

目录

1. 实验一：HBase 分布式部署	
1.1. 实验目的	
1.2. 实验要求	
1.3. 实验环境	
1.4. 实验视图	
1.5. 实验过程	
1.5.1. 实验任务一：部署前期准备	
1.5.2. 实验任务二：修改配置文件（master 节点）	
2. 实验二 HBase 库操作与表操作.....	
2.1. 实验内容与目标	
2.2. 相关知识	
2.3. 实验环境	
2.4. 实验过程	
2.4.1. 实验任务一： HBase 库操作	
2.4.2. 实验任务二：HBase 表管理	
3. 实验三 HBase 数据操作	
3.1. 实验内容与目标	
3.2. 相关知识	
3.3. 实验环境	
3.4. 实验过程	
3.4.1. 实验任务一：简单操作.....	
3.4.2. 实验任务二：模糊查询.....	
3.4.3. 实验任务三：批量导入/导出.....	
3.5. 注意事项	

1. 实验一：HBase 分布式部署

1.1. 实验目的

完成本实验，您应该能够：

- 了解 HBase 的安装流程
- 了解 HBase 的工作原理
- 了解 HBase 环境变量配置

1.2. 实验要求

- 熟悉常用 Linux 操作系统命令
- 熟悉 HBase 分布式部署
- 熟悉 HBase Shell 命令操作

1.3. 实验环境

本实验所需之主要资源环境如表 1-1 所示。

表 1-1 资源环境

服务器集群	3 个以上节点，节点间网络互通，各节点最低配置：双核 CPU、8GB 内存、100G 硬盘
运行环境	CentOS 7.4 (1708) gui 版本
用户名/密码	root/password hadoop/password
服务和组件	HDFS、YARN、MapReduce 等，其他服务根据实验需求安装

1.4. 实验视图

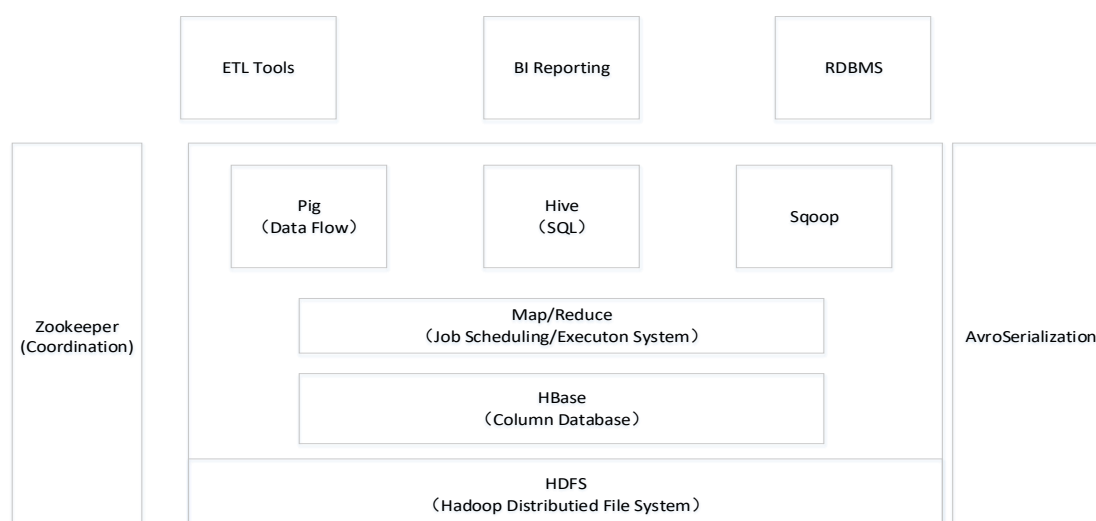


图 1-1 HBase 架构图

1.5. 实验过程

1.5.1. 实验任务一：部署前期准备

1.5.1.1. 步骤一：安装部署 hadoop ha 分布式环境

本实验采用 hadoop 2.7.1 部署三台大数据处理平台环境。如图 1-2 所示

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities -

Datanode Information

In operation

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
slaves2:50010 (192.168.88.152:50010)	0	In Service	16.99 GB	20 KB	2.02 GB	14.96 GB	1	20 KB (0%)	0	2.7.1
slaves1:50010 (192.168.88.151:50010)	0	In Service	16.99 GB	64 KB	2.02 GB	14.96 GB	5	64 KB (0%)	0	2.7.1
master:50010 (192.168.88.150:50010)	2	In Service	16.99 GB	80 KB	2.57 GB	14.42 GB	8	80 KB (0%)	0	2.7.1

