

《大数据基础》

第4章 分布式数据库HBase

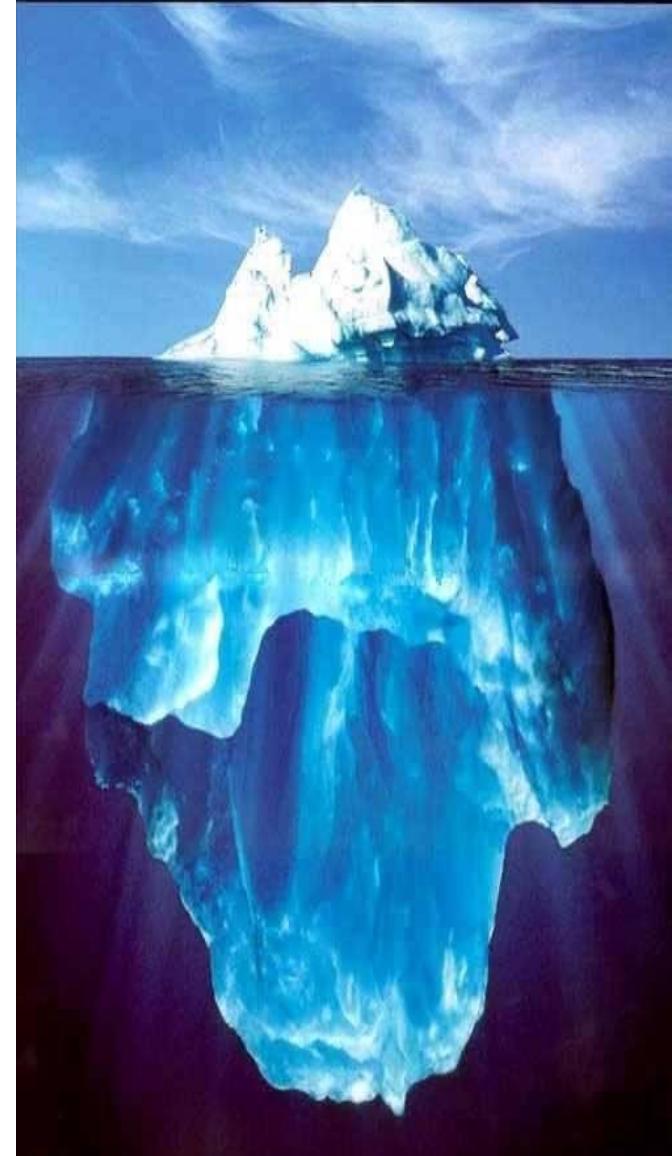
洪韬

E-mail: 2993400893@qq.com



提纲

- 4.1 概述
- 4.2 HBase访问接口
- 4.3 HBase数据模型
- 4.4 HBase的实现原理
- 4.5 HBase运行机制
- 4.6 HBase应用方案
- 4.7 HBase编程实践





4.1 概述

- 4.1.1 从BigTable说起
- 4.1.2 HBase简介
- 4.1.3 HBase与传统关系数据库的对比分析



4.1.1 从BigTable说起

BigTable是一个分布式存储系统

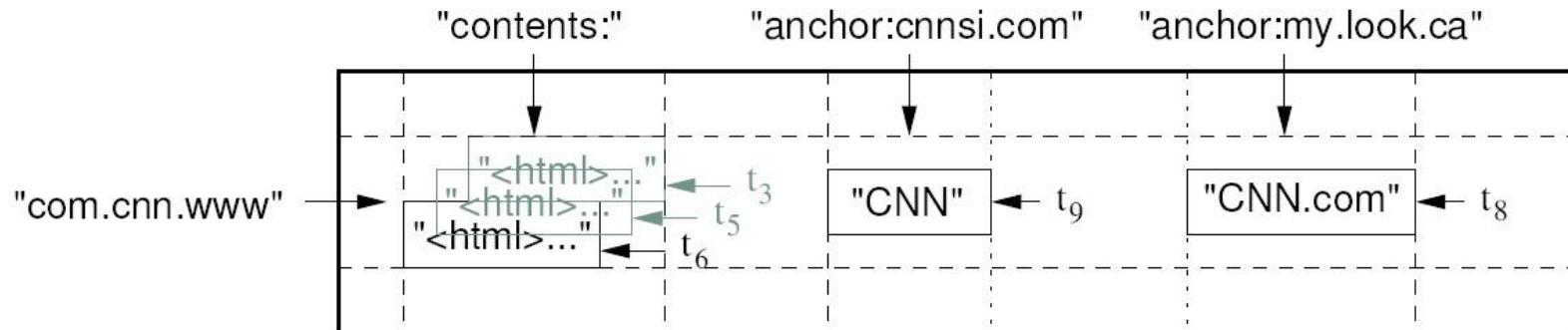
BigTable起初用于解决典型的互联网搜索问题

- 建立互联网索引

- 1 爬虫持续不断地抓取新页面，这些页面每页一行地存储到BigTable里
- 2 MapReduce计算作业运行在整张表上，生成索引，为网络搜索应用做准备

- 搜索互联网

- 3 用户发起网络搜索请求
- 4 网络搜索应用查询建立好的索引，从BigTable得到网页
- 5 搜索结果提交给用户



网页在BigTable中的存储样例



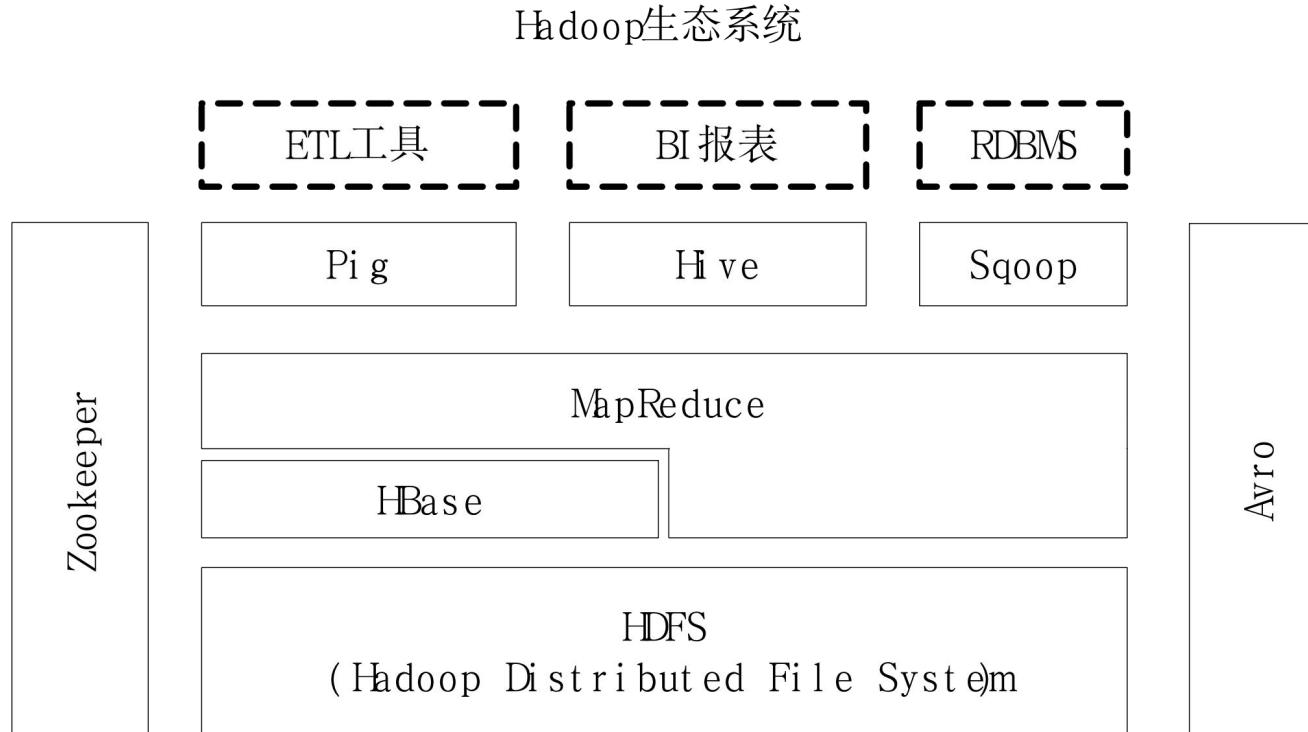
4.1.1从BigTable说起

- BigTable是一个分布式存储系统
- 利用谷歌提出的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
- 采用Chubby提供协同服务管理
- 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
- 谷歌的许多项目都存储在BigTable中，包括搜索、地图、财经、打印、社交网站Orkut、视频共享网站YouTube和博客网站Blogger等



