GO-07 03: Кодогенерация. Генерация кода из шаблонов в go

Описание:

Вторым крутым механизмом, уже более преближенным к понятию кодогенерации, является шаблонизация. Шаблонизация представляет из себя подмешивание каких-либо программных значений в любой текст html/yaml/json/txt. Например, в мире фронтенда есть такой html шаблонизатор, как pug, выглядит он следующим образом:

```
- var title = 'My Site'
- var message = 'Hello, World!'
html
  head
    title #{title}
  body
    h1 #{message}
```

Здесь при компиляции этого исходного файла непосредственно в html значения из переменных title и message подставляются в соответствующие места. HTML код после генерации будет выглядеть вот так:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Site</title>
</head>
<body>
  <h1>Hello, World!</h1>
</body>
</html>
```

Такой механизм дает нам большие возможностей при работе со статическими текстовыми файлами. Еще один яркий пример, но уже более приближенный к go, - это Helm чарты. Helm - это набор шаблонизированных конфигурационных файлов в yml формате для описания инфраструктуры на основе kubernetes, и как раз этот проект полностью написан на go и использует механизм шаблонизации языка на полную. Вот небольшой отрывок из чарта, показывающий наличие шаблонизации:

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: "{{  .Chart.Name }}-deployment"
  labels:
    chart: '{{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}'
```

Пакет text/template
Первый пакет, на который мы посмотрим, - text/template. Он нам позволяет шаблонизировать текстовые файлы любого формата. Давайте разберем на примере. Допустим, мы пишем письмо:

```
Dear Aunt Mildred,
It was a pleasure to see you at the wedding.
Thank you for the lovely bone china tea set.
Best wishes,
Josie
```

В этом письме мы благодарим человека за то, что он приехал к нам на свадьбу и подарил замечательный подарок. Но письмо мы хотим отправить не одному человеку, а нескольким, а еще, в зависимости от адресата, мы можем захотеть чутка изменить наше сообщение (например, кто-то не пришел на свадьбу или подарил другой подарок). Так вот здесь нам и поможет шаблонизация!

```
Dear {{.Name}},
{{if .Attended}}
It was a pleasure to see you at the wedding.
{{- else}}
It is a shame you couldn't make it to the wedding.
{{- end}}
{{with .Gift -}}
Thank you for the lovely {{.}}.
{{end}}
Best wishes,
Josie
```

Go позволяет достаточно мощно шаблонизировать наш текст и не только подставлять значения из переменных или структур. Он также предоставляет механизмы управления шаблонами, такими как конструкция {{if .Attended}}, в которой, как и в настоящем языке программирования, будет проверка «Если Attended == true, значит, мы выведем один текст, иначе выведем другой». Более полный список возможностей можно посмотреть в документации.
Теперь нам нужно сопоставить наши шаблоны с реальными данными, для этого нам и нужен пакет text/template.

```
package main
import (
```

```
  "html/template"
  "log"
  "os"
)
var letter = `
Dear {{.Name}},
{{if .Attended}}
It was a pleasure to see you at the wedding.
{{- else}}
It is a shame you couldn't make it to the wedding.
{{- end}}
{{with .Gift -}}
Thank you for the lovely {{.}}.
{{end}}
Best wishes,
Josie
`
type Recipient struct {
  Name     string
  Gift     string
  Attended bool
}
func main() {
  var recipients = []Recipient{
    {"Aunt Mildred", "bone china tea set", true},
    {"Uncle John", "moleskin pants", false},
    {"Cousin Rodney", "", false},
  }

  // Create a new template and parse the letter into it.
  t, err := template.New("letter").Parse(letter)
  if err != nil {
    panic(err)
  }

  // Execute the template for each recipient.
  for _, r := range recipients {
    err := t.Execute(os.Stdout, r)
    if err != nil {
      log.Println("executing template:", err)
```

```
    }
  }
}
```

Пользоваться пакетом достаточно просто. Нам нужно создать структуру, которая будет соответствовать шаблону, затем распарсить наш шаблон при помощи template.New("letter").Parse(letter), а дальше при помощи t.Execute подставить данные в шаблон.
Пакет html/template
Этот пакет имеет схожий интерфейс с text/template и выполняет, по сути, ту же работу, но этот пакет заточен больше на работу с html и предполагает, что генерируемый таким образом html будет защищен от внедрения вредоносного кода. Вот и все отличия.

Полезные ссылки:

- Go templates made easy
- Go text/template docs
- Yaml validator

Задание:

В этом задании вам нужно написать генератор YAML конфигов. На вход ему передается шаблонизированный YAML файл, который будет находиться в папке module07/assets/template/config_template.yml. Вам будет нужно дополнить шаблон информацией с различными вставками и сгенерировать по этому шаблону валидный YAML конфиг. Запуск осуществляется при помощи пакета module07/cmd/app и команды make run. Перед запуском нужно раскомментировать вызов функции Task03() в функции main.
Функция генератора имеет интерфейс:

```
func generate(tmpl string, outfilePath string, fields interface{})
error
```

Где:

- tmpl - это шаблон в виде строки;
- outFilePath - это путь до файла, в который будет сгенерирован конфиг;
- fields - значения, которые нужны шаблону для подстановки.

Это основная функция генератора, через которую дальше мы будем генерировать различный код.
Также в пакете есть функция с интерфейсом вида.

```
func ConfigGenerate(tmpl string, outFilePath string) error
```

Эта функция должна подготавливать данные для шаблона и вызывать функцию generate.
Порядок действий:

1. В вашем проекте module07 сделайте новую ветку module07_03.
2. В пакете module07/internal/generator заполните логикой функцию generate и ConfigGenerate.
3. Проверьте работоспособность генератора.
4. Проверьте через yaml validator сгенерированный вами конфиг.
5. В качестве ответа пришлите ссылку на merge request в ветку master вашего проекта ветки module07_03.