

# 🗨 继承

## # 基本概念

继承是面向对象的三大特征之一。

继承是软件可重用性的一种表现，新类可以在不增加自身代码的情况下，通过从现有类中继承其属性和方法，来实现充实自身内容，这种现象或行为就称为继承。此时新类称为子类，现有类称为父类。继承最基本的作用就是使得代码可以重用，增加软件的可扩展性。

**Java类中只支持单继承，即一个类只能有一个直接父类。**

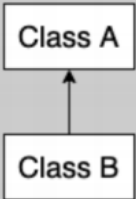
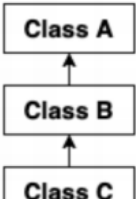
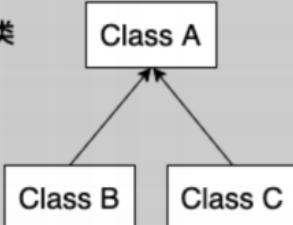
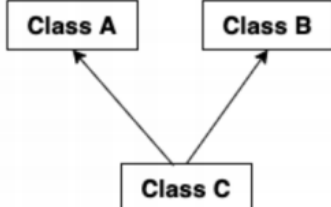
继承的语法规则如下：

```
1 [访问修饰符] class 类名 extends 父类名{
2
3 }
4 // 访问修饰符：public【项目任何地方都可以访问类】 缺省【当前包中】
5 // 访问修饰符：
6 // public : 项目任何地方都可以访问
7 // protected : 同包下任意类可以访问【其他包子类也可以访问 static
8 // private : 只能自身访问
9 // package-access : 同包下任意类可以访问
```

## 📖 Java中继承的一些基本概念：

1. 子类继承父类的非私有属性和方法。子类可以使用父类的属性和方法，不需要写相同的代码。
2. 子类可以添加自己的属性和方法。增加代码的灵活性、可扩展性。
3. 子类的构造方法可以调用父类的构造方法。子类构造方法中使用 `super` 关键字就可以调用父类的构造方法，从而初始化父类的属性。
4. Object类是所有类的根类。每个类都是Object类的子类，因此可以使用Object类中定义的一些通用方法，如`equals()`、`hashCode()`、`toString()`等。

5. Java中的类只支持单一继承，<sup>王衡</sup>每个类只能有一个父类（直接父类）。
6. 子类可以成为其他类的父类，从而建立多级继承关系。但是，过多的继承可能会导致代码难以理解和维护。

|           |  |   |
|-----------|--|---|
| 单继承       |  <pre>graph BT; B[Class B] --&gt; A[Class A]</pre>                        | <pre>public class A {<br/>    .....<br/>}<br/>public class B extends A {<br/>    .....<br/>}</pre>                                |
| 多重继承      |  <pre>graph BT; C[Class C] --&gt; B[Class B]; B --&gt; A[Class A]</pre>   | <pre>public class A {.....}<br/>public class B extends A {.....}<br/>public class C extends B {.....}</pre>                       |
| 不同类继承同一个类 |  <pre>graph BT; B[Class B] --&gt; A[Class A]; C[Class C] --&gt; A</pre>  | <pre>public class A {.....}<br/>public class B extends A {.....}<br/>public class C extends A {.....}</pre>                       |
| 多继承（不支持）  |  <pre>graph BT; C[Class C] --&gt; A[Class A]; C --&gt; B[Class B]</pre> | <pre>public class A {.....}<br/>public class B {.....}<br/>public class C extends A, B {<br/>    .....<br/>} // Java 不支持多继承</pre> |

继承是java的三大特征之一

继承就是子类继承父类的特征和行为，来实现充实自身内容

格式：使用extends 关键字申明一个类是从另一个类继承而来

现有类称为父类、超类、基类，新类称为子类、扩展类

继承的作用：代码复用，增加软件的可扩展性

继承分为单继承、多继承，java中类只支持单继承，也就是一个类只能有一个直接父类

## 继承的应用

Java中继承是一种面向对象编程的核心概念，它可以让子类继承父类的属性和方法，并且还可以在此基础上进行扩展和重写，提高了代码的复用性和可维护性。

```
1 package com.kfm.pm;
2 public class Pet {
3     private String name;
4     private int age;
5     public String color;
6     protected double weight;
7     public Pet(){}
8     public Pet(String name, int age, String color, double
weight,String voice) {
9         this.name = name;
10        this.age = age;
11        this.color = color;
12        this.weight = weight;
13        this.voice = voice;
14    }
15    String voice;
16    private void privateMethod () {
17        System.out.println("这是一个私有方法");
18    }
19    public void eat() {
20        System.out.println("这是一个eat方法");
21    }
22    protected void sleep() {
23        System.out.println("这是一个protected方法");
24    }
25    void packageAccessMethod() {
26        System.out.println("这是一个packageAccess方法");
27    }
28 }
```

