第7单HBase分布式数据库

(源自:https://biglab.site)

(版本:Ver1.0-20231116)

任务前言

任务背景

通信运营商的数据量存储量是非常多的，也是数据化较为完善的领域，数据价值也非常高。通信运营商每时每刻都在产生大量的通信数据，如查询当日话单、月度话单、季度话单、年度话单、通话详情、通话记录等。从通信运营商的数据中挖掘出有效信息，将数据转化为生产力是通信运营商非常紧迫的需求。现有某通信运营商保存的用户通话记录信息，其数据字段说明如下表所示。

字段名称	说明	字段名称	说明
rowkey	通话记录	length	通话时长（单位：分钟）
phoneNum	用户手机号码	type	通话类型（1表示主叫，0表示被叫）
dnum	对方用户手机号码	date	通话时间

对于运营商而言，对客户的通话记录进行分析，了解用户通话行为，可以为制订科学合理的手机套餐提供策略支持。为了保证较高的处理效率与灵活性，将选择使用HBase分布式数据库存储通话记录，并采用HBase Java API实现通话记录数据的查询分析

本章介绍内容如下。

1. HBase分布式数据仓库的系统架构、数据模型、读写流程。
2. HBase的安装配置过程及HBase常用的shell命令。
3. 接着将重点介绍HBase的Java API实现表创建、表数据导入。
4. 通过编写HBase的Java API实现通话记录数据表的创建与查询分析。

任务7.1认识HBase分布式数据库

任务描述

与MapReduce的离线批处理计算框架不同，HBase是一个可以随机访问的、用于存储和检索数据的框架，弥补了HDFS不能随机访问数据的缺陷。

HBase适合实时性要求不高的业务场景，HBase中的数据以Byte[]数组的方式存储，不区分数据类型，支持结构化、半结构化、非结构化数据，数据模型动态性强，十分灵活。

为了使读者对HBase有更加深入的理解，本小节的任务是从HBase的起源出发，了解HBase的特点、HBase与传统数据库的区别、系统架构、数据模型及读写流程

什么是HBase

BigTable是一个分布式存储系统，利用基于C语言的MapReduce分布式计算模型处理少量数据，使用GFS分布式文件系统作为底层数据存储方式，具备广泛应用性、可靠性、高性能和高可用性等特点。

HBase是BigTable的开源实现

HBase的起源

HBase属于Apache旗下的一个顶级开源项目，项目灵感起源于2006年BigTable论文的发表。HBase是对BigTable的开源实现，但两者底层使用的技术还是存在差别，具体如下表所示。

对比内容	BigTable	HBase
文件存储系统	GFS	HDFS
海量数据处理	基于C语言的MapReduce	基于Java语言的MapReduce
协同服务管理	Chubby	ZooKeeper

HBase是一个基于Java、开源的、高可靠、高性能、面向列、可伸缩的列式非关系型数据库，也可以称为列式分布式数据库，或简称为分布式数据库。

HBase的目标是存储并处理海量的非结构化和半结构化的松散数据，旨在仅使用普通的硬件配置，即可处理由成千上万的行和列组成的海量数据。

HBase的特点

- 海量存储。HBase通过多台廉价的机器实现存储PB级别的海量数据，并且可以在几十毫秒或几百毫秒内返回数据。
- 面向列。HBase面向列进行存储和权限控制，并支持独立检索。HBase是根据列族存储数据的，一个列族下可以有多列，列族在创建表时必须指定，并且可以单独对列进行各种操作。
- 多版本。HBase中表的每一个列的数据存储都有多个Version（版本，即同一条数据插入不同的时间戳）。虽然每一列对应着一条数据，但是有的数据会对应多个版本。例如，存储个人信息的HBase表中，如果某个人多次更换过家庭住址，那么记录家庭住址的数据会将有多个版本。
- 稀疏性。HBase的稀疏性主要体现出HBase列的灵活性。在列族中，可以指定任意多个列，在列数据为空的情况下，是不会占用存储空间的。
- 易扩展性。HBase的扩展性主要体现在两个方面，一个是基于上层处理能力（RegionServer）的扩展，一个是基于存储的扩展（HDFS）。HBase的底层依赖HDFS，当磁盘空间不足时，可以动态增加机器（即DataNode节点服务）解决，从而避免进行数据的迁移。
- 高可靠性。由于HBase底层使用的是HDFS，而HDFS的分布式集群具有备份机制，副本（Replication）机制能够保证数据不会发生丢失或损坏。

HBase与传统数据库的区别

1. 数据类型：关系型数据库采用关系模型，具有丰富的数据类型和存储方式。HBase则采用了更加简单灵活的数据模型，将数据存储为未经解释的字符串，用户可以将不同格式的结构化数据和非结构化数据都序列化成字符串保存至HBase中，用户再需要编写程序将字符串解析成不同的数据类型。
2. 数据操作：关系型数据库中提供了丰富的操作，如插入、删除、更新、查询等，一些操作会涉及复杂的多表连接，通常需要借助于多个表之间的主外键关联实现。HBase提供的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。
3. 存储模式：关系型数据库是基于行模式存储的，元组或行会被连续地存储在磁盘中。在读取数据时，需要顺序扫描每个元组，再从中筛选查询出所需要的数据。如果每个元组只有少量字段的值且查询操作是有用的，那么基于行模式存储会浪费许多磁盘空间和内存带宽。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，优点是可以降低I/O开销，支持大量并发用户查询（因为仅需要处理与

查询结果相关的列，而不需要处理与查询无关的大量数据行）；同一个列族中的数据会被一起压缩（同一列族内的数据相似度较高，因此可以获得较高的数据压缩比）

4. 数据索引：关系型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。与关系型数据库不同的是，HBase只有一个索引——行键。由于HBase位于Hadoop框架之上，所以可以使用Hadoop MapReduce快速、高效地生成索引表。
5. 数据维护：在关系型数据库中，更新操作会用最新的当前值去替换记录中原来的值（旧值），旧值被覆盖后就不会存在。而在HBase中执行更新操作时，并不会删除旧版本的数据，而是生成一个新版本的数据，旧版本的数据仍然保留。
6. 可伸缩性：关系型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分布式数据库是为了实现灵活的横向扩展而开发的，因此能够轻易地通过在集群中增加或减少硬件数量实现性能的伸缩。但是，相对关系型数据库，HBase也有自身的局限性，如HBase不支持事务，因此无法实现跨行的原子性

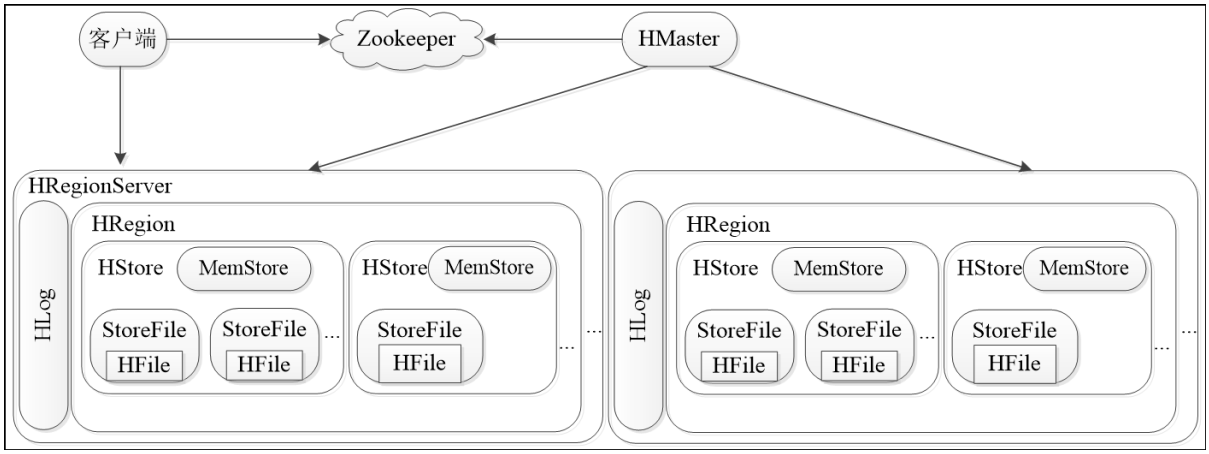
了解HBase系统架构

HBase采用Master/Slave架构搭建集群，属于Hadoop生态系统的组件，由Client（客户端）、HMaster、HRegionServer、ZooKeeper等部分组成，并将数据存储于HDFS中。

HMaster主要负责利用ZooKeeper为HRegionServer分配HRegion。ZooKeeper是一个高可靠、高可用、持久化的分布式协调系统。

客户端使用HBase的远程过程调用协议（RemoteProcedure Call Protocol，RPC）机制与HMaster和HRegionServer进行通信，对于管理类操作，客户端与HMaster进行RPC通信；对于数据读/写类操作，客户端与HRegionServer进行RPC通信。

