

大数据

大数据的概念

大数据：指无法在一定时间范围内用常规软件工具进行捕捉、管理和处理的数据集合，是需要新处理模式才能具有更强的决策力、洞察发现力和流程优化能力的海量、高增长率和多样化的信息资产。

存储单位按顺序：

bit、Byte、KB、MB、GB、TB、PB、EB、ZB、YB、BB、NB、DB

主要解决：

海量数据的存储

海量数据的分析和计算

大数据的特点

1、大量

截止目前，人类生产的所有印刷材料的数据量是200PB，而历史上全人类总共说过的话的数量大约是5EB。当前，典型个人计算机硬盘的容量为TB级别，而一些大企业的数据量接近EB级别。

2、高速

预计2020年，全球数据使用量将达到35.2ZB。如此海量的数据面前，处理数据的效率就是企业的生命。

天猫双11：

2017年3分01秒，交易额超过100亿

2018年2分，交易额超过100亿

3、多样

以数据库/文本为主的结构化数据，非结构化数据越来越多，包括网络日志，音频，视频，图片，地理位置等信息。这些多类型的数据对数据的处理能力提出了更高的要求。

4、低价值密度

价值密度的高低与数据总量的大小成反比，如何快速对有价值数据提纯成为目前大数据背景下待解决的难题。

大数据应用场景

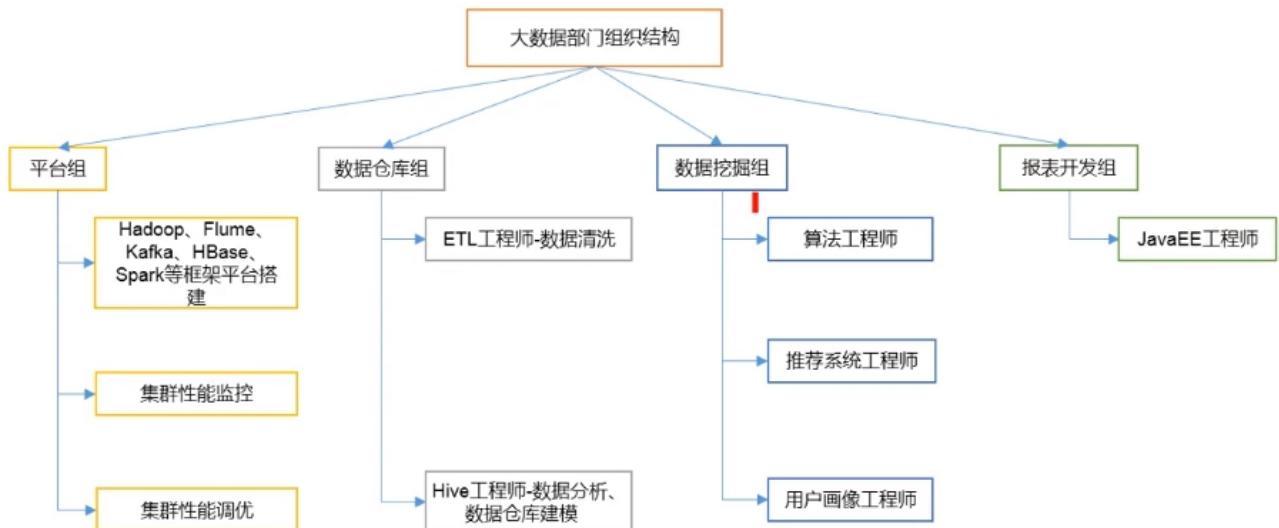
- 1、物流仓储：大数据分析系统助力商家精细化运营、提升销量，节约成本。
- 2、零售：分析用户消费习惯，为用户购买商品提供方便，从而提升商品销售，案例：纸尿布+啤酒。
- 3、旅游：深度结合大数据能力与旅游行业需求，共建旅游产业智慧管理，智慧服务和智慧营销的未来。
- 4、商品广告推荐：给用户推荐可能喜欢的商品。

- 5、保险：海量数据挖掘及风险预测，行业精准营销，提升精细化定价能力
- 6、金融：多维度体检用户特征，帮助金融机构推荐优质客户，防范欺诈风险。
- 7、房地产：打造精准投策与营销，选出更合适的地，建造更合适的楼，卖给更合适的人。

大数据部门业务流程分析

- 1、产品人员提需求(统计总用户数、日活用户数、回流用户数等)。
- 2、数据部门搭建数据平台，分析数据指标
- 3、数据可视化，报表展示，邮件，大屏幕展示

大数据部门组织结构



Hadoop

Hadoop_是什么

1. Hadoop是一个由Apache基金会所开发的分布式系统基础架构
2. 主要解决：海量数据的存储和分析计算问题
3. Hadoop通常是指---Hadoop生态圈，并不是单一的Hadoop生态圈如Hive,Hbase,Zookeeper,Cassandra,Solr....

发展历史

Lucene框架是Doug Cutting开创的开源软件，用java编写代码，实现与Google类似全文搜索功能，它提供了全文检索引擎的架构，包括完整的查询引擎和搜索引擎。

2001年年底Lucene称为Apache基金会的一个子项目

对于海量数据的场景，Lucene与Google同样有问题，存储困难，检索慢

学习和模仿Google解决这些问题的办法：微型版Nutch。

Google是Hadoop的思想之源，三大论文

GFS-----HDFS

Map-Reduce-----MR

BigTable-----HBase

奠定了Hadoop的核心。

2003-2004年，Google公开了部分GFS和MapReduce思想的细节，以此为基础Doug Cutting等人用了2年业余时间实现了DFS和MapReduce机制，使Nutch性能飙升。

2005年Hadoop作为Lucene的子项目Nutch的一部分正式引入Apache基金会

2006年3月，Map-Reduce和Nutch Distributed File System(NDFS)分别被纳入称为Hadoop的项目中

名字来源于他儿子的玩具大象。

Hadoop就此诞生，标志大数据时代的来临。

三大发行版本

Apache、Cloudera、Hortonworks

Apache版本是最原始的版本，用于学习

<http://hadoop.apache.org/releases.html>

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

Cloudera在大型互联网企业中用的较多

<https://www.cloudera.com/downloads/cdh/5-10-0.html>

下载地址：<http://archive-primary.cloudera.com/cdh5/cdh5/>

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，主要包括支持，咨询服务，培训

2009年Hadoop的创始人加盟Cloudera公司，其产品主要为CDH，Cloudera Manager，Cloudera Support

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

Cloudera Manager是集群的分发和管理监控平台，可以在几个小时内部署好一个Hadoop集群，并对集群节点和服务进行实时监控，Cloudera Support则是对Hadoop的技术支持

Cloudera的标价是每年每个节点4000美元，开发并贡献了实时处理大数据的Impala项目。

Hortonworks文档操作不错

<https://hortonworks.com/products/data-center/hdp/>

下载：<https://hortonworks.com/download/#data-platform>

2011年成立，是雅虎与硅谷风投公司Benchmark Capital合资组成。

公司成立之初就吸纳了25名到30名专门研究Hadoop的雅虎工程师，均在2005年时协助雅虎开发Hadoop，贡献了Hadoop80%的代码

主打产品是Hortonworks Data Platform(HDP)，也同样是开源的

优势（4高）

1. 高可靠性，Hadoop底层维护了3个数据副本，所以即使Hadoop某个计算元素或存储出现故障，也不会导致数据丢失。
2. 高扩展性，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.x和2.x区别

1.x的组成：

Common辅助工具，HDFS数据存储，MapReduce计算+资源调度

2.x的组成：

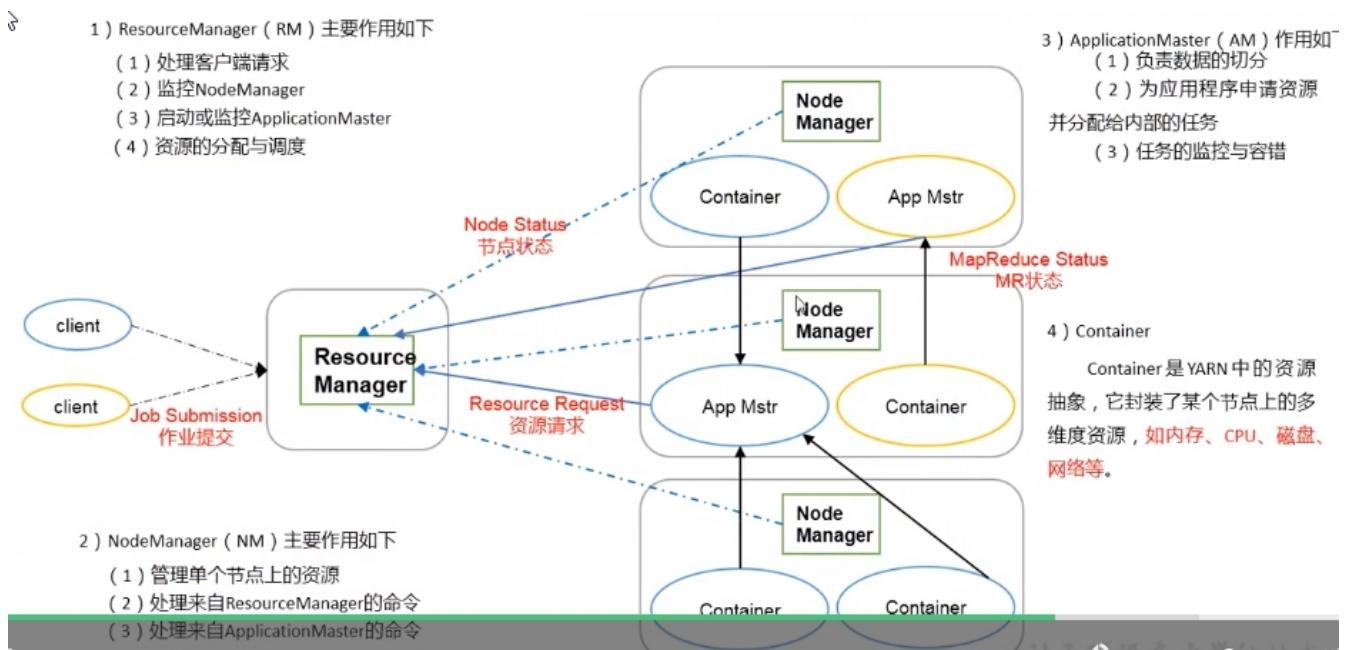
Common辅助工具，HDFS数据存储，MapReduce计算，Yarn资源调度

Hadoop_组成

HDFS

1. NameNode(nn)：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块所在的DataNode等。
2. DataNode(dn)：在本地文件系统存储文件块数据，以及块数据的校验和。
3. Secondary NameNode(2nn)：用来监控HDFS状态的辅助后台程序，每隔一段时间获取HDFS元数据的快照。

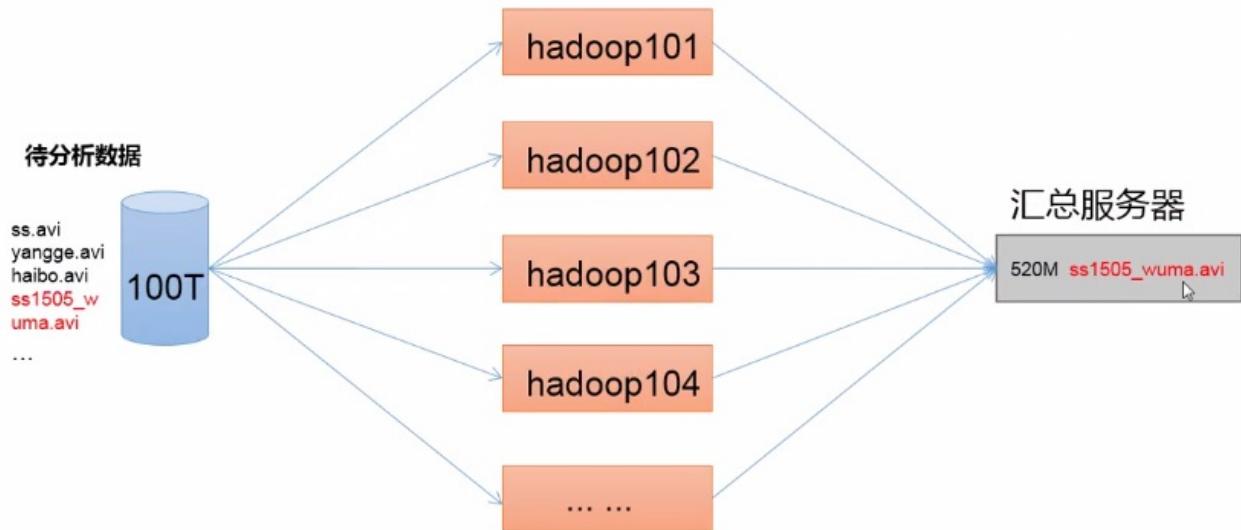
Yarn:



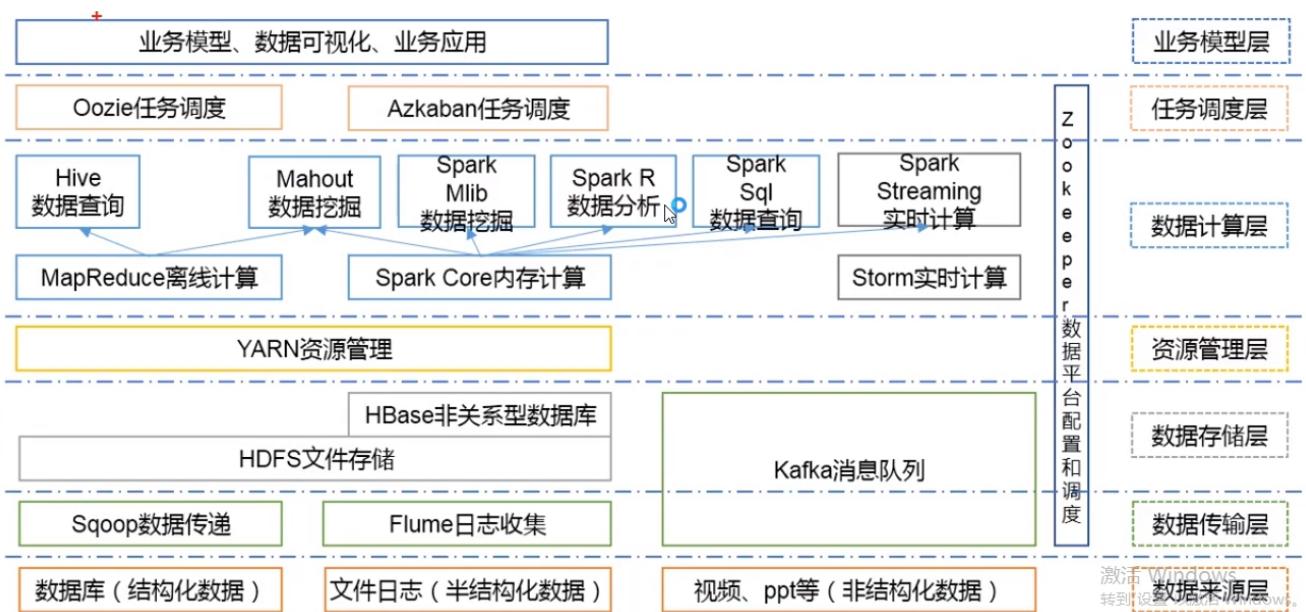
MapReduce：分为两个阶段Map和Reduce

Map阶段并行处理输入数据

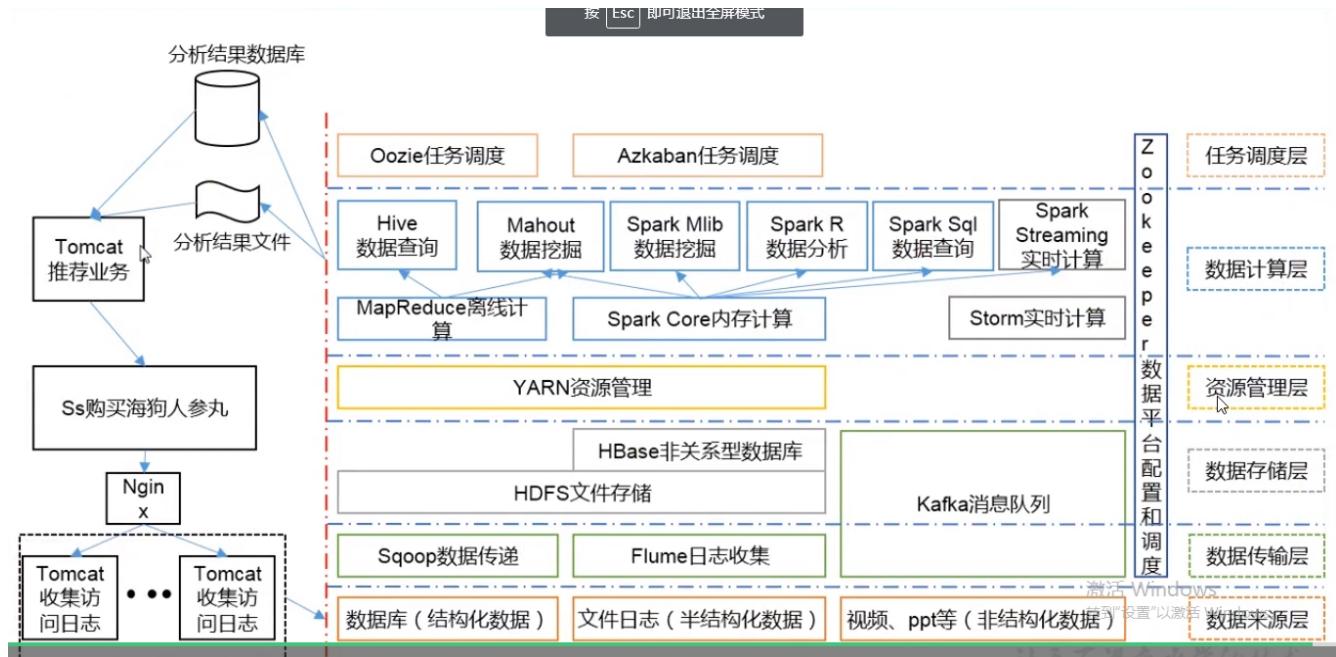
Reduce阶段对Map结果进行汇总



大数据技术生态体系



推荐系统框架图



环境搭建

虚拟机准备

1、新建虚拟机

2、配置网络

修改ifcfg-ens33文件

```

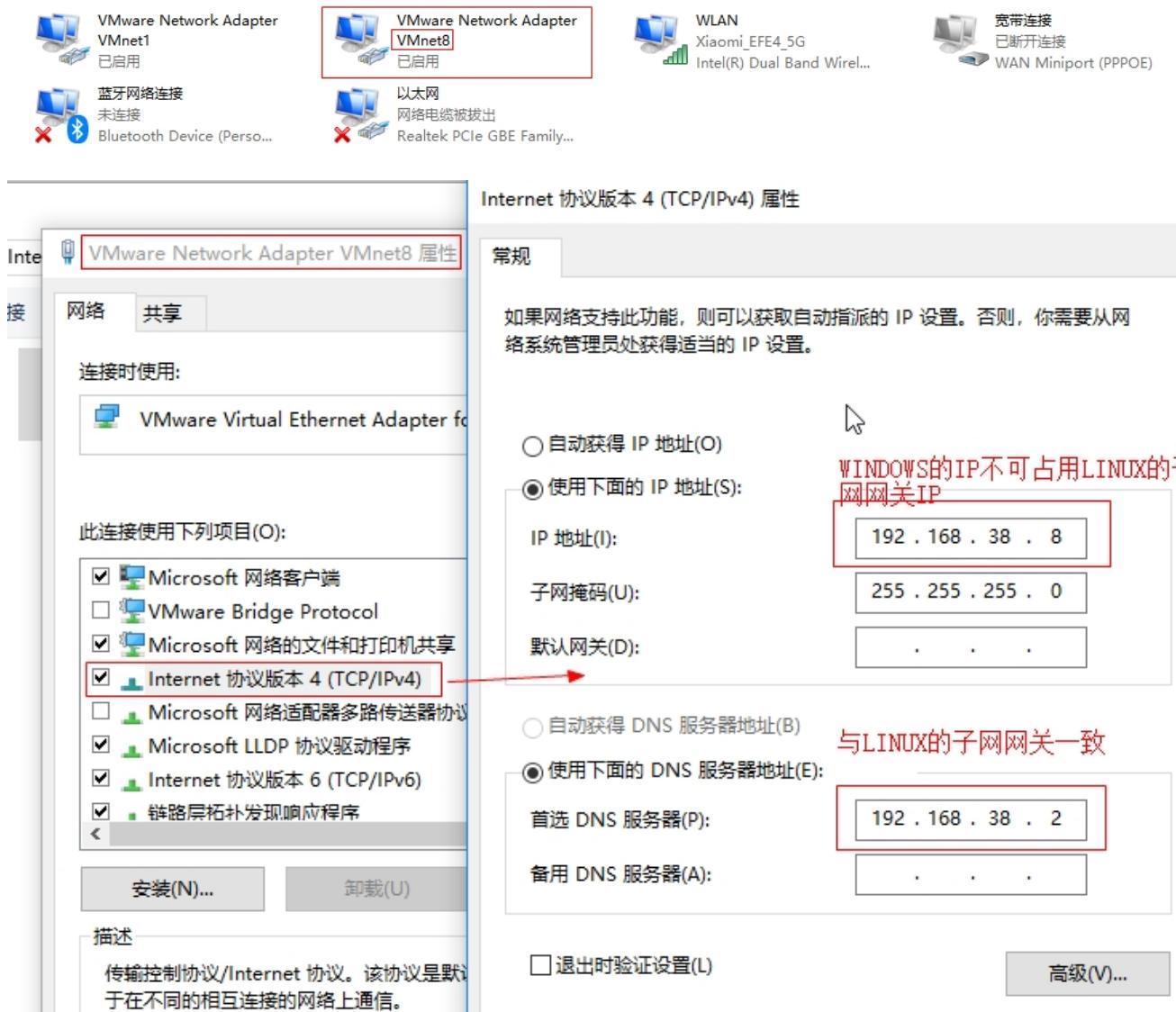
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
NETMASK=255.255.255.0
IPADDR=192.168.38.11
#GATEWAY和DNS1可以一样
GATEWAY=192.168.38.2
DNS1=192.168.38.2
PEERDNS=yes
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
UUID=c04c95f5-dfe6-4d35-8b6e-10aa8f414b0b
DEVICE=ens33
ONBOOT=yes

```

注意:



在WINDOWS中配置:



3、修改主机名

修改hostname文件，在文件中添加主机名如tarena.com

```
[root@tarena hadoop-2.9.2]# vim /etc/hostname
```

修改hosts文件，添加主机名，ip映射

```
[root@tarena hadoop-2.9.2]# vim /etc/hosts
-----进入添加最后一行-----
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1          localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.38.11 tarena
```

在WINDOWS的C盘windows--->system32---->driven----->etc----->hosts中添加

```
192.168.38.11  tarena
#ip地址以实际情况为准
```

4、关闭防火墙

```
#启动:  
systemctl start firewalld  
#关闭:  
systemctl stop firewalld  
#查看状态:  
systemctl status firewalld  
#开机禁用 :  
systemctl disable firewalld  
#开机启用 :  
systemctl enable firewalld
```

5、创建新用户并配置新用户有root权限

```
#创建新用户  
adduser [用户名]  
#修改新用户的密码  
passwd [用户名]  
#添加sudoers文件可写权限  
chmod -v u+w /etc/sudoers  
#修改sudoers文件  
#使用vim编辑器打开sudoers文件  
vim /etc/sudoers  
#在sudoers文件中找到如下位置并添加如下内容  
[用户名]      ALL=(ALL)      ALL (使用sudo时不用输密码，把最后一个ALL改为NOPASSWD:ALL即可)  
#收回sudoers文件可写权限  
chmod -v u-w /etc/sudoers
```

```
#3.新建用户同时增加工作组  
#新建testuser用户并增加到testgroup工作组  
useradd -g testgroup testuser
```

```
#注: : -g 所属组 -d 家目录 -s 所用的SHELL
```

```
#4.给已有的用户增加工作组  
usermod -G groupname username
```

```
#5.临时关闭  
#在/etc/shadow文件中属于该用户的行的第二个字段（密码）前面加上就可以了。想恢复该用户，去掉即可  
#或者使用如下命令关闭用户账号：  
passwd testuser -l  
#重新释放：  
passwd testuser -u
```

```
#删除用户  
#6.永久性删除用户账号  
userdel testuser  
groupdel testgroup  
# (强制删除该用户的主目录和主目录下的所有文件和子目录)  
usermod -G testgroup testuser
```

```
#我个人推测是在root用户下su 切换到xiaoming用户，然后在xiaoming用户下又切换回root，但是xiaoming用户还被某个进程占用着，所以进程不死，用户del不掉。  
#所以我们在命令行中使用Ctrl+D 来退出当前的登录，然后在按一次Ctrl+D退出xiaoming用户的登录，这时候我们回到的是root的用户下了，在使用  
userdel -r xiaoming
```

```
#查看用户和用户组  
#7.显示用户信息  
id user  
cat /etc/passwd
```

```
#补充:查看用户和用户组的方法  
#用户列表文件:  
/etc/passwd  
#用户组列表文件:  
/etc/group  
#查看系统中有哪些用户:  
cut -d : -f 1 /etc/passwd  
#查看可以登录系统的用户:  
cat /etc/passwd | grep -v /sbin/nologin | cut -d : -f 1  
#查看用户操作:  
w命令(需要root权限)  
#查看某一用户:  
w 用户名  
#查看登录用户:  
who  
#查看用户登录历史记录:  
last
```

6、在/opt目录下创建文件夹

在/opt目录下创建两个文件夹software和module

```
mkdir software  
mkdir module
```

JDK安装

下载jdk和hadoop通过xftp传输到linux的/opt/software

如果传输过程中出现传输状态错误需要修改文件夹权限

```
chmod 777 software (需要赋予权限的文件夹)
```

解压jdk

```
tar -zxvf jdk-8u181-linux-x64.tar.gz -C /opt/module/
```

配置环境变量

```
pwd  
#复制当前目录  
#进入root  
vim /etc/profile  
#在最末尾添加  
export JAVA_HOME=/opt/module/jdk1.8.0_181  
export PATH=$PATH:$JAVA_HOME/bin  
#加载修改后的profile文件  
source /etc/profile  
#测试环境是否正确  
java -version
```

Hadoop安装

解压

```
tar -zxvf hadoop-2.9.2.tar.gz -C /opt/module/
```

进入解压后的文件夹

```
pwd #记录路径地址
```

配置环境变量

```
vim /etc/profile
```

添加配置

```
#HADOOP_HOME
```

加载配置文件

```
source /etc/profile
```

检查配置情况

```
hadoop  
hadoop version
```

Hadoop的目录结构

重点掌握：bin, etc, sbin

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/>

查看官方文档直接在地址栏里：<https://hadoop.apache.org/docs/r2.9.1/>

阅读手册

本地模式

Grep官方案例

在Hadoop安装目录创建input文件夹

```
mkdir input
```

将etc下的所有.xml文件复制到input文件夹

```
cp etc/hadoop/*.xml input/
```

在有环境变量的情况下可以直接

```
#最后一个正则表达式表示需要匹配的文件名，这里千万注意在执行之前output文件夹一定不能存在，否则报出异常  
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.2.jar grep input/ output  
'dfs[a-z.]+'
```

执行完成后会在安装目录中看到output文件夹

```
drwxr-xr-x. 2 501 dialout 194 11月 13 23:15 bin
drwxr-xr-x. 3 501 dialout 20 11月 13 23:15 etc
drwxr-xr-x. 2 501 dialout 106 11月 13 23:15 include
drwxr-xr-x. 2 root root 187 12月 5 09:12 input
drwxr-xr-x. 3 501 dialout 20 11月 13 23:15 lib
drwxr-xr-x. 2 501 dialout 239 11月 13 23:15 libexec
-rw-r--r--. 1 501 dialout 106210 11月 13 23:15 LICENSE.txt
-rw-r--r--. 1 501 dialout 15917 11月 13 23:15 NOTICE.txt
drwxr-xr-x. 2 root root 88 12月 5 09:16 output
-rw-r--r--. 1 501 dialout 1366 11月 13 23:15 README.txt
drwxr-xr-x. 3 501 dialout 4096 11月 13 23:15 sbin
drwxr-xr-x. 4 501 dialout 31 11月 13 23:15 share
```

