Java 开发细节

0.java 源文件以.java为扩展名，.class是它的组成部分。

1.Java 语法严格区分大小写

2.一个源文件.java只能有一个public 类，其他类数量随意。编译后，每一个类对应一个.class文件。也可以将非public类中写入main方法，然后可以指定运行非public类，这样的入口方法就是非public类的main方法。

3.一个源文件里有一个public类 ，则该文件名必须以public类名一致。  
4.函数必须作为类的一部分声明，因为在Java中一切都必须存在于类中。

5.在Java中函数只能返回一个值。

6.Java反射机制Reflections javac 在程序运行前，编译器会检查程序中的所有的类型是否匹配，只有在程序经过检查且所有类型匹配的情况下，程序才能真正运行。

7.Java类中不仅能包含函数和方法，还可以包含基本数据data。

8.等号的黄金法则：它获取等号右边变量的所有位（bits）对应于命名的内存区域的1和0，然后将所有这些东西复制(copy all the bits)到等号左边变量中。

9.当你实例化一个对象时，实际上是创建了对该对象的引用。

10.new关键字的用处：标明存储对象在内存的位置。（方便使用时找到它）

11.Java有一个非常奇怪、诡异的库叫做反射库，它可以让你访问私有实例变量。

12.Java中有一个叫做泛型的东西，可以把类型的选择放到以后（用户在创建此对象时将要决定的类型）。

13.Java编程 从测试开始的编程，首先测试完成目标所需要的各种功能，其次测试可能会出现的各种情况。现在测试会错误的情况下，再来编写实际要实现功能的代码。

技术学习方法：

0.需求和供给

1.学习技术的基本原理和基本语法（不要考虑细节）

2.写一个快速入门案例

3.开始研究技术的注意事项，使用细节，使用规范，如何优化

//及时复习保持知识持有率

转义字符\  
\t 制表位，实现对齐功能  
\n 换行

\“一个

\r 回车 将光标定在当前行开头  
\\ 表示“\”  
\\\\ 表示“\\”

Java易犯错误

1. 找不到文件
2. 主类名和文件名不一致
3. 编程的语法错误 尝试根据报错修改错误。
4. 业务错误
5. 环境错误

注释（comment）

解释清楚代码的功能

提高代码阅读性 将自己的思想通过注释先整理出来，再用代码去体现。

文档注释：注释内容可以被JDK提供的工具javadoc所解析，生成一套以网页文件形式体现该程序的说明文档，一般写在类  
/\*\*

\*

\*/

Java代码规范

1. 类、方法的注释，要以javadoc的方式来写
2. 格式排版 选中，然后输入tab 整体右移动 选中，然后输入shift+tab整体左移动
3. 运算符两边习惯性加一个空格
4. 行宽不超过80个字符
5. 代码编写行尾风格。

练习一  
环境变量path配置及其作用  
环境变量的作用是为了在dos的任意目录，可以去使用java和javac命令  
java编写步骤

1. 编写java的源代码
2. Javac编译，得到对应的.class；字节码文件
3. Java运行，本质就是把.class文件加载到jvm运行

变量  
概念：变量表示内存中的一个数据存储空间（不同的变量，类型不同，占用的空间大小不同。）

程序中+号的使用  
1. 当左右两边都是数值型时，则作加法运算  
2. 当左右两边有一方为字符串，则做拼接运算  
  
java数据类型（data）  
1. 基本数据类型：数值型、字符型、布尔型boolean

使用基本数据类型时，可以直接把数据存放在内存盒子中

2. （不是八大基本数据类型的数据类型）引用数据类型：类、接口、数组

引用数据类型，并不持有对象数据。存放的只是对应数据的位置（location）or地址（address）。

java API(application programming interface) 的用法：  
Java开发者给程序员设计的工具箱

字符类型细节

1.字符常量是用单引号’’括起来的单个字符。  
2.在java中，char的本质是整数，在输出时，是Unicode码对应的字符。  
3.可以直接给char赋一个整数，输出时java会按照Unicode码输出相应字符。  
4.char类型可以进行运算的，相当于一个整数，因为它都有对应的Unicode码。

5.常用的字符编码表(ASCII Unicode utf-8 gbk big5码)：

ASCIIC编码表最早的编码表，（一个字节表示128个字符，实际上一个字节可以表示256个字符，只用了128个字符）

Unicode（固定大小的编码 使用两个字节表示字符，字母和汉字统一都是占用两个字符。）  
utf-8（改进版Unicode编码）（大小可变的编码 字母使用1个字节，汉字使用3个字节）  
gbk（可以表示汉字 字母使用1个字节，汉字使用2个字节）  
big5 码（可以表示繁体中文 台湾，香港使用）  
  
布尔类型值只有true 和 false，不可以为空。Boolean类型占用一个字节。  
  
基本数据类型转换  
自动类型转换  
当Java程序在进行赋值或者运算时，精度小的类型，自动转换成精度大的类型，这就是自动类型转换。  
有多种类型的数据混合运算时，系统首先自动将所有数据转换成容量最大的那种数据类型，然后再进行计算。  
当我们把精度（容量）大的数据类型赋值给精度小的数据类型时，就会报错。反之就会进行自动类型转换。

两大系列:数据类型按精度（容量）大小排序（这是Java的规则）  
（char→int→long→float→double）  
（byte→short→int→long→float→double）  
byte 、 short和char之间不会自动类型转换 Boolean类型不参与转变  
byte short char 三者可以计算，在计算时都要首先转换成int类型。

强制类型转换  
1.自动类型转换的逆过程，将容量大的数据类型转换为数据小的数据类型，使用时要加上强制转换符（），但可能会造成精度降低（省略小数数值）或溢出（超出数据类型的范围），需要注意。  
2.强转符号（）只针对最近的操作数有效，往往使用小括号提升优先级。  
3.char类型可以保存int的常量值，但不能保存int的变量值，需要强转。  
  
基本数据类型和String类型的转换  
1.在将String类型转成基本数据类型时，要确保String类型能够转成有效的数据，可以把“123”，转成一个整数，但是不能把“hello”转成一个整数。  
2.如果格式不正确，就会抛出异常，程序就会终止，这个问题在异常处理章节中，会遇到。  
3.基本数据类型转string类型 语法：将基本类型的值+“”即可。  
 String类型转基本数据类型 语法:通过基本类型的包装类调用parseXX方法即可。  
  
关系运算符的结果都是Boolean型，也就是要么是true，要么是false。  
关系运算符组成的表达式，我们称为关系表达式。  
== 判断左右两边是否相等；！= 不等于 instanceof 检查是否是类的对象:  
“lwt”instanceof String true。  
  
逻辑运算符  
  
赋值运算符  
赋值运算符的特点：1.运算顺序从右往左 2.赋值运算符的左边只能是变量，右边可以是变量、表达式、常量值3.复合赋值运算符会进行类型转换  
复合赋值运算符  
a + = b; [等价于a = a + b；]  
  
Java规则（必须严格遵守）  
标识符不能包含空格 不能用关键字和保留名（Java以后可能更新会用的关键字）

标识符的命名规范【更加专业】  
1.包名：多单词组成时所有字母都小写：aaa.bbb.ccc  
2.类名、接口名：多单词组成时，所有单词的首字母大写：TankShotGame  
3.变量名、方法名：多单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写:tankShotGame  
4.常量名：所有字母都大写。多单词时每个单词用下划线连接.TAX\_RATE  
  
键盘输入语句  
编程过程中，需要接受用户输入的数据，就需要用到键盘输入语句来获取。Input。java，需要一个扫描器（对象），就是Scanner，具体步骤：  
1）导入该类的所在包，java.util.\*  
2) 创建该类对象（声明变量）  
3）调用里面的功能。  
Scanner sc=new Scanner(System.*in*);//创建扫描器对象  
当程序执行到next方法时，会等待用户输入···  
String name=sc.next();//接受用户输入  
  
原码、反码、补码  
1.最高制位是符号位：0表示正数，1表示负数

2.正数的原码，反码，补码都一样（三码合一）  
3.负数的反码=它的原码符号位不变，其他位取反  
4.负数的补码=它的反码+1，负数的反码=负数的补码-1  
5.0的反码，补码都是0  
6.Java中的数都是有正负号的  
7.计算机在运算的时候，都是以补码的方式来运算的  
8.当我们看运算结果的时候，要看他的原码。  
00000000 00000000 0000000 00000010 //2的原码

00000000 00000000 0000000 00000010 //2的补码  
位运算符  
3个位运算符>> 、<<和>>>,运算规则：

1. 算术右移>>:低位溢出，符号位不变，并用符号位补溢出的高位  
   2.算数左移<<:符号位不变，低位补0  
   3.>>>逻辑右移也叫无符号右移，运算规则是：低位溢出，高位补0。  
   没有<<<符号  
   int a = 1>>2 ;1右移两位相当于1/2/2=0；int c = 1<<2;1左移两个位相当于1\*2\*2=4；

小数取余运算，公式=a-(int)a/b\*b  
  
程序设计逻辑结构  
顺序结构 语句按照程序从上到下逐行地执行，中间没有任何判断和跳转。  
多分支控制语句 if-else if { }else if { }else { }可以没有else语句，如果所有条件表达式都不成立，则一个执行入口都没有。2.如果有else，如果所有条件表达式都不成立，则默认执行else代码块。  
嵌套分支  
在一个分支结构中又完整的嵌套了另一个完整的分支结构，里面的分支的结构称为内层分支，外面的分支结构称为外层分支。建议：嵌套结构不要超过3层（可读性差）  
if（）{

if（）{

else if { } }else{ } }}  
char c= sc.next().charAt(0); 取输入行第一个字符。  
在Java中只要有表达式，就会有一个值返回。  
  
for循环控制细节  
1.循环条件是返回一个布尔值的表达式  
2.for（\_\_；循环判断条件；\_\_）中的初始化和变量迭代可以写到其他地方，但是两边的分号不能省略。  
3.for(\_\_；\_\_；\_\_){ }//表示一个死循环  
4.循环初始值可以有多条初始化语句，但要求类型一样，并且中间用逗号隔开，循环变量迭代也可以有多条变量迭代语句，中间用逗号隔开。for（int i=0，j=0；i<count;i++,j+=2）{ }  
5.编写程序的秘诀：  
化繁为简：将复杂的需求，拆解成简单的需求，逐步完成。  
先死后活：先考虑固定的值，然后转成可以灵活变化的值。  
  
多重循环  
1.将一个循环放在另一个循环体中，就形成了嵌套循环。  
2.实质上，嵌套循环就是把内层循环当成外层循环的循环体。  
3.设外层循环次数为m次，内层为n次，则内层循环体实际上需要执行m\*n次。

跳转语句-return  
return使用在方法，表示跳出所在的方法，注意：如果return写在main方法，会直接退出程序。  
  
数组介绍  
数组可以存放多个同一类型的数据。数组是引用数据类型。  
声明：数据类型[] 数组名=new 数据类型 [大小/长度];//动态初始化1  
或者 先声明数组 数据类型 数组名 [] //（只是一个空值，还没有分配空间）   
数据类型 数组名 [] 数组名=new 数据类型 [size]//动态初始化2

或 数据类型 数组名[]= {元素值，元素值… }//静态初始化 数据不多时使用  
方法：我们可以通过数组下标来访问数组元素。  
eg. Int a[] = new int[]; int a[] a=new int[];  
数组使用细节  
1.数组是多个相同类型数据的组合，实现对这些数据的统一管理。  
2.数组存储的数据类型可以是任何数据类型（基本数据类型和引用数据类型），但不要混用。  
3.数组创建后，如果没有赋值，有默认值。  
4.数组下标是从0开始的。  
5.超出数组容量会报错，数组越界。  
6.数组属于引用类型，数组型数据本质是对象（object）。  
  
数值赋值机制  
基本数据类型是简单的值赋值情况  
数组在默认情况是引用传递，赋的值是地址，赋值方式为引用赋值.  
  
int arr1 [5]={1,2,3,4,5};int arr2 [5]=arr1;(arr2与arr1的地址相同)；arr2[0]=10;  
arr1 [5]={10,2,3,4,5};  
  
数组拷贝  
将int[]arr1={10,20,30};拷贝到arr2数组，要求数据空间是独立的  
代码： int[] arr2 = new int[arr1.length];  
for(int i = 0;i<=arr1.length;i++) {

Arr2[i]=arr1[i];

}  
  
数组反转  
  
数组扩容与缩减  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*);  
 int arr1[] = {11, 22, 33, 44, 55, 66};  
 do{  
 int[] arrNew = new int[arr1.length+1];  
 for (int i = 0; i < arr1.length ; i++) {  
 arrNew[i] = arr1[i];  
 }  
 arrNew[arrNew.length-1] = sc.nextInt();  
  
 arr1=arrNew;  
 for (int i=0;i<arr1.length;i++){  
 System.*out*.print(arr1[i]+"\t");  
 }  
  
 char key=sc.next().charAt(0);  
 if(key=='n'){break;}  
}while (true);  
}

}  
  
排序介绍   
排序是将多个数据，依指定顺序进行排列的过程  
排序的分类：1.内部排序（数据量小）2.外部排序（数据量大）

查找   
顺序查找Seq Search （sequence顺序）  
编程技巧：设置一个标签 int index=-1；  
二分查找  
1.有序数组2.  
二维数组：实际上是由多个一维数组组成的 声明方式：int[][]y int[]y[]或者int y[][];  
二维数组中的每个行元素是一维数组

数据类型 [][] name =new 类型[size][size] ;  
或者先声明：数据类型[][] name;

Name=new datatype[][];  
二维数组中的行元素存储一维数组的地址  
二维数组的使用 二维数组它的各个一维数组的长度可以相同，也可以不同。

{{0，1}，{0，1，2}{0，1，2，3}}

Int [][] arr0=new int[3][];创建 二维数组，但是只是确定了一维数组的个数  
for(int i=0;i<arr0.length;i++){

Arr0[i]=new int[i+2];给一维数组开内存空间。

}  
静态声明：datatype name [][]={{ },{ },{ },…}  
  
二维数组的遍历  
public class Practice {  
 public static void main(String[] args) {  
 //杨辉三角  
Scanner SC=new Scanner(System.*in*);  
int n=SC.nextInt();  
 System.*out*.println("===杨辉三角行数===");  
 int arr[][]=new int[n][];  
 for(int i=0;i<arr.length;i++){  
 arr[i]=new int[i+1];//关键步骤，为二维数组的每个数据元素开空间  
 for ( int j=0;j<arr[i].length;j++){  
 if(j==0||j==arr[i].length-1){arr[i][j]=1;}  
 else {  
 arr[i][j]=arr[i-1][j]+arr[i-1][j-1];  
 }  
 }  
 }  
 for(int i=0;i<arr.length;i++){  
 for ( int j=0;j<arr[i].length;j++){  
 System.*out*.print(arr[i][j]+"\t");  
 }System.*out*.println();  
 }  
 }  
 }  
类(自定义数据类型)与对象  
类的属性注意事项  
1.属性的定义语法和变量差不多；（访问修饰符）属性类型 属性名；  
2.属性的定义类型可以为任意类型，包含基本类型和引用类型  
3.属性如果不赋值，有默认值。

创建对象的内存说明：例如创建一个Person对象；

Person p1= new Person（）；//p1是对象名（对象引用），放在栈里，指向堆里的对象  
new Person（）创建的对象，在堆里开辟一个空间。  
Java类和对象的内存分配机制  
1.栈：一般存放基本数据类型（局部变量）2.堆：存放对象（Cat cat，数组等。）   
3.方法区（a.常量池（常量、字符串） b.加载类信息  
  
创建对象语法：  
1.先声明，再创建 2. 直接创建  
Cat cat1; cat1=new Cat(); Cat cat1= new Cat();  
  
Java创建对象的流程分析：首先加载类信息（属性和方法信息，只会加载一次），然后在堆中分配空间，进行默认初始化，把地址赋给栈里的对象名。最后，进行指定初始化。  
  
方法的调用机制分析  
1.当程序执行到方法时，就会开辟一个独立的空间(栈空间)  
2.当方法执行完毕，或者执行到return语句时，就会返回。

3.返回到调用方法的地方

4.返回后，继续执行方法后面的代码

5.当main方法（栈）执行完毕，整个程序退出。  
  
方法使用细节  
1.一个方法最多有一个返回值 （如果要返回多个结果，可以返回数组）  
2.如果方法要求有返回数据类型，则方法体中最后的执行语句必须为return值，而且要求返回值必须和return的值类型一致和兼容。  
3.如果方法是void，则方法体中可以没有return语句，或者只写return。  
4.方法里不能再定义方法。  
5.方法调用：a.同一个类中的方法可以直接调用 b.跨类中的方法 A类调用B类方法：需要先创建B对象，然后通过对象名调用。对象名.方法名（参数）  
  
方法的传参机制（参数\_parameter）  
结论：基本数据类型，传递的是值（值拷贝），形参的任何改变不影响实参。  
引用类型传递的是地址，可以通过形参影响实参。  
克隆对象  
  
