数据一 课堂笔记

目录

------ day01 ------

一、主机配置命令

一、修改主机名 vi /etc/sysconfig/network
二、修改 ip 地址映射 vi /etc/hosts
三、ctrl + c 结束当前终端的前台进程
四、重启 reboot 、init 6
五、关机 halt 、shutdown -h now 、init 0
六、tab 键 ----> 命令的补全和提示功能
space 向下一屏
b 向上一屏
d 向下半屏
u 向上半屏
七、帮助命令
1. 内部命令 shell 自带开机会加载到内存中的
   help 可以查看内部命令的帮助
2. 外部命令 其他应用程序提供的
   man 可以查看外部命令的帮助

二、liunx 基本结构介绍

八、Linux 目录结构

```
dr-xr-xr-x.   2 root root  4096 May 31 04:46 bin
dr-xr-xr-x.   5 root root  1024 May 31 04:47 boot
drwxr-xr-x.  18 root root  3740 Jun  1 03:17 dev
drwxr-xr-x. 102 root root  4096 Jun  1 03:19 etc
drwxr-xr-x.   3 root root  4096 May 31 04:49 home
dr-xr-xr-x.  10 root root  4096 May 31 04:41 lib
dr-xr-xr-x.   9 root root 12288 May 31 04:41 lib64
drwx------.   2 root root 16384 May 31 04:35 lost+found
drwxr-xr-x.   2 root root  4096 Sep 23  2011 media
drwxr-xr-x.   3 root root  4096 May 31 04:50 mnt
drwxr-xr-x.   3 root root  4096 May 31 04:52 opt
dr-xr-xr-x. 148 root root     0 Jun  1 03:17 proc
dr-xr-x---.  24 root root  4096 Jun  1 03:18 root
dr-xr-xr-x.   2 root root 12288 May 31 04:50 sbin
drwxr-xr-x.   7 root root     0 Jun  1 03:17 selinux
drwxr-xr-x.   2 root root  4096 Sep 23  2011 srv
drwxr-xr-x.  13 root root     0 Jun  1 03:17 sys
drwxrwxrwt.  16 root root  4096 Jun  1 03:23 tmp
drwxr-xr-x.  13 root root  4096 May 31 04:36 usr
drwxr-xr-x.  21 root root  4096 May 31 04:44 var
```

在 linux 中所有以 . 开头的文件都是隐藏目录

1. bin ：存放二进制可执行文件(ls,cat,mkdir 等)，常用命令一般都在这里。
2. dev ：用于存放设备文件。/dev/null “黑洞”，所有写入该设备的信息都将消失。

/dev/zero 是类 Unix 操作系统中的一个特殊文件，用来提供一个空字符文件，其一个典型的应用就是提供字符流进行数据存储初始化

3. home : 存放所有用户文件的根目录，是用户主目录的基点。不包括 root 用户
   1).进入家目录的方式　cd ~　cd /home/jingyue
4. mnt　：系统管理员安装临时文件系统的安装点，系统提供这个目录是让用户临时挂载其他的文件系统。
5. root : root 目录是超级用户的目录。
6. sbin : 一般是命令的扩展命令目录
7. tmp : Linux 开关机时自动维护的临时目录,所以不要把文件创建或安装在这个目录下
8. usr : usr 是个很重要的目录，通常这一文件系统很大，因为所有程序安装在这里。

三、修改配置虚拟机网络

1、物理机网络共享设置
2、确定 VMware 网络连接模式 NAT
3、虚拟机中进行网络配置
九、如何修改网卡信息（红色部分注意替换）
ifconfig 显示当前的网络配置信息
1.vi /etc/udev/rules.d/70-persistent-net.rules　修改物理地址 网络连接描述信息
2.vi /etc/sysconfig/network-scripts/ifcfg-eth0　ONBOOT="yes" 启动时是否激活网卡

```
TYPE=Ethernet
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.137.101
NETMASK=255.255.255.0
GATEWAY=192.168.137.1
DNS1=192.168.137.1
HWADDR=00:0c:29:a8:a7:45
```

十、虚拟机的网络连接方式
仅主机 ：虚拟机只能和当前的宿主机相连
桥　接 ：会创建一个实际的 ip,用这个 ip 对外可提供放,可访问外网上网
NAT　：会和宿主机共享一块网卡,不会创建实际的 ip,但可以利用宿主机的 ip 访问外网上网

------ day02 ------

一、Liunx 的常用命令

1.防火墙
1) service iptables status 查看状态
2)service iptables stop 停止
3)chkconfig iptables --list 查看各种运行状态下的防火墙状态
4)chkconfig iptables off 关闭各种运行状态下的防火墙
Chkconfig iptables on --level 234
2.ctrl + l 清空屏幕
3.find -name soft 查找一个文件或目录
4.which pwd 查找一个命令所在的位置
5.pwd 查看当前所在的工作目录
6.ls 列出目录下所有内容
-l 显示详细信息 等价 ll
-a 显示所有文件包括隐藏文件(以.开头的文件是隐藏文件)
-h 人性化展示

7. vi 编辑器
1) 进入编辑模式
i 光标当前 a 光标的下一个位置 o 下一行
esc 退出编辑模式
#yy 复制 p 粘贴 #dd 删除 u 后退 ctrl+r 前进
G 光标定位到最后一行 gg 光标定位到第一行
: 命令行模式 :set nu 显示行号 :set nonu 取消行号
: w 保存 :q 退出 ！强制执行
ctrl + w ,s 水平分屏
ctrl + w ,v 垂直分屏
ctrl + w ,w 移动光标到下一个
ctrl + w ,c 关闭当前屏
ctrl + w ,p 切换分屏光标
/string 从上向下查找字符串 ?string 从下向上查找字符串

8. mkdir 在指定位置创建目录
1) -p 递归创建目录
9. rm 删除
1) -r 递归删除
2) -f 不提示强制执行

10.touch 文件目录 创建一个空文件
11.echo 在控制台输出语句
12.cat 读取一个文件内容到控制台
13.-rw-r--r--. 1 root root 0 Jun 1 09:55 test01
-rw-r--r-- 文件类型和权限

- 文本文件
d 目录
c/b 设备文件
l 连接文件

1 连接次数
root root 属主 属组

14.cp -r 递归拷贝 src path tarpath 将 srcpath 复制到 tarpath cp test01 test01_01
15.mv srcpath tarpath 将 srcpath 移动到 tarpath mv test02 test02_02
如果在当前目录下对文件 mv 相当于是对文件重命名
16.chmod 修改文件权限 r=4 w=2 x=1 chmod 755 test_01_01
chmod o+rx test_01_01
17.tail 读取一个文件尾部指定行数的数据 默认显示 10 条
tail -15 test01 tail -F test01 监控文件尾部数据变化
-F 参数如果文件被删除又重新创建依然可以监控空数据
-f 如果文件删除断开监控状态 -F 如果文件删除不会断开监控状态
18.date 查看时间 date -s "2018-6-4 04:15:19" 设置时间
date +'%Y-%m-%d'
19.kill -9 ps_id 强制关闭进程

二、linux 定时任务

service crond status 查看状态
crontab -e 配置定时任务的内容
* * * * *
分 时 日 月 周 ***3* commend 每下个单位的对应这个时间执行一次
*/1 **** commend 每隔当前对应的时间执行一次
在定时任务中有时会找不到系统配置的环境 尽量把命令写成全路径
如果有多个任务就每个任务一行
因为 linux 是多用户多线程的操作系统,所以控制台的功能有限

date –s “2018-12-16 9:35”查看时间

------ day03 ------

一、ntp 服务器时间同步 - 需要关闭防火墙

1. **编辑 ntpd 服务配置文件 （在时间服务器主机上，想作为时间同步的标准主机）**
   a) **vi /etc/ntp.conf**
      i. 文件中查找如下内容并修改： 修改红色部分
         1. 设置允许哪个网段来访问时间服务器(**ntpd**) 这里允许 227 网段
         2. restrict 192.168.**227.0** mask 255.255.255.0 nomodify notrap
      ii. 注释掉外网服务
         #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
      iii. # 添加如下内容，本机访问，无需外网时间同步时间
         server 127.127.1.0

2. 启动 **ntp** 服务
   a) 查看防火墙状态 确保防火墙关闭 (**service iptables stop**)
      i. [root@hadoop-2 ~]# **service iptables status**
      ii. iptables: Firewall is *not running*.
   b) **查看 ntp 服务状态**
      i. [root@hadoop-1 /]# **service ntpd status**
      ii. ntpd is stopped
   c) 启动 ntp 服务
      service ntpd start / service ntpd restart
      [root@hadoop-1 /]# **service ntpd start**
      Starting ntpd: [OK]

3. **测试 ntp** 时间同步服务功能 、 在其他机器，如 102 机器 ；同步时间
   a) **登录**其他机器
      i. 比如 192.168.137.102 hadoop-2 机器进行操作
   b) **修改**本机**时间**，修改错误了
      i. [root@hadoop-2 ~]# **date -s "2018-12-16 18:46"**
   c) **执行 ntp** 时间同步指令（与 101 ip 主机同步，因为 101 启动了 ntpd 服务）
      i. [root@hadoop-2 ~]# **ntpdate -u 192.168.137.101** (ntp 服务器的地址)
   d) 查看时间验证是否同步成功
      i. [root@hadoop-2 ~]# **date**

   、 102 机器、通过 crontab 定时**进行时间同步
   [root@hadoop-2 ~]# **crontab -e**
   编辑后自动保存到路径：**/var/spool/cron/**
   文件添加以下内容：表示每 2 分钟与 101 机器同步一次
   /2 * * * * /usr/sbin/ntpdate -u 192.168.137.101

13.server 0.cn.pool.ntp.org
fudge 0.cn.pool.ntp.org stratum 10 当前服务器等级如果需要向其他服务器同步时间不要设置成 0
ntpstat
ntpq -p

二、安装 Java 环境变量

1. 配置 java jdk 环境变量

a) Ftp == filezilla 工具
   i. 如何安装
   ii. 如何连接
      1. ip name password port

b) Java JDK 环境
   i. 通过 ftp 工具把 windows 里边的 jdk 复制到 远程机器
      1. 注意：不要放到 root 目录下，这里建议 **/usr/soft**
   ii. 访问、登陆远程机器，找到 那个文件
      1. cd /usr/soft
   iii. 解压文件
      1. tar -zxvf 要解压的 jdk.tar.gz 文件
      2. cd 解压后的 jdk 文件夹
      3. pwd
      4. 复制显示的 jdk 路径 备用
   iv. 配置环境变量
      1. 用 vi 编辑环境变量的配置文件
         a) vi /etc/profile
         b) 打开后在 profile 最下边 添加 三个 export 开头的配置
            i. export JAVA_HOME=/usr/soft/jdk1.8.0_40
            ii. export PATH=$PATH:$JAVA_HOME/bin:
            iii. export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
   v. 生效
      1. **source /etc/profile**
   vi. 测试
      1. java -version
      2. javac

