

多态的理解

不请自来，C++的多态不太清楚。java比较熟悉，就说说咯。

用一句话概括就是：**事物在运行过程中存在不同的状态**。先以教科书般举例说明,下文再举一个花木兰替父从军的例子帮助大家理解。多态的存在有三个前提:

1.要有继承关系

2.子类要重写父类的方法

3.父类引用指向子类对,

但是其中又有很多细节需要注意。首先我们定义两个类，一个父类Animal，一个子类Cat。

父类Animal

```
class Animal {
    int num = 10;
    static int age = 20;
    public void eat() {
        System.out.println("动物吃饭");
    }

    public static void sleep() {
        System.out.println("动物在睡觉");
    }

    public void run(){
        System.out.println("动物在奔跑");
    }
}
```

子类Cat

```
class Cat extends Animal {
    int num = 80;
    static int age = 90;
    String name = "tomCat";
    public void eat() {
        System.out.println("猫吃饭");
    }
    public static void sleep() {
        System.out.println("猫在睡觉");
    }
    public void catchMouse() {
        System.out.println("猫在抓老鼠");
    }
}
```

测试类Demo_Test1

```
class Demo_Test1 {  
    public static void main(String[] args) {  
        Animal am = new Cat();  
        am.eat();  
        am.sleep();  
        am.run();  
        //am.catchMouse();这里先注释掉，等会会说明  
        //System.out.println(am.name);这里先注释，待会说明  
        System.out.println(am.num);  
        System.out.println(am.age);  
    }  
}
```

以上的三段代码充分体现了多态的三个前提，即：

1、存在继承关系

[Cat类](#)继承了Animal类

2、子类要重写父类的方法

子类重写(override)了父类的两个成员方法eat(), sleep()。其中eat()是非静态的，sleep()是静态的 (static) 。

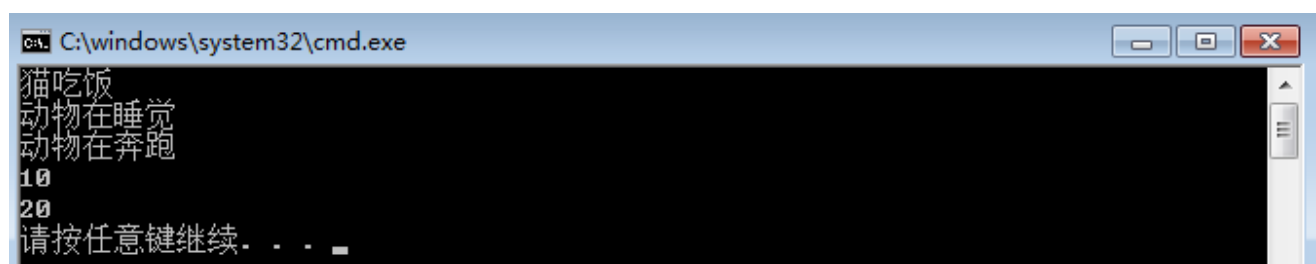
3、父类数据类型的引用指向子类对象。

测试类Demo_Test1中 **Animal am = new Cat();**语句在[堆内存](#)中开辟了子类(Cat)的对象，并把[栈内存](#)中的父类(Animal)的引用指向了这个Cat对象。

到此，满足了Java多态的必要三个前提。

-----华丽的分割线-----

如果再深究一点呢，我们可以看看上面测试类的输出结果，或许对多态会有更深层次的认识。猜一猜上面的结果是什么。



可以看出来

子类Cat重写了父类Animal的非静态成员方法am.eat();的输出结果为：猫吃饭。

子类重写了父类(Animal)的静态成员方法am.sleep();的输出结果为：动物在睡觉

未被子类（Cat）重写的父类（Animal）方法am.run()输出结果为：动物在奔跑

```
System.out.println(am.num); //输出结果为10  
System.out.println(am.age); //输出结果为20
```

那么我们可以根据以上情况总结出多态成员访问的特点：

成员变量

编译看左边(父类),运行看左边(父类)

成员方法

编译看左边(父类), 运行看右边(子类)。 [动态绑定](#)

静态方法

编译看左边(父类), 运行看左边(父类)。

(静态和类相关, 算不上重写, 所以, 访问还是左边的)

