实验目的：

（1）掌握在Hadoop完全分布式集群环境下搭建HBase集群的方法； （2）熟练掌握HBase shell的使用方法； （3）熟练掌握HBase shell常用命令的使用。

实验内容：

**一、HBase的安装配置**

**1.**HBase安装

1.1 解压安装包hbase-1.1.2-bin.tar.gz至路径 /usr/local，命令如下：

sudo tar -zxf ~/下载/hbase-1.1.2-bin.tar.gz -C /usr/local

1.2 将解压的文件名hbase-1.1.2改为hbase，以方便使用，命令如下：

sudo mv /usr/local/hbase-1.1.2 /usr/local/hbase

1.3 配置环境变量

将hbase下的bin目录添加到path中，这样，启动hbase就无需到/usr/local/hbase目录下，大大的方便了hbase的使用。教程下面的部分还是切换到了/usr/local/hbase目录操作，有助于初学者理解运行过程，熟练之后可以不必切换。

编辑~/.bashrc文件

vi ~/.bashrc

如果没有引入过PATH请在~/.bashrc文件尾行添加如下内容：

export PATH=$PATH:/usr/local/hbase/bin

Shell

如果已经引入过PATH请在export PATH这行追加/usr/local/hbase/bin，这里的“：”是分隔符。如下图：

配置环境变量

编辑完成后，再执行source命令使上述配置在当前终端立即生效，命令如下：

source ~/.bashrc

扩展阅读: 设置Linux环境变量的方法和区别

1.4 添加HBase权限

cd /usr/local

sudo chown -R hadoop ./hbase       #将hbase下的所有文件的所有者改为hadoop，hadoop是当前用户的用户名。

1.5 查看HBase版本，确定hbase安装成功,命令如下：

/usr/local/hbase/bin/hbase version

命令执行后，输出信息截图如下：

查看版本

看到以上输出消息表示HBase已经安装成功，接下来将分别进行HBase单机模式和伪分布式模式的配置。

2. HBase配置

HBase有三种运行模式，单机模式、伪分布式模式、分布式模式。作为学习，我们重点讨论单机模式和伪分布式模式。

以下先决条件很重要，比如没有配置JAVA\_HOME环境变量，就会报错。

– jdk

– Hadoop( 单机模式不需要，伪分布式模式和分布式模式需要)

– SSH

以上三者如果没有安装，请回到第二章的指南参考如何安装。

1.配置/usr/local/hbase/conf/hbase-env.sh。命令如下：

vi /usr/local/hbase/conf/hbase-env.sh

配置JAVA\_HOME，HBASE\_CLASSPATH，HBASE\_MANAGES\_ZK.

HBASE\_CLASSPATH设置为本机Hadoop安装目录下的conf目录（即/usr/local/hadoop/conf）

export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-amd64

export HBASE\_CLASSPATH=/usr/local/hadoop/conf

export HBASE\_MANAGES\_ZK=true

Shell

截图如下：

伪分布式hbase

2.配置/usr/local/hbase/conf/hbase-site.xml

用命令vi打开并编辑hbase-site.xml，命令如下：

vi /usr/local/hbase/conf/hbase-site.xml

修改hbase.rootdir，指定HBase数据在HDFS上的存储路径；将属性hbase.cluter.distributed设置为true。假设当前Hadoop集群运行在伪分布式模式下，在本机上运行，且NameNode运行在9000端口。

<configuration>

        <property>

                <name>hbase.rootdir</name>

                <value>hdfs://localhost:9000/hbase</value>

        </property>

        <property>

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

                <value>true</value>

        </property>

</configuration>

XML

hbase.rootdir指定HBase的存储目录；hbase.cluster.distributed设置集群处于分布式模式.