2. Hadoop 环境

a) ftp 拷贝同上
b) hadoop 环境配置
   i. ftp 拖拽同上
   ii. 访问 cd 同上
   iii. 解压同上
   iv. 配置 环境变量文件

1. 用 vi 编辑环境变量的配置文件
   a) **vi /etc/profile**
   b) **打开后在 profile 最下边**
      i. **添加**
         1. **export HADOOP_HOME=/usr/soft/hadoop-2.7.1**
      ii. **更新**
         1. **export PATH=$PATH:$JAVA_HOME/bin: $HADOOP_HOME/bin:$HADOOP_HOME/sbin**
2. 生效
   a) **source /etc/profile**
3. 测试
   a) java -version
   b) javac

三、安装 Hadoop 环境变量

1. 解压 hadoop 安装包
   /usr/soft/hadoop*****
2. 配置 hadoop 环境变量同上 ,（以下仅作参考，以自己实际地址为准）
   c) export HADOOP_HOME=/usr/soft/myhadoop/hadoop-2.7.1
   d) export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
3. 修改 hadoop 配置文件

   hadoop-env.sh
   slaves （这个文件单独添加所有机器的 对应 ip 和 hosts 名）
   core-site.xml
   hdfs-site.xml
   yarn-site.xml
   mapred-site.xml （特殊需要 cp，然后编辑）

hadoop-env.sh 环境变量的配置文件

文件中找到如下两行对应信息，修改等号后边红色部分，为自己的 java 和 hadoop 路径

1) export JAVA_HOME=**/usr/soft/myhadoop/jdk1.8.0_40**
2) export HADOOP_CONF_DIR=**/usr/soft/myhadoop/hadoop-2.7.1**/etc/hadoop

slaves

文件的配置 放值 hadoop 工作节点的目录 主机名 每台机器一行

hadoop-1
hadoop-2
hadoop-3

3.hadoop 的加载顺序是先加载 -default ---> -site ？？？

core-site.xml 找到对应配置内容 进行复制粘贴；红色修改为本机 hosts 名

<!--指定 hdfs 的主端口 namenode 要放在哪台机器-->
<!--配置默认文件系统的名称-->
<!--9000 随意指定的端口 默认端口是 8020-->

<property>
HDFS 对外提供服务的位置
<name>fs.defaultFS</name>
<value>hdfs://**hadoop-1**:9000</value>
</property>
<!--流缓冲区大小-->
<property>
<name>io.file.buffer.size</name>
<value>131072</value>
</property>

hdfs-site.xml　　找到对应配置内容　进行复制粘贴；红色修改为本机 hosts 名

<property>
指定副本数　小于等于 datanode 数
<name>dfs.replication</name>
<value>3</value>
</property>
<property>
在本地(linux)磁盘上存放 namenode 数据的目录
<name>dfs.namenode.name.dir</name>
<value>/usr/soft/name</value>
</property>
<property>
在本地(linux)磁盘上存放 datanode 数据的目录
<name>dfs.datanode.data.dir</name>
<value>/usr/soft/data</value>
</property>
<property>
配置 secondarynamenode 的 Web url
<name>dfs.namenode.secondary.http-address</name>
<value>**hadoop-1711-001**:50090</value>
</property>

yarn-site.xml　　找到对应配置内容　进行复制粘贴；红色修改为本机 hosts 名

<property>
配置服务类型 mapreduce_shuffle
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
指定 resourcemanager 启动在哪台
<name>yarn.resourcemanager.hostname</name>
<value>**hadoop-1711-001**</value>
</property>

mapred-site.xml　　（需要通过 mapred-site.xml.template，拷贝然后修改）

cp　mapred-site.xml.template　mapred-site.xml

vi　mapred-site.xml

<property>
把 mapreduce 的计算依赖给 yarn 框架
<name>mapreduce.framework.name</name>

```
        <value>yarn</value>
    </property>
    <property>
    程序操作历史日志的位置
        <name>mapreduce.jobhistory.address</name>
        <value>hadoop-1711-001:10020</value>
    </property>
    <property>
    web url 访问历史服务的地址
        <name>mapreduce.jobhistory.webapp.address</name>
        <value>hadoop-1711-001:19888</value>
    </property>
    <property>
    存放多少条历史服务
        <name>mapreduce.jobhistory.joblist.cache.size</name>
        <value>20000</value>
    </property>
```

四、克隆虚拟机

1. 右键--->管理--->克隆--->虚拟机中当前状态--->创建完整克隆--->
配置安装包---->完成
克隆完检查　1.ip　2.主机名和 ip 映射　3.防火墙是否关闭

1、 修改 hosts ip
　　a) [root@hadoop-1 hadoop]# **vi　/etc/hosts**
2、 修改主机名称
　　a) [root@hadoop-1 hadoop]# **vi　/etc/sysconfig/network**
3、 查看 ip 和 mac 地址
　　a) [root@hadoop-1 hadoop]# **ifconfig**
4、 查看　网络情况
　　a) [root@hadoop-1 hadoop]# **vi　/etc/udev/rules.d/70-persistent-net.rules**
5、 修改配置网络 ip 文件
　　a) [root@hadoop-1 hadoop]# **cd　/etc/sysconfig/network-scripts/**
　　b) [root@hadoop-1 network-scripts]# **vim　ifcfg-eth0**
　　c) **service　network　restart**
　　d) **ping　www.baidu.com**

五、配置 ssh 免密登录

1.编辑秘钥的配置文件
　　[root@hadoop-2 ~]# **vim　/etc/ssh/sshd_config**
2.找到下边内容，对应去掉前面的#
　　RSAAuthentication yes
　　PubkeyAuthentication yes
　　AuthorizedKeysFile　　.ssh/authorized_keys

3.**service sshd restart** 重启 ssh 服务

3.1 配置 ssdh 服务开机启动，保证 CRT 可以正常连接

chkconfig ssdh on

4.设置 ssh 的密码是无密码并且使用 rsa 非对称加密生成公私钥

设置生成 非对称 秘钥

[root@hadoop-1 ~]# **ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa**

重复 三遍 分别把秘钥 拷贝给 hadoop-1 hadoop-2 hadoop-3

[root@hadoop-1 ~]# **ssh-copy-id hadoop-3**

分别尝试登陆连接

[root@hadoop-1 ~]# **ssh hadoop-1**

退出登陆连接

[root@hadoop-3 ~]# **exit**

六、格式化 HDFS

1. 格式化执行

如果有错误，仔细查看错误提示文件，参考以上配置重新修改；

[root@hadoop-1 ~]# **hadoop namenode -format**

2. 启动两个服务，也可以单独使用

[root@hadoop-1 ~]# **start-dfs.sh**

[root@hadoop-1 ~]# **start-yarn.sh**

启动后查看 第一种 验证

[root@hadoop-1 ~]# **jps**

2787 NameNode

3605 Jps

3064 SecondaryNameNode

3304 NodeManager

3209 ResourceManager

2907 DataNode

启动后查看 第二种 验证

可以访问代表节点 3 的集群表示正常；

主机访问 hdfs 端口

http://192.168.137.101:**50070**

从属机器访问 yarn 端口

http://192.168.137.101:**8088**

3. 停止服务方法

stop-... 停止

[root@hadoop-1 ~]# **stop-dfs.sh**

[root@hadoop-1 ~]# **stop-yarn.sh**

------ day04 ------

hadoop 的作者 Doug Cutting 图标

一、从代码层看 hadoop

1. Hadoop Common: 其他组件的公共组件
2. Hadoop Distributed File System (HDFS™): 分布式的文件系统
3. Hadoop YARN: 资源调度框架
4. Hadoop MapReduce: 数据的计算框架

   从使用的框架 有 2,3,4

二、启动 hadoop

1. start-dfs.sh 启动 HDFS
   1) hdfs 访问的 WEB url 主机 + 50070 端口
   2) namenode 服务 HDFS 的主节点负责维护所有文件,是文件元数据(描述信息)
   3) datanode 服务 实际的数据负责维护实际的数据
   4) SecondaryNameNode 定期的改变 namenode 中的信息,它会拷贝 namenode,所以相当 nn 的冷备份。
2. start-yarn.sh 启动 yarn 负责资源调度的框架
   1) ResourceManager yarn 中的主节点负责分发资源和创建容器

   NodeManager 一般和 datanode 是伴随的服务,负责维护实际数据

三、block 的概念

1. 一个文件切分开存放,每一部分叫一个 block。
2. 默认切分规则是 128M 对应一个 block。
3. 实际的文件大小按照文件的实际值。

四、HDFS shell 命令

1. hdfs dfs -help 帮助
2. -cat 查看文本文件的内容
3. -chmod 修改权限 使用创建的那个用户 hdfs dfs -chmod o+w /test
4. -copyFromLocal == -put

   linux 的路径 hdfs 的路径

   hdfs dfs -copyFromLocal in_use.lock /

   hdfs dfs -put in_use.lock /
5. -copyToLocal == -get

   hdfs 的路径 linux 的路径

hdfs dfs -copyToLocal /in_use.lock ./

hdfs dfs -get /in_use.lock ./

6. -cp hdfs dfs -cp /LICENSE.txt /test/license -p 递归拷贝
7. -ls hdfs dfs -ls / 查看目录下的文件信息
8. -mkdir hdfs dfs -mkdir /test 指定位置创建目录 -p 递归创建
9. -appendToFile

由于 HDFS 上的文件修改不能支持很好,只能在文件末尾添加数据

linux 下的文件 HDFS 上的文件

hdfs dfs -appendToFile in_use.lock /test/in_use.lock

10. -text 高级的查看文本文件的内容
11. -rm hdfs dfs -rm /test/123.rar 删除文件 -f 不询问强制删除 -R 递归删除

hdfs dfs -rm -R -f /test 不支持参数的组合使用

12. -mv hdfs dfs -mv /Text.zip /output180401_2 移动文件
13. rm 删除文件 如果需要加入回收站可以 修改 如下配置文件

core-site.xml

扩展回收站

```
<property>
  <name>fs.trash.interval</name>
  <value>1440</value>
</property>
```

还原回收站文件 -mv 到其他目录即可

------ day05 ------

————————周总结考核

周考内容：

01、使用的 Linux 的名称、属于哪个主流发行版本的分支？

cnetos 6.5; 红帽系列的社区版本

02、查看 /usr/soft 下的详细信息，包括隐藏文件的命令是什么？

ls -la

03、同时创建一个多级目录、在/usr/soft/huagong （soft huagong 都不存在）

mkdir -p /usr/soft/huagong (递归创建 -p)

