第3章　类和对象

3.1　面向对象技术基础

3.1.1　面向对象基本概念

Java语言是基于面向对象技术的一种高级程序语言。面向对象技术强调在软件开发过程中面向客观世界或问题域中的事物，采用人类在认识客观世界的过程中普遍运用的思维方法，直观、自然地描述客观世界中的有关事物。

在面向对象技术出现之前，程序员用面向过程的方法开发程序。面向过程的方法把密切相关、相互依赖的数据和对数据的操作相互分离，这种实质上的依赖与形式上的分离使得大量程序不但难以编写，而且难以调试和修改。面向对象技术则是一种以对象为基础，以事件或消息来驱动对象执行处理的程序设计技术。它以数据为中心而不是以功能为中心来描述系统，数据相对于功能而言具有更强的稳定性。

面向对象技术中提到的“对象”代表的是客观世界中的某个具体事物，对象的概念是面向对象技术的核心，是现实世界中某个具体的物理实体在计算机逻辑中的映射和体现，它可以是有形的，也可以是无形的。以现实世界为例，人们日常生活中用的笔记本电脑就是一种具体存在的物理实体，它拥有外形、尺寸、颜色等外部特性，并且具有开、关等功能。

那么如何把笔记本电脑这样的物理实体转化为面向对象技术中的对象呢？这里需要介绍“类”的概念。面向对象技术将数据和对数据的操作封装在一起，作为一个整体来处理，采用数据抽象和信息隐蔽技术，将这个整体抽象成一种新的数据类型——类。考虑不同类之间的联系和类的重用性，面向对象的程序设计方法将客观事物抽象成为“类”，并通过类的“封装”“继承”和“多态”等特性实现软件的可扩充性和可重用性。

概括来讲，**类是同种对象的集合与抽象**。在面向对象的程序设计中，定义类的概念来表述同种对象的公共属性和特点。类是一种抽象的数据类型，它是具有一定共性的对象的抽象，而属于类的某一对象则被称为是类的一个实例，是类的一次实例化的结果。

因此，可以创建一个笔记本电脑类（见程序3-1），用类中的变量来表示笔记本电脑的各项属性（如外形、尺寸、颜色），用类的方法来表示笔记本电脑能执行的各项行为或功能（如开和关等）。笔记本电脑类是所有笔记本电脑对象的集合，同时又具有所有笔记本电脑都拥有的基本属性和基本行为功能。

【程序3-1】　NotebookPC.java。

```
public class NotebookPC{
    double measurement;                    //定义尺寸
    String color;                          //定义颜色
    String shape;                          //定义形状
```

```
    public void turnOn(){
        …//执行开机功能
}

    public void turnOff(){
        …//执行关机功能
    }
}
```

上面建立了 NotebookPC 类，由于它是具有一定共性的对象的抽象，所以它还仅仅是一个抽象概念，但是通过这个类，可以实例化对象，产生一个笔记本电脑的具体实例。例如，某台在商场销售的笔记本电脑：黑色、14 英寸、宽屏。该笔记本电脑在现实中是一个具体存在的物理实体，并且属性也是具体的，因此通过 NotebookPC 类可以实例化一个对象，来代表该笔记本电脑。可在程序 3-1 中增加 main 方法，来实例化该对象：

```
public static void main(String[] args){
    NotebookPC MyPC1=new NotebookPC();              //实例化对象
    MyPC1.color="black";                            //分配各项具体属性
    MyPC1.shape="width";
    MyPC1.measurement=14;
}
```

实例化 MyPC1 对象后，就可以对其进行操作，如执行 MyPC1 的方法，使用 MyPC1 的变量，或者与其他对象进行交互。同时系统会给对象分配内存空间，但注意，无论类写得多么庞大，系统都不会给类分配内存空间，因为类是抽象的描述，系统不可能给抽象的东西分配空间，而对象是具体的，实际存在的。

综上所述，用面向对象程序设计思想解决实际问题可以归纳为三步。

（1）将实际存在的实体对象抽象成概念世界的抽象数据类型，这个抽象数据类型里面包括了实体中与需要解决的问题相关的属性和功能。如前文提到的把笔记本电脑进行抽象概括，得到相关的属性（如外形、尺寸、颜色）和功能（如开和关）。

（2）再用面向对象的工具，如 Java 语言，将这个抽象数据类型用计算机逻辑表达出来，即构造计算机能够理解和处理的类，如程序 3-1 的 NotebookPC 类。

（3）将类实例化，就得到了现实世界实体的映射——对象，在程序中对对象进行操作，就可以模拟现实世界中的实体上的问题并解决之。如实例化 NotebookPC 类，得到对象 MyPC1。

下面再来分析一个典型案例，以加深对面向对象技术的理解与运用。

某高校要求开发一套简单的学生成绩管理系统，该系统功能如下。

① 教师登录系统后可输入授课课程的成绩供学生查询。

② 教师能统计学生的平均成绩和各等级的学生人数。

③ 学生登录系统后可查询自己的各门课程成绩。

结合面向对象程序设计思想来分析该系统需求，首先要确定问题域中的对象。有些对象有鲜明存在的，如学生、教师；有些对象是隐含的，如课程、成绩。一个系统的设计，有时并

不需要使用全部的对象，要思考对象是否在问题陈述的界限之内、系统是否必须有此对象才能完成任务、在用户与系统的交互中是否必须有此对象等相关问题。系统的设计并不是使用越多对象越好，要考虑到对象间可以是相关的，但仍是独立存在的实体。

在查找对象的过程中，需要确定每个对象都是有属性和功能的。属性是对象的特征，属性可以是数据，也可以是另一种对象。如对学生对象来说，属性可以包括学号和选修课程。功能是对象执行的动作行为，可以是对象做出的或施加给对象的动作，这些行为往往会影响对象的属性。如对教师对象来说，可能是上报成绩和修改成绩。

