Day16_Hbase底层原理及简单操作

大数据-张军锋 Day16 HBase Hbase

Hbase

简介

> HBase(Hadoop Database)是一个开源的、面向列（Column-Oriented）、适合存储海量非结构化数据或半结构化数据的、具备高可靠性、高性能、可灵活扩展伸缩的、支持实时数据读写的分布式存储系统。

存储在HBase中的表的典型特征：

- 大表（BigTable）：一个表可以有上亿行，上百万列
- 面向列：面向列(族)的存储、检索与权限控制
- 稀疏：表中为空(null)的列不占用存储空间

Hbase的应用场景

Hbase适合一次写入，多次读取的应用场景，例如：订单的查询，交易信息，银行流水,话单信息，日志信息

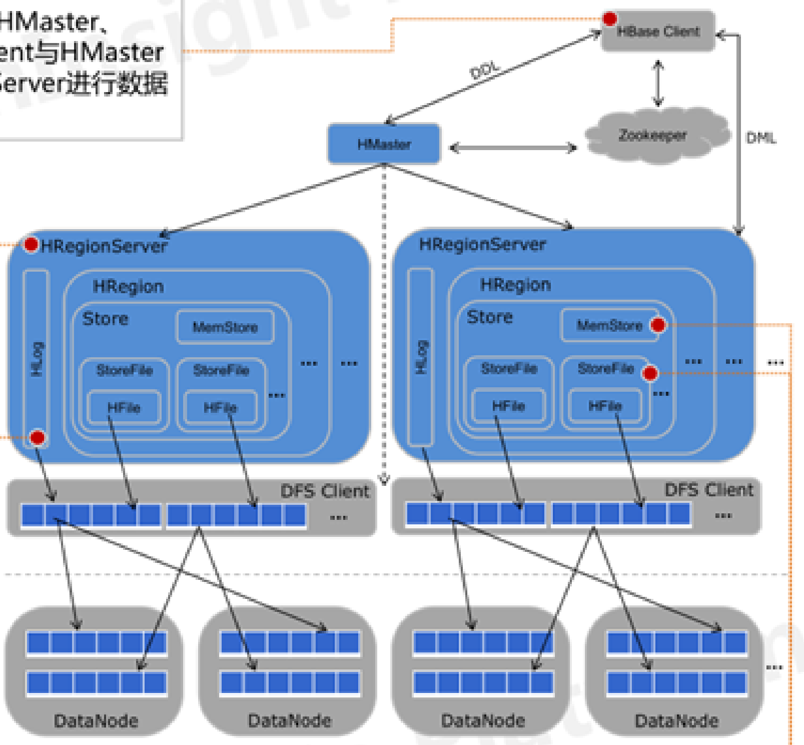
Hbase底层实现

Hbase的底层存储是一个key-value键值对
column Family冗余量比较大，所以强烈建议使用一个字母表示
Row key也是越短越好，但是需要唯一确定

HBase集群典型部署组网

HBase 系统架构

Client使用HBase的RPC机制与HMaster、HRegionServer 进行通信。Client与HMaster进行管理类通信，与HRegion Server进行数据操作类通信。



HRegionServer内部管理了一系列HRegion对象，每个HRegion对应Table中的一个Region。HRegion由多个Store组成。每个Store对应Table中的一个Column Family的存储，即一个Store管理一个Region上的一个列族（CF）。每个Store包含一个MemStore和0到多个StoreFile。Store是HBase的存储核心，由MemStore 和 StoreFile组成。

数据在写入时，首先写入预写日志（Write Ahead Log），每个HRegion Server服务的所有Region的写操作日志都存储在同一个日志文件中。数据并非直接写入HDFS，而是等缓存到一定数量再批量写入，写入完成后在日志中做标记。

MemStore是一个有序的内存缓存区，用户写入的数据首先放入MemStore，当MemStore满了以后Flush成一个StoreFile（存储时对应为HFile）， 当StoreFile数量增到一定阈值，触发Compact合并，将多个StoreFiles合并成一个StoreFile。StoreFiles 合并后逐步形成越来越大的StoreFile，当Region内所有StoreFiles（Hfile）的总大小超过阈值（hbase.hregion.max.filesize）即触发分裂Split，把当前的Region Split成2个Region，父Region下线，新Split出的2个孩子Region被HMaster分配到合适的HRegionServer 上，使得原先1个Region的压力得以分流到2个Region上。

