

---

# Hbase

---

Outline

HBase基础

【实践】HBase搭建

【实践】Hbase Shell

【实践】Hbase的Python操作

## Hbase 定义

- HBase是一个开源的**非关系型分布式数据库** (NoSQL) , 它参考了谷歌的**BigTable**建模, 实现的编程语言为 Java。  
HBase的底层实现竟然是java, java牛逼了
- 是Apache软件基金会的Hadoop项目的一部分, 运行于HDFS文件系统之上, 因此可以容错地存储海量稀疏的数据。  
HBase有少量数据存储在内存中, 可以快速随机访问?
- 特性:
  - **高可靠**  
HDFS的多副本机制
  - **高并发读写**  
这部分是基于 HBASE有部分少量在内存中的数据。每次记录的修改, 以时间戳追加的方式都存储下来。
  - **面向列**  
列存储  
rowKey---->Column Family---->Column Qualifier  
数据是存在在Qualifier上的
  - 可伸缩
  - 易构建

## 行存储 vs 列存储

- 行存储：
  - 优点：写入一次性完成，保持数据完整性
  - 缺点：数据读取过程中产生冗余数据，若有少量数据可以忽略
- 列存储
  - 优点：读取过程，不会产生冗余数据，特别适合对数据完整性要求不高的大数据领域
  - 缺点：写入效率差，保证数据完整性方面差

## Hbase 优势

- 海量数据存储
- 快速随机访问
- 大量写操作的应用。

这个怎么体现????  
使用了什么机制??  
莫非是因为部分数据  
在内存中.  
因为有BlockCache的  
存在么

前面不是说写效率差么?为什么还要大量写操作额应用????  
因为MemoryStore的存在, client在往Hbase写数据的时候先写入到MemoryStore, 然后MemoryStore在异步写入到StoreFile. 所以快. 所以适合写?

## H b a s e 应 用 场 景

- 互联网搜索引擎数据存储
- 海量数据写入
- 消息中心
- 内容服务系统 (schema-free)
- 大表复杂&多维度索引
- 大批量数据读取

## Hbase 数据模型

行键	时间戳	列族contens	列族anchor	列族mime
"com.cnn.www"	t9		anchor:cnnsi.com="CNN"	
	t8		anchor:my.look.ca="CNN.com"	
	t6	contens:html=""		mime:type="text/html"
	t5	contens:html=""		
	t3	contens:html=""		

- RowKey: 是Byte array, 是表中每条记录的“主键”, 方便快速查找, Rowkey的设计非常重要。
- Column Family: 列族, 拥有一个名称(string), 包含一个或者多个相关列
- Column: 属于某一个columnfamily, familyName:columnName, 每条记录可动态添加
- Version Number: 类型为Long, 默认值是系统时间戳, 可由用户自定义
- Value(Cell): Byte array

不同维度的属性 可以设计为不同的列族  
比如用户的基本属性 姓名 年龄 ... 作为一个列族  
喜欢的爱好 可以作为一个列族

## Hbase 数据模型

Table A

rowkey	column family	column qualifier	timestamp	value
a	cf1	"bar"	1368394583	7
			1368394261	"hello"
		"foo"	1368394583	22
			1368394925	13.6
	cf2	"2011-07-04"	1368393847	"world"
	cf2	"1.0001"	1368396302	"fourth of July"
			1368387684	"almost the loneliest number"
b	cf2	"thumb"	1368387247	[3.6 kb png data]



## Hbase 数据模型

数据存储的时候 从三个维度进行排序

1. rowKey 正序
2. column 正序 (包含 columnFamily 和 columnQualifier)
3. 时间戳 倒叙

三维有序!

Table A

Column Families

Rows

rowkey	column family	column qualifier	timestamp	value
a	cf1	"bar"	1368394583	7
			1368394261	"hello"
		"foo"	1368394583	22
			1368394925	13.6
	cf2	"2011-07-04"	1368393847	"world"
			1368396302	"fourth of July"
b	cf2	"thumb"	1.0001	"almost the loneliest number"
			1368387247	[3.6 kb png data]

• {rowkey => {family => {qualifier => {version => value}}}}

• a:cf1:bar:1368394583:7

• a:cf1:foo:1368394261:hello

八斗大数据内部资料，盗版必究——

## Hbase 数据模型

Rowkey	Column Family	Column Qualifier	Timest amp	Value

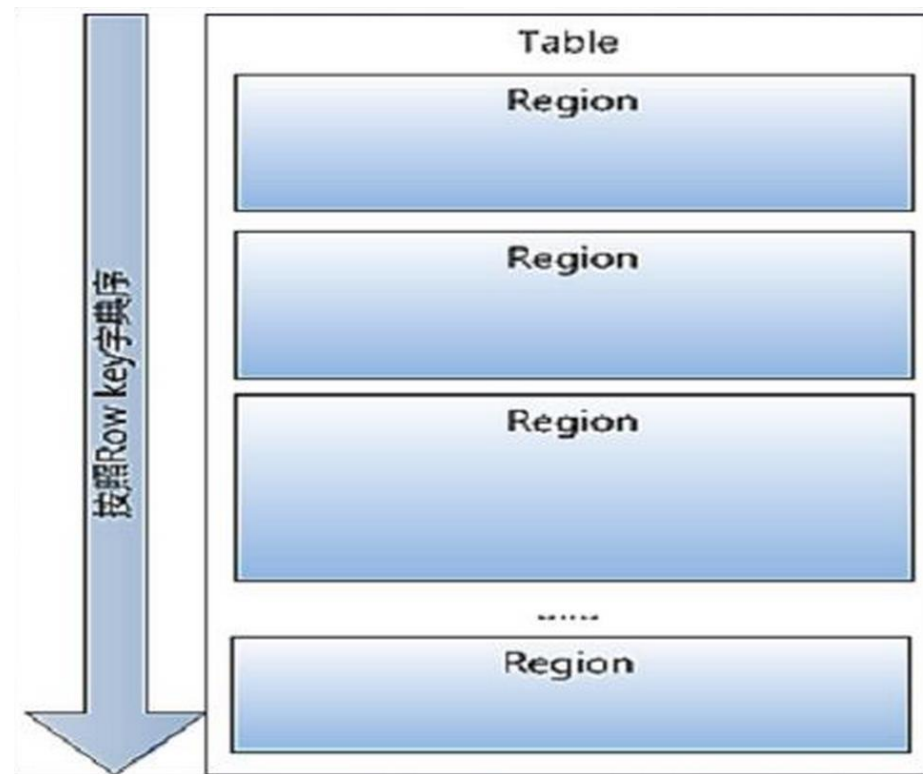
- {rowkey => {family => {qualifier => {version => value}}}}
- a:cf1:bar:1368394583:7
- a:cf1:foo:1368394261:hello

## Hbase 物理模型

- Hbase一张表由一个或多个 Hregion组成
- 记录之间按照Row Key的字典序排列

相当于数据库的水平分表。

Hregion 类似于 HDFS Block的概念, 如果一台Region Server 挂了 Region也会有数据迁移的动作



## Hbase 物理模型

- 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，当增大到一个阈值的时候，Hregion就会等分会两个新的Hregion。当table中的行不断增多，就会有越来越多的Hregion。

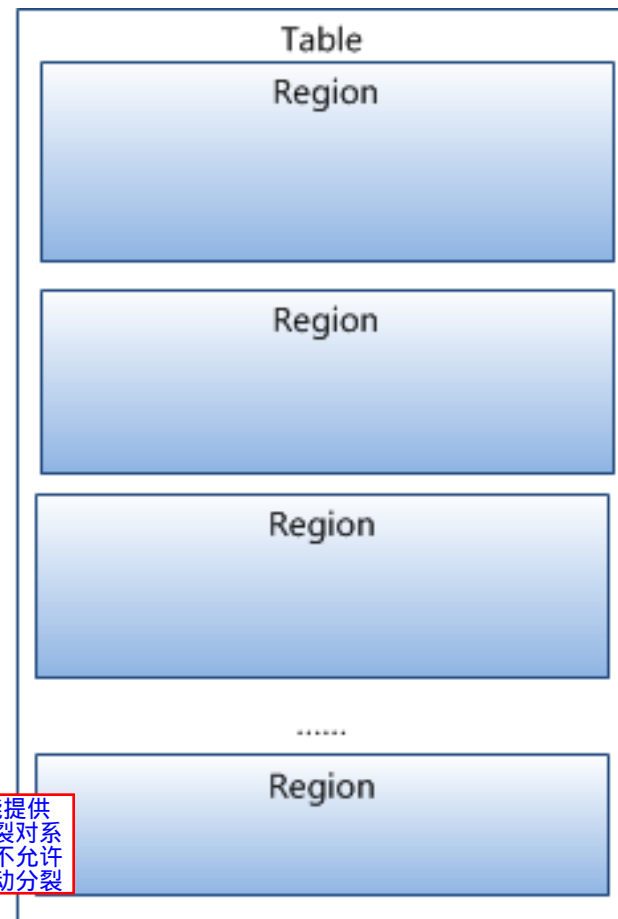
默认10G, 挺大挺牛逼



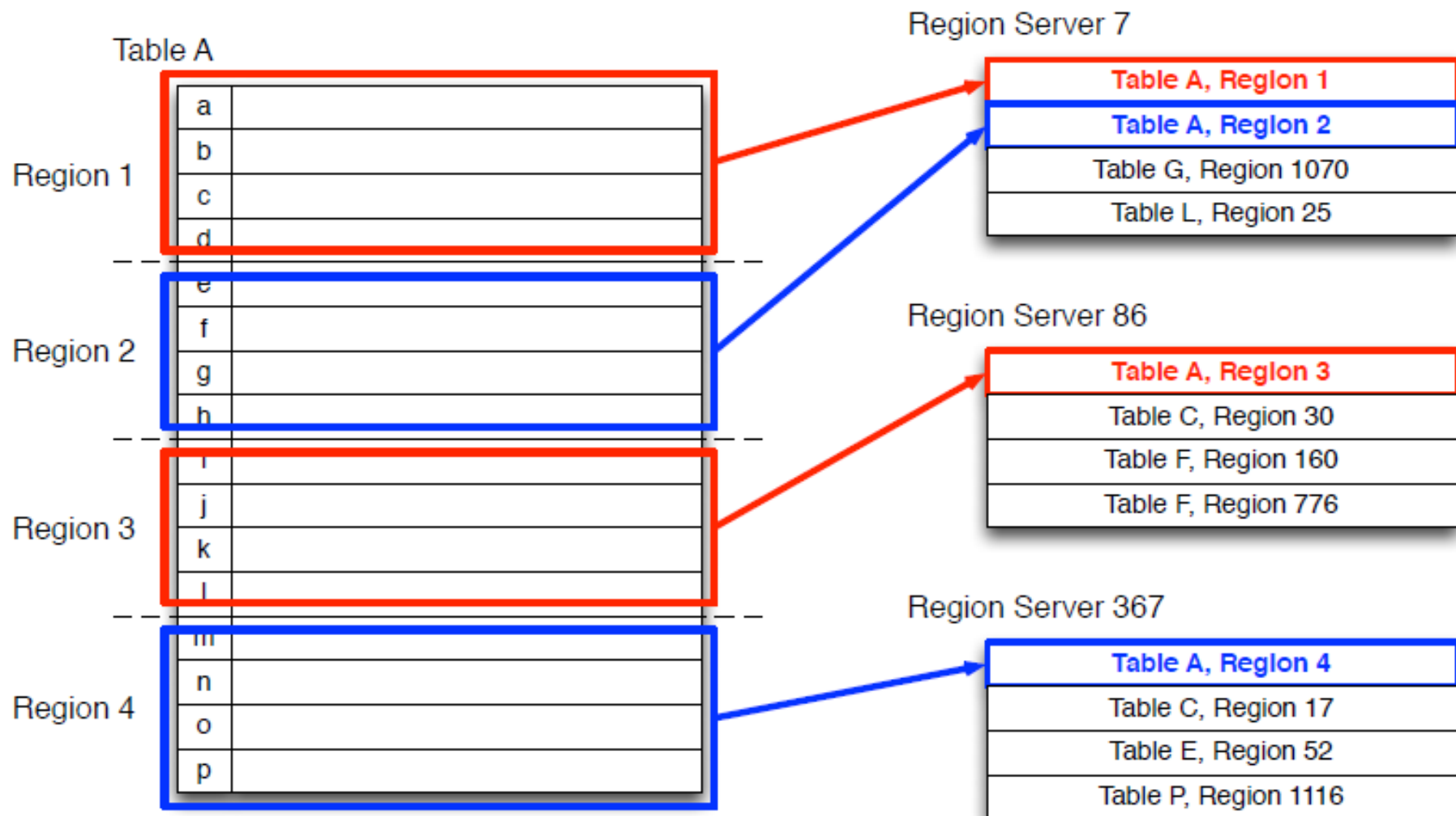
Region分裂过程中是不能对外提供服务的。一个Region 拆成多个Region rowKey的分布信息是会发生改变的 等分裂完成才能对外提供服务。



