering;")])

cluster.shutdown()#关闭连接



# MongoDB安装

## yum安装

创建yum源文件：

cd /etc/yum.repos.d

vim mongodb-org-4.0.repo

添加以下内容：

[mngodb-org]

name=MongoDB Repository

baseurl=http://mirrors.aliyun.com/mongodb/yum/redhat/7Server/mongodb-org/4.0/x86\_64/

gpgcheck=0

enabled=1

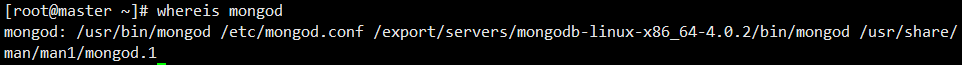
安装MongoDB

安装命令：

yum -y install mongodb-org

安装完成后,查看mongo安装位置

whereis mongod



systemctl start mongod.service

## 解压安装

MongoDB官网 <https://www.mongodb.com>

cd /export/softwares/

wget https://fastdl.mongodb.org/linux/mongodb-linux-x86\_64-4.0.2.tgz

tar zxvf mongodb-linux-x86\_64-4.0.2.tgz -C ../servers/

vi /etc/profile

export MONGODB\_HOME=/export/servers/mongodb-linux-x86\_64-4.0.2

export PATH=:$MONGODB\_HOME/bin:$PATH

cd /export/servers/mongodb-linux-x86\_64-4.0.2

mkdir -p /export/servers/mongodb-linux-x86\_64-4.0.2/db

开启mongodb

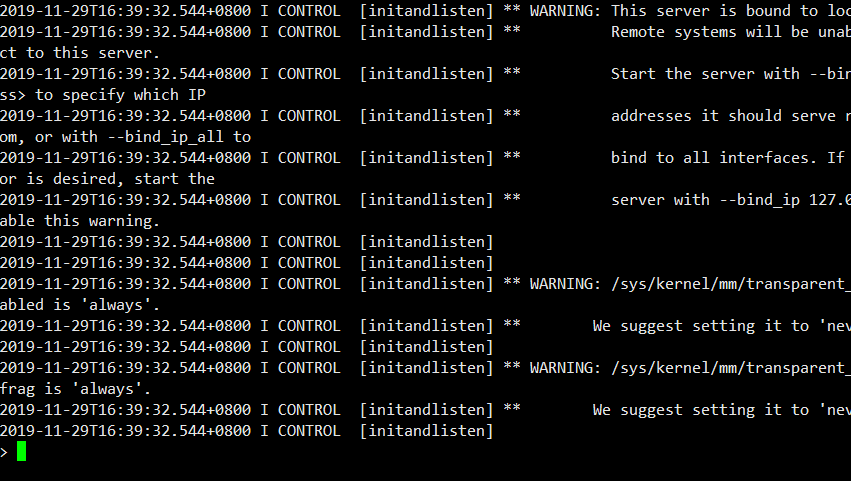
mongod --dbpath /export/servers/mongodb-linux-x86\_64-4.0.2/db

关闭mongodb

mongod -shutdown

进入shell

mongo --host 127.0.0.1:27017



## 配置文件

vi /etc/mongod.conf

bindIp: 0.0.0.0

# shell

mongo

# 数据库和集合操作

## 数据库操作

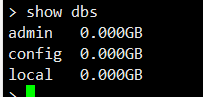
查看当前连接服务器

db.getMongo()



