# hadoop笔记

## Linux相关：

linux常用命令：

|  |
| --- |
| *添加可以使用 sudo命令的用户：*  Vim /etc/sudoers  如：hadoop ALL=(ALL) ALL |

|  |
| --- |
| 添加用户（必须root权限）：  sudo useradd test  sudo passwd test  修改用户名：  usermod -l hadoop hadoop1 --将hadoop1修改为hadoop |

|  |
| --- |
| 查看当前登录的是哪个用户：  whoami |

|  |  |
| --- | --- |
| Linux权限解释：  drwxrwxr-x. 2 yyf yyf 18 9月 6 18:05 aa  d:文件夹，-:文件，l:连接  rwxrwxr-x 三个一组，分别为：拥有者，用户组，其他用户的权限  r:可读，w:可写，x:可执行  （注：x对于文件夹 就是能不能进入，删除文件不在文件的rwx权限内，是否能删除该文件权限在文件的父文件夹是否有w权限）  更改文件的权限：  chmod u+x test.txt -- u:所属用户，g:所属组，o:其他用户   |  | | --- | | rwx:111 ---7  rw-:110 ---6  r--:100 ---4  --x:011 ---1  -w-:010 --2 |   chmod 777 test.txt  更改文件夹（子文件及目录）的权限：  chmod -R u+x test  更改文件/文件夹的所属用户以及组：  sudo chown -R space:space aa --只能使用管理员权限更改 |

|  |
| --- |
| 修改IP：  vim /etc/sysconfig/network-scripts/ifcfg-eno16777736  service network restart  即可生效 |

|  |
| --- |
| Vi/vim 常用快捷键：  命令模式下：  a在光标后一位开始插入  I 在该行最前面插入  A 在该行最后边插入  G 直接跳到文件的末行  dd 删除行，5dd 删除光标后的5行  yy 复制当前行，复制多行 5yy  p 粘贴  v 进入字符选择模式，选择完成后，按y复制，按p粘贴  Ctrl + v 进入块选择模式，选择完成后，按y复制，按p粘贴  Shift + v 进入行选择模式，选择完成后，按y复制，按p粘贴  ---------------------------------------------------------------------------------------------  查找并替换：  %s/yyf/space --查找yyf并替换为space  /yyf --查找文件中出现的yyf 按n下一个，按N上一个。 |

|  |
| --- |
| hostname --查看主机名  hostname space --将本次回话主机名改为 space(重启系统后又还原)  永久生效需要修改如下文件：  vim /etc/sysconfig/network  或者 hostnamectl set-hostname hadoop1  //查看系统版本信息  uname -a  //系统内核版本  uname -r  //查看日期，以及格式化时间  date +%Y-%m-%d  2017-09-14 |

|  |
| --- |
| 挂载外部存储设备  mkdir /mnt/cdrom --创建挂载点  将设备/dev/cdrom 挂载到/mnt/cdrom/：  mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom/  卸载：umount /mnt/cdrom/ |

|  |  |  |
| --- | --- | --- |
| 统计文件大小：  du -sh \*  如：   |  | | --- | | 512 CentOS\_BuildTag  5.9M EFI  512 EULA  18K GPL  466M images  84M isolinux  280M LiveOS  6.5G Packages  25M repodata  2.0K RPM-GPG-KEY-CentOS-7  2.0K RPM-GPG-KEY-CentOS-Testing-7  3.0K TRANS.TBL | | du -ah ---显示全部文件 | |

|  |
| --- |
| 查看磁盘使用情况：  df -h  关机：halt  重启：reboot |

|  |
| --- |
| //添加定时任务  crontab -e --用户添加  /etc/crontab --root 对所有用户都可用  crontab -l --查看定时任务列表  crontab 日志位置：/var/log/cron  crontab 格式：  ---表示每隔两分钟执行一次  \*/2 \* \* \* \* echo "test..." > /home/hadoop/test.out  第一列 分钟：1-59  第二列 小时：1-23(0表示子夜)  第三列 日 ：1-31  第四列 月 ：1-12  第五列 星期：0-6（0表示星期天）  第六列 要运行的命令  6列暂用Cn表示，可以用 \*;\*/n;T1-T2;a,b,c;四种形式  \*：表示每隔分钟，小时，日...都执行  T1-T2:表示从T1到T2时间要执行，如：20-30 ：22分到30分执行，以此类推日，月...  \*/n:表示每隔多长时间执行：如\*/2 每隔2两分钟执行一次  a,b,c,....：表示第 a,b,c分钟执行。 |

