Hadoop集群搭建笔记

一. Hadoop

1、安装虚拟机

VMware

CentOS 镜像

安装Linux虚拟机：

1）安装VMware（可修改配置）

2）添加CentOS镜像（启动安装 ->配置网络）

网络配置：NAT模式

网络重启：service network restart

关闭系统：shutdown -h now

2、远程连接

Xshell5

Xftp5

3、在Linux上搭建Hadoop集群

下载软件

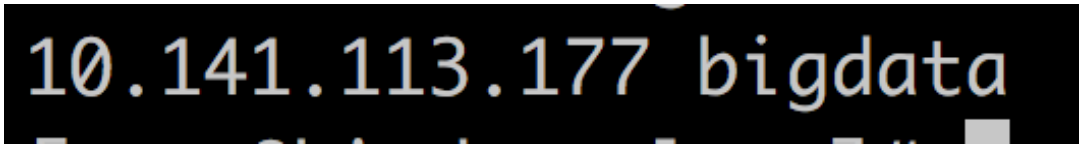
1. Jdk：http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

2. Hadoop包:http://apache.fayea.com/hadoop/common/hadoop-2.7.2/

步骤：

1、修改/etc/hosts

将内网ip映射到 bigdata上



2、配置JDK

安装：
$ rpm -ivh jdk-8u101-linux-x64.rpm

(linux下的java_home在 /usr/java/default下)
```
$ vi /etc/profile
```
JAVA_HOME (http://note.youdao.com/noteshare?id=bd547306634b4cbe9fba5c5b8239ba5d)
```
$ source /etc/profile
```

执行下面的命令，检查配置是否生效：
```
$ java -version
```

3、配置SSH（免密码登录）
```
$ ssh-keygen -t rsa
$ cat xxx.pub >> authorized_keys
$ chmod 644 authorized_keys
```
检验是否配置成功：
```
$ ssh IP/HOSTNAME
```

4、安装及配置Hadoop

```
$ tar zxf hadoop-2.7.2.tar.gz
$ cd /opt/hadoop-2.7.2/etc/hadoop/
```

关于配置文件的中文说明 http://m.blog.itpub.net/29800581/viewspace-2145076/

1) 修改core-site.xml

```
<property>
   <name>fs.default.name</name>
   <value>hdfs://bigdata:8082</value>
</property>
<property>
      <name>hadoop.tmp.dir</name>
      <value>/opt/hadoop-2.7.2/current/tmp</value>
</property>
<property>
      <name>fs.trash.interval</name>
      <value>4320</value>
</property>
```

2) 修改 hdfs-site.xml

```
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:/opt/hadoop-2.7.2/current/dfs/name</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>file:/opt/hadoop-2.7.2/current/data</value>
</property>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.webhdfs.enabled</name>
  <value>true</value>
</property>
<property>
  <name>dfs.permissions.superusergroup</name>
  <value>staff</value>
```

```
</property>
<property>
  <name>dfs.permissions.enabled</name>
  <value>false</value>
</property>
```

3) 修改 yarn-site.xml

```
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>bigdata</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
  <name>yarn.resourcemanager.address</name>
  <value>bigdata:18040</value>
</property>
<property>
  <name>yarn.resourcemanager.scheduler.address</name>
  <value>bigdata:18030</value>
</property>
<property>
  <name>yarn.resourcemanager.resource-tracker.address</name>
  <value>bigdata:18025</value>
</property><property>
  <name>yarn.resourcemanager.admin.address</name>
  <value>bigdata:18141</value>
```

```
</property>
<property>
  <name>yarn.resourcemanager.webapp.address</name>
  <value>bigdata:18088</value>
</property>
<property>
  <name>yarn.log-aggregation-enable</name>
  <value>true</value>
</property>
<property>
  <name>yarn.log-aggregation.retain-seconds</name>
  <value>86400</value>
</property>
<property>
  <name>yarn.log-aggregation.retain-check-interval-seconds</name>
  <value>86400</value>
</property>
<property>
  <name>yarn.nodemanager.remote-app-log-dir</name>
  <value>/tmp/logs</value>
</property>
<property>
  <name>yarn.nodemanager.remote-app-log-dir-suffix</name>
  <value>logs</value>
</property>
```

4) 修改 mapred-site.xml

```
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
<property>
   <name>mapreduce.jobtracker.http.address</name>
```

```
    <value>bigdata:50030</value>
</property>
<property>
    <name>mapreduce.jobhisotry.address</name>
    <value>bigdata:10020</value>
</property>
<property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>bigdata:19888</value>
</property>
<property>
    <name>mapreduce.jobhistory.done-dir</name>
    <value>/jobhistory/done</value>
</property>
<property>
    <name>mapreduce.intermediate-done-dir</name>
    <value>/jobhisotry/done_intermediate</value>
</property>
<property>
    <name>mapreduce.job.ubertask.enable</name>
    <value>true</value>
</property>
```

5) 修改 slaves

```
bigdata
```

6) 修改 hadoop-env.sh

```
JAVA_HOME=
```

7) 创建上面新增的文件目录

```
$ mkdir -p /opt/hadoop-2.7.5/current/tmp
$ mkdir -p /opt/hadoop-2.7.5/current/dfs/name
```

$ mkdir -p /opt/hadoop-2.7.5/current/data
$ /tmp/logs # yarn log所在位置

5、格式化HDFS

1)先配置 Hadoop环境变量, 在/etc/profile下添加hadoop的文件路径:
hadoop
export HADOOP_HOME=/Users/YourUserName/Documents/Dev/hadoop-2.7.3
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

2)执行下列操作
$ hdfs namenode -format

‘16/09/0403:07:30 INFO common.Storage: Storage directory /opt/hadoop-2.7.2/current/dfs/name has been successfully formatted.’

6、启动Hadoop集群
$/opt/hadoop-2.7.5/sbin/start-all.sh

如果mac出现:
ssh: connect to host localhost port 22: Connection refused
则:参照 ssh_prot 22 _Connection refused 解决
http://note.youdao.com/noteshare?id=3dad163cad5f4696120b8fde4a0aa45e

$ /opt/hadoop-2.7.5/sbin/stop-all.sh (**关闭集群**)

7、验证Hadoop集群

1） $ jps

2) 关闭防火墙 或者 在防火墙的规则中开放这些端口

hdfs http://bigdata:50070

yarn http://bigdata:18088

8. 开启job历史记录

$ /opt/hadoop-2.7.5/sbin/mr-jobhistory-daemon.sh start historyserver

关闭:

/opt/hadoop-2.7.5/sbin/mr-jobhistory-daemon.sh stop historyserver

二. 安装Scala

1.安装scala

$ rpm -ivh scala-2.11.8.rpm

(使用 yum remove scala 可以卸载已经安装好的scala)

2. 添加SCALA_HOME:

$ vim /etc/profile

然后添加:

export SCALA_HOME=/usr/share/scala

export PATH=$SCALA_HOME/bin:$PATH

保存后,

$ source /etc/profile

三. 安装Spark

1. 解压

$ tar -xvf spark-2.3.0-bin-hadoop2.7.tgz

2. 修改相应的配置文件:

