# 面向过程

分析解决问题的所需要的步骤，然后我们用函数来一步一步的调用实现。

举例：把大象放入冰箱

（1）打开冰箱门

（2）放入大象

（3）关闭冰箱

面向过程的思想是注重做事的过程。

范例2：如果使用面向过程来设计五子棋:

我们要考虑的步骤：

1.开始游戏

2.黑子先走

3.绘制画面

4.判断输赢

5.白子再走

6.绘制画面

7.判断输赢

# 面向对象

OOP（Object Oriented Programming）面向对象编程。

把做事情抽象出来实体

冰箱

行为：开门，关门

大象

行为：走路

面向对象关注的是谁有打开冰箱关闭冰箱的功能，谁有进入冰箱的功能。

面向对象注重的是哪些事物有什么样的功能，强调的是具体的事物。

分析： 1.五子棋分为黑白双方：双方的行为是一样的。

2.棋盘系统：负责绘制画面行为。

3.规则系统：有判定犯规，输赢的功能

面向对象是以功能和属性来划分，而不是步骤。

编程就是模拟现实生活，我们使用面向对象的思想可以认为“万物皆对象”。

# 类

我们把某些具有相同的属性和行为的事物抽象成一类。

属性和行为

狗：颜色，腿，眼睛，会汪汪

看家，吃饭，睡觉

小汽车：

属性：轮子，颜色，排量

行为：前进，后退，刹车…

…….

定义一个概念就是java中的类，这个类的一个具体的实例叫做对象。

狗类---------》旺财

小汽车-------》路边的那个某老师的小汽车

方法 :

权限修饰符 <static> 返回值[void] 方法名字(参数列表){

//方法体  
 [return xxx]

}

类的定义：

|  |
| --- |
| /\*\*  如果类前面有public 那么类名和文件名字一定要一致  \*/  public class Dog{    /\*\*  颜色  \*/  String color;    /\*\*  名字  \*/  String name;    /\*\*  性别 1.公，2母  \*/  int gender;      /\*\*  睡觉  \*/    public void sleep(){  System.out.println("狗狗在睡觉");  }    /\*\*  看家, 方法名字和类名都要使用驼峰模式来命名：  \*/  public void lookHome(){  System.out.println("狗狗在看家");  }    /\*\*  显示狗的信息  \*/    } |

# 对象

类就是用来创建对象。

我们可以根据类来创建对象

创建对象的语法：类名 变量名 = new 类名();

类也是一种数据类型

|  |
| --- |
| /\*\*  如果类前面有public 那么类名和文件名字一定要一致  \*/  public class Dog{      /\*\*  颜色  \*/  String color;    /\*\*  名字  \*/  String name;    /\*\*  性别 1.公，2母  \*/  int gender;      /\*\*  睡觉  \*/    public void sleep(){  System.out.println("狗狗在睡觉");  }    /\*\*  看家, 方法名字和类名都要使用驼峰模式来命名：  \*/  public void lookHome(){  System.out.println("狗狗在看家");  }    /\*\*  显示狗的信息  \*/      public static void main(String[] args){      //类名 变量名 = new 类名();  //数据类型（类） 变量名 = new 数据类型（类）();  Dog d = new Dog();  //初始化属性  d.color = "黄色";  d.name = "旺财";  d.gender = 2;    System.out.println("我家的一只狗：颜色是 "+d.color+" 名字是 "+d.name+" 性别是："+(d.gender == 1?"公狗狗":"母狗狗"));  //使用对象来调用方法 变量.方法(参数)  d.lookHome();  d.sleep();            Dog d1 = new Dog();  //初始化属性  d1.color = "白色";  d1.name = "小白";  d1.gender = 1;    System.out.println("我家的一只狗：颜色是 "+d1.color+" 名字是 "+d1.name+" 性别是："+(d1.gender == 1?"公狗狗":"母狗狗"));  d1.sleep();  d1.lookHome();    }  } |

# 内存结构图

|  |
| --- |
| public class Demo{  public static void main(String[] args){  int i = 10;  }  } |

