大数据技术之HBase

版本：V1.3

# HBase简介

## HBase定义

HBase是一种分布式、可扩展、支持海量数据存储的NoSQL数据库。

## HBase数据模型

逻辑上，HBase的数据模型同关系型数据库很类似，数据存储在一张表中，有行有列。但从HBase的底层物理存储结构（K-V）来看，HBase更像是一个multi-dimensional map。

### HBase逻辑结构



### HBase物理存储结构



### 数据模型

* + - 1. Name Space

命名空间，类似于关系型数据库的DatabBase概念，每个命名空间下有多个表。HBase有两个自带的命名空间，分别是hbase和default，hbase中存放的是HBase内置的表，default表是用户默认使用的命名空间。

* + - 1. Region

类似于关系型数据库的表概念。不同的是，HBase定义表时只需要声明列族即可，不需要声明具体的列。这意味着，往HBase写入数据时，字段可以动态、按需指定。因此，和关系型数据库相比，HBase能够轻松应对字段变更的场景。

* + - 1. Row

HBase表中的每行数据都由一个**RowKey**和多个**Column**（列）组成，数据是按照RowKey的字典顺序存储的，并且查询数据时只能根据RowKey进行检索，所以RowKey的设计十分重要。

* + - 1. Column

HBase中的每个列都由Column Family(列族)和Column Qualifier（列限定符）进行限定，例如info：name，info：age。建表时，只需指明列族，而列限定符无需预先定义。

* + - 1. Time Stamp

用于标识数据的不同版本（version），每条数据写入时，如果不指定时间戳，系统会自动为其加上该字段，其值为写入HBase的时间。

* + - 1. Cell

由{rowkey, column Family：column Qualifier, time Stamp} 唯一确定的单元。cell中的数据是没有类型的，全部是字节数组形式存贮。

## HBase基本架构



架构角色：

* + - 1. Region Server

Region Server为 Region的管理者，其实现类为HRegionServer，主要作用如下:

对于数据的操作：get, put, delete；

对于Region的操作：splitRegion、compactRegion。

* + - 1. Master

Master是所有Region Server的管理者，其实现类为HMaster，主要作用如下：

对于表的操作：create, delete, alter

对于RegionServer的操作：分配regions到每个RegionServer，监控每个RegionServer的状态，负载均衡和故障转移。

* + - 1. Zookeeper

HBase通过Zookeeper来做Master的高可用、RegionServer的监控、元数据的入口以及集群配置的维护等工作。

* + - 1. HDFS

HDFS为HBase提供最终的底层数据存储服务，同时为HBase提供高可用的支持。

# HBase快速入门

## HBase安装部署

### Zookeeper正常部署

首先保证Zookeeper集群的正常部署，并启动之：

[atguigu@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh start

[atguigu@hadoop103 zookeeper-3.5.7]$ bin/zkServer.sh start

[atguigu@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh start

### Hadoop正常部署

Hadoop集群的正常部署并启动：

[atguigu@hadoop102 hadoop-3.1.3]$ sbin/start-dfs.sh

[atguigu@hadoop103 hadoop-3.1.3]$ sbin/start-yarn.sh

### HBase的解压

解压Hbase到指定目录：

[atguigu@hadoop102 software]$ tar -zxvf hbase-2.0.5-bin.tar.gz -C /opt/module

### HBase的配置文件

修改HBase对应的配置文件。

* + - 1. hbase-env.sh修改内容：

export HBASE\_MANAGES\_ZK=false

* + - 1. hbase-site.xml修改内容：

<configuration>

<property>

<name>hbase.rootdir</name>

<value>hdfs://hadoop102:8020/hbase</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

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

<value>hadoop102,hadoop103,hadoop104</value>

</property>

<property>

<name>hbase.unsafe.stream.capability.enforce</name>

<value>false</value>

</property>

<property>

<name>hbase.wal.provider</name>

<value>filesystem</value>

</property>

</configuration>

* + - 1. regionservers：

hadoop102

hadoop103

hadoop104

* + - 1. 软连接hadoop配置文件到HBase：

[atguigu@hadoop102 module]$ ln -s /opt/module/hadoop-3.1.3/etc/hadoop/core-site.xml /opt/module/hbase/conf/core-site.xml

[atguigu@hadoop102 module]$ ln -s /opt/module/hadoop-3.1.3/etc/hadoop/hdfs-site.xml /opt/module/hbase/conf/hdfs-site.xml

### HBase远程发送到其他集群

[atguigu@hadoop102 module]$ xsync hbase/

### HBase服务的启动

* + - 1. 启动方式

[atguigu@hadoop102 hbase]$ bin/hbase-daemon.sh start master

[atguigu@hadoop102 hbase]$ bin/hbase-daemon.sh start regionserver

提示：如果集群之间的节点时间不同步，会导致regionserver无法启动，抛出ClockOutOfSyncException异常。

修复提示：

a、同步时间服务

请参看帮助文档：《尚硅谷大数据技术之Hadoop入门》

b、属性：hbase.master.maxclockskew设置更大的值

|  |
| --- |
| <property>  <name>hbase.master.maxclockskew</name>  <value>180000</value>  <description>Time difference of regionserver from master</description>  </property> |