学生成绩管理系统研究中的对象可能的属性和功能如下。

1. 学生

属性：姓名、性别、学号、班级、专业、院系、学校、登录名和密码等。

功能：登录、查询成绩和聊天等。

2. 教师

属性：姓名、性别、工号、院系、学校、登录名和密码等。

功能：登录、上报成绩、统计成绩、查询成绩、修改成绩等。

3. 课程

属性：课程名、课程编号、学时、学分、学期、授课教师和选修学生等。

功能：设置授课教师、获取授课教师、设置选修学生和获取选修学生等。

4. 成绩

属性：课程、学生和分数等。

功能：设置课程编号、获取课程编号、设置学生编号、获取学生编号、设置分数和获取分数等。

图 3-1 为学生、教师、课程、成绩四种对象进行了建模，模型描述了对象的各种属性和功能。

用 Java 语言实现模型，编写 Score 类、Course 类、Teacher 类和 Student 类。根据实际情况实例化对象，如学生张三使用系统，系统实例化 Student 类才产生“张三”这个对象：

```
Student  stu1=new Student();          //实例化对象
stu1.name="张三";                     //分配各项具体属性
stu1.sex="男";
stu1.stuID="201209196868";
…                                     //剩余代码略
```

3.1.2 面向对象基本特征

面向对象程序设计的特征主要包括抽象、封装、继承、多态性。本节将简单介绍这些特征，并在本章和第 4 章中详细讲解 Java 语言是如何运用这些特征，以使面向对象的思想得到具体的体现。

1. 抽象

“物以类聚，人以群分”就是分类的意思，分类所依据的原则是抽象。抽象就是忽略事物中与当前目标无关的非本质特征，更充分地注意与当前目标有关的本质特征，从而找出事物

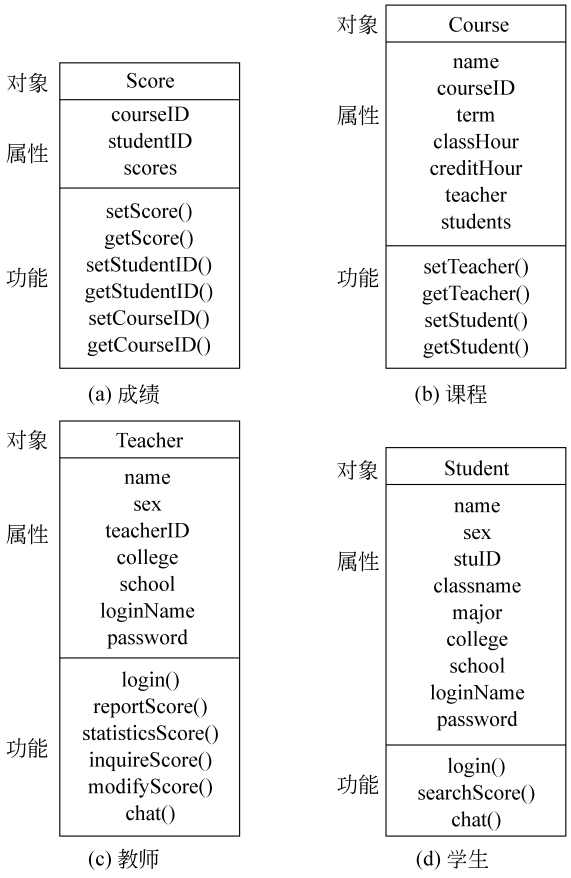

图 3-1 对象建模

的共性，并把具有共性的事物划为一类，得到一个抽象的概念。

一个类定义了一组对象。类具有行为功能，它描述一个对象能够做什么以及做的方法，它们是可以对这个对象进行操作的程序和过程，类是对象的抽象。一个对象是一个类的一个实例，它代表一个现实物理“事件”。

例如，在学生成绩管理系统中，考查学生张三这个对象时，只关心与设计系统相关的信息，如他的班级、学号、成绩等，而忽略他的兴趣、身高等信息。因此，抽象性是对事物的抽象概括描述，实现了客观世界向计算机世界的转换。将客观事物抽象成对象及类是比较难的过程，也是面向对象方法的第一步。

2. 封装

封装有两个含义：一是把对象的全部属性和功能结合在一起，形成一个不可分割的独立单位。对象的属性值一般只能由这个对象的功能来读取和修改。二是尽可能隐蔽对象的内部细节，对外形成一道屏障，与外部的联系只能通过外部接口实现。程序员只需要关心它对外所提供的接口，而不需要注意其内部细节，即怎么提供这些服务。

例如，Score 类对课程的成绩、设置成绩、读取成绩等属性和功能进行了封装，教师和学生对象都需要查询成绩，Score 类的 getScore 方法提供了能根据不同的对象查询成绩的功能，而教师和学生对象不需要了解系统是如何执行成绩查询的细节，只要在自身对象相应方

法中调用 Score 类的 getScore 方法即可。

封装将对象的使用者与设计者分开,使用者不必知道对象功能实现的细节,只需要用设计者提供的外部接口就可以去执行某个功能。封装的结果实际上隐蔽了复杂性,并提供了代码重用性,从而降低了软件开发的难度。

3. 继承

客观事物既有共性,又有特性。运用抽象的原则就是舍弃对象的特性,提取其共性,从而得到适合一个对象集的类。如果在这个类的基础上,再考虑抽象过程中各对象被舍弃的那部分特性,则可形成一个新的类,这个类具有前一个类的全部特征,又有自己的特性,从而形成一种层次结构,即继承结构。

继承是一种连接类与类的层次模型。继承是指特殊类的对象拥有其一般类的属性和功能。继承意味着"自动地拥有",即特殊类中不必重新定义已在一般类中定义过的属性和功能,而它却自动地、隐含地拥有其一般类的属性与功能。因此,继承是传递的,体现了大自然中特殊与一般的关系。

例如,学生对象中有留学生,为方便管理,需要记录留学生的国籍。可以设计一个 ForeignStudent 类来继承 Student 类,这样 ForeignStudent 类就拥有 Student 类中定义过的属性和功能,并且可以增加自己独有的属性国籍。如果不采用继承,重新写一个 ForeignStudent 类,那么就不得不在类中重新定义姓名、性别、学号、班级等属性和功能。

