第 7 章 面向对象编程(基础部分)

7.1 类与对象

7.1. 1看一个养猫猫问题

张老太养了两只猫猫:一只名字叫小白,今年 3 岁, 白色。还有一只叫小花,今年 100 岁,花色。请编写一个程序，当用户

输入小猫的名字时，就显示该猫的名字，年龄，颜色。如果用户输入的小猫名错误，则显示 张老太没有这只猫猫。

7.1.2使用现有技术解决 Object01.java

1) 单独的定义变量解决

2) 使用数组解决

7.1.3现有技术解决的缺点分析

不利于数据的管理

效率低===》 引出我们的新知识点 类与对象 哲学, 道家思想

java 设计者 引入 类与对象(OOP) ，根本原因就是现有的技术，不能完美的解决新的新的需求.

 代码

|  |
| --- |
| public class Object01 {  //编写一个 main 方法  public static void main(String[] args) {  /\*  张老太养了两只猫猫:一只名字叫小白,今年 3 岁, 白色。  还有一只叫小花,今年 100 岁,花色。请编写一个程序，当用户输入小猫的名字时，  就显示该猫的名字，年龄，颜色。如果用户输入的小猫名错误，  则显示 张老太没有这只猫猫。 |

|  |
| --- |
| \*/  //单独变量来解决 => 不利于数据的管理(你把一只猫的信息拆解)  //第 1 只猫信息  // String cat1Name = "小白";  // int cat1Age = 3;  // String cat1Color = " 白色";  // //第 2 只猫信息  // String cat2Name = "小花";  // int cat2Age = 100;  // String cat2Color = "花色";  //数组 ===>(1)数据类型体现不出来(2) 只能通过[下标]获取信息，造成变量名字和内容  // 的对应关系不明确(3) 不能体现猫的行为  //第 1 只猫信息  // String[] cat1 = {"小白", "3", " 白色"};  // String[] cat2 = {"小花", " 100", "花色"};  //使用 OOP 面向对象解决  //实例化一只猫[创建一只猫对象]  //老韩解读  //1. new Cat() 创建一只猫(猫对象)  //2. Cat cat1 = new Cat(); 把创建的猫赋给 cat1 |

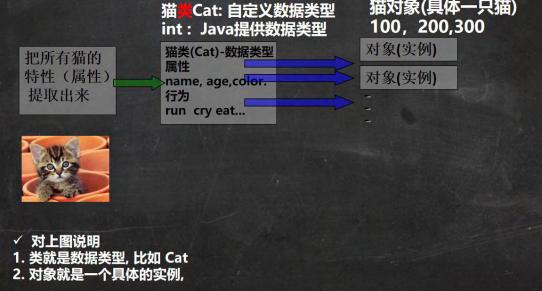
|  |
| --- |
| //3. cat1 就是一个对象  Cat cat1 = new Cat();  cat1.name = "小白";  cat1.age = 3;  cat1.color = " 白色";  cat1.weight = 10;  //创建了第二只猫，并赋给 cat2  //cat2 也是一个对象(猫对象)  Cat cat2 = new Cat();  cat2.name = "小花";  cat2.age = 100;  cat2.color = "花色";  cat2.weight = 20;  //怎么访问对象的属性呢  System.out.println("第 1 只猫信息" + cat1.name  + " " + cat1.age + " " + cat1.color + " " + cat1.weight);  System.out.println("第 2 只猫信息" + cat2.name  + " " + cat2.age + " " + cat2.color + " " + cat2.weight);  }  }  //使用面向对象的方式来解决养猫问题  // |

|  |
| --- |
| //定义一个猫类 Cat -> 自定义的数据类型  class Cat {  //属性/成员变量  String name; //名字  int age; //年龄  String color; //颜色  //double weight; //体重  //行为  } |

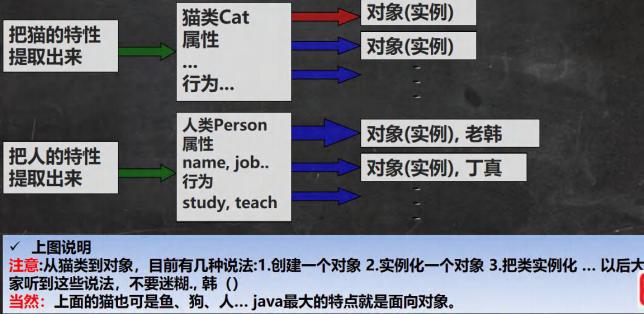
7.1.4 一个程序就是一个世界，有很多事物(对象[属性, 行为])



7.1.5类与对象的关系示意图



7.1.6类与对象的关系示意图



7.1.7快速入门-面向对象的方式解决养猫问题

Object01.java -> 看前面的代码

7.1.8类和对象的区别和联系

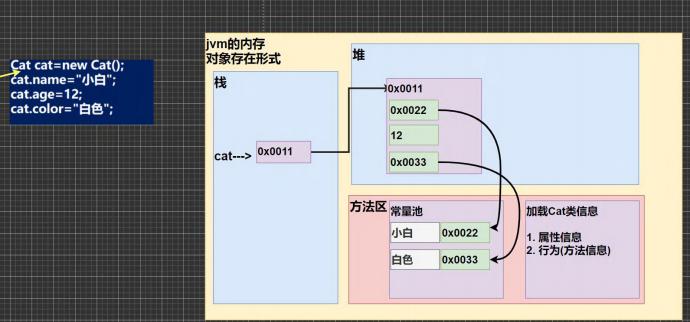
通过上面的案例和讲解我们可以看出:

1) 类是抽象的，概念的，代表一类事物, 比如人类,猫类.., 即它是数据类型.

2) 对象是具体的，实际的，代表一个具体事物, 即 是实例.

3) 类是对象的模板，对象是类的一个个体，对应一个实例

7.1.9对象在内存中存在形式(重要的)必须搞清楚。



7.1.10 属性/成员变量/字段

 基本介绍

1) 从概念或叫法上看： 成员变量 = 属性 =field(字段) （即 成员变量是用来表示属性的，授课中，统一叫 属性)

案例演示：Car(name,price,color) Object02.java

|  |
| --- |
| public class Object02 { |

|  |
| --- |
| //编写一个 main 方法  public static void main(String[] args) {  }  }  class Car {  String name;//属性, 成员变量, 字段 field  double price;  String color;  String[]master;//属性可以是基本数据类型，也可以是引用类型(对象，数组)  } |

2) 属性是类的一个组成部分，一般是基本数据类型,也可是引用类型(对象，数组)。比如我们前面定义猫类 的 int age 就

是属性

 注意事项和细节说明

PropertiesDetail.java

1) 属性的定义语法同变量，示例：访问修饰符 属性类型 属性名;

这里老师简单的介绍访问修饰符： 控制属性的访问范围

有四种访问修饰符 public, proctected, 默认, private ,后面我会详细介绍

2) 属性的定义类型可以为任意类型，包含基本类型或引用类型

3) 属性如果不赋值，有默认值，规则和数组一致。具体说: int 0，short 0, byte 0, long 0, float 0.0,double 0.0，char \u0000，

boolean false ，String null

案例演示：[Person 类]

代码

|  |
| --- |
| public class PropertiesDetail { |

|  |
| --- |
| //编写一个 main 方法  public static void main(String[] args) {  //创建 Person 对象  //p1 是对象名(对象引用)  //new Person() 创建的对象空间(数据) 才是真正的对象  Person p1 = new Person();  //对象的属性默认值，遵守数组规则:  //int 0 ，short 0, byte 0, long 0, float 0.0,double 0.0 ，char \u0000 ，boolean false ，String null  System.out.println("\n 当前这个人的信息");  System.out.println("age=" + p1.age + " name="  + p1.name + " sal=" + p1.sal + " isPass=" + p1.isPass) ;  }  }  class Person {  //四个属性  int age;  String name;  double sal;  boolean isPass;  } |

7.1.11 如何创建对象

1) 先声明再创建

Cat cat ; //声明对象 cat

cat = new Cat(); //创建

2) 直接创建

Cat cat = new Cat();

7.1.12 如何访问属性

 基本语法

对象名.属性名;

案例演示赋值和输出

cat.name ;

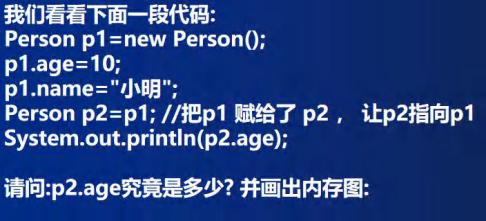
cat.age;

