refactoring.guru

Template Method em Go / Padrões de Projeto

4–5 minutes

O **Template Method** é um padrão de projeto comportamental que permite definir o esqueleto de um algoritmo em uma classe base e permitir que as subclasses substituam as etapas sem alterar a estrutura geral do algoritmo.

Exemplo conceitual

Vamos considerar o exemplo da funcionalidade One Time Password (OTP). Existem diferentes maneiras de entregar o OTP a um usuário (SMS, email, etc.). Mas, independentemente de ser um OTP SMS ou email, todo o processo OTP é o mesmo:

1. Gere um número aleatório de n dígitos.
2. Salve este número no cache para verificação posterior.
3. Prepare o conteúdo.
4. Envie a notificação.

Quaisquer novos tipos de OTP que serão introduzidos no futuro

provavelmente ainda passarão pelas etapas acima.

Portanto, temos um cenário em que as etapas de uma operação específica são as mesmas, mas a implementação dessas etapas pode ser diferente. Esta é uma situação apropriada para considerar o uso do padrão Template Method.

Primeiro, definimos um algoritmo template base que consiste em um número fixo de métodos. Esse será o nosso método modelo. Em seguida, implementaremos cada um dos métodos da etapa, mas deixaremos o método modelo inalterado.

otp.go: Template method

```
package main

type IOtp interface {
    genRandomOTP(int) string
    saveOTPCache(string)
    getMessage(string) string
    sendNotification(string) error
}
```

```
type Otp struct {
    iOtp IOtp
}

func (o *Otp) genAndSendOTP(otpLength int) error {
    otp := o.iOtp.genRandomOTP(otpLength)
    o.iOtp.saveOTPCache(otp)
    message := o.iOtp.getMessage(otp)
    err := o.iOtp.sendNotification(message)
    if err != nil {
        return err
    }
    return nil
}
```

sms.go: Implementação concreta

```
package main

import "fmt"

type Sms struct {
    Otp
}
```

```
func (s *Sms) genRandomOTP(len int) string {
    randomOTP := "1234"
    fmt.Printf("SMS: generating random otp %s\n", randomOTP)
    return randomOTP
}

func (s *Sms) saveOTPCache(otp string) {
    fmt.Printf("SMS: saving otp: %s to cache\n", otp)
}

func (s *Sms) getMessage(otp string) string {
    return "SMS OTP for login is " + otp
}

func (s *Sms) sendNotification(message string) error {
    fmt.Printf("SMS: sending sms: %s\n", message)
    return nil
}
```

email.go: Implementação concreta

```
package main

import "fmt"

type Email struct {
    Otp
}
```

```
func (s *Email) genRandomOTP(len int) string {
    randomOTP := "1234"
    fmt.Printf("EMAIL: generating random otp %s\n", randomOTP)
    return randomOTP
}

func (s *Email) saveOTPCache(otp string) {
    fmt.Printf("EMAIL: saving otp: %s to cache\n", otp)
}

func (s *Email) getMessage(otp string) string {
    return "EMAIL OTP for login is " + otp
}

func (s *Email) sendNotification(message string) error {
    fmt.Printf("EMAIL: sending email: %s\n", message)
    return nil
}
```

main.go: Código cliente

```
package main

import "fmt"

func main() {
```

```
    smsOTP := &Sms{}
    o := Otp{
        iOtp: smsOTP,
    }
    o.genAndSendOTP(4)

    fmt.Println("")
    emailOTP := &Email{}
    o = Otp{
        iOtp: emailOTP,
    }
    o.genAndSendOTP(4)

}
```

output.txt: Resultados da execução

```
SMS: generating random otp 1234
SMS: saving otp: 1234 to cache
SMS: sending sms: SMS OTP for login is 1234
```

```
EMAIL: generating random otp 1234
EMAIL: saving otp: 1234 to cache
EMAIL: sending email: EMAIL OTP for login is 1234
```

Template Method em outras linguagens