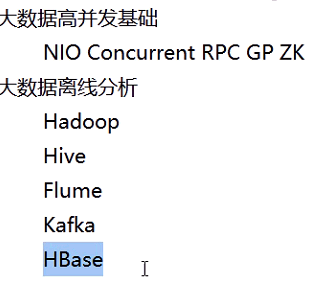
Big\_day17

HBase01



**HBase概述**

基于Hadoop数据库工具

来源于Google三篇论文之一 BIGTABLE，APACHE做了开源的实现

HBASE 技术是一种 NoSQL的 非关系型数据库 不符合关系型数据库的范式

适合存储 半结构化 非结构化的数据

适合存储稀疏的数据 空的数据不占用空间

面向列(族)进行存储

提供实时的增删改查的能力 是一种真正的数据库产品

可以存储海量数据 性能非常优良 可以实现 上亿条记录的毫秒级别的查询

但是不支持严格的事务控制 只能在行级别保证事务

是一个高可靠 高性能 面向列 可伸缩的分布式存储系统 利用hbase技术可以在廉价的PC上搭建起大规模结构化存储集群。

HBase利用HadoopHDFS作为其文件存储系统，利用Hadoop的MapReduce来处理HBase中的海量数据，利用Zookeeper作为协调工具。

谷歌三篇论文

Google File System

Google MapReduce

Google Bigtable

半结构化数据--网页,json

非结构化数据-视频等

**HBase的逻辑结构**

HBase使用表来存储数据 但是表的结构和特点和传统的关系型数据库有非常大的区别

**行键** - RowKey

就相当于是HBase表中的主键，HBase中的所有的表都要有行键

HBase中的所有的数据都要按照行键的字典顺序排序后存储

对HBase表中的数据的查询 只有三种方式：

1.根据指定行键查询

2.根据指定的行键范围查询

3.全表扫描查询

**列族(簇)** - ColumnFamily

是HBase表中垂直方向保存数据的结构，列族是HBase表的元数据的一部分，需要在定义HBase表时就指定好表具有哪些个列族，列族中可以包含一个或多个列

**列** - Column

HBase表中列族里可以包含一个或多个列，列并不是HBase表的元数据的一部分，不需要在创建表时预先定义，而是可以在后续使用表时随时为表的列族动态的增加列。

**单元格和时间戳** -  Cell TimeStamp

在HBase表中，水平方向的行 和 垂直方向的列 交汇 就得到了HBase中的一个存储单元，而在这个存储单元中，可以存储数据，并且可以保存数据的多个版本，这些个版本之间通过时间戳来进行区分。

所以在HBase中 可以通过 行键 列族 列 时间戳 来确定一个最小的存储数据的单元，这个单元就称之为单元格 Cell。

单元格中的数据都以二进制形式存储，没有数据类型的区别。

 行键--主键(必须有)

列族(簇)

列--随用随加

单元格和时间戳:在HBASE中,水平方向的行 和 垂直方向的列 交汇 .就得到了HBASE中的一个存储单元.--单元格:

**HBase的安装配置**

**完全分布式的配置:**

1.修改conf/hbase-env.sh修改JAVA\_HOME

export JAVA\_HOME=xxxx

2.修改conf/hbase-env.sh禁用对zookeeper的自动管理

export HBASE\_MANAGES\_ZK false

hbase默认HBASE\_MANAGES\_ZK为true，则HBase会自动管理zk，当HBase启动时，会自动去启动zk,在HBase关闭时，会自动关闭zk。而在很多的场景下，zk不是转为HBase服务器，不希望HBase在关闭时连带着关闭zk，此时需要 将此选项改为false

3.修改hbase-site.xml，配置开启完全分布式模式,配置hbase.cluster.distributed为true。配置hbase.rootdir设置为HDFS访问地址

<property>

<name>hbase.rootdir</name>指定底层存储位置

<value>hdfs://hadoop01:9000/hbase</value>

</property>

<property>

<name>dfs.replication</name>指定底层HDFS的副本存储数量

<value>1</value>

</property>

<property>

<name>hbase.cluster.distributed</name>是否开启集群模式

<value>true</value>

</property>

<property>

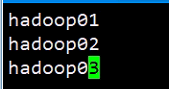
<name>hbase.zookeeper.quorum</name>

完全分布式模式下需要使用zk作为集群协调工具，通过这个选项配置使用的zk

<value>hadoop01:2181,hadoop02:2181,hadoop03:2181</value>

</property>

4.配置region服务器,修改conf/regionservers文件，其中配置所有hbase主机，每个主机名独占一行，hbase启动或关闭时会按照该配置顺序启动或关闭主机中的hbase



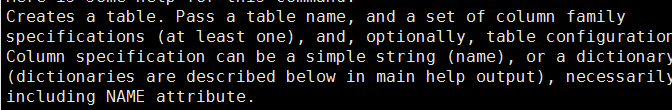
启动Hbase bin# ./start-hbase.sh

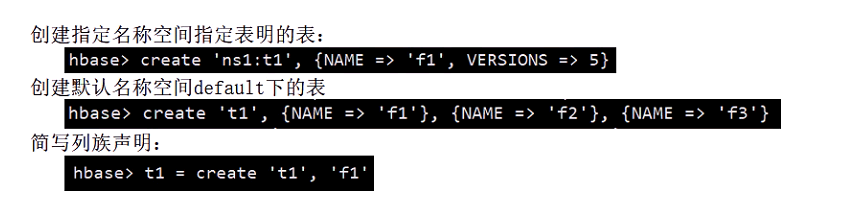
启动备用master实现高可用 bin# ./hbase-daemon.sh start master

通过hbase shell脚本来访问hbase bin# hbase shell

p:60010

**HBase 的shell命令行操作**





三种建表方式:

hbase(main):019:0> create\_namespace 'park'

hbase(main):019:0> create 'park:tab1',{NAME=>'cf1',VERSIONS=>3}

hbase(main):019:0> create 'tab2',{NAME=>'f1'},{NAME=>'f2'},{NAME=>'f3'}

hbase(main):019:0> create 'tab3','cf1','cf2'