HBase数据模型

存储在HBase表每一行数据都有可排序的关键字（Row Key）和任意列项（Column & Column Family）。在HBase中，仅能通过主键（Row Key）和主键版本号来检索数据，仅支持单行事务。下面以HBase存储搜索引擎的网页为例：

Row Key	**Time Stamp**	**ColumnFamily :** contents	**ColumnFamily :** anchor
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"
"com.cnn.www"	t6	contents:html = "<html>..."	
"com.cnn.www"	t5	contents:html = "<html>..."	
"com.cnn.www"	t3	contents:html = "<html>..."	

行键，相当于关系表的主键，每一行数据的唯一标识。字符串、整数、二进制串都可以作为RowKey。所有记录按照RowKey排序后存储。

每次数据操作对应的时间戳，数据按时间戳区分版本，每个Cell的多个版本的数据按时间倒序存储。

Column Family，列簇，一个表在水平方向上由一个或多个CF组成。一个CF可以由任意多个Column组成。Column是CF下的一个标签，可以在写入数据时任意添加，因此CF支持动态扩展，无需预先定义Column的数量和类型。HBase中表的列非常稀疏，不同行的列的个数和类型都可以不同。此外，每个CF都有独立的TTL（生存周期）。可以只对行上锁，对行的操作始终是原始的。

HBase访问接口

1. Native Java API，最常规和高效的访问方式，适合Hadoop MapReduce Job并行批处理HBase表数据
2. HBase Shell，HBase的命令行工具，最简单的接口，适合HBase管理使用
3. Thrift Gateway，利用Thrift序列化技术，支持C++，PHP，Python等多种语言，适合其他异构系统在线访问HBase表数据
4. REST Gateway，支持REST 风格的Http API访问HBase, 解除了语言限制
5. Pig，可以使用Pig Latin流式编程语言来操作HBase中的数据，和Hive类似，本质最终也是编译成MapReduce Job来处理HBase表数据，适合做数据统计
6. Hive，当前Hive的Release版本尚没有加入对HBase的支持，但在下一个版本Hive 0.7.0中将会支持HBase，可以使用类似SQL语言来访问HBase

Hbase shell

名称	命令表达式
创建表	create '表名称', '列名称1','列名称2','列名称N'
添加记录	put '表名称', '行名称', '列名称:', '值'
查看记录	get '表名称', '行名称'
查看表中的记录总数	count '表名称'
删除记录	delete '表名' ,'行名称' , '列名称'
删除一张表	先要屏蔽该表，才能对该表进行删除，第一步 disable '表名称' 第二步 drop '表名称'
查看所有记录	scan "表名称"
查看某个表某个列中所有数据	scan "表名称" , ['列名称:']
更新记录	就是重写一遍进行覆盖

一般操作

1.查询服务器状态

```
hbase(main):024:0>status
3 servers, 0 dead,1.0000 average load
```

2.查询hive版本

```
hbase(main):025:0>version
0.90.4, r1150278,Sun Jul 24 15:53:29 PDT 2011
```

DDL操作

1.创建一个表

```
hbase(main):011:0>create 'member','member_id','address','info'
0 row(s) in 1.2210seconds
```

2.获得表的描述

```
hbase(main):012:0>list
TABLE
member
1 row(s) in 0.0160seconds
hbase(main):006:0>describe 'member'
DESCRIPTION
ENABLED
{NAME => 'member', FAMILIES => [{NAME=> 'address', BLOOMFILTER =>
'NONE', REPLICATION_SCOPE => '0', true
  VERSIONS => '3', COMPRESSION => 'NONE',TTL => '2147483647', BLOCK
SIZE => '65536', IN_MEMORY => 'fa
lse', BLOCKCACHE => 'true'}, {NAME =>'info', BLOOMFILTER => 'NONE',
REPLICATION_SCOPE => '0', VERSI
ONS => '3', COMPRESSION => 'NONE', TTL=> '2147483647', BLOCKSIZE =>
'65536', IN_MEMORY => 'false',
BLOCKCACHE => 'true'}]}
1 row(s) in 0.0230seconds
```

3.删除一个列族，alter，disable，enable
我们之前建了3个列族，但是发现member_id这个列族是多余的，因为他就是主键，所以我们要将其删除。

```
hbase(main):003:0>alter 'member',{NAME=>'member_id',METHOD=>'delet
e'}
```

ERROR: Table memberis enabled. Disable it first before altering.