查看数据库列表

show dbs



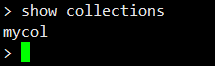
切换数据库

use test



查看数据库中所有集合

show collections



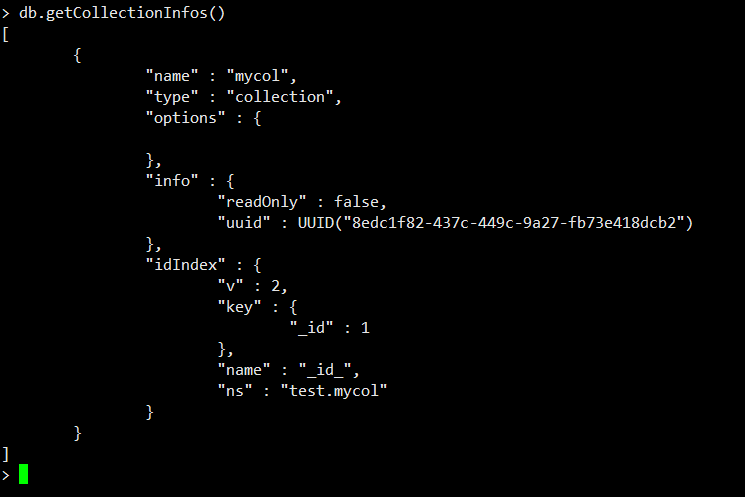
json显示集合名称

db.getCollectionNames()



集合详细信息

db.getCollectionInfos()



显示当前数据库名

db



删除数据库

db.dropDatabase()

## 集合操作

新建集合

db.createCollection("mycol")



删除集合

db.myCol.drop()

## 基本增删改查操作

文档插入

db.mycol.insert({

item1:'111111',

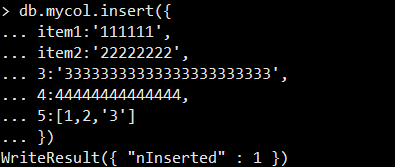
item2:'22222222',

3:'33333333333333333333333',

4:44444444444444,

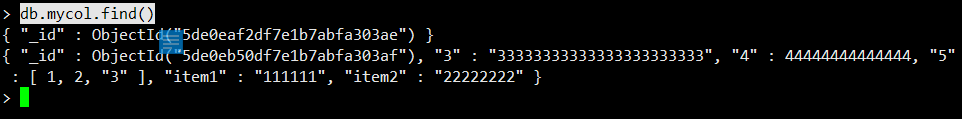
5:[1,2,'3']

})

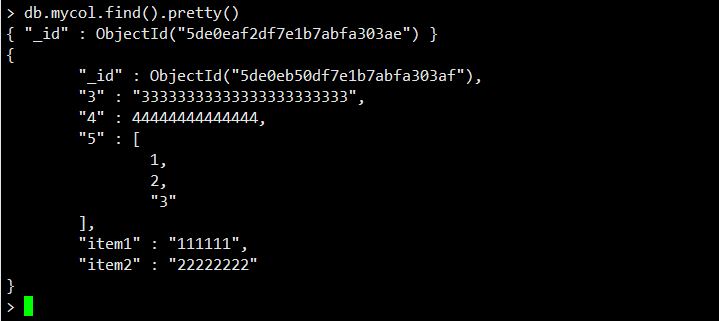


查看文档

db.mycol.find()

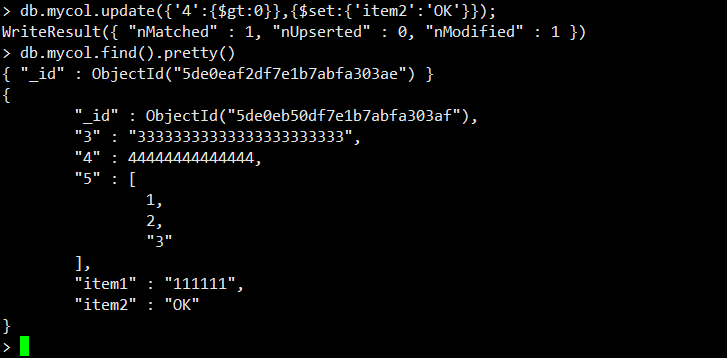


db.mycol.find().pretty()



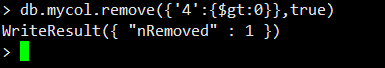
## 文档更新

db.mycol.update({'4':{$gt:0}},{$set:{'item2':'OK'}});



文档删除

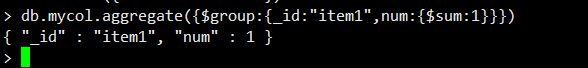
db.mycol.remove({'4':{$gt:0}},true)



db.mycol.remove({})

## 聚合和管道

db.mycol.aggregate({$group:{\_id:"item1",num:{$sum:1}}})

