# HBase简介

是一个**存储框架**（Hive是一个分析框架）

**面向列族**存储的**单表**非关系型数据库（本质上的**Key-Value**）

不适合做关联查询（类似join等操作）

不适合做聚合操作（类似sum）

只是作为数据存储，及简单查询

## HBase数据模型

### HBase逻辑结构

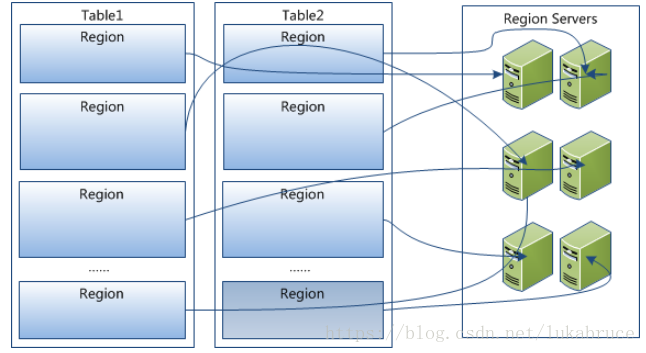


逻辑上和关系型数据库的表类似，实际上是K-V类型存储。

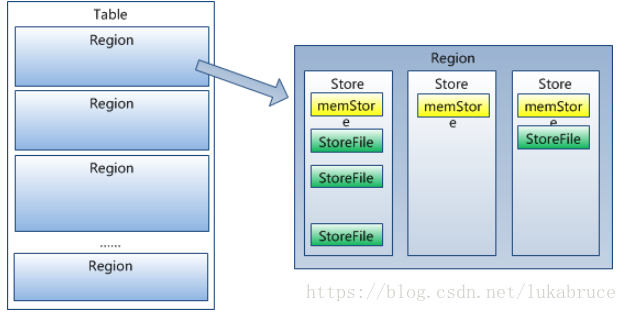
**Row key**：**唯一**。按**字典顺序**排序。1 10 2 20 21 3 31...... 需要用0做左填充

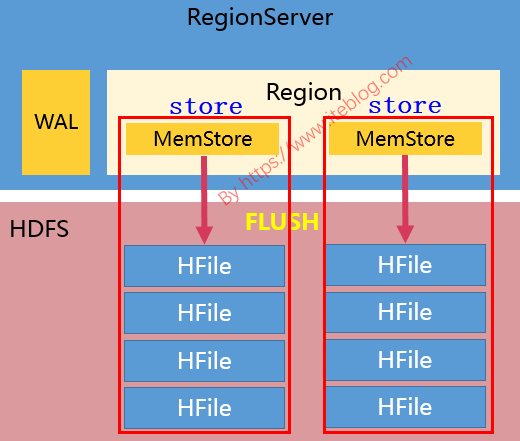
**列族**：纵向切分表

**Region**：横向切分表



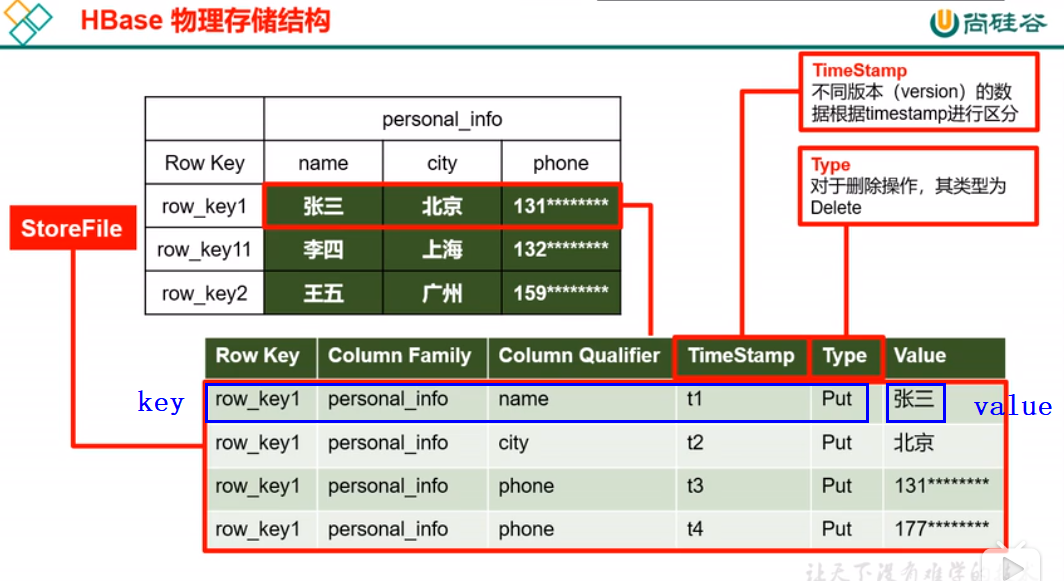
**Store**：一个Region中根据列族分成多个Store（每个Store保存一个列族）。每个Store由一个memStore和0至多个storeFile组成，storeFile以HFile的格式存储在HDFS上。

