

# 面向对象

Author：zhangzhang

Version：1.0.0

- 一、引言
  - 1.1 什么是程序
  - 1.2 现实世界的组成
- 二、什么是对象
  - 2.1 什么是对象
  - 2.2 现实中的对象
  - 2.3 程序中的对象
- 三、什么是类【重点】
  - 3.1 什么是类
  - 3.2 类的抽取
  - 3.3 类的定义
  - 3.4 对象的创建
  - 3.5 类与对象的关系
- 四、类的组成【重点】
  - 4.1 实例变量
  - 4.2 实例变量与局部变量的区别
  - 4.3 实例方法
  - 4.4 课堂案例
- 五、方法重载
  - 5.1 方法重载概念
  - 5.2 方法重载
  - 5.3 代码调错
- 六、构造方法【重点】
  - 6.1 构造方法
  - 6.2 对象创建过程
  - 6.3 对象的内存分配
  - 6.4 构造方法重载
  - 6.5 默认构造方法
  - 6.6 构造方法为属性赋值
- 七、this关键字
  - 7.1 this关键字
  - 7.2 this关键字的两种用法
    - 7.2.1 第一种用法
    - 7.2.2 第二种用法

## 一、引言

### 1.1 什么是程序

程序是为了模拟现实世界，解决现实问题而使用计算机语言编写的指令集合。



即时聊天



路况、导航



便捷支付



美颜、修图

### 1.2 现实世界的组成

- 世界是由什么组成的？
- 有人说：“世界是由无数原子组成的”。

- 有人说：“世界是由无数事物组成的”。
- 有人说：“世界是由无数物体组成的”。
- 有人说：“世界是由一切有生命的和一切没有生命的组成的”。
- 有人说：“你、我、他、大家组成的”。