|  |
| --- |
| 配置ssh免密登录如：A ssh登录 B机器   1. 先生成 一对公钥和私钥 2. 将公钥拷贝到B并加入到B的授权列表中去   ssh-keygen  （默认回车，公私钥产生的默认位置：/home/hadoop/.ssh，id\_rsa为私钥，id\_rsa.pub为公钥）  ssh-copy-id 192.168.19.100  （授权文件为：/home/hadoop/.ssh/authorized\_keys） |

|  |
| --- |
| tail -f test.log ---是按照文件的唯一标识(incode)跟踪,即使文件名更改  Tail -F test.log -- 大F 是按照文件名跟踪 |

|  |
| --- |
| 查看指定服务的状态,如：  service network status  所有服务的状态：  service --status-all |

|  |
| --- |
| 自启动的服务：  chkconfig  关闭/开启指定服务的自启  chkconfig iptables off  chkconfig iptables on |

|  |
| --- |
| Alt + P ---在crt中命令sftp上传文件  put 1.txt --上传  get 1.txt --下载 |

|  |
| --- |
| 打tar包  tar -cvf aa.tar aa  解tar包  tar -xvf aa.tar  压缩：  gzip aa.tar --压缩  gzip -d aa.tar.gz --解压  一步完成打包和压缩：  tar -zcvf aa.tar.gz aa/ --打包压缩  tar -zxvf aa.tar.gz -C test/ --解压 -C目标目录下 |

|  |
| --- |
| Java环境变量配置：   1. 解压 ：tar -zxvf jdk7.tar.gz -C ../apps/ 2. 配置环境变量 vi /etc/profile   export JAVA\_HOME=/home/hadoop1/apps/jdk1.7.0\_79  export PATH=$PATH:$JAVA\_HOME/bin |

|  |
| --- |
| rpm -qa |grep mysql --搜索安装的rpm软件  rpm -ivh test.rpm ----安装rpm软件 |

## 二、shell脚本

|  |
| --- |
| Shell 脚本执行方式：  1.输入脚本的绝对路径或相对路径，必须有执行权限X  如：/home/space/test.sh  ./test.sh   1. 不用赋予+x权限，而用解释器解释执行，bash或sh +脚本   sh test.sh   1. 在脚本路径前加点   . /root/helloWorld.sh --绝对路径  . ./helloWorld.sh --相对路径  区别：  第一种和第二种会新开一个bash，不同bash中的变量无法共享 |

|  |
| --- |
| Linux Shell 中变量分为 系统变量和 用户自定义变量  可以通过 set 命令查看系统命令  系统变量：$HOME、$PWD、$SHELL、$USER等等 |
|  |

