## HBase简介

HBase 是 BigTable 的开源（源码使用 Java 编写）版本。是 Apache Hadoop 的数据库，是建 立在 HDFS 之上，被设计用来提供高可靠性、高性能、列存储、可伸缩、多版本的 NoSQL 的分布式数据存储系统，实现对大型数据的实时、随机的读写访问。

HBase 依赖于 HDFS 做底层的数据存储，BigTable 依赖 Google GFS 做数据存储

HBase 依赖于 MapReduce 做数据计算，BigTable 依赖 Google MapReduce 做数据计算

HBase 依赖于 ZooKeeper 做服务协调，BigTable 依赖 Google Chubby 做服务协调

NoSQL = NO SQL

NoSQL = Not Only SQL：会有一些把 NoSQL 数据的原生查询语句封装成 SQL，比如 HBase 就有 Phoenix 工具

### 关系型数据库 和 非关系型数据库的典型代表

NoSQL：hbase, redis, mongodb

RDBMS：mysql,oracle,sql server,db2

### HBase 这个 NoSQL 数据库的要点

① 它介于 NoSQL 和 RDBMS 之间，**仅能通过主键(rowkey)和主键的 range 来检索数据**

② HBase 查询数据功能很简单，**不支持 join 等复杂操作**

③ 不支持复杂的事务，**只支持行级事务**(可通过 hive 支持来实现多表 join 等复杂操作)。

④ **HBase 中支持的数据类型：byte[]（底层所有数据的存储都是字节数组）**

⑤ **主要用来存储结构化和半结构化的松散数据。**

### 结构化、半结构化和非结构化

**结构化**：数据结构字段含义确定，清晰，典型的如数据库中的表结构

**半结构化**：具有一定结构，但语义不够确定，典型的如 HTML 网页，有些字段是确定的(title)， 有些不确定(table)

**非结构化**：杂乱无章的数据，很难按照一个概念去进行抽取，无规律性

与 Hadoop 一样，HBase 目标主要依靠横向扩展，通过不断增加廉价的商用服务器，来增加 计算和存储能力。

### HBase 中的表特点

1、**大**：一个表可以有上十亿行，上百万列

2、**面向列**：面向列(族)的存储和权限控制，列(簇)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。

4、**无模式**：每行都有一个可排序的主键和任意多的列，列可以根据需要动态的增加，同一 张表中不同的行可以有截然不同的列



## HBase表结构逻辑视图

初次接触HBase，可能看到以下描述会懵：**“基于列存储”，“稀疏MAP”，“RowKey”,“ColumnFamily”。**

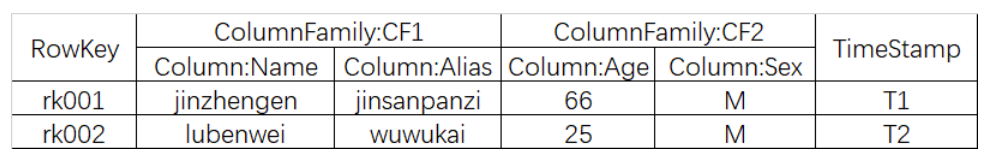
其实没那么高深，我们需要分两步来理解HBase, 就能够理解为什么HBase能够“快速地”“分布式地”处理“大量数据”了。

**1.内存结构**

**2.文件存储结构**

### ****名词概念****

加入我们有如下一张表



#### Rowkey的概念

**Rowkey的概念和mysql中的主键是完全一样的，Hbase使用Rowkey来唯一的区分某一行的数据。**

由于Hbase只支持3中查询方式：

1、基于Rowkey的单行查询

2、基于Rowkey的范围扫描

3、全表扫描

因此，Rowkey对Hbase的性能影响非常大，Rowkey的设计就显得尤为的重要。设计的时候要兼顾基于Rowkey的单行查询也要键入Rowkey的范围扫描。具体Rowkey要如何设计后续会整理相关的文章做进一步的描述。这里大家只要有一个概念就是Rowkey的设计极为重要。

rowkey 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)，最好是 16。在 HBase 内部，rowkey 保存为字节数组。**HBase 会对表中的数据按照 rowkey 排序 (字典顺序)**