（1）$ vim /etc/profie

#Spark enviroment

export SPARK_HOME=/opt/spark-2.3.0-bin-hadoop2.7

export PATH=$SPARK_HOME/bin:$PATH

（2）修改$SPARK_HOME/conf/spark-env.sh

$ cp spark-env.sh.template spark-env.sh

spark-env.sh 中增加如下配置:

export SPARK_LOCAL_IP=外网IP

export SPARK_MASTER_IP=10.141.113.177

export SPARK_MASTER_PORT=7077

export SPARK_MASTER_WEBUI_PORT=8080

export SPARK_CONF_DIR=/opt/spark-2.3.0-bin-hadoop2.7/conf

（3）修改$SPARK_HOME/conf/slaves

$cp slaves.template slaves

配置内容如下

bigdata

3. 新建spark-defaults.conf文件

以spark为我们创建好的模板创建一个slaves文件，命令是：

```
cp   spark-defaults.conf.template   spark-defaults.conf
```

编辑spark-defaults.conf文件，在里面新增配置：

spark.master spark://master:7077

spark.eventLog.enabled true

spark.eventLog.dir hdfs://master:9000/directory

spark.serializer org.apache.spark.serializer.KryoSerializer

spark.driver.memory 700M

spark.executor.extraJavaOptions -XX:+PrintGCDetails -Dkey=value-

Dnumbers="one two three"

6.2 在HDFS上创建目录

因为上面的配置中让spark将eventLog存到HDFS的directory目录下，所以需要执行hadoop命令，在HDFS上创建directory目录，创建目录命令是：

[plain] view plain copy

```
1. $HADOOP_HOME/bin/hadoop   fs  -mkdir  -p   /directory
```

授权命令是：

[plain] view plain copy

```
1. $HADOOP_HOME/bin/hadoop   fs  -chmod  777  /directory
```

3. 启动

1)使用shell

$ spark-shell

出现 👇

2)启动后看UI

$/opt/spark-2.3.0-bin-hadoop2.7/sbin/start-all.sh # 启动

然后查看 spark ui , http://bigdata:8080/

$/opt/spark-2.3.0-bin-hadoop2.7/sbin/stop-all.sh #关闭

4. 可能出现的问题:

(如果出现问题了可以去 /logs/ 下面看对应的log, 搜索 ERROR找到出错的地方, 然后去谷歌找解决办法)

(1)spark报错 Spark hostname resolving to loopback address warning in spark worker logs,解决办法 :在spark-env.sh中添加 SPARK_LOCAL_IP=内网ip.

下面是详细的说明:

Summary

Some nodes may resolve the hostname to the loopback address, while this is ok, it may generate a warning in the spark worker logs

Symptoms

The following error may be seen in the spark worker logs

WARN [Thread-378] 2015-06-11 13:41:39,712 ExternalLogger.java (line 73) SparkWorker: Your hostname, myhost1.somedomain.com resolves to a loopback address: 127.0.0.1; using 10.1.2.1 instead (on interface bond1)

WARN [Thread-378] 2015-06-11 13:41:39,714 ExternalLogger.java (line 73) SparkWorker: Set SPARK_LOCAL_IP if you need to bind to another address

Cause

The hostname resolved to the loopback address.

Solution

As the error suggests, add the SPARK_LOCAL_IP in the /etc/dse/spark/spark-env.sh file
`SPARK_LOCAL_IP="<IP address>"`

(2) $ spark-shell报错 WARN NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
解决办法:
You will also have to edit this `/etc/profile` line:
`export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native/:$LD_LIBRARY_PATH`
(3)错误：java.io.IOException: Incompatible clusterIDs 时常出现在namenode重新格式化之后

2014-04-29 14:32:53,877 FATAL org.apache.hadoop.hdfs.server.datanode.DataNode: Initialization failed for block pool Block pool BP-1480406410-192.168.1.181-1398701121586 (storage id DS-167510828-192.168.1.191-50010-1398750515421) service to hadoop-master/192.168.1.181:8082"
java.io.IOException: Incompatible clusterIDs in /data/dfs/data: namenode clusterID = CID-d1448b9e-da0f-499e-b1d4-78cb18ecdebb; datanode clusterID = CID-ff0faa40-2940-4838-b321-98272eb0dee3! U8 t) L- F(@0 ~' H0 N9 I
at org.apache.hadoop.hdfs.server.datanode.DataStorage.doTransition(DataStorage.java:391)5 ~"
at org.apache.hadoop.hdfs.server.datanode.DataStorage.recoverTransitionRead(DataStorage.java:191)8 {* e. t; f7 ?# I8 I: \- v
at org.apache.hadoop.hdfs.server.datanode.DataStorage.recoverTransitionRead(DataStorage.java:219)
at org.apache.hadoop.hdfs.server.datanode.DataNode.initStorage(DataNode.java:837)
at

org.apache.hadoop.hdfs.server.datanode.DataNode.initBlockPool(DataNode.java:808)9
at org.apache.hadoop.hdfs.server.datanode.BPOfferService.verifyAndSetNamespaceInfo(BPOfferService.java:280)
at org.apache.hadoop.hdfs.server.datanode.BPServiceActor.connectToNNAndHandshake(BPServiceActor.java:222)
at org.apache.hadoop.hdfs.server.datanode.BPServiceActor.run(BPServiceActor.java:664)* j) }
at java.lang.Thread.run(Thread.java:722)
2014-04-29 14:32:53,885 WARN org.apache.hadoop.hdfs.server.datanode.DataNode: Ending block pool service for: Block pool BP-1480406410-192.168.1.181-1398701121586 (storage id DS-167510828-192.168.1.191-50010-1398750515421) service to hadoop-master/192.168.1.181:8082 V9 G- G3 f* L
2014-04-29 14:32:53,889 INFO org.apache.hadoop.hdfs.server.datanode.DataNode: Removed Block pool BP-1480406410-192.168.1.181-1398701121586 (storage id DS-167510828-192.168.1.191-50010-1398750515421)
2014-04-29 14:32:55,897 WARN org.apache.hadoop.hdfs.server.datanode.DataNode: Exiting Datanode

原因：每次namenode format会重新创建一个namenodeId,而data目录包含了上次format时的id,namenode format清空了namenode下的数据,但是没有清空datanode下的数据,导致启动时失败,所要做的就是每次fotmat前,清空data下的所有目录.

