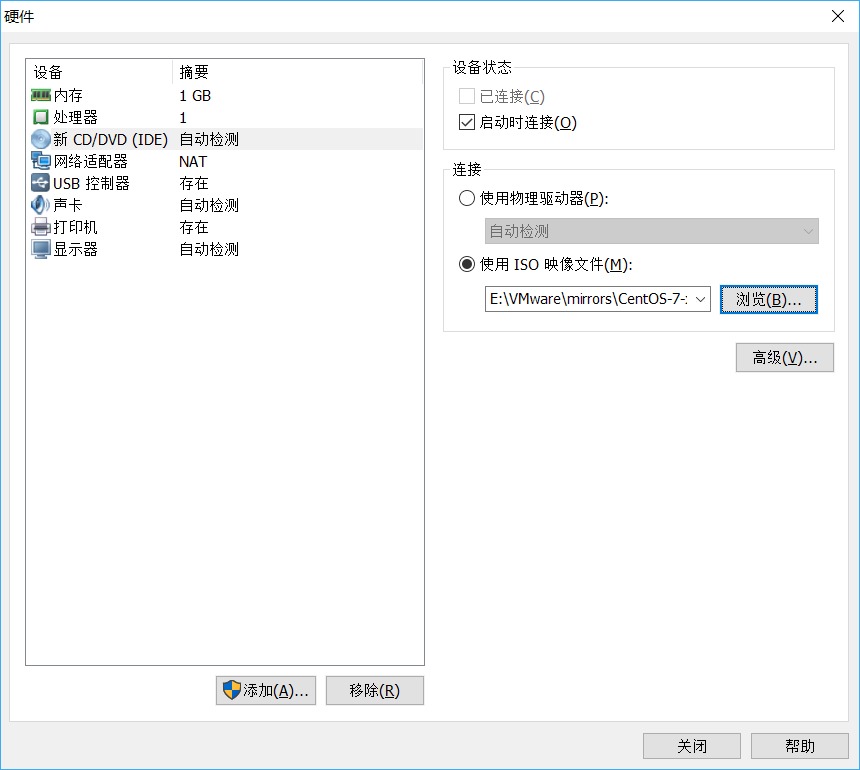
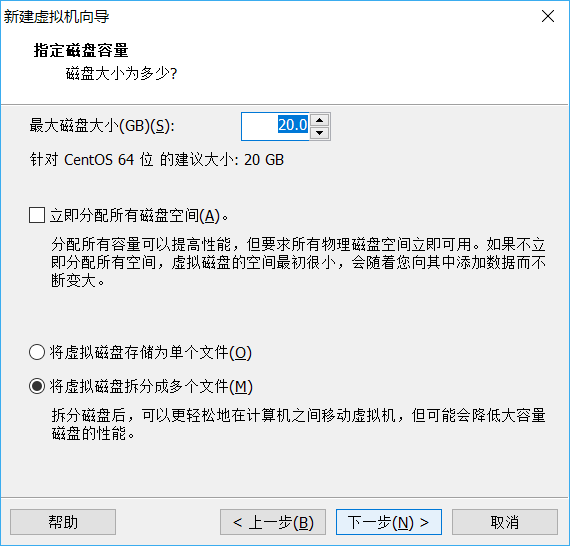
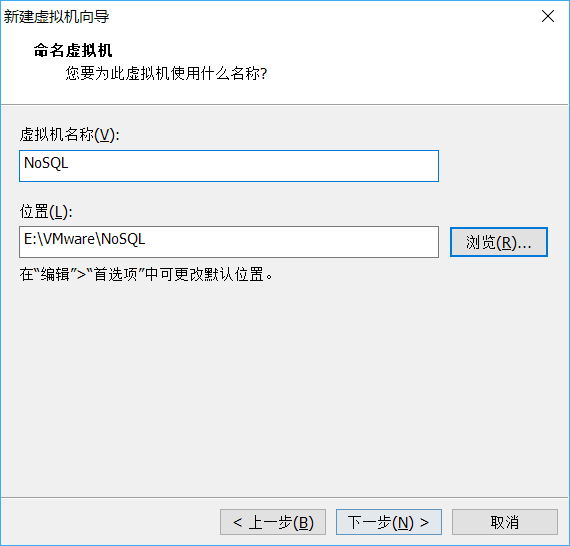
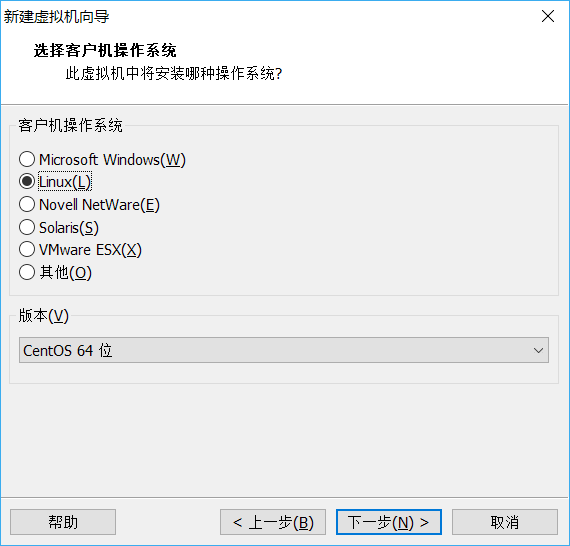
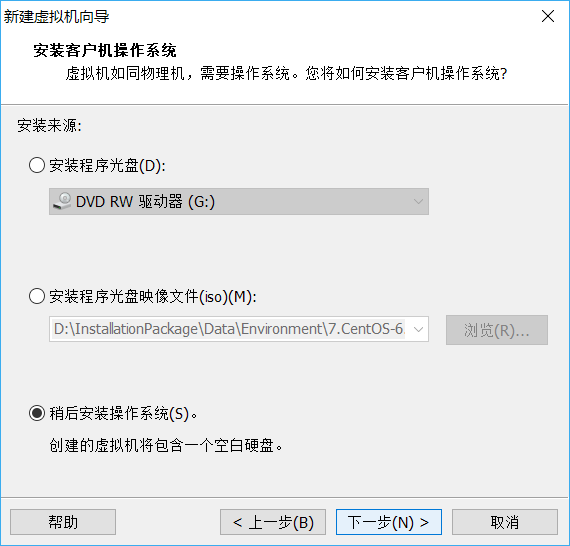
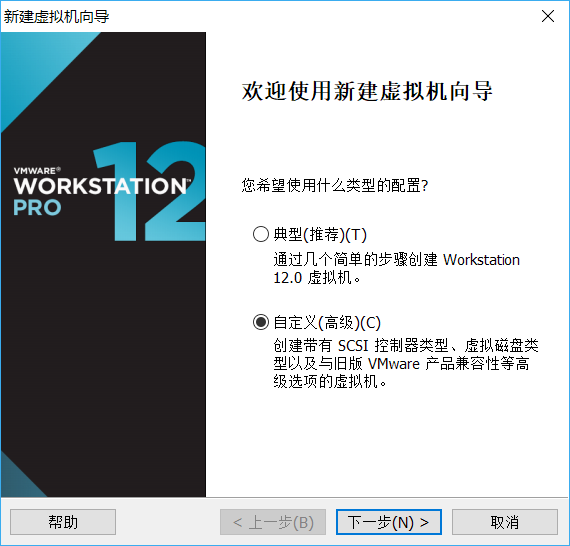
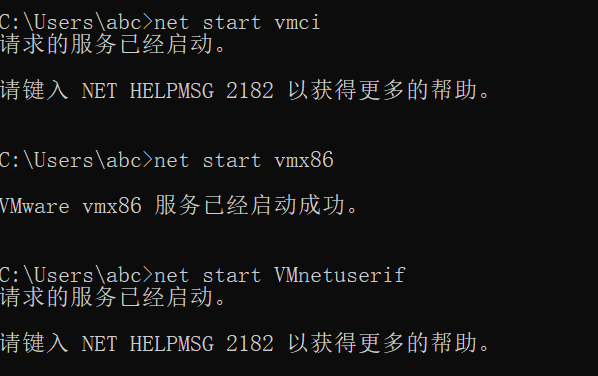
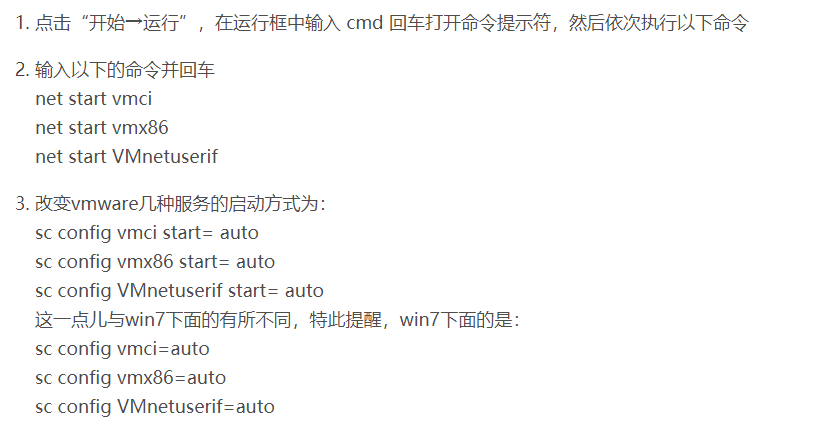
# HBase

