**电商交易记录分析系统**

一：1.课题概述

2.集群环境搭建

二：架构设计

三：数据导入

1. 使用navicat将数据.csv文件导入到mysql下
2. 使用sqoop将mysql中数据导入到hbase中

四: web应用的设计

五: 数据的备份与恢复

六: 关系型数据库与非关系型数据库的区别

一:

1. 课题概述

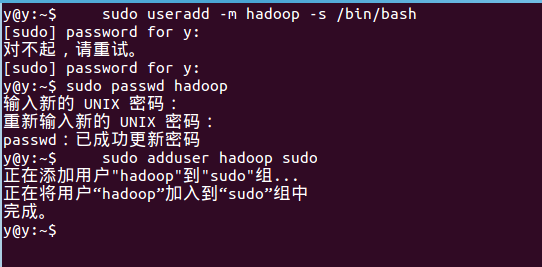
通过搭建hbase集群环境导入所提供的的电商交易记录数据文件进行操作。

2.集群环境搭建

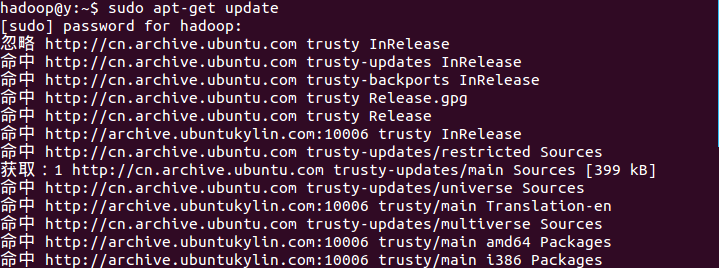
ubuntu+hadoop-2.6.0-cdh5.3.6+hbase1.1.2三个节点的集群分布.