* + - 1. 启动方式2

[atguigu@hadoop102 hbase]$ bin/start-hbase.sh

对应的停止服务：

[atguigu@hadoop102 hbase]$ bin/stop-hbase.sh

### 查看HBase页面

启动成功后，可以通过“host:port”的方式来访问HBase管理页面，例如：

[http://hadoop102:16010](http://linux01:16010)

### 高可用(可选)

在HBase中HMaster负责监控HRegionServer的生命周期，均衡RegionServer的负载，如果HMaster挂掉了，那么整个HBase集群将陷入不健康的状态，并且此时的工作状态并不会维持太久。所以HBase支持对HMaster的高可用配置。

* + - 1. 关闭HBase集群（如果没有开启则跳过此步）

[atguigu@hadoop102 hbase]$ bin/stop-hbase.sh

* + - 1. 在conf目录下创建backup-masters文件

[atguigu@hadoop102 hbase]$ touch conf/backup-masters

* + - 1. 在backup-masters文件中配置高可用HMaster节点

[atguigu@hadoop102 hbase]$ echo hadoop103 > conf/backup-masters

* + - 1. 将整个conf目录scp到其他节点

[atguigu@hadoop102 hbase]$ scp -r conf/ hadoop103:/opt/module/hbase/

[atguigu@hadoop102 hbase]$ scp -r conf/ hadoop104:/opt/module/hbase/

* + - 1. 打开页面测试查看

[http://hadooo102:16010](http://linux01:16010)

## HBase Shell操作

### 基本操作

* + - 1. 进入HBase客户端命令行

[atguigu@hadoop102 hbase]$ bin/hbase shell

* + - 1. 查看帮助命令

hbase(main):001:0> help

* + - 1. 查看当前数据库中有哪些表

hbase(main):002:0> list

### 表的操作

* + - 1. 创建表

hbase(main):002:0> create 'student','info'

* + - 1. 插入数据到表

hbase(main):003:0> put 'student','1001','info:sex','male'

hbase(main):004:0> put 'student','1001','info:age','18'

hbase(main):005:0> put 'student','1002','info:name','Janna'

hbase(main):006:0> put 'student','1002','info:sex','female'

hbase(main):007:0> put 'student','1002','info:age','20'

* + - 1. 扫描查看表数据

hbase(main):008:0> scan 'student'

hbase(main):009:0> scan 'student',{STARTROW => '1001', STOPROW => '1001'}

hbase(main):010:0> scan 'student',{STARTROW => '1001'}

* + - 1. 查看表结构

hbase(main):011:0> describe ‘student’

* + - 1. 更新指定字段的数据

hbase(main):012:0> put 'student','1001','info:name','Nick'

hbase(main):013:0> put 'student','1001','info:age','100'

* + - 1. 查看“指定行”或“指定列族:列”的数据

hbase(main):014:0> get 'student','1001'

hbase(main):015:0> get 'student','1001','info:name'

* + - 1. 统计表数据行数

hbase(main):021:0> count 'student'

* + - 1. 删除数据

删除某rowkey的全部数据：

hbase(main):016:0> deleteall 'student','1001'

删除某rowkey的某一列数据：

hbase(main):017:0> delete 'student','1002','info:sex'

* + - 1. 清空表数据

hbase(main):018:0> truncate 'student'

提示：清空表的操作顺序为先disable，然后再truncate。

* + - 1. 删除表

首先需要先让该表为disable状态：

hbase(main):019:0> disable 'student'

然后才能drop这个表：

hbase(main):020:0> drop 'student'

提示：如果直接drop表，会报错：ERROR: Table student is enabled. Disable it first.

* + - 1. 变更表信息

将info列族中的数据存放3个版本：

hbase(main):022:0> alter 'student',{NAME=>'info',VERSIONS=>3}

hbase(main):022:0> get 'student','1001',{COLUMN=>'info:name',VERSIONS=>3}

# HBase进阶

## 架构原理



* + - 1. StoreFile

保存实际数据的物理文件，StoreFile以HFile的形式存储在HDFS上。每个Store会有一个或多个StoreFile（HFile），数据在每个StoreFile中都是有序的。

* + - 1. MemStore

写缓存，由于HFile中的数据要求是有序的，所以数据是先存储在MemStore中，排好序后，等到达刷写时机才会刷写到HFile，每次刷写都会形成一个新的HFile。

* + - 1. WAL

由于数据要经MemStore排序后才能刷写到HFile，但把数据保存在内存中会有很高的概率导致数据丢失，为了解决这个问题，数据会先写在一个叫做Write-Ahead logfile的文件中，然后再写入MemStore中。所以在系统出现故障的时候，数据可以通过这个日志文件重建。