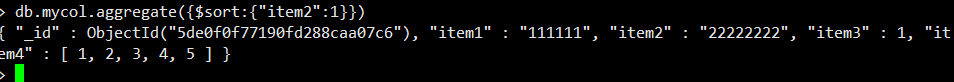


db.mycol.aggregate({$project:{item1:1, item2:1, item3:1}})

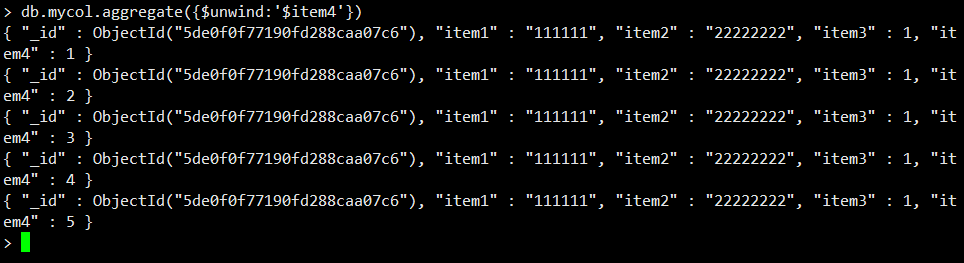


db.mycol.aggregate({$match:{"item3":{$gt:1,$gt:15}}})

db.mycol.aggregate({$sort:{"item2":1}})

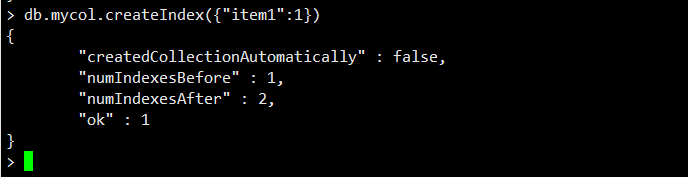


db.mycol.aggregate({$unwind:'$item4'})



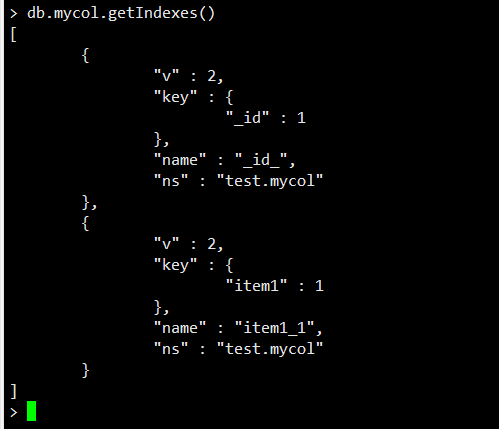
## 索引操作

db.mycol.createIndex({"item1":1})



查看索引

db.mycol.getIndexes()



删除索引

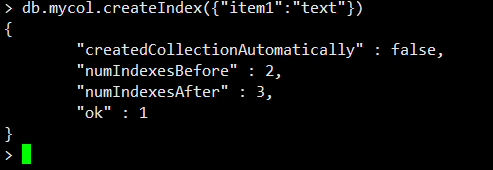
db.mycol.dropIndex("myindex")



db.mycol.dropIndexes()

全文索引

db.mycol.createIndex({"item1":"text"})



# python访问MongoDB

## 安装

pip install pymongo

## 导包

from pymongo import MongoClient

## 建立连接

client=MongoClient("192.168.52.129:27017")

## 切换数据库

db=client.get\_database("testdb")

db=client.testdb

## 切换集合

col=db.testcol

## 定义JSON文档

item={

"name":"fruits",

"count\_of":3,

"varieties":['banana',"cherry","orange"]

}

item1={

"name":"fruits",

"count\_of":4,

"varieties":['banana',"cherry","orange"]

}

item2={

"name":"fruits",

"count\_of":5,

"varieties":['banana',"cherry","orange"]

