**案例简介**

大数据课程实验案例：网站用户行为分析，本案例涉及数据预处理、存储、查询和可视化分析等数据处理全流程所涉及的各种典型操作，涵盖Linux、MySQL、Hadoop、HBase、Hive、Sqoop、R、Eclipse等系统和软件的安装和使用方法。通过本案例，将有助于学生综合运用大数据课程知识以及各种工具软件，实现数据全流程操作。各个高校可以根据自己教学实际需求，对本案例进行补充完善。

**案例目的**

1. 熟悉Linux系统、MySQL、Hadoop、HBase、Hive、Sqoop、R、Eclipse等系统和软件的安装和使用；
2. 了解大数据处理的基本流程；
3. 熟悉数据预处理方法；
4. 熟悉在不同类型数据库之间进行数据相互导入导出；
5. 熟悉使用R语言进行可视化分析；
6. 熟悉使用Elipse编写Java程序操作HBase数据库。

**时间安排**

本案例可以作为大数据入门级课程结束后的“大作业”，或者可以作为学生暑期或寒假大数据实习实践基础案例，完成本案例预计耗时7天。

**预备知识**

需要案例使用者，已经学习过大数据相关课程，了解大数据相关技术的基本概念与原理，了解Windows操作系统、Linux操作系统、大数据处理架构Hadoop的关键技术及其基本原理、列族数据库HBase概念及其原理、数据仓库概念与原理、关系型数据库概念与原理、R语言概念与应用。

本案例提供了全部操作细节，包括每个命令和运行结果，所以，即使没有相关背景知识，也可以按照操作说明顺利完成全部实验。

**硬件要求**

本案例可以在单机上完成，也可以在集群环境下完成。

单机上完成本案例实验时，建议计算机硬件配置为：500GB以上硬盘，8GB以上内存。

**软件工具**

本案例所涉及的系统及软件

1. Linux系统（Ubuntu16.04或14.04或18.04）
2. MySQL（版本无要求）
3. Hadoop（2.7.1或2.7.3，不能用3.0及以上版本，因为Sqoop工具无法支持Hadoop3.0以上版本）
4. HBase（1.1.2或1.1.5，HBase版本需要和Hadoop版本兼容）
5. Hive（1.2.1，Hive需要和Hadoop版本兼容，不要安装Hive3.0以上版本）
6. Sqoop（必须用1.4.6，注意，Sqoop无法支持Hadoop3.0以上版本）
7. R（版本无要求）
8. Eclipse（版本无要求）

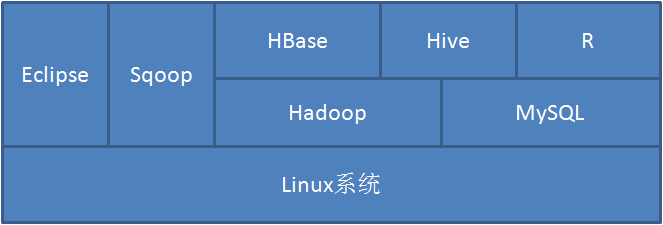


图 案例所涉及软件总体概览图

**数据集**

网站用户购物行为数据集2000万条记录。

**案例任务**

1. 安装Linux操作系统
2. 安装关系型数据库MySQL
3. 安装大数据处理框架Hadoop
4. 安装列族数据库HBase
5. 安装数据仓库Hive
6. 安装Sqoop
7. 安装R
8. 安装Eclipse
9. 对文本文件形式的原始数据集进行预处理
10. 把文本文件的数据集导入到数据仓库Hive中
11. 对数据仓库Hive中的数据进行查询分析
12. 使用Sqoop将数据从Hive导入MySQL
13. 使用Sqoop将数据从MySQL导入HBase
14. 使用HBase Java API把数据从本地导入到HBase中
15. 使用R对MySQL中的数据进行可视化分析

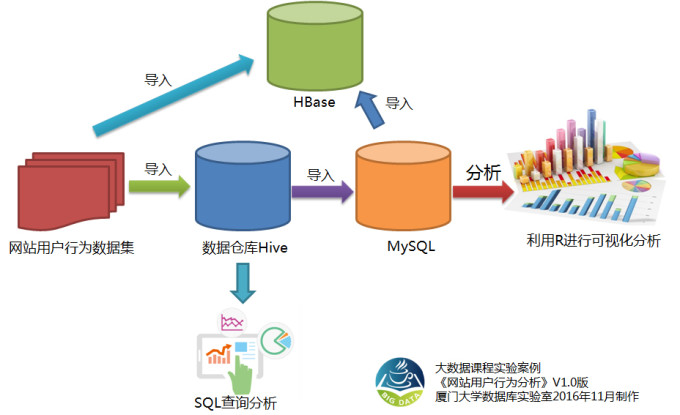


图  案例所涉及操作总体概览图

**实验步骤**

|  |
| --- |
| 步骤零：实验环境准备 |
| 步骤一：本地数据集上传到数据仓库Hive |
| 步骤二：Hive数据分析 |
| 步骤三：Hive、MySQL、HBase数据互导 |
| 步骤四：利用R进行数据可视化分析 |

每个实验步骤所需要的知识储备、训练技能和任务清单如下：

**步骤零：实验环境准备**

|  |  |
| --- | --- |
| 所需知识储备 | Windows操作系统、Linux操作系统、大数据处理架构Hadoop的关键技术及其基本原理、列族数据库HBase概念及其原理、数据仓库概念与原理、关系型数据库概念与原理 |
| 训练技能 | 双操作系统安装、虚拟机安装、Linux基本操作、Hadoop安装、HBase安装、Sqoop安装、Eclipse安装 |
| 任务清单 | 1. 安装Linux系统；2. 安装Hadoop；3. 安装MySQL；4. 安装HBase；5. 安装Hive；6. 安装Sqoop；7. 安装R；8. 安装Eclipse |

**步骤一：本地数据集上传到数据仓库Hive**

|  |  |
| --- | --- |
| 所需知识储备 | Linux系统基本命令、Hadoop项目结构、分布式文件系统HDFS概念及其基本原理、数据仓库概念及其基本原理、数据仓库Hive概念及其基本原理 |
| 训练技能 | Hadoop的安装与基本操作、HDFS的基本操作、Linux的安装与基本操作、数据仓库Hive的安装与基本操作、基本的数据预处理方法 |
| 任务清单 | 1. 安装Linux系统；2. 数据集下载与查看；3. 数据集预处理；4. 把数据集导入分布式文件系统HDFS中；5. 在数据仓库Hive上创建数据库 |

**步骤二：Hive数据分析**

|  |  |
| --- | --- |
| 所需知识储备 | 数据仓库Hive概念及其基本原理、SQL语句、数据库查询分析 |
| 训练技能 | 数据仓库Hive基本操作、创建数据库和表、使用SQL语句进行查询分析 |
| 任务清单 | 1. 启动Hadoop和Hive；2. 创建数据库和表；3. 简单查询分析；4. 查询条数统计分析；5. 关键字条件查询分析；6. 根据用户行为分析；7. 用户实时查询分析 |

**步骤三：Hive、MySQL、HBase数据互导**

|  |  |
| --- | --- |
| 所需知识储备 | 数据仓库Hive概念与基本原理、关系数据库概念与基本原理、SQL语句、列族数据库HBase概念与基本原理 |
| 训练技能 | 数据仓库Hive的基本操作、关系数据库MySQL的基本操作、Sqoop工具的使用方法、HBase API的Java编程、Eclipse开发工具使用方法 |
| 任务清单 | 1. Hive预操作；2. 使用Sqoop将数据从Hive导入MySQL；3. 使用Sqoop将数据从MySQL导入HBase；4. 使用HBase Java API把数据从本地导入到HBase中 |

**步骤四：利用R进行数据可视化分析**

|  |  |
| --- | --- |
| 所需知识储备 | 数据可视化、R语言 |
| 训练技能 | 利用R语言对MySQL数据库中的数据进行数据可视化分析、R的安装、相关可视化依赖包的安装与使用、各种可视化图表生成方法 |
| 任务清单 | 安装R语言包、安装可视化依赖包、柱状图可视化分析、散点图可视化分析、地图可视化分析 |

**所需知识储备**

Windows操作系统、Linux操作系统、大数据处理架构Hadoop的关键技术及其基本原理、列族数据库HBase概念及其原理、数据仓库概念与原理、关系型数据库概念与原理

**训练技能**

双操作系统安装、虚拟机安装、Linux基本操作、Hadoop安装、HBase安装、Sqoop安装、Eclipse安装

**任务清单**

1. 安装Linux系统
2. 安装Hadoop
3. 安装MySQL
4. 安装HBase
5. 安装Hive
6. 安装Sqoop
7. 安装R
8. 安装Eclipse

**系统和软件环境要求**

本案例的所有实验都在Linux操作系统下完成，需要涉及到以下软件（版本号仅供参考，可以使用不同版本）：

Linux: Ubuntu14.04

MySQL: 5.7.16

Hadoop: 2.7.1

HBase：1.1.2

Hive:1.2.1

Sqoop:1.4.6

R：3.2.3

Eclipse：3.8

**系统和软件的安装**

**Linux操作系统的安装**

本案例实验全部在Linux系统下开展，因此，必须要安装好Linux系统。

**Hadoop的安装**

本案例实验需要以Hadoop平台作为基础。

**MySQL的安装**

本案例实验需要把数据存入关系型数据库MySQL，同时，也需要安装MySQL为Hive提供元数据存储服务，因此，需要安装MySQL数据库。

**HBase的安装**

本实验需要把数据存入HBase，关于如何在Linux中安装HBase，完成HBase的安装。本教程把HBase安装在了“/usr/local/hbase”目录下，采用伪分布式配置，也就是HBase会使用HDFS来存储数据。

**Hive的安装**

本案例实验需要安装数据仓库Hive，完成Hive的安装，并且使用MySQL数据库保存Hive的元数据。本教程安装的是Hive2.1.0版本，安装目录是“/usr/local/hive”。

**Sqoop的安装**

本案例实验需要安装Sqoop，该工具支持在Hadoop和其他数据库之间进行数据互导操作。完成Sqoop的安装。本教程下载的是sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha.tar.gz，安装目录是“/usr/local/sqoop”。虽然这个sqoop是为hadoop2.0.4版本开发的，本教程Hadoop版本是2.7.1，但是，依然可以顺利使用。

**R的安装**

R的安装将在实验步骤四“利用R进行数据可视化分析”中再具体介绍。

**Eclipse的安装**

本案例实验需要采用Eclipse开发Java程序，利用Java API与HDFS进行交互”中的“在Ubuntu中安装Eclipse”部分有详细介绍。

**《大数据课程实验案例：网站用户行为分析—-步骤一:本地数据集上传到数据仓库Hive》**

**所需知识储备**

Linux系统基本命令、Hadoop项目结构、分布式文件系统HDFS概念及其基本原理、数据仓库概念及其基本原理、数据仓库Hive概念及其基本原理

**训练技能**

Hadoop的安装与基本操作、HDFS的基本操作、Linux的安装与基本操作、数据仓库Hive的安装与基本操作、基本的数据预处理方法

**任务清单**

1. 安装Linux系统
2. 数据集下载与查看
3. 数据集预处理
4. 把数据集导入分布式文件系统HDFS中
5. 在数据仓库Hive上创建数据库

**Linux系统的安装**

本实验全部在Linux系统下开展，因此，必须要安装好Linux系统。关于需要什么样的电脑硬件配置，以及如何安装Linux系统。

**实验数据集的下载**

本案例采用的数据集为user.zip，包含了一个大规模数据集raw\_user.csv（包含2000万条记录），和一个小数据集small\_user.csv（只包含30万条记录）。小数据集small\_user.csv是从大规模数据集raw\_user.csv中抽取的一小部分数据。之所以抽取出一少部分记录单独构成一个小数据集，是因为，在第一遍跑通整个实验流程时，会遇到各种错误，各种问题，先用小数据集测试，可以大量节约程序运行时间。等到第一次完整实验流程都顺利跑通以后，就可以最后用大规模数据集进行最后的测试。

