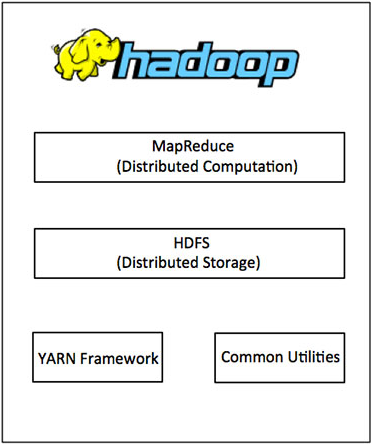
# 简介

Hadoop是由Apache基金会所开发的一个可靠的、可扩展的及可容错的分布式系统基础架构，其提供分布式大数据存储及计算功能。

Hadoop包括以下四个模块：

1. Hadoop Common（公共模块）：用于支撑其他模块；
2. Hadoop Distributed File System (HDFS，hadoop分布式文件系统): 提供高吞吐量（high throughput）来访问应用数据（海量数据存储能力）；
3. Hadoop YARN：hadoop作业调度和集群资源管理的框架；
4. Hadoop MapReduce：可提供海量数据计算的能力



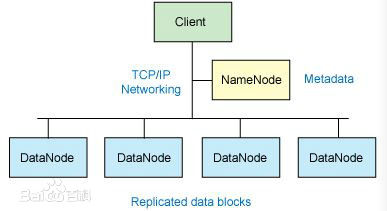
## Hadoop分布式文件系统（HDFS）描述

Hadoop分布式文件系统（HDFS）是基于谷歌文件系统（GFS），并提供了一个设计在普通硬件上运行的分布式文件系统。它与现有的分布式文件系统有许多相似之处。来自其他分布式文件系统的差别是显著。它高度容错并设计成部署在低成本的硬件。提供了高吞吐量的应用数据访问，并且适用于具有大数据集的应用程序。

## MapReduce描述

MapReduce是面向大数据并行处理的计算模型、框架和平台。

## 核心架构



对外部客户机而言，HDFS就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件等等。但是HDFS的架构是基于一组特定的节点构建的（如上图所示），这些节点包括 NameNode（仅一个），它在 HDFS 内部提供元数据服务；DataNode节点为 HDFS 提供存储块。由于仅存在一个 NameNode，因此这是 HDFS 的一个缺点（单点失败）。

存储在 HDFS 中的文件被分成块，然后将这些块复制到多个计算机中（DataNode）。这与传统的 RAID 架构大不相同。块的大小（通常为 64MB）和复制的块数量在创建文件时由客户机决定。NameNode 可以控制所有文件操作。HDFS 内部的所有通信都基于标准的 TCP/IP 协议。

### NameNode节点

NameNode 是一个通常在 HDFS 实例中的单独机器上运行的软件。它负责管理文件系统名称空间和控制外部客户机的访问。NameNode 决定是否将文件映射到 DataNode 上的复制块上。对于最常见的 3 个复制块，第一个复制块存储在同一机架的不同节点上，最后一个复制块存储在不同机架的某个节点上。注意，这里需要您了解集群架构。实际的 I/O事务并没有经过 NameNode，只有表示 DataNode 和块的文件映射的元数据经过 NameNode。当外部客户机发送请求要求创建文件时，NameNode 会以块标识和该块的第一个副本的 DataNode IP 地址作为响应。这个 NameNode 还会通知其他将要接收该块的副本的 DataNode。它执行以下任务：

* 管理文件系统命名空间。
* 规范客户端对文件的访问。
* 它也执行文件系统操作，如重命名，关闭和打开的文件和目录。

### DataNode节点

DataNode 也是一个通常在 HDFS实例中的单独机器上运行的软件。Hadoop 集群包含一个 NameNode 和大量 DataNode。DataNode 通常以机架的形式组织，机架通过一个交换机将所有系统连接起来。DataNode 响应来自 HDFS 客户机的读写请求。它们还响应来自 NameNode 的创建、删除和复制块的命令。ameNode 依赖来自每个 DataNode 的定期心跳（heartbeat）消息。每条消息都包含一个块报告，NameNode 可以根据这个报告验证块映射和其他文件系统元数据。如果 DataNode 不能发送心跳消息，NameNode 将采取修复措施，重新复制在该节点上丢失的块。根据客户的请求，数据节点上的文件系统执行的读写操作；还根据NameNode节点的指令执行操作，如块的创建，删除和复制。

## hadoop各个功能模块

### HDFS模块

HDFS负责大数据的存储，通过将大文件分块后进行分布式存储方式，突破了服务器硬盘大小的限制，解决了单台机器无法存储大文件的问题，HDFS是个相对独立的模块，可以为YARN提供服务，也可以为HBase等其他模块提供服务。

包括NameNode进程、SecondaryNameNode进程和多个DataNode进程。

1. NameNode

NameNode是整个文件系统的管理节点。它维护着1.整个文件系统的文件目录树，2.文件/目录的元信息和每个文件对应的数据块列表。3.接收用户的操作请求。

1. DataNode

DataNode提供真实文件数据的存储服务。

1. SecondaryNameNode

运行辅助namenode–Secondary NameNode，为主namenode内存中的文件系统元数据创建检查点，Secondary NameNode所做的不过是在文件系统中设置一个检查点来帮助NameNode更好的工作。它不是要取代掉NameNode也不是NameNode的备份，SecondaryNameNode有两个作用，一是镜像备份，二是日志与镜像的定期合并。两个过程同时进行，称为checkpoint（检查点）。可以不设定。

### YARN模块

YARN是一个通用的资源协同和任务调度框架，是为了解决Hadoop1.x中MapReduce里NameNode负载太大和其他问题而创建的一个框架。

YARN是个通用框架，不止可以运行MapReduce，还可以运行Spark、Storm等其他计算框架。

包括ResourceManager进程和多个NodeManager进程。

1. ResourceManager

在YARN中，ResourceManager负责集群中所有资源的统一管理和分配，它接收来自各个节点（NodeManager）的资源汇报信息，并把这些信息按照一定的策略分配给各个应用程序（实际上是ApplicationManager）。

1. NodeManager

NodeManager是执行在单个节点上的代理，它管理Hadoop集群中单个计算节点，功能包含与ResourceManager保持通信，管理Container的生命周期、监控每一个Container的资源使用(内存、CPU等）情况、追踪节点健康状况、管理日志和不同应用程序用到的附属服务等。

NodeManager是YARN中单个节点的代理，它须要与应用程序的ApplicationMaster和集群管理者ResourceManager交互;它从ApplicationMaster上接收有关Container的命令并执行(比方启动、停止Contaner);向ResourceManager汇报各个Container执行状态和节点健康状况，并领取有关Container的命令（比方清理Container）。

首先client向ResourceManager提交程序（包含ApplicationMaster程序，ApplicationMaster启动命令，用户程序）后，ResourceManager向资源调度器去申请资源，一旦申请的ApplicationMaster需要的资源，ApplicationMasterLaucher便与对应的NodeManager联系启动ApplicationMaster，同时向ApplicationMasterLivenessMonitor添加进监控列表，启动对ApplicationMaster的监控，ApplicationMaster启动后，向ApplicationMasterService注册报告自己的端口号，ip，track url等，之后ApplicationMaster会定期向ApplicationMasterService发送心跳，执行allocate，ApplicationMasterService会向ApplicationMasterLivenessMonitor更新ApplicationMaster的心跳时间，当用户程序执行完毕，ApplicationMaster向ApplicationMasterService报告完成，ApplicationMasterService通知ApplicationMasterLivenessMonitor从监控列表中删除ApplicationMaster，释放资源。

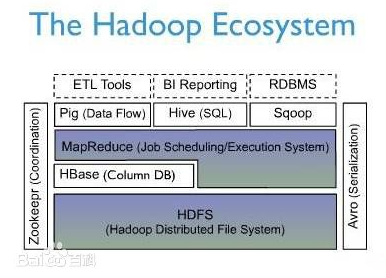
### MapReduce模块

MapReduce是一个计算框架，它给出了一种数据处理的方式，即通过Map阶段、Reduce阶段来分布式地流式处理数据。它只适用于大数据的离线处理，对实时性要求很高的应用不适用。

## hadoop优点

1. **高可靠性**。Hadoop按位存储和处理数据的能力值得人们信赖。
2. **高扩展性**。Hadoop是在可用的计算机集簇间分配数据并完成计算任务的，这些集簇可以方便地扩展到数以千计的节点中。
3. **高效性**。Hadoop能够在节点之间动态地移动数据，并保证各个节点的动态平衡，因此处理速度非常快。
4. **高容错性**。Hadoop能够自动保存数据的多个副本，并且能够自动将失败的任务重新分配。
5. **低成本**。与一体机、商用数据仓库以及QlikView、Yonghong Z-Suite等数据集市相比，hadoop是开源的，项目的软件成本因此会大大降低。

## hadoop子项目



### Ambari

### Avro

### Cassandra

### Chukwa

### Hbase

HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用HBase技术可在廉价PC Server上搭建起大规模结构化存储集群。

HBase是一个分布式的、面向列的开源数据库，HBase不同于一般的关系数据库，它是一个适合于非结构化数据存储的数据库。另一个不同的是HBase基于列的而不是基于行的模式。

HBase利用Hadoop HDFS作为其文件存储系统，利用Hadoop MapReduce来处理HBase中的海量数据。HBase位于结构化[存储](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8)层，Hadoop HDFS为HBase提供了高可靠性的底层存储支持，Hadoop MapReduce为HBase提供了高性能的计算能力，Zookeeper为HBase提供了稳定服务和failover机制。此外，Pig和Hive还为HBase提供了高层语言支持，使得在HBase上进行数据统计处理变的非常简单。 Sqoop则为HBase提供了方便的RDBMS数据导入功能，使得[传统数据库](https://baike.baidu.com/item/%E4%BC%A0%E7%BB%9F%E6%95%B0%E6%8D%AE%E5%BA%93)数据向HBase中迁移变的非常方便。

### Hive

hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。 其优点是学习成本低，可以通过类SQL语句快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

Hive是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工作。hive数据存储特点：

首先，Hive 没有专门的数据存储格式，也没有为数据建立索引，用户可以非常自由的组织 Hive 中的表，只需要在创建表的时候告诉 Hive 数据中的列分隔符和行分隔符，Hive 就可以解析数据。

其次，Hive 中所有的数据都存储在 HDFS 中，Hive 中包含以下数据模型：表(Table)，外部表(External Table)，分区(Partition)，桶(Bucket)。

### Mahout

### Pig

### Spark

Apache Spark是一个基于内存计算的分布式计算框架，旨在简化运行于计算机集群上的并行程序的编写。该框架对资源调度，任务的提交、执行和跟踪，节点间的通信以及数据并行处理的内在底层操作都进行了抽象。它提供了一个更高级别的API用于处理分布式数据。从这方面说，它与Apache Hadoop等分布式处理框架类似。但在底层架构上，Spark与它们有所不同。

### Tez

### ZooKeeper

# 生产环境搭建

本生产环境只设置两个节点，一个NameNode节点（192.168.1.119）及两个DataNode节点（192.168.1.120及192.168.1.121），系统版本为CentOS Linux release 7.3.1611。

以后可以增加DataNode节点。

## IP设置

首先是进入/etc/sysconfig/network-scripts中查看现有的配置文件，需要使用的配置文件是：**ifcfg-enoXX。在root用户下，使用vim命令编辑。该文件的内容为：**

TYPE=Ethernet

BOOTPROTO=dhcp

DEFROUTE=yes

PEERDNS=yes

PEERROUTES=yes

IPV4\_FAILURE\_FATAL=no

IPV6INIT=yes

IPV6\_AUTOCONF=yes

IPV6\_DEFROUTE=yes

IPV6\_PEERDNS=yes

IPV6\_PEERROUTES=yes

IPV6\_FAILURE\_FATAL=no

NAME=eno16777736

UUID=ae05ccde-6a29-4332-b486-f3042da73ac0

DEVICE=eno16777736

ONBOOT=no

修改后：

TYPE=Ethernet

BOOTPROTO=static

DEFROUTE=yes

PEERDNS=yes

PEERROUTES=yes

IPV4\_FAILURE\_FATAL=yes

IPADDR=192.168.1.121

GATEWAY=192.168.1.1

NETMASK=255.255.255.0

DNS1=192.168.1.1

IPV6INIT=yes

IPV6\_AUTOCONF=yes

IPV6\_DEFROUTE=yes

IPV6\_PEERDNS=yes

IPV6\_PEERROUTES=yes

IPV6\_FAILURE\_FATAL=no

IPV6\_ADDR\_GEN\_MODE=stable-privacy

NAME=ens33

UUID=7ea36373-d4e8-4368-885c-318eeff57f2c

DEVICE=ens33

ONBOOT=yes

保存退出后，使用service network restart重启网络服务。最后使用ifconfig查看配置后的IP。

## 修改主机名

在root用户下执行hostname或hostnamectl命令可以查看主机名，随后使用hostnamectl set-hostname xxx命令（永久修改）更改主机名，执行完后不需要重启系统，重新打开连接即可。对应机器ip的主机名如下：

192.168.1.119：namenode

192.168.1.120：datanode1

192.168.1.121：datanode2

集群中的所有机器需要能够通过主机名连接到相应的机器（如ssh hostname），必须手动更新/etc/hosts，将集群所有机器的ip及相应的主机对应上，使用vi /etc/hosts命令编辑即可。对每台机器所需添加的内容都一致。本案例需要添加以下两行。

192.168.1.119 namenode

192.168.1.120 datanode1

192.168.1.121 datanode2

## 创建一个单独的用户

在开始时，建议创建一个单独的用户hadoop（可以起别的用户名），以从Unix文件系统隔离Hadoop文件系统。建议所有节点的用户名与密码均相同。【所有节点】操作命令如下：

[root@localhost ~]# useradd hadoop

[root@localhost ~]# passwd hadoop

Changing password for user hadoop.

New password:

BAD PASSWORD: The password is shorter than 8 characters

Retype new password:

passwd: all authentication tokens updated successfully.