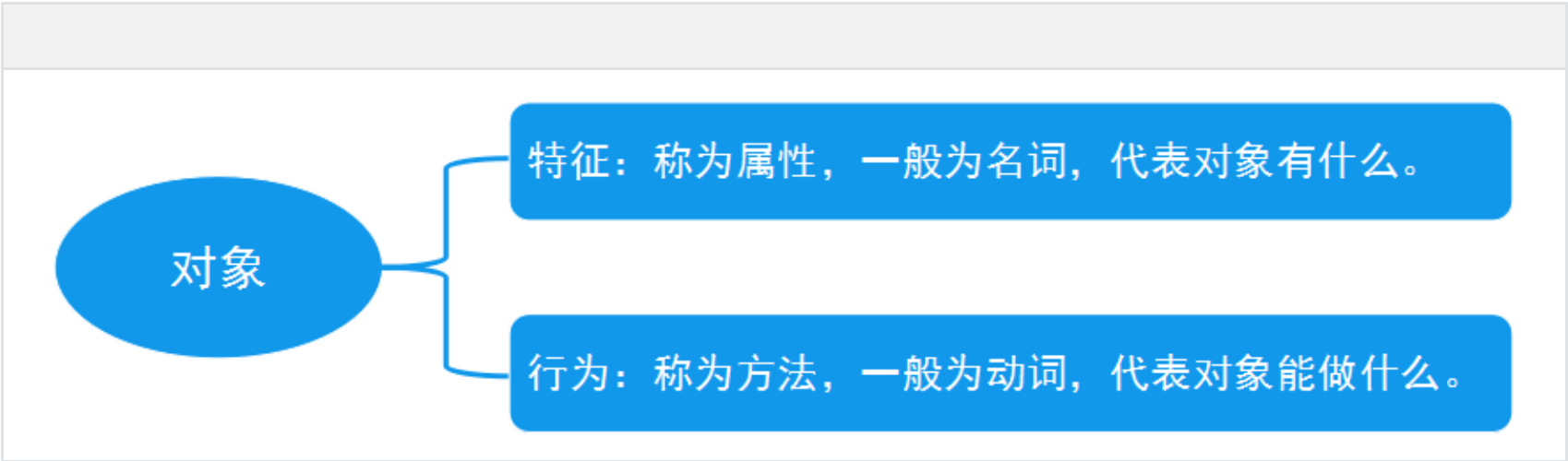
所有回答都很抽象，没有特别明确的答案。

在程序员的眼里，世界的组成最为明确：“世界是由无数个对象组成的”。

## 二、什么是对象

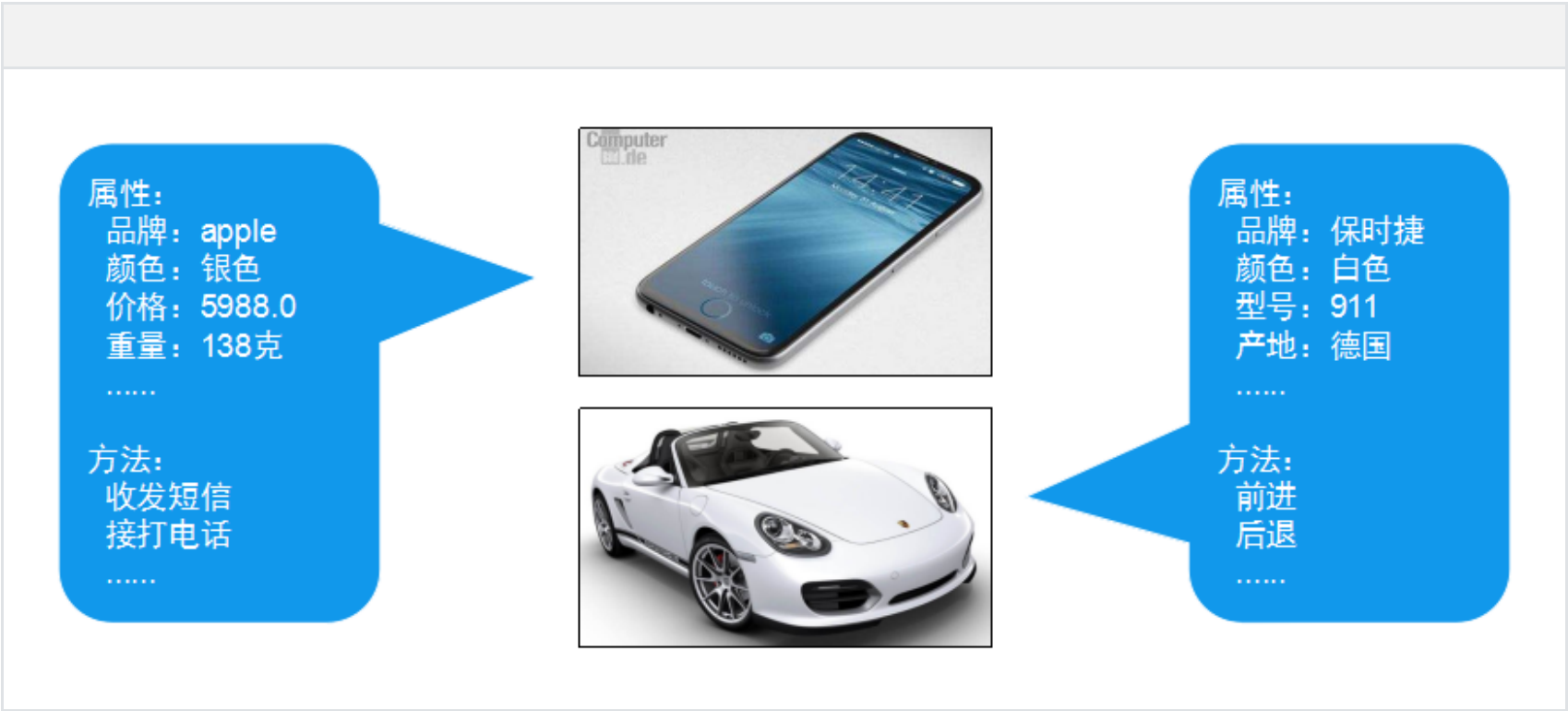
### 2.1 什么是对象

- 面向对象思想（Object Oriented Programming）：
- 一切客观存在的事物都是对象，万物皆对象。
- 任何对象，一定具有自己的特征和行为。