安装hadoop

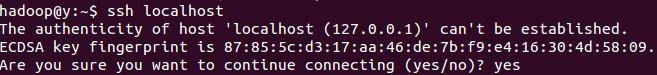
创建hadoop用户



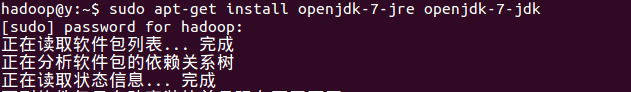
更新apt

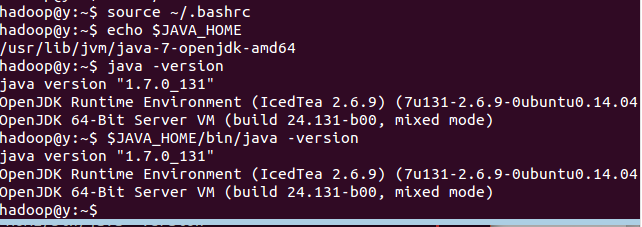


安装ssh

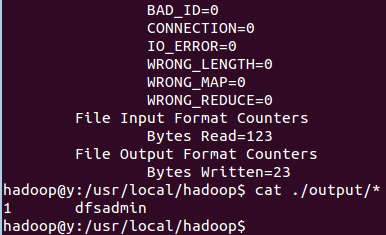


安装java环境

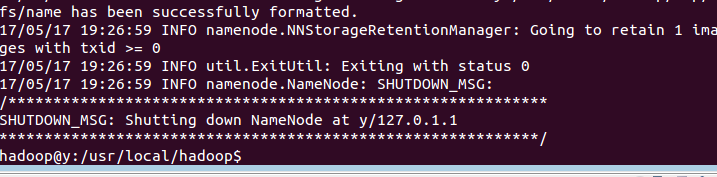


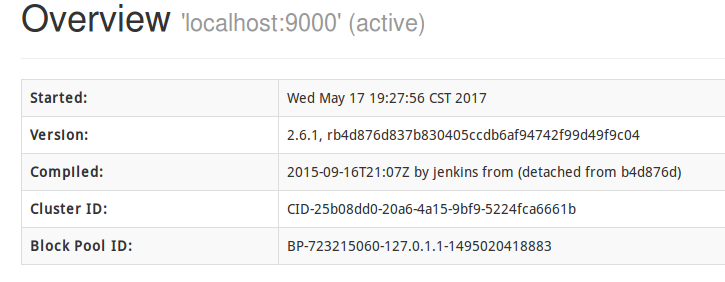


Hadoop单机配置

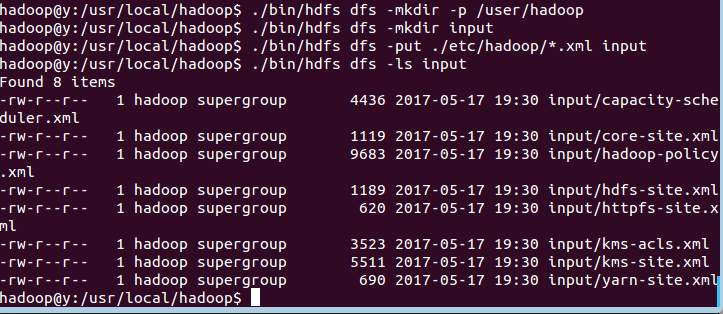


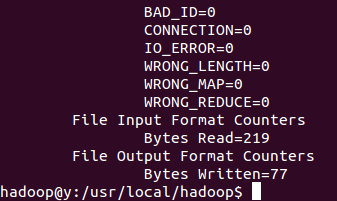
Hadoop伪分布式配置

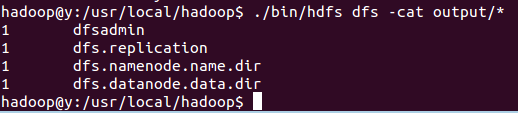




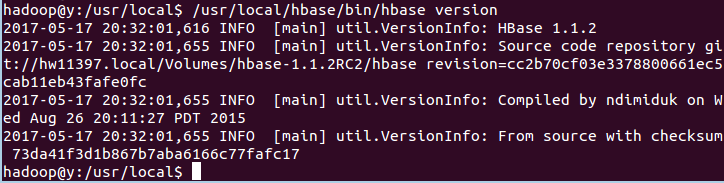
运行hadoop伪分布式



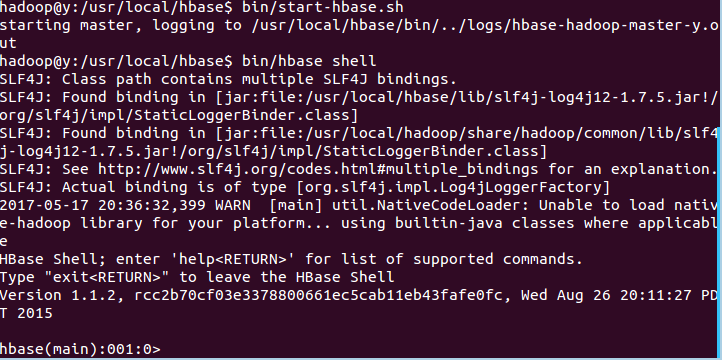




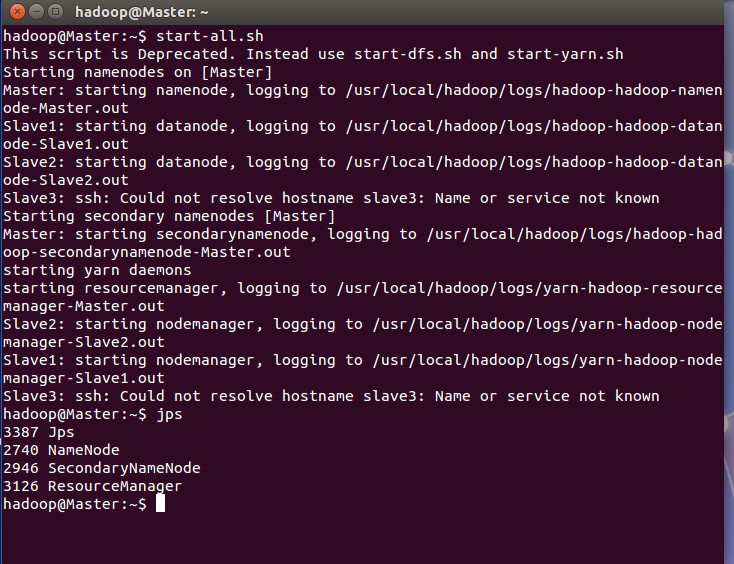
1. 安装hbase

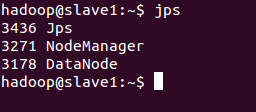


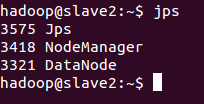
Hbase配置



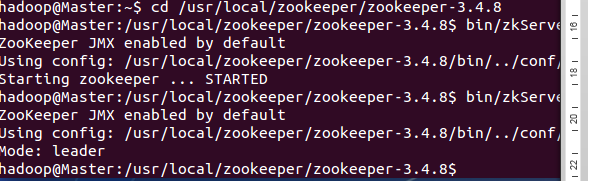
1. 在master 节点启动hadoop start-all.sh

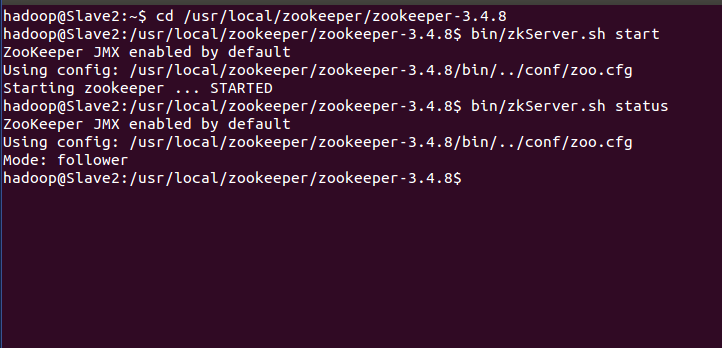
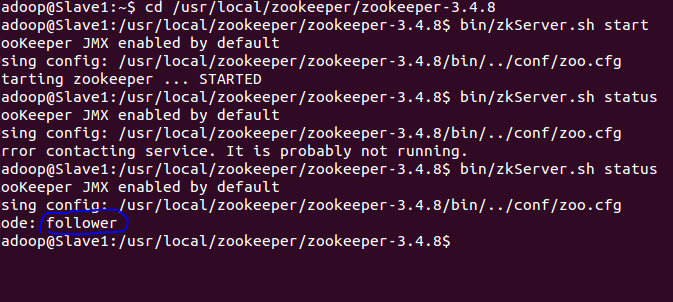