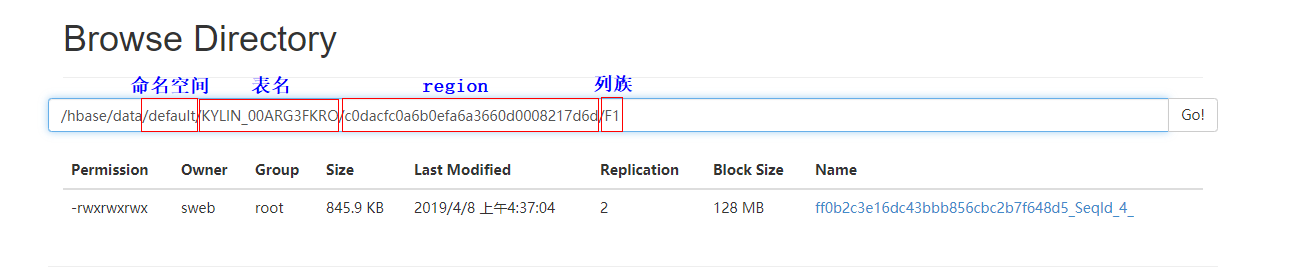




### HBase物理结构

storeFile是以Key-Value的格式存储在HDFS上的



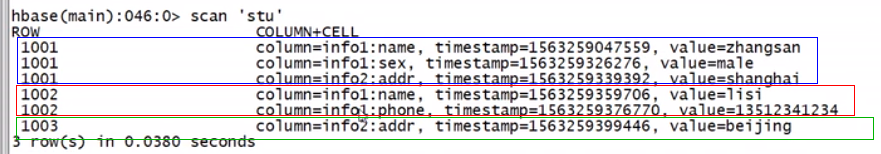


### 1.1.3HBase数据模型

**Name space**：类似于database。自带**hbase**和**default**。default是用户默认命名空间。

**Region**：类似于表。HBase定义表时**只指定列族，不指定列**。意味着HBase写入数据时，可以**动态**、**按需**指定字段，能够轻松应对字段变更的场景。随着数据增长，Region会拆分。是分布式存储的最小单元，但并不是存储的最小单元（以Region为单位分布在不同的机器上，单台机器上的单个Region里面有多个storeFile存在当前机器中）