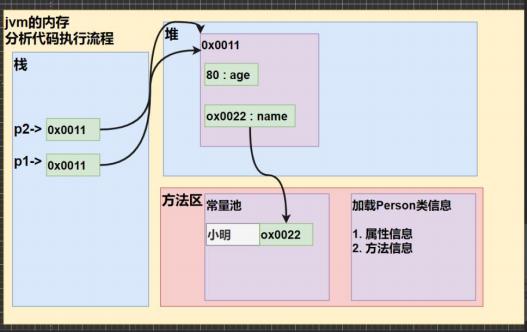
cat.color;

 类和对象的内存分配机制(重要)

看一个思考题

我们定义一个人类(Person)(包括 名字,年龄) 。 (Object03.java)





7.1.13 类和对象的内存分配机制

 Java 内存的结构分析

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：常量池(常量，比如字符串) ， 类加载信息

4) 示意图 [Cat (name, age, price)]

 Java 创建对象的流程简单分析

|  |
| --- |
| Person p = new Person();  p.name = “jack” ;  p.age = 10 |

1) 先加载 Person类信息(属性和方法信息, 只会加载一次)

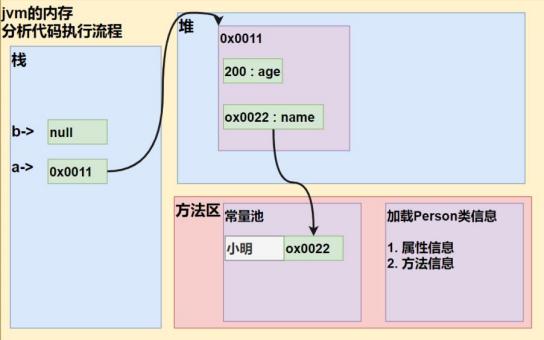
2) 在堆中分配空间, 进行默认初始化(看规则)

3) 把地址赋给 p , p 就指向对象

4) 进行指定初始化， 比如 p.name =”jack” p.age = 10

 看一个练习题，并分析画出内存布局图，进行分析





7.2 成员方法

7.2. 1基本介绍

在某些情况下，我们要需要定义成员方法(简称方法) 。比如人类: 除了有一些属性外( 年龄，姓名..),我们人类还有一

些行为比如:可以说话、跑步..,通过学习，还可以做算术题。这时就要用**成员方法**才能完成。现在要求对 Person类完善。

7.2.2成员方法快速入门

Method01.java

1) 添加 speak 成员方法,输出 “我是一个好人”

2) 添加 cal01 成员方法,可以计算从 1+..+1000 的结果

3) 添加 cal02 成员方法,该方法可以接收一个数 n ，计算从 1+..+n 的结果

4) 添加 getSum成员方法,可以计算两个数的和

代码

|  |
| --- |
| public class Method01 {  //编写一个 main 方法  public static void main(String[] args) {  //方法使用  //1. 方法写好后，如果不去调用(使用) ，不会输出  //2. 先创建对象 ,然后调用方法即可  Person p1 = new Person();  p1.speak(); //调用方法  p1.cal01(); //调用 cal01 方法  p1.cal02(5); //调用 cal02 方法，同时给 n = 5  p1.cal02(10); //调用 cal02 方法，同时给 n = 10  //调用 getSum 方法，同时 num1= 10, num2=20 |

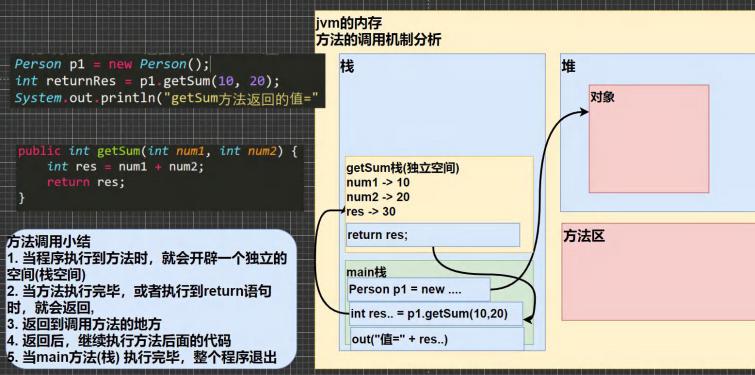
|  |
| --- |
| //把 方法 getSum 返回的值，赋给 变量 returnRes  int returnRes = p1.getSum(10, 20);  System.out.println("getSum 方法返回的值=" + returnRes);  }  }  class Person {  String name;  int age;  //方法(成员方法)  //添加 speak 成员方法,输出 “我是一个好人”  //老韩解读  //1. public 表示方法是公开  //2. void ： 表示方法没有返回值  //3. speak() : speak 是方法名， () 形参列表  //4. {} 方法体，可以写我们要执行的代码  //5. System.out.println("我是一个好人"); 表示我们的方法就是输出一句话  public void speak() {  System.out.println("我是一个好人");  }  //添加 cal01 成员方法,可以计算从 1+..+1000 的结果  public void cal01() {  //循环完成 |

|  |
| --- |
| int res = 0;  for(int i = 1; i <= 1000; i++) {  res += i;  }  System.out.println("cal01 方法 计算结果=" + res);  }  //添加 cal02 成员方法,该方法可以接收一个数 n ，计算从 1+..+n 的结果  //老韩解读  //1. (int n) 形参列表， 表示当前有一个形参 n, 可以接收用户输入  public void cal02(int n) {  //循环完成  int res = 0;  for(int i = 1; i <= n; i++) {  res += i;  }  System.out.println("cal02 方法 计算结果=" + res);  }  //添加 getSum 成员方法,可以计算两个数的和  //老韩解读  //1. public 表示方法是公开的  //2. int :表示方法执行后，返回一个 int 值  //3. getSum 方法名  //4. (int num1, int num2) 形参列表，2 个形参，可以接收用户传入的两个数  //5. return res; 表示把 res 的值， 返回  public int getSum(int num1, int num2) { |

|  |
| --- |
| int res = num1 + num2;  return res;  }  } |

7.2.3方法的调用机制原理：(重要!-示意图!!!)

提示：画出程序执行过程[getSum]+说明



7.2.4为什么需要成员方法

Method02.java

 看一个需求：

请遍历一个数组 , 输出数组的各个元素值。

 解决思路 1 ，传统的方法，就是使用单个 for循环，将数组输出，大家看看问题是什么？

 解决思路 2: 定义一个类 MyTools ,然后写一个成员方法，调用方法实现,看看效果又如何。

代码

|  |
| --- |
| public class Method02 {  //编写一个 main 方法  public static void main(String[] args) {  //请遍历一个数组 , 输出数组的各个元素值  int [][] map = {{0,0, 1}, {1, 1, 1}, {1, 1,3}};  //使用方法完成输出, 创建 MyTools 对象  MyTools tool = new MyTools();  //遍历map 数组  //传统的解决方式就是直接遍历  // for(int i = 0; i < map.length; i++) {  // for(intj = 0; j < map[i].length; j++) {  // System.out.print(map[i][j] + "\t");  // }  // System.out.println();  // }  //使用方法  tool.printArr(map);  //.... |

|  |
| --- |
| //  //要求再次遍历map 数组  // for(int i = 0; i < map.length; i++) {  // for(intj = 0; j < map[i].length; j++) {  // System.out.print(map[i][j] + "\t");  // }  // System.out.println();  // }  tool.printArr(map);  //...再次遍历  //  // for(int i = 0; i < map.length; i++) {  // for(intj = 0; j < map[i].length; j++) {  // System.out.print(map[i][j] + "\t");  // }  // System.out.println();  // }  tool.printArr(map);  }  }  //把输出的功能，写到一个类的方法中,然后调用该方法即可  class MyTools { |

|  |
| --- |
| //方法，接收一个二维数组  public void printArr(int[][] map) {  System.out.println("=======");  //对传入的map 数组进行遍历输出  for(int i = 0; i < map.length; i++) {  for(intj = 0; j < map[i].length; j++) {  System.out.print(map[i][j] + "\_");  }  System.out.println();  }  }  } |

7.2.5成员方法的好处

1) 提高代码的复用性

2) 可以将实现的细节封装起来，然后供其他用户来调用即可

7.2.6成员方法的定义

访问修饰符 返回数据类型 方法名（形参列表.. ） {//方法体

语句；

return 返回值;

}

1) 形参列表：表示成员方法输入 cal(int n) ， getSum(int num1, int num2)