### 自己第一个shell实例

循环某一文件夹下符合某格式的文件 进行调用jar解密，因文件数量太多 又进行文件的合并操作

|  |  |  |  |
| --- | --- | --- | --- |
| first\_exc.sh 总的一个shell   |  | | --- | | #!/bin/bash  key2=`date -d "yesterday" +%Y%m%d`  echo "---------------------------------decrypt date:"$key2"---------------------------------------"  time\_key=`date +%H`  file\_path1="smsc"$time\_key"\_12"  if [ $time\_key = "01" ]  then  file\_path1="smsc24\_12"  fi  file\_path0="/data/sxsmsc/197"  key=`date -d "yesterday" +%Y%m%d`  out\_path="/data/decrypt\_data/"$key"/"$time\_key  mkdir -p $out\_path  echo `date`  echo "input file path:"$file\_path0"/"$file\_path1  echo "output file path:"$out\_path  /home/dacp/sms\_decrypt\_data/defile.sh $key $out\_path $file\_path0"/"$file\_path1  /home/dacp/sms\_decrypt\_data/merge\_file.sh $out\_path $out\_path"/SMSCH\_"$key"\_"$time\_key  rm $out\_path"/"\*.del  echo `date`  echo "---------------------------------decrypt date:"$key2"---------------------------------------" |   defile.sh 循环需要解密的文件 并调用可执行jar包解密   |  | | --- | | #!/bin/bash  function defile(){  for file in `ls $1`  do  if test -d $1"/"$file  then  defile $1"/"$file $2  else  if [[ $file =~ "$2" ]] && [[ $file != \*-INDEX ]]  then  #echo "excute jar decrypt file:"$1"/"$file  java -jar defile.jar $1"/"$file $PRIVATE\_KEY $RESULT\_PATH"/"$file".txt"  iconv -f GBK -c -t UTF-8 $RESULT\_PATH"/"$file".txt" -o $RESULT\_PATH"/"$file".del"  fi  fi  done  }  INPUT\_PATH=$3  #INPUT\_PATH="/data/sxsmsc/197/smsc24\_12/2017121923/"  if [ ! -n "$1" ]  then  KEY=`date -d "yesterday" +%Y%m%d`  else  KEY=$1  fi  PRIVATE\_KEY="Seo6uOErxOHY5aPU"  RESULT\_PATH=$2  echo "create result filepath"  mkdir $RESULT\_PATH  start=`date +%s`  echo "foreach and decrypt file,waitting......."  defile $INPUT\_PATH $KEY  end=`date +%s`  temp\_time=$[ end - start ]  echo "total time(s):"$temp\_time  echo "result file total:"`ls -l $RESULT\_PATH |grep "^-"|wc -l`  rm $RESULT\_PATH"/"\*.txt |   merge\_file.sh 为 合并 零碎的del文件   |  | | --- | | #!/bin/bash  echo "######merge file start date:"`date`  from\_file=$1  to\_file=$2  file\_size=`du $from\_file | awk '{print $1}'`  flag\_size=$((1024\*500))  if [ $file\_size -gt $flag\_size ]  then  echo "gt:"$flag\_size  file\_count=`ls -l $from\_file |grep "^-"|wc -l`  to\_file\_count=$(( $file\_size/$flag\_size ))  to\_file\_index=$(( $file\_count/$to\_file\_count ))  echo "file\_count:"$to\_file\_count  echo "file\_index:"$to\_file\_index  index=1  file\_count=1  for file in `ls $from\_file`  do  if [ $index -lt $(( $file\_count\*to\_file\_index )) ]  then  cat $from\_file"/"$file >> $to\_file"\_"$file\_count".txt"  #echo $(( $file\_count\*to\_file\_index))  else  file\_count=`expr $file\_count + 1`  fi  index=`expr $index + 1`  #echo $index  done  else  echo "lt:"$flag\_size  cd $from\_file  cat \*.del >> $to\_file"\_"$file\_count".txt"  fi  echo "######merge file end date:"`date` | |

### 2.Crontab 执行shell遇到的问题

|  |
| --- |
| 如：手动执行没有问题，但假如定时任务后就不正确  29 14 \* \* \* /home/dacp/sms\_decrypt\_data/defile.sh > /home/dacp/sms\_decrypt\_data/defile.out  后修改为如下：  15 16 \* \* \* cd /home/dacp/sms\_decrypt\_data/ && ./defile.sh > /home/dacp/sms\_decrypt\_data/defile.out  如果加入的定时任务没有效果可查看如下是否正确：  /ect/security/limits.d/90-nproc.conf  \* soft nproc 1024  Root soft nproc unlimited  1024 改为unlimited |

### 清空一个文件

|  |
| --- |
| cat /dev/null > defile.out |

### 合并多文件到一个文件

|  |
| --- |
| cat \*.del >> temp.txt |

### 查询并删除某一目录及子目录下的某类型文件

|  |
| --- |
| find 20171224 -name "\*.del" -type f -print -exec rm -rf {} \;  注意：\; 是需要的 |

