前言

在我还不了解分布式和大数据的时候已经听说过**HBase**了，但对它一直都半知不解，这篇文章来讲讲吧。

在真实生活中，最开始听到这个词是我的一场面试，当年我还是个『小垃圾』，现在已经是个『大垃圾』了。

面试官当时给了一个场景题问我，具体的题目我忘得差不多了，大概就是考试与试题的一个场景，问我数据库要如何设计。

我答了关系型数据库的设计方案，他大概说：**这个场景比较复杂多变，为什么不考虑一下HBase这种NoSQL的数据库来存储呢**?

我就说：“**对对对，可以的**”（虽然我当时不知道HBase是什么，但是**气势一定要有**，你们说是不是）

最后面试官还是给我发了offer，但我没去，原因就是离家太远了。

一、介绍HBase

> Apache HBase™ is the Hadoop database, a distributed, scalable, big data store.
>
> HBase is a type of "NoSQL" database.

Apache HBase 是 Hadoop 数据库，一个分布式、可伸缩的**大数据存储**。

HBase是依赖Hadoop的。为什么HBase能存储海量的数据？**因为HBase是在HDFS的基础之上构建的，HDFS是分布式文件系统**。

二、为什么要用HBase

截至到现在，三歪已经学了不少的组件了，比如说分布式搜索引擎「Elasticsearch」、分布式文件系统「HDFS」、分布式消息队列「Kafka」、缓存数据库「Redis」等等...

能够处理数据的中间件(系统)，这些中间件基本都会有持久化的功能。为什么？如果某一个时刻挂了，那**还在内存但还没处理完的数据**不就凉了？

Redis有AOF和RDB、Elasticsearch会把数据写到translog然后结合FileSystemCache将数据刷到磁盘中、Kafka本身就是将数据顺序写到磁盘....
这些中间件会实现持久化（像HDFS和MySQL我们本身就用来存储数据的），为什么我们还要用HBase呢?
虽然没有什么可比性，但是在学习的时候总会有一个疑问：「既然已学过的系统都有类似的功能了，那为啥我还要去学这个玩意？」
三歪是这样理解的：

- MySQL？MySQL数据库我们是算用得最多了的吧？但众所周知，MySQL是**单机**的。MySQL能存储多少数据，取决于那台服务器的硬盘大小。以现在互联网的数据量，很多时候MySQL是没法存储那么多数据的。
- 比如我这边有个系统，一天就能产生1TB的数据，这数据是不可能存MySQL的。（如此大的量数据，我们现在的做法是先写到Kafka，然后落到Hive中）
- Kafka？Kafka我们主要用来处理消息的（解耦异步削峰）。数据到Kafka，Kafka会将数据持久化到硬盘中，并且Kafka是分布式的（很方便的扩展），理论上Kafka可以存储很大的数据。但是Kafka的数据我们**不会「单独」取出来**。持久化了的数据，最常见的用法就是重新设置offset，做「回溯」操作
- Redis？Redis是缓存数据库，所有的读写都在内存中，速度贼快。AOF/RDB存储的数据都会加载到内存中，Redis不适合存大量的数据（因为内存太贵了！）。
- Elasticsearch？Elasticsearch是一个分布式的搜索引擎，主要用于检索。理论上Elasticsearch也是可以存储海量的数据（毕竟分布式），我们也可以将数据用『索引』来取出来，似乎已经是非常完美的中间件了。
- 但是如果我们的数据**没有经常「检索」的需求**，其实不必放到Elasticsearch，数据写入Elasticsearch需要分词，无疑会浪费资源。
- HDFS？显然HDFS是可以存储海量的数据的，它就是为海量数据而生的。它也有明显的缺点：不支持随机修改，查询效率低，对小文件支持不友好。

文中的开头已经说了，HBase是基于HDFS分布式文件系统去构建的。换句话说，HBase的数据其实也是存储在HDFS上的。那肯定有好奇宝宝就会问：**HDFS和HBase有啥区别阿**?
HDFS是文件系统，而HBase是数据库，其实也没啥可比性。**「你可以把HBase当做是MySQL，把HDFS当做是硬盘。HBase只是一个NoSQL数据库，把数据存在HDFS上」**。
　　数据库是一个以某种**有组织的方式存储的数据集合**。
扯了这么多，那我们为啥要用HBase呢？HBase在HDFS之上提供了**高并发的随机写和支持实时查询**，这是HDFS不具备的。
我一直都说在学习某一项技术之前首先要了解它能干什么。如果仅仅看上面的”对比“，我们可以发现HBase可以以**低成本**来**存储海量**的数据并且支持高并发随机写和实时查询。
但HBase还有一个特点就是：**存储数据的”结构“可以地非常灵活**（这个下面会讲到，这里如果没接触过HBase的同学可能不知道什么意思）。

