实验3 HBase

实验要求

1.下载并安装HBase，尝试单机Standalone模式、伪分布式模式、集群模式（可选）。
2.以伪分布式运行HBase，编写Java程序，完成下列任务：（1）创建讲义中的students表；（2）扫描创建后的students表；（3）查询学生来自的省；（4）增加新的列Courses:English，并添加数据；（5）增加新的列族Contact和新列Contact:Email，并添加数据；（6）删除students表。
3.再用shell完成上述Java程序的任务。
4.撰写实验报告，要求提交代码，记录步骤，给出运行截图。汇报在安装运行HBase过程中出现的问题，并给出解决方案（未出现问题则可免）。

实验环境

操作系统：CentOS 7.8
Hadoop版本：2.7.7
HBase版本：1.2.6

HBase单机模式安装

下载和配置环境变量

先从官网上下载HBase，我下载的版本是1.2.6，然后传到CentOS主机上。

将压缩文件解压到指定目录：

```
[root@node3 ~]# tar -zxvf hbase-1.2.6-bin.tar.gz -C /usr/local
```

还可以在/etc/profile中设置一下环境变量，方便打命令：

```
export HBASE_HOME=/usr/local/hbase1.2
export PATH=$PATH:$HBASE_HOME/bin
```

配置文件

然后编辑hbase目录下的conf/hbase-env.sh，设置JAVA_HOME路径：

```
# The java implementation to use.  Java 1.7+ required.
export JAVA_HOME=/usr/local/jdk1.8
```

编辑conf/hbase-site.xml，这是主要的HBase配置文件。此时，只需要在本地文件系统上指定HBase和ZooKeeper写入数据的目录。默认情况下，在/ tmp下创建一个新目录。许多服务器配置为在重新引导时删除/tmp的内容，因此应该将数据存储在其他位置。以下配置会将HBase的数据存储在当前hbase1.2的目录中。

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///usr/local/hbase1.2/hbase</value>
  </property>
  <property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/usr/local/hbase1.2/zookeeper</value>
  </property>
</configuration>
```

运行HBase

保证HDFS已经启动了：

```
[root@node3 hbase1.2]# jps
20435 NameNode
20583 DataNode
21239 Jps
20778 SecondaryNameNode
```

然后启动HBase：

```
[root@node3 hbase1.2]# start-hbase.sh
starting master, logging to /usr/local/hbase1.2/logs/hbase-root-master-n
ode3.out
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option PermSize=128m
; support was removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=1
28m; support was removed in 8.0
```

可以看到已经出现了HMater进程。

```
[root@node3 hbase1.2]# jps
21488 HMaster
21762 Jps
20435 NameNode
20583 DataNode
20778 SecondaryNameNode
```

然后连接到HBase，输入hbase shell即可：

```
[root@node3 hbase1.2]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hbase1.2/lib/slf4j-log4j12-
1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop/share/hadoop/common/
lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an expl
anation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0>
```

为了检验是否成功安装，尝试一些命令：

创建一个表格

```
hbase(main):001:0> create 'test', 'cf'
0 row(s) in 1.6240 seconds

=> Hbase::Table - test
```

添加数据：

```
hbase(main):004:0> scan 'test'
ROW                    COLUMN+CELL
 row1                  column=cf:a, timestamp=1605693505784, value=value1
1 row(s) in 0.0950 seconds
```

查看表格：

```
hbase(main):004:0> scan 'test'
ROW                    COLUMN+CELL
 row1                  column=cf:a, timestamp=1605693505784, value=value1
1 row(s) in 0.0950 seconds
```

操作正常执行，单机模式安装成功！

HBase伪分布式安装

配置文件

编辑hbase-site.xml配置。首先，添加以下属性。它指示HBase在分布式模式下运行，每个守护程序一个JVM实例。

```
  <property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
  </property>
```

接下来，hbase.rootdir使用hdfs://////URI语法将本地文件系统更改为HDFS实例的地址。注意端口号的设置要与HDFS的core-site.xml中fs.defaultFS相同的端口号相同。

