大数据技术之HBase基础

# HBase的起源

HBase的原型是Google的BigTable论文，受到了该论文思想的启发，目前作为Hadoop的子项目来开发维护，用于支持结构化的数据存储。

官方网站：<http://hbase.apache.org>

\* 2006年Google发表BigTable白皮书

\* 2006年开始开发HBase

\* 2008年北京成功开奥运会，程序员默默地将HBase弄成了Hadoop的子项目

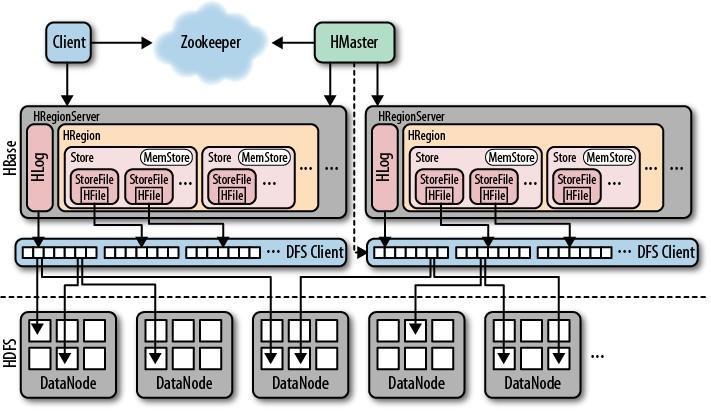
\* 2010年HBase成为Apache顶级项目

\* 现在很多公司二次开发出了很多发行版本，你也开始使用了

# 二、基于Hadoop的HBase架构C:\Users\61661\AppData\Local\Temp\1501410678(1).png

HBase内置有zookeeper，但一般我们会有其他的Zookeeper集群来监管master和regionserver，Zookeeper通过选举，保证任何时候，集群中只有一个活跃的HMaster，HMaster与HRegionServer 启动时会向ZooKeeper注册，存储所有HRegion的寻址入口，实时监控HRegionserver的上线和下线信息。并实时通知给HMaster，存储HBase的schema和table元数据，默认情况下，HBase 管理ZooKeeper 实例，Zookeeper的引入使得HMaster不再是单点故障。一般情况下会启动两个HMaster，非Active的HMaster会定期的和Active HMaster通信以获取其最新状态，从而保证它是实时更新的，因而如果启动了多个HMaster反而增加了Active HMaster的负担。

一个RegionServer可以包含多个HRegion，每个RegionServer维护一个HLog，和多个HFiles以及其对应的MemStore。RegionServer运行于DataNode上，数量可以与DatNode数量一致，请参考如下架构图：



# 三、RDBMS与HBase的对比

## 3.1、关系型数据库

**结构：**

\* 数据库以表的形式存在

\* 支持FAT、NTFS、EXT、文件系统

\* 使用Commit log存储日志

\* 参考系统是坐标系统

\* 使用主键（PK）

\* 支持分区

\* 使用行、列、单元格

**功能：**

\* 支持向上扩展

\* 使用SQL查询

\* 面向行，即每一行都是一个连续单元

\* 数据总量依赖于服务器配置

\* 具有ACID支持

\* 适合结构化数据

\* 传统关系型数据库一般都是中心化的

\* 支持事务

\* 支持Join

## 3.2、HBase

**结构：**

\* 数据库以region的形式存在

\* 支持HDFS文件系统

\* 使用WAL（Write-Ahead Logs）存储日志

\* 参考系统是Zookeeper

\* 使用行键（row key）

\* 支持分片

\* 使用行、列、列族和单元格

**功能：**

\* 支持向外扩展

\* 使用API和MapReduce来访问HBase表数据

\* 面向列，即每一列都是一个连续的单元

\* 数据总量不依赖具体某台机器，而取决于机器数量

\* HBase不支持ACID（Atomicity、Consistency、Isolation、Durability）

\* 适合结构化数据和非结构化数据

\* 一般都是分布式的

\* HBase不支持事务

\* 不支持Join

# 四、HBase特征简要

### 4.1、自动故障处理和负载均衡

HBase运行在HDFS上，所以HBase中的数据以多副本形式存放，数据也服从分布式存放，数据的恢复也可以得到保障。另外，HMaster和RegionServer也是多副本的。

### 4.2、自动分区

HBase表是由分布在多个RegionServer中的region组成的，这些RegionServer又分布在不同的DataNode上，如果一个region增长到了一个阈值，为了负载均衡和减少IO，HBase可以自动或手动干预的将region切分为更小的region，也称之为subregion。

### 4.3、集成Hadoop/HDFS

虽然HBase也可以运行在其他的分布式文件系统之上，但是与HDFS结合非常之方便，而且HDFS也非常之流行。

### 4.4、实时随机大数据访问

HBase采用log-structured merge-tree作为内部数据存储架构，这种架构会周期性地将小文件合并成大文件以减少磁盘访问同时减少NameNode压力。

### 4.5、MapReduce

HBase内建支持MapReduce框架，更加方便快速，并行的处理数据。

### 4.6、Java API

HBase提供原声的Java API支持，方便开发。

### 4.7、横向扩展

HBase支持横向扩展，这就意味着如果现有服务器硬件性能出现瓶颈，不需要停掉现有集群提升硬件配置，而只需要在现有的正在运行的集群中添加新的机器节点即可，而且新的RegionServer一旦建立完毕，集群会开始重新调整。

### 4.8、列存储

HBase是面向列存储的，每个列都单独存储，所以在HBase中列是连续存储的，而行不是。

### 4.9、HBase Shell

HBase提供了交互式命令行工具可以进行创建表、添加数据、扫描数据、删除数据等操作和其他一些管理命令。

# 五、HBase在集群中的定位

HBase一种是作为存储的分布式文件系统，另一种是作为数据处理模型的MR框架。因为日常开发人员比较熟练的是结构化的数据进行处理，但是在HDFS直接存储的文件往往不具有结构化，所以催生出了HBase在HDFS上的操作。如果需要查询数据，只需要通过键值便可以成功访问。

# 六、HBase内部存储架构

HBase是由row key，column family，column和cell组成，row key确定唯一的一行，column family由若干column组成，column是表的字段，cell存储了实际的值或数据。

# 七、HBase与Hadoop

## 7.1、HDFS

\* 为分布式存储提供文件系统

\* 针对存储大尺寸的文件进行优化，不需要对HDFS上的文件进行随机读写

\* 直接使用文件

\* 数据模型不灵活

\* 使用文件系统和处理框架

\* 优化一次写入，多次读取的方式

## 7.2、HBase

\* 提供表状的面向列的数据存储

\* 针对表状数据的随机读写进行优化

\* 使用key-value操作数据

\* 提供灵活的数据模型

\* 使用表状存储，支持MapReduce，依赖HDFS

\* 优化了多次读，以及多次写

# 八、HBase的优缺点

## 8.1、优点

\* 方便高效的压缩数据

\* 支持快速数据检索

\* 管理和配置简单，支持横向扩展，所以非常容易扩展

\* 聚合查询性能非常高

\* 可高效地进行分区，提供自动分区机制把大的region切分成小的subregion

## 8.2、缺点

\* 对JOIN以及多表合并数据的查询性能不好

\* 更新过程中有大量的写入和删除操作，需要频繁合并和分裂，降低存储效率

\* 对关系模型支持不好，分区和索引模式设计比较困难。

# 九、HBase的环境角色

## 9.1、HMaster

### 9.1.1、功能描述

\* 监控RegionServer

\* 处理RegionServer故障转移

\* 处理元数据的变更

\* 处理region的分配或移除

\* 在空闲时间进行数据的负载均衡

\* 通过Zookeeper发布自己的位置给客户端

## 9.2、RegionServer

### 9.2.1、功能描述

\* 负责存储HBase的实际数据

\* 处理分配给它的Region

\* 刷新缓存到HDFS

\* 维护HLog

\* 执行压缩

\* 负责处理Region分片

### 9.2.2、内含组件

**\* Write-Ahead logs**

HBase的修改记录，当对HBase读写数据的时候，数据不是直接写进磁盘，它会在内存中保留一段时间（时间以及数据量阈值可以设定）。如果机器突然原地爆炸，把数据保存在内存中会引起数据丢失，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，然后再写入内存中。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。

**\* HFile**

这是在磁盘上保存原始数据的实际的物理文件，是实际的存储文件。

**\* Store**

HFile存储在Store中，一个Store对应HBase表中的一个列族

**\* MemStore**

顾名思义，就是内存存储，位于内存中，用来保存当前的数据操作，所以当数据保存在WAL中之后，RegsionServer会在内存中存储键值对。

**\* Region**

Hbase表的分片，HBase表会根据RowKey值被切分成不同的region存储在RegionServer中，在一个RegionServer中可以有多个不同的region

## 9.3、Zookeeper

HMaster与HRegionServer 启动时会向ZooKeeper注册，存储所有HRegion的寻址入口，实时监控HRegionserver的上线和下线信息。并实时通知给HMaster，存储HBase的schema和table元数据，默认情况下，HBase 管理ZooKeeper 实例，Zookeeper的引入使得HMaster不再是单点故障。一般情况下会启动两个HMaster，非Active的HMaster会定期的和Active HMaster通信以获取其最新状态，从而保证它是实时更新的，因而如果启动了多个HMaster反而增加了Active HMaster的负担。

# 十、使用场景的探讨

## 10.1、何时使用

\* 如果数据有很多列，且包含很多空字段

\* 数据包含了不定数量的列

\* 需要维护数据的版本

\* 需要很高的横向扩展性

\* 需要大量的压缩数据

\* 需要大量的I/O

一般而言数百万行的数据和频率不高的读写操作，是不需要HBase的，如果有几十亿列数据，同时在单位时间内有数以千、万记的读写操作，可以考虑HBase。

## 10.2、何时不使用

\* 数据总量不大时（比如就几个G）

\* 当需要JOIN以及关系型数据库的一些特性时

\* 如果关系型数据库可以满足需求

# 十一、HBase的安装与部署

## 10.1、Zookeeper集群的正常部署并启动

$ /opt/modules/cdh/zookeeper-3.4.5-cdh5.3.6/bin/zkServer.sh start

## 10.2、Hadoop集群的正常部署并启动

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/start-dfs.sh

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/start-yarn.sh

## 10.3、解压HBase

$ tar -zxf /opt/softwares/hbase-0.98.6-cdh5.3.6.tar.gz -C /opt/modules/cdh/

## 10.4、修改HBase配置文件

### 10,.4.1、hbase-env.sh

C:\Users\61661\AppData\Local\Temp\1501432507(1).png

C:\Users\61661\AppData\Local\Temp\1501432485(1).png

### 10.4.2、hbase-site.xml



### 10.4.3、regionservers



## 10.5、替换HBase根目录下的lib目录下的jar包，以解决兼容问题

**\* 删除原有Jar包**

$ rm -rf /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hadoop-\*$ rm -rf lib/zookeeper-3.4.6.jar

（尖叫提示：如果lib目录下的zookeeper包不匹配也需要替换）

**\* 拷贝新的Jar包**

这里涉及到的jar包大概是：

hadoop-annotations-2.5.0.jar

hadoop-auth-2.5.0-cdh5.3.6.jar

hadoop-client-2.5.0-cdh5.3.6.jar

hadoop-common-2.5.0-cdh5.3.6.jar

hadoop-hdfs-2.5.0-cdh5.3.6.jar

hadoop-mapreduce-client-app-2.5.0-cdh5.3.6.jar

hadoop-mapreduce-client-common-2.5.0-cdh5.3.6.jar

hadoop-mapreduce-client-core-2.5.0-cdh5.3.6.jar

hadoop-mapreduce-client-hs-2.5.0-cdh5.3.6.jar

hadoop-mapreduce-client-hs-plugins-2.5.0-cdh5.3.6.jar

hadoop-mapreduce-client-jobclient-2.5.0-cdh5.3.6.jar

hadoop-mapreduce-client-jobclient-2.5.0-cdh5.3.6-tests.jar

hadoop-mapreduce-client-shuffle-2.5.0-cdh5.3.6.jar

hadoop-yarn-api-2.5.0-cdh5.3.6.jar

hadoop-yarn-applications-distributedshell-2.5.0-cdh5.3.6.jar

hadoop-yarn-applications-unmanaged-am-launcher-2.5.0-cdh5.3.6.jar

hadoop-yarn-client-2.5.0-cdh5.3.6.jar

hadoop-yarn-common-2.5.0-cdh5.3.6.jar

hadoop-yarn-server-applicationhistoryservice-2.5.0-cdh5.3.6.jar

hadoop-yarn-server-common-2.5.0-cdh5.3.6.jar

hadoop-yarn-server-nodemanager-2.5.0-cdh5.3.6.jar

hadoop-yarn-server-resourcemanager-2.5.0-cdh5.3.6.jar

hadoop-yarn-server-tests-2.5.0-cdh5.3.6.jar

hadoop-yarn-server-web-proxy-2.5.0-cdh5.3.6.jar

zookeeper-3.4.5-cdh5.3.6.jar

**我们可以通过find命令快速进行定位，例如我们可以执行：**

$ find /opt/modules/ -name hadoop-hdfs-2.5.0-cdh5.3.6.jar



然后将查找出来的Jar包根据指定位置复制到HBase的lib目录下，在这里我给大家整合好到一个文件夹中了，请依次执行：

$ tar -zxf /opt/softwares/CDH\_HadoopJar.tar.gz -C /opt/softwares/

$ cp -a /opt/softwares/HadoopJar/\* /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/

## 10.6、将整理好的HBase安装目录scp到其他机器节点

$ scp -r /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/ \

hadoop-senior02.itguigu.com:/opt/modules/cdh/

$ scp -r /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/ \

hadoop-senior03.itguigu.com:/opt/modules/cdh/

## 10.7、将Hadoop配置文件软连接到HBase的conf目录下

**\* core-site.xml**

$ ln -s /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/etc/hadoop/core-site.xml /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/conf/core-site.xml

**\* hdfs-site.xml**

$ ln -s /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/etc/hadoop/hdfs-site.xml /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/conf/hdfs-site.xml

（尖叫提示：不要忘记其他几台机器也要做此操作）

## 10.8、启动服务

$ bin/hbase-daemon.sh start master

$ bin/hbase-daemon.sh start regionserver

或者：

$ bin/start-hbase.sh

对应的停止命令：

$ bin/stop-hbase.sh

## 10.9、查看页面

启动成功后，可以通过主机名:60010地址来访问HBase的管理页面

例如，<http://hadoop-senior01.itguigu.com:60010>

# 十二、HBase常用操作

## 12.1、进入HBase客户端命令操作界面

$ bin/hbase shell

## 12.2、查看帮助命令

hbase(main):001:0> help

## 12.3、查看当前数据库中有哪些表

hbase(main):002:0> list

## 12.4、创建一张表

hbase(main):003:0> create 'student','info'