#### ****Column的概念****

列，可理解成MySQL列。

#### ColumnFamily的概念

列族, HBase引入的概念。

Hbase通过列族划分数据的存储，列族下面可以包含任意多的列，实现灵活的数据存取。就像是家族的概念，我们知道一个家族是由于很多个的家庭组成的。列族也类似，列族是由一个一个的列组成（任意多）。

Hbase表的创建的时候就必须指定列族。就像关系型数据库创建的时候必须指定具体的列是一样的。

Hbase的列族不是越多越好，官方推荐的是列族最好小于或者等于3。我们使用的场景一般是1个列族。

#### TimeStamp的概念

TimeStamp对Hbase来说至关重要，因为它是实现Hbase多版本的关键。在Hbase中使用不同的timestame来标识相同rowkey行对应的不通版本的数据。

HBase 中通过 rowkey 和 columns 确定的为一个存储单元称为 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：

**保存数据的最后 n 个版本**

**保存最近一段时间内的版本（设置数据的生命周期 TTL）。**

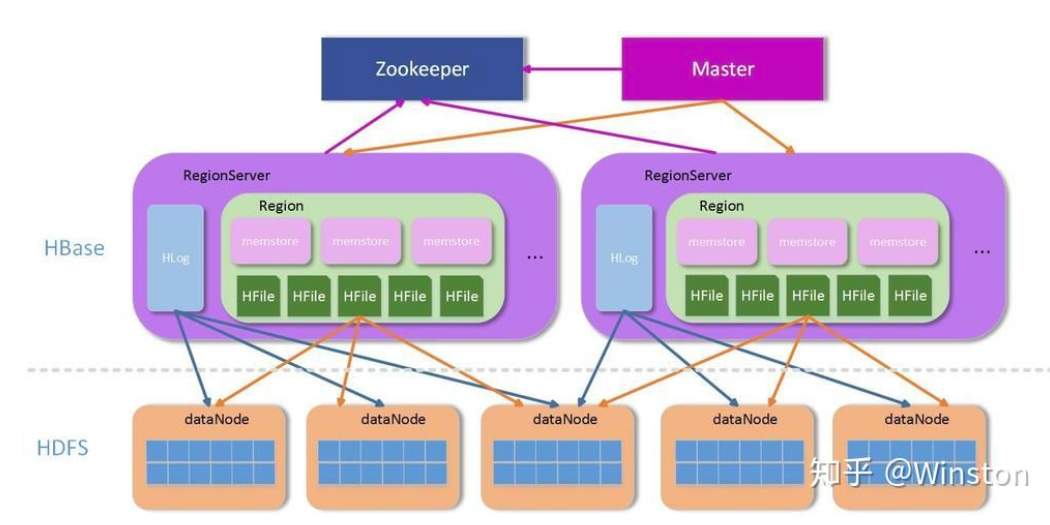
用户可以针对每个列簇进行设置。

#### 单元格（Cell）

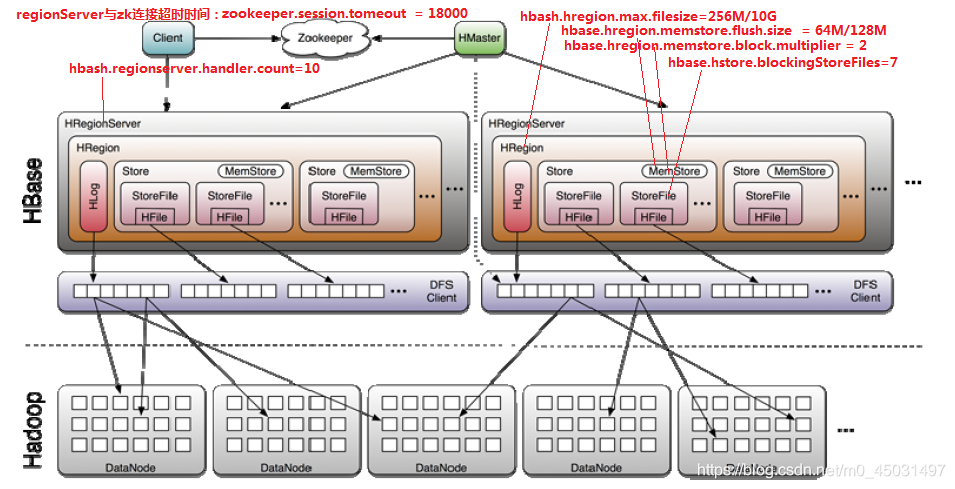
由{rowkey, column( = + ), version} 唯一确定的单元。 Cell 中的数据是没有类型的，全部是字节码形式存贮。

## HBase架构

在HBase中，表被分割成区域，并由区域服务器提供服务。区域被列族垂直分为“Stores”。Stores被保存在HDFS文件。下面显示的是HBase的结构。注意：术语“store”是用于区域来解释存储结构。 - 依赖于HDFS做底层的数据存储 - 依赖于MapReduce做数据计算 - 依赖于ZooKeeper做服务协调



HBase有三个主要组成部分：客户端库，主服务器和区域服务器。区域服务器可以按要求添加或删除。



### 写流程

client向hregionserver发送写请求。

hregionserver将数据写到hlog（write ahead log）。为了数据的持久化和恢复。

hregionserver将数据写到内存（memstore）

反馈client写成功。

### 数据flush过程

当memstore数据达到阈值（默认是64M），将数据刷到硬盘，将内存中的数据删除，同时删除Hlog中的历史数据。

并将数据存储到hdfs中。

在hlog中做标记点。

### 数据合并过程

当数据块达到4块，hmaster将数据块加载到本地，进行合并

当合并的数据超过256M，进行拆分，将拆分后的region分配给不同的hregionserver管理

