# [Java基础总结篇--JavaSE你必须要知道的基础](https://blog.csdn.net/u011200604/article/details/51893737)

|  |
| --- |
| **01.Java概述与语法** ****作者：**** 风离紫竹 |

|  |
| --- |
| java发展概述 |
| 一、  Java由SUN公司研发，SUN 被 Oracle 收购  Java 由1995年发布，正式版本由1996年1月发布（jdk1.0）  Java之父： James Gosling  二、  面向对象  分布式  多线程  简单化  安全  跨平台移植  ------    JVM   Java Virtual Machine Java虚拟机  三、  JavaSE  Java Standard Edition : Java标准版本  JavaEE Java Enterprise Edition ： Java企业版本  JavaME Java Micro Edition ： Java微型版    ----------------  Android  四、搭建开发环境  JVM   Java Virtual Machine Java虚拟机 ： 用于与操作系统进行交互  JRE   Java Runtime Enviroment Java运行环境： JVM  +    Java核心类库  JDK  Java Development Kit Java开发工具包 ： JRE  +  Java开发工具集（java.exe  javac.exe  javadoc.exe）  2.  ①下载安装JDK  ②通过命令提示符到JDK安装路径的bin路径下，执行 javac  ③配置path环境变量：JDK安装路径的bin路径下          流程：先在当前路径下找是否有 javac.exe，若没有再到 path 环境变量中从前往后依次查找          目的：在任意路径下执行 javac  ④JAVA\_HOME :  JDK安装根路径  五、开发第一个应用程序  步骤：  ①创建一个 .java 结尾的文件，被称为 java源文件。 如：  【HelloWorld.java】      public class HelloWorld{          public static void main(String[] args){              System.out.println("HelloWorld！");          }      }  ②编译： 通过 javac + 源文件名称 命令进行编译， 生成一个或多个 .class 字节码文件。 如：【javac HelloWorld.java】  ③运行： 通过 java + .class 字节码文件名 命令进行运行。（JVM会将一个或多个.class 字节码文件加载到内存中）。  如：【java HelloWorld】  2. 开发第一个应用程序的注意：  ①以  .java 结尾的文件，被称为 java源文件。  ②一个 .java 源文件中可以有多个类，但是，只能有一个 public 修饰的类  ③public 修饰类的名称必须与 .java 源文件名称一致  ④每条语句以 “;” 结束  ⑤Java 严格区分大小写  六、注释语句: 不会被JVM解释执行的语句  //单行注释  /\*      多行注释：不能嵌套  \*/  /\*\*      文档注释： 可以通过  javadoc -d mydoc -author -version HelloWorld.java 命令生成说明文档  \*/ |

|  |
| --- |
| Java基础语法 |
| 一、  标识符：凡是自己命名的地方都叫标识符。 如： 包名、类名、接口名、方法名、变量名、常量名  关键字：被Java赋予了特殊含义的单词。  1. 命名的规则 （必须遵守，若不遵守编译不能通过）  ①可使用 字母  A-Z  a-z  数字 0-9   特殊字符  下划线 "\_" 和  美元符 “$”  ②数字不能开头  ③其中不能包含空格  ④不能使用关键字和保留字，但是可以包含关键字和保留字  ⑤Java严格区分大小写，但是长度无限制。  2. 命名的规范 （可以不遵守，但是会收到鄙视）  ①包名： 所有字母都小写。 如： xxxyyyzzz  ②类名、接口名：若出现多个单词，每个单词首字母都大写。如： XxxYyyZzz  ③方法名、变量名： 若出现多个单词，第一个单词的首字母小写，其余单词首字母都大写。如： xxxYyyZzz  ④常量名： 所有字母都大写，每个单词之间以 "\_" 分隔。 如： XXX\_YYY\_ZZZ  二、  变量：用于保存数据  局部变量  &  成员变量  1. 变量的格式：   数据类型   变量名   =   值;      //声明一个变量      数据类型  变量名;    如:  int var;      //为变量赋值      变量名 =  值；    如： var = 10;      //声明一个变量并赋值      int var = 10;  2. 变量的概念：      ①在内存中开辟一块内存空间      ②该空间有名称（变量名）有类型（数据类型）      ③变量可以在指定的范围内不断的变化  3. 变量的注意：      ①作用域：变量作用在所属的那对 {}  内      ②局部变量在使用前必须赋初始值      ③先声明，后使用  三、进制之间的转换（了解）  四、变量的数据类型  基本数据类型（8种）：      整型： byte(8位)  short(16位)  int(32位)-默认类型 long(64位)      浮点型： float(32位)  double(64位)---默认类型      字符型： char(2个字节---16位)      布尔型： boolean  引用数据类型：      |---类(class)   --------------------  String 字符串      |---接口(interface)      |---数组([])  声明变量的注意：      ①声明 long 型变量，值后需要加 L 或 l .  如：  long l = 123456l;    long l1 = 123456L;      ②声明 float 型变量，值后必须加 F 或 f 。 如： float f = 15.6f;      ③声明 double 型变量，值后可以加 D 或 d 。  如： double d1 = 167.7D;      ④声明 char 型变量，值必须使用单引号。只能存储单个字符         存储 Unicode 编码（ASCII 、 中文、日文、特殊字符等等）      ⑤声明 String 变量，值必须使用双引号  五、变量的运算  数据类型的转换：  自动类型转换（自动升级）：小容量转大容量。（系统自动完成）      ①byte  short  char   --->   int  --->  long  ---> float  --->  double      ②byte short  char  三者之间不进行运算，若运算自动提升成 int 再做运算              char c1 = 'A';              char c2 = 'B';              int i = c1 + c2;      ③boolean 不参与运算      ④String与任何基本数据类型使用连接符(+) 进行运算，都将自动串接成 String 类型  强制类型转换：大容量转小容量。需要使用强转符 "(需要转换的类型)"                     但是可能损失精度。  六、运算符  算数运算符 :  +   -   +   -   \*   /  %   前++  后++  前--  后--   +(连接符)  //除法  int a = 12;  int b = 5;  int c = a / b; //2  //取模  int c = 12 % 5;  //前后++  int i = 10;  int j = i++;  System.out.println(i);//11  System.out.println(j);//10  System.out.println(++i);  int i = 10;  i = i++; //10  赋值运算符：  =   +=   -=   \*=   /=   %=  int i = 10;  i += 5; //i = i + 5;  System.out.println(i); //15  【面试题】  short s = 5;  s = s + 1;//编译？  NO  s += 1; //编译？ YES      s = (short)(s + 1);  比较运算符（关系运算符） :  ==    !=  >    <    >=   <=  （运算后结果都为 boolean 型）  int a = 10;  int b = 20;  boolean b1 =  a == b;  逻辑运算符： && -短路与   ||-短路或   !-逻辑非    &-逻辑与   |-逻辑或   ^-异或  （运算后结果都为 boolean 型）  //需求：判断一个数是否大于10 小于20  int a = 10;  //10 < a < 20; 错误的做法  boolean b2 = a > 10 && a < 20;  【面试题】 && 和  &  的区别？  && ： 称为短路与，当左边表达式结果为 false 时，右边表达式将不再运算  & ： 是位运算符，当用于逻辑运算时，无论左边表达式结果为true还是false，右边都运算  位运算符 ：  ~    |   &  ^    <<  >>   >>>   注意：没有 <<<  三元运算符（三目运算符）      条件表达式  ？  表达式1  :  表达式2;      ①当条件表达式结果为 true 时，执行表达式1 ，否则执行表达式2      ②表达式1和表达式2的结果类型需要保持一致！ |
| 一、流程控制：  顺序结构  分支结构      条件判断：              if(条件表达式){                  //当条件表达式结果为 true 时，需要执行的语句              }              if(条件表达式){                  //当条件表达式结果为 true 时，需要执行的语句              }else{                  //当条件表达式结果为 false 时，需要执行的语句              }              if(条件表达式1){                  //当条件表达式1 结果为 true 时，需要执行的语句              }else if(条件表达式2){                  //当条件表达式2 结果为 true 时，需要执行的语句              }else if(条件表达式3){                  //当条件表达式3 结果为 true 时，需要执行的语句              }              ……              else{                  //当上述条件表达式结果都为 false 时，需要执行的语句              }              注意：                  ①当某个条件表达式结果为true， 执行相应的语句，其他 else if 将不再执行                  ②if-else 语句可以嵌套的      选择结构：              switch(表达式){                  case 值1 :                      //执行的语句                      break;                  case 值2 :                      //执行的语句                      //break;                  case 值3 :                      //执行的语句                      break;                  ……                  default :                      //执行的语句                      break;              }              注意：                  ①表达式结果的数据类型，只能是 byte  short  char  int  枚举   String(jdk1.7后)                  ②表达式结果的数据类型要与case后值的类型需要保持一致！                  ③default 是可选的                  ④break 也是可选的，若与某个case后的值匹配成功，依次向下执行，直到遇到break为止。'                  ⑤case 后只能写常量值，不能写表达式                  //需求：若一个数大于2 小于5  打印 “2-5”                  int i = 2;                  switch(i){                      case 1:                          System.out.println("一");                          break;                      case 2:                      case 3:                      case 4:                      case 5:                          System.out.println("2-5");                          break;                  }  循环结构：  ①初始化值  ②循环条件  ③迭代条件  ④循环体      for(① ; ② ; ③){          ④      }      ①②④③②④③②④③②……④③②      ①      while(②){          ④          ③      }      ①      do{          ④          ③      }while(②);  while 和 do-while 的区别？  while ： 先判断循环条件，再执行循环体  do-while : 先执行循环体，再判断循环条件。（至少执行一次）  二、嵌套循环： 一个循环充当了另一个循环的循环体  打印100以内的质数：  boolean flag = true;  for(int i = 2; i <= 100; i++){      for(int j = 2; j < i; j++){          if(i % j == 0){              flag = false;              break;          }      }      if(flag){          System.out.println(i);      }      flag = true;  }  三、特殊流程控制语句  break: 结束“当前”循环 。当用于switch-case语句时，用于结束当前 switch-case 语句  continue:  结束“当次”循环  label:for(int i = 0; i < 100; i++){      for(int j = 0; j < 100; j++){          if(j == 13){              System.out.println(j);              break label;          }      }  } |

