7.8 构造方法 / 构造器

在面向对象编程中，对象的初始化是一个重要环节。当我们创建对象时，往往需要在对象创建之初就完成一些初始化操作（如属性赋值）。构造方法（又称构造器）就是专门用于对象初始化的特殊方法，本节将详细介绍构造方法的概念、语法及使用技巧。

7.8.1 需求引入：对象创建时的初始化问题

在之前的学习中，我们创建对象的流程通常是：先通过`new`关键字创建对象，再通过对象名给属性赋值。例如：

```
class Person {
    String name;
    int age;
}

// 创建对象并赋值
Person p = new Person();
p.name = "张三";
p.age = 20;
```

但在实际开发中，有时需要**在创建对象的同时直接指定属性值**（如要求创建人类对象时必须明确姓名和年龄）。此时，普通的属性赋值方式无法满足需求，必须使用构造方法。

7.8.2 构造方法的基本语法

构造方法是类中一种特殊的方法，其语法格式如下：

```
[修饰符] 类名(形参列表) {
    方法体; // 通常用于初始化对象的属性
}
```

语法说明：

1. **修饰符**：可省略（默认权限），或使用`public`、`protected`、`private`（与普通方法的访问修饰符规则一致）。
2. **方法名**：必须与**类名完全相同**（包括大小写），这是构造方法与普通方法的核心区别。
3. **返回值**：构造方法**没有返回值**，且不能写`void`（若写`void`，则成为普通方法，失去构造方法的功能）。
4. **形参列表**：与普通方法的形参规则一致，可包含 0 个或多个参数，用于接收初始化数据。
5. **调用时机**：构造方法**不能通过对象手动调用**，而是在创建对象时由`new`关键字自动触发。

7.8.3 构造方法的作用与基本使用

核心作用：

- **初始化对象属性**：在对象创建时，通过构造方法的参数为对象的属性赋值，确保对象创建后即处于可用状态。

快速入门示例：

```
public class ConstructorDemo {
    public static void main(String[] args) {
        // 创建对象时，通过构造方法直接指定姓名和年龄
        Person p = new Person("张三", 20);
        // 直接使用初始化后的对象
        System.out.println("姓名：" + p.name + "，年龄：" + p.age); // 输出：姓名：张三，年龄：20
    }
}

class Person {
    String name;
    int age;

    // 构造方法：用于初始化name和age属性
    public Person(String pName, int pAge) {
        name = pName; // 将参数值赋给成员变量
        age = pAge;
    }
}
```

执行流程解析：

1. 当执行`new Person("张三", 20)`时，`new`关键字会先在内存中为对象分配空间。
2. 自动调用与类名相同的构造方法`Person(String pName, int pAge)`，并将参数`"张三"`和`20`传入。
3. 执行构造方法的方法体，将参数值赋给对象的`name`和`age`属性。
4. 构造方法执行完毕后，对象创建完成，`new`关键字返回对象的引用（地址），赋值给变量`p`。

7.8.4 构造方法的注意事项与使用细节

构造方法的使用有诸多细节需要注意，尤其是默认构造方法的特性和构造方法的重载，这些是初学者容易出错的地方。

1. 默认构造方法：系统自动生成的无参构造器

如果一个类中**没有定义任何构造方法**，Java 编译器会自动为该类生成一个**无参构造方法**（又称默认构造器），其格式为：

```
类名() {} // 方法体为空
```

示例：

```
class Dog {
    // 未定义任何构造方法，系统自动生成默认无参构造器
}

public class Test {
    public static void main(String[] args) {
        Dog dog = new Dog(); // 调用系统生成的默认无参构造器
    }
}
```

验证：

通过`javap`命令（反编译工具）可查看默认构造器。步骤如下：

1. 编译`Test.java`生成`Dog.class`。
2. 执行

   ```
   javap Dog.class
   ```

   ，输出结果中会显示：

   ```
   class Dog {
     Dog(); // 系统自动生成的默认无参构造器
   }
   ```

2. 自定义构造方法后，默认构造器会被覆盖

如果在类中**自定义了构造方法**（无论带参还是无参），系统将**不再生成默认无参构造器**。此时若需使用无参构造器，必须**手动显式定义**。

错误示例：

```
class Cat {
    // 自定义带参构造器，覆盖了默认无参构造器
    public Cat(String name) {
        this.name = name;
    }

    String name;
}

public class Test {
    public static void main(String[] args) {
        // 编译错误：找不到无参构造器
        Cat cat = new Cat();
    }
}
```

