# 第01天【OOP：面向对象编程\_第一天】

## 主要内容

1. 面向过程和面向对象
2. 类和对象
3. 成员变量和成员方法
4. 局部变量
5. 构造方法及其重载
6. 方法调用
7. this

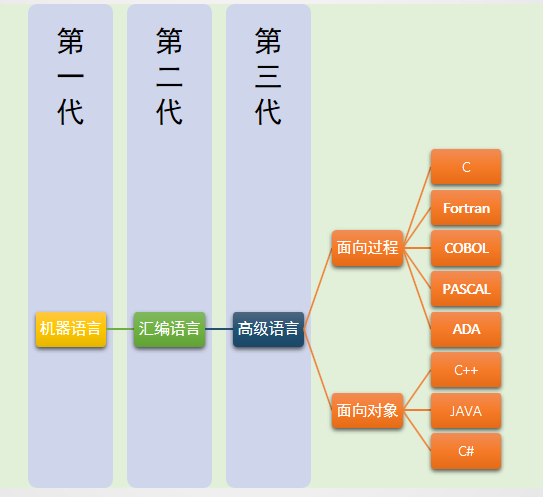
## 学习目标

|  |  |
| --- | --- |
| 知识点 | 要求 |
| 面向过程和面向对象 | 理解 |
| 类和对象 | 熟悉 |
| 成员变量 | 掌握 |
| 成员方法 | 掌握 |
| 局部变量 | 掌握 |
| 构造方法及其重载 | 掌握 |
| 对象数组 | 掌握 |
| 方法调用-基本数据类型形参 | 掌握 |
| 方法调用-引用数据类型形参 | 掌握 |
| this关键字 | 掌握 |

## 第一节 类和对象

### 1.1面向过程和面向对象

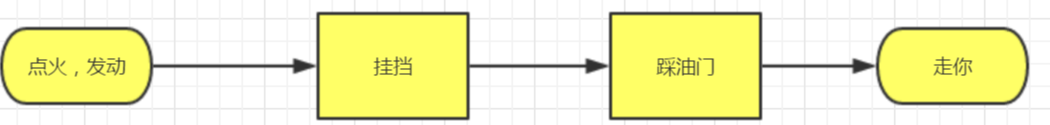
面向过程和面向对象都是对软件分析、设计和开发的一种思想,它指导着人们以不同的方式去分析、设计和开发软件。



早期先有面向过程思想，随着软件规模的扩大，问题复杂性的提高，面向过程的弊端越来越明显的显示出来，出现了面向对象思想并成为目前主流的方式。两者都贯穿于软件分析、设计和开发各个阶段，对应面向对象就分别称为面向对象分析（OOA）、面向对象设计（OOD）和面向对象编程（OOP）。C语言是一种典型的面向过程语言，Java是一种典型的面向对象语言。

**示例1：开车和造车**

面向过程思想思考问题时，我们首先思考“怎么按步骤实现？”并将步骤对应成方法，一步一步，最终完成。 这个适合简单任务，不需要过多协作的情况下。比如，如何开车？我们很容易就列出实现步骤：



面向过程适合简单、不需要协作的事务，重点关注如何执行。

但是当我们思考比较复杂的设计任务时，比如“如何造车？”，就会发现列出1234这样的步骤，是不可能的。那是因为，造车太复杂，需要很多协作才能完成。此时面向对象思想就应运而生了。

**面向对象(Oriented-Object)思想**更契合人的思维模式。我们首先思考的是“**怎么设计这个事物？”** 比如思考造车，我们就会先思考“车怎么设计？”，而不是“怎么按步骤造车的问题”。这就是思维方式的转变。

**比如，我们用面向对象思想思考“如何设计车”：**



**天然的，我们就会从“车由什么组成”开始思考。发现，车由如下对象组成：**

为了便于协作，我们找轮胎厂完成制造轮胎的步骤，发动机厂完成制造发动机的步骤；这样，发现大家可以同时进行车的制造，最终进行组装，大大提高了效率。但是，具体到轮胎厂的一个流水线操作，仍然是有步骤的，还是离不开执行者、离不开面向过程思维！

因此，**面向对象可以帮助我们从宏观上把握、从整体上分析整个系统。** 但是，具体到实现部分的微观操作（就是一个个方法），仍然需要面向过程的思路去处理。

我们千万不要把面向过程和面向对象对立起来。他们是相辅相成的。面向对象离不开面向过程！

**示例2：蛋炒饭和盖浇饭**

简单来说：用面向过程的方法写出来的程序是一份蛋炒饭，而用面向对象写出来的程序是一份盖浇饭。所谓盖浇饭，就是在米饭上面浇上一份盖菜，你喜欢什么菜，你就浇上什么菜。我觉得这个比喻还是比较贴切的。

蛋炒饭肯定是把米饭和鸡蛋混在一起炒匀。盖浇饭呢，则是把米饭和盖菜分别做好，你如果要一份红烧肉盖饭呢，就给你浇一份红烧肉；如果要一份青椒土豆盖浇饭，就给浇一份青椒土豆丝。

蛋炒饭的好处是入味均匀，吃起来香。如果恰巧你不爱吃鸡蛋，只爱吃青菜的话，那么唯一的办法就是全部倒掉，重新做一份青菜炒饭了。盖浇饭就没这么多麻烦，你只需要把上面的盖菜拨掉，更换一份盖菜就可以了。盖浇饭的缺点是入味不均，可能没有蛋炒饭那么香。

