HBase数据库

HBase 是建立在HDFS之上，提高高可靠性、高性能、列存储、可伸缩、实时读写的数据系统。

它介于nosql和RDBMS之间，仅能通过主键(row key)和主键的range来检索数据，仅支持单行事务(可通过hive支持来实现多表join等复杂操作)。主要用来存储非结构化和半结构化的松散数据。

HBase中的表一般有这样的特点：

1 大：一个表可以有上亿行，上百万列

2 面向列:面向列(族)的存储和权限控制，列(族)独立检索。

3 稀疏:对于为空(null)的列，并不占用存储空间，因此，表可以设计的非常稀疏。

1.逻辑视图

Row Key	TimeStamp	CF1	CF2	CF3
“RK000001”	T1		CF2:q1=val3	CF3:q2=val2
	T2	CF1:q3=val4		
	T3		CF2:q4=val5	

HBase以表存的形式存储数据，表有行和列，列分为若干个列族(column family)。

Row Key:

```
- 决定一行数据的唯一标识
- Row Key按照字典序排列
- Row Key最多存储64K的数据
```

Column Family和**qualifier**：

- 每个列都属于一个column family，column family必须在定义时给出
- 列名要以列族为前缀，例如CF1:q3
- 权限控制、存储和调优都是在列族层面进行
- HBase把同一列族的数据存在同一目录下，由几个文件保存
- 目前为止HBase能够很好处理最多不超过3个列族

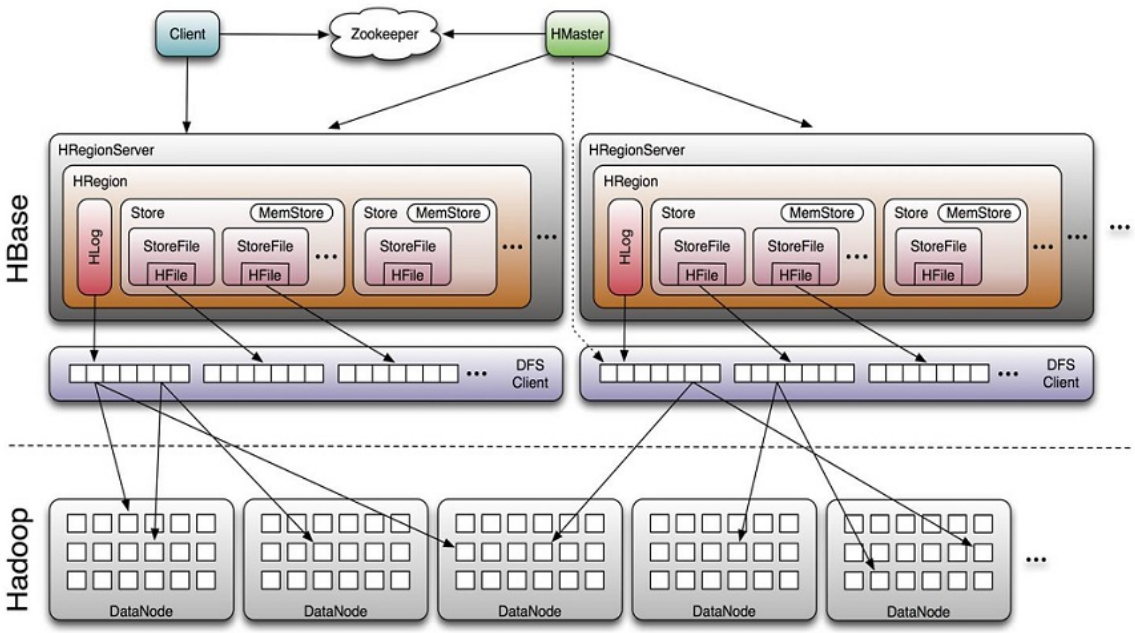
TimeStamp:

- HBase的每一个cell对同一个数据有多个版本，根据TimeStamp来区分版本，不同的版本按时间倒序排序
- 类型为int 64
- TimeStamp可以在数据写入时自动复制，是精确到毫秒的当前系统时间，也可以由用户自己赋值

Cell:

- 单元格的内容是未解析的字节数组（Byte[]），cell中的数据是没有类型的，全部是字节码形式存贮。
- 由{row key，column(=family +qualifier)，version}唯一确定的单元。

2.体系架构



Client:

- 包含访问HBase的接口并维护cache来加快对HBase的访问

Zookeeper:

ZooKeeper是高性能的分布式协作服务和分布式数据一致性解决方案，由雅虎创建，是Goole Chubby的开源实现，可以保证分布式一致性特性，包括顺序一致性、原子性、单一视图（无论客户端连接哪一个ZK服务器看到的都是一样的数据模型）、可靠性、实时性（在一定时间内客户端可以读取到最新数据状态而不是提交后所有服务器马上就全部更新。）

ZooKeeper的数据模型是一个树形节点，服务启动后，所有数据加载到内存中这样来提高服务器吞吐并减少延迟。 在分布式框架中，分布式应用面临的最大的问题就是数据一致性。那么Zookeeper就是一个比较好的解决方案。在分布式框架中起到协调作用。

```
- 保证任何时候集群中只有一个master
- 存储所有Region的寻址入口
- 实时监控Region Server的上线和下线信息，实时通知master
- 存储HBase的schema和table元数据
```

HMaster:

- 为Region Server分配Region
- 负责Region Server的负载均衡
- 发现失效的Region Server并重新分配其上的Region

- 管理Cilent对table的增删改操作

Region Server:

- Region Server维护region，处理对这些region的IO请求
- Region Server负责切分过大的region

HLog:

- HLog是一个Hadoop Sequence File，Sequence File 的Key是 HLogKey对象，HLogKey中记录了写入数据的归属信息：除table和region的名字外，还包括sequence number和timestamp，sequence number的起始值是0或最近一次存入文件系统中的sequence number。
- HLog Sequence File的Value是HBase的KeyValue对象，即对应HFile的KeyValue

Region:

- HBase自动把表划分为多个Region，每个Region会存一个表里某段连续的数据，每个表开始只有一个Region，随着表的增大，Region会增大到一个阈值，然后等分为两个Region

MemStore和**StoreFile**:

- 一个Region由多个Store组成，一个Store对应一个column family
- Store包含位于内存中MemStore和位于磁盘的StoreFile，写操作先写入MemStore，当MemStore中的数据达到某个阈值，HRegion Server会启动flash cache进程写入StoreFile，每次写入形成一个单独的StoreFile
- 当StoreFile文件的数量增长到一定阈值时，系统会进行合并，形成更大的StoreFile
- 当一个region所有storefile的大小和超过一定阈值后，会把当前的region 分割为两个，并由hmaster分配到相应的regionserver服务器，实现负载均衡。
- 客户端检索数据，先在memstore找，找不到再找storefile
- HRegion是HBase中分布式存储和负载均衡的最小单元。最小单元就表示不同的 HRegion可以分布在不同的HRegion server上。
- HRegion由一个或者多个Store组成，每个store保存一个columns family。
- 每个Strore又由一个memStore和0至多个StoreFile组成。

HFile:

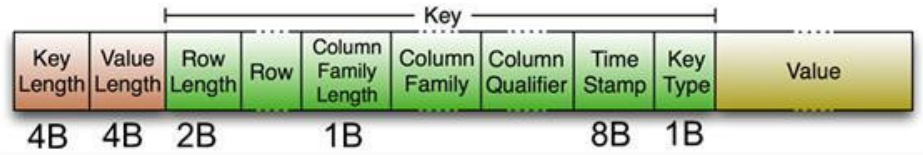
HBase中KeyValue数据的存储格式，是Hadoop的二进制格式文件。 首先HFile文件是不定长的，长度固定的只有其中的两块：Trailer和FileInfo。

Trailer中有指针指向其他数据块的起始点，FileInfo记录了文件的一些meta信息。Data Block是HBase IO的基本单元，为了提高效率，

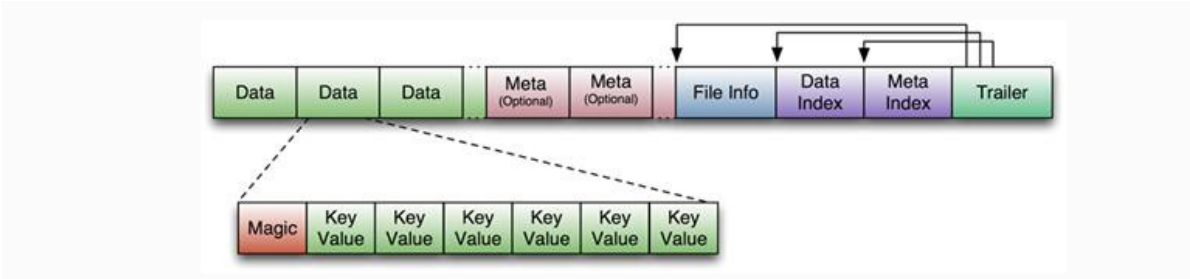
HRegionServer中有基于LRU的Block Cache机制。每个Data块的大小可以在创建一个Table的时候通过参数指定（默认块大小64KB），

