目录

[Hadoop 3](#_Toc23835)

[第1章 Hadoop框架 3](#_Toc8823)

[1.1 Hadoop三大发行版本 3](#_Toc30124)

[1.2 Hadoop的优势（4高） 3](#_Toc20198)

[1.3 Hadoop组成（面试重点） 4](#_Toc21657)

[1.4 大数据技术生态体系 4](#_Toc12679)

[1.5 推荐系统框架图 5](#_Toc10775)

[第2章 Hadoop运行环境搭建（开发重点） 6](#_Toc17932)

[2.1 虚拟机环境准备 6](#_Toc1583)

[2.2 安装JDK 6](#_Toc23278)

[2.3 安装Hadoop 8](#_Toc27048)

[2.4 Hadoop目录结构 9](#_Toc28998)

[第3章 Hadoop运行模式 9](#_Toc28410)

[3.1 本地运行模式 10](#_Toc26725)

[3.2 伪分布式运行模式 10](#_Toc21508)

[3.3 完全分布式运行模式（开发重点） 16](#_Toc15157)

[第4章 Hadoop编译源码（面试重点） 25](#_Toc12903)

[4.1 准备工作 25](#_Toc1456)

[4.2 jar包安装 25](#_Toc25515)

[4.3 编译源码 27](#_Toc31464)

[第5章 常见错误及解决方案 28](#_Toc14234)

[HDFS 29](#_Toc30837)

[第1章 HDFS概述 29](#_Toc5756)

[1.1 HDFS产出背景及定义 29](#_Toc2639)

[1.2 HDFS优缺点 29](#_Toc13841)

[1.3 HDFS组成架构 30](#_Toc2638)

[1.4 HDFS文件块大小（面试重点） 31](#_Toc32636)

[第2章 HDFS的Shell操作（开发重点） 31](#_Toc8434)

[第3章 HDFS客户端操作（开发重点） 33](#_Toc30084)

[3.1 HDFS客户端环境准备 33](#_Toc22813)

[3.2 HDFS的API操作 36](#_Toc9683)

[3.3 HDFS的I/O流操作（扩展） 39](#_Toc18146)

[第4章 HDFS的数据流（面试重点） 41](#_Toc17680)

[4.1 HDFS写数据流程 41](#_Toc416)

[4.2 HDFS读数据流程 43](#_Toc21783)

[第5章 NameNode和SecondaryNameNode（面试开发重点） 44](#_Toc29680)

[5.1 NN和2NN工作机制 44](#_Toc28053)

[5.2 Fsimage和Edits解析 45](#_Toc22597)

[5.3 CheckPoint时间设置 48](#_Toc17101)

[5.4 NameNode故障处理 48](#_Toc12062)

[5.5 集群安全模式 50](#_Toc9053)

[第6章 DataNode（面试开发重点） 51](#_Toc9279)

[6.1 DataNode工作机制 51](#_Toc11565)

[6.2 数据完整性 51](#_Toc16979)

[6.3 DataNode掉线时限参数设置 52](#_Toc15102)

[6.4 服役新数据节点 52](#_Toc4313)

[6.5 退役旧数据节点 53](#_Toc22310)

[6.6 Datanode多目录配置 55](#_Toc22099)

[第7章 HDFS 2.X新特性 55](#_Toc9601)

[7.1 集群间数据拷贝 55](#_Toc18698)

[7.2 小文件存档 56](#_Toc1442)

[第8章 HDFS HA高可用 56](#_Toc2797)

[8.1 HA概述 56](#_Toc30659)

[8.2 HDFS-HA工作机制 56](#_Toc29002)

[8.3 HDFS-HA集群配置 58](#_Toc25099)

[8.4 YARN-HA配置 63](#_Toc24439)

[8.5 HDFS Federation架构设计 65](#_Toc284)

[Zookeeper 67](#_Toc29605)

[第1章 Zookeeper入门 67](#_Toc26126)

[1.1 概述 67](#_Toc32631)

[1.2 特点 67](#_Toc13631)

[1.3 数据结构 68](#_Toc21006)

[1.4 应用场景 68](#_Toc13471)

[1.5 下载地址 70](#_Toc18339)

[第2章 Zookeeper安装 71](#_Toc28001)

[2.1 本地模式安装部署 71](#_Toc29429)

[2.2 配置参数解读 71](#_Toc1085)

[第3章 Zookeeper实战（开发重点） 72](#_Toc5900)

[3.1 分布式安装部署 72](#_Toc17566)

[3.2 客户端命令行操作 73](#_Toc7196)

[3.3 API应用 76](#_Toc32375)

[3.4 监听服务器节点动态上下线案例（扩展） 77](#_Toc16534)

[第4章 Zookeeper内部原理 81](#_Toc5016)

[4.1 节点类型 81](#_Toc4006)

[4.2 Stat结构体 81](#_Toc14511)

[4.3 监听器原理（面试重点） 82](#_Toc12459)

[4.4 Paxos算法（扩展） 82](#_Toc20321)

[4.5 选举机制（面试重点） 84](#_Toc32625)

[4.6 写数据流程 85](#_Toc10782)

[第5章 企业面试真题 86](#_Toc6960)

[5.1 请简述ZooKeeper的选举机制 86](#_Toc20379)

[5.2 ZooKeeper的监听原理是什么？ 86](#_Toc19109)

[5.3 ZooKeeper的部署方式有哪几种？集群中的角色有哪些？集群最少需要几台机器？ 86](#_Toc17176)

[5.4 ZooKeeper的常用命令 86](#_Toc17848)

# Hadoop

# 第1章 Hadoop框架

## 1.1 Hadoop三大发行版本

Hadoop三大发行版本：Apache、Cloudera、Hortonworks。

Apache版本最原始（最基础）的版本，对于入门学习最好。

Cloudera在大型互联网企业中用的较多。CDH

Hortonworks文档较好。HDP

1. Apache Hadoop

官网地址：http://hadoop.apache.org/releases.html

下载地址：<https://archive.apache.org/dist/hadoop/common/>

1. Cloudera Hadoop

官网地址：https://www.cloudera.com/downloads/cdh/5-10-0.html

下载地址：http://archive-primary.cloudera.com/cdh5/cdh/5/

（1）2008年成立的Cloudera是最早将Hadoop商用的公司，为合作伙伴提供Hadoop的商用解决方案，主要是包括支持、咨询服务、培训。

（2）2009年Hadoop的创始人Doug Cutting也加盟Cloudera公司。Cloudera产品主要为CDH，Cloudera Manager，Cloudera Support

（3）CDH是Cloudera的Hadoop发行版，完全开源，比Apache Hadoop在兼容性，安全性，稳定性上有所增强。

（4）Cloudera Manager是集群的软件分发及管理监控平台，可以在几个小时内部署好一个Hadoop集群，并对集群的节点及服务进行实时监控。Cloudera Support即是对Hadoop的技术支持。

（5）Cloudera的标价为每年每个节点4000美元。Cloudera开发并贡献了可实时处理大数据的Impala项目。

3. Hortonworks Hadoop

官网地址：https://hortonworks.com/products/data-center/hdp/

下载地址：https://hortonworks.com/downloads/#data-platform

（1）2011年成立的Hortonworks是雅虎与硅谷风投公司Benchmark Capital合资组建。

（2）公司成立之初就吸纳了大约25名至30名专门研究Hadoop的雅虎工程师，上述工程师均在2005年开始协助雅虎开发Hadoop，贡献了Hadoop80%的代码。

（3）雅虎工程副总裁、雅虎Hadoop开发团队负责人Eric Baldeschwieler出任Hortonworks的首席执行官。

（4）Hortonworks的主打产品是Hortonworks Data Platform（HDP），也同样是100%开源的产品，HDP除常见的项目外还包括了Ambari，一款开源的安装和管理系统。

（5）HCatalog，一个元数据管理系统，HCatalog现已集成到Facebook开源的Hive中。Hortonworks的Stinger开创性的极大的优化了Hive项目。Hortonworks为入门提供了一个非常好的，易于使用的沙盒。

（6）Hortonworks开发了很多增强特性并提交至核心主干，这使得Apache Hadoop能够在包括Window Server和Windows Azure在内的Microsoft Windows平台上本地运行。定价以集群为基础，每10个节点每年为12500美元。

## 1.2 Hadoop的优势（4高）

Hadoop的优势：

1）高可用：Hadoop底层维护多个数据副本，所以即使Hadoop某个计算元素或存储出现故障，也不会导致数据的丢失

2）高可扩展性：在集群间分配任务数据，可方便的扩展数以千计的节点

3）高效性：在MapReduce的思想下，Hadoop是并行工作的，以加快任务处理速度

4）高容错性：能够自动将失败的任务重新分配

## 1.3 Hadoop组成（面试重点）

|  |  |
| --- | --- |
| Hadoop 1.x组成 | Hadoop 2.x组成 |
| MapReduce（计算+资源调度） | MapReduce（计算） |
| Yarn（资源调度） |
| HDFS（数据存储） | HDFS（数据存储） |
| Common（辅助工具） | Common（辅助工具） |

### 1.3.1 HDFS架构概述

HDFS（Hadoop Distributed File System）的架构概述，如图2-23所示。

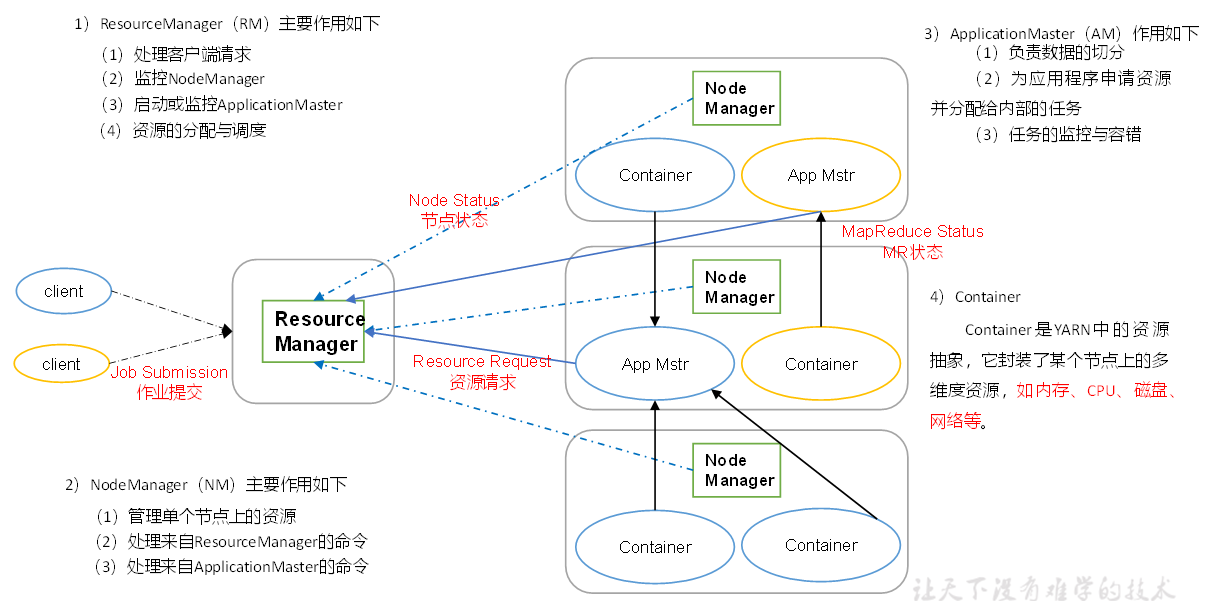
1）NameNode：存储文件的元数据，如文件名、文件目录结构，文件属性（生成时间，副本数、文件权限），以及每个文件的块列表和所在的DataNode等。

2）DataNode：在本地文件系统存储文件块数据以及块数据的校验和。

Ps：Hadoop1.0的时候存在SecondaryNameNode，用来辅助NameNode合并Fsimage和Edits。

### 1.3.2 YARN架构概述

YARN架构概述，如图2-24所示。



1）ResourceManager（RM）

处理客户端请求、监控NodeManager、启动或监控ApplicationMaster、资源的分配与调度

2）NodeManager（NM）

管理单个节点上的资源、处理来自ResourceManager的命令、处理来自ApplicationMaster的命令

3）ApplicationMaster（AM）

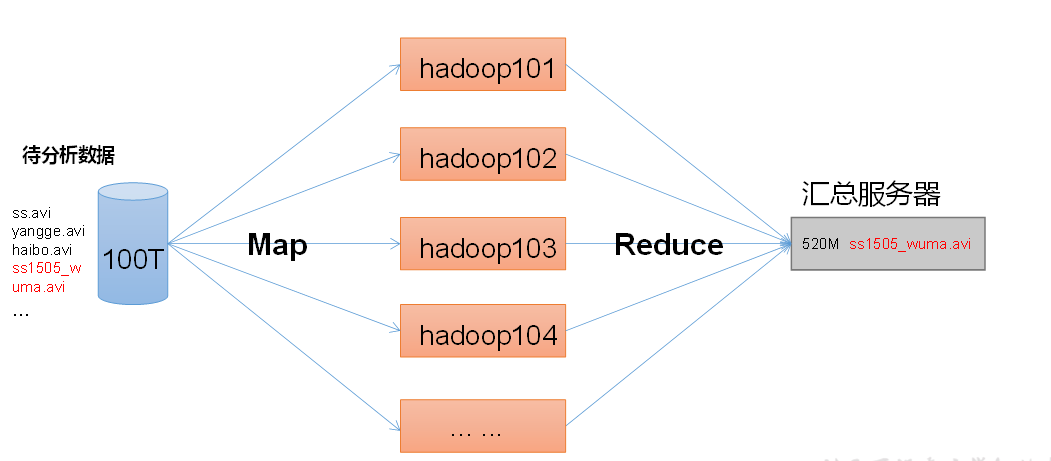
负责数据的切分、为应用程序申请资源并分配给内部的任务、任务的监控与容错

4）Container

Container是YARN中的资源抽象，它封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等

### 1.3.3 MapReduce架构概述

MapReduce将计算过程分为两个阶段：Map和Reduce，如图2-25所示



1）Map阶段并行处理输入数据；

2）Reduce阶段对Map结果进行汇总；

## 1.4 大数据技术生态体系

大数据技术生态体系如图2-26所示。



图中涉及的技术名词解释如下：

1）Sqoop：Sqoop是一款开源的工具，主要用于在Hadoop、Hive与传统的数据库(MySql)间进行数据的传递，可以将一个关系型数据库（例如：MySQL，Oracle等）中的数据导进到Hadoop的HDFS中，也可以将HDFS的数据导进到关系型数据库中。

2）Flume：Flume是一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，Flume支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

3）Kafka：Kafka是一种高吞吐量的分布式发布订阅消息系统，有如下特性：

（1）通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB的消息存储也能够保持长时间的稳定性能。

（2）高吞吐量：即使是非常普通的硬件Kafka也可以支持每秒数百万的消息。

（3）支持通过Kafka服务器和消费机集群来分区消息。

（4）支持Hadoop并行数据加载。

4）Storm：Storm用于“连续计算”，对数据流做连续查询，在计算时就将结果以流的形式输出给用户。

5）Spark：Spark是当前最流行的开源大数据内存计算框架。可以基于Hadoop上存储的大数据进行计算。

6）Oozie：Oozie是一个管理Hdoop作业（job）的工作流程调度管理系统。

7）Hbase：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。

8）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。

9）R语言：R是用于统计分析、绘图的语言和操作环境。R是属于GNU系统的一个自由、免费、源代码开放的软件，它是一个用于统计计算和统计制图的优秀工具。

10）Mahout：Apache Mahout是个可扩展的机器学习和数据挖掘库。

11）ZooKeeper：Zookeeper是Google的Chubby一个开源的实现。它是一个针对大型分布式系统的可靠协调系统，提供的功能包括：配置维护、名字服务、分布式同步、组服务等。ZooKeeper的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

## 1.5 推荐系统框架图

推荐系统项目架构如图2-27所示。

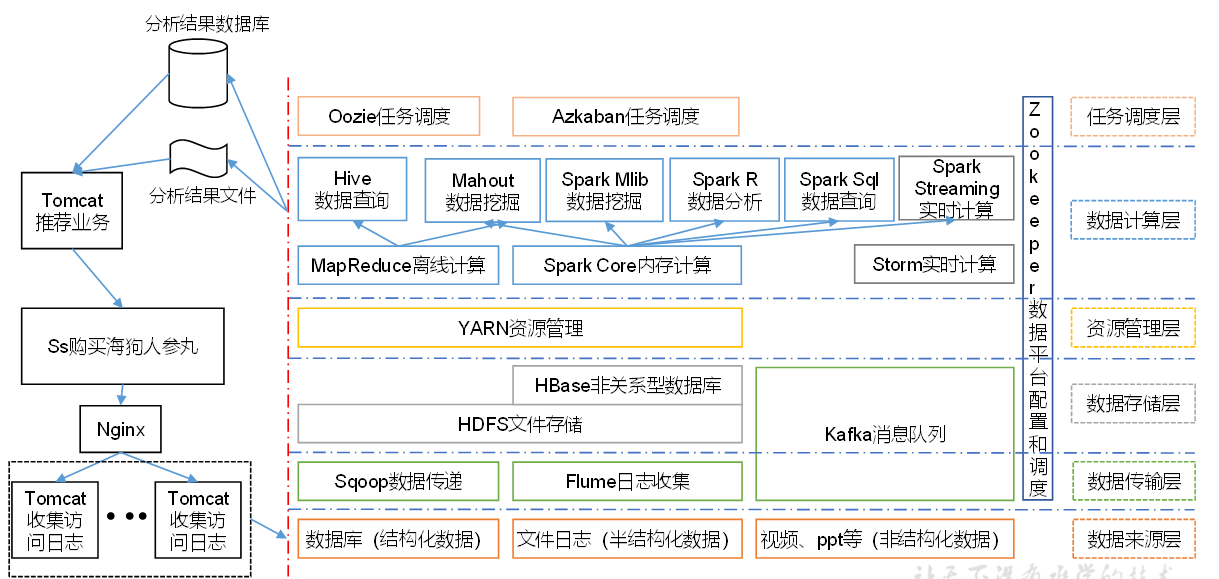


图2-27 推荐系统项目架构

# 第2章 Hadoop运行环境搭建（开发重点）

## 2.1 虚拟机环境准备

/etc/hosts

1. 克隆虚拟机

2. 修改克隆虚拟机的静态IP

3. 修改主机名

4. 关闭防火墙

5. 创建atguigu用户

6. 配置atguigu用户具有root权限（详见《尚硅谷大数据技术之Linux》）

7． 在/opt目录下创建文件夹

（1）在/opt目录下创建module、software文件夹

[atguigu@hadoop101 opt]$ sudo mkdir module

[atguigu@hadoop101 opt]$ sudo mkdir software

（2）修改module、software文件夹的所有者cd

[atguigu@hadoop101 opt]$ sudo chown atguigu:atguigu module/ software/

[atguigu@hadoop101 opt]$ ll

总用量 8

drwxr-xr-x. 2 atguigu atguigu 4096 1月 17 14:37 module

drwxr-xr-x. 2 atguigu atguigu 4096 1月 17 14:38 software

## 2.2 安装JDK

1. 卸载现有JDK

（1）查询是否安装Java软件：

[atguigu@hadoop101 opt]$ rpm -qa | grep java

（2）如果安装的版本低于1.7，卸载该JDK：

[atguigu@hadoop101 opt]$ sudo rpm -e 软件包

（3）查看JDK安装路径：

[atguigu@hadoop101 ~]$ which java

2. 用SecureCRT工具将JDK导入到opt目录下面的software文件夹下面，如图2-28所示

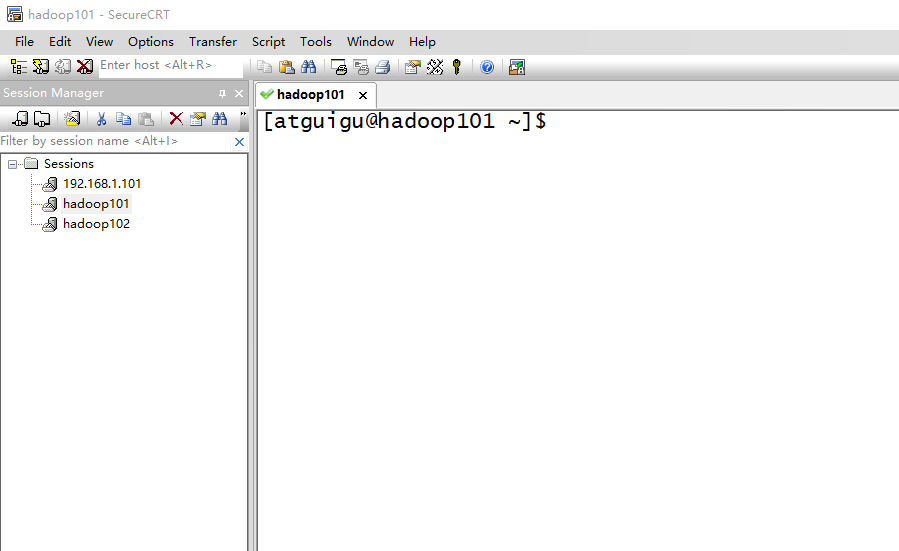


图2-28 导入JDK

“alt+p”进入sftp模式，如图2-29所示

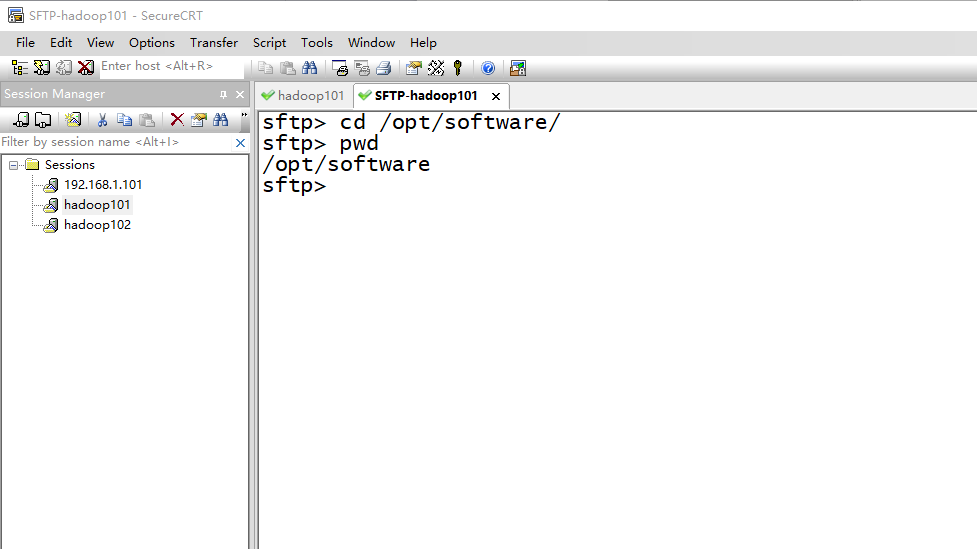


图2-29 进入sftp模式

选择jdk1.8拖入，如图2-30，2-31所示



图2-30 拖入jdk1.8

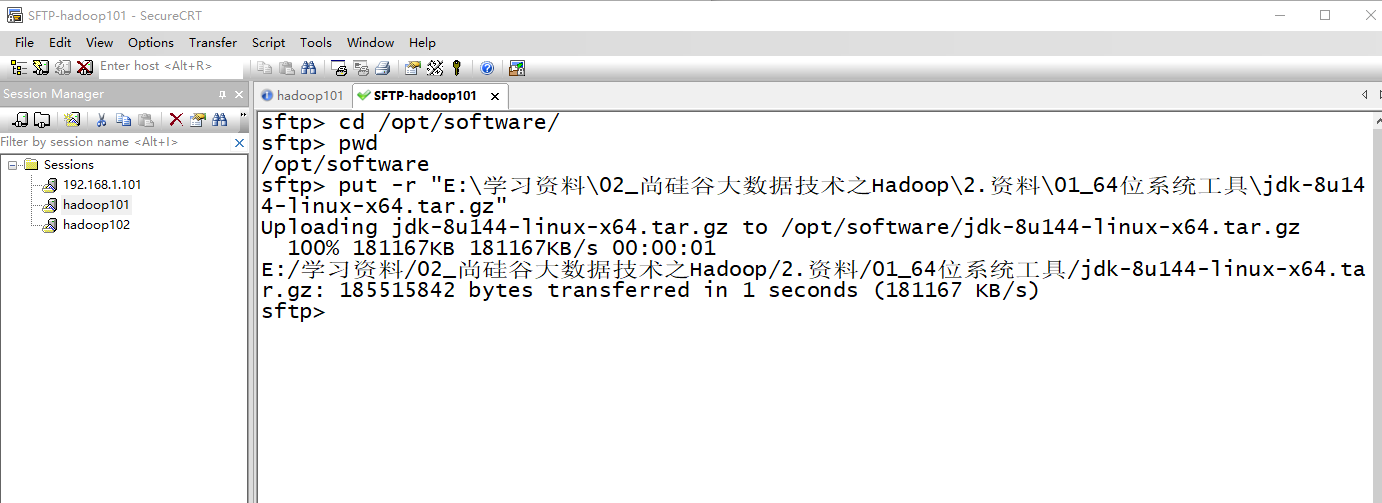


图2-31 拖入jdk1.8完成

1. 在Linux系统下的opt目录中查看软件包是否导入成功

[atguigu@hadoop101 opt]$ cd software/

[atguigu@hadoop101 software]$ ls

hadoop-2.7.2.tar.gz jdk-8u144-linux-x64.tar.gz

1. 解压JDK到/opt/module目录下

[atguigu@hadoop101 software]$ tar -zxvf jdk-8u144-linux-x64.tar.gz -C /opt/module/

5. 配置JDK环境变量

（1）先获取JDK路径

[atguigu@hadoop101 jdk1.8.0\_144]$ pwd

/opt/module/jdk1.8.0\_144

（2）打开/etc/profile文件

[atguigu@hadoop101 software]$ sudo vi /etc/profile

在profile文件末尾添加JDK路径

#JAVA\_HOME

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

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

（3）保存后退出

:wq

（4）让修改后的文件生效

[atguigu@hadoop101 jdk1.8.0\_144]$ source /etc/profile

6. 测试JDK是否安装成功

[atguigu@hadoop101 jdk1.8.0\_144]# java -version

java version "1.8.0\_144"

注意：重启（如果java -version可以用就不用重启）

[atguigu@hadoop101 jdk1.8.0\_144]$ sync

[atguigu@hadoop101 jdk1.8.0\_144]$ sudo reboot

## 2.3 安装Hadoop

0. Hadoop下载地址：

<https://archive.apache.org/dist/hadoop/common/hadoop-2.7.2/>

1. 用SecureCRT工具将hadoop-2.7.2.tar.gz导入到opt目录下面的software文件夹下面

切换到sftp连接页面，选择Linux下编译的hadoop jar包拖入，如图2-32所示



图2-32 拖入hadoop的tar包



图2-33 拖入Hadoop的tar包成功

1. 进入到Hadoop安装包路径下

[atguigu@hadoop101 ~]$ cd /opt/software/

1. 解压安装文件到/opt/module下面

[atguigu@hadoop101 software]$ tar -zxvf hadoop-2.7.2.tar.gz -C /opt/module/

1. 查看是否解压成功

[atguigu@hadoop101 software]$ ls /opt/module/

hadoop-2.7.2

5. 将Hadoop添加到环境变量

（1）获取Hadoop安装路径

[atguigu@hadoop101 hadoop-2.7.2]$ pwd

/opt/module/hadoop-2.7.2

（2）打开/etc/profile文件

[atguigu@hadoop101 hadoop-2.7.2]$ sudo vi /etc/profile

在profile文件末尾添加JDK路径：（shitf+g）

##HADOOP\_HOME

export HADOOP\_HOME=/opt/module/hadoop-2.7.2

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

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

（3）保存后退出

:wq

（4）让修改后的文件生效

[atguigu@ hadoop101 hadoop-2.7.2]$ source /etc/profile

6. 测试是否安装成功

[atguigu@hadoop101 hadoop-2.7.2]$ hadoop version

Hadoop 2.7.2

7. 重启(如果Hadoop命令不能用再重启)

[atguigu@ hadoop101 hadoop-2.7.2]$ sync

[atguigu@ hadoop101 hadoop-2.7.2]$ sudo reboot

## 2.4 Hadoop目录结构

1、查看Hadoop目录结构

[atguigu@hadoop101 hadoop-2.7.2]$ ll

总用量 52

drwxr-xr-x. 2 atguigu atguigu 4096 5月 22 2017 **bin**

drwxr-xr-x. 3 atguigu atguigu 4096 5月 22 2017 **etc**

drwxr-xr-x. 2 atguigu atguigu 4096 5月 22 2017 include

drwxr-xr-x. 3 atguigu atguigu 4096 5月 22 2017 **lib**

drwxr-xr-x. 2 atguigu atguigu 4096 5月 22 2017 libexec

-rw-r--r--. 1 atguigu atguigu 15429 5月 22 2017 LICENSE.txt

-rw-r--r--. 1 atguigu atguigu 101 5月 22 2017 NOTICE.txt

-rw-r--r--. 1 atguigu atguigu 1366 5月 22 2017 README.txt

drwxr-xr-x. 2 atguigu atguigu 4096 5月 22 2017 **sbin**

drwxr-xr-x. 4 atguigu atguigu 4096 5月 22 2017 **share**

2、重要目录

（1）bin目录：存放对Hadoop相关服务（HDFS,YARN）进行操作的脚本

（2）etc目录：Hadoop的配置文件目录，存放Hadoop的配置文件

（3）lib目录：存放Hadoop的本地库（对数据进行压缩解压缩功能）

（4）sbin目录：存放启动或停止Hadoop相关服务的脚本

（5）share目录：存放Hadoop的依赖jar包、文档、和官方案例

# 第3章 Hadoop运行模式

Hadoop运行模式包括：本地模式、伪分布式模式以及完全分布式模式。

Hadoop官方网站：<http://hadoop.apache.org/>

## 3.1 本地运行模式

### 3.1.1 官方Grep案例

1. 创建在hadoop-2.7.2文件下面创建一个input文件夹

[atguigu@hadoop101 hadoop-2.7.2]$ mkdir input

2. 将Hadoop的xml配置文件复制到input

[atguigu@hadoop101 hadoop-2.7.2]$ cp etc/hadoop/\*.xml input

3. 执行share目录下的MapReduce程序

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hadoop jar

share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar grep input output 'dfs[a-z.]+'

4. 查看输出结果

[atguigu@hadoop101 hadoop-2.7.2]$ cat output/\*

### 3.1.2 官方WordCount案例

1. 创建在hadoop-2.7.2文件下面创建一个wcinput文件夹

[atguigu@hadoop101 hadoop-2.7.2]$ mkdir wcinput

2. 在wcinput文件下创建一个wc.input文件

[atguigu@hadoop101 hadoop-2.7.2]$ cd wcinput

[atguigu@hadoop101 wcinput]$ touch wc.input

3. 编辑wc.input文件

[atguigu@hadoop101 wcinput]$ vi wc.input

在文件中输入如下内容

hadoop yarn

hadoop mapreduce

atguigu

atguigu

保存退出 **::**wq

4. 回到Hadoop目录/opt/module/hadoop-2.7.2

5. 执行程序

[atguigu@hadoop101 hadoop-2.7.2]$ hadoop jar

share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount wcinput wcoutput

6. 查看结果

[atguigu@hadoop101 hadoop-2.7.2]$ cat wcoutput/part-r-00000

atguigu 2

hadoop 2

mapreduce 1

yarn 1

## 3.2 伪分布式运行模式

### 3.2.1 启动HDFS并运行MapReduce程序

1. 分析

（1）配置集群

（2）启动、测试集群增、删、查

（3）执行WordCount案例

2. 执行步骤

（1）配置集群

（a）配置：hadoop-env.sh

Linux系统中获取JDK的安装路径：

[atguigu@ hadoop101 ~]# echo $JAVA\_HOME

/opt/module/jdk1.8.0\_144

修改JAVA\_HOME 路径：

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

（b）配置：core-site.xml

|  |
| --- |
| <!-- 指定HDFS中NameNode的地址 -->  <property>  <name>fs.defaultFS</name>  <value>hdfs://hadoop101:9000</value>  </property>  <!-- 指定Hadoop运行时产生文件的存储目录 -->  <property>  <name>hadoop.tmp.dir</name>  <value>/opt/module/hadoop-2.7.2/data/tmp</value>  </property> |

（c）配置：hdfs-site.xml

|  |
| --- |
| <!-- 指定HDFS副本的数量 -->  <property>  <name>dfs.replication</name>  <value>1</value>  </property> |

（2）启动集群

（a）**格式化NameNode**（第一次启动时格式化，以后就不要总格式化）

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs namenode -format

（b）启动NameNode

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/hadoop-daemon.sh start namenode

（c）启动DataNode

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/hadoop-daemon.sh start datanode

（3）查看集群

（a）查看是否启动成功

[atguigu@hadoop101 hadoop-2.7.2]$ jps

13586 NameNode

13668 DataNode

13786 Jps

注意：jps是JDK中的命令，不是Linux命令。不安装JDK不能使用jps

（b）web端查看HDFS文件系统

<http://hadoop101:50070/dfshealth.html#tab-overview>

注意：如果不能查看，看如下帖子处理

<http://www.cnblogs.com/zlslch/p/6604189.html>

（c）查看产生的Log日志

说明：在企业中遇到Bug时，经常根据日志提示信息去分析问题、解决Bug。

当前目录：/opt/module/hadoop-2.7.2/logs

[atguigu@hadoop101 logs]$ ls

hadoop-atguigu-datanode-hadoop.atguigu.com.log

hadoop-atguigu-datanode-hadoop.atguigu.com.out

hadoop-atguigu-namenode-hadoop.atguigu.com.log

hadoop-atguigu-namenode-hadoop.atguigu.com.out

SecurityAuth-root.audit

[atguigu@hadoop101 logs]# cat hadoop-atguigu-datanode-hadoop101.log

**（d）思考：为什么不能一直格式化NameNode，格式化NameNode，要注意什么？**

[atguigu@hadoop101 hadoop-2.7.2]$ cd data/tmp/dfs/name/current/

[atguigu@hadoop101 current]$ cat VERSION

**clusterID=CID-f0330a58-36fa-4a2a-a65f-2688269b5837**

[atguigu@hadoop101 hadoop-2.7.2]$ cd data/tmp/dfs/data/current/

**clusterID=CID-f0330a58-36fa-4a2a-a65f-2688269b5837**

注意：格式化NameNode，会产生新的集群id,导致NameNode和DataNode的集群id不一致，集群找不到已往数据。所以，格式NameNode时，一定要先删除data数据和log日志，然后再格式化NameNode。

（4）操作集群

（a）在HDFS文件系统上**创建**一个input文件夹

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs dfs -mkdir -p /user/atguigu/input

（b）将测试文件内容**上传**到文件系统上

[atguigu@hadoop101 hadoop-2.7.2]$bin/hdfs dfs -put wcinput/wc.input

/user/atguigu/input/

（c）**查看**上传的文件是否正确

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs dfs -ls /user/atguigu/input/

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs dfs -cat /user/atguigu/ input/wc.input

（d）运行MapReduce程序

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hadoop jar

share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount /user/atguigu/input/ /user/atguigu/output

（e）查看输出结果

命令行查看：

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs dfs -cat /user/atguigu/output/\*