到底是蛋炒饭好还是盖浇饭好呢？其实这类问题都很难回答，非要比个上下高低的话，就必须设定一个场景，否则只能说是各有所长。那么从饭馆角度来讲的话，做盖浇饭显然比蛋炒饭更有优势，他可以组合出来任意多的组合，而且不会浪费。

盖浇饭的好处就是“菜”“饭”分离，从而提高了制作盖浇饭的灵活性。饭不满意就换饭，菜不满意换菜。用软件工程的专业术语就是“可维护性”比较好，“饭” 和“菜”的耦合度比较低。蛋炒饭将“蛋”“饭”搅和在一起，想换“蛋”“饭”中任何一种都很困难，耦合度很高，以至于“可维护性”比较差。软件工程追求的目标之一就是可维护性。面向对象的好处之一就是显著的改善了软件系统的可维护性。

**示例3：如何统一中国（一项极其复杂的系统工程）**

|  |  |
| --- | --- |
|  |  |
| 蒋介石：面向过程的杰出代表 | 毛泽东：面向对象的杰出代表 |
| 1. 抢占战败日本物资，收编伪军，拉美国金援。 2. 和共产党谈判，拖延时间，争取准备时间 3. 抢占城市等战略要点 4. 开打！ 5. 赢！哦耶！ | 1. 共产党 2. 农民（贫农 中农 富农） 3. 工人 4. 国民党（进步派 中间派 反动派） 5. 各民主派别 |
| 着眼于一城一池的得失；  有一种失败叫占领 | 用统一战线战略处理这些对象的关系；  有一种胜利叫撤退 |

**·面向对象和面向过程思想的总结**

* 都是解决问题的思维方式，都是代码组织的方式。
* **面向过程是一种“执行者思维”**，解决简单问题可以使用面向过程。
* **面向对象是一种“设计者思维”**，解决复杂、需要协作的问题可以使用面向对象。
* 面向对象离不开面向过程：
  + 宏观上：通过面向对象进行整体设计
  + 微观上：执行和处理数据，仍然是面向过程。。

|  |  |  |
| --- | --- | --- |
|  | **面向过程** | **面向对象** |
| 区别 | **事物比较简单，可以用线性的思维去解决** | **事物比较复杂，使用简单的线性思维无法解决** |
| 共同点 | 面向过程和面向对象都是解决实际问题的一种思维方式 | |
| 二者相辅相成，并不是对立的。解决复杂问题，通过面向对象方式便于我们从宏观上把握事物之间复杂的关系、方便我们分析整个系统；具体到微观操作，仍然使用面向过程方式来处理 | |





### 1.2理解类和对象

我们人认识世界，其实就是面向对象的（此对象可不是男女谈对象的彼对象呀） 。比如现在让大家认识一下“天使”这个新事物，天使大家没见过吧，怎么样认识呢？最好的办法就是，给你们面前摆4个天使，带翅膀的美女，让大家看，看完以后，即使我不说，大家下一次是不是就都认识天使了。

|  |
| --- |
|  |

但是，看完10个天使后，我们总要总结一下，什么样的东东才算天使？天使是无数的，总有没见过的！所以必须总结抽象，便于认识未知事物！**总结的过程就是抽象的过程。**小时候，我们学自然数时怎么定义的？像1，2，3，4…这样的数就叫做自然数。 通过抽象，我们发现天使有这样一下特征：

1. 带翅膀（带翅膀不一定是天使，还可能是鸟人）
2. 女孩（天使掉下来脸着地，也是天使！）
3. 善良
4. 头上有光环

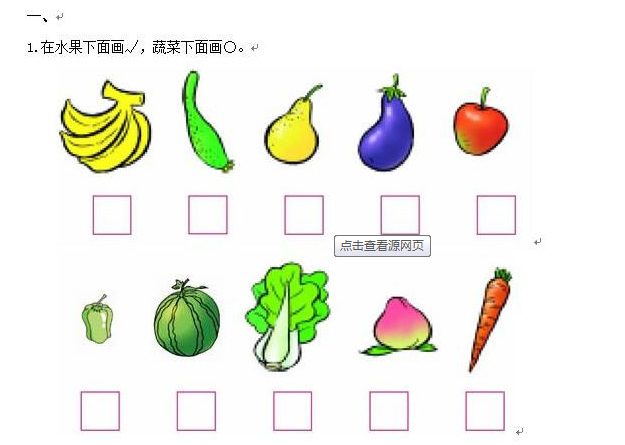
那么通过这4个具体的天使，我们进行抽象，抽象出了天使的特征，我们也可以归纳一个天使类。 通过这个过程，类就是对象的抽象。

**类可以看做是一个模版，或者图纸，系统根据类的定义来造出对象。我们要造一个汽车，怎么样造？类就是这个图纸，规定了汽车的详细信息，然后根据图纸将汽车造出来。**

**类：我们叫做class。 对象：我们叫做Object,instance(实例)。以后我们说某个类的对象，某个类的实例。是一样的意思。**

**示例1：水果、梨类和对象**

水果就是类，具体的香蕉、梨、苹果是对象。



**示例2：月饼模具和月饼**

月饼模具是类，使用月饼模具制作的一个个月饼就是对象



|  |
| --- |
| **总结**   * 类可以看成一类对象的模板，对象可以看成该类的一个具体实例。 * 类是用于描述同一类型的对象的一个抽象概念，类中定义了这一类对象所应具有的共同的属性、方法。 |

