

**大数据处理实验报告**

实验二：HBase的基本操作

专业班级： 计算机2011班

学 号： U202010755

姓 名： 路昊东

指导教师： 郑渤龙

报告日期： 2022年 3 月 24 日

**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 |  |
| 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**  **一. 华为云环境：**  1. 区域：“华北-北京四”   1. MRS集群： 2. 名称：mrs\_dong 3. 集群版本：1.9.2 4. 类型：分析集群 5. 组件：HBase、Hive、Tez 6. 计费模式：按需计费 7. 可用区：可用区2 8. 弹性公网IP绑定：是 9. CPU架构：鲲鹏计算（集群高可用关闭；core节点数：1） 10. 委托、数据盘加密默认，告警“关闭” 11. kerberos认证：关闭 12. 登录方式：密码 13. 虚拟私有云：默认创建 14. 弹性公网 IP： 15. 计费模式：按需计费 16. 线路：全动态 BGP 17. 公网带宽：按流量计费 18. 带宽大小：5M 19. 购买数量：1 20. 安全组规则：master1节点：入方向全部放通 21. **实验的本地软硬件环境如下：** 22. CPU型号：AMD Ryzen 7 4800H 23. CPU主频：2.9GHz 24. 核心/线程数：八核心/十六线程 25. 内存容量：16GB（8GB×2） 26. 操作系统：ubuntu20.04   **四、实验过程或步骤（源程序）**  下面是本次实验的具体内容及步骤、实验的详细记录、实验结果分析：  **1. 实验环境配置**   1. 服务购买： 2. 登录控制台 3. 购买MRS服务 4. 购买弹性公网IP 5. 绑定EIP 6. 创建集群 7. 配置安全组 8. 登录服务器远程master节点 9. 设置环境变量 10. **准备数据** 11. **进入 hbase shell**   **①** 打开**terminal,**使用ssh登录云服务器，命令行输入：  **source env\_file**  **hbase shell**  **②** 终端截图如下：    这里首先使用**source**命令，（通常用命令“.”来替代），代表在当前**bash**环境下读取并执行主目录下的**env\_file**中的命令，之后**hbase shell**命令成功启动**hbase shell**，如图所示：     1. **创建一个表，表名为学号，列族名为 cf1（create）**   **①** 命令行输入：  **Create ‘U202010755’, ’cf1’**  **②** 终端截图如下：    使用create指令，表名U202010755在前，列族cf1在后，成功创建。   1. **显示所有的表（list）**   **①** 命令行输入：  **list**  **②** 终端截图如下：    使用list命令，可见只有一张表名为U202010755的表。   1. **向表中增加两行数据（put）**     **①** 命令行输入：  **put'U202010755','20200001','cf1:name','tom'**  **put'U202010755','20200001','cf1:gender','male'**  **put'U202010755','20200001','cf1:age','20'**  **put'U202010755','20200002','cf1:name','hanmeimei'**  **put'U202010755','20200002','cf1:gender','female'**  **put'U202010755','20200002','cf1:age','19'**  **②** 终端截图如下：    使用**put**指令插入行数据，参数依次为：表名**U202010755**、行键**20200001**或**20200002**、**列族：列名**、以及此列名对应的值。  可见成功创建两行数据，行键为**20200001**和**20200002**，列祖都是**cf1**，列名为**name、gender、age**，并且有对应的值。   1. **查询数据** 2. **查找表中，列族名为 cf1 的数据（scan）**   ① 命令行输入：  **scan 'U202010755', {COLUMNS => 'cf1}**  ② 终端截图如下：    可见成功查找到两行列族为cf1的数据，与上步骤创建的数据相符，证明查找正确。   1. **查找表中，列族名为 cf1,列名为 name 的数据（scan）**   ① 命令行输入：  **scan 'U202010755', {COLUMNS => 'cf1:name'}**  ② 终端截图如下：    可见成功查找到两行数据，与之前创建的相符，证明成功查找。  **ROW COLUMN+CELL**  **20200001 column=cf1:name, timestamp=1648124149930, value=tom**  **20200002 column=cf1:name, timestamp=1648124165889, value=hanmeimei**   1. **查找表中，行键为 20200001 的行（get）**   ① 命令行输入：  **get 'U202010755', '20200001’**  ② 终端截图如下：    可见成功查找到行键为20200001的数据，与之前创建的相符，证明成功查找。  **COLUMN CELL**  **cf1:age timestamp=1648124160552, value=20**  **cf1:gender timestamp=1648124155291, value=male**  **cf1:name timestamp=1648124149930, value=tom**  同时，可以使用scan命令查看，输入：  **scan 'U202010755', {STARTROW => '20200001', STOPROW => '20200001'}**  终端截图如下：    查找出：  **ROW COLUMN+CELL**  **20200001 column=cf1:age, timestamp=1648124160552, value=20**  **20200001 column=cf1:gender, timestamp=1648124155291, value=male**  **20200001 column=cf1:name, timestamp=1648124149930, value=tom**  使用参数**STARTROW**和**STOPROW**，同样实现了查找此行的所有数据的功能。   1. **查找表中，行键为 20200001，列族为 cf1，列名为 name 的数据（get）**   ① 命令行输入：  **get 'U202010755', '20200001', 'cf1:name'**  ② 终端截图如下：    **COLUMN CELL**  **cf1:name timestamp=1648124149930, value=tom**  可见成功查找到数据，值为tom，与创建输入时相同。   1. **查看起始行键为 20200001，终止行键为 20200002(不包括)，限制长度为 2 的数据(scan)**   ① 命令行输入：  **scan 'U202010755', {STARTROW => '20200001', STOPROW => '20200002', LIMIT => 2}**  ② 终端截图如下：    **ROW COLUMN+CELL**  **20200001 column=cf1:age, timestamp=1648124160552, value=20**  **20200001 column=cf1:gender, timestamp=1648124155291, value=male**  **20200001 column=cf1:name, timestamp=1648124149930, value=tom**  可见成功查找到行键为20200001，限制长度为2的数据。   1. **查看有数据值为 20 的行(scan)**   ① 命令行输入：  **scan 'U202010755',{FILTER => "ValueFilter(=, 'binary:20')"}**  ② 终端截图如下：    **ROW COLUMN+CELL**  **20200001 column=cf1:age, timestamp=1648124160552, value=20**  使用filter参数，可见成功查找到数值为20的数据在cf1列族name行中。   1. **查看有数据值为 tom 的行(scan)**   ① 命令行输入：  **scan 'U202010755',{FILTER => "ValueFilter(=, 'binary:tom')"}**  ② 终端截图如下：    **ROW COLUMN+CELL**  **20200001 column=cf1:name, timestamp=1648124149930, value=tom**  可见成功查找到数值为tom的行为行键20200001的行，列族为cf1，列名为name。   1. **查看列名为 gender 的列(scan)**   ① 命令行输入：  **scan 'U202010755', {COLUMNS => 'cf1:gender'}**  ② 终端截图如下：  3  **ROW COLUMN+CELL**  **20200001 column=cf1:gender, timestamp=1648124155291, value=male**  **20200002 column=cf1:gender, timestamp=1648124169847, value=female**  可见成功查找到列名为gender的列，其行键分别为2020001、20200002，列族均为cf1，值分别为male和female。   1. **查看列名为 name，值为 hanmeimei 的行(scan)**   ① 命令行输入：  **scan 'U202010755', {COLUMNS => 'cf1:name',FILTER => "ValueFilter(=, 'binary:hanmeimei')"}**  ② 终端截图如下：    **ROW COLUMN+CELL**  **20200002 column=cf1:name, timestamp=1648124165889, value=hanmeimei**  此处使用scan指令，可见成功查找到列名为 name，值为 hanmeimei 的行,行键为20200002。   1. **查看表的属性（desc）**   ① 命令行输入：  **desc 'U202010755'**  ② 终端截图如下：    终端显示内容为**：**  **Table U202010755 is ENABLED**  **U202010755**  **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', BLOCKSIZ**  **E => '65536', REPLICATION\_SCOPE => '0'}**  可见成功查找到U202010755表的属性与信息，包括列族、版本（版本号为1）等等。  另外使用**describe**命令也能起到相同的作用。   1. **修改数据** 2. **改变表的 VERSIONS 为 5 以显示更多的历史版本（alter）**   ① 命令行输入：  **alter 'U202010755',{NAME => 'cf1', VERSIONS =>5}**  之后为检验操作正误，继续使用desc命令查看表属性：  **desc 'U202010755'**  ② 终端截图如下：    其中修改属性终端显示**：**  **Updating all regions with the new schema...