## 虚拟机安装CentOS7

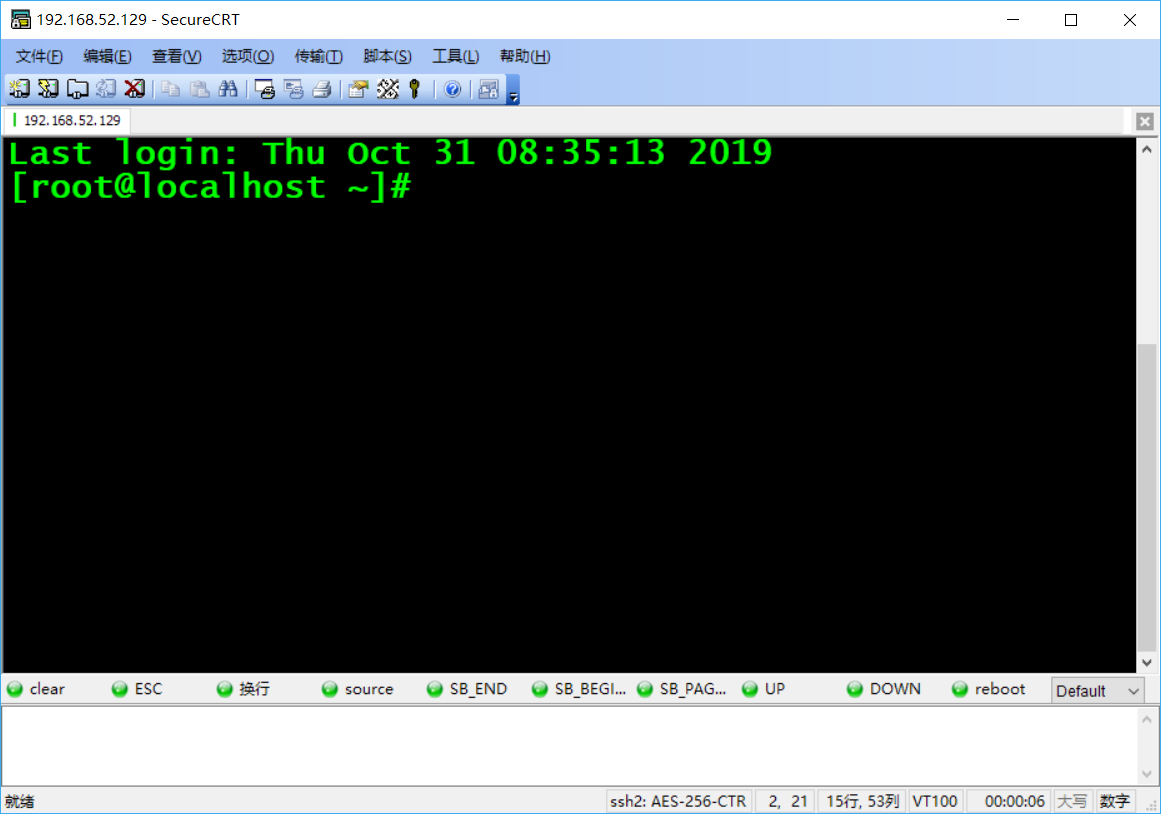
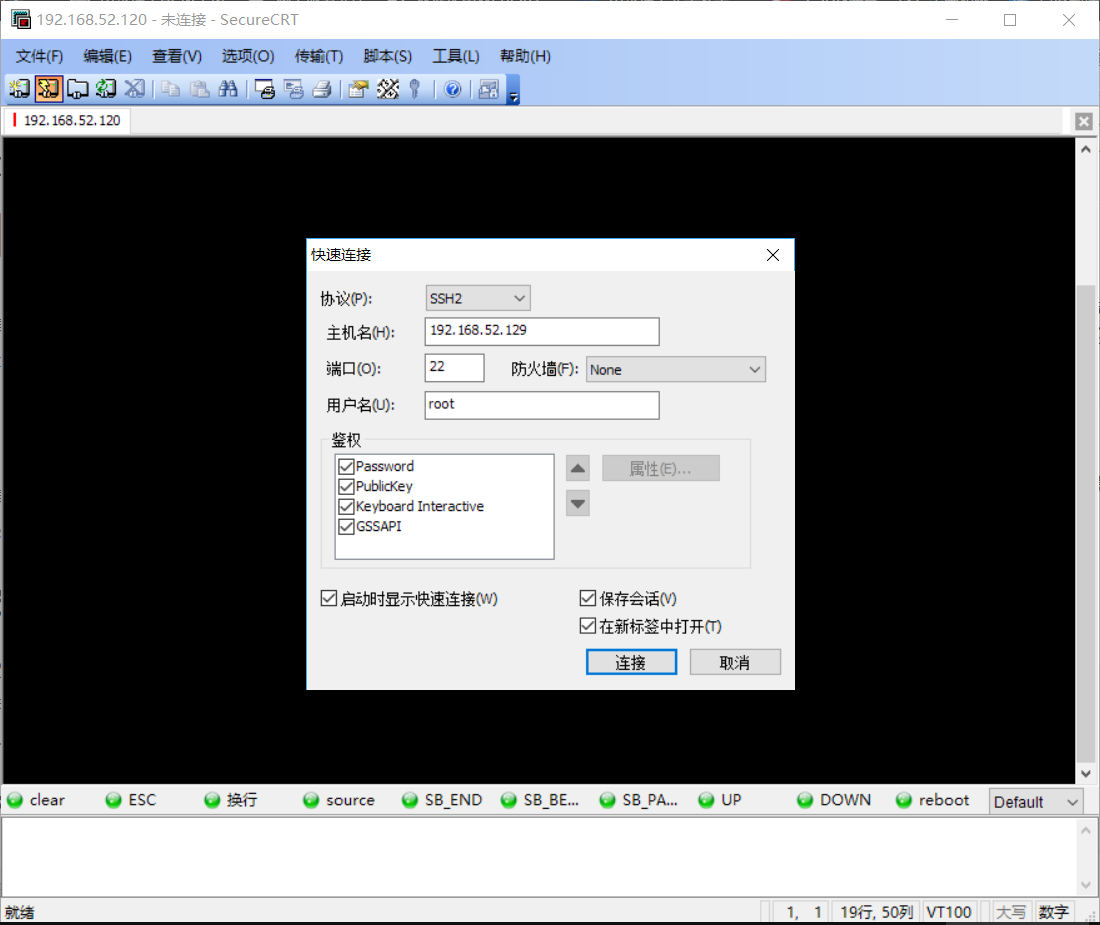
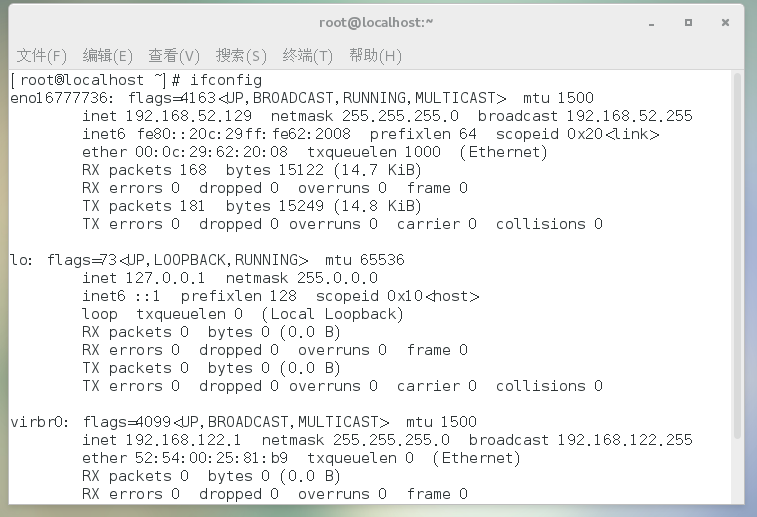






# 远程连接虚拟机

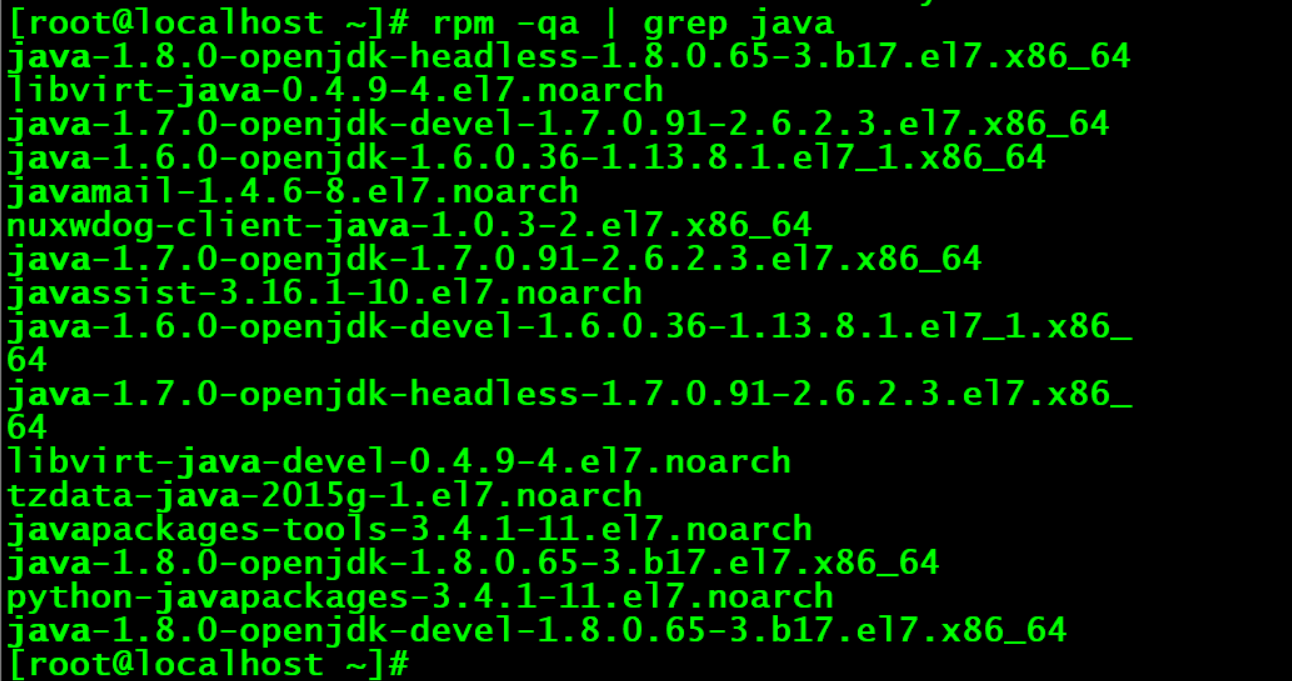
ifconfig



# HDFS伪分布式搭建

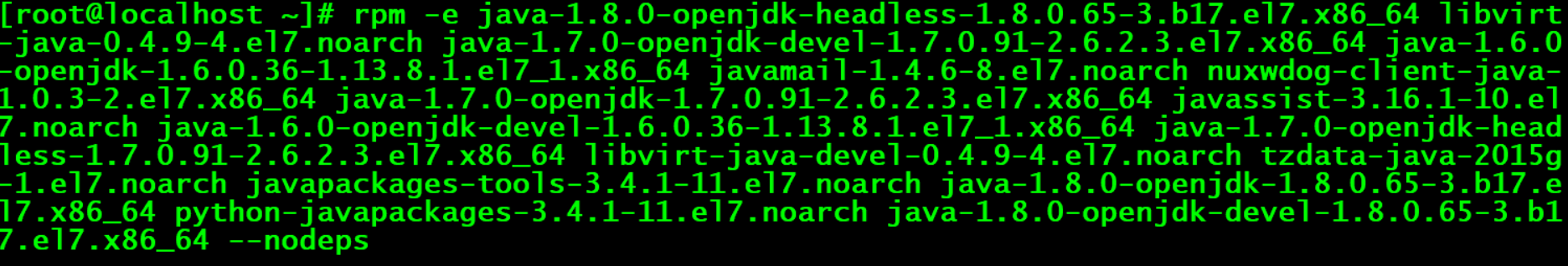
## 查看自带的openjdk

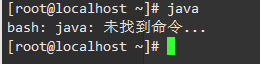
rpm -qa | grep java



## 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



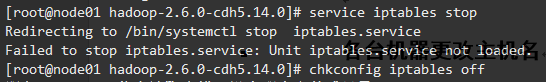


## 各台虚拟机关闭防火墙

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

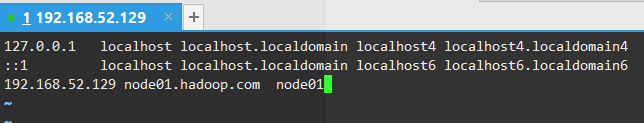
HOSTNAME=node01.hadoop.com



## 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

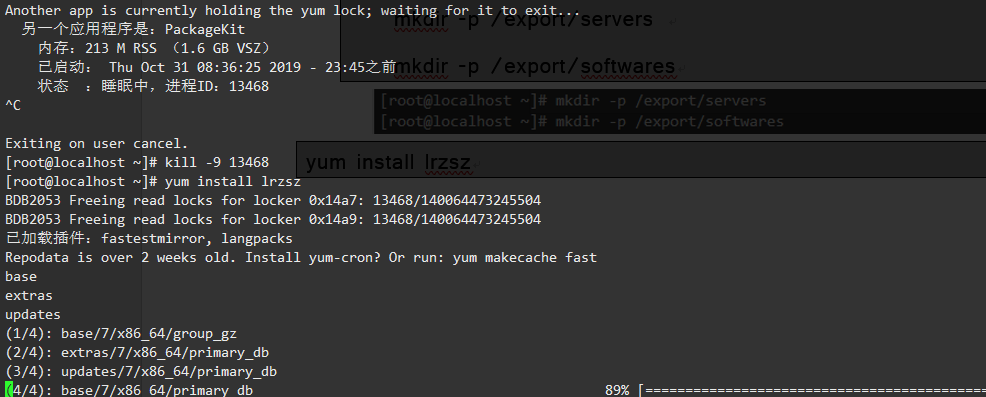
## 创建文件夹

mkdir -p /export/servers

mkdir -p /export/softwares

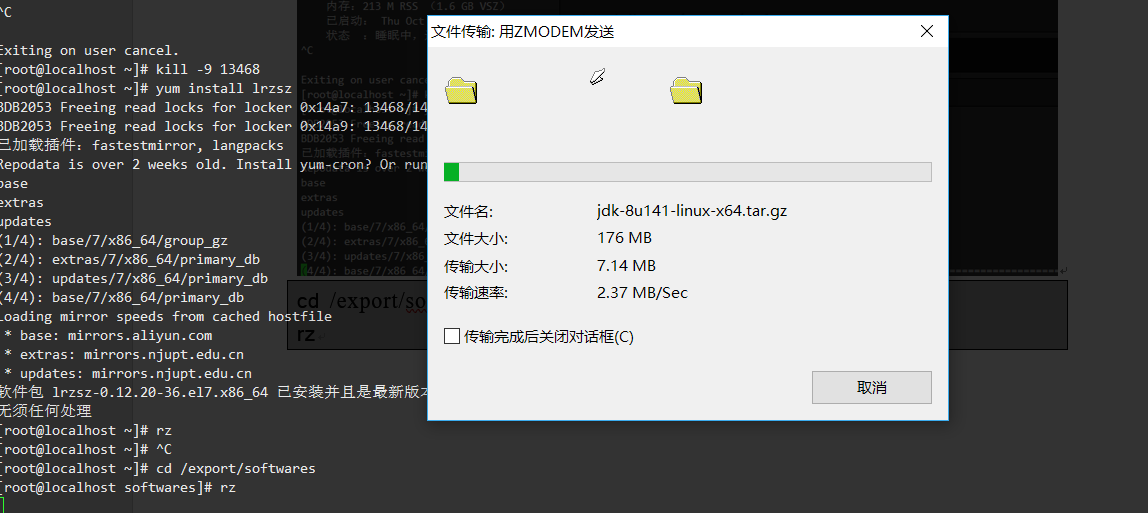


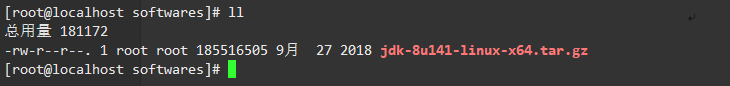
yum install lrzsz



cd /export/softwares

rz





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



## 配置环境变量

vim /etc/profile

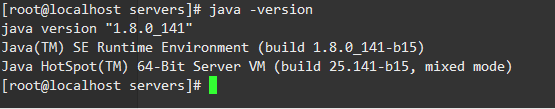
export JAVA\_HOME=/export/servers/jdk1.8.0\_141