### 本节作业

1. 理解面向过程和面向对象的异同
2. 理解类和对象的关系

## 第二节 类和对象

### 2.1 定义类和创建对象

做了关于对象的很多介绍，终于进入代码编写阶段。本节中重点介绍类和对象的基本定义，属性和方法的基本使用方式。

#### 属性（field 成员变量）

**属性用于定义该类或该类对象包含的数据或者说静态特征**。属性作用范围是整个类体。

在定义成员变量时可以对其初始化，如果不对其初始化，**Java使用默认的值对其初始化**。

|  |  |
| --- | --- |
| **数据类型** | **默认值** |
| 整型 | 0 |
| 浮点型 | 0.0 |
| 字符型 | '\u0000' |
| 布尔型 | false |
| 所有引用类型 | null |

**属性定义格式：**

|  |
| --- |
| [修饰符] 属性类型 属性名 = [默认值] ; |

#### 方法

**方法用于定义该类或该类实例的行为特征和功能实现。**方法是类和对象行为特征的抽象。方法很类似于面向过程中的函数。面向过程中，函数是最基本单位，整个程序由一个个函数调用组成。面向对象中，整个程序的基本单位是类，方法是从属于类和对象的。

**方法定义格式：**

|  |
| --- |
| [修饰符] 方法返回值类型 方法名(形参列表) {  // n条语句  } |

void代表没有返回值；方法的作用：重用代码，封装功能，便于修改

#### 【示例1】定义类，模拟人的行为：吃饭工作和休息

1. 面向对象分析OOA

从张三、李四、王五中提取出共同内容，提取一个类，人Person，包含共同的内容

静态的特征：姓名、年龄、性别、住址

动态的行为：eat、rest、work

1. 面向对象设计OOD Design

类：Person

对象：zhangsan、lisi

让对象完成功能

1. 面向对象编程OOP Programming

|  |
| --- |
| **package** com.bjsxt.oopDemo1; */\*  \* 面向过程 :先做什么,后做什么,强调顺序 C  \* 面向对象 :用谁对谁做点什么,有角色和功能 JAVA  \*  \* 开车:  \* 面向过程:点火,踩离合,拉手刹,换挡,松离合,加油 (强调过程,微观层的具体操作)  \* 面向对象:有一台车,有个司机, 让司机开车 (强调角色,宏观层的整体思维)  \*  \* 造车:  \* 面向过程:炼铁 >>> 铸造 >>> 设计>>> 加工 >>>> 造发动机 >>> 10000个零件 造第一个零件 造第二个零件 … …  \* 面向对象:获得轮胎, 获得发动机, 获得变速箱,获得车身 组装 >>> 完成  \*  \* 简单的问题,底层问题,面向过程  \* 复杂的问题,宏观问题,面向对象  \*  \* 类和对象  \* 类 对象  \* 水果 香蕉 苹果 橘子 葡萄 ... ...  \* 类 对一些性质相同,功能相似的实物的总结和归纳,类别 英雄 化妆品  \* 对象 某一类事物中,具体的,实际存在在,可以使用的实例,一个实际存在的例子 蒙多 提莫 盲僧 迪奥香水 香奈儿口红  \*  \* 在实际生活中  \* 先有对象 再有类  \* 众多的对象 >>> 归纳,总结,抽象 >>> 类别  \*  \* 程序中  \* 先有类 再有对象  \* 先创建类 >>> 实例化 >>> 众多的对象  \*  \* 类(模板) 组成部分  \* 性质 特征 属性 >>> 变量 姓名 String name; int age;  \* 行为 能力 功能 >>> 方法 吃饭 public void eat() 睡觉 public void sleep()  \* 类生产对象  \*  \* \*/* **public class** Person {  */\*\*  \* 属性:  \* 姓名、年龄、性别、住址  \* 直接定义在类中的变量,叫做成员变量  \* 是类的成员,用于代表该类具有什么属性  \*/* String **name**;  **int age**;  String **gender**;  String **address**;    */\*\*  \* 行为  \* eat、rest、work  \* 方法 成员方法  \* 类的成员,用于代表该类具有什么功能  \*/* **public void** eat(String food){  System.***out***.println(**name**+**"吃了"**+food);  }   **public void** rest(){  System.***out***.println(**age**+**"岁的"**+**name**+**"正在睡觉"**);  }   **public void** work(){  System.***out***.println(**name**+**"正在敲代码"**);  }     } |