在软件开发过程中,继承实现了软件模块的可重用性、独立性,缩短了开发周期,提高了软件开发的效率,同时使软件易于维护和修改。这是因为要修改或增加某一属性或行为,只需在相应的类中进行改动,而它派生的所有类都自动地、隐含地做了相应的改动。

4. 多态性

面向对象设计借鉴了客观世界的多态性,体现在收到不同的对象发来的消息时能产生多种不同的行为方式,即指类中同一方法名的方法能实现不同的功能,且可以使用相同的调用方式来调用这些具有不同功能的同名方法。

例如,Score 类可以编写两个 getScore 方法:一个为教师对象提供查询全班学生成绩的服务;另一个则为学生对象提供查询该学生成绩的服务。这两个方法名虽然相同,但系统能根据请求对象的不同而调用正确的 getScore 方法。

3.2 类

3.2.1 类的定义

类是同种对象的集合与抽象。一旦定义了类,就可以用这种类来创建对象。因此,也常说类就是对象的模板,而对象就是类的一个实例。类的定义分为类首声明和类主体两部分。

类首声明定义类的名字、访问权限以及与其他类的关系等。类首声明的格式([]中的内容表示可选)如下:

```
[<修饰符>] class<类名>[extends<超类名>] [implements<接口名>]
```

修饰符:表示类的访问权限(public、默认方式等)和一些其他特性(abstract、final 等);

一个类可以同时有多个修饰符(任意排序),但不能有相同的修饰符。关于修饰符将在 4.1 节中进行阐述。

class:类定义的关键字,一般定义一个类都需要用到该关键字。

extends:表示类和另外一些类(超类)的继承关系。将在 4.2 节中进行阐述。

implements:表示类实现了某些接口。将在 4.5 节中进行阐述。

类主体定义类的成员,包括变量和方法。类主体的格式如下:

```
{
    <成员变量的声明>
    <成员方法的声明及实现>
}
```

成员变量即类的数据,反映了对象的属性和状态。成员方法即类的行为,实现对数据的操作,反映了对象的功能。

现在定义一个 Triangle 类(见程序 3-2),来回顾类中的各元素。

【程序 3-2】 Triangle.java。

```
public class Triangle {                          //类首声明
    double length=10.0;                          //定义变量
    double height=5.0;                           //定义变量

    //定义方法
    double area(){
        return length * height/2.0;
}

    //定义 main 方法
    public static void main(String args[]){
        double s;                                //定义变量
        s=(new Triangle()).area();
        System.out.println("该三角形的面积是: "+s);
    }
}
```

类的开始和结束用{}来标示,类中的变量用于存放数据。由于数据有相应的类型,所以存放数据的变量也要规定类型。类中的方法用来对数据进行处理,从而实现程序的功能。方法名后面都有括号,括号中可能包括参数。方法的开始和结束也用{}来标示。main 方法是 Java 中非常特殊的一个成员方法,Java 程序是从 main 方法开始执行的。包含 main 方法的类叫作主类。

小贴士

Java 文件的命名规则是什么?

修饰符 public 代表该类能被所有的类访问,修饰符 public 表明所定义的类为公共类。注意一个 Java 文件可以包含多个类,但最多只能包含一个公共类,Java 程序的文件不能随便命名,要求公共类必须与其所在的文件同名。许多新手都喜欢在编程时自己定义文件名

称,结果因为与文件内的公共类名称不符,导致程序出错。回顾程序 1-1 的 HelloWorld 程序:

```
//程序 1-1HelloWorld.java
public class HelloWorld{
    …                                          //代码省略
}
```

由于 HelloWorld 类前面有 public 修饰符,因此这是一个公共类,在保存文件时,文件名必须是 HelloWorld.java。要注意 Java 程序的文件名必须和里面的公共类名完全对应,不然程序无法执行。假设 HelloWorld.java 程序中再写一个类 HelloWorld2。

【程序 3-3】 HelloWorld.java。

```
public class HelloWorld{
    public static void main(String argv[]){
        System.out.println("Hello World");
    }
}

class HelloWorld2{                             //定义第二个类
    public static void main(String argv[]){
        System.out.println("Hello World2");
    }
}
```

文件名为 HelloWorld.java,程序没有任何问题。但如果把“public class HelloWorld”改成“class HelloWorld”,把“class HelloWorld2”改为“public class HelloWorld2”。即把第一个类的修饰符 public 移至第二个类上。程序就会编译出错。更正的方法除了恢复原状,还可以更改文件名为 HelloWorld2.java。

如果把“class HelloWorld2”改为“public class HelloWorld2”,也会导致编译出错,因为文件中出现了两个公共类。

总之,记住 Java 文件命名最重要的两条原则:一个 Java 文件最多只能包含一个公共类;Java 文件要求与其内部的公共类同名。

3.2.2 成员变量与成员方法

成员变量定义的格式([]中的内容表示可选)如下:

```
[<修饰符>]<变量类型><变量名>
```

修饰符:表示类访问权限(public、默认方式等)和一些其他特性(static、final、transient 等)。

变量类型:变量的数据类型,可以是基本数据类型或引用数据类型。

变量名:该变量的名称。

如程序 3-2 中的语句“double length=10.0;”:double 表明变量的类型为双精度浮点型,length 是变量名称,初始值为 10.0。

成员方法定义的格式([]中的内容表示可选)如下：

```
[<修饰符>]<返回类型><方法名>([<参数列表>]) [throws<异常类>]{
    方法体
}
```

修饰符：表示类访问权限(public、默认方式等)和一些其他特性(static、final、abstract 等)。

返回类型：执行该方法后返回的数据类型，如果该方法没有返回值，则返回类型必须写为 void。

方法名：该方法的名称。

参数列表：该方法接收的参数。

throws：表示抛出异常，将在第 7 章进行具体阐述。

方法体：方法的具体执行代码。

如程序 3-2 中的语句 double area()：double 表明该方法返回的数据类型是双精度浮点型，area 是方法名称，()是方法的参数列表，即使一个方法没有任何参数，()也是必需的。

