Hbase

# 一、Hbase介绍

## 1. google三篇论文gfs\mapreduce\bigtable

- gfs -------- hdfs 分布式文件系统，离线数据存储

- mapreduce -------- mapreduce 分布式并行计算，离线（延迟性高）并行计算

- 移动计算比移动数据更划算---计算向数据靠拢，数据在哪里计算就去哪里

- mapreduce可以查询，但是高延迟的

- bigtable -------- hbase 大表机制，解决存储和数据访问

- 在hadoop生态中弥补了实时这块（是近实时的）

- 通常的实时是毫秒级别的，近实时是秒级的

- hbase使用跳表结构：通过多级索引，而且每个索引范围的数据存储在不同的节点上

- hbase的底层就是布隆过滤器+跳表结构

- hbase原始数据放在hdfs，索引存在hdfs（各个节点共享）

- 顶级索引（最上层的所以）需要一个记录它存储位置的地方，叫索引入口，由zookeeper存储

- hbase最重要依赖zookeeper，存储多份，需要解决数据同步问题，zookeeper解决

# 二、hbase安装和配置

## 1. 主从架构，一主多从

- 主节点：Hmaster; 从节点：Hregionserver

- 规划：安装3个节点

## 2. 安装

### （1）下载、上传、解压

### （2）配置etc/profile

- 配置完成后source etc/profile

- 测试hbase version

### （3）配置hbase/conf下的文件：配置三个，并创建一个

- hbase-env.sh: export JAVA\_HOME=/usr/local/jdk

- Tell HBase whether it should manage it's own instance of Zookeeper or not.

# export HBASE\_MANAGES\_ZK=true

- 如果为true，表示使用hbase自带的zookeeper， 但这个方案只适合单机模式

- 如果为false ，表示不使用hbase自带的，使用自己安装的zookeeper

- hbase-site.xml： hbase的核心配置文件

- hbase在hdfs上的存储路径

- <name>hbase.rootdir</name>

<value>hdfs://bd1803/hbase</value> 给的是命名空间

- 指定hbase是分布式的

- <name>hbase.cluster.distributed</name>

<value>true</value>

- 指定zookeeper的地址，多个用个“，”分割

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

<value>hadoopm:2181,hadoopm:2182,hadoopm:2183</value>

- regionservers: 配置从节点的主机名（只有一台机器，只能用hadoopm）

- hadoopm

- 添加一个文件backup-masters，配置hmaster的备份节点（类似standby master，防止单点故障）

- hadoopm

### （4）将hadoop的hdfs-site.xml和core-site.xml放到hbase的conf目录下面

- 这两个文件在/usr/local/hadoop/etc/hadoop

### （5）将配置好的文件发送到其他节点

## 3. 启动：按下面顺序：

### （1）启动hadoop

- zookeeper: 每个节点都要启动，并检查status

- hdfs：start-dfs.sh

- yarn: start-yarn.sh(多个节点还要再yarn的主节点单独启动resourcemanager: yarn-daemon.sh start resoucemanager)

### （2）启动hbase

- hbase/bin/start-hbase.sh

- 这个命令可以在任意节点执行，在哪个节点执行，就会在哪个节点启动一个Hmaster（active）

- 会多出来Hmaster 和HRegionServer两个进程

- 网页验证：端口16010

- 单独启动hmaster: hbase-daemon.sh start master，

- hbase中可以启动多个Hmaster，但是只有一个是active，剩下的全是backup的

# 三、hbase的特点和相关概念

## 1. hbase的特点

### （1）面向列的NoSQL数据库

- Nosql: 一种是不支持标准的sql（insert\delete\update\select）；另一种说法是not only sql 不仅仅支持sql

- hbase本身不支持sql，phoenix基于hbase，对外提供标准化sql

- nosql数据库特点：如redis、mongodb

- 不支持复杂事务，只支持行级事务

- 不支持多表连接join，想要多表连接，要用hive

- hbase数据库中，数据存储格式是byte(字节)，所以可以存储半结构化（xml\html）和非结构化(视频音频图片)数据

- hbase主要用于存储结构化（快速查询，是近实时的数据库）和半结构化数据（html\xml），也可以存储非结构化数据，但有比hbase更适合存储非结构化数据的数据库，所以一般非结构化数据不会选用hbase数据库