由于分裂时候不能提供服务, 所以高峰分裂对系统影响挺大, 一般不允许自动分裂, 都去手动分裂



## Hbase 物理模型



## Hbase 物理模型

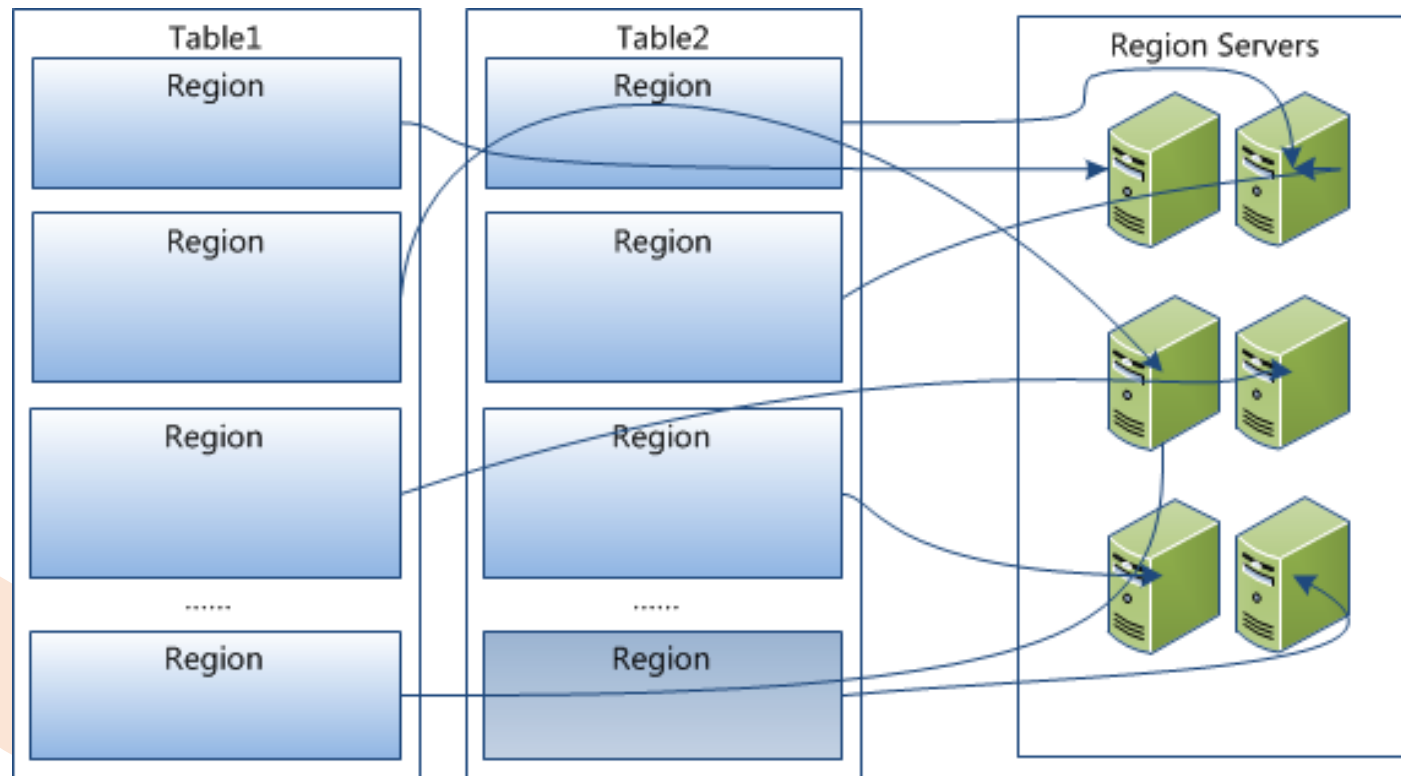
- 表 -> HTable
- 按RowKey范围分的Region-> HRegion -> Region Servers
- HRegion按列族 (Column Family) -> 多个 HStore
- HStore -> memstore + HFiles(均为有序的KV)
- HFiles -> HDFS



HBASE最近本的数据存储单元

## Hbase 物理模型

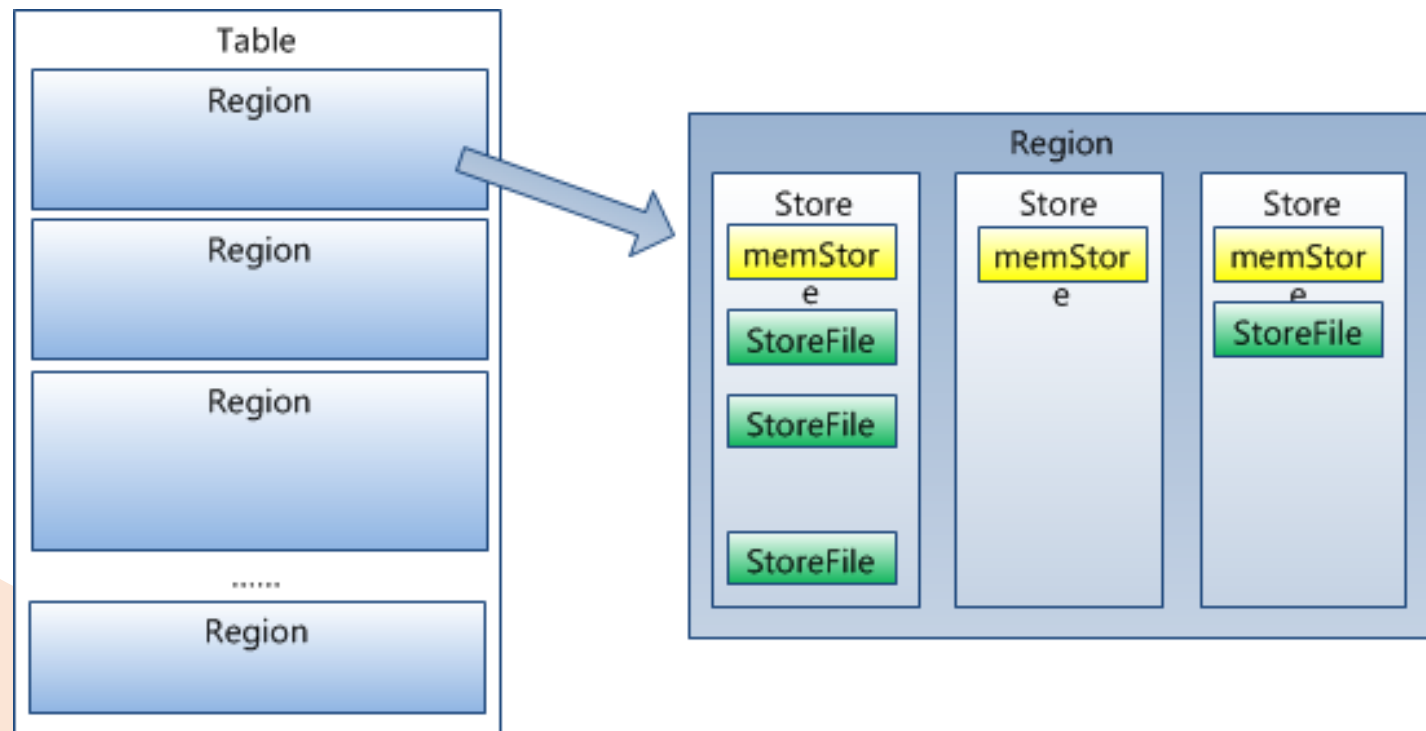
- HRegion是Hbase中分布式存储和负载均衡的最小单元。
- 最小单元就表示不同的Hregion可以分布在不同的HRegion server上。
- 但一个Hregion是不会拆分到多个server上的。



## Hbase 物理模型

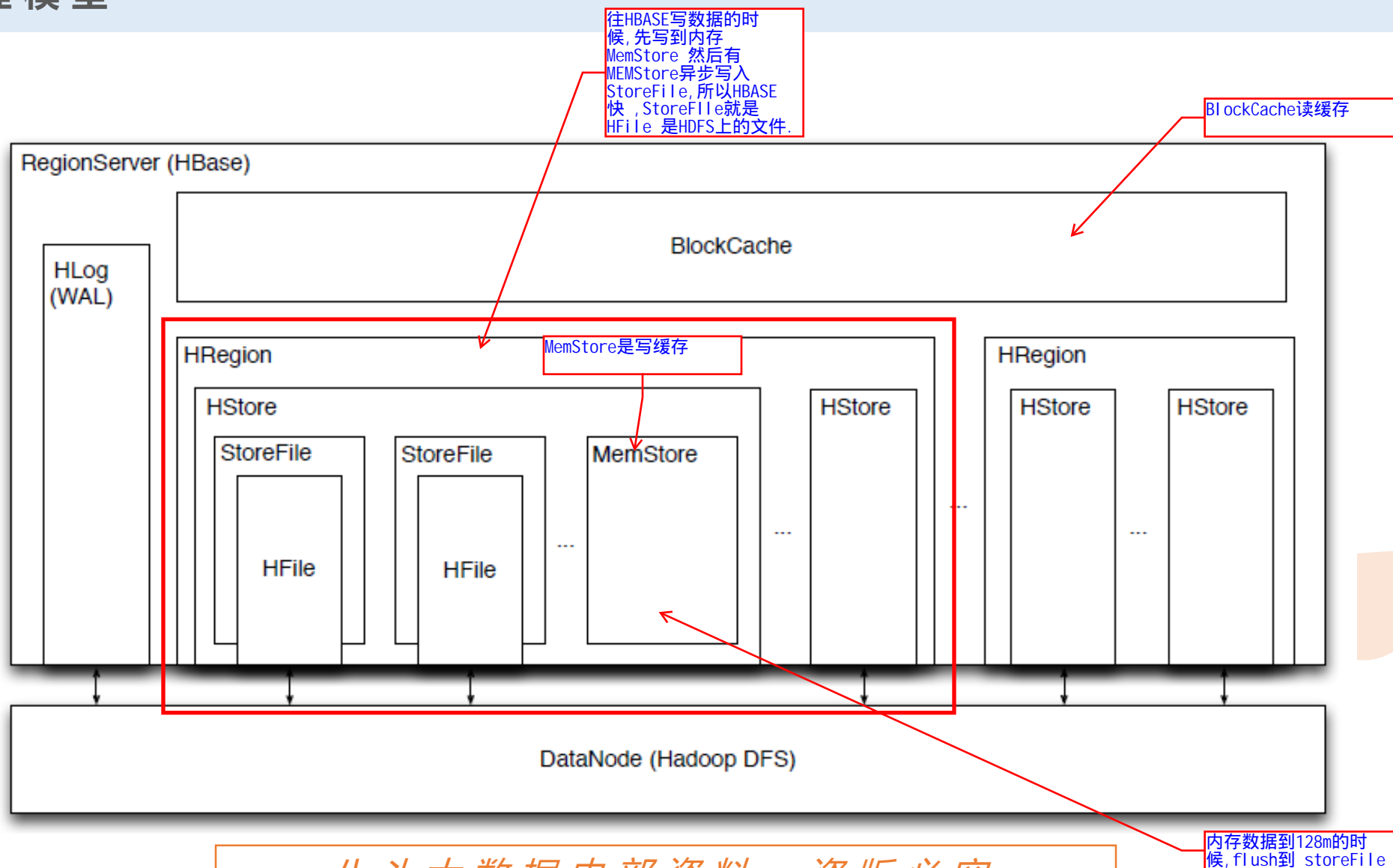
- HRegion虽然是分布式存储的最小单元，但并不是存储的最小单元。

HStore 是最小的存储单元





## Hbase 物理模型



## Hbase 系统架构

- Client

- 访问Hbase的接口，并维护Cache加速Region Server的访问

- Master

- 负载均衡，分配Region到RegionServer

- Region Server

- 维护Region，负责Region的IO请求

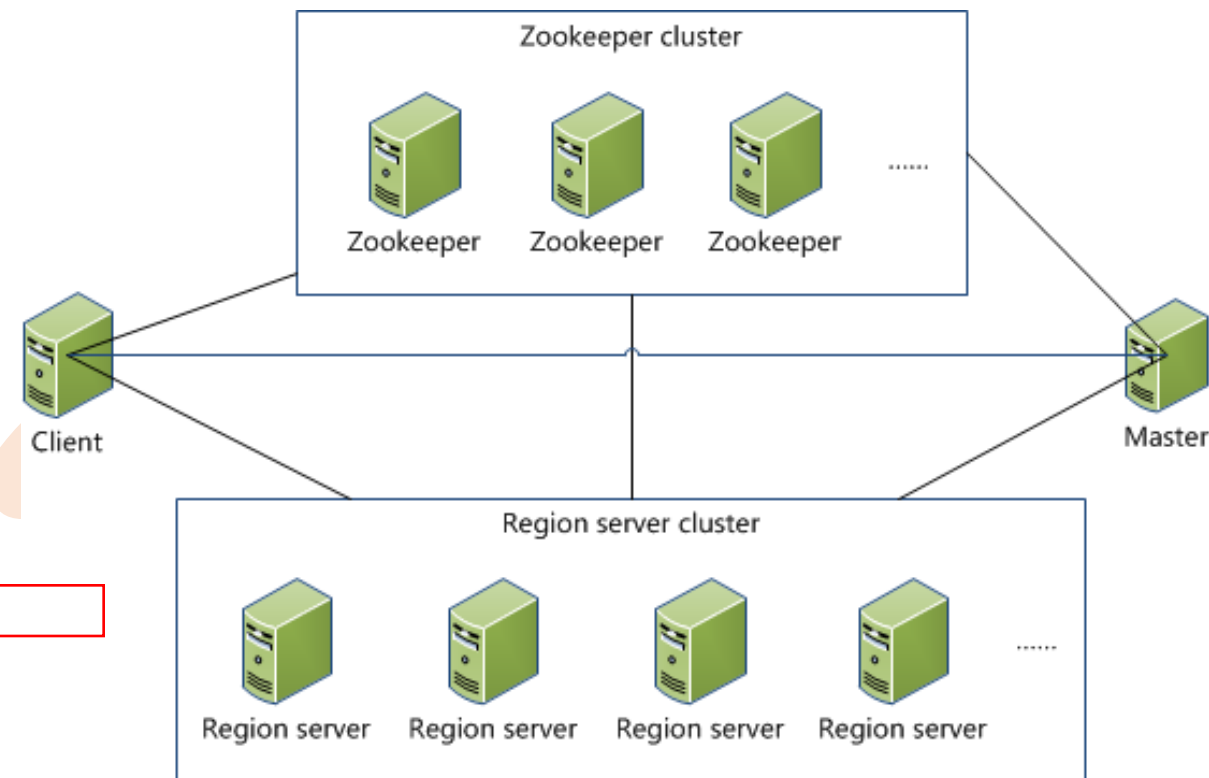
- Zookeeper

- 保证集群中只有一个Master
  - 存储所有Region的入口（ROOT）地址
  - 实时监控Region Server的上下线信息，并通知Master