修改密码时，需要输入两次。

之后，使用su - hadoop命令跳转新建的用户，用户新建的用户继续完成环境搭建。

## 关闭防火墙

CentOS 7.0默认使用的是firewall作为防火墙：

1. firewall-cmd --state//查看默认防火墙状态（关闭后显示notrunning，开启后显示running）
2. systemctl status firewalld.service
3. 关闭防火墙

systemctl stop firewalld.service #停止firewall  
 systemctl disable firewalld.service #禁止firewall开机启动

1. 其他

启动：systemctl start firewalld.service

关闭：systemctl stop firewalld.service

重启：systemctl restart firewalld.service

显示状态：systemctl status firewalld.service

开机时启用：systemctl enable firewalld.service

开机时禁用：systemctl disable firewalld.service

查看是否开机启动：systemctl is-enabled firewalld.service

查看已启动的服务列表：systemctl list-unit-files|grep enabled

注：每台机器上执行

关闭：systemctl stop firewalld.service

开机时禁用：systemctl disable firewalld.service

## SSH免密码登录

配置好SSH免密码登录之后，机器之间无须用户输入密码即可相互访问。下面介绍安装与配置过程：

1. 安装SSH：只需使用相应的用户（root或hadoop）执行一次ssh跳转到其他机器即可生成。
2. 生成公私钥：cd ~/.ssh，然后执行ssh-keygen -t rsa（连续按几次回车键即可），在目录下将生成id\_rsa.pub与id\_rsa两个密钥文件。其中id\_rsa.pub为公钥。
3. 将公钥id\_rsa.pub拷贝到指定一台机器上：首先使用cp -r ~/.ssh/id\_rsa.pub ~/.ssh/authorized\_keys将该机上的公钥放入authorized\_keys文件，然后在其他所有机器上使用：ssh-copy-id -i ~/.ssh/id\_rsa.pub <用户名>@IP地址，把这些机器的公钥也放入这台机器的authorized\_keys文件，最后修改authorized\_keys文件权限：chmod 600 ~/.ssh/authorized\_keys。
4. 把指定机器上的authorized\_keys文件拷贝到其他所有的机器上。scp -r authorized\_keys <用户名>@IP地址:~/.ssh/authorized\_keys

注：

* 第一、第二步每台服务器都必须执行一遍；
* 第三步指定一台机器，将其公钥先cp到authorized\_keys文件，然后使用ssh-copy-id把其他机器的公钥也放入这台机器的authorized\_keys文件。
* 如果是使用root用户操作，则仅对root用户可以免密码跳转到别的机器；其他用户（如hadoop）需要再执行一次操作才可以面密码登录。
* sshd为了安全，对属主的目录和文件权限有所要求。如果权限不对，则ssh的免密码登陆不生效。用户目录权限为 755 或者 700，就是不能是77x。.ssh目录权限一般为755或者700。rsa\_id.pub 及authorized\_keys权限一般为644，rsa\_id权限必须为600。

## 安装Jdk

1. 下载最新版本的jdk（建议使用jdk8），下载地址：http://www.oracle.com/technetwork/java/javase/downloads/index.html
2. 使用root用户操作，为了使Java提供给所有用户，将它放到目录 “/usr/local/”。解压：tar -zxvf jdk-8u171-linux-x64.tar.gz -C /usr/local/
3. 配置环境变量（vi /etc/profile）

export JAVA\_HOME=/usr/local/jdk1.8.0\_171

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export PATH=${JAVA\_HOME}/bin:$PATH

1. 重新加载：source /etc/profile
2. 测试：java –version（不同用户验证结果一致）

注：第二步在一台机器上执行解压即可，然后使用scp拷贝到其他机上。

## 集群服务器校时

Hadoop对集群中各个机器的时间同步要求比较高，要求各个机器的系统时间不能相差太多，不然会造成很多问题。可以配置集群中各个机器和互联网的时间服务器进行时间同步，但是在实际生产环境中，集群中大部分服务器是不能连接外网的，这时候可以在内网搭建一个自己的时间服务器（NTP服务器），集群的各个机器与这个时间服务器进行时间同步。

### 配置NTP服务器

1. 检查ntp服务是否已经安装（centos 7自带，不需要安装）

root用户下执行rpm -qa | grep ntp

ntp-4.2.6p5-25.el7.centos.x86\_64

python-ntplib-0.3.2-1.el7.noarch

fontpackages-filesystem-1.44-8.el7.noarch

ntpdate-4.2.6p5-25.el7.centos.x86\_64

显示已经安装过了ntp程序，其中ntpdate-4.2.6p5-25.el7.centos.x86\_64是用来和某台服务器进行同步的，ntp-4.2.6p5-25.el7.centos.x86\_64是用来提供时间同步服务的。

1. 修改配置文件ntp.conf（/etc/ntp.conf）

restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap

将这行的注释去掉，并且将网段改为集群的网段，这里网段是192.168.1。如果不添加，表明任何网段均可向该服务器校时。

以下是时间服务器的域名，默认是向centos校时，可以改掉server 时间服务器地址（域名或IP）。

server 0.centos.pool.ntp.org iburst

server 1.centos.pool.ntp.org iburst

server 2.centos.pool.ntp.org iburst

server 3.centos.pool.ntp.org iburst

1. 修改配置文件ntpd（/etc/sysconfig/ntpd）

添加一行：SYNC\_CLOCK=yes

1. 启动ntp服务

systemctl enable ntpd.service（开机自启动，一定要加）

systemctl status ntpd.service

systemctl restart ntpd.service

systemctl stop ntpd.service

systemctl start ntpd.service

### NTP校时设置

首先设定定时器，进行定时校时。使用“crontab -e”命令添加定时校时服务。每十分钟校时一次（实际中不需要这么频繁校时）：

02 \* \* \* \* /usr/sbin/ntpdate namenode >> /home/cctsoft/logs/ntpCrontabLog.log &

然后重启定时器：systemctl restart crond.service，即可。

## 安装hadoop

hadoop安装，全程使用hadoop用户执行。hadoop支持以下三种模式，下面主要从完全分布式模式讲解：

* 本地/独立模式：默认情况下之后，它会被配置在一个独立的模式，无需进行其他配置即可运行，用于运行Java程序，适合于开发期间运行MapReduce程序。测试如下：

第一条命令：

cd /home/hadoop/hadoop-2.9.0

第二条命令：

./bin/hadoop jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.0.jar

执行完，可以看到该程序所实现的一些功能，并可测试这些功能。

* 模拟分布式模式：这是在单台机器的分布式模拟。Hadoop守护每个进程，如 hdfs, yarn, MapReduce等，都将作为一个独立的java程序运行。Hadoop 可以在单节点上以伪分布式的方式运行，Hadoop 进程以分离的 Java 进程来运行，节点既作为 NameNode 也作为DataNode，同时，读取的是 HDFS 中的文件。
* 完全分布式模式：这种模式是完全分布式的最小两台或多台计算机的集群。
* HA完全分布式模式：

区分的依据是NameNode、DataNode、ResourceManager、NodeManager等模块运行在几个JVM进程、几个机器。

| **模式名称** | **各个模块占用的JVM进程数** | **各个模块运行在几个机器数上** |
| --- | --- | --- |
| 本地模式 | 1个 | 1个 |
| 伪分布式模式 | N个 | 1个 |
| 完全分布式模式 | N个 | N个 |
| HA完全分布式 | N个 | N个 |

### hadoop本地模式安装

本地模式是最简单的模式，所有模块都运行与一个JVM进程中，使用的本地文件系统，而不是HDFS，本地模式主要是用于本地开发过程中的运行调试用。下载hadoop安装包后不用任何设置，默认的就是本地模式。

运行hadoop自带的mapreduce单词统计程序例子：

bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.4.jar wordcount /home/hadoop/wc.input /home/hadoop/output（/home/hadoop/wc.input是存放单词的文件，/home/hadoop/output是结果输出目录）

### hadoop伪分布式安装

1. 登录hadoop官网http://hadoop.apache.org/releases.html下载（其他地址https://archive.apache.org/dist/hadoop/common/，下载稳定本版stable）
2. 解压到/home/hadoop目录下：tar -zxvf hadoop-2.7.4.tar.gz -C /opt/
3. 配置环境变量（vi ~/.bashrc），并执行source ~/.bashrc使其生效（也可以添加到/etc/profile文件中）

export HADOOP\_HOME=/opt/hadoop-2.7.4

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export HADOOP\_YARN\_HOME=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

export HDFS\_CONF\_DIR=$HADOOP\_HDFS\_HOME/etc/hadoop

export HADOOP\_LOG\_DIR=/home/hadoop/bigdata/logs

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

1. 在/opt/hadoop-2.7.4/etc/hadoop/hadoop-env.sh中，再显示地重新声明一遍JAVA\_HOME，即写明具体的存放目录。

export JAVA\_HOME=/usr/local/jdk1.8.0\_171

1. 配置core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://namenode:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>file:/home/bigdata/tmp</value>

</property>

<property>

<name>io.file.buffer.size</name>

<value>131072</value>

</property>

</configuration>

注：需要创建/home/bigdata/tmp目录。

1. 配置hdfs-site.xml

<configuration>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:/home/bigdata/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:/home/bigdata/datanode</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.blocksize</name>

<value>256m</value>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>namenode:50070</value>

</property>

</configuration>

注：需要创建/home/bigdata/namenode及/home/bigdata/datanode目录。

1. 配置mapred-site.xml

mapreduce计算模型配置。

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<property>

<name>mapreduce.map.memory.mb</name>

<value>128</value>

</property>

<property>

<name>mapreduce.map.java.opts</name>

<value>-Xmx512M</value>

</property>

<property>

<name>mapreduce.reduce.memory.mb</name>

<value>128</value>

</property>

<property>

<name>mapreduce.reduce.java.opts</name>

<value>-Xmx320M</value>

</property>

<property>

<name>mapreduce.task.io.sort.mb</name>

<value>150</value>

</property>

<property>

<name>mapreduce.task.io.sort.factor</name>

<value>100</value>

</property>

<property>

<name>mapreduce.reduce.shuffle.parallelcopies</name>

<value>50</value>

</property>

<property>

<name>mapreduce.application.classpath</name>

<value>

$HADOOP\_HOME/share/hadoop/mapreduce/\*,

$HADOOP\_HOME/share/hadoop/mapreduce/lib/\*

</value>

</property>

</configuration>

1. 配置yarn-site.xml

<configuration>

<property>

<name>yarn.acl.enable</name>

<value>false</value>

</property>

<property>

<name>yarn.admin.acl</name>

<value>\*</value>

</property>

<property>

<name>yarn.log-aggregation-enable</name>

<value>false</value>

</property>

<property>

<name>yarn.nodemanager.log.retain-seconds</name>

<value>3600</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>namenode:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>namenode:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>namenode:8031</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address</name>

<value>namenode:8033</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address</name>

<value>namenode:8088</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.https.address</name>

<value>namenode:8090</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>

1. 格式化HDFS

$HADOOP\_HOME/bin/hdfs namenode –format（仅第一次运行执行）

1. 启动

$HADOOP\_HOME/sbin/start-all.sh

或者

$HADOOP\_HOME/sbin/start-dfs.sh

$HADOOP\_HOME/sbin/start-yarn.sh

1. 停止

$HADOOP\_HOME/sbin/stop-all.sh

或者

$HADOOP\_HOME/sbin/stop-yarn.sh

$HADOOP\_HOME/sbin/stop-dfs.sh

注：可以使用jps命令查看相应的进程。

1. 示例（统计单词，这是基于hdfs）

新建一个文件夹存放文件：

$HADOOP\_HOME/bin/hdfs dfs -mkdir -p /wordcountdemo/input

上传文件到hdfs：

$HADOOP\_HOME/bin/hdfs dfs -put /home/hadoop/wc.input /wordcountdemo/input

统计单词数：

$HADOOP\_HOME/bin/yarn jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.4.jar wordcount /wordcountdemo/input/ /wordcountdemo/output

### hadoop分布式集群安装

三台机器部署情况：

|  |  |  |
| --- | --- | --- |
| **namenode** | **datanode1** | **datanode2** |
| NameNode  ResourceManager | DataNode  NodeManager  SecondaryNameNode | DataNode  NodeManager |

1. 登录hadoop官网http://hadoop.apache.org/releases.html下载（其他地址https://archive.apache.org/dist/hadoop/common/，下载稳定本版stable）
2. 解压到/opt/目录下：tar -zxvf hadoop-2.7.4.tar.gz -C /opt/

注：其他机器在配置好后，再使用scp拷贝相应的目录即可。

1. 配置环境变量（vi ~/.bashrc），并执行source ~/.bashrc使其生效，三个节点都需要配置。（也可以添加到/etc/profile文件中）

export HADOOP\_HOME=/opt/hadoop-2.7.4-distributed

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export HADOOP\_YARN\_HOME=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

export HDFS\_CONF\_DIR=$HADOOP\_HDFS\_HOME/etc/hadoop

export HADOOP\_LOG\_DIR=/home/bigdata/logs

export YARN\_LOG\_DIR=/home/bigdata/yarn/logs

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

1. 在/opt/hadoop-2.7.4-distributed/etc/hadoop/hadoop-env.sh中，再显示地重新声明一遍JAVA\_HOME，即写明具体的存放目录。

export JAVA\_HOME=/usr/local/jdk1.8.0\_171

1. 配置hadoop

进入Hadoop配置目录：/opt/hadoop-2.7.4-distributed/etc/hadoop，对slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml等 5个文件进行配置。

* slaves文件

将作为 DataNode 的主机名写入该文件，每行一个数据节点。本案例中有两个数据节点datanode1与datanode2，在文件中加入以下两行。

datanode1

datanode2

* 配置core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://namenode:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>file:/home/bigdata/tmp</value>

</property>

<property>

<name>io.file.buffer.size</name>

<value>131072</value>

</property>

</configuration>

|  |  |
| --- | --- |
| **配置项** | **描述** |
| fs.defaultFS | 缺省的hdfs访问URL入口设定，ip或主机名（namenode节点主机名）及端口号。 |
| hadoop.tmp.dir | 临时数据存放目录设定，加file:表示本地目录，事先手动建立目录。 |
| io.file.buffer.size | 流文件的缓冲区，设置太大会消耗内存131072B（128KB） |

* 配置hdfs-site.xml

创建/home/bigdata/namenode和/home/bigdata/datanode目录，并配置路径。

<configuration>

<property>

<name>dfs.namenode.http-address</name>

<value>namenode:50070</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:/home/bigdata/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:/home/bigdata/datanode</value>

</property>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.blocksize</name>

<value>256m</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>datanode1:50090</value>

</property>

<property>

<name>dfs.namenode.handler.count</name>

<value>20</value>

</property>

<property>

<name>dfs.datanode.handler.count</name>

<value>40</value>

</property>

</configuration>

|  |  |
| --- | --- |
| **配置项** | **描述** |
| dfs.namenode.name.dir | namenode数据的存放地点。也就是namenode元数据存放的地方，记录了hdfs系统中文件的元数据 |
| dfs.datanode.data.dir | datanode数据的存放地点。也就是block块存放的目录了。 |
| dfs.replication | hdfs的副本数设置，副本数目不能大于datanode数目，默认是3。 |
| dfs.replication.min（非必需） | 块副本的最少数量，默认是1 |
| dfs.replication.max（非必需） | 块副本的最多数量，默认是512 |
| dfs.namenode.http-address | Namenode节点的WEB UI管理端口。格式：0.0.0.0:50070 |
| **dfs.namenode.handler.count** | 设置更多的namenode线程，处理从 datanode发出的大量RPC请求，默认是10，建议设置大点。 |
| **dfs.datanode.handler.count** | datanode节点处理线程，默认是10，建议设置大点。 |
| dfs.namenode.secondary.http-address（必须配置，将其设置在一个数据节点上，不要配置在同一台机器上） | 备机namenode节点的WEB UI访问地址与端口，因为hadoop容易出现单点故障，hadoop做了namenode备份节点，一旦提供服务的namenode出现故障，由备份节点提供服务。参考值：0.0.0.0:50090，配置另外一台机器的ip |
| dfs.webhdfs.enabled（非必需） | 访问namenode的hdfs使用50070端口，访问datanode的webhdfs使用50075端口。访问文件、文件夹信息使用namenode的IP和50070端口，访问文件内容或者进行打开、上传、修改、下载等操作使用datanode的IP和50075端口。要想不区分端口，直接使用namenode的IP和端口进行所有的webhdfs操作，就需要在所有的datanode上都设置hefs-site.xml中的dfs.webhdfs.enabled为true。 |
| dfs.hosts  （用/分割ip或主机名，考虑到安全性可以设置） | 控制所包括的数据节点。 |
| dfs.hosts.exclude  （用户, 分割ip或主机名，考虑到安全性可以设置） | 控制不包括的数据节点。 |
| **dfs.blocksize** | HDFS块大小，k(kilo), m(mega), g(giga), t(tera), p(peta), e(exa) to specify the size (such as 128k, 512m, 1g, etc.) |