04、简述 NAT 模式、桥接模式、仅主机模式的作用？

NAT 没有独立 ip、与宿主机想用相同网卡设施；

桥接有独立 ip 相当于独立机器；

仅主机、只能和宿主机相连，不能访问互联网和局域网；

05、Apache hadoop 的四大核心模块儿是什么？

hadoop common : 公共组件、代码底层公用组件

hadoop hdfs： 分布式文件系统

mapraduce：

yarn：

06、core-site.xml 哪个属性配置 HDFS 主节点？（NameNode）

fs.defaultFS

07、如何启动 HDFS? start-dfs.sh

08、如何启动 YARN？ start-yarn.sh

09、搭建集群为何要配置 SSH 免秘钥登录？

hadoop 集群运行中需要通过 ssh 协议相互发送信息。

10、HDFS 中 block 的概念是什么？

如果文件存储在 hdfs 上，存储的单位叫做 block。

11、block 的默认大小多少？ 128M

12、50070 访问的功能是？ hdfs 浏览器【查看】端口；只能查看

13、8088 访问的功能是？ yarn 浏览器【查看】端口；只能查看

14、HDFS 启动后有哪些进程被启动？（jps 可以看到）

NameNode 存放文件描述信息位置；

DataNode 存放实际数据的内容； 可以有多个；

15、yarn 启动后有哪些进程被启动？

ResourceManager 一个集群只有一个，在主节点上；

NodeManager 根据多少个 DataNode 出现多少个 NodeManager

————————构建 Maven 项目

1. 新建项目(Eclipse / MyEclipse)
2. 选择 Maven
   ① org.apache.maven.archetypes maven-archetype-quickstart 1.1
3. 导入配置依赖路径 settings.xml 文件
   ① Windows - Preferences - Maven - User Settings
   ② user setting s (open file); browse 选择 settings.xml 文件
4. 编辑 pom.xml 配置文件

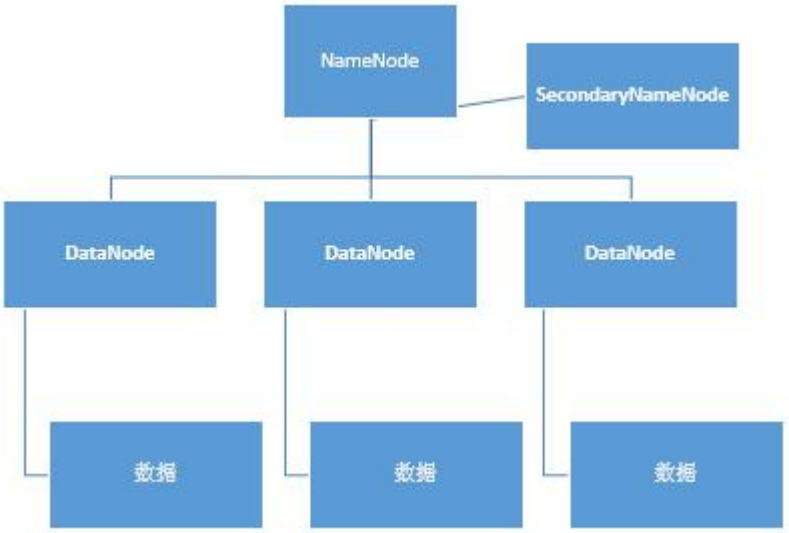
添加 eclipse maven HDFSapi 依赖

```
<dependency>
        <groupId>org.apache.hadoop</groupId>
        <artifactId>hadoop-client</artifactId>
        <version>2.7.1</version>
</dependency>
<build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

5. 测试项目运行

```
//import java.net.URI;
    //import org.apache.hadoop.conf.Configuration;
    //import org.apache.hadoop.fs.FileSystem;
    public class App
    {
        public static void main( String[] args ) throws Exception
        {
            System.out.println( "Hello World!" );
            Configuration conf = new Configuration();
            FileSystem fileSystem = FileSystem.get(new URI("hdfs://hadoop-1:9000"), conf, "root");
            fileSystem.close();
        }
    }
```

一、HDFS 基本架构关系图解



参考：http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html#Introduction

HDFS（Hadoop Distributed File System ）Hadoop 分布式文件系统

基本结构分 NameNode、SecondaryNameNode、DataNode 这几个

NameNode：是 Master 节点，有点类似 Linux 里的根目录。管理数据块映射；处理客户端的读写请求；配置副本策略；管理 HDFS 的名称空间；

Secondary NameNode：保存着 NameNode 的部分信息（不是全部信息 NameNode 宕掉之后恢复数据用），是 NameNode 的冷备份；合并 fsimage 和 edits 然后再发给 namenode。（防止 edits 过大的一种解决方案）

DataNode：负责存储 client 发来的数据块 block；执行数据块的读写操作。是 NameNode 的小弟。

热备份：b 是 a 的热备份，如果 a 坏掉。那么 b 马上运行代替 a 的工作。

冷备份：b 是 a 的冷备份，如果 a 坏掉。那么 b 不能马上代替 a 工作。但是 b 上存储 a 的一些信息，减少 a 坏掉之后的损失。

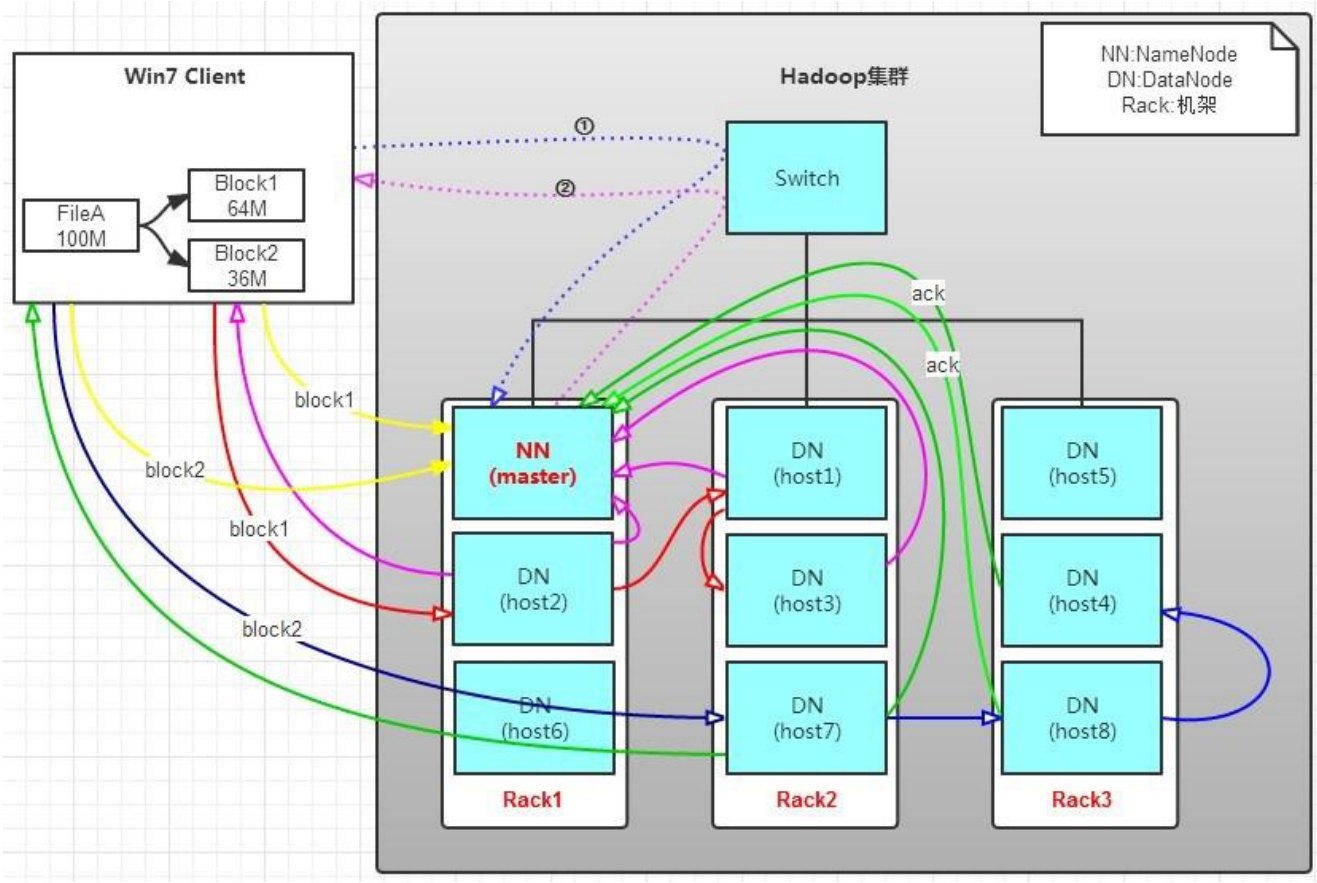
fsimage:元数据镜像文件（文件系统的目录树。）

edits：元数据的操作日志（针对文件系统做的修改操作记录）

namenode 内存中存储的是=fsimage+edits。

参考：https://www.cnblogs.com/wxplmm/p/7239342.html

二、HDFS 文件读写过程图解



a. Client 将 FileA 按 128M 分块。分成两块，block1 和 Block2;

b. Client 向 nameNode 发送写数据请求，如图蓝色虚线①------>。

c. NameNode 节点，记录 block 信息。并返回可用的 DataNode，如粉色虚线②--------->。

Block1: host2,host1,host3

Block2: host7,host8,host4

d. client 向 DataNode 发送 block1；发送过程是以流式写入。

hdfs 流式写入过程

1>将 128M 的 block1 按 128k 的 package 划分;

2>然后将第一个 package 发送给 host2;

3>host2 接收完后，将第一个 package 发送给 host1，同时 client 想 host2 发送第二个 package；

4>host1 接收完第一个 package 后，发送给 host3，同时接收 host2 发来的第二个 package。

5>以此类推，如图**红线实线**所示，直到将 block1 发送完毕。

6>host2,host1,host3 向 NameNode，host2 向 Client 发送通知，说"消息发送完了"。如图粉红颜色实线所示。

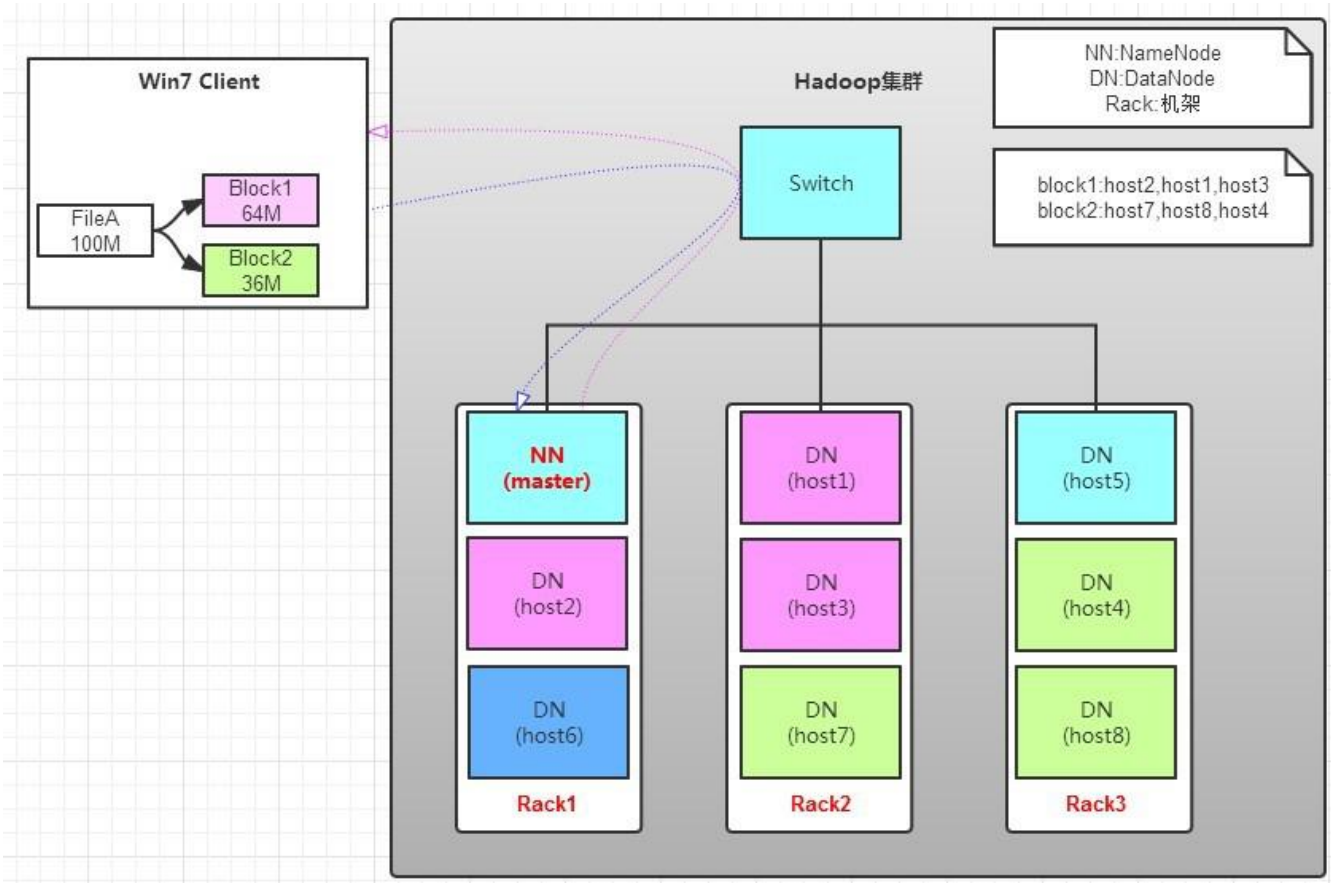
7>client 收到 host2 发来的消息后，向 namenode 发送消息，说我写完了。这样就真完成了。如图**黄色粗实线**

8>发送完 block1 后，再向 host7，host8，host4 发送 block2，如图**蓝色实线**所示。

9>发送完 block2 后，host7,host8,host4 向 NameNode，host7 向 Client 发送通知，如图浅**绿色实线**所示。

10>client 向 NameNode 发送消息，说我写完了，如图**黄色粗实线**。。。这样就完毕了。

hdfs 文件读操作



读操作就简单一些了，如图所示，client 要从 datanode 上，读取 FileA。而 FileA 由 block1 和 block2 组成。

那么，读操作流程为：

a. client 向 namenode 发送读请求。

b. namenode 查看 Metadata 信息，返回 fileA 的 block 的位置。

block1:host2,host1,host3

block2:host7,host8,host4

c. block 的位置是有先后顺序的，先读 block1，再读 block2。而且 block1 去 host2 上读取；然后 block2，去 host7 上读取；

上面例子中，client 位于机架外，那么如果 client 位于机架内某个 DataNode 上，例如,client 是 host6。那么读取的时候，遵循的规律是：

优选读取本机架上的数据。

运算和存储在同一个服务器中，每一个服务器都可以是本地服务器

三、HDFS 基础理论知识

一、HDFS 的安全模式 安全模式下 HDFS 是只读

1. hadoop dfsadmin -safemode **get** 查看是否是安全模式
2. hdfs 在启动的时候回去检测各节点上的文件是否正常,进入安全模式
3. hadoop dfsadmin -safemode **leave** 关闭安全模式
4. hadoop dfsadmin -safemode **enter** 开启安全模式
5. 没有文件不会进入安全模式

二、namenode 的大小是固定的------>无法高效存储大量小文件。

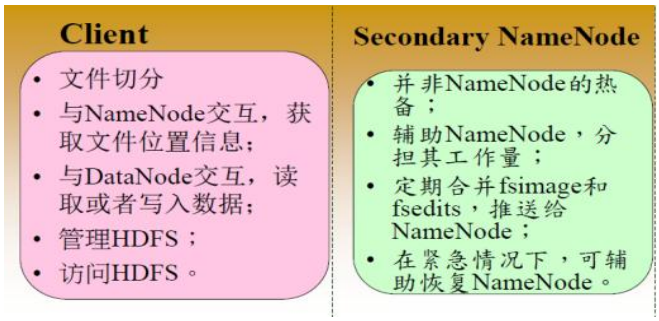
三、namenode 在启动后会把数据加载到**内存**中

四、查看块报告

（1）hdfs dfsadmin -report **1 小时报告一次**

五、心跳 3 秒一次 超过 10 分钟认为这个节点不可用

六、checksum 文件创建后的三周开始检测



------ day06 ------

一、HDFS 常用 API 调用前提

1. HDFS 服务器启动正常

2. eclipse 创建 maven 环境正常

————参考

源码参考：https://blog.csdn.net/ccorg

笔记参考：https://github.com/nmww/bigdata01

二、HDFS 常用 API 应用

1. 下载 HDFS 服务器文件

————下载 HDFS 服务器文件

