华中科技大学

大数据处理实验报告

实验二：HBase 的基本操作

专业班级：CS2005 班

学　　号：U202090063

姓　　名：董玲晶

指导教师：石宣化

报告日期：2022.03.28

计算机科学与技术学院

《大数据处理》课程实验报告

实验地点	南一楼 804	课程名称	大数据处理		
实验题目	HBase 的基本操作	成绩		指导教师	石宣化
教师评价	☐ 实验过程正确； ☐ 源程序/实验内容提交； ☐ 程序结构/实验步骤合理； ☐ 实验结果正确； ☐ 语法、语义/命令正确； ☐ 报告规范； 其他：				

一、实验目的

1. 了解 HBase 的用途
2. 掌握 HBase 的基本命令

二、实验内容

1. 实验环境配置
2. 准备数据（20’）
3. 查询数据（30’）
4. 修改数据（20’）
5. Region 初探（20’）
6. Hive 初探（10’）
7. 实验总结（10’）

三、实验环境

1.软件：系统搭载 Hadoop2.8.3，HBase1.3.1，Hive2.3.3，使用弹性公网 IP 访问 MRS。

2.硬件：使用 MRS1.9.2 分析集群。Master 节点和分析 Core 节点均搭载 4 个 Cortex 虚拟 CPU，16G 内存，以及高 IO/100G 数据盘和系统盘。

四、实验过程或步骤（源程序）

2.1 数据准备

2.1.1 设置环境变量并进入 hbase shell

```
# source env_file
# hbase shell
```

2.1.2 用 create 命令，创建一个表，表名为 U202090063（本人学号），列族名为 cf1. 如图 2.1.a 所示。

```
# create ‘U202090063’,{ NAME=> ‘cf1’ }
```

```
hbase(main):001:0> create 'U202090063', {NAME=>'cf1'}
2022-03-24 19:54:55,126 INFO  [main] client.HBaseAdmin: Created U202090063
0 row(s) in 4.7100 seconds

=> Hbase::Table - U202090063
hbase(main):002:0>
```

图 2.1.a

2.1.3 用 list 命令显示所有的表。如图 2.1.b 所示。

```
# list
```

```
hbase(main):002:0> list
TABLE
U202090063
1 row(s) in 0.0840 seconds

=> ["U202090063"]
hbase(main):003:0>
```

图 2.1.b

2.1.4 用 put 命令向表中增加两行数据，数据如图 2.1.c 所示，操作结果如图 2.1.d 所示。注意：操作指令的前三行是为了添加第一行行键为 20200001 的数据，后三行是为了添加第二行行键为 20200002 的数据。全部添加完毕后，用 scan 语句查看目前表中的所有数据，如图 2.1.e 所示。

```
# put ‘U202090063’, ‘20200001’, ‘cf1:name’, ‘tom’
# put ‘U202090063’, ‘20200001’, ‘cf1:gender’, ‘male’
# put ‘U202090063’, ‘20200001’, ‘cf1:age’, ‘20’
```

```
# put ‘U202090063’, ‘20200002’, ‘cf1:name’, ‘hanmeimei’
# put ‘U202090063’, ‘20200002’, ‘cf1:gender’, ‘female’
# put ‘U202090063’, ‘20200002’, ‘cf1:age’, ‘19’
```

行键	列族	列名	单元格的值
20200001	cf1	name	tom
20200001	cf1	gender	male
20200001	cf1	age	20
20200002	cf1	name	hanmeimei
20200002	cf1	gender	female
20200002	cf1	age	19

图 2.1.c

```
hbase(main):003:0> put'U202090063','20200001','cf1:name','tom'
0 row(s) in 0.1070 seconds

hbase(main):004:0> put'U202090063','20200001','cf1:gender','
male'
0 row(s) in 0.0200 seconds

hbase(main):005:0> put'U202090063','20200001','cf1:age','20'
0 row(s) in 0.0100 seconds

hbase(main):006:0> put'U202090063','20200002','cf1:name','hanmeimei'
0 row(s) in 0.0290 seconds

hbase(main):007:0> put'U202090063','20200002','cf1:gender','female'
0 row(s) in 0.0130 seconds
                   put'U202090063','20200002','cf1:age','19'
0 row(s) in 0.0100 seconds
```