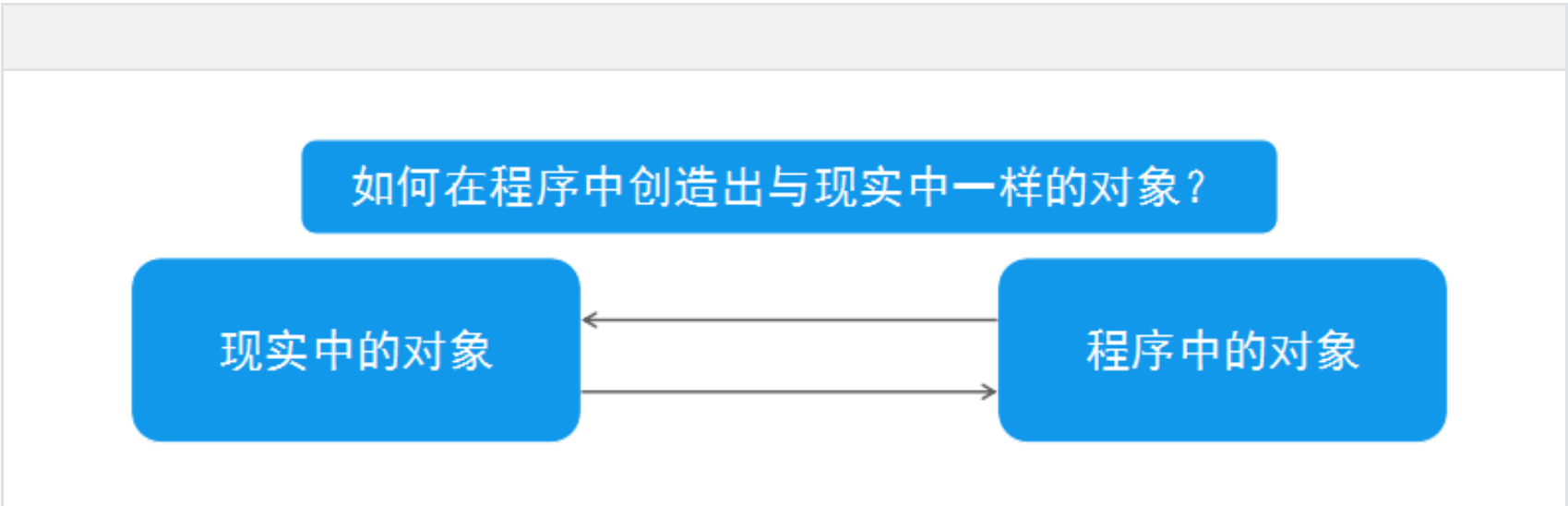
### 2.2 现实中的对象

请分析以下对象都有哪些属性和方法？



### 2.3 程序中的对象

- 如何使用程序模拟现实世界，解决现实问题？
- 首先，在程序当中，必须具有和现实中相同的对象，用以模拟现实世界。
- 然后，使用程序中的对象代表现实中的对象，并执行操作，进而解决现实问题。



现实中的对象多数来自于“模板”，程序中的对象也应该具有“模板”。

三、什么是类【重点】

3.1 什么是类




汽车设计图纸规定了该款汽车所有的组成部分，包括外观形状、内部结构、发动机型号、安全参数等具体的信息。这即为现实对象的模板。程序中的模板也有相同作用，称之为“类”。

按照设计图纸创造出来的汽车，才是真实存在、切实可用的实体，所以汽车实体被称为现实中的对象。而通过程序中的模板创造出来的实体，即为程序中的对象，称之为“对象”。

3.2 类的抽取

在一组相同或类似的对象中，抽取出具体的特征和行为，保留所关注的部分。



属性：  
品种  
年龄  
性别  
毛色  
.....

方法：  
吃  
睡  
.....

3.3 类的定义

```
public class Dog {  
  
    String breed; //品种  
    int age; //年龄  
    String sex; //性别  
    String furColor; //毛色  
  
    public void eat(){  
        System.out.println("eating...");  
    }  
  
    public void sleep(){  
        System.out.println("sleeping...");  
    }  
}
```

类名

属性：通过变量表示，又称实例变量。  
语法：数据类型 属性名；  
位置：类的内部，方法的外部。

方法：通过方法表示，又称实例方法。  
语法：  
public 返回值类型 方法名(形参){  
 //方法的主体  
}  
注意：不再书写static，后续详解。

- 属性：通过变量表示， 又称实例变量。
- 语法：数据类型 属性名；
- 位置：类的内部，方法的外部。
- 方法：通过方法表示， 又称实例方法。

- 语法：  
public 返回值类型 方法名(形参){  
    //方法的主体  
}
- 注意：不再书写static，后续详解。

3.4 对象的创建

将对象保存在相同类型的myDog变量中，myDog变量称为“对象名”或“引用名”

```
public class TestCreateObject {  
    public static void main(String[] args) {  
  
        Dog myDog = new Dog();  
  
        myDog.breed = "萨摩";  
        myDog.age = 2;  
        myDog.sex = "公";  
        myDog.furColor = "白色";  
  
        System.out.println(myDog.breed + "\t" + myDog.age + "\t" + myDog.sex + "\t" + myDog.furColor);  
  
        myDog.eat();  
        myDog.sleep();  
  
    }  
}
```