HBase的架构：



ZooKeeper

1. ZooKeeper是一个开放源代码的分布式应用程序协调服务，是Hadoop和HBase的重要组件。
2. 分布式HBase依赖于ZooKeeper集群，所有节点和客户端必须能够正常访问ZooKeeper。
3. HBase默认管理一个单点的ZooKeeper集群，HBase可以将ZooKeeper当作自身的一部分启动和关闭进程。
4. HMaster启动时将HBase系统表加载至ZooKeeper，通过ZooKeeper可以获取当前系统表元数据的存储所对应的HRegionServer信息。

HMaster

1. HBase中可以启动多个HMaster，通过ZooKeeper的Master选举机制保证总有一个HMaster运行。
2. HMaster管理HRegionServer的负载均衡，调整HRegion的分布，如在Region分片后，负责将HRegion分配至HRegionserver，在HRegionserver宕机后，HMaster会将HRegionserver内的HRegion迁移至其他HRegionserver上。

HRegionServer

1. HRegionServer主要负责响应用户I/O请求，在HDFS文件系统中读/写数据，是HBase中核心的模块。HRegionServer内部管理了一系列HRegion，HRegion对应了Table中的一个Region，HRegion由多个HStore组成。每个HStore对应了HBase表中的一个Column Family的存储，每个Column Family指的是一个集中的存储单元，因此最好将具备共同I/O特性的列放在一个Column Family中，查询会更加高效。
2. HStore存储是HBase存储的核心，由两部分组成，一部分是MemStore缓冲区；另一部分是StoreFile文件。用户写入的数据首先会放入MemStore，当MemStore满了以后会刷新成一个StoreFile（底层实现是HFile）；当StoreFile文件数量增长到一定阈值时，将触发Compact合并操作，将多个StoreFile合并为一个StoreFile，合并过程中会进行版本合并和数据删除。
3. 可以看出HBase其实只有增加数据，所有的更新和删除操作都是在后续的Compact过程中进行的，这使得用户的写操作只要进入内存中即可立即返回，保证了HBase I/O的高性能。
4. 当单个StoreFile大小超过一定阈值后，会触发Split分割操作，同时把当前Region分片为两个Region，父Region将下线，新分割出的两个子Region会被HMaster分配到相应的HRegionServer上，使得原先一个Region的压力得以分流至两个Region上。
5. 实际的存储文件功能是由HFile实现的。HFile被专门创建用于有效存储HBase数据，基于Hadoop的TFile类

HLog

1. Write-Ahead-Log（WAL）是HBase的HRegionServer在处理数据插入和删除的过程中用于记录操作内容的一种日志。客户端初始化一个更改数据的动作，每一种改动都被包装进KeyValue对象使用远程调用发送到ReginoServer对应这次改动的Region中。
2. 数据在HRegionServer中首先被写入WAL，再被写入MemStore，最后当MemStore达到一定的大小或到达指定的时刻之后，数据被异步地持久化到文件系统上，在这之前数据是存储在内存中的，在这段时间里如果HRegionServer崩溃了，那么内存的数据就没有了，在这种情况下如果有WAL，那么即可恢复数据
3. 每个HRegionServer中都有一个HLog对象，HLog是一个实现WAL的类，在每次用户操作写入MemStore的同时，也会写一份数据到HLog文件中，HLog文件定期会滚动出新的文件，并删除旧的文件（已持久化到StoreFile中的数据）。
4. 当HRegionServer意外终止后，HMaster会通过ZooKeeper感知HMaster首先会处理遗留的HLog文件，将其中不同Region的Log数据进行拆分，分别放到相应的Region目录下，再将失效的Region重新分配给HRegionServer。HRegionServer在加载Region的过程中，将发现有历史HLog需要处理，因此会重置HLog中的数据到MemStore中，最后刷新至StoreFile文件中完成数据恢复。

了解HBase数据模型

HBase实际上是一个稀疏、多维、持久化存储的映射表，采用行键（Row Key）、列族（Column Family）、列限定符（Column Qualifier）和时间戳（Timestamp）进行索引，每个值都是未经解释的字节组成byte[]，没有数据类型数据模型，具体如下表所示

Row Key	**Timestamp	Column Family:c1	Column Family:c1	Column Family:c2	Column Family:c2	Column Family:c3	Column Family:c3
		Column	Value	Column	Value	Column	Value
r1	t7	c1:col-1	value-1	/	/	c3:col-1	value-1
/	t6	c1:col-2	value-2	/	/	c3:col-2	value-2
/	t5	c1:col-3	value-3	/	/	/	/
/	t4	/	/	/	/	/	/
r2	t3	c1:col-1	value-1	c2:col-1	value-1	c2:col-1	value1
/	t2	c1:col-2	value-2	/	/	/	/
/	t1	c1:col-3	value-3	/	/	/	/

- 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。列族支持动态扩展，可以很轻松地添加一个列族或列，无须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。由于同一张表里面的每一行数据都可以有截然不同的列，对于整个映射表的每行数据而言，有些列的值可以是空的，因此HBase是稀疏的。
- 在HBase中执行更新操作时，会生成一个新的版本，旧的版本仍然保留，HBase可以对允许保留的版本的数量进行设置。数据在存储时是按照时间戳排序的，客户端可以选择获取距离某个时间最近的版本，或一次获取所有版本。如果在查询的时候不提供时间戳，那么会返回距离现在最近的一个版本的数据。HBase提供了两种数据版本回收方式，一是保存数据的最后n个版本；二是保存最近一段时间内的版本（如最近7天）。

行键（Row Key）

1. Row Key表示行键，每个HBase表中只能有一个行键，类似于主键。由于Row Key是HBase表的唯一标识，因此Row Key的设计非常重要。数据的存储规则是相近的数据存储在一起。
2. 例如，Row Key格式为www.cqyti.com、sxy.cqyti.com、dsj.cqyti.com和zngc.cqyti.com的网站名称时，可以将网站名称进行反转，反转后为com.cqyti.www、com.cqyti.sxy、com.cqyti.dsj和com.cqyti.zngc，再进行存储，所有com.cqyti域名将会存储在一起，避免子域名(即www，sxy、dsj、zngc)分散在各处。

列族（Column Family）

在HBase中，列族由一个或多个列组成。HBase会尽量把同一个列族的列放在同一个服务器上，可以提高读写数据的性能，并且可以批量管理多个有关联的列。HBase中数据的属性均是定义在列族上，同一个列族内的所有列具有相同的属性。在HBase中创建数据表时，定义的是列族，而不是列。在表中，cl、c2、c3均为列族名

列名（Column

HBase表的列是由列族名、限定符以及列名组成，如列名c1:col-1，其中，“c1”为列族名，“:”为限定符，“col-1”为列名。创建HBase表不需要指定列，因为列是可变的，非常灵活

时间戳（Timestamp）

1. 在HBase表中，通过行键、列族和列名确定一个单元格（Cell）。单元格中存储的数据没有数据类型，被视为byte[]字节数组。每个单元格都保存着同一份数据的多个版本，每个版本对应一个不同的时间戳。每次对一个单元格执行操作（新建、修改、删除）时，HBase将隐式地自动生成并存储一个时间戳。
2. 时间戳一般是64位整型数据，可以由用户自己赋值（自己生成唯一时间戳可以避免应用程序中出现数据版本冲突），也可以由HBase在数据写入时自动赋值。一个单元格的不同版本根据时间戳进行降序存储，因此，最新版本的数据可以被优先读取，通常将记录每次操作数据的时间戳记作数据的版本号。