**  **1/1 regions updated.**  **Done.**  显示信息，终端显示**：**  **Table U202010755 is ENABLED**  **U202010755**  **COLUMN FAMILIES DESCRIPTION**  **{NAME => 'cf1', BLOOMFILTER => 'ROW', VERSIONS => '5', IN\_MEMORY => 'false',**  **KEEP\_DELETED\_CELLS => 'FALSE', DATA\_BLOCK\_ENCODING => 'NONE', TTL => 'FOREV**  **ER', COMPRESSION => 'NONE', MIN\_VERSIONS => '0', BLOCKCACHE => 'true', BLOCK**  **SIZE => '65536', REPLICATION\_SCOPE => '0'}**  可见version由之前的1成功变成了想得到的版本5，操作成功。   1. **添加行键 20200001，列族 cf1，列名 name 的多个历史 版本（put）**     ① 命令行输入：  **put 'U202010755','20200001','cf1:name','LiSi'**  **put 'U202010755','20200001','cf1:name','ZhangSan'**  **put 'U202010755','20200002','cf1:name','WangWu'**  ② 终端截图如下：      可见成功插入数据，在**20200001**行中**name**列中插入**lisi**和**zhangsan**，在**20200002**行中**name**列中插入**wangwu**数据。   1. **查看所有行键为 20200001，列簇为 cf1 的多版本数据 （get）**   ① 命令行输入：  **get 'U202010755', '20200001',{COLUMN => 'cf1',VERSIONS =>5}**  ② 终端截图如下：    **COLUMN CELL**  **cf1:age timestamp=1648124160552, value=20**  **cf1:gender timestamp=1648124155291, value=male**  **cf1:name timestamp=1648127098474, value=ZhangSan**  **cf1:name timestamp=1648127006478, value=LiSi**  **cf1:name timestamp=1648124149930, value=tom**  可见成功查找**20200001**行的多版本数据，包括之前插入的和刚插入的数据都有显示。   1. **删除行键为 20200002，列名为 age，的数据（delete）**   ① 命令行输入：  **delete 'U202010755','20200002','cf1:age'**  删除之后需要检验，就查看了20200002行的数据，命令如下：  **get 'U202010755','20200002'**  ② 终端截图如下：    **COLUMN CELL**  **cf1:gender timestamp=1648124169847, value=female**  **cf1:name timestamp=1648127145856, value=WangWu**  delete命令表示删除单项，可见查找到的20200002行的数据被删除了age列的数据，证明删除成功。   1. **删除行键为 20200002 的行（deleteall）**   ① 命令行输入：  **deleteall 'U202010755','20200002'**  删除之后需要检验，就查看了20200002行的数据，命令如下：  **get 'U202010755','20200002'**  ② 终端截图如下：    deleteall命令表示删除整行，可见删除命令输入后再进行查找，发现**20200002**行中没有了任何数据，表明之前删除成功。   1. **删除整个表（disable，drop）**   ① 命令行输入：  **disable 'U202010755'**  **drop 'U202010755'**  ② 终端截图如下：      以下是disable命令之后的信息显示：  **INFO [main] client.HBaseAdmin: Started disable of U202010755**  **INFO [main] client.HBaseAdmin: Disabled U202010755**  以下是drop命令之后的信息显示：  **INFO [main] client.HBaseAdmin: Deleted U202010755**  HBase 使用 **drop** 命令删除表，但是在删除表之前需要先使用 **disable** 命令禁用表。可见成功删除**U202010755**表以及其中的所有内容。  此指令与**truncate**不同，**disable**是禁用表，**drop**是删除，而**truncate**是清空表中的所有数据。   1. **Region 初探**   HBase默认建表时只有一个 region，这个region的 rowkey是没有边界的，即没有 startkey， 也没有 endkey。在数据写入时，所有数据都会写入这个默认的 region，随着数据量的不断增 加，此 region 已经不能承受不断增长的数据量，会进行 split，分成 2 个 region。在此过程 中，会产生两个问题：   1. 数据往一个 region 上写,会有写热点问题。 2. 2.region split 会消耗宝贵的集群 I/O 资源。 