**《大数据应用项目实践》课程设计**

**（2023/2024学年第二学期）**

**指导教师：夏劲松、霍戌文**

实

践

总

结

报

告

**计算机科学与技术（人工智能）学院**

**2024年05月浙江理工大学实践总结报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **陈昊天** | **性别** | 男 | **学号** | 2021329600006 |
| **分院** | 计算机科学与技术学院 | **专业** | 计算机科学与技术 | | |
| **实践**  **单位** | 浙江理工大学 | **实践**  **岗位** | 大数据应用项目实践 | **实践**  **时间** | 5月21日-6月23日 |
| **（一）实践主要内容及进程** （1）Linux操作系统与虚拟机 模块一 环境搭建  1 安装Hyper-V  在控制面板-程序-启用或关闭WIndows功能中启用Hyper-V    2 安装Debian 12  在 https://www.debian.org/download.zh-cn.html 下载Debian 12镜像，并创建虚拟机。    3 复制2台镜像机  导出虚拟机，然后再导入虚拟机      4 网络配置和系统管理操作  新建内部虚拟交换机（NAT），分配到3台虚拟机    设置虚拟交换机，手动设置IP地址    在3台虚拟机内分别手动设置IP地址为192.168.137.21，192.168.137.22，192.168.137.23    5 安装Termius远程登录  在虚拟机上安装openssh-server  apt update  apt install openssh-server  修改/etc/ssh/sshd\_config，使允许root登录  修改 PermitRootLogin 属性为 yes，随后重启ssh  随后在Termius上填写虚拟机IP地址、root密码即可远程登陆   （2）Linux常用命令与Shell编程  |  |  |  | | --- | --- | --- | | 功能 | 步骤 | 结果 | | 分区格式化和挂载 | 使用lsblk列出所有可用的块设备 | root@D12B1:~# lsblk NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS sda 8:0 0 127G 0 disk  |-sda1 8:1 0 512M 0 part /boot/efi |-sda2 8:2 0 125.5G 0 part / `-sda3 8:3 0 976M 0 part [SWAP] sdb 8:16 0 1G 0 disk | | 使用fdisk /dev/sdb创建分区 | root@D12B1:~# fdisk /dev/sdb  Welcome to fdisk (util-linux 2.38.1). Changes will remain in memory only, until you decide to write them. Be careful before using the write command.  Device does not contain a recognized partition table. Created a new DOS (MBR) disklabel with disk identifier 0xec07a6c4.  Command (m for help): g Created a new GPT disklabel (GUID: AA5EC750-BD2D-5A47-84BD-87304C4727C4).  Command (m for help): n Partition number (1-128, default 1):  First sector (2048-2097118, default 2048):  Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-2097118, default 2095103):   Created a new partition 1 of type 'Linux filesystem' and of size 1022 MiB.  Command (m for help): w The partition table has been altered. Calling ioctl() to re-read partition table. Syncing disks. | | 使用mkfs格式化分区 | root@D12B1:~# mkfs.ext4 /dev/sdb1 mke2fs 1.47.0 (5-Feb-2023) Discarding device blocks: done  Creating filesystem with 261632 4k blocks and 65408 inodes Filesystem UUID: 801282f5-1f3b-4022-8454-1dc589827c7c Superblock backups stored on blocks:   32768, 98304, 163840, 229376  Allocating group tables: done  Writing inode tables: done  Creating journal (4096 blocks): done Writing superblocks and filesystem accounting information: done | | 使用mount挂载分区 | root@D12B1:~# mkdir /mnt/mydisk root@D12B1:~# mount /dev/sdb1 /mnt/mydisk root@D12B1:~# df -h Filesystem Size Used Avail Use% Mounted on udev 919M 0 919M 0% /dev tmpfs 191M 980K 190M 1% /run /dev/sda2 124G 6.0G 111G 6% / tmpfs 951M 0 951M 0% /dev/shm tmpfs 5.0M 0 5.0M 0% /run/lock /dev/sda1 511M 5.9M 506M 2% /boot/efi tmpfs 191M 44K 190M 1% /run/user/0 tmpfs 191M 60K 190M 1% /run/user/113 /dev/sdb1 988M 24K 921M 1% /mnt/mydisk | | 进程线程类 | 使用ps aux | grep xxx 查看xxx相关进程 | root@D12B1:~# ps aux | grep gnome  Debian-+ 702 0.0 0.4 159644 8192 tty1 Ssl+ 14:37 0:00 /usr/libexec/gdm-wayland-session dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart Debian-+ 708 0.0 0.0 6260 1492 tty1 S+ 14:37 0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart Debian-+ 714 0.0 1.1 520956 22372 tty1 Sl+ 14:37 0:00 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart Debian-+ 747 0.2 12.4 5267812 242040 tty1 Sl+ 14:37 0:02 /usr/bin/gnome-shell Debian-+ 877 0.0 1.4 2793204 29104 tty1 Sl+ 14:37 0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.Notifications Debian-+ 880 0.0 0.4 164312 9444 tty1 Sl+ 14:37 0:00 /usr/libexec/at-spi2-registryd --use-gnome-session Debian-+ 1101 0.0 1.6 2727652 31480 tty1 Sl+ 14:37 0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.ScreenSaver root 1175 0.0 0.0 3324 1444 pts/0 S+ 14:56 0:00 grep gnome | | 使用pkill终止进程 | root@D12B1:~# tmux [detached (from session 0)] root@D12B1:~# ps aux | grep tmux root 1181 0.0 0.2 9044 3924 ? Ss 14:58 0:00 tmux root 1189 0.0 0.0 3324 1408 pts/0 S+ 14:59 0:00 grep tmux root@D12B1:~# pkill tmux root@D12B1:~# ps aux | grep tmux root 1195 0.0 0.0 3324 1488 pts/0 S+ 14:59 0:00 grep tmux | | 使用lsof -i:端口号 查看端口占用情况 | root@D12B1:~# lsof -i:22 COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME sshd 612 root 3u IPv4 19913 0t0 TCP \*:ssh (LISTEN) sshd 612 root 4u IPv6 19915 0t0 TCP \*:ssh (LISTEN) sshd 648 root 4u IPv4 15136 0t0 TCP D12B1:ssh->9900K:62018 (ESTABLISHED) | | 目录和文件操作 | pwd 显示当前工作目录的绝对路径 | root@D12B1:~# pwd  /root | | ls 列出目录的内容 | root@D12B1:/# ls -a . bin etc initrd.img.old lost+found opt run sys var .. boot home lib media proc sbin tmp vmlinuz .cache dev initrd.img lib64 mnt root srv usr vmlinuz.old root@D12B1:/# | | cd 切换目录 | root@D12B1:/# cd /etc root@D12B1:/etc# | | mkdir 创建一个新的目录 | root@D12B1:~# mkdir test root@D12B1:~# ls test root@D12B1:~# | | rmdir 删除一个空的目录 | root@D12B1:~# rmdir test root@D12B1:~# ls root@D12B1:~# | | touch 创建空文件 | root@D12B1:~# touch 1.txt root@D12B1:~# ls 1.txt root@D12B1:~# | | cp 复制文件或目录 | root@D12B1:~# cp 1.txt 2.txt root@D12B1:~# ls 1.txt 2.txt | | rm 移除文件或目录 | root@D12B1:~# rm 1.txt root@D12B1:~# ls 2.txt | | mv 移动文件与目录或重命名 | root@D12B1:~# mv 2.txt 3.txt root@D12B1:~# ls 3.txt root@D12B1:~# | | cat 查看文件内容 | root@D12B1:~# echo 123 >> 3.txt root@D12B1:~# cat 3.txt 123 | | more 文件内容分屏查看器 | root@D12B1:~# more 3.txt | | less 分屏显示文件内容 | root@D12B1:~# less 3.txt | | echo输出内容到控制台 | root@D12B1:~# echo 123  123 | | head 显示文件头部内容 | root@D12B1:~# head 3.txt 123 | | tail 输出文件尾部内容 | root@D12B1:~# tail 3.txt 123 | | > 输出重定向和 >> 追加 | root@D12B1:~# echo 123456 > 3.txt root@D12B1:~# cat 3.txt 123456 root@D12B1:~# echo 666 >> 3.txt root@D12B1:~# cat 3.txt 123456 666 root@D1 | | ln 软链接 | root@D12B1:~# ln -s 3.txt 4.txt root@D12B1:~# ls 3.txt 4.txt root@D12B1:~# cat 4.txt 123456 666 root@D12B1:~# | | history 查看已经执行过历史命令 | root@D12B1:~# history  1 cd  2 apt install openssh-server  3 vim /etc/ssh/sshd\_config  4 apt install vim  5 vim /etc/ssh/sshd\_config  6 systemctl restart ssh  7 uname -a  8 uptime  9 tmux  10 apt install tmux  11 tmux | | 时间日期类 | date 显示当前时间 | root@D12B1:~# date Sat Mar 9 15:33:34 CST 2024 | | ncal 查看日历 | root@D12B1:~# ncal  March 2024  Su 3 10 17 24 31 Mo 4 11 18 25  Tu 5 12 19 26  We 6 13 20 27  Th 7 14 21 28  Fr 1 8 15 22 29  Sa 2 9 16 23 30 | | 用户管理命令 | useradd 添加新用户 | root@D12B1:~# useradd test | | passwd 设置用户密码 | root@D12B1:~# passwd test New password:  Retype new password:  passwd: password updated successfully | | id 查看用户是否存在 | root@D12B1:~# id test uid=1001(test) gid=1001(test) groups=1001(test) | | cat /etc/passwd 查看创建了哪些用户 | root@D12B1:~# cat /etc/passwd  root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin \_apt:x:42:65534::/nonexistent:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin tss:x:100:107:TPM software stack,,,:/var/lib/tpm:/bin/false systemd-timesync:x:997:997:systemd Time Synchronization:/:/usr/sbin/nologin messagebus:x:101:108::/nonexistent:/usr/sbin/nologin usbmux:x:102:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin sshd:x:103:65534::/run/sshd:/usr/sbin/nologin dnsmasq:x:104:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin avahi:x:105:112:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin speech-dispatcher:x:106:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false fwupd-refresh:x:107:115:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin saned:x:108:117::/var/lib/saned:/usr/sbin/nologin geoclue:x:109:118::/var/lib/geoclue:/usr/sbin/nologin polkitd:x:996:996:polkit:/nonexistent:/usr/sbin/nologin rtkit:x:110:119:RealtimeKit,,,:/proc:/usr/sbin/nologin colord:x:111:120:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin gnome-initial-setup:x:112:65534::/run/gnome-initial-setup/:/bin/false Debian-gdm:x:113:121:Gnome Display Manager:/var/lib/gdm3:/bin/false vps:x:1000:1000:vps,,,:/home/vps:/bin/bash test:x:1001:1001::/home/test:/bin/sh | | su 切换用户 | root@D12B1:~# su test $ | | userdel 删除用户 | root@D12B1:~# userdel test | | who 查看登录用户信息 | root@D12B1:~# who am i root pts/0 Mar 9 14:37 (192.168.2.12) root@D12B1:~# whoami root | | sudo 设置普通用户具有root权限 | root@D12B1:~# useradd bigdata root@D12B1:~# passwd bigdata New password:  Retype new password:  passwd: password updated successfully root@D12B1:~# vi /etc/sudoers root@D12B1:~# su bigdata $ pwd  /root $ sudo touch 1.txt $ | | usermod 修改用户（modify） | root@D12B1:~# useradd test root@D12B1:~# usermod -g root test | | 用户组管理命令 | groupadd 新增组 | root@D12B1:~# groupadd testgroup | | groupdel 删除组 | root@D12B1:~# groupdel testgroup | | groupmod 修改组 | root@D12B1:~# groupmod -n ttt testgroup | | cat /etc/group 查看创建了哪些组 | root@D12B1:~# cat /etc/group root:x:0: daemon:x:1: bin:x:2: sys:x:3: adm:x:4: tty:x:5: disk:x:6: lp:x:7: mail:x:8: news:x:9: uucp:x:10: man:x:12: proxy:x:13: kmem:x:15: dialout:x:20: fax:x:21: voice:x:22: cdrom:x:24:vps floppy:x:25:vps tape:x:26: sudo:x:27: audio:x:29:vps dip:x:30:vps www-data:x:33: backup:x:34: operator:x:37: list:x:38: irc:x:39: src:x:40: shadow:x:42: utmp:x:43: video:x:44:vps sasl:x:45: plugdev:x:46:vps staff:x:50: games:x:60: users:x:100:vps nogroup:x:65534: systemd-journal:x:999: systemd-network:x:998: crontab:x:101: input:x:102: sgx:x:103: kvm:x:104: render:x:105: netdev:x:106:vps tss:x:107: systemd-timesync:x:997: messagebus:x:108: \_ssh:x:109: ssl-cert:x:110: bluetooth:x:111:vps avahi:x:112: lpadmin:x:113:vps pipewire:x:114: fwupd-refresh:x:115: scanner:x:116:saned,vps saned:x:117: geoclue:x:118: polkitd:x:996: rtkit:x:119: colord:x:120: Debian-gdm:x:121: vps:x:1000: gnome-initial-setup:x:995: bigdata:x:1001: test:x:1002: ttt:x:1003: | | 文件权限类 | 文件属性 | root@D12B1:~# ls -l total 4 -rw-r--r-- 1 root root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt | | chmod 改变权限 | root@D12B1:~# chmod 777 1.txt root@D12B1:~# ls -l total 4 -rwxrwxrwx 1 root root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt | | chown 改变所有者（属主） | root@D12B1:~# chown bigdata 1.txt root@D12B1:~# ls -l total 4 -rwxrwxrwx 1 bigdata root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt | | chgrp 改变所属组 | root@D12B1:~# chgrp root 1.txt root@D12B1:~# ls -l total 4 -rwxrwxrwx 1 bigdata root 0 Mar 9 15:45 1.txt -rw-r--r-- 1 root root 11 Mar 9 15:31 3.txt lrwxrwxrwx 1 root root 5 Mar 9 15:32 4.txt -> 3.txt | | 搜索查找类 | find 查找文件或者目录 | root@D12B1:~# find ./ -name 1.txt ./1.txt | | locate快速定位文件路径 | root@D12B1:~# updatedb /usr/bin/find: '/run/user/113/gvfs': Permission denied root@D12B1:~# locate 1.txt /home/vps/.mozilla/firefox/y1a0d2u1.default-esr/pkcs11.txt /root/1.txt /usr/share/doc/debian/constitution.1.1.txt.gz /usr/share/doc/debian/social-contract.1.1.txt.gz /usr/share/doc/kbd/charsets/iso8859-1.txt.gz /usr/share/doc/kbd/charsets/iso8859-11.txt.gz /usr/share/doc/libsane-common/teco/teco1.txt.gz /usr/share/vim/vim90/doc/ft\_ps1.txt /usr/share/vim/vim90/doc/gui\_x11.txt /usr/share/vim/vim90/doc/usr\_01.txt /usr/share/vim/vim90/doc/usr\_11.txt /usr/share/vim/vim90/doc/usr\_21.txt /usr/share/vim/vim90/doc/usr\_31.txt /usr/share/vim/vim90/doc/usr\_41.txt /usr/share/vim/vim90/doc/usr\_51.txt | | grep过滤查找及“|”管道符 | root@D12B1:~# ls | grep -n 1.txt 1:1.txt root@D12B1:~# | | 压缩和解压类 | gzip/gunzip 压缩 | root@D12B1:~# gzip 1.txt root@D12B1:~# ls 1.txt.gz 3.txt 4.txt root@D12B1:~# gunzip 1.txt.gz  root@D12B1:~# ls 1.txt 3.txt 4.txt | | zip/unzip 压缩 | root@D12B1:~# rm -rf ./\* root@D12B1:~# touch 1.txt root@D12B1:~# touch 2.txt root@D12B1:~# zip 3.zip 1.txt 2.txt  adding: 1.txt (stored 0%)  adding: 2.txt (stored 0%) root@D12B1:~# ls 1.txt 2.txt 3.zip root@D12B1:~# unzip 3.zip Archive: 3.zip replace 1.txt? [y]es, [n]o, [A]ll, [N]one, [r]ename: A  extracting: 1.txt   extracting: 2.txt  root@D12B1:~# | | tar 打包 | root@D12B1:~# tar -zcvf all.tar.gz ./\* ./1.txt ./2.txt ./3.zip root@D12B1:~# ls 1.txt 2.txt 3.zip all.tar.gz root@D12B1:~# rm 1.txt 2.txt 3.zip  root@D12B1:~# tar -zxvf all.tar.gz  ./1.txt ./2.txt ./3.zip root@D12B1:~# ls 1.txt 2.txt 3.zip all.tar.gz root@D12B1:~# | | 系统定时任务 | crond 服务管理 | root@D12B1:~# service cron restart | | crontab 定时任务设置 | root@D12B1:~# crontab -e  no crontab for root - using an empty one  Select an editor. To change later, run 'select-editor'.  1. /bin/nano <---- easiest  2. /usr/bin/vim.basic  3. /usr/bin/vim.tiny  Choose 1-3 [1]: 2 crontab: installing new crontab | | apt软件包管理 | apt update更新软件包列表 | root@D12B1:~# apt update Hit:1 http://mirrors.ustc.edu.cn/debian bookworm InRelease Hit:2 http://mirrors.ustc.edu.cn/debian bookworm-updates InRelease  Hit:3 http://security.debian.org/debian-security bookworm-security InRelease  Reading package lists... Done Building dependency tree... Done Reading state information... Done All packages are up to date. | | apt upgrade升级软件包 | root@D12B1:~# apt upgrade Reading package lists... Done Building dependency tree... Done Reading state information... Done Calculating upgrade... Done 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded. | | apt install安装新的软件包 | root@D12B1:~# apt install nginx Reading package lists... Done Building dependency tree... Done Reading state information... Done The following additional packages will be installed:  nginx-common Suggested packages:  fcgiwrap nginx-doc The following NEW packages will be installed:  nginx nginx-common 0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded. Need to get 640 kB of archives. After this operation, 1696 kB of additional disk space will be used. Do you want to continue? [Y/n]  Get:1 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 nginx-common all 1.22.1-9 [112 kB] Get:2 http://mirrors.ustc.edu.cn/debian bookworm/main amd64 nginx amd64 1.22.1-9 [527 kB] Fetched 640 kB in 0s (4555 kB/s) Preconfiguring packages ... Selecting previously unselected package nginx-common. (Reading database ... 