定义在类内部方法外的变量称为全局变量，全局变量的作用域是整个类，即在整个类中都能使用该变量。而定义在类内部方法内的变量称为局部变量，该变量的作用域仅限于方法内部。如程序 3-2 中的 length、height 变量都是全局变量，而定义在 main 方法内的变量 s 则是局部变量。如果在另一个方法 area()中，是无法使用变量 s 的，但可以使用全局变量 length 和 height。

3.2.3 构造方法

类的成员方法简称方法，用来实现类的各种功能。另外，Java 语言还提供了一种特殊的方法——构造方法，用来在创建对象时让 Java 系统调用构造方法去初始化新建对象的成员变量。

每次用类来实例化对象时，经常要对类中的变量进行初始化。能在一个对象最初被实例化时就把相应的变量都设置好，程序将更简单并且更简明。Java 语言允许对象在被创建时初始化成员变量，而这种自动初始化的过程就是通过使用构造方法来完成。

构造方法必须有名称，不然编译器无法自动调用构造方法初始化变量。但是在设计构造方法命名问题上，存在着不小的麻烦，原因是构造方法使用的任何名字都可能与类成员方法的名字冲突，所以在 Java 采用构造方法的名字与类名相同。这样一来，可保证构造方法会在对象初始化期间的自动调用。

构造方法的格式([]中的内容表示可选)如下：

```
[<修饰符>] <类名>([<参数列表>]){
    方法体
}
```

修饰符：可以有表示方法访问权限的修饰词(public、protected、private 和默认方式等)，但不能有以下非访问性质的修饰词：abstract、final、native、static 或 synchronized。

类名：类名即构造方法名。

参数列表：构造方法接收的参数，可以是 0 个、1 个或多个。

比较成员方法的格式，可以看到两者间差异最大的就是构造方法没有返回类型。在成员方法中，如果没有返回类型，需要把返回类型标为 void，但构造方法连 void 都不需要。

下面来看一个简单的例子(见程序 3-4)，学习如何通过构造方法初始化新建对象的成员变量。

【程序 3-4】 Student.java。

```
public class Student{
    String name;
    char sex;
    int stuID;

    //构造方法 1
    public Student(String stuName, char sex, int stuID){
        name=stuName;
        this.sex=sex;
        this.stuID=stuID;
    }

    //构造方法 2
    public Student(){}

    //定义 main 方法
    public static void main(String args[]){
        //通过构造方法 1 初始化变量
        Student s1=new Student("张三", '男', 20130301);
        //通过构造方法 2 初始化变量
        Student s2=new Student();
    }
}
```

name、sex、stuID 是 Student 类的 3 个成员变量，本例提供了两个构造方法：public Student(String stuName, char sex, int stuID)和 public Student()。可以看到构造方法的名称和类名是相同的，都是 Student。这两个构造方法的唯一区别就是参数列表不同，第一个构造方法接收 3 个参数(stuName、sex、stuID)，第二个构造方法没有任何输入参数，这也是 Java 语言多态性的一种体现，即同一名称的方法能根据参数的不同实现不同的功能。

创建类的实例对象可以通过 new 运算符和构造方法进行，格式如下：

```
new 构造方法名(构造方法参数列表)
```

在定义的 main 方法中，对 Student 类实例化，得到了两个对象 s1 和 s2(即变量 s1 和 s2，它们分别指向相应的两个对象)。观察语句“Student s1 = new Student("张三", '男', 20130301);”，“Student s1”表明创建的对象是 Student 类型，对象名字叫 s1，在等号右边的语句“new Student("张三", '男', 20130301)”中，new 运算符用来表明要创建某个类的实例

对象，“Student("张三", '男', 20130301)”表明调用了例中的“public Student(String stuName, char sex, int stuID)”这个构造方法，stuName 的值为“张三”，sex 的值为“男”，stuID 的值为 20130301。随后程序执行构造方法中的语句：

```
name=stuName;
this.sex=sex;
this.stuID=stuID;
```

执行完毕后，对象 s1 中的变量 name 的值变为“张三”，变量 sex 的值变为“男”，变量 stuID 的值变为 20130301。至此，新建对象的成员变量自动初始化完毕。对象 s2 的实例化过程原理和 s1 相同，只不过它调用的构造方法不需要任何参数传入，因此最后并没有对变量进行赋值。

构造方法在每个类实例化对象时都会用到，因此非常重要。构造方法具有如下三大特点。

(1) 类的构造方名必须和类名相同，这是区别它与成员方法的第一条准则。

(2) 构造方法没有返回值(在构造方法名字前连 void 也不要加)，这是区别它与成员方法的第二条准则。

(3) 如果在类中没有自定义构造方法，则 Java 调用类的默认构造方法，将使用默认值来初始化成员变量。

下面通过改编程序 3-4 了解这一特点。

【程序 3-5】 Student2.java。

```
public class Student2{
    String name;
    char sex;
    int stuID;

    public static void main(String args[]){
        //下面语句删除注释后会发生编译错误
        //Student2 s1=new Student2("张三", '男', 20130301);
        Student2 s2=new Student2();          //通过默认构造方法初始化变量
    }
}
```

在本例中没有写一个构造方法，但是语句“Student2 s2=new Student2();”仍旧能够正常执行，这是因为当系统发现没有为某个类定义构造方式时，它会智能地定义一个默认的构造方法，默认定义的构造方法是不含任何参数的。这也就是程序正常执行的原因。如果这时调用含参数的构造方法(如程序 3-5 中注释掉的语句)，系统无法智能定义一个含参数的构造方法，因此程序编译出错。