4.1.2 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





4.1.2 HBase简介

	BigTable	HBase
文件存储系统	GFS	HDFS
海量数据处理	MapReduce	Hadoop MapReduce
协同服务管理	Chubby	Zookeeper



4.1.2 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
- 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（分库分表也不能很好解决）
- 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
- 因此，业界出现了一类面向半结构化数据存储和处理的高可扩展、低写入/查询延迟的系统，例如，键值数据库、文档数据库和列族数据库（如BigTable和HBase等）
- HBase已经成功应用于互联网服务领域和传统行业的众多在线式数据分析处理系统中



4.1.3 HBase与传统关系数据库的对比分析

- HBase与传统的关系数据库的区别主要体现在以下几个方面：
- (1) **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则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
- (2) **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
- (3) **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，每个列族都由几个文件保存，不同列族的文件是分离的



4.1.3 HBase与传统关系数据库的对比分析

- HBase与传统的关系数据库的区别主要体现在以下几个方面：
- (4) **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——行键，通过巧妙的设计，HBase中的所有访问方法，或者通过行键访问，或者通过行键扫描，从而使得整个系统不会慢下来
- (5) **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
- (6) **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。相反，HBase和BigTable这些分布式数据库就是为了实现灵活的水平扩展而开发的，能够轻易地通过在集群中增加或者减少硬件数量来实现性能的伸缩



4.2 HBase访问接口

表4-2 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	使用Pig Latin流式编程语言来处理HBase中的数据	适合做数据统计
Hive	简单	当需要以类似SQL语言方式来访问HBase的时候



4.3 HBase数据模型

- 4.3.1 数据模型概述
- 4.3.2 数据模型相关概念
- 4.3.3 数据坐标
- 4.3.4 概念视图
- 4.3.5 物理视图
- 4.3.6 面向列的存储



4.3.1 数据模型概述

- **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（这是和**HDFS**只允许追加不允许修改的特性相关的）



4.3.2 数据模型相关概念

- 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

	Info		
	name	major	email
201505001	Luo Min	Math	luo@qq.com
201505002	Liu Jun	Math	liu@qq.com
201505003	Xie You	Math	xie@aa.com you@163.com

Diagram annotations:

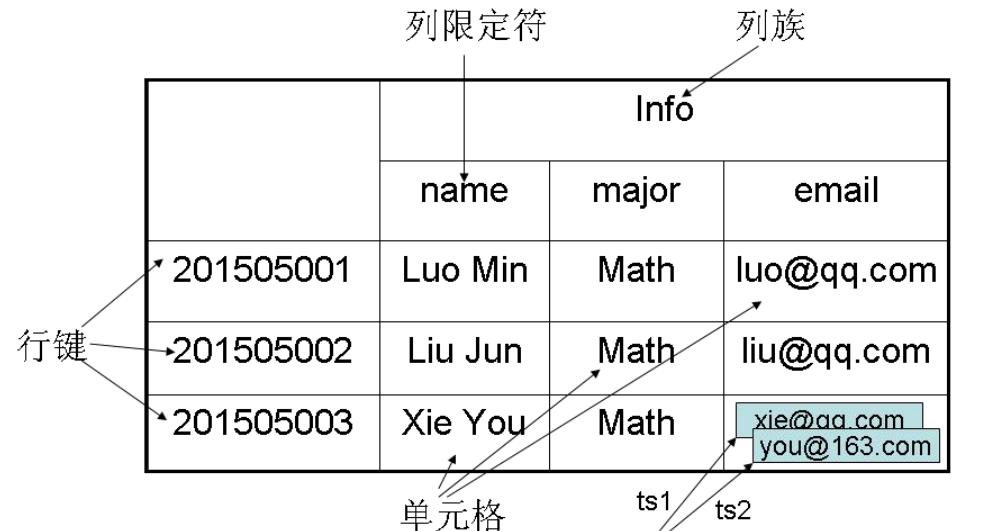
- 列限定符 (Column Qualifier) points to the column headers "name", "major", and "email".
- 列族 (Column Family) points to the header "Info".
- 行键 (Row Key) points to the row identifiers "201505001", "201505002", and "201505003".
- 单元格 (Cell) points to the individual data entries like "Luo Min" and "Math".
- ts1 and ts2 point to the two time stamps associated with the cell "Xie You" in the third row.
- A note states: "该单元格有2个时间戳ts1和ts2" (This cell has 2 time stamps, ts1 and ts2).
- Another note states: "每个时间戳对应一个数据版本" (Each time stamp corresponds to a data version).
- Timestamp values: ts1=1174184619081 ts2=1174184620720



4.3.3 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"



ts1=1174184619081 ts2=1174184620720



4.3.4概念视图

表4-4 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>..."	



4.3.5物理视图

表4-5 HBase数据的物理视图
列族contents

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

列族anchor

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



4.3.6面向列的存储

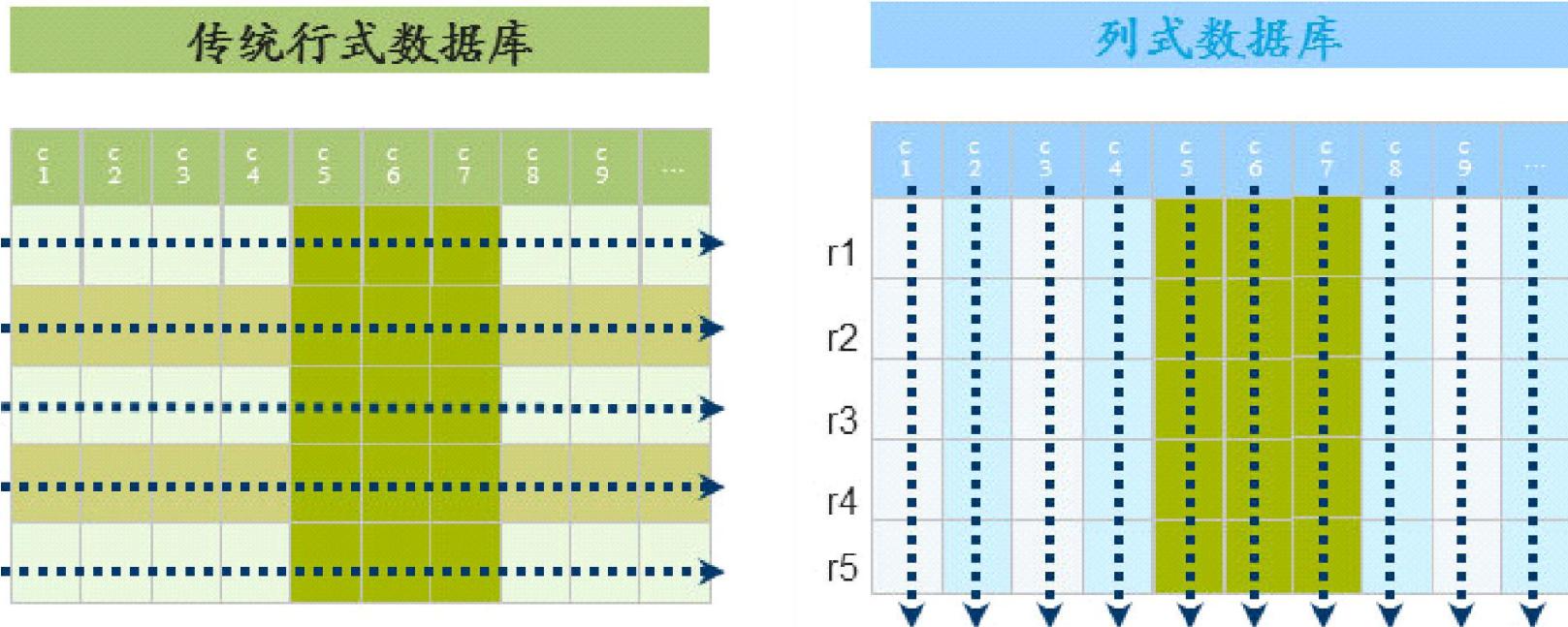


图4-3 行式数据库和列式数据库示意图



4.3.6面向列的存储

SQL模式

Log					
Log_id	user	age	sex	ip	action
1	Marry	34	F	55.237.104.36	Logout
2	Bob	18	M	122.158.130.90	New_tweet
3	Tom	38	M	93.24.237.12	Logout
4	Linda	58	F	87.124.79.252	Logout

