7.10 this 关键字

在面向对象编程中，当方法或构造器中的局部变量与类的属性同名时，如何区分它们？当需要在类的方法中引用当前对象时，又该如何操作？`this`关键字正是为解决这些问题而存在的。本节将从实际案例出发，逐步解析`this`关键字的含义、用法及注意事项。

7.10.1 从案例看`this`的必要性

我们先通过一个具体案例，观察未使用`this`时可能出现的问题，进而理解`this`的核心作用。

案例代码：未使用`this`的困惑

```
public class This01 {
    public static void main(String[] args) {
        Dog dog1 = new Dog("大壮", 3);
        System.out.println("dog1 的 hashCode=" + dog1.hashCode());
        dog1.info();  // 调用信息方法

        System.out.println("============");

        Dog dog2 = new Dog("大黄", 2);
        System.out.println("dog2 的 hashCode=" + dog2.hashCode());
        dog2.info();  // 调用信息方法
    }
}

class Dog {
    String name;  // 狗的名字
    int age;      // 狗的年龄

    // 构造器：参数名与属性名相同
    public Dog(String name, int age) {
        // 问题：这里的name和age是局部变量还是类的属性？
        name = name;  // 局部变量赋值给局部变量（无效操作）
        age = age;    // 局部变量赋值给局部变量（无效操作）
    }

    // 输出狗的信息
    public void info() {
        System.out.println("name=" + name + ", age=" + age);
    }
}
```

运行结果与问题分析

上述代码运行后，`info()`方法会输出`name=null, age=0`，显然与预期不符。问题出在构造器中：

- 构造器的参数名`name`和`age`与类的属性名完全相同；
- 根据“变量作用域优先原则”，构造器中直接使用的`name`和`age`会被识别为**局部变量**（参数），而非类的属性；

- 因此，`name = name`实际是“局部变量给局部变量赋值”，类的属性并未被初始化，仍保持默认值（`null`和`0`）。

如何让构造器明确区分“局部变量”和“类的属性”？这就需要`this`关键字。

使用`this`解决命名冲突

修改上述构造器，通过`this`关键字指定类的属性：

```
class Dog {
    String name;
    int age;
    // 使用this区分属性和局部变量
    public Dog(String name, int age) {
        this.name = name;  // this.name 表示类的属性name
        this.age = age;    // this.age 表示类的属性age
        System.out.println("this 的 hashCode=" + this.hashCode());  // 打印this的哈希值
    }

    public void info() {
        System.out.println("this 的 hashCode=" + this.hashCode());  // 打印this的哈希值
        System.out.println("name=" + name + ", age=" + age);
    }
}
```

运行结果与结论

再次运行程序，输出如下：

```
dog1 的 hashCode=356573597
this 的 hashCode=356573597
this 的 hashCode=356573597
name=大壮, age=3
============
dog2 的 hashCode=1735600054
this 的 hashCode=1735600054
this 的 hashCode=1735600054
name=大黄, age=2
```

结论：

- `this`关键字在构造器和方法中代表**当前对象**（即调用该方法 / 构造器的对象）；
- `this.name`明确指向类的属性`name`，解决了与局部变量的命名冲突；
- `this`的哈希值与对象的哈希值完全一致，证明`this`就是当前对象的引用。

7.10.2 深入理解`this`的本质

`this`的本质是**当前对象的引用**，它在以下场景中发挥核心作用：

1. **区分属性与局部变量**
   当方法或构造器的局部变量与类的属性同名时，用`this.属性名`表示类的属性，用`变量名`表示局部变量。

示例：

```
class Student {
    String name;

    // 方法参数与属性同名
    public void setName(String name) {
        this.name = name;  // this.name 是属性，name 是参数
    }
}
```

2. **在方法中引用当前对象的其他方法**
   可以通过`this.方法名()`调用当前对象的其他方法（通常可省略`this.`，但明确使用能提高可读性）。

   示例：

```
class Teacher {
    public void teach() {
        System.out.println("正在教学");
    }

    public void work() {
        this.teach();  // 调用当前对象的teach()方法，this.可省略
        System.out.println("备课");
    }
}
```

3. **在构造器中调用其他构造器**
   通过`this(参数列表)`在一个构造器中调用同一类的其他构造器，简化代码复用。

   示例：

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

    // 无参构造器
    public Person() {
        this("未知", 0);  // 调用有参构造器，必须放在第一行
    }

    // 有参构造器
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

4. **作为方法的返回值**
   在某些场景下（如链式调用），方法可返回`this`，表示返回当前对象。

   示例：

```
class Builder {
    String result;

    public Builder add(String part) {
        result += part;
        return this;  // 返回当前对象，支持链式调用
    }
}

// 链式调用
Builder b = new Builder();
b.add("a").add("b").add("c");  // 连续调用add()方法
```