截图如下：

配置hbase-site.xml

3. 接下来测试运行HBase。

第一步：首先登陆ssh，之前设置了无密码登陆，因此这里不需要密码；再切换目录至/usr/local/hadoop ；再启动hadoop，如果已经启动hadoop请跳过此步骤。命令如下：

ssh localhost

cd /usr/local/hadoop

./sbin/start-dfs.sh

输入命令jps，能看到NameNode,DataNode和SecondaryNameNode都已经成功启动，表示hadoop启动成功，截图如下：

hadoop jps

第二步：切换目录至/usr/local/hbase;再启动HBase.命令如下：

cd /usr/local/hbase

bin/start-hbase.sh

启动成功，输入命令jps，看到以下界面说明hbase启动成功

jps

进入shell界面：

bin/hbase shell

截图如下：

hbase shell

4.停止HBase运行,命令如下：

bin/stop-hbase.sh

注意：如果在操作HBase的过程中发生错误，可以通过{HBASE\_HOME}目录（/usr/local/hbase）下的logs子目录中的日志文件查看错误原因。

这里启动关闭Hadoop和HBase的顺序一定是：

启动Hadoop—>启动HBase—>关闭HBase—>关闭Hadoop

三、 编程实践

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：

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。

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。

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。

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。

查看数据

HBase中有两个用于查看数据的命令：1. get命令，用于查看表的某一行数据；2. scan命令用于查看某个表的全部数据

1. get命令

  get 'student','95001'

， 返回的是‘student’表‘95001’行的数据。

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。

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：

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Eclipse编写java程序，来对HBase数据库进行增删改查等操作，Eclipse可以在Ubuntu软件中心搜索下载并安装。