## 12.5、向表中存储一些数据

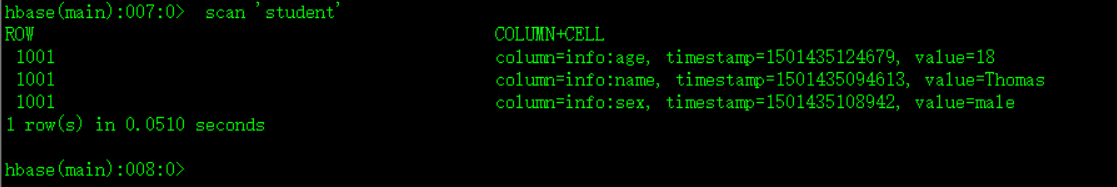
hbase(main):004:0> put 'student','1001','info:name','Thomas'

hbase(main):005:0> put 'student','1001','info:sex','male'

hbase(main):006:0> put 'student','1001','info:age','18'

## 12.6、扫描查看存储的数据

hbase(main):007:0> scan 'student'

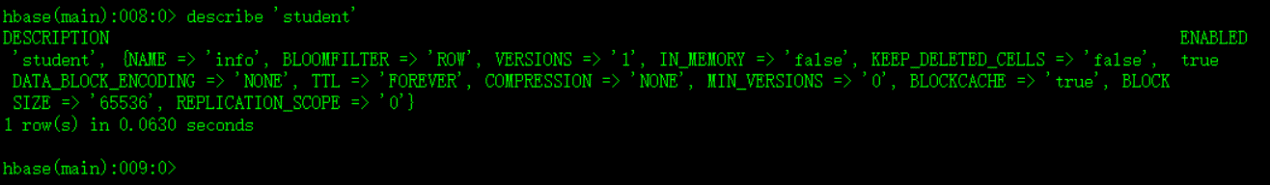


或：查看某个rowkey范围内的数据

hbase(main):014:0> scan 'student',{STARTROW => '1001',STOPROW => '1007'}

## 12.7、查看表结构

hbase(main):009:0> describe ‘student’

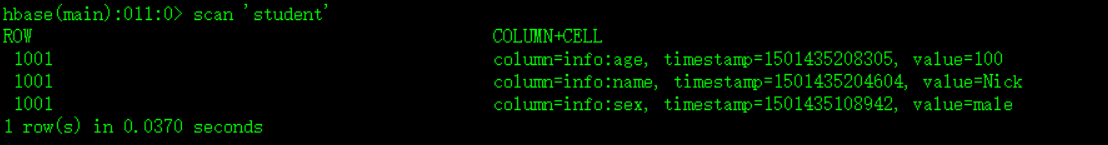


## 12.8、更新指定字段的数据

hbase(main):009:0> put 'student','1001','info:name','Nick'

hbase(main):010:0> put 'student','1001','info:age','100'

查看更新后的数据：



## 12.9、查看指定行的数据

hbase(main):012:0> get 'student','1001'

或：查看指定行指定列或列族的数据

hbase(main):013:0> get 'student','1001','info:name'

## 12.10、删除数据

### 12.10.1、删除某一个rowKey全部的数据

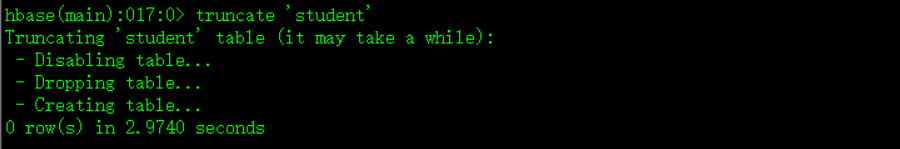
hbase(main):015:0> deleteall 'student','1001'

### 12.10.2、删除掉某个rowKey中某一列的数据

hbase(main):016:0> delete 'student','1001','info:sex'

## 12.11、清空表数据

hbase(main):017:0> truncate 'student'



## 12.12、删除表

首先需要先让该表为disable状态，使用命令：

hbase(main):018:0> disable 'student'然后才能drop这个表，使用命令：hbase(main):019:0> drop 'student'(尖叫提示：如果直接drop表，会报错：Drop the named table. Table must first be disabled)

## 12.13、统计一张表有多少行数据

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

# 十三、HMaster的高可用

## 13.1、确保HBase集群已正常停止

$ bin/stop-hbase.sh

## 13.2、在conf目录下创建backup-masters文件

$ touch conf/backup-masters

## 13.3、在backup-masters文件中配置高可用HMaster节点

$ echo hadoop-senior02.itguigu.com > conf/backup-masters

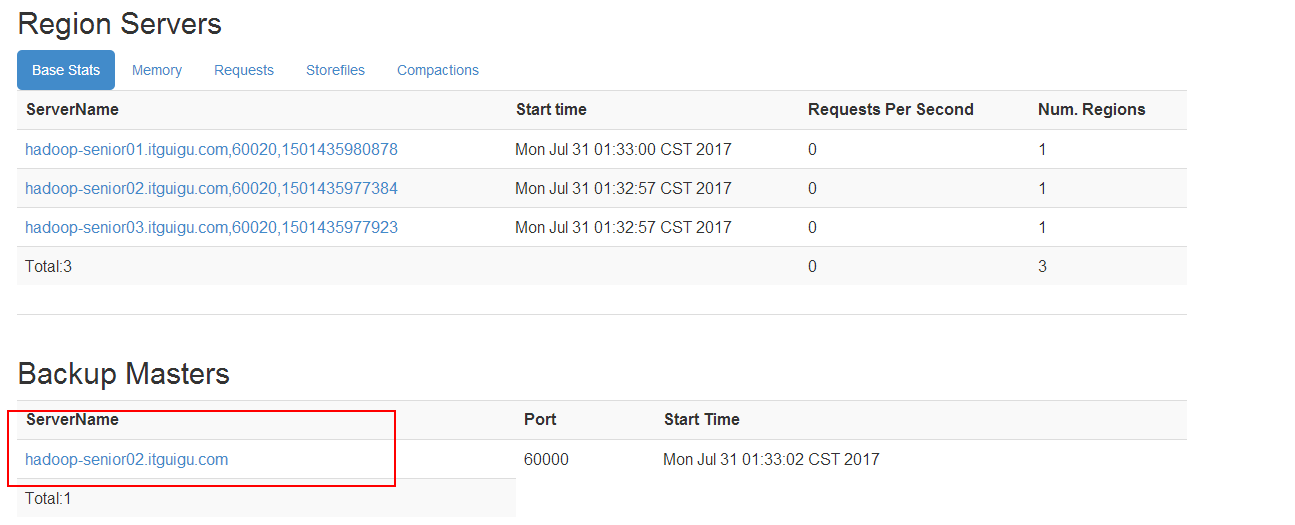
## 13.4、将整个conf目录scp到其他节点

$ scp -r conf/ hadoop-senior02.itguigu.com:/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/

$ scp -r conf/ hadoop-senior03.itguigu.com:/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/

## 13.5、打开页面测试

<http://hadoop-senior01.itguigu.com:60010>



最后，可以尝试关闭第一台机器的HMaster：

$ bin/hbase-daemon.sh stop master

然后查看第二台的HMaster是否会直接启用

# 十四、HBase和Hadoop的集群类型

## 14.1、单机模式

主要用于开发工作，一台机器上运行所有的守护进程，或者一台机器运行多个虚拟机。一般用于评估和测试。

## 14.2、小型集群

20台机器以内的集群，不同的机器运行不同的守护进程，适用于数据量和处理请求较少的小型生产环境。

## 14.3、中型集群

20到1000台机器集群，3到5个zookeeper节点，适用于成熟的生产环境。

## 14.4、大型集群

1000台机器以上的集群，属于超大规模集群了，适用于大规模生产环境。

# 十五、集群配置举例

## 15.1、NameNode/HMaster 常见配置

内存：16~128G

CPU：2\*（8~24）核处理器

硬盘：1TB-SATA硬盘+1个元数据备份盘（转速7200R/MIN+）能使用固态更好。

网卡：2\*1GB网卡

为了有更好的性能，所有的元数据都缓存在内存中，因此内存需要拥有较快的速度和较好的质量。大内存意为着可以存储更多的文件，从而支持NameNode更大的命名空间。同时NameNode不需要很大的磁盘，小容量的磁盘就可以满足需求，元数据要存储加载到内存中，数据副本以及修改日志存储在磁盘上。

## 15.2、ResourceManager

可以运行在NameNode机器上，也可以运行在单独的机器上。硬件配置和NameNode一直，因为只是用于作业分发，因此不需要较大的磁盘和较强的运算能力。

## 15.3、DataNode、RegionServer

实际的数据存储于这些节点，因此这些节点需要较大的存储和较强的运算能力。较小的集群可以使用一般的磁盘，内存和CPU，如果集群规模较大，可以考虑：

内存：16~128G

CPU：2\*（8~24）核处理器

硬盘：2TB，转速7200

网卡：2\*1GB

# 十六、CDH配置

备选资源：

内存：64~512GB

硬盘：1TB~4TB

CPU：2\*（8~24）核CPU，主频2~2.5GHZ

网卡：千、万兆以太网

## 16.1、CPU

工作负载核心，推荐DataNode配置为双CPU插槽，配置中等主频的CPU，高端CPU太烧钱，所以我们可以增加数量。

## 16.2、电源

耐热性，稳定。

## 16.3、内存

需要足量的内存以保证不需要等待数据频繁的装载到内存中，因此8~48G内存比较合适，HBase会使用大量的内存，将文件存放在内存中（如果开启了内存表的话），对于HBase集群，我们需要比单独的Hadoop集群更大的内存。如果HBase开启缓存，Hbase会尝试将整张表缓存在内存中。

## 16.4、磁盘

不建议在某台机器上配置很大容量的磁盘，这样当这台机器出现问题，不容易将数据分散到其他机器节点中。必须不能低于SATA 7200转

## 16.5、网络

Hadoop或者HBase在执行任务，读取数据和写入数据时，会在节点之间传输数据块，因此建议配置高速的网络和交换机。对于中小集群，1GB/s的网络足矣。对于排序和shuffle这类操作，需要节点间传输大量数据，如果带宽不足，会导致一些节点连接超时，比如RegionServer、Zookeeper。

# 十七、容量规划

运算公式：T = （S\* R）\* 1.25

尖叫提示：

S表示存储数据量

R表示副本数

T表示整个集群需要的空间

# 十八、HBase读写流程

## 18.1、HBase读数据流程

HRegionServer保存着meta表以及表数据，要访问表数据，首先Client先去访问zookeeper，从zookeeper里面获取meta表所在的位置信息，即找到这个meta表在哪个HRegionServer上保存着。

接着Client通过刚才获取到的HRegionServer的IP来访问Meta表所在的HRegionServer，从而读取到Meta，进而获取到Meta表中存放的元数据。

Client通过元数据中存储的信息，访问对应的HRegionServer，然后扫描所在HRegionServer的Memstore和Storefile来查询数据。

最后HRegionServer把查询到的数据响应给Client。

## 18.2、HBase写数据流程

Client也是先访问zookeeper，找到Meta表，并获取Meta表元数据。

确定当前将要写入的数据所对应的HRegion和HRegionServer服务器。

Client向该HRegionServer服务器发起写入数据请求，然后HRegionServer收到请求并响应。

Client先把数据写入到HLog，以防止数据丢失。

然后将数据写入到Memstore。

如果HLog和Memstore均写入成功，则这条数据写入成功

如果Memstore达到阈值，会把Memstore中的数据flush到Storefile中。

当Storefile越来越多，会触发Compact合并操作，把过多的Storefile合并成一个大的Storefile。

当Storefile越来越大，Region也会越来越大，达到阈值后，会触发Split操作，将Region一分为二。

# 十九、HBase中的3个重要机制

## 19.1、flush机制

当MemStore达到阈值，将Memstore中的数据Flush进Storefile

涉及属性：

hbase.hregion.memstore.flush.size：134217728

即：128M就是Memstore的默认阈值

hbase.regionserver.global.memstore.upperLimit：0.4

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，模拟生产消费模式来异步处理的。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.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

## 19.2、compact机制

把小的Memstore文件合并成大的Storefile文件。

## 19.3、split机制

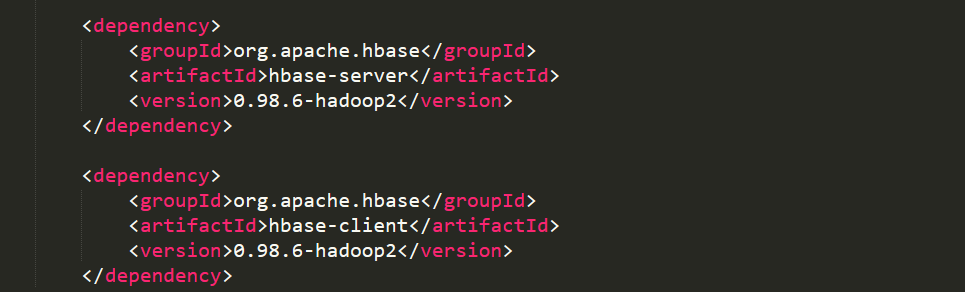
当Region达到阈值，会把过大的Region一分为二。

# 二十、HBaseAPI的使用

## 20.1、解压Maven离线仓库到指定目录

$ tar -zxf /opt/softwares/hbase+hadoop\_repository.tar.gz -C ~/.m2/

## 20.2、新建Eclipse的Maven Project，添加pom.xml的dependency如下：



## 20.3、编写HBaseAPI代码

详见项目代码

# 二十一、文件格式的说明

## 21.1、tsv格式的文件：字段之间以制表符\t分割

## 21.2、csv格式的文件：字段之间以逗号,分割

# 二十二、HBase的MapReduce的调用

## 22.1、查看HBase执行MapReduce所依赖的Jar包

执行命令：

$ bin/hbase mapredcp

出现如下内容：



## 22.2、执行环境变量导入

$ export HBASE\_HOME=/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/$ export HADOOP\_HOME=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6$ export HADOOP\_CLASSPATH=`${HBASE\_HOME}/bin/hbase mapredcp`

## 22.3、运行官方的MapReduce任务

### 22.3.1、案例一：统计student表中有多少行数据

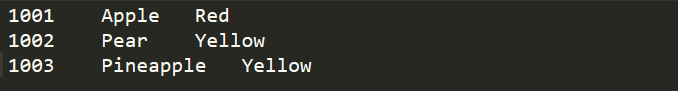
\* **执行代码**

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar lib/hbase-server-0.98.6-cdh5.3.6.jar rowcounter student