* 配置mapred-site.xml

mapreduce计算模型配置。

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<property>

<name>mapreduce.map.memory.mb</name>

<value>512</value>

</property>

<property>

<name>mapreduce.map.java.opts</name>

<value>-Xmx1024M</value>

</property>

<property>

<name>mapreduce.reduce.memory.mb</name>

<value>512</value>

</property>

<property>

<name>mapreduce.reduce.java.opts</name>

<value>-Xmx1024M</value>

</property>

<property>

<name>mapreduce.task.io.sort.mb</name>

<value>150</value>

</property>

<property>

<name>mapreduce.task.io.sort.factor</name>

<value>100</value>

</property>

<property>

<name>mapreduce.reduce.shuffle.parallelcopies</name>

<value>50</value>

</property>

<property>

<name>mapreduce.application.classpath</name>

<value>

$HADOOP\_HOME/share/hadoop/mapreduce/\*,

$HADOOP\_HOME/share/hadoop/mapreduce/lib/\*

</value>

</property>

</configuration>

|  |  |
| --- | --- |
| **配置项** | **描述** |
| mapreduce.framework.name | 运行mapreduce的执行与调度框架，配置在yarn上运行。 |
| mapreduce.map.memory.mb | 每个Map Task需要的内存量，默认1024MB |
| mapreduce.map.java.opts | Map的jvm内存，大概是0.6至0.75的mapreduce.map.memory.mb大小 |
| mapreduce.reduce.memory.mb | 每个Reduce Task需要的内存量，默认1024MB |
| mapreduce.reduce.java.opts | reduce的jvm内存，大概是0.6至0.75的mapreduce.reduce.memory.mb大小 |
| mapreduce.task.io.sort.mb | 任务内部文件排序缓冲区大小，默认100MB |
| mapreduce.task.io.sort.factor | 打开文件句柄的数量，默认10个。 |
| mapreduce.reduce.shuffle.parallelcopies | Reduce Task启动的并发拷贝数据的线程数目，默认5个。 |
| mapreduce.application.classpath | 默认：  For Linux:  $HADOOP\_MAPRED\_HOME/share/hadoop/mapreduce/\*,  $HADOOP\_MAPRED\_HOME/share/hadoop/mapreduce/lib/\*.  For Windows:  %HADOOP\_MAPRED\_HOME%/share/hadoop/mapreduce/\*,  %HADOOP\_MAPRED\_HOME%/share/hadoop/mapreduce/lib/\*  但$HADOOP\_MAPRED\_HOME失效，要改成$HADOOP\_HOME |

* 配置yarn-site.xml

<configuration>

<property>

<name>yarn.resourcemanager.recovery.enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.nodemanager.recovery.enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.acl.enable</name>

<value>false</value>

</property>

<property>

<name>yarn.admin.acl</name>

<value>\*</value>

</property>

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<name>yarn.log-aggregation.retain-seconds</name>

<value>172800</value>

</property>

<property>

<name>yarn.nodemanager.log.retain-seconds</name>

<value>14400</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>namenode:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>namenode:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>namenode:8031</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address</name>

<value>namenode:8033</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address</name>

<value>namenode:8088</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.https.address</name>

<value>namenode:8090</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>

</property>

<property>

<name>yarn.resourcemanager.client.thread-count</name>

<value>100</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.client.thread-count</name>

<value>100</value>

</property>

<property>

<name>yarn.resourcemanager.admin.client.thread-count</name>

<value>10</value>

</property>

<property>

<name>yarn.nodemanager.delete.thread-count</name>

<value>10</value>

</property>

<property>

<name>yarn.scheduler.minimum-allocation-mb</name>

<value>512</value>

</property>

<property>

<name>yarn.scheduler.maximum-allocation-mb</name>

<value>1800</value>

</property>

<property>

<name>yarn.scheduler.minimum-allocation-vcores</name>

<value>1</value>

</property>

<property>

<name>yarn.scheduler.maximum-allocation-vcores</name>

<value>8</value>

</property>

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>1800</value>

</property>

<property>

<name>yarn.nodemanager.resource.cpu-vcores</name>

<value>8</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

</configuration>

|  |  |
| --- | --- |
| **配置项** | **描述** |
| yarn.acl.enable | 通过设置配置属性yarn.acl.enable为true来启动应用程序的ACL。查看访问的ACL决定谁可以通过RPC接口查看一些所有应用程序的相关细节。默认是false |
| yarn.admin.acl | 谁能够执行ACL操作，集群中的管理员，用逗号隔开，默认是\*，表示谁都可以 |
| yarn.nodemanager.delete.thread-count | 用于清理临时结果的线程数。 |
| yarn.log-aggregation-enable | 是否开启日志聚类运算。日志聚合开启后在任务完成后会保存到HDFS上。 |
| yarn.nodemanager.remote-app-log-dir | 当应用程序运行结束后，日志被转移到的HDFS目录（启用日志聚集功能时有效），修改为保存的日志文件夹。  默认值：/tmp/logs |
| yarn.nodemanager.remote-app-log-dir-suffix | 远程日志目录子目录名称（启用日志聚集功能时有效）。  默认值：logs 日志将被转移到目录${yarn.nodemanager.remote-app-log-dir}/${user}/${thisParam}下 |
| yarn.log-aggregation.retain-seconds | 聚合后的日志在HDFS上保存多长时间，单位为s。  默认值：-1（不启用日志聚合），例如设置为86400，24小时 |
| yarn.log-aggregation.retain-check-interval-seconds | 删除任务在HDFS上执行的间隔，执行时候将满足条件的日志删除（超过yarn.log-aggregation.retain-seconds参数设置的时间的日志），如果是0或者负数，则为yarn.log-aggregation.retain-seconds参数设置值的1/10。 |
| yarn.nodemanager.log.retain-seconds | NodeManager上日志最多存放时间（不启用日志聚集功能时有效）。默认值：10800（3小时） |
| yarn.resourcemanager.client.thread-count | 客户端线程计数，用于处理应用程序管理请求的线程数量。 |
| yarn.resourcemanager.scheduler.client.thread-count | 调度程序线程计数，用于通过调度器接口处理请求的线程数量。 |
| yarn.resourcemanager.admin.client.thread-count | 管理客户端线程计数，用于处理 ResourceManager 管理接口的线程数量。 |
| yarn.resourcemanager.recovery.enabled | 是否开启ResourceManager自启，true启动，false关闭，默认false。 |
| yarn.resourcemanager.hostname |  |
| yarn.resourcemanager.address | ResourceManager 对客户端暴露的地址。客户端通过该地址向RM提交应用程序，杀死应用程序等。  默认值：${yarn.resourcemanager.hostname}:8032 |
| yarn.resourcemanager.scheduler.address | ResourceManager 对ApplicationMaster暴露的访问地址。ApplicationMaster通过该地址向RM申请资源、释放资源等。默认值：${yarn.resourcemanager.hostname}:8030 |
| yarn.resourcemanager.resource-tracker.address | ResourceManager 对NodeManager暴露的地址.。NodeManager通过该地址向RM汇报心跳，领取任务等。  默认值：${yarn.resourcemanager.hostname}:8031 |
| yarn.resourcemanager.admin.address | ResourceManager 对管理员暴露的访问地址。管理员通过该地址向RM发送管理命令等。  默认值：${yarn.resourcemanager.hostname}:8033 |
| yarn.resourcemanager.webapp.address | ResourceManager对外web ui地址。用户可通过该地址在浏览器中查看集群各类信息。  默认值：${yarn.resourcemanager.hostname}:8088 |
| yarn.resourcemanager.webapp.https.address | web https 地址，认值：${yarn.resourcemanager.hostname}:8090 |
| yarn.resourcemanager.scheduler.class | 启用的资源调度器主类。目前可用的有FIFO、Capacity Scheduler和Fair Scheduler。  默认值：  org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler  在Yarn中有三种调度器可以选择：FIFOScheduler ，CapacityScheduler，FairS cheduler。   1. FIFOScheduler把应用按提交的顺序排成一个队列，这是一个先进先出队列，在进行资源分配的时候，先给队列中最头上的应用进行分配资源，待最头上的应用需求满足后再给下一个分配，以此类推。FIFO Scheduler是最简单也是最容易理解的调度器，也不需要任何配置，但它并不适用于共享集群。大的应用可能会占用所有集群资源，这就导致其它应用被阻塞。在共享集群中，更适合采用CapacityScheduler或FairScheduler，这两个调度器都允许大任务和小任务在提交的同时获得一定的系统资源。 2. 对于Capacity调度器，有一个专门的队列用来运行小任务，但是为小任务专门设置一个队列会预先占用一定的集群资源，这就导致大任务的执行时间会落后于使用FIFO调度器时的时间。 3. 在Fair调度器中，我们不需要预先占用一定的系统资源，Fair调度器会为所有运行的job动态的调整系统资源。 |
| yarn.scheduler.minimum-allocation-mb  yarn.scheduler.maximum-allocation-mb | 单个可申请的最小/最大内存资源量。比如设置为1024和3072，则运行MapRedce作业时，每个Task最少可申请1024MB内存，最多可申请3072MB内存。单位是MB。默认是1024和8192。 |
| yarn.scheduler.minimum-allocation-vcores  yarn.scheduler.maximum-allocation-vcores | 单个可申请的最小/最大虚拟CPU个数。比如设置为1和4，则运行MapRedce作业时，每个Task最少可申请1个虚拟CPU，最多可申请4个虚拟CPU。默认是1和4 |
| yarn.resourcemanager.nodes.include-path /  yarn.resourcemanager.nodes.exclude-path |  |
| yarn.nodemanager.resource.memory-mb | NodeManager总的可用物理内存。注意，该参数是不可修改的，一旦设置，整个运行过程中不可动态修改。另外，该参数的默认值是8192MB，即使你的机器内存不够8192MB，YARN也会按照这些内存来使用 |
| yarn.nodemanager.vmem-pmem-ratio | 每使用1MB物理内存，最多可用的虚拟内存数。默认值：2.1，yarn.nodemanager.resource.memory-mb\*2.1 |
| yarn.nodemanager.resource.cpu-vcores | NodeManager总的可用虚拟CPU个数。 |
| yarn.nodemanager.recovery.enabled | 是否开启NodeManager自启，true启动，false关闭，默认false。 |
| yarn.nodemanager.local-dirs | 中间结果存放位置，类似于1.0中的mapred.local.dir。注意，这个参数通常会配置多个目录，已分摊磁盘IO负载。  默认值：${hadoop.tmp.dir}/nm-local-dir |
| yarn.nodemanager.log-dirs | 日志存放地址（可配置多个目录）。  默认值：${yarn.log.dir}/userlogs |
| yarn.nodemanager.aux-services | NodeManager上运行的附属服务。需配置成mapreduce\_shuffle，才可运行MapReduce程序，默认值”” |
| yarn.application.classpath | For Linux:  $HADOOP\_CONF\_DIR, $HADOOP\_COMMON\_HOME/share/hadoop/common/\*,  $HADOOP\_COMMON\_HOME/share/hadoop/common/lib/\*,  $HADOOP\_HDFS\_HOME/share/hadoop/hdfs/\*,  $HADOOP\_HDFS\_HOME/share/hadoop/hdfs/lib/\*,  $HADOOP\_YARN\_HOME/share/hadoop/yarn/\*,  $HADOOP\_YARN\_HOME/share/hadoop/yarn/lib/\*  For Windows:  %HADOOP\_CONF\_DIR%,  %HADOOP\_COMMON\_HOME%/share/hadoop/common/\*,  %HADOOP\_COMMON\_HOME%/share/hadoop/common/lib/\*,  %HADOOP\_HDFS\_HOME%/share/hadoop/hdfs/\*,  %HADOOP\_HDFS\_HOME%/share/hadoop/hdfs/lib/\*,  %HADOOP\_YARN\_HOME%/share/hadoop/yarn/\*,  %HADOOP\_YARN\_HOME%/share/hadoop/yarn/lib/\* |

* 将配置好的hadoop-2.7.4-distributed复制到datanode1和datanode2节点上；

scp -r hadoop-2.7.4-distributed root@datanode1:/opt/

scp -r hadoop-2.7.4-distributed root@datanode2:/opt/

改变权限：chown -R hadoop:hadoop /opt/hadoop-2.7.4-distributed

chmod 775 /opt/hadoop-2.7.4-distributed/

* 启动与测试（在主节点上操作）

第一步：格式化HDFS文件系统

$HADOOP\_HOME/bin/hdfs namenode -format（仅第一次运行执行）

第二步：启动集群：$HADOOP\_HOME/sbin/start-all.sh

或者

$HADOOP\_HOME/sbin/start-dfs.sh

$HADOOP\_HOME/sbin/start-yarn.sh

第三步：访问

hdfs访问：<http://master-ip:50070>

yarn访问：http://master-ip:8088

第四步：使用jps查看进程。

* 停止集群（在主节点上操作）

$HADOOP\_HOME/sbin/stop-all.sh

或者

$HADOOP\_HOME/sbin/stop-yarn.sh

$HADOOP\_HOME/sbin/stop-dfs.sh

* 示例（统计单词，这是基于hdfs）

新建一个文件夹存放文件：

$HADOOP\_HOME/bin/hdfs dfs -mkdir -p /wordcountdemo/input

上传文件到hdfs：

$HADOOP\_HOME/bin/hdfs dfs -put /opt/wc.input /wordcountdemo/input

统计单词数：

$HADOOP\_HOME/bin/yarn jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.4.jar wordcount /wordcountdemo/input/ /wordcountdemo/output

* 历史job

$HADOOP\_HOME/bin/mapred --daemon start historyserver

<http://192.168.1.119:19888>

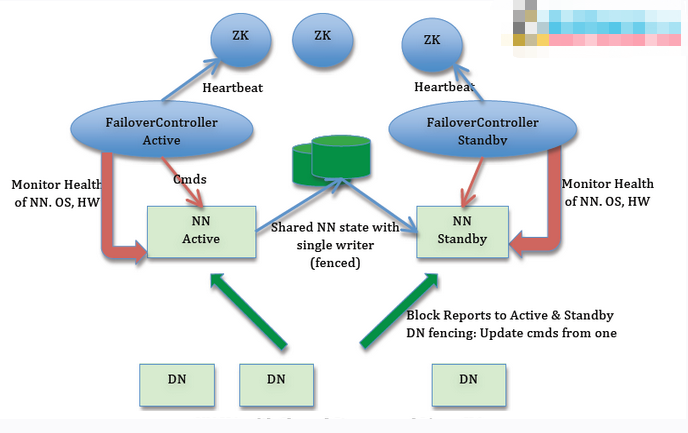
### hadoop HA（QJM）分布式集群安装

HA的意思是High Availability高可用，指当当前工作中的机器宕机后，会自动处理这个异常，并将工作无缝地转移到其他备用机器上去，以来保证服务的高可用。

HA方式安装部署才是最常见的生产环境上的安装部署方式。Hadoop HA是Hadoop 2.x中新添加的特性，包括NameNode HA 和 ResourceManager HA。

Hadoop HA 即Hadoop的高可用，不同于普通的namenode+second namenode的模式，second namenode只能作为namenode的冷备份，当namenode挂掉后，second namenode不能自动充当namenode的角色（存在单点故障）。所以为了保证高可用性，就有了Hadoop HA的机制。

#### HDFS HA架构



基于QJM的HDFS HA方案如上图所示，其处理流程为：集群启动后一个NameNode处于Active状态，并提供服务，处理客户端和DataNode的请求，并把editlog写到本地和share editlog（这里是QJM）中。另外一个NameNode处于Standby状态，它启动的时候加载fsimage，然后周期性的从share editlog中获取editlog，保持与Active节点的状态同步。为了实现Standby在Active挂掉后迅速提供服务，需要DataNode同时向两个NameNode汇报，使得Stadnby保存block to DataNode信息，因为NameNode启动中最费时的工作是处理所有DataNode的blockreport。为了实现热备，增加FailoverController和Zookeeper，FailoverController与Zookeeper通信，通过Zookeeper选举机制，FailoverController通过RPC让NameNode转换为Active或Standby。

hdfs HA中即有至少有两个namenode，这儿以两个namenode为例，一个可以对外提供服务的namenode成为active namenode，待命状态的namenode成为standby namenode。

HDFS的自动故障转移主要由Zookeeper和ZKFC两个组件组成。

1. Zookeeper集群作用主要有：一是故障监控。每个NameNode将会和Zookeeper建立一个持久session，如果NameNode失效，那么此session将会过期失效，此后Zookeeper将会通知另一个Namenode，然后触发Failover；二是NameNode选举。ZooKeeper提供了简单的机制来实现Acitve Node选举，如果当前Active失效，Standby将会获取一个特定的排他锁，那么获取锁的Node接下来将会成为Active。
2. ZKFC是一个Zookeeper的客户端，它主要用来监测和管理NameNodes的状态，每个NameNode机器上都会运行一个ZKFC程序，它的职责主要有：一是健康监控。ZKFC间歇性的ping NameNode，得到NameNode返回状态，如果NameNode失效或者不健康，那么ZKFS将会标记其为不健康；二是Zookeeper会话管理。当本地NaneNode运行良好时，ZKFC将会持有一个Zookeeper session，如果本地NameNode为Active，它同时也持有一个“排他锁”znode，如果session过期，那么次lock所对应的znode也将被删除；三是选举。当集群中其中一个NameNode宕机，Zookeeper会自动将另一个激活。

