北京理工大學
BEIJING INSTITUTE OF TECHNOLOGY

分布式数据库
HBase

大数据处理技术

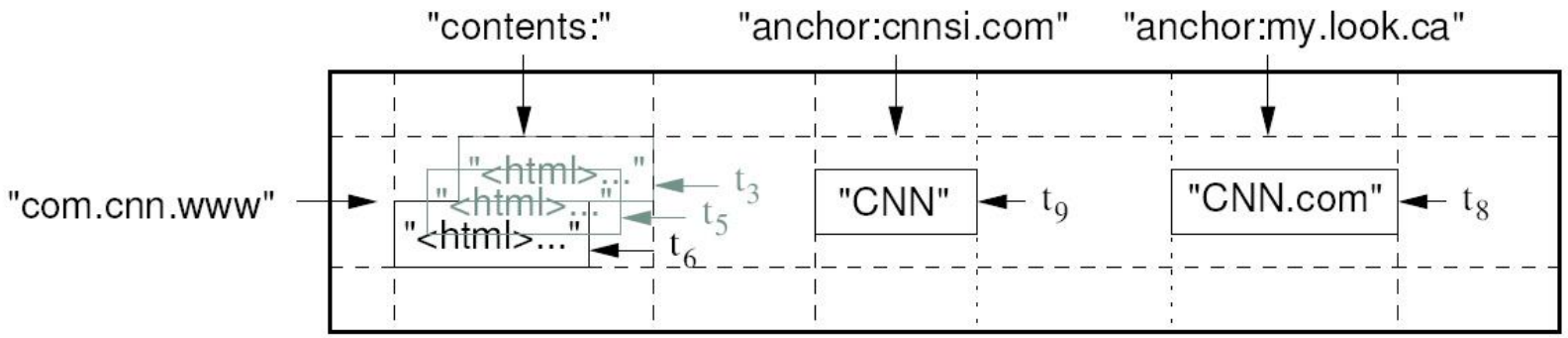
计算机学院

课程提纲

- HBase简介
- HBase数据模型
- HBase体系结构
- HBase实现原理
- Hbase运行机制
- HBase编程实践

从BigTable说起

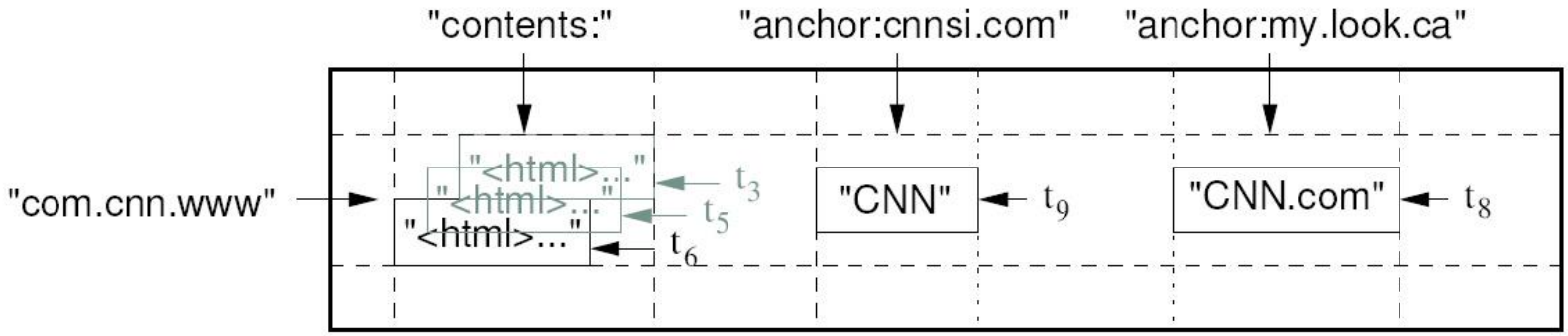
- HBase是谷歌BigTable的开源实现
- BigTable是一个分布式存储系统
- 利用MapReduce分布式并行计算模型来处理海量数据
- 使用谷歌分布式文件系统GFS作为底层数据存储
- 可以扩展到PB级别的数据和上千台机器，具备广泛应用性、可扩展性、高性能和高可用性等特点



网页在BigTable中的存储样例

从BigTable说起

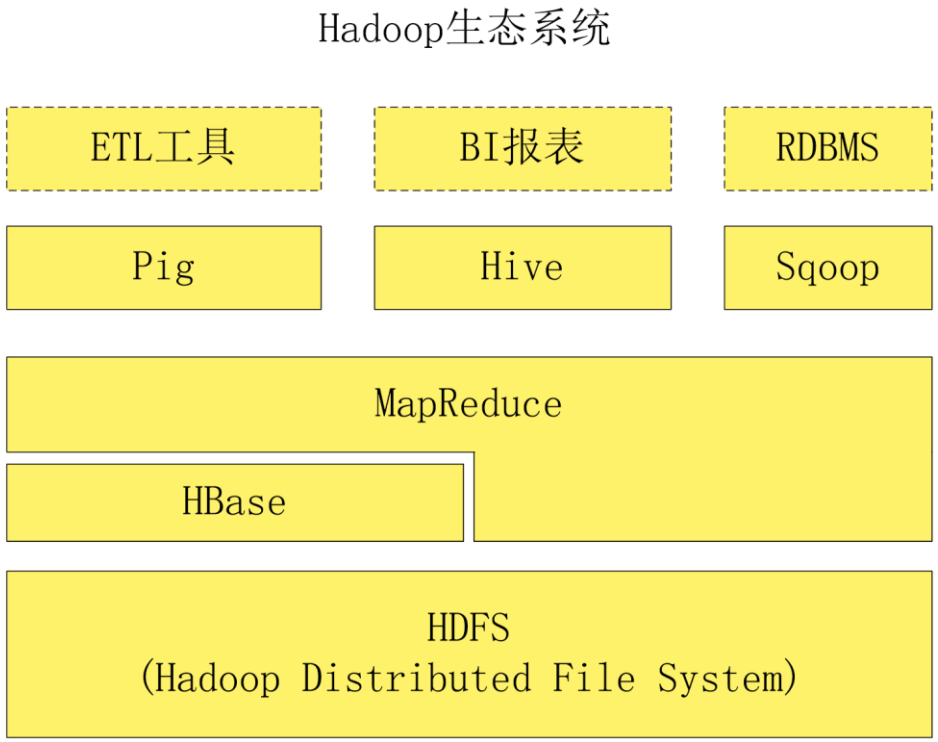
- BigTable起初用于解决大规模网页搜索问题
- 建立网页索引
  - 爬虫抓取新页面，页面每页一行地存储到BigTable里
  - MapReduce计算作业运行在整张表上，生成索引
- 搜索互联网
  - 用户发起网络搜索请求
  - 网络搜索应用查询索引，从BigTable得到网页
  - 搜索结果返回给用户



网页在BigTable中的存储样例

HBase简介

HBase是一个高可靠、高性能、面向列、可伸缩的分布式数据库，是谷歌BigTable的开源实现，主要用来存储非结构化和半结构化的松散数据。HBase的目标是处理非常庞大的表，可以通过水平扩展的方式，利用廉价计算机集群处理由超过10亿行数据和数百万列元素组成的数据表



- 利用MR来处理hbase中的海量数据，实现高性能运算
- 利用HDFS作为高可靠的底层存储，利用廉价集群提供海量数据存储能力
- 利用zookeeper提供协同服务，实现稳定服务和错误恢复
- Sqoop为Hbase提供高效便捷的RDBMS数据导入功能
- Pip和hive提供了高层语言支持

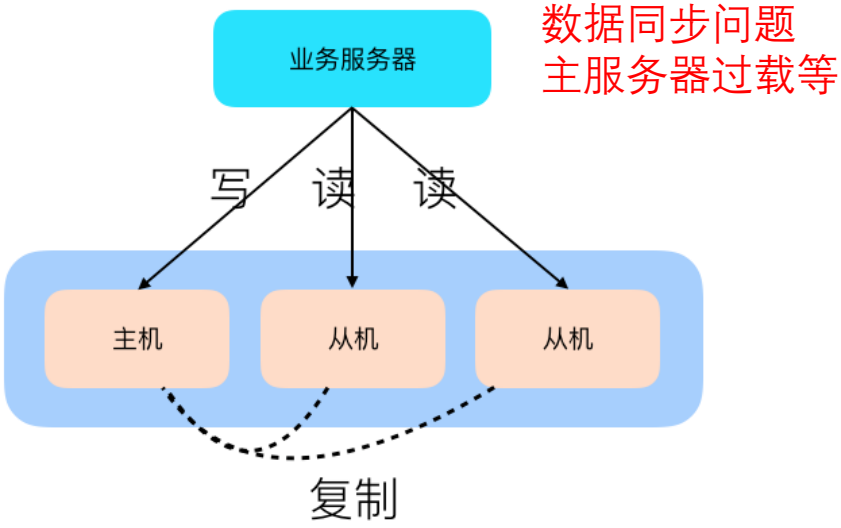
HBase简介

关系数据库已经流行很多年，并且Hadoop已经有了HDFS和MapReduce，为什么需要HBase?

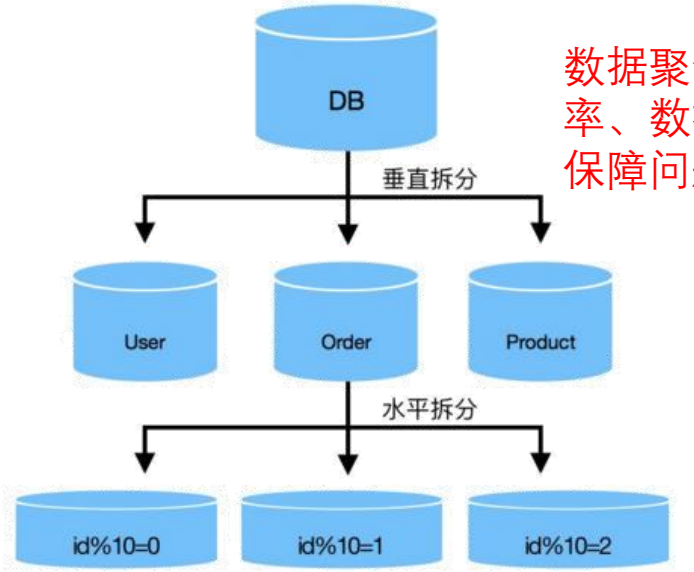
- Hadoop可以很好地解决大规模数据的离线批量处理问题，但是受限于Hadoop MapReduce编程框架的高延迟数据处理机制，使得Hadoop无法满足大规模数据实时处理应用的需求
- HDFS面向批量访问模式，不是随机访问模式

HBase简介

- 传统的通用关系型数据库无法应对在数据规模剧增时导致的系统扩展性和性能问题
  - 主从复制：读写分流
  - 分库分表
- 传统关系数据库在数据结构变化时一般需要停机维护；空列浪费存储空间



数据同步问题
主服务器过载等



数据聚合查询效率、数据一致性保障问题等

HBase VS. 传统关系数据库

- 数据类型：关系数据库采用关系模型，具有丰富的数据类型和存储方式，HBase采用了更加简单的数据模型，它把数据存储为未经解释的字符串
- 数据操作：关系数据库中包含了丰富的操作，其中会涉及复杂的多表连接。HBase操作则不存在复杂的表与表之间的关系，只有简单的插入、查询、删除、清空等，因为HBase在设计上就避免了复杂的表和表之间的关系
- 存储模式：关系数据库是基于行模式存储的，HBase是基于列存储的，不同列族的文件是分离的

HBase VS. 传统关系数据库

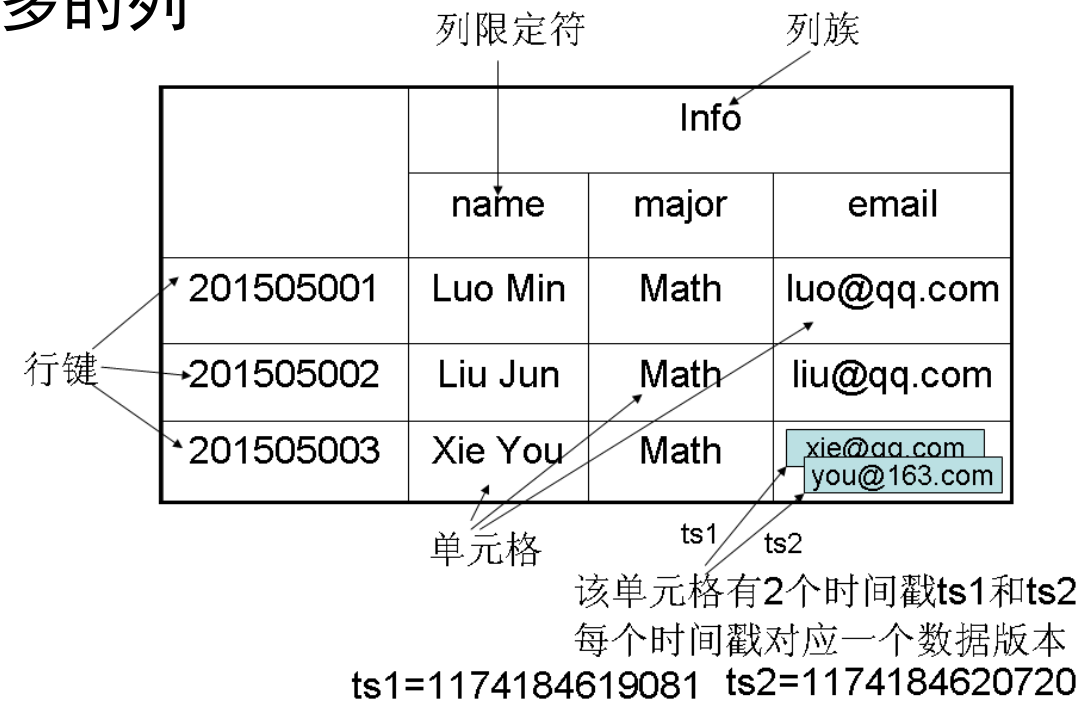
- 数据索引：关系数据库通常可以针对不同列构建复杂的多个索引，以提高数据访问性能。HBase只有一个索引即行键，HBase中的所有访问方法，或通过行键访问，或通过行键扫描
- 数据维护：在关系数据库中，更新操作会用最新的当前值去替换记录中原来的旧值，旧值被覆盖后就不会存在。而在HBase中执行更新操作时，并不会删除数据旧的版本，而是生成一个新的版本，旧有的版本仍然保留
- 可伸缩性：关系数据库很难实现横向扩展，纵向扩展的空间也比较有限。HBase分布式数据库能够通过在集群中增加或者减少硬件数量来实现性能的伸缩