Java Demo的时候在内存中创建jvm，jvm是内存中的一块区域，这块区域里面有栈，堆，数据共享区

Jvm

栈 堆

数据共享区

main

Int i

10

Demo.class

main

栈：存储基本数据类型，引用数据类型的引用

堆：存储引用数据类型

Jvm

栈 堆

数据共享区

0x98 new int[3];

|  |  |
| --- | --- |
| 0 | 10 |
| 1 | 1 |
| 2 | 6 |

main

int[] arr

0x98

Demo.class

main

Jvm

栈 堆

数据共享区

0x98 New Dog();

0x99 New Dog();

color= "黄色"

name= "旺财"

gender=2

main

Dog d1

Dog d

0x99

0x98

color= "白色"

name= "小白"

gender=1

Dog.class

Main()

Sleep()

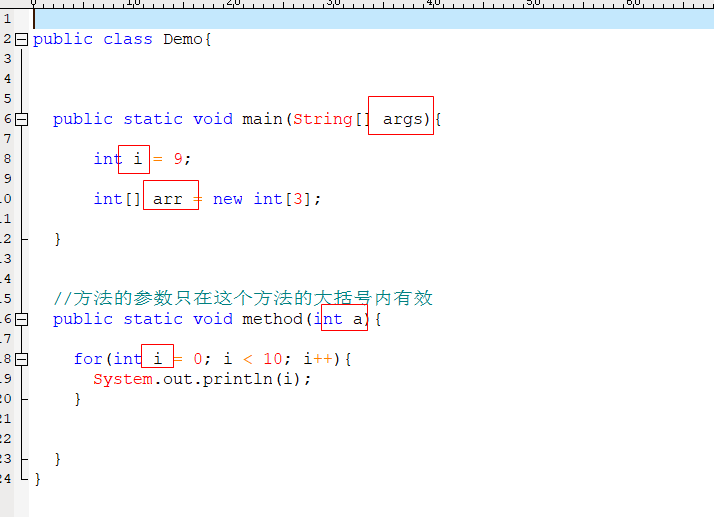
lookHome();

掌握对象的创建是在堆中，对象和对象之间的数据是完全隔离的，引用是存储在栈中。

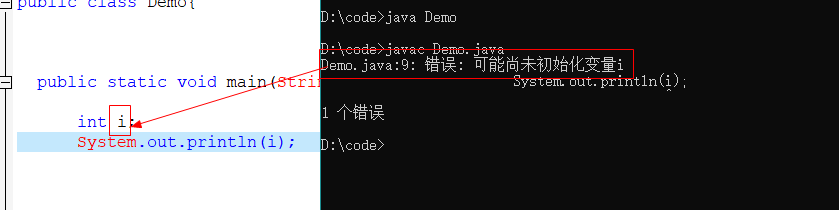
同学们至少画一遍内存结构图。

# 局部变量

除了类中的属性以外的变量都是局部变量。



* 作用范围：只限于所定义的大括号。
* 存储的位置：存储在栈内存中。
* 默认值：无，局部变量在使用之前必须赋值。



* 生命周期：所属的大括号结束就消失。

# 成员变量（属性）

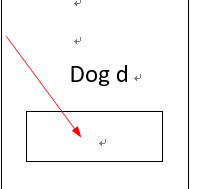
* 成员变量是定义在类中，也可以叫类的属性。
* 作用范围：在整个类中
* 存储位置：堆内存或者数据共享区
* 默认值：有

|  |  |
| --- | --- |
| 数据类型 | 默认值 |
| byte | 0 |
| short | 0 |
| int | 0 |
| long | 0 |
| float | 0.0 |
| double | 0.0 |
| char | ‘ ’ |
| boolean | false |
| 引用数据类型 | null |

* 生命周期：由jvm通过垃圾回收机制来自动回收。

Jvm里面有垃圾回收器：一些匿名对象（没有引用的对象）会被回收。

Null只在引用数据类型里面

Jvm

栈 堆

数据共享区

0x98 New Dog();

color= null

name= null

gender=0

main

Dog d

Dog.class

Main()

Sleep()

lookHome();

# 匿名对象

匿名对象就是没有栈中的引用的对象，这种对象很快就会被gc（垃圾回收器）回收掉。

|  |
| --- |
| public class Person{    String name;    int age;    String addr;      public void sleep(){  System.out.println("人在睡觉");    }    public void info(){  System.out.println("展示这个人的信息");    }      public static void main(String[] args){  //类名 变量名 = new 类名();    Person person = new Person();  person.name = "亮哥";  person.age = 18;  person.addr = "火星";        }    } |