#### 【示例2】模拟实现具体人的特征和行为

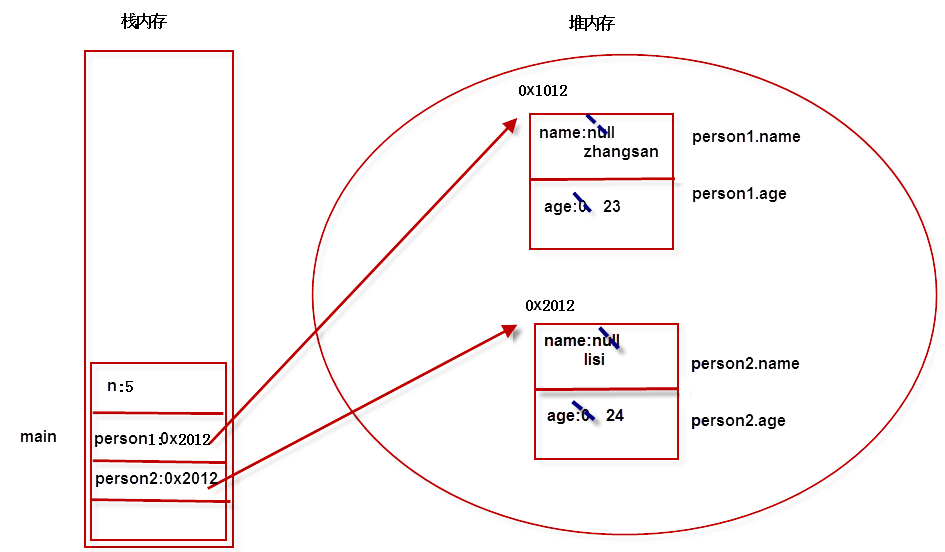
|  |
| --- |
| **package** com.bjsxt.oopDemo1;  **public class** Test1 {  **public static void** main(String[] args) {  *//实例化对象* Person p=**new** Person();  *//给对象属性赋值* p.**age**=10;  p.**gender**=**"boy"**;  p.**name**=**"张思锐"**;  p.**address**=**"象牙山村小河边"**;  *//调用对象方法,完成功能* p.eat(**"臭豆腐"**);  p.rest();  p.work();   *// 再次创建Person类对象* Person p2=**new** Person();  p2.**name**=**"日向广坤"**;  p2.**age**=60;  p2.**gender**=**"男"**;  p2.**address**=**"象牙山山脚下"**;   p2.eat(**"酸奶"**);  p2.rest();  p2.work();   System.***out***.println(**"一年以后"**);  p2.**age**=61;  p2.**address**=**"象牙村山货收购点"**;  p2.eat(**"土豆"**);  p2.rest();  p2.work();  System.***out***.println(p2.**name**);  System.***out***.println(p2.**age**);  System.***out***.println(p2.**gender**);  System.***out***.println(p2.**address**);  */\*\*  \* 对象可以创建多个  \* 属性都是每个对象独立的  \* 属性是可以修改的  \* 方法是调用多次的  \*/* } } |

总结1：如何创建对象  
 Scanner input = new Scanner(System.in);  
 Person person1 = new Person();  
 new 后面是构造方法，具体含义稍后讲解

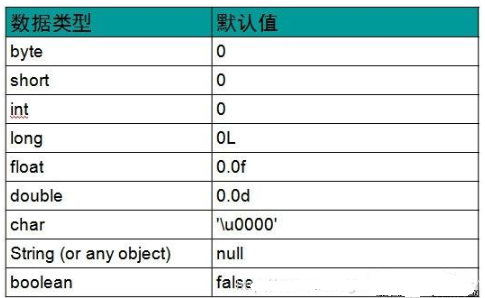
总结2：如何操作属性：对象名.属性名  
 person1.name="zhangsan";  
 person1.age = 23;

总结3：如何调用方法: 对象名.方法名(实参列表)  
 person1.eat();  
 person1.work();  
 person1.rest("宿舍");

总结4：内存分配图



1. 栈内存
2. stack memory
3. 存放局部变量、数组名，没有默认值
4. 每调用一个方法会开辟一个栈帧，方法执行完立刻回收该栈帧，后进先出
5. 堆内存
6. heap memory
7. 存放真正的数组元素/对象内容，有默认值
8. 方法执行完毕后不会立刻释放资源，由垃圾回收器负责回收；程序员不用操心 但时间不确定
9. 数组元素/对象成员变量的默认值



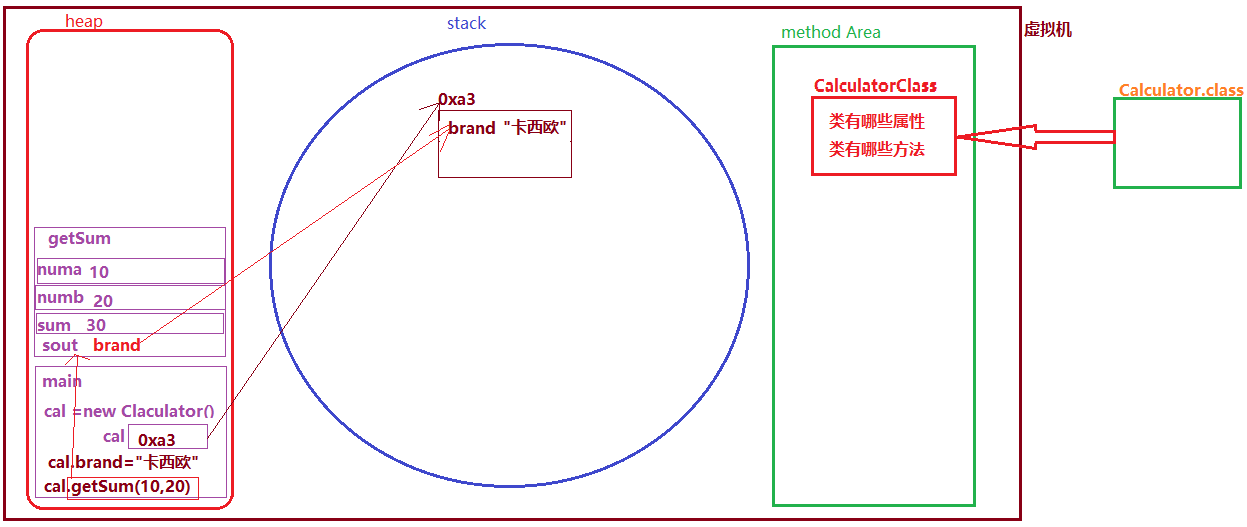
关于虚拟机内存模型的更多细节我们在面向对象一章会陆续讲解，并在面向对象一章内容结束后进行系统总结。