方法递归（recursion）调用  
递归就是方法自己调用自己，每次调用时传入不同的变量。  
递归调用重要规则  
1.执行一个方法时，就创建一个新的受保护的独立空间（栈空间）  
2.方法的局部变量是独立的，不会相互影响  
3.如果方法中使用的时引用类型变量，该变量就会共享。  
4.递归必须向推出递归的条件逼近，否则就是无限递归。  
5.当一个方法执行完毕，或遇到return，就会返回，遵守谁调用，就将结果返回给谁，同时当方法执行完毕或返回时，该方法也就执行完毕。  
方法重载（overload）  
Java中允许同一个类中，多个同名方法的存在，但要求形参列表不一致（个数或类型），  
返回类型无要求。  
  
可变参数  
Java允许将同一个类中多个同名同功能但参数个数不同的方法，封装成一个方法。  
eg. Public int sum (int n1 , int n2, int n3, int n4, int n5,…){ }  
简化成可变参数形式 public int sum（int… nums）{ }//int… 表示接受的是可变参数，类型是int，即可以接收多个int  
使用可变参数时，可以当作数组来使用。 nums可以当作数组。  
注意细节：

1.可变参数的实参可以时0个或者任意多个。  
2.可变参数的实参可以为数组。

3.可变参数的本质就是数组。  
4.可变参数可以和普通类型的参数一起放在形参列表，但必须保证可变参数在最后。  
5.一个形参列表中，只能存在一个可变参数  
  
作用域  
1.在Java中，主要的变量就是属性（成员变量）和局部变量。  
2.我们说的局部变量一般是指在成员方法中定义的变量。作用域为定义它的代码块中。  
3.全局变量（属性），作用域为整个类体中的各个方法使用。

4.全局变量（属性）可以不赋值，直接使用，因为有默认值，局部变量必须赋值使用。  
作用域的细节  
1.属性和局部变量可以重名，访问时遵循就近原则  
2.在同一个作用域中，比如在同一个成员方法中，两个局部变量，不能重名。  
3.属性生命周期较长，伴随着对象的创建而创建，伴随着对象的销毁而销毁。局部变量，生命周期较短，伴随着它的代码块的执行而创建，伴随着代码块的结束而销毁。  
4.作用域范围不同：全局变量/属性：可以被本类使用，或其他类使用（通过对象调用）  
局部变量：只能在本类中对应的方法中使用  
5.修饰符不同  
全局变量/属性可以加修饰符

局部变量不可以加修饰符  
  
构造器/构造方法

基本语法  
[修饰符]方法名 （形参列表）{

方法体；

}  
1.构造器的修饰符可以默认，也可以是public，protected ，private  
2.构造器没有返回值  
3.方法名和类名 必须一样

1. 参数列表和成员方法一样的规则
2. （创建对象时）系统会自动的调用该类的构造器,完成对象的属性初始化。

构造方法是类的一种特殊的方法，它的主要作用是完成新对象的初始化。  
构造器的注意事项和使用细节  
1.一个类可以定义多个不同的构造器，即构造器的重载  
2.构造器名和类名要相同

3.构造器没有返回值  
4.构造器是完成对象的初始化，并不是创建对象

5.在创建对象时，系统自动的调用该类的构造方法。  
6.如果没有定义构造器，系统会自动给类生成一个默认无参构造器  
7.一旦定义了自己的构造器，默认的构造器就被覆盖了，就不能再使用默认的无参构造器，除非显式的定义一下，即类名（）{ }  
8初始化对象属性时，尽量复用构造器。 259.class  
this 关键字  
Java虚拟机会给每个对象分配this，代表当前对象。  
public person(String name ,int age ){  
 // this.age 就是当前对象的属性 ，当前对象就是 调用创建的对象  
 this.age=age;  
 this.name=name;  
}

this小结：那个对象调用，this就代表那个对象  
this使用细节

1.this关键字可以用来访问本类的属性、方法、构造器  
2.this用于区分当前类的属性和局部变量  
3.访问成员方法的语法：this.方法名（参数列表）

4.访问构造器语法：this（参数列表）；（只能在构造器中访问另外一个构造器）。该语句必须放置在第一条语句。不能用普通方法中调用构造器。  
5.this不能在类定义的外部使用，只能在类定义的方法中使用。  
   
匿名对象 没有对象名（对象引用），没有指向。只能使用一次，用完后销毁。  
  
//常用快捷键 删除当前行ctrl+D ctrl+/ 快速注释  
//ctrl+shift+↑/↓ 移动当前行  
//生成构造器 Alt+insert ctrl+H 查看一个类的层级关系  
//将光标放在一个方法上，输入ctrl+B，可以快速定位到方法  
//自动分配变量名 .var 270

IDEA模板/自定义模板 template  
s out模板快捷键：System.*out*.println();

fori模板快捷键：for (int i = 0; i < ; i++) { }

IDEA模板位置：设置->编辑器->实时模板（查看有哪些模板快捷键/可以自己增加模板）  
模板可以高效的完成开发，提高速度。  
  
包的基本介绍  
一个文件夹（一个包里）下面，不能有两个同名的类。  
包 ：1.区分相同名字的类。2.当类很多时，能够管理类。（Java API文档）3.控制访问范围  
包的基本语法  
package java.lwtedu 关键字 包名  
包的本质是：创建不同文件夹来保存类文件。  
包的命名规则：如同一般标识符命名规则一样。命名规范：一般是小写字母+.   
例如：com.公司名.项目名.业务模块名 com.sina.crm.user 用户模块

常用的包  
java.lang.\* lang包是默认的包

Java.util.\*

Java.net.\*

Java.awt.\*

导入包 语法 import 包 我们引入一个包主要是要使用包下面的类。

建议：需要使用哪个类，就导入那个类即可。不建议使用\*导入包中的所有类。  
包的注意事项和细节

1. Package是声明当前类所在的包，需要放在类的最上面，一个类中最多只有一句package
2. Import指令放在package的下面在类定义的前面，可以有多句且没有顺序要求。

访问修饰符

Java提供了四种访问控制修饰符号用于控制方法和属性（成员变量）的访问权限（范围）：  
1.public公开级别：

2.protected受保护级别：

3.default（默认）：

4.private私有级别：  


面向对象程序设计三大特点：封装、继承和多态。  
封装：（encapsulation）就是把抽象出的数据[属性]和对数据的操作[方法]封装在一起，数据被保护在内部，程序的其它部分只有通过被授权的操作[方法],才能对数据进行操作。  
封装好处1.隐藏实现细节2.可以对数据进行验证，保证安全合理（控制数据的合理性）  
  
  
封装的实现步骤  
1.将属性进行私有化private[不能直接修改属性]  
2.提供一个公共（public）的set方法，用于对属性判断并赋值  
public void setXXX(类型 参数名){//XXX表示某个属性

//加入数据验证的业务逻辑

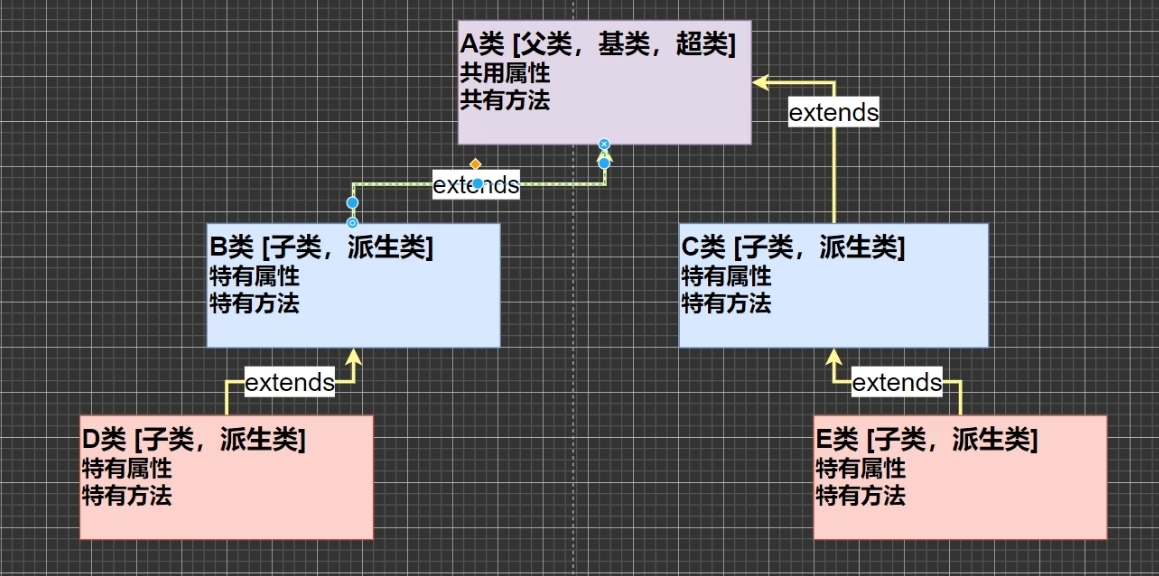
属性=参数名;

}  
3.提供一个公共的get方法，用于获取属性的值  
public XX getXXX（）{//权限判断

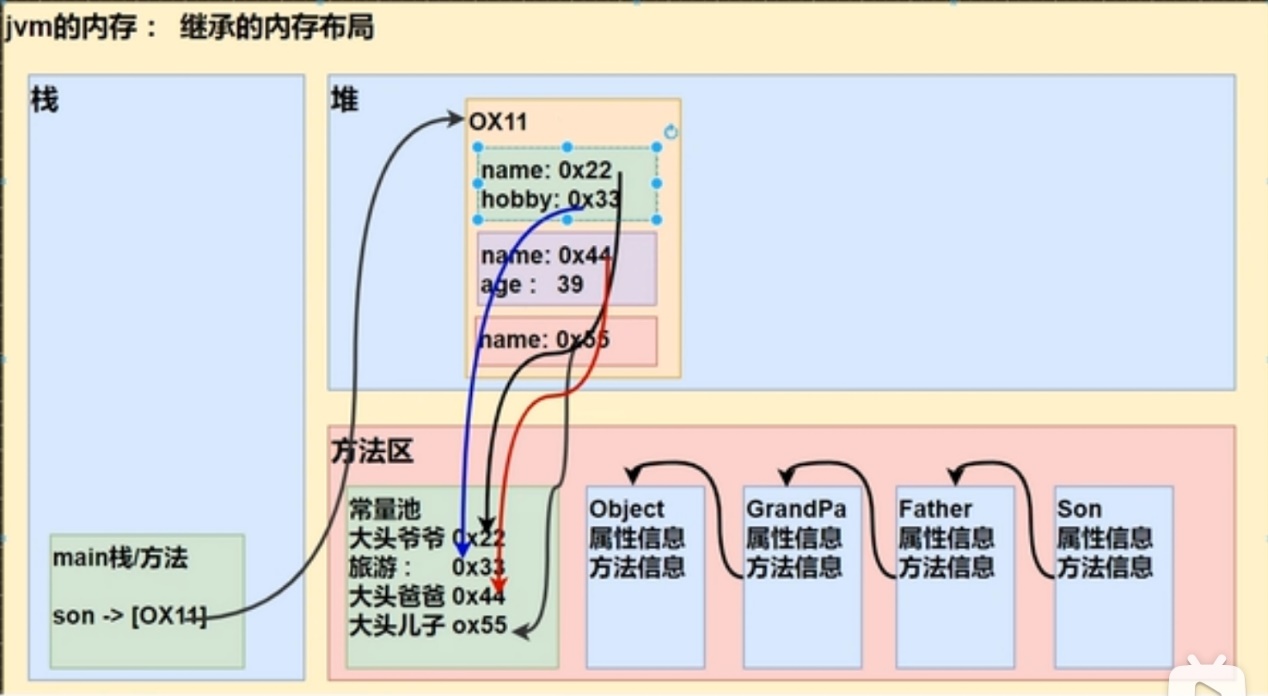
Return XX;

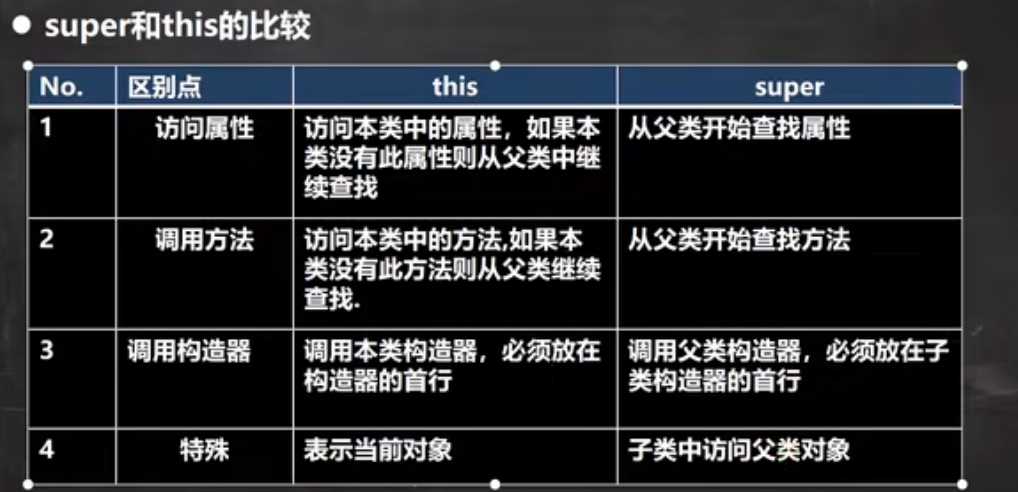
}  
将构造器和setXxx结合 把setXxx方法写进构造器之中，仍然可以验证数据的合理性。  
  
继承的基本介绍  
两个或多个类的方法和属性有很多是相同的，可以从这些类中抽象出父类,在父类中定义这些相同的属性和方法，所以的子类不需要重新定义这些属性和方法，只需要通过声明extends来继承父类即可，使用类的继承减少代码的复用性。  
继承的基本语法

Class 子类extends父类{

}//子类（派生类）就会自动拥有父类定义的属性和方法。父类的别名（超类、基类）

继承给编程带来的便利：1.代码的复用性提高了2.代码的扩展性和维护性提高了  
继承的细节问题：  
1.子类继承了所有的属性和方法，非私有的属性和方法可以在子类直接访问，但是私有属性或私有方法不能在子类直接访问，子类要通过（父类提供）公共方法(public function)去访问私有方法和私有属性。  
2.子类必须调用父类的构造器，完成父类的初始化。子类构造器隐含默认语句super（）//该语句功能：默认调用父类的无参构造器；  
3.当创建子类对象时，不管使用子类的哪个构造器，默认情况下总会去调用父类的无参构造器，如果父类没有提供无参构造器，则必须在子类的构造器中用super去指定使用父类的哪个构造器完成对父类的初始化工作，否则，编译不会通过。  
4.如果希望指定去调用父类的某个构造器，则显示的调用一下。形式：super（参数）  
5.super语句在使用时，必须放在构造器的第一行

6.super（）和this（）都只能放在构造器第一行，因此这两个方法不能共存在一个构造器  
7.java所有类都是Object类的子类，Object类是所有类的基类。  
Ctrl+H可以看到类继承的关系  
8.父类构造器的调用不限于直接父类！将一直往上追溯直到Object类（顶级父类）  
9.子类最多只能继承一个父类（直接继承）,java中是单继承机制。  
10.不能滥用继承，子类和父类之间必须满足 is-a 的逻辑关系（谁是谁的什么什么）。  
  
继承的本质  
创建子类对象时，内存中到底发生了什么？当子类对象创建好后，建立查找的关系。  
  
super关键字  
super代表父类的引用，用于访问父类的属性、方法、构造器  
1.访问父类的属性和方法，但不能访问父类的私有属性和私有方法。  
2.调用父类构造器的好处（分工明确，父类属性由父类初始化，子类属性由子类初始化）  
3.当子类中有和父类中的成员（属性和方法）重名时，为了访问父类的成员，必须通过super。如果没有重名，使用super、this、直接访问是一样的效果！  
4.super的访问不限于直接父类，如果爷爷类和本类中有同名的成员，也可以使用super去访问爷爷类的成员；如果多个基类（上级类）中都有同名的成员，使用super访问遵循就近原则；根据查找顺序从下往上找属性或者方法。

super和this的比较  
  
  
方法重写和覆盖（override）  
基本概念：方法覆盖就是子类有一个方法，和父类（基类）的某个方法的名称、返回类型、参数一样，那么我们就说子类的这个方法覆盖了父类的方法。  
  
override细节  
1.子类的方法的形参列表、方法名称，要和父类方法的参数，方法名称完全一样。  
2.子类方法的返回类型和父类方法的返回类型一样，或者是父类返回类型的子类。  
比如父类方法的返回类型是Object类型，子类方法返回的类型是String  
3.子类方法不能缩小父类方法的访问权限，可以相等或者扩大。（儿子起码要比爹强）  
  
  
多态  
引出多态：传统方法代码的复用性不高，而且不利于代码维护。  
方法或对象具有多种形态。是面向对象的第三大特征，多态是建立在封装和继承基础之上的。  
  
多态的具体体现  
1.方法的多态：重写和重载体现多态。  
2.对象的多态（核心）：（披着羊皮（编译类型）的狼（运行类型））  
a.一个对象的编译类型和运行类型可以不一致  
(一个父类的对象引用可以指向它的子类的一个对象)向上转型

b.编译类型（javac）在定义对象时，就确定了，不能改变  
c.运行类型（java实际真正运行，堆里的对象）是可以变化的.