## 使用继承和重写实现子类

使用继承定义子类

```
1 package com.kfm.pm;
2 public class Dog extends Pet {
3     // Dog is a Pet Dog 是子类 Pet 是父类 子类是对于父类的扩展
4     private String lookHouse;
5     public Dog() {
6
7     }
8     public Dog(String name, int age, String color, double
weight,String voice, String lookHouse) {
9         super(name, age, color, weight, voice);
10        this.lookHouse = lookHouse;
```

```
11     }
12     public void lookingHouse() {
13         System.out.println("狗狗看家");
14     }
15 }
```

### 使用super关键字调用父类成员

当需要在子类中调用父类的构造方法时，可以使用super关键字调用。当函数参数或函数中的局部变量和成员变量同名时，成员变量会被遮蔽，此时若要访问成员变量则需要使用“this.成员变量名”的方式来引用成员变量。super关键字和this关键字的作用类似，都是将被隐藏了的成员变量、成员方法变得可见、可用，也就是说，用来引用被隐藏的成员变量或成员方法。不过super用在子类中，目的只有一个就是访问直接父类中被隐藏的内容，进一步提高代码的重用性和灵活性。super关键字不仅可以访问父类的构造方法，还可以访问父类的成员，包括父类的字段、普通方法等。

通过super访问父类成员语法规则如下：

调用父类构造方法： `super([实参列表]);`

调用父类属性和方法： `super.<父类字段名/方法名>`

1. super只能出现在子类（子类的实例方法或构造方法）中，而不是其他位置。
2. super用于访问父类成员，如父类的属性、方法、构造方法。
3. 具有访问权限的限制，如无法通过super访问父类private成员。
4. super用在子类构造函数中时，必须是子类构造函数的第一行代码。

## 方法重写

在子类中可以根据需求对从父类继承的方法进行重新编写，这称为方法的重写或方法的覆盖（overriding）。方法重写必须满足如下要求：

- 在继承关系中。
- 重写方法与被重写方法必须有 相同的方法名称 。
- 重写方法与被重写方法必须有 相同的参数列表 。
- 重写方法的返回值类型必须和被重写方法的 返回值类型相同 或是其子类。
- 重写方法 不能缩小 被重写方法的 访问权限 。

- 不能 用子类的非静态方法重写（覆盖）父类的静态方法，否则编译报错。
- 不能 重写父类中的最终方法。
- 不能 用子类的静态方法重写父类的实例方法。

方法重载: 同一个类 同名 参数列表不同（类型、数量、顺序不同）

方法重写: 发生在继承关系中，不同类，方法名相同，参数列表相同，返回值类型一致，访问修饰符权限相同或变大，抛出异常相同缩小。

## 方法隐藏

父类和子类拥有相同名字的属性或者方法（方法隐藏只有一种形式，就是父类和子类存在签名相同的静态方法）时，父类的同名的属性或者方法形式上不见了，实际是还是存在的。

隐藏是对于静态方法和成员变量（静态变量和实例变量）而言的：

- 当发生隐藏的时候，声明类型是什么类，就调用对应类的方法，而不会发生动态绑定
- 属性只能被隐藏，不能被覆盖
- 变量可以交叉隐藏：子类实例变量/静态变量可以隐藏父类的实例/静态变量
- 不能用子类的静态方法隐藏父类中的非静态方法，否则编译报错。

## 子类实例化过程

A extends B 实例化过程：

- 1 先有父类再有子类
- 2
- 3 加载类 A 时，要先看类 B 是否被加载了(类 B 是否是第一次使用)，如果是第一次使用
- 4 类 B 则先加载类 B 到内存，
- 5
- 6 进行静态初始化(static 变量， static初始化器 <clinit>)，此时类 B 被加载到了内存中。
- 7
- 8 然后再加载子类 A，进行静态初始化(static 变量， static初始化器)。此时类 A 也被加载到内存中。
- 9

- 10 ps: 类只有在第一次使用时进行加载及静态初始化
- 11
- 12 实例化 (new 对象) A: 现有父类对象再创建子类对象
- 13 类 B 的实例初始化 (实例初始化器, 构造 <init>), 此时父类对象就创建完成了。
- 14 类 A 的实例初始化 (实例初始化器, 构造 <init>), 此时对象就创建完成了。