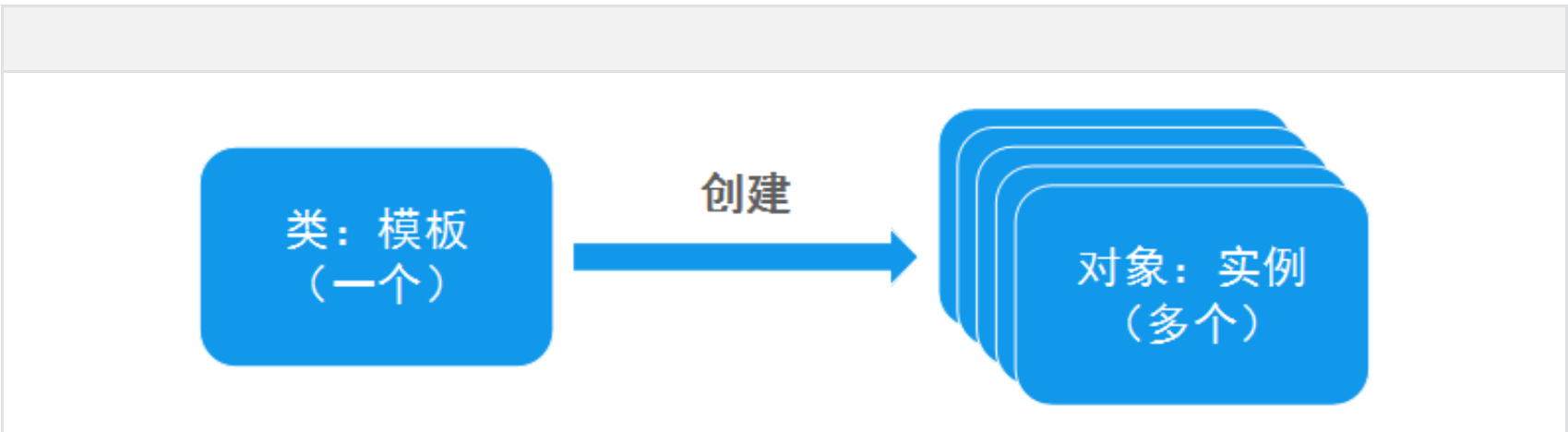
基于Dog类创建对象

访问属性：对象名.属性名 = 值；//赋值

访问属性：对象名.属性名；//取值

调用方法：对象名.方法名()；

3.5 类与对象的关系



- 类：定义了对象应具有的特征和行为，类是对象的模板。
- 对象：拥有多个特征和行为的实体，对象是类的实例。

```
public class TestOOP {  
    public static void main(String[] args) {  
  
        // 1. 创建Dog类型的对象，并保存在dog1变量当中  
        Dog dog1 = new Dog();  
  
        // 2. 访问属性：为各个属性赋值  
        dog1.breed = "萨摩"; // 句点 = 的  
        dog1.age = 3;  
        dog1.sex = "公";  
        dog1.furColor = "白色";  
  
        // 3. 访问属性：从各个属性中取值  
        System.out.println(dog1.breed + "\t" + dog1.age + "\t" + dog1.sex + "\t" + dog1.furColor);  
  
        // 4. 调用对象的方法  
        dog1.eat();  
        dog1.sleep();  
  
        System.out.println("-----");  
  
        Dog dog2 = new Dog();// The value of the local variable dog2 is not used  
        // 这个局部变量 dog2 没有使用过（变量一旦使用，这个黄色警告线就没有了）  
  
        dog2.breed = "哈士奇";  
        dog2.age = 2;  
        dog2.sex = "母";  
        dog2.furColor = "灰白";  
  
        System.out.println(dog2.breed + "\t" + dog2.age + "\t" + dog2.sex + "\t" + dog2.furColor);  
  
        dog2.eat();  
        dog2.sleep();  
  
    }  
}
```



```
class Dog {
    // 在现实中的一组相同或类似的对象中，提取共性的特征和行为，保存在程序中的模板里（类）

    // 属性-实例变量
    String breed; // 品种
    int age; // 年龄
    String sex; // 性别
    String furColor;// 毛色

    // 方法-实例方法
    public void eat() {
        System.out.println("eating...");
    }

    public void sleep() {
        System.out.println("sleeping...");
    }

}
```

四、类的组成【重点】

4.1 实例变量

思考：之前学习局部变量时，要求必须先赋值再使用，否则编译错误。对于实例变量而言，未赋值并不会编译错误，能否直接访问？

```
public class TestCreateObject {
    public static void main(String[] args) {

        Dog myDog = new Dog();

        //省略赋值语句

        System.out.println(myDog.breed + "\t" + myDog.age + "\t" + myDog.sex + "\t" + myDog.furColor);
    }
}
```

实例变量的默认值：  
整数：0  
小数：0.0  
字符：\u0000 （空格）  
布尔：false  
其他：null

运行结果：null 0 null null

4.2 实例变量与局部变量的区别

	局部变量	实例变量
定义位置	方法或方法内的结构当中	类的内部，方法的外部
默认值	无默认值	字面值（与数组相同）
使用范围	从定义行到包含其结构结束	本类有效
命名冲突	不允许与局部变量重名	不允许与实例变量重名 可与局部变量重名，局部变量优先

```
public class TestInstanceVsLocal {

    public static void main(String[] args) {

        int a = 0; //局部变量，先赋值，再使用

        System.out.println(a); //快速补全快捷键 alt + /

        if(true) {
            int b = 20;
        }

        Cat cat1 = new Cat(); //局部变量

        System.out.println(cat1.breed);

        System.out.println(cat1.age);
    }
}
```

```

//-----

cat1.eat();

}
}

class Cat{
    String breed; //实例变量、属性
    int age;
    String sex;

    public void eat() {
//        System.out.println("品种: " + breed);

        int age = 10;

        System.out.println(age);
    }
}