d.编译类型看定义时 = 号的左边，运行类型看 = 号的右边。  
多态注意事项和细节讨论  
多态的前提是：两个对象（类）存在继承关系 多态的向上转型  
向上转型： 父类的引用指向了子类的对象  
语法 ： 父类类型 引用名= new 子类类型（）；  
特点： 编译类型看左边，运行类型看右边。可以调用父类中的所有成员（需遵守访问权限），不能调用子类中的特有成员（因为在编译阶段，能调用那些成员，是由编译类型类决定的）； 最终运行效果看子类的具体实现！  
  
多态的向下转型（为了调用子类的特有成员）  
1.语法：子类类型 引用名=（子类类型）父类引用；  
2.只能强转父类的引用，不能强转父类的对象  
3.要求父类的引用必须指向的是当前目标类型的对象  
4.当向下转型后，就可以调用子类类型中所有的成员。

属性没有重写之说！访问属性的值，看编译类型 访问方法，首先看运行类型  
instanceOf比较操作符，用于判断对象的类型是否为XX类型或XX类型的子类型  
语法 对象的类型 instanceof XX类型  
  
Java的动态绑定机制  
1.当调用对象方法的时候，该方法会和该对象的内存地址/运行类型绑定。  
2.当调用对象属性时，没有动态绑定机制，哪里声明，哪里使用。  
  
多态的应用  
1.多态数组：数组的定义类型为父类类型，里面保存的实际元素类型为子类类型  
2.多态参数：方法定义的形参类型为父类类型，实参类型允许为子类类型。

object类详解  
==和equals的对比   
==：是一个比较运算符.既可以判断基本类型，又可以判断引用类型（如果判断基本类型，判断值是否相等。如果判断引用类型，判断的是地址是否相等，即判定是不是同一个对象）  
equals：是object类中的方法，只能判断引用类型（地址是否相同）。默认判断的是地址是否相等，子类中往往重写该方法，用于判断内容是否相等。（比如String和Integer）  
  
hashCode方法  
1.提高具有哈希结构的容器的效率  
2.两个引用，如果指向的是同一个对象，则哈希值肯定是一样的！  
3.两个引用，如果指向的是不同对象，则哈希值是不一样的。  
4.哈希值主要根据地址号来的，不能完全将哈希值等价于地址。  
  
toString方法  
基本介绍：  
默认返回：全类名（包名+类名）+@+哈希值的十六进制  
（具体查看Object的toString方法）

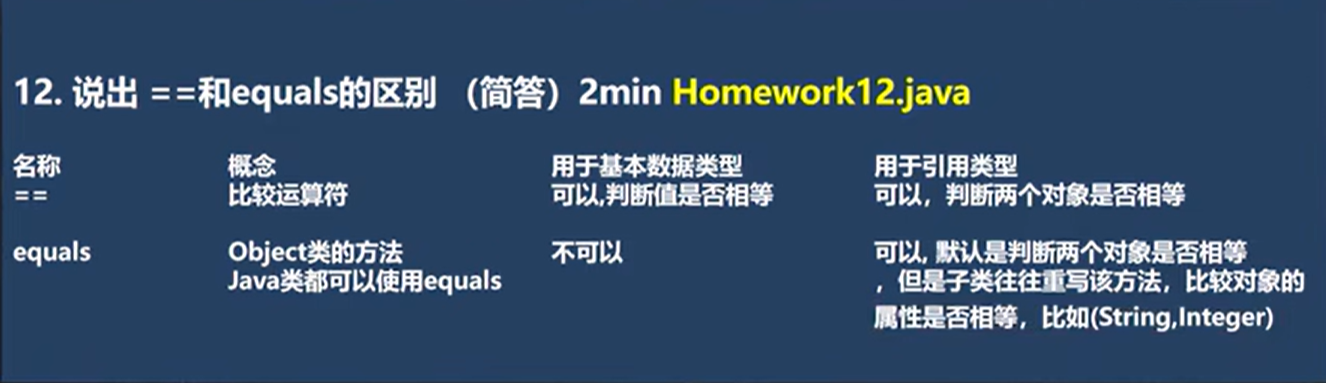
重写toString方法，输出对象的属性。使用快捷键alt+insert->toString

当直接输出一个对象时，toString方法会被默认的调用，比如System.out.println(obj),就会默认调用obj。toString（）。  
  
finalize方法  
1.当对象被回收时，系统自动调用该对象的finalize方法。可以重写该方法，做一些释放资源的操作。如果不重写finalize，那么就会调用object类的finalize，即默认处理。  
2.什么时候被回收：当某个对象没有任何引用时，则jvm就认为这个对象是一个垃圾对象，就会使用垃圾回收机制来销毁该对象，在销毁该对象前，会调用该对象的finalize方法。  
3.垃圾回收器的调用，是由系统来决定（有自己的GC算法），也可以通过System.gc()主动触发垃圾回收机制。

断点调试（debug）

程序从头开始，运行到断点处（暂停住）。

断点调试：一步一步的看源码执行的过程，从而发现错误所在。  
在断点调试过程中，是运行状态，是以对象的运行类型来执行的。   
  
建议：一个代码完成一个功能，尽量不要混在一起。降低代码的耦合度。  
Java代码中if条件语句常使用“如果不这样”的否定条件判断形式。



什么是多态？请具体举例：方法或对象具有多种形态，是OOP的第三大特征。

1. 方法多态  
   方法重载（overload）与方法重写（override）
2. 对象多态

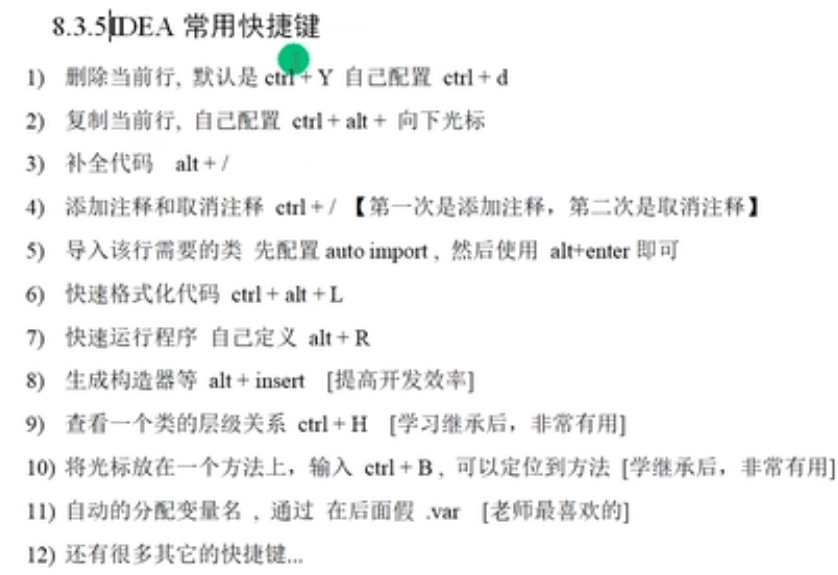
（1）对象的编译类型和运行类型可以不一致，编译类型在定义时就确定，不能变化。

（2）对象的运行类型是可以变化的，可以通过getclass（）Object类方法

（3）编译类型看定义时 = 号左边，运行类型看等号右边

Java动态绑定机制

1. 当调用对象方法时，该方法会和对象的内存地址/运行类型绑定
2. 当调用对象属性时，没有动态绑定机制，哪里声明，哪里使用。



老韩：“每个人都可以活的更精彩，不要让惰性毁了你的聪明才智。”

休息一下再扬帆起航。

JAVA高级部分

Static 修饰词，要修饰的是通用的功能（function）或数据（data）。

类变量（静态变量）和类方法

类变量的引入，如果变量是独立于对象的变量，以后再访问对象会很麻烦，没有使用OOP。使用静态变量解决问题。

（特点：static变量被类的所有对象实例共享，static变量在类加载（面向对象内存分析提到）时就生成了）

类变量也叫静态变量/静态属性，是该类的所有对象共享的变量，任何一个该类的对象去访问它时，取到的都是相同的值，同样任何一个该类的对象去修改它时，修改的也是同一个变量。

定义静态变量

访问修饰符 static 数据类型 变量名

访问类变量

类名.类变量名【推荐】或者 对象名.类变量名

（静态变量的访问修饰符和访问的权限和普通变量一样。）

说明：类变量是随着类的加载而创建，所以即使没有创建对象实例也可以访问。

类变量使用注意事项和细节

1. 什么时候使用类变量

当我们需要让某个类所有对象都共享一个变量时，就可以考虑使用类变量

1. 类变量和实例变量的区别

类变量是该类的所有对象共享的，而实例变量是每个对象独享的。

1. 加上static称为类变量/静态变量，否则称为实例变量/非静态变量/普通变量
2. 实例变量不能通过 类名.类变量名 方式访问
3. 类变量是在类加载时就初始化了，也就是说，即使没有创建对象，只要类加载了，就可以使用类变量了。
4. 类变量的生命周期是随类的加载开始，随着类消亡而销毁。

类方法 静态方法

定义：访问修饰符 static 数据返回类型 方法名 （）{ }【推荐】

类方法的调用

使用方式：类名.类方法名 或 对象名.类方法名[前提是满足访问修饰符的访问权限和范围]

类方法经典的使用场景

当方法中不涉及到任何和对象相关的成员，则可以将方法设计成静态方法，提高开发效率。

比如：工具类中的方法 ： utils

Math类、 Arrays类、Collections集合类（好处：如果我们希望不创建实例，也可以调用某个方法。即当作工具来使用，这时把方法做成静态方法时非常合适）

开发自己的工具类，可以将方法做成静态的，方便调用。

类方法使用注意事项和细节

1. 类方法和普通方法都是随着类的加载而加载，将结构信息存储在方法区：
2. 类方法中无this的参数 普通方法中隐含着this的参数
3. 类方法可以通过类名调用，也可以通过对象名调用。普通方法和对象有关，需要通过对象名调用，不能通过类名调用。
4. 类方法中不允许使用和对象有关的关键字，比如this和super。普通方法可以。
5. 类方法（静态方法）中只能访问静态变量或静态方法。
6. 普通成员方法，可以访问非静态成员，也可以访问静态成员。（必须遵守访问权限）

理解main方法语法

解释main方法的形式 pubic static void main （String[] args）{}

1. main方法是Java虚拟机调用的
2. Java虚拟机需要调用类的main（）方法，所以该方法的访问权限必须是public
3. Java虚拟机在执行main（）方法时不必创建对象，所以该方法必须是static
4. 该方法接受String类型的数组参数，该数组中保存执行Java命令时传递给所有允许的类的参数//args形参 是如何传入
5. Java执行的程序 参数1 参数2 参数3…

特别提示：

1. 在main（）方法中，我们可以直接调用main方法所在类的静态方法或静态属性。
2. 但是，不能直接访问该类中的非静态成员，必须创建该类的一个实例对象后，才能通过这个对象去访问类中的非静态成员。

代码块（初始化块）

基本介绍：属于类中的成员，类似于方法，将逻辑语句封装在方法体中，通过{}包围起来。

但和方法不同，没有方法名，没有返回，没有参数，只有方法体，而且不用通过对象或类

显示调用，而是加载类时，或创建对象时隐式调用。

基本语法

[修饰符]{

代码

}；

注意：

1. 修饰符 可选，要写的话，也只能写static
2. 代码块分为两类：

使用static修饰的叫静态代码块，没有static修饰的，叫普通代码块。

1. 逻辑语句可以为任何逻辑语句（输入、输出、方法调用、循环、判断等）
2. ；号可以写上，也可以省略。

代码块的好处：

1. 相当于另外一种形式的构造器（对构造器的补充机制），可以做初始化的操作
2. 场景：如果多个构造器中都有重复的语句，可以抽取到初始化块中，这样当我们不管调用哪个构造器，创建对象，都会先调用代码块的内容。代码块调用的顺序优于构造器。从而提高代码的重用性。

代码块的注意事项和使用细节

1. static代码块也叫静态代码块，作用就是对类进行初始化，而且它随着类的加载而执行，并且只会执行一次。如果是普通代码块，每创建一个对象，就执行。
2. 类什么时候被加载
   1. 创建对象实例时（new）
   2. 创建子类对象实例，父类也会被加载
   3. 使用类的静态成员时（静态属性，静态方法）

案例演示：A类extends B类的静态块

1. 普通的代码块，在创建对象实例时，会被隐式的调用。被创建一次，就会调用一次。

如果只是使用类的静态成员时，普通代码块并不会执行（需要创建对象才能使用）。简单来说：普通代码块是构造器的补充。

小结：1.static代码块是类加载时执行的，只会执行一次。2.普通代码块是在创建对象时调用的，创建一次，调用一次。3.类加载的3种情况，需要记住。

1. 创意一个对象，在一个类 调用顺序是：①调用静态代码块和静态属性初始化（注意：静态代码块和静态属性初始化调用的优先级一样，如果有多个静态代码块和多个静态变量初始化，则按他们定义的顺序调用。）②调用普通代码块和普通属性的初始化（注意：普通代码块和普通属性初始化调用的优先级一样，如果有多个普通代码块和多个普通变量初始化，则按定义的顺序调用。）③最后调用构造方法（构造器）。（一定要记住！①静态相关的代码块，属性初始化，在类加载时，就执行完毕，静态是最先的，因此是优先于构造器和普通代码块执行的）
2. 构造器的最前面其实隐含了（1）super（）和（2）调用普通代码块
3. 我们看一下创建一个子类对象时（继承关系），他们的静态代码块，静态属性初始化，普通代码块，普通属性初始化，构造方法的调用顺序如下：
   1. 父类的静态代码块和静态属性（优先级一样，按定义顺序执行）
   2. 子类的静态代码块和静态属性（优先级一样，按定义顺序执行）
   3. 父类的普通代码块和普通属性初始化（优先级一样，按定义顺序执行）
   4. 父类构造方法（构造器）
   5. 子类的普通代码块和普通属性初始化（优先级一样，按定义顺序执行）
   6. 子类的构造方法

7.静态代码块只能直接调用静态成员（静态属性和静态方法），普通代码块可以调用任意成员。

单例设计模式

什么是设计模式（Java中有23种）

1. 静态方法和属性的经典使用
2. 设计模式是大量的实践中总结和理论化之后优选的代码结构、编程风格、以及解决问题的思考方式。设计模式就像是经典的棋谱，不同的棋局，我们用不同的棋谱，免去我们自己再思考和摸索

什么是单例模式

单例（单个的实例）

1. 所谓类的单例设计模式，就是采取一定的方法保证在整个的软件系统中，对某个类只能存在一个对象实例，并且该类只提供一个取得其对象实例的方法。
2. 单例模式有两种方式：1.饿汉式2.懒汉式

饿汉式单例模式的实现

步骤如下：（如何保证我们只能创建一个实例对象）

1. 构造器私有化（private只能在本类调用） ——>防止直接new
2. 类的内部创建对象（该对象是static）
3. 类种提供一个公共的静态方法（ public static ）。getinstance（获取实例）
4. 代码实现

小结：饿汉式单例模式，有时即使没有使用实例对象，也已经创建好了。

懒汉式单例模式

1. 构造器私有化
2. 定义一个static静态属性对象
3. 提供一个公共的静态方法，可以返回一个类的对象
4. 懒汉式，只有当用户使用getinstance时，才返回类的对象，后面再次调用时，会返回上次创建的类的对象，从而保证了单例。

饿汉式vs懒汉式

1. 二者最主要的区别在于创建对象的时机不同：饿汉式是在类加载就创建了对象实例，而懒汉式是在使用时才创建。
2. 饿汉式不存在线程安全问题，懒汉式存在线程安全问题。
3. 饿汉式存在浪费资源的可能。因为如果程序员一个对象实例都没有使用，饿汉式创建的对象就浪费了，懒汉式是使用时才创建，就不存在这个问题。
4. 在JavaSE标准类中，java.lang.Runtime就是经典的单例模式。

Final 关键字

Final可以修饰类、属性、方法和局部变量

在某些情况下，就会使用final：

1. 当不希望类被继承时，可以用final修饰
2. 当不希望父类的某个方法被子类覆盖/重写（override）时。
3. 当不希望类的某个属性的值被修改，可以用final关键字修饰
4. 当不希望类的某个局部变量被修改，可以用final关键字修饰

Final的注意事项和细节

1. final修饰的属性又叫常量，一般用XX\_XX\_XX来命名
2. final修饰的属性在定义时，必须赋初值，并且以后不能再修改，赋值可以在如下位置之一：①定义时：如public final double TAX\_RATE=0.08;②在构造器中③在代码块中
3. 如果final修饰的属性是静态的，则初始化的位置只能是：

①定义时②在静态代码块 不能在构造器中赋值。

1. Final类不能继承，但是可以实例化对象。
2. 如果类不是final类，但是含有final方法，则该方法虽然不能重写，但是可以被继承。
3. 一般来说， 如果一个类已经是final类了，就没有必要再将方法修饰成final方法。
4. Final不能修饰构造方法
5. Final和static往往搭配使用，效率更高，不会导致类加载，底层编译器做了优化处理
6. 包装类（Integer，Double，Float，Boolean等都是final），String也是final类。

抽象类的引入：当父类的某些方法，需要声明，但是又不确定如何实现时，可以将其声明为抽象方法（就是没有实现的方法，没有实现就是指没有方法体{}），那么这个类就是抽象类。当一个类中存在抽象方法时，需要将该类声明为abstract类