下面，请登录Linux系统（本教程统一采用hadoop用户登录），并在Linux系统中打开浏览器（一般都是火狐Firefox浏览器）。如果在下载的时候，你没有修改文件保存路径，火狐浏览器会默认把文件保存在你的当前用户的下载目录下，因为本教程是采用hadoop用户名登录了Linux系统，所以，下载后的文件会被浏览器默认保存到”/home/hadoop/下载/”这目录下面。  
现在，请在Linux系统中打开一个终端（可以使用快捷键Ctrl+Alt+T），执行下面命令：

1. cd /home/hadoop/下载
2. ls

Shell 命令

通过上面命令，就进入到了user.zip文件所在的目录，并且可以看到有个user.zip文件。注意，如果你把user.zip下载到了其他目录，这里请进入到你自己的存放user.zip的目录。  
下面需要把user.zip进行解压缩，我们需要首先建立一个用于运行本案例的目录bigdatacase，请执行以下命令：

1. cd /usr/local
2. ls
3. sudo mkdir bigdatacase
4. //这里会提示你输入当前用户（本教程是hadoop用户名）的密码
5. //下面给hadoop用户赋予针对bigdatacase目录的各种操作权限
6. sudo chown -R hadoop:hadoop ./bigdatacase
7. cd bigdatacase
8. //下面创建一个dataset目录，用于保存数据集
9. mkdir dataset
10. //下面就可以解压缩user.zip文件
11. cd ~ //表示进入hadoop用户的目录
12. cd 下载
13. ls
14. unzip user.zip -d /usr/local/bigdatacase/dataset
15. cd /usr/local/bigdatacase/dataset
16. ls

Shell 命令

现在你就可以看到在dataset目录下有两个文件：raw\_user.csv和small\_user.csv。  
我们执行下面命令取出前面5条记录看一下：

1. head -5 raw\_user.csv

Shell 命令

可以看到，前5行记录如下：

user\_id,item\_id,behavior\_type,user\_geohash,item\_category,time

10001082,285259775,1,97lk14c,4076,2014-12-08 18

10001082,4368907,1,,5503,2014-12-12 12

10001082,4368907,1,,5503,2014-12-12 12

10001082,53616768,1,,9762,2014-12-02 15

可以看出，每行记录都包含5个字段，数据集中的字段及其含义如下：

user\_id（用户id）

item\_id(商品id)

behaviour\_type（包括浏览、收藏、加购物车、购买，对应取值分别是1、2、3、4）

user\_geohash(用户地理位置哈希值，有些记录中没有这个字段值，所以后面我们会用脚本做数据预处理时把这个字段全部删除)

item\_category（商品分类）

time（该记录产生时间）

**数据集的预处理**

**1.删除文件第一行记录，即字段名称**  
raw\_user和small\_user中的第一行都是字段名称，我们在文件中的数据导入到数据仓库Hive中时，不需要第一行字段名称，因此，这里在做数据预处理时，删除第一行

1. cd /usr/local/bigdatacase/dataset
2. //下面删除raw\_user中的第1行
3. sed -i '1d' raw\_user //1d表示删除第1行，同理，3d表示删除第3行，nd表示删除第n行
4. //下面删除small\_user中的第1行
5. sed -i '1d' small\_user
6. //下面再用head命令去查看文件的前5行记录，就看不到字段名称这一行了
7. head -5 raw\_user.csv
8. head -5 small\_user.csv

Shell 命令

接下来的操作中，我们都是用small\_user.csv这个小数据集进行操作，这样可以节省时间。等所有流程都跑通以后，你就可以使用大数据集raw\_user.csv去测试一遍了。

**2.对字段进行预处理**

下面对数据集进行一些预处理，包括为每行记录增加一个id字段（让记录具有唯一性）、增加一个省份字段（用来后续进行可视化分析），并且丢弃user\_geohash字段（后面分析不需要这个字段）。  
下面我们要建一个脚本文件pre\_deal.sh，请把这个脚本文件放在dataset目录下，和数据集small\_user.csv放在同一个目录下：

1. cd /usr/local/bigdatacase/dataset
2. vim pre\_deal.sh

Shell 命令

上面使用vim编辑器新建了一个pre\_deal.sh脚本文件，请在这个脚本文件中加入下面代码：

#!/bin/bash

#下面设置输入文件，把用户执行pre\_deal.sh命令时提供的第一个参数作为输入文件名称

infile=$1

#下面设置输出文件，把用户执行pre\_deal.sh命令时提供的第二个参数作为输出文件名称

outfile=$2

#注意！！最后的$infile > $outfile必须跟在}’这两个字符的后面

awk -F "," 'BEGIN{

srand();

id=0;

Province[0]="山东";Province[1]="山西";Province[2]="河南";Province[3]="河北";Province[4]="陕西";Province[5]="内蒙古";Province[6]="上海市";

Province[7]="北京市";Province[8]="重庆市";Province[9]="天津市";Province[10]="福建";Province[11]="广东";Province[12]="广西";Province[13]="云南";

Province[14]="浙江";Province[15]="贵州";Province[16]="新疆";Province[17]="西藏";Province[18]="江西";Province[19]="湖南";Province[20]="湖北";

Province[21]="黑龙江";Province[22]="吉林";Province[23]="辽宁"; Province[24]="江苏";Province[25]="甘肃";Province[26]="青海";Province[27]="四川";

Province[28]="安徽"; Province[29]="宁夏";Province[30]="海南";Province[31]="香港";Province[32]="澳门";Province[33]="台湾";

}

{

id=id+1;

value=int(rand()\*34);

print id"\t"$1"\t"$2"\t"$3"\t"$5"\t"substr($6,1,10)"\t"Province[value]

}' $infile > $outfile

上面的代码的基本形式是：

awk -F "," '处理逻辑' $infile > $outfile

使用awk可以逐行读取输入文件，并对逐行进行相应操作。其中，-F参数用于指出每行记录的不同字段之间用什么字符进行分割，这里是用逗号进行分割。处理逻辑代码需要用两个英文单引号引起来。 $infile是输入文件的名称，我们这里会输入raw\_user.csv，$outfile表示处理结束后输出的文件名称，我们后面会使用user\_table.txt作为输出文件名称。

在上面的pre\_deal.sh代码的处理逻辑部分，srand()用于生成随机数的种子，id是我们为数据集新增的一个字段，它是一个自增类型，每条记录增加1，这样可以保证每条记录具有唯一性。我们会为数据集新增一个省份字段，用来进行后面的数据可视化分析，为了给每条记录增加一个省份字段的值，这里，我们首先用Province[]数组用来保存全国各个省份信息，然后，在遍历数据集raw\_user.csv的时候，每当遍历到其中一条记录，使用value=int(rand()\*34)语句随机生成一个0-33的整数，作为Province省份值，然后从Province[]数组当中获取省份名称，增加到该条记录中。

substr($6,1,10)这个语句是为了截取时间字段time的年月日，方便后续存储为date格式。awk每次遍历到一条记录时，每条记录包含了6个字段，其中，第6个字段是时间字段，substr($6,1,10)语句就表示获取第6个字段的值，截取前10个字符，第6个字段是类似”2014-12-08 18″这样的字符串（也就是表示2014年12月8日18时），substr($6,1,10)截取后，就丢弃了小时，只保留了年月日。  
另外，在print id”\t”$1″\t”$2″\t”$3″\t”$5″\t”substr($6,1,10)”\t”Province[value]这行语句中，我们丢弃了每行记录的第4个字段，所以，没有出现$4。我们生成后的文件是“\t”进行分割，这样，后续我们去查看数据的时候，效果让人看上去更舒服，每个字段在排版的时候会对齐显示，如果用逗号分隔，显示效果就比较乱。

最后，保存pre\_deal.sh代码文件，退出vim编辑器。  
下面就可以执行pre\_deal.sh脚本文件，来对small\_user.csv进行数据预处理，命令如下：

1. cd /usr/local/bigdatacase/dataset
2. bash ./pre\_deal.sh small\_user.csv user\_table.txt

Shell 命令

可以使用head命令查看生成的user\_table.txt，不要直接打开，文件过大，会出错，下面查看前10行数据：

1. head -10 user\_table.txt

Shell 命令

可以得到如下结果：

1 10001082 285259775 1 4076 2014-12-08 广东

2 10001082 4368907 1 5503 2014-12-12 河南

3 10001082 4368907 1 5503 2014-12-12 甘肃

4 10001082 53616768 1 9762 2014-12-02 北京市

5 10001082 151466952 1 5232 2014-12-12 安徽

6 10001082 53616768 4 9762 2014-12-02 北京市

7 10001082 290088061 1 5503 2014-12-12 山东

8 10001082 298397524 1 10894 2014-12-12 福建

9 10001082 32104252 1 6513 2014-12-12 湖南

10 10001082 323339743 1 10894 2014-12-12 山东

**3.导入数据库**  
下面要把user\_table.txt中的数据最终导入到数据仓库Hive中。为了完成这个操作，我们会首先把user\_table.txt上传到分布式文件系统HDFS中，然后，在Hive中创建一个外部表，完成数据的导入。

**a.启动HDFS**  
HDFS是Hadoop的核心组件，因此，需要使用HDFS，必须安装Hadoop。这里假设你已经安装了Hadoop，本教程使用的是Hadoop2.7.1版本，安装目录是“/usr/local/hadoop”。

下面，请登录Linux系统，打开一个终端，执行下面命令启动Hadoop：

1. cd /usr/local/hadoop
2. ./sbin/start-all.sh

Shell 命令

然后，执行jps命令看一下当前运行的进程：

1. jps

Shell 命令

如果出现下面这些进程，说明Hadoop启动成功了。

3765 NodeManager

3639 ResourceManager

3800 Jps

3261 DataNode

3134 NameNode

3471 SecondaryNameNode

**b.把user\_table.txt上传到HDFS中**  
现在，我们要把Linux本地文件系统中的user\_table.txt上传到分布式文件系统HDFS中，存放在HDFS中的“/bigdatacase/dataset”目录下。  
首先，请执行下面命令，在HDFS的根目录下面创建一个新的目录bigdatacase，并在这个目录下创建一个子目录dataset，如下：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -mkdir -p /bigdatacase/dataset

Shell 命令

然后，把Linux本地文件系统中的user\_table.txt上传到分布式文件系统HDFS的“/bigdatacase/dataset”目录下，命令如下：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -put /usr/local/bigdatacase/dataset/user\_table.txt /bigdatacase/dataset

Shell 命令

下面可以查看一下HDFS中的user\_table.txt的前10条记录，命令如下：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -cat /bigdatacase/dataset/user\_table.txt | head -10