进入output, 如果刚才的执行成功可以看见

```
-rw-r--r--. 1 root root 11 12月 5 09:16 part-r-00000
-rw-r--r--. 1 root root 0 12月 5 09:16 _SUCCESS
```

查看其中的part-r-00000可以看到匹配的文件被复制到其中

```
cat part-r-00000
```

看见结果

```
1      dfsadmin
```

WordCount官方案例

1、在hadoop-2.9.2文件下面创建一个wcinput文件夹

```
mkdir wcinput
```

2、在wcinput文件下创建一下wc.input文件

```
touch wc.input
```

3、编辑wc.input文件，可以随意写

4、回到安装目录

5、执行程序

```
hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.2.jar wordcount wcinput/
wcoutput
```

6、查看结果

```
cd wcoutput/
```

```
-rw-r--r--. 1 root root 101 12月 5 09:37 part-r-00000
-rw-r--r--. 1 root root 0 12月 5 09:37 _SUCCESS
```

7、进入part-r-00000

```
cat part-r-00000
```

结果:

```
clare 1
david 1
dick 1
jack 1
jackson 1
john 1
johnson 1
lily 1
lucy 1
meimei 1
rose 1
stan 1
wood 1
```

伪分布式

启动HDFS并运行MR程序

进入/opt/module/hadoop-2.9.2/etc/hadoop

按照官方文档修改core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <!--以hdfs协议访问主机名: 端口号-->
    <value>hdfs://192.168.38.11:9000</value>
  </property>
  <!--数据的存放默认是根目录下/tmp, 改为如下-->
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/opt/module/hadoop-2.9.2/data/tmp</value>
  </property>
</configuration>
```

修改hdfs-site.xml

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <!--设置副本数量-->
    <value>1</value>
  </property>
</configuration>
```

修改hadoop-env.sh, 修改其中JAVA_HOME的值

```
export JAVA_HOME=/opt/module/jdk1.8.0_181
```

格式化, 注意路径, 不要进错了, 没事别老格式化

```
[root@localhost hadoop-2.9.2]# bin/hdfs namenode -format
```

启动namenode, 注意路径, 不要进错了

```
[root@localhost hadoop-2.9.2]# sbin/hadoop-daemon.sh start namenode
```

查看是否成功, jps属于JDK, 如果JDK没有安装成功则会报错, 如果jps没有生效可以试一试source /etc/profile

```
[root@localhost hadoop-2.9.2]# jps
```

启动datanode

```
root@localhost hadoop-2.9.2]# sbin/hadoop-daemon.sh start datanode
```

测试网页：

<http://192.168.38.11:50070>

HDFS的简单操作

向HDFS中创建一个文件夹

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -mkdir -p /user/tarena/input
```

完成后可以在网页中查看到

The screenshot shows the HDFS Web UI interface. At the top, there is a green navigation bar with tabs: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities (which is highlighted with a red box). Below the navigation bar, the title 'Browse Directory' is displayed. In the center, there is a search bar with the path '/user/tarena/' and a 'Go!' button. To the right of the search bar are three icons: a folder, a file, and a magnifying glass. Below the search bar, there are filters: 'Show 25 entries' and a 'Search:' input field. The main area displays a table of directory entries. The columns are: Permission, Owner, Group, Size, Last Modified, Replication, Block Size, and Name. One entry is shown: 'drwxr-xr-x root supergroup 0 B Dec 05 21:57 input'. The 'input' folder is highlighted with a red box. At the bottom of the table, it says 'Showing 1 to 1 of 1 entries'. On the far right, there are buttons for 'Previous', '1' (highlighted in blue), and 'Next'. The footer of the page says 'Hadoop, 2018.' and has a small logo.

同样可以使用命令查看HDFS根目录下的情况

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -ls /
-----
Found 1 items
drwxr-xr-x  - root supergroup          0 2018-12-05 21:57 /user
```

以及多级查看

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -ls -R /
-----
drwxr-xr-x  - root supergroup          0 2018-12-05 21:57 /user
drwxr-xr-x  - root supergroup          0 2018-12-05 21:57 /user/tarena
drwxr-xr-x  - root supergroup          0 2018-12-05 21:57 /user/tarena/input
```

将本地的文件上传到HDFS中的input文件夹中

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -put wcinput/wc /user/tarena/input
```

Browse Directory

/user/tarena/input								Go!			
Show 25 entries								Search:			
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name			
	-rw-r--r--	root	supergroup	0 B	Dec 05 22:04	1	128 MB	wc.input			
权限 Showing 1 to 1 of 1 entries											

Hadoop, 2018.

可以下载，也可以通过查看

如果下载显示错误，可以将主机名替换成ip地址，是因为在windows的hosts文件中没有配置主机名和ip映射

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -cat /user/tarena/input/wc
-----
jack rose dick meimei lily lucy
john jackson johnson wood stan david
clare
```

用wordcount作为测试

```
[root@tarena hadoop-2.9.2]# bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.2.jar wordcount /user/tarena/input /user/tarena/output
```

在HDFS中查看

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -cat /user/tarena/output/p*
```

Log日志查看和NN格式化前强调

```
drwxr-xr-x. 2      501 dialout   194 11月 13 23:15 bin
drwxr-xr-x. 3 root    root     17 12月 5 15:30 data
drwxr-xr-x. 3      501 dialout   20 11月 13 23:15 etc
drwxr-xr-x. 2      501 dialout   106 11月 13 23:15 include
drwxr-xr-x. 2 root    root     187 12月 5 09:12 input
drwxr-xr-x. 3      501 dialout   20 11月 13 23:15 lib
drwxr-xr-x. 2      501 dialout   239 11月 13 23:15 libexec
-rw-r--r--. 1      501 dialout 106210 11月 13 23:15 LICENSE.txt
drwxr-xr-x. 2 hwhadoop root    4096 12月 6 18:07 logs
-rw-r--r--. 1      501 dialout 15917 11月 13 23:15 NOTICE.txt
drwxr-xr-x. 2 root    root     88 12月 5 09:16 output
-rw-r--r--. 1      501 dialout 1366 11月 13 23:15 README.txt
drwxr-xr-x. 3      501 dialout 4096 11月 13 23:15 sbin
drwxr-xr-x. 4      501 dialout 31 11月 13 23:15 share
drwxr-xr-x. 2 root    root     32 12月 5 22:08 wcinput
drwxr-xr-x. 2 root    root     88 12月 5 09:37 wcoutput
```

如果需要格式化服务器，顺序执行2步操作
首先将jps命令下看到的nameNode和
dataNode正常关闭
再将此两个文件夹删除
再格式化服务器
否则起100次也没用

```
[root@tarena hadoop-2.9.2]# cd logs
[root@tarena logs]# ll
总用量 424
-rw-r--r-- 1 root      root      30090 12月  5 15:42 hadoop-hwhadoop-datanode-localhost.localdomain.log
-rw-r--r-- 1 root      root      720 12月  5 15:35 hadoop-hwhadoop-datanode-localhost.localdomain.out
-rw-r--r-- 1 root      root    119336 12月  6 18:07 hadoop-hwhadoop-datanode-tarena.log
-rw-r--r-- 1 root      root      720 12月  6 18:07 hadoop-hwhadoop-datanode-tarena.out
-rw-r--r-- 1 root      root      720 12月  5 16:41 hadoop-hwhadoop-datanode-tarena.out.1
-rw-r--r-- 1 root      root      720 12月  5 16:02 hadoop-hwhadoop-datanode-tarena.out.2
-rw-r--r-- 1 root      root      720 12月  5 15:55 hadoop-hwhadoop-datanode-tarena.out.3
-rw-r--r-- 1 root      root    33849 12月  5 15:42 hadoop-hwhadoop-namenode-localhost.localdomain.log
-rw-r--r-- 1 root      root      720 12月  5 15:32 hadoop-hwhadoop-namenode-localhost.localdomain.out
-rw-r--r-- 1 root      root    182041 12月  6 18:08 hadoop-hwhadoop-namenode-tarena.log
-rw-r--r-- 1 root      root      6058 12月  6 18:08 hadoop-hwhadoop-namenode-tarena.out
-rw-r--r-- 1 root      root      6058 12月  5 21:45 hadoop-hwhadoop-namenode-tarena.out.1
-rw-rw-r-- 1 hwhadoop hwhadoop   2790 12月  5 16:41 hadoop-hwhadoop-namenode-tarena.out.2
-rw-r--r-- 1 root      root      720 12月  5 16:02 hadoop-hwhadoop-namenode-tarena.out.3
-rw-r--r-- 1 root      root      720 12月  5 16:01 hadoop-hwhadoop-namenode-tarena.out.4
-rw-r--r-- 1 root      root      720 12月  5 15:54 hadoop-hwhadoop-namenode-tarena.out.5
-rw-rw-r-- 1 hwhadoop hwhadoop     0 12月  5 16:41 SecurityAuth-hwhadoop.audit
-rw-r--r-- 1 root      root      0 12月  5 15:32 SecurityAuth-root.audit
```

可在此参看NameNode和DataNode的日志信息

NameNode格式化注意事项

要先关闭NameNode和DataNode之后再删除logs和data文件夹

为什么不能一直格式化NameNode，格式化NameNode要注意什么

可以进入

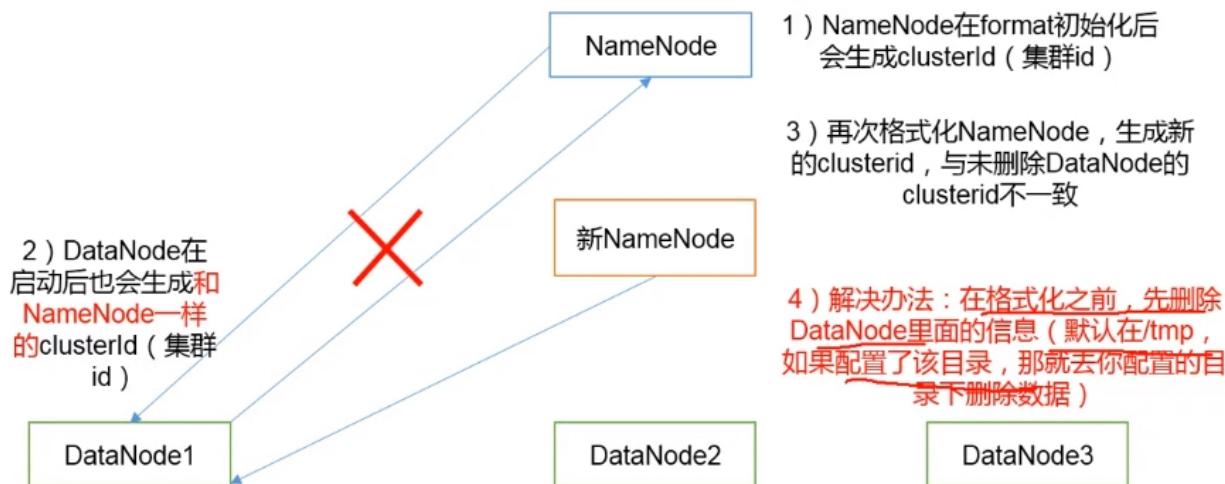
```
[root@tarena hadoop-2.9.2]# cd /opt/module/hadoop-2.9.2/data/tmp/dfs/data/current
-----
BP-178131963-127.0.0.1-1543995036045 VERSION
-----
[root@tarena current]# cat VERSION
#Thu Dec 06 18:07:42 CST 2018
storageID=DS-06574629-7333-41cc-a89e-af4435885594
#记住这里
clusterID=CID-cea8954d-9790-4bd3-96e7-2534420f85b8
cTime=0
datanodeuuid=837887ab-cdf6-4d6f-aca7-6da4d286c8d2
storageType=DATA_NODE
layoutVersion=-57
```

```
[root@tarena current]# cd /opt/module/hadoop-2.9.2/data/tmp/dfs/name/current
-----
[root@tarena current]# cat VERSION
-----
#Thu Dec 06 18:07:35 CST 2018
namespaceID=504371809
clusterID=CID-cea8954d-9790-4bd3-96e7-2534420f85b8
cTime=1543995036045
storageType=NAME_NODE
blockpoolID=BP-178131963-127.0.0.1-1543995036045
layoutVersion=-63
```

可以发现NameNode和DataNode的clusterID是一样的，这是正确的。

但如果随意格式化NameNode很可能造成clusterID就不一样了，从而导致一个起一个被关闭：

DataNode和NameNode进程同时只能有一个工作问题分析



启动YARN并运行MR程序

- 1、配置集群在Yarn上运行
- 2、启动、测试集群增、删、查
- 3、在Yarn上执行WordCount案例

步骤：

修改Yarn-env.sh文件添加JDK

```
[root@tarena hadoop-2.9.2]# cd etc/hadoop/
[root@tarena hadoop]# echo $JAVA_HOME
/opt/module/jdk1.8.0_181
[root@tarena hadoop]# vim yarn-env.sh
#文件中配置
# some Java parameters
export JAVA_HOME=/opt/module/jdk1.8.0_181
if [ "$JAVA_HOME" != "" ]; then
    #echo "run java in $JAVA_HOME"
    JAVA_HOME=$JAVA_HOME
fi
```

修改yarn-site.xml文件

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <!--这里是当前的主机名以实际情况为准-->
    <value>tarena</value>
  </property>
</configuration>
```

修改mapred-env.sh, 同样找里面的JAVA_HOME改为自己的JDK

```
export JAVA_HOME=/opt/module/jdk1.8.0_181
```

将mapred-site.xml.template改名为mapred-site.xml

```
[root@tarena hadoop]# mv mapred-site.xml.template mapred-site.xml
```

```
<configuration>
    <!--指定MR运行在Yarn上-->
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

启动集群，启动之前保证NameNode和DataNode已经启动

启动resourcemanager

```
[root@tarena hadoop-2.9.2]# sbin/yarn-daemon.sh start resourcemanager
starting resourcemanager, logging to /opt/module/hadoop-2.9.2/logs/yarn-hwhadoop-
resourcemanager-tarena.out
[root@tarena hadoop-2.9.2]# jps
```

```
-----  
1648 DataNode  
1554 NameNode  
2003 ResourceManager  
2045 Jps
```