图 2.1.d

```
hbase(main):009:0> scan 'U202090063'
ROW                                COLUMN+CELL
 20200001                          column=cf1:age, timestamp=1648123346278, value=20
 20200001                          column=cf1:gender, timestamp=1648123301617, value=male
 20200001                          column=cf1:name, timestamp=1648123239108, value=tom
 20200002                          column=cf1:age, timestamp=1648123457639, value=19
 20200002                          column=cf1:gender, timestamp=1648123408652, value=female
 20200002                          column=cf1:name, timestamp=1648123373180, value=hanmeimei
2 row(s) in 0.0590 seconds
```

图 2.1.e

2.2 查询数据

2.2.1 用 scan 语句查找表中，列族名为 cf1 的数据。如图 2.2.a 所示。

```
# scan ‘U202090063’, { COLUMNS => ‘cf1’ }
```

```
hbase(main):011:0> scan 'U202090063', {COLUMNS => 'cf1'}
ROW                                   COLUMN+CELL
 20200001                             column=cf1:age, timestamp=1648123346278, value=20
 20200001                             column=cf1:gender, timestamp=1648123301617, value=male
 20200001                             column=cf1:name, timestamp=1648123239108, value=tom
 20200002                             column=cf1:age, timestamp=1648123457639, value=19
 20200002                             column=cf1:gender, timestamp=1648123408652, value=female
 20200002                             column=cf1:name, timestamp=1648123373180, value=hanmeimei
2 row(s) in 0.0250 seconds
```

图 2.2.a

2.2.2 用 scan 语句查找表中列族名为 cf1，列名为 name 的数据。如图 2.2.b 所示。

scan ‘U202090063’,{ COLUMNS => [‘cf1:name’] }

```
hbase(main):012:0> scan 'U202090063', {COLUMNS => ['cf1:name']}
ROW                                   COLUMN+CELL
 20200001                             column=cf1:name, timestamp=1648123239108, value=tom
 20200002                             column=cf1:name, timestamp=1648123373180, value=hanmeimei
2 row(s) in 0.0290 seconds
```

图 2.2.b

2.2.3 用 get 语句查找表中行键为 20200001 的行。如图 2.2.c 所示。

get ‘U202090063’, ‘20200001’

```
hbase(main):013:0> get 'U202090063','20200001'
COLUMN                                CELL
 cf1:age                              timestamp=1648123346278, value=20
 cf1:gender                           timestamp=1648123301617, value=male
 cf1:name                             timestamp=1648123239108, value=tom
1 row(s) in 0.0290 seconds
```

图 2.2.c

2.2.4 用 get 语句查找表中，行键为 20200001，列族为 cf1，列名为 name 的数据。如图 2.2.d 所示。

get ‘U202090063’, ‘20200001’, ‘cf1:name’

```
hbase(main):014:0> get 'U202090063','20200001','cf1:name'
COLUMN                                CELL
 cf1:name                             timestamp=1648123239108, value=tom
1 row(s) in 0.0140 seconds
```

图 2.2.d

2.2.5 用 scan 语句查看表中起始行键为 20200001，终止行键为 20200002（不包括），限制长度为 2 的数据。如图 2.2.e 所示。

scan ‘U202090063’, { LIMIT=>2, STARTROW=>‘20200001’, STOPROW=>‘20200002’ }

```
hbase(main):015:0> scan 'U202090063', {LIMIT => 2, STARTROW=>'20200001', STOPROW=>'20200002'}
ROW                                   COLUMN+CELL
 20200001                             column=cf1:age, timestamp=1648123346278, value=20
 20200001                             column=cf1:gender, timestamp=1648123301617, value=male
 20200001                             column=cf1:name, timestamp=1648123239108, value=tom
1 row(s) in 0.0650 seconds
```