```
//用到的包
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
/**
 * @throws Exception
 * HDFS API   从  hdfs 服务器下载指定文件到本地 windows 指定路径
 */
private static void getHdfsFileToWindowLocal() {
    //  创建一个 hdfs 运行环境的一个对象
    Configuration conf = new Configuration();
    FileSystem fs = null;
    Path src = new Path("/NOTICE.txt");//修改对应自己  hdfs 服务器路径文件
    Path dst = new Path("F:\\ptest");   //注意 java 中 windows 路径的转义符
    //Path dst1 = new Path("F:/ptest");   //   可以用两种   \\ OR   /
    try {
        fs = FileSystem.get(new URI("hdfs://hadoop-1:9000"), conf, "root");

        // fs.copyToLocalFile(src, dst);   //windows  和  linux 系统  交互不适用
        /*
```

```
             * windows 需要有 hdfs 环境，或者设置兼容本地操作系统
             * 1.是否删除源文件；
             * 2.src  源文件地址
             * 3.dst 目标文件地址
             * 4.useRawLocalFileSystem  是否兼容本地操作系统
             */
            fs.copyToLocalFile(false, src, dst, true);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                fs.close();//占用资源的对象，用完及时关闭；
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }//占用资源的对象，用完及时关闭；
        }

    }
```

2. 读取 HDSF 服务器文件

```
————读取 HDSF 服务器文件
    /**
     * 从 hdfs 服务器读取一个文件，打印到控制台
     * import org.apache.hadoop.io.IOUtils;
     */
    private static void catFileToConsole() {
        Configuration conf = new Configuration();
        FileSystem fs = null;
        try {
            fs = FileSystem.get(new URI("hdfs://hadoop-1:9000"), conf, "root");
            //获取输入流；对应路径是 hdfs 已经存在的文件绝对路径
            FSDataInputStream in = fs.open(new Path("/p1"));
            //1.输入流；  参数 2：输出流；输出到控制台；  3.当前环境
            IOUtils.copyBytes(in, System.out, conf);
            IOUtils.closeStream(in);//关闭输入流  -  工具类
        } catch (Exception e) {
            e.printStackTrace();//  打印异常栈中信息
        } finally {
            if (null != fs)
                try {
                    fs.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
```

```
                }
        }

    }
```

3. 查看 HDFS 服务器上文件状态

————查看 HDFS 服务器上文件状态

```
    /**
     * 查看 HDFS 服务器上，指定文件或者目录的所有文件的状态
     * @see import org.apache.hadoop.fs.FileStatus;
     * @see listStatus
     */
    private static void statusFile() {
        Configuration conf = new Configuration();
        FileSystem fs = null;
        try {
            fs = FileSystem.get(new URI("hdfs://hadoop-1:9000"), conf, "root");
            // 获取指定文件,或文件夹的所有文件，返回一个数组
            FileStatus[] listStatus = fs.listStatus(new Path("/"));
            for (int i = 0; i < listStatus.length; i++) {
                //打印文件名称
                System.out.println(listStatus[i].getPath().getName());
                System.out.println("绝对路径： "+listStatus[i].getPath().toString());
                //打印文件大小
                System.out.println(listStatus[i].getLen() + " byte");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (null != fs)
                try {
                    fs.close();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
        }
    }
```

4. 将 HDFS 文件写入本地文件

————将 HDFS 文件写入本地文件

```
    /**
     * 将 hdfs 指定文件，写入本地的指定文件
     * @param conf
     * @param fs
     * @throws Exception
```

```
     * @see getFileByStream(conf,fileSystem);
     */
    private static void getFileByStream(Configuration conf, FileSystem fs) throws Exception{
        //打开 hdfs 服务器指定文件
        FSDataInputStream open = fs.open(new Path("/p1"));
        // 写入指定的文件中，文件需要存在
        FileOutputStream out = new FileOutputStream("F:\\ptest\\ceshi.txt");
        IOUtils.copyBytes(open, out, conf);
        IOUtils.closeStream(open);
        IOUtils.closeStream(out);
    }
```

5. 在 HDFS 上创建目录

```
————在 HDFS 上创建目录
    /**
     * 在 HDFS 上创建目录
     * @param fs
     * @throws Exception
     * @see mkdirOnHdfs(fileSystem);
     */
    private static void mkdirOnHdfs(FileSystem fs) throws Exception{
        boolean isOK= fs.mkdirs(new    Path("/hdfsTest1"));
        System.out.println(isOK ? "mkdir ok" : "mkdir false");
    }
```

6. 上传、拷贝本地文件到 HDFS