|  |
| --- |
| **02.方法的声明和使用** ****作者：**** 风离紫竹 |

|  |
| --- |
| 方法的声明和使用 |
| 一、方法：也叫函数，是一个功能的定义。是一个类中最基本的功能单元  //需求： 计算一个数 \* 2 + 1 的结果  int a = 10;  System.out.println(a \* 2 + 1);  a = 5;  System.out.println(a \* 2 + 1);  a = 8;  System.out.println(a \* 2 + 1);  1.方法的格式：  修饰符  返回值类型  方法名(参数列表){      //功能语句      return 返回值;  }  返回值类型：说明方法运行后有结果，那个结果的数据类型。  参数列表：就是局部变量，可以有0个或多个，每个参数之间以 “,” 分隔  return ： 用于结束当前方法  返回值：方法运行结束后，返回具体结果的值  public static int result(int a){ //形式参数： 用于接收调用者传递的实际参数      return a \* 2 + 1;  }  result(10);//实际参数  2. 方法的注意：  ①“返回值类型”与 “返回值”的数据类型需要保持一致  ②调用方法通过方法名和参数列表，注意方法名和参数列表（参数的个数、参数的类型）必须一致  ③若声明了“返回值类型”说明该方法运行后有结果，若调用者需要用到该结果     可以声明同类型变量接收。  ④若方法运行结束后，不需要返回任何结果给调用者时，方法“返回值类型”处声明为 void      void ： 表示没有返回值  ⑤方法中只能调用方法，不能声明其他方法  3. 声明方法的两点明确  ①方法是否需要返回结果给调用者      明确是否需要返回值类型，返回值类型是什么  ②方法是否有未知的数据参与运算      明确是否需要参数列表，需要几个  二、跨类调用方法：      通过“类名.方法名" 方式调用。 （暂时使用static修饰的方法为例） |
| 一、参数的值传递  基本数据类型：将基本数据类型作为参数，传递给方法，方法运行结束后，原值不会发生改变          //改变两个数的值          public static void add(int a, int b){              a += 1;              b += 2;          }          public static void main(String[] args){              int a = 10;              int b = 20;              add(a, b);              System.out.println("a=" + a + " b=" + b);//a=10, b=20          }  引用数据类型： |
| 方法重载 Overload ：  前提：在同一个类中  ①方法名相同  ②参数列表不同（参数的个数、参数的类型）  注意：与返回值类型无关  public void show(char c, int a){}  public void show(int a, char c){}  public void show(int a, char c, double d){}  public void show(String str, int a){}  public void shows(char c, int a){} |