```

### 4.3 实例方法

- 对象的实例方法包含两部分：方法的声明和方法的实现。
- 方法的声明：
  - 代表对象能做什么。
  - 组成：修饰符 返回值类型 方法名(形参列表)
- 方法的实现：
  - 代表对象怎么做：即如何实现对应的功能。
  - 组成：{ 逻辑代码 }

### 4.4 课堂案例

- 定义学生类：
- 属性：姓名(name)、年龄(age)、性别(sex)、分数(score)
- 方法：打招呼(sayHi) //打印学生所有信息
- 创建多个学生对象，为其各个属性赋值，并调用方法。

```

public class TestStudent {
    public static void main(String[] args) {

        Student stu1 = new Student();

        stu1.name = "tom";
        stu1.age = 20;
        stu1.sex = "男";
        stu1.score = 99.0;

        stu1.sayHi();

        Student stu2 = new Student();

        stu2.name = "marry";
        stu2.age = 18;
        stu2.sex = "女";
        stu2.score = 99.5;

        stu2.sayHi();

    }
}

class Student{

    String name;
    int age;
    String sex;
    double score;

    public void sayHi() {

        System.out.println("大家好, 我是" + name + ", 今年" + age
            + "岁, 性别不太明显, 我直说了, 我是" + sex + "的, 此次考试分数: " + score);

    }

}

```

## 五、方法重载

### 5.1 方法重载概念

有些情况下，对象的同一种行为可能存在多种实现过程。

例如：人对对象的“吃”行为，吃饭和吃药的过程就存在差异。

```
public class Person {
    public void eat(食物 a){
        //食物放入口中
        //咀嚼
        //咽下
    }
    public void eat(药物 b){
        //药物放入口中
        //喝水
        //咽下
    }
    public void eat(口香糖 c){
        //口香糖放入口中
        //咀嚼
        //吐出
    }
}
```

到底采用哪种实现过程，需要取决于调用者给定的参数

### 5.2 方法重载

**重载（Overload）**：一个类中定义多个相同名称的方法。

**要求：**

- **方法名称相同。**
- **参数列表不同(类型、个数、顺序)。**
- **与访问修饰符、返回值类型无关。**

调用带有重载的方法时，需要根据传入的实参去找到与之匹配的方法。

好处：灵活、方便、屏蔽使用差异。

```
public class TestCalculator {
    public static void main(String[] args) {

        Calculator calc1 = new Calculator();

        System.out.println( calc1.add( 10.0 , 5.2) );

        System.out.println( calc1.add(20.0, 6.6) );


        //可能存在3个值相加求和
        System.out.println( calc1.add(11,22,33) );


        /*
        System.out.println("Hello Everyone");
        System.out.println(123);

        int a = 10;
        System.out.println(a);
        System.out.println('A');

        System.out.println("hello");
        */

    }
}
```

```
//计算器
class Calculator{

    public double add(double num1 , double num2) {
        return num1 + num2;
    }

    public double add(double num1 , double num2 , double num3) {
        return num1 + num2 + num3;
    }
}
```

### 5.3 代码调错

思考：以下方法是不是重载？

- public void m(int a){}
- public void m(int b){}

两个方法的方法名称和参数列表都相同，只有参数名称不一样，编译报错。

注意：只是参数名称不同，并不能构成方法的重载。

## 六、构造方法【重点】

### 6.1 构造方法

构造方法：类中的特殊方法，主要用于创建对象。

特点：

- 名称与类名完全相同。
- 没有返回值类型。
- 创建对象时，触发构造方法的调用，不可通过句点手动调用。

注意：如果没有在类中显示定义构造方法，则编译器默认提供无参构造方法。

### 6.2 对象创建过程

```
public class TestConstructors {
    public static void main(String[] args) {
        Student s = new Student();
    }
}

class Student{
    String name;
    int age;
    String sex;
    double score;

    public Student(){
        System.out.println("Student() Executed");
    }
}
```