## 写流程



写流程：

* + - * 1. Client先访问zookeeper，获取hbase:meta表位于哪个Region Server。
        2. 访问对应的Region Server，获取hbase:meta表，根据读请求的namespace:table/rowkey，查询出目标数据位于哪个Region Server中的哪个Region中。并将该table的region信息以及meta表的位置信息缓存在客户端的meta cache，方便下次访问。
        3. 与目标Region Server进行通讯；
        4. 将数据顺序写入（追加）到WAL；
        5. 将数据写入对应的MemStore，数据会在MemStore进行排序；
        6. 向客户端发送ack；
        7. 等达到MemStore的刷写时机后，将数据刷写到HFile。

## MemStore Flush



MemStore刷写时机：

* + - * 1. 当某个memstroe的大小达到了**hbase.hregion.memstore.flush.size（默认值128M）**，其所在region的所有memstore都会刷写。

当memstore的大小达到了

hbase.hregion.memstore.flush.size（默认值128M）

\* hbase.hregion.memstore.block.multiplier（默认值4）

时，会阻止继续往该memstore写数据。

* + - * 1. 当region server中memstore的总大小达到

java\_heapsize

\*hbase.regionserver.global.memstore.size（默认值0.4）

\*hbase.regionserver.global.memstore.size.upper.limit（默认值0.95），

region会按照其所有memstore的大小顺序（由大到小）依次进行刷写。直到region server中所有memstore的总大小减小到**hbase.regionserver.global.memstore.size.lower.limit**以下。

当region server中memstore的总大小达到

java\_heapsize\*hbase.regionserver.global.memstore.size（默认值0.4）

时，会阻止继续往所有的memstore写数据。

* + - * 1. 到达自动刷写的时间，也会触发memstore flush。自动刷新的时间间隔由该属性进行配置hbase.regionserver.optionalcacheflushinterval（默认1小时）。
        2. 当WAL文件的数量超过**hbase.regionserver.max.logs**，region会按照时间顺序依次进行刷写，直到WAL文件数量减小到**hbase.regionserver.max.log**以下（该属性名已经废弃，现无需手动设置，最大值为32）。

## 读流程



读流程

* + - * 1. Client先访问zookeeper，获取hbase:meta表位于哪个Region Server。
        2. 访问对应的Region Server，获取hbase:meta表，根据读请求的namespace:table/rowkey，查询出目标数据位于哪个Region Server中的哪个Region中。并将该table的region信息以及meta表的位置信息缓存在客户端的meta cache，方便下次访问。
        3. 与目标Region Server进行通讯；
        4. 分别在Block Cache（读缓存），MemStore和Store File（HFile）中查询目标数据，并将查到的所有数据进行合并。此处所有数据是指同一条数据的不同版本（time stamp）或者不同的类型（Put/Delete）。
        5. 将从文件中查询到的数据块（Block，HFile数据存储单元，默认大小为64KB）缓存到Block Cache。
        6. 将合并后的最终结果返回给客户端。

## StoreFile Compaction

由于memstore每次刷写都会生成一个新的HFile，且同一个字段的不同版本（timestamp）和不同类型（Put/Delete）有可能会分布在不同的HFile中，因此查询时需要遍历所有的HFile。为了减少HFile的个数，以及清理掉过期和删除的数据，会进行StoreFile Compaction。

Compaction分为两种，分别是Minor Compaction和Major Compaction。Minor Compaction会将临近的若干个较小的HFile合并成一个较大的HFile，但**不会**清理过期和删除的数据。Major Compaction会将一个Store下的所有的HFile合并成一个大HFile，并且**会**清理掉过期和删除的数据。



## Region Split

默认情况下，每个Table起初只有一个Region，随着数据的不断写入，Region会自动进行拆分。刚拆分时，两个子Region都位于当前的Region Server，但处于负载均衡的考虑，HMaster有可能会将某个Region转移给其他的Region Server。

Region Split时机：

* + - * 1. 当1个region中的某个Store下所有StoreFile的总大小超过hbase.hregion.max.filesize，该Region就会进行拆分（0.94版本之前）。
        2. 当1个region中的某个Store下所有StoreFile的总大小超过Min(R^2 \* "hbase.hregion.memstore.flush.size",hbase.hregion.max.filesize")，该Region就会进行拆分，其中R为当前Region Server中属于该Table的个数（0.94版本之后）。