栈 堆

main

0x98 new Person()

Name = "亮哥";

Age = 18;

Addr = "火星";

Person person

0x98

数据共享区

Person.class

Main()

sleep()

info()

匿名对象

|  |
| --- |
| public class Person{    String name;    int age;    String addr;      public void sleep(){  System.out.println("人在睡觉");    }    public void info(){  System.out.println("展示这个人的信息");    }      public static void main(String[] args){  //类名 变量名 = new 类名();    new Person();    }    } |

栈 堆

main

0x98 new Person()

0x99 new Person()

Name = “亮哥”;

Age = 0;

Addr = null;

Name = null;

Age = 18;

Addr = null;

数据共享区

Person.class

Main()

sleep()

info()

平时开发的时候要避免匿名对象。

# 封装

面向对象三大特征：封装，继承，多态。

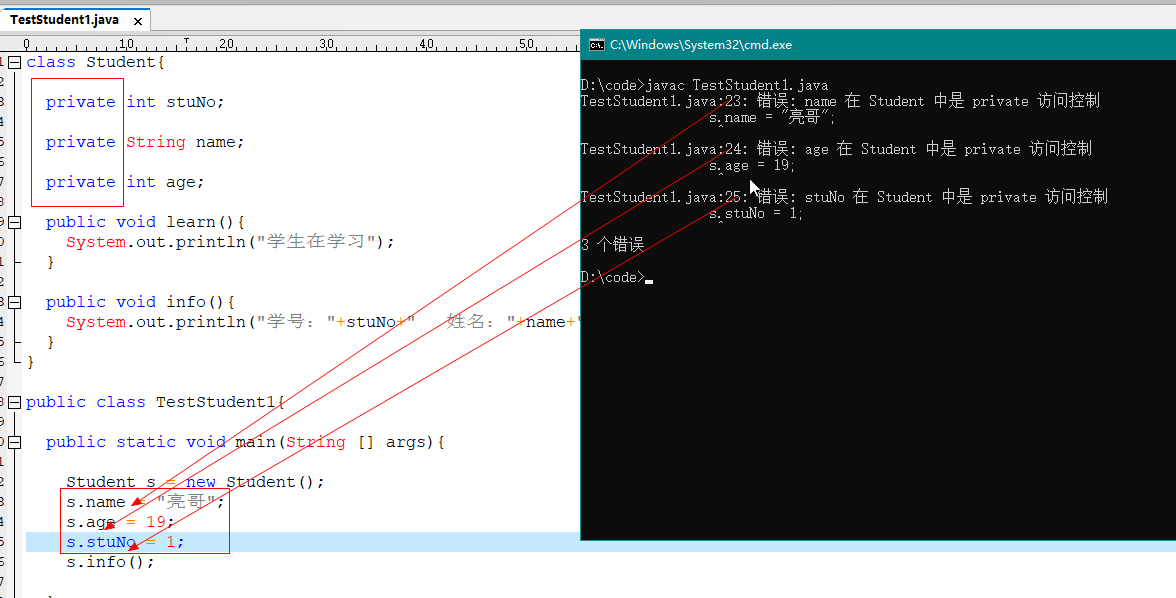
目的：提高数据安全性，通过封装，可以实现对属性的访问权限控制，同时增加了程序的可维护性。

封装：

将属性私有化，使用private关键字完成属性私有化，这种属性只有本类能访问。

我们一般对每一个私有属性提供公共public的set和get这种方法供外界来访问我们的私有属性。

|  |
| --- |
| class Student{    private int stuNo;    private String name;    private int age;    public void learn(){  System.out.println("学生在学习");  }    public void info(){  System.out.println("学号："+stuNo+" 姓名："+name+" 年龄："+age);  }  }  public class TestStudent1{    public static void main(String [] args){    Student s = new Student();  s.name = "亮哥";  s.age = 19;  s.stuNo = 1;  s.info();    }    } |