图4-4 行式存储结构和列式存储结构



4.3.6面向列的存储

行式存储

行1

1	Marry	34	F	55.237.104.36	Logout
---	-------	----	---	---------------	--------

行2

2	Bob	18	M	122.158.130.90	New_tweet
---	-----	----	---	----------------	-----------

行3

3	Tom	38	M	93.24.237.12	Logout
---	-----	----	---	--------------	--------

.....



4.3.6面向列的存储

列式存储

列1: user	Marry	Bob	Tom	Li nda
列2: age	34	18	38	58
列3: sex	F	M	M	F
列4: i p	55. 237. 104. 36	122. 158. 130. 90	93. 24. 237. 12	87. 124. 79. 252
列5: action	Logout	New_tweet	Logout	Logout

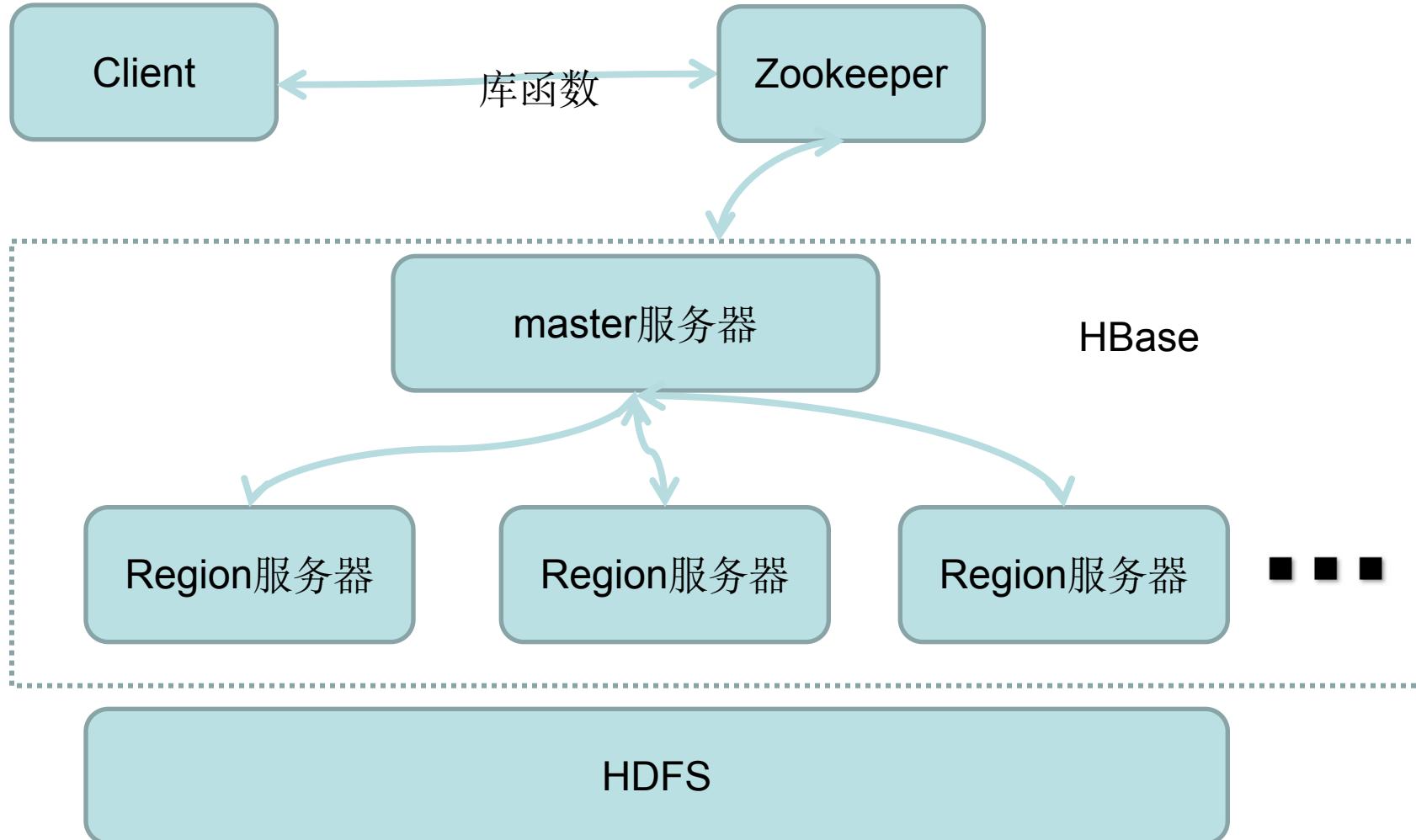


4.4 HBase的实现原理

- 4.4.1 HBase功能组件
- 4.4.2 表和Region
- 4.4.3 Region的定位



4.4.1 HBase功能组件





4.4.1 HBase功能组件

- HBase的实现包括三个主要的功能组件：
 - (1) 库函数：链接到每个客户端
 - (2) 一个Master主服务器
 - (3) 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**
- 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
- 客户端并不依赖Master，而是通过Zookeeper来获得Region位置信息，大多数客户端甚至从来不和Master通信，这种设计方式使得Master负载很小



4.4.2 表和Region

该单元格有2个时间戳ts1和ts2
每个时间戳对应一个数据版本
ts1=1174184619081 ts2=1174184620720

		Info		
		name	major	email
→	201505001	Luo Min	Math	luo@qq.com
→	201505002	Liu Jun	Math	liu@qq.com
→	201505003	Xie You	Math	xie@aa.com you@163.com