本地化原则，自身管理的Region对应的HDFS文件存放在当前DataNode上

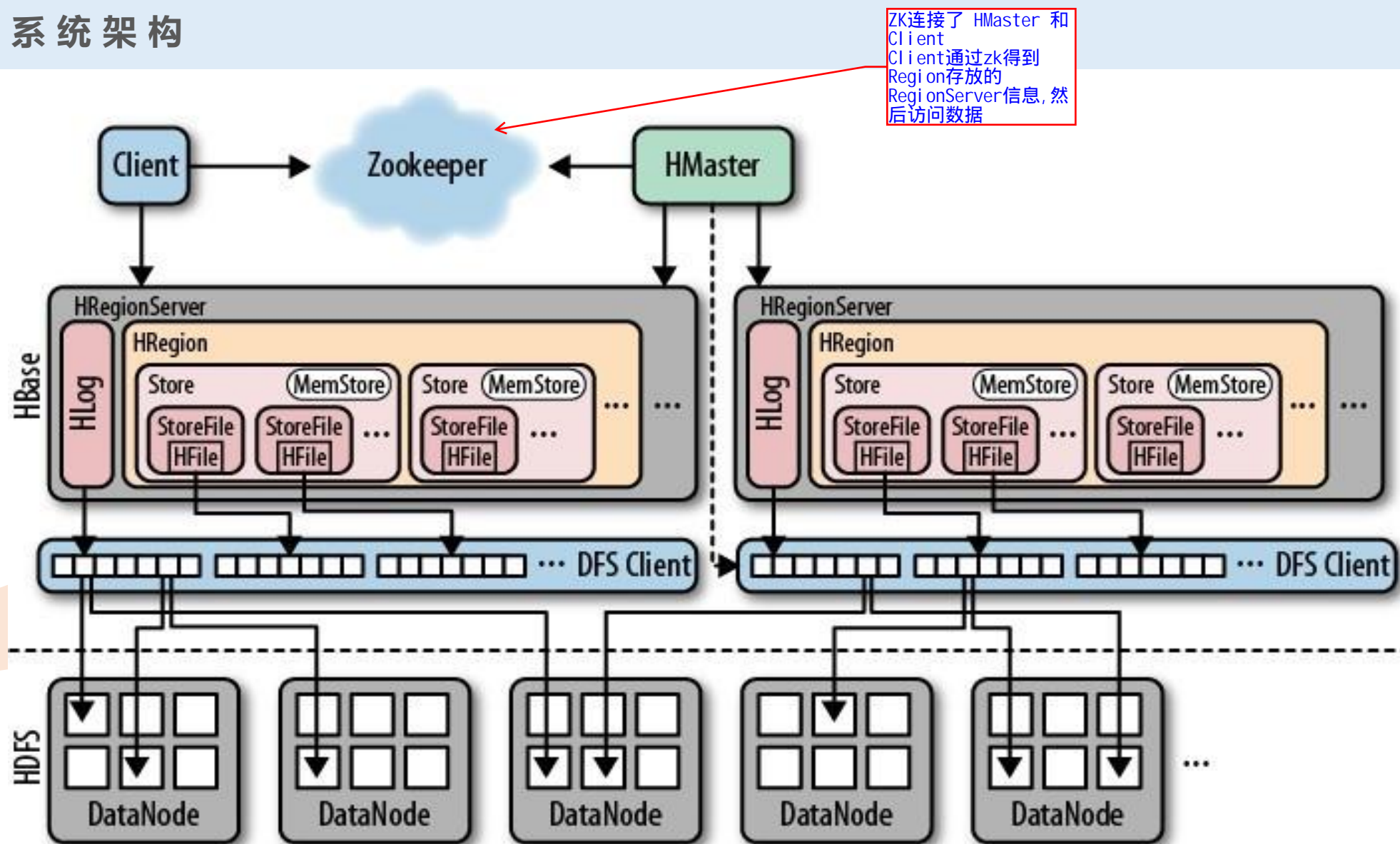
利用ZK的选主机制，多个服务器抢一个锁，抢到就是Master，抢不到监控锁的状态，Master挂了再抢。

RootRegion



所有的RegionServer在Zk /HBase/rs/HostName 创建自己的临时节点。  
HMaster监控这个节点，一旦RegionServer出现故障，zk删掉这个临时节点，通知HMaster

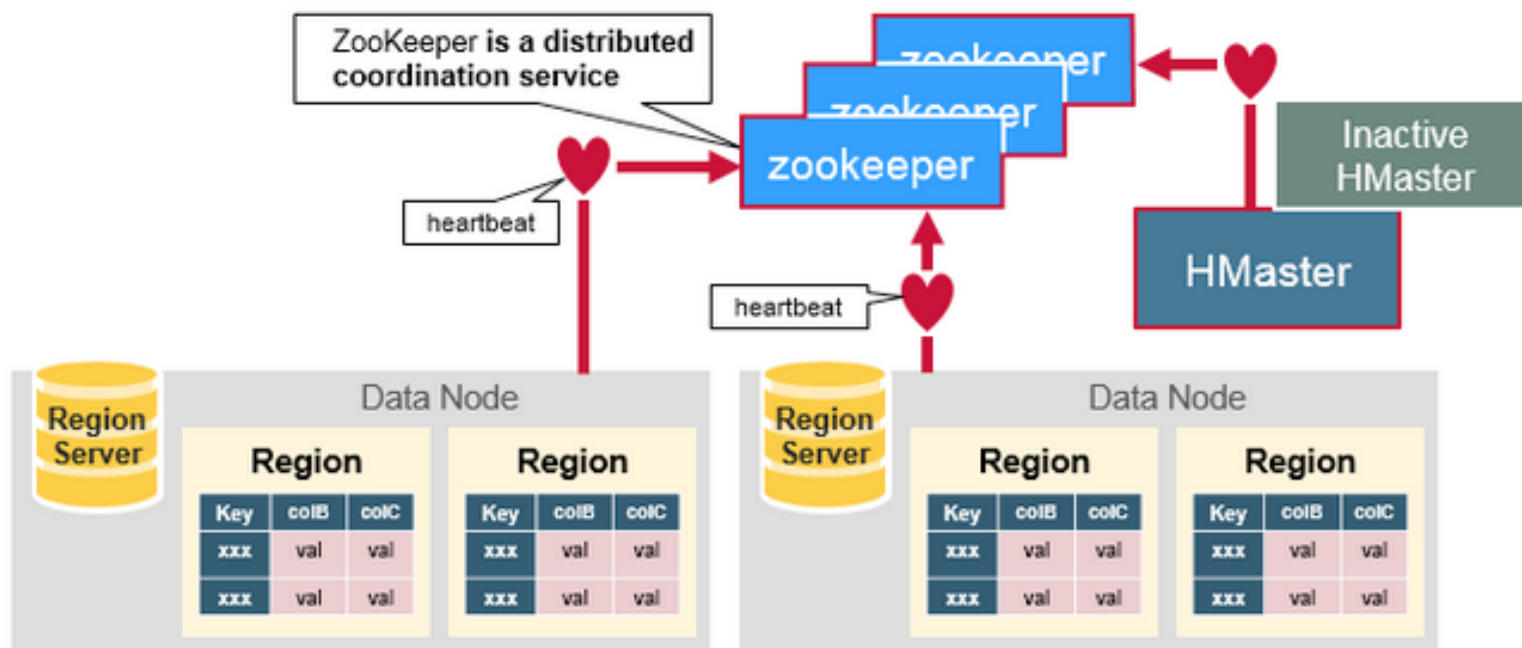
## Hbase 系统架构



## Hbase 的容错

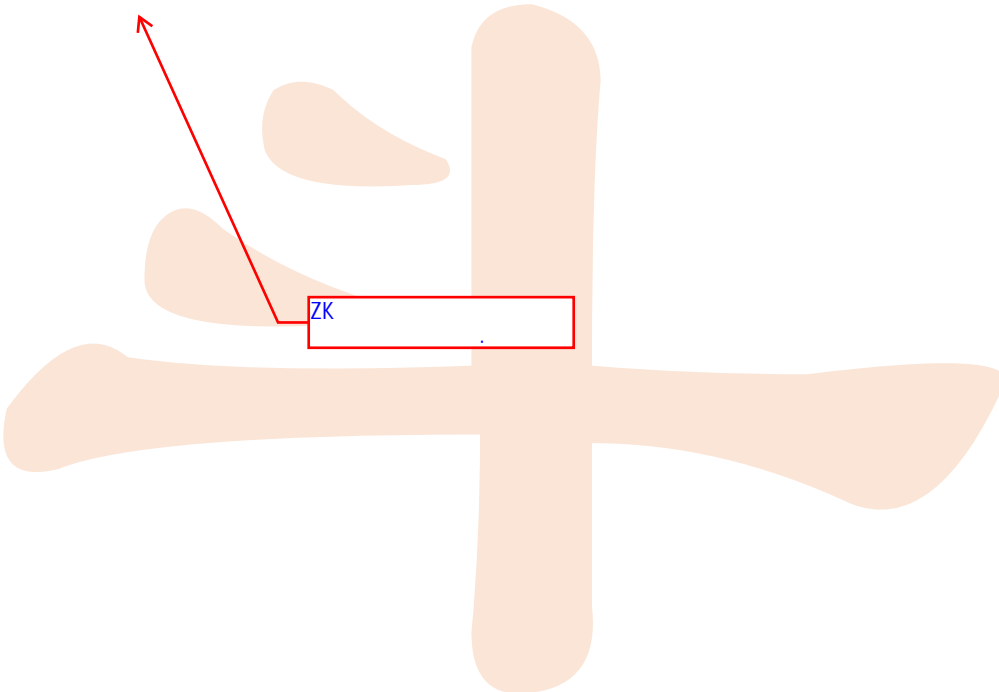
- ZooKeeper协调集群所有节点的共享信息，在HMaster和HRegionServer连接到ZooKeeper后创建Ephemeral节点，并使用Heartbeat机制维持这个节点的存活状态，如果某个Ephemeral节点失效，则HMaster会收到通知，并做相应的处理。

通过临时节点，  
RegionServer掉线zk  
删节点 然后通知  
HMaster



## Hbase 的容错

- 除了HDFS存储信息，HBase还在Zookeeper中存储信息，其中的znode信息：
  - /hbase/root-region-server，Root region的位置
  - /hbase/table/-ROOT-，根元数据信息
  - /hbase/table/.META.，元数据信息
  - /hbase/master，当选的Master
  - /hbase/backup-masters，备选的Master
  - /hbase/rs，Region Server的信息
  - /hbase/unassigned，未分配的Region



ZK管理了一些元数据信息 和 集群信息.