浏览器查看，如图2-34所示

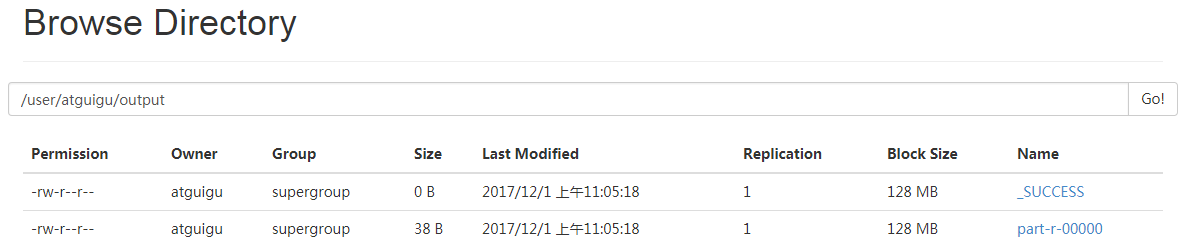


图2-34 查看output文件

（f）将测试文件内容**下载**到本地

[atguigu@hadoop101 hadoop-2.7.2]$ hdfs dfs -get /user/atguigu/output/part-r-00000 ./wcoutput/

（g）**删除**输出结果

[atguigu@hadoop101 hadoop-2.7.2]$ hdfs dfs -rm -r /user/atguigu/output

### 4.2.2 启动YARN并运行MapReduce程序

1. 分析

（1）配置集群在YARN上运行MR

（2）启动、测试集群增、删、查

（3）在YARN上执行WordCount案例

2. 执行步骤

（1）配置集群

（a）配置yarn-env.sh

配置一下JAVA\_HOME

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

（b）配置yarn-site.xml

<!-- Reducer获取数据的方式 -->

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<!-- 指定YARN的ResourceManager的地址 -->

<property>

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

<value>hadoop101</value>

</property>

（c）配置：mapred-env.sh

配置一下JAVA\_HOME

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

（d）配置： (对mapred-site.xml.template重新命名为) mapred-site.xml

[atguigu@hadoop101 hadoop]$ mv mapred-site.xml.template mapred-site.xml

[atguigu@hadoop101 hadoop]$ vi mapred-site.xml

<!-- 指定MR运行在YARN上 -->

<property>

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

<value>yarn</value>

</property>

（2）启动集群

（a）启动前必须保证NameNode和DataNode已经启动

（b）启动ResourceManager

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/yarn-daemon.sh start resourcemanager

（c）启动NodeManager

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/yarn-daemon.sh start nodemanager

（3）集群操作

（a）YARN的浏览器页面查看，如图2-35所示

http://hadoop101:8088/cluster

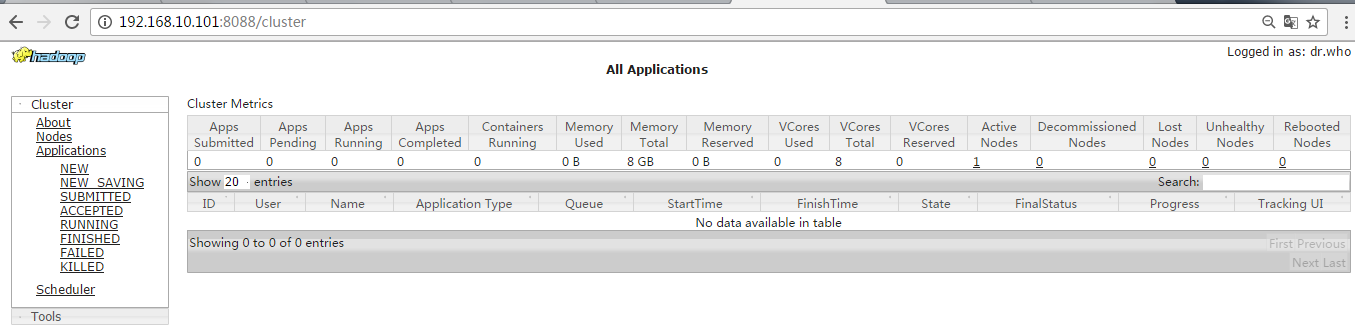


图2-35 YARN的浏览器页面

（b）删除文件系统上的output文件

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs dfs -rm -R /user/atguigu/output

（c）执行MapReduce程序

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hadoop jar

share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount /user/atguigu/input /user/atguigu/output

（d）查看运行结果，如图2-36所示

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs dfs -cat /user/atguigu/output/\*

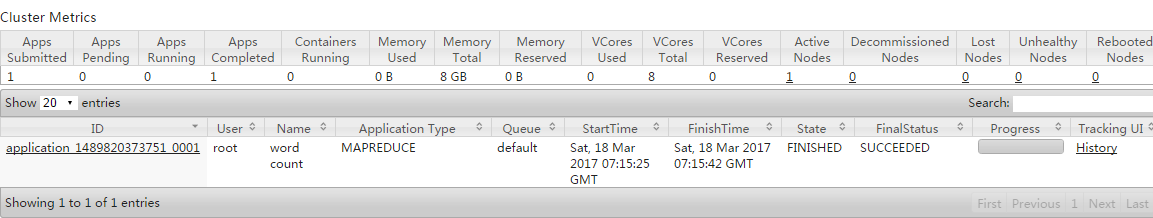


图2-36 查看运行结果

### 3.2.3 配置历史服务器

为了查看程序的历史运行情况，需要配置一下历史服务器。具体配置步骤如下：

1. 配置mapred-site.xml

[atguigu@hadoop101 hadoop]$ vi mapred-site.xml

在该文件里面增加如下配置。

<!-- 历史服务器端地址 -->

<property>

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

<value>hadoop101:10020</value>

</property>

<!-- 历史服务器web端地址 -->

<property>

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

<value>hadoop101:19888</value>

</property>

2. 启动历史服务器

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/mr-jobhistory-daemon.sh start historyserver

3. 查看历史服务器是否启动

[atguigu@hadoop101 hadoop-2.7.2]$ jps

4. 查看JobHistory

<http://hadoop101:19888/jobhistory>

### 3.2.4 配置日志的聚集

日志聚集概念：应用运行完成以后，将程序运行日志信息上传到HDFS系统上。

日志聚集功能好处：可以方便的查看到程序运行详情，方便开发调试。

注意：开启日志聚集功能，需要重新启动NodeManager 、ResourceManager和HistoryManager。

开启日志聚集功能具体步骤如下：

1. 配置yarn-site.xml

[atguigu@hadoop101 hadoop]$ vi yarn-site.xml

在该文件里面增加如下配置。

<!-- 日志聚集功能使能 -->

<property>

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

<value>true</value>

</property>

<!-- 日志保留时间设置7天 -->

<property>

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

<value>604800</value>

</property>

1. 关闭NodeManager 、ResourceManager和HistoryServer

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/yarn-daemon.sh stop resourcemanager

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/yarn-daemon.sh stop nodemanager

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/mr-jobhistory-daemon.sh stop historyserver

1. 启动NodeManager 、ResourceManager和HistoryServer

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/yarn-daemon.sh start resourcemanager

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/yarn-daemon.sh start nodemanager

[atguigu@hadoop101 hadoop-2.7.2]$ sbin/mr-jobhistory-daemon.sh start historyserver

1. 删除HDFS上已经存在的输出文件

[atguigu@hadoop101 hadoop-2.7.2]$ bin/hdfs dfs -rm -R /user/atguigu/output

1. 执行WordCount程序

[atguigu@hadoop101 hadoop-2.7.2]$ hadoop jar

share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar wordcount /user/atguigu/input /user/atguigu/output

1. 查看日志，如图2-37，2-38，2-39所示

<http://hadoop101:19888/jobhistory>

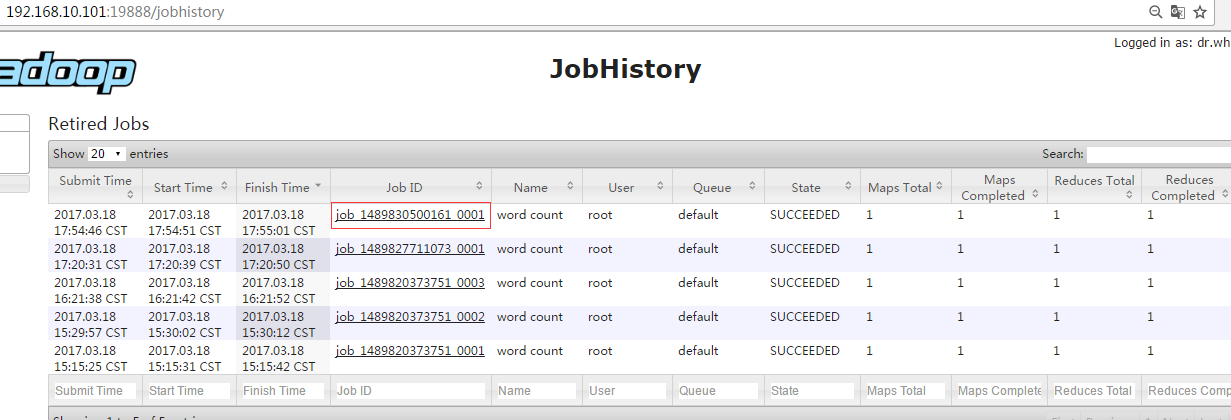


图2-37 Job History

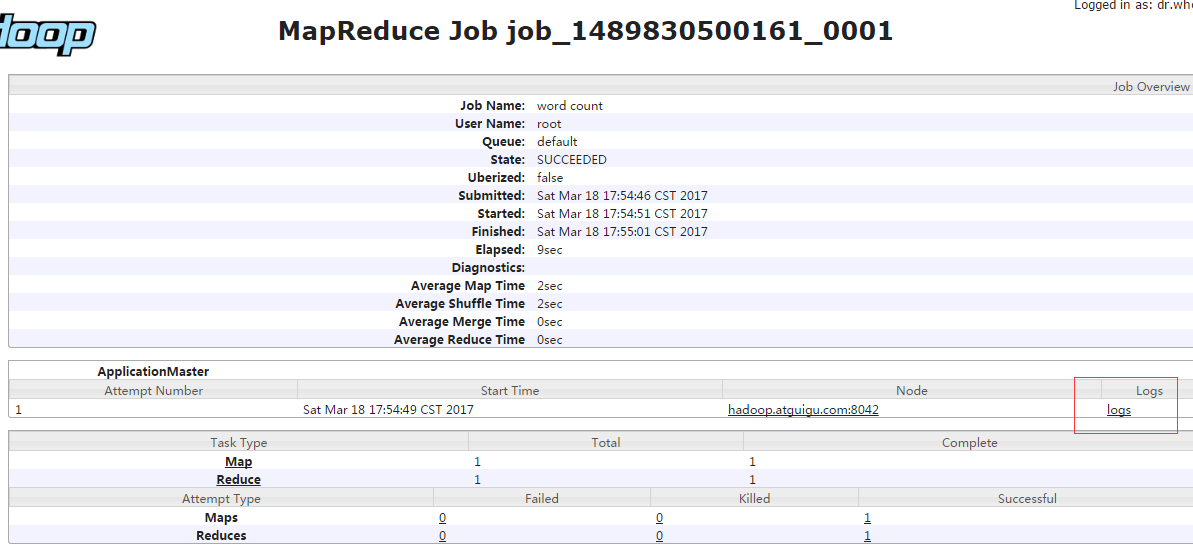


图2-38 job运行情况



图2-39 查看日志

### 3.2.5 配置文件说明

Hadoop配置文件分两类：默认配置文件和自定义配置文件，只有用户想修改某一默认配置值时，才需要修改自定义配置文件，更改相应属性值。

（1）默认配置文件：

表2-1

|  |  |
| --- | --- |
| 要获取的默认文件 | 文件存放在Hadoop的jar包中的位置 |
| [core-default.xml] | hadoop-common-2.7.2.jar/ core-default.xml |
| [hdfs-default.xml] | hadoop-hdfs-2.7.2.jar/ hdfs-default.xml |
| [yarn-default.xml] | hadoop-yarn-common-2.7.2.jar/ yarn-default.xml |
| [mapred-default.xml] | hadoop-mapreduce-client-core-2.7.2.jar/ mapred-default.xml |

（2）自定义配置文件：

**core-site.xml、hdfs-site.xml、yarn-site.xml、mapred-site.xml**四个配置文件存放在$HADOOP\_HOME/etc/hadoop这个路径上，用户可以根据项目需求重新进行修改配置。

## 3.3 完全分布式运行模式（开发重点）

分析：

1）准备3台客户机（关闭防火墙、静态ip、主机名称）

2）安装JDK

3）配置环境变量

4）安装Hadoop

5）配置环境变量

6）配置集群

7）单点启动

8）配置ssh

9）群起并测试集群

### 3.3.1 虚拟机准备

详见3.1章。

### 3.3.2 编写集群分发脚本xsync

1. scp（secure copy）安全拷贝

（1）scp定义：

scp可以实现服务器与服务器之间的数据拷贝。（from server1 to server2）

（2）基本语法

scp -r $pdir/$fname $user@hadoop$host:$pdir/$fname

命令 递归 要拷贝的文件路径/名称 目的用户@主机:目的路径/名称

（3）案例实操

（a）在hadoop101上，将hadoop101中/opt/module目录下的软件拷贝到hadoop102上。

[atguigu@hadoop101 /]$ scp -r /opt/module root@hadoop102:/opt/module

（b）在hadoop103上，将hadoop101服务器上的/opt/module目录下的软件拷贝到hadoop103上。

[atguigu@hadoop103 opt]$sudo scp -r atguigu@hadoop101:/opt/module root@hadoop103:/opt/module

（c）在hadoop103上操作将hadoop101中/opt/module目录下的软件拷贝到hadoop104上。

[atguigu@hadoop103 opt]$ scp -r atguigu@hadoop101:/opt/module root@hadoop104:/opt/module

注意：拷贝过来的/opt/module目录，别忘了在hadoop102、hadoop103、hadoop104上修改所有文件的，所有者和所有者组。sudo chown atguigu:atguigu -R /opt/module

（d）将hadoop101中/etc/profile文件拷贝到hadoop102的/etc/profile上。

[atguigu@hadoop101 ~]$ sudo scp /etc/profile root@hadoop102:/etc/profile

（e）将hadoop101中/etc/profile文件拷贝到hadoop103的/etc/profile上。

[atguigu@hadoop101 ~]$ sudo scp /etc/profile root@hadoop103:/etc/profile

（f）将hadoop101中/etc/profile文件拷贝到hadoop104的/etc/profile上。

[atguigu@hadoop101 ~]$ sudo scp /etc/profile root@hadoop104:/etc/profile

注意：拷贝过来的配置文件别忘了source一下/etc/profile。

2. rsync远程同步工具

rsync主要用于备份和镜像。具有速度快、避免复制相同内容和支持符号链接的优点。

rsync和scp区别：用rsync做文件的复制要比scp的速度快，rsync只对差异文件做更新。scp是把所有文件都复制过去。

（1）基本语法

rsync -av $pdir/$fname $user@hadoop$host:$pdir/$fname

命令 选项参数 要拷贝的文件路径/名称 目的用户@主机:目的路径/名称

选项参数说明

表2-2

|  |  |
| --- | --- |
| 选项 | 功能 |
| -a | 归档拷贝 |
| -v | 显示复制过程 |

（2）案例实操

（a）把hadoop101机器上的/opt/software目录同步到hadoop102服务器的root用户下的/opt/目录

[atguigu@hadoop101 opt]$ rsync -av /opt/software/ hadoop102:/opt/software

3. xsync集群分发脚本

（1）需求：循环复制文件到所有节点的相同目录下

（2）需求分析：

（a）rsync命令原始拷贝：

rsync -av /opt/module root@hadoop103:/opt/

（b）期望脚本：

xsync要同步的文件名称

（c）说明：在/home/atguigu/bin这个目录下存放的脚本，atguigu用户可以在系统任何地方直接执行。

（3）脚本实现

（a）在/home/atguigu目录下创建bin目录，并在bin目录下xsync创建文件，文件内容如下：

[atguigu@hadoop102 ~]$ mkdir bin

[atguigu@hadoop102 ~]$ cd bin/

[atguigu@hadoop102 bin]$ touch xsync

[atguigu@hadoop102 bin]$ vi xsync

在该文件中编写如下代码

#!/bin/bash

#1 获取输入参数个数，如果没有参数，直接退出

pcount=$#

if ((pcount==0)); then

echo no args;

exit;

fi

#2 获取文件名称

p1=$1

fname=`basename $p1`

echo fname=$fname

#3 获取上级目录到绝对路径

pdir=`cd -P $(dirname $p1); pwd`

echo pdir=$pdir

#4 获取当前用户名称

user=`whoami`

#5 循环

for((host=103; host<105; host++)); do

echo ------------------- hadoop$host --------------

rsync -av $pdir/$fname $user@hadoop$host:$pdir

done

（b）修改脚本 xsync 具有执行权限

[atguigu@hadoop102 bin]$ chmod 777 xsync

（c）调用脚本形式：xsync 文件名称

[atguigu@hadoop102 bin]$ xsync /home/atguigu/bin

注意：如果将xsync放到/home/atguigu/bin目录下仍然不能实现全局使用，可以将xsync移动到/usr/local/bin目录下。

### 3.3.3 集群配置

1. 集群部署规划

表2-3

|  |  |  |  |
| --- | --- | --- | --- |
|  | hadoop102 | hadoop103 | hadoop104 |
| HDFS | NameNode  DataNode | DataNode | SecondaryNameNode  DataNode |
| YARN | NodeManager | ResourceManager  NodeManager | NodeManager |

2. 配置集群

（1）核心配置文件

配置core-site.xml

[atguigu@hadoop102 hadoop]$ vi core-site.xml

在该文件中编写如下配置

<!-- 指定HDFS中NameNode的地址 -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://hadoop102:9000</value>

</property>

<!-- 指定Hadoop运行时产生文件的存储目录 -->

<property>

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

<value>/opt/module/hadoop-2.7.2/data/tmp</value>

</property>

（2）HDFS配置文件

配置hadoop-env.sh

[atguigu@hadoop102 hadoop]$ vi hadoop-env.sh

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

配置hdfs-site.xml

[atguigu@hadoop102 hadoop]$ vi hdfs-site.xml

在该文件中编写如下配置

<property>

<name>dfs.replication</name>

<value>3</value>

</property>

<!-- 指定Hadoop辅助名称节点主机配置 -->

<property>

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

<value>hadoop104:50090</value>

</property>

（3）YARN配置文件

配置yarn-env.sh

[atguigu@hadoop102 hadoop]$ vi yarn-env.sh

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

配置yarn-site.xml

[atguigu@hadoop102 hadoop]$ vi yarn-site.xml

在该文件中增加如下配置

<!-- Reducer获取数据的方式 -->

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<!-- 指定YARN的ResourceManager的地址 -->

<property>

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

<value>hadoop103</value>

</property>

（4）MapReduce配置文件

配置mapred-env.sh

[atguigu@hadoop102 hadoop]$ vi mapred-env.sh

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

配置mapred-site.xml

[atguigu@hadoop102 hadoop]$ cp mapred-site.xml.template mapred-site.xml

[atguigu@hadoop102 hadoop]$ vi mapred-site.xml

在该文件中增加如下配置

<!-- 指定MR运行在Yarn上 -->

<property>

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

<value>yarn</value>

</property>

3．在集群上分发配置好的Hadoop配置文件

[atguigu@hadoop102 hadoop]$ xsync /opt/module/hadoop-2.7.2/

4．查看文件分发情况

[atguigu@hadoop103 hadoop]$ cat /opt/module/hadoop-2.7.2/etc/hadoop/core-site.xml

### 3.3.4 集群单点启动

（1）如果集群是第一次启动，需要**格式化NameNode**

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs namenode -format

（2）在hadoop102上启动NameNode

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop-daemon.sh start namenode

[atguigu@hadoop102 hadoop-2.7.2]$ jps

3461 NameNode

（3）在hadoop102、hadoop103以及hadoop104上分别启动DataNode

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop-daemon.sh start datanode

[atguigu@hadoop102 hadoop-2.7.2]$ jps

3461 NameNode

3608 Jps

3561 DataNode

[atguigu@hadoop103 hadoop-2.7.2]$ hadoop-daemon.sh start datanode

[atguigu@hadoop103 hadoop-2.7.2]$ jps

3190 DataNode

3279 Jps

[atguigu@hadoop104 hadoop-2.7.2]$ hadoop-daemon.sh start datanode

[atguigu@hadoop104 hadoop-2.7.2]$ jps

3237 Jps

3163 DataNode

（4）思考：每次都一个一个节点启动，如果节点数增加到1000个怎么办？

早上来了开始一个一个节点启动，到晚上下班刚好完成，下班？

### 3.3.5 SSH无密登录配置

1. 配置ssh

（1）基本语法

ssh另一台电脑的ip地址

（2）ssh连接时出现Host key verification failed的解决方法

[atguigu@hadoop102 opt] $ ssh 192.168.1.103

The authenticity of host '192.168.1.103 (192.168.1.103)' can't be established.

RSA key fingerprint is cf:1e:de:d7:d0:4c:2d:98:60:b4:fd:ae:b1:2d:ad:06.

Are you sure you want to continue connecting (yes/no)?

Host key verification failed.

（3）解决方案如下：直接输入yes

2. 无密钥配置

（1）免密登录原理，如图2-40所示

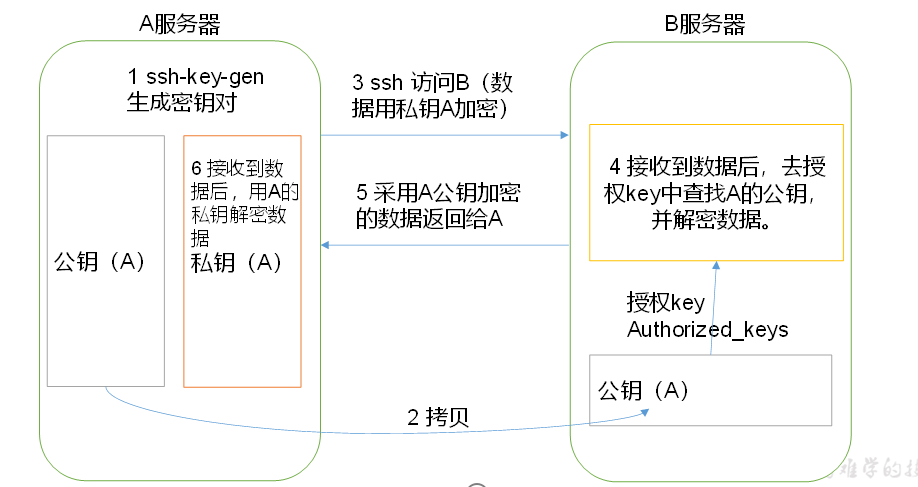


图2-40 免密登陆原理

（2）生成公钥和私钥：

[atguigu@hadoop102 .ssh]$ ssh-keygen -t rsa

然后敲（三个回车），就会生成两个文件id\_rsa（私钥）、id\_rsa.pub（公钥）

（3）将公钥拷贝到要免密登录的目标机器上

[atguigu@hadoop102 .ssh]$ ssh-copy-id hadoop102

[atguigu@hadoop102 .ssh]$ ssh-copy-id hadoop103

[atguigu@hadoop102 .ssh]$ ssh-copy-id hadoop104

注意：

还需要在hadoop102上采用root账号，配置一下无密登录到hadoop102、hadoop103、hadoop104；

还需要在hadoop103上采用atguigu账号配置一下无密登录到hadoop102、hadoop103、hadoop104服务器上。

3. .ssh文件夹下（~/.ssh）的文件功能解释

表2-4

|  |  |
| --- | --- |
| known\_hosts | 记录ssh访问过计算机的公钥(public key) |
| id\_rsa | 生成的私钥 |
| id\_rsa.pub | 生成的公钥 |
| authorized\_keys | 存放授权过得无密登录服务器公钥 |

### 3.3.6 群起集群

1. 配置slaves

/opt/module/hadoop-2.7.2/etc/hadoop/slaves

[atguigu@hadoop102 hadoop]$ vi slaves

在该文件中增加如下内容：

hadoop102

hadoop103

hadoop104

注意：该文件中添加的内容结尾不允许有空格，文件中不允许有空行。

同步所有节点配置文件

[atguigu@hadoop102 hadoop]$ xsync slaves

2. 启动集群

（1）如果集群是第一次启动，需要格式化NameNode（注意格式化之前，一定要先停止上次启动的所有namenode和datanode进程，然后再删除data和log数据）

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hdfs namenode -format

（2）启动HDFS

[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh

[atguigu@hadoop102 hadoop-2.7.2]$ jps

4166 NameNode

4482 Jps

4263 DataNode

[atguigu@hadoop103 hadoop-2.7.2]$ jps

3218 DataNode

3288 Jps

[atguigu@hadoop104 hadoop-2.7.2]$ jps

3221 DataNode

3283 SecondaryNameNode

3364 Jps

（3）启动YARN

[atguigu@hadoop103 hadoop-2.7.2]$ sbin/start-yarn.sh

注意：NameNode和ResourceManger如果不是同一台机器，不能在NameNode上启动 YARN，应该在ResouceManager所在的机器上启动YARN。

（4）Web端查看SecondaryNameNode

（a）浏览器中输入：<http://hadoop104:50090/status.html>

（b）查看SecondaryNameNode信息，如图2-41所示。

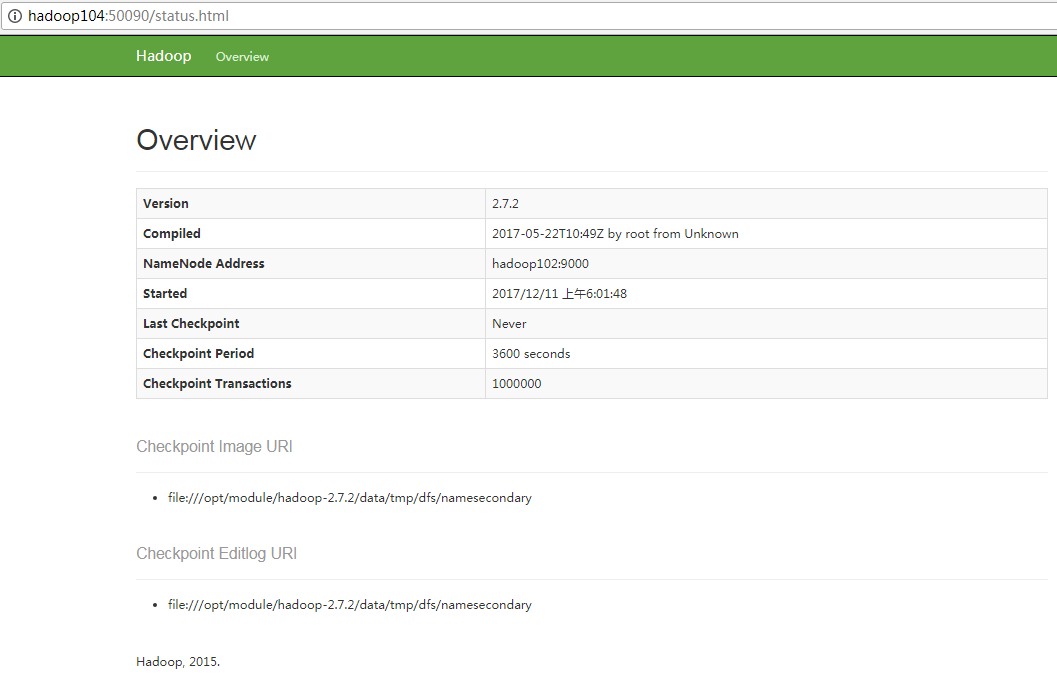


图2-41 SecondaryNameNode的Web端

3. 集群基本测试

（1）上传文件到集群

上传小文件

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -mkdir -p /user/atguigu/input

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfs -put wcinput/wc.input /user/atguigu/input

上传大文件

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs -put

/opt/software/hadoop-2.7.2.tar.gz /user/atguigu/input

（2）上传文件后查看文件存放在什么位置

（a）查看HDFS文件存储路径

[atguigu@hadoop102 subdir0]$ pwd

/opt/module/hadoop-2.7.2/data/tmp/dfs/data/current/BP-938951106-192.168.10.107-1495462844069/current/finalized/subdir0/subdir0

（b）查看HDFS在磁盘存储文件内容

[atguigu@hadoop102 subdir0]$ cat blk\_1073741825

hadoop yarn

hadoop mapreduce

atguigu

atguigu

（3）拼接

-rw-rw-r--. 1 atguigu atguigu 134217728 5月 23 16:01 **blk\_1073741836**

-rw-rw-r--. 1 atguigu atguigu 1048583 5月 23 16:01 blk\_1073741836\_1012.meta

-rw-rw-r--. 1 atguigu atguigu 63439959 5月 23 16:01 **blk\_1073741837**

-rw-rw-r--. 1 atguigu atguigu 495635 5月 23 16:01 blk\_1073741837\_1013.meta

[atguigu@hadoop102 subdir0]$ cat blk\_1073741836>>tmp.file

[atguigu@hadoop102 subdir0]$ cat blk\_1073741837>>tmp.file

[atguigu@hadoop102 subdir0]$ tar -zxvf tmp.file

（4）下载

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs -get

/user/atguigu/input/hadoop-2.7.2.tar.gz ./

### 3.3.7 集群启动/停止方式总结

1. 各个服务组件逐一启动/停止

（1）分别启动/停止HDFS组件

hadoop-daemon.sh start / stop namenode / datanode / secondarynamenode

（2）启动/停止YARN

yarn-daemon.sh start / stop resourcemanager / nodemanager

2. 各个模块分开启动/停止（配置ssh是前提）常用

（1）整体启动/停止HDFS

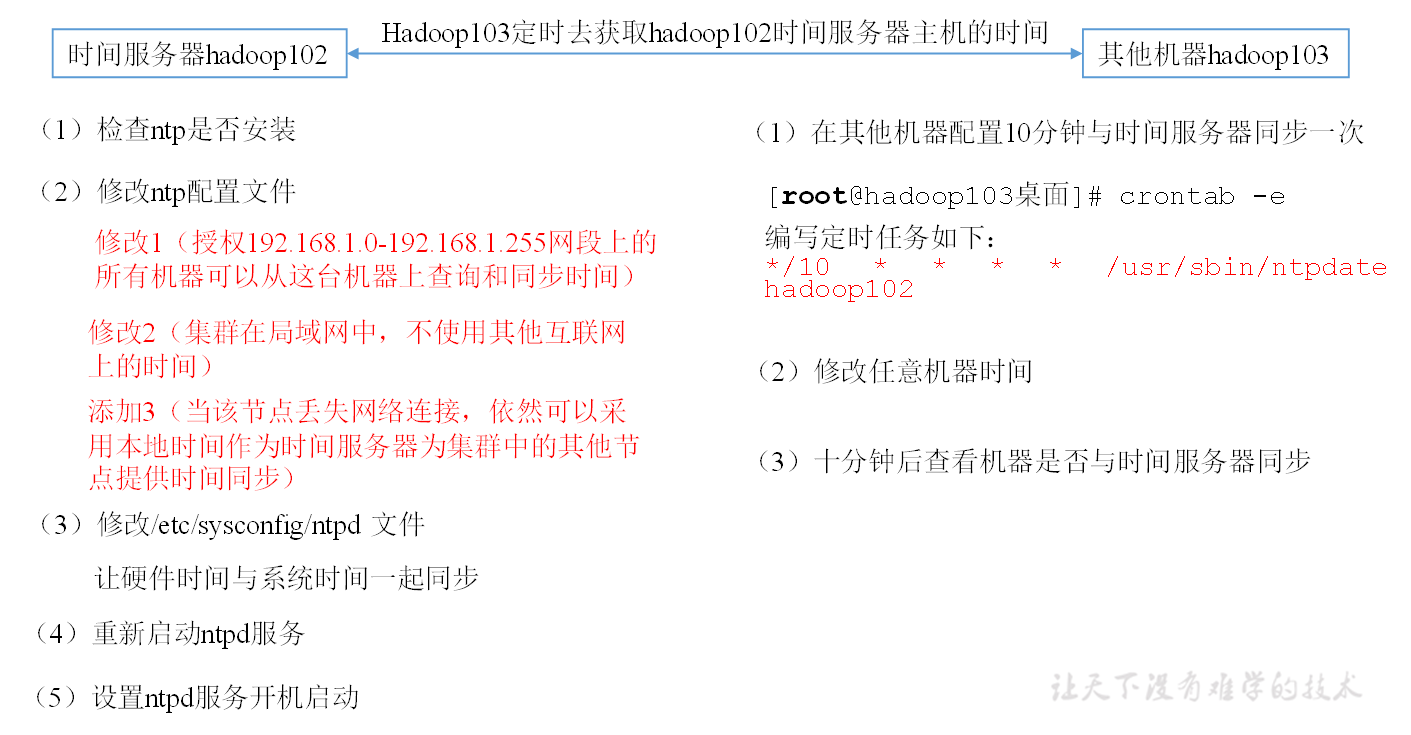
start-dfs.sh / stop-dfs.sh

（2）整体启动/停止YARN

start-yarn.sh / stop-yarn.sh

### 3.3.8 集群时间同步

时间同步的方式：找一个机器，作为时间服务器，所有的机器与这台集群时间进行定时的同步，比如，每隔十分钟，同步一次时间。



**配置时间同步具体实操：**

1. 时间服务器配置（必须root用户）

（1）检查ntp是否安装

[**root**@hadoop102 桌面]# rpm -qa|grep ntp

ntp-4.2.6p5-10.el6.centos.x86\_64

fontpackages-filesystem-1.41-1.1.el6.noarch

ntpdate-4.2.6p5-10.el6.centos.x86\_64

（2）修改ntp配置文件

[**root**@hadoop102 桌面]# vi /etc/ntp.conf

修改内容如下

a）修改1（授权192.168.1.0-192.168.1.255网段上的所有机器可以从这台机器上查询和同步时间）

**#**restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap为

restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap

b）修改2（集群在局域网中，不使用其他互联网上的时间）

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为

**#**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

c）添加3（当该节点丢失网络连接，依然可以采用本地时间作为时间服务器为集群中的其他节点提供时间同步）

server 127.127.1.0

fudge 127.127.1.0 stratum 10

（3）修改/etc/sysconfig/ntpd 文件

[**root**@hadoop102 桌面]# vim /etc/sysconfig/ntpd

增加内容如下（让硬件时间与系统时间一起同步）

SYNC\_HWCLOCK=yes

（4）重新启动ntpd服务

[**root**@hadoop102 桌面]# service ntpd status

ntpd 已停

[**root**@hadoop102 桌面]# service ntpd start

正在启动 ntpd： [确定]

（5）设置ntpd服务开机启动

[**root**@hadoop102 桌面]# chkconfig ntpd on

2. 其他机器配置（必须root用户）

（1）在其他机器配置10分钟与时间服务器同步一次

[**root**@hadoop103桌面]# crontab -e

编写定时任务如下：

\*/10 \* \* \* \* /usr/sbin/ntpdate hadoop102

（2）修改任意机器时间

[**root**@hadoop103桌面]# date -s "2017-9-11 11:11:11"

（3）十分钟后查看机器是否与时间服务器同步

[**root**@hadoop103桌面]# date

说明：测试的时候可以将10分钟调整为1分钟，节省时间。

# 第4章 Hadoop编译源码（面试重点）

为什么要编译源码？简单来说，hadoop是用java写的，但是某些操作不适合用java实现，所以用的是c/c++的动态库（本地库），所以需要根据不同的处理器架构，重新编译，他们以库的形式提供接口供上层调用。