#### YARN HA架构

YARN的HA架构和HDFS的HA类似，需要启动两个ResourceManager，这两个ResourceManager会向ZooKeeper集群注册，通过ZooKeeper管理它们的状态（Active或Standby）并进行自动故障转移。

#### HADOOP HA（QJM）集群安装

1. 服务器角色规划

| **namenode** | **datanode1** | **datanode2** |
| --- | --- | --- |
| NameNode | NameNode |  |
|  | ResourceManage | ResourceManage |
| Zookeeper | Zookeeper | Zookeeper |
| DataNode | DataNode | DataNode |
| NodeManager | NodeManager | NodeManager |

1. 登录官网下载并安装zookeeper-3.4.10.tar.gz

* 解压：tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/
* 添加环境变量（每个节点都必须执行）：

export ZOOKEEPER\_HOME=/opt/zookeeper-3.4.10

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

* 添加配置：

mv zoo\_sample.cfg zoo.cfg生成配置文件，配置如下：

tickTime：客户端心跳时间(毫秒)

initLimit：允许心跳间隔的最大时间

syncLimit：同步时限

dataDir：数据存储目录

dataLogDir：数据日志存储目录

maxClientCnxns：服务器最大连接数，默认为10，改为0表示无限制

autopurge.snapRetainCount：快照数

autopurge.purgeInterval：快照清理时间，默认为0

# The number of milliseconds of each tick

tickTime=2000

# The number of ticks that the initial

# synchronization phase can take

initLimit=10

# The number of ticks that can pass between

# sending a request and getting an acknowledgement

syncLimit=5

# the directory where the snapshot is stored.

# do not use /tmp for storage, /tmp here is just

# example sakes.

dataDir=/home/bigdata/zookeeper/data

dataLogDir=/home/bigdata/zookeeper/logs

# the port at which the clients will connect

clientPort=2181

maxClientCnxns=0

autopurge.snapRetainCount=3

autopurge.purgeInterval=1

server.1=namenode:2888:3888

server.2=datanode1:2888:3888

server.3=datanode2:2888:3888

* 创建myid文件（每个节点都必须生成myid，数字由server决定）

在dataDir所指定的目录（/home/bigdata/zookeeper/data）下新建一个名为myid的文件，文件内容为server点后面的数字。

namenode节点：echo 1 >> myid

datanode1节点：echo 2 >> myid

datanode2节点：echo 3 >> myid

* $ZOOKEEPER\_HOME/conf/log4j.properties设定日志输出级别

添加一行：

zookeeper.root.logger=INFO,ROLLINGFILE

* 将配置好的zookeeper复制到其他机器上

scp -r zookeeper-3.4.10/ root@datanode1:/opt/

scp -r zookeeper-3.4.10/ root@datanode2:/opt/

改变权限：

chown -R hadoop:hadoop /opt/zookeeper-3.4.10/

chmod 775 /opt/zookeeper-3.4.10/

* 单节点启动zookeeper（每个节点都必须执行）

$ZOOKEEPER\_HOME/bin/zkServer.sh start

* 单节点停止zookeeper（每个节点都必须执行）

$ZOOKEEPER\_HOME/bin/zkServer.sh stop

* 查看单个节点状态

$ZOOKEEPER\_HOME/bin/zkServer.sh status

leader：主节点

follower：从节点

* 创建脚本启动与停止所有节点

停止zookeeper集群脚本：

/opt/script/zookeeper/stopAll.sh

#!/bin/bash

ssh namenode $ZOOKEEPER\_HOME/bin/zkServer.sh stop

ssh datanode1 $ZOOKEEPER\_HOME/bin/zkServer.sh stop

ssh datanode2 $ZOOKEEPER\_HOME/bin/zkServer.sh stop

启动zookeeper集群脚本：

/opt/script/zookeeper/startAll.sh

#!/bin/bash

ssh namenode $ZOOKEEPER\_HOME/bin/zkServer.sh start

ssh datanode1 $ZOOKEEPER\_HOME/bin/zkServer.sh start

ssh datanode2 $ZOOKEEPER\_HOME/bin/zkServer.sh start

重启zookeeper集群脚本：

/opt/script/zookeeper/restartAll.sh

#!/bin/bash

ssh namenode $ZOOKEEPER\_HOME/bin/zkServer.sh restart

ssh datanode1 $ZOOKEEPER\_HOME/bin/zkServer.sh restart

ssh datanode2 $ZOOKEEPER\_HOME/bin/zkServer.sh restart

1. 登录hadoop官网http://hadoop.apache.org/releases.html下载hadoop-2.8.4.tar.gz，解压到/home/hadoop目录下：tar -zxvf hadoop-2.8.4.tar.gz -C /opt/，重命名为hadoop-2.8.4-ha

注：其他机器在配置好后，再使用scp拷贝相应的目录即可。

1. 配置环境变量（vi ~/.bashrc），并执行source ~/.bashrc使其生效（也可以添加到/etc/profile文件中）

export HADOOP\_HOME=/opt/hadoop-2.8.4-ha

export HADOOP\_COMMON\_HOME=$HADOOP\_HOME

export HADOOP\_HDFS\_HOME=$HADOOP\_HOME

export HADOOP\_YARN\_HOME=$HADOOP\_HOME

export HADOOP\_MAPRED\_HOME=$HADOOP\_HOME

export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

export HDFS\_CONF\_DIR=$HADOOP\_HDFS\_HOME/etc/hadoop

export HADOOP\_LOG\_DIR=/home/bigdata/hadoop/logs

export YARN\_LOG\_DIR=/home/bigdata/yarn/logs

export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin

注意创建目录：/home/bigdata/hadoop/logs

1. 在/home/hadoop/hadoop-2.8.4-ha/etc/hadoop/hadoop-env.sh、mapred-env.sh、yarn-env.sh中，再显示地重新声明一遍JAVA\_HOME，即写明具体的存放目录。

export JAVA\_HOME=/usr/local/jdk1.8.0\_171

1. 配置hadoop

进入Hadoop配置目录：/home/hadoop/hadoop-2.8.4-ha/etc/hadoop，对slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml等 5个文件进行配置。

* salves文件

将作为 DataNode 的主机名写入该文件，每行一个数据节点。本案例中有两个数据节点datanode1与datanode2，在文件中加入以下两行。

namenode

datanode1

datanode2

* 配置hdfs-site.xml

<configuration>

<!-- 指定hdfs元数据存储的路径 -->

<property>

<name>dfs.namenode.name.dir</name>

<value>/home/bigdata/hadoop/namenode</value>

</property>

<!-- 指定hdfs数据存储的路径 -->

<property>

<name>dfs.datanode.data.dir</name>

<value>/home/bigdata/hadoop/datanode</value>

</property>

<!-- 数据备份的个数 -->

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<!-- 关闭权限验证 -->

<property>

<name>dfs.permissions.enabled</name>

<value>false</value>

</property>

<!-- 开启WebHDFS功能（基于REST的接口服务） -->

<property>

<name>dfs.webhdfs.enabled</name>

<value>true</value>

</property>

<property>

<!-- 为namenode集群定义一个services name -->

<name>dfs.nameservices</name>

<value>ns1</value>

</property>

<property>

<!-- nameservice 包含哪些namenode，为各个namenode起名 -->

<name>dfs.ha.namenodes.ns1</name>

<value>nn1,nn2</value>

</property>

<property>

<!-- 名为nn1的namenode 的rpc地址和端口号，rpc用来和datanode通讯 -->

<name>dfs.namenode.rpc-address.ns1.nn1</name>

<value>namenode:8020</value>

</property>

<property>

<!-- 名为nn2的namenode 的rpc地址和端口号，rpc用来和datanode通讯 -->

<name>dfs.namenode.rpc-address.ns1.nn2</name>

<value>datanode1:8020</value>

</property>

<property>

<!--名为nn1的namenode 的http地址和端口号，web客户端 -->

<name>dfs.namenode.http-address.ns1.nn1</name>

<value>namenode:50070</value>

</property>

<property>

<!--名为nn2的namenode 的http地址和端口号，web客户端 -->

<name>dfs.namenode.http-address.ns1.nn2</name>

<value>datanode1:50070</value>

</property>

<property>

<!-- namenode间用于共享编辑日志的journal节点列表 指定NameNode的元数据在JournalNode上的存放位置 -->

<name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://namenode:8485;datanode1:8485;datanode2:8485/ns1</value>

</property>

<property>

<!-- journalnode 上用于存放edits日志的目录 指定JournalNode在本地磁盘存放数据的位置 -->

<name>dfs.journalnode.edits.dir</name>

<value>/home/bigdata/journalnode</value>

</property>

<property>

<!-- 客户端连接可用状态的NameNode所用的代理类 -->

<name>dfs.client.failover.proxy.provider.ns1</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<property>

<!-- 配置隔离机制 -->

<name>dfs.ha.fencing.methods</name>

<value>sshfence</value>

</property>

<property>

<!-- 使用隔离机制时需要ssh免密码登陆 -->

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/home/hadoop/.ssh/id\_rsa</value>

</property>

<property>

<!-- 开启故障自动转移 -->

<name>dfs.ha.automatic-failover.enabled</name>

<value>true</value>

</property>

</configuration>

注意创建目录：

/home/bigdata/hadoop/namenode

/home/bigdata/hadoop/datanode

/home/bigdata/journalnode

* 配置core-site.xml

<configuration>

<!-- 指定hdfs的nameservices名称为ns1，与hdfs-site.xml的HA配置相同 -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://ns1</value>

</property>

<!-- 指定缓存文件存储的路径 -->

<property>

<name>hadoop.tmp.dir</name>

<value>/home/bigdata/hadoop/tmp</value>

</property>

<!-- 指定zookeeper地址，配置HA时需要 -->

<property>

<name>ha.zookeeper.quorum</name>

<value>namenode:2181,datanode1:2181,datanode2:2181</value>

</property>

</configuration>

注意创建目录：/home/hadoop/bigdata/hadoop/tmp

* 配置mapred-site.xml

mapreduce计算模型配置。

<configuration>

<!-- 指定MapReduce计算框架使用YARN -->

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

<!-- 指定jobhistory server的rpc地址 -->

<property>

<name>mapreduce.jobhistory.address</name>

<value>datanode1:10020</value>

</property>

<!-- 指定jobhistory server的http地址 -->

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>datanode1:19888</value>

</property>

<!-- 提交任务时使用的临时目录 -->

<property>

<name>yarn.app.mapreduce.am.staging-dir</name>

<value>/yarn/tmp/hadoop-yarn/staging</value>

</property>

<!-- 开启uber模式（针对小作业的优化） -->

<property>

<name>mapreduce.job.ubertask.enable</name>

<value>true</value>

</property>

<!-- 配置启动uber模式的最大map数 -->

<property>

<name>mapreduce.job.ubertask.maxmaps</name>

<value>9</value>

</property>

<!-- 配置启动uber模式的最大reduce数 -->

<property>

<name>mapreduce.job.ubertask.maxreduces</name>

<value>1</value>

</property>

<property>

<name>mapreduce.map.memory.mb</name>

<value>512</value>

</property>

<property>

<name>mapreduce.map.java.opts</name>

<value>-Xmx1024M</value>

</property>

<property>

<name>mapreduce.reduce.memory.mb</name>

<value>512</value>

</property>

<property>

<name>mapreduce.reduce.java.opts</name>

<value>-Xmx1024M</value>

</property>

<property>

<name>mapreduce.task.io.sort.mb</name>

<value>150</value>

</property>

<property>

<name>mapreduce.task.io.sort.factor</name>

<value>100</value>

</property>

<property>

<name>mapreduce.reduce.shuffle.parallelcopies</name>

<value>50</value>

</property>

<property>

<name>mapreduce.application.classpath</name>

<value>

$HADOOP\_HOME/share/hadoop/mapreduce/\*,

$HADOOP\_HOME/share/hadoop/mapreduce/lib/\*

</value>

</property>

</configuration>

* 配置yarn-site.xml

<configuration>

<!-- NodeManager上运行的附属服务，需配置成mapreduce\_shuffle才可运行MapReduce程序 -->

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<!-- 配置Web Application Proxy安全代理（防止yarn被攻击） -->

<property>

<name>yarn.web-proxy.address</name>

<value>datanode2:8888</value>

</property>

<!-- 开启日志 -->

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<!-- 配置日志删除时间为7天，-1为禁用，单位为秒 -->

<property>

<name>yarn.log-aggregation.retain-seconds</name>

<value>604800</value>

</property>

<!-- 修改日志目录 此目录在hdfs上 -->

<property>

<name>yarn.nodemanager.remote-app-log-dir</name>

<value> /yarn/logs</value>

</property>

<!-- 配置nodemanager可用的资源内存 -->

<property>

<name>yarn.nodemanager.resource.memory-mb</name>

<value>1800</value>

</property>

<!-- 开启yarn resourcemanager restart -->

<property>

<name>yarn.resourcemanager.recovery.enabled</name>

<value>true</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>

</property>

<property>

<name>yarn.scheduler.minimum-allocation-mb</name>

<value>512</value>

</property>

<property>

<name>yarn.scheduler.maximum-allocation-mb</name>

<value>1800</value>

</property>

<property>

<name>yarn.scheduler.minimum-allocation-vcores</name>

<value>1</value>

</property>

<property>

<name>yarn.scheduler.maximum-allocation-vcores</name>

<value>8</value>

</property>

<!-- 开启YARN HA -->

<property>

<name>yarn.resourcemanager.ha.enabled</name>

<value>true</value>

</property>

<!-- 启用自动故障转移 -->

<property>

<name>yarn.resourcemanager.ha.automatic-failover.enabled</name>

<value>true</value>

</property>

<!-- 当启用自动failover后，使用内置的leader选举来选主RM。 -->

<property>

<name>yarn.resourcemanager.ha.automatic-failover.embedded</name>

<value>true</value>

</property>

<!-- 指定YARN HA的名称 -->

<property>

<name>yarn.resourcemanager.cluster-id</name>

<value>yarncluster</value>

</property>

<!-- 指定两个resourcemanager的名称 -->

<property>

<name>yarn.resourcemanager.ha.rm-ids</name>

<value>resourcemanager1,resourcemanager2</value>

</property>

<!-- 配置resourcemanager1，resourcemanager2的主机 -->

<property>

<name>yarn.resourcemanager.hostname.resourcemanager1</name>

<value>datanode1</value>

</property>

<property>

<name>yarn.resourcemanager.hostname.resourcemanager2</name>

<value>datanode2</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address.resourcemanager1</name>

<value>datanode1:8030</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address.resourcemanager2</name>

<value>datanode2:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address.resourcemanager1</name>

<value>datanode1:8031</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address.resourcemanager2</name>

<value>datanode2:8031</value>

</property>

<property>

<name>yarn.resourcemanager.address.resourcemanager1</name>

<value>datanode1:8032</value>

</property>

<property>

<name>yarn.resourcemanager.address.resourcemanager2</name>

<value>datanode2:8032</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address.resourcemanager1</name>

<value>datanode1:8033</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address.resourcemanager2</name>

<value>datanode2:8033</value>

</property>

<!-- 配置YARN的http端口 -->

<property>

<name>yarn.resourcemanager.webapp.address.resourcemanager1</name>

<value>datanode1:8088</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address.resourcemanager2</name>

<value>datanode2:8088</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.https.address.resourcemanager1</name>

<value>datanode1:8090</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.https.address.resourcemanager2</name>