在启动nodemanager

```
[root@tarena hadoop-2.9.2]# sbin/yarn-daemon.sh start nodemanager
starting nodemanager, logging to /opt/module/hadoop-2.9.2/logs/yarn-hwhadoop-nodemanager-
tarena.out
[root@tarena hadoop-2.9.2]# jps
-----  
1648 DataNode  
1554 NameNode  
2003 ResourceManager  
2278 NodeManager  
2326 Jps
```

浏览器中输入<http://192.168.38.11:8088/cluster>看见展示页面表示成功

添加一个任务测试

先删除HDFS下的output文件夹

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -rm -r /user/tarena/output
```

执行WordCount

```
[root@tarena hadoop-2.9.2]# hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-
2.9.2.jar wordcount /user/tarena/input /user/tarena/output
```

在页面中显示

StartTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster	Progress
Thu Dec 6 19:20:32 +0800 2018	Thu Dec 6 19:21:15 +0800 2018	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0	

但点击后面的History查看历史是无法显示的，是因为还没有配置历史服务器

配置历史服务器

修改etc目录下的mapred-site.xml

```
[root@tarena hadoop-2.9.2]# vim etc/hadoop/mapred-site.xml
```

添加内容

```
<!--历史服务器的地址-->
<property>
    <name>mapreduce.jobhistory.address</name>
    <value>192.168.38.11:10020</value>
</property>
<!--历史服务器的web端地址-->
<property>
    <name>mapreduce.jobhistory.webapp.address</name>
    <value>192.168.38.11:19888</value>
</property>
```

打开历史服务器

```
[root@tarena hadoop-2.9.2]# sbin/mr-jobhistory-daemon.sh start historyserver
-----
starting historyserver, logging to /opt/module/hadoop-2.9.2/logs/mapred-hwhadoop-
historyserver-tarena.out
[root@tarena hadoop-2.9.2]# jps
-----
1648 DataNode
1554 NameNode
2003 ResourceManager
2278 NodeManager
#可以看到历史服务器启动
3566 JobHistoryServer
3614 Jps
```

在页面上点击History测试

配置日志聚集

日志聚集的概念：

应用运行完成以后，将程序运行日志信息上传到HDFS系统上。

日志聚集功能的好处：

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，需要重启NodeManager、ResourceManager和HistoryManager

```
[root@tarena hadoop-2.9.2]# sbin/mr-jobhistory-daemon.sh stop historyserver
[root@tarena hadoop-2.9.2]# sbin/yarn-daemon.sh stop nodemanager
[root@tarena hadoop-2.9.2]# sbin/yarn-daemon.sh stop resourcemanager
```

步骤如下：

- 1、配置yarn-site.xml，添加

```

<!--开启日志-->
<property>
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
</property>
<!--日志保留7天-->
<property>
    <name>yarn.log-aggregation.retain-seconds</name>
    <value>604800</value>
</property>

```

2、打开NodeManager、 ResourceManager和HistoryManager

3、删除output，并再次执行wordcount

```
[root@tarena hadoop-2.9.2]# bin/hdfs dfs -rm -r /user/tarena/output
```

```
[root@tarena hadoop-2.9.2]# hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.2.jar wordcount /user/tarena/input /user/tarena/output
```

4、查看任务产生的日志

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中相应的值

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，可以根据实际情况修改配置文件。

完全分布式

虚拟机环境准备

3个服务器节点（关闭防火墙，静态IP，主机名称，用户名，权限）

安装JDK

安装HADOOP

配置JDK和Hadoop环境变量

配置集群

单点启动

ssh免密登录

群起测试

scp案例

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，

2、 基本语法

```
scp -r $pid/$fname $user@hadoop$host:$pdir/$fname  
命令 递归 源文件路径 目的用户@主机:目的路径/名称
```

注意在做之前：

- 如果要操作的文件不属于当前用户，要切到root或将当前目录修改成所在用户

```
sudo chown hwhadoop:hwhadoop module/ -R
```

- 要能成功连接到其他服务器在当前服务器的hosts文件中补习加入集群中其他服务器

```
sudo vim /etc/hosts
```

```
-----  
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4  
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6  
192.168.38.13 tarena03  
192.168.38.14 tarena01  
192.168.38.15 tarena04  
...
```

3、 实际操作

- 站在源服务器上，将当前服务器上的/opt/module目录下的软件拷贝到新的服务器上

```
scp -r module root@tarena04:/opt/
```

- 站在新服务器上，将源服务器上的/opt/module目录下的内容拉取到当前服务器上

```
#这里要根据实际权限情况加入sudo  
sudo scp -r hwhadoop@tarena03:/opt/module ./
```

- 站在中间服务器上，将源服务器上的/opt/module目录下的内容推送到目标服务器上

```
scp -r hwhadoop@tarena03:/opt/module/ root@tarena04:/opt/
```

- 完成后发现还是无法运行java -version，是因为profile还没有拷贝过来

```
sudo scp -r root@tarena03:/etc/profile /etc/profile
```

- 记得

```
source /etc/profile
```

rsync案例

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

rsync和scp的区别，用sync做文件的复制比scp的速度快，rsync只对差异文件更新，scp是把所有文件复制

- 基本语法

```
rsync -rvl $pdir/$fname $user@hadoop$host:$pdir/$fname
```

命令 选项参数 源文件 目标用户@主机:目的路径/名称

选项	功能
-r	递归
-v	显示复制功能
-l	拷贝符号连接

- 具体操作

```
#在源服务器建立/opt/temp并在内部创建文件，使用下列命令同步到目标服务器上  
rsync -rvl temp/ root@tarena04:/opt/
```

注意：

要使用rsync必须要先安装该命令，且原服务器和目标服务器上都必须安装才能连接成功

```
#安装rsync命令  
yum install rsync -y
```

集群分发脚本xsync

需求：循环复制文件到所有节点的相同目录下

需求分析：

rsync命令原始拷贝

```
rsync -rvl /opt/module root@tarena01:/opt/
```

配置rsync脚本

在~目录下建立bin目录，并在bin目录下建立xsync文件。

```
#!/bin/bash  
#1 获取输入参数的个数，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目录  
pdir=`cd -P $(dirname $p1); pwd`
```

```

echo pdir=$pdir

#4获取当前用户名
user=`whoami` 

#5循环
rsync -rvl $pdir/$fname $user@tarena04:$pdir
rsync -rvl $pdir/$fname $user@tarena01:$pdir
#正确应该用一下循环，为了测试方便用了上述设置
#for((host=101; host<104; host++));do
#    rsync -rvl $pdir/$fname $user@tarena$host:$pdir
#done

```

最后chmod a+x xsync给文件添加执行权限即可。

存放位置：

如果将脚本存放到/home/hwhadoop/bin位置，可以在系统的任何位置直接执行，

也可以放在环境变量中的其他可以直接访问的目录下

```
echo $PATH      #就可以看见那些目录是可以的
```

集群配置

1、集群部署规划

tarena01	tarena02	tarena03
HDFS NameNode,DataNode	DataNode	SecondaryNameNode,DataNode
YARN NodeManager	ResourceManager,NodeManager	NodeManager

2、配置集群

配置core-site.xml，进入/opt/module/hadoop-2.9.2/etc/hadoop/core-site.xml

```

<property>
    <name>fs.defaultFS</name>
    <!--以hdfs协议访问主机名：端口号-->
    <!--根据情况修改这里的配置-->
    <value>hdfs://tarena03:9000</value>
</property>
<!--数据的存放默认是根目录下/tmp，改为如下-->
<property>
    <name>hadoop.tmp.dir</name>
    <value>/opt/module/hadoop-2.9.2/data/tmp</value>
</property>

```

配置HDFS：

配置hadoop-env.sh

只需要添加JAVA_HOME的值就可以

配置hdfs-site.xml

使用默认3个副本节点，也可以直接将副本设置删除，将启用默认设置

```
<property>
    <name>dfs.namenode.secondary.http-address</name>
    <!--根据规划配置这里-->
    <value>tarena04:50090</value>
</property>
```

配置YARN：

配置yarn-env.sh

只需要添加JAVA_HOME的值就可以

配置yarn-site.xml

```
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
    <name>yarn.resourcemanager.hostname</name>
    <!--这里以规划进行配置-->
    <value>tarena01</value>
</property>
```

配置MapReduce：

配置mapred-env.sh

只需要添加JAVA_HOME的值就可以

配置mapred-site.xml，先将mapred-site.xml.template改名为mapred-site.xml

进入，修改

(伪分布式已经配置过了)

```
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
```

同步配置到集群：

```
#注意这里的路径是/opt/module/hadoop-2.9.2/etc
[hwhadoop@tarena03 etc]$ xsync hadoop/
```

集群单节点启动

在主服务器上格式化namenode，但在格式化之前记得每个节点/opt/module/hadoop-2.9.2/目录下的data和logs目录一定要先删除干净，停掉所有的namenode,datanode,resourceManager,nodeManager....只剩下jps。

完成后重新启动namenode, datanode，只需namenode服务器节点启动namenode，其他服务器节点只需要启动datanode即可

并通过网页tarena03:50070测试

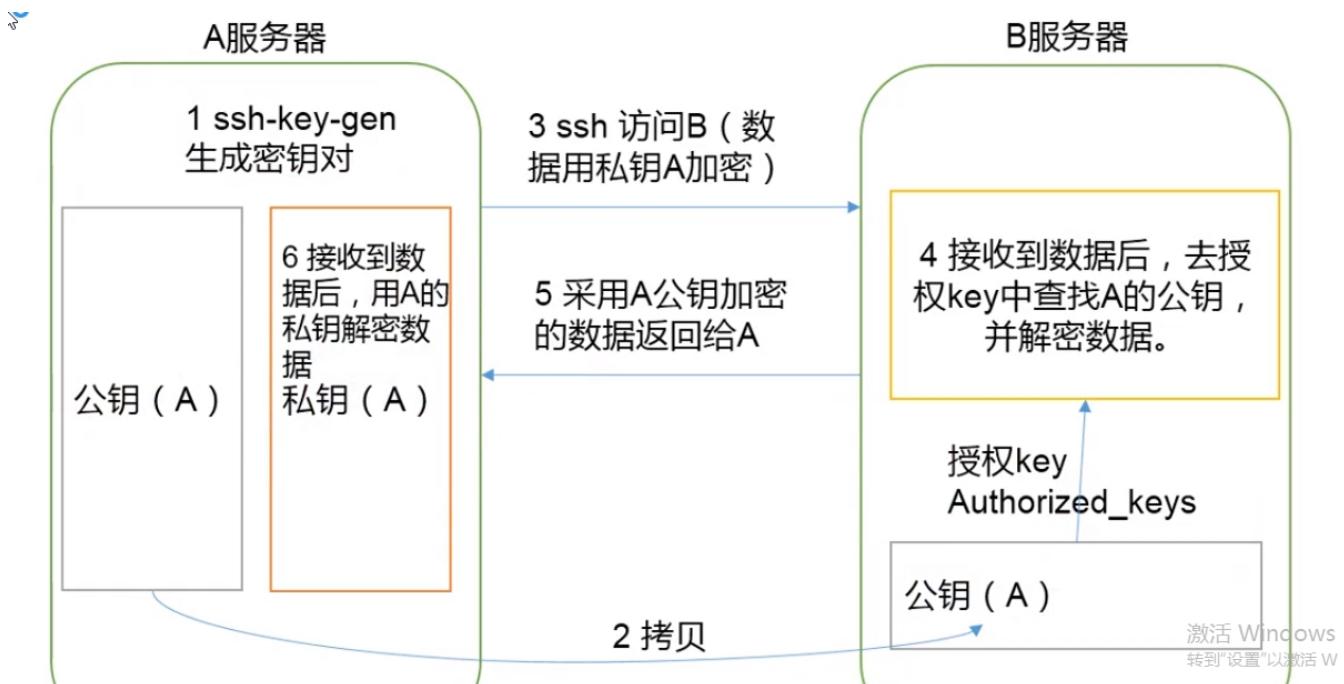
集群ssh配置

配置SSH

1、基本语法

ssh 另一台电脑的ip地址

2、实现免密登录



步骤:

1、进入用户目录

```
#进入用户家目录  
cd  
#查看目录下所有文件  
ls -al  
#进入.ssh文件夹  
cd .ssh  
#查看.ssh下known_hosts文件的内容, 这里存放已访问过的节点  
[hwhadoop@tarena101 .ssh]$ cat known_hosts  
-----  
tarena102,192.168.40.102 ecdsa-sha2-nistp256  
AAAAE2VjZHNhLXNoYTItbm1zdHAYNTYAAAIBm1zdHAYNTYAAABBBJCBw+bUuM5SpkYc6EFZgtgC42me+zFufwigQRSw0  
ZISFVuufx6izu2bZcmMJ5cg3Qch7L3nnEUM6Rx5GqRLAo=
```

tarena103,192.168.40.103 ecdsa-sha2-nistp256

```
AAAAE2VjZHNhLXNoYTItbm1zdHAYNTYAAAIBm1zdHAYNTYAAABBPCKxcd4E8pynu16twEya0ARtMMYgCz3f2LD0L4iK+  
ALtbyGJHsfwbk69xeJ1e/hBdMLdHRVJHbKfkowtDncitM=
```



```
#输入命令生成公钥和私钥  
[hwhadoop@tarena03 .ssh]$ ssh-keygen -t rsa  
#创建完成后出现文件  
-rw----- 1 hwhadoop hwhadoop 1675 12月 26 20:13 id_rsa #私钥  
-rw-r--r-- 1 hwhadoop hwhadoop 400 12月 26 20:13 id_rsa.pub #公钥  
-rw-r--r-- 1 hwhadoop hwhadoop 372 12月 26 18:07 known_hosts
```

2、拷贝公钥到其他服务器

```
#在tarena03下执行，将公钥拷贝到tarena01
[hwhadoop@tarena03 .ssh]$ ssh-copy-id tarena01

#在tarena01下看见
[hwhadoop@tarena01 .ssh]$ ll
-----
总用量 8
-rw-----. 1 hwhadoop hwhadoop 399 12月 10 12:43 authorized_keys
-rw-r--r--. 1 hwhadoop hwhadoop 184 12月  8 13:48 known_hosts
```

3、拷贝到其他服务器，实现免密登录