}

## 插入记录

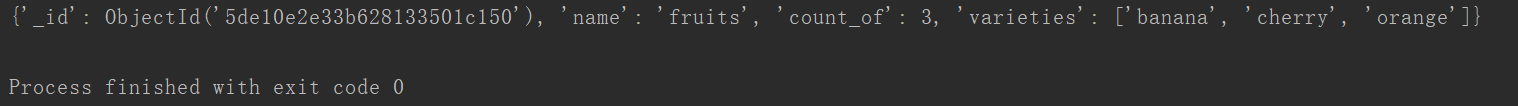
col.insert\_one(item)

col.insert\_one(item1)

col.insert\_one(item2)

## 查看数据

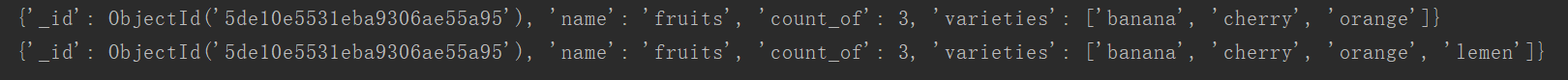
print(col.find\_one())



## 更新数据

col.update\_many({"name":"fruits"},{"$push":{"varieties":"lemen"}})

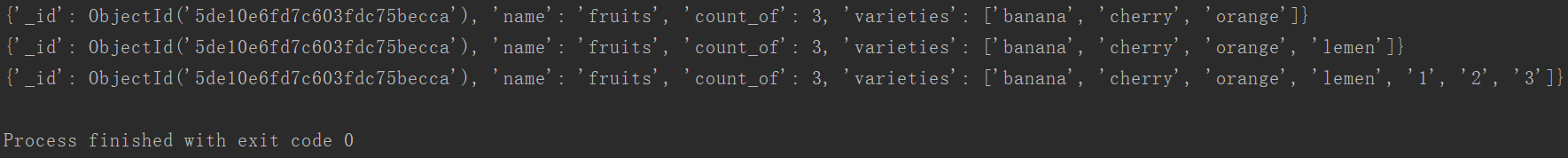
print(col.find\_one())



## $push $each插入多个元素

col.update\_many({"name":"fruits"},{"$push":{"varieties":{"$each":["1",'2','3']}}})

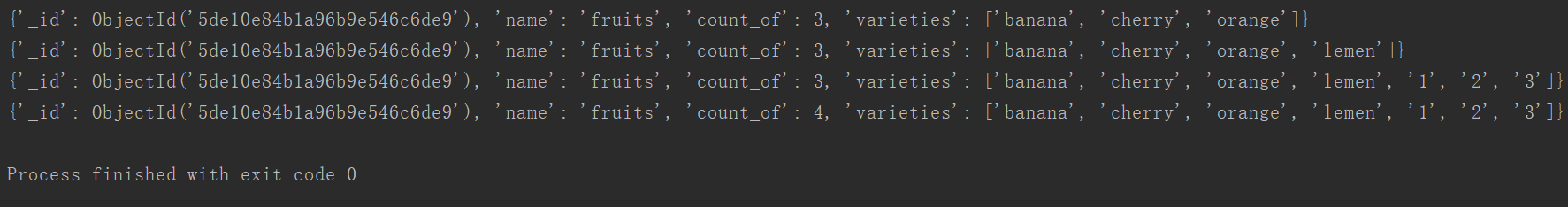
print(col.find\_one())



## 累加方式更新数字类型元素

col.update\_many({"name":"fruits"},{"$inc":{"count\_of":1}})

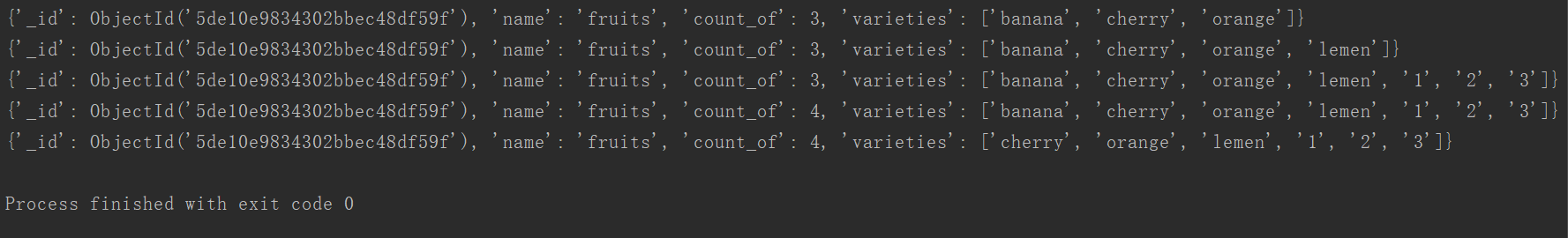
print(col.find\_one())