了解HBase读写流程

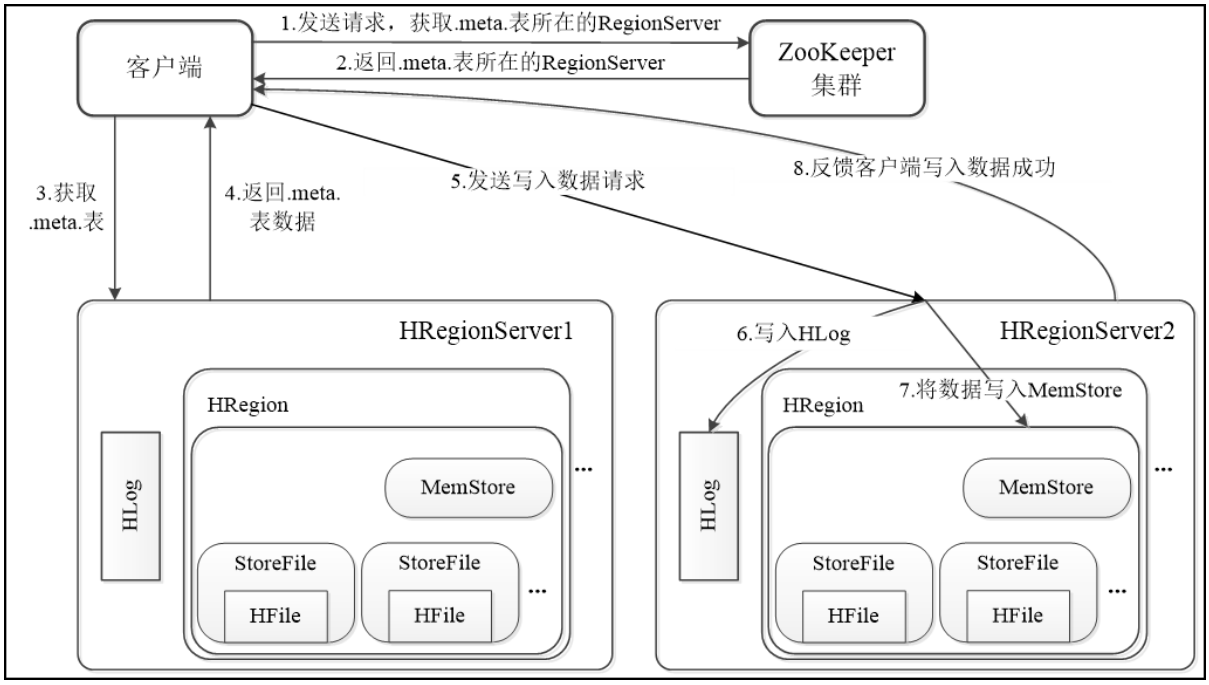
当对HBase进行读写操作时，需要提前知道客户端需要操作的Region的所在位置，即Region存在于哪个HRegionServer上，因此在HBase中存在一张表，元数据表（即.meta.表）元数据表属于HBase的内置表，专门存储了表的元数据信息，以及Region位于哪个Regionserver上。.meta.表的结构如下表所示。

RowKey	Info	Info	Info	Historian
	Regioninfo	Server	Serverstartcode	
TableName,startKey,Timestamp	StartKey, EndKey,Family List{Family,B1oomFilter,Compress, TTL,InMemory,BlockSize,B1ockCache)	Region的Regionserver地址		

- meta.表的RowKey由3部分组成：TableName（表名）、StartKey（起始键）和TimeStamp（时间戳）。TimeStamp使用十进制的数字字符串进行表示。将组成RowKey的3个部分用逗号分隔组成了整个完整的RowKey。
- .meta.表的Info为表最主要的列族，含有3个列信息，包括Regioninfo、Server、Serverstartcode。Regioninfo存储的是Region的详细信息，包括StartKey 、EndKey和每个Family信息。Server存储的是管理该Region的Regionserver地址。Serverstartcode存储RegionServer开始托管该Region的时间。
- .meta.表的historian是用来追踪记录一些Region操作的，如open、close、compact等操作。
- 由于.meta.存储的是Region的信息，如果当HBase中表的数据非常大会被分成很多个Region，那么此时Region信息在.meta.表中所占的空间也会变大，而.meta.本身也是一张表，在存储数据非常大的情况下，也会被分割成多个Region存储于不同的Regionserver上，此时如果将.meta.表的Region位置信息存储在ZooKeeper集群中那么可行性较低。
- .meta.表Region的位置信息是会发生变化的，可以通过另外一张表存储.meta.表的元数据信息，根数据表（即-ROOT-表），-ROOT-只会有一个Region，这个Region的信息存在于HBase中，而管理-ROOT-表的位置信息（Regionserver地址）存储在ZooKeeper中。

写流程

当用户向HRegionServer发起HTable.put请求，即写入数据请求时，会将请求交给对应的HRegion实例处理，具体流程如图所示

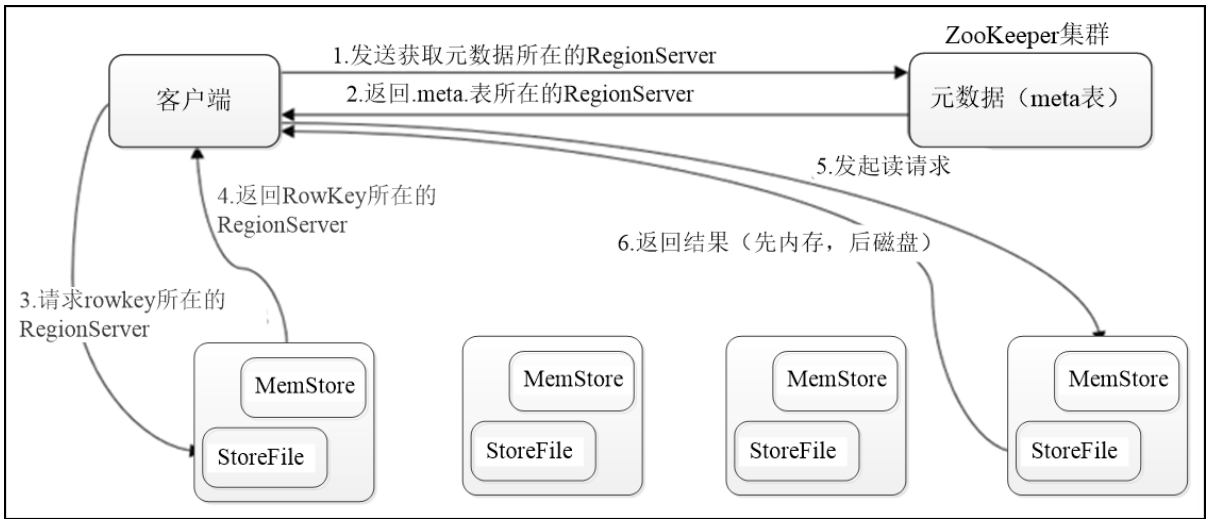


1. 根据图可知，客户端访问ZooKeeper集群，查询-ROOT-表Region所在的RegionServer地址信息。如RS1，客户端将连接RS1，访问-ROOT-表，根据写入信息查询.meta.表的Region所在的Regionserver地址信息，将得到的结果返回给客户端。

2. 客户端将连接相应的RS，访问.meta.表，根据写入的NameSpace（命名空间）、表名和RowKey找到对应的Region信息；为了持久化和恢复，将数据先写至HLog中，再将数据写入至MemStore，当MemStore达到预设阈值后，将创建一个新的MemStore，而旧的MemStore就会加入Flush队列，由单独的线程Flush至磁盘上，形成一个StoreFile。
3. 同时，系统将在ZooKeeper集群中记录一个CheckPoint（检查点），表示该时刻前的数据变更已经持久化了，当系统出现意外可能导致MemStore中的数据丢失时，即可通过HLog恢复CheckPoint之后的数据；StoreFile文件是只读的，一旦创建后不可修改，因此HBase的更新是不断追加的操作。

读流程

当用户向HRegionServer发起读取数据请求时，会将请求交给对应的HRegion实例处理，具体流程如图所示:



1. 客户端访问ZooKeeper集群，查找元数据表在哪个RegionServer上，并访问对应RegionServer上元数据表中的数据，查找要操作RowKey所在表对应Region所在的RegionServer。接着客户端将读取RegionServer上的Region数据。
2. 客户端定位到真正的数据所在的Region时，先从MemStore进行查找，如果MEMstore没有，那么再从块缓存查找；如果块缓存没有，那么再从StoreFile查找数据，查找到数据后同时将数据进行缓存

任务7.2安装部署HBase集群

任务描述

HBase是Hadoop生态系统中的一个组件，但是Hadoop集群安装后，本身并不包含HBase，为了更好的使用HBase，还需要单独安装HBase。

本小节的任务主要是

1. ZooKeeper原理。
2. ZooKeeper与HBase之间的关系。
3. ZooKeeper的安装配置过程。
4. HBase集群的安装配置。

了解并安装ZooKeeper

在HBase集群中，包含了一个Master主服务器和多个Region服务器。

Master即为HBase集群的总管，Master必须知道Region服务器的状态。ZooKeeper服务器可以帮助维护当前的集群中机器的服务状态。HBase需要使用ZooKeeper提供的稳定、可靠的协同服务，帮助维护集群中各节点的服务状态

ZooKeeper介绍

1. ZooKeeper起源于雅虎，当时雅虎的研究人员发现，在雅虎内部很多大型系统基本都需要依赖一个类似的系统来进行分布式协调，但是这些系统往往都会存在分布式单点问题。
2. 单点问题是指在整个分布式系统中，如果某个独立功能的程序或角色只运行在某一台服务器上时，那么这个节点就被称为单点。
3. 雅虎的开发人员试图开发一个通用的无单点问题的分布式协调框架，以便让开发人员将精力集中在处理业务逻辑上，因此开发了ZooKeeper分布式协调服务组件。
4. ZooKeeper是一个分布式协调服务的开源框架，主要用于解决分布式集群中应用系统的一致性问题，如避免同时操作同一数据造成数据无效读取及操作的问题等。