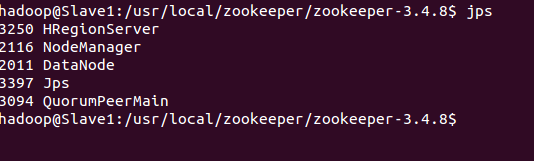
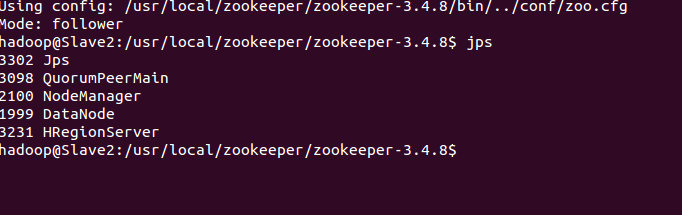
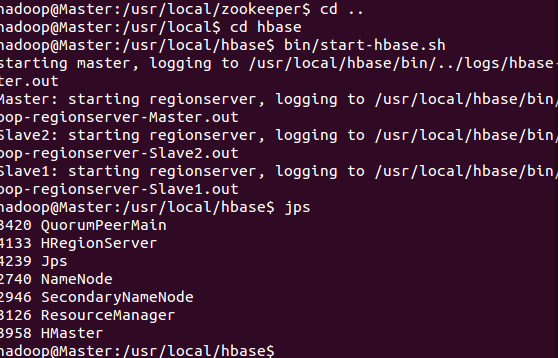