```
#都不再需要输入密码
ssh tarena01
ssh tarena04
```

4、同时也要将公钥拷贝到自己，否则自己访问自己也需要输入密码，因为linux在判断是否需要密码都需要去authorized_keys，

5、tarena03上有namenode，必须能够让集群其他服务器无密访问

6、tarena01也需要将公钥发送给集群其他节点，因为它上面有resourcemanager，使用上述同样方法

7、tarena03的root用户也需要设置ssh免密登录，在root用户权限下进行上述所有操作。

表 2-4

<u>known hosts</u>	记录 ssh 访问过计算机的公钥(public key)。
<u>id_rsa</u>	生成的私钥。
<u>id_rsa.pub</u>	生成的公钥。
<u>authorized keys</u>	存放授权过得无密登录服务器公钥。

集群群起

1、配置slaves

```
#/etc/module/hadoop-2.9.2/etc/hadoop/slaves
vim slaves
```

2、添加服务器节点、**千万注意不允许有任何空格**

```
#其中存放的都是datanode的节点
tarena01
tarena03
tarena04
```

3、将slaves文件复制到其他的服务器

```
[root@tarena101 hadoop]# xsync slaves
```

4、群起服务器

```
[root@tarena101 hadoop-2.9.2]# sbin/start-dfs.sh
```

5、启动resourceManager，千万记住，哪个服务器配置为resourceManager服务器就去哪个服务器启动

```
[hwhadoop@tarena103 hadoop-2.9.2]$ sbin/start-yarn.sh
```

6、启动tarena101:50070页面测试。

集群文件存储路径说明

上传一个大文件和一个小文件测试集群

```
[hwhadoop@tarena101 hadoop-2.9.2]$ bin/hdfs dfs -put wcinput/wc.input /
```

```
[hwhadoop@tarena101 hadoop-2.9.2]$ bin/hdfs dfs -put /opt/software/hadoop-2.9.2.tar.gz /
```

在tarena101:50070页面中可以看到任务信息，打开相关文件的信息，可以看到所有文件都有3个副本，并且大型的文件被分为了多块，每块最大为128M，如果下载将会把所有的快合并后完整下载

文件被放在哪里？

```
/opt/module/hadoop-2.9.2/data/tmp/dfs/data/current/BP-1721744875-192.168.80.11-  
1544527186020/current/finalized/subdir0/subdir0
```

```
-rw-rw-r--. 1 hwhadoop hwhadoop 30 12月 11 19:29 blk_1073741825  
-rw-rw-r--. 1 hwhadoop hwhadoop 11 12月 11 19:29 blk_1073741825_1001.meta  
-rw-rw-r--. 1 hwhadoop hwhadoop 134217728 12月 11 19:32 blk_1073741826  
-rw-rw-r--. 1 hwhadoop hwhadoop 1048583 12月 11 19:32 blk_1073741826_1002.meta  
-rw-rw-r--. 1 hwhadoop hwhadoop 134217728 12月 11 19:33 blk_1073741827  
-rw-rw-r--. 1 hwhadoop hwhadoop 1048583 12月 11 19:33 blk_1073741827_1003.meta  
-rw-rw-r--. 1 hwhadoop hwhadoop 98011993 12月 11 19:36 blk_1073741828  
-rw-rw-r--. 1 hwhadoop hwhadoop 765727 12月 11 19:36 blk_1073741828_1004.meta  
-rw-rw-r--. 1 hwhadoop hwhadoop 31961312 12月 11 19:48 blk_1073741829  
-rw-rw-r--. 1 hwhadoop hwhadoop 249707 12月 11 19:48 blk_1073741829_1005.met
```

可以cat命令来检查每个文件，也可以将块文件合并在一起

```
[hwhadoop@tarena101 subdir0]$ cat blk_1073741826 >> tmp.txt  
[hwhadoop@tarena101 subdir0]$ cat blk_1073741827 >> tmp.txt  
[hwhadoop@tarena101 subdir0]$ cat blk_1073741828 >> tmp.txt
```

合并完成后的tmp.txt文件其实就是一个tar.gz文件，可以正常解压缩。

集群启动停止方式总结

熟练使用各种单点启动和集群启动的方法，注意sbin/start-all.sh以及sbin/stop-all.sh的方法已经不再建议使用

Crontab定时任务

1、重启crond服务

```
service crond restart
```

2、定时任务的设置

- 基本语法

```
crontab [选项]
```

- 选项说明

选项	功能
-e	编辑crontab定时任务

选项	功能
-l	查询crontab任务
-r	删除当前用户所有的crontab任务

- 参数说明

crontab -e

会进入crontab编辑页面，会打开vim编辑你的工作

* * * * * 执行的任务

项目	含义	范围
第一个*	一小时当中的第几分钟	0-59
第二个*	一天当中的第几小时	0-23
第三个*	一个月当中的第几天	1-31
第四个*	一年当中的第几个月	1-12
第五个*	一周当中的星期几	0-7 (0和7都代表星期日)

特殊符号：

特殊符号 含义

*	代表任何时间，比如第一个*就代表一个小时中每分钟都执行一次
,	代表不连续的时间，比如"0 8,12,16 * * * 命令"，就代表在每天的8点0分，12点0分，16点0分都执行一次
-	代表连续的时间，比如"0 5 * * 1-6 命令"，就代表周一到周六的5点0分执行命令
*/n	代表每隔多长时间执行一次，比如"0/10 * * * * 命令"，代表每隔10分钟就执行一次命令

特定时间执行命令：

语法	含义
45 22 * * * 命令	在22点45分执行命令
0 17 * * 1 命令	每周1的17点0分执行命令
0 5 1,15 * * 命令	每月1号和15号的5点0分执行命令
40 4 * * 1-5 命令	每周1到周5的4点40分执行命令
*/10 4 * * * 命令	每天4点开始，每隔10分钟执行一次
0 0 1,15 * 1 命令	每月1号和15号，每周1的0点0分执行命令。 注意，星期几和几号最好不要同时出现，容易出现混乱

实例：

每隔1分钟，向/root/baoliniao.txt文件中添加一个"hello"

```
#进入编辑模式
[hwhadoop@tarena101 software]$ crontab -e
-----
#添加如下任务
*/1 * * * * /bin/echo "hello" >> /opt/software/baoliniao.txt
-----
#重启服务
[hwhadoop@tarena101 software]$ service crond restart
```

可以使用，监控文件的变化

tail -f baoliniao.txt

集群时间同步

这里开始要使用root用户进行操作，每隔10分钟同步一次时间。

- 检查ntp服务是否安装

```
[hwhadoop@tarena101 software]$ rpm -qa | grep ntp
-----
ntp-4.2.6p5-28.el7.centos.x86_64
fontpackages-filesystem-1.44-8.el7.noarch
ntpdate-4.2.6p5-28.el7.centos.x86_64
#看到以上3项打印说明ntp服务已经安装完毕
#否则请下载安装
[hwhadoop@tarena101 software]$ sudo yum install ntp
```

- 修改ntp配置文件

```
vim /etc/ntp.conf
```

修改让该网段的所有主机都能访问当前服务器，查看并同步时间

```
#将注释打开并修改
#restrict 192.168.1.0 mask 255.255.255.0 nomodify notrap
restrict 192.168.80.0 mask 255.255.255.0 nomodify notrap
```

修改集群在局域网中，不使用其他互联网上的时间

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
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

-----注释掉所有server-----

# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
#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
```

当节点丢失网络连接，依然可以采用本地时间作为时间服务器为集群中其他节点提供时间同步

```
#在末尾添加
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

- 修改/etc/sysconfig/ntp文件

进入文件，与硬件时钟同步

```
[root@tarena101 software]# vim /etc/sysconfig/ntp
```

添加如下内容

```
SYNC_HWCLOCK=yes
```

- 重新启动ntp服务

查看服务状态

```
[root@tarena101 software]# service ntpd status
```

启动

```
[root@tarena101 software]# service ntpd start
```

设置开机启动

```
chkconfig ntpd on  
#企业中一定打开，教学就不需要了，影响性能
```

- 其他机器的配置

注意：一定要是root用户

在其他机器配置每10分钟与时间服务器同步一次

```
[root@tarena102 hadoop-2.9.2]# crontab -e  
-----  
*/1 * * * * /usr/sbin/ntpdate tarena101
```

注意：如果tarena102中没有安装ntp，需要先安装一下。

在tarena102上设置时间进行测试

```
date #查看当前时间  
date -s #设置当前时间  
date -s '1999-2-21 13:30:21'
```

Hadoop源码编译

意义

官网下载的hadoop多数是32位的，需要将源码编译成64位的

说明

准备工作

1、centos联网

配置centos能连接外网，Linux虚拟机ping www.baidu.com是畅通的

注意：采用root角色编译，减少文件权限出错的问题

2、jar包准备，JDK8，maven，ant，protobuf

hadoop-2.9.2-src.tar.gz

jdk-8u181-linux-x64.tar.gz

apache-ant-1.9.9-bin.tar.gz

apache-maven-3.0.5-bin.tar.gz

protobuf-2.5.0.tar.gz

3、jar包的安装

注意：所有操作都要在root用户下

1、JDK解压，配置环境变量JAVA_HOME和PATH，验证java -version

2、Maven解压，配置MAVEN_HOME和PATH

#与jdk的安装没有什么区别

#完成后测试maven

mvn -version

配置maven

[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
  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>
```

3、ant解压、配置ANT_HOME和PATH

#与jdk的安装没有什么区别

#完成后测试maven

ant -version

4、安装glibc-headers和g++命令

[root@code apache-ant-1.10.5]# yum install glibc-headers

[root@code apache-ant-1.10.5]# yum install gcc-c++

5、安装make和cmake

[root@code apache-ant-1.10.5]# yum install make

[root@code apache-ant-1.10.5]# yum install cmake

6、解压protobuf，进入到解压后protobuf主目录，然后执行相应命令

#解压

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

#进入解压目录执行命令

```
[root@code protobuf-2.5.0]# ./configure
[root@code protobuf-2.5.0]# make
[root@code protobuf-2.5.0]# make check
[root@code protobuf-2.5.0]# make install
[root@code protobuf-2.5.0]# ldconfig
#配置环境变量
[root@code protobuf-2.5.0]# vim /etc/profile
-----
#LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/opt/module/protobuf-2.5.0
export PATH=$PATH:$LD_LIBRARY_PATH
-----
[root@code protobuf-2.5.0]# source /etc/profile
```

#测试

```
[root@code protobuf-2.5.0]# protoc --version
-----
libprotoc 2.5.0
```

7、安装openssl库

```
[root@code protobuf-2.5.0]# yum install openssl-devel
```

8、安装ncurses-devel库

```
[root@code protobuf-2.5.0]# yum install ncurses-devel
```

编译源码

1，解压源码到opt目录

```
[root@code software]# tar -zxvf hadoop-2.9.2-src.tar.gz -C /opt/
```

2、进入hadoop源码主目录

```
[root@code opt]# cd hadoop-2.9.2-src/
```

3、通过maven执行编译命令

```
[root@code hadoop-2.9.2-src]# mvn package -Pdist,native -DskipTests -Dtar
```

HDFS

HDFS概述

HDFS产生和背景

1、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，但是不方便管理和维护，迫切需要一个系统来维护和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中的一种。

2、HDFS (Hadoop Distributed File System) 他是一个文件系统，用于存储文件，通过目录树来定位文件，其次，他是分布式的，有很多服务器联合起来实现其功能，集群中的服务器有各自的角色。

3、HDFS的适用场景，适合一次写入，多次读出的场景，且不支持文件修改，适合用来做数据分析，并不适合用来做网盘应用。

优缺点

优点：

1.2.1 优点

1) 高容错性

(1) 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。



(2) 某一个副本丢失以后，它可以自动恢复。



2) 适合处理大数据

(1) 数据规模：能够处理数据规模达到GB、TB、甚至PB级别的数据；

(2) 文件规模：能够处理百万规模以上的文件数量，数量相当之大。

3) 可构建在廉价机器上，通过多副本机制，提高可靠性。

缺点：

1.2.2 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。

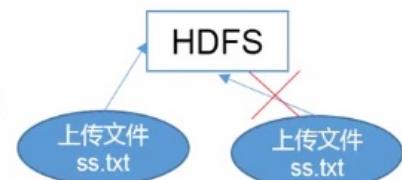
(1) 存储大量小文件的话，它会占用NameNode大量的内存来存储文件目录和块信息。这样是不可取的，因为NameNode的内存总是有限的；

(2) 小文件存储的寻址时间会超过读取时间，它违反了HDFS的设计目标。

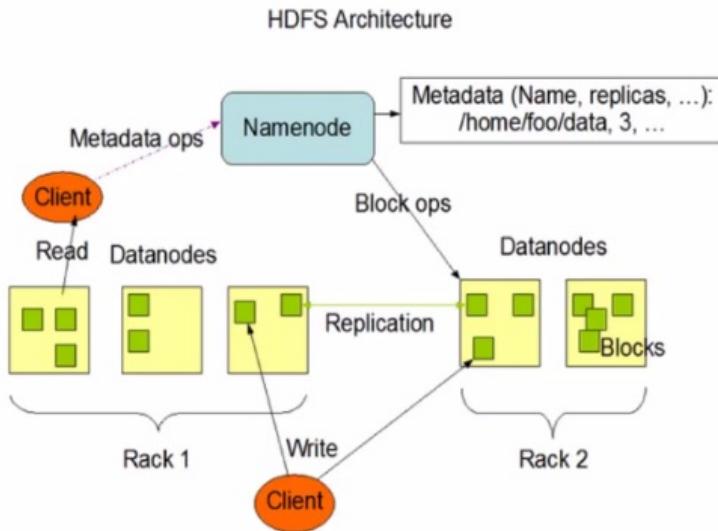
3) 不支持并发写入、文件随机修改。

(1) 一个文件只能有一个写，不允许多个线程同时写；

(2) 仅支持数据append (追加)，不支持文件的随机修改。



组成架构



1) NameNode (nn) : 就是Master , 它是一个主管、管理者。

- (1) 管理HDFS的名称空间；
- (2) 配置副本策略；
- (3) 管理数据块 (Block) 映射信息；
- (4) 处理客户端读写请求。

2) DataNode : 就是Slave。 NameNode 下达命令， DataNode 执行实际的操作。

- (1) 存储实际的数据块；
- (2) 执行数据块的读/写操作。

3) Client : 就是客户端。

- (1) 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；
- (2) 与NameNode交互 , 获取文件的位置信息；
- (3) 与DataNode交互 , 读取或者写入数据；
- (4) Client提供一些命令来管理HDFS , 比如NameNode格式化；
- (5) Client可以通过一些命令来访问HDFS , 比如对HDFS增删查改操作；