ZooKeeper集群角色

1. ZooKeeper本质上是一个分布式的小文件存储系统，如统一命名服务、分布式配置管理、分布式协调、负载均衡、分布式协调通知、集群管理、Master选举、分布式消息队列、分布式锁等功能。只要所有服务器节点超过半数可用，整个ZooKeeper集群即可用。
2. ZooKeeper可以保证分布式顺序一致性、单一视图、可靠性和实时性。
3. ZooKeeper集群是一个主从集群结构，一般由一个Leader（领导者）和多个Follower（跟随者）组成，此外，针对访问量较大的ZooKeeper集群，还可新增Observer（观察者）。
4. ZooKeeper集群中的所有节点通过一个Leader选举过程选定一个节点为Leader，为客户端提供读和写服务；Follower和Observer则提供读服务，Observer不参与Leader选举过程

ZooKeeper角色如下表所示：

角色		描述
领导者（Leader）		领导者负责进行投票的发起和决议，更新系统状态
学习者（Learner）	跟随者（Follower）	跟随者用于接收客户请求并向客户端返回结果，在选举过程中参与投票
	观察者（Observer）	观察者可以接收客户端连接，将写请求转发给领导者节点。但观察者不参与投票过程，只同步领导者的状态，其目的是扩展系统、提高读取速度
客户端		请求发起方

ZooKeeper集群中的3种角色各司其职，共同完成分布式协调服务，具体说明如下。

1. Leader是ZooKeeper集群工作的核心，也是事务性请求（写操作）的唯一调试和处理者，保证集群事务处理的顺序性，同时负责进行投票的发起和决议，以及更新系统状态。
2. Follower负责处理客户端的非事务（读操作）请求，如果接收到客户端发来的事务性请求，那么会转发给Leader，让Leader进行处理，同时还负责Leader选举过程中参与投票。
3. Observer负责观察ZooKeeper集群的最新状态的变化，并且将这些状态进行同步。对于非事务性请求可以进行独立处理，对于事务性请求，将会转发给Leader进行处理。

ZooKeeper选举机制

ZooKeeper为了保证各节点的协同工作，在工作时需要一个Leader角色，而ZooKeeper默认采用FastLeaderElection算法，且投票数大于半数则胜出的机制，在介绍选举机制前，首先了解选举涉及的相关概念。

1. 服务器ID。在配置集群时设置的myid参数文件，且参数分别表示为服务器1，服务器2和服务器3，编号越大在FastLeaderElection算法中的权重越大。
2. 选举状态。在选举过程中，ZooKeeper服务器有4种状态，分别为竞选状态（LOKING）、随从状态（FOLLOWING，同步Leader状态，参与投票）、观察状态（OBSERVIONG，同步Leader状态，不参与投

票）和领导者状态（LEADING）
3. 数据ID。这是服务器中存放的最新数据的版本号，该值越大说明数据越新，在选举过程中数据越新权重越大。
4. 逻辑时钟。逻辑时钟被称为投票次数。同一轮投票过程中的逻辑时钟值是相同的，逻辑时钟起始值为0，每投完一次票，逻辑时钟将会增加。逻辑时钟与接收到其他服务器返回的投票信息中的数值相比较，根据不同的值做出不同的判断，如果某台机器宕机，那么这台机器不会参与投票，因此逻辑时钟也会比其他的低。

ZooKeeper分布式集群安装部署

ZooKeeper分布式集群部署指的是ZooKeeper分布式模式安装。

ZooKeeper集群搭建通常是由2n+1台服务器组成的，这是为了保证Leader选举能够通过半数以上服务器选举支持，因此，ZooKeeper集群的节点数量为奇数。

在master上下载安装zookeeper

```
cd /root
wget -c \
https://repo.huaweicloud.com/apache/zookeeper/zookeeper-3.7.1/apache-zookeeper-3.7.1-bin.tar.gz \
--no-check-certificate
tar -xzvf ./apache-zookeeper-3.7.1-bin.tar.gz
mv ./apache-zookeeper-3.7.1-bin /usr/local/zookeeper
mkdir -p /home/hadoop/zookeeper/data
cd /usr/local/zookeeper/conf
cp ./zoo_sample.cfg ./zoo.cfg
echo "1" > /home/hadoop/zookeeper/data/myid
vi ./zoo.cfg
```

修改zoo.cfg内容如下

```
#修改行:
dataDir=/home/hadoop/zookeeper/data
#在最后添加
server.1=master:2888:3888
server.2=slave1:2888:3888
server.3=slave2:2888:3888
forceSync=no
zookeeper.client.sasl=false
```

修改/etc/profile

```
vi /etc/profile
```

添加内容如下:

```
export ZOOKEEPER_HOME=/usr/local/zookeeper
export PATH=$PATH:$ZOOKEEPER_HOME/bin
```

立即生效/etc/profile

```
source /etc/profile
```

分发zookeeper到从机

```
scp -r /usr/local/zookeeper/ slave1:/usr/local/
scp -r /usr/local/zookeeper/ slave2:/usr/local/
scp /etc/profile slave1:/etc/
scp /etc/profile slave2:/etc/

```

切换到slave1上执行

```
source /etc/profile
mkdir -p /home/hadoop/zookeeper/data
echo "2" > /home/hadoop/zookeeper/data/myid

```

切换到slave2上执行

```
source /etc/profile
mkdir -p /home/hadoop/zookeeper/data
echo "3" > /home/hadoop/zookeeper/data/myid

```

启动ZooKeeper服务

切换到master和执行

```
cd /usr/local/zookeeper/bin
vi ./zkstart-all.sh
vi ./zkstop-all.sh
chmod 755 ./zkstart-all.sh
chmod 755 ./zkstop-all.sh
```

zkstart-all.sh的内容如下

```
#!/bin/bash
echo “starting ZooKeeper cluster ......”
echo “starting ZooKeeper master ......”
ssh master "/usr/local/zookeeper/bin/zkServer.sh start"
echo “starting ZooKeeper slave1 ......”
ssh slave1 "/usr/local/zookeeper/bin/zkServer.sh start"
echo “starting ZooKeeper slave2 ......”
ssh slave2 "/usr/local/zookeeper/bin/zkServer.sh start"

```

zkstop-all.sh的内容如下：

```
#!/bin/bash
echo “stopping ZooKeeper cluster ......”
echo “stopping ZooKeeper master ......”
ssh master "/usr/local/zookeeper/bin/zkServer.sh stop"
echo “stopping ZooKeeper slave1 ......”
ssh slave1 "/usr/local/zookeeper/bin/zkServer.sh stop"
echo “stopping ZooKeeper slave2 ......”
ssh slave2 "/usr/local/zookeeper/bin/zkServer.sh stop"

```

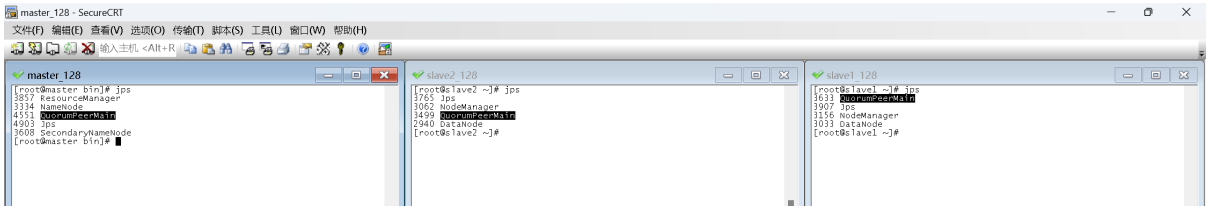
启动ZooKeeper集群

```
zkstart-all.sh
```

结果如

```
[root@master bin]# zkstart-all.sh
“starting ZooKeeper cluster ......”
“starting ZooKeeper master ......”
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
“starting ZooKeeper slave1 ......”
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
“starting ZooKeeper slave2 ......”
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@master bin]#
```

使用jps查看进程如，主从机都将看到新增加的“QuorumPeerMain”进程



安装及配置HBase集群

HBase集群一般由一个HBase节点和多个HRegionServer节点组成，ZooKeeper使用7.2.3小节中所配置的分布式集群，因此，需要将HBase自带的ZooKeeper进行屏蔽。

由于HBase集群的运行需要Java及Hadoop环境支持，所以需要提前安装JDK1.8和Hadoop3.1.4。从HBase的官网下载HBase安装包，本书使用的HBase版本为HBase 2.2.2。将下载的HBase安装包上传至Linux系统的/root目录下。