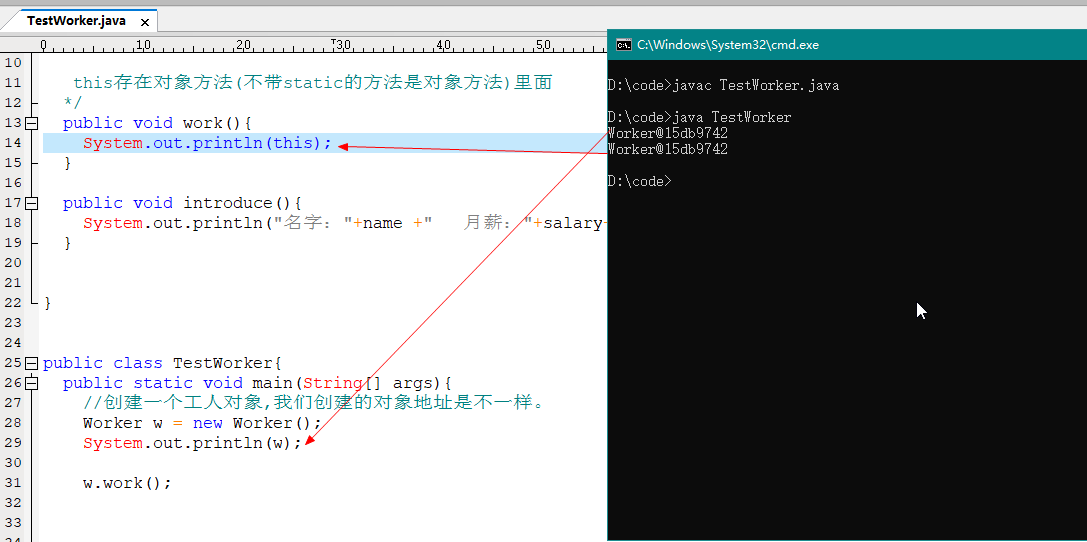
注意：Private的属性只有本类中的方法能访问。

属性的操作无非就是赋值和取值

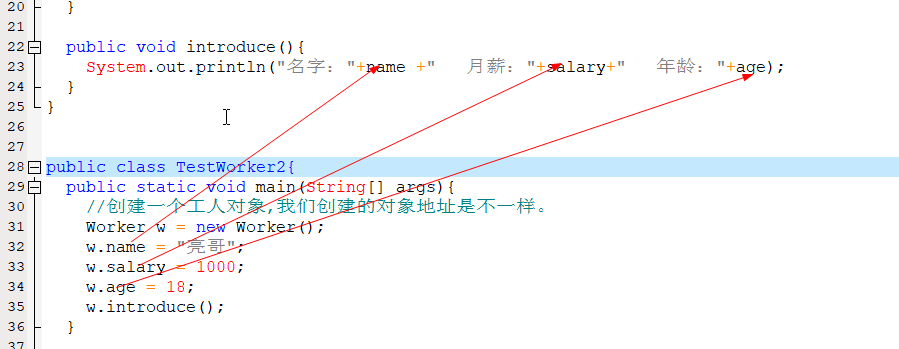
|  |
| --- |
| class Student{    private int stuNo;    private String name;    private int age;    /\*\*  \* 给stuNo赋值的方法  \*/  public void setStuNo(int myStuNo){  stuNo = myStuNo;  }  /\*\*  \* 给stuNo取值的方法  \*/  public int getStuNo(){  return stuNo;  }    public void setName(String myName){  name = myName;  }    public String getName(){  return name;  }    public void setAge(int myAge){  age = myAge;  }    public int getAge(){  return age;  }          public void learn(){  System.out.println("学生在学习");  }    public void info(){  System.out.println("学号："+stuNo+" 姓名："+name+" 年龄："+age);  }  }  public class TestStudent1{    public static void main(String [] args){    Student s = new Student();  s.setStuNo(1);  s.setName("亮哥");  s.setAge(18);  s.info();      }    } |

# This关键字

|  |
| --- |
| class Worker{    String name;    int salary;    int age;    /\*\*    this存在对象方法(不带static的方法是对象方法)里面  \*/  public void work(){  System.out.println(this);  }    public void introduce(){  System.out.println("名字："+name +" 月薪："+salary+" 年龄："+age);  }      }  public class TestWorker{  public static void main(String[] args){  //创建一个工人对象,我们创建的对象地址是不一样。  Worker w = new Worker();  System.out.println(w);  w.work();  }    } |



|  |
| --- |
| class Worker{    String name;    int salary;    int age;    /\*\*    this存在对象方法(不带static的方法是对象方法)里面    对象方法中隐藏了this关键字，this表示调用这个方法的那个对象    对象方法中this是可以省略的，效果是一样的  \*/  public void work(){  System.out.println(name);    }    public void introduce(){  System.out.println("名字："+name +" 月薪："+salary+" 年龄："+age);  }  }  public class TestWorker2{  public static void main(String[] args){  //创建一个工人对象,我们创建的对象地址是不一样。  Worker w = new Worker();  w.name = "亮哥";  w.salary = 1000;  w.age = 18;  w.introduce();  }    } |



this关键字的内存结构图：对象方法为什么是共享的。

栈 堆

main

0x98 new Worker()