图 1-2 hadoop 分布式环境

注意：各主机中的操作系统版本需保持一致。安装过程中都在 root 用户下，本教材中密码统一采用：**password**(注意大小写)

1.5.1.2. 步骤二：解压安装文件

```
[root@master ~]# cd /opt/software/
```

```
[root@master software]# tar -zxvf hbase-1.2.1-bin.tar.gz -C /usr/local/src/ # 解压到统一
```

安装目录

```
[root@master software]# cd
```

```
[root@master ~]# mv /usr/local/src/hbase-1.2.1 /usr/local/src/hbase
```

编辑/etc/profile 文件。

```
[root@master ~]# vi /etc/profile
```

```
[root@slave1 ~]# vi /etc/profile
```

```
[root@slave2 ~]# vi /etc/profile
```

将以下配置信息添加到/etc/profile 文件的末尾。

```
# set hbase environment
```

```
export HBASE_HOME=/usr/local/src/hbase # hbase 安装目录
```

```
export PATH=$PATH:$HBASE_HOME/bin # 将 hbase 的 bin 目录添加到系统环境变量 PATH
```

执行 source /etc/profile 命令，使配置的环境变量在系统全局范围生效。

```
[root@master ~]# source /etc/profile
```

```
[root@slave1 ~]# source /etc/profile
```

```
[root@slave2 ~]# source /etc/profile
```

1.5.2. 实验任务二：修改配置文件（master 节点）

1.5.2.1. 步骤一：conf 下文件修改

HBase 的配置文件放置在安装目录下的 conf 文件夹内，切换到该目录首先修改 HBase 环境配置文件 hbase-env.sh，设置 JAVA_HOME 为自己安装的版本。将以下配置信息添加到 hbase-env.sh 的末尾。

```
[root@master ~]# cd /usr/local/src/hbase/conf
[root@master conf]# vi hbase-env.sh
export JAVA_HOME=/usr/local/src/java
export HADOOP_HOME=/usr/local/src/hadoop
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
export HBASE_MANAGES_ZK=false
export HBASE_LOG_DIR=${HBASE_HOME}/logs
export HBASE_PID_DIR=${HBASE_HOME}/pid
```

JAVA_HOME 为 java 程序所在位置，HBASE_MANAGES_ZK 表示是否使用 HBase 自带的 zookeeper 环境，由于 hadoop ha 已配置 zookeeper 环境，此处设置为 false (默认为 true)，即不使用 hbase 自带的 zookeeper，HBASE_CLASSPATH 指向 hbase 配置文件的路径。HBASE_LOG_DIR 与 HBASE_PID_DIR 分别为日志与 pid 文件输出目录。

修改配置文件 hbase-site.xml，添加相关信息。将以下配置信息添加到 hbase-site.xml 文件<configuration>与</configuration>之间。

```
[root@master conf]# vi hbase-site.xml
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://master:8020/hbase</value>
</property>
<property>
    <name>hbase.master.info.port</name>
    <value>16010</value>
</property>
<property>
    <name>hbase.zookeeper.property.clientPort</name>
    <value>2181</value>
</property>
<property>
    <name>hbase.tmp.dir</name>
```

```

    <value>/usr/local/src/hbase/tmp</value>
</property>
<property>
    <name>zookeeper.session.timeout</name>
    <value>120000</value>
</property>
<property>
    <name>hbase.cluster.distributed</name>
    <value>true</value>
</property>
<property>
    <name>hbase.zookeeper.quorum</name>
    <value>master,slave1,slave2</value>
</property>
<property>
    <name>hbase.zookeeper.property.dataDir</name>
    <value>/usr/local/src/hbase/tmp/zookeeper-hbase</value>
</property>

