Go 语言面向对象

面向对象

面向对象就是：把数据及对数据的操作方法放在一起，作为一个相互依存的整体——对象。对同类对象抽象出其共性，形成类。类中的大多数数据，只能用本类的方法进行处理。类通过一个简单的外部接口与外界发生关系，对象与对象之间通过消息进行通信。程序流程由用户在使用中决定。对象即为人对各种具体物体抽象后的一个概念，人们每天都要接触各种各样的对象，如手机就是一个对象。

封装、继承、多态是面向对象的3个基本特征。

封装

封装：将属性和方法封装到一起，隐藏内部实现细节，仅对外提供公共的访问方式

优势是便于使用，提高了安全性。提供了重复性

继承

继承：继承是类与类之间的关系，一个类A继承另一个类·B时，A会将B属性和方法全部继承过来，A叫子类也叫派生类，一个子类可以继承多个父类·

目的是提高了代码的重用性，减少代码冗余，规范子类，实现规一化

多态

多种状态，一个类可以创建多个对象每个对象可以有不同的属性和方法，称为多态

多态按字面的意思就是"多种状态"，用通俗一点的说法来说：多态就是指不同对象调用同一个方法功能的表现形式不一样，例如:不同的两个对象，列表的加法和整数的加法，同样是加法，实现的功能是不一样的。

Go语言中面向对象

Go语言中没有明确的OOP（Object Oriented Programming）面向对象的概念。

Go语言只提供了两个关键类型：struct，interface。

- Go支持面向对象(OOP)，并不是纯粹的面向对象语言；
- Go没有类的概念，结构体(struct)相当于其它编程语言的类(class)；
- Go面向对象编程非常简洁，通过接口(interface)关联，耦合性低，也非常灵活；

封装

封装就是把抽象出来的字段和操作方法封装在一起，数据被保护在内部，只有通过操作方法，才能对字段进行操作。

我们先定义一个手机结构体，在Go语言中视为类，类私有成员包括：品牌、型号、颜色、价格所组成。

```
// 定义一个手机结构体，在Go语言中视为类，类私有成员包括：品牌、型号、颜色、价格所组成
type Phone struct {
    brand string // 品牌
    model string // 型号
    color string // 颜色
    price int    // 价格
}
```

构造方法：

在Go语言中我们一般情况下使用New+结构体名称的规则来定义构造方法，构造函数中实例Phone类和初始化类成员。

```
// 构造方法
func NewPhone(brand, model, color string, price int) *Phone {
    return &Phone{
        brand: brand,
        model: model,
        color: color,
        price: price,
    }
}
```

封装方法：

接下来我们定义一个类成员方法，该方法用于打印类的所有成员变量。

```
// 封装方法获取phone的信息
func (p *Phone) GetPhoneInfo() {
    fmt.Printf("手机的品牌是：%s，型号是：%s，颜色是：%s，价格是：%d\n", p.brand,
p.model, p.color, p.price)
}
```

该方法用于获取Phone的信息，打印出所有类成员变量的值，func和方法名之间的“p *Phone”是指针类型（p为指针变量用于操作类成员，Phone为具体归属于哪个类），如果使用指针方式是为了需要修改类成员值，非指针时不能覆盖成员变量的值。

我们来测试一下上面的程序：

```
func main() {
    var phone = NewPhone("小米", "小米13 pro", "黑色", 1999)
    phone.GetPhoneInfo()
}
```

输出结果：

```
手机的品牌是：小米，型号是：小米13 pro，颜色是：黑色，价格是：1999
```

我们可以看到，我们私有成员变量的值在构造的时候被修改了，在调用打印的时候被输出出来了。

继承

Go语言中没有显示的类的继承，我们可以通过组合的形式来实现类的继承。

Go没有显式的继承，而是通过组合实现继承。

继承顾名思义，可以解决代码复用，通过**结构体嵌套**实现这个特性。

定义一个UseingPhone 正在用的手机类，通过组合的方式集成于Phone，并有自己的类成员为ChargeSpeed来表示充电速度。

```
// 定义一个UseingPhone 正在用的手机类，通过组合的方式集成于Phone，并有自己的类成员为ChargeSpeed来表示充电速度。
type UseingPhone struct {
    Phone
    ChargeSpeed string // 定义充电速度
}
```

封装方法 获取正在使用的手机信息

```
// 封装方法 获取正在使用的手机信息
func (up *UseingPhone) GetUsingPhoneInfo() {
    fmt.Printf("我是正在使用的手机，品牌是：%s，型号是：%s，颜色是：%s，价格是：%d，充电速度为：%s\n", up.brand, up.model, up.color, up.price, up.ChargeSpeed)
}
```

我们来运行一下程序：

```
func main() {
    // 继承
    var phone = NewPhone("小米", "小米10 pro", "黑色", 3999)
    var usingPhone = UseingPhone{
        Phone:       *phone,
        ChargeSpeed: "40min",
    }
    usingPhone.GetUsingPhoneInfo()
}
```

运行结果：

```
我是正在使用的手机，品牌是：小米，型号是：小米10 pro，颜色是：黑色，价格是：3999，充电速度为：40min
```

多态

把它们共同的方法提炼出来定义一个抽象的接口，就是多态。

多态是运行时的特性，而继承是编译时的特性。继承关系在编译的时候就就是已经确定的，而多态是运行的时候动态绑定的。

苹果手机和安卓手机都属于手机，具备打电话和上网的功能，我们把他们共同的属性抽象成接口表达：

```
// 定义手机功能
type Phone interface {
```

```
    Call(to string) // 打电话
    OnInternet()    // 上网
}

// 定义苹果手机结构体
type IPhone struct{}

// 苹果手机实现打电话功能
func (p *IPhone) Call(to string) {
    fmt.Printf("使用IPhone手机打电话给：%s\n", to)
}

// 苹果手机实现上网功能
func (p *IPhone) OnInternet() {
    fmt.Println("使用IPhone手机上网")
}

// 定义安卓手机结构体
type Android struct{}

// 安卓手机实现打电话功能
func (p *Android) Call(to string) {
    fmt.Printf("使用Android手机打电话给：%s\n", to)
}

// 安卓手机实现上网功能
func (p *Android) OnInternet() {
    fmt.Println("使用Android手机上网")
}
```

多态实现：

```
// 打电话
func PCall(p Phone, to string) {
    p.Call(to)
}

// 上网
func POnInternet(p Phone) {
    p.OnInternet()
}

func main() {
    // 多态
    var iphone = &IPhone{}
    PCall(iphone, "包子")
    POnInternet(iphone)

    fmt.Println()

    var android = &Android{}
    PCall(android, "肉包子")
    POnInternet(android)
}
```

运行结果

```
使用IPhone手机打电话给：包子
使用IPhone手机上网

使用Android手机打电话给：肉包子
使用Android手机上网
```

到这里面我们可以看到，通过抽象的接口和方法完成了Go语言运行时的动态绑定，这也就是Go语言中的抽象和多态了。