157709 files and directories currently installed.) Preparing to unpack .../nginx-common\_1.22.1-9\_all.deb ... Unpacking nginx-common (1.22.1-9) ... Selecting previously unselected package nginx. Preparing to unpack .../nginx\_1.22.1-9\_amd64.deb ... Unpacking nginx (1.22.1-9) ... Setting up nginx-common (1.22.1-9) ... Created symlink /etc/systemd/system/multi-user.target.wants/nginx.service -> /lib/systemd/system/nginx.service. Setting up nginx (1.22.1-9) ... Upgrading binary: nginx. Processing triggers for man-db (2.11.2-2) ... root@D12B1:~# | | apt remove卸载软件包 | root@D12B1:~# apt remove nginx Reading package lists... Done Building dependency tree... Done Reading state information... Done The following package was automatically installed and is no longer required:  nginx-common Use 'apt autoremove' to remove it. The following packages will be REMOVED:  nginx 0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded. After this operation, 1362 kB disk space will be freed. Do you want to continue? [Y/n]  (Reading database ... 157759 files and directories currently installed.) Removing nginx (1.22.1-9) ... Processing triggers for man-db (2.11.2-2) ... root@D12B1:~# | | apt purge完全卸载软件包 | root@D12B1:~# apt purge nginx Reading package lists... Done Building dependency tree... Done Reading state information... Done Package 'nginx' is not installed, so not removed The following package was automatically installed and is no longer required:  nginx-common Use 'apt autoremove' to remove it. 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded. | | apt autoremove自动删除不再需要的软件包 | root@D12B1:~# apt autoremove Reading package lists... Done Building dependency tree... Done Reading state information... Done The following packages will be REMOVED:  nginx-common 0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded. After this operation, 334 kB disk space will be freed. Do you want to continue? [Y/n]  (Reading database ... 157753 files and directories currently installed.) Removing nginx-common (1.22.1-9) ... root@D12B1:~# | | Shell编程 | helloworld | root@D12B1:~# touch helloworld.sh root@D12B1:~# vim helloworld.sh  root@D12B1:~# chmod +x helloworld.sh  root@D12B1:~# ./helloworld.sh  Hello, world! | | 位置参数变量 | #!/bin/bash greeting="Hello" echo $greeting echo "Positional Parameters:" echo "1st: $1" echo "2nd: $2" echo "3rd: $3" echo "All positional parameters: $\*" echo "Number of positional parameters: $#" | | 运算符 | root@D12B1:~# S=$[(2+3)\*4] root@D12B1:~# echo $S 20 | | 判断语句 | root@D12B1:~# ls 1.sh 1.txt 2.txt 3.zip all.tar.gz helloworld.sh root@D12B1:~# [ 1.txt ] && echo true true | | 流程控制 | #!/bin/bash  if [ $1 -eq "123" ] then  echo "123" elif [ $1 -eq "456" ] then   echo "456" else  echo "789" fi | | case 语句 | #!/bin/bash  case $1 in "1")  echo "1" ;;  "2")  echo "2" ;; \*)  echo "other" ;; esac | | for 循环 | #!/bin/bash for i in "$\*"  do  echo "The num is $i "  done  for j in "$@"  do  echo "The num is $j"  done | | while 循环 | #!/bin/bash s=0 i=1 while [ $i -le 100 ] do  s=$[$s+$i]  i=$[$i+1] done  echo $s | | read读取控制台输入 | #!/bin/bash  read -t 7 -p "please 7 miao input your name " NAME echo $NAME | | 系统函数 | root@D12B1:~# basename /root/1.txt 1.txt | | 自定义函数 | #!/bin/bash function sum() {  s=0  s=$[ $1 + $2 ]  echo "$s" }  read -p "Please input the number1: " n1; read -p "Please input the number2: " n2; sum $n1 $n2; |  （3）Hadoop HDFS集群环境配置 1 在所有机器安装 JDK8  使用 Adoptium 源安装 temurin-8-jdk  sudo apt-get install -y wget apt-transport-https gnupg wget -O - https://packages.adoptium.net/artifactory/api/gpg/key/public | sudo apt-key add - echo "deb https://packages.adoptium.net/artifactory/deb $(lsb\_release -sc) main" | sudo tee /etc/apt/sources.list.d/adoptium.list sudo apt-get update sudo apt-get install temurin-8-jdk    2 SSH、环境变量、hosts、主机名设置  vim /etc/ssh/sshd\_config  修改PermitRootLogin所在行为PermitRootLogin yes    环境变量  echo 'export JAVA\_HOME=/usr/lib/jvm/temurin-8-jdk-amd64' >> ~/.bashrc echo 'export HADOOP\_HOME=/usr/local/hadoop' >> ~/.bashrc echo 'export PATH=$PATH:$HADOOP\_HOME/bin:$HADOOP\_HOME/sbin' >> ~/.bashrc source ~/.bashrc    hosts 文件  vim /etc/hosts  所有机器文件内容一致。  127.0.0.1 localhost 10.0.3.2 node1 10.0.3.3 node2 10.0.3.4 node3    主机名  分别修改3台虚拟机的主机名  sudo hostnamectl set-hostname node1 sudo hostnamectl set-hostname node2 sudo hostnamectl set-hostname node3  修改后reboot重启    3 在所有机器下载 Hadoop  cd wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz sudo tar zxvf hadoop-3.3.6.tar.gz -C /usr/local/ sudo mv /usr/local/hadoop-3.3.6 /usr/local/hadoop rm hadoop-3.3.6.tar.gz    4 在 node1 配置 SSH 免密登录  在 NameNode 上执行。  编辑~/.ssh/config文件  vim ~/.ssh/config  填入以下内容：  Host node1 HostName 10.0.3.2 Port 22 User root  Host node2 HostName 10.0.3.3 Port 22 User root  Host node3 HostName 10.0.3.4 Port 22 User root  生成密钥并拷贝到3台机器上  ssh-keygen -t rsa -P ''  ssh-copy-id node1 ssh-copy-id node2 ssh-copy-id node3  5 在所有机器配置 Hadoop  hadoop-env.sh  vim $HADOOP\_HOME/etc/hadoop/hadoop-env.sh  在文件末尾追加  export JAVA\_HOME=/usr/lib/jvm/temurin-8-jdk-amd64  export HDFS\_NAMENODE\_USER=root  export HDFS\_DATANODE\_USER=root  export HDFS\_SECONDARYNAMENODE\_USER=root  export YARN\_RESOURCEMANAGER\_USER=root  export YARN\_NODEMANAGER\_USER=root  core-site.xml  vim $HADOOP\_HOME/etc/hadoop/core-site.xml  在configuration标签中添加以下内容  <!-- 默认文件系统的名称。通过URI中schema区分不同文件系统 -->  <!-- file://本地文件系统 hdfs://hadoop分布式文件系统 -->  <!-- gfs://google文件系统 -->  <!-- hdfs文件系统访问地址：http://node1:8020 -->  <property>  <name>fs.defaultFS</name>  <value>hdfs://node1:8020</value>  </property>  <!-- 设置Hadoop本地保存数据路径 -->  <property>  <name>hadoop.tmp.dir</name>  <value>/usr/local/hadoop/tmp</value>  </property>  <!-- 设置HDFS web UI用户身份 -->  <property>  <name>hadoop.http.staticuser.user</name>  <value>root</value>  </property>  hdfs-site.xml  vim $HADOOP\_HOME/etc/hadoop/hdfs-site.xml  在configuration标签中添加以下内容  <property>  <name>dfs.replication</name>  <value>3</value>  </property>  <property>  <name>dfs.namenode.secondary.http-address</name>  <value>node1:50090</value>  </property>  mapred-site.xml  vim $HADOOP\_HOME/etc/hadoop/mapred-site.xml  在configuration标签中添加以下内容  <!-- 设置MR程序默认运行模式： yarn集群模式 local本地模式 -->  <property>  <name>mapreduce.framework.name</name>  <value>yarn</value>  </property>  <!-- MR程序历史服务地址 -->  <property>  <name>mapreduce.jobhistory.address</name>  <value>node1:10020</value>  </property>  <!-- MR程序历史服务器web端地址 -->  <property>  <name>mapreduce.jobhistory.webapp.address</name>  <value>node1:19888</value>  </property>  <property>  <name>yarn.app.mapreduce.am.env</name>  <value>HADOOP\_MAPRED\_HOME=${HADOOP\_HOME}</value>  </property>  <property>  <name>mapreduce.map.env</name>  <value>HADOOP\_MAPRED\_HOME=${HADOOP\_HOME}</value>  </property>  <property>  <name>mapreduce.reduce.env</name>  <value>HADOOP\_MAPRED\_HOME=${HADOOP\_HOME}</value>  </property>  yarn-site.xml  vim $HADOOP\_HOME/etc/hadoop/yarn-site.xml  在configuration标签中添加以下内容  <!-- 设置YARN集群主角色运行机器位置 -->  <property>  <name>yarn.resourcemanager.hostname</name>  <value>node1</value>  </property>  <!-- ModeManager上运行的附属服务，需配置成mapreduce\_shuffle才可运行程序。 -->  <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  workers  添加主机名称或IP  rm $HADOOP\_HOME/etc/hadoop/workers  vim $HADOOP\_HOME/etc/hadoop/workers  node1  node2  node3  7 格式化 NameNode  在 NameNode 上执行格式化命令，只需执行一次  hdfs namenode -format  8 启动 Hadoop 集群  在 NameNode 执行启动命令。