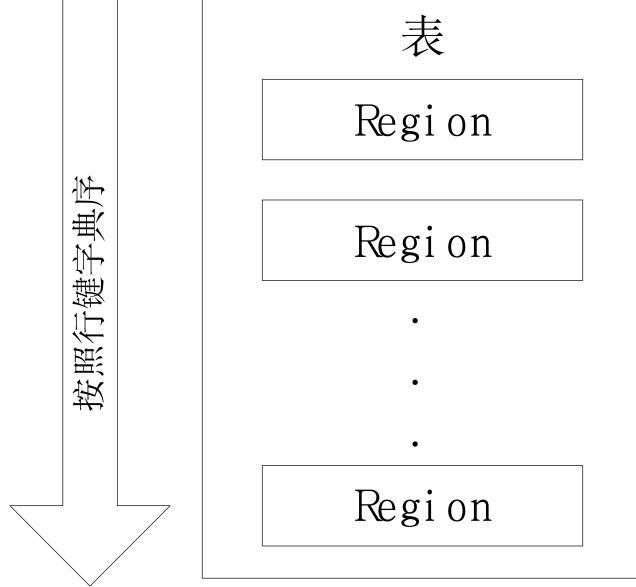


什么是Region呢？





4.4.2 表和Region



表太大，
在一台机器上存不下
，对表进行水平方向对截取，
截取对行区间，称为**region**

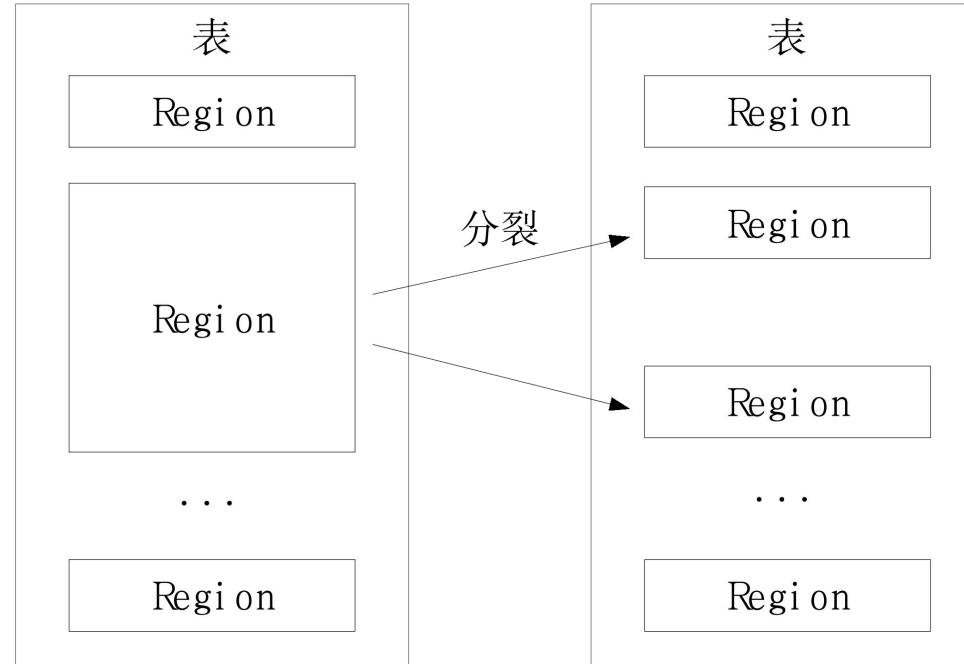


为什么说一台机器放
不下，需要截取，**HBase**是基于
HDFS的，以块为基础可以存放
任意大小的文件？



4.4.2 表和Region

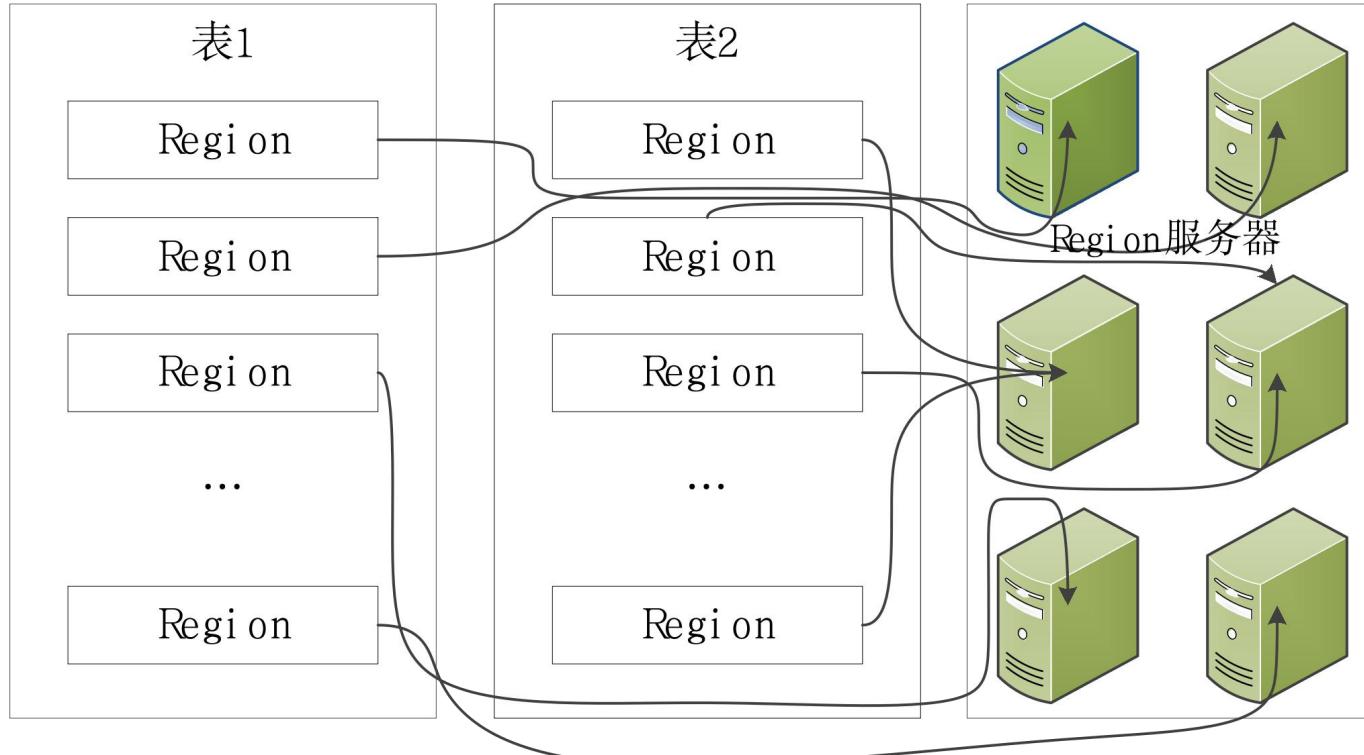
- 开始只有一个Region，随着数据的不断插入，Region不断增大，后来开始分裂
- Region拆分操作非常快，接近瞬间，因为拆分之后的Region读取的仍然是原存储文件，直到“合并”过程把存储文件异步地写到独立的文件之后，才会读取新文件





4.4.2 表和Region

- 每个Region默认大小是100MB到200MB (2010年以前的硬件配置)
 - 每个Region的最佳大小取决于单台服务器的有效处理能力
 - 目前每个Region最佳大小建议1GB-2GB (2013年以后的硬件配置)
- 同一个Region不会被分拆到多个Region服务器
- 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



4.4.2 表和Region

Region大小

数百兆

1GB

10GB

?

01

Region及
Region
Server设置

?

Server 中存储20-200个
Region最好

Region server存储
的Region数量

02



4.4.3 Region的定位

- Region 标识符
 - 表名 + 开始主键 + RegionID
- Region Server 标识符

Region 标识符	Region Server 标识符
Region 1	Region Server 1
Region 2	Region Server 1
Region 3	Region Server 2

- Region 和 Region Server 的映射关系表 -Meta表



4.4.3 Region的定位

- 元数据表，又名.META.表，存储了Region和Region服务器的映射关系
- 当HBase表很大时，.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，名字是在程序中被写死的
- Zookeeper文件记录了-ROOT-表的位置

ROOT-表 表关系表的表关系表

Meta Region标识	Meta Region Server
Meta Region 1	Meta Region Server 1
Meta Region 2	Meta Region Server 2
Meta Region 3	Meta Region Server 1