|  |
| --- |
| **03. 面向对象编程(上)** ****作者：**** 风离紫竹 |

|  |
| --- |
| 一、面向对象编程（OOP）  面向对象和面向过程：  面向过程：强调的是功能行为  面向对象：将功能行为封装进对象，强调的是具备了功能行为的对象  （理解）把大象装冰箱一共分几步？  ①打开冰箱   ②把大象装进去（存储大象）  ③关闭冰箱  如何使用面向对象思想思考上述问题呢？  人｛      拉（冰箱）｛          冰箱.打开（）      ｝      指挥（动物）｛          动物.进入（）      ｝      推（冰箱）｛          冰箱.关闭（）      ｝  ｝  冰箱｛      打开（）｛｝      存储（）｛｝      关闭（）｛｝  ｝  大象｛      进入（）｛｝  ｝  猴｛      进入（）｛｝  ｝  狮子｛      进入（）｛｝  ｝  面向对象更加注重前期的设计  ①就是对类的设计  ②设计类就是设计类的成员：属性  &  方法  面向对象：将现实生活中一类事物的共性内容进行提取，抽象成相应Java类，用Java中类对其进行描述  现实生活中的事物：  小猫   小狗  大象  共性内容： 名称   性别   年龄    吃饭的功能    睡觉的功能  class Animal{      //属性      String name;      char gender;      int age;      //方法-行为      public void eat(){          System.out.println("吃饭");      }      public void sleep(){          System.out.println("睡觉");      }  }  若需要具体到某一个事物，通过  new 关键字创建对象  Animal a1 = new Animal(); //  a1.name = "大象";  a1.gender = '男';  a1.age = 2;  a1.eat();  a1.sleep();  System.out.println(a1.name + "," + a1.age);  类和对象：  类：对现实生活中一类事物的描述，抽象的  对象：是一个实实在在的个体 |
| 一、属性：也叫成员变量，也叫实例变量  成员变量  &  局部变量 的区别？  ①作用域不同  ②内存中的位置不同  ③成员变量有默认值，而局部变量没有默认值（局部变量使用前必须赋初始值）  成员变量的默认值：  基本数据类型：  byte short int ---> 0  long ---> 0L  float ---> 0.0F  double ---> 0.0D  char ---> '\u0000'  boolean ---> false  引用数据类型： ---> null      |-- 类(class)      |-- 接口(interface)      |-- 数组([])  2.为属性赋初始化值的方式  ①使用默认值  ②直接显示赋值  二、参数的值传递：  基本数据类型： 将基本数据类型作为参数，传递给方法，方法运行结束后，原值不会发生改变  引用数据类型： 将引用数据类型作为参数，传递给方法，方法运行结束后，原值会发生改变 |
| 内存管理：  分配： JVM自动为其分配内存空间  释放：JVM通过垃圾回收机制自动的释放内存空间  垃圾回收机制： 将内存中的垃圾对象释放  垃圾对象：不再被任何引用指向的对象  Person p = new Person();  p  = null;          System.gc(); //通知垃圾回收机制可以释放内存，但是并不能保证立即释放，加快释放。 |
| 一、面向对象的特性之一：封装性  封装的理解： 该隐藏的隐藏起来，该暴露的暴露出来  访问控制修饰符：  public ： 公共的，可用于修饰属性、方法、类。  在任何地方都可以访问  private ： 私有的，可用于修饰属性、方法。 只能在本类中访问  封装的步骤：  ①属性私有化（private）  ②提供公共的（public） get/set 方法  class Animal{      //1.      private String name;      private int legs; //描述腿的个数      //2.      public void setName(String n){          name = n;      }      public String getName(){          return name;      }      public ovid setLegs(int l){          if(l > 0 &&  l <= 4){              legs = l;          }      }      public int getLegs(){          return legs;      }  }  Animal ani = new Animal();  ani.name = "大象";  ani.legs = -1000; |
| 一、this 关键字：使用在本类中，代表当前对象。可以用于调用属性、方法、构造器  this.属性  this.方法  this(……)：      ①this调用本类构造器，必须写在构造器中可执行代码的首行      ②若一个类中有多个构造器，至少有一个构造器中不使用this (避免递归构造器调用)  谁让拥有this关键字的方法运行，谁就是当前对象  class Person{      private String name;      private int age;      public void setName(String name){          this.name = name;//可以区分局部变量和成员变量      }  }  Person p = new Person();  p.setName("张三"); |
| 一、构造器：也叫构造器方法，是类的成员之一。      属性      方法      构造器  1. 构造器的格式  访问控制修饰符  类名(参数列表){      //初始化语句  }  2. 构造器的作用  ①创建对象  ②为对象进行初始化  3. 构造器的注意：  ①构造器的名称必须与类名一致！  ②若一个类中没有显示提供任何构造器，系统会自动提供一个默认无参构造器          public Person(){}  ③若一个类中显示的提供了任何构造器，系统默认构造器将不再提供  ④构造器只能调用一次，并且是在创建对象时调用  4. 构造器的重载  ①构造器的名称必须相同  ②构造器的参数列表必须不同（参数的个数、参数的类型）  5. 为属性赋初始值的方式  ①使用默认赋值  ②直接显示赋值  ③构造器  顺序：①②③  class Person{      private String name;      private int age;      private char gender;      public Person(){          cry();      }      public Person(String name){          this.name = name;      }      public Person(String name, int age){          this(name);          this.age = age;      }      public Person(String name, int age, char gender){          this(name, age);          this.gender = gender;      }      public void setName(String name){          this.name = name;//可以区分局部变量和成员变量      }      public void cry(){          System.out.println("哭");      }  }  Person p = new Person("张三");  //p.cry();  Person p1 = new Person();  //p1.cry(); |

|  |
| --- |
| 一、包的作用：  ①用于区分重命名  ②用于控制访问权限  ③用于划分项目的结构层次，通常将功能相近的类划分到同一个包中。  package : 用于确定当前类的位置      ①使用在当前 .java 源文件可执行代码的首行      ②包的命名规范：所有字母都小写。 （通常将所在公司域名的倒置）          如：  com.atguigu.项目名.模块名;      ③每个“.”代表一层目录  import :　用于确定需要引入那个类的位置      ①使用在 package 和  class 之间      ②可以有多条，并排列出      ③ import com.atguigu.aaa.\* :    代表导入 aaa 包中所有的类或接口。 注意：包除外      ④ 若在一个类中使用了两个相同类名不同包名的两个类。 如：  java.util.Date;   java.sql.Date;          选择一个使用导入的方式：  import java.util.Date;          选择另外一个使用全限定类名（全类名）的方式：  java.sql.Date date = new java.sql.Date();      ⑤静态导入：          import static com.atguigu.aaa.StaticClass.\*; // 导入一个类中所有的静态内容 |

|  |
| --- |
| **04.声明和使用数组** ****作者：**** 风离紫竹 |

一、数组：用于批量保存一类数据。是引用数据类型之一。

    //变量的格式 ： 数据类型   变量名  =   值;

    int j = 0;

    int j;

    j = 0;

2. 声明数组

int[] scores;

String[] names;

Person[] persons;

3.为数组初始化并赋值

//静态初始化： 初始化操作和赋值操作同时进行

scores = new int[]{0,1,2,3,4,5};

//动态初始化：初始化操作和赋值操作分开进行

names = new String[5]; //{null, "张三", null, "李四", null}

names[1] = "张三";

names[3] = "李四";

4. 获取数组中的元素

String str = names[0];

System.out.println(str);

System.out.println(names[1]);

System.out.println(names[2]);

System.out.println(names[3]);

System.out.println(names[4]);

5. 遍历数组的两种方式

方式一：使用普通for循环遍历数组

for(int i = 0; i < names.length; i++){

    System.out.println(names[i]);

}

方式二：使用增强for循环遍历数组

for(被遍历数组中元素的数据类型  变量名 : 被遍历的数组){

}

for(String s : names){

    System.out.println(s);

}

6. 数组的注意：

    ①无论是静态初始化还是动态初始化必须指明长度

    ②数组中每个元素都有索引值（下角标、下标），索引值从0开始，到 数组的长度 - 1

    ③数组的属性： length  用于获取数组的长度

7. 数组的默认值

基本数据类型：

byte shor int ---> 0

long  ---> 0L

float ---> 0.0F

double  ---> 0.0D

char ---> '\u0000'

boolean ---> false

引用数据类型： ---> null

    |-- 类(class)

    |-- 接口（interface）

    |-- 数组（[]）

二、二维数组

//声明一个二维数组

int[][] arr;

//为二维数组初始化并赋值

//静态初始化：初始化和赋值操作同时进行

arr = new int[][]{ {1,2,3}, {4,5,6}, {7,8} }

//动态初始化

//动态初始化-1

arr = new int[5][6]; //{ {0,0,11,0,0,0}, {0,0,0,0,0,0}, {0,0,0,0,0,0}, {0,0,0,15,0,0}, {0,0,0,0,0,0} }

arr[0][2] = 11;

arr[3][3] = 15;

//动态初始化-2

arr = new int[5][]; //{ null, {0,11,0}, null, {0,0,0,22,0}, null };

arr[1] = new int[3];

arr[3] = new int[5];

arr[1][1] = 11;

arr[3][3] = 22;

//遍历二维数组

for(int i = 0; i < arr.length; i++){

    for(int j = 0; j < arr[i].length; j++){

        System.out.prin

    for(int a : as){t(arr[i][j] + "\t");

    }

    System.out.println();

}

for(int[] as : arr){

    for(int a : as){

        System.out.print(a + "\t");

    }

    System.out.println();

}

三、用于操作数组的工具类：   java.util.Arrays;

四、命令行参数

通过： java HelloWorld  abc ddd eee "Jane Smith"

public static void main(String[] args){

    for(int i = 0; i < args.length; i++){

        System.out.println(args[i]);

    }

}

五、可变参数

//需求：计算两个整数的和

/\*public int add(int a, int b){

    return a + b;

}

public int add(int a, int b, int c){

    return a + b + c;

}\*/

/\*public int add(int[] arr){//可变参数与数组参数之间不能构成重载

    int sum = 0;

    for(int i = 0; i < arr.length; i++){

        sum += arr[i];

    }

    return sum;

}\*/

public int add(int ... args){ //当调用方法时，可以传递0个或多个 int 型实际参数

    int sum = 0;

    for(int i = 0; i < args.length; i++){

        sum += args[i];

    }

    return sum;