HBase访问接口

类型	特点	场合
Native Java API	最常规和高效的访问方式	适合Hadoop MapReduce作业并行批处理HBase表数据
HBase Shell	HBase的命令行工具，最简单的接口	适合HBase管理使用
Thrift Gateway	利用Thrift序列化技术，支持C++、PHP、Python等多种语言	适合其他异构系统在线访问HBase表数据
REST Gateway	解除了语言限制	支持REST风格的Http API访问HBase
Pig/Hive	提供类似SQL的查询语言	适合做数据统计分析

HBase数据模型

- HBase是一个稀疏、多维度、排序的映射表，这张表的索引是行键、列族、列限定符和时间戳
- 每个值是一个未经解释的字符串，没有数据类型
- 用户在表中存储数据，每一行都有一个可排序的行键和任意多的列

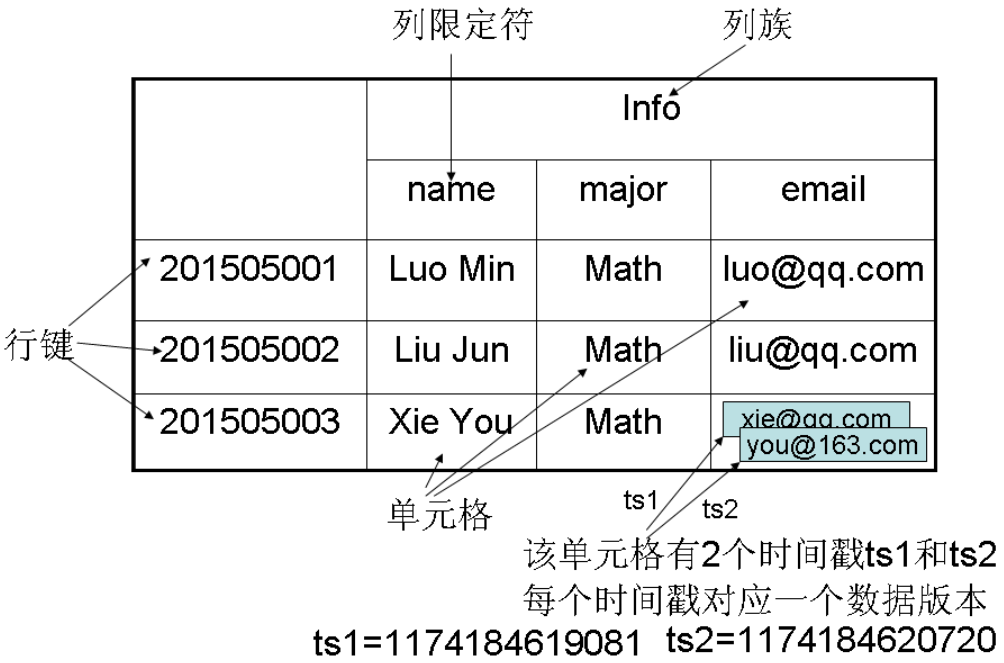


HBase数据模型

- HBase是一个稀疏、多维度、排序的映射表，这张表的索引是行键、列族、列限定符和时间戳
- 每个值是一个未经解释的字符串，没有数据类型
- 用户在表中存储数据，每一行都有一个可排序的行键和任意多的列
- 表在水平方向由一个或者多个列族组成，一个列族中可以包含任意多个列，同一个列族里面的数据存储在一起
- 列族支持动态扩展，可以很轻松地添加一个列族或列，无需预先定义列的数量以及类型，所有列均以字符串形式存储，用户需要自行进行数据类型转换
- HBase中执行更新操作时，并不会删除数据旧的版本，而是生成一个新的版本，旧有的版本仍然保留

HBase数据模型

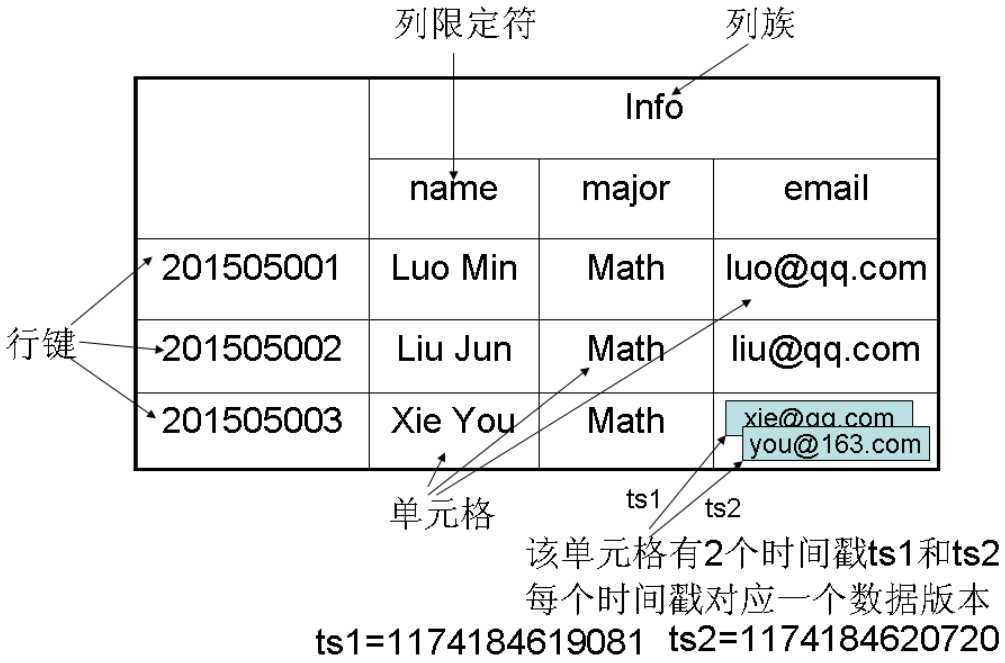
- 表：HBase采用表来组织数据，表由行和列组成，列划分为若干个列族
- 行：每个HBase表都由若干行组成，每个行由行键（row key）来标识。
- 列族：一个HBase表被分组成许多“列族”（Column Family）的集合，它是基本的访问控制单元
- 列限定符：列族里的数据通过列限定符（或列）来定位
- 单元格：在HBase表中，通过行、列族和列限定符确定一个“单元格”（cell），单元格中存储的数据没有数据类型，总被视为字节数组byte[]
- 时间戳：每个单元格都保存着同一份数据的多个版本，这些版本采用时间戳进行索引



数据坐标

- HBase中需要根据行键、列族、列限定符和时间戳来确定一个单元格，因此，可以视为一个“四维坐标”，即[行键, 列族, 列限定符, 时间戳]

键	值
[“201505003”, “Info”, “email”, 1174184619081]	“xie@qq.com”
[“201505003”, “Info”, “email”, 1174184620720]	“you@163.com”



概念视图 Conceptual View

HBase数据的概念视图

行键	时间戳	列族 contents	列族 anchor
"com.cnn.www"	t5		anchor:cnnsi.com=”CNN”
	t4		anchor:my.look.ca="CNN.com"
	t3	contents:html="<html>..."	
	t2	contents:html="<html>..."	
	t1	contents:html="<html>..."	

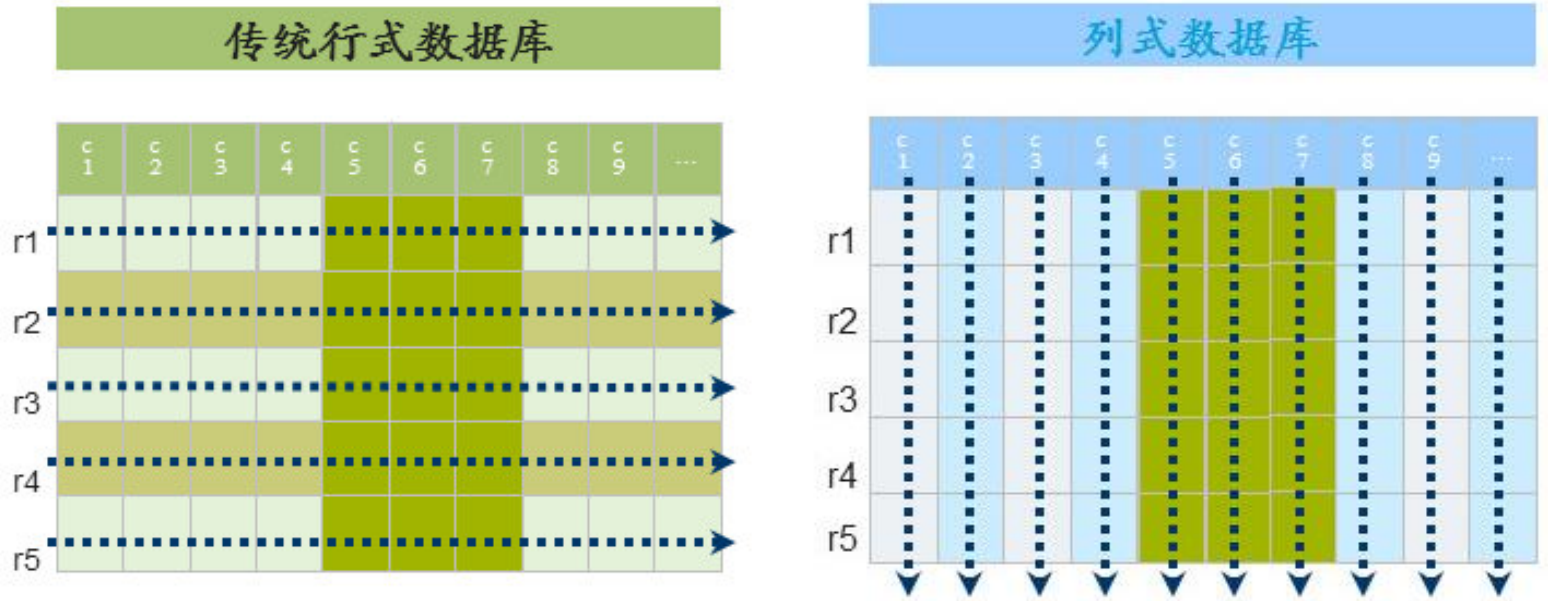
物理视图 Physical View

HBase数据的物理视图

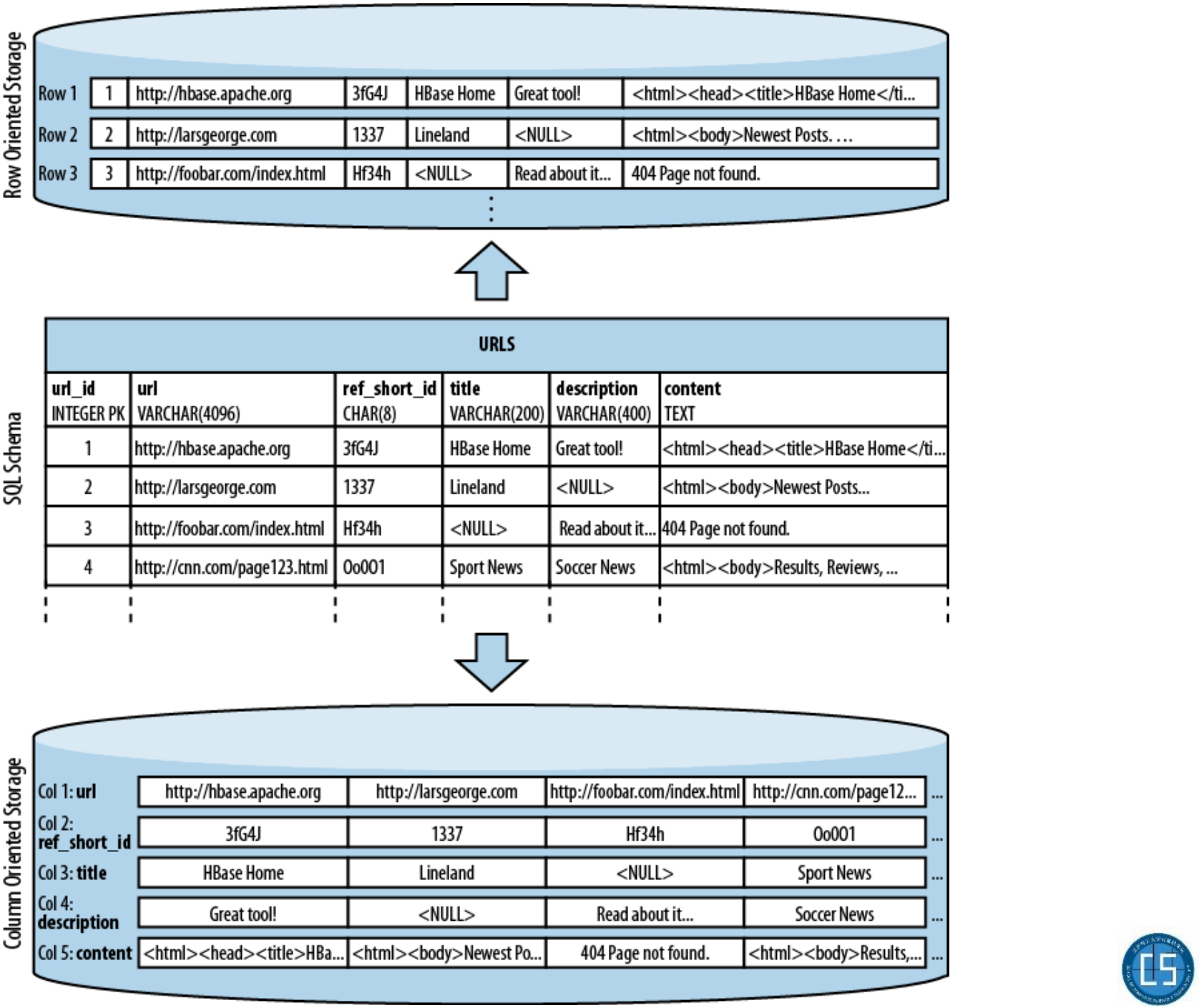
行键	时间戳	列族 contents
"com.cnn.www"	t3	contents:html="<html>..."
	t2	contents:html="<html>..."
	t1	contents:html="<html>..."