new Student ()；触发对象创建

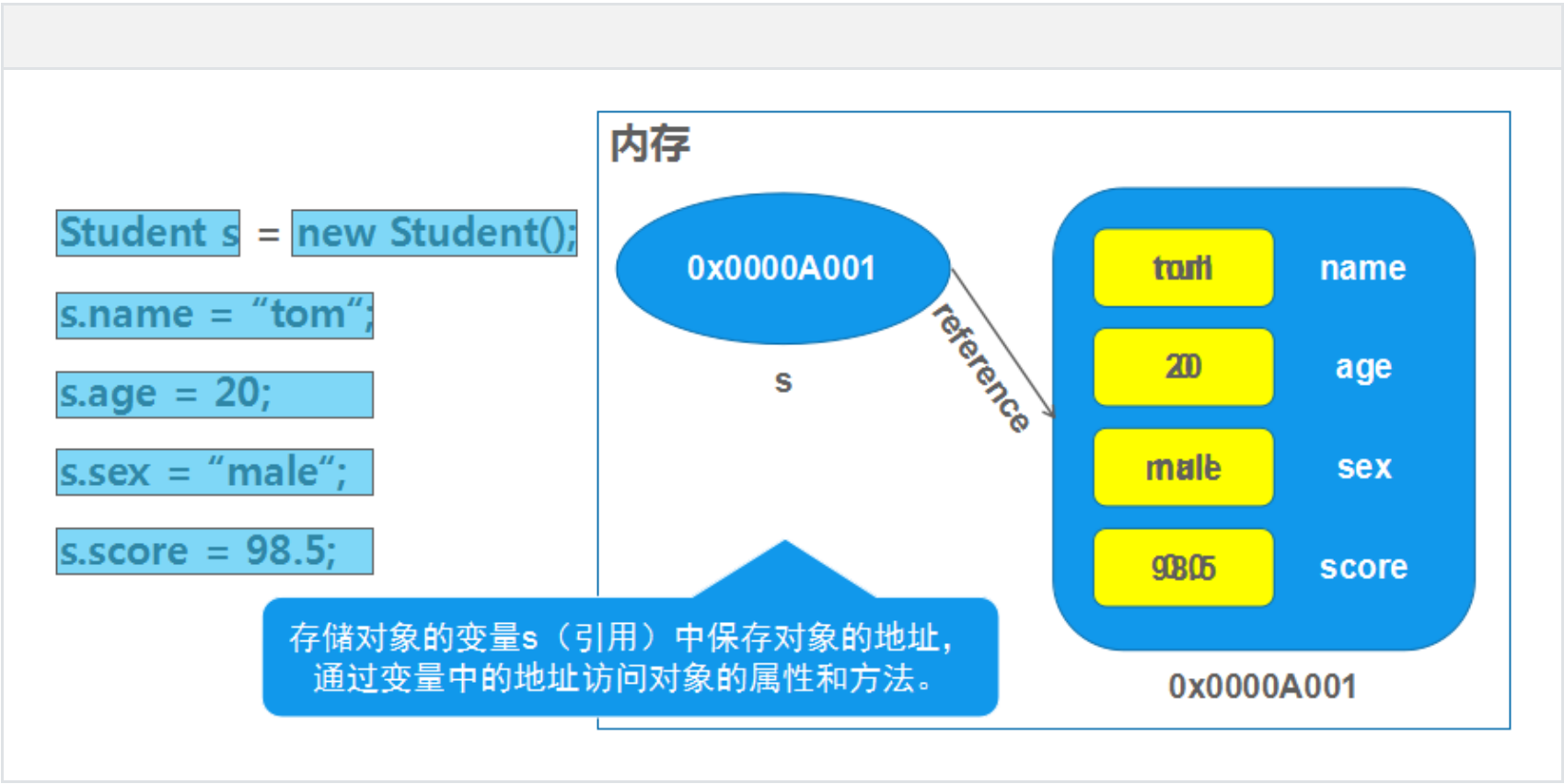
- 对象的创建过程：
  - 内存中开辟对象空间
  - 为各个属性赋予初始值
  - 执行构造方法中的代码
  - [将对象的地址赋值给变量]

对象的创建过程：

- 内存中开辟对象空间
- 为各个属性赋予初始值
- 执行构造方法中的代码
- [将对象的地址赋值给变量]