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



source /etc/profile



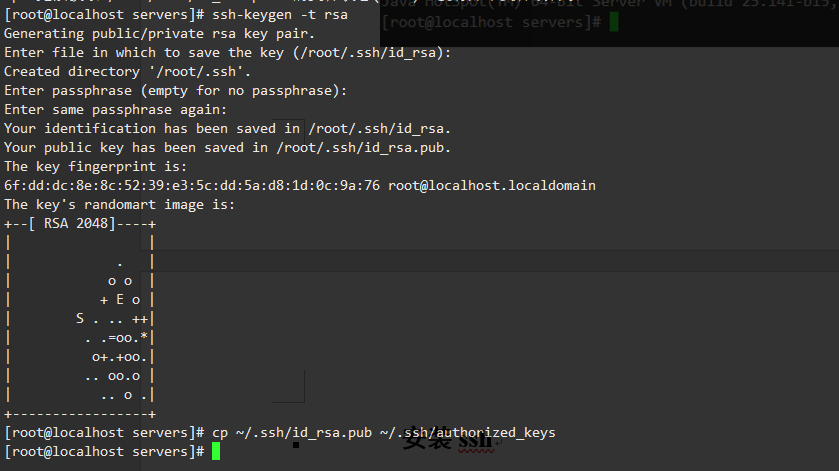


## 安装ssh

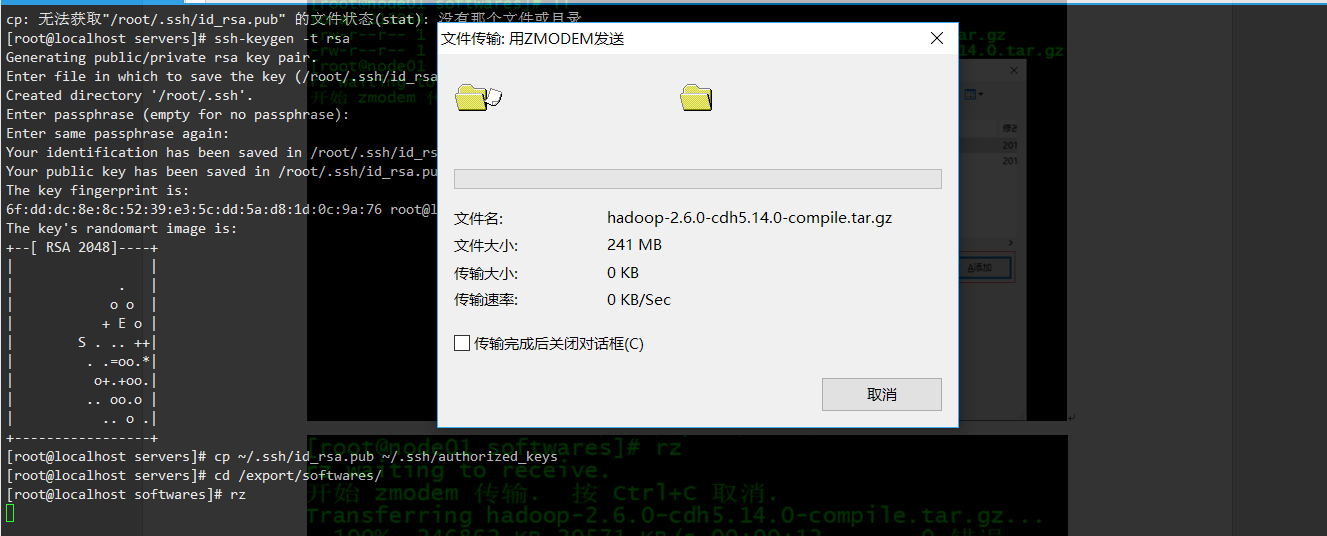
sudo yum install ssh

ssh-keygen -t rsa

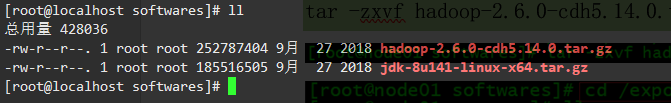
cp ~/.ssh/id\_rsa.pub ~/.ssh/authorized\_keys



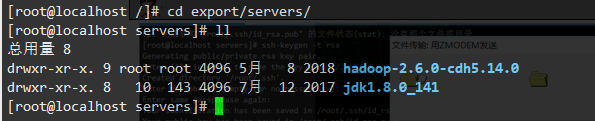
cd /export/softwares/  
rz



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

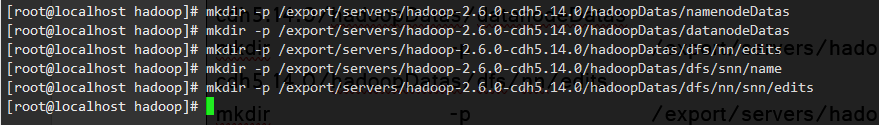
mkdir -p /export/servers/hadoop-2.6.0-cdh5.14.0/hadoopDatas/namenodeDatas

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

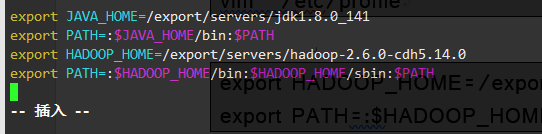


## 配置hadoop的环境变量

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

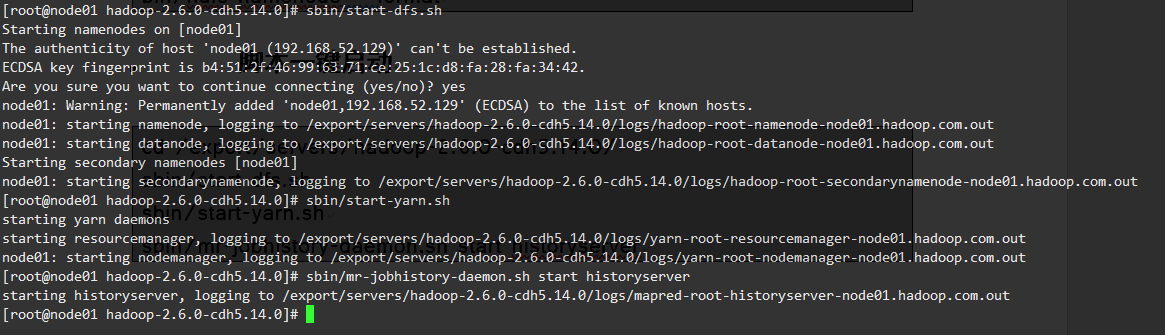
## 脚本一键启动

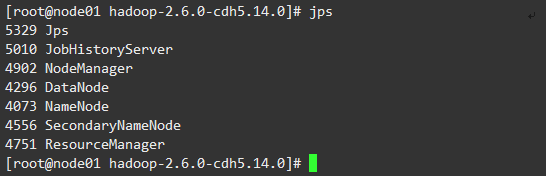
cd /export/servers/hadoop-2.6.0-cdh5.14.0/