- hbase无模式

- mysql-写模式（增删改，会在写数据的时候进行校验），hive-读模式的数据仓库（在读数据的时候进行校验）

- hbase无严格模式的数据库，既不是读模式，也不是写模式

### （2）hbase存储表的特点：

- 大。hbase的数据最终存储在hdfs上，上层的数据计算依赖mapreduce

- 与hive类似，只不过它可以存储半结构和非结构数据，而hive专门存结构化数据

- 理论上存储数据无上限，可以存千亿级别数据

- 面向列存储，以列为单位存储

- mysql\oracle\sql server都是面向行存储的数据库，一行的数据存储在同一个文件当中

- hbase中一列（列簇）存储为一个文件

- 面向列存储的优点：提升查询性能，减少io

- 原理：关系型数据库做的最多的是对列的查询(select col\_1, col\_2 from …)，面向行存储要进行全表扫描，而面向列存储只需要扫描相应列存储的文件

- 同一列数据类型一般相同，压缩比更大

- hbase的表是稀疏的

- 在存储结构上，mysql中有大量null值存在，占用存储空间；hbase中null值不占用存储空间，直接忽略

- 列对应关系上说，mysql在插入数据时，每行必须写与列数量相等的字段，没有数据的列用null替代；hbase插入时，该列有值的就写，没有的就不写

## 2. hbase中的相关概念

### （1） rowkey

- 一行数据的标志；类似mysql中的id

- 不同行的rowkey不同

- 行键在插入数据的时候指定，可以没有顺序

- 注意点：

- 在hbase中，rowkey不宜过大，通常是10-100字节，最好是16个byte，因为涉及到存储问题

- 原因是每个列簇的物理文件中都会存储一个，过大会占用过多存储空间

- hbase中存储时，默认按照rowkey的字典顺序升序排序

- 5 12 23 2 14 1 31 25 46 按字典顺序升序排序结果

1 12 14 2 23 25 31 46 5

- 要保持整型的自然序，行键必须用0做左填充

- hbase中数据查询的三种方式

- 全表扫描：效率低

- 通过rowkey的范围查询（按字典升序排序）

- 通过单个rowkey查询数据

### （2）列簇

- 包含一个或多个列

- 列簇是在建表的时候指定

- 列名和列的值是在数据插入的时候指定

- 注意点：

- 通常情况下，一个表中列簇个数不要超过3个，因为列簇是hbase存储的无力切分单位，hbase存储是面向列簇的，每个列簇存储为一个文件

- 什么样的列会划分到一个列簇中？

- 具有相同io特性的列（通常情况下会一起访问的列）一般放到一个列簇中

- 如果一个表字段和io特性多，将一部分不同的io特性列强制分到一个列簇中

- 实际生产中通常分1-2个列簇，很少为3个。列簇越多，存储的文件会越多，扫描次数越多

### （3）时间戳

- 插入数据时，每插入一条数据，会自动生成一个时间戳

- 作用是记录数据的版本，因此在hbase中可以存储多个版本的数据

- 时间戳默认是系统时间（精确到ms），也可以手动指定

### （4）单元格

- 某一条（行）数据中某一列的某一个值

- 在hbase中要定位一个单元格需要：行键+列簇+时间戳

- ｛rowkey, column(=<family>+<column>),version(timestamp)｝

# 四、hbase的架构和存储

## 1. hbase的架构：主从架构

- 主：hmaster; 从：hregionserver

- region:hbase在行方向上的逻辑切分概念（列簇：在列方向上的物理切分概念）

- 寻址路径：存在zookeeper中

## 2. hbase中的存储

- region:hbase在行方向上的逻辑切分概念（列簇：在列方向上的物理切分概念）

- 逻辑切分的作用是便于建查询索引

- 一个表数据比较少的时候开始只有一个region,数据增加region会进行分裂

- 分裂时按行方向分裂（1-2-4-8），从原region的中间开始分

- region分裂时按照文件大小分，hbase0.9版本中256M, 1.x中调到10G

# 五、hbase中shell操作

## 0. zookeeper操作

- 启动：zkServer.sh start

- 察看状态：zkServer.sh status

- 停止：zkServer.sh stop

- 重启：zkServer.sh restart

## 1. 进入hbase-shell

- 启动hbase服务: start-hbase.sh

- /usr/local/hbase/bin/，运行hbase shell

- hbase中，弱化了库的概念，称为namespace命名空间，建命名空间相当于建库

- quit: 退出hbase

- 停止hbase服务：stop-hbase.sh

## 2. 查询命令

- > help：帮助。包括shell命令的解析。

- help ‘命令’：解析具体某个命令

- 创建命名空间： create\_namespace 名称;