2) 返回数据类型：表示成员方法输出, void 表示没有返回值

3) 方法主体：表示为了实现某一功能代码块

4) return 语句不是必须的。

5) 老韩提示: 结合前面的题示意图, 来理解

7.2.7注意事项和使用细节

MethodDetail.java

|  |
| --- |
| public class MethodDetail {  public static void main(String[] args) {  AA a = new AA();  int[] res = a.getSumAndSub(1, 4);  System.out.println("和=" + res[0]);  System.out.println("差=" + res[1]);  //细节: 调用带参数的方法时，一定对应着参数列表传入相同类型或兼容类型 的参数  byte b1 = 1;  byte b2 = 2;  a.getSumAndSub(b1, b2);//byte -> int  //a.getSumAndSub(1.1, 1.8);//double ->int(×)  //细节: 实参和形参的类型要一致或兼容、个数、顺序必须一致 |

|  |
| --- |
| //a.getSumAndSub(100);//× 个数不一致  a.f3("tom", 10); //ok  //a.f3(100, "jack"); // 实际参数和形式参数顺序不对  }  }  class AA {  //细节: 方法不能嵌套定义  public void f4() {  //错误  // public void f5() {  // }  }  public void f3(String str, int n) {  }  //1. 一个方法最多有一个返回值 [思考，如何返回多个结果 返回数组 ]  public int[] getSumAndSub(int n1, int n2) {  int[] resArr = new int[2]; //  resArr[0] = n1 + n2; |

|  |
| --- |
| resArr[1] = n1 - n2;  return resArr;  }  //2. 返回类型可以为任意类型，包含基本类型或引用类型(数组，对象)  // 具体看 getSumAndSub  //  //3. 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;  // 而且要求返回值类型必须和 return 的值类型一致或兼容  public double f1() {  double d1 = 1.1 \* 3;  int n = 100;  return n; // int ->double  //return d1; //ok? double -> int  }  //如果方法是 void ，则方法体中可以没有 return 语句，或者 只写 return ;  //老韩提示：在实际工作中，我们的方法都是为了完成某个功能，所以方法名要有一定含义  // ，最好是见名知意  public void f2() {  System.out.println("hello1");  System.out.println("hello1");  System.out.println("hello1");  int n = 10; |

|  |
| --- |
| //return ;  }  } |

 访问修饰符 (作用是控制 方法使用的范围)

如果不写默认访问，[有四种: public, protected, 默认, private], 具体在后面说

 返回数据类型

1) 一个方法最多有一个返回值 [思考，如何返回多个结果 返回数组 ]

2) 返回类型可以为任意类型，包含基本类型或引用类型(数组，对象)

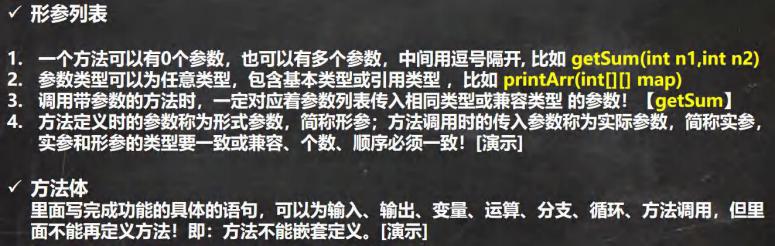
3) 如果方法要求有返回数据类型，则方法体中最后的执行语句必须为 return 值; 而且要求返回值类型必须和 return 的

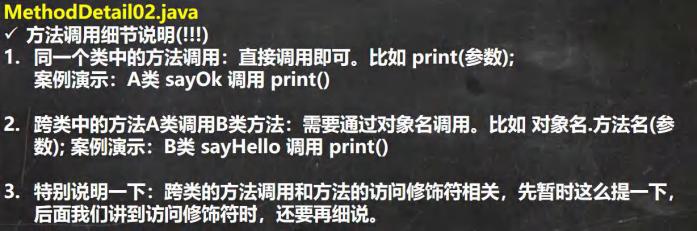
值类型一致或兼容

4) 如果方法是 void ，则方法体中可以没有 return 语句，或者 只写 return ;

 方法名

遵循驼峰命名法，最好见名知义，表达出该功能的意思即可, 比如 得到两个数的和 getSum, 开发中按照规范

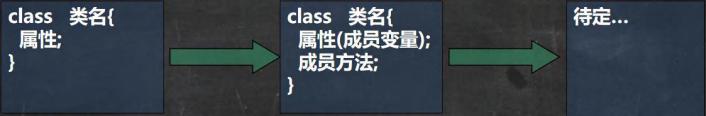




|  |
| --- |
| public class MethodDetail02 {  //编写一个 main 方法  public static void main(String[] args) {  A a = new A();  //a.sayOk();  a.m1();  }  }  class A {  //同一个类中的方法调用：直接调用即可  //  public void print(int n) {  System.out.println("print()方法被调用 n=" + n); |

|  |
| --- |
| }  public void sayOk() { //sayOk 调用 print(直接调用即可)  print(10);  System.out.println("继续执行 sayOK()~~~");  }  //跨类中的方法 A 类调用 B 类方法：需要通过对象名调用  public void m1() {  //创建 B 对象, 然后在调用方法即可  System.out.println("m1() 方法被调用");  B b = new B();  b.hi();  System.out.println("m1() 继续执行:)");  }  }  class B {  public void hi() {  System.out.println("B 类中的 hi()被执行");  }  } |

7.2.8类定义的完善



7.2.9课堂练习题

MethodExercise01.java

1) 编写类 AA ，有一个方法：判断一个数是奇数odd 还是偶数, 返回 boolean

2) 根据行、列、字符打印 对应行数和列数的字符，比如：行：4 ，列：4 ，字符#,则打印相应的效果

老师建议：一定要自己写一遍，不要嫌太简单.

代码

|  |
| --- |
| public class MethodExercise01 {  //编写一个 main 方法  public static void main(String[] args) {  AA a = new AA();  // if(a.isOdd(2)) {//T , 这样的写法以后会看到很多  // System.out.println("是奇数");  // } else {  // System.out.println("是偶数");  // }  // |

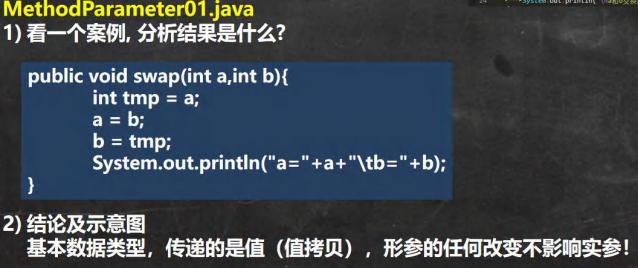
|  |
| --- |
| //  // 使用 print 方法  a.print(4, 4, '#');  }  }  //编写类 AA ，有一个方法：判断一个数是奇数odd 还是偶数, 返回 boolean  class AA {  //思路  //1. 方法的返回类型 boolean  //2. 方法的名字 isOdd  //3. 方法的形参 (int num)  //4. 方法体 , 判断  public boolean isOdd(int num) {  // if(num % 2 != 0) {  // return true;  // } else {  // return false;  // }  //return num % 2 != 0 ? true; false;  //  return num % 2 != 0; |

|  |
| --- |
| }  //根据行、列、字符打印 对应行数和列数的字符，  //比如：行：4 ，列：4 ，字符#,则打印相应的效果  /\*  ####  ####  ####  ####  \*/  //思路  //1. 方法的返回类型 void  //2. 方法的名字 print  //3. 方法的形参 (int row, int col, char c)  //4. 方法体 , 循环  public void print(int row, int col, char c) {  for(int i = 0; i < row; i++) {  for(intj = 0; j < col; j++) {//输出每一行  System.out.print(c);  }  System.out.println(); //换行  }  }  } |

7.3 成员方法传参机制(非常非常重要)

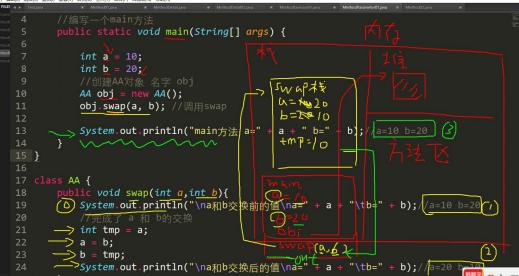
方法的传参机制对我们今后的编程非常重要，一定要搞的清清楚楚明明白白。我们通过案例来学习

7.3. 1基本数据类型的传参机制



|  |
| --- |
| public class MethodParameter01 {  //编写一个 main 方法  public static void main(String[] args) {  int a = 10;  int b = 20;  //创建 AA 对象 名字 obj  AA obj = new AA();  obj.swap(a, b); //调用 swap  System.out.println("main 方法 a=" + a + " b=" + b);//a= 10 b=20  }  } |

|  |
| --- |
| class AA {  public void swap(int a,int b){  System.out.println("\na 和 b 交换前的值\na=" + a + "\tb=" + b);//a= 10 b=20  //完成了 a 和 b 的交换  int tmp = a;  a = b;  b = tmp;  System.out.println("\na 和 b 交换后的值\na=" + a + "\tb=" + b);//a=20 b= 10  }  } |



7.3.2引用数据类型的传参机制

1) 看一个案例 MethodParameter02.java

B 类中编写一个方法 test100 ，可以接收一个数组，在方法中修改该数组，看看原来的数组是否变化？会变化

B 类中编写一个方法 test200，可以接收一个 Person(age,sal)对象，在方法中修改该对象属性，看看原来的对象是否变

化？会变化.

