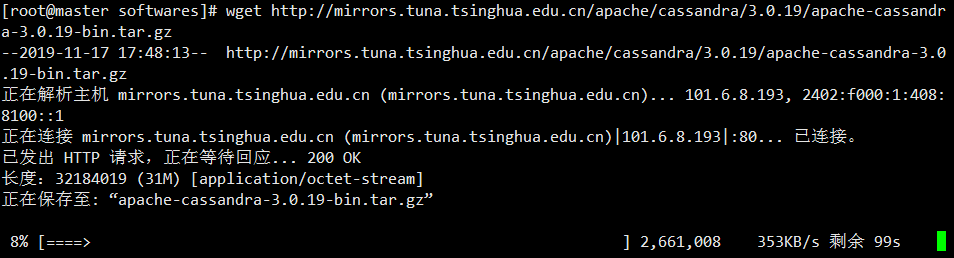
## 安装Cassandra

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

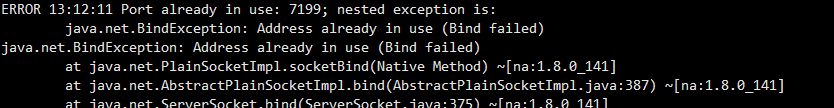
scp -r apache-cassandra-3.0.19/ node02:$PWD

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

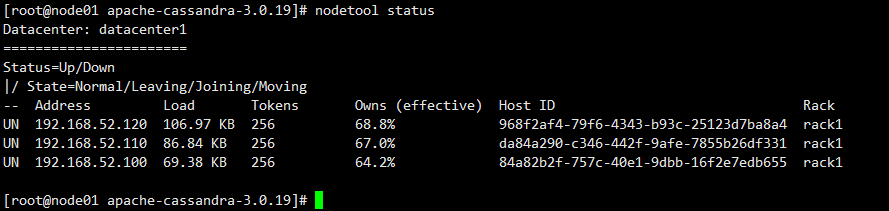


netstat -tunlp |grep 7199

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/

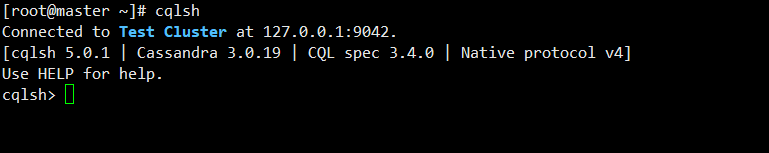
cp -p /usr/lib64/python2.6/site-packages/sqlitecachec.pyc /usr/local/lib/python2.7/site-packages/

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

## cqlsh基本用法

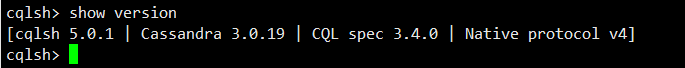
进入shell

cqlsh



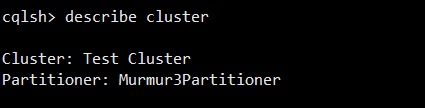
查看版本信息

show version



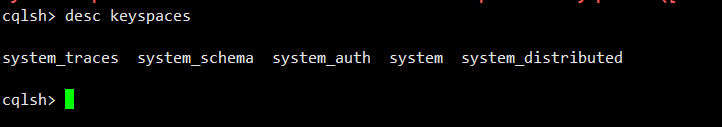
描述集群信息

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)

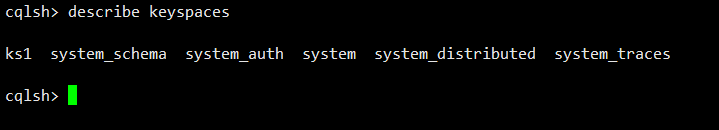
create keyspace ks1 with replication={'class':'NetworkTopologyStrategy','dc1':3,'dc2':2 } AND DURABLE\_WRITES=false;