基于此我们可以在建表的时候，创建多个空 region，并确定每个 region 的起始和终止 16 rowky，这样只要我们的 rowkey 设计能均匀的命中各个 region，就不会存在写热点问题，自 然 split的几率也会大大降低。hbase 提供了两种 pre-split算法：HexStringSplit和 UniformSplit， 前者适用于十六进制字符的 rowkey，后者适用于随机字节数组的 rowkey。以 rowkey 切分， 随机分为 4 个 region。 3. **创建具有四个 region 的表，表名为”学号\_uniform”，presplit 算法选择 UniformSplit（create）**   ① 命令行输入：  **create 'U202010755\_uniform','cf1',{NUMREGIONS => 4,SPLITALGO => "UniformSplit"}**  ② 终端截图如下：    终端显示如下：  **INFO [main] client.HBaseAdmin: Created U202010755\_uniform**  **=> Hbase::Table - U202010755\_uniform**  使用参数numregions => 4，以及要求的pre split算法，可见成功创建四分区的表。   1. **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（create）**   ① 命令行输入：  **create 'U202010755\_num', 'cf1', {SPLITS=> ['10000000','20000000', '30000000']}**  ② 终端截图如下：    **INFO [main] client.HBaseAdmin: Created U202010755\_num**  **=> Hbase::Table - U202010755\_num**  可见成功创建四分区表。   1. **在 Manager 中查看 HBase**   **① MRS Manager 界面，点击“HBase”服务**  **② 点击 HMaster(主)进入 HBase UI**  **③ “User Tables”下点击创建好的表名**  **④ 查看分区**    **进入managerUI，选择Hbase服务，选择主节点：**    **可以清晰见到创建了新的两张表 U202010755\_uniform和U202010755\_num。**    **这是U202010755\_num的分区，可见创建表的时候就分了行键，**  **观察start key 以及end key，发现已按照创建时行键的划分分区；**  **初始情况所有region的request都为0。**    **这是U202010755\_nun的分区，可见自动生成区边界**  **同样的，初始情况所有region的request都为0。**     1. **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（put）**   **U202010755\_uniform表：**  ① 命令行输入：  **put 'U202010755\_uniform','@\x00\x00\x00\x00\x00\x00\x01', 'cf1:**  **name', 'dong'**  **put 'U202010755\_uniform','\x80\x00\x00\x00\x00\x00\x00\x02', 'cf1: name', 'dong\_2'**  **put 'U202010755\_uniform','\xC0\x00\x00\x00\x00\x00\x00\x00','cf1: name','dong\_3'**  **put 'U202010755\_uniform','\x80\x00\x00\x00\x00\x00\x00\x10','cf1: name','dong\_4'**  **put 'U202010755\_uniform','\x90\x00\x00\x00\x00\x00\x00\x10','cf1: name','dong\_5'**  **put 'U202010755\_uniform','\xB0\x00\x00\x00\x00\x00\x10\x10','cf1: name','dong\_6'**  **put 'U202010755\_uniform','\xC0\x00\x00\x00\x00\x00\x00\x11','cf1: name','dong\_7'**  **put 'U202010755\_uniform','41111111','cf1:my\_name','dong\_9'**  ② 终端截图如下：              **开始没有搞明白十六机制的start key和end key含义，在同一个region创建了很多数据行，加上后台可能出了一些小bug，导致request前后时间段显示不一样，，当然最终成功在不同的region创建了多个request。**  **下面使用scan命令扫描一下这张表，查看创建的数据并进行验证，成功。**    **U202010755\_uniform表：**  ① 命令行输入：  **put 'U202010755\_num','10000055','cf1:name','dong\_1'**  **put 'U202010755\_num','20000055','cf1:name','dong\_2'**  **scan ‘U202010755\_num’**  ② 终端以及nrs主节点后台截图如下所示        **可见成功按照行键，在不同的分区成功插入数据。**   1. **删除所有表**   ① 命令行输入：  **disable 'U202010755\_uniform'**  **drop 'U202010755\_uniform'**  **disable 'U202010755\_num'**  **drop 'U202010755\_num'**  ② 终端截图如下：      **在本地扫描，发现已没有上述表，在mrs后台上查看，同样发现没有创建的表，证明删除成功。