sbin/start-dfs.sh

sbin/start-yarn.sh

sbin/mr-jobhistory-daemon.sh start historyserver





## 停止集群

sbin/stop-dfs.sh

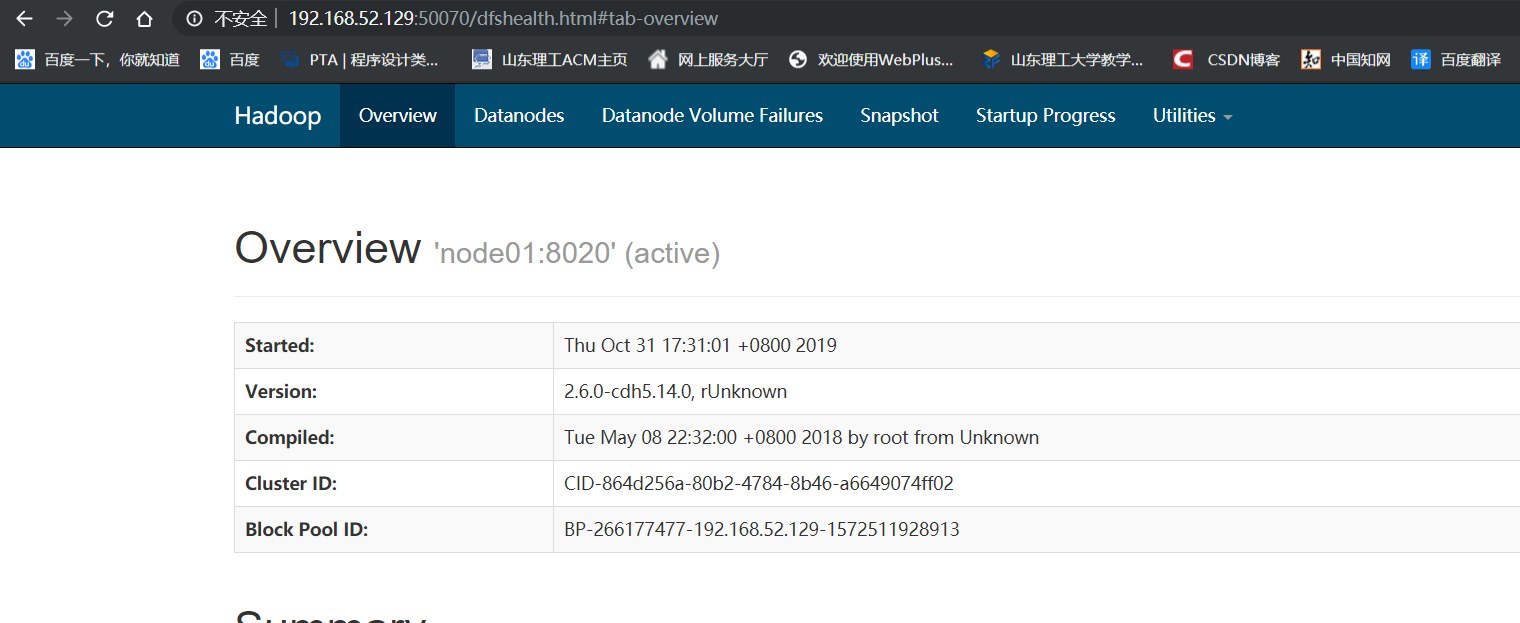
sbin/stop-yarn.sh

sbin/mr-jobhistory-daemon.sh stop historyserver

## 浏览器查看启动页面

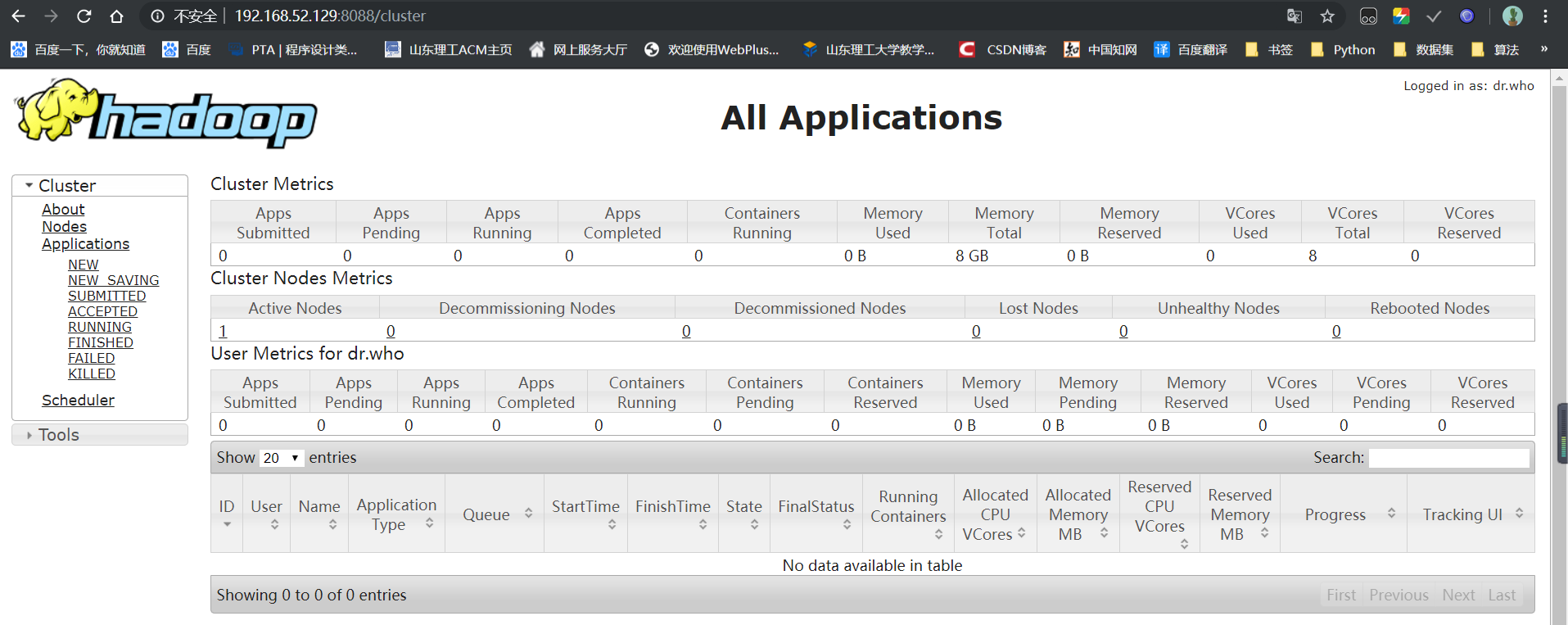
### hdfs集群访问地址

<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

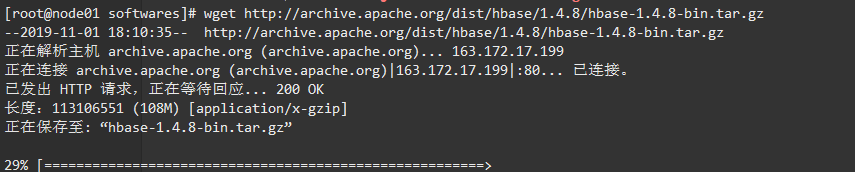
<http://192.168.52.129:19888/jobhistory>



# Hbase安装

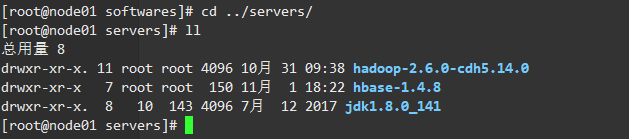
## 使用wget下载Hbase

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/



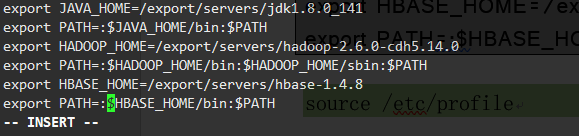
mkdir /export/servers/hbase-1.4.8/zk\_data

## 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>

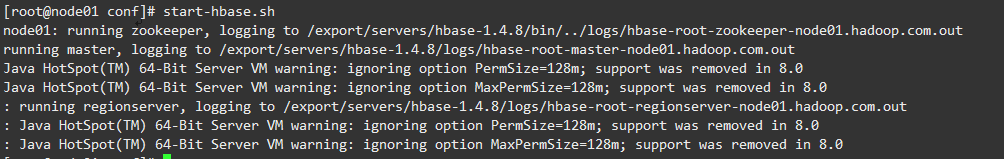
<value>/export/servers/hbase-1.4.8/zk\_data</value>

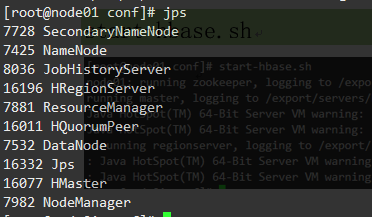
</property>

</configuration>

## 启动hbase

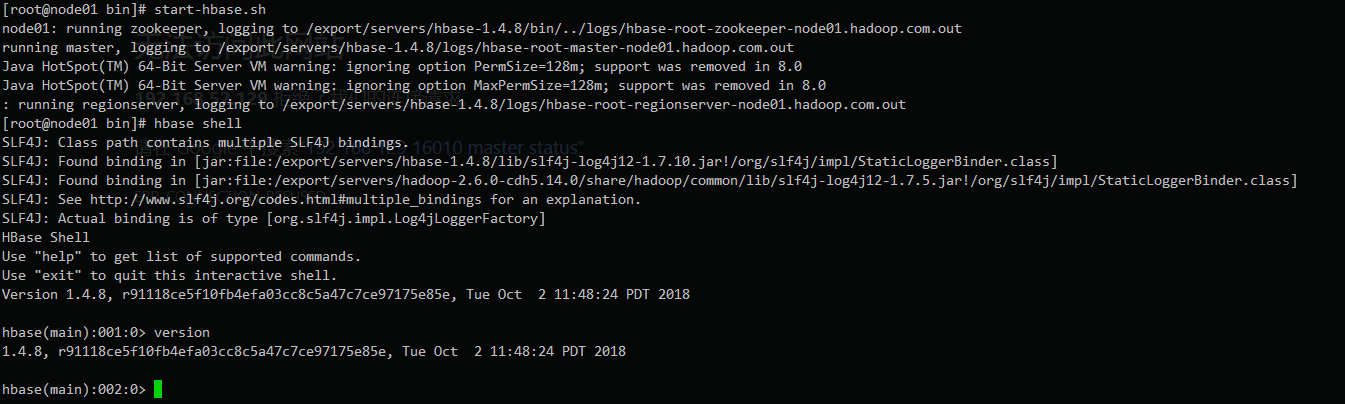
start-hbase.sh





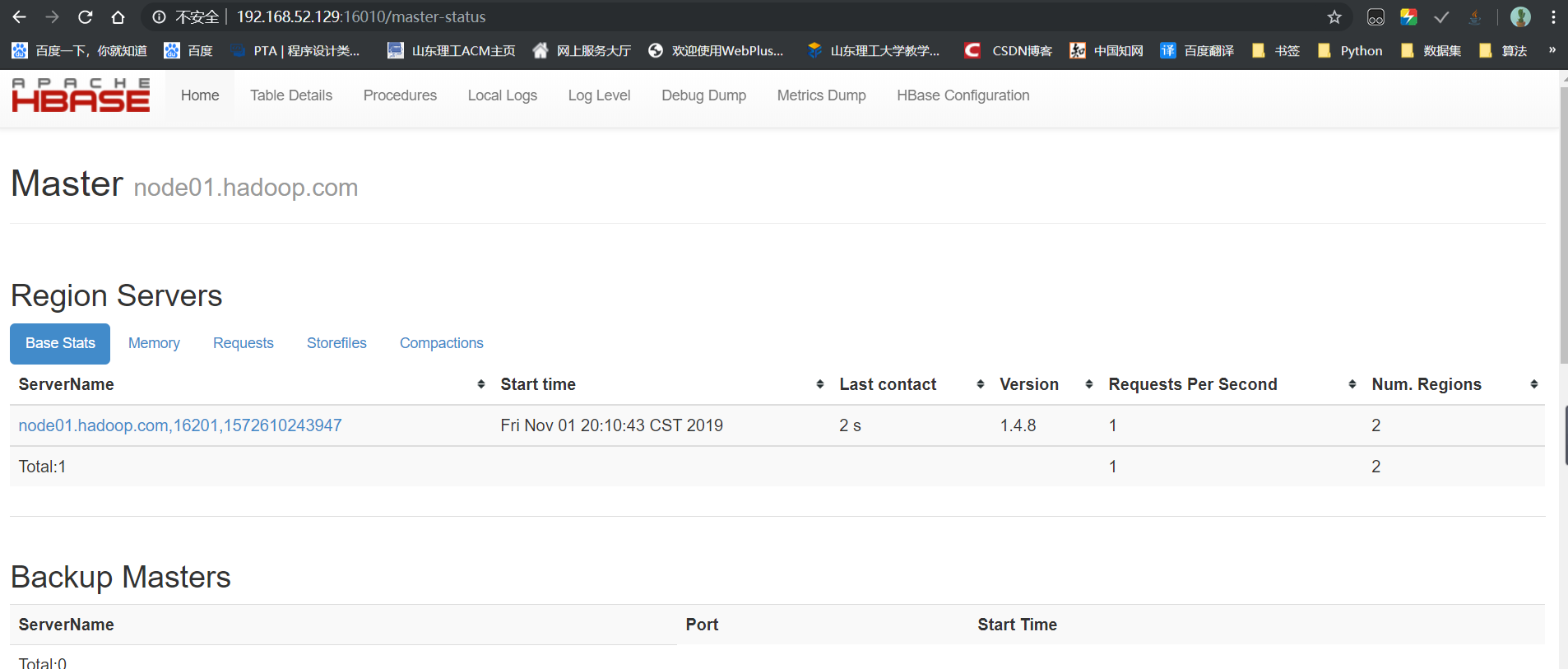
## 进入hbase shell

hbase shell

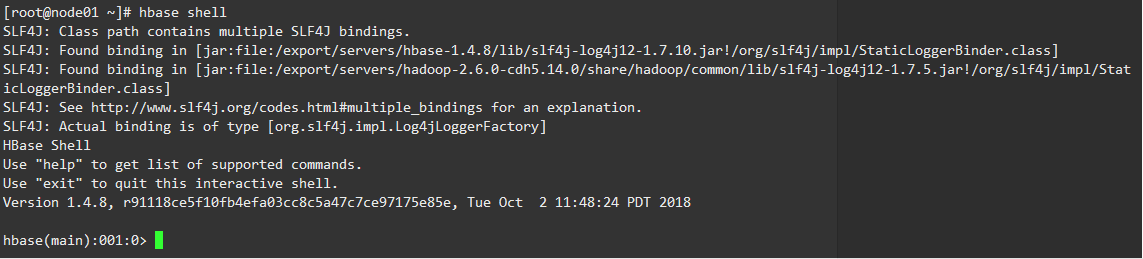


## 进入hbase的web页面

<http://192.168.52.129:16010/>



# HBase shell



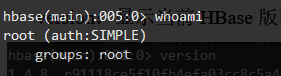
version：显示当前HBase版本号



status：显示各个主节点状态



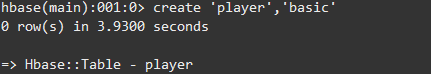
whoami：显示当前用户名



# 表和列族操作

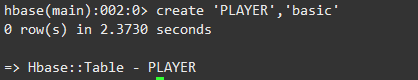
## 创建表

create 'player','basic'



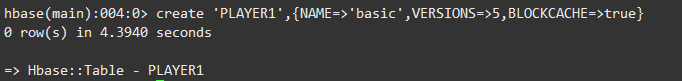
大小写参数敏感

create 'PLAYER','basic'



建表时指定列族的参数

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

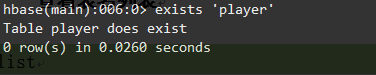


## 查看表名列表

list

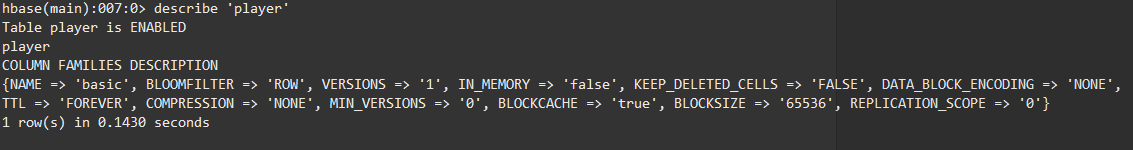


exists 'player'