- list: 察看表

- describe ‘表名’: 察看表的描述信息

- 包含主要列簇信息：列簇名name、布隆过滤器bloomfilter、保存的版本version（默认保存1个）、是否内存存储in\_memory、TTL(生命周期，默认forever表示永久保存)、是否压缩compressioin、块缓存blockchche(模式开启true)、块大小(默认128M)

- version:如果建表的时候不指定保存版本数（同一条数据的），默认为1，而且是最新版本（时间戳最大的版本）。如果指定保存版本数，表示最多保存的最新版本数

- TTL生命周期是数据保存的时间，如果超过这个周期，数据会被自动删除。默认是forever永久保存。建表的时候可以指定（单位是秒），按数据插入时间开始计时

## 3. 表操作

### （1）建表

- 建表：

- create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'} 或 create ‘表名’,’列簇1’,’列簇2’；

- t1：表名

- f\*: 列簇（建表时指定，列名是插入数据时指定）

- create ‘test01’, {NAME=>’info’},{ NAME=>’score’}

Create ‘test01’,’info’,’score’

- 建表的时候可以修改属性信息，放在花括号里面

- 修改版本信息：create 't1', {NAME => 'f1'，versions=>’3’}, {NAME => 'f2'，versioin=>’2’}

- 向表中插入数据put

- put ‘表名’,‘行键名’,’列簇:列名’,’列的值’

- 行键相同的会认为是同一行

- put ‘test02’, ‘rk001’,’info:name’,’jialefu’

- 执行写入命令之后，数据是先写入缓存中，关闭hbase后重启才能在网页中看到数据，在hbase关闭的时候缓存中的数据会flush

- 数据存储在hdfs的 /hbase/data/命名空间/表名/region编号/列簇名/列簇存储的文件（Hfile，对应的是一个列簇）

### （2）删除表和数据

- 删除数据delete

- hbase> delete '表名', '列簇', 'column', timestamp

hbase> delete 't1', 'r1', 'c1', ts1, {VISIBILITY=>'PRIVATE|SECRET'}

- 清空表数据truncate

- truncate ‘表名’：过程是先禁用后清空，然后又启用

- 删除表：先禁用，在删除drop

- disable ‘表名’

- drop ‘表名’

### （3）修改表

- alter命令：修改数据

- alter '表名', NAME => '列簇名', VERSIONS => 3 修改存储版本数

- 列簇存在的情况下，执行后面的版本数操作

- 如果列簇不存在，则添加一个列簇，并执行版本数设置

- alter 't1', 'f1', {NAME => 'f2', IN\_MEMORY =>true}, {NAME => 'f3', VERSIONS => 5}

- 列簇一旦创建，列簇名修改不了；如果想修改名字，先删除后重新添加

### （4）查询

- 全表扫描某一个表：scan ‘表名’，类似select \*

- 数据查询get方式

hbase> get 't1', 'r1' 表名+行键查询

hbase> get 't1', 'r1', {TIMERANGE => [ts1, ts2]} 表名+行键+时间戳的范围

hbase> get 't1', 'r1', {COLUMN => 'c1'}

hbase> get 't1', 'r1', {COLUMN => ['c1', 'c2', 'c3']}

hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1}

hbase> get 't1', 'r1', {COLUMN => 'c1', TIMERANGE => [ts1, ts2], VERSIONS => 4}

hbase> get 't1', 'r1', {COLUMN => 'c1', TIMESTAMP => ts1, VERSIONS => 4}

hbase> get 't1', 'r1', {FILTER => "ValueFilter(=, 'binary:abc')"}

hbase> get 't1', 'r1', 'c1'

hbase> get 't1', 'r1', 'c1', 'c2'

hbase> get 't1', 'r1', ['c1', 'c2']