**Row**：每行数据由一个**Row Key**和多个列组成，数据按照**Row Key字典排序**。Scan操作的时候，只要row key相同，那就是同一条数据，只不过把每个列上的数据都展示出来而已。



**Column**：HBase的每个列有**Column Family**（**列族**）和**Column Qualifier**（**列限定符**）限定。建表时，只需要列族，不需要列限定符。插入数据时自定义列限定符。

**Timestamp**：用于表示数据的不同版本，写入数据时如果不指定时间戳，HBase会自动加上。

**Cell**：由{**Row Key、Column Family、Column Qualifier、Timestamp**}唯一确定的单元，cell中的数据没有类型，全部是字节码格式存储。

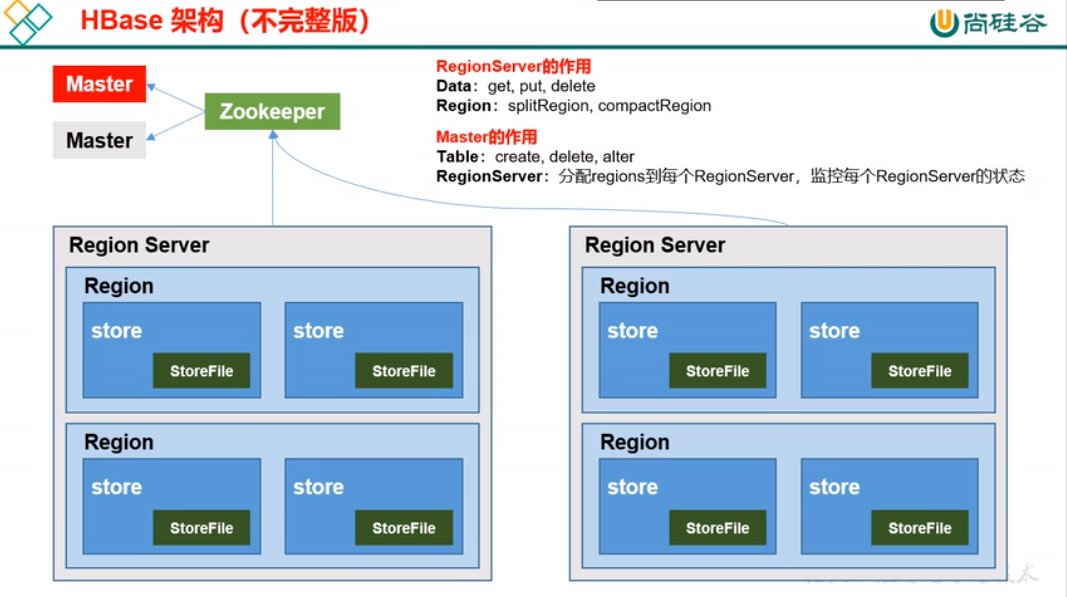
**Row Key、Column Family、Column Qualifier唯一确定一条数据，Timestamp表示数据版本。**

## HBase基本架构

Zookeeper和多个master配合组成高可用。

Master：DDL，对表的操作，用来管理RegionServer

RegionServer：DML，对表中数据的操作，用来管理Region



# HBase集群搭建

# HBase集群启动和关闭

## 1.1 bin目录

bin目录下：

**hbase-daemon.sh**、**hbase-daemons.sh**是针对单个节点

**start-hbase.sh**、**stop-hbase.sh**是针对整个集群

**hbase**是进入hbase内部sh

## 1.2启动

**hbase-daemon.sh start master ：**启动当前机器作为master

**hbase-daemon.sh start regionserver ：**启动当前机器作为regionserver

**start-hbase.sh ：**启动当前机器作为master，其他机器作为regionserver启动

**stop-hbase.sh ：**关闭所有机器

./hbase shell ：进入shell

Hbase在一台节点上start-hbase.sh，就可以将自身作为HMaster，其他作为HRegionServer（已启动的节点会提示已启动，只会启动未启动的）。然后在另外一台节点上start-hbase.sh就会将自己作为backup的master。这样集群就是一个active的master，一个backup的master，加若干个regionServer。

或者，在某一台节点的conf下创建backup\_master文件，里面是要指定的backup节点的ip或者域名，然后start-hbase.sh，这时当前节点就是active的master，backup\_master文件中的节点就是backup的master，加若干个regionServer

使用 masterIp:16010可以查看hbase集群的信息

## 1.3 DDL命令

### 1.3.1 表操作

**list ：** 查看所有的表

**create** ：创建表。create ‘**spacename:**table\_name’, ‘列族1’,’列族2’,......

**describe** ： 查看表属性。describe ‘table\_name’

NAME => ‘列族’, VERSIONS => ‘1’字段，表示列族下的某个数据可以有多个版本，多于该值后，将删除timestamp小的。

**alter** ：修改表属性。

alter ‘table\_name’ , {NAME => ‘列族名’, VERSIONS => 3} 将表中某个列族的VERSIONS改成3

**disable ：**将表下线

**drop** ：删除表

### 1.3.2 命名空间操作

**list\_namespace** ： 查看所有命名空间

**create\_namespace** ： 创建命名空间 create\_namespce ‘空间名’

**drop\_namespce** ：删除命名空间，需要先删除其中的所有表

### 1.4 DML操作

**put** ：新增数据

put ‘表名’ , ‘row key’ , ‘列族:列名’ , ’value值’ , ‘时间戳’