在master上下载并安装hbase

```
cd /root
wget -c https://mirrors.huaweicloud.com/apache/hbase/2.4.17/hbase-2.4.17-bin.tar.gz --no-check-certificate
tar -xzvf ./hbase-2.4.17-bin.tar.gz
mv ./hbase-2.4.17 /usr/local/hbase

```

修改/etc/profile

```
vi /etc/profile
```

添加内容如下:

```
export HBASE_HOME=/usr/local/hbase
export PATH=$PATH:$HBASE_HOME/bin
```

立即生效/etc/profile

```
source /etc/profile
```

修改hbase-env.sh

```
 cd /usr/local/hbase/conf
 vi ./hbase-env.sh
```

在hbase-env.sh文件末尾添加如下:

```
export HBASE_MANAGES_ZK=false
export JAVA_HOME=/usr/java/jdk1.8.0_281-amd64
```

修改hbase-site.xml

```
 mkdir -p /home/hadoop/hbase/zookeeper
 hdfs dfs -mkdir -p /data/hbase_db
 cd /usr/local/hbase/conf
 vi ./hbase-site.xml
```

修改hbase-site.xml内容中下:

```
<configuration>
<property>
<name>hbase.rootdir</name>
<value>hdfs://master:8020/data/hbase_db</value>
</property>
<property>
<name>hbase.master</name>
<value>master</value>
</property>
<property>
<name>hbase.cluster.distributed</name>

```

```
<value>true</value>
</property>
<property>
<name>dfs.support.append</name>
<value>true</value>
</property>
<property>
<name>hbase.zookeeper.quorum</name>
<value>master,slave1,slave2</value>
</property>
<property>
<name>hbase.zookeeper.property.datadir</name>
<value>/home/hadoop/hbase/zookeeper</value>
</property>
<property>
<name>hbase.zookeeper.property.clientPort</name>
<value>2181</value>
</property>
</configuration>
```

修改regionservers

```
 cd /usr/local/hbase/conf
 vi ./regionservers
```

修改regionservers 内容如下

```
master
slave1
slave2
```

分发hbase到从机

```
scp -r /usr/local/hbase/ slave1:/usr/local/
scp -r /usr/local/hbase/ slave2:/usr/local/
scp /etc/profile slave1:/etc/
scp /etc/profile slave2:/etc/
```

切换到slave1上执行

```
source /etc/profile
```

切换到slave2上执行

```
source /etc/profile
```

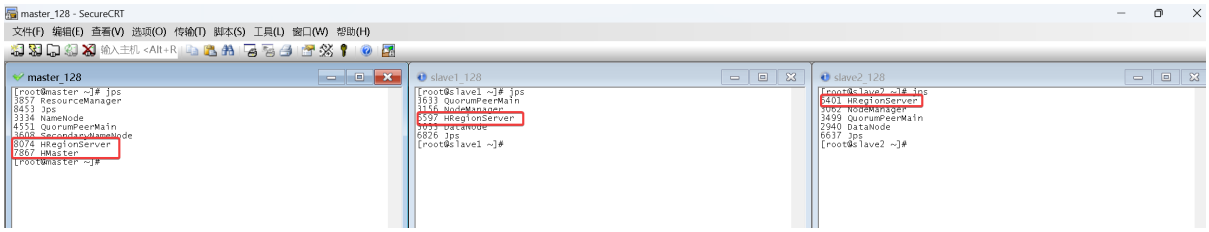
启动HBase集群

在master主机上执行:

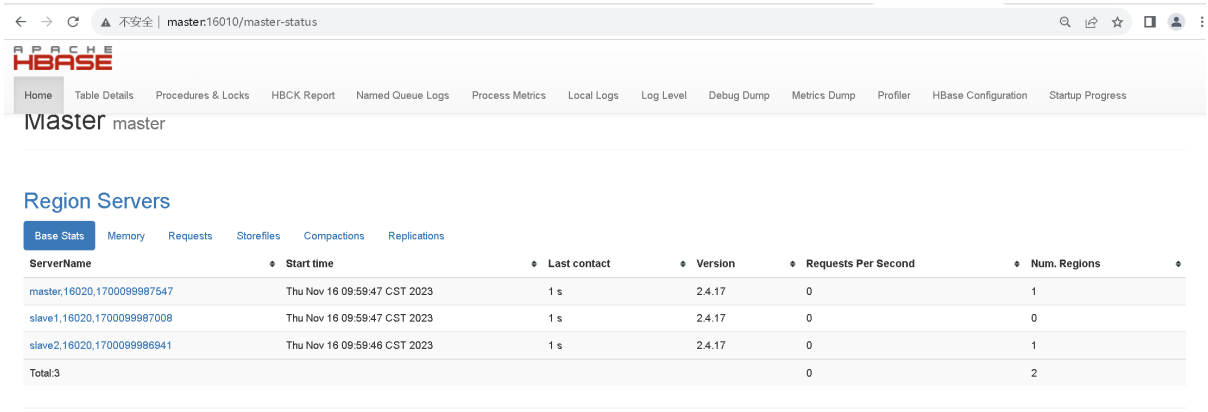
```
start-hbase.sh
```

检查HBase集群

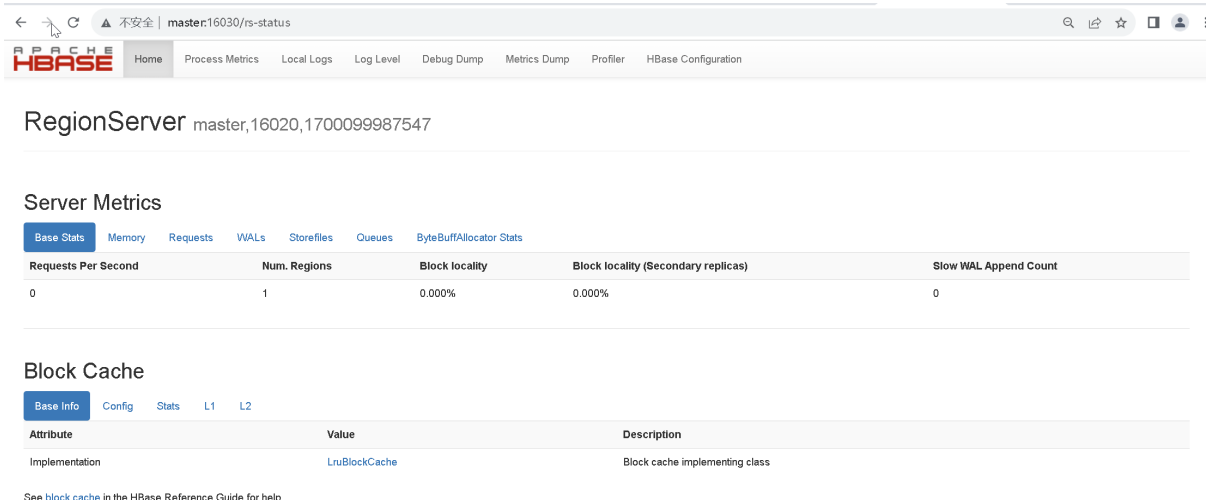
使用jps检查 进程，正常结果如：



访问master:16010,查看Master信息如：



访问master:16030,查看RegionServer信息如：



任务7.3掌握HBase常用的Shell命令

HBase的Shell提供了大量操作HBase表的命令。通过Shell命令可以很方便地操作HBase数据库中的表，如创建、删除及修改表、向表中添加数据、列出表中的相关信息等操作，具体表格如下所示

命令	描述
create	创建表
alter	修改列族（column Family）
describe	显示表详细信息
list	列出HBase中存在的所有表
count	统计表中行的数量
put	向指向的表单元添加值
get	获取行或单元（Cell）的值
delete	指定对象的值（可以为表、行、列对应的值，另外也可以指定时间戳的值）
incr	增加指定表的行或列的值
scan	通过对表的扫描来获取单元信息
truncate	重新创建指定表
disable	使表无效
enable	使表有效
exists	测试表是否存在
drop	删除表
tools	列出HBase所支持的工具
status	返回HBase集群的状态信息
shutdown	关闭HBase集群
version	返回HBase的版本信息
exit	退出HBase Shell

进入hbase的shell模式

在master主机或slave从机上执行

```
hbase shell
```

进入shell后结果如:

```
[root@master ~]# hbase shell
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hbase/lib/client-facing-thirdparty/slf4j-reload4j-1.7.33.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.4.17, r7fd096f39b4284da9a71da3ce67c48d259ffa79a, Fri Mar 31 18:10:45 UTC 2023
Took 0.0020 seconds
hbase:001:0>
```