行键	时间戳	列族 anchor
"com.cnn.www"	t5	anchor:cnnsi.com=”CNN”
	t4	anchor:my.look.ca="CNN.com"

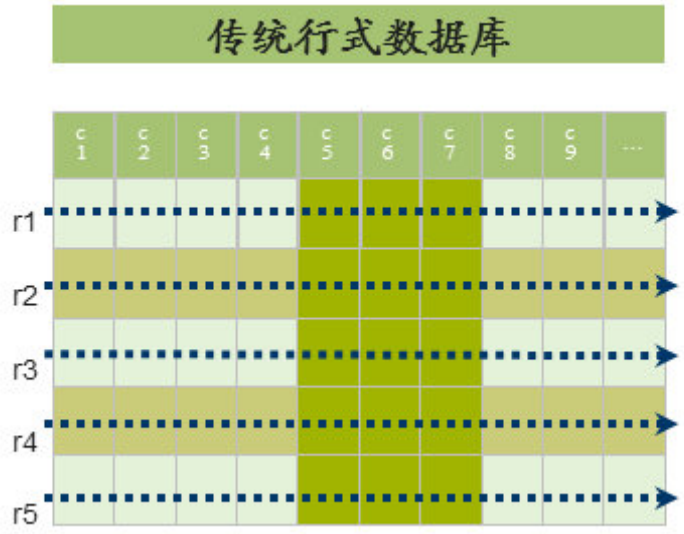
面向列的存储



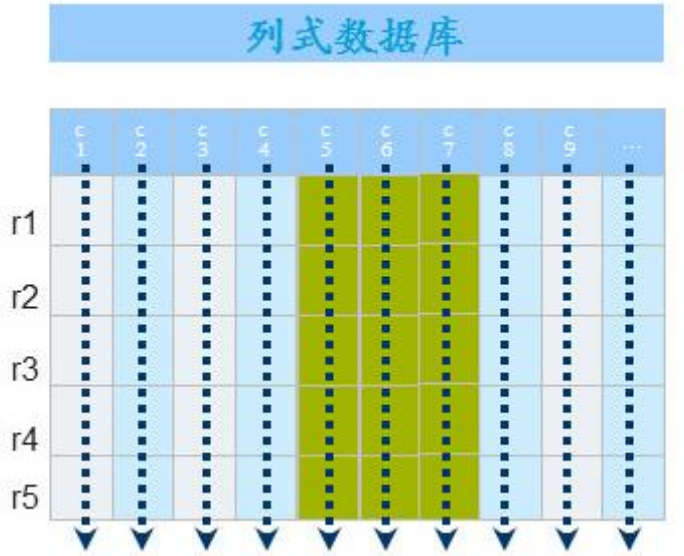
行式数据库和列式数据库示意图


Row Oriented Storage
Row 1 1 http://hbase.apache.org 3fG4J HBase Home Great tool! <html><head><title>HBase Home</ti...
Row 2 2 http://larsgeorge.com 1337 Lineland <NULL> <html><body>Newest Posts. ...
Row 3 3 http://foobar.com/index.html Hf34h <NULL> Read about it... 404 Page not found.
SQL Schema
URLS
url_id INTEGER PK
url VARCHAR(4096)
ref_short_id CHAR(8)
title VARCHAR(200)
description VARCHAR(400)
content TEXT
1 http://hbase.apache.org 3fG4J HBase Home Great tool! <html><head><title>HBase Home</ti...
2 http://larsgeorge.com 1337 Lineland <NULL> <html><body>Newest Posts...
3 http://foobar.com/index.html Hf34h <NULL> Read about it... 404 Page not found.
4 http://cnn.com/page123.html 0o001 Sport News Soccer News <html><body>Results, Reviews, ...
Column Oriented Storage
Col 1: url http://hbase.apache.org http://larsgeorge.com http://foobar.com/index.html http://cnn.com/page12... ...
Col 2: ref_short_id 3fG4J 1337 Hf34h 0o001 ...
Col 3: title HBase Home Lineland <NULL> Sport News ...
Col 4: description Great tool! <NULL> Read about it... Soccer News ...
Col 5: content <html><head><title>HBa... <html><body>Newest Po... 404 Page not found. <html><body>Results,... ...

面向列的存储



- 数据按行存储
- 操作某列必须读入整行
- 没有索引的查询占用大量I/O
- 建索引或连接表需要花费大量时间和资源
- 适用于联机事务处理



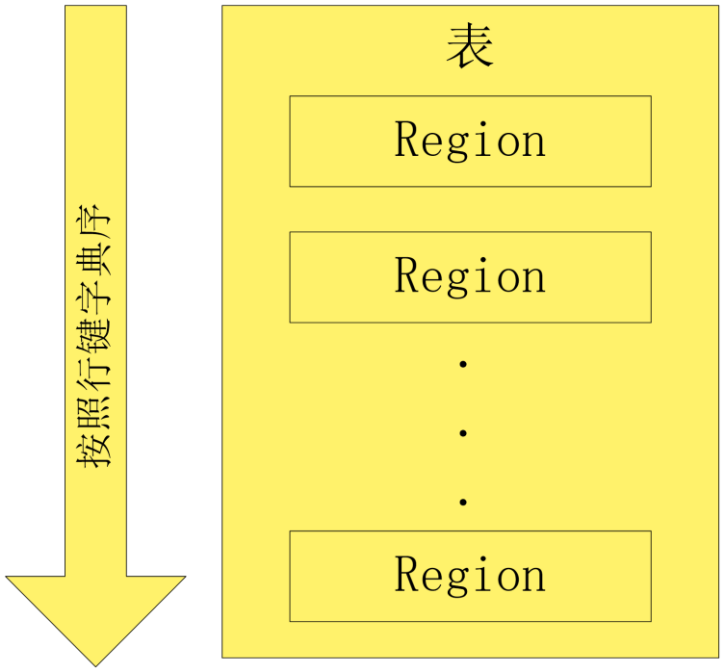
- 数据按列存储
- 只访问查询涉及的列，大量降低I/O开销
- 支持大量并发查询
- 数据类型一致特征相似，高压缩比
- 适用于查询分析型系统

HBase实现原理

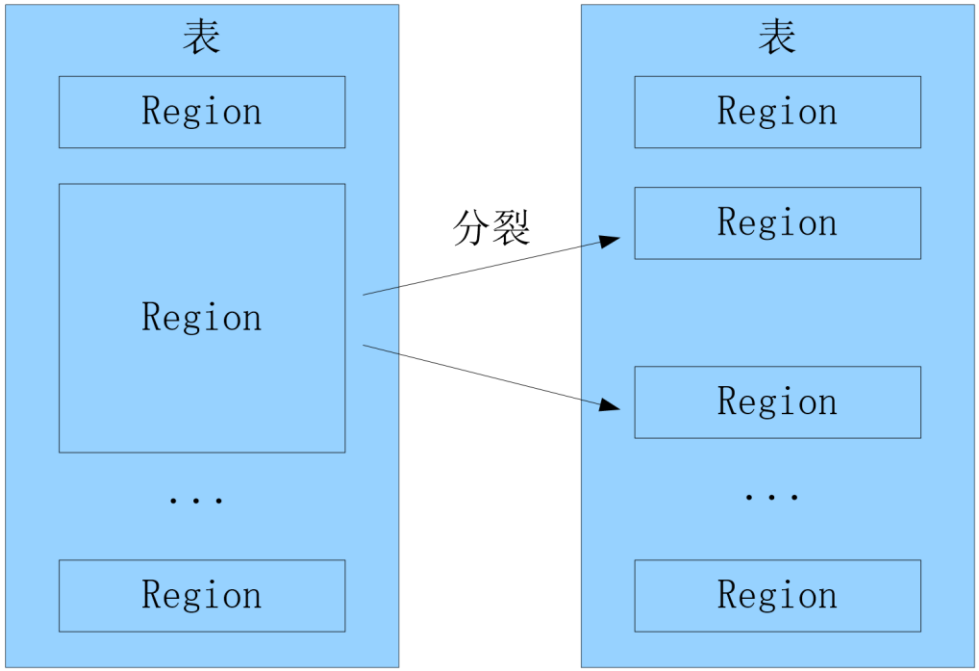
- HBase的实现包括三个主要的功能组件：
  - 库函数：链接到每个客户端
  - 一个Master主服务器
  - 许多个Region服务器
- 主服务器Master负责管理和维护HBase表的分区信息，维护Region服务器列表，分配Region，负载均衡，以及处理schema的变化，如表和列族的创建
- Region服务器负责存储和维护分配给自己的Region，处理来自客户端的读写请求
- 客户端并不是直接从Master主服务器上读取数据，而是在获得Region的存储位置信息后，直接从Region服务器上读取数据

表和Region

一个HBase表被划分成多个Region



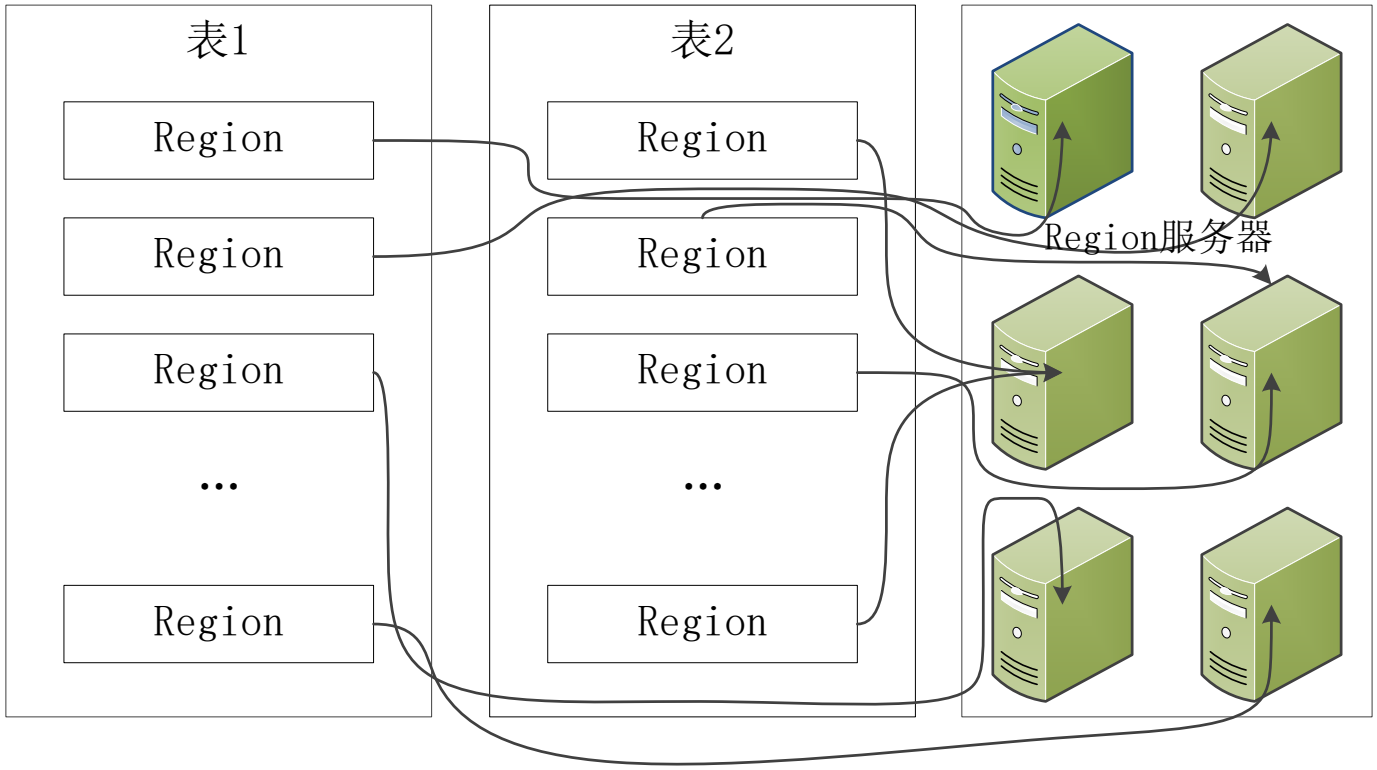
一个Region分裂成多个新的Region



- HBase中的表是根据row key的值水平分割成region的。一个region包含表中所有row key位于region的起始键值和结束键值之间的行。
- 每一个表格最初都对应于一个region，随着region中数据量的增加，region会被分割成两个子region，每一个子region中存储原来一半的数据。随着表中行的数量继续增加，分裂出越来越多的region。

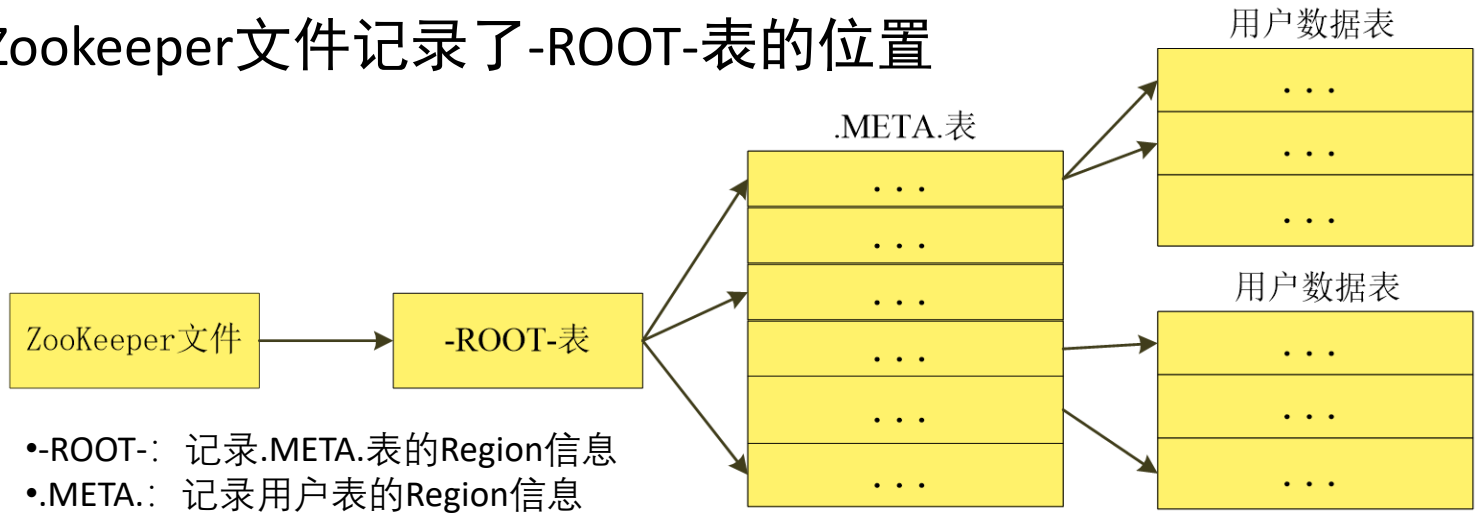
表和Region

- 每个Region的最佳大小取决于单台服务器的有效处理能力
- 通常每个Region建议1GB-2GB
- 通常每个Region服务器存储10-1000个Region
- 不同的Region可以分布在不同的Region服务器上
- 同一个Region不会被分拆到多个Region服务器