```
 <property>
   <name>hbase.rootdir</name>
   <value>hdfs://localhost:9000/hbase</value>
 </property>
```

运行HBase

启动hbase：

```
[root@node3 conf]# start-hbase.sh
localhost: starting zookeeper, logging to /usr/local/hbase1.2/bin/../log
s/hbase-root-zookeeper-node3.out
starting master, logging to /usr/local/hbase1.2/logs/hbase-root-master-n
ode3.out
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option PermSize=128m
; support was removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=1
28m; support was removed in 8.0
starting regionserver, logging to /usr/local/hbase1.2/logs/hbase-root-1-
regionserver-node3.out
```

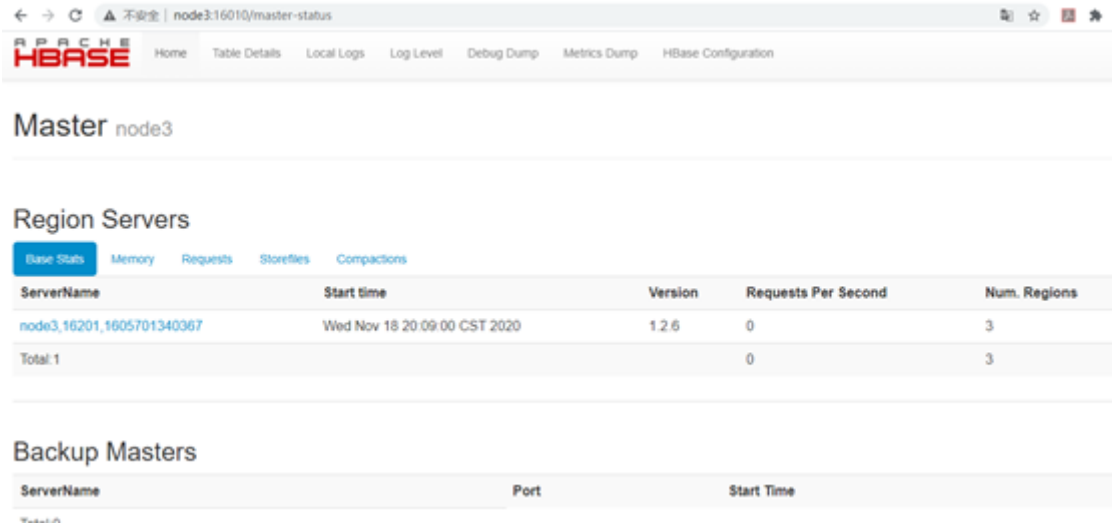
查看进程，HMaster，HRegionServer，zookeeper守护进程HQuorumPeer均已启动：

```
[root@node3 conf]# jps
544 DataNode
1153 HQuorumPeer
1220 HMaster
1353 HRegionServer
1451 Jps
747 SecondaryNameNode
399 NameNode
```

查看HDFS文件系统，出现了hbase的相关文件：

```
[root@node3 hbase1.2]# hdfs dfs -ls /hbase
Found 7 items
drwxr-xr-x   - root supergroup          0 2020-11-18 18:56 /hbase/.tmp
drwxr-xr-x   - root supergroup          0 2020-11-18 18:56 /hbase/Master
ProcWALs
drwxr-xr-x   - root supergroup          0 2020-11-18 18:56 /hbase/WALs
drwxr-xr-x   - root supergroup          0 2020-11-18 18:56 /hbase/data
-rw-r--r--   1 root supergroup         42 2020-11-18 18:56 /hbase/hbase.
id
-rw-r--r--   1 root supergroup          7 2020-11-18 18:56 /hbase/hbase.
version
drwxr-xr-x   - root supergroup          0 2020-11-18 18:56 /hbase/oldWAL
s
```

访问16010端口：



在hbase shell中尝试运行命令：

```
hbase(main):005:0> create 'test', 'cf'
0 row(s) in 1.3330 seconds

=> Hbase::Table - test
hbase(main):006:0> list 'test'
TABLE
test
1 row(s) in 0.0230 seconds