}

public int add(String str, int ... args){//可变参数要写在参数列表的末尾

}

|  |
| --- |
| **05.面向对象编程(下)** ****作者：**** 风离紫竹 |

|  |
| --- |
| 一、面向对象的特性之二： 继承  1. 为什么使用继承  ①提高代码的复用性  ②提高维护性  ③有了继承让类与类之间产生了关系，能够创建出更加特殊的类型（多态）  2. 如何使用继承      关键字： extends  -----  "扩展"  明确子类是父类的扩展      如：          class A extends B{          }          子类： A   父类(超类、基类、SuperClass)：B  3. 通过继承，子类可以继承父类中所有的属性和方法。（包括私有的）      私有的属性也会被继承，但是因为 private 修饰符的作用，子类不能直接访问      需要通过 公共的 get / set 方法进行访问  4. 继承的注意：      ①不能为了简化代码，获取某功能而继承，若要完成继承两个类之间要有一定的所属关系：is a      ②Java只支持单继承，不支持多继承。（一个父类可以有多个子类，但是一个子类只能有一个父类）      ③Java支持多层继承。  class A{      void test1(){}      void test2(){}  }  class B extends A{      //void test1(){}      //void test2(){}  }  ---------------------------  class A{      void test1(){          //111111111111      }  }  class B{      void test1(){          //22222222222      }  }  class C extends A, B{}  C c  = new C();  c.test1();  二、方法的重写：当父类中的方法对于子类来说不适用的情况下，子类可以对父类中方法进行“重写”  前提： 要有继承关系，使用在子类中  ①方法名和参数列表必须相同  ②返回值类型可以不同，但是有规则（若重写方法返回值类型是父类被重写方法返回值类型的子类）  ③子类重写方法的访问控制修饰符不能小于父类被重写方法的访问控制修饰符  【面试题】 Override 和 Overload 的区别？  三、super 和 this 的使用方式完全一致！  this : 使用在本类中，代表当前对象的引用  super ： 使用在子类中，代表父类对象的引用  super.属性  super.方法  super(……); 调用父类构造器      ①当子类继承父类后，子类“所有”构造器中默认第一行第一句都有一句： super()         super : 当子类继承父类后，子类继承父类中所有的属性和方法，子类需要知道父类如何为对象进行初始化          ②若父类中没有提供无参构造器，子类“所有”构造器中必须显示调用父类有参构造器          （无论如何必须保证创建子类对象前，先初始化父类）      ③super() 调用父类构造器，必须写在构造器中可执行代码的首行          因此，this() 和 super() 不能同时出现  四、四种访问控制修饰符  public : 公共的，可用于修饰 属性、方法、类。  在任何地方都可以使用  protected: 受保护的，可用于修饰 属性、方法。  可以在 本类中、本包中、子类中  default : 默认的（缺省的） ，可用于修饰 属性、方法、类。 可以在 本类中、本包中      注意：default 并不是访问控制修饰符的关键字，在什么都不加的情况下就是 default  private : 私有的，可用于修饰 属性、方法。  只能在 本类中 使用 |
| 一、多态：一类事物的多种表现形态。    人 - 男人  女人  1. 多态的体现： ①方法的重载与重写     ②对象的多态性  2.对象的多态性：父类的引用指向子类的对象      Person p = new Man(); // 多态-向上转型      p.eat();      p.walk(); //虚拟方法调用      //p.smoking();      Man man = (Man)p; //向下转型      man.smoking();      Java程序的运行分为两种状态：      在多态的情况下，编译时, “看左边”，看的是父类的引用（父类中不具备子类特有的方法）                    运行时，“看右边”，看的是子类对象，实际运行的是子类重写父类的方法                  ———— 以上过程被称为“虚拟方法调用（动态绑定）”  3. 多态的前提：①要有继承关系  ②方法的重写（完成虚拟方法调用）  4. 引用数据类型之间的转换：  前提：要有继承关系  向上转型： 子类转父类。系统自动完成转换  向下转型： 父类转子类。需要使用强转符 “(需要转换的类型)”              可能引发  java.lang.ClassCastException  Person p = new Man();  Woman woman = (Woman)p; //编译？ YES       运行？ NO  5. Java 为了解决上述问题，提供了相应的解决办法  instanceof 运算符：      如：          p instanceof Man :  判断 p 引用指向的对象是不是 Man 的本类类型及 Man 的子类类型，如果是返回 true  Person p = new Man();  if(p instanceof Woman){      Woman woman = (Woman)p;  }  二、多态的应用  多态的应用之一：多态数组      Person[] persons = new Person[3];//该多态数组中可以存放Person本类类型的对象及Person子类类型的对象      persons[0] = new Person();      persons[1] = new Man();      persons[2] = new Woman();      for(int i = 0; i < persons.length; i++){          persons[i].eat();//虚拟方法调用      }  多态的应用之二： 多态参数  //需求：展示一个男人吃饭和走路的功能  /\*public void show(Man man){      man.eat();      man.walk()  }  public void show(Woman woman){      woman.eat();      woman.walk();  }\*/  public void show(Person p){ //多态参数：当调用方法时，可以传递Person本类类型的对象及Person子类类型的对象      p.eat();      p.walk();//虚拟方法调用      if(p instanceof Man){          Man man = (Man)p;          man.smoking();      }  } |
| 一、对象的关联：简单的说，是指一个对象中使用了另一个对象  class Teacher｛      String name;      int age;      Computer com;  ｝  class Computer{      String cpu;      String ram;      String hdd;  }  二、java.lang.Object 类： 是所有类的父类。若一个类没有显示的 extends 任何类时，默认的 extends java.lang.Object  ①既然 java.lang.Object 类是所有类的父类，那么Object类中的内容是最具共性的，所有类都适用  ②既然 java.lang.Object 类是所有类的父类，那么Object类中方法都会被“继承”  ③既然 java.lang.Object 类是所有类的父类，若Object类中的方法对于子类来说不适用，子类可以重写Object类中的方法  1. public boolean equals(Object obj) : 用于比较当前对象与参数对象是否相等      ①在 java.lang.Object 类中      ②只能比较引用数据类型是否相等      ③Object类中的equals方法比较两个对象的地址值。（通过查看源代码发现实际上使用 == 完成）      ④若Object类中的equals() 方法对于我们来说不适用，我们可以重写 Object类中 equals()  “==”运算符：      基本数据类型： 比较两个基本数据类型的值是否相等，若相等返回 true      引用数据类型： 比较两个引用数据类型的地址值是否相等，若相等返回 true  class Person /\* extends java.lang.Object\*/{      String name;      int age;      public Person(){}      public Person(String name, int age){          this.name = name;          this.age = age;      }      //重写      public boolean equals(Object obj){          if(this == obj){              return true;          }          if(obj instanceof Person){              Person p = (Person)obj;              if(this.name.equals(p.name) && this.age == p.age){                  return true;              }          }          return false;      }      public String toString(){          return "姓名：" + name  + “年龄：” + age;      }  }  //需求：若两个人的姓名年龄都一样，视为同一个人  Person p1 = new Person("张三", 18);  Person p2 = new Person("张三", 18);  System.out.println(p1.equals(p2)); //重写equals之前- false----重写后：true  System.out.println(p1);  System.out.println(p1.toString());  2. public String toString() : 返回当前对象的字符串表现形式      ①在 java.lang.Object 类中      ②Object类中的toString方法返回的格式为：          getClass.getName() + '@' + Integer.toHexString(hashCode());            因此，Object类中的toString() 方法对于我们来说不适用，我们可以重写toString()      ③当直接输出对象的引用时，默认调用 toString(); |
| 一、static 修饰符：代表静态的，可用于修饰 属性、方法、代码块、\*\*内部类  1. static 修饰的属性（静态变量或类变量）  ①随着类的加载而加载，随着类的消失而消失（生命周期最长）  ②静态变量被该类所有对象所共享  ③一旦某个对象修改该属性值，其他对象也会随之改变  ④静态变量的存在优先于对象  ⑤可以通过 “类名.类变量”的方式调用  2. static 修饰的方法（静态方法或类方法）  ①随着类的加载而加载  ②静态方法的存在优先于对象  ③通过“类名.类方法”的方式调用  ④静态方法中不能调用非静态成员，非静态方法中可以调用静态成员  ⑤静态方法中不能使用 this 和 super  3. 类变量和实例变量的区别？  ①生命周期不同  ②内存中的位置不同  二、类的成员之一： 代码块  非静态代码块：  ①格式： 类的一对 { }  ②每次创建对象时执行  ③非静态代码块的执行优先于构造器  ④用于为对象进行初始化。（通常为共性内容进行初始化）  ⑤代码块可以有多个，依次向下的顺序执行  静态代码块：  ①格式：static{ }  ②随着类的加载而加载，并且只加载一次  ③静态代码块的执行优先于非静态代码块  ④静态代码块中不可以调用非静态成员  ⑤静态代码块可以有多个，依次向下的顺序执行  三、final 修饰符： 代表最终的，可用于修饰 变量、方法、类。  final 修饰的类不能被继承  final 修饰的方法不能被重写  final 修饰的变量叫常量，一旦被赋值，值不能改变      ①常量的命名规范：所有字母都大写，每个单词之间以 "\_" 分隔      ②常量没有默认值，因此在使用前必须为常量赋初始值         赋值方式（直接显示赋值、构造器、代码块）         若选择使用构造器为常量赋值，必须保证“所有”构造器都为该常量赋值 |
|  |