如果不成功，可能是SSH、hosts文件、主机名配置问题。主机名不能相同。  HDFS集群  start-dfs.sh stop-dfs.sh  YARN集群  start-yarn.sh stop-yarn.sh  所有集群  start-all.sh stop-all.sh  9 验证集群状态  在 NameNode 查看HDFS集群健康状态  hdfs dfsadmin -report  如果一切顺利，将如下显示：  root@node1:~# hdfs dfsadmin -report Configured Capacity: 24163061760 (22.50 GB) Present Capacity: 5636493312 (5.25 GB) DFS Remaining: 5636395008 (5.25 GB) DFS Used: 98304 (96 KB) DFS Used%: 0.00% Replicated Blocks:  Under replicated blocks: 0  Blocks with corrupt replicas: 0  Missing blocks: 0  Missing blocks (with replication factor 1): 0  Low redundancy blocks with highest priority to recover: 0  Pending deletion blocks: 0 Erasure Coded Block Groups:   Low redundancy block groups: 0  Block groups with corrupt internal blocks: 0  Missing block groups: 0  Low redundancy blocks with highest priority to recover: 0  Pending deletion blocks: 0  ------------------------------------------------- Live datanodes (3):  Name: 10.0.3.2:9866 (node1) Hostname: node1 Decommission Status : Normal Configured Capacity: 8054353920 (7.50 GB) DFS Used: 32768 (32 KB) Non DFS Used: 5746188288 (5.35 GB) DFS Remaining: 1876959232 (1.75 GB) DFS Used%: 0.00% DFS Remaining%: 23.30% Configured Cache Capacity: 0 (0 B) Cache Used: 0 (0 B) Cache Remaining: 0 (0 B) Cache Used%: 100.00% Cache Remaining%: 0.00% Xceivers: 0 Last contact: Sun May 26 14:36:19 UTC 2024 Last Block Report: Sun May 26 14:29:56 UTC 2024 Num of Blocks: 0   Name: 10.0.3.3:9866 (node2) Hostname: node2 Decommission Status : Normal Configured Capacity: 8054353920 (7.50 GB) DFS Used: 32768 (32 KB) Non DFS Used: 5743456256 (5.35 GB) DFS Remaining: 1879691264 (1.75 GB) DFS Used%: 0.00% DFS Remaining%: 23.34% Configured Cache Capacity: 0 (0 B) Cache Used: 0 (0 B) Cache Remaining: 0 (0 B) Cache Used%: 100.00% Cache Remaining%: 0.00% Xceivers: 0 Last contact: Sun May 26 14:36:17 UTC 2024 Last Block Report: Sun May 26 14:29:49 UTC 2024 Num of Blocks: 0   Name: 10.0.3.4:9866 (node3) Hostname: node3 Decommission Status : Normal Configured Capacity: 8054353920 (7.50 GB) DFS Used: 32768 (32 KB) Non DFS Used: 5743403008 (5.35 GB) DFS Remaining: 1879744512 (1.75 GB) DFS Used%: 0.00% DFS Remaining%: 23.34% Configured Cache Capacity: 0 (0 B) Cache Used: 0 (0 B) Cache Remaining: 0 (0 B) Cache Used%: 100.00% Cache Remaining%: 0.00% Xceivers: 0 Last contact: Sun May 26 14:36:17 UTC 2024 Last Block Report: Sun May 26 14:29:49 UTC 2024 Num of Blocks: 0  也可以前往HDFS Web UI 界面 http://node1:9870/    YARN集群UI界面：http://node1:8088   （4）Hadoop CLI 和 Java API 1 Hadoop CLI  此部分包括hadoop常用命令的实操。   |  |  | | --- | --- | | 常用命令 | 运行结果 | | 列出目录内容 | root@namenode:~# hadoop fs -ls / Found 2 items drwx------ - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user | | 创建目录 | root@namenode:~# hadoop fs -mkdir /directory | | 删除目录 | root@namenode:~# hadoop fs -rm -r /directory 24/06/01 08:46:47 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes. Deleted /directory | | 上传本地文件到HDFS | root@namenode:~# touch localfile root@namenode:~# hadoop fs -put localfile / root@namenode:~# hadoop fs -ls / Found 3 items -rw-r--r-- 2 root supergroup 0 2024-06-01 08:48 /localfile drwx------ - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user | | 从HDFS复制文件到本地 | root@namenode:~# rm localfile  root@namenode:~# hadoop fs -get / localfile root@namenode:~# ls 1.txt hdfs input localfile run-wordcount.sh start-hadoop.sh | | 复制文件 | root@namenode:~# hadoop fs -cp /localfile /copyfile root@namenode:~# hadoop fs -ls / Found 4 items -rw-r--r-- 2 root supergroup 0 2024-06-01 08:51 /copyfile -rw-r--r-- 2 root supergroup 0 2024-06-01 08:48 /localfile drwx------ - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user | | 移动文件 | root@namenode:~# hadoop fs -mv /localfile /movefile root@namenode:~# hadoop fs -ls / Found 4 items -rw-r--r-- 2 root supergroup 0 2024-06-01 08:51 /copyfile -rw-r--r-- 2 root supergroup 0 2024-06-01 08:48 /movefile drwx------ - root supergroup 0 2024-06-01 08:25 /tmp drwxr-xr-x - root supergroup 0 2024-06-01 08:25 /user | | 查看文件 | root@namenode:~# hadoop fs -cat /hello  Hello, hadoop! | | 更改文件或目录的权限 | root@namenode:~# hadoop fs -chmod 755 /hello | | 统计目录中的文件数量和占用空间 | root@namenode:~# hadoop fs -count /  13 10 156456 / | | 设置文件的副本数量 | root@namenode:~# hadoop fs -setrep 5 /hello Replication 5 set: /hello |   2 HDFS的Java客户端API编程  （1）在IDEA中新建Java Maven JDK1.8项目，设置pom.xml添加依赖：  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>  <dependency>  <groupId>org.apache.logging.log4j</groupId>  <artifactId>log4j-core</artifactId>  <version>2.8.2</version>  </dependency>  <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-common</artifactId>  <version>2.7.2</version>  </dependency>  <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-client</artifactId>  <version>2.7.2</version>  </dependency>  <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-hdfs</artifactId>  <version>2.7.2</version>  </dependency> </dependencies>  （2）创建HdfsClient类测试连接  public class HdfsClient {  @Test  public void check() throws Exception { // uploadFile("hello.txt", "/hello.txt");  printFileList("/");  }  public FileSystem getFileSystem() throws Exception {  Configuration configuration = new Configuration();  String fileSystemURL = "hdfs://namenode.vayki.com:59000";  return FileSystem.get(new URI(fileSystemURL), configuration, "root");  }    3 HDFS API的文件操作  3.1 文件上传  public void uploadFile(String source, String destination) throws Exception {  FileSystem fs = getFileSystem();  fs.copyFromLocalFile(new Path(source), new Path(destination));  fs.close(); }  3.2 文件下载  public void downloadFile(String hdfsPath, String localPath) throws Exception {  FileSystem fs = getFileSystem();  fs.copyToLocalFile(new Path(hdfsPath), new Path(localPath));  fs.close(); }  3.3 文件夹删除  public void deleteDirectory(String directory) throws Exception {  FileSystem fs = getFileSystem();  fs.delete(new Path(directory), true); // true 表示递归删除  fs.close(); }  3.4 文件名更改  public void renameFile(String source, String destination) throws Exception {  FileSystem fs = getFileSystem();  fs.rename(new Path(source), new Path(destination));  fs.close(); }  3.5 文件和文件夹判断  public void checkFileOrDirectory(String path) throws Exception {  FileSystem fs = getFileSystem();  FileStatus status = fs.getFileStatus(new Path(path));  if (status.isDirectory()) {  System.out.println(path + " 是一个文件夹");  } else {  System.out.println(path + " 是一个文件");  }  fs.close(); }    3.6 I/O流操作HDFS  （1）文件上传  public void putFileToHDFS() throws Exception {  // 1. 获取对象  Configuration conf = new Configuration();  FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  // 2. 输入流  FileInputStream fis = new FileInputStream(new File("testio.txt"));  // 3. 输出流  FSDataOutputStream fos = fs.create(new Path("/testio.txt"));  // 4. 输入输出流相互拷贝  IOUtils.copyBytes(fis, fos, conf);  // 5. 关闭流  IOUtils.closeStream(fos);  IOUtils.closeStream(fis);  fs.close(); }  （2）文件下载  public void getFileFromHDFS() throws IOException, InterruptedException, URISyntaxException {  // 1. 