大号的Block有利于顺序Scan，小号的Block利于随机查询。每个Data块除了开头的Magic以外就是一个个KeyValue对拼接而成，

Magic内容就是一些随机数字，目的是防止数据损坏，结构如下。



HFile结构图如下：



3.连接前设置

1.修改虚拟机hosts文件

```
vim /etc/hosts
```

将 127.0.0.1 ubuntu 改为 192.168.6.130 ubuntu

2.修改window hosts文件

hosts文件地址为：C:\Windows\System32\drivers\etc

添加 192.168.6.130 ubuntu

4.Shell命令

1. 连接到HBase

```
$ ./bin/hbase shell
hbase(main):001:0>
```

2. 显示HBase Shell帮助文本

键入“help”并按“Enter”，以显示HBase Shell的一些基本用法信息以及几个示例命令。请注意，表名、行、列都必须用引号字符括起来。

3. 创建一个表

使用"create"命令来创建一个表，必须指定表名称和ColumnFamily名称

```
hbase(main):001:0> create 'test', 'cf'
0 row(s) in 0.4170 seconds

=> Hbase::Table - test
```

4. 列出表的信息

通过使用“list”命令来实现

```
hbase(main):002:0> list 'test'
TABLE
test
1 row(s) in 0.0180 seconds

=> ["test"]
```

5. 插入数据

把数据放到表中，请使用“put”命令

```
hbase(main):003:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0850 seconds

hbase(main):004:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0110 seconds

hbase(main):005:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0100 seconds
```

在这里，插入三个值，一次一个。第一个插入是在row1，列cf:a，值为value1。HBase 中的列由列族前缀组成，在此示例中为cf，后跟一个冒号，然后是一个列限定符后缀（在本例中为 a）。

6. 删除数据

使用"delete"命令

```
hbase(main):006:0> delete 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0850 seconds
```

7. 一次扫描表中的所有数据

从HBase获取数据的方法之一是扫描。使用“scan”命令扫描表中的数据。你可以限制你的扫描，但现在，所有的数据都被提取。

```
hbase(main):007:0> scan 'test'
ROW                                   COLUMN+CELL
 row1                                 column=cf:a, timestamp=1421762485768,
value=value1
 row2                                 column=cf:b, timestamp=1421762491785,
value=value2
 row3                                 column=cf:c, timestamp=1421762496210,
value=value3
3 row(s) in 0.0230 seconds
```

8. 清空表的所有记录

使用"truncate"命令

```
hbase(main):008:0> truncate 'test'
```

9. 统计表的行数

使用"count"命令

```
hbase(main):009:0> count 'test'
```

5.Java API

1. 建立连接

```
public static void init(){
    configuration  = HBaseConfiguration.create();
    configuration.set("hbase.rootdir","hdfs://192.168.6.130:9000/opt");
    configuration.set("hbase.zookeeper.quorum","ubuntu：2181")

    try{
        connection = ConnectionFactory.createConnection(configuration);
        admin = connection.getAdmin();
    }catch (IOException e){
        e.printStackTrace();
    }
}
```

2. 关闭连接

```
public static void close(){
    try {
        if (admin != null) {
            admin.close();
        }
        if (null != connection) {
            connection.close();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

3. 创建表

```
public static void createTable(String tableName, String[] fields) throws
IOException {
    init();

    TableName tablename = TableName.valueOf(tableName);

    if (admin.tableExists(tableName)) {
        System.out.println("表已存在");
        admin.disableTable(tablename);
        admin.deleteTable(tablename);//删除原来的表
    }

    HTableDescriptor tableDescriptor = new HTableDescriptor(tableName);
    for(String str:fields){
        HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(str);
        hTableDescriptor.addFamily(hColumnDescriptor);
    }
    admin.createTable(hTableDescriptor);
    close();
}
```

3. 列出表的信息