查看列表: list

查看表信息 desc 'tab1'

删除表先禁用表

disable 'tab1' --对应的是enable 'tab1'

drop 'tab1'

put

put 'tab11','rk2','cf2:c2','v2222'

get

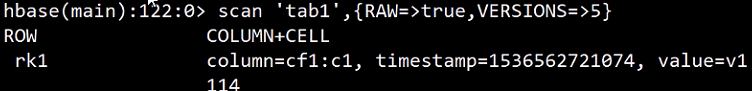
get 'tab11','rk1','cf1:c1'

get 'tab11','rk1'

scan

scan 'tab11'

scan 'tab11',{RAW=>true,VERSIONS=>5}



delete删除数据

delete 'tab11','rk2','cf2:c2'

truncate 摧毁表,表中的数据全部摧毁

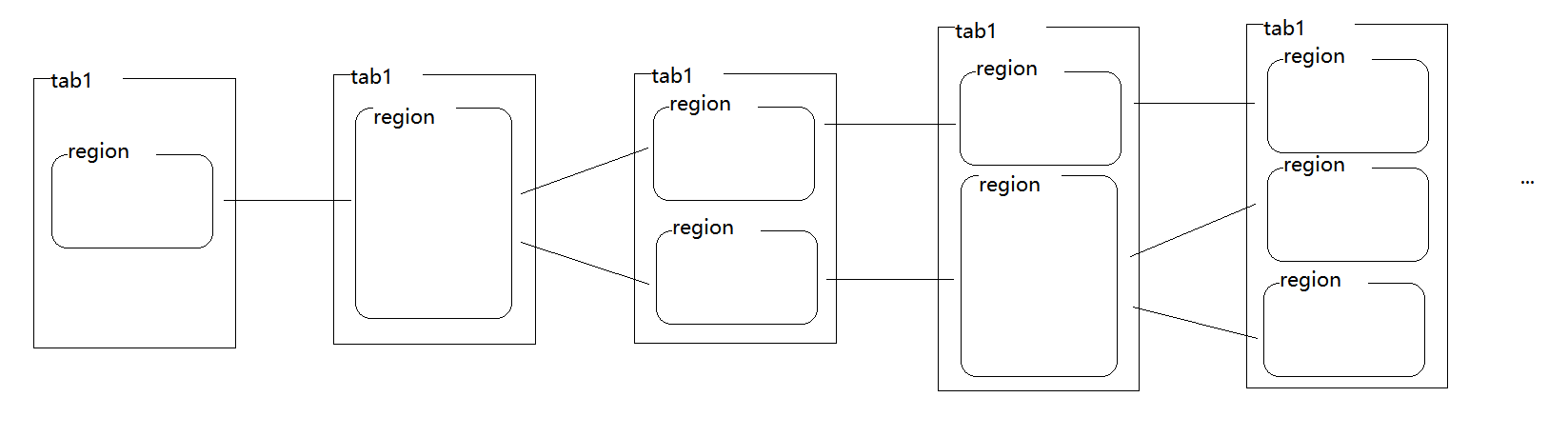
truncate 'tab11'