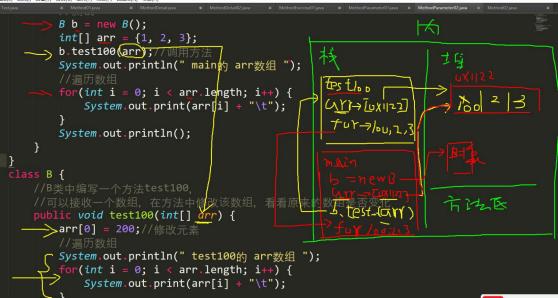
|  |
| --- |
| public class MethodParameter02 {  //编写一个 main 方法  public static void main(String[] args) {  //测试  B b = new B();  // int[] arr = {1, 2, 3};  // b.test100(arr);//调用方法  // System.out.println(" main 的 arr 数组 ");  // //遍历数组  // for(int i = 0; i < arr.length; i++) {  // System.out.print(arr[i] + "\t");  // }  // System.out.println();  //测试  Person p = new Person();  p.name = "jack";  p.age = 10;  b.test200(p);  //测试题, 如果 test200 执行的是 p = null ,下面的结果是 10  //测试题, 如果 test200 执行的是 p = new Person();..., 下面输出的是 10  System.out.println("main 的 p.age=" + p.age);//10000  }  } |

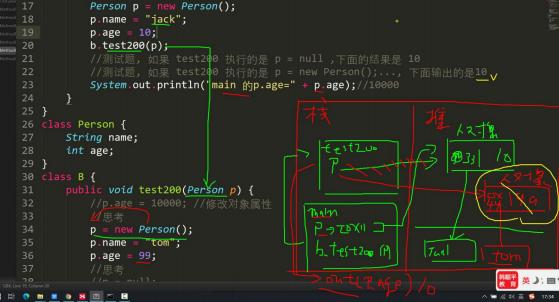
|  |
| --- |
| class Person {  String name;  int age;  }  class B {  public void test200(Person p) {  //p.age = 10000; //修改对象属性  //思考  p = new Person();  p.name = "tom";  p.age = 99;  //思考  //p = null;  }  //B 类中编写一个方法 test100，  //可以接收一个数组，在方法中修改该数组，看看原来的数组是否变化  public void test100(int[] arr) {  arr[0] = 200;//修改元素  //遍历数组  System.out.println(" test100 的 arr 数组 ");  for(int i = 0; i < arr.length; i++) {  System.out.print(arr[i] + "\t");  }  System.out.println();  } |

|  |
| --- |
| } |

2) 结论及示意图

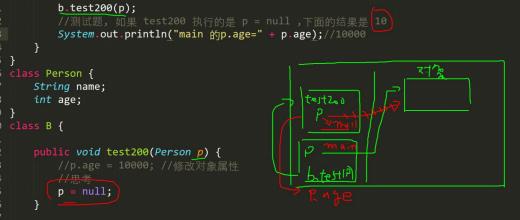
引用类型传递的是地址（传递也是值，但是值是地址），可以通过形参影响实参！

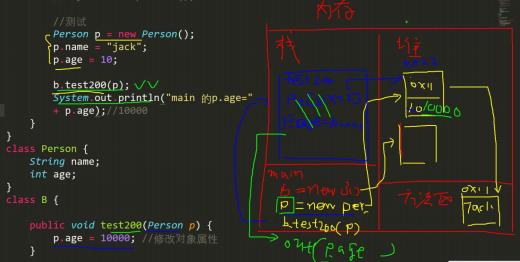




3) 在看一个案例，下面的方法会对原来的对象有影响吗？

p=null 和 p = new Person(); 对应示意图





7.3.3成员方法返回类型是引用类型应用实例

MethodExercise02.java

1) 编写类 MyTools类，编写一个方法可以打印二维数组的数据。

2) 编写一个方法 copyPerson，可以复制一个 Person 对象，返回复制的对象。克隆对象， 注意要求得到新对象和原来的

对象是两个独立的对象，只是他们的属性相同

代码

|  |
| --- |
| public class MethodExercise02 { |

|  |
| --- |
| //编写一个 main 方法  public static void main(String[] args) {  Person p = new Person();  p.name = "milan";  p.age = 100;  //创建 tools  MyTools tools = new MyTools();  Person p2 = tools.copyPerson(p);  //到此 p 和 p2 是 Person对象，但是是两个独立的对象，属性相同  System.out.println("p 的属性 age=" + p.age + " 名字=" + p.name);  System.out.println("p2 的属性 age=" + p2.age + " 名字=" + p2.name);  //这里老师提示： 可以同 对象比较看看是否为同一个对象  System.out.println(p == p2);//false  }  }  class Person {  String name;  int age;  }  class MyTools { |

|  |
| --- |
| //编写一个方法 copyPerson ，可以复制一个 Person 对象，返回复制的对象。克隆对象，  //注意要求得到新对象和原来的对象是两个独立的对象，只是他们的属性相同  //  //编写方法的思路  //1. 方法的返回类型 Person  //2. 方法的名字 copyPerson  //3. 方法的形参 (Person p)  //4. 方法体, 创建一个新对象，并复制属性，返回即可  public Person copyPerson(Person p) {  //创建一个新的对象  Person p2 = new Person();  p2.name = p.name; //把原来对象的名字赋给 p2.name  p2.age = p.age; //把原来对象的年龄赋给 p2.age  return p2;  }  } |

|  |  |
| --- | --- |
|  |  |

7.4 方法递归调用(非常非常重要，比较难)

7.4. 1基本介绍

简单的说: **递归就是方法自己调用自己**,每次调用时传入不同的变量.递归有助于编程者解决复杂问题, 同时可以让代码变

得简洁

7.4.2递归能解决什么问题?



7.4.3递归举例

列举两个小案例,来帮助大家理解递归调用机制

1) 打印问题

2) 阶乘问题

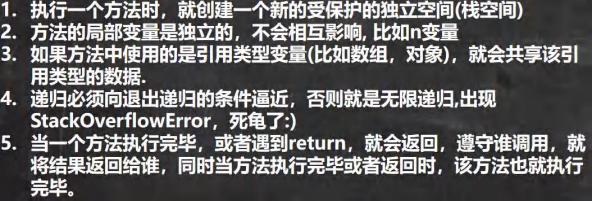
代码

|  |
| --- |
| public class Recursion01 {  //编写一个 main 方法  public static void main(String[] args) {  T t1 = new T();  t1.test(4);//输出什么？ n=2 n=3 n=4  int res = t1.factorial(5);  System.out.println("5 的阶乘 res =" + res);  }  }  class T {  //分析  public void test(int n) {  if (n > 2) {  test(n - 1);  }  System.out.println("n=" + n);  } |

|  |
| --- |
| //factorial 阶乘  public int factorial(int n) {  if (n == 1) {  return 1;  } else {  return factorial(n - 1) \* n;  }  }  } |