Shell 命令

**c.在Hive上创建数据库**  
关于什么是数据仓库Hive？Hive的运行基本原理是什么？如何开展Hive简单编程实践？   
本案例教程需要安装数据仓库Hive，这里假设你已经完成了Hive的安装，并且使用MySQL数据库保存Hive的元数据。本教程安装的是Hive2.1.0版本，安装目录是“/usr/local/hive”。  
下面，请在Linux系统中，再新建一个终端（可以在刚才已经建好的终端界面的左上角，点击“终端”菜单，在弹出的子菜单中选择“新建终端”）。因为需要借助于MySQL保存Hive的元数据，所以，请首先启动MySQL数据库：

1. service mysql start //可以在Linux的任何目录下执行该命令

Shell 命令

由于Hive是基于Hadoop的数据仓库，使用HiveQL语言撰写的查询语句，最终都会被Hive自动解析成MapReduce任务由Hadoop去具体执行，因此，需要启动Hadoop，然后再启动Hive。由于前面我们已经启动了Hadoop，所以，这里不需要再次启动Hadoop。下面，在这个新的终端中执行下面命令进入Hive：

1. cd /usr/local/hive
2. ./bin/hive //启动Hive

Shell 命令

启动成功以后，就进入了“hive>”命令提示符状态，可以输入类似SQL语句的HiveQL语句。  
下面，我们要在Hive中创建一个数据库dblab，命令如下：

1. hive> create database dblab;
2. hive> use dblab;

hive

**d.创建外部表**  
关于数据仓库Hive的内部表和外部表的区别，本教程采用外部表方式。  
这里我们要在数据库dblab中创建一个外部表bigdata\_user，它包含字段（id, uid, item\_id, behavior\_type, item\_category, date, province），请在hive命令提示符下输入如下命令：

1. hive> CREATE EXTERNAL TABLE dblab.bigdata\_user(id INT,uid STRING,item\_id STRING,behavior\_type INT,item\_category STRING,visit\_date DATE,province STRING) COMMENT 'Welcome to xmu dblab!' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE LOCATION '/bigdatacase/dataset';

hive

对于这条语句中LOCATION参数的含义如果有疑问，

**e.查询数据**  
上面已经成功把HDFS中的“/bigdatacase/dataset”目录下的数据加载到了数据仓库Hive中，我们现在可以使用下面命令查询一下：

1. hive> select \* from bigdata\_user limit 10;
2. hive> select behavior\_type from bigdata\_user limit 10;

hive

步骤一的实验顺利结束。可以继续访问下一个步骤：

**《大数据课程实验案例：网站用户行为分析—-步骤二：Hive数据分析》**  
本文介绍大数据课程实验案例“网站用户行为分析”的第二个步骤，Hive数据分析。在实践本步骤之前，请先完成该实验案例的第一个步骤。这里假设你已经完成了前面的第一个步骤。

**所需知识储备**

数据仓库Hive概念及其基本原理、SQL语句、数据库查询分析

**训练技能**

数据仓库Hive基本操作、创建数据库和表、使用SQL语句进行查询分析

**任务清单**

1. 启动Hadoop和Hive
2. 创建数据库和表
3. 简单查询分析
4. 查询条数统计分析
5. 关键字条件查询分析
6. 根据用户行为分析
7. 用户实时查询分析

**一、操作Hive**

请登录Linux系统（本教程统一采用hadoop用户名登录系统），然后，打开一个终端（可以按快捷键Ctrl+Alt+T）。  
本教程中，Hadoop的安装目录是“/usr/local/hadoop”，Hive的安装目录是“/usr/local/hive”。  
因为需要借助于MySQL保存Hive的元数据，所以，请首先启动MySQL数据库，请在终端中输入下面命令：

1. service mysql start //可以在Linux的任何目录下执行该命令

Shell 命令

由于Hive是基于Hadoop的数据仓库，使用HiveQL语言撰写的查询语句，最终都会被Hive自动解析成MapReduce任务由Hadoop去具体执行，因此，需要启动Hadoop，然后再启动Hive。

请执行下面命令启动Hadoop（如果你已经启动了Hadoop就不用再次启动了）：

1. cd /usr/local/hadoop
2. ./sbin/start-all.sh

Shell 命令

然后，执行jps命令看一下当前运行的进程：

1. jps

Shell 命令

如果出现下面这些进程，说明Hadoop启动成功了。

3765 NodeManager

3639 ResourceManager

3800 Jps

3261 DataNode

3134 NameNode

3471 SecondaryNameNode

下面，继续执行下面命令启动进入Hive：

1. cd /usr/local/hive
2. ./bin/hive //启动Hive

Shell 命令

通过上述过程，我们就完成了MySQL、Hadoop和Hive三者的启动。  
启动成功以后，就进入了“hive>”命令提示符状态，可以输入类似SQL语句的HiveQL语句。

然后，在“hive>”命令提示符状态下执行下面命令：

1. hive> use dblab; *//使用dblab数据库*
2. hive> show tables; *//显示数据库中所有表。*
3. hive> show create table bigdata\_user; *//查看bigdata\_user表的各种属性；*

hive

执行结果如下：

OK

CREATE EXTERNAL TABLE `bigdata\_user`(

`id` int,

`uid` string,

`item\_id` string,

`behavior\_type` int,

`item\_category` string,

`visit\_date` date,

`province` string)

COMMENT 'Welcome to xmu dblab!'

ROW FORMAT SERDE

'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'

WITH SERDEPROPERTIES (

'field.delim'='\t',

'serialization.format'='\t')

STORED AS INPUTFORMAT

'org.apache.hadoop.mapred.TextInputFormat'

OUTPUTFORMAT

'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

LOCATION

'hdfs://localhost:9000/bigdatacase/dataset'

TBLPROPERTIES (

'numFiles'='1',

'totalSize'='15590786',

'transient\_lastDdlTime'='1480217306')

Time taken: 0.715 seconds, Fetched: 24 row(s)

可以执行下面命令查看表的简单结构：

1. hive> desc bigdata\_user;

hive

执行结果如下：

OK

id int

uid string

item\_id string

behavior\_type int

item\_category string

visit\_date date

province string

Time taken: 0.267 seconds, Fetched: 7 row(s)

**二、简单查询分析**

先测试一下简单的指令：

1. hive> select behavior\_type from bigdata\_user limit 10;*//查看前10位用户对商品的行为*

hive

执行结果如下：

OK

1

1

1

1

1

4

1

1

1

1

Time taken: 2.561 seconds, Fetched: 10 row(s)

如果要查出每位用户购买商品时的多种信息，输出语句格式为 select 列1，列2，….，列n from 表名；  
比如我们现在查询前20位用户购买商品时的时间和商品的种类

1. hive> select visit\_date,item\_category from bigdata\_user limit 20;

hive

执行结果如下：

OK

2014-12-08 4076

2014-12-12 5503

2014-12-12 5503

2014-12-02 9762

2014-12-12 5232

2014-12-02 9762

2014-12-12 5503

2014-12-12 10894

2014-12-12 6513

2014-12-12 10894

2014-12-12 2825

2014-11-28 2825

2014-12-15 3200

2014-12-03 10576

2014-11-20 10576

2014-12-13 10576

2014-12-08 10576

2014-12-14 7079

2014-12-02 6669

2014-12-12 5232

Time taken: 0.401 seconds, Fetched: 20 row(s)

有时我们在表中查询可以利用嵌套语句，如果列名太复杂可以设置该列的别名，以简化我们操作的难度，以下我们可以举个例子：

1. hive> select e.bh, e.it from (select behavior\_type as bh, item\_category as it from bigdata\_user) as e limit 20;

hive

执行结果如下：

OK

1 4076

1 5503

1 5503

1 9762

1 5232

4 9762

1 5503

1 10894

1 6513

1 10894

1 2825

1 2825

1 3200

1 10576

1 10576

1 10576

1 10576

1 7079

1 6669

1 5232

Time taken: 0.374 seconds, Fetched: 20 row(s)

这里简单的做个讲解，behavior\_type as bh ,item\_category as it就是把behavior\_type 设置别名 bh ,item\_category 设置别名 it，FROM的括号里的内容我们也设置了别名e，这样调用时用e.bh,e.it,可以简化代码。

**三、查询条数统计分析**

经过简单的查询后我们同样也可以在select后加入更多的条件对表进行查询,下面可以用函数来查找我们想要的内容。  
**(1)用聚合函数count()计算出表内有多少条行数据**

1. hive> select count(\*) from bigdata\_user;*//用聚合函数count()计算出表内有多少条行数据*

hive

执行结果如下：

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = hadoop\_20161127192506\_cb00d4a1-b9e3-426e-a2be-d971a61f686b

Total jobs = 1

Launching Job 1 out of 1

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 19:25:10,555 Stage-1 map = 0%, reduce = 0%

2016-11-27 19:25:12,606 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_local1072810798\_0001

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 62395912 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

300000

Time taken: 6.418 seconds, Fetched: 1 row(s)

我们可以看到，得出的结果为OK下的那个数字300000（因为我们的small\_user.csv中包含了300000条记录，导入到Hive中）。  
**(2)在函数内部加上distinct，查出uid不重复的数据有多少条**  
下面继续执行操作：

1. hive> select count(distinct uid) from bigdata\_user;*//在函数内部加上distinct，查出uid不重复的数据有多少条*

hive

执行结果如下：

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = hadoop\_20161127192506\_cb00d4a1-b9e3-426e-a2be-d971a61f686b

Total jobs = 1

Launching Job 1 out of 1

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 19:25:10,555 Stage-1 map = 0%, reduce = 0%

2016-11-27 19:25:12,606 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_local1072810798\_0001

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 62395912 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

300000

Time taken: 6.418 seconds, Fetched: 1 row(s)

hive> select count(distinct uid) from bigdata\_user;

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = hadoop\_20161127193437\_9e0a95ac-b5f2-44b5-acf8-97c90fc1e8c8

Total jobs = 1

Launching Job 1 out of 1

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 19:34:40,045 Stage-1 map = 0%, reduce = 0%

2016-11-27 19:34:42,074 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_local2080512810\_0002

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 93577484 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

270

Time taken: 4.689 seconds, Fetched: 1 row(s)

\*\* (3)查询不重复的数据有多少条(为了排除客户刷单情况) \*\*

1. hive>select count(\*) from (select uid,item\_id,behavior\_type,item\_category,visit\_date,province from bigdata\_user group by uid,item\_id,behavior\_type,item\_category,visit\_date,province having count(\*)=1)a;

hive

执行结果如下：

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = hadoop\_20161127195152\_81d536c5-d492-4b59-a7db-4352717c8a70

Total jobs = 2

Launching Job 1 out of 2

Number of reduce tasks not specified. Estimated from input data size: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 19:51:55,380 Stage-1 map = 0%, reduce = 0%

2016-11-27 19:52:03,401 Stage-1 map = 100%, reduce = 0%

2016-11-27 19:52:06,413 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_local15333446\_0003

Launching Job 2 out of 2

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 19:52:08,283 Stage-2 map = 100%, reduce = 100%

Ended Job = job\_local2040099087\_0004

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 124759056 HDFS Write: 0 SUCCESS

Stage-Stage-2: HDFS Read: 124759056 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

284183

Time taken: 16.05 seconds, Fetched: 1 row(s)

可以看出，排除掉重复信息以后，只有284183条记录。  
注意：嵌套语句最好取别名，就是上面的a，否则很容易出现如下错误.  
qq%e6%88%aa%e5%9b%be20161115160822

**四．关键字条件查询分析**

1.以关键字的存在区间为条件的查询  
使用where可以缩小查询分析的范围和精确度，下面用实例来测试一下。  
(1)查询2014年12月10日到2014年12月13日有多少人浏览了商品