0x99 new Worker ()

name = "亮哥";

salary = 1000;

age = 18;

introduce ()

Worker w

Worker w1

0x98

0x99

name = "明哥";

salary = 2000;

age = 16;

数据共享区

Worker.class

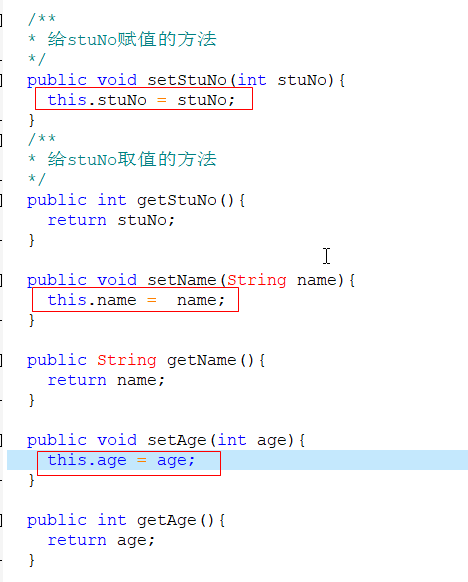
work()

introduce (){

System.out.println("名字："+this.name +" 月薪："+this.salary+" 年龄："+this.age);

}

This在封装中的应用：



# 构造器

用来创建对象的方法。

作用：创建对象，给对象赋初始值；

构造器本身是一个方法：

没有返回值，也没有void。

构造器的方法名必须和类名一致

在方法中定义要初始化的参数列表

## 默认构造器

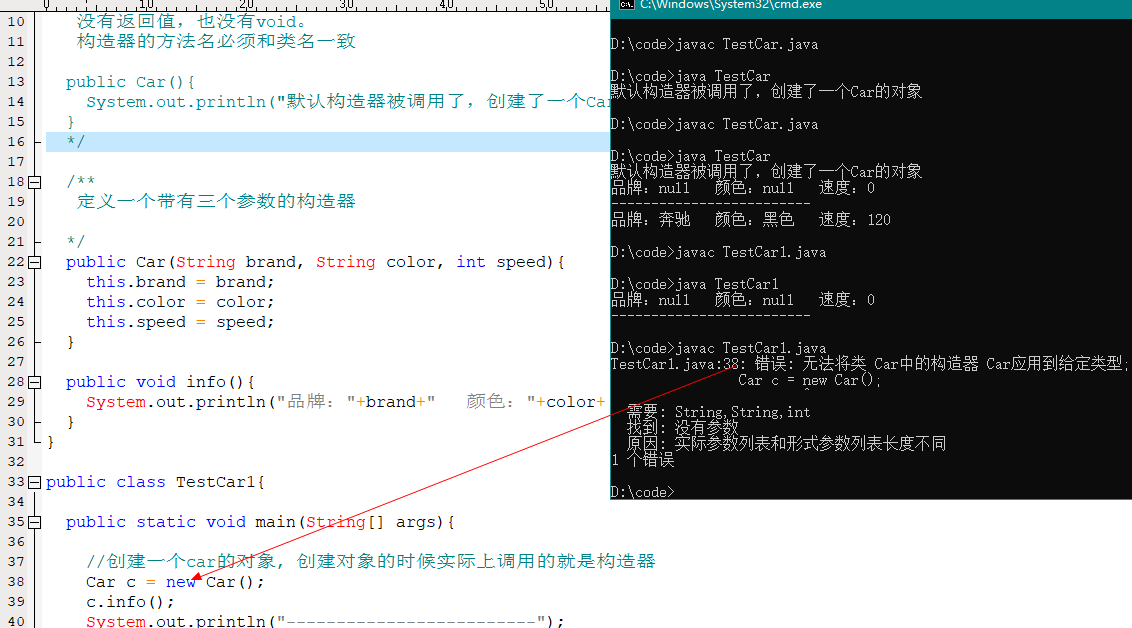
默认构造器是隐藏的

|  |
| --- |
| class Car{    String brand;    String color;    int speed;    /\*\*  没有返回值，也没有void。  构造器的方法名必须和类名一致  \*/  public Car(){  System.out.println("默认构造器被调用了，创建了一个Car的对象");  }    public void info(){  System.out.println("品牌："+brand+" 颜色："+color+ " 速度："+speed);  }  }  public class TestCar{    public static void main(String[] args){    //创建一个car的对象, 创建对象的时候实际上调用的就是构造器  Car c = new Car();    }  } |

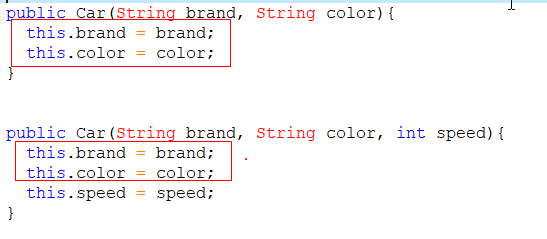
## 有参数的构造器

|  |
| --- |
| class Car{    String brand;    String color;    int speed;    /\*\*  没有返回值，也没有void。  