报错，删除列族的时候必须先将表给disable掉。

```
hbase(main):004:0>disable 'member'
0 row(s) in 2.0390seconds
hbase(main):005:0>alter'member',NAME=>'member_id',METHOD=>'delete'
0 row(s) in 0.0560seconds
hbase(main):006:0>describe 'member'
DESCRIPTION
ENABLED
{NAME => 'member', FAMILIES => [{NAME=> 'address', BLOOMFILTER =>
'NONE', REPLICATION_SCOPE => '0',false
  VERSIONS => '3', COMPRESSION => 'NONE',TTL => '2147483647', BLOCK
SIZE => '65536', IN_MEMORY => 'fa
lse', BLOCKCACHE => 'true'}, {NAME =>'info', BLOOMFILTER => 'NONE',
REPLICATION_SCOPE => '0', VERSI
ONS => '3', COMPRESSION => 'NONE', TTL=> '2147483647', BLOCKSIZE =>
'65536', IN_MEMORY => 'false',
BLOCKCACHE => 'true'}]}
1 row(s) in 0.0230seconds
```

该列族已经删除，我们继续将表enable

```
hbase(main):008:0> enable 'member'
0 row(s) in 2.0420seconds
```

4.列出所有的表

```
hbase(main):028:0>list
TABLE
member
temp_table
2 row(s) in 0.0150seconds
```

5.drop一个表

```
hbase(main):029:0>disable 'temp_table'
0 row(s) in 2.0590seconds

hbase(main):030:0>drop 'temp_table'
0 row(s) in 1.1070seconds
```

6.查询表是否存在

```
hbase(main):021:0>exists 'member'
Table member doesexist
0 row(s) in 0.1610seconds
```

7.判断表是否enable

```
hbase(main):034:0>is_enabled 'member'
true
0 row(s) in 0.0110seconds
```

8.判断表是否disable

```
hbase(main):032:0>is_disabled 'member'
false
0 row(s) in 0.0110seconds
```

DML操作

1.插入几条记录

```
put'member','scutshuxue','info:age','24'
put'member','scutshuxue','info:birthday','1987-06-17'
put'member','scutshuxue','info:company','alibaba'
put'member','scutshuxue','address:contry','china'
put'member','scutshuxue','address:province','zhejiang'
put'member','scutshuxue','address:city','hangzhou'

put'member','xiaofeng','info:birthday','1987-4-17'
put'member','xiaofeng','info:favorite','movie'
put'member','xiaofeng','info:company','alibaba'
put'member','xiaofeng','address:contry','china'
put'member','xiaofeng','address:province','guangdong'
put'member','xiaofeng','address:city','jieyang'
put'member','xiaofeng','address:town','xianqiao'
```

2.获取一个id的所有数据

```
hbase(main):001:0>get 'member','scutshuxue'
COLUMN                                      CELL
address:city                              timestamp=1321586240244, val
ue=hangzhou
address:contry                            timestamp=1321586239126, val
ue=china
address:province                          timestamp=1321586239197, val
ue=zhejiang
info:age                                  timestamp=1321586238965, val
ue=24
info:birthday                             timestamp=1321586239015, val
ue=1987-06-17
info:company                              timestamp=1321586239071, val
ue=alibaba
6 row(s) in 0.4720seconds
```

3. 获取一个id，一个列族的所有数据

```
hbase(main):002:0>get 'member','scutshuxue','info'
COLUMN                                      CELL
info:age                                  timestamp=1321586238965, val
ue=24
info:birthday                             timestamp=1321586239015, val
ue=1987-06-17
info:company                              timestamp=1321586239071, val
ue=alibaba
3 row(s) in 0.0210seconds
```