4) Secondary NameNode : 并非NameNode的热备。当NameNode挂掉的时候 , 它并不能马上替换NameNode并提供服务。

- (1) 辅助NameNode , 分担其工作量 , 比如定期合并Fsimage和Edits , 并推送给NameNode ；
- (2) 在紧急情况下 , 可辅助恢复NameNode。

块的大小设置

HDFS 中的文件在物理上是分块存储 (Block) , 块的大小可以通过配置参数 (dfs.blocksize) 来规定 , 默认大小在 Hadoop2.x 版本中是 128M , 老版本中是 64M 。

2 如果寻址时间为 10ms , 即查找到目标 block 的时间为 10ms 。

3 寻址时间为传输时间的 1% 时 , 则为最佳状态。
因此 , 传输时间
 $=10\text{ms}/0.01=1000\text{ms}=1\text{s}$

4 而目前磁盘的传输速率普遍为 100MB/s 。

1 集群中的 block

block1

block2

... ...

blockn

5 block 大小
 $=1\text{s} \times 100\text{MB/s} = 100\text{MB}$

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) 如果块设置的 **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**。导致程序在处理这块数据时 , 会非常慢。

总结 : HDFS 块的大小设置主要取决于磁盘传输速率。

Shell 命令 (开发重点)

1、基本语法

bin/hadoop fs 具体命令

bin/hdfs dfs 具体命令

2、命令大全

群启服务器集群

```
#看到hdfs dfs下的所有命令
[hwhadoop@tarena101 hadoop-2.9.2]$ hdfs dfs
#看到hadoop fs下的所有命令
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs
```

3、常用命令实践

-help 输出这个命令的参数

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -help rm
-----
-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ... :
  Delete all files that match the specified file pattern. Equivalent to the Unix
  command "rm <src>"

-f      If the file does not exist, do not display a diagnostic message or
       modify the exit status to reflect an error.
-[rR]   Recursively deletes directories.
-skipTrash option bypasses trash, if enabled, and immediately deletes <src>.
-safely  option requires safety confirmation, if enabled, requires
       confirmation before deleting large directory with more than
       <hadoop.shell.delete.limit.num.files> files. Delay is expected when
       walking over large directory recursively to count the number of
       files to be deleted before the confirmation.
```

-ls显示目录信息

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -ls /
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -ls -R /
-----
Found 3 items
-rw-r--r--  3 hwhadoop supergroup  31961312 2018-12-11 19:48 /xftp-6.0.0103p.exe
-rw-r--r--  3 hwhadoop supergroup  366447449 2018-12-11 19:36 /hadoop-2.9.2.tar.gz
-rw-r--r--  3 hwhadoop supergroup         30 2018-12-11 19:29 /wc.input
```

-mkdir在HDFS上创建目录

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -mkdir -p /zg/cq/yb/
```

-moveFromLocal从本地剪切文件到HDFS

#在本地创建一个文件

```
[hwhadoop@tarena101 hadoop-2.9.2]$ vim konggang.txt
```

#将文件剪切到/zg/cq/yb

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -moveFromLocal ./konggang.txt /zg/cq/yb
```

-appendToFile, 追加一个文件到已经存在的文件的末尾

#在本地创建一个文件

```
[hwhadoop@tarena101 hadoop-2.9.2]$ vim jiangbeiairport.txt
```

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -appendToFile ./jiangbeiairport.txt
```

/zg/cq/yb/konggang.txt

#-cat显示文件内容

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -cat /zg/cq/yb/konggang.txt
```

konggang

空港新区

江北国际机场

-chgrp、-chmod、-chown跟LINUX文件系统中的作用是一样的。修改文件的权限

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -chgrp hwhadoop /zg/cq/yb/konggang.txt
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -chgrp -R hwhadoop /java/
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -chmod 777 /java/net/okk.txt
#也可以Linux那样 chown hwhadoop:hwhadoop /java/ -R
```

-copyFromLocal从本地复制文件到HDFS

#与moveFromLocal用法一样，只不过一个是剪切另一个是拷贝

-copyToLocal从HDFS拷贝到本地

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -copyToLocal /zg/cq/yb/konggang.txt ./
```

-cp从HDFS一个路径拷贝到另一个路径

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -cp /zg/cq/yb/konggang.txt /zg/cq/
```

-mv从HDFS一个路径剪切到另一个路径

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -mv /konggang.txt /zg/
```

-get

#效果同于copyToLocal

-getmerge将多个文件合并下载

#第一个是元数据，空格后是要下载到的地址

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -getmerge /zg/* ./yubei
```

-put

#等同于copyFromLocal

-tail显示一个文件的末尾

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -tail /zg/konggang.txt
```

#动态监控一个文件

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -tail -f /zg/konggang.txt
```

-rm删除文件或文件夹

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -rm /zg/jiangbeiairport.txt
```

#递归删除文件夹

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -rm -r /zg/jiangbeiairport.txt
```

-rmdir删除空目录

-du统计文件夹的大小信息

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -du /zg
```

41 /zg/cq

41 /zg/konggang.txt

#显示优化格式

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -du -h /
```

30.5 M /xftp-6.0.0103p.exe

349.5 M /hadoop-2.9.2.tar.gz

30 /wc.input

82 /zg

#看总大小

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hadoop fs -du -h -s /
```

380.0 M /

-setrep设置HDFS文件的副本数量

设置之前进入每一个服务器的:/opt/module/hadoop-2.9.2/data/tmp/dfs/data/current/BP-1721744875-192.168.80.11-1544527186020/current/finalized/subdir0/subdir0路径下查看相关副本信息是否存在。

```
[hwhadoop@tarena101 subdir0]$ hadoop fs -setrep 2 /zg/cq/yb/konggang.txt
```

设置后在进入上述路径查看副本变化。

#如果副本数设置为10，当有新的服务器节点创建时会自动向其添加一个副本，知道10个副本添加完成为止

```
[hwhadoop@tarena101 subdir0]$ hadoop fs -setrep 10 /zg/cq/yb/konggang.txt
```

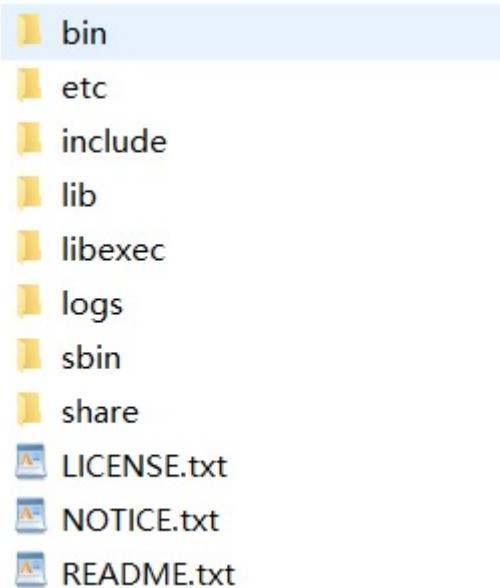
客户端环境准备

HDFS客户端环境的准备

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，如:D:/developer/hadoop-2.9.2

编译win10的hadoop

- 到hadoop官网下载hadoop的最新版本压缩包，我下载的是2.9.0版本的，链接如下：<https://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/stable/>，解压过后目录如下所示：



- 安装配置hadoop运行环境
- 下载win10下的依赖包，解压过后{链接：<https://pan.baidu.com/s/1ahSpPoBSLsouUNkKHQHSSg> 密码：hre7}将bin中我所框选的文件复制到hadoop的bin文件中，为解决依赖性问题，之后要将hadoop.dll复制到C:/WINDOW/SYSTEM32下面

└ container-executor	2017/11/14 7:28	文件
└ hadoop	2017/11/14 7:28	文件
└ hadoop.cmd	2017/11/14 7:28	Window
└ hadoop.dll	2015/8/10 14:07	应用程序
└ hadoop.exp	2015/8/10 14:07	Exports
└ hadoop.lib	2015/8/10 14:07	Object F
└ hadoop.pdb	2015/8/10 14:07	Program
└ hdfs	2017/11/14 7:28	文件
└ hdfs.cmd	2017/11/14 7:28	Window
└ libwinutils.lib	2015/8/10 14:07	Object F
└ mapred	2017/11/14 7:28	文件
└ mapred.cmd	2017/11/14 7:28	Window
└ rcc	2017/11/14 7:28	文件
└ test-container-executor	2017/11/14 7:28	文件
└ winutils.exe	2015/8/10 14:07	应用程序
└ winutils.pdb	2015/8/10 14:07	Program

- 配置环境变量
- 打开命令行输入 version,返回版本信息，则配置成功
- 创建maven项目，在pom.xml中导入相关依赖

```

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.logging.log4j/log4j-core --&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.apache.logging.log4j&lt;/groupId&gt;
    &lt;artifactId&gt;log4j-core&lt;/artifactId&gt;
    &lt;version&gt;2.8.2&lt;/version&gt;
&lt;/dependency&gt;
<!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-common --&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.apache.hadoop&lt;/groupId&gt;
    &lt;artifactId&gt;hadoop-common&lt;/artifactId&gt;
    &lt;version&gt;2.9.2&lt;/version&gt;
&lt;/dependency&gt;
<!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-hdfs --&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.apache.hadoop&lt;/groupId&gt;
    &lt;artifactId&gt;hadoop-hdfs&lt;/artifactId&gt;
    &lt;version&gt;2.9.2&lt;/version&gt;
&lt;/dependency&gt;
<!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client --&gt;
&lt;dependency&gt;
    &lt;groupId&gt;org.apache.hadoop&lt;/groupId&gt;
    &lt;artifactId&gt;hadoop-client&lt;/artifactId&gt;
</pre>

```

```
<version>2.9.2</version>
</dependency>
```

- 在类路径下添加log4j.properties文件

```
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
```

- 创建一个类，准备写代码

客户端环境测试

```
public class HDFSClient {
    public static void main(String[] args) {

        Configuration conf = new Configuration();
        //conf.set("fs.defaultFS","hdfs://tarena101:9000");
        try {
            //FileSystem fs = FileSystem.get(conf);
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"),conf,"hwhadoop");
            fs.mkdirs(new Path("/java/servlet"));
            fs.close();
            System.out.println("over");
        } catch (IOException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

坑

1、出现如下错误，表示当前是Administrator用户却想要访问hwhadoop用户，这是肯定不可以的，所以在请求服务器的时候应该设置用户。

```
org.apache.hadoop.security.AccessControlException: Permission denied: user=Administrator,
access=WRITE, inode="/":hwhadoop:supergroup:drwxr-xr-x
```

2、No FileSystem for scheme: hdfs如果出现这个错误，请检查maven下载是否出现问题。删除仓库中的对应文件夹重新下载。

API操作

文件上传_案例

```
public class HDFSFileUpLoad {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        //1,获取FileSystem对象
        try {
```

```

        FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf, "hwhadoop");
        //2,执行上传逻辑
        fs.copyFromLocalFile(new Path("e:/a.txt"), new Path("/java/servlet/a.txt"));
        //3,记得释放资源
        fs.close();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}

}

```

参数优先级

如果在类路径下创建如下hdfs-site.xml，则优先级高于集群中的hdfs-site.xml的配置

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
    <property>
        <name>dfs.replication</name>
        <!--设置副本数量-->
        <value>2</value>
    </property>
</configuration>

```

如果在代码中设置conf.set("dfs.replication","1");配置将是最高优先级。

```

public class HDFSfileUpLoad {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        conf.set("dfs.replication", "1");
        //1,获取FileSystem对象
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf, "hwhadoop");
            //2,执行上传逻辑
            fs.copyFromLocalFile(new Path("e:/a.txt"), new Path("/java/servlet/a2.txt"));
            //3,记得释放资源
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
}

```

文件下载

```

public class HDFSfileDownLoad {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {

```

```

        FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
        //fs.copyToLocalFile(new Path("/java/servlet/a.txt"),new Path("h:/acopy.txt"));
        //第一个参数boolean值, 为true时表示剪切
        //最后一个参数为true时表示不打开crc校验, 默认为false, 在下载后会看到crc文件
        fs.copyToLocalFile(true,new Path("/java/servlet/a.txt"),new
Path("h:/acopy.txt"),true);
        fs.close();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}
}

```

文件夹删除

```

public class HDFSFileDelete {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            //第二个参数boolean值, 为true时代表递归删除, false则不递归删除
            fs.delete(new Path("/java/servlet"),true);
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
}

```

修改文件的名称

```

public class HDFSFileReName {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            fs.rename(new Path("/java/a.txt"),new Path("/java/b.txt"));
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
}

```

查看文件的详情

```
public class HDFSFileSelect {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            RemoteIterator<LocatedFileStatus> lr = fs.listFiles(new Path("/"), true);
            while(lr.hasNext()){
                LocatedFileStatus next = lr.next();
                //显示文件名称, 权限, 长度, 块信息
                System.out.println(next.getPath().getName());
                System.out.println(next.getPermission());
                System.out.println(next.getLength());
                BlockLocation[] blockLocations = next.getBlockLocations();
                for (BlockLocation b: blockLocations) {
                    String[] hosts = b.getHosts();
                    for (int i = 0; i < hosts.length; i++) {
                        System.out.println(hosts[i]);
                    }
                }
                System.out.println("-----");
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
}
```

判断是文件还是文件夹

```
public class HDFSFileStatus {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            FileStatus[] filestatuses = fs.listStatus(new Path("/java/servlet/"));
            for (int i = 0; i < filestatuses.length; i++) {
                if(filestatuses[i].isFile()){
                    System.out.println("file:"+filestatuses[i].getPath().getName());
                }else{
                    System.out.println("dir:"+filestatuses[i].getPath().getName());
                }
            }
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
}
```

```
    }
}
```

文件IO流上传_案例

```
public class HDFSFileupLoadByIO {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            //1,获取输入流
            InputStream in = new FileInputStream(new File("e:/up.txt"));
            //2,获取输出流
            FSDataOutputStream fsDataOutputStream = fs.create(new
Path("/java/servlet/up.txt"));
            //3,双流对通
            IOUtils.copyBytes(in,fsDataOutputStream,conf);
            //4,关闭流
            IOUtils.closeStream(fsDataOutputStream);
            IOUtils.closeStream(in);
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
}
```