1. hive>select count(\*) from bigdata\_user where behavior\_type='1' and visit\_date<'2014-12-13' and visit\_date>'2014-12-10';

hive

执行结果如下：

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = hadoop\_20161127200005\_fb5120dc-acbb-43f3-97aa-ee766ead98e5

Total jobs = 1

Launching Job 1 out of 1

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 20:00:07,362 Stage-1 map = 0%, reduce = 0%

2016-11-27 20:00:08,376 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_local1606042318\_0005

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 155940628 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

26329

Time taken: 3.058 seconds, Fetched: 1 row(s)

（2）以月的第n天为统计单位，依次显示第n天网站卖出去的商品的个数

1. hive> select count(distinct uid), day(visit\_date) from bigdata\_user where behavior\_type='4' group by day(visit\_date);

hive

执行结果如下：

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = hadoop\_20161127200220\_738d202d-242c-415d-a986-f76faaff791a

Total jobs = 1

Launching Job 1 out of 1

Number of reduce tasks not specified. Estimated from input data size: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 20:02:22,945 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_local727434228\_0006

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 187122200 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

37 1

48 2

42 3

38 4

42 5

33 6

42 7

36 8

34 9

40 10

43 11

98 12

39 13

43 14

42 15

44 16

42 17

66 18

38 19

50 20

33 21

34 22

32 23

47 24

34 25

31 26

30 27

34 28

39 29

38 30

Time taken: 2.378 seconds, Fetched: 30 row(s)

2.关键字赋予给定值为条件，对其他数据进行分析  
取给定时间和给定地点，求当天发出到该地点的货物的数量

1. hive> select count(\*) from bigdata\_user where province='江西' and visit\_date='2014-12-12' and behavior\_type='4';

hive

执行结果如下：

WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.

Query ID = hadoop\_20161127200520\_4cdc4b74-5dbe-48d9-8e39-bdfaff90ff2e

Total jobs = 1

Launching Job 1 out of 1

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2016-11-27 20:05:22,874 Stage-1 map = 0%, reduce = 0%

2016-11-27 20:05:23,879 Stage-1 map = 100%, reduce = 100%

Ended Job = job\_local1743806040\_0007

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 218303772 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

7

Time taken: 3.421 seconds, Fetched: 1 row(s)

**五．根据用户行为分析**

从现在开始，我们只给出查询语句，将不再给出执行结果。  
1．查询一件商品在某天的购买比例或浏览比例

1. hive> select count(\*) from bigdata\_user where visit\_date='2014-12-11'and behavior\_type='4';*//查询有多少用户在2014-12-11购买了商品*

hive

1. hive> select count(\*) from bigdata\_user where visit\_date ='2014-12-11';*//查询有多少用户在2014-12-11点击了该店*

hive

根据上面语句得到购买数量和点击数量，两个数相除即可得出当天该商品的购买率。  
2.查询某个用户在某一天点击网站占该天所有点击行为的比例（点击行为包括浏览，加入购物车，收藏，购买）

1. hive> select count(\*) from bigdata\_user where uid=10001082 and visit\_date='2014-12-12';*//查询用户10001082在2014-12-12点击网站的次数*

hive

1. hive> select count(\*) from bigdata\_user where visit\_date='2014-12-12';*//查询所有用户在这一天点击该网站的次数*

hive

上面两条语句的结果相除，就得到了要要求的比例。  
3.给定购买商品的数量范围，查询某一天在该网站的购买该数量商品的用户id

1. hive> select uid from bigdata\_user where behavior\_type='4' and visit\_date='2014-12-12' group by uid having count(behavior\_type='4')>5;*//查询某一天在该网站购买商品超过5次的用户id*

hive

**六.用户实时查询分析**

某个地区的用户当天浏览网站的次数

1. hive> create table scan(province STRING,scan INT) COMMENT 'This is the search of bigdataday' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;*//创建新的数据表进行存储*
2. hive> insert overwrite table scan select province,count(behavior\_type) from bigdata\_user where behavior\_type='1' group by province;*//导入数据*
3. hive> select \* from scan;*//显示结果*

hive

执行结果如下：

上海市 8364

云南 8454

内蒙古 8172

北京市 8258

台湾 8382

吉林 8272

四川 8359

天津市 8478

宁夏 8205

安徽 8205

山东 8236

山西 8503

广东 8228

广西 8358

新疆 8316

江苏 8226

江西 8403

河北 8363

河南 8382

浙江 8310

海南 8391

湖北 8183

湖南 8368

澳门 8264

甘肃 8415

福建 8270

西藏 8347

贵州 8505

辽宁 8292

重庆市 8506

陕西 8379

青海 8427

香港 8386

黑龙江 8309

Time taken: 0.248 seconds, Fetched: 34 row(s)

到这里，Hive数据分析实验顺利结束。

**《大数据课程实验案例：网站用户行为分析—-步骤三：Hive、MySQL、HBase数据互导》**  
本教程介绍大数据课程实验案例“网站用户行为分析”的第三个步骤，Hive、MySQL、HBase数据互导。在实践本步骤之前，请先完成该实验案例的第一个步骤，和第二个步骤。这里假设你已经完成了前面的这两个步骤。

**所需知识储备**

数据仓库Hive概念与基本原理、关系数据库概念与基本原理、SQL语句、列族数据库HBase概念与基本原理

**训练技能**

数据仓库Hive的基本操作、关系数据库MySQL的基本操作、Sqoop工具的使用方法、HBase API的Java编程、Eclipse开发工具使用方法

**任务清单**

1. Hive预操作
2. 使用Sqoop将数据从Hive导入MySQL
3. 使用Sqoop将数据从MySQL导入HBase
4. 使用HBase Java API把数据从本地导入到HBase中

**一、准备工作**

本教程需要安装Hive、MySQL、HBase和Sqoop。在前面的第一个步骤中，我们在安装Hive的时候就已经一起安装了MySQL（因为我们采用MySQL来存储Hive的元数据），所以，现在你只需要再安装HBase和Sqoop。  
（1）完成HBase的安装。本教程把HBase安装在了“/usr/local/hbase”目录下，采用伪分布式配置，也就是HBase会使用HDFS来存储数据。  
（2）完成Sqoop的安装。本教程下载的是sqoop-1.4.6.bin\_\_hadoop-2.0.4-alpha.tar.gz，安装目录是“/usr/local/sqoop”。虽然这个sqoop是为hadoop2.0.4版本开发的，本教程Hadoop版本是2.7.1，但是，依然可以顺利使用。

**二、Hive预操作**

如果你还没有启动Hive，请首先启动Hive。  
请登录Linux系统（本教程统一采用hadoop用户名登录系统），然后，打开一个终端（可以按快捷键Ctrl+Alt+T）。  
本教程中，Hadoop的安装目录是“/usr/local/hadoop”，Hive的安装目录是“/usr/local/hive”。  
因为需要借助于MySQL保存Hive的元数据，所以，请首先启动MySQL数据库，请在终端中输入下面命令：

1. service mysql start //可以在Linux的任何目录下执行该命令

Shell 命令

由于Hive是基于Hadoop的数据仓库，使用HiveQL语言撰写的查询语句，最终都会被Hive自动解析成MapReduce任务由Hadoop去具体执行，因此，需要启动Hadoop，然后再启动Hive。

请执行下面命令启动Hadoop（如果你已经启动了Hadoop就不用再次启动了）：

1. cd /usr/local/hadoop
2. ./sbin/start-all.sh

Shell 命令

然后，执行jps命令看一下当前运行的进程：

1. jps

Shell 命令

如果出现下面这些进程，说明Hadoop启动成功了。

3765 NodeManager

3639 ResourceManager

3800 Jps

3261 DataNode

3134 NameNode

3471 SecondaryNameNode

下面，继续执行下面命令启动进入Hive：

1. cd /usr/local/hive
2. ./bin/hive //启动Hive

Shell 命令

通过上述过程，我们就完成了MySQL、Hadoop和Hive三者的启动。  
启动成功以后，就进入了“hive>”命令提示符状态，可以输入类似SQL语句的HiveQL语句。

然后，在“hive>”命令提示符状态下执行下面命令：

1、创建临时表user\_action

1. hive> create table dblab.user\_action(id STRING,uid STRING, item\_id STRING, behavior\_type STRING, item\_category STRING, visit\_date DATE, province STRING) COMMENT 'Welcome to XMU dblab! ' ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE;

hive

这个命令执行完以后，Hive会自动在HDFS文件系统中创建对应的数据文件“/user/hive/warehouse/dblab.db/user\_action”。  
我们可以新建一个终端，执行命令查看一下，确认这个数据文件在HDFS中确实被创建了，请在新建的终端中执行下面命令：

1. cd /usr/local/hadoop
2. ./bin/hdfs dfs -ls /user/hive/warehouse/dblab.db/user\_action

Shell 命令

可以看到如下结果：

-rwxr-xr-x 1 hadoop supergroup 15590786 2016-11-27 21:57 /user/hive/warehouse/dblab.db/user\_action/000000\_0

这说明，这个数据文件在HDFS中确实被创建了。注意，这个HDFS中的数据文件，在我们后面的“使用HBase Java API把数据从本地导入到HBase中”操作中会使用到。  
2、将bigdata\_user表中的数据插入到user\_action(执行时间：10秒左右)  
在第二个步骤中，我们已经在Hive中的dblab数据库中创建了一个外部表bigdata\_user。下面把dblab.bigdata\_user数据插入到dblab.user\_action表中，命令如下：

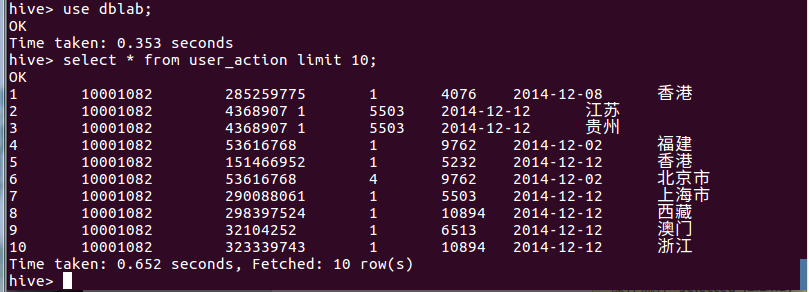
1. hive> INSERT OVERWRITE TABLE dblab.user\_action select \* from dblab.bigdata\_user;

hive

请执行下面命令查询上面的插入命令是否成功执行：

1. hive>select \* from user\_action limit 10;

hive

user\_action-灌入数据成功

**三、使用Sqoop将数据从Hive导入MySQL**

1、启动Hadoop集群、MySQL服务  
前面我们已经启动了Hadoop集群和MySQL服务。这里请确认已经按照前面操作启动成功。

2、将前面生成的临时表数据从Hive导入到 MySQL 中，包含如下四个步骤。  
(1)登录 MySQL  
请在Linux系统中新建一个终端，执行下面命令：

1. mysql –u root –p

Shell 命令

为了简化操作，本教程直接使用root用户登录MySQL数据库，但是，在实际应用中，建议在MySQL中再另外创建一个用户。  
执行上面命令以后，就进入了“mysql>”命令提示符状态。  
(2)创建数据库