### 2.2 局部变量和成员变量

类中定义的变量是成员变量，而方法中定义的变量，包括方法的参数，代码块中定义的变量被称为局部变量。两个的区别主要表现在以下几方面：

|  |  |  |
| --- | --- | --- |
|  | **成员变量** | **局部变量** |
| **代码中位置不同** | 类中定义的变量 | 方法或代码块中定义的变量 |
| **内存中位置不同** | 堆内存 | 栈内存 |
| **是否有默认值** | 有 | 没有 |
| **代码作用范围（空间）** | 当前类的方法 | 当前一个方法或代码块 |
| **作用时间不同** | 当前对象从创建到销毁 | 定义变量到所属方法或代码块执行完毕 |

代码执行的内存分配图如下：



### 本节作业

1. 定义Animal类并创建多个Animal对象，模拟动物的行为
2. 定义一个计算机类并模拟其属性和行为

属性 品牌 型号 内存大小 CPU型号 GPU型号

行为:开机 运行游戏 运行JAVA代码 关机 看视频 播放音乐

创建电脑类,实例化该类的两个对象,给属性赋值,调用方法,完成功能

|  |
| --- |
| **package** com.bjsxt.oopDemo1;  **public class** Computer {  String **brand**;  String **type**;  String **cpu**;  String **gpu**;  **int ram**;  *//开机 运行游戏 运行JAVA代码 关机 看视频 播放音乐* **public void** start(){  System.***out***.println(**"一台"**+**ram**+**"G内存,显卡为"**+**gpu**+**":CPU为:"**+**cpu**+**"的"**+**brand**+**type**+**"正在开机..."**);  }   **public void** playVideo(String video){  System.***out***.println(**"一台"**+**ram**+**"G内存,显卡为"**+**gpu**+**":CPU为:"**+**cpu**+**"的"**+**brand**+**type**+**"正在播放:"**+video);  }   **public static void** main(String[] args) {  Computer com =**new** Computer();  com.**brand**=**"华硕"**;  com.**type**=**"败家之眼"**;  com.**cpu**=**"i9"**;  com.**gpu**=**"2080Ti"**;  com.**ram**=128;  com.start();;  com.playVideo(**"许老师.avi"**);   Computer com2 =**new** Computer();  com2.**brand**=**"戴尔"**;  com2.**type**=**"外星人"**;  com2.**cpu**=**"i9"**;  com2.**gpu**=**"2080"**;  com2.**ram**=64;  com2.start();;  com2.playVideo(**"佳奇老师.avi"**);   }   } |

## 第三节 构造方法及其重载

### 3.1 构造方法

对于一个类来说，**一般有三种常见的成员：属性field、方法method、构造器constructor**。这三种成员都可以定义零个或多个。

构造方法(constructor)也叫构造器，用于对象的初始化。构造器是一个创建对象时被自动调用的特殊方法，目的是对象的初始化。构造器的名称应与类的名称一致。**Java通过new关键字来调用构造器，从而返回该类的实例，**是一种特殊的方法。

**声明格式：**

|  |
| --- |
| [修饰符] 类名(形参列表){  //n条语句  } |

**构造方法的4个要点：**

* 构造方法的方法名必须和类名一致！
* 构造方法通过new关键字调用！！
* 构造方法虽然有返回值，但是不能定义返回值类型(返回值的类型肯定是本类)，不能在构造器里使用return返回某个值。
* 如果我们没有定义构造方法，则编译器会自动定义一个无参的构造方法。如果已定义则编译器不会自动添加！

构造方法也是方法，只不过有特殊的作用而已。与普通方法一样，构造方法也可以重载。

#### 【示例3】定义Dog类并模拟其操作

|  |
| --- |
| **package** demo4;  **public class** Dog {  *// 属性* String **type**;  String **color**;  String **gender**;  **int month**;  *// 行为 犬吠* **public void** bark(){  System.***out***.println(**"一条"**+**color**+**"颜色,"**+**month**+**"个月的"**+**gender**+**type**+**"叫了一声汪... ..."**);  }   **public void** eat(String food){  System.***out***.println(**"一条"**+**color**+**"颜色,"**+**month**+**"个月的"**+**gender**+**type**+**"正在吃"**+food);  }   **public void** play(){  System.***out***.println(**"一条"**+**color**+**"颜色,"**+**month**+**"个月的"**+**gender**+**type**+**"追着球跑... ... "**);  }   */\*  \* 一个类三大成员  \* 1成员变量 用于模拟属性  \* 2成员方法 用于模拟行为  \* 3构造方法/构造器 帮助我们初始化对象  \* 一般方法的组成部分 方法头(访问修饰符,返回值类型,方法名,参数列表,异常列表) 方法体  \* 构造方法也叫作构造器,不是传统意义上的方法  \* 构造方法帮助我们初始化对象属性的  \* 构造方法语法 方法头(访问修饰符 方法名必须和类名同名,参数列表,异常列表) 方法体  \* 构造方法没有返回值也没有void关键字  \* 构造方法的方法名必须和类名相同  \* 构造方法需要用new关键字调用  \* 当我们不定义构造方法,JAVA会默认给我们提供一个参数缺省的构造方法  \* 当我们自己定义了构造方法,JAVA提供的构造方法会被覆盖  \*  \* 调用构造方法时都发生了哪些事情  \* 1JVM在内存上开辟一段独立的空间,用于存储对象信息  \* 2给成员变量赋初始值  \* 3执行构造方法中的代码  \* \*/* **public** Dog(String t,String g,String c,**int** m){  **type**=t;  **gender**=g;  **color**=c;  **month**=m;  System.***out***.println(**"Dog 类的构造方法执行了"**);  }   }  主方法测试代码  **public static void** main(String[] args) {  *// 实例化对象* Dog dog =**new** Dog(**"哈士奇"**,**"公"**,**"黑白"**,5);  *// 给对象属性赋值  /\* dog.type="哈士奇";  dog.gender="公";  dog.month=12;  dog.color="黑白";\*/   // 调用对象方法* dog.bark();  dog.eat(**"狗粮"**);  dog.play();    *// 当我们获得一个Dog类对象时,要求month属性值默认为2* Dog dog2=**new** Dog(**"泰迪"**,**"公"**,**"棕"**,3);  dog2.play();  dog2.eat(**"骨头"**);  dog2.bark(); } |