# 整合Phoenix

## Phoenix简介

### Phoenix定义

Phoenix是HBase的开源SQL皮肤。可以使用标准JDBC API代替HBase客户端API来创建表，插入数据和查询HBase数据。

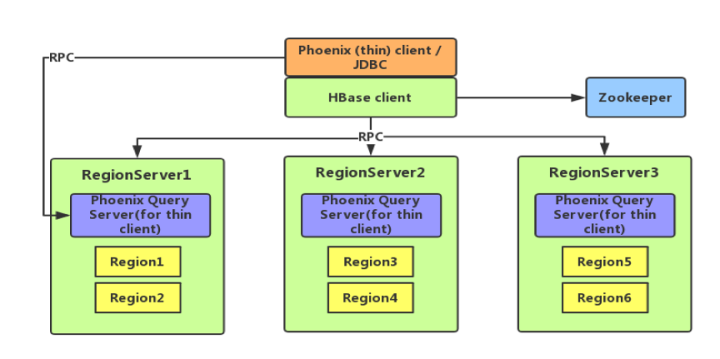
### Phoenix特点

1）容易集成：如Spark，Hive，Pig，Flume和Map Reduce；

2）操作简单：DML命令以及通过DDL命令创建和操作表和版本化增量更改；

3）支持HBase二级索引创建。

### Phoenix架构



## Phoenix快速入门

### 安装

* + - 1. 官网地址

<http://phoenix.apache.org/>

* + - 1. Phoenix部署

0）安装bsdtar3

sudo yum install -y epel-release

sudo yum install -y bsdtar3

1）上传并解压tar包（这里可能会报不识别的文件头，忽略即可，或者可以改用bsdtar）

[atguigu@hadoop101 module]$ tar -zxvf /opt/software/apache-phoenix-5.0.0-HBase-2.0-bin.tar.gz -C /opt/module

[atguigu@hadoop101 module]$ mv apache-phoenix-5.0.0-HBase-2.0-bin phoenix

2）复制server包并拷贝到各个节点的hbase/lib

[atguigu@hadoop102 module]$ cd /opt/module/phoenix/

[atguigu@hadoop102 phoenix]$ cp phoenix-5.0.0-HBase-2.0-server.jar /opt/module/hbase/lib/

[atguigu@hadoop102 phoenix]$ scp phoenix-5.0.0-HBase-2.0-server.jar hadoop103:/opt/module/hbase/lib/

[atguigu@hadoop102 phoenix]$ scp phoenix-5.0.0-HBase-2.0-server.jar hadoop104:/opt/module/hbase/lib/

3）复制client包并拷贝到各个节点的hbase/lib

[atguigu@hadoop102 module]$ cd /opt/module/phoenix/

[atguigu@hadoop102 phoenix]$ cp phoenix-5.0.0-HBase-2.0-client.jar /opt/module/hbase/lib/

[atguigu@hadoop102 phoenix]$ scp phoenix-5.0.0-HBase-2.0-client.jar hadoop103:/opt/module/hbase/lib/

[atguigu@hadoop102 phoenix]$ scp phoenix-5.0.0-HBase-2.0-client.jar hadoop104:/opt/module/hbase/lib/

4）配置环境变量

#phoenix

export PHOENIX\_HOME=/opt/module/phoenix

export PHOENIX\_CLASSPATH=$PHOENIX\_HOME

export PATH=$PATH:$PHOENIX\_HOME/bin

5）启动Phoenix

[atguigu@hadoop101 phoenix]$ /opt/module/phoenix/bin/sqlline.py hadoop102,hadoop103,hadoop104:2181

### Phoenix Shell操作

* + - 1. 表的操作

1）显示所有表

!table 或 !tables

2）创建表

直接指定单个列作为RowKey

CREATE TABLE IF NOT EXISTS student(

id VARCHAR primary key,

name VARCHAR;

在phoenix中，表名等会自动转换为大写，若要小写，使用双引号，如"us\_population"。

指定多个列的联合作为RowKey

CREATE TABLE IF NOT EXISTS us\_population (

State CHAR(2) NOT NULL,

City VARCHAR NOT NULL,

Population BIGINT

CONSTRAINT my\_pk PRIMARY KEY (state, city));

3）插入数据

upsert into student values('1001','zhangsan');

4）查询记录

select \* from student;

select \* from student where id='1001';

5）删除记录

delete from student where id='1001';

6）删除表

drop table student;

7）退出命令行

!quit

* + - 1. 表的映射

1）表的关系

默认情况下，直接在HBase中创建的表，通过Phoenix是查看不到的。如果要在Phoenix中操作直接在HBase中创建的表，则需要在Phoenix中进行表的映射。映射方式有两种：视图映射和表映射。

2）命令行中创建表test

HBase 中test的表结构如下，两个列族info1、info2。

|  |  |  |
| --- | --- | --- |
| Rowkey | info1 | info2 |
| id | name | address |

启动HBase Shell

[atguigu@hadoop101 ~]$ /opt/module/hbase-1.3.1/bin/hbase shell

创建HBase表test

hbase(main):001:0> create 'test','info1','info2'

3）视图映射

Phoenix创建的视图是只读的，所以只能用来做查询，无法通过视图对源数据进行修改等操作。在phoenix中创建关联test表的视图

0: jdbc:phoenix:hadoop101,hadoop102,hadoop103> create view "test"(id varchar primary key,"info1"."name" varchar, "info2"."address" varchar);

删除视图

0: jdbc:phoenix:hadoop101,hadoop102,hadoop103> drop view "test";