图 2.2.e

2.2.6 用 scan 语句查看所有数据值为 20 的行。如图 2.2.f 所示。

scan ‘U202090063’, FILTER=>“ValueFilter(=, ‘binary:20’)”

```
hbase(main):017:0> scan 'U202090063',FILTER=>"ValueFilter(=,'binary:20')"
ROW                                   COLUMN+CELL
 20200001                             column=cf1:age, timestamp=1648123346278, value=20
1 row(s) in 0.0250 seconds
```

图 2.2.f

2.2.7 用 scan 语句查看所有数据值为 tom 的行。如图 2.2.g 所示。

scan ‘U202090063’, FILTER=>“ValueFilter(=, ‘binary:tom’)”

```
hbase(main):016:0> scan 'U202090063',FILTER=>"ValueFilter(=,'binary:tom')"
ROW                                   COLUMN+CELL
 20200001                             column=cf1:name, timestamp=1648123239108, value=tom
1 row(s) in 0.0350 seconds
```

图 2.2.g

2.2.8 用 scan 语句查看所有列名为 gender 的列。如图 2.2.h 所示。

scan ‘U202090063’, FILTER=>"ColumnPrefixFilter(‘gender’)"

```
hbase(main):018:0> scan 'U202090063',FILTER=>"ColumnPrefixFilter('gender')"
ROW                                   COLUMN+CELL
 20200001                             column=cf1:gender, timestamp=1648123301617, value=male
 20200002                             column=cf1:gender, timestamp=1648123408652, value=female
2 row(s) in 0.0280 seconds
```

图 2.2.h

2.2.9 用 scan 语句查看列名为 name，值为 hanmeimei 的行。如图 2.2.i 所示

scan ‘U202090063’, FILTER => "ColumnPrefixFilter(‘gender’) AND ValueFilter(=, ‘binary:hanmeimei’)"

```
hbase(main):019:0> scan 'U202090063',FILTER=>"ColumnPrefixFilter('name') AND ValueFilter(=,'binary:hanmeimei')"
ROW                                   COLUMN+CELL
 20200002                             column=cf1:name, timestamp=1648123373180, value=hanmeimei
1 row(s) in 0.0830 seconds
```

图 2.2.i

2.2.10 用 desc 语句查看表的属性。如图 2.2.j 所示。

desc ‘U202090063’

```
hbase(main):020:0> desc 'U202090063'
Table U202090063 is ENABLED
U202090063
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf1', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER
', COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}
1 row(s) in 0.0410 seconds
```

图 2.2.j

2.3 修改数据

2.3.1 用 alter 语句改变表的 VERSIONS 为 5 以显示更多的历史版本。如图 2.3.a 所示。

```
# alter ‘U202090063’, NAME=>‘cf1’, VERSIONS=>5
```

```
hbase(main):021:0> alter 'U202090063',NAME=>'cf1',VERSIONS=>5
Updating all regions with the new schema...
0/1 regions updated.
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 4.4720 seconds
```

图 2.3.a

2.3.2 用 put 语句添加行键为 20200001，列族为 cf1，列名 name 的多个历史版本，用 get 语句查看添加结果。添加数据操作如图 2.3.b 所示，添加操作如图 2.3.c 所示，查看结果如图 2.3.d 所示。

```
# put ‘U202090063’, ‘20200001’, ‘cf1:name’, ‘LiSi’
# put ‘U202090063’, ‘20200001’, ‘cf1:name’, ‘ZhangSan’
# put ‘U202090063’, ‘20200001’, ‘cf1:name’, ‘WangWu’
```

行键	列族	列名	单元格的值
20200001	cf1	name	LiSi
20200001	cf1	name	ZhangSan'
20200002	cf1	name	WangWu

图 2.3.b

```
hbase(main):023:0> put 'U202090063', '20200001', 'cf1:name', 'LiSi'
0 row(s) in 0.0140 seconds

hbase(main):024:0> put 'U202090063', '20200001', 'cf1:name', 'ZhangSan'
0 row(s) in 0.0100 seconds

hbase(main):025:0> put 'U202090063', '20200001', 'cf1:name', 'WangWu'
0 row(s) in 0.0050 seconds
```