=> ["test"]
```

可以正常操作，HBase伪分布式搭建成功！

在HBase Shell中完成任务

完整的命令放在了HBaseShellCode.txt文件中。

(1) 创建讲义中的students表；

命令：
create 'students','ID','Description','Courses','Home'
put 'students','001','Description:Name','Li Lei'
.....

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.3360 seconds

=> []
hbase(main):002:0> create 'students','ID','Description','Courses','Home'
0 row(s) in 1.4120 seconds

=> Hbase::Table - students
```

```
hbase(main):003:0> put 'students','001','Description:Name','Li Lei'
0 row(s) in 0.1200 seconds

hbase(main):004:0> put 'students','001','Description:Height','176'
0 row(s) in 0.0430 seconds

hbase(main):005:0> put 'students','001','Courses:Chinese','80'
0 row(s) in 0.0170 seconds

hbase(main):006:0> put 'students','001','Courses:Math','90'
0 row(s) in 0.0340 seconds
```

(2) 扫描创建后的students表；

命令：scan 'students'

```
hbase(main):022:0> scan 'students'
ROW                                    COLUMN+CELL
 001                                   column=Courses:Chinese, timestamp=1605704928522, value=80
 001                                   column=Courses:Math, timestamp=1605704935463, value=90
 001                                   column=Courses:Physics, timestamp=1605704941978, value=95
 001                                   column=Description:Height, timestamp=1605704922225, value=176
 001                                   column=Description:Name, timestamp=1605704911860, value=Li Lei
 001                                   column=Home:Province, timestamp=1605704944983, value=Zhejiang
 002                                   column=Courses:Chinese, timestamp=1605704967802, value=88
 002                                   column=Courses:Math, timestamp=1605704967838, value=77
 002                                   column=Courses:Physics, timestamp=1605704967885, value=66
 002                                   column=Description:Height, timestamp=1605704967766, value=183
 002                                   column=Description:Name, timestamp=1605704967722, value=Han Meimei
 002                                   column=Home:Province, timestamp=1605704969664, value=Beijing
 003                                   column=Courses:Chinese, timestamp=1605704977293, value=90
 003                                   column=Courses:Math, timestamp=1605704977323, value=90
 003                                   column=Courses:Physics, timestamp=1605704977355, value=95
 003                                   column=Description:Height, timestamp=1605704977243, value=162
 003                                   column=Description:Name, timestamp=1605704977210, value=Xiao Ming
 003                                   column=Home:Province, timestamp=1605704978017, value=Shanghai
```

(3) 查询学生来自的省；

命令：scan 'students',{COLUMNS => 'Home:Province'}

```
hbase(main):025:0> scan 'students',{COLUMNS => 'Home:Province'}
ROW                                    COLUMN+CELL
 001                                   column=Home:Province, timestamp=1605704944983, value=Zhejiang
 002                                   column=Home:Province, timestamp=1605704969664, value=Beijing
 003                                   column=Home:Province, timestamp=1605704978017, value=Shanghai
3 row(s) in 0.1160 seconds
```

(4) 增加新的列Courses:English，并添加数据；

命令：put 'students','001','Courses:English','95'

.....

```
hbase(main):026:0> put 'students','001','Courses:English','95'
0 row(s) in 0.0220 seconds

hbase(main):027:0> put 'students','002','Courses:English','95'
0 row(s) in 0.0240 seconds

hbase(main):028:0> put 'students','003','Courses:English','95'
0 row(s) in 0.0170 seconds
```

```
hbase(main):030:0> scan 'students',{COLUMNS => 'Courses:English'}
ROW                                    COLUMN+CELL
 001                                   column=Courses:English, timestamp=1605705263981, value=95
 002                                   column=Courses:English, timestamp=1605705264020, value=95
 003                                   column=Courses:English, timestamp=1605705265367, value=95
3 row(s) in 0.0740 seconds
```

(5) 增加新的列族Contact和新列Contact:Email，并添加数据；

命令：

alter 'students', NAME => 'Contact', VERSIONS => 5

put 'students','001','Contact:Email','lilei@qq.com'