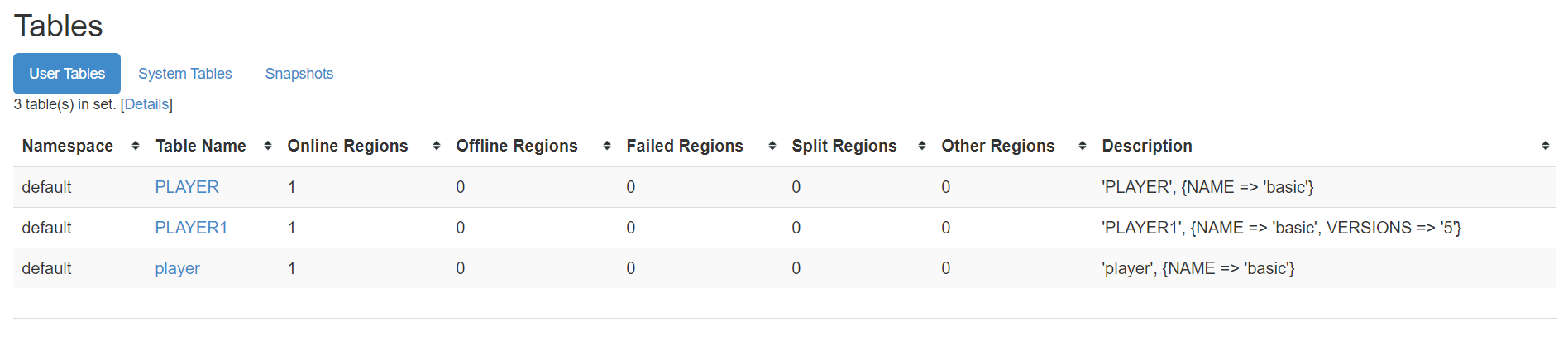


## 描述表结构

describe 'player'



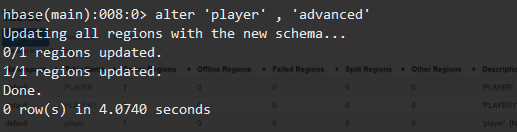
Hbase Web界面



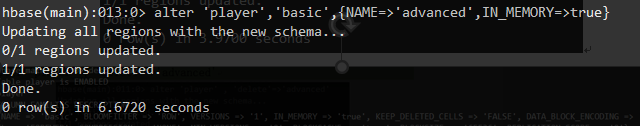
## 修改表结构

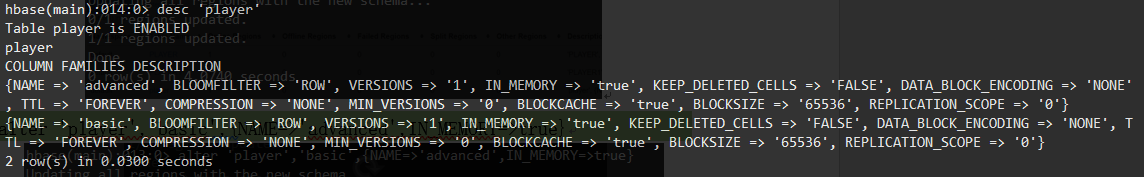
增加列族

alter 'player','advanced'



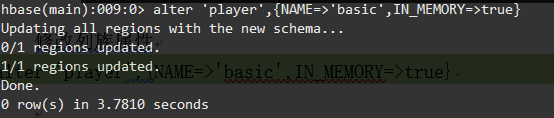
alter 'player','basic',{NAME=>'advanced',IN\_MEMORY=>true}

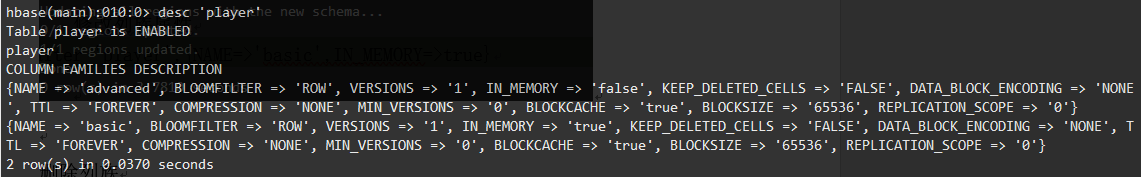




修改列族属性

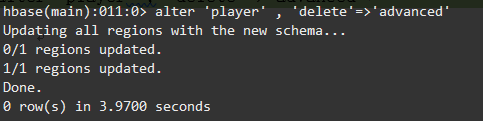
alter 'player',{NAME=>'basic',IN\_MEMORY=>true}

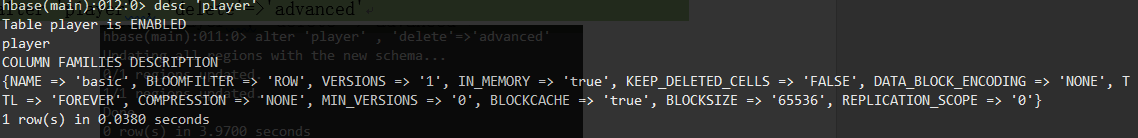




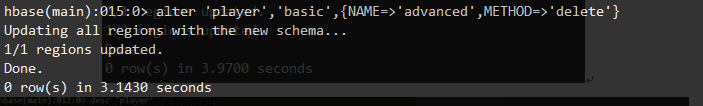
删除列族

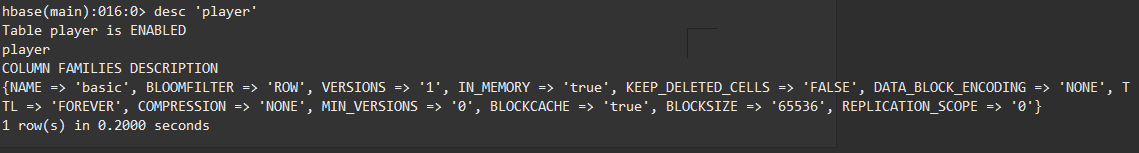
alter 'player' , 'delete'=>'advanced'





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



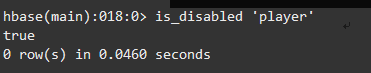


## 删除表

disable 'player'



is\_disabled 'player'



drop 'player'



数据清空

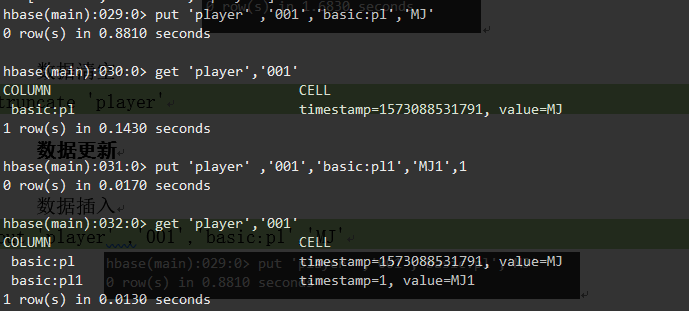
truncate 'player'

## 数据更新

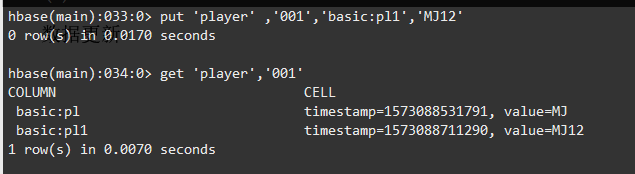
数据插入

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



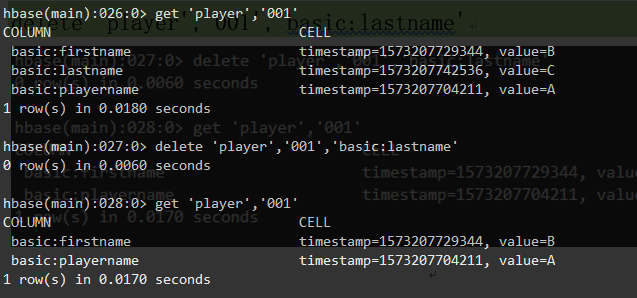


数据更新



数据删除

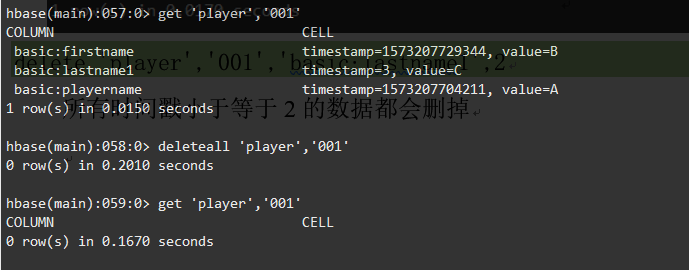
delete 'player','001','basic:lastname'



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

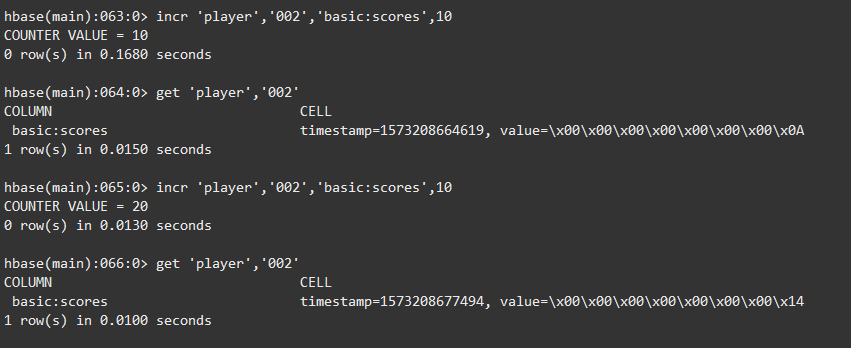
所有时间戳小于等于2的数据都会删掉

deleteall 'player','001'



## 计数器

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

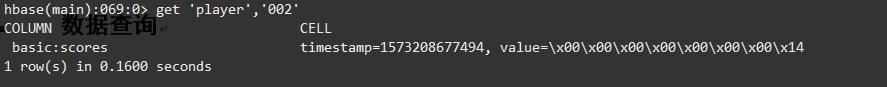


get\_counter 'player' ,'002','basic:scores'



## 数据查询

get 'player','002'



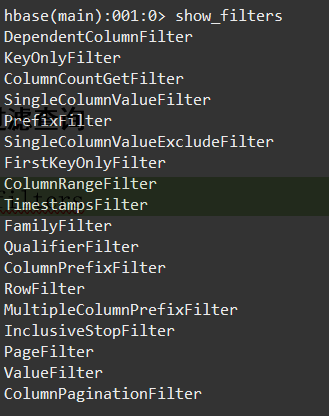
数据扫描

scan 'player'



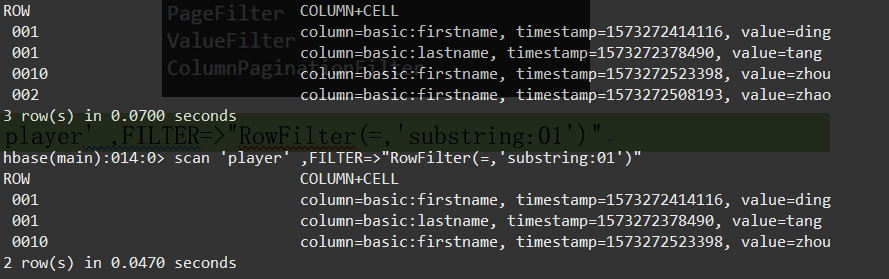
## 过滤查询

show\_filters

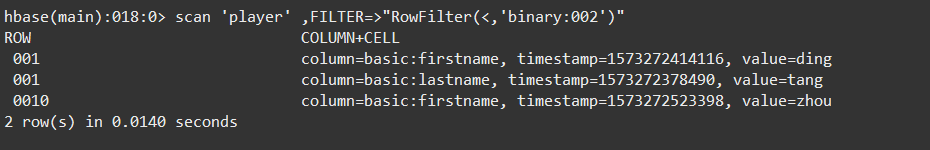


### 行键过滤器

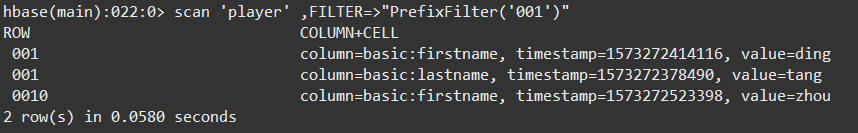
scan 'player' ,FILTER=>"RowFilter(=,'substring:01')"