ROOT表-不可再分



4.4.3 Region的定位

HBase的三层结构中各层次的名称和作用

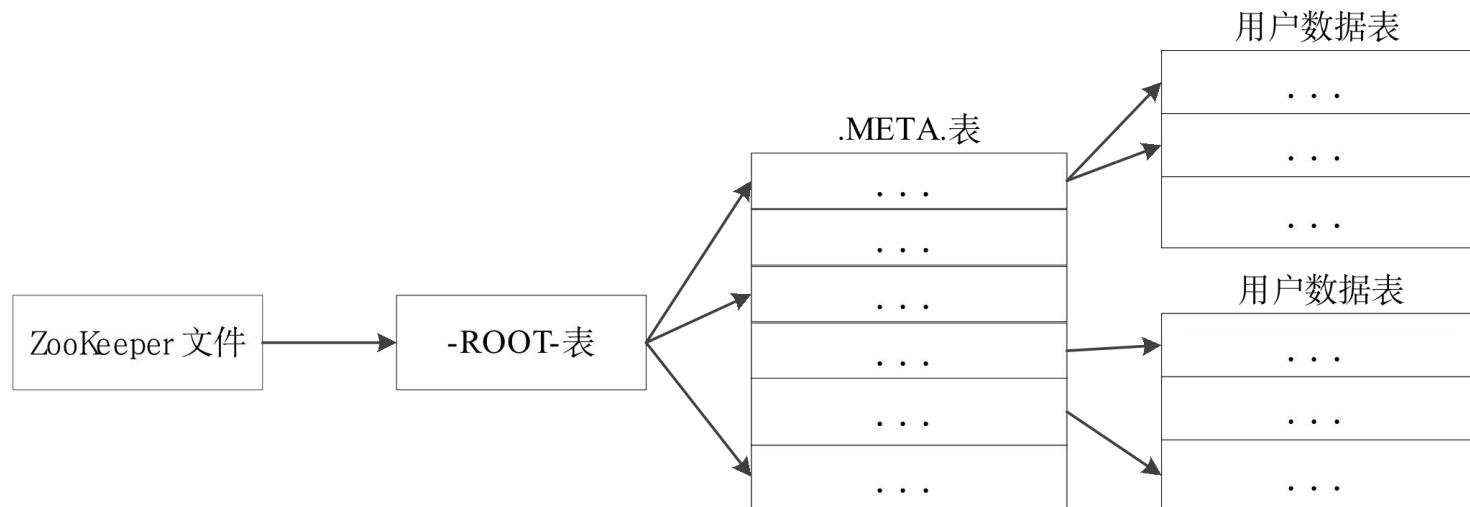
层次	名称	作用
第一层	Zookeeper文件	记录了-ROOT-表的位置信息
第二层	-ROOT-表	记录了.META.表的Region位置信息 -ROOT-表只能有一个Region。通过-ROOT-表，就可以访问.META.表中的数据
第三层	.META.表	记录了用户数据表的Region位置信息，.META.表可以有多个Region，保存了HBase中所有用户数据表的Region位置信息



4.4.3Region的定位

客户端访问数据时的“三级寻址”

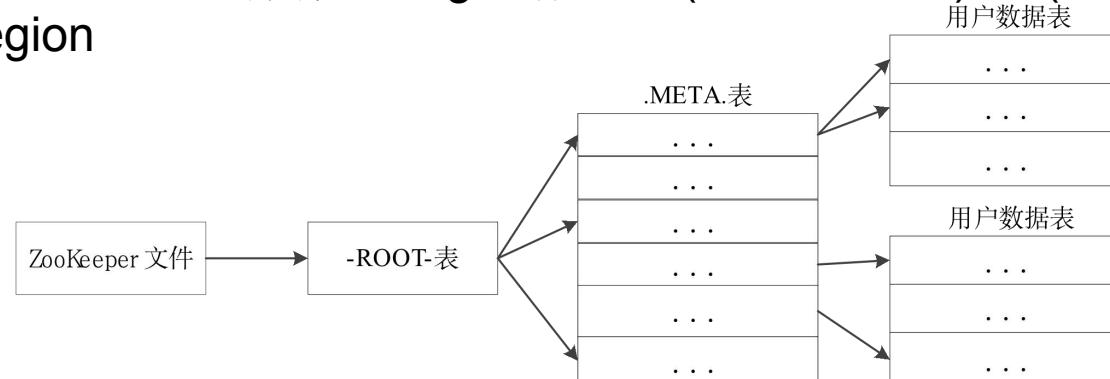
- 为了加速寻址，客户端会缓存位置信息，同时，需要解决**缓存失效**问题
- 寻址过程客户端只需要询问Zookeeper服务器，不需要连接Master服务器





4.4.3 Region的定位

- 为了加快访问速度，.META.表的全部Region都会被保存在内存中
- 假设.META.表的每行（一个映射条目）在内存中大约占用1KB，并且假设每个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数目的计算方法是：
 - (-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， $128MB/1KB=2^{17}$ 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 $(128MB/1KB) \times (128MB/1KB) = 2^{34}$ 个Region