```

hbase.rootdir:指定 HBase 的存储目录。

hbase.master.info.port: 浏览器的访问端口

hbase.zookeeper.property.clientPort: 指定 zk 的连接端口。

hbase.tmp.dir:指定 hbase 在本地下生成文件路径, 类似于 hadoop.tmp.dir。

zookeeper.session.timeout: RegionServer 与 Zookeeper 间的连接超时时间。当超时时间到后, RegionServer 会被 Zookeeper 从 RS 集群清单中移除, HMaster 收到移除通知后, 会对这台 server 负责的 regions 重新 balance, 让其他存活的 RegionServer 接管。

hbase.cluster.distributed: HBase 是否为分布式模式。

hbase.zookeeper.quorum: 默认值是 localhost, 列出 zookeeper ensemble 中的 servers。

hbase.zookeeper.property.dataDir: 这里表示 HBase 在 ZooKeeper 上存储数据的位置。

修改 regionservers 文件, 删除 localhost, 添加以下内容。

```
[root@master conf]# vi regionservers
```

```
slave1
```

```
slave2
```

为了让 Hbase 读取到 hadoop 的配置, 将 core-site.xml 和 hdfs-site.xml 两个文件拷

贝到 \$HBASE_HOME/conf/ 目录下。

```
[root@master conf]# cp /usr/local/src/hadoop/etc/hadoop/core-site.xml
/usr/local/src/hbase/conf/
```

```
[root@master conf]# cp /usr/local/src/hadoop/etc/hadoop/hdfs-site.xml
/usr/local/src/hbase/conf/
```

1.5.2.2. 步骤二：集群分发

将 master 节点配置好的 HBase 安装包分发给 slave1, slave2 节点。

```
[root@master conf]# cd
```

```
[root@master ~]# scp -r /usr/local/src/hbase root@slave1:/usr/local/src # 从 master
远程拷贝 hbase 文件到 slave1 节点
```

```
[root@master ~]# scp -r /usr/local/src/hbase root@slave2:/usr/local/src # 从 master
远程拷贝 hbase 文件到 slave2 节点
```

```
[root@master ~]# chown -R hadoop:hadoop /usr/local/src/hbase
```

```
[root@slave1 ~] # chown -R hadoop:hadoop /usr/local/src/hbase
```

```
[root@slave2 ~] # chown -R hadoop:hadoop /usr/local/src/hbase
```

```
[root@master ~]# su - hadoop
```

```
[hadoop@master ~]$ source /etc/profile
```

```
[root@slave1 ~] # su - hadoop
```

```
[hadoop@slave1 ~]$ source /etc/profile
```

```
[root@slave2 ~] # su - hadoop
```

```
[hadoop@slave2 ~]$ source /etc/profile
```

1.5.2.3. 步骤三：HBase 集群启动

在 master 主节点, 使用 hadoop 用户切换到 /usr/local/src/hbase/bin 目录下。使用 ./start-hbase.sh 命令启动。

```
[hadoop@master ~]$ zkServer.sh start
```

```
[hadoop@slave1 ~]$ zkServer.sh start
```

```
[hadoop@slave2 ~]$ zkServer.sh start
```

```
[hadoop@master ~]$ start-all.sh
```

```
[hadoop@master ~]$ cd /usr/local/src/hbase/bin
```

```
[hadoop@master bin]$ ./start-hbase.sh
```

用 webUI 查看集群, 特别强调 hbase2.0 的端口是 16010。

Region Servers

Base Stats	Memory	Requests	Storefiles	Compactions
ServerName	Start time	Version	Requests Per Second	Num. Regions
slave1,16020,1595384731524	Wed Jul 22 10:25:31 CST 2020	1.2.1	0	1
slave2,16020,1595384731534	Wed Jul 22 10:25:31 CST 2020	1.2.1	0	2
Total:2			0	3

图 1-3 HBase 浏览器界面

2. 实验二 HBase 库操作与表操作

2.1. 实验内容与目标

完成本实验，您应该能够：

- 掌握 HBase 库节点动态增减
- 掌握 HBase 简单数据表操作

2.2. 相关知识

表：HBase 采用表来组织数据，表由行和列组成，列划分为若干个列族

行：每个 HBase 表都由若干行组成，每个行由行键（row key）来标识。


列族：一个 HBase 表被分组成许多“列族”（Column Family）的集合，它是基本的访问控制单元

列限定符：列族里的数据通过列限定符（或列）来定位


单元格：在 HBase 表中，通过行、列族和列限定符确定一个“单元格”（cell），单元格中存储的数据没有数据类型，总被视为字节数组 byte[]