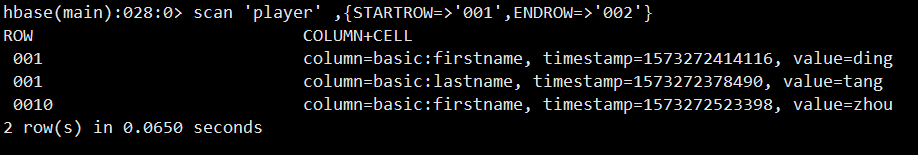
scan 'player' ,FILTER=>"RowFilter(<,'binary:002')"



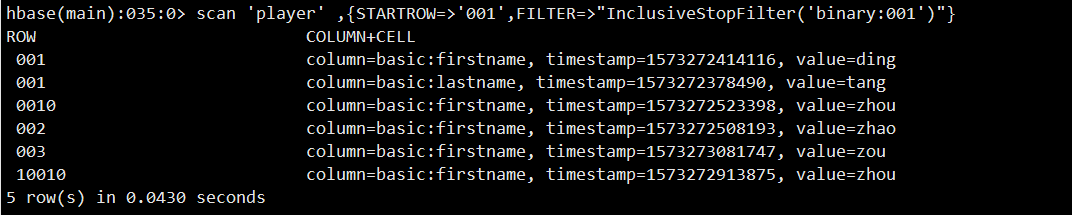
scan 'player' ,FILTER=>"PrefixFilter('001')"



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

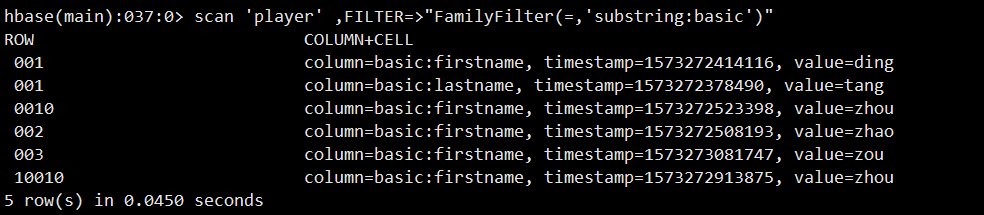


scan 'player' ,{STARTROW=>'001',FILTER=>"InclusiveStopFilter('binary:001')"}

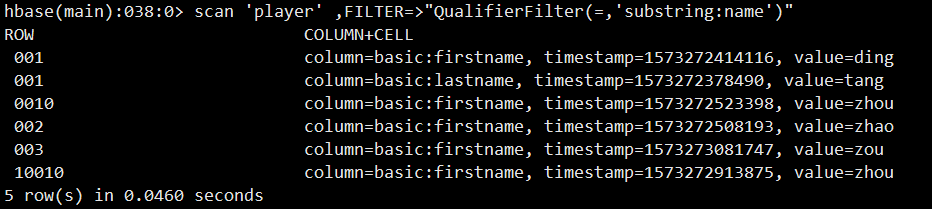


### 列族和列过滤器

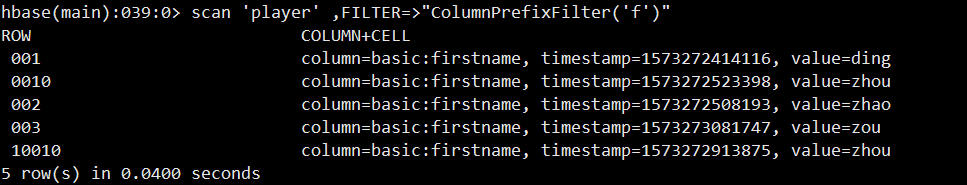
scan 'player' ,FILTER=>"FamilyFilter(=,'substring:basic')"



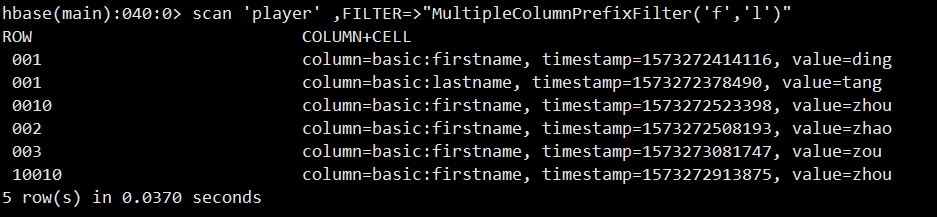
scan 'player' ,FILTER=>"QualifierFilter(=,'substring:name')"



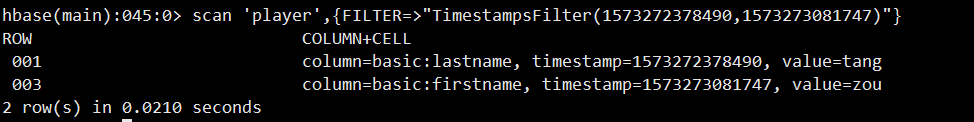
scan 'player' ,FILTER=>"ColumnPrefixFilter('f')"



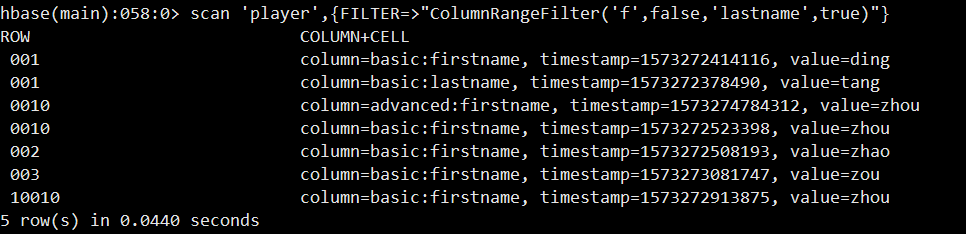
scan 'player' ,FILTER=>"MultipleColumnPrefixFilter('f','l')"



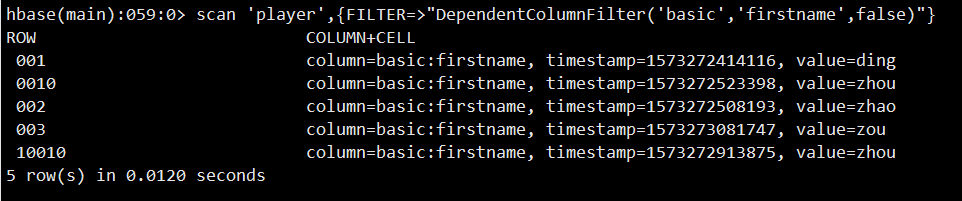
scan 'player',{FILTER=>"TimestampsFilter(1573272378490,1573273081747)"}



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

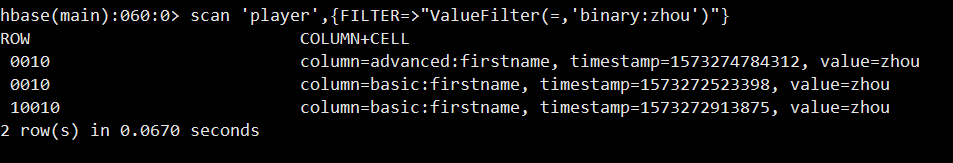


scan 'player',{FILTER=>"DependentColumnFilter('basic','firstname',false)"}

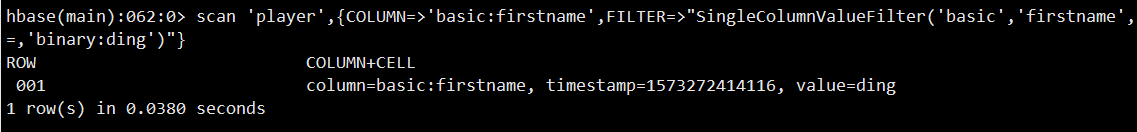


### 值过滤器

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

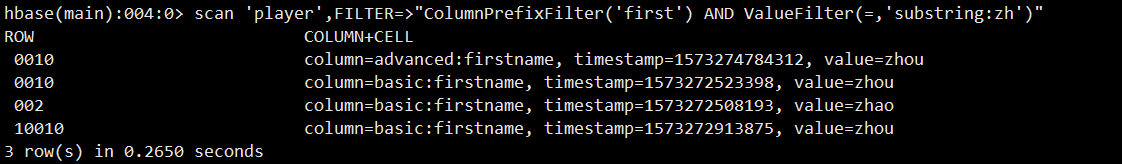


scan 'player',{COLUMN=>'basic:firstname',FILTER=>"SingleColumnValueFilter('basic','firstname',=,'binary:ding')"}



## 其他过滤器

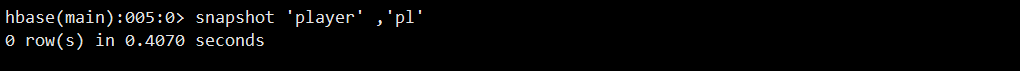
scan 'player',FILTER=>"ColumnPrefixFilter('first') AND ValueFilter(=,'substring:zh')"



## 快照操作

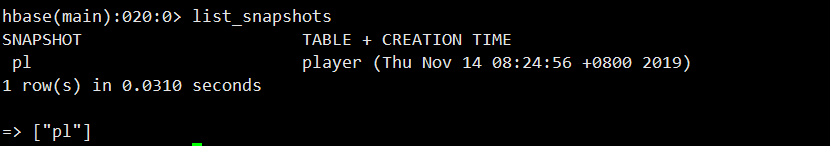
建立快照

snapshot 'player' ,'pl'



显示快照列表

list\_snapshots



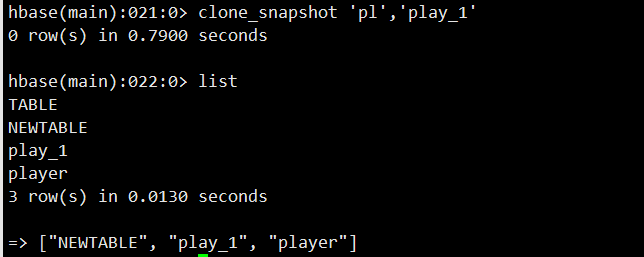
删除快照

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{

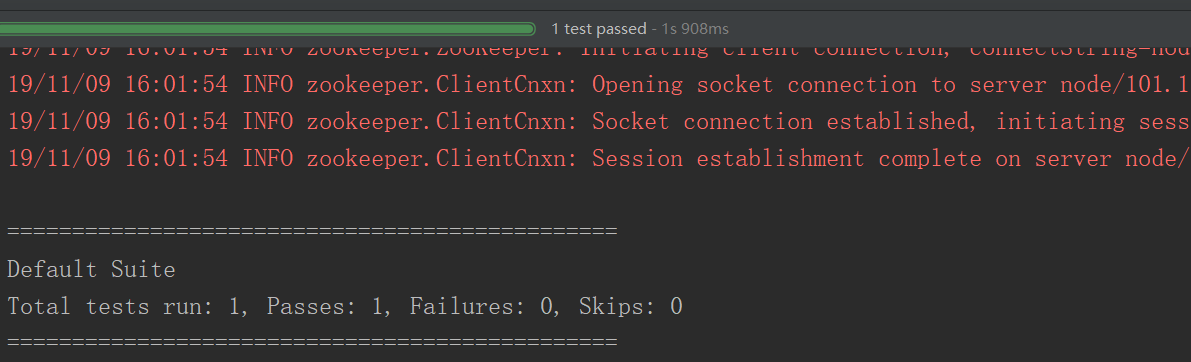
connection=ConnectionFactory.createConnection(conf);

} 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);

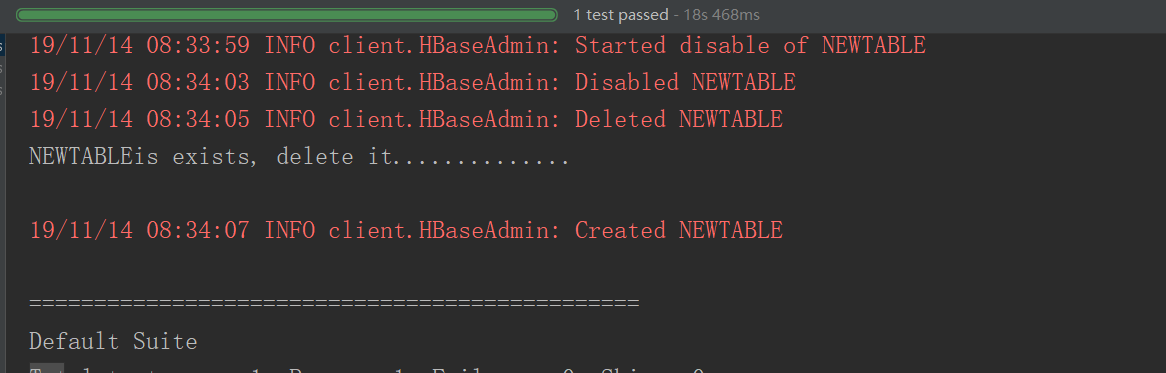
descriptor.addFamily(columnDescriptor);