构造器的方法名必须和类名一致  \*/  public Car(){  System.out.println("默认构造器被调用了，创建了一个Car的对象");  }      /\*\*  定义一个带有三个参数的构造器    \*/  public Car(String brand, String color, int speed){  this.brand = brand;  this.color = color;  this.speed = speed;  }    public void info(){  System.out.println("品牌："+brand+" 颜色："+color+ " 速度："+speed);  }  }  public class TestCar{    public static void main(String[] args){    //创建一个car的对象, 创建对象的时候实际上调用的就是构造器  Car c = new Car();  c.info();  System.out.println("-------------------------");    //使用有参数的构造器创建对象  Car c1 = new Car("奔驰","黑色",120);  c1.info();      }  } |

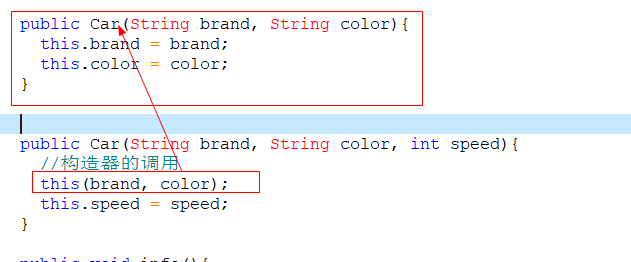
如果类存在有参的构造器的话，默认的构造器会被覆盖。如果还想要使用这个默认的构造器，就必须要显式的定义出来。



## 构造器之间的相互调用

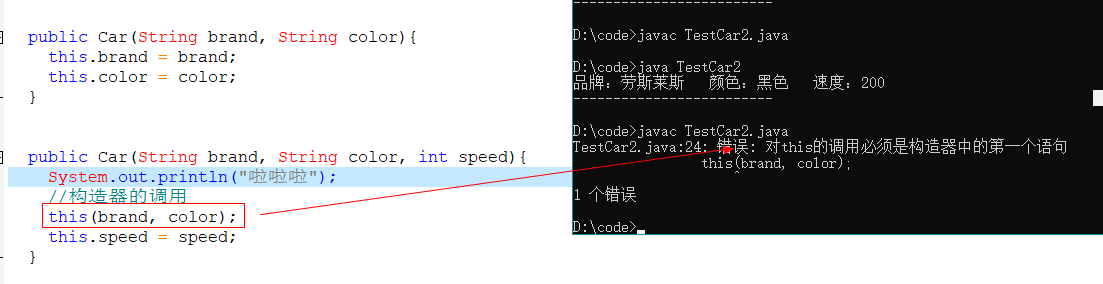


调用2个参数的构造器



|  |
| --- |
| class Car{    String brand;    String color;    int speed;      public Car(){    }      public Car(String brand, String color){  this.brand = brand;  this.color = color;  }    public Car(String brand, String color, int speed){  //构造器的调用  this(brand, color);  this.speed = speed;  }    public void info(){  System.out.println("品牌："+brand+" 颜色："+color+ " 速度："+speed);  }  }  public class TestCar2{    public static void main(String[] args){      Car c = new Car("劳斯莱斯", "黑色", 200);  c.info();  System.out.println("-------------------------");      }  } |