## Hbase 的容错

## • Master容错:

## – Zookeeper重新选择一个新的Master

- 无Master过程中, 数据读取仍照常进行;
- 无master过程中, region切分、负载均衡等无法进行

因为 Region对应的元数据信息 以及 rowKey对应的Region信息 在ZK上存储, 所以Master挂了 Client照样可以获取到这些信息, 访问 RegionServer

## • Region Server容错:

- 定时向Zookeeper汇报心跳, 如果一旦时间内未出现心跳, Master将该RegionServer上的Region重新分配到其他RegionServer上, 失效服务器上“预写”日志由主服务器进行分割并派送给新的RegionServer

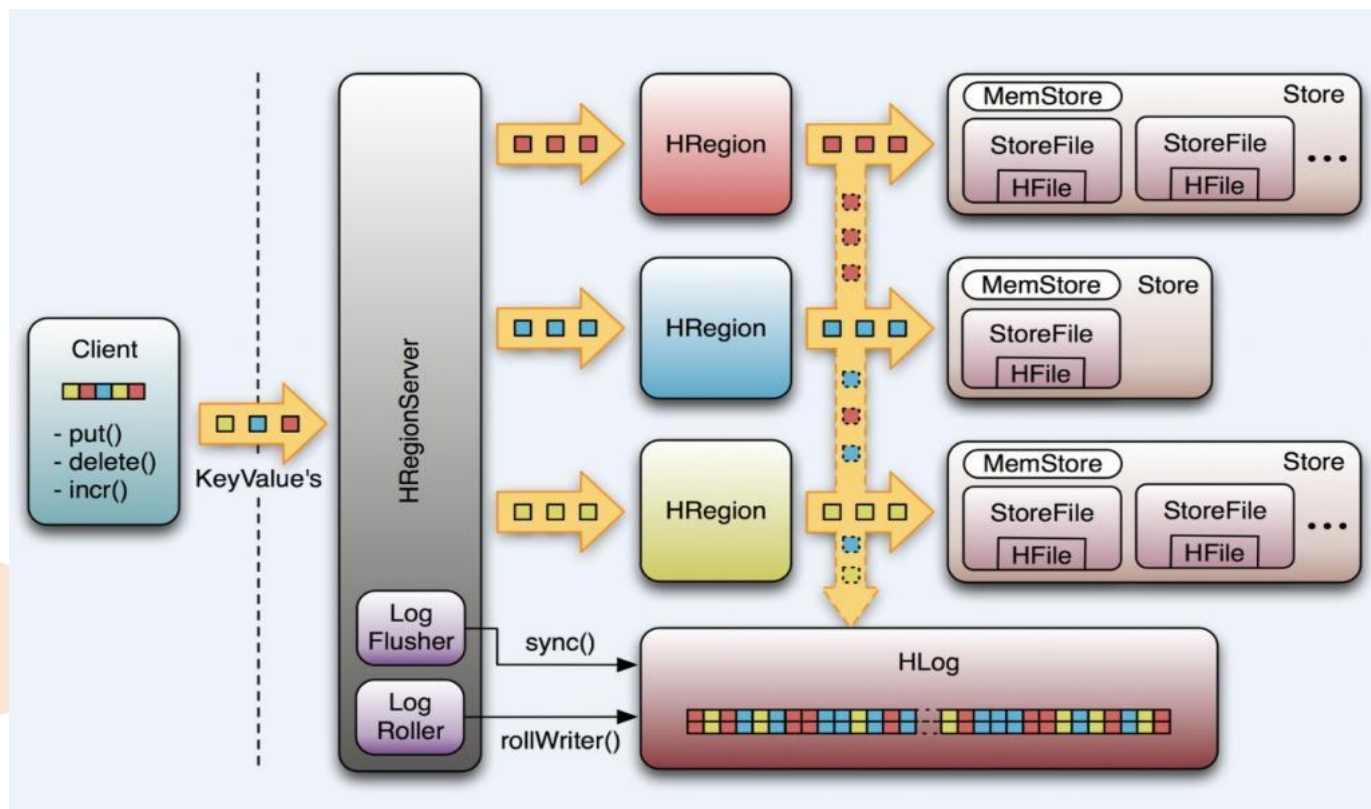
当某一个RegionServer挂了, 而客户端写入的数据还没有来的及写入到HFile的时候 需要从HLog中恢复这部分数据, 步骤如下:  
1: HMaster遍历该RegionServer的HLog, 并做切片处理SplitLog. HMaster会在zk上面创建一个节点, 把哪个RegionServer需要处理哪个Region以列表的形式放在该节点上.  
2: RegionServer 自己过来领取任务, 并把处理结果写入到该节点. HMaster监控该节点的数据变更, 进行接下来的任务分配工作. zk在中间起到了服务协调的作用.

## • Zookeeper容错:

- Zookeeper是一个可靠地服务, 一般配置3或5个Zookeeper实例

## Hbase 的容错

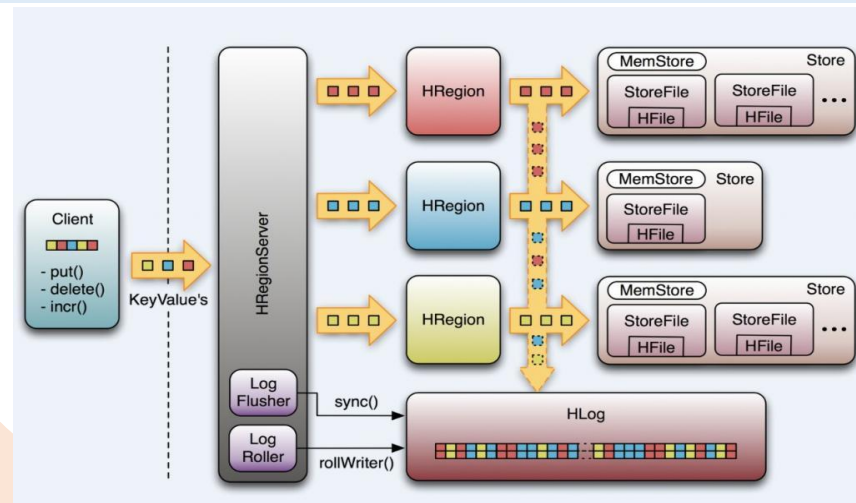
- WAL(Write-Ahead-Log)预写日志
- 是Hbase的RegionServer在处理数据插入和删除的过程中用来记录操作内容的一种日志
- 在每次Put、Delete等一条记录时，首先将其数据写入到  
🗨 RegionServer对应的HLog文件的过程





## Hbase 的容错

- 客户端往RegionServer端提交数据的时候，会写WAL日志，只有当WAL日志写成功以后，客户端才会被告诉提交数据成功，如果写WAL失败会告知客户端提交失败
- 数据落地的过程
- 在一个RegionServer上的所有的Region都共享一个HLog，一次数据的提交是先写WAL，写入成功后，再写memstore。当memstore值到达一定阈值，就会形成一个StoreFile（理解为HFile格式的封装，本质上还是以HFile的形式存储的）





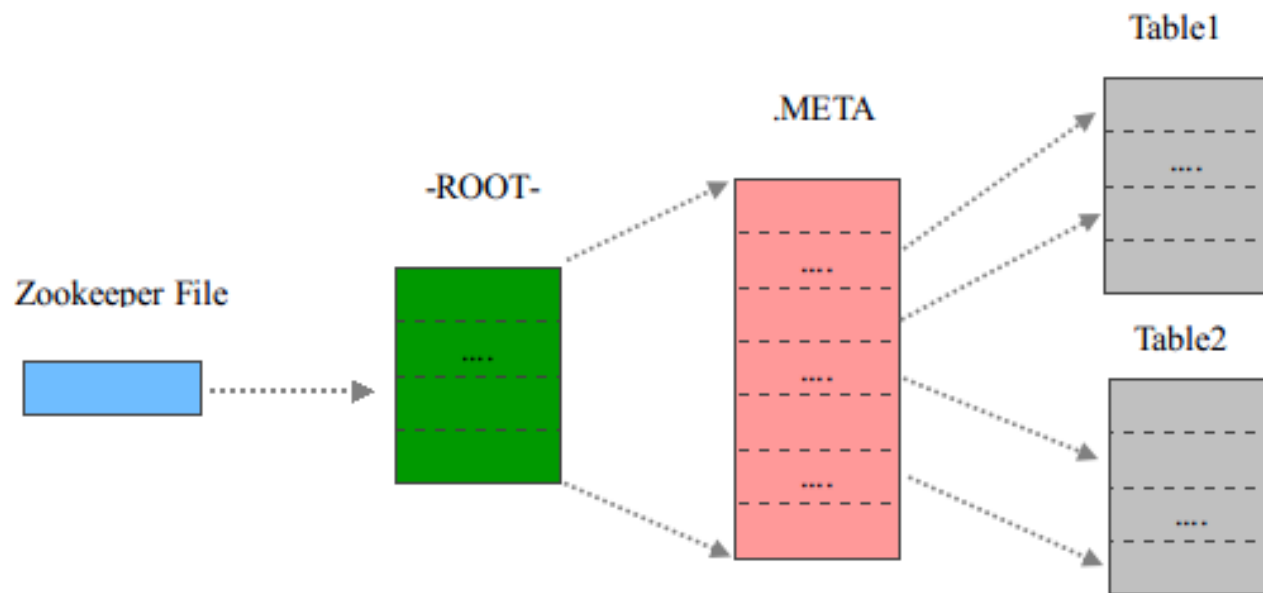
## Hbase 的操作

删除数据的时候 先用标志位 逻辑删除  
region合并的时候 再压缩删除数据 做物理删除。

- 基本的单行操作：PUT, GET, DELETE
- 扫描一段范围的Rowkey: SCAN
  - 由于Rowkey有序而让Scan变得有效
- GET和SCAN支持各种Filter, 将逻辑推给Region Server
  - 以此为基础可以实现复杂的查询
- 支持一些原子操作：INCREMENT、APPEND、CheckAnd{Put,Delete}
- MapReduce
- 注：在单行上可以加锁，具备**强一致性**。这能满足很多应用的需求。

## Hbase 的特殊表

- -ROOT- 表和.META.表是两个比较特殊的表
- .META.记录了用户表的Region信息，.META.可以有多个region。
- -ROOT-记录了.META.表的Region信息，-ROOT-只有一个region，Zookeeper中记录了-ROOT-表的location



## Hbase 的特殊表

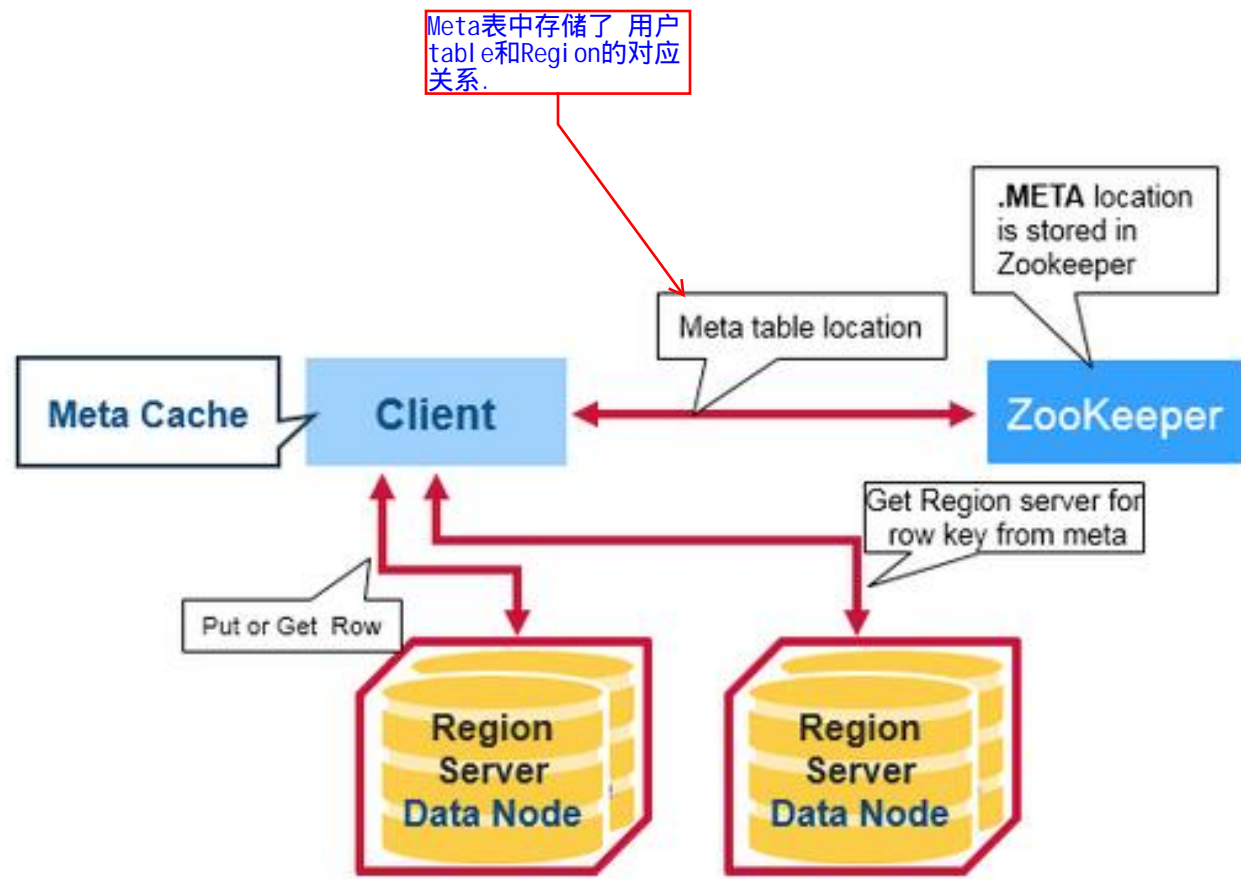
- Hbase 0.96之后去掉了-ROOT- 表，因为：
  - 三次请求才能直到用户Table真正所在的位置也是性能低下的
  - 即使去掉-ROOT- Table，也还可以支持 $2^{17}(131072)$ 个Hregion，对于集群来说，存储空间也足够
- 所以目前流程为：