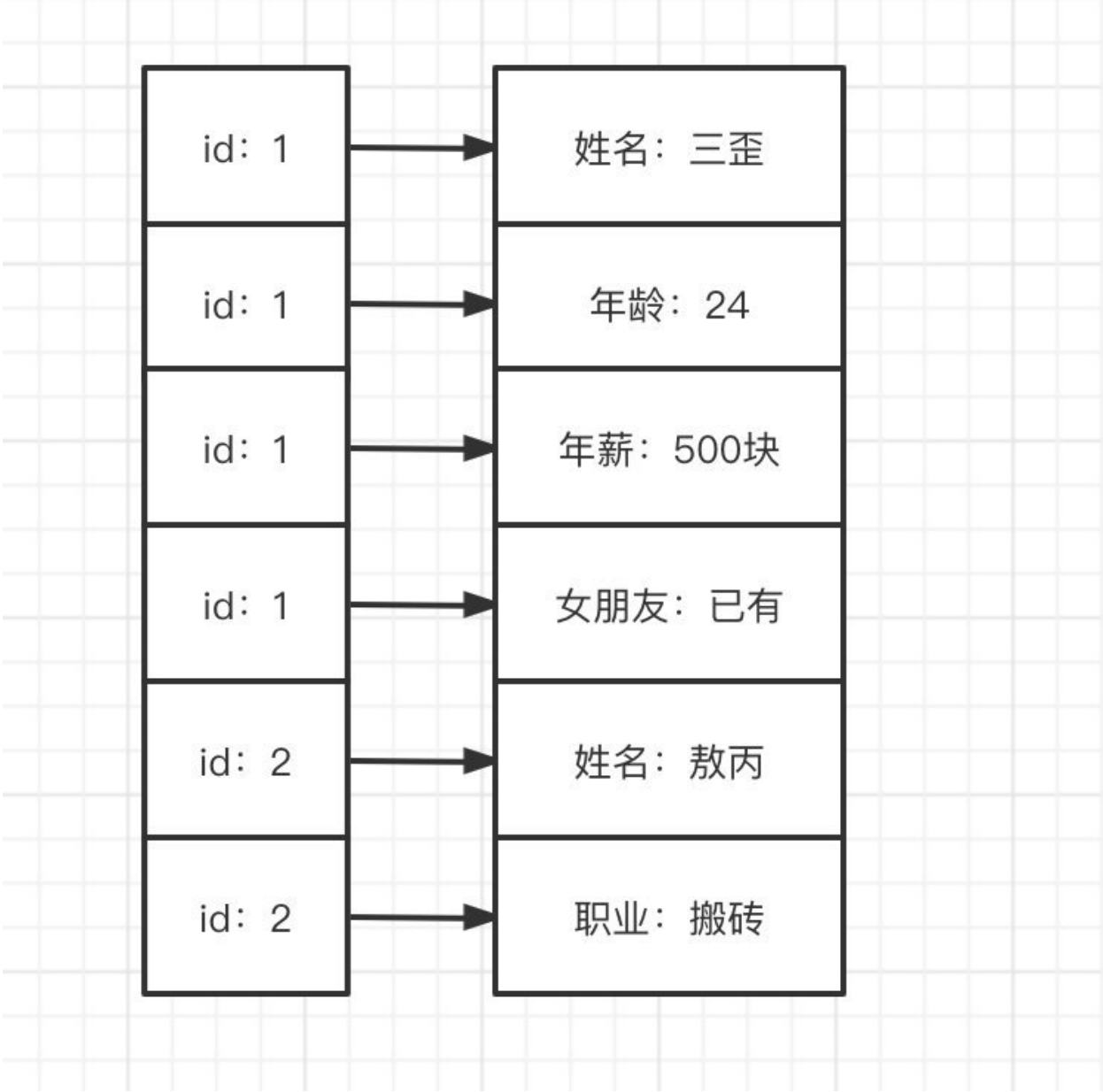
三、入门HBase

听过HBase的同学可能都听过「列式存储」这个词。我最开始的时候觉得HBase很难理解，就因为它这个「列式存储」我一直理解不了它为什么是「列式」的。

在网上也有很多的博客去讲解什么是「列式」存储，它们会举我们现有的数据库，比如MySQL。存储的结构我们很容易看懂，就是一行一行数据嘛。

id	姓名	年龄	职业	年薪	女朋友	身高
1	三歪	24		500块	已有	
2	敖丙		搬砖			

转换成所谓的列式存储是什么样的呢?



可以很简单的发现，无非就是把**每列抽出来，然后关联上Id**。这个叫列式存储吗？**我在这打个问号**。

转换后的数据**从我的角度来看**，数据还是一行一行的。

这样做有什么好处吗？很明显以前我们一行记录多个属性(列)，有部分的列是空缺的，但是我们还是需要空间去存储。现在把这些列全部拆开，有什么我们就存什么，这样**空间就能被我们充分利用**。

这种形式的数据更像什么？明显是`Key-Value`嘛。那我们该怎么理解HBase所谓的列式存储和`Key-Value`结构呢？走进三歪的小脑袋，一探究竟。

3.1 HBase的数据模型

在看HBase数据模型的时候，其实最好还是不要用「关系型数据库」的知识去理解它。

> In HBase, data is stored in tables, which have rows and columns. This is a terminology overlap withrelational databases (RDBMSs), **but this is not a helpful analogy**.

HBase里边也有表、行和列的概念。

- 表没什么好说的，就是一张表
- 一行数据由**一个行键**和**一个或多个相关的列以及它的值**所组成

好了，现在比较抽象了。在HBase里边，定位一行数据会有一个唯一的值，这个叫做行键(**RowKey**)。而在HBase的列不是我们在关系型数据库所想象中的列。

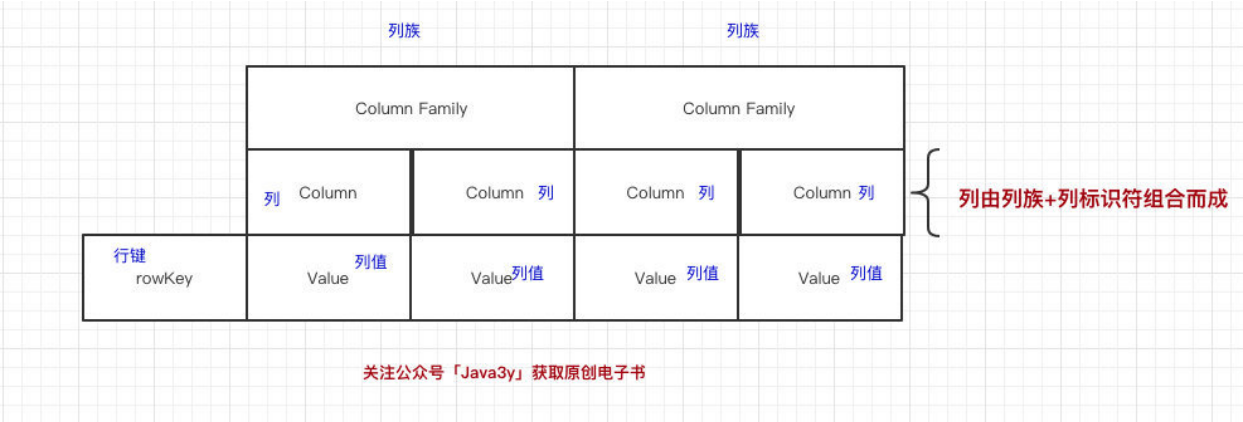
HBase的列（**Column**）都得归属到列族（**Column Family**）中。在HBase中用列修饰符（**Column Qualifier**）来标识每个列。