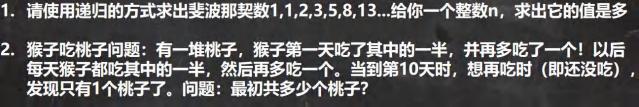
|  |  |
| --- | --- |
|  |  |

7.4.4递归重要规则



7.4.5课堂练习

RecursionExercise01.java



代码

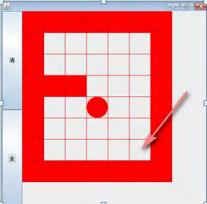
|  |
| --- |
| public class RecursionExercise01 {  //编写一个 main 方法  public static void main(String[] args) {  T t1 = new T();  // int n = 7;  // int res = t1.fibonacci(n);  // if(res != - 1) {  // System.out.println(" 当 n="+ n +" 对应的斐波那契数=" + res);  // }  //  //桃子问题  int day = 0;  int peachNum = t1.peach(day);  if(peachNum != - 1) {  System.out.println("第 " + day + "天有" + peachNum + "个桃子");  } |

|  |
| --- |
| }  }  class T {  /\*  请使用递归的方式求出斐波那契数 1, 1,2,3,5,8, 13...给你一个整数 n ，求出它的值是多  思路分析  1. 当 n = 1 斐波那契数 是 1  2. 当 n = 2 斐波那契数 是 1  3. 当 n >= 3 斐波那契数 是前两个数的和  4. 这里就是一个递归的思路  \*/  public int fibonacci(int n) {  if( n >= 1) {  if( n == 1 || n == 2) {  return 1;  } else {  return fibonacci(n- 1) + fibonacci(n-2);  }  } else {  System.out.println("要求输入的 n>= 1 的整数");  return - 1;  }  } |

|  |
| --- |
| /\*  猴子吃桃子问题：有一堆桃子，猴子第一天吃了其中的一半，并再多吃了一个！  以后每天猴子都吃其中的一半，然后再多吃一个。当到第 10 天时，  想再吃时（即还没吃），发现只有 1 个桃子了。问题：最初共多少个桃子？  思路分析 逆推  1. day = 10 时 有 1 个桃子  2. day = 9 时 有 (day10 + 1) \* 2 = 4  3. day = 8 时 有 (day9 + 1) \* 2 = 10  4. 规律就是 前一天的桃子 = (后一天的桃子 + 1) \*2//就是我们的能力  5. 递归  \*/  public int peach(int day) {  if(day == 10) {//第 10 天，只有 1 个桃  return 1;  } else if ( day >= 1 && day <=9 ) {  return (peach(day + 1) + 1) \* 2;//规则， 自己要想  } else {  System.out.println("day 在 1- 10");  return - 1;  }  }  } |

7.4.6递归调用应用实例-迷宫问题

MiGong.java



代码

|  |
| --- |
| public class MiGong {  //编写一个 main 方法  public static void main(String[] args) {  //思路  //1. 先创建迷宫，用二维数组表示 int[][] map = new int[8][7];  //2. 先规定 map 数组的元素值: 0 表示可以走 1 表示障碍物  int[][] map = new int[8][7];  //3. 将最上面的一行和最下面的一行，全部设置为 1  for(int i = 0; i < 7; i++) {  map[0][i] = 1;  map[7][i] = 1;  }  //4.将最右面的一列和最左面的一列，全部设置为 1  for(int i = 0; i < 8; i++) { |

|  |
| --- |
| map[i][0] = 1;  map[i][6] = 1;  }  map[3][1] = 1;  map[3][2] = 1;  map[2][2] = 1; //测试回溯  // map[2][1] = 1;  // map[2][2] = 1;  // map[1][2] = 1;  //输出当前的地图  System.out.println("=====当前地图情况======");  for(int i = 0; i < map.length; i++) {  for(intj = 0; j < map[i].length; j++) {  System.out.print(map[i][j] + " ");//输出一行  }  System.out.println();  }  //使用 findWay 给老鼠找路  T t1 = new T();  //下右上左  t1.findWay(map, 1, 1);  System.out.println("\n====找路的情况如下====="); |

|  |
| --- |
| for(int i = 0; i < map.length; i++) {  for(intj = 0; j < map[i].length; j++) {  System.out.print(map[i][j] + " ");//输出一行  }  System.out.println();  }  }  }  class T {  //使用递归回溯的思想来解决老鼠出迷宫  //老韩解读  //1. findWay 方法就是专门来找出迷宫的路径  //2. 如果找到，就返回 true ,否则返回 false  //3. map 就是二维数组，即表示迷宫  //4. i,j 就是老鼠的位置，初始化的位置为(1, 1)  //5. 因为我们是递归的找路，所以我先规定 map 数组的各个值的含义  // 0 表示可以走 1 表示障碍物 2 表示可以走 3 表示走过，但是走不通是死路  //6. 当 map[6][5] =2 就说明找到通路,就可以结束，否则就继续找.  //7. 先确定老鼠找路策略 下->右->上->左 |

|  |
| --- |
| public boolean findWay(int[][] map , int i, intj) {  if(map[6][5] == 2) {//说明已经找到  return true;  } else {  if(map[i][j] == 0) {//当前这个位置 0,说明表示可以走  //我们假定可以走通  map[i][j] = 2;  //使用找路策略，来确定该位置是否真的可以走通  //下->右->上->左  if(findWay(map, i + 1, j)) {//先走下  return true;  } else if(findWay(map, i, j + 1)){//右  return true;  } else if(findWay(map, i- 1, j)) {//上  return true;  } else if(findWay(map, i, j- 1)){//左  return true;  } else {  map[i][j] = 3;  return false;  }  } else { //map[i][j] = 1 , 2, 3  return false;  }  } |

|  |
| --- |
| }  //修改找路策略，看看路径是否有变化  //下->右->上->左 ==> 上->右->下->左  public boolean findWay2(int[][] map , int i, intj) {  if(map[6][5] == 2) {//说明已经找到  return true;  } else {  if(map[i][j] == 0) {//当前这个位置 0,说明表示可以走  //我们假定可以走通  map[i][j] = 2;  //使用找路策略，来确定该位置是否真的可以走通  //上->右->下->左  if(findWay2(map, i - 1, j)) {//先走上  return true;  } else if(findWay2(map, i, j + 1)){//右  return true;  } else if(findWay2(map, i+1, j)) {//下  return true;  } else if(findWay2(map, i, j- 1)){//左  return true;  } else {  map[i][j] = 3;  return false;  }  } else { //map[i][j] = 1 , 2, 3 |

|  |
| --- |
| return false;  }  }  }  } |

7.4.7递归调用应用实例-汉诺塔

 汉诺塔传说

汉诺塔：汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子， 在一根柱子上从下往上按照大小顺序摞着 64 片圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一

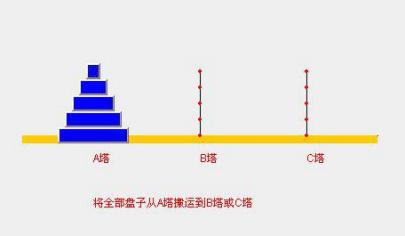
根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

假如每秒钟移动一次，共需多长时间呢？移完这些金片需要 5845.54 亿年以上，太阳系的预期寿命据说也就是数百亿年。

真的过了 5845.54 亿年，地球上的一切生命，连同梵塔、庙宇等，都早已经灰飞烟灭

 汉诺塔代码实现

看老师代码演示 HanoiTower.java



代码

|  |
| --- |
| public class HanoiTower {  //编写一个 main 方法  public static void main(String[] args) {  Tower tower = new Tower();  tower.move(64, 'A', 'B', 'C');  }  }  class Tower {  //方法  //num 表示要移动的个数, a, b, c 分别表示 A 塔，B 塔, C 塔  public void move(int num , char a, char b ,char c) {  //如果只有一个盘 num = 1 |

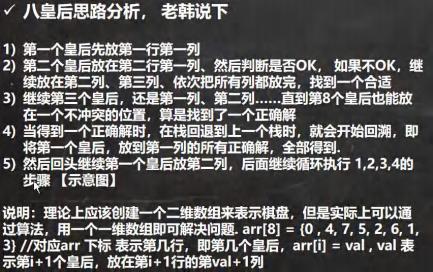
|  |
| --- |
| if(num == 1) {  System.out.println(a + "->" + c);  } else {  //如果有多个盘，可以看成两个 , 最下面的和上面的所有盘(num- 1)  //(1)先移动上面所有的盘到 b, 借助 c  move(num - 1 , a, c, b);  //(2)把最下面的这个盘，移动到 c  System.out.println(a + "->" + c);  //(3)再把 b 塔的所有盘，移动到 c ,借助 a  move(num - 1, b, a, c);  }  }  } |

7.4.8递归调用应用实例-八皇后问题[同学们先尝试做，后面老师评讲.]