一般来说，抽象类会被继承，由它的子类实现抽象方法。

抽象类介绍

1. 用abstract关键字来修饰一个类，这个类就叫抽象类 访问修饰符 abstract 类名{ }
2. 用abstract关键字来修饰一个方法，这个方法就叫抽象方法， 访问修饰符 abstract 返回类型 方法名 （参数列表）；//没有方法体
3. 抽象类的价值更多作用是在于设计，是设计者设计好后，让子类继承并实现抽象类（）
4. 抽象类，在框架和设计模式使用较多。

抽象类注意事项和细节

1. 抽象类不能被实例化
2. 抽象类不一定要包含abstract方法。也就是说，抽象类可以没有abstract方法。
3. 一旦类包含abstract方法，则这个类必须声明为abstract类。
4. Abstract只能修饰类和方法，不能修饰属性和其它的。
5. 抽象类可以有任意成员（抽象类本质还是类），比如：非抽象方法、构造器、静态属性…
6. 抽象方法不能有主体{ }，即不能实现。
7. 如果一个类继承了抽象类，则它必须实现抽象类的所有抽象方法，除非它自己也声明为abstract类。
8. 抽象方法不能使用private、final和static修饰，因为这些关键字和重写是相违背的。

抽象类最佳实践—模板设计模式

接口

基本介绍

接口就是给出一些没有实现的方法，封装到一起，到某个类要使用的时候，在根据具体情况把这些方法写出来，语法：

Interface 接口名{

//属性

//方法

}

Class 类名 implements 接口{

自己属性；

自己方法；

必须实现的接口的抽象方法

}

小结：1.在jdk7.0前，接口里的所有方法都没有方法体。（都是1.抽象方法）2.jdk8.0后接口可以有2.静态方法，3.默认方法（需要使用default关键字修饰），接口中可以有方法的具体实现。

在接口中抽象方法可以省略abstract关键字

接口的注意事项和细节

1. 接口不能被实例化
2. 接口中的方法都是public方法，接口中的抽象方法，可以不用abstract修饰。
3. 一个普通类实现接口，就必须将该接口所有方法都实现。
4. 抽象类去实现接口时，可以不用实现接口的抽象方法。
5. 一个类同时可以实现多个接口
6. 接口中的属性，只能是final的，而且是public static final修饰符。比如 int n1=10，实际是public static final int n1 = 10；
7. 接口中的属性访问形式：接口名.属性名
8. 接口不能继承其它的类，但是可以继承多个个别的接口
9. 接口的修饰符 只能是public 和 默认 ，这点和类的修饰符是一样的。

接口vs继承类（Interface vs Extends）

Java提供的实现机制是对Java单继承机制的补充。（eg.：小猴子继承猴子的能力，如果小猴子想游泳就需要像鱼一样实现游泳功能）

小结：当子类继承了父类，就自动的拥有父类的功能，如果子类需要扩展功能，可以通过实现接口的方式扩展。

接口和继承解决的问题不同

继承的价值主要在于：解决代码的复用性和可维护性。

接口的价值主要在于：设计，设计好各种规范（方法），让其他类去实现这些方法，可以更加灵活，继承是满足 is — a 的关系，而接口只需满足 like — a 的关系。

接口在一定程度上实现代码解耦[接口规范性+动态绑定机制]

接口的多态特性

1. 多态参数：在USB接口案例，Usb，即可以接收手机（实现了USB接口）对象，又可以接收相机（实现了USB接口）对象，就体现了接口多态（接口引用可以指向实现了接口的类的对象）。
2. 多态数组
3. 接口存在多态传递现象//接口类型的变量可以指向，实现了该接口的类的对象实例。

内部类（是难点也是重点，底层源码，有大量的内部类）

基本介绍：一个类的内部又完整的嵌套了另一个类结构。被嵌套的类称为内部类（inner class），嵌套其他类的类称为外部类（outer class）。是我们类的第五大成员（类的五大成员？属性、方法、构造器、代码块、内部类），内部类最大的特点就是可以直接访问私有属性，并且可以体现类与类之间的包含关系。 基本语法：

Class Outer { //外部类

Class Inner { //内部类

}

}

Class Other { //外部其他类

}

内部类一共有四种，分为两类。

1. 定义在外部类局部位置上（比如方法内）：①局部内部类（有类名）②匿名内部类（没有类名，重点！！！）
2. 定义在外部类的成员位置上：①成员内部类（没用static修饰）②静态内部类（使用static修饰）

局部内部类

说明：局部内部类是定义在外部类的局部位置，比如方法中，并且有类名。

1. 可以直接访问外部类的所有成员，包含私有的
2. 不能添加访问修饰符，因为它的地位就是一个局部变量，局部变量是不能使用修饰符的。但是可以使用final修饰，因为局部变量也可以使用final
3. 作用域：仅仅在定义它的方法或代码块中。
4. 局部内部类---访问--->外部类的成员[访问方式：直接访问]
5. 外部类---访问-🡪局部内部类的成员 访问方式：创建对象，再访问（注意：必须在作用域中）

记住：（1）局部内部类定义在方法中/代码块（2）作用域在方法体或者代码块中（3）本质仍然是一个类

1. 外部其他类---不能访问---🡪局部内部类（因为局部内部类地位是一个局部变量）
2. 如果外部类和局部内部类的成员重名时，局部内部类访问的话，默认遵循就近原则，如果想访问外部类的成员，则可以使用（外部类名.this.成员）去访问

解释：外部类名.this：本质就是外部类的对象。

匿名内部类的使用 1.本质是类（2）内部类 （3）该类没有名字（实际上有名字，由系统分配的名字，我们看不到）(4)同时还是一个对象

说明：匿名内部类是定义在外部类的局部位置，比如方法中，并且没有类名

1. 匿名内部类的基本语法

new类或接口（参数列表）{

类体

}；

匿名内部类存在的需求：1.有些类只是使用一次，后面再不使用（传统方法太啰嗦）。2.使用匿名内部类来简化开发。3.匿名内部类使用一次，就不能再使用。

匿名内部类注意事项和使用细节

1. 匿名内部类的语法比较奇特，因为匿名内部类既是一个类的定义，同时它本身也是一个对象，因此从语法上看，它既有定义类的特征，也有创建对象的特征。
2. 可以访问外部类的所有成员，包含私有的
3. 不能添加修饰符
4. 作用域：仅仅定义在它的方法或代码块中
5. 匿名内部类---访问-🡪外部类成员[访问方式：直接访问]
6. 外部其他类---不能访问---🡪匿名内部类
7. 如果外部类和匿名内部类的成员重名时，匿名内部类访问的话，默认遵循就近原则，如果想访问外部类的成员，则可以使用（外部类名.this.成员）去访问

匿名内部类的最佳实践

当做实参直接传递，简洁高效

成员内部类

说明：成员内部类是定义在外部类的成员位置，并且没有static修饰。

1. 可以直接访问外部类的所有成员，包含私有的
2. 可以添加任意访问修饰符（public、protected、默认、private），因为它的地位就是一个成员
3. 作用域和外部类的其他成员一样，为整个类体。案例，在外部类的成员方法中创建成员内部类对象，再调用方法。
4. 成员内部类---访问-🡪外部类（eg.属性）【访问方式：直接访问】
5. 外部类---访问---🡪内部类（说明）访问方式：创建成员内部类的对象，再访问
6. 外部其他类---访问-🡪成员内部类 两种方法：①在外部其他类创建一个内部类对象，注意语法就行，不用特别纠结。②在外部类，编写一个方法，可以返回成员内部类对象。
7. 如果外部类和内部类的成员重名时，内部类访问的话，默认遵循就近原则，如果想访问外部类的成员，则可以使用（外部类名.this.成员）去访问。

静态内部类

说明：静态内部类是定义在外部类的成员位置，并且有static修饰。

1. 可以直接访问外部类的所有静态成员，包含私有的，但不能直接访问非静态成员。
2. 可以添加任意访问修饰符（public、protected、默认、private），因为它的地位就是一个成员
3. 作用域和外部类的其他成员一样，为整个类体。
4. 静态内部类---访问-🡪外部类（eg.静态属性）【访问方式：直接访问所有静态成员】
5. 外部类---访问---🡪静态内部类访问方式：创建静态内部类的对象，再访问
6. 外部其他类---访问-🡪静态内部类 方法和成员内部类类似，注意静态的特性就行。
7. 如果外部类和静态内部类的成员重名时，静态内部类访问的话，默认遵循就近原则，如果想访问外部类的成员，则可以使用（外部类名.成员）去访问。

枚举和注解

枚举类引入：传统方法设计季节类可能会造成季节的混乱，四个以上的季节，这显然不合适。因此引入枚举类（enumeration，简称enum），枚举是一组常量的集合，可以这样理解：枚举属于一种特殊的类，里面只包含一组有限的特定的对象。

枚举的两种实现方式

1. 自定义类实现枚举
2. 使用enum关键字实现枚举

自定义类实现枚举

1. 在定义类中，将构造器私有化，防止直接 new
2. 可以提供get方法，但是去掉set方法，防止属性被修改，枚举对象通常为只读
3. 在类内部，直接创建固定的对象
4. 对外暴露对象（通过为对象添加public final static 修饰符）
5. 枚举对象名通常使用全部大写，常量的命名规范
6. 枚举对象根据需要，也可以有多个属性。

Enum实现枚举类

1. 使用关键字enum 替代 class
2. 将自定义类中创建对象的对象名（常量名）直接使用为 常量名（实参列表）
3. 如果有多个常量（对象），使用逗号间隔即可
4. 如果使用enum来实现枚举，要求将定义常量对象，写在前面

Enum关键字实现枚举注意事项

1. 当我们使用enum关键字开发一个枚举类时，默认会继承Enum类而且是个final类
2. 如果我们使用的是无参构造器，创建常量对象，则可以省略（）

说明：enum关键字使用时，会隐式继承Enum类，这样我们就可以使用Enum类相关的方法。

Enum实现接口

1. 使用enum关键字后，就不能再继承其他类了，因为enum会隐式继承enum，而Java是单继承机制。
2. 枚举类和普通类一样，可以实现接口，如下形式：

enum 类名 implements 接口1，接口2 { }

注解的理解

1. 注解（Annotation）也被称为元数据（Metadata），用于修饰解释 包、类、方法、属性、构造器、局部变量等数据信息。
2. 和注释一样，注解不影响程序逻辑，但注解可以被编译或运行，相当于嵌入在代码中的补充信息。
3. 在JavaSE中，注解的使用目的比较简单，例如标记过时的功能，忽略警告等。在JavaEE中，注解占据了更重要的角色，例如用来配置应用程序的任何切面，代替JavaEE旧版中所遗留的繁冗代码和XML配置等。

基本的Annotation介绍

使用Annotation时要在其前面增加@符号，并把该Annotation当成一个修饰符使用。用于修饰它支持的程序元素

三个基本的Annotation：

[1.@override](mailto:1.@override)：限定某个方法，是重写父类方法，该注解只能用于方法

[2.@Deprecated](mailto:2.@Deprecated)（意思：已废弃）:用于表示某个程序元素（类，方法等）已过时

[3.@SuppressWarnings](mailto:3.@SuppressWarnings)（suppress意思：抑制）：抑制编译器警告

Override使用说明

[1.@override](mailto:1.@override)表示指定重写父类的方法（从编译层面验证），如果父类没有子类重写的方法，则会报错。

2．[如果不写@override](mailto:2.如果不写@override)注解，而父类仍有子类的重写方法，仍然构成重写

[3.@override](mailto:3.@override)只能修饰方法，不能修饰其他类，包，属性等等。

1. [查看@override](mailto:4.查看@override)注解源码为@Target（ElementType.METHOD）,说明只能修饰方法
2. @Target是修饰注解的注解，称为元注解

@Deprecated 修饰某个元素，表示该元素已经过时了，即不在推荐使用，但是仍然可以使用。

1. [查看@Deprecated](mailto:1.查看@Deprecated) 注解类的源码
2. 可以修饰方法，类，字段，包，参数 等等
3. @Deprecated可以做版本升级过渡使用

@SuppressWarnings：抑制编译器警告

1. 当我们不希望看到这些警告的时候，可以使用SuppressWarnings注解来抑制警告信息
2. 在@SuppressWarnings {“”,“”，“”,…}中，可以写入你希望抑制（不显示）警告信息
3. 关于SuppressWarnings作用范围是和你放置的范围有关
4. 关于“ ”中的各种值
5. unchecked是忽略没有检查的警告
6. rawtype是忽略没有指定泛型的警告
7. unused是忽略没有使用某个变量的警告
8. @SuppressWarnings可以修饰的程序元素为{TYPE,FIELD,METHOD,…}，请查看@Target
9. 生成@SuppressWarnings时，直接查看右侧黄色提示

异常处理Exception

Java设计者提供了一个叫异常处理机制来解决程序易崩溃问题（程序不够健壮）。

如果程序员，认为一个代码可能出现异常/问题，可以使用try-catch异常处理机制来解决，从而保证程序的健壮性。具体处理方法，将该代码块->选中->快捷键 ctrl + alt + t —>选中 try-catch

如果进行异常处理，那么即使出现了异常，程序可以继续执行。

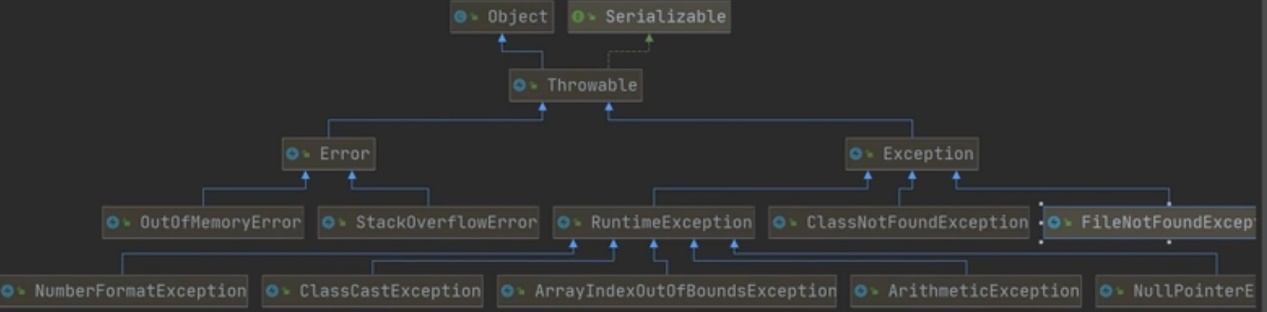
基本概念：

Java语言中，将程序执行中发生的不正常情况称为“异常”。（开发过程中的语法错误和逻辑错误不是异常）

执行过程中所发生的异常事件可分为两类

1. Error（错误）：Java虚拟机无法解决的严重问题，程序崩溃，直接挂掉。如：JVM系统内部错误、资源耗尽等严重情况。比如：StackOverflowError和OOM（out of memory），Error是严重错误，程序会崩溃。
2. Exception：其它因编程错误或偶然的外在因素导致的一般性问题（小问题），可以使用针对性的代码处理。例如空指针异常、网络连接中断、试图读取不存在的文件等等，Exception分为两大类：运行时异常[程序运行时，发生的异常]和编译时异常[编程时，编译器检查出的异常]。

异常体系图



1. Exception分为两大类，运行时异常和编译时异常。
2. 运行时异常，编译器检查不出来，一般是指编程时的逻辑错误，是程序员应该避免其出现的异常。
3. 对于运行时异常，可以不作处理，因为这类异常很普遍，若全处理可能会对程序的可读性和运行效率产生影响。
4. 编译时异常，是编译器要求必须处置的异常。

五大常见运行时异常

1. NullPointerException 空指针异常

当应用程序试图在需要对象的地方使用null时，抛出该异常。（对象还没创建出来就使用了，会抛出该异常）

1. ArrayIndexOutOfBoundsException 数组下标越界异常
2. ClassCastException（类转换异常）

当试图将对象强制转换为不是实例的子类时，抛出该异常。

1. NumberFormatException数字格式不正确异常

当应用程序试图将字符串转换成一种数值类型，但该字符串不能转换为适当格式时，抛出该异常—>使用异常我们可以确保输入是满足条件数字。

5.算术异常

常见的编译异常

SQLException//操作数据库时，查询表可能发生异常

IOException//操作文件时，发生的异常

FileNotFoundException//当操作一个不存在的文件时，发生异常

ClassNotFoundException//加载类，而该类不存在时，异常

EOFException //操作文件，到文件末尾，发生异常

ILeagalArguementException //参数异常

异常处理

异常处理的方式

1. try-catch-finally

程序员在代码中捕获发生的异常，自行处理

try{

代码/可能有异常

}catch（Exception e）{

//捕获到异常

1.当异常发生时，系统将异常封装成Exception 对象e ，传递给catch

2.得到异常对象后，程序员自己处理

3.注意，如果没有发生异常catch代码块不执行

}finally{

1. 不管try代码块是否有异常发生，始终要执行finally
2. 所有，通常将释放资源的代码，放在finally

}

try-catch方式处理异常-注意事项

* 1. 如果异常发生了，则异常发生后面的代码不会执行，直接进入到catch块。
  2. 如果异常没有发生，则顺序执行try的代码块，不会进入catch.
  3. 如果希望不管发生什么，都执行某段代码（比如关闭连接，释放资源）使用finally{ }
  4. 可以有多个catch语句，捕获不同的异常，要求父类异常在后，子类异常在前，比如(Null\_Pointer\_Exception在前，Exception在后)如果发生异常，只会匹配一个catch
  5. 可以进行try-finally配合使用，这种用法相当于没有捕获异常，因此程序会直接崩掉。应用场景，执行一段代码，不管是否发生异常，都必须执行某个业务逻辑。

try{

//代码…

}

finally{

//总是执行

}

1. throws（投掷、抛出）

将发生的异常抛出，交给调用者（方法）来处理，最顶级的处理者就是JVM

基本介绍

1.如果一个方法（中的语句执行时）可能生成某种异常，但是并不能确定如何处理这种异常，则此方法应显示地声明抛出异常，表明该方法将不对这些异常进行处理，而由该方法的调用者负责处理。