**HBase原理**

**HRegion的分裂**

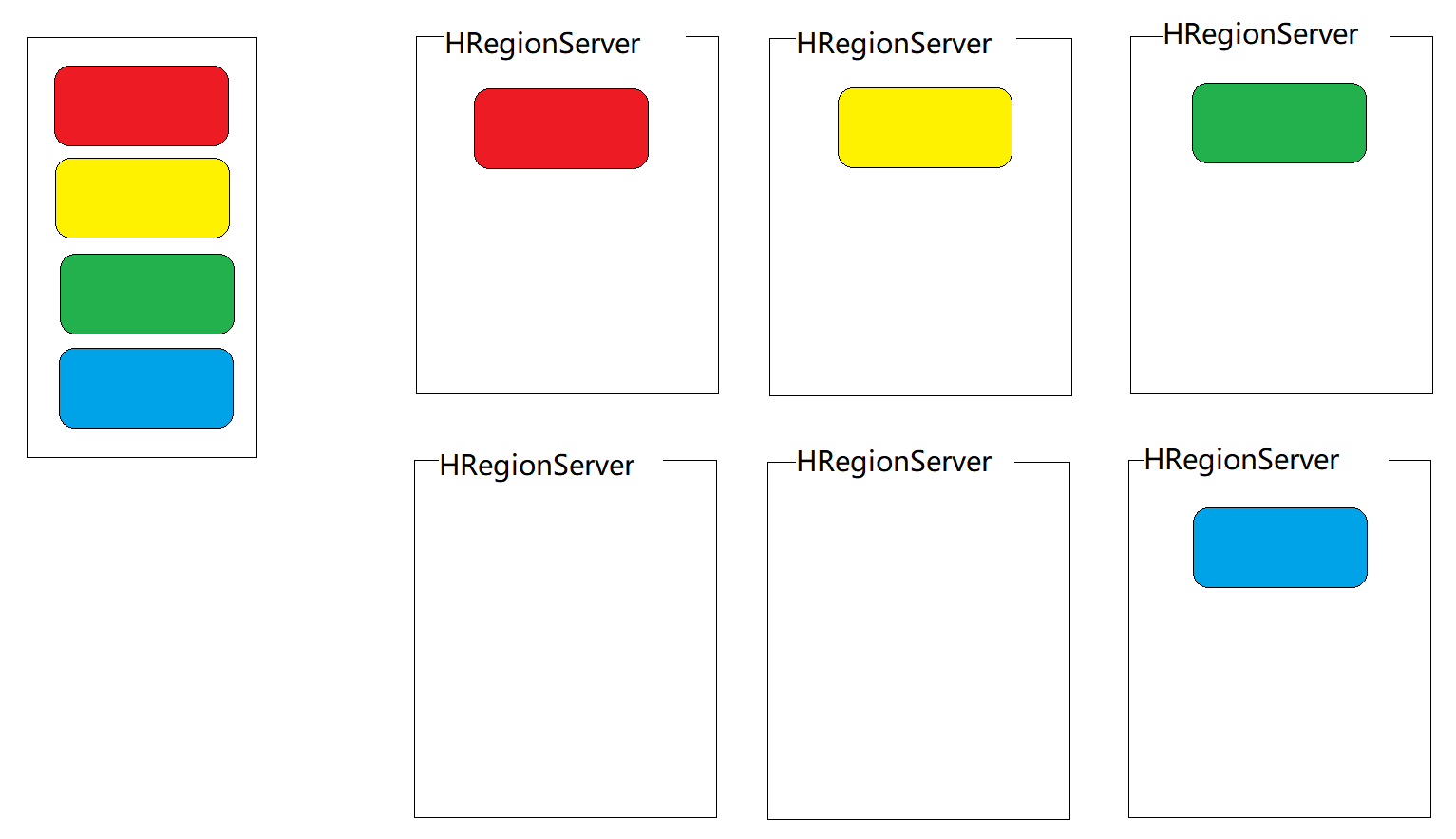
HBase表中的数据按照行键的字典顺序进行排列

HBase表最初只有一个HRegion保存数据，随着数据写入，HRegion越来越大，达到一定的阈值后会自动分裂为两个Hregion,随着这个过程不停的进行，HBase表中的数据会被划分为若干个HRegion进行管理



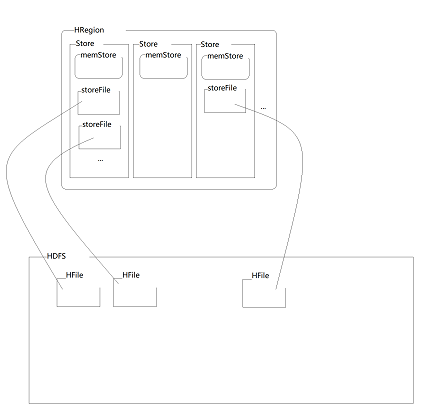
**HRegion是HBase表中的数据分布式存储和负载均衡的基本单元**

HBase表中的数据会以HRegion为单位分布式的存放在集群的HRegionServer服务器中，从而实现分布式存储和负载均衡



**HRegion内部结构--列族的数量决定了store的数量**

在Region内部存在复杂的结构，Region内部划分出若干个Store，有几个Store取决于HBase表有几个列族，一个列族对应一个Store，在Store内部又有一个memStore和0个或若干个storeFile，而**storeFile本质上是存在于HDFS中的称为HFile的文件**



**HBase的写数据流程**

当客户端联系HBase要**写入**一条数据时，根据表名和行键确定要操作的是哪个HRegion，找到存储着该HRegion的HRegionServer，对该HRegion进行操作，根据要操作的列族确定要操作的store，向该sotre中的memStore中写入当前数据，并在HLog中记录操作日志，之后返回表示写入成功。

**问题1：内存满了怎么办**

当不停的写入数据，将store中的memStore填满时，重新生成一个新的memStore继续工作，而不再对旧的memStore写入数据，此时HBase会启动一个独立的线程，将旧的memStore中的数据写入到HDFS中的一个新的HFile中，最终将数据持久化保存在了HDFS中。

在不停的产生HFile过程中，同一个Store的先后产生的多个HFile中可能存在对同一个数据的多个不同的版本，其中旧的版本的数据很可能已经是失效的垃圾数据了，但是由于HDFS只能一次写入多次读取不支持行级别的增删改，这些垃圾数据无法及时清理。最终造成浪费存储空间，降低查询性能。

因此当HFile的数量达到一定的量，或达到一定的时间间隔，HBase将会触发HFile的合并操作，将同一个Store的先后产生的多个HFile合并成一个HFile，在合并的过程中，会将垃圾数据清理掉。而当不停的合并产生了达到一定大小的HFile后，HFile还会被拆分为若干个小的HFile，防止HFile过大。

这个过程中看似 先合并 又拆分，小到大 大到 小，其实 在这个过程集中，垃圾数据就被 清理掉了 。

**问题2：内存断电丢数据怎么办**

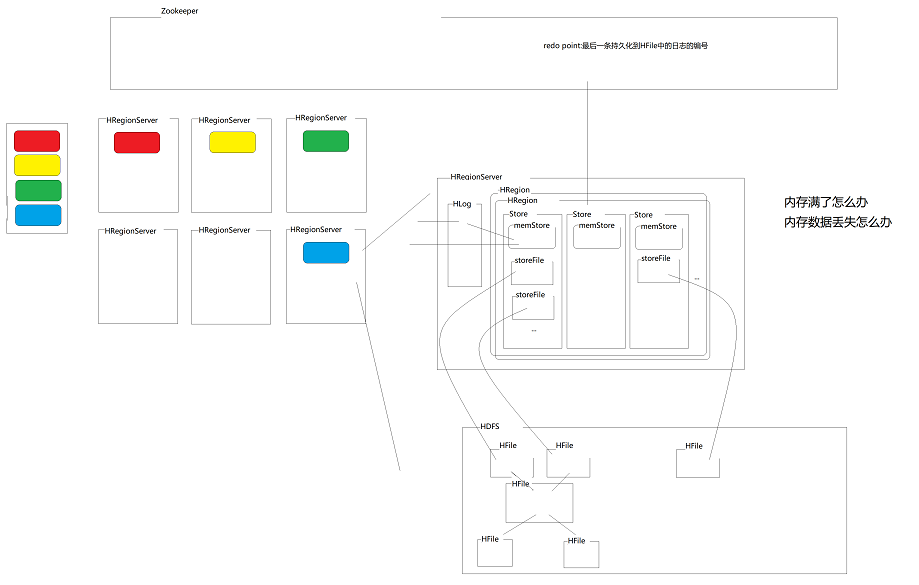
在HBase的HRegionServer中存在名为HLog的日志文件，在向memStore写入数据时，数据需要同时写入HLog中记录操作日志。