### 6.文本中某一列MD5加密

|  |
| --- |
| #!/bin/bash  cat $1 | while read line  do  c1=`echo $line | cut -d ',' -f 1`  tel=`echo $line | cut -d ',' -f 2`  new\_c3=`echo -n $tel|md5sum|cut -d ' ' -f1`  c3=`echo $line | cut -d ',' -f 3`  c4=`echo $line | cut -d ',' -f 4`  c5=`echo $line | cut -d ',' -f 5`  c6=`echo $line | cut -d ',' -f 6`  echo $c1 $new\_c3 $c3 $c4 $c5 $c6 >> $2  done |

## JAVA内存结构及调优

堆内存结构：两个幸存区一样大

老年代

伊甸区（eden）

幸存区s0（surviv）

幸存区s1（surviv）

新生代 老年代

* 当一个对象被创建后，如果对象比较大会直接被放在老年代 ，否则会被放在新生代伊甸区
* 在新生代中的对象经过一次gc后没有被回收，该对象会被存放到一个幸存区如：s0，再次gc后，s0中还没被回收会被放到s1中并把伊甸区中未被回收的对象放到s1中，再次gc后，s1中对象会被放到s0中并把伊甸区未被回收的对象放入，两个幸存区来回拷贝。（两个幸存区互相拷贝效率很高）
* 经过多次gc后幸存区中还没被回收的对象会被放入老年代

### 垃圾回收

垃圾：没有被引用的对象，以及互相引用或者环形引用如下图：

A B

A

B

C

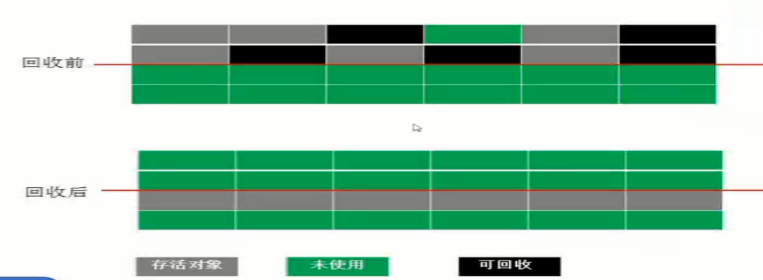
以上也视为垃圾

如何确定垃圾：1.引用计数：会有上图循环引用的问题

2.正向可达：从根部顺藤摸瓜，不可以摸到的为垃圾

垃圾回收算法：1.Mark-Sweep标记清除，缺点内存碎片化了，会FUll GC

2.Coyping复制:（新生代幸存区使用）效率高但会内存浪费，它会将内存分成两份进行互相复制。存活的对象比较少，占用内存较少，效率也高。



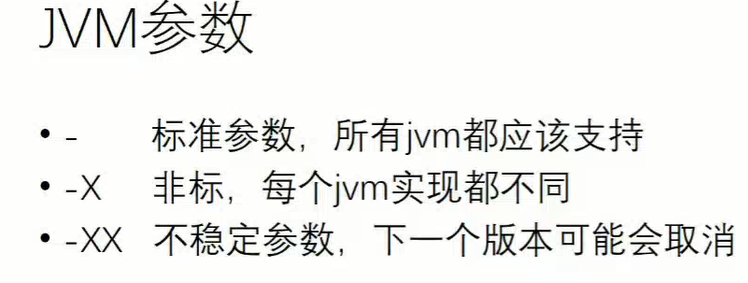
3.Mark-compact标记压缩：垃圾少，老年代中使用的算法

### JVM参数

|  |
| --- |
| GC打印：  -XX:+PrintGC  -XX:+PrintGCDetails  GC最大内存和最小：  -Xms10M  -Xmx10M |