Region的定位

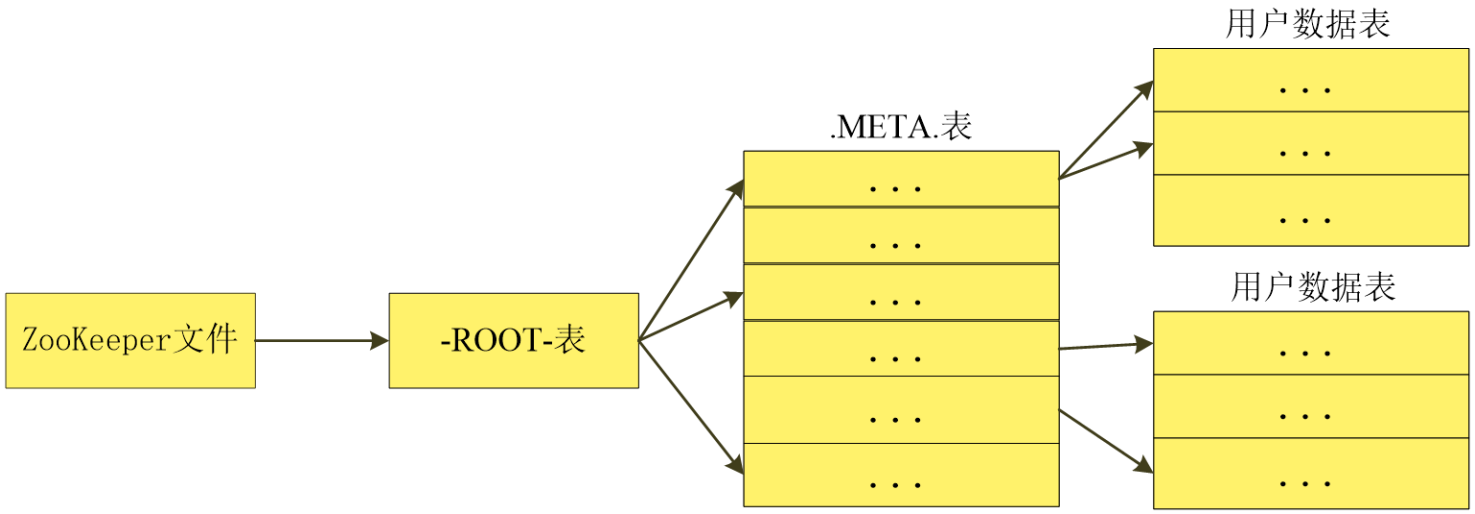
- 元数据表，又名.META.表，存储了Region和Region Server的映射关系
- 当HBase表中的Region数据非常庞大时， .META.表也会被分裂成多个Region
- 根数据表，又名-ROOT-表，记录所有元数据的具体位置
- -ROOT-表只有唯一一个Region
- Zookeeper文件记录了-ROOT-表的位置



HBase的三层结构实现Region的寻址和定位

Region的定位

- 假设.META.表的每行（一个映射条目）在内存中大约占用1KB，并且每个Region限制为128MB，那么，上面的三层结构可以保存的用户数据表的Region数目是：



Region的定位

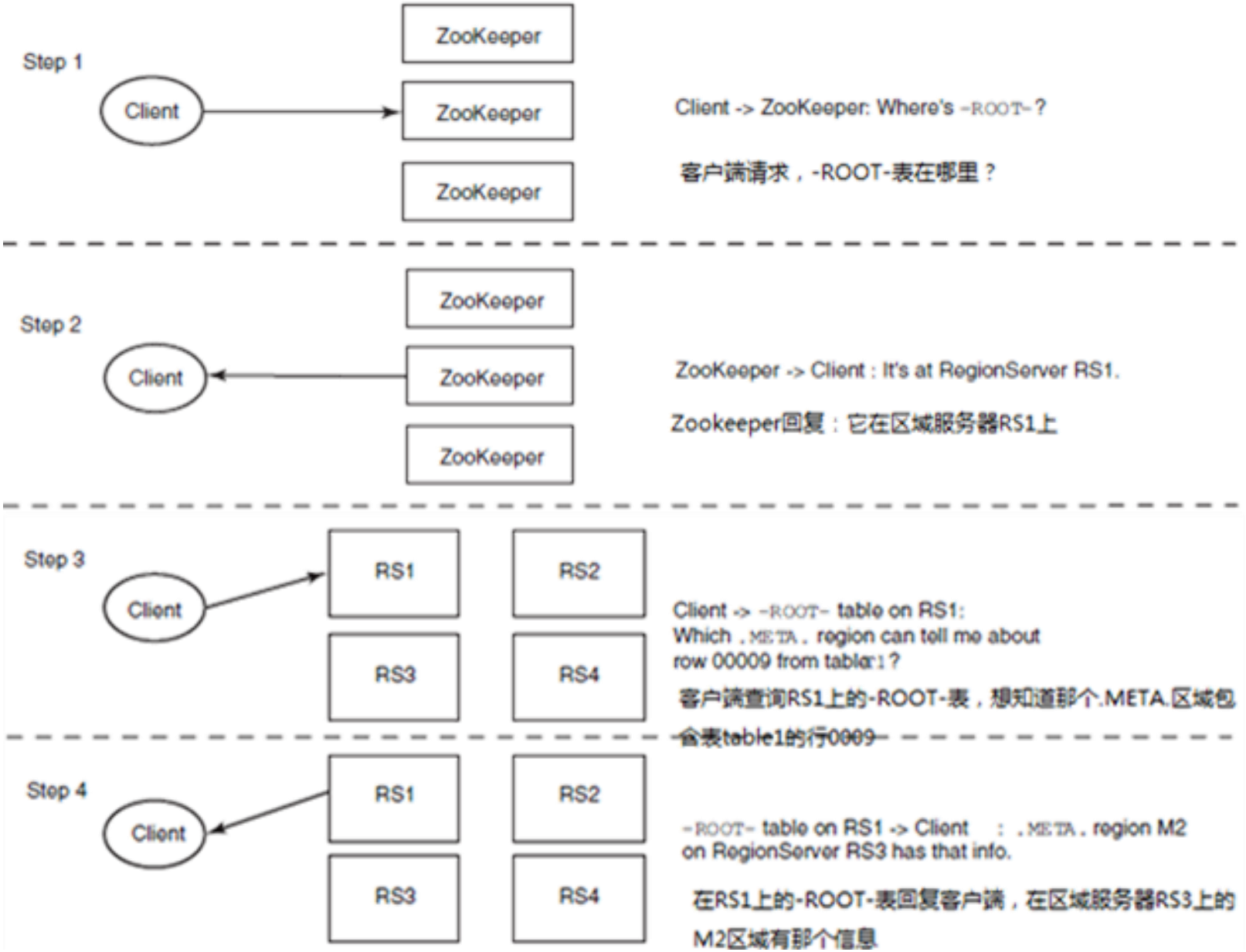
- 假设.META.表的每行（一个映射条目）在内存中大约占用1KB，并且每个Region限制为128MB，那么，上面的三层结构可以保存的用户数据表的Region数目是：
- （-ROOT-表能够寻址的.META.表的Region个数）×（每个.META.表的 Region可以寻址的用户数据表的Region个数）
- 一个-ROOT-表最多只能有一个Region，也就是最多只能有128MB，按照每行（一个映射条目）占用1KB内存计算，128MB空间可以容纳128MB/1KB=2^{17}行，也就是说，一个-ROOT-表可以寻址2^{17}个.META.表的Region。
- 同理，每个.META.表的 Region可以寻址的用户数据表的Region个数是128MB/1KB=2^{17}。
- 最终，三层结构可以保存的Region数目是(128MB/1KB) × (128MB/1KB) = 2^{34}个Region

Region的定位

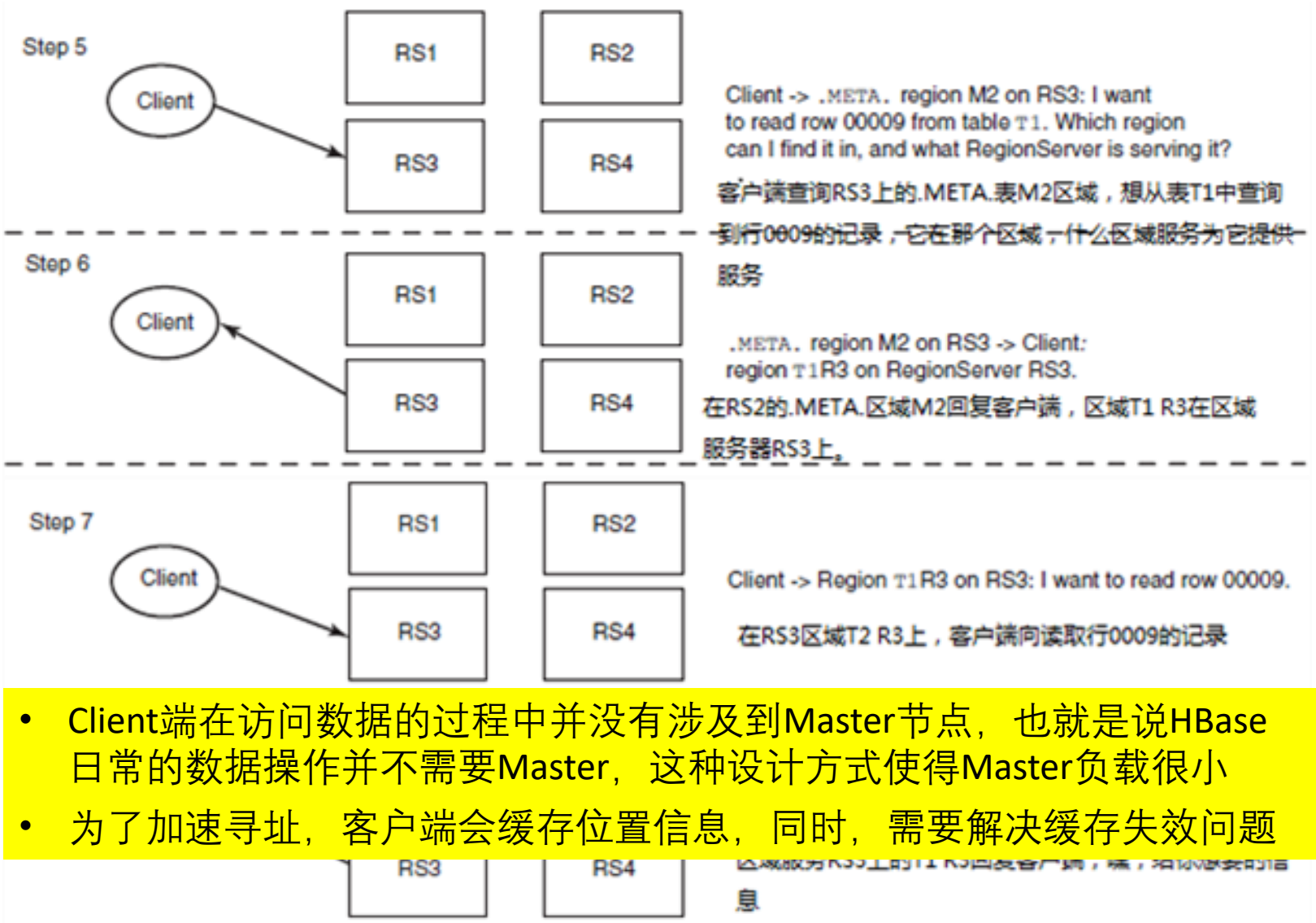
HBase的三层结构中各层次的名称和作用

层次	名称	作用
第一层	Zookeeper文件	记录了-ROOT-表的位置信息
第二层	-ROOT-表	记录了.META.表的Region位置信息 -ROOT-表只能有一个Region 通过-ROOT-表，就可以访问.META.表中的数据
第三层	.META.表	记录了用户数据表的Region位置信息 .META.表可以有多个Region 保存了所有用户数据表的Region位置信息

