java_11

```
内容：
    多态
```

Java 多态

多态（Polymorphism）按字面的意思就是"多种形态,多种状态"。在面向对象语言中，**接口的多种不同的实现方式即为多态。**一句话：允许将**子类类型**的指针**赋值**给**父类类型**的指针。

多态指**同一个**方法调用，由于对象的不同而产生不同的**行为**方法。现实生活中，同一个方法，具体实现完全不同。（如前面的学过的吃方法eat，对应每个人所吃方法不一样）

多态的优点

- 消除类型之间的耦合关系(继承耦合过高)
- 可替换性
- 可扩充性
- 接口性[后面学]

多态要点

```
1).多态只是方法的多态，而不是属性的多态（多态因此与属性没有关系，只与方法有关）
2).多态存在必须达到三个条件： 继承，方法重写，父类引用指向子类对象
3).父类引用指向子类对象后，用该父类调用子类重写方法，此时则产生多态
```

如：

```
//前面学过的继承类
Son son = new Son("李世民", 20);
```

当使用多态方式调用方法时，首先检查父类中是否有该方法，如果没有，则编译错误；如果有，再去调用子类的同名方法。

多态的好处：可以使程序有良好的**扩展**，并可以对所有类的对象进行通用处理。

创建多态

1)多态（三个类）代码编写：

```
/**
 * 多态方式
 * @author Administrator
 * @date   2019年1月13日
 */
public class Father {
    //定义eat方法
    public void eat(){
        System.out.println("李渊正在吃饭");
    }
}
```

```
public class Son extends Father{

    //继承Father类有的方法，Son类去实现
    public void eat() {
        System.out.println("李世民正在吃饭");
    }

    //Son类自己编写的方法
    public void sleeping() {
        System.out.println("李世民准备睡觉了");
    }
}
```

```
public class Daughter extends Father{

    //继承Father有的方法，并实现方法
    public void eat() {
        System.out.println("李秀宁正在吃饭");
    }

    //Daughter类自有方法
    public void watchTV() {
        System.out.println("李秀宁在看电视");
    }

}
```

*：父类的方法，子类重写

2).代码测试

```
/**
 * 多态测试
 * @author Administrator
 * @version
 */
public class PolyTest {

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

```
        //不是多态情况下，调用则写很多的重载方法（如果子类几十个或是上百，多态的优势就显示出来）
        getChileObj(new Son());//只能传入Son类
        getChileObj(new Daughter());//只能传入Daughter类

        //多态则可以传入不同的对象（当传入什么对象，父类引用则指向子类对象，对应调用子类相应的子方法）
        getPolyObj(new Son());
        getPolyObj(new Father());
        getPolyObj(new Daughter());
    }

    //没有多态方式，假如Father下面还有更多子类，那么不断的写入很多的重载方法
    public static void getChileObj(Son s) {
        s.eat();
    }

    public static void getChileObj(Daughter d) {
        d.eat();
    }

    //多态方式，只需要传入类即可(当传入什么对象，父类引用则指向子类对象)
    public static void getPolyObj(Father f) {
        f.eat();
    }
}
```

多态转型：

```
/**
 * 多态测试
 * @author Administrator
 * @version
 */
/**
 * 多态测试
 * @author Administrator
 * @date   2019年7月7日
 * @version
 */
public class PolyTest2 {

    public static void main(String[] args) {
        // 多态接收三要素(继承，方法重写，父类引用指向子类对象)
        //自动向上转型，向下转型编译之后报异常[ClassCastException]错误提示

        Father fs = new Son();// 自动向上转型
        //向上转型接收可以
        fs.eat();//只能调用一个eat方法
        //fs.sleeping();//编译则出错，因为fs是Father类接收，认为只是Father类型

        Son son = (Son) fs;//强制向下转型，此时已经是Son子类
        //fs是父类接收，如果再转成子类接收，则需要强制()转型
        son.sleeping();//此处可以调用到子类的方法
```

```
        //可用Father类接收，也可用Son类接收

        // Father父类向下的子类转型不可以
        // Son son = new Father();//编译出错

        //如果fs强制转成Daughter
        //Daughter sd = (Daughter)fs;//编译不报错
        //运行则会报错:Son cannot be cast to Daughter

        System.out.println("====================");
        // 自身类接收
        Son son2 = new Son();// 自身类型接收
        son2.eat();
        son2.sleeping();// 子方法

        System.out.println("====================");
        // 自身类接收
        Daughter daughter = new Daughter();// 自身类型接收
        daughter.eat();
        daughter.watchTV();// 子方法
    }
}
```

Java 重写(Override)与重载(Overload)的不同：

```
a).重写：方法重写
b).重载:带不同参数构造方法
```

多态中使用最多的是接口：

```
a).生活中的接口(插座)。
b).java 中的接口类似于生活中的接口，就是一些方法特征的集合，但没有方法的实现体
```

final关键字作用

1).修饰变量：被修饰的变量不可变，只要赋初始值，就不能重新再被赋值（因此不可变）

```
public final USERNAME = "root";//数据库用户账号
public final PASSWORD = "root123";//数据库密码
```

2).修饰方法：一旦用final修饰过，此方法不可以被子类重写，但可以被重载

```
public final getConnet(){}
```

3).修饰类：使用final修饰过的类不能被继承，如String,System...

```
public final DabaBase(){}
```

如前面的Father类中的eat方法加上： public final void eat()，则下面的两个子类都报错，方法是final之后不可以被重写，或是Father类加上final，所有的子类都会报错，因此只是final的类都不能被继承，方法也不能重写。