图 2.3.c

```
hbase(main):028:0> get 'U202090063', '20200001', {COLUMN=>'cf1:name', VERSIONS=>5}
COLUMN                                  CELL
 cf1:name                               timestamp=1648126422674, value=WangWu
 cf1:name                               timestamp=1648126413740, value=ZhangSan
 cf1:name                               timestamp=1648126399164, value=LiSi
 cf1:name                               timestamp=1648123239108, value=tom
1 row(s) in 0.0090 seconds
```

图 2.3.d

2.3.3 用 get 语句查看所有行键为 20200001，列簇为 cf1 的多版本数据。如图 2.3.e 所示。

```
# get ‘U202090063’, ‘20200001’, { COLUMN => ‘cf1’, VERSIONS => 5 }
```

```
hbase(main):029:0> get 'U202090063', '20200001', {COLUMN=>'cf1', VERSIONS=>5}
COLUMN                                  CELL
 cf1:age                                timestamp=1648123346278, value=20
 cf1:gender                             timestamp=1648123301617, value=male
 cf1:name                               timestamp=1648126422674, value=WangWu
 cf1:name                               timestamp=1648126413740, value=ZhangSan
 cf1:name                               timestamp=1648126399164, value=LiSi
 cf1:name                               timestamp=1648123239108, value=tom
1 row(s) in 0.0160 seconds
```

图 2.3.e

2.3.4 用 delete 语句删除行键为 20200002，列名为 age 的数据。如图 2.3.f 所示。

```
# delete ‘U202090063’, ‘20200002’, ‘cf1:age’
```

```
hbase(main):030:0> delete 'U202090063', '20200002', 'cf1:age'
0 row(s) in 0.0540 seconds
```

图 2.3.f

2.3.5 用 deleteall 语句删除行键为 20200002 的行，再用 scan 语句查看表，可发现行键为 20200002 的行已被删除。如图 2.3.g 所示。

```
# deleteall ‘U202090063’, ‘20200002’
# scan ‘U202090063’
```

```
hbase(main):031:0> deleteall 'U202090063', '20200002'
0 row(s) in 0.0130 seconds

hbase(main):032:0> scan 'U202090063'
ROW                                     COLUMN+CELL
 20200001                               column=cf1:age, timestamp=1648123346278, value=20
 20200001                               column=cf1:gender, timestamp=1648123301617, value=male
 20200001                               column=cf1:name, timestamp=1648126422674, value=WangWu
1 row(s) in 0.0140 seconds
```

图 2.3.g

2.3.6 用 disable 和 drop 语句删除整个表。如图 2.3.h 所示。

disable ‘U202090063’

drop ‘U202090063’

```
hbase(main):033:0> disable 'U202090063'
2022-03-24 21:01:44,409 INFO  [main] client.HBaseAdmin: Started disable of U202090063
2022-03-24 21:01:46,671 INFO  [main] client.HBaseAdmin: Disabled U202090063
0 row(s) in 2.2870 seconds

hbase(main):034:0> drop 'U202090063'
2022-03-24 21:02:08,843 INFO  [main] client.HBaseAdmin: Deleted U202090063
0 row(s) in 1.2560 seconds
```

图 2.3.h

2.4 Region 初探

2.4.1 用 create 语句创建具有四个 region 的表，表名为“学号_uniform”，pre-split 算法选择 UniformSplit。如图 2.4.a 所示。

create ‘U202090063_uniform’, ‘cf1’, { NUMREGIONS => 4, SPLITALGO => ‘UniformSplit’ }

```
hbase(main):036:0> create 'U202090063_uniform', 'cf1', {NUMREGIONS => 4, SPLITALGO => 'UniformSplit'}
2022-03-24 21:14:02,659 INFO  [main] client.HBaseAdmin: Created U202090063_uniform
0 row(s) in 4.3890 seconds

=> Hbase::Table - U202090063_uniform
```