Client-->zk(获取的HRegionServer列表)-->确定请求数据对应的RegionServer-->找到Region-->读取Row

  - 从ZooKeeper(/hbase/meta-region-server)中获取hbase:meta的位置（HRegionServer的位置），缓存该位置信息
  - 从HRegionServer中查询用户Table对应请求的RowKey所在的HRegionServer，缓存该位置信息
  - 从查询到HRegionServer中读取Row。

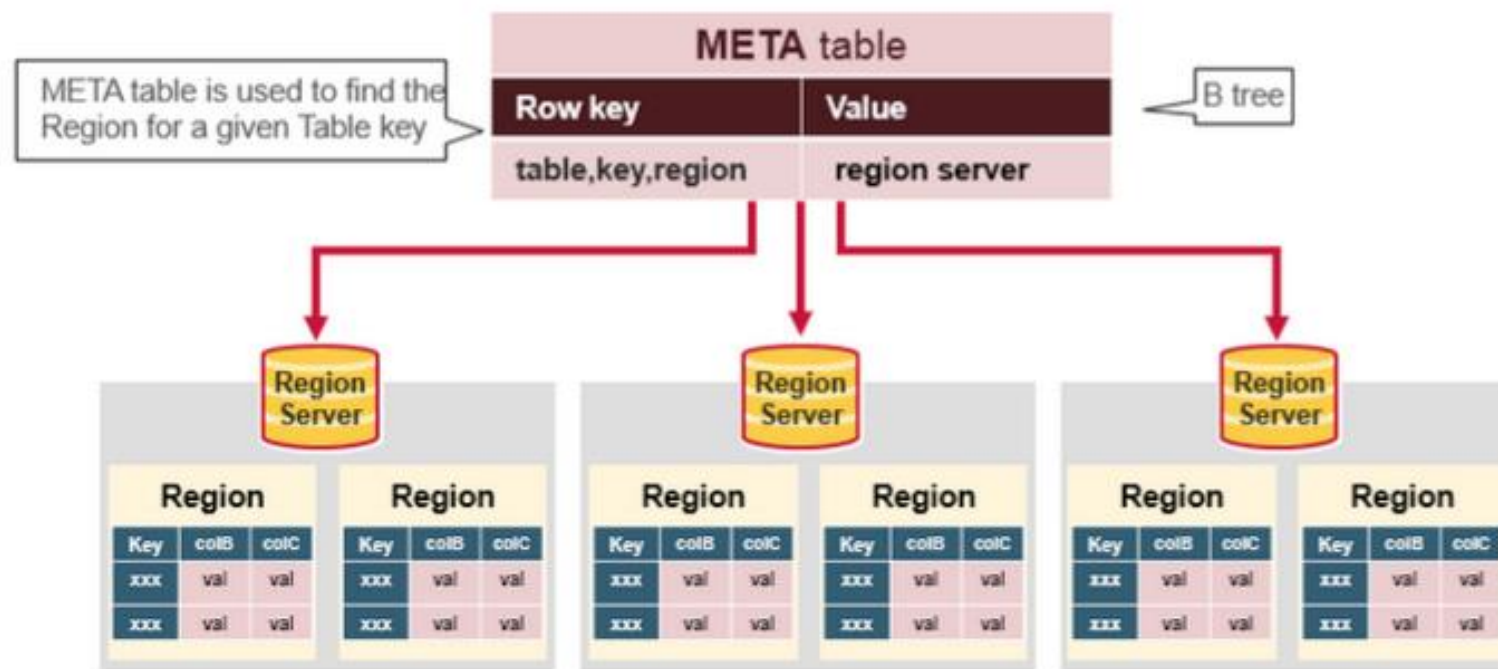
## Hbase 的写入流程——寻址

- 从这个过程中，我们发现客户会缓存这些位置信息，然而第二步它只是缓存当前RowKey对应的HRegion的位置，因而如果下一个要查的RowKey不在同一个HRegion中，则需要继续查询hbase:meta所在的HRegion，然而随着时间的推移，客户端缓存的位置信息越来越多，以至于不需要再次查找hbase:meta Table的信息，除非某个HRegion因为宕机或Split被移动，此时需要重新查询并且更新缓存



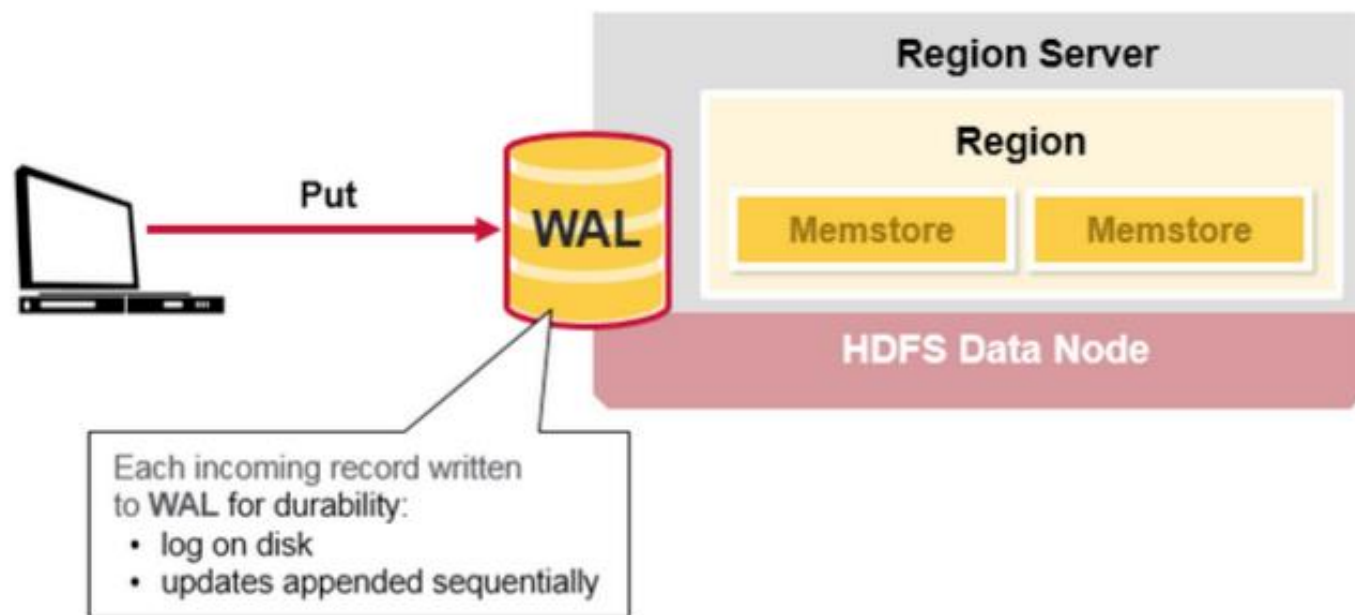
## Hbase 的写入流程——寻址

- hbase:meta表存储了所有用户HRegion的位置信息



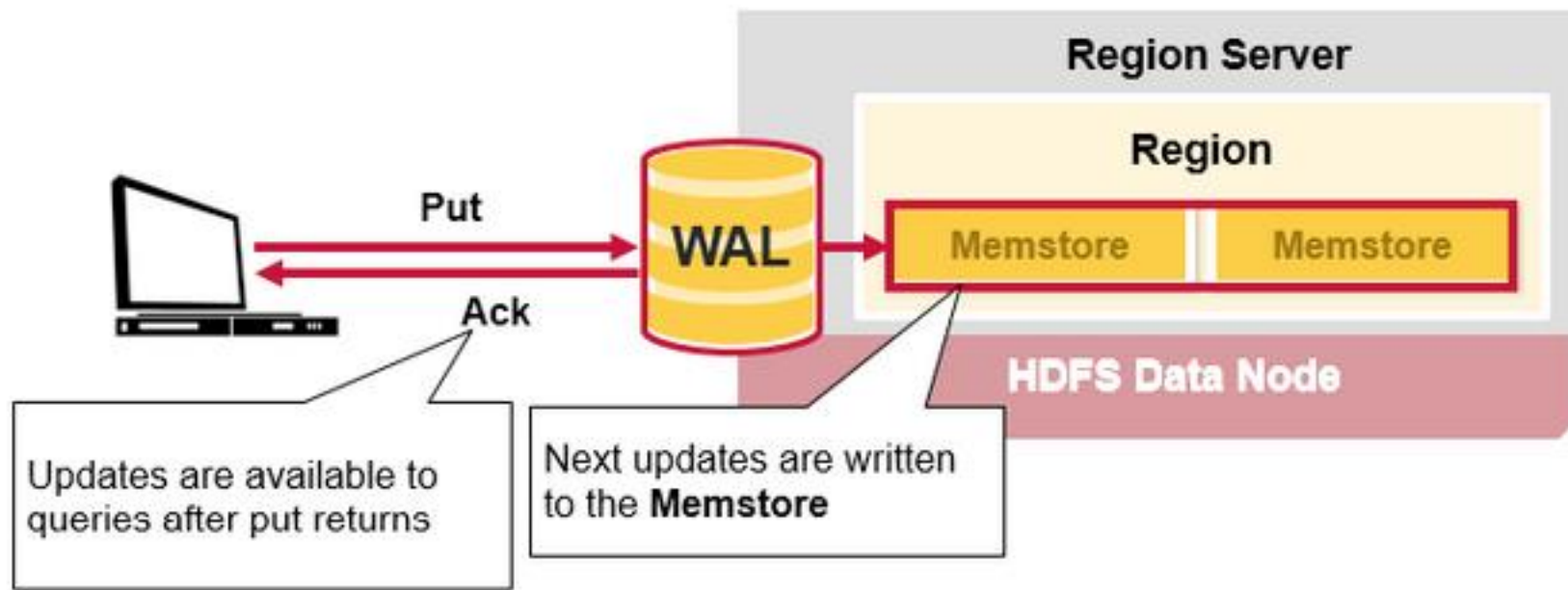
## Hbase 的写入流程——写入

- 当客户端发起一个Put请求时，首先它从hbase:meta表中查出该Put数据最终需要去的HRegionServer。然后客户端将Put请求发送给相应的HRegionServer，在HRegionServer中它首先会将该Put操作写入WAL日志(Flush到磁盘中)。
- Memstore是一个写缓存，每一个Column Family有一个自己的MemStore



## Hbase 的写入流程——写入

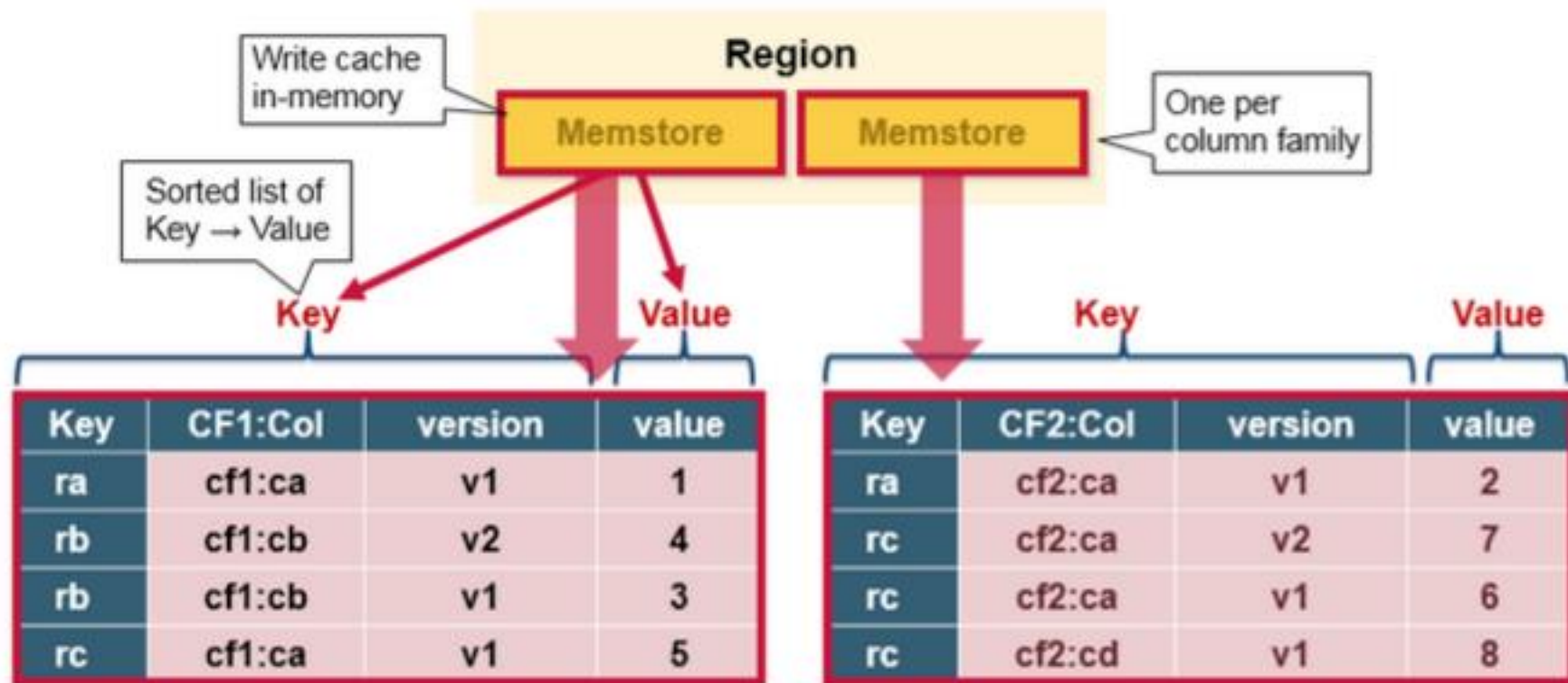
- 写完WAL日志文件后，HRegionServer根据Put中的TableName和RowKey找到对应的HRegion，并根据Column Family找到对应的HStore，并将Put写入到该HStore的MemStore中。此时写成功，并返回通知客户端。