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

jre

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。

引入外部jar包

这里给出一个编程实例,，以下是源代码：

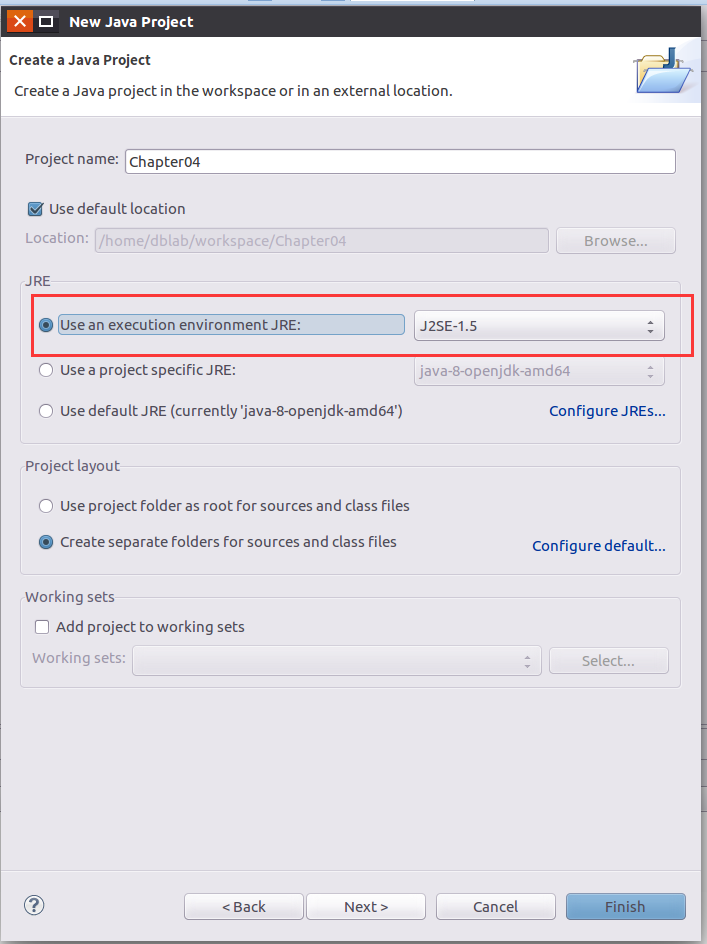