在HBase里边，先有列族，后有列。

什么是列族？可以简单理解为：列的属性**类别**

什么是列修饰符？**先有列族后有列，在列族下用列修饰符来标识一列。**

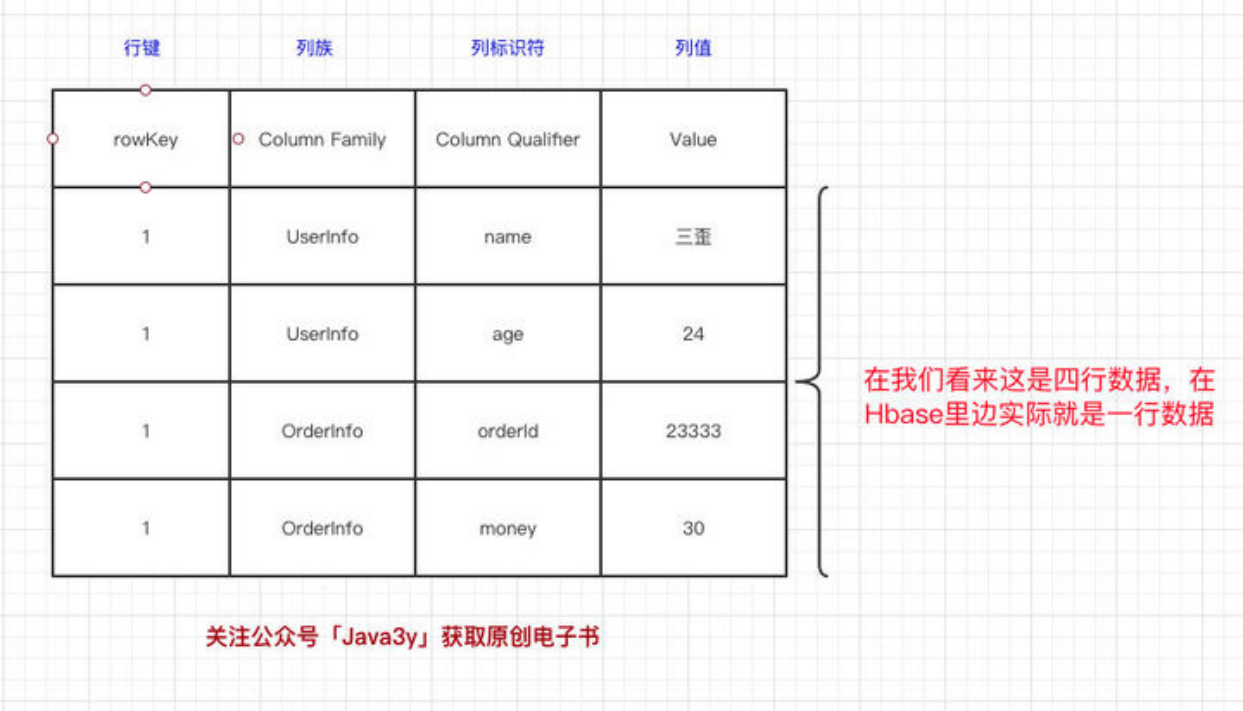
还很抽象是不是？三歪来画个图：



我们再放点具体的值去看看，就更加容易看懂了：

这张表我们有两个列族，分别是`UserInfo`和`OrderInfo`。在UserInfo下有两个列，分别是`UserInfo:name`和`UserInfo:age`，在`OrderInfo`下有两个列，分别是`OrderInfo:orderId`和`OrderInfo:money`。`UserInfo:name`的值为：三歪。`UserInfo:age`的值为24。`OrderInfo:orderId`的值为23333。`OrderInfo:money`的值为30。这些数据的主键(RowKey)为1

上面的那个图看起来可能不太好懂，我们再画一个我们比较熟悉的：



HBase表的每一行中，列的组成都是**灵活**的，**行与行之间的列不需要相同**。如图下：

RowKey	Columns
Row1	{ID, Name, Phone}
Row2	{ID, Name, Address, Title , Email}
Row3	{ID, Address, Email}

Key	联系方式（Column Family）			课程成绩（Column Family）		
001	Weibo：li_zhihui	分机：233		历史：85		地理：77
002		分机：809	QQ：523		英语：78	地理：87
003		分机：523	QQ：908	历史：91	英语：88	

换句话说：**一个列族下可以任意添加列，不受任何限制**

数据写到HBase的时候都会被记录一个**时间戳**，这个时间戳被我们当做一个**版本**。比如说，我们**修改或者删除**某一条的时候，本质上是往里边**新增**一条数据，记录的版本加一了而已。

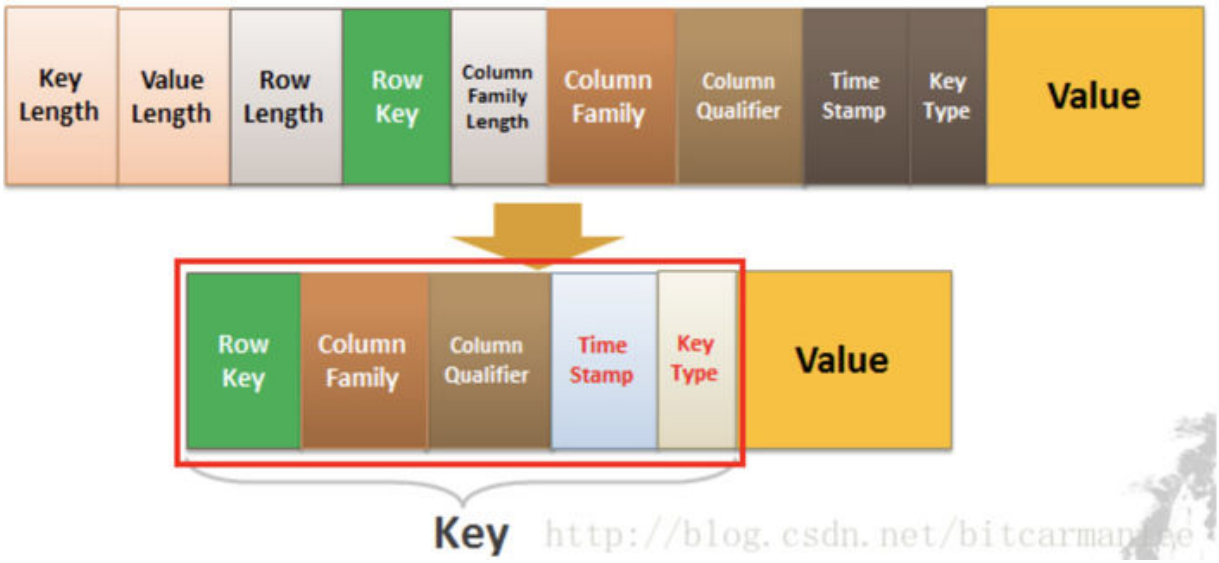
比如现在我们有一条记录：

现在要把这条记录的值改为40，实际上就是多添加一条记录，在读的时候按照时间戳**读最新**的记录。在外界「看起来」就是把这条记录改了。

3.2 HBase 的Key-Value

HBase本质上其实就是`Key-Value`的数据库，上一次我们学`Key-Value`数据库还是Redis呢。那在HBase里边，Key是什么？Value是什么？