### 22.3.2、案例二：使用MapReduce任务将数据从文件中导入到HBase

**Step1、创建一个tsv格式的文件**

$ vi fruit.tsv，内容如下：



**Step2、创建HBase表**

$ bin/hbase shellhbase(main):001:0> create 'fruit','info'

**Step3、在HDFS中创建input\_fruit文件夹并上传fruit.tsv文件**

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/hdfs dfs -mkdir /input\_fruit/

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/hdfs dfs -put fruit.tsv /input\_fruit/

**Step4、执行MapReduce到HBase的fruit表中**

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar lib/hbase-server-0.98.6-cdh5.3.6.jar

importtsv -Dimporttsv.columns=HBASE\_ROW\_KEY,info:name,info:color fruit \

hdfs://hadoop-senior01.itguigu.com:8020/input\_fruit

**Step5、使用scan命令查看导入后的数据即可**

# 二十三、BulkLoad加载文件到HBase表

## 23.1、功能

将本地数据导入到HBase中

## 23.2、原理

BulkLoad会将tsv/csv格式的文件编程hfile文件，然后再进行数据的导入，这样可以避免大量数据导入时造成的集群写入压力过大。

## 23.3、作用

\* 减小HBase集群插入数据的压力

\* 提高了Job运行的速度，降低了Job执行时间

## 23.4、案例

### Step1、配置临时环境变量

$ export HBASE\_HOME=/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/$ export HADOOP\_HOME=/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6$ export HADOOP\_CLASSPATH=`${HBASE\_HOME}/bin/hbase mapredcp`

### Step2、创建一个新的HBase表

$ bin/hbase shell

hbase(main):001:0> create 'fruit\_bulkload','info'

### Step3、将tsv/csv文件转化为HFile （别忘了要确保你的fruit格式的文件fruit.tsv在input目录下）

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar \

/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-server-0.98.6-cdh5.3.6.jar importtsv \

-Dimporttsv.bulk.output=/output\_file \

-Dimporttsv.columns=HBASE\_ROW\_KEY,info:name,info:color \

fruit hdfs://hadoop-senior01.itguigu.com:8020/input\_fruit

### Step4、把HFile导入到HBase表fruit\_bulkload

上一步完成之后，你会发现在HDFS的根目录下出现了一个output\_file文件夹，里面存放的就是HFile文件，紧接着：把HFile导入到HBase表fruit\_bulkload

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/bin/yarn jar \

/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/lib/hbase-server-0.98.6-cdh5.3.6.jar \

completebulkload /output\_file fruit\_bulkload

### Step5、查看使用bulkLoad方式导入的数据

hbase(main):001:0> scan ‘fruit\_bulkload’

# 二十四、HBase自定义MapReduce

## 案例1、HBase表数据的转移

在Hadoop阶段，我们编写的MR任务分别进程了Mapper和Reducer两个类，而在HBase中我们需要继承的是TableMapper和TableReducer两个类。

### 目标：将fruit表中的一部分数据，通过MR迁入到fruit\_mr表中

### Step1、构建ReadFruitMapper类，用于读取fruit表中的数据

package com.z.hbase\_mr;

import java.io.IOException;

import org.apache.hadoop.hbase.Cell;

import org.apache.hadoop.hbase.CellUtil;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.client.Result;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableMapper;

import org.apache.hadoop.hbase.util.Bytes;

public class ReadFruitMapper extends TableMapper<ImmutableBytesWritable, Put> {

@Override

protected void map(ImmutableBytesWritable key, Result value, Context context)

throws IOException, InterruptedException {

//将fruit的name和color提取出来，相当于将每一行数据读取出来放入到Put对象中。

Put put = new Put(key.get());

//遍历添加column行

for(Cell cell: value.rawCells()){

//添加/克隆列族:info

if("info".equals(Bytes.toString(CellUtil.cloneFamily(cell)))){

//添加/克隆列：name

if("name".equals(Bytes.toString(CellUtil.cloneQualifier(cell)))){

//将该列cell加入到put对象中

put.add(cell);

//添加/克隆列:color

}else if("color".equals(Bytes.toString(CellUtil.cloneQualifier(cell)))){

//向该列cell加入到put对象中

put.add(cell);

}

}

}

//将从fruit读取到的每行数据写入到context中作为map的输出

context.write(key, put);

}

}

### Step2、构建WriteFruitMRReducer类，用于将读取到的fruit表中的数据写入到fruit\_mr表中

package com.z.hbase\_mr;

import java.io.IOException;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableReducer;

import org.apache.hadoop.io.NullWritable;

public class WriteFruitMRReducer extends TableReducer<ImmutableBytesWritable, Put, NullWritable> {

@Override

protected void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context context)

throws IOException, InterruptedException {

//读出来的每一行数据写入到fruit\_mr表中

for(Put put: values){

context.write(NullWritable.get(), put);

}

}

}

### Step3、构建Fruit2FruitMRJob extends Configured implements Tool，用于组装运行Job任务

//组装Job

public int run(String[] args) throws Exception {

//得到Configuration

Configuration conf = this.getConf();

//创建Job任务

Job job = Job.getInstance(conf, this.getClass().getSimpleName());

job.setJarByClass(Fruit2FruitMRJob.class);

//配置Job

Scan scan = new Scan();

scan.setCacheBlocks(false);

scan.setCaching(500);

//设置Mapper，注意导入的是mapreduce包下的，不是mapred包下的，后者是老版本

TableMapReduceUtil.initTableMapperJob(

"fruit", //数据源的表名

scan, //scan扫描控制器

ReadFruitMapper.class,//设置Mapper类

ImmutableBytesWritable.class,//设置Mapper输出key类型

Put.class,//设置Mapper输出value值类型

job//设置给哪个JOB

);

//设置Reducer

TableMapReduceUtil.initTableReducerJob("fruit\_mr", WriteFruitMRReducer.class, job);

//设置Reduce数量，最少1个

job.setNumReduceTasks(1);

boolean isSuccess = job.waitForCompletion(true);

if(!isSuccess){

throw new IOException("Job running with error");

}

return isSuccess ? 0 : 1;

}

### Step4、主函数中调用运行该Job任务

public static void main( String[] args ) throws Exception{

Configuration conf = HBaseConfiguration.create();

int status = ToolRunner.run(conf, new Fruit2FruitMRJob(), args);

System.exit(status);

}

## 案例2、将文件中的数据导入到HBase数据表

其实该案例的思想和案例1没有太大不同，思路总体还是一样的，只不过这次Mapper不是从HBase的表里读取数据了，而是从HDFS上的文件中读取数据，所以Mapper可直接继承自HDFS的Mapper。

### Step1、构建Mapper用于读取HDFS中的文件数据

package com.z.hbase.mr2;

import java.io.IOException;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.util.Bytes;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class ReadFruitFromHDFSMapper extends Mapper<LongWritable, Text, ImmutableBytesWritable, Put> {

@Override

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

//从HDFS中读取的数据

String lineValue = value.toString();

//读取出来的每行数据使用\t进行分割，存于String数组

String[] values = lineValue.split("\t");

//根据数据中值的含义取值

String rowKey = values[0];

String name = values[1];

String color = values[2];

//初始化rowKey

ImmutableBytesWritable rowKeyWritable = new ImmutableBytesWritable(Bytes.toBytes(rowKey));

//初始化put对象

Put put = new Put(Bytes.toBytes(rowKey));

//参数分别:列族、列、值

put.add(Bytes.toBytes("info"), Bytes.toBytes("name"), Bytes.toBytes(name));

put.add(Bytes.toBytes("info"), Bytes.toBytes("color"), Bytes.toBytes(color));

context.write(rowKeyWritable, put);

}

}

### Step2、构建WriteFruitMRFromTxtReducer类

package com.z.hbase.mr2;

import java.io.IOException;

import org.apache.hadoop.hbase.client.Put;

import org.apache.hadoop.hbase.io.ImmutableBytesWritable;

import org.apache.hadoop.hbase.mapreduce.TableReducer;

import org.apache.hadoop.io.NullWritable;

public class WriteFruitMRFromTxtReducer extends TableReducer<ImmutableBytesWritable, Put, NullWritable> {

@Override

protected void reduce(ImmutableBytesWritable key, Iterable<Put> values, Context context) throws IOException, InterruptedException {

//读出来的每一行数据写入到fruit\_hdfs表中

for(Put put: values){

context.write(NullWritable.get(), put);

}

}

}

### Step3、组装Job

public int run(String[] args) throws Exception {

//得到Configuration

Configuration conf = this.getConf();

//创建Job任务

Job job = Job.getInstance(conf, this.getClass().getSimpleName());

job.setJarByClass(HDFS2HBaseDriver.class);

// Path inPath = new Path("/input/fruit.txt");

Path inPath = new Path("hdfs://hadoop-senior01.itguigu.com:8020/input\_fruit/fruit.tsv");

FileInputFormat.addInputPath(job, inPath);

//设置Mapper

job.setMapperClass(ReadFruitFromHDFSMapper.class);

job.setMapOutputKeyClass(ImmutableBytesWritable.class);

job.setMapOutputValueClass(Put.class);

//设置Reducer

TableMapReduceUtil.initTableReducerJob("fruit\_hdfs", WriteFruitMRFromTxtReducer.class, job);

//设置Reduce数量，最少1个

job.setNumReduceTasks(1);

boolean isSuccess = job.waitForCompletion(true);

if(!isSuccess){

throw new IOException("Job running with error");

}

return isSuccess ? 0 : 1;

}

### Step4、提交运行Job

public static void main(String[] args) throws Exception {

Configuration conf = HBaseConfiguration.create();

int status = ToolRunner.run(conf, new HDFS2HBaseDriver(), args);

System.exit(status);

}

# 二十五、HBase与Hive的对比

## 25.1、Hive

### 25.1.1、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。

### 25.1.2、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

### 25.1.3、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。（不要钻不需要执行MapReduce代码的情况的牛角尖）

## 25.2、HBase

### 25.2.1、数据库

是一种面向列存储的非关系型数据库。

### 25.2.2、用于存储结构化和非结构话的数据

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。

### 25.2.3、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。

### 25.2.4、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。

### 总结：Hive与HBase

Hive和Hbase是两种基于Hadoop的不同技术，Hive是一种类SQL的引擎，并且运行MapReduce任务，Hbase是一种在Hadoop之上的NoSQL 的Key/vale数据库。这两种工具是可以同时使用的。就像用Google来搜索，用FaceBook进行社交一样，Hive可以用来进行统计查询，HBase可以用来进行实时查询，数据也可以从Hive写到HBase，或者从HBase写回Hive。

# 二十六、HBase与Hive交互操作

## 26.1、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/modules/cdh/hbase-0.98.6-cdh5.3.6/

$ export HIVE\_HOME=/opt/modules/cdh/hive-0.13.1-cdh5.3.6/

$ ln -s $HBASE\_HOME/lib/hbase-common-0.98.6-cdh5.3.6.jar $HIVE\_HOME/lib/hbase-common-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE\_HOME/lib/hbase-server-0.98.6-cdh5.3.6.jar $HIVE\_HOME/lib/hbase-server-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE\_HOME/lib/hbase-client-0.98.6-cdh5.3.6.jar $HIVE\_HOME/lib/hbase-client-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE\_HOME/lib/hbase-protocol-0.98.6-cdh5.3.6.jar $HIVE\_HOME/lib/hbase-protocol-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE\_HOME/lib/hbase-it-0.98.6-cdh5.3.6.jar $HIVE\_HOME/lib/hbase-it-0.98.6-cdh5.3.6.jar

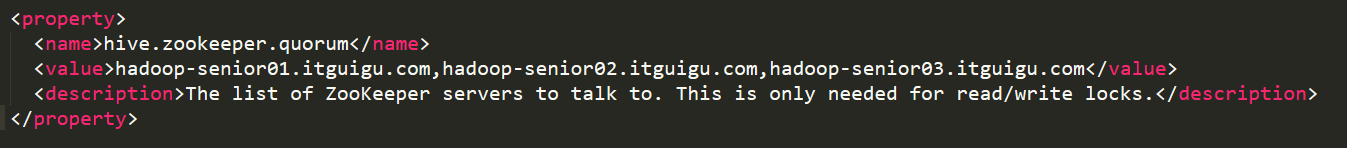
$ ln -s $HBASE\_HOME/lib/htrace-core-2.04.jar $HIVE\_HOME/lib/htrace-core-2.04.jar

$ ln -s $HBASE\_HOME/lib/hbase-hadoop2-compat-0.98.6-cdh5.3.6.jar $HIVE\_HOME/lib/hbase-hadoop2-compat-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE\_HOME/lib/hbase-hadoop-compat-0.98.6-cdh5.3.6.jar $HIVE\_HOME/lib/hbase-hadoop-compat-0.98.6-cdh5.3.6.jar

$ ln -s $HBASE\_HOME/lib/high-scale-lib-1.1.1.jar $HIVE\_HOME/lib/high-scale-lib-1.1.1.jar

同时在**hive-site.xml**中修改zookeeper的属性，如下：



## 26.2、案例1：创建Hive表，关联HBase表，插入数据到Hive表的同时能够影响HBase

### Step1、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");

（尖叫提示：完成之后，可以分别进入Hive和HBase查看，都生成了对应的表）

### Step2、在Hive中创建临时中间表，用于load文件中的数据（注：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';

### Step3、向Hive中间表中load数据

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

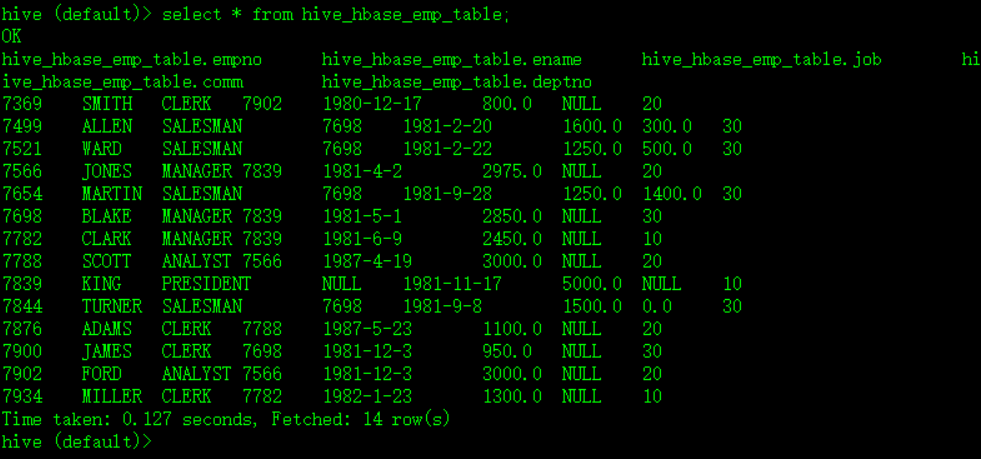
### Step4、通过insert命令将中间表中的数据导入到Hive关联HBase的那张表中