时间戳：每个单元格都保存着同一份数据的多个版本，这些版本采用时间戳进行索引

	lofo		
	name	major	tel
201536487	ZhangSan	Math	13502548963
202002150	LiSi	Math	15036954852
201582389	Wlen	Math	12569584525



行键



单元格

图 1-4 HBase 表结构

2.3. 实验环境

服务器集群	4 个以上节点，节点间网络互通，各节点最低配置：双核 CPU、8GB 内存、100G 硬盘
运行环境	CentOS 7.4
用户名/密码	root/password hadoop/password
服务和组件	HDFS、Yarn、MapReduce

2.4. 实验过程

2.4.1. 实验任务一： HBase 库操作

2.4.1.1. 步骤一： HBase 集群启动(上一个实验已经启动)

HBase 依赖 hdfs 服务,通过相互之间的依赖关系得到启动顺序为:Zookeeper > hadoop > HBase。

首先启动 Zookeeper, 在所有节点上执行命令。

```
[hadoop@master ~]$ ./zkServer.sh start
```

```
[hadoop@slave1 ~]$ ./zkServer.sh start
```

```
[hadoop@slave2 ~]$ ./zkServer.sh start
```

Zookeeper 选举机制会自动选择 Leader 节点，在 master 节点启动 hadoop 服务。

```
[hadoop@master ~]$ ./start-all.sh
```

hadoop 从节点会自行启动。最后启动 HBase (master 节点)

```
[hadoop@master ~]$ ./start-hbase.sh
```

```
[hadoop@master ~]$ jps
```

```
[root@master hbase]# jps
4368 DFSZKFailoverController
4049 DataNode
44148 HMaster
40532 Main
3621 QuorumPeerMain
5125 JobHistoryServer
3735 JournalNode
7383 NameNode
4506 ResourceManager
45565 Jps
4623 NodeManager
```

图 1-5 jps 查看相关进程

2.4.1.2. 步骤二： HBase 动态删除节点

节点升级或者硬盘扩容在存储服务器上属于正常现象，当某存储节点需要扩容升级短暂下线后需要该节点下线。

假设 slaves3 节点扩容升级, 执行以下命令, 停止该节点上 HBase 服务。

```
[hadoop@master ~]$ cd /usr/local/src/hbase/bin
```

```
[hadoop@master bin]$ graceful_stop.sh slave2
```

graceful_stop.sh 脚本会自行关闭平衡器, 移动 slaves2 节点上的数据到其他节点上, 此步骤会消耗大量时间等待。

同时需要 hadoop 中删除节点。在 hdfs-site.xml 中添加配置。需要新建 exclude 文件, 该文件写入删除节点名称。

```
[hadoop@master bin]$ vi /usr/local/src/hadoop/etc/hadoop/exclude
```

```
slave2
```

```
[hadoop@master bin]$ vi /usr/local/src/hadoop/etc/hadoop/hdfs-site.xml
```

```
<property>
```

```
<name>dfs.hosts.exclude</name>
```

```
<value>/usr/local/src/hadoop/etc/hadoop/exclude</value>
```

```
</property>
```

dfs.hosts.exclude:表示需要删除 exclude 中的节点。

(3) 刷新配置生效。

```
[hadoop@master bin]$ cd
```

```
[hadoop@master ~]$ hadoop dfsadmin -refreshNodes
```

打开 Web UI 监控页面查看, 发现此节点显示 (Decommission In Progress), 表示节点正在做数据迁移, 等待后节点停止, dead node 列表显示下线节点。

(4) 节点下线后需要将 slaves 与 exclude 文件中 slave2 删除, 刷新 hadoop 命令, 此时全部结束。

Region Servers				
Base Stats	Memory	Requests	Storefiles	Compactions
ServerName	Start time	Version	Requests Per Second	Num. Regions
slave1,16020,1595384731524	Wed Jul 22 10:25:31 CST 2020	1.2.1	0	3
Total:1			0	3

Dead Region Servers	
ServerName	Stop time
slave2,16020,1595384731534	Wed Jul 22 10:52:03 CST 2020
Total:	servers: 1