我们看一下下面的HBase`Key-Value`结构图：



Key由RowKey(行键)+ColumnFamily（列族）+Column Qualifier（列修饰符）+TimeStamp（时间戳--版本）+KeyType（类型）组成，而Value就是实际上的值。

对比上面的例子，其实很好理解，因为我们修改一条数据其实上是在原来的基础上增加一个版本的，那我们要**准确定位**一条数据，那就得（RowKey+Column+时间戳）。

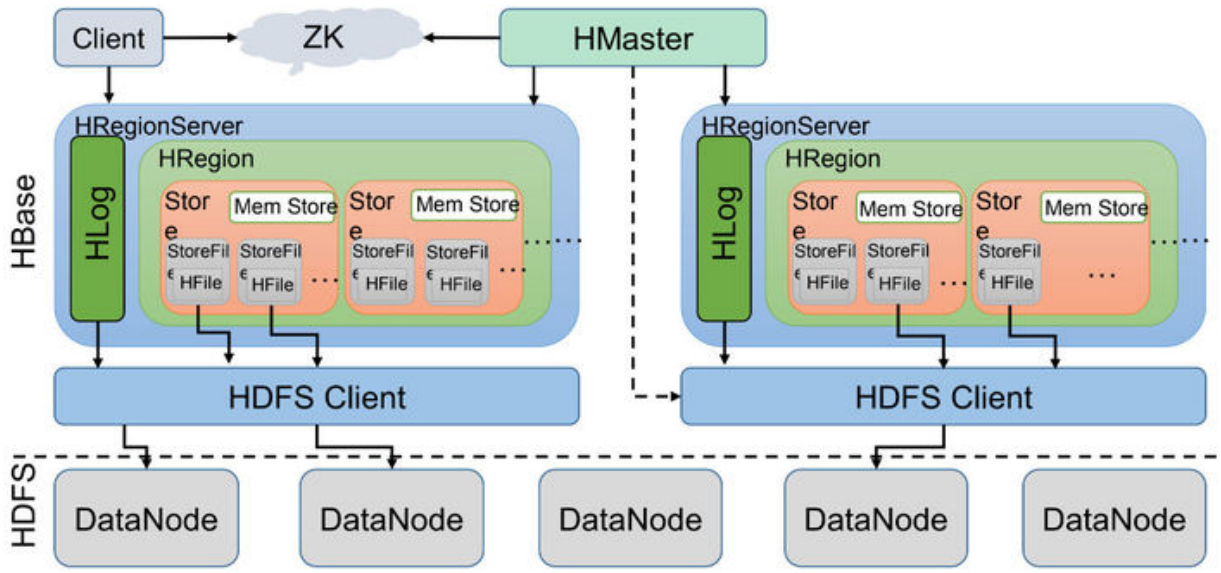
KeyType是什么？我们上面只说了「修改」的情况，你们有没有想过，如果要删除一条数据怎么做？实际上也是增加一条记录，只不过我们在KeyType里边设置为“Delete”就可以了。

3.3 HBase架构

扯了这么一大堆，已经说了HBase的数据模型和Key-Value了，我们还有一个问题：「为什么经常会有人说HBase是列式存储呢？」

其实HBase更多的是「**列族存储**」，要谈列族存储，就得先了解了解HBase的架构是怎么样的。

我们先来看看HBase的架构图：

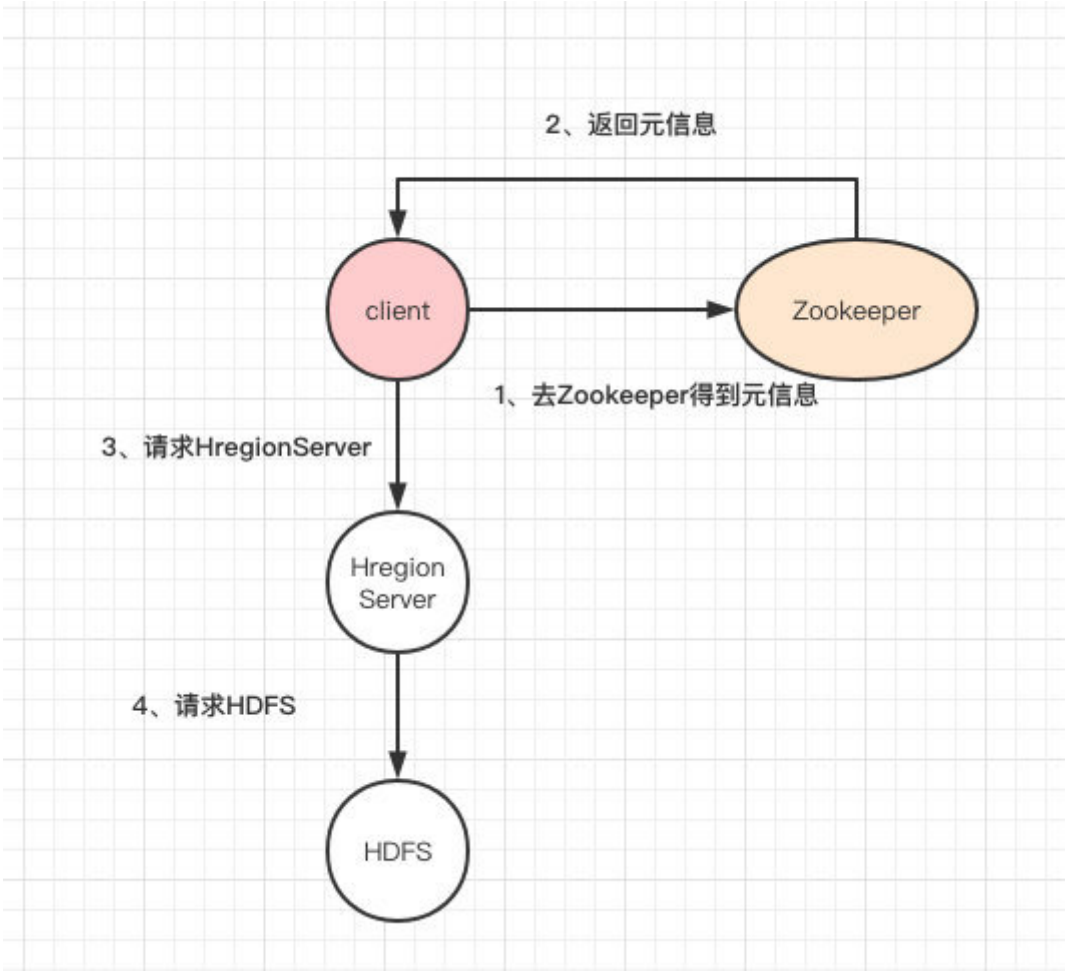


1、**Client**客户端，它提供了访问HBase的接口，并且维护了对应的cache来加速HBase的访问。

2、**Zookeeper**存储HBase的元数据（meta表），无论是读还是写数据，都是去Zookeeper里边拿到meta元数据**告诉给客户端去哪台机器读写数据**

