# 特点概述

HBase中的表一般有这样的特点：

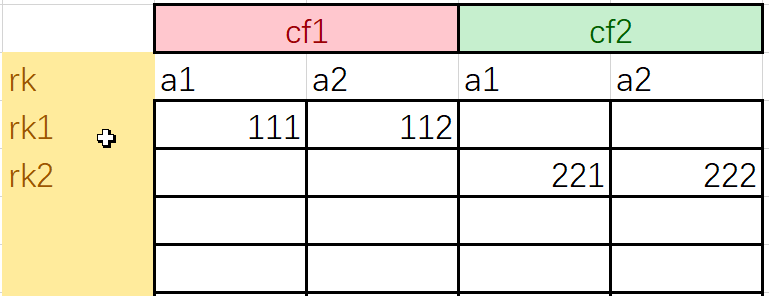
1 大：一个表可以有上亿行，上百万列

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。

# ****逻辑视图****

HBase以表的形式存储数据。表由行和列族(row family)组成。列族划分为若干个列 。



**Row Key**

与nosql数据库们一样,row key是用来检索记录的主键。访问hbase table中的行，只有三种方式：

1 通过单个row key访问

2 通过row key的range

3 全表扫描

Row key行键可以是任意字符串(最大长度是 64KB，实际应用中长度一般为 10-100bytes)，在hbase内部，row key保存为字节数组。

存储时，数据按照Row key的字典顺序(byte order)排序存储。设计key时，要充分发挥字典排序存储这个特性，将经常一起读取的行存储放到一起。(位置相关性)

注意：

字典序对int排序的结果是要保持整形的自然序，行键必须用0作左填充。

例：01,02,03,04,05,06,07,08,09,10,11,12,13……

**列族**

hbase表中的每个列，都归属于某个列族。列族是表的schema的一部分(而列不是)，必须在使用表之前定义。列名都以列族作为前缀。例如courses:history ， courses:math 都属于 courses 这个列族。

**时间戳**

HBase中通过row和columns确定的为一个存贮单元称为cell。每个 cell都保存着同一份数据的多个版本。版本通过时间戳来索引。时间戳的类型是 64位整型。时间戳可以由hbase(在数据写入时自动 )赋值，此时时间戳是精确到毫秒的当前系统时间。时间戳也可以由客户显式赋值。如果应用程序要避免数据版本冲突，就必须自己生成具有唯一性的时间戳。每个 cell中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。

为了避免数据存在过多版本造成的的管理 (包括存贮和索引)负担，hbase提供了两种数据版本回收方式。一是保存数据的最后n个版本，二是保存最近一段时间内的版本（比如最近七天）。用户可以针对每个列族进行设置。

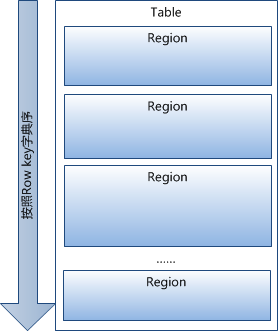
**Cell**

由{row key, column( =<family> + < label>), version} 唯一确定的单元。cell中的数据是没有类型的，全部是字节码形式存贮。

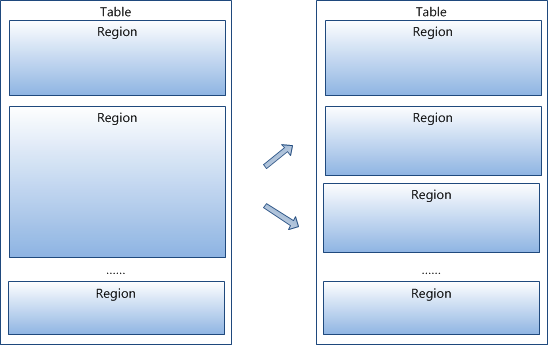
# ****物理存储****

1 已经提到过，Table中的所有行都按照row key的字典序排列。

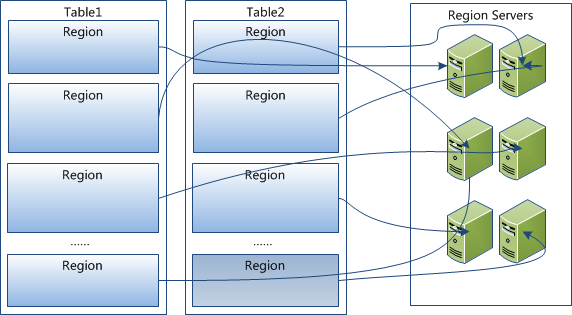
2 Table 在行的方向上分割为多个Hregion。



3 region按大小分割的，每个表一开始只有一个region，随着数据不断插入表，region不断增大，当增大到一个阀值的时候，Hregion就会等分为两个新的Hregion。当table中的行不断增多，就会有越来越多的Hregion。



4 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。但一个Hregion是不会拆分到多个server上的。