### 删除键空间

drop keyspace ks1

### 查看键空间列表

describe keyspaces

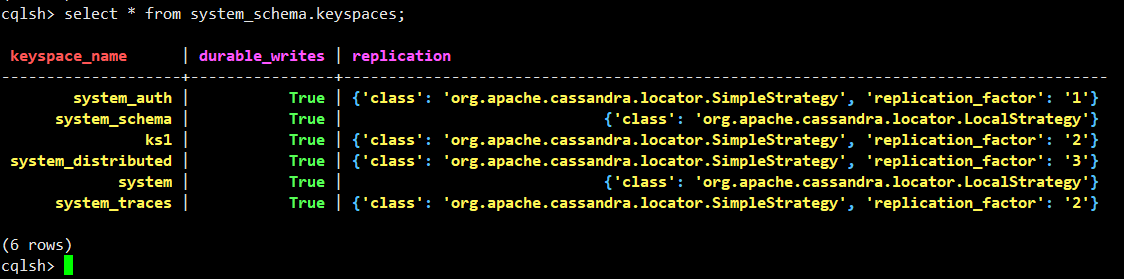


### 修改键空间属性

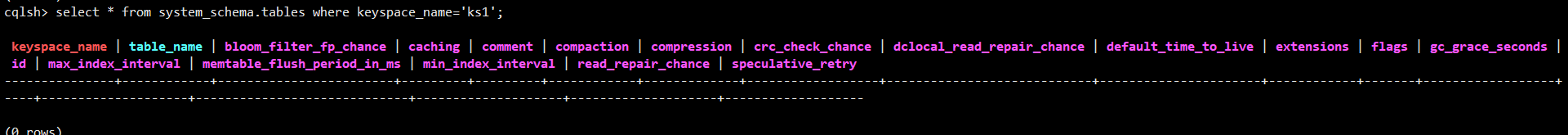
alter keyspace ks1 with replication ={'class':'SimpleStrategy','replication\_factor':'2'};

### 系统键空间

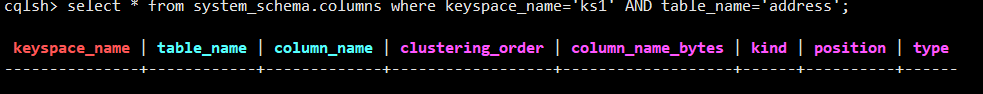
select \* from system\_schema.keyspaces;



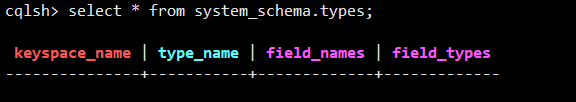
select \* from system\_schema.tables where keyspace\_name='ks1';



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



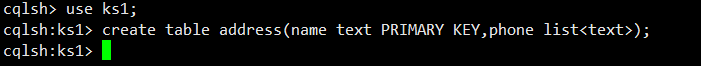
select \* from system\_schema.types;



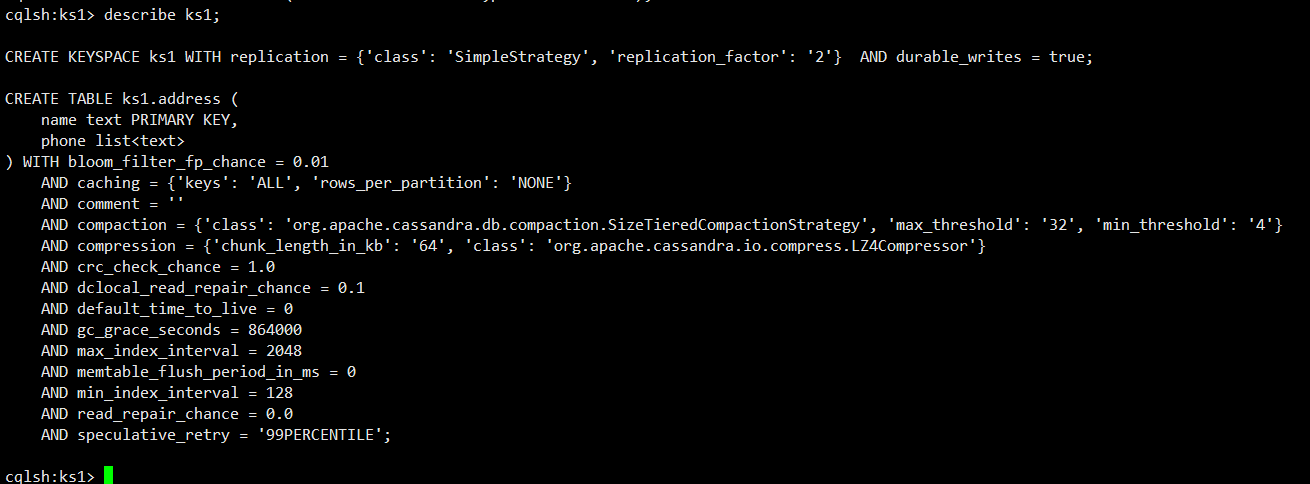
## 数据表管理

### 建立数据表

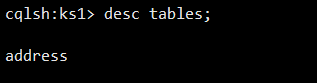
create table address(name text PRIMARY KEY,phone list<text>);