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

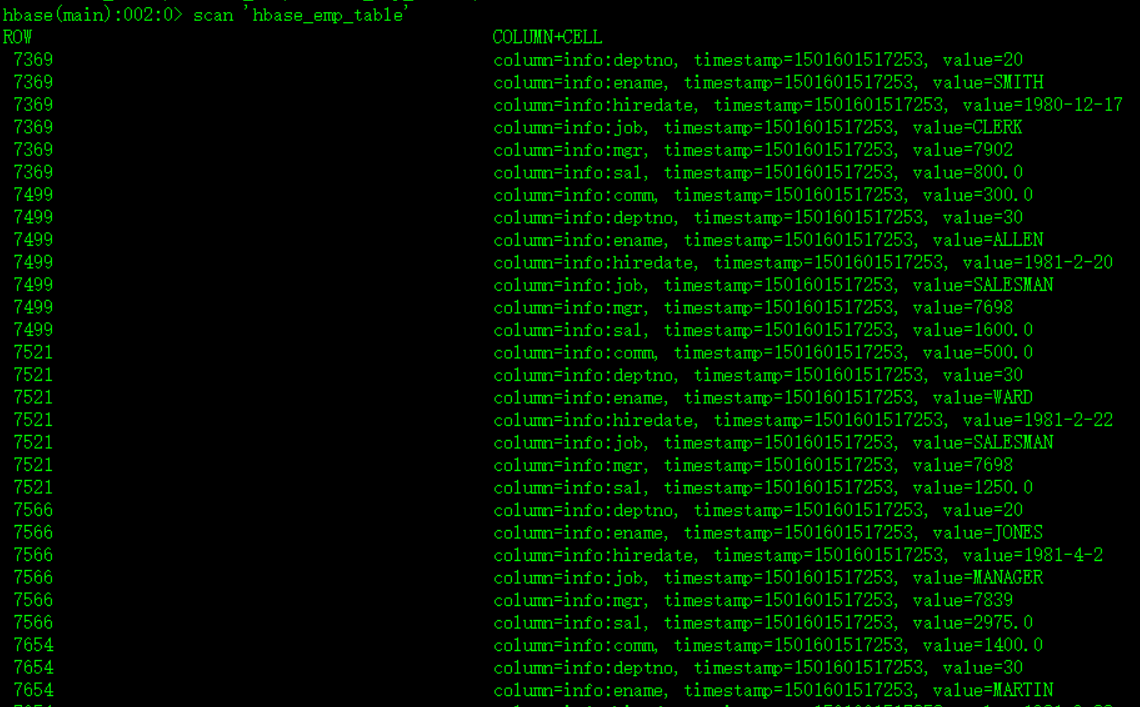
### Step5、测试，查看Hive以及关联的HBase表中是否已经成功的同步插入了数据

如图所示：

Hive中：



HBase中：



## 26.3、案例2：比如在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。

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。

### Step1、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");

### Step2、关联后就可以使用Hive函数进行一些分析操作了

在此，我们查询一下所有数据试试看

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

# 二十七、HBase与Sqoop集成

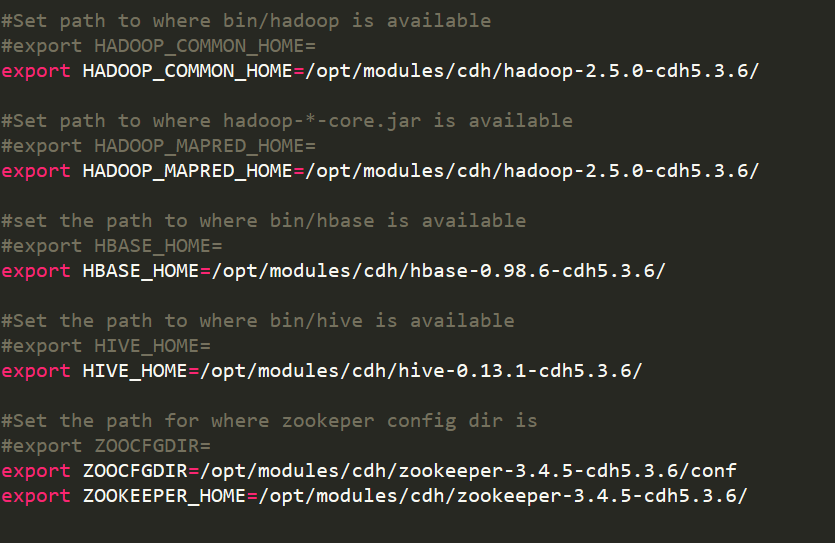
回顾之前Sqoop案例：

1. RDBMS到HDFS的数据导入
2. RDBMS到Hive的数据导入
3. Hive/HDFS到RDBMS的数据导出

今天我们来讨论一下如何使用Sqoop将RDBMS中的数据导入到HBase当中。

## 27.1、案例：将RDBMS中的数据抽取到HBase中

### Step1、配置sqoop-env.sh如下：



### Step2、在Mysql中创建一张数据库library，一张表book

CREATE DATABASE library;

CREATE TABLE book(

id int(4) PRIMARY KEY NOT NULL AUTO\_INCREMENT,

name VARCHAR(255) NOT NULL,

price VARCHAR(255) NOT NULL);

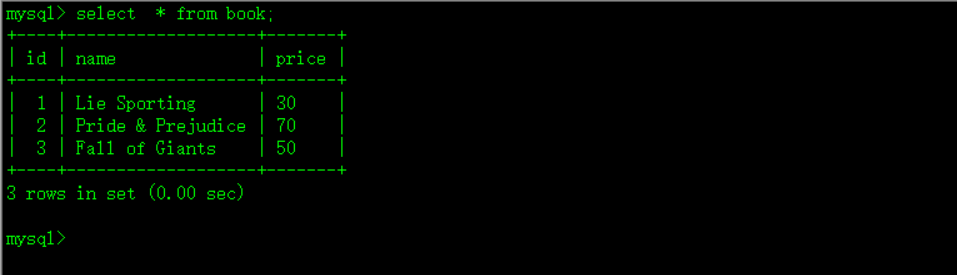
### Step3、向表中插入一些数据

INSERT INTO book(name, price) VALUES('Lie Sporting', '30');

INSERT INTO book (name, price) VALUES('Pride & Prejudice', '70');

INSERT INTO book (name, price) VALUES('Fall of Giants', '50');

完成后如图：



### Step4、执行Sqoop导入数据的操作

$ bin/sqoop import \

--connect jdbc:mysql://hadoop-senior01.itguigu.com:3306/db\_library \

--username root \

--password 123456 \

--table book \

--columns "id,name,price" \

--column-family "info" \

--hbase-create-table \

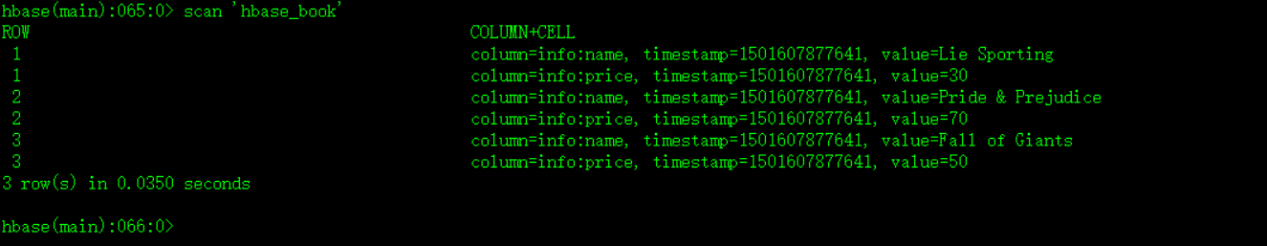
--hbase-row-key "id" \

--hbase-table "hbase\_book" \

--num-mappers 1 \

--split-by id

### Step5、在HBase中scan这张表得到如下内容



（尖叫提示：在导入之前，HBase中的表如果不存在则会自动创建）

# 二十八、HBase中计算存储数据的大小

## 28.1、固定大小 fixed size

即，当前表设计时，预算的字段名所占用的空间，比如：

RowKey：6个字节

Value：4个字节

TimeStamp：9个字节

KeyType：1个字节

CF：1个字节

那么固定大小fixed size = 6 + 4 + 9 + 1 + 1 = 21字节

当然这只是一个近似估计。

## 28.2、可变大小 variable size

即，存入的数据的具体指可能会产生波动，比如全世界人类表，俄罗斯的人名比较长，从早上读到晚上都没有念完，中国人名短，这就造成了存入的数据可能是一个可变的大小。

那么总的数据量估算就是：total = fixed size + variable size

# 二十九、HBase Shell

## 29.1、status

例如：显示服务器状态

hbase> status ‘hadoop-senior01.itguigu.com’

## 29.2、whoami

显示HBase当前用户，例如：

hbase> whoami

## 29.3、list

显示当前所有的表

## 29.4、count

统计指定表的记录数，例如：

hbase> count 'hbase\_book'

## 29.5、describe

展示表结构信息

## 29.6、exist

检查表是否存在，适用于表量特别多的情况

## 29.7、is\_enabled、is\_disabled

检查表是否启用或禁用

### 29.8、alter

该命令可以改变表和列族的模式，例如：

**为当前表增加列族：**

hbase> alter 'hbase\_book', NAME => 'CF2', VERSIONS => 2

**为当前表删除列族：**

hbase> alter 'hbase\_book', 'delete' => ’CF2’

### 29.9、disable

禁用一张表

### 29.10、drop

删除一张表，记得在删除表之前必须先禁用

### 29.11、delete

删除一行中一个单元格的值，例如：

hbase> delete ‘hbase\_book’, ‘rowKey’, ‘CF:C’

### 29.11、truncate

禁用表-删除表-创建表

### 29.12、create

创建表，例如：

hbase> create ‘table’, ‘cf’

创建多个列族：

hbase> create 't1', {NAME => 'f1'}, {NAME => 'f2'}, {NAME => 'f3'}

### 29.13、更多后续拓展

# 三十、HBase节点的管理

## 30.1、服役（commissioning）

当启动regionserver时，regionserver会向Hmaster注册并开始接收本地数据，开始的时候，新加入的节点不会有任何数据，平衡器开启的情况下，将会有新的region移动到开启的RegionServer上。如果启动和停止进程是使用ssh和HBase脚本，那么会将新添加的节点的主机名加入到conf/regionservers文件中。

## 30.2、退役（decommissioning）

顾名思义，就是从当前HBase集群节点中删除某个RegionServer，这个过程分为如下几步：

### Step1、使用以下命令停止负载平衡器

hbase> balance\_switch false

### Step2、在退役节点上停止RegionServer

hbase> hbase-daemon.sh stop regionserver

### Step3、RegionServer一旦停止，会关闭维护的所有region

### Step4、Zookeeper上的该RegionServer节点消失

### Step5、Master节点检测到该RegionServer下线

### Step6、RegionServer的region服务得到重新分配

该关闭方法比较传统，需要花费一定的时间，而且会造成部分region短暂的不可用。

我们有一种更骚气的关闭方法：

### Step1、RegionServer先卸载所管理的region

$ bin/graceful\_stop.sh <RegionServer-hostname>

例如：

$ bin/graceful\_stop.sh hadoop-senior02.itguigu.com

### Step2、自动平衡数据

### Step3、和之前的2~6步是一样的

# 三十一、集群失效的可能性

集群机器上不同的Java版本可能会引起崩溃问题，不同版本的Hadoop和HBase也会引起兼容性问题，如下几种组件出现问题，也会导致集群失效。

### 30.1、磁盘

例如磁盘转速过慢，导致一些其他服务超时，或者磁盘损坏等。

### 30.2、操作系统

例如操作系统在升级补丁时，升级到了一个无法兼容当前集群的版本。

### 30.3、网络

网络震荡，阻塞等原因造成的集群服务连接超时。

### 30.4、内存

内存的损坏或者内存溢出，或者在执行一些具体任务时，由于编码问题造成的内存泄漏导致了内存溢出。

# 三十二、HBase数据的备份与恢复

HBase中存储着大量的在线业务数据和离线业务数据，所以备份对于我们而言会非常重要。接下来讲解一下如何执行在线和离线备份。

## 32.1、离线备份

此方法时完全停止HBase服务后，使用distcp命令运行MapReduce任务进行备份，将数据备份到另一个地方，可以是同一个集群，也可以是专用的备份集群。如果你的集群是线上集群，不能下线停止服务，则此方法不可取，请阅读在线备份方法。

### 32.1.1、同一集群的备份

即，把数据转移到当前集群的其他目录下，可以使用命令：

$ bin/hadoop distcp \

hdfs://hadoop-senior01.itguigu.com:8020/hbase \

hdfs://hadoop-senior01.itguigu.com:8020/HbaseBackup/backup20170803

### 32.1.2、备份数据到另外一个集群

即，把数据复制转移到另一个集群的存储空间中，使用如下命令：

$ bin/hadoop distcp \

hdfs://hadoop-senior01.itguigu.com:8020/hbase \

hdfs://z01:8020/HbaseBackup/backup20170803

这种备份方式其实更适用于建立专门的备份集群。

尖叫提示：复制时，一定要开启Yarn服务。

### 32.1.3、离线恢复

与备份方法一样，将数据整个移动回来即可。

## 32.1、在线备份

即，在HBase不离线的情况下，使用快照来备份数据。

# 三十三、HBase的过滤器

这里所说的是布隆过滤器，就是当需要在扫描过程中过滤掉大部分数据的时候使用，它可以减少内部数据的查找，从而加快扫描速度。这些数据存储在Hfiles的元数据区，一旦写入就不会再改变。布隆过滤器通过折叠的方式减少占用的空间，当分配region给RegionServer时，Hfile被打开，布隆过滤器载入内存。

## 33.1、主要功能：提高随机读性能

## 33.2、参考文章：<http://blog.csdn.net/opensure/article/details/46453681，http://blog.csdn.net/dadoneo/article/details/6847481>

# 三十四、HBase的协处理

## 34.1、简介

协处理器（coprocessor），每个RegionServer都会有一个协处理器进程，所有的region都包含对协处理器实现的引用。可以通过RegionServer的类路径加载，也可以通过HDFS类加载器加载。协处理器的设计是为了方便开发者向HBase中添加额外的功能，而不是给普通使用者使用。它可以用来做服务器端的操作，比如region分裂、合并以及客户端的创建，读取，删除，更新等操作，实现用户自定义功能。