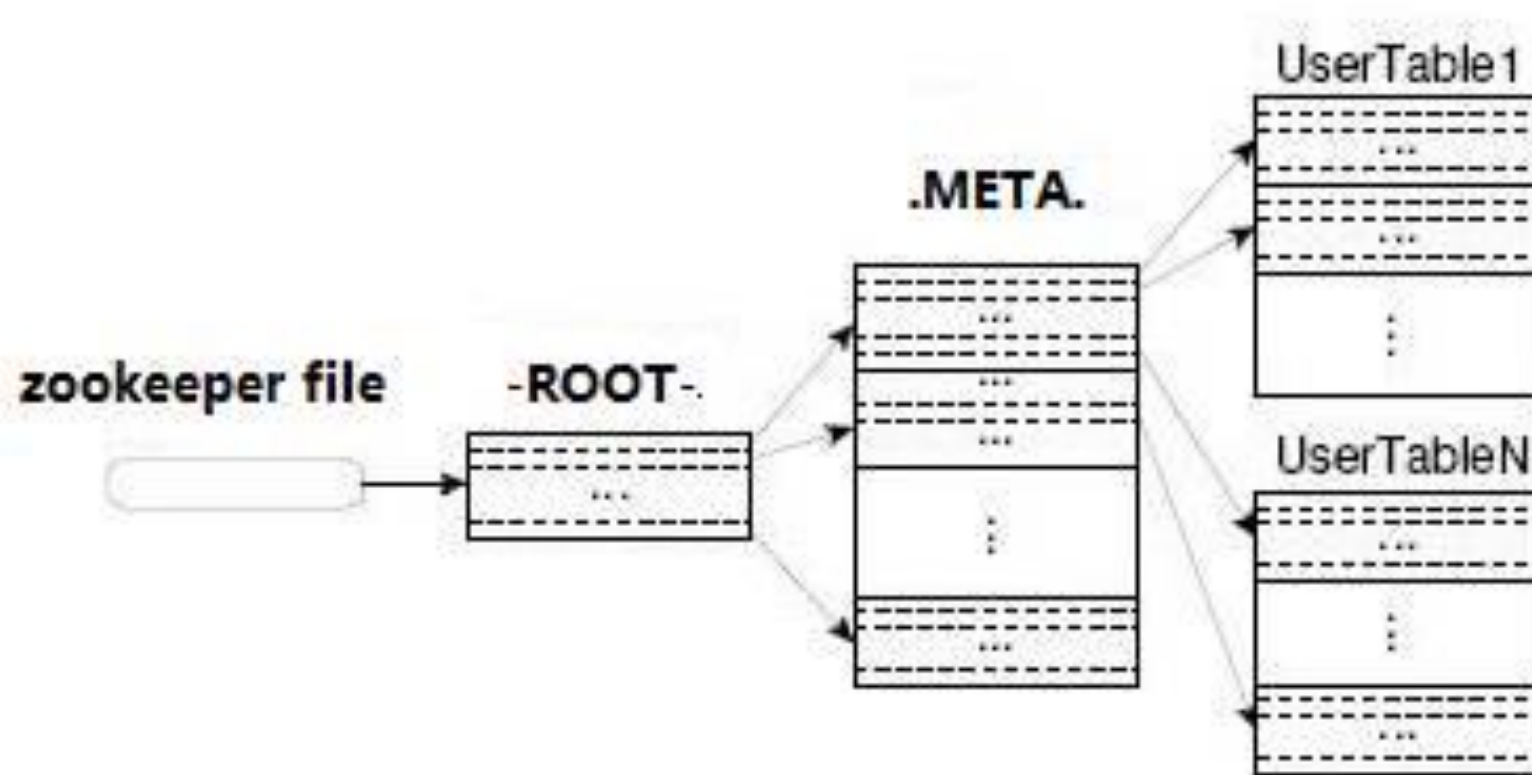
## Hbase 的写入流程——写入

- MemStore是一个In Memory Sorted Buffer，在每个HStore中都有一个MemStore，即它是一个HRegion的一个Column Family对应一个实例。它的排列顺序以RowKey、Column Family、Column的顺序以及Timestamp的倒序。

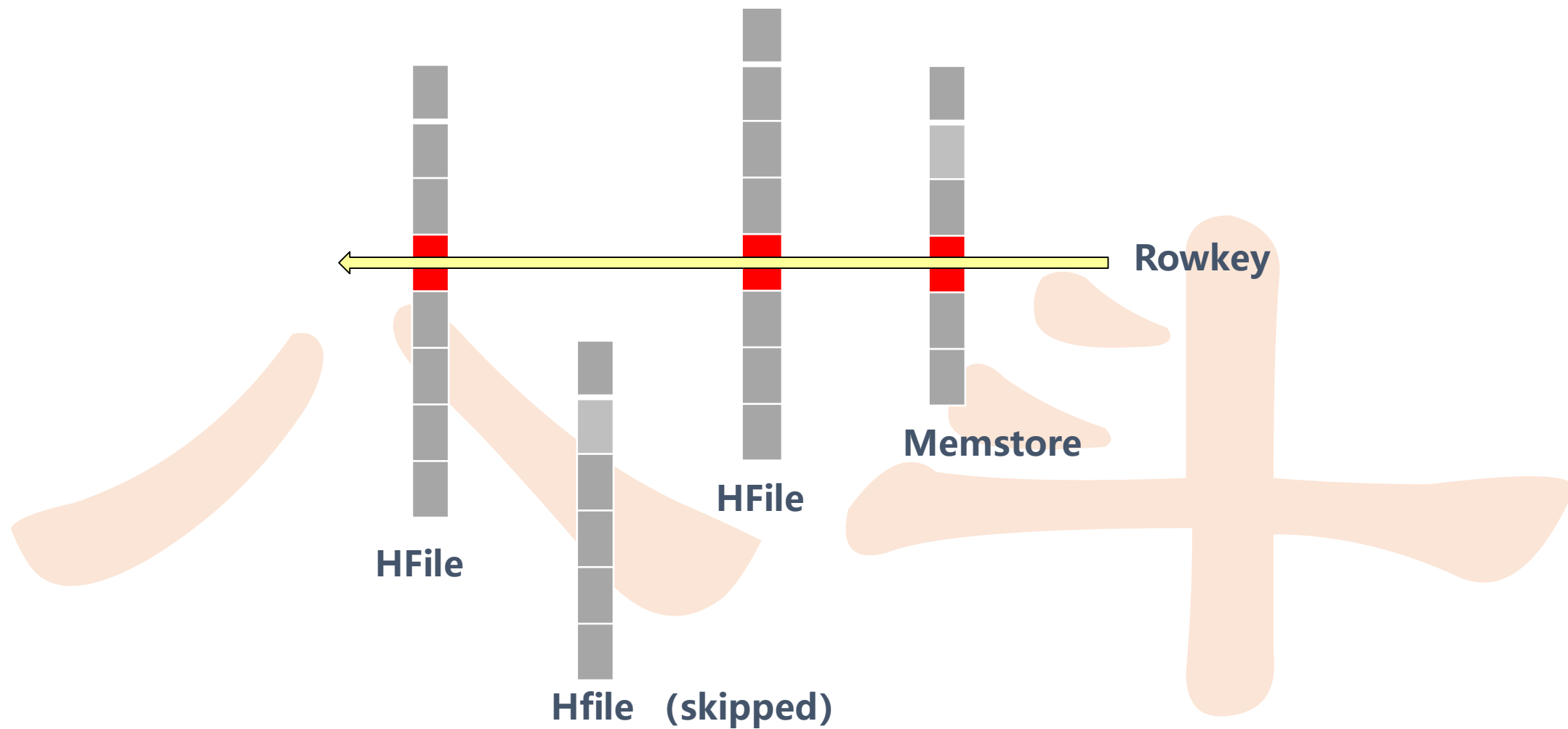




## Hbase 的读取流程

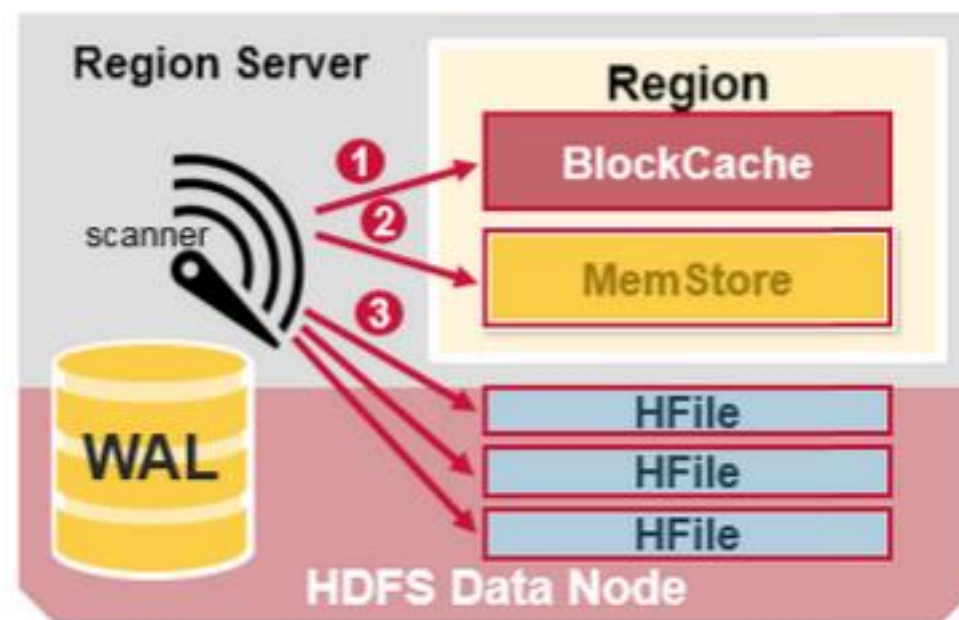
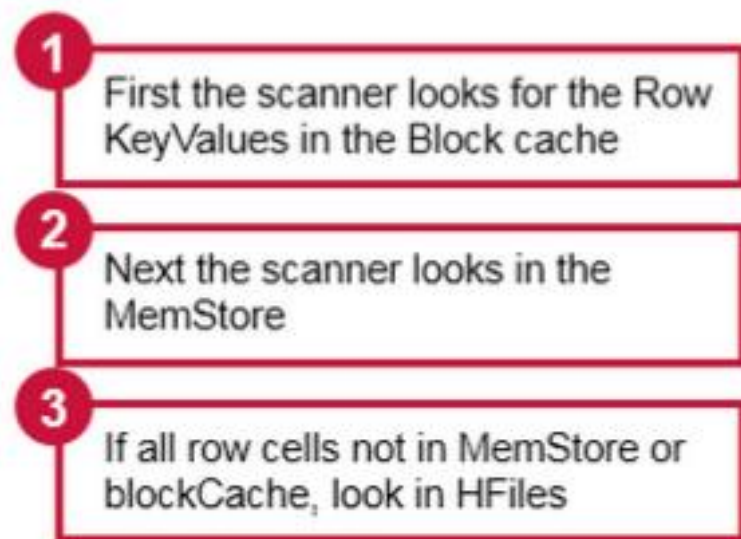


## Hbase 的读取流程



## Hbase 的读取流程

- HBase中扫描的顺序依次是：BlockCache、MemStore、StoreFile(HFile)