1. mysql> show databases; *#显示所有数据库*
2. mysql> create database dblab; *#创建dblab数据库*
3. mysql> use dblab; *#使用数据库*

mysql

注意：请使用下面命令查看数据库的编码：

1. mysql>show variables like "char%";

mysql

会显示类似下面的结果：

+--------------------------+----------------------------+

| Variable\_name | Value |

+--------------------------+----------------------------+

| character\_set\_client | utf8 |

| character\_set\_connection | utf8 |

| character\_set\_database | latin1 |

| character\_set\_filesystem | binary |

| character\_set\_results | utf8 |

| character\_set\_server | latin1 |

| character\_set\_system | utf8 |

| character\_sets\_dir | /usr/share/mysql/charsets/ |

+--------------------------+----------------------------+

8 rows in set (0.00 sec)

请确认当前编码为utf8，否则无法导入中文，下面是笔者电脑上修改了编码格式后的结果：

+--------------------------+----------------------------+

| Variable\_name | Value |

+--------------------------+----------------------------+

| character\_set\_client | utf8 |

| character\_set\_connection | utf8 |

| character\_set\_database | utf8 |

| character\_set\_filesystem | binary |

| character\_set\_results | utf8 |

| character\_set\_server | utf8 |

| character\_set\_system | utf8 |

| character\_sets\_dir | /usr/share/mysql/charsets/ |

+--------------------------+----------------------------+

8 rows in set (0.00 sec)

(3)创建表  
下面在MySQL的数据库dblab中创建一个新表user\_action，并设置其编码为utf-8：

1. mysql> CREATE TABLE `dblab`.`user\_action` (`id` varchar(50),`uid` varchar(50),`item\_id` varchar(50),`behavior\_type` varchar(10),`item\_category` varchar(50), `visit\_date` DATE,`province` varchar(20)) ENGINE=InnoDB DEFAULT CHARSET=utf8;

mysql

提示：语句中的引号是反引号`，不是单引号’。  
创建成功后，输入下面命令退出MySQL：

1. mysql> exit

mysql

(4)导入数据(执行时间：20秒左右)  
注意，刚才已经退出MySQL，回到了Shell命令提示符状态。下面就可以执行数据导入操作，

1. cd /usr/local/sqoop
2. ./bin/sqoop export --connect jdbc:mysql://localhost:3306/dblab --username root --password hadoop --table user\_action --export-dir '/user/hive/warehouse/dblab.db/user\_action' --fields-terminated-by '\t'; *#导入命令*

Shell 命令

字段解释：

./bin/sqoop export ##表示数据从 hive 复制到 mysql 中

--connect jdbc:mysql://localhost:3306/dblab

--username root #mysql登陆用户名

--password hadoop #登录密码

--table user\_action #mysql 中的表，即将被导入的表名称

--export-dir '/user/hive/warehouse/dblab.db/user\_action ' #hive 中被导出的文件

--fields-terminated-by '\t' #Hive 中被导出的文件字段的分隔符

16/11/28 09:18:12 INFO mapreduce.Job: Job job\_local1006738657\_0001 completed successfully

16/11/28 09:18:12 INFO mapreduce.Job: Counters: 20

File System Counters

FILE: Number of bytes read=72216458

FILE: Number of bytes written=73973600

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=38989562

HDFS: Number of bytes written=0

HDFS: Number of read operations=78

HDFS: Number of large read operations=0

HDFS: Number of write operations=0

Map-Reduce Framework

Map input records=300000

Map output records=300000

Input split bytes=696

Spilled Records=0

Failed Shuffles=0

Merged Map outputs=0

GC time elapsed (ms)=145

Total committed heap usage (bytes)=511180800

File Input Format Counters

Bytes Read=0

File Output Format Counters

Bytes Written=0

16/11/28 09:18:12 INFO mapreduce.ExportJobBase: Transferred 37.1833 MB in 29.2772 seconds (1.27 MB/sec)

16/11/28 09:18:12 INFO mapreduce.ExportJobBase: Exported 300000 records.

3、查看MySQL中user\_action表数据  
下面需要再次启动MySQL，进入“mysql>”命令提示符状态：

1. mysql -u root -p

Shell 命令

会提示你输入MySQL的root用户的密码，本教程中安装的MySQL数据库的root用户的密码是hadoop。  
然后执行下面命令查询user\_action表中的数据：

1. mysql> use dblab;
2. mysql> select \* from user\_action limit 10;

mysql

会得到类似下面的查询结果：

+--------+-----------+-----------+---------------+---------------+------------+-----------+

| id | uid | item\_id | behavior\_type | item\_category | visit\_date | province |

+--------+-----------+-----------+---------------+---------------+------------+-----------+

| 225653 | 102865660 | 164310319 | 1 | 5027 | 2014-12-08 | 香港 |

| 225654 | 102865660 | 72511722 | 1 | 1121 | 2014-12-13 | 天津市 |

| 225655 | 102865660 | 334372932 | 1 | 5027 | 2014-11-30 | 江苏 |

| 225656 | 102865660 | 323237439 | 1 | 5027 | 2014-12-02 | 广东 |

| 225657 | 102865660 | 323237439 | 1 | 5027 | 2014-12-07 | 山西 |

| 225658 | 102865660 | 34102362 | 1 | 1863 | 2014-12-13 | 内蒙古 |

| 225659 | 102865660 | 373499226 | 1 | 12388 | 2014-11-26 | 湖北 |

| 225660 | 102865660 | 271583890 | 1 | 5027 | 2014-12-06 | 山西 |

| 225661 | 102865660 | 384764083 | 1 | 5399 | 2014-11-26 | 澳门 |

| 225662 | 102865660 | 139671483 | 1 | 5027 | 2014-12-03 | 广东 |

+--------+-----------+-----------+---------------+---------------+------------+-----------+

10 rows in set (0.00 sec)

从Hive导入数据到MySQL中，成功！

**四、使用Sqoop将数据从MySQL导入HBase**

1、启动Hadoop集群、MySQL服务、HBase服务  
之前我们已经启动了Hadoop集群、MySQL服务，这里请确认已经按照前面操作启动成功。这里我们再启动HBase服务。本教程中，HBase的安装目录是“/usr/local/hbase”，而且本教程中，HBase配置为使用HDFS存储数据。  
请新建一个终端，执行下面命令：

1. cd /usr/local/hbase
2. ./bin/start-hbase.sh

Shell 命令

2、启动HBase shell

1. cd /usr/local/hbase
2. ./bin/hbase shell

Shell 命令

启动成功后，就进入了“hbase>”命令提示符状态。  
3、创建表user\_action

1. hbase> create 'user\_action', { NAME => 'f1', VERSIONS => 5}

hbase

上面命令在HBase中创建了一个user\_action表，这个表中有一个列族f1（你愿意把列族名称取为其他名称也可以，比如列族名称为userinfo），历史版本保留数量为5。  
4、导入数据(执行时间：30秒左右)  
下面新建一个终端，执行下面命令导入数据：

1. cd /usr/local/sqoop
2. ./bin/sqoop import --connect jdbc:mysql://localhost:3306/dblab --username root --password hadoop --table user\_action --hbase-table user\_action --column-family f1 --hbase-row-key id --hbase-create-table -m 1

Shell 命令

注意：IP部分改为本机IP地址或localhost。同时，HBase只支持十六进制存储中文。  
命令解释如下：

./bin/sqoop import --connect jdbc:mysql://localhost:3306/dblab

--username root

--password hadoop

--table user\_action

--hbase-table user\_action #HBase中表名称

--column-family f1 #列簇名称

--hbase-row-key id #HBase 行键

--hbase-create-table #是否在不存在情况下创建表

-m 1 #启动 Map 数量

执行上面的sqoop import命令以后，会得到类似下面的结果（省略了很多非重要的屏幕信息）：

16/11/28 10:03:40 INFO mapreduce.Job: Counters: 20

File System Counters

FILE: Number of bytes read=39428226

FILE: Number of bytes written=40095167

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=0

HDFS: Number of bytes written=0

HDFS: Number of read operations=0

HDFS: Number of large read operations=0

HDFS: Number of write operations=0

Map-Reduce Framework

Map input records=300000

Map output records=300000

Input split bytes=87

Spilled Records=0

Failed Shuffles=0

Merged Map outputs=0

GC time elapsed (ms)=1018

Total committed heap usage (bytes)=127795200

File Input Format Counters

Bytes Read=0

File Output Format Counters

Bytes Written=0

16/11/28 10:03:40 INFO mapreduce.ImportJobBase: Transferred 0 bytes in 69.7527 seconds (0 bytes/sec)

16/11/28 10:03:40 INFO mapreduce.ImportJobBase: Retrieved 300000 records.

5、查看HBase中user\_action表数据  
现在，再次切换到HBase Shell运行的那个终端窗口，在“hbase>”命令提示符下，执行下面命令查询刚才导入的数据：

1. habse> scan 'user\_action',{LIMIT=>10} *#只查询前面10行*

hbase

因为有30万条记录，所以，我们用LIMIT只查询前10行的记录，会得到类似下面的结果：

1 column=f1:behavior\_type, timestamp=1480298573684, value=1

1 column=f1:item\_category, timestamp=1480298573684, value=4076

1 column=f1:item\_id, timestamp=1480298573684, value=285259775

1 column=f1:province, timestamp=1480298573684, value=\xE5\xB9\xBF\xE4\xB8\x9C

1 column=f1:uid, timestamp=1480298573684, value=10001082

1 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-08

10 column=f1:behavior\_type, timestamp=1480298573684, value=1

10 column=f1:item\_category, timestamp=1480298573684, value=10894

10 column=f1:item\_id, timestamp=1480298573684, value=323339743

10 column=f1:province, timestamp=1480298573684, value=\xE5\xB1\xB1\xE4\xB8\x9C

10 column=f1:uid, timestamp=1480298573684, value=10001082

10 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-12

100 column=f1:behavior\_type, timestamp=1480298573684, value=1

100 column=f1:item\_category, timestamp=1480298573684, value=10576

100 column=f1:item\_id, timestamp=1480298573684, value=275221686

100 column=f1:province, timestamp=1480298573684, value=\xE6\xB9\x96\xE5\x8C\x97

100 column=f1:uid, timestamp=1480298573684, value=10001082

100 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-02

1000 column=f1:behavior\_type, timestamp=1480298573684, value=1

1000 column=f1:item\_category, timestamp=1480298573684, value=3381

1000 column=f1:item\_id, timestamp=1480298573684, value=168463559

1000 column=f1:province, timestamp=1480298573684, value=\xE5\xB1\xB1\xE8\xA5\xBF

1000 column=f1:uid, timestamp=1480298573684, value=100068031

1000 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-02

10000 column=f1:behavior\_type, timestamp=1480298575888, value=1

10000 column=f1:item\_category, timestamp=1480298575888, value=12488

10000 column=f1:item\_id, timestamp=1480298575888, value=45571867

10000 column=f1:province, timestamp=1480298575888, value=\xE6\xB9\x96\xE5\x8C\x97

10000 column=f1:uid, timestamp=1480298575888, value=100198255

10000 column=f1:visit\_date, timestamp=1480298575888, value=2014-12-05

100000 column=f1:behavior\_type, timestamp=1480298594850, value=1

100000 column=f1:item\_category, timestamp=1480298594850, value=6580

100000 column=f1:item\_id, timestamp=1480298594850, value=78973192

100000 column=f1:province, timestamp=1480298594850, value=\xE5\xB1\xB1\xE4\xB8\x9C

100000 column=f1:uid, timestamp=1480298594850, value=101480065

100000 column=f1:visit\_date, timestamp=1480298594850, value=2014-11-29

100001 column=f1:behavior\_type, timestamp=1480298594850, value=1

100001 column=f1:item\_category, timestamp=1480298594850, value=3472

100001 column=f1:item\_id, timestamp=1480298594850, value=34929314

100001 column=f1:province, timestamp=1480298594850, value=\xE5\x8C\x97\xE4\xBA\xAC\xE5\xB8\x82