1. 在各节点启动zookeeper



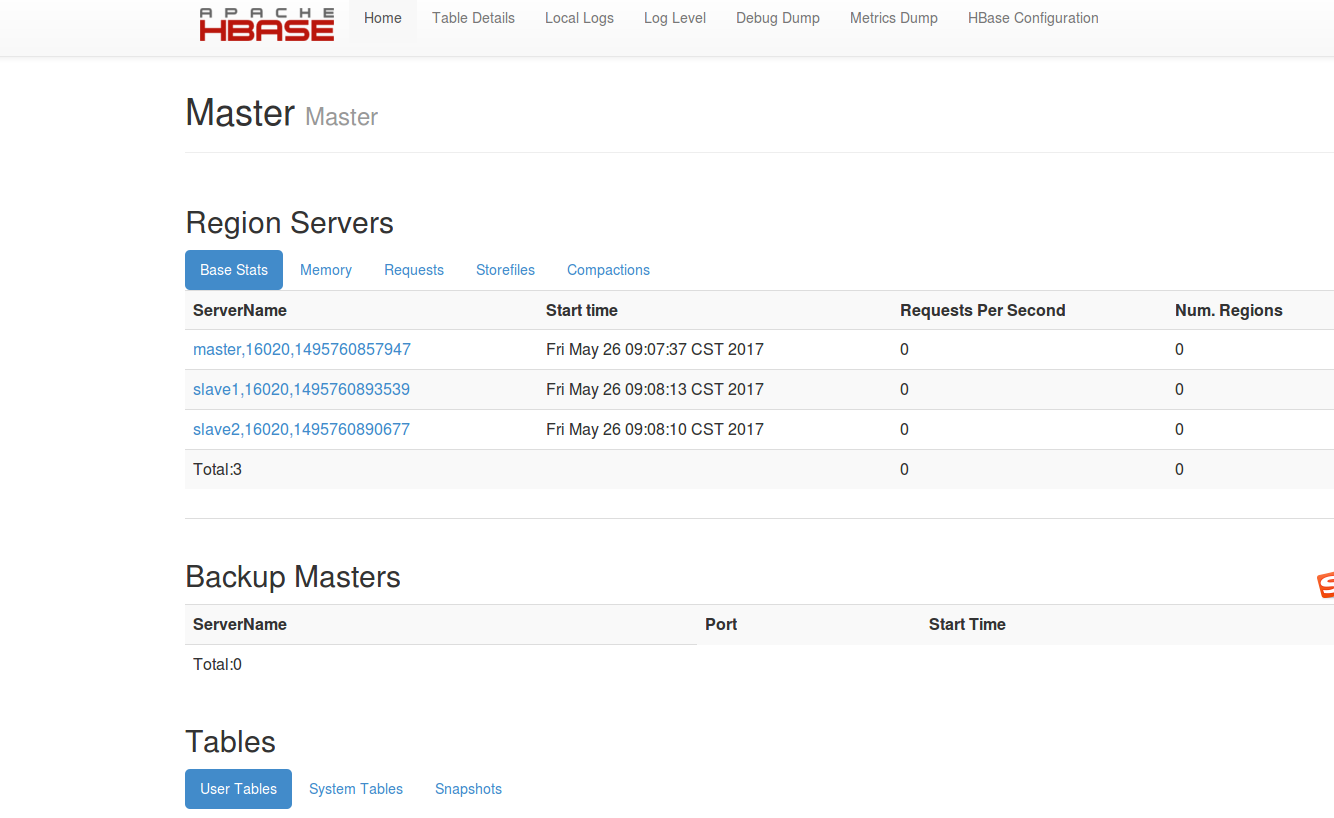
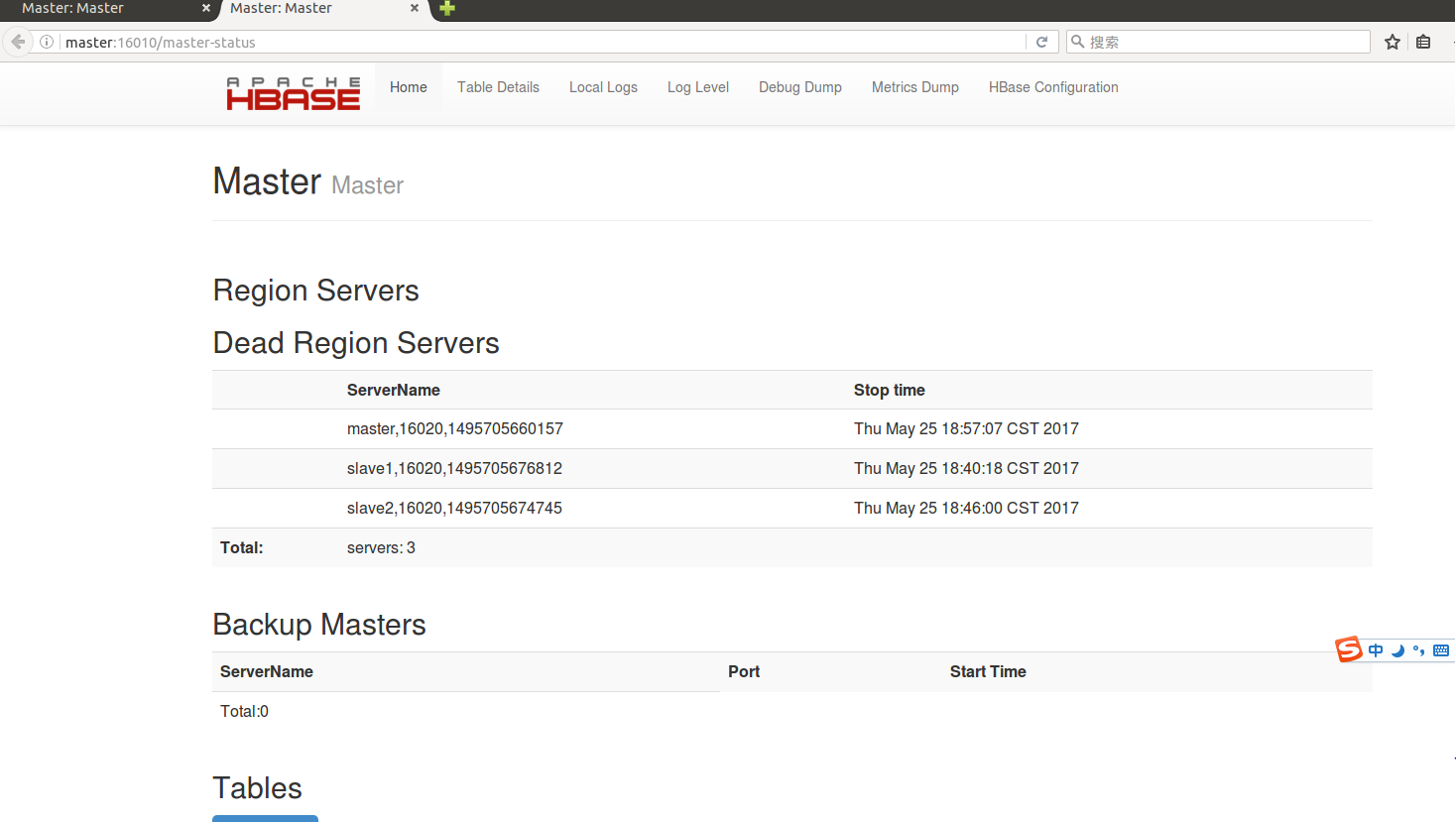