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,

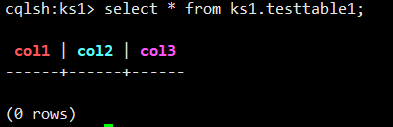
col2 int,

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);

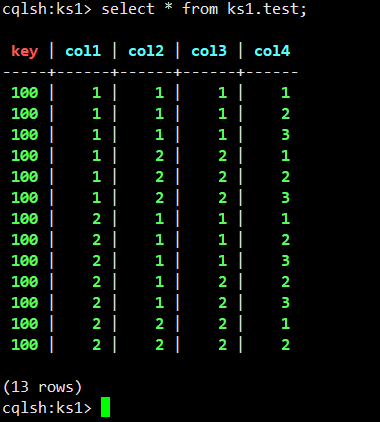
insert into ks1.test(key,col1,col2,col3,col4) values(100,2,1,1,2);

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

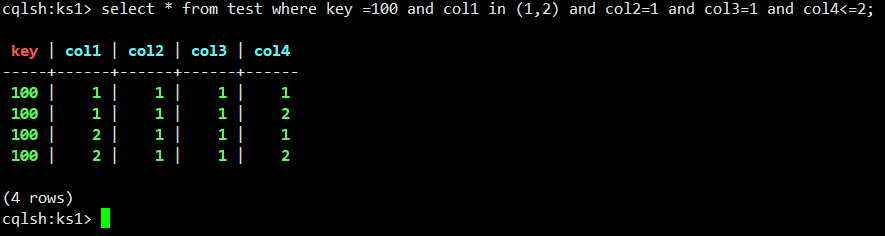
insert into ks1.test(key,col1,col2,col3,col4) values(100,2,2,2,1);

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);

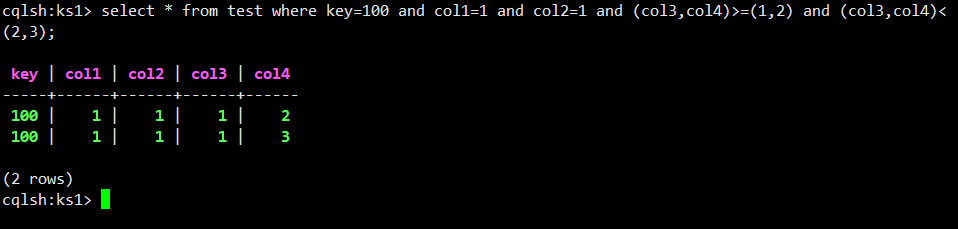


select \* from test where key =100 and col1 in (1,2) and col2=1 and col3=1 and col4<=2;



### 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

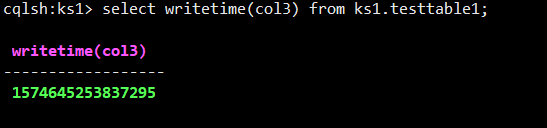
create index indexofaddress on address(age) ;

删除索引

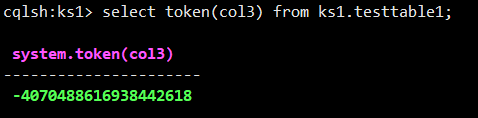
drop index indexofaddress;

## 使用标量函数

select writetime(col3) from ks1.testtable1;



select token(col3) from ks1.testtable1;