 八皇后问题说明

八皇后问题，是一个古老而著名的问题，是回溯算法的典型案例。该问题是国际西洋棋棋手马克斯·贝瑟尔于 1848 年 提出：在 8×8 格的国际象棋上摆放八个皇后，使其不能互相攻击，即：任意两个皇后都不能处于同一行、同一列或同

一斜线上，问有多少种摆法。



7.5 方法重载(OverLoad)

7.5. 1基本介绍

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！

比如：System.out.println(); out 是 PrintStream 类型

OverLoad01.java

7.5.2重载的好处

1) 减轻了起名的麻烦

2) 减轻了记名的麻烦

7.5.3快速入门案例

OverLoad01.java

案例：类：MyCalculator 方法：calculate

1) calculate(int n1, int n2) //两个整数的和

2) calculate(int n1, double n2) //一个整数，一个 double 的和

3) calculate(double n2, int n1)//一个 double ,一个 Int 和

4) calculate(int n1, int n2,int n3)//三个 int 的和

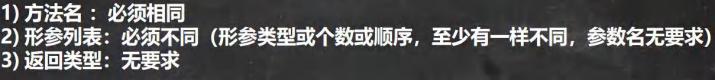
代码

|  |
| --- |
| public class OverLoad01 {  //编写一个 main 方法  public static void main(String[] args) {  // System.out.println(100);  // System.out.println("hello,world");  // System.out.println('h');  // System.out.println(1. 1);  // System.out.println(true);  //  MyCalculator mc = new MyCalculator();  System.out.println(mc.calculate(1, 2));  System.out.println(mc.calculate(1.1, 2));  System.out.println(mc.calculate(1, 2. 1));  }  }  class MyCalculator {  //下面的四个 calculate 方法构成了重载  //两个整数的和  public int calculate(int n1, int n2) {  System.out.println("calculate(int n1, int n2) 被调用"); |

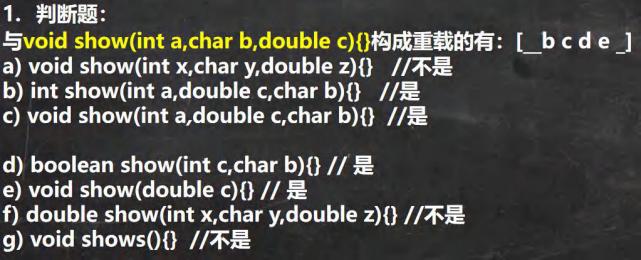
|  |
| --- |
| return n1 + n2;  }  //没有构成方法重载, 仍然是错误的，因为是方法的重复定义  // public void calculate(int n1, int n2) {  // System.out.println("calculate(int n1, int n2) 被调用");  // int res = n1 + n2;  // }  //看看下面是否构成重载, 没有构成，而是方法的重复定义,就错了  // public int calculate(int a1, int a2) {  // System.out.println("calculate(int n1, int n2) 被调用");  // return a1 + a2;  // }  //一个整数，一个 double 的和  public double calculate(int n1, double n2) {  return n1 + n2;  }  //一个 double ,一个 Int 和  public double calculate(double n1, int n2) {  System.out.println("calculate(double n1, int n2) 被调用..");  return n1 + n2;  }  //三个 int 的和  public int calculate(int n1, int n2,int n3) { |

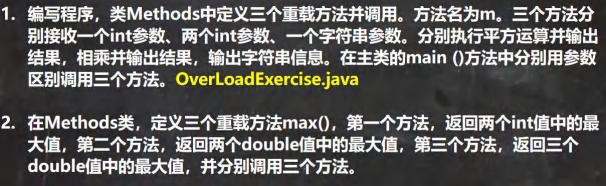
|  |
| --- |
| return n1 + n2 + n2;  }  } |

7.5.4注意事项和使用细节



7.5.5课堂练习题





|  |
| --- |
| public class OverLoadExercise {  //编写一个 main 方法  public static void main(String[] args) {  //在主类的 main ()方法中分别用参数区别调用三个方法  Methods method = new Methods();  method.m(10);//100  method.m(10, 20);//200  method.m("韩顺平教育 hello");//字符串信息  //测试  System.out.println(method.max(10, 24)); // 24  System.out.println(method.max(10.0, 21.4)); // 21.4  System.out.println(method.max(10.0, 1.4, 30.0)); // 30.0  }  }  /\*  编写程序，类 Methods 中定义三个重载方法并调用。方法名为 m。  三个方法分别接收一个 int参数、两个 int参数、一个字符串参数。分别执行平方运算并输出结果，  相乘并输出结果，输出字符串信息。在主类的 main ()方法中分别用参数区别调用三个方法  定义三个重载方法 max() ，第一个方法，返回两个 int值中的最大值，  第二个方法，返回两个 double值中的最大值，第三个方法， |

|  |
| --- |
| 返回三个 double值中的最大值，并分别调用三个方法  \*/  class Methods {  //分析  //1 方法名 max  //2 形参 (int,int)  //3.int  public int max(int n1, int n2) {  return n1 > n2 ? n1 : n2;  }  //分析  //1 方法名 max  //2 形参 (double,double)  //3.double  public double max(double n1, double n2) {  return n1 > n2 ? n1 : n2;  }  //分析  //1 方法名 max  //2 形参 (double,double,double)  //3.double  public double max(double n1, double n2, double n3) { |

|  |
| --- |
| System.out.println("max(double n1, double n2, double n3)");  //求出n1 和 n2 的最大值  double max1 = n1 > n2 ? n1 : n2;  return max1 > n3 ? max1 : n3;  }  public double max(double n1, double n2, int n3) {  System.out.println("max(double n1, double n2, int n3)");  //求出n1 和 n2 的最大值  double max1 = n1 > n2 ? n1 : n2;  return max1 > n3 ? max1 : n3;  }  //分析  //1 方法名 m  //2 形参 (int)  //3.void  public void m(int n) {  System.out.println("平方=" + (n \* n));  }  //1 方法名 m  //2 形参 (int, int)  //3.void |

|  |
| --- |
| public void m(int n1, int n2) {  System.out.println("相乘=" + (n1 \* n2));  }  //1 方法名 m  //2 形参 (String)  //3.void  public void m(String str) {  System.out.println("传入的 str=" + str);  }  } |

7.6 可变参数

7.6. 1基本概念

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。

就可以通过可变参数实现

7.6.2基本语法

访问修饰符 返回类型 方法名(数据类型... 形参名) {

}

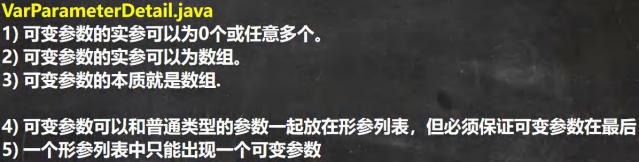
7.6.3快速入门案例(VarParameter01.java)

看一个案例 类 HspMethod ，方法 sum 【可以计算 2 个数的和，3 个数的和 ， 4. 5 ， 。。】

|  |
| --- |
| public class VarParameter01 {  //编写一个 main 方法  public static void main(String[] args) {  HspMethod m = new HspMethod();  System.out.println(m.sum(1, 5, 100)); //106  System.out.println(m.sum(1, 19)); //20  }  }  class HspMethod {  //可以计算 2 个数的和，3 个数的和 ， 4. 5 ， 。。  //可以使用方法重载  // public int sum(int n1, int n2) {//2 个数的和  // return n1 + n2;  // }  // public int sum(int n1, int n2, int n3) {//3 个数的和  // return n1 + n2 + n3;  // }  // public int sum(int n1, int n2, int n3, int n4) {//4 个数的和  // return n1 + n2 + n3 + n4;  // }  //.....  //上面的三个方法名称相同，功能相同, 参数个数不同-> 使用可变参数优化  //老韩解读 |

|  |
| --- |
| //1. int... 表示接受的是可变参数，类型是 int , 即可以接收多个 int(0-多)  //2. 使用可变参数时，可以当做数组来使用 即 nums 可以当做数组  //3. 遍历 nums 求和即可  public int sum(int... nums) {  //System.out.println("接收的参数个数=" + nums.length);  int res = 0;  for(int i = 0; i < nums.length; i++) {  res += nums[i];  }  return res;  }  } |

7.6.4注意事项和使用细节

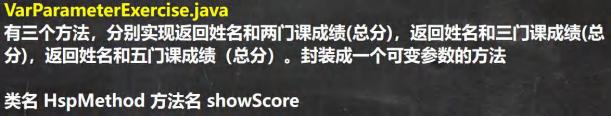


代码

|  |
| --- |
| public class VarParameterDetail {  //编写一个 main 方法  public static void main(String[] args) { |

|  |
| --- |
| //细节: 可变参数的实参可以为数组  int[] arr = {1, 2, 3};  T t1 = new T();  t1.f1(arr);  }  }  class T {  public void f1(int... nums) {  System.out.println("长度=" + nums.length);  }  //细节: 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  public void f2(String str, double... nums) {  }  //细节: 一个形参列表中只能出现一个可变参数  //下面的写法是错的.  // public void f3(int... nums1, double... nums2) {  // }  } |