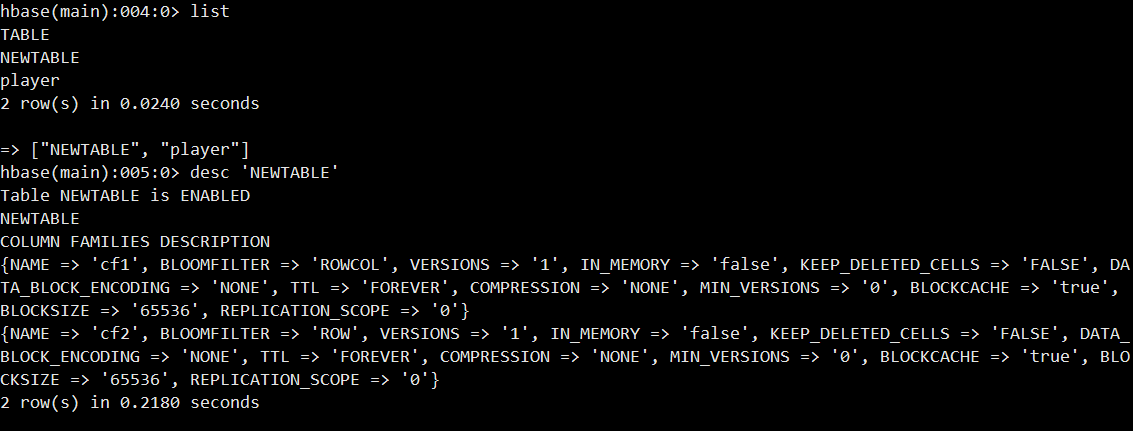
descriptor.addFamily(new HColumnDescriptor("cf2"));

admin.createTable(descriptor);

admin.close();

}





## 描述表结构

public void desctable() throws IOException {

getconncet();

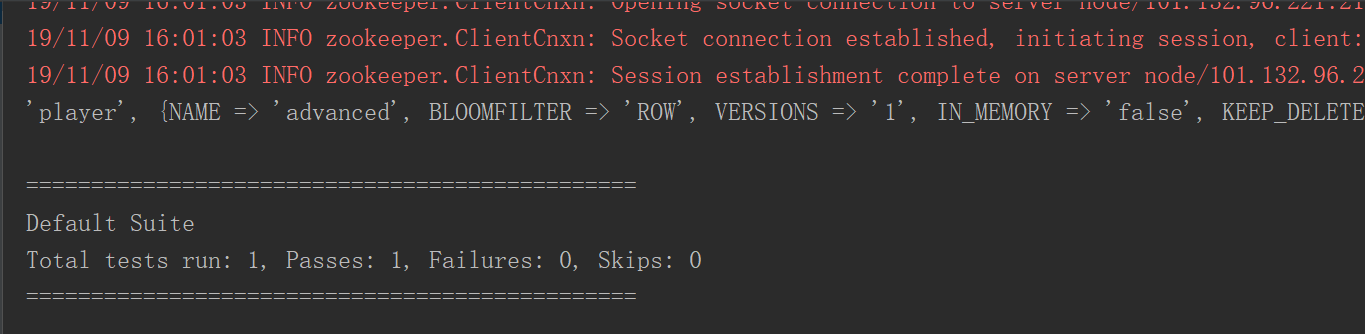
TableName tableName =TableName.valueOf("player");

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());

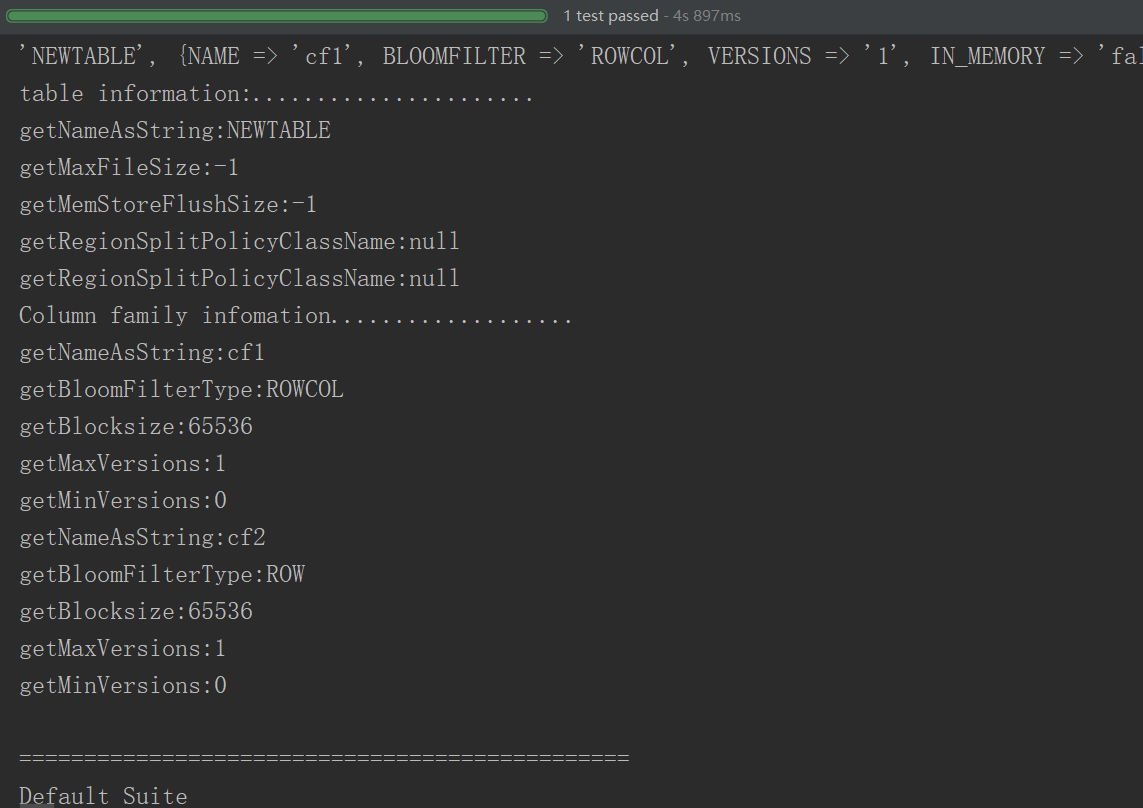
System.out.println("getMaxVersions:"+result.getMaxVersions());

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);

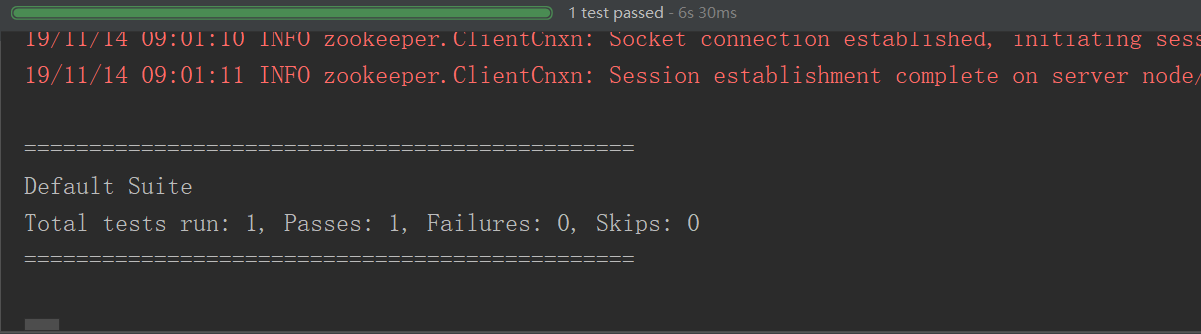
putList.add(put4);

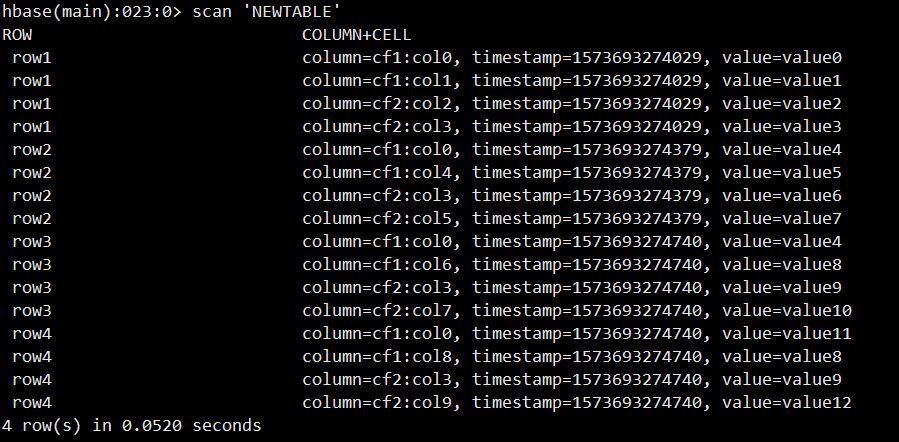
table.put(putList);

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);

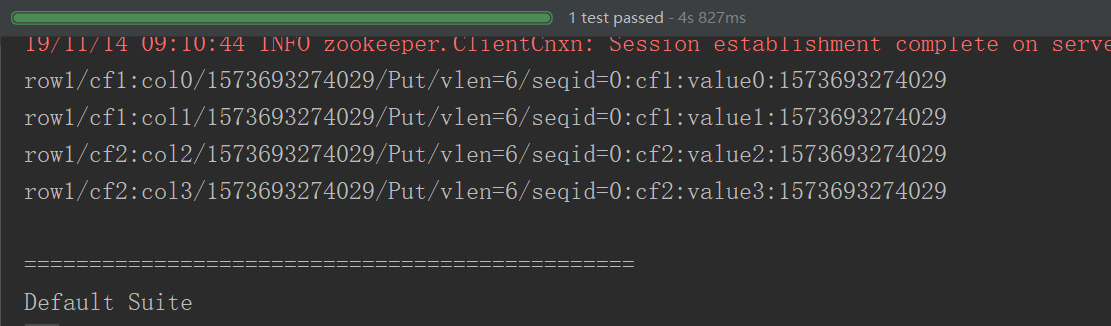
for (Cell cell:result.rawCells()){

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"));

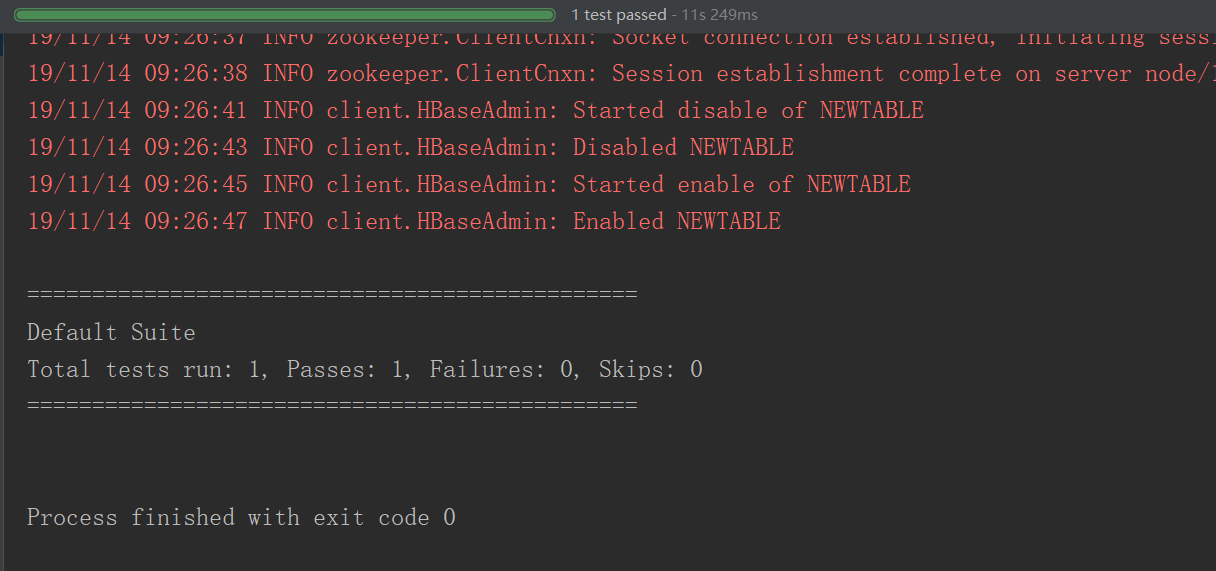
admin.disableTable(tableName);