这个HLog文件本质上是存在于HDFS中的一个文件，通过对HDFS中的这个文件不停的追加数据记录操作日志。

而在memStore满了溢写到HFile中完成后，HBase会将最后一条持久化到HFile中的日志的编号记录到Zookeeper中redo point。

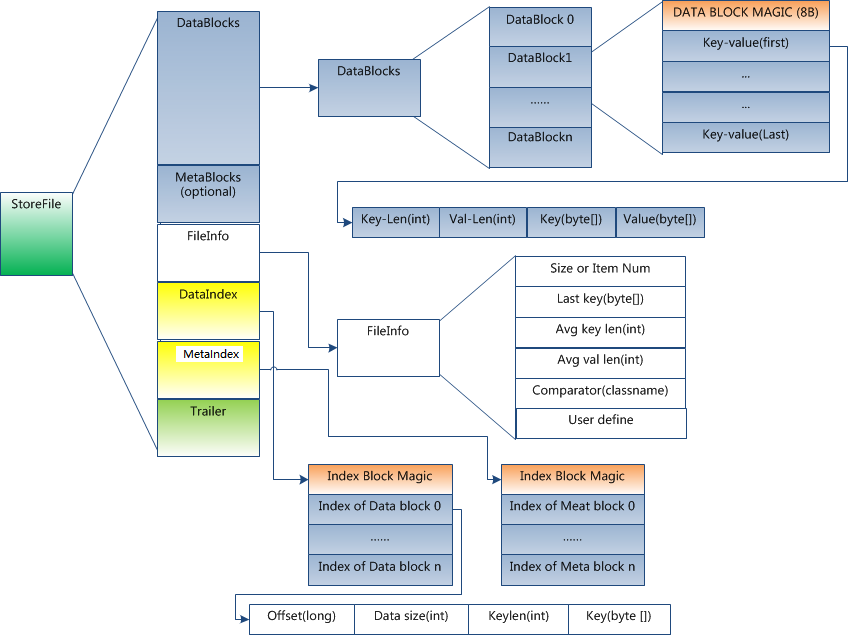
这样一旦断电丢失内存数据，只需要到Zookeeper中找到最后一条持久化的日志的编号，再从HLog中将这个编号之后的数据恢复到内存中即可以找回所有的数据。

为了防止HLog文件过多，分摊写入性能，HBase中一个HRegionServer一个HLog，这个HRegionServer中的所有的HRegion的日志都会记在这同一个HLog文件中。



**HBase的写可以认为是基于内存来实现的，速度非常的块，最终通过溢写到HFile中数据持久化高可靠的保存在HDFS中，保证了数据可靠。**

* 1. HFile的文件结构



**一个StoreFile分为DataBlock MetaBlock FileInfo DataIndex MetaIndex Trailer**

其中：

**Data Blocks**--保存表中的数据，这部分可以被压缩

DataBlocks中存放了大量的DataBlock，其中以键值对的形式保存着表中的数据，其中 键是行键，值是该行的某一个列的值，所以一个HBase表中的一个行可能在底层存在多键值对保存

**Meta Blocks** (可选的)--保存用户自定义的kv对，可以被压缩。

**File Info--**Hfile的元信息，不被压缩，用户也可以在这一部分添加自己的元信息。

**Data Block Index--**Data Block的索引。

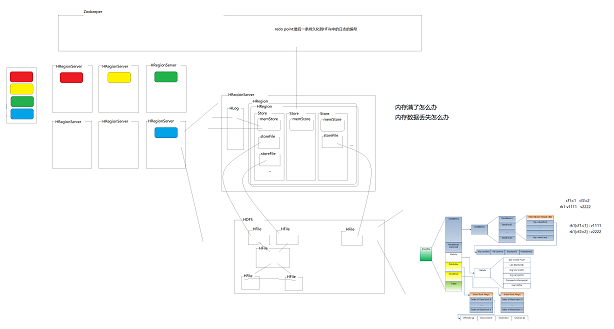
**Meta Block Index** (可选的)--Meta Block的索引。

**Trailer**--这一段是定长的。保存了每一段的偏移量，读取一个HFile时，会首先 读取Trailer，Trailer保存了每个段的起始位置

**HBase的读数据流程**

当客户端联系HBase标识要读取某一张表时，根据表和行键确定出HRegion，找到存有该HRegion的HRegionServer，找到HRegion，根据要 查找的列族，确定出要查询的Store，首先在memStore中寻找要查询的数据，如果能查到，直接返回查询到的数据，查询结束。如果在memStore中找不到要查询的数据，要查询该store对应的所有的storeFile，在这个过程中，解析storeFile,先读取storeFile中的Trailer块，找到DataBlockIndex，根据判断要找到数据在当前storeFile中是否存在，如果不存在直接返回空 ，如果存在则找到对应的DataBlocks中的DataBlock返回。这样多个storeFile可能返回了多个DataBlock，其中包含着多个版本的查询的数据结果，之后在内存中将这些DataBlock信息合并，得到最新的数据返回，完成查询。

在理想的情况下 ，HBase的查询可以基于内存完成，效率很高，在最不理想的情况下，需要大量的查询底层的HDFS文件，性能会有所下降，但是，由于这些storeFile都增加了索引，所以查询的速度仍然是由保证的，但是仍然会比最理想的情况慢大概一个数量级。