图 1-6 HBase Web 页面查看 Dead Region 节点

2.4.1.3. 步骤三: HBase 动态增加节点

集群的分布式扩展是非关系数据库与传统数据库相比最大的优点。在原有集群基础上增

加新的节点 slave2。增加新节点首先保证新的 hadoop 集群已经运行正常,不需要关闭集群,执行以下命令即可。

(1) 在新的节点上启动服务。切换到新增节点上,使用以下命令。

```
[hadoop@slave2 ~]$ cd /usr/local/src/hbase/bin
```

```
[hadoop@slave2 bin]$ ./hbase-daemon.sh start regionserver
```

Region Servers				
Base Stats	Memory	Requests	Storefiles	Compactions
ServerName	Start time	Version	Requests Per Second	Num. Regions
slave1,16020,1595384731524	Wed Jul 22 10:25:31 CST 2020	1.2.1	0	2
slave2,16020,1595387097657	Wed Jul 22 11:04:57 CST 2020	1.2.1	0	1
Total:2			0	3

Backup Masters		
ServerName	Port	Start Time
Total:0		

图 1-7 HBase 启动

注: 以上步骤的前提是此节点已增加到 hadoop 集群中,且正常使用。

2.4.2. 实验任务二: HBase 表管理

2.4.2.1. 步骤一: 建立表, 两个列簇: name 和 num

打开浏览器,输入 DataEngine 安装完成后提供的 URL,初始账号密码为 admin/admin。如图 1-5 所示。

进入 HBase 命令行

```
[hadoop@master ~]$ hbase shell
```

建立表 student, 两个列簇: name 和 num

```
hbase(main):001:0> create 'student',{NAME=>'name'},{NAME=>'num'}
```

```
0 row(s) in 1.5420 seconds
```

```
=>HBase:: Table - scores
```

新建学生表, 存储姓名与学号。

语法: create <table>, {NAME => <family>, VERSIONS => <VERSIONS>}

Tables

User Tables

System Tables

Snapshots

1 table(s) in set: [\[Details\]](#)

Namespace	Table Name	Online Regions	Offline Regions	Failed Regions	Split Regions	Other Regions	Description
default	student	1	0	0	0	0	'student', (NAME => 'name', VERSIONS => '2'), (NAME => 'num')

图 1-8 HBase Web 查看创建结果

2.4.2.2. 步骤二: 查看所有表与详细信息

```
hbase(main):002:0> list
```

```
TABLE
```

```
student
```

1 row(s) in 0.0100 seconds

=>["student"]

查看建表详细信息

hbase(main):003:0> describe 'student'

Table student is ENABLED

student

COLUMN FAMILIES DESCRIPTION

{NAME => 'name', 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'}

{NAME => 'num', 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'}

2 row(s) in 0.1310 seconds

在建立表时没有指定列的详细属性，系统根据默认设置。

语法: describe <table>

2.4.2.3. 步骤三: 修改表

hbase(main):004:0> alter 'student' ,{NAME=>'tel'}

Updating all regions with the new schema...

1/1 regions updated.

Done.

0 row(s) in 2.2360 seconds 2 row(s) in 0.0230 seconds

新增加新的列 tel, alter 也可以对列删除, 对属性进行修改。

hbase(main):005:0> alter 'student' ,{'NAME'=>'name',VERSIONS=>'2'}

Updating all regions with the new schema...

1/1 regions updated.

Done.

0 row(s) in 2.0350 seconds

hbase(main):006:0> alter 'student' ,{NAME=>'tel',METHOD=>'delete'}

Updating all regions with the new schema...

1/1 regions updated.

Done.

0 row(s) in 2.1230 seconds

修改原 name 列的 VERSIONS 属性为 2。删除刚增加的 tel 列。

2.4.2.4. 步骤四: 删除表

hbase(main):007:0> disable 'student'

0 row(s) in 2.2930 seconds

hbase(main):009:0> drop 'student'

0 row(s) in 1.2530 seconds

HBase(main) :023:0> list

TABLE

0 row(s) in 0.0150 seconds

==>[]