文件IO流下载操作_案例

```
public class HDFSFileDownLoadByIO {
    public static void main(String[] args) {
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            //1,获取输入流
            FSDataInputStream open = fs.open(new Path("/java/servlet/a2.txt"));
            //2,获取输出流
            FileOutputStream out = new FileOutputStream("e:/copya2.txt");
            //3,双流对通
            IOUtils.copyBytes(open,out,conf);
            //4,关闭流
            IOUtils.closeStream(out);
            IOUtils.closeStream(open);
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }
}
```

定位读取文件_案例

```
public class HDFSFileDownLoadBySeek {
    public static void main(String[] args) {
        download128();
        download4Seek();
    }

    //下载文件前半部分
    public static void download128(){
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            //1,获取输入流
            FSDataInputStream open = fs.open(new Path("/java/jdk-8u181-linux-x64.tar.gz"));
            //2,获取输出流
            FileOutputStream out = new FileOutputStream("e:/jdk-8u181-linux-
x64.tar.gz.part1");
            //3,双流对通
            byte[] buf = new byte[1024];
            for (int i = 0; i < 1024*128; i++) {
                open.read(buf);
                out.write(buf);
            }
            out.flush();
            //4,关闭流
            IOUtils.closeStream(out);
            IOUtils.closeStream(open);
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    //下载文件后半部分
    public static void download4Seek(){
        Configuration conf = new Configuration();
        try {
            FileSystem fs = FileSystem.get(new URI("hdfs://tarena101:9000"), conf,
"hw.hadoop");
            //1,获取输入流
            FSDataInputStream open = fs.open(new Path("/java/jdk-8u181-linux-x64.tar.gz"));
            //2,获取输出流
            FileOutputStream out = new FileOutputStream("e:/jdk-8u181-linux-
x64.tar.gz.part2");
            //3,双流对通
            open.seek(1024*1024*128);
            IOUtils.copyBytes(open,out,conf);
            //4,关闭流
            IOUtils.closeStream(out);
            IOUtils.closeStream(open);
            fs.close();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
```

```

        e.printStackTrace();
    } catch (URISyntaxException e) {
        e.printStackTrace();
    }
}
}

```

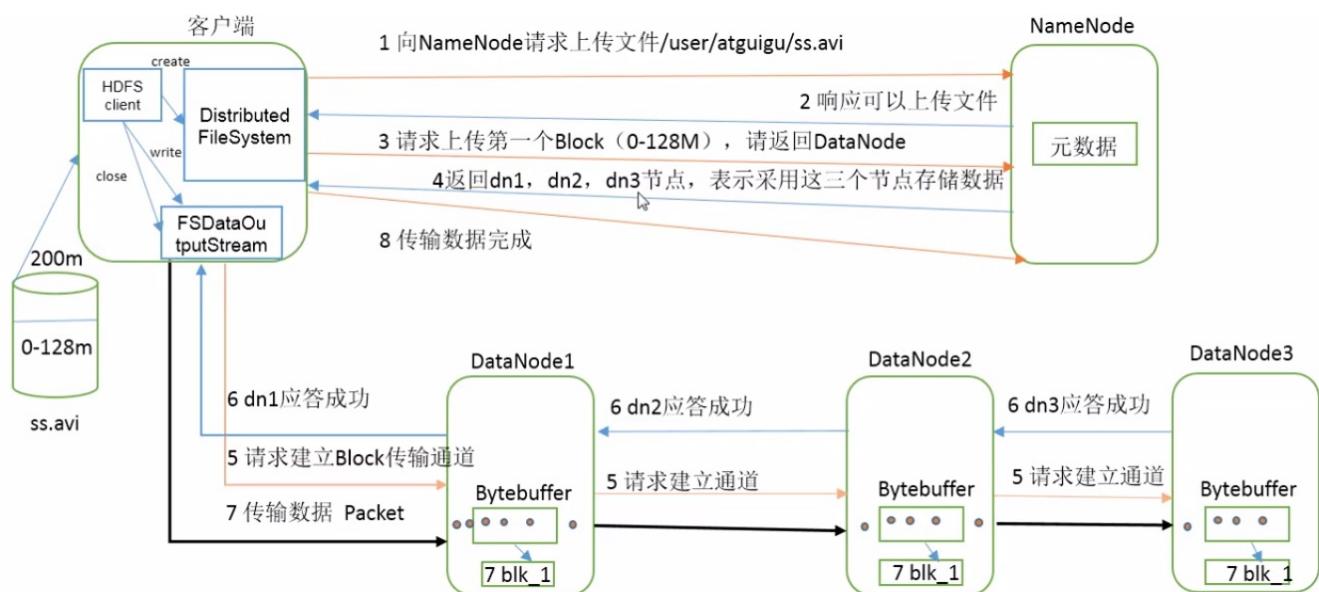
在CMD中进入两个part所在的目录输入命令

```
type hadoop-2.9.2.tar.gz.part1 >> hadoop-2.9.2.tar.gz.part2
```

合并后修改文件名，测试文件完整性。

HDFS的数据流（面试重点）

HDFS写数据流程



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接受数据，

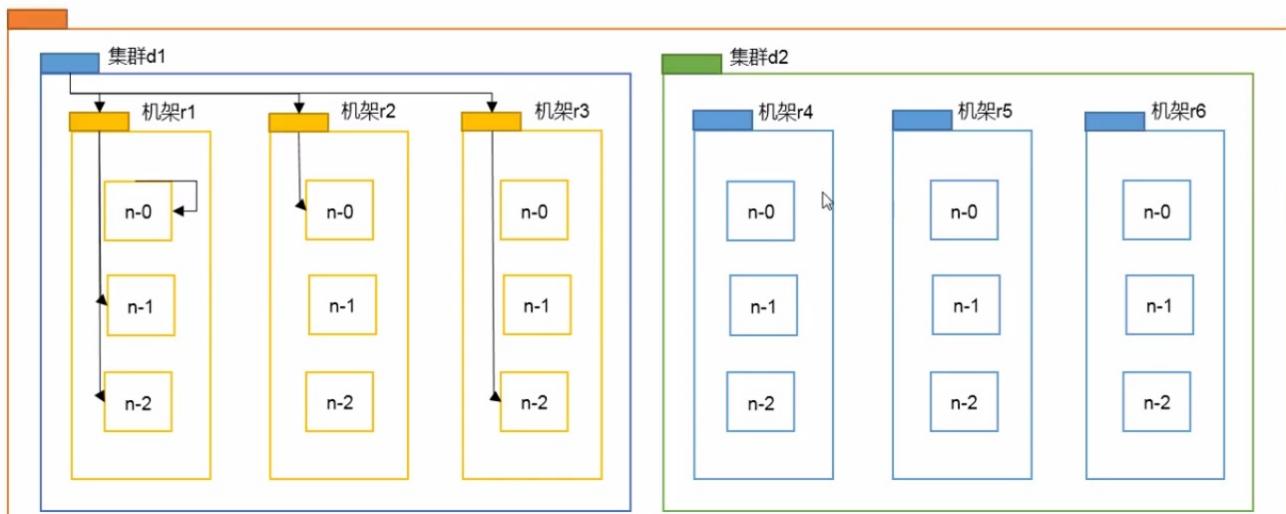
节点距离：两个节点到达最近的共同祖先的距离总和

Distance(/d1/r1/n0, /d1/r1/n0)=0 (同一节点上的进程)

Distance(/d1/r1/n1, /d1/r1/n2)=2 (同一机架上的不同节点)

Distance(/d1/r2/n0, /d1/r3/n2)=4 (同一数据中心不同机架上的节点)

Distance(/d1/r2/n1, /d2/r4/n1)=6 (不同数据中心的节点)



机架感知-副本存储节点选择

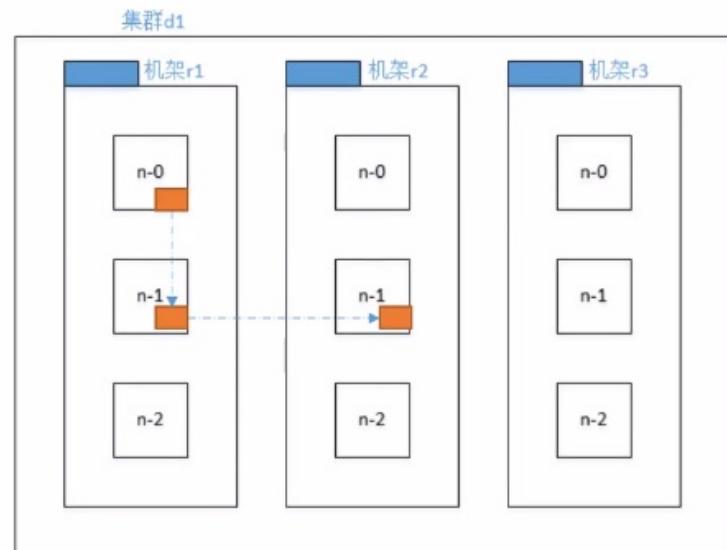
官方说明：

http://hadoop.apache.org/docs/r2.9.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Data_Replication

第一个副本在Client所处的节点上。
如果客户端在集群外，随机选一个。

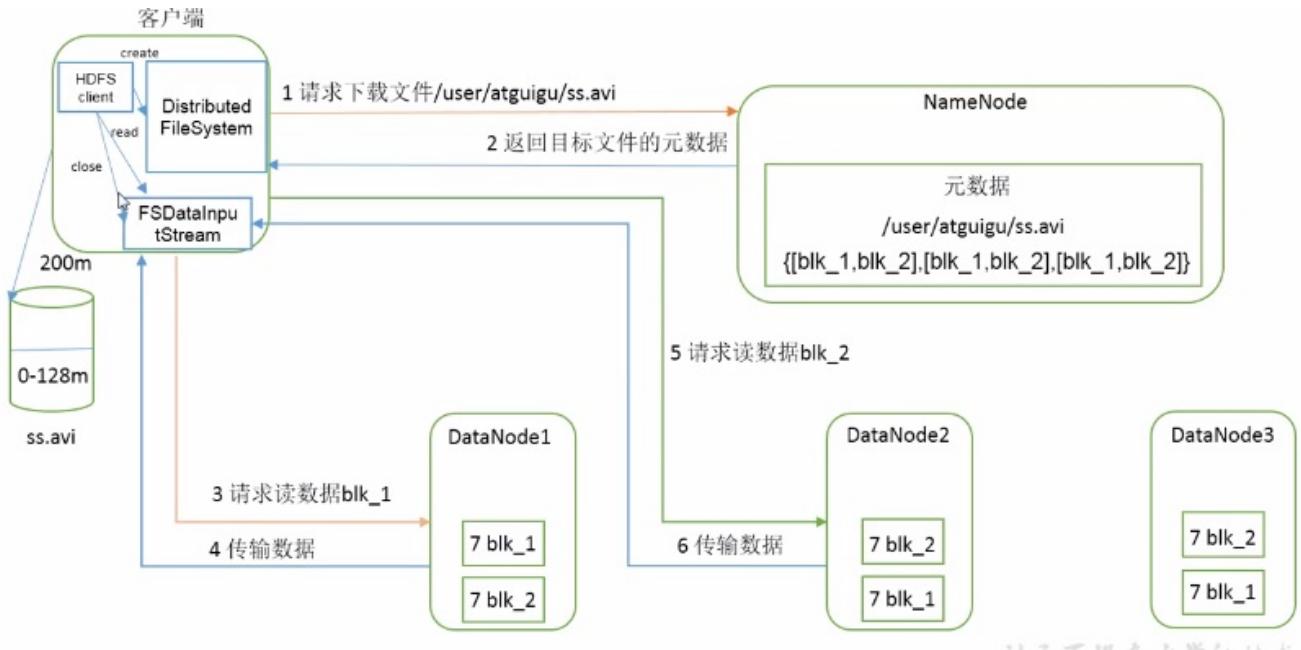
第二个副本和第一个副本位于相
同机架，随机节点。

第三个副本位于不同机架，随机节点。



让天下没有难学的技术

读数据流程（面试重点）



NameNode和2NameNode工作机制（面试重点）

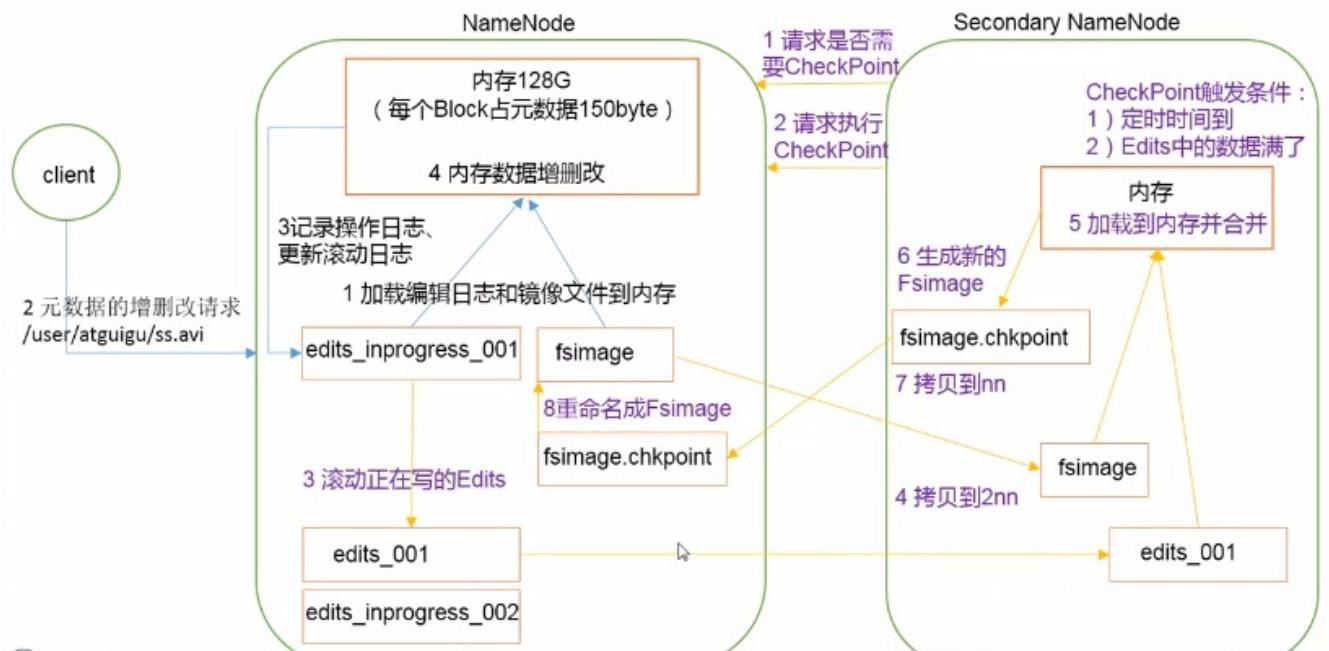
工作机制

NameNode 中元数据存在哪里？

如果存储在 NameNode 节点的磁盘中，因为经常需要进行随机访问，还要响应客户请求，必然效率很低，因此，元素数据需要存放在内存中，但如果只存在于内存中，一旦断电，元数据丢失。因此产生了在磁盘中备份元数据的 FsImage