100001 column=f1:uid, timestamp=1480298594850, value=101480065

100001 column=f1:visit\_date, timestamp=1480298594850, value=2014-12-15

100002 column=f1:behavior\_type, timestamp=1480298594850, value=1

100002 column=f1:item\_category, timestamp=1480298594850, value=10392

100002 column=f1:item\_id, timestamp=1480298594850, value=401104894

100002 column=f1:province, timestamp=1480298594850, value=\xE6\xB1\x9F\xE8\xA5\xBF

100002 column=f1:uid, timestamp=1480298594850, value=101480065

100002 column=f1:visit\_date, timestamp=1480298594850, value=2014-11-29

100003 column=f1:behavior\_type, timestamp=1480298594850, value=1

100003 column=f1:item\_category, timestamp=1480298594850, value=5894

100003 column=f1:item\_id, timestamp=1480298594850, value=217913901

100003 column=f1:province, timestamp=1480298594850, value=\xE9\xBB\x91\xE9\xBE\x99\xE6\xB1\x9F

100003 column=f1:uid, timestamp=1480298594850, value=101480065

100003 column=f1:visit\_date, timestamp=1480298594850, value=2014-12-04

100004 column=f1:behavior\_type, timestamp=1480298594850, value=1

100004 column=f1:item\_category, timestamp=1480298594850, value=12189

100004 column=f1:item\_id, timestamp=1480298594850, value=295053167

100004 column=f1:province, timestamp=1480298594850, value=\xE6\xB5\xB7\xE5\x8D\x97

100004 column=f1:uid, timestamp=1480298594850, value=101480065

100004 column=f1:visit\_date, timestamp=1480298594850, value=2014-11-26

10 row(s) in 2.1550 seconds

注意，我们用limit10是返回HBase表中的前面10行数据，但是，上面的结果，从“行数”来看，给人一种错误，似乎不是10行，要远远多于10行。这是因为，HBase在显示数据的时候，和关系型数据库MySQL是不同的，每行显示的不是一行记录，而是一个“单元格”。

**五、使用HBase Java API把数据从本地导入到HBase中**

1、启动Hadoop集群、HBase服务  
请首先确保启动了Hadoop集群和HBase服务。如果还没有启动，请在Linux系统中打开一个终端。  
首先，按照下面命令启动Hadoop：

1. cd /usr/local/hadoop
2. ./sbin/start-all.sh

Shell 命令

然后，按照下面命令启动HBase：

1. cd /usr/local/hbase
2. ./bin/start-hbase.sh

Shell 命令

2、数据准备  
实际上，我们也可以编写Java程序，直接从HDFS中读取数据加载到HBase。但是，这里我们展示的是如何用JAVA程序把本地数据导入到HBase中。你只要把程序做简单修改，就可以实现从HDFS中读取数据加载到HBase。  
首先，请将之前的user\_action数据从HDFS复制到Linux系统的本地文件系统中，命令如下：

1. cd /usr/local/bigdatacase/dataset
2. /usr/local/hadoop/bin/hdfs dfs -get /user/hive/warehouse/dblab.db/user\_action .
3. *#将HDFS上的user\_action数据复制到本地当前目录，注意'.'表示当前目录*
4. cat ./user\_action/\* | head -10 *#查看前10行数据*
5. cat ./user\_action/00000\* > user\_action.output *#将00000\*文件复制一份重命名为user\_action.output，\*表示通配符*
6. head user\_action.output *#查看user\_action.output前10行*

Shell 命令

3、编写数据导入程序  
我们这里采用Eclipse编写Java程序实现HBase数据导入功能。关于如何使用Eclipse编写Java程序调用HBase API，

请使用Eclipse编写ImportHBase程序（Java代码在本文最后的附录部分），并打包成可执行jar包，命名为ImportHBase.jar。  
然后，请在“/usr/local/bigdatacase/”目录下面新建一个hbase子目录，用来存放ImportHBase.jar。  
4、数据导入  
现在开始执行数据导入操作。  
使用Java程序将数据从本地导入HBase中，导入前，请先清空user\_action表。  
请在之前已经打开的HBase Shell窗口中（也就是在“hbase>”命令提示符下）执行下面操作：

1. hbase> truncate 'user\_action'
2. Truncating 'user\_action' table (it may take a while):
3. - Disabling table...
4. - Truncating table...
5. 0 row(s) in 4.0120 seconds
6. *//删除以后再查看就没有记录了*
7. hbase> scan 'user\_action',{LIMIT=>10}
8. ROW COLUMN+CELL
9. 0 row(s) in 0.4010 seconds

hbase

下面就可以运行hadoop jar命令运行程序：

1. /usr/local/hadoop/bin/hadoop jar /usr/local/bigdatacase/hbase/ImportHBase.jar HBaseImportTest /usr/local/bigdatacase/dataset/user\_action.output

Shell 命令

命令解释如下：

/usr/local/hadoop/bin/hadoop jar #hadoop jar包执行方式

/usr/local/bigdatacase/hbase/ImportHBase.jar #jar包的路径

HBaseImportTest #主函数入口

/usr/local/bigdatacase/dataset/user\_action.output #main方法接收的参数args，用来指定输入文件的路径

这个命令大概会执行3分钟左右，执行过程中，屏幕上会打印出执行进度，每执行1万条，对打印出一行信息，所以，整个执行过程屏幕上显示如下信息：

10000

20000

30000

40000

50000

60000

70000

80000

90000

100000

110000

120000

130000

140000

150000

160000

170000

180000

190000

200000

210000

220000

230000

240000

250000

260000

270000

280000

290000

300000

Total Time: 259001 ms

5、查看HBase中user\_action表数据  
下面，再次切换到HBase Shell窗口，执行下面命令查询数据：

1. habse> scan 'user\_action',{LIMIT=>10} *#只查询前面10行*

hbase

就可以得到类似下面的查询结果了：

1 column=f1:behavior\_type, timestamp=1480298573684, value=1

1 column=f1:item\_category, timestamp=1480298573684, value=4076

1 column=f1:item\_id, timestamp=1480298573684, value=285259775

1 column=f1:province, timestamp=1480298573684, value=\xE5\xB9\xBF\xE4\xB8\x9C

1 column=f1:uid, timestamp=1480298573684, value=10001082

1 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-08

10 column=f1:behavior\_type, timestamp=1480298573684, value=1

10 column=f1:item\_category, timestamp=1480298573684, value=10894

10 column=f1:item\_id, timestamp=1480298573684, value=323339743

10 column=f1:province, timestamp=1480298573684, value=\xE5\xB1\xB1\xE4\xB8\x9C

10 column=f1:uid, timestamp=1480298573684, value=10001082

10 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-12

100 column=f1:behavior\_type, timestamp=1480298573684, value=1

100 column=f1:item\_category, timestamp=1480298573684, value=10576

100 column=f1:item\_id, timestamp=1480298573684, value=275221686

100 column=f1:province, timestamp=1480298573684, value=\xE6\xB9\x96\xE5\x8C\x97

100 column=f1:uid, timestamp=1480298573684, value=10001082

100 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-02

1000 column=f1:behavior\_type, timestamp=1480298573684, value=1

1000 column=f1:item\_category, timestamp=1480298573684, value=3381

1000 column=f1:item\_id, timestamp=1480298573684, value=168463559

1000 column=f1:province, timestamp=1480298573684, value=\xE5\xB1\xB1\xE8\xA5\xBF

1000 column=f1:uid, timestamp=1480298573684, value=100068031

1000 column=f1:visit\_date, timestamp=1480298573684, value=2014-12-02

10000 column=f1:behavior\_type, timestamp=1480298575888, value=1

10000 column=f1:item\_category, timestamp=1480298575888, value=12488

10000 column=f1:item\_id, timestamp=1480298575888, value=45571867

10000 column=f1:province, timestamp=1480298575888, value=\xE6\xB9\x96\xE5\x8C\x97

10000 column=f1:uid, timestamp=1480298575888, value=100198255

10000 column=f1:visit\_date, timestamp=1480298575888, value=2014-12-05

100000 column=f1:behavior\_type, timestamp=1480298594850, value=1

100000 column=f1:item\_category, timestamp=1480298594850, value=6580

100000 column=f1:item\_id, timestamp=1480298594850, value=78973192

100000 column=f1:province, timestamp=1480298594850, value=\xE5\xB1\xB1\xE4\xB8\x9C

100000 column=f1:uid, timestamp=1480298594850, value=101480065

100000 column=f1:visit\_date, timestamp=1480298594850, value=2014-11-29

100001 column=f1:behavior\_type, timestamp=1480298594850, value=1

100001 column=f1:item\_category, timestamp=1480298594850, value=3472

100001 column=f1:item\_id, timestamp=1480298594850, value=34929314

100001 column=f1:province, timestamp=1480298594850, value=\xE5\x8C\x97\xE4\xBA\xAC\xE5\xB8\x82

100001 column=f1:uid, timestamp=1480298594850, value=101480065

100001 column=f1:visit\_date, timestamp=1480298594850, value=2014-12-15

100002 column=f1:behavior\_type, timestamp=1480298594850, value=1

100002 column=f1:item\_category, timestamp=1480298594850, value=10392

100002 column=f1:item\_id, timestamp=1480298594850, value=401104894

100002 column=f1:province, timestamp=1480298594850, value=\xE6\xB1\x9F\xE8\xA5\xBF

100002 column=f1:uid, timestamp=1480298594850, value=101480065

100002 column=f1:visit\_date, timestamp=1480298594850, value=2014-11-29

100003 column=f1:behavior\_type, timestamp=1480298594850, value=1

100003 column=f1:item\_category, timestamp=1480298594850, value=5894

100003 column=f1:item\_id, timestamp=1480298594850, value=217913901

100003 column=f1:province, timestamp=1480298594850, value=\xE9\xBB\x91\xE9\xBE\x99\xE6\xB1\x9F

100003 column=f1:uid, timestamp=1480298594850, value=101480065

100003 column=f1:visit\_date, timestamp=1480298594850, value=2014-12-04

100004 column=f1:behavior\_type, timestamp=1480298594850, value=1

100004 column=f1:item\_category, timestamp=1480298594850, value=12189

100004 column=f1:item\_id, timestamp=1480298594850, value=295053167

100004 column=f1:province, timestamp=1480298594850, value=\xE6\xB5\xB7\xE5\x8D\x97