Region的定位

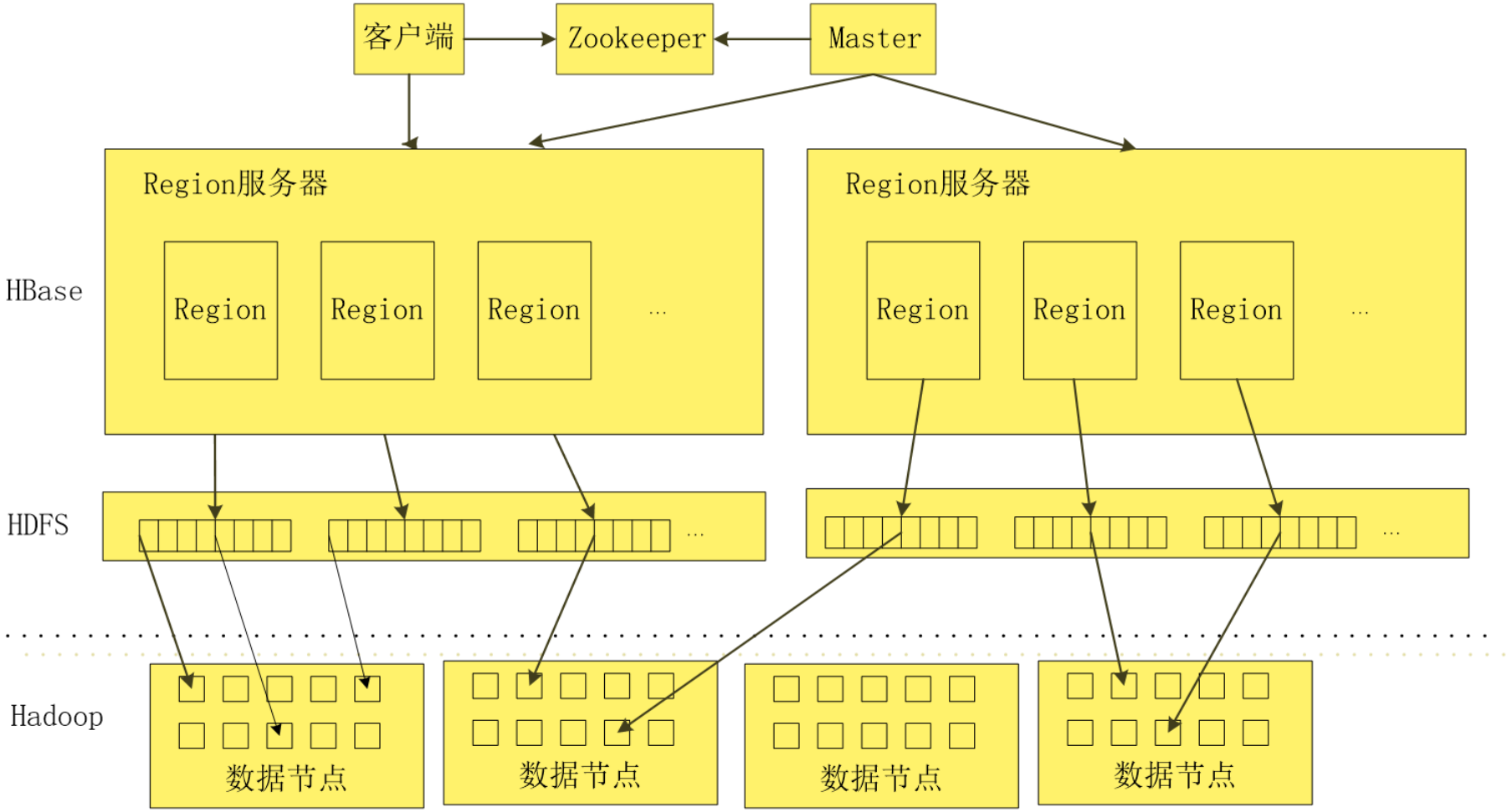

Step 1
Client
ZooKeeper
ZooKeeper
ZooKeeper
Client -> ZooKeeper: Where's -ROOT-?
客户端请求，-ROOT-表在哪里？
Step 2
Client
ZooKeeper
ZooKeeper
ZooKeeper
ZooKeeper -> Client : It's at RegionServer RS1.
Zookeeper回复：它在区域服务器RS1上
Step 3
Client
RS1
RS2
RS3
RS4
Client -> -ROOT- table on RS1:
Which .META. region can tell me about
row 00009 from table1?
客户端查询RS1上的-ROOT-表，想知道那个.META.区域包
含表table1的行0009
Step 4
Client
RS1
RS2
RS3
RS4
-ROOT- table on RS1 -> Client : .META. region M2
on RegionServer RS3 has that info.
在RS1上的-ROOT-表回复客户端，在区域服务器RS3上的
M2区域有那个信息

Region的定位



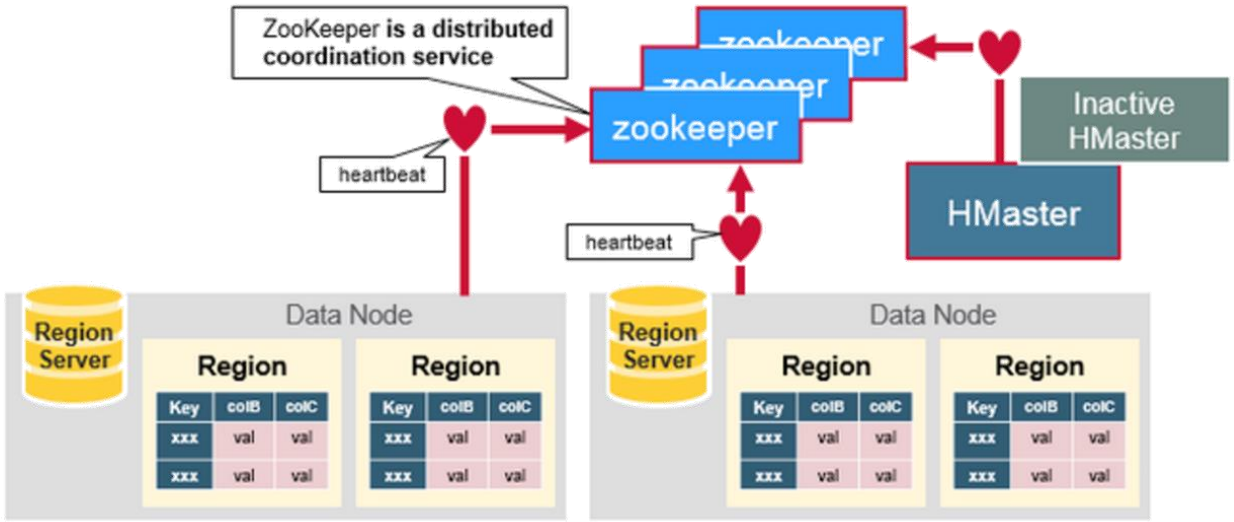
- Client端在访问数据的过程中并没有涉及到Master节点，也就是说HBase日常的数据操作并不需要Master，这种设计方式使得Master负载很小
- 为了加速寻址，客户端会缓存位置信息，同时，需要解决缓存失效问题

HBase系统架构


客户端
Zookeeper
Master
Region服务器
Region服务器
Region
Region
Region
...
Region
Region
Region
...
HBase
HDFS
...
...
Hadoop
数据节点
数据节点
数据节点
数据节点

HBase系统架构

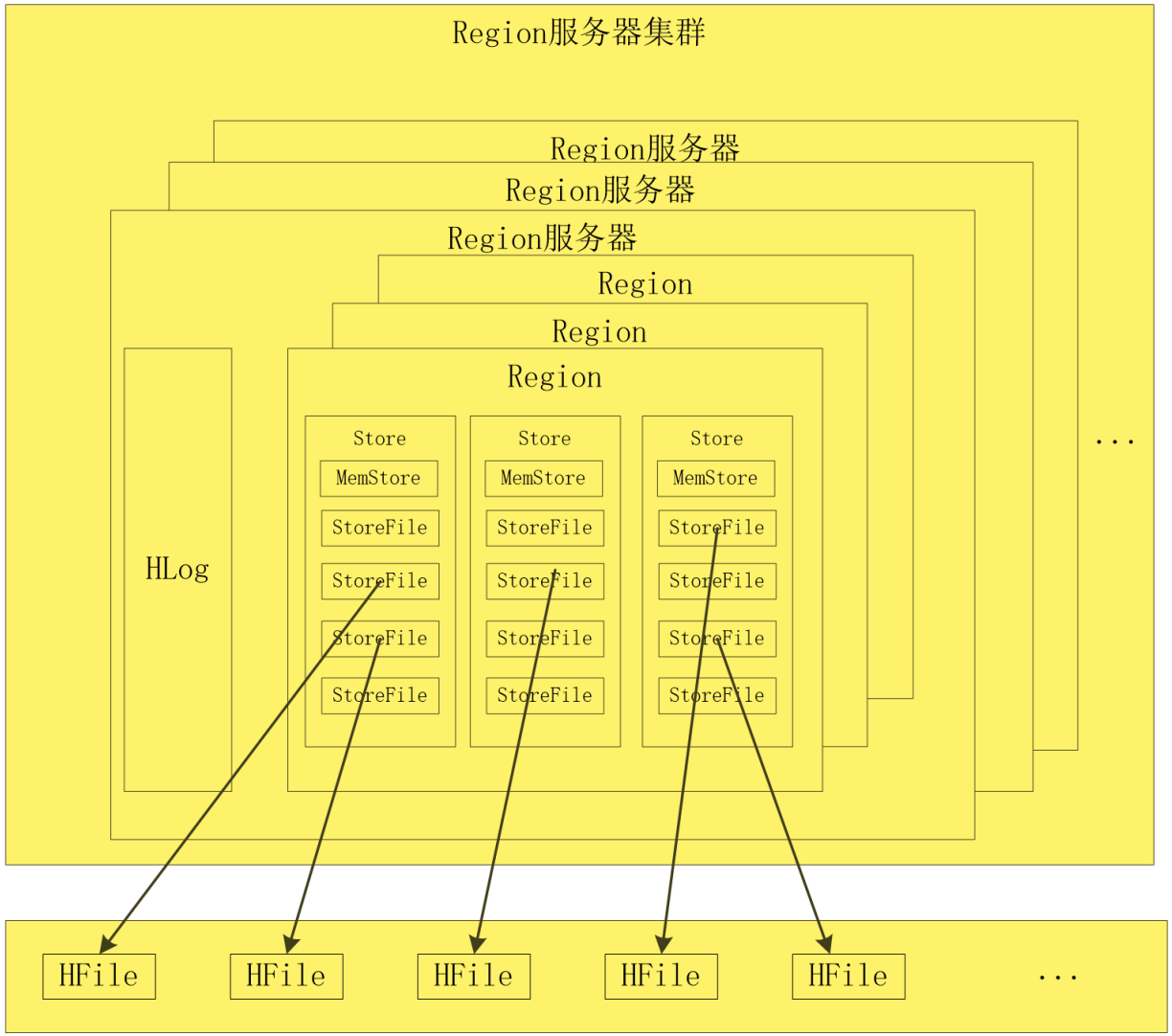
- 客户端：包含访问HBase的接口，同时在缓存中维护着已经访问过的Region位置信息，用来加快后续数据访问过程
- Zookeeper：Zookeeper存储-ROOT-表的地址和Master地址，Region Server主动向Zookeeper注册，使得Master可随时感知各Region Server的健康状态。Zookeeper可以帮助选举出一个Master作为集群的总管，并保证在任何时刻总有唯一一个Master在运行，这就避免了Master的“单点失效”问题



HBase系统架构

- Master主服务器：Master主要负责表和Region的管理工作
  - 管理用户对表的增加、删除、修改等操作
  - 实现不同Region服务器之间的负载均衡
  - 在Region分裂或合并后，负责重新调整Region的分布
  - 对发生故障失效的Region服务器上的Region进行迁移
- Region服务器：HBase中最核心的模块，负责维护分配给自己的Region，并响应用户的读写请求

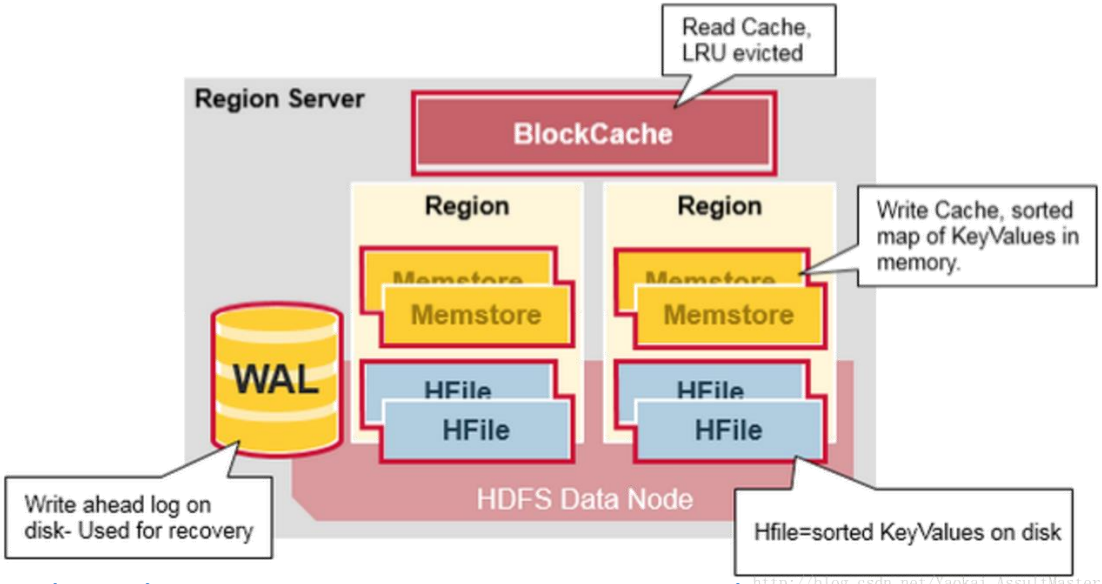
Region服务器



- Region服务器内部管理了一系列Region对象和一个共用的HLog文件（分布式文件系统中的文件，记录了所有的更新操作）
- 每个Region包含多个Store，一个Store对应一个列族
- 每个Store包含一个MemStore(缓存最近更新的数据)和若干个StoreFile（底层实现是HDFS系统中的HFile）