在hbase shell执行常用脚本

```
#表的创建
create 'student','info'
#获取表信息
describe 'student'
#查询表student是否存在
exists 'student'
#判断表是否enable/disable
is_enabled 'student'
is_disabled 'student'
#增加表列族relationship
disable 'student'
alter 'student',NAME=> 'relationship'
#删除列族relationship
disable 'student'
alter 'student','delete'=>'relationship'
#删除student表
disable 'student'
drop 'student'
exists 'student'

#插入数据
create 'student','info'
disable 'student'
alter 'student',NAME=> 'relationship'
enable 'student'
put 'student','07112001','info:name','Ben'
put 'student','07112001','relationship:father','Bill'
put 'student','07112001','relationship:mather','Rose'
put 'student','07112002','info:name','Bobby'
put 'student','07112002','relationship:father','Bert'
put 'student','07112002','relationship:mather','Anna'
put 'student','07112003','info:name','Jerry'
put 'student','07112003','relationship:father','Jason'
put 'student','07112003','relationship:mather','Lori'

#代码7-26 查询表数据student
get 'student','07112002'
```

```
get 'student','07112002','relationship'

#扫描student表数据
scan 'student'

#统计student表记录数
count 'student'

#删除列
delete 'student','07112003','relationship:father'
get 'student','07112003','relationship'

#删除RowKey为07112003的所有行
deleteall 'student','07112003'
scan 'student'

#删除表中所有的数据
truncate 'student'
```

运行结果如：

```
hbase:047:0> #表的创建
hbase:048:0> create 'student','info'
#获取表信息
describe 'student'
#查询表student是否存在
exists 'student'
#判断表是否enable/disable
is_enabled 'student'
is_disabled 'student'
#增加表列族relationship
disable 'student'
alter 'student',NAME=> 'relationship'
#删除列族relationship
disable 'student'
alter 'student','delete'=>'relationship'
#删除student表
disable 'student'
drop 'student'
exists 'student'Created table student
Took 1.1444 seconds


=> Hbase::Table - student
hbase:049:0> #获取表信息
hbase:050:0> describe 'student'
Table student is ENABLED


student


COLUMN FAMILIES DESCRIPTION


{NAME => 'info', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS => '1',
KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', COMPRESSION =>
'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', BLOCKCACHE => 'tru
```

```
e', BLOCKSIZE => '65536', REPLICATION_SCOPE => '0'}



1 row(s)
Quota is disabled
Took 0.0323 seconds


hbase:051:0> #查询表student是否存在
hbase:052:0> exists 'student'
Table student does exist


Took 0.0042 seconds


=> true
hbase:053:0> #判断表是否enable/disable
hbase:054:0> is_enabled 'student'
true


Took 0.0087 seconds


=> true
hbase:055:0> is_disabled 'student'
false


Took 0.0060 seconds


=> false
hbase:056:0> #增加表列族relationship
hbase:057:0> disable 'student'
Took 0.6822 seconds


hbase:058:0> alter 'student',NAME=> 'relationship'
Updating all regions with the new schema...
All regions updated.
Done.
Took 1.2510 seconds


hbase:059:0> #删除列族relationship
hbase:060:0> disable 'student'
Took 0.0105 seconds


hbase:061:0> alter 'student','delete'=>'relationship'
Updating all regions with the new schema...
All regions updated.
Done.
Took 1.1383 seconds
```

```
hbase:062:0> #删除student表
hbase:063:0> disable 'student'
Took 0.0135 seconds


hbase:064:0> drop 'student'
Took 0.4228 seconds


hbase:065:0> exists 'student'
Table student does not exist


Took 0.0052 seconds


=> false
hbase:066:0>
hbase:160:0> #插入数据
hbase:161:0> create 'student','info'
alter 'student',NAME=> 'relationship'
enable 'student'

ERROR: Table already exists: student!

For usage try 'help "create"'

Took 0.0324 seconds


hbase:162:0> disable 'student'
put 'student','07112001','info:name','Ben'
put 'student','07112001','relationship:father','Bill'
put 'student','07112001','relationship:mather','Rose'
put 'student','07112002','info:name','Bobby'
put 'student','07112002','relationship:father','Bert'
put 'student','07112002','relationship:mather','Anna'
put 'student','07112003','info:name','Jerry'
put 'student','07112003','relationship:father','Jason'
put 'student','07112003','relationship:mather','Lori'

#代码7-26 查询表数据student
get 'student','07112002'
get 'student','07112002','relationship'

#扫描student表数据
scan 'student'

#统计student表记录数
count 'student'
Took 0.3306 seconds


hbase:163:0> alter 'student',NAME=> 'relationship'

#删除列
delete 'student','07112003','relationship:father'
get 'student','07112003','relationship'

```

```
#删除RowKey为07112003的所有行
deleteall 'student','07112003'
scan 'student'

#删除表中所有的数据
truncate 'student'Updating all regions with the new schema...
All regions updated.
Done.
Took 1.2103 seconds


hbase:164:0> enable 'student'
Took 0.6591 seconds


hbase:165:0> put 'student','07112001','info:name','Ben'
Took 0.1151 seconds


hbase:166:0> put 'student','07112001','relationship:father','Bill'
Took 0.0050 seconds


hbase:167:0> put 'student','07112001','relationship:mather','Rose'
Took 0.0098 seconds


hbase:168:0> put 'student','07112002','info:name','Bobby'
Took 0.0089 seconds


hbase:169:0> put 'student','07112002','relationship:father','Bert'
Took 0.0090 seconds


hbase:170:0> put 'student','07112002','relationship:mather','Anna'
Took 0.0039 seconds


hbase:171:0> put 'student','07112003','info:name','Jerry'
Took 0.0231 seconds


hbase:172:0> put 'student','07112003','relationship:father','Jason'
Took 0.0070 seconds


hbase:173:0> put 'student','07112003','relationship:mather','Lori'
Took 0.0072 seconds


hbase:174:0>
hbase:175:0> #代码7-26 查询表数据student
hbase:176:0> get 'student','07112002'
COLUMN                                                  CELL
```

```
 info:name                                          timestamp=2023-11-16T10:58:14.527, value=Bobby
 relationship:father                                timestamp=2023-11-16T10:58:14.553, value=Bert
 relationship:mather                                timestamp=2023-11-16T10:58:14.590, value=Anna
1 row(s)
Took 0.0087 seconds


hbase:177:0> get 'student','07112002','relationship'
COLUMN                                              CELL


 relationship:father                                timestamp=2023-11-16T10:58:14.553, value=Bert
 relationship:mather                                timestamp=2023-11-16T10:58:14.590, value=Anna
1 row(s)
Took 0.0111 seconds


hbase:178:0>
hbase:179:0> #扫描student表数据
hbase:180:0> scan 'student'
ROW                                                 COLUMN+CELL


 07112001                                           column=info:name, timestamp=2023-11-16T10:58:14.472, value=Ben
 07112001                                           column=relationship:father, timestamp=2023-11-16T10:58:14.485, value=Bill
 07112001                                           column=relationship:mather, timestamp=2023-11-16T10:58:14.502, value=Rose
 07112002                                           column=info:name, timestamp=2023-11-16T10:58:14.527, value=Bobby
 07112002                                           column=relationship:father, timestamp=2023-11-16T10:58:14.553, value=Bert
 07112002                                           column=relationship:mather, timestamp=2023-11-16T10:58:14.590, value=Anna
 07112003                                           column=info:name, timestamp=2023-11-16T10:58:14.632, value=Jerry
 07112003                                           column=relationship:father, timestamp=2023-11-16T10:58:14.663, value=Jason
```

```
 07112003
column=relationship:mather, timestamp=2023-11-16T10:58:14.681, value=Lori

3 row(s)
Took 0.0454 seconds



hbase:181:0>
hbase:182:0> #统计student表记录数
hbase:183:0> count 'student'
3 row(s)
Took 0.0041 seconds



=> 3
hbase:184:0>
hbase:185:0> #删除列
hbase:186:0> delete 'student','07112003','relationship:father'
Took 0.0115 seconds



hbase:187:0> get 'student','07112003','relationship'
COLUMN                                                  CELL


 relationship:mather                                    timestamp=2023-11-
16T10:58:14.681, value=Lori

1 row(s)
Took 0.0107 seconds



hbase:188:0>
hbase:189:0> #删除RowKey为07112003的所有行
hbase:190:0> deleteall 'student','07112003'
Took 0.0381 seconds



hbase:191:0> scan 'student'
ROW                                                     COLUMN+CELL


 07112001                                               column=info:name,
timestamp=2023-11-16T10:58:14.472, value=Ben

 07112001
column=relationship:father, timestamp=2023-11-16T10:58:14.485, value=Bill

 07112001
column=relationship:mather, timestamp=2023-11-16T10:58:14.502, value=Rose

 07112002                                               column=info:name,
timestamp=2023-11-16T10:58:14.527, value=Bobby

 07112002
column=relationship:father, timestamp=2023-11-16T10:58:14.553, value=Bert
```

```
 07112002
column=relationship:mather, timestamp=2023-11-16T10:58:14.590, value=Anna

2 row(s)
Took 0.0160 seconds



hbase:192:0>
hbase:193:0> #删除表中所有的数据
hbase:194:0> truncate 'student'
Truncating 'student' table (it may take a while):
Disabling table...
Truncating table...
Took 1.7966 seconds



hbase:195:0>
```