100004 column=f1:uid, timestamp=1480298594850, value=101480065

100004 column=f1:visit\_date, timestamp=1480298594850, value=2014-11-26

10 row(s) in 0.6380 seconds

实验顺利结束！

**附录：ImportHBase.java**

1. import java.io.BufferedReader;
2. import java.io.FileInputStream;
3. import java.io.IOException;
4. import java.io.InputStreamReader;
5. import java.util.List;
7. import org.apache.hadoop.conf.Configuration;
8. import org.apache.hadoop.hbase.HBaseConfiguration;
9. import org.apache.hadoop.hbase.KeyValue;
10. import org.apache.hadoop.hbase.client.Get;
11. import org.apache.hadoop.hbase.client.HBaseAdmin;
12. import org.apache.hadoop.hbase.client.HTable;
13. import org.apache.hadoop.hbase.client.Put;
14. import org.apache.hadoop.hbase.client.Result;
15. import org.apache.hadoop.hbase.client.ResultScanner;
16. import org.apache.hadoop.hbase.client.Scan;
17. import org.apache.hadoop.hbase.util.Bytes;
19. public class HBaseImportTest extends Thread {
20. public Configuration config;
21. public HTable table;
22. public HBaseAdmin admin;
24. public HBaseImportTest() {
25. config = HBaseConfiguration.create();
26. *// config.set("hbase.master", "master:60000");*
27. *// config.set("hbase.zookeeper.quorum", "master");*
28. try {
29. table = new HTable(config, Bytes.toBytes("user\_action"));
30. admin = new HBaseAdmin(config);
31. } catch (IOException e) {
32. e.printStackTrace();
33. }
34. }
36. public static void main(String[] args) throws Exception {
37. if (args.length == 0) { *//第一个参数是该jar所使用的类，第二个参数是数据集所存放的路径*
38. throw new Exception("You must set input path!");
39. }
41. String fileName = args[args.length-1]; *//输入的文件路径是最后一个参数*
42. HBaseImportTest test = new HBaseImportTest();
43. test.importLocalFileToHBase(fileName);
44. }
46. public void importLocalFileToHBase(String fileName) {
47. long st = System.currentTimeMillis();
48. BufferedReader br = null;
49. try {
50. br = new BufferedReader(new InputStreamReader(new FileInputStream(
51. fileName)));
52. String line = null;
53. int count = 0;
54. while ((line = br.readLine()) != null) {
55. count++;
56. put(line);
57. if (count % 10000 == 0)
58. System.out.println(count);
59. }
60. } catch (IOException e) {
61. e.printStackTrace();
62. } finally {
64. if (br != null) {
65. try {
66. br.close();
67. } catch (IOException e) {
68. e.printStackTrace();
69. }
70. }
72. try {
73. table.flushCommits();
74. table.close(); *// must close the client*
75. } catch (IOException e) {
76. e.printStackTrace();
77. }
79. }
80. long en2 = System.currentTimeMillis();
81. System.out.println("Total Time: " + (en2 - st) + " ms");
82. }
84. @SuppressWarnings("deprecation")
85. public void put(String line) throws IOException {
86. String[] arr = line.split("\t", -1);
87. String[] column = {"id","uid","item\_id","behavior\_type","item\_category","date","province"};
89. if (arr.length == 7) {
90. Put put = new Put(Bytes.toBytes(arr[0]));*// rowkey*
91. for(int i=1;i<arr.length;i++){
92. put.add(Bytes.toBytes("f1"), Bytes.toBytes(column[i]),Bytes.toBytes(arr[i]));
93. }
94. table.put(put); *// put to server*
95. }
96. }
98. public void get(String rowkey, String columnFamily, String column,
99. int versions) throws IOException {
100. long st = System.currentTimeMillis();
102. Get get = new Get(Bytes.toBytes(rowkey));
103. get.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(column));
105. Scan scanner = new Scan(get);
106. scanner.setMaxVersions(versions);
108. ResultScanner rsScanner = table.getScanner(scanner);
110. for (Result result : rsScanner) {
111. final List<KeyValue> list = result.list();
112. for (final KeyValue kv : list) {
113. System.out.println(Bytes.toStringBinary(kv.getValue()) + "\t"
114. + kv.getTimestamp()); *// mid + time*
115. }
117. }
118. rsScanner.close();
120. long en2 = System.currentTimeMillis();
121. System.out.println("Total Time: " + (en2 - st) + " ms");
122. }
124. }

Java

到这里，第三个步骤的实验内容顺利结束，

**《大数据课程实验案例：网站用户行为分析—-步骤四:利用R进行数据可视化分析》**

本教程介绍大数据课程实验案例“网站用户行为分析”的第四个步骤，利用R进行数据可视化分析。在实践本步骤之前，请先完成该实验案例的第一个步骤，第二个步骤，和第三个步骤：。这里假设你已经完成了前面的这三个步骤。  
R是用于统计分析、绘图的语言和操作环境。R是属于GNU系统的一个自由、免费、源代码开放的软件，它是一个用于统计计算和统计制图的优秀工具。在实际的案例中，更多的是跟其他数据分析工具结合起来，如:MySQL,Hive等。这里使用R的ggplot2绘图工具和recharts的绘图工具来进行可视化分析消费者行为的实例。

# 环境

本步骤需要涉及以下环境：  
操作系统:Linux系统（比如Ubuntu16.04）  
可视化：R（安装在Linux系统下）  
数据库：MySQL（安装在Linux系统下）

R语言包的安装方式如下：  
Ubuntu自带的APT包管理器中的R安装包总是落后于标准版，因此需要添加新的镜像源把APT包管理中的R安装包更新到最新版。  
请登录Linux系统，打开一个终端，然后执行下面命令（并注意保持网络连通，可以访问互联网，因为安装过程要下载各种安装文件）：  
利用vim打开/etc/apt/sources.list文件

1. sudo vim /etc/apt/sources.list

Shell 命令

在文件的最后一行添加厦门大学的镜像源：

deb http://mirrors.tuna.tsinghua.edu.cn/CRAN/bin/linux/ubuntu/ trusty/

退出vim，更新软件源列表

1. sudo apt-get update

Shell 命令

**如果更新软件源出现由于没有公钥，无法验证下列签名的错误，请执行如下命令**

sudo apt-key adv –keyserver keyserver.ubuntu.com –recv-keys 51716619E084DAB9

安装R语言

1. sudo apt-get install r-base

Shell 命令

会提示“您希望继续执行吗？[Y/n]”，可以直接键盘输入“Y”，就可以顺利安装结束。  
安装结束后，可以执行下面命令启动R：

1. R

Shell 命令

启动后，会显示如下信息，并进入“>”命令提示符状态：

R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"

Copyright (C) 2016 The R Foundation for Statistical Computing

Platform: x86\_64-pc-linux-gnu (64-bit)

R是自由软件，不带任何担保。

在某些条件下你可以将其自由散布。

用'license()'或'licence()'来看散布的详细条件。

R是个合作计划，有许多人为之做出了贡献.

用'contributors()'来看合作者的详细情况

用'citation()'会告诉你如何在出版物中正确地引用R或R程序包。

用'demo()'来看一些示范程序，用'help()'来阅读在线帮助文件，或

用'help.start()'通过HTML浏览器来看帮助文件。

用'q()'退出R.

>

“>”就是R的命令提示符，你可以在后面输入R语言命令。  
可以执行下面命令退出R：

1. >q()

R

# 可视化分析MySQL中的数据

## 安装依赖库

为了完成可视化功能，我们需要为R安装一些依赖库，包括：RMySQL、ggplot2、devtools和recharts。  
RMySQL是一个提供了访问MySQL数据库的R语言接口程序的R语言依赖库。  
ggplot2和recharts则是R语言中提供绘图可视化功能的依赖库。  
请启动R进入R命令提示符状态,执行如下命令安装RMySQL：

1. > install.packages('RMySQL')

R

上面命令执行后， 屏幕会提示”Would you like to user a personal library instead?(y/n)”等问题，只要遇到提问，都在键盘输入y后回车即可。然后，屏幕会显示“—在此连线阶段时请选用CRAN的镜子—”，并会弹出一个白色背景的竖条形窗口，窗口标题是“HTTPS CRAN mirros”，标题下面列出了很多国家的镜像列表，我们可以选择位于China的镜像，比如，选择“China(Beijing)[https]”，然后点击“ok”按钮，就开始安装了。安装过程需要几分钟（当然，也和当前网络速度有关系）。  
由于不同用户的Ubuntu开发环境不一样，安装有很大可能因为缺少组件导致失败，如果出现如下错误信息：

Configuration failed because libmysqlclient was not found. Try installing:

\* deb: libmariadb-client-lgpl-dev (Debian, Ubuntu 16.04)

libmariadbclient-dev (Ubuntu 14.04)

\* rpm: mariadb-devel | mysql-devel (Fedora, CentOS, RHEL)

\* csw: mysql56\_dev (Solaris)

\* brew: mariadb-connector-c (OSX)

.................

ERROR: configuration failed for package ‘RMySQL’

\* removing ‘/home/hadoop/R/x86\_64-pc-linux-gnu-library/3.3/RMySQL’

下载的程序包在

‘/tmp/RtmpvEArxz/downloaded\_packages’里

Warning message:

In install.packages("RMySQL") : 安装程序包‘RMySQL’时退出狀態的值不是0

只要根据错误给出的错误信息，进行操作即可。q()退出R命令提示符状态，回到Shell状态,笔者的系统是Ubuntu 16.04,那么，根据上面的英文错误信息，就需要在Shell命令提示符状态下执行下面命令安装libmariadb-client-lgpl-dev：

1. sudo apt-get install libmariadb-client-lgpl-dev

Shell 命令

然后，再次输入下面命令进入R命令提示符状态：

1. R

Shell 命令

启动后，并进入“>”命令提示符状态。然后，执行如下命令安装绘图包ggplot2，如果还出现缺少组件的错误，请按照上面的解决方案解决！

1. > install.packages('ggplot2')

R

然后，屏幕会显示“—在此连线阶段时请选用CRAN的镜子—”，并会弹出一个白色背景的竖条形窗口，窗口标题是“HTTPS CRAN mirros”，标题下面列出了很多国家的镜像列表，我们可以选择位于China的镜像，比如，选择“China(Beijing)[https]”，然后点击“ok”按钮，就开始安装了。这个命令运行后，大概需要安装10分钟时间（当然，也和当前网络速度有关系）。  
下面继续运行下面命令安装devtools：

1. > install.packages('devtools')

R

如果在上面安装devtools的过程中，又出现了错误，处理方法很简单，还是按照上面介绍的方法，根据屏幕上给出的英文错误信息，缺少什么软件，就用sudo apt-get install命令安装该软件就可以了。笔者在Ubuntu16.04上执行devtools安装时，出现了三次错误，笔者根据每次错误的英文提示信息，安装了三个软件libssl-dev、libssh2-1-dev、libcurl4-openssl-dev，安装命令如下：

1. sudo apt-get install libssl-dev
2. sudo apt-get install libssh2-1-dev
3. sudo apt-get install libcurl4-openssl-dev

Shell 命令

读者在安装过程中，可能会出现不同的错误，按照同样的处理方法可以顺利解决。

下面在R命令提示符下再执行如下命令安装taiyun/recharts：

1. > devtools::install\_github('taiyun/recharts')

R

## 分析

以下分析使用的函数方法，都可以使用如下命令查询函数的相关文档。例如：查询sort()函数如何使用

1. ?sort

R

这时，就会进入冒号“:”提示符状态（也就是帮助文档状态），在冒号后面输入q即可退出帮助文档状态，返回到R提示符状态！

1. 连接MySQL,并获取数据

请在Linux系统中新建另外一个终端，然后执行下面命令启动MySQL数据库：

1. service mysql start

Shell 命令

屏幕上会弹出窗口提示你输入密码，本教程的MySQL数据库的用户名是root，密码是hadoop，所以，直接输入密码hadoop，这样就成功启动了MySQL数据库。  
下面，让我们查看一下MySQL数据库中的数据，请执行下面命令进入MySQL命令提示符状态：

1. mysql -u root -p

Shell 命令

会提示你输入密码，我们输入密码hadoop，就进入了“mysql>”提示符状态，下面就可以输入一些SQL语句查询数据：

1. mysql> use dblab;
2. mysql> select \* from user\_action limit 10;

mysql

这样，就可以查看到数据库dblab中的user\_action表的前10行记录，如下：

+--------+-----------+-----------+---------------+---------------+------------+-----------+

| id | uid | item\_id | behavior\_type | item\_category | visit\_date | province |