Big\_day18

HBase02

**7.HBase的HRegion寻址**

在Hbase中存在一张特殊的**meta表**,其中存放着HBase的**元数据信息**,包括,有哪些表,表有哪些HRegion,每个HRegion分布在哪个HRegionServer中。meta表很特殊，**永远有且仅有一个HRegion**，这个HRegion存放在某一个HRegionServer中，并且会将这个持有meta表的Region的HRegionServer的地址存放在**Zookeepe**r中meta-region-server下。

所以当在进行HBase表的**读写操作**时，需要先根据表名 和 行键 定位到HRegion，这个过程就是HRegion的**寻址过程**。

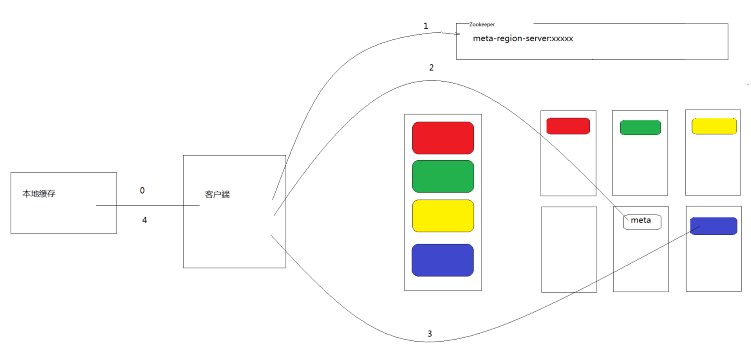
HRgion的寻址过程首先由客户端开始，访问zookeeper 得到其中meta-region-server的值,根据该值找到meta表唯一的HRegion存储的HRegionServer,得到meta表唯一的HRegion,从中读取真正要查询的表和行键 对应的HRgion的地址,再根据该地址,找到真正的操作的HRegionServer和HRegion,完成HRgion的定位,继续读写操作.

客户端会缓存之前已经查找过的HRegion的地址信息,之后的HRgion定位中,如果能在本地缓存中的找到地址,就直接使用该地址提升性能。

自我总结:

**meta表**---永远仅有一个HRegion,存放着HBase的元数据信息,包括有哪些表,表有哪些HRegion,每个 HRegion分布在哪个HRegionServer中,

**meta-region-server:**存放meta表的Region的HRegionServer的地址,通过此表可以得到对应的HRegion

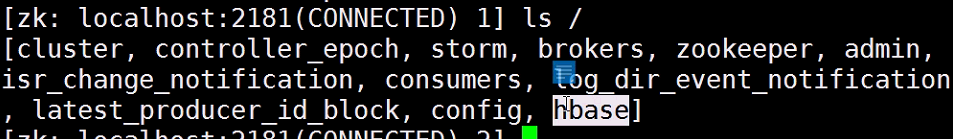


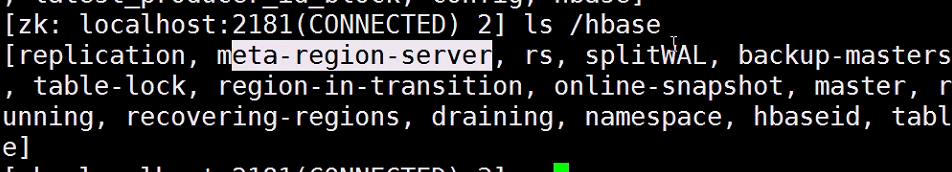
create 'tab111','cf1'

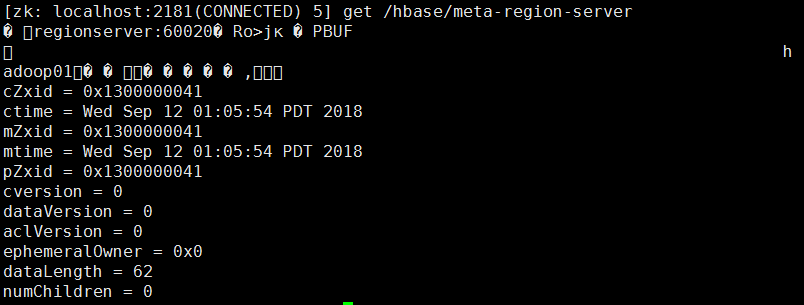
put 'tab111','rk2','cf1:c1','v1112'

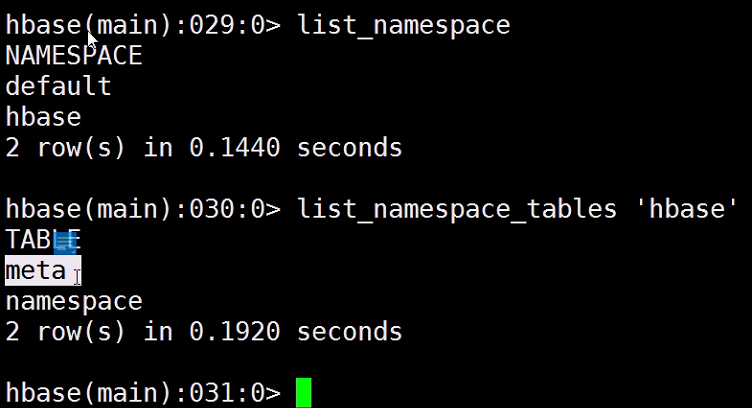
scan 'tab111'

启动zookeeper客户端:



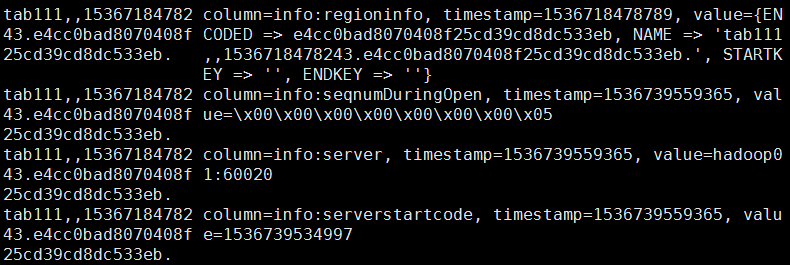






查看所有的表的meta信息

IMG_261



**8.三种常见存储系统**

--详见 <http://www.cnblogs.com/yanghuahui/p/3483754.html>

1. hash存储

读写效率非常高，无法顺序扫描,对应的是key-value存储系统 --HashMap

1. B树存储

读写效率不错，可以顺序扫描 --MySql

其读写效率没有hash高,需要遍历树

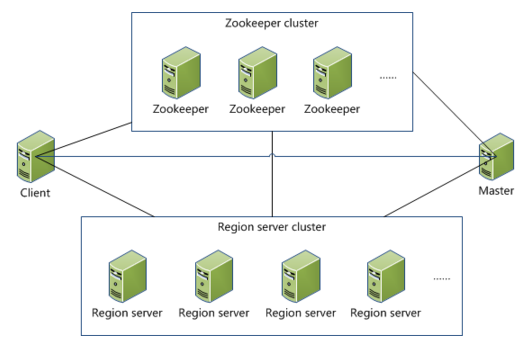
1. LSM树存储--Log-Structured Merge Tree

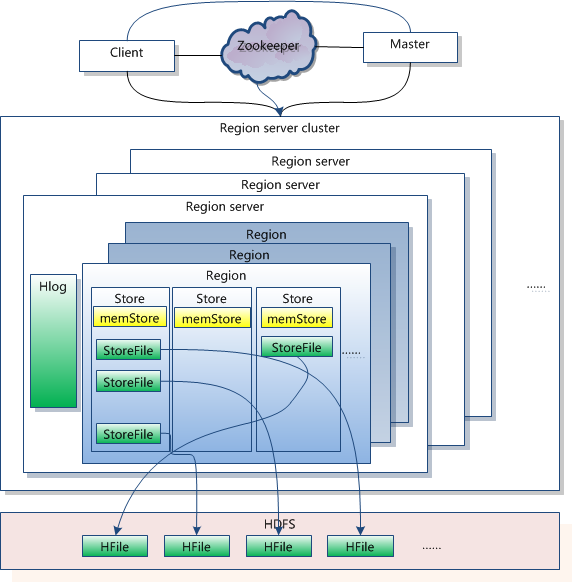
写性能优良，读性能看是否命中内存，支持顺序扫描,可以通过批量存储技术规避磁盘随机写入问题. --Hbase

**LSM树和B+树相比，LSM树牺牲了部分读性能，用来大幅提高写性能**

**9.HBase系统架构**

详见 <https://blog.csdn.net/gaijianwei/article/details/46271011>





HBase中的老大叫HMaster 小弟叫HRegionServer,客户端叫Client,Zookeepr为hbase提供集群协调

1)Client客户端

访问hbase 保留一些缓存信息提升效率

2)zookeeper -redo point , meta-region-server

保证任何时候集群只有一个HMaster

监控regionServer的状态 将其上线下线信息通知mater

存储meta表Region的地址

存储hbase的元数据信息 包括 有哪些表 有哪些列族等等

3)HMater

为RegionServer分配Region

为RegionServer进行负载的均衡

GFS上的垃圾回收 分-合-分过程中产生的数据的清理

处理对Schema数据的更新请求

4)HRegionServer

维护Master分配给它的region，处理对这些region的IO请求

负责切分在运行过程中变得过大的region

**10.HBase总结**

1)为什么hbase可以很快

内部有memStore做缓冲,读写都是有限基于内存实现的效率高

数据按照行键排序 查询效率高

数据从水平方向上切分为若干个HRegion 分布式是的存储 提高效率

2)为什么HBASE可以存储很多数据-HDFS是分布式的

数据最终存储到HDFS上,而HDFS是分布式的存储系统,可以动态扩容,基本可以认为容量没有限制

空的数据不占用空间,当数据比较稀疏时,不会浪费空间

hbase按列存储数据,而同一个列中的数据数据结构往往类似,可以实现高下率的数据压缩,节省空间

3)为什么hbase是可靠的--hdfs

数据最终存储在HDFS中,而HDFS自带多副本机制保证高可靠

存在多个HRegionServer,即使某些HRegionServer宕机,HBase也可以恢复数据到其他 HRegionServer继续工作

HMaster提供了备用机制,自动在HMaster 和 BackUpHMaster之间切换

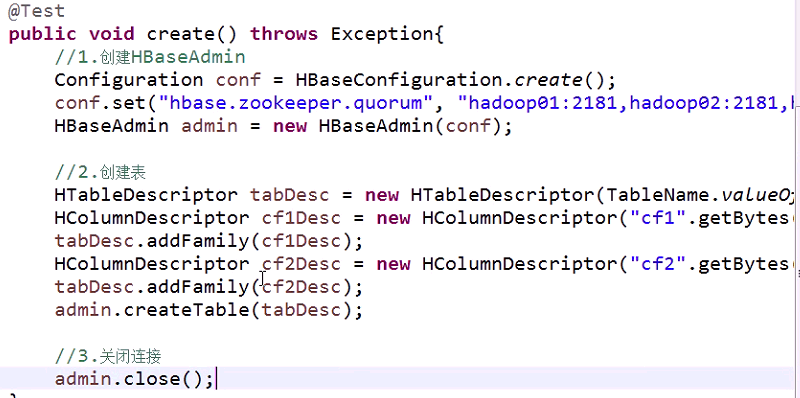
4)hbase和hive和传统的关系型数据库的比较

比起传统的关系型数据库，可以存储半结构化非结构化的数据，可以存储和处理更大级别的数据，提供高效的查询，对于稀疏数据的处理更好，具有更好的横向扩展性，免费开源性价比很高。但是不能支持非常好的事务特性，只支持行级的事务。只能通过行键来查询，表设计时难度更高。而mysql用来存储结构化的数据提供更好的事务控制。