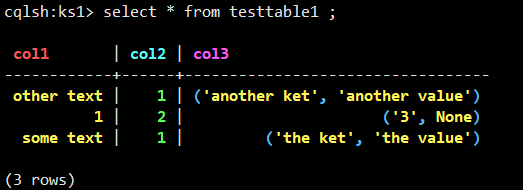
# 数据更新

## 插入更新删除

### 数据插入

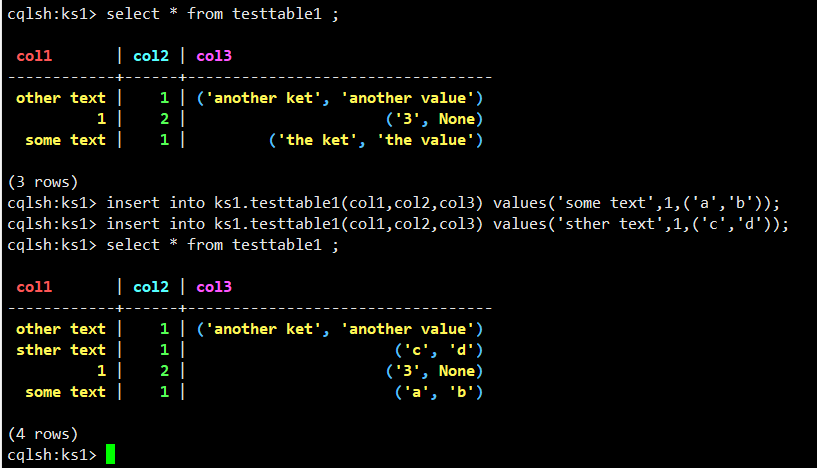
insert into ks1.testtable1(col1,col2,col3) values('some text',1,('the ket','the value'));

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



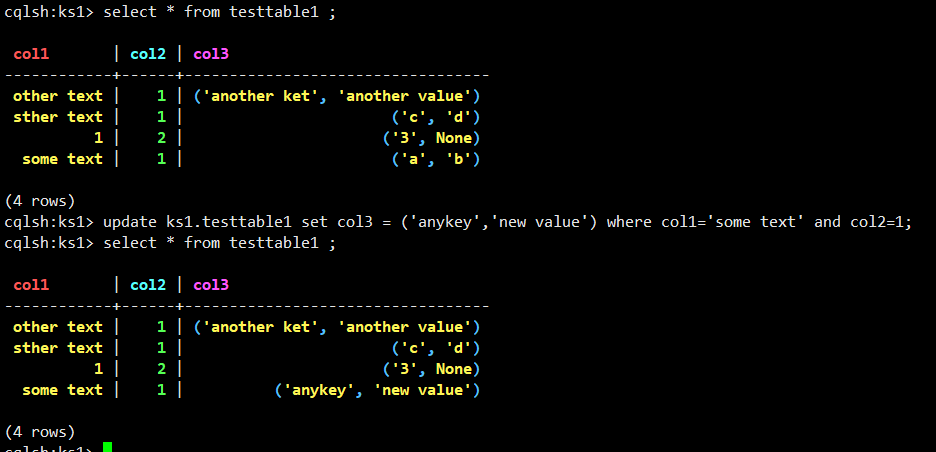
insert into ks1.testtable1(col1,col2,col3) values('some text',1,('a','b'));

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;

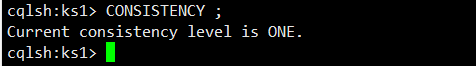
### json格式插入数据

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>,

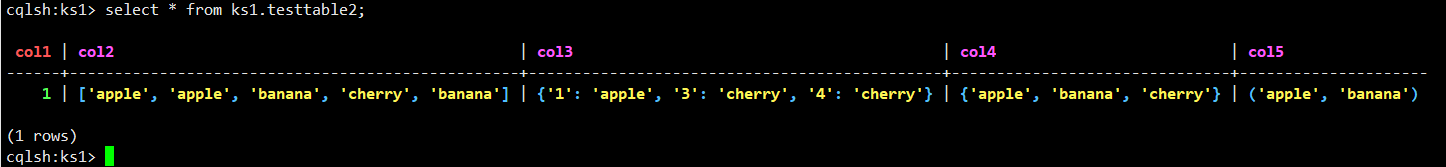
col3 map<text,text>,

col4 set<text>,

col5 frozen<tuple<text,text>>

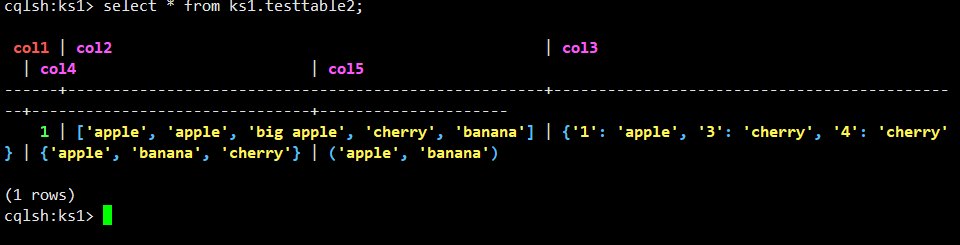
);