## 4.1 准备工作

1. CentOS联网

配置CentOS能连接外网。Linux虚拟机ping [www.baidu.com](http://www.baidu.com) 是畅通的

注意：采用root角色编译，减少文件夹权限出现问题

2. jar包准备(hadoop源码、JDK8、maven、ant 、protobuf)

（1）hadoop-2.7.2-src.tar.gz

（2）jdk-8u144-linux-x64.tar.gz

（3）apache-ant-1.9.9-bin.tar.gz（build工具，打包用的）

（4）apache-maven-3.0.5-bin.tar.gz

（5）protobuf-2.5.0.tar.gz（序列化的框架）

## 4.2 jar包安装

注意：所有操作必须在root用户下完成

1. JDK解压、配置环境变量 JAVA\_HOME和PATH，验证[java](http://lib.csdn.net/base/javase" \t "_blank" \o "Java SE知识库)-version(如下都需要验证是否配置成功)

[root@hadoop101 software] # tar -zxf jdk-8u144-linux-x64.tar.gz -C /opt/module/

[root@hadoop101 software]# vi /etc/profile

#JAVA\_HOME：

export JAVA\_HOME=/opt/module/jdk1.8.0\_144

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

[root@hadoop101 software]#source /etc/profile

**验证命令：java -version**

1. Maven解压、配置  MAVEN\_HOME和PATH

[root@hadoop101 software]# tar -zxvf apache-maven-3.0.5-bin.tar.gz -C /opt/module/

[root@hadoop101 apache-maven-3.0.5]# vi conf/settings.xml

<mirrors>

<!-- mirror

| Specifies a repository mirror site to use instead of a given repository. The repository that

| this mirror serves has an ID that matches the mirrorOf element of this mirror. IDs are used

| for inheritance and direct lookup purposes, and must be unique across the set of mirrors.

|

<mirror>

<id>mirrorId</id>

<mirrorOf>repositoryId</mirrorOf>

<name>Human Readable Name for this Mirror.</name>

<url>http://my.repository.com/repo/path</url>

</mirror>

-->

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>central</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>

</mirrors>

[root@hadoop101 apache-maven-3.0.5]# vi /etc/profile

#MAVEN\_HOME

export MAVEN\_HOME=/opt/module/apache-maven-3.0.5

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

[root@hadoop101 software]#source /etc/profile

**验证命令：mvn -version**

1. ant解压、配置ANT \_HOME和PATH

[root@hadoop101 software]# tar -zxvf apache-ant-1.9.9-bin.tar.gz -C /opt/module/

[root@hadoop101 apache-ant-1.9.9]# vi /etc/profile

#ANT\_HOME

export ANT\_HOME=/opt/module/apache-ant-1.9.9

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

[root@hadoop101 software]#source /etc/profile

**验证命令：ant -version**

1. 安装glibc-headers和g++命令如下

[root@hadoop101 apache-ant-1.9.9]# yum install glibc-headers

[root@hadoop101 apache-ant-1.9.9]# yum install gcc-c++

1. 安装make和cmake

[root@hadoop101 apache-ant-1.9.9]# yum install make

[root@hadoop101 apache-ant-1.9.9]# yum install cmake

1. 解压protobuf ，进入到解压后protobuf主目录，/opt/module/protobuf-2.5.0，然后相继执行命令

[root@hadoop101 software]# tar -zxvf protobuf-2.5.0.tar.gz -C /opt/module/

[root@hadoop101 opt]# cd /opt/module/protobuf-2.5.0/

[root@hadoop101 protobuf-2.5.0]#./configure

[root@hadoop101 protobuf-2.5.0]# make

[root@hadoop101 protobuf-2.5.0]# make check

[root@hadoop101 protobuf-2.5.0]# make install

[root@hadoop101 protobuf-2.5.0]# ldconfig

[root@hadoop101 hadoop-dist]# vi /etc/profile

#LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=/opt/module/protobuf-2.5.0

export PATH=$PATH:$LD\_LIBRARY\_PATH

[root@hadoop101 software]#source /etc/profile

验证命令：protoc --version

1. 安装openssl库

[root@hadoop101 software]#yum install openssl-devel

1. 安装 ncurses-devel库

[root@hadoop101 software]#yum install ncurses-devel

到此，编译工具安装基本完成。

## 4.3 编译源码

1. 解压源码到/opt/目录

[root@hadoop101 software]# tar -zxvf hadoop-2.7.2-src.tar.gz -C /opt/

2. 进入到hadoop源码主目录

[root@hadoop101 hadoop-2.7.2-src]# pwd

/opt/hadoop-2.7.2-src

3. 通过maven执行编译命令

[root@hadoop101 hadoop-2.7.2-src]#mvn package -Pdist,native -DskipTests -Dtar

等待时间30分钟左右，最终成功是全部SUCCESS，如图2-42所示。

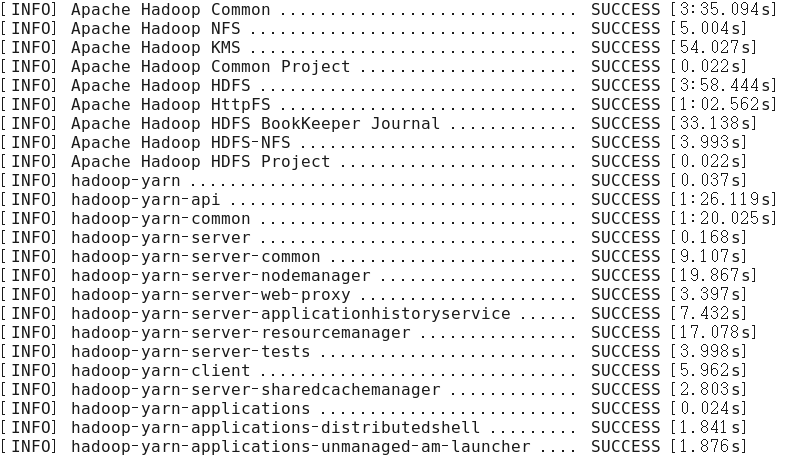


图2-42 编译源码

4. 成功的64位hadoop包在/opt/hadoop-2.7.2-src/hadoop-dist/target下

[root@hadoop101 target]# pwd

/opt/hadoop-2.7.2-src/hadoop-dist/target

5. 编译源码过程中常见的问题及解决方案

（1）MAVEN install时候JVM内存溢出

处理方式：在环境配置文件和maven的执行文件均可调整MAVEN\_OPT的heap大小。（详情查阅MAVEN 编译 JVM调优问题，如：http://outofmemory.cn/code-snippet/12652/maven-outofmemoryerror-method）

（2）编译期间maven报错。可能网络阻塞问题导致依赖库下载不完整导致，多次执行命令（一次通过比较难）：

[root@hadoop101 hadoop-2.7.2-src]#mvn package -Pdist,nativeN -DskipTests -Dtar

（3）报ant、protobuf等错误，插件下载未完整或者插件版本问题，最开始链接有较多特殊情况，同时推荐2.7.0版本的问题汇总帖子 <http://www.tuicool.com/articles/IBn63qf>

# 第5章 Hadoop任务调度机制

在目前的Hadoop 系统中，默认的调度器为FIFO调度，主要适合单队列的批处理作业需求，针对多用户多队列的控制需求，雅虎开发并且向开源社区贡献了容量调度器(Capacity Scheduler)和Facebook开发并贡献了公平调度器(FAIR Scheduler)。

## 5.1 作业调度概述

### 5.1.1 作业调度相关概念

**作业管理**:在调度系统中作业管理包括作业提交权限控制，作业运行状态查看权限控制等。

**用户和分组**：在Hadoop系统中以组为单位组织管理作业，每个用户只能向固定分组中提交作业，只能使用固定分组中配置的资源。同时可以限制每个用户提交的作业数、使用的资源等。

**资源池**：是Hadoop公平调度Fair Scheduler中的概念，一个资源池可以对应一个用户(User)、一个分组(Group)、或者一个队列(Queue)。

**队列**：队列是Hadoop中提出的概念，一个队列（Queue）可以由任意几个分组（Group）和任意几个用户（User）组成。

**资源槽位**：是Hadoop分布式系统进行资源管理的基本单位，是集群计算资源的抽象化，每个资源槽位都代表可以运行的一个任务（Map任务和Reduce任务）。Hadoop集群中的每个计算节点都拥有一定数量的资源槽，具体数目需要每个用户依据每个节点的内存、CPU等信息确定并配置，默认每个节点两个资源槽位，表示每个计算节点可以并发运行两个任务。

**作业调度和任务调度**：第一级是作业调度，也就是作业选择，作业调度器选取作业集合中的一个等待调度的作业。第二级是任务调度，也就是任务分配，由任务调度器在第一级选择的作业中选取一个就绪的任务来运行。

心跳：主节点负责管理所有从节点的资源，而这种管理是通过主从节点之间的心跳信息来互相通信的，也就是从节点定时向主节点发送状态信息——心跳信息来报告自己当前的状况

**本地化资源和非本地化资源**：待调度资源的作业集合中有一个作业至少存一个任务的待处理数据存位于该计算节点上，那么就可以称为这个计算节点是这个作业的本地化资源。

### 5.1.2 Hadoop作业调度流程



1）Client通过submitJob()函数向JobTracker提交一个作业。

2）JobTacker接收到用户的作业提交后会通过notify()函数通知调度器TaskScheduler有新作业

3）TaskTracker通过heartbeat心跳机制向JobTacker汇报TaskTracker的资源情况，JobTreacker同时获取TaskTrackerStatus信息，如果TaskTracker资源是空闲的，则主动向JobTracker请求分配任务。

4）JobTracker根据对TaskTracker资源的管理情况，请求调度器TaskScheduler分配作业，TaskScheduler根据对应的资源情况和任务数量，分配作业列表返回给JobTracker。

5）JobTracker接收到分配的作业列表，再通过HeartBeat心跳信息将任务分发给具体的TaskTracker，最终启动Task任务完成作业。

简单版：client提交作业到JobTracker，JobTracker告知TaskScheduler有新任务，TaskTracker通过心跳机制向JobTracker汇报资源的情况，JobTracker再告知到TaskScheduler，TaskScheduler根据资源情况分配任务并把任务列表返回给JobTracker，JobTracker告知TaskTracker启动任务，最后TaskTracker启动任务。

### 5.1.3 集群资源组织和管理

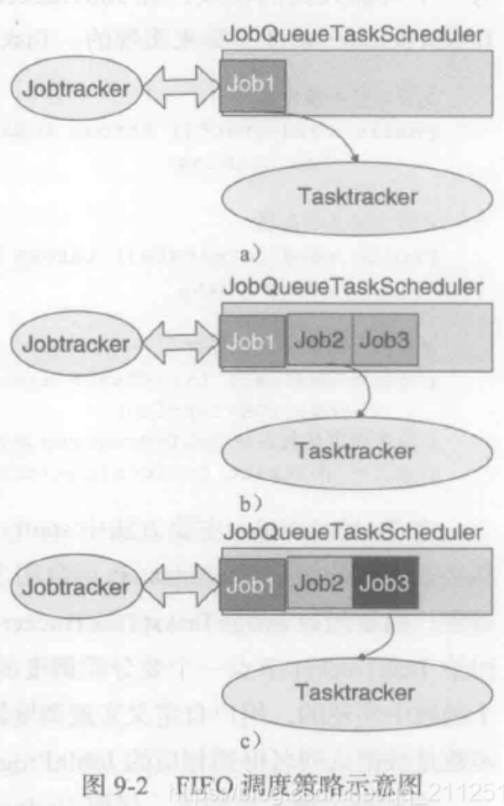
Hadoop系统作业调度是以计算槽位slot来组织集群计算资源的，计算槽位可以认为是Hadoop集群中计算资源的抽象，是集群资源管理的基本单位，计算槽位slot资源包括两种类型：Map slot和Reduce Slot。

## 5.2 FIFO调度器

FIFO调度器是Hadoop默认的调度器，其调度策略简单，容易实现，并且调度效率最高。

### 5.2.1 基本调度策略

类似于Linux操作系统中的FIFO先进先出进程调度算法，Hadoop FIFO调度器的调度策略是将用户提交的作业按照先后顺序放在一个队列中，然后依据先后顺序和优先级顺序被依次调度执行，整体上遵循先进先出基本原则。



FIFO调度策略遵循以下基本原则：

1）所有用户提交的作业会统一按照提交的先后顺序排列在同一个队列中。

2）支持优先级，包括VERY\_HIGH、HIGH、NORMAL、LOW、VERY\_LOW

3）在优先级相同的情况下，按照先来先服务的模式调度执行。

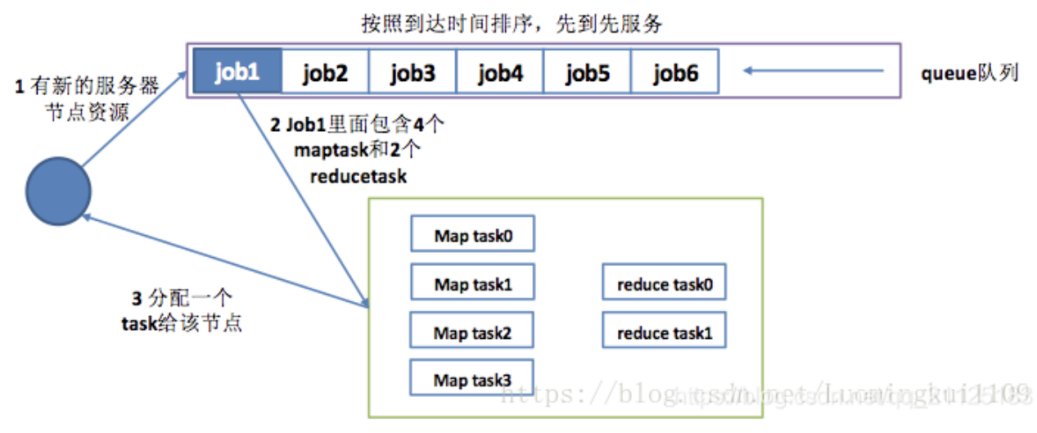
4）在优先级不同的情况下，优先级高的作业先被调度执行。

FIFO调度器有三种基本情况：

1）队列中只有一个作业时直接调度执行，如上图a只有一个Job1，则调度器JobQueueTaskSchedule直接将Job1调度分配给集群的计算节点TaskTracker执行；

2）用户在Job1之后先后提交了Job2和Job3，并且Job2和Job3的优先级是相同的。这个时候先运行Job2，然后再运行Job3；

3）用户在Job1之后先后提交了Job2和Job3，而Job3的优先级比Job2的优先级高，那么Job1运行完成之后，先调度Job3然后调度Job1。



总结：FIFO调度器逻辑设计简洁，对于单用户集群系统来讲比较适合，同时系统利用率高，响应时间也很短，但是对于多用户共享集群资源的情况下就会出现不能区别不同用户、不同作业类型的情况。

### 5.2.2 任务分配算法

获取当前TaskTracker的Map和Reduce数量信息。

计算需要运行的Map和Reduce的数量（也就是处于正在运行状态和挂起状态的）

计算Map和Rduce的负载因子：负载因子表示整个集群的负载情况，很明显负载因子越大集群的负载越高，随着任务的执行集群负载会减少，而用户也会不断地向队列中提交作业，这样集群整个负载因子又会增加，因此在实际的集群环境中比较高效的状态是负载因子维持在一个动态的范围之内。

计算tasktracker当前Map容量以及最大可分配的Map任务数量，然后调度分配map任务。

计算tasktracker当前Reduce容量以及最大可分配的reduce任务数量，然后调度分配reduce任务。

## 5.3 公平调度器

FIFO调度器的问题：FIOF调度器不支持多个队列，也不支持多个用户共享集群，这样就造成了集群资源利用率过低，对于不同类型的作业不能保证公平调度器等问题，公平调度器FairScheduler就这样产生了。

### 5.3.1 主要功能

实现了类似Linux内核的Fair-Share调度算法，以保证各个作业基本上能公平共享整个集群的资源。

支持多用户、多队列和划分资源池。

支持资源池最小共享保证和最大作业限制。

共享权重可分配：通过调整共享权重使得高优先级作业、大作业共享更多的资源。

支持资源抢占：当一个资源池有空闲资源槽时，调度器会将其共享给需要资源调度的资源池。

实现了延迟调度：延迟调度机制可以调数据本地任务的效率。

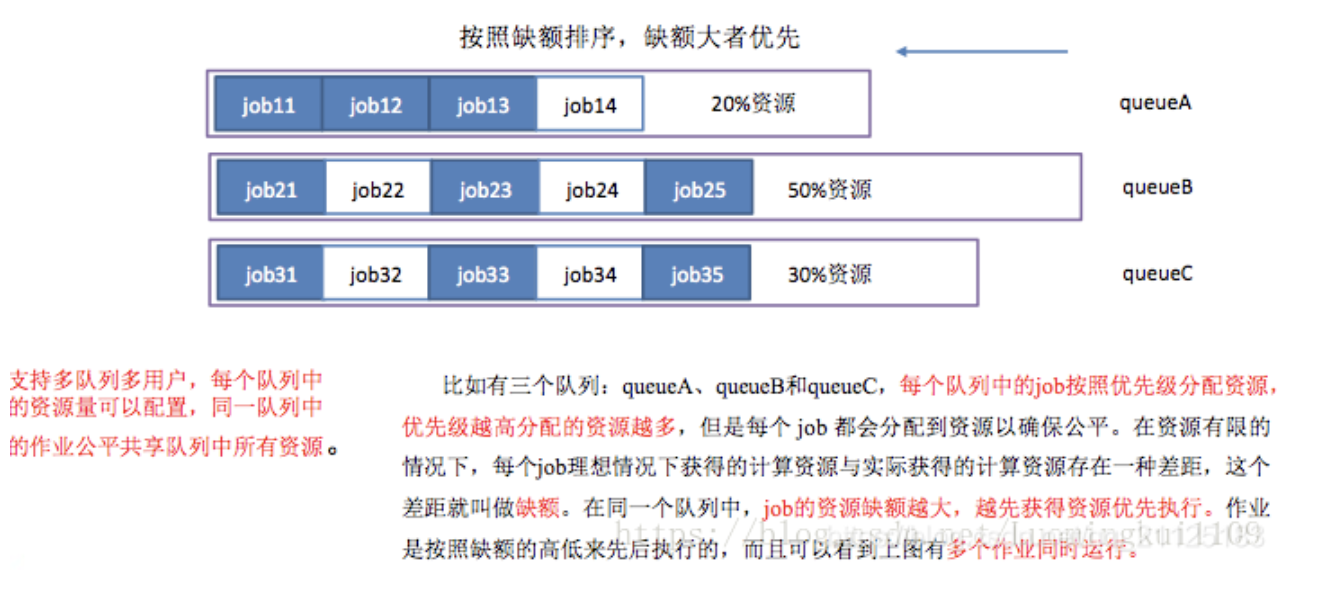
### 5.3.2 基本调度策略

公平调度策略的核心概念是随着时间的推移能平均获取同等的共享资源。当单独一个作业任务运行时，它将使用整个集群；当有其他作业被提交时，系统会将任务（task）空闲时间片（slot）赋给新的作业，以使得每一个作业都能获取等量的CPU时间。

公平调度器中有一个资源池pool的概念，并通过资源池来组织作业，把资源公平地分配到这些资源池里面。在默认的情况下，每一个用户都拥有一个独立的资源池，以使得每一个用户都能获取一份同等的集群资源而不管其提交了多少作业。

在每一个资源池内，会使用公平共享的方式在运行的作业间共享资源。用户可以给予资源池相应的权重，以不按比例的方式共享集群。除了提供公平共享的方法之外，公平调度器允许赋给资源池以保证最小的共享资源。

公平调度器也支持在可配置的超时时间后对允许的作业进行抢占。如果新的作业在一定时间内还获取不到最小的共享资源，这个作业别允许去终结已运行的作业中任务以获取运行所需要的资源。



## 5.4 容量调度器

容量调度器是雅虎结合自己的集群业务类型，提出的一种多用户调度器，这种调度器支持多用户、多队列，每一个队列都可以单独配置一定的资源量，每个队列采用FIFO策略，可以看做是FIFO调度器的多队列版本。

### 5.4.1 产生背景

公平调度器强调的是各个作业的公平共享原则，主要是保证资源池之间以公平地共享整个集群资源，然而整个集群的资源利用率的角度来看，公平调度并不能保证整个集群的资源的利用率最高，也就是虽然各种作业对资源的使用可以达到公平地原则，保证用户都可以公平地分享整个集群的资源，但是仍然存在计算资源利用率不高的问题，容量调度器就是用来解决这个问题。

在容量调度器中，每个作业被提交到一个队列中，每个队列分配整个集群资源的容量（capacity）的一定比例，队列中的作业以FIFO的方式占用整个队列分配的资源（capacity）。调度的基本策略是首先选择一个容量capacity实际占用率最低的队列，这样最需要资源的队列优先调度，然后从FIFO队列中选择一个合适的作业。

### 5.4.2 主要功能

支持多用户多队列

层次化队列机制

资源容量保证

作业权限控制

弹性资源分配

运行时的控制功能

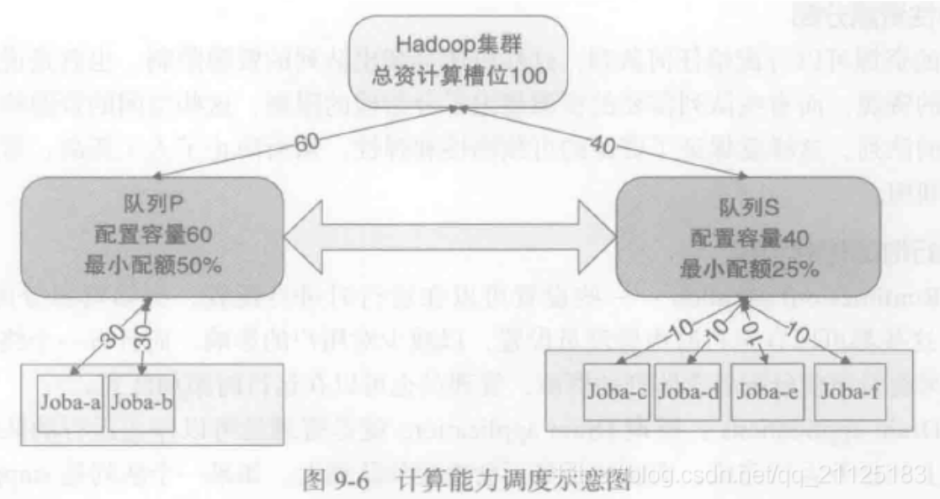
基于资源的调度

作业内存控制

延迟调度机制

### 5.4.3 基本调度策略

容量调度器的计算能力调度示意图如下：假定Hadoop集群中有100个计算资源槽位。调度器中配置了队列P和队列S，队列P中配置容量为60；S队列中配置容量为40。P中队列最小配额为50%，也就是P队列中可以同时运行两个作业，每个作业分配30个计算资源；S队列中最小配额为25%，也就是可以同时运行4个作业。在队列之间也会共享空闲的计算资源。容量调度算法中最重要的就是在选择作业的时候，会关注作业所属的用户是否已经超过了他所能使用的计算资源限制。



容量调度器还可以有效地对集群中资源中的内存资源进行管理，从而可以有效的支持内存密集型作业。如果一个作业对内存的资源需求比较高，那么调度算法就要保证将该作业的相关任务分派到具有充足内存资源的TaskTracker上执行，已避免由于内存不足造成无法执行。因此，在作业选择的过程中，计算能力调度算法还需要检查空闲TaskTracekr上的内存资源是否能够满足作业的内存需求。



## 5.5 公平和容量的区别

公平调度是以pool为单位分配任务slots的，容量调度以queue的方式分配tasktracker的，当都只有一个job的时候，两种调度器都可以利用整个集群的资源。在每个pool内部可以是以FIFO方式调度也可以是公平方式调度，但是在queue内部只能是以FIFO方式调度，这是公平调度和容量调度的一个区别，虽然在queue内部可以设置优先级但是并不支持抢占，在pool内部是支持抢占的，就是可以为某个pool（通常是生产上的用户）分配最低的任务slot，当这个pool得不到最低的资源的时候他就会kill其他的job的task以使自己获得最低的资源，这并不会引起其他的job的失败，因为job也是master-slaver的结构，也是容错的。

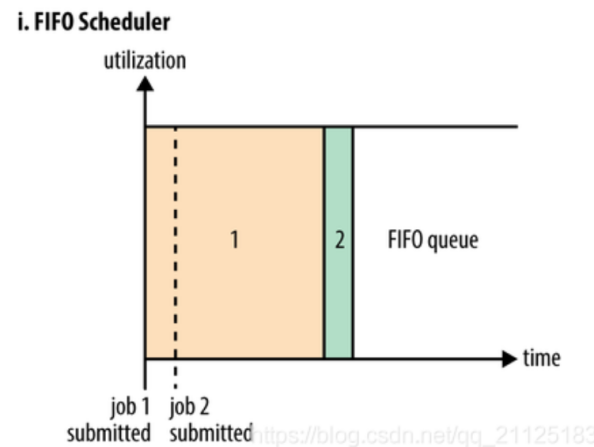
## 5.6 Yarn资源调度策略

Yarn调度器实际上就是Hadoop调度器，在Yarn中有三种调度器可以选择：FIFO Scheduler ，Capacity Scheduler容量调度器，Fair Scheduler公平调度器。

### 5.6.1 FIFO Scheduler

FIFO Scheduler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。

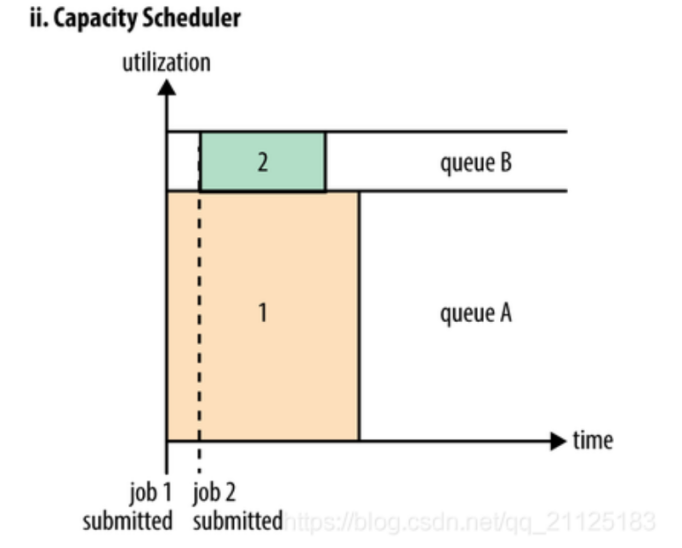


### 5.6.2 Capacity Scheduler

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。

Capacity 调度器允许多个组织共享整个集群，每个组织可以获得集群的一部分计算能力。通过为每个组织分配专门的队列，然后再为每个队列分配一定的集群资源，这样整个集群就可以通过设置多个队列的方式给多个组织提供服务了。除此之外，队列内部又可以垂直划分，这样一个组织内部的多个成员就可以共享这个队列资源了，在一个队列内部，资源的调度是采用的是先进先出(FIFO)策略。

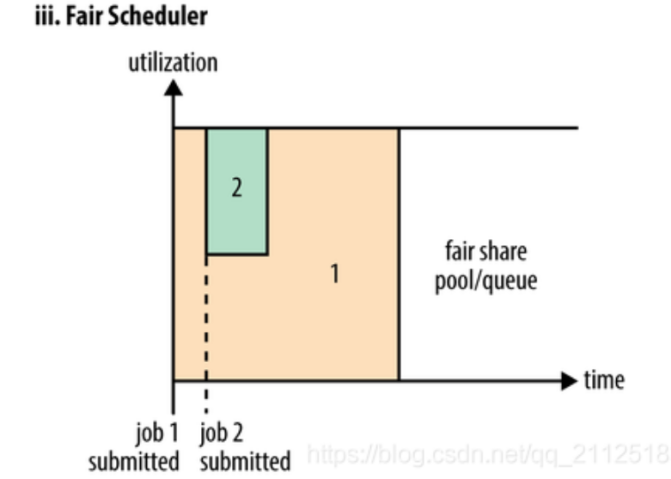
当队列已满，Capacity调度器不会强制释放Container，当一个队列资源不够用时，这个队列只能获得其它队列释放后的Container资源，这个称为“弹性队列”，也可以设置最大值，防止过多占用其他队列的资源。



### 5.6.3 Fair Scheduler

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。如下图所示，当第一个大job提交时，只有这一个job在运行，此时它获得了所有集群资源；当第二个小任务提交后，Fair调度器会分配一半资源给这个小任务，让这两个任务公平的共享集群资源。

需要注意的是，在下图Fair调度器中，从第二个任务提交到获得资源会有一定的延迟，因为它需要等待第一个任务释放占用的Container。小任务执行完成之后也会释放自己占用的资源，大任务又获得了全部的系统资源。最终的效果就是Fair调度器即得到了高的资源利用率又能保证小任务及时完成。



# 第6章 Hadoop集群添加/删除节点

## 6.1 添加节点

A：新节点中添加账户，设置无密码登陆

B：在NameNode中设置到新节点的无密码登陆

C：在NameNode slaves文件中添加新节点

D：在所有节点/etc/hosts文件中增加新节点（所有节点保持一致）

E：在新节点中执行

|  |  |
| --- | --- |
| 1  2 | ./bin/hadoop-daemon.sh start datanode  ./bin/hadoop-daemon.sh start tasktracker |

均衡各个数据节点中的数据  
./bin/start-balancer.sh

注意

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 1）如果不balance，那么cluster会把新的数据都存放在新的node上，这样会降低mapred的工作效率  2）设置平衡阈值，默认是10%，值越低各节点越平衡，但消耗时间也更长  ./bin/start-balancer.sh -threshold 5  3）设置balance的带宽，默认只有1M/s  <property>  　　<name>dfs.balance.bandwidthPerSec</name>  　　<value>1048576</value>  　　<description>  　　　　Specifies the maximum amount of bandwidth that each datanode  　　　　can utilize for the balancing purpose in term of  　　　　the number of bytes per second.  　　</description>  </property> |

## 6.2 删除节点

A：修改Name节点的hdfs-site.xml，增加

|  |  |
| --- | --- |
| 1  2  3  4 | <property>      <name>dfs.hosts.exclude</name>      <value>/soft/hadoop/conf/excludes</value>  </property> |

B：修改Name节点的mapred-site.xml，增加

|  |  |
| --- | --- |
| 1  2  3  4  5 | <property>     <name>mapred.hosts.exclude</name>     <value>/soft/hadoop/conf/excludes</value>     <final>true</final>  </property> |

C：新建excludes文件，文件里写要删除节点的hostname

D：Name节点执行

|  |  |
| --- | --- |
| 1  2 | hadoop mradmin –refreshNodes  hadoop dfsadmin –refreshNodes<br>(task进程可以kill进程ID) |

查看关闭进程

|  |  |
| --- | --- |
| 1 | hadoop dfsadmin -report |

当节点处于Decommissioned，表示关闭成功。

注意

1：如果刚删除的节点要再添加到集群要先删除excludes文件，执行 hadoop dfsadmin –refreshNodes

2：节点的状态可以通过http://NameNodes:50070查看

3：如果数据较大则关闭的时间可能会很长

# 第7章 常见错误及解决方案

1）防火墙没关闭、或者没有启动YARN

*INFO client.RMProxy: Connecting to ResourceManager at hadoop108/192.168.10.108:8032*

2）主机名称配置错误

3）IP地址配置错误

4）ssh没有配置好

5）root用户和atguigu两个用户启动集群不统一

6）配置文件修改不细心

7）未编译源码

*Unable to load native-hadoop library for your platform... using builtin-java classes where applicable*

*17/05/22 15:38:58 INFO client.RMProxy: Connecting to ResourceManager at hadoop108/192.168.10.108:8032*

8）不识别主机名称

java.net.UnknownHostException: hadoop102: hadoop102

at java.net.InetAddress.getLocalHost(InetAddress.java:1475)

at org.apache.hadoop.mapreduce.JobSubmitter.submitJobInternal(JobSubmitter.java:146)

at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1290)

at org.apache.hadoop.mapreduce.Job$10.run(Job.java:1287)

at java.security.AccessController.doPrivileged(Native Method)

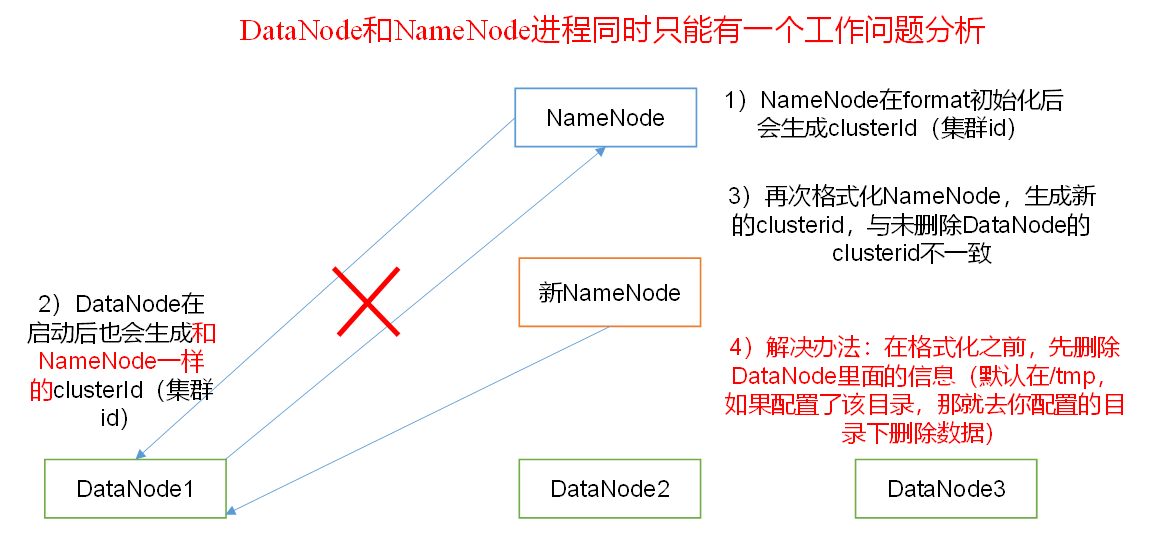
at javax.security.auth.Subject.doAs(Subject.java:415)

解决办法：

（1）在/etc/hosts文件中添加192.168.1.102 hadoop102

（2）主机名称不要起hadoop hadoop000等特殊名称

9）DataNode和NameNode进程同时只能工作一个。



10）执行命令不生效，粘贴word中命令时，遇到-和长–没区分开。导致命令失效

解决办法：尽量不要粘贴word中代码。

11）jps发现进程已经没有，但是重新启动集群，提示进程已经开启。原因是在linux的根目录下/tmp目录中存在启动的进程临时文件，将集群相关进程删除掉，再重新启动集群。

12）jps不生效。

原因：全局变量hadoop java没有生效。解决办法：需要source /etc/profile文件。

13）8088端口连接不上

[atguigu@hadoop102 桌面]$ cat /etc/hosts

注释掉如下代码

#127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4

#::1 hadoop102

# HDFS

# 第1章 HDFS概述

## 1.1 HDFS产出背景及定义

HDFS产生的背景：随着数据量越来越大，在一个操作系统存不下所有的数据，那么就分配到更多的操作系统管理的磁盘中，但是不方便维护，迫切需要一种系统来管理多台机器上的文件，这就是分布式文件管理系统。HDFS只是分布式文件管理系统种的一种。