## $pop删除数据

col.update\_many({"name":"fruits"},{"$pop":{"varieties":-1}})

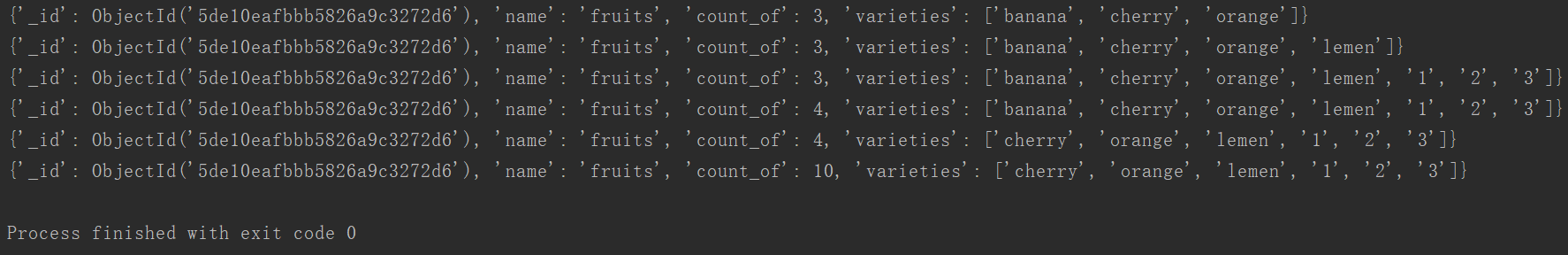
print(col.find\_one())



## 更新数据

col.update\_many({"name":"fruits"},{"$set":{"count\_of":10}})

print(col.find\_one())



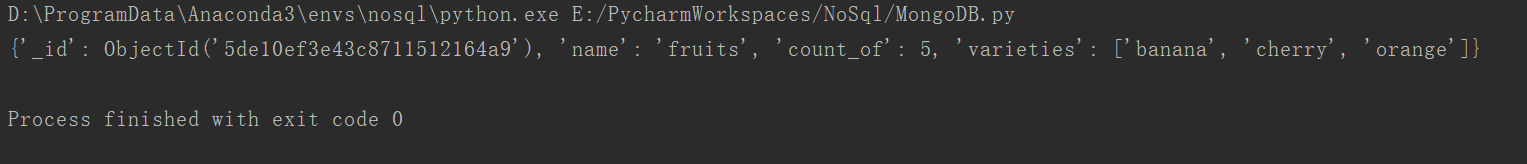
## 查看数据

print(col.find\_one({"name":"fruits"}))

## 排序 限制 跳过

for r in col.find({"name":"fruits"}).sort("count\_of").limit(3).skip(2):

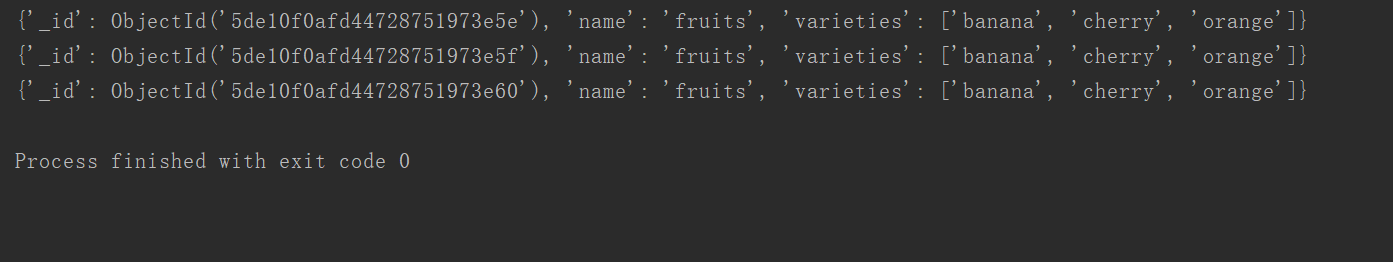
print(r)



