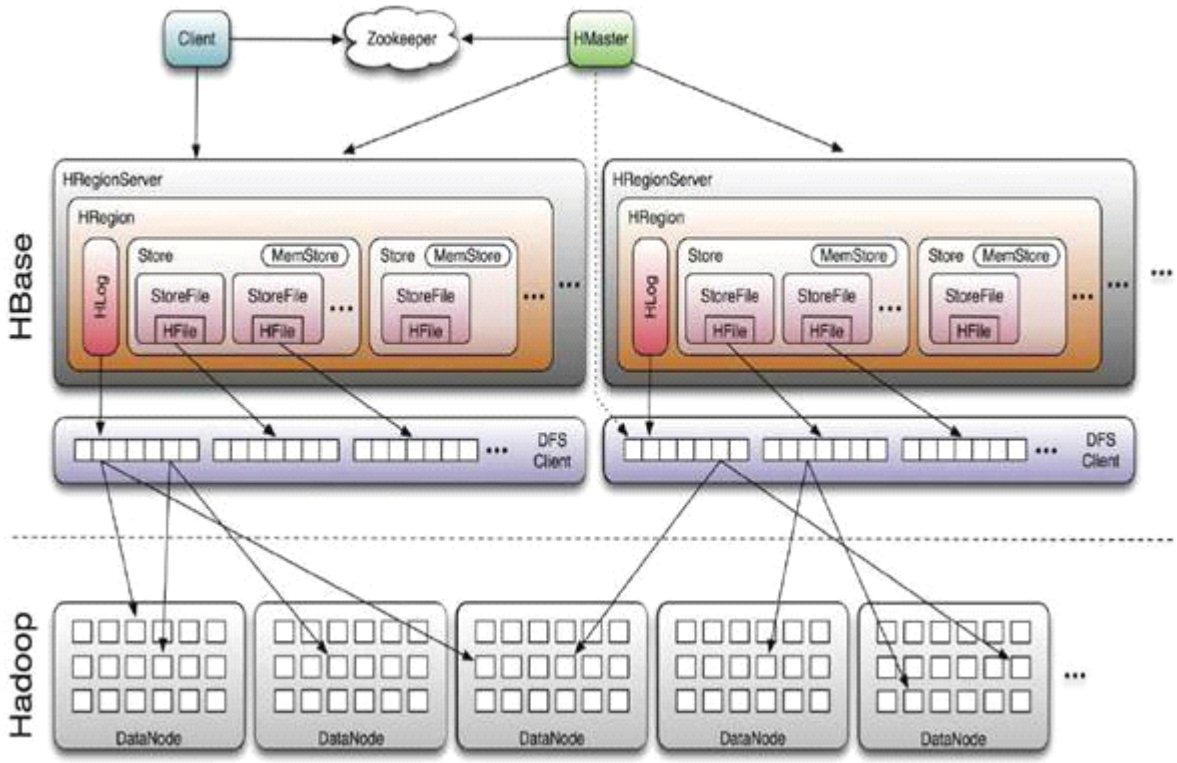
Hbase Java API 详解

HBase 是 Hadoop 的数据库，能够对大数据提供随机、实时读写访问。他是开源的，分布式的，多版本的，面向列的，存储模型。

HBase 的整体结构

在讲解的时候我首先给大家讲解一下 HBase 的整体结构，如下图：



HBase Master 是服务器负责管理所有的 ***HRegion*** 服务器，但不存储 ***HBase*** 服务器的任何数据。

HBase 逻辑上的表可能会划分为多个 ***HRegion***，然后存储在 ***HRegion Server*** 群中。

HBase Master 中存储的是从数据到 ***HRegion Server*** 的映射。

一台机器只能运行一个 ***HRegion*** 服务器，数据的操作会记录在 ***Hlog*** 中，在读取数据时候，***HRegion*** 会先访问 **Hmemcache** 缓存，如果 缓存中没有数据才回到 **Hstore** 中上找，没一个列都会有一个 ***Hstore*** 集合，每个 ***Hstore*** 集合包含了很多具体的 ***HstoreFile*** 文件，这些文件是 B 树结构的，方便快速读取。

再看下 HBase 数据物理视图如下：

Row Key	***Timestamp***	***Column Family***	
		URI	***Parser***
r1	t3	url=http://www.taobao.com	title=天天特价
	t2	host=taobao.com	
	t1		

r2	t5	url=http://www.alibaba.com	content=每天…
	t4	host=alibaba.com	

Row Key: 行键，Table 的主键，Table 中的记录按照 Row Key 排序

Timestamp: 时间戳，每次数据操作对应的时间戳，可以看作是数据的 version number

Column Family: 列簇，Table 在水平方向有一个或者多个 Column Family 组成，一个 Column Family 中可以由任意多个 Column 组成，即 Column Family 支持动态扩展，无需预先定义 Column 的数量以及类型，所有 Column 均以二进制格式存储，用户需要自行进行类型转换。