|  |
| --- |
| **06. 高级类特性** ****作者：**** 风离紫竹 |

|  |
| --- |
| 一、抽象类      创建类用于描述现实生活中一类事物，类中有属性有方法，方法都有方法体。      某种情况下，父类只能知道子类应该具备一个怎样的方法，但是不能明确知道子类如何实现该方法。      例如：几何图形（多态练习），所有几何图形都应该具备一个计算面积的方法，但是不同几何图形计算面积的方式不同  Java为上述问题提供了相应的解决办法  Java允许父类中只是提供一个方法的声明，不提供具体的实现  具体的实现交给子类来完成，该方法被称“抽象方法”  拥有一个或多个抽象方法的类，称为“抽象类”  二、如何使用抽象  ： abstract  关键字  1. 使用 abstract 修饰的类称为“抽象类”  ①格式： 访问控制修饰符  abstract class 类名{}  ②拥有一个或多个抽象方法的类必须是抽象类  ③抽象类中可以有非抽象方法  ④抽象类中可以没有抽象方法的  ⑤\*\*\*抽象类不能创建实例  ⑥抽象类中可以声明构造器。作用：当子类继承父类后继承父类中所有的属性和方法，因此子类需要知道父类如何为对象进行初始化  2. 使用 abstract 修饰的方法称为“抽象方法”  ①格式： 访问控制修饰符  abstract 返回值类型  方法名(参数列表);      注意：抽象方法没有方法体，因此也不需要那对 {}  ②当子类继承父类后，若重写了父类中“所有”的抽象方法，该类为具体类，可以创建实例  ③当子类继承父类后，若没有重写父类中“所有”的抽象方法，该类必须是抽象类，不可以创建实例  3.  abstract 和 static 不能同时使用  abstract 和 final 不能同时使用  abstract 和 private 不能同时使用 |
| 一、接口：可以定义多个不相关事物的相同功能  二、如何使用接口  ①接口与类是平级的      关键字： interface      public interface Flyer{}  ②可以把接口理解为一个特殊的抽象类，因为接口中只能定义“全局静态常量”和“抽象方法”      //全局静态常量      int NUM = 10;//public static  final      //抽象方法      void fly();//public abstract  ③接口中不能定义一般方法、变量、构造器、代码块  ④\*\*接口不能创建实例  ⑤接口就是用来被实现的      关键字 ： implements      public class Bird implements Flyer{}  ⑥实现接口的类被称为“实现类”，实现类的功能和“继承”功能一致， 都可以获取接口中所有的成员  ⑦若实现类实现了接口中“所有”的抽象方法，该类为具体类，可以创建实例      若实现类没有实现接口中“所有”的抽象方法，该类必须是抽象类，不能创建实例  ⑧接口可以多实现 -----  解决了Java中单继承的局限性      如：          public class Bird implements Flyer, Runner{}  ⑨接口不可以继承任何类， 接口可以继承接口，接口可以多继承接口  ⑩一个类可以继承另一个类，同时实现多个接口      如：          public class Bird extends Animal implements Flyer, Runner{}          注意：先继承，后实现 |
| 一、内部类：在一个类中声明另一个类。 里面的类：内部类   外面的类：外部类      1. 成员内部类：          ①是类的成员之一。 （属性、方法、构造器、代码块）          ②内部类可以使用四种访问控制修饰符（public protected default private）          ③static  final              //创建静态内部类对象              Person.Mobile pm = new Person.Mobile();              pm.message();              //创建非静态内部类对象              //Person p = new Person();              //Person.Computer pc = p.new Computer();              Person.Computer pc = new Person().new Computer();              pc.show();          ④内部类的外部类拥有相同的特性。          class Person{              Stirng name;              int age;              //成员内部类              public class Computer{                  String name;                  public void show(){                      System.out.println(this.name);                      System.out.println(Person.this.name);//区分内部类属性和外部类属性                  }              }              static class Mobile{                  public void message(){                  }              }          }      2. 局部内部类：          //如下方式使用非常少          public void show(){              class InnerClass{}          }          //若某个类仅适用于当前方法时，可以声明为局部内部类          public Comparator getComparator(){              class MyComparator implements Comparator{                  public int compare(Object o1, Object o2){                      return 0;                  }              }              return new MyComparator();          }          public Comparator getComparotor(){              //匿名内部类              Comparator com = new Comparator(){                  public int compare(Object o1, Object o2){                      return 0;                  }              };              return com;          }          pulbic Comparator getComparator(){              return new Comparator(){                  public int compare(Object o1, Object o2){                      return 0;                  }              };          } |
| 一、枚举类：jdk1.5后出的新特性，可以定义有限数量的可穷举数据集。             简而言之，当确定一个类有几个对象时，使用枚举。  1. 自定义枚举类  ①私有化构造器  ②类内部创建对象  class Season{      //类内部创建对象      public static final Season SPRING = new Season();      public static final Season SUMMER = new Season();      public static final Season AUTUMN = new Season();      public static final Season WINTER = new Season();      //私有化构造器      private Season(){}  }  2. 使用 enum 关键字创建枚举类  enum Season{      SPRING,      SUMMER,      AUTUMN,      WINTER;  }  枚举类常用方法：  > valueOf(String name) : 根据name枚举类对象名称，获取指定的枚举类对象  > values() : 获取当前枚举类中所有枚举类对象的数组  3. 枚举类实现接口  public enum Season implements MyInterface{      SPRING{          public void show(){              System.out.println("春天");          }      },      SUMMER{          public void show(){              System.out.println("夏天");          }      },      AUTUMN{          public void show(){              System.out.println("秋天");          }      },      WINTER{          public void show(){              System.out.println("冬天");          }      };  //    @Override  //    public void show() {  //        System.out.println("季节");  //    }  } |
| 一、注解： jdk1.5后出的新特性，注解是一个元数据。是一个代码级别的说明。          在 Java 中以 “@注解名”的方式呈现  1. JDK 内置的常用注解  @Override: 限定重写父类方法, 该注释只能用于方法  @Deprecated: 用于表示某个程序元素(类, 方法等)已过时  @SuppressWarnings: 抑制编译器警告  2. 自定义注解  public @interface MyAnnotation{      String value() default "atguigu";  }  3. 元注解   \*  @Retention : 描述注解的生命周期   \*  @Target : 描述注解可以修饰哪些程序元素   \*  @Documented : 描述注解可以随之生成说明文档 (该注解生命周期必须为 RUNTIME)   \*  @Inherited: 被它修饰的 Annotation 将具有继承性. |
|  |