最后可查看数据库状态，包括正在运行的节点，死亡节点等信息。

hbase(main) :025 :0> status

1 active master, 0 backup masters, 2

3. 实验三 HBase 数据操作

3.1. 实验内容与目标

完成本实验，您应该能够：

- 学习 HBase Shell 基本操作。
- 数据的模糊查询
- 批量数据的导入导出

3.2. 相关知识

DML (data manipulation language):

它们是 SELECT、UPDATE、INSERT、DELETE，就象它的名字一样，这 4 条命令是用来对数据库里的数据进行操作的语言

DDL (data definition language):

DDL 比 DML 要多，主要的命令有 CREATE、ALTER、DROP 等，DDL 主要是用在定义或改变表 (TABLE) 的结构，数据类型，表之间的链接和约束等初始化工作上，他们大多在建立表时使用。

3.3. 实验环境

服务器集群	3 个以上节点，节点间网络互通，各节点最低配置：双核 CPU、8GB 内存、100G 硬盘
运行环境	CentOS 7.4
用户名/密码	root/password hadoop/password
服务和组件	HDFS、Yarn、MapReduce

3.4. 实验过程

3.4.1. 实验任务一：简单操作

3.4.1.1. 步骤一：插入数据和修改

建立表 student，两个列簇：name 和 num

```
hbase(main):001:0> create 'student',{NAME=>'name'},{NAME=>'num'}
```

```
hbase(main):002:0> list
```

插入两条数据：

```
hbase(main):003:0> put 'student','rk1','name','Tom'
```

```
hbase(main):004:0> put 'student','rk1','num','123456'
```

```
hbase(main):005:0> put 'student','rk2','name','Sun'
hbase(main):006:0> put 'student','rk2','num','123456'
hbase(main):007:0> put 'student','rk3','name:cha','wangyu'
查看整个表记录
```

修改操作也是用 put 命令，就是重新添加内容把，把以前的内容覆盖。

语法: put <table>,<rowkey>,<family:column>,<value>,<timestamp>。其中, 'table_name' 为表名, 'rk1'为 rowkey, 'name: cha ' name 为列族, cha 为列, 'Tom'为值, 同一个列族下可以有多个列, 同一个 rowkey 视为同一行。

3.4.1.2. 步骤二: 读取指定行、指定行中的列的信息

```
hbase(main):009:0> get 'student','rk1'
COLUMN          CELL
name:           timestamp=1596186987758, value=Tom
num:            timestamp=1596186991960, value=123456
2 row(s) in 0.0630 seconds
```

```
hbase(main):009:0> get 'student','rk1','name'
COLUMN          CELL
name:           timestamp=1596186987758, value=Tom
1 row(s) in 0.0200 seconds
语法: get <table>,<rowkey>,[<family:column>,....]
```

3.4.1.3. 步骤三: scan 命令扫描全表

语法: scan <table>, {COLUMNS => [<family:column>,....], LIMIT => num}

注: 数据导入时, 要注意数据的格式, 否则显示为十六进制。

```
hbase(main):013:0>scan 'student'
ROW              COLUMN+CELL
rk1              column=name:, timestamp=1596186987758, value=Tom
rk1              column=num:, timestamp=1596186991960, value=123456
rk2              column=name:, timestamp=1596186995797, value=Sun
rk2              column=num:, timestamp=1596187000182, value=123456
rk3              column=name:cha, timestamp=1596187003825, value=wang
                  yu
3 row(s) in 0.0520 seconds
```

3.4.1.1. 步骤四: 删除指定行中的列、指定行, 清空表。

```
hbase(main):014:0>delete 'student','rk2','name'
hbase(main):014:0>deleteall 'student','rk2'
```

```
hbase(main):014:0>truncate 'student'
```

Truncating 'student' table (it may take a while):

- Disabling table...

- Truncating table...

