

Java面向对象多态

多态概述

Java 多态包括以下三种方式

方法重写 (Override)

向上转型 (Upcasting)

实现多态

多态概述

Java 多态是指同一种类型的对象，在不同的情况下有着不同的状态和行为。它是基于继承、重写和向上转型等特性实现的，多态是面向对象编程的三大特征之一，其他两个分别是封装和继承。

Java 多态包括以下三种方式

方法重写 (Override)

子类可以对从父类继承的方法进行重写，以便根据子类的需要来覆盖掉父类中的方法实现。这样在调用时，可以根据对象的不同来执行对应的方法。

例如：

```
1 class Animal {  
2     public void move() {  
3         System.out.println("动物移动");  
4     }  
5 }  
6  
7 class Cat extends Animal {  
8     @Override  
9     public void move() {  
10        System.out.println("猫在走路");  
11    }  
12 }  
13  
14 public class Test {  
15     public static void main(String[] args) {  
16         Animal animal = new Animal();  
17         Animal cat = new Cat();  
18         animal.move();  
19         cat.move();  
20     }  
21 }  
22
```

向上转型 (Upcasting)

将子类对象转换成父类引用，这样就可以使用父类中定义的方法和属性，但不能访问子类独有的方法和属性。这种特征使得可以将子类对象作为父类的参数进行传递，提高代码的灵活性。

例如：

```
1 class Animal {
2     public void move() {
3         System.out.println("动物移动");
4     }
5 }
6
7 class Cat extends Animal {
8     @Override
9     public void move() {
10        System.out.println("猫在走路");
11    }
12    public void catchMouse() {
13        System.out.println("猫在抓老鼠");
14    }
15 }
16
17 public class Test {
18     public static void main(String[] args) {
19         Animal animal = new Cat();
20         animal.move(); // 调用子类中的方法
21         //animal.catchMouse(); // 编译错误，不能访问子类独有的方法
22     }
23 }
```

实现多态

通过父类引用指向子类对象，可以实现多态性，使得同一个方法调用可以在不同的对象上具有不同的行为。这种特征使得程序更加灵活，可以根据需要来确定对象的类型。

例如：

```
1 class Animal {
2     public void move() {
3         System.out.println("动物移动");
4     }
5 }
6
7 class Cat extends Animal {
8     @Override
9     public void move() {
10        System.out.println("猫在走路");
11    }
12 }
13
14 class Dog extends Animal {
15     @Override
16     public void move() {
17        System.out.println("狗在跑步");
18    }
19 }
20
21 public class Test {
22     public static void main(String[] args) {
23         Animal animal1 = new Cat();
24         Animal animal2 = new Dog();
25         animal1.move(); // 同一种方法调用不同的对象
26         animal2.move(); // 同一种方法调用不同的对象
27     }
28 }
29
```