4）表映射

使用Apache Phoenix创建对HBase的表映射，有两种方法：

（1）HBase中不存在表时，可以直接使用create table指令创建需要的表,系统将会自动在Phoenix和HBase中创建person\_infomation的表，并会根据指令内的参数对表结构进行初始化。

（2）当HBase中已经存在表时，可以以类似创建视图的方式创建关联表，只需要将create view改为create table即可。

0: jdbc:phoenix:hadoop101,hadoop102,hadoop103> create table "test"(id varchar primary key,"info1"."name" varchar, "info2"."address" varchar) column\_encoded\_bytes=0;

### Phoenix JDBC操作

* + - 1. 创建项目并导入依赖

<dependencies>

<dependency>

<groupId>org.apache.phoenix</groupId>

<artifactId>phoenix-core</artifactId>

<version>5.0.0-HBase-2.0</version>

</dependency>

<dependency>  
 <groupId>com.lmax</groupId>  
 <artifactId>disruptor</artifactId>  
 <version>3.3.6</version>  
</dependency>

</dependencies>

* + - 1. 编写代码

package com.atguigu;

import java.sql.\*;

public class PhoenixTest {

public static void main(String[] args) throws ClassNotFoundException, SQLException {

//1.定义参数

String driver = "org.apache.phoenix.jdbc.PhoenixDriver";

String url = "jdbc:phoenix:hadoop102,hadoop103,hadoop104:2181";

//2.加载驱动

Class.forName(driver);

//3.创建连接

Connection connection = DriverManager.getConnection(url);

//4.预编译SQL

PreparedStatement preparedStatement = connection.prepareStatement("SELECT \* FROM test");

//5.查询获取返回值

ResultSet resultSet = preparedStatement.executeQuery();

//6.打印结果

while (resultSet.next()) {

System.out.println(resultSet.getString(1) + resultSet.getString(2));

}

//7.关闭资源

resultSet.close();

preparedStatement.close();

connection.close();

}

}

## Phoenix二级索引

### HBase协处理器

* + - 1. 案例需求

编写协处理器，实现在往A表插入数据的同时让HBase自身（协处理器）向B表中插入一条数据。

* + - 1. 实现步骤

1）创建一个maven项目，并引入以下依赖。

<dependencies>

<dependency>

<groupId>org.apache.hbase</groupId>

<artifactId>hbase-client</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>org.apache.hbase</groupId>

<artifactId>hbase-server</artifactId>

<version>1.3.1</version>

</dependency>

</dependencies>

2）定义FruitTableCoprocessor类并继承BaseRegionObserver类

package com.atguigu;

import org.apache.hadoop.hbase.HBaseConfiguration;

import org.apache.hadoop.hbase.TableName;

import org.apache.hadoop.hbase.client.\*;

import org.apache.hadoop.hbase.coprocessor.BaseRegionObserver;

import org.apache.hadoop.hbase.coprocessor.ObserverContext;

import org.apache.hadoop.hbase.coprocessor.RegionCoprocessorEnvironment;

import org.apache.hadoop.hbase.regionserver.wal.WALEdit;

import java.io.IOException;

public class FruitTableCoprocessor extends BaseRegionObserver {

@Override

public void postPut(ObserverContext<RegionCoprocessorEnvironment> e, Put put, WALEdit edit, Durability durability) throws IOException {

//获取连接

Connection connection = ConnectionFactory.createConnection(HBaseConfiguration.create());

//获取表对象

Table table = connection.getTable(TableName.valueOf("fruit"));

//插入数据

table.put(put);

//关闭资源

table.close();

connection.close();

}

}

### 二级索引配置文件

* + - 1. 添加如下配置到HBase的HRegionserver节点的hbase-site.xml

<!-- phoenix regionserver 配置参数-->

<property>

<name>hbase.regionserver.wal.codec</name>

<value>org.apache.hadoop.hbase.regionserver.wal.IndexedWALEditCodec</value>

</property>

<property>

<name>hbase.region.server.rpc.scheduler.factory.class</name>

<value>org.apache.hadoop.hbase.ipc.PhoenixRpcSchedulerFactory</value>

<description>Factory to create the Phoenix RPC Scheduler that uses separate queues for index and metadata updates</description>

</property>

<property>

<name>hbase.rpc.controllerfactory.class</name>

<value>org.apache.hadoop.hbase.ipc.controller.ServerRpcControllerFactory</value>

<description>Factory to create the Phoenix RPC Scheduler that uses separate queues for index and metadata updates</description>

</property>

* + - 1. 添加如下配置到HBase中HMaster节点的hbase-site.xml中

<!-- phoenix master 配置参数 -->

<property>

<name>hbase.master.loadbalancer.class</name>

<value>org.apache.phoenix.hbase.index.balancer.IndexLoadBalancer</value>

</property>

<property>

<name>hbase.coprocessor.master.classes</name>

<value>org.apache.phoenix.hbase.index.master.IndexMasterObserver</value>

</property>

### 全局二级索引

Global Index是默认的索引格式，创建全局索引时，会在HBase中建立一张新表。也就是说索引数据和数据表是存放在不同的表中的，因此全局索引适用于多读少写的业务场景。