```
————上传、拷贝本地文件到 HDFS
    /**
     * @param source
     * @param dest
     * @throws Exception
     * @throws URISyntaxException
     * @see example：copyFromLocal("F:/ptest","/ceshi.txt");
     */
    public static void copyFromLocal(String source, String dest) throws Exception{
        Configuration conf = new Configuration();
        URI uri = new URI("hdfs://hadoop-1:9000");
        FileSystem fileSystem = FileSystem.get(uri, conf, "root");
        Path srcPath = new Path(source);
        Path dstPath = new Path(dest);
        if (!(fileSystem.exists(dstPath))) {
            // 如果路径不存在，即刻创建
            fileSystem.mkdirs(dstPath);
        }
        String filename = source.substring(source.lastIndexOf('/') + 1,source.length());
```

```
        fileSystem.copyFromLocalFile(srcPath, dstPath);
        System.out.println("File " + filename + " copied to " + dest);
        fileSystem.close();
    }
```

————下午 HDFS API

7. 在 HDFS 上创建文件，并写入内容

————在 HDFS 上创建文件，并写入内容

```
    /**
     * 在 HDFS 上创建文件，并写入内容
     * @param fs
     * @throws Exception
     */
    public static void createFile(FileSystem fs)throws Exception{
        /*
         * 第二个参数；默认 true 会覆盖，如果是 false，文件存在，会抛异常
         * org.apache.hadoop.fs.FileAlreadyExistsException
         */
        FSDataOutputStream out = fs.create(new Path("/a"));
        out.writeUTF("111this is hdfs java api create \n");
        out.close();
    }
```

8. 在 HDFS 上的文件进行追加内容

————在 HDFS 上的文件进行追加内容

```
    /**
     * 在 HDFS 上的文件进行追加内容
     * @param fs
     * @throws Exception
     */
    public static void appendFile(FileSystem fs)throws Exception{
        /*
         * @parm  2 ：缓冲区，IO 交互提高效率
         */
        FSDataOutputStream out = fs.append(new Path("/a"),2048);
        out.writeUTF("append this hdfs java api append \n");
        out.close();
    }
```

9. 删除 hdfs 指定文件夹

————删除 hdfs 指定文件夹

```
/**
 * hdfs 删除指定文件夹
 * @param fs
 * @throws Exception
 */
public static void deleteDir(FileSystem fs)throws Exception{
    if(fs.exists(new Path("/output01"))){
        FileStatus[] status = fs.listStatus(new Path("/output01"));
        for (int i = 0; i < status.length; i++) {
            boolean directory = fs.isDirectory(new Path(status[i].getPath().toString()));
            if(directory){

            }else{
                fs.delete(new Path(status[i].getPath().toString()),false);
            }
        }
    }
}
```

10. 递归删除 HDFS 上指定的文件夹

————递归删除 HDFS 上指定的文件夹

```
/**
 * deleteEmptyDirAndFile(new Path("/output01"));
 *  递归删除  给定目录的全部内容
 * @throws Exception
 */
public static void deleteEmptyDirAndFile(Path path) throws Exception {
    Configuration conf = new Configuration();
    FileSystem fs = FileSystem.get(new URI("hdfs://hadoop-1:9000"),conf,"root");
    //当是空文件夹时
    FileStatus[] listStatus = fs.listStatus(path);
    System.out.println(listStatus.length);
    if(listStatus.length == 0){
        fs.delete(path, true);
        return;
    }
    // 该方法的结果：包括指定目录的   文件 和 文件夹
    RemoteIterator<LocatedFileStatus> listLocatedStatus = fs.listLocatedStatus(path);
    while (listLocatedStatus.hasNext()) {
        LocatedFileStatus next = listLocatedStatus.next();
        Path currentPath = next.getPath();
        // 获取父目录
        Path parent = next.getPath().getParent();
```

```
            // 如果是文件夹，继续往下遍历，删除符合条件的文件（空文件夹）
            if (next.isDirectory()) {
                // 如果是空文件夹
                if(fs.listStatus(currentPath).length == 0){
                    // 删除掉
                    fs.delete(currentPath, true);
                }else{
                    // 不是空文件夹，那么则继续遍历
                    if(fs.exists(currentPath)){
                        deleteEmptyDirAndFile(currentPath);
                    }
                }
            // 如果是文件
            } else {
                // 获取文件的长度
                long fileLength = next.getLen();
                // 当文件是空文件时， 删除
            //    if(fileLength == 0){
                    fs.delete(currentPath, false);
            //    }
            }
            // 当空文件夹或者空文件删除时，有可能导致父文件夹为空文件夹，
            // 所以每次删除一个空文件或者空文件的时候都需要判断一下，如果真是如此，那么就需要把该文件夹也删除掉
            int length = fs.listStatus(parent).length;
            if(length == 0){
                fs.delete(parent, true);
            }
        }
    }
```

------ day07 ------

0. HDFS 环境配置 - 顺序

01.创建虚拟机安装系统
02.配置虚拟机 IP
03.设置主机名、hosts 名
04.拷贝 jdk、hadoop-jdk
05.解压 jdk 和 hadoop
06.配置 java jdk 、测试
07.配置 hadoop jdk 、 测试
08.开机关闭防火墙，开机启动 sshd 服务
09.配置 hadoop 参数文件
10.重启系统并克隆 2、3 机器
11.设置 2 号机 ip，设置主机名，hosts 名，重启
12.设置 3 号机 ip，设置主机名，hosts 名，重启
13.通过 CRT 连接三台机器
14.分别配置三台机器的 ssh 免秘钥认证、测试
15.格式化主机 HDFS 系统
16.启动服务 hdfs 、yarn
17.配置访问机器 hosts，用 ip 和 hosts 名字访问 50070 8088
18.镜像快照备份

1. 操作举例 MR 计算

使用 MR 先计算一个 wordcount 出来
/usr/soft/hadoop-2.7.1/share/hadoop/mapreduce
[root@hadoop-1 mapreduce]# yarn jar hadoop-mapreduce-examples-2.7.1.jar wordcount /NOTICE.txt /output02

java --> jvm 中
MapReduce 离线计算框架：当前计算统计一个文件中单词出现次数、

2. uber 模式（节省资源）

2.1、配置 uber 模式 ：

如果每次运行时间很短,但是运行次数很多,会重复的开启和销毁 JVM,
开启 Uber 可以复用 JVM，避免频繁的开关 JVM 的资源浪费。
将配置文件追加到 hadoop 配置文件 **mapred-site.xml** 后边
通过下边命令可以快速配置给其他机器； 三台机器都要配置

```
[root@hadoop-1 hadoop]# scp mapred-site.xml hadoop-3:/usr/soft/hadoop-2.7.1/etc/hadoop/mapred-site.xml
<!--开启 uber 模式-->
<property>
    <name>mapreduce.job.ubertask.enable</name>
    <value>true</value>
</property>
<!---同时间可以容纳的最大数据量  单位 : byte-->
<property>
    <name>mapreduce.job.ubertask.maxbytes</name>
    <value>102400000</value>
</property>
<!--可以同时运行多少个 job(计算)-->
<property>
   <name>mapreduce.job.jvm.numtasks</name>
   <value>10</value>
</property>
```

2.2、进入路径

/usr/soft/hadoop-2.7.1/share/hadoop/mapreduce

2.3、配置后再次运行计算

```
[root@hadoop-1 mapreduce]# yarn jar hadoop-mapreduce-examples-2.7.1.jar wordcount /NOTICE.txt /output03
```

2.4、没有错误，查看如下一行记录 uber true

```
18/12/12 09:45:29 INFO mapreduce.Job: Job job_1544575783224_0003 running in uber mode : true
```

3.聚合日志（查看日志）

浏览器查看运行情况历史记录，无法正常查看；
配置聚合日志后可以浏览器查看历史记录；

3.1 配置到 **yarn-site.xml** 文件中

```
<property>
<!--开启聚合日志-->
    <name>yarn.log-aggregation-enable</name>
    <value>true</value>
</property>
<property>
<!--聚合日志的过期时间 单位秒，当前设置 1 天-->
    <name>yarn.log-aggregation.retain-seconds</name>
    <value>86400</value>
</property>
<!--检查过期日志的时间 超过指定日期删除过期日志-->
<property>
    <name>yarn.log-aggregation.retain-check-interval-seconds</name>
    <value>3600</value>
</property>
```

```
<property>
<!--公共访问的路径 日志存储位置-->
    <name>yarn.nodemanager.remote-app-log-dir</name>
    <value>/tmp/logs</value>
</property>
```

3.2 重启 yarn 服务

```
2944 DataNode
4466 Jps
2837 NameNode
4069 NodeManager
3111 SecondaryNameNode
3960 ResourceManager
```

3.3 **启动历史服务器** (停止服务替换成 stop 即可)

mr-jobhistory-daemon.sh start historyserver

starting historyserver, logging to /usr/soft/hadoop-2.7.1/logs/mapred-root-historyserver-hadoop-1.out

3.4 jps 查看

4431 JobHistoryServer

3.5 测试 mr 计算文件单词数量

[root@hadoop-1 mapreduce]# yarn jar hadoop-mapreduce-examples-2.7.1.jar wordcount /NOTICE.txt /outputhistory01

查看生成的结果（注意，对应自己有的路径和文件名）

[root@hadoop-1 mapreduce]# hdfs dfs -cat /outputhistory01/part-r-00000

3.6 查看链接 【MR 学习调试的重要工具】

http://hadoop-1:**19888**

---> **Job Id** 可以点击查看详情

---> 右侧点击 **logs** 进入详情

---> **here** 点击 （看到 mr 运行栈信息，可以在这里查看错误，参考调试）

3.7 为什么要用聚合日志：

mr 运行的时候日志分发到不同机器中，不方便进行查看，
通过配置聚合日志可以方便的通过浏览器进行查看日志，
并在后续学习 mr 过程中，可以参考日志进行错误调试。

注意：

hdfs 启动，只能在 hdfs.site.xml 配置文件指定的机器启动；
yarn 启动，只能在 yarn.site.xml 配置文件指定的机器启动；

备忘：

crontab -e #配置的内容在下边路径生成文件，长期保持

[root@hadoop-2 ~]# cd /var/spool/cron/

Linux 设置节省资源的方式：

0-6 的模式 ； 默认 id 5 是图形界面

[root@hadoop-1 ~]# vi /etc/inittab
编辑后 id 修改成 3 只用命令行模式，不用图形化界面，可以节省资源
id:3:initdefault:

------ day08 ------

0. SHELL 脚本语言

1、shell 定义

Shell Script ， Shell 脚本与 Windows/Dos 下的批处理相似，也就是用各类命令预先放入到一个文件中，方便一次性执行的一个程序文件，主要是方便管理员进行设置或者管理用的。但是它比 Windows 下的批处理更强大，比用其他编程程序编辑的程序效率更高，它使用了 Linux/Unix 下的命令。

2、Linux 的发展回顾

2.1 简介
2.2 版本
2.3 应用领域
2.4 Linux vs Windows

3、第一个 shell 脚本

注：linux 中文件不依据文件后缀区分文件类型；
默认 shell 文件的扩展名是 （.sh）

创建 test01.sh 文件
#—— 注： echo -n 不换行输出
#—— 注： echo 换行输出