7.10.3 `this`的注意事项和使用细节

1. **`this`只能在类的方法或构造器中使用**
   `this`代表当前对象，而对象是在类实例化后才存在的，因此不能在静态方法（`static`）或类的静态代码块中使用`this`（静态成员属于类，不依赖对象）。

   错误示例：

```
class Test {
    static {
        System.out.println(this);  // 编译错误：静态代码块中不能使用this
    }

    public static void method() {
        System.out.println(this);  // 编译错误：静态方法中不能使用this
    }
}
```

2. **`this(参数列表)`只能在构造器中使用，且必须放在第一行**
   - 构造器中调用其他构造器时，`this(...)`必须是构造器的第一条语句，否则会编译错误；
   - 不能在构造器中通过`this(...)`相互调用（会导致无限递归）。

   错误示例：

```
class A {
    public A() {
        this(1);  // 正确：调用有参构造器，放在第一行
    }

    public A(int n) {
        this();  // 错误：相互调用导致无限递归
    }
}
```

3. **`this`不能为`null`**
   `this`代表当前对象，而对象一定是已实例化的，因此`this`永远不会为`null`。

7.10.4 课堂案例：使用`this`实现对象比较

案例需求

定义 `Person` 类，包含 `name` 和 `age` 属性，提供 `compareTo` 方法用于判断当前对象与另一个 `Person` 对象是否相等（名字和年龄完全相同则返回 `true`，否则返回 `false`），并通过测试类验证。

代码实现

```
public class TestPerson {
    public static void main(String[] args) {
        Person p1 = new Person("mary", 20);
        Person p2 = new Person("mary", 20);
        Person p3 = new Person("jack", 20);
        System.out.println("p1 和 p2 比较：" + p1.compareTo(p2));  // true
        System.out.println("p1 和 p3 比较：" + p1.compareTo(p3));  // false
    }
}

class Person {
    String name;
    int age;

    // 构造器：初始化属性
    public Person(String name, int age) {
        this.name = name;  // 区分属性和参数
        this.age = age;
    }

    // 比较当前对象与另一个对象是否相等
    public boolean compareTo(Person p) {
        // this 代表当前对象（调用方法的对象），p 代表参数对象
        return this.name.equals(p.name) && this.age == p.age;
    }
}
```

代码解析

- `compareTo` 方法中，`this` 代表调用该方法的对象（如 `p1.compareTo(p2)` 中，`this` 就是 `p1`）；
- 通过 `this.name` 和 `this.age` 访问当前对象的属性，通过 `p.name` 和 `p.age` 访问参数对象的属性；
- 当两个对象的 `name` 和 `age` 完全相同时，返回 `true`，否则返回 `false`。

课后练习

一、选择题（10 题）

1. 下列关于 `this` 关键字的说法，正确的是（ ）
   A. `this` 可以在静态方法中使用
   B. `this` 代表当前对象的引用
   C. `this` 是一个关键字，必须显式声明才能使用
   D. `this` 可以指向类的静态属性

2. 当构造器的参数与类的属性同名时，为属性赋值应使用（）
   A. 直接赋值（如`name = name`）
   B. `super.属性名 = 参数名`
   C. `this.属性名 = 参数名`
   D. 无需特殊处理，编译器会自动区分

3. 在成员方法中，`this.方法名()`的作用是（）
   A. 调用父类的方法
   B. 调用当前对象的其他成员方法
   C. 调用静态方法
   D. 调用其他类的方法

4. 关于`this(参数列表)`的用法，错误的是（）
   A. 只能在构造器中使用
   B. 可以在构造器中调用其他类的构造器
   C. 必须放在构造器的第一行
   D. 目的是复用其他构造器的代码

5. 下列代码中，`this`的哈希值与哪个对象的哈希值一致（）

```
class A {
    public A() { System.out.println(this.hashCode()); }
}
A a = new A();
```

   A. 类`A`的`Class`对象
   B. 引用变量`a`
   C. 对象`a`所指向的实例
   D. 以上都不一致

6. 下列场景中，必须使用`this`关键字的是（）
   A. 成员方法中调用其他成员方法
   B. 构造器中参数与属性同名时为属性赋值
   C. 成员方法中访问本类的静态属性
   D. 无参构造器中调用有参构造器

7. 静态方法中不能使用`this`的原因是（）
   A. 静态方法没有参数
   B. 静态方法属于类，不依赖具体对象
   C. `this`在静态方法中会导致编译错误
   D. 静态方法只能访问静态属性

8. 下列代码的运行结果是（）

```
class B {
    int x;
    public B(int x) { this.x = x; }
    public void print() { System.out.println(x); }
}
B b = new B(10);
b.print();
```

A. 0
B. 10
C. null
D. 编译错误

9. 关于`this`作为方法返回值，说法正确的是（）
A. 方法返回`this`时，返回的是当前对象的副本
B. 返回`this`可以实现方法的链式调用
C. 只有静态方法才能返回`this`
D. 返回`this`会导致对象被销毁