如上可以打印内存剩余近似值。

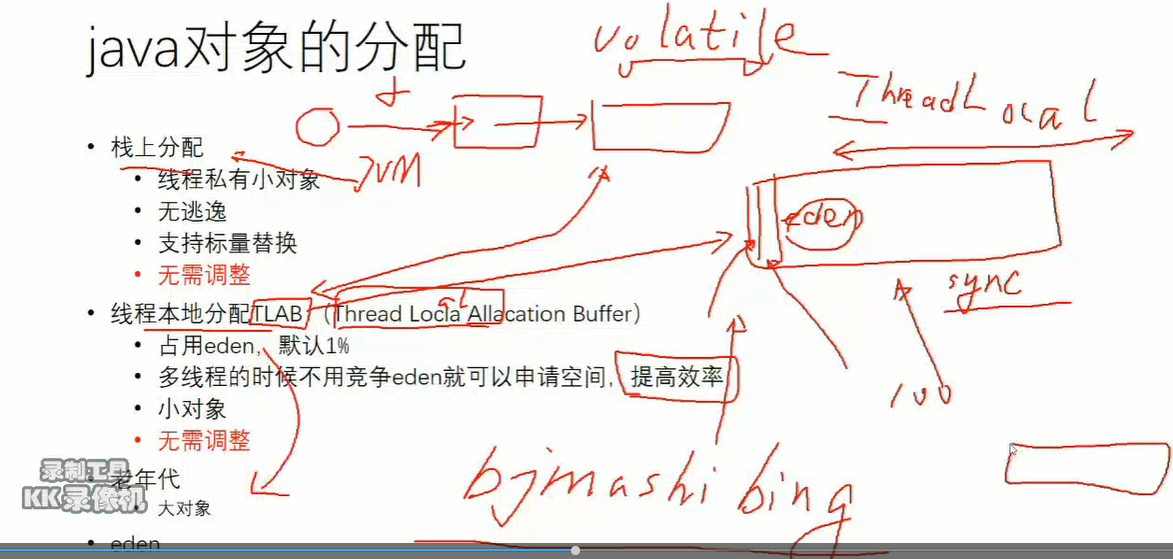


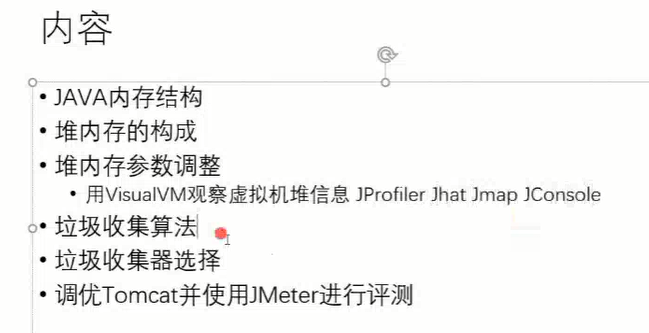
### 垃圾收集器



### Java对象的分配

如果开启了 栈上分配优化（默认 server 虚拟机都开启）如果对象比较小会首先在栈上分配





## 高并发

### 多线程

线程安全：当多个线程访问某一个类（对象或方法）时，整个类始终都能表现出正确的行为,那么这个类（对象或方法）时线程安全的。

|  |
| --- |
| Synchronized：可以在任意对象或方法上加锁。加锁后，当另一个线程试图去调用这块代码就必须先去尝试获取锁，获得则执行，否则等待着去获取锁。（会有锁竞争的问题）  当Synchronized修饰static方法时，则表加锁的对象为 该类的class对象，因此，其他任意线程调用该类时都需要先获取 class对象锁。 |

|  |
| --- |
| public static synchronized void pt(String tag){  try {  if("a".equals(tag)){  *num* = 100;  System.*out*.println("set tag a num value");  Thread.*sleep*(1000);  }else{  *num* = 200;  System.*out*.println("set tag b num value");  }  System.*out*.println("num = " + *num*);  }catch (Exception e){  e.printStackTrace();  }   } |

|  |
| --- |
| 加Synchronized为同步否则异步 |

### 脏读

给了写操作加锁但没有给读操作加锁，导致读取的数据并非最新数据。

|  |
| --- |
| IMG_256 |

读写操作都加上锁。

Oracle等关系型数据库，遵循ACID规则

9点发出查询，9:10才能查询完

9:05点发出update操作

B客户端

A客户端

最终会返回9：00时的数据值：关系型数据库的****独立性****，oracle 有一个 undo的东西去存修改前的值用来出错后的回滚，此处查询也会去取undo中的历史值以保证****独立性****。