hbase> get 't1', 'r1', {COLUMN => 'c1', ATTRIBUTES => {'mykey'=>'myvalue'}}

hbase> get 't1', 'r1', {COLUMN => 'c1', AUTHORIZATIONS => ['PRIVATE','SECRET']}

hbase> get 't1', 'r1', {CONSISTENCY => 'TIMELINE'}

hbase> get 't1', 'r1', {CONSISTENCY => 'TIMELINE', REGION\_REPLICA\_ID => 1}

* 数据查询scan方式

hbase> scan 'hbase:meta'

hbase> scan 'hbase:meta', {COLUMNS => 'info:regioninfo'}

hbase> scan 'ns1:t1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'} limit设置查询条数

hbase> scan 't1', {COLUMNS => ['c1', 'c2'], LIMIT => 10, STARTROW => 'xyz'}

hbase> scan 't1', {COLUMNS => 'c1', TIMERANGE => [1303668804, 1303668904]}

hbase> scan 't1', {REVERSED => true}

hbase> scan 't1', {ALL\_METRICS => true}

hbase> scan 't1', {METRICS => ['RPC\_RETRIES', 'ROWS\_FILTERED']}

hbase> scan 't1', {ROWPREFIXFILTER => 'row2', FILTER => "(QualifierFilter (>=, 'binary:xyz')) AND (TimestampsFilter ( 123, 456))"}

hbase> scan 't1', {FILTER =>org.apache.hadoop.hbase.filter.ColumnPaginationFilter.new(1, 0)}

hbase> scan 't1', {CONSISTENCY => 'TIMELINE'}

# 六、hbase中API操作（需要java编程）

## 1.管理对象

- hbaseconfiguration: hbase的配置文件对象

- hbaseadmin/admin: ddl操作句柄，ddl语句的管理对象

- htable/table: 表数据的管理对象

- hTableDescriptor: hbase列簇描述对象

- hTableColumnDescriptor: 列描述对象

## 2. 基本操作对象

- put

- get

- delete

- scan

## 3. 结果管理对象

- result:

- resultScanner

## 4. 其他

- key-value

- cell

# 七、hbase中的过滤器

## 1. 过滤器中常用的比较运算符

- greater >； greater equal >=； equal==； less <； less equal <=； not equal !=； no\_op 排除所有

## 2. 比较机制：使用的比较方式

- BinaryComparator: 按字节索引顺序比较指定字节数组，通常用于文本

- BinaryPrefixComparator：用法与上面的相同，但是只比较左端数据是否相同

- BitComparator: 按位比较

- NullComparator: 判断给定的是否为空

- RegexStringComparator：提供一个正则比较器，仅支持equal和not equal

- SubStringComparator：判断提供的子串是否出现在value中

## 3. 过滤器的用法

- 过滤器是在表扫描scan操作时添加

## 4. 常用过滤器分类

- 列簇过滤器

- 列过滤器

- 时间戳过滤器

- 专用过滤器

- 单列值过滤器

- 前缀过滤器

- 列前缀过滤器

- 分页过滤器

# 八、hbase与hdfs的数据互通

## 1. hdfs数据量大，但延迟高，所以可以把hdfs中的数据放在hbase中

## 2. 过程

- map端：读取hdfs数据，发送到reduce端

- reduce端：读取map端发过来的数据，写入hbase

# 九、hbase和hive的整合

## 0. 原理

- hive与hbase利用两者本身对外的API来实现整合，主要是靠HBaseStorageHandler进行通信：hive可以获得hive表对应的hbase表名、列簇和列，inputformat和outputformat类，创建和删除hbase表等

## 1. hive端操作

- 相关配置操作

- set hbase.zookeeper.quorum=hadoopm:2181,hadoopm:2182,hadoopm:2183; 指定hbase所使用的zookeeper集群地址，默认端口2181，可以不写

- set zookeeper.znode.parent=/hbase; 指定hbase在zookeeper中所使用的根目录

- add jar /usr/local/hive/lib/hive-hbase-handler-2.3.4.jar; 加入指定的处理jar包

- 创建基于hbase的hive表

- hive> create external table mingxing(rowkey string, base\_info map<string,string>,extra\_info map<string,string>)

> row format delimited fields terminated by '\t'

> stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' hbase和hive转换的处理器

> with serdeproperties('habase.columns.mapping'=':key,base\_info,extra\_info:') hbase的列簇和列到hive的映射关系

> tblproperties('hbase.table.name'='mingxing'); 定义hbase表名