<value>datanode2:8090</value>

</property>

<!-- 配置zookeeper的地址 -->

<property>

<name>yarn.resourcemanager.zk-address</name>

<value>namenode:2181,datanode1:2181,datanode2:2181</value>

</property>

<!-- 配置zookeeper的存储位置 hdfs上 -->

<property>

<name>yarn.resourcemanager.zk-state-store.parent-path</name>

<value>/yarn/rmstore</value>

</property>

<!-- 配置resourcemanager的状态存储到zookeeper中 -->

<property>

<name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>

</configuration>

* 集群初始化

**第一步：**启动zookeeper集群（详见zookeeper安装启动部分）

**第二步：**格式化ZKFC（在一个namenode节点上执行，即nn1）

$HADOOP\_HOME/bin/hdfs zkfc -formatZK

注：出现Bad argument: –formatZK错误时，手动输入，不要复制命令。

**第三步：**启动Journalnode（配置文件中指定的journal节点

都要执行启动，仅初始化需要手动启动）

$HADOOP\_HOME/bin/hdfs --daemon start journalnode（3.1.0版）

$HADOOP\_HOME/sbin/hadoop-daemon.sh start journalnode（2.8.4版）

启动所有Journalnode脚本：

/opt/script/journalnode/startAll.sh

#!/bin/bash

ssh namenode $HADOOP\_HOME/sbin/hadoop-daemon.sh start journalnode

ssh datanode1 $HADOOP\_HOME/sbin/hadoop-daemon.sh start journalnode

ssh datanode2 $HADOOP\_HOME/sbin/hadoop-daemon.sh start journalnode

停止所有Journalnode脚本：

/opt/script/journalnode/stopAll.sh

#!/bin/bash

ssh namenode $HADOOP\_HOME/sbin/hadoop-daemon.sh stop journalnode

ssh datanode1 $HADOOP\_HOME/sbin/hadoop-daemon.sh stop journalnode

ssh datanode2 $HADOOP\_HOME/sbin/hadoop-daemon.sh stop journalnode

**第四步：**格式化HDFS文件系统

在一个namenode节点（即nn1）上执行格式化：

$HADOOP\_HOME/bin/hdfs namenode -format（仅第一次运行执行）

然后将格式化后nn1节点namenode目录的元数据复制到nn2上。

scp -r /home/bigdata/hadoop/namenode/ hadoop@datanode1:/home/bigdata/hadoop/

**第五步**：初始化完毕，停止journalnode、zookeeper。

/opt/script/journalnode/stopAll.sh

/opt/script/zookeeper/stopAll.sh

* 启动与测试

**第一步**：启动zookeeper集群（详见zookeeper安装启动部分）

/opt/script/zookeeper/startAll.sh

**第二步：**启动HDFS

$HADOOP\_HOME/sbin/start-dfs.sh

**第三步**：启动YARN

$HADOOP\_HOME/sbin/start-yarn.sh

$HADOOP\_HOME/sbin/yarn-daemon.sh start resourcemanager（2.8.4需要单独启动每台的resourcemanager）

**第四步**：访问

hdfs访问：<http://nn1-ip:50070>

<http://nn2-ip:50070>

yarn访问：<http://rm1-ip:8088>

<http://rm2-ip:8088>

**第五步**：使用jps查看进程。

* 停止集群（在主节点上操作）

**第一步：**$HADOOP\_HOME/sbin/stop-yarn.sh

**第二步：**$HADOOP\_HOME/sbin/stop-dfs.sh

**第三步：**/opt/script/zookeeper/stopAll.sh

* 示例（统计单词，这是基于hdfs）

新建一个文件夹存放文件：

$HADOOP\_HOME/bin/hdfs dfs -mkdir -p /wordcountdemo/input

上传文件到hdfs：

$HADOOP\_HOME/bin/hdfs dfs -put /opt/wc.input /wordcountdemo/input

统计单词数：

$HADOOP\_HOME/bin/yarn jar $HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.4.jar wordcount /wordcountdemo/input/ /wordcountdemo/output

* 历史job

$HADOOP\_HOME/bin/mapred --daemon start historyserver（在datanode1上执行）

<http://192.168.1.120:19888>

### hadoop HA（NFS）分布式集群安装

# Hadoop分布式文件系统（HDFS）

## HDFS命令操作

### 格式化配置HDFS文件系统

打开NameNode(HDFS服务器)，然后执行以下命令：

$HADOOP\_HOME/bin/hadoop namenode -format

### HDFS的文件列表

使用'ls' 可以找出文件列表中的目录，文件状态。可以传递一个目录或文件名作为参数的语法。

$HADOOP\_HOME/bin/hadoop fs -ls <args>

ls命令后面还可以加其他的参数，如[-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]。下面加-R递归列出所有文件及目录。

格式：hadoop fs -ls -R <path>

### 查看一个文件或目录大小

格式：hadoop fs -du <path>（以字节为单位）

du命令后面还可以加其他参数，指定显示单位等。如[-du [-s] [-h] [-x] <path> ...]。如以KB为单位显示：hadoop fs -du -h <path>

### 创建一个目录

格式：hadoop fs -mkdir <args>，如hadoop fs -mkdir /checkpoint。

注意：多级目录，必须确保上一次的目录已创建。

### 创建一个空文件

格式：hadoop fs -touchz URI [URI ...]

### 将输入插入到HDFS

传输并使用本地系统put命令，Hadoop文件系统中存储的数据文件。格式：hadoop fs -put <文件参数> <目录参数>

如：hadoop fs -put /home/file.txt /user/input

[-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]

### 将本地文件追加到HDFS上的一个文件

使用“appendToFile”将一个或者多个本地文件追加到目的文件。格式：hadoop fs -appendToFile <一个或者多个本地文件> <HDFS文件>

如：hadoop fs -appendToFile b.txt /hadoop/txt/a.txt

hadoop fs -appendToFile c.txt d.txt /hadoop/txt/a.txt

hadoop fs -appendToFile c.txt d.txt hdfs://192.168.0.119:9000/hadoop/txt/a.txt

### 查看文件里的数据信息

使用cat命令来查看来自HDFS的数据。格式：hadoop fs -cat <文件>，如下所示：

hadoop fs -cat /user/input/file.txt

### 计算paths下的目录数，文件数和字节数

格式：hadoop fs -count [-q] [-h] [-v] <paths>

### 改变文件所属的组

格式：hadoop fs -chgrp [-R] GROUPNAME URI [URI ...]，使用-R 将使改变在目录结构下递归进行。

hadoop fs -chgrp supergroup hdfs://192.168.0.119:9000/hadoop/txt/a.txt

### 改变文件的所有者

格式：hadoop fs -chown [-R] [OWNER][:[GROUP]] URI [URI ]，使用-R 将使改变在目录结构下递归进行。命令的使用者必须是超级用户。

hadoop fs -chown hadoop:hadoop hdfs://192.168.0.119:9000/hadoop/txt/a.txt

### 改变文件访问权限

格式：hadoop fs -chmod [-R] <MODE[,MODE]... | OCTALMODE> URI [URI ...]，使用-R 将使改变在目录结构下递归进行。权限设置可以参考 linux下文件系统的chmod的用法，基本类似。

### 查找满足表达式的文件和文件夹

格式：hadoop fs -find <path> ... <expression>，没有配置path的话，默认的就是全部目录/；如果表达式没有配置，则默认为-print。

打印：

hadoop fs -find /hadoop/txt/

hadoop fs -find /hadoop/txt/ -print

hadoop fs -find /hadoop/txt/ -print0 打印在一行上

### 从HDFS取回文件到本地文件系统

格式：hadoop fs -get <文件或目录> <本地文件系统目录>，

如下所示：

hadoop fs -get /user/input/ /home/hadoop\_tp/(将input目录取回本地文件系统，并存放在/home/hadoop\_tp/下)

[-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]

### getfacl

[-getfacl [-R] <path>]

### getfattr

[-getfattr [-R] {-n name | -d} [-e en] <path>]

### getmerge

格式：hadoop fs -getmerge <src> <localdst>，将源文件地址的内容合并到目标文件上，将覆盖目标文件的内容。

[-getmerge [-nl] [-skip-empty-file] <src> <localdst>]

### 移动文件或文件夹

格式：hadoop fs -mv <原文件或目录> <移动后的文件或目录>

### 复制文件或文件夹

格式：hadoop fs -cp <原文件或目录> <复制后的文件或目录>

[-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]

-f 参数选项：当文件存在时，进行覆盖。

-p 参数选项：将权限、所属组、时间戳、ACL以及XAttr等也进行拷贝。

### copyFromLocal

格式：hadoop fs -copyFromLocal <localsrc> URI，类似于put命令，和put不同的是，拷贝的源地址必须是本地文件地址。-f参数当拷贝的目标文件存在时，进行覆盖。

[-copyFromLocal [-f] [-p] [-l] [-d] <localsrc> ... <dst>]

### copyToLocal

格式：hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>，类似于get指令。和get不同的是，拷贝的目的地址必须是本地文件地址。

[-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]

### 删除文件或目录

格式：hadoop fs -rm <文件>

删除目录需要加-R参数，如hadoop fs -rm -R <文件夹>

### 递归删除文件或目录

格式：hadoop fs -rm -r <文件或目录>

### 清空回收站

用法：hadoop fs -expunge

### moveFromLocal

[-moveFromLocal <localsrc> ... <dst>]

### moveToLocal

[-moveToLocal <src> <localdst>]

### setfacl

hadoop fs -setfacl [-R] [-b |-k -m |-x <acl\_spec> <path>] |[--set <acl\_spec> <path>]

### 将HDFS中文件以文本形式输出（包括zip包，jar包等形式）

hadoop fs -text <src>

### 将文件按照length进行截取

hadoop fs -truncate [-w] <length> <paths>

### 返回命令的help信息

hadoop fs -usage command

### 帮助文档

hadoop fs -help

### 查看文件校验和

格式：hadoop fs -checksum URI

hadoop fs -checksum /hadoop/txt/a.txt

# Hbase

## Hbase简介

Apache HBase是一种Key/Value系统，它建立在HDFS之上的数据库。和Hive不一样，Hbase提供对数据的随机实时读/写访问（延时较低），而不是运行MapReduce任务，因此Hbase非常适合用来进行大数据的实时查询。它支持四种主要的操作：增加或者更新行，查看一个范围内的cell，获取指定的行，删除指定的行、列或者是列的版本。详看1.6.5

### Hbase组件

1. 主节点HMaster

HMaster没有单节点问题，HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master在运行，主要负责Table和Region的管理工作。

在*backup-masters配置文件可以添加备份主节点，*

1. Region节点HRegionServer

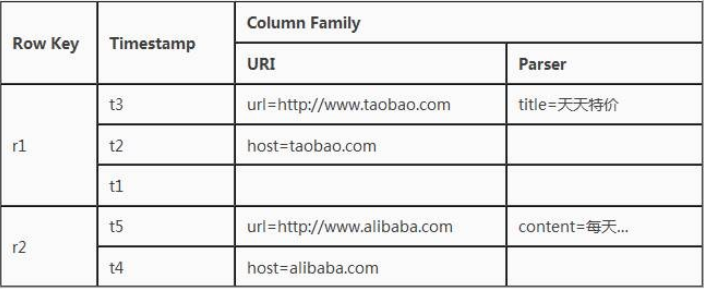
HRegionServer负责维护HRegion，处理HRegion的IO请求，向HDFS文件系统中读写数据；

### Hbase访问接口

1. Native Java API，最常规和高效的访问方式，适合Hadoop MapReduce Job并行批处理HBase表数据；
2. HBase Shell，HBase的命令行工具，最简单的接口，适合HBase管理使用；
3. Thrift Gateway，利用Thrift序列化技术，支持C++，PHP，Python等多种语言，适合其他异构系统在线访问HBase表数据；
4. REST Gateway，支持REST 风格的Http API访问HBase, 解除了语言限制；
5. Pig，可以使用Pig Latin流式编程语言来操作HBase中的数据，和Hive类似，本质最终也是编译成MapReduce Job来处理HBase表数据，适合做数据统计；
6. Hive，当前Hive的Release版本尚没有加入对HBase的支持，但在下一个版本Hive 0.7.0中将会支持HBase，可以使用类似SQL语言来访问Hbase；
7. Phoenix：phoenix是构建在Hbase之上的，使用标准的SQL操作Hbase，可以做联机事务处理，拥有低延迟的特性。phoenix会把SQL编译成一系列的Hbase的scan操作，然后把scan结果生成标准的JDBC结果集，其底层由于使用了Hbase的API，协处理器，过滤器，处理千万级行的数据也只用毫秒或秒级就搞定。

### Hbase数据模型

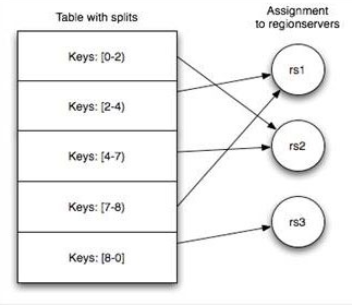
#### Table & Column Family



1. Row Key: 行键，Table的主键，Table中的记录按照Row Key排序；
2. Column Family：列簇，Table在水平方向有一个或者多个Column Family组成，一个Column Family中可以由任意多个Column组成，即Column Family支持动态扩展，无需预先定义Column的数量以及类型，所有Column均以二进制格式存储，用户需要自行进行类型转换。

#### Table & Region

当Table随着记录数不断增加而变大后，会逐渐分裂成多份splits，成为regions，一个region由[startkey,endkey)表示，不同的region会被Master分配给相应的RegionServer进行管理。



HBase中有两张特殊的Table，-ROOT-和.META.：

* .META.：记录了用户表的Region信息，.META.可以有多个regoin；
* -ROOT-：记录了.META.表的Region信息，-ROOT-只有一个region；
* Zookeeper中记录了-ROOT-表的location。

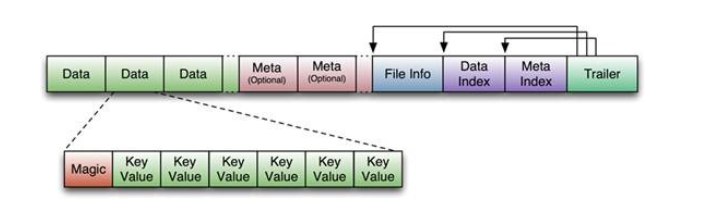
Client访问用户数据之前需要首先访问zookeeper，然后访问-ROOT-表，接着访问.META.表，最后才能找到用户数据的位置去访问，中间需要多次网络操作，不过client端会做cache缓存。

### HBase存储格式

HBase中的所有数据文件都存储在Hadoop HDFS文件系统上，主要包括上述提出的两种文件类型：

1. HFile，HBase中KeyValue数据的存储格式，HFile是Hadoop的二进制格式文件，实际上StoreFile就是对HFile做了轻量级包装，即StoreFile底层就是Hfile；
2. HLog File，HBase中WAL(Write Ahead Log) 的存储格式，物理上是Hadoop的Sequence File；

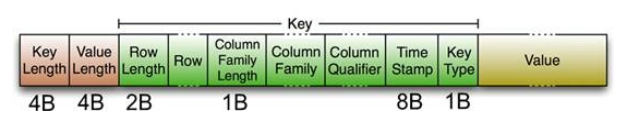
#### HFile



首先HFile文件是不定长的，长度固定的只有其中的两块：Trailer和FileInfo。正如图中所示的，Trailer 中有指针指向其他数据块的起始点。File Info中记录了文件的一些Meta信息，例如：AVG\_KEY\_LEN, AVG\_VALUE\_LEN, LAST\_KEY, COMPARATOR, MAX\_SEQ\_ID\_KEY等。Data Index和Meta Index块记录了每个Data块和Meta块的起始点。

Data Block是HBase I/O的基本单元，为了提高效率，HRegionServer中有基于LRU的Block Cache机制。每个Data块的大小可以在创建一个Table的时候通过参数指定，大号的Block有利于顺序Scan，小号Block利于随机查询。 每个Data块除了开头的Magic以外就是一个个KeyValue对拼接而成, Magic内容就是一些随机数字，目的是防止数据损坏。后面会详细介绍每个KeyValue对的内部构造。