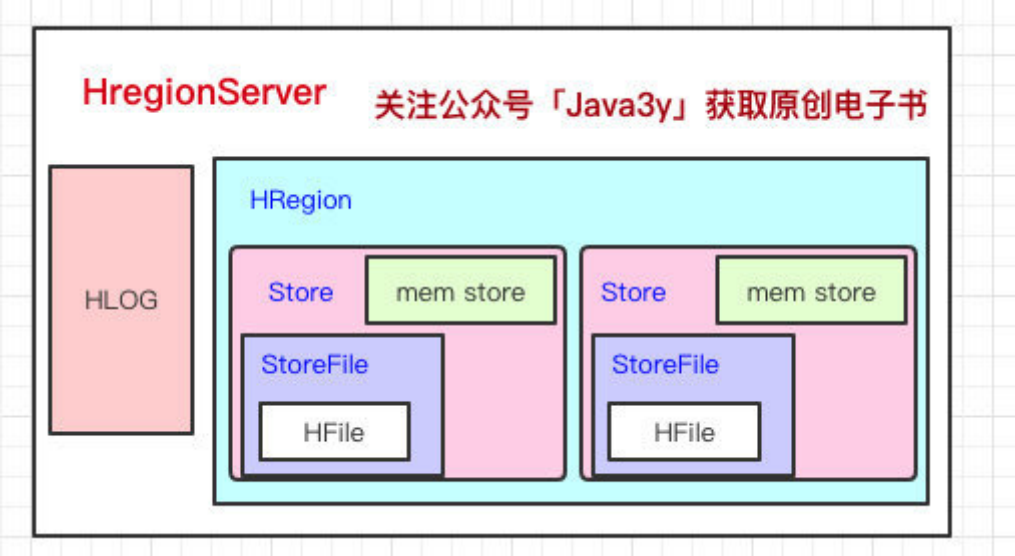
3、**HRegionServer**它是处理客户端的读写请求，负责与HDFS底层交互，是真正干活的节点。

总结大致的流程就是：client请求到Zookeeper，然后Zookeeper返回HRegionServer地址给client，client得到Zookeeper返回的地址去请求HRegionServer，HRegionServer读写数据后返回给client。