HDFS定义：HDFS是一个文件系统，用于存储文件，通过目录树来定位文件；其次，它是分布式的，由很多服务器联合起来实现其功能，集群种的服务器有各自的角色。

HDFS的使用场景：适合一次写入多次读出的场景，且不支持文件的修改。适合用来做数据分析，并不适合当网盘应用。

## 1.2 HDFS优缺点

优点：

1. 高容错性
2. 数据自动保存多个副本，它通过增加副本的形式提高容错性
3. 某一个副本丢失后，可以自动恢复
4. 适合大数据处理

（1）文件规模：能处理高达GB级别以上的数据

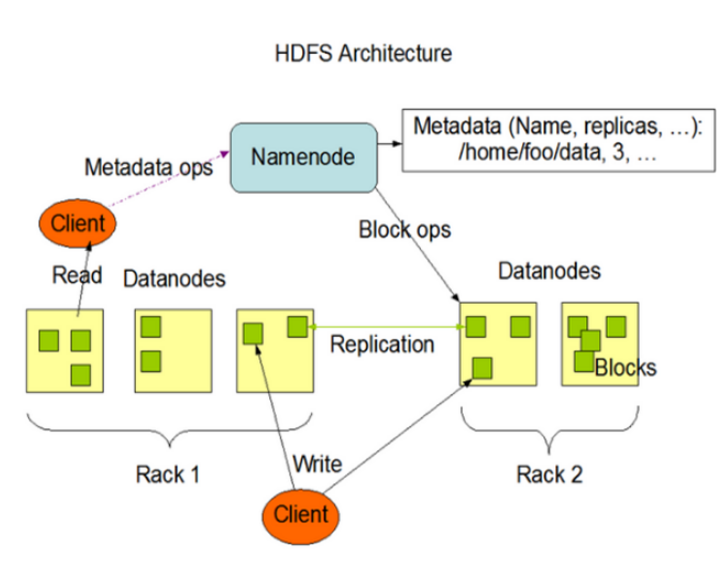
（2）文件规模：能处理百万规模以上的文件数量

3）可构建在廉价机器上，通过多副本机制提高可靠性

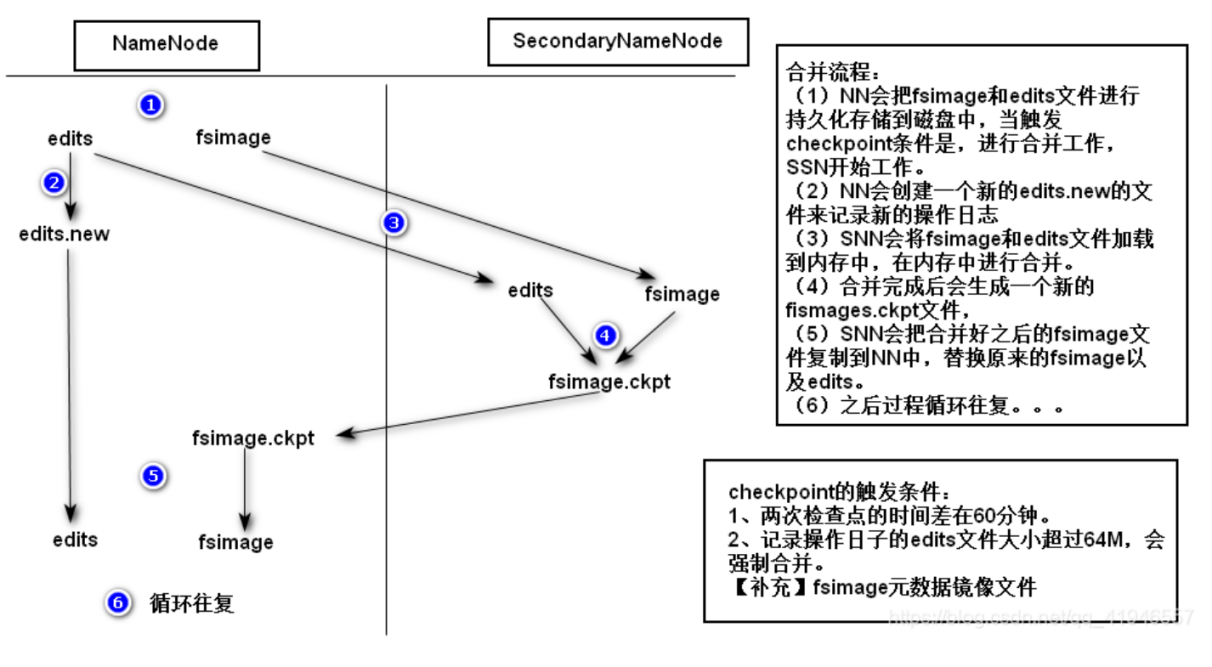
缺点：

1. 不适合低延时数据访问，比如毫秒级的存储数据是做不到的
2. 无法高效的对大量小文件进行存储
3. 大量小文件会占用NameNode大量的存储空间来存储文件目录和块信息，而NameNode的存储空间是有限的；
4. 小文件存储的寻址时间会超过读取时间，违反了HDFS的设计目标
5. 不支持并发写入，不支持文件随机修改
6. 一个文件只能一个写，不允许多个线程同时写；
7. 仅支持数据append（追加），不支持文件的随机修改

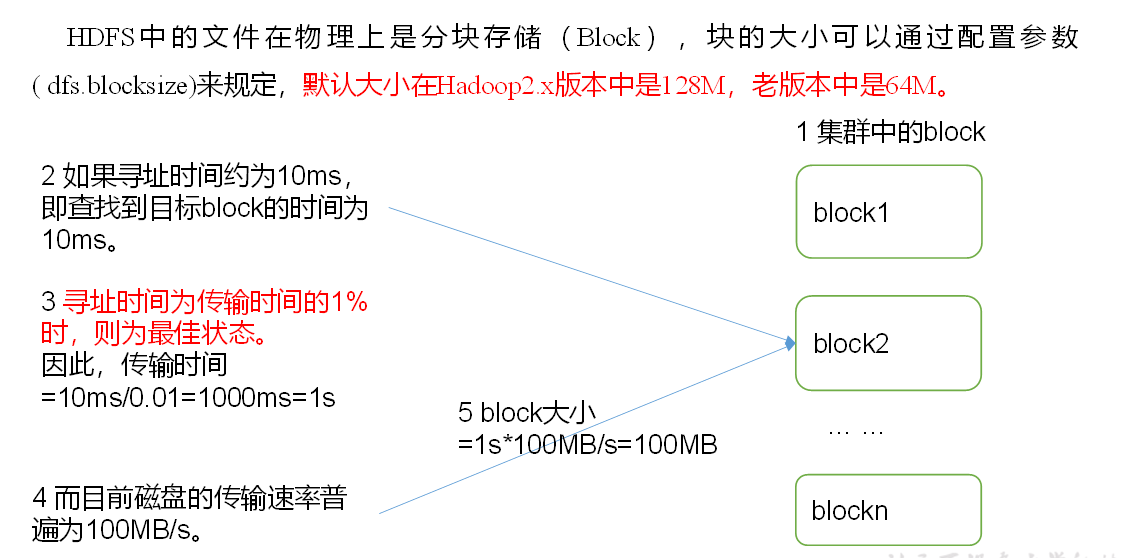
## 1.3 HDFS组成架构



1. NameNode：Master，它是一个主管、管理者
2. 管理HDFS的名称空间；
3. 配置副本策略；
4. 管理数据块映射信息
5. 处理客户端读写请求
6. DataNode：slave，执行NameNode下达的命令
7. 存储实际的数据块
8. 执行数据块的读写操作
9. Client：客户端
10. 文件切分。文件上传HDFS的时候，Client将文件切分成一个一个的Block，然后进行上传
11. 与NameNode交互，获取文件的位置信息
12. 与DataNode交互，读取或写入数据
13. Client提供一些命令来管理HDFS，比如NameNode格式化
14. Cilent可以通过一些命令来访问HDFS，比如对HDFS增删改查；
15. Secondary NameNode：并非NameNode的热备，当NameNode挂掉的时候，它并不能马上代替NameNode并提供服务。
16. 辅助NameNode，分担工作量，比如定期合并Fsimage和Edits并推送给NameNode；
17. 紧急情况下可以辅助恢复NameNode



## 1.4 HDFS文件块大小（面试重点）



思考：为什么块的大小不能设置太小，也不能设置太/大？

（1）HDFS的块设置太小，会增加寻址时间，程序一直在找块的开始位置；

（2）HDFS的块设置太大，从磁盘传输数据的时间会明显大于定位这个块开始位置所需的时间，导致程序在处理这些数据时，会非常慢。

一秒定律：如果磁盘普通传输速率100MB/s，那block设置为128MB；如果是固态传输速率300MB/s，那block设置应参考1s\*300MB/s=300MB， 则设置为256MB。（因为大小要为2的n次方，选择最靠近即可）

总结：HDFS块的大小设置主要取决于磁盘传输的速率。

# 第2章 HDFS的Shell操作（开发重点）

1．基本语法

bin/hadoop fs 具体命令 OR bin/hdfs dfs 具体命令

dfs是fs的实现类。

2．命令大全

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop fs

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

[-cat [-ignoreCrc] <src> ...]

[-checksum <src> ...]

[-chgrp [-R] GROUP PATH...]

[-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]

[-chown [-R] [OWNER][:[GROUP]] PATH...]

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

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

[-count [-q] <path> ...]

[-cp [-f] [-p] <src> ... <dst>]

[-createSnapshot <snapshotDir> [<snapshotName>]]

[-deleteSnapshot <snapshotDir> <snapshotName>]

[-df [-h] [<path> ...]]

[-du [-s] [-h] <path> ...]

[-expunge]

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

[-getfacl [-R] <path>]

[-getmerge [-nl] <src> <localdst>]

[-help [cmd ...]]

[-ls [-d] [-h] [-R] [<path> ...]]

[-mkdir [-p] <path> ...]

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

[-moveToLocal <src> <localdst>]

[-mv <src> ... <dst>]

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

[-renameSnapshot <snapshotDir> <oldName> <newName>]

[-rm [-f] [-r|-R] [-skipTrash] <src> ...]

[-rmdir [--ignore-fail-on-non-empty] <dir> ...]

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

[-setrep [-R] [-w] <rep> <path> ...]

[-stat [format] <path> ...]

[-tail [-f] <file>]

[-test -[defsz] <path>]

[-text [-ignoreCrc] <src> ...]

[-touchz <path> ...]

[-usage [cmd ...]]

3．常用命令实操

（0）启动Hadoop集群（方便后续的测试）

[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-dfs.sh

[atguigu@hadoop103 hadoop-2.7.2]$ sbin/start-yarn.sh

（1）-help：输出这个命令参数

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -help rm

（2）-ls: 显示目录信息

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -ls /

（3）-mkdir：在HDFS上创建目录

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -mkdir -p /sanguo/shuguo

（4）-moveFromLocal：从本地剪切粘贴到HDFS

[atguigu@hadoop102 hadoop-2.7.2]$ touch kongming.txt

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -moveFromLocal ./kongming.txt /sanguo/shuguo

（5）-appendToFile：追加一个文件到已经存在的文件末尾

[atguigu@hadoop102 hadoop-2.7.2]$ touch liubei.txt

[atguigu@hadoop102 hadoop-2.7.2]$ vi liubei.txt

输入

san gu mao lu

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -appendToFile liubei.txt /sanguo/shuguo/kongming.txt

（6）-cat：显示文件内容

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -cat /sanguo/shuguo/kongming.txt

（7）-chgrp 、-chmod、-chown：Linux文件系统中的用法一样，修改文件所属权限

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -chmod 666 /sanguo/shuguo/kongming.txt

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -chown atguigu:atguigu /sanguo/shuguo/kongming.txt

（8）-copyFromLocal：从本地文件系统中拷贝文件到HDFS路径去

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -copyFromLocal README.txt /

（9）-copyToLocal：从HDFS拷贝到本地

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -copyToLocal /sanguo/shuguo/kongming.txt ./

（10）-cp ：从HDFS的一个路径拷贝到HDFS的另一个路径

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -cp /sanguo/shuguo/kongming.txt /zhuge.txt

（11）-mv：在HDFS目录中移动文件

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -mv /zhuge.txt /sanguo/shuguo/

（12）-get：等同于copyToLocal，就是从HDFS下载文件到本地

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -get /sanguo/shuguo/kongming.txt ./

（13）-getmerge：合并下载多个文件，比如HDFS的目录 /user/atguigu/test下有多个文件:log.1, log.2,log.3,...

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -getmerge /user/atguigu/test/\* ./zaiyiqi.txt

（14）-put：等同于copyFromLocal

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -put ./zaiyiqi.txt /user/atguigu/test/

（15）-tail：显示一个文件的末尾

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -tail /sanguo/shuguo/kongming.txt

（16）-rm：删除文件或文件夹

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -rm /user/atguigu/test/jinlian2.txt

（17）-rmdir：删除空目录

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -mkdir /test

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -rmdir /test

（18）-du统计文件夹的大小信息

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -du -s -h /user/atguigu/test

2.7 K /user/atguigu/test

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -du -h /user/atguigu/test

1.3 K /user/atguigu/test/README.txt

15 /user/atguigu/test/jinlian.txt

1.4 K /user/atguigu/test/zaiyiqi.txt

（19）-setrep：设置HDFS中文件的副本数量

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -setrep 10 /sanguo/shuguo/kongming.txt



图3-3 HDFS副本数量

这里设置的副本数只是记录在NameNode的元数据中，是否真的会有这么多副本，还得看DataNode的数量。因为目前只有3台设备，最多也就3个副本，只有节点数的增加到10台时，副本数才能达到10。

# 第3章 HDFS客户端操作（开发重点）

## 3.1 HDFS客户端环境准备

1．根据自己电脑的操作系统拷贝对应的编译后的hadoop jar包到非中文路径（例如：D:\Develop\hadoop-2.7.2），如图3-4所示。

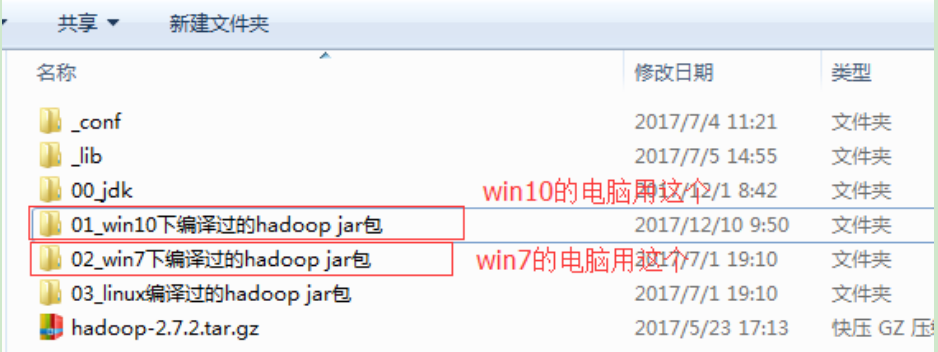


图3-4 编译后的hadoop jar包

2．配置HADOOP\_HOME环境变量，如图3-5所示。

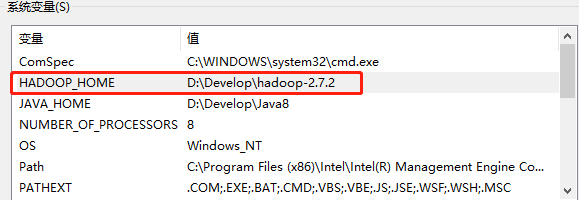


图3-5 配置HADOOP\_HOME环境变量

3. 配置Path环境变量，如图3-6所示。

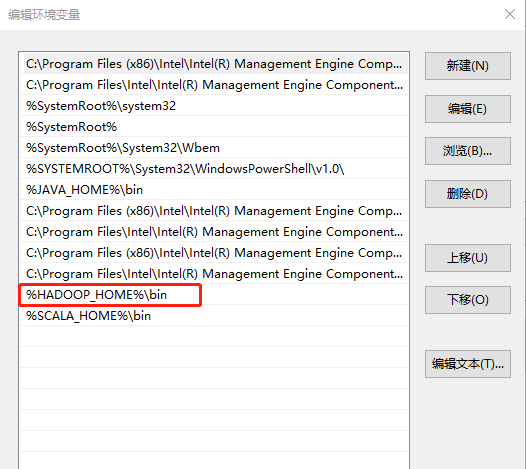


图3-6 配置Path环境变量

4．创建一个Maven工程HdfsClientDemo

5．导入相应的依赖坐标+日志添加

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>RELEASE</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-core</artifactId>

<version>2.8.2</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-common</artifactId>

<version>2.7.2</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>2.7.2</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-hdfs</artifactId>

<version>2.7.2</version>

</dependency>

<dependency>

<groupId>jdk.tools</groupId>

<artifactId>jdk.tools</artifactId>

<version>1.8</version>

<scope>system</scope>

<systemPath>${JAVA\_HOME}/lib/tools.jar</systemPath>

</dependency>

</dependencies>

注意：如果Eclipse/Idea打印不出日志，在控制台上只显示如下内容

1.log4j:WARN No appenders could be found for logger (org.apache.hadoop.util.Shell).

2.log4j:WARN Please initialize the log4j system properly.

3.log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

需要在项目的src/main/resources目录下，新建一个文件，命名为“log4j.properties”，在文件中填入

log4j.rootLogger=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.appender.logfile=org.apache.log4j.FileAppender

log4j.appender.logfile.File=target/spring.log

log4j.appender.logfile.layout=org.apache.log4j.PatternLayout

log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n

6．创建包名：com.atguigu.hdfs

7．创建HdfsClient类

public class HdfsClient{

@Test

public void testMkdirs() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

// 配置在集群上运行

// configuration.set("fs.defaultFS", "hdfs://hadoop102:9000");

// FileSystem fs = FileSystem.get(configuration);

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 创建目录

fs.mkdirs(new Path("/1108/daxian/banzhang"));

// 3 关闭资源

fs.close();

}

}

8．执行程序

运行时需要配置用户名称，如图3-7所示

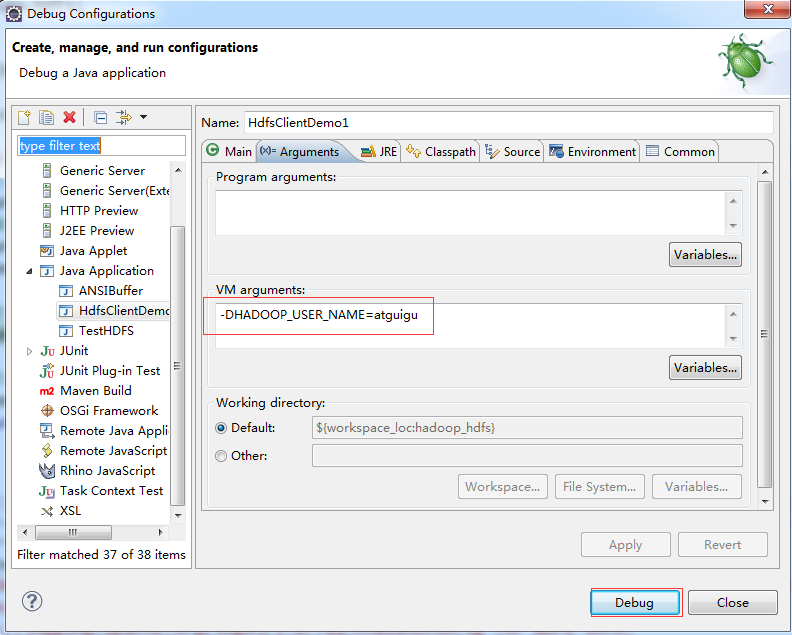


图3-7 配置用户名称

客户端去操作HDFS时，是有一个用户身份的。默认情况下，HDFS客户端API会从JVM中获取一个参数来作为自己的用户身份：-DHADOOP\_USER\_NAME=atguigu，atguigu为用户名称。

## 3.2 HDFS的API操作

### 3.2.2 HDFS文件下载

@Test

public void testCopyToLocalFile() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 执行下载操作

// boolean delSrc 指是否将原文件删除

// Path src 指要下载的文件路径

// Path dst 指将文件下载到的路径

// boolean useRawLocalFileSystem 是否开启文件校验

fs.copyToLocalFile(false, new Path("/banzhang.txt"), new Path("e:/banhua.txt"), true);

// 3 关闭资源

fs.close();

}

### 3.2.1 HDFS文件上传（测试参数优先级）

1．编写源代码

|  |
| --- |
| @Test  public void testCopyFromLocalFile() throws IOException, InterruptedException, URISyntaxException {  // 1 获取文件系统  Configuration configuration = new Configuration();  configuration.set("dfs.replication", "2");  FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");  // 2 上传文件  fs.copyFromLocalFile(new Path("e:/banzhang.txt"), new Path("/banzhang.txt"));  // 3 关闭资源  fs.close();  System.out.println("over");  } |

2．将hdfs-site.xml拷贝到项目的根目录下

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  <configuration>  <property>  <name>dfs.replication</name>  <value>1</value>  </property>  </configuration> |

3．参数优先级

参数优先级排序：（1）客户端代码中设置的值 >（2）ClassPath下的用户自定义配置文件 >（3）然后是服务器的默认配置

### 3.2.3 HDFS文件夹删除

@Test

public void testDelete() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 执行删除

fs.delete(new Path("/0508/"), true);

// 3 关闭资源

fs.close();

}

### 3.2.4 HDFS文件名更改

@Test

public void testRename() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 修改文件名称

fs.rename(new Path("/banzhang.txt"), new Path("/banhua.txt"));

// 3 关闭资源

fs.close();

}

### 3.2.5 HDFS文件详情查看

查看文件名称、权限、长度、块信息

@Test

public void testListFiles() throws IOException, InterruptedException, URISyntaxException{

// 1获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 获取文件详情

RemoteIterator<LocatedFileStatus> listFiles = fs.listFiles(new Path("/"), true);

while(listFiles.hasNext()){

LocatedFileStatus status = listFiles.next();

// 输出详情

// 文件名称

System.out.println(status.getPath().getName());

// 长度

System.out.println(status.getLen());

// 权限

System.out.println(status.getPermission());

// 分组

System.out.println(status.getGroup());

// 获取存储的块信息

BlockLocation[] blockLocations = status.getBlockLocations();

for (BlockLocation blockLocation : blockLocations) {

// 获取块存储的主机节点

String[] hosts = blockLocation.getHosts();

for (String host : hosts) {

System.out.println(host);

}

}

System.out.println("-----------班长的分割线----------");

}

// 3 关闭资源

fs.close();

}

### 3.2.6 HDFS文件和文件夹判断

@Test

public void testListStatus() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件配置信息

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 判断是文件还是文件夹

FileStatus[] listStatus = fs.listStatus(new Path("/"));

for (FileStatus fileStatus : listStatus) {

// 如果是文件

if (fileStatus.isFile()) {

System.out.println("f:"+fileStatus.getPath().getName());

}else {

System.out.println("d:"+fileStatus.getPath().getName());

}

}

// 3 关闭资源

fs.close();

}

## 3.3 HDFS的I/O流操作（扩展）

上面我们学的API操作HDFS系统都是框架封装好的。那么如果我们想自己实现上述API的操作该怎么实现呢？

我们可以采用IO流的方式实现数据的上传和下载。

### 3.3.1 HDFS文件上传

1．需求：把本地e盘上的banhua.txt文件上传到HDFS根目录

2．编写代码

@Test

public void putFileToHDFS() throws IOException, InterruptedException, URISyntaxException {

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 创建输入流

FileInputStream fis = new FileInputStream(new File("e:/banhua.txt"));

// 3 获取输出流

FSDataOutputStream fos = fs.create(new Path("/banhua.txt"));

// 4 流对拷

IOUtils.copyBytes(fis, fos, configuration);

// 5 关闭资源

IOUtils.closeStream(fos);

IOUtils.closeStream(fis);

fs.close();

}

### 3.3.2 HDFS文件下载

1．需求：从HDFS上下载banhua.txt文件到本地e盘上

2．编写代码

// 文件下载

@Test

public void getFileFromHDFS() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 获取输入流

FSDataInputStream fis = fs.open(new Path("/banhua.txt"));

// 3 获取输出流

FileOutputStream fos = new FileOutputStream(new File("e:/banhua.txt"));

// 4 流的对拷

IOUtils.copyBytes(fis, fos, configuration);

// 5 关闭资源

IOUtils.closeStream(fos);

IOUtils.closeStream(fis);

fs.close();

}

### 3.3.3 定位文件读取

1．需求：分块读取HDFS上的大文件，比如根目录下的/hadoop-2.7.2.tar.gz

2．编写代码

（1）下载第一块

@Test

public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 获取输入流

FSDataInputStream fis = fs.open(new Path("/hadoop-2.7.2.tar.gz"));

// 3 创建输出流

FileOutputStream fos = new FileOutputStream(new File("e:/hadoop-2.7.2.tar.gz.part1"));

// 4 流的拷贝

byte[] buf = new byte[1024];

for(int i =0 ; i < 1024 \* 128; i++){

fis.read(buf);

fos.write(buf);

}

// 5关闭资源

IOUtils.closeStream(fis);

IOUtils.closeStream(fos);

fs.close();

}

（2）下载第二块

@Test

public void readFileSeek2() throws IOException, InterruptedException, URISyntaxException{

// 1 获取文件系统

Configuration configuration = new Configuration();

FileSystem fs = FileSystem.get(new URI("hdfs://hadoop102:9000"), configuration, "atguigu");

// 2 打开输入流

FSDataInputStream fis = fs.open(new Path("/hadoop-2.7.2.tar.gz"));

// 3 定位输入数据位置

fis.seek(1024\*1024\*128);

// 4 创建输出流

FileOutputStream fos = new FileOutputStream(new File("e:/hadoop-2.7.2.tar.gz.part2"));

// 5 流的对拷

IOUtils.copyBytes(fis, fos, configuration);

// 6 关闭资源

IOUtils.closeStream(fis);

IOUtils.closeStream(fos);

}

（3）合并文件

在Window命令窗口中进入到目录E:\，然后执行如下命令，对数据进行合并

type hadoop-2.7.2.tar.gz.part2 >> hadoop-2.7.2.tar.gz.part1

合并完成后，将hadoop-2.7.2.tar.gz.part1重新命名为hadoop-2.7.2.tar.gz，解压发现该tar包非常完整。

# 第4章 HDFS的数据流（面试重点）

## 4.1 HDFS写数据流程

### 4.1.1 剖析文件写入

HDFS写数据流程，如图3-8所示。

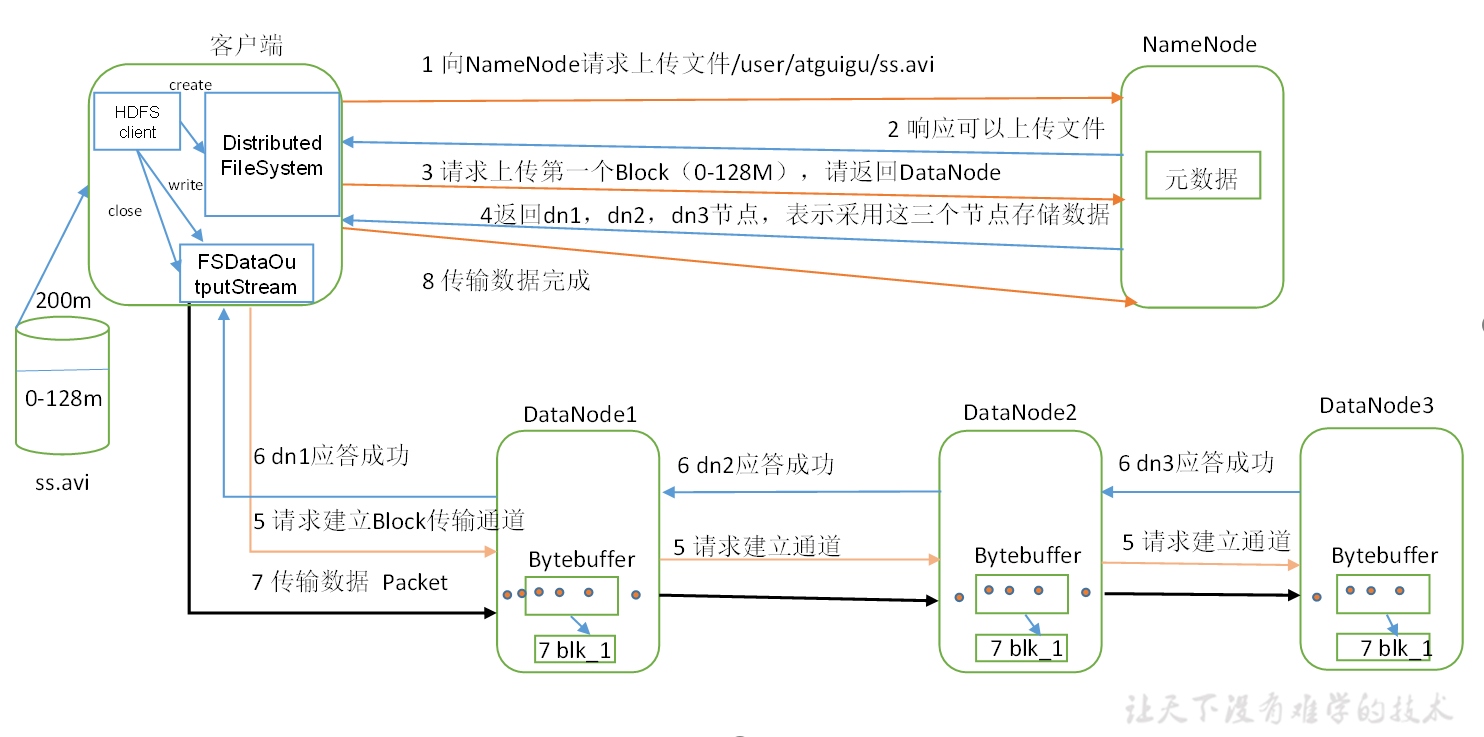


图3-8 配置用户名称

1）客户端通过Distributed FileSystem模块向NameNode请求上传文件，NameNode检查目标文件是否已存在，父目录是否存在。

2）NameNode返回是否可以上传。

3）客户端请求第一个 Block上传到哪几个DataNode服务器上。

4）NameNode返回3个DataNode节点，分别为dn1、dn2、dn3。

5）客户端通过FSDataOutputStream模块请求dn1上传数据，dn1收到请求会继续调用dn2，然后dn2调用dn3，将这个通信管道建立完成。

6）dn1、dn2、dn3逐级应答客户端。

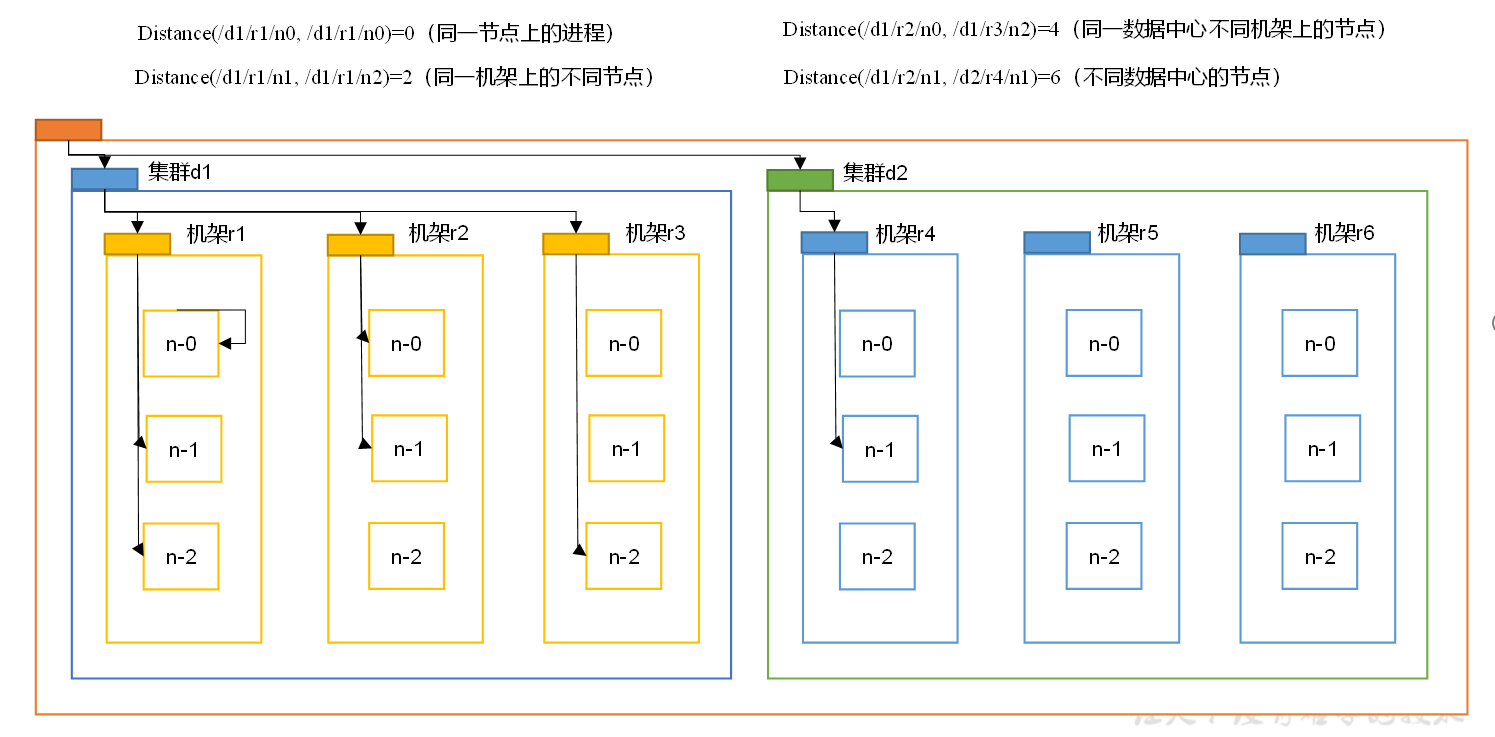
7）客户端开始往dn1上传第一个Block（先从磁盘读取数据放到一个本地内存缓存），以Packet为单位，dn1收到一个Packet就会传给dn2，dn2传给dn3；dn1每传一个packet会放入一个应答队列等待应答。

8）当一个Block传输完成之后，客户端再次请求NameNode上传第二个Block的服务器。（重复执行3-7步）。

### 4.1.2 网络拓扑-节点距离计算

在HDFS写数据的过程中，NameNode会选择距离待上传数据最近距离的DataNode接收数据。那么这个最近距离怎么计算呢？

节点距离：两个节点到达最近的共同祖先的距离总和。



例如，假设有数据中心d1机架r1中的节点n1。该节点可以表示为/d1/r1/n1。利用这种标记，这里给出四种距离描述，如图3-9所示。

大家算一算每两个节点之间的距离，如图3-10所示。

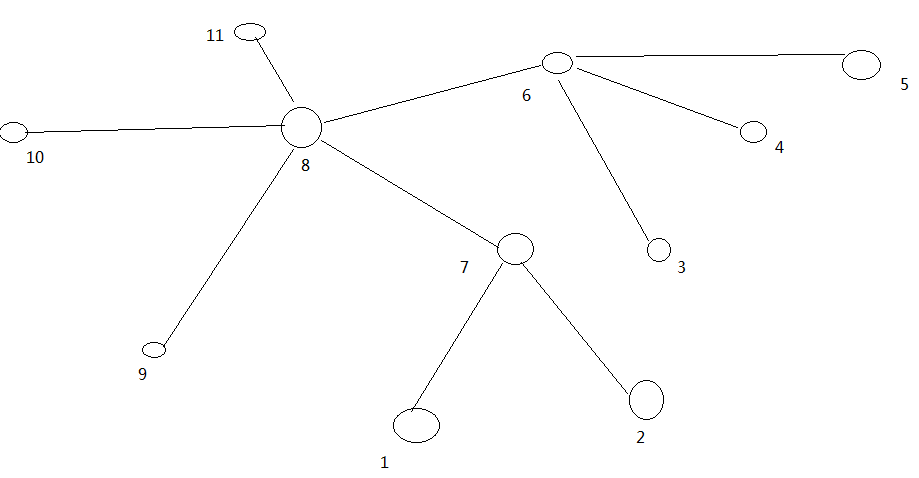


图3-10 网络拓扑

### 4.1.3 机架感知（副本存储节点选择）

1. 官方ip地址

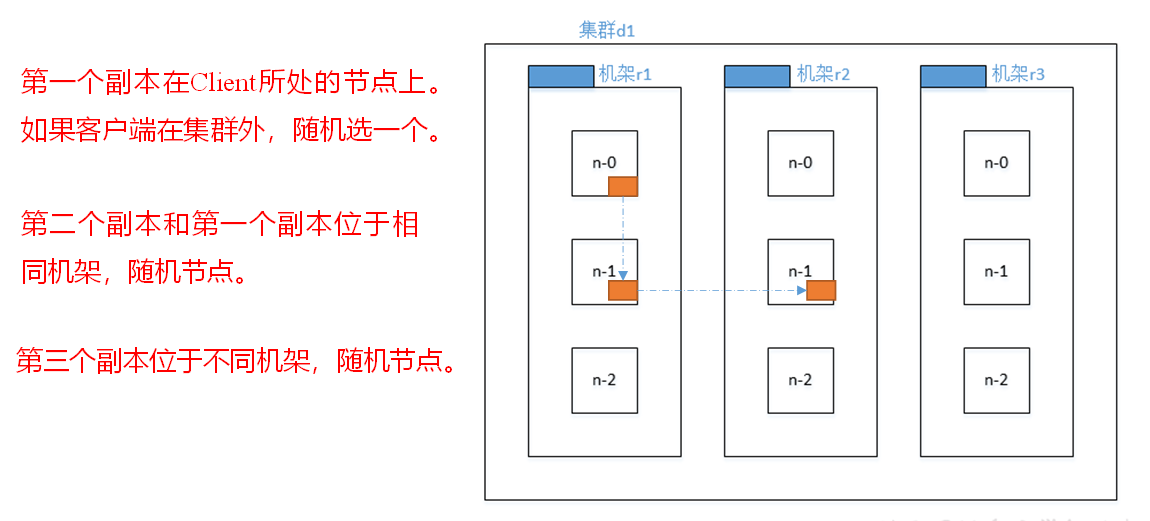
机架感知说明

<http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data_Replication>

For the common case, when the replication factor is three, HDFS’s placement policy is to put one replica on one node in the local rack, another on a different node in the local rack, and the last on a different node in a different rack.