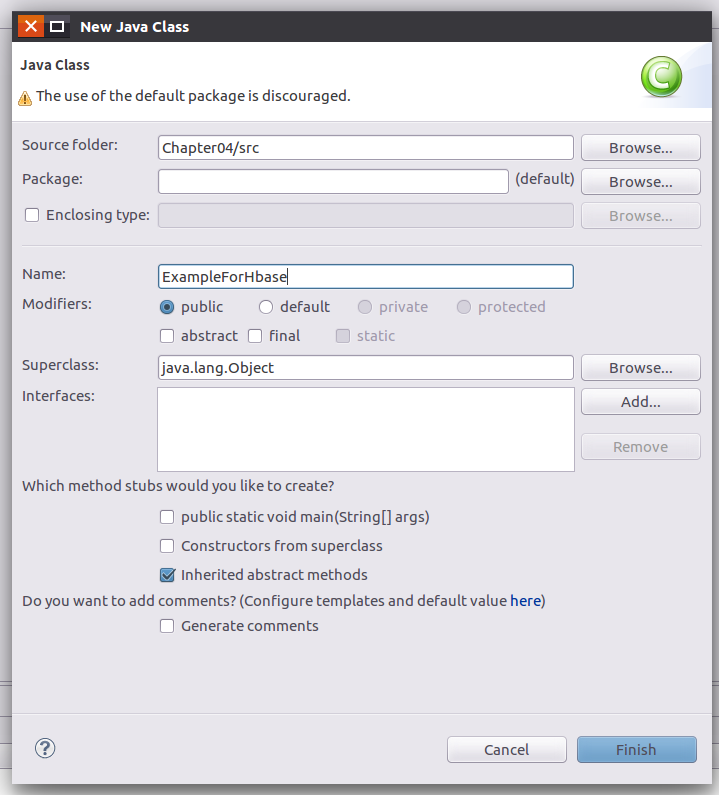
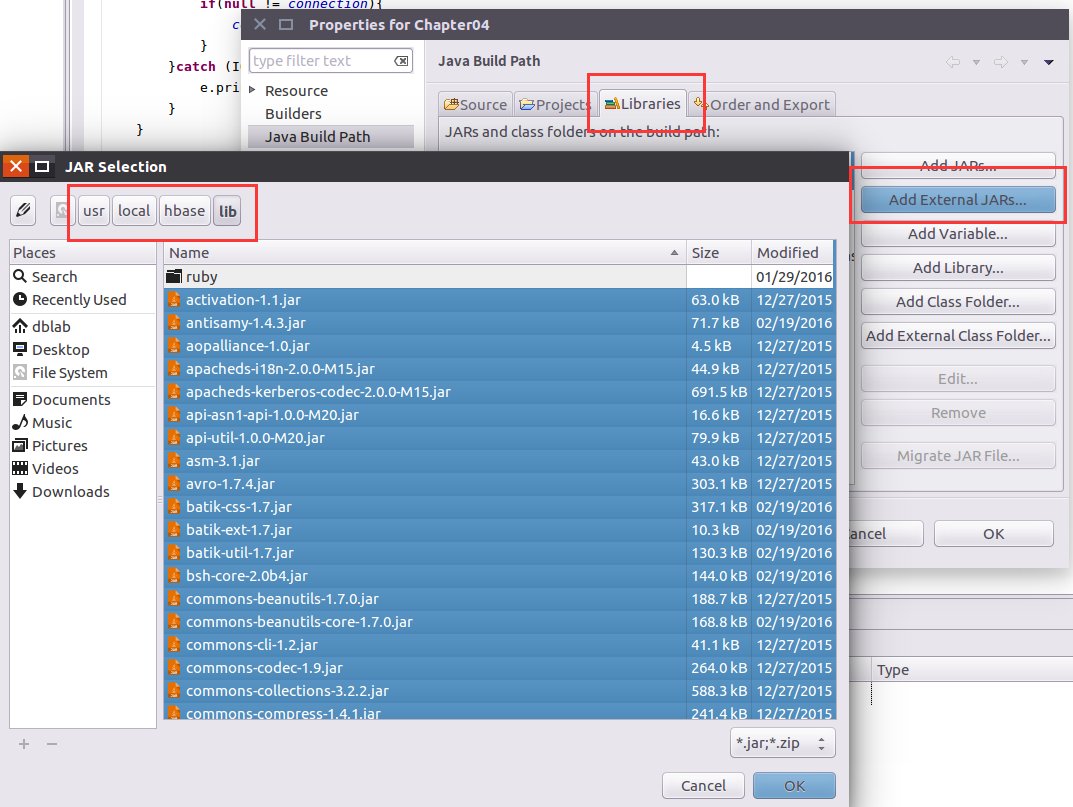
[hbase.txt](http://172.31.36.7/Lab/ueditor/jsp/upload/file/20201102/1604278668394009944.txt)