写数据的时候会消耗大量开销，因为索引表也要更新，而索引表是分布在不同的数据节点上的，跨节点的数据传输带来了较大的性能消耗。

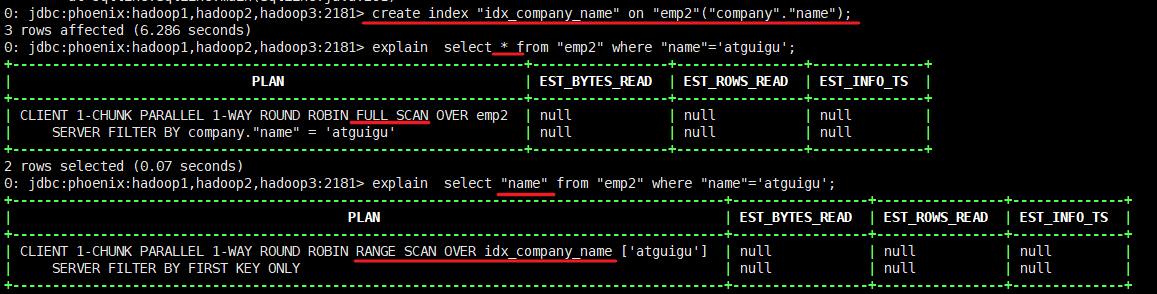
在读数据的时候Phoenix会选择索引表来降低查询消耗的时间。

1）创建单个字段的全局索引

CREATE INDEX my\_index ON my\_table (my\_col);



如果想查询的字段不是索引字段的话索引表不会被使用，也就是说不会带来查询速度的提升。



2）创建携带其他字段的全局索引

CREATE INDEX my\_index ON my\_table (v1) INCLUDE (v2);



### 本地二级索引

Local Index适用于写操作频繁的场景。

索引数据和数据表的数据是存放在同一张表中（且是同一个Region），避免了在写操作的时候往不同服务器的索引表中写索引带来的额外开销。查询的字段不是索引字段索引表也会被使用，这会带来查询速度的提升。

CREATE LOCAL INDEX my\_index ON my\_table (my\_column);



新建项目后在pom.xml中添加依赖：

|  |
| --- |
| <dependency>  <groupId>org.apache.hbase</groupId>  <artifactId>hbase-server</artifactId>  <version>1.3.1</version>  </dependency>  <dependency>  <groupId>org.apache.hbase</groupId>  <artifactId>hbase-client</artifactId>  <version>1.3.1</version>  </dependency> |

# 与Hive的集成

## HBase与Hive的对比

1．Hive

(1) 数据仓库

Hive的本质其实就相当于将HDFS中已经存储的文件在Mysql中做了一个双射关系，以方便使用HQL去管理查询。

(2) 用于数据分析、清洗

Hive适用于离线的数据分析和清洗，延迟较高。

(3) 基于HDFS、MapReduce

Hive存储的数据依旧在DataNode上，编写的HQL语句终将是转换为MapReduce代码执行。

2．HBase

(1) 数据库

是一种面向列族存储的非关系型数据库。

(2) 用于存储结构化和非结构化的数据

适用于单表非关系型数据的存储，不适合做关联查询，类似JOIN等操作。

(3) 基于HDFS

数据持久化存储的体现形式是HFile，存放于DataNode中，被ResionServer以region的形式进行管理。

(4) 延迟较低，接入在线业务使用

面对大量的企业数据，HBase可以直线单表大量数据的存储，同时提供了高效的数据访问速度。

## HBase与Hive集成使用

尖叫提示：HBase与Hive的集成在最新的两个版本中无法兼容。所以，我们只能含着泪勇敢的重新编译：hive-hbase-handler-1.2.2.jar！！好气！！

环境准备

因为我们后续可能会在操作Hive的同时对HBase也会产生影响，所以Hive需要持有操作HBase的Jar，那么接下来拷贝Hive所依赖的Jar包（或者使用软连接的形式）。

|  |
| --- |
| export HBASE\_HOME=/opt/module/hbase  export HIVE\_HOME=/opt/module/hive  ln -s $HBASE\_HOME/lib/hbase-common-1.3.1.jar $HIVE\_HOME/lib/hbase-common-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-server-1.3.1.jar $HIVE\_HOME/lib/hbase-server-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-client-1.3.1.jar $HIVE\_HOME/lib/hbase-client-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-protocol-1.3.1.jar $HIVE\_HOME/lib/hbase-protocol-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-it-1.3.1.jar $HIVE\_HOME/lib/hbase-it-1.3.1.jar  ln -s $HBASE\_HOME/lib/htrace-core-3.1.0-incubating.jar $HIVE\_HOME/lib/htrace-core-3.1.0-incubating.jar  ln -s $HBASE\_HOME/lib/hbase-hadoop2-compat-1.3.1.jar $HIVE\_HOME/lib/hbase-hadoop2-compat-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-hadoop-compat-1.3.1.jar $HIVE\_HOME/lib/hbase-hadoop-compat-1.3.1.jar |