## 控制显示的列

for r in col.find({"name":"fruits"},projection={'count\_of':False}):

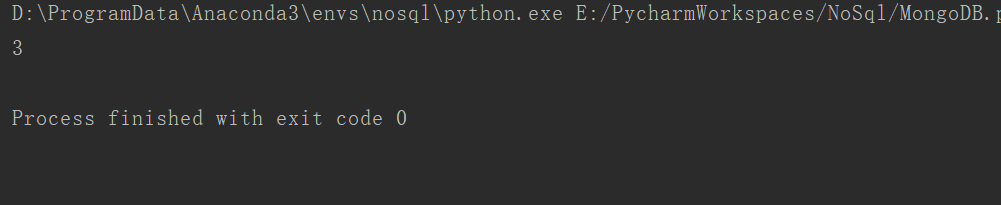
print(r)



## 聚合查询

print(col.find({"name":"fruits"}).count())

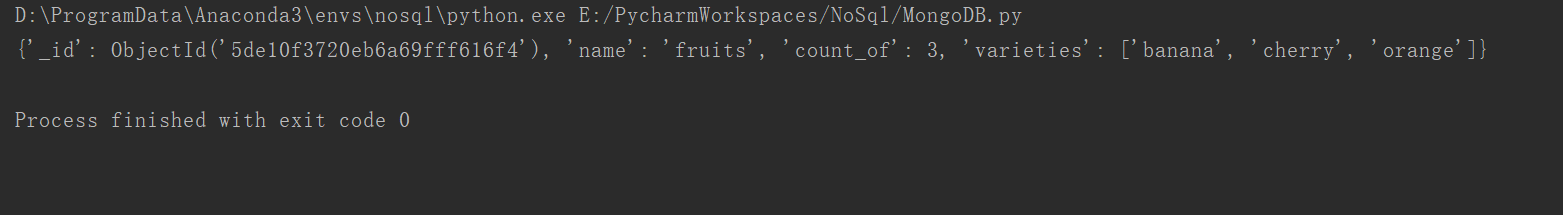
print(col.count\_documents({"name":"fruits"}))



## 比较运算符

for r in col.find({"count\_of":{"$lt":4}}):

print(r)



## 地理索引查询

from pymongo import GEO2D

db.places.create\_index([("loc",GEO2D)])

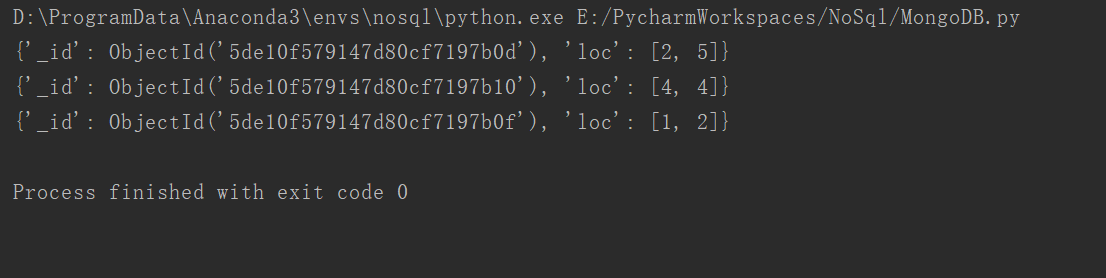
#插入经纬度信息

result = db.places.insert\_many([{"loc": [2, 5]},{"loc":[30, 5]},{"loc": [1, 2]},{"loc": [4, 4]}])

# $near 查询

for doc in db.places.find({"loc": {"$near": [3, 6]}}).limit(3):

print(doc)



## Gridfs操作

import gridfs

fs=gridfs.GridFS(db)

fileid=fs.put(b"hello word",filename="testfile")

print(fs.exists({'filename':'testfile'}))

print(fs.list())

for doc in fs.find({"filename":"testfile"}):