创建配置对象  Configuration conf = new Configuration();  FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  // 2. 输入流  FSDataInputStream fis = fs.open(new Path("/testio.txt"));  // 3. 输出流  FileOutputStream fos = new FileOutputStream(new File("testio1.txt"));  // 4. 流互拷贝  IOUtils.copyBytes(fis, fos, conf);  // 5. 关闭流对象  IOUtils.closeStream(fos);  IOUtils.closeStream(fis);  fs.close(); }  （3）定位文件读取  @Test /\*\*  \* 下载第1块内容  \*/ public void readFileSeek1() throws IOException, InterruptedException, URISyntaxException {  // 1. 获取对象  Configuration conf = new Configuration();  FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  // 2.获取输入流  FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));  // 3. 获取输出流  FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part1"));  // 4. 流的互拷贝（这里只拷贝指定大小的数据流128M）  byte[] buffer = new byte[1024];  for (int i = 0; i < 1024 \* 128; i++) {  fis.read(buffer);  fos.write(buffer);  }  // 5. 关闭资源  IOUtils.closeStream(fos);  IOUtils.closeStream(fis);  fs.close(); } /\*\*  \* 下载第2块内容  \*/ @Test public void readFileSeek2() throws IOException, InterruptedException, URISyntaxException {  // 1. 获取对象  Configuration conf = new Configuration();  FileSystem fs = FileSystem.get(new URI("hdfs://namenode.vayki.com:59000"), conf, "root");  // 2. 获取输入流  FSDataInputStream fis = fs.open(new Path("/hadoop-3.3.6.tar.gz"));  // 3. 指定输入流读取位置  fis.seek(1024\*1024\*128);  // 4. 获取输出流  FileOutputStream fos = new FileOutputStream(new File("hadoop-3.3.6.tar.gz.part2"));  // 5. 流的互拷贝  IOUtils.copyBytes(fis, fos, conf);  // 6. 关闭资源  IOUtils.closeStream(fos);  IOUtils.closeStream(fis);  fs.close(); }    合并文件  cat hadoop-3.3.6.tar.gz.part2 >> hadoop-3.3.6.tar.gz.part1  合并完成后进行解压和哈希值比对，发现与源文件相同。    分析实验方案中“文件写入”的过程  文件写入的流程包括：  1）客户端通过Distributed FileSystem模块向NameNode请求上传文件，NameNode检查目标文件是否已存在，父目录是否存在。  2）NameNode返回是否可以上传。  3）客户端请求第一个 block上传到哪几个datanode服务器上。  4）NameNode返回2个datanode节点，分别为datanode1、datanode2。  5）客户端通过FSDataOutputStream模块请求datanode1上传数据，datanode1收到请求会继续调用datanode2，将这个通信管道建立完成。  6）datanode1、datanode2逐级应答客户端。  7）客户端开始往datanode1上传第一个block（先从磁盘读取数据放到一个本地内存缓存），以packet为单位，datanode1收到一个packet就会传给datanode2；  8）当一个block传输完成之后，客户端再次请求NameNode上传第二个block的服务器。（重复执行3-7步）。 （5）NameNode和SecondaryNameNode工作机制 NameNode和SecondaryNameNode是Hadoop分布式文件系统中的两个关键组件，它们共同工作以确保文件系统的元数据得到有效管理。  1. NameNode的工作机制  NameNode负责管理整个文件系统的元数据，包括文件和目录的命名空间信息以及每个文件的数据块信息。当NameNode启动时，如果是首次启动，它会格式化并创建文件系统映像（fsimage）和编辑日志（edits）文件。如果不是首次启动，它会加载fsimage和edits文件到内存中。客户端对元数据进行增删改的请求时，NameNode会在内存中对数据进行相应的操作，并记录操作日志到edits文件中，但不包括查询操作，因为查询不会更改元数据。  2. SecondaryNameNode的工作机制  SecondaryNameNode的主要作用是辅助NameNode，通过定期合并fsimage和edits文件来减少NameNode的负担。SecondaryNameNode会询问NameNode是否需要执行检查点（checkpoint），这通常是基于时间间隔或edits文件大小的阈值来触发的。当触发检查点时，NameNode会滚动edits日志，创建一个新的edits文件，并将当前的edits和fsimage文件发送给SecondaryNameNode。SecondaryNameNode接收到这些文件后，会将它们加载到内存中，合并edits中的更改到fsimage中，生成一个新的fsimage文件fsimage.chkpoint，然后将这个新文件发送回NameNode。NameNode接收到新的fsimage.chkpoint文件后，会将其重命名为fsimage，这样在下次启动时就可以使用这个更新后的fsimage文件，从而加快启动过程并减少内存中元数据的恢复时间。  SecondaryNameNode不是NameNode的备份，而是一个帮助NameNode管理元数据的辅助节点。它通过设置检查点来帮助NameNode更有效地工作，确保元数据的一致性和可用性。通过这种方式，NameNode和SecondaryNameNode共同确保了HDFS文件系统的稳定性和可靠性。  **Fsimage**  定义：fsimage文件系统映像是HDFS文件系统的一个快照，它包含了在NameNode启动时的文件系统命名空间的状态。这个文件是序列化后的元数据，它记录了所有的文件和目录信息，以及它们的属性和块信息。  作用：fsimage是NameNode内存中元数据的持久化存储形式。在NameNode启动时，fsimage被加载到内存中，以恢复文件系统的命名空间状态。  更新：fsimage通常在NameNode正常启动或通过SecondaryNameNode执行检查点操作时更新。    **Edits**  定义：edits编辑日志记录了自上次NameNode启动以来对文件系统的所有修改操作，如文件创建、删除、重命名等。这些操作以事务日志的形式记录，用于追踪对文件系统所做的更改。  作用：edits的主要目的是记录文件系统状态的变化。在NameNode重启时，edits中的操作会被应用到fsimage上，以确保文件系统的状态是最新的。  更新：每当客户端对HDFS执行写操作时，相应的操作就会被记录到edits文件中。    **合并过程Checkpointing**  触发条件：SecondaryNameNode会根据配置的检查点间隔触发检查点操作。  过程：  (1)SecondaryNameNode请求NameNode进行检查点。  (2)NameNode滚动edits文件，创建一个新的edits文件（通常是edits.new），并将当前的edits和fsimage发送给SecondaryNameNode。  (3)SecondaryNameNode接收到这些文件后，将edits中的更改应用到fsimage上，生成一个新的fsimage文件（通常是fsimage.chkpoint）。  (4)SecondaryNameNode将新的fsimage文件发送回NameNode。  (5)NameNode接收新的fsimage文件，并将其重命名为当前的fsimage，同时将edits.new重命名为edits，从而完成检查点操作。  **DataNode工作机制理解**  DataNode是负责存储实际数据的节点。DataNode通常部署在集群中的多个服务器上，每个DataNode负责管理它所在物理服务器的存储。DataNode的工作机制与NameNode密切协作，后者负责管理文件系统的元数据。  当一个文件被上传到HDFS时，该文件被切分为多个数据块，然后这些块被分散存储在多个DataNode上。每个块通常会有多个副本，分布在不同的DataNode上，以提供高可靠性和容错能力。客户端在上传或下载数据时，都是直接与DataNode进行交互，而NameNode则提供必要的块位置信息。  在正常操作中，DataNode会定期向NameNode发送心跳信号和块报告。心跳信号表明DataNode是活跃的，而块报告包含了DataNode上所有数据块的详细列表，帮助NameNode维护整个文件系统的数据块位置信息。如果DataNode失败或其硬盘损坏，NameNode将依据剩余的副本重新复制数据块到其他DataNode，以确保数据的副本数不会低于设定的阈值。  DataNode参与数据块的校验和计算，以确保数据的完整性。当客户端从DataNode读取数据时，DataNode会计算数据块的校验和，并与存储时计算的校验和进行比对，如果不匹配，说明数据可能已经损坏，系统会尝试从其他DataNode获取该数据块的正确副本。    **Hadoop存档理解**  Hadoop存档是Hadoop生态系统中用于高效存储和管理大型数据集的一种机制。它允许用户将Hadoop文件系统中的文件或目录打包成一个单一的存档文件，这个文件可以是序列化的形式，从而减少存储空间的使用并提高数据访问速度。存档文件可以包含一个或多个文件，以及目录结构，使得数据的组织和检索更加方便。  当用户创建一个Hadoop存档时，系统会将指定的文件或目录压缩并存储在一个单独的文件中。这个过程中，用户可以指定不同的压缩算法，以进一步减少存档文件的大小。Hadoop存档还支持元数据的存储，比如文件权限、所有者信息和时间戳等，这有助于在恢复数据时保持数据的完整性和一致性。  使用Hadoop存档的一个主要优势是它支持并行处理和分布式计算。这意味着存档文件可以被Hadoop集群中的多个节点并行读取和处理，从而提高数据处理的效率。存档文件可以被存储在Hadoop分布式文件系统上，这为大规模数据集的存储和处理提供了一个稳定和可靠的平台。  在Hadoop生态系统中，存档也常用于数据备份和迁移。由于存档文件是自包含的，它们可以轻松地在不同的Hadoop集群之间传输，或者作为数据备份存储在不同的存储介质上。存档文件的创建和读取操作都可以通过Hadoop命令行工具或编程API进行，这为用户提供了灵活性和自动化处理数据的能力。   （6）MapReduce实操 **1 WordCount案例实操**  1．需求  在给定的文本文件中统计输出每一个单词出现的总次数  2．数据准备    3．编写程序  （1）编写mapper类   |  | | --- | |  |   （2）编写reducer类   |  | | --- | |  |   （3）编写驱动类   |  | | --- | |  |   4．集群上测试  （1）将程序打成jar包，然后拷贝到hadoop集群中    （2）启动hadoop集群  （3）执行wordcount程序          **2 序列化案例实操**  1. 需求  统计每一个手机号耗费的总上行流量、下行流量、总流量  2．编写mapreduce程序  （1）编写流量统计的bean对象   |  | | --- | |  |   （2）编写mapper   |  | | --- | |  |   （3）编写reducer   |  | | --- | |  |   （4）编写驱动   |  | | --- | |  |   **MapReduce框架原理**  在MapReduce的工作流程开始时，整个数据集被划分为一系列的输入分片，这些分片被分配到集群中的不同节点上。每个节点运行Map函数，Map函数接收输入分片并将其转换为一系列的中间键值对。这些键值对通常由键和值组成，键是数据的一个标识符，而值是与该键相关联的数据。Map函数的目的是将原始数据转换为一种格式，使得后续的Reduce函数可以更容易地对数据进行汇总。  Map函数处理完毕后，MapReduce框架会自动对所有节点生成的中间键值对进行排序和合并。这一步骤是至关重要的，因为它确保了所有具有相同键的值会被聚集在一起，为Reduce函数的执行提供了必要的前提。排序通常是基于键的字典顺序进行的，这样可以保证数据的一致性和可预测性。  排序完成后，数据进入Reduce阶段。在这个阶段，每个唯一的键及其对应的所有值被传递给Reduce函数。Reduce函数的目的是将这些值进行汇总或合并，以生成最终的输出。例如，如果Map函数输出的是每个单词出现的次数，Reduce函数可能会计算所有单词的总数，或者找出出现次数最多的单词。  **MapTask工作机制**  MapTask从输入分片开始。输入数据通常存储在分布式文件系统中，系统会将大文件分割成多个分片，每个分片由一个MapTask来处理。每个输入分片会被分配给一个MapTask，MapTask会从分片中读取数据。  MapTask读取数据时，会先将数据解析成记录，这些记录通常是键值对。在处理文本文件时，每一行文本可以被视为一个记录。输入格式决定了数据如何被解析成键值对。常见的输入格式包括TextInputFormat、KeyValueTextInputFormat等。  MapTask对读取到的记录应用用户定义的Map函数。Map函数是用户在MapReduce程序中编写的逻辑，用于处理每一个输入键值对，并生成零个或多个中间键值对。这些中间键值对是Map阶段的输出，供后续的Shuffle和Sort阶段使用。  在Map函数处理完所有输入记录后，MapTask会将生成的中间键值对临时存储在内存中。当内存中的数据达到一定阈值时，MapTask会将数据溢写到磁盘上。溢写过程包括对中间键值对进行排序和分区，以便后续的Shuffle和Sort阶段能高效进行。  MapTask会创建多个分区，每个分区对应一个ReduceTask。MapTask在将中间键值对写入磁盘时，会按照键值对的键进行分区，这样相同键的键值对会被发送到同一个分区。分区的数量通常等于ReduceTask的数量，用户可以通过Partitioner函数来自定义分区逻辑。  在所有输入数据都被处理完，并且所有中间键值对都被写入磁盘后，MapTask会将这些分区文件传输给相应的ReduceTask。这一过程称为Shuffle阶段，MapTask会将中间数据传输到Reduce节点上。  **Shuffle机制**  在Map阶段完成后，每个Map任务会产生一组中间键值对。首先，Map任务会将这些键值对根据键的哈希值进行分区，不同的键被分配到不同的分区，每个分区对应一个Reduce任务。