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);

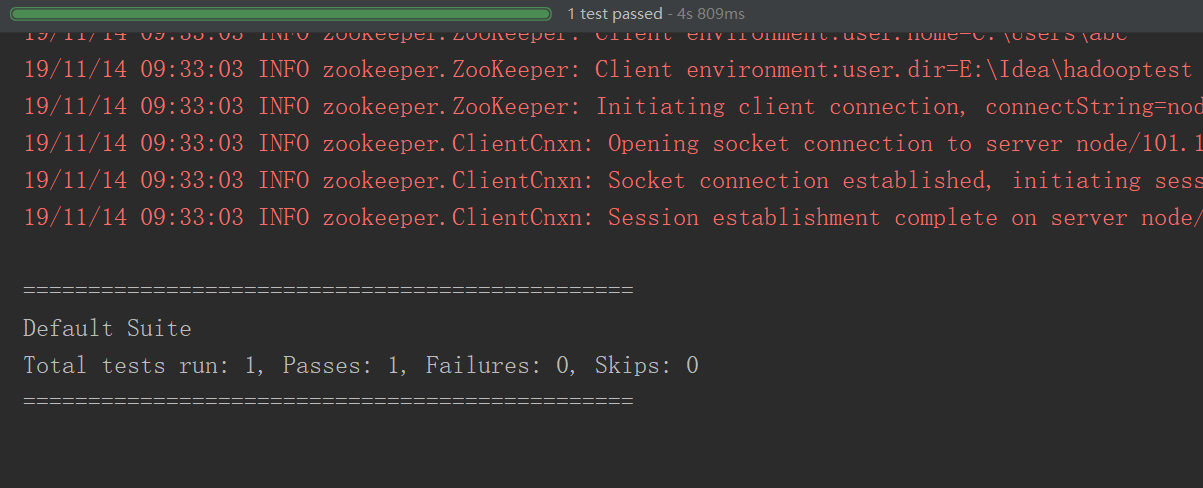
table.delete(delete3);

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());

}

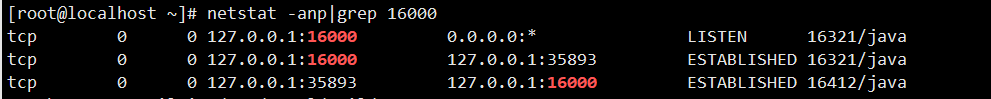
}

}

## 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



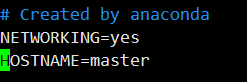
配置Linux的hostname

vim /etc/sysconfig/network

NETWORKING=yes

HOSTNAME=master

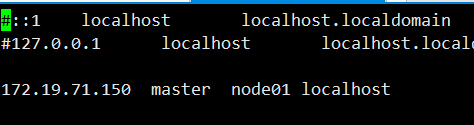
这里配置的hostname要Linux重启才生效，为了不重启就生效，我们可以执行：hostname master命令，暂时设置hostname



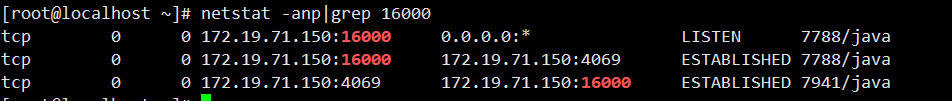
配置Linux的hosts，映射ip的hostname的关系

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

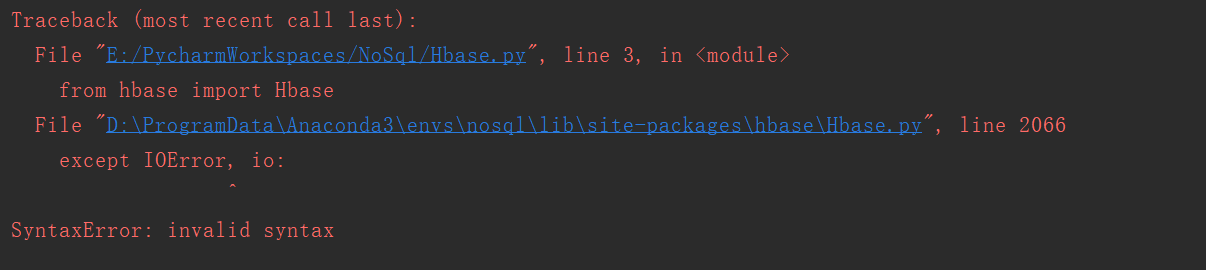
conda activate nosql

conda install thrift

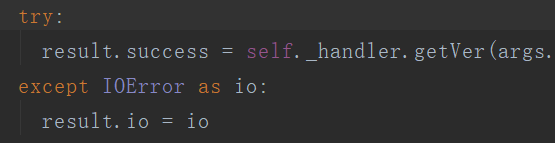
pip install hbase-thrift

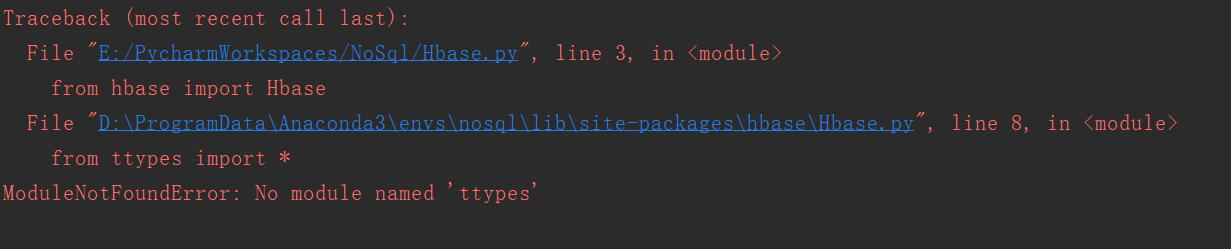
## 修改代码文件

将Python2风格代码改为Python3

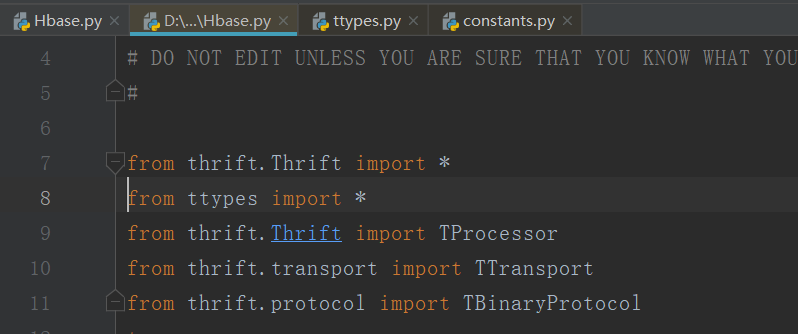
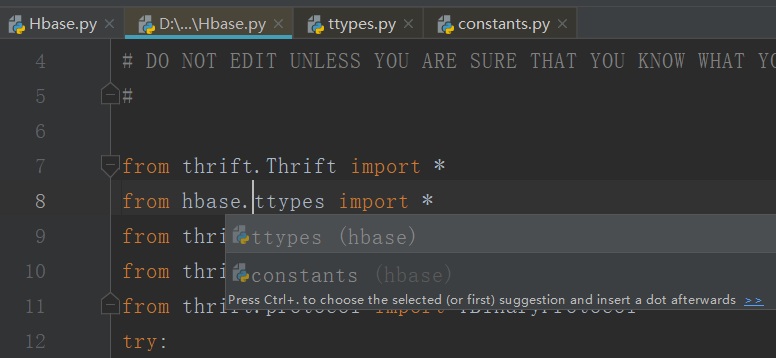


改为



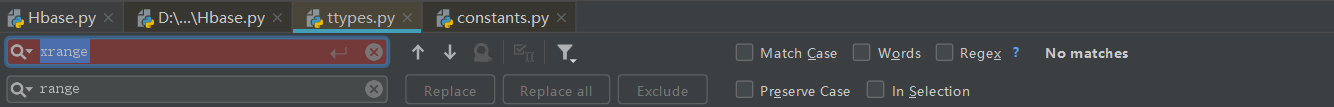


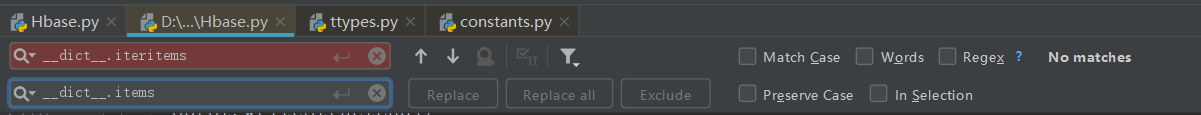
改为

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

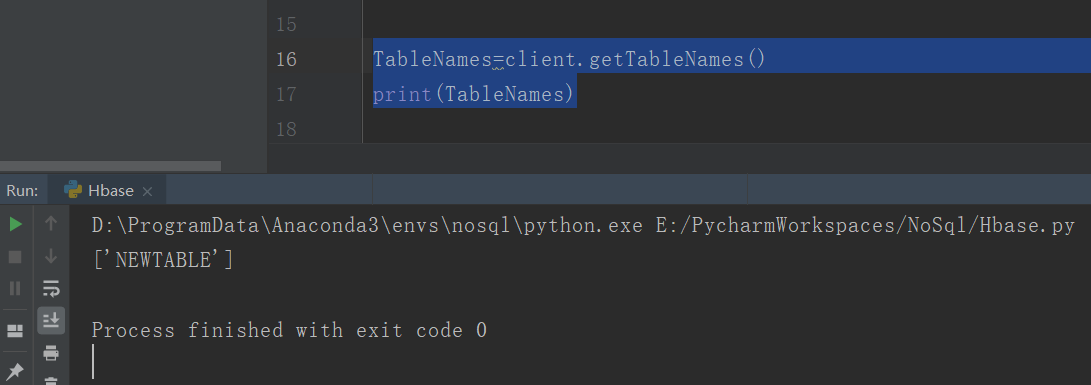
transport.close()



## 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:

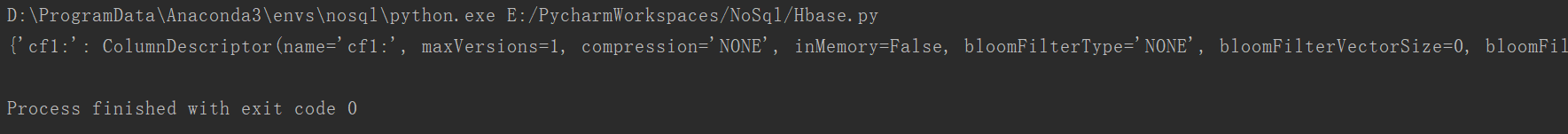
pass



## 查看表结构

ColumnDescriptors=client.getColumnDescriptors('testtable')

print(ColumnDescriptors)



## 插入数据

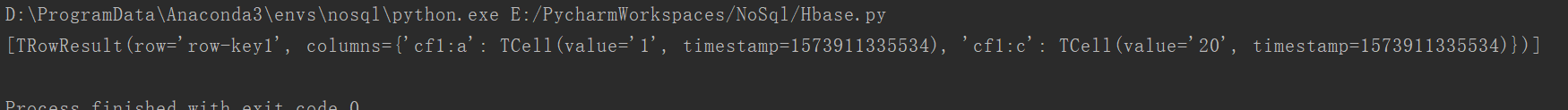
mutations=[Mutation(column='cf1:a',value='1'),Mutation(column='cf1:c',value='20')]

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

## 检索数据

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

print(result)



for r in result:

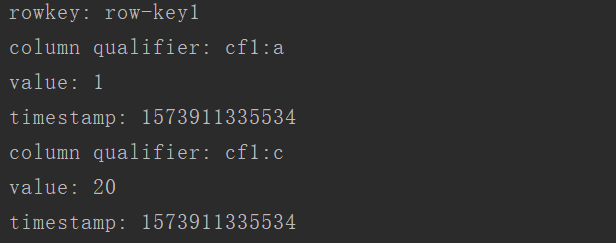
print('rowkey:',r.row)

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)