### 6.3 对象的内存分配



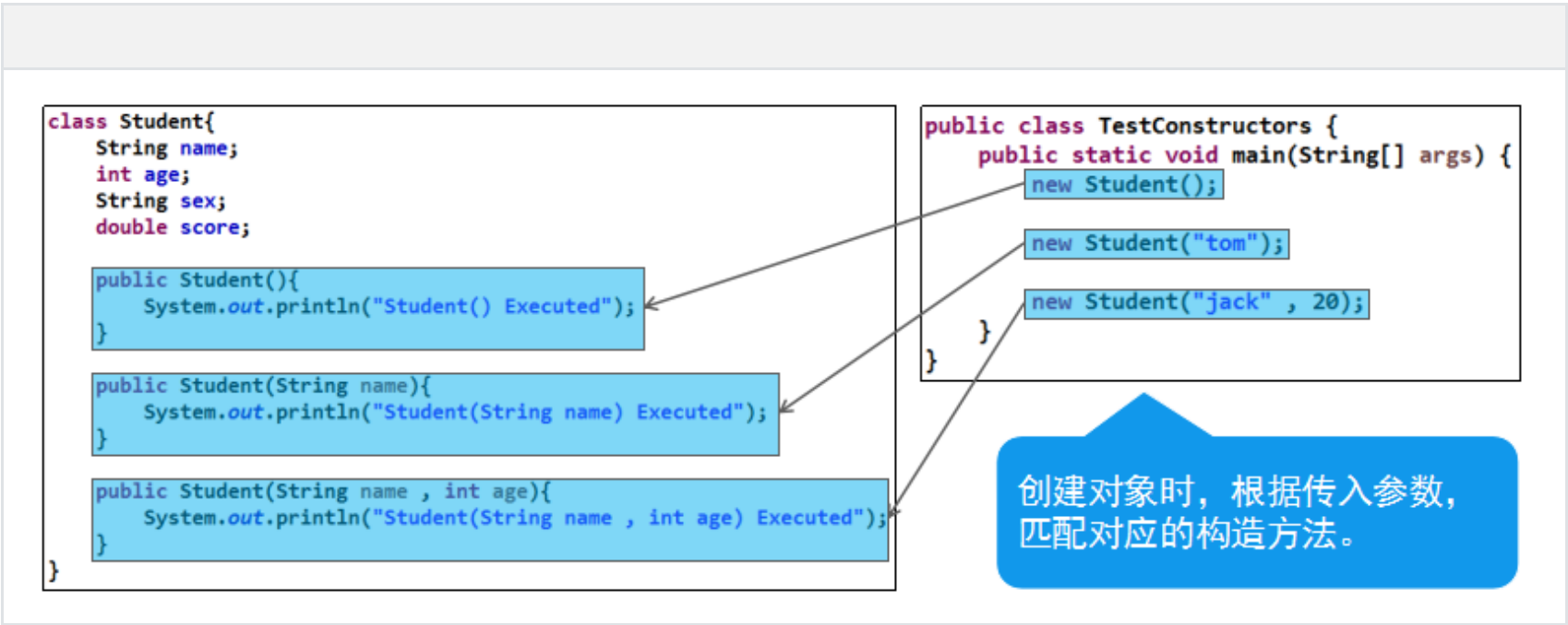


存储对象的变量s（引用）中保存对象的地址，通过变量中的地址访问对象的属性和方法。

```
public class TestConstructor {  
  
    public static void main(String[] args) {  
  
        //1.执行new Student(), 进行构造对象  
        //2.堆内存中开辟对象空间  
        //5.将构造后的对象地址，赋值给s1变量  
  
        Student s1 = new Student();//0x00000001  
  
        System.out.println(s1.name);//null  
  
        s1.sayHi();  
    }  
}  
  
class Student{  
    String name;  
    int age;  
    char sex;  
    double score;  
  
    public Student() {  
        //3.初始化属性，为属性赋默认值  
        //4.执行构造方法中的逻辑代码  
        System.out.println("name当前的值为: " + name);  
        System.out.println("---Student() Executed---");  
    }  
  
    public void sayHi() {  
        System.out.println("大家好");  
    }  
}
```

6.4 构造方法重载

构造方法也可重载，遵循重载规则。



创建对象时，根据传入参数，匹配对应的构造方法。

```
public class TestConstructorOverload {

    public static void main(String[] args) {

        new Teacher();

        new Teacher("eric");

        new Teacher("marry",20);
    }
}

class Teacher{

    String name;
    int age;
    String sex;
    double salary;

    public Teacher() {
        System.out.println("---Teacher() Executed---");
    }

    public Teacher(String name) {
        System.out.println("---Teacher(String name) Executed---");
    }

    public Teacher(String name, int age) {
        System.out.println("---Teacher(String name, int age) Executed---");
    }

}
```

6.5 默认构造方法

```
public class TestConstructors {
    public static void main(String[] args) {
        Student s = new Student();
    }
}

class Student{
    String name;
    int age;
    String sex;
    double score;

    public Student(String n, int a, String s, double sc) {
    }
}
```

编译错误：无参构造方法未定义

在类中，如果没有显示定义构造方法，则编译器默认提供无参构造方法。

如已手动添加有参构造方法，则无参构造方法不再默认提供，可根据需求自行添加。

- 在类中，如果没有显示定义构造方法，则编译器默认提供无参构造方法。
- 如已手动添加有参构造方法，则无参构造方法不再默认提供，可根据需求自行添加。

6.6 构造方法为属性赋值

```
public class TestConstructors {
    public static void main(String[] args) {
        Student s = new Student("tom",20,"male",98.5);
        System.out.println(s.name+"\t"+s.age+"\t"+s.sex+"\t"+s.score);
    }
}

class Student{
    String name;
    int age;
    String sex;
    double score;

    public Student(String n, int a, String s, double sc) {
        name = n;
        age = a;
        sex = s;
        score = sc;
    }
}
```

创建对象的同时，将值传入构造方法

运行结果：tom 20 male 98.5

由构造方法为各个属性赋值

- 创建对象的同时，将值传入构造方法
- 由构造方法为各个属性赋值

## 七、this关键字

### 7.1 this关键字

- 类是模板，可服务于此类的所有对象；
- this是类中的默认引用，代表当前实例；
- 当类服务于某个对象时，this则指向这个对象

```
public class TestThisKeyword {
    public static void main(String[] args) {
        Student s1 = new Student();//0x0000A001
        s1.sayHi();

        Student s2 = new Student();//0x0000B002
        s2.sayHi();
    }
}

class Student{
    String name;
    int age;
    String sex;
    double score;

    public void sayHi(){
        System.out.println(this.name);
    }
}
```