当hregionser宕机后，将hregionserver上的hlog拆分，然后分配给不同的hregionserver加载，修改.META.

注意：hlog会同步到hdfs

### hbase的读流程

通过zookeeper和-ROOT- .META.表定位hregionserver。

数据从内存和硬盘合并后返回给client

数据块会缓存

### hmaster的职责

管理用户对Table的增、删、改、查操作；

记录region在哪台Hregion server上

在Region Split后，负责新Region的分配；

新机器加入时，管理HRegion Server的负载均衡，调整Region分布

在HRegion Server宕机后，负责失效HRegion Server 上的Regions迁移。

### hregionserver的职责

 HRegion Server主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBASE中最核心的模块。

 HRegion Server管理了很多table的分区，也就是region。

### client职责

Client

 HBASE Client使用HBASE的RPC机制与HMaster和RegionServer进行通信

 管理类操作：Client与HMaster进行RPC；

 数据读写类操作：Client与HRegionServer进行RPC。

8.MapReduce操作Hbase

8.1.实现方法

Hbase对MapReduce提供支持，它实现了TableMapper类和TableReducer类，我们只需要继承这两个类即可。

1、写个mapper继承TableMapper<Text, IntWritable>

参数：

Text：mapper的输出key类型；

IntWritable：mapper的输出value类型。

其中的map方法如下：

map(ImmutableBytesWritable key, Result value,Context context)

参数：

key：rowKey；

value： Result ，一行数据；

context：上下文

2、写个reduce继承TableReducer<Text, IntWritable, ImmutableBytesWritable>

参数：

Text:reducer的输入key；

IntWritable：reduce的输入value；

ImmutableBytesWritable：reduce输出到hbase中的rowKey类型。

其中的reduce方法如下：

reduce(Text key, Iterable values,Context context)

参数：

key：reduce的输入key；

values：reduce的输入value；

https://blog.csdn.net/m0\_45031497/article/details/94629345