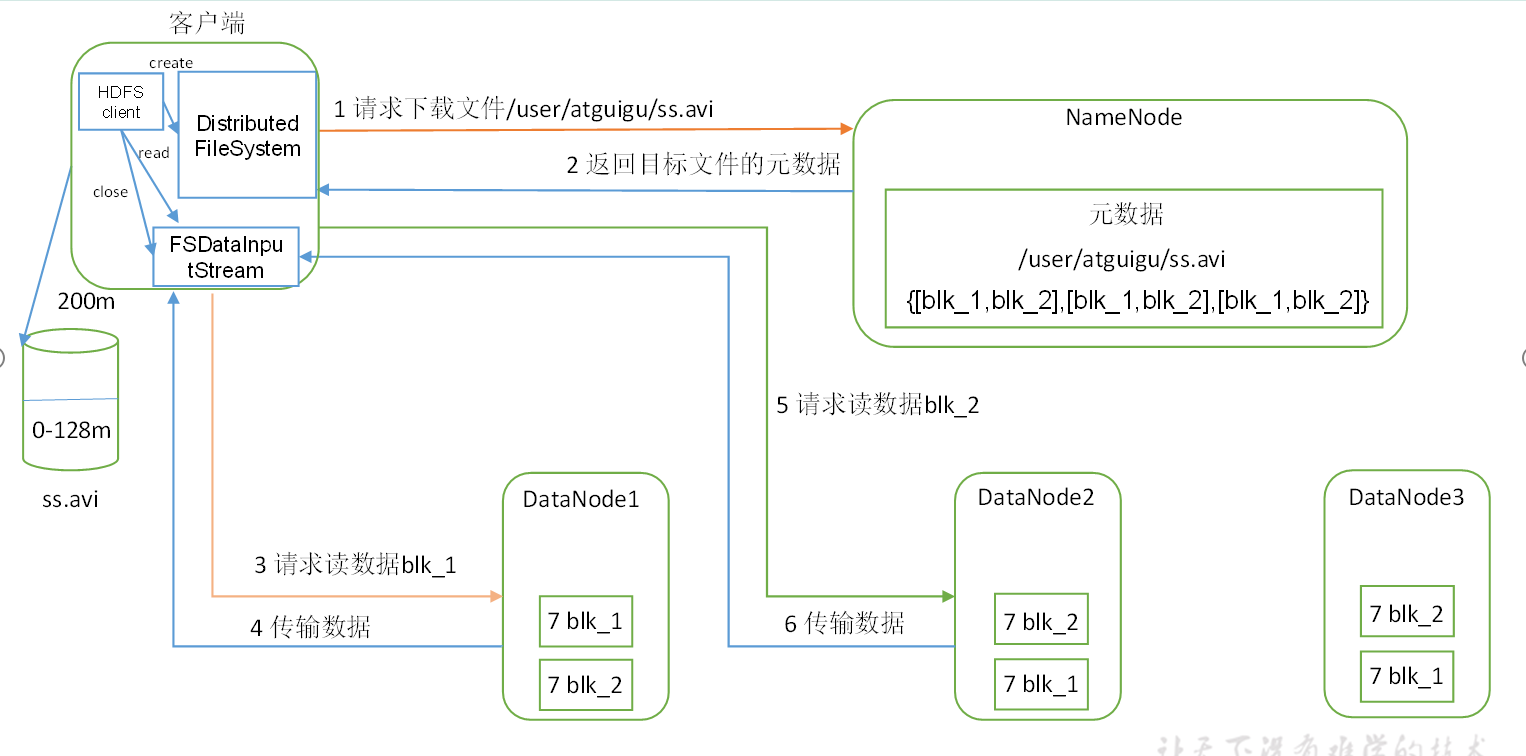
在常见情况下，当复制因子为3时，HDFS的放置策略是将一个副本放置在本地机架中的一个节点上，将另一个副本放置在本地机架中的另一个节点上，最后一个副本放置在不同机架中的另一个节点上。

2. Hadoop2.7.2副本节点选择



## 4.2 HDFS读数据流程

HDFS的读数据流程，如图3-13所示。



1）客户端通过Distributed FileSystem向NameNode请求下载文件，NameNode通过查询元数据，找到文件块所在的DataNode地址。

2）挑选一台DataNode（就近原则，然后随机）服务器，请求读取数据。

3）DataNode开始传输数据给客户端（从磁盘里面读取数据输入流，以Packet为单位来做校验）。

4）客户端以Packet为单位接收，先在本地缓存，然后写入目标文件。

# 第5章 HDFS1.0和2.0

在Hadoop2.0之前，NameNode只有一个，存在单点问题（虽然Hadoop1.0有SecondaryNameNode，CheckPointNode，BackupNode这些，但是单点问题依然存在），在hadoop2.0引入了HA机制。Hadoop2.0的HA机制官方介绍了有2种方式，一种是NFS（Network File System）方式，另外一种是QJM（Quorum Journal Manager）方式。

## 5.1 HDFS 1.0

### 5.1.1 NN和2NN工作机制

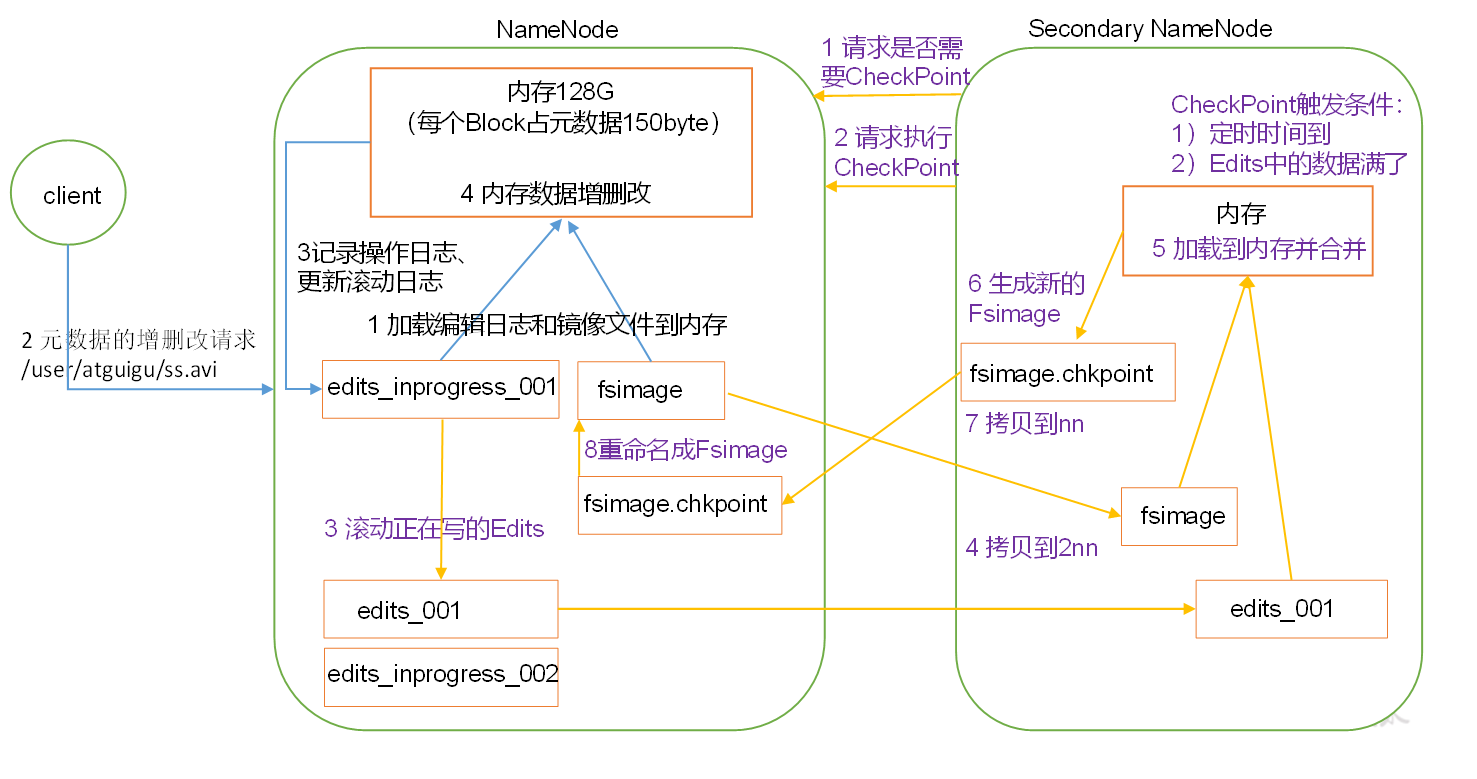
思考：NameNode中的元数据是存储在哪里的？

首先，我们做个假设，如果存储在NameNode节点的磁盘中，因为经常需要进行随机访问，还有响应客户请求，必然是效率过低。因此，元数据需要存放在内存中。但如果只存在内存中，一旦断电，元数据丢失，整个集群就无法工作了。因此产生在磁盘中备份元数据的FsImage。

这样又会带来新的问题，当在内存中的元数据更新时，如果同时更新FsImage，就会导致效率过低，但如果不更新，就会发生一致性问题，一旦NameNode节点断电，就会产生数据丢失。因此，引入Edits文件(只进行追加操作，效率很高)。每当元数据有更新或者添加元数据时，修改内存中的元数据并追加到Edits中。这样，一旦NameNode节点断电，可以通过FsImage和Edits的合并，合成元数据。

但是，如果长时间添加数据到Edits中，会导致该文件数据过大，效率降低，而且一旦断电，恢复元数据需要的时间过长。因此，需要定期进行FsImage和Edits的合并，如果这个操作由NameNode节点完成，又会效率过低。因此，引入一个新的节点SecondaryNamenode，专门用于FsImage和Edits的合并。

NN和2NN工作机制，如图3-14所示。



1. 第一阶段：NameNode启动

（1）第一次启动NameNode格式化后，创建Fsimage和Edits文件。如果不是第一次启动，直接加载编辑日志和镜像文件到内存。

（2）客户端对元数据进行增删改的请求。

（3）NameNode记录操作日志，更新滚动日志。

（4）NameNode在内存中对元数据进行增删改。

2. 第二阶段：Secondary NameNode工作

（1）Secondary NameNode询问NameNode是否需要CheckPoint。直接带回NameNode是否检查结果。

（2）Secondary NameNode请求执行CheckPoint。

（3）NameNode滚动正在写的Edits日志。

（4）将滚动前的编辑日志和镜像文件拷贝到Secondary NameNode。

（5）Secondary NameNode加载编辑日志和镜像文件到内存，并合并。

（6）生成新的镜像文件fsimage.chkpoint。

（7）拷贝fsimage.chkpoint到NameNode。

（8）NameNode将fsimage.chkpoint重新命名成fsimage。

|  |
| --- |
| **NN和2NN工作机制详解：**  Fsimage：NameNode内存中元数据序列化后形成的文件。  Edits：记录客户端更新元数据信息的每一步操作（可通过Edits运算出元数据）。  NameNode启动时，先滚动Edits并生成一个空的edits.inprogress，然后加载Edits和Fsimage到内存中，此时NameNode内存就持有最新的元数据信息。Client开始对NameNode发送元数据的增删改的请求，这些请求的操作首先会被记录到edits.inprogress中（查询元数据的操作不会被记录在Edits中，因为查询操作不会更改元数据信息），如果此时NameNode挂掉，重启后会从Edits中读取元数据的信息。然后，NameNode会在内存中执行元数据的增删改的操作。  由于Edits中记录的操作会越来越多，Edits文件会越来越大，导致NameNode在启动加载Edits时会很慢，所以需要对Edits和Fsimage进行合并（所谓合并，就是将Edits和Fsimage加载到内存中，照着Edits中的操作一步步执行，最终形成新的Fsimage）。SecondaryNameNode的作用就是帮助NameNode进行Edits和Fsimage的合并工作。  SecondaryNameNode首先会询问NameNode是否需要CheckPoint（触发CheckPoint需要满足两个条件中的任意一个，定时时间到和Edits中数据写满了），直接返回NameNode是否检查结果。SecondaryNameNode执行CheckPoint操作，首先会让NameNode滚动Edits并生成一个空的edits.inprogress，滚动Edits的目的是给Edits打个标记，以后所有新的操作都写入edits.inprogress，其他未合并的Edits和Fsimage会拷贝到SecondaryNameNode的本地，然后将拷贝的Edits和Fsimage加载到内存中进行合并，生成fsimage.chkpoint，然后将fsimage.chkpoint拷贝给NameNode，重命名为Fsimage后替换掉原来的Fsimage。NameNode在启动时就只需要加载新的Edits和新的Fsimage即可，旧的Edits数据已经合并到新的Fsimage中。 |

### 5.1.2 Fsimage和Edits解析

1. 概念

NameNode被格式化后会在/opt/module/hadoop-2.7.2/data/tmp/dfs/name/current目录中产生如下文件

fsimage\_00000000000000

fsimage\_00000000000000.md5

seen\_txid

VERSION

（1）Fsimage文件：HDFS文件系统元数据的一个永久性的检查点，其中包括HDFS文件系统的所有目录和文件inode的序列化信息

（2）Edits文件：存放HDFS文件系统的所有更新操作的路径，文件系统客户端执行的所有写操作优先记录到Edits文件种。

（3）seen\_txid文件保存的是一个数字，就是最后一个edits的数字

每次NameNode启动的时候都会将Fsimage文件读入内存，加载Edits里面的更新操作，保证内存中的元数据信息是最新的、同步的，可以看成NameNode启动的时候就将Fsimage和Edits文件进行合并。

2. oiv查看Fsimage文件

（1）查看oiv和oev命令

[atguigu@hadoop102 current]$ hdfs

**oiv** apply the offline fsimage viewer to an fsimage

**oev** apply the offline edits viewer to an edits file

（2）基本语法

hdfs oiv -p 文件类型 -i镜像文件 -o 转换后文件输出路径

（3）案例实操

[atguigu@hadoop102 current]$ pwd

/opt/module/hadoop-2.7.2/data/tmp/dfs/name/current

[atguigu@hadoop102 current]$ hdfs oiv -p XML -i fsimage\_0000000000000000025 -o /opt/module/hadoop-2.7.2/fsimage.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-2.7.2/fsimage.xml

将显示的xml文件内容拷贝到Eclipse中创建的xml文件中，并格式化。部分显示结果如下。

<inode>

<id>16386</id>

<type>DIRECTORY</type>

<name>user</name>

<mtime>1512722284477</mtime>

<permission>atguigu:supergroup:rwxr-xr-x</permission>

<nsquota>-1</nsquota>

<dsquota>-1</dsquota>

</inode>

<inode>

<id>16387</id>

<type>DIRECTORY</type>

<name>atguigu</name>

<mtime>1512790549080</mtime>

<permission>atguigu:supergroup:rwxr-xr-x</permission>

<nsquota>-1</nsquota>

<dsquota>-1</dsquota>

</inode>

<inode>

<id>16389</id>

<type>FILE</type>

<name>wc.input</name>

<replication>3</replication>

<mtime>1512722322219</mtime>

<atime>1512722321610</atime>

<perferredBlockSize>134217728</perferredBlockSize>

<permission>atguigu:supergroup:rw-r--r--</permission>

<blocks>

<block>

<id>1073741825</id>

<genstamp>1001</genstamp>

<numBytes>59</numBytes>

</block>

</blocks>

</inode >

思考：可以看出，Fsimage中没有记录块所对应DataNode，为什么？

在集群启动后，要求DataNode上报数据块信息，并间隔一段时间后再次上报。

3. oev查看Edits文件

（1）基本语法

hdfs oev -p 文件类型 -i编辑日志 -o 转换后文件输出路径

（2）案例实操

[atguigu@hadoop102 current]$ hdfs oev -p XML -i edits\_0000000000000000012-0000000000000000013 -o /opt/module/hadoop-2.7.2/edits.xml

[atguigu@hadoop102 current]$ cat /opt/module/hadoop-2.7.2/edits.xml

将显示的xml文件内容拷贝到Eclipse中创建的xml文件中，并格式化。显示结果如下。

<?xml version="1.0" encoding="UTF-8"?>

<EDITS>

<EDITS\_VERSION>-63</EDITS\_VERSION>

<RECORD>

<OPCODE>OP\_START\_LOG\_SEGMENT</OPCODE>

<DATA>

<TXID>129</TXID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_ADD</OPCODE>

<DATA>

<TXID>130</TXID>

<LENGTH>0</LENGTH>

<INODEID>16407</INODEID>

<PATH>/hello7.txt</PATH>

<REPLICATION>2</REPLICATION>

<MTIME>1512943607866</MTIME>

<ATIME>1512943607866</ATIME>

<BLOCKSIZE>134217728</BLOCKSIZE>

<CLIENT\_NAME>DFSClient\_NONMAPREDUCE\_-1544295051\_1</CLIENT\_NAME>

<CLIENT\_MACHINE>192.168.1.5</CLIENT\_MACHINE>

<OVERWRITE>true</OVERWRITE>

<PERMISSION\_STATUS>

<USERNAME>atguigu</USERNAME>

<GROUPNAME>supergroup</GROUPNAME>

<MODE>420</MODE>

</PERMISSION\_STATUS>

<RPC\_CLIENTID>908eafd4-9aec-4288-96f1-e8011d181561</RPC\_CLIENTID>

<RPC\_CALLID>0</RPC\_CALLID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_ALLOCATE\_BLOCK\_ID</OPCODE>

<DATA>

<TXID>131</TXID>

<BLOCK\_ID>1073741839</BLOCK\_ID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_SET\_GENSTAMP\_V2</OPCODE>

<DATA>

<TXID>132</TXID>

<GENSTAMPV2>1016</GENSTAMPV2>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_ADD\_BLOCK</OPCODE>

<DATA>

<TXID>133</TXID>

<PATH>/hello7.txt</PATH>

<BLOCK>

<BLOCK\_ID>1073741839</BLOCK\_ID>

<NUM\_BYTES>0</NUM\_BYTES>

<GENSTAMP>1016</GENSTAMP>

</BLOCK>

<RPC\_CLIENTID></RPC\_CLIENTID>

<RPC\_CALLID>-2</RPC\_CALLID>

</DATA>

</RECORD>

<RECORD>

<OPCODE>OP\_CLOSE</OPCODE>

<DATA>

<TXID>134</TXID>

<LENGTH>0</LENGTH>

<INODEID>0</INODEID>

<PATH>/hello7.txt</PATH>

<REPLICATION>2</REPLICATION>

<MTIME>1512943608761</MTIME>

<ATIME>1512943607866</ATIME>

<BLOCKSIZE>134217728</BLOCKSIZE>

<CLIENT\_NAME></CLIENT\_NAME>

<CLIENT\_MACHINE></CLIENT\_MACHINE>

<OVERWRITE>false</OVERWRITE>

<BLOCK>

<BLOCK\_ID>1073741839</BLOCK\_ID>

<NUM\_BYTES>25</NUM\_BYTES>

<GENSTAMP>1016</GENSTAMP>

</BLOCK>

<PERMISSION\_STATUS>

<USERNAME>atguigu</USERNAME>

<GROUPNAME>supergroup</GROUPNAME>

<MODE>420</MODE>

</PERMISSION\_STATUS>

</DATA>

</RECORD>

</EDITS >

思考：NameNode如何确定下次开机启动的时候合并哪些Edits？

### 5.1.3 CheckPoint时间设置

（1）通常情况下，SecondaryNameNode每隔一小时执行一次。

[hdfs-default.xml]

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>3600</value>

</property>

（2）一分钟检查一次操作次数，当操作次数达到1百万时，SecondaryNameNode执行一次。

<property>

<name>dfs.namenode.checkpoint.txns</name>

<value>1000000</value>

<description>操作动作次数</description>

</property>

<property>

<name>dfs.namenode.checkpoint.check.period</name>

<value>60</value>

<description> 1分钟检查一次操作次数</description>

</property >

### 5.1.4 NameNode故障处理

NameNode故障后，可以采用如下两种方法恢复数据。

**方法一：将SecondaryNameNode中数据拷贝到NameNode存储数据的目录；**

1. kill -9 NameNode进程

2. 删除NameNode存储的数据（/opt/module/hadoop-2.7.2/data/tmp/dfs/name）

[atguigu@hadoop102 hadoop-2.7.2]$ rm -rf /opt/module/hadoop-2.7.2/data/tmp/dfs/name/\*

3. 拷贝SecondaryNameNode中数据到原NameNode存储数据目录

[atguigu@hadoop102 dfs]$ scp -r atguigu@hadoop104:/opt/module/hadoop-2.7.2/data/tmp/dfs/namesecondary/\* ./name/

4. 重新启动NameNode

[atguigu@hadoop102 hadoop-2.7.2]$ sbin/hadoop-daemon.sh start namenode

**方法二：使用-importCheckpoint选项启动NameNode守护进程，从而将SecondaryNameNode中数据拷贝到NameNode目录中。**

1. 修改hdfs-site.xml中的

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>120</value>

</property>

<property>

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

<value>/opt/module/hadoop-2.7.2/data/tmp/dfs/name</value>

</property>

2. kill -9 NameNode进程

3. 删除NameNode存储的数据（/opt/module/hadoop-2.7.2/data/tmp/dfs/name）

[atguigu@hadoop102 hadoop-2.7.2]$ rm -rf /opt/module/hadoop-2.7.2/data/tmp/dfs/name/\*

4. 如果SecondaryNameNode不和NameNode在一个主机节点上，需要将SecondaryNameNode存储数据的目录拷贝到NameNode存储数据的平级目录，并删除in\_use.lock文件

[at@hadoop102 dfs]$ scp -r atguigu@hadoop104:/opt/module/hadoop-2.7.2/data/tmp/dfs/namesecondary ./

[at@hadoop102 namesecondary]$ rm -rf in\_use.lock

[at@hadoop102 dfs]$ pwd

/opt/module/hadoop-2.7.2/data/tmp/dfs

[at@hadoop102 dfs]$ ls

data name namesecondary

5. 导入检查点数据（等待一会ctrl+c结束掉）

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hdfs namenode -importCheckpoint

6. 启动NameNode

[atguigu@hadoop102 hadoop-2.7.2]$ sbin/hadoop-daemon.sh start namenode

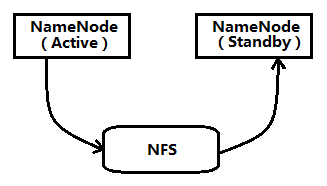
## 5.2 HDFS 2.0

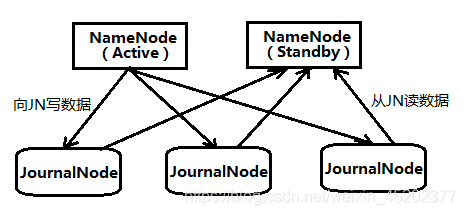
Hadoop2.0引入了HA机制（高可用），HA机制有两个NameNode，一个是Active状态，另一个是Standby状态。两者的状态可以切换，但同时最多只有1个是Active状态。只有Active Namenode提供对外的服务。Active NameNode和Standby NameNode之间通过NFS或者JN（JournalNode，QJM方式）来同步数据。

Active NameNode会把最近的操作记录写到本地的一个edits文件中（edits file），并传输到NFS或者JN中。Standby NameNode定期的检查，从NFS或者JN把最近的edit文件读过来，然后把edits文件和fsimage文件合并成一个新的fsimage，合并完成之后会通知Active NameNode获取这个新fsimage。Active NameNode获得这个新的fsimage文件之后，替换原来旧的fsimage文件。

这样，保持了Active NameNode和Standby NameNode的数据实时同步，Standby NameNode可以随时切换成Active NameNode（譬如Active NameNode挂了），而且还有一个原来Hadoop1.0才有的SecondaryNameNode，CheckpointNode，BackupNode的功能：合并edits文件和fsimage文件，使fsimage文件一直保持更新。所以启动了hadoop2.0的HA机制之后，SecondaryNameNode、CheckpointNode、BackupNode这些都不需要了。

**数据同步方式**

1、NFS  
  
 NFS作为Active NameNode和Standby NameNode之间数据共享的存储。Active NameNode会把最近的edits文件写到NFS，而Standby NameNode从NFS中把数据读过来。这个方式的缺点是，如果Active NameNode或者Standby Namenode有一个和NFS之间网络有问题，则会造成他们之前数据的同步出问题。

2、 QJM（Quorum Journal Manager ）  
  
 QJM的方式可以解决上述NFS容错机制不足的问题。Active NameNode和Standby NameNode之间是通过一组JournalNode（数量是奇数，可以是3,5,7…,2n+1）来共享数据。Active NameNode把最近的edits文件写到2n+1个JournalNode上，只要有n+1个写入成功就认为这次写入操作成功了，然后Standby NameNode就可以从JournalNode上读取了。可以看到，QJM方式有容错机制，可以容忍n个JournalNode的失败。

Active和Standby两个NameNode之间的数据交互流程为：

1）NameNode在启动后，会先加载Fsimage文件和共享目录上的EditLog Segment文件；

2）Standby NameNode会启动EditLogTailer线程和StandbyCheckpointer线程，正式进入Standby模式；

3）Active NameNode把EditLog提交到JournalNode集群；

4）Standby NameNode上的EditLogTailer线程定时从JournalNode集群上同步EditLog；

5）Standby NameNode上的StandbyCheckpointer线程定时进行Checkpoint，并将Checkpoint之后的Fsimage文件上传到Active NameNode。（在Hadoop 2.0中不再有Secondary NameNode这个角色了，StandbyCheckpointer线程的作用其实是为了替代Hadoop 1.0版本中的Secondary NameNode的功能。）

QJM方式有明显的优点，一是本身就有fencing的功能，二是通过多个Journal节点增强了系统的健壮性，所以建议在生产环境中采用QJM的方式。JournalNode消耗的资源很少，不需要额外的机器专门来启动JournalNode，可以从Hadoop集群中选几台机器同时作为JournalNode。

主备NameNode切换  
  
 Active NameNode和Standby NameNode可以随时切换，既可以人工，也可以自动。人工切换是通过执行HA管理命令来改变NameNode的状态，从Standby到Active，或从Active到Standby。自动切换则在Active NameNode挂掉的时候，Standby NameNode自动切换成Active状态。

主备NameNode的自动切换需要配置Zookeeper。Active NameNode和Standby NameNode把他们的状态实时记录到Zookeeper中，Zookeeper监视他们的状态变化。当Zookeeper发现Active NameNode挂掉后，会自动把Standby NameNode切换成Active NameNode。

## 5.3 集群安全模式

1. 安全模式

1）什么时候集群会进入安全模式？

NameNode启动时：在NameNode启动的时候，他会加载俩个文件，一个是Fsimage，一个是Edits。这个俩个文件里面保存的是HDFS运行以来保存的全部元数据信息。在刚启动集群时，NameNode需要将历史的元数据加载到内存中，然后才可以继续工作。而这个时候集群是出于安全模式，用户对集群是出于只读模式。

DataNode启动时：DataNode中数据的块信息不是在NameNode中持久保存的，而是在集群启动时候，DataNode主动将数据的块信息同步到NameNode中，而执行这个同步的过程时，集群时处于安全模式的。

2）什么时候会退出安全模式？

满足“最小副本条件”，NameNode会在30秒钟后退出安全模式。所谓的最小副本条件指的是在整个文件系统中99.9%的块满足最小副本级别（默认值：dfs.replication.min=1）。。也就是说，在启动集群时，内存中加载文件系统中之前的数据块，每个块满足了配置文件中最小的副本数的条件，安全模式就会退出。在启动一个刚刚格式化的HDFS集群时，因为系统中还没有任何块，所以NameNode不会进入安全模式。

3）安全模式有什么用呢？

安全模式的用处在于“安全”这俩字。正是因为安全模式的存在，用户在数据未恢复完整之前不可以对其进行操作，进而避免了数据更新时会出现的一些错误。

4）NameNode启动

NameNode启动时，首先将镜像文件（Fsimage）载入内存，并执行编辑日志（Edits）中的各项操作。一旦在内存中成功建立文件系统元数据的映像，则创建一个新的Fsimage文件和一个空的编辑日志。此时，NameNode开始监听DataNode请求。这个过程期间，NameNode一直运行安全模式，即NameNode的文件系统对于客户端来说是只读的。

5）DataNode启动

系统中的数据块的位置并不是由NameNode维护的，而是以块列表的形式存储在DataNode中。在系统的正常操作期间，NameNode会在内存中保留所有块位置的映射信息。在安全模式下，各个DataNode会向NameNode发送最新的块列表信息，NameNode了解到足够多的块位置信息之后，即可高效运行文件系统。

2. 基本语法

集群处于安全模式，不能执行重要操作（写操作）。集群启动完成后，自动退出安全模式。

（1）bin/hdfs dfsadmin -safemode get （功能描述：查看安全模式状态）

（2）bin/hdfs dfsadmin -safemode enter （功能描述：进入安全模式状态）

（3）bin/hdfs dfsadmin -safemode leave （功能描述：离开安全模式状态）

（4）bin/hdfs dfsadmin -safemode wait （功能描述：等待安全模式状态）

3. 案例

模拟等待安全模式

（1）查看当前模式

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -safemode get

Safe mode is OFF

（2）先进入安全模式

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hdfs dfsadmin -safemode enter

（3）创建并执行下面的脚本

在/opt/module/hadoop-2.7.2路径上，编辑一个脚本safemode.sh

[atguigu@hadoop102 hadoop-2.7.2]$ touch safemode.sh

[atguigu@hadoop102 hadoop-2.7.2]$ vim safemode.sh

#!/bin/bash

hdfs dfsadmin -safemode wait

hdfs dfs -put /opt/module/hadoop-2.7.2/README.txt /

[atguigu@hadoop102 hadoop-2.7.2]$ chmod 777 safemode.sh

[atguigu@hadoop102 hadoop-2.7.2]$ ./safemode.sh

（4）再打开一个窗口，执行

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hdfs dfsadmin -safemode leave

（5）观察

（a）再观察上一个窗口

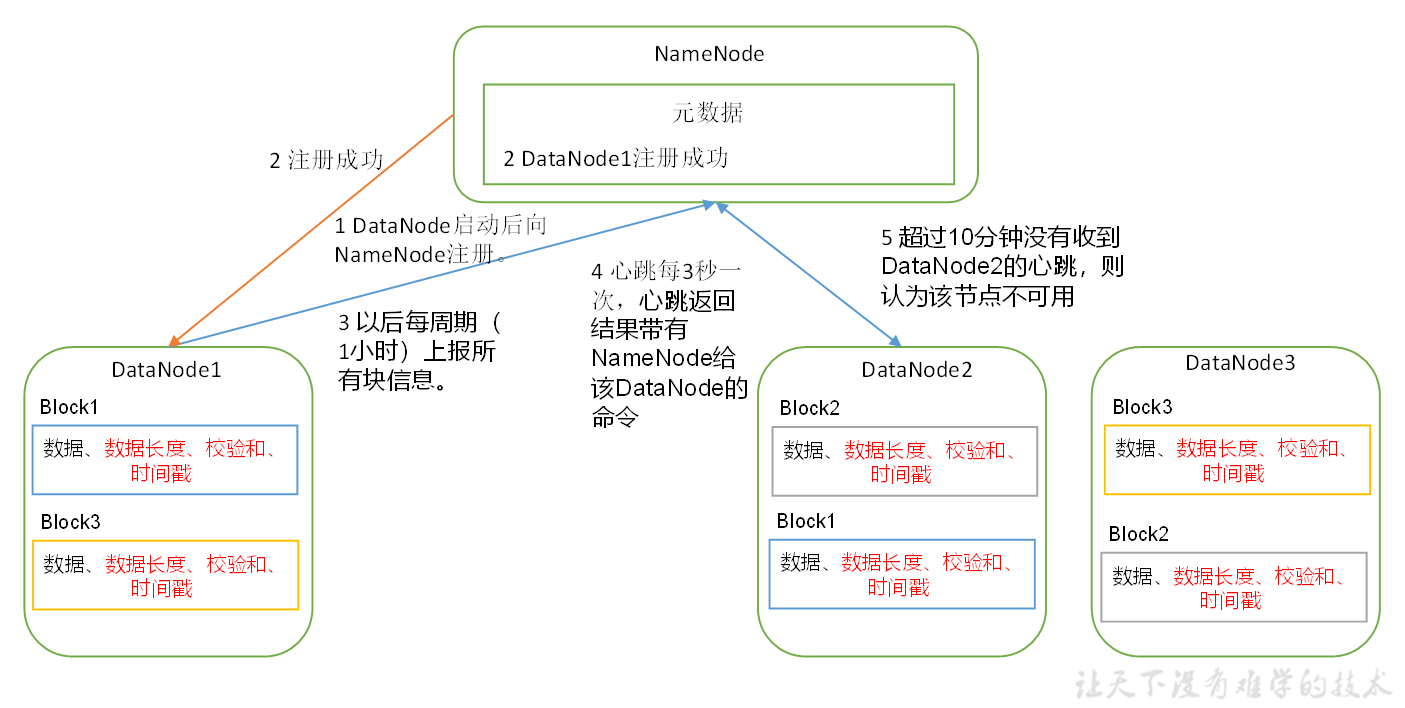
Safe mode is OFF

（b）HDFS集群上已经有上传的数据了。

# 第6章 DataNode（面试开发重点）

## 6.1 DataNode工作机制

DataNode工作机制，如图3-15所示。



1）一个数据块在DataNode上以文件形式存储在磁盘上，包括两个文件：一个是数据本身，一个是元数据包括数据块的长度、校验和以及时间戳。

2）DataNode启动后向NameNode注册，通过后，周期性（每1小时）的向NameNode上报所有的块信息。

3）心跳是每3秒一次，心跳返回结果带有NameNode给该DataNode的命令（比如复制块数据到另一台机器，或删除某个数据块）。如果超过10分钟没有收到某个DataNode的心跳，则认为该节点不可用。