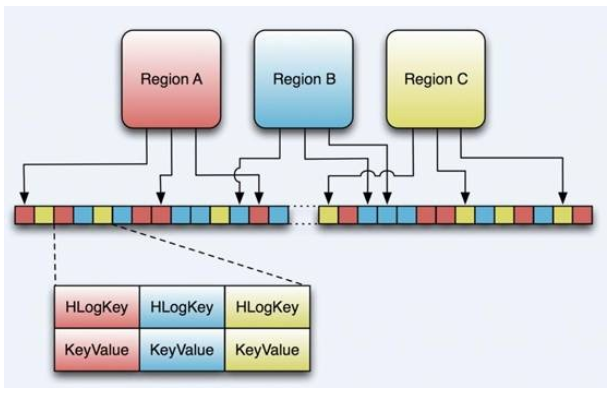
HFile里面的每个KeyValue对就是一个简单的byte数组。但是这个byte数组里面包含了很多项，并且有固定的结构。我们来看看里面的具体结构：



开始是两个固定长度的数值，分别表示Key的长度和Value的长度。紧接着是Key，开始是固定长度的数值，表示RowKey 的长度，紧接着是RowKey，然后是固定长度的数值，表示Family的长度，然后是Family，接着是Qualifier，然后是两个固定长度的数 值，表示Time Stamp和Key Type(Put/Delete)。Value部分没有这么复杂的结构，就是纯粹的二进制数据了。

#### Hlog

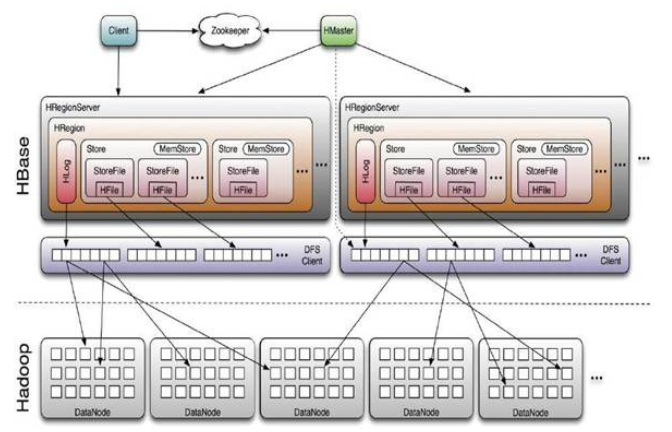
在理解了上述HStore的基本原理后，还必须了解一下HLog的功能，因为上述的HStore在系统正常工作的前提下是没有问 题的，但是在分布式系统环境中，无法避免系统出错或者宕机，因此一旦HRegionServer意外退出，MemStore中的内存数据将会丢失，这就需要引入HLog了。每个HRegionServer中都有一个HLog对象，HLog是一个实现Write Ahead Log的类，在每次用户操作写入MemStore的同时，也会写一份数据到HLog文件中(HLog文件格式见后续)，HLog文件定期会滚动出新的，并 删除旧的文件(已持久化到StoreFile中的数据)。当HRegionServer意外终止后，HMaster会通过Zookeeper感知 到，HMaster首先会处理遗留的 HLog文件，将其中不同Region的Log数据进行拆分，分别放到相应region的目录下，然后再将失效的region重新分配，领取 到这些region的HRegionServer在Load Region的过程中，会发现有历史HLog需要处理，因此会Replay HLog中的数据到MemStore中，然后flush到StoreFiles，完成数据恢复。



上图中示意了HLog文件的结构，其实HLog文件就是一个普通的Hadoop Sequence File，Sequence File 的Key是HLogKey对象，HLogKey中记录了写入数据的归属信息，除了table和region名字外，同时还包括 sequence number和timestamp，timestamp是“写入时间”，sequence number的起始值为0，或者是最近一次存入文件系统中sequence number。

HLog Sequece File的Value是HBase的KeyValue对象，即对应HFile中的KeyValue。

## Hbase系统架构



### Client

HBase Client使用HBase的RPC机制与HMaster和HRegionServer进行通信，对于管理类操作，Client与HMaster进行RPC;对于数据读写类操作，Client与HRegionServer进行RPC。

### Zookeeper

Zookeeper Quorum中除了存储了-ROOT-表的地址和HMaster的地址，HRegionServer也会把自己以Ephemeral方式注册到 Zookeeper中，使得HMaster可以随时感知到各个HRegionServer的健康状态。此外，Zookeeper也避免了HMaster的单点问题。

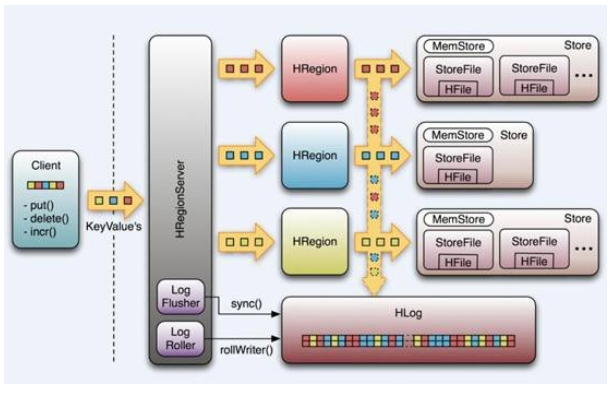
### HMaster

HMaster没有单点问题，HBase中可以启动多个HMaster，通过Zookeeper的Master Election机制保证总有一个Master运行，HMaster在功能上主要负责Table和Region的管理工作：

* 管理用户对Table的增、删、改、查操作
* 管理HRegionServer的负载均衡，调整Region分布
* 在Region Split后，负责新Region的分配
* 在HRegionServer停机后，负责失效HRegionServer 上的Regions迁移

### HRegionServer

HRegionServer主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBase中最核心的模块。



HRegionServer内部管理了一系列HRegion对象，每个HRegion对应了Table中的一个 Region，HRegion中由多个HStore组成。每个HStore对应了Table中的一个Column Family的存储，可以看出每个Column Family其实就是一个集中的存储单元，因此最好将具备共同IO特性的column放在一个Column Family中，这样最高效。

HStore存储是HBase存储的核心了，其中由两部分组成，一部分是MemStore，一部分是StoreFiles。 MemStore是Sorted Memory Buffer，用户写入的数据首先会放入MemStore，当MemStore满了以后会Flush成一个StoreFile(底层实现是HFile)， 当StoreFile文件数量增长到一定阈值，会触发Compact合并操作，将多个StoreFiles合并成一个StoreFile，合并过程中会进 行版本合并和数据删除，因此可以看出HBase其实只有增加数据，所有的更新和删除操作都是在后续的compact过程中进行的，这使得用户的写操作只要 进入内存中就可以立即返回，保证了HBase I/O的高性能。当StoreFiles Compact后，会逐步形成越来越大的StoreFile，当单个StoreFile大小超过一定阈值后，会触发Split操作，同时把当前 Region Split成2个Region，父Region会下线，新Split出的2个孩子Region会被HMaster分配到相应的HRegionServer 上，使得原先1个Region的压力得以分流到2个Region上。

## Hbase安装

登录官网<https://www.apache.org/>下载需要的hbase安装包。（注意：hadoop版本、hbase版本、zookeeper版本必须匹配。）

### 单机模式

单机模式下不需要HDFS，因此不需要事先启动Hadoop，直接启动HBase即可。

1. 设置变量

export HBASE\_HOME=/opt/hbase-2.1.0-singleton

export HBASE\_LOG\_DIR=/home/bigdata/hbase/logs

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

1. hbase-env.sh

为HBase设置Java目录，并从conf文件夹打开hbase-env.sh文件。编辑JAVA\_HOME环境变量，改变路径到当前JAVA\_HOME变量。

export JAVA\_HOME=/usr/local/jdk1.8.0\_171

1. hbase-site.xml

这是HBase的主配置文件，设置数据目录到合适的位置。

<configuration>

<!-- 数据存放目录 -->

<property>

<name>hbase.rootdir</name>

<value>file:/home/bigdata/hbase/hfiles</value>

</property>

<!-- 自带zookeeper数据存放目录 -->

<property>

<name>hbase.zookeeper.property.dataDir</name>

<value>/home/bigdata/hbase/zookeeper</value>

</property>

<!--在本地文件系统中跑hbase，请禁掉此项-->

<property>

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

<value>false</value>

</property>

</configuration>

都是本地文件。

1. 启动hbase

$HBASE\_HOME/bin/start-hbase.sh

1. 访问hbase

<http://ip:16010/master-status>

1. 停止hbase

$HBASE\_HOME/bin/stop-hbase.sh

### 模拟分布式模式

在本模式配置及运行前，请确保hdfs正常运行。

1. 设置变量（vi ~/.bashrc source ~/.bashrc）

export HBASE\_HOME=/opt/hbase-2.1.0-singdistributed

export HBASE\_LOG\_DIR=/home/bigdata/hbase/logs

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

1. hbase-env.sh

为HBase设置Java目录，并从conf文件夹打开hbase-env.sh文件。编辑JAVA\_HOME环境变量，改变路径到当前JAVA\_HOME变量。

export JAVA\_HOME=/usr/local/jdk1.8.0\_171

export HBASE\_MANAGES\_ZK=false

**HBASE\_MANAGES\_ZK表示是否使用自带的zookeeper，默认是使用true，此处配置不使用。**

1. hbase-site.xml

这是HBase的主配置文件，设置数据目录到合适的位置。

<configuration>

<!-- 数据存放目录 hadoop的ha集群hdfs://ns1/hbase -->

<property>

<name>hbase.rootdir</name>

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

</property>

<!-- 自带zookeeper数据存放目录 -->

<property>

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

<value>namenode:2181,datanode1:2181,datanode2:2181</value>

</property>

<!-- 开启集群模式 -->

<property>

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

<value>true</value>

</property>

</configuration>

**hdfs://ns1****/hbase配置的是hadoop ha服务，如果非ha，则写成hdfs://namenode:9000/hbase。**

1. 启动hbase

$HBASE\_HOME/bin/start-hbase.sh

1. 检查**在HDFS的HBase目录**

$HADOOP\_HOME/bin/hadoop fs -ls /hbase

1. 访问hbase的web页面

<http://ip:16010/master-status>

1. 停止hbase

$HBASE\_HOME/bin/stop-hbase.sh

### 完全分布式集群模式

首先安装了zookeeper、hadoop，并相关的服务器做了ntp校时。

1. 修改服务器ulimit

在root用户下，修改 /etc/security/limits.conf 文件，在最后加上nofile（文件数量）、nproc（进程数量）属性，如下：

hadoop - nofile 32768

hadoop - nproc 32000

在root用户下，修改/etc/security/limits.d/20-nproc.conf（文件名可能不一样）文件，如下：

\* soft nproc 32000

root soft nproc unlimited

注意：修改完后，使用scp将其复制到其他机器上。然后重启所有机器（注意先把所有服务都停止）。

1. 设置变量（vi ~/.bashrc source ~/.bashrc）

export HBASE\_HOME=/opt/hbase-2.1.0

export HBASE\_LOG\_DIR=/home/bigdata/hbase/logs

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

注意：修改完后，使用scp将其复制到其他机器上，并且使其生效（source ~/.bashrc）。

1. hbase-env.sh

为HBase设置Java目录，并从conf文件夹打开hbase-env.sh文件。编辑JAVA\_HOME环境变量，改变路径到当前JAVA\_HOME变量。

export JAVA\_HOME=/usr/local/jdk1.8.0\_171

export HBASE\_MANAGES\_ZK=false

export HBASE\_HEAPSIZE=1G

**HBASE\_MANAGES\_ZK表示是否使用自带的zookeeper，默认是使用true，此处配置不使用。**

1. hbase-site.xml配置

<configuration>

<!-- 数据存放目录 这里配置的是hdfs -->

<property>

<name>hbase.rootdir</name>

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

</property>

<!-- dfs支持文件追加功能 -->

<property>

<name>dfs.support.append</name>

<value>true</value>

</property>

<!-- 打开备份 -->

<property>

<name>hbase.replication</name>

<value>true</value>

</property>

<!-- HBase数据备份数 -->

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<!-- rpc请求的线程数量，默认值是10，生产环境建议使用100，也不是越大越好，特别是当请求内容很大的时候，比如scan/put几M的数据，会占用过多的内存，有可能导致频繁的GC，甚至出现内存溢出 -->

<property>

<name>hbase.regionserver.handler.count</name>

<value>100</value>

</property>

<!-- zookeeper集群列表 取值zoo.cfg 配置的server列表 -->

<property>

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

<value>namenode:2181,datanode1:2181,datanode2:2181</value>

</property>

<!-- zookeeper数据目录 取值zoo.cfg 配置的dataDir -->

<property>

<name>hbase.zookeeper.property.dataDir</name>

<value>/home/bigdata/zookeeper/data</value>

</property>

<!-- 开启hbase集群模式 -->

<property>

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

<value>true</value>

</property>

<!-- 本地临时文件目录 -->

<property>

<name>hbase.tmp.dir</name>

<value>/home/bigdata/hbase/tmp/hbase-${user.name}</value>

</property>

<!-- 本地存储的目录 -->

<property>

<name>hbase.local.dir</name>

<value>/home/bigdata/hbase/local</value>

</property>

<!-- HDFS中一个用来存放临时数据的目录 -->

<property>

<name>hbase.fs.tmp.dir</name>

<value>/hbase/user/${user.name}/hbase-staging</value>

</property>

<!-- 各节点时间不一致，导致RegionServer 退出，增大容忍度，默认是30s，但不要太大 单位：秒 -->

<property>

<name>hbase.master.maxclockskew</name>

<value>180000</value>

</property>

<!-- ZooKeeper 会话超时.HBase把这个值传递改zk集群，向他推荐一个会话的最大超时时间 单位：秒 -->

<property>

<name>zookeeper.session.timeout</name>

<value>180000</value>

</property>

</configuration>

1. 配置HRegionServer节点regionservers

datanode1

datanode2

配置上hadoop的datanode的所有节点。

1. *配置*HMaster备份*节点backup-masters*

datanode1

当主节点挂了后，备份主节点将会代替其工作。

1. *将配置好的hbase复制其他机器上*

scp -r /op/hbase-2.1.0/ root@datanode1:/opt/

scp -r /opt/hbase-2.1.0/ root@datanode2:/opt/

修改权限：

chown -R hadoop:hadoop hbase-2.1.0/

chmod 775 hbase-2.1.0/

1. 启动hbase集群

第一步：启动hbase master（在主节点上执行）

$HBASE\_HOME/bin/hbase-daemon.sh start master &

第二步：启动hbase regionserver（在每个子节点上执行）

$HBASE\_HOME/bin/hbase-daemon.sh start regionserver &

第三步：启动hbase master backup（在备份主节点上执行）

$HBASE\_HOME/bin/hbase-daemon.sh start master --backup &

或者

$HBASE\_HOME/bin/start-hbase.sh

$HBASE\_HOME/bin/hbase-daemon.sh start master --backup &（在备份主节点上执行）

1. 访问

<http://ip:16010/master-status>（主节点或备份主节点ip）

1. 停止hbase集群

第一步：停止hbase regionserver（在每个子节点上执行）

$HBASE\_HOME/bin/hbase-daemon.sh stop regionserver &

第三步：停止hbase master backup（在备份主节点上执行）

$HBASE\_HOME/bin/hbase-daemon.sh stop master --backup &

第三步：停止hbase master（在主节点上执行）

$HBASE\_HOME/bin/hbase-daemon.sh stop master &

或者

$HBASE\_HOME/bin/stop-hbase.sh

## Hbase Shell命令

使用以下命令启动HBase shell：

$HBASE\_HOME/bin/hbase shell

通用命令：

* **status:** 提供HBase的状态，例如，服务器的数量。
* **version:** 提供正在使用HBase版本。
* **table\_help:** 表引用命令提供帮助。
* **whoami:** 提供有关用户的信息。