只有非静态的成员方法,编译看左边,运行看右边

-----华丽的分割线-----

那么多态有什么弊端呢? 有的, 即多态后不能使用子类特有的属性和方法。往上面的代码看, 子类Cat有一个特有的属性String name = "tomCat"; 并且还有一个特有的抓老鼠的方法catchMouse()。但是在测试类 (Demo_Test) 中, 我们尝试调用子类特有的方法catchMouse()和打印子类特有的成员属性String name = "tomCat"; 就会报错。

```
am.catchMouse();
System.out.println(am.name);
```

```
----- javac -----
Demo_Test.java:8: 错误: 找不到符号
    am.catchMouse();
      ^
  符号:   方法  catchMouse()
  位置:  类型为Animal的变量 am
Demo_Test.java:9: 错误: 找不到符号
    System.out.println(am.name);
                        ^
  符号:   变量 name
  位置:  类型为Animal的变量 am
2 个错误

输出完成 (耗时 0 秒) - 正常终止
```

原因就是多态的弊端, 就是: 不能使用子类特有的成员属性和子类特有的成员方法。

-----华丽的分割线-----

如果在代码执行过程中还想使用Cat类中特有的属性String name和它特有的成员方法catchMouse()了怎么办呢? 那我们就可以把这个父类引用指向了子类对象的家伙am再强制变回Cat类型。这样am就是Cat类型的引用了, 指向的也是Cat对象了, 自然也能使用Cat类的一切属性和一切的成员方法。

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

        Animal am = new Cat();
        am.eat();
        am.sleep();
        am.run();
        // am.catchMouse();
    }
}
```

```
// System.out.println(am.name);
System.out.println(am.num);
System.out.println(am.age);

System.out.println("-----");
Cat ct = (Cat)am;
ct.eat();
ct.sleep();
ct.run();
ct.catchMouse();
}
}
```

很明显，执行强转语句 `Cat ct = (Cat)am;` 之后，`ct` 就指向最开始在堆内存中创建的那个 `Cat` 类型的对象了。这就是多态的魅力吧，虽然它有缺点，但是它确实十分灵活，减少多余对象的创建，不用说为了使用子类的某个方法又去重新再堆内存中开辟一个新的子类对象。以上。。

-----分割线-----

啦啦啦,收到这么多赞.很惊讶,举个通俗点的例子吧.

花木兰替父从军

大家都知道花木兰替父从军的例子，花木兰替父亲花弧从军。那么这时候花木兰是子类，花弧是父类。花弧有自己的成员属性年龄，姓名，性别。花木兰也有这些属性，但是很明显二者的属性完全不一样。花弧有自己的非静态成员方法‘骑马杀敌’，同样花木兰也遗传了父亲一样的方法‘骑马杀敌’。花弧还有一个静态方法‘自我介绍’，每个人都可以问花弧姓甚名谁。同时花木兰还有一个自己特有的非静态成员方法‘涂脂抹粉’。但是，现在花木兰替父从军，女扮男装。这时候相当于父类的引用（花弧这个名字）指向了子类对象（花木兰这个人），那么在其他类（其他人）中访问子类对象（花木兰这个人）的成员属性（姓名，年龄，性别）时，其实看到的都是花木兰她父亲的名字（花弧）、年龄（60岁）、性别（男）。当访问子类对象（花木兰这个人）的非静态成员方法（骑马打仗）时，其实都是看到花木兰自己运用十八般武艺在骑马打仗。当访问花木兰的静态方法时（自我介绍），花木兰自己都是用她父亲的名字信息在向别人作自我介绍。并且这时候花木兰不能使用自己特有的成员方法‘涂脂抹粉’。-----多态中的向上转型

那么终于一将功成万骨枯，打仗旗开得胜了，花木兰告别了战争生活。有一天，遇到了自己心爱的男人，这时候爱情的力量将父类对象的引用（花弧这个名字）强制转换为子类对象本来的引用（花木兰这个名字），那么花木兰又从新成为了她自己，这时候她完全是她自己了。名字是花木兰，年龄是28，性别是女，打仗依然那样生猛女汉子，自我介绍则堂堂正正地告诉别人我叫花木兰。OMG！终于，终于可以使用自己特有的成员方法‘涂脂抹粉’了。从此，花木兰完全回到了替父从军前的那个花木兰了。并且和自己心爱的男人幸福的过完了一生。-----多态中的向下转型

-----华丽的分割线-----

大家记得哈，向上转型向下转型一定是在多态这个前提下哈，否则强制将女儿变成父亲，或者将父亲变成女人，就变成东方不败了，系统此时就会报错非法类型转换。哈哈哈哈哈。另外开发中一般是利用多态声明形式参数，并将创建子类的匿名对象作为实际参数。以上。