+--------+-----------+-----------+---------------+---------------+------------+-----------+

| 225653 | 102865660 | 164310319 | 1 | 5027 | 2014-12-08 | 香港 |

| 225654 | 102865660 | 72511722 | 1 | 1121 | 2014-12-13 | 天津市 |

| 225655 | 102865660 | 334372932 | 1 | 5027 | 2014-11-30 | 江苏 |

| 225656 | 102865660 | 323237439 | 1 | 5027 | 2014-12-02 | 广东 |

| 225657 | 102865660 | 323237439 | 1 | 5027 | 2014-12-07 | 山西 |

| 225658 | 102865660 | 34102362 | 1 | 1863 | 2014-12-13 | 内蒙古 |

| 225659 | 102865660 | 373499226 | 1 | 12388 | 2014-11-26 | 湖北 |

| 225660 | 102865660 | 271583890 | 1 | 5027 | 2014-12-06 | 山西 |

| 225661 | 102865660 | 384764083 | 1 | 5399 | 2014-11-26 | 澳门 |

| 225662 | 102865660 | 139671483 | 1 | 5027 | 2014-12-03 | 广东 |

+--------+-----------+-----------+---------------+---------------+------------+-----------+

10 rows in set (0.00 sec)

然后切换到刚才已经打开的R命令提示符终端窗口：

1. library(RMySQL)
2. conn <- dbConnect(MySQL(),dbname='dblab',username='root',password='hadoop',host="127.0.0.1",port=3306)
3. user\_action <- dbGetQuery(conn,'select \* from user\_action')

R

1. 分析消费者对商品的行为
   1. summary(user\_action$behavior\_type)

R

summary() 函数可以得到样本数据类型和长度,如果样本是数值型,我们还能得到样本数据的最小值、最大值、四分位数以及均值信息。  
得到结果：

Length Class Mode

300000 character character

可以看出原来的MySQL数据中,消费者行为变量的类型是字符型。这样不好做比较，需要把消费者行为变量转换为数值型变量

* 1. summary(as.numeric(user\_action$behavior\_type))

R

得到结果：

Min. 1st Qu. Median Mean 3rd Qu. Max.

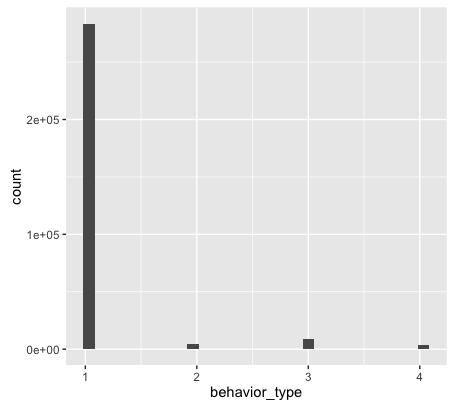
1.000 1.000 1.000 1.105 1.000 4.000

接下来用柱状图表示：

* 1. library(ggplot2)
  2. ggplot(user\_action,aes(as.numeric(behavior\_type)))+geom\_histogram()

R

在使用ggplot2库的时候，需要使用library导入库。ggplot()绘制时，创建绘图对象，即第一个图层，包含两个参数(数据与变量名称映射).变量名称需要被包含aes函数里面。ggplot2的图层与图层之间用“+”进行连接。ggplot2包中的geom\_histogram()可以很方便的实现直方图的绘制。

分析结果如下图:  
  
从上图可以得到:大部分消费者行为仅仅只是浏览。只有很少部分的消费者会购买商品。

1. 分析哪一类商品被购买总量前十的商品和被购买总量
   1. temp <- subset(user\_action,as.numeric(behavior\_type)==4) *# 获取子数据集*
   2. count <- sort(table(temp$item\_category),decreasing = T) *#排序*
   3. print(count[1:10]) *# 获取第1到10个排序结果*

R

subset()函数，从某一个数据框中选择出符合某条件的数据或是相关的列.table()对应的就是统计学中的列联表，是一种记录频数的方法.sort()进行排序，返回排序后的数值向量。  
得到结果：

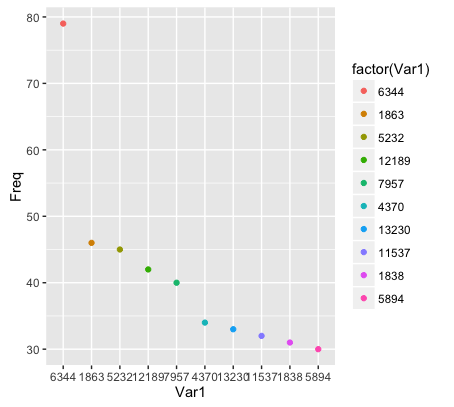
6344 1863 5232 12189 7957 4370 13230 11537 1838 5894

79 46 45 42 40 34 33 32 31 30

结果第一行表示商品分类,该类下被消费的数次。  
接下来用散点图表示:

* 1. result <- as.data.frame(count[1:10]) *#将count矩阵结果转换成数据框*
  2. ggplot(result,aes(Var1,Freq,col=factor(Var1)))+geom\_point()

R

通过 as.data.frame() 把矩阵等转换成为数据框.  
分析结果如下图:  


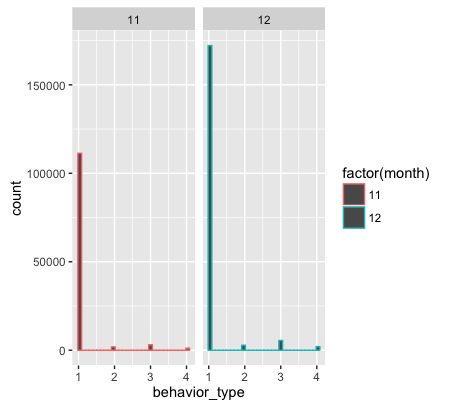
1. 分析每年的哪个月份购买商品的量最多  
   从MySQL直接获取的数据中visit\_date变量都是2014年份,并没有划分出具体的月份,那么可以在数据集增加一列月份数据。
   1. month <- substr(user\_action$visit\_date,6,7) *# visit\_date变量中截取月份*
   2. user\_action <- cbind(user\_action,month) *# user\_action增加一列月份数据*

R

接下来用柱状图分别表示消费者购买量

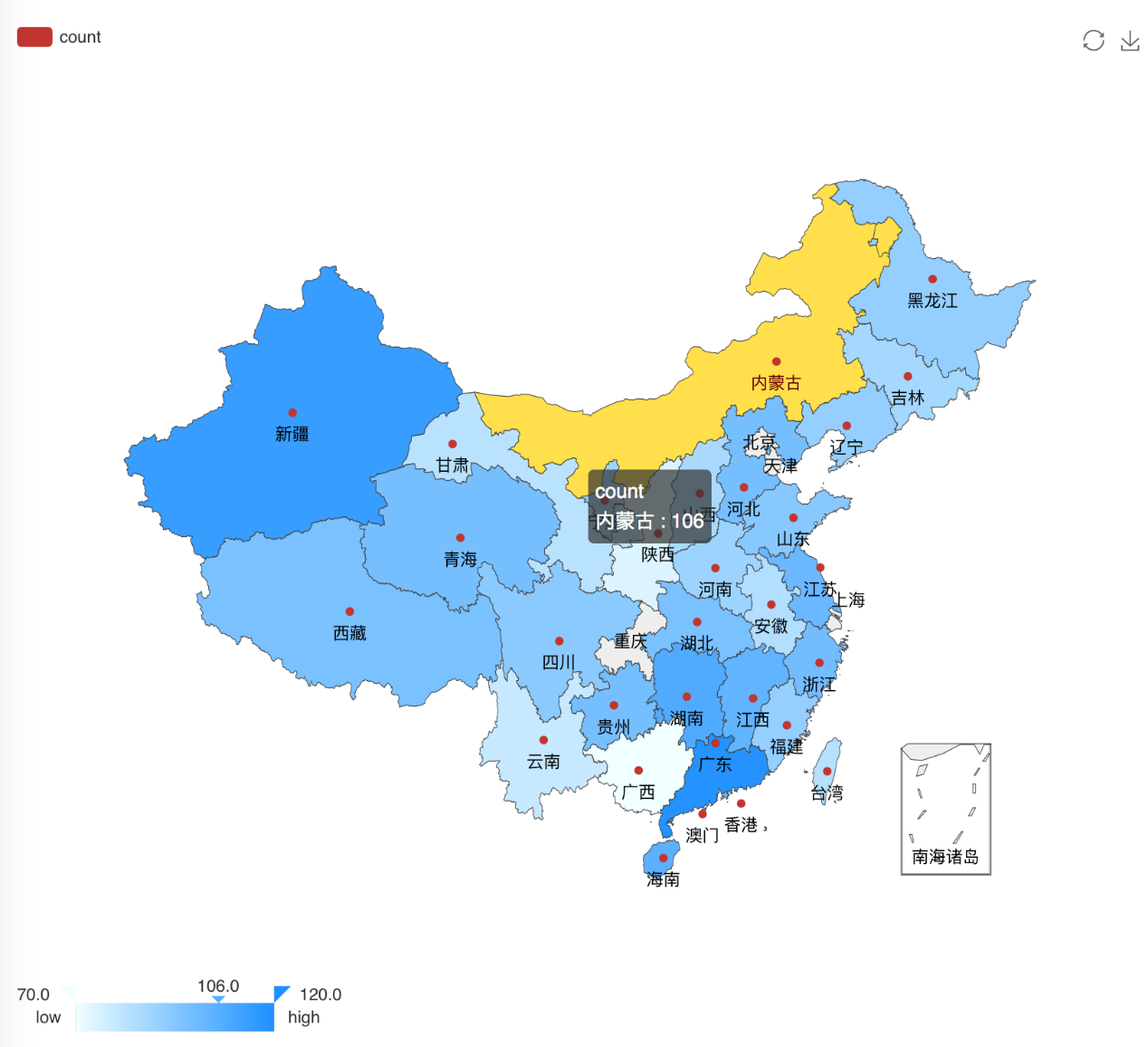
* 1. ggplot(user\_action,aes(as.numeric(behavior\_type),col=factor(month)))+geom\_histogram()+facet\_grid(.~month)

R

aes()函数中的col属性可以用来设置颜色。factor()函数则是把数值变量转换成分类变量,作用是以不同的颜色表示。如果不使用factor()函数，颜色将以同一种颜色渐变的颜色表现。 facet\_grid(.~month)表示柱状图按照不同月份进行分区。  
由于MySQL获取的数据中只有11月份和12月份的数据,所以上图只有显示两个表格。  
分析结果如下图：  


1. 分析国内哪个省份的消费者最有购买欲望
   1. library(recharts)
   2. rel <- as.data.frame(table(temp$province))
   3. provinces <- rel$Var1
   4. x = c()
   5. for(n in provinces){
   6. x[length(x)+1] = nrow(subset(temp,(province==n)))
   7. }
   8. mapData <- data.frame(province=rel$Var1,
   9. count=x, stringsAsFactors=F) *# 设置地图信息*
   10. eMap(mapData, namevar=~province, datavar = ~count) *#画出中国地图*

R

nrow()用来计算数据集的行数。  
分析结果如下图:  


大数据案例网站用户购物行为分析所有实验步骤到此结束！