**   1. **Hive初探** 2. **准备 file1.txt, 内容为”hello hust”，file2.txt，内容为”hello 学号”（vim）**   ① 命令行输入：  **vim file1.txt**  **vim file2.txt**  ② 终端截图如下：    **file1的编辑界面：**    **file2的编辑界面：**    **使用:wq成功保存两文件。**   1. **将创建的文件移动到 HDFS 中/test 文件夹内（见上次实验）**   ① 命令行输入：  **hdfs dfs -mkdir /test**  **hdfs dfs -put file1.txt /test**  **hdfs dfs -put file2.txt /test**  之后查看一下是否移入/test文件夹内：  **hdfs dfs -ls /test**  ② 终端截图如下：    **Found 2 items**  **-rw-r--r-- 1 root ficommon 11 2022-03-24 23:33 /test/file1.txt**  **-rw-r--r-- 1 root ficommon 18 2022-03-24 23:33 /test/file2.txt**  首先使用类似unix命令的 **-mkdir /test** 创建父目录/test；之后在当前目录使用 **-put** 指令将本地U202010755文件上传至hdfs，之后使用 **-ls** 命令查看/test目录下内容，包括文件名，权限，所有者，大小和修改时间，如上图所示，可见成功移入文件夹。   1. **在 hive 中创建表，tablename 替换为学号U202010755**   ① 命令行输入：  首先启动hive：  **hive**  **create table U202010755(line string);**  ② 终端截图如下：  打开hive如下图所示：    使用sql语句创建名为U202010755的表如下图所示：    可见成功创建。   1. **加载 hdfs 中的数据到 hive 中**   ① 命令行输入：  **load data inpath 'hdfs:///test' overwrite into table U202010755;**  ② 终端截图如下：    可见成功加载数据至hive中。   1. **通过 HiveQL 语句创建词频统计表**   ① 命令行输入：  **create table word\_count as**  **select word,count(1) as count from**  **(select explode(split(line, ' '))as word from tablename) w**  **group by word**  **order by word;**  **即：**  **hive> create table word\_count as**  **> select word,count(1) as count from**  **> (select explode(split(line, ' '))as word from U202010755) w**  **> group by word**  **> order by word;**  ② 终端截图如下：        可见成功创建词频统计表。   1. **通过 HiveQL 语句创建词频统计表**   ① 命令行输入：  **select \* from word\_count;**  ② 终端截图如下：    终端显示内容如下：  **mapred.FileInputFormat: Total input files to process : 1**  **1**  **U202010755 1**  **hello 2**  **hust 1**  可见成功查找到两个文件的词语词频，查找结果为“内容 + 频次”的形式。   1. **实验总结（具体详见下部分）**   本次实验是结合课上的学习与实验的实践，我对于HBase有了更为深入的了解。相比于其它数据库，HBase 解决不了所有的问题，但是针对某些特点的数据可以使用 HBase 高效地解决，比如需要很高的吞吐量，瞬间写入量很大的数据；需要利用多版本来维护的数据（HBase 利用时间戳来区分不同版本的数据）等等。  通过实验，我了解到HBase的一些基本特点：HBase有“列族”的概念，按column-family进行自动分裂，存放不同的服务器的HDFS文件中，这样就支持大容量的水平自动扩展。 其次每个cell没有字段类型，而且里面可以任意增删数据。此外，HBase可以按RowKey进行自动分裂，可以设置手动、自动分裂。通过自动化扩展，HBase可以在廉价的pc服务器上存储大量的数据文件。更有趣的一点，HBase可以自动进行join操作，属于是数据查找的好工具了。  本次实验还使我掌握了一些HBase的基本指令，数据库指令简单来分就是增、删、查、改等。关于增（put）、删（delete和deleteall）和修改（alter等），加上一些对于数据库、表的基本操作比如新建、禁用、删除、查看属性等等都是比较固定简单的操作，主要是关于“查询”功能指令包括scan、get，用法复杂但是颇有趣味，课下也会学习学习，包括背后的实现原理。  最后一项实验中，我发现HBase与Hive有类似的地方，都是管理数据表。但是Hive支持HSQL，是一种类SQL，也真是由于这种机制导致Hive最大的缺点是慢。  总之，本次实验属于是对于我理论学习的一个比较好的实践，我将继续学习，拓展自己。  **五、出现的问题与解决方案**  **问题与解决方案**   1. 