解决办法：停掉集群，删除问题节点的data目录下的所有内容。即hdfs-site.xml文件中配置的dfs.data.dir目录。重新格式化namenode。

四. 安装Sqoop

1.解压
$ tar zxf sqoop-1.4.7.bin__hadoop-2.6.0.tar.gz
$ mv sqoop-1.4.7.bin__hadoop-2.6.0 sqoop-1.4.7

2.修改/etc/profile, 增加如下内容
export SQOOP_HOME=/opt/sqoop-1.4.7
export PATH=$PATH:$SQOOP_HOME/bin

3. $ source /stc/profile

4. 下载mysql-connector-java包,解压后将bin.jar导入到sqoop的lib中
$ cd /opt/mysql-connector-java-5.1.46
$ cp mysql-connector-java-5.1.46-bin.jar /opt/sqoop-1.4.7/lib

5.# 进入conf下
 $ cp sqoop-env-template.sh sqoop-env.sh
 $ cp sqoop-site-template.xml sqoop-site.xml

6. 添加haoop所在路径
$ echo $HADOOP_HOME #将该输出结果写到下面

$ vim sqoop-env.sh
添加
 export HADOOP_COMMON_HOME=
export HADOOP_MAPRED_HOME=

7.问题
(1) 执行sqoop job时遇到Sqoop import exception
java.lang.NoClassDefFoundError: org/json/JSONObject
解决办法如下:
downloaded java-json.jar file from location
http://www.java2s.com/Code/Jar/j/Downloadjavajsonjar.htm
stored this jar file at location /usr/lib/sqoop/lib/java-json.jar

五. 安装Hive

1.解压

2.修改/etc/profile, 增加如下内容

export HIVE_HOME=/opt/hive-2.3.2

export PATH=$PATH:$HIVE_HOME/bin

3.添加haoop所在路径

$ cp hive-env.sh.template hive-env.sh

$ echo $HADOOP_HOME #将该输出结果写到下面

$ vim hive-env.sh

添加

export HADOOP_HOME=

4.

$ cp hive-default.xml.template hive-site.xml

修改配置, 并将拷到$HIVE_HOME/lib/下

5. 开启服务

hive --service metastore

Hive的配置和启动

7.1 执行命令创建HDFS目录

hive的文件存储在hadoop提供的HDFS分布式文件系统里，需要调用hadoop命令，在hdfs上创建几个目录。

执行创建命令：

[plain] view plain copy

```
1. $HADOOP_HOME/bin/hadoop   fs   -mkdir  -p
   /user/hive/warehouse
```

给刚才新建的目录赋予读写权限，执行命令：

[plain] view plain copy

```
1. $HADOOP_HOME/bin/hadoop  fs   -chmod   777
   /user/hive/warehouse
```

执行创建命令：

[plain] view plain copy

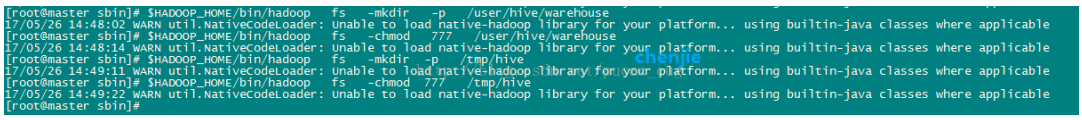
```
1. $HADOOP_HOME/bin/hadoop   fs   -mkdir -p   /tmp/hive
```

执行授权命令：

[plain] view plain copy

```
1. $HADOOP_HOME/bin/hadoop  fs   -chmod  777   /tmp/hive
```

如图：



Center

7.2 对conf目录下的配置文件进行配置

对/opt/hive/apache-hive-2.1.1-bin/conf目录下的一系列文件做配置，这些配置很关键。

7.2.1 新建hive-env.sh文件并进行修改

进入到/opt/hive/apache-hive-2.1.1-bin/conf目录，命令是：

[plain] view plain copy

```
1. cd   /opt/hive/apache-hive-2.1.1-bin/conf
```

将hive-env.sh.template文件复制一份，并且改名为hive-env.sh，命令是：

[plain] view plain copy

```
1. cp    hive-env.sh.template    hive-env.sh
```

打开hive-env.sh配置并且添加以下内容：

[plain] view plain copy

```
1. export  HADOOP_HOME=/opt/hadoop/hadoop-2.8.0
2. export  HIVE_CONF_DIR=/opt/hive/apache-hive-2.1.1-bin/conf
3. export  HIVE_AUX_JARS_PATH=/opt/hive/apache-hive-2.1.1-bin/lib
```

7.2.2 对hive-site.xml文件进行配置

首先要创建hive-site.xml文件

进入到/opt/hive/apache-hive-2.1.1-bin/conf目录，命令是：

[plain] view plain copy

```
1. cd  /opt/hive/apache-hive-2.1.1-bin/conf
```

将hive-default.xml.template文件复制一份，并且改名为hive-site.xml，命令是：

[plain] view plain copy

```
1. cp  hive-default.xml.template  hive-site.xml
```

7.2.2.1 目录相关的配置

首先在master机器上上创建临时目录/opt/hive/tmp

将hive-site.xml文件中的所有${system:java.io.tmpdir}替换为/opt/hive/tmp

将hive-site.xml文件中的所有${system:user.name}都替换为root

7.2.2.2 MySQL数据库相关的配置

搜索javax.jdo.option.ConnectionURL，将该name对应的value修改为MySQL的地址，例如我修改后是：

[plain] view plain copy

```
1. <name>javax.jdo.option.ConnectionURL</name>
2. <value>jdbc:mysql://192.168.27.138:3306/hive?
createDatabaseIfNotExist=true</value>
```

搜索javax.jdo.option.ConnectionDriverName，将该name对应的value修改为MySQL驱动类路径：

[plain] view plain copy

```
1. <name>javax.jdo.option.ConnectionDriverName</name>
2. <value>com.mysql.jdbc.Driver</value>
```

搜索javax.jdo.option.ConnectionUserName，将对应的value修改为MySQL数据库登录名：

[plain] view plain copy

```
1. <name>javax.jdo.option.ConnectionUserName</name>
2. <value>root</value>
```

搜索javax.jdo.option.ConnectionPassword，将对应的value修改为MySQL数据库的登录密码：

[plain] view plain copy

```
1. <name>javax.jdo.option.ConnectionPassword</name>
2. <value>cj</value>
```

搜索hive.metastore.schema.verification，将对应的value修改为false：

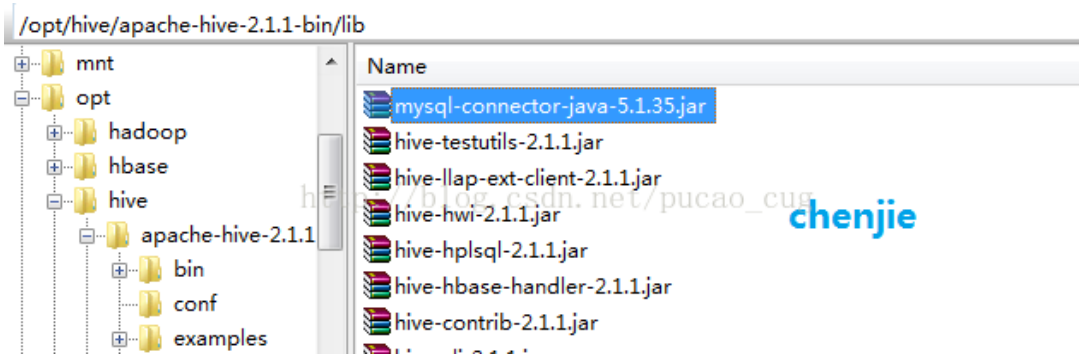
[plain] view plain copy

```
1. <name>hive.metastore.schema.verification</name>
2. <value>false</value>
```