这样新的问题又来了，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，但如果不行更新就会发生一致性问题，一旦 NameNode 节点断电，就会数据丢失，因此，引入 Edit 文件（只进行追加操作，效率很低）每当元数据有更新或者追加元数据时，修改内存中的元数据并追加到 Edit 中，这样，一旦 NameNode 断电，可以通过 FsImage 和 Edit 的合并，合并成元数据

但是长时间添加数据到 Edit 中，会导致文件数据过大，效率降低，而且一旦断电，恢复元数据的时间过长，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，因此引入一个新节点 SecondaryNameNode 专门用户 FsImage 和 Edits 的合并



CheckPoint时间设置

通常情况下，SecondaryNameNode每隔一个小时执行一次

hdfs-default.xml

```
<property>
    <name>dfs.namenode.checkpoint.period</name>
    <value>3600</value>
    <descript>1小时</descript>
</property>
```

一分钟检查一次操作次数，当操作次数达到100万时，SecondaryNameNode必须执行一次

```
<property>
    <name>dfs.namenode.checkpoint.txns</name>
    <value>1000000</value>
    <descript>操作次数</descript>
</property>
<property>
    <name>dfs.namenode.checkpoint.check.period</name>
    <value>60</value>
    <descript>1分钟问一次</descript>
</property>
```

NN故障处理_案例

NameNode故障后，可以采用如下两种方法恢复数据。

1、将SecondaryNameNode中数据拷贝到NameNode存储数据的目录：

- kill -9 NameNode 进程
- 删除NameNode中的数据 (/opt/module/hadoop-2.9.2/data/tmp/dfs/name)

```
[hwhadoop@tarena101 hadoop-2.9.2]$ rm -rf name/*
```

- 拷贝SecondaryNameNode中数据到原NameNode存储数据的目录

```
#进入 (/opt/module/hadoop-2.9.2/data/tmp/dfs/name)
scp -r hwhadoop@tarena103:/opt/module/hadoop-2.9.2/data/tmp/dfs/namesecondary/current ./
```

- 重启NameNode

```
sbin/hadoop-daemon.sh start namenode
```

- 查看tarena101:50070页面查看数据，有延迟稍微等一下

2、使用 -importCheckpoint选项启动NameNode守护进程，从而将SecondaryNameNode数据拷贝到NameNode目录中

- 修改hdfs-site.xml中的
- kill -9 NameNode进程号
- 删除NameNode存储的数据 (/opt/module/hadoop-2.9.2/data/tmp/dfs/name)
- 删除NameNode中的数据 (/opt/module/hadoop-2.9.2/data/tmp/dfs/name)

```
[hwhadoop@tarena101 hadoop-2.9.2]$ rm -rf name/*
```

- 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

```
[hwhadoop@tarena101 dfs]$ scp -r hwhadoop@tarena103:/opt/module/hadoop-
2.9.2/data/tmp/dfs/namesecondary ./
```

```
[hwhadoop@tarena101 dfs]$ ll
```

```
总用量 0
drwx----- 3 hwhadoop hwhadoop 40 12月 12 19:52 data
drwxrwxr-x 2 hwhadoop hwhadoop 6 12月 13 21:16 name
drwxrwxr-x 3 hwhadoop hwhadoop 40 12月 13 21:19 namesecondary
```

```
[hwhadoop@tarena101 namesecondary]$ rm -r in_use.lock
```

- 导入检查点数据，等待一会就可以ctrl+c推出

```
[hwhadoop@tarena101 hadoop-2.9.2]$ bin/hdfs namenode -importCheckpoint
```

- 重启NameNode

```
sbin/hadoop-daemon.sh start namenode
```

- 查看tarena101:50070页面查看数据，有延迟稍微等一下

安全模式

1、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一直运行在安全模式，即NameNode的文件系统对于客户端来说是只读的。

2、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。

3、安全模式退出判断

如果满足“最小副本条件”，NameNode会在30秒钟之后就退出安全模式。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）。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。

集群安全模式_案例

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

基本语法：

```
#查看安全模式状态  
bin/hdfs dfsadmin -safemode get  
#进入安全模式状态  
bin/hdfs dfsadmin -safemode enter  
#离开安全模式状态  
bin/hdfs dfsadmin -safemode leave  
#等待安全模式状态  
bin/hdfs dfsadmin -safemode wait
```

案例：

等待安全模式退出后，执行后续命令

- 查看当前安全模式状态

```
[hwhadoop@tarena101 hadoop-2.9.2]$ bin/hdfs dfsadmin -safemode get  
-----  
Safe mode is OFF
```

- 进入安全模式

```
[hwhadoop@tarena101 hadoop-2.9.2]$ bin/hdfs dfsadmin -safemode enter  
-----  
Safe mode is ON
```

- 添加safemode.sh脚本文件

```
[hwhadoop@tarena101 hadoop-2.9.2]$ vim safemode.sh  
-----  
#!/bin/bash  
hdfs dfsadmin -safemode wait  
#注意记得创建readme.txt  
hdfs dfs -put /opt/module/hadoop-2.9.2/readme.txt /
```

- 执行脚本文件将会看到程序挂起

```
[hwhadoop@tarena101 hadoop-2.9.2]$ bash safemode.sh
```

- 克隆一个窗口，在新的窗口中退出安全模式，挂起的程序将会继续执行脚本命令，文件被上传

```
[hwhadoop@tarena101 hadoop-2.9.2]$ hdfs dfsadmin -safemode leave
```

HDFS_NN多目录配置案例

NameNode的本地目录可以配置成多个，且每个目录存放内容相同，增加可靠性

具体配置：

- 1, 在hdfs-site.xml中添加：(千万别弄错了文件)

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

- 2, 向集群中其他主机同步

```
#如果同步后查看文件提示有替换文件，按D删除重新操作一次即可
xsync etc/hadoop/
```

- 3, 删除hadoop-2.9.2下的data和logs文件夹
- 4, 格式化namenode

```
bin/hdfs namenode -format
```

- 5, 群起服务器

```
sbin/start-dfs.sh
```

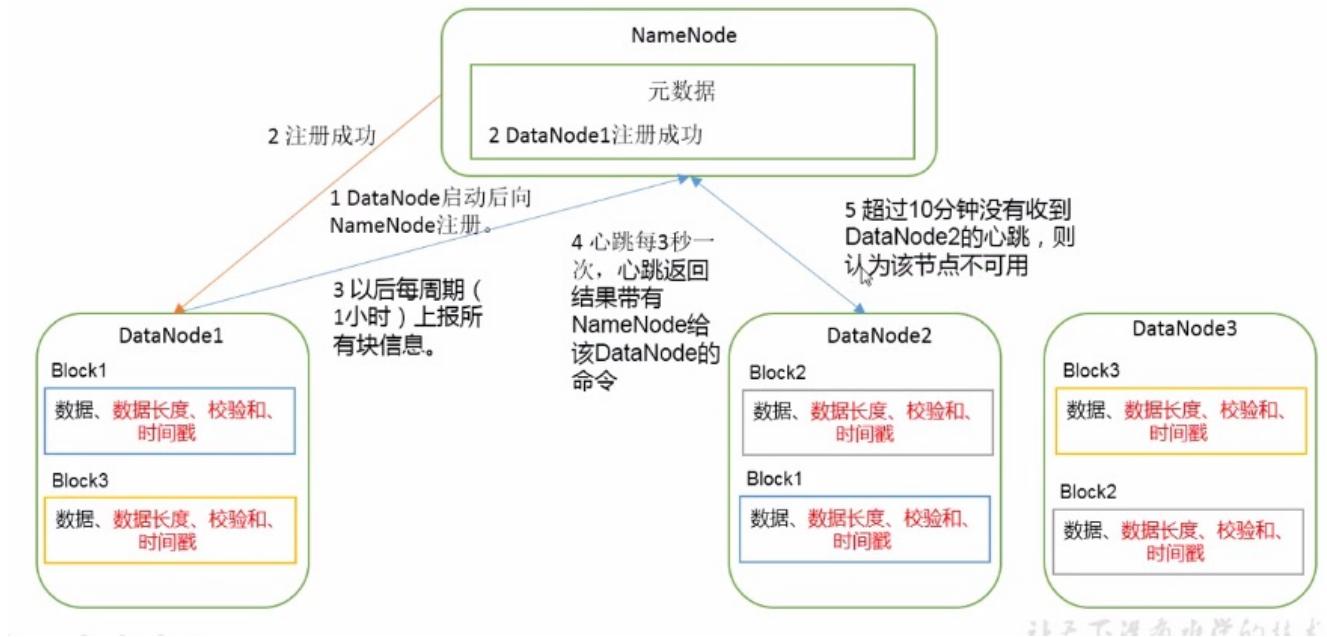
- 6, 上传文件到集群

```
hadoop -fs -put readme.txt /java/servlet/
```

- 7, 进入data下查看name1和name2是否相同

```
cd /opt/module/hadoop-2.9.2/data/tmp/dfs/name1
cd /opt/module/hadoop-2.9.2/data/tmp/dfs/name2
```

DataNode工作机制（面试重点）



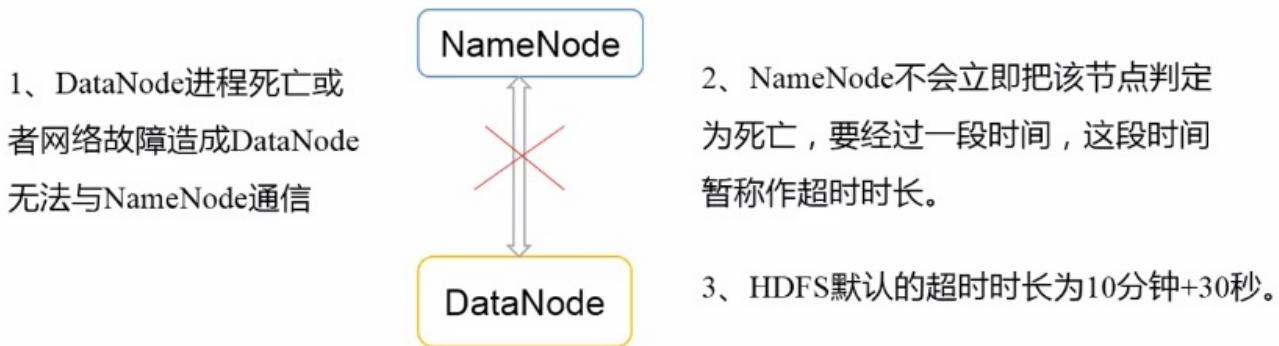
数据完整性

如果电脑磁盘里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) , 但是存储数据的磁盘坏了, 一直显示绿灯? 同理datanode节点上的数据损坏了, 却没有发现, 也很危险, 如何解决

- 当datanode读取block的时候, 他会计算checksum。
- 如果计算后checksum, 与block创建时值不一样, 则认为block损坏。
- client读取其他datanode上的block。
- datanode在其文件后创建周期验证checksum



掉线时限参数设置



4、如果定义超时时间为TimeOut，则超时时长的计算公式为：

`TimeOut = 2 * dfs.namenode.heartbeat.recheck-interval + 10 * dfs.heartbeat.interval。`

而默认的`dfs.namenode.heartbeat.recheck-interval`大小为5分钟，`dfs.heartbeat.interval`默认为3秒。

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>

```

服役新节点_案例

需求：

随着数据量越来越大，原有的数据节点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。

环境准备：

- 在新建一个hadoop104主机
- 修改主机名和网络配置
- **修改主机名后一定要重启**
- **删除原来HDFS文件系统留存的文件（/opt/module/hadoop-2.9.2/data 和 logs 很重要）**
- source一下配置文件

`source /etc/profile`

- 在hosts文件中添加新的主机hosts

```
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1          localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.40.11 tarena101
192.168.40.12 tarena102
192.168.40.13 tarena103
192.168.40.14 tarena104
-----
#记得同步到集群
xsync /etc/hosts
```

- 在新节点上启动datanode

```
[root@tarena104 hadoop-2.9.2]# sbin/hadoop-daemon.sh start datanode
```

- 打开tarena101:50070查看datanode页签，看到tarena104节点加入到集群中。
- 使用新创建的节点向集群的HDFS中创建文件夹和文件。

添加白名单_案例

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

步骤：

- 在NameNode的/opt/module/hadoop-2.9.2/etc/hadoop目录下创建dfs.hosts文件（文件名可以随便）
- 在文件中添加被允许的主机名

```
#不允许空格和多余的换行
```

```
tarena101
tarena102
tarena103
```

- 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-2.9.2/etc/hadoop/dfs.hosts</value>
</property>
```

- 在slaves文件中添加节点
- 配置文件分发

```
xsync hdfs-site.xml
```

- 刷新NameNode

```
[hwhadoop@tarena101 hadoop]$ hdfs dfsadmin -refreshNodes
```

- 有的刷新并不管用，需要重启集群
- 刷新Yarn

```
yarn rmadmin -refreshNodes
```

- 均衡负载

```
sbin/start-balancer.sh
```

黑名单退役_案例

在黑名单上面的主机都会被强制退出

- 在/opt/module/hadoop-2.9.2/etc/hadoop/下创建dfs.hosts.exclude文件

- 在文件中添加黑名单节点名
- 去掉slaves中被加入没名单的节点
- 编辑hdfs-site.xml中添加

```
<property>
  <name>dfs.hosts.exclude</name>
  <value>/opt/module/hadoop-2.9.2/etc/hadoop/dfs.hosts.exclude</value>
</property>
```

- 刷新

```
[hwhadoop@tarena101 hadoop]$ hdfs dfsadmin -refreshNodes
[hwhadoop@tarena101 hadoop]$ yarn rmadmin -refreshNodes
```

- 黑名单中节点在退役时会耗时较长时间