0 row(s) in 7.2190 seconds

语法: `delete <table>, <rowkey>, <family:column>, <timestamp>`, 必须指定列名, 这里需要注意, 如果该列保存有多个版本的数据, 将一并被删除。

使用 `deleteall` 命令, 删除 `table_name` 表中 `rowkey002` 这行数据。

语法: `deleteall <table>, <rowkey>, <family:column>, <timestamp>`, 可以不指定列名, 删除整行数据。

使用 `truncate` 命令, 删除 `table_name` 表中的所有数据。

语法: `truncate <table>` 其具体过程是: `disable table -> drop table -> create table`。

3.4.2. 实验任务二: 模糊查询

3.4.2.1. 步骤一: 限制查询

```
hbase(main):003:0> put 'student','rk1','name','Tom'
```

```
hbase(main):004:0> put 'student','rk1','num','123456'
```

```
hbase(main):005:0> put 'student','rk2','name','Sun'
```

```
hbase(main):006:0> put 'student','rk2','num','123456'
```

```
hbase(main):007:0> put 'student','rk3','name:cha','wangyu'
```

```
hbase(main):014:0> scan 'student',{COLUMNS=>'name'}
```

ROW	COLUMN+CELL
rk1	column=name:, timestamp=1596187375037, value=Tom
rk2	column=name:, timestamp=1596187384917, value=Sun
rk3	column=name:cha, timestamp=1596187394040, value=wangyu

3 row(s) in 0.0310 seconds

```
hbase(main):002:0> scan 'student',{COLUMNS=>['name','num'],LIMIT=>2}
```

ROW	COLUMN+CELL
rk1	column=name:, timestamp=1596187375037, value=Tom
rk1	column=num:, timestamp=1596187380066, value=123456
rk2	column=name:, timestamp=1596187384917, value=Sun
rk2	column=num:, timestamp=1596187389628, value=123456

2 row(s) in 0.0160 seconds

语法: `scan <table>,{COLUMNS=>'column'}`

`count` 对表计数时 `INTERVAL`: 每隔多少行显示一次 `count`, 默认是 1000, `CACHE`: 每次去取的缓存区大小, 默认是 10, 调整该参数可提高查询速度, 大表查询通过参数设置可以加快计算速度。

语法: `count <table>, {INTERVAL => intervalNum, CACHE => cacheNum}`

```
hbase(main):002:0> count 'student'
```

```
3 row(s) in 0.6010 seconds
```

```
=> 3
```

3.4.2.2. 步骤二：限制时间范围

```
hbase(main):004:0> scan 'student', {TIMERANGE => [1595397845355,1595397925166]}
```

```
ROW COLUMN+CELL
```

```
0 row(s) in 0.0200 seconds
```

时间戳是 1970 年 01 月 01 日 00 时 00 分 00 秒起至当下的总秒数。通常表示提供一份电子证据，以证明用户的某些数据的产生时间。

3.4.2.3. 步骤三：PrefixFilter:rowKey 前缀过滤

```
hbase(main):005:0> scan 'student',{FILTER=>"PrefixFilter('rk')"}{
```

```
ROW COLUMN+CELL
```

```
rk1 column=name:, timestamp=1596187375037, value=Tom
```

```
rk1 column=num:, timestamp=1596187380066, value=123456
```

```
rk2 column=name:, timestamp=1596187384917, value=Sun
```

```
rk2 column=num:, timestamp=1596187389628, value=123456
```

```
rk3 column=name:cha, timestamp=1596187394040, value=wang
yu
```

```
3 row(s) in 0.0600 seconds
```

同时也有 QualifierFilter:列名过滤器、TimestampsFilter:时间戳过滤器等，支持“且”操作。

ValueFilter:值确定查询（value=Tom）与模糊查询（value 包含 m）

```
hbase(main):005:0> scan 'student',FILTER=>"ValueFilter(=,'binary:Tom')"
```

```
ROW COLUMN+CELL
```

```
rk1 column=name:, timestamp=1596187375037, value=Tom
```

```
1 row(s) in 0.0480 seconds
```

```
hbase(main):006:0> scan 'student',FILTER=>"ValueFilter(=,'substring:m')"
```

```
ROW COLUMN+CELL
```

```
rk1 column=name:, timestamp=1596187375037, value=Tom
```

```
1 row(s) in 0.0270 seconds
```