2.在方法声明中throws语句可以声明抛出异常的列表，throws后面的异常类型可以是方法中产生的异常类型，也可以是它的父类。



T throws使用细节

1. 对于编译异常，程序中必须处理，比如try-catch或者throws
2. 对于运行时异常，程序中如果没有处理，默认就是throws的方式处理
3. 子类重写父类的方法时，对抛出异常的规定：子类重写的方法，所抛出的异常类型要么和父类抛出的异常一致，要么为父类抛出的异常的类型的子类型。
4. 在throws过程中，如果有方法try-catch，就相当于处理异常，就可以不必throws

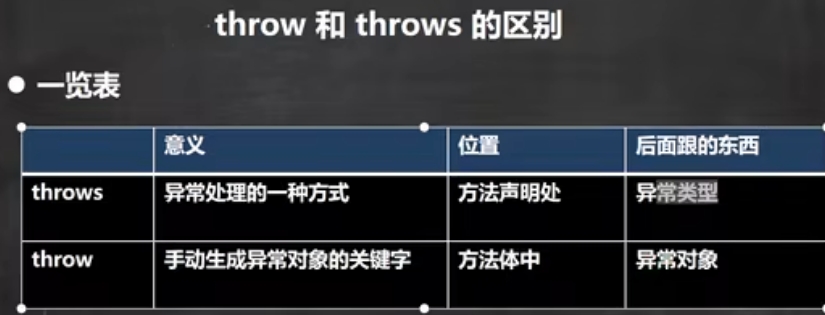
自定义异常

当程序中出现了某些“错误”，但该错误信息并没有在Throwable子类中描述处理，这个时候可以自己设计异常类，用于描述该错误信息。

自定义异常的步骤

1. 定义类：自定义异常类名继承Exception或RuntimeException
2. 如果继承Exception，属于编译异常
3. 如果继承RuntimeException，属于运行异常（一般来说，都是继承RuntimeException，把自定义异常做成运行时异常，好处是，我们可以使用默认的异常处理机制）

Throws vs throw



包装类Wrapper类

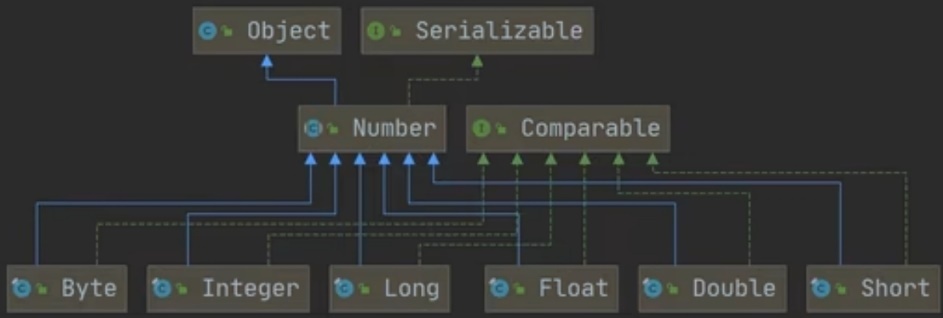
包装类的分类

1. 针对八种基本数据类型相应的引用类型—包装类
2. 有了类的特点，就可以调用类的方法。



Boolean类和Character类继承Object类，实现Comparable接口和Serializable接口

黄颜色包装类父类（Number类）



包装类和基本数据类型的相互转换

1. jdk5前的手动装箱和拆箱方式，装箱：基本类型→包装类型，反之拆箱。
2. jdk5以后的自动装箱和自动拆箱方式
3. 自动装箱底层调用的是valueOf方法，比如Integer.valueOf()

// jdk5之前的手动装箱  
// int->Integer  
 int n1=100;  
 Integer integer=new Integer(n1); 方法①  
 Integer integer1=Integer.*valueOf*(n1); 方法②  
// 手动拆箱  
// Integer->int  
 int i=integer.intValue();  
// jdk5之后的自动装箱  
// int->Integer  
 int n2=1;  
 Integer INTEGER =n2;  
// 自动拆箱  
// Integer->int  
 int n3=INTEGER;

包装类型和String类型转换

// 包装类->String  
 Integer i=100;  
// 方法一  
 String str1=i+"";  
// 方法二  
 String str2=i.toString();  
// 方法三  
 String str3=String.*valueOf*(i);  
  
// String->包装类  
 String str4="123";  
 // 方法①  
 Integer i1=Integer.*parseInt*(str4);//使用到自动装箱  
 // 方法②  
 Integer i2=new Integer(str4);//构造器

记得看包装类的源码

String类

String类的理解和创建对象

1. String对象用于保存字符串，也就是一组字符序列
2. 字符串常量对象是用双引号括起的字符序列。Eg.“你好”、“Hello”。
3. 字符串的字符使用Unicode字符编码，一个字符占两个字节。
4. String类较常用的构造方法：

String s1 = new String();

String s2 = new String(String original);

String s3 = new String(char[] a);

String s4 = new String(char[] a,int startIndex,int count);

String s5 = new String(byte[] b);

1. String 实现了 Serializable接口，说明String可以串行化 String对象可以在网络上进行传输。String 实现了 Comparable接口，说明String对象可以比较大小。
2. String 是final 类，不能被其他类继承
3. String 有属性 private final char value[]；//用于存放字符串内容
4. Value是final类型，(内容可以修改)地址不可修改

创建String对象的两种方式

方式①：直接赋值String s = “lwt”；

方式②：调用构造器String s = new String（“lwt”）；

1. 方式一：先从常量池查看是否有“lwt”数据空间，如果有，直接指向；如果没有则重新创建，然后指向。s最终指向的是常量池的空间地址。
2. 方式二：先在堆中创建空间，里面维护了value属性，指向常量池的lwt空间。如果常量池没有“lwt”，重新创建，如果有，直接通过value指向，最终指向的是堆中的空间地址。

重要规则：String s1 = “a”+“b”；常量相加，看的是池。String s2 = a + b；变量相加，是在堆中。

String类是保存字符常量（final）的，每次更新都需要重新开辟空间，效率较低，因此Java设计者还提供了StringBuilder和StringBuffer来增强String的功能，并提高效率。

String format方法 格式字符串

占位符

1.%s,%d,%.2f,%c称为占位符

2.这些占位符由后面变量来替换

3.%s 表示后面由字符串来替换

4.%d 表示整数来替换

5.%.2f 表示使用小数来替换，替换后，只会保留小数点后两位，并进行四舍五入处理。

6.%c 使用char类型来替换

String s1 = String.format(“我的姓名是%s 年龄是%d 成绩是%.2f 性别是%c 希望大家喜欢我”,name,age,score,gender)

StringBuffer类

Java.lang. StringBuffer代表可变的字符序列，可以对字符串内容进行增删

很多方法与String相同，但是StringBuffer是可变长度的。

StringBuffer是一个容器

1. StringBuffer的直接父类是AbstractStringBuilder
2. StringBuffer实现了Serializable，即StringBuffer的对象可以串行化。
3. 在父类中AbstractStringBuilder有属性char[] vaule,不是final修饰的

该value数组存放字符串内容，放置在堆中。不再是常量池了。

1. StringBuffer是一个final类，不能被继承。

Stiring vs StringBuffer

1. String保存的是字符串常量，里面的值不能更改，每次String类的更新实际上就是更改地址，效率较低。//private final char value[]；
2. StringBuffer保存的是字符串变量，里面的值可以更改，每次StringBuffer的更新实际上可以更新内容，不用更新地址，效率较高。//放在堆中 char value[]

String与StringBuffer之间的转换

// String->StringBuffer  
// 方式一 使用构造器  
// 注意:返回的才是SringBuffer对象，对string对象 str本身没有影响  
 String str ="lwt";  
 StringBuffer stringBuffer = new StringBuffer(str);  
// 方式二 使用append()方法  
 StringBuffer stringBuffer1 = new StringBuffer();  
 stringBuffer1.append(str);  
  
// StringBuffer->String  
// 方式一 使用StirngBuffer提供的 toString()方法  
 StringBuffer stringBuffer2 = new StringBuffer(str);  
 String str1 = stringBuffer2.toString();  
// 方式二 使用构造器  
 String str2 = new String(stringBuffer2);

StringBuilder类

1. 一个可变的字符序列，此类提供一个与StringBuffer兼容的API，但不保证同步（StringBuilder不是线程安全的）。该类被设计用作StringBuffer的一个简易替换，用在字符串缓冲区被单个线程使用的时候。如果可能，建议优先采用该类，因为在大多数实现中，它比StringBuffer要快。
2. 在StringBuilder上的主要操作是append和insert方法，可重载这些方法，来接受任意类型的数据。
3. StringBuilder的直接父类是AbstractStringBuilder
4. StringBuilder实现了Serializable，即StringBuilder的对象是可以串行化（对象可以网络传输，可以保存到文件）。
5. StringBuilder对象字符序列仍然是存放在父类中AbstractStringBuilder有属性char[] vaule,不是final修饰的
6. value数组存放字符串内容，放置在堆中。不再是常量池了。
7. StringBuilder是一个final类，不能被继承。
8. StringBuilder 的方法，没有做互斥的处理，即没有synchronized（同步） 关键字，因此在单线程的情况下使用StringBuilder

String、StringBuffer和StringBuilder比较

1. StringBuilder和StringBuffer非常类似，均代表可变的字符序列，而且方法也一样。
2. String：不可变字符序列，效率低，但是复用率高（常量池一直存在）。应用场景：如果我们字符串很少修改，被多个对象引用，使用String，比如配置信息。
3. StringBuffer：可变字符序列、效率高（增删）、线程安全。
4. StringBuilder：可变字符序列、效率最高、线程不安全。
5. String使用注意说明： String s=“abc”；//创建了一个字符串

s+=“d”；//实际上原来的“abc”字符串对象已经丢弃了，现在又产生了一个字符串s+“d”。如果多次执行这些改变串内容的操作，会导致大量副本字符串对象存留在内存中，降低效率。如果这样的操作放到循环中，会极大影响程序的性能=>结论：如果我们对String做大量修改，不要使用String。

Math类

Math包含用于执行基本数学运算的方法(指数、对数、开方)

Math类方法均为静态方法

使用Math方法，获取a-b之间的一个随机整数：

int num = （int）（Math.random()\*(b-a+1)+a）；

Arrays类

Arrays里面包含了一系列静态方法，用于管理或操作数组（比如排序和搜索）

1. toString 返回数组的字符串形式

Arrays.toString（arr）

1. sort排序（自然排序 和 定制排序（需要传入一个Comparator接口）详细内容P482
2. binarySearch通过二分搜索法进行查找，要求必须排好序

int index = Arrays.binarySearch(arr,3);

* 1. 使用binarySearch 二叉查找
  2. 要求该数组是有序的，如果该数组是无序的，不能使用binarySearch

1. copyOf数组元素的复制

eg. Integer[] newArr = Arrays.copyOf(arr,arr.length)//从arr数组中，拷贝arr.length个元素到newArr数组中 底层是System.arraycopy

如果拷贝的长度 > arr.length 就在新数组的后面 增加 null

如果拷贝的长度小于0就抛出NegativeArraySizeException异常

1. 数组元素的填充

Arrays.fill(arr,\_\_) \_\_填充元素

//使用\_\_ 去填充arr数组

1. Arrays.equals(arr1,arr2) //比较两个数组元素内容是否完全一致 返回布尔值
2. asList将一组值，转换成list

List<Integer>asList = Arrays.asList(2,3,4,5,6,1);//asList方法会将(2,3,4,5,6,1)数据转成一个List集合（一组数据）

System.out.println(“asList=”+asList);

System类 系统类

常见方法和案例

1. exit 退出程序 System.exit(0); 0是状态值表示正常退出，
2. arraycopy：复制数组元素，比较适合底层调用，一般使用Arrays.copyOf 完成调用
3. currentTimeMillens:返回当前时间距离1970-01-01的毫秒数
4. gc：运行垃圾回收机制System.gc();

大数处理方案 BigInteger和 BigDecimal类

应用场景：

1. BigInteger适合保存比较大的整型//当我们编程时，需要处理很大的整数，long 不够用 可以使用BigInteger来搞定

2. BigDecimal适合保存精度更高的浮点型（小数）

BigInteger和 BigDecimal 常见方法

1. add 加
2. subtract 减
3. multiply 乘
4. divide 除
   1. 在对BigInteger进行加减乘除的时候，需要使用对应的方法，不能直接使用 + - \* /
   2. 可以创建一个 要操作的BigInteger然后进行操作

日期类 第一代日期类Date

1. Date：精确到毫秒，代表特定的瞬间
2. SimpleDateFormat：格式和解析日期的类 SimpleDateFormat格式化和解析日期的具体类。它允许进行格式化（日期->文本） 解析（文本→日期）和规范化
3. Date 类实践

Date date = new Date（）；//①.获取当前系统时间。②.Date类 是在java.util包中

* 1. .默认输出的日期格式是国外的格式

4.创建SimpleDateFormat，可以指定相应的格式 这里格式使用的字母是规定好的，不能乱写 ：SimpleDateFormat sdf = new SimpleDateFormat（“yyyy年MM月dd日 hh:mm:ss E”）

Srtirng format = sdf.format(date);//format 将日期转换成指定格式的字符串

System.out.println(“更新格式后的日期：”+format)；

// 1.可以把一个格式化的String 转成对应的Date  
// 2.得到Date输出时仍然还是国外的日期格式，如果希望指定格式输出，需要转换  
// 3.在把String→Date，使用的sdf格式需要和给出的String格式一样，否则会抛出ParseException异常  
 String date01 = "02月24日 00:24:24 星期一 ";  
 Date parse = sdf.parse(date01);  
 System.*out*.println("打印时间是："+parse);

Calendar 第二代日期类

1. Calendar是一个抽象类，并且构造器是private new Calendar创建不了对象。
2. 可以通过 getInstance（）来获取实例
3. 提供大量的方法和字段提供给程序员
4. Calendar没有提供对应的格式化的类，因此需要程序员自己组合来输出。
5. 如果需要按照24小时进制来获取时间， Calendar.HOUR 改成=> Calendar.HOUR\_OF\_DAY

第三代日期类 JDK8 常见方法 ①LocalDate（日期） ②LocalTime（时间）③LocalDateTime

前面两代日期类的不足

JDK1.0中包含Date类，但是它的大多数方法已经在JDK1.1引入Calendar类之后被弃用了。而Calendar也存在问题是：

1. 可变性：像日期和时间这样的类应该是不可变的
2. 偏移性：Date中的年份是从1970开始的，而Calendar月份从0开始
3. 格式化：只有Date类使用，Calendar则没有使用
4. 此外他们也不是线程安全的，还不能处理闰秒等（每隔两天，多出1s）

LocalDateTime（年月日时分秒）

LocalDateTime localDateTime = LocalDateTime.now();  
System.*out*.println(localDateTime);  
System.*out*.println(localDateTime.getDayOfMonth());  
System.*out*.println(localDateTime.getHour());  
System.*out*.println(localDateTime.getMinute());  
LocalDate ld = LocalDate.now();  
System.*out*.println(ld);  
LocalTime lt =LocalTime.now();  
System.*out*.println(lt);

1. 使用now（）返回表示当前日期时间的对象
2. 使用DateTimeFormatter 对象来进行格式化 //创建DateTimeFormatter对象

DateTimeFormatterdateTimeFormatter

= DateTimeFormatter.ofPattern(“yyyy年MM月dd日HH时mm分ss秒”);

Stirng format =dateTimeFormatter.format(localdatetime);

Instant时间戳

类似于Date 提供了一系列和Date类转换的方式

Instant→Date；//1.通过静态方法now（）获取表示当前时间戳的对象

Instant now = Instant.now();

//2.通过from可以把Instant转成Date

Date date = Date.from(Instant);

Date→Instant:

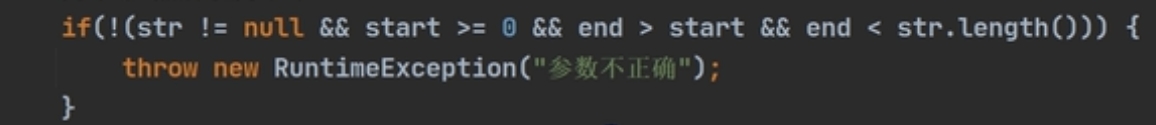
3. 通过toInstant可以把Date转成Instant对象

Instant instant = date.toInstant();