1. 在master节点启动hbase start-hbase.sh



所有节点上的所有进程都能正常启动（**注意查看安全模式状态，可能需要手动关闭安全模式**）

**Web界面同样正常：**



二:架构设计

ubuntu+hadoop-2.6.0+hbase1.1.2+zookeeper3.4.8三个节点的集群分布。

Master节点master：192.168.49.131

Slave节点slave1：192.168.49.132

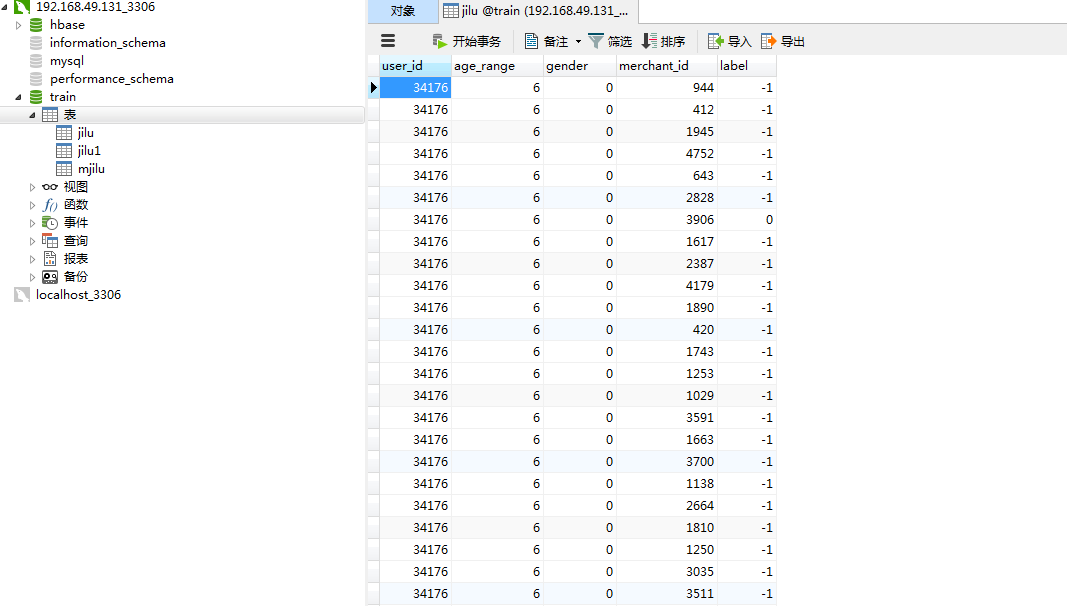
slave2：192.168.49.133

三：数据导入

1. navicat的使用

Windows下安装navicat，连接上虚拟机环境下节点。

新建表设计好字段属性，导入csv文件。



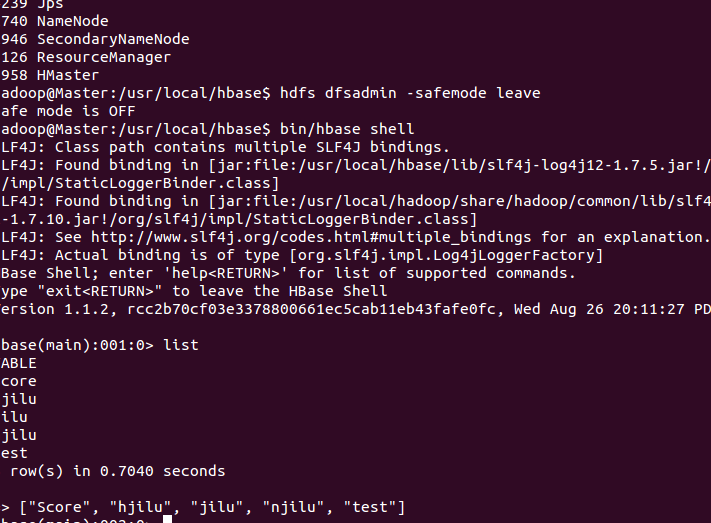
**同时应该注意，在/etc/mysql下需要修改my.cnf配置文件，注释掉ip绑定，同时在终端中给予所有用户mysql的远程连接权限，不然会导致连接失败。**

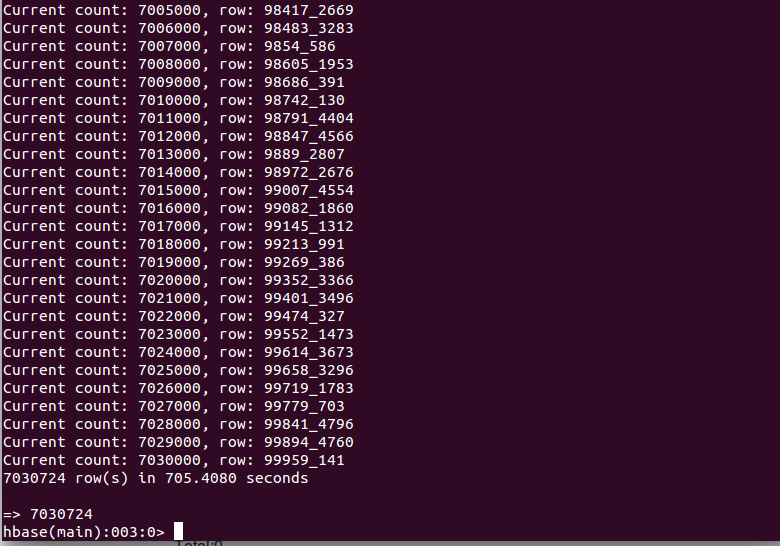
1. sqoop的使用

配置完毕环境变量后，运行sqoop语句。

sqoop import --connect jdbc:mysql://192.168.49.131:3306/train --username 'root' --password 123456 --table jilu --columns"user\_id,age\_range,gender,merchant\_id,label" --hbase-table jilu --column-family cf1 --hbase-row-key user\_id,merchant\_id -m 1

导入成功后的数据查看结果如下：





四：web应用的设计

1. 添加数据

package servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.Table;

import hbase.lianjie;

public class InsertServlet extends HttpServlet{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

request.setCharacterEncoding("gb2312"); //设置请求编码

String tableName=request.getParameter("tableName");

String rowkey=request.getParameter("rowKey"); //获取提交的rowkey

String colFamily=request.getParameter("colFamily");

String col=request.getParameter("col");

String val=request.getParameter("val");

PrintWriter out=response.getWriter();

lianjie.init();

Table table = lianjie.connection.getTable(TableName.valueOf(tableName));

Put put = new Put(rowkey.getBytes());

put.addColumn(colFamily.getBytes(), col.getBytes(), val.getBytes());

table.put(put);

out.println("<p>success:"+"</p> ");

lianjie.close();

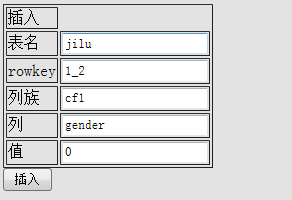
}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

doGet(request,response);

}

}



（2）查询数据

package servlet;

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

import org.apache.hadoop.hbase.Cell;

import org.apache.hadoop.hbase.CellUtil;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.Get;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.client.Table;

import hbase.lianjie;

public class FindServlet extends HttpServlet{

private static final long serialVersionUID = 1L;

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

request.setCharacterEncoding("gb2312"); //设置请求编码

String tableName=request.getParameter("tableName");

String rowkey=request.getParameter("rowKey"); //获取提交的rowkey

PrintWriter out=response.getWriter();

lianjie.init();

Table table = lianjie.connection.getTable(TableName.valueOf(tableName));

Get get = new Get(rowkey.getBytes());

Result result = table.get(get);

if(result==null)

{

out.println("<p>fail"+"</p> ");

}

String[] s=rowkey.split("\_");

out.println("user\_id="+s[0]+"\t");

out.println("merchant\_id="+s[1]+"\t");