4）集群运行中可以安全加入和退出一些机器。

## 6.2 DataNode数据完整性

思考：如果电脑磁盘里面存储的数据是控制高铁信号灯的红灯信号（1）和绿灯信号（0），但是存储该数据的磁盘坏了，一直显示是绿灯，是否很危险？同理DataNode节点上的数据损坏了，却没有发现，是否也很危险，那么如何解决呢？

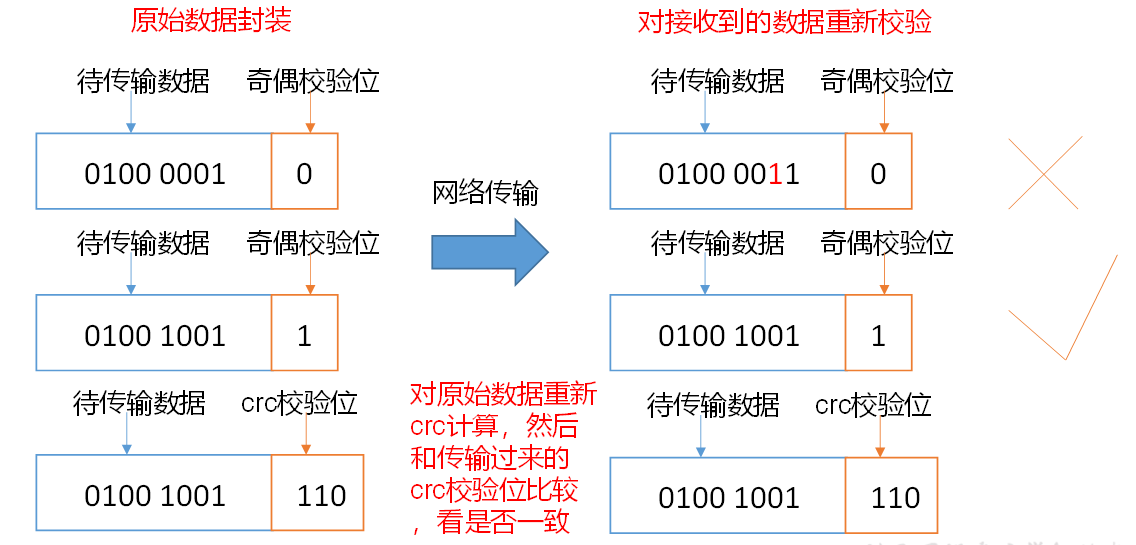
如下是DataNode节点保证数据完整性的方法：

1）当DataNode读取Block的时候，它会计算CheckSum。

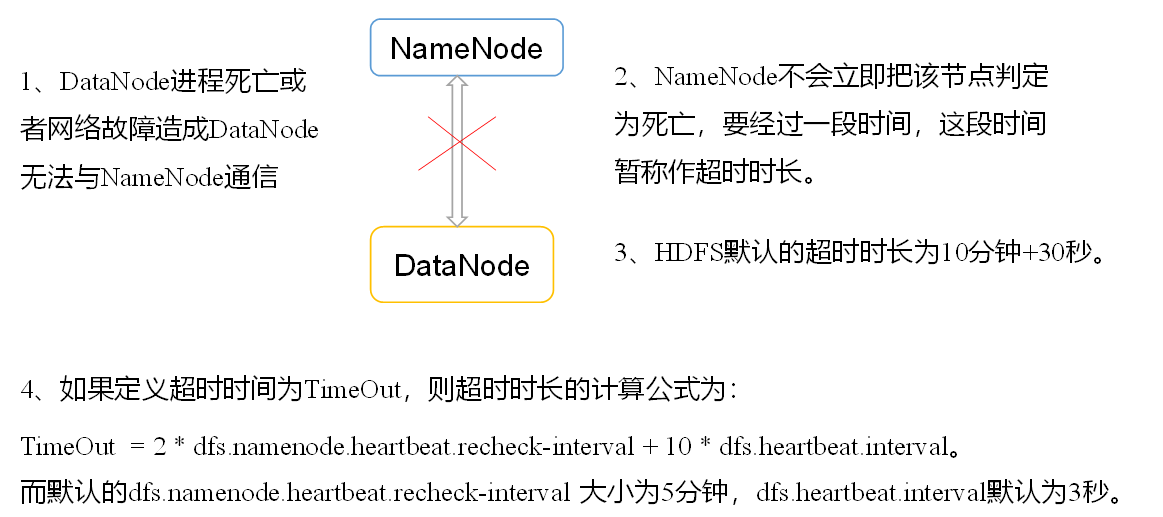
2）如果计算后的CheckSum与Block创建时值不一样时说明Block已经损坏。

3）Client读取其他DataNode上的Block。

4）DataNode在其文件创建后周期验证CheckSum，如图3-16所示。



## 6.3 DataNode掉线时限参数设置



需要注意的是hdfs-site.xml配置文件中的heartbeat.recheck.interval的单位为毫秒，dfs.heartbeat.interval的单位为秒。

<property>

<name>dfs.namenode.heartbeat.recheck-interval</name>

<value>300000</value>

</property>

<property>

<name>dfs.heartbeat.interval</name>

<value>3</value>

</property>

## 6.4 服役新数据节点

0. 需求

随着公司业务的增长，数据量越来越大，原有的数据节点的容量已经不能满足存储数据的需求，需要在原有集群基础上动态添加新的数据节点。

1. 环境准备

（1）在hadoop104主机上再克隆一台hadoop105主机

（2）修改IP地址和主机名称

（3）**删除原来HDFS文件系统留存的文件（/opt/module/hadoop-2.7.2/data和log）**

（4）source一下配置文件

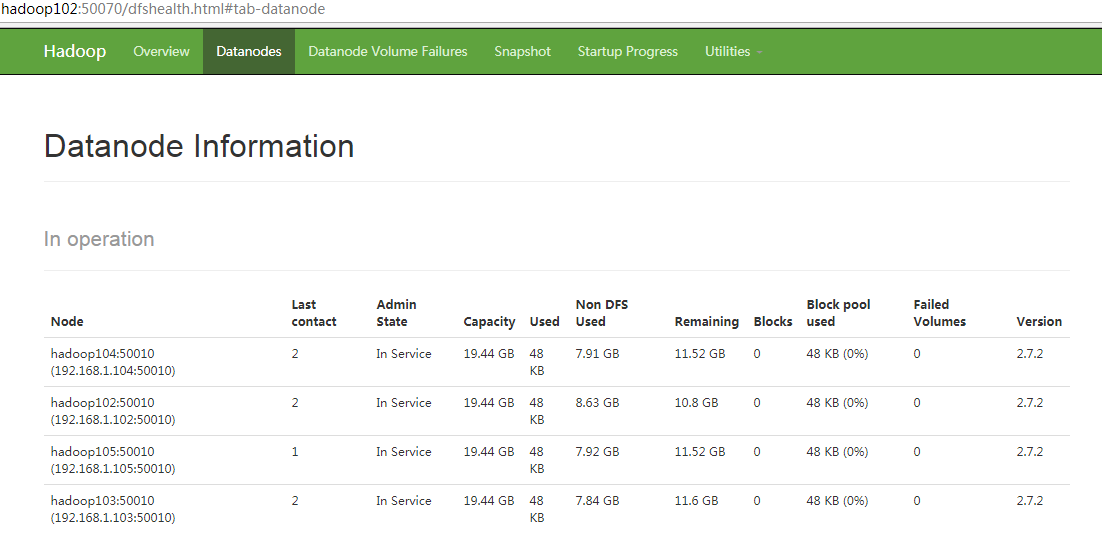
[atguigu@hadoop105 hadoop-2.7.2]$ source /etc/profile

2. 服役新节点具体步骤

（1）直接启动DataNode，即可关联到集群

[atguigu@hadoop105 hadoop-2.7.2]$ sbin/hadoop-daemon.sh start datanode

[atguigu@hadoop105 hadoop-2.7.2]$ sbin/yarn-daemon.sh start nodemanager



（2）在hadoop105上上传文件

[atguigu@hadoop105 hadoop-2.7.2]$ hadoop fs -put /opt/module/hadoop-2.7.2/LICENSE.txt /

（3）如果数据不均衡，可以用命令实现集群的再平衡

[atguigu@hadoop102 sbin]$ ./start-balancer.sh

starting balancer, logging to /opt/module/hadoop-2.7.2/logs/hadoop-atguigu-balancer-hadoop102.out

Time Stamp Iteration# Bytes Already Moved Bytes Left To Move Bytes Being Moved

## 6.5 退役旧数据节点

### 6.5.1 添加白名单

添加到白名单的主机节点，都允许访问NameNode，不在白名单的主机节点，都会被退出。

配置白名单的具体步骤如下：

（1）在NameNode的/opt/module/hadoop-2.7.2/etc/hadoop目录下创建dfs.hosts文件

[atguigu@hadoop102 hadoop]$ pwd

/opt/module/hadoop-2.7.2/etc/hadoop

[atguigu@hadoop102 hadoop]$ touch dfs.hosts

[atguigu@hadoop102 hadoop]$ vi dfs.hosts

添加如下主机名称（不添加hadoop105）

hadoop102

hadoop103

hadoop104

（2）在NameNode的hdfs-site.xml配置文件中增加dfs.hosts属性

<property>

<name>dfs.hosts</name>

<value>/opt/module/hadoop-2.7.2/etc/hadoop/dfs.hosts</value>

</property>

（3）配置文件分发

[atguigu@hadoop102 hadoop]$ xsync hdfs-site.xml

（4）刷新NameNode

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -refreshNodes

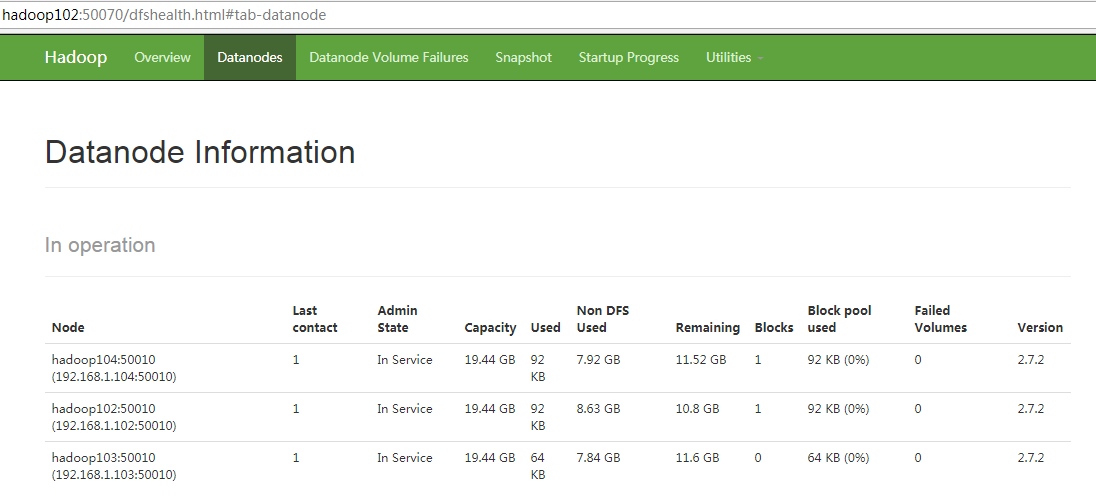
Refresh nodes successful

（5）更新ResourceManager节点

[atguigu@hadoop102 hadoop-2.7.2]$ yarn rmadmin -refreshNodes

17/06/24 14:17:11 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8033

（6）在web浏览器上查看



如果数据不均衡，可以用命令实现集群的再平衡

[atguigu@hadoop102 sbin]$ ./start-balancer.sh

starting balancer, logging to /opt/module/hadoop-2.7.2/logs/hadoop-atguigu-balancer-hadoop102.out

Time Stamp Iteration# Bytes Already Moved Bytes Left To Move Bytes Being Moved

### 6.5.2 黑名单退役

在黑名单上面的主机都会被强制退出。

1．在NameNode的/opt/module/hadoop-2.7.2/etc/hadoop目录下创建dfs.hosts.exclude文件

[atguigu@hadoop102 hadoop]$ pwd

/opt/module/hadoop-2.7.2/etc/hadoop

[atguigu@hadoop102 hadoop]$ touch dfs.hosts.exclude

[atguigu@hadoop102 hadoop]$ vi dfs.hosts.exclude

添加如下主机名称（要退役的节点）

hadoop105

2．在NameNode的hdfs-site.xml配置文件中增加dfs.hosts.exclude属性

<property>

<name>dfs.hosts.exclude</name>

<value>/opt/module/hadoop-2.7.2/etc/hadoop/dfs.hosts.exclude</value>

</property>

3．刷新NameNode、刷新ResourceManager

[atguigu@hadoop102 hadoop-2.7.2]$ hdfs dfsadmin -refreshNodes

Refresh nodes successful

[atguigu@hadoop102 hadoop-2.7.2]$ yarn rmadmin -refreshNodes

17/06/24 14:55:56 INFO client.RMProxy: Connecting to ResourceManager at hadoop103/192.168.1.103:8033

4．检查Web浏览器，退役节点的状态为decommission in progress（退役中），说明数据节点正在复制块到其他节点，如图3-17所示

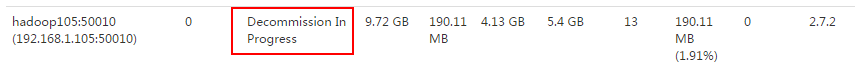


图3-17 退役中

5．等待退役节点状态为decommissioned（所有块已经复制完成），停止该节点及节点资源管理器。注意：如果副本数是3，服役的节点小于等于3，是不能退役成功的，需要修改副本数后才能退役，如图3-18所示

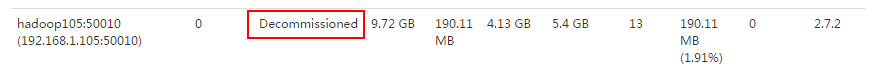


图3-18 已退役

[atguigu@hadoop105 hadoop-2.7.2]$ sbin/hadoop-daemon.sh stop datanode

stopping datanode

[atguigu@hadoop105 hadoop-2.7.2]$ sbin/yarn-daemon.sh stop nodemanager

stopping nodemanager

6．如果数据不均衡，可以用命令实现集群的再平衡

[atguigu@hadoop102 hadoop-2.7.2]$ sbin/start-balancer.sh

starting balancer, logging to /opt/module/hadoop-2.7.2/logs/hadoop-atguigu-balancer-hadoop102.out

Time Stamp Iteration# Bytes Already Moved Bytes Left To Move Bytes Being Moved

注意：不允许白名单和黑名单中同时出现同一个主机名称。

## 6.6 Datanode多目录配置

1．DataNode可以配置成多个目录，每个目录存储的数据不一样。即：数据不是副本

2. 为何要配置DataNode多目录？

DataNode配置多目录是因为一个目录下面挂的硬盘不够用，所以多加了几个目录。这里要和副本数区分开，副本数是配置hdfs中数据存放的份数，也就是备份数，防止数据丢失的。

3. 具体配置如下

在hdfs-site.xml文件中增加如下内容：

<property>

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

<value>file:///${hadoop.tmp.dir}/dfs/data1,file:///${hadoop.tmp.dir}/dfs/data2</value>

</property>

4. 为什么要配置NameNode多目录？

多目录配置是为了增加集群的安全性，可以将第一个磁盘挂载到第一个目录上，第二个磁盘挂载到第二个目录上。这样当其中一块磁盘出现故障时，NameNode储存元数据也不会丢失。

5. 具体配置如下

在hdfs-site.xml文件中增加如下内容：

<property>

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

<value>file:///${hadoop.tmp.dir}/dfs/name1,file:///${hadoop.tmp.dir}/dfs/name2</value>

</property>

# 第7章 HDFS 2.X新特性

## 7.1 集群间数据拷贝

1．scp实现两个远程主机之间的文件复制

scp -r hello.txt [root@hadoop103:/user/atguigu/hello.txt](mailto:root@hadoop103:/user/atguigu/hello.txt) // 推 push

scp -r [root@hadoop103:/user/atguigu/hello.txt hello.txt](mailto:root@hadoop103:/user/atguigu/hello.txt%20%20hello.txt) // 拉 pull

scp -r [root@hadoop103:/user/atguigu/hello.txt](mailto:root@hadoop103:/user/atguigu/hello.txt) root@hadoop104:/user/atguigu //是通过本地主机中转实现两个远程主机的文件复制；如果在两个远程主机之间ssh没有配置的情况下可以使用该方式。

2．采用distcp命令实现两个Hadoop集群之间的递归数据复制

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop distcp

hdfs://haoop102:9000/user/atguigu/hello.txt hdfs://hadoop103:9000/user/atguigu/hello.txt

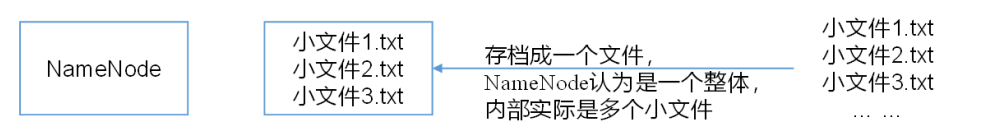
## 7.2 小文件存档

1、HDFS存储小文件弊端

每个文件均按块存储，每个块的元数据存储在NameNode的内存中，因此HDFS存储小文件会非常低效。因为大量的小文件会耗尽NameNode中的大部分内存。但注意，存储小文件所需要的磁盘容量和数据块的大小无关。例如，一个1MB的文件设置128MB的块存储，实际使用的是1MB的磁盘空间，而不是128MB。

1. 解决存储小文件办法之一

HDFS存档文件或者HAR文件，是一个高效的文件存档工具，它将文件存入HDFS块，在减少NameNode内存使用的同时，允许对文件透明的访问。具体来说，HDFS存档文件对内还是一个一个独立文件，对NameNode而言却是一个整体，减少NameNode的内存。



3．案例实操

（1）需要启动YARN进程

[atguigu@hadoop102 hadoop-2.7.2]$ start-yarn.sh

（2）归档文件

把/user/atguigu/input目录里面的所有文件归档成一个叫input.har的归档文件，并把归档后文件存储到/user/atguigu/output路径下。

[atguigu@hadoop102 hadoop-2.7.2]$ bin/hadoop archive -archiveName input.har –p /user/atguigu/input /user/atguigu/output

（3）查看归档

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -lsr /user/atguigu/output/input.har

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -lsr har:///user/atguigu/output/input.har

（4）解归档文件

[atguigu@hadoop102 hadoop-2.7.2]$ hadoop fs -cp har:/// user/atguigu/output/input.har/\* /user/atguigu

# 第8章 HDFS HA高可用

## 8.1 HA概述

1）所谓HA（High Available），即高可用（7\*24小时不中断服务）。

2）实现高可用关键就是消除单点故障。HA严格来说应分成各个组件的HA机制：HDFS的HA和YARN的HA。

3）Hadoop2.0之前，在HDFS集群中NameNode存在单点故障（SPOF）。

4）NameNode主要在以下两个方面影响HDFS集群

NameNode机器发生意外，如宕机，集群将无法使用，直到管理员重启

NameNode机器需要升级，包括软件、硬件升级，此时集群也将无法使用

HDFS HA功能通过配置Active/Standby两个NameNodes实现在集群中对NameNode的热备来解决上述问题。如果出现故障，如机器崩溃或机器需要升级维护，这时可通过此种方式将NameNode很快的切换到另外一台机器。

## 8.2 HDFS-HA工作机制

通过双NameNode消除单点故障。

### 8.2.1 HDFS-HA工作要点

1. 元数据管理方式需要改变

内存中各自保存一份元数据；

Edits日志只有Active状态的NameNode节点可以做写操作；

两个NameNode都可以读取Edits；

共享的Edits放在一个共享存储中管理（qjournal和NFS两个主流实现）；

2. 需要一个状态管理功能模块

实现了一个zkfailover，常驻在每一个namenode所在的节点，每一个zkfailover负责监控自己所在NameNode节点，利用zk进行状态标识，当需要进行状态切换时，由zkfailover来负责切换，切换时需要防止brain split现象的发生。

3. 必须保证两个NameNode之间能够ssh无密码登录

4. 隔离（Fence），即同一时刻仅仅有一个NameNode对外提供服务

### 8.2.2 HDFS-HA自动故障转移工作机制

前面学习了使用命令hdfs haadmin -failover手动进行故障转移，在该模式下即使现役NameNode已经失效，系统也不会自动从现役NameNode转移到待机NameNode，下面学习如何配置部署HA自动进行故障转移。自动故障转移为HDFS部署增加了两个新组件：ZooKeeper和ZKFailoverController（ZKFC）进程，如图3-20所示。ZooKeeper是维护少量协调数据，通知客户端这些数据的改变和监视客户端故障的高可用服务。HA的自动故障转移依赖于ZooKeeper的以下功能：

1）**故障检测**：集群中的每个NameNode在ZooKeeper中维护了一个持久会话，如果机器崩溃，ZooKeeper中的会话将终止，ZooKeeper通知另一个NameNode需要触发故障转移。

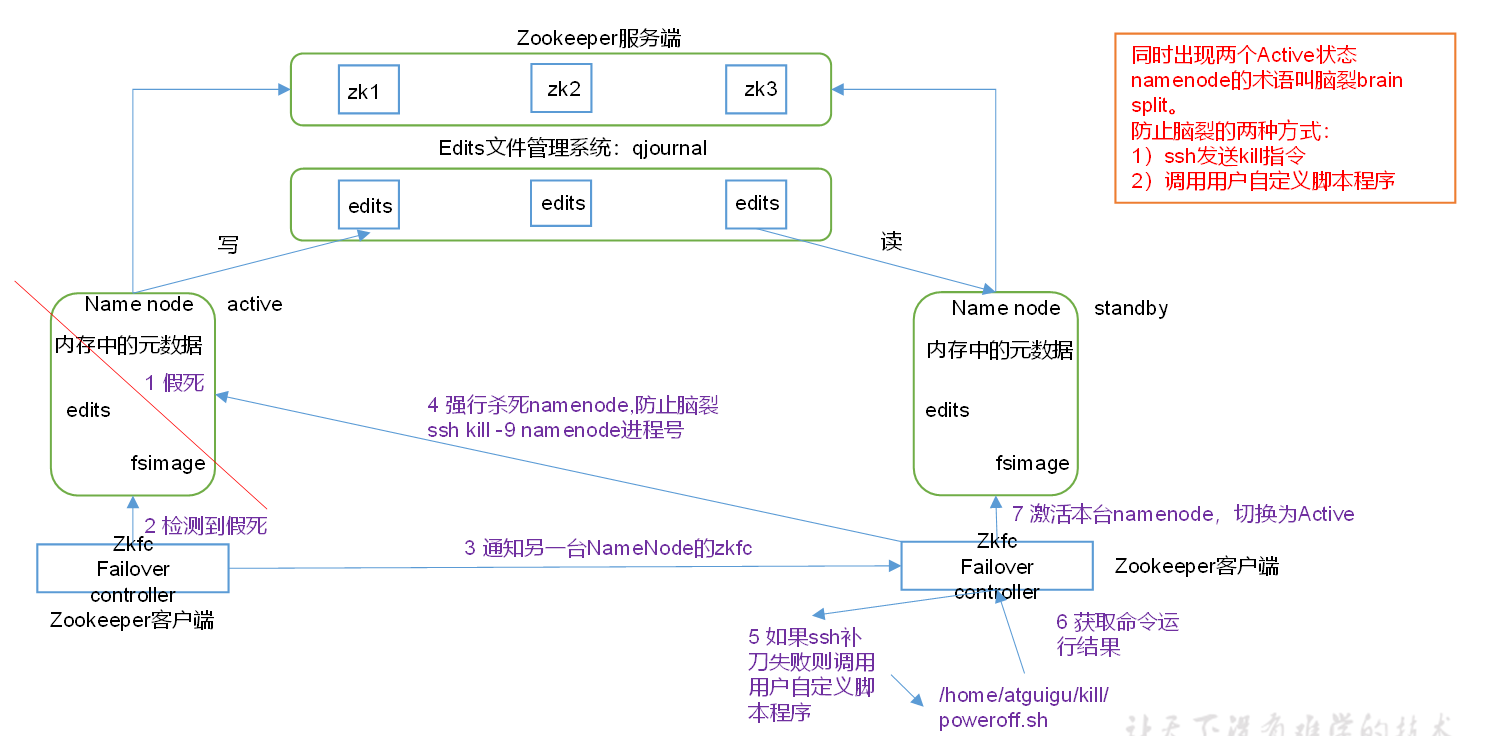
2）**现役NameNode选择**：ZooKeeper提供了一个简单的机制用于唯一的选择一个节点为active状态。如果目前现役NameNode崩溃，另一个节点可能从ZooKeeper获得特殊的排外锁以表明它应该成为现役NameNode。

ZKFC是自动故障转移中的另一个新组件，是ZooKeeper的客户端，也监视和管理NameNode的状态。每个运行NameNode的主机也运行了一个ZKFC进程，ZKFC负责：

**1）健康监测：**ZKFC使用一个健康检查命令定期地ping与之在相同主机的NameNode，只要该NameNode及时地回复健康状态，ZKFC认为该节点是健康的。如果该节点崩溃，冻结或进入不健康状态，健康监测器标识该节点为非健康的。

**2）ZooKeeper会话管理：**当本地NameNode是健康的，ZKFC保持一个在ZooKeeper中打开的会话。如果本地NameNode处于active状态，ZKFC也保持一个特殊的znode锁，该锁使用了ZooKeeper对短暂节点的支持，如果会话终止，锁节点将自动删除。

**3）基于ZooKeeper的选择：**如果本地NameNode是健康的，且ZKFC发现没有其它的节点当前持有znode锁，它将为自己获取该锁。如果成功，则它已经赢得了选择，并负责运行故障转移进程以使它的本地NameNode为Active。故障转移进程与前面描述的手动故障转移相似，首先如果必要保护之前的现役NameNode，然后本地NameNode转换为Active状态。



## 8.3 HDFS-HA集群配置

### 8.3.1 环境准备

1. 修改IP

2. 修改主机名及主机名和IP地址的映射

3. 关闭防火墙

4. ssh免密登录

5. 安装JDK，配置环境变量等

### 8.3.2 规划集群

表3-1

|  |  |  |
| --- | --- | --- |
| hadoop102 | hadoop103 | hadoop104 |
| NameNode | NameNode |  |
| JournalNode | JournalNode | JournalNode |
| DataNode | DataNode | DataNode |
| ZK | ZK | ZK |
|  | ResourceManager |  |
| NodeManager | NodeManager | NodeManager |

### 8.3.3 配置Zookeeper集群

1. 集群规划

在hadoop102、hadoop103和hadoop104三个节点上部署Zookeeper。

2. 解压安装

（1）解压Zookeeper安装包到/opt/module/目录下

[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/

（2）在/opt/module/zookeeper-3.4.10/这个目录下创建zkData

mkdir -p zkData

（3）重命名/opt/module/zookeeper-3.4.10/conf这个目录下的zoo\_sample.cfg为zoo.cfg

mv zoo\_sample.cfg zoo.cfg

3. 配置zoo.cfg文件

（1）具体配置

dataDir=/opt/module/zookeeper-3.4.10/zkData

增加如下配置

#######################cluster##########################

server.2=hadoop102:2888:3888

server.3=hadoop103:2888:3888

server.4=hadoop104:2888:3888

（2）配置参数解读

Server.A=B:C:D。

A是一个数字，表示这个是第几号服务器；

B是这个服务器的IP地址；

C是这个服务器与集群中的Leader服务器交换信息的端口；

D是万一集群中的Leader服务器挂了，需要一个端口来重新进行选举，选出一个新的Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

集群模式下配置一个文件myid，这个文件在dataDir目录下，这个文件里面有一个数据就是A的值，Zookeeper启动时读取此文件，拿到里面的数据与zoo.cfg里面的配置信息比较从而判断到底是哪个server。

4. 集群操作

（1）在/opt/module/zookeeper-3.4.10/zkData目录下创建一个myid的文件

touch myid

添加myid文件，注意一定要在linux里面创建，在notepad++里面很可能乱码

（2）编辑myid文件

vi myid

在文件中添加与server对应的编号：如2

（3）拷贝配置好的zookeeper到其他机器上

scp -r zookeeper-3.4.10/ [root@hadoop103.atguigu.com:/opt/app/](mailto:root@hadoop103.atguigu.com:/opt/app/)

scp -r zookeeper-3.4.10/ [root@hadoop104.atguigu.com:/opt/app/](mailto:root@hadoop104.atguigu.com:/opt/app/)

并分别修改myid文件中内容为3、4

（4）分别启动zookeeper

[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh start

[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh start

[root@hadoop104 zookeeper-3.4.10]# bin/zkServer.sh start

（5）查看状态

[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: leader

[root@hadoop104 zookeeper-3.4.5]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

### 8.3.4 配置HDFS-HA集群

1. 官方地址：<http://hadoop.apache.org/>

2. 在opt目录下创建一个ha文件夹

mkdir ha

3. 将/opt/app/下的 hadoop-2.7.2拷贝到/opt/ha目录下

cp -r hadoop-2.7.2/ /opt/ha/

4. 配置hadoop-env.sh

|  |
| --- |
| export JAVA\_HOME=/opt/module/jdk1.8.0\_144 |

5. 配置core-site.xml

|  |
| --- |
| <configuration>  <!-- 把两个NameNode）的地址组装成一个集群mycluster -->  <property>  <name>fs.defaultFS</name>  <value>hdfs://mycluster</value>  </property>  <!-- 指定hadoop运行时产生文件的存储目录 -->  <property>  <name>hadoop.tmp.dir</name>  <value>/opt/ha/hadoop-2.7.2/data/tmp</value>  </property>  </configuration> |

6. 配置hdfs-site.xml

|  |
| --- |
| <configuration>  <!-- 完全分布式集群名称 -->  <property>  <name>dfs.nameservices</name>  <value>mycluster</value>  </property>  <!-- 集群中NameNode节点都有哪些 -->  <property>  <name>dfs.ha.namenodes.mycluster</name>  <value>nn1,nn2</value>  </property>  <!-- nn1的RPC通信地址 -->  <property>  <name>dfs.namenode.rpc-address.mycluster.nn1</name>  <value>hadoop102:9000</value>  </property>  <!-- nn2的RPC通信地址 -->  <property>  <name>dfs.namenode.rpc-address.mycluster.nn2</name>  <value>hadoop103:9000</value>  </property>  <!-- nn1的http通信地址 -->  <property>  <name>dfs.namenode.http-address.mycluster.nn1</name>  <value>hadoop102:50070</value>  </property>  <!-- nn2的http通信地址 -->  <property>  <name>dfs.namenode.http-address.mycluster.nn2</name>  <value>hadoop103:50070</value>  </property>  <!-- 指定NameNode元数据在JournalNode上的存放位置 -->  <property>  <name>dfs.namenode.shared.edits.dir</name>  <value>qjournal://hadoop102:8485;hadoop103:8485;hadoop104:8485/mycluster</value>  </property>  <!-- 配置隔离机制，即同一时刻只能有一台服务器对外响应 -->  <property>  <name>dfs.ha.fencing.methods</name>  <value>sshfence</value>  </property>  <!-- 使用隔离机制时需要ssh无秘钥登录-->  <property>  <name>dfs.ha.fencing.ssh.private-key-files</name>  <value>/home/atguigu/.ssh/id\_rsa</value>  </property>  <!-- 声明journalnode服务器存储目录-->  <property>  <name>dfs.journalnode.edits.dir</name>  <value>/opt/ha/hadoop-2.7.2/data/jn</value>  </property>  <!-- 关闭权限检查-->  <property>  <name>dfs.permissions.enable</name>  <value>false</value>  </property>  <!-- 访问代理类：client，mycluster，active配置失败自动切换实现方式-->  <property>  <name>dfs.client.failover.proxy.provider.mycluster</name>  <value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>  </property>  </configuration> |

7. 拷贝配置好的hadoop环境到其他节点

### 8.3.5 启动HDFS-HA集群

1. 在各个JournalNode节点上，输入以下命令启动journalnode服务

sbin/hadoop-daemon.sh start journalnode

2. 在[nn1]上，对其进行格式化，并启动

bin/hdfs namenode -format

sbin/hadoop-daemon.sh start namenode

3. 在[nn2]上，同步nn1的元数据信息

bin/hdfs namenode -bootstrapStandby

4. 启动[nn2]

sbin/hadoop-daemon.sh start namenode

5. 查看web页面显示，如图3-21，3-22所示

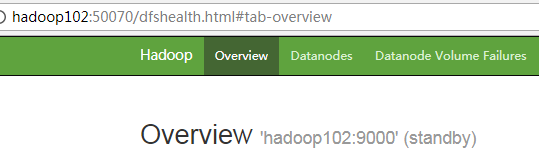


图3-21 hadoop102(standby)

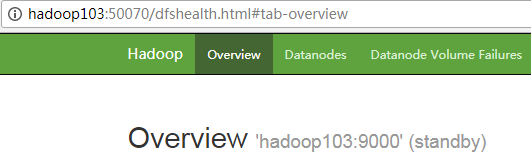


图3-22 hadoop103(standby)

6. 在[nn1]上，启动所有datanode

sbin/hadoop-daemons.sh start datanode

7. 将[nn1]切换为Active

bin/hdfs haadmin -transitionToActive nn1

8. 查看是否Active

bin/hdfs haadmin -getServiceState nn1

### 8.3.6 配置HDFS-HA自动故障转移

1.具体配置

（1）在hdfs-site.xml中增加

<property>

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

<value>true</value>

</property>

（2）在core-site.xml文件中增加

<property>

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

<value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value>

</property>

2.启动

（1）关闭所有HDFS服务：

sbin/stop-dfs.sh

（2）启动Zookeeper集群：

bin/zkServer.sh start

（3）初始化HA在Zookeeper中状态：

bin/hdfs zkfc -formatZK

（4）启动HDFS服务：

sbin/start-dfs.sh

3.验证

（1）将Active NameNode进程kill

kill -9 namenode的进程id

（2）将Active NameNode机器断开网络

service network stop

## 8.4 YARN-HA配置

### 8.4.1 YARN-HA工作机制

1. 官方文档：

<http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html>

2. YARN-HA工作机制，如图3-23所示

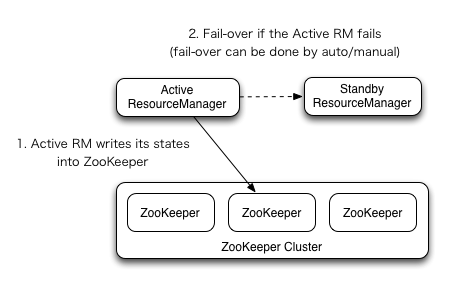


图3-22 YARN-HA工作机制

### 8.4.2 配置YARN-HA集群

1. 环境准备

（1）修改IP

（2）修改主机名及主机名和IP地址的映射

（3）关闭防火墙

（4）ssh免密登录

（5）安装JDK，配置环境变量等

（6）配置Zookeeper集群

2. 规划集群

表3-2

|  |  |  |
| --- | --- | --- |
| hadoop102 | hadoop103 | hadoop104 |
| NameNode | NameNode |  |
| JournalNode | JournalNode | JournalNode |
| DataNode | DataNode | DataNode |
| ZK | ZK | ZK |
| ResourceManager | ResourceManager |  |
| NodeManager | NodeManager | NodeManager |