```
#!/bin/bash
echo "Hello Shell!"   # 显示输出
```

4、运行 Shell 脚本有两种方法

4.1 作为可执行程序
所有用户 添加/减少 x 权限

```
[root@hadoop-1 testShell]# chmod +x test01.sh
[root@hadoop-1 testShell]# chmod -x test01.sh
```

当前用户 添加/减少 x 权限

```
[root@hadoop-1 testShell]# chmod u+x test01.sh
[root@hadoop-1 testShell]# chmod u-x test01.sh
```

-rwxr--r--. 1 root root 32 Dec 13 08:54 test01.sh
./test01.sh

4.2 作为解释器参数

[root@hadoop-1 testShell]# /bin/sh test02.sh

创建文件 test02.sh --> for 循环的应用

```
#!/bin/bash
echo "Hello Shell!"   #显示输出引号中内容
for i in 123456
do
    echo $i ======
done
```

正常执行

[root@hadoop-1 testShell]# /bin/sh test02.sh

执行并检查是否有错误语法 -n

[root@hadoop-1 testShell]# /bin/sh -n test02.sh

错误提示

```
test02.sh: line 3: syntax error near unexpected token `index'
test02.sh: line 3: `for i index 123456'
```

执行 显示执行过程 -x

[root@hadoop-1 testShell]# /bin/sh -x test02.sh

```
+ echo 'Hello Shell!'
Hello Shell!
+ for i in 123456
+ echo 123456======
123456======
```

5、Shell 变量

————注：空格不能随意使用；

第一行需要写：

（脚本第一行不写的情况，是要前提配置了默认的 sh 脚本解析环境变量）

#!/bin/bash

5.1 定义

```
your_name="huagong"   # 使用双引号拼接
your_name='huagong'   # 使用单引号拼接
greeting="hello, "$your_name" !"
```

命令引入符号 ` ` ，数字 1 左边的符号

`ls /etc `

————注：如果一行存在两条语句 中间需要 （ ; ）分号分割

否则每行语句单独一行

#!/bin/bash

```
# ls /etc show name    set to file
for file in `ls /etc` ; do
    echo "local file name is : ${file}"
done
```

5.2 使用

```
#参数的定义和输出两种方式
#!/bin/bash
your_name="qinjx"
echo $your_name
echo ${your_name}        #显示变量的内容
echo ${#your_name}       #获取长度：5
echo ${your_name:1:4}    #获取 injx
打印输出变量的内容
    echo    $greeting    $greeting_1
```

5.3 只读变量

```
myUrl="www.baidu.com"
readonly myUrl
#此后不可用在修改或者注销 myUrl 否则报错，提示
test04.sh: line 7: myUrl: readonly variable
```

5.4 删除变量

```
myUrl="baidu.com"
unset myUrl          #注销修饰的变量
echo myUrl #无内容输出，因为已经注销了变量
```

5.5 变量类型

6、Shell 字符串

6.1 单引号

单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的

6.2 双引号

```
#双引号可以打印    ${your_name}
#单引号不能打印    ${your_name}
#!/bin/bash
echo " ===\"====double yinhao "
your_name="huagong"
g="hello1 , "$your_name" ! "
g1="hello2 , ${your_name}"
echo $g $g1
echo " ===\'====double yinhao "
your_name="danyinhao"
g='dan hello1 , '$your_name' ! '
g1='dan can not show \${name} hello2 , ${your_name}'
echo $g $g1
————注：
    #解析特殊转义字符 echo -e   这里解析 \n 换行
    b="nihao \n"
```

```
        echo -e $b
```

6.3 拼接字符串

```
your_name=abc
greeting="hello, "$your_name" !"
greeting="hello, ${your_name} !"
```

6.4 获取字符串长度

```
#参数的定义和输出两种方式
#!/bin/bash
your_name="qinjx"
echo $your_name
echo ${your_name}
echo ${#your_name} #获取长度：5
```

6.5 提取子字符串

```
echo ${your_name:1:4} #获取给定下标间的字符串 injx
```

6.6 查找子字符串

```
your_name="qinjx"
echo `expr index "$your_name" i`   #得到 i 的第一个位置 1
```

7、Shell 数组

7.1 定义数组

```
#!/bin/bash
arr=(v1 v2 v3 v4 v5) #声明一个数组
```

7.2 读取数组

```
echo ${arr[0]}   #输出第一个数据
echo $arr       #输出第一个数据
echo after change arr[0]
arr[0]=v0       #改变数组指定下表的对应元素
echo ${arr[0]} #输出修改后的 0 位置的元素
echo after change arr[100] no continue index
arr[100]=v100       #数组的角标可以不连续
echo ${arr[90]}           #没有设置 显示空行
echo ${arr[100]}    #输出 100 位置的元素
echo 输出数组的所有元素'${arr[@]}'
echo ${arr[@]}            #获得数组所有的元素
echo 输出数组的所有元素'${arr[*]}'
echo ${arr[*]}
#====用 for 循环遍历数组
      echo for 显示 arr
      for i in ${arr[@]} ;do
        echo $i ; done
```

7.3 获取数组的长度

```
echo 获取数组 【长度】 '${#arr[@]}'
echo ${#arr[@]}
echo 获取当前元素值的长度'${#arr[100]}'
```

```
echo ${#arr[100]}
```

8、Shell 注释

8.1 单行注释 #

8.2 多行注释

```
:<< abc
    abc 是自定义
    在 abc 之间的是多行注释
    ':<<' 是固定的开始
abc
```

9、Shell 传递参数

```
[root@hadoop-1 testShell]# sh argument.sh a b c d e
    #!/bin/bash
    echo ==显示第一个参数   '$1'
    echo  第一个参数是:$1
    echo ==显示所有参数   '$@'
    echo  所有参数是：$@
    echo ==单一字符串返回所有参数   '$*'
    echo  单一字符串获得所有参数是：$*
    echo ==脚本运行的当前进程 ID 号:'$$'
    echo  脚本运行的当前进程 ID 号: $$
    echo ==后台运行的最后一个进程的 ID 号:'$!'
    echo  后台运行的最后一个进程的 ID 号:$!

    echo ==双引号中 '$@ 和 $*的区别'
    echo "双引号中 \$*"
    for i in "$*" ;do
            echo $i ;done
    echo "双引号中 \$@"
    for i in "$@" ;do
            echo $i ;done
#结果是显示：
    ==双引号中 $@ 和 $*的区别
    双引号中 $*
    a b c d e
    双引号中 $@
    a
    b
    c
    d
    e
```

————shell 流程控制

#————注意：

对比数字使用既能使用-eq、-ne、-gt、-ge、-lt、-le

-gt 大于 是 greater than

-ge 大于等于 是 greater than or equal to

-lt 小于 是 less than

-le 小于等于 是 less than or equal to

-eq 相等 是 equal

-ne　不相等　　是 unequal
也能使用 ==、<、>、!=

10、流程控制

10.1 条件表达式 if

```
# ————————if　开始 fi 结束
    #!/bin/bash
    # 注意：if 的后边一定要有 空格
    # 注意：[　]　符号的左右要有 空格
    if [ $(ps -ef | grep -c "ssh") -gt 1 ]
    # -gt　是判断是否大于的意思 ＞
    then
      echo "true"
    fi

# ————————if elif else fi
    #!/bin/bash
    a=10
    b=20
    # 注：== 符号左右要有空格
    if [ $a == $b ]
     then
      echo "a 等于 b"
    elif [ $a -gt $b ] # -gt 大于
     then
      echo "a 大于 b"
    elif [ $a -lt $b ] # -lt 小于
     then
      echo "a 小于 b"
    else # 里边没有 then
      echo "nothing"
    fi # if 语句的结束标志

# ————————if 判断两个表达式是否相等 -eq
    #——注：test 命令用于检查某个条件是否成立，
        它可以进行数值、字符和文件三个方面的测试
    #!/bin/bash
    num1=$[2*3]
    num2=$[1+5]
    echo num1 = ${num1}
    echo num2 = ${num2}
    # if 判断两个表达式是否相等　-eq
    if test ${num1} -eq ${num2}　# 数值是否相同
     then
      echo "两个表达式相等"
```

```
else
   echo "不相等"
fi
```

10.2 for 知道固定次数使用

```
# ————————FOR 循环
#!/bin/bash
# 注:每行一条语句，如果多条需要分号 ； 分割
for loop in 1 2 3 4 5
do
 echo $loop # 循环输出每个值
done
```

10.3 while：知道退出条件使用

```
#————————— while 满足条件循环
#—— 注： while 右边 括号 空格
    #!/bin/bash
    a=1
    while [ $a -lt 20 ]
     do
      echo $a
      # `` 命令引入 需要包含等号右侧加法
      a=`expr $a + 1`
     done
#————————— until 不满足条件循环
    #!/bin/bash
    b=1
    # until 不满足条件执行 [ 表示空格
    until [ $b -ge 20 ]    # -ge 大于等于
     do
      echo $b
      #` ` 命令引入 需要包含等号右侧加法
      b=`expr $b + 1`
     done
#————————— 死循环
    #!/bin/bash
    while :
    do
     echo "死循环 while 后边需要冒号 ： "
    done
```

10.4 分支语句 case

```
#————————case 开始 --> esac 结束
    #!/bin/bash
    echo "输入 1-4 之间的额数字:"
    ehco "输入的数字为： "
    # 分支语句 case
    # read 从控制台读取输入的内容
    read aNum
    case $aNum in
     # 每个 case 都需要两个分号结束
     1) echo "你输入了 1" ;;
     2) echo "你输入了 2" ;;
     3) echo "你输入了 3" ;;
     4) echo "你输入了 4" ;;
     5) echo "你输入了 5" ;;
     *) echo " *) 表示 default 你输入的不是有效数字" ;;
    esac #case 反过来写作为结束标识
#————————— break
    #!/bin/bash
    while :
     do
      echo -n "输入 1-5 之间的数字:"
      read aNume   # 读取控制台输入，赋值给变量
      case $aNume in
        1|2|3|4|5) echo "你输入的数字是： " $aNume;;
        *) echo "输入的数字无效"
            break ;;
      esac
     done
#————————— continue
    #!/bin/bash
    # 无限循环语法格式 while 后边有 冒号 :
    while :
     do
      echo -n "输入 1-5 数字： "
      read aNum
      case $aNum in
       [1-5]) echo  aNum  =  $aNum ;;
       "x") echo 'x quit'
          break;;
       *) echo "your input out of 1-5!!!!!"
          continue ;;
      esac
    done
```

#————练习