Cell[] cells = result.rawCells();

for(Cell cell:cells){

out.println("<p>owName:"+new String(CellUtil.cloneRow(cell))+"</p> ");

out.println("<p>Timetamp:"+cell.getTimestamp()+"</p> ");

out.println("<p>column Family:"+new String(CellUtil.cloneFamily(cell))+"</p> ");

out.println("<p>row Name:"+new String(CellUtil.cloneQualifier(cell))+" </p>");

out.println("<p>value:"+new String(CellUtil.cloneValue(cell))+"</p> ");

}

lianjie.showCell(result);

table.close();

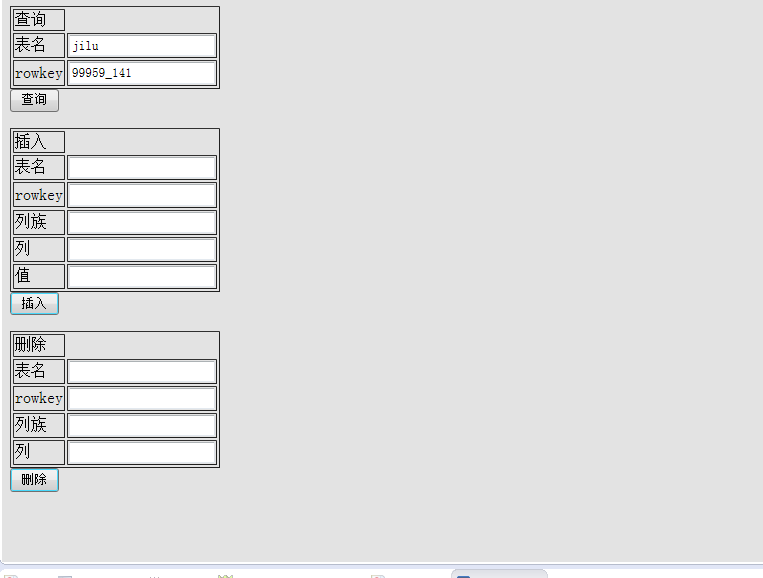
lianjie.close();

}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

doGet(request,response);

}

}

1. 删除数据

package servlet;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.Delete;

import org.apache.hadoop.hbase.client.Table;

import hbase.lianjie;

public class DeleteServlet extends HttpServlet{

private static final long serialVersionUID = 1L;

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

request.setCharacterEncoding("gb2312"); //设置请求编码

String tableName=request.getParameter("tableName");

String rowkey=request.getParameter("rowKey"); //获取提交的rowkey

String colFamily=request.getParameter("colFamily");

String col=request.getParameter("col");

PrintWriter out=response.getWriter();

lianjie.init();

Table table = lianjie.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();

out.println("<p>success:"+"</p> ");

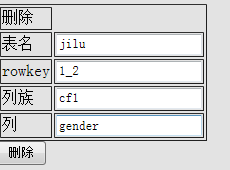
lianjie.close();

}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{

doGet(request,response);

}

}

1. jsp页面

<%@ page language="java" pageEncoding="gb2312" import="java.util.\*,java.sql.\*,hbase.\*"%>