.....

```
hbase(main):055:0> alter 'students', NAME => 'Contact', VERSIONS => 5
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 3.0010 seconds
```

```
hbase(main):056:0> put 'students','001','Contact:Email','lilei@qq.com'
0 row(s) in 0.0230 seconds

hbase(main):057:0> put 'students','002','Contact:Email','hanmeimei@qq.com'
0 row(s) in 0.0180 seconds

hbase(main):058:0> put 'students','003','Contact:Email','xiaoming@qq.com'
0 row(s) in 0.0110 seconds
```

```
hbase(main):059:0> scan 'students',{COLUMNS => 'Contact:Email'}
ROW                                    COLUMN+CELL
 001                                   column=Contact:Email, timestamp=1605856839006, value=lilei@qq.com
 002                                   column=Contact:Email, timestamp=1605856839044, value=hanmeimei@qq.com
 003                                   column=Contact:Email, timestamp=1605856839851, value=xiaoming@qq.com
3 row(s) in 0.0270 seconds
```

(6) 删除students表。

命令：
disable 'students'
drop 'students'

```
hbase(main):038:0> disable 'students'
0 row(s) in 2.2910 seconds

hbase(main):039:0> drop 'students'
0 row(s) in 1.2850 seconds

hbase(main):040:0> list
TABLE
0 row(s) in 0.0090 seconds

=> []
```

编写Java程序完成任务

对应的Java源程序为HbOperation.java，程序针对实验要求的每一项任务都打印出了操作结果，而且在代码中编写了注释。下面是一些代码和输出结果的说明：

创建连接

实现方法：`connect()`
主要任务是配置configuration对象，设置数据库所在的主机名称等，然后利用工厂方法创建connection类，获得Admin。

```
//创建一个configuration
Configuration conf = HBaseConfiguration.create();
//目标主机名设为mastername
conf.set("hbase.zookeeper.quorum", mastername);
//连接数据库
conn = ConnectionFactory.createConnection(conf);
//获得admin
admin=conn.getAdmin();
```

(1) 创建讲义中的students表；

实现方法：`createTable(String tablename, String[] families)`方法定义表的Schema。创建`HTableDescriptor`对象储存表的信息，然后添加需要创建的列族，最后由`Admin`对象提交。

```
//先创建descriptor
HTableDescriptor newtable = new HTableDescriptor(TableName.valueOf(tablename));
//添加列族名
for (String family:families) {
    newtable.addFamily(new HColumnDescriptor(family));
}
//提交创建
admin.createTable(newtable);
```

`putData(String tablename, String rowKey, String family, String qualifier, String value)`方法将单条数据插入到指定Cell中。首先从connection对象获取table，创建`Put`储存插入内容，然后提交到table。

```
//获取table
Table table=conn.getTable(TableName.valueOf(tablename));
//设置row key
Put put = new Put(Bytes.toBytes(rowKey));
//将value插入到family:qualifier中
put.addColumn(Bytes.toBytes(family), Bytes.toBytes(qualifier),
Bytes.toBytes(value));
table.put(put);
```

最后由`putAlldata(String tablename)`方法完成全部信息的添加。
程序输出：在创建表之前，程序会打印数据库中所有的表，创建表后查询表的所有列族，以检查创建是否成功，每条信息插入后都会打印插入状态

```
Connect successfully!
All tables:

-------(1) Create table: students-------
Create table students successfully!
Column Families of students: Courses, Description, Home, ID
One record inserted.
One record inserted.
One record inserted.
```

(2) 扫描创建后的students表；

实现方法：`scanTable(String tablename)`
从table获取scanner，然后读取每一个row key的内容即可。

```
//获取table
Table table=conn.getTable(TableName.valueOf(tablename));
ResultScanner resscan=table.getScanner(new Scan());
```

```
//遍历每一行
for (Result result:resscan) {
    //获取row key
    String row = new String(result.getRow());
    //将cell的内容放到list中
    List<Cell> cells = result.listCells();
    //打印每个列族列属性和对应value
    for (Cell c:cells) {
        ...
    }
}
```