Region服务器

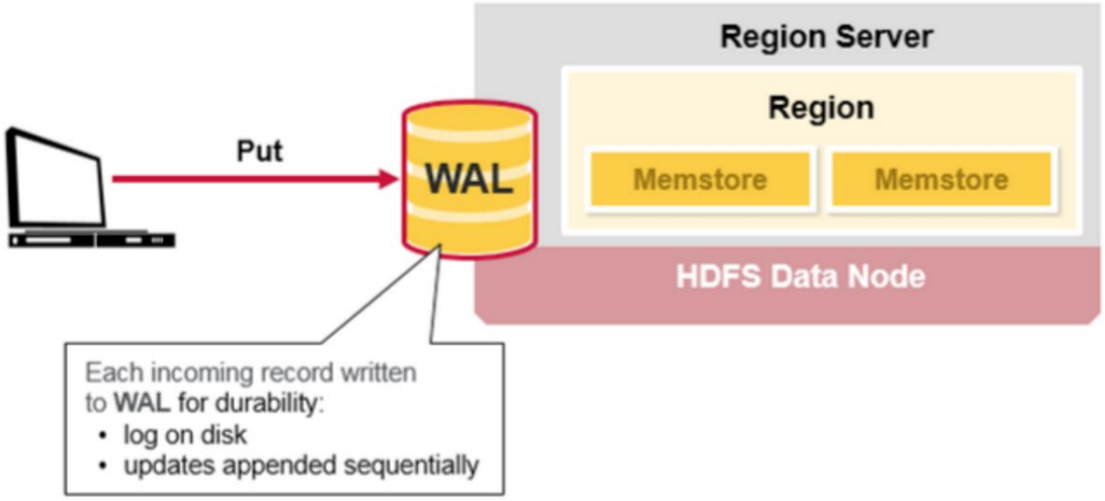
- WAL（HLog文件）：既Write Ahead Log。WAL是HDFS分布式文件系统中的一个文件。WAL用来存储尚未写入永久性存储区中的新数据。WAL也用来在服务器发生故障时进行数据恢复。
- Block Cache：Block cache是读缓存。Block cache将经常被读的数据存储在内存中来提高读取数据的效率。当Block cache的空间被占满后，其中被读取频率最低的数据将会被杀出。
- MemStore：MemStore是写缓存。其中存储了从WAL中写入但尚未写入硬盘的数据。MemStore中的数据在写入硬盘之前会先进行排序操作。每一个region中的每一个column family对应一个MemStore。
- Hfiles：Hfiles存在于硬盘上，根据排序好的键存储数据行。



https://mapr.com/blog/in-depth-look-hbase-architecture/

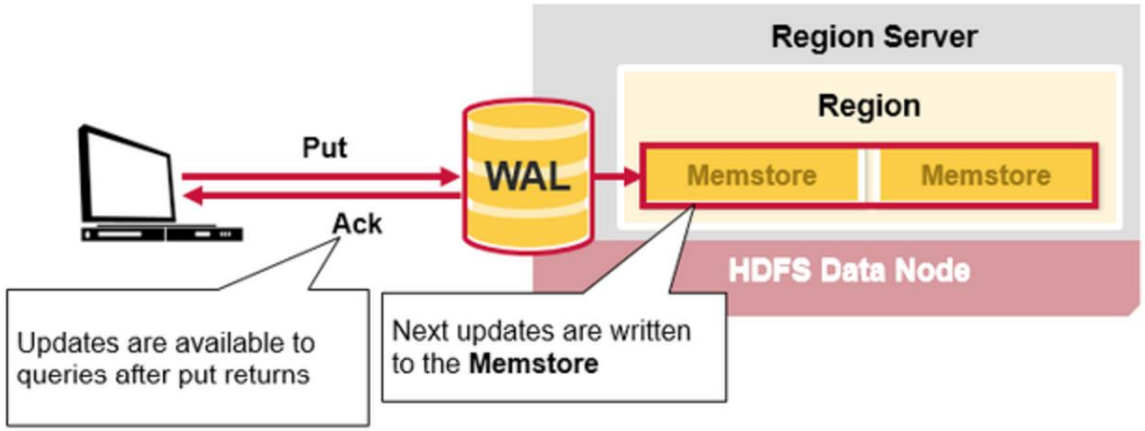
HBase的写操作步骤

- 步骤一：当用户发出一个PUT请求时（也就是HBase的写请求），HBase先是将数据写入HBase的write-ahead log（WAL）中。
- 当server出现问题之后，WAL可以被用来恢复尚未写入HBase中的数据



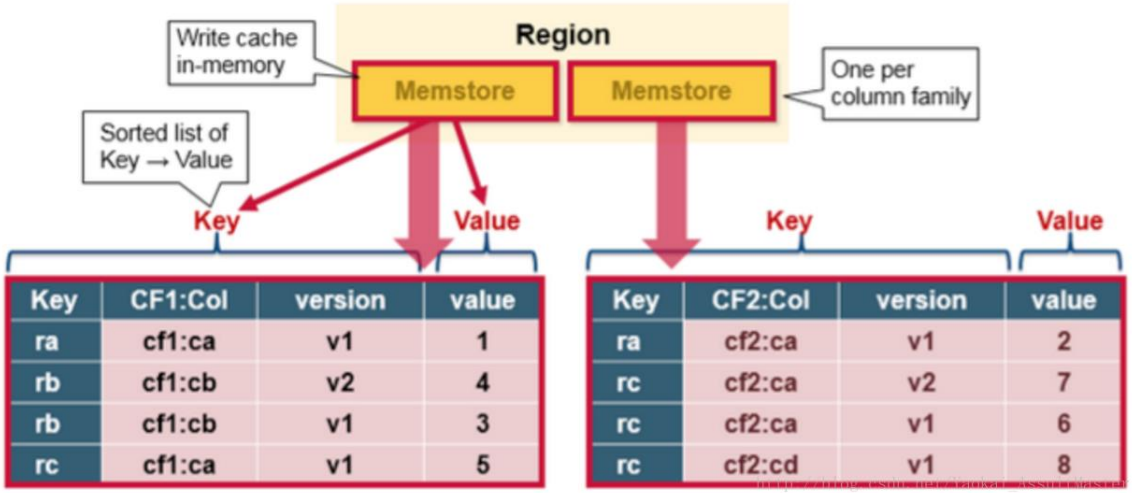
HBase的写操作步骤

- 步骤二：当数据被成功写入WAL后，HBase将数据存入MemStore。这时HBase就会通知用户PUT操作已经成功了。



缓存的刷新

- Memstore中存储的是按键排好序的待写入硬盘的数据。数据也是按键排好序写入HFile中的。每一个Region中的每一个column family对应一个Memstore
- MemStore存储在内存中，这也是为什么HBase中Column family的数目有限制的原因

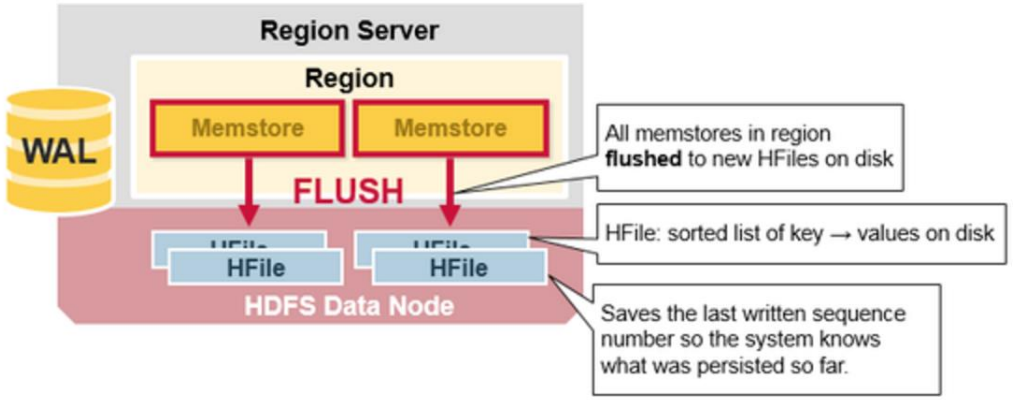


Key	CF1:Col	version	value
ra	cf1:ca	v1	1
rb	cf1:cb	v2	4
rb	cf1:cb	v1	3
rc	cf1:ca	v1	5

Key	CF2:Col	version	value
ra	cf2:ca	v1	2
rc	cf2:ca	v2	7
rc	cf2:ca	v1	6
rc	cf2:cd	v1	8

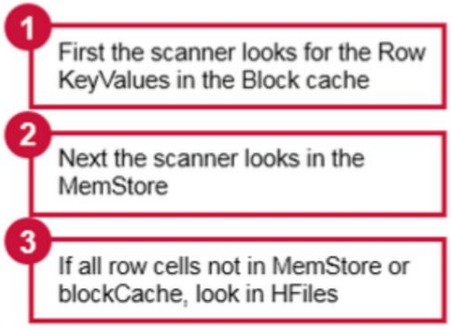
缓存的刷新

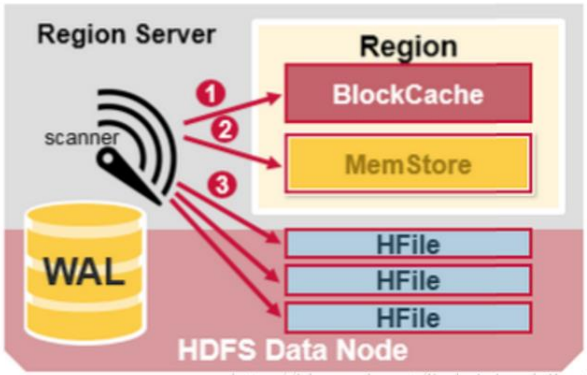
- 当MemStore中积累了足够多的数据之后，整个MemStore中的数据会被一次性写入到HDFS里的一个新的HFile中，清空缓存，并在HLog里面写入一个标记，用来表示缓存中的内容已经被写入HFile了
- 因此一个Column family可能对应多个HFile



HBase的读操作

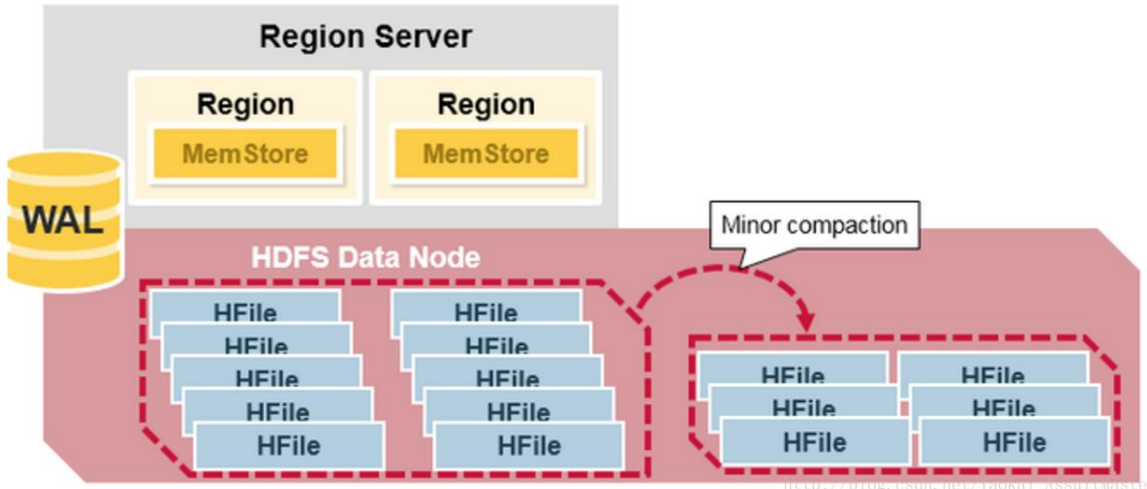
- HBase中对应于某一行数据的cell可能位于多个不同的文件或存储介质中
  - 已经存入硬盘的行位于硬盘上的HFile中
  - 新加入或更新的数据位于内存中的MemStore中
  - 最近读取过的数据则位于内存中的Block cache中
- 一个MemStore对应的数据可能存储于多个不同的HFile中（由于多次的flush），因此在读操作的时候，HBase可能需要读取多个HFile来获取想要的数据。这会影响HBase的性能表现





