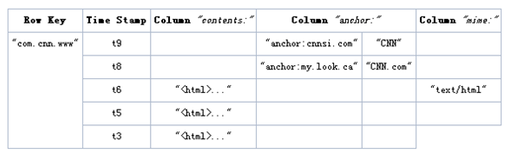
# 表，列，行和单元格

HBase表的特点：

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）独立索引
3. 稀疏：对于空（null）的列，不占用存储空间

Hbase最基本 的单位是列(column)

存储的逻辑视图



* Row（行）

行键总是唯一的；

row key是用来检索记录的主键，通过三种方式访问：

1.通过单个row key 访问

2.通过row key进行range访问

3.全表扫描

Row key可以是任意字符串（最大64K），在HBASE内部，rowkey作为字节数组保存，在排序时，也是根据字节数组（byte order）排序，设计key时要利用这个特性，将经常一区读取的行存放在一起（例如可以将一类数据的rowkey首部都使用相同的字符串做标识）

行的读写总是原子操作（无论读写多少列），天然支持并发

* Column（列）

HBase的每个列（column）都属于一个列族（column family），列族是chema的一部分，必须在表之前定义，并且不能改的太频繁，数量也不能多（2位数以内，实际更少），列族可以动态扩展。

列名总是以列族作为前缀，例如 course:history就属于course列族，列名必须是能够被打印的字符串

列族有助于构建予以边界和局部边界，数据的访问控制，磁盘和内存的使用都是在列族层面进行的，还有助于设置某些特性（如压缩），一个列族的所有列存储在一个底层文件（HFile）中。

每个列族下列的数量无限制，列值也没有类型和长度限制

* Cell（单元格）

HBase中通过row和columns确定的一个存储单元称作cell。

Cell中的数据没有类型，全部使用字节数组存储。

HBase提供两种数据版本回收方式，一是保存数据的最后若干个版本，而是保存最近一段时间的版本（比如最近7天），用户针对每个列族进行配置

* Timestamp（时间戳）

每个cell都能有多个版本，版本通过时间戳（timestamp）索引，时间戳默认系统指定，也可以用户定义。时间戳能被使用，例如用时间戳来区分不同版本的值，时间戳是64位整型。每个cell中，不同的版本按时间戳降序排列，最新的数据排在前面。

也就是每个cell中存储的数据如此确定：

(Table,RowKey ,Column,Timestamp)-🡪value，由行表，行号，列族，列，版本唯一确定一个值

普通数据库的存储

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Column A  (int) | Column B  (varchar) | Column C  (boolean) | Column D  (date) |
| Row A |  |  |  |  |
| Row B |  |  |  |  |
| Row C |  |  | Null |  |
| Row D |  |  |  |  |

HBase采用了标签描述存储，就是信息保存在一个特定标签下



逻辑结构

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Row key | Time stamp | Column  “data:” | Column  “meta:mine” | Column  “meta:size” |
| “row1” | T3 | “{123}” |  |  |
| T6 | “{456}” |  | 2333 |
| T7 | “{789}” | “json” |  |
| T9 | “{321}” |  |  |

如图，表示的是1行(row1)和3列(data:,meta:mine,meta:size)组合成的3个单元格，每个单元格当前有4个版本（时间戳T3，T6，T7，T9）,其中meta:mine和meta:size同属列族meta

例如取出行”row1”的”meta:mine”的T7版本的数据，值就是”json”

# API

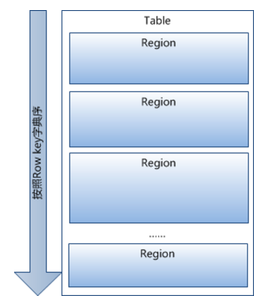
BigTable不支持完成的关系数据模型，它提供简单的数据模型客户端

API提供了建表，删表，增加列族和删除列族的操作；还提供了元素的相关操作。

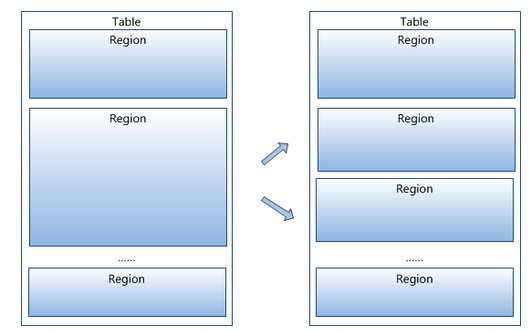
Scan API提供了高效遍历某个范围的行的操作。

# 物理存储

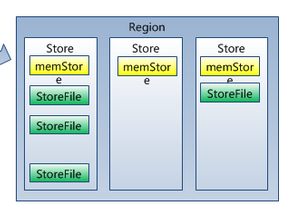
* Table中所有行都是按row key进行排序，Table在行的方向上划分region，它本质上是以行健排序的连续存储区间，每个表最开始只有一个region：



* Region会自动拆分或合并，当表不断扩大，到达一个阈值就会在中间处分为2个大小相等的region，当数据量更大就会划分更多的region；反之就会将多个region组合为一个：



* HBase中扩展和负载均衡的基本单元称为region，每个region只能被一个region服务器加载，每个region服务器可以加载多个region。Region不可被拆分加载。
* Region是分布式存储的最小单元，但不是存储的最小单元。Region由多个store组成，每个store保存一个列族；每个store由一个memstore（内存）和多个sotre file（存储文件）组成：



* Store File是存储的最小单元，它以HFile的格式存储在HDFS（Hadoop分布式文件系统）上它存储的是经过排序的键值映射结构。文件内部由连续的块组成，块的索引信息在文件尾部。每次打开HFile文件，块索引会先被加载到内存中，在查找指定键时，通过在索引中的二分检索，找到可能包含了给定键的块，然后读取磁盘找到实际要找的键。

# 关键算法/流程

Region定位

Bigtable使用三层类似B+树的结构用来保存region位置。

* 读写过程

HBase使用MemStore和StoreFile对表进行更新。

数据在更新时首先写入Log(WAL Log)和内存（memstore）中，MemStore中的数据是排序的，当它的数据量到达一定阈值，就会创建一个新的memstore，并且将老的memstore添加到flush队列，由单独的线程flush到磁盘上，成为一个StoreFile。

当系统出现意外，导致memstore中的数据丢失，就可以使用log来回复checkpoint之后的数据。

SotreFile是只读文件，一旦创建无法修改，因此Hbase的更新其实是不断追加的操作，当一个Store中的SotreFile到达一定阈值后，就会进行一次合并，将对同一个key的修改合并到一处，形成一个大的StoreFile。

由于表的更新不断追加，处理读请求时，需要访问Store中全部的StoreFile和memStore，将它们按照rowkey进行合并，由于StoreFile和MemStore都经过排序，并且StoreFile中带有内存索引，所以效率很高。

# 客户端基础

所有修改数据的操作都保证了 行级别 的原子性

## Put方法（添加数据）

每一个put操作实际就是一个RPC（远程过程调用）操作，只适合少量

# 其他

* 详解

<http://my.oschina.net/mkh/blog/349866>

* 实例

<http://www.myexception.cn/c-sharp/1670107.html>