同时在hive-site.xml中修改zookeeper的属性，如下：

|  |
| --- |
| <property>  <name>hive.zookeeper.quorum</name>  <value>hadoop102,hadoop103,hadoop104</value>  <description>The list of ZooKeeper servers to talk to. This is only needed for read/write locks.</description>  </property>  <property>  <name>hive.zookeeper.client.port</name>  <value>2181</value>  <description>The port of ZooKeeper servers to talk to. This is only needed for read/write locks.</description>  </property> |

1．案例一

**目标：**建立Hive表，关联HBase表，插入数据到Hive表的同时能够影响HBase表。

分步实现：

(1) 在Hive中创建表同时关联HBase

|  |
| --- |
| CREATE TABLE hive\_hbase\_emp\_table(  empno int,  ename string,  job string,  mgr int,  hiredate string,  sal double,  comm double,  deptno int)  STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,info:comm,info:deptno")  TBLPROPERTIES ("hbase.table.name" = "hbase\_emp\_table"); |

提示：完成之后，可以分别进入Hive和HBase查看，都生成了对应的表

(2) 在Hive中创建临时中间表，用于load文件中的数据

提示：不能将数据直接load进Hive所关联HBase的那张表中

|  |
| --- |
| CREATE TABLE emp(  empno int,  ename string,  job string,  mgr int,  hiredate string,  sal double,  comm double,  deptno int)  row format delimited fields terminated by '\t'; |

(3) 向Hive中间表中load数据

|  |
| --- |
| hive> load data local inpath '/home/admin/softwares/data/emp.txt' into table emp; |

(4) 通过insert命令将中间表中的数据导入到Hive关联Hbase的那张表中

|  |
| --- |
| hive> insert into table hive\_hbase\_emp\_table select \* from emp; |

(5) 查看Hive以及关联的HBase表中是否已经成功的同步插入了数据

Hive：

|  |
| --- |
| hive> select \* from hive\_hbase\_emp\_table; |

HBase：

|  |
| --- |
| Hbase> scan ‘hbase\_emp\_table’ |

2．案例二

**目标：**在HBase中已经存储了某一张表hbase\_emp\_table，然后在Hive中创建一个外部表来关联HBase中的hbase\_emp\_table这张表，使之可以借助Hive来分析HBase这张表中的数据。

**注：**该案例2紧跟案例1的脚步，所以完成此案例前，请先完成案例1。

分步实现：

(1) 在Hive中创建外部表

|  |
| --- |
| CREATE EXTERNAL TABLE relevance\_hbase\_emp(  empno int,  ename string,  job string,  mgr int,  hiredate string,  sal double,  comm double,  deptno int)  STORED BY  'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  WITH SERDEPROPERTIES ("hbase.columns.mapping" =  ":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,info:comm,info:deptno")  TBLPROPERTIES ("hbase.table.name" = "hbase\_emp\_table"); |

(2) 关联后就可以使用Hive函数进行一些分析操作了

|  |
| --- |
| hive (default)> select \* from relevance\_hbase\_emp; |

# HBase优化

## 预分区

每一个region维护着StartRow与EndRow，如果加入的数据符合某个Region维护的RowKey范围，则该数据交给这个Region维护。那么依照这个原则，我们可以将数据所要投放的分区提前大致的规划好，以提高HBase性能。

* + - 1. 手动设定预分区

Hbase> create 'staff1','info','partition1',SPLITS => ['1000','2000','3000','4000']

* + - 1. 生成16进制序列预分区

create 'staff2','info','partition2',{NUMREGIONS => 15, SPLITALGO => 'HexStringSplit'}

* + - 1. 按照文件中设置的规则预分区

创建splits.txt文件内容如下：

|  |
| --- |
| aaaa  bbbb  cccc  dddd |

然后执行：

create 'staff3','partition3',SPLITS\_FILE => 'splits.txt'

* + - 1. 使用JavaAPI创建预分区

|  |
| --- |
| //自定义算法，产生一系列hash散列值存储在二维数组中  byte[][] splitKeys = 某个散列值函数  //创建HbaseAdmin实例  HBaseAdmin hAdmin = new HBaseAdmin(HbaseConfiguration.create());  //创建HTableDescriptor实例  HTableDescriptor tableDesc = new HTableDescriptor(tableName);  //通过HTableDescriptor实例和散列值二维数组创建带有预分区的Hbase表  hAdmin.createTable(tableDesc, splitKeys); |

## RowKey设计

一条数据的唯一标识就是RowKey，那么这条数据存储于哪个分区，取决于RowKey处于哪个一个预分区的区间内，设计RowKey的主要目的 ，就是让数据均匀的分布于所有的region中，在一定程度上防止数据倾斜。接下来我们就谈一谈RowKey常用的设计方案。