**scan** ： 扫描全表

scan ‘表名’

scan ‘表名’ , **{RAW => true , VERSIONS => 10}** ：显示十个version内的数据

**get** ：查询

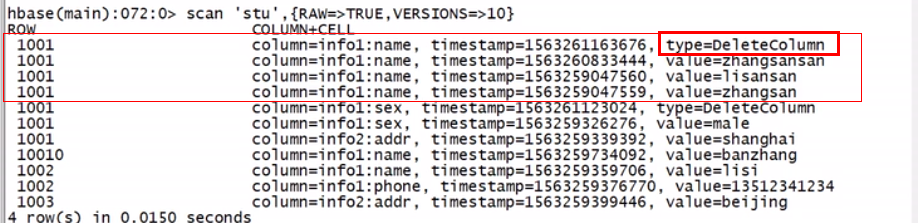
get ‘表名’ , ‘row key’ , ‘列族:列’

**put** ：修改

put ‘表名’ , ‘row key’ , ‘列族:列’ , ‘value’ , 时间戳 ：修改数据，自定义时间戳

**delete** ：删除

delete ‘表名’ , ‘row key’ , ‘列族:列’ ：执行删除后，它会增加一个**type=DeleteColumn**的数据，表示其被删除了。

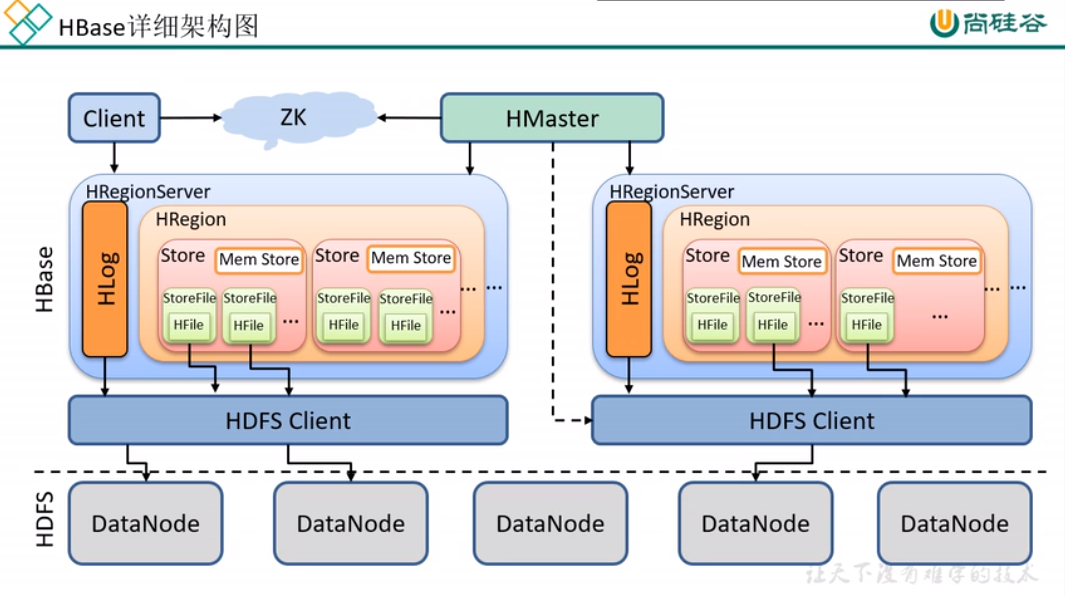


**deleteall** ：删除row key

deleteall ‘表名’ , ‘row key’ ：执行删除后，它会增加一个**type=DeleteFamily**的数据

# HBase高级

## 1.1 HBase详细架构



## 1.2 HBase写流程

1、client请求zk，找hbase:meta表在哪台regionserver上（rs1）

2、请求rs1中hbase:meta表，查找要查询的表和row key在哪台regionserver上（rs2），并将hbase:meta表和该表的region信息缓存到client的meta cache中，方便下次访问。