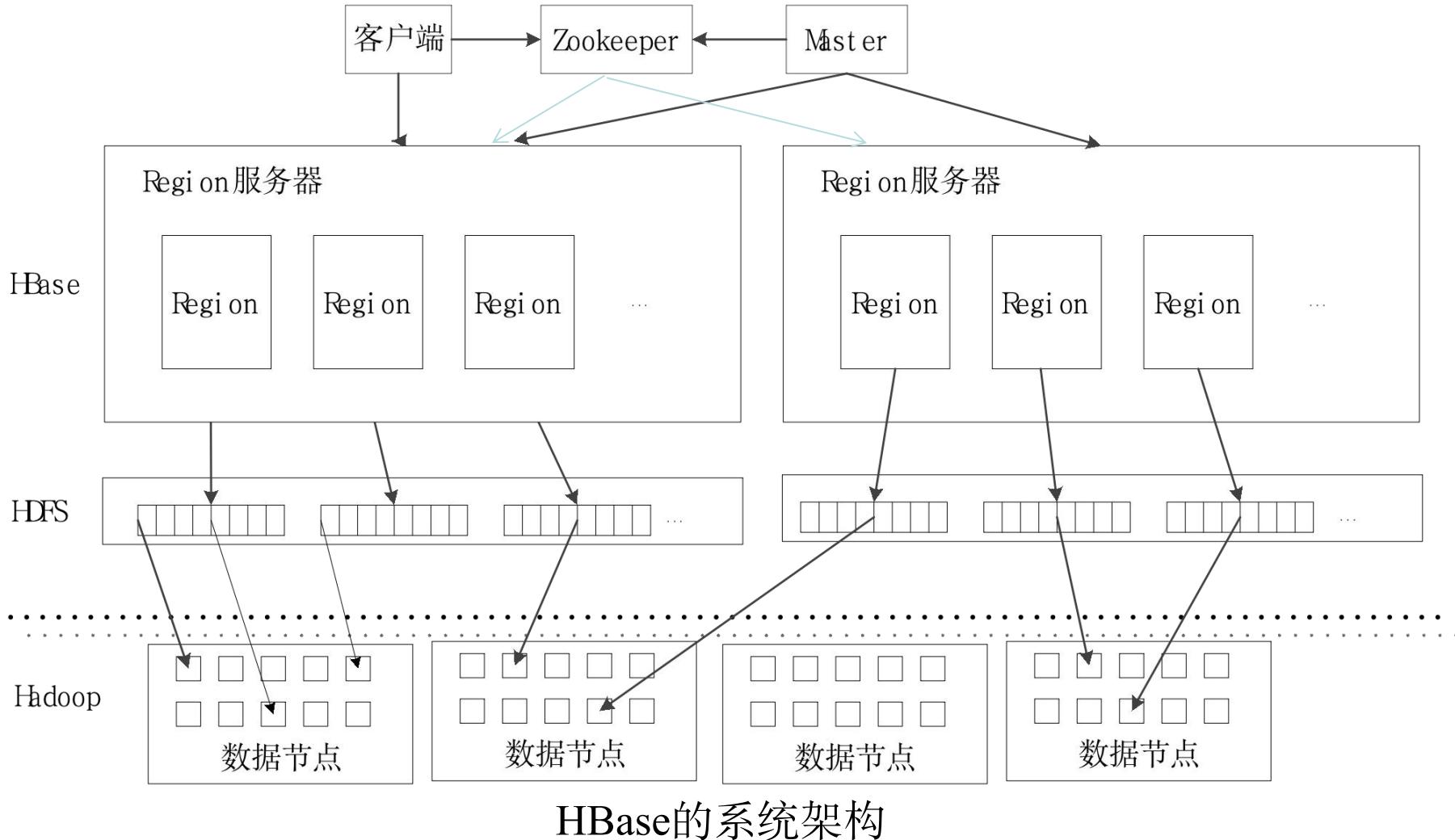


4.5 HBase运行机制

- 4.5.1 HBase系统架构
- 4.5.2 Region服务器工作原理
- 4.5.3 Store工作原理
- 4.5.4 HLog工作原理

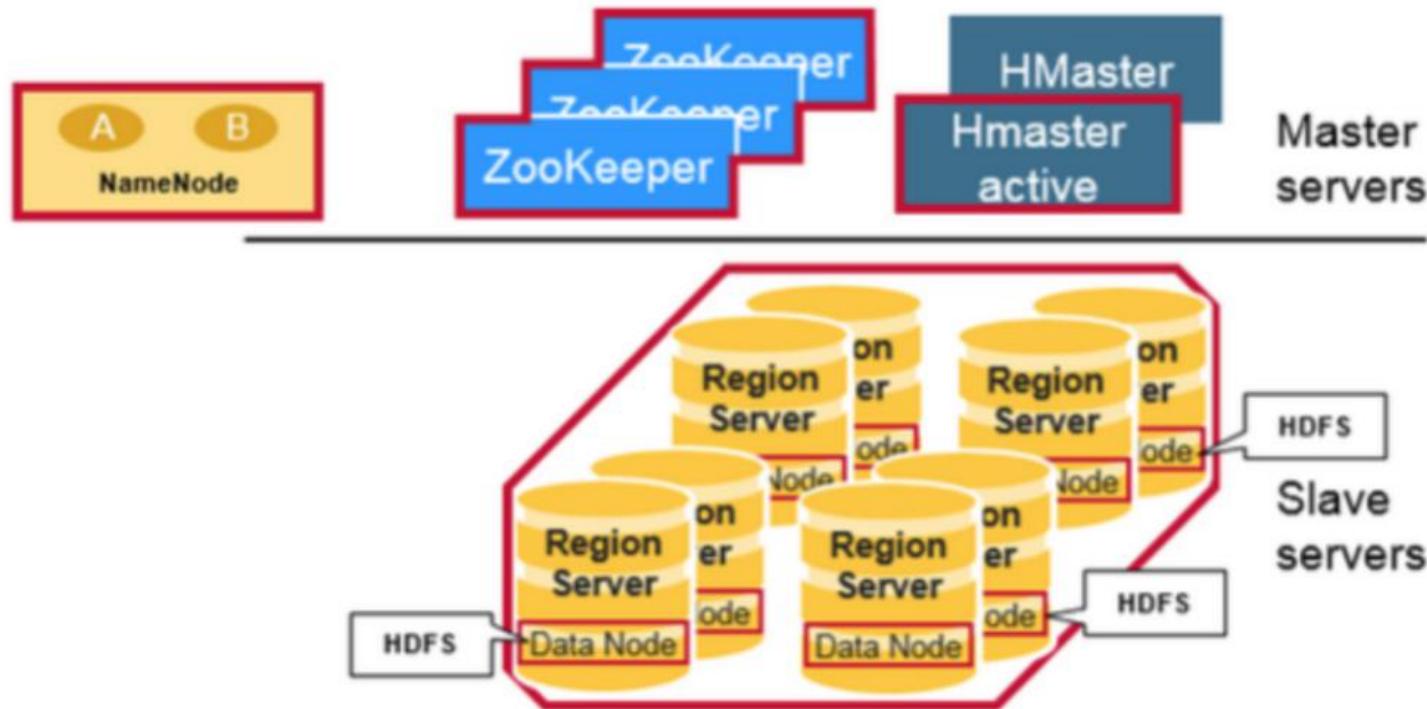


4.5.1 HBase系统架构





4.5.1 HBase系统架构



HBase的系统架构

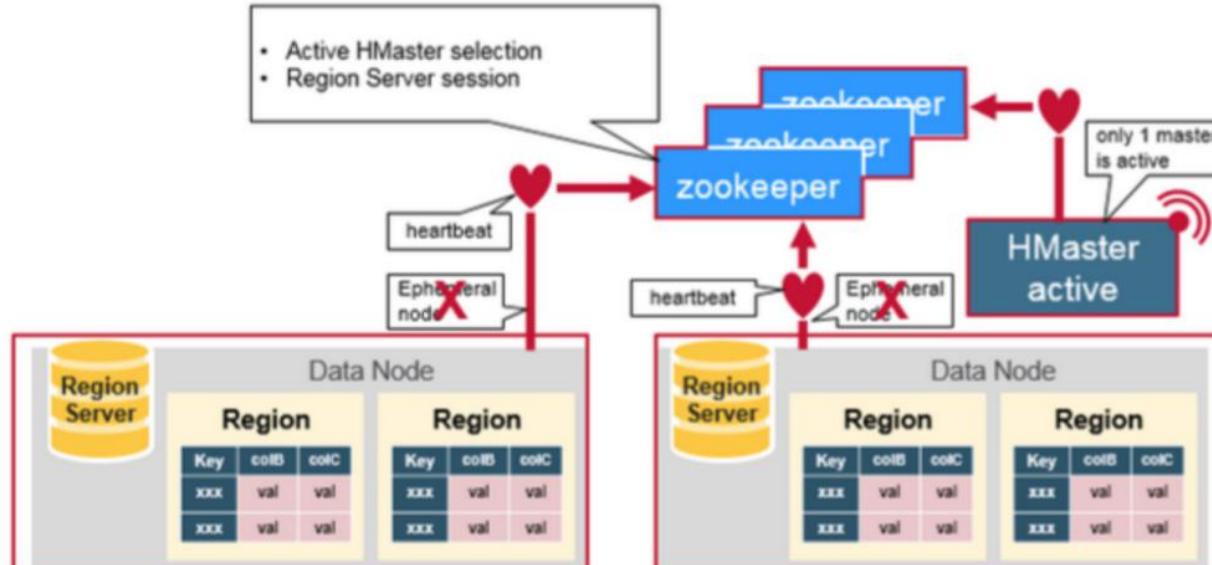


4.5.1 HBase系统架构

- 1. 客户端
 - 客户端包含访问HBase的接口，库函数，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
- 2. Zookeeper服务器

Zookeeper是一个很好的集群管理工具，被大量用于分布式计算，提供配置维护、域名服务、分布式同步、组服务等。

 - 帮助HMaster监听Region Server的状态。HBase启动后，HMaster、Region Servers在zookeeper中注册znode。



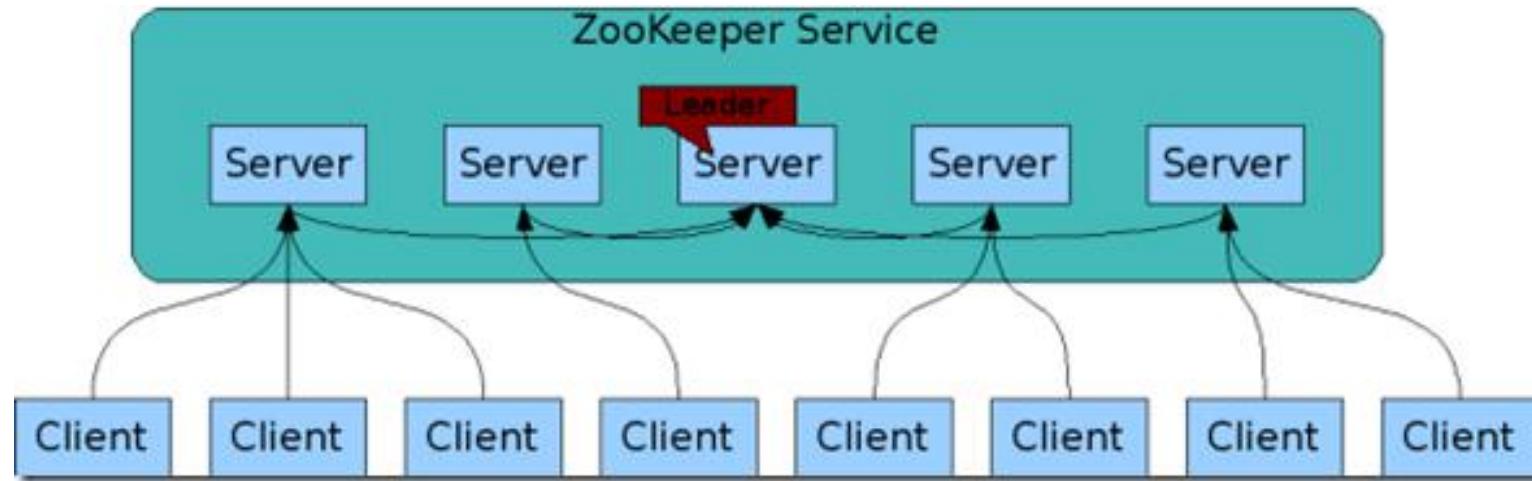
Zookeeper监听Server状态



4.5.1 HBase系统架构

2. Zookeeper服务器

- Zookeeper可以帮助选举出一个Master作为集群的总管（这是对于多Master节点的HBase，存在一些热备的HMaster），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