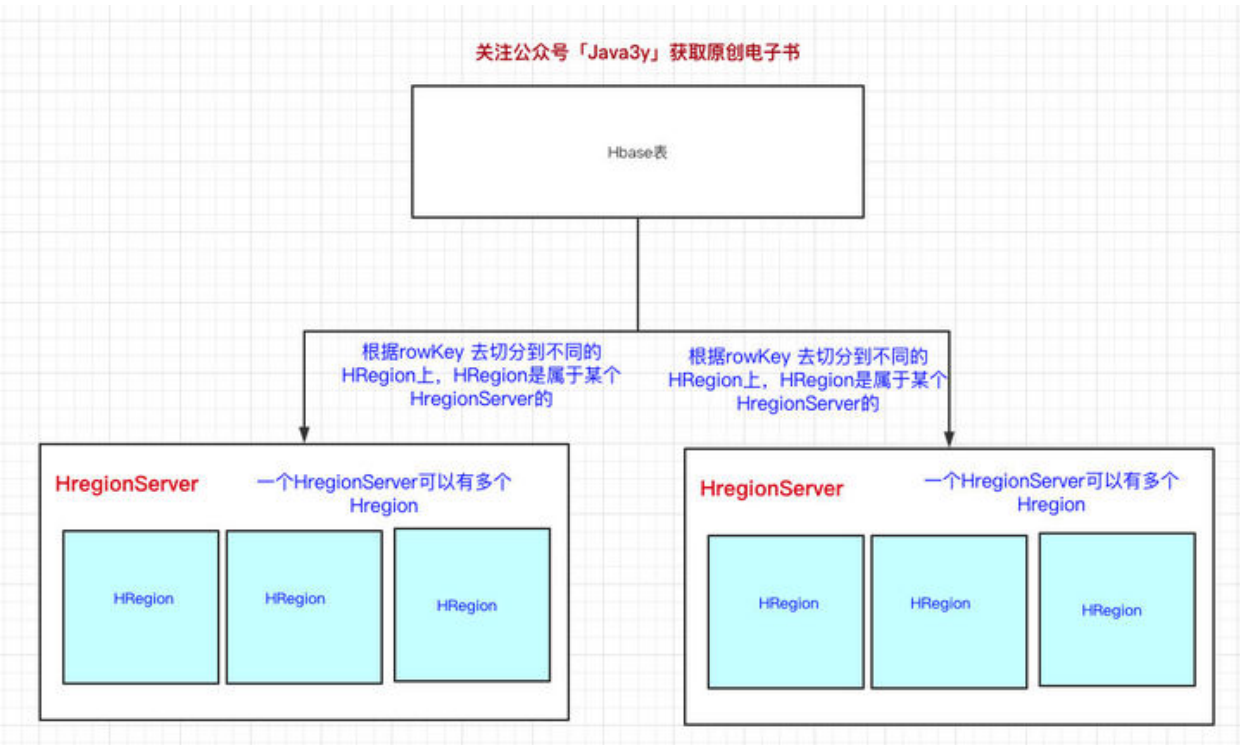


3.4 HRegionServer内部

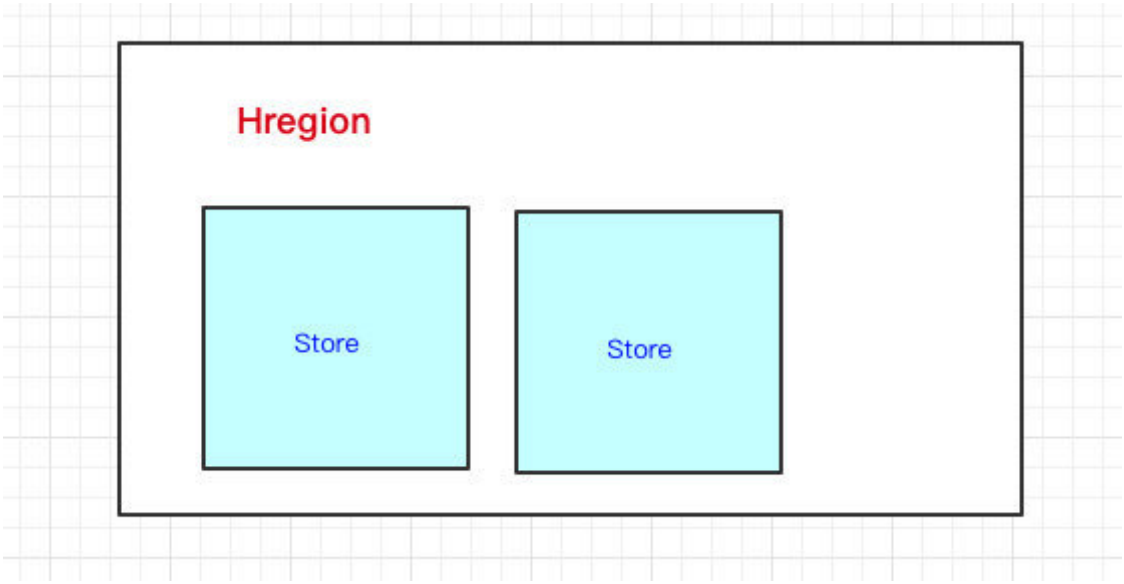
我们来看下面的图：



前面也提到了，HBase可以存储海量的数据，HBase是分布式的。所以我们可以断定：**HBase一张表的数据会分到多台机器上的**。那HBase是怎么切割一张表的数据的呢？用的就是**RowKey**来切分，其实就是表的**横向**切割。



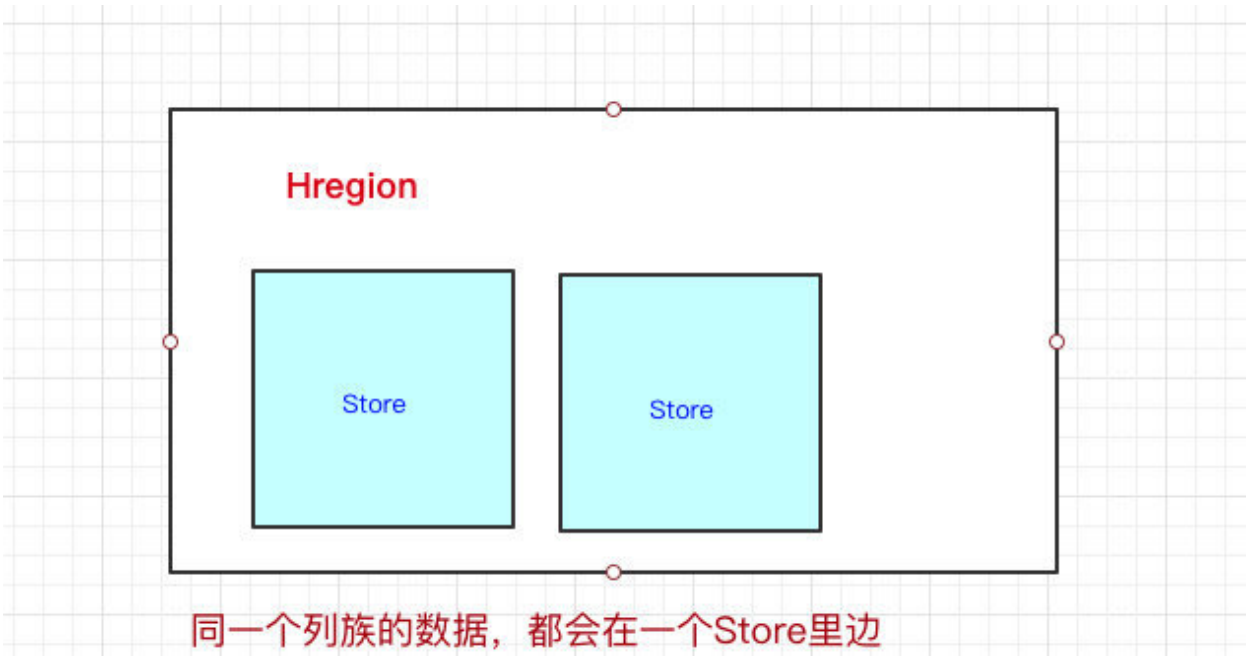
说白了就是一个**HRegion**上，存储HBase表的一部分数据。



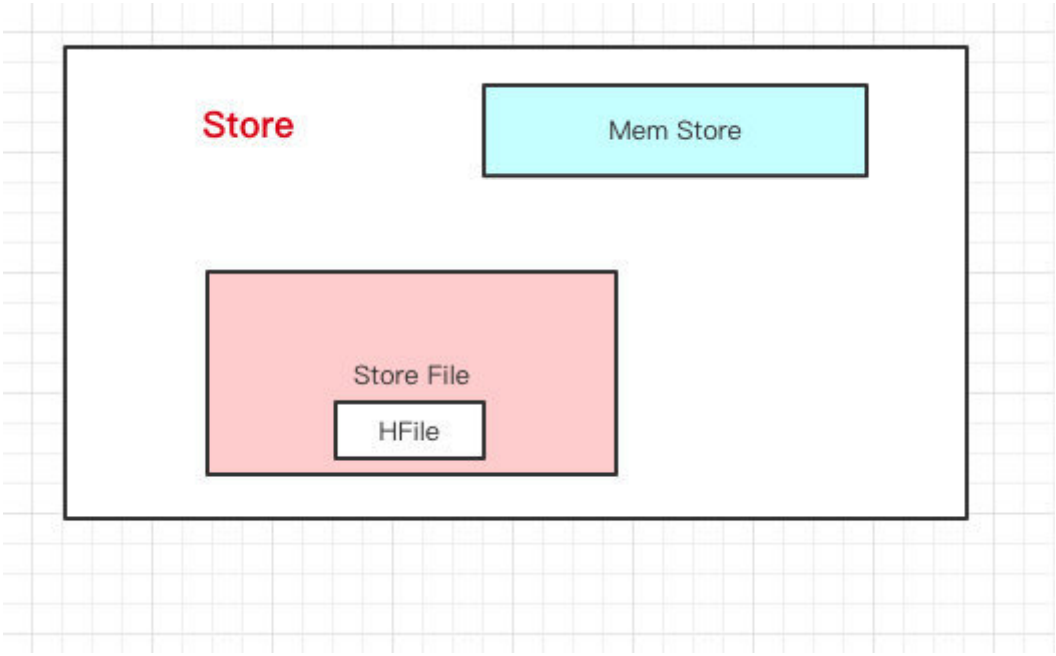
HRegion下面有**Store**，那Store是什么呢？我们前面也说过，一个HBase表首先要定义**列族**，然后列是在列族之下的，列可以随意添加。