二. Java API编程实例

本实例使用Eclipse编写java程序，来对HBase数据库进行增删改查等操作，Eclipse可以在Ubuntu软件中心搜索下载并安装。  
第一步：启动hadoop，启动hbase

1. cd /usr/local/hadoop
2. ./sbin/start-dfs.sh
3. cd /usr/local/hbase
4. ./bin/start-hbase.sh

Shell 命令

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：

1. import org.apache.hadoop.conf.Configuration;
2. import org.apache.hadoop.hbase.\*;
3. import org.apache.hadoop.hbase.client.\*;
4. import java.io.IOException;
6. public class ExampleForHbase{
7. public static Configuration configuration;
8. public static Connection connection;
9. public static Admin admin;
11. //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
12. public static void main(String[] args)throws IOException{
13. //创建一个表，表名为Score，列族为sname,course
14. createTable("Score",new String[]{"sname","course"});
16. //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）
17. //等价命令：put 'Score','95001','sname','Mary'
18. //insertRow("Score", "95001", "sname", "", "Mary");
19. //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）
20. //等价命令：put 'Score','95001','score:Math','88'
21. //insertRow("Score", "95001", "course", "Math", "88");
22. //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）
23. //等价命令：put 'Score','95001','score:English','85'
24. //insertRow("Score", "95001", "course", "English", "85");
26. //1、删除Score表中指定列数据，其行键为95001,列族为course，列为Math
27. //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
28. //等价命令：delete 'Score','95001','score:Math'
29. //deleteRow("Score", "95001", "course", "Math");
31. //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）
32. //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
33. //等价命令：delete 'Score','95001','score'
34. //deleteRow("Score", "95001", "course", "");
36. //3、删除Score表中指定行数据，其行键为95001
37. //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
38. //等价命令：deleteall 'Score','95001'
39. //deleteRow("Score", "95001", "", "");
41. //查询Score表中，行键为95001，列族为course，列为Math的值
42. //getData("Score", "95001", "course", "Math");
43. //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）
44. //getData("Score", "95001", "sname", "");
46. //删除Score表
47. //deleteTable("Score");
48. }
50. //建立连接
51. public static void init(){
52. configuration = HBaseConfiguration.create();
53. configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");
54. try{
55. connection = ConnectionFactory.createConnection(configuration);
56. admin = connection.getAdmin();
57. }catch (IOException e){
58. e.printStackTrace();
59. }
60. }
61. //关闭连接
62. public static void close(){
63. try{
64. if(admin != null){
65. admin.close();
66. }
67. if(null != connection){
68. connection.close();
69. }
70. }catch (IOException e){
71. e.printStackTrace();
72. }
73. }
75. /\*\*
76. \* 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
77. \* @param myTableName 表名
78. \* @param colFamily 列族名
79. \* @throws IOException
80. \*/
81. public static void createTable(String myTableName,String[] colFamily) throws IOException {
83. init();
84. TableName tableName = TableName.valueOf(myTableName);
86. if(admin.tableExists(tableName)){
87. System.out.println("talbe is exists!");
88. }else {
89. HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
90. for(String str:colFamily){
91. HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(str);
92. hTableDescriptor.addFamily(hColumnDescriptor);
93. }
94. admin.createTable(hTableDescriptor);
95. System.out.println("create table success");
96. }
97. close();
98. }
99. /\*\*
100. \* 删除指定表
101. \* @param tableName 表名
102. \* @throws IOException
103. \*/
104. public static void deleteTable(String tableName) throws IOException {
105. init();
106. TableName tn = TableName.valueOf(tableName);
107. if (admin.tableExists(tn)) {
108. admin.disableTable(tn);
109. admin.deleteTable(tn);
110. }
111. close();
112. }
114. /\*\*
115. \* 查看已有表
116. \* @throws IOException
117. \*/
118. public static void listTables() throws IOException {
119. init();
120. HTableDescriptor hTableDescriptors[] = admin.listTables();
121. for(HTableDescriptor hTableDescriptor :hTableDescriptors){
122. System.out.println(hTableDescriptor.getNameAsString());
123. }
124. close();
125. }
126. /\*\*
127. \* 向某一行的某一列插入数据
128. \* @param tableName 表名
129. \* @param rowKey 行键
130. \* @param colFamily 列族名
131. \* @param col 列名（如果其列族下没有子列，此参数可为空）
132. \* @param val 值
133. \* @throws IOException
134. \*/
135. public static void insertRow(String tableName,String rowKey,String colFamily,String col,String val) throws IOException {
136. init();
137. Table table = connection.getTable(TableName.valueOf(tableName));
138. Put put = new Put(rowKey.getBytes());
139. put.addColumn(colFamily.getBytes(), col.getBytes(), val.getBytes());
140. table.put(put);
141. table.close();
142. close();
143. }
145. /\*\*
146. \* 删除数据
147. \* @param tableName 表名
148. \* @param rowKey 行键
149. \* @param colFamily 列族名
150. \* @param col 列名
151. \* @throws IOException
152. \*/
153. public static void deleteRow(String tableName,String rowKey,String colFamily,String col) throws IOException {
154. init();
155. Table table = connection.getTable(TableName.valueOf(tableName));
156. Delete delete = new Delete(rowKey.getBytes());
157. //删除指定列族的所有数据
158. //delete.addFamily(colFamily.getBytes());
159. //删除指定列的数据
160. //delete.addColumn(colFamily.getBytes(), col.getBytes());
162. table.delete(delete);
163. table.close();
164. close();
165. }
166. /\*\*
167. \* 根据行键rowkey查找数据
168. \* @param tableName 表名
169. \* @param rowKey 行键
170. \* @param colFamily 列族名
171. \* @param col 列名
172. \* @throws IOException
173. \*/
174. public static void getData(String tableName,String rowKey,String colFamily,String col)throws IOException{
175. init();
176. Table table = connection.getTable(TableName.valueOf(tableName));
177. Get get = new Get(rowKey.getBytes());
178. get.addColumn(colFamily.getBytes(),col.getBytes());
179. Result result = table.get(get);
180. showCell(result);
181. table.close();
182. close();
183. }
184. /\*\*
185. \* 格式化输出
186. \* @param result
187. \*/
188. public static void showCell(Result result){
189. Cell[] cells = result.rawCells();
190. for(Cell cell:cells){
191. System.out.println("RowName:"+new String(CellUtil.cloneRow(cell))+" ");
192. System.out.println("Timetamp:"+cell.getTimestamp()+" ");
193. System.out.println("column Family:"+new String(CellUtil.cloneFamily(cell))+" ");
194. System.out.println("row Name:"+new String(CellUtil.cloneQualifier(cell))+" ");
195. System.out.println("value:"+new String(CellUtil.cloneValue(cell))+" ");
196. }
197. }
198. }

Java

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'。截图如下:  