编程技巧

对输入参数做一个验证

1. 写出正确的情况
2. 然后取反即可。
3. 这样写，思路不会乱



集合的理解和好处

前面我们保存多个数据使用的是数组，但是数组有不足的地方

数组

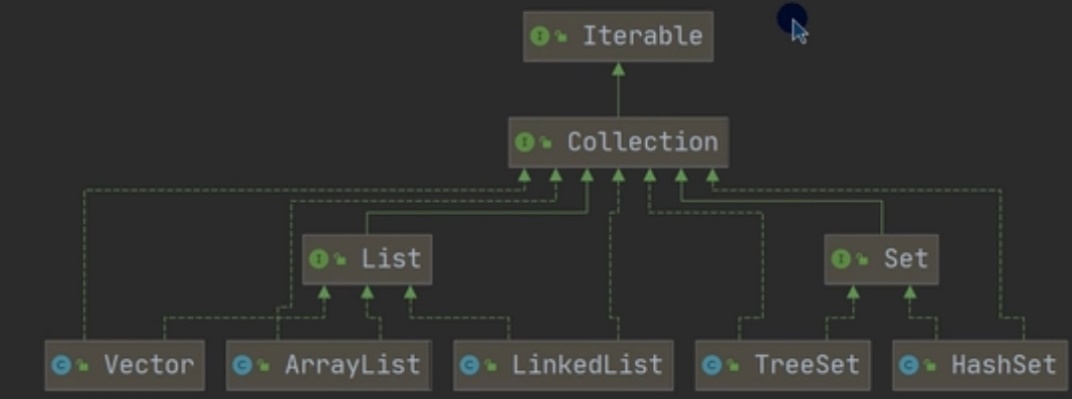
1. 数组长度开始时必须指定，而且一旦指定，就不能更改
2. 保存的必须为同一类型的元素
3. 使用数组进行增加/删除元素，比较麻烦。

集合

1. 可以动态保存任意多个元素，使用比较方便
2. 提供了一系列方便的操作元素的方法：add、remove、set、get等
3. 使用集合添加，删除新元素，简洁了

集合框架体系图

1. 集合主要是两组（单列集合，双列集合）
2. Collection 接口有两个重要的子接口List Set ，他们的实现子类都是单列集合
3. Map 接口的实现子类 是双列集合，存放K-V



Collection接口实现类的特点  
public interface Collection<E> extends Iterable<E>

1. Collection实现子类可以存放多个元素，每个元素可以是Object
2. 有些Collection的实现类，可以存放重复的元素，有些不可以
3. 有些Collection的实现类，有些是有序的（List），有些不是有序的（Set）
4. Collection接口没有直接的实现子类，是通过它的子接口Set和List实现的
5. Collection接口常用方法（接口不能直接被实例化，所以使用Collection接口的实现子类（ArrayList）来举例说明Collection接口方法）
   1. add:添加单个元素（单列集合） 可以是任意元素
   2. remove：删除指定元素
   3. contains：查找元素是否存在
   4. size：获取元素个数
   5. isEmpty：判断是否为空
   6. clear：清空
   7. addAll：添加多个元素
   8. containsAll：查找多个元素是否都存在
   9. removeAll：删除多个元素
6. Collection接口遍历元素方式1-使用Iterator（迭代器）

Ⅰ.Iterator对象称为迭代器，主要用于遍历Collection集合中的元素

Ⅱ.所以实现了Collection接口的集合类都有一个iterator（）方法，用以返回一个实现了Iterator接口的对象，即返回一个迭代器

Ⅲ.Iterator的结构

Ⅳ.Iterator仅用于遍历集合，Iterator本身并不存放对象。

在调用iterator.next()方法//1.指针下移2.将下移以后集合位置上的元素返回。之前必须要调用iterator.hasNext()进行检测。若不调用，且下一条记录无效，直接调用next()方法会抛出NoSuchElementException异常

//1.遍历集合，先创建一个collection对应的迭代器  
 Iterator iterator = collection.iterator();

2.使用while循环遍历  
 while (iterator.hasNext()){//判断是否还有数据  
 //返回下一个元素，类型是Object  
 Object object = iterator.next();  
 System.*out*.println(object);

3.当退出while循环后，这时iterator迭代器指向最后元素

4.如果希望再次遍历，需要重置迭代器

Iterator iterator = collection.iterator();（重置指针）

Collection接口遍历元素方式2-使用增强for循环

增强for循环，可以代替Iterator迭代器。特点：增强for循环就是简化版的Iterator，本质一样。只能用于遍历集合或数组。

基本语法 for (元素类型 元素名 ：集合名或数组名){

访问元素

}

快捷键 I

List接口方法

List接口是Collection接口的子接口

1. List集合类中元素有序（即添加顺序和取出顺序一致）、且可重复
2. List集合中的每个元素都有其对应的顺序索引，即支持索引。//索引是从0开始的
3. List容器中的元素都对应一个整数型的序号记载其在容器中的位置，可以根据序号存取容器中的元素。

List接口的常用方法

void add （int index ，Object ele）：在index位置插入element

boolean addAll（int index ，Collection eles）在index位置插入elements

Object get（int index）：获取指定index位置的元素

Int indexOf（Object obj）：返回obj在集合中首次出现的位置。

Int lastindexOf（Object obj）：返回obj在集合中末次出现的位置。

Object remove（int index）：移除指定index位置的元素，并返回此元素

Object set（int index ，Object ele）：设置指定index位置的元素为ele，相当于是替换。

List sublist（int fromIndex，int toIndex）：返回从fromIndex到toIndex位置的子集合。//返回的自己和fromIndex <= sublist < toIndex

ArrayList注意事项和细节

1. permits all elements，including null ，ArrayList可以加入null，并且可以多个。
2. ArrayList是由数组来实现数据存储的
3. ArrayList基本等同于Vector，除了ArrayList是线程不安全（执行效率高），在多线程情况下，不建议使用ArrayList。

ArrayList的底层操作机制

1. ArrayList中维护了一个Object类型的数组elementData

//transient Object[] elementData transient 表示瞬间，短暂的，该属性不会被序列化。

1. 当创建ArrayList对象时，如果使用的是无参构造器，则初始elementData容量为0，1次添加，则扩容elementData为10，如需要再次扩容，则扩容elementData为1.5倍。
2. 如果使用的是指定大小的构造器，则初始elementData容量为指定大小，如果需要扩容，则直接扩容elementData为1.5倍。

Vector类

1. Vector底层也是一个对象数组，protectd Object[] elementData
2. Vector是线程同步的，即线程安全，Vector类的操作方法带有synchronized
3. 在开发中，需要线程同步安全时，考虑使用Vector.



LinkedList底层结构

1. LinkedList底层实现了双向链表和双端队列特点
2. 可以添加任意元素（元素可以重复），包括null
3. 线程不安全，没有实现同步

LinkedList的底层操作机制

1. LinkedList底层维护了一个双向链表
2. LinkedList中维护了两个属性first和last分别指向头节点和尾节点。
3. 每个节点（Node对象），里面又维护了prev、next、item三个属性，其中通过prev指向前一个，通过next指向后一个节点。最终实现双向链表。
4. 所以LinkedList的元素的添加和删除，不是通过数组完成的，相对来说效率较高。
5. 模拟一个简单的双向链表。//P0514
6. LinkedList的增删改查—看源码
7. LinkedList vs ArrayList
   1. 如果改查的操作多，选择ArrayList
   2. 如果增删的操作多，选择LinkedList
   3. 一般来说，在程序中，80%~90%都是查询，因此大部分情况下都是选择ArrayList

Set接口

基本介绍

1. 无序（添加和取出的顺序不一致），没有索引。 实现Set接口的类
2. 不允许重复元素，所以最多包含一个null
3. JDK API中Set接口的实现类有：TreeSet、HashSet。

Set接口的常用方法

和List接口一样，Set接口也是Collection的子接口，因此，常用方法和Collection接口一样。

Set接口的遍历方式

同Collection的遍历方式一样

1. 可以使用迭代器 2.增强for循环 3.不能使用索引的方式来获取。

Set接口的实现类（TreeSet、HashSet）的对象（Set接口对象），不能存放重复的元素。

Set接口对象对象存放数据是无序的（即添加的顺序和取出的顺序不一致）注意：取出的顺序虽然不是添加的顺序，但是取出后就固定了，不会一会儿一变。

Set接口的实现类-HashSet

1. HashSet实现了Set接口
2. HashSet实际上是HashMap public HashSet（）{ map = new HashMap<>(); }
3. 可以存放null值，但是只能有一个null
4. HashSet不保证元素是有序的，取决于hash后，再确定索引的结果。

HashSet底层机制

分析HashSet底层HashMap，HashMap底层是（数组+链表+红黑树）P519（老韩模拟简单的数组+链表）

为什么不把数据直接放到一个数组里面去？程序运行效率太低，（数组+链表+红黑树）存储效率非常高。

HashSet的添加元素底层是如何实现的

1. HashSet底层是HashMap
2. 添加一个元素时，先得到hash值-会转成-索引值
3. 找到存储数据表table，看这个索引位置是否已经存放的有元素
4. 如果没有，直接加入
5. 如果有，调用equals比较，如果相同，就放弃添加，如果不相同，则挂到节点后，形成一个链表。Node FirstNode = new Node（“item\_1”，null）; Node SecondNode = new Node(“item\_2”,null); FirstNode.next = SecondNode;
6. 在Java8中，如果一条链表的元素个数超过TREEIFY\_THRSEHOLD（默认是8），并且table的大小>=MIN\_TREEIFY\_CAPACITY(默认64)就会进行树化（红黑树）
7. hash值不等于hashcode，hashvalue = (object.hashcode())^(h>>>16)

(hashvalue = //异或 //无符号右移16位 object.hashCode())^(hashvalue >>> 16);

8.一个开发技巧：在需要局部（辅助）变量时，再创建。

分析HashSet的扩容和转成红黑树机制

1. HashSet底层是HashMap，第一次添加时，table数组扩容到16，临界值（threshold）是16\*加载因子（loadFactor）是0.75 = 12 （一种缓冲机制）
2. 如果table数组使用到了临界值12，就会扩容到16\*2=32，新的临界值就是32\*0.75=24。以此类推。HashSet的扩容机制，是只要加入一个节点就算一次，不论节点是加入到table数组中，还是加入进链表里。
3. 在Java8中，如果一条链表的元素个数到达TREEIFY\_THRESHOLD(默认是8)，并且table的大小>=MIN\_TREEIFY\_CAPACITY(默认64)就会进行树化（红黑树），否则仍然采用数组扩容机制。
4. 在Java中如果要让属性相同，返回相同的hash值。具体操作：alt + intsert 添加equals()and hashcode()。//如果属性相同，在使用equals时，返回true，计算hashcode（）时，返回相同的结果。如果添加对象与链表元素相比hashcode相同，但是equals//内容不同，也会添加进入链表中。具体添加逻辑查看HashSet add方法的源码。

Set接口实现类-LinkedHashSet

LinkedHashSet的全面说明

1. LinkedHashSet是HashSet的子类
2. LinkedHashSet底层是一个LinkedHashMap（HashMap的子类。），底层维护了一个

数组table+双向链表

1. LinkedHashSet根据元素的hashcode值来决定元素的存储位置，同时使用链表维护元素的次序，这使得元素看起来是以插入顺序保存的。
2. LinkedHashSet不允许添加重复元素。

LinkedHashSet的细节解读

1. LinkedHashSet中维护了一个hash表+双向链表（LinkedHashSet有head和tail）
2. 每一节点有pre和next属性，这样可以形成双向链表
3. 在添加一个元素时，先求hash值，再求索引，确定该元素在hashtable的位置，然后将添加的元素加入到双向链表（如果已经存在，不添加[原则和hashset一样]）

\_ tail.next = newElement;//从左往右读

\_ newElement.pre = tail;

\_ tail = newElement;//重置指针位置

1. 这样的话，我们遍历LinkedHashSet也能确保插入顺序和遍历顺序一致。
2. LinkedHashSet的add方法中：第一次添加一个对象时，直接将数组table扩容到16，存放的节点类型是LinkedHashMap$Entry
3. 数组是HashMap$Node[] 存放的元素/数据是 LinkedHashMap$Entry类型 数组的多态
4. LinkedHashMap$Entry要么实现HashMap$Node类，要么继承了HashMap$Node类。

Map接口和常用方法

Map接口实现类的特点[很实用]

1. Map与Collection并列存在。用于保存具有映射关系的数据：Key-Value（双列元素）
2. Map中的key 和 value 可以是任何引用类型的数据，会封装到HashMap$Node对象中。
3. Map中的key不允许重复，原因和HashSet一样（底层是通过hashcode给出的索引值）。如果出现重复的key，value会替换原来的value。
4. Map的value可以重复
5. Map的key可以为null，value也可以为null，注意key为null，只能有一个，value为null，可以多个。
6. 常用String类作为Map的key，key可以是任意对象。
7. Key和value之间存在单向一对一关系，即通过指定的key总能找到对应的value。//可以通过get方法，传入key，会返回对应的value。
8. K-v 最后是 HashMap$Node node = newNode(hash,key,value,null)
9. K-V 为了方便程序员遍历，还会创建EntrySet集合，该集合存放的元素类型Entry

// 对象就有K,V EntrySet<Entry<K,V>>

1. EntrySet中，定义的类型是Map.Entry,但是实际上存放的还是HashMap$Node.

这是因为HashMap$Node implements Map.Entry

1. 当把HashMap$Node对象 存放到EntrySet就方便我们的遍历，因为Map.Entry提供了重要的方法K getKey(); V getValue();

Map常用方法

1. put方法：添加
2. remove：根据键删除映射关系
3. get：根据键获取值
4. size：获取元素个数
5. clear：清除K-V
6. containsKey：查找键是否存在

Map接口遍历方法

1. containsKey：查找键是否存在
2. keySet：获取所有的键
3. entrySet：获取所有关系k-v
4. values：获取所有的值

HashMap小结

1. Map接口的常用实现类：HashMap、Hashtable和Properties。
2. HashMap是Map接口使用频率最高的实现类。
3. HashMap是以K-V 的方式来存储数据，存储的数据类型是（HashMap$Node）[案例Entry]。
4. Key不能重复，但是值可以重复，允许使用null键和null值。
5. 如果添加相同的key，则会覆盖原来的k-v，等同于修改。（key不会替换，val会替换）
6. 与HashSet一样，不保证映射的顺序，因为底层是以hash表的方式来存储的。（JDK8的HashMap 底层 数组+链表+红黑树）
7. HashMap没有实现同步，因此是线程不安全的。方法上没有做同步互斥的操作，即没有synchronized

HashMap底层机制和源码剖析

1. HashMap <K,V>是一个Node (HashMap$Node)，实现了Map.Entry<K,V>,看源码可以知道。
2. HashMap扩容机制和HashSet完全一样 【HashMap是HashSet的底层】
3. HashMap底层维护了Node类型的数组table,默认为null

HashTable的基本介绍

1. 存放的元素是键值对：即 K-V
2. HashTable的键和值都不能为null，不能存储空键和空值。
3. HashTable使用方法基本上和HashMap一样
4. HashTable是线程安全的，HashMap是线程不安全的
5. HashTable的底层机制
   1. 底层有数组HashTable$Entry[] 初始大小为11
   2. 临界值 threshold 8 = 11 \* 0.75
   3. 扩容：按照自己的扩容机制进行，

当if（count >= threshold）满足时，就进行扩容

按照 int newCapacity = (oldCapacity << 1)+1;的大小扩容

HashTable vs HashMap

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 版本 | 线程安全（同步） | 效率 | 允许null键null值 |
| HashTable | 1.0 | 安全 | 较低 | 不可以 |
| HashMap | 1.2 | 不安全 | 高 | 可以 |

Properties Map接口实现类

基本介绍

1. Properties类继承自Hashtable类并且实现了Map接口，也是使用一种键值对的形式来保存数据。
2. 使用特点和Hashtable类似

Properties还可以用于从xxx.properties（配置文件）中，加载数据到properties类对象，并进行读取和修改。

总结-开发中如何选择集合实现类

开发中，选择什么集合实现类，主要取决于业务操作特点，然后根据集合实现类的特性进行选择。

1. 先判断存储对象的类型（一组对象或一组键值对）
2. 一组对象[单例]：Collection接口

允许数据重复:List

增删多：LinkedList [底层维护一个双向链表]

改查多：ArrayList [底层维护Object类型的可变数组]

不允许数据重复：Set

无序：HashSet [底层是HashMap，维护了一个哈希表（数组+链表+红黑树）]

排序：TreeSet

插入和取出顺序一致：LinkedHashSet，维护数组+双向链表

1. 一组键值对[双列]：Map

键无序：HashMap

键排序：TreeMap

键插入和取出顺序一致LinkedHashMap

读取文件：Properties

TreeSet（排序）

1. 当我们使用无参构造器，创建TreeSet时，仍然是无序的
2. 如果想要按照一定顺序排序可以使用TreeSet提供的一个构造器，可以传入一个比较器（匿名内部类）指定排序规则

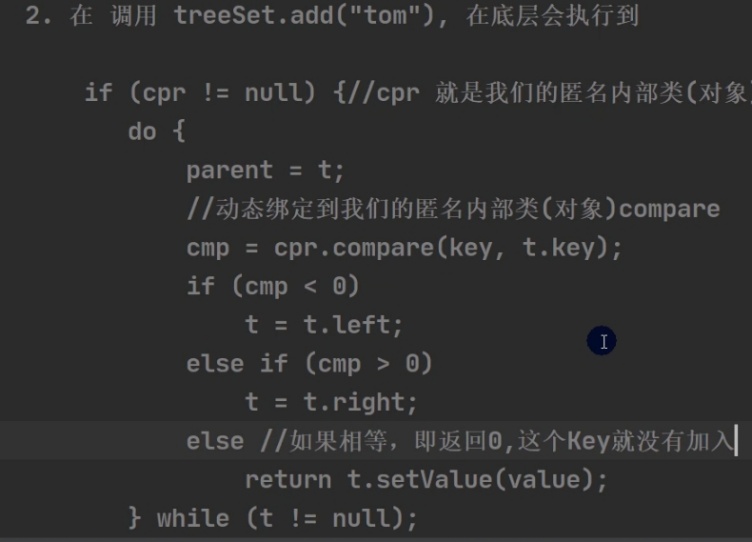
TreeSet treeSet = new TreeSet(new Comparator() {  
 @Override  
 public int compare(Object o1, Object o2) {  
 return 0;  
 }  
});

1.构造器把传入的比较器对象，赋给TreeSet的底层的TreeMap的属性this.comparator

public TreeMap(Comparator<? super K> comparator) {

this.comparator = comparator;

}

2.