构造器调用构造器，被调用的构造器一定要放在第一行。



# Static关键字

## Static修饰在属性上

语法：static数据类型 变量名

使用static修饰的属性我们认为是类的属性，不带static修饰的属性我们认为是对象的属性。

栈 堆

name = “亮哥”

Age = 18

Addr = “火星”

0x98 New Person()

0x99 New Person()

Person p

Person p1

0x98

name = “发哥”

Age = 68

Addr = “地球”

0x99

数据共享区

Person.class

static int pcount =0;

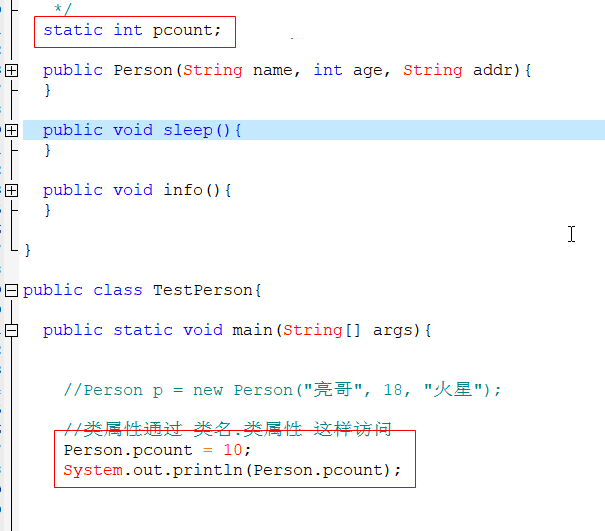
对象属性的生命周期

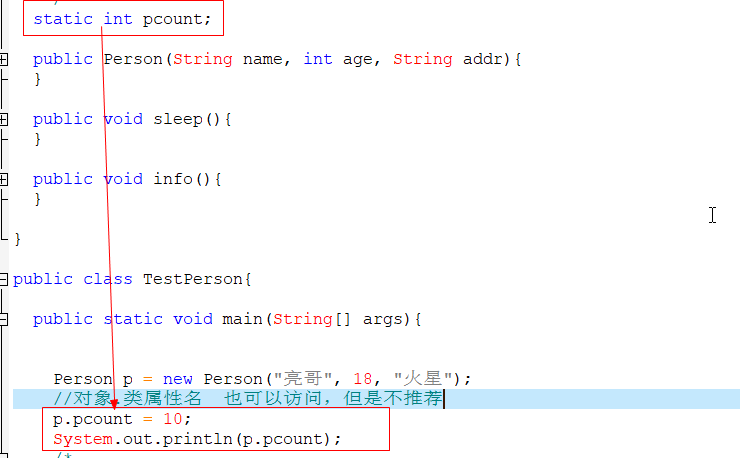
随着对象的创建二产生，对象如果被gc回收，属性自然也就不存在了

Static类属性生命周期

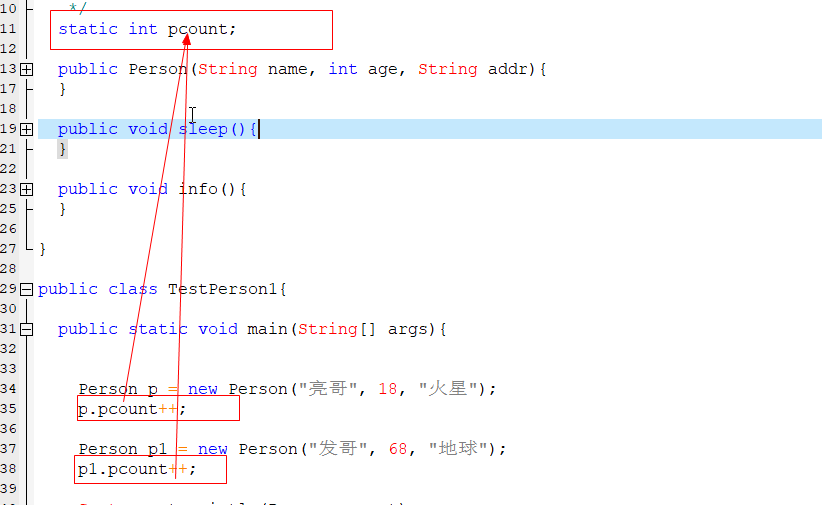
类文件被载入jvm的时候就会自动的初始化static的属性，jvm销毁后就会销毁类文件，static的属性也就被销毁了。