图 2.4.a

2.4.2 用 create 语句创建具有四个 region 的表，表名为“学号_num”指定 region 以行键 10000000,20000000,30000000 划分。如图 2.4.b 所示。

create ‘U202090063_num’, ‘cf1’, SPLITS => [‘10000000’, ‘20000000’, ‘30000000’]

```
hbase(main):037:0> create 'U202090063_num', 'cf1', SPLITS=>['10000000', '20000000', '30000000']
2022-03-24 21:17:13,096 INFO  [main] client.HBaseAdmin: Created U202090063_num
0 row(s) in 4.2570 seconds

=> Hbase::Table - U202090063_num
hbase(main):038:0>
```

图 2.4.b

2.4.3 在 Manager 中查看 HBase。如图 2.4.c 和 2.4.d 和 2.4.e 所示。

Tables

User Tables | System Tables | Snapshots

2 table(s) in set. [Details]

Namespace	Table Name	Online Regions	Offline Regions	Failed Regions	Split Regions	Other Regions	Description
default	U202090063_num	4	0	0	0	0	'U202090063_num', {NAME => 'cf1'}
default	U202090063_uniform	4	0	0	0	0	'U202090063_uniform', {NAME => 'cf1'}

图 2.4.c

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
U202090063_uniform,,1648127638314.ec69d4ba4d86ce992d122d45b1b93cc0.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471		@\x00\x00\x00\x00\x00\x00\x00	0.0	0
U202090063_uniform,@\x00\x00\x00\x00\x00\x00\x00,1648127638314.f088826b26ef83121d65999214b39a35.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	@\x00\x00\x00\x00\x00\x00\x00	\x80\x00\x00\x00\x00\x00\x00\x00	0.0	0
U202090063_uniform,\x80\x00\x00\x00\x00\x00\x00,1648127638314.601641107ece872e04b38db79c7962f9.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	\x80\x00\x00\x00\x00\x00\x00\x00	\xC0\x00\x00\x00\x00\x00\x00\x00	0.0	0
U202090063_uniform,\xC0\x00\x00\x00\x00\x00\x00,1648127638314.65896f0ebb44b0f66eff6d8a0892fa50.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	\xC0\x00\x00\x00\x00\x00\x00\x00		0.0	0

图 2.4.d

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
U202090063_num,,1648127828853.2281dade76c789e97407bc27d1017bf0.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471		10000000	0.0	0
U202090063_num,10000000,1648127828853.fe0316a3e902bf38997e92668c563b2d.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	10000000	20000000	0.0	0
U202090063_num,20000000,1648127828853.de27e1d63f1c369a7ba456ccd02c62d0.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	20000000	30000000	0.0	0
U202090063_num,30000000,1648127828853.2430ff14f0db3a83c56b1b11d92fa15b.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	30000000		0.0	0

图 2.4.e

2.4.4 用 put 语句根据两个表的 End key 和 Start Key，选择适当的行键往两个表的不同 region 中添加任意两个数据，使得每个表至少有两个不同 region 中 Requests 不为 0。操作如图 2.4.f 所示，结果如图 2.4.g 和 2.4.h 所示。

```
hbase(main):038:0> put 'U202090063_num', '20000000', 'cf1:name', 'LiSi'
0 row(s) in 0.0270 seconds

hbase(main):039:0> put 'U202090063_num', '10000000', 'cf1:name', 'LiSi'
0 row(s) in 0.0100 seconds

hbase(main):040:0> put 'U202090063_uniform', '10000000', 'cf1:name', 'LiSi'
0 row(s) in 0.0400 seconds

hbase(main):041:0> put 'U202090063_uniform', '20000000', 'cf1:name', 'LiSi'
0 row(s) in 0.0060 seconds

hbase(main):042:0> put 'U202090063_uniform', '@\x00\x00\x00\x00\x00\x00\x00', 'cf1:name', 'LiSi'
0 row(s) in 0.0380 seconds
```