HBase 具体的 API

HBaseConfiguration 是每一个 hbase client 都会使用到的对象，它代表的是 HBase 配置信息。它有两种构造方式：

public HBaseConfiguration()

public HBaseConfiguration(final Configuration c)

默认的构造方式会尝试从 hbase-default.xml 和 hbase-site.xml 中读取配置。如果 classpath 没有这两个文件，就需要你自己设置配置。

```
Configuration HBASE_CONFIG = new Configuration();
HBASE_CONFIG.set("hbase.zookeeper.quorum", "zkServer");
HBASE_CONFIG.set("hbase.zookeeper.property.clientPort", "2181″);
HBaseConfiguration cfg = new HBaseConfiguration(HBASE_CONFIG);
```

创建表

创建表是通过 HBaseAdmin 对象来操作的。HBaseAdmin 负责表的 META 信息处理。

HBaseAdmin 提供了 createTable 这个方法：

public void createTable(HTableDescriptor desc)

HTableDescriptor 代表的是表的 schema, 提供的方法中比较有用的有

setMaxFileSize，指定最大的 region size

setMemStoreFlushSize 指定 memstore flush 到 HDFS 上的文件大小

增加 family 通过 addFamily 方法

public void addFamily(final HColumnDescriptor family)

HColumnDescriptor 代表的是 column 的 schema，提供的方法比较常用的有

setTimeToLive:指定最大的 TTL,单位是 ms,过期数据会被自动删除。

setInMemory:指定是否放在内存中，对小表有用，可用于提高效率。默认关闭

setBloomFilter:指定是否使用 BloomFilter,可提高随机查询效率。默认关闭

setCompressionType:设定数据压缩类型。默认无压缩。

setMaxVersions:指定数据最大保存的版本个数。默认为 3。

Example

```
HBaseAdmin hAdmin = new HBaseAdmin(hbaseConfig);
HTableDescriptor t = new HTableDescriptor(tableName);
t.addFamily(new HColumnDescriptor("f1″));
```

```
t.addFamily(new HColumnDescriptor("f2"));
t.addFamily(new HColumnDescriptor("f3"));
t.addFamily(new HColumnDescriptor("f4"));
hAdmin.createTable(t);
```

删除表

删除表也是通过 HBaseAdmin 来操作，删除表之前首先要 disable 表。这是一个非常耗时的操作，所以不建议频繁删除表。
disableTable 和 deleteTable 分别用来 disable 和 delete 表。

Example

```
HBaseAdmin hAdmin = new HBaseAdmin(hbaseConfig);
if (hAdmin.tableExists(tableName)) {
        hAdmin.disableTable(tableName);
        hAdmin.deleteTable(tableName);
}
```

查询数据

查询分为单条随机查询和批量查询。
单条查询是通过 rowkey 在 table 中查询某一行的数据。HTable 提供了 get 方法来完成单条查询。
批量查询是通过制定一段 rowkey 的范围来查询。HTable 提供了个 getScanner 方法来完成批量查询。

```
public Result get(final Get get)
public ResultScanner getScanner(final Scan scan)
```

Get 对象包含了一个 Get 查询需要的信息。它的构造方法有两种：

```
public Get(byte [] row)
public Get(byte [] row, RowLock rowLock)
```

Rowlock 是为了保证读写的原子性，你可以传递一个已经存在 Rowlock，否则 HBase 会自动生成一个新的 rowlock。
Scan 对象提供了默认构造函数，一般使用默认构造函数。

Get/Scan 的常用方法有：

addFamily/addColumn:指定需要的 family 或者 column,如果没有调用任何 addFamily 或者 Column,会返回所有的 columns.
setMaxVersions:指定最大的版本个数。如果不带任何参数调用 setMaxVersions,表示取所有的版本。如果不掉用 setMaxVersions,只会取到最新的版本。
setTimeRange:指定最大的时间戳和最小的时间戳，只有在此范围内的 cell 才能被获取。
setTimeStamp:指定时间戳。
setFilter:指定 Filter 来过滤掉不需要的信息

Scan 特有的方法：

setStartRow:指定开始的行。如果不调用，则从表头开始。
setStopRow:指定结束的行（不含此行）。
setBatch:指定最多返回的 Cell 数目。用于防止一行中有过多的数据，导致 OutofMemory 错误。
ResultScanner 是 Result 的一个容器，每次调用 **ResultScanner** 的 next 方法，会返回 **Result**.

```
public Result next() throws IOException;
public Result [] next(int nbRows) throws IOException;
```