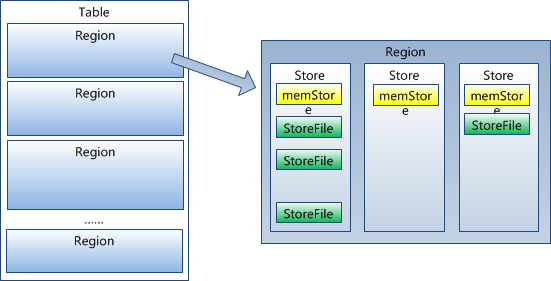


5 HRegion虽然是分布式存储的最小单元，但并不是存储的最小单元。

事实上，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。如图：

StoreFile以HFile格式保存在HDFS上。



**HLog(WAL log)**

WAL 意为Write ahead log(http://en.wikipedia.org/wiki/Write-ahead\_logging)，类似mysql中的binlog,用来 做灾难恢复时用，Hlog记录数据的所有变更,一旦数据修改，就可以从log中进行恢复。

每个Region Server维护一个Hlog,而不是每个Region一个。这样不同region(来自不同table)的日志会混在一起，这样做的目的是不断追加单个 文件相对于同时写多个文件而言，可以减少磁盘寻址次数，因此可以提高对table的写性能。带来的麻烦是，如果一台region server下线，为了恢复其上的region，需要将region server上的log进行拆分，然后分发到其它region server上进行恢复。

HLog文件就是一个普通的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，除了table和region名字外，同时还包括 sequence number和timestamp，timestamp是”写入时间”，sequence number的起始值为0，或者是最近一次存入文件系统中sequence number。HLog Sequece File的Value是HBase的KeyValue对象，即对应HFile中的KeyValue，可参见上文描述。

# 读写过程

上文提到，hbase使用MemStore和StoreFile存储对表的更新。

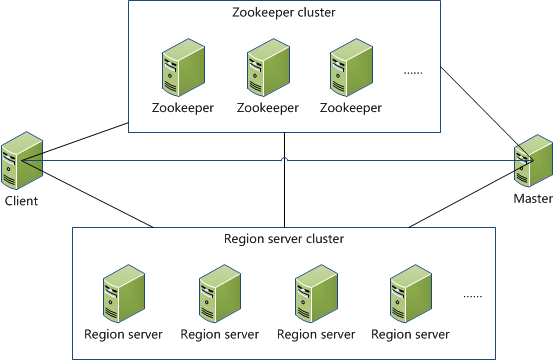
数据在更新时首先写入Log(WAL log)和内存(MemStore)中，MemStore中的数据是排序的，当MemStore累计到一定阈值时，就会创建一个新的MemStore，并 且将老的MemStore添加到flush队列，由单独的线程flush到磁盘上，成为一个StoreFile。于此同时，系统会在zookeeper中 记录一个redo point，表示这个时刻之前的变更已经持久化了。(minor compact)

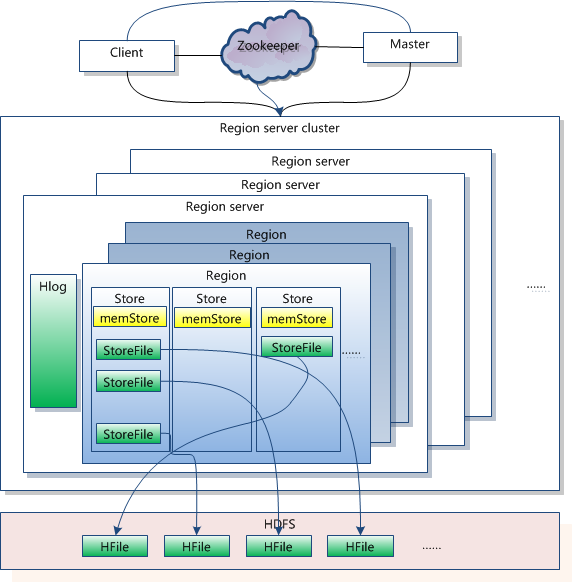
当系统出现意外时，可能导致内存(MemStore)中的数据丢失，此时使用Log(WAL log)来恢复redo point之后的数据。

StoreFile是只读的，一旦创建后就不可以再修改。因此Hbase的更 新其实是不断追加的操作。当一个Store中的StoreFile达到一定的阈值后，就会进行一次合并(major compact),将对同一个key的修改合并到一起，形成一个大的StoreFile，当StoreFile的大小达到一定阈值后，又会对 StoreFile进行split，等分为两个StoreFile。

由于对表的更新是不断追加的，处理读请求时，需要访问Store中全部的 StoreFile和MemStore，将他们的按照row key进行合并，由于StoreFile和MemStore都是经过排序的，并且StoreFile带有内存中索引，合并的过程还是比较快。

# ****系统架构****





**Client**

1 包含访问hbase的接口，client维护着一些cache来加快对hbase的访问，比如regione的位置信息。

**Zookeeper**

1 保证任何时候，集群中只有一个master

2 存贮所有Region的寻址入口。

3 实时监控Region Server的状态，将Region server的上线和下线信息实时通知给Master

4 存储Hbase的schema,包括有哪些table，每个table有哪些column family

**Master**

1 为Region server分配region

2 负责region server的负载均衡

3 发现失效的region server并重新分配其上的region

4 HDFS上的垃圾文件回收

5 处理schema更新请求

**Region Server**

1 Region server维护Master分配给它的region，处理对这些region的IO请求

2 Region server负责切分在运行过程中变得过大的region

可以看到，client访问hbase上数据的过程并不需要master参与（寻址访问zookeeper和region server，数据读写访问regione server），master仅仅维护者table和region的元数据信息，负载很低。