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'));



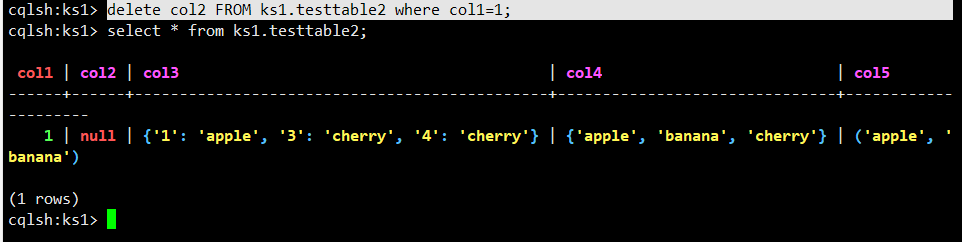
### list类型更新删除

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



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

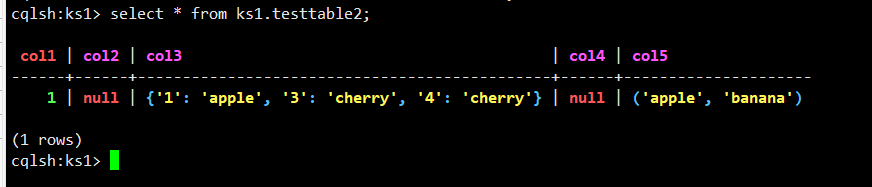
delete col2 FROM ks1.testtable2 where col1=1;