构造方法的重载

|  |
| --- |
| **public** Dog(String type,String gender,String color,**int** month){  */\*在方法中 出现了和成员变量同名的局部变量  \* 在当前方法中,使用变量时,会优先使用方法内部的局部变量  \* 如果当前方法中成员变量和局部变量的名称不同, this.可以省略不写  \* \*/* **this**.**type**=type;  **this**.**gender**=gender;  **this**.**color**=color;  **this**.**month**=month;  System.***out***.println(**"Dog 类的构造方法执行了"**); } */\* \* 构造方法是可以进行重载 \* 重载:在一个类中,多个同名方法的定义 在一个类中,多个方法使用同一个名字 \* 好处:功能相似或者相同的方法,使用同一个名字,方便调用 记忆 可以减少方法的大量命名 \* 方法名必须相同,参数列表必须不同 \* \*/* **public** Dog(String type){  **this**.**type**=type; } **public** Dog(**int** month){  **this**.**month**=month; } */\* 一般在定义类时,我们都准备两个构造方法 \* 1包含全部属性的构造方法 \* 2空参构造方法 \* \*/* **public** Dog(){   } |

面试题:构造方法都有哪些调用方式?分别举例

|  |
| --- |
| 1通过new关键字调用 帮助我们初始化对象  2 构造方法可以调用当前类的其他构造方法 通过this()的形式  3 在继承中, 子类构造方法一定会调用父构造方法, 通过super()形式 |

|  |
| --- |
| **新手雷区**  对象的创建完全是由构造方法实现的吗？  不完全是。构造方法是创建Java对象重要途径，通过new关键字调用构造器时，构造器也确实返回了该类对象，但这个对象并不是完全由构造器负责创建的。创建一个对象分为如下四步：  1. 分配对象空间，对象成员变量初始化为0或空  2. 执行属性值的显示初始化（变量初始化为0或空）  3. 执行构造方法  4. 返回对象的地址给相关的变量 |

|  |
| --- |
| **新手雷区**  如果方法构造中形参名与属性名相同时，需要使用this关键字区分属性与形参。  this.id 表示属性id；id表示形参id |

### 3.2 this

**对象创建的过程和this的本质**

构造方法是创建Java对象的重要途径，通过new关键字调用构造器时，构造器也确实返回该类的对象，但这个对象并不是完全由构造器负责创建。创建一个对象分为如下四步：

1. 分配对象空间，并将对象成员变量初始化为0或空

2. 执行属性值的显示初始化

3. 执行构造方法

4. 返回对象的地址给相关的变量

this的本质就是“创建好的对象的地址”！ 由于在构造方法调用前，对象已经创建。因此，在构造方法中也可以使用this代表“当前对象” 。

**this最常的用法：**

* 调用成员变量：如果成员变量和局部变量同名，this必须书写，用来区分两者；如果没有同名的局部变量，this可以不写
* 调用成员方法：这种情况下，this可以省略
* 调用构造方法：使用this关键字调用重载的构造方法，避免相同的初始化代码。但只能在构造方法中用，并且必须位于构造方法的第一句。
* this不能用于static方法中。

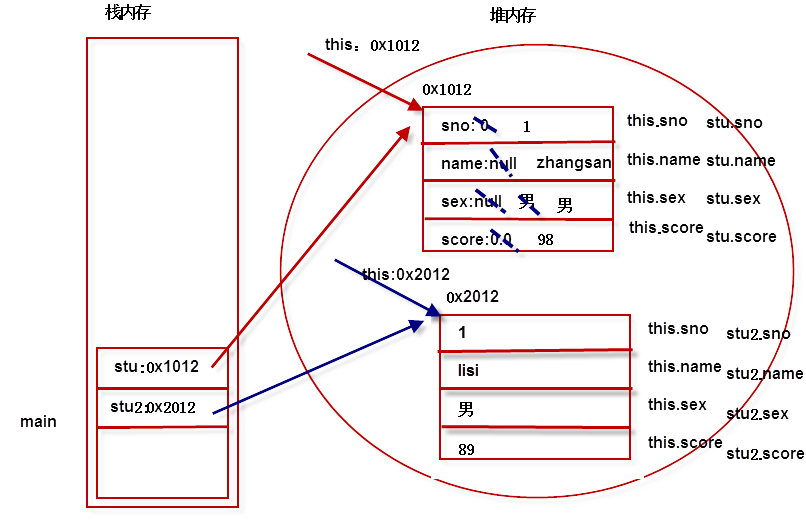
#### 【示例8】this关键字的使用

使用this关键字区分成员变量和局部变量

|  |
| --- |
| **public** Dog(String type,String gender,String color,**int** month){  */\*在方法中 出现了和成员变量同名的局部变量  \* 在当前方法中,使用变量时,会优先使用方法内部的局部变量  \* 如果当前方法中成员变量和局部变量的名称不同, this.可以省略不写  \* \*/* **this**.**type**=type;  **this**.**gender**=gender;  **this**.**color**=color;  **this**.**month**=month;  System.***out***.println(**"Dog 类的构造方法执行了"**); } |