任务7.4使用HBase Java API实现表设计

HBase是由Java语言开发的，因此HBase对外也提供了Java API的编程接口。进行HBase编程开发时，通常选用IDEA作为HBase的编程开发工具，本小节的任务如下。

1. 创建Java项目。
2. 实现表的创建。
3. 向表中添加数据。

使用idea编写api脚本

```
package chap7_hbase;
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 HBaseTest {
    public static Configuration configuration;  //管理HBase的配置信息
    public static Connection connection;  //管理HBase的连接
    public static Admin admin;   //管理HBase数据库表信息
    public static void main(String[] args)throws IOException{
        init();
        createTable("students",new String[]{"score"});
        insertData("students","George","score","Bigdata","69");
        insertData("students","George","score","Python","86");
        insertData("students","George","score","JavaWeb","77");
        close();
    }
    /**
     *
     * @param myTableName 表名
     * @param colFamily 列族数组
     * @throws IOException
     */
    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 =

ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes(str)).build();
                tableDescriptor.setColumnFamily(family);
            }
            admin.createTable(tableDescriptor.build());
        }
    }

    /**
     * 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 IOException
     */
    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();
    }

    public static void init(){
        configuration  = HBaseConfiguration.create();
//        <name>hbase.rootdir</name>
//<value>hdfs://master:8020/data/hbase_db</value>
        configuration.set("hbase.rootdir","hdfs://master:8020/data/hbase_db");
//        <name>hbase.zookeeper.quorum</name>
//<value>master,slave1,slave2</value>
        configuration.set("hbase.zookeeper.quorum","master,slave1,slave2");
        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();
```

```
        }
    }
}

```

在IDEA上运行脚本

```
"C:\Program Files\Java\jdk1.8.0_281\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.3.6\lib\idea_rt.jar=58326:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.3.6\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\Java\jdk1.8.0_281\jre\lib\charsets.jar;C:\Program ...
2023/11/16 13:40:24,408- ZooKeeper: Client environment:java.io.tmpdir=C:\Users\yuxm\AppData\Local\Temp\
2023/11/16 13:40:24,408- ZooKeeper: Client environment:java.compiler=<NA>
2023/11/16 13:40:24,408- ZooKeeper: Client environment:os.name=Windows 10
2023/11/16 13:40:24,408- ZooKeeper: Client environment:os.arch=amd64
2023/11/16 13:40:24,408- ZooKeeper: Client environment:os.version=10.0
2023/11/16 13:40:24,408- ZooKeeper: Client environment:user.name=yuxm
2023/11/16 13:40:24,408- ZooKeeper: Client environment:user.home=C:\Users\yuxm
2023/11/16 13:40:24,408- ZooKeeper: Client environment:user.dir=D:\yuxm\wordc2
2023/11/16 13:40:24,409- ZooKeeper: Initiating client connection, connectString=master:2181,slave1:2181,slave2:2181 sessionTimeout=90000 watcher=org.apache.hadoop.hbase.zookeeper.ReadOnlyZKClient$$Lambda$8/1709624311@2f456084
2023/11/16 13:40:24,659- ClientCnxn: Opening socket connection to server slave1/192.168.128.131:2181. Will not attempt to authenticate using SASL (unknown error)
2023/11/16 13:40:24,660- ClientCnxn: Socket connection established to slave1/192.168.128.131:2181, initiating session
2023/11/16 13:40:24,679- ClientCnxn: Session establishment complete on server slave1/192.168.128.131:2181, sessionid = 0x20000203b260005, negotiated timeout = 40000
2023/11/16 13:40:27,866- HBaseAdmin: Operation: CREATE, Table Name: default:students, procId: 72 completed
2023/11/16 13:40:27,969- ConnectionImplementation: Closing master protocol: MasterService

Process finished with exit code 0
```

使用hbase shell查询验证结果

```
hbase:002:0> scan 'student'
ROW                           COLUMN+CELL

0 row(s)
Took 0.2577 seconds

hbase:003:0> scan 'students'
ROW                           COLUMN+CELL

 George                       column=score:Bigdata, timestamp=2023-11-
16T13:40:27.942, value=69
 George                       column=score:JavaWeb, timestamp=2023-11-
16T13:40:27.968, value=77
 George                       column=score:Python, timestamp=2023-11-
16T13:40:27.963, value=86
1 row(s)
Took 0.0547 seconds

hbase:004:0>
```

可见，在idea运行的程序中入库的三条记录通过hbase shell的scan能查询到。

任务7.5查询分析需求

运营商借助自身庞大的用户体系，从而建立海量的大数据体系。通过建模来分析和抓取语音通话记录，制订科学手机套餐，创新精准获客和营销模式，本小节的任务如下。

1. 查询分析通话记录数据。
2. 任务实现

设计通话数据结构

根据HBase分布式数据库的数据存储结构，本次需要存储通话记录数据的表结构如下所示

rowkey（行键）**	**列族**（basic）**		
dnum	length	type	date
r1			

分析查询需求

创建一个表名为phone_log，列族为basic的数据表。

通过HBase Java API生成模拟用户10个，同时每个用户模拟生成1000条通话记录。

将总计10000条通话记录添加到表phone_log中，统计指定用户3月份的所有通话记录数，具体步骤如下。

1. 创建表，同时检测表是否存在。
2. 随机生成10个用户10000条通话记录。
3. 统计查询指定手机号3月份的全部通话记录。

任务实现

PhoneLogDemo类

```
package chap7_hbase;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Random;

public class PhoneLogDemo {
    public static Configuration configuration;  //管理HBase的配置信息
    public static Connection connection;  //管理HBase的连接
    public static Admin admin;   //管理HBase数据库表信息
    public static Random random;
    public static SimpleDateFormat sdf;
    public static void main(String[] args) throws IOException, ParseException {
        random = new Random();
        sdf= new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        init();
//        createTable("phone_log",new String[]{"basic"});
        insertData("phone_log","basic");

        close();
    }
    /**
     *
     * @param myTableName 表名
     * @param colFamily 列族数组
     * @throws IOException
     */
    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 =

ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes(str)).build();
                tableDescriptor.setColumnFamily(family);
            }
            admin.createTable(tableDescriptor.build());
        }
    }

    /**
```

```
     * 添加数据
     * @param tableName 表名
     * @param colFamily 列族
     * @throws IOException
     */
    public static void insertData(String tableName,String colFamily) throws
IOException, ParseException {
        Table table = connection.getTable(TableName.valueOf(tableName));
        List<Put> putList = new ArrayList<Put>();
        for (int i=1; i<=10; i++)
        {
            String phoneNum = getPhoneNum("158");
            System.out.println(phoneNum);
            putList.clear();
            for (int j=1;j<=1000; j++){
                String dnum = getPhoneNum("199");
                int length = random.nextInt(99) +1;
                int type = random.nextInt(2);
                String date = getDate(2019);
                String rowKey= phoneNum + "_" + (Long.MAX_VALUE -
sdf.parse(date).getTime());
                Put put = new Put(rowKey.getBytes());

put.addColumn(colFamily.getBytes(),"dnum".getBytes(),Bytes.toBytes(dnum));

put.addColumn(colFamily.getBytes(),"length".getBytes(),Bytes.toBytes(length));

put.addColumn(colFamily.getBytes(),"type".getBytes(),Bytes.toBytes(type));

put.addColumn(colFamily.getBytes(),"date".getBytes(),Bytes.toBytes(date));
                putList.add(put);
            }
        }
        table.put(putList);
        table.close();
    }

    private static String getDate(int year){
        Calendar calendar = Calendar.getInstance();
        calendar.set(year,0,1);
        calendar.add(Calendar.MONTH, random.nextInt(12));
        calendar.add(Calendar.DAY_OF_MONTH,random.nextInt(31));
        calendar.add(Calendar.HOUR_OF_DAY,random.nextInt(12));
        calendar.add(Calendar.MINUTE,random.nextInt(60));
        calendar.add(Calendar.MILLISECOND,random.nextInt(60));
        return  sdf.format(calendar.getTime());
    }

    private static String getPhoneNum(String prefixNum){
        return prefixNum + String.format("%08d", random.nextInt(99999999));
    }
    public static void init(){
        configuration  = HBaseConfiguration.create();
//        <name>hbase.rootdir</name>
//<value>hdfs://master:8020/data/hbase_db</value>
        configuration.set("hbase.rootdir","hdfs://master:8020/data/hbase_db");
//        <name>hbase.zookeeper.quorum</name>
//<value>master,slave1,slave2</value>
        configuration.set("hbase.zookeeper.quorum","master,slave1,slave2");
```

```
        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();
        }
    }
}