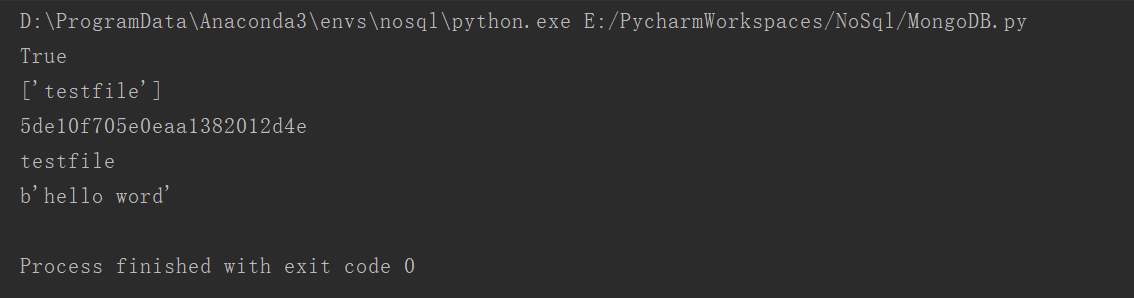
print(doc.\_id)

print(doc.filename)

print(doc.read())

#删除

fs.delete(doc.\_id)



## 删除数据

col.delete\_one({"name":"fruits"})

print(col.find\_one())

## 删除集合

db.places.delete\_many({})

db.drop\_collection("testcol")

# Neo4j安装

## 下载Neo4j

1. 访问neo4j官网下载，官网地址：[https://neo4j.com/](https://links.jianshu.com/go?to=https%3A%2F%2Fneo4j.com%2F)



1. 访问微云数据主页：[http://we-yun.com/index.php/blog/releases-56.html](https://links.jianshu.com/go?to=http%3A%2F%2Fwe-yun.com%2Findex.php%2Fblog%2Freleases-56.html)，提供国内ftp下载地址：[ftp://neo4j.55555.io/neo4j/3.5.6/](https://links.jianshu.com/go?to=ftp%3A%2F%2Fneo4j.55555.io%2Fneo4j%2F3.5.6%2F)

## 安装

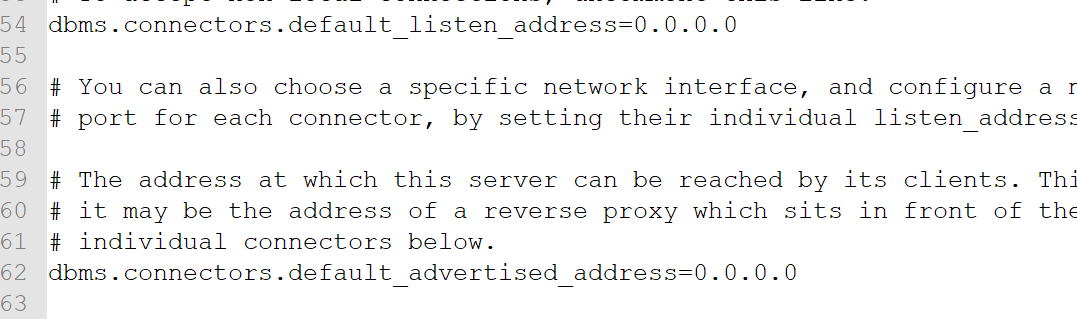
tar zxvf neo4j-community-3.5.6-unix.tar.gz -C ../servers/