## 34.2、协处理器类型

### 34.2.1、Coprocessor

提供region生命周期的管理，比如打开，关闭，分裂，合并等等。

### 34.2.2、RegionServer

提供对表修稿操作的监控，比如get，put，scan，delete

### 34.2.3、Endpoint

提供在region端执行任意函数的功能，比如RegionServer上的列聚集函数。

# 三十五、HBase版本确界

## 35.1、版本是下界

默认的版本下界是0，即禁用。row版本使用的最小数目是与生存时间（TTL Time To Live）相结合的，并且我们根据实际需求可以有0或更多的版本，使用0，即只有1个版本的值写入cell。

## 35.2、版本的上界

默认的版本上界是3，也就是一个row保留3个副本（基于时间戳的插入）。该值不要设计的过大，一般的业务不会超过100。如果cell中存储的数据版本号超过了3个，再次插入数据时，最新的值会将最老的值覆盖。

# 三十六、HBase在目前行业前景中的应用

## 36.1、FaceBook



### 36.1.1、为何选择

\* 提高读写吞吐量

\* 相比于其他数据库有更高的随机读取性能

\* 水平可伸缩性

\* 自动故障转移

\* 超强的一致性

\* 超强的容错性

### 36.1.2、在HBase中存储什么

\* 即时消息

\* 搜索数据

### 36.1.3、流程控制

客户端请求用户目录服务获取用户详细信息，得到用户详细信息之后，客户端发送请求给应用服务器，其可以是一个运行特定应用程序的Tomcat服务器，使用定制服务的应用程序可以通过HBase索引来搜索用户相关的单元格数据，单元格可以存放写入的消息、追加或者检索的数据。

### 36.1.4、规模

**每天：**

\* 消息量：发送和接收的消息数超过60亿

\* 将近1000亿条数据的读写

\* 高峰期每秒150万左右操作

\* 整体读取数据占有约55%，写入占有45%

\* 超过2PB的数据，涉及冗余共6PB数据

\* 数据使用LZO压缩

\* 数据每月大概增长300千兆字节。

# 三十七、一些优化技巧

## 37.1、通用优化

### 37.1.1、NameNode的元数据备份使用SSD

### 37.1.2、定时备份NameNode上的元数据，每小时或者每天备份，如果数据极其重要，可以5~10分钟备份一次。备份可以通过定时任务复制元数据目录即可。

### 37.1.3、为NameNode指定多个元数据目录，使用dfs.name.dir或者dfs.namenode.name.dir指定。一个指定本地磁盘，一个指定网络磁盘。这样可以提供元数据的冗余和健壮性，以免发生故障。

### 37.1.4、设置dfs.namenode.name.dir.restore为true，允许尝试恢复之前失败的dfs.namenode.name.dir目录，在创建checkpoint时做此尝试，如果设置了多个磁盘，建议允许。

### 37.1.5、NameNode节点必须配置为RAID1（镜像盘）结构。

### 37.1.6、补充：什么是Raid0、Raid0+1、Raid1、Raid5



**Standalone**

最普遍的单磁盘储存方式。

**Cluster**

集群储存是通过将数据分布到集群中各节点的存储方式,提供单一的使用接口与界面,使用户可以方便地对所有数据进行统一使用与管理。

**Hot swap**

用户可以再不关闭系统,不切断电源的情况下取出和更换硬盘,提高系统的恢复能力、拓展性和灵活性。

**Raid0**

Raid0是所有raid中存储性能最强的阵列形式。其工作原理就是在多个磁盘上分散存取连续的数据,这样,当需要存取数据是多个磁盘可以并排执行,每个磁盘执行属于它自己的那部分数据请求,显著提高磁盘整体存取性能。但是不具备容错能力,适用于低成本、低可靠性的台式系统。

**Raid1**

又称镜像盘,把一个磁盘的数据镜像到另一个磁盘上,采用镜像容错来提高可靠性,具有raid中最高的数据冗余能力。存数据时会将数据同时写入镜像盘内,读取数据则只从工作盘读出。发生故障时,系统将从镜像盘读取数据,然后再恢复工作盘正确数据。这种阵列方式可靠性极高,但是其容量会减去一半。广泛用于数据要求极严的应用场合,如商业金融、档案管理等领域。只允许一颗硬盘出故障。

**Raid0+1**

将Raid0和Raid1技术结合在一起,兼顾两者的优势。在数据得到保障的同时,还能提供较强的存储性能。不过至少要求4个或以上的硬盘，但也只允许一个磁盘出错。是一种三高技术。

**Raid5**

Raid5可以看成是Raid0+1的低成本方案。采用循环偶校验独立存取的阵列方式。将数据和相对应的奇偶校验信息分布存储到组成RAID5的各个磁盘上。当其中一个磁盘数据发生损坏后,利用剩下的磁盘和相应的奇偶校验信息 重新恢复/生成丢失的数据而不影响数据的可用性。至少需要3个或以上的硬盘。适用于大数据量的操作。成本稍高、储存新强、可靠性强的阵列方式。

RAID还有其他方式，请自行查阅。

### 37.1.7、保持NameNode日志目录有足够的空间，这些日志有助于帮助你发现问题。

### 37.1.8、因为Hadoop是IO密集型框架，所以尽量提升存储的速度和吞吐量（类似位宽）。

## 37.2、Linux优化

### 37.2.1、开启文件系统的预读缓存可以提高读取速度

$ sudo blockdev --setra 32768 /dev/sda

（尖叫提示：ra是readahead的缩写）

### 37.2.2、关闭进程睡眠池

$ sudo sysctl -w vm.swappiness=0

### 37.2.3、调整ulimit上限，默认值为比较小的数字

$ ulimit -n 查看允许最大进程数

$ ulimit -u 查看允许打开最大文件数

修改：

$ sudo vi /etc/security/limits.conf 修改打开文件数限制

末尾添加：

\* soft nofile 1024000

\* hard nofile 1024000

Hive - nofile 1024000

hive - nproc 1024000

$ sudo vi /etc/security/limits.d/20-nproc.conf 修改用户打开进程数限制

修改为：

#\* soft nproc 4096

#root soft nproc unlimited

\* soft nproc 40960

root soft nproc unlimited

### 37.2.4、开启集群的时间同步NTP，请参看之前文档

### 37.2.5、更新系统补丁（尖叫提示：更新补丁前，请先测试新版本补丁对集群节点的兼容性）

## 37.3、HDFS优化（hdfs-site.xml）

### 37.3.1、保证RPC调用会有较多的线程数

属性：dfs.namenode.handler.count

解释：该属性是NameNode服务默认线程数，的默认值是10，根据机器的可用内存可以调整为50~100

属性：dfs.datanode.handler.count

解释：该属性默认值为10，是DataNode的处理线程数，如果HDFS客户端程序读写请求比较多，可以调高到15~20，设置的值越大，内存消耗越多，不要调整的过高，一般业务中，5~10即可。

### 37.3.2、副本数的调整

属性：dfs.replication

解释：如果数据量巨大，且不是非常之重要，可以调整为2~3，如果数据非常之重要，可以调整为3~5。

### 37.3.3.、文件块大小的调整

属性：dfs.blocksize

解释：块大小定义，该属性应该根据存储的大量的单个文件大小来设置，如果大量的单个文件都小于100M，建议设置成64M块大小，对于大于100M或者达到GB的这种情况，建议设置成256M，一般设置范围波动在64M~256M之间。

## 37.4、MapReduce优化（mapred-site.xml）

### 37.4.1、Job任务服务线程数调整

mapreduce.jobtracker.handler.count

该属性是Job任务线程数，默认值是10，根据机器的可用内存可以调整为50~100

### 37.4.2、Http服务器工作线程数

属性：mapreduce.tasktracker.http.threads

解释：定义HTTP服务器工作线程数，默认值为40，对于大集群可以调整到80~100

### 37.4.3、文件排序合并优化

属性：mapreduce.task.io.sort.factor

解释：文件排序时同时合并的数据流的数量，这也定义了同时打开文件的个数，默认值为10，如果调高该参数，可以明显减少磁盘IO，即减少文件读取的次数。

### 37.4.5、设置任务并发

属性：mapreduce.map.speculative

解释：该属性可以设置任务是否可以并发执行，如果任务多而小，该属性设置为true可以明显加快任务执行效率，但是对于延迟非常高的任务，建议改为false，这就类似于迅雷下载。

### 37.4.6、MR输出数据的压缩

属性：mapreduce.map.output.compress、mapreduce.output.fileoutputformat.compress

解释：对于大集群而言，建议设置Map-Reduce的输出为压缩的数据，而对于小集群，则不需要。

### 37.4.7、优化Mapper和Reducer的个数

属性：

mapreduce.tasktracker.map.tasks.maximum

mapreduce.tasktracker.reduce.tasks.maximum

解释：以上两个属性分别为一个单独的Job任务可以同时运行的Map和Reduce的数量。

设置上面两个参数时，需要考虑CPU核数、磁盘和内存容量。假设一个8核的CPU，业务内容非常消耗CPU，那么可以设置map数量为4，如果该业务不是特别消耗CPU类型的，那么可以设置map数量为40，reduce数量为20。这些参数的值修改完成之后，一定要观察是否有较长等待的任务，如果有的话，可以减少数量以加快任务执行，如果设置一个很大的值，会引起大量的上下文切换，以及内存与磁盘之间的数据交换，这里没有标准的配置数值，需要根据业务和硬件配置以及经验来做出选择。

在同一时刻，不要同时运行太多的MapReduce，这样会消耗过多的内存，任务会执行的非常缓慢，我们需要根据CPU核数，内存容量设置一个MR任务并发的最大值，使固定数据量的任务完全加载到内存中，避免频繁的内存和磁盘数据交换，从而降低磁盘IO，提高性能。

大概配比：

|  |  |  |  |
| --- | --- | --- | --- |
| CPU CORE | MEM（GB） | Map | Reduce |
| 1 | 1 | 1 | 1 |
| 1 | 5 | 1 | 1 |
| 4 | 5 | 1~4 | 2 |
| 16 | 32 | 16 | 8 |
| 16 | 64 | 16 | 8 |
| 24 | 64 | 24 | 12 |
| 24 | 128 | 24 | 12 |

大概估算公式：

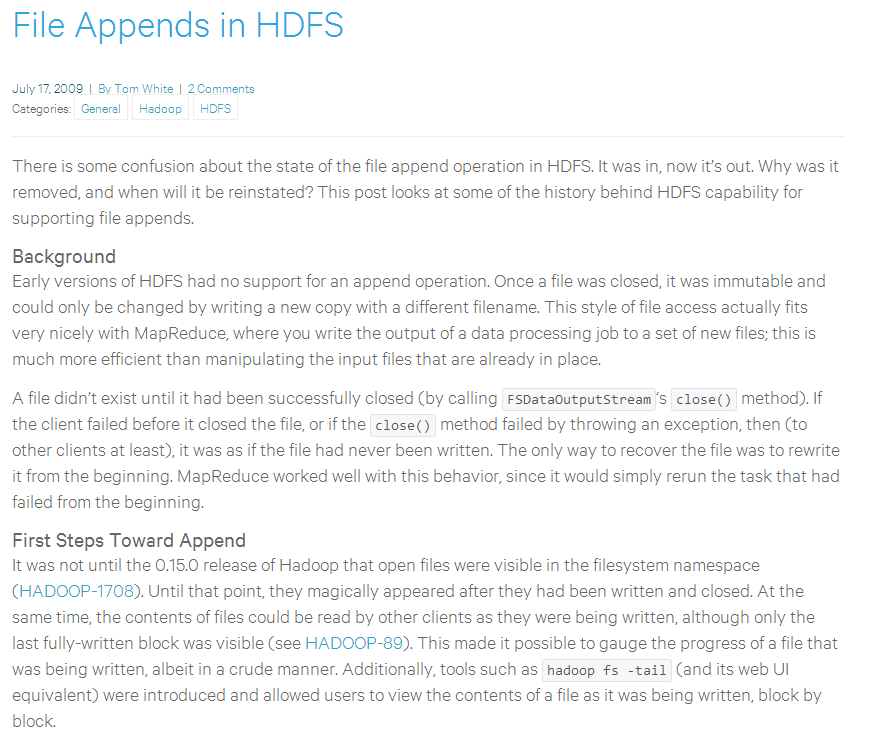
map = 2 + ⅔cpu\_core

reduce = 2 + ⅓cpu\_core

## 37.5、HBase优化

### 37.5.1、在HDFS的文件中追加内容

不是不允许追加内容么？没错，请看背景故事：



属性：dfs.support.append

文件：hdfs-site.xml、hbase-site.xml

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。

### 37.5.2、优化DataNode允许的最大文件打开数

属性：dfs.datanode.max.transfer.threads

文件：hdfs-site.xml

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

### 37.5.3、优化延迟高的数据操作的等待时间

属性：dfs.image.transfer.timeout

文件：hdfs-site.xml

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。

### 37.5.4、优化数据的写入效率

属性：

mapreduce.map.output.compress

mapreduce.map.output.compress.codec

文件：mapred-site.xml

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

### 37.5.5、优化DataNode存储

属性：dfs.datanode.failed.volumes.tolerated

文件：hdfs-site.xml

解释：默认为0，意思是当DataNode中有一个磁盘出现故障，则会认为该DataNode shutdown了。如果修改为1，则一个磁盘出现故障时，数据会被复制到其他正常的DataNode上，当前的DataNode继续工作。

### 37.5.6、设置RPC监听数量

属性：hbase.regionserver.handler.count

文件：hbase-site.xml

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。

### 37.5.7、优化HStore文件大小

属性：hbase.hregion.max.filesize

文件：hbase-site.xml

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。

### 37.5.8、优化hbase客户端缓存

属性：hbase.client.write.buffer

文件：hbase-site.xml

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。

### 37.5.9、指定scan.next扫描HBase所获取的行数

属性：hbase.client.scanner.caching

文件：hbase-site.xml

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。

## 37.6、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。

## 37.7、JVM优化

涉及文件：hbase-env.sh

### 37.7.1、并行GC

参数：-XX:+UseParallelGC

解释：开启并行GC

### 37.7.2、同时处理垃圾回收的线程数

参数：-XX:ParallelGCThreads=cpu\_core – 1

解释：该属性设置了同时处理垃圾回收的线程数。

### 37.7.3、禁用手动GC

参数：-XX:DisableExplicitGC

解释：防止开发人员手动调用GC

## 37.8、Zookeeper优化

### 37.8.1、优化Zookeeper会话超时时间

参数：zookeeper.session.timeout

文件：hbase-site.xml

解释：In hbase-site.xml, set zookeeper.session.timeout to 30 seconds or less to bound failure detection (20-30 seconds is a good start).该值会直接关系到master发现服务器宕机的最大周期，默认值为30秒，如果该值过小，会在HBase在写入大量数据发生而GC时，导致RegionServer短暂的不可用，从而没有向ZK发送心跳包，最终导致认为从节点shutdown。一般20台左右的集群需要配置5台zookeeper。