一个列族的数据是存储在一起的，所以一个列族的数据是存储在一个Store里边的。

看到这里，其实我们可以认为HBase是基于列族存储的（毕竟物理存储，一个列族是存储到同一个Store里的）



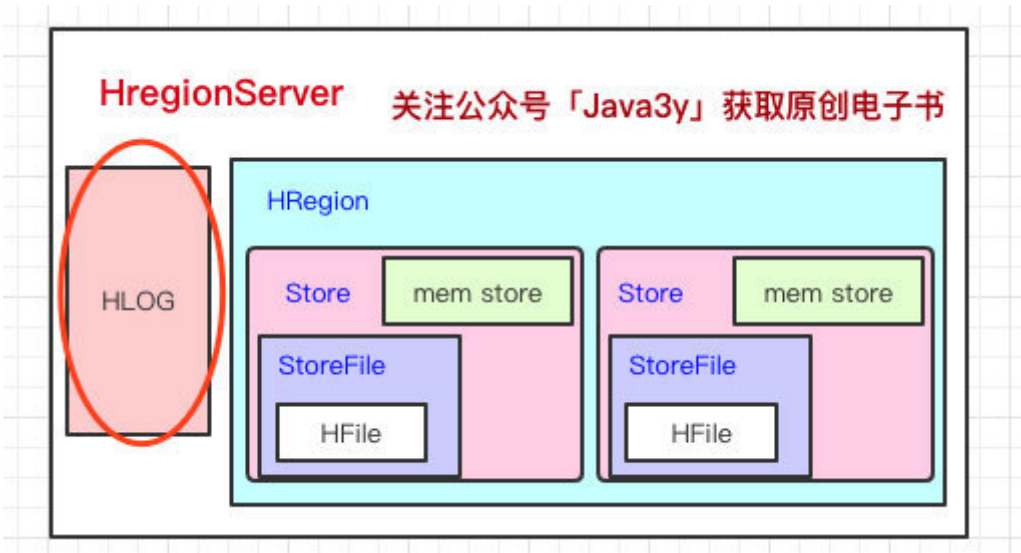
Store里边有啥？有`Mem Store`、`Store File`、`HFile`，我们再来看看里边都代表啥含义。



HBase在写数据的时候，会先写到Mem Store，当MemStore超过一定阈值，就会将内存中的数据刷写到硬盘上，形成**StoreFile**，而StoreFile底层是以HFile的格式保存，HFile是HBase中KeyValue数据的存储格式。

所以说：Mem Store我们可以理解为内存 buffer，HFile是HBase实际存储的数据格式，而StoreFile只是HBase里的一个名字。

回到HRegionServer上，我们还漏了一块，就是HLog。



这里其实特别好理解了，我们写数据的时候是先写到内存的，为了防止机器宕机，内存的数据没刷到磁盘中就挂了。我们在写Mem store的时候还会写一份HLog。

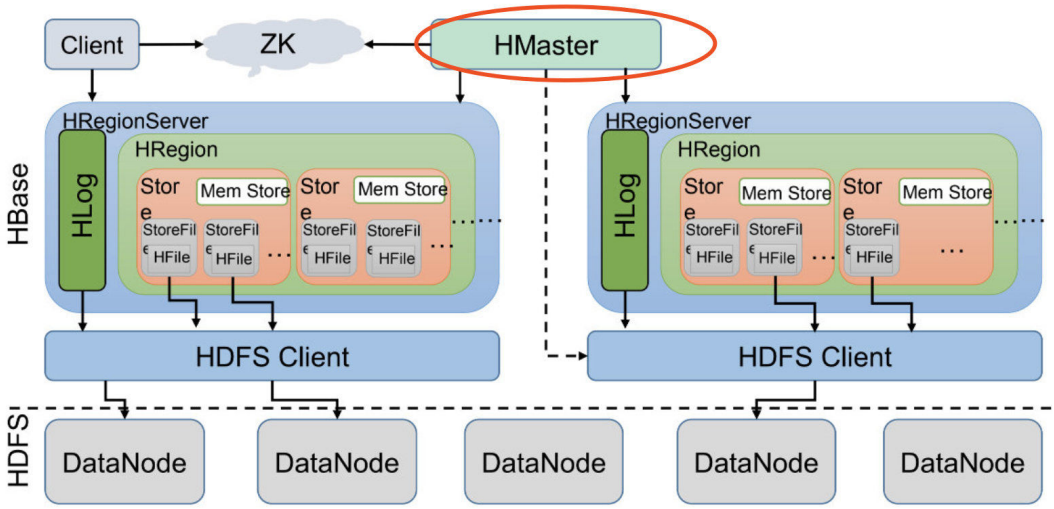
这个HLog是顺序写到磁盘的，所以速度还是挺快的（是不是有似曾相似的感觉）...

稍微总结一把：

- HRegionServer是真正干活的机器（用于与hdfs交互），我们HBase表用RowKey来横向切分表
- HRegion里边会有多个Store，每个Store其实就是一个列族的数据（所以我们可以说HBase是基于列族存储的）
- Store里边有Men Store和StoreFile(HFile)，其实就是先走一层内存，然后再刷到磁盘的结构

3.5 被遗忘的HMaster

我们在上面的图会看到有个**Hmaster**，它在HBase的架构中承担一种什么样的角色呢？**读写请求都没经过Hmaster呀**。



那HMaster在HBase里承担什么样的角色呢？？

> `HMaster` is the implementation of the Master Server. The Master server is responsible for monitoring all RegionServer instances in the cluster, and is the interface for all metadata changes.