4. 获取一个id，一个列族中一个列的所有数据

```
hbase(main):002:0>get 'member','scutshuxue','info:age'
COLUMN                                      CELL
info:age                                  timestamp=1321586238965, val
ue=24
1 row(s) in 0.0320seconds
```

5. 将scutshuxue的年龄改成99

```
hbase(main):004:0>put 'member','scutshuxue','info:age' ,'99'
0 row(s) in 0.0210seconds

hbase(main):005:0>get 'member','scutshuxue','info:age'
COLUMN                                    CELL
info:age                                timestamp=1321586571843, val
ue=99
1 row(s) in 0.0180seconds
```

6. 通过timestamp来获取两个版本的数据

```
hbase(main):010:0>get 'member','scutshuxue',{COLUMN=>'info:age',TIM
ESTAMP=>1321586238965}
COLUMN                                    CELL
info:age                                timestamp=1321586238965, val
ue=24
1 row(s) in 0.0140seconds

hbase(main):011:0>get 'member','scutshuxue',{COLUMN=>'info:age',TIM
ESTAMP=>1321586571843}
COLUMN                                    CELL
info:age                                timestamp=1321586571843, val
ue=99
1 row(s) in 0.0180seconds
```

7. 全表扫描：

```
hbase(main):013:0>scan 'member'
ROW                                        COLUMN+CELL
scutshuxue                                column=address:city, timesta
mp=1321586240244, value=hangzhou
scutshuxue                                column=address:contry, times
tamp=1321586239126, value=china
scutshuxue                                column=address:province, tim
estamp=1321586239197, value=zhejiang
scutshuxue                                 column=info:age,timestamp=1
321586571843, value=99
scutshuxue                                column=info:birthday, timest
amp=1321586239015, value=1987-06-17
scutshuxue                                column=info:company, timesta
mp=1321586239071, value=alibaba
temp                                      column=info:age, timestamp=1
321589609775, value=59
xiaofeng                                  column=address:city, timesta
mp=1321586248400, value=jieyang
xiaofeng                                  column=address:contry, times
tamp=1321586248316, value=china
xiaofeng                                  column=address:province, tim
estamp=1321586248355, value=guangdong
xiaofeng                                  column=address:town, timesta
mp=1321586249564, value=xianqiao
xiaofeng                                  column=info:birthday, timest
amp=1321586248202, value=1987-4-17
xiaofeng                                  column=info:company, timesta
mp=1321586248277, value=alibaba
xiaofeng                                  column=info:favorite, timest
amp=1321586248241, value=movie
3 row(s) in 0.0570seconds
```

8. 删除id为temp的值的'info:age'字段

```
hbase(main):016:0>delete 'member','temp','info:age'
0 row(s) in 0.0150seconds
hbase(main):018:0>get 'member','temp'
COLUMN                                      CELL
0 row(s) in 0.0150seconds
```

9.删除整行

```
hbase(main):001:0>deleteall 'member','xiaofeng'
0 row(s) in 0.3990seconds
```

10. 查询表中有多少行：

```
hbase(main):019:0>count 'member'
2 row(s) in 0.0160seconds
```

11. 给'xiaofeng'这个id增加'info:age'字段，并使用counter实现递增

```
hbase(main):057:0*incr 'member','xiaofeng','info:age'
COUNTER VALUE = 1

hbase(main):058:0>get 'member','xiaofeng','info:age'
COLUMN                                   CELL
info:age                               timestamp=1321590997648, val
ue=\x00\x00\x00\x00\x00\x00\x00\x01
1 row(s) in 0.0140seconds

hbase(main):059:0>incr 'member','xiaofeng','info:age'
COUNTER VALUE = 2

hbase(main):060:0>get 'member','xiaofeng','info:age'
COLUMN                                   CELL
info:age                               timestamp=1321591025110, val
ue=\x00\x00\x00\x00\x00\x00\x00\x02
1 row(s) in 0.0160seconds

获取当前count的值
hbase(main):069:0>get_counter 'member','xiaofeng','info:age'
COUNTER VALUE = 2
```