特别需要注意的是，如果类中已经有了构造方法，系统就再也不会创建这个默认的无参构造方法了。

关于更多构造方法的用法，将在 4.2 节进行介绍。

3.2.4 main 方法

main 方法是一种特殊的成员方法,是所有 Java 应用程序执行的入口,如果编写的程序希望能单独执行,则必须含有 main 方法。main 方法的写法如下:

```
public static void main (String args[])
```

public 修饰符:public 修饰符表明所有能访问该方法所在类的对象,都能使用该方法。因为 main 方法是 JVM(Java 虚拟机)自动调用的,需要让 JVM 可见,所以 main 方法需要 public 修饰。

static 修饰符:如果一个方法被声明为 static,它就能够在它的类的任何对象被创建之前访问,而不必引用任何对象。由于 main 方法是所有程序的入口,也就是 main 被调用时没有任何对象创建,不通过对象调用某一方法,只有将该方法定义为 static 方法,所以 main 方法是一个静态方法,需要 static 修饰。

void 返回值:JVM 对于 Java 程序来说已经是系统的最底层,由它调用的方法的返回值已经没有任何地方可去,因此,main 方法返回值为空,需用 void 修饰。

String args[]参数:能够接收命令行传入的参数,参数是一个 String 类型的数组。

回顾程序 3-5 的 main 方法,可知通过执行程序,生成了 s1 和 s2 这两个对象。如果把 main 方法注释掉,那么程序 3-5 将无法执行。

3.3 对　　象

3.3.1 对象的生成与使用

当创建一个类时,就可以实例化该种类型的对象。实例化就是为对象分配存储空间,并同时对对象进行初始化,通过用 new 运算符和类的构造方法共同来完成。

实例化的过程可分两步。

第一步,必须声明该类类型的一个变量,这个变量没有定义一个对象。实际上,它只是一个能够引用对象的简单变量。例如:

```
Student s;     //声明一个 Student 类的变量 s,此时 s 的值为 null,没有引用任何对象
```

第二步,创建一个对象的实际的物理副本,并把对于该对象的引用赋给该变量。这是通过使用 new 运算符实现的。new 运算符为对象动态分配内存空间,并返回对它的一个引用值。引用值就是系统分配给对象的内存地址,由于类是一种引用数据类型,因此变量中存储的值不是对象本身,而是引用值。例如:

```
s=new Student();
                  //实例化对象,并把引用值存储在变量 s 中,以后通过调用 s 就可以得到该对象
```

对象内存分配机制如图 3-2 所示。

当然,可以把两步合并成一步,格式如下:

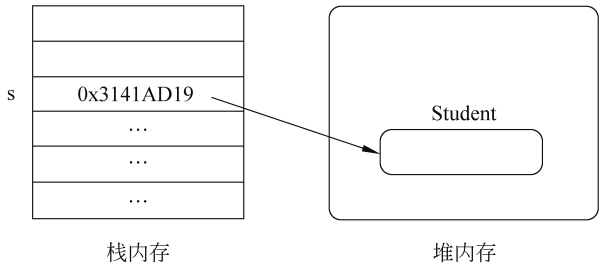

图 3-2 对象内存分配机制

<类名><对象名>=new<构造方法名/类名>(参数);

例如：

```
Student s=new Student();                    //构造方法名和类名都是相同的
```

类具有变量和方法，实例化后的对象当然也包含类中的变量和方法。
成员变量的引用格式如下：

引用对象名.变量名

成员方法的调用格式如下：

引用对象名.方法名([实际参数列表])

阅读程序 3-6 的 Student3 程序代码。

【程序 3-6】 Student3.java。

```
public class Student3{
    String name;
    char sex;
    int stuID;

    public void setName(String stuName){
        name=stuName;
    }

    public static void main(String args[]){
        Student3 s1=new Student3();
        s1.name="张三";
        System.out.println("变量 name 的值为: "+s1.name);
        s1.setName("李四");
        System.out.println("变量 name 的值现在为: "+s1.name);
    }
}
```

分析下列语句可知：

```
Student s1=new Student();                    //得到对象变量 s1
s1.name="张三";                              //为对象 s1 的变量 name 直接赋值
s1.setName("李四");
          //直接调用对象 s1 的方法 setName,注意参数必须和方法中的参数个数、类型要匹配
```

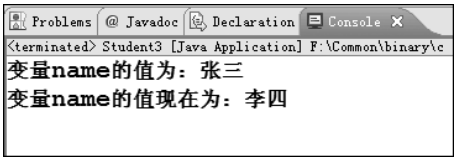

图 3-3　程序 3-6 的输出结果

本例两次对变量 name 进行赋值，第一次通过对象名.变量的方式，第二次通过对象名.方法的方式，setName 方法接收参数 stuName，并把 stuName 的值赋予变量 name。程序输出结果如图 3-3 所示。

下面再次复习类和对象之间的区别。类是一个逻辑构造，是一种新的数据类型，该种类型能被用来创建对象，而对象有物理的真实性，对象占用真正的内存空间。真正了解类和对象的概念与区别是非常重要的。

3.3.2　变量的作用域

3.2.2 节简单讨论了类中全局变量和局部变量的作用范围，本节将结合实例进行更详细讨论。

【程序 3-7】　Student4.java。

```
public class Student4{
    String name;
    char sex;
    int stuID;

    public JStudent4(){}

    public JStudent4(String stuName, char sex, int stuID){
        name=stuName;
        this.sex=sex;
        this.stuID=stuID;
    }

    public void setName(String stuName){
        name=stuName;
    }

    public void setSex(char sex){
        this.sex=sex;
    }

    public void setStuID(int stuID){
        this.stuID=stuID;
    }

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

```
        Student4[] s=new Student4[30];
        for (int i=0; i<s.length; i++){
            s[i]=new Student4();
        }
    }
}
```

分析语句：

```
String name;
char sex;
int stuID;
```

这三个成员变量都是全局变量，它们的作用范围是整个类。

```
public void setName(String stuName)
```

变量 stuName 作为 setName 方法的参数存在，因此它是一个局部变量，仅仅作用于 setName 方法内部。