4.5.1 HBase系统架构

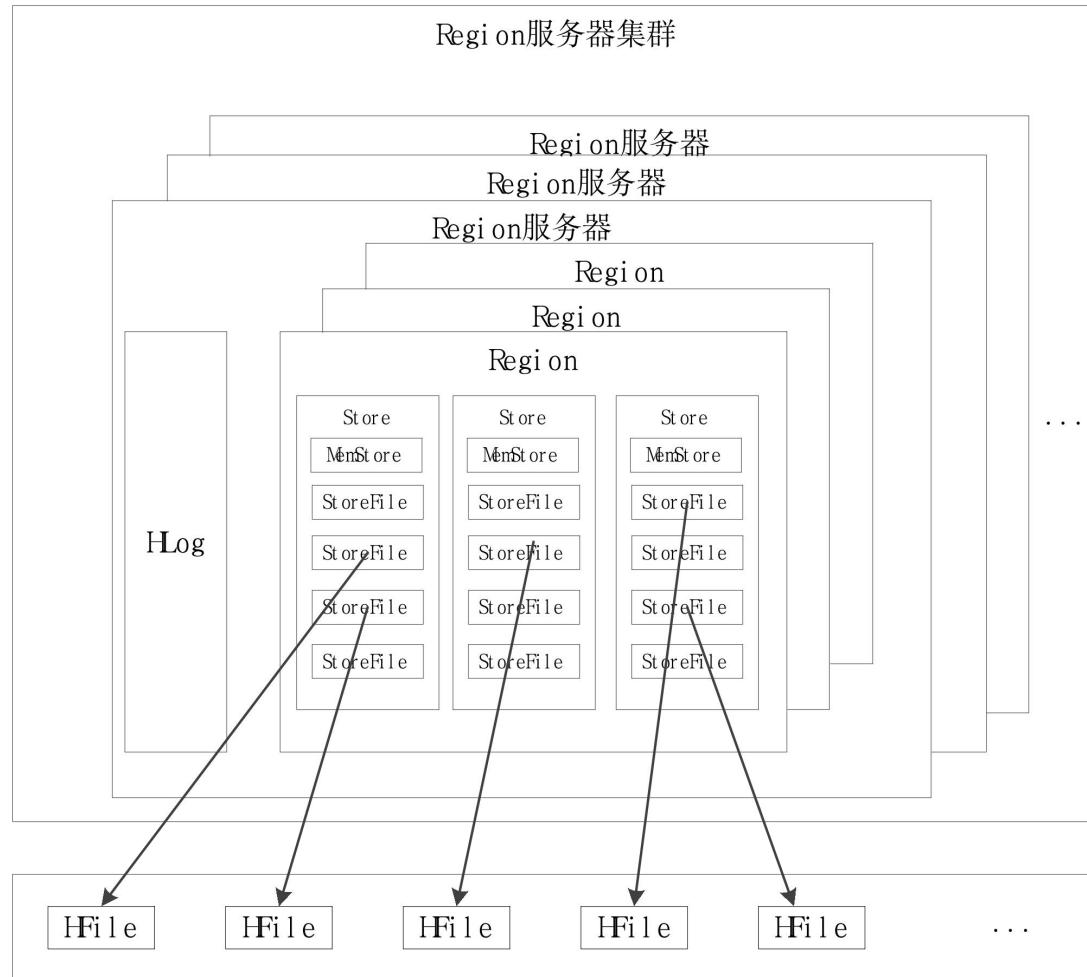
- 3. Master
- 主服务器Master主要负责表和Region的管理工作:
 - 管理用户对表的增加、删除、修改、查询等操作
 - 实现不同Region服务器之间的**负载均衡**
 - 在Region分裂或合并后，负责重新调整Region的分布
 - 对发生故障失效的Region服务器上的**Region**进行迁移

4. Region服务器

- Region服务器是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



4.5.2 Region服务器工作原理



Store:

一个**Store**对应着表中一个列族的存储

MemStore:

MemStore是内存中的缓存，保存最近更新的数据

StoreFile:

磁盘中的文件，**B**树的结构，方便快速读取

HLog:

预写式的日志，记录用户更新数据等操作

HFile:

StoreFile在**HDFS**的底层实现

Region服务器向HDFS文件系统中读写数据



4.5.2 Region服务器工作原理

1. 用户读写数据过程

- 用户写入数据时，被分配到相应Region服务器去执行
- 用户数据首先被写入到MemStore和Hlog中
- 只有当操作写入Hlog之后，commit()调用才会将其返回给客户端
- 当用户读取数据时，Region服务器会首先访问MemStore缓存，如果找不到，再去磁盘上面的StoreFile中寻找



4.5.2 Region服务器工作原理

2. 缓存的刷新

- 系统会周期性地把**MemStore**缓存里的内容刷写到磁盘的**StoreFile**文件中，清空缓存，并在**Hlog**里面写入一个标记
- 每次刷写都生成一个新的**StoreFile**文件，因此，每个**Store**包含多个**StoreFile**文件
- 每个**Region**服务器都有一个自己的**HLog**文件，每次启动都检查该文件，确认最近一次执行缓存刷新操作之后是否发生新的写入操作；如果发现更新，则先写入**MemStore**，再刷写到**StoreFile**，最后删除旧的**Hlog**文件，开始为用户提供服务



4.5.2 Region服务器工作原理

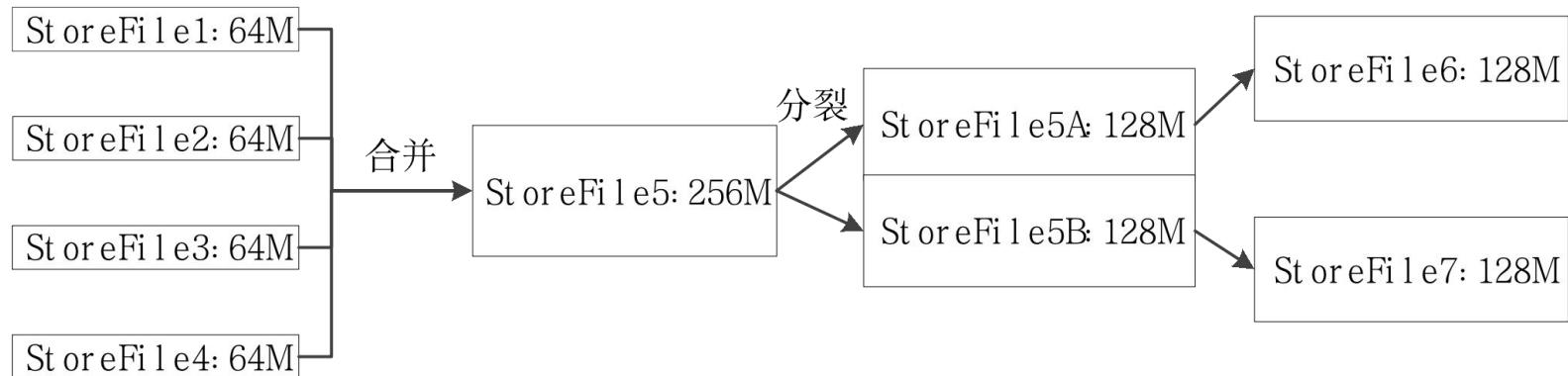
3. StoreFile的合并

- 每次刷写都生成一个新的StoreFile，数量太多，影响查找速度
- 调用Store.compact()把多个合并成一个
- 合并操作比较耗费资源，只有数量达到一个阈值才启动合并