* + - 1. 生成随机数、hash、散列值

|  |
| --- |
| 比如：  原本rowKey为1001的，SHA1后变成：dd01903921ea24941c26a48f2cec24e0bb0e8cc7  原本rowKey为3001的，SHA1后变成：49042c54de64a1e9bf0b33e00245660ef92dc7bd  原本rowKey为5001的，SHA1后变成：7b61dec07e02c188790670af43e717f0f46e8913  在做此操作之前，一般我们会选择从数据集中抽取样本，来决定什么样的rowKey来Hash后作为每个分区的临界值。 |

* + - 1. 字符串反转

|  |
| --- |
| 20170524000001转成10000042507102  20170524000002转成20000042507102 |

这样也可以在一定程度上散列逐步put进来的数据。

* + - 1. 字符串拼接

|  |
| --- |
| 20170524000001\_a12e  20170524000001\_93i7 |

## 内存优化

HBase操作过程中需要大量的内存开销，毕竟Table是可以缓存在内存中的，一般会分配整个可用内存的70%给HBase的Java堆。但是不建议分配非常大的堆内存，因为GC过程持续太久会导致RegionServer处于长期不可用状态，一般16~48G内存就可以了，如果因为框架占用内存过高导致系统内存不足，框架一样会被系统服务拖死。

## 基础优化

* + - 1. 允许在HDFS的文件中追加内容

hdfs-site.xml、hbase-site.xml

|  |
| --- |
| 属性：dfs.support.append  解释：开启HDFS追加同步，可以优秀的配合HBase的数据同步和持久化。默认值为true。 |

* + - 1. 优化DataNode允许的最大文件打开数

hdfs-site.xml

|  |
| --- |
| 属性：dfs.datanode.max.transfer.threads  解释：HBase一般都会同一时间操作大量的文件，根据集群的数量和规模以及数据动作，设置为4096或者更高。默认值：4096 |

* + - 1. 优化延迟高的数据操作的等待时间

hdfs-site.xml

|  |
| --- |
| 属性：dfs.image.transfer.timeout  解释：如果对于某一次数据操作来讲，延迟非常高，socket需要等待更长的时间，建议把该值设置为更大的值（默认60000毫秒），以确保socket不会被timeout掉。 |

* + - 1. 优化数据的写入效率

mapred-site.xml

|  |
| --- |
| 属性：  mapreduce.map.output.compress  mapreduce.map.output.compress.codec  解释：开启这两个数据可以大大提高文件的写入效率，减少写入时间。第一个属性值修改为true，第二个属性值修改为：org.apache.hadoop.io.compress.GzipCodec或者其他压缩方式。 |

* + - 1. 设置RPC监听数量

hbase-site.xml

|  |
| --- |
| 属性：Hbase.regionserver.handler.count  解释：默认值为30，用于指定RPC监听的数量，可以根据客户端的请求数进行调整，读写请求较多时，增加此值。 |

* + - 1. 优化HStore文件大小

hbase-site.xml

|  |
| --- |
| 属性：hbase.hregion.max.filesize  解释：默认值10737418240（10GB），如果需要运行HBase的MR任务，可以减小此值，因为一个region对应一个map任务，如果单个region过大，会导致map任务执行时间过长。该值的意思就是，如果HFile的大小达到这个数值，则这个region会被切分为两个Hfile。 |

* + - 1. 优化HBase客户端缓存

hbase-site.xml

|  |
| --- |
| 属性：hbase.client.write.buffer  解释：用于指定Hbase客户端缓存，增大该值可以减少RPC调用次数，但是会消耗更多内存，反之则反之。一般我们需要设定一定的缓存大小，以达到减少RPC次数的目的。 |

* + - 1. 指定scan.next扫描HBase所获取的行数

hbase-site.xml

|  |
| --- |
| 属性：hbase.client.scanner.caching  解释：用于指定scan.next方法获取的默认行数，值越大，消耗内存越大。 |

* + - 1. flush、compact、split机制

当MemStore达到阈值，将Memstore中的数据Flush进Storefile；compact机制则是把flush出来的小文件合并成大的Storefile文件。split则是当Region达到阈值，会把过大的Region一分为二。

涉及属性：

即：128M就是Memstore的默认阈值

|  |
| --- |
| hbase.hregion.memstore.flush.size：134217728 |

即：这个参数的作用是当单个HRegion内所有的Memstore大小总和超过指定值时，flush该HRegion的所有memstore。RegionServer的flush是通过将请求添加一个队列，模拟生产消费模型来异步处理的。那这里就有一个问题，当队列来不及消费，产生大量积压请求时，可能会导致内存陡增，最坏的情况是触发OOM。

|  |
| --- |
| hbase.regionserver.global.memstore.upperLimit：0.4  hbase.regionserver.global.memstore.lowerLimit：0.38 |

即：当MemStore使用内存总量达到hbase.regionserver.global.memstore.upperLimit指定值时，将会有多个MemStores flush到文件中，MemStore flush 顺序是按照大小降序执行的，直到刷新到MemStore使用内存略小于lowerLimit