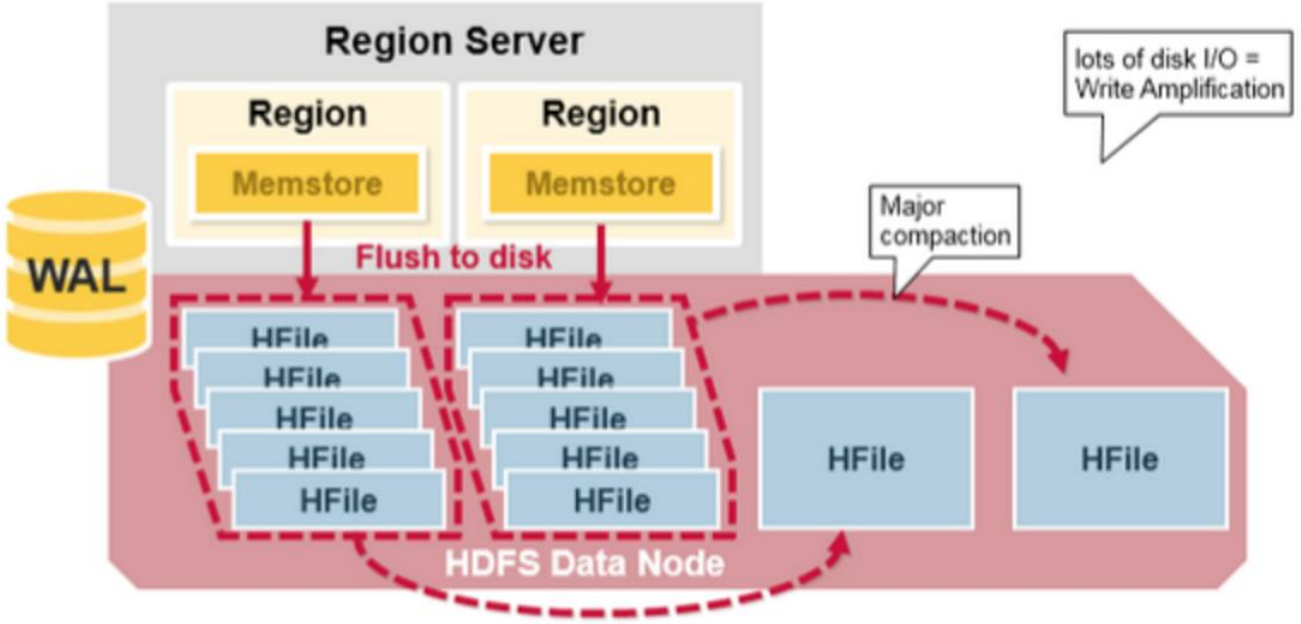
HBase的Compaction

- **Minor Compaction：** 将较小的文件合并为较大的文件，从而减少了存储的HFile的数量，提升HBase的性能



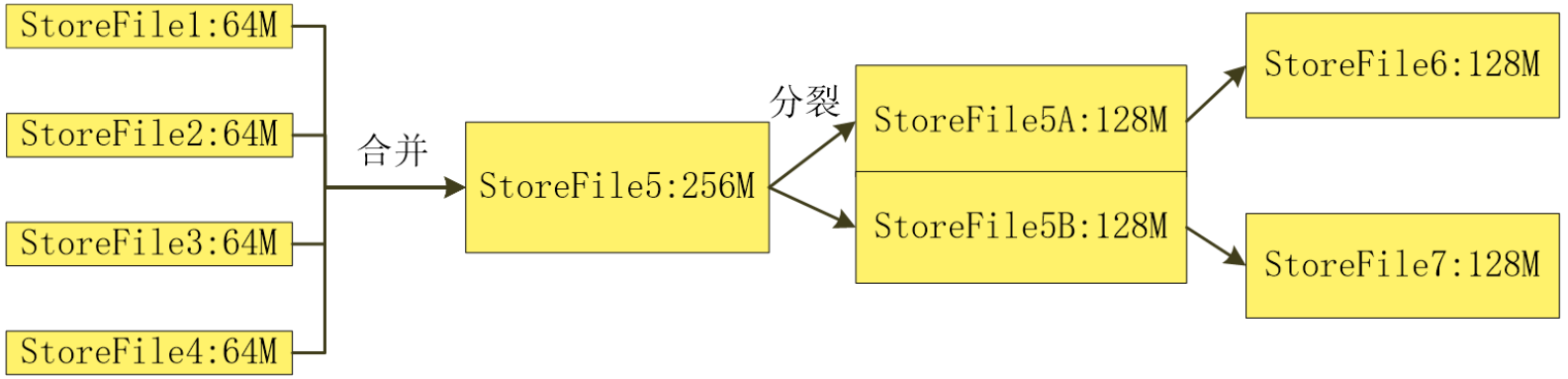
HBase的Compaction

- **Major Compaction**：HBase将对应于某一个Column family的所有HFile重新整理并合并为一个HFile，并在这一过程中删除已经删除或过期的cell，更新现有cell的值。这一操作大大提升读的效率
- Major compaction需要重新整理所有的HFile并写入一个HFile，这一过程包含大量的硬盘I/O操作以及网络数据通信（远程数据可能由于服务器故障或者负载均衡等原因而存储在于远端服务器上），在Major compaction进行的过程中，当前Region基本是处于不可访问的状态



StoreFile的合并和分裂过程

- StoreFile数量达到一个阈值时触发合并，多个StoreFile合并成一个文件
- 单个StoreFile过大时，又触发分裂操作，1个父Region被分裂成两个子Region

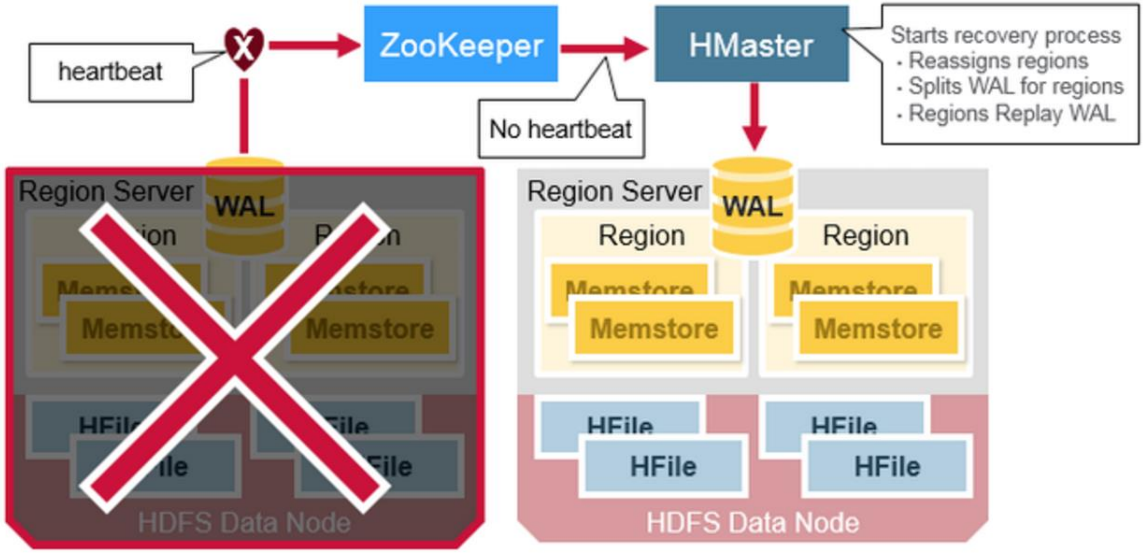


HBase的异常恢复（Crash Recovery）

- 分布式环境必须要考虑系统出错。HBase采用HLog保证系统恢复
- HBase系统为每个Region服务器配置了一个HLog文件，它是一种预写式日志（Write Ahead Log）
- 用户更新数据必须首先写入日志后，才能写入MemStore缓存，并且，直到MemStore缓存内容对应的日志已经写入磁盘，该缓存内容才能被刷写到磁盘

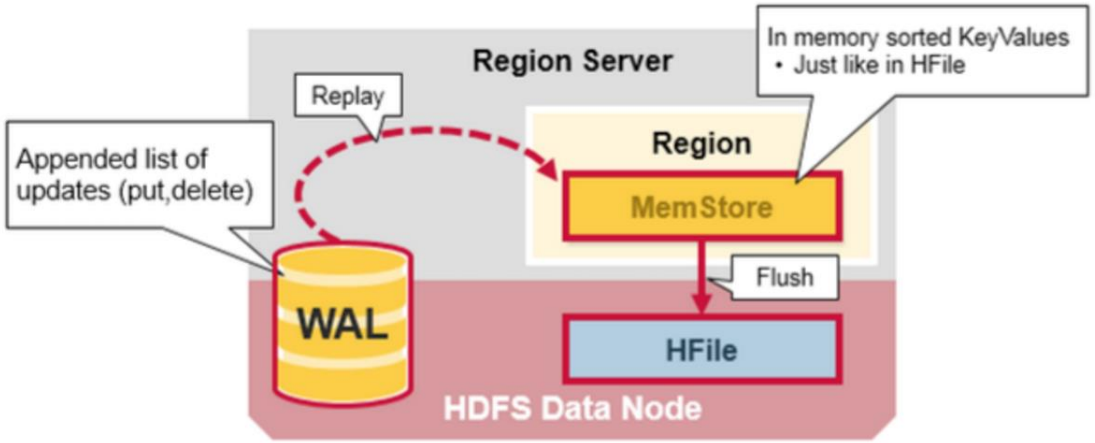
HBase的异常恢复（Crash Recovery）

- Zookeeper会实时监测每个Region服务器的状态，当某个Region服务器发生故障时，Zookeeper会通知Master
- Master首先会处理该故障Region服务器上面遗留的HLog文件，这个遗留的HLog文件中包含了来自多个Region对象的日志记录
- 系统会根据每条日志记录所属的Region对象对HLog数据进行拆分，分别放到相应Region对象的目录下，然后，再将失效的Region重新分配到可用的Region服务器中，并把与该Region对象相关的HLog日志记录也发送给相应的Region服务器

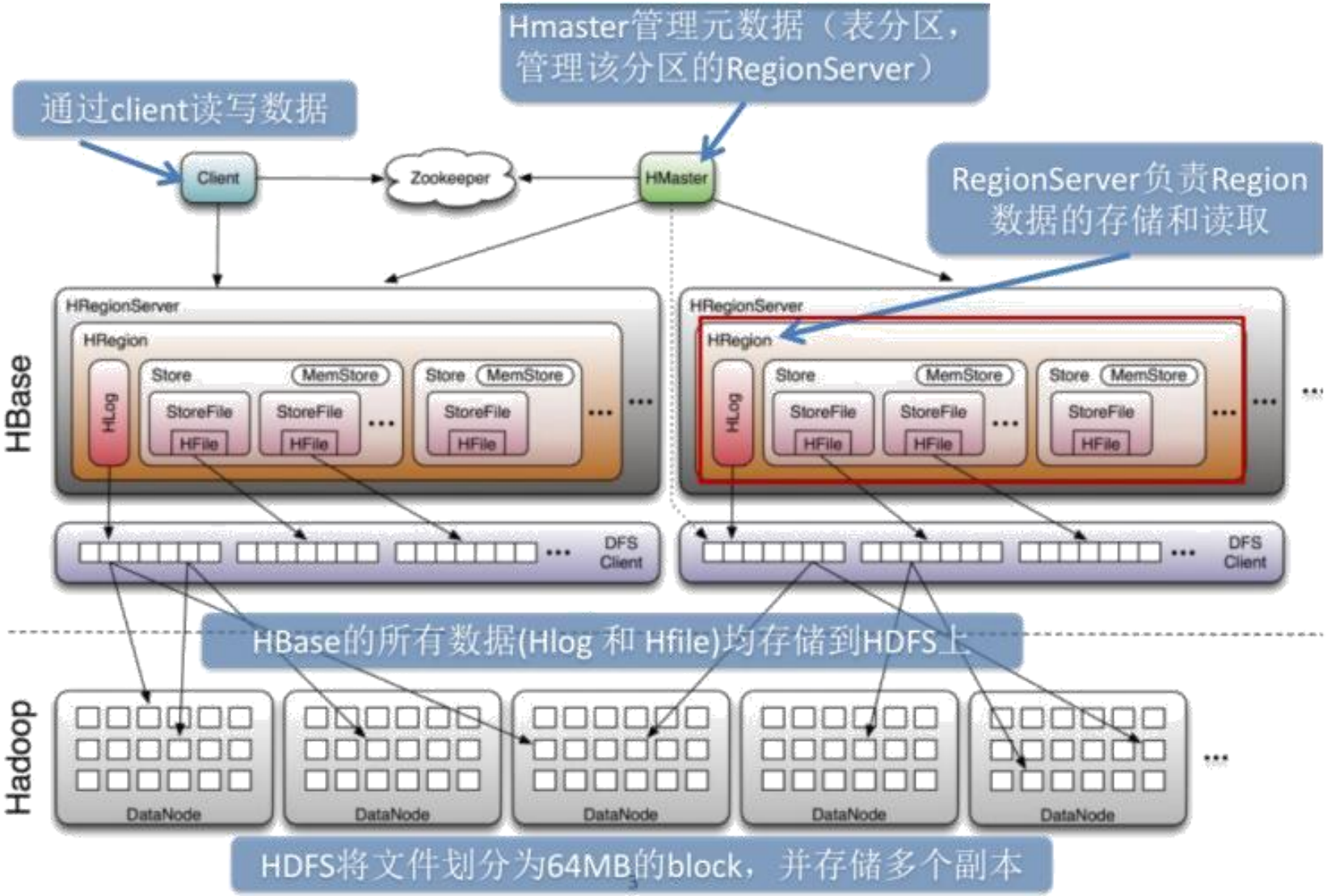


HBase的异常恢复（Crash Recovery）

- Region服务器领取到分配给自己的Region对象以及与之相关的HLog日志记录以后，会重新做一遍日志记录中的各种操作，把日志记录中的数据写入到MemStore缓存中，然后，刷新到磁盘的StoreFile文件中，完成数据恢复
- 共用日志优点：提高对表的写操作性能；缺点：恢复时需要分拆日志



HBase系统架构


Hmaster管理元数据（表分区，管理该分区的RegionServer）
通过client读写数据
Client
Zookeeper
HMaster
RegionServer负责Region数据的存储和读取
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
HBase的所有数据(Hlog 和 Hfile)均存储到HDFS上
Hadoop
DataNode
HDFS将文件划分为64MB的block，并存储多个副本

HBase实际应用中的性能优化方法

- 行键优化：行键是按照**字典序**存储，因此，设计行键时，要充分利用这个排序特点，将经常一起读取的数据存储到一起，将最近可能会被访问的数据放在一起。
- 举个例子：如果最近写入HBase表中的数据是最可能被访问的，可以考虑将时间戳作为行键的一部分，由于是字典序排序，所以可以使用Long.MAX_VALUE - timestamp作为行键，这样能保证新写入的数据在读取时可以被快速命中。

HBase实际应用中的性能优化方法

- 缓存策略（setCaching）：创建表的时候，可以通过`HColumnDescriptor.setInMemory(true)`将列族放到RegionServer的缓存中，保证在读取的时候被cache命中。
- 设置最大版本数：创建表的时候，可以通过`HColumnDescriptor.setMaxVersions(int maxVersions)`为每个列族设置表中数据的最大版本数，如果只需要保存最新版本的数据，那么可以设置 setMaxVersions(1)。
- 设置存储生命期：创建表的时候，可以通过`HColumnDescriptor.setTimeToLive(int timeToLive)`设置数据的存储生命期，过期数据将自动被删除，例如如果只需要存储最近两天的数据，那么可以设置setTimeToLive(2 * 24 * 60 * 60)。
- https://hbase.apache.org/apidocs/

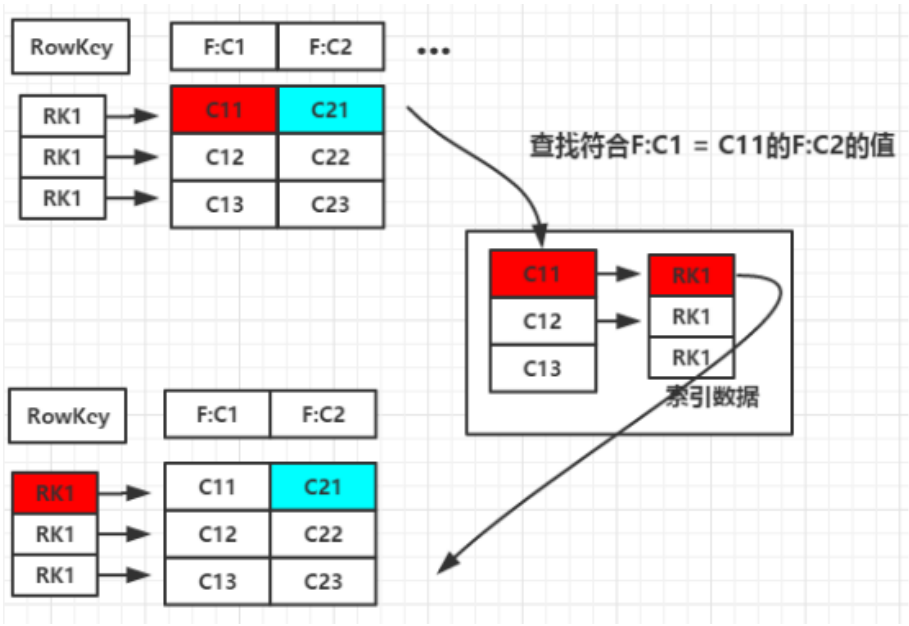
在HBase之上构建SQL引擎

- Hive整合Hbase：Hive与HBase的整合功能从Hive0.6.0版本已经开始出现，利用两者对外的API接口互相通信，通信主要依靠hive-hbase-handler.jar工具包。由于HBase有一次比较大的版本变动，所以并不是每个版本的Hive都能和现有的HBase版本进行整合，所以在使用过程中特别注意的就是两者版本的一致性。
- Phoenix：Phoenix由Salesforce.com开源，是构建在Apache HBase之上的一个SQL中间层，可以让开发者在HBase上执行SQL查询。