图 2.4.f

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
U202090063_uniform,,1648127638314.ec69d4ba4d86ce992d122d45b1b93cc0.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471		@\x00\x00\x00\x00\x00\x00\x00	0.0	2
U202090063_uniform,@\x00\x00\x00\x00\x00\x00\x00,1648127638314.f088826b26ef83121d65999214b39a35.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	@\x00\x00\x00\x00\x00\x00\x00	\x80\x00\x00\x00\x00\x00\x00\x00	0.0	1
U202090063_uniform,\x80\x00\x00\x00\x00\x00\x00,1648127638314.601641107ece872e04b38db79c7962f9.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	\x80\x00\x00\x00\x00\x00\x00\x00	\xC0\x00\x00\x00\x00\x00\x00\x00	0.0	0
U202090063_uniform,\xC0\x00\x00\x00\x00\x00\x00,1648127638314.65896f0ebb44b0f66eff6d8a0892fa50.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	\xC0\x00\x00\x00\x00\x00\x00\x00		0.0	0

图 2.4.g

Table Regions

Name	Region Server	Start Key	End Key	Locality	Requests
U202090063_num,,1648127828853.2281dade76c789e97407bc27d1017bf0.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471		10000000	0.0	0
U202090063_num,10000000,1648127828853.fe0316a3e902bf38997e92668c563b2d.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	10000000	20000000	0.0	1
U202090063_num,20000000,1648127828853.de27e1d63f1c369a7ba456ccd02c82d0.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	20000000	30000000	0.0	1
U202090063_num,30000000,1648127828853.2430ff14f0db3a83c56b1b11d92fa15b.	node-ana-coresfsZ.mrs-lnnj.com,16020,1648121160471	30000000		0.0	0

图 2.4.h

2.4.5 删除所有的表。

```
# disable  'U202090063_uniform'
# drop  'U202090063_uniform'
# disable  'U202090063_num'
# drop  'U202090063_num'
```

2.5 Hive 初探

2.5.1 准备 file1.txt，内容为“hello hust”，file2.txt，内容为“hello 学号”。按 i 进入编辑模式，编辑完毕后按 Esc 退出编辑模式，在大写模式下按输入‘ZZ’退出 vim 编辑。如图 2.5.a 所示。

```
# vim file1.txt
# vim file2.txt
```

```
hbase(main):043:0> [root@node-master1fnKx ~]# vim file1.txt
[root@node-master1fnKx ~]# vim file2.txt
```

图 2.5.a

2.5.2 将创建的文件移动到 HDFS 中/test 文件夹内。如图 2.5.b 所示。

```
# hdfs dhs -mkdir /test                    /* 创建文件夹/test */
# hdfs dfs -put file1.txt /test            /* 将 file1.txt 放入 */
# hdfs dfs -put file2.txt /test            /* 将 file2.txt 放入 */
# hdfs dfs -ls /test                       /* 查看文件夹下文件 */
```

```
[root@node-master1fnKx ~]# hdfs dfs -mkdir /test
[root@node-master1fnKx ~]# hdfs dfs -put file1.txt /test
[root@node-master1fnKx ~]# hdfs dfs -put file2.txt /test
[root@node-master1fnKx ~]# hdfs dfs -ls /test
Found 2 items
-rw-r--r--   1 root ficommon         11 2022-03-24 21:40 /test/file1.txt
-rw-r--r--   1 root ficommon         17 2022-03-24 21:40 /test/file2.txt
```

图 2.5.b

2.5.3 在 hive 中创建表，tablename 替换为学号，即表名为 U202090063。如图 2.5.c 和 2.5.d 所示。

```
# hive
# create table U202090063(line string);
```

```
[root@node-master1fnKx ~]# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/share/slf4j-log4j12-1.7.10/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/share/log4j-slf4j-impl-2.6.2/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
WARNING: Use "yarn jar" to launch YARN applications.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/share/log4j-slf4j-impl-2.6.2/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/share/slf4j-log4j12-1.7.10/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]

Logging initialized using configuration in jar:file:/opt/share/hive-common-2.3.3-mrs-1.9.0/hive-common-2.3.3-mrs-1.9.0.jar!/hive-log4j2.properties Async:
false
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.
```

图 2.5.c

```
hive> create table U202090063(line string);
OK
Time taken: 1.568 seconds
```