当创建s1对象时，this指向0x0000A001，访问的name属性即是0x0000A001地址中的name空间；当创建s2对象时，this指向0x0000B002，访问的name属性即是0x0000B002地址中的name空间；

### 7.2 this关键字的两种用法

#### 7.2.1 第一种用法

this第一种用法：调用实例属性、实例方法。如：this.name、this.sayHi()。

```
public class TestThisKeyword {
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.sayHi();
    }
}

class Student{
    String name = "tom";

    public void sayHi(){
        String name = "jack";
        System.out.println(name);
        System.out.println(this.name);
    }
}
```

当实例变量和局部变量重名时，优先访问局部变量；此时，如需访问实例变量，需要增加this.前缀。不存在重名时，则可省略this.

运行结果：  
jack  
tom

- 当实例变量和局部变量重名时，优先访问局部变量。
- 此时，如需访问实例变量，需要增加this.前缀。
- 不存在重名时，则可省略this。

```
public class TestThisKeyword {
    public static void main(String[] args) {

        /*
        * Student s2 = new Student();//0x00000001
        *
        * s2.setName("annie");
        *
        * System.out.println(s2.name);
        *
        *
        * Student s3 = new Student();//0x00000002
        *
        * s3.setName("abby");
        *
        * System.out.println(s3.name);
        */
    }
}
```

```
Student s1 = new Student("tom", 20, "male", 99.0);//0x00009999

System.out.println(s1.name + "\t" + s1.age + "\t" + s1.sex + "\t" + s1.score);
}
}

//this = 0x00009999
class Student {

    String name;
    int age;
    String sex;
    double score;

    public Student() {

    }

    public Student(String name, int age, String sex, double score) {
        System.out.println("---四参构造被执行---");
        this.name = name;
        this.age = age;
        this.sex = sex;
        this.score = score;
    }

    public void setName(String n) {
        /* 0x00000002 */ name = n;
    }
}
```

```
public class TestThisKeyword2 {

    public static void main(String[] args) {

        Car myCar = new Car();//0x0000AAA

        myCar.start();
    }

}

class Car{//this = 0x0000AAA

    String brand;
    int speed;
    double price;

    //一键启动
    public void start() {
        System.out.println("全程通电");
        System.out.println("发动机发动");
        System.out.println("中控屏亮起");
        System.out.println(".....");

        this.run();//this可以省略，隐式存在

        System.out.println(brand);
    }

    public void run() {
        System.out.println("前进.....");
    }
}
```

### 7.2.2 第二种用法

this第二种用法：调用本类中的其他构造方法。如：this()、this(实参)。

```
class Student{

    String name;
    int age;
    String sex;
    double score;

    public Student(String name, int age, String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
    }

    public Student(String name, int age, String sex, double score) {
        this(name , age , sex);
        this.score = score;
    }

}
```

在构造方法中，调用本类的其他构造方法，即可复用构造方法中的逻辑代码。

包含多条冗余代码。

this()：调用无参构造  
this(实参)：调用有参构造  
注：必须在构造方法的首行

四参构造将接收到的实参直接传递给三参构造进行属性赋值。

- this()：调用无参构造
- this(实参)：调用有参构造
- 注：必须在构造方法的首行

```
public class TestThisKeyword3 {

    public static void main(String[] args) {

        //Teacher t1 = new Teacher();

        Teacher t2 = new Teacher("michael" , 30 , "male" , 15000.0);
        System.out.println(t2.name + "\t" + t2.age + "\t" + t2.sex + "\t" + t2.salary);

        //Teacher t3 = new Teacher("marry",29,"female");
        //System.out.println(t3.name + "\t" + t3.age + "\t" + t3.sex + "\t" + t3.salary);

    }

}

class Teacher{

    String name;
    int age;
    String sex;
    double salary;

    public Teacher() {
        System.out.println("无参构造方法执行完毕");
    }

    public Teacher(String name) {
        this.name = name;
    }

    public Teacher(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    public Teacher(String name , int age) {
        this.name = name;
        this.age = age;
    }

    public Teacher(String name , int age , String sex) {
        this.name = name;
        this.age = age;
        this.sex = sex;
        System.out.println("三参构造执行完毕, name、age、sex赋值成功");
    }

    //          "michael" ,    30 ,    "male" ,    15000.0
    public Teacher(String name , int age , String sex , double salary) {
        this(name , age , sex);//this([实参])必须在构造方法的首行，仅可在构造方法中，不能在普通方法中
        this.salary = salary;
        System.out.println("四参构造执行完毕, salary赋值成功");
    }

}
```