数据定义语言，这些是关于HBase在表中操作的命令：

* **create:** 创建一个表。
* **list:** 列出HBase的所有表。
* **disable:** 禁用表。
* **is\_disabled:** 验证表是否被禁用。
* **enable:** 启用一个表。
* **is\_enabled:** 验证表是否已启用。
* **describe:** 提供了一个表的描述。
* **alter:** 改变一个表。
* **exists:** 验证表是否存在。
* **drop:** 从HBase中删除表。
* **drop\_all:** 丢弃在命令中给出匹配“regex”的表。

数据操纵语言：

* **put:** 把指定列在指定的行中单元格的值在一个特定的表。
* **get:** 取行或单元格的内容。
* **delete:** 删除表中的单元格值。
* **deleteall:** 删除给定行的所有单元格。
* **scan:** 扫描并返回表数据。
* **count:** 计数并返回表中的行的数目。
* **truncate:** 禁用，删除和重新创建一个指定的表。

### 基本命令

status：返回包括在系统上运行的服务器的细节和系统的状态。

version：返回HBase系统使用的版本。

table\_help：表引用命令提供帮助。

whoami：提供有关用户的信息。

### 创建表（create）

shell实现：

create 'student', 'name', 'sex', 'age'

java实现：

### 列出表

shell实现：

list

java实现：

### 表描述

shell实现：

describe 'student'

### 禁用表

shell实现：

disable 'student'

### 判断表是否被禁用

shell实现：

is\_disabled 'student'

### 启用表

shell实现：

enable 'student'

### 判断表是否存在

shell实现：

exists 'student'

### 修改表

shell实现：

删除一列：alter 'student', 'delete' => 'age'

增加一列：

### 删除表

在删除一个表之前必须先将其禁用。

shell实现：

disable 'student'

drop 'student'

### 表权限

可以授予和撤销HBase用户的权限。也有出于安全目的，三个命令：grant, revoke 和user\_permission。

* grant命令授予特定的权限，如读，写，执行和管理表给定一个特定的用户。 grant命令的语法如下：

grant <user> <permissions> [<table> [<column family> [<column; qualifier>]]

可以从RWXCA组，其中给予零个或多个特权给用户

R - 代表读取权限

W - 代表写权限

X - 代表执行权限

C - 代表创建权限

A - 代表管理权限

* revoke命令用于撤销用户访问表的权限。它的语法如下：

revoke <user>

* user\_permission命令用于列出特定表的所有权限。user\_permission的语法如下：

user\_permission 'tablename'

### 插入数据

在HBase表中创建的数据，可以下面的命令和方法：

* put 命令

put '<table name>', 'row1', '<colfamily:colname>', '<value>'

如：

put 'student', '1', 'name', 'chen gao ke'

put 'student', '1', 'age', '20'

* add() - Put类的方法
* put() - HTable 类的方法

### 读取数据

shell命令：

* 读取一行

get '<table name>', 'row1'

如：

get 'student', '1'

* 读取指定列：

get 'table name', 'rowid', {COLUMN => 'column family:column name'}

如：

get 'student', '1', {COLUMN => 'age'}

### 更新数据

shell命令：

put 'table name', 'row ', 'Column family:column name', 'new value'

如：

put 'student', '1', 'age', '22'

### 删除数据

shell命令：

* 删除指定单元格（删除新版本的数据后，将显示旧版本的数据）：

delete '<table name>', '<row>', '<column name >', '<time stamp>'

如：

delete 'student', '1', 'age'

* 使用“deleteall”命令，可以删除一行中所有单元格（彻底删除一行数据）。

deleteall '<table name>', '<row>'

如：

deleteall 'student', '1'

### 清空数据

truncate命令将禁止删除并重新创建一个表，数据将被清空。

语法：truncate 'table name'

如：

truncate 'student'

### 查询总量

可以使用count命令计算表的行数量。

语法：count '<table name>'

如：

count 'student'

### 扫描表

scan 命令用于查看HTable数据。使用 scan 命令可以得到表中的数据。

语法：scan '<table name>'

如：

scan 'student'

### 退出shell

shell实现：

键入exit命令退出shell

## Hbase JAVA API

## Hbase预分区

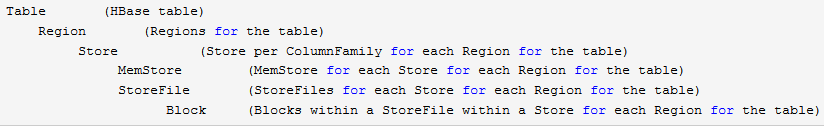
HBase表在刚刚被创建时，只有1个分区（region），当一个region过大（达到hbase.hregion.max.filesize属性中定义的阈值，默认10GB）时，表将会进行split，分裂为2个分区。表在进行split的时候，会耗费大量的资源，频繁的分区对HBase的性能有巨大的影响。HBase提供了预分区功能，即用户可以在创建表的时候对表按照一定的规则分区。

在建表的时候，不预先设定好分区，随着表内数据的增多，HBase会自动把表进行split，但是这样做有几点问题：

1. 数据往一个region上写，会有写热点问题；
2. split操作会消耗宝贵的集群I/O资源，如果数据增长过快，可能会相对频繁的发生。

### Region 概念

Region是表获取和分布的基本元素，由每个列族的一个Store组成。对象层级图如下：



Region的大小是一个棘手的问题，需要考量如下几个因素：

* Region是HBase中分布式存储和负载均衡的最小单元。不同Region分布到不同RegionServer上，但并不是存储的最小单元；
* Region由一个或者多个Store组成，每个store保存一个columns family，每个Strore又由一个memStore和0至多个StoreFile 组成。memStore存储在内存中， StoreFile存储在HDFS上；
* HBase通过将region切分在许多机器上实现分布式。也就是说，你如果有16GB的数据，只分了2个region， 你却有20台机器，有18台就浪费了，因此需要进行预分区；
* region数目太多就会造成性能下降，现在比以前好多了。但是对于同样大小的数据，700个region比3000个要好；
* region数目太少就会妨碍可扩展性，降低并行能力。有的时候导致压力不够分散。这就是为什么，你向一个10节点的HBase集群导入200MB的数据，大部分的节点是idle的；
* RegionServer中1个region和10个region索引需要的内存量没有太多的差别。

### 如何使数据分区

根据hbase中rowkey的分布，可以设置相应的region（分区）。这样就可以防止大数据插入时的热点问题，提高数据插入的效率。但是与此同时我们会遇到以下几个问题：

1. 首先得知道：数据的rowkey是什么样的？（rowkey设计）
2. 根据rowkey，需要规划一下region数目？每个region的startkey和endkey是多少？（分区数设计）

每个分区都由一对分区键控制，**如果需要创建一个分区表，那么首先得确定该表以什么键分区**。一般来说，因为在ASCII码表中，最大的可打印字符是~，再者就是|。所以可以选择这两种字符作为分区键的标识来结束。

**rowkey设计**：

比如rowkey组成：两位随机数+时间戳+学生号，两位随机数的范围从00-99，则可划分10个region来存储数据，每个region对应的rowkey范围如下：

-10，10-20，20-30，30-40，40-50，50-60，60-70，70-80，80-90，90-

分区键设计为：10|，20|，30|，40|，50|，60|，70|，80|，90|

实现：

create 'student', 'name', 'sex', 'age', SPLITS => ['10','20','30','40','50','60','70','80','90']

或

指定分区文件，create 'student', 'name', 'sex', 'age', {SPLITS\_FILE => 'splits.txt'}，文件内容如下：

10|

20|

30|

40|

50|

60|

70|

80|

90|

**分区数设计**：

应该指定为region server总CPU核数的0.5~1倍之间。测试集群region sever每个节点40核，4个节点共160核（则分区个数为80至160）

# Phoenix

## Phoenix简介

hbase提供很方便的shell脚本以及java API等方式对Hbase进行操作，但是对于很对已经习惯了关系型数据库操作的开发来说，有一定的学习成本，如果可以像操作mysql等一样通过sql实现对Hbase的操作，那么很大程度降低了Hbase的使用成本。Apache Phoenix 组件就完成了这种需求，官方注解为 “Phoenix -we put the SQL back in NoSql”，通过官方说明，Phoenix的性能很高，相对于hbase 原生的scan并不会差多少，而对于类似的组件hive、Impala等，性能有着显著的提升。

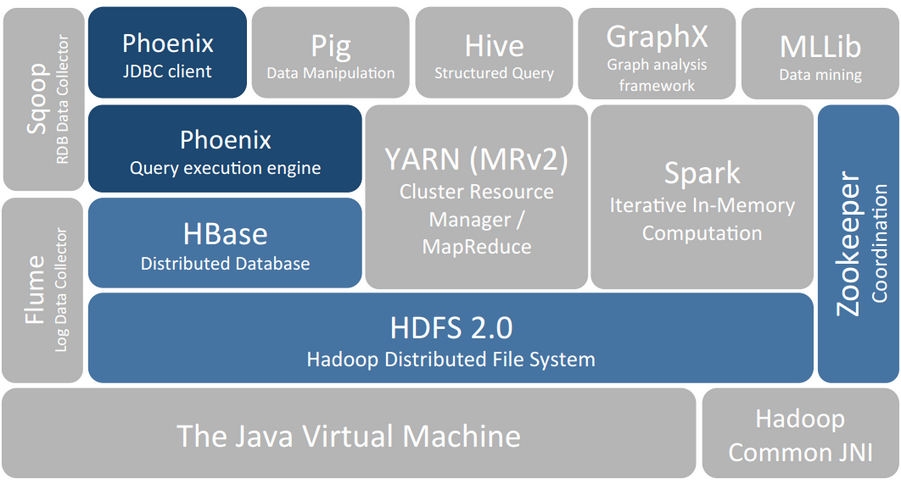
phoenix是构建在Hbase之上的，使用标准的SQL操作Hbase，可以做联机事务处理，拥有低延迟的特性。phoenix会把SQL编译成一系列的Hbase的scan操作，然后把scan结果生成标准的JDBC结果集，其底层由于使用了Hbase的API，协处理器，过滤器，处理千万级行的数据也只用毫秒或秒级就搞定。

相应的元数据存储在Hbase表中，元数据用于描述phoenix表与hbase之间的关系，映射。

phoenix表可能通过两种方式创建：

* 从头开始，这种方式Hbase的表与列族都会自动创建。
* 映射到一个存在的Hbase表。

Phoenix在Hadoop生态系统中的位置：



## Phoenix安装

1. 下载

下载地址：http://phoenix.apache.org/download.html，选择合适的版本下载，此处下载apache-phoenix-5.0.0-HBase-2.0-bin.tar.gz。

1. 解压

tar -zxvf apache-phoenix-5.0.0-HBase-2.0-bin.tar.gz -C /opt/，改名为phoenix-5.0.0-HBase-2.0

1. 将phoenix-5.0.0-HBase-2.0下的phoenix-5.0.0-HBase-2.0-server.jar拷贝到每个hbase节点的lib目录下。
2. 环境变量（vi ~/.bashrc，source ~/.bashrc）

export PHOENIX\_HOME=/opt/phoenix-5.0.0-HBase-2.0/

export PHOENIX\_CLASSPATH=$PHOENIX\_HOME

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

1. 删除/opt/phoenix-5.0.0-HBase-2.0下的\*javadoc.jar、\*sources.jar与\*tests.jar

rm -rf \*javadoc.jar

rm -rf \*sources.jar

rm -rf \*tests.jar

1. 配置hbase并重启
2. 启动Phoenix

/opt/phoenix-5.0.0-HBase-2.0/bin/sqlline.py namenode,datanode1,datanode2:2181

注：如果启动Phoenix客户端失败，则多次重启hbase，再重试。

## Phoenix Shell命令

执行以下命令启动Phoenix Shell：

/opt/phoenix-5.0.0-HBase-2.0/bin/sqlline.py namenode,datanode1,datanode2:2181

### 基本命令

* 帮助：!help
* 退出：!quit
* 查看历史操作：!history
* 查看表结构：!describe table\_name，如!describe student
* 列出所有表：!tables

## SquirreL连接Phoenix工具

### 安装

1. 下载地址：<http://squirrel-sql.sourceforge.net/>
2. 在cmd下，进入到squirrel-sql-3.8.1-standard.jar所在目录，执行如下命令并完成安装：

java -jar squirrel-sql-3.8.1-standard.jar

1. 配置Squirrel客户端：

* 首先将phoenix-5.0.0-HBase-2.0-client.jar，拷贝到squirrel安装目录的lib文件夹下，并点击squirrel-sql.bat启动；
* C:\Windows\System32\drivers\etc\hosts添加域名解析，必须添加且后面需要用域名替代ip；

192.168.1.119 namenode

192.168.1.120 datanode1

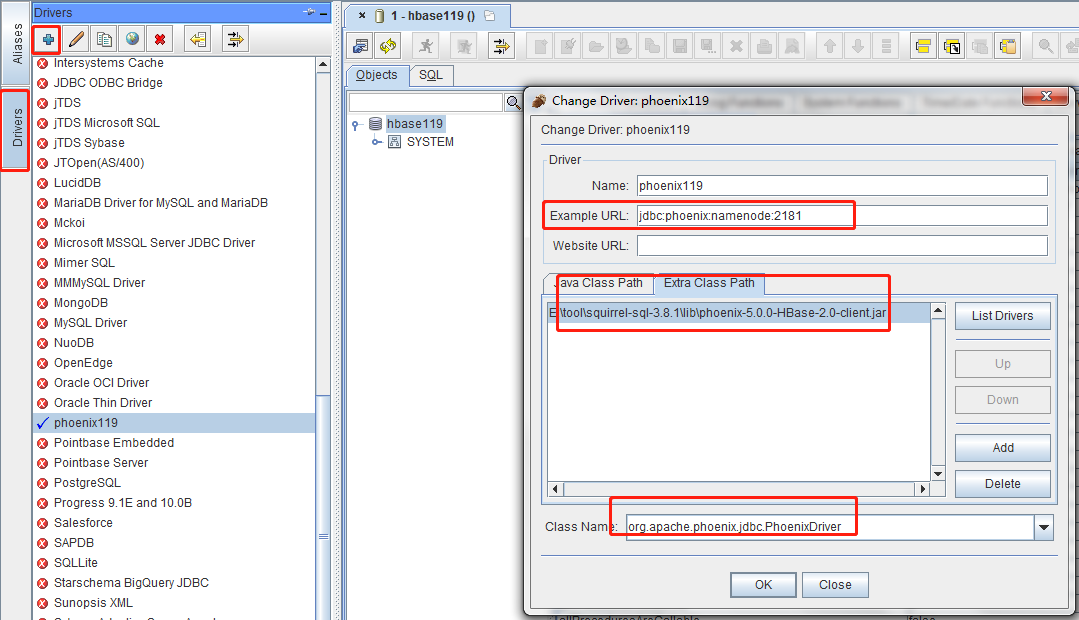
192.168.1.121 datanode2

* 添加驱动(Drivers -> New Driver)；

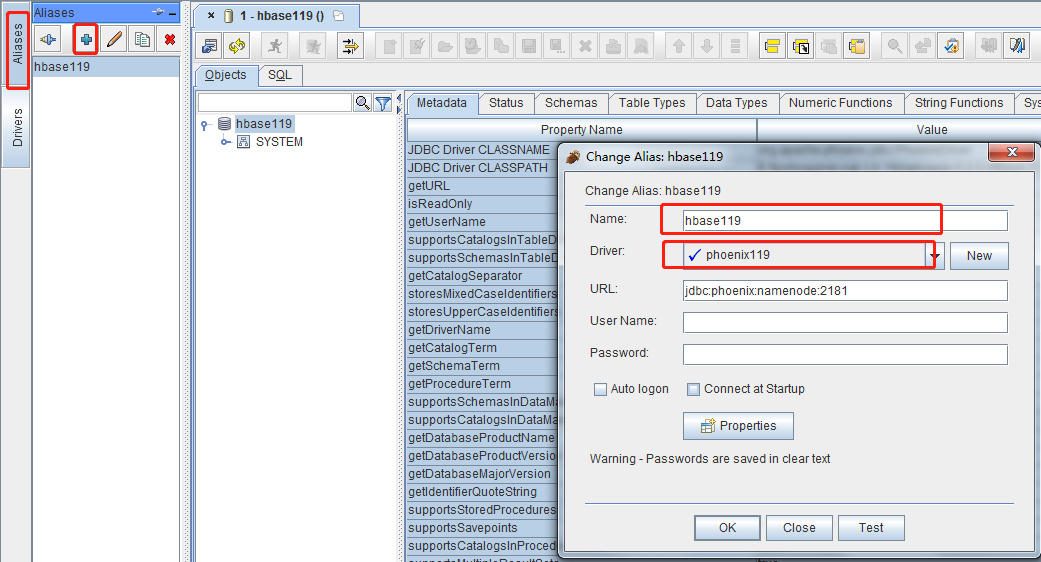
Class Name：org.apache.phoenix.jdbc.PhoenixDriver