<html>

<head>

<title></title>

</head>

<body bgcolor="#E3E3E3">

<form action="findServlet" method="post">

<table border="1">

<tbody>

<tr>

<td>查询</td>

</tr>

<tr>

<td>表名</td>

<td><input type="text" name="tableName"/></td>

</tr>

<tr>

<td>rowkey</td>

<td><input type="text" name="rowKey"/></td>

</tr>

</tbody>

</table>

<input type="submit" value="查询"/>

</form>

<form action="insertServlet" method="post">

<table border="1">

<tbody>

<tr>

<td>插入</td>

</tr>

<tr>

<td>表名</td>

<td><input type="text" name="tableName"/></td>

</tr>

<tr>

<td>rowkey</td>

<td><input type="text" name="rowKey"/></td>

</tr>

<tr>

<td>列族</td>

<td><input type="text" name="colFamily"/></td>

</tr>

<tr>

<td>列</td>

<td><input type="text" name="col"/></td>

</tr>

<tr>

<td>值</td>

<td><input type="text" name="val"/></td>

</tr>

</tbody>

</table>

<input type="submit" value="插入"/>

</form>

<form action="deleteServlet" method="post">

<table border="1">

<tbody>

<tr>

<td>删除</td>

</tr>

<tr>

<td>表名</td>

<td><input type="text" name="tableName"/></td>

</tr>

<tr>

<td>rowkey</td>

<td><input type="text" name="rowKey"/></td>

</tr>

<tr>

<td>列族</td>

<td><input type="text" name="colFamily"/></td>

</tr>

<tr>

<td>列</td>

<td><input type="text" name="col"/></td>

</tr>

</tbody>

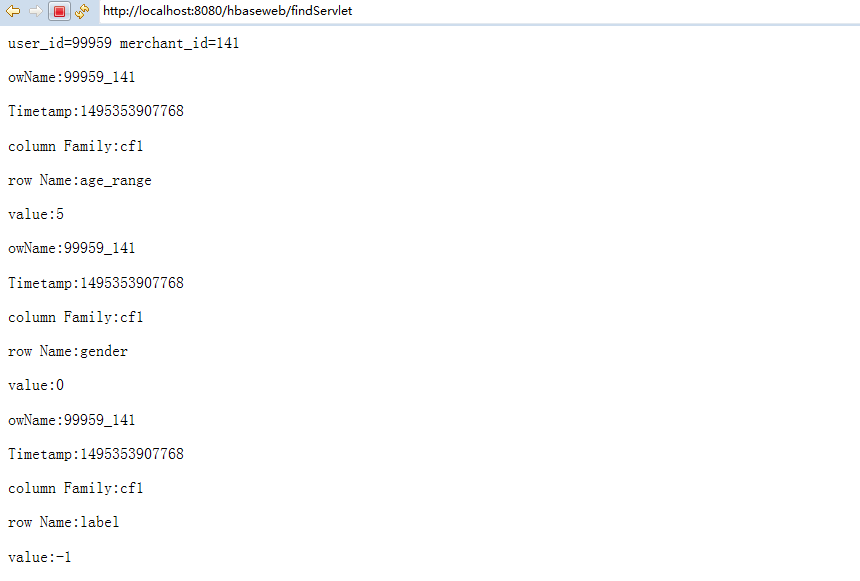
</table>

<input type="submit" value="删除"/>

</form>

</body>

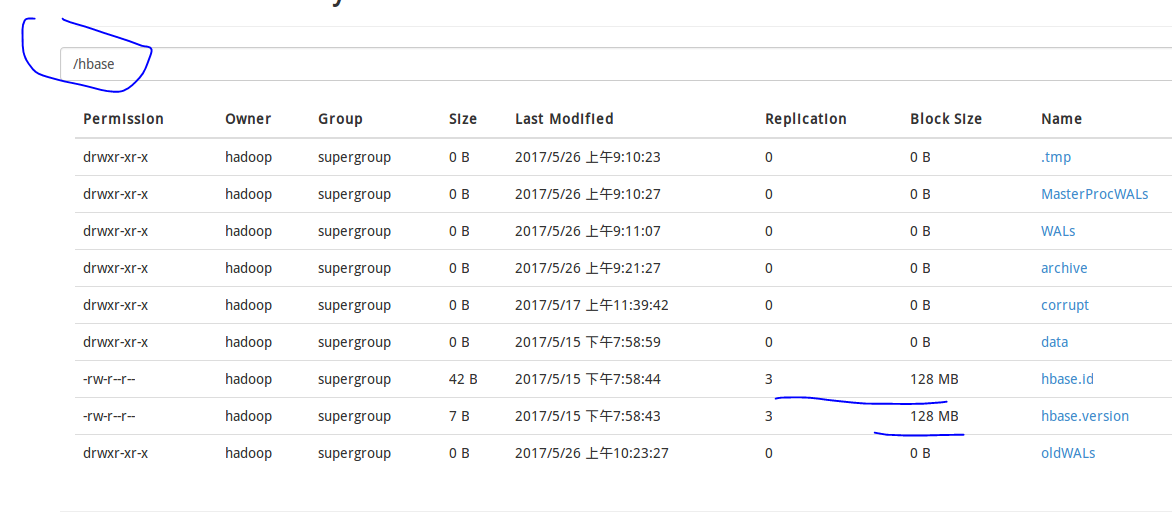
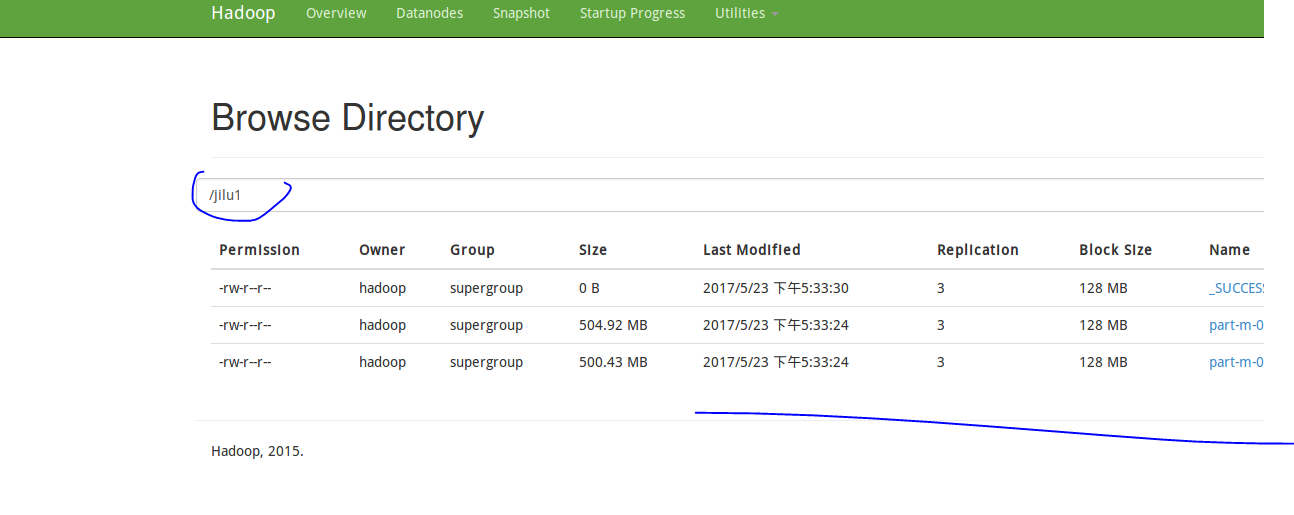
</html>



五：数据的备份与恢复

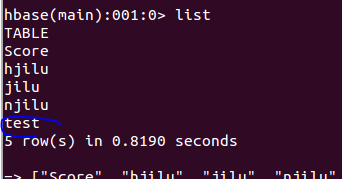
1.进行热备份：./hbase org.apache.hadoop.hbase.mapreduce.Export test hdfs://master:8020/jilu1