```

运行结果如：

```
"C:\Program Files\Java\jdk1.8.0_281\bin\java.exe" -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:50943,suspend=y,server=n -javaagent:C:\Users\yuxm\.IdeaIC2018.3\system\captureAgent\debugger-agent.jar -Dfile.encoding=UTF-8 -classpath "C:\Program Files\Java\jdk1.8.0_281\jre\lib\charsets.jar;C:\Program Files\Java\jdk1.8.0_281\jre\lib\deploy.jar;C:\Program Files\Java\jdk1.8.0_281\jre\lib\ext\access-bridge-64.jar;C:\Program Files\Java\jdk1.8.0_281\jre\lib\ext\cldrdata.jar;......;.
2023/11/16 23:07:18,569- ZooKeeper: Client environment:java.io.tmpdir=C:\Users\yuxm\AppData\Local\Temp\
2023/11/16 23:07:18,569- ZooKeeper: Client environment:java.compiler=<NA>
2023/11/16 23:07:18,569- ZooKeeper: Client environment:os.name=Windows 10
2023/11/16 23:07:18,569- ZooKeeper: Client environment:os.arch=amd64
2023/11/16 23:07:18,569- ZooKeeper: Client environment:os.version=10.0
2023/11/16 23:07:18,569- ZooKeeper: Client environment:user.name=yuxm
2023/11/16 23:07:18,569- ZooKeeper: Client environment:user.home=C:\Users\yuxm
2023/11/16 23:07:18,569- ZooKeeper: Client environment:user.dir=D:\yuxm\wordc2
2023/11/16 23:07:18,570- ZooKeeper: Initiating client connection, connectString=master:2181,slave1:2181,slave2:2181 sessionTimeout=90000 watcher=org.apache.hadoop.hbase.zookeeper.ReadOnlyZKClient$$Lambda$8/454645053@21f3418e
2023/11/16 23:07:18,586- ClientCnxn: Opening socket connection to server master/192.168.128.130:2181. Will not attempt to authenticate using SASL (unknown error)
2023/11/16 23:07:18,586- ClientCnxn: Socket connection established to master/192.168.128.130:2181, initiating session
2023/11/16 23:07:18,683- ClientCnxn: Session establishment complete on server master/192.168.128.130:2181, sessionid = 0x10000021c510005, negotiated timeout = 40000
15885699303
2023/11/16 23:07:48,816- ClientCnxn: Client session timed out, have not heard from server in 30131ms for sessionid 0x10000021c510005
```

```
2023/11/16 23:07:48,816- ClientCnxn: Client session timed out, have not heard from server in 30131ms for sessionid 0x10000021c510005, closing socket connection and attempting reconnect
15863018220
15806683014
15873643192
15835033656
15813336804
15894711418
15879010905
15806775945
15830981018
2023/11/16 23:07:49,282- ClientCnxn: Opening socket connection to server slave1/192.168.128.131:2181. Will not attempt to authenticate using SASL (unknown error)
2023/11/16 23:07:49,282- ClientCnxn: Socket connection established to slave1/192.168.128.131:2181, initiating session
2023/11/16 23:07:49,284- ClientCnxn: Session establishment complete on server slave1/192.168.128.131:2181, sessionid = 0x10000021c510005, negotiated timeout = 40000
Disconnected from the target VM, address: '127.0.0.1:50943', transport: 'socket'

Process finished with exit code 0

```

Search类

```
package chap7_hbase;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.CellUtil;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Random;

public class Search {
    public static Configuration configuration;  //管理HBase的配置信息
    public static Connection connection;  //管理HBase的连接
    public static Admin admin;   //管理HBase数据库表信息
    public static Random random;
    public static SimpleDateFormat sdf;
    public static void main(String[] args) throws IOException, ParseException{
        random = new Random();
        sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        init();
        scan("phone_log","15830981018");
        close();

    }
```

```
    public static void scan(String tableName,String num) throws ParseException,IOException
    {
        String phoneNume = num;
        Scan scan = new Scan();
        String startRow = phoneNume +"_" +(Long.MAX_VALUE - sdf.parse("2019-12-23 21:38:13").getTime());
        scan.setStartRow(startRow.getBytes());
        String stopRow = phoneNume +"_" +(Long.MAX_VALUE - sdf.parse("2019-12-22 21:38:13").getTime());
        scan.setStopRow(stopRow.getBytes());
        Table table= connection.getTable(TableName.valueOf(tableName));
        ResultScanner resultScanner = table.getScanner(scan);
        for (Result result:resultScanner){
            printMsg(result);
        }
        resultScanner.close();
    }
    public  static void printMsg(Result result){

System.out.print(Bytes.toString(CellUtil.cloneValue(result.getColumnLatestCell("basic".getBytes(),"dnum".getBytes()))+"\t");

System.out.print(Bytes.toInt(CellUtil.cloneValue(result.getColumnLatestCell("basic".getBytes(),"type".getBytes()))+"\t");

System.out.print(Bytes.toInt(CellUtil.cloneValue(result.getColumnLatestCell("basic".getBytes(),"length".getBytes()))+"\t");

System.out.println(Bytes.toString(CellUtil.cloneValue(result.getColumnLatestCell("basic".getBytes(),"date".getBytes()))));
    }
    public static void init(){
        configuration  = HBaseConfiguration.create();
//        <name>hbase.rootdir</name>
//<value>hdfs://master:8020/data/hbase_db</value>
        configuration.set("hbase.rootdir","hdfs://master:8020/data/hbase_db");
//        <name>hbase.zookeeper.quorum</name>
//<value>master,slave1,slave2</value>
        configuration.set("hbase.zookeeper.quorum","master,slave1,slave2");
        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();
        }
```

```
    }
}

```

运行结果：

```
"C:\Program Files\Java\jdk1.8.0_281\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.3.6\lib\idea_rt.jar=51865:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2018.3.6\bin" -Dfile.encoding=UTF-8 -classpath "C:\Program Files\Java\jdk1.8.0_281\jre\lib\charsets.jar;C:\Program Files\Java\jdk1.8.0_281\jre\lib\deploy.jar;C:\Program Files\Java\jdk1.8.0_281\jre\lib\ext\access-bridge-64.jar;C:\Program Files\Java\jdk1.8.0_281\jre\lib\ext\cldrdata.jar;C:\Program ......;D:\hadoop\hadoop-3.1.4\sbin;C:\Progra~1\Java\jdk1.8.0_281\bin;C:\Users\yuxm\AppData\Local\Microsoft\WindowsApps;;.
2023/11/16 23:18:07,432- ZooKeeper: Client environment:java.io.tmpdir=C:\Users\yuxm\AppData\Local\Temp\
2023/11/16 23:18:07,432- ZooKeeper: Client environment:java.compiler=<NA>
2023/11/16 23:18:07,432- ZooKeeper: Client environment:os.name=Windows 10
2023/11/16 23:18:07,432- ZooKeeper: Client environment:os.arch=amd64
2023/11/16 23:18:07,432- ZooKeeper: Client environment:os.version=10.0
2023/11/16 23:18:07,432- ZooKeeper: Client environment:user.name=yuxm
2023/11/16 23:18:07,432- ZooKeeper: Client environment:user.home=C:\Users\yuxm
2023/11/16 23:18:07,432- ZooKeeper: Client environment:user.dir=D:\yuxm\wordc2
2023/11/16 23:18:07,433- ZooKeeper: Initiating client connection, connectString=master:2181,slave1:2181,slave2:2181 sessionTimeout=90000 watcher=org.apache.hadoop.hbase.zookeeper.ReadOnlyZKClient$$Lambda$8/45218718@7d675eb8
2023/11/16 23:18:07,650- ClientCnxn: Opening socket connection to server slave2/192.168.128.132:2181. Will not attempt to authenticate using SASL (unknown error)
2023/11/16 23:18:07,651- ClientCnxn: Socket connection established to slave2/192.168.128.132:2181, initiating session
2023/11/16 23:18:07,658- ClientCnxn: Session establishment complete on server slave2/192.168.128.132:2181, sessionid = 0x300000ab1ec0006, negotiated timeout = 40000
19909555153 0   5   2019-12-23 09:16:48
19959261742 1   79  2019-12-23 08:46:48
19954630640 1   67  2019-12-23 03:17:48
19939535113 0   26  2019-12-22 23:34:48

Process finished with exit code 0

```

小结

1. HBase 的基本概念、系统架构、数据模型及读写流程的理论知识。
2. HBase、 ZooKeeper的安装配置步骤。
3. 常用的HBase Shell命令的使用方法。
4. HBase的Java应用程序开发方法，并使用HBase的Java API实现了通话记录的查询。