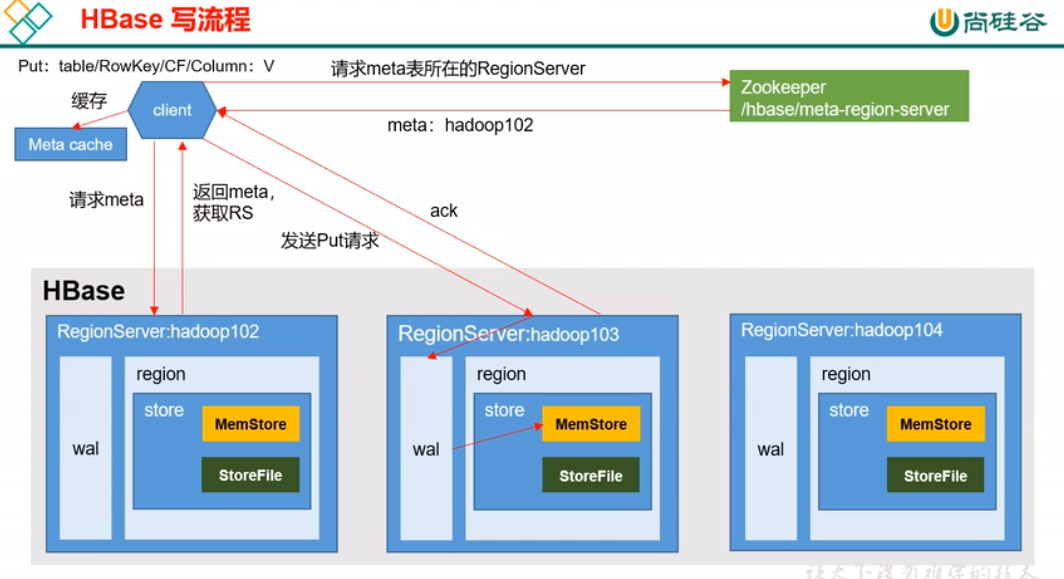
3、与目标regionserver进行通讯

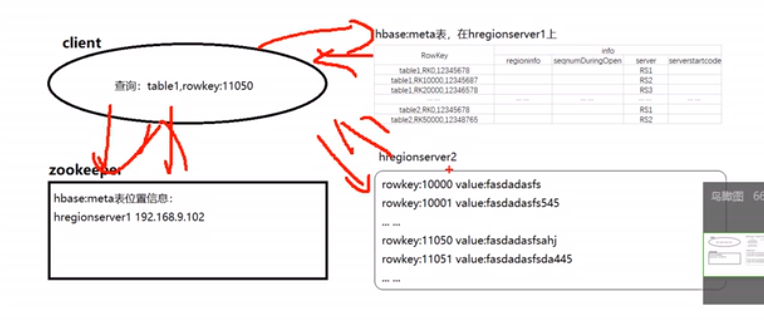
4、将数据顺序写入（追加）到WAL中

5、将数据写入对应的MemStore中，数据会在MemStore进行排序

6、regionserver向client发送ack，表示已经写入数据

7、等到达MemStore的flush时机后，regionserver将数据刷写到HFile中（通过HDFS Client落到HDFS中）





## HBase flush流程

刚写入数据到HBase之后，在HDFS文件目录中是看不到有文件生成的，因为文件还存在HBase的内存中，等到了flush刷写时机之后，才会把数据写到HDFS中。

store的memStore中的数据（内存）写到storeFile中（磁盘）

## 1.4 HBase读流程

1、client请求zk，找hbase:meta表在哪台regionserver上（rs1）

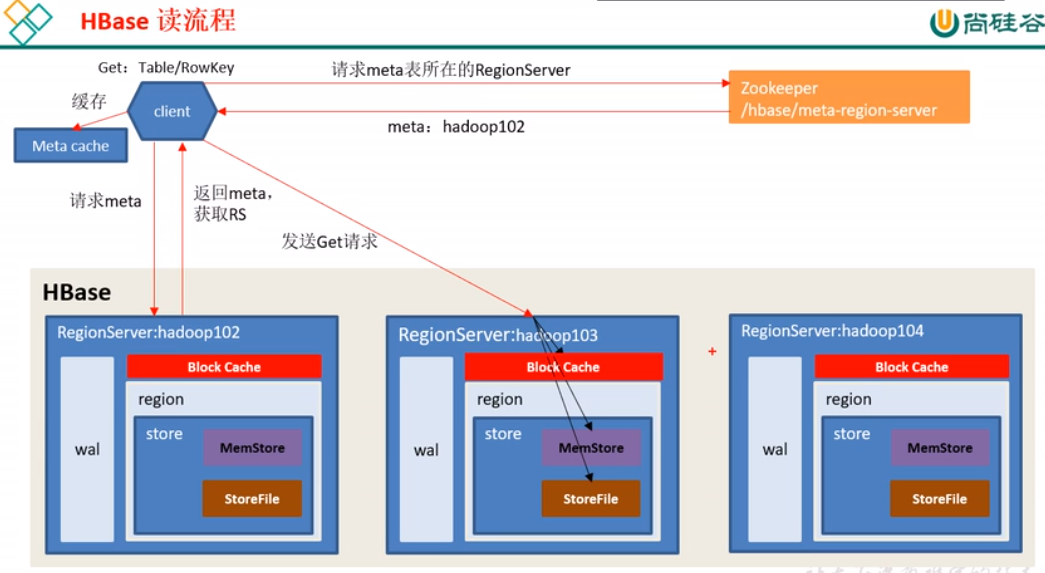
2、请求rs1中hbase:meta表，查找要查询的表和row key在哪台regionserver上（rs2），并将hbase:meta表和该表的region信息缓存到client的meta cache中，方便下次访问。

3、与目标regionserver进行通讯

4、分别在**block cache（读缓存）、MemStore和StoreFile（HFile）**中查询目标数据，并将

查到的**所有数据**进行合并。此处的**所有数据**是指同一条数据的不同版本（timestamp）或者不同类型（put/delete）。

1. 将从文件中查询到的数据块（Block，HFile数据存储单元，默认大小是64KB）缓存到Block Cache。
2. 将合并后的最终结果返回给客户端。