正确示例（显式定义无参构造器）：

```
class Cat {
    String name;

    // 自定义带参构造器
    public Cat(String name) {
        this.name = name;
    }

    // 显式定义无参构造器（解决上述错误）
    public Cat() {
        // 可根据需求添加初始化逻辑，如默认值
        this.name = "无名猫";
    }
}

public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat(); // 调用无参构造器，name默认值为"无名猫"
        Cat cat2 = new Cat("橘猫"); // 调用带参构造器
    }
}
```

3. 构造方法的重载

与普通方法一样，构造方法也支持重载（Overload），即一个类中可以定义多个构造方法，只要它们的**形参列表不同**（参数个数、类型或顺序不同）。

构造方法重载的目的是：为对象初始化提供**多种灵活的方式**（如允许创建对象时传入不同的参数组合）。

示例：

```
class Student {
    String name;
    int age;
    String id; // 学号

    // 无参构造器：初始化默认值
    public Student() {
        this.name = "未知";
        this.age = 0;
        this.id = "000000";
    }

    // 带1个参数的构造器：仅初始化姓名
    public Student(String name) {
        this.name = name;
        this.age = 0;
        this.id = "000000";
    }

    // 带3个参数的构造器：初始化所有属性
```

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

// 使用不同构造器创建对象
public class Test {
    public static void main(String[] args) {
        Student s1 = new Student(); // 无参构造器
        Student s2 = new Student("李四"); // 1个参数
        Student s3 = new Student("王五", 18, "2023001"); // 3个参数
    }
}
```

4. 构造方法中调用其他构造方法（`this`关键字的使用）

当一个类中存在多个构造方法时，可能会出现重复的初始化逻辑。为了简化代码，可在一个构造方法中通过`this(参数)`调用另一个构造方法。

语法：

```
this(实参列表); // 必须放在构造方法的第一行
```

```
class Teacher {
    String name;
    int age;

    // 无参构造器：调用带参构造器，设置默认值
    public Teacher() {
        this("未知姓名", 30); // 调用带2个参数的构造器
    }

    // 带2个参数的构造器：初始化所有属性
    public Teacher(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

说明：

- `this(参数)`必须放在构造方法的**第一行**（否则编译报错），因为初始化操作需要按顺序执行。
- 不能在两个构造方法中互相调用（如 A 调用 B，B 又调用 A），会导致无限循环。

7.8.5 课堂练习：构造方法的综合应用

需求：定义`Person`类，包含以下两个构造方法：

1. 无参构造器：将所有人的`age`属性初始值设为 18。
2. 带两个参数（`pName`、`pAge`）的构造器：创建对象时直接初始化`name`和`age`属性。

```
```

```java
public class ConstructorExercise {
    public static void main(String[] args) {
        // 使用无参构造器创建对象
        Person p1 = new Person();
        System.out.println("p1: 姓名=" + p1.name + "，年龄=" + p1.age); // 输出：
p1: 姓名=null，年龄=18

        // 使用带参构造器创建对象
        Person p2 = new Person("张三", 25);
        System.out.println("p2: 姓名=" + p2.name + "，年龄=" + p2.age); // 输出：
p2: 姓名=张三，年龄=25
    }
}

class Person {
    String name; // 默认值为null
    int age;     // 默认值为0

    // 1. 无参构造器：将age初始化为18
    public Person() {
        this.age = 18; // 显式设置年龄初始值
    }

    // 2. 带参构造器：初始化name和age
    public Person(String pName, int pAge) {
        this.name = pName; // 为姓名赋值
        this.age = pAge;   // 为年龄赋值
    }
}
```