```
public static void listTables() throws IOException {
    init();//建立连接
    HTableDescriptor hTableDescriptors[] = admin.listTables();
    for(HTableDescriptor hTableDescriptor :hTableDescriptors){
        System.out.println("表名:"+hTableDescriptor.getNameAsString());
    }
    close();//关闭连接
}
```

4. 一次扫描表中的所有数据

```
public static void getData(String tableName) throws  IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Scan scan = new Scan();
    ResultScanner scanner = table.getScanner(scan);
    for (Result result:scanner){
        printRecoder(result);
    }
    close();
}
//打印一条记录的详情
public  static void printRecoder(Result result)throws IOException{
    for(Cell cell:result.rawCells()){
        System.out.print("行健: "+new String(CellUtil.cloneRow(cell)));
        System.out.print("列簇: "+new String(CellUtil.cloneFamily(cell)));
        System.out.print(" 列: "+new String(CellUtil.cloneQualifier(cell)));
        System.out.print(" 值: "+new String(CellUtil.cloneValue(cell)));
        System.out.println("时间戳: "+cell.getTimestamp());
    }
}
```

5. 插入和删除数据

插入数据:

```
public static void insertRow(String tableName,String rowKey,String
colFamily,String col,String val) throws IOException {
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Put put = new Put(rowKey.getBytes());
    put.addColumn(colFamily.getBytes(), col.getBytes(), val.getBytes());
    table.put(put);
    table.close();
    close();
}
```

删除数据:

```
public static void deleteRow(String tableName,String rowKey,String
colFamily,String col) throws IOException {
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Delete delete = new Delete(rowKey.getBytes());
    //删除指定列族
    delete.addFamily(Bytes.toBytes(colFamily));
    //删除指定列
    delete.addColumn(Bytes.toBytes(colFamily),Bytes.toBytes(col));
    table.delete(delete);
    table.close();
    close();
}
```

6. 清空表的记录

```
public static void clearRows(String tableName)throws IOException{
    init();
    TableName tablename = TableName.valueOf(tableName);
    admin.disableTable(tablename);
    admin.deleteTable(tablename);
    HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
    admin.createTable(hTableDescriptor);
    close();
}
```

7. 统计表的行数

```
public static void countRows(String tableName)throws IOException{
    init();
    Table table = connection.getTable(TableName.valueOf(tableName));
    Scan scan = new Scan();
    ResultScanner scanner = table.getScanner(scan);
    int num = 0;
    for (Result result = scanner.next();result!=null;result=scanner.next()){
        num++;
    }
    System.out.println("行数:"+ num);
    scanner.close();
    close();
}
```

6.代码示例：查看表是否存在

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.HColumnDescriptor;
import org.apache.hadoop.hbase.HTableDescriptor;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.Admin;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;
import org.apache.log4j.BasicConfigurator;
```

```
import java.io.IOException;

public class test {
    public static Connection connection=null;
    public static Admin admin=null;
    public static void init() {
        try {
            //1、获取配置信息
            Configuration configuration = HBaseConfiguration.create();
            configuration.set("hbase.rootdir",
"hdfs://192.168.51.190:9000/opt");
            configuration.set("hbase.zookeeper.quorum","ubuntu:2181");//hbase 服
务地址
            //2、创建连接对象
            connection= ConnectionFactory.createConnection(configuration);
            //3、创建Admin对象
            admin = connection.getAdmin();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //判断表是否存在
    public static boolean isTableExiat(String tableName) throws IOException {
        boolean exists = admin.tableExists(TableName.valueOf(tableName));
        return exists;
    }

    public static void close(){
        if (admin!=null){
            try {
                admin.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if (connection!=null){
            try {
                connection.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args) throws IOException {
        BasicConfigurator.configure();
        createTable("Score",new String[]{"sname","course"});
    }

    public static void createTable(String myTableName,String[] colFamily) throws
IOException {

        init();
        TableName tableName = TableName.valueOf(myTableName);

        if(admin.tableExists(tableName)){
            System.out.println("talbe is exists!");
        }else {
```

```
            HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
            for(String str:colFamily){
                HColumnDescriptor hColumnDescriptor = new
HColumnDescriptor(str);
                hTableDescriptor.addFamily(hColumnDescriptor);
            }
            admin.createTable(hTableDescriptor);
            System.out.println("create table success");
        }
        close();
    }
}
```