这样做的目的是确保相同的键最终会被同一个Reduce任务处理。用户可以自定义分区逻辑，通过实现Partitioner接口来控制数据分区方式。  Map任务会对每个分区中的键值对进行排序。排序的目的是将相同的键聚集在一起，便于Reduce任务后续处理。这一步骤通常在内存中进行，当内存中存储的数据达到阈值时，会触发溢写操作，将数据写入磁盘。溢写时的数据是有序的，多个溢写文件会被合并成一个大的有序文件。  所有的Map任务完成并产生中间数据，Shuffle阶段便正式开始。每个Reduce任务会从所有的Map任务中获取它所需的分区数据。这涉及到网络传输，因为Map任务和Reduce任务通常在不同的节点上运行。为了减小网络带宽的消耗，Shuffle机制会对数据进行压缩。Map任务在发送数据之前会先压缩数据，Reduce任务在接收数据后会解压缩。  在Reduce节点，Shuffle机制会从不同的Map任务接收到相同分区的数据，这些数据仍然是有序的。Reduce任务会将这些数据进行合并，确保所有键值对按照键排序并准备好进行Reduce阶段的处理。这一过程需要处理大量的网络传输和磁盘I/O操作，因此Shuffle机制的效率直接影响到整个MapReduce作业的性能。  **Reduce join案例实操**  1）创建商品和订单合并后的bean类   |  | | --- | |  | |  |   2）编写TableMapper程序   |  | | --- | |  |   3）编写TableReducer程序   |  | | --- | |  |   4）编写TableDriver程序   |  | | --- | |  |   5）运行程序查看结果   |  | | --- | |  |  （7）Yarn Yarn基本架构  Yarn 是 Hadoop 生态系统中的一个关键组件，负责集群资源的管理和作业调度。它通过将资源管理与作业调度分离，解决了原有 Hadoop MapReduce 的扩展性问题。Yarn 的基本架构由以下几个主要组件构成：资源管理器、节点管理器、应用程序主控和容器。  资源管理器Resource Manager是整个集群的中央控制组件，负责接收作业请求、分配资源和调度任务。它包括两个关键部分：调度器Scheduler和应用管理器Application Manager。调度器根据可用资源和预定义的策略，动态地分配资源给各个应用程序，而不直接涉及任务的具体执行。应用管理器则负责管理应用程序的生命周期，包括启动应用程序主控、监控应用程序运行状态等。  节点管理器Node Manager运行在集群中的每个节点上，负责管理该节点上的资源使用情况和任务执行。它定期向资源管理器汇报节点的资源使用情况，并接收资源管理器的指令来启动或停止容器。节点管理器负责监控容器的运行状态和资源使用情况，确保任务在规定的资源限制内运行。  应用程序主控Application Master是每个应用程序独有的组件，负责整个应用程序的任务调度和监控。应用程序主控在启动时向资源管理器请求资源，并在获得资源后，在相应的节点上启动任务容器。它负责处理任务失败的重试、任务进度的跟踪等，确保应用程序的正常执行。  容器Container是 Yarn 中资源分配的基本单元，包含特定数量的 CPU、内存和其他资源。应用程序主控从资源管理器请求到的资源以容器的形式分配，节点管理器负责在具体的节点上启动这些容器，并将任务分配给它们执行。每个容器运行一个任务进程，并在任务完成后释放资源。  Yarn工作机制  IMG_256  Yarn 的工作机制涉及多个步骤和组件之间的交互，确保作业能够高效地在集群中执行。首先，用户提交作业时，作业描述和资源需求被发送到资源管理器。资源管理器作为集群的中央控制组件，接收到作业请求后，会将资源请求分配给相应的节点管理器，并启动应用程序主控Application Master。  应用程序主控是每个作业独有的组件，它在获得资源后启动，并负责管理作业的整个生命周期。应用程序主控首先在资源管理器上注册，并向资源管理器请求必要的资源来执行作业。资源管理器通过调度器来分配这些资源，调度器根据当前集群资源的使用情况和预定义的策略来决定如何分配资源。  应用程序主控获得容器后，将任务分配给这些容器执行。每个任务在一个容器内运行，应用程序主控负责监控任务的执行状态，并处理任务失败的重试和任务进度跟踪。如果任务失败，应用程序主控会根据策略决定是否重试任务或者请求更多资源。  作业完成后，应用程序主控会向资源管理器注销，并释放所有占用的资源，节点管理器停止并清理相关的容器，确保资源能够被其他作业使用。资源管理器更新作业的状态，并将结果反馈给用户，整个作业流程结束。  资源调度器  IMG_256  YARN中的资源调度器是资源管理器Resource Manager的一个关键组件，负责决定如何将集群中的可用资源分配给多个应用程序和作业。资源调度器不直接执行任务，而是根据集群资源的状态和调度策略，动态分配资源。YARN默认提供了几种调度器，包括FIFO调度器、容量调度器Capacity Scheduler和公平调度器Fair Scheduler，每种调度器都有不同的特点和适用场景。  FIFO调度器First In First Out Scheduler是最简单的一种调度器，它按照作业提交的先后顺序分配资源。FIFO调度器的优点是实现简单，但它不考虑作业的资源需求和执行时间，可能导致某些大作业长时间占用资源，从而延迟其他作业的执行。  容量调度器Capacity Scheduler旨在支持多租户环境，确保资源在不同的队列之间按配置的容量比例分配。每个队列可以配置不同的容量和优先级，队列内部的作业按照FIFO顺序调度。容量调度器允许用户根据业务需求灵活配置资源使用策略，并提供资源保障机制，确保关键作业能够获得足够的资源。它支持队列的层级结构，可以对队列进行子队列划分，以更细粒度地管理资源。  公平调度器Fair Scheduler通过动态调整资源分配，使每个作业都能获得公平的资源份额。它会平衡资源使用，防止某些作业长时间占用资源而导致其他作业得不到足够资源。公平调度器支持多种调度策略，例如基于资源份额的分配和基于作业优先级的分配，还可以配置资源池，确保不同类型的作业得到合理的资源分配。它适合在资源竞争激烈的环境中使用，能够提高集群资源的整体利用率和作业的响应速度。  MapReduce优化方法  fig-1-2x  优化数据的布局和分区。优化输入数据的格式和分区方式，可以减少数据传输和I/O操作。将数据预先分区，确保数据块大小均衡，避免数据倾斜问题。使用合适的文件格式可以提高读取和写入性能。  合理配置 MapReduce 作业的参数。作业的分片数量splits和每个分片的大小split size直接影响任务的并行度和执行时间。设置合理的 map 和 reduce 任务数量，避免任务过多导致调度开销增大或任务过少导致资源浪费。调整任务的内存和CPU资源配置，如 mapreduce.map.memory.mb 和 mapreduce.reduce.memory.mb 参数，确保任务在资源充足的情况下运行。  优化 Mapper 和 Reducer 的实现。在 Mapper 中尽量减少复杂的计算逻辑和外部资源的访问，确保快速处理输入数据。使用 Combiner 在 Mapper 阶段进行部分聚合，减少传输到 Reducer 的数据量。Reducer 的实现要注意优化聚合逻辑，避免使用全局排序等高开销操作。  数据的本地化处理。尽量将计算任务调度到数据所在的节点，减少数据传输的开销。配置集群的调度策略，使其优先考虑数据本地化，将计算任务分配到存储数据的节点上。 （8）MapReduce综合 需求：对每一个maptask的输出局部汇总  统计过程中对每一个maptask的输出进行局部汇总，以减小网络传输量即采用Combiner功能。  增加一个WordcountCombiner类继承Reducer      需求4：大量小文件的切片优化  将输入的大量小文件合并成一个切片统一处理。  在WordcountDriver中增加如下代码    运行程序，并观察运行的切片个数为1    流量汇总案例  需求2：将统计结果按照手机归属地不同省份输出到不同文件中  （1）Mapreduce中会将map输出的kv对，按照相同key分组，然后分发给不同的reducetask。默认的分发规则为：根据key的hashcode%reducetask数来分发  （2）如果要按照我们自己的需求进行分组，则需要改写数据分发（分组）组件Partitioner  自定义一个CustomPartitioner继承抽象类：Partitioner  （3）在job驱动中，设置自定义partitioner： job.setPartitionerClass(CustomPartitioner.class)  在需求1的基础上，增加一个分区类    在驱动函数中增加自定义数据分区设置和reduce task设置        需求3：将统计结果按照总流量倒序排序（全排序）  （1）把程序分两步走，第一步正常统计总流量，第二步再把结果进行排序  （2）context.write(总流量，手机号)  （3）FlowBean实现WritableComparable接口重写compareTo方法  （1）FlowBean对象在在需求1基础上增加了比较功能  package com.bigdata.mapreduce.sort; import java.io.DataInput; import java.io.DataOutput; import java.io.IOException; import org.apache.hadoop.io.WritableComparable;  public class FlowBean implements WritableComparable<FlowBean> {   private long upFlow;  private long downFlow;  private long sumFlow;   *// 反序列化时，需要反射调用空参构造函数，所以必须有* public FlowBean() {  super();  }   public FlowBean(long upFlow, long downFlow) {  super();  this.upFlow = upFlow;  this.downFlow = downFlow;  this.sumFlow = upFlow + downFlow;  }   public void set(long upFlow, long downFlow) {  this.upFlow = upFlow;  this.downFlow = downFlow;  this.sumFlow = upFlow + downFlow;  }   public long getSumFlow() {  return sumFlow;  }   public void setSumFlow(long sumFlow) {  this.sumFlow = sumFlow;  }   public long getUpFlow() {  return upFlow;  }   public void setUpFlow(long upFlow) {  this.upFlow = upFlow;  }   public long getDownFlow() {  return downFlow;  }   public void setDownFlow(long downFlow) {  this.downFlow = downFlow;  }   */\*\*  \* 序列化方法  \* @param out  \* @throws IOException  \*/* @Override  public void write(DataOutput out) throws IOException {  out.writeLong(upFlow);  out.writeLong(downFlow);  out.writeLong(sumFlow);  }   */\*\*  \* 反序列化方法 注意反序列化的顺序和序列化的顺序完全一致  \* @param in  \* @throws IOException  \*/* @Override  public void readFields(DataInput in) throws IOException {  upFlow = in.readLong();  downFlow = in.readLong();  sumFlow = in.readLong();  }   @Override  public String toString() {  return upFlow + "\t" + downFlow + "\t" + sumFlow;  }   @Override  public int compareTo(FlowBean o) {  *// 倒序排列，从大到小* return this.sumFlow > o.getSumFlow() ? -1 : 1;  } }  （2）编写mapper  package com.bigdata.mapreduce.sort; import java.io.IOException; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Mapper;  public class FlowCountSortMapper extends Mapper<LongWritable, Text, FlowBean, Text>{  FlowBean bean = new FlowBean();  Text v = new Text();   @Override  protected void map(LongWritable key, Text value, Context context)  throws IOException, InterruptedException {   *// 1 获取一行* String line = value.toString();   *// 2 截取* String[] fields = line.split("\\s+");   *// 3 封装对象* String phoneNbr = fields[0];  long upFlow = Long.*parseLong*(fields[1]);  long downFlow = Long.*parseLong*(fields[2]);   bean.set(upFlow, downFlow);  v.set(phoneNbr);   *// 4 输出* context.write(bean, v);  } }  （3）编写reducer  package com.bigdata.mapreduce.sort; import java.io.IOException; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Reducer;  public class FlowCountSortReducer extends Reducer<FlowBean, Text, Text, FlowBean>{   @Override  protected void reduce(FlowBean key, Iterable<Text> values, Context context)  throws IOException, InterruptedException {   *// 循环输出，避免总流量相同情况* for (Text text : values) {  context.write(text, key);  }  } }  （4）编写driver  package com.bigdata.mapreduce.sort; import java.io.IOException; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Job; import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class FlowCountSortDriver {   public static void main(String[] args) throws ClassNotFoundException, IOException, InterruptedException {   *// 1 获取配置信息，或者job对象实例* Configuration configuration = new Configuration();  Job job = Job.