代码说明：

- 无参构造器`Person()`中，通过`this.age = 18`强制将年龄初始化为 18，覆盖了默认值 0。
- 带参构造器`Person(String pName, int pAge)`接收外部传入的参数，直接为`name`和`age`赋值，实现了创建对象时的灵活初始化。

7.8.6 小结

1. 构造方法是专门用于对象初始化的特殊方法，在创建对象时由`new`关键字自动调用。
2. 构造方法的名称必须与类名相同，且没有返回值。
3. 若类中未定义构造方法，系统会自动生成默认无参构造器；若自定义了构造方法，默认构造器会被覆盖，需手动显式定义才能使用。
4. 构造方法支持重载，可通过不同的参数列表提供多种初始化方式。
5. 可通过`this(参数)`在一个构造方法中调用另一个构造方法，简化代码复用。

课后练习

一、选择题（10 道）

1. 下列关于构造方法的说法，正确的是（ ）
   A. 构造方法必须有返回值类型
   B. 构造方法的名称可以与类名不同
   C. 构造方法在创建对象时被自动调用

D. 构造方法不能被重载

2. 若一个类中未定义任何构造方法，Java 编译器会自动生成的构造器是（ ）
   A. 带一个参数的构造器
   B. 无参构造器
   C. 带两个参数的构造器
   D. 不会生成任何构造器

3. 以下构造方法的声明中，正确的是（ ）
   A. public void Person() {}
   B. public Person(String name) {}
   C. public Person(int age) { return age; }
   D. Person(String name, int age) { void; }

4. 关于构造方法重载的条件，下列说法错误的是（ ）
   A. 方法名必须相同
   B. 参数列表必须不同
   C. 返回值类型必须不同
   D. 可以有不同的访问修饰符

5. 若类 `Student` 中定义了如下构造方法

```
public Student(String name) { ... }
```

   则下列创建对象的方式中，会导致编译错误的是（ ）
   A. Student s = new Student ("张三");
   B. Student s = new Student();
   C. Student s = new Student ("李四", 18);
   D. 以上都不会报错

6. 构造方法与普通方法的主要区别是（ ）
   A. 构造方法可以有返回值，普通方法不能
   B. 构造方法的名称必须与类名相同，普通方法可以不同
   C. 普通方法可以被 `new` 关键字调用，构造方法不能
   D. 构造方法不能被重载，普通方法可以

7. 下列代码中，`Person` 类的构造方法被调用的次数是（ ）

```
class Person {
    public Person() {}
    public Person(String name) {}
}
public class Test {
    public static void main(String[] args) {
        Person p1 = new Person();
        Person p2 = new Person("张三");
        Person p3 = p2;
    }
}
```

   A. 1 次
   B. 2 次
   C. 3 次
   D. 0 次

8. 关于`this`关键字在构造方法中的使用，下列说法正确的是（ ）
   A. `this()`可以调用本类的其他构造方法，必须放在构造方法的第一行
   B. `this()`可以在普通方法中调用构造方法
   C. `this`关键字可以在静态方法中使用
   D. `this`关键字不能区分成员变量和局部变量
9. 若要在无参构造器中调用带参构造器`Person(String name, int age)`，正确的写法是（ ）
   A. Person ("默认姓名", 18);
   B. this.Person ("默认姓名", 18);
   C. this ("默认姓名", 18);
   D. super ("默认姓名", 18);
10. 下列关于默认构造器的说法，错误的是（ ）
    A. 默认构造器是无参的
    B. 若自定义了构造器，默认构造器仍然存在
    C. 默认构造器由编译器自动生成
    D. 默认构造器的方法体为空

二、填空题（10 道）

1. 构造方法的名称必须与完全相同。
2. 构造方法没有，且不能写`void`关键字。
3. 当一个类中定义了自定义构造器后，系统默认的会被覆盖。
4. 构造方法的主要作用是完成对的初始化。
5. 在创建对象时，构造方法由关键字自动调用，不能通过对象手动调用。
6. 构造方法重载的核心是不同。
7. 若要在构造方法中调用本类的另一个构造方法，需使用关键字。
8. 若类`Dog`中只定义了带参构造器`public Dog(String name)`，则创建无参对象`new Dog()`时会出现错误。
9. 无参构造器的语法格式是：`[修饰符] ____() { ... }`。
10. 构造方法的访问修饰符可以是`public`、`protected`、`private`或（默认权限）。