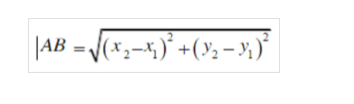
构造方法可以调用当前类的其他构造方法 通过this()的形式

|  |
| --- |
| */\* \* 构造方法之间是可以调用的 但是不能使用方法名,要是使用this \* 调用其他构造方法的代码必须是头一行代码 \* \*/* **public** Dog(String type,String gender,String color,**int** month){  **this**(type,gender);  **this**.**color**=color;  **this**.**month**=month;  System.***out***.println(**"Dog 类的构造方法执行了"**); }  *//定义一个构造方法,专门用于初始化 type属性和 gender属性* **public** Dog(String type,String gender){  **this**.**type**=type;  **this**.**gender**=gender; }   **public** Dog(){ } |



### 本节作业

1. 定义一个“点”（Point）类用来表示二维空间中的点（有二个坐标）。要求如下：
   1. 可以生成具有特定坐标的点对象。
   2. 提供可以设置二个坐标的方法。
   3. 提供可以计算该“点”距另外点距离的方法。



1. 使用对象数组存储多个对象并输出内容

## 第四节 方法调用

java中的数据类型

1基本数据类型 8个

2引用类型 类和对象 数组 接口 抽象类 枚举 … …

方法调是Java开发中的基本操作。理解方法调用的内存分配过程，实参形参的传递过程非常必要。方法参数分为基本数据类型和引用数据类型两种，传递参数有着实质的区别。

实参给形参传递数据有两种方式

1按值传递 参数为基本数据类型

2按引(地址)用传递 参数为引用类型

### 4.1 对象数组

**对象作为方法参数:**

假设某高校浴室坏了一间,学校安排学生分时段洗澡:上午男孩洗,下午女孩洗.浴室门口有个老头,专门检查学生是否在规定时间内洗澡;

学生类

属性:姓名,性别,年龄

|  |
| --- |
| **public class** Student {  String **stuName**;  String **stugender**;  **int stuAge**;   **public** Student(String stuName, String stugender, **int** stuAge) {  **this**.**stuName** = stuName;  **this**.**stugender** = stugender;  **this**.**stuAge** = stuAge;  }   **public** Student() {  } } |

老头类

属性:

功能:通知学生洗澡时间段的功能

|  |
| --- |
| **package** com.bjsxt.oopDemo4;  **public class** OldMan {  */\*  \* 对象作为方法参数  \* 简化参数列表  \*/* **public void** checkStudent(Student stu){  **if**(**"男"**.equals(stu.**stugender**)){  System.***out***.println(stu.**stuName**+**"是男孩,上午洗"**);  }**else**{  System.***out***.println(stu.**stuName**+**"是女孩,下午洗"**);  }  }   **public** OldMan() {  } } |

测试

|  |
| --- |
| **public class** Test {  **public static void** main(String[] args) {  Student stu1=**new** Student(**"日向广坤"**,**"男"**,20);  Student stu2=**new** Student(**"千手老七"**,**"男"**,22);  Student stu3=**new** Student(**"猿飞大脚"**,**"女"**,21);   OldMan oldMan=**new** OldMan();   oldMan.checkStudent(stu1);  oldMan.checkStudent(stu2);  oldMan.checkStudent(stu3);    } } |

已经学过数组了，但是数组元素都是基本数据类型或者String类型。学习了类和对象之后，可以定义数组的元素类型是更加复杂的引用引用数据类型，每个元素可以是一个具体的对象，称为对象数组。