2.web查看hdfs文件



1. 进行数据恢复

./hbase org.apache.hadoop.hbase.mapreduce.Import test hdfs://master:8020/test



数据恢复成功

六：总结

对于两种数据库的认识

关系型[**数据库**](http://lib.csdn.net/base/mysql)，是指采用了关系模型来组织数据的数据库。简单来说，关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。

关系模型中常用的概念：

关系，元组，属性，域，关键字，主键，关系模式，表结构

关系型数据库的优点：

* 容易理解：二维表结构是非常贴近逻辑世界的一个概念，关系模型相对网状、层次等其他模型来说更容易理解
* 使用方便：通用的SQL语言使得操作关系型数据库非常方便
* 易于维护：丰富的完整性(实体完整性、参照完整性和用户定义的完整性)大大减低了数据冗余和数据不一致的概率

关系型数据库瓶颈

* 高并发读写需求
* 海量数据的高效率读写
* 高扩展性和可用性

对网站来说，关系型数据库的很多特性不再需要了：

* 事务一致性
* 读写实时性
* 复杂SQL，特别是多表关联查询

NoSQL：非关系数据库

NoSQL一词首先是Carlo Strozzi在1998年提出来的，指的是他开发的一个没有SQL功能，轻量级的，开源的关系型数据库。2009年初，Johan Oskarsson举办了一场关于开源分布式数据库的讨论，Eric Evans在这次讨论中再次提出了NoSQL一词，用于指代那些非关系型的，分布式的，且一般不保证遵循ACID原则的数据存储系统。

注：数据库事务必须具备ACID特性，ACID是Atomic原子性，Consistency一致性，Isolation隔离性，Durability持久性。

非关系型数据库提出另一种理念，如，以键值对存储，且结构不固定，每一个元组可以有不一样的字段，每个元组可以根据需要增加一些自己的键值对，这样就不会局限于固定的结构，可以减少一些时间和空间的开销。使用这种方式，用户可以根据需要去添加自己需要的字段，这样，为了获取用户的不同信息，不需要像关系型数据库中，要对多表进行关联查询。仅需要根据id取出相应的value就可以完成查询。但非关系型数据库由于很少的约束，他也不能够提供像SQL所提供的where这种对于字段属性值情况的查询。并且难以体现设计的完整性。他只适合存储一些较为简单的数据，对于需要进行较复杂查询的数据，SQL数据库显的更为合适。

**关系型数据库  V.S.  非关系型数据库**

关系型数据库的最大特点就是事务的一致性：传统的关系型数据库读写操作都是事务的，具有ACID的特点，这个特性使得关系型数据库可以用于几乎所有对一致性有要求的系统中，如典型的银行系统。

但是，在网页应用中，一致性却不是显得那么重要因此，关系型数据库的最大特点在这里起是那么重要了。

相反地，关系型数据库为了维护一致性所付出的巨大代价就是其读写性能比较差，因此，必须用新的一种数据结构存储来代替关系数据库。

关系数据库的另一个特点就是其具有固定的表结构，因此，其扩展性极差，系统的升级，功能的增加，往往意味着数据结构巨大变动，这一点关系型数据库也难以应付，需要新的结构化数据存储。

于是，非关系型数据库应运而生，由于不可能用一种数据结构化存储应付所有的新的需求，因此，非关系型数据库严格上不是一种数据库，应该是一种数据结构化存储方法的集合。

但是，数据的持久存储，尤其是海量数据的持久存储，还是需要一种关系数据库的.

**非关系型数据库分类**

**键值(**[*Key-Value*](http://baike.baidu.com/item/Key-Value)**)存储**[**数据库**](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93)

这一类数据库主要会使用到一个[哈希表](http://baike.baidu.com/item/%E5%93%88%E5%B8%8C%E8%A1%A8)，这个表中有一个特定的键和一个指针指向特定的数据。Key/value模型对于IT系统来说的优势在于简单、易部署。但是如果[DBA](http://baike.baidu.com/item/DBA/3349)只对部分值进行查询或更新的时候，Key/value就显得效率低下了。[3]  举例如：Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB.

**列存储数据库。**

这部分数据库通常是用来应对分布式存储的海量数据。键仍然存在，但是它们的特点是指向了多个列。这些列是由列家族来安排的。如：Cassandra, HBase, Riak.

**文档型数据库**

文档型数据库的灵感是来自于Lotus Notes办公软件的，而且它同第一种键值存储相类似。该类型的数据模型是版本化的文档，半结构化的文档以特定的格式存储，比如JSON。文档型数据库可 以看作是键值数据库的升级版，允许之间嵌套键值。而且文档型数据库比键值数据库的查询效率更高。如：CouchDB, MongoDb. 国内也有文档型数据库SequoiaDB，已经开源。

**图形(Graph)数据库**

图形结构的数据库同其他行列以及刚性结构的SQL数据库不同，它是使用灵活的图形模型，并且能够扩展到多个服务器上。NoSQL数据库没有标准的查询语言(SQL)，因此进行数据库查询需要制定数据模型。许多NoSQL数据库都有REST式的数据接口或者查询API。[2]  如：Neo4J, InfoGrid, Infinite Graph.

因此，我们总结NoSQL数据库在以下的这几种情况下比较适用：1、数据模型比较简单；2、需要灵活性更强的IT系统；3、对数据库性能要求较高；4、不需要高度的数据一致性；5、对于给定key，比较容易映射复杂值的环境。