7.6.5课堂练习

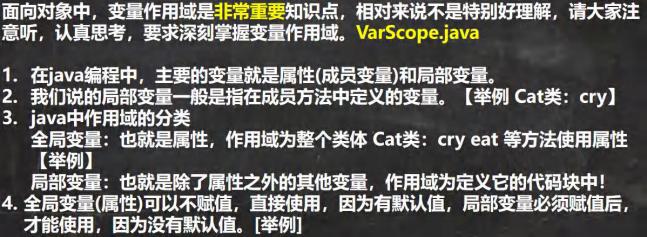


|  |
| --- |
| public class VarParameterExercise {  //编写一个 main 方法  public static void main(String[] args) {  HspMethod hm = new HspMethod();  System.out.println(hm.showScore("milan" , 90.1, 80.0 ));  System.out.println(hm.showScore("terry" , 90.1, 80.0, 10,30.5,70 ));  }  }  class HspMethod {  /\*  有三个方法，分别实现返回姓名和两门课成绩(总分)，  返回姓名和三门课成绩(总分) ，返回姓名和五门课成绩（总分）。  封装成一个可变参数的方法  \*/  //分析 1. 方法名 showScore 2. 形参(String ,double... ) 3. 返回 String  //听课小伙伴，老师要求必须自己动手写  public String showScore(String name ,double... scores ) { |

|  |
| --- |
| double totalScore = 0;  for(int i = 0; i < scores.length; i++) {  totalScore += scores[i];  }  return name + " 有 " +scores.length + " 门课的成绩总分为=" + totalScore;  }  } |

7.7 作用域

7.7. 1基本使用

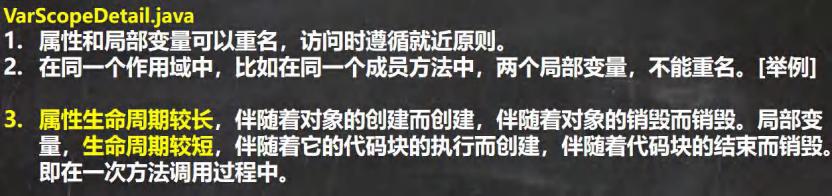


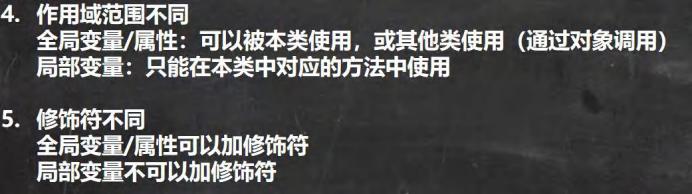
|  |
| --- |
| public class VarScope {  //编写一个 main 方法  public static void main(String[] args) {  }  } |

|  |
| --- |
| class Cat {  //全局变量：也就是属性，作用域为整个类体 Cat类：cry eat 等方法使用属性  //属性在定义时，可以直接赋值  int age = 10; //指定的值是 10  //全局变量(属性)可以不赋值，直接使用，因为有默认值，  double weight; //默认值是 0.0  public void hi() {  //局部变量必须赋值后，才能使用，因为没有默认值  int num = 1;  String address = "北京的猫";  System.out.println("num=" + num);  System.out.println("address=" + address);  System.out.println("weight=" + weight);//属性  }  public void cry() {  //1. 局部变量一般是指在成员方法中定义的变量  //2. n 和 name 就是局部变量  //3. n 和 name 的作用域在 cry 方法中  int n = 10;  String name = "jack";  System.out.println("在 cry 中使用属性 age=" + age);  } |

|  |
| --- |
| public void eat() {  System.out.println("在 eat 中使用属性 age=" + age);  //System.out.println("在 eat 中使用 cry 的变量 name=" + name);//错误  }  } |

7.7.2注意事项和细节使用





|  |
| --- |
| public class VarScopeDetail { |

|  |
| --- |
| //编写一个 main 方法  public static void main(String[] args) {  Person p1 = new Person();  /\*  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。即在一次方法调用过程中  \*/  //p1.say();//当执行 say 方法时，say 方法的局部变量比如 name,会创建，当 say 执行完毕后  //name 局部变量就销毁,但是属性(全局变量)仍然可以使用  //  T t1 = new T();  t1.test(); //第 1 种跨类访问对象属性的方式  t1.test2(p1);//第 2 种跨类访问对象属性的方式  }  }  class T {  //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)  public void test() {  Person p1 = new Person();  System.out.println(p1.name);//jack  } |

|  |
| --- |
| public void test2(Person p) {  System.out.println(p.name);//jack  }  }  class Person {  //细节: 属性可以加修饰符(public protected private..)  // 局部变量不能加修饰符  public int age = 20;  String name = "jack";  public void say() {  //细节 属性和局部变量可以重名，访问时遵循就近原则  String name = "king";  System.out.println("say() name=" + name);  }  public void hi() {  String address = "北京";  //String address = "上海";//错误,重复定义变量  String name = "hsp";//可以  }  } |

7.8 构造方法/构造器

7.8. 1看一个需求

我们来看一个需求：前面我们在创建人类的对象时，是先把一个对象创建好后，再给他的年龄和姓名属性赋值，如

果现在我要求，在创建人类的对象时，就直接指定这个对象的年龄和姓名，该怎么做? 这时就可以使用构造器。

7.8.2基本语法

[修饰符] 方法名(形参列表){

方法体;

}

> 老韩说明：

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 和类名字必须一样

4) 参数列表 和 成员方法一样的规则

5) 构造器的调用, 由系统完成

7.8.3基本介绍

构造方法又叫构造器(constructor) ，是类的一种特殊的方法，它的主要作用是完成对新对象的初始化。它有几个特点：

1) 方法名和类名相同

2) 没有返回值

3) 在创建对象时，系统会自动的调用该类的构造器完成对象的初始化。

7.8.4快速入门

现在我们就用构造方法来完成刚才提出的问题: 在创建人类的对象时 ，就直接指定这个对象的年龄和姓名

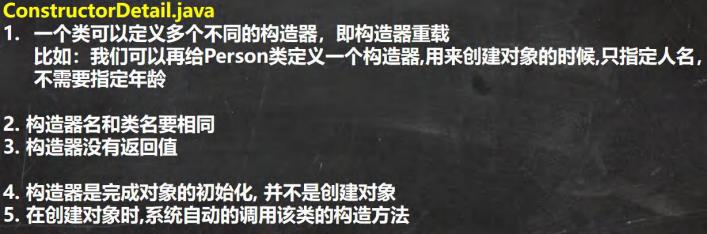
Constructor01.java

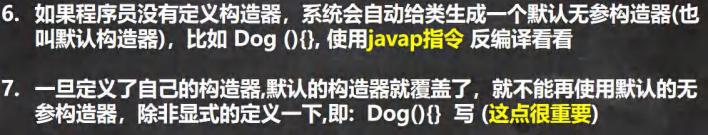
|  |
| --- |
| public class Constructor01 { |

|  |
| --- |
| //编写一个 main 方法  public static void main(String[] args) {  //当我们 new 一个对象时，直接通过构造器指定名字和年龄  Person p1 = new Person("smith", 80);  System.out.println("p1 的信息如下");  System.out.println("p1 对象 name=" + p1.name);//smith  System.out.println("p1 对象 age=" + p1.age);//80  }  }  //在创建人类的对象时，就直接指定这个对象的年龄和姓名  //  class Person {  String name;  int age;  //构造器  //老韩解读  //1. 构造器没有返回值, 也不能写 void  //2. 构造器的名称和类 Person 一样  //3. (String pName, int pAge) 是构造器形参列表，规则和成员方法一样  public Person(String pName, int pAge) {  System.out.println("构造器被调用~~ 完成对象的属性初始化");  name = pName;  age = pAge;  } |

|  |
| --- |
| } |

7.8.5注意事项和使用细节





|  |
| --- |
| public class ConstructorDetail {  //编写一个 main 方法  public static void main(String[] args) {  Person p1 = new Person("king", 40);//第 1 个构造器  Person p2 = new Person("tom");//第 2 个构造器  Dog dog1 = new Dog();//使用的是默认的无参构造器 |

|  |
| --- |
| }  }  class Dog {  //如果程序员没有定义构造器，系统会自动给类生成一个默认无参构造器(也叫默认构造器)  //使用javap 指令 反编译看看  /\*  默认构造器  Dog() {  }  \*/  //一旦定义了自己的构造器,默认的构造器就覆盖了，就不能再使用默认的无参构造器，  //除非显式的定义一下, 即: Dog(){} 写 (这点很重要)  //  public Dog(String dName) {  //...  }  Dog() { //显式的定义一下 无参构造器  }  }  class Person {  String name;  int age;//默认 0  //第 1 个构造器 |

|  |
| --- |
| public Person(String pName, int pAge) {  name = pName;  age = pAge;  }  //第 2 个构造器, 只指定人名，不需要指定年龄  public Person(String pName) {  name = pName;  }  } |

7.8.6课堂练习题

ConstructorExercise.java

在前面定义的 Person类中添加两个构造器：

第一个无参构造器：利用构造器设置所有人的 age 属性初始值都为 18

第二个带 pName和 pAge 两个参数的构造器：使得每次创建 Person对象的同时初始化对象的 age 属性值和 name 属性值。

分别使用不同的构造器，创建对象.