7.3 将MySQL驱动包上载到lib目录

将MySQL驱动包上载到Hive的lib目录下，例如我是上载到/opt/hive/apache-hive-2.1.1-bin/lib目录下。

如图：

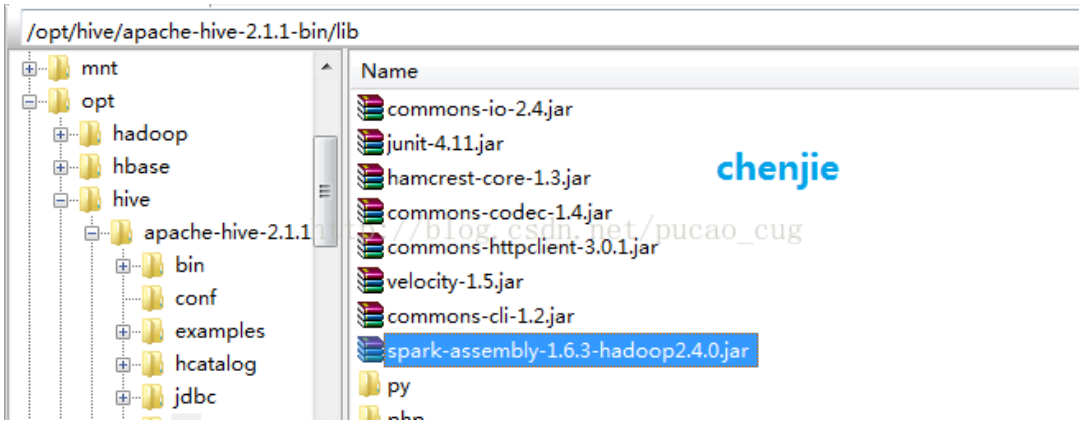


Center

7.4 将Spark下的某些jar包拷贝到hive目录下

在master 机器上，将/opt/spark/spark-1.6.3-bin-hadoop2.4-without-hive/lib目录下的spark-assembly-1.6.3-hadoop2.4.0.jar包拷贝到/opt/hive/apache-hive-2.1.1-bin/lib目录下。

如图：



Center

7.5 对hive所要连接的数据库做初始化

进入到hive的bin目录执行命令：

[plain] view plain copy

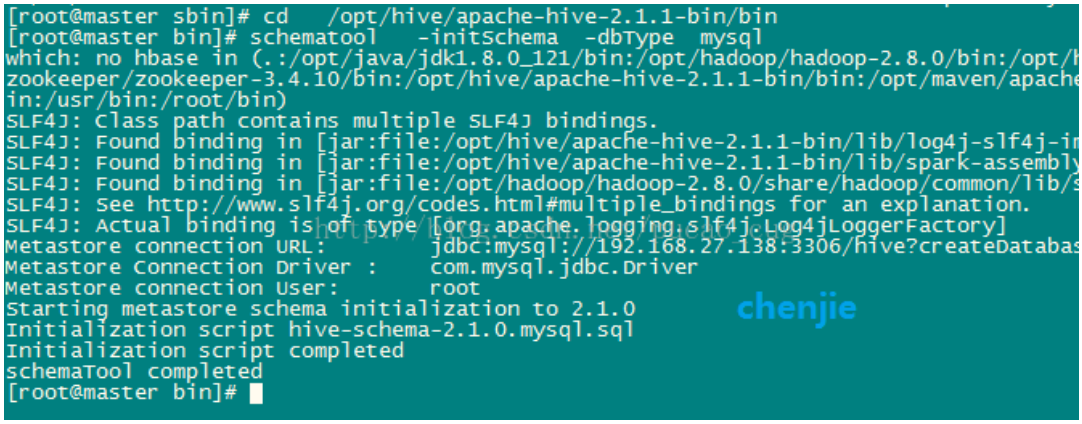
```
1. cd  /opt/hive/apache-hive-2.1.1-bin/bin
```

对数据库进行初始化，执行命令：

[plain] view plain copy

```
1. schematool  -initSchema  -dbType  mysql
```

如图：



Center

完整输出是：

[plain] view plain copy

```
[root@mastersbin]# cd  /opt/hive/apache-hive-2.1.1-bin/bin
[root@master bin]# schematool   -initSchema -dbType  mysql
which: no hbase
in(.:/opt/java/jdk1.8.0_121/bin:/opt/hadoop/hadoop-
2.8.0/bin:/opt/hadoop/hadoop-2.8.0/sbin:/opt/spark/spark-1.6.3-
bin-hadoop2.4-without-hive/bin:/opt/zookeeper/zookeeper-
3.4.10/bin:/opt/hive/apache-hive-2.1.1-bin/bin:/opt/maven/apache-
maven-3.3.9/bin:/opt/scala/scala-
2.11.8/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/roo
t/bin)
SLF4J: Class path contains multiple SLF4Jbindings.
SLF4J: Found binding in[jar:file:/opt/hive/apache-hive-2.1.1-
bin/lib/log4j-slf4j-impl-
2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in[jar:file:/opt/hive/apache-hive-2.1.1-
bin/lib/spark-assembly-1.6.3-
hadoop2.4.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in[jar:file:/opt/hadoop/hadoop-
2.8.0/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Seehttp://www.slf4j.org/codes.html#multiple_bindings
for an explanation.
SLF4J: Actual binding is of
type[org.apache.logging.slf4j.Log4jLoggerFactory]
Metastore connection URL:
jdbc:mysql://192.168.27.138:3306/hive?
createDatabaseIfNotExist=true
Metastore Connection Driver :    com.mysql.jdbc.Driver
Metastore connection User:       root
Starting metastore schema initialization to2.1.0
Initialization scripthive-schema-2.1.0.mysql.sql
Initialization script completed
schemaTool completed
[root@master bin]#
```