Collections工具类

1. Collections是一个操作Set、List和Map等集合的工具类
2. Collections中提供了一系列静态的方法对集合元素进行排序、查询和修改等操作。

泛（广泛）型（类型）（generic） E = Integer || E = String ,是数据类型的符号表达。

泛型的好处

1. 编译时，检查添加元素的类型，提高了安全性。
2. 减少了遍历集合时的类型转换次数，提高效率
3. 不再提示编译警告。Java泛型可以保证如果程序在编译时没有发出警告，运行时就不会产生ClassCastException异常。同时，代码更加简洁、健壮。

泛型的作用是：可以在类声明时通过一个标识表示类中某个属性的类型，或者是某个方法的返回值的类型，或者是参数类型。

解释：class A<E> {  
 E s;//E表示 s的数据类型，该数据类型在定义A对象时指定，在编译期间，就确定了E是什么类型  
 public A(E s) {//E也可以是参数类型  
 this.s = s;  
 }  
 public E func(){//返回类型使用E  
 return s;  
 }  
}

泛型的语法

泛型的声明

Interface接口<T>{}和class类<K,V>{}

比如：List，ArrayList

说明：

1. 其中，T,K,V不代表值，而是表示类型的符号。
2. 任意字母都可以。常用T代表，是Type的缩写。

泛型的实例化：

要在类名后面指定类型参数的值。Eg.

1. List<String>strList = new ArrayList<String>();
2. Iterator<Customer>iterator = customers.iterator();

泛型使用的注意事项和细节

1. interface List<T>{} , public class HashSet<E>{}…

说明：T,E只能是引用类型

1. 在指定泛型具体类型后，可以传入该类型或者其子类类型
2. 泛型使用形式

List<Integer>list1 = new ArrayList<Integer>();

List<Integer>list2 = new ArrayList<>();

4.如果这样写List list3 = new ArrayList();默认给它的泛型是<E> E就是Object类型。

自定义泛型

基本语法

Class 类名<T,E,R,…>{

成员

}

注意细节

1. 普通成员可以使用泛型（属性、方法）
2. 使用泛型的数组，不能初始化//因为数组在new时不能确定T的类型，就无法在内存开空间。
3. 静态方法中不能使用泛型。//因为静态是和类相关的，在类加载时，对象还没有创建。而泛型是对象创建时才指定的。所以静态方法和静态属性使用了泛型，jvm就无法完成初始化。
4. 泛型类的类型<T,E,…>，是在创建对象时确定的（因为创建对象时，需要指定确定类型）
5. 如果在创建对象时，没有指定类型，默认为Object。

自定义泛型接口

基本语法

Interface 接口名<T,R,…>{ }

注意细节：

1. 接口中，静态成员也不能使用泛型（和泛型类规定一样）
2. 泛型接口的类型，在继承接口或者实现接口时确定类型
3. 没有指定类型，默认为Object

自定义泛型方法

基本语法

修饰符<T,R,…>返回类型 方法名（参数列表）{ }

注意细节

1. 泛型方法，可以定义在普通类中，也可以定义在泛型类中
2. 当泛型方法被调用时，类型会确定
3. Public void func(E e){}, 修饰符后没有<T,R,…> func方法不是泛型方法，而是使用了泛型。
4. 泛型方法，可以使用类声明的泛型，也可以使用自己声明的泛型。

泛型的继承和通配符

说明：

1. 泛型不具备继承性

List<Object>list = new ArrayList<String>();//wrong

通配符

2.<?>:支持任意泛型类型

3.<? extends A>:支持A类以及A类的子类，规定了泛型的上限。

4.<？super A>：支持A类以及A类的父类，不限于直接父类，规定了泛型的下限。

线程（基础）

基本概念  
程序：程序是一组指令（代码），用来让计算机执行特定的任务。

进程：是指运行中的程序。进程是程序一次执行过程，或是正在运行的一个程序。

是动态过程：有它自身产生、存在和消亡的过程。

进程（Process）定义

进程是程序在内存中运行的**动态实例**，是操作系统进行资源分配和调度的基本单位。每个进程拥有独立的地址空间、资源（如内存、文件句柄）和执行状态。

什么是线程

1. 线程是由进程创建的，是进程的一个实体。
2. 一个进程可以拥有多个线程。
3. 单线程：同一个时刻，只允许执行一个线程。
4. 多线程：同一个时刻，可以执行 多个线程，比如一个QQ进程，可以同时打开多个聊天窗口，一个迅雷进程，可以同时下载多个文件。
5. 并发和并行



3.并发：单核CPU实现多任务之间来回切换运行。

4.并行：多核CPU可以让每个核心处理单个任务，达成同一时刻处理多任务状态。

线程基本使用

创建线程的两种方法

1. 继承Thread类，重写run方法。
2. 实现Runnable接口，重写run方法。

在一个进程中可以同时存在多个线程，在main线程中可以start（启动）另一个线程，且只有所以线程都执行结束时，进程才会关闭。

为什么开线程时要用到.start()，而不是直接调用重写的run方法？

因为重写的run方法还只是一个普通方法，没有真正启动一个线程，程序还是会串行化执行。把run方法执行完毕，才向下执行。

Start方法源代码是如何启动线程呢？

源码分析:

1. public synchronized void start(){

start0();

}

(2)

// start0();是本地方法，是JVM调用，底层是c/c++实现

Private native void start0(); //真正实现多线程的效果是start0，不是run（）。

Start()方法调用start0()方法后，该线程并不一定会立马执行，只是将线程变成了可运行状态。具体什么时候执行，取决于CPU，由CPU统一调度。

实现Runnable接口

说明

1. java是单继承的，在某些情况下一个类可能已经继承了某个父类，这时在用继承Thread类方法来创建线程显然不可能。
2. Java设计者提高了另外一个方式创建线程，就是实现Runnable接口来创建线程。

继承Thread VS 实现 Runnable的区别

1. 二者从Java编程设计来看没有本质区别。
2. 实现Runnable接口方式更加适合多个线程共享一个资源的情况，并且避免了单继承的限制，建议使用Runnable。



线程终止

基本说明：

1. 当线程完成任务后，会自动退出。自然终止。
2. 还可以通过使用变量来控制run方法退出的方式来停止线程，即通知方式。

具体控制方法

package exercise\_580;public class terminate{  
 public static void main(String[] args) throws InterruptedException {  
 T t = new T();  
 t.start();  
 Thread.*sleep*(5\*1000);//让主线程休眠5秒，再通知T线程退出。  
 t.setLoop(false);//使用main线程去控制T线程的终止，必须可以修改loop  
 }  
}  
class T extends Thread {  
 boolean loop = true;//控制变量loop  
 int i=0;  
 @Override  
 public void run() {  
 super.run();  
  
 while (loop){//让T线程退出run方法，从而终止T线程 ->通知方式  
 System.*out*.println("线程T在运行..."+" "+ (++i) );  
 try {  
 *sleep*(1000);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 public void setLoop (boolean loop){  
 this.loop = loop;  
 }  
}

线程常用方法

1. setName //设置线程名称;
2. getName //返回该线程名称；
3. start //使该线程开始执行；
4. run //调用线程对象run方法；
5. setPriority //更改线程的优先级
6. getPriority //获取线程的优先级
7. sleep //在指定的毫秒数内让当前正在执行的线程休眠（暂停执行）
8. interrupt //中断线程

注意事项和细节

1. start底层会创建新的线程，调用run，run就是一个简单的方法调用，不会启动新线程。
2. 线程优先级的范围
3. Interrupt，中断线程，但并没有真正的结束线程。所以一般用于中断正在休眠线程。

（唤醒休眠中的线程）

1. Sleep：线程的静态方法，使当前线程休眠。

常用方法第二组

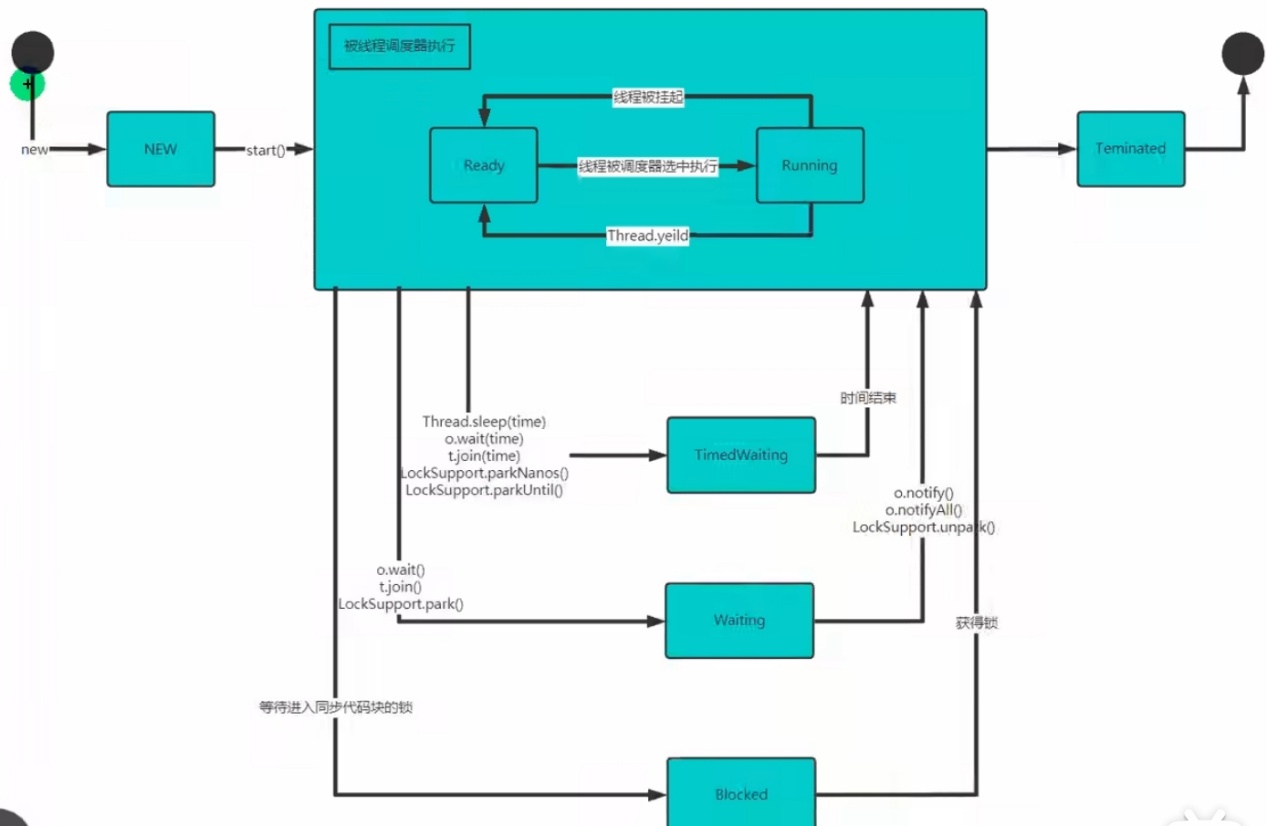
yield：线程的礼让。让出cpu，让其他线程执行，但礼让的时间不确定，所以也不一定礼让成功。

Join：线程的插队。插队的线程插队成功，则肯定先执行完插入的线程所有的任务。

用户线程和守护线程

1. 用户线程：也叫工作线程，当线程的任务执行完毕或以通知方式结束。
2. 守护线程，一般是为工作线程服务的，当所有的用户线程结束，守护线程自动结束。
3. 常见的守护线程：垃圾回收机制。
4. 如果我们希望当main线程结束后，子线程自动结束，只需将子线程设为守护线程即可。

线程的生命周期



线程同步机制（synchronized）

1. 在多线程编程，一些敏感数据不允许被多个线程同时访问，此时就使用同步访问技术，保证数据在任何时刻，最多有一个线程访问，以保证数据的完整性。
2. 也可以这样理解：线程同步，即当有一个线程在对内存进行操作时，其他线程都不可以对这个内存地址进行操作，直到该线程完成操作，其他线程才能对该内存地址进行操作。

同步具体方法-Synchronized

1. 同步代码块

Synchronized（对象）{ //得到对象的锁，才能操作同步代码

//需要被同步的代码；

}

1. synchronized还可以放在方法声明中，表示整个方法为同步方法

public synchronized void func（String name）{

//需要被同步的代码

}//不管有多少个线程来访问func，在同一时刻只能有一个线程操作func。



互斥锁

基本介绍

1. 在Java中，引入了对象互斥锁的概念，来保证共享数据操作的完整性。
2. 每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。
3. 关键字synchronized来与对象的互斥锁联系。当某个对象用synchronized修饰时，表明该对象在任一时刻只能由一个线程访问。
4. 同步的局限性：导致程序的执行效率降低。
5. 同步方法（非静态的）的锁可以是this，也可以是其他对象（要求是同一个对象）
6. 同步方法（静态的）的锁为当前类本身。

注意事项和细节

1. 同步方法如果没有使用static修饰：默认锁对象为this
2. 如果方法使用static修饰，默认锁对象为：当前类.class
3. 实现的落地步骤

需要先分析上锁的代码

选择同步代码块或同步方法

要求多个线程的锁对象必须为同一个！

线程的死锁

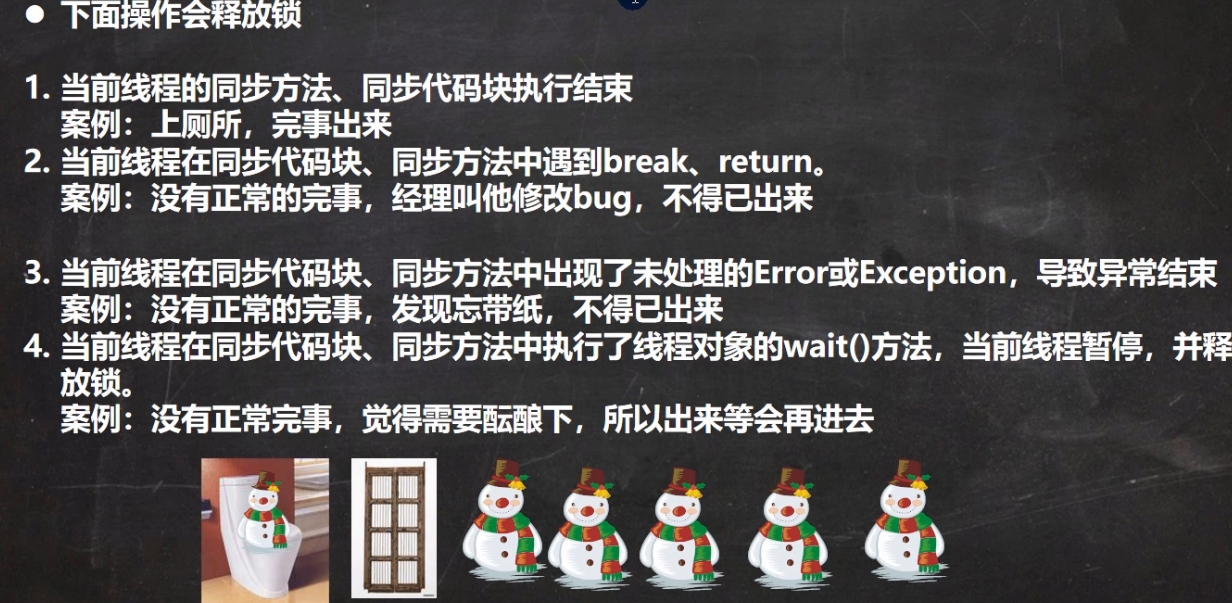
基本介绍

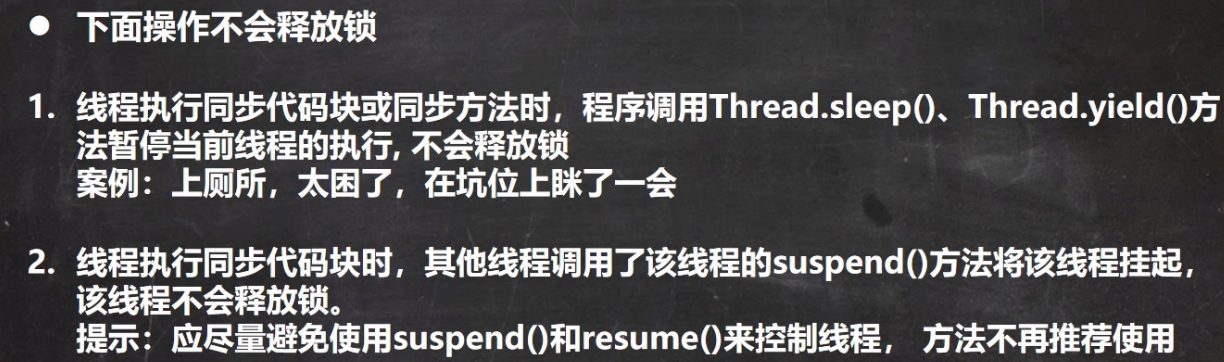
多个线程都占用了对方的锁资源，但不肯相让，导致了死锁，在编程中是一定要避免死锁的发生。