10. 下列代码中存在编译错误的是（）

```
A. class C {
     public C() { this(5); }
     public C(int x) {}
   }
B. class C {
     public C() {
         System.out.println("无参");
         this(5); // 错误行
     }
     public C(int x) {}
   }
C. class C {
     public C() { this(); } // 错误行
     public C(int x) {}
   }
D. class C {
     public C() { this("a"); }
     public C(String s) {}
   }
```

二、填空题（10 题）

1. `this`关键字的本质是__。
2. 在构造器中，`this(参数列表)`的作用是__。
3. 当成员方法的局部变量与类的属性同名时，用__表示类的属性。
4. `this`关键字不能在__方法中使用，因为这类方法属于类而非对象。
5. 方法返回`this`的目的通常是__，例如`builder.add().remove()`。
6. `this`在__和__中自动存在，代表当前对象。
7. 构造器中调用其他构造器时，`this(...)`必须放在__，否则会编译错误。
8. 若一个方法中未出现与属性同名的局部变量，则访问属性时`this.`__（可以 / 必须）省略。
9. `this`关键字不能指向__（类 / 对象），只能指向具体的实例。
10. 代码`obj.method()`中，方法`method`内部的`this`指向__。

三、判断题（10 题）

1. `this`可以在类的静态代码块中使用。（）
2. `this`关键字必须显式写出才能使用，编译器不会自动添加。（）
3. 一个构造器中可以通过`this(...)`调用多个其他构造器。（）
4. `this`的哈希值与当前对象的哈希值相同。（）

5. 在成员方法中，`this.属性名`与直接写`属性名`的效果完全相同（无同名变量时）。（）
6. `this`可以作为参数传递给其他方法。（）
7. 构造器中，`this(...)`可以放在任意位置，只要在方法体内部即可。（）
8. `this`代表当前对象，因此可以通过`this`修改对象的引用（如`this = new Object()`）。（）
9. 非静态成员方法中，`this`一定不为`null`。（）
10. 当属性与局部变量不同名时，使用`this.属性名`是多余的。（）

四、简答题（10 题）

1. 简述`this`关键字的主要作用。
2. 为什么静态方法中不能使用`this`？
3. 构造器中`this(参数列表)`的使用规则是什么？
4. 当属性与局部变量同名时，`this`如何区分两者？
5. 举例说明`this`作为方法返回值的应用场景（如链式调用）。
6. `this`与对象的引用变量有什么关系？
7. 成员方法中，`this.方法名()`与直接写`方法名()`的区别是什么？
8. 为什么`this(...)`在构造器中必须放在第一行？
9. 简述`this`关键字在面向对象编程中的意义。
10. 下列代码中，this分别代表什么？

```
class Student {
    String name;
    public Student(String name) { this.name = name; }
    public void print() { System.out.println(this.name); }
}
Student s1 = new Student("张三");
Student s2 = new Student("李四");
s1.print();
s2.print();
```

五、编程题（10 题）

1. 定义一个`Phone`类，包含属性`brand`和`price`，构造器参数与属性同名，使用`this`为属性赋值，并定义`show()`方法打印属性值。创建对象测试效果。
2. 定义一个`Calculator`类，包含`int result`属性，提供`add(int num)`、`subtract(int num)`方法，方法内部用`this`访问`result`，并返回`this`实现链式调用。测试`calculator.add(5).subtract(3)`的结果。
3. 定义一个`Person`类，提供无参构造器和有参构造器（`String name, int age`），在无参构造器中通过`this(...)`调用有参构造器，为属性设置默认值（"未知"，0）。
4. 编写代码证明`this`代表当前对象：在构造器和成员方法中打印`this.hashCode()`，并与对象的`hashCode()`对比。
5. 定义一个`Book`类，属性`title`和`price`，提供`setTitle(String title)`和`setPrice(double price)`方法，参数与属性同名，使用`this`赋值。
6. 定义一个`Point`类，包含`x`和`y`属性，提供`move(int dx, int dy)`方法，用`this`修改`x`和`y`（`this.x += dx`），并返回`this`。测试链式调用`point.move(2,3).move(1,1)`。
7. 编写代码演示：静态方法中使用`this`会导致编译错误。

8. 定义一个`Dog`类，包含`name`属性，提供`rename(String name)`方法，用`this`区分属性和参数，将属性修改为新名称。
9. 定义一个`Rectangle`类，提供两个构造器：`Rectangle()`和`Rectangle(int width, int height)`，在无参构造器中用`this(10, 20)`调用有参构造器，设置默认宽高。
10. 定义一个`User`类，属性`username`和`password`，提供`equals(User other)`方法，用`this`比较当前对象与`other`的`username`和`password`是否完全相同，返回布尔值。