图 2.5.d

2.5.4 加载 hdfs 中的数据到 hive 中. 如图 2.5.e 所示。

```
# load data inpath 'hdfs:///test' overwrite into table U202090063;
```

```
hive> load data inpath 'hdfs:///test' overwrite into table U202090063;
Loading data to table default.u202090063
chgrp: changing ownership of 'hdfs://hacluster/user/hive/warehouse/u202090063': User null does not belong to hive
OK
Time taken: 0.88 seconds
```

图 2.5.e

2.5.5 从 2.5.3 中创建的表 U202090063 中按空格切割每个词语，通过 HiveQL 语句创建词频统计表。如图 2.5.f 和 2.5.g 所示。

```
# create table word_count as
  select word,count(1) as count from
      (select explode(split(line, ' '))as word from U202090063) w
      group by word
      order by word;
```

```
hive> create table word_count as
    > select word,count(1) as count from
    > (select explode(split(line, ' '))as word from U202090063) w
    > group by word
    > order by word;
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 = root_20220324214356_8d7de0f6-01bb-474a-a3ea-ddd60e71a5a9
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>
2022-03-24 21:44:01,203 INFO  [64b2892f-f7de-4713-b5dd-9776b14b097c main] Configuration.deprecation: mapred.submit.replication is deprecated. Instead, use
 mapreduce.client.submit.file.replication
2022-03-24 21:44:03,131 WARN  [64b2892f-f7de-4713-b5dd-9776b14b097c main] mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed.
 Implement the Tool interface and execute your application with ToolRunner to remedy this.
2022-03-24 21:44:08,659 INFO  [64b2892f-f7de-4713-b5dd-9776b14b097c main] input.FileInputFormat: Total input files to process : 2
2022-03-24 21:44:08,673 INFO  [64b2892f-f7de-4713-b5dd-9776b14b097c main] input.CombineFileInputFormat: DEBUG: Terminated node allocation with : Completed
Nodes: 1, size left: 0
2022-03-24 21:44:11,156 INFO  [64b2892f-f7de-4713-b5dd-9776b14b097c main] mapreduce.JobSubmitter: number of splits:1
2022-03-24 21:44:11,654 INFO  [64b2892f-f7de-4713-b5dd-9776b14b097c main] mapreduce.JobSubmitter: Submitting tokens for job: job_1648121135888_0001
2022-03-24 21:44:12,390 INFO  [64b2892f-f7de-4713-b5dd-9776b14b097c main] impl.YarnClientImpl: Submitted application application_1648121135888_0001
2022-03-24 21:44:12,489 INFO  [64b2892f-f7de-4713-b5dd-9776b14b097c main] mapreduce.Job: The url to track the job: http://node-master1fnkx.mrs-lnnj.com:80
88/proxy/application_1648121135888_0001/
Starting Job = job_1648121135888_0001, Tracking URL = http://node-master1fnkx.mrs-lnnj.com:8088/proxy/application_1648121135888_0001/
Kill Command = /opt/client/HDFS/hadoop/bin/hadoop job  -kill job_1648121135888_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-03-24 21:44:22,896 WARN  [64b2892f-f7de-4713-b5dd-9776b14b097c main] mapreduce.Counters: Group org.apache.hadoop.mapred.Task$Counter is deprecated. U
se org.apache.hadoop.mapreduce.TaskCounter instead
2022-03-24 21:44:22,889 Stage-1 map = 0%,  reduce = 0%
2022-03-24 21:44:31,511 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.83 sec
```