|  |
| --- |
| **07.基础API与异常处理** ****作者：**** 风离紫竹 |

|  |
| --- |
| 一、包装类（包裹类  Wrapper）  Java针对八种基本数据类型提供了对应的包装类  基本数据类型            包装类  byte                    Byte  short                Short  int                     Integer  long                    Long  float                    Float  double                Double  char                    Character  boolean                 Boolean  1. 基本数据类型与包装类之间的转换   \* 装箱：将基本数据类型转换成对应的包装类   \*     ①通过对应包装类的构造器   \*      ②通过对应包装类的静态方法 valueOf()   \*   \* 拆箱：将包装类转换成对应的基本数据类型   \*     ①通过对应包装类  xxxValue() 方法。   xxx:表示基本数据类型  2. 自动装箱和自动拆箱  int i = 10;  Integer num = i; //自动装箱  int i1 = num; //自动拆箱  注意： Integer提供了一个小的缓存（-128~127）之间，若需要装箱的值在该取值范围内         则从缓存中取一个实例。若超出该取值范围则重新 new 一个 Integer 实例  Integer num1 = 100;  Integer num2 = 100;  System.out.println(num1 == num2);//true  Integer num3 = 129;  Integer num4 = 129;  System.out.println(num3 == num4); //false  3. 基本数据类型、包装类  与  String之间的转换   \* 1.基本数据类型、包装类  转  String   \*         ① String str = i + "";   \*         ② 使用 String 类的静态方法  valueOf()   \*         ③ 通过对应包装类的静态方法 toString()   \*   \* 2.String 转  基本数据类型、包装类   \*         ① 通过对应包装类的构造器   \*         ② 通过对应包装类的静态方法 parseXxx() : Xxx :表示基本数据类型   \*             注意：没有parseChar()   \*         ③ 通过对应包装类的静态方法  valueOf()  二、java.lang.String  类： 不可变的字符序列  String str1 = "abc";  String str2 = new String("abc");  二者之间的区别？  str1 : 代表一个对象，至少在内存中开辟一块内存空间  str2 : 代表两个对象，至少在内存中开辟两块内存空间  2. String 类的常用方法   \* 1. 获取字符串的方法：   \*         ①String concat(String str)：串接字符串   \*         ②String substring(int beginIndex)：获取取字符串的子串   \*           String substring(int beginIndex, endIndex) : 包含头不包含尾   \*         ③String toLowerCase()和String toUpperCase()：转换为小写/大写   \*         ④String trim()：删除首尾空格或制表符   \* 2. 搜索方法：   \*         ①int indexOf(int ch) : 获取指定字符在字符串中的位置,若没有指定的字符，返回 -1   \*          int indexOf(int ch, int fromIndex) : 从指定位置开始搜索   \*          int indexOf(String str)   \*          int indexOf(String str, int fromIndex)   \*          int lastIndexOf(int ch) : 反向获取指定字符位置   \* 3. 判断方法：   \*      ① boolean equals(Object obj)：判断是否相等   \*         boolean equalsIgnoreCase(String str)：判断是否相等,不考虑大小写   \*      ② boolean contains(String str) :判断是否包含某字符串   \*           ③ boolean startsWith(String str)和 boolean endsWith(String str)：判断是否以指定字符串开始/结尾   \*           ④ boolean isEmpty():判断字符串是否为空   \* 4. 其它方法：   \*         ①length()：返回字符串长度   \*         ②char charAt(int index):返回索引处的字符   \*         ③将字符数组转换为字符串   \*             构造器：   \*                   String(char[] ch)   \*                   String(char[] ch, offset, count) : 将数组中一部分转换为字符串   \*             静态方法：   \*                   static String copyValueOf(char[] ch)   \*                   static String copyValueOf(char[] ch, offset, count)   \*                   static String valueOf(char[])   \*          将字符串转换字符数组: char[] toCharArray()   \*         ④String replace(char oldCahr, char newCahr) ： 替换字符串中字符   \*           String replace(String oldStr, String newStr)：替换字符串中字符串   \*        ⑤String[] split(String r):根据指定符号切割  三、StringBuffer 和 StringBuilder : 可变的字符序列，二者具备相同的API  StringBuffer 和 StringBuilder 的区别？  StringBuffer : 是线程安全的，因此效率低  StringBuilder ： 是线程不安全的，因此效率高   \*  StringBuffer 和 StringBuilder 的常用方法：   \* ① StringBuffer append(String str) : 添加   \*      StringBuffer insert(int offset, String str) ： 插入   \*      StringBuffer replace(int start, int end, String str)：替换   \*   \*  ② int indexOf(String str) ：返回子串的位置索引   \*      int lastIndexOf()   \*   \*  ③ String substring(int start, int end)：取子字符串序列   \*  ④ StringBuffer delete(int start, int end)：删除一段字符串   \*      StringBuffer deleteCharAt(int index):删除指定位置字符   \*  ⑤ String toString()：转换为String对象  四、其他常用类  1、java.util.Date : 表示特定的瞬间，精确到毫秒  2、java.text.DateFormat : 用于格式化时间/日期。但是是一个抽象类      |---java.text.SimpleDateFormat : 是 DateFormat 的子类  SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss E");  String strDate = sdf.format(date);  System.out.println(strDate);//2015-12-07 09:23:37 星期一  Date newDate = sdf.parse(strDate);  System.out.println(newDate);  3、java.lang.Math : 用于操作数学运算      double ceil(double d) : 返回不小于d的最小整数       double floor(double d): 返回不大于d的最大整数       int round(float f) : 返回最接近f的int型（四舍五入）       long round(double d):返回最接近d的long型       double abs(double d): 绝对值       double max(double d1, double d2) : 返回较大值       int min(int i1, int i2) :　返回较小值       double random() : 返回一个大于等于0.0并且小于1.0的随机数  五【了解】  java.lang.System : 系统类  java.util.Calendar : 日历类  java.math.BigInteger : 支持任意精度的整数  java.math.BigDecimal : 支持任意精度的小数 |
| 一、异常：不可预知的非正常情况  Java中的异常都是以对象形式存在的，一旦某句代码发生异常，会在该代码处生成一个异常对象  然后以堆栈式抛出，若不对其进行处理，程序终止运行。  二、异常的结构体系：  java.lang.Throwable ： 是所有错误和异常的父类      |--- java.lang.Error:  错误，一些严重的错误。 如： 内存溢出，系统错误等。 我们在代码中不做处理      |--- java.lang.Exception: 异常，我们需要尽可能预知并处理的异常。如： 用于输入有误、网络连接中断等等。          |--- 编译时异常（受检异常 checked）：编译时对其进行检查，若不处理编译不能通过          |--- 运行时异常（非受检异常 unchecked）：可以保证程序的正常运行。一旦发生该异常，会在该代码处生成一个异常对象                                            以堆栈式抛出，若不处理，系统终止运行。  三、异常的处理机制： Java中异常的处理采用的是抓抛模型  “抛”：一旦某句代码发生异常，会在该代码处生成一个异常对象然后以堆栈式抛出。（自动抛出/手动抛出）  “抓”：将上述抛出的异常进行捕获处理  异常的处理方式一：  try{      //可能发生异常的语句  }catch(Exception1 e1){//异常的类型  变量名      //异常的处理语句  }catch(Exception2 e2){      //异常的处理语句  }  ……  finally{      //一定被执行的语句  }  ①catch 块可以有多个，一旦与某个catch块匹配成功，执行相应的语句，其他catch块将不再执行  ②catch 块可以有多个，若catch块中的异常类型具备子父类关系，必须“子上父下”  ③try-catch可以嵌套的  ④finally 是可选的，一旦写上，一定会被执行，即便有 return  异常的处理方式二：  throws 关键字 (处理异常的方式是将异常抛出给调用者做具体的处理)      格式：使用在方法的声明处，后面跟异常的类型      如:          public static void dic(int a, int b) throws NumberFormatException, Exception{}  四、throw 关键字： 制造异常      格式： 使用在方法体内，后面跟异常的对象      如：          public static void div(int a, int b){              if(b == 0){                  throw new RuntimeException("除数不能为零");              }          }  【面试题】 throw 和 throws 的区别？  throw 不仅可以抛出Java提供的异常，还可以抛出自定义异常  五、自定义异常  ①声明一个类继承一个异常类（继承Exception 该异常为编译时异常，继承RuntimeException该异常为运行时异常）  ②编写构造器（通常利用构造器为 getMessage() 方法设置值）  六、异常处理的常用方法  printStackTrace() : 打印异常的详细信息。  String getMessage() : 返回异常的描述信息 |
|  |