比起hive，hive只是在mapreduce上包了一层壳，本质上还是离线数据的处理的工具，实时查询性能有限，本质上是一个基于hadoop的数据仓库工具，不能支持行级别的新增修改和删除。hbase可以提供实时的数据的处理能力，具有完整的增删改查的能力，本质上是一种数据库工具。

**11.HBase的JavaApi操作**

1)创建表



2)插入数据

HTable tab=new HTable(conf,"tabx1");---2-3-4都是这句

Put put = **new** Put("rk1".getBytes());

put.add("cf1".getBytes(), "c1".getBytes(), "v1111".getBytes());

tab.put(put);

3)获取数据--按照行键

Get get = **new** Get("rk1".getBytes());

Result result = tab.get(get);

**byte**[] value = result.getValue("cf1".getBytes(), "c1".getBytes());

String str = **new** String(value);

System.out.println(str);

4)删除数据

Delete del = **new** Delete("rk1".getBytes());

tab.delete(del);

5)删除表 创建HBaseAdmin

admin.disableTable("tabx1".getBytes());

admin.deleteTable("tabx1".getBytes());

**6.扫描全表**

全表扫描:Scan scan = new Scan();

范围扫描:scan.setStartRow("rk3".getBytes());

scan.setStopRow("rk6".getBytes());

**7.过滤器的使用**

详见 <https://blog.csdn.net/cnweike/article/details/42920547>

HBase中只能按照指定行键 行键范围 或 全表扫描来查询数据,而除此之外,HBase还提供了过滤器机制,可以在原有的查询结果的基础上 在服务器端实现进一步的过滤,返回符合过滤条件的数据

HBase支持自定义过滤器,但也同时提供了大量内置的过滤器,可以直接使用

**\*\*1)RowFilter** 行过滤器,可以筛选出匹配的所有的行

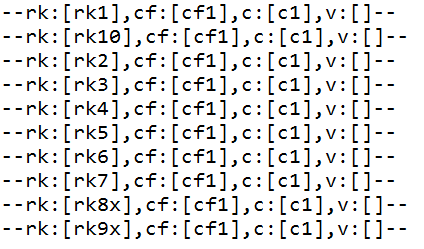
Filter filter = new RowFilter(CompareOp.NOT\_EQUAL, new BinaryComparator("rk3".getBytes()));

Filter filter = **new** RowFilter(CompareOp.EQUAL,**new** RegexStringComparator("^[^1]+1[^1]+$|^.\*x$")

**2)KeyOnlyFilter**

这个过滤器唯一的功能就是只返回每行的行键，值全部为空，这对于只关注于行键的应用场景来说非常合适，这样忽略掉其值就可以减少传递到客户端的数据量，能起到一定的优化作用

 Filter filter = new KeyOnlyFilter();



**3)RandomRowFilter**

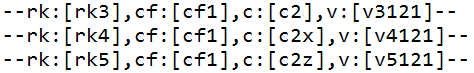
本过滤器的作用就是按照一定的几率（<=0会过滤掉所有的行，>=1会包含所有的行）来返回随机的结果集，对于同样的数据集，多次使用同一个RandomRowFilter会返回不通的结果集，对于需要随机抽取一部分数据的应用场景，可以使用此过滤器

 Filter filter = new RandomRowFilter(0.5f);

**4)CloumnPrefixFiler**

按照列名的前缀来筛选单元格的，如果我们想要对返回的列的前缀加以限制的话，可以使用这个过滤器

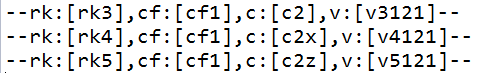
 Filter filter = new ColumnPrefixFilter("c2".getBytes());



**\*\*5)ValueFilter**

按照具体的值来筛选单元格的过滤器，这会把一行中值不能满足的单元格过滤掉

Filter filter = new ValueFilter(CompareOp.EQUAL, new RegexStringComparator("^v[^2].\*2.\*$"));



**\*\*6)SingleRowFilter**

按照指定列的值 决定整行是否返回-----值中不含2

Filter filter = new SingleColumnValueFilter("cf1".getBytes(), "c1".getBytes(), CompareOp.EQUAL, new RegexStringComparator("^[\\w&&[^2]]\*$"));

**\*\*7)Filter List**

用于综合使用多个过滤器。其有两种关系：FilterList.Operator.MUST\_PASS\_ONE和FilterList.Operator.MUST\_PASS\_ALL，默认的是FilterList.Operator.MUST\_PASS\_ALL，顾名思义，它们分别是AND和OR的关系

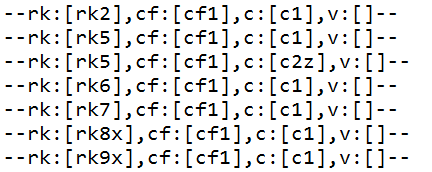
 Filter f1 = new KeyOnlyFilter();

Filter f2 = new RandomRowFilter(0.5f);

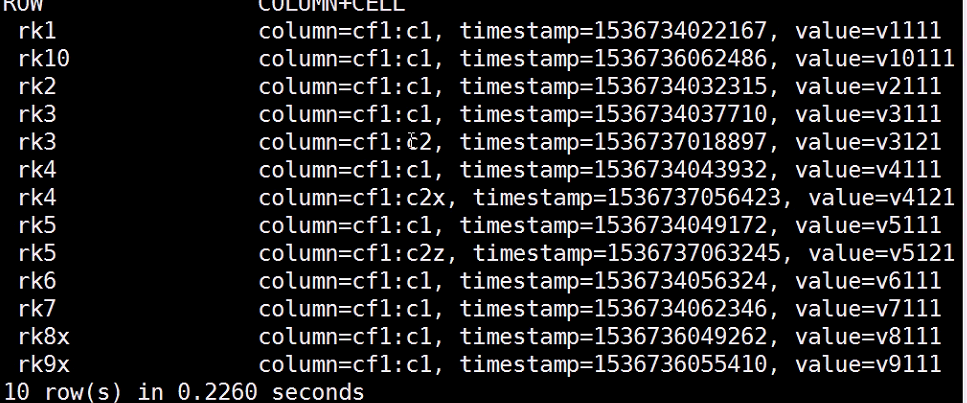
Filter flist = new FilterList(FilterList.Operator.MUST\_PASS\_ALL,f1,f2);

scan.setFilter(flist);