7.6 启动hive命令行窗口

现在打开到hive的bin目录中，打开命令是：

[plain] view plain copy

```
cd   /opt/hive/apache-hive-2.1.1-bin/bin
```

执行hive脚本，也就是执行命令：

[plain] view plain copy

```
1. ./hive
```

如图：

Center

完整输出是：

[plain] view plain copy

```
1. [root@mastersbin]# cd  /opt/hive/apache-hive-2.1.1-bin/bin
2. [root@master bin]# schematool  -initSchema -dbType  mysql
3. which: no hbase
in(.:/opt/java/jdk1.8.0_121/bin:/opt/hadoop/hadoop-
2.8.0/bin:/opt/hadoop/hadoop-2.8.0/sbin:/opt/spark/spark-1.6.3-
bin-hadoop2.4-without-hive/bin:/opt/zookeeper/zookeeper-
3.4.10/bin:/opt/hive/apache-hive-2.1.1-bin/bin:/opt/maven/apache-
maven-3.3.9/bin:/opt/scala/scala-
2.11.8/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/roo
t/bin)
4. SLF4J: Class path contains multiple SLF4Jbindings.
5. SLF4J: Found binding in[jar:file:/opt/hive/apache-hive-2.1.1-
bin/lib/log4j-slf4j-impl-
2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
6. SLF4J: Found binding in[jar:file:/opt/hive/apache-hive-2.1.1-
bin/lib/spark-assembly-1.6.3-
hadoop2.4.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
7. SLF4J: Found binding in[jar:file:/opt/hadoop/hadoop-
2.8.0/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
8. SLF4J: Seehttp://www.slf4j.org/codes.html#multiple_bindings
for an explanation.
9. SLF4J: Actual binding is of
type[org.apache.logging.slf4j.Log4jLoggerFactory]
10. Metastore connection URL:
jdbc:mysql://192.168.27.138:3306/hive?
createDatabaseIfNotExist=true
11. Metastore Connection Driver :    com.mysql.jdbc.Driver
12. Metastore connection User:       root
```

```
13. Starting metastore schema initialization to2.1.0
14. Initialization scripthive-schema-2.1.0.mysql.sql
15. Initialization script completed
16. schemaTool completed
17. [root@master bin]# cd   /opt/hive/apache-hive-2.1.1-bin/bin
18. [root@master bin]# ./hive
19. which: no hbase
in(.:/opt/java/jdk1.8.0_121/bin:/opt/hadoop/hadoop-
2.8.0/bin:/opt/hadoop/hadoop-2.8.0/sbin:/opt/spark/spark-1.6.3-
bin-hadoop2.4-without-hive/bin:/opt/zookeeper/zookeeper-
3.4.10/bin:/opt/hive/apache-hive-2.1.1-bin/bin:/opt/maven/apache-
maven-3.3.9/bin:/opt/scala/scala-
2.11.8/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/roo
t/bin)
20. SLF4J: Class path contains multiple SLF4Jbindings.
21. SLF4J: Found binding in[jar:file:/opt/hive/apache-hive-2.1.1-
bin/lib/log4j-slf4j-impl-
2.4.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
22. SLF4J: Found binding in[jar:file:/opt/hive/apache-hive-2.1.1-
bin/lib/spark-assembly-1.6.3-
hadoop2.4.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
23. SLF4J: Found binding in[jar:file:/opt/hadoop/hadoop-
2.8.0/share/hadoop/common/lib/slf4j-log4j12-
1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
24. SLF4J: See
http://www.slf4j.org/codes.html#multiple_bindingsfor an
explanation.
25. SLF4J: Actual binding is of
type[org.apache.logging.slf4j.Log4jLoggerFactory]
26.
27. Logging initialized using configuration
infile:/opt/hive/apache-hive-2.1.1-bin/conf/hive-
log4j2.properties Async: true
28. hive>
```


本指南详细指引读者在macOS系统环境下安装Hbase。请务必仔细阅读完厦门大学林子雨编著的《大数据技术原理与应用》第4章节，再结合本指南进行学习。

HBase是一个开源的非关系型分布式数据库（NoSQL），它参考了谷歌的BigTable建模，实现的编程语言为 Java。它是Apache软件基金会的Hadoop项目的一部分，运行于HDFS文件系统之上，为 Hadoop 提供类似于BigTable 规模的服务。因此，它可以容错地存储海量稀疏的数据。

本教程将指导如何在苹果macOS系统安装Hbase。

安装Hbase

方法一:
从官网下载后解压
方法二:

```
# 利用homebrew安装Hbase
$ brew install Hbase
```

安装后,Hbase的安装路径：/usr/local/Cellar/hbase/1.1.2/

分布式配置

- 修改/etc/profile,添加如下内容:

```
export HBASE_HOME=/opt/hbase-1.2.6
export PATH=$PATH:$HBASE_HOME/bin
```

- 修改hbase-env.sh

开启HBASE_MANAGES_ZK,改值默认是注释的，作用是：使用自带的ZooKeeper。

```
export
JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home
export HBASE_CLASSPATH=/opt/hadoop-2.7.5/etc/hadoop
export HBASE_MANAGES_ZK=false
```

- 修改hbase-site.xml

在原来的configuration标签内部添加如下配置数据

```
<!-- HDFS的NameNode的位置 -->
    <property>
        <name>hbase.rootdir</name>
        <value>hdfs://localhost:9000/user/hbase</value>
    </property>
    <!-- false是单机模式，true是分布式模式 -->
    <property>
        <name>hbase.cluster.distributed</name>
        <value>false</value>
    </property>
```

- 需要注意的是：
- hbase.rootdir这个值需要设置成之前Hadoop的core-site.xml配置的fs.default.name值。忘记的话，请自己打开Hadoop的配置文件core-site.xml查看。

运行Hbase

运行Hbase前，我们需要先运行Hadoop伪分布式模式,再运行Hbase

```
cd /usr/local/Cellar/hadoop/2.7.1  # 进入hadoop目录
sbin/start-dfs.sh  # 运行hadoop
cd /usr/local/Cellar/hbase/1.1.2  # 进入Hbase目录
bin/start-hbase.sh   # 运行Hbase
```

运行后，我们利用jps验证是否运行成功

```
jps
```

结果如下：

```
1461 HRegionServer
2165 Jps
919 DataNode
```

```
1305 HQuorumPeer
827 NameNode
1356 HMaster
1037 SecondaryNameNode
```

编程实践

执行命令

```
hbase shell
```

进入shell界面

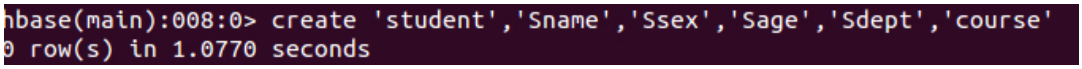
1. 利用Shell命令

1.1 HBase中创建表

HBase中用create命令创建表，具体如下：

```
create 'student','Sname','Ssex','Sage','Sdept','course'
```

命令执行截图如下：



此时，即创建了一个“student”表，属性有：Sname,Ssex,Sage,Sdept,course。因为HBase的表中会有一个系统默认的属性作为行键，无需自行创建，默认为put命令操作中表名后第一个数据。创建完“student”表后，可通过describe命令查看“student”表的基本信息。命令执行截图如下：

```
hbase(main):009:0> describe 'student'
DESCRIPTION                                                                                       ENABLED
 'student', {NAME => 'Sage', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NON true
 E', REPLICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERS
 IONS => '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE =>
 '65536', IN_MEMORY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}
 , {NAME => 'Sdept', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPL
 ICATION_SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS =>
 '0', TTL => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536',
  IN_MEMORY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME
 => 'Sname', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_
 SCOPE => '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL
  => '2147483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMO
 RY => 'false', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME => 'Ssex
 ', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE =>
 '0', VERSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147
 483647', KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'fa
 lse', ENCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}, {NAME => 'course', DATA
 _BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VE
 RSIONS => '3', COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647'
 , KEEP_DELETED_CELLS => 'false', BLOCKSIZE => '65536', IN_MEMORY => 'false', E
 NCODE_ON_DISK => 'true', BLOCKCACHE => 'true'}
1 row(s) in 0.0380 seconds
```