#### 【示例4】使用对象数组存储多个Student信息

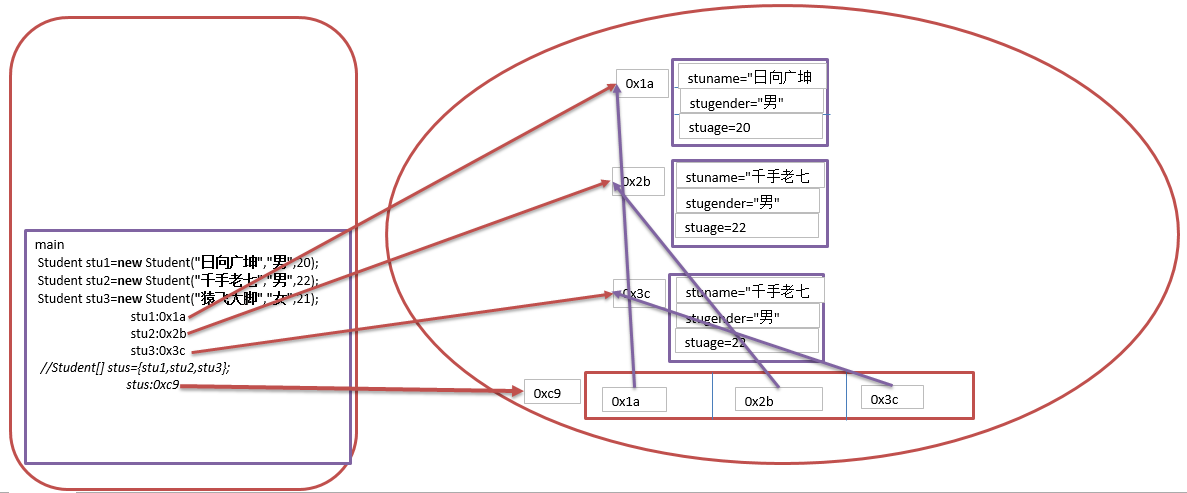
oldMan代码

|  |
| --- |
| **package** com.bjsxt.oopDemo4;  **public class** OldMan {  */\*  \* 检查多个学生的方法  \* 对象数组作为方法参数  \* \*/* **public void** checkStudents(Student... stus){  **for** (**int** i = 0; i <stus.**length** ; i++) {  checkStudent(stus[i]);  }  }   */\*  \* 检查单个学生的方法  \* 对象作为方法参数  \* 简化参数列表  \*/* **public void** checkStudent(Student stu){  **if**(**"男"**.equals(stu.**stugender**)){  System.***out***.println(stu.**stuName**+**"是男孩,上午洗"**);  }**else**{  System.***out***.println(stu.**stuName**+**"是女孩,下午洗"**);  }  }   **public** OldMan() {  } } |

测试代码

|  |
| --- |
| **package** com.bjsxt.oopDemo4;  **public class** Test {  **public static void** main(String[] args) {  Student stu1=**new** Student(**"日向广坤"**,**"男"**,20);  Student stu2=**new** Student(**"千手老七"**,**"男"**,22);  Student stu3=**new** Student(**"猿飞大脚"**,**"女"**,21);   *//Student[] stus={stu1,stu2,stu3};* Student[] stus =**new** Student[3];  stus[0]=stu1;  stus[1]=stu2;  stus[2]=stu3;   OldMan oldMan=**new** OldMan();  oldMan.checkStudents(stus);  oldMan.checkStudents(stu1,stu2,stu3);  } } |

内存分配图入下图所示。

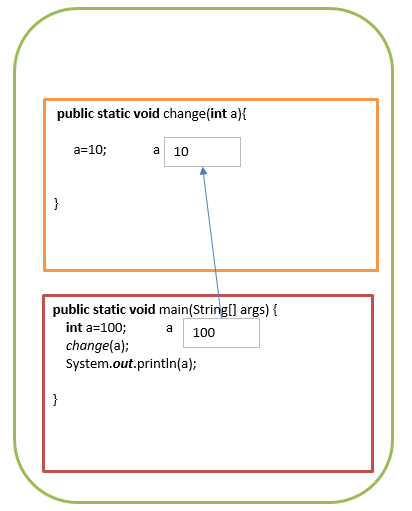


### 4.2 基本数据类型的方法调用

#### 【示例5】基本数据类型的方法参数

|  |
| --- |
| **package** com.bjsxt.oopDemo4;  **public class** Test2 {  **public static void** main(String[] args) {  **int** a=100;  *change*(a);  System.***out***.println(a);   }  **public static void** change(**int** a){a=10;  } } |

内存分配过程如图所示

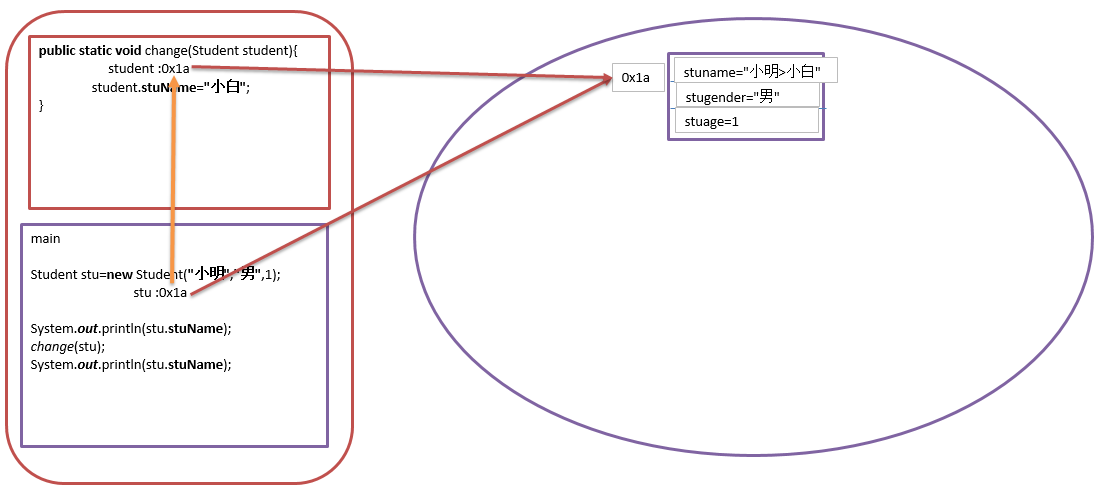


### 4.3 引用数据类型的方法调用

#### 【示例6】引用数据类型的方法参数

|  |
| --- |
| **public class** Test3 {  **public static void** main(String[] args) {  Student stu=**new** Student(**"小明"**,**"男"**,1);  System.***out***.println(stu.**stuName**);  *change*(stu);  System.***out***.println(stu.**stuName**);  }   **public static void** change(Student student){  student.**stuName**=**"小白"**;  } } |

内存分配过程如图所示。



|  |
| --- |
| **新手雷区**  基本数据类型的参数是值传递，引用数据类型的参数传递是引用（地址），本质上也是值传递。 |

### 本节作业

1. 练习基本数据类型的参数传递
2. 练习引用数据类型的参数传递
3. this关键字的作用和用法
4. 练习this关键字的使用