ResultScanner rs = tab.getScanner(scan);



FirstKeyOnlyFilter过滤出第一列的数据



**12.HBase表设计**

1)HBase表设计概述

HBase的表设计将会直接影响 HBase表使用的效率和 便利性 并且 HBase的表的结构一旦确立下来之后很难更改,所以HBase的表是需要设计的

2)HBase表中的列族设计

在设计HBase表时,列族不宜过多,越少越好,官方推荐不要超过3个

经常要一起查询的数据不要放在不同的列族中,尽量减少跨列族的数据访问

如果真的要设计多个列族,要提前考虑好列族数据不均匀的问题

3)HBase表中的行键的设计

hbase表中行键是唯一标识一个表中行的字段，所以行键设计的好不好将会直接影响未来对hbase的查询的性能和查询的便利性

**行键设计的基本原则:**

1)行键必须唯一 必须唯一才能唯一标识数据

2)行键必须有意义 这样才能方便数据的查询

3)行键最好是字符串类型 因为数值类型在不同的系统中处理的方式可能不同

4)行键最好具有固定的长度 不同长度的数据可能会造成自然排序时排序的结果和预期不一致

5)行键不宜过长 行键最多可以达到64KB,但是最好是在10~100字节之间，最好不要超过16字节，越短越好，最好是8字节的整数倍。

**行键设计的最佳实践**:

1)散列原则: 行键的设计将会影响数据在hbase表中的排序方式，这会影响region切分后的结果，要注意，在设计行键时应该让经常被查询的热点数据分散在不同的region中，防止某一个或某几个regionserver成为热点。

2)有序原则: 行键的设计将会影响数据在hbase表中的排序方式，所以一种策略是通过设计行键将将经常连续查询的数据排列在一起，这样一来可以方便批量查询

**案例**

 001      zhang      19      男      zhang**@qq**.com        [www.baidu.com](http://www.baidu.com)       2016-12-20

001       zhang      19      男      zhang**@qq**.com        [www.sina.com](http://www.sina.com)        2016-11-10

001       zhang      19      男      zhang**@qq**.com        [www.souhu.com](http://www.souhu.com)       2016-11-09

002       wang       20      男      wang**@qq**.com        [www.baidu.com](http://www.baidu.com)        2016-12-20

002       wang       20      男      wang**@qq**.com        [www.163.com](http://www.163.com)          2016-12-20

1)列族设计: 一个列族

2)行键设计:

唯一 rand

有意义 rand time host age

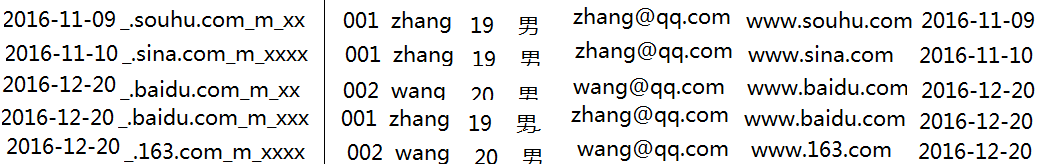
最好是字符串 "rand time host age"

具有固定长度 "rand[28-8]\_time[10]\_host[5-25]\_age[2]"

行键不宜过程 10<48<100 且是8的倍数

散列原则 有序原则 "time[10]\_host[5-25]\_age[2]\_rand[28-8]"

最终结果



**13.Phoenix**

HBase是非关系型的数据库 不满足关系型数据库的范式 所以无法使用sql来进行操作

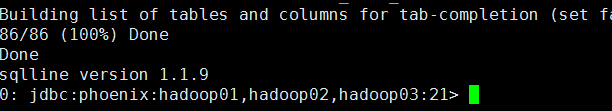
HBase基础上架构的SQL中间件 让使用者 可以通过类sql的语法 操作hbase

让我们可以通过SQL/JDBC来操作HBase

启动

IMG_272

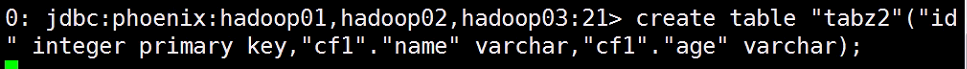
linux最新启动

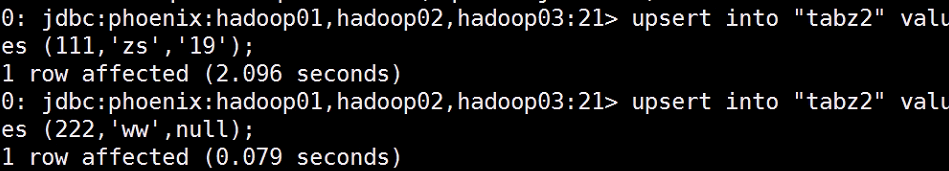


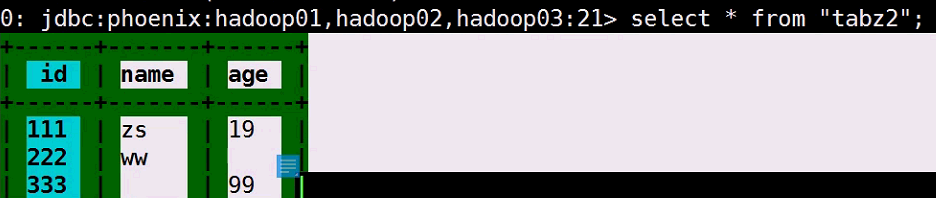
必须有主键

! tables

!desc Tabzi







可以用jdbc操作