构建HBase二级索引

- 访问HBase表中的行，只有三种方式：
  - 通过单个行健访问
  - 通过一个行健的区间来访问
  - 全表扫描
- HBase默认只支持对行键的索引，如果需要针对其它的列来进行查询，就只能全表扫描了
- 二级索引设计思路：

  建立各列值与行键之间的映射关系

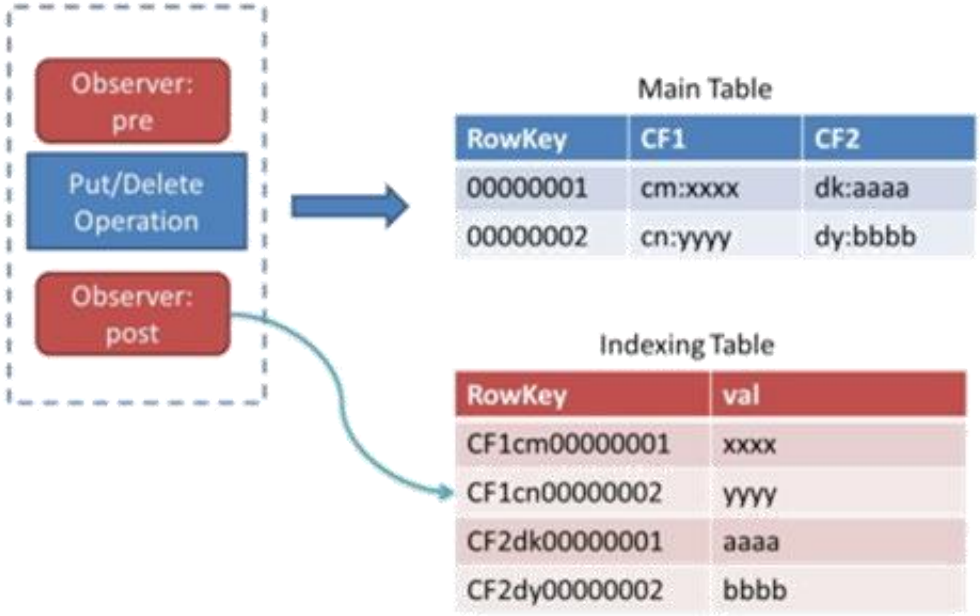


构建HBase二级索引

- Coprocessor构建二级索引：Coprocessor提供了两个实现：endpoint和observer，endpoint相当于关系型数据库的存储过程（Stored Procedure），而observer则相当于触发器（Trigger）
- observer允许我们在记录put前后做一些处理，因此，而我们可以在插入数据时同步写入索引表

优点：
非侵入性：引擎构建在HBase之上，既没有对HBase进行任何改动，也不需要上层应用做任何妥协

•缺点：每插入一条数据需要向索引表插入数据，即耗时是双倍的，对HBase的集群的压力也是双倍的



HBase编程实践

- HBase安装
  - 下载安装包
  - 解压安装包至路径 /usr/local
  - 配置环境变量,将hbase下的bin目录添加到PATH中
- HBase配置 http://dblab.xmu.edu.cn/blog/install-hbase/
  - HBase有三种运行模式，单机模式、伪分布式模式、分布式模式
  - 配置hbase-env.sh hbase-site.xml
  - 启动Hadoop—>启动HBase—>关闭HBase—>关闭Hadoop
  - start-hbase.sh, stop-hbase.sh
  - hbase shell
  - Web UI: http://[MasterIP]:16010/master-status

关于版本

- ✅ = Tested to be fully-functional
- ❌ = Known to not be fully-functional
- ⚠ = Not tested, may/may-not function

Hadoop version support matrix

	HBase-1.2.x, HBase-1.3.x	HBase-1.4.x	HBase-2.0.x	HBase-2.1.x
Hadoop-2.4.x	✅	❌	❌	❌
Hadoop-2.5.x	✅	❌	❌	❌
Hadoop-2.6.0	❌	❌	❌	❌
Hadoop-2.6.1+	✅	❌	✅	❌
Hadoop-2.7.0	❌	❌	❌	❌
Hadoop-2.7.1+	✅	✅	✅	✅
Hadoop-2.8.[0-1]	❌	❌	❌	❌
Hadoop-2.8.2	⚠	⚠	⚠	⚠
Hadoop-2.8.3+	⚠	⚠	✅	✅
Hadoop-2.9.0	❌	❌	❌	❌
Hadoop-2.9.1+	⚠	⚠	⚠	⚠
Hadoop-3.0.[0-2]	❌	❌	❌	❌
Hadoop-3.0.3+	❌	❌	✅	✅
Hadoop-3.1.0	❌	❌	❌	❌
Hadoop-3.1.1+	❌	❌	✅	✅

HBase常用Shell命令

- create: 创建表
- list: 列出HBase中所有的表信息
- 例子1：创建一个表，该表名称为tempTable，包含3个列族f1，f2和f3

```
hbase(main):002:0> create 'tempTable', 'f1', 'f2', 'f3'
0 row(s) in 1.3560 seconds

hbase(main):003:0> list
TABLE
tempTable
testTable
wordcount
3 row(s) in 0.0350 seconds
```

HBase常用Shell命令

- put：向表、行、列指定的单元格添加数据
- scan：浏览表的相关信息
- 例子2：继续向表tempTable中的第r1行、第“f1:c1”列，添加数据值为“hello,dblab”

```
hbase(main):005:0> put 'tempTable', 'r1', 'f1:c1', 'hello, dblab'
0 row(s) in 0.0240 seconds

hbase(main):006:0> scan 'tempTable'
ROW                           COLUMN+CELL
 r1                           column=f1:c1, timestamp=1430036599391, value=hello, dblab
1 row(s) in 0.0160 seconds
```

在添加数据时，HBase会自动为添加的数据添加一个时间戳，
也可以在添加数据时人工指定时间戳的值

HBase常用Shell命令

- get：通过表名、行、列、时间戳、时间范围和版本号来获得相应单元格的值
- 例子3：
  - 从tempTable中，获取第r1行、第“f1:c1”列的值
  - 从tempTable中，获取第r1行、第“f1:c3”列的值

```
hbase(main):012:0> get 'tempTable', 'r1', {COLUMN=>'f1:c1'}
COLUMN                          CELL
 f1:c1                          timestamp=1430036599391, value=hello, dblab
1 row(s) in 0.0090 seconds

hbase(main):013:0> get 'tempTable', 'r1', {COLUMN=>'f1:c3'}
COLUMN                          CELL
0 row(s) in 0.0030 seconds
```

从运行结果可以看出： tempTable中第r1行、第“f1:c3”列的值当前不存在

HBase常用Shell命令

- enable/disable：使表有效或无效
- drop：删除表
- 例子4：使表tempTable无效、删除该表

```
hbase(main):016:0> disable 'tempTable'
0 row(s) in 1.3720 seconds

hbase(main):017:0> drop 'tempTable'
0 row(s) in 1.1350 seconds

hbase(main):018:0> list
TABLE
testTable
wordcount
2 row(s) in 0.0370 seconds
```

HBase常用Java API及应用实例

- 任务要求：创建表、插入数据、浏览数据
- 创建一个学生信息表，用来存储学生姓名（姓名作为行键，并且假设姓名不会重复）以及考试成绩，其中，考试成绩是一个列族，分别存储了各个科目的考试成绩。

学生信息表的表结构

name	score		
	English	Math	Computer

需要添加的数据

name	score		
	English	Math	Computer
Alice	69	86	77
Bob	55	100	88

HBase常用Java API及应用实例

Configuration

对配置信息管理的一个类

Connection

对连接进行管理的一个类

Admin

对数据库进行管理的一个类
用于管理对表的创建删除等

```java
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;
import java.io.IOException;

public class HBaseExample {
    public static Configuration configuration;
    public static Connection connection;
    public static Admin admin;

    public static void main(String[] args)throws IOException{
        init();
        createTable("student", new String[]{"score"});
        insertData("student","Alice","score","English","69");
        insertData("student","Alice","score","Math","86");
        insertData("student","Alice","score","Computer","77");
        insertData("student","Bob","score","English","55");
        insertData("student","Bob","score","Math","100");
        insertData("student","Bob","score","Computer","88");
        getData("student", "Alice", "score", "English");
        close();
    }

    ……
    public static void init(){……}//建立连接
    public static void close(){……}//关闭连接
    public static void createTable(){……}//创建表
    public static void insertData() {……}//插入数据
    public static void getData(){……}//浏览数据
}
```

HBase常用Java API及应用实例

```java
//建立连接
public static void init(){
    configuration  = HBaseConfiguration.create();
    configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");
    try{
        connection = ConnectionFactory.createConnection(configuration);
        admin = connection.getAdmin();
    }catch (IOException e){
        e.printStackTrace();
    }
}
//关闭连接
public static void close(){
    try{
        if(admin != null){
            admin.close();
        }
        if(null != connection){
            connection.close();
        }
    }catch (IOException e){
        e.printStackTrace();
    }
}
```

```
<configuration>
<property>
<name>hbase.rootdir</name>
<value>hdfs://localhost:9000/hbase</value>
</property>
</configuration>
```

hbase-site.xml

HBase常用Java API及应用实例

在运行程序时，需要指定参数myTableName为“student”，colFamily为“{“score”}”
程序的运行效果与如下HBase Shell命令等效：
create ‘student’, ‘score’

```
50      /*创建表*/
51      /**
52       * @param myTableName 表名
53       * @param colFamily列族数组
54       * @throws Exception
55       */
56      public static void createTable(String myTableName,String[] colFamily) throws IOException {
57
58          TableName tableName = TableName.valueOf(myTableName);
59          if(admin.tableExists(tableName)){
60              System.out.println("table exists!");
61          }else {
62              HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
63              for(String str: colFamily){
64                  HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(str);
65                  hTableDescriptor.addFamily(hColumnDescriptor);
66              }
67              admin.createTable(hTableDescriptor);
68              System.out.println("create table success");
69          }
70  }
```

HBase常用Java API及应用实例

设置相应参数来添加数据，如：
insertData("student",“Alice","score","English","69");
上述代码与如下HBase Shell命令等效：
put 'student', ’Alice’, ’score:English’, ’69’;

```
72      /*添加数据*/
73      /**
74          * @param tableName 表名
75          * @param rowKey 行键
76          * @param colFamily 列族
77          * @param col 列限定符
78          * @param val 数据
79          * @throws Exception
80          */
81          public static void insertData(String tableName, String rowKey,
              String colFamily, String col, String val) throws IOException {

82              Table table = connection.getTable(TableName.valueOf(tableName));
83              Put put = new Put(Bytes.toBytes(rowKey));
84              put.addColumn(Bytes.toBytes(colFamily), Bytes.toBytes(col),
                  Bytes.toBytes(val));
85              table.put(put);
86              table.close();
87      }
```

HBase常用Java API及应用实例

设置相应参数来读取数据，如：
getData("student", “Alice", "score", "English");
上述代码与如下HBase Shell命令等效：
get ‘student’, ‘Alice’, {COLUMN=>'score:English'}

```
89        /*获取某单元格数据*/
90        /**
91         * @param tableName 表名
92         * @param rowKey 行键
93         * @param colFamily 列族
94         * @param col 列限定符
95         * @throws IOException
96         */
97      public static void getData(String tableName,String rowKey,String colFamily,String col)throws  IOException{
98            Table table = connection.getTable(TableName.valueOf(tableName));
99            Get get = new Get(Bytes.toBytes(rowKey));
100           get.addColumn(Bytes.toBytes(colFamily),Bytes.toBytes(col));
101           Result result = table.get(get);
102           showCell(result);
103           table.close();
104       }
105
106       /**
107       * 格式化输出
108       * @param result
109       */
110     public static void showCell(Result result){
111         Cell[] cells = result.rawCells();
112         for(Cell cell:cells){
113             System.out.println("Row:"+new String(CellUtil.cloneRow(cell)));
114             System.out.println("Timestamp:"+cell.getTimestamp());
115             System.out.println("Column Family:"+new String(CellUtil.cloneFamily(cell)));
116             System.out.println("Column Qualifier:"+new String(CellUtil.cloneQualifier(cell)));
117             System.out.println("Value:"+new String(CellUtil.cloneValue(cell)));
118         }
119     }
120
```

小结

- 本章详细介绍了HBase数据库的知识。HBase数据库是BigTable的开源实现，和BigTable一样，支持大规模海量数据，分布式并发数据处理效率极高，易于扩展且支持动态伸缩，适用于廉价设备
- HBase可以支持Native Java API、HBase Shell、Thrift Gateway、REST Gateway、Pig、Hive等多种访问接口，可以根据具体应用场合选择相应访问方式
- HBase实际上就是一个稀疏、多维、持久化存储的映射表，它采用行键、列键和时间戳进行索引，每个值都是未经解释的字符串。本章介绍了HBase数据在概念视图和物理视图中的差别
- HBase采用分区存储，一个大的表会被分拆许多个Region，这些Region会被分发到不同的服务器上实现分布式存储
- HBase的系统架构包括客户端、Zookeeper服务器、Master主服务器、Region服务器。客户端包含访问HBase的接口；Zookeeper服务器负责提供稳定可靠的协同服务；Master主服务器主要负责表和Region的管理工作；Region服务器负责维护分配给自己的Region，并响应用户的读写请求
- 本章最后详细介绍了HBase运行机制和编程实践的知识