1.2 HBase数据库基本操作

本小节主要介绍HBase的增、删、改、查操作。在添加数据时，HBase会自动为添加的数据添加一个时间戳，故在需要修改数据时，只需直接添加数据，HBase即会生成一个新的版本，从而完成“改”操作，旧的版本依旧保留，系统会定时回收垃圾数据，只留下最新的几个版本，保存的版本数可以在创建表的时候指定。

- 添加数据

HBase中用put命令添加数据，注意：一次只能为一个表的一行数据的一个列，也就是一个单元格添加一个数据，所以直接用shell命令插入数据效率很低，在实际应用中，一般都是利用编程操作数据。

当运行命令：put ‘student’,’95001’,’Sname’,’LiYing’时，即为student表添加了学号为95001，名字为LiYing的一行数据，其行键为95001。

```
put 'student','95001','Sname','LiYing'
```

命令执行截图如下，即为student表添加了学号为95001，名字为LiYing的一行数据，其行键为95001。

```
hbase(main):003:0> put 'student','95001','Sname','LiYing'
0 row(s) in 0.6000 seconds
```

插入数据

```
put 'student','95001','course:math','80'
```

命令执行截图如下，即为95001行下的course列族的math列添加了一个数据。

```
hbase(main):023:0> put 'student','95001','course:math','80'
0 row(s) in 0.0030 seconds
```

- 删除数据

在HBase中用delete以及deleteall命令进行删除数据操作，它们的区别是：1. delete用于删除一个数据，是put的反向操作；2. deleteall操作用于删除一行数据。

1. delete命令

```
delete 'student','95001','Ssex'
```

命令执行截图如下， 即删除了student表中95001行下的Ssex列的所有数据。

```
hbase(main):026:0> delete 'student','95001','Ssex'
0 row(s) in 0.0020 seconds

hbase(main):027:0> get 'student','95001'
COLUMN                          CELL
 Sage:                          timestamp=1442912525676, value=20
 Sdept:                         timestamp=1442912586483, value=CS
 Sname:                         timestamp=1442912495442, value=LiYing
 course:math                    timestamp=1442912802499, value=80
4 row(s) in 0.0120 seconds
```

2. deleteall命令

```
deleteall 'student','95001'
```

命令执行截图如下，即删除了student表中的95001行的全部数据。

```
hbase(main):028:0> deleteall 'student','95001'
0 row(s) in 0.0020 seconds

hbase(main):029:0> scan 'student'
ROW                                COLUMN+CELL
0 row(s) in 0.0030 seconds
```

- 查看数据

HBase中有两个用于查看数据的命令：1. get命令，用于查看表的某一个单元格数据；2. scan命令用于查看某个表的全部数据

1. get命令

```
get 'student','95001'
```

命令执行截图如下， 返回的是‘student’表‘95001’行的数据。

```
hbase(main):024:0> get 'student','95001'
COLUMN                          CELL
 Sage:                          timestamp=1442912525676, value=20
 Sdept:                         timestamp=1442912586483, value=CS
 Sname:                         timestamp=1442912495442, value=LiYing
 Ssex:                          timestamp=1442912510852, value=male
 course:math                    timestamp=1442912802499, value=80
5 row(s) in 0.0080 seconds
```

2. scan命令

```
scan 'student'
```

命令执行截图如下， 返回的是‘student’表的全部数据。

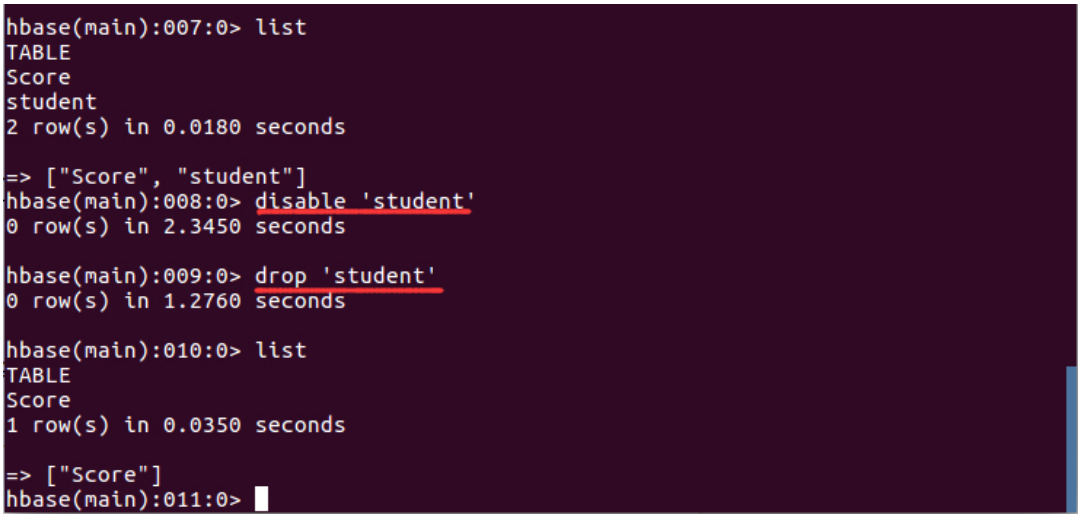
```
hbase(main):025:0> scan 'student'
ROW                             COLUMN+CELL
 95001                          column=Sage:, timestamp=1442912525676, value=20
 95001                          column=Sdept:, timestamp=1442912586483, value=CS
 95001                          column=Sname:, timestamp=1442912495442, value=LiYing
 95001                          column=Ssex:, timestamp=1442912510852, value=male
 95001                          column=course:math, timestamp=1442912802499, value=80
1 row(s) in 0.0120 seconds
```

- 删除表

删除表有两步，第一步先让该表不可用，第二步删除表。

```
disable 'student'
drop 'student'
```

命令执行截图如下：



删除表

1.3 查询表历史数据

查询表的历史版本，需要两步。

1、在创建表的时候，指定保存的版本数（假设指定为5）

```
create 'teacher',{NAME=>'username',VERSIONS=>5}
```

2、插入数据然后更新数据，使其产生历史版本数据，注意：这里插入数据和更新数据都是用put命令

```
put 'teacher','91001','username','Mary'
put 'teacher','91001','username','Mary1'
put 'teacher','91001','username','Mary2'
put 'teacher','91001','username','Mary3'
put 'teacher','91001','username','Mary4'
put 'teacher','91001','username','Mary5'
```

3、查询时，指定查询的历史版本数。默认会查询出最新的数据。（有效取值为1到5）

```
get 'teacher','91001',{COLUMN=>'username',VERSIONS=>5}
```

