refactoring.guru

Observer em Go / Padrões de Projeto

4–5 minutes

O **Observer** é um padrão de projeto comportamental que permite que um objeto notifique outros objetos sobre alterações em seu estado.

O padrão Observer fornece uma maneira de assinar e cancelar a assinatura desses eventos para qualquer objeto que implemente uma interface de assinante.

Exemplo conceitual

No site de comércio eletrônico, os itens ficam sem estoque de vez em quando. Pode haver clientes que estejam interessados em um item específico que ficou fora de estoque. Existem três soluções para este problema:

1. O cliente continua verificando a disponibilidade do item com alguma frequência.
2. O e-commerce bombardeia os clientes com todos os novos itens disponíveis, que estão em estoque.

3. O cliente assina apenas o item específico no qual está interessado e é notificado se o item estiver disponível. Além disso, vários clientes podem assinar o mesmo produto.

   A opção 3 é a mais viável e é disso que trata o padrão Observer. Os principais componentes do padrão Observer são:

- Alvo, a instância que publica um evento quando algo acontece.
- Observer, que se inscreve nos eventos do alvo e é notificado quando eles acontecem.

subject.go: Alvo

```
package main

type Subject interface {
    register(observer Observer)
    deregister(observer Observer)
    notifyAll()
}
```

item.go: Alvo concreto

```
package main

import "fmt"

type Item struct {
    observerList []Observer
    name         string
    inStock      bool
```

```
}

func newItem(name string) *Item {
    return &Item{
        name: name,
    }
}
func (i *Item) updateAvailability() {
    fmt.Printf("Item %s is now in stock\n", i.name)
    i.inStock = true
    i.notifyAll()
}
func (i *Item) register(o Observer) {
    i.observerList = append(i.observerList, o)
}

func (i *Item) deregister(o Observer) {
    i.observerList = removeFromslice(i.observerList, o)
}

func (i *Item) notifyAll() {
    for _, observer := range i.observerList {
        observer.update(i.name)
    }
}

func removeFromslice(observerList []Observer,
observerToRemove Observer) []Observer {
    observerListLength := len(observerList)
    for i, observer := range observerList {
```

```
        if observerToRemove.getID() == observer.getID() {
            observerList[observerListLength-1], observerList[i] =
observerList[i], observerList[observerListLength-1]
            return observerList[:observerListLength-1]
        }
    }
    return observerList
}
```

observer.go: Observer

```
package main

type Observer interface {
    update(string)
    getID() string
}
```

customer.go: Observer concreto

```
package main

import "fmt"

type Customer struct {
    id string
}

func (c *Customer) update(itemName string) {
    fmt.Printf("Sending email to customer %s for item %s\n", c.id,
```

```
itemName)
}

func (c *Customer) getID() string {
    return c.id
}
```

main.go: Código cliente

```
package main

func main() {

    shirtItem := newItem("Nike Shirt")

    observerFirst := &Customer{id: "abc@gmail.com"}
    observerSecond := &Customer{id: "xyz@gmail.com"}

    shirtItem.register(observerFirst)
    shirtItem.register(observerSecond)

    shirtItem.updateAvailability()
}
```

output.txt: Resultados da execução

```
Item Nike Shirt is now in stock
Sending email to customer abc@gmail.com for item Nike Shirt
Sending email to customer xyz@gmail.com for item Nike Shirt
```

Observer em outras linguagens