# 三十八、HBase表类型的设计

## 38.1、短宽

这种设计一般适用于：

\* 有大量的列

\* 有很少的行

## 38.2、高瘦

这种设计一般适用于：

\* 有很少的列

\* 有大量的行

## 38.3、短宽-高瘦的对比

### 38.3.1、短宽

\* 使用列名进行查询不会跳过行或者存储文件

\* 更好的原子性

\* 不如高瘦设计的可扩展性

### 38.3.2、高瘦

\* 如果使用ID进行查询，会跳过行

\* 不利于原子性

\* 更好的扩展

# 三十九、HBase的预分区

## 39.1、为何要预分区？

\* 增加数据读写效率

\* 负载均衡，防止数据倾斜

\* 方便集群容灾调度region

\* 优化Map数量

## 39.2、如何预分区？

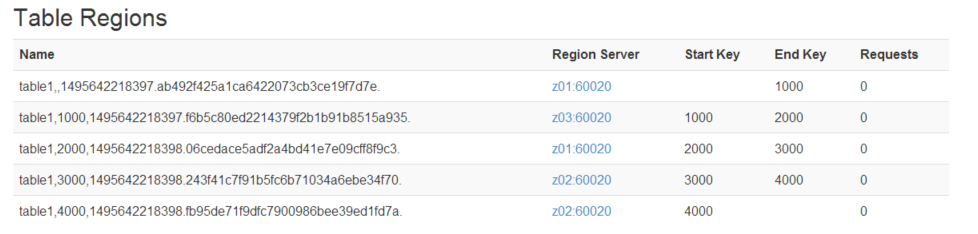
每一个region维护着startRow与endRowKey，如果加入的数据符合某个region维护的rowKey范围，则该数据交给这个region维护。

## 39.3、如何设定预分区？

### 39.3.1、手动指定预分区

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

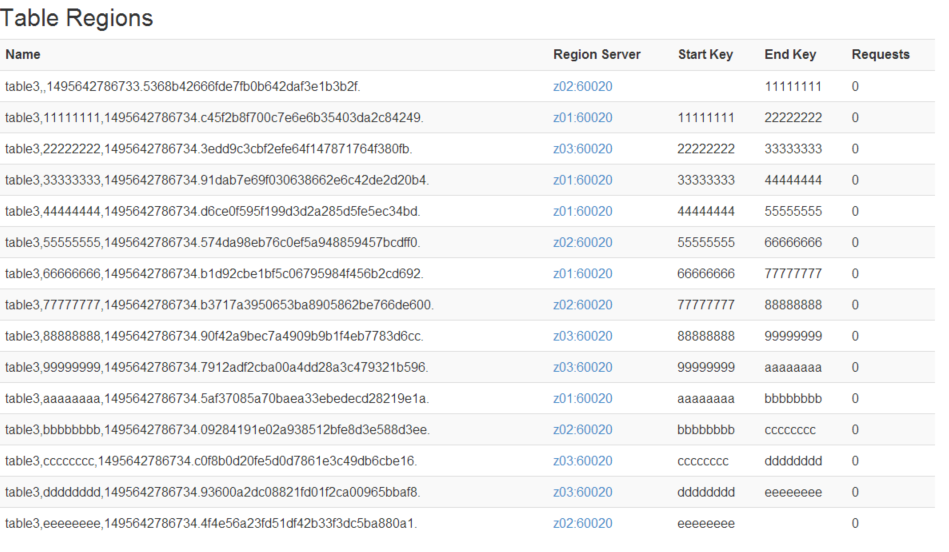
完成后如图：



### 39.3.2、使用16进制算法生成预分区

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

完成后如图：



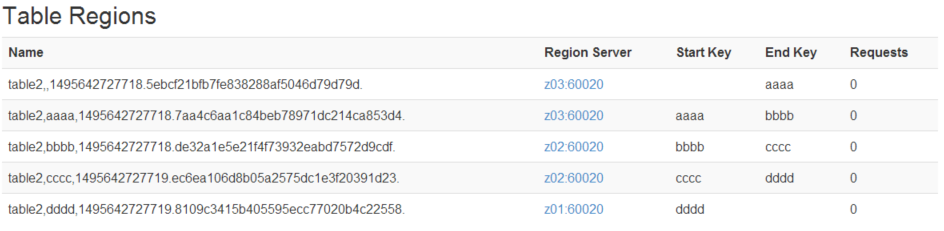
### 39.3.3、分区规则创建于文件中

创建splits.txt文件内容如下：



然后执行：

create 'table2','partition2',SPLITS\_FILE => 'splits.txt'，成功后如图：



### 39.3.4、使用JavaAPI创建预分区

Java代码如下：



|  |
| --- |
| //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); |

# 四十、HBase的rowKey设计技巧

## 40.1、设计宗旨与目标

主要目的就是针对特定的业务模型，按照rowKey进行预分区设计，使之后面加入的数据能够尽可能的分散于不同的rowKey中。比如复合RowKey。

## 40.2、设计方式案例

### 40.2.1、案例一：生成随机数、hash、散列值

比如：

原本rowKey为1001的，MD5后变成：b8c37e33defde51cf91e1e03e51657da

原本rowKey为3001的，MD5后变成：908c9a564a86426585b29f5335b619bc

原本rowKey为5001的，MD5后变成：03b264c595403666634ac75d828439bc

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。

### 40.2.2、案例二：字符串反转

比如：

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。

### 40.2.3、案例三：字符串拼接

比如：

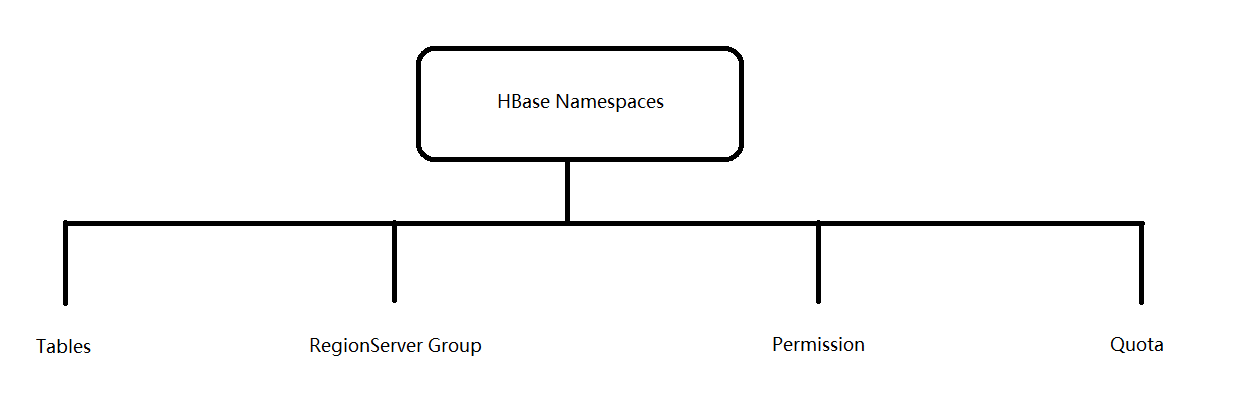
20170524000001\_a12e

20170524000001\_93i7

# 四十一、设计案例

## 41.1、牵扯概念：命名空间

### 41.1.1、命名空间结构图



### 41.1.2、组成部分

**-Table**：表，所有的表都是命名空间的成员，即表必属于某个命名空间，如果没有指定，则在default默认的命名空间中。

**-RegionServer group：**一个命名空间包含了默认的RegionServer Group。

**-Permission：**权限，命名空间能够让我们来定义访问控制列表ACL（Access Control List）。例如，创建表，读取表，删除，更新等等操作。

**-Quota**：限额，可以强制一个命名空间可包含的region的数量

### 41.1.3、命名空间命令

**-创建命名空间**

例如：

|  |
| --- |
| hbase(main):002:0> create\_namespace 'student\_namespace' |

**-创建表时指定命名空间**

例如：

|  |
| --- |
| hbase(main):004:0> create 'student\_namespace:student\_table','student\_info' |

### 41.1.4、观察HDFS中的目录结构的变化

如图所示：



## 41.2、微博项目

### 41.2.1、需求分析

-微博内容的浏览，数据库表设计

-用户社交体现：关注用户，取关用户

-拉取关注的人的微博内容

### 41.2.2、步骤拆解

**创建命名空间**

**创建微博内容表**

|  |  |
| --- | --- |
| **方法** | creatTableeContent |
| Table Name | weibo:content |
| RowKey | 用户ID\_时间戳 |
| ColumnFamily | info |
| ColumnLabel | 标题 内容 图片 |
| Version | 1个版本 |

**创建用户关系表**

|  |  |
| --- | --- |
| **方法** | createTableRelations |
| Table Name | weibo:relations |
| RowKey | 用户ID |
| ColumnFamily | attends、fans |
| ColumnLabel | 关注用户ID，粉丝用户ID |
| ColumnValue | 用户ID |
| Version | 1个版本 |

**创建用户微博内容接收邮件表**

|  |  |
| --- | --- |
| **方法** | createTableReceiveContentEmails |
| Table Name | weibo:receive\_content\_email |
| RowKey | 用户ID |
| ColumnFamily | info |
| ColumnLabel | 用户ID |
| ColumnValue | 取微博内容的RowKey |
| Version | 1000 |

**发布微博内容**

1. 微博内容表中添加1条数据
2. 微博收件箱表对所有粉丝用户添加数据

**添加关注用户**

1. 在微博用户关系表中，对当前主动操作的用户添加新关注的好友
2. 在微博用户关系表中，对被关注的用户添加新的粉丝
3. 微博收件箱表中添加所关注的用户发布的微博

**移除（取关）用户**

a、在微博用户关系表中，对当前主动操作的用户移除取关的好友(attends)

b、在微博用户关系表中，对被取关的用户移除粉丝

c、微博收件箱中删除取关的用户发布的微博

**获取关注的人的微博内容**

a、从微博收件箱中获取所关注的用户的微博RowKey

b、根据获取的RowKey，得到微博内容

### 41.2.3、代码实现

见代码。

# End、HUE工具集成

## End.1、HUE简介

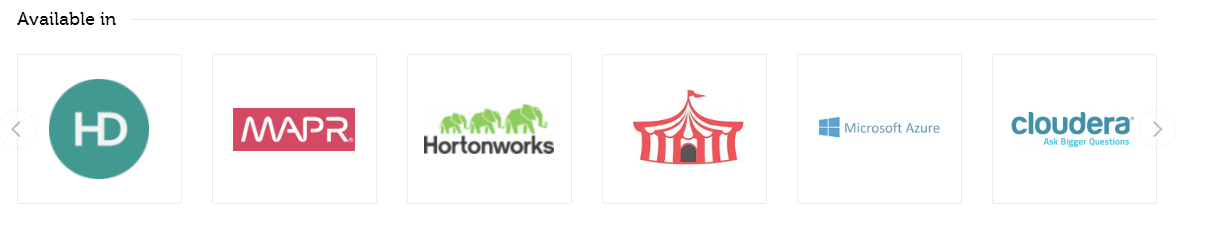
### End.1.1、来源

HUE=HadoopUser Experience，看这名字就知道怎么回事了吧，没错，直白来说就是Hadoop用户体验，是一个开源的Apache Hadoop UI系统，由Cloudera Desktop演化而来，最后Cloudera公司将其贡献给Apache基金会的Hadoop社区，它是基于Python Web框架Django实现的。通过使用HUE我们可以在浏览器端的Web控制台上与Hadoop集群进行交互来分析处理数据。

### End.1.2、官网及使用者

官网网站：<http://gethue.com/>

使用的公司：



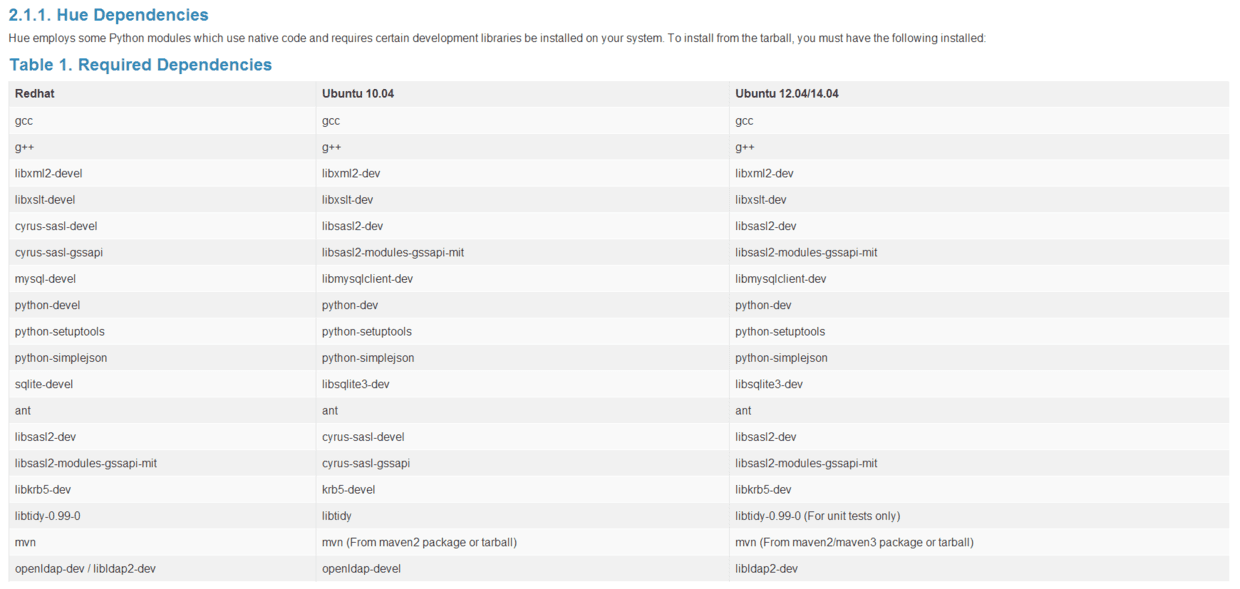
## End.2、安装HUE

### End.2.1、帮助文档

<http://archive.cloudera.com/cdh5/cdh/5/hue-3.7.0-cdh5.3.0/manual.html>

### End.2.2、准备环境依赖

打开官方网站可以找到对应的部署HUE所需的各种依赖列表，如下图所示：



如上图所示，这部分内容是告诉你，安装编译Hue需要依赖哪些Linux安装包，你只需要使用yum命令一次安装就可以了，在此给大家整理好该命令（注意使用root权限安装）：