3. 具体配置

（1）yarn-site.xml

|  |
| --- |
| <configuration>  <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  <!--启用resourcemanager ha-->  <property>  <name>yarn.resourcemanager.ha.enabled</name>  <value>true</value>  </property>    <!--声明两台resourcemanager的地址-->  <property>  <name>yarn.resourcemanager.cluster-id</name>  <value>cluster-yarn1</value>  </property>  <property>  <name>yarn.resourcemanager.ha.rm-ids</name>  <value>rm1,rm2</value>  </property>  <property>  <name>yarn.resourcemanager.hostname.rm1</name>  <value>hadoop102</value>  </property>  <property>  <name>yarn.resourcemanager.hostname.rm2</name>  <value>hadoop103</value>  </property>    <!--指定zookeeper集群的地址-->  <property>  <name>yarn.resourcemanager.zk-address</name>  <value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value>  </property>  <!--启用自动恢复-->  <property>  <name>yarn.resourcemanager.recovery.enabled</name>  <value>true</value>  </property>    <!--指定resourcemanager的状态信息存储在zookeeper集群-->  <property>  <name>yarn.resourcemanager.store.class</name>  <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>  </property>  </configuration> |

（2）同步更新其他节点的配置信息

4. 启动hdfs

（1）在各个JournalNode节点上，输入以下命令启动journalnode服务：

sbin/hadoop-daemon.sh start journalnode

（2）在[nn1]上，对其进行格式化，并启动：

bin/hdfs namenode -format

sbin/hadoop-daemon.sh start namenode

（3）在[nn2]上，同步nn1的元数据信息：

bin/hdfs namenode -bootstrapStandby

（4）启动[nn2]：

sbin/hadoop-daemon.sh start namenode

（5）启动所有DataNode

sbin/hadoop-daemons.sh start datanode

（6）将[nn1]切换为Active

bin/hdfs haadmin -transitionToActive nn1

5. 启动YARN

（1）在hadoop102中执行：

sbin/start-yarn.sh

（2）在hadoop103中执行：

sbin/yarn-daemon.sh start resourcemanager

（3）查看服务状态，如图3-24所示

bin/yarn rmadmin -getServiceState rm1

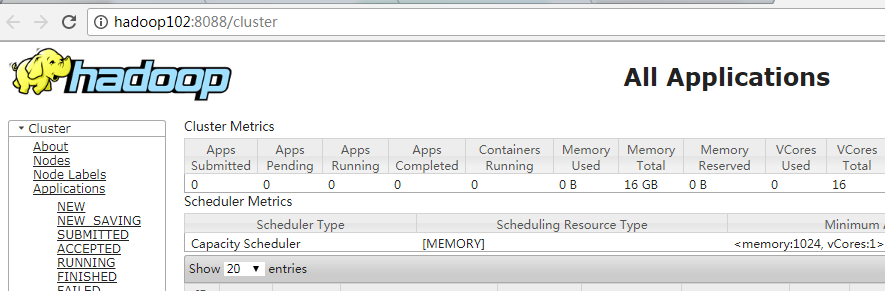


图3-24 YARN的服务状态

## 8.5 HDFS Federation架构设计

1. NameNode架构的局限性

（1）Namespace（命名空间）的限制

由于NameNode在内存中存储所有的元数据（metadata），因此单个NameNode所能存储的对象（文件+块）数目受到NameNode所在JVM的heap size的限制。50G的heap能够存储20亿（200million）个对象，这20亿个对象支持4000个DataNode，12PB的存储（假设文件平均大小为40MB）。随着数据的飞速增长，存储的需求也随之增长。单个DataNode从4T增长到36T，集群的尺寸增长到8000个DataNode。存储的需求从12PB增长到大于100PB。

（2）隔离问题

由于HDFS仅有一个NameNode，无法隔离各个程序，因此HDFS上的一个实验程序就很有可能影响整个HDFS上运行的程序。

（3）性能的瓶颈

由于是单个NameNode的HDFS架构，因此整个HDFS文件系统的吞吐量受限于单个NameNode的吞吐量。

2. HDFS Federation架构设计，如图3-25所示

能不能有多个NameNode

表3-3

|  |  |  |
| --- | --- | --- |
| NameNode | NameNode | NameNode |
| 元数据 | 元数据 | 元数据 |
| Log | machine | 电商数据/话单数据 |



图3-25 HDFS Federation架构设计

3. HDFS Federation应用思考

不同应用可以使用不同NameNode进行数据管理

图片业务、爬虫业务、日志审计业务

Hadoop生态系统中，不同的框架使用不同的NameNode进行管理NameSpace。（隔离性）

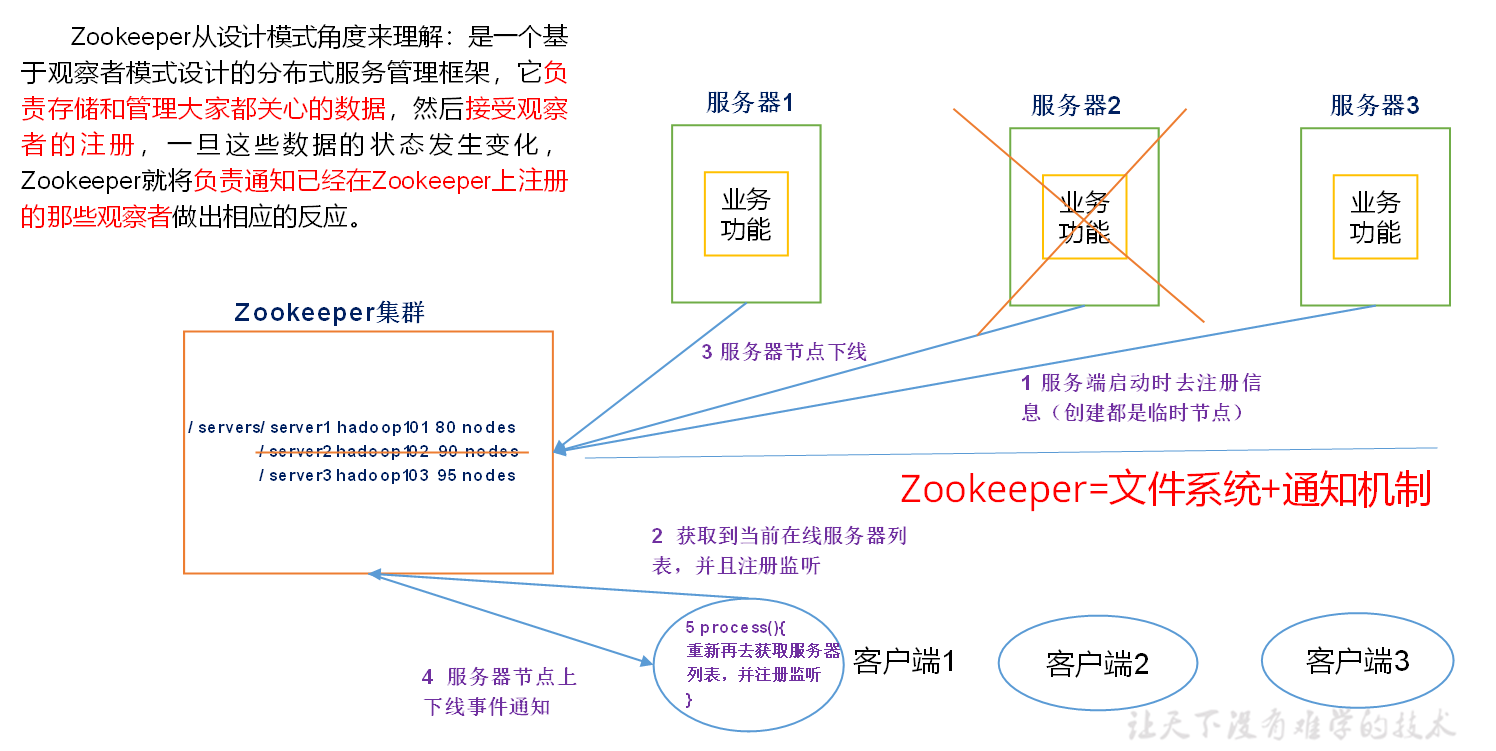
# Zookeeper

# 第1章 Zookeeper入门

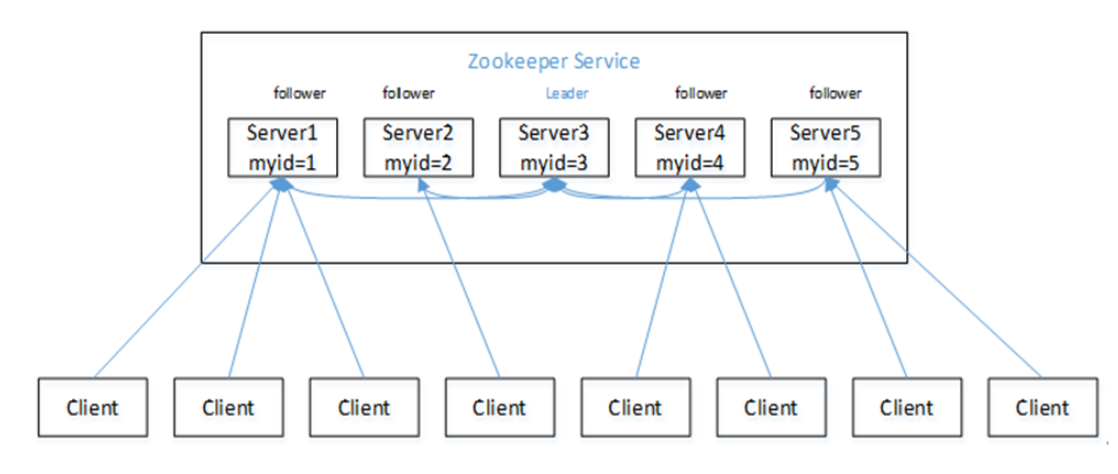
## 1.1 概述

Zookeeper是一个开源的分布式的，为分布式应用提供协调服务的Apache项目。

**Zookeeper工作机制如图：**



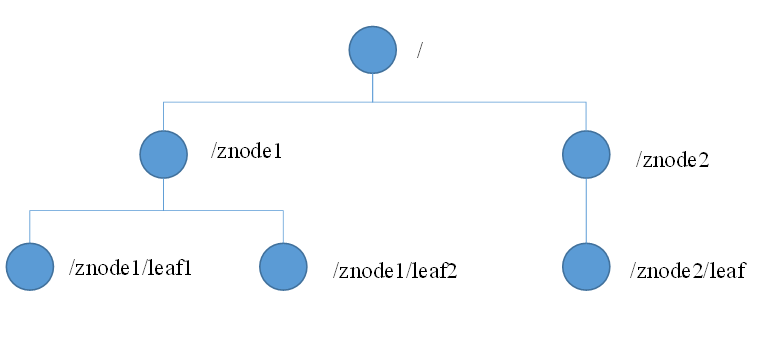
## 1.2 特点



1. Zookeeper：一个领导者（leader），多个跟随者（follower）组成的集群。
2. 集群中只要有半数以上节点存活，Zookeeper集群就能正常服务。
3. 全局数据一致：每个Server保存一份相同的数据副本，Client无论连接到哪个Server，数据都是一致的。
4. 更新请求顺序进行，来自同一个Client的更新请求按其发送顺序依次执行。
5. 数据更新原子性，一次数据更新要么成功要么失败。
6. 实时性，在一定时间范围内，Client能读到最新数据

## 1.3 数据结构

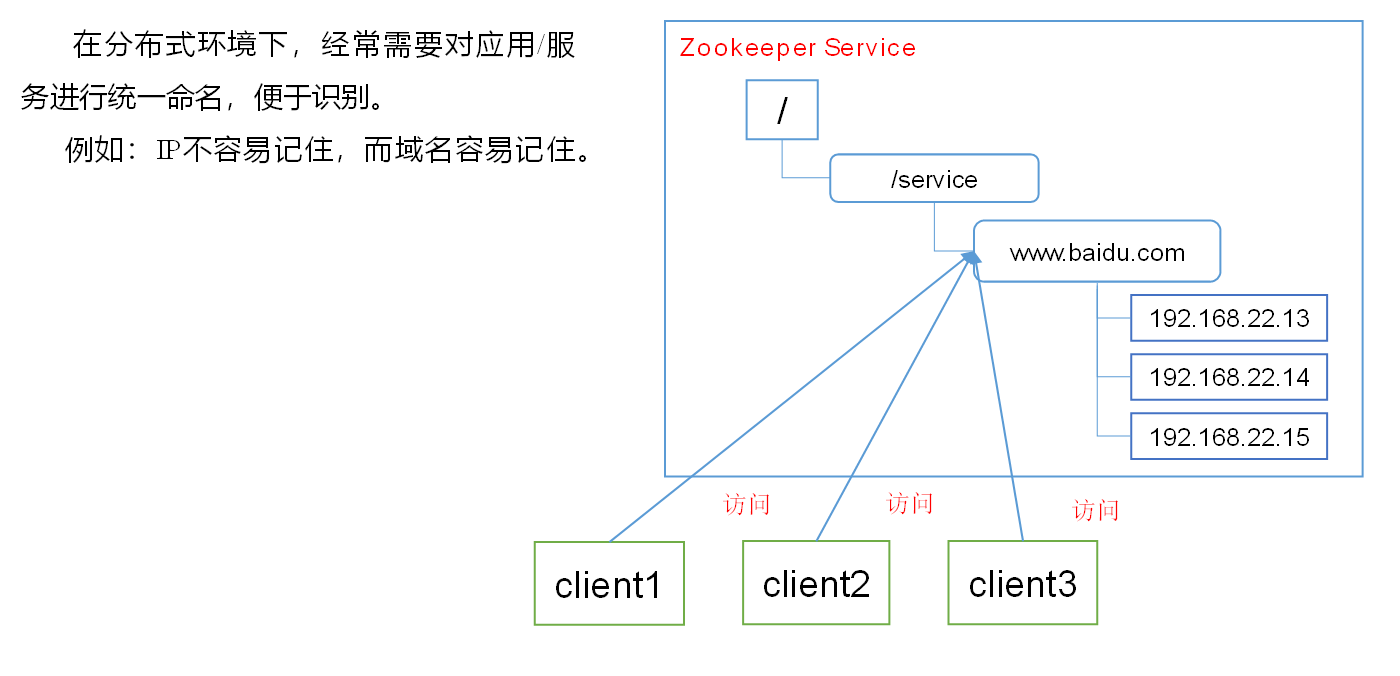
Zookeeper数据模型的结构与Unix文件系统很类似，整体上可以看作是一棵树，每个节点称作一个ZNode。每一个ZNode默认能够存储1MB的数据，每个ZNode都可以通过其路径唯一标识。



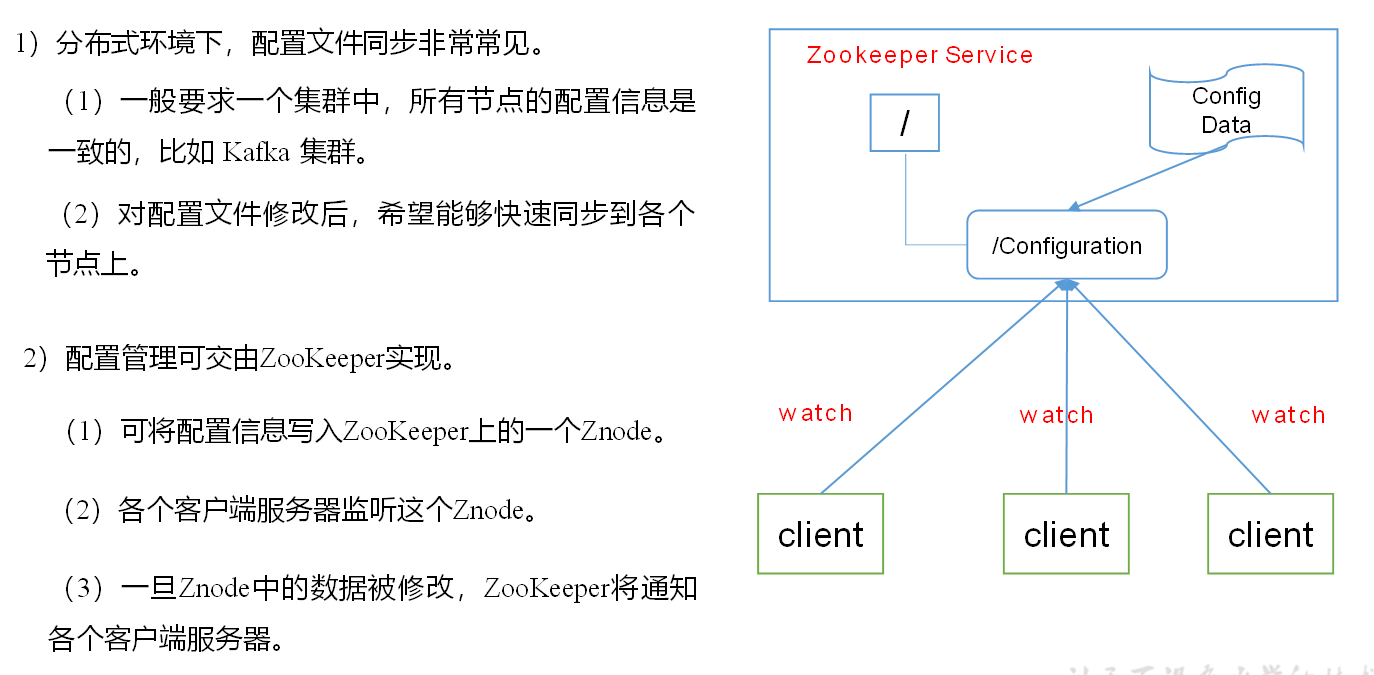
## 1.4 应用场景

提供的服务包括：统一命名服务、统一配置管理、统一集群管理、服务器节点动态上下线、软负载均衡等。

**统一命名服务：**



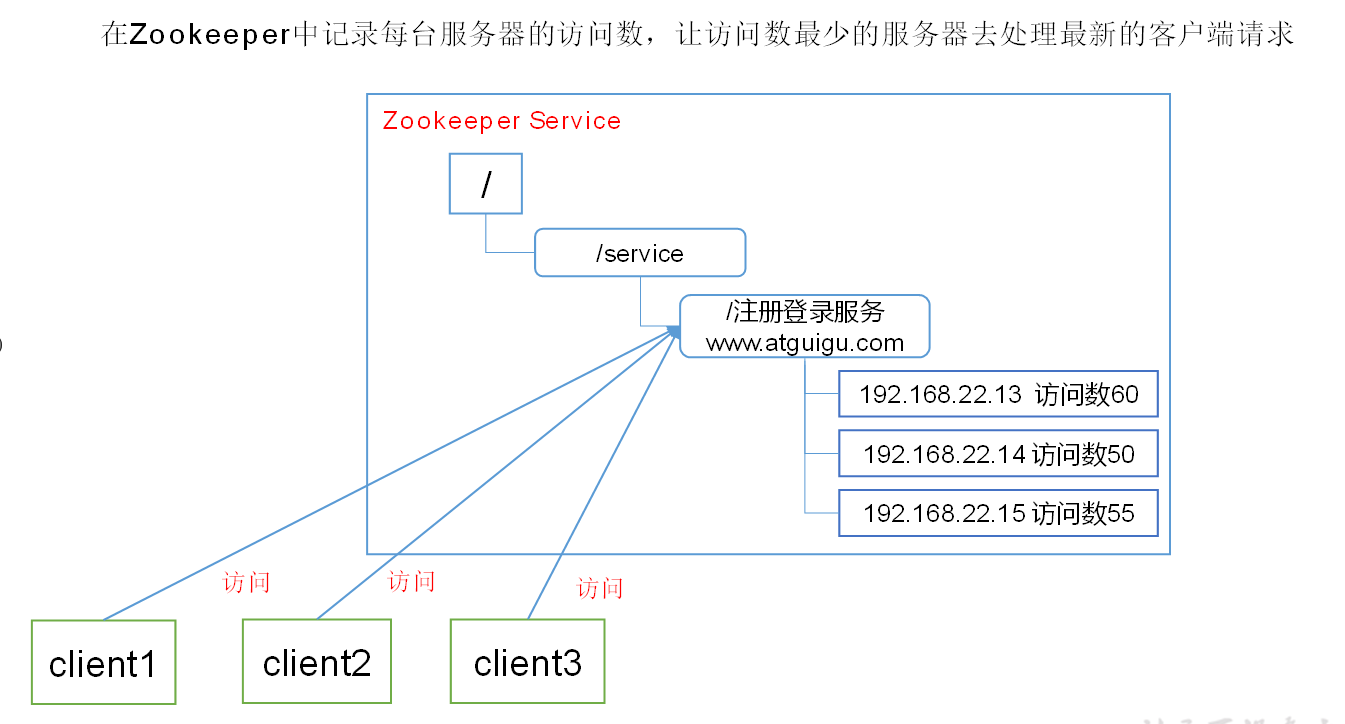
**统一配置管理：**



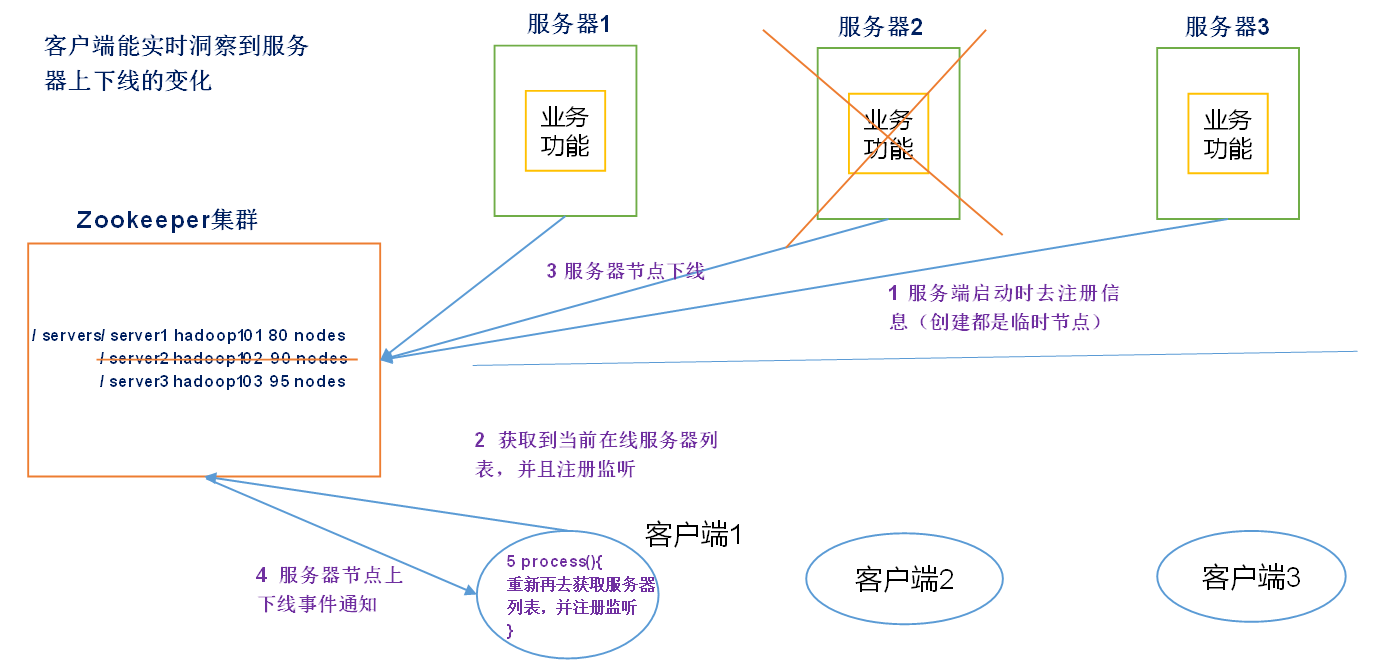
**统一集群管理：**



**软负载均衡：**



**服务器动态上下线：**



## 1.5 下载地址

1．官网首页：

https://zookeeper.apache.org/

2．下载截图，如图5-5，5-6，5-7所示

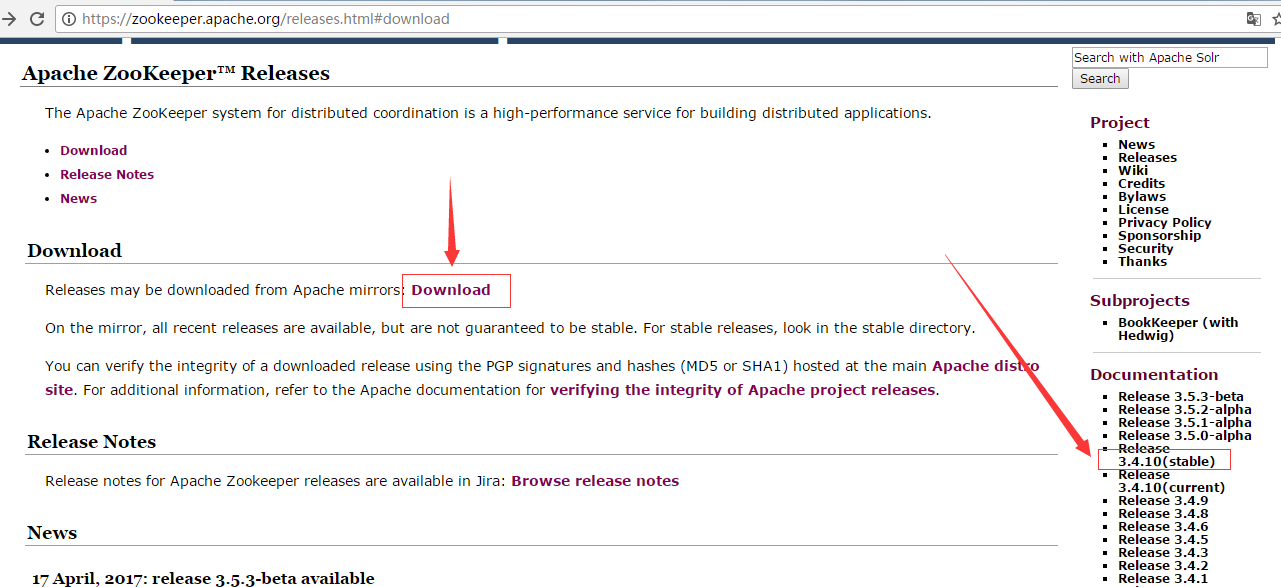


图5-5 Zookeeper下载（一）



图5-6 Zookeeper下载（二）

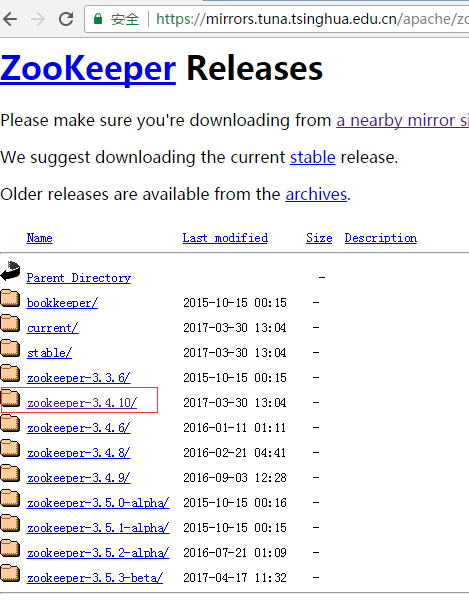


图5-7 Zookeeper下载（三）

# 第2章 Zookeeper安装

## 2.1 本地模式安装部署

1．安装前准备

（1）安装Jdk

（2）拷贝Zookeeper安装包到Linux系统下

（3）解压到指定目录

[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/

2．配置修改

（1）将/opt/module/zookeeper-3.4.10/conf这个路径下的zoo\_sample.cfg修改为zoo.cfg；

[atguigu@hadoop102 conf]$ mv zoo\_sample.cfg zoo.cfg

（2）打开zoo.cfg文件，修改dataDir路径：

[atguigu@hadoop102 zookeeper-3.4.10]$ vim zoo.cfg

修改如下内容：

dataDir=/opt/module/zookeeper-3.4.10/zkData

（3）在/opt/module/zookeeper-3.4.10/这个目录上创建zkData文件夹

[atguigu@hadoop102 zookeeper-3.4.10]$ mkdir zkData

3．操作Zookeeper

（1）启动Zookeeper

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkServer.sh start

（2）查看进程是否启动

[atguigu@hadoop102 zookeeper-3.4.10]$ jps

4020 Jps

4001 QuorumPeerMain

（3）查看状态：

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: standalone

（4）启动客户端：

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkCli.sh

（5）退出客户端：

[zk: localhost:2181(CONNECTED) 0] quit

（6）停止Zookeeper

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkServer.sh stop

## 2.2 配置参数解读

Zookeeper中的配置文件zoo.cfg中参数含义解读如下：

1．tickTime = 2000：通信心跳数，Zookeeper服务器与客户端心跳时间，单位毫秒

Zookeeper使用的基本时间，服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个tickTime时间就会发送一个心跳，时间单位为毫秒。

它用于心跳机制，并且设置最小的session超时时间为两倍心跳时间。(session的最小超时时间是2\*tickTime)

2．initLimit =10：LF初始通信时限

集群中的Follower跟随者服务器与Leader领导者服务器之间初始连接时能容忍的最多心跳数（tickTime的数量），用它来限定集群中的Zookeeper服务器连接到Leader的时限。

3．syncLimit =5：LF同步通信时限

集群中Leader与Follower之间的最大响应时间单位，假如响应超过syncLimit \* tickTime，Leader认为Follwer死掉，从服务器列表中删除Follwer。

4．dataDir：数据文件目录+数据持久化路径

主要用于保存Zookeeper中的数据。

5．clientPort =2181：客户端连接端口

监听客户端连接的端口。

# 第3章 Zookeeper实战（开发重点）

## 3.1 分布式安装部署

1．集群规划

在hadoop102、hadoop103和hadoop104三个节点上部署Zookeeper。

2．解压安装

（1）解压Zookeeper安装包到/opt/module/目录下

[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/

（2）同步/opt/module/zookeeper-3.4.10目录内容到hadoop103、hadoop104

[atguigu@hadoop102 module]$ xsync zookeeper-3.4.10/

3．配置服务器编号

（1）在/opt/module/zookeeper-3.4.10/这个目录下创建zkData

[atguigu@hadoop102 zookeeper-3.4.10]$ mkdir -p zkData

（2）在/opt/module/zookeeper-3.4.10/zkData目录下创建一个myid的文件

[atguigu@hadoop102 zkData]$ touch myid

添加myid文件，注意一定要在linux里面创建，在notepad++里面很可能乱码

（3）编辑myid文件

[atguigu@hadoop102 zkData]$ vi myid

在文件中添加与server对应的编号：

2

（4）拷贝配置好的zookeeper到其他机器上

[atguigu@hadoop102 zkData]$ xsync myid

并分别在hadoop103、hadoop104上修改myid文件中内容为3、4

4．配置zoo.cfg文件

（1）重命名/opt/module/zookeeper-3.4.10/conf这个目录下的zoo\_sample.cfg为zoo.cfg

[atguigu@hadoop102 conf]$ mv zoo\_sample.cfg zoo.cfg

（2）打开zoo.cfg文件

[atguigu@hadoop102 conf]$ vim zoo.cfg

修改数据存储路径配置

dataDir=/opt/module/zookeeper-3.4.10/zkData

增加如下配置

#######################cluster##########################

server.2=hadoop102:2888:3888

server.3=hadoop103:2888:3888

server.4=hadoop104:2888:3888

（3）同步zoo.cfg配置文件

[atguigu@hadoop102 conf]$ xsync zoo.cfg

（4）配置参数解读

server.A=B:C:D。

**A**是一个数字，表示这个是第几号服务器；

集群模式下配置一个文件myid，这个文件在dataDir目录下，这个文件里面有一个数据就是A的值，Zookeeper启动时读取此文件，拿到里面的数据与zoo.cfg里面的配置信息比较从而判断到底是哪个server。

**B**是这个服务器的地址；

**C**是这个服务器Follower与集群中的Leader服务器交换信息的端口；

**D**是万一集群中的Leader服务器挂了，需要一个端口来重新进行选举，选出一个新的Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

5．集群操作

（1）分别启动Zookeeper

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkServer.sh start

[atguigu@hadoop103 zookeeper-3.4.10]$ bin/zkServer.sh start

[atguigu@hadoop104 zookeeper-3.4.10]$ bin/zkServer.sh start

（2）查看状态

[atguigu@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

[atguigu@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: leader

[atguigu@hadoop104 zookeeper-3.4.5]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

## 3.2 客户端命令行操作

表5-1

|  |  |
| --- | --- |
| 命令基本语法 | 功能描述 |
| help | 显示所有操作命令 |
| ls path [watch] | 使用 ls 命令来查看当前znode中所包含的内容 |
| ls2 path [watch] | 查看当前节点数据并能看到更新次数等数据 |
| create | 普通创建  -s 含有序列  -e 临时（重启或者超时消失） |
| get path [watch] | 获得节点的值 |
| set | 设置节点的具体值 |
| stat | 查看节点状态 |
| delete | 删除节点 |
| rmr | 递归删除节点 |

1．启动客户端

[atguigu@hadoop103 zookeeper-3.4.10]$ bin/zkCli.sh

2．显示所有操作命令

[zk: localhost:2181(CONNECTED) 1] help

3．查看当前znode中所包含的内容

[zk: localhost:2181(CONNECTED) 0] ls /

[zookeeper]

4．查看当前节点详细数据

[zk: localhost:2181(CONNECTED) 1] ls2 /

[zookeeper]

cZxid = 0x0

ctime = Thu Jan 01 08:00:00 CST 1970

mZxid = 0x0

mtime = Thu Jan 01 08:00:00 CST 1970

pZxid = 0x0

cversion = -1

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 0

numChildren = 1

5．分别创建2个普通节点

[zk: localhost:2181(CONNECTED) 3] create /sanguo "jinlian"

Created /sanguo

[zk: localhost:2181(CONNECTED) 4] create /sanguo/shuguo "liubei"

Created /sanguo/shuguo

6．获得节点的值

[zk: localhost:2181(CONNECTED) 5] get /sanguo

jinlian

cZxid = 0x100000003

ctime = Wed Aug 29 00:03:23 CST 2018

mZxid = 0x100000003

mtime = Wed Aug 29 00:03:23 CST 2018

pZxid = 0x100000004

cversion = 1

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 7

numChildren = 1

[zk: localhost:2181(CONNECTED) 6]