程序输出：

```
-------(2) Scan table: students-------
ROW     COLUMN+CELL
001     Courses:Chinese, value=80
001     Courses:Math, value=90
001     Courses:Physics, value=95
001     Description:Height, value=176
001     Description:Name, value=Li Lei
001     Home:Province, value=Zhejiang
002     Courses:Chinese, value=88
002     Courses:Math, value=77
002     Courses:Physics, value=66
002     Description:Height, value=183
002     Description:Name, value=Han Meimei
002     Home:Province, value=Beijing
003     Courses:Chinese, value=90
003     Courses:Math, value=90
003     Courses:Physics, value=95
003     Description:Height, value=162
003     Description:Name, value=Li Lei
003     Home:Province, value=Shanghai
Connection closed.
```

（3）查询学生来自的省；

实现方法：`scanByColumn(String tablename, String family, String qualifier)`
查询某一列，大致和扫描整个表差不多，只需要在`getScanner()`方法中设置列族和列名即可。

```
ResultScanner resscan=table.getScanner(family.getBytes(), qualifier.getBytes());
```

程序输出：

```
-------(3) Query Home:Province-------
ROW     COLUMN+CELL
001     Home:Province, value=Zhejiang
002     Home:Province, value=Beijing
003     Home:Province, value=Shanghai
```

(4) 增加新的列Courses:English，并添加数据；

实现方法：`putEnglish(String tablename)`
由于HBase不需要预先定义列，所以直接执行插入就可以了。

```
putData(tablename, "001", "Courses", "English", "95");
......
```

程序输出：添加完成后，程序会查询新增的列，以检查是否添加成功

```
-------(4) Add new column, Courses:English-------
One record inserted.
One record inserted.
One record inserted.
Data added:
ROW     COLUMN+CELL
001     Courses:English, value=95
002     Courses:English, value=95
003     Courses:English, value=95
```

(5) 增加新的列族Contact和新列Contact:Email，并添加数据；

实现方法：`addFamily(String tablename, String family)`添夹新的列族，从table获取表的定义信息，然后添加一条新的列族定义，最后将修改提交到`Admin`

```
//获得原来表的定义信息
HTableDescriptor tableDescriptor =
admin.getTableDescriptor(TableName.valueOf(tablename));
//构造新的列族定义
HColumnDescriptor nColumnDescriptor = new HColumnDescriptor(family);
//将列族添加到表的定义中
tableDescriptor.addFamily(nColumnDescriptor);
//将修改后的表的定义提交到admin
admin.modifyTable(TableName.valueOf(tablename), tableDescriptor);
```

然后添加数据，操作与之前相似，不再赘述。
程序输出：添加完数据后查询新加的列，确保插入成功。

```
-------(5) Add new column family: Contact,new column Contact:Email-------
Add family Contact successfully!
Column Families of students: Contact, Courses, Description, Home, ID
One record inserted.
One record inserted.
One record inserted.
Data added:
ROW     COLUMN+CELL
001     Contact:Email, value=lilei@qq.com
002     Contact:Email, value=hanmeimei@qq.com
003     Contact:Email, value=xiaoming@qq.com
```

(6) 删除students表；

实现方法：`dropTable(String tablename)`
和shell中的操作类似，通过`Admin`先disable，然后再delete即可。

```
admin.disableTable(TableName.valueOf(tablename));
admin.deleteTable(TableName.valueOf(tablename));
```

程序输出：删除前后查询所有表，执行删除后表名不存在

```
-------(6) Drop table: students-------
All tables:
students
Drop table students successfully!
All tables:
Connection closed
```