Static属性的使用。





类属性对每一个这个类所产生的对象都是共享的



## static关键字修饰在方法上

栈 堆

name = “亮哥”

Age = 18

Addr = “火星”

0x98 New Person()

0x99 New Person()

Person p

Person p1

0x98

name = “发哥”

Age = 68

Addr = “地球”

0x99

数据共享区

Person.class

Static int pcount;

Sleep(){

System.out.println(this.name+"在睡觉");

}

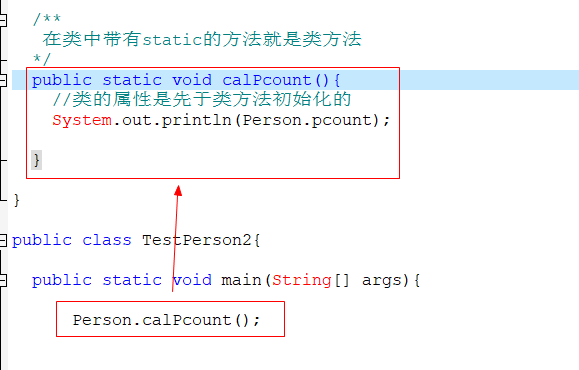
Static calPcount(){

System.out.println(Person.pcount);

}

对象方法是共享的，但是这个对象方法一定是创建了对象之后才能调用。对象方法的生命周期是在对象创建之后。

类方法是在static的属性初始化完毕初始化。类方法的调用也是通过类名.方法名



Static类方法只能访问static类属性。



对象方法可以访问static的类属性和类方法。

|  |
| --- |
| class Person{    String name;    int age;    String addr;  /\*\*  类属性  \*/  static int pcount;    public Person(String name, int age, String addr){  this.name = name;  this.age = age;  this.addr = addr;  }      /\*\*    在类中不带有static的方法是对象方法    \*/  public void sleep(){  System.out.println(name+"在睡觉");  System.out.println(pcount);  //调用对象方法  info();  //对象方法能够调用类方法  calPcount();  }    public void info(){  System.out.println("姓名："+name+ " 年龄："+age+ " 地址："+addr);  }    /\*\*  在类中带有static的方法就是类方法  \*/  public static void calPcount(){  //如果在同一个类中static的方法访问static的属性可以省略类名字。  System.out.println(pcount);    }    }  public class TestPerson4{    public static void main(String[] args){    Person p = new Person("aa", 18, "ggg");  p.sleep();          }  } |

# 单例模式

懒汉

|  |
| --- |
| class Singleton{    //定义一个当前类的属性  static Singleton s;      /\*\*  构造器的初始化  \*/  private Singleton(){    }    /\*\*  创建一个公有的方法返回一个Singleton这个类的对象的实例  \*/  public static Singleton getInstance(){  if(s == null){  s = new Singleton();  }  return s;  }      }  /\*\*  单例模式：一个类只能创建一个对象, 懒汉  \*/  public class TestSingleton{    public static void main(String[] args){  Singleton s = Singleton.getInstance();  System.out.println(s);    Singleton s1 = Singleton.getInstance();  System.out.println(s1);    Singleton s2 = Singleton.getInstance();  System.out.println(s2);    }    } |

饿汉

|  |
| --- |
| class Singleton{    //定义一个当前类的属性  static Singleton s = new Singleton();  /\*\*  构造器的初始化  \*/  private Singleton(){    }    /\*\*  创建一个公有的方法返回一个Singleton这个类的对象的实例  \*/  public static Singleton getInstance(){    return s;  }      }  /\*\*  单例模式：一个类只能创建一个对象, 饿汉  \*/  public class TestSingleton1{    public static void main(String[] args){  Singleton s = Singleton.getInstance();  System.out.println(s);    Singleton s1 = Singleton.getInstance();  System.out.println(s1);    Singleton s2 = Singleton.getInstance();  System.out.println(s2);    }    } |