本次实验在向’U202010755\_uniform’表中不同region插入数据时，由于没有搞明白pre-spliting中UniformSplit的机制，一连串创建了很多行，但是request都显示在第二个region中，于是搜集了关于pre-spliting的资料，看源码了解了row key的设计：（row key是字节数组arbitrary bytes的时候）某个hbase的表查询只是以随机查询为主，可以用UniformSplit的方式进行，按照原始byte值（从0x00~0xFF）右边以00填充。以这种方式分区的表在插入的时候需要对row key进行一个技巧性的改造， 比如原来的rowkey为rawStr，则需要对其取hashCode，然后进行按照比特位反转后放在最初rowkey串的前面。最终实现在多个region中创建request。   同时对于hbase分区的机制产生了较为浓厚的兴趣，特别时对于预分区相关的知识，我了解到RegionSplitter提供三个用于Pre-splitting的工具：HexStringSplit、UniformSplit、SplitAlgorithm，并且通过查看源码对其机制和特点得到了解。   1. 本次实验仍旧遇到了一些系统问题，比如MRS创建集群20%自动失败、创建70%失败、绑定IP出现问题、间歇性显示类似“无服务”的提示等等，最终通过一次次地重启和等待得到解决。   **六、实验总结**  本次实验同样使我收获颇丰。  首先，也是结合课上的学习与实验的实践，我对于HBase有了更为深入的了解。我们平常在存储数据时，会想到用Mysql[关系型数据库](https://cloud.tencent.com/product/cdb-overview?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)、大硬盘文档存储等。但是，现在存储微信类评论数据、零碎图片，采用Mysql的[数据库](https://cloud.tencent.com/solution/database?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)已经力不从心：现在：Mysql数据库字段固定、字段存储内容无法任意增加或删除、数据库水平扩展麻烦（分库分表依靠人手管理，非常麻烦）等等。当然其海量的数据存取存在瓶颈。面临此类问题，Apache在HDFS的基础上推出了[HBase](https://cloud.tencent.com/product/hbase?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)的NoSQL数据库，解决此类问题。  相比于其它数据库，HBase 解决不了所有的问题，但是针对某些特点的数据可以使用 HBase 高效地解决，比如数据模式是动态的或者可变的，且支持半结构化和非结构化的数据；需要很高的吞吐量，瞬间写入量很大的数据；需要利用多版本来维护的数据（HBase 利用时间戳来区分不同版本的数据）；具有高可扩展性的数据等等。  现在浅谈一下我了解的HBase的特点：HBase有“列族”的概念，比如column-family-1是一个列族，表示人员基本信息，此列族中column-A可能姓名，column-B可能为性别。我们当然还可以再建列族column-family-2，表示人员学籍信息；此外，HBase按column-family进行自动分裂，存放不同的服务器的HDFS文件中，这样就支持大容量的水平自动扩展。 其次每个cell没有字段类型，而且里面可以任意增删数据。我们可以在同一个cell中按时间戳进行任意的数据增加，甚至可以放个视频、图像等大文件。  此外，HBase可以按RowKey进行自动分裂，可以设置手动、自动分裂。通过自动化扩展，HBase可以在廉价的pc服务器上存储大量的数据文件。 对于null空的column，HBase不会进行数据存储，相比存在cell类型的Mysql而言，可以大量节省存储空间。  更有趣的一点，HBase可以自动进行join操作，属于是数据查找的好工具了！！  本次实验还使我掌握了一些HBase的基本指令，数据库指令简单来分就是增、删、查、改等。关于增（put）、删（delete和deleteall）和修改（alter等），加上一些对于数据库、表的基本操作比如新建、禁用、删除、查看属性等等都是比较固定简单的操作，主要是关于“查询”功能指令包括scan、get，用法复杂但是颇有趣味，课下也会学习学习，包括背后的实现原理。  再谈谈HBase的用途吧。和其他数据库相似，用户首先通过client端发起HBase的数据访问请求。Client一般为Java、C++等程序端，并绑定了HBase的开发工具包SDK。Client从ZooKeeper中拿到HBase的HRegionServer的主节点地址，ZooKeeper负责RegionServer的主节点的动态管理、选择。  实验中，我们使用的HMaster是HBase 的主节点，负责整个集群的状态感知、负载分配、负责用户表的元数据管理。  实验中，RegionServer是HBase 中真正负责管理 Region 的服务器，也就是负责为客户端进行表数据读写的服务器。每一台 RegionServer 会管理很多的 Region。并不是同一台RegionServer只能为一个表服务，而是为许多表提供服务。RegionServer将正在服务的Region放入内存中，并最终持久化至HDFS中进行数据的落盘。  实验“region初探”中，Region是 HBase 将一个表中的所有数据按照 RowKey 的不同范围进行切割的逻辑单元，每个 Region 负责一定范围数据的读写访问。该Region在RegionServer中进行管理，并最终存储在HDFS中。  当然，最后一项实验中，我发现HBase与Hive有类似的地方，都是管理数据表。hive可以将结构化的数据文件映射为一张数据库表，并提供完整的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。但是Hive支持HSQL，是一种类SQL，也真是由于这种机制导致Hive最大的缺点是慢。  总之，本次实验属于是对于我理论学习的一个比较好的实践，感谢老师给了我们免费体验实践的机会，让我们学到了有趣的新知识，我将继续学习有关知识，拓展自己，丰富自己的了解，提升学习深度。 | | | | | | | |