## Hbase 的 Compaction 和 Split

- 问题：随着写入不断增多，flush次数不断增多，Hfile文件越来越多,所以Hbase需要对这些文件进行合并
- Compaction会从一个region的一个store中选择一些hfile文件进行合并。合并说来原理很简单，先从这些待合并的数据文件中读出KeyValues，再按照由小到大排列后写入一个新的文件中。之后，这个新生成的文件就会取代之前待合并的所有文件对外提供服务
- **Minor Compaction**：是指选取一些小的、相邻的StoreFile将他们合并成一个更大的StoreFile，在这个过程中不会处理已经Deleted或Expired的Cell。一次Minor Compaction的结果是更少并且更大的StoreFile
- **Major Compaction**：是指将所有的StoreFile合并成一个StoreFile，这个过程还会清理三类无意义数据：被删除的数据、TTL过期数据、版本号超过设定版本号的数据
- Major Compaction时间会持续比较长，整个过程会消耗大量系统资源，对上层业务有比较大的影响。因此线上业务都会将关闭自动触发Major Compaction功能，改为手动在业务低峰期触发

## Hbase 的 Compaction 和 Split

- Compaction本质：使用短时间的IO消耗以及带宽消耗换取后续查询的低延迟
- compact的速度远远跟不上HFile生成的速度，这样就会使HFile的数量会越来越多，导致读性能急剧下降。为了避免这种情况，在HFile的数量过多的时候会限制写请求的速度
- Split
  - 当一个Region太大时，将其分裂成两个Region
- Split和Major Compaction可以手动或者自动做。

## Outline

HBase基础

【实践】HBase搭建

【实践】Hbase Shell

【实践】Hbase的Python操作

## H b a s e 安 装

- 工具包: hbase-0.98.24-hadoop1-bin.tar.gz
- 分别配置:
  - bashrc: 环境变量
  - regionservers: 节点host
  - hbase-env.sh: 环境变量
  - hbase-site.xml:
    - 指定hadoop目录, 指定zookeeper

```
23 <configuration>
24   <property>
25     <name>hbase.tmp.dir</name>
26     <value>/var/hbase</value>
27   </property>
28   <property>
29     <name>hbase.rootdir</name>
30     <value>hdfs://master:9000/hbase</value>
31   </property>
32   <property>
33     <name>hbase.cluster.distributed</name>
34     <value>true</value>
35   </property>
36   <property>
37     <name>hbase.zookeeper.quorum</name>
38     <value>master,slave1,slave2</value>
39   </property>
40   <property>
41     <name>hbase.zookeeper.property.dataDir</name>
42     <value>/usr/local/src/hbase-0.98.24-hadoop1/zookeeper</value>
43   </property>
44 </configuration>
```

## Hbase 安装

- 启动集群: ]# start-hbase.sh

```
[root@master src]# start-hbase.sh
slave1: starting zookeeper, logging to /usr/local/src/hbase-0.98.24-hadoop1/logs/hbase-root-zookeeper-slave1.out
slave2: starting zookeeper, logging to /usr/local/src/hbase-0.98.24-hadoop1/logs/hbase-root-zookeeper-slave2.out
master: starting zookeeper, logging to /usr/local/src/hbase-0.98.24-hadoop1/logs/hbase-root-zookeeper-master.out
starting master, logging to /usr/local/src/hbase-0.98.24-hadoop1/logs/hbase-badou-master-master.out
slave1: starting regionserver, logging to /usr/local/src/hbase-0.98.24-hadoop1/logs/hbase-root-regionserver-slave1.out
slave2: starting regionserver, logging to /usr/local/src/hbase-0.98.24-hadoop1/logs/hbase-root-regionserver-slave2.out
master: starting regionserver, logging to /usr/local/src/hbase-0.98.24-hadoop1/logs/hbase-root-regionserver-master.out
[root@master src]# jps
67979 HMaster
55877 JobTracker
68224 Jps
55639 NameNode
68122 HRegionServer
55791 SecondaryNameNode
67869 HQuorumPeer
[root@master src]# start-hbase.sh
```



Outline

HBase基础

【实践】HBase搭建

【实践】Hbase Shell

【实践】Hbase的Python操作

## Hbase Shell

- 执行hbase shell命令

```
[root@master src]# hbase shell
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.24-hadoop1, r9c13a1c3d8cf999014f30104d1aa9d79e74ca3d6, Thu Dec 22 02:28:55 UTC 2016

hbase(main):001:0> help
```

- 查看数据库状态 (status)

```
hbase(main):004:0> status
1 active master, 0 backup masters, 3 servers, 0 dead, 0.6667 average load
```

- 表示有3台机器活着，0台机器down掉，当前负载0.67（数字越大，负载越大）

## Hbase Shell

- 执行help查询帮助
  - General: 普通命令组
  - Ddl: 数据定义语言命令组
  - Dml: 数据操作语言命令组
  - Tools: 工具组
  - Replication: 复制命令组
  - SHELL USAGE: shell语法

```
hbase(main):005:0> help
HBase Shell, version 0.98.24-hadoop1, r9c13a1c3d8cf999014f30104d1aa9d79e74ca3d6, Thu Dec 22 02:28:55 UTC 2016
Type 'help "COMMAND"', (e.g. 'help "get"' -- the quotes are necessary) for help on a specific command.
Commands are grouped. Type 'help "COMMAND_GROUP"', (e.g. 'help "general"') for help on a command group.

COMMAND GROUPS:
  Group name: general
  Commands: processlist, status, table_help, version, whoami

  Group name: ddl
  Commands: alter, alter_async, alter_status, create, describe, disable, disable_all, drop, drop_all, enable,
ist, show_filters

  Group name: namespace
  Commands: alter_namespace, create_namespace, describe_namespace, drop_namespace, list_namespace, list_names

  Group name: dml
  Commands: append, count, delete, deleteall, get, get_counter, get_splits, incr, put, scan, truncate, truncat

  Group name: tools
  Commands: assign, balance_switch, balancer, balancer_enabled, catalogjanitor_enabled, catalogjanitor_run, c
lush, hlog_roll, major_compact, merge_region, move, split, trace, unassign, zk_dump

  Group name: replication
  Commands: add_peer, disable_peer, disable_table_replication, enable_peer, enable_table_replication, get_pee
ables, remove_peer, set_peer_tableCFs, show_peer_tableCFs, update_peer_config

  Group name: snapshots
  Commands: clone_snapshot, delete_all_snapshot, delete_snapshot, delete_table_snapshots, list_snapshots, lis

  Group name: security
  Commands: grant, list_security_capabilities, revoke, user_permission

  Group name: visibility labels
  Commands: add_labels, clear_auths, get_auths, list_labels, set_auths, set_visibility

SHELL USAGE:
Quote all names in HBase Shell such as table and column names. Commas delimit
command parameters. Type <RETURN> after entering a command to run it.
Dictionaries of configuration used in the creation and alteration of tables are
Ruby Hashes. They look like this:

  {'key1' => 'value1', 'key2' => 'value2', ...}

and are opened and closed with curly-braces. Key/values are delimited by the
'=>' character combination. Usually keys are predefined constants such as
NAME, VERSIONS, COMPRESSION, etc. Constants do not need to be quoted. Type
'Object.constants' to see a (messy) list of all constants in the environment.

If you are using binary keys or values and need to enter them in the shell, use
double-quoted hexadecimal representation. For example:

hbase> get 't1', "key\x03\x3f\xcd"
hbase> get 't1', "key\003\023\011"
hbase> put 't1', "test\xef\xff", 'f1:', "\x01\x33\x40"

The HBase shell is the (J)Ruby IRB with the above HBase-specific commands added.
For more on the HBase Shell, see http://hbase.apache.org/book.html
hbase(main):006:0>
```

## Hbase Shell

- 命令 create / list / describe

- hbase(main):006:0> create 'music\_table','meta\_data','action'

- 表名: music\_table

- 列簇1: meta\_data

- 列簇2: 'action'

```
hbase(main):006:0> create 'music_table','meta_data','action'
0 row(s) in 0.9170 seconds

=> Hbase::Table - music_table
hbase(main):007:0> list
TABLE
music_table
1 row(s) in 0.1400 seconds

=> ["music_table"]
hbase(main):008:0> desc
desc          describe          describe_namespace
hbase(main):008:0> describe
describe      describe_namespace
hbase(main):008:0> describe 'music_table'
Table music_table is ENABLED
music_table
COLUMN FAMILIES DESCRIPTION
{NAME => 'action', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLI
FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => '
{NAME => 'meta_data', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', RE
> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY =
2 row(s) in 0.1460 seconds

hbase(main):009:0>
```

## Hbase Shell

- 命令alter / disable / enable

- 凡是要修改表的结构hbase规定，必须先禁用表->修改表->启用表 直接修改会报错
- 删除表中的列簇：alter 'music\_table',{NAME=>'action',METHOD=>'delete'}

```
hbase(main):013:0> alter 'music_table',{NAME=>'action',METHOD=>'delete'}
Updating all regions with the new schema...
0/1 regions updated.
1/1 regions updated.
Done.
0 row(s) in 2.4080 seconds

hbase(main):014:0> describe 'music_table'
Table music_table is ENABLED
music_table
COLUMN FAMILIES DESCRIPTION
{NAME => 'meta_data', DATA_BLOCK_ENCODING => 'NONE', BLOOMFILTER => 'ROW', REPLICATION_S
> 'FOREVER', KEEP_DELETED_CELLS => 'FALSE', BLOCKSIZE => '65536', IN_MEMORY => 'false',
1 row(s) in 0.0540 seconds

hbase(main):015:0>
```

## Hbase Shell

- 命令drop / exists

- 同样对表进行任何的操作都需要先禁用表->修改->启用表，删除同样

- 禁用表: `disable 'music_table'`

- 删除表: `drop 'music_table'`

- 利用list或exists命令判断表是否存在

```
disable all  
hbase(main):015:0> disable 'music_table'  
0 row(s) in 1.4100 seconds
```

```
hbase(main):016:0> drop 'music_table'  
0 row(s) in 0.2420 seconds
```

```
hbase(main):017:0> list  
TABLE  
0 row(s) in 0.0080 seconds
```

```
=> []  
hbase(main):018:0> exists 'music_table'  
Table music_table does not exist  
0 row(s) in 0.0320 seconds
```

## Hbase Shell

- 命令is\_enabled
  - 判断表是否enable或者disable

```
hbase(main):019:0> create 'music_table','meta_data','action'
0 row(s) in 0.4310 seconds

=> Hbase::Table - music_table
hbase(main):020:0> list
TABLE
music_table
1 row(s) in 0.0120 seconds

=> ["music_table"]
hbase(main):021:0> exists 'music_table'
Table music_table does exist
0 row(s) in 0.0310 seconds

hbase(main):022:0> is_enabled 'music_table'
true
0 row(s) in 0.0430 seconds

hbase(main):023:0> █
```

## Hbase Shell

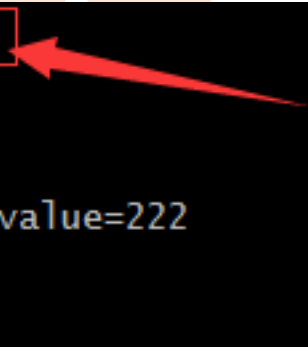
- 插入命令put
  - 对于hbase来说insert update其实没有什么区别，都是插入原理
  - 在hbase中没有数据类型概念，都是“字符类型”，至于含义在程序中体现
  - 每插入一条记录都会自动建立一个时间戳，由系统自动生成。也可手动“强行指定”

```
hbase(main):023:0> put 'music_table', '111', 'meta_data:name', '222'
0 row(s) in 0.3300 seconds

hbase(main):024:0> get 'music_table', '111'
COLUMN                                CELL
meta_data:name                        timestamp=1491049900823, value=222
1 row(s) in 0.0280 seconds

hbase(main):025:0> scan 'music_table'
ROW
111
1 row(s) in 0.0260 seconds

hbase(main):026:0>
```





## Hbase Shell

## • 插入命令put

```
hbase(main):005:0> put 'music_table', '111', 'meta_data:desc', 'ddd'
0 row(s) in 0.0390 seconds

hbase(main):006:0> scan 'music_table'
ROW                                COLUMN+CELL
 111                                column=meta_data:desc, timestamp=1491196047756, value=ddd
 111                                column=meta_data:name, timestamp=1491049900823, value=222
1 row(s) in 0.0170 seconds

hbase(main):007:0>
```

```
hbase(main):015:0> get 'music_table', '111', 'meta_data'
COLUMN                                CELL
 meta_data:desc                      timestamp=1491196047756, value=ddd
 meta_data:name                      timestamp=1491049900823, value=222
2 row(s) in 0.0290 seconds

hbase(main):016:0> put 'music_table', '111', 'meta_data:desc', 'ccc'
0 row(s) in 0.0140 seconds

hbase(main):017:0> get 'music_table', '111', 'meta_data'
COLUMN                                CELL
 meta_data:desc                      timestamp=1491196550083, value=ccc
 meta_data:name                      timestamp=1491049900823, value=222
2 row(s) in 0.0620 seconds

hbase(main):018:0> █
```