```
# ——1、判断本地文件扩展名，将 .sh 的扩展名都去掉
    #!/bin/bash
    for file in `ls ./`
     do
      #prefix=${file:`expr ${#file} - 3`}
      l=`expr ${#file} - 3` # 得到文件的长度，不包含扩展名
      k=${file:l:l+3}    # 得到扩展名部分  .扩展名
      #echo $k              # 打印所有扩展名
      if [ $k == '.sh' ] # 判断扩展名是否 是  .sh
       then
        newName=${file:0:l}  # 得到去掉扩展名后的名称
        mv ${file} ${newName} # 修改文件名 为新的没有扩展名的名称
        #echo $newName
      fi   # 结束 if
     done # 结束 for

# ——2、判断输入的 名称 是一个 文件 还是一个 文件夹
    #!/bin/bash
    if [ -f $1 ]       # 判断参数值 文件 类型
      then
       echo file =   $1
      elif [ -d $1 ]  # 判断参数是 文件夹 类型
       then
        echo dir =   $1
      else
        echo ' $1 nothing ' = $1
    fi
# ——3、修改本地所有文件的名称，文件夹之外的文件，添加.sh
    #!/bin/bash
    # 1.获取到所有文件
    # 2.判断是文件 or 文件夹
    # 3.判断每个文件是否有 .sh 扩展名
    # 4.没有.sh 的 file 进行添加
    for name in `ls ./`
     do
      if [ -f $name ]
       then
        fileLen=${#name}
        kzName=${name:fileLen-3:fileLen}
        #echo $kzName
        if [ $kzName == '.sh' ]
         then
          echo 当前的文件名称是：$name
        else
          mv ${name} ${name}.sh
        fi
```

```
    fi
done
```

11、函数

11.1 #————— 自定义函数

```
#!/bin/bash
# 自定义函数 function 可有可无
function demoFun(){ # 函数定义
  echo "this is fun"   # 函数语句
}
echo "start function"
demoFun     # 函数调用
echo "end function"
```

11.2 #————— 函数参数传递　函数名前的 function 可有可无

```
#!/bin/bash
# 自定义函数　参数传递
demoFun(){
  echo "argument 1....." $1
  echo "argument 2....." $2
  echo "argument all....." $@
}
echo "start function"
# 参数之间　空格 分割
demoFun ^-^ -_-      # 参数传递
echo "end function"
```

1. 输出重定向

需要注意的是文件描述符

0 通常是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）

将 ls -al 显示的详细内容 重定向输入到 tt 文件中

ls 可有替换成任何命令，结果都可以通过 > 重定向到指定的文件 文件名自定义可以不存在

> 覆盖重定向

```
[root@hadoop-1 testShell]# ls -al > tt
```

>> 追加重定向

```
[root@hadoop-1 testShell]# date >> tt
```

其他重定向 2>>&1 2> 2>>

重定向 符号 前边加上错误级别 **2>> 2>** 可以将错误的提示信息 出入到文件

[root@hadoop-1 testShell]# cattt tt **2>>** tt

[root@hadoop-1 testShell]# cattt tt **2>** tt

正确和错误都希望重定向 **2>&1 2>>&1**

正确 追加

[root@hadoop-1 testShell]# ls >> zz **2>&1**

错误 追加

[root@hadoop-1 testShell]# lssss >> zz 2>&1

正确 覆盖

[root@hadoop-1 testShell]# ls > zz 2>&1

错误 覆盖

[root@hadoop-1 testShell]# ls > zz 2>&1

2. 补充

举例：httpd 服务的应用： （PHP 有关）

[root@hadoop-1 ~]# service **httpd** start

访问：http://192.168.137.101/

可以看到 ： Apache 2 Test Page - powered by CentOS

#———— **let** 计算

```
a=2
echo "a init is $a"
let "a+=1"
echo "a+=1 is $a"
```

运行结果：

```
a init is 2
a+=1 is 3
```

#———— 比较符号

-eq 等于 -ne 不等于 -gt 大于 -lt 小于 -ge 大于等于 -le 小于等于 -a 与（and） -o 或（or）

[condition] && action; #如果 conditon 为真，则执行 acation；

[conditon] || action; #如果 condition 为假，执行 action。

#———— 字符串比较

文件系统相关测试：

[-f $file_var] 如果给定的变量包含正常的文件路径或者文件名，返回真

[-x $var] 如果变量包含的文件可执行，返回真

[-d $var] var 包含的是目录，返回真

[-e $var] var 包含的 文件存在，返回真
[-c $var] var 包含的是一个字符设备文件的路径，返回真
[-b $var] var 包含的是一个块设备文件的路径，返回真
[-w $var] var 包含的文件可写，返回真
[-r $var] var 包含的 文件可读，返回真
[-L $var] var 包含的是一个符号链接，返回真

#————— 作业练习

1. for 循环 1-100
2. while 计算 1-100 的和
3. case 打印 给定 路径的 所有文件名称
4. 通过函数 实现 计算器 加减乘除计算

------ day09 ------

1.动态的 增/删 节点

1.1 ————操作前准备新节点

集群启动的状态中，添加一台新的机器（节点）进入；

第一次三台机器加入集群，是通过 slaves 中添加的。

这里创建一个新的 hadoop4 号机器，启动服务，稍后动态添加。

1.1.1 克隆一台新机器：hadoop4;

————注：（一定注意，克隆中的集群生成的 data 和 name 文件夹要删除）

配置 hadoop4 IP，修改主机名称，增加 hosts 名称。

hosts 里边有四台机器，发送给原来 hadoop1，hadoop2，hadoop3

配置 hadoop4 的 ssh 免秘钥认证：分别发给四台机器；

1.1.2 登录 1/2/3 号机器分别设置 给 hadoop4 的免秘钥认证；

1.1.3 启动正常集群：只包含 1/2/3 机器（dfs/yarn）

1.2 ————动态增加节点

1.2.1 找到主机的 slaves 配置文件

cd /usr/soft/hadoop-2.7.1/etc/hadoop

slaves 中添加 hadoop4 号机器（要添加的节点名称）

[root@hadoop-1 hadoop]# vi slaves

1.2.2 手动刷新节点 （**主机上**）

hdfs dfsadmin -refreshNodes

1.2.3 第**四**台机器启动 datanode (在 4 号机器执行)

hadoop-daemon.sh start datanode

1.2.4 刷新 50070 页面 看连接 datanode 数

1.2.5 第**四**台机器启动 nodemanager （会自动加入集群）

yarn-daemon.sh start nodemanager

集群中 datanode 和 nodemanager 是相伴的，所以启动 nodemanager 之后节点会自动加入 yarn

1.3 ————动态删除节点

1.3.1 提前创建好自定义路径的 exclude 空文件（用于保存要删除的节点机名称）

/usr/soft/exclude

1.3.2 配置动态删除 datanode

修改 每台机器的 hdfs-site.xml

```
<property>
  <name>dfs.hosts.exclude</name>
  <value>/usr/soft/exclude</value>
```

</property>

1.3.3 配置动态删除 namemanager

修改 每台机器的 yarn-site.xml

<property>

<name>yarn.resourcemanager.nodes.exclude-path</name>

<value>/usr/soft/exclude</value>

</property>

1.3.4 启动集群

start-dfs.sh

start-yarn.sh

1.3.5 需要动态删除哪个节点机器

a)对应的在主机的 /usr/soft/exclude 文件中添加哪个机器名称

b)刷新 操作 即可

比如删除三号机器

1） echo "hadoop-3" > exclude

2） hadoop dfsadmin -refreshNodes

3） yarn rmadmin -refreshNodes 刷新 NodeManager

1.3.6 查看 50070 看到 3 号节点状态 变化 后去可以删除

hadoop-3:50010 (192.168.137.103:50010) 0 Decommission In Progress 17.23 GB 932 KB 5.96 GB 11.27 GB 20 932 KB (0.01%) 0 2.7.1

————8088 查看 已经少了一个节点。

1.3.7 删除节点操作 在 50070 不再显示

————去 3 号机器 停止 datenode 和 nodemanager

hadoop-daemon.sh stop datanode

yarn-daemon.sh stop nodemanager

————50070 查看 3 号节点 时间越来越长 目前 97

hadoop-3:50010 (192.168.137.103:50010) 97 Decommissioned 17.23 GB 932 KB 5.96 GB 11.27 GB 20 932 KB (0.01%) 0 2.7.1

————当时间超过 10 分钟没有响应会被关闭

————安全状态如下 Decommissioned

Node Last contact Admin State Capacity Used Non DFS Used Remaining Blocks Block pool usedFailed Volumes Version

hadoop-3:50010 (192.168.137.103:50010) Fri Dec 14 2018 15:13:50 GMT+0800 (中国标准时间) Dead, Decommissioned

- - - - - - - -

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
hadoop-1:50010 (192.168.137.101:50010)	0	In Service	17.23 GB	932 KB	6.46 GB	10.77 GB	20	932 KB (0.01%)	0	2.7.1
hadoop-2:50010 (192.168.137.102:50010)	1	In Service	17.23 GB	932 KB	5.96 GB	11.27 GB	20	932 KB (0.01%)	0	2.7.1
hadoop-4:50010 (192.168.137.104:50010)	1	In Service	17.23 GB	932 KB	6.44 GB	10.79 GB	20	932 KB (0.01%)	0	2.7.1
hadoop-3:50010 (192.168.137.103:50010)	Fri Dec 14 2018 15:13:50 GMT+0800 (中国标准时间)	Dead, Decommissioned	-	-	-	-	-	-	-	-

当前状态之后，可以安全的进行 3 号机器 关机操作，和历史文件查看操作等。

1.4 还原 3 号节点的操作两种

生产环境：（真实工作环境服务器从不轻易停机）

3 号不能再复原回到集群，exclude 中只添加删除的机器，不做修改操作；
进群只能新节点添加，或者变更 3 号为其他节点在走动态添加集群流程。

学习中环境：

注释 exclude 中主机名称。
在 3 号机器重启服务；
hadoop-daemon.sh start datanode
yarn-daemon.sh start nodemanager

可以注释掉 hdfs.site.xml 中 删除配置属性，
注释 yarn-site.xml 中 删除配置属性，
重启服务即可 3 号回到集群；

------ day10 ------

1. 周考

1.1. DataNode 与 NameNode 心跳周期？ 3 秒
1.2. 多久没有心跳认为节点丢失（损坏）？ 10 分钟
1.3. 块报告多长时间发送一次？ 1 小时 (3600 秒)
1.4. 谁负责维护 NameNode（fsimage、editlog）？ SecondaryNameNode
1.5. 安全模式的作用？如何和功手动关闭安全模式？
安全模式：hadoop 的一种保护机制，用于保证集群中的数据块的安全性；（开机检验、副本异常修复）
hadoop dfsadmin -safemode leave
hadoop dfsadmin -safemode enter
1.6. MR 中 uber 模式的作用？聚合日志的作用？
节省 jvm 开启关闭的资源消耗；
聚合日志：日志分布各个节点，为了方便查看，配置聚合日志，可以 web 查看。
1.7. 开启/关闭历史日志的命令？
mr-jobhistory-deamon.sh start historyserver
mr-jobhistory-deamon.sh stop historyserver
1.8. HDFS 读数据流程？
client --> NameNode ---> DataNode
1.9. HDFS 写数据的流程？
client --> DataNode
1.10 单独开启一个 DataNode 的命令？单独开启一个 NodeManager 的命令？
start-dfs.sh --> DataNode
start-yarn.sh --> NodeManager

2. 删除一台机器(节点)

2.1 修改文件

hosts
slaves
hdfs-site.xml
yarn-site.xml
/usr/soft/exclude

2.2 重启服务

3. yarn 分布资源管理框架理论知识

3.1 长短应用

长/短应用程序 365 天*24 小时 长期运行 ： 长应用程序

一个计算，无论时间长短，都会停止 ： 短应用程序

3.2 yarn 的组件

client；

ResourceManager、Application Master

NodeManager、Container

3.3 yarn 通信协议

4. ZooKeeper 安装配置

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现，是 Hadoop 和 Hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

数据维护与同步；

4.1 上传 ZooKeeper

到服务器安装路径（创建 myzookeeper 文件夹）

上传：/usr/soft/myzookeeper/zookeeper-3.4.7.tar.gz

解压：[root@hadoop-1 zookeeper-3.4.7]# tar -zxvf zookeeper-3.4.7.tar.gz

4.2 修改 ZooKeeper 配置文件

更改操作目录： /usr/soft/myzookeeper/zookeeper-3.4.7/conf

拷贝一份配置文件：[root@hadoop-1 conf]# cp zoo_sample.cfg zoo.cfg

修改 **zoo.cfg**；（文件名是固定的）

dataDir 修改成自己的路径： 确保 data 文件夹存在

dataDir=/usr/soft/myzookeeper/data

add your self master pc

2888 集群通讯端口

3888 如果 leader 宕机，新 leader 选取端口

server.1=hadoop-1:2888:3888

server.2=hadoop-2:2888:3888

server.3=hadoop-3:2888:3888

4.3 设置 data 文件（dataDir 对应文件夹）

在 data 下创建 myid 文件内容保存 1（这里的 1 和 zoo.cfg 配置文件中 server. 后边的 id 对应）

[root@hadoop-1 data]# echo 1 > myid

4.4 同步其他节点机器

myzookeeper 分别发给 2/3 机器

[root@hadoop-1 soft]# scp -r ./myzookeeper/ hadoop-2:/usr/soft/myzookeeper

[root@hadoop-1 soft]# scp -r ./myzookeeper/ hadoop-3:/usr/soft/myzookeeper

分别修改 myid 为对应机器的序号

修改 hadoop-2: 中 data 下 myid 内容修改为 2：echo 2 > myid

修改 hadoop-3: 中 data 下 myid 内容修改为 3：echo 3 > myid

4.5 配置环境变量

找到 zookeeper 解压路径：

/usr/soft/myzookeeper/zookeeper-3.4.7

编辑环境变量文件：vi /etc/profile

加入以下对应内容 （注意保留 PATH 原有内容、使用冒号连接）