|  |
| --- |
| **08.集合与泛型** ****作者：**** 风离紫竹 |

一、集合：就像是一种容器。用于存储、获取、操作对象的容器。

1. 数组的弊端

①数组的长度不可变     ②数组没有提供可以查看有效元素个数的方法

2. 集合的特点

①集合的长度是可变的

②集合可以存储任意类型的对象

③集合只能存储对象

3. 集合框架

java.util.Collection : 集合层次的根接口

    |--- java.util.List: 有序的，可以重复的。

        |--- ArrayList: 采用数组结构存储元素。 查询操作多时选择

        |--- LinkedList: 采用链表结构存储元素。 增删操作多时选择

        |--- Vector:

    |--- java.util.Set: 无序的，不允许重复。

        |--- HashSet : 是 Set 接口的典型实现类。

            判断元素是否存在的依据是：先比较 hashCode 值，若 hashCode 存在，再通过 equals() 比较内容

                                     若 hashCode 值不存在，则直接存储

            注意：重写 hashCode 和 equals 二者需要保持一致！

            |--- LinkedHashSet: 相较于 HashSet 多了链表维护元素的顺序。遍历效率高于 HashSet ， 增删效率低于 HashSet

        |--- TreeSet : 拥有自己排序方式

            |-- 自然排序（Comparable）：

                ①需要添加 TreeSet 集合中对象的类实现  Comparable 接口

                ②实现 compareTo(Object o) 方法

            |-- 定制排序（Comparator）

                ①创建一个类实现 Comparator 接口

                ②实现 compare(Object o1, Object o2) 方法

                ③将该实现类的实例作为参数传递给 TreeSet 的构造器

4. 集合的遍历

① 增强 for 循环

for(被遍历集合中元素的数据类型  变量名 :  被遍历的集合){

}

ArrayList al = new ArrayList();

al.add("AA");

al.add("BB");

for(Object obj : al){

    System.out.println(obj);

}

② 使用 Iterator 迭代器

//1）获取当前集合的迭代器

Iterator it = al.iterator();

while(it.hasNext()){

    Object obj = it.next();

    System.out.println(obj);

}

/\*错误的做法：通常一个 hasNext() 配合一个 next() 使用

Iterator it = al.iterator();

while(it.hasNext()){

    Object obj = it.next();

    System.out.println(it.next());

}\*/

③ ListIterator ： 列表迭代器，是List特有的迭代器(了解)

ListIterator li = al.listIterator();

while(li.hasNext()){

    Object obj = li.next();

    if(obj.equals("BB")){

        li.set("BBBBBBBBBBb");

    }

}

二、Map系列集合

java.util.Map : 用于存储成对对象的集合。具有 key(键)-value(值)对映射关系的集合。一个 key 对应着一个 value。 key不允许重复的。

    |--- HashMap：是 Map接口的典型实现类。

        |--- LinkedHashMap: 相较于 HashMap 多了链表维护元素的顺序

    |--- Hashtable： 是线程安全的，因此效率低

        |--- Properties ： 用于操作属性文件

    |--- TreeMap ： 根据 key 拥有自己的排序方式

        |-- 自然排序（Comparable）：

        |-- 定制排序（Comparator）：

    //使用 Properties 操作属性文件

    @Test

    public void test1() throws FileNotFoundException, IOException{

        //1. 创建 Properties 对象

        Properties props = new Properties();

        //2. 通过 load() 方法加载属性文件

        props.load(new FileInputStream("hello.properties"));

        //3. 通过  getProperty() 方法根据key获取对应的value

        String userName = props.getProperty("username");

        String password = props.getProperty("password");

        System.out.println(userName);

        System.out.println(password);

    }

1. Map的常用方法：

添加、删除操作：

Object put(Object key,Object value)

Object remove(Object key)

void putAll(Map t)

void clear()

元素查询的操作：

Object get(Object key)

boolean containsKey(Object key)

boolean containsValue(Object value)

int size()

boolean isEmpty()

boolean equals(Object obj)

2. Map 的遍历：

Map map = new HashMap();

map.put("AA", 123);

map.put("BB", 456);

keySet();

//遍历Map的方式一: 获取 Map 中所有的 key

Set set = map.keySet();

values();

//遍历Map的方式二：获取 Map中所有的 value

Collection coll = map.values();

//遍历Map的方式三： 获取Map中所有的 Entry (是Map 的一个内部类，一个Entry对应着Map中的一个key和一个value)

Set entrySet = map.entrySet();

for(Object obj : entrySet){

    Entry entry = (Entry)obj;

    Object key = entry.getKey();

    Object value = entry.getValue();

}

Iterator it = entrySet.iterator();

while(it.hasNext()){

    Entry entry = (Entry)it.next();

    Object key = entry.getKey();

    Object value = entry.getValue();

}

三、

为什么使用泛型：若集合中不使用泛型，意味着集合中可以添加任意类型的对象。若需要具体到某一个类型时，需要强制类型转换

              可能引发 ClassCastException

泛型： 在 Java 中以 "<>" 的形式呈现，<> 中写引用数据类型

       用于限制集合中存放元素的类型

1. 在集合中应用泛型

2. 自定义泛型类、接口、方法

class DAO<T>{ // T : Type   E:Element   K:Key   V:Value

    private List<T> list = new ArrayList<T>();

    public void add(T t){

        list.add(t);

    }

    public T get(int id){

        return list.get(id);

    }

    //自定义泛型方法

    public <E> E[] srot(E[] e){

    }

}

3. 通配符    ?