4.5.3 Store工作原理

- Store是Region服务器的核心
- 多个StoreFile合并成一个
- 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



StoreFile的合并和分裂过程



4.5.4 HLog工作原理

- 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



4.5.4 HLog工作原理

- 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
- 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



4.6 HBase编程实践

4.6.1 HBase的安装与配置

4.6.1 HBase常用Shell命令

4.6.3 HBase常用Java API及应用实例



4.6.1 HBase的安装与配置

1. HBase安装

- 下载安装包 hbase-2.2.2-bin.tar.gz
- 解压安装包 hbase-2.2.2-bin.tar.gz 至路径 /usr/local
- 配置系统环境, 将 hbase 下的 bin 目录添加到系统的 path 中

备注: 第2章安装完Hadoop时, 只包含HDFS和MapReduce等核心组件, 并不包含HBase, 因此, HBase需要单独安装

2. HBase配置

HBase有三种运行模式, 单机模式、伪分布式模式、分布式模式。
以下先决条件很重要, 比如没有配置JAVA_HOME环境变量, 就会报错。

- JDK
- Hadoop(单机模式不需要, 伪分布式模式和分布式模式需要)
- SSH

启动关闭Hadoop和HBase的顺序一定是:

启动Hadoop—>启动HBase—>关闭HBase—>关闭Hadoop

HBASE_MANAGES_ZK=true, 则由HBase自己管理Zookeeper

否则, 启动独立的Zookeeper

建议: 单机版HBase, 使用自带Zookeeper; 集群安装HBase则采用单独Zookeeper集群



4.6.2 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
```

备注：后面的例子都在此基础上继续操作



4.6.2 HBase常用Shell命令

put: 向表、行、列指定的单元格添加数据

一次只能为一个表的一行数据的一个列添加一个数据

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，当然，也可以在添加数据时人工指定时间戳的值



4.6.2 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:

- (1) 从tempTable中，获取第r1行、第 “f1:c1” 列的值
- (2) 从tempTable中，获取第r1行、第 “f1:c3” 列的值

备注: f1是列族, c1和c3都是列

```
hbase(main):012:0> get 'tempTable', 'r1', {COLUMN=>'f1:c1'}
COLUMN          CELL
f1:c1          timestamp=1430036599391, value=hello, db=lab
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



4.6.2 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
```



4.6.3 HBase常用Java API及应用实例

HBase是Java编写的，它的原生的API也是Java开发的，不过，可以使用Java或其他语言调用API来访问HBase

首先要在工程中导入一下jar包：

这里只需要导入hbase安装目录中的lib目录下的所有jar包，并且导入“lib/client-facing-thirdparty”目录下的所有jar包。此处不用再导入第3章Hadoop中的jar包，避免由于Hadoop和HBase的版本冲突引起错误。



4.6.3 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。逻辑视图如表4-18所示。

表4-18 学生信息表的表结构

name	score		
	English	Math	Computer

表4-19 需要添加的数据

name	score		
	English	Math	Computer
zhangsan	69	86	77
lisi	55	100	88



4.6.3 HBase常用Java API及应用实例

```
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 Chapter4{
    public static Configuration configuration;
    public static Connection connection;
    public static Admin admin;

    public static void main(String[] args) throws IOException{
        createTable("student", new String[]{"score"});
        insertData("student", "zhangsan", "score", "English", "69");
        insertData("student", "zhangsan", "score", "Math", "86");
        insertData("student", "zhangsan", "score", "Computer", "77");
        getData("student", "zhangsan", "score", "English");
    }
    .....
    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
}
```



4.6.3 HBase常用Java API及应用实例

- 建立连接，关闭连接

```
//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();
    }
}
```

备注: hbase-site.xml

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

(单机版) file:/// DIRECTORY/hbase



4.6.3 HBase常用Java API及应用实例

- 建立连接，关闭连接

```
//关闭连接
public static void close(){
    try{
        if(admin != null){
            admin.close();
        }
        if(null != connection){
            connection.close();
        }
    }catch (IOException e){
        e.printStackTrace();
    }
}
```



4.6.3 HBase常用Java API及应用实例

① 创建表

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。逻辑视图如表4-18所示。

表4-18 学生信息表的表结构

name	score		
	English	Math	Computer



4.6.3 HBase常用Java API及应用实例

```
/*创建表*/
/**
 * @param myTableName 表名
 * @param colFamily列族数组
 * @throws Exception
 */
public static void createTable(String myTableName, String[] colFamily) throws IOException {
    TableName tableName = TableName.valueOf(myTableName);
    if(admin.tableExists(tableName)){
        System.out.println("talbe is exists!");
    }else {
        TableDescriptorBuilder tableDescriptor = TableDescriptorBuilder.newBuilder(tableName);
        for(String str:colFamily){
            ColumnFamilyDescriptor family =
                ColumnFamilyDescriptor.newBuilder(Bytes.toBytes(str)).build();
            tableDescriptor.setColumnFamily(family);
        }
        admin.createTable(tableDescriptor.build());
    }
}
```

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’



4.6.3 HBase常用Java API及应用实例

②添加数据

现在向表student中添加如表4-19所示的数据。

表4-19 需要添加的数据

表4-19 需要添加的数据

name	score		
	English	Math	Computer
zhangsan	69	86	77
lisi	55	100	88



4.6.3 HBase常用Java API及应用实例

```
/*添加数据*/
/**
 * @param tableName 表名
 * @param rowKey 行键
 * @param colFamily 列族
 * @param col 列限定符
 * @param val 数据
 * @throws Exception
 */
```

```
public static void insertData(String tableName, String rowKey, String colFamily, String col, String val) throws IOException {
    Table table = connection.getTable(TableName.valueOf(tableName));
    Put put = new Put(rowKey.getBytes());
    put.addColumn(colFamily.getBytes(), col.getBytes(), val.getBytes());
    table.put(put);
    table.close();
}
```



4.6.3 HBase常用Java API及应用实例

添加数据时，需要分别设置参数myTableName、rowkey、colFamily、col、val的值，然后运行上述代码

例如添加表4-19第一行数据时，为insertData()方法指定相应参数，并运行如下3行代码：

```
insertData("student","zhangsan","score","English","69");
insertData("student","zhangsan","score","Math","86");
insertData("student","zhangsan","score","Computer","77");
```

上述代码与如下HBase Shell命令等效：

```
put 'student','zhangsan','score:English','69';
put 'student','zhangsan','score:Math','86';
put 'student','zhangsan','score:Computer','77';
```



4.6.3 HBase常用Java API及应用实例

③浏览数据

```
/*获取某单元格数据*/
/**    * @param tableName 表名
     * @param rowKey 行键
     * @param colFamily 列族
     * @param col 列限定符
     * @throws IOException */
public static void getData(String tableName, String rowKey, String colFamily,
String col) throws IOException{
    Table table = connection.getTable(TableName.valueOf(tableName));
    Get get = new Get(rowKey.getBytes());
    get.addColumn(colFamily.getBytes(), col.getBytes());
    Result result = table.get(get);
    System.out.println(new
String(result.getValue(colFamily.getBytes(), col==null?null:col.getBytes())));
    table.close();
}
```



4.6.3 HBase常用Java API及应用实例

比如，现在要获取姓名为“zhangsan”在“English”上的数据，就可以在运行上述代码时，指定参数tableName为“student”、rowKey为“zhangsan”、colFamily为“score”、col为“English”。

```
getData("student", "zhangsan", "score", "English");
```

上述代码与如下HBase Shell命令等效：

```
get 'student','zhangsan',{COLUMN=>'score:English'}
```



本章小结

- 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



Thank You!