三、判断题（10 道）

1. 构造方法可以被`static`关键字修饰。（ ）
2. 一个类中可以有多个构造方法，只要它们的参数列表不同。（ ）
3. 构造方法的返回值类型可以是`void`。（ ）
4. 若类中没有定义构造方法，系统会自动生成一个默认无参构造器。（ ）
5. 构造方法不能被重载。（ ）
6. 调用`new Person()`时，会先执行`Person`类的构造方法，再为对象分配内存。（ ）
7. 子类可以继承父类的构造方法。（ ）
8. 构造方法中的`this()`调用必须放在方法体的第一行。（ ）
9. 自定义构造器后，若要使用无参构造器，必须显式定义。（ ）
10. 构造方法可以像普通方法一样被对象调用（如`p.Person()`）。（ ）

四、简答题（10 道）

1. 简述构造方法的作用和基本语法特点。
2. 构造方法与普通方法的区别有哪些？
3. 什么是构造方法的重载？重载的目的是什么？
4. 为什么自定义构造器后，创建无参对象会出现编译错误？如何解决？
5. 简述默认构造器的生成规则。

6. 如何在一个构造方法中调用本类的另一个构造方法？需要注意什么？
7. 构造方法的访问修饰符为`private`时，有什么作用？（提示：单例模式）
8. 若类`A`有两个构造器：`A()`和`A(int a)`，执行`A a = new A(10)`时，哪个构造器会被调用？
9. 为什么构造方法不能有返回值？
10. 举例说明构造方法在实际开发中的应用场景。

五、编程题（10 道）

1. 定义`Car`类，包含属性`brand`（品牌）和`price`（价格），设计一个带参构造器，在创建对象时初始化这两个属性，然后创建对象并打印属性值。
2. 为`Car`类添加无参构造器，将`brand`默认值设为“未知品牌”，`price`默认值设为 0，分别使用无参和带参构造器创建对象并测试。
3. 定义`Student`类，包含属性`name`（姓名）、`age`（年龄）、`id`（学号），设计三个构造器：
   - 无参构造器：初始化默认值（姓名“未知”，年龄 0，学号“000”）
   - 单参构造器：仅初始化姓名
   - 三参构造器：初始化所有属性
     并通过创建对象验证各构造器的效果。
4. 编写程序，定义`Book`类，要求：
   - 包含属性`title`（书名）和`page`（页数）
   - 无参构造器：将页数默认设为 100
   - 带参构造器：接收书名和页数，并判断页数是否为正数（若为负数则设为 100）
     创建对象测试构造器的功能。
5. 定义`Person`类，使用构造方法实现以下需求：
   - 无参构造器：将年龄初始化为 18
   - 带参构造器：接收姓名和年龄，若年龄小于 0 则设为 0
     并验证构造器的初始化逻辑。
6. 在构造器中使用`this()`调用，优化第 5 题的`Person`类：无参构造器调用带参构造器，设置默认姓名为“张三”和年龄 18。
7. 定义`Circle`类，包含属性`radius`（半径），设计构造器：
   - 无参构造器：半径默认值为 1
   - 带参构造器：接收半径，若半径小于 0 则抛出异常（提示：`throw new IllegalArgumentException("半径不能为负")`）
     测试构造器的异常处理。
8. 定义`Teacher`类，包含属性`name`和`subject`（学科），要求：
   - 必须通过构造器初始化`name`（不允许创建对象时不指定姓名）
   - 可选初始化`subject`（若不指定则默认“语文”）
     设计构造器并测试。
9. 编写程序，创建`Animal`类，包含属性`name`，通过构造器实现：
   - 无参构造器调用带参构造器，默认姓名为“动物”
   - 带参构造器设置姓名
     验证`this()`的调用效果。
10. 定义`Phone`类，包含属性`brand`和`price`，设计构造器并在构造器中打印“手机初始化完成”，创建对象观察构造器的执行时机。