HMaster会处理 HRegion 的分配或转移。如果我们HRegion的数据量太大的话，HMaster会对拆分后的Region**重新分配RegionServer**。（如果发现失效的HRegion，也会将失效的HRegion分配到正常的HRegionServer中）

HMaster会处理元数据的变更和监控RegionServer的状态。

四、RowKey的设计

到这里，我们已经知道RowKey是什么了。不难理解的是，我们肯定是要保证RowKey是**唯一**的，毕竟它是行键，有了它我们才可以唯一标识一条数据的。

在HBase里边提供了三种的查询方式：

1. 全局扫描
2. 根据一个RowKey进行查询
3. 根据RowKey过滤的**范围**查询

4.1 根据一个RowKey查询

首先我们要知道的是RowKey是会按**字典序**排序的，我们HBase表会用RowKey来横向切分表。

无论是读和写我们都是用RowKey去定位到HRegion，然后找到HRegionServer。这里有一个很关键的问题：**那我怎么知道这个RowKey是在这个HRegion上的**?

HRegion上有两个很重要的属性：`start-key`和`end-key`。

我们在定位HRegionServer的时候，实际上就是定位我们这个RowKey在不在这个HRegion的`start-key`和`end-key`范围之内，如果在，说明我们就找到了。

这个时候会带来一个问题：由于我们的RowKey是以字典序排序的，如果我们对RowKey没有做任何处理，那就有可能存在**热点数据**的问题。

举个例子，现在我们的RowKey如下：

```
java3y111 java3y222 java3y333 java3y444 java3y555 aaa bbb java3y777 java3y666 java3y...
```

`Java3yxxx`开头的RowKey很多，而其他的RowKey很少。如果我们有多个`HRegion`的话，那么存储`Java3yxxx`的HRegion的数据量是最大的，而分配给其他的HRegion数量是很少的。

关键是我们的查询也几乎都是以`java3yxxx`的数据去查，这会导致某部分数据会集中在某台HRegionServer上存储以及查询，而其他的HRegionServer却很空闲。

如果是这种情况，我们要做的是什么？对**RowKey散列**就好了，那分配到HRegion的时候就比较均匀，少了热点的问题。

HBase优化手册：

建表申请时的预分区设置，对于经常使用HBase的小伙伴来说,HBase管理平台里申请HBase表流程必然不陌生了。

'给定split的RowKey组例如:aaaaa,bbbbb,ccccc;或给定例如:startKey=00000000,endKey=xxxxxxxx,regionsNum=x'

第一种方式:

是自己指定RowKey的分割点来划分region个数.比如有一组数据RowKey为[1,2,3,4,5,6,7],此时给定split RowKey是3,6,那么就会划分为[1,3),[3,6),[6,7)的三个初始region了.如果对于RowKey的组成及数据分布非常清楚的话,可以使用这种方式精确预分区.

第二种方式：

如果只是知道RowKey的组成大致的范围,可以选用这种方式让集群来均衡预分区,设定始末的RowKey,以及根据数据量给定大致的region数,一般建议region数最多不要超过集群的rs节点数,过多region数不但不能增加表访问性能,反而会增加master节点压力.如果给定始末RowKey范围与实际偏差较大的话,还是比较容易产生数据热点问题.

最后:生成RowKey时,尽量进行加盐或者哈希的处理,这样很大程度上可以缓解数据热点问题.

4.2根据RowKey范围查询

上面的情况是针对通过RowKey单个查询的业务的，如果我们是根据RowKey范围查询的，那没必要上面那样做。

HBase将RowKey设计为字典序排序，如果不做限制，那很可能类似的RowKey存储在同一个HRegion中。那我正好有这个场景上的业务，那我查询的时候不是快多了吗？**在同一个HRegion就可以拿到我想要的数据了**。

举个例子：我们会间隔几秒就采集直播间热度，将这份数据写到HBase中，然后业务方经常要把主播的一段时间内的热度给查询出来。

我设计好的RowKey，将该主播的一段时间内的热度都写到同一个HRegion上，拉取的时候只要访问一个HRegionServer就可以得到全部我想要的数据了，那查询的速度就快很多。

最后

最后三歪再来带着大家回顾一下这篇文章写了什么：

1. HBase是一个NoSQL数据库，一般我们用它来存储海量的数据（因为它基于HDFS分布式文件系统上构建的）
2. HBase的一行记录由一个RowKey和一个或多个的列以及它的值所组成。先有列族后有列，列可以随意添加。
3. HBase的增删改记录都有「版本」，默认以时间戳的方式实现。
4. RowKey的设计如果没有特殊的业务性，最好设计为散列的，这样避免热点数据分布在同一个HRegionServer中。
5. HBase的读写都经过Zookeeper去拉取meta数据，定位到对应的HRegion，然后找到HRegionServer

参考资料：

- https://blog.csdn.net/bitcarmanlee/article/details/78979836
- https://hbase.apache.org/book.html#arch.overview
- https://zhuanlan.zhihu.com/p/54184168
- 硬核干货长文！Hbase来了解一下不?
- https://www.jianshu.com/p/569106a3008f
- 趣谈Hbase架构
- https://www.cnblogs.com/BIG-BOSS-ZC/p/11807304.html
- 一条数据的HBase之旅，简明HBase入门教程-开篇
- https://chenhy.com/post/hbase-quickstart/
- https://www.cnblogs.com/zmoumou/p/10292676.html
- https://www.cnblogs.com/duanxz/p/3154487.html
- https://www.jianshu.com/p/4e412f48e820
- HBase 基本入门篇
- 什么是列式存储?

涵盖Java后端所有知识点的开源项目（已有14K+ star）：

- GitHub
- Gitee访问更快

最近一直在连载《**对线面试官**》系列，目前已经连载25篇啦！一个**说人话**的系列！

- **【对线面试官】Java注解**
- **【对线面试官】Java泛型**
- **【对线面试官】 Java NIO**
- **【对线面试官】Java反射 && 动态代理**
- **【对线面试官】多线程基础**
- **【对线面试官】 CAS**
- **【对线面试官】synchronized**
- **【对线面试官】AQS&&ReentrantLock**
- **【对线面试官】线程池**
- **【对线面试官】ThreadLocal**
- **【对线面试官】CountDownLatch和CyclicBarrier**
- **【对线面试官】为什么需要Java内存模型?**

- 【对线面试官】List
- 【对线面试官】Map
- 【对线面试官】SpringMVC
- 【对线面试官】Spring基础
- 【对线面试官】SpringBean生命周期
- 【对线面试官】Redis基础
- 【对线面试官】Redis持久化
- 【对线面试官】Kafka基础
- 【对线面试官】使用Kafka会考虑什么问题?
- 【对线面试官】MySQL索引
- 【对线面试官】MySQL 事务&&锁机制&&MVCC
- 【对线面试官】MySQL调优