代码:

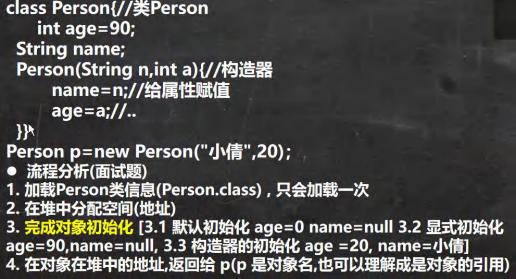
|  |
| --- |
| public class ConstructorExercise {  //编写一个 main 方法  public static void main(String[] args) {  Person p1 = new Person();//无参构造器  //下面输出 name = null, age = 18 |

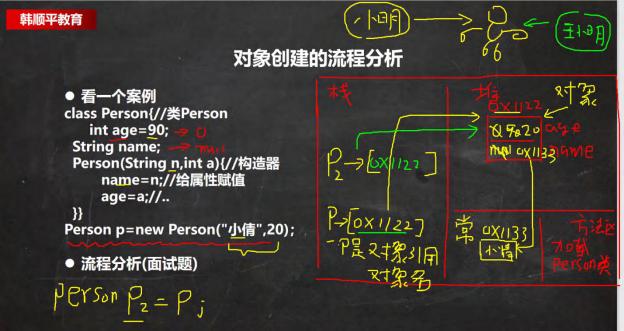
|  |
| --- |
| System.out.println("p1 的信息 name=" + p1.name + " age=" + p1.age);  Person p2 = new Person("scott", 50);  //下面输出 name = scott, age = 50  System.out.println("p2 的信息 name=" + p2.name + " age=" + p2.age);  }  }  /\*\*  \* 在前面定义的 Person类中添加两个构造器:  \* 第一个无参构造器：利用构造器设置所有人的 age 属性初始值都为 18  \* 第二个带 pName 和 pAge 两个参数的构造器：  \* 使得每次创建 Person对象的同时初始化对象的 age 属性值和 name 属性值。  \* 分别使用不同的构造器，创建对象.  \*/  class Person {  String name;//默认值 null  int age;//默认 0  //第一个无参构造器：利用构造器设置所有人的 age 属性初始值都为 18  public Person() {  age = 18;//  }  //第二个带 pName 和 pAge 两个参数的构造器  public Person(String pName, int pAge) { |

|  |
| --- |
| name = pName;  age = pAge;  }  } |

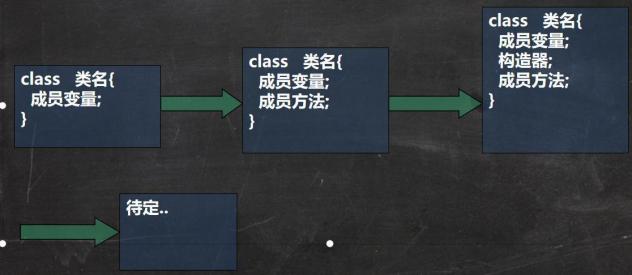
7.9 对象创建的流程分析

7.9. 1看一个案例



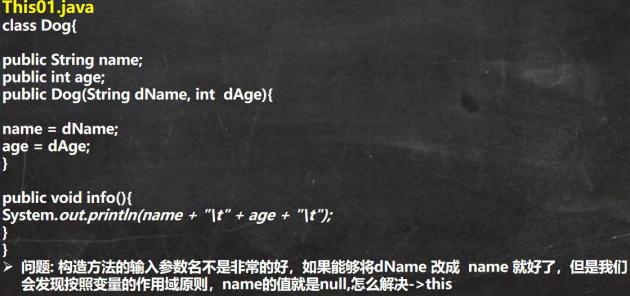


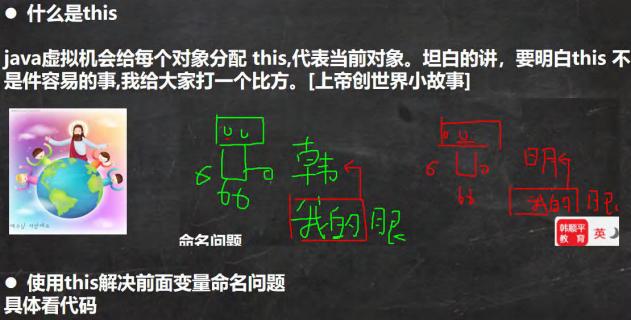
学习完构造器后,我们类的定义就应该更加完善了



7.10 this 关键字

7.10.1 先看一段代码,并分析问题





代码:

|  |
| --- |
| public class This01 {  //编写一个 main 方法 |

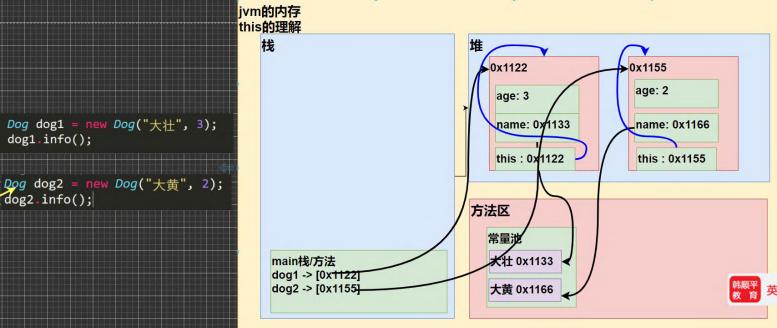
|  |
| --- |
| public static void main(String[] args) {  Dog dog1 = new Dog("大壮", 3);  System.out.println("dog1 的 hashcode=" + dog1.hashCode());  //dog1 调用了 info()方法  dog1.info();  System.out.println("============");  Dog dog2 = new Dog("大黄", 2);  System.out.println("dog2 的 hashcode=" + dog2.hashCode());  dog2.info();  }  }  class Dog{ //类  String name;  int age;  // public Dog(String dName, int dAge){//构造器  // name = dName;  // age = dAge;  // }  //如果我们构造器的形参，能够直接写成属性名，就更好了  //但是出现了一个问题，根据变量的作用域原则  //构造器的 name 是局部变量，而不是属性  //构造器的 age 是局部变量，而不是属性 |

|  |
| --- |
| //==> 引出 this 关键字来解决  public Dog(String name, int age){//构造器  //this.name 就是当前对象的属性 name  this.name = name;  //this.age 就是当前对象的属性 age  this.age = age;  System.out.println("this.hashCode=" + this.hashCode());  }  public void info(){//成员方法,输出属性 x 信息  System.out.println("this.hashCode=" + this.hashCode());  System.out.println(name + "\t" + age + "\t");  }  } |

7.10.2 深入理解 this

为了进一步理解 this,我们再看一个案例 (This01.java)





7.10.3 this 的注意事项和使用细节

ThisDetail.java

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); 注意只能在构造器中使用(即只能在构造器中访问另外一个构造器, 必须放在第一

条语句)

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。

7.10.4 this 的课堂案例

TestPerson.java

定义 Person类，里面有 name、age 属性，并提供compareTo 比较方法，用于判断是否和另一个人相等，提供测试类 TestPerson

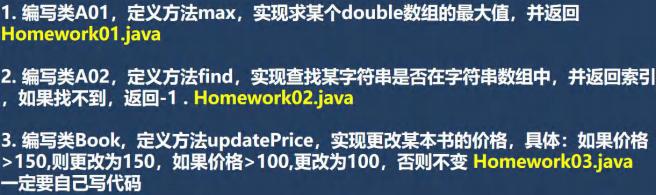
用于测试, 名字和年龄完全一样，就返回 true, 否则返回 false

代码

|  |
| --- |
| public class TestPerson {  //编写一个 main 方法  public static void main(String[] args) {  Person p1 = new Person("mary", 20);  Person p2 = new Person("mary", 20);  System.out.println("p1 和 p2 比较的结果=" + p1.compareTo(p2));  }  }  /\*  定义 Person 类，里面有 name 、age 属性，并提供 compareTo 比较方法，  用于判断是否和另一个人相等，提供测试类 TestPerson用于测试,  名字和年龄完全一样，就返回 true, 否则返回 false |

|  |
| --- |
| \*/  class Person {  String name;  int age;  //构造器  public Person(String name, int age) {  this.name = name;  this.age = age;  }  //compareTo 比较方法  public boolean compareTo(Person p) {  //名字和年龄完全一样  // if(this.name.equals(p.name) && this.age == p.age) {  // return true;  // } else {  // return false;  // }  return this.name.equals(p.name) && this.age == p.age;  }  } |

7. 11 本章作业



眼高手低->千万不要这样.->零基础