[zk: localhost:2181(CONNECTED) 6] get /sanguo/shuguo

liubei

cZxid = 0x100000004

ctime = Wed Aug 29 00:04:35 CST 2018

mZxid = 0x100000004

mtime = Wed Aug 29 00:04:35 CST 2018

pZxid = 0x100000004

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 6

numChildren = 0

7．创建短暂节点

[zk: localhost:2181(CONNECTED) 7] create -e /sanguo/wuguo "zhouyu"

Created /sanguo/wuguo

（1）在当前客户端是能查看到的

[zk: localhost:2181(CONNECTED) 3] ls /sanguo

[wuguo, shuguo]

（2）退出当前客户端然后再重启客户端

[zk: localhost:2181(CONNECTED) 12] quit

[atguigu@hadoop104 zookeeper-3.4.10]$ bin/zkCli.sh

（3）再次查看根目录下短暂节点已经删除

[zk: localhost:2181(CONNECTED) 0] ls /sanguo

[shuguo]

8．创建带序号的节点

（1）先创建一个普通的根节点/sanguo/weiguo

[zk: localhost:2181(CONNECTED) 1] create /sanguo/weiguo "caocao"

Created /sanguo/weiguo

（2）创建带序号的节点

[zk: localhost:2181(CONNECTED) 2] create -s /sanguo/weiguo/xiaoqiao "jinlian"

Created /sanguo/weiguo/xiaoqiao0000000000

[zk: localhost:2181(CONNECTED) 3] create -s /sanguo/weiguo/daqiao "jinlian"

Created /sanguo/weiguo/daqiao0000000001

[zk: localhost:2181(CONNECTED) 4] create -s /sanguo/weiguo/diaocan "jinlian"

Created /sanguo/weiguo/diaocan0000000002

如果原来没有序号节点，序号从0开始依次递增。如果原节点下已有2个节点，则再排序时从2开始，以此类推。

9．修改节点数据值

[zk: localhost:2181(CONNECTED) 6] set /sanguo/weiguo "simayi"

10．节点的值变化监听

（1）在hadoop104主机上注册监听/sanguo节点数据变化

[zk: localhost:2181(CONNECTED) 26] [zk: localhost:2181(CONNECTED) 8] get /sanguo watch

（2）在hadoop103主机上修改/sanguo节点的数据

[zk: localhost:2181(CONNECTED) 1] set /sanguo "xisi"

（3）观察hadoop104主机收到数据变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/sanguo

11．节点的子节点变化监听（路径变化）

（1）在hadoop104主机上注册监听/sanguo节点的子节点变化

[zk: localhost:2181(CONNECTED) 1] ls /sanguo watch

[aa0000000001, server101]

（2）在hadoop103主机/sanguo节点上创建子节点

[zk: localhost:2181(CONNECTED) 2] create /sanguo/jin "simayi"

Created /sanguo/jin

（3）观察hadoop104主机收到子节点变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/sanguo

12．删除节点

[zk: localhost:2181(CONNECTED) 4] delete /sanguo/jin

13．递归删除节点

[zk: localhost:2181(CONNECTED) 15] rmr /sanguo/shuguo

14．查看节点状态

[zk: localhost:2181(CONNECTED) 17] stat /sanguo

cZxid = 0x100000003

ctime = Wed Aug 29 00:03:23 CST 2018

mZxid = 0x100000011

mtime = Wed Aug 29 00:21:23 CST 2018

pZxid = 0x100000014

cversion = 9

dataVersion = 1

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 4

numChildren = 1

## 3.3 API应用

### 3.3.1 Eclipse环境搭建

1．创建一个Maven工程

2．添加pom文件

|  |
| --- |
| <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>RELEASE</version>  </dependency>  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j-core</artifactId>  <version>2.8.2</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.zookeeper/zookeeper -->  <dependency>  <groupId>org.apache.zookeeper</groupId>  <artifactId>zookeeper</artifactId>  <version>3.4.10</version>  </dependency>  </dependencies> |

3．拷贝log4j.properties文件到项目根目录

需要在项目的src/main/resources目录下，新建一个文件，命名为“log4j.properties”，在文件中填入。

log4j.rootLogger=INFO, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.appender.logfile=org.apache.log4j.FileAppender

log4j.appender.logfile.File=target/spring.log

log4j.appender.logfile.layout=org.apache.log4j.PatternLayout

log4j.appender.logfile.layout.ConversionPattern=%d %p [%c] - %m%n

### 3.3.2 创建ZooKeeper客户端

|  |
| --- |
| private static String connectString =  "hadoop102:2181,hadoop103:2181,hadoop104:2181";  private static int sessionTimeout = 2000;  private ZooKeeper zkClient = null;  @Before  public void init() throws Exception {  zkClient = new ZooKeeper(connectString, sessionTimeout, new Watcher() {  @Override  public void process(WatchedEvent event) {  // 收到事件通知后的回调函数（用户的业务逻辑）  System.out.println(event.getType() + "--" + event.getPath());  // 再次启动监听  try {  zkClient.getChildren("/", true);  } catch (Exception e) {  e.printStackTrace();  }  }  });  } |

### 3.3.3 创建子节点

|  |
| --- |
| // 创建子节点  @Test  public void create() throws Exception {  // 参数1：要创建的节点的路径； 参数2：节点数据 ； 参数3：节点权限 ；参数4：节点的类型  String nodeCreated = zkClient.create("/atguigu", "jinlian".getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);  } |

### 3.3.4 获取子节点并监听节点变化

|  |
| --- |
| // 获取子节点  @Test  public void getChildren() throws Exception {  List<String> children = zkClient.getChildren("/", true);  for (String child : children) {  System.out.println(child);  }  // 延时阻塞  Thread.sleep(Long.MAX\_VALUE);  } |

### 3.3.5 判断Znode是否存在

|  |
| --- |
| // 判断znode是否存在  @Test  public void exist() throws Exception {  Stat stat = zkClient.exists("/eclipse", false);  System.out.println(stat == null ? "not exist" : "exist");  } |

## 3.4 监听服务器节点动态上下线案例（扩展）

1．需求

某分布式系统中，主节点可以有多台，可以动态上下线，任意一台客户端都能实时感知到主节点服务器的上下线。

2．需求分析，如图5-12所示

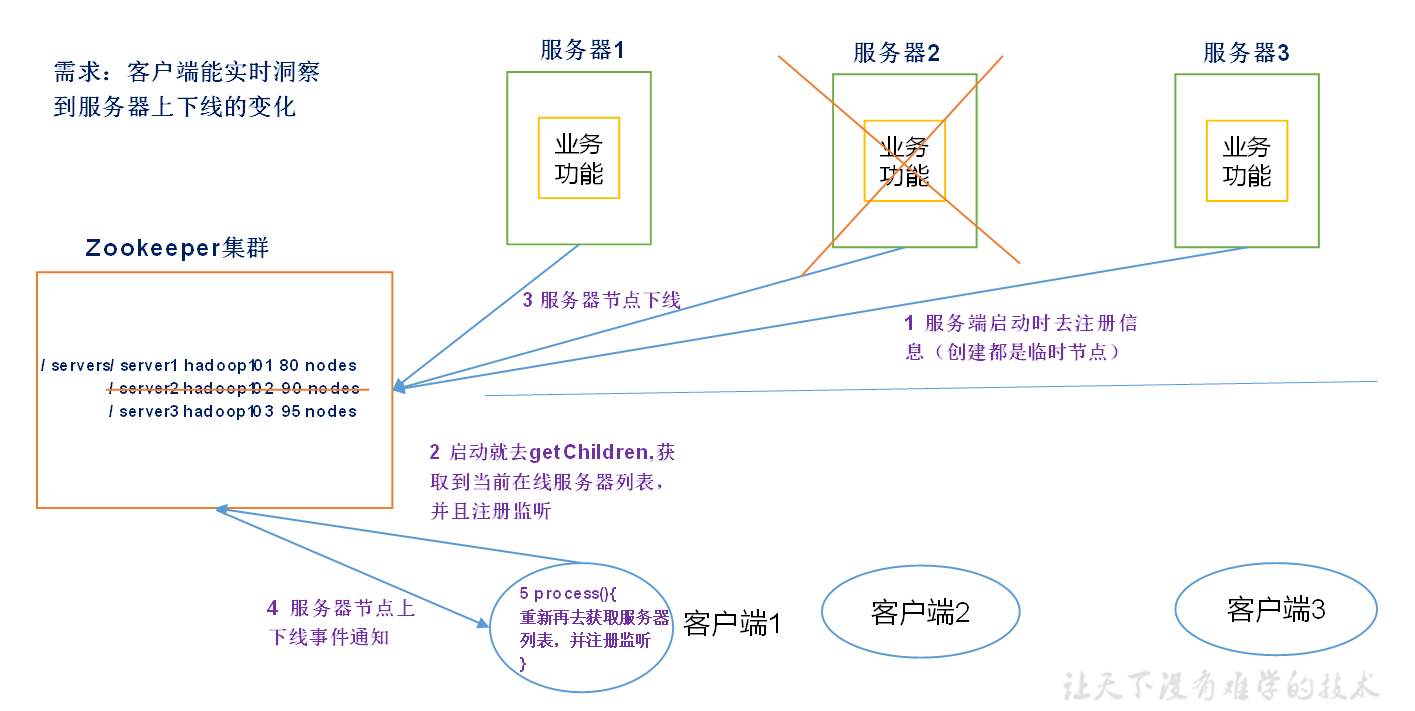


图5-12 服务器动态上下线

3．具体实现

（0）先在集群上创建/servers节点

[zk: localhost:2181(CONNECTED) 10] create /servers "servers"

Created /servers

（1）服务器端向Zookeeper注册代码

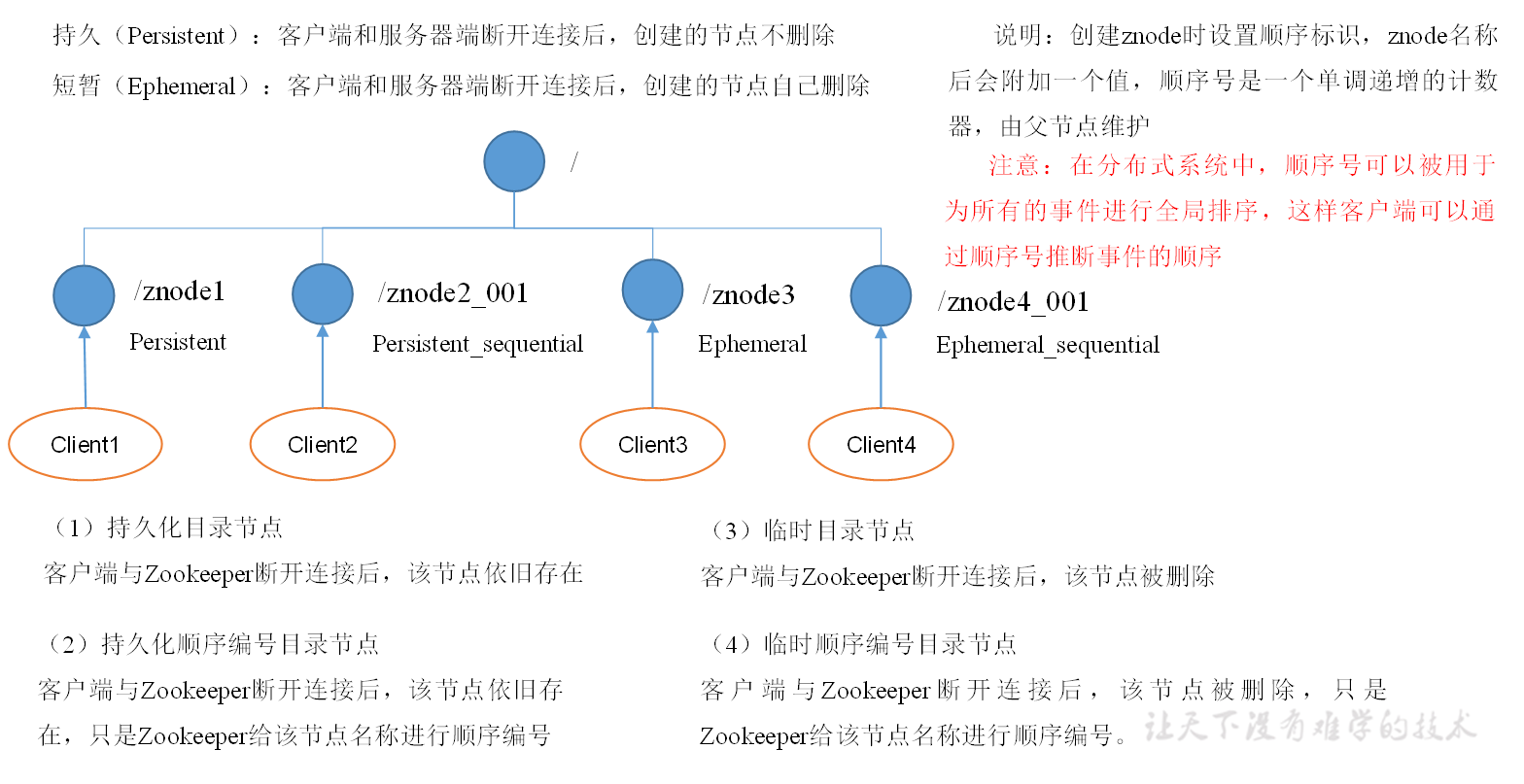
|  |
| --- |
| package com.atguigu.zkcase;  import java.io.IOException;  import org.apache.zookeeper.CreateMode;  import org.apache.zookeeper.WatchedEvent;  import org.apache.zookeeper.Watcher;  import org.apache.zookeeper.ZooKeeper;  import org.apache.zookeeper.ZooDefs.Ids;  public class DistributeServer {  private static String connectString = "hadoop102:2181,hadoop103:2181,hadoop104:2181";  private static int sessionTimeout = 2000;  private ZooKeeper zk = null;  private String parentNode = "/servers";    // 创建到zk的客户端连接  public void getConnect() throws IOException{  zk = new ZooKeeper(connectString, sessionTimeout, new Watcher() {  @Override  public void process(WatchedEvent event) {  }  });  }    // 注册服务器  public void registServer(String hostname) throws Exception{  String create = zk.create(parentNode + "/server", hostname.getBytes(), Ids.OPEN\_ACL\_UNSAFE, CreateMode.EPHEMERAL\_SEQUENTIAL);  System.out.println(hostname +" is online "+ create);  }    // 业务功能  public void business(String hostname) throws Exception{  System.out.println(hostname+" is working ...");  Thread.sleep(Long.MAX\_VALUE);  }    public static void main(String[] args) throws Exception {  // 1获取zk连接  DistributeServer server = new DistributeServer();  server.getConnect();  // 2 利用zk连接注册服务器信息  server.registServer(args[0]);  // 3 启动业务功能  server.business(args[0]);  }  } |

（2）客户端代码

|  |
| --- |
| package com.atguigu.zkcase;  import java.io.IOException;  import java.util.ArrayList;  import java.util.List;  import org.apache.zookeeper.WatchedEvent;  import org.apache.zookeeper.Watcher;  import org.apache.zookeeper.ZooKeeper;  public class DistributeClient {  private static String connectString = "hadoop102:2181,hadoop103:2181,hadoop104:2181";  private static int sessionTimeout = 2000;  private ZooKeeper zk = null;  private String parentNode = "/servers";  // 创建到zk的客户端连接  public void getConnect() throws IOException {  zk = new ZooKeeper(connectString, sessionTimeout, new Watcher() {  @Override  public void process(WatchedEvent event) {  // 再次启动监听  try {  getServerList();  } catch (Exception e) {  e.printStackTrace();  }  }  });  }  // 获取服务器列表信息  public void getServerList() throws Exception {  // 1获取服务器子节点信息，并且对父节点进行监听  List<String> children = zk.getChildren(parentNode, true);  // 2存储服务器信息列表  ArrayList<String> servers = new ArrayList<>();    // 3遍历所有节点，获取节点中的主机名称信息  for (String child : children) {  byte[] data = zk.getData(parentNode + "/" + child, false, null);  servers.add(new String(data));  }  // 4打印服务器列表信息  System.out.println(servers);  }  // 业务功能  public void business() throws Exception{  System.out.println("client is working ...");  Thread.sleep(Long.MAX\_VALUE);  }  public static void main(String[] args) throws Exception {  // 1获取zk连接  DistributeClient client = new DistributeClient();  client.getConnect();  // 2获取servers的子节点信息，从中获取服务器信息列表  client.getServerList();  // 3业务进程启动  client.business();  }  } |

# 第4章 Zookeeper内部原理

## 4.1 节点类型



## 4.2 Stat结构体

1）czxid-创建节点的事务zxid

每次修改ZooKeeper状态都会收到一个zxid形式的时间戳，也就是ZooKeeper事务ID。

事务ID是ZooKeeper中所有修改总的次序。每个修改都有唯一的zxid，如果zxid1小于zxid2，那么zxid1在zxid2之前发生。

2）ctime - znode被创建的毫秒数(从1970年开始)

3）mzxid - znode最后更新的事务zxid

4）mtime - znode最后修改的毫秒数(从1970年开始)

5）pZxid-znode最后更新的子节点zxid

6）cversion - znode子节点变化号，znode子节点修改次数

7）dataversion - znode数据变化号

8）aclVersion - znode访问控制列表的变化号

9）ephemeralOwner- 如果是临时节点，这个是znode拥有者的session id。如果不是临时节点则是0。

10）dataLength- znode的数据长度

11）numChildren - znode子节点数量

## 4.3 监听器原理（面试重点）

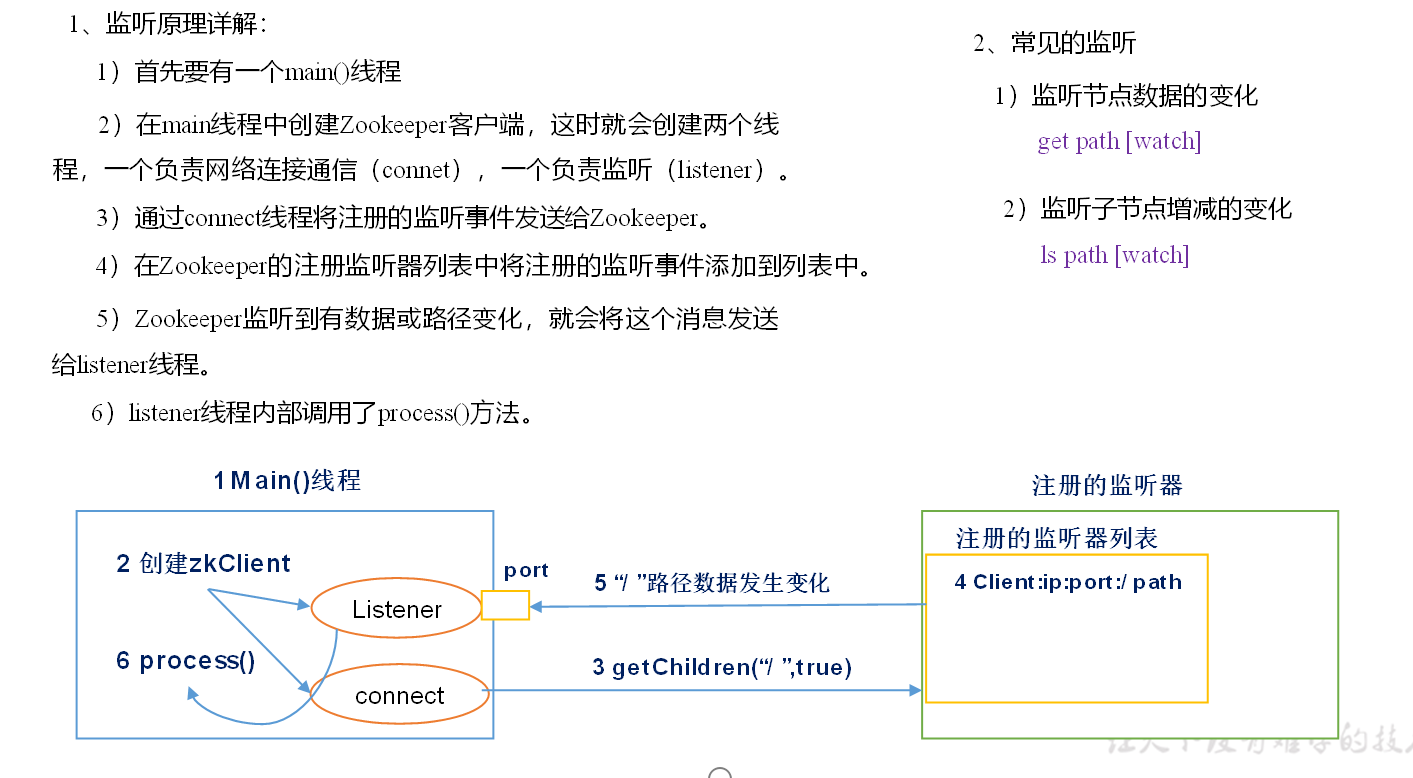


图5-10 监听器原理

## 4.4 Paxos算法（扩展）

Paxos算法一种基于消息传递且具有高度容错特性的一致性算法。

分布式系统中的节点通信存在两种模型：共享内存（Shared memory）和消息传递（Messages passing）。基于消息传递通信模型的分布式系统，不可避免的会发生以下错误：进程可能会慢、被杀死或者重启，消息可能会延迟、丢失、重复，在基础 Paxos 场景中，先不考虑可能出现消息篡改即拜占庭错误的情况。Paxos 算法解决的问题是在一个可能发生上述异常的分布式系统中如何就某个值达成一致，保证不论发生以上任何异常，都不会破坏决议的一致性。



Paxos算法流程中的每条消息描述如下：

1. Prepare: Proposer生成全局唯一且递增的Proposal ID (可使用时间戳加Server ID)，向所有Acceptors发送Prepare请求，这里无需携带提案内容，只携带Proposal ID即可。
2. Promise: Acceptors收到Prepare请求后，做出“两个承诺，一个应答”。

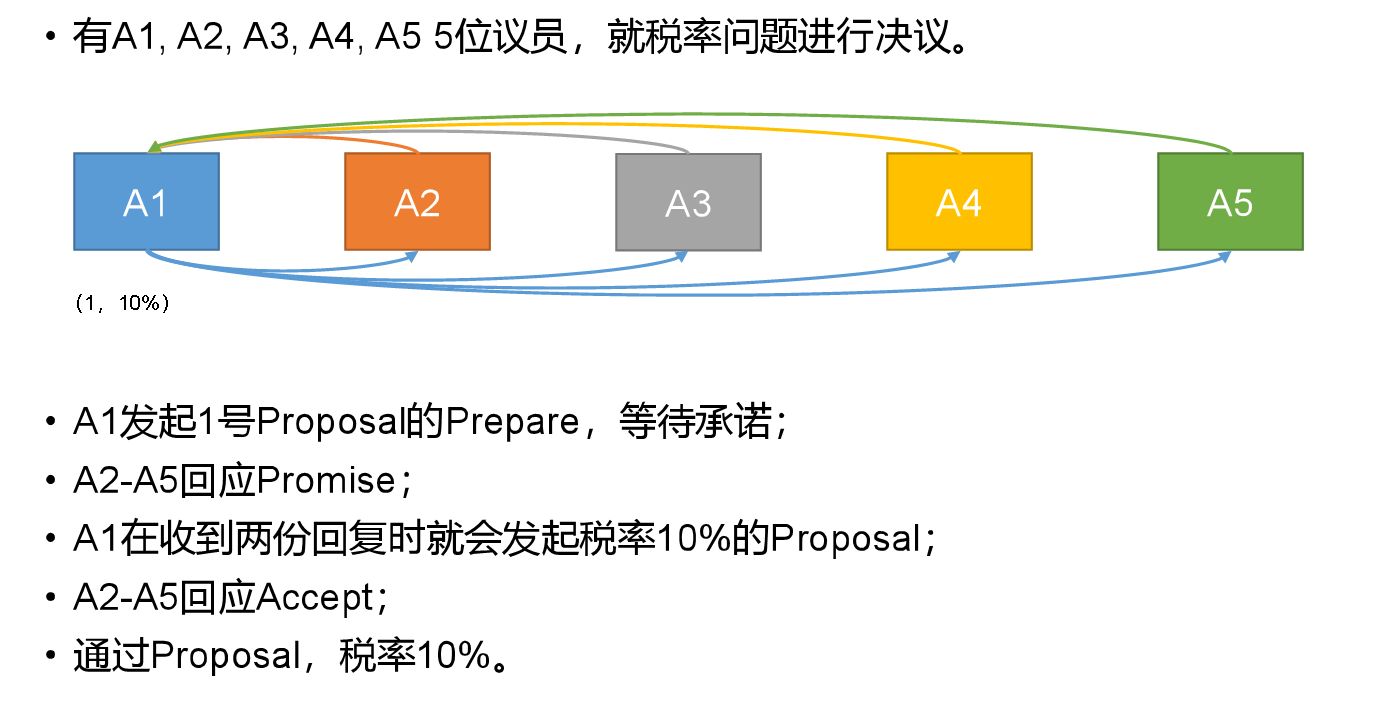
两个承诺：

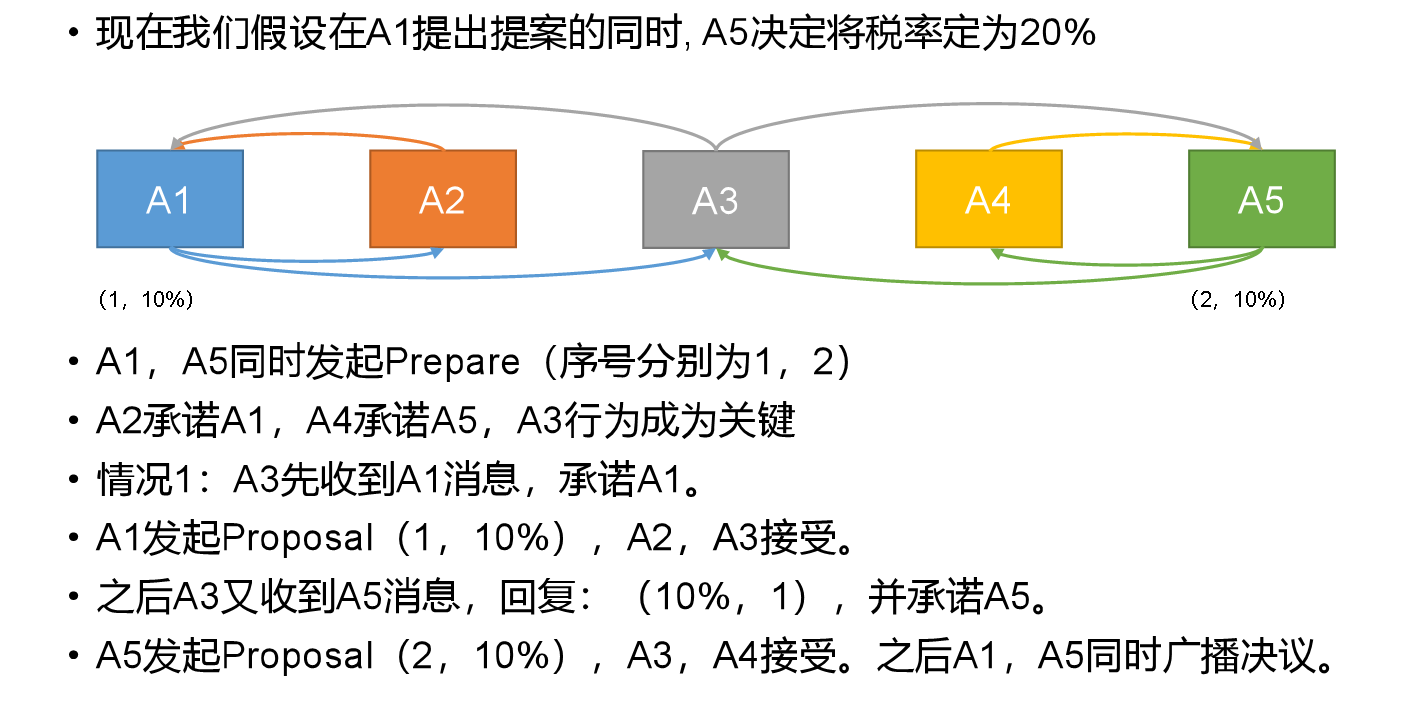
1. 不再接受Proposal ID小于等于（注意：这里是<= ）当前请求的Prepare请求。
2. 不再接受Proposal ID小于（注意：这里是< ）当前请求的Propose请求。

一个应答：

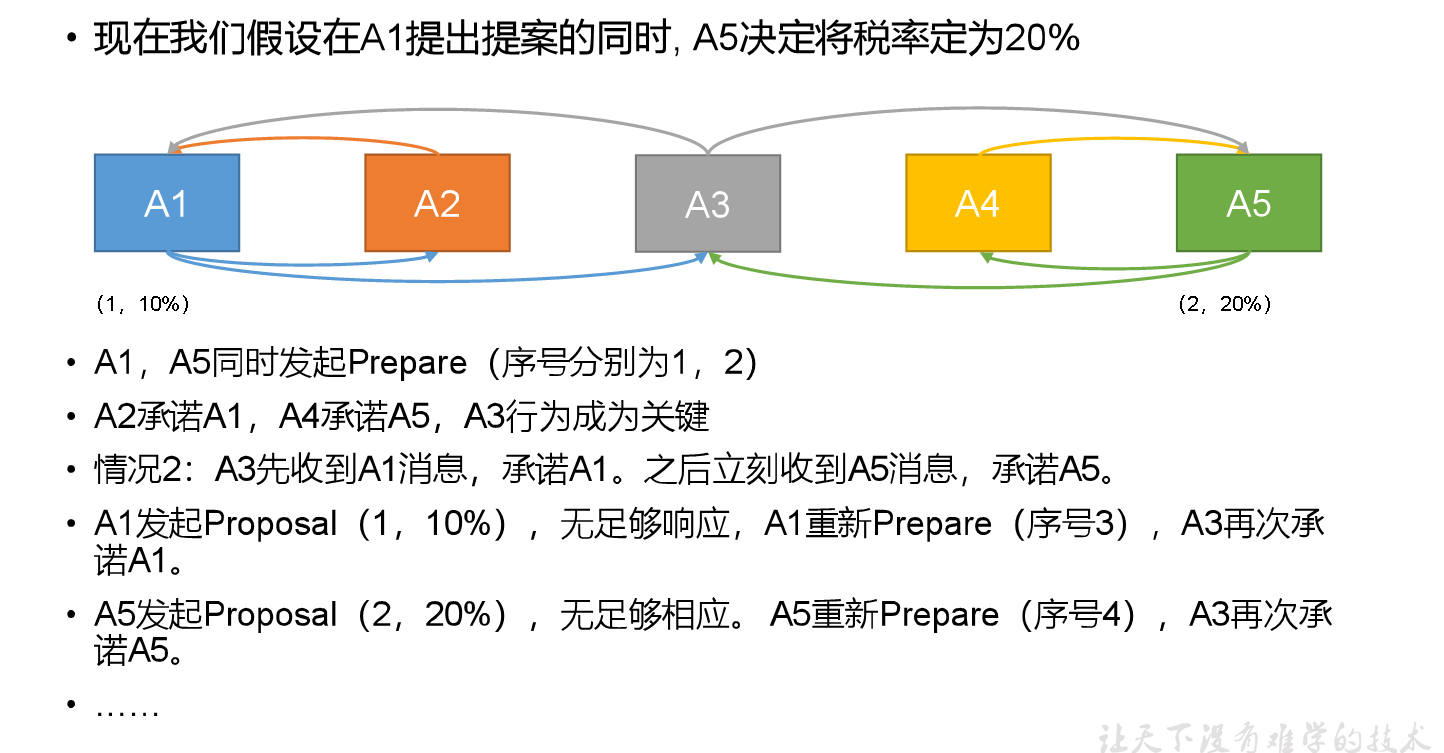
1. 不违背以前做出的承诺下，回复已经Accept过的提案中Proposal ID最大的那个提案的Value和Proposal ID，没有则返回空值。
2. Propose: Proposer 收到多数Acceptors的Promise应答后，从应答中选择Proposal ID最大的提案的Value，作为本次要发起的提案。如果所有应答的提案Value均为空值，则可以自己随意决定提案Value。然后携带当前Proposal ID，向所有Acceptors发送Propose请求。
3. Accept: Acceptor收到Propose请求后，在不违背自己之前做出的承诺下，接受并持久化当前Proposal ID和提案Value。
4. Learn: Proposer收到多数Acceptors的Accept后，决议形成，将形成的决议发送给所有Learners。

下面我们针对上述描述做三种情况的推演举例：为了简化流程，我们这里不设置Learner。





Paxos算法缺陷：在网络复杂的情况下，一个应用Paxos算法的分布式系统，可能很久无法收敛，甚至陷入活锁的情况。



造成这种情况的原因是系统中有一个以上的Proposer，多个Proposers相互争夺Acceptors，造成迟迟无法达成一致的情况。针对这种情况，一种改进的Paxos算法被提出：从系统中选出一个节点作为Leader，只有Leader能够发起提案。这样，一次Paxos流程中只有一个Proposer，不会出现活锁的情况，此时只会出现例子中第一种情况。

## 4.5 选举机制（面试重点）

1）半数机制：集群中半数以上机器存活，集群可用。所以Zookeeper适合安装奇数台服务器。

2）Zookeeper虽然在配置文件中并没有指定Master和Slave。但是，Zookeeper工作时，是有一个节点为Leader，其他则为Follower，Leader是通过内部的选举机制临时产生的。

3）以一个简单的例子来说明整个选举的过程。

假设有五台服务器组成的Zookeeper集群，它们的id从1-5，同时它们都是最新启动的，也就是没有历史数据，在存放数据量这一点上，都是一样的。假设这些服务器依序启动，来看看会发生什么，如图5-8所示。



图5-8 Zookeeper的选举机制

（1）服务器1启动，发起一次选举。服务器1投自己一票。此时服务器1票数一票，不够半数以上（3票），选举无法完成，服务器1状态保持为LOOKING；

（2）服务器2启动，再发起一次选举。服务器1和2分别投自己一票并交换选票信息：此时服务器1发现服务器2的ID比自己目前投票推举的（服务器1）大，更改选票为推举服务器2。此时服务器1票数0票，服务器2票数2票，没有半数以上结果，选举无法完成，服务器1，2状态保持LOOKING

（3）服务器3启动，发起一次选举。此时服务器1和2都会更改选票为服务器3。此次投票结果：服务器1为0票，服务器2为0票，服务器3为3票。此时服务器3的票数已经超过半数，服务器3当选Leader。服务器1，2更改状态为FOLLOWING，服务器3更改状态为LEADING；

（4）服务器4启动，发起一次选举。此时服务器1，2，3已经不是LOOKING状态，不会更改选票信息。交换选票信息结果：服务器3为3票，服务器4为1票。此时服务器4服从多数，更改选票信息为服务器3，并更改状态为FOLLOWING；

（5）服务器5启动，同4一样当小弟。

## 4.6 写数据流程



# 第5章 企业面试真题

## 5.1 请简述ZooKeeper的选举机制

半数机制：集群中半数以上机器存活，集群可用。

## 5.2 ZooKeeper的监听原理是什么？

main()线程启动zk客户端，同时启动connect线程和listener线程，connect线程会将监听事件发送给zk客户端，在zk的注册监听器列表中添加监听事件，zk监听到事件或者路径发生变化，则会将该消息发送给listener线程，listener线程调用process()方法

## 5.3 ZooKeeper的部署方式有哪几种？集群中的角色有哪些？集群最少需要几台机器？

（1）部署方式单机模式、集群模式

（2）角色：Leader和Follower

（3）集群最少需要机器数：3

## 5.4 ZooKeeper的常用命令

ls create get delete set…