程序完整输出

```
Connect successfully!
All tables:

-------(1) Create table: students-------
Create table students successfully!
Column Families of students: Courses, Description, Home, ID
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.
One record inserted.

-------(2) Scan table: students-------
ROW     COLUMN+CELL
001     Courses:Chinese, value=80
001     Courses:Math, value=90
001     Courses:Physics, value=95
001     Description:Height, value=176
001     Description:Name, value=Li Lei
001     Home:Province, value=Zhejiang
002     Courses:Chinese, value=88
002     Courses:Math, value=77
002     Courses:Physics, value=66
002     Description:Height, value=183
002     Description:Name, value=Han Meimei
```

```
002     Home:Province, value=Beijing
003     Courses:Chinese, value=90
003     Courses:Math, value=90
003     Courses:Physics, value=95
003     Description:Height, value=162
003     Description:Name, value=Li Lei
003     Home:Province, value=Shanghai

-------(3) Query Home:Province-------
ROW     COLUMN+CELL
001     Home:Province, value=Zhejiang
002     Home:Province, value=Beijing
003     Home:Province, value=Shanghai

-------(4) Add new column, Courses:English-------
One record inserted.
One record inserted.
One record inserted.
Data added:
ROW     COLUMN+CELL
001     Courses:English, value=95
002     Courses:English, value=95
003     Courses:English, value=95

-------(5) Add new column family: Contact,new column Contact:Email-------
Add family Contact successfully!
Column Families of students: Contact, Courses, Description, Home, ID
One record inserted.
One record inserted.
One record inserted.
Data added:
ROW     COLUMN+CELL
001     Contact:Email, value=lilei@qq.com
002     Contact:Email, value=hanmeimei@qq.com
003     Contact:Email, value=xiaoming@qq.com

-------(6) Drop table: students-------
All tables:
students
Drop table students successfully!
All tables:
Connection closed.
```

HBase集群模式安装

节点分配

node1: HMaster, HRegionServer
node2: HRegionServer

配置文件

node1和node2的文件配置完全相同，在一个节点上配置好之后复制到另一个节点上即可。
首先还是编辑conf/hbase-env.sh，设置JAVA_HOME路径：

```
# The java implementation to use.  Java 1.7+ required.
export JAVA_HOME=/usr/local/jdk1.8
```

修改conf/regionservers，将localhost改为需要运行regionserver的主机名，这里我想让两个主机都运行regionserver：

```
node1
node2
~
```

然后是conf/hbase-site.xml：
分布式选项和zookeeper文件的路径还是和伪分布式相同

```
<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/usr/local/hbase1.2/zookeeper</value>
</property>
<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>
```

而hbase.rootdir的路径需要改为node1:9000，这和HDFS的core-site.xml中fs.defaultFS的配置逻辑相同。

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://node1:9000/hbase</value>
</property>
```

最后需要指定zookeeper运行的节点，这里两个节点都需要写上