# yum -y install ant asciidoc cyrus-sasl-devel cyrus-sasl-gssapi gcc gcc-c++ krb5-devel libtidy libxml2-devel libxslt-devel openldap-devel python-devel sqlite-devel openssl-devel mysql-devel gmp-devel

（尖叫提示：使用yum安装这些包的同时，也会自动安装openJDK的依赖，所以，请自行删除安装后的openJDK，忘记的同学请参考Linux基础）

（查询：# rpm -qa | grep java）

（删除：# rpm -e --nodeps xxxxxxx-java-xxxx.rpm）

### End.2.3、解压HUE

$ tar -zxf /opt/softwares/hue-3.7.0-cdh5.3.6.tar.gz -C /opt/modules/cdh/

### End.2.4、编译HUE

到hue安装目录下，执行make apps

$ make apps

尖叫提示：使用普通用户编译

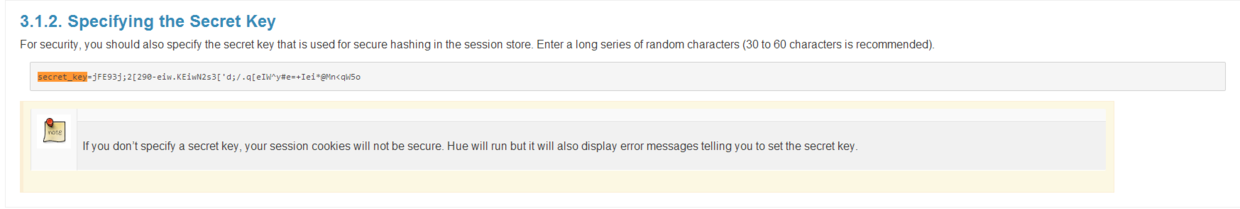
大概等个几分钟之后，就编译成功了。

### End.2.5、配置HUE

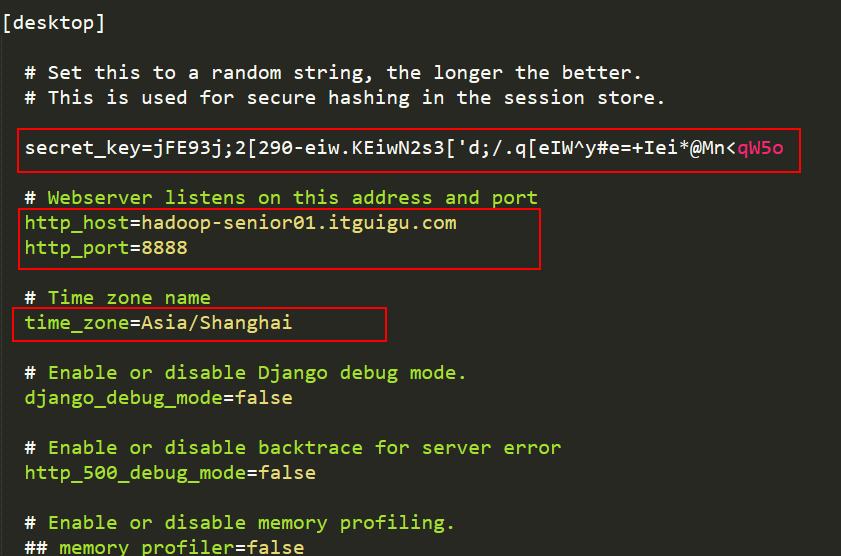
修改Hue.ini文件

文件位置：/opt/modules/cdh/hue-3.7.0-cdh5.3.6/desktop/conf/hue.ini

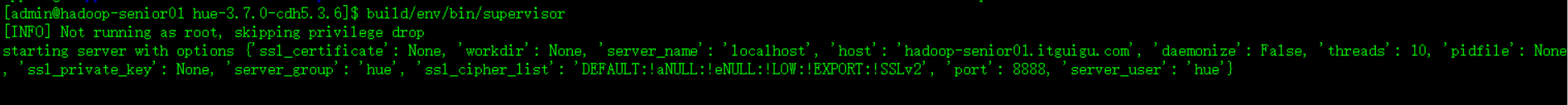
其中的secret\_key请参照官方网站配置：



修改内容参照如下：

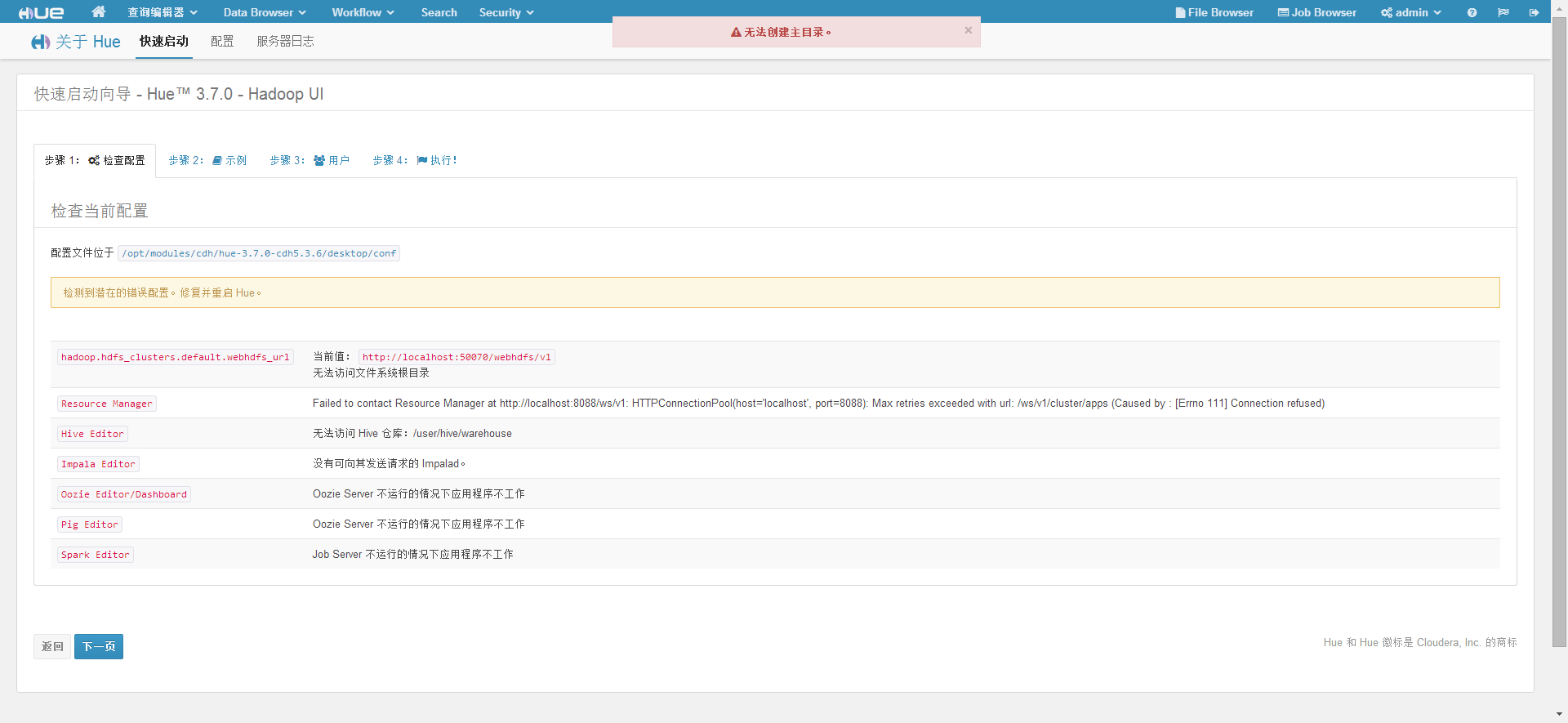


### End.2.6、启动HUE

完成之后呢，保存退出，我们来使用命令启动Hue$ build/env/bin/supervisor，出现如下界面表示启动成功： 

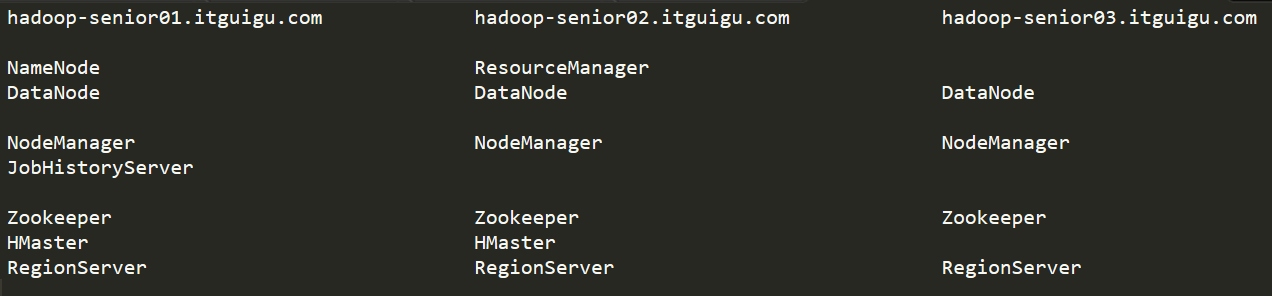
接下来使用浏览器来查看hue界面：http://hadoop-senior01.itguigu.com:8888，接着我们就看到如下界面：



这句话是在提示你，第一次使用本工具，需要创建一个用户及密码，且会成为hue的超级用户凭证，在此呢，我设置为admin用户名，密码随意，那就123456吧，然后呢就可以见到如下界面了： 

## End.3、HUE与HDFS

### End.3.1、梳理集群环境



### End.3.2、配置hdfs-site.xml

**添加**

属性：dfs.webhdfs.enabled

属性值：true

解释：Enable WebHDFS (REST API) in Namenodes and Datanodes.

### End.3.3、配置core-site.xml

添加

属性：hadoop.proxyuser.hue.hosts

变更为：hadoop.proxyuser.admin.hosts

属性值：\*

解释：代理的用户

属性：hadoop.proxyuser.hue.groups

变更为：hadoop.proxyuser.admin.groups

属性值：\*

解释：代理的用户组

如果你的Hadoop配置了高可用，则必须通过httpfs来访问，需要添加如下属性，反则则不必须。（如果HUE服务与Hadoop服务不在同一节点，则必须配置）

属性：hadoop.proxyuser.hue.hosts

变更为：hadoop.proxyuser.httpfs.hosts

属性值：\*

属性：hadoop.proxyuser.hue.groups

变更为：hadoop.proxyuser.httpfs.groups

属性值：\*

### End.3.4、httpfs-site.xml

**添加**

属性：httpfs.proxyuser.hue.hosts

属性值：\*

属性：httpfs.proxyuser.hue.groups

属性值：\*

解释：以上两个属性主要用于HUE服务与Hadoop服务不在同一台节点上所必须的配置。

尖叫提示：

\* 如果没有配置NameNode的HA，HUE可以用WebHDFS来管理HDFS

\* 如果配置了NameNodeHA，则HUE只可用HttpFS来管理HDFS

### End.3.5、scp同步配置

$ scp -r etc/ hadoop-senior02.itguigu.com:/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/

$ scp -r etc/ hadoop-senior03.itguigu.com:/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/

### End.3.6、启动httpfs服务

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/httpfs.sh start &

### End.3.7、配置hue.ini

找到[hadoop]标签



### End3.8、测试

$ build/env/bin/supervisor

打开HUE的页面，进行HDFS管理。

尖叫提示：

如果提示错误根目录应该归属于hdfs，请修改python变量，位置如下：

/opt/modules/cdh/hue-3.7.0-cdh5.3.6/desktop/libs/hadoop/src/hadoop/fs/webhdfs.py

修改其中的变量值为：

DEFAULT\_HDFS\_SUPERUSER = 'admin'

然后重启HUE服务即可。

尖叫提示：

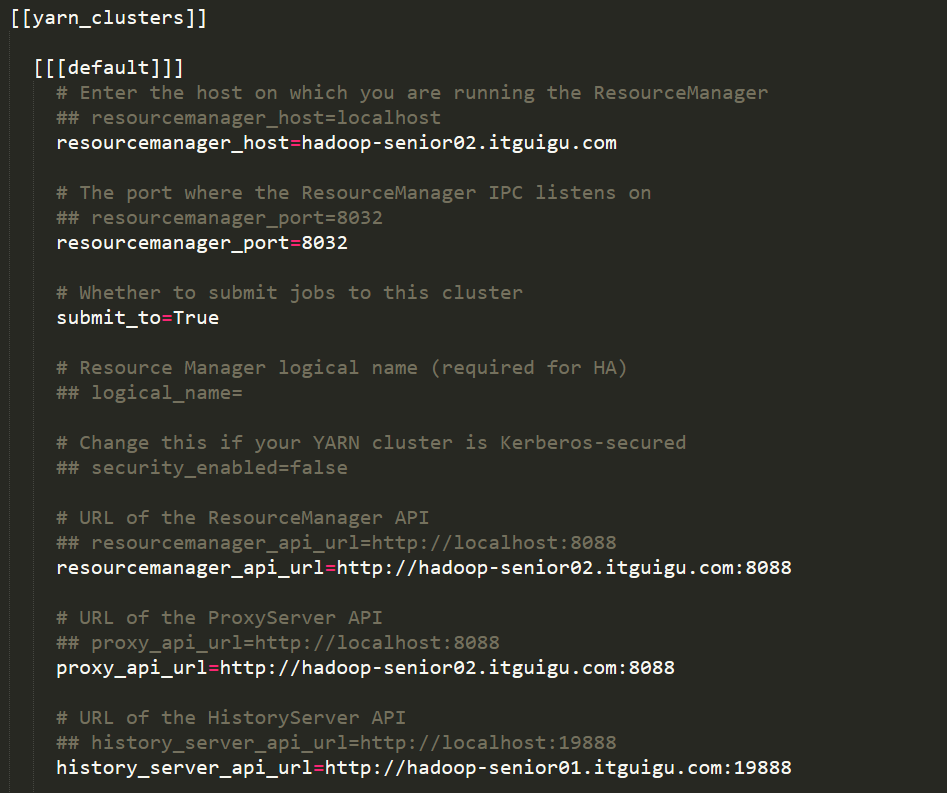
启动HUE服务时，请先kill掉之前的HUE服务，如果提示地址被占用，请使用如下命令查看占用8888端口的进程并kill掉：

