GO-09 01: Поднимаем сервер, изучаем роутинг, пишем первый handler

Описание:

В этом модуле мы будем рассматривать написание http сервера и все, что с ним связано. Например:

- Роутинг, в котором мы посмотрим, как работать со стандартным net/http роутером, а также глянем на различные роутер библиотеки, такие как Gorilla и Chi.
- Научимся работать с параметрами http. Как в плане получения запроса и обработки того, что нам пришло, так и отправки ответа с изменениями тела ответа, его заголовков и т.д.
- Научимся работать с контекстами, помещать и получать данные из них, выставлять тайм-ауты и т.д.
- Узнаем, что такое middleware и для чего их можно использовать.

И первым этапом мы посмотрим, как поднять наш первый http сервер.
В go поднять простой http сервер можно всего несколькими строчками.

```
package main

import "net/http"

func main() {
    http.ListenAndServe(":8080", nil)
}
```

Первым делом нам нужно импортировать стандартный пакет net/http, а затем в функции main вызвать метод пакета ListenAndServe, в котором:

- Первым аргументом передается строка вида host:port, которая отвечает за то, по какому адресу будет доступен наш сервер (часто можно опустить указание хоста и писать эту строку в виде :port, тогда хост будет равен IP-адресу машины).
- Вторым аргументом выступает ServerMux или в простонародье роутер, который отвечает за то, какие пути будет обрабатывать наш сервер /handle, /order/create и т.д. Если передавать туда nil, тогда будет использоваться роутер по умолчанию из пакета net/http.

Теперь, если запустить наш код, то у нас откроется TCP-соединение на порту 8080, но наш сервер сейчас ничего не делает, кроме того, что слушает, что происходит на порту 8080. Давайте научим его обрабатывать запросы.

Для этого нам нужно зарегистрировать http-обработчик на какой-нибудь запрос. В go со стандартным роутером из net/http это делается достаточно просто.

```
package main

import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request)
{
        fmt.Fprint(w, "hello")
    })

    http.ListenAndServe(":8080", nil)
}
```

В метод HandleFunc первым параметром мы передаем путь, при запросе на который будет работать наш обработчик. Вторым параметром является сама функция-обработчик с аргументами ResponseWrite, который служит для создания и отправки ответа на запрос, а также Request, в котором хранится информация о запросе.
Внутри функции обработчика при помощи функции fmt.Fprintf(w, "hello") мы отвечаем на запрос по пути / ответом hello, первым параметром передав наш ResponseWriter.
Еще один вариант регистрации обработчика - это создание структуры, которая будет реализовывать интерфейс http.Handler, который выглядит следующим образом:

```
type Handler interface {
    ServeHTTP(w http.ResponseWriter, r *http.Request)
}
```

Пример такого обработчика:

```
package main

import (
    "fmt"
    "net/http"
)

type ResponseHandler struct {
     message string
}
```

```
func (rh ResponseHandler) ServeHTTP(w http.ResponseWriter, r
*http.Request) {
      fmt.Fprint(w, rh.message)
}

func main() {
    resp := ResponseHandler{message: "hello"}
    http.Handle("/hello", resp)
    http.ListenAndServe(":8080", nil)
}
```

Также у пакета net/http есть некоторые стандартные реализации таких структур, например:

- http.FileServer возвращает обработчик, который обслуживает HTTP-запросы с содержимым файловой системы.

```
func main() {
      http.Handle("/", http.FileServer(http.Dir("./")))
      http.ListenAndServe(":8080", nil)
}
```

- http.NotFoundHandler возвращает простой обработчик 404 ошибки.

```
func main() {
      http.Handle("/", http.NotFoundHandler())
      http.ListenAndServe(":8080", nil)
}
```

- http.RedirectHandler возвращает обработчик, который перенаправляет пользователя по указанному URL.

```
func main() {
      http.Handle("/", http.RedirectHandler("https://google.com", 301))
      http.ListenAndServe(":8080", nil)
}
```

- http.StripPrefix возвращает обработчик, который будет обрезать префикс пути и перенаправлять клиента на обработчик, который мы указали в качестве параметра. Вот небольшой пример:

У нас есть обработчик вида:

```
http.Handle("/tmpfiles/", http.StripPrefix("/tmpfiles/",
http.FileServer(http.Dir("/tmp"))))
```

Теперь, когда клиент захочет получить файл /tmpfiles/example.txt, то обработчик удалит из адреса путь tmpfiles, дальше обработка пойдет по пути /tmp/example.txt и выдаст желаемый файл, если он существует, иначе - 404.

- http.TimeoutHandler обертка над другими обработчиками, которая возвращает новый обработчик с таймером на выполнение, когда время таймера истечет, реквест будет завершен как 503 Service unavailable.

```
func main() {
	http.Handle("/",  http.TimeoutHandler(myHandler, time.Second*10,
"Request Timeout"))
	http.ListenAndServe(":8080", nil)
}
```

Полезные ссылки:

- Go by examples HTTP Server
- Маршрутизация в GO
- Routing in net/http

Задание:

В этом задании вам надо будет поднять http-сервер и зарегистрировать несколько обработчиков:

- /hello обработчик должен просто поздороваться с нами, отправив сообщение hello human;
- /source обработчик должен отдавать в качестве ответа файлы из текущей директории;
- /longPing обработчик должен имитировать долгую работу:
    - Обработчик должен быть зарегистрирован через http.Handle.
    - Обработчик должен иметь тайм-аут на свое выполнение в 1 секунду, то есть через секунду ожидания в ответ должно прийти Request timeout.

Порядок действий:

1. Форкните репозиторий module09 с кодом данного задания - в группу с вашими репозиториями - golang_users_repos/<your_gitlab_id>.
2. В вашем проекте module09 сделайте новую ветку 01_task.
3. В пакете module09/cmd/app зарегистрируйте все нужные обработчики и создайте сервер.
4. Проверьте работоспособность.
5. В ответе пришлите ссылку на merge request в ветку master своего проекта ветки 01_task.