11. 将整张表清空：

```
hbase(main):035:0>truncate 'member'
Truncating 'member'table (it may take a while):
- Disabling table...
- Dropping table...
- Creating table...
0 row(s) in 4.3430seconds
```

可以看出，hbase是先将掉disable掉，然后drop掉后重建表来实现truncate的功能的。

javaAPI 操作Hbase

初始化构造方法，获取连接

```
private static Connection connection;
    private static Admin admin;
    private static Configuration conf = HBaseConfiguration.creat
e();

    public MemberDataTest() {
        try {
            connection = ConnectionFactory.createConnection(conf);
            admin = connection.getAdmin();
        } catch (IOException e) {
            System.out.println("连接失败");
            e.printStackTrace();
        }
    }
```

关闭连接

```
public void cleanUp() throws Exception {
        connection.close();
    }
```

创建表

```
/**
    * createTable 创建表
    * @param @param tablename 表名
    * @param @param cf 可变参数，列族(Column Family)
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void createTable(String tablename, String... cf) throws
Exception {
        TableName tName = TableName.valueOf(tablename);
        if (!admin.tableExists(tName)) {
            TableDescriptorBuilder aBuilder = TableDescriptorBuilde
r.newBuilder(tName);
            for (String cf1 : cf) {
                ColumnFamilyDescriptor familyDescriptor = ColumnFam
ilyDescriptorBuilder.newBuilder(cf1.getBytes())
                        .build();
                aBuilder.addColumnFamily(familyDescriptor);
            }
            admin.createTable(aBuilder.build());
            System.out.println("create " + tablename + " success");
        } else {
            System.out.println("create " + tablename + "Exceptio
n");
        }
    }
```

列举出数据库下的表

```
/**
    * listDBTables 列举出数据库下的表
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void listDBTables() throws Exception {
        TableName[] tableNames = admin.listTableNames();
        for (TableName tableName : tableNames) {
            System.out.println(tableName);
        }
    }
```

获取表的描述信息

```
/**
    * getTableDesc 获取表的描述信息
    * @param @param tableName
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void getTableDesc(String tableName) throws Exception {
        TableName tName = TableName.valueOf(tableName);
        if (admin.tableExists(tName)) {
            TableDescriptor descriptor = admin.getDescriptor(tNam
e);
            System.out.println(descriptor.toString());
        } else {
            System.out.println(tableName + "no exists");
        }
    }
```

删除表

```
/**
    * dropTable 删除表
    * @param @param tableName
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void dropTable(String tableName) throws Exception {
        TableName tName = TableName.valueOf(tableName);
        if (admin.tableExists(tName)) {
            admin.disableTable(tName);
            admin.deleteTable(tName);
            System.out.println("delete " + tableName + " success");
        }
    }
```

插入数据

```
/**
     * putData 插入数据
     * @param @param tableName 表名称
     * @param @throws Exception 参数
     * @return void 返回类型
     * @Exception 异常对象
     */
    public void putData(String tableName) throws Exception {
        TableName tName = TableName.valueOf(tableName);
        Table table = connection.getTable(tName);
        List<Put> puts = new ArrayList<>();
        Random random = new Random();

        if (admin.tableExists(tName)) {
            for (int i = 0; i < 10; i++) {
                Put put = new Put(("rowKey_" + i).getBytes());
                put.addColumn("info".getBytes(), "name".getBytes(),
("xiao" + i).getBytes());
                put.addColumn("info".getBytes(), "age".getBytes(),
(random.nextInt(50) + 1 + "").getBytes());
                put.addColumn("info".getBytes(), "email".getByte
s(),
                        ((random.nextInt(10000) + 1000) + "@163.co
m").getBytes());
                put.addColumn("address".getBytes(), "city".getByte
s(), "北京".getBytes());
                put.addColumn("address".getBytes(), "town".getByte
s(), "长安街".getBytes());
                puts.add(put);
            }
            table.put(puts);
            System.out.println("data insert over");
        } else {
            System.out.println(tableName + " no exists");
        }
    }
```