```
<property>
  <name>hbase.zookeeper.quorum</name>
  <value>node1,node2</value>
</property>
```

在node1上配置好文件之后复制到node2上即可。

运行HBase

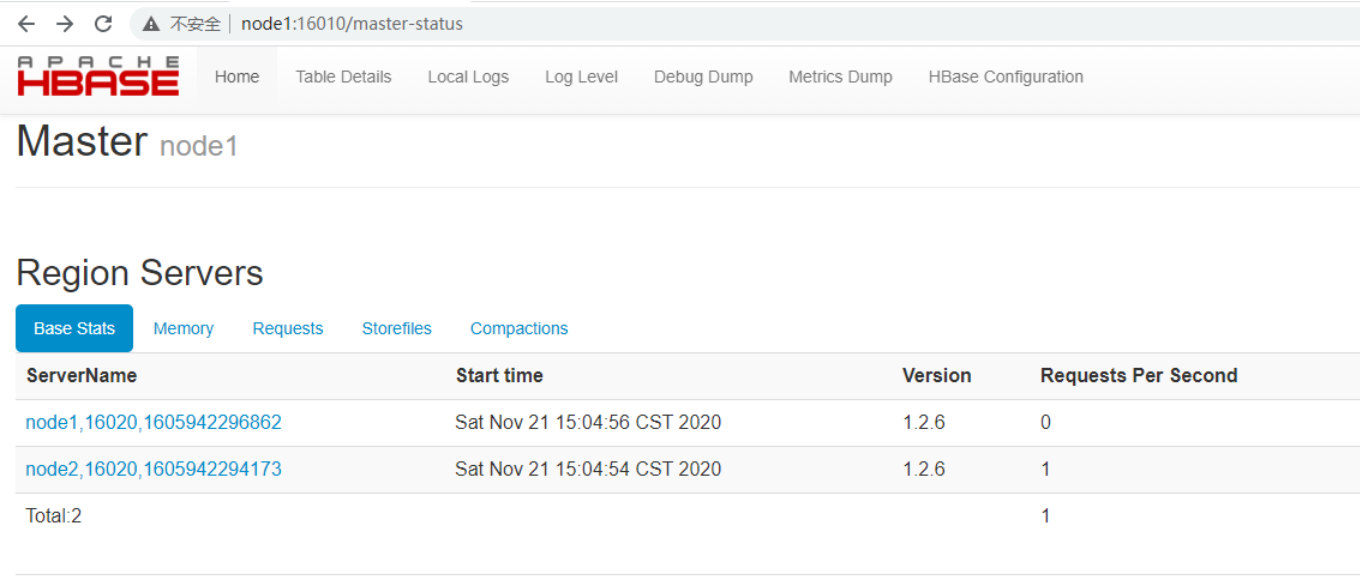
确保HDFS已经运行，然后输入start-hbase.sh，分别查看两个节点上运行的进程

```
[root@node1 hbase1.2]# jps
6068 NameNode
8376 Jps
6203 DataNode
16205 JobHistoryServer
8317 HRegionServer
8189 HMaster
6398 SecondaryNameNode
8127 HQuorumPeer
```

```
[root@node2 hbase1.2]# jps
25904 DataNode
26470 HQuorumPeer
26551 HRegionServer
```

两个节点上的进程都正常运行。

访问node1的16010端口



可以看见两个regionserver。

在hbase shell中输入命令：

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.3680 seconds

=> []
hbase(main):002:0> create 'students','ID','Description','Courses','Home'
0 row(s) in 68.3970 seconds

=> Hbase::Table - students
hbase(main):003:0> status
1 active master, 0 backup masters, 2 servers, 0 dead, 1.5000 average load
```

可以看到有一个master和两个server，集群模式可以运行。

问题和解决方案

(1) 搭建伪分布式时HBase shell报错：Can't get master address from Zookeeper;

原因：hbase-site.xml文件中的rootdir端口号与core-site.xml中fs.defaultFS的端口号不一致，官方指导文档的示例里面用的是8020，很具有误导性，我的HDFS端口是9000，因此这里改成9000即可。

```
<property>
  <name>hbase.rootdir</name>
  <value>hdfs://localhost:9000/hbase</value>
</property>
```

(2) 遗留问题：集群模式时node2上没有分配region

Region Servers

Base Stats | Memory | Requests | Storefiles | Compactions

ServerName	Start time	Version	Requests Per Second	Num. Regions
node1,16020,1605942296862	Sat Nov 21 15:04:56 CST 2020	1.2.6	0	2
node2,16020,1605942294173	Sat Nov 21 15:04:54 CST 2020	1.2.6	1	0
Total:2			1	2

我查了log，貌似又是node2尝试用内网ip访问node1的问题，但是HBase好像没有提供按照本地hosts文件解析域名的选项，导致node2其实是一个假节点，并没有多大用。hbase shell虽然可以成功执行命令，但是创建一个表非常非常慢，竟然需要68秒。

```
hbase(main):001:0> list
TABLE
0 row(s) in 0.3680 seconds

=> []
hbase(main):002:0> create 'students','ID','Description','Courses','Home'
0 row(s) in 68.3970 seconds
```

想都不要想就知道肯定是一个节点在写入的时候一直没成功，等了很久，然后将任务换到了另一个节点上做。总之根本问题还是两台ECS内网不能互通，搞得我实在是心累。有时间的话打算搭个k8s，希望能让我不再碰到这些问题。