## Hbase Shell

- 指定版本

```
hbase(main):020:0> get 'music_table', '111', {COLUMN=>'meta_data:desc', TIMESTAMP=>1491196550083}
COLUMN                                CELL
meta_data:desc                        timestamp=1491196550083, value=ccc
1 row(s) in 0.0170 seconds

hbase(main):021:0> get 'music_table', '111', {COLUMN=>'meta_data:desc', TIMESTAMP=>1491196047756}
COLUMN                                CELL
meta_data:desc                        timestamp=1491196047756, value=ddd
1 row(s) in 0.0080 seconds

hbase(main):022:0> get 'music_table', '111', {COLUMN=>'meta_data:desc'}
COLUMN                                CELL
meta_data:desc                        timestamp=1491196550083, value=ccc
1 row(s) in 0.0150 seconds

hbase(main):023:0>
```

- 修改版本存储个数:

- alter 'music\_table',{NAME=>'meta\_data', VERSIONS=>3}

```
hbase(main):025:0> get 'music_table', '111', {COLUMN=>'meta_data:desc', VERSIONS=>3}
COLUMN                                CELL
meta_data:desc                        timestamp=1491198737667, value=zzz
meta_data:desc                        timestamp=1491196550083, value=ccc
2 row(s) in 0.0170 seconds
```

## Hbase Shell

- 查看有多少条记录count
  - count 'music\_table'

```
hbase(main):023:0> count 'music_table'  
1 row(s) in 0.0720 seconds  
  
=> 1  
hbase(main):024:0> █
```

## Hbase Shell

## • 删除delete

- 删除指定列簇
- 删除整行

```
hbase(main):002:0> delete 'music_table', '111', 'meta_data:desc'
0 row(s) in 0.0210 seconds

hbase(main):003:0> scan 'music_table'
ROW                                COLUMN+CELL
111                                column=meta_data:name, timestamp=1491049900823, value=222
1 row(s) in 0.0500 seconds
```

## • 截断表truncate

- 注意：truncate表的处理过程：由于Hadoop的HDFS文件系统不允许直接修改，所以只能先删除表在重新创建已达到清空表的目的

```
hbase(main):005:0> deleteall 'music_table', '111'
0 row(s) in 0.0220 seconds

hbase(main):006:0> get 'music_table', '111', {COLUMN=>'meta_data:desc', VERSIONS=>3}
COLUMN                                CELL
0 row(s) in 0.0090 seconds

hbase(main):007:0> scan 'music_table'
ROW                                COLUMN+CELL
0 row(s) in 0.0090 seconds

hbase(main):008:0>
```

```
hbase(main):011:0> truncate 'music_table'
Truncating 'music_table' table (it may take a while):
- Disabling table...
- Truncating table...
0 row(s) in 2.0080 seconds

hbase(main):012:0> scan 'music_table'
ROW                                COLUMN+CELL
0 row(s) in 0.1530 seconds

hbase(main):013:0>
```

## H b a s e   S h e l l

- Split
  - 手动
    - `split 'music_table', 'bc31bc83af45aab95d5d8a62962b23f5'`
  - 建表时预设
    - `create 'test_table', 'f1', SPLITS=> ['a', 'b', 'c']`
- Compact
  - `merge_region '759a217c34ad5203801866dab4b6b209', '939affd918502d5e46792367a0a4a59a', true`
  - `major_compact 'music_table'`

## Outline

HBase基础

【实践】HBase搭建

【实践】Hbase Shell

【实践】Hbase的Python操作

## Hbase 的 Python 操作

## • 安装Thrift:

- ]# wget http://archive.apache.org/dist/thrift/0.8.0/thrift-0.8.0.tar.gz
- ]# tar xzf thrift-0.8.0.tar.gz
- ]# yum install automake libtool flex bison pkgconfig gcc-c++ boost-devel libevent-devel zlib-devel python-devel ruby-devel openssl-devel
- ]# yum install boost-devel.x86\_64
- ]# yum install libevent-devel.x86\_64

## Hbase 的 Python 操作

- 安装Thrift:
  - [root@master thrift-0.8.0]# pwd
  - /home/badou/hbase\_test/thrift-0.8.0
  - ]# ./configure --with-cpp=no --with-ruby=no
  - ]# make
  - ]# make install



## Hbase 的 Python 操作

- 产生针对Python的Hbase的API:
  - 下载hbase源码:
  - ]# wget http://mirrors.hust.edu.cn/apache/hbase/0.98.24/hbase-0.98.24-src.tar.gz
  - [root@master hbase-0.98.24]# find . -name Hbase.thrift
  - ./hbase-thrift/src/main/resources/org/apache/hadoop/hbase/thrift/Hbase.thrift
  - [root@master hbase-0.98.24]# cd ./hbase-thrift/src/main/resources/org/apache/hadoop/hbase/thrift
  - ]# thrift -gen py Hbase.thrift
  - ]# cp -raf gen-py/hbase/ /home/badou/hbase\_test

## Hbase 的 Python 操作

- 启动Thrift服务
  - ]# hbase-daemon.sh start thrift

```
searching on file, logging to /usr/local/hbase/
[root@master hbase_test]# jps
113183 HRegionServer
113887 ThriftServer
113469 Main
113043 HMaster
55877 JobTracker
72268 HQuorumPeer
55639 NameNode
113967 Jps
55791 SecondaryNameNode
[root@master hbase_test]#
```

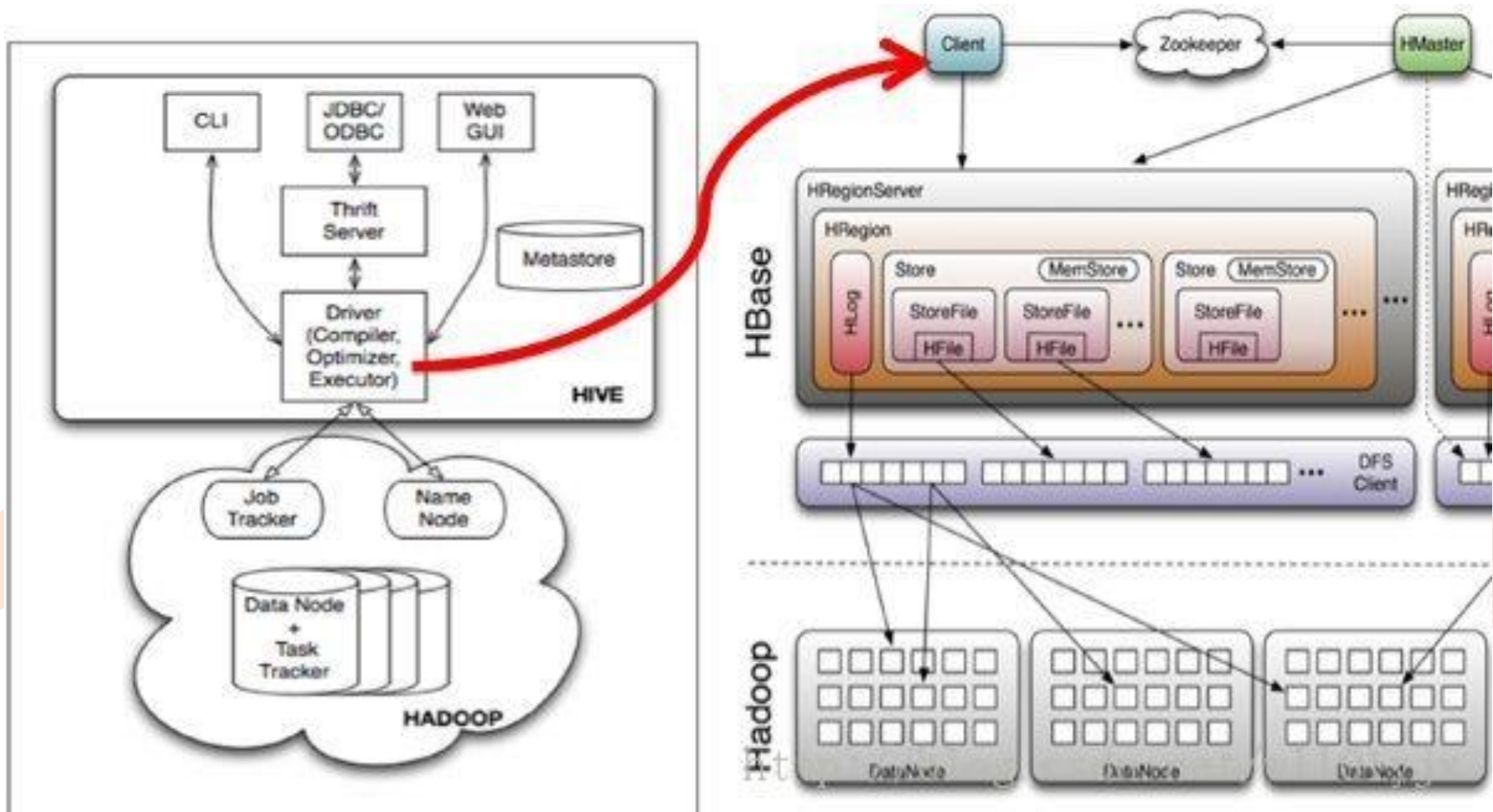
## Hbase 的 Python 操作

- 实例1：创建表
- 实例2：插入数据
- 实例3：读取指定row key记录
- 实例4：读取多条记录

## Hive 整合 Hbase

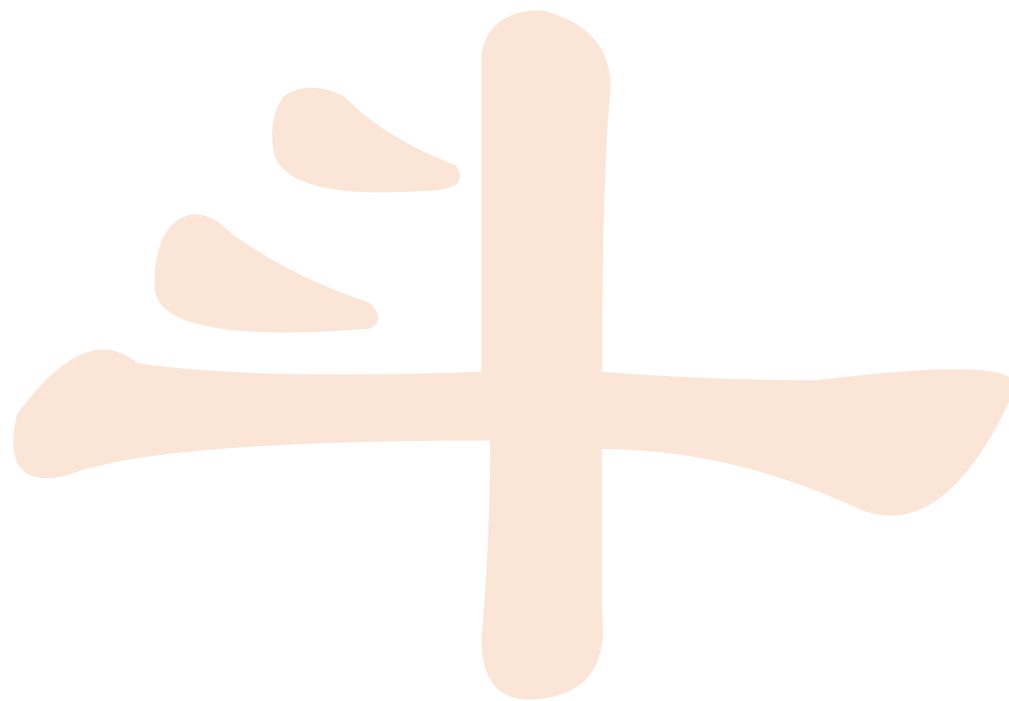
- HBase是被设计用来做k-v查询的，但有时候，也会遇到基于HBase表的复杂统计，写MR很不方便。Hive考虑到了这点，提供了操作HBase表的接口。
- Hive读取HBase表，通过MR,最终使用HiveHBaseTableInputFormat来读取数据，在getSplit()方法中对 HBase表进行切分，切分原则是根据该表对应的HRegion，将每一个Region作为一个InputSplit，即，该表有多少个Region,就有多少个Map Task;
- 每个Region的大小由参数hbase.hregion.max.filesize控制，默认10G，这样会使得每个map task处理的数据文件太大，map task性能自然很差;
- 为HBase表预分配Region，使得每个Region的大小在合理的范围;

## Hive 整合 Hbase



## H i v e 整 合 H b a s e

- 创建Hbase表：
  - create 'classes','user'
- 加入数据：
  - put 'classes','001','user:name','jack'
  - put 'classes','001','user:age','20'
  - put 'classes','002','user:name','liza'
  - put 'classes','002','user:age','18'



## Hive 整合 Hbase

- 创建Hive表并验证：
  - create external table classes(id int, name string, age int)
  - STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
  - WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,user:name,user:age")
  - TBLPROPERTIES("hbase.table.name" = "classes");
- 再添加数据到Hbase：
  - put 'classes','003','user:age','1820183291839132'

---

# Q & A

@八斗学院

---