获取表中数据

```
/**
    * getData 获取表中数据
    * @param @param tableName 表名称
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void getData(String tableName) throws Exception {
        TableName tName = TableName.valueOf(tableName);
        Table table = connection.getTable(tName);
        List<Get> gets = new ArrayList<>();

        for (int i = 0; i < 5; i++) {
            Get get = new Get(("rowKey_" + i).getBytes());
            get.addColumn("info".getBytes(), "age".getBytes());
            get.addColumn("info".getBytes(), "name".getBytes());
            get.addColumn("info".getBytes(), "email".getBytes());
            get.addColumn("info".getBytes(), "city".getBytes());
            gets.add(get);
        }

        Result[] results = table.get(gets);

        for (Result result : results) {
            CellScanner cellScanner = result.cellScanner();
            while (cellScanner.advance()) {
                Cell cell = cellScanner.current();
                String family = Bytes.toString(CellUtil.cloneFamil
y(cell));
                String quality = new String(CellUtil.cloneQualifie
r(cell));
                String rowKey = new String(CellUtil.cloneRow(cel
l));
                String value = new String(CellUtil.cloneValue(cel
l));
                System.out.println(
                        "rowKey: " + rowKey + " ,family: " + family
+ " ,quality: " + quality + " ,value: " + value);
            }

            /*
            NavigableMap<byte[], NavigableMap<byte[], NavigableMa
p<Long, byte[]>>> maps = result.getMap();
            Set<byte[]> keySet = maps.keySet();
            for (byte[] cf : keySet) {
                NavigableMap<byte[], NavigableMap<Long, byte[]>> na
vigableColumnQualify = maps.get(cf);
                Set<byte[]> keySet2 = navigableColumnQualify.keySe
t();
                for (byte[] columnQualify : keySet2) {
                    NavigableMap<Long, byte[]> navigableMap = navig
ableColumnQualify.get(columnQualify);
                    for (Long ts : navigableMap.keySet()) {
                        byte[] value = navigableMap.get(ts);
                        System.out.println("rowkey: " + Bytes.toStr
ing(result.getRow())+
                                ",columnFamliy :  " + Bytes.toStrin
g(cf)+ ",columnqualify : " + Bytes.toString(columnQualify)+
                                "timestamp: " + new Date(ts) + "val
ue : " + Bytes.toString(value));
                    }
                }
            }
        */
        // 使用字段名称和column Family来获取value的值
            /*System.out.println("rowkey: " + Bytes.toString(resul
t.getRow()) + ",columnFamily: i,columnquality:username,value: " +
                    Bytes.toString(result.getValue(Bytes.toByte
s("i"), Bytes.toBytes("username")))
            );
            System.out.println("rowkey: " + Bytes.toString(result.g
etRow()) + ",columnFamily: i,columnquality:age,value: " +
                    Bytes.toString(result.getValue(Bytes.toByte
s("i"), Bytes.toBytes("age")))
                    );*/
        }
        System.out.println("get data over");
    }
```

删除数据

```
/**
    * deleteData 删除数据
    * @param @param tableName 表名称
    * @param @param rowkey 行键
    * @param @param family 列族
    * @param @param qualifier 行名称
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void deleteData(String tableName, String rowkey, String
family, String qualifier) throws Exception {
        TableName tName = TableName.valueOf(tableName);
        Table table = connection.getTable(tName);
        if (admin.tableExists(tName)) {
            Delete delete = new Delete(rowkey.getBytes());
            delete.addColumn(family.getBytes(), qualifier.getByte
s());
            table.delete(delete);
            System.out.println("delete data success");
        }
    }
```

删除所有数据

```
/**
    * deleteAllData 删除所有数据
    * @param @param tableName 表名称
    * @param @param rowkey
    * @param @throws Exception 参数
    * @return void 返回类型
    * @Exception 异常对象
    */
    public void deleteAllData(String tableName, String rowkey) thro
ws Exception {
        TableName tName = TableName.valueOf(tableName);
        Table table = connection.getTable(tName);
        if (admin.tableExists(tName)) {
            Delete delete = new Delete(rowkey.getBytes());
            table.delete(delete);
            System.out.println("delete data success");
        }
    }
```