Если вы видите что-то необычное, просто сообщите мне.

Заместитель (Proxy)

Паттерн Proxy относится к структурным паттернам уровня объекта.

Паттерн Proxy предоставляет объект для контроля доступа к другому объекту.

Другое название паттерна - "Суррогат". В этом понимании, это предмет или продукт, заменяющий собой какой-либо другой предмет или продукт, с которым суррогат имеет лишь некоторые общие свойства, но он не обладает всеми качествами оригинального предмета или продукта.

Паттерна Proxy выдвигается ряд важных требований, а именно то, что оригинальный объект и его суррогат должны взаимодействовать друг с другом, а также должна быть возможность, замещения оригинальным объектом, суррогата в месте его использования, соответственно интерфейсы взаимодействия оригинального объекта и его суррогата должны совпадать.

Вам будет легче понять паттерн, если вы смотрели фильм "Суррогаты".

Требуется для реализации:

1. Интерфейс Subject, являющейся общим интерфейсом для реального объекта и его суррогата;
2. Класс RealSubject, реализующий реальный объект;
3. Класс Proxy, реализующий объект суррогата. Хранит в себе ссылку на реальный объект, что позволяет заместителю обращаться к реальному объект напрямую;

Например, паттерн Proxy можно использовать, если нам нужно управлять ресурсоемкими объектами, но мы не хотим создавать экземпляры таких объектов до момента их реального использования.

Вы можете подумать, что это тоже самое, что и Adapter или Decorator. Но...

Proxy предоставляет своему объекту тот же интерфейс. Adapter предоставляет другой интерфейс. Decorator предоставляет расширенный интерфейс.

[!] В описании паттерна применяются общие понятия, такие как Класс, Объект, Абстрактный класс. Применимо к языку Go, это Пользовательский Тип, Значение этого Типа и Интерфейс. Также в языке Go за место общепринятого наследования используется агрегирование и встраивание.

```
//proxy.go
// Package proxy is an example of the Adapter Pattern.
package proxy

// Subject provides an interface for a real subject and its surrogate.
type Subject interface {
	Send() string
}

// Proxy implements a surrogate.
type Proxy struct {
	realSubject Subject
}

// Send sends a message
func (p *Proxy) Send() string {
	if p.realSubject == nil {
		p.realSubject = &RealSubject{}
	}
	return "<strong>" + p.realSubject.Send() + "</strong>"
}

// RealSubject implements a real subject
type RealSubject struct {
}

// Send sends a message
func (s *RealSubject) Send() string {
	return "I’ll be back!"
}
```

```
//proxy_test.go
package proxy

import (
	"testing"
)

func TestProxy(t *testing.T) {

	expect := "<strong>I’ll be back!</strong>"

	proxy := new(Proxy)

	result := proxy.Send()

	if result != expect {
		t.Errorf("Expect result to equal %s, but %s.\n", expect, result)
	}
}
```

Revision #1
Created 2022-07-03 10:33:19 UTC by gasick
Updated 2022-07-03 10:34:01 UTC by gasick