### 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;



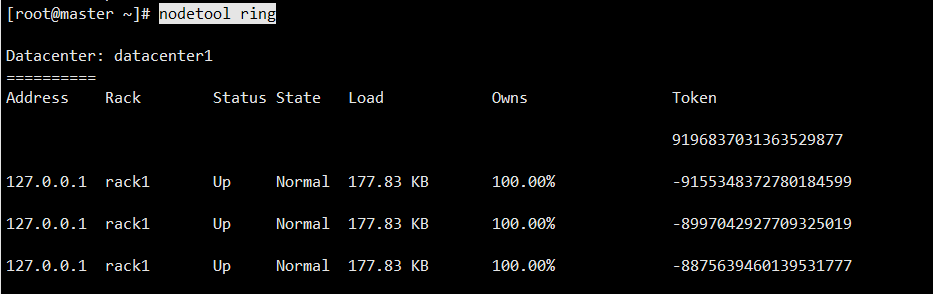
## nodetool工具

## 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();

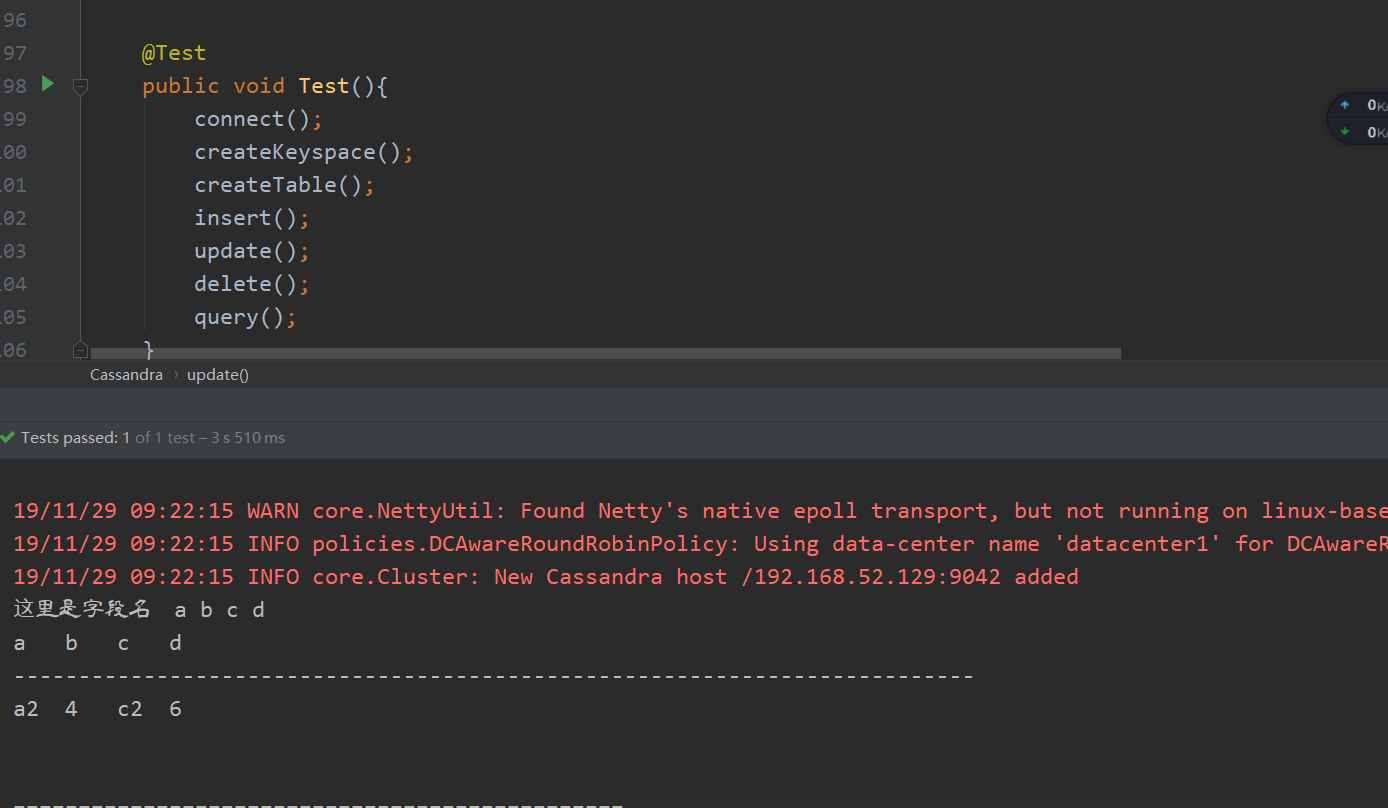
update();

delete();

query();

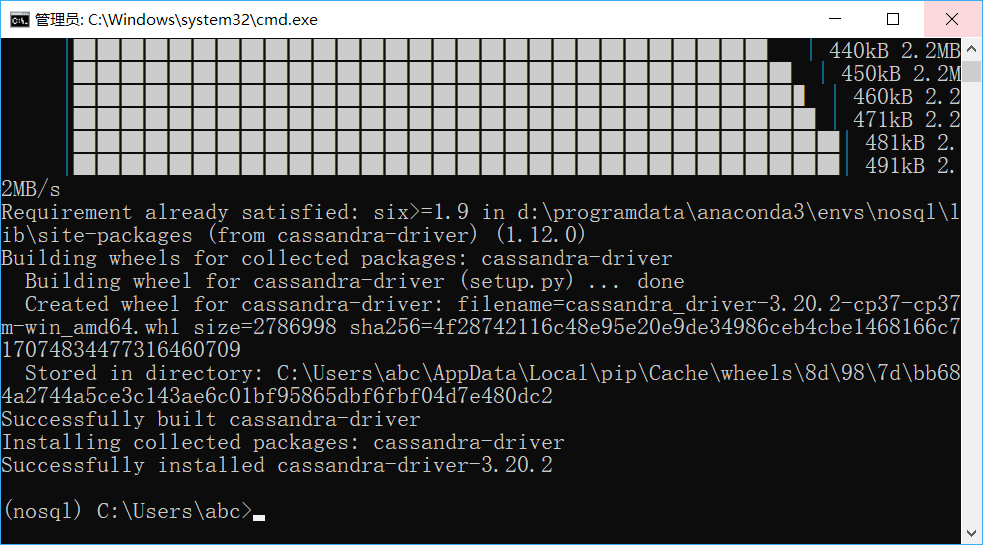
}

}



## 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