```
#zookeeper
export ZOOKEEPER_HOME=/usr/soft/myzookeeper/zookeeper-3.4.7
#system
export PATH=:$ZOOKEEPER_HOME/bin:$PATH:
```

生效配置文件：source /etc/profile

同步配置文件到 hadoop-2/hadoop-3 节点，并使配置文件生效；

4.6 验证启动

（zookeeper 要求半数以上集群启动才会显示 follower 状态）节点显示 leader

[root@hadoop-1 soft]# zkServer.sh start

```
ZooKeeper JMX enabled by default
Using config: /usr/soft/myzookeeper/zookeeper-3.4.7/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
```

去 2 号机器启动：zkServer.sh start

[root@hadoop-2 data]# zkServer.sh status

```
ZooKeeper JMX enabled by default
Using config: /usr/soft/myzookeeper/zookeeper-3.4.7/bin/../conf/zoo.cfg
Mode: leader
```

查询 1 号机器状态

[root@hadoop-1 soft]# zkServer.sh status

```
ZooKeeper JMX enabled by default
Using config: /usr/soft/myzookeeper/zookeeper-3.4.7/bin/../conf/zoo.cfg
Mode: follower
```

4.7 功能演示

实时的数据同步

zk 客户端命令

ZooKeeper 命令行工具类似于 Linux 的 shell 环境，不过功能肯定不及 shell 啦，但是使用它我们可以简

单的对 ZooKeeper 进行访问，数据创建，数据修改等操作. 使用 zkCli.sh -server 127.0.0.1:2181 连接到 ZooKeeper 服务，连接成功后，系统会输出 ZooKeeper 的相关环境以及配置信息。

命令行工具的一些简单操作如下：

1. 显示根目录下、文件： ls / 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容
2. 显示根目录下、文件： ls2 / 查看当前节点数据并能看到更新次数等数据
3. 创建文件，并设置初始内容： create /zk "test" 创建一个新的 znode 节点“ zk ”以及与它关联的字符串
4. 获取文件内容： get /zk 确认 znode 是否包含我们所创建的字符串
5. 修改文件内容： set /zk "zkbak" 对 zk 所关联的字符串进行设置
6. 删除文件： delete /zk 将刚才创建的 znode 删除
   rmr /节点名称 删除非空节点
7. 退出客户端： quit
8. 帮助命令： help

进入 zoo 命令模式

```
[root@hadoop-1 conf]# zkCli.sh
```

参考：

[ZooKeeper 客户端命令-参考](https://blog.csdn.net/ccorg/article/details/85051321)

[ZooKeeper 的参考链接](https://www.apache.org/dyn/closer.cgi/zookeeper/)

[zookeeper-3.4.7.tar.gz](https://download.csdn.net/download/flyingsir_zw/10855702)

ZooKeeper 是学习：kafuka、hbase、高可用的基础

------ day11 ------

————理论知识

YARN

YARN 是一个资源调度平台，负责为运算程序提供服务器运算资源，相当于一个分布式的操 作系统平台，而 MapReduce 等运算程序则相当于运行于操作系统之上的应用程序

ResourceManager

ResourceManager 是基于应用程序对集群资源的需求进行调度的 YARN 集群主控节点，负责 协调和管理整个集群（所有 NodeManager）的资源，响应用户提交的不同类型应用程序的 解析，调度，监控等工作。ResourceManager 会为每一个 Application 启动一个 MRAppMaster， 并且 MRAppMaster 分散在各个 NodeManager 节点

它主要由两个组件构成：调度器（Scheduler）和应用程序管理器（ApplicationsManager， ASM）

YARN 集群的主节点 ResourceManager 的职责

1、处理客户端请求

2、启动或监控 MRAppMaster

3、监控 NodeManager

4、资源的分配与调度

NodeManager

NodeManager 是 YARN 集群当中真正资源的提供者，是真正执行应用程序的容器的提供者， 监控应用程序的资源使用情况（CPU，内存，硬盘，网络），并通过心跳向集群资源调度器 ResourceManager 进行汇报以更新自己的健康状态。同时其也会监督 Container 的生命周期 管理，监控每个 Container 的资源使用（内存、CPU 等）情况，追踪节点健康状况，管理日 志和不同应用程序用到的附属服务（auxiliary service）。

YARN 集群的从节点 NodeManager 的职责

1、管理单个节点上的资源

2、处理来自 ResourceManager 的命令

3、处理来自 MRAppMaster 的命令

MRAppMaster

MRAppMaster 对应一个应用程序，职责是：向资源调度器申请执行任务的资源容器，运行 任务，监控整个任务的执行，跟踪整个任务的状态，处理任务失败以异常情况

Container

Container 容器是一个抽象出来的逻辑资源单位。容器是由 ResourceManager Scheduler 服务 动态分配的资源构成，它包括了该节点上的一定量 CPU，内存，磁盘，网络等信息，MapReduce 程序的所有 Task 都是在一个容器里执行完成的，容器的大小是可以动态调整的

ASM

应用程序管理器 ASM 负责管理整个系统中所有应用程序，包括应用程序提交、与调度器协 商资源以启动 MRAppMaster、监控 MRAppMaster 运行状态并在失败时重新启动它等

Scheduler

调度器根据应用程序的资源需求进行资源分配，不参与应用程序具体的执行和监控等工作 资源分配的单位就是 Container，调度器是一个可插拔的组件，用户可以根据自己的需求实 现自己的调度器。YARN 本身为我们提供了多种直接可用的调度器，比如 FIFO，Fair Scheduler 和 Capacity Scheduler 等

————zkJavaApi

创建 Maven 项目

配置 pom.xml

```
<!-- https://mvnrepository.com/artifact/org.apache.zookeeper/zookeeper -->
<dependency>
    <groupId>org.apache.zookeeper</groupId>
    <artifactId>zookeeper</artifactId>
    <version>3.4.6</version>
</dependency>
```

运行测试代码

```
import java.io.IOException;
import java.util.concurrent.CountDownLatch;

import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.Watcher.Event.KeeperState;
import org.apache.zookeeper.ZooKeeper;

public class Zookeeper_Constructor_Usage_Simple implements Watcher {
    private static CountDownLatch connectedSemaphore = new CountDownLatch(1);
    @Override
    public void process(WatchedEvent event) {
        System.out.println("Receive watched event : " + event);
```

```
                if (KeeperState.SyncConnected == event.getState()) {
                    connectedSemaphore.countDown();
                }
            }
            public static void main(String[] args) throws IOException {
                ZooKeeper zookeeper = new ZooKeeper("127.0.0.1:2181", 5000, new
Zookeeper_Constructor_Usage_Simple());
                System.out.println(zookeeper.getState());
                try {
                    connectedSemaphore.await();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                System.out.println("Zookeeper session established");
            }
        }
```

——————运行结果

```
log4j:WARN No appenders could be found for logger (org.apache.zookeeper.ZooKeeper).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
CONNECTING
```

——————zk Java Api

```
ZooKeeper zk = new ZooKeeper(connectString, sessionTimeout, watcher);
zk.create(path, data, acl, createMode);
zk.create(path, data, acl, createMode, cb, ctx);
zk.delete(path, version);
zk.delete(path, version, cb, ctx);
zk.setData(path, data, version);
zk.setData(path, data, version, cb, ctx);
zk.getData(path, watch, stat);
zk.getData(path, watch, cb, ctx);
zk.exists(path, watch);
zk.exists(path, watch, cb, ctx);
```

[源码参考](https://blog.csdn.net/ccorg/article/details/85070023)

——————补充

ACL

访问控制列表（Access Control List，ACL） 是路由器和交换机接口的指令列表，用来控制端口进出的

数据包。ACL 适用于所有的被路由协议，如 IP、IPX、AppleTalk 等。