## 1.5 合并流程

由于MemStore每次刷写都会生成一个新的HFile，且同一字段的不同版本（timestamp）和不同类型（put/delete）可能分布到不同的HFile中，查询时需要遍历所有的HFile。为了**减少HFile个数**，以及**清理过期和删除的数据**，会进行**StoreFile Compaction**。

**Minor compaction**

将一些**小的、相邻**的HFile合并成一个更大的HFile

**Major compaction**

将一个**Store**下的的所有HFile合并成一个HFile，并执行**物理删除操作**

## 1.6 数据真正删除的时间

## 1.7 拆分流程

# HBase优化

## 高可用

1、选择一台习惯性将至作为master的节点，在config/backup-masters中配置备份master节点名。

2、使用start-hbase.sh启动：当前节点为master，backup-masters中的节点为备份master节点，所有节点都是regionserver节点。

## 预分区

每一个region维护着startRow和endRow，如果加入的数据符合某个region维护的RowKey范围，则该数据交给这个region维护。那么按照这个原则，我们可以将数据所要投放的分区提前大致的规划好，以提高HBase性能。

## RowKey设计

一条数据的唯一标识就是RowKey，那么这条数据存储在哪个分区，取决于RowKey处于哪个预分区的区间内，设计RowKey的主要目的，就是让数据均匀的分布于所有的region中，在一定程度上防止数据倾斜。

### 设计原则

**散列性**，**唯一性**，**长度原则（70-100）**

### 设计方案

#### 生成随机数、hash、散列值

要考虑散列性和集中性（？？？）

#### 字符串反转

#### 1.3.2.3 字符串拼接

### 1.3.3设计情景

## 内存优化

## 基础优化