    虽然 Person 是  Student 的父类，但是 List<Person> 就不是  List<Student> 的父类

    //需求：

    //public void show(List<Student> list){}

    //public void show1(List<Man> list){}

    public void show(List<? extends Person> list){}

    List<?> : 可以接收任意带泛型类型的集合

    List<? extends Person>  : 可以接收 Person 本类类型及 Person子类类型带泛型类型的集合

    List<? super Person> :  可以接收 Person 本类类型及 Person父类类型带泛型类型的集合

|  |
| --- |
| **09.IO流与多线程** ****作者：**** 风离紫竹 |

|  |  |  |
| --- | --- | --- |
| 一、IO流的分类  1.按流向不同：输入流、输出流（以程序为主体）  2.按数据不同：字节流、字符流（字节流操作非文本文件  .jpg  .avi  .rmvb  .mp3     字符流操作文本本件  .txt   .java）  3.按角色不同：节点流、处理流  二、IO流的结构体系  抽象基类                节点流                    缓冲流（处理流的一种）  InputStream            FileInputStream            BufferedInputStream  OutputStream            FileOutputStream            BufferedOutputStream(flush() - 用于清空缓冲区)  Reader                FileReader                BufferedReader  （readLine()）  Writer                FileWriter                    BufferedWriter  (newLine() )  //使用缓冲流完成文件的复制  public static void copyFile(String src, String dest){      //2. 创建缓冲流对象（BufferedInputStream），包装现有节点流      BufferedInputStream bis = null;      //4. 创建缓冲流对象（BufferedOutputStream）, 包装现有节点流      BufferedOutputStream bos = null;      try {          //1. 创建 FileInputStream 对象，同时打开指定文件          FileInputStream fis = new FileInputStream(src);          bis = new BufferedInputStream(fis);          //3. 创建  FileOutputStream 对象，同时打开指定文件          FileOutputStream fos = new FileOutputStream(dest);          bos = new BufferedOutputStream(fos);          //5. 使用缓冲流读取指定文件的内容          byte[] b = new byte[1024];          int len = 0;          while((len = bis.read(b)) != -1){              //6. 使用缓冲流将读取的内容写到目标文件              bos.write(b, 0, len);          }  //            bos.flush(); //用于强制清空缓冲区      } catch (IOException e) {          e.printStackTrace();      } finally {          //7. 关闭流          if(bos != null){              try {                  bos.close();              } catch (IOException e) {                  e.printStackTrace();              }          }          if(bis != null){              try {                  bis.close();              } catch (IOException e) {                  e.printStackTrace();              }          }      }  }  三、对象流：   ObjectInputStream  &  ObjectOutputStream  序列化：将内存中的对象永久的以二进制形式保存到磁盘中  ①创建节点流对象  ②使用缓冲流包装节点流（可选的）  ③使用对象流包装缓冲流对象  ④进行序列化操作  ⑤关闭流  ⑥需要序列化对象所属的类需要实现 java.io.Serializable 接口  ⑦提供一个序列号  private static final long serialVersionUID = 1234256L;  注意：static 和 transient 修饰的属性不能被序列化  // 对象的序列化      @Test      public void test3() {          Person p1 = new Person("张三", 18, new Computer(), "唐朝");          Person p2 = new Person("李四", 28, new Computer(), "唐朝");          Person p3 = new Person("王五", 25, new Computer(), "唐朝");          ObjectOutputStream oos = null;          try {              FileOutputStream fos = new FileOutputStream("person.dat");              BufferedOutputStream bos = new BufferedOutputStream(fos);              oos = new ObjectOutputStream(bos);              oos.writeObject(p1);              oos.writeObject(p2);              oos.writeObject(p3);          } catch (IOException e) {              e.printStackTrace();          } finally {              if (oos != null) {                  try {                      oos.close();                  } catch (IOException e) {                      e.printStackTrace();                  }              }          }      }  反序列化：将磁盘中的对象读取  //对象的反序列化      @Test      public void test4() {          ObjectInputStream ois = null;          try {              FileInputStream fis = new FileInputStream("person.dat");              BufferedInputStream bis = new BufferedInputStream(fis);              ois = new ObjectInputStream(bis);              Person p1 = (Person) ois.readObject();              Person p2 = (Person) ois.readObject();              Person p3 = (Person) ois.readObject();              System.out.println(p1);              System.out.println(p2);              System.out.println(p3);          } catch (ClassNotFoundException e) {              e.printStackTrace();          } catch (IOException e) {              e.printStackTrace();          } finally {              if (ois != null) {                  try {                      ois.close();                  } catch (IOException e) {                      e.printStackTrace();                  }              }          }      }  四、控制台IO ：  System.in : “标准”的输入流  System.out : “标准”的输出流 ---- 通过 System 类中 setOut() 方法可以改变 println() 的默认输出位置  System.err : “标准”的错误输出流  打印流 ： PrintStream &  PrintWriter  五、转换流： InputStreamReader  &  OutputStreamWriter      @Test      public void test1(){          BufferedReader br = null;          try {              InputStream in = System.in;              InputStreamReader isr = new InputStreamReader(in);              br = new BufferedReader(isr);              String str = null;              while((str = br.readLine()) != null){                  System.out.println("--" + str);              }          } catch (IOException e) {              e.printStackTrace();          } finally {              if(br != null){                  try {                      br.close();                  } catch (IOException e) {                      e.printStackTrace();                  }              }          }      }  七、其他IO流  数据流： DataInputStream & DataOutputStream  随即存取文件流： RandomAccessFile      >seek(long l)      >getFilePointer();  八、java.io.File 类 ： 用于表示文件/目录。可用于新建、删除、重命名等基本功能的操作                  但是若需要操作文件的内容，File 就无能为力，需要使用IO流                  通常File对象与IO流配合使用  访问文件名：  getName()  getPath()  getAbsoluteFile()  getAbsolutePath()  getParent()  renameTo(File newName)  文件检测  exists()  canWrite()  canRead()  isFile()  isDirectory()  获取常规文件信息  lastModified()  length()  文件操作相关  createNewFile()  delete()  目录操作相关  mkDir()  mkDirs()  list()  listFiles() |  |  |
| 一、多线程  什么是程序？   为了完成某项特定的任务，使用某种语言，编写一组指令的集合。  什么是进程？   一个正在进行中的程序。  什么是线程？   在一个进程中执行的一套功能流程，称为线程                 在一个进程中执行的多套功能流程，称为多线程  二、为什么使用多线程？  抢占式策略系统： 系统会分配给每个执行任务的线程一个很小的时间段，当该时间段用完后                 系统会剥夺其使用权，交给其他线程去执行  1. 多线程可以提高程序的效率，可以尽可能的利用cpu的资源  2. 增强用户体验  三、如何使用多线程？  创建执行线程的方式一：  ①创建一个类继承 Thread 类  ②重写 run() 方法，同时编写线程执行体  ③创建该子类的实例  ④调用 start() 方法，启动线程。默认执行 run() 方法  创建执行线程的方式二：  ①创建一个类实现 Runnable 接口  ②实现接口中的 run() 方法，同时编写线程执行体  ③创建该实现类的实例  ④创建 Thread 实例，将实现类的实例作为参数，传递给Thread的构造器  ⑤调用 Thread 类的 start() 方法，启动线程。默认执行 run() 方法  继承方式与实现方式的区别？  ①当需要多个线程访问共享数据时，首选使用实现 Runnable 接口的方式  ②实现 Runnable 接口，解决了Java中单继承的局限性  四、线程的常用方法  currentThread() : 获取当前线程  getName() : 获取线程名称  setName() : 设置线程名称  start() : 启动线程  sleep(long millis) : 是一个静态方法，使当前线程进入睡眠状态  join() / join(long millis) : 是一个实例方法， 使当前线程进入阻塞状态  interrupt() : 用于唤醒阻塞状态的线程  yield() : 线程让步  isAlive() : 判断线程是否处于存活状态  五、线程的优先级（1-10）：默认的优先级为 5。 优先级高并不意味着线程一定先执行， 只不过更多的获取cpu的资源  MAX\_PRIORITY : 10  NORM\_PRIORITY : 5  MIN\_PRIORITY : 1  getPriority() : 获取线程的优先级  setPriority() : 设置线程的优先级  六、线程的生命周期  七、线程同步：  模拟售票程序：实现三个窗口同时售票100张  问题： 当三个线程同时访问共享数据时，出现了 无序、重复、超额售票等多线程安全问题  解决办法：将多个线程需要访问的共享数据包装起来，确保一次只能有一个线程执行流访问该共享数据  Java为上述问题也提供了相应的解决办法  1. 同步代码块：  synchronized(同步监视器){      //需要访问的共享数据  }  同步监视器：俗称“锁”，可以使用任意对象充当。但是确保多个线程持有同一把锁（同一个对象）  2. 同步方法：使用在方法的声明处，加 synchronized 关键字      如：          public synchronized void show(){ }  3. 同步锁： Lock 接口      注意：必须保证手动的释放锁   (unlock()) |  |  |