图 2.5.f

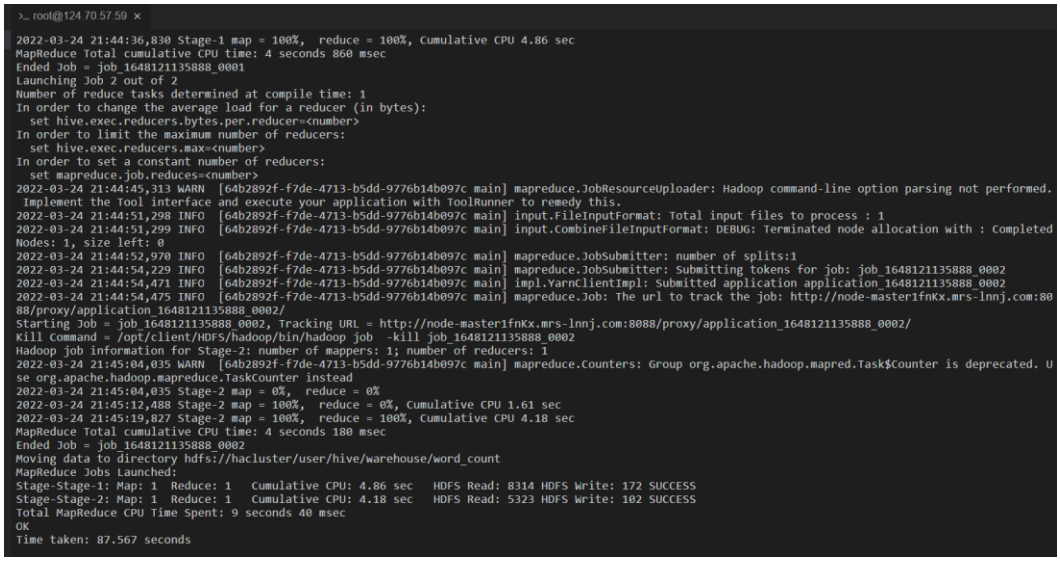

图 2.5.g

2.5.6 通过 HiveQL 语句创建词频统计表。由结果可知‘U202090063’出现了一次，‘hello’出现了两次，‘hust’出现了一次，符合 2.5.1 步骤中写入的内容（hello U202090063 和 hello hust）。如图 2.5.h 所示。

```
# select * from word_count;
```

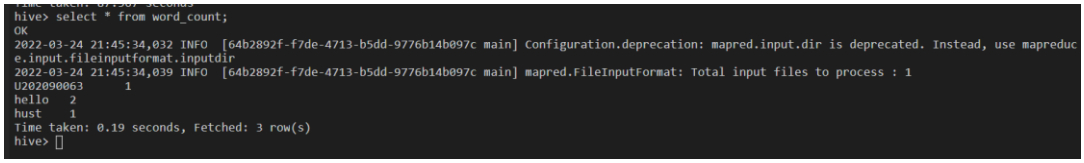

图 2.5.h

五、出现的问题与解决方案

（1）2.1-2.3 主要的问题是不熟悉各种 HBase 的基本命令，看到题目比较茫然不太懂得要如何实现。解决方法是开始上网搜索“HBase 基本命令”先大致地学习了一些基础的命令，如 scan 语句、get 语句、put 语句等。正式开始操作后一边看教程一边实验，有个别类型地题目筛选条件比较多试了很多次都报错，也和同学一起交流探讨或者是上网找资源一起学习。

（2）2.4 的一个问题是不熟悉指令，但另一个比较大的问题是一开始不太懂 region 的概念，看了很多遍任务书还是觉得不太好理解，于是请教了比较厉害的同学，同学讲的更通俗易懂一些,同时也能面对面解答我的疑惑。

（3）2.5 Hive 初探里一开始就遇到了很大的问题——vim 的使用，在输入“vim file1.txt”后出现了好多波浪线，于是又上网搜索了起来，学会了如何在 vim 下编辑文件。

总结来说，感觉问题都主要在于不熟悉各种指令的使用，本次实验花了将近三节课的时间，大部分时间都是在上网找资料和花时间学习指令的使用。

六、实验总结

1.总结与收获

（1）学会了 HBase 基本命令的使用。

（2）学会了在 vim 下便捷编辑文件。

（3）复习巩固 HDFS 基本语句。

（4）学会了 HiveQL 语句（感觉和 SQL 比较像）。

2.实验评价

（1）难度适中，虽然大家都不太会算是零基础，但是网上有很多的资料，只要花点时间操作起来就不难。

（2）算是比较有趣的实验，特别是最后的词频统计也挺有意思的。