举例：  
妈妈：“你先写完作业，才让你玩手机。”

小明：“你先让我玩手机，我才写完作业。”

释放锁



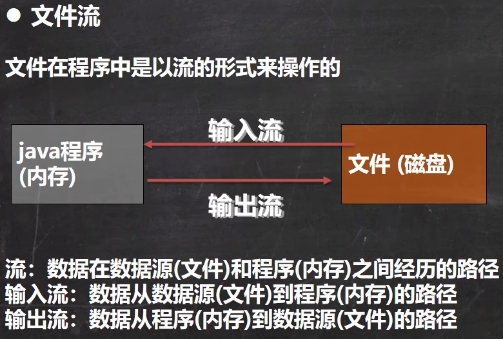


IO流

什么是文件

文件对我们来说并不陌生，文件就是保存数据的地方。

文件流概念

  
常用文件操作  
创建文件对象相关构造器和方法

相关方法

* 1. new File (String pathname)//根据路径构建一个File对象
  2. new File (File parent,String child)//根据父目录文件+子路径构建
  3. new File (String parent,String child)//根据父目录+子路径构建

createNewFile 创建新文件

常用的文件操作

获取文件的相关信息

File类的method方法中有

目录的操作和文件删除

创建一级目mkdir（make directory）、创建多级目录mkdirs、delete删除空目录或文件。

delete删除需要把目录下的文件和子目录清掉才能删除目录。

在Java编程中，目录也被当作文件。

IO流原理及流的分类

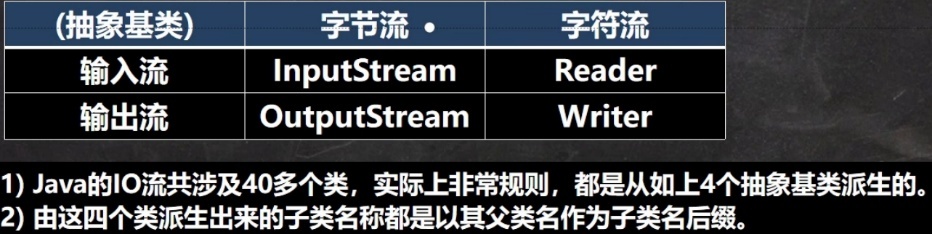
1. I/O是Input/Output的缩写，I/O技术是非常实用的技术，用于处理数据传输，如读/写文件，网络通讯等。
2. Java程序中，对于数据的输入/输出操作以“流（stream）”的方式进行。
3. Java.io包下提供各种“流”类和接口，用以获取不同种类的数据，并通过方法输入或输出数据。
4. 输入Input:读取外部数据（磁盘、光盘等存储设备的数据）到程序（内存）中。
5. 输出Output：将程序（内存）数据输出到磁盘、光盘等存储设备中。

流的分类

按操作数据单位不同分为：字节流（8bit）常用于二进制文件（输入/输出时可能会出现乱码情况），字符流（按字符）常用于文本文件。

按数据流的流向不同分为：输入流，输出流

按流的角色的不同分为：节点流，处理流/包装流

  
具体的I/O流可查看I/O流体系图-常用类。

InputStream:字节输入流

InputStream抽象类是所有字节输入流的超类

InputStream常用的子类

1. FileInputStream：文件输入流
2. BufferedInputStream：缓冲字节输入流
3. ObjectInputStream：对象字节输入流

FileOutputStream介绍 P616

字节流-文件拷贝练习 P617

FileReader和FileWriter介绍

FileReader和FileWriter是字符流，即按照字符来操作io。

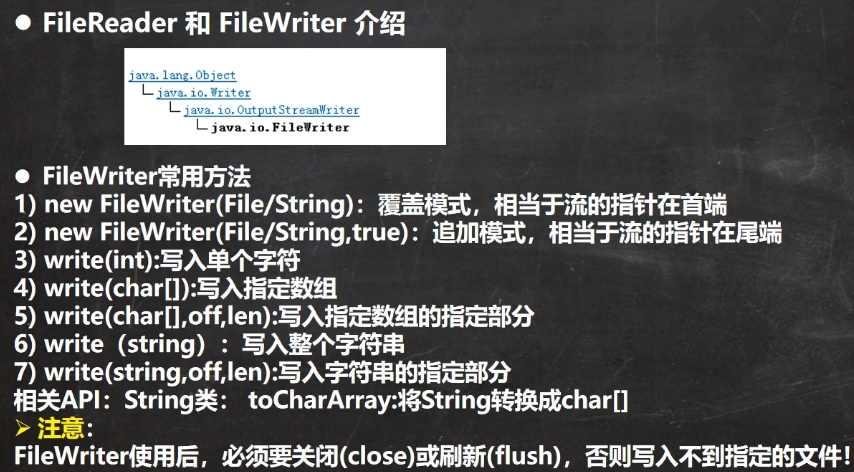
FileReader相关方法：

1. new FileReader（File/String）
2. read：每次读取单个字符，返回该字符，如果到文件末尾返回-1
3. read（char[]）:批量读取多个字符到数组，返回读取到的字符数，如果读取到文件末尾返回-1。

相关API

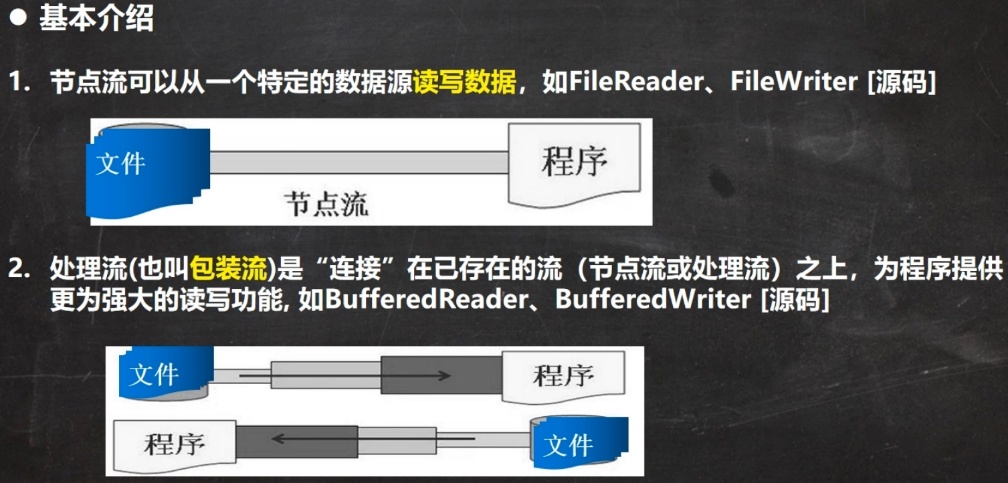
1. new String(char[]):将char[]转换成String
2. new String(char[],off,len):将char[]的指定部分转换成String

FileWriter常用方法

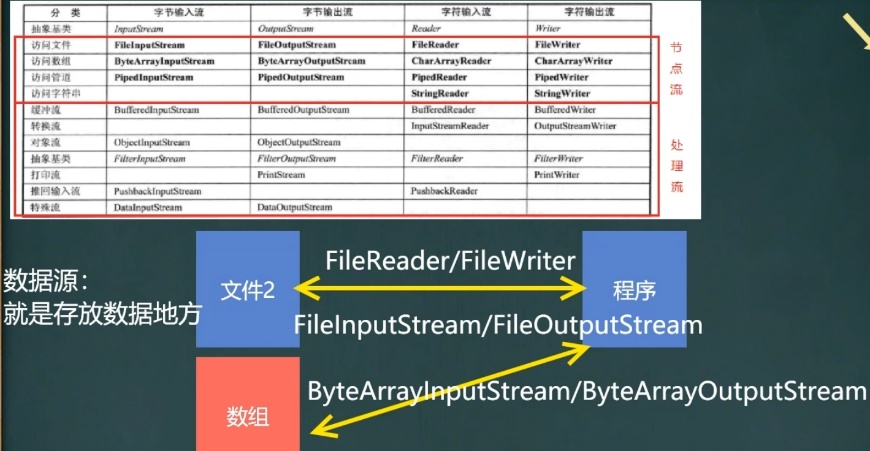
真正写入数据的是

close（）方法。 flush（刷新） + 关闭流（具体操作可看源码）

节点流和处理流（也叫包装流，对节点流进行一个包装，让这个流功能更强大。）



节点流和处理流一览图 



包装流 BufferedReader类，有属性Reader，即可以封装一个节点流，该节点是任意的，只要是Reader子类。相较于节点流处理数据源的方式更加灵活。

节点流和处理流的区别和联系

1. 节点流是底层流，直接跟数据源相接。
2. 处理流（也叫包装流）包装节点流，既可以消除不同节点流的实现差异，也可以提供更方便的方法来完成输入输出。
3. 处理流对节点流进行包装，使用了修饰器设计模式，不会直接与数据源相连。

P622模拟修饰器设计模式

处理流的功能主要体现在以下两个方面：  
1.性能的提高：主要以增加缓冲的方式来提高输入输出的效率。

2.操作的便捷：处理流可能提供一系列便捷的方法来一次输入输出大批量的数据，使用更加灵活方便。  
  
包装流 - BufferedReader和BufferedWriter  
BufferedReader和BufferedWriter属于字符流，是按照字符来读取数据的

关闭时包装流，只需要关闭外层流\_包装流（会自动关闭节点流）即可。（真正处理数据流的是节点流）  
  
（P625）综合使用BufferedReader和BufferedWriter完成文本文件拷贝，注意文件编码。  
1.BufferedReader和BufferedWriter是按照字符操作

1. 不要去操作二进制文件[声音、视频、.doc、.pdf]，可能造成文件损坏。  
     
   处理流（包装流）— BufferedInputStream和BufferedOutputStream  
   介绍BufferedInputStream  
   BufferedInputStream是字节流，在创建BufferedInputStream时，会创建一个内部缓冲区数组。  
     
   介绍BufferedOutputStream  
   BufferedOutputStream是字节流，实现缓冲的输出流，可以将多个字节写入底层输出流中，而不必对每次字节写入调用底层系统。  
     
   对象流 ObjectInputStream和ObjectOutStream

一个需求：保存数据int 100，不仅要保存数值100，还要把100的数据类型int保存下来，并且能够从文件中直接恢复int 100. （或者） Dog dog = new Dog(“小黄”，3)；这个dog对象保存到文件中，并且能够从文件里恢复。

这种要求，就是能够将基本数据类型或者对象进行 序列化 和 反序列化操作

序列化和反序列化

1. 序列化就是在保存数据时，保存数据的值和数据类型
2. 反序列化就是在恢复数据时，恢复数据的值和数据类型
3. 需要让某个对象支持序列化机制，则必须让其类是可序列化的，为了让某个类是可序列化的，该类必须实现如下两个接口之一:  
   Serializable(可序列化的) //这是一个标记接口，没有方法。

Externalizable

对象流 ObjectInputStream和ObjectOutStream提供了对基本类型或对象类型的序列化和反序列化的方法。

ObjectInputStream提供 序列化功能

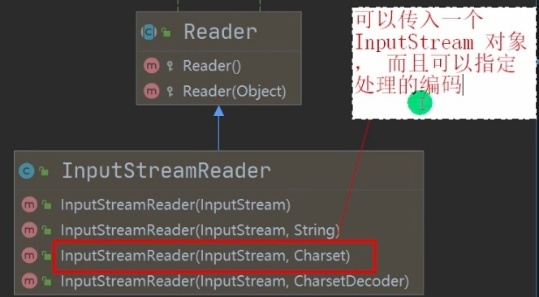
ObjectOutStream提供 反序列化功能

注意事项和细节说明

1. 读写顺序要一致
2. 要求序列化和反序列化对象，需要实现Serializable
3. 序列化的类中建议添加SerialVersionUID（序列化的版本号），为了提高版本的兼容性。
4. 序列化对象时，默认将里面所有属性都进行序列化，但除了static或transient修饰的成员。
5. 序列化对象时，要求里面属性的类型也需要实现序列化接口
6. 序列化具备可继承性，也就是如果某类已经实现了序列化，则它的所有子类也已经默认实现了序列化。

标准输入输出流  
介绍  
System.in标准输入 类型 InputStream 默认设备 键盘

System.out标准输出 类型 PrintStream 默认设备 显示器  
  
转换流- InputStreamReader和OutputStreamWriter  
顾名思义把字节流转成字符流  
1.InputStreamReader:Reader的子类，可以将InputStream（字节流）包装成（转换）Reader（字符流）

指定编码Charset

2. OutputStreamWriter：Writer的子类，实现将OutputStream（字节流）包装成Writer（字符流）  
OutputStreamWriter下有OutputStreamWriter（OutputStream，Charset）方法。

3.当处理纯文本数据时，如果使用字符流效率更高，并且可以有效解决中文问题，所以建议将字节流转换成字符流。

4.可以在使用时指定编码格式（eg.UTF-8、gbk、gb2312,…）

打印流- PrintStream（字节流） 和PrintWriter（字符流）

打印流只有输出流，没有输入流。  
  
Properties类  
java.util.Properties是java.util.Hashtable<object,object>的子类  
1.专门用于读写配置文件的集合类

配置文件的格式：  
键=值  
键=值  
2.注意：键值对不需要有空格，值不需要用引号括起来。默认类型是String

Properties.store(OutputStream os，注释)方法，可以将setProperty(key,value):设置的键值对写入到File.properties文件中。

网络相关概念  
网络通讯  
1.概念：两台设备之间通过网络实现数据传输。

2.网络通信：将数据通过网络从一台设备传输到另一台设备。

3.java.net包下提供了一系列的类或接口，供程序员使用，完成网络通信。  
4.在网络通讯时，IP负责把数据包送到正确的计算机，UDP（端口号+校验和）负责把数据包送到正确的程序。UDP协议传输数据，不提供数据修复或数据重发机制。接受方接受到损坏的数据，一般只是扔掉。而且UDP无法得知数据是否到达。不过UDP传输效率高。  
5.为了使“所有数据必须到达”就用“传输控制协议”，简称TCP和UDP一样传输数据时，在数据包头部存放IP HEADER,同时也有“端口号+校验和”。但TCP有更高级的功能，比如①TCP数据包有序号，序号使得接受方可以把数据包排成正确顺序，即使到达时间不同。  
②TCP要求接收方的电脑收到数据包并且“校验和”检查无误后（数据没有损坏）给发送发发一个确认码，代表收到了。发送方得知上一个数据包成功抵达后，会发下一个数据包，如果这次发出去之后，没收到确认码，那么肯定哪里出错了。过了一定时间还没收到确认码，发送方会再发一次。

③收件方有序列号如果收到重复的数据包就删掉。  
④TCP不是只能一个包一个包的发送，可以同时发多个数据包，收多个确认码。确认码的成功率和来回时间可以推测网络的拥挤程度。TCP利用这个信息调整发包数量，解决拥堵问题。  
  
网络  
1.概念：两台或多态设备通过一定物理设备连接起来构成了网络。

2.根据网络的覆盖范围不同分为：局域网（LAN）、城域网（MAN）、广域网（WAN）

ip地址  
1.概念：用于唯一标识网络中的每台计算机/主机。  
2.ip地址由四个字节组成。一个字节有八位。在ip地址中一个字节的范围是：0~255

3.ip地址的组成 = 网络地址（哪个停车场的）+ 主机地址（哪个停车位）。  
4.IPV6使用128位表示地址。16个字节的ip地址是IPV4 ip地址的4倍。

域名  
1.www.baidu.com  
2.好处：方便记忆，解决记ip 110.242.69.21的困难。  
3.概念：将ip地址映射成域名。

端口号  
1.概念：用于标识计算机上某个特定的申请网络服务的应用程序。

2.表示形式：以整数形式，范围0~65535  
3.0~1024已经被占用，比如ssh 22, ftp 21, smtp 25, http 80  
4.常见的网络程序端口号：

Tomcat：8080

Mysql:3306

Oracle:1521

DNS（Domain Name System）域名系统

负责把域名和IP地址一一对应，就像电话簿一样。  
运行原理：在浏览器输入域名，浏览器会去访问DNS服务器，它的IP地址是多少。一般DNS服务器是互联网供应商提供的，DNS会查表，如果域名存在，就返回对应的IP地址。  
输入正确的域名，DNS返回对应IP地址，然后浏览器会给这个IP地址发送TCP请求。