查询结果截图如下：

```
hbase(main):020:0>   get 'teacher','91001',{COLUMN=>'username',VERSIONS=>5}
COLUMN                CELL
 username:            timestamp=1469451374420, value=Mary5
 username:            timestamp=1469451369561, value=Mary4
 username:            timestamp=1469451366448, value=Mary3
 username:            timestamp=1469451363530, value=Mary2
 username:            timestamp=1469451351102, value=Mary1
5 row(s) in 0.0290 seconds

hbase(main):021:0>   get 'teacher','91001',{COLUMN=>'username',VERSIONS=>3}
COLUMN                CELL
 username:            timestamp=1469451374420, value=Mary5
 username:            timestamp=1469451369561, value=Mary4
 username:            timestamp=1469451366448, value=Mary3
3 row(s) in 0.0310 seconds

hbase(main):022:0>
```

查看历史数据

1.4 退出HBase数据库表操作

最后退出数据库操作，输入exit命令即可退出，注意：这里退出HBase数据库是退出对数据库表的操作，而不是停止启动HBase数据库后台运行。

```
exit
```

2. Java API编程实例

本实例使用Intellij IDEA编写java程序，来对HBase数据库进行增删改查等操作.

第一步：启动hadoop，启动hbase

```
cd /usr/local/hadoop
./sbin/start-dfs.sh
cd /usr/local/hbase
./bin/start-hbase.sh
```

第二步，新建Java Project——>新建Class

第三步：在工程中导入外部jar包：

这里只需要导入hbase安装目录中的lib文件中的所有jar包。

新版的Hbase 1.1.2的java api已经发生变化，旧版的部分api已经停止使用，教材上第四章编程实例部分，请以本教程为准。

这里给出一个编程实例,，以下是源代码：

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import java.io.IOException;

public class ExampleForHbase{
    public static Configuration configuration;
    public static Connection connection;
    public static Admin admin;

    //主函数中的语句请逐句执行，只需删除其前的//即可，如：执行insertRow时请将其他语句注释
    public static void main(String[] args)throws IOException{
        //创建一个表，表名为Score，列族为sname,course
        createTable("Score",new String[]{"sname","course"});

        //在Score表中插入一条数据，其行键为95001,sname为Mary（因为sname列族下没有子列所以第四个参数为空）
        //等价命令：put 'Score','95001','sname','Mary'
        //insertRow("Score", "95001", "sname", "", "Mary");
        //在Score表中插入一条数据，其行键为95001,course:Math为88（course为列族，Math为course下的子列）
        //等价命令：put 'Score','95001','score:Math','88'
        //insertRow("Score", "95001", "course", "Math", "88");
```

```
        //在Score表中插入一条数据，其行键为95001,course:English为85（course为列族，English为course下的子列）
        //等价命令：put 'Score','95001','score:English','85'
        //insertRow("Score", "95001", "course", "English", "85");

        //1、删除Score表中指定列数据，其行键为95001,列族为course，列为Math
        //执行这句代码前请deleteRow方法的定义中，将删除指定列数据的代码取消注释注释，将删除制定列族的代码注释
        //等价命令：delete 'Score','95001','score:Math'
        //deleteRow("Score", "95001", "course", "Math");

        //2、删除Score表中指定列族数据，其行键为95001,列族为course（95001的Math和English的值都会被删除）
        //执行这句代码前请deleteRow方法的定义中，将删除指定列数据的代码注释，将删除制定列族的代码取消注释
        //等价命令：delete 'Score','95001','score'
        //deleteRow("Score", "95001", "course", "");

        //3、删除Score表中指定行数据，其行键为95001
        //执行这句代码前请deleteRow方法的定义中，将删除指定列数据的代码注释，以及将删除制定列族的代码注释
        //等价命令：deleteall 'Score','95001'
        //deleteRow("Score", "95001", "", "");

        //查询Score表中，行键为95001，列族为course，列为Math的值
        //getData("Score", "95001", "course", "Math");
        //查询Score表中，行键为95001，列族为sname的值（因为sname列族下没有子列所以第四个参数为空）
        //getData("Score", "95001", "sname", "");

        //删除Score表
        //deleteTable("Score");
```

```
    }

    //建立连接
    public static void init(){
        configuration  = HBaseConfiguration.create();

configuration.set("hbase.rootdir","hdfs://localhost:9000/hb
ase");
        try{
            connection =
ConnectionFactory.createConnection(configuration);
            admin = connection.getAdmin();
        }catch (IOException e){
            e.printStackTrace();
        }
    }
    //关闭连接
    public static void close(){
        try{
            if(admin != null){
                admin.close();
            }
            if(null != connection){
                connection.close();
            }
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    /**
     * 建表。HBase的表中会有一个系统默认的属性作为主键，主键无需自行
创建，默认为put命令操作中表名后第一个数据，因此此处无需创建id列
     * @param myTableName 表名
```

```
     * @param colFamily 列族名
     * @throws IOException
     */
    public static void createTable(String
myTableName,String[] colFamily) throws IOException {

        init();
        TableName tableName =
TableName.valueOf(myTableName);

        if(admin.tableExists(tableName)){
            System.out.println("talbe is exists!");
        }else {
            HTableDescriptor hTableDescriptor = new
HTableDescriptor(tableName);
            for(String str:colFamily){
                HColumnDescriptor hColumnDescriptor = new
HColumnDescriptor(str);

hTableDescriptor.addFamily(hColumnDescriptor);
            }
            admin.createTable(hTableDescriptor);
            System.out.println("create table success");
        }
        close();
    }
    /**
     * 删除指定表
     * @param tableName 表名
     * @throws IOException
     */
    public static void deleteTable(String tableName) throws
IOException {
        init();
```

```
        TableName tn = TableName.valueOf(tableName);
        if (admin.tableExists(tn)) {
            admin.disableTable(tn);
            admin.deleteTable(tn);
        }
        close();
    }

    /**
     * 查看已有表
     * @throws IOException
     */
    public static void listTables() throws IOException {
        init();
        HTableDescriptor hTableDescriptors[] =
admin.listTables();
        for(HTableDescriptor hTableDescriptor
:hTableDescriptors){

System.out.println(hTableDescriptor.getNameAsString());
        }
        close();
    }
    /**
     * 向某一行的某一列插入数据
     * @param tableName 表名
     * @param rowKey 行键
     * @param colFamily 列族名
     * @param col 列名（如果其列族下没有子列，此参数可为空）
     * @param val 值
     * @throws IOException
     */
    public static void insertRow(String tableName,String
rowKey,String colFamily,String col,String val) throws
```

```
IOException {
        init();
        Table table =
connection.getTable(TableName.valueOf(tableName));
        Put put = new Put(rowKey.getBytes());
        put.addColumn(colFamily.getBytes(), col.getBytes(),
val.getBytes());
        table.put(put);
        table.close();
        close();
    }