```
public void setSex(char sex){
    this.sex=sex;
}
```

成员变量 sex 与方法中的局部变量 sex 同名时，成员变量在该方法中被隐藏，因此如果在该方法中出现变量 sex，系统均认为是局部变量 sex，若要引用成员变量，则用 this 关键字。

```
for (int i=0; i<s.length; i++)
```

for 语句块中的变量 i 也是局部变量，作用域仅局限于该 for 语句块中。

程序 3-7 在 main 方法中使用了数组，语句“Student4[] s=new Student4[30];”创建了一个数组对象 s，s 内部包含 30 个元素，每个元素的类型都是 Student4，但这时 s 的值为 null，还没有引用任何对象。通过 for 循环和“s[i]=new Student4();”来创建 Student4 对象，为数组中的每一个元素都进行赋值。

3.3.3 对象的内存分配机制

在 Java 语言中，当将一个对象引用赋值给另一个对象引用时，并没有创建该对象的一个副本，而是仅仅对引用的一个拷贝，这也是 Java 面向对象技术的一个重要原则。程序 3-8 使用程序 3-7 的 Student4 类来生成对象，并进行相关操作。

【程序 3-8】 Student4Test.java。

```
public class Student4Test{
    public static void main(String args[]){
        //调用 Student4 类的无参构造方法生成对象 s1
```

```
        Student4 s1=new Student4();
        s1.setName("张三");              //调用了 Student4 类的 setName 方法
        s1.setSex('男');
        s1.setStuID(20130201);
        //调用 Student4 类的有参构造方法生成对象 s2
        Student4 s2=new Student4("李四", '男', 20130301);
        s2=s1;                           //把 s1 赋值给 s2
        s1=null;                         //把 s1 的值设为空值
    }
}
```

首先,本例第一次出现了类与类之间的交互,在面向对象程序设计中,对象协作是必需的,每个对象都能够接收信息、处理数据和向其他对象发送信息,最后共同协作完成复杂的工作任务。由于 Student4 是一个公共类(类的定义有修饰符 public),Student4Test 能直接调用 Student4 的构造方法生成 Student4 类型的对象。

语句"Student4 s1=new Student4();"调用了 Student4 类的无参构造方法,因此系统将采用默认值来初始化 Student4 中的成员变量。也就是说,如果变量是基本数据类型,变量值采用基本数据类型的默认值,如 boolean 型数据默认值为 false,int 型数据默认值为 0 等;如果变量本身是引用数据类型,则默认值为 null。因此,该语句执行完毕后,内存的分配情况如图 3-4 所示。

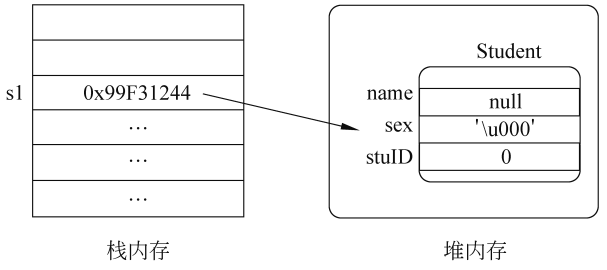

图 3-4　内存分配图(一)

变量 name 的数据类型是 String,String 是一种引用数据类型,因此,默认值为 null。变量 sex 和变量 stuID 都是基本数据类型,系统分别给它们分配相应的默认值。接下来程序执行下列语句为对象 s1 的变量赋值:

```
s1.setName("张三");
s1.setSex('男');
s1.setStuID(20130201);
```

内存分配情况如图 3-5 所示。

注意: String 也是引用数据类型,因此变量 name 存储的值也仅仅是 String 对象存放的内存地址。当赋值完成后,执行语句"Student4 s2 = new Student4("李四", '男', 20130301);"实例化第二个 Student4 的变量 s2,此时的内存分配情况如图 3-6 所示。

执行语句"s2=s1;"后,并没有重复复制该对象,而仅仅是复制一个引用。因此,内存分

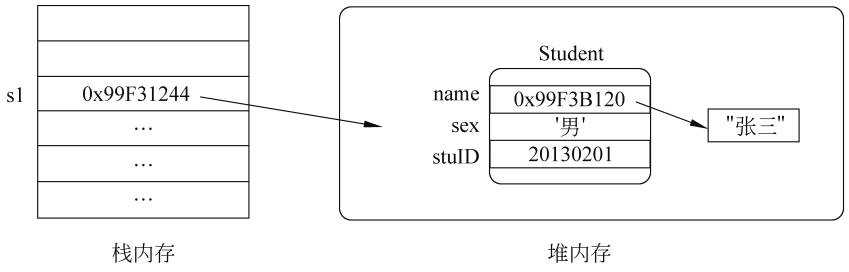

图 3-5　内存分配图(二)

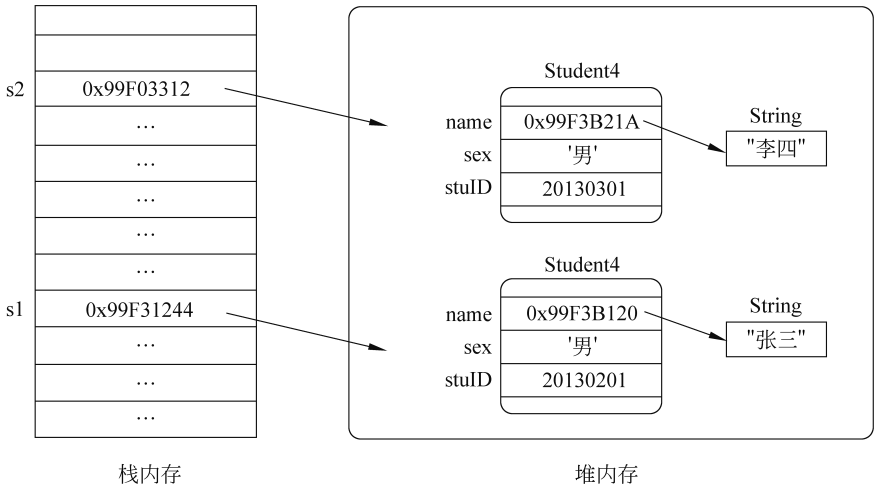

图 3-6　内存分配图(三)

配情况如图 3-7 所示。

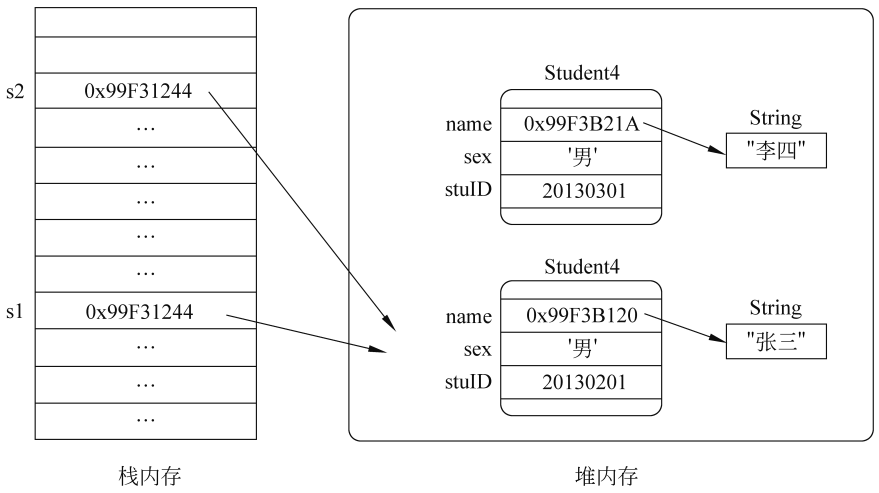

图 3-7　内存分配图(四)

从图 3-7 中可以看到 s2 的值和 s1 的值一样,但该值仅仅只是对对象内存地址的一个引用,并不是真正把对象复制了一遍,而原来变量 s2 所引用的对象并没有被清除掉,仍然留

在内存中，Java 会通过垃圾回收机制自动清除这一对象。最后，程序执行“s1＝null;”，虽然 s1 值为空，但只作用于引用值，并没用真正清空该对象，因此对变量 s2 没有任何影响，内存分配情况如图 3-8 所示。

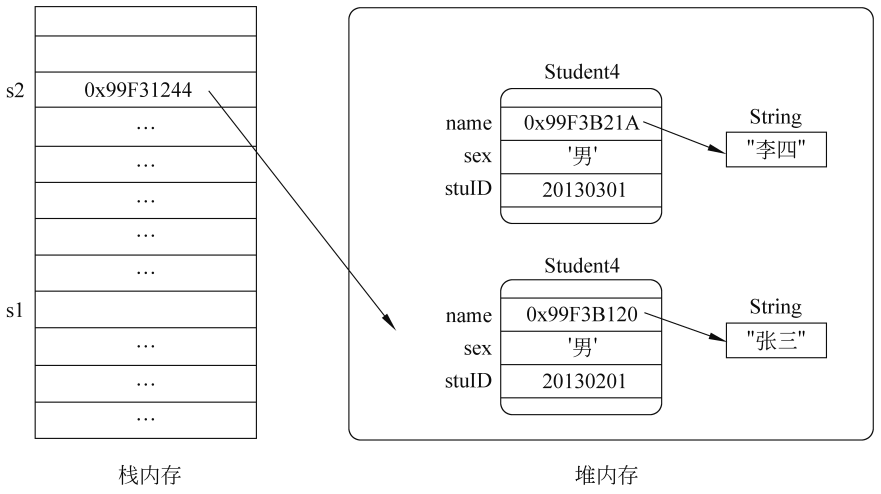

图 3-8　内存分配图（五）

3.3.4　方法参数的传递

方法参数的传递指的是在方法调用时从方法的调用参数带入方法定义的参数的方式。在 Java 语言中，参数传递方式是值传递，即把实际参数的值传递给形式参数。根据参数的数据类型，值传递也可分两种情况。

（1）参数是基本数据类型时，参数的传递为实际值，如 int 型参数 a 的值为 10，传递的值即该参数的实际值 10。

（2）参数是引用数据类型时，由于参数值存储的只是引用对象的地址值，因此参数的传递为引用对象的地址值传递，如 Student4 型参数 s2 的值是一个内存地址值，传递的值也是这个引用值，并不是把真正的对象复制过去。

程序 3-9 是基本数据类型参数和引用数据类型参数传值的例子，下面对其进行详尽分析。

【程序 3-9】　PassTest.java。

```
public class PassTest{
    float ptValue;

    //参数类型是基本数据类型
    public void changeInt(int value){
        value=55;
    }

    //参数类型是引用数据类型
    public void changeStr(String value){
```

```
        value=new String("world");
    }

    //参数类型是引用数据类型
    public void changeObjValue(PassTest ref){
        ref.ptValue=99.0f;
    }

    public static void main(String args[]){
        String str;
        int val;
        //创建 PassTest 类的对象
        PassTest pt=new PassTest();
        //测试基本数据类型参数的传递
        val=11;
        pt.changeInt(val);
        System.out.println("Int value is: "+val);
        //测试引用数据类型参数的传递
        str=new String("Hello");
        pt.changeStr(str);
        System.out.println("Str value is: "+str);
        //测试引用数据类型参数的传递
        pt.ptValue=101.0f;
        pt.changeObjValue(pt);
        System.out.println("Pt value is: "+pt.ptValue);
    }
}
```

程序执行后的输出结果如图 3-9 所示。

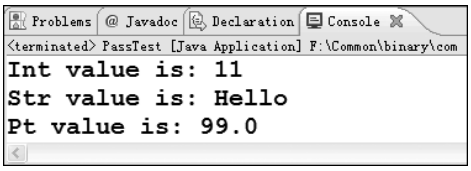

图 3-9 程序 3-9 的输出结果

程序先定义了一个 float 型的成员变量 ptValue，该类型是基本数据类型。随后程序又定义了 3 个方法：changeInt(int value)、changeStr(String value)和 changeObjValue(PassTest ref)。其中，changeInt 方法的输入参数 value 是 int 型(基本数据类型)，而 changeStr 方法和 changeObjValue 方法的输入参数都是引用数据类型。尤其值得注意的是 changeObjValue 方法，它支持以 PassTest 类实例化后得到的对象变量 ref 作为调用参数输入，也就是以该类本身作为参数传入。

在 main 方法中，测试了参数的传递，首先定义了 str(String 型)和 val(int 型)两个局部变量，然后创建了 PassTest 类的对象 pt。由于 PassTest 类没有写任何构造方法，因此在创建过程中使用了系统自动生成的无参构造方法。接下来程序执行以下代码：