$ netstat -tunlp | grep 8888

## End.4、HUE与YARN

### End.4.1、配置hue.ini

找到[[yarn\_clusters]]标签，修改配置如下图所示：



### End.4.2、重启HUE测试查看

$ build/env/bin/supervisor

## End.5、HUE与Hive

### End.5.1、修改Hive配置文件hive-site.xml

HUE与hive集成需要hive开启HiveServer2服务

属性：hive.server2.thrift.port

属性值：10000

属性：hive.server2.thrift.bind.host

属性值：hadoop-senior01.itguigu.com

属性：hive.server2.long.polling.timeout

属性值：5000

属性：hive.metastore.uris

属性值：thrift://hadoop-senior01.itguigu.com:9083

### End.5.2、启动Hive

$ bin/hive --service metastore &

$ bin/hive --service hiveserver2 &

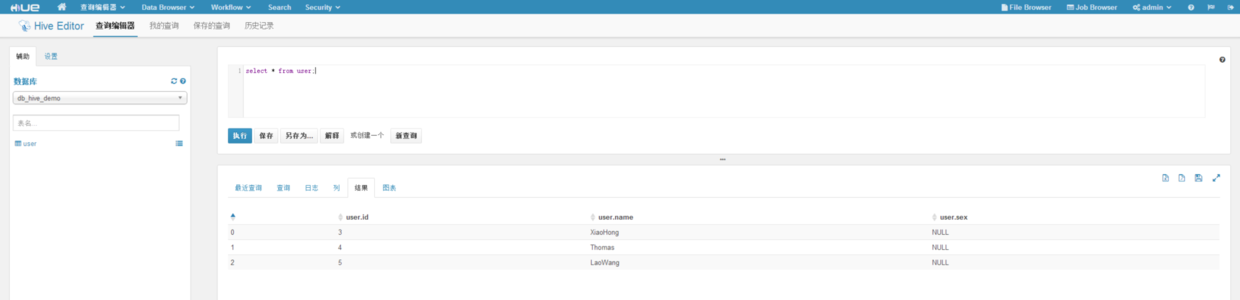
尖叫提示：如果设置了uris，在今后使用Hive时，那么必须启动如上两个命令，否则Hive无法正常启动。

### End.5.3、配置hue.ini

找到[beeswax]属性标签，配置如图：



### End.5.4、重启hue进行Hive测试



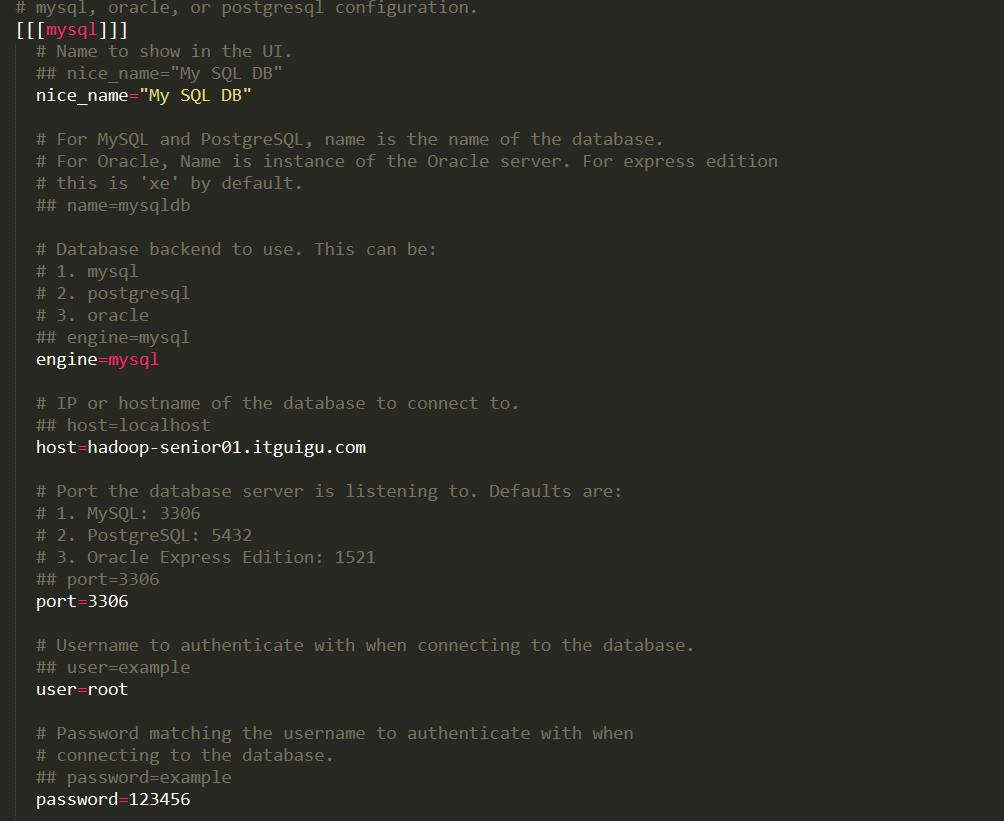
来到这样的界面进行简单的查询即可测试

$ build/env/bin/supervisor

## End.6、HUE与Mysql

### End.6.1、配置hue.ini

找到[[[mysql]]]标签，并删掉标签注释，配置如下：



### End.6.2、重启hue.ini测试

启动后即可测试是否成功连接Mysql服务，并且测试是否可以看到数据

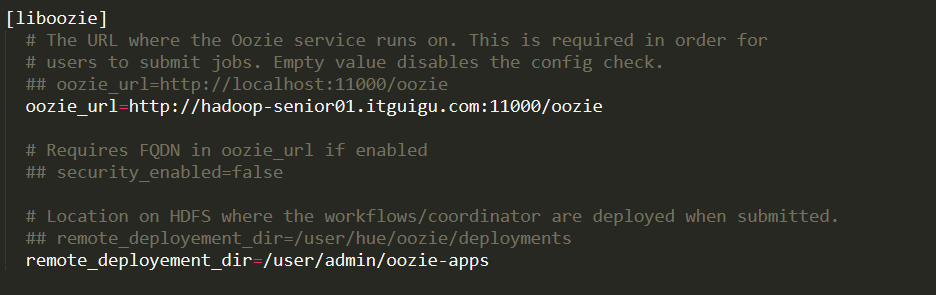
$ build/env/bin/supervisor

## End.7、HUE与Oozie

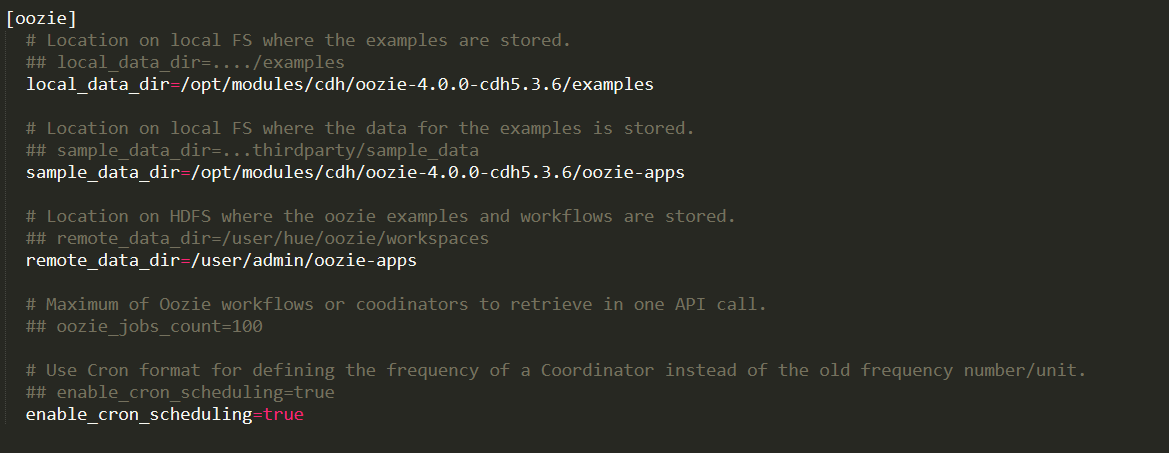
### End.7.1、配置hue.ini

找到[liboozie]标签以及[oozie]标签配置如下

[liboozie]：



[oozie]：



### End.7.2、启动Oozie相关服务

$ bin/oozied.sh start

### End.7.3、重启HUE测试查看Oozie

$ build/env/bin/supervisor

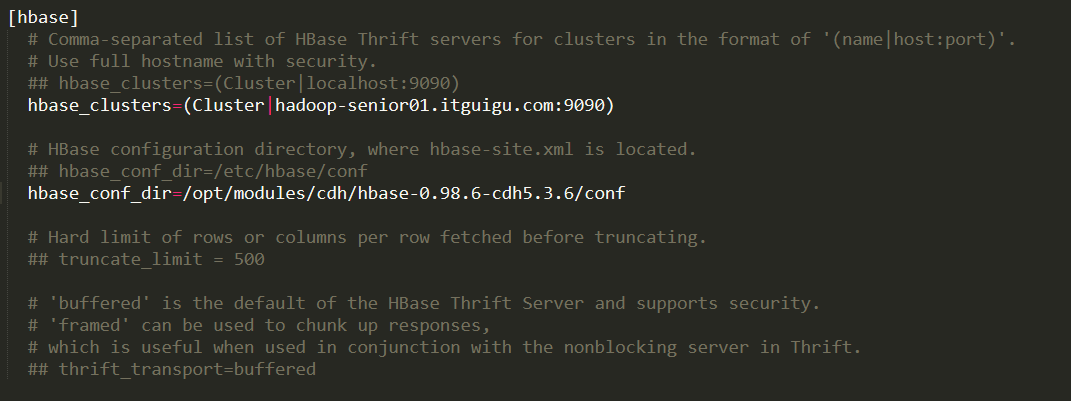
尖叫提示：如果提示无法关联oozie的share/lib，请使用hdfs命令创建该目录即可：

$ bin/hdfs dfs -mkdir -p /user/oozie/share/lib

## End.8、HUE与HBase

### End.8.1、修改hue.ini配置

找到[hbase]标签，修改内容如图：



### End.8.2、启动HBase的thrift服务

$ bin/hbase-daemon.sh start thrift

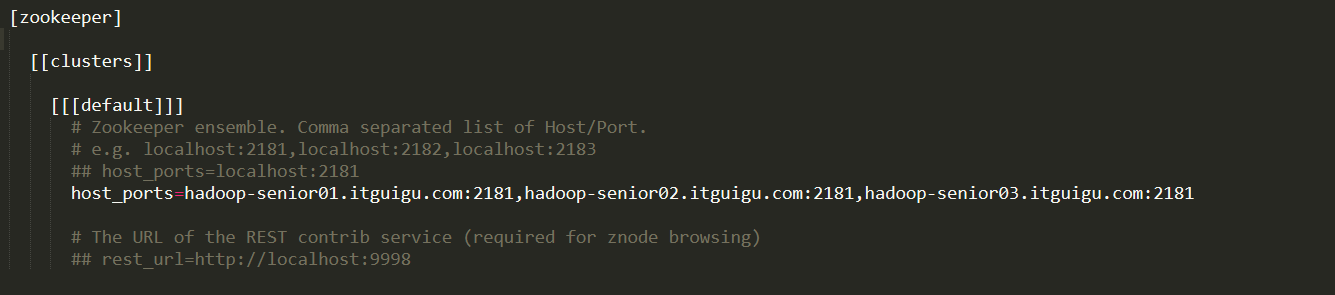
### End.8.3、重启HUE进行测试

$ build/env/bin/supervisor

# End.9、HUE与Zookeeper

### End.9.1、配置hue.ini

找到[zookeeper]标签，配置如下：



### End.9.2、重启HUE查看即可

$ build/env/bin/supervisor

# End.10、HUE与Sqoop2

### End.10.1、如何配置

尖叫提示：HUE只支持Sqoop2的集成，不支持Sqoop1，在此不再演示。

http://archive.cloudera.com/cdh5/cdh/5/hue-3.7.0-cdh5.3.0/user-guide/sqoop.html

# End.11、总结

在此我们总结一下集成HUE时，我们开启的后台服务项

### End.11.1、Hadoop

$ /opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/httpfs.sh start &

### End.11.2、Hive

$ /opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service metastore &

$ /opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service hiveserver2 &

### End.11.3、HBase

$ /opt/modules/cdh/hbase-0.98.6-cdh5.3.6/bin/hbase-daemon.sh start thrift &

### End.11.4、Oozie

$ /opt/modules/cdh/oozie-4.0.0-cdh5.3.6/bin/oozied.sh start &

为了方便，我们把这些服务加在群起脚本中，如图所示：



完整脚本如下：

|  |
| --- |
| #!/bin/bash  echo "===================================正在开启集群服务======================================="  echo "===================================正在开启Zookeeper节点=================================="  for i in admin@hadoop-senior01.itguigu.com admin@hadoop-senior02.itguigu.com admin@hadoop-senior03.itguigu.com  do  ssh $i '/opt/modules/cdh/zookeeper-3.4.5-cdh5.3.6/bin/zkServer.sh start'  done  echo "===================================正在开启NameNode节点=================================="  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/hadoop-daemon.sh start namenode'  echo "===================================正在开启DataNode节点=================================="  for i in admin@hadoop-senior01.itguigu.com admin@hadoop-senior02.itguigu.com admin@hadoop-senior03.itguigu.com  do  ssh $i '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/hadoop-daemon.sh start datanode'  done  echo "===================================正在开启SeondaryNameNode节点=========================="  ssh admin@hadoop-senior03.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/hadoop-daemon.sh start secondarynamenode'  echo "===================================正在开启ResourceManager节点==========================="  ssh admin@hadoop-senior02.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/yarn-daemon.sh start resourcemanager'  echo "===================================正在开启NodeManager节点==============================="  for i in admin@hadoop-senior01.itguigu.com admin@hadoop-senior02.itguigu.com admin@hadoop-senior03.itguigu.com  do  ssh $i '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/yarn-daemon.sh start nodemanager'  done  echo "===================================正在开启JobHistoryServer节点=========================="  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/mr-jobhistory-daemon.sh start historyserver'  echo "===================================正在开启HBase节点====================================="  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/bin/start-hbase.sh'  echo "===================================正在第1节点开启httpfs服务=============================="  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hadoop-2.5.0-cdh5.3.6/sbin/httpfs.sh start &'  echo "===================================正在第1节点开启metastore与hiveserver2服务=============="  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service metastore &'  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hive-0.13.1-cdh5.3.6/bin/hive --service hiveserver2 &'  echo "===================================正在第1节点开启HBase thrift服务========================"  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hbase-0.98.6-cdh5.3.6/bin/hbase-daemon.sh start thrift &'  echo "===================================正在第1节点开启Oozie服务==============================="  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/oozie-4.0.0-cdh5.3.6/bin/oozied.sh start &'  echo "===================================正在第1节点开启HUE服务================================="  ssh admin@hadoop-senior01.itguigu.com '/opt/modules/cdh/hue-3.7.0-cdh5.3.6/build/env/bin/supervisor &' |