Result 代表是一行的数据。常用方法有：
getRow:返回 rowkey
raw:返回所有的 key value 数组。
getValue:按照 column 来获取 cell 的值

Example

```
Scan s = new Scan();
s.setMaxVersions();
ResultScanner ss = table.getScanner(s);
for(Result r:ss){
    System.out.println(new String(r.getRow()));
    for(KeyValue kv:r.raw()){
        System.out.println(new String(kv.getColumn()));
    }
}
```

插入数据

HTable 通过 put 方法来插入数据。

```
public void put(final Put put) throws IOException
public void put(final List puts) throws IOException
```

可以传递单个批 Put 对象或者 List put 对象来分别实现单条插入和批量插入。
Put 提供了 3 种构造方式：

```
public Put(byte [] row)
public Put(byte [] row, RowLock rowLock)
public Put(Put putToCopy)
```

Put 常用的方法有：

add:增加一个 Cell
setTimeStamp:指定所有 cell 默认的 timestamp,如果一个 Cell 没有指定 timestamp,就会用到这个值。如果没有调用，HBase 会将当前时间作为未指定 timestamp 的 cell 的 timestamp.
setWriteToWAL: WAL 是 Write Ahead Log 的缩写，指的是 HBase 在插入操作前是否写 Log。默认是打开，关掉会提高性能，但是如果系统出现故障(负责插入的 Region Server 挂掉)，数

据可能会丢失。
另外 HTable 也有两个方法也会影响插入的性能
setAutoFlash: AutoFlush 指的是在每次调用 HBase 的 Put 操作，是否提交到 HBase Server。默认是 true,每次会提交。如果此时是单条插入，就会有更多的 IO,从而降低性能.
setWriteBufferSize: Write Buffer Size 在 AutoFlush 为 false 的时候起作用，默认是 2MB,也就是当插入数据超过 2MB,就会自动提交到 Server

Example

```
HTable table = new HTable(hbaseConfig, tableName);
table.setAutoFlush(autoFlush);
List lp = new ArrayList();
int count = 10000;
byte[] buffer = new byte[1024];
Random r = new Random();
for (int i = 1; i <= count; ++i) {
        Put p = new Put(String.format("row%09d",i).getBytes());
        r.nextBytes(buffer);
        p.add("f1".getBytes(), null, buffer);
        p.add("f2".getBytes(), null, buffer);
        p.add("f3".getBytes(), null, buffer);
        p.add("f4".getBytes(), null, buffer);
        p.setWriteToWAL(wal);
        lp.add(p);
        if(i%1000==0){
            table.put(lp);
            lp.clear();
        }
}
```

删除数据

HTable 通过 delete 方法来删除数据。
public void delete(final Delete delete)

Delete 构造方法有：

public Delete(byte [] row)
public Delete(byte [] row, long timestamp, RowLock rowLock)
public Delete(final Delete d)

Delete 常用方法有：

deleteFamily/deleteColumns:指定要删除的 family 或者 column 的数据。如果不调用任何这样的方法，将会删除整行。
注意：如果某个 Cell 的 timestamp 高于当前时间，这个 Cell 将不会被删除，仍然可以查出来。

Example

```
HTable table = new HTable(hbaseConfig, “mytest”);
Delete d = new Delete(“row1″.getBytes());
table.delete(d)
```

切分表

HBaseAdmin 提供 split 方法来将 table 进行 split.

```
public void split(final String tableNameOrRegionName)
```

如果提供的 tableName，那么会将 table 所有 region 进行 split ;如果提供的 region Name，那么只会 split 这个 region.

由于 split 是一个异步操作，我们并不能确切的控制 region 的个数。

Example

```
public void split(String tableName,int number,int timeout) throws Exception {
    Configuration HBASE_CONFIG = new Configuration();
    HBASE_CONFIG.set(“hbase.zookeeper.quorum”, GlobalConf.ZOOKEEPER_QUORUM);
    HBASE_CONFIG.set(“hbase.zookeeper.property.clientPort”,
GlobalConf.ZOOKEEPER_PORT);
    HBaseConfiguration cfg = new HBaseConfiguration(HBASE_CONFIG);
    HBaseAdmin hAdmin = new HBaseAdmin(cfg);
    HTable hTable = new HTable(cfg,tableName);
    int oldsize = 0;
    t =    System.currentTimeMillis();
    while(true){
        int size = hTable.getRegionsInfo().size();
        logger.info(“the region number=”+size);
        if(size>=number ) break;
        if(size!=oldsize){
            hAdmin.split(hTable.getTableName());
            oldsize = size;
        }           else if(System.currentTimeMillis()-t>timeout){
            break;
        }
        Thread.sleep(1000*10);
    }
}
```