```
val=11;
pt.changeInt(val);
```

设置变量 val 的值为 11，并将 val 传入 changeInt 方法中。因此，程序执行流程跳转至

以下代码:

```
public void changeInt(int value){
    value=55;
}
```

val 的值传递给方法的调用参数 value,因为基本数据类型的参数传递是实际值传递,因此 value 的值为 11,但当执行语句“value=55;”后,value 的值变为 55。随后,程序跳转回 main 方法内,执行语句:

```
System.out.println("Int value is: "+val);
```

变量 value 的值是 55,但对变量 val 没有任何影响,因此 val 的值仍旧保持 11 不变。

接下来,程序执行:

```
str=new String("Hello");
pt.changeStr(str);
```

创建 String 对象,内容是 Hello,并将该对象的引用值存储在变量 str 值内。然后将变量 str 传入方法 changeStr 中。程序执行流程跳转至以下代码:

```
public void changeStr(String value){
    value=new String("World");
}
```

str 的值传递给方法的调用参数 value,因为引用数据类型的参数传递是地址值传递,因此 value 的值为引用对象的内存地址(见图 3-10(a)),但当执行语句“value=new String("World");”后,value 的值指向了新的 String 对象(见图 3-10(b))。随后,程序跳转回 main 方法内,执行语句:

```
System.out.println("Str value is: "+str);
```

变量 str 的结果仍旧是 Hello。因此控制台输出“Str value is: Hello”。

随后,程序执行以下代码:

```
pt.ptValue=101.0f;
pt.changeObjValue(pt);
```

定义 pt 的成员变量 ptValue 值为 101.0f(见图 3-11(a)),然后调用 changeObjValue 方法,把对象 pt 的值传给该方法的调用参数 ref,由于参数 ref 是 PassTest 类型,因此和对象 pt 的类型符合,传值能顺利进行,此时变量 pt 和 ref 的值均指向了堆内存中 PassTest 类的对象(见图 3-11(b))。程序执行流程跳转至以下代码:

```
public void changeObjValue(PassTest ref){
    ref.ptValue=99.0f;
}
```

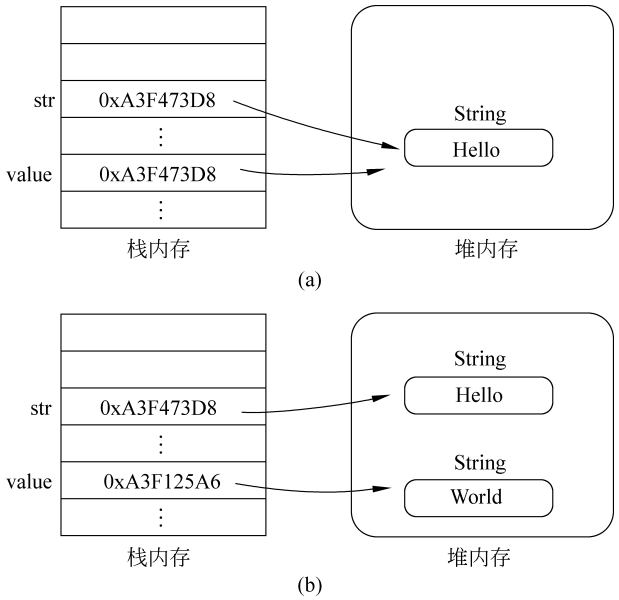

图 3-10　引用数据类型参数传递

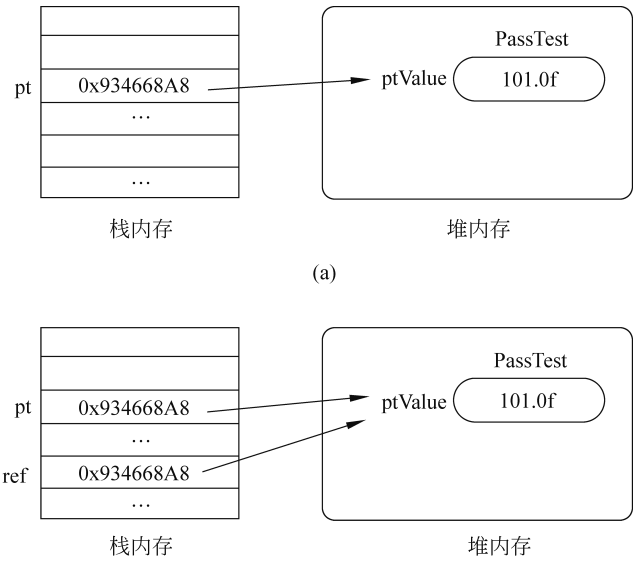

图 3-11　引用数据类型参数传递

语句“ref.ptValue=99.0f;”的作用是通过变量 ref 的引用值，改变它所指的对象的变量 ptValue 的值。由于没有新对象产生，pt 和 ref 的引用值也没有发生修改，变化的只是对象内部的成员变量，因此得到的结果如图 3-12 所示。

执行语句“System.out.println("Pt value is: "+pt.ptValue);”后，由于堆内存中的对象成员变量 ptValue 已经变为 99.0f 了，因此控制台输出“Pt value is: 99.0”。