3.4.3. 实验任务三：批量导入/导出

3.4.3.1. 步骤一：ImportTsv 工具

命令：bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv

Usage: importtsv -Dimporttsv.columns=a,b,c <tablename> <inputdir>

首先数据存入到.csv 文件，上传至 hdfs 服务器中。hbase 调用 MapReduce 服务,当数据量较大时需等待。

```
[hadoop@master ~]$ hdfs dfs -put /opt/software/student.csv /input

[hadoop@master ~]$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
-Dimporttsv.separator="," -Dimporttsv.columns=HBASE_ROW_KEY,name,num student
/input/student.csv

2020-07-22 14:53:25,147 INFO [main] mapreduce.Job: map 0% reduce 0%
2020-07-22 14:53:29,318 INFO [main] mapreduce.Job: map 100% reduce 0%
2020-07-22 14:53:30,328 INFO [main] mapreduce.Job: Job job_1595312762674_0004 completed successfully
2020-07-22 14:53:30,423 INFO [main] mapreduce.Job: Counters: 31
    File System Counters
        FILE: Number of bytes read=0
        FILE: Number of bytes written=148821
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=8795
        HDFS: Number of bytes written=0
        HDFS: Number of read operations=2
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=0
    Job Counters
        Launched map tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=2861
        Total time spent by all reduces in occupied slots (ms)=0
        Total time spent by all map tasks (ms)=2861
        Total vcore-seconds taken by all map tasks=2861
        Total megabyte-seconds taken by all map tasks=2929664
    Map-Reduce Framework
        Map input records=51
        Map output records=14
        Input split bytes=99
        Spilled Records=0
        Failed Shuffles=0
```

图 1-9 MR 操作数据导入

3.4.3.2. 步骤二: Export 数据导出

命令: bin/hbase org.apache.hadoop.hbase.mapreduce.Export

Usage: <tablename> <hdfsdir>

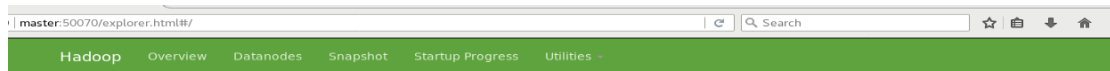
```
[hadoop@master ~]$ cd /usr/local/src/hbase/bin
```

```
[hadoop@master bin]$ hbase org.apache.hadoop.hbase.mapreduce.Export student
/output/hbase-data-back
```

```

2020-07-22 15:02:08,315 INFO [main] mapreduce.Job: map 0% reduce 0%
2020-07-22 15:02:13,375 INFO [main] mapreduce.Job: map 100% reduce 0%
2020-07-22 15:02:14,388 INFO [main] mapreduce.Job: Job job_1595312762674_0006 completed successfully
2020-07-22 15:02:14,518 INFO [main] mapreduce.Job: Counters: 30
    File System Counters
      FILE: Number of bytes read=0
      FILE: Number of bytes written=149107
      FILE: Number of read operations=0
      FILE: Number of large read operations=0
      FILE: Number of write operations=0
      HDFS: Number of bytes read=96
      HDFS: Number of bytes written=6354
      HDFS: Number of read operations=4
      HDFS: Number of large read operations=0
      HDFS: Number of write operations=2
    Job Counters
      Launched map tasks=1
      Rack-local map tasks=1
      Total time spent by all maps in occupied slots (ms)=3041
      Total time spent by all reduces in occupied slots (ms)=0
      Total time spent by all map tasks (ms)=3041
      Total vcore-seconds taken by all map tasks=3041
      Total megabyte-seconds taken by all map tasks=3113984
    Map-Reduce Framework
      Map input records=17
      Map output records=17
      Input split bytes=96
      Spilled Records=0
      Failed Shuffles=0
      Merged Map outputs=0

```



Browse Directory

/								Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
drwxr-xr-x	hadoop	supergroup	0 B	8/9/2020, 12:03:18 PM	0	0 B	hbase	
drwxr-xr-x	hadoop	supergroup	0 B	8/9/2020, 1:00:56 PM	0	0 B	input	
drwxr-xr-x	hadoop	supergroup	0 B	8/9/2020, 1:02:26 PM	0	0 B	output	
drwxrwx---	hadoop	supergroup	0 B	7/31/2020, 3:21:04 PM	0	0 B	tmp	



Browse Directory

/output							Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	hadoop	supergroup	0 B	7/31/2020, 3:21:17 PM	2	128 MB	_SUCCESS
drwxr-xr-x	hadoop	supergroup	0 B	8/9/2020, 1:02:34 PM	0	0 B	hbase-data-back
-rw-r--r--	hadoop	supergroup	25 B	7/31/2020, 3:21:17 PM	2	128 MB	part-r-00000

图 1-10 数据导出与查看