*getInstance*(configuration);   *// 6 指定本程序的jar包所在的本地路径* job.setJarByClass(FlowCountSortDriver.class);   *// 2 指定本业务job要使用的mapper/Reducer业务类* job.setMapperClass(FlowCountSortMapper.class);  job.setReducerClass(FlowCountSortReducer.class);   *// 3 指定mapper输出数据的kv类型* job.setMapOutputKeyClass(FlowBean.class);  job.setMapOutputValueClass(Text.class);   *// 4 指定最终输出的数据的kv类型* job.setOutputKeyClass(Text.class);  job.setOutputValueClass(FlowBean.class);   *// 5 指定job的输入原始文件所在目录* FileInputFormat.*setInputPaths*(job, new Path(args[0]));  FileOutputFormat.*setOutputPath*(job, new Path(args[1]));   *// 7 将job中配置的相关参数，以及job所用的java类所在的jar包， 提交给yarn去运行* boolean result = job.waitForCompletion(true);  System.*exit*(result ? 0 : 1);  } }    辅助排序和⼆次排序  有如下订单数据  订单id 商品id 成交金额  0000001 Pdt\_01 222.8  0000001 Pdt\_06 25.8  0000002 Pdt\_03 522.8  0000002 Pdt\_04 122.4  0000002 Pdt\_05 722.4  0000003 Pdt\_01 222.8  0000003 Pdt\_02 33.8  现在需要求出每一个订单中最贵的商品。  （1）利用“订单id和成交金额”作为key，可以将map阶段读取到的所有订单数据按照id分区，按照金额排序，发送到reduce。  （2）在reduce端利用groupingcomparator将订单id相同的kv聚合成组，然后取第一个即是最大值。  代码实现  package com.bigdata.mapreduce.order; import java.io.DataInput; import java.io.DataOutput; import java.io.IOException; import org.apache.hadoop.io.WritableComparable;  public class OrderBean implements WritableComparable<OrderBean> {   private int order\_id; *// 订单id号* private double price; *// 价格* public OrderBean() {  super();  }   public OrderBean(int order\_id, double price) {  super();  this.order\_id = order\_id;  this.price = price;  }   @Override  public void write(DataOutput out) throws IOException {  out.writeInt(order\_id);  out.writeDouble(price);  }   @Override  public void readFields(DataInput in) throws IOException {  order\_id = in.readInt();  price = in.readDouble();  }   @Override  public String toString() {  return order\_id + "\t" + price;  }   public int getOrder\_id() {  return order\_id;  }   public void setOrder\_id(int order\_id) {  this.order\_id = order\_id;  }   public double getPrice() {  return price;  }   public void setPrice(double price) {  this.price = price;  }   *// 二次排序* @Override  public int compareTo(OrderBean o) {   int result;   if (order\_id > o.getOrder\_id()) {  result = 1;  } else if (order\_id < o.getOrder\_id()) {  result = -1;  } else {  *// 价格倒序排序* result = price > o.getPrice() ? -1 : 1;  }   return result;  } }  package com.bigdata.mapreduce.order; import java.io.IOException; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.mapreduce.Job; import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  public class OrderDriver {   public static void main(String[] args) throws Exception, IOException {   *// 1 获取配置信息* Configuration conf = new Configuration();  Job job = Job.*getInstance*(conf);   *// 2 设置jar包加载路径* job.setJarByClass(OrderDriver.class);   *// 3 加载map/reduce类* job.setMapperClass(OrderMapper.class);  job.setReducerClass(OrderReducer.class);   *// 4 设置map输出数据key和value类型* job.setMapOutputKeyClass(OrderBean.class);  job.setMapOutputValueClass(NullWritable.class);   *// 5 设置最终输出数据的key和value类型* job.setOutputKeyClass(OrderBean.class);  job.setOutputValueClass(NullWritable.class);   *// 6 设置输入数据和输出数据路径* FileInputFormat.*setInputPaths*(job, new Path(args[0]));  FileOutputFormat.*setOutputPath*(job, new Path(args[1]));   *// 10 设置reduce端的分组* job.setGroupingComparatorClass(OrderGroupingComparator.class);   *// 7 设置分区* job.setPartitionerClass(OrderPartitioner.class);   *// 8 设置reduce个数* job.setNumReduceTasks(3);   *// 9 提交* boolean result = job.waitForCompletion(true);  System.*exit*(result ? 0 : 1);  } }  package com.bigdata.mapreduce.order; import org.apache.hadoop.io.WritableComparable; import org.apache.hadoop.io.WritableComparator;  public class OrderGroupingComparator extends WritableComparator {   protected OrderGroupingComparator() {  super(OrderBean.class, true);  }   @SuppressWarnings("rawtypes")  @Override  public int compare(WritableComparable a, WritableComparable b) {   OrderBean aBean = (OrderBean) a;  OrderBean bBean = (OrderBean) b;   int result;  if (aBean.getOrder\_id() > bBean.getOrder\_id()) {  result = 1;  } else if (aBean.getOrder\_id() < bBean.getOrder\_id()) {  result = -1;  } else {  result = 0;  }   return result;  } }  package com.bigdata.mapreduce.order; import java.io.IOException; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Mapper;  public class OrderMapper extends Mapper<LongWritable, Text, OrderBean, NullWritable> {  OrderBean k = new OrderBean();   @Override  protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {   *// 1 获取一行* String line = value.toString();   *// 2 截取* String[] fields = line.split("\t");   *// 3 封装对象* k.setOrder\_id(Integer.*parseInt*(fields[0]));  k.setPrice(Double.*parseDouble*(fields[2]));   *// 4 写出* context.write(k, NullWritable.*get*());  } }  package com.bigdata.mapreduce.order; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.mapreduce.Partitioner;  public class OrderPartitioner extends Partitioner<OrderBean, NullWritable> {   @Override  public int getPartition(OrderBean key, NullWritable value, int numReduceTasks) {   return (key.getOrder\_id() & Integer.*MAX\_VALUE*) % numReduceTasks;  } }  package com.bigdata.mapreduce.order; import java.io.IOException; import org.apache.hadoop.io.NullWritable; import org.apache.hadoop.mapreduce.Reducer;  public class OrderReducer extends Reducer<OrderBean, NullWritable, OrderBean, NullWritable> {   @Override  protected void reduce(OrderBean key, Iterable<NullWritable> values, Context context)  throws IOException, InterruptedException {   context.write(key, NullWritable.*get*());  } }  运行：  hadoop jar order-1.0-SNAPSHOT.jar com.bigdata.mapreduce.order.OrderDriver /order/input /order/output     （9）Hive实操 **安装 MySQL**  1. 安装MySQL 服务器  在node1上执行以下命令：  sudo apt update  sudo apt-get install mysql-server  2. 配置MySQL  启动MySQL服务并进行安全设置：  sudo systemctl start mysql  sudo mysql\_secure\_installation  按照提示完成安全设置，创建root用户密码等。  3. 创建Hive数据库  使用MySQL登录并创建Hive所需的数据库和用户：  sudo mysql -u root -p  在MySQL shell中执行以下命令：  CREATE DATABASE metastore;  CREATE USER 'hiveuser'@'localhost' IDENTIFIED BY 'hivepassword';  GRANT ALL PRIVILEGES ON metastore.\* TO 'hiveuser'@'localhost';  FLUSH PRIVILEGES;  EXIT;      **安装 Hive**  1. 下载并解压Hive  在node1上执行以下命令来下载并解压Hive：  cd /usr/local  wget https://downloads.apache.org/hive/hive-3.1.3/apache-hive-3.1.3-bin.tar.gz  sudo tar -zxvf apache-hive-3.1.3-bin.tar.gz  sudo mv apache-hive-3.1.3-bin hive  rm apache-hive-3.1.3-bin.tar.gz    2. 配置环境变量  在~/.bashrc文件中添加以下内容：  export HIVE\_HOME=/usr/local/hive  export PATH=$PATH:$HIVE\_HOME/bin  使更改生效：  source ~/.bashrc    3. 