cd /export/servers/neo4j-community-3.5.6/

vim conf/neo4j.conf

dbms.connectors.default\_listen\_address=0.0.0.0

dbms.connectors.default\_advertised\_address=0.0.0.0



vi /etc/profile

export NEO4J\_HOME=/export/servers/neo4j-community-3.5.6

export PATH=:$NEO4J\_HOME/bin:$PATH

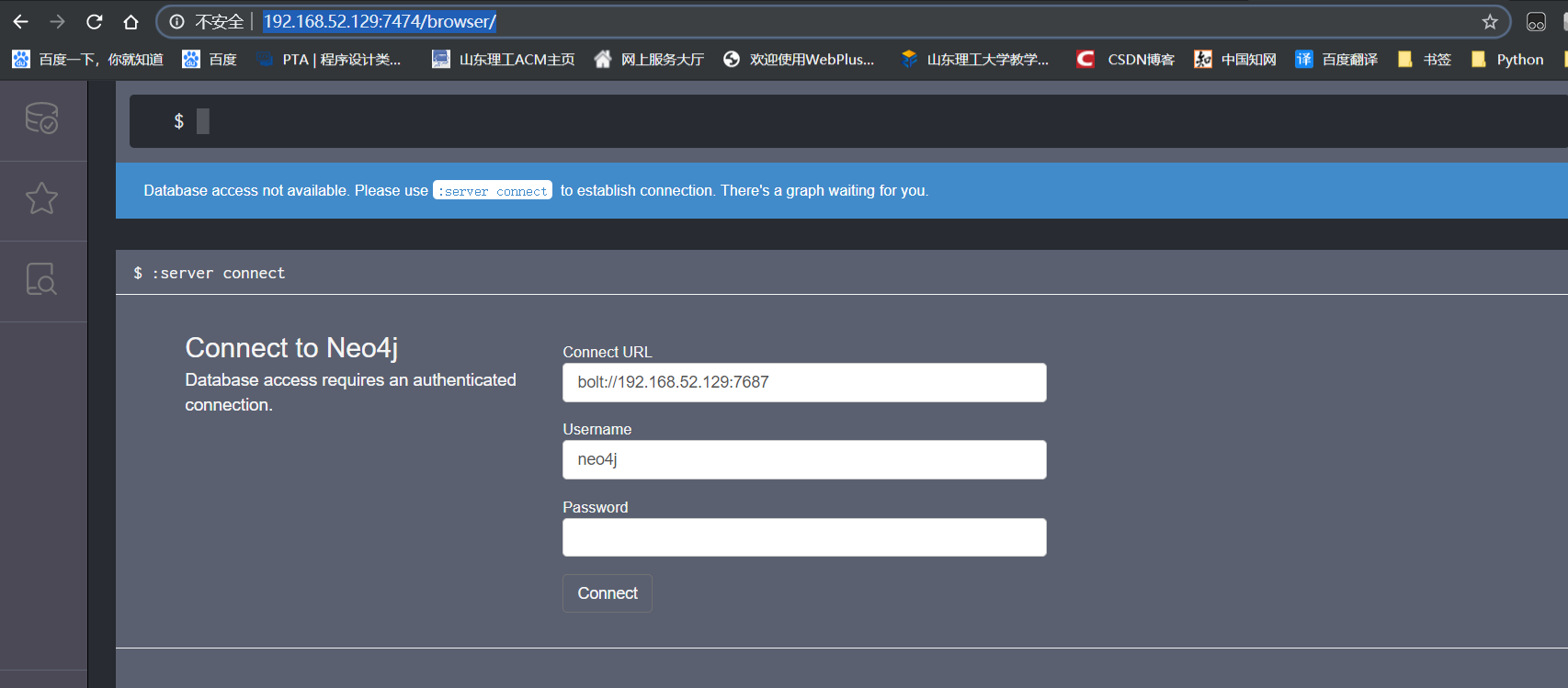
source /etc/profile

## 启动neo4j服务

neo4j start

## 访问web

<http://192.168.52.129:7474/browser/>



登录界面默认填充了用户名neo4j。

输入默认密码neo4j后，修改密码123456，进入neo4j

# Python访问Neo4j

安装驱动

pip install neo4j-driver

导包

from neo4j import GraphDatabase

建立连接

driver=GraphDatabase.driver("bolt://192.168.52.129",auth=("neo4j", "123456"))

session = driver.session()

传递Cypher语句

session.run("CREATE (a:Persion {name:'alice'})")  
result=session.run("MATCH (a:Persion) RETURN a.name")  
for i in result:  
 print(i['a.name'])  
  
result = session.run("MATCH (a:Persion) RETURN a")  
for i in result:  
 print(i['a'])

关闭连接

session.close()

