课程安排，请关注**微信公众平台**或者**官方微博**

编程语言： Golang 与 html5

编程工具： Goland 和 HBuilder

预计平均一周左右更新一或二节课程

授人以鱼，不如授人以渔业。

大家好，

欢迎来到 **字节教育** 课程的学习

字节教育官网：www.ByteEdu.Com

腾讯课堂地址：Gopher.ke.qq.Com

技术交流群 ： 221 273 219

微信公众号 ：Golang 语言社区

微信服务号 ：Golang 技术社区

目录：

第一季　Golang 语言社区-设计模式

第四节 观察者模式

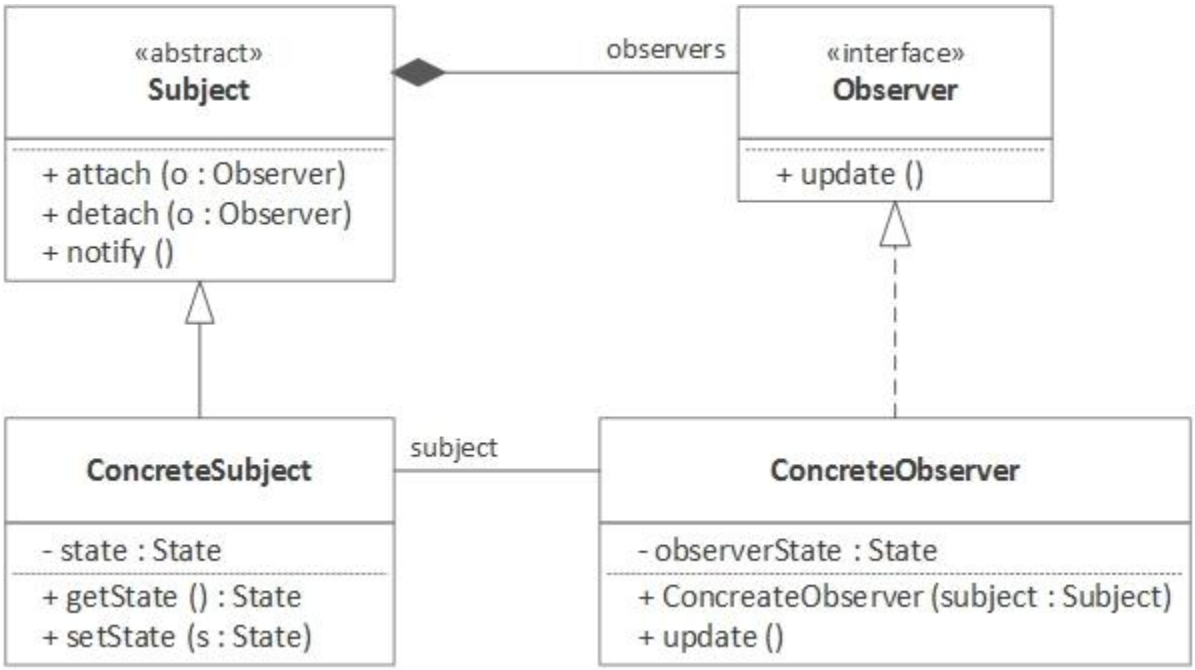
一、公众账号：

回复关键字：客服　　获取课程助教的微信，申请加入课程微信群

二、观察者模式讲解：

观察者模式简单一句话说就是当特定事件出现时，一个对象实例把事件发布到对应的观察者实例上执行相应的更新操作。一个观察目标可以对应多个观察者，而且这些观察者之间没有相互联系，可以根据需要增加和删除观察者，使得系统更易于扩展，这就是观察者模式的模式动机。

其类图如下：



1. 首先，定一个事件类型，发生事件驱动时将事件传递给观察者们。这里数据是一个 string 类型的 data， 实际情况可更具需要而定。

```
type Event struct {
    Data string
}
```

1. 定义观察者和观察对象的接口。Observer 定义了一个更新发生事件的标准接口，Subject 是具体被观察的接口，他有注册观察者、注销观察者和发布通知的三个主要函数接口。

```
type Observer interface {
    //更新事件
    Update(*Event)
}

// 被观察的对象接口
type Subject interface {
    //注册观察者
    Regist(Observer)
    //注销观察者
    Deregist(Observer)

    //通知观察者事件
    Notify(*Event)
}
```

1. 实现观察者和对象的接口。

```
type ConcreteObserver struct {
    Id int
}

func (co *ConcreteObserver) Update(e *Event) {
    fmt.Printf("observer [%d] recieved msg: %s.\n", co.Id, e.Data)
```

```
}

type ConcreteSubject struct {
    Observers map[Observer]struct{}
}

func (cs *ConcreteSubject) Regist(ob Observer) {
    cs.Observers[ob] = struct{}{}
}

func (cs *ConcreteSubject) Deregist(ob Observer) {
    delete(cs.Observers, ob)
}

// 通知每个观察者事件
func (cs *ConcreteSubject) Notify(e *Event) {
    for ob, _ := range cs.Observers {
        ob.Update(e)
    }
}
```

使用：

```
func main() {
    cs := &ConcreteSubject{
        Observers: make(map[Observer]struct{}),
    }

    //实例化两个观察者
    cobserver1 := &ConcreteObserver{1}
    cobserver2 := &ConcreteObserver{2}

    //注册观察者
    cs.Regist(cobserver1)
    cs.Regist(cobserver2)

    for i := 0; i < 5; i++ {
        e := &Event{fmt.Sprintf("msg [%d]", i)}
        cs.Notify(e)

        time.Sleep(time.Duration(1) * time.Second)
    }
}
```