Example URL：jdbc:phoenix:namenode:2181

Extra Class Path：选择并添加驱动phoenix-5.0.0-HBase-2.0-client.jar



* 添加实例(Aliases -> New Aliases)；



## 表操作

### 创建表

#### 创建一般的表

create table "student" (

"id" varchar(64) not null PRIMARY KEY,

"name" varchar(50),

"sex" varchar(1),

"age" integer(3)

);

以上创建了一个student表，包括id、name、sex、age等字段，其中id为主键。

注意：默认表名和字段都是大写，如果不用双引号，则创建后表名和字段名称是大写的。之后使用也必须要双引号。

#### 使用多列族

列族包含相关的数据都在独立的文件中，在Phoenix设置多个列族可以提高查询性能。例如：

create table "test" (

"id" varchar(64),

"a.col1" varchar(20),

"a.col2" varchar(10),

"b.col3" varchar(30),

"b.col4" varchar(10),

constraint pk PRIMARY KEY("id")

) SALT\_BUCKETS=5;

#### 使用压缩

在数据量大的表上使用压缩算法来提高性能。例如：

create table "test" (

"id" varchar(64),

"a.col1" varchar(20),

"a.col2" varchar(10),

"b.col3" varchar(30),

"b.col4" varchar(10),

constraint pk PRIMARY KEY("id")

) COMPRESSION='GZ';

#### 创建可变与不可变表

有些数据一次写入后就不再需要变更，则可创建不可变表。但一般情况下，表的数据都会被增删改，默认创建的表是可变的。

如果需要创建不可变的表，则需要增加IMMUTABLE\_ROWS=true参数。也可以修改表的属性，如ALTER TABLE my\_table SET IMMUTABLE\_ROWS=false。

### 删除表

删除表和其他的数据库类似。不同的是可以加上CASCADE关键字，用于删除表的同时删除基于该表的所有视图。

## 视图操作

### 创建视图

例如：

create view "student\_view" as select \* from "student" where "age" > 20;

注意：Phoenix的视图是没办法只获取一部分数据的数据的（必须使用\*），即使使用子查询也不行。

### 删除视图

例如：

drop view "student\_view";

drop view if exists "student\_view";

## 表预分区

创建表时，默认只有一个分区，可以指定划分多个数据分区，并可以精确的指定要根据什么值来做预分区，使数据划分为多个分区，以提高数据的读写性能。

### 表预分区设计

create table "student" (

"id" varchar(64),

"name" varchar(50),

"sex" varchar(1),

"age" integer(3),

constraint pk PRIMARY KEY("id")

) SALT\_BUCKETS=120;

以上创建一个student表，包括id、name、sex、age等字段，其中id为主键，并通过SALT\_BUCKETS关键字来设定120个预分区。

### 数据分区实现

使用Salting方式能提升读写性能，Salting能够通过预分区(pre-splitting)数据到多个region中来显著提升读写性能。

Phoenix实现数据分区：本质是在hbase中，rowkey的byte数组的第一个字节位置设定一个系统生成的byte值，这个byte值是由主键生成rowkey的byte数组做一个哈希算法，计算得来的。Salting之后可以把数据分布到不同的region上，这样有利于phoenix并发的读写操作。

简而言之，如果我们用Phoenix创建了一个saltedtable，那么向该表中写入数据时，原始的rowkey的前面会被自动地加上一个byte（不同的rowkey会被分配不同的byte，值范围在1 ~ 256），使得连续的rowkeys也能被均匀地分布到多个regions。

注意：

* Salt bucket的个数必须在1-256之间；官网上并没有对分区数应设为多少比较合适给出建议，网上有博客说应该指定为region server总CPU核数的0.5~1倍之间。测试集群region sever每个节点40核，4个节点共160核（则分区个数为80至160）。
* 创建salted table后，应该使用Phoenix SQL来读写数据，而不要混合使用Phoenix SQL和HBase API；
* 如果通过Phoenix创建了一个salted table，那么只有通过Phoenix SQL插入数据才能使得被插入的原始rowkey前面被自动加上一个byte，通过HBase shell插入数据无法prefix原始的rowkey；

### 手动控制数据分区

Salting能够自动的设置表预分区，但是还需要控制表是如何分区的，所以在建phoenix表时，可以精确的指定要根据什么值来做预分区。比如rowkey组成：两位随机数+时间戳+学生号，两位随机数的范围从00-99，则可划分10个region来存储数据，每个region对应的rowkey范围如下：

-10，10-20，20-30，30-40，40-50，50-60，60-70，70-80，80-90，90-

分区键设计为：10，20，30，40，50，60，70，80，90

实现（以下自动分10个区）：

create table "student" (

"id" varchar(64),

"name" varchar(50),

"sex" varchar(1),

"age" integer(3),

constraint pk PRIMARY KEY("id")

) SPLIT ON ('10','20','30','40','50','60','70','80','90');

## 二级索引

二级索引这个特性应该是大部分用户引入Phoenix主要考虑的因素之一。HBase因其历史原因只支持rowkey索引，当使用rowkey来查询数据时可以很快定位到数据位置。现实中，业务查询需求条件往往比较复杂，带有多个查询字段组合，如果用HBase查的话，只能全表扫描进行过滤，效率很低。而Phoenix支持除rowkey外的其它字段的索引创建，即二级索引，查询效率可大幅提升。

索引有以下类型：

* Covered Indexes（覆盖索引）：即索引表中就包含想要的全部字段数据，这样就只需要通过访问索引表而无需访问主表就能得到数据。
* Functional indexes（函数索引）：从Phoeinx4.3以上就支持函数索引，其索引不局限于列，可以合适任意的表达式来创建索引，当在查询时用到了这些表达式时就直接返回表达式结果。
* Global indexes（全局索引）：全局索引适用于多读少写的场景，在写操作上会给性能带来极大的开销，因为所有的更新和写操作（DELETE,UPSERT VALUES和UPSERT SELECT）都会引起索引的更新,在读数据时，Phoenix将通过索引表来达到快速查询的目的。
* Local indexes（本地索引）：本地索引适用于写多读少，空间有限的场景，和全局索引一样，Phoneix在查询时会自动选择是否使用本地索引，使用本地索引，为避免进行写操作所带来的网络开销，索引数据和表数据都存放在相同的服务器中，当查询的字段不完全是索引字段时本地索引也会被使用，与全局索引不同的是，所有的本地索引都单独存储在同一张共享表中，由于无法预先确定region的位置，所以在读取数据时会检查每个region上的数据因而带来一定性能开销。

Phoenix支持两种索引：可变索引跟不可变索引。在可变表上建的索引是可变索引，在不可变表上建的索引是不可变索引。可变索引是指插入或删除数据的时候会同时更新索引；不可变索引适用于只写入一次不再更改的表，索引只建立一次，再插入数据不会更新索引。

### 启用二级索引

支持二级索引的必要配置：

1. 每个HregionServer节点在hbase-site.xml中添加如下配置；

<!-- 支持二级索引 regionserver节点添加-->

<property>

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

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

</property>

### Covered Indexes（覆盖索引）

即索引表中就包含想要的全部字段数据，这样就只需要通过访问索引表而无需访问主表就能得到数据。

覆盖索引的特点是把原数据存储在索引数据表中，这样在查询到索引数据时就不需要再次返回到原表查询，可以直接拿到查询结果。

创建Covered Indexes：

* 在student表的name字段上创建索引，包括sex，age字段：

create index "student\_index01" on "student"("name") include("sex", "age");

* 在student表的name与sex字段上创建索引，包括age字段：

create index "student\_index02" on "student"("name", "sex") include("age");

### Functional indexes（函数索引）

从Phoeinx4.3以上就支持函数索引，其索引不局限于列，可以合适任意的表达式来创建索引，当在查询时用到了这些表达式时就直接返回表达式结果。

函数索引的特点是能根据表达式创建索引，适用于对查询表，过滤条件是表达式的表创建索引。

创建Functional indexes：

* 在student表使用UPPER函数创建函数索引使查询出的name和age里字母都是大写的：

create index "student\_fun\_index01" on "student"(upper("name" || ' ' || "age"))

### Global indexes（全局索引）

全局索引适用于多读少写的场景，在写操作上会给性能带来极大的开销，因为所有的更新和写操作（DELETE,UPSERT VALUES和UPSERT SELECT）都会引起索引的更新,在读数据时，Phoenix将通过索引表来达到快速查询的目的。

创建Global indexes：

* 在student表的name字段上创建全局索引：

create index "student\_index03" on "student"("name");

* 在student表的name与sex字段上创建全局索引：

create index "student\_index04" on "student"("name", "sex");

### Local indexes（本地索引）

本地索引适用于写多读少，空间有限的场景，和全局索引一样，Phoneix在查询时会自动选择是否使用本地索引，使用本地索引，为避免进行写操作所带来的网络开销，索引数据和表数据都存放在相同的服务器中，当查询的字段不完全是索引字段时本地索引也会被使用，与全局索引不同的是，所有的本地索引都单独存储在同一张共享表中，由于无法预先确定region的位置，所以在读取数据时会检查每个region上的数据因而带来一定性能开销。

本地索引和原数据是存储在同一个表中的，所以更适合写多的场景。对于本地索引，查询中无论是否指定hint或者是查询的列是否都在索引表中，都会使用索引表。

创建Local indexes：

* 在student表的name字段上创建本地索引：

create local index "student\_index05" on "student"("name");

* 在student表的name与sex字段上创建本地索引：

create local index "student\_index06" on "student"("name", "sex");

### 同步与异步创建索引

默认情况下，创建索引时会进行索引数据的同步，这种同步创建索引的方法适用于数据量较小的情况。

异步创建索引需要借助MR，创建异步索引语法和同步索引相差一个关键字：ASYNC。如下：

create index "student\_index08" on "student"("name") include("sex", "age") ASYNC;

### Mutable index（可变索引）与Immutable index（不可变索引）

可变索引主要创建在可变表上，不可变索引主要创建在不可变表上。

#### Mutable index（可变索引）

mutable index，可变索引，适用于数据有增删改的场景。索引更新涉及WAL，即主表数据更新时，会把索引数据也同步更新到WAL，只有当WAL同步到磁盘时才会去更新实际的primary/index数据，以保证当中间任何一个环节异常时可通过WAL来恢复主表和索引表数据。

Phoenix默认情况创建的索引都是可变索引，除非在create table的时候显式地指定IMMUTABLE\_ROWS = true。

#### Immutable index（不可变索引）

immutable index，不可变索引，适用于数据只增加不更新并且按照时间先后顺序存储（time-series data）的场景，如保存日志数据或者事件数据等。不可变索引的存储方式是write one，append only。

不可变索引主要创建在不可变表上，适用于数据只写一次不会有Update等操作。

Phoenix默认情况创建的索引都是可变索引，当在Phoenix使用create table语句时指定IMMUTABLE\_ROWS = true表示该表上创建的索引将被设置为不可变索引。

不可变索引分为Global immutable index和Local immutable index两种。

### 索引预分区

默认索引的分区数跟表分区数一样，也可以使用SALT\_BUCKETS参数指定分区数量（本地索引不允许）。例如：

create index "student\_index09" on "student"("name") include("sex", "age") SALT\_BUCKETS=10;

### 删除索引



DROP INDEX my\_index ON my\_table

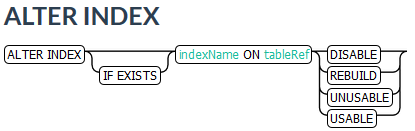
DROP INDEX my\_index ON schemaName.my\_table

DROP INDEX IF EXISTS my\_index ON my\_table

例如：

drop index "student\_index09" on "student";

### 修改索引



ALTER INDEX my\_idx ON sales.opportunity DISABLE

ALTER INDEX IF EXISTS my\_idx ON server\_metrics REBUILD

例如：

alter index "student\_index08" on "student" disable;

alter index "student\_index08" on "student" rebuild;

### 索引的优化

## 数据操作

### 插入数据

1. UPSERT INTO

类似于insert into的语句，旨在单条插入外部数据。例如：

upsert into "student"("id", "name", "sex", "age") values('123456', 'lang', '1', 25);

1. UPSERT SELECT

类似于Hive中的insert select语句，旨在批量插入其他表的数据。例如：

upsert into "student"("id", "name", "sex", "age") select "id", "name", "sex", "age" from "student\_bak";

### 删除数据

删除数据和其他数据库相似。例如：

delete from "student";

delete from "student" where name like 'lan%';

### 更新数据

由于HBase的主键设计，相同rowkey的内容可以直接覆盖，这就变相的更新了数据。例如：

upsert into "student"("id", "name", "sex", "age") values('123456', 'lang', '1', 25);

变更年龄

upsert into "student"("id", "name", "sex", "age") values('123456', 'lang', '1', 22);

简化（必须包含生成HBase的rowkey的所有字段，有些rowkey包含多个字段）：

upsert into "student"("id", "age") values('123456', 22);

### 查询数据

Phoenix作为SQL On HBase引擎必不可少的就是SQL查询语句了，它能兼容大部分的SQL查询语句，比如UNION ALL GROUP BY ORDER BY LIMIT。

## 创建SCHEMA

<http://phoenix.apache.org/language/index.html>

## 创建序列

<http://phoenix.apache.org/language/index.html>

## 创建视图

<http://phoenix.apache.org/language/index.html>

## 自定义函数

<http://phoenix.apache.org/language/index.html>

## 授权

<http://phoenix.apache.org/language/index.html>

## 使用Spark操作Phoenix

https://www.cnblogs.com/kekukekro/p/6339587.html#wiz\_toc\_33

# 遇到的问题

## Hadoop datanode 无法启动，报错：java.io.IOException: All specified directories are failed to load或者ERROR org.apache.hadoop.hdfs.server.datanode.DataNode: RECEIVED SIGNAL 15: SIGTERM

解决办法：

可能是由于在关机的时候，按照正常顺序退出集群造成的，导致dfs中的id不同。

进入dfs/data，修改VERSION文件即可，将version里面的内容修改成和master一直的。

或者直接删除dfs，格式化hdfs即可（./hdfs namenode -format）

## hadoop创建文件夹与授权

(创建文件夹)

hadoop fs -mkdir /checkpoint/ssyj

hadoop dfs –chmod 777 /checkpoint（修改文件夹权限）

## Name node is in safe mode

$HADOOP\_HDFS\_HOME/bin/hdfs dfsadmin -safemode leave

## Error: Could not find or load main class org.apache.hadoop.mapreduce.v2.app.MRAppMaster

mapred-site.xml中mapreduce.application.classpath默认的$HADOOP\_MAPRED\_HOME失效，需要改成$HADOOP\_HOME。

## hbase 错误：ERROR: Can't get master address from ZooKeeper; znode data == null

进入zookeeper中删除hbase数据。

//进入命令行

$ZOOKEEPER\_HOME/bin/zkCli.sh

命令：

显示根目录下、文件： ls /

创建文件，并设置初始内容： create /zk "test" 创建一个新的 znode节点“ zk ”以及与它关联的字符串

获取文件内容： get /zk 确认 znode 是否包含我们所创建的字符串

修改文件内容： set /zk "zkbak" 对 zk 所关联的字符串进行设置

删除文件： delete /zk 将刚才创建的 znode 删除

rmr /zk包括子节点

退出客户端： quit

帮助命令： help