### Synchronized

|  |
| --- |
| 当 synchronized 拥有锁重入功能，也就是使用synchronized 时，当一个线程得到了对象锁，再次请求此对象的锁是可以获取到的如下代码：  public class Test4 {  public synchronized void method1(){  System.*out*.println("method1");  method2();  }  public synchronized void method2(){  System.*out*.println("method2");  method3();  }  public synchronized void method3(){  System.*out*.println("method3");  }  public static void main(String[] args) {  final Test4 t4 = new Test4();  new Thread(new Runnable() {  @Override  public void run() {  t4.method1();  }  }).start();  } } |

|  |
| --- |
| synchronized 的锁重入在继承中适应。  public class Test5 {  static class A1{  protected int count = 10;  public synchronized void supM(){  try {  count--;  System.*out*.println("A1 count = " + count);  Thread.*sleep*(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  static class A2 extends A1{  public synchronized void subM(){  try {  while (count > 0){  count--;  System.*out*.println("A2 count = " + count);  Thread.*sleep*(1000);  this.supM();  }  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  public static void main(String[] args) {  final Test5.A2 a = new Test5.A2();  new Thread(new Runnable() {  @Override  public void run() {  a.subM();  }  }).start();  } } |

|  |
| --- |
| synchronized 修饰的方法出现异常默认情况下会释放对象锁，所以需要注意异常的处理。 |

|  |
| --- |
| synchronized 的使用方式  public class Test6 {  int age = 0;  public void test1(){  synchronized (this){//对象锁  age++;  }  }  public void test2(){  synchronized (Test6.class){//类锁  age++;  }  } } |

注意：

|  |
| --- |
| 不要使用String 常量加锁，否会出现问题。  即使使用了也不要修改字符串变量的值，否则锁的对象会改变  对象锁发生改变时，持有的锁就改变了  但如果修改一个对象的属性是不影响该对象锁的 |

### Volatile

## JAVA

### 内部类

在Java中，可以将一个类定义在另一个类里面或者一个方法里面，这样的类称为内部类。

成员内部类：如果外部类和内部类中有同名的属性或方法必须如下调用

|  |
| --- |
| JTest.this.属性  JTest.this.方法  IMG_256  外部类如果要访问内部类的属性或方法就必须new |

|  |
| --- |
| class B{  public static void main(String[] args) {  JTest jt = new JTest();  JTest.A a = jt.new A();  a.prt();  } }  内部类是依附于外部的所以要使用内部类必须构造外部类 如上代码  内部类可以拥有private访问权限、protected访问权限、public访问权限及包访问权限  内部类就像外部类的属性一样 |

|  |
| --- |
| 局部内部类：它和成员内部类的区别在于局部内部类的访问仅限于方法内或者该作用域内。  class B{   public String getName(){  class C{  public String getName(){  return "11";  }  }  return new C().getName();  }    }  注意，局部内部类就像是方法里面的一个局部变量一样，是不能有public、protected、private以及static修饰符的。 |
| 匿名内部类：  Thread t = new Thread(new Runnable() {  @Override  public void run() {    } }); |
|  |

|  |
| --- |
| 静态内部类：  class Outer{  private int age;  private static String *name* = "TC";   static class Inner{   void test(){  *name* = "DC";  //age = 10;  }  } }  静态内部类中只能调用静态的外部成员变量、方法 |

局部内部类和匿名内部类只能访问局部final变量

|  |
| --- |
| 最后补充一点知识：关于成员内部类的继承问题。一般来说，内部类是很少用来作为继承用的。但是当用来继承的话，要注意两点：  　　1）成员内部类的引用方式必须为 Outter.Inner.  　　2）构造器中必须有指向外部类对象的引用，并通过这个引用调用super()。 |

### JAVA基础

#### 1、类型转化相关

|  |
| --- |
| 常量赋给一个小于该常量范围的变量类型，如果该常量在该变量范围内则编译通过否则不通过  如：byte b = 100; //编译正常  int c = b;  byte e = c; //编译出错，c可能超出e的范围，可以强制转换byte e = (byte)c;  运算结果类型会和参加运算的最高范围一致  如：System.out.println(‘a’); // a  System.out.println(‘a’ + 1); // 98  char,byte,short三种类型参加计算时会被提升为int类型  byte b1 = 1; byte b2 = 2; byte b3 = b1 + b2; //错误 |
|  |

## 微信小游戏

|  |
| --- |
| game.js //小程序的入口文件  game.json //配置文件 |

## Git

|  |
| --- |
|  |