配置Hive  创建Hive配置目录并编辑配置文件：  sudo mkdir -p $HIVE\_HOME/conf  cd $HIVE\_HOME/conf  # sudo cp $HIVE\_HOME/conf/hive-default.xml.template $HIVE\_HOME/conf/hive-site.xml  sudo vim $HIVE\_HOME/conf/hive-site.xml  `hive-site.xml`：  <configuration>  <property>  <name>javax.jdo.option.ConnectionURL</name>  <value>jdbc:my://localhost/metastore</value>  <description>JDBC connect string for a JDBC metastore</description>  </property>  <property>  <name>javax.jdo.option.ConnectionDriverName</name>  <value>com.my.cj.jdbc.Driver</value>  <description>Driver class name for a JDBC metastore</description>  </property>  <property>  <name>javax.jdo.option.ConnectionUserName</name>  <value>hiveuser</value>  <description>Username to use against metastore database</description>  </property>  <property>  <name>javax.jdo.option.ConnectionPassword</name>  <value>hivepassword</value>  <description>Password to use against metastore database</description>  </property>  <property>  <name>hive.metastore.warehouse.dir</name>  <value>/user/hive/warehouse</value>  <description>location of default database for the warehouse</description>  </property>  <property>  <name>hive.cli.print.header</name>  <value>true</value>  </property>  <property>  <name>hive.cli.print.current.db</name>  <value>true</value>  </property>  </configuration>      4. 下载MySQL JDBC驱动  下载MySQL JDBC驱动并放到Hive的lib目录：  cd /usr/local  wget  https://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-8.0.26.tar.gz  tar -zxvf mysql-connector-java-8.0.26.tar.gz  sudo cp mysql-connector-java-8.0.26/mysql-connector-java-8.0.26.jar $HIVE\_HOME/lib/  rm mysql-connector-java-8.0.26.tar.gz  # rm -r mysql-connector-java-8.0.26    5. 初始化Hive Metastore  运行以下命令初始化Hive Metastore：  schematool -initSchema -dbType mysql    6. 验证安装  启动Hive CLI并验证安装：  hive    7. 修改HDFS权限  hdfs dfs -mkdir -p /user/hive/warehouse  hdfs dfs -chmod -R 777 /user/hive/warehouse    **验证 Hive**  1. 创建一个目录来存放数据文件  hdfs dfs -mkdir -p /user/hive/warehouse/user\_data    2. 上传数据文件到新目录  创建一个数据文件：  echo -e "1,John\n2,Jane\n3,Bob" > data.csv  然后将数据文件上传到 HDFS 的 `/user/hive/warehouse/user\_data` 目录：  hdfs dfs -put data.csv /user/hive/warehouse/user\_data/    3. 在Hive CLI中，运行以下命令创建一个示例表并查询：  CREATE EXTERNAL TABLE IF NOT EXISTS user\_data (  id INT,  name STRING  )  ROW FORMAT DELIMITED  FIELDS TERMINATED BY ','  STORED AS TEXTFILE  LOCATION 'hdfs://node1:8020/user/hive/warehouse/user\_data';    SELECT \* FROM user\_data;  IMG_256  **外部表 案例实操**  分别创建部门和员工外部表，并向表中导入数据。  1 原始数据  数据以制表符分隔  dept.txt  10 ACCOUNTING 1700  20 RESEARCH 1800  30 SALES 1900  40 OPERATIONS 1700  emp.txt  7369 SMITH CLERK 7902 1980-12-17 800.00 0.00 20  7499 ALLEN SALESMAN 7698 1981-2-20 1600.00 300.00 30  7521 WARD SALESMAN 7698 1981-2-22 1250.00 500.00 30  7566 JONES MANAGER 7839 1981-4-2 2975.00 0.00 20  7654 MARTIN SALESMAN 7698 1981-9-28 1250.00 1400.00 30  7698 BLAKE MANAGER 7839 1981-5-1 2850.00 0.00 30  7782 CLARK MANAGER 7839 1981-6-9 2450.00 0.00 10  7788 SCOTT ANALYST 7566 1987-4-19 3000.00 0.00 20  7839 KING PRESIDENT NULL 1981-11-17 5000.00 0.00 10  7844 TURNER SALESMAN 7698 1981-9-8 1500.00 0.00 30  7876 ADAMS CLERK 7788 1987-5-23 1100.00 0.00 20  7900 JAMES CLERK 7698 1981-12-3 950.00 0.00 30  7902 FORD ANALYST 7566 1981-12-3 3000.00 0.00 20  7934 MILLER CLERK 7782 1982-1-23 1300.00 0.00 10  2 建表语句  创建部门表  create external table if not exists dept(  deptno int,  dname string,  loc int  )  row format delimited fields terminated by '\t';  创建员工表  create external table if not exists emp(  empno int,  ename string,  job string,  mgr int,  hiredate string,  sal double,  comm double,  deptno int)  row format delimited fields terminated by '\t';  3 查看创建的表  show tables;  IMG_256  4 向外部表中导入数据  导入数据，数据文件位于虚拟机上  load data local inpath '/root/dept.txt' into table default.dept;  load data local inpath '/root/emp.txt' into table default.emp;  IMG_256  查询结果  select \* from dept;  select \* from emp;  IMG_256  5 查看表格式化数据  desc formatted dept;  IMG_256 主要收获与体会 在此次大数据应用项目实践中，我获得了许多宝贵的收获与体会。通过实践，我对大数据技术的核心组件有了深入的了解。在Hadoop生态系统中，如HDFS、YARN和MapReduce等关键技术的工作原理和实现细节，通过实操和代码实现，我从理论理解走向了实际操作，极大地提升了自己的技术水平。通过配置HDFS集群环境，我深刻理解了NameNode和DataNode的协同工作机制，这是对其在实际应用中重要性的切身体会。  本次项目实践让我深刻认识到数据处理过程中的优化技巧和最佳实践。数据布局和分区的优化、合理的MapReduce作业参数配置、以及Mapper和Reducer实现的优化，这些都直接影响到大数据处理的效率和性能。在实际操作中，通过对这些方面的优化实践，我学会了如何通过调整作业的分片数量、合理配置任务资源等方法，来提升数据处理的效率，并有效地避免了数据倾斜和资源浪费的问题。  通过对MapReduce编程模型的深度实践，我加深了对分布式计算思想的理解。在实际的WordCount案例中，从数据准备、编写Mapper类和Reducer类，到最后的集群测试，我将理论知识应用到实际编程中去。这个过程中，锻炼了我的编程能力，让我体会到分布式计算在处理大规模数据时的强大优势和复杂性。  通过这次实践，我深刻体会到了不断学习和更新知识的重要性。大数据技术发展迅速，新技术和新工具层出不穷。通过此次项目实践，我意识到必须保持持续学习的态度，不断更新自己的知识储备，才能在快速变化的技术环境中保持竞争力。这次大数据应用项目实践让我在技术层面上收获颇丰，在实践过程中提升了团队合作和项目管理的能力。这些宝贵的收获和体会将为我未来的学习和职业发展奠定坚实的基础。  **（三）实践成果** （1）Hadoop集群  在此次实践中，我成功搭建了一个功能完善的Hadoop集群，包括HDFS、YARN和MapReduce组件。这一过程让我深入了解了Hadoop生态系统的核心组件及其工作原理，提升了我的实际操作能力和解决问题的能力。  1 环境搭建  在三台虚拟机上分别安装了Debian 12操作系统，并通过配置静态IP地址和启用SSH服务，确保了虚拟机之间的网络通信畅通。在所有虚拟机上安装了JDK，是Hadoop运行的基础环境。通过配置环境变量JAVA\_HOME和HADOOP\_HOME，确保Hadoop可以正确识别和使用JDK。    2 HDFS配置  在HDFS配置方面，在每台虚拟机上安装了Hadoop，并配置了核心文件core-site.xml和hdfs-site.xml。在core-site.xml中指定HDFS的默认文件系统，配置Hadoop临时目录。在hdfs-site.xml中设置数据块的副本数量，确保数据的高可靠性。在NameNode上执行了格式化操作，初始化HDFS文件系统。    3 NameNode和DataNode配置  NameNode是HDFS的核心，负责管理文件系统的命名空间和元数据。在实践中，在主节点上配置NameNode，并确保其高可用性。设置SecondaryNameNode，用于定期合并元数据快照和编辑日志，减轻NameNode的负担。DataNode是实际存储数据块的节点，在其他虚拟机上配置DataNode，确保数据块在多个节点上均匀分布，提高数据的容错性和可用性。  4 YARN配置  YARN是Hadoop的资源管理和作业调度框架。在YARN配置中，在主节点上配置ResourceManager，在所有节点上配置了NodeManager。在yarn-site.xml中指定ResourceManager的主机名和端口和NodeManager的附加服务。通过这些配置，YARN可以高效管理和调度集群资源，确保作业的顺利执行。    5 启动和验证  完成所有配置后，依次启动了HDFS和YARN集群。在NameNode上执行start-dfs.sh命令启动HDFS集群，执行start-yarn.sh命令启动YARN集群。启动完成后通过Web UI界面验证了集群的健康状态。在HDFS的Web UI上查看集群的存储使用情况和数据块分布。在YARN的Web UI上，监控资源的使用情况和作业的运行状态。    6 集群优化与维护  在实践过程中学习如何优化Hadoop集群的性能。通过调整数据块大小和副本数量，优化数据分布策略，提升了HDFS的读写性能。学习如何监控集群的运行状态，使用工具如Ganglia和Nagios来实时监控集群的资源使用情况和节点状态。通过这些工具可以及时发现解决集群运行中的问题，确保集群的高可用性和可靠性。  （2）MapReduce Java 编程  在MapReduce编程部分，我完成了多个案例的实操，包括经典的WordCount案例、序列化案例、Partition分区案例、WritableComparable排序案例、Reduce join案例、辅助排序和二次排序案例。在这些案例中，我不仅掌握了MapReduce编程模型的基本原理，还深入学习了数据分片、任务调度、数据序列化与反序列化等关键技术。特别是在WordCount案例中，我从数据准备、编写Mapper类和Reducer类，到最后的集群测试，完整地经历了一个MapReduce作业的开发流程。这些实践让我更深刻地理解了分布式计算的优势和复杂性。  1 WordCount 案例实操  WordCount是MapReduce的经典入门案例，其主要目的是统计文本文件中每个单词的出现次数。在这个案例中，Mapper类用于将输入的文本文件分割成一个个单词，并输出每个单词的键值对。Reducer类用于汇总每个单词的出现次数。Driver类用于配置作业并提交到集群运行。  IMG_256  2 序列化案例实操  在序列化案例中使用Hadoop的Writable接口对自定义数据类型进行序列化和反序列化。创建FlowBean类，用于记录手机号的上行流量、下行流量和总流量。FlowBean的write和readFields方法实现了数据的序列化和反序列化。    3 Partition 分区案例  Partition分区案例使用自定义Partitioner类将Map输出的键值对分配到不同的Reducer进行处理。自定义Partitioner类用于根据手机号的归属地将数据分配到不同的Reducer中。    4 WritableComparable 排序案例  WritableComparable排序案例对Map输出的键值对进行排序。自定义的Key类实现了WritableComparable接口，用于对键值对进行排序。GroupingComparator类用于在Reducer端对键值对进行分组处理。  5 Reduce Join 案例  Reduce join案例在MapReduce作业中实现表连接操作。两个Mapper类分别读取订单表和商品表的数据，并输出键为商品ID的键值对。Reducer类将同一商品ID的订单数据和商品数据进行连接，并输出完整的订单信息。    6 辅助排序和二次排序案例  辅助排序和二次排序案例通过自定义Comparator类实现复杂的排序逻辑。自定义Comparator类用于对Map输出的键值对进行二次排序，Mapper和Reducer类处理排序后的数据。  （3）Hive数据仓库  在Hive数据仓库的学习和实践中，通过MySQL、HDFS与Hive的联合使用，并进行了外部表案例的实操。通过创建部门和员工外部表，并向表中导入数据，在操作过程中，在Hive中创建表、导入数据、执行查询以及优化查询性能。配置Hive与HDFS的联合使用使得数据在分布式存储系统中的管理更加高效。 | | | | | |
|  | | | | | |