    /**
     * 删除数据
     * @param tableName 表名
     * @param rowKey 行键
     * @param colFamily 列族名
     * @param col 列名
     * @throws IOException
     */
    public static void deleteRow(String tableName,String
rowKey,String colFamily,String col) throws IOException {
        init();
        Table table =
connection.getTable(TableName.valueOf(tableName));
        Delete delete = new Delete(rowKey.getBytes());
        //删除指定列族的所有数据
        //delete.addFamily(colFamily.getBytes());
        //删除指定列的数据
        //delete.addColumn(colFamily.getBytes(),
col.getBytes());

        table.delete(delete);
        table.close();
```

```
        close();
    }
    /**
     * 根据行键rowkey查找数据
     * @param tableName 表名
     * @param rowKey 行键
     * @param colFamily 列族名
     * @param col 列名
     * @throws IOException
     */
    public static void getData(String tableName,String
rowKey,String colFamily,String col)throws  IOException{
        init();
        Table table =
connection.getTable(TableName.valueOf(tableName));
        Get get = new Get(rowKey.getBytes());
        get.addColumn(colFamily.getBytes(),col.getBytes());
        Result result = table.get(get);
        showCell(result);
        table.close();
        close();
    }
    /**
     * 格式化输出
     * @param result
     */
    public static void showCell(Result result){
        Cell[] cells = result.rawCells();
        for(Cell cell:cells){
            System.out.println("RowName:"+new
String(CellUtil.cloneRow(cell))+" ");

System.out.println("Timetamp:"+cell.getTimestamp()+" ");
            System.out.println("column Family:"+new
```

```
String(CellUtil.cloneFamily(cell))+" ");
            System.out.println("row Name:"+new
String(CellUtil.cloneQualifier(cell))+" ");
            System.out.println("value:"+new
String(CellUtil.cloneValue(cell))+" ");
        }
    }
}
```

每次执行完，都可以回到shell界面查看是否执行成功，如：执行完插入数据后，在shell界面中执行`scan 'Score'`。

3. 遇到的问题

1>. 修改hive-env.sh文件

配置HADOOP_HOME，将其设为hadoop所在目录，配置HIVE_CONF_DIR为hive配置文件所在目录，HIVE_AUX_JARS_PATH配置hive用到的jar包所在的路径主要是在hive、hbase集成时会用到

配置完成，就可以到hive的bin目录下，执行./hive，如果报错：

Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/hadoop/hive/conf/HiveConf

还需要找到hadoop配置所在路径，修改hadoop-env.sh中的export HADOOP_CLASSPATH项，

export HADOOP_CLASSPATH=$HBASE_HOME/hbase-0.94.9.jar:$HBASE_HOME/hbase-0.94.8-test.jar:${HBASE_HOME}/lib/zookeeper-3.4.5.jar:${HBASE_HOME}/lib/guava-11.0.2.jar

改成

export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_HOME/hbase-0.94.9.jar:$HBASE_HOME/hbase-0.94.8-test.jar:${HBASE_HOME}/lib/zookeeper-3.4.5.jar:${HBASE_HOME}/lib/guava-11.0.2.jar

再执行bin/hive，就可以看到hive> 了

但是，还没完，这时候看一下hive的配置文件/tmp/root/hive.log(hive配置文件的路径在conf/hive-log4j.properties下配置)，提示

2014-01-21 12:13:11,556 WARN common.LogUtils (LogUtils.java:logConfigLocation(142)) - hive-site.xml not found on CLASSPATH

说明hive-site.xml没有被找到，这也需要修改hadoop的hadoop-env.sh，再将HADOOP_CLASSPATH改为：

export HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HBASE_HOME/hbase-0.94.9.jar:$HBASE_HOME/hbase-0.94.8-test.jar:${HBASE_HOME}/lib/zookeeper-3.4.5.jar:${HBASE_HOME}/lib/guava-11.0.2.jar:/opt/apache/hive-0.12.0-bin/conf/:/opt/apache/hive-0.12.0-bin/lib/:$HBASE_HOME/conf

即再加入hive配置文件、jar包所在路径、hbase配置文件所在路径（如果需要整合hbase的话），重新执行bin/hive，日志中就没有hive-site.xml not found on CLASSPATH的提示了。

参考 http://genius-bai.iteye.com/blog/643691

http://stackoverflow.com/questions/14353394/hive-site-xml-not-found-on-classpath

2>.报错

hive启动后运行show tables;出现报错

FAILED: SemanticException org.apache.hadoop.hive.ql.metadata.HiveException: java.lang.RuntimeException: Unable to instantidata.SessionHiveMetaStoreClient

FAILED: SemanticException org.apache.hadoop.hive.ql.metadata.HiveException: java.lang.RuntimeException: Unable to instantidata.SessionHiveMetaStoreClient

这个问题的原因是HivedMetaStore服务没有启动需要手动启动一下
`hive --service metastore &`
接着上面的问题HivedMetaStore服务启动失败 报错
Caused by:
org.datanucleus.store.rdbms.exceptions.MissingTableException:
Required table missing : “`DBS`” in Catalog “” Schema “”.
DataNucleus requires this table to perform its persistence
operations. Either your MetaData is incorrect, or you need to enable
“datanucleus.schema.autoCreateTables”
需要到hive-site.xml中找到datanucleus.schema.autoCreateAll 把value改成true。再启动HivedMetaStore服务，接着启动hive。show tables;成功运行。

参考

https://cwiki.apache.org/confluence/display/Hive/GettingStarted
http://blog.csdn.net/asia_kobe/article/details/50866382
http://blog.csdn.net/an342647823/article/details/46048403

六. 安装Hbase

六.总结

centos7 最终 /etc/profile 的配置效果为(新增下如下内容):

```
# User Setting
export JAVA_HOME=/usr/java/default
export HADOOP_HOME=/alidata/opt/hadoop-2.7.5
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$PATH

export SCALA_HOME=/usr/share/scala
export PATH=$SCALA_HOME/bin:$PATH

export SPARK_HOME=/alidata/opt/spark-2.0.0
export PATH=$SPARK_HOME/bin:$PATH
```

```
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native/:$LD_LIBRARY_PATH

export SQOOP_HOME=/alidata/opt/sqoop-1.4.7
export PATH=$PATH:$SQOOP_HOME/bin

export HIVE_HOME=/alidata/opt/hive-2.1.1
export PATH=$PATH:$HIVE_HOME/bin

export HBASE_HOME=/alidata/opt/hbase-1.2.6
export PATH=$PATH:$HBASE_HOME/bin

export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
export HIVE_CONF_DIR=$HIVE_HOME/conf
export
HADOOP_CLASSPATH=$HADOOP_CLASSPATH:$HIVE_HOME/lib/*:$HBASE_H
OME/lib/*:$HIVE_HOME/conf/
export SPARK_CLASSPATH=$SPARK_HOME/jars/*:hive-2.1.1/lib/hive-exec-
2.1.1.jar:/alidata/opt/hive-2.1.1/lib/hive-common-2.1.1.jar:/alidata/opt/hbase-
1.2.6/lib/*:/alidata/opt/mysql-connector-java-5.1.46.jar
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```